TRƯỜNG CAO ĐẲNG NGHỀ KTCN VIỆT NAM – HÀN QUỐC
KHOA ĐIỆN TỬ

GIÁO TRÌNH

VI ĐIỀU KHIỂN HỌ 8051

Biên soạn: Vương Đạo Nhân

Năm 2011

LỜI NÓI ĐẦU

Các tiến bộ khoa học ngày nay, nhất là các tiến bộ trong lĩnh vực điện tử và tự động hóa có ảnh hưởng mạnh mẽ đến sự phát triển của nền kinh tế, xã hội và các ngành khoa học kỹ thuật khác. Sự ra đời và ứng dụng rộng rãi các bộ vi xử lý và vi điều khiển là một cuộc cách mạng lớn trong điều khiển các thiết bị công nghiệp và dân dụng. Các thiết bị công nghiệp và dân dụng ngày càng có xu hướng ứng dụng vi điều khiển để tăng tính năng tác dụng, đáp ứng ngày càng cao yêu cầu của người sử dụng. Vi điều khiển được ứng dụng rộng rãi trong các dây chuyền sản xuất, các loại robot, các thiết bị gia dụng như điều hòa nhiệt độ, máy giặt, lò vi sóng …Vi điều khiển có nhiều họ và nhiều công ty khác nhau thiết kế sản xuất do đó phong phú và đa dạng về chủng loại. Đối với người học vi điều khiển tốt nhất là bắt đầu bằng họ 8051 bởi vì họ vi điều khiển này rất phổ biến, giá thành rẻ và phù hợp với các ứng dụng vừa và nhỏ. Khi đã có nền tảng là vi điều khiển họ 8051 thì việc tiếp cận ứng dụng các vi điều khiển khác không mấy khó khăn.

Xuất phát từ yêu cầu thực tế trong việc ứng dụng vi điều khiển và chương trình giảng dạy môn vi điều khiển cho nghề Điện tử công nghiệp do Tổng cục dạy nghề ban hành, tác giả đã hoàn thành cuốn giáo trình ***"Vi điều khiển họ 8051".*** Cuốn giáo trình này phù hợp với người học trung cấp, cao đẳng nghề và những người giảng dạy, thiết kế ứng dụng họ vi điều khiển 8051. Nội dung cuốn giáo trình gồm 7 chương:

Chương 1: Sơ lược về lịch sử và hướng phát triển của vi điều khiển. Chương này người đọc có cái nhìn tổng quan về vi điều khiển.

Chương 2: Cấu trúc họ vi điều khiển 8051. Đề cập chi tiết cấu trúc vi điều khiển họ 8051 bao gồm các cổng, bộ nhớ.

Chương 3: Tập lệnh 8051. Tổng hợp đầy củ các lệnh của vi điều khiển họ 8051 được sắp xếp theo từng nhóm lệnh.

Chương 4*:* Bộ định thời/đếm. Trong chương này nêu tác dụng các thanh ghi liên quan, cách truy xuất các thanh ghi và thiết lập các chế độ hoạt động cho bộ định thời/đếm.

Chương 5: Cổng nối tiếp. Đề cập phương pháp truyền tin qua cổng nối tiếp, cách truy xuất các thanh ghi, thiết lập các tốc độ truyền thông tin.

Chương 6: Hoạt động ngắt. Khái niệm về ngắt, các nguồn ngắt, thiết lập các chế độ cho hoạt động ngắt.

Chương 7: Trình dịch Keil C cho họ vi điều khiển 8051. Chương này hướng dẫn chi tiết cách cài đặt, sử dụng trình biên dịch Keil C là trình biên dịch C cho họ vi điều khiển 8051 hiện nay được nhiều người dùng. Cuối chương có các bài tập ứng dụng được viết bằng ngôn ngữ C sử dụng trình biên dịch Keil C.

Chân thành cảm ơn Ban giám hiệu Trường CĐN KTCN Việt Nam – Hàn Quốc, các đồng nghiệp cùng gia đình đã tạo điều kiện và đóng góp ý kiến để tôi hoàn thành cuốn giáo trình này.

