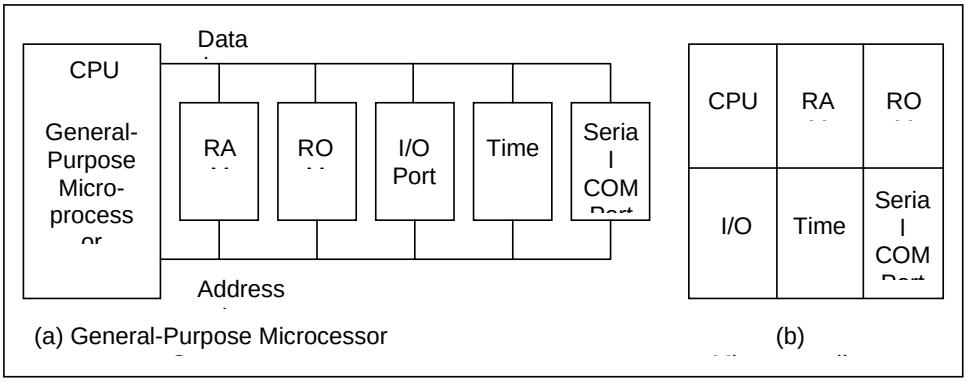
CHƯƠNG I
Các bộ vi điều khiển 8051

1.1 các bộ vi điều khiển và các bộ xử lý nhúng.

Trong mục này chúng ta bàn về nhu cầu đối với các bộ vi điều khiển (VĐK) và so sánh chúng với các bộ vi xử lý cùng dạng chung như Pentium và các bộ vi xử lý × 86 khác. Chúng ta cùng xem xét vai trò của các bộ vi điều khiển trong thị trường các sản phẩm nhúng. Ngoài ra, chúng ta cung cấp một số tiêu chuẩn về cách lựa chọn một bộ vi điều khiển như thế nào.

1.1.1 Bộ vi điều khiển so với bộ vi xử lý cùng dùng chung

Sự khác nhau giữa một bộ vi điều khiển và một bộ vi xử lý là gì? Bộ vi xử lý ở đây là các bộ vi xử lý công dung chung như họ Intell × 86 (8086, 80286, 80386, 80486 và Pentium) hoặc họ Motorola 680 × 0(68000, 68010, 68020, 68030, 68040 v.v...). Những bộ VXL này không có RAM, ROM và không có các cổng vào ra trên chíp. Với lý do đó mà chúng được gọi chung là các bộ vi xử lý công dụng chung.



Hình 1.1: Hệ thống vi xử lý được so sánh với hệ thống vi điều khiển.

a) Hệ thống vi xử lý công dụng chung

b) Hệ thống vi điều khiển

Một nhà thiết kế hệ thống sử dụng một bộ vi xử lý công dụng chung chẳng hạn như Pentium hay 68040 phải bổ xung thêm RAM , ROM, các cổng vào ra và các bộ định thời ngoài để làm cho chúng

hoạt động được. Mặc dù việc bổ xung RAM, ROM và các cổng vào ra bên ngoài làm cho hệ thống cồng cềnh và đắt hơn, nhưng chúng có ưu điểm là linh hoạt chẳng hạn như người thiết kế có thể quyết định về số lượng RAM, ROM và các cổng vào ra cần thiết phù hợp với bài toán trong tầm tay của mình.

Điều này không thể có được đối với các bộ vi điều khiển. Một bộ vi điều khiển có một CPU (một bộ vi xử lý) cùng với một lượng cố định RAM, ROM, các cổng vào ra và một bộ định thời tất cả trên cùng một chíp. Hay nói cách khác là bộ xử lý, RAM, ROM các cổng vào ra và bộ định thời đều được nhúng với nhau trên một chíp; do vậy người thiết kế không thể bổ xung thêm bộ nhớ ngoài, cổng vào ra hoặc bộ định thời cho nó. Số lượng cố định của RAM, ROM trên chíp và số các cổng vào - ra trong các bộ vi điều khiển làm cho chúng trở nên lý tưởng đối với nhiều ứng dụng mà trong đó giá thành và không gian lại hạn chế. Trong nhiều ứng dụng, ví dụ một điều khiển TV từ xa thì không cần công suất tính toán của bộ vi sử lý 486 hoặc thậm chí như 8086. Trong rất nhiều ứng dụng thì không gian nó chiếm, công suất nó tiêu tốn và giá thành trên một đơn vị là những cân nhắc nghiêm ngặt hơn nhiều so với công suất tính toán. Những ứng dụng thường yêu cầu một số thao tác vào - ra để đọc các tín hiệu và tắt - mở những bit nhất định. Vì lý do này mà một số người gọi các bộ xử lý này là IBP ("Itty-Bitty-Processor"), (tham khảo cuốn "Good things in small packages are Generating Big product opportunities" do Rick Grehan viết trên tạp BYTE tháng 9.1994; WWW. Byte. Com để biết về những trao đổi tuyệt vời về các bộ vi điều khiển).

Điều thú vị là một số nhà sản xuất các bộ vi điều khiển đã đi xa hơn là tích hợp cả một bộ chuyển đổi ADC và các ngoại vi khác vào trong bộ vi điều khiển.

Bảng 1.1: Một số sản phẩm được nhúng sử dụng các bộ vi điều khiển

Thiết bị nội thất gia đình	**Văn phòng**	**ô tô**
Đồ điện trong nhà	Điện thoại	Máy tính hành trình
Máy đàm thoại	Máy tính	Điều khiển động cơ
Máy điện thoại	Các hệ thống an toàn	Túi đệm khí
Các hệ thống an toàn	Máy Fax	Thiết bị ABS
Các bộ mở cửa ga-ra xe	Lò vi sóng	Đo lường
		Hệ thống bảo mật

Máy trả lời Máy Fax Máy tính gia đình Tivi Truyền hình cáp VCR Máy quy camera Điều khiển từ xa Trò chơi điện tử Điện thoại tổ ong Các nhạc cụ điện tử Máy khâu Điều khiển ánh sáng Máy nhắn tin Máy chơi Pootball Đồ chơi Các dụng cụ tập thể hình	Máy sao chụp Máy in lazer Máy in màu Máy nhắn tin	Đíũu khiển truyền tin Giải trí Điều hoà nhiệt độ Điện thoại tổ ong Mở cửa không cần chìa khoá

1.1.2 Các bộ VĐK cho các hệ thống nhúng.

Trong tài liệu về các bộ vi xử lý ta thường thấy khái niệm hệ thống nhúng (Embeded system). Các bộ vi xử lý và các bộ vi điều khiển được sử dụng rộng rãi trong các sản phẩm hệ thống nhúng. Một sản phẩm nhúng sử dụng một bộ vi xử lý (hoặc một bộ vi điều khiển để thực hiện một nhiệm vụ và chỉ một mà thôi. Một máy in là một ví dụ về một việc nhúng vì bộ xử lý bên trong nó chỉ làm một việc đó là nhận dữ liệu và in nó ra. Điều này khác với một máy tình PC dựa trên bộ xử lý Pentium (hoặc một PC tương thích với IBM × 86 bất kỳ). Một PC có thể được sử dụng cho một số bất kỳ các trạm dịch vụ in, bộ đầu cuối kiểm kê nhà băng, máy chơi trò chơi điện tử, trạm dịch vụ mạng hoặc trạm đầu cuối mạng Internet. Phần mềm cho các ứng dụng khác nhau có thể được nạp và chạy. Tất nhiên là lý do hiển nhiên để một PC thực hiện hàng loạt các công việc là nó có bộ nhớ RAM và một hệ điều hành nạp phần mềm ứng dụng thường được đốt vào trong ROM. Một máy tính PC × 86 chứa hoặc được nối tới các sản phẩm nhúng khác nhau chẳng hạn như bàn phím, máyin, Modem, bộ điều khiển đĩa, Card âm thanh, bộ điều khiển CD = ROM. Chuột v.v... Một nội ngoại vi này có một bộ vi điều khiển bên trong nó để thực hiện chỉ một công việc, ví dụ bên trong mỗi con chuột có một bộ vi

điều khiển để thực thi công việc tìm vị trí chuột và gửi nó đến PC Bảng 1.1 liệt kê một số sản phẩm nhúng.

4.1.3 Các ứng dụng nhúng của PC × 86.

Mặc dù các bộ vi điều khiển là sự lựa chọn ưa chuộng đối với nhiều hệ thống nhúng nhưng có nhiều khi một bộ vi điều khiển không đủ cho công việc. Vì lý do đó mà những năm gần đây nhiều nhà sản xuất các bộ vi sử lý công dụng chung chẳng hạn như Intel, Motorla, AMD (Advanced Micro Devices, Inc...). Và Cyric (mà bây giờ là một bộ phận của National Senicon ductir, Inc) đã hướng tới bộ vi xử lý cho hiệu suất cao của thị trường nhúng. Trong khi Intel, AMD và Cyrix đẩy các bộ xử lý × 86 của họ vào cho cả thị trường nhúng và thị trường máy tính PC để bán thì Motorola vẫn kiên định giữ họ vi xử lý 68000 lại chủ yếu hướng nó cho các hệ thống nhúng hiệu suất cao và bây giờ Apple không còn dùng 680 × trong các máy tính Macintosh nữa. Trong những năm đầu thập kỷ 90 của thế kỷ 20 máy tính Apple bắt đầu sử dụng các bộ vi xử lý Power PC (như 603, 604, 620 v.v...) thay cho 680 ×0 đối với Macintosh. Bộvi xử lý Power PC là kết quả liên doanh đầu tư của IBM và Motorola và nó được hướng cho thị trướng nhúng hiệu suất cao cũng như cho cả thị trường máy tính PC. Cần phải lưu ý rằng khi một công ty hướng một bộ vi xử lý công dụng chung cho thị trường nhúng nó tối ưu hoá bộ xử lý được sử dụng cho các hệ thống nhúng. Vì lý do đó mà các bộ vi xử lý này thường được gọi là các bộ xử lý nhúng hiệu suất cao. Do vậy các khái niệm các bộ vi điều khiển và bộ xử lý nhúng thường được sử dụng thay đổi nhau.

Một trong những nhu cầu khắt khe nhất của hệ thống nhúng là giảm công suất tiêu thụ và không gian.

Điều này có thể đạt được bằng cách tích hợp nhiều chức năng vào trong chíp CPU. Tất cả mọi bộ xử lý nhúng dựa trên × 86 và 680 × 0 đều có công suất tiêu thu thấp ngoài ra được bổ xung một số dạng cổng vào - ra, cổng COM và bộ nhớ ROM trên một chíp.

Comment [URG1]:

Trong các bộ xử lý nhúng hiệu suất cao có xu hướng tích hợp nhiều và nhiều chức năng hơn nữa trên chíp CPU và cho phép người thiết kế quyết định những đặc tính nào họ muốn sử dụng. Xu hướng này cũng đang chiếm lĩnh thiết kế hệ thống PC. Bình thường khi thiết kế bo mạch chủ của PC (Motherboard) ta cần một CPU cộng một chíp - set có chứa các cổng vào - ra, một bộ điều khiển cache, một bộ nhớ Flash ROM có chứa BIOS và cuối cùng là bộ nhớ cache thứ cấp. Những thiết kế mới đang khẩn trương đi vào công nghiệp sản xuất hàng loạt. Ví dụ Cyrix đã tuyên bố rằng họ đang làm việc trên một

chíp có chứa toàn bộ một máy tính PC ngoại trừ DRAM. Hay nói cách khác là chúng ta xắp nhìn thấy một máy tính PC trên một chíp.

Hiện nay do chuẩn hoá MS - DOS và Windows nên các hệ thống nhúng đang sử dụng các máy tình PC × 86 . Trong nhiều trường hợp việc sử dụng các máy tính PC × 86 cho các ứng dụng nhúng hiệu suất cao là không tiết kiệm tiền bạc, nhưng nó làm rút ngắn thời gian phát triển vì có một thư viện phần mềm bao la đã được viết cho nền DOS và Windows. Thực tế là Windows là một nền được sử dụng rộng rãi và dễ hiểu có nghĩa là việc phát triển một sản phẩm nhúng dựa trên Windows làm giảm giá thành và rút ngắn thời gian phát triển đáng kể.

1.1.4 Lựa chọn một bộ vi điều khiển.

Có 4 bộ vi điều khiển 8 bit chính. Đó là 6811 của Motorola, 8051 của Intel z8 của Xilog và Pic 16 × của Microchip Technology. Mỗi một kiểu loại trên đây đều có một tập lệnh và thanh ghi riêng duy nhất, nếu chúng đều không tương thích lẫn nhau. Cũng có những bộ vi điều khiển 16 bit và 32 bit được sản xuất bởi các hãng sản xuất chíp khác nhau. Với tất cả những bộ vi điều khiển khác nhau như thế này thì lấy gì làm tiêu chuẩn lựa chọn mà các nhà thiết kế phải cân nhắc? Có ba tiêu chuẩn để lựa chọn các bộ vi điều khiển là:

1) Đáp ứng nhu cầu tính toán của bài toán một cách hiệu quả về mặt giá thành và đầy đủ chức năng có thể nhìn thấy được (khả dĩ).

2) Có sẵn các công cụ phát triển phần mềm chẳng hạn như các trình biên dịch, trình hợp ngữ và gỡ rối.

3) Nguồn các bộ vi điều khiển có sẵn nhiều và tin cậy.

1.1.5 Các tiêu chuẩn lựa chọn một bộ vi điều khiển.

1. Tiêu chuẩn đầu tiênvà trước hết tronglựa chọn một bộ vi điều khiển là nó phải đáp ứng nhu cầu bài toán về một mặt công suất tính toán và giá thành hiệu quả. Trong khi phân tích các nhu cầu của một dự án dựa trên bộ vi điều khiển chúng ta trước hết phải biết là bộ vi điều khiển nào 8 bit, 16 bit hay 32 bit có thể đáp ứng tốt nhất nhu cầu tính toán của bài toán mộ tcách hiệu quả nhất? Những tiêu chuẩn được đưa ra để cân nhắc là:

a) Tốc độ: Tốc độ lớn nhất mà bộ vi điều khiển hỗ trợ là bao nhiêu.

b) Kiểu đóng vỏ: Đó là kiểu 40 chân DIP hay QFP hay là kiểu đóng vỏ khác (DIP -đóng vỏ theo 2 hàng chân. QFP là đóng vỏ vuông dẹt)? Đây là điều quan trọng đối với yêu cầu về không gian, kiểu lắp ráp và tạo mẫu thử cho sản phẩm cuối cùng.

c) Công suất tiêu thụ: Điều này đặc biệt khắt khe đối với những sản phẩm dùng pin, ắc quy.

d) Dung lượng bộ nhớ RAM và ROM trên chíp.
e) Số chân vào - ra và bộ định thời trên chíp
f) Khả năng dễ dàng nâng cấp cho hiệu suất cao hoặc giảm công suất tiêu thụ.
g) Giá thành cho một đơn vị: Điều này quan trọng quyết định giá thành cuối cùng của sản phẩm mà một bộ vi điều khiển được sử dụng. Ví dụ có các bộ vi điều khiển giá 50 cent trên đơn vị khi được mua 100.000 bộ một lúc.
2) Tiêu chuẩn thứ hai trong lựa chọn một bộ vi điều khiển là khả năng phát triển các sản phẩm xung quanh nó dễ dàng như thế nào? Các câu nhắc chủ yếu bao gồm khả năng có sẵn trình lượng ngữ, gỡ rối, trình biên dịch ngôn ngữ C hiệu quả về mã nguồn, trình mô phỏng hỗ trợ kỹ thuật và khả năng sử dụng trong nhà và ngoài môi trường. Trong nhiều trường hợp sự hỗ trợ nhà cung cấp thứ ba (nghĩa là nhà cung cấp khác không phải là hãng sản xuất chíp) cho chíp cũng tốt như, nếu không được tốt hơn, sự hỗ trợ từ nhà sản xuất chíp.
3) Tiêu chuẩn thứ ba trong lựa chọn một bộ vi điều khiển là khả năng sẵn sàng đáp ứng về số lượng trong hiện tại và tương lai. Đối với một số nhà thiết kế điều này thậm chí còn quan trong hơn cả hai tiêu chuẩn đầu tiên. Hiện nay, các bộ vi điều khiển 8 bit dấu đầu, họ 8051 là có số lương lớn nhất các nhà cung cấp đa dạng (nhiều nguồn). Nhà cung cấp có nghĩa là nhà sản xuất bên cạnh nhà sáng chế của bộ vi điều khiển. Trong trường hợp 8051 thì nhà sáng chế của nó là Intel, nhưng hiện nay có rất nhiều hãng sản xuất nó (cũng như trước kia đã sản xuất).

Các hãng này bao gồm: Intel, Atmel, Philips/signe-tics, AMD, Siemens, Matra và Dallas, Semicndictior.

Bảng 1.2: Địa chỉ của một số hãng sản xuất các thành viên của họ 8051.

Hãng	**Địa chỉ Website**
Intel	www.intel.com/design/mcs51
Antel	www.atmel.com
Plips/ Signetis	www.semiconductors.philips.com
Siemens	www.sci.siemens.com
Dallas Semiconductor	www.dalsemi.com

Cũng nên lưu ý rằng Motorola, Zilog và Mierochip Technology đã dành một lượng tài nguyên lớn để đảm bảo khả năng sẵn sàng về một thời gian và phạm vi rộng cho các sản phẩm của họ từ khi các sản phẩm của họ đi vào sản xuất ổn định, hoàn thiện và trở thành nguồn chính. Trong những năm gần đây họ cũng đã bắt đầu bán tế bào thư viện Asic của bộ vi điều khiển.

1.2 Tổng quan về họ 8051.

Trong mục này chúng ta xem xét một số thành viên khác nhau của họ bộ vi điều khiển 8051 và các đặc điểm bên trong của chúng. Đồng thời ta điểm qua một số nhà sản xuất khác nhau và các sản phẩm của họ có trên thị trường.

1.2.1 Tóm tắt về lịch sử của 8051.

Vào năm 1981. Hãng Intel giới thiệu một số bộ vi điều khiển được gọi là 8051. Bộ vi điều khiển này có 128 byte RAM, 4K byte ROM trên chíp, hai bộ định thời, một cổng nối tiếp và 4 cổng (đều rộng 8 bit) vào ra tất cả được đặt trên một chíp. Lúc ấy nó được coi là một "hệ thống trên chíp". 8051 là một bộ xử lý 8 bit có nghĩa là CPU chỉ có thể làm việc với 8 bit dữ liệu tại một thời điểm. Dữ liệu lớn hơn 8 bit được chia ra thành các dữ liệu 8 bit để cho xử lý. 8051 có tất cả 4 cổng vào - ra I/O mỗi cổng rộng 8 bit (xem hình 1.2). Mặc dù 8051 có thể có một ROM trên chíp cực đại là 64 K byte, nhưng các nhà sản xuất lúc đó đã cho xuất xưởng chỉ với 4K byte ROM trên chíp. Điều này sẽ được bàn chi tiết hơn sau này.

8051 đã trở nên phổ biến sau khi Intel cho phép các nhà sản xuất khác sản xuất và bán bất kỳ dạng biến thế nào của 8051 mà họ thích với điều kiện họ phải để mã lại tương thích với 8051. Điều này dẫn đến sự ra đời nhiều phiên bản của 8051 với các tốc độ khác nhau và dung lượng ROM trên chíp khác nhau được bán bởi hơn nửa các nhà sản xuất. Điều này quan trọng là mặc dù có nhiều biến thể khác nhau của 8051 về tốc độ và dung lương nhớ ROM trên chíp, nhưng tất cả chúng đều tương thích với 8051 ban đầu về các lệnh. Điều này có nghĩa là nếu ta viết chương trình của mình cho một phiên bản nào đó thì nó cũng sẽ chạy với mọi phiên bản bất kỳ khác mà không phân biệt nó từ hãng sản xuất nào.

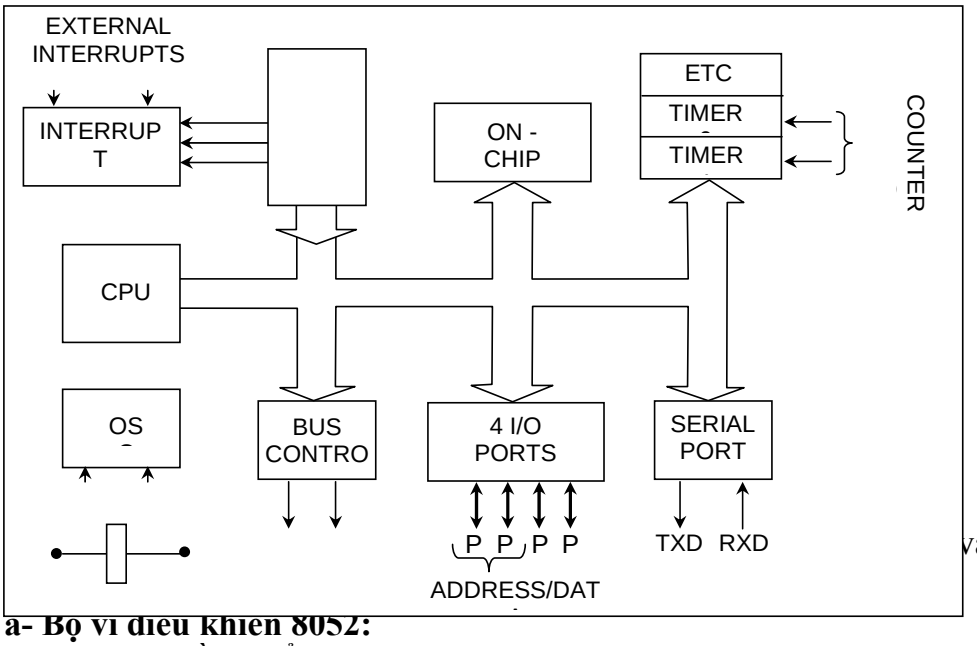
Bảng 1.3: Các đặc tính của 8051 đầu tiên.

Đặc tính	**Số lượng**
ROM trên chíp	4K byte
RAM	128 byte

Bộ định thời	2
Các chân vào - ra	32
Cổng nối tiếp	1
Nguồn ngắt	6

1.2.2 Bộ ví điều khiển 8051

Bộ vi điều khiển 8051 là thành viên đầu tiên của họ 8051. Hãng Intel ký hiệu nó như là MCS51. Bảng 3.1 trình bày các đặc tính của 8051.



và

a- Bộ vi điều khiển 8052:

Bộ vi điều khiển 8052 là một thành viên khác của họ 8051, 8052 có tất cả các đặc tính chuẩn của 8051 ngoài ra nó có thêm 128 byte RAM và một bộ định thời nữa. Hay nói cách khác là 8052 có 256 byte RAM và 3 bộ định thời. Nó cũng có 8K byte ROM. Trên chíp thay vì 4K byte như 8051. Xem bảng 1.4.

Bảng1.4: so sánh các đặc tính của các thành viên họ 8051.

Đặc tính	**8051**	**8052**	**8031**
ROM trên chíp	4K byte	8K byte	OK
RAM	128 byte	256 byte	128 byte
Bộ định thời	2	3	2
Chân vào - ra	32	32	32
Cổng nối tiếp	1	1	1
Nguồn ngắt	6	8	6

Như nhìn thấy từ bảng 1.4 thì 8051 là tập con của 8052. Do vậy tất cả mọi chương trình viết cho 8051 đều chạy trên 8052 nhưng điều ngược lại là không đúng.

b- Bộ vi điều khiển 8031:

Một thành viên khác nữa của 8051 là chíp 8031. Chíp này thường được coi như là 8051 không có ROM trên chíp vì nó có OK byte ROM trên chíp. Để sử dụng chíp này ta phải bổ xung ROM ngoài cho nó. ROM ngoài phải chứa chương trình mà 8031 sẽ nạp và thực hiện. So với 8051 mà chương trình được chứa trong ROM trên chíp bị giới hạn bởi 4K byte, còn ROM ngoài chứa chương trinh được gắn vào 8031 thì có thể lớn đến 64K byte. Khi bổ xung cổng, như vậy chỉ còn lại 2 cổng để thao tác. Để giải quyết vấn đề này ta có thể bổ xung cổng vào - ra cho 8031. Phối phép 8031 với bộ nhớ và cổng vào - ra chẳng hạn với chíp 8255 được trình bày ở chương 14. Ngoài ra còn có các phiên bản khác nhau về tốc độ của 8031 từ các hãng sản xuất khác nhau.

1.2.4. Các bộ vi điều khiển 8051 từ các hãng khác nhau.

Mặc dù 8051 là thành viên phổi biến nhất của họ 8051 nhưng chúng ta sẽ thấy nó trong kho linh kiện. Đó là do 8051 có dưới nhiều dạng kiểu bộ nhớ khác nhau như UV - PROM, Flash và NV - RAM mà chúng đều có số đăng ký linh kiện khác nhau. Việc bàn luận về các kiểu dạng bộ nhớ ROM khác nhau sẽ được trình bày ở chương 14. Phiên bản UV-PROM của 8051 là 8751. Phiên bản Flash ROM được bán bởi nhiều hãng khác nhau chẳng hạn của Atmel corp với tên gọi là AT89C51 còn phiên bản NV-RAM của 8051 do Dalas Semi Conductor cung cấp thì được gọi là DS5000. Ngoài ra còn có phiên bản OTP (khả trình một lần) của 8051 được sản xuất bởi rất nhiều hãng.

a- Bộ vi điều khiển 8751:

Chíp 8751 chỉ có 4K byte bộ nhớ UV-EPROM trên chíp. Để sử dụng chíp này để phát triển yêu cầu truy cập đến một bộ đốt PROM cũng như bộ xoá UV- EPROM để xoá nội dung của bộ nhớ UV-EPROM bên trong 8751 trước khi ta có thể lập trình lại nó. Do một thực tế là ROM trên chíp đối với 8751 là UV-EPROM nên cần phải mất 20 phút để xoá 8751 trước khi nó có thể được lập trình trở lại. Điều này đã dẫn đến nhiều nhà sản xuất giới thiệu các phiên bản Flash Rom và UV-RAM của 8051. Ngoài ra còn có nhiều phiên bản với các tốc độ khác nhau của 8751 từ nhiều hãng khác nhau.

b- Bộ vi điều khiển AT8951 từ Atmel Corporation.

Chíp 8051 phổ biến này có ROM trên chíp ở dạng bộ nhớ Flash. Điều này là lý tưởng đối với những phát triển nhanh vì bộ nhớ Flash có thể được xoá trong vài giây trong tương quan so với 20 phút hoặc hơn mà 8751 yêu cầu. Vì lý do này mà AT89C51 để phát triển một hệ thống dựa trên bộ vi điều khiển yêu cầu một bộ đốt ROM mà có hỗ trợ bộ nhớ Flash. Tuy nhiên lại không yêu cầu bộ xoá ROM. Lưu ý rằng trong bộ nhớ Flash ta phải xoá toàn bộ nội dung của ROM nhằm để lập trình lại cho nó. Việc xoá bộ nhớ Flash được thực hiện bởi chính bộ đốt PROM và đây chính là lý do tại sao lại không cần đến bộ xoá. Để loại trừ nhu cầu đối với một bộ đốt PROM hãng Atmel đang nghiên cứu một phiên bản của AT 89C51 có thể được lập trình qua cổng truyền thông COM của máy tính IBM PC .

Bảng 1.5: Các phiên bản của 8051 từ Atmel (Flash ROM).

Số linh kiện	**ROM**	**RAM**	**Chân I/O**	**Timer**	**Ngắt**	**Vcc**	**Đóng v**
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

Chữ C trong ký hiệu AT89C51 là CMOS.

Cũng có những phiên bản đóng vỏ và tốc độ khác nhau của những sản phẩm trên đây. Xem bảng 1.6. Ví dụ để ý rằng chữ “C” đứng trước số 51 trong AT 89C51 -12PC là ký hiệu cho CMOS “12” ký hiệu cho 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). Thông thường AT89C51 - 12PC rát lý tưởng cho các dự án của học sinh, sinh viên.

Bảng 1.6: Các phiên bản 8051 với tốc độ khác nhau của Atmel.

Mã linh kiện	**Tốc độ**	**Số chân**	**Đóng vỏ**	**Mục đích**
AT89C51-	42MHZ	40	DTP	Thương

12PC				mại

c- Bộ vi điều khiển DS5000 từ hãng Dallas Semiconductor.

Một phiên bản phổ biến khác nữa của 8051 là DS5000 của hãng Dallas Semiconductor. Bộ nhớ ROM trên chíp của DS5000 ở dưới dạng NV-RAM. Khả năng đọc/ ghi của nó cho phép chương trình được nạp vào ROM trên chíp trong khi nó vẫn ở trong hệ thống (không cần phải lấy ra). Điều này còn có thể được thực hiện thông qua cổng nối tiếp của máy tính IBM PC. Việc nạp chương trình trong hệ thống (in-system) của DS5000 thông qua cổng nối tiếp của PC làm cho nó trở thành một hệ thống phát triển tại chỗ lý tưởng. Một ưu việt của NV-RAM là khả năng thay đổi nội dung của ROM theo từng byte tại một thời điểm. Điều này tương phản với bộ nhớ Flash và EPROM mà bộ nhớ của chúng phải được xoá sạch trước khi lập trình lại cho chúng.

Bảng 1.7: Các phiên bản 8051 từ 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

Chữ "T" đứng sau 5000 là có đồng hồ thời gian thực.

Lưu ý rằng đồng hồ thời gian thực RTC là khác với bộ định thời Timer. RTC tạo và giữ thời gian l phút giờ, ngày, tháng - năm kể cả khi tắt nguồn.

Còn có nhiều phiên bản DS5000 với những tốc độ và kiểu đóng gói khác nhau.(Xem bảng 1.8). Ví dụ DS5000-8-8 có 8K NV-RAM và tốc đọ 8MHZ. Thông thường DS5000-8-12 hoặc DS5000T-8-12 là lý tưởng đối với các dự án của sinh viên.

Bảng 1.8: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 (with RTC)
DS5000-32-12	32K	12MHz
DS5000-8-12	8K	12MHz (with RTC)

d- Phiên bản OTP của 8051.

Các phiên bản OTP của 8051 là các chíp 8051 có thể lập trình được một lần và được cung cấp từ nhiều hãng sản xuất khác nhau. Các phiên bản Flash và NV-RAM thường được dùng để phát triển sản phẩm mẫu. Khi một sản pohẩm được thiết kế và được hoàn thiện tuyệt đối thì phiên bản OTP của 8051 được dùng để sản hàng loạt vì nó sẽ hơn rất nhiều theo giá thành một đơn vị sản phẩm

e- Họ 8051 từ Hãng Philips

Một nhà sản xuất chính của họ 8051 khác nữa là Philips Corporation. Thật vậy, hãng này có một dải lựa chọn rộng lớn cho các bộ vi điều khiển họ 8051. Nhiều sản phẩm của hãng đã có kèm theo các đặc tính như các bộ chuyển đổi ADC, DAC, cổng I/0 mở rộng và cả các phiên bản OTP và Flash.

Chương 2
Lập trình hợp ngữ 8051

2.1 Bên trong 8051.

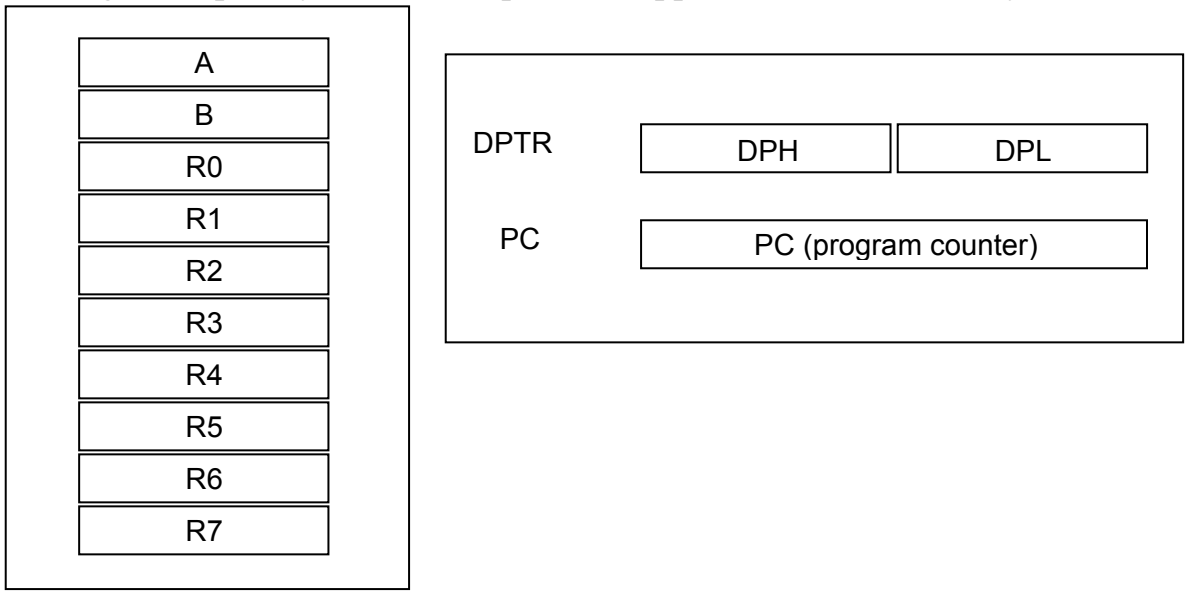
Trong phần này chúng ta nghiên cứu các thanh ghi chính của 8051 và trình bày cách sử dụng với các lệnh đơn giản MOV và ADD.

2.1.1 Các thanh ghi.

Trong CPU các thanh ghi được dùng để lưu cất thông tin tạm thời, những thông tin này có thể là một byte dữ liệu cần được sử lý hoặc là một địa chỉ đến dữ liệu cần được nạp. Phần lớn các thanh ghi của 8051 là các thanh ghi 8 bit. Trong 8051 chỉ có một kiểu dữ liệu: Loại 8 bit, 8 bit của một thanh ghi được trình bày như sau:

D7	D6	D5	D4	D3	D2	D1	D0

với MSB là bit có giá trị cao nhất D7 cho đến LSB là bit có giá trị thấp nhất D0. (MSB - Most Sigfican bit và LSB - Leart Significant Bit). Với một kiểu dữ liệu 8 bit thì bất kỳ dữ liệu nào lớn hơn 8 bit đều phải được chia thành các khúc 8 bit trước khi được xử lý. Vì có một số lượng lớn các thanh ghi trong 8051 ta sẽ tập trung vào một số thanh ghi công dụng chung đặc biệt trong các chương kế tiếp. Hãy tham khảo phụ lục Appendix A.3 để biết đầy đủ về các



thanh ghi của 8051.

Hình 2.1: a) Một số thanh ghi 8 bit của 8051

b) Một số thanh ghi 16 bit của 8051

Các thanh ghi được sử dụng rộng rãi nhất của 8051 là A (thanh ghi tích luỹ), B, R0 - R7, DPTR (con trỏ dữ liệu) và PC (bộ đếm chương trình). Tất cả các dữ liệu trên đều là thanh g hi 8 bit trừ DPTR và PC là 16 bit. Thanh ghi tích luỹ A được sử dụng cho tất cả mọi phép toán số học và lô-gíc. Để hiểu sử

dụng các thanh ghi này ta sẽ giới thiệu chúng trong các ví dụ với các lệnh đơn giản là ADD và MOV.

2.1.2 Lệnh chuyển MOV.

Nói một cách đơn giản, lệnh MOV sao chép dữ liệu từ một vị trí này đến một ví trí khác. Nó có cú pháp như sau:

```
MOV             ; Đích, nguồn; sao chép nguồn vào đích
```

Lệnh này nói CPU chuyển (trong thực tế là sao chép) toán hạng nguồn vào toán hạng đích. Ví dụ lệnh "MOV A, R0" sao chép nội dung thanh ghi R0 vào thanh ghi A. Sau khi lênh này được thực hiện thì thanh ghi A sẽ có giá trị giống như thanh ghi R0. Lệnh MOV không tác động toán hạng nguồn. Đoạn chương trình dưới đây đầu tiên là nạp thanh ghi A tới giá trị 55H 9là giá trị 55 ở dạng số Hex) và sau đó chuyển giá trị này qua các thanh ghi khác nhau bên trong CPU. Lưu ý rằng dấu "#" trong lệnh báo rằng đó là một giá trị. Tầm quan trọng của nó sẽ được trình bày ngay sau ví dụ này.

```
MOV A, #55H;    ; Nạp trí trị 55H vào thanh ghi A (A = 55H)
MOV R0, A       ; Sao chép nội dung A vào R0 (bây giờ R0=A)
MOV R1, A       ; Sao chép nội dung A và R1 (bây giờ R1=R0=A)
MOV R2, A       ; Sao chép nội dung A và R2 (bây giờ R2=R1=R0=A)
MOV R3, #95H    ; Nạp giá trị 95H vào thanh ghi R3 (R3 = 95H)
MOV A, R3       ; Sáo chép 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 vào trực tiếp bất kỳ thanh ghi nào A, B, R0 - R7. Tuy nhiên, để thông báo đó là giá trị tức thời thì phải đặt trước nó một ký hiệu "#" như chỉ ra dưới đây.

```
MOV A, #23H     ; Nạp giá trị 23H vào A (A = 23H)
MOV R0, #12H    ; Nạp giá trị 12H vào R0 (R0 = 2BH)
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 giá trị thập phân 12 = 0CH vào R6
                (trong R6 có giá trị 0CH).
```

Để ý trong lệnh "MOV R5, #0F9H" thì phải có số 0 đứng trước F và sau dấu # báo rằng F là một số Hex chứ không phải là một ký tự. Hay nói cách khác "MOV R5, #F9H" sẽ gây ra lỗi.

2. Nếu các giá trị 0 đến F được chuyển vào một thanh ghi 8 bit thì các bit còn lại được coi là tất cả các số 0. Ví dụ, trong lệnh "MOV A,#5" kết quả là A=0.5, đó là A = 0000 0101 ở dạng nhị phân.
3. Việc chuyển một giá trị lớn hơn khả năng chứa của thanh ghi 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 phải gán dấu "#" trước giá trị đó. Nếu không có dấu thì nó hiểu rằng nạp từ một vị trí nhớ. Ví dụ "MOV A, 17H" có nghĩa là nạp giá trị trong ngăn nhớ có giá trị 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 là 17H vào thanh ghi A thì cần phải có dấu "#" trước 17H như thế này. "MOV A, #17H". Cần lưu ý rằng nếu thiếu dấu "#" trước một thì 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. Đây sẽ là một lỗi thường hay gặp đối với lập trình viên mới.

2.1.3 Lệnh cộng ADD.

Lệnh cộng ADD có các phép như sau:

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

Lệnh cộng ADD nói CPU cộng byte nguồn vào thanh ghi A và đặt kết quả thanh ghi A. Để cộng hai số như 25H và 34H thì mỗi số có thể chuyển đến một thanh ghi và sau đó cộng lại với nhau như:

```
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 (vì 25H + 34H = 59H) và R2 = 34H, chú ý là nội dụng R2 không thay đổi. Chương trình trên có thể viết theo nhiều cách phụ thuộc vào thanh ghi được sử dụng. Một trong cách viết khác có thể là:

```
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 nội dung R7 vào A (A = A + R7 = 25H + 34H)
```

Chương trình trên có kết quả trong A Là 59H, có rất nhiều cách để viết chương trình giống như vậy. Một câu hỏi có thể đặt ra sau khi xem đoạn chương trình trên là liệu có cần chuyển cả hai dữ liệu vào các thanh ghi trước

khi cộng chúng với nhau không? Câu trả lời là không cần. Hãy xem đoạn chương trình dưới đây:

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

Trong trường hợp trên đây, khi thanh ghi A đã chứa số thứ nhất thì giá trị thứ hai đi theo một toán hạng. Đây được gọi là toán hạng tức thời (trực tiếp).
Các ví dụ trước cho đến giờ thì lệnh ADD báo rằng toán hạng nguồn có thể hoặc là một thanh ghi hoặc là một dữ liệu trực tiếp (tức thời) nhưng thanh ghi đích luôn là thanh ghi A, thanh ghi tích luỹ. Hay nói cách khác là một lệnh như "ADD R2, #12H" là lệnh không hợp lệ vì mọi phép toán số học phải cần đến thanh ghi A và lệnh "ADD R4, A" cũng không hợp lệ vì A luôn là thanh ghi đích cho mọi phép số học. Nói một cách đơn giản là trong 8051 thì mọi phép toán số học đều cần đến thanh A với vai trò là toán hạng đích. Phần trình bày trên đây giải thích lý do vì sao thanh ghi A như là thanh thi tích luỹ. Cú pháp các lệnh hợp ngữ mô tả cách sử dụng chúng và liệt kê các kiểu toán hạng hợp lệ được cho trong phụ lục Appendix A.1.
Có hai thanh ghi 16 bit trong 8051 là bộ đếm chương trình PC và con trỏ dữ liệu DPTR. Tầm quan trọng và cách sử dụng chúng được trình bày ở mục 2.3. Thanh ghi DPTR được sử dụng để truy cập dữ liệu và được làm kỹ ở chương 5 khi nói về các chế độ đánh địa chỉ.

2.2 Giới thiệu về lập trình hợp ngữ 8051.

Trong phần này chúng ta bàn về dạng thức của hợp ngữ và định nghĩa một số thuật ngữ sử dụng rộng rãi gắn liền với lập trình hợp ngữ.
CPU chỉ có thể làm việc với các số nhị phân và có thể chạy với tốc độ rất cao. Tuy nhiên, thật là ngán ngậm và chậm chạp đối với con người phải làm việc với các số 0 và 1 để lập trình cho máy tính. Một chương trình chứa các số 0 và 1 được gọi là ngôn ngữ máy.
Trong những ngày đầu của máy tính, các lập trình viên phải viết mã chương trình dưới dạng ngôn ngữ máy. Mặc dụ hệ thống thập lục phân (số Hex) đã được sử dụng như một cách hiệu quả hơn để biểu diễn các số nhị phân thì quá trình làm việc với mã máy vẫn còn là công việc cồng kềnh đối với con người. Cuối cùng, các nguồn ngữ hợp ngữ đã được phát, đã cung cấp các từ gợi nhớ cho các lệnh mã máy cộng với những đặc tính khác giúp cho việc lập trình nhanh hơn và ít mắc lỗi hơn. Thuật ngữ từ gợi nhớ (mnemonic) thường xuyên sử dụng trong tài liệu khoa học và kỹ thuật máy tính để tham chiếu cho các mã và từ rút gọn tương đối dễ nhớ, các chương trình hợp ngữ phải được dịch ra thanh mã máy bằng một chương trình được là trình hợp ngữ (hợp dịch). Hợp ngữ được coi như là một ngông ngữ bậc thấp vì nó giao tiếp trực tiếp với cấu trúc bên trong của CPU. Để lập trình trong hợp ngữ, lập trình viên phải biết tất cả các thanh ghi của CPU và kích thước của chúng cũng như các chi tiết khác.

Ngày nay, ta có thể sử dụng nhiều ngôn ngữ lập trình khác nhau, chẳng hạn như Basic, Pascal, C, C^{++}, Java và vô số ngôn ngữ khác. Các ngôn ngữ này được coi là nhưng ngôn ngữ bậc cao vì lập trình viên không cần phải tương tác với các chi tiết bên trong của CPU. Một trình hợp dịch được dùng để dịch chương trình hợp ngữ ra mã máy còn (còn đôi khi cũng còn được gọi mà đối tượng (Object Code) hay mã lệnh Opcode), còn các ngôn ngữ bậc cao được dịch thành các ngôn ngữ mã máy bằng một chương trình gọi là trình biên dịch. Ví dụ, để viết một chương trình trong C ta phải sử dụng một trình biên dịch C để dịch chương trình về dạng mã máy. Bây giờ ta xét dạng thức hợp ngữ của 8051 và sử dụng trình hợp dịch để tạo ra một chương trình sẵn sàng chạy ngay được.

2.2.1 Cấu trúc của hợp ngữ.

Một chương trình hợp ngữ bao gồm một chuỗi các dòng lệnh hợp ngữ. Một lệnh hợp ngữ có chứa một từ gợi nhớ (mnemonic) và tuy theo từng lệnh và sau nó có một hoặc hai toán hạng. Các toán hạng là các dữ liệu cần được thao tác và các từ gợi nhớ là các lệnh đối với CPU nói nó làm gì với các dữ liệu.

```
        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 nôi dụng 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 này
        END              ; Kết thúc tệp nguồn hợp ngữ
```

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

Chương trình 2.1 cho trên đây là một chuỗi các câu lệnh hoặc các dòng lệnh được viết hoặc bằng các lệnh hợp ngữ như ADD và MOV hoặc bằng các câu lệnh được gọi là các chỉ dẫn. Trong khi các lệnh hợp ngữ thì nói CPU phải làm gì thì các chỉ lệnh (hay còn gọi là giả lệnh) thì đưa ra các chỉ lệnh cho hợp ngữ. Ví dụ, trong chương trình 2.1 thì các lệnh ADD và MOV là các lệnh đến CPU, còn ORG và END là các chỉ lệnh đối với hợp ngữ. ORG nói hợp ngữ đặt mã lệnh tại ngăn nhớ 0 và END thì báo cho hợp ngữ biết kết thúc mã nguồn. Hay nói cách khác một chỉ lệnh để bắt đầu và chỉ lệnh thứ hai để kết thúc chương trình.

Cấu trúc của một lệnh hợp ngữ có 4 trường như sau:

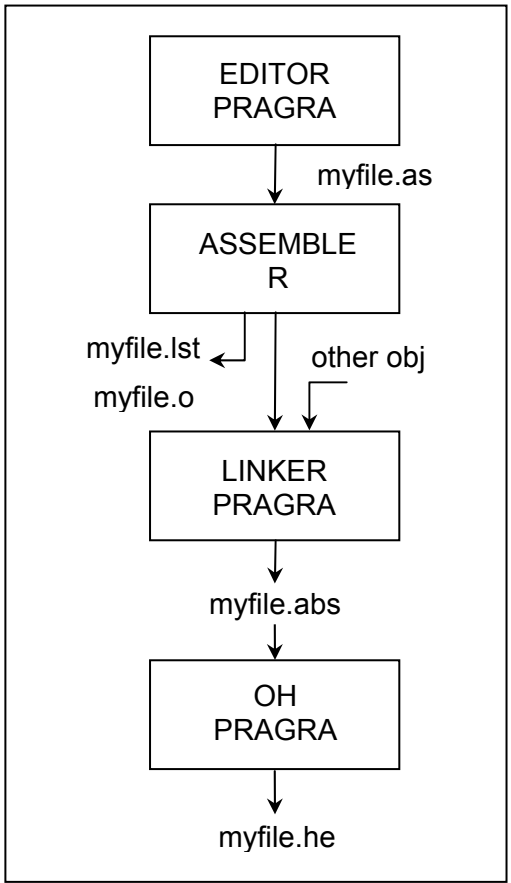
[nhãn:] [từ gợi nhớ] [các toán hạng] [; chú giải]

Các trường trong dấu ngoặc vuông là tuỳ chọn và không phải dòng lệnh nào cũng có chúng. Các dấu ngoặc vuông không được viết vào. Với dạng thức trên đây cần lưu ý các điểm sau:

1. Trường nhãn cho phép chương trình tham chiếu đến một dòng lệnh bằng tên. Nó không được viết quá một số ký tự nhất định. Hãy kiểm tra quy định này của hợp ngữ mà ta sử dụng.
2. Từ gợi nhớ (lệnh) và các toán hạng là các trường kết hợp với nhau thực thi công việc thực tế của chương trình và hoàn thiện các nhiệm vụ mà chương trình được viết cho chúng. Trong hợp ngữ các câu lệnh như:

   " ADD A, B"
   "MOV A, #67H"

   thì ADD và MOV là những từ gợi nhớ tạo ra mã lệnh, còn "A, B" và "A, #67H" là những toán hạng thì hai trường có thể chứa các lệnh giả hoặc chỉ lệnh của hợp ngữ. Hãy nhớ rằng các chỉ lệnh không tạo ra mã lệnh nào (mã máy) và chúng chỉ dùng bởi hợp ngữ, ngược lại đối với các lệnh là chúng được dịch ra mã máy (mã lênh) cho CPU thực hiện. Trong chương trình 2.1 các lệnh ORG và END là các chỉ lệnh (một số hợp ngữ của 8051 sử dụng dạng .ORG và .END). Hãy đọc quy định cụ thể của hợp ngữ ta sử dụng.
3. Chương chú giải luôn phải bắt đầu bằng dấu chấm phẩy (;). Các chú giải có thể bắt đầu ở đầu dòng hoặc giữa dòng. Hợp ngữ bỏ qua (làm ngơ) các chú giải nhưng chúng lại rất cần thiết đối với lập trình viên. Mặc dù các chú giải là tuỳ chọn, không bắt buộc nhưng ta nên dùng chúng để mô tả chương trình để giúp cho người khác đọc và hiểu chương trình dễ dàng hơn.
4. Lưu ý đến nhãn HERE trong trường nhãn của chương trình 2.1. Một nhãn bất kỳ tham chiếu đến một lệnh phải có dấu hai chấm (:) đứng ở sau. Trong câu lệnh nhảy ngắn SJMP thì 8051 được ra lệnh ở lại trong vòng lặp này vô hạn. Nếu hệ thống của chúng ta có một chương trình giám sát thì takhông cần dòng lệnh này và nó có thể được xoá đi ra khỏi chương trình.



2.3 Hợp dịch và chạy một chương trình 8051.

Như vậy cấu trúc của một chương trình hợp ngữ ta đã được biết, câu hỏi đặt ra là chương trình sẽ được tạo ra và hợp dịch như thế nào và làm thế nào để có thể chạy được? Các bước để tạo ra một chương trình hợp ngữ có thể chạy được là:

1. Trước hết ta sử dụng mộ trình soạn thảo để gỡ vào một chương trình giống như chương trình 2.1. Có nhiều trình soạn thảo tuyệt vời hoặc các bộ sử lý từ được sử dụng để tạo ra và/ hoặc để soạn thảo chương trình. Một trình soạn thảo được sử dụng rộng rãi là trình soạn thảo EDIT của MS-DOS (hoặc Noterad của Windows) đều chạy trên hệ điều hành Microsoft. Lưu ý rằng, trình soạn thảo phải có khả năng tạo ra tệp mã ASCII. Đối với nhiều trình hợp ngữ thì các tên tệp tuân theo các quy ước thường lệ củ DOS, nhưng phần mở rộng của các tệp nguồn phải là "asm" hay "src" tuỳ theo trình hợp ngữ mà ta sử dụng.
2. Tệp nguồn có phần mở rộng "asm" chứa mã chương trình được tạo ra ở bước 1 được nạp vào trình hợp dịch của 8051. Trình hợp dịch chuyển các lệnh ra mã máy. Trình hợp dịch sẽ tạo ra một tệp đối tượng và một tệp liệt kê với các thành phần mở rộng "obj" và "lst" tương ứng.
3. Các trình hợp dịch yêu cầu một bước thứ ba gọi là liên kết. Chương trình liên kết lấy một hoặc nhiều tệp đối tượng và tạo ra một tệp đối tượng tuyệt đối với thành phần mở rộng "abs". Tệp "abs" này được sử dụng bởi thùng chứa của 8051 có một chương trình giám sát.
4. Kế sau đó tệp "abs" được nạp vào một chương trình được gọi là "0H" (chuyển đối tượng object về dạng số Hex) để tạo ra một tệp với đuôi mở rộng "Hex" có thể nạp tốt vào trong ROM. Chương trình này có trong tất cả mọi trình hợp ngữ của 8051 các trình hợp ngữ dựa trên Windows hiện nay kết hợp các bước 2 đến 4 vào thành một bước.
   Hình 2.2: Các bước để tạo ra một chương trình.

2.3.1 Nói thêm về các tệp ".asm" và ".object".

Tệp ".asm" cũng được gọi là tệp nguồn và chính vì lý do này mà một số trình hợp ngữ đòi hỏi tệp này phải có một phần mở rộng "src" từ chữ "source" là nguồn. Hãy kiểm tra hợp ngữ 8051 mà ta sử dụng xem nó có đòi hỏi như vậy không? Như ta nói trước đây tệp này được tạo ra nhờ một trình biên tập chẳng hạn như Edit của DOS hoặc Notepad của Windows. Hợp ngữ của 8051 chuyển đổi các tệp hợp ngữ trong tệp .asm thành ngôn ngữ mã máy và cung cấp tệp đối tượng .object. Ngoài việc tạo ra tệp đối tượng trình hợp ngữ cũng cho ra tệp liệt kê "lst" (List file).

2.3.2 Tệp liệt kê ".lst".

Tệp liệt kê là một tuỳ chọn, nó rất hữu ích cho lập trình viên vì nó liệt kê tất cả mọi mã lệnh và địa chỉ cũng như tất cả các lỗi mà trình hợp ngữ phát hiện ra. Nhiều trình hợp ngữ giả thiết rằng, tệp liệt kê là không cần thiết trừ khi ta báo rằng ta muốn tạo ra nó. Tệp này có thể được truy cập bằng một trình biên dịch như Edit của DOS hoặc Notepad của Window và được hiển thị trên màn hình hoặc được gửi ra máy in. Lập trình viên sử dụng tệp liệt kê để tìm 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 giá trị 25H vào R5
3 0002 7F34         MOV R7, #34H     ; Nạp giá trị 34H vào R7
4 0004      7400        MOV A, #0        ; Nạp 0 vào A (xoá A)
5 0006 2D           ADD A, R5        ; Cộng nội dung R5 vào A (A = A +
R5)
6 0007 2F           ADD A, R7        ; Cộng nội dung R7 vào A (A = A +
R7)
7 0008 2412         ADD A, #12H      ; Cộng giá trị 12H vào A (A = A +
12H)
8 00A BCEF HERE:    SJMP HERE              ; ở lại vòng lặp này
9 000C                  END              ; Kết thúc tệp .asm
```

Chương trình 2.2: Tệp liệt kê.

2.4 Bộ đếm chương trình và không gian ROM trong 8051.

2.4.1 Bộ đếm chương trình trong 8051.

Một thanh ghi quan trọng khác trong 8051 là bộ đếm chương trình . Bộ đếm chương trình chỉ đếm địa chỉ của lệnh kế tiếp cần được thực hiện. Khi CPU nạp mã lệnh từ bộ nhớ ROM chương trình thì bộ đếm chương trình tăng lên chỉ đếm lệnh kết tiếp. Bộ đếm chương trình trong 8051 có thể truy cập các địa chỉ chương trình trong 8051 rộng 16 bit. Điều này có nghĩa là 8051 có thể truy cập các địa chỉa chương trình từ 0000 đến FFFFH tổng cộng là 64k byte mã lệnh. Tuy nhiên, không phải tất cả mọi thành viên của 8051 đều có tất cả 64k byte ROM trên chíp được cài đặt. Vậy khi 8051 được bật nguồn thì nó đánh thức ở địa chỉa nào?

2.4.2 Địa chỉ bắt đầu khi 8051 được cấp nguồn.

Một câu hỏi mà ta phải hỏi về bộ vi điều khiển bất kỳ là thì nó được cấp nguồn thì nó bắt đầu từ địa chỉ nào? Mỗi bộ vi điều khiển đều khác nhau. Trong trường hợp họ 8051 thì mọi thành viên kể từ nhà sản xuất nào hay phiên bản nào thì bộ vi điều khiển đều bắt đầu từ địa chỉ 0000 khi nó được bật nguồn. Bật nguồn ở đây có nghĩa là ta cấp điện áp V_{cc} đến chân RESET như sẽ trình bày ở chương 4. Hay nói cách khác, khi 8051 được cấp nguồn thì bộ đếm chương trình có giá trị 0000. Điều này có nghĩa là nó chờ mã lệnh đầu tiên được lưu ở địa chỉa ROM 0000H. Vì lý do này mà trong vị trí nhớ 0000H của bộ nhớ ROM chương trình vì đây là nơi mà nó tìm lệnh đầu tiên khi bật nguồn. Chúng ta đạt được điều này bằng câu lệnh ORG trong chương trình nguồn như đã trình bày trước đây. Dưới đây là hoạt động từng bước của bộ đếm chương trình trong qúa trình nạp và thực thi một chương trình mẫu.

2.4.3 Đặt mã vào ROM chương trình.

Để hiểu tốt hơn vai trò của bộ đếm chương trình trong quá trình nạp và thực thi một chương trình, ta khảo sát một hoạt động của bộ đếm chương trình khi mỗi lệnh được nạp và thực thi. Trước hết ta khảo sát một lần nữa tệp liệt kê

của chương trình mẫu và cách đặt mã vào ROM chương trình 8051 như thế nào? Như ta có thể thấy, mã lệnh và toán hạng đối với mỗi lệnh được liệt kê ở bên trái của lệnh liệt kê.

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

Chương trình 2.1 cho trên đây là một chuỗi các câu lệnh hoặc các dòng lệnh được viết hoặc bằng các lệnh hợp ngữ như ADD và MOV hoặc bằng các câu lệnh được gọi là các chỉ dẫn. Trong khi các lệnh hợp ngữ thì nói CPU phải làm gì thì các chỉ lệnh (hay còn gọi là giả lệnh) thì đưa ra các chỉ lệnh cho hợp ngữ. Ví dụ, trong chương trình 2.1 thì các lệnh ADD và MOV là các lệnh đến CPU, còn ORG và END là các chỉ lệnh đối với hợp ngữ. ORG nói hợp ngữ đặt mã lệnh tại ngăn nhớ 0 và END thì báo cho hợp ngữ biết kết thúc mã nguồn. Hay nói cách khác một chỉ lệnh để bắt đầu và chỉ lệnh thứ hai để kết thúc chương trình.

Cấu trúc của một lệnh hợp ngữ có 4 trường như sau:

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

Các trường trong dấu ngoặc vuông là tuỳ chọn và không phải dòng lệnh nào cũng có chúng. Các dấu ngoặc vuông không được viết vào. Với dạng thức trên đây cần lưu ý các điểm sau:

Trường nhãn cho phép chương trình tham chiếu đến một dòng lệnh bằng tên. Nó không được viết quá một số ký tự nhất định. Hãy kiểm tra quy định này của hợp ngữ mà ta sử dụng.

Từ gợi nhớ (lệnh) và các toán hạng là các trường kết hợp với nhau thực thi công việc thực tế của chương trình và hoàn thiện các nhiệm vụ mà chương trình được viết cho chúng. Trong hợp ngữ các câu lệnh như:

“ ADD A, B”
“MOV A, #67H”

Thì ADD và MOV là những từ gơi nhớ tạo ra mã lệnh, còn “A, B” và “A, #67H” là những toán hạng thì hai trường có thể chứa các lệnh giả hoặc chỉ lệnh của hợp ngữ. Hãy nhớ rằng các chỉ lệnh không tạo ra mã lệnh nào (mã máy) và chúng chỉ dùng bởi hợp ngữ, ngược lại đối với các lệnh là chúng được dịch ra mã máy (mã lênh) cho CPU thực hiện. Trong chương trình 2.1 các lệnh ORG và END là các chỉ lệnh (một số hợp ngữ của 8051 sử dụng dạng .ORG và .END). Hãy đọc quy định cụ thể của hợp ngữ ta sử dụng.

Trương chú giải luôn phải bắt đầu bằng dấu chấm phẩy (;). Các chú giải có thể bắt đầu ở đầu dòng hoặc giữa dòng. Hợp ngữ bỏ qua (làm ngơ) các chú giải nhưng chúng lại rất cần thiết đối với lập trình viên. Mặc dù các chú giải là tuỳ chọn, không bắt buộc nhưng ta nên dùng chúng để mô tả chương trình để giúp cho người khác đọc và hiểu chương trình dễ dàng hơn.

Lưu ý đến nhãn HERE trong trường nhãn của chương trình 2.1. Một nhãn bất kỳ tham chiếu đến một lệnh phải có dấu hai chấm (:) đứng ở sau. Trong câu lệnh nhảy ngắn SJMP thì 8051 được ra lệnh ở lại trong vòng lặp

này vô hạn. Nếu hệ thống của chúng ta có một chương trình giám sát thì takhông cần dòng lệnh này và nó có thể được xoá đi ra khỏi chương trình.

Chương trình 2.1: Tệp liệt kê

Sau khi chương trình được đốt vào trong ROM của thành viên họ 8051 như 8751 hoặc AT 8951 hoặc DS 5000 thì mã lệnh và toán hạng được đưa vào các vị trí nhớ ROM bắt đầu từ địa chỉ 0000 như bảng liệt kê dưới đây.

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

Bảng nội dung ROM của chương trình 2.1.

Địa chỉ	Mã lệnh
0000	7D
0001	25
0002	F7
0003	34
0004	74
0005	00
0006	2D
0007	2F
0008	24
0009	12
000A	80
000B	FE

Bảng liệt kê chỉ ra địa chỉ 0000 chứa mã 7D là mã lệnh để chuyển một giá trị vào thanh ghi R5 và địa chỉ 0001 chứa toán hạng (ở đây là giá trị 254) cần được chuyển vào R5. Do vậy, lệnh "MOV R5, #25H" có mã là "7D25" trong đó 7D là mã lệnh, còng 25 là toán hạng. Tương tự như vậy, mã máy "7F34" được đặt trong các ngăn nhớ 0002 và 0003 và biểu diễn mã lệnh và toán hạng đối với lệnh "MOV R7, #34H". Theo cách như vậy, mã máy "7400" được đặt tại địa chỉ 0004 và 0005 và biểu diễn mã lệnh và toán hạng đối với lệnh "MOV A, #0". Ngăn nhớ 0006 có mã 2D là mã đối với lệnh "ADD A, R5" và ngăn nhớ 0007 có nội dung 2F là mã lệnh cho "ADD A, R7". Mã lệnh đối với lệnh "ADD A, #12H" được đặt ở ngăn nhớ 0008 và toán hạng 12H được đặt ở ngăn nhớ 0009. Ngăn nhớ 000A có mã lệnh của lệnh SJMP và địa chỉ đích của nó được đặt ở ngăn nhớ 000B. Lý do vì sao địa chỉ đích là FE được giải thích ở chương 3.

2.4.4 Thực hiện một chương trình theo từng byte.

Giả sử rằng chương trình trên được đốt vào ROM của chíp 8051 hoặc(8751, AT 8951 hoặc DS 5000) thì dưới đây là mô tả hoạt động theo từng bước của 8051 khi nó được cấp nguồn.

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. Trong trường hợp của chương trình này là mã 7D để chuyển một toán hạng vào R5. Khi thực hiện mã lệnh CPU nạp giá trị 25 vào bộ đếm chương trình được tăng lên để chỉ đến 0002 (PC = 0002) có chứa mã lệnh 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. Ngăn nhớ 0004 chứa mã lệnh của lệnh "MOV A, #0". Lệnh này được thực hiện và bây giờ PC = 0006. Lưu ý rằng tất cả các lệnh trên đều là những lệnh 2 byte, nghĩa là mỗi lệnh chiếm hai ngăn nhớ.
4. Bây giờ PC = 0006 chỉ đến lệnh kế tiếp là "ADD A, R5". Đây là lệnh một byte, sau khi thực hiện lệnh này PC = 0007.
5. Ngăn nhớ 0007 chứa mã 2F là mã lệnh của "ADD A, R7". Đây cũng là lệnh một byte, khi thực hiện lệnh này PC được tăng lên 0008. Qúa trình này cứ tiếp tục cho đến khi tất cả moi lệnh đều được nạp và thực hiện. Thực tế mà bộ đếm chương trình chỉ đến lệnh kế tiếp cần được thực hiện giải thích tại sao một số bộ vi xử lý (đáng nói là × 86) gọi bộ đếm là con trỏ lệnh (Instruction Pointer).

2.4.5 Bản đồ nhớ ROM trong họ 8051.

Như ta đã thấy ở chương trước, một số thành viên họ 8051 chỉ có 4k byte bộ nhớ ROM trên chíp (ví dụ 8751, AT 8951) và một số khác như AT 8952 có 8k byte ROM, DS 5000-32 của Dallas Semiconductor có 32k byte ROM trên chíp. Dallas Semiconductor cũng có motọ 8051 với ROM trên chíp là 64k byte. Điểm cần nhớ là không có thành viên nào của họ 8051 có thể truy cập được hơn 64k byte mã lệnh vì bộ đếm chương trình của 8051 là 16 bit (dải địa chỉ từ 0000 đến FFFFH). Cần phải ghi nhớ là lệnh đầu tiên của ROM chương trình đều đặt ở 0000, còn lệnh cuối cùng phụ thuộc vào dung lượng ROM trên chíp của mỗi thành viên họ 8051. Trong số các thành viên họ 8051 thì 8751 và AT 8951 có 4k byte ROM trên chíp. Bộ nhớ ROM trên chíp này có các địa chỉ từ 0000 đến 0FFFH. Do vậy, ngăn nhớ đầu tiên có địa chỉ 0000 và ngăn nhớ cuối cùng có địa chỉa 0FFFH. Hãy xét ví dụ 2.1.

Ví dụ 2.1:

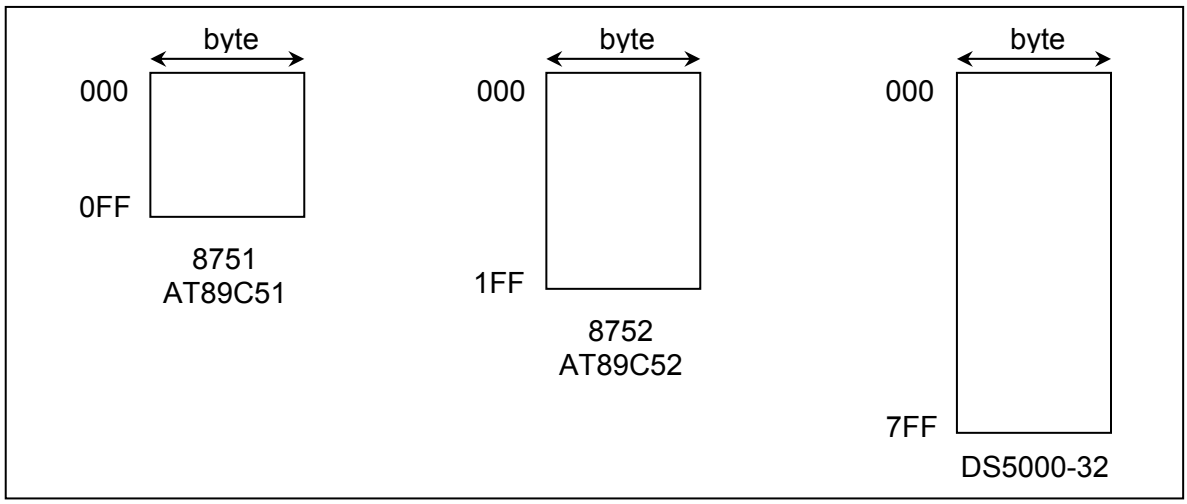
Tìm địa chỉ bộ nhớ ROM của mỗi thành viên họ 8051 sau đây.

a) AT 8951 (hoặc 8751) với 4k byte
b) DS 5000-32 với 32k byte

Lời giải:

a) Với 4k byte của không gian nhớ ROM trên chíp ta có 4096 byte bằng 1000H ở dạng Hex (4 × 1024 = 4096 hay 1000 ở dạng Hex). Bộ nhớ này được xắp xếp trong các ngăn nhớ từ 0000 đến 0FFFH. Lưu ý 0 luôn là ngăn nhớ đầu tiên.

b) Với 32k byte nhớ ta có 32.768 byte (32 × 1024). Chuyển đổi 32.768 về số Hex 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 chíp một số thành viên họ 8051.

2.5 Các kiểu dữ liệu và các chỉ lệnh.

2.5.1 Kiểu dữ liệu và các chỉ lệnh của 8051.

Bộ vi điều khiển chỉ có một kiểu dữ liệu, nó là 8 bit và độ dài mỗi thanh ghi cũng là 8 bit. Công việc của lập trình viên là phân chia dữ liệu lớn hơn 8 bit ra thành từng khúc 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. Các dữ liệu được sử dụng bởi 8051 có thể là số âm hoặc số dương và về xử lý các số có dấu được bàn ở chương 6.

2.5.2 Chỉ lệnh DB (định nghĩa byte).

Chỉ lệnh DB là một chỉ lệnh dữ liệu được sử dụng rộng rãi nhất trong hợp ngữ. Nó được dùng để định nghĩa dữ liệu 8 bit. Khi DB được dùng để định nghĩa byte dữ liệu thì các số có thể ở dạng thập phân, nhị phân, Hex hoặc ở dạng thức ASII. Đối với dữ liệu thập phân thì cần đặt chữ "D" sau số thập phân, đối với số nhị phân thì đặt chữ "B" và đối với dữ liệu dạng Hex thì cần đặt chữ "H". Bất kể ta sử dụng số ở dạng thức nào thì hợp ngữ đều chuyển đổi chúng về thành dạng Hex. Để báo dạng thức ở dạng mã ASCII thì chỉ cần đơn giản đặt nó vào dấu nháy đơn 'như thế này'. Hợp ngữ sẽ gán mã ASCII cho các số hoặc các ký tự một cách tự động. Chỉ lệnh DB chỉ là chỉ lệnh mà có thể được sử dụng để định nghĩa các chuỗi ASCII lớn hơn 2 ký tự. Do vậy, nó có thể được sử dụng cho tất cả mọi định nghĩa dữ liệu ASCII. Dưới đây là một số ví dụ về DB:

```
        ORG  500H
DATA1:  DB   20          ; Số thập phân (1C ở dạng Hex)
DATA2:  DB   00110101B   ; Số nhị phân (35 ở dạng Hex)
```

```
DATA3:   DB   39H                ; Số dạng Hex
         ORG  510H
DATA4:   DB   "2591"                    ; Các số ASCII
         ORG  518H
DATA5    DB   "My name is Joe"          ; Các ký tự ASCII
```

Các chuỗi ASCII có thể sử dụng dấu nháy đơn 'như thế này' hoặc nháy kép "như thế này". Dùng dấu phẩy kép sẽ hữu ích hơn đối với trường hợp dấu nháy đơn được dùng sở hữu cách như thế này "Nhà O' Leary". Chỉ lệnh DB cũng được dùng để cấp phát bộ nhớ theo từng đoạn kích thước một byte.

2.5.3 Các chỉ lệnh của hợp ngữ.

1. Chỉ lệnh ORG: Chỉ lệnh ORG được dùng để báo bắt đầu của địa chỉ. Số đi sau ORG có kể ở dạng Hex hoặc thập phân. Nếu số này có kèm chữ H đằng sau thì là ở dạng Hex và nếu không có chữ H ở sau là số thập phân và hợp ngữ sẽ chuyển nó thành số Hex. Một số hợp ngữ sử dụng dấu chấm đứng trước "ORG" thay cho "ORG". Hãy đọc kỹ về trình hợp ngữ ta sử dụng.
2. Chỉ lệnh EQU: Được dùng để định nghĩa một hằng số mà không chiếm ngăn nhớ nào. Chỉ lệnh EQU không dành chỗ cất cho dữ liệu nhưng nó gắn một giá trị hằng số với nhãn dữ liệu sao cho khi nhãn xuất hiện trong chương trình giá trị hằng số của nó sẽ được thay thế đối với nhãn. Dưới đây sử dụng EQU cho hằng số bộ đếm và sau đó hằng số được dùng để nạp thanh ghi RS.

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

   Khi thực hiện lện "MOV R3, #COUNT" thì thanh ghi R3 sẽ được nạp giá trị 25 (chú ý đến dấu #). Vậy ưu điểm của việc sử dụng EQU là gì? Giả sử có một hằng số (một giá trị cố định) được dùng trong nhiều chỗ khác nhau trong chương trình và lập trình viên muốn thay đổi giá trị của nó trong cả chương trình. Bằng việc sử dụng chỉ lệnh EQU ta có thể thay đổi một số lần và hợp ngữ sẽ thay đổi tất cả mọi lần xuất hiện của nó là tìm toàn bộ chương trình và gắng tìm mọi lần xuất hiện.
3. Chỉ lệnh END: Một lệnh quan trọng khác là chỉ lệnh END. Nó báo cho trình hợp ngữ kết thúc của tệp nguồn "asm" chỉ lệnh END là dòng cuối cùng của chương trình 8051 có nghĩa là trong mã nguồn thì mọi thứ sau chỉ lệnh END để bị trình hợp ngữ bỏ qua. Một số trình hợp ngữ sử dụng .END có dấu chấm đứng trước thay cho END.

2.5.4 Các quy định đối với nhãn trong hợp ngữ.

Bằng cách chọn các tên nhãn có nghĩa là một lập trình viên có thể làm cho chương trình dễ đọc và dễ bảo trì hơn, có một số quy định mà các tên nhãn

phải tuân theo. Thứ nhất là mỗi tên nhãn phải thống nhất, các tên được sử dụng làm nhãn trong hợp ngữ gồm các chữ cái viết hoa và viết thường, các số từ 0 đến 9 và các dấu đặc biệt như: dấu hỏi (?), dấu (≅), dấu gạch dưới (_), dấu đô là ($) và dấu chu kỳ (.). Ký tự đầu tiên của nhãn phải là một chữ cái. Hay nói cách khác là nó không thể là số Hex. Mỗi trình hợp ngữ có một số từ dự trữ là các từ gợi nhớ cho các lệnh mà không được dùng để làm nhãn trong chương trình. Ví dụ như "MOV" và "ADD". Bên cạnh các từ gợi nhớ còn có một số tự dự trữ khác, hãy kiểm tra bản liệt kê các từ dự phòng của hợp ngữ ta 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 như bit nhớ. Thanh ghi cờ trong 8051 được gọi là thanh ghi từ trạng thái chương trình PSW. Trong phần này và đưa ra một số ví dụ về cách thay đổi chúng.

2.6.1 Thanh ghi từ trạng thái chương trình PSW.

Thanh ghi PSW là thanh ghi 8 bit. Nó cũng còn được coi như là thanh ghi cờ. Mặc dù thanh ghi PSW rộng 8 bit nhưng chỉ có 6 bit được 8051 sử dụng. Hai bit chưa dùng là các cờ ch người dùng định nghĩa. Bốn trong số các cờ được gọi là các cờ có điều kiện, có nghĩa là chúng báo một số điều kiện do kết quả của một lệnh vừa được thực hiện. Bốn cờ này là cờ nhớ CY (carry), cờ AC (auxiliary cary), cờ chẵn lẻ P (parity) và cờ tràn OV (overflow).

Như nhìn thấy từ hình 2.4 thì các bit PSW.3 và PSW.4
được gán như RS0 và RS1 và chúng được sử dụng để thay đổi các thanh ghi băng. Chúng sẽ được giải thích ở phần kế sau. Các bit PSW.5 và PSW.1 là các bit cờ trạng thái công dụng chung và lập trình viên có thể sử dụng cho bất kỳ mục đích nào.

CY	AC	F0	RS1	RS0	OV	–	P

CY PSW.7 ; Cờ nhớ
AC PSW.6 ; Cờ
 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 bận
 PSW.1 ; Bit dành cho người dùng định nghĩa
P PSW.0 ; Cờ chẵn, lẻ. Thiết lập/ xoá bằng phần cứng mỗi chu kỳ lệnh báo tổng các số bit 1 trong thanh ghi A là chẵn/ lẻ.

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: **Các bit của thanh ghi PSW**

Dưới đây là giải thích ngắn gọn về 4 bit cờ của thanh ghi PSW.

1. Cờ nhớ CY: Cờ này được thiết lập mỗi khi có nhớ từ bit D7. Cờ này được tác động sau lệnh cộng hoặc trừ 8 bit. Nó cũng được thiết lập lên 1 hoặc xoá về 0 trực tiếp bằng lệnh "SETB C" và "CLR C" nghĩa là "thiết lập cờ nhớ" và "xoá cờ nhớ" tương ứng. Về các lệnh đánh địa chỉ theo bit được bàn kỹ ở chương 8.
2. Cờ AC: Cờ này báo có nhớ từ bit D3 sang D4 trong phép cộng ADD hoặc trừ SUB. Cờ này được dùng bởi các lệnh thực thi phép số học mã BCD (xem ở chương 6).
3. Cờ chẵn lẻ P: Cờ chẵn lẻ chỉ phản ánh số bit một trong thanh ghi A là chẵn hay lẻ. Nếu thanh ghi A chứa một số chẵn các bit một thì P = 0. Do vậy, P = 1 nếu A có một số lẻ các bit một.
4. Cờ chàn OV: Cờ này được thiết lập mỗi khi kết quả của một phép tính số có dấu quá lớn tạo ra bit bậc cao làm tràn bit dấu. Nhìn chung cờ nhớ được dùng để phát hiện lỗi trong các phép số học không dấu. Còn cờ tràn được dùng chỉ để phát hiện lỗi trong các phép số học có dấu và được bàn kỹ ở chương 6.

2.6.2 Lệnh ADD và PSW.

Bây giờ ta xét tác động của lệnh ADD lên các bit CY, AC và P của thanh ghi PSW. Một số ví dụ sẽ làm rõ trạng thái của chúng, mặc dù các bit cờ bị tác động bởi lệnh ADD là CY, P, AC và OV nhưng ta chỉ tập trung vào các cờ CY, AC và P, còn cờ OV sẽ được nói đến ở chương 6 vì nó liên quan đến phép tính số học số có dấu.

Các ví dụ 2.2 đến 2.4 sẽ phản ánh tác động của lệnh ADD lên các bit nói trên.

Bảng 2.1: Các lệnh tác động lên các bit cờ.

Ví dụ 2.2: Hãy trình bày trạng thái các bit cờ CY, AC và P sau lệnh cộng 38H với 2FH dưới đây:

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

Lời 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 trình bày trạng thái các cờ CY, AC và P sau phép cộng 9CH với 64H.

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

Lời 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 trình bày trạng thái các cờ CY, AC và P sau phép cộng 88H với 93H.

Lời 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 Các băng thanh ghi và ngăn xếp của 8051.

Bộ vi điều khiển 8051 có tất cả 128 byte RAM. Trong mục này ta bàn vệ phân bố của 128 byte RAM này và khảo sát công dụng của chúng như các thanh ghi và ngăn xếp.

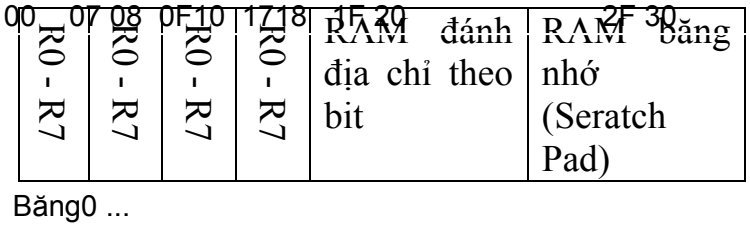
2.7.1 Phân bố không gian bộ nhớ RAM trong 8051.

Có 128 byte RAM trong 8051 (một số thành viên đang chú ý là 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:

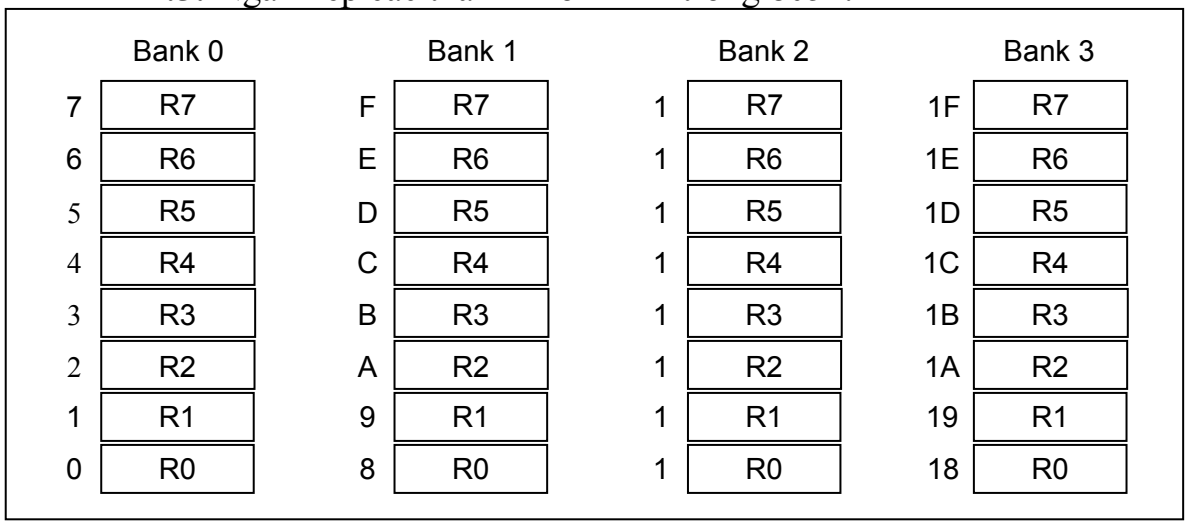
1. Tổng cộng 32 byte từ ngăn nhớ 00 đến 1FH được dành cho các thanh ghi và ngăn xếp.
2. Tổng cộng 16 byte từ ngăn nhớ 20H đến 2FH được dành cho bộ nhớ đọc/ ghi đánh địa chỉ được theo bit. Chương 8 sẽ bàn chi tiết về bộ nhớ và các lệnh đánh địa chỉ được theo bit.
3. Tổng cộng 80 byte từ ngăn nhớ 30H đến 7FH được dùng cho lưu đọc và ghi hay như vẫn thường gọi là bảng nháp (Serach pad). Những ngăn nhớ này (80 byte) của RAM được sử dụng rộng rãi cho mục đích lưu dữ liệu và tham số bởi các lập trình viên 8051. Chúng ta sẽ sử dụng chúng ở các chương sau để lưu dữ liệu nhận vào CPU qua các cổng vào-ra.

2.7.2 Các băng thanh ghi trong 8051.

Như đã nói ở trước, tổng cộng 32 byte RAM được dành riêng cho các băng thanh ghi và ngăn xếp. 32 byte này được chia ra thành 4 băng các thanh ghi trong đó mỗi băng có 8 thanh ghi từ R0 đến R7. Các ngăn nhớ RAM số 0, R1 là ngăn nhớ RAM số 1, R2 là ngăn nhớ RAM số 2 v.v... Băng thứ hai của các thanh ghi R0 đến R7 bắt đầu từ thanh nhớ RAM số 2 cho đến ngăn nhớ RAM số 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 các thanh ghi R0 đến R7 thứ tư.



Hình 2.5: Ngăn xếp các thanh nhớ RAM trong 8051.



Hình 2.6: Các băng thanh ghi của 8051 và địa chỉ của chúng.

Như ta có thể nhìn thấy từ hình 2.5 băng 1 sử dụng cùng không gian RAM như ngăn xếp. Đây là một vấn đề chính trong lập trình 8051. Chúng ta phải hoặc là không sử dụng băng 1 hoặc là phải đánh một không gian khác của RAM cho ngăn xếp.

Ví dụ 2.5:

Hãy phát biểu các nội dung của các ngăn nhớ RAM sau đoạn chương trình sau:

```
MOV R0, #99H        ; Nạp R0 giá trị 99H
MOV R1, #85H        ; Nạp R1 giá trị 85H
MOV R2, #3FH        ; Nạp R2 giá trị 3FH
MOV R7, #63H        ; Nạp R7giá trị 63H
MOV R5, #12H        ; Nạp R5 giá trị12H
```

Lời 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

2.6.3 Băng thanh ghi mặc định.

Nếu các ngăn nhớ 00 đến 1F được dành riêng cho bốn băng thanh ghi, vậy băng thanh ghi R0 đến R7 nào ta phải truy cập tới khi 8051 được cấp nguồn? Câu trả lời là các 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 R0, R1, R2, R3, R4, R5, R6 và R7 khi lập trình 8051. Nó dễ dàng hơn nhiều khi tham chiếu các ngăn nhớ RAM này ơví các tên R0, R1 v.v... hơn là số vị trí của các ngăn nhớ. Ví dụ 2.6 làm rõ khái niệm này.

Ví dụ 2.6:

Hãy viết lại chương trình ở ví dụ 2.5 sử dụng các địa chỉ RAM thay tên các thanh ghi.

Lời giải:

Đây được gọi là chế độ đánh địa chỉ trực tiếp và sử dụng địa chỉ các vị trí ngăn nhớ RAM đối với địa chỉ đích. Xem chi tiết ở chương 5 về chế độ đánh 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 trị12H
```

2.6.4 Chuyển mạch các băng thanh ghi như thế nào?

Như đã nói ở trên, băng thanh ghi 0 là mặc định khi 8051 được cấp nguồn. Chúng ta có thể chuyển mạch sang các băng thanh ghi khác bằng cách sử dụng bit D3 và D4 của thanh ghi PSW như chỉ ra theo bảng 2.2.

Bảng 2.2: Bit lựa chọn các 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

Bit D3 và D4 của thanh ghi PSW thường được tham chiếu như là PSW.3 và PSW.4 vì chúng có thể được truy cập bằng các lệnh đánh địa chỉ theo bit như SETB và CLR. Ví dụ "SETB PSW.3" sẽ thiết lập PSW.3 và chọn băng thanh ghi 1. Xem ví dụ 2.7 dưới đây.

Ví dụ 2.7:

Hãy phát biểu nội dung các ngăn nhớ RAM sau đoạn chương trình dưới đây:

```
SETB PSW.4          ; Chọn băng thanh ghi 4
MOV R0, #99H        ; Nạp thanh ghi R0 giá trị 99H
MOV R1, #85H        ; Nạp thanh ghi R1 giá trị 85H
MOV R2, #3FH        ; Nạp thanh ghi R2 giá trị 3FH
MOV R7, #63H        ; Nạp thanh ghi R7giá trị 63H
MOV R5, #12H        ; Nạp thanh ghi R5 giá trị12H
```

Lời 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, bằng lệnh như vậy 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

2.6.5 Ngăn xếp trong 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 dự liệu, có thể là địa cỉ CPU cần không gian lưu trữ này vì số các thanh ghi bị hạn chế.

2.6.6 Cách truy cập các ngăn xếp trong 8051.

Nếu ngăn xếp là một vùng của bộ nhớ RAM thì phải có các thanh ghi trong CPU chỉ đến nó. Thanh được dùng để chỉ đến ngăn xếp được gọi là thanh ghi con trỏ ngăn xếp SP (Stack Pointer). Con trỏ ngăn xếp trong 8051 chỉ rộng 8 bit có nghĩa là nó chỉ có thể có thể 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 cho ngăn xếp trong 8051. Việc lưu lại một thanh ghi PCU trong ngăn xếp được gọi là một lần cất vào PUSH và việc nạp nội dung của ngăn xếp trở lại thanh ghi CPU được gọi là lấy ra POP. Hay nói cách khác là một thanh ghi được cất vào ngăn xếp để lưu cất và được lấy ra từ ngăn xếp để dùng tiếp công việc của SP là rất nghiêm ngặt mỗi khi thao tác cất vào (PUSH) và lấy ra (POP) được thực thi. Để biết ngăn xếp làm việc như thế nào hãy xét các lệnh PUSH và POP dưới đây.

2.6.7 Cất thanh ghi vào ngăn xếp.

Trong 8051 thì con trỏ ngăn xếp chỉ đến ngăn nhớ sử dụng cuối cùng của ngăn xếp. Khi ta cất dữ liệu vào ngăn xếp thì con trỏ ngăn xếp SP được tăng lên 1. Lưu ý rằng điều này đối với các bộ vi xử lý khác nhau là khác nhau, đáng chú ý là các bộ vi xử lý × 86 là 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. Lưu ý là đối với mỗi byte của dữ liệu được cất vào ngăn xếp thì SP được tăng lên 1 lần. Cũng lưu ý rằng để cất các thanh ghi vào ngăn xếp ta phải 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 biểu diễn ngăn xếp và con trỏ ngăn xếp đối với đoạn chương trình sau đây. Giả thiết vùng ngăn xếp là mặc định.

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

Lời giải:

	Sau PUSH 6	Sau PUSP 1	Sau 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

2.6.8 Lấy nôi dung thanh ghi ra từ ngăn xếp.

Việc lấy nội dung ra từ ngăn xếp trở lai thanh ghi đã cho là qúa trình ngược với các nội dung thanh ghi vào ngăn xếp. Với mỗi lần lấy ra thì byte trên đỉnh ngăn xếp được sao chép vào thanh ghi được xác định bởi lệnh và con trỏ ngăn xếp được 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 hãy trình bày nội dung của các thanh ghi và SP sau khi thực hiện đoạn chương trình sau đây:

```
POP  3      ; Lấy ngăn xếp trở lại R3
POP  5      ; Lấy ngăn xếp trở lại R5
POP  2      ; Lấy ngăn xếp trở lại R2
```

Lời giải:

	Sau POP3	Sau POP 5	Sau POP 2
0B 54	0B	0B	0B
0A F9	0A F9	0A	0A
09 76	09 76	09 76	09
08 6C	08 6C	08 6C	08 6C
Bắt đầu SP = 0B	SP = 0A	SP = 09	SP = 08

2.6.9 Giới hạn trên của ngăn xếp.

Như đã nói ở trên, các ngăn nhớ 08 đến 1FH của RAM trong 8051 có thể được dùng làm ngăn nhớ 20H đến 2FH của RAM được dự phòng cho bộ nhớ đánh địa chỉ được theo bit và không thể dùng trước cho ngăn xếp. Nếu trong một chương trình đã cho ta cần ngăn xếp nhiều hơn 24 byte (08 đến 1FH = 24 byte) thì ta có thể đổi SP chỉ đến các ngăn nhớ 30 đến 7FH. Điều này được thực hiển bởi lệnh "MOV SP, #XX".

2.6.10 Lệnh gọi CALL và ngăn xếp.

Ngoài việc sử dụng ngăn xếp để lưu cất các thanh ghi thì CPU cũng sử dụng ngăn xếp để lưu cất tam thời địa chỉ của lệnh đứng ngay dưới lệnh CALL. Điều này chính là để PCU biết chỗ nào để quay trở về thực hiện tiếp các lệnh

sau khi chọn chương trình con. Chi tiết về lệnh gọi CALL được trình bỳ ở chương 3.

2.6.11 Xung đột ngăn xếp và băng thanh ghi số 1.

Như ta đa nói ở trên thì thanh ghi con trỏ ngăn xếp có thể chỉ đến vị trí RAM hiện thời dành cho ngăn xếp. Khi dữ liệu được lưu cất cào ngăn xếp thì SP được tăng lên và ngược lại khi dữ liệu được lấy ra từ ngăn xếp thì SP giảm xuống. Lý do là PS được tăng lên sau khi PUSH là phải biết lấy chắc chắn rằng ngăn xếp đang tăng lên đến vị trí ngăn nhớ 7FH của RAM từ địa chỉ thấp nhất đến địa chỉ cao nhất. Nếu con trỏ ngăn xếp đã được giảm sau các lệnh PUSH thì ta nên sử dụng các ngăn nhớ 7, 6, 5 v.v... của RAM thuộc các thanh ghi R7 đến R0 của băng 0, băng thanh ghi mặc định. Việc tăng này của con trỏ ngăn xếp đối với các lệnh PUSH cũng đảm bảo rằng ngăn xếp sẽ không với tới ngăn nhớ 0 của RAM (đáy của RAM) và do vậy sẽ nhảy ra khỏi không gian dành cho ngăn xếp. Tuy nhiên có vấn đề nảy sinh với thiết lập mặc định của ngăn xếp. Ví dụ SP = 07 khi 8051 được bật nguồn nên RAM và cũng thuộc về thanh ghi R0 củ băng thanh ghi số 1. Hay nói cách khác băng thanh ghi số 1 và ngăn xếp đang dùng chung một không gian của bộ nhớ RAM. Nếu chương trình đã cho cần sử dụng các băng thanh ghi số 1 và số 2 ta có thể đặt lại vùng nhớ RAM cho ngăn xếp. Ví dụ, ta có thể cấp vị trí ngăn nhớ 60H của RAM và cao hơn cho ngăn xếp trong ví dụ 2.10.

Ví dụ 2.10:

Biểu diễn ngăn xếp và con trỏ ngăn xếp đối với các lệnh sau:

```
    MOV SP, #5FH          ; Đặt ngăn nhớ từ 60H của RAM cho ngăn xếp
    MOV R2, #25H
    MOV R1, #12H
    MOV R4, #0F3H
    PUSH2
    PUSH1
  PUSH 4
```

Lời giải:

	Sau PUSH 2	Sau PUSP 3	Sau PUSH 4
63	63	63	63
62	62	62	62 F3
61	61	61 12	61 12
60	60 25	60 25	60 25
Bắt đầuSP=5F	SP = 60	SP = 61	SP = 62

CHƯƠNG 3
Các lệnh nhảy, vòng lặp và lệnh gọi

Trong một chuỗi lệnh cần thực hiện thường có nhu cần cần chuyển điều khiển chương trình đến một vị trí khác. Có nhiều lệnh để thực hiện điều này trong 8051, ở chương này ta sẽ tìm hiểu các lệnh chuyển điều khiển có trong hợp ngữ của 8051 như các lệnh sử dụng cho vòng lặp, các lệnh nhảy có và không có điều khiển, lệnh gọi và cuối cùng là mô tả về một chương trình con giữ chậm thời gian.

3.1 Vòng lặp và các lệnh nhảy.

3.1.1 Tạo vòng lặp trong 8051.

Qúa trình lặp lại một chuỗi các lệnh với một số lần nhất định được gọi là vòng lặp. Vòng lặp là một trong những hoạt động được sử dụng rộng rãi nhất mà bất kỳ bộ vi sử lý nào đều thực hiện. Trong 8051 thì hoạt động vòng lặp được thực hiện bởi lệnh "DJNZ thanh ghi, nhãn". Trong lệnh này thanh ghi được giảm xuống, nếu nó không bằng không thì nó nhảy đến địa chỉ đích được tham chiếu bởi nhãn. Trước khi bắt đầu vòng lặp thì thanh ghi được nạp với bộ đếm cho số lần lặp lại. Lưu ý rằng, trong lệnh này việc giảm thanh ghi và quyết định để nhảy được kết hợp vào trong một lệnh đơn.

Ví dụ 3.1:

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

Lời giải:

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

Trong chương trình trên đây thanh ghi R2 được sử dụng như là bộ đếm. Bộ đếm lúc đầu được đặt bằng 10. Mỗi lần lặp lại lệnh DJNZ giảm R2 không bằng 0 thì nó nhảy đến địa chỉ đích gắn với nhãn "AGAIN". Hoạt động lặp lại này tiếp tục cho đến khi R2 trở về không. Sau khi R2 = 0 nó thoát khỏi vòng lặp và thực hiện đứng ngay dưới nó trong trường hợp này là lệnh "MOV R5, A".

Lưu ý rằng trong lệnh DJNZ thì các thanh ghi có thể là bất kỳ thanh ghi nào trong các thanh ghi R0 - R7. Bộ đếm cũng có thể là một ngăn nhớ trong RAM như ta 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?

Lời giải:

Vì thanh ghi R2 chứa số đếm và nó là thanh ghi 8 bit nên nó 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.

3.2.1 Vòng lặp bền trong một vòng lặp.

Như trình bày ở ví dụ 3.2 số đếm cực đại là 256. Vậy điều gì xảy ra nếu ta muốn lặp một hành động nhiều hơn 256 lần? Để làm điều đó thì ta sử dụng một vòng lặp bên trong một vòng lặp được gọi là vòng lặp lồng (Nested Loop). Trong một vòng lặp lồng ta sử dụng 2 thanh ghi để giữ số đếm. Xét ví dụ 3.3 dưới đây.

Ví dụ 3.3:

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

Lời giải:

Vì 700 lớn hơn 256 (là số cực đại mà một thanh ghi vó thể chứa được) nên ta phải dùng hai thanh ghi để chứa số đếm. Đoạn mã 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. Trong lệnh "DJNZ R2, AGAIN" thì mỗi khi R2 = 0 nó đi thẳng xuống và lệnh "DJNZ R3, NEXT" được thực hiện. Lệnh này ép CPU nạp R2 với số đếm 70 và vòng lặp trong khi bắt đầu lại quá trình này tiếp tục cho đến khi R3 trở về không và vòng lặp ngoài kết thúc.

3.1.3 Các lệnh nhảy có điều kiện.

Các lệnh nhảy có điều kiện đối với 8051 được tổng hợp trong bảng 3.1. Các chi tiết về mỗi lệnh được cho trong phụ lục AppendixA. Trong bảng 3.1 lưu ý rằng một số lệnh như JZ (nhảy nếu A = 0) và JC (nhảy nếu có nhớ) chỉ nhảy nếu một điều kiện nhất định được thoả mãn. Kế tiếp ta xét một số lệnh nhảy có điều kiện với các **Ví dụ minh hoạ sau.**

a- Lệnh JZ (nhảy nếu A = 0). Trong lệnh này nội dung của thanh ghi A được kiểm tra. Nếu nó bằng không thì nó nhảy đến địa chỉ đích. Ví dụ xét đoạn mã 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 ...
```

Trong chương trình này nếu R0 hoặc R1 có giá trị bằng 0 thì nó nhảy đến địa chỉ có nhãn OVER. Lưu ý rằng lệnh JZ chỉ có thể được sử dụng đối với thanh ghi A. Nó chỉ có thể kiểm tra xem thanh ghi A có bằng không không và nó không áp dụng cho bất kỳ thanh ghi nào khác. Quan trọng hơn là ta không phải thực hiện một lệnh số học nào như đếm giảm để sử dụng lệnh JNZ như ở ví dụ 3.4 dưới đây.

Ví dụ 3.4:

Viết một chương trình để xác định xem R5 có chứa giá trị 0 không? Nếu bằng thì nạp nó cho giá trị 55H.

Lời giải:

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

b- Lệnh JNC (nhảy nếu không có nhớ, cờ CY = 0).

Trong lệnh này thì bit cờ nhớ trong thanh ghi cờ PSW được dùng để thực hiện quyết định nhảy. Khi thực hiện lệnh "JNC nhãn" thì bộ xử lý kiểm tra cờ nhớ xem nó có được bật không (CY = 1). Nếu nó không bật thì CPU bắt đầu nạp và thực hiện các lệnh từ địa chỉ của nhãn. Nếu cờ CY = 1 thì nó sẽ không nhảy và thực hiện lệnh kế tiếp dưới JNC.

Cần phải lưu ý rằng cũng có lệnh "JC nhãn". Trong lệnh JC thì nếu CY = 1 nó nhảy đến địa chỉ đích là nhãn. Ta sẽ xét các ví dụ về các lệnh này trong các ứng dụng ở các chương sau.

Ngoài ra 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.

Bảng 3.1: Các lệnh nhảy có điều kiện.

Lệnh	**Hoạt động**
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 3.5:

Hãy tìm tổng của các giá trị 79H, F5H và E2H. Đặt vào trong các thanh ghi R0 (byte thấp) và R5 (byte cao).

Lời giải:

```
            MOV A, #0       ; Xoá thanh ghi A = 0
            MOV R5, A       ; Xoá R5
            ADD A, #79H         ; Cộng 79H vào A (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    ; Cộng F5H vào A (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    ; Cộng E2H vào A (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
```

c- Tất cả các lệnh nhảy có điều kiện đều là những phép nhảy ngắn.

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

3.1.4 Các lệnh nhảy không điều kiện.

Lệnh nhảy không điều kiện là một phép nhảy trong đó điều khiển được truyền không điều kiện đến địa chỉ đích. Trong 8051 có hai lệnh nhảy không điều kiện đó là: LJMP - nhảy xa và SJMP - nhảy gần.

a- Nhảy xa LJMP:

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

Hãy nhớ rằng, mặc dù bộ đếm chương trình trong 8051 là 16 bit, do vậy cho không gian địa chỉ là 64k byte, nhưng bộ nhớ chương trình ROM trên chíp lớn như vậy. 8051 đầu tiên chỉ có 4k byte ROM trên chíp cho không gian chương trình, do vậy mỗi byte đều rất quý giá. Vì lý do đó mà có cả lệnh nhảy gần SJMP chỉ có 2 byte so với lệnh nhảy xa LZ0MP dài 3 byte. Điều này có thể tiết kiệm được một số byte bộ nhớ trong rất nhiều ứng dụng mà không gian bộ nhớ có hạn hẹp.

b- Lệnh nhảy gồm SJMP.

Trong 2 byte này thì byte đầu tiên là mã lệnh và byte thứ hai là chỉ tương đối của địa chỉ đích. Đích chỉ tương đối trong phạm vi 00 - FFH được chia thành các lệnh nhảy tới và nhảy lùi: Nghĩa là -128 đến +127 byte của bộ nhớ tương đối so với địa chỉ hiện thời của bộ đếm chương trình. Nếu là lệnh nhảy tới 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 thì nó có thể nằm trong khoảng -128 byte từ giá trị hiện hành của PC.

3.1.5 Tính toán địa chỉ lệnh nhảy gần.

Ngoài lệnh nhảy gần SJMP thì tất cả mọi lệnh nhảy có điều kiện như JNC, JZ và DJNZ đều là các lệnh nhảy gần bởi một thực tế là chúng đều lệnh 2 byte. Trong những lệnh này thì byte thứ nhất đều là mã lệnh, còn byte thứ hai là địa chỉ tương đối. Địa chỉ đích là tương đối so với giá trị của bộ đếm chương trình. Để tính toán địa chỉ đích byte thứ hai được cộng vào thanh ghi PC của lệnh đứng ngay sau lệnh nhảy. Để hiểu điều này hãy xét ví dụ 3.6 dưới đây.

Ví dụ 3.6:

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

```
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
      SHERE
18    0017                          END
```

Lời giải:

Trước hết lưu ý rằng các lệnh JZ và JNC đều là lệnh nhảy về trước. Địa chỉ đích đối với lệnh nhảy về trước được tính toán bằng cách cộng giá trị PC của lệnh đi ngay sau đó vào byte thứ hai của lệnh nhảy gần được gọi là địa chỉ tương đối. Ở dòng 04 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 so với địa chỉ của lệnh kế tiếp là: "INC R0" và đó là 0006. Bằng việc cộng 0006 vào 3 thì địa chỉ đích của nhãn NEXT là 0009 được tạo ra. Bằng cách tương tự như vậy đối với dòng 9 thì lệnh "JNC OVER" có mã lệnh và toán hạng là 50 và 05 trong đó 50 là mã lệnh và 05 là địa chỉ tương đối. Do vậy, 05 được cộng vào OD là địa chỉ của lệnh "CLA A" đứng ngay sau lệnh "JNC OVER" và cho giá trị 12H chính là địa chỉ của nhãn OVER.

Ví dụ 3.7:

Hãy kiểm tra tính toán địa chỉ của các lệnh nhảy lùi trong ví dụ 3.6.

Lời giải:

Trong danh sách liệt kê chương trình đó thì lệnh "JNC AGAIN" có mã lệnh là 50 và địa chỉ tương đối là F2H. Khi địa chỉ tương đối của F2H được cộng vào 15H là địa chỉ của lệnh đứng dưới lệnh nhảy ta có 15H + F2H = 07 (và phần nhớ được bỏ đi). Để ý rằng 07 là địa chỉ nhãn AGAIN. Và hãy cũng xét lệnh "SJMP HERE" có mã lệnh 80 và địa chỉ tương đối FE giá trị PC của lệnh kế tiếp là 0017H được cộng vào địa chỉ tương đối FEH ta nhận được 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.

3.1.6 Tính toán địa chỉ đích nhảy lùi.

Trong khi ở trường hợp nhảy tới thì giá trị thay thế là một số dương trong khoảng từ 0 đến 127 (00 đến 7F ở dạng Hex) thì đối với lệnh nhảy lùi giá trị thay thế là một số âm nằm trong khoảng từ 0 đến -128 như được giải thích ở ví dụ 3.7.

Cần phải nhấn mạnh rằng, bất luận SJMP nhảy tới hay nhảy lùi thì đối với một lệnh nhảy bất kỳ địa chỉ của địa chỉ đích không bao giờ có thể lớn hơn 0 -128 đến +127 byte so với địa chỉ gắn liền với lệnh đứng ngay sau lệnh SJMP. Nếu có một sự nỗ lực nào vi phạm luật này thì hợp ngữ sẽ tạo ra một lỗi báo rằng lệnh nhảy ngoài phạm vi.

3.2 Các lệnh gọi CALL.

Một lệnh chuyển điều khiển khác là lệnh CALL được dùng để gọi một chương trình con. Các chương trình con thường được sử dụng để thực thi các

công việc cần phải được thực hiện thường xuyên. Điều này làm cho chương trình trở nên có cấu trúc hơn ngoài việc tiết kiệm được thêm không gian bộ nhớ. Trong 8051 có 2 lệnh để gọi đó là: Gọi xa LCALL và gọi tuyệt đối ACALL mà quyết định sử dụng lệnh nào đó phụ thuộc vào địa chỉ đích.

3.2.1 Lệnh gọi xa LCALL.

Trong lệnh 3 byte này thì byte đầu tiên là mã lệnh, còn hai byte sau được dùng cho địa chỉ của chương trình con đích. Do vậy LCALL có thể được dùng để gọi các chương trình con ở bất kỳ vị trí nào trong phạm vi 64k byte, không gian địa chỉ của 8051. Để đảm bảo rằng sau khi thực hiện một chương trình được gọi để 8051 biết được chỗ quay trở về thì nó tự động cất vào ngăn xếp địa chỉ của lệnh đứng ngay sau lệnh gọi LCALL. Khi một chương trình con được gọi, điều khiển được chuyển đến chương trình con đó và bộ xử lý cất bộ đếm chương trình PC vào ngăn xếp và bắt đầu nạp lệnh vào vị trí mới. Sau khi kết thúc thực hiện chương trình con thì lệnh trở về RET chuyển điều khiển về cho nguồn gọi. Mỗi chương trình con cần lệnh RET như là lệnh cuối cùng (xem ví dụ 3.8).

Các điểm sau đây cần phải được lưu ý từ ví dụ 3.8.

1. Lưu ý đến chương trình con DELAY khi thực hiện lệnh "LCALL DELAY" đầu tiên thì địa chỉ của lệnh ngay kế nó là "MOV A, #0AAH" được đẩy vào ngăn xếp và 8051 bắt đầu thực hiện các lệnh ở địa chỉ 300H.
2. Trong chương trình con DELAY, lúc đầu bộ đếm R5 được đặt về giá trị 255 (R5 = FFH). Do vậy, vòng lặp được lặp lại 256 lần. Khi R5 trở về 0 điều khiển rơi xuống lệnh quay trở về RET mà nó kéo địa chỉ từ ngăn xếp vào bộ đếm chương trình và tiếp tục thực hiện lệnh sau lệnh gọi CALL.

Ví dụ 3.8:

Hãy viết một chương trình để chốt tất cả các bit của cổng P1 bằng cách gửi đến nó giá trị 55H và AAH liên tục. Hãy đặt một độ trễ thời gian giữa mỗi lần xuất dữ liệu tới cổng P1. Chương trình này sẽ được sử dụng để kiểm tra các cổng của 8051 trong chương tiếp theo.

Lời giải:

```
        ORG 0000
BACK:   MOV A, #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 chương trình con trễ thời gian ở địa chỉ 300H
DELAY:  MOV R5, #00H    ; Nạp bộ đếm R5 = 255H (hay FFH)
AGAIN:  DJNZR5, AGAIN   ; Tiếp tục cho đến khi R5 về không
        RET             ; Trả điều khiển về nguồn gọi (khi R5 = 0)
        END             ; Kết thúc tệp tin của hợp ngữ
```

Lượng thời gian trễ trong ví dụ 8.3 phục 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 độ trễ bằng cách sử dụng vòng lặp lồng như chỉ ra dưới đây.

```
DELAY:                  ; Vòng lặp lồng giữ chậm
        MOV R4, #255    ; Nạp R4 = 255 (FFH dạng hex)
NEXT:   MOV R5, #255    ; Nạp R5 = 255 (FFH dạng hex)
AGAIN:  DJNZR5, AGAIN   ; Lặp lại cho đến khi R5 = 0
        DJNZR4, NEXT    ; Giảm R4
                        ;Tiếp tục nạp R5 cho đến khi R4 = 0
        RET             ; Trở về (khi R4 = 0)
```

3.2.2 Lệnh gọi CALL và vai trò của ngăn xếp.

Ngăn xếp và con trỏ ngăn xếp ta sẽ nghiên cứu ở chương cuối. Để hiểu được tầm quan trọng của ngăn xếp trong các bộ vi điều khiển bây giờ khảo sát nội dung của ngăn xếp và con trỏ ngăn xếp đối với ví dụ 8.3. Điều này được trình bày ở ví dụ 3.9 dưới đây.

Ví dụ 3.9:

Hãy phân tích nội dung của ngăn xếp sau khi thực hiện lệnh LCALL đầu tiên dưới đây.

```
001  0000                OR6
002  0000 7455     BACK:  MOV A, #55H     ; Nạp A với giá trị 55H
003  0002 F590            MOV P1, A      ; Gửi 55H tới cổng P1
004  0004 120300          LCALL DELAY    ; Tạo trễ thời gian
005  0007 74AA            MOV A, #0AAH   ; Nạp A với giá trị AAH
006  0009 F590            MOV P1, A      ; Gửi AAH tới cổng P1
007  000B 120300          LCALL DELAY    ; Tạo trễ thời gian
008  000E 80F0            SJMP BACK      ; Tiếp tục thực hiện
009  0010
```

```
010  0010                    ; ..... Đây là chương trình con giữ chậm
011  0300                    MOV 300H
012  0300              DELAY:
013  0300  7DFF               MOV R5, #FFH    ; Nạp R5 = 255
014  0302  DDFE               AGAIN:DJNZ      R5, AGAIN ; Dừng ở đây
015  0304  22                 RET             ; Trở về nguồn gọi
016  0305                    END             ; Kết thúc nạp tin hợp ngữ
```

Lời giải:

Khi lệnh LCALL đầu tiên được thực hiện thì địa chỉ của lệnh "MOV A, #0AAH" được cất vào ngăn xếp. Lưu ý rằng byte thấp vào trước và byte cao vào sau. Lệnh cuối cùng của chương trình con được gọi phải là lệnh trở về RET để chuyển CPU kéo (POP) các byte trên đỉnh của ngăn xếp vào bộ đếm chương trình PC và tiếp tục thực hiện lệnh tại địa chỉ 07. Sơ đồ bên chỉ ra khung của ngăn xếp sau lần gọi LCALL đầu tiên.

	0A	
09		00
08		07
SP	=	09

3.2.3 Sử dụng lệnh PUSH và POP trong các chương trình con.

Khi gọi một chương trình con thì ngăn xếp phải bám được vị trí mà CPU cần trở về. Sau khi kết thúc chương trình con vì lý do này chúng ta phải cẩn thận mỗi khi thao tác với các nội dung của ngăn xếp. Nguyên tắc là số lần đẩy vào (PUSH) và kéo ra (POP) luôn phải phù hợp trong bất kỳ chương trình con được gọi vào. Hay nói cách khác đối với mỗi lệnh PUSH thì phải có một lệnh POP. Xem ví dụ 3.10.

3.2.4 Gọi các chương trình con.

Trong lập trình hợp ngữ thường có một chương trình chính và rất nhiều chương trình con mà chúng được gọi từ chương trình chính. Điều này cho phép ta tạo mới chương trình con trong một mô-đun riêng biệt. Mỗi mô-đun có thể được kiểm tra tách biệt và sau đó được kết hợp với nhau cùng với chương trình chính. Quan trọng hơn là trong một chương trình lớn thì các mô-đun có thể được phân cho các lập trình viên khác nhau nhằm rút ngắn thời gian phát triển.

Ví dụ 3.10:

Phân tích ngăn xếp đối với lệnh LCALL đầu tiên trong đoạn mã.

```
01  0000                ORG     0
```

```
02  0000  7455      BACK:       MOV     A, #55H         ; Nạp A với giá trị 55H
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 giữ chậm thời gian
07  000B  74AA                  MOV     A, #0AAH  ; Nạp A với AAH
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 chương trình con 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,  #0FFH      ; 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
```

Lời giải:

Trước hết lưu ý rằng đối với các lệnh PUSH và POP ta phải xác định địa chỉ trực tiếp của thanh ghi được đẩy vào, kéo ra từ ngăn xếp. Dưới đây là sơ đồ khung của ngăn xếp.

Sau lệnh LCALL thứ nhất	Sau lệnh PUSH 4	Sau lệnh POSH 5
0B	0B	0B 67
		R5
0A	0A 99	0A 09
	R4	R4
09 00	09 00	09 00
PCH	PCH	PCL
08 0B	0B 0B	08 0B
PCL	PCL	PCL

Cần phải nhấn mạnh rằng trong việc sử dụng LCALL thì địa chỉ đích của các chương trình con có thể ở đâu đó trong phạm vi 64k byte không gian bộ nhớ của 8051. Điều này không áp dụng cho tất cả mọi lệnh gọi CALL chẳng hạn như đối với ACALL dưới đây:

```
; MAIN program calling subroutines
                ORG             0
MAIN:           LCALL           SUBR-1
                LCALL           SUBR-2
                LCALL           SUBR-3

HERE:                   SJMP            MAIN
;----------------- end of MAIN
;
SUBR-1I         ...
                ...
                RET
; ----------------- end of subroutinel  1
; SUBR-1I       ...
                ...
                RET
; ----------------- end of subroutinel  2
 ; SUBR-1I      ...
                ...
                RET
; ----------------- end of subroutinel  3
                END                     ; end of the asm        file
```

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

3.2.5 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 2k byte địa chỉ vì chỉ có 11bit của 2 byte được sử dụng cho địa chỉ. Không có sự khác biệt nào giữa ACALL và LCALL trong khái niệm cất bộ đếm chương trình vào ngăn xếp hay trong chức năng của lệnh trở về RET. Sự khác nhau duy nhất là địa chỉ đích của lệnh LCALL có thể nằm bất cứ đâu trong phạm vi 64k byte không gian địa chỉ của 8051, còn trong khi đó địa chỉ của lệnh ACALL phải nằm trong khoảng 2 byte. Trong nhiều biến thế của 8051 do các hãng cung cấp thì ROM trên chíp chỉ có 1k byte.. Trong những trường hợp như vậy thì việc sử dụng ACALL thay cho LCALL có thể tiết kiệm được một số byte bộ nhớ của không gian ROM chương trình.

Ví dụ 3.11:

Một nhà phát triển sử dụng chíp vi điều khiển Atmel AT89C1051 cho một sản phẩm. Chíp này chỉ có 1k byte ROM Flash trên chíp. Hỏi trong khi lệnh LCALL và ACALL thì lệnh nào hữu ích nhất trong lập trình cho chíp này.

Lời giải:

Lệnh ACALL là hữu ích hơn vì nó là lệnh 2 byte. Nó tiết kiệm một byte mỗi lần gọi được sử dụng.

Tất nhiên, việc sử dụng các lệnh gọn nhẹ, chúng ta có thể lập trình hiệu quả bằng cách có một hiểu biết chi tiết về tất cả các lệnh được hỗ trợ bởi bộ vi xử lý đã cho và sử dụng chúng một cách khôn ngoan. Xét ví dụ 3.12 dưới đây.

Ví dụ 3.12:

Hãy viết lại chương trình ở ví dụ 3.8 một cách hiệu quả mà bạn có thể:

Lời giải:

```
        ORG     0
        MOV     A, #55H         ; Nạp Avới giá trị 55H
BACK:   MOV     P1, A           ; Xuất giá trị trong A ra cổng P1
        ACALL   DELAY           ; Giữ chậm
        CPL     A               ; Bù thành ghi A
        SJMP    BACK            ; Tiếp tục thực hiện vô hạn
; -------- Đây là chương trình con giữ chậm DELAY
DELAY:
        MOV     R5, #0FFH       ; Nạp R5 = 255 (hay FFH) làm cho
bộ đếm
AGAIN:  DJNZ    R5, AGAIN ; Dừng ở đây cho đến khi R5  = 0
        RET                     ; Trở về
        END                     ; Kết thúc
```

3.3 Tạo và tính toán thời gian giữ chậm.

3.3.1 Chu kỳ máy:

Đối với CPU để thực hiện một lệnh thì mất một chu kỳ đồng hồ này được coi như các chu kỳ máy. Phụ lục AppendixA.2 cung cấp danh sách liệt kê các lệnh 8051 và các chu kỳ máy của chúng. Để tính toán một độ trễ thời gian, ta sử dụng danh sách liệt kê này. Trong họ 8051 thì độ dài của chu kỳ máy phụ thuộc vào tần số của bộ dao động thạch anh được nối vào hệ thống 8051. Bộ dao động thạch anh cùng với mạch điện trên chip cung cấp xung đồng hồ cho CPU của 8051 (xem chương 4). Tần số của tinh thể thạch anh được nối tới họ 8051 dao động trong khoảng 4MHz đến 30 MHz phụ thuộc vào tốc độ chíp và nhà sản xuất. Thường xuyên nhất là bộ dao động thạch anh tần số 11.0592MHz được sử dụng để làm cho hệ 8051 tương thích với cổng nối tiếp của PC IBM (xem chương 10). Trong 8051, một chu kỳ máy kéo dài 12 chu kỳ dao động. Do vậy, để tính toán chu kỳ máy ta lấy 1/12 của tần số tinh thể thạch anh, sau đó lấy giá trị nghịch đảo như chỉ ra trong ví dụ 3.13.

Ví dụ 3.13:

Đoạn mã dưới đây trình bày tần số thạch anh cho 3 hệ thống dựa trên 8051 khác nhau. Hãy tìm chu kỳ máy của mỗi trường hợp: a) 11.0592MHz b) 16MHz và c) 20MHz.

Lời giải:

a) 11.0592/12 = 921.6kHz; Chu kỳ máy là 1/921.6kHz = 1.085μs (micro giây)
b) 16MHz/12 = 1.333MHz; Chu kỳ máy MC = 1/1.333MHz = 0.75μs
c) 20MHz/12 = 1.66MHz $\Rightarrow$ MC = 1/1.66MHz = 0.60μs

Ví dụ 3.14:

Đối với một hệ thống 8051 có 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

Lời giải:

Chu kỳ máy cho hệ thống 8051 có tần số đồng hồ là 11.0592MHz Là 1.085μs như đã tính ở ví dụ 3.13. Bảng A-1 trong phụ lục Appendix A trình bày số chu kỳ máy đối với các lệnh trên. Vậy ta có:

Lệnh	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

3.3.2 Tính toán độ trễ.

Như đã trình bày ở trên đây, một chương trình con 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 tìm kích thước của thời gian giữ chậm trong chương trình sau, nếu tần số giao động thach anh là 11.0592MHz.

```
        MOV A, #55H
AGAIN:  MOV     P1, A
        ACALL   DELAY
        CPL  A
        SJMP    AGAIN
; --------  Time delay
DELAY:  MOV     R3, #200
HERE  :  DJNZR3, HERE
        RET
```

Lời giải:

Từ bảng A-1 của phụ lục Appendix A ta có các chu kỳ máy sao cho các lệnh của chương trình con giữ chậm là:

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

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

Thông thường ta tính thời gian giữ chậm dựa trên các lệnh bên trong vòng lặp và bỏ qua các chu kỳ đồng hồ liên quan với các lệnh ở ngoài vòng lặp.

Trong ví dụ 3.15 giá trị lớn nhất mà R3 có thể chứa là 255, do vậy một cách tăng độ trễ là sử dụng lệnh NOP (không làm gì) trong vòng lặp để tiêu

tốn thời gian một cách đơn giản. Điều này được chỉ ra trong ví dụ 3.16 dưới đây.

Ví dụ 3.16:

Hãy tìm độ trễ thời gian cho 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
```

Lời 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.

3.3.3 Độ trễ thời gian của vòng lặp trong vòng lặp.

Một cách khác để nhận được giá trị từ độ trễ lớn là sử dụng một vòng lặp bên trong vòng lặp và cũng được gọi là vòng lặp lồng nhau. Xem ví dụ 3.17 dưới đây.

Ví dụ 3.17:

Đối với một chu kỳ máy 1.085μs hãy tính thời gian giữ chậm trong chương trình con sau:

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

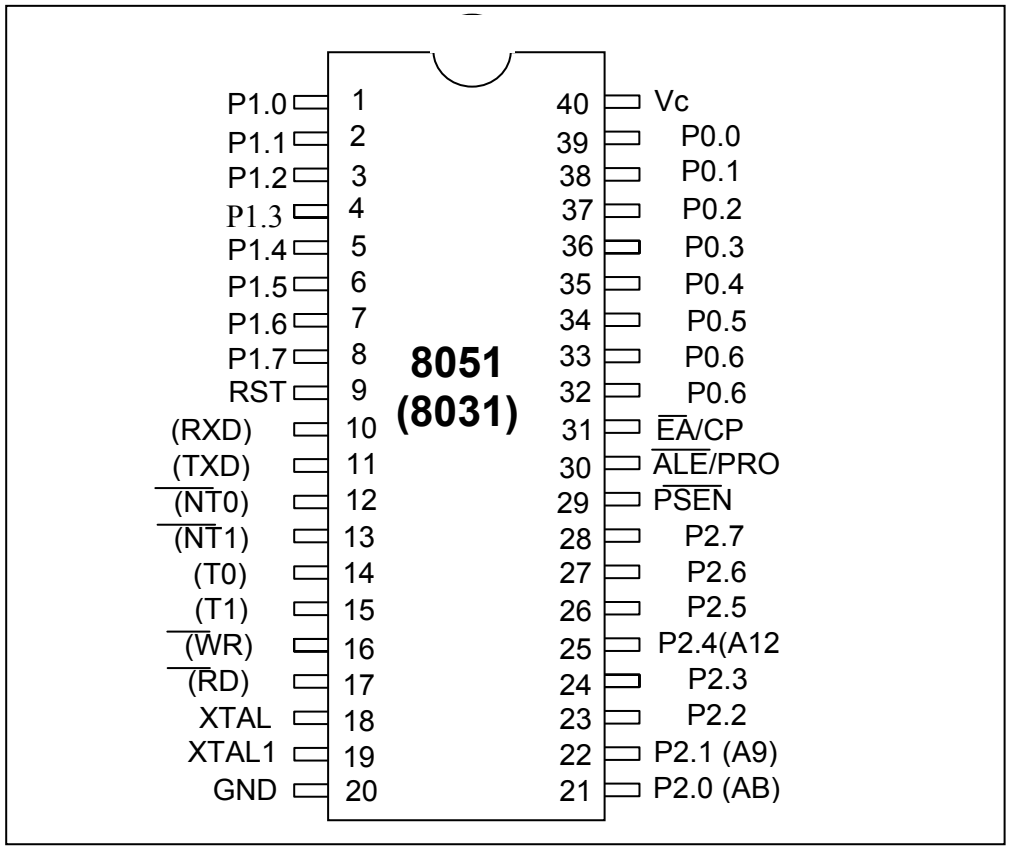
Lời giải:

Đối với vòng lặp HERE ta có (4 × 250) × 1.085μs = 1085μs. Vòng lặp AGAIN lặp vòng lặp HERE 200 lần, do vậy thời gian trễ là 200 × 1085μs 217000μs, nên ta không tính tổng phí. Tuy nhiên, các lệnh "MOV R3, #250" và "DJNZ R2, AGAIN" ở đầu và cuối vòng lặp AGAIN cộng (3 × 200 × 1.085μs) = 651μs vào thời gian trễ và kết quả ta có 217000 + 651 = 217651μs = 217.651 miligiây cho tổng thời gian trễ liên quan đến chương trình con giữ chậm DELAY nói trên. Lưu ý rằng, trong trường hợp vòng lặp lồng nhau cũng như trong mọi vòng lặp giữ chậm khác thời gian xấp xỉ gần dúng vì ta bỏ qua các lệnh đầu và cuối trong chương trình con.

CHƯƠNG 4
Lập trình cho cổng vào - ra I/0

4.1 Mô tả chân của 8051.

Mặc dù các thành viên của họ 8051 (ví dụ 8751, 89C51, DS5000) đều có các kiểu đóng vỏ khác nhau, chẳng hạn như hai hàng chân DIP (Dual In-Line Pakage) dạng vỏ dẹt vuông QFP (Quad Flat Pakage) và dạng chíp không có chân đỡ LLC (Leadless Chip Carrier) thì chúng đều có 40 chân cho các chức năng khác nhau như vào ra I/0, đọc $\overline{RD}$, ghi $\overline{WR}$, địa chỉ, dữ liệu và ngắt. Cần phải lưu ý rằng một số hãng cung cấp một 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 sử dụng chíp đóng vỏ 40 chân với hai hàng chân DIP nên ta chỉ tập chung mô tả phiên bản này.



Hình 4.1: Sơ đồ bố trí chân của 8051.

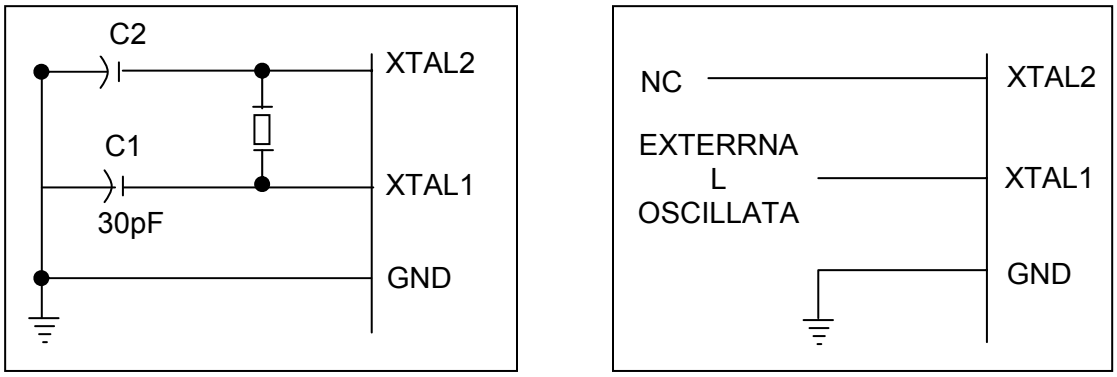
Trên hình 4.1 là sơ đồ bố trí chân của 8051. Ta thấy rằng trong 40 chân thì có 32 chân dành cho các cổng P0, P1, P2 và P3 với mỗi cổng có 8 chân. Các chân còn lại được dành cho nguồn V_{CC}, đất GND, các chângiao động XTAL1 và XTAL2 tái lập RST cho phép chốt địa chỉ ALE truy cập được địa chỉ ngoài $\overline{EA}$, 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. Hay nói cách khác là chúng phải được nối để cho hệ thống làm việc mà không cần biết bộ vi điều khiển thuộc họ 8051 hay 8031. Còn hai chân khác là $\overline{PSEN}$ và ALE được sử dụng chủ yếu trong các hệ thống dựa trên 8031.

1. Chân V_{CC}: Chân số 40 là V_{CC} cấp điện áp nguồn cho chíp. Nguồn điện áp là +5V.
2. Chân GND: Chân GND: Chân số 20 là GND.
3. Chân XTAL1 và XTAL2:

8051 có một bộ giao động trên chíp nhưng nó yêu cầu có một xung đồng hồ ngoài để chạy nó. Bộ giao động thạch anh thường xuyên nhất được nối tới các chân đầu vào XTAL1 (chân 19) và XTAL2 (chân 18). Bộ giao động thạch anh được nối tới XTAL1 và XTAL2 cũng cần hai tụ điện giá trị 30pF. Một phía của tụ điện được nối xuống đất như được trình bày trên hình 4.2a.

Cần phải lưu ý rằng có nhiều tốc độ khác nhau của họ 8051. Tốc độ được coi như là tần số cực đại của bộ giao động được nối tới chân XTAL. Ví dụ, một chíp 12MHz hoặc thấp hơn. Tương tự như vậy thì một bộ vi điều khiển cũng yêu cầu một tinh thể có tần số không lớn hơn 20MHz. Khi 8051 được nối tới một bộ giao động tinh thể thạch anh và cấp nguồn thì ta có thể quan sát tần số trên chân XTAL2 bằng máy hiện sóng. Nếu ta quyết định sử dụng một nguồn tần số khác bộ giao động thạch anh chẳng hạn như là bộ giao động TTL thì nó sẽ được nối tới chân XTAL1, còn chân XTAL2 thì để hở không nối như hình 4.2b.



Hình 4.2: a) Nối XTAL tới 8051 b) Nối XTAL tới nguồn đồng bộ ngoài.

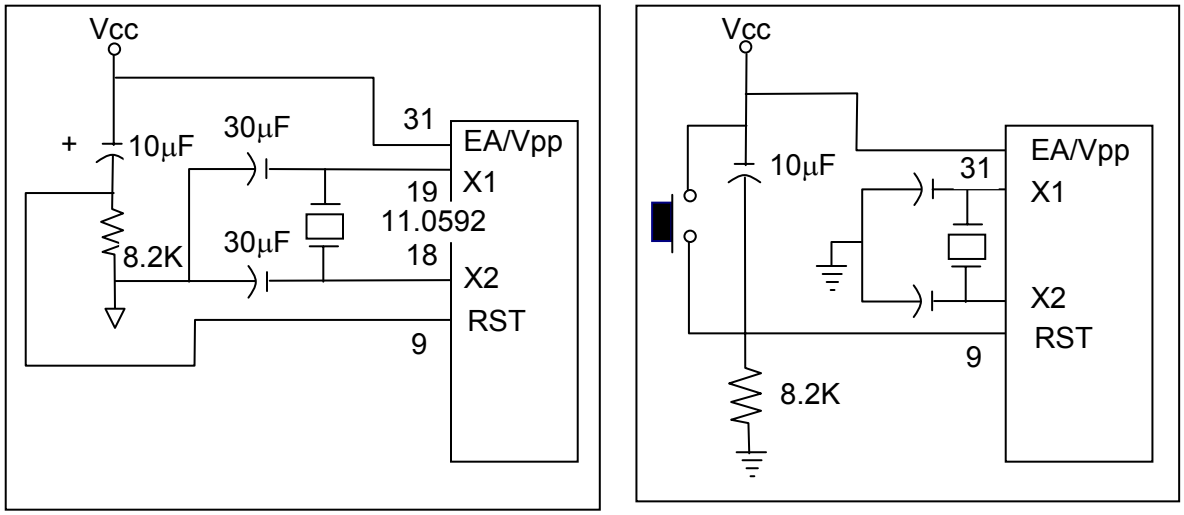
4. Chân RST.

Chân số 9 là chân tái lập RESET. Nó là một đầu vào và có mức tích cực cao (bình thường ở mức thấp). Khi cấp xung cao tới chân này thì bộ vi điều khiển sẽ tái lập và kết thúc mọi hoạt động. Điều này thường được coi như là sự tái bật nguồn. Khi kích hoạt tái bật nguồn sẽ làm mất mọi giá trị trên các thanh ghi. Bảng 4.1 cung cấp một cách liệt kê các thanh ghi của 8051 và các giá trị của chúng sau khi tái bật nguồn.

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
	0000

Lưu ý rằng giá trị của bộ đếm chương trình PC là 0 khi tái lập để ép CPU nạp mã lệnh đầu tiên từ bộ nhớ ROM tại vị trí ngăn nhớ 0000. Điều này có nghĩa là ta phải đặt dòng đầu tiên của mã nguồn tại vị trí ngăn nhớ 0 của ROM vì đây là mã CPU tĩnh thức và tìm lệnh đầu tiên. Hình 4.3 trình bày hai cách nối chân RST với mạch bật nguồn.



Hình 4.3: a) Mạch tái bật nguồn RESET.
b) Mạch tái bật nguồn với Debounce.

Nhằm làm cho đầu vào RESET có hiệu quả thì nó phải có tối thiểu 2 chu kỳ máy. Hay nói cách khác, xung cao phải kéo dài tối thiểu 2 chu kỳ máy trước khi nó xuống thấp.

Trong 8051 một chu kỳ máy được định nghĩa bằng 12 chu kỳ dao động như đã nói ở chương 3 và được trình bày tại vị trí 4.1.

5. Chân $\overline{EA}$:

Các thành viên họ 8051 như 8751, 98C51 hoặc DS5000 đều có ROM trên chíp lưu cất chương trình. Trong các trường hợp như vậy thì chân $\overline{EA}$ được nối tới V_{CC}. Đối với các thành viên củ họ như 8031 và 8032 mà không có ROM trên chíp thì mã chương trình được lưu cất ở trên bộ nhớ ROM ngoài và chúng được nạp cho 8031/32. Do vậy, đối với 8031 thì chân $\overline{EA}$ phải được nối đất để báo rằng mã

chương trình được cất ở ngoài. $\overline{EA}$ có nghĩa là truy cập ngoài (External Access) là chân số 31 trên vỏ kiểu DIP. Nó là một chân đầu vào và phải được nối hoặc với V_{CC} hoặc GND. Hay nói cách khác là nó không được để hở.

Ở chương 14 chúng ta sẽ trình bày cách 8031 sử dụng chân này kết hợp với $\overline{PSEN}$ để truy cập các chương trình được cất trên bộ nhớ ROM ở ngoài 8031. Trong các chip 8051 với bộ nhớ ROM trên chip như 8751, 89C51 hoặc DS5000 thì $\overline{EA}$ được nối với V_{CC}.

Ví dụ 4:

Hãy tìm chu kỳ máy đối với a) XTAL = 11.0592MHz b) XTAL = 16MHz.

Lời 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.

Các chân mô tả trên đây phải được nối mà không cần thành viên nào được sử dụng. Còn hai chân dưới đây được sử dụng chủ yếu trong hệ thống dựa trên 8031 và sẽ được trình bày chi tiết ở chương 11.

6. Chân $\overline{PSEN}$:

Đây là chân đầu ra cho phép cất chương trình (Program Store Enable) trong hệ thống dựa trên 8031 thì chương trình được cất ở bộ nhớ ROM ngoài thì chân này được nối tới chân OE của ROM. Chi tiết được bàn ở chương 14.

7. Chân ALE:

Chân cho phép chốt địa chỉ ALE là chân đầu ra và được 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 chíp 74LS373. Điều này được nói chi tiết ở chương 14.

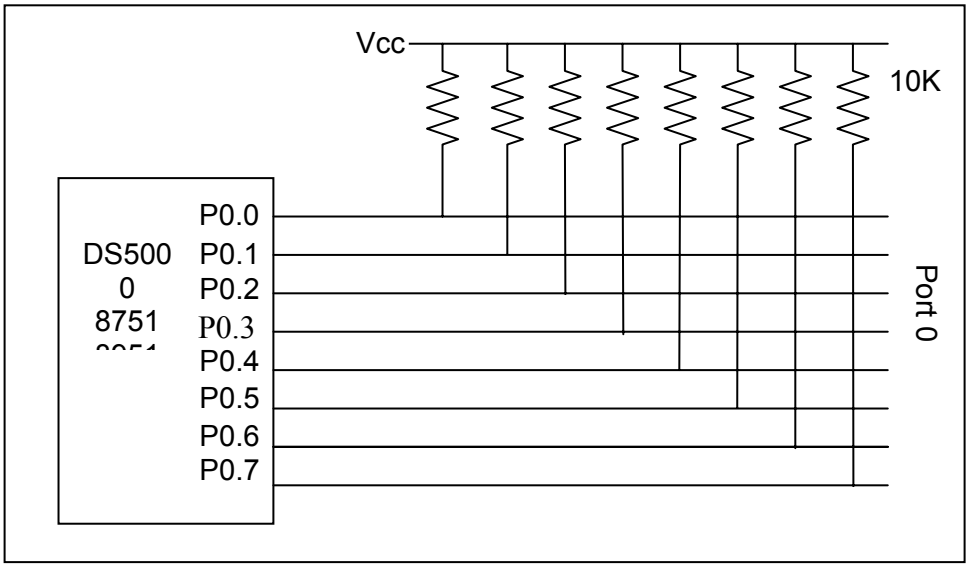
8. Các chân cổng vào ra và các chức năng của chúng.

Bốn cổng P0, P1, P2 và P3 đều sử dụng 8 chân và tạo thành cổng 8 bít. Tất cả các cổng khi RESET đều được cấu hình như các đầu ra, sẵn sàng để được sử dụng như các cổng đầu ra. Muốn sử dụng cổng nào trong số các cổng này làm đầu vào thì nó phải được lập trình.

9. Cổng P0.

Cổng 0 chiếm tất cả 8 chân (từ chân 32 đến 39). Nó có thể được dùng như cổng đầu ra, để sử dụng các chân của cổng 0 vừa làm đầu ra, vừa làm đầu vào thì mỗi chân phải được nối tới một điện trở kéo bên ngoài 10kΩ. Điều này là do một thực tế là cổng P0 là một màng mở khác với các cổng P1, P2 và P3. Khái niệm máng mở được sử dụng trong các chíp MOS về chừng mực nào đó nó giống như Cô-lec-tơ hở đối với các chíp TTL. Trong bất kỳ hệ thống nào sử dụng 8751, 89C51 hoặc DS5000 ta thường nối cổng P0 tới các điện trở kéo, Xem hình 4.4 bằng cách này ta có được các ưu điểm của cổng P0 cho cả đầu ra và đầu vào. Với những điện trở kéo ngoài được nối khi tái lập cổng P0 được cấu hình như một cổng đầu ra. Ví dụ, đoạn mã sau đây sẽ liên tục gửi ra cổng P0 các giá trị 554 và AAH.

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



Hình 4.4: Cổng P0 với các điện trở kéo.

a) Cổng P0 đầu vào: Với các điện trở được nối tới cổng P0 nhằm để tạo nó thành cổng đầu vào thì nó phải được lập trình bằng cách ghi 1 tới tất cả các bit. Đoạn mã dưới đây sẽ cấu hình P0 lúc đầu là đầu vào bằng cách ghi 1 đến nó và sau đó dữ liệu nhận được từ nó được gửi đến P1.

b)

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

b) Vai trò kép của cổng P0: Như trình bày trên hình 4.1, cổng P0 được gán AD0 - AD7 cho phép nó được sử dụng vừa cho địa chỉ, vừa cho dữ liệu. Khi nối 8051/31 tới bộ nhớ ngoài thì cổng 0 cung cấp cả địa chỉ và dữ liệu 8051 dồn dữ liệu và địa chỉ qua cổng P0 để tiết kiệm số chân. ALE báo nếu P0 có địa chỉ hay dữ liệu khi ALE - 0 nó cấp dữ liệu D0 - D7. Do vậy, ALE được sử dụng để tách địa chỉ và dữ liệu với sự trợ giúp của chốt 74LS373 mà ta sẽ biết cụ thể ở chương 14.

10. Cổng P1.

Cổng P1 cũng chiếm tất cả 8 chân (từ chân 1 đến chân 8) nó có thể được sử dụng như đầu vào hoặc đầu ra. So với cổng P0 thì cổng này không cần đến điện trở kéo vì nó đã có các điện trở kéo bên trong. Trong quá trình tái lạp thì cổng P1 được

cấu hình như một cổng đầu ra. Ví dụ, đoạn mã sau sẽ gửi liên tục các giá trị 55 và AAH ra cổng P1.

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

Cổng P1 như đầu vào: Để biến cổng P1 thành đầu vào thì nó phải được lập trình bằng cách ghi một đến tất cả các bit của nó. Lý do về điều này được bàn ở mục lục Appendix C.2. Trong đoạn mã sau, cổng P1 lúc đầu đươc cấu hình như cổng đầu vào bằng cách ghi 1 vào các bit của nó và sau đó dữ liệu nhận được từ cổng này được cất vào R7, R6 và R5.

```
        MOV     A, #0FFH    ; Nạp A = FF ở dạng hex
        MOV     P1, A       ; Tạo cổng P1 thành cổng đầu vào bằng
                            ; cách ghi 1 vào các bit của nó.
        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 nó vào thanh ghi R5
```

11. Cổng P2:

Cổng P2 cũng chiếm 8 chân (các chân từ 21 đến 28). Nó có thể được sử dụng như đầu vào hoặc đầu ra giống như cổng P1, cổng P2 cũng không cần điện trở kéo vì nó đã có các điện trở kéo bên trong. Khi tái lập, thì cổng P2 được cấu hình như một cổng đầu ra. Ví dụ, đoạn mã sau sẽ gửi liên tục ra cổng P2 các giá trị 55H và AAH. Đó là tất cả các bit của P2 lên xuống liên tục.

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

a) Cổng P2 như đầu vào.

Để tạo cổng P2 như đầu vào thì nó phải được lập trình bằng cách ghi các số 1 tới tất cả các chân của nó. Đoạn mã sau đây đầu tiên cấu hinh P2 là cổng vào bằng cách ghi một đến tất cả các chân của nó và sau đó dữ liệu nhận được từ P2 được gửi liên tục đến P1.

```
        MOV     A, 0FFH                 ; Gán A giá trị FF dạng Hex
        MOV     P2, A               ; Tạo P2 là cổng đầu vào bằng cách
```

```
                                        ; ghi một đến các chân của nó
BACK:       MOV     A, P2               ; Nhận dữ liệu từ P2
            MOV     P1, A               ; Gửi nó đến P1
            SJMP    BACK                ; Lặp lại
```

b) Vai trò kép của P2.

Trong các hệ thống dựa trên 8751, 89C51 và DS5000 thì P2 được dùng như đầu ra đơn giản. Tuy nhiên trong hệ thống dựa trên 80312 thì cổng P2 phải được dùng cùng với P0 để tạo ra địa chỉ 16 brt đối với bộ nhớ ngoài. Như chỉ ra trên hình 4.1 cổng P2 cũng được chỉ định như là A8 - A15 báo chức năng kép của nó. Vì một bộ 8031 có khả năng truy cập 64k byte bộ nhớ ngoài, nó cần một đường địa chỉ 16 bít. Trong khi P.0 cung cấp 8 bit thấp qua A0 - A7. Công việc của P2 là cung cấp các bít địa chỉ A8 - A15. Hay nói cách khác khi 8031được nối tới bộ nhớ ngoài thì P2 được dùng cho 8 bít cao của địa chỉ 16 bit và nó không thể dùng cho vào ra. Điều này sẽ được trình bày chi tiết ở chương 14.

Từ những trình bày trên đây ta có thể kết luận rằng trong các hệ thống dựa trên các bộ vi điều khiển 8751, 89C51 hoặc DS5000 thì ta có 3 cổng P0, P1và P2 cho các thao tác vào ra và như thế là có thể đủ cho các ứng dụng với hầu hết các bộ vi điều khiển. Còn cấp P3 làđể dành cho ngắt và ta sẽ cùng bàn dưới đây.

11 - Cổng P3:

Cổng P3 chiếm tổng cộng là 8 chân từ chân 10 đến chân 17. Nó có thể được sử dụng như đầu vào hoặc đầu ra. Cổng P3 không cần các điện trở kéo cũng như P1 và P2. Mặc dù cổng P3 được cấu hình như một cổng đầu ra khi tái lập, nhưng đây không phải là cách nó được ứng dụng phổ biến nhất. Cổng P3 có chức năng bổ xung là cung cấp một số tín hiệu quan trọng đặc biệt chẳng hạn như các ngắt. Bảng 4.2 cung cấp các chức năng khác của cổng P3. Thông tin này áp dụng cho cả 8051 và 8031.

Bảng 4.2: Các chức năng khác của cổng P3

Bít của cổng P3	Chức năng	chân số
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 (TO)	14
P3.5	1 Bộ định thời 1(T1)	15
P3.6	Ghi (WR)	16
P3.7	Đọc (RD)	17

Các bit P3.0 và P3.1 được dùng cho các tín hiệu nhận và phát dữ liệu trong truyền thông dữ liệu nối tiếp. Xem chương 10 đết biết các chúng được nối ghép như thế nào. Các bit P3.2 và P3.3 được dành cho các ngắt ngoài và chúng được trình bày chi tiết ở chương 11. Bit P3.4 và P3.5 được dùng cho các bộ định thêm 0 và 1và chi tiết được trình bày ở chương 9. Cuối cùng các bit P3.6 và P3.7 được cấp cho các tín hiệu ghi và đọc các bộ nhớ ngoài được nối tới các hệ thống dựa trên 8031. Chương

14 sẽ trình bày cách chúng được sử dụng như thế nào trong các hệ thống dựa trên 8031. Trong các hệ thống dựa trên 8751, 89C51 hoặc D35000 thì các chân P3.6 và P3.7 được dùng cho vào - ra còn các chân khác của P3 được sử dụng bình thường trong vai trò chức năng thay đổi.

4.2 Lập trình vào - ra: thao tác bit.

4.2.1 các cách khác nhau để truy cập toàn bộ 8 bit.

Trong đoạn mà dưới đây cũng như trong nhiều ví dụ vào ra trước đây toàn bộ 8 bit của cổng P1 được cập.

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

Đoạn mà trên chốt mỗi bit của P1 một cách liên tục. Chúng ta đã thắng một biến thể của chương trình trên trước đó. Bây giờ ta có thể viết lại đoạn mã trên theo cách hiệu quả hơn bằng cách truy cập trực tiếp cổng mà không qua thanh ghi tổng như sau:

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

Ta có thể viết một dạng khác của đoạn mà trên bằng kỹ thuật đọc - sửa đổi ghi như ở mục 4.2.2 dưới đây.

4.2.2 Đặc điểm Đọc- sửa đổi - ghi (Read - Modify – Write).

Các cổng trong 8051 có thể được truy cập bằngkỹ thuật được gọi là Đọc-sửa đổi-ghi. Đặc điểm này tiết kiệm rất nhiều dòng lệnh bằng cách kết hợp tất cả 3 thao tác: 1đọc cổng, 2 sửa đổi nóvà 3 ghi nó ra cổng vào một lệnh đơn. Đoạn mà dưới đây trước hết đặt 01010101(nhị phân) vào cổng 1. Sau đólệnh "XLR P1, #0FFH" thực hiện phép lô-gích OR loại trừ là XOR trên cổng p1 với 1111 1111 (nhị phân) và sau đó ghi kết quả trở lại cổng P1.

```
        MOV     P1, #55H        ; P1 = 01010101
AGAIN:  XLR     P1,# 0FFH       ; EX - 0R P1 với 1111 1111
        ACALL   DELAY
        SJMP    AGAIN
```

Lưu ý rằng lệnh X0R của 55H và FFH sẽ cho kết quả là AAH. Tương tự như vậy lệnh X0R của AAH với FFH lại cho giá trị kết quả là 55H. Các lệnh lô-gích được trình bày ở chương 7.

4.2.3. Khả năng đánh địa chỉ theo bít của các cổng

Có nhiều lúc chúng ta cần truy cập chỉ 1 hoặc2 bít của cổng thay vì truy cập cả 8 bit của cổng. Một điểm mạnh của các cổng 8051 là chúng có khả năng truy cập từng bít riêng rẽ mà không làm thay đổi các bít còn lại trong cổng đó ví dụ, đoạn mà dưới đây chốt bit P1.2 liên tục:

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

Một biến thể khác của đoạn mã trên là:

```
AGACN:          SETB    P1.2        ; Chỉ thay đổi bít P1.2 lên cao
        ACALL   DELAY
        CLR     P1.2        ; Xoá bít P1.2 xuống thấp
        ACALL   DELAY
        SJMP    AGAIN
```

Lưu ý rằng bít P1.2 là bít thứ 3 của cổng P1, vì bít thứ nhất là P1.0 và bit thứ hai là P1.1 v.v...

Bảng 4.3 trình bày các bít của các cổng vào ra của 8051. Xem ví dụ 4.2 về thao tác bít của các bít vào - ra. Lưu ý rằng trong ví dụ 4.2 các bit không dùng đến là không bị ảnh hưởng. Đây là khả năng đánh địa chỉ theo bit của các cổng vào - ra và là một trong những điểm mạnh nhất của bộ vi điều khiển 8051.

Ví dụ 4.2: hãy viết chương trình thực hiện các công việc sau:

a) Duy trì hiển thị bit P1.2 cho đến khi nó lên cáp
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

Lời 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            ; Tạo P2.3 xuống thấp để có xung H-
T0-L
```

Trong chương trình này lệnh "JNB P1.2, AGCN" (JNB có nghĩa là nhảy nếu không bit) ở lại vòng lặp cho đến khi P1.2 chưa lên cao. Khi P1.2 lên cao nó thoát ra khỏi vòng lặp ghi giá trị 45H tới cổng P0 và tạo ra xung H-to-L bằng chuỗi các lệnh SETB và CLR.

CHƯƠNG 5
Các chế độ đánh địa chỉ của 8051

CPC có thể truy cập dữ liệu theo nhiều cách khác nhau. Dữ liệu có thể ở trong một thanh ghi hoặc trong bộ nhớ hoặc được cho như một giá trị tức thời các cách truy cập dữ liệu khác nhau được gọi là các chế độ đánh địa chỉ. Chương này chúng ta bàn luận về các chế độ đánh địa chỉ của 8051 trong phạm vi một số ví dụ.

Các chế độ đánh địa chỉ khác nhau của bộ vi xử lý được xác định như nó được thiết kế và do vậy người lập trình không thể đánh địa chỉ khác nhau là:

1. tức thời 2. Theo thanh ghi 3. Trực tiếp
4. gián tiếp qua thanh ghi 5. Theo chỉ số

5.1 Các chế độ đánh địa chỉ tức thời và theo thanh ghi

5.1.1 Chế độ đánh địa chỉ tức thời

Trong chế độ đánh địa chỉ này toán hạng nguồn là một hằng số. Và như tên gọi của nó thì khi một lệnh được hợp dịch toán hạng đi tức thi ngay sau mã lệnh. Lưu ý rằng trước dữ liệu tức thời phải được đặt dấu (#) chế độ đánh địa chỉ này có thể được dùng để nạp thông tin vào bất kỳ thanh ghi nào kể cả thanh ghi con trỏ dữ liệu DPTR. Ví dụ:

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

Mặc dù thanh ghi DPTR là 16 bit nó cũng có thể được truy cập như 2 thanh ghi 8 bit DPH và DPL trong đó DPH là byte cao và DPL là byte thấp. Xét đoạn mã dưới đây:

```
MOV         DPTR, #2550H
MOV         A, #50H
MOV         DPH, #25H
```

Cũng lưu ý rằng lệnh dưới đây có thể tạo ra lỗi vì giá trị nạp vào DPTR lớn hơn16 bit:

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

Ta có thể dùng chỉ lệnh Eqw để truy cập dữ liệu tức thời như sau

```
            COUNT           EDU   30
            ...             ...
            MOV             R4, #COUNT          ; R4 = 1E (30 = 1EH)
            MOV             DPTR, #MYDATA       ; DPTR = 200H

            ORG   200H
MYDATA:     DB    “America”
```

Lưu ý rằng ta cũng có thể sử dụng chế độ đánh được chỉ tức thời để gửi dữ liệu đến các cổng của 8051.
Ví dụ "MOV P1, #55H" là một lệnh hợp lệ.

5.1.2 chế độ đánh địa chỉ theo thanh ghi:

Chế độ đánh địa chỉ theo thanh ghi liên quan đến việc sử dụng các thanh ghi để dữ liệu cần được thao tác các ví dụ về đánh địa chỉ theo thanh ghi như sau:

```
MOV   A,  RO        ; 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         ; Lưu nội dung thanh ghi A vào thanh ghi R6
```

Cũng nên lưu ý rằng các thanh ghi nguồn và đích phải phù hợp về kích thước. Hay nói cách khác, nếu viết " MOV DPTR, A" sẽ cho một lỗi vì nguồn là thanh ghi 8 bit và đích lại là thanh ghi 16 bit. Xét đoạn mã 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 từ 0 đến 7) nhưng việc chuyển dữ liệu giữa các thanh ghi Rn thì không được phép. Ví dụ, lệnh "MOV R4, R7" là không hợp lệ.

Trong hai chế độ đánh địa chỉ đầu tiên, các toán hạng có thể hoặc ở bên trong một trong các thanh ghi hoặc được gắn liền với lệnh. Trong hầu hết các chương trình dữ liệu cần được xử lý thường ở trong một số ngăn của bộ nhớ RAM hoặc trong không gian mà của ROM. Có rất nhiều cách để truy cập dữ liệu này mà phần tiếp theo sẽ xét đến.

5.2 Truy cập bộ nhớ sử dụng các chế độ đánh địa chỉ khác nhau.

5.2.1 Chế độ đánh địa chỉ trực tiếp.

Như đã nói ở chương 2 trong 8051 có 128 byte bộ nhớ RAM. Bộ nhớ RAM được gán các địa chỉ từ 00 đến FFH và được phân chia như sau:
1. Các ngăn nhớ từ 00 đến 1FH được gán cho các băng thanh ghi và ngăn xếp.
2. Các ngăn nhớ từ 20H đến 2FH được dành cho không gian đánh địa chỉ theo bit để lưu các dữ liệu 1 bit.
3. Các ngăn nhớ từ 30H đến 7FH là không gian để lưu dữ liệu có kích thước 1byte.

Mặc dù toàn bộ byte của bộ nhớ RAM có thể được truy cập bằng chế độ đánh địa chỉ trực tiếp, nhưng chế độ này thường được sử dụng nhất để truy cập các ngăn nhớ RAM từ 30H đến 7FH. Đây là do một thực tế là các ngăn nhớ dành cho băng ghi được truy cập bằng thanh ghi theo các tên gọi của chúng là R0 - R7 còn các ngăn nhớ khác của RAM thì không có tên như vậy. Trong chế độ đánh địa chỉ trực tiếp thì dữ liệu ở trong một ngăn nhớ RAM mà địa chỉ của nó được biết và địa chỉ này được cho như là một phần của lệnh. Khác với chế độ đánh địa chỉ tức thì mà toán hạng tự

nó được cấp với lệnh. Dấu (# 0 là sự phân biệt giữa hai chế độ đánh địa chỉ. Xét các ví dụ dưới đây và lưu ý rằng các lệnh không có dấu (#):

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

Như đã nói ở trước thì các ngăn nhớ trừ 0 đến 7 của RAM được cấp cho bằng 0 của các thanh ghi R0 - R7. Các thanh ghi này có thể được truy cập theo 2 cách như sau:

```
MOV   A, 4          ; Hai lệnh này giống nhau đều sao nội dung thanh ghi  R4 vào A
MOV   A, R4

MOV   A, 7          ; Hai lệnh này đều như nhau là sao nội dung R7 vào thanh ghi A
MOV   A,R7
```

Để nhấn mạnh sự quan trọng của dấu (#) trong các lệnh của 8051. Xét các mã cho sau đây:

```
MOV   R2, #05          ; Gán R2=05
MOV   A, 2        ; Sao nội dung thanh ghi R2 vào A
MOV   B, 2        ; Sao nội dung thanh ghi R2 vào  B
MOC   7,2         ; Sao nội dung thanh ghi R7 vì lệnh “MOV R7, R2” là không hợp  lệ.
```

Mặc dù sử dụng các tên R0 - R7 dễ hơn các địa chỉ bộ nhớ của chúng nhưng các ngăn nhớ 30H đến 7FH của RAM không thể được truy cập theo bất kỳ cách nào khác là theo địa chỉ của chúng vì chúng không có tên.

5.2.2 các thanh ghi SFSR và các địa chỉ của chúng.

Trong các thanh ghi được nói đến từ trước đến giờ ta thấy rằng các thanh ghi R0 - R7 là một phần trong 128 byte của bộ nhớ RAM. Vậy còn các thanh ghi A, B, PSW và DPTR là một bộ phận của nhóm các thanh ghi nhìn chung được gọi là các thanh ghi đặc biệt SFR (Special Funtion Register). Có rất nhiều thanh ghi với chức năng đặc biệt và chúng được sử dụng rất rộng rãi mà ta sẽ trình bày ở các chương sáu. Các thanh ghi FR có thể được truy cập theo tên của chúng (mà dễ hơn rất nhiều) hoặc theo các địa chỉ của chúng. Ví dụ địa chỉ của thanh ghi A là EOH và thanh ghi B là FOH như cho ở trong bảng 5.1. Hãy để ý đến những cặp lệnh có cùng ý nghĩa dưới đây:

```
MOV   0E0H, #55H       ; Nạp 55H vào thanh ghi A(A=55H)
MOV   A,    #55H       ;

MOV   0F0H, #25H       ; Nạp 2SH vào thanh ghi  B ( B = 25)
MOV   3, #25H                  ;
```

```
MOV   0E0H          ; Sao nội dung thanh ghi R2 vào A
MOV   A, R2         ;

MOV   0F0           ; Sao nội dung thanh ghi R0 vào B
MOV   B, R0         ;
```

Bảng 5.l dưới đây liệt kê các thanh ghi chức năng đặc biệt SFR của 8051 và các địa chỉ của chúng. Cần phải lưu ý đến hai điểm sau về các địa chỉ của SFR:

1. Các thanh ghi SFR có địa chỉ nàm giữa 80H và FFH các địa chỉ này ở trên 80H, vì các địa chỉ từ 00 đến 7FH là địa chỉ của bộ nhớ RAM bên trong 8051.
2. không phải tất cả mọi địa chỉ từ 80H đến FFH đều do SFH sử dụng, nhưng vị trí ngăn nhớ từ 80H đến FFH chưa dùnglà để dữ trữ và lập trình viên 8051 cũng không được sử dụng.

Bảng 5.1: Các 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	A08H
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

*Các thanh ghi có thể đánh địa chỉ theo bit.

Xét theo chế độ đánh địa chỉ trực tiếp thì cần phải lưu ý rằng giá trị địa chỉ được giới hạn đến 1byte, 00 - FFH. Điều này có nghĩa là việc sử dụng của chế độ

đánh địa chỉ này bị giới hạn bởi việc truy cập các vị trí ngăn nhớ của RAM và các thanh ghi với địa chỉ được cho 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

Lời giải:

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

b) Từ bảng 5.1 ta lấy đại chỉ cổng P1 là 80H và P2 là A0H

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

5.2.3 Ngăn xếp và chế độ đánh địa chỉ trực tiếp.

Một công dụng chính khác của chế độ đánh địa chỉ trực tiếp là ngăn xếp. Trong họ 8051 chỉ có chế độ đánh địa chỉ trực tiếp là được phép đẩy vào ngăn xếp. Do vậy, một lệnh như "PVSH A" là không hợp lệ. Việc đẩy thanh ghi A vào ngăn xếp phải được viết dưới dạng "PVAH 0E0H" với 0E0H là địa chỉ của thanh ghi A. Tương tự như vậy để đẩy thanh ghi R3 rãnh 0 vào ngăn xếp ta phải viết là "PVSH 03". Chế độ đánh địa chỉ trực tiếp phải được sử dụng cho cả lệnh POP. Vì dụ "POP 04" sẽ kéo đỉnh của ngăn xếp vào thanh ghi R4 rãnh 0.

Ví dụ 5.2:

Trình bày mã để đẩy thanh ghi R5, R6 và A vào ngăn xếp và sau đó kéo chùng ngược trở lại R2, R3 và B tương ứng.

Lời 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
```

5.2.4 chế độ đánh địa chỉ gián tiếp thanh ghi.

Trong chế độ này, một thanh ghi được sử dụng như một con trỏ đến dữ liệu. Nếu dữ liệu ở bên trong CPU thì chỉ các thanh ghi R0 và R1 được sử dụng cho mục đích này. Hay nói cách khác các thanh ghi R2 - R7 không có thể dùng được để giữ địa chỉ của toán hạng nằm trong RAM khi sử dụng chế độ đánh địa chỉ này khi Ro

và R1 được dùng như các con trỏ, nghĩa là khi chúng giữ các địa chỉ của các ngăn nhớ RAM thì trước chúng phải đặt dấu (@) như chỉ ra dưới đây.

```
        MOV   A, @ R0              ; Chuyển nội dung của ngăn nhớ RAM có địa chỉ
trong RO và A
        MOV   @ R1, B              ; Chuyển nội dung của B vào ngăn nhớ RAM có địa
chỉ ở R1
```

Lưu ý rằng R0 cũng như R1 luôn có dấu "@" đứng trước. Khi không có dấu này thì đó là lệnh chuyển nội dung các thanh ghi Ro và R1 chứ không phải dữ liệu ngăn nhớ mà địa chỉ có trong R0 và R1.

Ví dụ 5.3:

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

a) Chế độ đánh địa chỉ trực tiếp

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

c) Chế độ b có dùng vòng lặp

Lời giải:

```
                MOV   A, #55H              ; Nạp A giá trị  55H
                MOV   40H, A       ; Sao chép A vào ngăn nhớ RAM  40H
                MOV   41H, A       ; Sao chép A vào ngăn nhớ RAM  41H
                MOV   42H, A       ; Sao chép A vào ngăn nhớ RAM  42H
                MOV   43H, A       ; Sao chép A vào ngăn nhớ RAM  43H
                MOV   44H, A       ; Sao chép A vào ngăn nhớ RAM  44H
b)
                MOV   A, # 55H     ; Nạp vào A giá trị 55H
                MOV   R0, #40H     ; Nạp con trỏ  R0 = 40 H
                MOV   @R0, A               ; Sao chép A vào vị trí ngăn nhớ RAM do R0
        chỉ đến
                INC   R0           ; Tăng con trỏ. Bây gì R0 = 41H
                MOV   @R0, A               ; Sao chép A vào vị trí ngăn nhớ RAM do R0
        chỉ
                INC   R0           ; Tăng con trỏ. Bây giờ R0 = 42H
                MOV   @R0,A        ; Sao chép Avào vị trí ngăn nhớ RAM do R0 chỉ
                INC   R0           ; Tăng con trỏ. Bây giờ R0 = 43H
                MOV   @R0, A               ; Sao chép A vào vị trí ngăn nhớ RAM do R0
        chỉ
                MOV   @R0, A               ;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 ngăn nhớ RAM R0 = 40H
                MOV   R2, #05              ; Nạp bộ đếm R2 = 5
 AGAIN:         MOV   @R0, A               ; Sao chép A vào vị trí ngăn nhớ RAM
do Ro chi đến
                INC   R0           ; Tăng con trỏ Ro
                DJNZ  R2, AGAIN    ; Lặp lại cho đến khi bộ đếm = 0.
```

5.2.5 Ưu điểm của chế độ đánh địa chỉ gián tiếp thanh ghi.

Một trong những ưu điểm của chế độ đánh địa chỉ gián tiếp thanh ghi là nó làm cho việc truy cập dữ liệu năng động hơn so với chế độ đánh địa chỉ trực tiếp.

Ví dụ 5.3 trình bày trường hợp sao chép 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 với một số lần. Ta có thể tạo ra vòng lặp với hai lệnh này như ở lời giải c). Lời giải c) là hiệu quả nhất và chỉ có thể khi sử dụng chế độ đánh địa chỉ gián tiếp qua thanh ghi. Vòng lặp là không thể trong chế độ đánh địa chỉ trực tiếp. Đây là sự khác nhau chủ yếu giữa đánh địa chỉ trực tiếp và gián tiếp.

Ví dụ 5.4:

Hãy viết chương trình để xoá 16 vị trí ngăn nhớ RAM bắt đầu tại địa chỉ 60H.

Lời giải:

```
        CLR   A              ; Xoá A=0
        MOV   R1, #60H       ; Nạp con trỏ. R1= 60H
        MOV   R7, #16H       ;Nạp bộ đếm, R7 = 1 6  (10 H dạng hex)
AGAIN:  MOV   @R1, A         ; Xoá vị trí ngăn nhớ RAM do R1 chỉ đến
        INC   R1             ; Tăng R1
        DJNZ  R7, AGAiN      ; Lặp lại cho đến khi bộ đếm = 0
```

Một ví dụ về cách sử dụng cả R0 và R1 trong chế độ đánh địa chỉ gián tiếp thanh ghi khi truyền khối được cho trong ví dụ 5.5.

Ví dụ 5.5:

Hãy viết chương trình để sao chép 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

Lời giải:

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

5.2.6 Hạn chế của chế độ đánh địa chỉ gián tiếp thanh ghi trong 8051.

Như đã nói ở phần trước rằng R0 và R1 là các thanh ghi duy nhất có thể được dùng để làm các con trỏ trong chế độ đánh địa chỉ gián tiếp thanh ghi. Vì R0 và R1 là các thanh ghi 8 bit, nên việc sử dụng của chúng bị hạn chế ở việc truy cập mọi thông tin trong các ngăn nhớ RAM bên trong (các ngăn nhớ từ 30H đến 7FH và các thanh ghi SFR). Tuy nhiên, nhiều khi ta cần truy cập dữ liệu được cất trong RAM ngoài hoặc trong không gian mã lệnh của ROM trên chip. Hoặc là truy cập bộ nhớ RAM ngoài hoặc ROM trên chíp thì ta cần sử dụng thanh ghi 16 bit đó là DPTR.

5.2.7 Chế độ đánh địa chỉ theo chỉ số và truy cập bộ nhớ ROM trên chíp.

Chế độ đánh địa chỉ theo chỉ số được sử dụng rộng rãi trongviệc truy cập các phân tử dữ liệu của bảng trong không gian ROM chương trình của 8051. Lệnh được dùng cho mục đích này là "Move A, @ A + DPTR". Thanh ghi 16 bit DPTR là thanh ghi A được dùng để tạo ra địa chỉ của phân tử dữ liệu được lưu cất trong ROM trên chíp. Do các phân tử dữ liệu được cất trong không gian mã (chương trình) của ROM trên chip của 8051, nó phải dùng lệnh Move thay cho lệnh Mov (chủ C ở cuối lệnh là chỉ mà lệnh Code). Trong lệnh này thì nội dung của A được bổ xung vào thanh ghi 16 bit DPTR để tạo ra địa chỉ 16 bit của dữ liệu cần thiết. Xét ví dụ 5.6.

Ví dụ 5.6:

Giả sử từ "VSA" được lưu trong ROM có đĩa chỉ bắt đầu từ 200H và chương trình được ghi vào ROM bắt đầu từ địa chỉ 0. Hãy phân tích cách chương trình hoạt động và hãy phát biểu xem từ "VSA" sau chương trình này được cất vào đâu?

Lời giải:

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

Ở trong chương trình nói trên thì các vị trí ngăn nhớ ROM chương trình 200H - 2002H có các nội dung sau:

200 = ('U'); 201= ('S') và 202 = ('A').

Chúng ta bắt đầu với DPTR = 200H và A = 0.Lệnh "MOVC A, @ A + DPTR chuyền nội dung của vị trí nhớ 200H trong ROM (200H + 0 = 200H) vào A.

Thanh ghi A chứa giá trị 55H là giá trị mà ASC của ký tự "U". 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 chưa ký tự "S". Sau khi chương trình này chạy ta có R0 = 55H, R1 = 53H và R2 = 41H là các mã ASCII của các ký tự "U", "S" và "A".

Ví dụ 5.7:

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

Lời giải

```
; (a)  Phương pháp này 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
      khong gian mã
            MOV         R0, A                   ;Cất nó 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:       SJMP        HERE
;-------------- -- không gian mã của ROM trên chíp  dùng để cất dữ liệu ORG  250H

MYDATA:     DB          "AMER1CA"
            END

;(b) phương pháp này 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 nó vào ngăndn 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 chíp dùng để cất dữ liệu ORG 250H
MYADTA:     DB "AMER1CA", 0                     ; Ký tự Null để kết thúc chuỗi
            END
```

Lưu ý đến cách ta sử dụng lệnh JZ để phát hiện ký tự NOLL khi kết thúc chuỗi

5.2.8 Bảng xắp xếp và sử dụng chế độ đánh địa chỉ theo chỉ số.

Bảng xắp xế là khái niệm được sử dụng rất rộng rãi trong lập trình các bộ vi xử lý. Nó cho phép truy cập các phần từ của một bảng thường xuyên được sử dụng với thao tác cực tiểu. Như một ví dụ, hãy giả thiết rằng đối với một ứng dụng nhất định ta cần x^2 giá trị trong phạm vi 0 đến 9. Ta có thể sử dụng một bảng xắp xếp thay cho việc tính toán nó. Điều này được chỉ ra trong ví dụ 5.8.

Ví dụ 5.8

Hãy viết một chương trình để lấy x giá trị cổng P1 và gửi giá trị x^2 tới cổng P2 liên tục.

Lời giải:

```
        ORG   000
        MOV   DPTR, #300 H      ; Nạp địa chỉ bảng xắp xêlps
        MOV   A, #0FFH          ; Nạp A giá trị FFH
        MOV   P1, A             ; Đặt cổng P1 là đầu vào
BACK:   MOV   A, P1             ; Lấy giá trị X từ P1
        MOVC A, @A + DPTR               ; Lấy giá trị  X  từ bảng  XSDQ-TABLE
        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 bảng lệnh đầu tiên có thể thay bằng "MOV DPTR, #XSQR - TABLE".

Ví dụ 5.9:

Trả lời các câu hỏi sau cho ví dụ 5.8.

a) Hãy chỉ ra nội dung các vị trí 300 - 309H của ROM

b) Tại vị trí nào của ROM có giá trị 6 và giá trị bào nhiêu

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

Lời giải:

a) Các giá trị trong các ngăn nhớ 300H - 309H của ROM là:

300 = (00) 301 = (01) 302 = (04) 303 = (09)

304 = (10) $4 \times 4 = 16 = 10$ in hex

305 = (19) $5 \times 5 = 25 = 19$ in hex

306 = (24) $6 \times 6 = 36 = 24H$

307 = (31) 308 = (40) 309 = (51)

b) vị trí chứa giá trị 306H và giá trị là 24H

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

Ngoài việc sử dụng DPTR để truy cập không gian bộ nhớ ROM chương trình thì nó còn có thể được sử dụng để truy cập bộ nhớ ngoài nối với 8051 (chương 14).

Một thanh ghi khác nữa được dùng trong chế độ đánh địa chỉ theo chỉ số là bộ đếm chương trình (AppendixA).

Trong nhiều ví dụ trên đây thì lệnh MOV đã được sử dụng để đảm bảo đính rõ ràng, mặc dù ta có thể sử dụng bất kỳ lệnh nào khác chừng nào nó hỗ trợ cho chế độ đánh địa chỉ. Ví dụ lệnh "ADD A, @R0" sẽ cộng nội dung ngăn nhớ cho RO chỉ đến vào nội dung của thanh ghi A.

CHƯƠNG 6
Các lệnh số học và các chương trình

6.1 Phép cộng và trừ không dấu.

Các số không dấu được định nghĩa như những dữ liệu mà tất cả mọi bit của chúng đều được dùng để biểu diễn dữ liệu và khó có bit dành cho dấu âm hoặc dương. Điều này có nghĩa là toán hạng có thể nằm giữa 00 và FFH (0 đến 255 hệ thập phân) đối với dữ liệu 8 bit.

6.1.1 Phép cộng các số không dấu.

Trong 8051 để cộng các số với nhau thì thanh ghi tổng (A) phải được dùng đến. Dạng lệnh ADD là:

ADD A, nguồn; A = A + nguồn

Lệnh ADD được dùng để cộng hai toán hạng. Toán hạng đích luôn là thanh ghi A trong khi đó toán hạng nguồn có thể là một thanh ghi dữ liệu trực tiếp hoặc là ở trong bộ nhớ. Hãy nhớ rằng các phép toán số học từ bộ nhớ đến bộ nhớ không bao giờ được phép trong hợp ngữ. Lệnh này có thể thay đổi một trong các bit AF, CF hoặc PF của thanh ghi cờ phụ thuộc vào các toán hạng liên quan. Tác động của lệnh ADD lên cờ tràn sẽ được trình bày ở mục 6.3 vì nó chủ yếu được sử dụng trong các phép toán với số có dấu. Xét ví dụ 6.1 dưới đây:

Ví dụ 6.1:

Hãy biểu diễn xem cá lệnh dưới đây tác động đến thanh ghi cờ như thế nào?

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

Lời giải:

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

Sau phép cộng, thanh ghi A (đích) chứa 00 và 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 đặt lên 1.

AC = 1 vì có phép nhớ từ D3 sang D4

6.1.1.1 Phép cộng các byte riêng rẽ.

ở chương 2 đã trình bày một phép cộng 5 byte dữ liệu. Tổng số đã được cất theo chú ý nhỏ hơn FFH là giá trị cực đại một thanh ghi 8 bit có thể được giữ. Để tính tổng số của một số bất kỳ các toán hạng thì cờ nhớ phải được kiểm tra sau mỗi lần cộng một toán hạng. Ví dụ 6.2 dùng R7 để tích luỹ số lần nhớ mỗi khi các toán hạng được cộng vào A.

Ví dụ 6.2:

Giả sử các ngăn nhớ 40 - 44 của RAM có giá trị sau: 40 = (7D); 41 = (EB); 42 = (C5); 43 = (5B) và 44 = (30). Hãy viết một chương trình tính tổng của các giá

trị trên. Cuối chương trình giá trị thanh ghi A chứa byte thấp và R7 chứa byte cao (các giá trị trên được cho ở dạng Hex).

Lời 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 chỉ 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:

Ba lần lặp lại của vòng lặp được chỉ ra dưới đây. Phần dò theo chương trình dành cho người đọc tự thực hiện.

Trong lần lặp lại đầu tiên của vòng lặp thì 7DH được cộng vào A với CY = 0 và R7 = 00 và bộ đếm R2 = 04.

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

Trong lần lặp lại thứ ba thì C5H được cộng vào A nên A = 2DH và cờ nhớ lại bận. Do vậy R7 lại được tăng lên và bộ đệm R2 = 02.

Ở phần cuối khi vòng lặp kết thúc, tổng số được giữ bởi thanh ghi A và R7, trong đó A giữ byte thấp và R7 chứa byte cao.

6.1.1.2 Phép cộng vó nhớ và phép cộng các số 16 bit.

Khi cộng hai toán hạng dữ liệu 16 bit thì ta cần phải quan tâm đến phép truyền của cờ nhớ từ byte thấp đến byte cao. Lệnh ADDC (cộng có nhớ) được sử dụng trong những trường hợp như vậy. Ví dụ, xét phép cộng hai số sau: 3CE7H + 3B8DH.

```
    3C  E7
+   3B  8D
    78  74
    79
```

Khi byte thứ nhất được cộng (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 trên trong 8051.

Ví dụ 6.3:

Hãy viết chương trình cộng hai số 16 bit. Các số đó là 3CE7H và 3B8DH. Cất tổng số vào R7và R6 trong đó R6 chứa byte thấp.

Lời 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
```

6.1.1.3 Hệ thống số BCD (số thập phân mã hoá theo nhị phân).

Số BCD là số thập phân được mã hoá theo nhị phân 9 mà không dùng số thập phân hay số thập lục (Hex). Biểu diễn nhị phân của các số từ 0 đến 9 được gọi là BCD (xem hình 6.1). Trong tài liệu máy tính ta thường gặp hai khái niệm đối với các số BCD là: BCD được đóng gói và BCD không đóng gói.

Digit	BCD	Digit	BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Hình 6.1: Mã BCD.

a- BCD không đóng gói.

Trong số BCD không đóng gói thì 4 bít thấp của số biểu diễn số BCD còn 4 bit còn lại là số 9. Ví dụ "00001001" và "0000 0101" là những số BCD không đóng gói của số 9 và số 5. Số BCD không đóng gói đòi hỏi một byte bộ nhớ hay một thanh ghi 8 bit để chứa nó.

b- BCD đóng gói.

Trong số BCD đóng gói thì một byte có 2 số BCD trong nó một trong 4 bit thấp và một trong 4 bit cao. Ví dụ "0101 1001" là số BCD đóng gói cho 59H. Chỉ mất 1 byte bộ nhớ để lưu các toán hạng BCD. Đây là lý do để dùng số BCD đóng gói vì nó hiệu quả gấp đôi trong lưu giữ liệu.

Có một vấn để khi cộng các số BCD mà cần phải được khắc phục. Vấn đề đó là sau khi cộng các số BCD đóng gói thì kết quả không còn là số BCD. 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). Hay nói cách khác phép 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. Vấn đề tương tự cũng có thể xảy ra trong số cao (ví dụ khi cộng hai số 52H + 87H = D94). Để giải quyết vấn đề này ta lại phải cộng 6 vào số cao (D9H + 60H = 139). Vấn đề này phổ biến đến mức mọi bộ xử lý như 8051 đều có một lệnh để sử lý vấn đề này. Trong 8051 đó là lệnh "DA A" để giải quyết vấn đề cộng các số BCD.

6.1.1.4 Lệnh DA.

Lệnh DA (Decimal Adjust for addition điều chỉnh thập phân đối với phép cộng) trong 8051 để dùng hiệu chỉnh sự sai lệch đã nói trên đây liên quan đến phép cộng các số BCD. Lệnh giả "DA". Lệnh DA sẽ cộng 6 vào 4 bit thấp hoặc 4 bit cao nếu cần. Còn bình thường nó đê nguyên kết quả tìm được. Ví dụ sau sẽ làm rõ các điểm này.

```
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 hex (nhị phân) A = 6CH
DA      A           ; Điều chỉnh cho 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. Hay nói cách khác trong thanh ghi nguồn có thể là một toán hạng của chế độ đánh địa chỉ bất kỳ thì đích phải là thanh ghi A để DA có thể làm việc được. Cũng cần phải nhấn mạnh rằng lệnh DA phải được sử dụng sau phép cộng các toán hạng BCD và các toán hạng BCD không bao giờ có thể có số lớn hơn 9. Nói cách khác là không cho phép có các số A - F. Điều quan trọng cũng phải lưu ý là DA chỉ làm việc sau phép cộng ADD, nó sẽ không bao giờ làm việc theo lệnh tăng INC.

Tóm tắt về hoạt động của lệnh DA.

Hoạt động sau lệnh ADD hoặc ADDC.

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

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

Lời 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              ; Tăng R0 dịch con trỏ lên ô nhớ kế tiếp
        DJNZ  R2, AGAIN       ; Lặp lại cho đến khi R2 = 0
```

6.1.2 Phép trừ các số không dấu.

Cú pháp: SUBB A, nguồn; A = A - nguồn - CY.

Trong rất nhiều các bộ xử lý có hai lệnh khác nhau cho phép trừ đó là SUB và SUBB (trừ có mượn - Sub, tract with Borrow). Trong 8051 ta chỉ có một lệnh SUBB duy nhất. Để thực hiện SUB từ SUBB, do vậy có hai trường hợp cho lệnh SUBB là: với CY = 0 và với CY = 1. Lưu ý rằng ở đây ta dùng cờ CY để mượn.

6.1.2.1 Lệnh SUBB với CY = 0.

Trong phép trừ thì các bộ vi xử lý 8051 (thực tế là tất cả mọi CPU hiện đại) đều sử dụng phương pháp bù 2. Mặc dù mỗi CPU đều có mạch cộng, nó có thể quá cồng kềnh (và cần nhiều bóng bán dẫn) để thiết kế mạch trừ riêng biệt. Vì ly do đó mà 8051 sử dụng mạch cộng để thực hiện lệnh trừ. Giả sử 8051 sử dụng mạch cộng để thực hiện lệnh trừ và rằng CY - 0 trước khi thực hiện lệnh thì ta có thể tóm tắt các bước mà phần cứng CPU thực hiện lệnh SUBB đối với các số không dấu như sau:

1. Thực hiện lấy bù 2 của số trừ (toán hạng nguồn)
2. Cộng nó vào số bị trừ (A)
3. Đảo nhớ

Đây là 3 bước thực hiện bởi phần cứng bên trong của CPU 8051 đối với mỗi lệnh trừ SUBB bất kể đến nguồn của các toán hạng được cấp có được hỗ trợ chế độ đánh địa chỉ hay không? Sau ba bước này thì kết quả có được và các cờ được bật. Ví dụ 6.5 minh hoạ 3 bước trên đây:

Ví dụ 6.5:

Trình bày các bước liên quan dưới đây:

```
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
```

Lời giải:

```
    A =  3F     0011  1111        0011  1111
-   R3 = 23     0010  0011    +   1101  1101   bù 2 của R3 (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 được nhìn đến cờ nhớ để xác định xem kết quả là âm hay dương.

Nếu sau khi thực hiện SUBB mà CY = 0 thì kết quả là dương. Nếu CY = 1 thì kết quả âm và đích có giá trị bù 2 của kết quả. Thông thường kết quả được để ở dạng bù 2 nhưng các lệnh bù CPL và tăng INC có thể được sử dụng để thay đổi nó. Lệnh CPL thực hiện bù 1 của toán hạng sau đó toán hạng được tăng lên 1 (INC) để 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
```

Lời 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)
                                            đảo CY = 1(bước 3)
```

Cờ CY = 1, kết quả âm ở dạng bù 2.

6.1.2.2 Lệnh SUBB khi CY = 1.

Lệnh này được dùng đối với các số nhiều byte và sẽ theo dõi việc mượn của toán hạng thấp. Nếu CY = 1 trước khi xem thực hiện SUBB thì nó cũng trừ 1 từ kết quả. Xem ví dụ 6.7.

Ví dụ 6.7:

Phân tích chương trình sau:

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

Lời giải:

Sau khi SUBB thì 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 2762H - 1296H = 14CCH.

6.2 Nhân và chia các số không dấu.

Khi nhân và chia hai số trong 8051 cần phải sử dụng hai thanh ghi A và B vì các lệnh nhân và chia chỉ hoạt động với những thanh ghi này.

6.2.1 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:

```
MOV  AB     ; Là phép nhân A × B và kết quả 16 bit được đặt trong A và B.
```

Khi nhận byte với byte thì một trong các toán hạng phải trong thanh ghi A và toán hạng thứ hai phải ở trong thanh ghi B. Sau khi nhân kết quả ở trong các thanh ghi A và B. Phần tiếp thấp ở trong A, còn phần cao ở trong B. Ví dụ dưới đây trình bày phép nhân 25H với 65H. Kết quả là dữ liệu 16 bit được đặt trong 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

6.2.2 Chia hai số không dấu.

8051 cùng chỉ hỗ trợ phép chia hai số không dấu byte cho byte với cú pháp:

```
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 và mẫu số (số chia) phải ở trong thanh ghi B. Sau khi lệnh chia DIV được thực hiện thì thương số 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)
```

Lưu ý các điểm sau khi thực hiện "DIV AB"

Lệnh này luôn bắt CY = 0 và OV = 0 nếu tử số không phải là số 0

Nếu tử số là số 0 (B = 0) thì OV =1 báo lỗi và CY = 0. Thực tế chuẩn trong tất cả mọi bộ vi xử lý khi chia một số cho 0 là bằng cách nào đó báo có kết quả không xác định. Trong 8051 thì 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

6.2.3 Một ứng dụng cho các lệnh chia.

Có những thời điểm khi một bộ ADC được nối tới một cổng và ADC biểu diễn một số dư nhiệt độ hay áp suất. Bộ ADC cấp dữ liệu 8 bit ở dạng Hex trong dải 00 - FFH. Dữ liệu Hex này phải được chuyển đổi về dạng thập phân. Chúng ta thực hiện chia lặp nhiều lần cho 10 và lưu số dư vào như ở ví dụ 6.8.

Ví dụ 6.8:

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

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

Lời 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 Hex (10 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 Hex 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 số dư được lưu cất. Trong trường hợp một số nhị phân 8 bit như FDH chẳng hạn ta có 253 số thập phân như sau (tất cả trong dạng Hex)

	Thương số	**Số dư**	
FD/0A	19	3	(Số thấp - cuối)
19/0A	2	5	(Số giữa)
		2	(Số đầu)

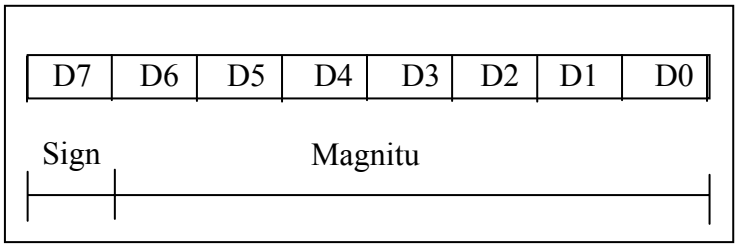
Do vậy, ta có FDH = 253. Để hiển thị dữ liệu này thì nó phải được chuyển đổi về ASCII mà sẽ được mô tả ở chương sau.

6.3 Các khái niệm về số có dấu và các phép tính số học.

Tất cả mọi dữ liệu từ trước đến giờ đều là các số không dấu, có nghĩa là toàn bộ toán hạng 8 bit đều được dùng cho bộ lớn. Có nhiều ứng dụng yêu cầu dữ liệu có dấu, phần này sẽ bàn về những lệnh liên quan đến các số có dấu.

6.3.1 Khái niệm về các số có dấu trong máy tính.

Trong cuộc sống hàng ngày các số được dùng có thể là số 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. Các máy tính cũng phải có khả năng đáp ứng phù hợp với các số ấy. Để làm được điều ấy các nhà khoa học máy tính đã phát minh ra sự xắp xếp biểu diễn các số âm có dấu và số dương có dấu như sau: Bit cao nhất MSB được để dành cho bit dấu (+) hoặc (-), còn các bit còn lại được dùng biểu diễn độ lớn. Dấu được biểu diễn bởi 0 đối với các số dương và một số đối với các số âm (-). Biểu diễn của một byte có dấu được trình bày trên hình 6.2.



Hình 6.2: Các toán hạng 8 bit có dấu.

a- Các toán hạng 8 bit có dấu: Trong các toán hạng A byte có dấu thì bit cao nhất MSB là D7 được dùng để biểu diễn dấu, còn 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 nó là toán hạng âm.

b- Các số dương: Dải của các số dương có thể được biểu diễn theo dạng cho trên hình 6.2 là từ 0 đến +127 thì phải sử dụng toán hạng 16 bit. Vì 8051 không hỗ trợ dữ liệu 16 bit nên ta không bàn luận đến.
c- Các số âm: Đối với các số âm thì D7 = 1, tuy nhiên độ lớn được biểu diễn ở dạng số bù 2 của nó. Mặc dù hợp ngữ thực hiện việc chuyển đổi song điều quan trọng là hiểu việc chuyển đổi diễn ra như thế nào. Để chuyển đổi về dạng biểu diễn số âm (bù 2) thì tiến hành theo các bước sau:

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

Ví dụ 6.9: Hãy trình bày cách 8051 biểu diễn số - 5.
Lời giải:

Hãy quan sát các bước sau:

0000	0101	Biểu diễn số 5 ở dạng 8 bit nhị phân
1111	1010	Đảo các bit
1111	1011	Cộng (thành số FB ở dạng Hex)

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 - 34H.
Lời giải:

Hãy quan sát các bước sau:

0011	0200	Số 34 được cho ở dạng nhị phân
1100	1011	Đảo các bit
1100	1100	Cộng 1 (thành số CC ở dạng Hex)

Vậy số CCH là biểu diễn dạng bù 2 có dấu của - 34H.
Ví dụ 6.11: Trình bày cách 8051 biểu diễn - 128:
Lời giải:

Quan sát các bước sau:

1000	0000	Số 128 ở dạng nhị phân 28 bit
0111	1111	Đảo các bit
1000	0000	Cộng 1 (trở thành số 80 dạng Hex)

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

Từ các ví dụ trên đây ta thấy rõ ràng rằng dải của các số âm có dấu 8 bit là - 1 đến - 128. Dưới đây là liệt kê các số có dấu 8 bit:

Số thập phân	Số nhị phân	Số Hex
-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

6.3.2 Vấn đề tràn trong các phép toán với số có dấu.

Khi sử dụng các số có dấu xuất hiện một vấn đề rất nghiêm trọng mà phải được sử lý. Đó là vấn đề tràn, 8051 báo có lỗi bằng cách thiết lập cờ tràn OV nhưng trách nhiệm của lập trình viên là phải cẩn thận với kết quả sai. CPU chỉ hiểu 0 và 1 và nó làm ngơ với việc chuyển đổi số âm, số dương của con người. Vậy tràn số là gì? Nếu kết quả của một phép toán trên các số có dấu mà quá lớn đối với thanh ghi thì xuất hiện sự tràn số và lập trình viên phải được cảnh báo. Xét ví dụ 6.12 dưới đây.

Ví dụ 6.12:

Khảo sát đoạn mã 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 = - 90
                          Sai !!!
```

Lời giải:

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

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

Trong ví dụ 6.12 thì + 96 được cộng với + 70 và kết quả theo CPU là - 90. Tại sao vậy? Lý do là kết quả của + 96 + 70 = 172 lớn hơn số mà thanh ghi A có thể chứa được. Cũng như tất cả mọi thanh ghi 8 bit khác, thanh ghi A chỉ chứa được đến số + 127. Các nhà thiết kế của PCU tạo ra cờ tràn OV phục vụ riêng cho mục đích báo cho lập trình viên rằng kết quả của phép toán số có dấu là sai.

6.3.3 Khi nào thì cờ tràn OV được thiết lập?

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

Hay nói cách khác là 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. Điều này có nghĩa là nếu có nhớ cả từ D6 sang D7 và từ D7 ra thì cờ OV = 0. Trong ví dụ 6.12 vì chỉ có nhớ từ D7 ra nên cờ OV = 1. Trong ví dụ 6.13, ví dụ 6.14 và 6.15 có minh hoạ thêm về sử dụng cờ tràn trong các phép số học với số có dấu.

Ví dụ 6.13:

Hãy quan sát đoạn mã sau 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)
```

Lời giải:

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

Theo CPU thì kết quả + 126 là kết quả sai, nên cờ OV = 1.

Ví dụ 6.14:

Hãy quan sát đoạn mã 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, correct, OV = 0)
```

Lời giải:

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

Theo CPU thì kết quả - 7 là đúng nên cờ OV = 0.

Ví dụ 6.15:

Theo dõi đoạn mã sau, chú ý vai trò của cờ OV.

```
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, correct, OV = 0)
```

Lời giải:

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

Theo CPU thì kết quả - 25 là đúng nên cờ OV = 0.

Từ các ví dụ trên đây ta có thể kết luận rằng trọng bất kỳ phép cộng số có dấu nào, cờ OV đều báo kết quả là đúng hay sai. Nếu cờ OV = 1 thì kết quả là sai, còn nếu OV = 0 thì kết quả là đúng. Chúng ta có thể nhấn mạnh rằng, trong phép cộng các số không dấu ta phải hiển thị trạng thái của cờ CY (cờ nhớ) và trong phép cộng các số có dấu thì cờ tràn OV phải được theo dõi bởi lập trình viên. Trong 8051 thì các lệnh như JNC và JC cho phép chương trình rẽ nhánh ngay sau phép cộng các số không dấu như ở phần 6.1. Đối với cờ tràn OV thì không có như vậy. Tuy nhiên,

điều này có thể đạt được bằng lệnh “JB PSW.2” hoặc “JNB PSW.2” vì PSW thanh ghi cờ có thể đánh địa chỉ theo bit.

CHƯƠNG 7
Các lệnh lô - gíc và các chương trình

7.1 Các lệnh lô-gíc và so sánh.

7.1.1 Lệnh VÀ (AND).

Cú pháp: ANL đích, nguồn; đích = đích Và nguồn (kẻ bảng).

Lệnh này sẽ thực hiện một phép Và lô-gíc trên hai toán hạng đích và nguồn và đặt kết quả vào đích. Đích thường là thanh ghi tổng (tích luỹ). Toán hạngnguồn có thể là thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. Hãy xem phụ lục Appendix A1 để biết thêm về các chế độ đánh địa chỉ dành cho lệnh này. Lệnh ANL đối với toán hạng theo byte không có tác động lên các cờ. Nó thường được dùng để che (đặt về 0) những bit nhất định của một toán hạng. Xem ví dụ 7.1.

Ví dụ:

Trình bày kết quả của các lệnh sau:

```
MOV   A, #35H          ; Gán A = 35H
ANL   A, #0FH          ; Thực hiện Và lô-gíc A và 0FH (Bây giờ A = 05)
```

Lời giải:

```
35H   0  0  1  1  0  1  0  1
0FH   0  0  0  0  1  1  1  1
05H   0  0  0  0  0  1  0  1          35H  và 0FH = 05H
```

7.1.2: Lệnh HOẶC (OR).

Cú pháp ORL đích = đích Hoặc nguồn (kẻ bảng)

Các toán hạng đích và nguồn được Hoặc với nhau và kết quả được đặt vào đích. Phép Hoặc có thể được dùng để thiết lập những bit nhất định của một toán hạng 1. Đích thường là thanh ghi tổng, toán hạng nguồn có thể là một thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. Hãy tham khảo phụ lục Appendix A để biết thêm về các chế độ đánh địa chỉ được hỗ trợ bởi lệnh này. Lệnh ORL đối với các toán hạng đánh địa chỉ theo byte sẽ không có tác động đến bất kỳ cờ nào. Xem ví dụ 7.2.

Ví dụ 7.2: Trình bày kết quả của đoạn mã sau:

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

Lời giải:

```
04H   0000  0100
68H   0110  1000
6CH   0110  1100          04  OR  68 = 6CH
```

7.1.3 Lệnh XOR (OR loại trừ?).

Cú pháp: XRL đích, nguồn; đích = đích Hoặc loại trừ nguồn (kẻ bảng).

Lệnh này sẽ thực hiện phép XOR trên hai toán hạng và đặt kết quả vào đích. Đích thường là thanh ghi tổng. Toán hạng nguồn có thể là một thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. Xem phụ lục Appendix A.1 để biết thêm về chế độ đánh địa chỉ của lệnh này. Lệnh XRL đối với các toán hạng đánh địa chỉ theo byte sẽ không có tác động đến bất kỳ cờ nào. Xét ví dụ 7.3 và 7.4.

Ví dụ 7.3: Trình bày kết quả của đoạn mã sau:

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

Lời giải:

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

Ví dụ 7.4:

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

Lời giải:

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

Lệnh XRL cũng có thể được dùng để xem nếu hai thanh ghi có giá trị giống nhau không? Lệnh "XRL A, R1" sẽ hoặc loại trừ với thanh ghi R1 và đặt kết quả vào A. Nếu cả hai thanh ghi có cùng giá trị thì trong A sẽ là 00. Sau đó có thể dùng lệnh nhả JZ để thực hiện theo kết quả. Xét ví dụ 7.5.

Ví dụ 7.5:

Đọc và kiểm tra cổng P1 xem nó có chứa giá trị 45H không? Nếu có gửi 99H đến cổng P2, nếu không xoá nó.

Lời 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: ...
```

Trong chương trình của ví dụ 7.5 lưu ý việc sử dụng lệnh nhảy JNZ. Lệnh JNZ và JZ kiểm tra các nội dung chỉ của thanh ghi tổng. Hay nói cách khác là trong 8051không có cờ 0.

Một ứng dụng rộng rãi khác của bộ xử lý là chọn các bit của một toán hạng. Ví dụ để chọn 2 bit của thnh ghi A ta có thể sử dụng mã sau. Mã này ép bit D2 của thanh ghi A chuyển sang giá trị nghịch đảo, còn các bit khác không thay đổi.

```
XRL    A, #04H          ; Nghĩa hoặc loại trừ thanh ghi A với
                        ; Giá trị 0000 0100
```

7.1.4 Lệnh bù thanh ghi tổng CPL A.

Lệnh này bù nội dung của thanh ghi tổng A. Phép bù là phép biến đổi các số 0 thành các số 1 và đổi các số 1 sang số 0. Đây cũng còn được gọi là phép bù 1.

```
MOV     A, #55H
CPL     A           ; Bây giờ nội dung của thanh ghi A là AAH
                    ; Vì 0101 0101 (55H) → 1010 1010 (AAH)
```

Để nhận được kết quả bù 2 thì tất cả mọi việc ta cần phải làm là cộng 1 vào kết quả bù 1. Trong 8051 thì không có lệnh bù 2 nào cả. Lưu ý rằng trong khi bù một byte thì dữ liệu phải ở trong thanh ghi A. Lệnh CPL không hỗ trợ một chế độ đánh địa chỉ nào cả. Xem ví dụ 7.6 dưới đây.

Ví dụ 7.6: Tìm giá trị bù 2 của 85H.

Lời 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           ; Cổng 1 vào A thành bù 2 A = 0111 1011 (7BH)
```

Ví dụ 7.1.5 Lệnh so sánh.

8051 có một lệnh cho phép so sánh. Nó có cú pháp như sau:

CJNE đích, nguồn, địa chỉ tương đối.

Trong 8051 thì phép so sánh và nhảy được kết hợp thành môt lệnh có tên là CJNE (so sánh và nhảy nếu kết quả không bằng nhau). 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 nó thay đổi cờ nhớ CY để báo nếu toán hạng đích lớn hơn hay nhỏ hơn. Điều quan trọng cần để là các toán hạng vẫn không giữ nguyên không thay thay đổi. Ví dụ, sau khi thực hiện lệnh "CJNE A, #67H, NEXT" thì thanh ghi A vẫn có giá trị ban đầu của nó (giá trị trước lệnh CJNE). Lệnh này so sánh nội dung thanh ghi A với giá trị 67H và nhảy đến giá trị đích NEXT chỉ khi thanh ghi A có giá trị khác 67H.

Ví dụ 7.7:

Xét đoạn mã dưới đây sau đó trả lời câu hỏi:

a) Nó sẽ nhảy đến NEXT không?

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

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

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

Trong lệnh CJNE thì toán hạng đích có thể trong thanh ghi tổng hoặc trong một các thanh ghi Rn. Toán hạng nguồn có thể trong một thanh ghi, trong bộ nhớ hoặc giá trị cho sẵn. Hãy xem phụ lục Appendix A để biết thêm chi tiết về các chế độ đánh địa chỉ cho lệnh này. Lệnh này chỉ tác động cờ nhớ CY. Cờ này được thay đổi như chỉ ra trên bảng 7.1. Dưới đây trình bày phép so sánh hoạt động như thế nào đối với tất cả các điều kiện có thể:

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

Bảng 7.1: Thiết kế cờ CY cho lệnh CJNE.

Compare	**Carry Flag**
Destinatio > Source	CY = 0
Destination < Source	CY = 1

Để ý rằng trong lệnh CJNE thì không có thanh ghi Rn nào có thể được so sánh với giá trị cho sẵn. Do vậy không cần phải nói đến thanh ghi A. Cũng cần lưu ý rằng cờ nhớ CY luôn được kiểm tra để xem lớn hơn hay nhỏ hơn, nhưng chỉ khi đã xác định là nó không bằng nhau. Xét ví dụ 7.8 và 7.9 dưới đây.

Ví dụ 7.8:

Hãy viết mã xác định xem thanh ghi A có chứa giá trị 99H không? Nếu có thì hãy tạo R1 = FFH còn nếu không tạo R1 = 0.

Lời giải:

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

Ví dụ 7.9:

Giả sử P1 là một cổng đầu vào được nối tới một cảm biến nhiệt. Hãy viết chương trình đọc nhiệt độ và kiểm tra nó đối với giá trị 75. Theo kết quả kiểm tra hãy đặt giá trị nhiệt độ vào các thanh ghi được chỉ định như sau:

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

Lời 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:   JNC   NEXT              ; Nếu CY = 0 thì A > 75 nhảy đến NEXT
        MOV   R1, A             ; Nếu CY = 1 thì A < 75 lưu vào R1
        SJMP  EXIT              ; Và thoát
NEXT:   MOV   R2, A             ; A > 75 lưu nó vào R2
EXIT:   ...
```

Lệnh so sánh thực sự là một phép trừ, ngoại trừ một điều là giá trị của các toán hạng không thay đổi. Các cờ được thay đổi tuỳ theo việc thực hiện lệnh trừ SUBB. Cần phải được nhấn mạnh lại rằng, trong lệnh CJNE các toán hạng không bị tác động bất kể kết quả so sánh là như thế nào. Chỉ có cờ CY là bị tác động, điều này bị chi phối bởi thực tế là lệnh CJNE sử dụng phép trừ để bật và xoá cờ CY.

Ví dụ 7.10:

Viết một chương trình để hiển thị liên tục cổng P1 đối với giá trị 63H. Nó chỉ mất hiển thị khi P1 = 63H.

Lời 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 hiển thị trừ khi P1 = 63H
```

Ví dụ 7.11:

Giả sử các ngăn nhớ của RAM trong 40H - 44H chứa nhiệt độ hàng ngày của 5 ngày như được chỉ ra dưới đây. Hãy tìm để xem có giá trị nào bằng 65 không? Nếu giá trị 65 có trong bảng hãy đặt ngăn nhớ của nó vào R4 nếu không thì đặt R4 = 0.

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

Lời 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          ; Tiếp tục kiểm tra cho đến khi bộ đếm bằng 0.
EXIT:   ...
```

7.2 Các lệnh quay vào trao đổi.

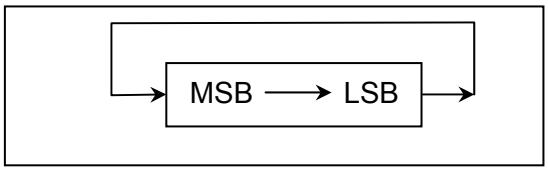
Trong rất nhiều ứng dụng cần phải thực hiện phép quay bit của một toán hạng. Các lệnh quay 8051 là R1, RR, RLC và RRC được thiết kế đặc biệt cho mục đích này. Chúng cho phép một chương trình quay thanh ghi tổng sang trái hoặc phải. Trong 8051 để quay một byte thì toán hạng phải ở trong thanh ghi tổng A. Có hai kiểu quay là: Quay đơn giản các bit của thanh ghi A và quay qua cờ nhớ (hay quay có nhớ).

7.2.1 Quay các bit của thanh ghi A sang trái hoặc phải.

a) Quay phải: RR A ; Quay các bit thanh ghi A sang phải.

Trong phép quay phải, 8 bit của thanh ghi tổng được quay sang phải một bit và bit D0 rời từ vị trí bit thấp nhất và chuyển sang bit cao nhất D7. Xem đoạn mã dưới đây.

```
MOV   A, #36H          ; A = 0011 011
RR    A                ; A = 0001 1011
RR    A                ; A = 1000 1101
RR    A                ; A = 1100 0110
RR    A                ; A = 0110 0011
```



b) Quay trái:

Cú pháp: RL A ; Quay trái các bit của thanh ghi A (hình vẽ)

Trong phép quay trái thì 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 biểu đồ mã dưới đây.