Mặc dù có nhiều cố gắng trong việc biên soạn cuốn giáo trình này tuy nhiên khó có thể tránh khỏi những sai sót. Rất mong nhận được sự góp ý của người đọc và các đồng nghiệp để cuốn giáo trình hoàn thiện hơn. Mọi góp ý xin gửi về Khoa điện tử Trường CĐN KTCN Việt Nam – Hàn Quốc, đường Hồ Tông Thốc - TP Vinh - Nghệ An hoặc email: vuongdaonhan@gmail.com.

Vinh, tháng 4 năm 2011

MỤC LỤC

Chương 1
SƠ LƯỢC VỀ LỊCH SỬ VÀ HƯỚNG PHÁT TRIỂN CỦA VI ĐIỀU KHIỂN

1.1. LỊCH SỬ PHÁT TRIỂN

Máy tính số là các mạch điện xử lý tín hiệu dạng số được điều khiển bởi chương trình, có thể làm những công việc mà con người mong muốn. Chương trình sẽ điều khiển các mạch điện số thực hiện di chuyển và xử lý dữ liệu (Data) bằng cách điều khiển các mạch logic số học, các bộ nhớ (Memory), các thiết bị vào/ra (I/O - Input/output). Cách thức các mạch điện logic của máy tính số kết hợp lại với nhau tạo thành các mạch logic số học, các vi mạch nhớ và các thiết bị vào/ra được gọi là cấu trúc.

Vi xử lý có cấu trúc giống như máy tính số và có thể xem nó là máy tính số vì cả hai đều hoạt động dưới sự điều khiển của chương trình.

Lịch sử phát triển của vi xử lý gắn liền với sự phát triển của các vi mạch điện tử vì vi xử lý là vi mạch điện tử chế tạo theo công nghệ LSI (Large Scale Integrated) cho đến VLSI (Very Large Scale Integrated).

Với sự khám phá ra transistor và phát triển của công nghệ chế tạo vi mạch SSI, MSI, máy tính vẫn còn là một nhóm gồm nhiều IC kết hợp lại với nhau, cho đến thập niên 70 với sự phát triển của công nghệ LSI, cấu trúc máy tính được rút gọn bởi các nhà thiết kế và được chế tạo thành một IC duy nhất được gọi là vi xử lý (Microprocessor).

Năm 1971 tập đoàn Intel đã giới thiệu 8080 là bộ vi xử lý thành công đầu tiên. Sau đó không lâu thì các hãng: Motorola, RCA, MOS Technology và Zilog đã giới thiệu các bộ vi xử lý tương tự: 6800, 1801, 6502 và Z80. Bản thân các mạch này tuy không có nhiều hiệu quả sử dụng nhưng khi là một phần của máy tính đơn board thì chúng trở thành trung tâm.

Vi xử lý là sự kết hợp của 2 kỹ thuật công nghệ quan trọng nhất đó là máy tính dùng kỹ thuật số (Digital Computer) và các mạch vi điện tử (Integrated Circuit).

Chức năng chính của vi xử lý là xử lý dữ liệu. Để làm được điều này vi xử lý phải có các mạch logic cho việc xử lý và điều khiển dữ liệu, các mạch logic điều khiển. Các mạch logic xử lý sẽ di chuyển dữ liệu từ nơi này sang nơi khác và thực hiện các phép toán trên dữ liệu, mạch logic điều khiển sẽ quyết định mạch điện nào cho việc xử lý dữ liệu. Vi xử lý thực hiện một lệnh với trình tự như sau: đọc lệnh từ bộ nhớ, tiếp theo mạch logic điều khiển sẽ giải mã lệnh để xem lệnh đó yêu cầu vi xử lý thực hiện công việc gì, sau đó vi xử lý sẽ thực hiện đúng công việc của lệnh đã yêu cầu, quá trình này được gọi là chu kỳ đọc và thực hiện lệnh.

Ngoài chức năng đọc và thực hiện lệnh, các mạch logic điều khiển còn điều khiển các mạch điện giao tiếp bên ngoài kết nối với vi xử lý. Vi xử lý cần phải có sự trợ giúp của các mạch điện bên ngoài. Các mạch điện dùng để lưu trữ lệnh để vi xử lý xử lý được gọi là bộ nhớ, các mạch điện giao tiếp để di chuyển dữ liệu từ bên ngoài vào bên trong vi xử lý và xuất dữ liệu từ bên trong vi xử lý ra ngoài được gọi là các thiết bị vào/ra hay các thiết bị ngoại vi hay nói cách khác các thiết bị ngoại vi là các mạch điện giúp người sử dụng giao tiếp với vi xử lý.

Vi xử lý kết hợp với các thiết bị khác tạo ra các máy tính có khả năng tính toán rất lớn như máy vi tính và có thể tạo ra các sản phẩm khác các máy điện thoại, các tổng đài điện thoại, các hệ thống điều khiển tự động...

Để kết nối các khối trên tạo thành một hệ thống vi xử lý đòi hỏi người thiết kế phải rất hiểu biết về tất cả các thành phần vi xử lý, bộ nhớ và các thiết bị ngoại vi. Hệ thống tạo ra khá phức tạp, chiếm nhiều không gian, mạch in và đòi hỏi người thiết kế, người sử dụng hiểu thật rõ về hệ thống. Vi xử lý thường xử lý dữ liệu theo byte hoặc word trong khi đó các đối tượng điều khiển trong công nghiệp thường điều khiển theo bit.

Chính vì sự phức tạp nên các nhà chế tạo đã tích hợp một ít bộ nhớ và một số các thiết bị ngoại vi cùng với vi xử lý trên một chip gọi là vi điều khiển (MCU – Micro Controller Unit).

Năm 1976 hãng Intel giới thiệu bộ vi điều khiển 8748, vi điều khiển này tích hợp hơn 17.000 transistor bao gồm một CPU, 1kbyte bộ nhớ EPROM, 64byte RAM, một bộ định thời/đếm 8bit và 27 chân vào ra. Vi điều khiển này mở đầu cho họ vi điều khiển MCS-48.

Sau 8748 các bộ vi điều khiển mới liên tục được các hãng các hãng sản xuất như Intel, Atmel, Siemen, Microchip … giới thiệu với tính năng ngày càng mạnh.

Vi điều khiển có thể coi là một máy tính được tích hợp trên một chíp, nó có thể hoạt động với một vài linh kiện phụ trợ bên ngoài, nó thường được sử dụng trong các hệ thống nhúng và trong công nghiệp.

Vi điều khiển, thực chất, là một hệ thống bao gồm một vi xử lý có hiệu suất đủ dùng và giá thành thấp (khác với các bộ vi xử lý đa năng dùng trong máy tính) kết hợp với các khối ngoại vi như bộ nhớ, các mô đun vào/ra, các mô đun biến đổi số sang tương tự và tương tự sang số... Ở máy tính thì các mô đun thường được xây dựng bởi các chíp và mạch ngoài.

Khi vi điều khiển ra đời đã mang lại sự tiện lợi là dễ dàng sử dụng trong điều khiển công nghiệp, việc sử dụng vi điều khiển không đòi hỏi người sử dụng phải hiểu biết một lượng kiến thức quá nhiều như người sử dụng vi xử lý – dĩ nhiên người sử dụng hiểu biết càng nhiều thì càng tốt nhưng đối với người bắt đầu thì việc sử dụng vi

xử lý là điều rất phức tạp trong khi đó mong muốn của người sử dụng là đi nhanh vào ứng dụng.

1.2. ĐIỂM KHÁC BIỆT GIỮA VI ĐIỀU KHIỂN VÀ VI XỬ LÝ

Vi điều khiển được phát triển trên cơ sở thừa hưởng công nghệ vi xử lý và giữa vi điều khiển với vi xử lý có những khác biệt nhất định:

1.2.1. Về cấu trúc

Vi xử lý là một CPU trên một chip còn vi điều khiển là một chip chứa CPU, bộ nhớ, mạch vào/ra và các mạch đặc biệt khác như bộ định thời/đếm, mạch biến đổi AD… Như vậy về cấu trúc thì vi điều khiển là một hệ vi xử lý thu nhỏ.