```
MOV   A, #72H          ; A = 0111 001
RL    A                ; A = 1110 0100
RL    A                ; A = 1100 1001
```

Lưu ý rằng trong các lệnh RR và RL thì không có cờ nào bị tác động.

7.2.2 Quay có nhớ.

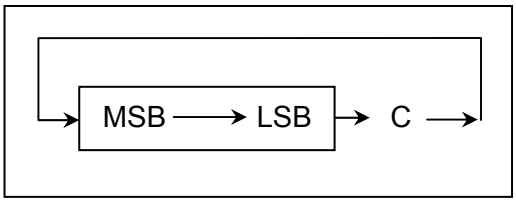
Trong 8051 còn có 2 kênh quay nữa là quay phải có nhớ và quay trái có nhớ.

Cú pháp: RRC A và RLC A

a) Quay phải có nhớ: RRC A

Trong quay phải có nhớ thì các bit của thanh ghi A được quay từ trái sang phải 1 bit và bit thấp nhất được đưa vào cờ nhớ CY và sau đó cờ CY được đưa vào vị trí bit cao nhất. Hay nói cách khác, trong phép RRC A thì LSB được chuyển vào CY và CY được chuyển vào MSB. Trong thực tế thì cờ nhớ CY tác động như là một bit bộ phận của thanh ghi A làm nó trở thành thanh ghi 9 bit.

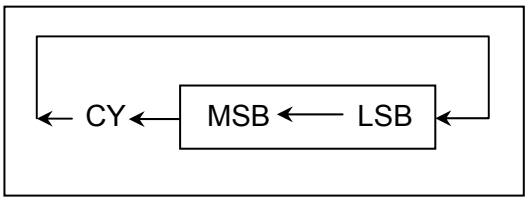
```
CLR   C                ; make CY = 0
MOV   A #26H           ; A = 0010 0110
RRC   A                ; A = 0001 0011  CY = 0
RRC   A                ; A = 0000 1001  CY = 1
RCC   A                ; A = 1000 0100  CY = 1
```



b) Quay trái có nhớ (hình vẽ): RLC A.

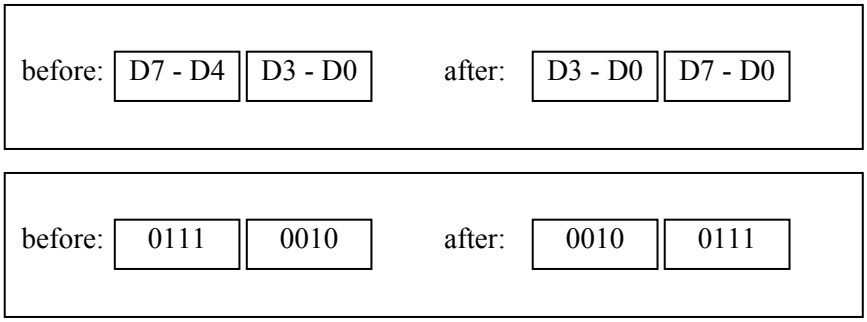
Trong RLC A thì các bit được dịch phải một bit và đẩy bit MSB vào cờ nhớ CY, sau đó CY được chuyển vào bit LSB. Hay nói cách khác, trong RLC thì bit MSB được chuyển vào CY và CY được chuyển vào LSB. Hãy xem đoạn mã sau.

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



7.2.3 Lệnh trao đổi thanh ghi A: SWAP A

Một lệnh hữu ích khác nữa là lệnh trao đổi SWAP. Nó chỉ hoạt động trên thanh ghi A, nó trao đổi nửa phần cao của byte và nửa phần thấp của byte với nhau. Hay nói cách khác 4 bit cao được chuyển thành 4 bit thấp và 4 bit thấp thành 4 bit cao.



Ví dụ 7.12:

a) Hãy tìm nội dung của thanh ghi A ở đoạn mã sau.
b) Trong trường hợp không có lệnh SWAP thì cần phải làm như thế nào để trao đổi những bit này? Hãy viết một mã chương trình đơn giản về quá trình đó.

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

Lời giải:

```
        MOV   R1, #0        ; Chọn R1 giữ 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 1 byte dữ liệu nói tiếp thì dữ liệu có thể được chuyển đổi từ song song sang nối tiếp bằng các lệnh quay như sau:

```
RRC   A           ; Bít 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 mã trên đây là một phương pháp được sử dụng rộng rãi trong truyền dữ liệu tới các bộ nhớ nối tiếp như các EEPROM nối tiếp.

7.3 Các chương trình ứng dụng của mã BCD và ASCII.

Các số mã BCD đã được trình ở chương 6. Như đã nói ở đó rằng trong rất nhiều bộ vi điều khiển mới đều có một đồng hồ thời gian thực RTC (Real Time Clock) để giữ cho thời gian và cả lịch cho cả khi bị tắt nguồn. Các bộ vi điều khiển này cung cấp thời gian và lịch dưới dạng BCD. Tuy nhiên, để hiển thị chúng thì chúng phải được chuyển về mã ASCII. Trong phần này ta trình bày ứng dụng của các lệnh quay và các lệnh lô-gíc trong việc chuyển đổi mã BCD và ASCII.

Bảng 7.2: Mã ASCII cho các chữ số từ 0- 9.

Phím	Mã ASCII (Hex)	Mã ASCII nhị phân	Mã BCD (không đóng gói)
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	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

7.3.1 Các số mã ASCII.

Trên các bàn phím ASCII khi phím "0" được kích hoạt thì "011 0001" (30H) được cấp tới máy tính. Tương tự như vậy 31H (011 0001) được cấp cho phím "1" v.v... như cyhỉ ra trong bảng 7.2.

Cần phải ghi nhớ rằng mặc dù mã ASCII là chuẩn ở mỹ (và nhiều quốc gia khác) nhưng các số mã BCD là tổng quát. Vì bàn phìm, máy in và màn hình đều sử dụng mã ASCII nên cần phải thực hiện đổi chuyển giữa các số mã ASCII về số mã BCD và ngược lại.

7.3.2 Chuyển đổi mã BCD đóng gói về ASCII.

Các bộ vi điều khiển DS5000T đều có đồng bộ thời gian thực RTC. Nó cung cấp hiển thị 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 cấp ở dạng mã BCD đóng gói. Để hiển thị dữ liệu này trên một LCD hoặc in ra trên máy in thì nó phải được chuyển về dạng mã ASCII.

Để chuyển đổi mã BCD đóng gói về mã ASCII thì trước hết nó phải được chuyển đổi thành mã BCD không đóng gói. Sau đó mã BCD chưa đóng gói được móc với 011 0000 (30H). Dưới đây minh hoạ việc chuyển đổi từ mã BCD đóng gói về mã ASCII. Xem ví dụ 7.14.

Mã BCD đóng gói	Mã BCD không đóng gói	Mã ASCII
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 & 0000 1001	0011 0010 & 0011 1001

7.3.3 Chuyển đổi mã ASCII về mã BCD đóng gói.

Để chuyển đổi mã ASCII về BCD đóng gói trước thì trước hết nó phải được chuyển về mã BCD không đóng gói (để có thêm 3 số) và sau đó được kết hợp để tạo ra mã BCD đóng gói. 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 đóng gói. Qúa trình này như sau:

Phím	Mã ASCII	Mã BCD không đóng gói	Mã BCD đóng gói
4	34	0000 0100	
7	37	0000 0111	0100 0111 hay 47H

```
MOV     A, # '4'        ; Gán A = 34H mã ASCII của số 4
MOV     R1, # '7'           ; Gán R1 = 37H mã ASCII của số 4
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 đóng gói
```

Sau phép chuyển đổi này các số BCD đóng gói được xử lý và kết quả sẽ là dạng BCD đóng gói. Như ta đã biết ở chương 6 có một lệnh đặt biệt là "DA A" đòi hỏi dữ liệu phải ở dạng BCD đóng gói.

Ví dụ 7.14:

Giả sử thanh ghi A có số mã BCD đóng gói hãy viết một chương trình để chuyển đổi mã BCD về hai số ASCII và đặt chúng vào R2 và R6.

Lời giải:

```
MOV     A, #29H             ; Gán A = 29, mã BCD đóng gói
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             ; Tạo nó thành mã ASCII A = 39H (số
9)
MOV     R6, A           ; Lưu nó vào R6 (R6 = 39H ký tự của ASCII)
MOV     A, R2           ; Lấy lại 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                 ; Tạo nó thành mã ASCII (A = 32H, số
2)
        MOV     R2, A           ; Lưu ký tự ASCII vào R2
```

Trong ví dụ trên tất nhiên là ta có thể thay 4 lệnh RR quay phải bằng một lệnh trao đổi WAPA.

CHƯƠNG 8
Các lệnh một bít và lập trình

8.1 Lập trình với các lệnh một bít.

Trong hầu hết các bộ vi xử lý (BVXL) thì dữ liệu được truy cập theo từng byte. Trong các bộ vi xử lýnh địa chỉ theo byte này thì các nội dung của một thanh ghi, bộ nhớ RAM hay cổng đều phải được truy cập từng byte một. Hay nói cách khác, lượng dữ liệu tối thiểu có thể được truy cập là một byte. Ví dụ, trong bộ vi xử lý Pentium cổng vào/ ra (I/O) được định hướng theo byte, có nghĩa là để thay đổi một bít thì ta phải truy cập toàn bộ 8 bít. Trong khi đó có rất nhiều ứng dụng thì ta phải chỉ cần thay đổi giá trị của một bít chẳng hạn như là bật hoặc tắt một thiết bị. Do vậy khả năng đánh địa chỉ đến từng bít của 8051 rất thích hợp cho ứng dụng này. Khả năng truy cập đến từng bít một thay vì phải truy cập cả byte làm cho 805 trở thành trong những bộ vi điều khiển (BVĐK) 8 bít mạnh nhất trên thị trường. Vậy những bộ phận nào của CPU, RAM, các thanh ghi, cổng I/O hoặc ROM là có thể đánh địa chỉ theo bít được. Vì ROM chỉ đơn giản dữ mã chương trình thực thi nên nó không cần khả năng đánh địa chỉ theo bít. Tất cả mọi mã lệnh đều định hướng theo byte chỉ có các thanh ghi, RAM và các cổng I/O là cần được đánh địa chỉ theo bít. Trong 8051 thì rất nhiều vị trí của RAM trong một số thanh ghi và tất cả các cổng I/O là có thể đánh địa chỉ theo từng bít. Dưới đây ta chỉ đi sâu vào từng phần một.

8.1.1 Các lệnh một bít.

Các lệnh dùng các phép tính một bít được cho ở bảng 8.1. Trong phần này chúng ta làm về các lệnh này và đưa ra nhiều ví dụ về cách sử dụng chúng, các lệnh một bít khác mà chỉ liên quan đến cờ nhớ CY (Cary Flag) sẽ làm ở mục khác.

Bảng 8.1: Các lệnh một bít của 8051

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

8.1.2 Các cổng I/O và khả năng đánh địa chỉ theo bít.

Bộ vi điều khiển 8051 có bốn cổng I/O 8 bít là P0, P1, P2 và P3. Chúng ta có thể truy cập toàn bộ 8 bít hoặc theo một bít bất kỳ mà không làm thay đổi các bít khác còn lại. Khi truy cập một cổng theo từng bít, chúng ta sử dụng các cú pháp "SETB Y, Y" với X là số của cổng 0, 1, 2 hoặc 3, còn Y là vị trí bít từ 0 đến 7 đối với các bít dữ liệu đo đến 7. Ví dụ "SETB P1.5" là thiết lập bít cao số 5 của cổng 1. Hãy nhớ rằng D0 là bít có nghĩa thấp nhất LSB và D7 là bít có nghĩa là cao nhất MSB. Xem ví dụ 8.1.

Ví dụ 8.1: Viết các chương trình sau:

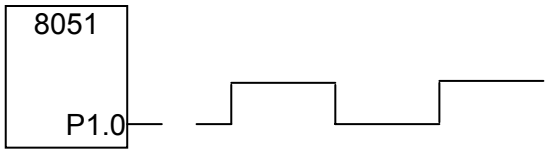
a) Tạo một sóng vuông (hàm xung vuông) với độ đầy xung 50% trên bít 0 của cổng 1.

b) Tạo một hàm xung vuông với 66% độ đầy xung trên bít 3 của cổng 1.

Lời 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 chốt P1.0 với thời gian giữ chậm giữa các trạng thái.

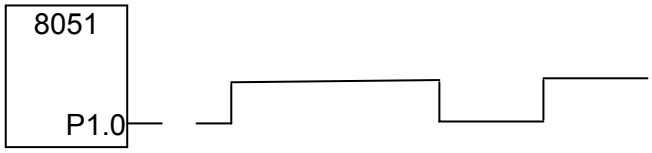
```
HERE:    SETB  P1.0        ;Thiết lập bít 0 cổng 1 lên 1.
         LCALL DELAY       ;Gọi chương trình con giữ chậm DELAY
         CLR P1.0          ;P1.0 = 0
         SJMP HERE         ;Tiếp tục thực hiện nó.
                           Có thể viết chương trình này theo cách khác:
HERE:    CPL   P1.0        ;Bù bít 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 độ dài gấp đôi trạng thái "tắt".

```
BACK:    SETB  P1.3        ;Thiết lập bít 3 cổng 1 lên 1.
         LCALL DELAY       ;Gọi chương trình con DELAY
         LCALL DELAY       ;Gọi chương trình con DELAY lần nữa.
         CLR   P1.3        ;Xoá bít 3 của cổng 1 và 0.
         LCALL DELAY       ;Gọi chương trình con

         SJMP BACK         ;Tiếp tục thực hiện nó.
```



Lưu ý rằng, khi mã "P1.0" được hợp dịch nó trở thành "SETB 90H" vì P1.0 có địa chỉ trong RAM là 90h. Từ hình vẽ 8.1 ta thấy rằng các địa chỉ bít cho P0 là 80H đến 87H và cho P là 90H đến 97H v.v... Hình 8.1 cũng chỉ ra tất cả các thanh ghi có khả năng đánh địa chỉ theo bít.

Bảng 8.2: Khả năng đánh địa chỉ theo bít của các cổng.

P0	P1	P2	P3	Port's 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:

Đối với các lệnh dưới đây thì trạng thái của bít nào của SFR sẽ bị tác động (hãy sử dụng hình 8.1).

a) SETB 86H, b) CLR 87H, c) SETB 92H

b) SETB DA7H, e) CLR 0F2H, f) SETB OE7H

Lời giải

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

8.1.3 Kiểm tra một bít đầu vào.

Lệnh JNB (nhảy nếu bít = 0) và JB (nhảy nếu bít bằng 1) cũng là các phép thao tác đơn bít được sử dụng rộng rãi. Chúng cho phép ta hiển thị một bít và thực hiện quyết định phụ thuộc vào việc liệu nó là 0 hay là 1.

Ví dụ 8.3: Giả sử bít P2.3 là một đầu vào và biểu diễn điều kiện của một lô. Nếu nó bật lên 1 thì có nghĩa là lô nóng. Hãy hiển thị liên tục, mỗi khi nó lên cao thì hãy gửi một xung cao-xuống-thấp (Aigh-to-low) đến cổng P1.5 để bật còi báo.

Lời giải:

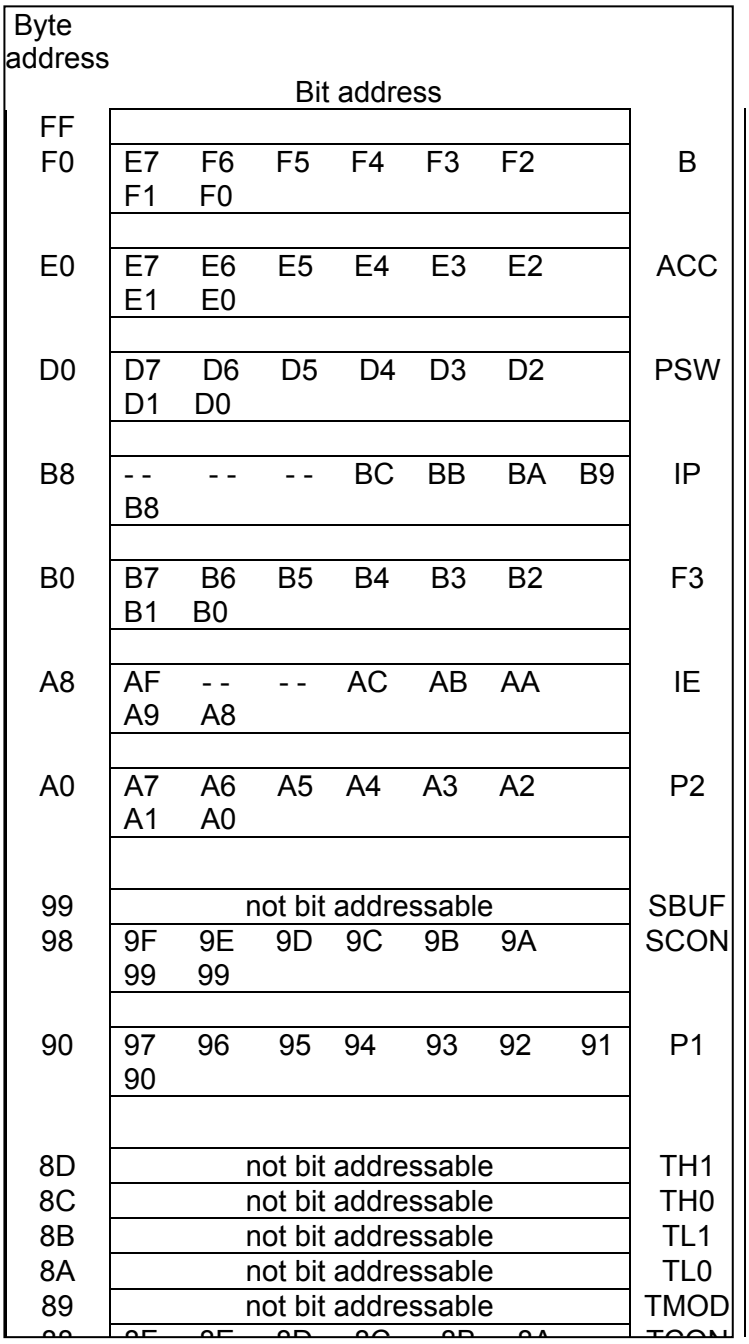
```
HERE:  JNB  P2.3, HERE   ; Duy trì hiển thị cao.
       SETB       P1.5          ; Thiết lập P1.5 = 1
       CLR P1.5          ; Thực hienẹ chuyển xung từ cao-xuống-thấp
```

Các lệnh JNB và JB có thể được dùng đối với các bít bất kỳ của các cổng I/O 0, 1, 2 và 3 vì tất cả các cổng này đều có khả năng đánh địa chỉ theo bít. Tuy nhiên, cổng 3 hầu như để dùng cho các tín hiệu ngắt và truyền thông nối tiếp và thông thường không dùng cho bất cứ vào/ ra theo bít hoặc theo byte nào. Điều này sẽ được bàn ở chương 10 và 11.

8.1.4 Các thanh ghi và khả năng đánh địa chỉ theo bít.

Trong tất cả các cổng I/O đều có khả năng đánh địa chỉ theo bít thì các thanh ghi lại không được như vậy. Ta có thể nhìn thấy điều đó từ hình 8.1: Chỉ thanh gh B, PSW, IP, IE, ACC, SCON và TCON là có thể đánh địa chỉ theo bít, ở đây ta sẽ tập trung vào các thanh ghi A, B và PSW còn các thanh ghi khác sẽ đề cập ở các chương sau. Từ hình 8.1 hãy để ý rằng cổng PO được gán địa chỉ bít 80H-87H. Còn đại chỉ bít 88-8FH được gán cho thanh ghi TCON.

Cuối cùng địa chỉ bít F0-F7H được gán cho thanh ghi B. Xét ví dụ 8.4 và 8.5 về việc sử dụng các thanh ghi này với khả năng đánh địa chỉ theo bít.



Byte address	Bit address							
FF								
F0	E7	F6	F5	F4	F3	F2		B
	F1	F0						
E0	E7	E6	E5	E4	E3	E2		ACC
	E1	E0						
D0	D7	D6	D5	D4	D3	D2		PSW
	D1	D0						
B8	- -	- -	- -	BC	BB	BA	B9	IP
	B8							
B0	B7	B6	B5	B4	B3	B2		F3
	B1	B0						
A8	AF	- -	- -	AC	AB	AA		IE
	A9	A8						
A0	A7	A6	A5	A4	A3	A2		P2
	A1	A0						
99	not bit addressable							SBUF
98	9F	9E	9D	9C	9B	9A		SCON
	99	99						
90	97	96	95	94	93	92	91	P1
	90							
8D	not bit addressable							TH1
8C	not bit addressable							TH0
8B	not bit addressable							TL1
8A	not bit addressable							TL0
89	not bit addressable							TMOD
88	8F	8E	8D	8C	8B	8A		TCON

Hình 8.1: Địa chỉa theo Byte và bít của bộ nhớ RAM các thanh ghi chức năng đặc biệt.

Ví dụ 8.4: Hãy viết chương trình để kiểm tra xem thanh ghi tích luỹ có chứa một số chẵn không? Nếu có thì chia nó cho 2, nếu không thì hãy làm chẵn nó và sau đó chia nó cho 2.

Lời giải:

```
        MOVB, # 2          ; Gán B = 2
        JNB  ACC 0, YES    ; DO của thanh ghi A có bằng 0?
        INC  A             ; Nếu có thì nhảy về YES
YES:    DiV  AB            ; Nếu là sô lẻ thì tăng lên 1 để thành chẵn
                           ; Chia A/B
```

Ví dụ 8.5: Hãy viết đoạn chương trình để kiểm tra xem các bít 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 bộ.
Lời giải:

```
        JNB  OFOH, NEXT-1          ; Nhảy về NEXT-1 nếu B.0 = 0
        SETB       OFOH            ; Đặt B.0 = 1
NEXT-1: JNB     OF5H, NEXT-2       ; Nhảy về NEXT-2 nếu B.5 = 0
        SETB       OF5H            ; Đặt B.5 = 1
NEXT-2: MOV     RO, B          ; Cắt thanh ghi B
```

CY	AC	- -	RS1	RS0	OV	- -	P

RS1	RS0	Register Bank	Address
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

Hình 8.2: Các bít của thanh ghi PSW.

Như đã nói ở chương 2, trong than ghi PSW có hai bít dành riêng để chọn các bảng thanh ghi. Khi RESET thì bằng 0 được chọn, chúng ta có thể chọn các băng bất kỳ khác bằng cách sử dụng khả năng đánh địa chỉ theo bít của PSW.
Ví dụ 8.6: Hãy viết chương trình để lưu thanh ghi tích luỹ vào R7 của băng 2.
Lời giải:

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

Ví dụ 8.7: Trong khi có hai lệnh JNC và JC để kiểm tra bít cờ nhớ CY thì lại không có các lệnh cho bít cờ tràn (0V) làm thế nào để ta có thể viết mã kiểm tra 0V.
Lời giải: Cờ 0V là bít PSW.2 của thanh ghi PSW. PSW là thanh ghi có thể đánh địa chỉ theo bít, do vậy ta có thể sử dụng lệnh sau để kiểm tra cờ 0V:

```
    JB      PSW.2, TARGET          ; Nhảy về TARGET nếu 0V = 1
```

8.15 Vùng nhớ RAM có thể đánh địa chỉ theo bít.

Trong 128 byte RAM trong của 8051 thì chỉ có 16 byte của nó là có thể đánh địa chỉ theo bít được. Phần còn lại được định dạng byte. Các cùng RAM có thể đánh địa chỉ theo bít là 20H đến 2FH. Với 16 byte này của RAM có thể cung cấp khả năng đánh địa chỉ theo bít là 128 bít, vì $16 \times 8 = 128$. Chúng được đánh địa chỉ từ 0 đến 127. Do vậy, những địa chỉ bít từ 0 đến 7 dành cho byte đầu tiên, vị trí RAM trong 20H và các bít từ 8 đến OFH là địa chỉ bít của byte thứ hai của vị trí RAM trong 21H v.v... Byte cuối cùng của 2FH có địa chỉ bít từ 78H đến 7FH (xem hình 8.3). Lưu ý rằng các vị trí RAM trong 20H đến 2FH vừa có thể đánh địa chỉ theo byte vừa có thể đánh địa chỉ theo bít.

Để ý từ hình 8.3 và 8.1 ta thấy rằng các địa chỉ bít 00 - 7FH thuộc về các địa chỉ byte của RAM từ 20 - 2FH và các địa chỉ bít 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 bít sau đây thuộc byte nào? Hãy cho địa chỉ của byte RAM ở dạng Hex.

```
a) SETB   42H          ; Set bit 42H to 1
b) CLR    67H          ; Clear bit 67
c) CLR    0FH          ; Clear bit OFH
d) SETB   28H          ; Set bit 28H to 1
e) CLR    12           ; Clear bit 12 (decimal)
f) SETB   05
```

Lời giải:

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

Ví dụ 8.9: Trạng thái của các bít P1.2 và P1.3 của cổng vào/ra P1 phải được lưu cất trước khi chúng được thay đổi. Hãy viết chương trình để lưu trạng thái của P1.2 vào vị trí bít 06 và trạng thái P1.3 vào vị trí bít 07.

Lời giải:

```
        CLR     06          ;Xoá địa chỉ bít 06
        CLR     07          ; Xoá địa chỉ bít 07
        JNB     P1.2, OVER  ;Kiểm tra bít P1.2 nhảy về OVER nếu P1.2 = 0
        SETB    06          ; Nếu P1.2 thì thiết lập vị trí bít 06 = 0
OVER:   JNB     P1.3, NEXT  ;Kiểm tra bít P1.3 nhảy về NEXT nếu nó = 0
        SETB    07          ;Nếu P1.3 = 1thì thiết lập vị trí bít 07 = 1
NEXT: ....
```

Các câu hỏi ôn luyện:

1. Tất cả các cổng I/O của 8051 đều có khả năng đánh địa chỉ theo bít? (đúng sai)
2. Tất cả mọi thanh ghi của 8051 đều có khả năng đánh địa chỉ theo bít? (đúng sai)
3. Tất cả các vị trí RAM của 8051 đều có khả năng đánh địa chỉ theo bít? (đúng sai)
4. Hãy chỉ ra những thanh ghi nào sau đây có khả năng đánh địa chỉ theo bít:
   a) A, b) B, (c) R4 (d) PSW (e) R7
5. Trong 128 byte RAM của 8051 những byte nào có khả năng đánh địa chỉ theo bít. Hãy liệt kê chúng.
6. Làm thế nào để có thể kiểm tra xem bít D0 của R3 là giá trị cao hay thấp.
7. Hãy tìm xem các bít dau thuộc những byte nào? Hãy cho địa chỉ của các byte RAM theo số Hex:
   a) SETB 20 b) CLR 32 c) SETB 12H
   d) SETB 95 e) SETB 0ETB 12H
8. Các địa chỉ bít 00 - 7FH và 80 - F7H thuộc các vị trí nhớ nào?
9. Các cổng P0, P1, P2 và P3 là một bộ phận của SFR? (đúng sai)

10. Thanh ghi TCON có thể đánh địa chỉ theo bít (đúng sai)

8.2 Các phép toán một bít với cờ nhớ CY.

Ngoài một thực tế là cờ nhớ CY được thay đổi bởi các lệnh lô-gíc và số học thì trong 8051 còn có một số lệnh mà có thể thao tác trực tiếp cờ nhớ CY. Các lệnh này được cho trong bảng 8.3.

Trong các lệnh được chỉ ra sau trong bảng 8.3 thì chúng ta đã trình bày công dụng của lệnh JNC, CLR và SETB trong nhiều ví dụ trong một số chương trước đây. Dưới đây ta tiếp tục làm quen với một số ví dụ về cách sư dụng một số lệnh khác từ bảng 8.3.

Một số lệnh cho trong bảng 8.3 làm việc với các phép toán lô-gíc AND và OR. Các ví dụ ở mục này sẽ chỉ ra cách sử dụng chúng như thế nào?

Ở chương tiếp theo chúng ta sẽ chỉ ra nhiều ví dụ hơn về việc sử dụng của các lệnh đơn trong phạm vi các ứng dụng thực tế.

Bảng 8.3: Các lệnh liên quan đến cờ nhớ CY

Lệnh	chức năng
SETB C	Thực hiện (tạo) CY = 1
CLR C	Xoá bít nhớ CY = 0
CPL C	Bù bít nhớ
MOV b, C	Sao chép trạng thái bít nhớ vào vị trí bít b = CY
MOV C, b	Sao chép bít b vào trạng thái bít nhớ CY = b
JNC đích	Nhảy tới đích nếu CY = 0
JC đích	Nhảy tới đích nếu CY = 1
ANL C. bít	Thực hiện phép AND với bít b và lưu vào CY
ANL C./ bít	Thực hiện phép AND với bít đảo và lưu vào CY
ORL C. bít	Thực hiện phép OR với bít và lưu vào CY
ORL C./ bít	Thực hiện phép OR với bít đảo và lưu vào CY

Ví dụ 8.10: Hãy viết một chương trình để lưu cất trạng thái của các bít P1.2 và P1.3 vào vị trí nhớ tương ứng trong RAM 6 và 7.

Lời giải:

```
MOV  C,  P1.2 ; Lưu trạng thái P1.2 vào CY.
MOV  06, C    ; Lưu trạng thái CY vào bít 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 vị trí nhớ 12H trong RAM giữ trạng thái của việc có điện thoại hay không. Nếu nó ở trạng thái cao có nghĩa là đã có một cuộc gọi mới vì nó được kiểm tra lần cuối. Hãy viết một chương trình để hiển thị "có lời nhắn mới" ("New Message") trên màn hình LCD nếu bít 12H của RAM có giá trị cao. Nếu nó có giá trị thấp thì LCD hiển thị "không có lời nhắn mới" ("No New Message").

Lời giải:

```
      MOV  C, 12H          ; Sao trạng thái bít 12H của RAM vào CY
      JNC   NO             ; Kiểm tra xem cờ CY có giá trị cao không.
      MOV  DPTR, # 400H    ; Nếu nó nạp địa chỉ của lời nhắn.
      LCAL DISPLAY                 ; Hiển thị lời nhắn.
      SJMP  NEXT           ; Thoát
NO:   MOV  DSTR, #420H     ; Nạp địa chỉ không có lời nhắn.
      LCAL DISPLAY                 ; Hiển thị nó.
```

```
EXIT:                       Thoát
;  ____________________     data to be displayed on LCD
            ORG  400H
YES-MG:     DB   "NEW Message"
            ORG  420H
NO-MG:      DB   "No New Message"
```

Ví dụ 8.12:

Giả sử rằng bít P2.2 được dùng để kiểm tra đèn ngoài và bít P2.5 dùng để kiểm tra đèn trong của một toà nhà. Hãy trình bày làm thế nào để bật đèn ngoài và tắt đèn trong nhà.

Lời giải:

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

Câu hỏi ôn luyện:

1. Tìm trạng thái của cờ CY sau đoạn mã dưới đây:

```
a) CLR   A          b) CLR C          c) CLR      C
   ADD   A, #OFFH      JNC  OVER          JC   OVER
   JWC   OVER          SETB C                  CPL  C
   CPL   C               OVER: ...        OVER: ...
OVER: ...
```

2. Hãy trình bày cách làm thế nào để lưu trạng thái bít P2.7 vào vị trí bít 31 của RAM.
3. Hãy trình bày các chuyển trạng thái bít 09 của RAM đến bít P1.4.

8.3 Đọc các chân đầu vào thông qua chốt cổng.

Trong việc đọc cổng thì một số lệnh đọc trạng thái của các chân cổng, còn một số lệnh khác thì đọc một số trạng thái của chốt cổng trong. Do vậy, khi đọc các cổng thì có hai khả năng:

1. Đọc trạng thái của chân vào.
2. Đọc chốt trong của cổng ra.

Chúng ta phải phân biệt giữa hai dạng lệnh này vì sự lẫn lộn giữa chúng là nguyên nhân chính của các lỗi trong lập trình cho 8051, đặc biệt khi đã kết nối với phần cứng bên ngoài. Trong phần này ta bàn về sơ qua các lệnh này. Tuy nhiên, độc giả phải nghiên cứu và hiểu về các nội dung của chủ đề này và về hoạt động bên trong của các cổng được cho trong phụ lục Appendix C2.

8.3.1 Các lệnh đọc cổng vào.

Như đã nói ở chương 4 thì để biến một bít bất kỳ của cổng 8051 nào đó thành một cổng đầu vào, chúng ta phải ghi (lô-gíc cao) vào bít đó. Ssu khi cấu hình các bít của cổng là đầu vào, ta có thể sử dụng những lệnh nhất định để nhận dữ liệu ngoài trên các chân vào trong CPU. Bảng 8.4 là những lệnh nói trên.

Bảng 8.4: Các lệnh đọc một cổng vào.

Giả lệnh	Ví dụ	Mô tả

MOV A, PX	MOV A, P2	Chuyển dữ liêuj 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

8.3.2 Đọc chốt cho cổng đầu ra.

Một số lệnh nội dung của một chốt cổng trong thay cho việc đọc trạng thái của một chân ngoài. Bảng 8.5 cung cấp danh sách những lệnh này. Ví dụ, xét lệnh "ANL P1, A". Trình tự thao tác được thực hiện bởi lệnh này như sau:

1. Nó đã chốt trong của một 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 kết luận 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 như là toán hạng đích hay nói cách khác, chúng ta chỉ được dùng cho các cổng được cấu hình như các cổng ra.

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

giả 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
DJN2 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

Lưu ý: Chúng ta nên nghiên cứu phần C2 của phụ lục Appendix C nếu ta nối phần cứng ngoài vào hệ 8051 của mình. Thực hiện sai các chỉ dẫn hoặc nối sai các chân có thể làm hỏng các cổng của hệ 8051.

8.4 Tóm lược.

Chương này đã mô tả một trong các đặc tính mạnh nhất của 8051 là phép toán một bít. Các phép toán một bít này cho phép lập trình viên thiết lập, xoá, di chuyển và bù các bít riêng rẽ của các cổng, bộ nhớ hoặc các thanh ghi.

Ngoài ra có một số lệnh cho phép thao tác trực tiếp với cờ nhớ CY. Chúng ta cũng đã bàn về các lệnh đọc các chân cổng thông qua việc đọc chốt cổng.

8.5 Các câu hỏi kiểm tra.

1. Các lệnh "SETB A", "CLR A", "CPL A" đúng hay sai?
2. Các cổng vào/ ra nào và các thanh ghi nào có thể đánh địa chỉ theo bít.
3. Các lệnh dưới đây đúng hay sai? Đánh dấu lệnh đúng.

a) SETB P1 e) SETB B4
b) SETB P2.3 f) CLR 80H
c) CLR ACC.5 g) CLR PSW.3
d) CRL 90H h) CLR 87H

4. Hãy giết chương trình tạo xung vuông với độ đầy xung 75%, 80% trên các chân P1.5 và P2.7 tương ứng.
5. Viết chương trình hiển thị P1.4 nếu nó có giá trị cao thì chương trình tạo ra một âm thanh (sóng dung vuông 50% độ đầy xung) trên chân P2.7.
6. Nhưng địa chỉ bít nào được gán cho các cổng P0, P1, P2 và P3 cho các thanh ghi PCON, A, B và PSW.
7. Những địa chỉ bít dưới đây thuộc về cổng hay thanh ghi nào?
a) 85H b)87H c) 88H d)8DH e)93H
f) A5H g)A7H h) B3H i) D4H j) D8H
8. Hãy viết chương trình lưu các thanh ghi A, B vào R3 và R5 băng nhớ 2 tương ứng.
9. Cho một lệnh khác cho "CLR C", so sánh chúng.
10. Làm thế nào để kiểm tra trạng thái các bít cờ OV, CY, P và AC. Hãy tìm địa chỉ bít của các cờ này.
11. Các cùng nhớ 128 byte của RAM thì những vùng nào là đánh địa chỉ theo bít được? Hãy đánh dấu chúng.
12. Các địa chỉ sau thuộc vùng RAM nào?
a) 05H b) 47 c) 18H d) 2DH e) 53H
g) 15H h) 67H h) 55H i) 14H k) 37FH
13. Các địa chỉ nhỏ hơn 80H được gán cho địa chỉ 20-2FH của RAM phải không? (Đúng/ sai).
14. Viết các lệnh để lưu cờ CY, AC, D vào vị trí bít 4, 16H và 12H tương ứng.
15. Viết chương trình kiểm tra D7 của thanh ghi A. Nếu D7 = 1 thì gửi thông báo sang LCD báo rằng ACC có một số âm.

CHƯƠNG 9
Lập trình cho bộ đếm/ bộ định thời trong 8051

8051 có hai bộ định thời/ bộ đếm. Chúng có thể được dùng như các bộ định thời để tạo một bộ trễ thời gian hoặc như các bộ đếm để đếm các sự kiện xảy ra bên ngoài bộ BVĐK. Trong chương này chúng ta sẽ tìm hiểu về cách lập trình cho chúng và sử dụng chúng như thế nào?

9.1 Lập trình các bộ định thời gian của 8051.

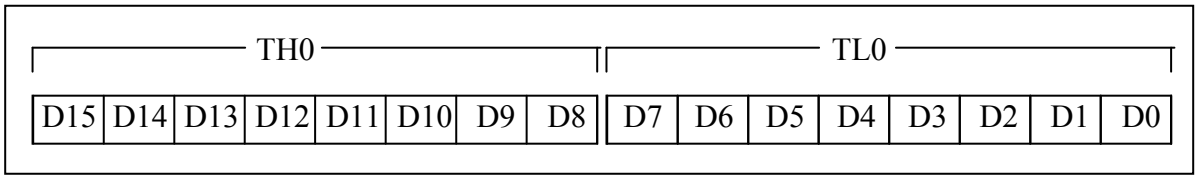
8051 có hai bộ định thời là Timer 0 và Timer1, ở phần này chúng ta bàn về các thanh ghi của chúng và sau đó trình bày cách lập trình chúng như thế nào để tạo ra các độ trễ thời gian.

9.1.1 Các thanh ghi cơ sở của bộ định thời.

Cả hai bộ định thời Timer 0 và Timer 1 đều có độ dài 16 bít được truy cập như hai thanh ghi tách biệt byte thấp và byte cao. Chúng ta sẽ bàn riêng về từng thanh ghi.

9.1.1.1 Các thanh ghi của bộ Timer 0.

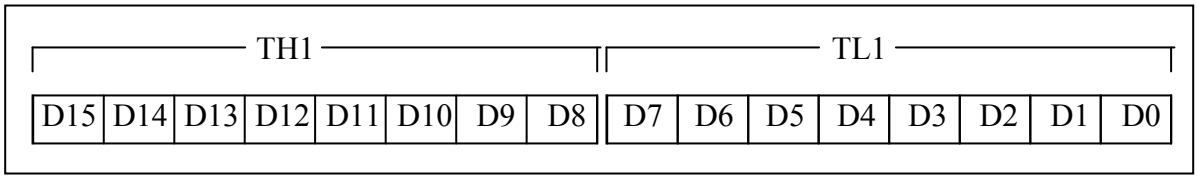
Thanh ghi 16 bít của bộ Timer 0 được truy cập như byte thấp và byte cao. Thanh ghi byte thấp được gọi là TL0 (Timer 0 bow byte) và thanh ghi byte cao là TH0 (Timer 0 High byte). Các thanh ghi này có thể được truy cập như mọi thanh ghi khác chẳng hạn như A, B, R0, R1, R2 v.v... Ví dụ, lệnh "MOV TL0, #4FH" 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: Các thanh ghi của bộ Timer 0.

9.1.1.2 Các thanh ghi của bộ Timer 1.

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



Hình 9.2: Các thanh ghi của bộ Timer 1.

9.1.2 Thanh ghi TMOD (chế độ của bộ định thời).

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

(MSB)							(MSB)
GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer1				Timer0			

Hình 9.3: Thanh ghi IMOD.

9.1.2.1 Các bít M1, M0:

Là các bít chế độ của các bộ Timer 0 và Timer 1. Chúng chọn chế độ của các bộ định thời: 0, 1, 2 và 3. Chế độ 0 là một bộ định thời 13, chế độ 1 là một bộ định thời 16 bít và chế độ 2 là bộ định thời 8 bít. Chúng ta chỉ tập chung vào các chế độ thường được sử dụng rộng rãi nhất là chế độ 1 và 2. Chúng ta sẽ sớm khám phá ra các đặc tính củ các chế độ này sau khi khám phần còn lại của thanh ghi TMOD. Các chế độ được thiết lập theo trạng thái của M1 và M0 như sau:

M1	M0	Chế độ	Chế độ hoạt động
0	0	0	Bộ định thời 13 bít gồm 8 bít là bộ định thời/ bộ đếm 5 bít đặt trước
0	1	1	Bộ định thời 16 bít (không có đặt trước)
1	0	2	Bộ định thời 8 bít tự nạp lại
1	1	3	Chế độ bộ định thời chia tách

9.1.2.2 C/ T (đồng hồ/ bộ định thời).

Bít này trong thanh ghi TMOD được dùng để quyết định xem bộ định thời được dùng như một máy tạo độ trễ hay bộ đếm sự kiện. Nếu bít C/T = 0 thì nó được dùng như một bộ định thời tạo độ chễ thời gian. Nguồn đồng hồ cho chế độ trễ thời gian là tần số thạch anh của 8051. Ở phần này chỉ bàn về lựa chọn này, công dụng của bộ định thời như bộ đếm sự kiện thì sẽ được bàn ở phần kế tiếp.

Ví dụ 9.1: Hãy hiển thị xem chế độ nào và bộ định thời nào đối với các trường hợp sau:

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

Lời giải: Chúng ta chuyển đổi giá trị từ số Hex sang nhị phân và đối chiếu hình 93 ta có:

a) TMOD = 0000 0001, chế độ 1 của bộ định thời Timer 0 được chọn.
b) TMOD = 0010 0000, chế độ 1 của bộ định thời Timer 1 được chọn.
c) TMOD = 0001 0010, chế độ 1 của bộ định thời Timer 0 và chế độ 1 của Timer 1 được chọn.

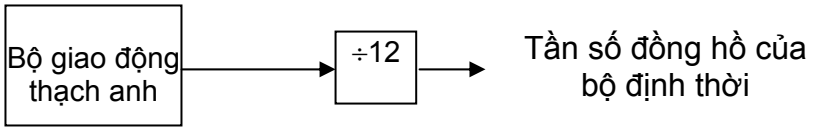
9.1.2.3 Nguồn xung đồng hồ cho bộ định thời:

Như chúng ta biết, mỗi bộ định thời cần một xung đồng hồ để giữ nhịp. Vậy nguồn xung đồng hồ cho các bộ định thời trên 8051 lấy ở đâu? Nếu C/T = 0 thì tần số thạch anh đi liền với 8051 được làm nguồn cho đồng hồ của bộ định thời. Điều đó có nghĩa là độ lớn của tần số thạch anh đi kèm với 8051 quyết định tốc độ nhịp của các bộ định thời trên 8051. Tần số của bộ định thời luôn bằng 1/12 tần số của thạch anh gắn với 8051. Xem ví dụ 9.2.

Ví dụ 9.2:

Hãy tìm tần số đồng bộ và chu kỳ của bộ định thời cho các hệ dựa trên 8051 với các tần số thạch anh sau:

a) 12MHz
b) 16MHz
c) 11,0592MHz



Lời giải:

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

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

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

Mặc dù các hệ thống dựa trên 8051 khác với tần số thạch anh từ 10 đến 40MHz, song ta chỉ tập chung vào tần số thạch anh 11,0592MHz. Lý do đằng sau một số lẻ như vậy là hải làm việc với tần suất bouid đối với truyền thông nối tiếp của 8051. Tần số XTAL = 11,0592MHz cho phép hệ 8051 truyền thông với IBM PC mà không có lỗi, điều mà ta sẽ biết ở chương 10.

9.1.3 Bít cổng GATE.

Một bít khác của thanh ghi TMOD là bít cổng GATE. Để ý trên hình 9.3 ta thấy cả hai bộ định thời Timer0 và Timer1 đều có bít GATE. Vậy bít GATE dùng để làm gì? Mỗi bộ định thời thực hiện điểm khởi động và dừng. Một số bộ định thời thực hiện điều này bằng phần mềm, một số khác bằng phần cứng và một số khác vừa bằng phần cứng vừa bằng phần mềm. Các bộ định thời tren 8051 có cả hai. Việc khởi động và dừng bộ định thời được khởi động bằng phần mềm bởi các bít khởi động bộ định thời TR là TR0 và TR1. Điều này có được nhờ các lệnh "SETB TR1" và "CLR TR1" đối với bộ Timer1 và "SETB TRO" và "CLR TR0" đối với bộ Timer0. Lệnh SETB khởi động bộ định thời và lệnh CLR dùng để dừng nó. Các lệnh này khởi động và dừng các bộ định thời khi bít GATE = 0 trong thanh ghi TMOD. Khởi động và ngừng bộ định thời bằng phần cứng từ nguồn ngoài bằng cách đặt bít GATE = 1 trong thanh ghi TMOD. Tuy nhiên, để tránh sự lẫn lộn ngay từ bây giờ ta đặt GATE = 0 có nghĩa là không cần khởi động và dừng các bộ định thời bằng phần cứng từ bên ngoài. Để sử dụng phần mềm để khởi động và dừng các bộ định thời phần mềm để khởi động và dừng các bộ định thời khi GATE = 0. Chúng ta chỉ cần các lệnh "SETB TRx" và "CLR TRx". Việc sử dụng phần cứng ngoài để khởi động và dừng bộ định thời ta sẽ bàn ở chương 11 khi bàn về các ngắt.

Ví dụ 9.3:

Tìm giá trị cho TMOD nếu ta muốn lập trình bộ Timer0 ở chế độ 2 sử dụng thạch anh XTAL 8051 làm nguồn đồng hồ và sử dụng các lệnh để khởi động và dừng bộ định thời.

Lời giải:

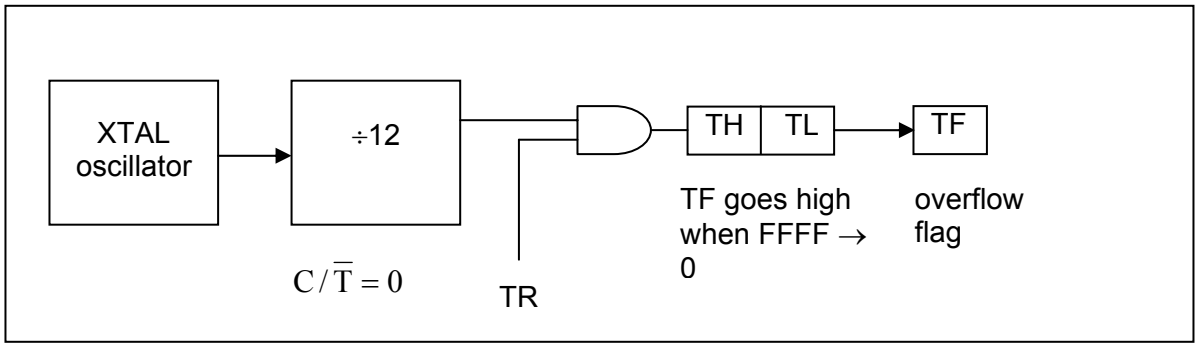
TMOD = 0000 0010: Bộ định thời Timer0, chế độ 2 C/T = 0 dùng nguồn XTAL GATE = 0 để dùng phần mềm trong để khởi động và dừng bộ định thời.

Như vậy, bây giờ chúng ta đã có hiểu biết cơ bản về vai trò của thanh ghi TMOD, chúng ta sẽ xét chế độ của bộ định thời và cách chúng được lập trình như thế nào để tạo ra một độ trễ thời gian. Do chế độ 1 và chế độ 2 được sử dụng rộng rãi nên ta đi xét chi tiết từng chế độ một.

9.1.4 Lập trình cho mỗi chế độ Mode1.

Dưới đây là những đặc tính và những phép toán của chế độ Mode1:

1. Nó là bộ định thời 16 bít, do vậy nó cho phép các giá trị 0000 đến FFFFH được nạp vào các thanh ghi TL và TH của bộ định thời.
2. Sau khi TL và TH được nạp một giá trị khởi tạo 16 bít thì bộ định thời phải được khởi động. Điều này được thực hiện bởi "SETB TR0" đối với Timer 0 và "SETB TR1" đối với Timer1.
3. Sau khi bộ định thời được khởi động, nó bắt đầu đếm lên. Nó đếm lên cho đến khi đạt được giới hạn FFFFH của nó. Khi nó quay qua từ FFFFH về 0000 thì nó bật lên bít cờ TF được gọi là cờ bộ định thời. Cờ bộ định thời này có thể được hiển thị. Khi cờ bộ định thời này được thiết lập từ một trong các phương án để dừng bộ định thời bằng các lệnh "CLR TR0" đối với Timer0 hoặc "CLR TR1" đối với Timer1. ở đây cũng cần phải nhắc lại là đối với bộ định thời đều có cờ TF riêng của mình: TF6 đối với Timer0 và TF1 đối với Timer1.



4. Sau khi bộ định thời đạt được giới hạn của nó và quay quan giá trị FFFFH, muốn lặp lại quá trình thì các thanh ghi TH và TL phải được nạp lại với giá trị ban đầu và TF phải được duy trì về 0.

9.1.4.1 Các bước lập trình ở chế độ Mode 1.

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

1. Nạp giá trị TMOD cho thanh ghi báo độ định thời nào (Timer0 hay Timer1) được sử dụng và chế độ nào được chọn.
2. Nạp các thanh ghi TL và TH với các giáa trị đếm ban đầu.
3. Khởi động bộ định thời.
4. Duy trì hiển thị cờ bộ định thời TF bằng lệnh "JNB TFx, đích" để xem nó được bật không. Thoát vòng lặp khi TF được 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 ta có thể phát triển một công thức tính toán độ trễ sử dụng chế độ Mode1 (16 bít) của bộ định thời đối với tần số thạch anh XTAL = 11, 0592MHz (xem hình 9.4). Máy tính trong thư mục Accessrry của Microsoft Windows có thể giúp ta tìm các giá trị TH và TL. Máy tính này hỗ trợ các phép tính theo số thập phân, nhị phân và thập lục.

a) Tính theo số Hex	b) Tính theo số thập phân
(FFFF - YYXX + 1). 1,085μs trong đó YYXX là các giá trị khởi tạo của TH, TL tương ứng. Lưu ý rằng các giá trị YYXX là theo số Hex.	Chuyển đổi các giá trị YYXX của TH, TL về số thập phân để nhận một số thập phân NNNNN sau đó lấy (65536 - NNNNN).1,085μs.

Hình 9.4: Công thức tính toán độ trễ thời gian đối với tần số XTAL = 11, 0592MHz.

Ví dụ 9.4:

Trong chương trình dưới đây ta tạo ra một sóng vuông với độ đầy xung 50% (cùng tỷ lệ giữa phần cao và phần thấp) trên chân P1.5. Bộ định thời Timer0 được dùng để tạo độ trễ thời gian. Hãy phân tích chương trình này.

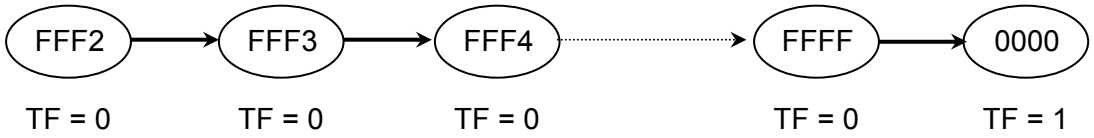
```
          MOV   TMOD, #01          ; Sử dụng Timer0 và chế độ 1(16 bít)
HERE:     MOV   TL0, #0F2H         ; TL0 = F2H, byte thấp
          MOV   TH0, #0FFH         ; TH0 = FFH, byte cao
          CPL   P1.5               ; Sử dụng chân P1.5
          ACALL DELAY
          SJMP  HERE               ; Nạp lại TH, TL
;_________ delay using timer0.
DELAY:
          SETB  TR0                ; Khởi động bộ định thời Timer0
AGAIN:    JNB   TF0, AGAIN         ; Hiển thị cờ bộ định thời cho đến khi nó vượt qua FFFFH.
          CLR   TR0                ; Dừng bộ Timer
          CLR   TF0                ; Xoá cờ bộ định thời 0
          RET
```

Lời giải:

Trong chương trình trên đây chú ý các bước sau:

1. TMOD được nạp.
2. Giá trị FFF2H được nạp và TH0 - TL0
3. Chân P1.5 được chọn dùng cho phần cao thấp của xung.
4. Chương trình con DELAY dùng bộ định thời được gọi.
5. Trong chương trình con DELAY bộ định thời Timer0 được khởi động bởi lệnh "SETB TR0"
6. Bộ Timer0 đếm lên với mỗi xung đồng hồ được cấp bởi máy phát thạch anh. Khi bộ định thời đếm tăng qua các trạng thái FFF3, FFF4 ... cho đến khi đạt giá trị FFFFH. Và một xung nữa là nó quay về không và bật cờ bộ định thời TF0 = 1. Tại thời điểm này thì lệnh JNB hạn xuống.
7. Bộ Timer0 được dùng 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 ta phải nạp lại các thanh ghi TH và TL và khởi động lại bộ định thời với giả thiết tần số XTAL = 11, 0592MHz.



Ví dụ 9.5:

Trong ví dụ 9.4 hãy tính toán lượng thời gian trễ trong chương trình con DELAY được tạo ra bởi bộ định thời với giá thiết tần số XTAL = 11,0592MHz.

Lời giải:

Bộ định thời làm việc với 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 $T = \frac{1}{0,9216\text{MHz}} = 1,085\mu s$. Hay nói cách khác, bộ Timer0 đếm tăng sau 1,085μs để tạo ra bộ trễ bằng số đếm ×1,085μs.

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

Ví dụ 9.6:

Trong ví dụ 9.5 hãy tính toán tần số của xung vuông được tạo ra trên chân P1.5.

Lời giải:

Trong tính toán độ thời gian trễ của ví dụ 9.5 ta không tính đến tổng phí của các lệnh trong vòng lặp. Để tính toán chính xác hơn ta cần bổ xung thêm các chu kỳ thời gian của các lệnh trong vòng lặp. Để làm điều đó ta sử dụng các chu kỳ máy từ bảng A-1 trong phụ lục Appendix A được chỉ dưới đây.

```
HERE:           MOV   TL0, #0F2H          2
                MOV   TH0, #0FFH          2
                CPL   P1-5                1
                ACALL DELAY               2
                SJMP HERE                 2
; _____        delay using timer0
DELAY:
                SETB  TR0                 1
AGAIN:          JNB   TF0, AGAIN          1
                CLR   TR0                 1
                CLR   TF0                 1
                RET                       1
                              Total       27
```

T = (2 × 27 × 1.085μs and 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:

Hãy tìm ra độ trễ được tạo ra bởi Timer0 trong đoạn mã sau sử dụng cả hai phương pháp của hình 9.4. Không tính các tổng phí của các lệnh.

```
        CLR   P2.3              ; Xoá P2.3
        MOV   TMOD, #01         ; Chọn Timer0, chế độ 1 (16 bít)
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 Timer0
AGAIN:  JNB   TF0, AGAIN        ; Hiển thị cờ bộ định thời TF0
        CLR   TR0               ; Dừng bộ định thời.
        CLR   TF0               ; Xoá cờ bộ định thời cho vòng sau
        CLR   P2.3
```

Lời giải:

a) Độ trễ được tạo ra trong mã trên là:

(FFFF - B83E + 1) = 47C2H = 18370 hệ thập phân 18370 × 1,085μs = 19, 93145μs.

b) Vì TH - TL = B83EH = 47166 (số thập phân) ta có 65536 - 47166 = 18370.

Điều này có nghĩa là bộ định thời gian đếm từ B83EH đến FFFF. Nó được cộng với một số đếm để về 0 thành một bộ tổng là 18370μs. Do vậy ta có 18370 × 1,085μs = 19,93145ms là độ rộng xung.

Ví dụ 9.8:

Sửa giá trị của TH và TL trong ví dụ 9.7 để nhận được độ trễ thời gian lớn nhất có thể. Hãy tính độ trễ theo miligiây. Trong tính toán cần đưa vào cả tổng phí của các lệnh.

Để nhận độ trễ thời gian lớn nhất có thể ta đặt TH và TL bằng 0. Điều này làm cho bộ định thời đếm từ 0000 đến FFFFH và sau đó quay qua về 0.

```
        CLR   P2.3              ; Xoá P2.3
        MOV   TMOD, #01         ; Chọn Timer0, chế độ 1 (16 bít)
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 Timer0
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
```

Thực hiện biến TH và TL bằng 0 nghĩa là bộ định thời đếm tăng từ 0000 đến FFFFH và sau đó quay qua về 0 để bật cờ bộ định thời TF. Kết quả là nó đi qua 65536 trạng thái. Do vậy, ta có độ trễ = (65536 - 0) × 1.085μs = 71.1065μs.

Trong ví dụ 9.7 và 9.8 chúng ta đã không nạp lại TH và TL vì nó là một xung đơn. Xét ví dụ 9.9 dưới đây để xem việc nạp lại làm việc như thế nào ở chế độ 1.

Ví dụ 9.9:

Chương trình dưới đây tạo ra một sóng vuông trên chân P2.5 liên tục bằng việc sử dụng bộ Timer1 để tạo ra độ trễ thời gian. Hãy tìm tần số của sóng vuông nếu tần số XTAL = 11.0592MHz. Trong tính toán không đưa vào tổng phí của các lệnh vòng lặp:

```
        MOV   TMOD, #01H        ; Chọn Timer0, chế độ 1 (16 bít)
```

```
HERE:       MOV   TL1, #34H        ; Đặt byte thấp TL1 = 34H
            MOV   TH0, #76H        ; Đặt byte cao TH1 = 76H
                                   ; (giá trị bộ định thời là 7634H)
            SETB  TR1              ; Khởi động bộ Timer1
AGAIN:      JNB   TF1, BACK        ; ở lại cho đến khi bộ định thời đếm qua 0
            CLR   TR1              ; Dừng bộ định thời.
            CPL   P1.5             ; Bù chân P1.5 để nhận Hi, L0
            CLR   TF               ; Xoá cờ bộ định thời
            SJMP  AGAIN            ; Nạp lại bộ định thời do chế độ 1 không tự
                                   động nạp lại .
```

Lời giải:

Trong chương trình trên đây ta lưu ý đến đích của SJMP. Ở chế độ 1 chương trình phải nạp lại thanh ghi. TH và TL mỗi lần nếu ta muốn có sóng dạng liên tục. Dưới đây là kết quả tính toán:

Vì FFFFH - 7634H = 89CBH + 1 = 89CCH và 90CCH = 35276 là số lần đếm xung đồng hồ, độ trễ là 35276 × 1.085μs = 38274ms và tần số là $\frac{1}{38274}(\text{Hz}) = 26127\text{Hz}.$

Cũng để ý rằng phần cao và phần thấp của xung sóng vuông là bằng nhau. Trong tính toán trên đây là chưa kể đến tổng phí các lệnh vòng lặp.

9.1.4.2 Tìm các giá trị cần được nạp vào bộ định thời.

giả sử rằng chúng ta biết lượng thời gian trễ mà ta cần thì câu hỏi đặt ra là làm thế nào để tìm ra được các giá trị cần thiết cho các thanh thi TH và TL. Để tính toán các giá trị cần được nạp vào các thanh ghi TH và TL chúng ta hãy nhìn vào ví dụ sau với việc sử dụng tần số dao động XTAL = 11. 0592MHz đối với hệ 8051.

Từ ví dụ 9.10 ta có thể sử dụng những bước sau để tìm các giá trị của các thanh ghi TH và TL.

1. Chia thời gian trễ cần thiết cho 1.0592μs
2. Thực hiện 65536 - n với n là giá trị thập phân nhận được từ bước 1.
3. Chuyển đổi kết quả ở bước 2 sang số Hex với yyxx là giá trị .hex ban đầu cần phải 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 tìm các giá trị cần được nạp vào các thanh ghi vào các thanh ghi TH và TL nếu ta muốn độ thời gian trễ là 5μs. Hãy trình bày chương trình cho bộ Timer0 để tạo ra bộ xung với độ rộng 5μs trên chân P2.3.

Lời giải:

Vì tần số XTAL = 11.0592MHz nên bộ đếm tăng sau mỗi chu kỳ 1.085μs. Điều đó có nghĩa là phải mất rất nhiều khoảng thời gian 1,085μs để có được một xung 5μs. Để có được ta chia 5ms cho 1.085μs và nhận được số n = 4608 nhịp. Để nhận được giá trị cần được nạp vào TL và TH thì ta tiến hành lấy 65536 trừ đi 4608 bằng 60928. Ta đổi số này ra số hex thành EE00H. Do vậy, giá trị nạp vào TH là EE Và TL là 00.

```
            CLR   P2.3             ; Xoá bít P2.3
            MOV   TMOD, #01        ; Chọn Timer0, chế độ 1 (16 bít)
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 Timer0
AGAIN:  JNB   TF0, AGAIN       ; Hiển thị cờ TF0 cho đến khi bộ đếm quay về 0
        CLR   TR0              ; Dừng bộ định thời.
        CLR   TF0              ; Xoá cờ TF0 cho vòng sau.
```

Ví dụ 9.11:

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

Đây là trường hợp giống với ví dụ 9.10 ngoài trừ một việc là ta phải chọn bít để tạo ra sóng vuông. Xét các bước sau:

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

b) Khoảng thời gian cao và phần thấp là $\frac{1}{2}T$ 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 cần tìm là 65536 - 230 = 65306 và ở dạng hex là FF1AH.

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

Chương trình cần viết là:

```
        MOV   TMOD, #10H       ; Chọn bộ định thời Timer0, chế độ 1 (16 bít)
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 Timer1
BACK:   JNB   TF1, BACK        ; giữ nguyên cho đến khi bộ định thời quay về 0
        CLR   TR1              ; Dừng bộ định thời.
        CPL   P1.5             ; Bù bít P1.5 để nhận giá trị cao, thấp.
        CLR   TF1              ; Xoá cờ TF1
        SUMP AGAIN             ; Nạp lại bộ định thời vì chế độ 1 không tự nạp lại.
```

Ví dụ 9.12:

Trước hết ta thực hiện các bước sau:

a) Tính chu kỳ sóng vuông: $T = \frac{1}{50Hz} = 20\mu s$

b) Tính thời gian nửa chu kỳ cho phần cao: $\frac{1}{2}T = 10\mu s$

c) Tính số nhịp đồng hồ: $n = \frac{10\mu s}{1,085\mu s} = 9216$

d) Tính giá trị cần nạp vào TH và TL: 65536 - 9216 = 56320 chuyển về dạng Hex là DC00H và TH = DCH và TL = 00H.

```
        MOV   TMOD, #10H       ; Chọn bộ định thời Timer0, chế độ 1 (16 bít)
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 Timer1
BACK:   JNB   TF1, BACK        ; giữ nguyên cho đến khi bộ định thời quay về 0
```

```
        CLR   TR1              ; Dừng bộ định thời.
        CPL   P2.3             ; Bù bít P1.5 để nhận giá trị cao, thấp.
        CLR   TF1              ; Xoá cờ TF1
        SUMP AGAIN             ; Nạp lại bộ định thời vì chế độ 1 không tự nạp
                                 lại.
```

9.1.4.3 Tạo một độ trễ thời gian lớn.

Như ta đã biết từ các ví dụ trên là lượng thời gian trễ cần tạo ra phụ thuộc vào hai yếu tố:

a) Tần số thạch anh XTAL
b) Thanh ghi 16 bít 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 như ta đã biết giá trị lớn nhất của độ trễ thời gian có thể đạt được bằng cách đặt cả TH và TL bằng 0. Nhưng điều này xảy ra khi như vậy đều không đủ? Ví dụ 9.13 dưới đây cách làm thế nào để có giá trị độ trễ thời gian lớn.

9.1.4.4 Sử dụng bàn tính của Windows để tìm TH và TL.

Bàn tính Calculator của Windows có ngay trong máy tính PC của chúng ta và rất dễ sử dụng để tìm ra 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 số 35.000 vào từ bàn phím.
3. Chuyển về chế độ Hex trên Calculator nó cho ta giá trị 88B8H.
4. Chọn +/- để nhận số đổi dấu - 35.000 dạng thập phân và chuyển về dạng Hex là 7748H.
5. Hai số hex cuối là cho TL = 48 và hai số Hex tiếp theo là cho TH = 77. Ta bỏ quan các số F ở phía bên phải trên Calculator vì số của ta là 16 bít.

Ví dụ 9.13:

Hãy kiểm tra chương trình sau và tìm độ trễ thời gian theo giây, không tính đến tổng phí các lệnh trong vòng lặp.

```
        MOV   TMOD, #10H       ; Chọn bộ Timer1, chế độ 1 (16 bít)
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 Timer1
BACK:   JNB   TF1, BACK        ; giữ nguyên cho đến khi bộ định thời quay
về 0
        CLR   TR1              ; Dừng bộ định thời.
        CLR   TF1              ; Xoá cờ bộ định thời TF1
        DJNZ  R3, AGAIN        ; Nếu R3 không bằng không thì nạp lại bộ định
                                 thời.
```

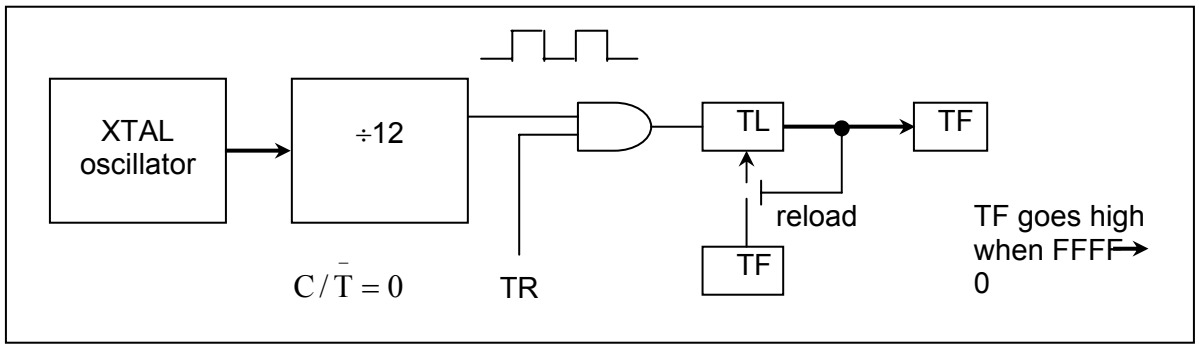
9.1.5 Chế độ O.

Chế độ 0 hoàn toàn giống chế độ 1 chỉ khác là bộ định thời 16 bít được thay bằng 13 bít. Bộ đếm 13 bít có thể giữ các giá trị giữa 0000 đến 1FFFF trong TH - TL. Do vậy khi bộ định thời đạt được giá trị cực đại của nó là 1FFFH thì nó sẽ quay trở về 0000 và cờ TF được bật lên.

9.1.6 Lập trình chế độ 2.

Các đặc trưng và các phép tính của chế độ 2:

1. Nó là một bộ định thời 8 bít, do vậy nó 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 với giá trị 8 bít thì 8051 lấy một bản sao của nó đưa vào TL. Sau đó bộ định thời phải được khởi động. Điều này được thực hiện bởi lệnh "SETB TR0" đối với Timer0 và "SETB TR1" đối với Timer1 giống như ở chế độ 1.
3. Sau khi bộ định thời được khởi động, nó bắt đầu đếm tăng lên bằng cách tăng thanh ghi TL. Nó đếm cho đến khi đại giá trị giới hạn FFH của nó. Khi nó quay trở về 00 từ FFH, nó thiết lập cờ bộ định thời TF. Nếu ta sử dụng bộ định thời Timer0 thì đó là cờ TF0, còn Timer1 thì đó là cờ TF1.



4. Khi thanh ghi TL quay trở về 00 từ FFH thì TF được bật lên 1 thì thanh ghi TL được tự động nạp lại với giá trị ban đầu được giữ bởi thanh ghi TH. Để lặp lại quá trình chúng ta đơn giản chỉ việc xoá cờ TF và để cho nó chạy mà không cần sự can thiệp của lập trình viên để nạp lại giả trị ban đầu. Điều này làm cho chế độ 2 được gọi là chế độ từ nạp lại so với chế độ 1 thì ta phải nạp lại các thanh ghi TH và TL.

Cần phải nhấn mạnh rằng, chế độ 2 là bộ định thời 8 bít. Tuy nhiên, nó lại có khả năng tự nạp khi tự nạp lại thì TH thực chất là không thay đổi với giá trị ban đầu được giữ nguyên, còn TL được nạp lại giá trị được sao từ TH. Chế độ này có nhiều ứng dụng bao gồm việc thiết lập tần số baud trong truyền thông nối tiếp như ta sẽ biết ở chương 10.

9.1.5.1 Các bước lập trình cho chế độ 2.

Để tạo ra một thời gian trễ sử dụng chế độ 2 của bộ định thời cần thực hiện các bước sau:

1. Nạp thanh ghi giá trị TMOD để báo bộ định thời gian nào (Timer0 hay Timer1) được sử dụng và chế độ làm việc nào của chúng được chon.
2. Nạp lại các thanh ghi TH với giá trị đếm ban đầu.
3. Khởi động bộ định thời.
4. Duy trì hiển thị cờ bộ định thời TF sử dụng lệnh "JNB TFx, đích" để xem nó sẽ được bật chưa. Thoát 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 những điều này. Để có được độ chễ lớn chúng ta có thể dùng nhiều thanh ghi như được chỉ ra trong 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 và b) tần số nhỏ nhất có thể có được bằng chương trình này và giá trị TH để đạt được điều đó.

```
        MOV   TMOD, #20H        ; Chọn Timer1/ chế độ 2/ 8 bít/ tự nạp lại.
        MOV   TH1, #5                   ; TH1 = 5
        SETB  TR1               ; Khởi động Timer1
BACK:   JNB   TF1, BACK         ; giữ nguyên cho đến khi bộ định thời quay 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.
```

Lời giải:

a) Trước hết để ý đến đích của lệnh SJMP. Trong chế độ 2 ta không cần phải nạp lại TH vì nó là chế độ tự nạp. Bây giờ ta lấy (256 - 05).1.085μs = 251×1.085μs = 272.33μs là phần cao của xung. Cả chu kỳ của xung là T = 544.66μs và tần số là $\frac{1}{T} = 1{,}83597\text{kHz}.$

b) Để nhận tần số nhỏ nhất có thể ta cần tạo T chu kỳ lớn nhất có thể có nghĩa là TH = 00. Trong trường hợp này ta có T = 2 × 256 × 1.085μs = 555.52μs và tần số nhỏ nhất sẽ là $\frac{1}{T} = 1{,}8\text{kHz}.$

Ví dụ 9.15:

Hãy tìm tần số của xung vuông được tạo ra trên P1.0.

Lời giải:

```
        MOV   TMOD, #2H         ; Chọn Timer0, chế độ 1 (8 bít tự nạp lại)
AGAIN:  MOV   TH0, #0                   ; Nạp TH0 = 00
        MOV   R5, #250          ; Đếm cho độ trễ lớn
        ACALL DELAY
        CPL   P1.0
        SJMP  AGAIN
DELAY:  SETB  TR0               ; Khởi động Timer0
BACK:   JNB   TF1, BACK         ; giữ nguyên cho đến khi bộ định thời quay về 0
        CLR   TR0               ; Dừng Timer0.
        CLR   TF0               ; Xoá cờ TF0 cho vòng sau.
        DJNZ  R5, DELAY
        RET
```

T = 2 × (250 × 256 × 1.085μs) = 1.38.88ms và f = 72Hz.

Ví dụ 9.16:

Giả sử ta đang lập trình chế độ 2 hãy tìm các giá trị (dạng Hex) 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

Lời giải:

Chúng ta có thể sử dụng bàn tính Calculator của Windows để kiểm tra kết quả được cho bởi trình hợp ngữ. Hãy chọn Calculator ở chế độ Decimal và nhập vào số 200. Sau đó chọn Hex, rồi ấn +/ - để nhận giá trị của TH. Hãy nhớ rằng chúng ta chỉ sử dụng đúng hai chữ số và bỏ qua phần bên trái vì dữ liệu chúng ta là 8 bít. Kết quả ta nhận được như sau:

Dạng thập phân	*Số bù hai (giá trị TH)*
- 200	38H
- 60	C4H
- 3	FDH
- 12	F4H
- 48	DOH

9.1.5.2 Các trình hợp ngữ và các giá trị âm.

Vì bộ định thời là 8 bít trong chế độ 2 nên ta có thể để cho trình hợp ngữ tính giá trị cho TH. Ví dụ, trong lệnh "MOV TH0, # - 100" thì trình hợp ngữ sẽ tính toán – 100 = 9C và gán TH = 9CH. Điều này làm cho công việc của chúng ta dễ dàng hơn.

Ví dụ 9.17:

Hãy tìm a) tần số sóng vuông được tạo ra trong đoạn mã dưới đây và độ đầy xung của sóng này.

```
        MOV   TMOD, #2H          ; Chọn bộ Timer0/ chế độ 2/ (8 bít, tự nạp lại).
        MOV   TH0, # - 150       ; Nạp TH0 = 6AH là số bù hai của  - 150
        SETB  TR1                ; Khởi động Timer1
AGAIN:          SETB P1.3                ; P1.3 = 1
        ACALL DELAY
        ACALL P1.3               ; P1.3 = 0
        ACALL DELAY
        SJMP AGAIN

        SETB  TR0                ; Khởi động Timer0
BACK:           JNB    TF0, BACK         ;giữ nguyên cho đến khi bộ định thời quay về 0
        CLR   TR0                ; Dừng Timer0
        CLR   TF0                ; Xoá cờ TF cho vòng sau.
        RET
```

Lời giải:

Để tìm giá trị cho TH ở chế độ 2 thì trình hợp ngữ cần thực hiện chuyển đổi số âm khi ta nhập vào. Điều này cũng làm cho việc tính toán trở nê dễ dàng. Vì ta đang sử dụng 150 xung đồng hồ, nên ta có thời gian trễ cho chương trình con DELAY là $150 \times 1.085\mu s$ và tần số là $f = \frac{1}{T} = 2{,}048\text{kHz}.$

Để ý rằng trong nhiều tính toán thời gian trễ ta đã bỏ các xung đồng hồ liên quan đến tổng phí các lệnh trong vòng lặp. Để tính toán chính xác hơn thời gian trễ và cả tần số ta đang cần phải đưa chúng vào. Nếu ta dùng một máy hiện sóng số và ta không nhận

được tần số đúng như ta tính toán thì đó là do tổng phí liên quan đến các lệnh gọi trong vòng lặp.

Trong phần này ta đã dùng bộ định thời 8051 để tạo thời gian trễ. Tuy nhiên, công dụng mạnh hơn và sáng tạo hơn của các bộ định thời này là sử dụng chúng như các bộ đếm sự kiện. Chúng ta sẽ bàn về công dụng của bộ đếm này ở phần kế tiếp.

9.2 Lập trình cho bộ đếm.

Ở phần trên đây ta đã sử dụng các bộ định thời của 8051 để tạo ra các độ trễ thời gian. Các bộ định thời này cũng có thể được dùng như các bộ đếm các sự kiện xảy ra bên ngoài 8051. Công dụng của bộ đếm/ bộ định thời như bộ đếm sự kiện sẽ được tình bày ở phần này. Chừng nào còn liên quan đến công dụng củ bộ định thời như bộ đếm sự kiện thì mọi vấn đề mà ta nói về lập trình bộ định thời ở phần trước cũng được áp dụng cho việc lập trình như là một bộ đếm ngoại trừ nguồn tần số. Đối với bộ định thời/ bộ đếm khi dùng nó như bộ định thời thì nguồn tần số là tần số thạch anh của 8051. Tuy nhiên, khi nó được dùng như một bộ đếm thì nguồn xung để tăng nội dung các thanh ghi TH và TL là từ bên ngoài 8051. Ở chế độ bộ đếm, hãy lưu ý rằng các thanh ghi TMOD và TH, TL cũng giống như đối với bộ định thời được bàn ở phần trước, thậm chí chúng vẫn có cùng tên gọi. Các chế độ của các bộ định thời cũng giống nhau.

9.2.1 Bít C/T trong thanh ghi TMOD.

Xem lại phần trên đây về bít C/T trong thanh ghi TMOD ta thấy rằng nó quyết định nguồn xung đồng hồ cho bộ định thời. Nếu bít C/T = 0 thì bộ định thời nhận các xung đồng hồ từ bộ giao động thạch anh của 8051. Ngược lại, khi C/T = 1 thì bộ định thời được sử dụng như bộ đếm và nhận các xung đồng hồ từ nguồn bên ngoài của 8051. Do vậy, khi bít C/T = 1 thì bộ đếm lên, khi các xung được đưa đến chân 14 và 15. Các chân này có tên là T0 (đầu vào của bộ định thời Timer0) và T1 (đầu vào của bộ Timer1). Lưu ý rằng hai chân này thuộc về cổng P3. Trong trường hợp của bộ Timer0 khi C/T = 1 thì chân P3.4 cấp xung đồng hồ và bộ đếm tăng lên đối với mỗi xung đồng hồ đi đến từ chân này. Tương tự như vậy đói với bộ Timer1 thì khi C/T = 1 với mỗi xung đồng hồ đi đến từ P3.5 bộ đếm sẽ đếm tăng lên 1.

Bảng 9.1: Các chân cổng P3 được dùng cho Timer0 và Timer1.

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:

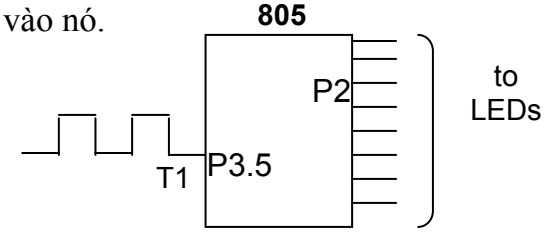
Giả sử rằng xung đồng hồ được cấp tới chân T1, hãy viết chương trình cho bộ đếm 1 ở chế độ 2 để đếm các xung và hiển thị trạng thái của số đếm TL1 trên cổng P2.

Lời giải:

```
        MOV   TMOD, #01100000B  ; Chọn bộ đếm 1, chế độ 2, bít 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.
```

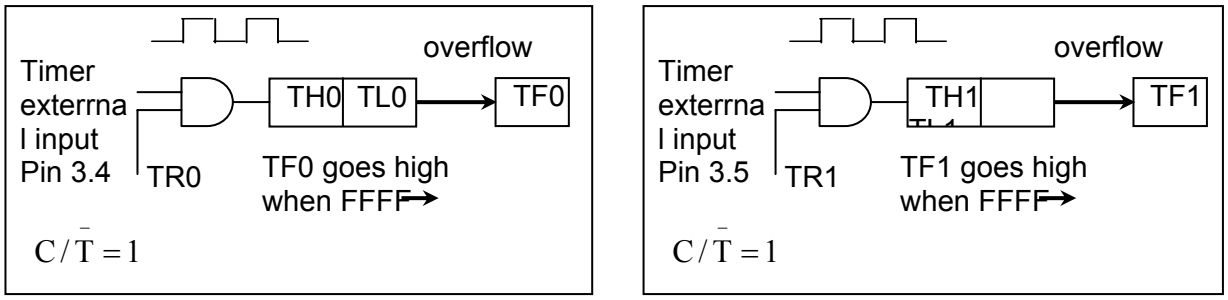
```
        JNB    TF1, Back          ; Duy trì nó nếu TF = 0
        CLR    TR1                ; Dừng bộ đếm
        CLR    TF1                ; Xoá cờ TF
        SJMP   AGAIN              ; Tiếp tục thực hiện
```

Để ý trong chương trình trên về vai trò của lệnh "SETB P3.5" vì các cổng được thiết lập dành cho đầu ra khi 8051 được cấp nguồn nên ta muốn P3.5 trở thành đầu vào thì phải bật nó lên cao. Hay nói cách khác là ta phải cấu hình (đưa lên cao) chân T1 (P3.5) để cho phép các xung được cấp vào nó.



Trong ví dụ 9.18 chúng ta sử dụng bộ Timer1 như bộ đếm sự kiện để nó đếm lên mỗi khi các xung đồng hồ được 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 khác mà có thể chuyển đổi thành các xung.

Trong ví dụ 9.19 các thanh ghi TL được chuyển đổi về mã ASCII để hiển thị trên một LCD.



Hình 9.5: a) Bộ Timer0 với đầu vào ngoài (chế độ 1)
b) Bộ Timer1 với đầu vào ngoài (chế độ 1)

Ví dụ 9.19:

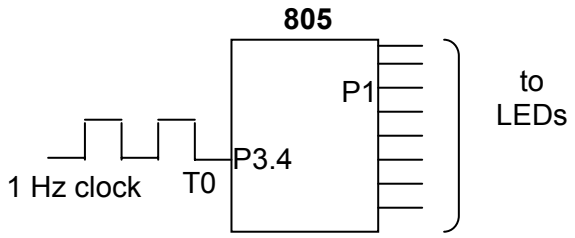
giả sử rằng một xung tần số 1Hz được nối tới chân đầu vào P3.4. Hãy viét chương trình hiển thị bộ đếm 0 trên một LCD. Hãy đặt số ban đầu của TH0 là - 60.

Lời giải:

Để hiển thị số đếm TL trên một LCD ta phải thực hiện chuyển đổi giữ liệu 8 bít nhị phân về ASCII.

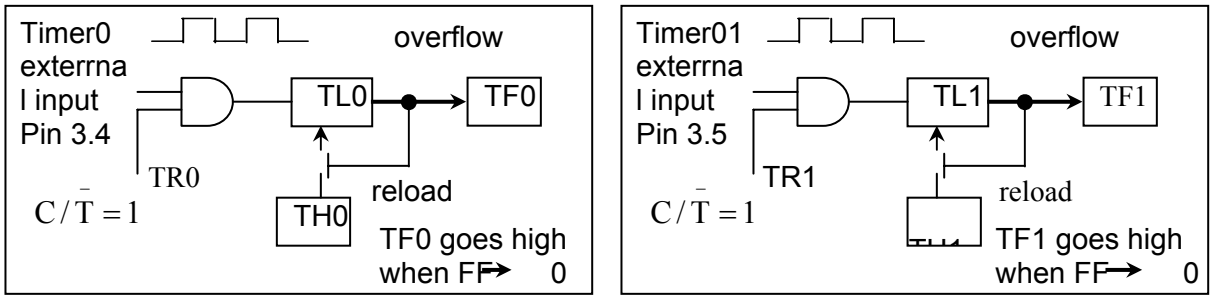
```
        ACALL LCD-SET UP          ; Gọi chương trình con khởi tạo CLD
        MOV   TMOD, #000110B      ; Chọn bộ đếm 0, chế độ 2, bít C/T = 1
        MOV   TH0, # - 60         ; Đếm 60 xung
        SETB  P3.4                ; Lấy đầu vào T0
AGAIN:  SETB  TR0                 ; Sao chép số đếm TL0
BACK:   MOV   A, TL0              ; Gọi chương trình con để chuyển đổi
                                  trong các thanh ghi R2, R3, R4.
        ACALL CONV                ; Gọi chương trình con hiển thị trên LCD
```

```
            ACALL DISLAY                ; Thực hiện vòng lặp nếu TF = 0
            JNB   TF0, BACK             ; Dừng bộ đếm 0
            CLR   TR0                   ; Xoá cờ TF0 = 0
            CLR   TF0                   ; Tiếp tục thực hiện
            SJMP  AGAIN                 ; Việc chuyển đổi nhị phân về mã ASCII
khi trả dữ liệu ASCII có trong các thanh ghi R4, R3, R2 (R2 có LSD) - chữ số nhỏ nhất.
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 nó về ASCII
            MOV   R4, A                 ; Lưu chữ số có nghĩa lớn nhất MSD
            MOV   A, B                  ;
            ORL   A, #30H               ; Đổi số thứ hai về ASCII
            MOV   R3, A                 ; Lưu nó
            MOV   A, R2
            ORL   A, #30H               ; Đổi số thứ ba về ASCII
            MOV   R2, A                 ; Lưu số ASCII vào R2.
            RET
```



Sử dụng tần số 60Hz ta có thể tạo ra các giây, phút, giờ.

Lưu ý rằng trong vòng đầu tiên, nó bắt đầu từ 0 vì khi RESET thì TL0 = 0; Để giải quyết vấn đề này hãy nạp TL0 với giá trị - 60 ở đầu chương trình.



Hình 9.6: Bộ Timer0 với đầu vào ngoài (chế độ 2)

Hình 9.7: Bộ Timer0 với đầu vào ngoài (chế độ 2)

Như một ví dụ ứng dụng khác của bộ định thời gian với bít C/T = 1, ta có thể 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. Từ đầu vào này ta hiển thị lên một LCD. Đây sẽ

là một đồng hồ số tuyệt vời nhưng nó không thật chính xác. Ví dụ này có thể tìm thấy ở phụ lục Appendix E.

Trước khi kết thúc chương này ta cần nhắc lại hai vấn đề quan trọng.

1. Chúng ta có thể nghĩ rằng công dụng của lệnh "JNB TFx, đích" để hiển thị mức cao của cờ TF là một sự lãng phí thời gian của BVĐK. Điều đó đúng có một giải pháp cho vấn đề này là sử dụng các ngắt. Khi sử dụng các ngắt ta có thể đi thực hiện các công việc khác với BVĐK. Khi cờ TF được bật thì nó báo cho ta biết đây là điểm quan trọng về thế mạnh của 8051 (mà ta sẽ bàn ở chương 11).
2. Chúng ta muốn biết các thanh ghi TR0 và TR1 thuộc về đâu. Chúng thuộc về một thanh ghi gọi là TCON mã sẽ được ban sau ở 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 Timer0	
	SETB TR0 = SETB TCON.4
	CLR TR0 = CLR TCON.4
	SETB TF = SETB TCON.5
	CLR TF0 = CLR TCON.5
Đối với Timer1	
	SETB TR1 = SETB TCON.6
	CLR TR1 = CLR TCON.6
	SETB TF1 = SETB TCON.7
	CLR TF1 = CLR TCON.7

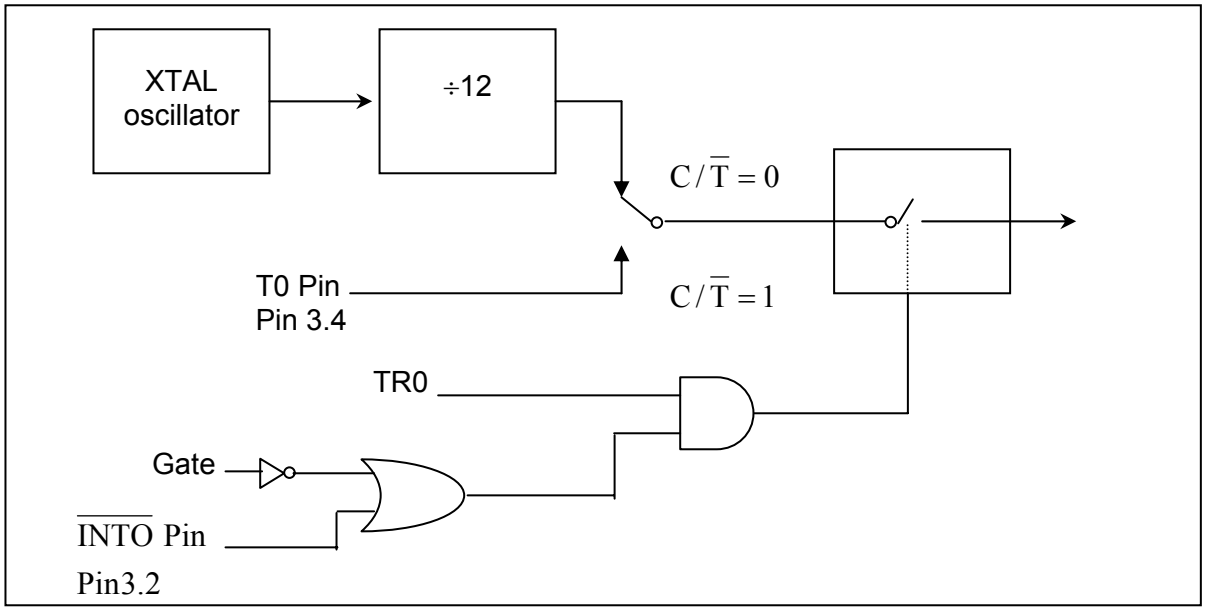
9.2.2 Thanh ghi TCON.

Trong các ví dụ trên đây ta đã thấy công dụng của các cờ TR0 và TR1 để bật/ tắt các bộ định thời. Các bít này là một bộ phận của thanh ghi TCON (điều khiển bộ định thời). Đây là thanh ghi 8 bít, như được chỉ ra trong bảng 9.2 thì bốn bít trên được dùng để lưu cất các bít TF và TR cho cả Timer0 và Timer1. Còn bốn bít thấp được thiết lập dành cho điều khiển các bít ngắt mà ta sẽ bàn ở chương 11. Chúng ta phải lưu ý rằng thanh ghi TCON là thanh ghi có thể đánh địa chỉ theo bít được. Nên ta có thể thay các lệnh như "SETB TR1" là "CLR TR1" bằng các lệnh tương ứng như "SET TCON.6" và "CLR TCON.6" (Bảng 9.2).

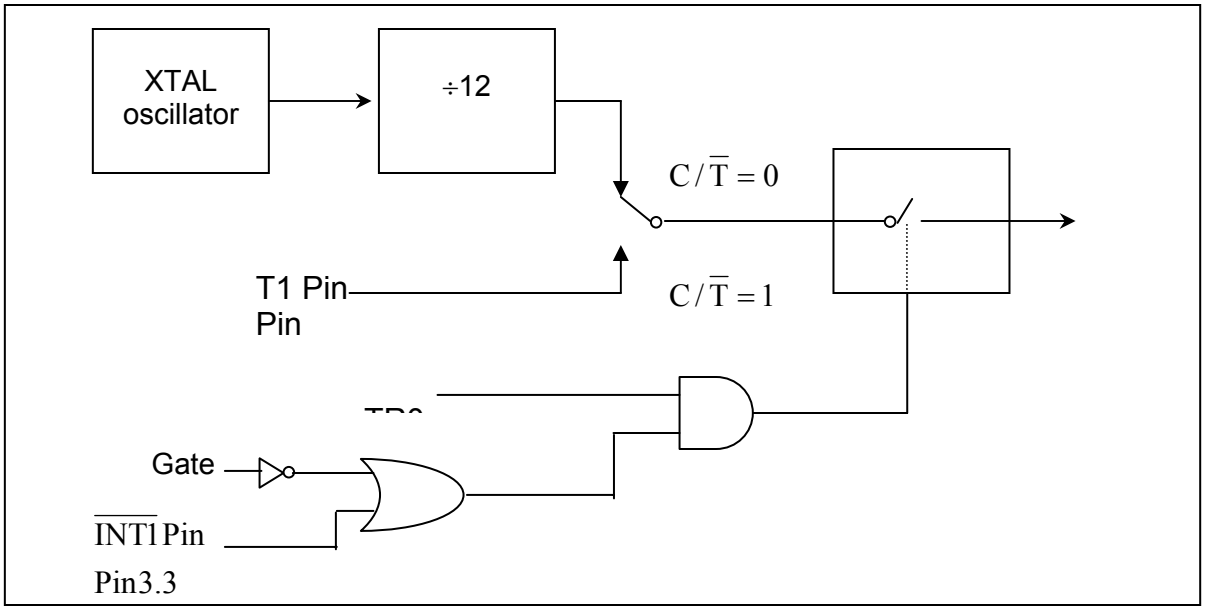
9.3 Trường hợp khi bít GATE = 1 trong TMOD.

Trước khi kết thúc chương ta cần bàn thêm về trường hợpợ khi bít GATE = 1trong thanh ghi TMOD. Tất cả những gì chúng ta vừa nói trong chương này đều giả thiết GATE = 0. Khi GATE = 0 thì bộ định thời được khởi động bằng các lệnh "SETB TR0" và "SETB TR1" đối với Timer0 và Timer1 tương ứng. Vậy điều gì xảy ra khi bít GATE = 1? Như ta có thể nhìn thây trên hình 9.8 và 9.9 thì nếu GATE = 1 thì việc khởi động và dừng bộ định thời được thực hiện từ bên ngoài qua chân P2.3 và P3.3 đối với Timer0 và Timer1 tương ứng. Mặc dù rằng TRx được bật lên bằng lệnh "SETB TRx" thì cũng cho phép ta 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 điều khiển phần cứng để dừng

và khởi động bộ định thời nay có thể có rất nhiều ứng dụng. Ví dụ, chẳng hạn 8051 được dùng trong một sản phẩm phát báo động mỗi giây dùng bộ Timer0 theo nhiều việc khác. Bộ Timer0 được bật lên bằng phần mềm qua lệnh "SETB TR0" và nằm ngoài sự kiểm soát của người dùng sản phẩm đó. Tuy nhiên, khi nối một công tắc chuyển mạch tới chân P2.3 ta có thể dừng và khởi động bộ định thời gian bằng cách đó để tắt báo động.



Hình 9.8: Bộ định thời/ bộ đếm 0.



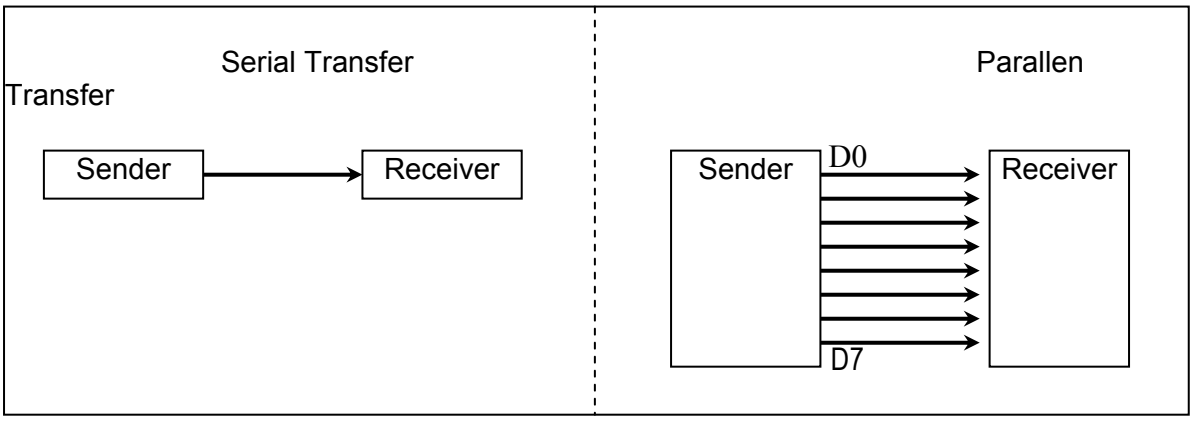
Hình 9.9: Bộ định thời/ bộ đếm 1.

CHƯƠNG 10
Truyền thông nối tiếp của 8051

Các máy tính truyền dữ liệu theo hai cách: Song song và nối tiếp. Trong truyền dữ liệu song song thường cần 8 hoặc nhiều đường dây dẫn để truyền dữ liệu đến một thiết bị chỉ cách xa vài bước. Ví dụ của truyền dữ liệu song song là các máy in và các ổ cứng, mỗi thiết bị sử dụng một đường cáp với nhiều dây dẫn. Mặc dù trong các trường hợp như vậy thì nhiều dữ liệu được truyền đi trong một khoảng thời gian ngắn bằng cách dùng nhiều dây dẫn song song nhưng khoảng cách thì không thể lớn được. Để truyền dưdx liệu đi xa thì phải sử dụng phương pháp truyền nối tiếp. Trong truyền thông nối tiếp dữ liệu được gửi đi từng bít một so với truyền song song thì một hoặc nhiều byte được truyền đi cùng một lúc. Truyền thông nối tiếp của 8051 là chủ đề của chương này. 8051 đã được cài sẵn khả năng truyền thông nối tiếp, do vậy có thể truyền nhánh dữ liệu với chỉ một số ít dây dẫn.

10.1 Các cơ sở của truyền thông nối tiếp.

Khi một bộ vi xử lý truyền thông với thế giới bên ngoài thì nó cấp dữ liệu dưới dạng từng khúc 8 bít (byte) một. Trong một số trường hợp chẳng hạn như các máy in thì thông tin đơn giản được lấy từ đường bus dữ liệu 8 bít và được gửi đi tới bus dữ liệu 8 bít của máy in. Điều này có thể làm việc chỉ khi đường cáp bus không quá dài vì các đường cáp dài làm suy giảm thậm chí làm méo tín hiệu. Ngoài ra, đường dữ liệu 8 bít giá thường đắt. Vì những lý do này, việc truyền thông nối tiếp được dùng để truyền dữ liệu giữa hai hệ thống ở cách xa nhau hàng trăm đến hàng triệu dặm. Hình 10.1 là sơ đồ truyền nối tiếp so với sơ đồ truyền song song.



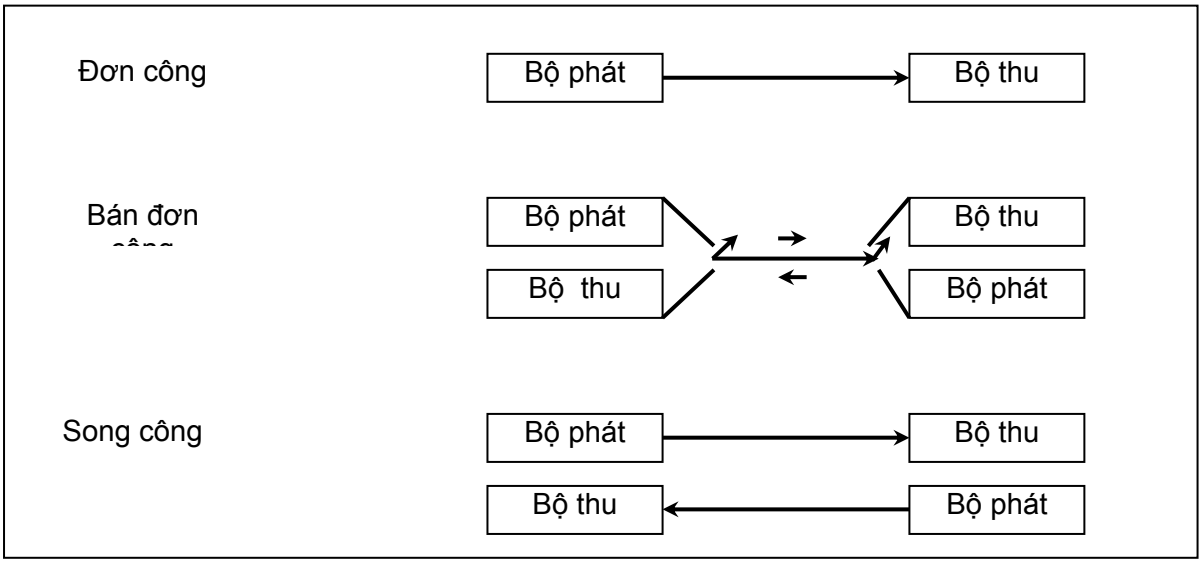
Hình 10.1: Sơ đồ truyền dữ liệu nối tiếp so với sơ đồ truyền song song.

Thực tế là trong truyền thông nối tiếp là một đường dữ liệu duy nhất được dùng thay cho một đường dữ liệu 8 bít của truyền thông song song làm cho nó không chỉ rẻ hơn rất nhiều mà nó còn mở ra khả năng để hai máy tính ở cách xa nhau có truyền thông qua đường thoại.

Đối với truyền thông nối tiếp thì để làm được các byte dữ liệu phải được chuyển đổi thành các bít nối tiếp sử dụng thanh ghi giao dịch vào - song song - ra - nối tiếp. Sau đó nó có thể được truyền quan một đường dữ liệu đơn. Điều này cũng có nghĩa là ở đầu thu cũng phải có một thanh ghi 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 nó phải được chuyển đổi từ các số 0 và 1 sang âm thanh ở dạng sóng hình sin. Việc chuyển đổi này thực thi bởi một thiết bị có tên gọi là Modem là chữ viết tắt của “Modulator/ demodulator” (điều chế/ giải điều chế).

Khi cự ly truyền ngắn thì tín hiệu số có thể được truyền như nói ở trên, một dây dẫn đơn giản và không cần điều chế. Đây là cách các bàn PC và IBM truyền dữ liệu đến bo mạch mẹ. Tuy nhiên, để truyền dữ liệu đi xa dùng các đường truyền chẳng hạn như đường thoại thì việc truyền thông 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).

Truyền thông dữ liệu nối tiếp sử dụng hai phương pháp đồng bộ và dị bộ. Phương pháp đồng bộ truyền một khối dữ liệu (các ký tự) tại cùng thời điểm trong khi đó truyền dị bộ chỉ truyền từng byte một. Có thể viết phần mềm để sử dụng một trong hai phương pháp này, những chương trình có thể rất dài và buồn tẻ. Vì lý do này mà nhiều nhà sản xuất đã cho ra thị trường nhiều loại IC chuyên dụng phục vụ cho truyền thông dữ liệu nối tiếp. Những IC này phục vụ như các bộ thu - phát dị bộ tổng hợp VART (Universal Asynchronous Receiver Transmitter) và các bộ thu - phát đồng - dị bộ tổng hợp UBART (Universal Asynchronous Receiver Transmitter). Bộ vi điều khiển 8051 có một cài sẵn một UART mà nó sẽ được bàn kỳ ở mục 10.3.



Hình 10.2: Truyền dữ liệu đơn công, bán công và song công.

10.1.1 Truyền dữ liệu bán công và song công.

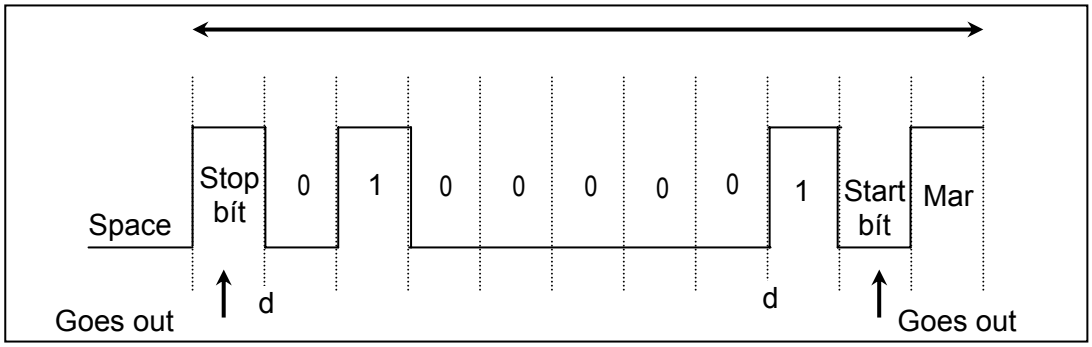
Trong truyền dữ liệu nếu dữ liệu có thể được vừa phát và vừa được thu thì gọi là truyền song công. Điều này tương phản với truyền đơn công chẳng hạn như các máy in chỉ nhận dữ liệu từ máy tính. Truyền song công có thể có hai loại là bán song công và song công hoàn toàn phụ thuộc vào truyền dữ liệu có thể xảy ra đồng thời không? Nếu dữ liệu được truyền theo một đường tại một thời điểm thì được gọi là truyền bán song công. Nếu dữ liệu có thể đi theo cả hai đường cùng một lúc thì gọi là song công toàn phần. Tất nhiên, truyền song công đòi hỏi hai đường dữ liệu (ngoài đường âm của tín hiệu), một để phát và một để thu dữ liệu cùng một lúc.

10.1.2 Truyền thông nối tiếp dị bộ và đóng khung dữ liệu.

Dữ liệu đi vào ở đầu thu của đường dữ liệu trong truyền dữ liệu nói tiếp toàn là các số 0 và 1, nó thật là khó làm cho dữ liệu ấy có nghĩa là nếu bên phát và bên thu không cùng thống nhất về một tệp các luật, một thủ tục, về cách dữ liệu được đóng gói, bao nhiêu bít tạo nên một ký tự và khi nào dữ liệu bắt đầu và kết thúc.

10.1.3 Các bít bắt đầu và dừng.

Truyền thông dữ liệu nối tiếp dị bộ được sử dụng rộng rãi cho các phép truyền hướng kỹ tự, còn các bộ truyền dữ liệu theo khối thì sử dụng phương phát đồng bộ. Trong phương pháp dị bộ, mỗi ký tự được bố trí giữa các bít bắt đầu (start) và bít dừng (stop). Công việc này gọi là đóng gói dữ liệu. Trong đóng gói dữ liệu đối với truyền thông dị bộ thì dữ liệu chẳng hạn là các ký tự mã ASCII được đóng gói giữa một bít bắt đầu và một bít dừng. Bít bắt đầu luôn luôn chỉ là một bít, còn bít dừng có thể là một hoặc hai bít. Bít bắt đầu luôn là bít thấp (0) và các bít dừng luôn là các bít cao (bít 1). Ví dụ, hãy xét ví dụ trên hình 10.3 trong đó ký tự "A" của mã ASCII (8 bít nhị phân là 0100 0001) đóng gói khung giữa một bít bắt đầu và một bít dừng. Lưu ý rằng bít thấp nhất LSB được gửi ra đầu tiên.



Hình 10.3: Đóng khung một ký tự "A" của mã ASCII (41H) có tín hiệu là 1 (cao) được coi như là một dấu (mark), còn không có tín hiệu tức là 0 (thấp) thì được coi là khoảng trống (space). Lưu ý rằng phép truyền bắt đầu với start sau đó bít D0, bít thấp nhất LSB, sau các bít còn lại cho đến bít D7, bít cao nhất MSB và cuối cùng là bít dừng stop để báo kết thúc ký tự "A".

Trong truyền thông nối tiếp dị bộ thì các chíp IC ngoại vi và các modem có thể được lập trình cho dữ liệu với kích thước theo 7 bít hoặc 8 bít. Đây là chưa kể các bít dừng stop có thể là 1 hoặc 2 bít. Trong khi các hệ ASCII cũ hơn (trước đây) thì các ký tự là 7 bít thì ngay nay do việc mở rộng các ký tự ASCII nên dữ liệu nhìn chung là 8 bít. Trong các hệ cũ hơn do tốc độ chậm của các thiết bị thu thì phải sử dụng hai bít dừng để đảm bảo thời gian tổ chức truyền byte kế tiếp. Tuy nhiên, trong các máy tính PC hiện tại chỉ sử dụng 1 bít stop như là chuẩn.

Giả sử rằng chúng ta đang truyền một tệp văn bản các ký tự ASCII sử dụng 1 bít stop thì ta có tổng cộng là 10 bít cho mỗi ký tự gồm: 8 bít cho ký tự ASCII chuẩn và 1 bít start cùng 1 bít stop. Do vậy, đối với mỗi ký tự 8 bít thì cần thêm 2 bí vị chi là mất 25% tổng phí.

Trong một số hệ thống để nhằm duy trì tính toàn vẹn của dữ liệu thì người ta còn thêm một bít lẻ (parity bít). Điều này có nghĩa là đối với mỗi ký tự (7 hoặc 8 bít tuỳ từng hệ) ta có thêm một bít ngoài các bít start và stop. Bít chẵn lẻ là bít chẵn hoặc bít lẻ. Nếu là bít lẻ là số bít của dữ liệu bao gồm cả bít chẵn lẻ sẽ là một số lẻ các sô 1. Tương tự như vậy đối với trường hợp bít chẵn thì số bít của dữ liệu bao gồm cả bít chẵn - lẻ sẽ là một số chẵn của các số 1. Ví dụ, ký tự "A" của mã ASCII ở dạng nhị phân là 0100 0001, có bít 0 là bít chẵn. Các chíp UART đều cho phép việc lập trình bít chẵn - lẻ về chẵn, lẻ hoặc không phân biệt chẵn lẻ.

10.1.4 Tốc độ truyền dữ liệu.

Tốc độ truyền dữ liệu trong truyền thông dữ liệu nối tiếp được gọi là bít trong giây bps (bít per second). Ngoài ra, còn được sử dụng một thuật ngữ rộng rãi nữa là tốc độ baud. Tuy nhiên, các tốc baud và bps là hoàn toàn không bằng nhau. Điều này

là do tốc baud là thuật ngữ của modem và được định nghĩa như là sô lần thay đổi của tín hiệu trong một giây. Trong các modem có những trường hợp khi một sự thay đổi của tín hiệu thì nó truyền vài bít dữ liệu. Nhưng đối với một dây dẫn thì tốc độ baud và bps là như nhau nên trong cuốn sách này chúng ta có thể dùng thay đổi các thuật ngữ này cho nhau.

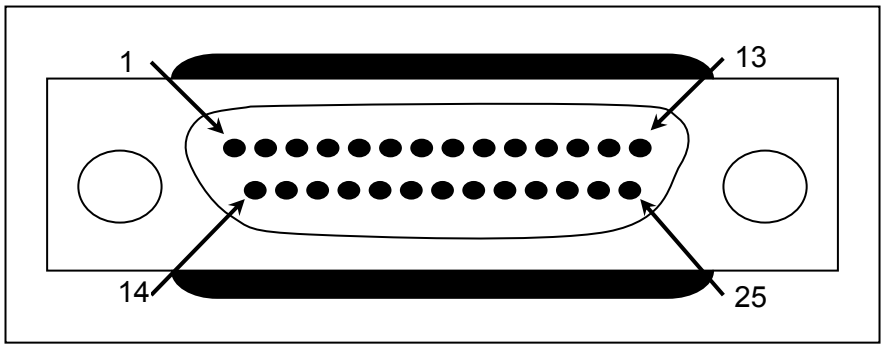
Tốc độ truyền dữ liệu của một hệ máy tính đã cho phụ thuộc vào các cổng truyền thông kết nối vào trong hệ thống đo. Ví dụ, các máy tính PC/XT trước đây của IBM có thể truyền dữ liệu với tốc độ 100 đến 9600 bps. Tuy nhiên, trong những năm gần đây thì các máy tính PC dựa trên Pentium truyền dữ liệu với tốc độ lên tới 56kbps. Cần phải nói thêm rằng trong truyền thông dữ liệu nối tiếp dị bộ thì tốc độ baud nhìn chung là bị giới hạn ở 100.000 bps.

10.1.5 Các chuẩn RS232.

Để cho phép tương thích giữa các thiết bị truyền thông dữ liệu được sản xuất bởi các hãng khác nhau thì một chuẩn giao diện được gọi là RS232 đã được thiết lập bởi hiệp hội công nghiệp điện tử EIA vào năm 19960. Năm 1963 nó được sửa chỉnh và được gọi là RS232A và vào các năm 1965 và 1969 thì được đổi thành RS232B và RS232C. Ở đây chúng ta đơn giản chỉ nói đến RS232. Ngày nay RS232 là chuẩn giao diện I/O vào - ra nối tiếp được sử dụng rộng rãi nhất. Chuẩn này được sử dụng trong máy tính PC và hàng loạt các thiết bị khác nhau. Tuy nhiên, vì nó được thiết lập trước họ lô-gíc TTL rất lâu do vậy điện áp đầu vào và đầu ra của nó không tương thích với mức TTL. Trong RS232 thì mức 1 được biểu diễn bởi - 3v đến 25v trong khi đó mức 0 thì ứng với điện áp + 3v đến +25v làm cho điện áp - 3v đến + 3v là không xác định. Vì lý do này để kết nối một RS232 bất kỳ đến một hệ vi điều khiển thì ta phải sử dụng các bộ biến đổi điện áp như MAX232 để chuyển đổi các mức lô-gíc TTL về mức điện áp RS232 và ngược lại. Các chíp IC MAX232 nhìn chung được coi như cá bộ điều khiển đường truyền. Kết nối RS232 đến MAX232 được thoả thuận ở phần 10.2.

10.1.6 Các chân của RS232.

Bảng 10.1 cung cấp sơ đồ chân của cáp RSE232 và các tên gọi của chúng thường được gọi là đầu nối DB - 25. Trong lý hiệu thì đầu nối cắm vào (đầu đực) gọi là DB - 25p và đầu nối cái được gọi là DB - 25s.



Hình 10.4: Đầu nối DB - 25 của RS232.

Vì không phải tất cả mọi chân đều được sử dụng trong cáp cảu máy tính PC, nên IBM đưa ra phiên bản của 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.

Bảng 10.1: Các chân của RS232, 25 chân (DB - 25).

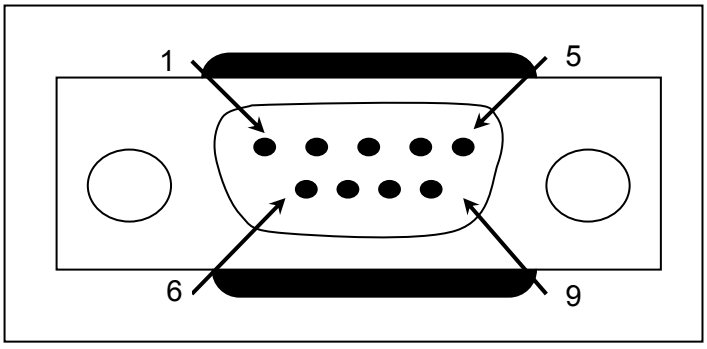
Số chân	Mô tả
1	Đất cách ly (Protective Cround)
2	Dữ liệu được truyền TxD (TráNsmitted data)
3	Dữ liệu được phân RxD (Received data)
4	Yêu cầu gửi RTS (Request To Send)
5	Xoá để gửi CIS (Clear To Send)
6	Dữ liệu sẵn sàng DSR (Data Set Ready)
7	Đất của tín hiệu GND (Signal Cround)
8	Tách tín hiệu mạng dữ liệu DCD (Data Carrier Detect)
9/10	Nhận để kiểm tra dữ liệu (Received for data testing)
11	Chưa dùng
12	Tách tín hiệu mạng dữ liệu thứ cấp (Secondary data carrier
13	detect)
14	Xoá đẻ nhận dữ liệu thứ cấp (Secondary Clear to Send)
15	Dữ liệu được truyền thứ cấp (Secondary Transmit Signal
16	Element Timing)
17	Truyền phân chia thời gian phần tử tín hiệu (Transmit Signal
18	Element Timing)
19	Dữ liệu được nhận thứ cấp (Secondary Received data)
20	Nhận phân chia thời gian phần tử tín hiệu (Receiveo Signal
21	Element Timing)
22	Chưa dùng
23	Yêu cầu để nhận thứ cấp (Secondary Request to Send)
24	Đầu dữ liệu sẵn sàng (Data Terminal Ready)
25	Phát hiện chất lượng tín hiệu (Signal Qualyty Detector)
	Báo chuông (Ring Indicator)
	Chọn tốc độ tín hiệu dữ liệu (Data Signal Rate Select)
	Truyền phân chia thời gian tín hiệu (Transmit Signal Element Timing)
	Chưa dùng

10.1.7 Phân loại truyền thông dữ liệu.

Thuật ngữ hiện nay phân chia thiết bị truyền thông dữ liệu thành một 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 các 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 trong các bảng 10.1 và 10.2 đều xuất phát từ gốc độ của DTE.

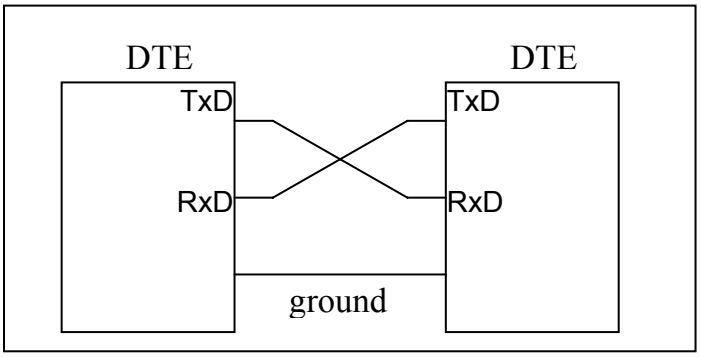
Kết nối đơn giản nhất giữa một PC và bộ vi điều khiển yêu cầu tối thiểu là những chân sau: TxD, RxD và đất như chỉ ra ở hình 10.6. Để ý rằng trên hình này thì các chân TxD và RxD được đổi cho nhau.

Hình 10.5: Sơ đồ đầu nối DB - 9 của RS232.



Bảng 10.2: Các tín hiệu của các chân đầu nối DB - 9 trên máy tính IBM PC.

Mô tả	Số chân	
1	Da ta carrier detect (DCD)	Tránh tín hiệu mạng dữ liệu
2	Received data (RxD)	Dữ liệu được nhận
3	Transmitted data (TxD)	Dữ liệu được gửi
4	Data terminal ready (DTR)	Đầu 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



Hình 10.6: Nối kết không modem.

10.1.8 Kiểm tra các 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 một thực tế là trong truyền thông dữ liệu nối tiếp thiết bị thu có thể không có chỗ để chứa dữ liệu, do đó phải có cách để báo cho bên phát dừng gửi dữ liệu. Rất nhiều chân của RS232 được dùng cho các tín hiệu bắt tay. Dưới đây là mô tả về chúng như là một tham khảo và chúng có thể được bỏ qua vì chúng không được hỗ trợ bởi chíp UART của 8051.

1. Đầu dữ liệu sẵn sàng DTR: Khi thiết bị đầu cuối (hoặc một cổng COM của PC) được bật thì sau khi tự kiểm tra nó gửi một tín hiệu DTR báo rằng nó sẵn sàng cho truyền thông. Nếu có một cái gì đó trục trặc với cổng COM thì tín hiệu này không được kích hoạt. Đây là tín hiệu tích cực mức thấp và có thể được dùng để báo cho modem biết rằng máy tính đang hoạt động và đang sẵn sàng. Đây là chân đầu ra từ DTC (cổng COM của PC) và chân đầu ra của modem.
2. Dữ liệu sẵn sàng QSR: Khi DCE (chẳng hạn modem) được bật lên và đã chạy xong chương trình tự kiểm tra thì nó đòi hỏi DSR để báo rằng có đã sẵn sàng cho truyền thông. Do vậy, nó là đầu ra của modem (DCE) và đầu vào của PC (DTE). Đây là tín hiệu tích cực mức thấp. Nếu vì lý do nào đó mà modem không kích hoạt báo cho PC biết (hoặc thiết bị đầu cuối) rằng nó không thể nhận hoặc gửi dữ liệu.
3. Yêu cầu gửi RTS: Khi thiết bị DTE (chẳng hạn một PC) có một byte dữ liệu cần gửi thì nó yêu cầu RTS để báo cho modem biết rằng nó có một byte cần phải gửi đi. RTS là một đầu ra tích cực mức thấp từ DTE và một đầu vào tới modem.

4. Tín hiệu xáo để gửi CTS: Để đáp lại RTS thì khi modem có để chứa dữ liệu mà nó cần nhận thì nó gửi một tín hiệu CTS tới DTE (PC) để báo rằng bây giờ nó có thể nhận dữ liệu. Tín hiệu đầu vào này tới DTE dùng để khởi động việc truyền dữ liệu.
5. Tách tín hiệu mang dữ liệu DCD: Modem yêu cầu tín hiệu DCD báo cho DTE biết rằng đã tách được một tín hiệu mang dữ liệu hợp lệ và rằng kết nối giữa nó và modem khác đã được thiết lập. Do vậy, DCD là một đầu ra của modem và đầu vào của PC (DTE).
6. Báo chuông RI: Một đầu ra từ modem (DCE) và một đầu vào tới máy tính PC (DTE) báo rằng điện thoại đang báo chuông. Nó tắt và bật đồng bộ với âm thanh đang đổ chuông. Trong 6 tín hiệu bắt tay thì tín hiệu này là ít được dùng nhất do một thực tế là các modem đã chịu trách nhiệm về trả lời điện thoại. Tuy nhiên, nếu trong một hệ thống đã cho mà PC phải chịu trách nhiệm trả lời điện thoại thì tín hiệu này có thể được dùng.

Từ mô tả trên thì việc truyền thông PC và modem có thể được tóm tắt như sau: Trong khi các tín hiệu DTR và DSR được dùng bởi PC và modem để báo rằng chúng đang hoạt động tốt thì các tín hiệu RTS và CTS thực tế đang kiểm tra luồng dữ liệu. Khi PC muốn gửi dữ liệu thì nó yêu cầu RTS và đáp lại, nếu modem sẵn sàng (có chỗ chứa dữ liệu) để nhận dữ liệu thì nó gửi lại tín hiệu CTS. Còn nếu không có chỗ cho dữ liệu thì modem không kích hoạt CTS và PC thôi không yêu cầu DTR và thử lại. Các tín hiệu RTS và CTS cũng được cọi như tín hiệu luồng điều khiển phần cứng.

Đến đây kết thúc sự mô tả 9 chân quan trọng nhất của các tín hiệu bắt tay RS232 và các tín hiệu TxD, RxD và đất (Ground). Tín hiệu Ground này cũng được coi như là tín hiệu SG - đất của tín hiệu.

10.1.9 Các cổng COM của IBM PC và tương thích.

Các 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 các đầu nối kiểu 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 COM1 được dùng cho chuột và COM2 được dùng cho các thiết bị chẳng hạn như Modem. Chúng ta có thể nối cổng nối tiếp của 8051 đến cổng COM2 của một máy tính PC cho các thí nghiệm về truyền thông nối tiếp.

Với nền kiến thức về truyền thông nối tiếp này chúng ta đã sẵn sàng làm việc với 8051.

10.2 Nối ghép 8051 tới RS232.

Như đã nói ở phần 10.1, chuẩn RS232 không tương thích với mức lô-gíc TTL, do vậy nó yêu cầu một bộ điều khiển đường truyền chẳng hạn như chíp 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 các đầu nối RS232 thông qua chíp MAX232.

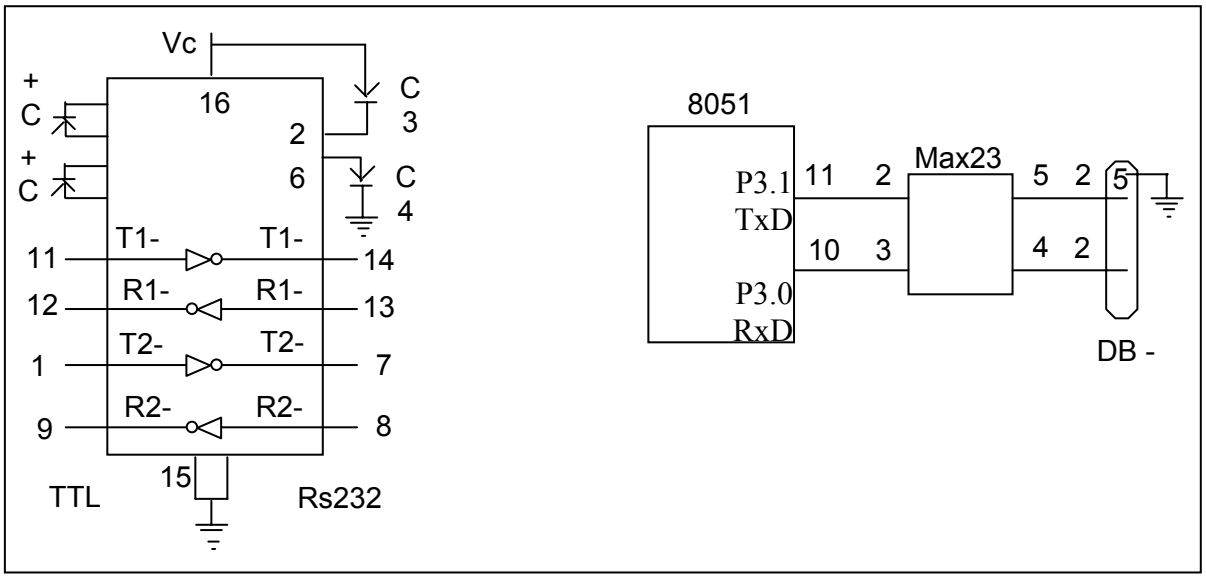
10.2.1 Các chân RxD và TxD trong 8051.

8051 có hai chân được dùng chuyên cho truyền và nhận dữ liệu nối tiếp. Hai chân này được gọi là TxD và RxD và là một phần của cổng P3 (đó là P3.0 và P3.1). chân 11 của 8051 là P3.1 được gán cho TxD và chân 10 (P3.0) được dùng cho RxD. Các chân này tương thích với mức lô-gích TTL. Do vậy chúng đòi hỏi một bộ điều khiển đường truyền để chúng tương thích với RS232. Một bộ điều khiển như vậy là chíp MAX232.

10.2.2 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 sẽ được chấp nhận bởi các chân TxD và RxD của 8051. Một ví dụ của một bộ chuyển đổi như vậy là chíp MAX232 từ hàng Maxim địa chỉ Website của hãng www.maxim-ic.com. Bộ MAX232 chuyển đổi từ các mức điện áp RS232 sẽ về mức điện áp TTL và ngược lại. Một điểm mạnh của chíp MAX232 là nó dùng điện áp nguồng +5v cùng với điện áp nguồn của 8051. Hay nóic cách khác với nguồn điện áp nuối +5 chúng ta mà có thể nuôi 8051 và MAX232 mà không phải dùng hai nguồn nuôi khác nhau như phổ biến trong các hệ thống trước đây.

Bộ điều khiển MAX232 có hai bộ điều khiển thường để nhận và truyền dữ liệu như trình bày trên hình 10.7. Các bộ điều khiển đường được 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 đối với TxD và RxD của 8051, còn cặp R2 và T2 thì chưa dùng đến. Để ý rằng trong MAX232 bộ điều khiển T1 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 của đầu nối DB của RS232 và chân $R1_{out}$ (chân số 12) là ở phía TTL mà nó được nối tới chân RxD của bộ vi điều khiển, xem hình 10.7. Để ý rằng nối ghép modem không là nối ghép mà chân TxD bên phát được nối với RxD của bên thu và ngược lại.



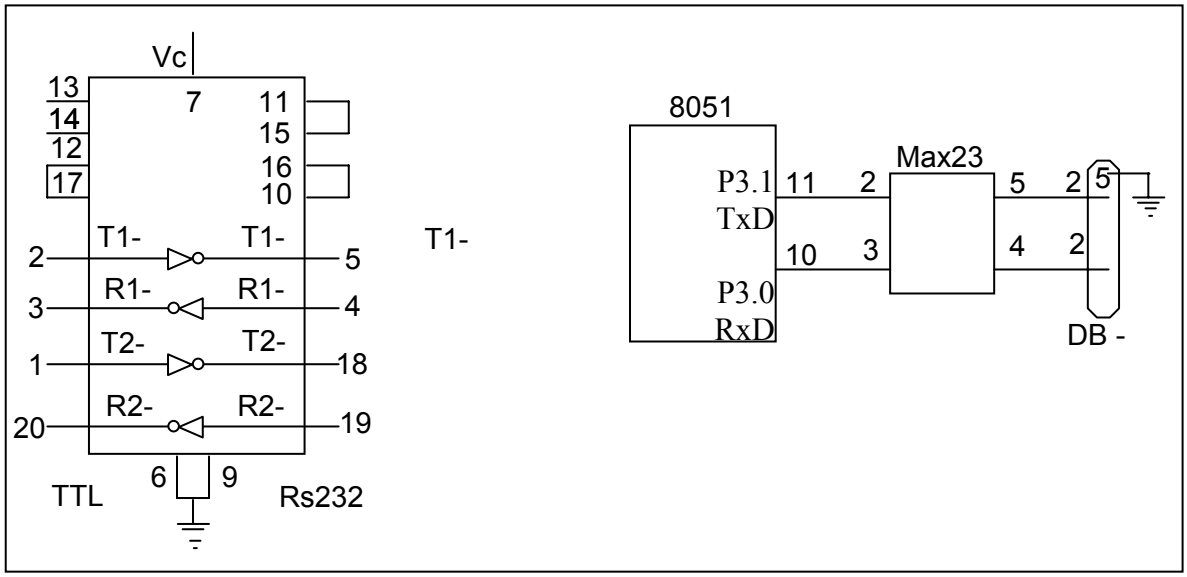
Hình 10.7: a) Sơ đồ bên trong của MAX232
b) Sơ đồ nối ghép của MAX232 với 8051 theo moden không.

Bộ MAX232 đòi hỏi 4 tụ điện giá trị từ 1 đến 22µF. Giá trị phổ biến nhất cho các tụ này là 22µF.

10.2.3 Bộ điều khiển MAX232.

Để tiết kiệm không gian trên bảng mạch, nhiều nhà thiết kế sử dụng chíp MAX232 từ hãng Maxim. Bộ điều khiển MAX232 thực hiện cùng những công việc như MAX232 lại không cần đến các tụ điện. Tuy nhiên, chíp MAX232 lại đắt hơn rất nhiều so với MAX233 không có sơ đồ chân giống nhau (không tương thích). Chúng

ta không thể lấy một chíp MAX232 ra khỏi một bảng mạch và thay vào đó RS233. Hãy xem hình 10.8 để thấy MAX233 không cần đến tụ.



Hình 10.8: a) Sơ đồ bên trong của MAX233.
b) Sờ đồ nối ghép của MAX233 với 8051 theo modem không.

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 chúng để truyền và nhận dữ liệu nối tiếp. Vì các máy tính IBM PC và tương thích được sử dụng rất rộng rãi để truyền thông với các hệ dựa trên 8051, do vậy ta chủ yếu tập trung vào truyền thông nối tiếp của 8051 với cổng COM của PC. Để cho phép truyền dữ liệu giữa máy tín PC và hệ thống 8051 mà không có bất kỳ lỗi nào thì chúng ta phải biết chắc rằng tốc độ baud của hệ 8051 phải phù hợp với tốc độ baud của công COM máy tính PC được cho trong bảng 10.3. Chúng ta có thể kiểm tra các tốc độ baud này bằng cách vào chương trình Windows Terminal và bấm chuột lên tuỳ chọn Communication Settings. Chương trình Terminal.exe của Window3.1 cũng làm việc tốt trên Windows95 và Window98. Trong Window95 và cao hơn ta có thể sử dụng chức năng Hyperterrminal. Hàm Hyperterminal hỗ trợ các tốc độ Baud cao hơn nhiều so với các tốc độ cho trong bảng 10.3.

Bảng 10.3: Các tốc độ Baud của máy tính PC486 và Pentium cho trong BIOS.

100	150	300	600	1200	2400	4800

Ví dụ 10.1:

Với tần số XTAL là 11.0592MHz. Hãy tìm giá trị TH1 cần thiết để có tốc độ baud sau:

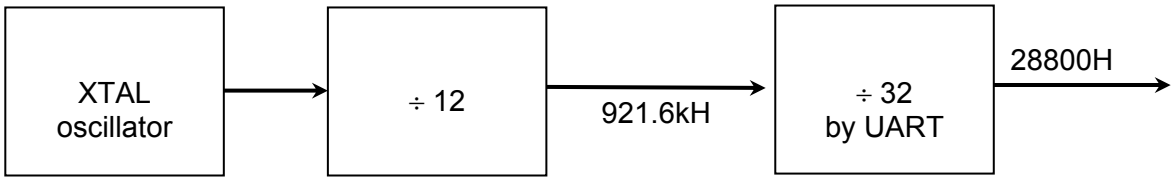
a) 9600 b) 2400 c) 1200

Lời giải:

Với tần số XTAL là 11.0592MHz thì ta có tần số chu trình máy của 8051 là 11.0592MHz : 12 = 921.6kHz và sau đó lấy 921.6kHz/32 = 28.800Hz là tần số được cấp bởi UART tới bộ định thời Timer1 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. Điều này sẽ được giải thích ở cuối chương.



10.3.1 Tốc độ baud trong 8051.

8051 truyền và nhận dữ liệu nối tiếp theo nhiều tốc độ khác nhau. Tốc độ truyền của nó có thể lập trình được. Điều ày thực hiện nhờ sự trợ giúp của bộ định thời Timer1. Trước khi ta đi vào bàn cách làm điều đó như thế nào thì ta sẽ xét quan hệ giữa tần số thạch anh và tốc độ baud trong 8051.

Như ta đã nói ở chương trước đây thì 8051 chia sóo thạch anh cho 12 để lấy tần số chu trình máy. Trong trường hợp XTAL = 11.0592MHz thì tần số chu trình là 921.6kHz (11.0592MHz : 12 = 921.6kHz). Mạch điện UART truyền thông nối tiếp của 8051 lại chia tần số chu trình máy cho 32 một lần nữa trước khi nó được dùng bởi bộ định thời gian Timer1 để tạo ra tốc độ baud. Do vậy, 921.6kHz : 32 = 28.800Hz. Đây là số ta sẽ dùng trong cả phần này để tìm giá trị của Timer1 để đặt tốc độ baud. Muốn Timer1 đặt tốc độ baud thì nó phải được lập trình về chế độ làm việc mode2, đó là chế độ thanh ghi 8 bít 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 10.3: Các giá trị của thanh ghi TH1 trong Timer1 cho các tốc độ baud khác nhau.

Tốc độ baud	TH1 (thập phân)	TH1 (số Hex)
9600	- 3	FD
4800	- 6	FA
2400	- 12	F4
1200	- 24	F8

10.3.2 Thanh ghi SBUF.

SBUF là thanh ghi 8 bít được dùng riêng cho truyền thông nối tiếp trong 8051. Đối với một byte dữ liệu cần phải được truyền qua đường TxD thì nó phải được đặt trong thanh ghi SBUF. Tương tự như vậy SBUF giữ một byte dữ liệu khi nó được nhận bở đường RxD của 8051. SBUF có thể được truy cập bởi mọi thanh ghi bất kỳ trong 8051. Xét một ví dụ dưới đây để thấy SBUF được truy cập như thế nào?