1.2.2. Về ứng dụng

Các bộ vi xử lý thường dùng làm CPU trong các máy tính còn vi điều khiển thường được dùng trong các ứng dụng hướng điều khiển.

1.2.3. Về tập lệnh

Tập lênh cho vi xử lý là những lệnh mang tính tổng quát nên chúng được dùng với nhiều kiểu định địa chỉ, cho phép thao tác với lượng dữ liệu lớn. Ngược lại tập lênh vi điều khiển chủ yếu là những lện vào/ra đơn giản và các lệnh xử lý bit.

1.3. ĐIỂM KHÁC BIỆT GIỮA VI ĐIỀU KHIỂN VÀ MÁY TÍNH

Như trên đã nói vi điều khiển có thể coi là máy tính thu nhỏ tuy nhiên vi điều khiển và máy tính có những khác biệt ở chỗ:

1.3.1. Về ứng dụng

Máy tính được thiết kế để có thể thực chức năng là một thiết bị lập trình đa dụng. Vi điều khiển được thiết kế cho các ứng dụng hướng điều khiển, chương trình mà nó thực hiện (chương trình điều khiển) được các hãng phát triển sản phẩm viết ra và nạp vào trong bộ nhớ chương trình, nội dung của chương trình hầu như không thay đổi trong suốt thời gian tồn tại của sản phẩm.

1.3.2. Về bộ nhớ

Máy tính là thiết bị đa dụng nên các chương trình ứng dụng thường được lưu trên các thiết bị nhớ ngoài như đĩa cứng, đĩa quang… Khi cần thực thi chương trình được nạp vào bộ nhớ RAM của máy tính và vi xử lý sẽ đọc mã lệnh từ RAM và giải mã lệnh để thực thi. Như vậy với máy tính thì RAM là bộ nhớ chương trình còn ROM trong máy tính thì lưu cấu hình của máy và các vào/ra cơ bản. Chính vì vậy mà trong máy tính dung lượng của RAM lớn hơn ROM nhiều lần. Ngược lại thì vi điều khiển chương trình được lưu trong ROM vì chúng là chương trình điều khiển ứng dụng hầu như không thay đổi nội dung còn RAM thì được dùng để chứa dữ liệu tạm thời cho

chương trình như trạng thái các chân vào/ra, nội dung các biến được khai bảo trong chương trình do đó ở vi điều khiển thì dung lượng ROM lớn hơn RAM nhiều lần.

1.3.3. Về hiệu năng

Do phải tích hợp nhiều thành phần trên một chip duy nhất nên việc nâng cao hiệu năng của vi điều khiển là khó thực hiện hơn máy tính. Do đó các ứng dụng đòi hỏi hiệu năng cao thì vi điều khiển không đáp ứng được như xử lý ảnh từ camera, quản trị cơ sở dữ liệu… Khi cần thực hiện những ứng dụng này thì giải pháp được lựa chọn là máy tính và thường là máy tính nhúng (Bộ vi xử lý, các ROM, RAM, cổng vào/ra đầu cắm thẻ nhớ, ổ đĩa ... được tích hợp trên một mạch in; hệ điều hành, chương trình điều khiển ... được cài đặt trên thẻ nhớ).