```
MOV   SBUF, # “D”    ; Nạp vào SBUF giá trị 44H mã ACSII của ký tự D.
MOV   SBUF, A              ; Sao thanh ghi A vào SBUF.
MOV   A, SBUF              ; Sao SBUF vào thanh ghi A.
```

Khi một byte được ghi vào thanh ghi SBUF nó được đóng khung với các bít Start và Stop và đường truyền nối tiếp quan chân TxD. Tương tự như vậy, khi các bít được nhận nối tiếp từ RxD thì 8051 mở khung nó để loại trừ các bít Start và Stop để lấy ra một byte từ dữ liệu nhận được và đặt nó vào thanh ghi SBUF.

10.3.3 Thanh ghi điều khiển nối tiếp SCON.

Thanh ghi SCON là thanh ghi 8 bít được dùng để lập trình việc đóng khung bít bắt đầu Start, bít dừng Stop và các bít dữ liệu cùng với việc khác.

Dưới đây là mô tả các bít khác nhau của SCON:

SM0	SM1	SM2	REN	TB8	RB8	T1

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 chế độ 1. đặt bằng phần cứng khi bắt đầu bít Stop ở
R1	SCON.0	Cờ ngắt thu Xoá bằng phần mềm.

Hình 10.2: Thanh ghi điều khiển cổng nối tiếp SCON.

10.3.3.1 Các bít SM0, SM1.

Đây là các bít D7 và D6 của thanh ghi SCON. Chúng được dùng để xác định chế độ đóng khung dữ liệu bằng cách xác định số bít của một ký tự và các bít 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 bít dữ liệu, Start, Stop
1	0	Chế độ nối tiếp 2
1	1	Chế độ nối tiếp 3

Trong bốn chế độ ta chỉ quan tâm đến chế độ 1, các chế độ khác được giải thích ở Appendisk A3. Trong thanh ghi SCON khi chế độ 1 được chọn thì dữ liệu được đóng khung gồm 8 bít dữ liệu, 1 bít Start, 1 bít Stop để tương thích với cổng COM của IBM PC và các PC tương thích khác. Quan trọng hơn là chế độ nối tiếp 1 cho phép tốc độ baud thay đổi và được thiết lập bởi Timer1 của 8051. Trong chế độ nối tiếp 1 thì mỗi ký tự gồm có 10 bít được truyền trong đó có bít đầu là bít Start, sau đó là 8 bít dữ liệu và cuối cùng là bít Stop.

10.3.3.2 Bít SM2.

Bít SM2 là bít D5 của thanh ghi SCON. Bít này cho phép khả năng đa xử lý của 8051 và nó nằm ngoài phạm vi trình bày của chương này. Đối với các ứng dụng của chúng ta đặt SM2 = 0 vì ta không sử dụng 8051 trong môi trường đa xử lý.

10.3.3.3 Bít REN.

Đây là bít cho phép thu (Receive Enable), bít D4 của thanh ghi SCON. Bít REN cũng được tham chiếu như là SCON.4 vì SCON là thanh ghi có thể đánh địa chỉ theo bít. Khi bít REN cao thì nó cho phép 8051 thu dữ liệu trên chân RxD của nó. Và kết quả là nếu ta muốn 8051 vừa truyền và nhận dữ liệu thì bít REN phải được đặt lên 1. Khi đặt REN thì bộ thu bị cấm. Việc đặt REN = 1 hay REN = 0 có thể đạt được

bằng lệnh "SETB SCON.4" và "CLR SCON.4" tương ứng. Lưu ý rằng các lệnh này sử dụng đặc điểm đánh địa chỉ theo bít của thanh ghi SCON. Bít này có thể được dùng để khống chế mọi việc nhận dữ liệu nối tiếp và nó là bít cực kỳ quan trọng trong thanh ghi SCON.

10.3.3.4 Bít TB8 và RB8.

Bít TB8 là bít SCON.3 hay là bít D3 của thanh ghi SCON. Nó được dùng để cho chế độ nối tiếp 2 và 3. Ta đặt TB8 vì nó không được sử dụng trong các ứng dụng của mình.

Bít RB8 (bít thu 8) là bít D2 của thanh ghi SCON. Trong chế độ nối tiếp 1 thì bít này nhận một bản sao của bít Stop khi một dữ liệu 8 bít được nhận. Bít này cũng như bít TB8 rất hiếm khi được sử dụng. Trong các ứng dụng của mình ta đặt RB8 = 0 vì nó được sử dụng cho chế độ nối tiếp 2 và 3.

10.3.3.5 Các bít TI và RI.

Các bít ngắt truyền TI và ngắt thu RI là các bít D1 và D0 của thanh ghi SCON. Các bít này là cực kỳ quan trọng của thanh ghi SCON. Khi 8051 kết thúc truyền một ký tự 8 bít thì nó bật TI để báo rằng nó sẵn sàng truyền một byte khác. Bít TI được bật lên trước bít Stop. Còn khi 8051 nhận được dữ liệu nối tiếp qua chân RxD và nó tách các bít Start và Stop để lấy ra 8 bít dữ liệu để đặt vào SBUF, sau khi hoàn tất nó bật cờ RI để báo rằng nó đã nhận xong một byte và cần phải lấy đi kẻo nó bị mất cờ RI được bật khi đang tách bít Stop. Trong các ví dụ dưới đây sẽ nói về vai trò của các bít TI và RI.

10.3.4 Lập trình 8051 để truyền dữ liệu nối tiếp.

Khi lập trình 8051 để truyền các byte ký tự nối tiếp thì cần phải thực hiện các bước sau đây:

1. Nạp thanh ghi TMOD giá trị 204 báo rằng sử dụng Timer1 ở chế độ 2 để thiết lập chế độ baud.
2. Nạp thanh ghi TH1 các giá trị cho trong bảng 10.4 để thiết lập chế độ baud truyền dữ liệu nối tiếp (với giả thiết tần số XTAL = 11.0592MHz).
3. Nạp thanh ghi SCON giá trị 50H báo chế độ nối tiếp 1 để đóng khung 8 bít dữ liệu, 1 bít Start và 1 bít Stop.
4. Bật TR1 = 1để khởi động Timer1.
5. Xoá bít TI bằng lệnh "CLR TI"
6. Byte ký tự cần phải truyền được ghi vào SBUF.
7. Bít cờ TI được hiển thị bằng lệnh "JNB TI, xx" để báo ký tự đã được truyền hoàn tất chưa.
8. Để truyền ký tự tiếp theo quay trở về bước 5.

Ví dụ 10.2 trình bày chương trình để 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 một ký tự "A" với tốc độ 4800 baud liên tục.

Lời giải:

```
        MOV   TMOD, #20H          ; Chọn Timer1, chế độ 2 (tự động nạp lại)
        MOV   TH1, # - 6          ; Chọn tốc độ 4800 baud
        MOV   SCON, #A"           ; Truyền 8 bít dữ liệu, 1 bít Stop cho phép thu
        SETB  TR1                 ; Khởi động Timer1
AGAIN:      MOV   SBUF, #"A"          ; Cần truyền ký tự "A"
HERE:       JNB   TI, HERE            ; Chờ đến bít cuối cùng
        CLR   TI                  ; Xoá bít TI cho ký tự kế tiếp
```

```
        SJMP  AGAIN                ; Tiếp tục gửi lại chữ A
```

Ví dụ 10.3:

Hãy viết chương trình để truyền chữ "YES" nối tiếp liên tục với tốc độ 9600 baud (8 bít dữ liệu, 1 bít Stop).

Lời giải:

```
        MOV   TMOD, #20H          ; Chọn bộ Timer1, chế độ 2
        MOV   TH1, # - 3          ; Chọn tốc độ 9600 baud
        MOV   SCON, #50H          ; Truyền 8 bít dữ liệu, 1 bít 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 cho đến khi truyền bít cuối cùng
                CLR   TI                ; Chờ sẵn cho một byte kế tiếp
                RET
```

10.3.4.1 Tầm quan trọng của cờ TI.

Để hiểu tầm quan trọng của cờ ngắt TI ta hãy xét trình tự các bước dưới đây mà 8051 phải thực hiện khi truyền một ký tự quan đường TxD:

1. Byte ký tự cần phải truyền được ghi vào SBUF.
2. Truyền bít Start
3. Truyền ký tự 8 bít lần lượt từng bít một.
4. Bít Stop được truyền xong, trong quá trình truyền bít Stop thì cờ TI được bật (TI = 1) bởi 8051 để báo sẵn sàng để truyền ký tự kế tiếp.
5. Bằng việ hiển thị cờ TI ta biết chắc rằng ta không nạp quá vào thanh ghi SBUF. Nếu ta nạp một byte vào SBUF trước ghi TI được bật thì phần dữ liệu của byte trước chưa truyền hết sẽ bị mất. Hay nói cách khác là 8051 bật cờ TI khi đã truyền xong một byte và nó sẵn sàng để truyền byte kế tiếp.
6. Sau khi SBOF được nạp một byte mới tì cờ nhằm để có thể truyền byte mới này.

Từ phần trình bày trên đây ta kết luận rằng bằng việc kiểm tra bít cờ ngắt TI ta biết được 8051 có sẵn sàng để truyền một byte khác không. Quan trọng hơn cần phải nói ở đây là bít cờ TI được bật bởi từ 8051 khi nó hoàn tất việc truyền một byte dữ liệu, còn việc xoá nó thì phải được lập trình viên thực hiện bằng lệnh "CLR TI". Cũng cần lưu ý rằng, nếu ta ghi một byte vào thanh ghi SBUF trước khi cờ TI được bật thì sẽ có nguy cơ mất phần dữ liệu đang truyền. Bít cờ TI có thể kiểm tra bằng lệnh "JNB TI …" hoặc có thể sử dụng ngắt như ta sẽ thấy trong chương 11.

10.3.5 Lập trình 8051 để nhận dữ liệu.

Trong lập trình của 8051 để nhận các byte ký tự nối tiếp thì phải thực hiện các bước sau đây.

1. Nạp giá trị 20H vào thanh ghi TMOD để báo sử dụng bộ Timer1, chế độ 2 (8 bítm, tự động nạp lại) để thiết lập tốc độ baud.
2. Nạp TH1 các giá trị cho trong bảng 10.4 để tạo ra tốc độ baud với giả thiết XTAL = 10.0592MHz.

3. Nạp giá trị 50H vào thanh ghi SCON để báo sử dụng chế độ truyền nối tiếp 1 là dữ liệu được đóng gói bởi 8 bít dữ liệu, 1 bít Start và 1 bít Stop.
4. Bật TR1 = 1 để khởi động Timer1.
5. Xoá cờ ngắt RI bằng lệnh "CLR RI"
6. Bít 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. Để nhận một ký tự tiếp theo quay trở về bước 5.

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 bít dữ liệu và 1 bít Stopl.

Lời giải:

```
        MOV   TMOD, #20H          ; Chọn bộ Timer1, chế độ 2 (tự động nạp lại)
        MOV   TH1, # - 6          ; Chọn tốc độ 4800 baud
        MOV   SCON, #50H          ; Chọn khung dữ liệu 8 bít Stop, bít.
        SETB  TR1                 ; Khởi động bộ Timer1
HERE:   JNB   R1, HERE                  ; Đợi nhận toàn bộ lý tự vào hết
        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 nhạn dữ liệu
```

Ví dụ 10.5:

Giả sử cổng nối tiếp của 8051 được nối vào cổng COM của máy tính IBM CP và mà đang sử chương trình Termina. Exe để gửi và nhận dữ liệu nối tiếp. Cổng P1 và P2 của 8051 được nố tới các đầ LED và các công tắc chuyển mạch tương ứng. Hãy viết một chương trrình cho 8051.

a) Gửi thông báo "We Are Ready" (chúng tôi đã sẵn sàng) tới máy tính PC.

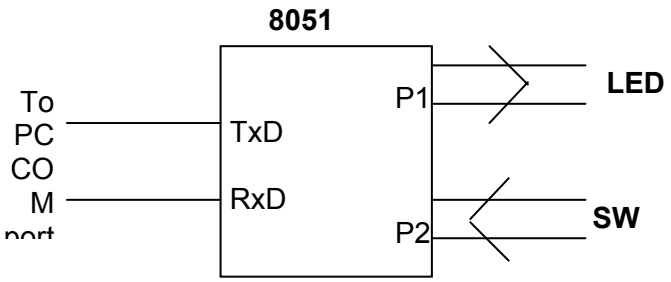
b) Nhận bất kỳ dữ liệu gì được PC gửi đến và chuyển đến các đèn LED đang nối đến các chân của cổng P1.

c) Nhận dữ liệu trên các chuyển mạch được nối tới P2 và gửi nó tới máy tính PC nối tiếp. Chương trình phải thực hiện một lần a), nhưng b) và c) chạy liên tục với tốc độ 4800 baud.

Lời giải:

```
        ORG   0
        MOV   P2, #0FFH           ; Lấy cổng P2 làm cổng vào
        MOV   TMOD, #20H          ; Chọn bộ Timer1, chế độ 2 (tự động nạp lại)
        MOV   TH1, # 0FAH         ; Chọn tốc độ 4800 baud
        MOV   SCON, #50H          ; Tạo khung dữ liệu 8 bít, 1bít 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 chương 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 nó nối tiếp
        ACALL RECV                ; Nhận dữ liệu nối tiếp
```

```
                MOV   F1, A                 ; Hiển thị nó ra các đền 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
; ------------------      Truyền dữ liệu nối tiếp ACC có dữ liệu
RECV:           JNB   RI, RECV              ; Nạp dữ liệu
                MOV   A, SBUF               ; ở lại đây cho đến khi gửi bít cuối cùng
                CLR   RI                    ; Sẵn sàng cho ký tự mới
                RET                         ; Trở về mời gọi
; ------------------      Nhận dữ liệu nối tiếp trong ACC
RECV:           JNB   RI, RECV              ; Đợi ở đây nhận ký tự
                MOV   A, SBUF               ; Lưu nó vào trong ACC
                CLR   RI                    ; Sẵn sàng nhận ký tự mã tiếp theo
                RET                         ; Trở về nời gọi
; --------------------      Ngăn xếp chưa thông báo
MYDATA:   DB    "Chúng tôi đã sẵn sàng" 0
                END
```



10.3.5.1 Tầm quan trọng của cờ RT.

Khi nhận các bít quan chân RxD của nó thì 8051 phải đi quan các bước sau:

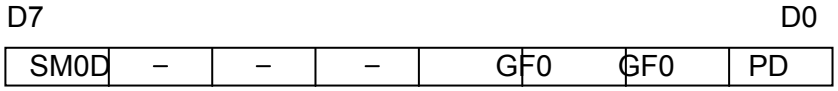
1. Nó nhận bít Start báo rằng bít sau nó là bít dữ liệu đầu tiên cần phải nhận.
2. Ký tự 8 bít được nhận lần lượt từng bít một. Khi bít cuối cùng được nhận thì một byte được hình thành và đặt vào trong SBUF.
3. Khi bít Stop được nhận thì 8051 bật RT = 1 để báo rằng toàn bộ ký tự được nhận và phải lấy đi trước khi nó bị byte mới nhận về ghi đè lên.
4. Bằng việc kiểm tra bít cờ RI khi nó được bật lên chúng ta biết rằng một ký tự đã được nhận và đang nằm trong SBUF. Tại sao nội dung SBUF vào nơi an toàn trong một thanh ghi hay bộ nhớ khác trước khi nó bị mất.
5. Sau khi SBUF được ghi vào nơi an toàn thì cờ RI được xoá về 0 bằng lệnh "CLR RI" nhằm cho các ký tự kế tiếp nhận được đưa vào SBUF. Nếu không làm được điều này thì gây ra mất ký tự vừa nhận được.

Từ mô tả trên đây ta rút ra kết luận rằng bằng việc kiểm tra cờ RI ta biết 8051 đã nhận được một byte ký tự chưa hay rồi. Nếu ta không sao được nội dung của thanh ghi SBUF vào nơi an toàn thì có nguy cơ ta bị mất ký tự vừa nhận được. Quan trọng hơn là phải nhớ rằng cờ RI được 8051 bật lên như lập trình viên phải xoá nó bằng lệnh "CLR RI". Cũng nên nhờ rằng, nếu ta sao nội dung SBUF vào nơi an toàn trước khi RI được bật ta mạo hiểm đã sao dữ liệu chưa đầy đủ. Bít cờ RI có thể được kiểm tra bởi lệnh "JNB RI, xx" hoặc bằng ngắt sẽ được bàn ở chương 11.

10.3.6 Nhân đôi tốc độ baud trong 8051.

Có hai cách để tăng tốc độ baud truyền dữ liệu trong 8051.

1. Sử dụng tần số thạch anh cao hơn.
2. Thay đổi một bít trong thanh ghi điều khiển công suất PCON (Power Control) như chỉ ra dưới đây.



Phương án một là không thực thi trong nhiều trường hợp vì tần số thạch anh của hệ thống là cố định. Quan trọng hơn là nó không khả thi vì tần số thạch anh mới không tương thích với tốc độ baud của các cổng COM nối tiếp của IBM PC. Do vậy, ta sẽ tập trung thăm dò phương án hai, có một cách nhân đôi tần số baud bằng phần mềm trong 8051 với tần số thạch anh không đổi. Điều này được thực hiện nhờ thanh ghi PCON, đây là thanh ghi 8 bít. Trong 8 bít này thì có một số bít không được dùng để điều khiển công suất của 8051. Bít dành cho truyền thông là D7, bít SMOD (chế độ nối tiếp - serial mode). Khi 8051 được bật nguồn thì bít SMOD của thanh ghi PCON ở mức thấp 0. Chúng ta có thể đặt nó lên 1 bằng phần mềm và do vậy nhân đôi được tốc độ baud. Thứ tự các lệnh được sử dụng để thiết lập bít D7 của PCON lên cao như sau (thanh ghi PCON là thể đánh địa chỉ theo bít).

```
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 mà không thay đổi bất kỳ bít nào khác.
```

Để biết tốc độ baud được tăng lên gấp đôi như thế nào bằng phương pháp này ta xét vai trò của bít SMOD trong PCON khi nó là 0 và 1.

a) Khi SMOD = 0.

Khi SMOD = 0 thì 8051 chia 1/12 tần số thạch anh cho 32 và sử dụng nó cho bộ Timer1 để thiết lập tốc độ baud. Trong 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 được Timer1 sử dụng để đặt tốc độ baud. Đây là cơ sở cho tất cả ví dụ từ trước đến giờ vì nó là giá trị mặc định của 8051 khi bật nguồn. Các tốc độ baud đối với SMOD = 0 được cho trong bảng 10.4.

b) Khi SMOD = 1.

Với tần số cố định thạch anh ta có thể nhân đôi tốc độ baud bằng cách đặt bít SMOD = 1. Khi bít D7 của PCON (bít SMOD) được đưa lên 1 thì 1/12 tần số XTAL được chia cho 16 (thay vì chia cho 32 như khi SMOD = 0) và đây là tần số được Timer 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.

Đây là tần số mà Timer1 dùng để đặt tốc độ baud. Bảng 10.5 là các giá trị cần được nạp vào TH1 cùng với các tốc độ baud của 8051 khi SMOD = 0 và 1.

Bảng 10.5: So sánh tốc độ baud khi SMOD thay đổi.

TH1 (thập phân)	TH1 (Hex)	Tốc độ baud	
		SMOD = 0	SMOD = 1
-3	FD	9600	19200

-6	DA	4800	9600
-12	F4	2400	4800
-24	E8	1200	2400

Ví dụ 10.6:

Giả sử tần số XTAL = 11.0592MHz cho chương trình dưới đây, hãy phát biểu a) chương trình này làm gì? b) hãy tính toán tần số được Timer1 sử dụng để đặt tốc độ baud? và c) hãy tìm tốc độ baud truyền dữ liệu.

```
        MOV   A, PCON       ; Sao nội dung thanh ghi PCON vào thanh ghi ACC
        SETB  ACC.7         ; Đặt D7 = 0
        MOV   PCON, A       ; Đặt SMOD = 1 để tăng gấp đôi tần số baud với tần số XTAL
cố định
                            ;
        MOV   TMOD, #20H    ; Chọn bộ Timer1, chế độ 2, tự động nạp lại
        MOV   TH1, - 3              ; Chọn tốc độ baud 19200 (57600/3=19200) vì SMOD
= 1
                            ;
        MOV   SCON, #50H    ; Đóng khung dữ liệu gồm 8 bít dữ liệu, 1 Stop và cho phép
RI.
        SETB  TR1           ; Khởi động Timer1
        MOV   A, #"B"       ; Truyền ký tự B
 A-1:   CLR   TI            ; Khẳng định TI = 0
        MOV   SBUF, A               ; Truyền nó
 H-1:   JNB   TI, H-1       ; Chờ ở đây cho đến khi bít cuối được gửi đi
        SJMP  A-1           ; Tiếp tục gửi "B"
```

Lời giải:

a) Chương trình này truyền liên tục mã ASCII của chữ B (ở dạng nhị phân là 0100 0010)

b) Với tần số XTAL = 11.0592MHz và SMOD = 1 trong chương trình trên 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:

Tìm giá trị TH1 (ở dạng thập phân và hex) để đạt tốc độ baud cho các trường hợp sau.

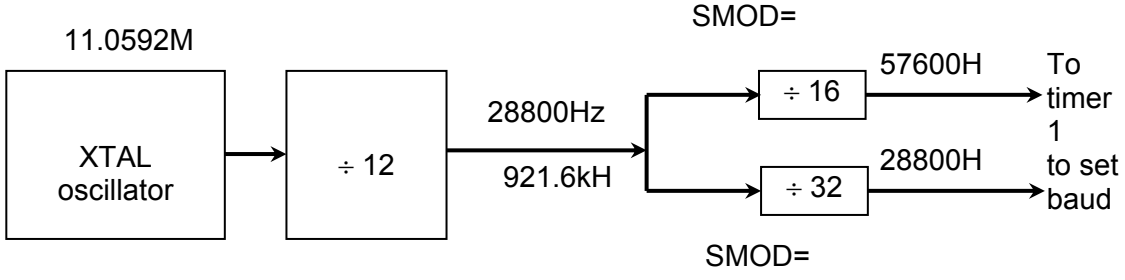
a) 9600 b) 4800 nếu SMOD = 1 và tần số XTAL = 11.0592MHz

Lời giải:

Với tần số XTAL = 11.0592MHz và SMOD = 1 ta có tần số cấp cho Timer1 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 hỗ trợ bởi các máy tính IBM PC và tương thích không?

Lời 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 hỗ trợ bởi các máy tính IBM PC và tương thích. Tuy nhiên, PC có thể được lập trình để truyền dữ liệu với tốc độ như vậy. Phần mềm của nhiều modem có thể làm cho điều này và Hyperterminal của Windows 95 cũng có thể hỗ trợ tốc độ này và các tốc độ khác nữa.

10.3.7 Truyền dữ liệudựa trên các ngắt.

Ta phải thấy rằng thật lãng phí để các bộ vi điều khiển phải bật lên xuống các cờ TI và RI. Do vậy, để tăng hiệu suất của 8051 ta có thể lập trình các cổng truyền thông nối tiếp của nó bằng các ngắt. Đây chính là nội dung chính sẽ bàn luận ở chương 11 dưới đây.

CHƯƠNG 11
Lập trình các ngắt

Một ngắt là một sự kiện bên trong hoặc bên ngoài làm ngắt bộ vi điều khiển để báo cho nó biết rằng thiết bị cần dịch vụ của nó. Trong chương này ta tìm hiểu khái niệm ngắt và lập trình ngắt.

11.1 Các ngắt của 8051.

11.1.1 Các ngắt ngược với thăm dò.

Một bộ vi điều khiển có thể phục vụ một vài thiết bị, có hai cách để thực hiện điều này đó là sử dụng các ngắt và thăm dò (polling). Trong phương pháp sử dụng các ngắt thì mỗi khi có một thiết bị bất kỳ cần đến dịch vụ của nó thì nó bao cho bộ vi điều khiển bằng cách gửi một tín hiệu ngắt. Khi nhận được tín hiệu ngắt thì bộ vi điều khiển ngắt tất cả những gì nó đang thực hiện để chuyển sang phục vụ thiết bị. Chương trình đi cùng với ngắt được gọi là trình dịch vụ ngắt ISR (Interrupt Service Routine) hay còn gọi là trình quản lý ngắt (Interrupt handler). Còn trong phương pháp thăm dò thì bộ vi điều khiển hiển thị liên tục tình trạng của một thiết bị đã cho và điều kiện thoả mãn thì nó phục vụ thiết bị. Sau đó nó chuyển sang hiển thị tình trạng của thiết bị kế tiếp cho đến khi tất cả đều được phục vụ. Mặc dù phương pháp thăm dò có thể hiển thị tình trạng của một vài thiết bị và phục vụ mỗi thiết bị khi các điều kiện nhất định được thoả mãn nhưng nó không tận dụng hết cộng dụng của bộ vi điều khiển. Điểm mạnh của phương pháp ngắt là bộ vi điều khiển có thể phục vụ được rất nhiều thiết bị (tất nhiên là không tại cùng một thời điểm). Mỗi thiết bị có thể nhận được sự chú ý của bộ vi điều khiển dựa trên mức ưu tiên được gán cho nó. Đối với phương pháp thăm dò thì không thể gán mức ưu tiên cho các thiết bị vì nó kiểm tra tất cả mọi thiết bị theo kiểu hơi vòng. Quan trọng hơn là trong phương pháp ngắt thì bộ vi điều khiển cũng còn có thể che hoặc làm lơ một yêu cầu dịch vụ của thiết bị. Điều này lại một lần nữa không thể thực hiện được trong phương pháp thăm dò. Lý do quan trọng nhất là phương pháp ngắt được ưu chuộng nhất là vì phương pháp thăm dò làm lãng phí thời gian của bộ vi điều khiển bằng cách hỏi dò từng thiết bị kể cả khi chúng không cần đến dịch vụ. Nhằm để tránh ….. thì người ta sử dụng phương pháp ngắt. Ví dụ trong các bộ định thời được bàn đến ở chương 9 ta đã dùng lệnh "JNB TF, đích" và đợi cho đến khi bộ định thời quay trở về 0. Trong ví dụ đó, trong khi chờ đợi thì ta có thể làm việc được gì khác có ích hơn, chẳng hạn như khi sử dụng phương pháp ngắt thì bộ vi điều khiển có thể đi làm các việc khác và khi cờ TF bật lên nó sẽ ngắt bộ vi điều khiển cho dù nó đang làm bất kỳ điều gì.

11.1.2 Trình phục vụ ngắt.

Đối với mỗi ngắt thì phải có một trình phục vụ ngắt ISR hay trình quản lý ngắt. khi một ngắt được gọi thì bộ vi điều khiển phục vụ ngắt. Khi một ngắt được gọi thì bộ vi điều khiển chạy trình phục vụ ngắt. Đối với mỗi ngắt thì có một vị trí cố định trong bộ nhớ để giữ địa chỉ ISR của nó. Nhóm các vị trí nhớ được dành riêng để gửi các địa chỉ của các ISR được gọi là bảng véc tơ ngắt (xem hình 11.1).

11.1.3 Các bước khi thực hiện một ngắt.

Khi kích hoạt một ngắt bộ vi điều khiển đi qua các bước sau:

1. Nó kết thúc lệnh đang thực hiện và lưu địa chỉ của lệnh kế tiếp (PC) vào ngăn xếp.
2. Nó cũng lưu tình trạng hiện tại của tất cả các ngắt vào bên trong (nghĩa là không lưu vào ngăn xếp).
3. Nó nhảy đến một vị trí cố định trong bộ nhớ được gọi là bảng véc tơ ngắt nới lưu giữ địa chỉ của một trình phục vụ ngắt.
4. Bộ vi điều khiển nhận địa chỉ ISR từ bảng véc tơ ngắt và nhảy tới đó. Nó bắt đầu thực hiện trình phục vụ ngắt cho đến lệnh cuối cùng của ISR là RETI (trở về từ ngắt).
5. Khi thực hiện lệnh RETI bộ vi điều khiển quay trở về nơi nó đã bị ngắt. Trước hết nó nhận địa chỉ của bộ đếm chương trình PC từ ngăn xếp bằng cách kéo hai byte trên đỉnh của ngăn xếp vào PC. Sau đó bắt đầu thực hiện các lệnh từ địa chỉ đó.

Lưu ý ở bước 5 đến vai trò nhạy cảm của ngăn xếp, vì lý do này mà chúng ta phải cẩn thận khi thao tác các nội dung của ngăn xếp trong ISR. Đặc biệt trong ISR cũng như bất kỳ chương trình con CALL nào số lần đẩy vào ngăn xếp (Push) và số lần lấy ra từ nó (Pop) phải bằng nhau.

11.1.4 Sáu ngắt trong 8051.

Thực tế chỉ có 5 ngắt dành cho người dùng trong 8051 nhưng nhiều nhà sản xuất đưa ra các bảng dữ liệu nói rằng có sáu ngắt vì họ tính cả lệnh tái thiết lập lại RESET. Sáu ngắt của 8051 được phân bố như sau:

1. RESET: Khi chân RESET được kích hoạt từ 8051 nhảy về địa chỉ 0000. Đây là địa chỉ bật lại nguồn được bàn ở chương 4.
2. Gồm hai ngắt dành cho các bộ định thời: 1 cho Timer0 và 1 cho Timer1. Địa chỉ của các ngắt này là 000B4 và 001B4 trong bảng véc tơ ngắt dành cho Timer0 và Timer1 tương ứng.
3. Hai ngắt dành cho các ngắt phần cứng bên ngoài chân 12 (P3.2) và 13 (P3.3) của cổng P3 là các ngắt phần cứng bên ngoài INT0 và INT1 tương ứng. Các ngắt ngoài cũng còn được coi như EX1 và EX2 vị trí nhớ trong bảng véc tơ ngắt của các ngắt ngoài này là 0003H và 0013H gán cho INT0 và INT1 tương ứng.
4. Truyền thông nối tiếp có một ngắt thuộc về cả thu và phát. Địa chỉ của ngắt này trong bảng véc tơ ngắt là 0023H.

Chú ý rằng trong bảng 11.1 có một số giới hạn các byte dành riêng cho mỗi ngắt. Ví dụ, đối với ngắt INT0 ngắt phần cứng bên ngoài 0 thì có tổng cộng là 8 byte từ địa chỉ 0003H đến 000AH dành cho nó. Tương tự như vậy, 8 byte từ địa chỉ 000BH đến 0012H là dành cho ngắt bộ định thời 0 là TI0. Nếu trình phục vụ ngắt đối mặt với một ngắt đã cho mà ngắn đủ đặt vừa không gian nhớ được. Nếu không vừa thì một lệnh LJMP được đặt vào trong bảng véc tơ ngắt để chỉ đến địa chỉ của ISR, ở trường hợp này thì các byte còn lại được cấp cho ngắt này không dùng đến. Dưới đây là các ví dụ về lập trình ngắt minh hoạ cho các điều trình bày trên đây.

Từ bảng 11.1 cùng để ý thấy một thực tế rằng chí có 3 byte của không gian bộ nhớ ROM được gán cho chân RESET. Đó là những vị trí địa chỉ 0, 1 và 2 của ROM. Vị trí địa chỉ 3 thuộc về ngắt phần cứng bên ngoài 0 với lý do này trong chương trình chúng ta phaỉ đặt lệnh LJMP như là lệnh đầu tiên và hướng bộ xử lý lệnh khỏi bảng véc tơ ngắt như chỉ ra trên hình 11.1.

Bảng 11.1: Bảng véc tơ ngắt của 8051.

Ngắt	Địa chỉ ROM	Chân
Bật lại nguồn (RESET)	0000	9
Ngắt phần cứng ngoài (INT0)	0003	12 (P3.2)
Ngắt bộ Timer0 (TF0)	000B	
Ngắt phần cứng ngoài 1 (INT1)	0013	13 (P3.3)
Ngắt bộ Timer1 (TF1)	001B	
Ngắt COM nối tiếp (RI và TI)	0023	

11.1.5 Cho phép và cấm ngắt.

Khi bật lại nguồn thì tất cả mọi ngắt đều bị cấm (bị che) có nghĩa là không có ngắt nào sẽ được bộ vi điều khiển đáp ứng nếu chúng được kích hoạt. Các ngắt phải được kích hoạt bằng phần mềm để bộ vi điều khiển đáp ứng chúng. 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 trình bày thanh ghi IE, lưu ý rằng IE là thanh ghi có thể đánh địa chỉ theo bít.

Từ hình 11.2 ta thấy rằng D7 của thanh ghi IE được gọi là bít cho phép tất cả các ngắt EA (Euable All). Bít này phải được thiết lập lên 1 để phần còn lại của thanh ghi hoạt động được. Bít D6 chưa được sử dụng. Bít D54 được dành cho 8051, còn bít D4 dùng cho ngắt nối tiếp v.v…

11.1.6 Các bước khi cho phép ngắt.

Để cho phép một ngắt ta phải thực hiện các bước sau:

1. Bít D7 của thanh ghi IE là EA phải được bật lên cao để cho phép các bít còn lại của thanh ghi nhận được hiệu ứng.
2. Nếu EA = 1 thì tất cả mọi ngắt đều được phép và sẽ được đáp ứng nếu các bít tương ứng của chúng trong IE có mức cao. Nếu EA = 0 thì không có ngắt nào sẽ được đáp ứng cho dù bít tương ứng của nó trong IE có giá trị cao.

Để hiểu điểm quan trong này hãy xét ví dụ 11.1.

Hình 11.2: Thanh ghi cho phép ngắt IE.

D7 D0

EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA IE.7 Nếu EA = 0 thì mọi ngắt bị cấm
Nếu EA = 1 thì mỗi nguồn ngắt được cho phép hoặc bị cấm bằng các bật hoặc xoá bít cho phép của nó.

- - IE.6 Dự phòng cho tương lai
ET2 IE.5 Cho phép hoặc cấm ngắt tràn hoặc thu của Timer2 (8051)
ES IE.4 Cho phép hoặc cấm ngắt cổng nối tiếp
ET1 IE.3 Cho phép hoặc cấm ngắt tràn của Timer1
EX1 IE.2 Cho phép hoặc cấm ngắt ngoài 1

ET0 IE.1 Cho phép hoặc cấm ngắt tràn của Timer0
EX0 IE.0 Cho phép hoặc cấm ngắt ngoài 0

* Người dùng không phải ghi 1 vào bít dự phòng này. Bít này có thể dùng cho các bộ vi điều khiển nhanh với đặc tính mới

Ví dụ 11.1:

Hãy chỉ ra những lệnh để a) cho phép ngắt nối tiếp ngắt Timer0 và ngắt phần cứng ngoài 1 (EX1) và b) cấm (che) ngắt Timer0 sau đó c) trình bày cách cấm tất cả mọi ngắt chỉ bằng một lệnh duy nhất.

Lời giải:

```
a) MOV    IE, #10010110B    ; Cho phép ngắt nối tiếp, cho phép ngắt Timer0 và cho phép ngắt phần cứng ngoài.
```

Vì IE là thanh ghi có thể đánh địa chỉ theo bít nên ta có thể sử dụng các lệnh sau đây để truy cập đến các bít 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 Timer1
SETB  IE.2      ; Cho phép ngắt phần cứng ngoài 1
```

(tất cả những lệnh này tương đương với lệnh "MOV IE, #10010110B" trên đây).

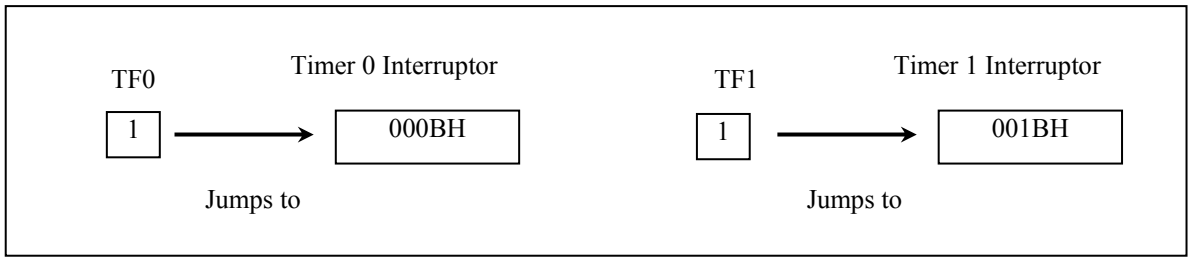
```
b) CLR IE.1     ; Xoá (che) ngắt Timer0
c) CLR IE.7     ; Cấm tất cả mọi ngắt.
```

11.2 Lập trình các ngắt bộ định thời.

Trong chương 9 ta đã nói cách sử dụng các bộ định thời Timer0 và Timer1 bằng phương pháp thăm dò. Trong phần này ta sẽ sử dụng các ngắt để lập trình cho các bộ định thời của 8051.

11.2.1 Cờ quay về 0 của bộ định thời và ngắt.

Trong chương 9 chúng ta đã nói rằng cờ bộ định thời TF được đặt lên cao khi bộ định thời đạt giá trị cực đại và quay về 0 (Roll - over). Trong chương trình này chúng ta cũng chỉ ra cách hiển thị cờ TF bằng lệnh "JNB TF, đích". Khi thăm dò cờ TF thì ta phải đợi cho đến khi cờ TF được bật lên. Vấn đề với phương pháp này là bộ vi điều khiển bị trói buộc khi cờ TF được bật lên và không thể làm được bất kỳ việc gì khác. Sử dụng các ngắt giải quyết được vấn đề này và tránh được sự trói buộc của bộ vi điều khiển. Nếu bộ ngắt định thời trong thanh ghi IE được phép thì mỗi khi nó quay trở về 0 cờ TF được bật lên và bộ vi điều khiển bị ngắt tại bất kỳ việc gì nó đang thực hiện và nhảy tới bảng véc tơ ngắt để phục vụ ISR. Bằng cách này thì bộ vi điều khiển có thể làm những công việc khác cho đến khi nào nó được thông báo rằng bộ định thời đã quay về 0. Xem hình 11.3 và ví dụ 11.2.



Hình 11.3: Ngắt bộ định thời TF0 và TF1.

Hãy để những điểm chương trình dưới đây của chương trình trong ví dụ 11.2.

1. Chúng ta phải tránh sử dụng không gian bộ nhớ dành cho bảng véc tơ ngắt. Do vậy, ta đặt tất cả mã khởi tạo tại địa chỉ 30H của bộ nhớ. Lệnh LJMP là lệnh đầu tiên mà 8051 thực hiện khi nó được cấp nguồn. Lệnh LJMP lái bộ điều khiển tránh khỏi bảng véc tơ ngắt.
2. Trình phục vụ ISR của bộ Timer0 được đặt ở trong bộ nhớ bắt đầu tự địa chỉ 000BH và vì nó quá nhỏ đủ cho vào không gian nhớ dành cho ngắt này.
3. Chúng ta cho phép ngắt bộ Timer0 với lệnh "MOV IE, #1000 010H" trong chương trình chính MAIN.
4. Trong khi dữ liệu ở cổng P0 được nhận vào và chuyển liên tục sang công việc P1 thì mỗi khi bộ Timer0 trở về 0, cờ TF0 được bật lên và bộ vi điều khiển thoát ra khỏi vòng lặp BACK và đi đến địa chỉ 000BH để thực hiện ISR gắn liền với bộ Timer0.
5. Trong trình phục vụ ngắt ISR của Timer0 ta thấy rằng không cần đến lệnh "CLR TF0" trước khi lệnh RETI. Lý do này là vì 8051 xoá cờ TF bên trong khi nhảy đến bảng véc tơ ngắt.

Ví dụ 11.2:

Hãy viết chương trình nhân liên tục dữ liệu 8 bít ở cổng P0 và gửi nó đến cổng P1 trong khi nó cùng lúc tạo ra một sóng vuông chu kỳ 200μs trên chân P2.1. Hãy sử dụng bộ Timer0 để tạo ra sóng vuông, tần số của 8051 là XTAL = 11.0592MHz.

Lời giải:

Ta sử dụng bộ Timer0 ở chế độ 2 (tự động nạp lại) giá trị nạp cho TH0 là 100/1.085μs = 92.

```
; - - Khi khởi tạo vào chương trình main tránh dùng không gian.
; Địa chỉ dành cho bảng véc tơ ngắt.
            ORG   0000H
            CPL   P2.1              ; Nhảy đến bảng véc tơ ngắt.
;
; - - Trình ISR dành cho Timer0 để tạo ra sóng vuông.
            ORG   0030H             ; Ngay sau địa chỉ bảng véc-tơ ngắt
MAIN:       TMOD, #02H              ; Chọn bộ Timer0, chế độ 2 tự nạp lại
            MOV   P0, #0FFH         ; Lấy P0 làm cổng vào nhận dữ liệu
            MOV   TH0, # - 92       ; Đặt TH0 = A4H cho – 92
            MOV   IE, #82H                  ; IE = 1000 0010 cho phép Timer0
            SETB  TR0               ; Khởi động bộ Timer0
BACK:             MOV   A, P0               ; Nhận dữ liệu vào từ cổng P0
            MOV   P1, A             ; Chuyển dữ liệu đến cổng P1
            SJMP  BACK              ; Tiếp tục nhận và chuyển dữ liệu
                                    ; Chừng nào bị ngắt bởi TF0
            END
```

Trong ví dụ 11.2 trình phục vụ ngắt ISR ngắn nên nó có thể đặt vừa vào không gian địa chỉ dành cho ngắt Timer0 trong bảng véc tơ ngắt. Tất nhiên không phải lúc nào cũng làm được như vậy. Xét ví dụ 11.3 dưới đây.

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 với giả thiết tần số XTAL = 11.0592MHz. Hãy sử dụng bộ định thời Timer1.

Lời giải:

Vì 1085μs là 1000 × 1085μs nên ta cần sử dụng chế độ 1 của bộ định thời Timer1.

```
; - - Khi khởi tạo tránh sử dụng không gian dành cho bảng véc tơ ngắt.
            ORG   0000H
            LJMP  MAIN              ; Chuyển đến bảng véc tơ ngắt.
;
; - - Trình ISR đối với Timer1 để tạo ra xung vuông
            OR6   001BH             ; Địa chỉ ngắt của Timer1 trong bảng véc tơ
ngắt
            LJMP  ISR-T1            ; Nhảy đến ISR
;
; - - Bắt đầu các chương trình chính MAIN.
            ORG   0030H             ; Sau bảng véc tơ ngắt
MAIN:       MOV   TMOD, #10H        ; Chọn Timer1 chế độ 1
            MOV   P0, #0FFH         ; Chọn cổng P0 làm đầu vào nhận dữ liệu
            MOV   TL1, #018H        ; Đặt TL1 = 18 byte thấp của - 1000
            MOV   TH1, #0FCH        ; Đặt TH1 = FC byte cao của  - 1000
            MOV   IE, #88H                  ; IE = 10001000 cho phép ngắt Timer1
            SETB  TR1               ; Khởi động bộ Timer1
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 Timer1 phải được nạp lại vì ở chế độ 1
ISR-T1:           CLR   TR1                 ; Dừng bộ Timer1
            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 Timer1    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 mức MC và mỗi MC = 1.085μs và 14 × 1.085μs = 15.19μs.

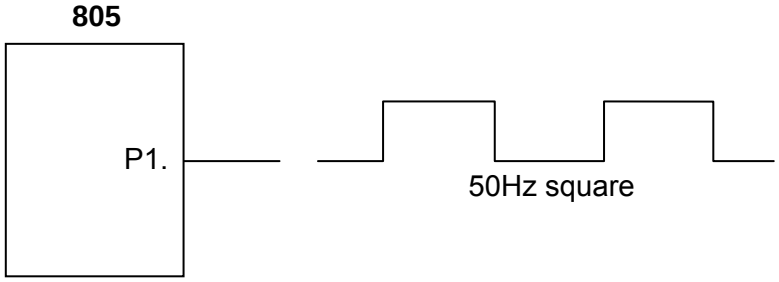
Ví dụ 11.4:

Viết một chương trình để tạo ra một sóng vuông tần số 50Hz trên chân P1.2. Ví dụ này tương tự ví dụ 9.12 ngoại trừ ngắt Timer0, giả sử XTAL = 11.0592MHz.

Lời giải:

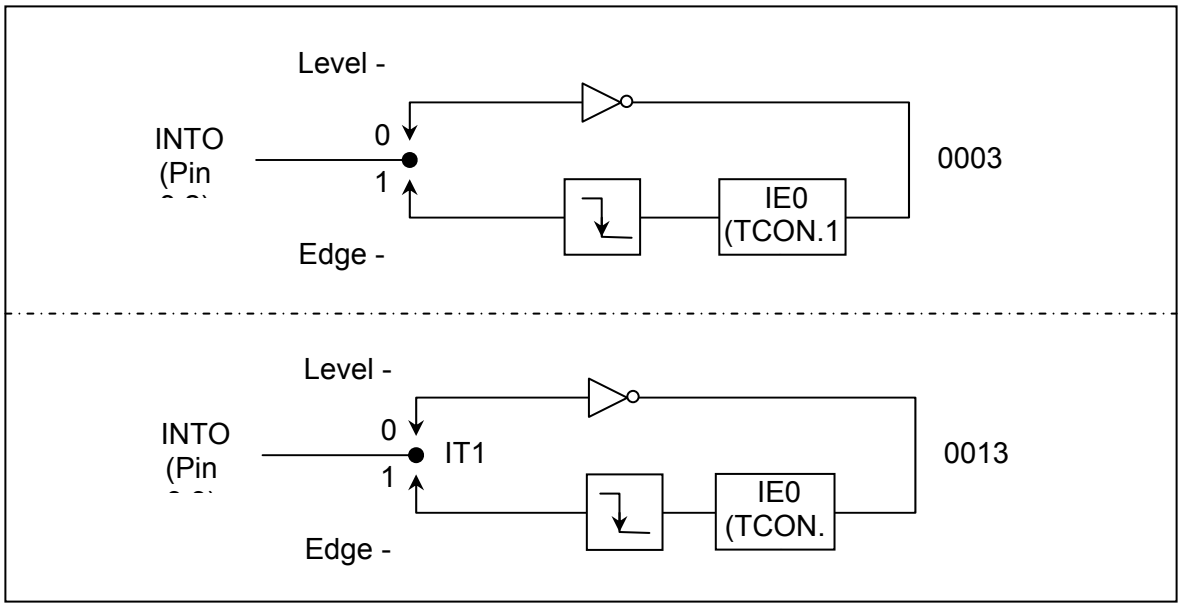
```
                ORG   0
                LJMP  MAIN
                ORG   000BH                ; Chương trình con phục vụ ngắt cho Timer0

                CPL   P1.2
                MOV   TL0, # 00
                MOV   TH0, # 0DCH
                RETI
                ORG   30H
; ------------ main  program for initialization
MAIN:           MOV   TMOD, # 00000001B        ; Chọn Timer0 chế độ 1
                MOV   TL0, # 0DCH
                MOV   IE, # 82H            ; Cho phép ngắt Timer0
                SETB  TR0
HERE:           SJMP  HERE
                END
```



11.3 Lập trình các ngắt phần cứng bên 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 bị ngắt tại bất kỳ công việc nào mà nó đang thực hiện và nó nhảy đến bảng véc tơ ngắt để thực hiện trình phục vụ ngắt.



11.3.1 Các ngắt ngoài INT0 và INT1.

Chỉ có hai ngắt phần cứng ngoài trong 8051 là INT0 và INT1. Chúng được bố trí trên chân P3.2 và P3.3 và địa chỉ của chúng trong bảng véc tơ ngắt là 0003H và 0013H. Như đã nói ở mục 11.1 thì chúng được ghép và bị cấm bằng việc sử dụng thanh ghi IE. Vậy chúng được kích hoạt như thế nào? Có hai mức kích hoạt cho các ngắt phần cứng ngoài: Ngắt theo mức và ngắt theo sườn. Dưới đây là mô tả hoạt động của mỗi loại.

11.3.2 Ngắt theo mức.

Ở chế độ ngắt theo mức thì các chân INT0 và INT1 bình thường ở mức cao (giống như tất cả các chân của cổng I/O) và nếu một tín hiệu ở mức thấp được cấp tới chúng thì nó ghi nhãn ngắt. Sau đó bộ vi điều khiển dừng tất cả mọi công việc nó đang thực hiện và nhảy đến bảng véc tơ ngắt để phục vụ ngắt. Điều này được gọi là ngắt được kích hoạt theo mức hay ngắt theo mức và là chế độ ngắt mặc định khi cấp nguồn lại cho 8051. Tín hiệu mức thấp tại chân INT phải được lâýu đi trước khi thực hiện lệnh cuối cùng của trình phục vụ ngắt RETI, nếu không một ngắt khác sẽ lại được tạo ra. Hay nói cách khác, nếu tín hiệu ngắt mức thấp không được lấy đi khi ISR kết thúc thì nó không thể hiện như một ngắt khác và 8051 nhảy đến bảng véc tơ ngắt để thực hiện ISR. Xem ví dụ 11.5.

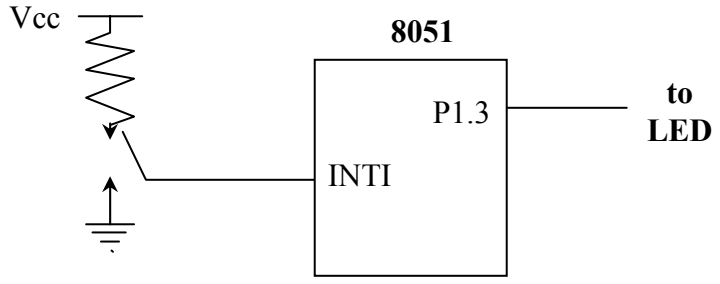
Ví dụ 11.5.

Giả sử chân INT1 được nối đến công tắc bình thường ở mức cao. Mỗi khi nó xuống thấp phải bật một đèn LED. Đèn LED được nối đến chân P1.3 và bình thường ở chế độ tắt. Khi nó được bật lên nó phải sáng vài phần trăm giây. Chừng nào công tắc được ấn xuống thấp đèn LED phải sáng liên tục.

Lời giải:

```
            ORG   0000H
            LJMP  MAIN                ; Nhảy đến bảng véc tơ ngắt
; - - Chương trình con ISR cho ngắt cứng INT1 để bật đèn LED.
            ORG   0013H               ; Trình phục vụ ngắt 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 từ ISR
; - - Bắt đầu chương trình chính Main.
            ORG   30H
MAIN:       MOV   IE, #10000100B          ; Cho phép ngắt dài
            SJMP  HERE                ; Chờ ở đây cho đến khi được ngắt
      END
```

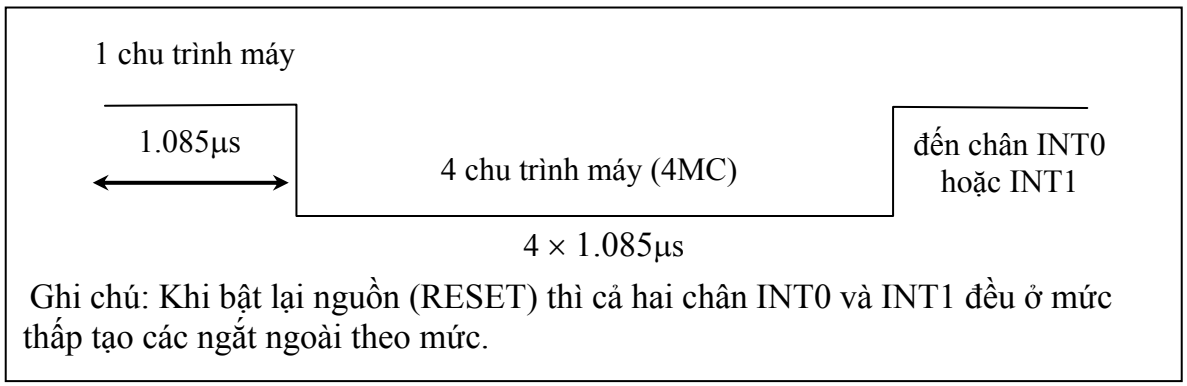
Ấn công tắc xuống sẽ làm cho đèn LED sáng. Nếu nó được giữ ở trạng thái được kích hoạt thì đèn LED sáng liên tục.



Trong 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 véc tơ ngắt tại địa chỉ 0013H. Trình ISR cho INT1 bật đèn LED lên giữ nó một lúc và tắt nó trước khi trở về. Nếu trong lúc nó thực hiện lệnh quay trở về RET1 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.

11.3.3 Trích mẫu ngắt theo mức.

Các chân P3.2 và P3.3 bình thường được dùng cho vào - ra nếu các bít INT0 và INT1 trong thanh ghi IE không được kích hoạt. Sau khi các ngắt phần cứng trong thanh gi IE được kích hoạt thì bộ vi điều khiển duy trì trích mẫu trên chân INTn đối với tín hiệu mức thấp một lần trong một chu trình máy. Theo bảng dữ liệu của nhà sản xuất của bộ vi điều khiển 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 có ngắt nào xảy ra". Tuy nhiên trong quá trình kích hoạt ngắt theo mức thấp nên nó lại phải đưa lên mức cao trước khi thực hiện lệnh RET1 và lại theo bảng dữ liệu của nhà sản xuất thì "nếu chân INTn vẫn ở mức thấp 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 hơn. Điều này là do một thực tế là ngắt theo mức không được chốt. Do vậy chân ngắt phải được giưa ở mức thấp cho đến khi bắt đầu thực hiện ISR.



Ghi chú: Khi bật lại nguồn (RESET) thì cả hai chân INT0 và INT1 đều ở mức thấp tạo các ngắt ngoài theo mức.

Hình 11.5: Thời gian tối thiểu của ngắt theo mức thấp (XTAL = 11.0592MHz)

11.3.4 Các ngắt theo sườn.

Như đã nói ở trước đây trong quá trình bật lại nguồn thì 8051 làm các chân INT0 và INT1 là các ngắt theo mức thấp. Để biến các chân này trở thành các ngắt theo sườn thì chúng ta phải viết chương trình cho các bít của thanh ghi TCON. Thanh thi TCON giữ các bít cờ IT0 và IT1 xác định chế độ ngắt theo sườn hay ngắt theo mức của các ngắt phần cứng IT0 và IT1 là các bít D0 và D2 của thanh ghi

TCON tương ứng. Chúng có thể được biểu diễn như TCON.0 và TCON.2 vì thanh ghi TCON có thể đánh địa chỉ theo bít. Khi bật lại nguồn thì 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. Bằng việc chuyển các bít TCON.0 và TCON.2 lên cao qua các 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 mà được gọi là ngắt theo sườn trong đó khi một tín hiệu chuyển từ cao xuống thấp được cấp đến chân P3.3 thì ở trường hợp này bộ vi điều khiển sẽ bị ngắt và bị cưỡng bức nhảy đến bảng véc tơ ngắt tại địa chỉ 0013H để thực hiện trình phục vụ ngắt. Tuy nhiên là với giải thiết rằng bít ngắt đã được cho phép trong thanh ghi IE.

D7 D0

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Hình 11.6: Thanh ghi TCON.

- Bít TF1 hay TCON.7 là cờ tràn của bộ Timer1. Nó được lập bởi phần cứng khi bộ đếm/ bộ định thời 1 tràn, nó được xoá bởi phần cứng khi bộ xử lý chỉ đến trình phục vụ ngắt.
- Bít TR1 hay TCON.6 là bít điều khiển hoạt động của Timer1. Nó được thiết lập và xoá bởi phần mềm để bật/ tắt Timer1.
- Bít TF0 hay TCON.5 tương tự như TF1 dành cho Timer0.
- Bít TR0 hay TCON.4 tương tự như TR1 dành cho Timer0.
- Bít IE1 hay TCON.3 cờ ngắt ngoài 1 theo sườn. Nó được thiết lập bởi CPU khi sườn ngắt ngoài (chuyển từ cao xuống thấp) được phát hiện. Nó được xóa bởi CPU khi ngắt được xử lý. Lưu ý: Cờ này không chốt những ngắt theo mức thấp.
- Bít IT1 hay TCON.2 là bít điều khiển kiểu ngắt. Nó được thiết lập và xoá bởi phần mềm để xác định kiểu ngắt ngoài theo sườn xuống hay mức thấp.
- Bít IE0 hay TCON.1 tương tự như IE1 dành cho ngắt ngoài 0.
- Bít IT0 hay TCON.0 tương tự như bít IT1 dành cho ngắt ngoài 0.

Xét ví dụ 11.6, chú ý rằng sự khác nhau duy nhất giữa vì dụ này và ví dụ 11.5 là ở trong hàng đầu tiên của MAIN khi lệnh "SETB TCON.2" chuyển ngắt INT1 về kiểu ngắt theo sườn. Khi sườn xuống của tín hiệu được cấp đến chân INT1 thì đèn LED sẽ 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 phải có một sườn xung xuống khác được cấp đến chân P3.3. Điều này ngược với ví dụ 11.5. Trong ví dụ 11.5 do bản chất ngắt theo mức của ngắt thì đèn LED còn sáng chừng nào tín hiệu ở chân INT1 vẫn còn ở mức thấp. Nhưng trong ví dụ này để bật lại đèn LED thì xung ở chân INT1 phải được đưa lên cao rồi sau đó bị hạ xuống thấp để tạo ra một sườn xuống làm kích hoạt ngắt.

Ví dụ 11.6:

Giả thiết chân P3.3 (INT1) được nối với một máy tạo xung, hãy viết một chương trình trong đó sườn xuống của xung sẽ gửi một tín hiệu cao đến chân P1.3

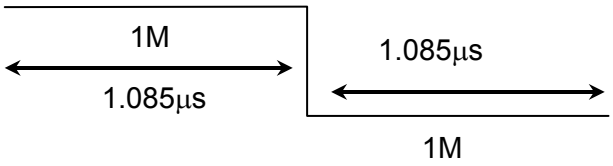
đang được nối tới đèn LED (hoặc một còi báo). Hay nói cách khác, đèn LED được bật và tắt cùng tần số với các xung được cấp tới chân INT1. Đây là phiên bản ngắt theo sườn xung của ví dụ 11.5 đã trình bày ở trên.

Lời giải:

```
                ORG   0000H
                LJMP  MAIN
; - - Trình  phục vụ ngắt ISR dành cho ngắt INT1 để bật đèn LED
                ORG   0013H          ; Nhảy đến địa chỉ của trình 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 ngắt INT1 về kiểu ngắt theo sườn
xung
                MOV   IE, #10001B    ; Cho phép ngắt ngoài INT1
HERE:           SJMP  HERE           ; Dừng ở đây cho đến khi bị ngắt
                END
```

11.3.5 Trình mẫu ngắt theo sườn.

Trước khi kết thúc phần này ta cần trả lời câu hỏi vậy thì ngắt theo sườn được trích mẫu thường xuyên như thế nào? Trong các ngắt theo sườn, nguồn ngoài phải giữ ở mức cao tối thiểu là một chu trình máy nữa đê đảm bảo bộ vi điều khiển nhìn thấy được sự chuyển dịch từ cao xuống thấp của sườn xung.



Thời hạn xung tối thiểu để phát hiện ra các ngắt theo sườn xung với tần số XTAL = 11.0592MHz

Sườn xuống của xung được chốt bởi 8051 và được giữa bởi thanh ghi TCON. Các bít 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 bít IE0 và IE1 như chỉ ra trên hình 11.6. Chúng 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 thế giới thực bên ngoài rằng ngắt hiện nay đang được xử lý và trên chân INTn này sẽ không có ngắt nào được đáp ứng chừng nào ngắt này chưa được phục vụ xong. Đây giống như tín hiệu báo bận ở máy điện thoại. Cần phải nhấn mạnh hạt điểm dưới đây khi quan tâm đến các bít IT0 và IT1 của thanh ghi TCON.

1. Khi các trình phục vụ ngắt ISR kết thúc (nghĩa là trong thanh ghi thực hiện lệnh RETI). Các bít này (TCON.1 và TCON.3) được xoá để báo rằng ngắt được hoàn tất và 8051 sẵn sàng đáp ứng ngắt khác trên chân đó. Để ngắt khác được nhận và

thì tín hiệu trên chân đó phải trở lại mức cao và sau đó nhảy xuống thấp để được phát hiện như một ngắt theo sườn.

2. Trong thời gian trình phục vụ ngắt đang được thực hiện thì chân INTn bị làm ngơ không quan tâm đến nó có bao nhiêu lần chuyển dịch từ cao xuống thấp. Trong thực tế nó là một trong các chức năng của lệnh RETI để xoá bít tương ứng trong thanh ghi TCON (bít TCON.1 và TCON.3). Nó báo cho ta rằng trình phục vụ ngắt xắp kết thúc. Vì lý do này mà các bít TCON.1 và TCON.3 được gọi là các cơ báo "ngắt đang được phục vụ" cờ này sẽ lên cao khi một sườn xuống được phát hiện trên chân INT và dừng ở mức cao trong toàn bộ quá trình thực hiện ISR. Nó chỉ bị xoá bởi lệnh RETI là lệnh cuối cùng của ISR. Do vậy, sẽ không báo giờ cần đến các lệnh xoá bít này như "CLR TCON.1" hay "CLR TCON.3" trước lệnh RETI trong trình phục vụ ngắt đối với các ngắt cứng INT0 và INT1. Điều này không đúng với trường hợp của ngắt nối tiếp.

Ví dụ 11.7:

Sự khác nhau giữa các 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.

Lời giải:

Các hai lệnh RET và RETI đều thực thi các hành vi giống nhau là lấy hai byte trên đỉnh ngăn xếp vào bộ đếm chương trình và đưa 8051 trở về nơi đó đã bỏ đi. Tuy nhiên, lệnh RETI còn thực thi 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 trên chân này. Nếu ta dùng lệnh RET thay cho RETI như là lệnh cuối cùng của trình phục vụ ngắt như vậy là ta đã vô tình khoá mọi ngắt mới trên chân này sau ngắt đầu tiên vì trạng thái của chân báo rằng ngắt vẫn đang được phục vụ. Đây là trường hợp mà các cờ TF0, TF1, TCON.1 và TCON.3 được xoá bởi lệnh RETI.

11.3.6 Vài điều bổ xung về thanh ghi TCON.

Bây giờ ta xét kỹ về các bít của thanh ghi TCON để hiểu vai trò của nó trong việc duy trì các ngắt.

11.3.6.1 Các bít IT0 và IT1.

Các bít TCON.0 và TCON.2 được coi như là các bít IT0 và IT1 tương ứng. Đây là các bít xác định kiểu ngắt theo sườn xung hay theo mức xung của các ngắt phần cứng trên chân INT.0 và INT.1 tương ứng. Khi bật lại nguồn cả hai bít này đều có mức 0 để biến chúng thành ngắt theo tín hiệu mức thấp. Lập trình viên có thể điều khiển một trong số chúng lên cao để chuyển ngắt phần cứng bên ngoài thành ngắt theo ngưỡng. Trong một hệ thống dựa trên 8051 đã cho thì một khi ta đã đặt về 0 hoặc 1 thì các bít này sẽ không thay đổi vì người thiết kế đã cố định kiểu ngắt là ngắt theo sườn hay theo mức rỗi.

11.3.6.2 Các bít IE0 và IE1.

Các bít TCON.1 và TCON.3 còn được gọi là IE0 và IE1 tương ứng. Các bít này được 8051 dùng để bám kiểu ngắt theo sườn xung. Nói các khác là nếu IT0 và IT1 bằng 0 thì có nghĩa là các ngắt phần cứng là ngắt theo mức thấp, các bít IE0 và IE1 không dùng đến làm gì. Các bít IE0 và IE1 được 8051 chỉ dùng để chốt sườn xung từ cao xuống thấp trên các chân INT0 và INT1. Khi có chuyển dịch sườn xung trên chân INT0 (hay INT1) thì 8051 đánh dấu (bật lên cao) các bít IEx trên thanh ghi TCON nhảy đến bảng véc tơ 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 ghi nhận trên chân INT0 (hay INT1) để ngăn mọi ngắt trong ngắt. Chỉ trong khi thực hiện lệnh RETI ở cuối trình phục vụ ngắt ISR thì các bít IEx mới bị 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 ta thấy rằng các bít IE0 và IE1 được 8051 sử dụng bên trong để báo có một ngắt đang được xử lý hay không. Hay nói cách khác là lập trình viên không phải quan tâm đến cá bít này.

11.3.6.3 Các bít TR0 và TR1.

Đây là những bít D4 và D6 (hay TCON.4 và TCON.6) của thanh ghi TCON. Các bít này đã được giới thiệu ở chương 9 chúng được dùng để khởi động và dừng các bộ định thời Timer0 và Timer1 tương ứng. Vì thanh ghi TCON có thể đánh địa chỉ theo bít nên có thể sử dụng các lệnh "SETB TRx" và "CLR TRx" cũng như các lệnh "SETB TCON.4" và "CLR TCON.4".

11.3.6.4 Các bít TF0 và TF1.

Các bít này là D5 (TCON.5) và D7 (TCON.7) của thanh ghi TCON mà đã được giới thiệu ở chương 9. Chúng ta được sử dụng bởi các bộ Timer0 và Timer1 tương ứng để báo rằng các bộ định thời bị tràn hay quay về không. Mặc dù ta đã dùng các lệnh "JNB TFx, đích" và "CLR TFx" nhưng chúng ta cũng không thể sử dụng các lệnh như "SETB TCON.5, đích" và "CLR TCON.5" vì TCON là thanh ghi có thể đánh địa chỉ theo bít.

11.4 Lập trình ngắt truyền thông nối tiếp.

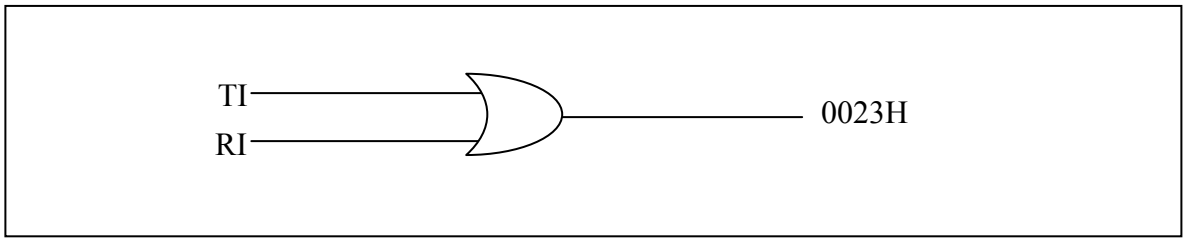
Trong chương 10 chúng ta đã nghiên cứu về truyền thông nối tiếp của 8051. Tất cả các ví dụ trong chương ấy đều sử dụng phương pháp thăm dò (polling). ậ chương này ta khám phá truyền thông dựa trên ngắt mà nó cho phép 8051 làm việc rất nhiều việc ngoài việc truyền và nhận dữ liệu từ cổng truyền thông nối tiếp.

11.4.1 Các cờ RI và TI và các ngắt.

Như đã nói ở chương 10 thì cờ ngắt truyền TI (Transfer interrupt) được bật lên khi bít cuối cùng của khung dữ liệu, bít stop được truyền đi báo rằng thanh ghi SBUF sẵn sàng truyền byte kế tiếp. Trong trường hợp cờ RI (Receive Interrupt) thì nó được bật lên khi toàn bộ khung dữ liệu kể cả bít stop đã đươc nhận. Hay nói cách khác khi thanh ghi SBUF đã có một byte thì cờ RI bật lên báo rằng byte dữ liệu nhận được cần lấy đi cất vào nơi an toàn trước khi nó bị mất (bị ghi đè) bởi dữ liệu mới nhận được. Chừng nào còn nói về truyền thông nối tiếp thì tất cả mọi khái niệm trên đây đều áp dụng giống như nhau cho dù sử dụng phương pháp thăm dò hay sử dụng phương pháp ngắt. Sự khác nhau duy nhất giữa hai phương pháp này là ở cách phục vụ quá trình truyền thông nối tiếp như thế nào. Trong phương pháp thăm dò thì chúng ta phải đợi cho cờ (TI hay RI) bật lên và trong lúc chờ đợi thì ta không thể làm gì được cả. Còn trong phương pháp ngắt thì ta được báo khi 8051 đã nhận được một byte hoặc nó sẵn sàng chuyển (truyền) byte kế tiếp và ta có thể làm các công việc khác trong khi truyền thông nối tiếp đang được phục vụ.

Trong 8051 chỉ có một ngắt dành riêng cho truyền thông nối tiếp. Ngắt này được dùng cho cả truyền và nhận dữ liệu. Nếu bít ngắt trong thanh gi IE (là bít 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 véc tơ ngắt để thực

hiện nó. Trong trình ISR này chúng ta phải kiểm tra các cờ TI và RI để xem cờ nào gây ra ngắt để đáp ứng một cách phù hợp (xem ví dụ 11.8).



Hình 11.7: Ngắt truyền thông có thể do hai cờ TI và RI gọi.

11.4.2 Sử dụng cổng COM nối tiếp trong 8051.

Trong phần lớn các ứng dụng, ngắt nối tiếp chủ yếu được sử dụng để nhận dữ liệu và không bao giờ được sử dụng để truyền dữ liệu nối tiếp. Điều này giống như việc báo chuông để nhận điện thoại, còn nếu ta muốn gọi điện thoại thì có nhiều cách khác ngắt ta chứ không cần đến đổ chuông. Tuy nhiên, trong khi nhận điện thoại ta phải trả lời ngay không biết ta đang làm gì nếu không thuộc gọi sẽ (mất) đi qua. Tương tự như vậy, ta sử dụng các ngắt nối tiếp khi nhận dữ liệu đi đến để sao chép cho nó không bị mất: Hãy xét ví dụ 11.9 dưới đây.

Ví dụ 11.8:

Hãy viết chương trình trong đó 8051 đọc dữ liệu từ cổng P1 và ghi nó tới cổng P2 liên tục trong khi 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 baud là 9600.

Lời giải:

```
                ORG   0
                LJMP  MAIN
                ORG   23H
                LJMP  SERIAL                    ; Nhảy đến trình phục vụ ngắt truyền
thông nối tiếp
MAIN:           MOVQP1, # 0FFH          ; Lấy cổng P1 làm cổng đầu vào
                MOV   TMOD, # 20h       ; Chọn Timer1, chế độ 2 tự nạp lại
                MOV   TH1, # 0FDH       ; Chọn tốc độ baud = 9600
                MOV   SCON, # 50H       ; Khung dữ liệu: 8 bít dữ liệu, 1 stop à cho
phép REN
                MOV   IE, # 10010000B           ; Cho phép ngắt nối tiếp
                SETB  TR1               ; Khởi động Timer1
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 đến cờ TI cao
                MOV   A, SBUF                   ; Nếu không tiếp tục nhận dữ liệu
                CLR   RI                ; Xoá cờ RI vì CPU không làm điều này
                RETI                    ; Trở về từ trình phục vụ ngắt
```

```
TRANS:      CLR   TI              ; Xoá cờ TI vì CPU không làm điều
này
            RETI                  ; Trở về từ ISR
            END
```

Trong vấn đề trên thấy chú ý đến vai trò của cờ TI và RI. Thời điểm một byte được ghi vào SBUF thì nó được đóng khung và truyền đi nối tiếp. Kết quả là khi bít cuối cùng (bít stop) được truyền đi thì cờ TI bật lên cao và nó gây ra ngắt nối tiếp được gọi khi bít tương ứng của nó trong thanh ghi IE được đưa lên cao. Trong trình phục vụ ngắt nối tiếp, ta phải kiểm tra cả cờ TI và cờ RI vì cả hai đều có thể gọi ngắt hay nói cách khác là chỉ có một ngắt cho cả truyền và nhận.

Ví dụ 11.9:

Hãy viết chương trình trong đó 8051 nhận dữ liệu từ cổng P1 và gửi liên tục đến cổng P2 trong khi đó dữ liệu đi vào từ cổng nối tiếp COM được gửi đến cổng P0. Giả thiết tần số XTAL là 11.0592MHz và tốc độ baud 9600.

Lời 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 và chế độ hai tự nạp lại
            MOV   TH1, # 0FDH     ;Khung dữ liệu: 8 bít dữ liệu, 1 stop, cho phép
REN
            MOV   SCON, # 50H     ; Cho phép ngắt nối tiếp
            MOV   IE, # 10010000B       ; Khởi động Timer1
            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 tiếp tục nhận dữ liệu
            MOV   P0, A           ; Gửi dữ lệu đầu vào đến cổng P0
            CLR   RI              ; Xoá  vờ RI vì CPU không xoá cờ này
            RETI                  ; Trở về từ ISR
TRANS:      CLS   TI                    ; Xoá cờ TI và CUP không xoá cờ này.
            RETI                  ; ; trở về từ ISR
            END
```

11.4.3 Xoá cờ RI và TI trước lệnh RETI.

Để ý rằng lệnh cuối cùng trước khi trở về từ ISR là RETI là lệnh xoá các cờ RI và TI. Đây là điều cần thiết bởi vì đó là ngắt duy nhất dành cho nhận và truyền 8051 không biết được nguồn gây ra ngắt là nguồn nào, do vậy trình phục vụ ngắt phải được xoá các cờ này để cho phép các ngắt sau đó được đáp ứng sau khi kết thúc ngắt. Điều này tương phản với ngắt ngoài và ngắt bộ định thời đều được 8051 xoá

các cờ. Các lệnh xoá các cờ ngắt bằng phần mềm qua các lệnh "CLR TI" và "CLR RI". Hãy xét ví dụ 11.10 dưới đây và để ý đến các lệnh xoá cờ ngắt trước lệnh RETI.

Ví dụ 11.10:

Hãy viết một chương trình sử dụng các ngắt để thực hiện các công việc sau:

a) Nhận dữ liệu nối tiếp và gửi nó đến cổng P0.
b) Lấy cổng P1 đọc và truyền nối tiếp và sao đến cổng P2.
c) Sử dụng Timer0 tạo sóng vuông tần số 5kHz trên chân P0.1 giả thiết tần số XTAL = 11.0592MHz và tốc độ baud 4800.

Lời giải:

```
                ORG   0
                LJMP  MAIN
                ORG   000BH                ; Trình phục vụ ngắt dành cho Timer0
                CPL   P0.1                 : Tạo xung ở chân P0.1
                RETI                       ; Trở về từ ISR
                ORG   23H                  ; Nhảy đến địa chỉ ngắt truyền nối tiếp
                LJMP  SERIAL                     ; Lấy cổng P1 làm cổng đầu vào
                ORG   30H
 MAIN :         MOV   P1, # 0FFH                 ; Chọn Timer0 và Timer1 chế độ 2 tự
nạp lại
                MOV   TMOD, # 22H          ; Chọn Timer0 và Timer1 chế độ 2 tự nạp lại
                MOV   TH1, # 0F6H          ; Chọn tốc độ baud 4800
                MOV   SCON, # 50H          ; Khung dữ liệu: 8 bít dữ liệu, 1 stop, cho
phép REN
                MOV   TH0, # - 92          ; Tạo tần số 5kHz
                MOV   IE, # 10010010B            ; Cho phép ngắt nối tiếp
                SETB  TR1                  ; Khởi động Timer1
                SETB  TR0                  ; Khởi động Timer0
 BACK:          MOV   A, P1                      ; Đọc dữ liệu từ cổng P1
                MOV   SBUF, A                    ; Lấy một lần bản sao dữ liệu
                MOV   P2, A                ; Ghi nó 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 vào
                MOV   A, SBUF                    ; Nếu không tiếp tục nhận dữ liệu
                MOV   P0, A                ; Gửi dữ liệu nối tiếp đến P0
                CLR   RI                   ; Xoá cờ RI vì 8051 không làm điều này
                RETI                       ; Trở về từ ISR
 TRANS:         CLR   TI                         ; Xoá cờ TI vì 8051 không xoá
                RETI                       ; Trở về từ ISR.
                END
```

Trước khi kết thúc phần này hãy để ý đến danh sách tất cả mọi cờ ngắt được cho trong bảng 11.2. Trong khi thanh thi TCON giữ 4 cờ ngắt còn hai cờ TI và RI ở trong thanh ghi SCON của 8051.

Bảng 11.2: Các bít cờ ngắt.

Ngắt	Cờ	Bít của thanh ghi SFR
Ngắt ngoài 0	IE0	TCON.1

Ngắt ngoài 1	IE1	TCON.3
Ngắt Timer0	TF0	TCON.5
Ngắt Timer1	TF1	TCON.7
Ngắt cổng nối tiếp	T1	SCON.1
Ngắt Timer2	TF2	T2CON.7 (TA89C52)
Ngắt Timer2	EXF2	T2CON.6 (TA89C52)

11.5 Các mức ưu tiên ngắt trong 8051.

11.5.1 Các mức ưu tiên trong quá trình bật lại nguồn.

Khi 8051 được cấp nguồn thì các mức ưu tiên ngắt được gán theo bảng 11.3. Từ bảng này ta thấy ví dụ nếu các 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. Trong thực tế sơ đồ mức ưu tiên ngắt trong bảng không có ý nghĩa gì cả mà một quy trình thăm dò trong đó 8051 thăm dò các ngắt theo trình tự cho trong bảng 11.3 và đáp ứng chúng một cách phù hợp.

Bảng 11.3: Mức ưu tiên các ngắt trong khi cấp lại nguồn.

Mức ưu tiên 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 11.1:

Hãy bình luận xem điều gì xảy ra nếu các ngắt INT0, TF0 và INT1 được kích hoạt cùng một lúc. Giả thiết rằng các mức ưu tiên được thiết lập như khi bật lại nguồn và các ngắt ngoài là ngắt theo sườn xung.

Lời giải:

Nếu ba ngắt này được kích hoạt cùng một thời điểm thì chúng được chốt và được giữ ở bên trong. Sau đó 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 theo trình tự. Do vậy, khi cả ba ngắt trên đây cùng được kích hoạt một lúc thì ngắt ngoài 0 (IE0) được phục vụ trước hết sau đó đến ngắt Timer0 (TF0) và cuối cùng là ngắt ngoài 1 (IE1).

D7 D0

--	--	PT2	PS	PT1	PX1	PT0	PX0

Hình 11.8: Thanh ghi mức ưu tiên ngắt IP, bít ưu tiên = 1 là mức ưu tiên cao, bít ưu tiên = 0 là mức ưu tiên thấp.

- Bít D7 và D6 hay IP.7 và IP.6 - chưa dùng.
- Bít D5 hay IP.5 là bít ưu tiên ngắt Timer2 (dùng cho 8052)
- Bít D4 hay IP.4 là bít ưu tiên ngắt cổng nối tiếp

- Bít D3 hay IP.3 là bít ưu tiên ngắt Timer1
- Bít D2 hay IP.2 là mức ưu tiên ngắt ngoài 1
- Bít D1 hay IP.1 là mức ưu tiên ngắt Timer 0
- Bít D0 hay IP.0 là mức ưu tiên ngắt ngoài 0

Người dùng không được viết phần mềm ghi các số 1 vào các bít chưa dùng vì chúng dành cho các ứng dụng tương lại.

11.5.2 Thiết lập mức ưu tiên ngắt với thanh ghi IP.

Chúng ta có thể thay đổi trình tự trong 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 bít của thanh ghi này, khi bật lại nguồn thanh thi 1P chứa hoàn toàn các số 0 để tạo ra trình tự ưu tiên ngắt theo bảng 11.3. Để một ngắt nào đó mức ưu tiên cao hơn ta thực hiện đưa bít tương ứng lên cao. Hãy xem ví dụ 11.12.

Một điểm khác nữa cần được làm sáng tỏ là mức ưu tiên ngắt khi hai hoặc nhiều bít ngắt trong thanh ghi IP được đặt lên cao. Trong trường hợp này thì trong khi các ngắt này có mức ưu tiên cao hơn các ngắt khác chúng sẽ được phục vụ theo trình tự cho trong bảng 11.3. 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) sau đó.
b) Hãy phân tích điều gì xảy ra khi INT0, INT1 và TF0 được kích hoạt cùng lúc. Giả thiết tất cả các ngắt đều là các ngắt theo sườn.

Lời giải:

a) MOV IP, #0000 0100B ; Đặt bít IP.2 = 1 để gán INT1 mức ưu tiên cao nhất. Lệnh "SETB IP.2" cũng tác động tương tự bởi vì IP là thanh ghi có thể đánh địa chỉ theo bít.
b) Lệnh trong bước a) gán mức ưu tiên cao hơn INT1 so với các ngắt khác, do vậy khi INT0, INT1 và TF0 được kích hoạt cùng lúc thì trước hết INT1 được phục vụ trước rồi sau đó đến INT0 và cuối cùng là TF0. Điều này là do INT1 có mức ưu tiên cao hơn hai ngắt kia ở bước a). Sau khi thực hiện song ngắt INT1 thì 8051 trở về phục vụ ngắt còn lại theo trình tự ưu tiên trong bảng 11.3.

Ví dụ 11.13:

Giả thiết rằng sau khi bật lại nguồn thì mức ưu tiên ngắt được thiết lập bởi lệnh "MOV IP, #0000 1100B". Hãy bình luận về quá trình các ngắt được phục vụ như thế nào?

Lời giải:

Lệnh "MOV IP, #0000 1100B" (chữ B là giá trị thập phân) thiết lập ngắt ngoài (INT1) và ngắt bộ Timer1 (TF1) có mức ưu tiên cao hơn các ngắt khác. Tuy nhiên, vì chúng được thăm dò theo bảng 11.3 nên chúng sẽ được phục vụ theo trình tự sau:

Mức ưu tiên cao nhất: Ngắt ngoài 1 (INT1)
Ngắt bộ Timer 1 (TF1)
Ngắt ngoài 0 (INT0)
Ngắt bộ Timer0 (TF0)
Mức ưu tiên thấp nhất: Ngắt cổng truyền thông nối tiếp (RI + RT).

11.5.3 Ngắt trong ngắt.

Điều gì xảy ra nếu 8051 đang thực hiện một trình phục vụ ngắt thuộc 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 trong ngắt. Trong 8051 một ngắt ưu tiên thấp có thể bị ngắt bởi một ngắt có mức ưu tiên cao hơn chứ không bị ngắt bởi một ngắt có mức ưu tiên thấp hơn. Mặc dù tất cả mọi ngắt đều được chốt và gửi bên trong nhưng không có ngắt mức thấp nào được CPU quan tâm ngay tức khắc nếu 8051 chưa kết thúc phục vụ các ngắt mức cao.

11.5.4 Thu chộp ngắt bằng phần mềm (Triggering).

Có nhiều lúc ta cần kiểm tra một trình phục vụ ngắt bằng con đường mô phỏng. Điều này có thể được thực hiện bằng các lệnh đơn giản để thiết lập các ngắt lên cao và bằng cách đó buộc 8051 nhảy đến bảng véc tơ ngắt. Ví dụ, nếu bít IE dành cho bộ Timer1 được bật lên 1 thì một lệnh như "SETB TF1" sẽ ngắt 8051 ngừng thực hiện công việc đang làm bất kỳ và buộc nó nhảy đến bảng véc tơ ngắt. Hay nói cách khác, ta không cần đợi cho Timer1 quay trở về 0 mới tạo ra ngắt. Chúng ta có thể gây ra một ngắt bằng các lệnh đưa các bít của ngắt tương ứng lên cao.

Như vậy ở chương này chúng ta đã biết ngắt là một sự kiện bên trong hoặc bên ngoài gây ra ngắt bộ vi điều khiển để báo cho nó biết rằng thiết bị cần được phục vụ. Mỗi một ngắt có một chương trình đi kèm với nó được gọi là trình phục vụ ngắt ISR. Bộ vi điều khiển 8051 có sáu ngắt, trong đó năm ngắt người dùng có thể truy cập được. Đó là hai ngắt cho các thiết bị phần cứng bên ngoài INT0 và INT1, hai ngắt cho các bộ định thời là TF0 và TF1 và ngắt lành cho truyền thông nối tiếp.

8051 có thể được lập trình cho phép hoặc cấm một ngắt bất kỳ cũng như thiết lập mức ưu tiên cho nó theo yêu cầu của thuật toán ứng dụng.

CHƯƠNG 12
Phối ghép với thế giới thực: LCD, ADC và các cảm biến

Chương này khám phá một số ứng dụng của 8051 với thế giới thực. Chúng ta giải thích làm cách nào phối ghép 8051 với các thiết bị như là LCD, ADC và các cảm biến.

12.1 Phối ghép một LCD với 8051.

Ở phần này ta sẽ mô tả các chế độ hoạt động của các LCD và sau đó mô tả cách lập trình và phối ghép một LCD tới 8051.

12.1.1 Hoạt động của LCD.

Trong những năm gần đây LCD đang ngày càng được sử dụng rộng rãi thay thế dần cho các đèn LED (các đèn LED 7 đoạn hay nhiều đoạn). Đó là vì các nguyên nhân sau:

1. Các LCD có giá thành hạ.
2. Khả năng hiển thị các số, các ký tự và đồ hoạ tốt hơn nhiều so với các đèn LED (vì các đèn LED chỉ hiển thị được các số và một số ký tự).
3. Nhờ kết hợp một bộ điều khiển làm tươi vào LCD làm giải phóng cho CPU công việc làm tươi LCD. Trong khi đèn LED phải được làm tươi bằng CPU (hoặc bằng cách nào đó) để duy trì việc hiển thị dữ liệu.
4. Dễ dàng lập trình cho các ký tự và đồ hoạ.

12.1.2 Mô tả các chân của LCD.

LCD được nói trong mục này có 14 chân, chức năng của các chân được cho trong bảng 12.1. Vị trí của các chân được mô tả trên hình 12.1 cho nhiều LCD khác nhau.

1. Chân V_{CC}, V_{SS} và V_{EE}: Các chân V_{CC}, V_{SS} và V_{EE}: Cấp dương nguồn - 5v và đất tương ứng thì V_{EE} được dùng để điều khiển độ tương phản của LCD.
2. Chân chọn thanh ghi RS (Register Select).

Có hai thanh ghi rất quan trọng bên trong LCD, chân RS được dùng để chọn các thanh ghi này như sau: Nếu RS = 0 thì thanh ghi mà lệnh được chọn để 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 cho phép người dùng gửi dữ liệu cần hiển thị trên LCD.

3. Chân đọc/ ghi (R/W).

Đầu vào đọc/ ghi cho phép người dùng ghi thông tin lên LCD khi R/W = 0 hoặc đọc thông tin từ nó khi R/W = 1.

4. Chân cho phép E (Enable).

Chân cho phép E được sử dụng bởi LCD để chốt thông tin hiện hữu trên chân dữ liệu của nó. Khi dữ liệu được cấp đến chân dữ liệu thì một xung mức cao xuống thấp phải được áp đến chân này để LCD chốt dữ liệu trên các chân dữ liêu. Xung này phải rộng tối thiểu là 450ns.

5. Chân D0 - D7.

Đây là 8 chân dữ liệu 8 bít, được dùng để gửi thông tin lên LCD hoặc đọc nội dung của các thanh ghi trong LCD.

Để hiển thị các chữ cái và các con số, chúng ta gửi các mã ASCII của các chữ cái từ A đến Z, a đến f và các con số từ 0 - 9 đến các chân này khi bật RS = 1.

Cũng có các mã lệnh mà có thể 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 lênh.

Chúng ta cũng sử dụng RS = 0 để kiểm tra bít cờ bận để xem LCD có sẵn sàng nhân thông tin. Cờ bận là D7 và có thể đượcđọc khi R/W = 1 và RS = 0 như sau:

Nếu R/W = 1, RS = 0 khi D7 = 1 (cờ bận 1) thì LCD bận bởi các công việc bên trong và sẽ không nhận bất kỳ thông tin mới nào. Khi D7 = 0 thì LCD sẵn sàng nhận thông tin mới. Lưu ý chúng ta nên kiểm tra cờ bận trước khi ghi bất kỳ dữ liệu nào lên LCD.

Bảng 12.1: Mô tả các chân của LCD.

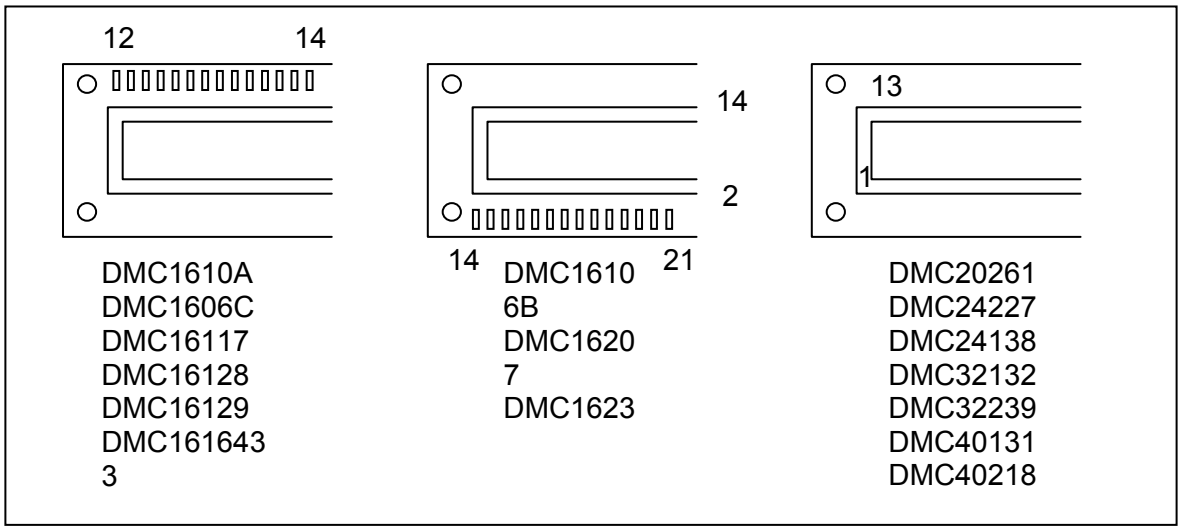
Chân	Ký hiệu	I/O	Mô tả
1	V_{SS}	-	Đất
2	V_{CC}	-	Dương nguồn 5v
3	V_{EE}	-	Cấp nguồn điều khiển phản
4	RS	I	RS = 0 chọn thanh ghi lệnh. RS = 1 chọn thanh dữ liệu
5	R/W	I	R/W = 1 đọc dữ liệu. R/W = 0 ghi
6	E	I/O	Cho phép
7	DB0	I/O	Các bít dữ liệu
8	DB1	I/O	Các bít dữ liệu
9	DB2	I/O	Các bít dữ liệu
10	DB3	I/O	Các bít dữ liệu
11	DB4	I/O	Các bít dữ liệu
12	DB5	I/O	Các bít dữ liệu
13	DB6	I/O	Các bít dữ liệu
14	DB7	I/O	Các bít dữ liệu

Bảng 12.2: Các mã lệnh LCD.

Mã (Hex)	Lệnh đến thanh ghi của LCD
1	Xoá màn hình hiển thị
2	Trở về đầu dòng
4	Giả con trỏ (dịch con trỏ sang trái)
6	Tăng con trỏ (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	Ép con trỏ Vũ đầu dòng thứ nhất
C0	Ép con trỏ Vũ đầu dòng thứ hai
38	Hai dòng và ma trận 5 × 7

Ghi chú: Bảng này được mở rộng từ bảng 12.4.



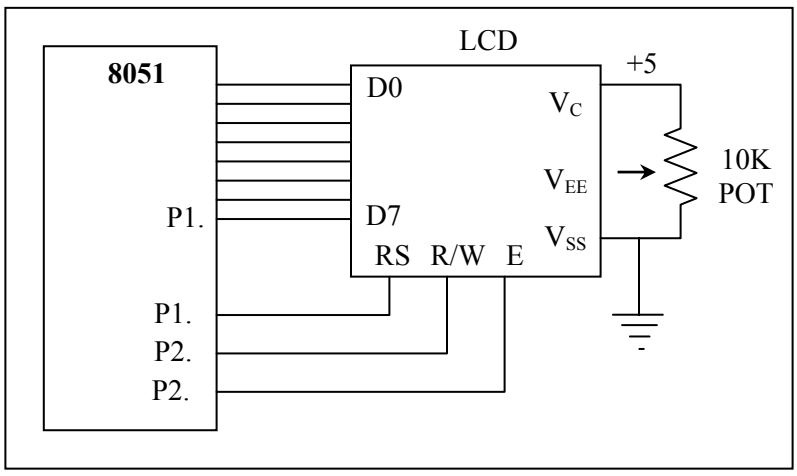
Hình 12.1: Các vị trí chân của các LCD khác nhau của Optrex.

12.1.3 Gửi các lệnh và dữ liệu đến LCD với một độ trễ.

Để gửi một lệnh bất kỳ từ bảng 12.2 đến LCD ta phải đưa chân RS về 0. Đối với 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. Điều này được chỉ ra trong đoạn mã chương trình dưới đây (xem hình 12.2).

```
; gọi độ thời gian trễ trước khi gửi dữ liệu/ lệnh kế tiếp.
; chân P1.0 đến P1.7 được nối tới chân dữ 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                ; Khởi tạo LCD hai dòng với ma trận 5
× 7
            ACALL       COMNWRT   ; Gọi chương trình con lệnh
            ACALL       DELAY                   ; Cho LCD một độ trễ
            MOV         A, # 0EH    ; Hiển thị màn hình và con trỏ
            ACALL       COMNWRT   ; Gọi chương trình con lênh
            ACALL             DELAY                     ; Cấp một độ trễ cho LCD
            MOV         AM # 01                 ; Xoá LCD
            ACALL       COMNWRT   ; Gọi chương trình con lệnh
            ACALL       DELAY                   ; Tạo độ trễ cho LCD
```

```
                MOV         A, # 06H            ; Dịch con trỏ sang phải
                ACALL       COMNWRT   ; Gọi chương trình con lệnh
                ACALL             DELAY             ; Tạo độ trễ cho LCD
                MOV         AM # 48H  ; Đưa con trỏ về dòng 1 cột 4
                ACALL       COMNWRT   ; Gọi chương trình con lệnh
                ACALL       DELAY             ; Tạo độ trễ cho LCD
                MOV         A, # "N"            ; Hiển thị chữ N
                ACALL       DATAWRT   ; Gọi chương trình con hiển thij DISPLAY
                ACALL             DELAY             ; Tạo độ trễ cho LCD
                MOV         AM # "0"            ; Hiển thị chữ 0
                ACALL       DATAWRT   ; Gọi DISPLAY
AGAIN:                SJMP          AGAIN         ; Chờ ở đây
COMNWRT:                                ; Gửi lệnh đến LCD
                MOV         P1, A       ; Sao chép 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 cho xung cao xuống thấp
                RET
DATAWRT:                                ; Ghi dữ liệu ra LCD
                MOV         P1, A       ; Sao chép 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 cho xung cao xuống thấp
                RET
DELAY:          MOV         R3, # 50            ; Đặt độ trễ 50μs hoặc cao hơn cho
CPU nhanh
HERE2:                MOV           R4, # 255     ; Đặt R4 = 255
HERE:                 DJNZ          R4, HERE      ; Đợi ở đây cho đến khi R4 = 0
                DJNZ        R3, HERE2
                RET
                END
```



Hình 12.2: Nối ghép LCD.

12.1.4 Gửi mã lệnh hoặc dữ liệu đến LCD có kiểm tra cờ bận.

Đoạn chương trình trên đây đã chỉ ra cách gửi các 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 ảtình 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 một chương trình như vậy.

```
; Kiểm tra cờ bận trước khi gửi dữ liệu, lệnh ra LCD
; Đặt P1 là cổng dữ liệu
; Đặt P2.0 nối tới cổng RS
; Đặt P2.1 nối tới chân R/W
; Đặt P2.2 nối tới chân E
            ORG
            MOV         A, # 38H            ; Khởi tạo LCD hai dòng với ma trận 5
× 7
            ACALL       COMMAND   ; Xuất lệnh
            MOV         A, # 0EH    ; Dịch con trỏ sang phải
            ACALL       COMMAND   ; Xuất lệnh
            MOV         A, # 01H            ; Xoá lệnh 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, # “0”            ; Hiển thị chữ 0
            ACALL       DATA DISPLAY
HERE:           SJMP        HERE            ; Chờ ở đây
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        ; Đặt R/W = 0 để ghi dữ liệu tới LCD
            SETB        P2.2        ; Đặt E = 1 đối với 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        ; Đặt R/W = 0 để ghi dữ liệu ra LCD
            SETB        P2.2        ; Đặt E = 1 đối với 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        ; Đặt RS = 0 để truy cập thanh ghi lệnh
            SETB        P2.1        ; Đặt R/W = 1 đọc thanh ghi lệnh
; Đọc thanh ghi lệnh và kiểm tra cờ lệnh
BACK:           CLR         P2.2            ; E = 1 đối với xung cao xuống thấp
            SETB        P2.2        ; E = 0 cho xung cao xuống thấp?
            JB          P1.7, BACK  ; Đợi ở đây cho đến khi cờ bận = 0
            RET
            END
```

Lưu ý rằng trong chương trình cờ bận D7 của thanh ghi lệnh. Để đọc thanh ghi lệnh ta phải đặt RS = 0, R/W = 1 và xung cao - xuống - thấp cho bít E để cấp

thanh ghi lệnh cho chúng ta. Sau khi đọc thanh ghi lệnh, nếu bít D7 (cờ bận) ở mức cao thì LCD bận và không có thông tin (lệnh) nào được xuất đến nó chỉ khi nào D7 = 0 mới có thể gửi dữ liệu hoặc lệnh đến LCD. Lưu ý trong phương phát này không sử dụng độ trễ thời gian nào vì ta đang kiểm tra cờ bận trước khi xuất lệnh hoặc dữ liệu lên LCD.

12.1.5 Bảng dữ liệu của LCD.

Trong LCD ta có thể đặt dữ liệu vào bất cứ chỗ nào. dưới đây là các vị trí 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

Khi AAAAAAA = 0000000 đến 0100111 cho dòng lệnh 1 và AAAAAAA = 1100111 cho dòng lệnh2. Xem bảng 12.3.

Bảng 12.3: Đánh địa chỉ cho LCD.

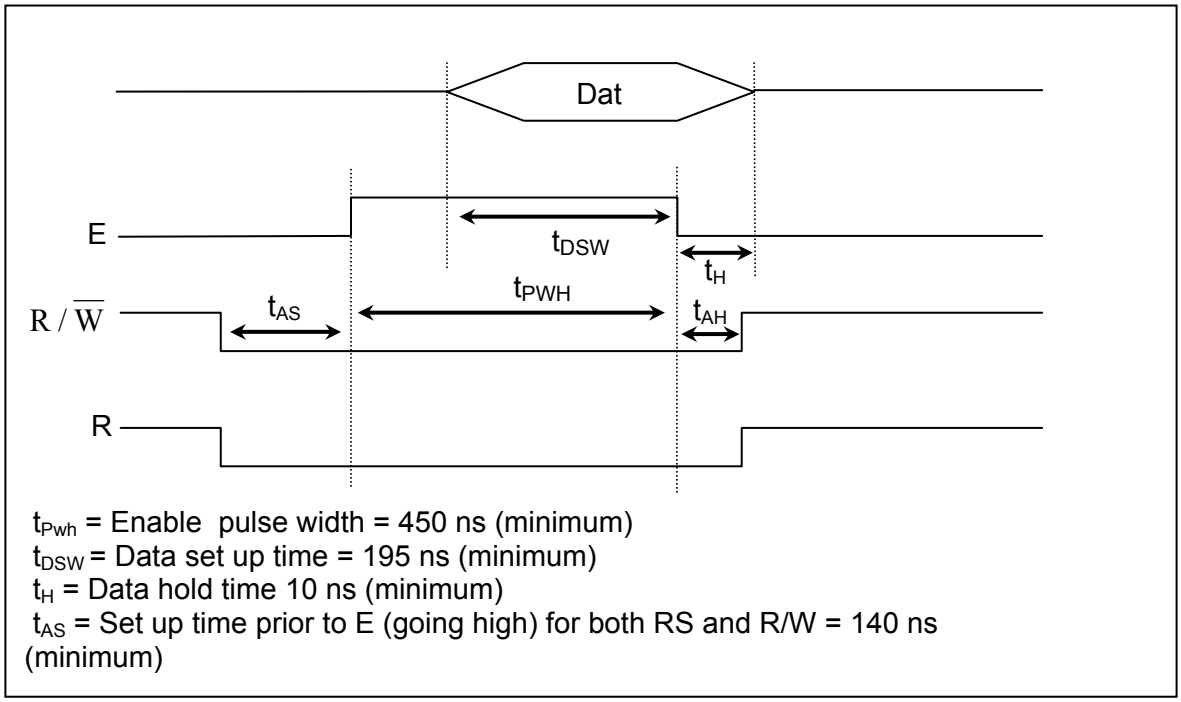
	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Dòng 1 (min)	1	0	0	0	0	0	0	0
Dòng 1 (max)	1	0	1	0	0	1	1	1
Dòng 2 (min)	1	1	0	0	0	0	0	0
Dòng 2 (max)	1	1	1	0	0	1	1	1

Dải địa chỉ cao có thể là 0100111 cho LCD. 40 ký tự trong khi đối với CLD 20 ký tự chỉ đến 010011 (19 thập phân = 10011 nhị phân). Để ý rằng dải trên 0100111 (nhị phân) = 39 thập phân ứng với vị trí 0 đến 39 cho LCD kích thước 40 × 2.

Từ những điều nói ở trên đây ta có thể nhận được các địa chỉ của vị trí con trỏ có các kích thước LCD khác nhau. Xem hình 12.3 chú ý rằng tất cả mọi địa chỉ đều ở dạng số Hex. Hình 12.4 cho một biểu đồ của việc phân thời gian của LCD. Bảng 12.4 là danh sách liệt kê chi tiết các lệnh và chỉ lệnh của LCD. Bảng 12.2 được mở rộng từ bảng này.

16 × 2 LCD	80	81	82	83	84	85	86	Through	8F
	C0	C0	C2	C3	C4	C5	C6	Through	CF
20 × 1 LCD	80	81	82	83	Through	93			
20 × 2 LCD	80	81	82	83	Through	93			
	C0	C0	C2	C3	Through	D3			
20 × 4 LCD	80	81	82	83	Through	93			
	C0	C0	C2	C3	Through	D3			
	94	95	96	97	Through	A7			
	D4	D5	D6	D7	Through	E7			
20 × 2 LCD	80	81	82	83	Through	A7			
	C0	C0	C2	C3	Through	E7			
Note: All data is in hex.									

Hình 12.3: Các địa chỉ con trỏ đối với một số LCD.



t_{Pwh} = Enable pulse width = 450 ns (minimum)
t_{DSW} = Data set up time = 195 ns (minimum)
t_H = Data hold time 10 ns (minimum)
t_{AS} = Set up time prior to E (going high) for both RS and R/W = 140 ns (minimum)

Hình 12.4: Phân khe thời gian của LCD.

Bảng 12.4: Danh sách liệt kê các lệnh và địa chỉ 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 bộ 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 chuyển dịch con trỏ và xác định dịch hiển thị các 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 và con trỏ	0	0	0	0	0	1	S/C	R/L	-	-	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 phòng 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 là thời gian cực đại khi tần số f_{CP} hoặc f_{osc} là 250KHz

2. Thời gian thực thay đổi khi tần số thay đổi. Khi tần số f_{EP} hay f_{osc} Là 270kHz thì thời gian thực hiện được tính $250/270 \times 40 = 35\mu s$ v.v…
3. Các ký hiệu viết tắt trong bảng là:
4.

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) được dùng cho các địa chỉ DD RAM và CG RAM.		
1/D = 1	Tăng	1/D = 0	Giảm
S = 1	Kèm 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 bít	DL = 0	4 bít
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 Phối ghép 8051 với ADC và các cảm biến.

Phần này sẽ khám phá ghép các chíp ADC (bộ chuyển đổi tương tự số) và các cảm biến nhiệt với 8051.

12.1.1 Các thiết bị ADC.

Các bộ chuyển đổi ADC thuộc trong những thiết bị được sử dụng rộng rãi nhất để thu dữ liệu. Các máy tính số sử dụng các giá trị nhị phân, nhưng trong thế giới vật lý thì mọi đại lượng ở dạng tương tự (liên tục). Nhiệt độ, áp suất (khí hoặc chất lỏng), độ ẩm và vận tốc và một số ít trọng những đại lượng vật lý của thế giới thực mà ta gặp hàng ngày. Một đại lượng vật lý được chuyển về dòng điện hoặc điện áp qua một thiết bị được gọi là các bộ biến đổi. Các bộ biến đổi cũng có thể được coi như các 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 cho ra các tín hiệu dạng dòng điện hoặc điện áp ở dạng liên tục. Do vậy, ta cần một bộ chuyển đổi tương tự số sao cho bộ vi điều khiển có thể đọc được chúng. Một chíp ADC được sử dụng rộng rãi là ADC 804.

12.2.2 Chíp ADC 804.

Chíp ADC 804 là bộ chuyển đổi tương tự số trong họ các loạt ADC 800 từ hãng National Semiconductor. Nó cũng được nhiều hãng khác sản xuất, nó làm việc với +5v và có độ phân giải là 8 bít. Ngoài độ phân giải thì thời gian chuyển đổi cũng là một yếu tố quan trọng khác khi đánh giá một bộ ADC. Thời gian chuyển đổi được định nghĩa như là thời gian mà bộ ADC cần để chuyển một đầu vào tương tự thành một số nhị phân. Trong ADC 804 thời gian chuyển đổi thay đổi phụ thuộc vào tần số đồng hồ được cấp tới chân CLK và CLK IN nhưng không thể nhanh hơn 110μs. Các chân của ADC 804 được mô tả như sau:

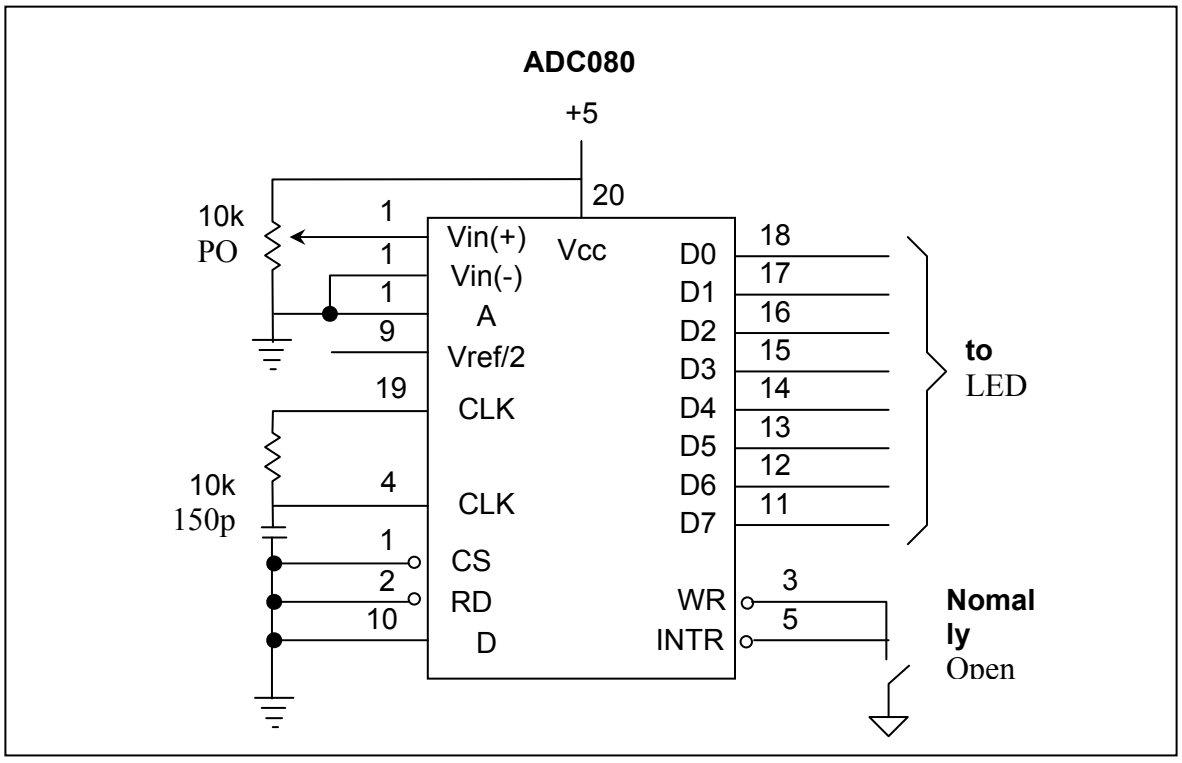
1. Chân $\overline{CS}$ - chọn chíp: Là một đầu vào tích cực mức thấp được sử dụng để kích hoạt chíp ADC 804. Để truy cập ADC 804 thì chân này phải ở mức thấp.

2. Chân $\overline{RD}$ (đọc): Đây là một tín hiệu đầu vào được tích cực mức thấp. Các bộ ADC chuyển đổi đầu vào tương tự thành số nhị phân tương đương với nó và giữ nó trong một thanh ghi trong. $\overline{RD}$ được sử dụng để nhận dữ liệu được chuyển đổi ở đầu ra của ADC 804. Khi CS = 0 nếu một xung cao - xuống - thấp được áp đến chân $\overline{RD}$ thì đầu ra số 8 bít được hiển diện ở các chân dữ liệu D0 - D7. Chân $\overline{RD}$ cũng được coi như cho phép đầu ra.
3. Chân ghi $\overline{WR}$ (thực ra tên chính xác là "Bắt đầu chuyển đổi"). Đây là chân đầu vào tích cực mức thấp được dùng để báo cho ADC 804 bắt đầu quá trình chuyển đổi. Nếu CS = 0 khi $\overline{WR}$ tạo ra xung cao - xuống - thấp thì bộ ADC 804 bắt đầu chuyển đổi giá trị đầu vào tương tự V_{in} về số nhị phân 8 bít. 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 ép xuống thấp bởi ADC 804.
4. Chân CLK IN và CLK R.

Chân CLK IN là một chân đầu vào được nối tới một nguồn đồng hồ ngoài khi đồng hồ ngoài được sử dụng để tạo ra thời gian. Tuy nhiên 804 cũng có một máy tạo xung đồng hồ. Để sử dụng máy tạo xung đồng hồ trong (cũng còn được gọi là máy tạo 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 trên hình 12.5. Trong trường hợp này tần số đồng hồ được xác định bằng biểu thức:

$$f = \frac{1}{1{,}1RC}$$

Giá trị tiêu biểu của các đại lượng trên là R = 10kΩ và C= 150pF và tần số nhận được là f = 606kHz và thời gian chuyển đổi sẽ mất là 110μs.



Hình 12.5: Kiểm tra ADC 804 ở chế độ chạy tự do.

5. Chân ngắt $\overline{\text{INTR}}$ (ngắt hay gọi chính xác hơn là "kết thúc chuyển đổi').

Đây là chân đầu ra tích cực mức thấp. Bình thường nó 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 sẵn sàng để lấy đi. Sau khi $\overline{\text{INTR}}$ xuống thấp, ta đặt CS = 0 và gửi một xung cao 0 xuống - thấp tới chân $\overline{\text{RD}}$ lấy dữ liệu ra của 804.

6. Chân V_{in} (+) và V_{in} (-).

Đây là các đầu vào tương tự vi sai mà $V_{in} = V_{in}$ (+) - V_{in} (-). Thông thường V_{in} (-) được nối xuống đất và V_{in} (+) được dùng như đầu vào tương tự được chuyển đổi về dạng số.

7. Chân V_{CC}.

Đây là chân nguồn nuối +5v, nó cũng được dùng như điện áp tham chiếu khi đầu vào $V_{ref/2}$ (chân 9) để hở.

8. Chân $V_{ref/2}$.

Chân 9 là một điện áp đầu vào được dùng cho điện áp tham chiếu. Nếu chân này hở (không được nối) thì điện áp đầu vào tương tự cho ADC 804 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} cần phải khác ngoài dải 0 đến 5v. Chân $V_{ref/2}$ đượcdùng để thực thi các điện áp đầu vào khác ngoài dải 0 - 5v. Ví dụ, nếu dải đầu vào tương tự cần phải là 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: Điện áp $V_{ref/2}$ liên hệ với dải V_{in}.

V_{ref}/ 2(V)	**V_{in}(V)**	**Step Size (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.

9. Các chân dữ liệu D0 - D7.

Các chân dữ liệu D0 - D7 (D7 là bít cao nhất MSB và D0 là bít thấp nhất LSB) là các chân đầu ra dữ liệu số. Đây là những chân được đệm ba trạng thái và dữ

liệu được chuyển đổi chỉ được truy cập khi chân CS = 0 và chân $\overline{RD}$ bị đưa xuống thấp. Để tính điện áp đầu ra ta có thể sử dụng công thức sau:

$$D_{out} = \frac{V_{in}}{\text{kich thuoc buoc}}$$

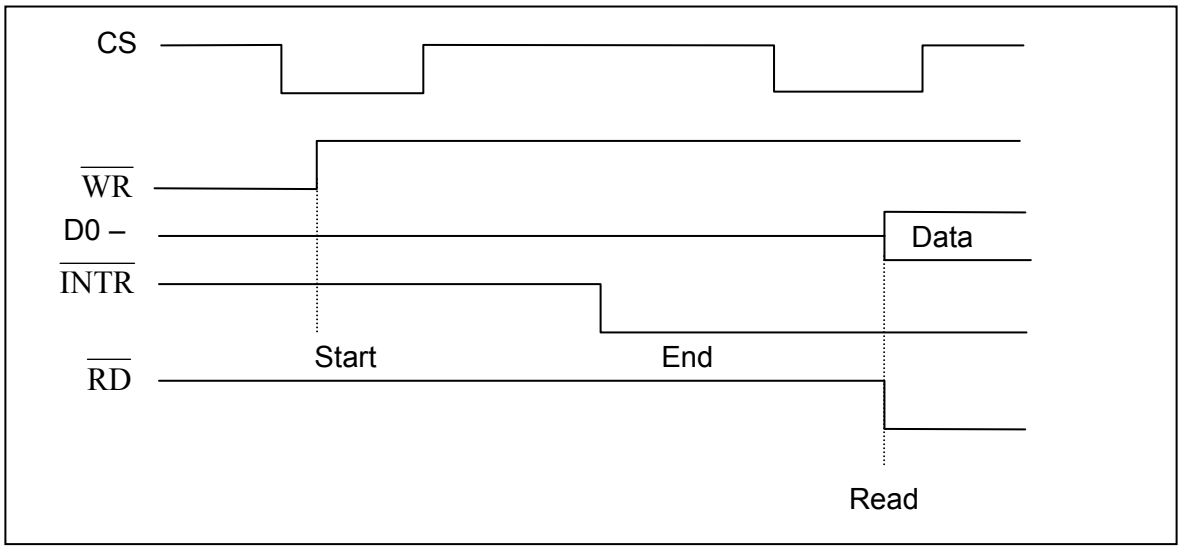
Với D_{out} là đầu ra dữ liệu số (dạng thập phân). V_{in} là điện áp đầu vào tương tự và độ phân dải là sự thay đổi nhỏ nhất được tính như là $(2 \times V_{ref}/2)$ chia cho 256 đối với ADC 8 bít.

10. 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 mà ta phải có hai đất là để cách ly tín hiệu tương tự V_{in} từ các điện áp ký sinh tạo ra việc chuyển mạch số được chính xác. Trong phần trình bày của chúng ta thì các chân này được nối chung với một đất. Tuy nhiên, trong thực tế thu đo dữ liệu các chân đất này được nối tách biệt.

Từ những điều trên ta kết luận rằng các bước cần phải thực hiện khi chuyển đổi dữ liệu bởi ADC 804 là:

a) Bật CS = 0 và gửi một xung thấp lên cao tới chân $\overline{WR}$ để bắt đầu chuyển đổi.
b) Duy trì hiển thị chân $\overline{INTR}$. Nếu $\overline{INTR}$ xuống thấp thì việc chuyển đổi được hoàn tất và ta có thể sang bước kế tiếp. Nếu $\overline{INTR}$ cao tiếp tục thăm dò cho đến khi nó xuống thấp.
c) Sau khi chân $\overline{INTR}$ xuống thấp, ta bật CS = 0 và gửi một xung cao - xuống - thấp đến chân $\overline{RD}$ để lấy dữ liệu ra khỏi chíp ADC 804. Phân chia thời gian cho quá trình này được trình bày trên hình 12.6.

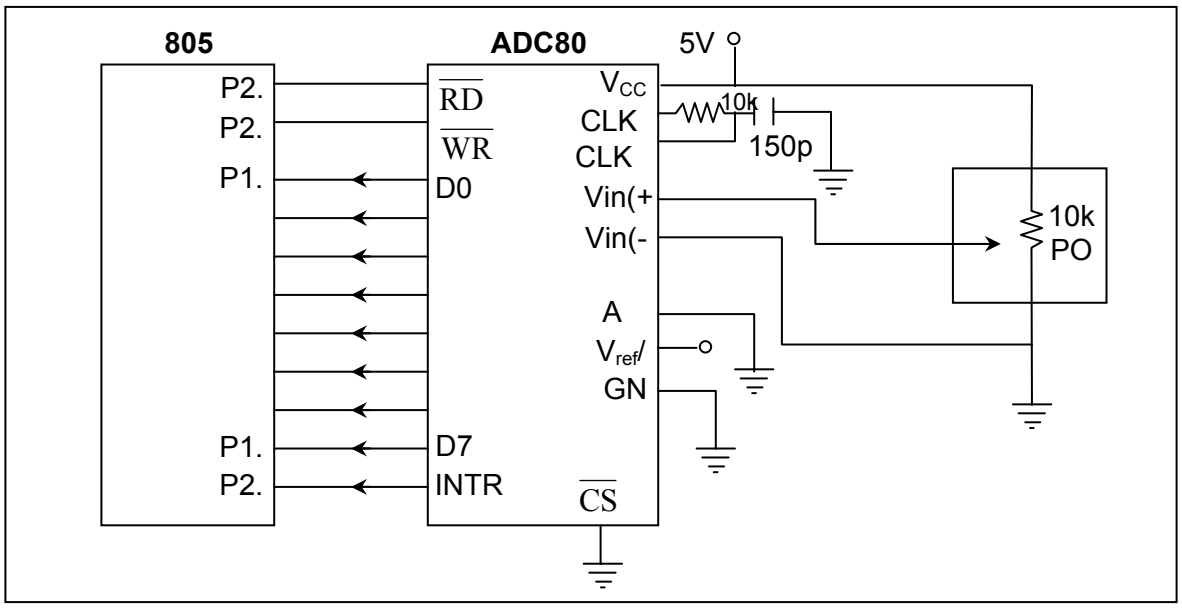


Hình 12.6: Phân chia thời gian đọc và ghi của ADC 804.

12.2.3 Kiểm tra ADC 804.

Chúng ta có thể kiểm tra ADC 804 bằng cách sử dụng sơ đồ mạch trên hình 12.7. thiết lập này được gọi là chế độ kiểm tra chạy tự do và được nhà sản xuất khuyến cao nên sử dụng. Hình 12.5 trình bày một biến trở được dùng để cấp một điện áp tương tự từ 0 đến 5V tới chân đầu vào.

$V_{in}(+)$ của ADC 804 các đầu ra nhị phân được hiển thị trên các đèn LED của bảng huấn luyện số. Cần phải lưu ý rằng trong chế độ kiểm tra chạy tự do thì đầu vào CS được nối tới đất và đầu vào $\overline{WR}$ được nối tới đầu ra $\overline{INTR}$. Tuy nhiên, theo tài liệu của hãng National Semiconductor "nút WR và INTR phải được tạm thời đưa xuống thấp kế sau chu trình cấp nguồn để bảo đảm hoạt động".



Hình 12.7: Nối ghép ADC 804 với nguồn đồng hồ riêng.

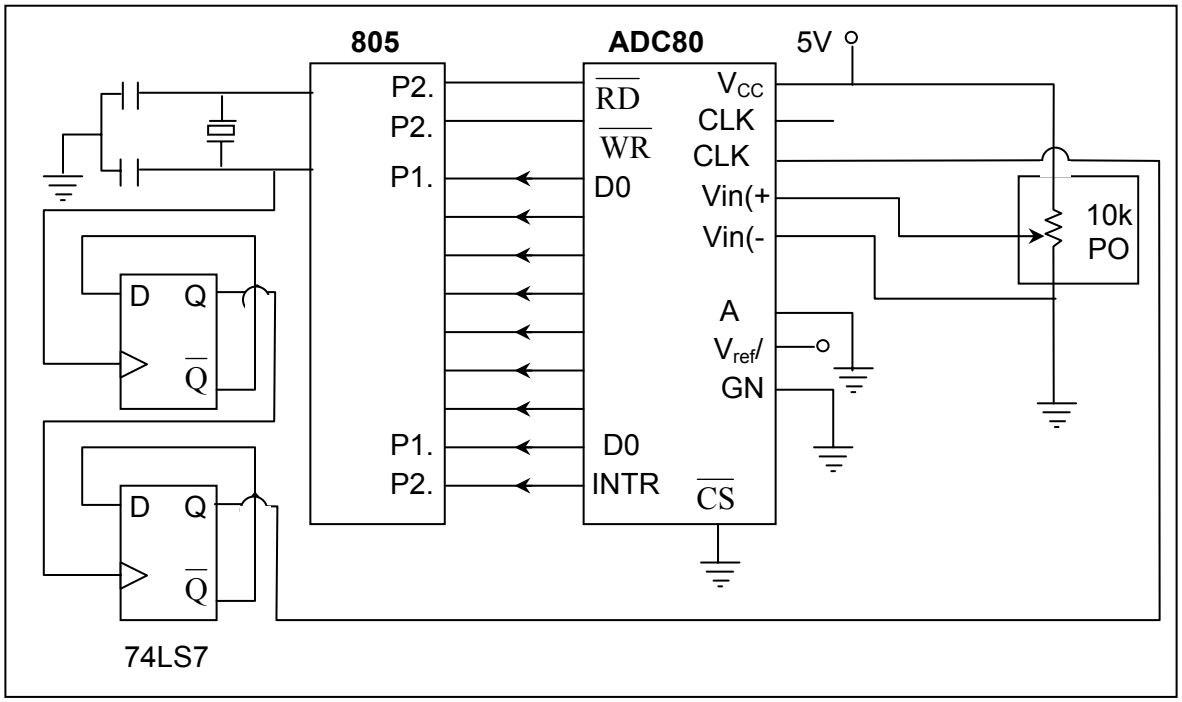
Ví dụ 12.7:

Hãy thử nối ghép ADC 804 với 8051 theo sơ đồ 12.7. Viết một chương trình để hiển thị chân INTR và lấy đầu vào tương tự vào thanh ghi A. Sau đó gọi một chương trình chuyển đổi mã Hex ra ASCII và một chương trình hiển thị dữ liệu. Thực hiện điều này liên tục.