1.4. GIỚI THIỆU MỘT SỐ HỌ VI ĐIỀU KHIỂN THÔNG DỤNG

1.4.1. Vi điều khiển của hãng Atmel

Atmel là một hãng cung cấp vi điều khiển lớn với các dòng sản phẩm tiêu biểu sau:

- Dòng vi điều khiển dựa trên kiến trúc 8051 của Intel như: 83xx, 87xx, 89xx...
- Dòng vi điều khiển AT91CAP như AT91CAP7S250A, AT91CAP7S450A ... với tần số xung nhịp từ 80MHz đến 200MHz, 2 đến 4 kênh PWM, 10 kênh ADC 10 bit, ghép nối được với các module SDRAM ngoài.
- Dòng vi điều khiển AT91SAM 32 bit ARM với bộ nhớ chương trình lên đến 2MB, tần số hoạt động lên đến 240MHz.
- Dòng AVR 8 bit kiến trúc RISC như AT90PWM1, ATmega128, ATmega16, ATmega32

1.4.2. Vi điều khiển của Microchip

- Dòng 8 bit như PIC10, PIC 12, PIC16, PIC18 với bộ nhớ kiểu Flash, OPT, ROM có dung lượng từ 0.5kbyte đến 256kbyte hoặc ROMless.
- Dòng 16 bit như PIC24F, PIC24H.
- Dòng vi điều khiển 16 bit có hỗ trợ xử lý tín hiệu số như dsPIC30Fxxxx, dsPIC33FJxxxx.

1.4.3. Vi điều khiển của Motorola

Motorola sản xuất dòng vi điều khiển 68xx như 6801, 6805, 6809, 6811… Một sản phẩm tiêu biểu của Motorola đó là vi điều khiển 68HC11, đây là một bộ vi điều khiển 8 bit, 16 bit địa chỉ, tập lệnh tương thích với các phiên bản trước như 6801, 6805, 6809, 6811 có tích hợp bộ biến đổi ADC, bộ tạo xung PWM, cổng truyền thông đồng bộ/không đồng bộ RS232, SPI.

1.5. LĨNH VỰC ỨNG DỤNG VÀ HƯỚNG PHÁT TRIỂN

Vi điều khiển ra đời tạo sự thuận lợi cho người sử dụng trong việc điều khiển hướng đối tượng đặc biệt là điều khiển bit và có thể không cần thêm ngoại vi. Ứng dụng chủ yếu của vi điều khiển là điều khiển hướng đối tượng và hệ thống nhúng như trong điều hòa, máy giặt, điều khiển nhiệt độ, điều khiển động cơ…

Hiện nay các hãng sản xuất vi điều khiển vẫn tập trung nghiên cứu phát triển các dòng vi điều khiển thế hệ mới có tích hợp nhiều tính năng, nhiều ngoại vi, tăng tốc độ xử lý, tăng dung lượng bộ nhớ và chống nhiễu tốt để ứng trong công nghiệp nơi có môi trường làm việc khắc nghiệt và yêu cầu ổn định cao. Một số hãng còn tích hợp tính năng xử lý tín hiệu số trong vi điều khiển tiêu biểu là dòng dsPIC của Microchip.

Việc phát triển vi điều khiển kéo theo nhiều ngành công nghiệp điện điện tử phát triển như điện tử dân dụng, điện tử công nghiệp, điện tử y sinh…

1.6. ƯU VÀ NHƯỢC ĐIỂM KHI THIẾT KẾ HỆ THỐNG VỚI VI ĐIỀU KHIỂN

Trước đây chúng ta đã quen với việc thiết kế các hệ thống số với hàng chục thậm chí hàng trăm vi mạch số. Việc chuyển sang thiết kế sử dụng vi điều khiển cho phép thực hiện những hệ thống tương đương với một vài linh kiện phụ trợ làm cho thời gian phát triển ngắn hơn, độ tin cậy của hệ thống cao hơn, công suất tiêu thụ thấp hơn, giá thành rẻ hơn.

Tuy nhiên phát sinh vấn đề mới đó là tốc độ. Các giải pháp dựa trên vi điều khiển không thể cho tốc độ xử lý tín hiệu nhanh như các giải pháp sử dụng các linh kiện số rời rạc bởi hệ thống dựa trên vi điều khiển thường xuyên phải thực hiện chu trình "Đọc – Giải mã – Thi hành lệnh" trong khi đó với các hệ thống dựa trên các linh kiện rời rạc thì tín hiệu chạy trực tiếp từ đầu ra của phần tử này tới đầu vào của phần tử kia với thời gian không đáng kể.