Lời 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 của ADC 804
        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 cho P2.7 to để kết thúc chuyển đổi
        CLR     P2.5            ; Kết thúc chuyển đổi, cho phép đọc RD
        MOV     A, P1           ; Đọc dữ liệu vào thanh ghi A
        ACALL   CONVERSION      ; Chuyển đổi số Hex ra mã ASCII
        ACALL   DATA-DISPLAY    ; Hiển thị dữ liệu
        SETB    P2.5            ; Đưa RD = 1 để cho lần đọc sau.
        SJMP    BACK
```



Hình 12.8: Nối ghép ADC 804 với đồng hồ từ XTAL2 của 8051.

Trên hình 12.8 ta có thể thấy rằng tín hiệu đồng hồ đi vào ADC 804 là từ tần số thạch anh của 8051. Vì tần số này quá cao nên ta sử dụng hai mạch lật Rlip - Flop kiểu D (74LS74) để chia tần số này cho 4. Một mạch lật chia tần số cho 2 nếu ta nối đầu $\overline{Q}$ tới đầu vào D. Đối với tần số cao hơn thì ta cần sử dụng nhiều mạch Flip - Plop hơn.

12.2.4 Phối ghép với một cảm biến nhiệt của 8051.

Các bộ biến đổi (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 biến đổi 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 về các tín hiệu điện sử dụng một bộ biến đổi gọi là Rhermistor (bộ cảm biến nhiệt), một bộ cảm biến nhiệt đáp ứng sự thay đổi nhiệt độ bằng cách thay đổi trở kháng nhưng đáp ứng của nó không tuyến tính (xem bảng 12.6).

Bảng 12.6: Trở kháng của bộ cảm biến nhiệt theo nhiệt độ.

Nhiệt độ (^{0}C)	**Trở kháng của cảm biến (kΩ)**

0	29.490
25	10.000
50	3.893
75	1.700
100	0.817

Bảng 12.7: Hướng dẫn chọn loạt các cảm biến 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: Hướng dẫn chọn loạt các 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

Tính chất gắn liện với việc viết phần mềm cho các thiết bị phi tuyến như vậy đã đưa nhiều nhà sản xuất tung 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 các loạt họ LM34 và LM35 của hãng National Semiconductor Corp.

12.2.5 Các bộ cảm biến nhiệt họ LM34 và LM35.

Loạt các bộ cảm biến LM34 là các bộ cảm biến nhiệt mạch tích hợp chính xác cao mà điện áp đầu ra của nó tỷ lệ tuyến tính vơi nhiệt độ Fahrenheit (xem hình 12.7). loạt LM34 không yêu cầu cân chỉnh bên ngoài vì vốn nó đã được cân chỉnh rồi. Nó đưa ra điện áp 10mV cho sự thay đổi nhiệt độ 1^0F. bảng 12.7 hướng dẫn ta chọn các cảm biến loạt LM34.

Loạt các bộ cảm biến LM35 cũng là các bộ cảm biến nhiệt mách tích hợp chính xác cao mà điện áp đầu ra của nó tỷ lệ tuyến tính với nhiệt độ theo thang độ Celsius. Chúng cũng không yêu cầu cân chỉnh ngoài vì vốn chúng đã được cân chỉnh. Chúng đưa ra điện áp 10Mv cho mỗi sự thay đổi 1^0C. Bảng 12.8 hướng dẫn ta chọn các cảm biến họ LM35.

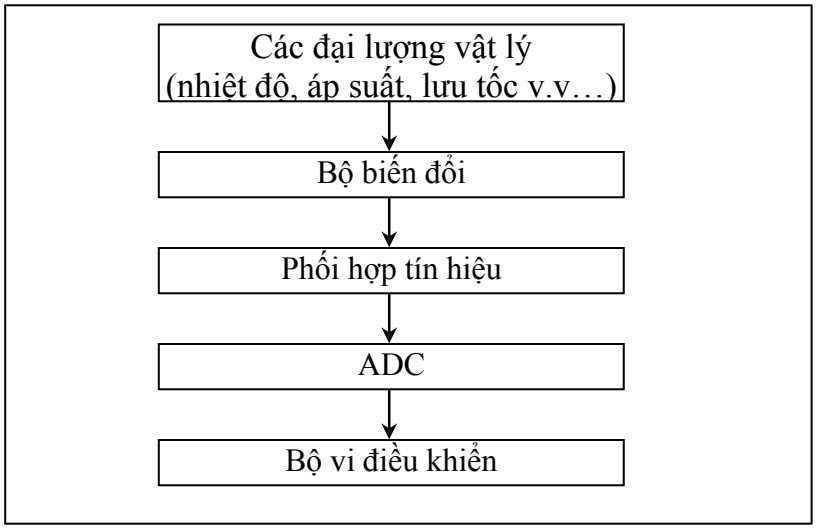
12.2.6 Phối hợp tín hiệu và phối ghép LM35 với 8051.

Phối hợp tín hiệu là một thuật ngữ được sử dụng rộng rãi trong lĩnh vực thu đo dữ liệu. Hầu hết các bộ biến đổi đều đưa ra các tín hiệu điện dạng điện áp, dòng điện, dung kháng hoặc trở kháng. Tuy nhiên, chúng ta cần chuyển đổi các tín hiệu này về điện áp nhằm gửi đầu vào đến bộ chuyển đổi ADC. Sự chuyển đổi (biến đổi)

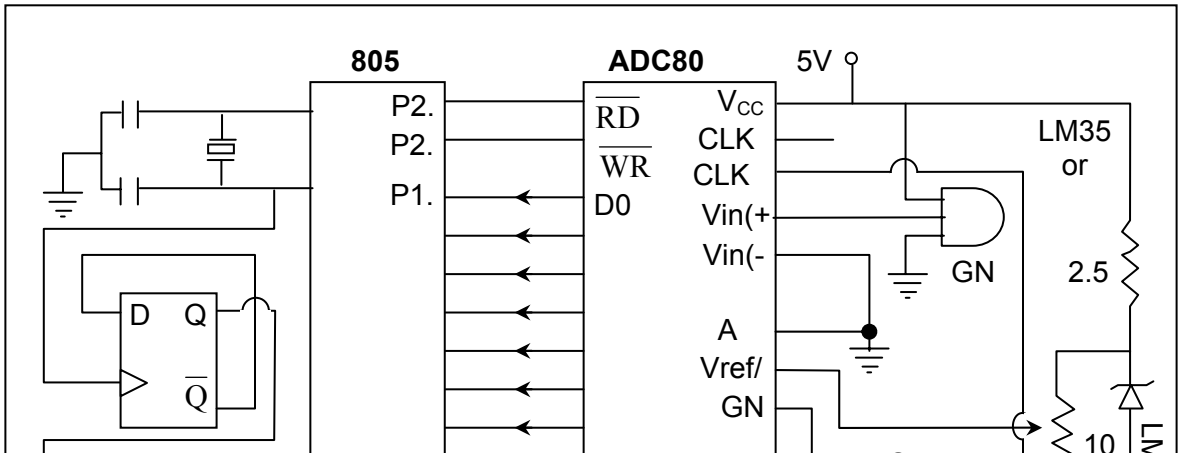
này được gọi chung là phối hợp tín hiệu. Phối hợp tín hiệu có thể là việc chuyển đổi dòng điện thành điện áp hoặc sự khuyếch đại tín hiệu. Ví dụ, bộ cảm biến nhiệt thay đổi trở kháng với nhiệt độ. Sự thay đổi trở kháng phải được chuyển thành điện áp để có thể được sử dụng cho các ADC. Xét trường hợp nối một LM35 tới một ADC 804 vì ADC 804 có độ phân dải 8 bít với tối đa 256 bước (2^8) và LM35 (hoặc ML34) tạo điện áp 10mV cho mỗi sự thay đổi nhiệt độ 1^0C nên ta có thể tạo điều kiện V_{in} của ADC 804 tạo ra một V_{out} = 2560mV (2,56V) cho đầu ra đầu thang đo. do vậy, nhằm tạo ra V_{out} đầy thang 2,56V cho ADC 804 ta cần đặt điện áp $V_{ref}/2$ = 1,28V. Điều này làm cho V_{out} của ADC 804 đáp ứng trực tiếp với nhiệt độ được hiển thị trên LM35 (xem bảng 12.9). Các giá trị của $V_{ref}/2$ được cho ở bảng 12.5.

Bảng 12.9: Nhiệt độ.

Nhiệt độ (^{0}C)	**V_{in} (mV)**	**V_{out} (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 12.9: Thu đo các đại lượng vật lý.



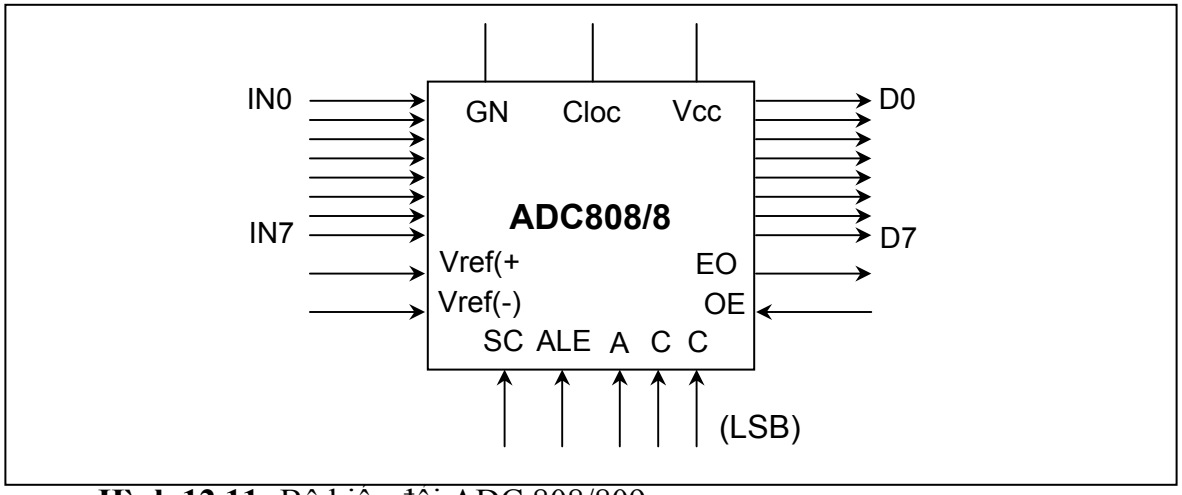
Hình 12.10

Hình 12.10: Nối ghép 8051 với DAC 804 và cảm biến nhiệt độ.

Hình 12.10 biểu diễn nối ghép của bộ cảm biến nhiệt đến ADC 804. Lưu ý rằng ta sử dụng đi ốt zener LM336 - 2.5 để cố định điện áp qua biến trở 10kΩ tại 2,5V. Việc sử dụng LM336 - 2.5 có thể vượt qua được mọi dao động lên xuống của nguồn nuôi.

12.2.7 Chíp ADC 808/809 với 8 kênh tương tự.

Một chíp hữu ích khác của National Semiconductor là ADC 808/809 (xem hình 12.11). Trong khi ADC 804 chỉ có một đầu vào tương tự thì chíp này có 8 kênh đầu vào. Như vậy nó cho phép ta hiển thị lên 8 bộ biến đổi khác nhau chỉ qua một chíp duy nhất. Lưu ý rằng, ADC 808/809 có đầu ra dữ liệu 8 bít như ADC 804. 8 kênh đầu vào tương tự được dồn kênh và được chọn theo bảng 12.10 sử dụng 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 ADC 808.

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

Trong ADC 808/809 thì $V_{rer}(+)$ 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 = 19,53mV. Do vậy, để có độ phân dải 10mV ta cần đặt $V_{ref}(+)$ = 2,56V và $V_{ref}(-)$ = Gnd. Từ hình 12.11 ta thấy có chân ALE. Ta sử dụng các địa chỉ A, B và C để chọn kênh đầu vào IN0 – IN7 và kích hoạt chân ALE để chốt địa chỉ. Chân SetComplete để bắt đầu chuyển đổi (Start Conversion). Chân EOC được dùng để kết thúc chuyển đổi (End - Of - Conversion) và chân OE là cho phép đọc đầu ra (Out put Enable).

12.2.7 Các bước lập trình cho ADC 808/809.

Các bước chuyển dữ liệu từ đầu vào của ADC 808/809 vào bộ vi điều khiển như sau:

1. Chọn một kênh tương tự bằng cách tạo địa chỉ A, B và C theo bảng 12.10.
2. Kích hoạt chân ALE (cho phép chốt địa chỉ Address Latch Enable). Nó cần xung thấp lên cao để chốt địa chỉ.
3. Kích hoạt chân SCbằng xung cao xuống thấp để bắt đầu chuyển đổi.
4. Hiển thị OEC 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 OE cho phép đọc dữ liệu ra của ADC. Một xung cao xuống thấp tới chân OE sẽ đem dữ liệu số ra khỏi chíp ADC.

Lưu ý rằng trong 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 nó không nhanh hơn 100ms.

Chương 14
Phối phép 8031/51 với bộ nhớ ngoài

14.1 Bộ nhớ bán dẫn.

Trong phần này ta nhớ về các kiểu loại bộ nhớ bán dẫn khác nhau và các đặc tính của chúng như dụng lượng, tổ chức và thời gian truy cập. Trong thiết kế của tất cả các hệ thống dựa trên bộ vi sử lý thì các bộ nhớ bán dẫn được dùng như hơi lưu giữ chương trình và dữ liệu chính. Các bộ nhớ bán dẫn được nối trực tiếp với CPU và chúng là bộ nhớ mà CPU đầu tiên hỏi về thông tin chương trình và dữ liệu. Vì lý do này mà các bộ nhớ nhiều khi được coi như là nó phải đáp ứng nhanh cho CPU mà các điều này chỉ có các bộ nhớ bán dẫn mới có thể làm được. Các bộ nhớ bán dẫn được sử dụng rộng rãi nhất là ROM và RAM. Trước khi đi vào bàn các kiểu bộ nhớ ROM và RAM chúng ta làm quen với một số thuật ngữ quan trọng chung cho tất cả mọi bộ nhớ bán dẫn như là dung lượng, tổ chức và tốc độ.

14.1.1 Dung lượng nhớ.

Số lượng các bít mà một chíp nhớ bán dẫn có thể lưu được gọi là dung lượng của chíp, nó có đơn vị có thể là ki-lô-bít (Kbít), mê-ga-bit (Mbít) v.v… Điều này phải phân biết với dung lượng lưu trữ của hệ thống máy tính. Trong khi dung lượng nhớ của một IC nhớ luôn được tính theo bít, còn dung lượng nhớ của một hệ thống máy tính luôn được cho tính byte. Chẳng hạn, trên tạp chí kỹ thuật có một bài báo nói rằng chíp 16M đã trở nên phổ dụng thì mặc dù nó không nói rằng 16M nghĩa là 16 mê-ga-bit thì nó vẫn được hiểu là bài báo nói về chíp IC nhớ. Tuy nhiên, nếu một quảng cáo nói rằng một máy tính với bộ nhớ 16M vì nó đang nói về hệ thống máy tính nên nó được hiểu 16M có nghĩa là 16 mê-ga-byte.

14.1.2 Tổ chức bộ nhớ.

Các chíp được tổ chức vào một số ngăn nhớ bên trong mạch tích hợp IC. Mỗi ngăn nhớ có chứa bộ bít, 4 bít, 8 bít thậm chí đến 16 bít phụ thuộc vào cách nó được thiết kế bên trong như thế nào? Số bít mà mỗi ngăn nhớ bên trong chíp nhớ có thể chứa được luôn bằng số chân dữ liệu trên chíp. Vậy có bao nhiêu ngăn nhớ bên trong một chíp nhớ? Nó phụ thuộc vào số chân địa chỉ, số ngăn nhớ bên trong một IC nhớ luôn bằng 2 luỹ thừa với số chân địa chỉ. Do vậy, tổng số bít mà IC nhớ có thể lưu trữ là bằng số ngăn nhớ nhân với bít dữ liệu trên mỗi ngăn nhớ. Có thể tóm tắt lại như sau:

1. Một chíp nhớ có thể chứa 2^x ngăn nhớ, với x là số chân địa chỉ.
2. Mỗi ngăn nhớ chứa y bít với y là số chân dữ liệu trên chíp.
3. Toàn bộ chíp chứa ($2^x \times y$)bít với x là số chân địa chỉ và y là số chân dữ liệu trên chíp.

14.1.3 Tốc độ.

Một trong những đặc tính quan trọng nhất của chíp nhớ là tốc độ truy cập dữ liệu của nó. Để truy cập dữ liệu thì địa chỉ phải có ở các chân địa chỉa, chân đọc READ được tích cực và sau một khoảng thời gian thì dữ liệu sẽ xuất hiện ở các chân dữ liệu. Khoảng thời gian này càng ngắn càng tốt và tất nhiên là chíp nhớ càng đắt. Tốc độ của chíp nhớ thường được coi như là thời gian truy cập của nó. Thời gian truy cập của các chíp nhớ thay đổi từ vài na-nô-giây đến hàng trăm na-nô-giây phụ thuộc vào công nghệ sử dụng trong quá trình thiết kế và sản xuất IC.

Cả ba đặc tính quan trọng của bộ nhớ là dung lượng nhớ, tổ chức bộ nhớ và thời gian truy cập sẽ được sử dụng nhiều trong chương trình. Bảng 14.1 như một tham chiếu để tính toán các đặc tính của bộ nhớ. Các ví dụ 14.1 và 14.2 sẽ minh hoạ những khái niệm vừa trình bày.

Bảng 14.1: Dung lượng bộ nhớ với số chân địac chỉ của IC.

x	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
$\mathbf{2^x}$	1K	2K	4K	8K	16 K	32 K	64 K	12 8K	25 6K	51 2K	1 M	2 M	4 M	8 M	16 M

Ví dụ 14.1:

Một chíp nhớ có 12 chân địa chỉ và 4 chân dữ liệu. Hãy tìm tổ chức bộ nhớ và dung lượng nhớ của nó.

Lời giải:

a) Chíp nhớ này có 4096 ngăn nhớ ($2^{12} = 4096$) và mỗi ngăn nhớ chứa 4 bít dữ liệu nên tổ chức nhớ của nó là 4096×4 và thường được biểu diễn là $4K \times 4$.

b) Dung lượng nhớ của chíp nhớ là 16K vì có 4 K ngăn nhớ và mỗi ngăn nhớ có 4 bít dữ liệu.

Ví dụ 14.2:

Một chíp nhớ 512k có 8 chân dữ liệu. Hãy tìm a) tổ chức của nó và b) số chân địa chỉ của chíp này.

Lời giải:

a) Một chíp có 8 chân dữ liệu có nghĩa là mỗi ngăn nhớ có 8 bít dữ liệu. Số ngăn nhớ trên chíp này bằng dung lượng nhớ chia cho số chân dữ liệu = 512k/8 = 64. Do vậy tổ chức nhớ của chíp là $64k \times 8$.

b) Số đường địa chỉ của chíp sẽ là 16 vì $2^{16} = 64k$.

14.1.4 Bộ nhớ ROM.

Bộ nhớ ROM là bộ nhớ chỉ đọc (Read - only Memory). Đây là một kiểu bộ nhớ mà không mất các nội dung của nó khi tắt nguồn. Với lý do này mà bộ nhớ ROM còn được gọi là bộ nhớ không bay hơi, có nhiều loại bộ nhớ ROM khác nhau như: PROM, EPROM, EEPROM, EPROM nhanh (flash) và ROM che.

14.1.4.1 Bộ nhớ PROM.

Bộ nhớ PROM là bộ nhớ ROM có thể lập trình được. Đây là loại bộ nhớ mà người dùng có thể đốt ghi thông tin vào. hay nói cách khác, PROM là bộ nhớ người dùng có thể lập trình được. Đối với mỗi bít của PROM có một cầu chì. Bộ nhớ PROM được lập trình bằng cách làm đứt những cầu trì. Nếu thông tin được đốt vào trong PROM mà sau thì phải bỏ vì các cầu trì của nó bên trong bị đứt vĩnh viễn với lý do này mà PROM mà được gọi là bộ nhớ ROM lập trình một lần. Việc lập trình ROM cũng được gọi là đốt ROM và nó đòi hỏi phải có một thiết bị đặc biệt gọi là bộ đốt ROM hay còn gọi là thiết bị lập trình ROM.

14.1.4.2 Bộ nhớ EPROM và UV - EPROM.

Bộ nhớ EPROM được phát minh ra để cho phép thực hiện thay đổi nội dung của PROM sau khi nó đã được đốt. Trong bộ nhớ EPROM ta có thể lập trình chíp nhớ và xoá nó hàng nghìn lần. Điều này là đặc biệt cần thiết trong quá trình phát triển mẫu thử của một dự án dựa trên bộ vi xử lý. Một EPR sử dụng rộng rãi được

gọi là UV - EPROM (UV là chữ viết tắt của tia cực tím Ultra - Violet). Vấn đề tồn tại duy nhất của UV - EPROM là thời gian xoá của nó quá lâu (20 phút).

Tất cả các chíp nhớ UV - EPROM có một cửa sổ được dùng để chiếu tia bức xạ cực tím xoá các nội dung của nó. Với lý do này mà EPROM cũng còn được coi như là bộ nhớ EPROM được xoá bằng tia cực tím hay đơn giản được gọi là UV – EPROM. Hình 14.1 trình bày các chân của một chíp UV - EPROM.

Để lập trình cho một UV - EPROM cần thực hiện các bước.

1. Xoá các nội dung của nó, để xoá một chíp thì phải tháo nó ra khỏi để cắm trên bảng mạch hệ thống và đặt nó vào thiết bị xoá EPROM để chiếu xạ tia cực tím khoảng 15 - 20 phút.
2. Lập trình cho chíp. Để lập trình cho một chíp UV - EPROM thì đặt nó vào thiết bị đốt (thiết bị lập trình). Để đốt chương trình và dữ liệu vào EPROM thì thiết bị đốt ROM sử dụng điện áp 12.5V hoặc cao hơn phụ thuộc vào loại EPROM. Điện áp này được gọi là Vpp trong bảng dữ liệu của UV - EPROM.
3. Lắp chíp nhớ trở lại để cắm trên hệ thống.

Từ các bước trên đây ta thấy cũng như các thiết bị đốt EPROM thì cũng có các thiết bị xoá EPROM khác nhau. Và tất cả các kiểu bộ nhớ UV – EPROM đều có một nhược điểm chính là không thể được lập trình trực tiếp trên bảng mạch của hệ thống. Do vậy, bộ nhớ EPROM đã được ra đời để giải quyết vấn đề này.

Hãy để ý đến các ký hiệu mã số của các IC trong bảng 14.2. ví dụ, mã số bộ phận 27128 - 25 dành cho UV - EPROM có dung lượn và thời gian truy cập là 250ns. Dung lượng của chíp nhỏ được ký hiệu trên mã số bộ phận (part number) và thời gian truy cập được bỏ đi một số 0. Trong các mã số bộ phận thì chữ C là cộng nghệ CMOS, còn 27xx luôn chỉ các chíp nhớ EPROM. Để biết thêm chi tiết ta vào mục sổ tay các chíp nhớ của các hãng chẳng hạn JAMECO (jameco.com) hay JDR (jdr.com) theo đường mạng internet.

Bảng 14.2: Một số chíp nhớ UV - EPROM.

HìNH 14.1

Hình 14.1: Bố trí các chân của họ các chíp nhớ ROM 27xx.

Ví dụ 14.3:

Đối với ROM có mã bộ phận là 27128 có dung lượng nhớ là 128k bít. Tra bảng ta thấy tổ chức của nó là 16k × 8 (tất cả mọi ROM đều có 8 chân dữ liệu) điều này nói lên rằng nó có 8 chân dữ liệu và số chân địa chỉ được tìm thấy là 14 vì 2^{14} = 16k.

14.1.4.3 Bộ nhớ PROM có thể xoá được bằng điện EEPROM.

Bộ nhớ EEPROM có một số ưu điểm so với EPROM là do vì được xoá bằng điện nên quá trình xoá rất nhanh, nhanh rất nhiều so với thời gian xoá 20 phút của UEPROM. Ngoài ra trong EEPROM ta phải xoá toàn bộ nội dung của ROM. Tuy nhiên, ưu điểm chính của EEPROM là ta có thể lập trình và xoá khi chíp nhớ vẫn ở trên giá cắm của bảng mạch hệ thống mà không phải lấy ra như đối với UV – EPROM. Hay nói cách khác là EPROM không cần thiết bị lập trình và thiết bị xoá ngoài như đối với UV - EPROM. Để hoàn toàn tạo thuận lợi cho EPROM nhà thiết

kế phải kết hợp vào bảng mạch của hệ thống cả mạch điện để lập trình EEPROM sử dụng điện áp Vpp = 5V nhưng chúng rất đắt. nhìn chung, giá thành cho 1 bít của EEPROM đắt hơn rất nhiều so với UV - EPROM.

Bảng 14.3: Một số chíp EEPROM và chíp nhớ Flash (nhanh).

Cuối trang 160

CHƯƠNG 15
Phép ghép 8031/51 với 8255

Như đã nói ở chương 14 trong quá trình nối ghép 8031/51 với bộ nhớ ngoài thì hai cổng P0 và P2 bị mất. Trong chương này chúng ta sẽ trình bày làm thế nào để mở rộng các cổng vào/ ra I/O của 8031/51 bằng việc nối nó tới chíp 8255.

15.1 Lập trình 8255.

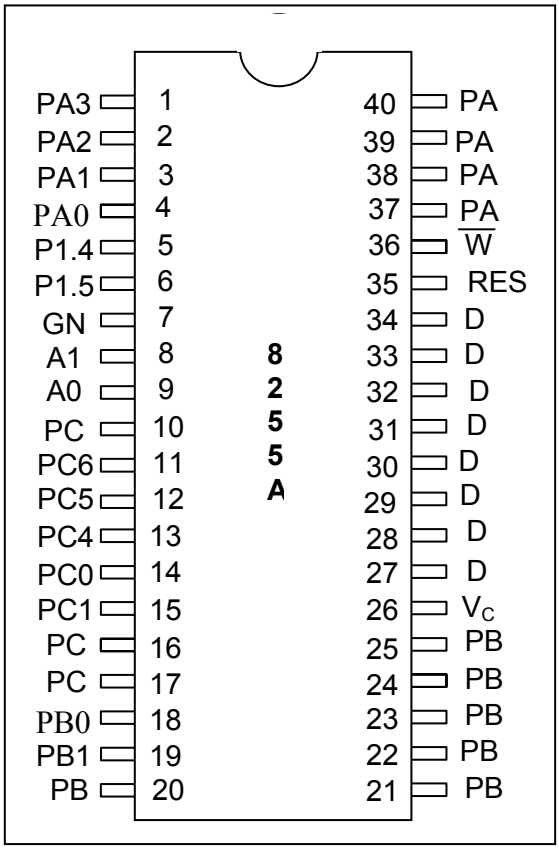
Trong mục này ta nghiên cứu 8255 như là một trong những chíp vào/ ra được sử dụng rộng rãi nhất. Trước hết ta mô tả những đặc tính của nó và sau đsó chỉ ra cách nối 8031/51 với 8255 như thế nào?

15.1 Lập trình 8255.

Trong mục này ta nghiên cứu 8255 như là một trong những chíp vào/ ra được sử dụng rộng rãi nhất. Trước hết ta mô tả những đặc tính của nó và sau đó chỉ ra cách nối 8031/51 với 8255 như thế nào?

15.1.1 Các đặc tính của 8255.

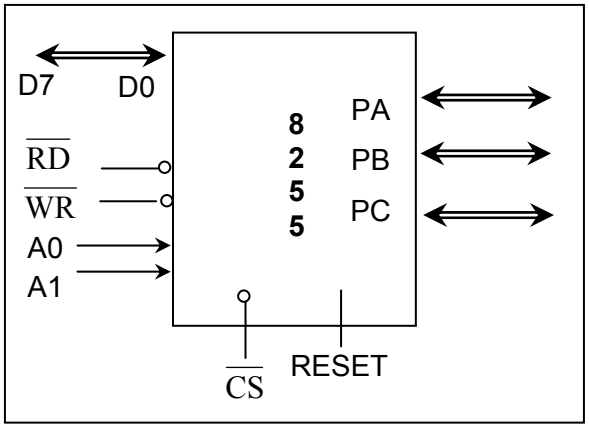
8255 là một chíp DIP 40 chân (xem hình 15.1). Nó có 3 cổng truy cập được riêng biệt. Các cổng đó có tên A, B và C đều là các cổng 8 bit. Các cổng này đều có thể lập trình như cổng đầu vào hoặc đầu ra riêng rẽ và có thể thay đổi một cách năng động. Ngoài ra, các cổng 8255 có khả năng bắt tay. Do vậy cho phép giao diện với các thiết bị khác cũng có giá trị tín hiệu bắt tay như các máy in chẳng hạn. Khả năng bắt tay của 8255 sẽ được bàn tới ở mục 15.3.



Hình 15.1: Chíp 8255.

15.1.1.1 Các chân PA0 - PA7 (cổng A).

Cả 8 bít của cổng A PA0 - PA7 có thể được lập trình như 8 bit đầu vào hoặc 8 bit đầu ra hoặc cả 8 bít hai chiều vào/ ra.S



Hình 15.2: Sơ đồ khối của 8255.

15.1.1.2 Các chân PB0 - PB7 (cổng B).

Cả 8 bit của cổng B có thể được lập trình hoặc như 8 bit đầu vào hoặc 8 bit đầu ra hoặc cả 8 bit hai chiều vào/ ra.

15.1.1.3 Các chân PC0 - PC7 (cổng C).

Tất cả 8 bit của cổng C (PC0 - PC7) đều có thể được lập trình như các bit đầu vào hoặc các bit đầu ra. 8 bit này cũng có thể được chia làm hai phần: Các bit cao (PC4 - PC7) là CU và các bit thấp (PC0 - PC3) là CL. Mỗi phần có thể được dùng hoặc 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.

15.1.1.4 Các chân $\overline{RD}$ và $\overline{WR}$.

Đây là hai tín hiệu điều khiển tích cực mức thấp tới 8255 được nối tới các chân dữ liệu $\overline{RD}$ và $\overline{WR}$ từ 8031/51 được nối tới các chân đầu vào này.

15.1.1.5 Các chân dữ liệu 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 nó gửi dữ liệu qua lại giữa bộ vi điều khiển và chíp 8255.

15.1.1.6 Chân RESET.

Đây là đầu vào tín hiệu tích cực mức cao tới 8255 được dùng để xoá thanh ghi điều khiển. Khi chân RESET được kích hoạt thì tất cả các cổng được khởi tạo lại như 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 kích hoạt nó. Cũng như tất cả các chân đầu vào của IC thì nó cũng có thể để hở.

15.1.1.7 Các chân A0, A1 và $\overline{CS}$.

Trong khi $\overline{CS}$ chọn toàn bộ chíp thì A0 và A1 lại chọn các cổng riêng biệt. Các chân này được dùng để truy cập các cổng A, B, C hoặc thanh ghi điều khiển theo bảng 15.1. Lưu ý $\overline{CS}$ là tích cực mức thấp.

15.1.2 Chọn chế độ của 8255.

Trong khi các cổng A, B và C được dùng để nhập và xuất dữ liệu thì thanh ghi điều khiển phải được lập trình để chọn chế độ làm việc của các cổng này. Các cổng của 8255 có thể được lập trình theo một chế độ bất kỳ dưới đây.

1. Chế độ 0 (Mode0): Đây là chế độ vào/ ra đơn giản. Ở chế độ này các cổng A, B CL và CU có thể được lập trình như đầu vào hoặc đầu ra. Trong chế độ này thì tất cả các bit hoặc là đầu vào hoặc là đầu ra. Hay nói cách khác là không có điều khiển theo từng bit riêng rẽ như ta đã thấy ở các cổng P0 - P3 của 8051. Vì đa phần các ứng dụng liên quan đến 8255 đều sử dụng chế độ vào/ ra đơn giản này nên ta sẽ tập chung đi sâu vào chế độ này.

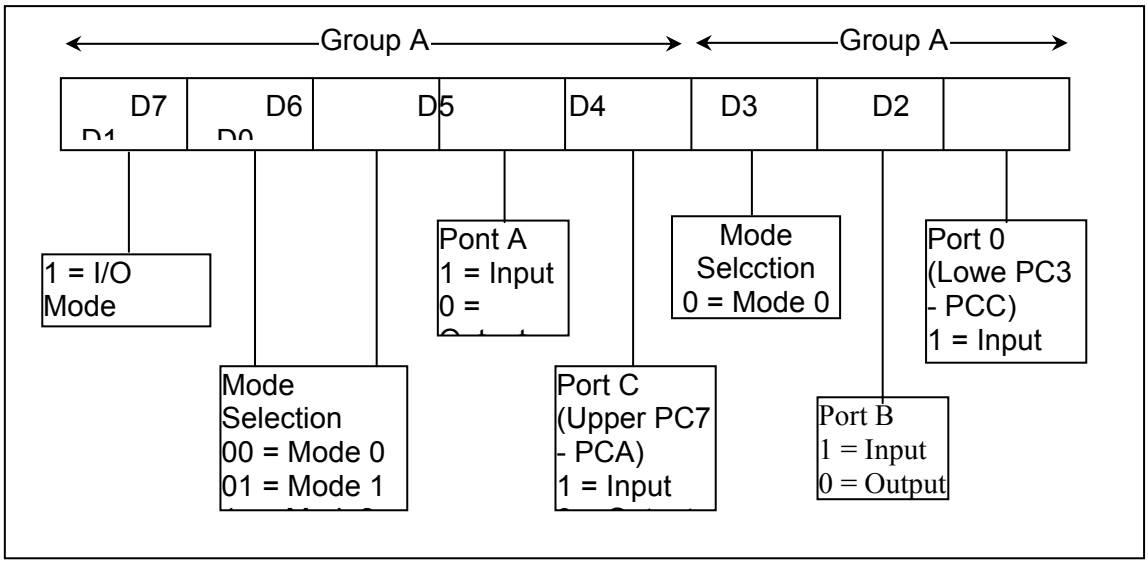
2. Chế độ 1 (Mode1): Trong chế độ này các cổng A và B có thể được dùng như các cổng đầu vào hoặc đầu ra với các khả năng bắt tay. Tín hiệu bắt tay được cấp bởi các bit của cổng C (sẽ được trình bày ở mục 15.3).

3. Chế độ 2 (Mode2): Trong chế độ này cổng A có thể được dùng như cổng vào/ ra hai chiều với khả năng bắt tay và 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 có bắt tay Mode1. Chế độ này sẽ không được trình bày trong tài liệu này.

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 15.3).

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



Hình 15.3: Định dạng từ điều khiển của 8255 (chế độ vào/ ra).

15.1.3 Lập trình chế độ vào/ ra đơn giản.

Hãng Intel gọi chế độ 0 là chế độ vào/ ra cơ sở. Một thuật ngữ được dùng chung hơn là vào/ ra đơn giản. Trong chế độ này thì một cổng bất kỳ trong A, B, C được lập trình như là cổng đầu vào hoặc cổng đầu ra. Cần lưu ý rằng trong chế độ này một cổng đã cho không thể vừa làm đầu vào lại vừa làm đầu ra cùng một lúc.

Ví dụ 15.1:

Hãy tìm từ điều khiển của 8255 cho các cấu hình sau:

Tất cả các cổng A, B và C đều là các cổng đầu ra (chế độ 0).

PA là đầu vào, PB là đầu ra, PCL bằng đầu vào và PCH bằng đầu ra.

Lời giải:

Từ hình 15.3 ta tìm được:

a) 1000 0000 = 80H; b) 1001 000 = 90H

15.1.4 Nối ghép 8031/51 với 8255.

Chíp 8255 được lập trình một trong bốn chế độ vừa trình bày ở trên bằng cách gửi một byte (hãng Intel gọi là một từ điều khiển) tới thanh ghi điều khiển của 8255. Trước hết chúng ta phải tìm ra các 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 nhìn thấy từ hình 15.4 thì 8255 được nối tới một 8031/51 như thế nó là bộ nhớ RAM. Để việc sử dụng các tín hiệu $\overline{RD}$ và $\overline{WR}$. Phương pháp nối một chíp vào/ ra bộ nhớ vì nó được ánh xạ vào không gian bộ nhớ. Hay nói cách khác, ta sử dụng không gian bộ nhớ để truy cập các thiết bị vào/ ra. Vì ly??? do này mà ta dùng lệnh MOVX để truy cập RAM và ROM. Đối với một 8255 được nối tới 8031/51 thì ta cũng phải dùng lệnh MOVX để truyền thông với nó. Điều này được thể hiện trên ví dụ 15.2.

Ví dụ 15.2:

Đối với hình 15.4:

a) Hãy tìm các địa chỉ vào/ ra được gán cho cổng A, B, C và thanh ghi điều khiển.

b) Hãy lập trình 8255 cho các cổng A, B và C thành các cổng đầu ra.

c) Viết một chương trình để gửi 55H và AAH đến cổng liên tục.

Lời giải:

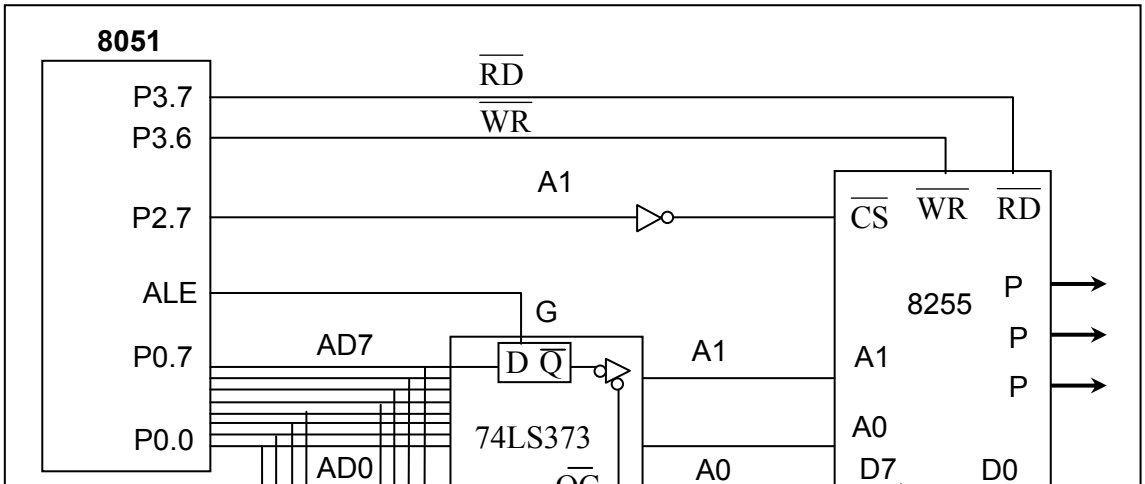
a) Địa chỉ cơ sở dành cho 8255 như sau:

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 như đầu ra là 80H như được tính ở ví dụ 15.1.

c)

```
                MOV     A, #80H                 ; Từ điển khiển
                MOV     DPTR, # 4003H           ; Nạp địa chỉ cổng của
thanh ghi 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        ; Lấy các bit cổng PA
                INC     DPTR            ; Địa chỉ cổng PB
                MOVX    @DPTR, A        ; Lấy các bít cổng PB
                INC     DPTR            ; Địa chỉ cổng PC
                MOVX    @DPTR, A        ; Lấy các bít cổng PC
                CPL     A               ; Lấy các bít thanh ghi A
                ACALL   DELAY                   ; Chờ
                SJMP    AGAIN           ; Tiếp tục
```



Hình 15.4: Nối ghép 8051 với 8255 cho ví dụ 15.2.

Ví dụ 15.3:

Đối với hình 15.5:

a) Tìm các địa chỉ cổng vào ra được gán cho các cổng A, B, C và thanhg ghi điều khiển.

b) Tìm byte điều khiển đối với PA bằng đầu vào, PB bằng đầu ra, PC bằng đầu ra

c) Viết một chương trình để nhận dữ liệu từ PA gửi nó đến cả cổng B và cổng C.

Lời giải:

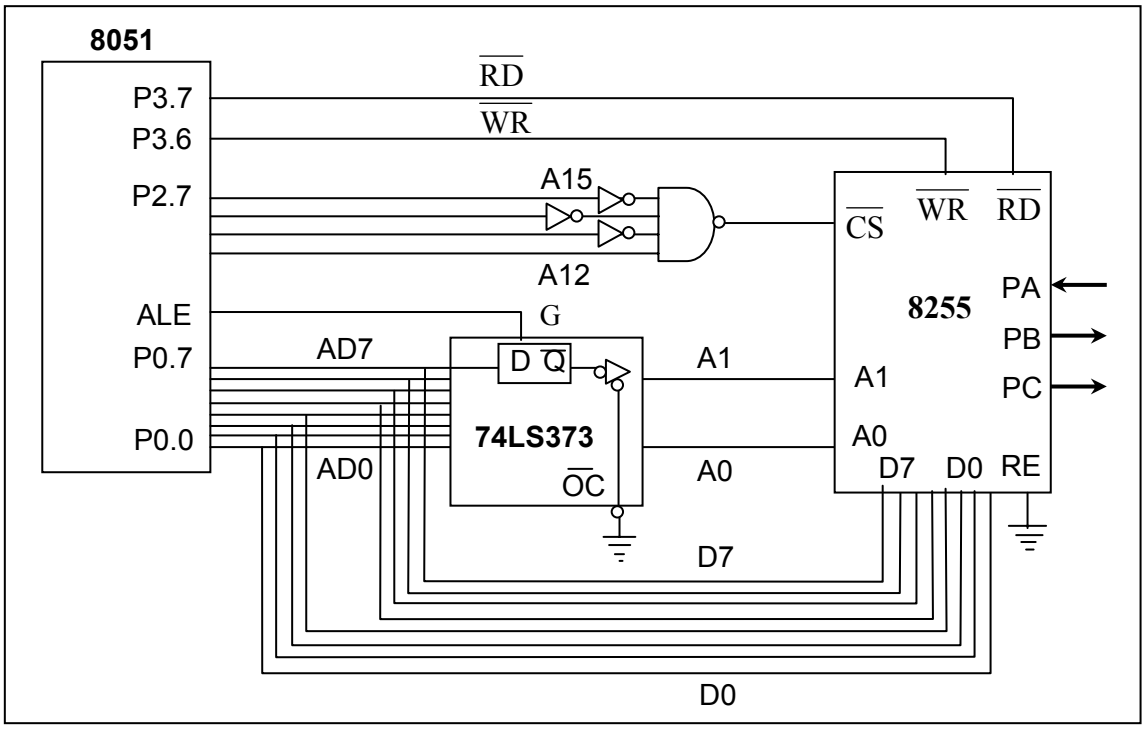
a) Giả sử tất các các bit không dùng đến là 0 thì địa chỉ cổng cơ sở cho 8255 là 1000H. Do vậy ta có:

1000H là PA; 1001H là PB; 1002H là PC và 1003H là thanh ghi điều khiển.

b) Từ điều khiển cho trường hợp này là 10010000 hay 90H.

c)

```
        MOV     A, #90H         ; PA là đầu vào, PB là đầu ra, PC là đầu ra
        MOV     DPTR, #1003H    ; Nạp địa chỉ cổng của thanh ghi điều khiển
        MOVX    @DPTR, A        ; Xuất từ điều khiển
        MOV     DPTR, #1000H    ; 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 tới 8255 cho ví dụ 15.3.

Đối với ví dụ 15.3 ta nên dùng chỉ lệnh EQU cho 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 là đầu ra, PC là đầu ra
        MOV         DPTR, #CNTPORT  ; Nạp địa chỉ của cổng thanh ghi 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                   Xác định PA đầu vào, PB và PC đầu 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 ta đã sử dụng thanh ghi DPTR vì địa chỉ cơ sở gán cho 8255 là 16 bit. Nếu địa chỉ cơ sở dành cho 8255 là 8 bit, ta có thể sử dụng các lệnh "MOVX A, @R0" và "MOVX @R0, A" trong đó R0 (hoặc R1) giữ địa chỉ cổng 8 bit của cổng. Xem ví dụ 15.4, chú ý rằng trong ví dụ 15.4 ta sử dụng một cổng lôgíc đơn giản để giải mã địa chỉa cho 8255. Đối với hệ thống có nhiều 8255 ta có thể sử dụng 74LS138 để giải mã như sẽ trình bày ở ví dụ 15.5.

15.1.5 Các bí danh của địa chỉ (Addren Alias).

Trong các ví dụ 15.4 và 15.4 ta giải mã các bít địa chỉ A0 - A7, tuy nhiên trong ví dụ 15.3 và 15.2 ta đã giải mã một phần các địa chỉ cao của A8 - A15. Việc giải mã từng phần này dẫn đến cái gọi là các bí danh của địa chỉ (Address Aliases). Hay nói cách khác, cùng cổng vật lý giống nhau có các địa chỉ khác nhau, do vậy cùng một cổng mà được biết với các tên khác nhau. Trong ví dụ 15.2 và 15.3 ta có thể thay đổi tốt x thành các tổ hợp các số 1 và 0 khác nhau thành các địa chỉ khác nhau, song về thực chất chúng tham chiếu đến cùng một cổng vật lý. Trong tài liệu thuyết minh phần cứng của mình chúng ta cần phải bảo đảm ghi chú đầy đủ các bí danh địa chỉa nếu có sao cho mọi người dùng biết được các địa chỉ có sẵn để họ có thể mở rộng hệ thống.

Ví dụ 15.4:

Cho hình 15.6:

a) Hãy tìm các địa chỉ cổng vào/ ra được gán cho 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 một chương trình để nhận dữ liệu từ PB và gửi nó ra PA. Ngoài ra, dữ liệu từ PC1 được gửi đến CPU.

Lời giải:

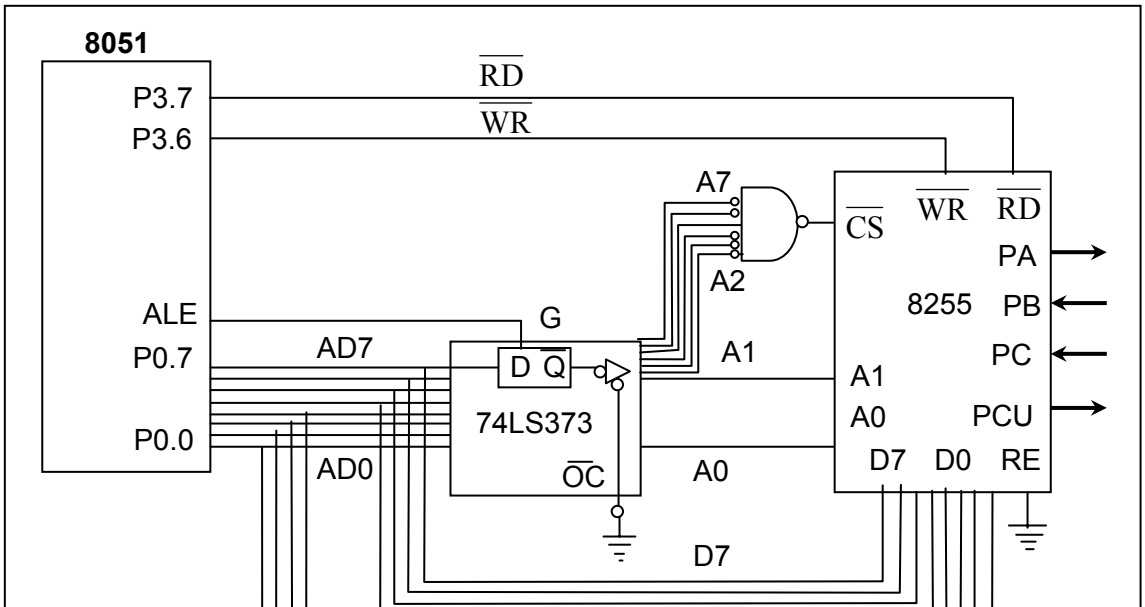
a) Các địa chỉ cổng được tìm thấy như sau:

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 là đầu ra, PB,PCL là đầu vào
APORT               EQU         20H
BPORT               EQU         21H
CPORT               EQU         22H
CNTPORT         EQU         23H
                ...
                MOV         A, #CONTRBYT
                MOV         A, #CONTRBYT        ; PA, PCU là đầu ra, PB và
PCL là đầu vào
                MOV         R0, #CNTPORT        ; Nạp địa chỉ của cổng
        thanh ghi 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           ; Trao đổi phần cao và thấp
                MOVX        @R0, A              ; Gửi đến PCU
```



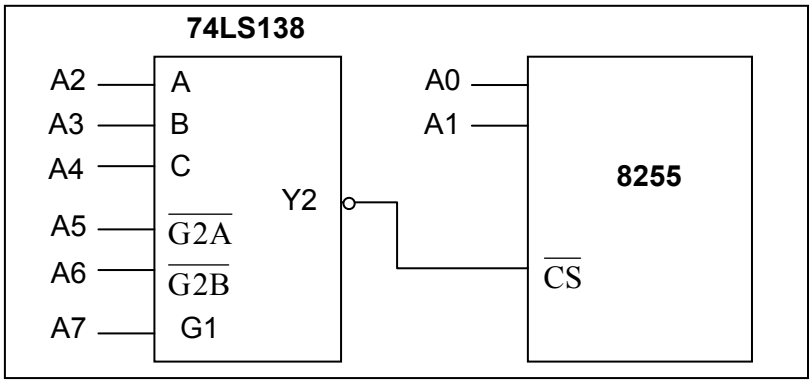
Hình 15.6: Nối ghép 8051 với 8255 cho ví dụ 15.4.

Ví dụ 15.5:

Hãy tìm địa chỉ cơ sở cho 8255 trên hình 15.7.

Lời giải:

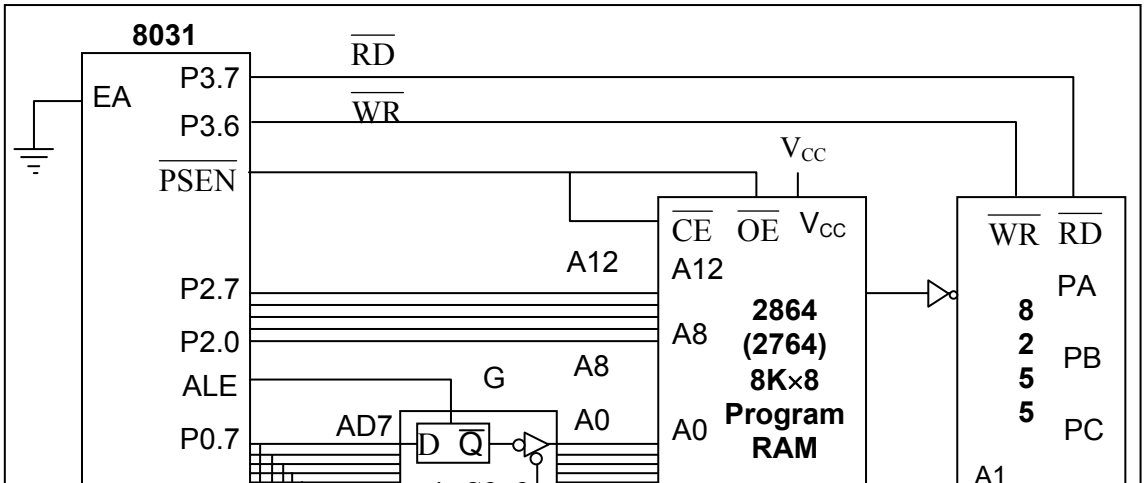
GA	$\overline{G2B}$	$\overline{G2A}$	C	B	A			Địa chỉ
A7	A6	A5	A4	A3	A2	A1	A0	
1	0	0	0	1	0	0	0	88H



Hình 15.7: Giải mã địa chỉ của 8255 sử dụng 74LS138.

15.1.6 Hệ 8031 với 8255.

Trong một hệ thống dựa trên 8031 mà bộ nhớ chương trình ROM ngoài là một sự bắt buộc tuyệt đối thì sử dụng một 8255 là rất được trào đón. Điều này là do một thực tế là trong giải trình phối ghép 8031 với bộ nhớ chương trình ROM ngoài ta bị mất hai cổng P0 và P2 và chỉ còn lại duy nhất cổng P1. Do vậy, việc nối với một 8255 là cách tốt nhất để có thêm một số cổng. Điều này được chỉ ra trên hình 15.8.

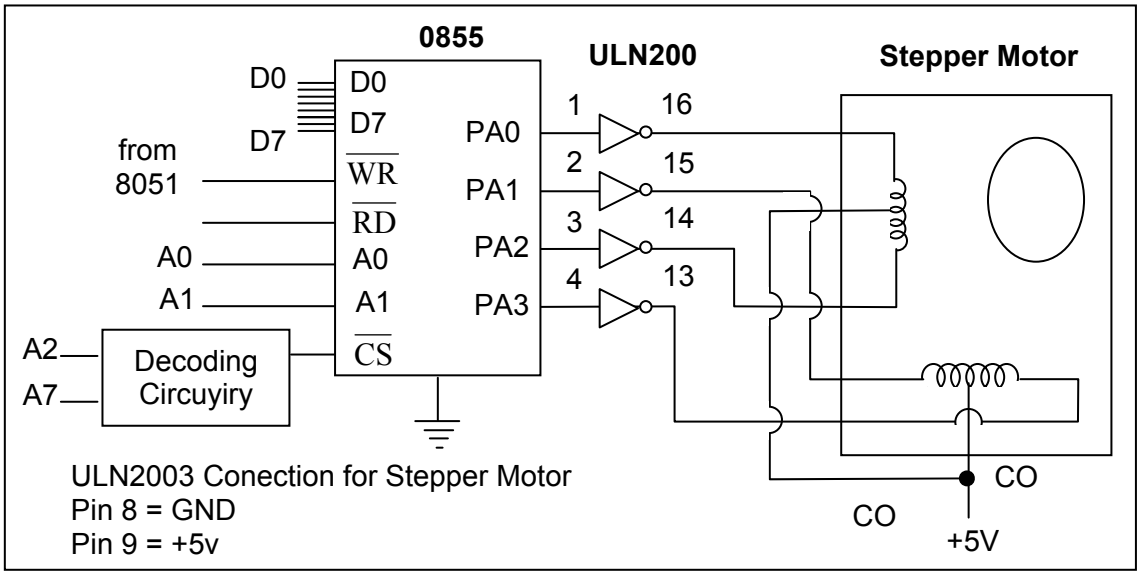


Hình 15.8: Nối 8031 tới một ROM chương trình ngoài và 8255.

15.2 Nối ghép với thế giới thực.

15.2.1 Phối ghép 8255 với động cơ bước.

Chương 13 đã nói chi tiết về phối ghép động cơ bước với 8051, ở đây ta trình bày nối ghép động cơ bước tới 8255 và lập trình (xem hình 15.9).



Hình 15.9: Nối ghép 8255 với một động cơ bước.

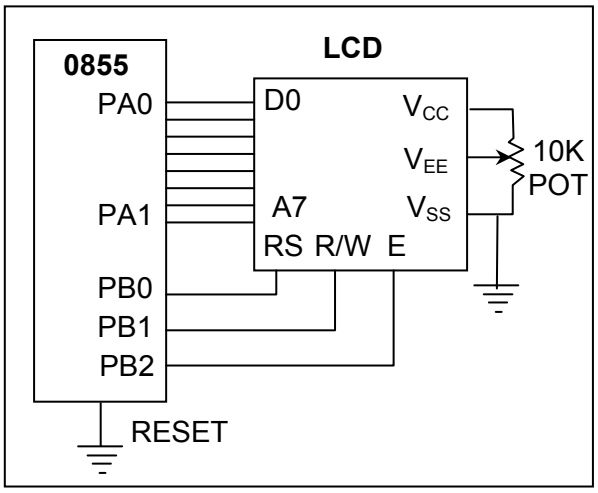
Chương trình cho sơ đồ nối ghép này như sau:

```
                MOV         A, #80H             ; Chọn 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             ; Gán A = 66H, chuyển xung của
động cơ bước
 AGAIN:                 MOVX        @R1, A              ; Xuất chuỗi động cơ đến
PA
```

```
        RR      A       ; Quay chuỗi theo chiều kim đồng hồ
        ACALL   DELAY           ; Chờ
        SJMP    AGAIN
```

15.2.2 Phối ghép 8255 với LCD.

Chương trình 15.1 trình bày cách xuất các lệnh và dữ liệu tới một LCD được nối tới 8255 theo sơ đồ hình 15.10. Trong chương trình 15.1 ta phải đặt một độ trễ trước mỗi lần xuất thông tin bất kỳ (lệnh hoặc dữ liệu) tới LCD. Một cách tốt hơn là kiểm tra cờ bận trước khi xuất bất kỳ thứ gì tới LCD như đã nói ở chương 12. Chương trình 15.2 lặp lại chương trình 15.1 có sử dụng kiểm tra cờ bận. Để ý rằng lúc này không cần thời gian giữ chậm như ở vị trí 15.1



Hình 5.10: Nối ghép 8255 với LCD.

Chương 15.1:

```
; Ghi các 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
; IB - RS, PB1 = R/W, PB2 = E để nối các chân điều khiển LCD
        MOV     A, #80H         ; Đặt tất cả các cổng 8255 là đầu ra
        MOV     R0, #CNTPORT    ; Nẹp địa chỉ thanh ghi điều khiển
        MOVX    @R0, A          ; Xuất từ điều khiển
        MOV     A, #38H         ; Cấu hình LCD có hai dòng và ma trận 5×7
        ACALL   CMDWRT  ; Ghi lệnh ra LCD
        ACALL   DELAY           ; Chờ đến lần xuất kế tiếp (2ms)
        MOV     A, # 0EH  ; Bật con trỏ cho LCD
        ACALL   CMDWRT  ; Ghi lệnh này ra LCD
        ACALL   DELAY           ; Chờ lần xuất kế tiếp
        MOV     A, # 01H        ; Xoá LCD
        ACALL   CMDWRT  ; Ghi lệnh này ra LCD
        ACALL   DELAY           ; Dịch con trỏ sang phải
        MOV     A, # 06   ; Ghi lệnh này ra LCD
        ACALL   CMDWRT  ; Chờ lần xuất sau
        ACALL   DELAY           ; Ghi lệnh này ra LCD
        ...                     ; v.v... cho tất cả mọi lệnh LCD
        MOV     A, # 'N'  ; Hiển thị dữ liệu ra (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
        ...                     ; v.v... cho các dữ liệu khác
; Chương trình con ghi lệnh CMDWRT ra LCD
CMDWRT: MOV     R0, # APORT ; Nạp địa chỉ cổng A
        MOVX    @R0, A          ; Xuất thông tin tới chân dữ liệu của LCD
        MOV     R0, # BPORT ; Nạp địa chỉ cổng B
```

```
                MOV         A, # 00000100B      ; RS=0, R/W=1, E=1 cho xung cao
xuống thấp
                MOVX        @R0, A              ; Kích hoạt các chânRS, R/W, E
của LCD
                NOP                         ; Tạo độ xung cho chân E
                NOP
                MOV         A, # 00000000B      ; RS=0, R/W=1, E=1 cho xung cao
xuống thấp
                MOVX        @R0, A              ; Chốt thông tin trên chân dữ liệu
        của LCD
                RET
 ; Chương trình con ghi lệnh DATAWRT ghi dữ liệu ra LCD.
 CMDWRT:    MOV         R0, # APORT ; Nạp địa chỉ cổng A
                MOVX        @R0, A              ; Xuất thông tin tới chân dữ liệu
của LCD
                MOV         R0, # BPORT ; Đặt RS=1, R/W=0, E=0 cho xung cao
xuống thấp
                MOV         A, # 00000101B      ; Kích hoạt các chân RS, R/W, E
                MOVX        @R0, A              ; Tạo độ xung cho chân E
                NOP
                NOP
                MOV         A, # 00000001B      ; Đặt RS=1, R/W=0, E=0 cho xung
cao xuống thấp
                MOVX        @RC, A              ; Chốt thông tin trên chân dữ liệu
của LCD
                RET
```

Chương trình 15.2:

```
 ; Ghi các 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 đối với 8255 tới các chân điều khiển LCD

                MOV         A, #80H             ; Đặt tất cả các cổng 8255 là đầu
ra
                MOV         R0, #CNTPORT        ; Nạp địa chỉ thanh ghi điều khiển
                MOVX        @R0, A              ; Xuất từ điều khiển
                MOV         A, #38H             ; Chọn LCD có hai dòng và ma trận
5×7
                ACALL       NMDWRT      ; Ghi lệnh ra LCD
                MOV         A, # 0EH    ; Lệnh của LCD cho con trỏ bật
                ACALL       NMDWRT      ; Ghi lệnh ra LCD
                MOV         A, # 01H    ; Xoá LCD
                ACALL       NMDWRT      ; Ghi lệnh ra LCD
                MOV         A, # 06     ; Lệnh dịch con trỏ sang phải
                ACALL       CMDWRT      ; Ghi lệnh ra LCD
                ...                     ; v.v... cho tất cả mọi lệnh LCD
                MOV         A, # 'N'    ; Hiển thị dữ liệu ra (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
                ...                     ; v.v... cho các dữ liệu khác
 ; Chương trình con ghi lệnh NCMDWRT có hiển thị cờ bận
 NCMDWRT:   MOV         R2,  A      ; Lưu giá trị thanh ghi A
                MOV         A, #90H             ; Đặt PA là cổng đầu vào để đọc
trạng thái LCD
```

```
                MOV     R0, # CNTPORT       ; Nạp địa chỉ thanh ghi 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 các chân RD và RS

                MOV     R0, APORT   ; Nạp địa chỉ cổng A
READY:          MOVX    @R0         ; Đọc thanh ghi lệnh
                RLC     A           ; Chuyển D7 (cờ bận) vào bit nhớ carry
                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 các chân dữ
liệu của LCD
                MOV     R0, #BPORT  ; Nạp địa chỉ cổng B
                MOV     A, #00000100B       ; Đặt RS=0, R/W=0, E=1 cho xung
thấp lên cao
                MOVX    @R0, A              ; Kích hoạt RS, R/W, E của LCD
                NOP                 ; Tạo độ rộng xung của chân E
                NOP
                MOV     A, #00000000B       ; Đặt RS=0, R/W=0, E=0 cho xung
cao xuống thấp
                MOVX    @R0, A              ; Chốt thông tin ở chân dữ liệu
LCD
                RET
; Chương trình con ghi dữ liệu mới NDATAWRT sử dụng cờ bận
NCMDWRT:        MOV     R2,  A      ; Lưu giá trị thanh ghi A
                MOV     A, #90H             ; Đặt PA là cổng đầu vào để đọc
trạng thái LCD
                MOV     R0, # CNTPORT       ; Nạp địa chỉ thanh ghi 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 các chân RD và RS

                MOV     R0, APORT   ; Nạp địa chỉ cổng A
READY:          MOVX    @R0         ; Đọc thanh ghi lệnh
                RLC     A           ; Chuyển D7 (cờ bận) vào bit nhớ carry
                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 các chân dữ
liệu của LCD
                MOV     R0, #BPORT  ; Nạp địa chỉ cổng B
                MOV     A, #00000101B       ; Đặt RS=1, R/W=0, E=1 cho xung
thấp lên cao
                MOVX    @R0, A              ; Kích hoạt RS, R/W, E của LCD
                NOP                 ; Tạo độ rộng xung của chân E
                NOP
                MOV     A, #00000001B       ; Đặt RS=1, R/W=0, E=0 cho xung
cao xuống thấp
```

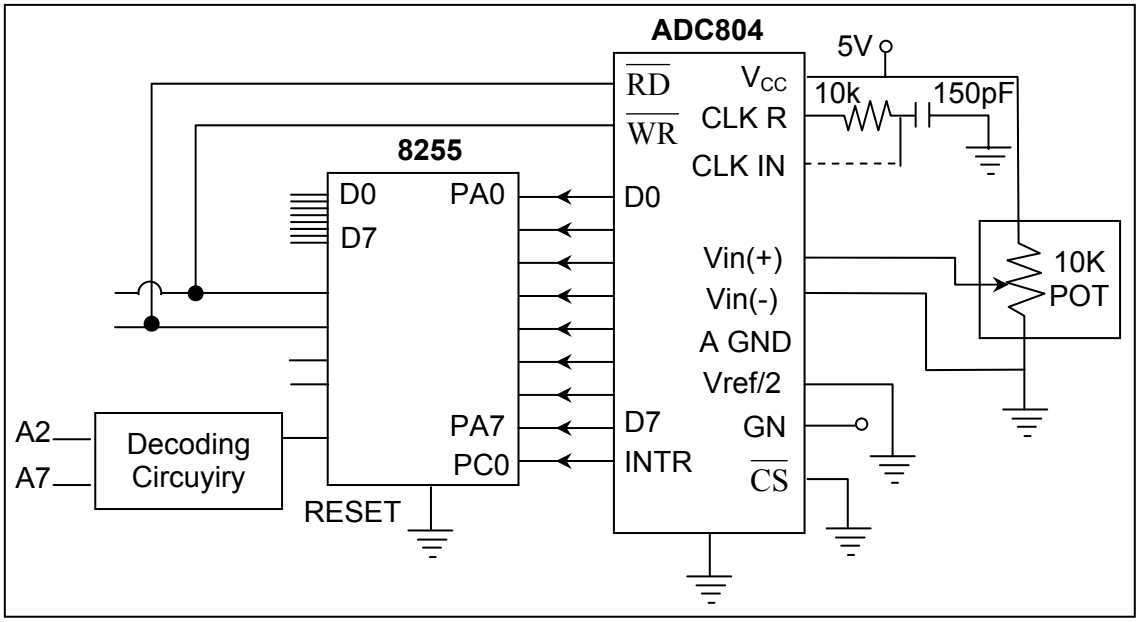
```
                MOVX        @R0, A              ; Chốt thông tin ở chân dữ liệu
LCD
                RET
```

15.2.3 Nối ghép ADC tới 8255.

Các bộ ADC đã được trình bày ở chương 12. Dưới đây một chương trình chỉ một bộ ADC được nối tới 8255 theo sơ đồ cho trên hình 115.11.

```
                MOV         A, #80H             ; Từ điều khiển với 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 địa chỉ cổng C để xem ADC
đã sẵn sàng chưa
                ANL         A,, #00000001B      ; Che tất cả các bít cổng C để xem
ADC đã sẵn sàng chưa
                JNZ         BACK        ; Giữ hiển thị PC0 che EOC
                                        ; Kết thúc hội thoại và bây giờ nhận dữ
liệu của ADC
                MOV         R1, #APORT  ; Nạp địa chỉ PA
                MOVX        A, @R1              ; A = đầu vào dữ liệu tương tự
```

Cho đến đây ta đã được trao đổi chế độ vào/ ra đơn giản của 8255 và trình bày nhiều ví dụ về nó. Ta xét tiếp các chế độ khác.

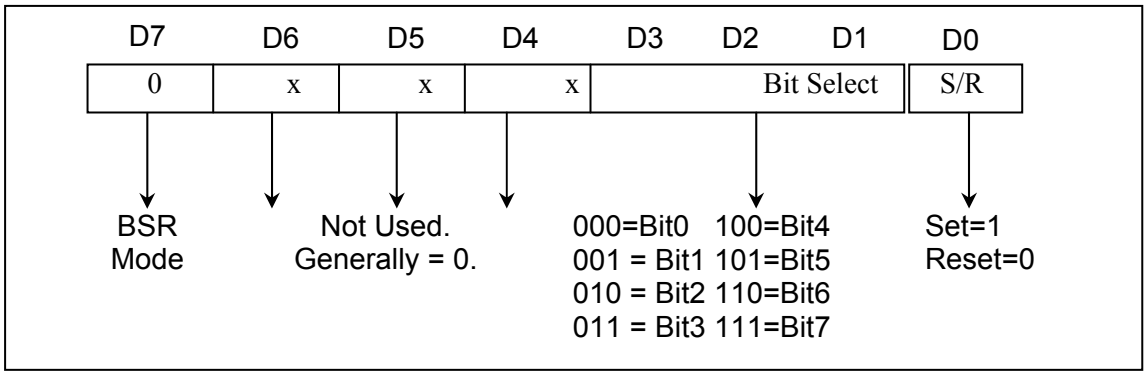


Hình 15.11: Nối ghép ADC 804 với 8255.

15.3 Các chế độ khác của 8255.

15.3.1 Chế độ thiết lập/ xoá bit BSR.

Một đặc tính duy nhất của cổng C là các bit có thể được điều khiển riêng rẽ. Chế độ BSR cho phép ta thiết lập các bit PC0 - PC7 lên cao xuống thấp như được chỉ ra trên hình 15.12. Ví dụ 15.6 và 15.7 trình bày cách sử dụng chế độ này như thế nào?

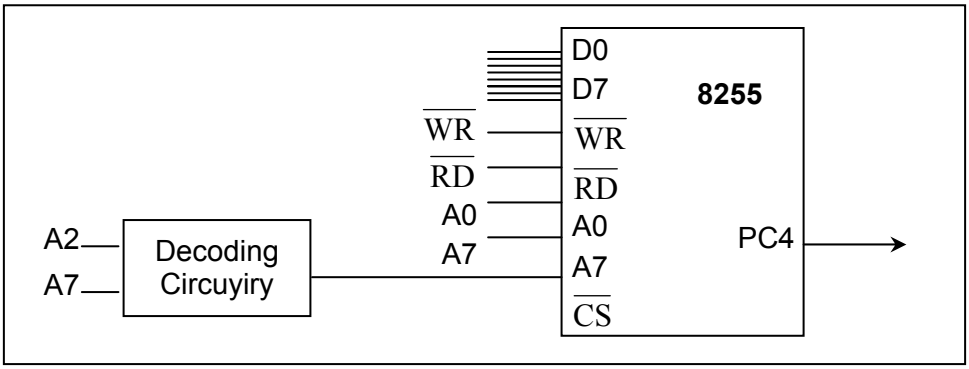


Hình 15.12: Từ điều khiển của chế độ BSR.

Ví dụ 15.6:

Hãy lập trình PCA của 8255 ở chế độ BSR thì bit D7 của từ điều khiển phải ở mức thấp. Để PC4 ở mức cao, ta cần một từ điều khiển là "0xxx1001" và ở mức thấp ta cần "0xxx1000". Các bít được đánh dấu x là ta không cần quan tâm và thường chúng 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: Cấu hình cho ví dụ 15.6 và 15.7.

Ví dụ 15.7:

Hãy lập trình 8255 theo sơ đồ 15.13 để:

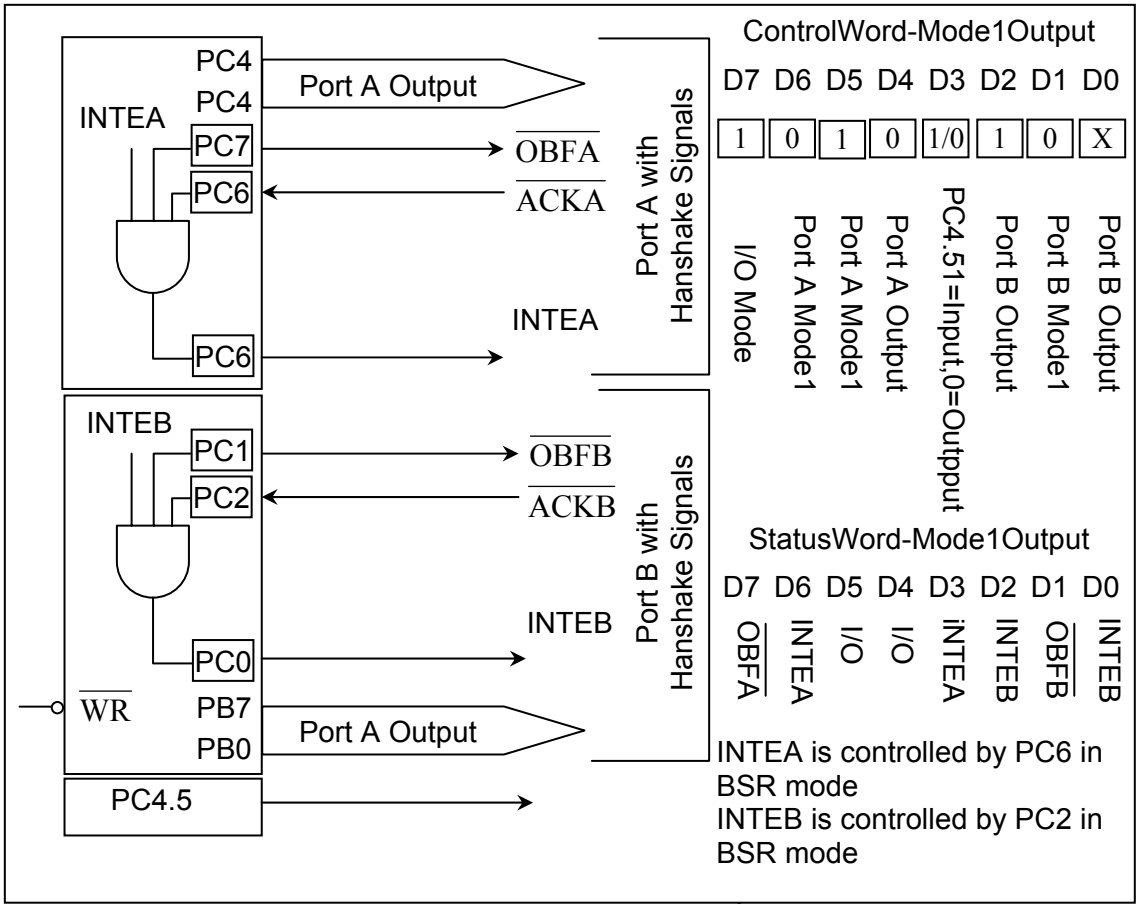
a) Đặt PC2 lên cao

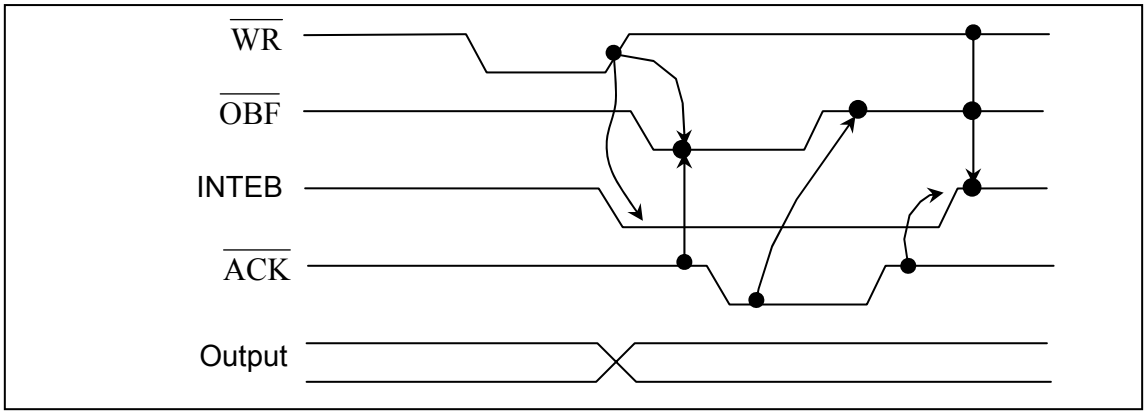
b) Sử dụng PC6 để tạo xung vuông liên tục với 66% độ đầy xung.

Lời 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 thanh ghi 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 15.15: Biểu đồ định thời của 8255 ở chế độ 1.

15.3.2 8255 ở chế độ 1: Vào/ ra với khả năng này bắt tay.

Một trong những đặc điểm mạnh nhất của 8255 là khả năng bắt tay với các thiết bị khác. Khả năng bắt tay là một quá trình truyền thông qua lại của hai thiết bị thông minh. Ví dụ về một thiết bị có các tín hiệu bắt tay là máy in. Dưới đây ta trình bày các tín hiệu bắt tay của 8255 với máy in.

Chế độ 1: Xuất dữ liệu ra với các tín hiệu bắt tay.

Như trình bày trên hình 15.14 thì cổng A và B có thể được sử dụng như các cổng đầu ra để gửi dữ liệu tới một thiết bị với các tín hiệu bắt tay. Các tín hiệu bắt tay cho cả hai cổng A và B được cấp bởi các bit của cổng C. Hình 15.15 biểu đồ định thời của 8255.

Dưới đây là các phần giải thích về các tín hiệu bắt tay và tính hợp lý của chúng đối với cổng A, còn cổng B thì hoàn toàn tương tự.

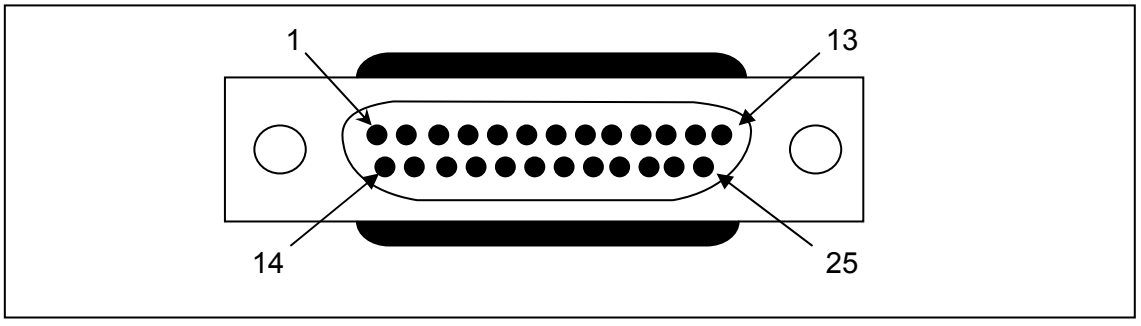
Tín hiệu $\overline{\text{OBFa}}$***:*** Đây là tín hiệu bộ đệm đầu ra đầy của cổng A được tích cực mức thấp đi ra từ chân PC7 để báo rằng CPU đã ghi 1 byte dữ liệu tới cổng A. Tín hiệu này phải được nối tới chân STROBE của thiết bị thu nhận dữ liệu (chẳng hạn như máy in) để báo rằng nó bây giờ đã có thể đọc một byte dữ liệu từ chốt cổng.

Tín hiệu $\overline{\text{ACKa}}$***:*** Đây là tín hiệu chấp nhận do cổng A có mức tích cực mức thấp được nhân tại chân PC6 của 8255. Qua tín hiệu $\overline{\text{ACKa}}$ thì 8255 biết rằng tín hiệu tại cổng A đã được thiết bị thu nhận lấy đi. Khi thiết bị nhận lấy dữ liệu đi từ cổng A nó báo 8255 qua tín hiệu $\overline{\text{ACKa}}$. Lúc này 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...

Tín hiệu INTRa: Đây là tín hiệu yêu cầu ngắt của cổng A có mức tích cực cao đi ra từ chân PC3 của 8255. Tín hiệu $\overline{\text{ACK}}$ là tín hiệu có độ dài hạn chế. Khi nó xuống thấp (tích cực) thì nó làm cho $\overline{\text{OBFa}}$ không tích cực, nó ở mức thấp một thời gian ngắn và sau đó trở nên cao (không tích cực). Sườn lên của $\overline{\text{ACK}}$ kích hoạt INTRa lên cao. Tín hiệu cao này trên chân INTRa có thể được dùng để gây chú ý của CPU. CPU được thông báo qua tín hiệu INTRa rằng máy in đã nhận byte cuối cùng và nó sẵn sàng để nhận byte dữ liệu khác. INTRa ngắt CPU ngừng mọi thứ đang làm và ép nó gửi byte kế tiếp tới cổng A để in. Điều quan trọng là chú ý rằng INTRa được bật lên 1 chỉ khi nếu INTRa, $\overline{\text{OBFa}}$ và $\overline{\text{ACKa}}$ đều ở mức cao. Nó được xoá về không khi CPU ghi một byte tới cổng A.

Tín hiệu INTEa: Đây là tín hiệu cho phép ngắt cổng A 8255 có thể cấm INTRa để ngăn nó không được ngắt CPU. Đây là chức năng của tín hiệu INTEa. Nó là một mạch lật Flip - Flop bên trong thiết kế để che (cấm) INTRa. Tín hiệu INTRa có thể được bật lên hoặc bị xoá qua cổng C trong chế độ BSR vì INTEa là Flip - Plop được điều khiển bởi PC6.

Từ trạng thái: 8255 cho phép hiển thị trạng thái của các tín hiệu INTR, OBF và INTE cho cả hai cổng A và B. Điều này được thực hiện bằng cách đọc cổng C vào thanh ghi tổng và kiểm tra các bit. Đặc điểm này cho phép thực thi thăm dò thay cho ngắt phần cứng.



Hình 15.16: Đầu cắm DB-25.

(hình 15.17 mờ quá không vẽ được)

Hình 15.17: Đầu cáp của máy in Centronics.

Bảng 15.2: Các chân tín hiệu của máy in Centronics.

Chân số	Mô tả	Chân số	Mô tả
1	STROBE	11	Bận (busy)
2	Dữ liệu D0	12	Hết giấy (out of paper)
3	Dữ liệu D1	13	Chọn (select)
4	Dữ liệu D2	14	Tự nạp ($\overline{\text{Autofeed}}$)
5	Dữ liệu D3	15	Lỗi ($\overline{\text{Error}}$)
6	Dữ liệu D4	16	Khởi tạo máy in
7	Dữ liệu D5	17	Chọn đầu vào ($\overline{\text{Select input}}$)
8	Dữ liệu D6	18-25	Đất (ground)
9	Dữ liệu D7		
10	$\overline{\text{ACK}}$ (chấp nhận)		

Các bước truyền thông có bắt tay giữa máy in và 8255.

Một byte dữ liệu được gửi đến bus dữ liệu máy in.

Máy in được báo có 1 byte dữ liệu cần được in bằng cách kích hoạt tín hiệu đầu vào STROBE của nó.

Khi máy nhận được dữ liệu nó báo bên gửi bằng cách kích hoạt tín hiệu đầu ra được gọi là $\overline{\text{ACK}}$ (chấp nhận).

Tín hiệu $\overline{\text{ACK}}$ khởi tạo quá trình cấp một byte khác đến máy in.

Như ta đã thấy từ các bước trên thì chỉ khi một byte dữ liệu tới máy in là không đủ. Máy in phải được thông báo về sự hiện diện của dữ liệu. Khi dữ liệu được gửi thì máy in có thể bận hoặc hết giấy, do vậy máy in phải được báo cho bên gửi khi nào nó nhận và lấy được dữ liệu của nó. Hình 15.16 và 15.17 trình các ổ cắm DB25 và đầu cáp của máy in Centronics tương ứng.