Với hệ thống dựa trên vi điều khiển muốn cải thiện tốc độ, ngoài việc tối ưu hóa chương trình điều khiển thì phải cải tiến kiến trúc và tốc độ của vi điều khiển, kết quả là giá thành hệ thống tăng mạnh. Tuy nhiên với đa số các ứng dụng thì tốc độ không phải là yêu cầu số một nên trong thực tế các thiết kế dựa trên vi điều khiển ngày càng trở nên phổ biến.

Chương 2

CẤU TRÚC HỌ VI ĐIỀU KHIỂN 8051

2.1. TỔNG QUAN

2.1.1. Tóm tắt lịch sử họ vi điều khiển 8051

Năm 1981 hãng Intel giới thiệu bộ vi điều khiển 8051. Bộ vi điều khiển này có 128 byte RAM, 4 kbyte ROM, hai bộ định thời, một cổng nối tiếp và bốn cổng vào/ra song song độ rộng 8 bit (bảng 2-1). 8051 là bộ vi điều khiển 8 bit có nghĩa là CPU chỉ có thể làm việc được 8 bit dữ liệu tại một thời điểm, dữ liệu lớn hơn 8 bit thì được chia thành các dữ liệu 8 bit để xử lý.

Vi điều khiển 8051 đã trở nên phổ biến sau khi Intel cho phép các nhà sản xuất khác sản xuất và bán các dạng biến thể của 8051. Điều này dẫn đến sự ra đời của nhiều phiên bản của 8051 với các tốc độ khác nhau và dung lượng ROM trên chip khác nhau nhưng tất cả các lệnh đều tương thích với 8051 ban đầu. Như vậy nếu ta viết chương trình cho một phiên bản nào của 8051 thì cũng chạy được với mọi phiên bản khác cùng cấu hình mà không phụ thuộc vào hãng sản xuất.

Bảng 2-1. Các thông số cơ bản của vi điều khiển 8051 đầu tiên

Thông số	Số lượng
ROM	4 kbyte
RAM	128 byte
Bộ đếm/định thời	2
Chân vào/ra	32
Cổng nối tiếp	1
Nguồn ngắt	5

2.1.2. Các thành viên khác của họ 8051

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

Bảng 2-2. So sánh các thông số của các thành viên họ 8051

Thông số	8051	8052	8031
ROM	4 kbyte	8 kbyte	Không có
RAM	128 byte	256 byte	128 byte
Bộ định thời/đếm	2	3	2
Chân vào/ra	32	32	32
Cổng nối tiếp	1	1	1
Nguồn ngắt	5	6	1

Bộ vi điều khiển 8052 là một thành viên họ 8051, 8052 có các đặc tính chuẩn của 8051 ngoài ra có thêm 128 byte RAM, 4 kbyte ROM và một bộ định thời/đếm nữa (bảng 2-2). Như vậy 8052 là mở rộng của 8051 nên các chương trình viết cho 8051 đều chạy được trên 8052 nhưng điều ngược lại thì không phải lúc nào cũng đúng.

Một thành viên khác nữa của 8051 là chip 8031. Chip này thường được coi như là 8051 không có ROM. Để sử dụng chip này ta phải kết nối ROM ngoài với nó, ROM ngoài phải chứa chương trình mà 8031 sẽ nạp và thực hiện, ROM ngoài được gắn thêm trên hệ thống 8031 thì có thể lên đến 64k kbyte. Khi sử dụng ROM ngoài thì chỉ còn lại 2 cổng trên 8031 sử dụng cho mục đích vào/ra, để tăng số lượng cổng vào ra thì sử dụng vi mạch PPI 8255 hoặc một số vi mạch cổng khác.

2.1.3. Bộ vi điều khiển AT8951 của hãng Atmel

AT8951 là phiên bản 8051 có ROM trên chip ở dạng bộ nhớ Flash, phiên bản này là lý tưởng trong công việc nghiên cứu lập trình vì bộ nhớ này xóa nạp nhanh chỉ trong vài giây.

Bảng 2-3. Các phiên bản 8051 của hãng Atmel

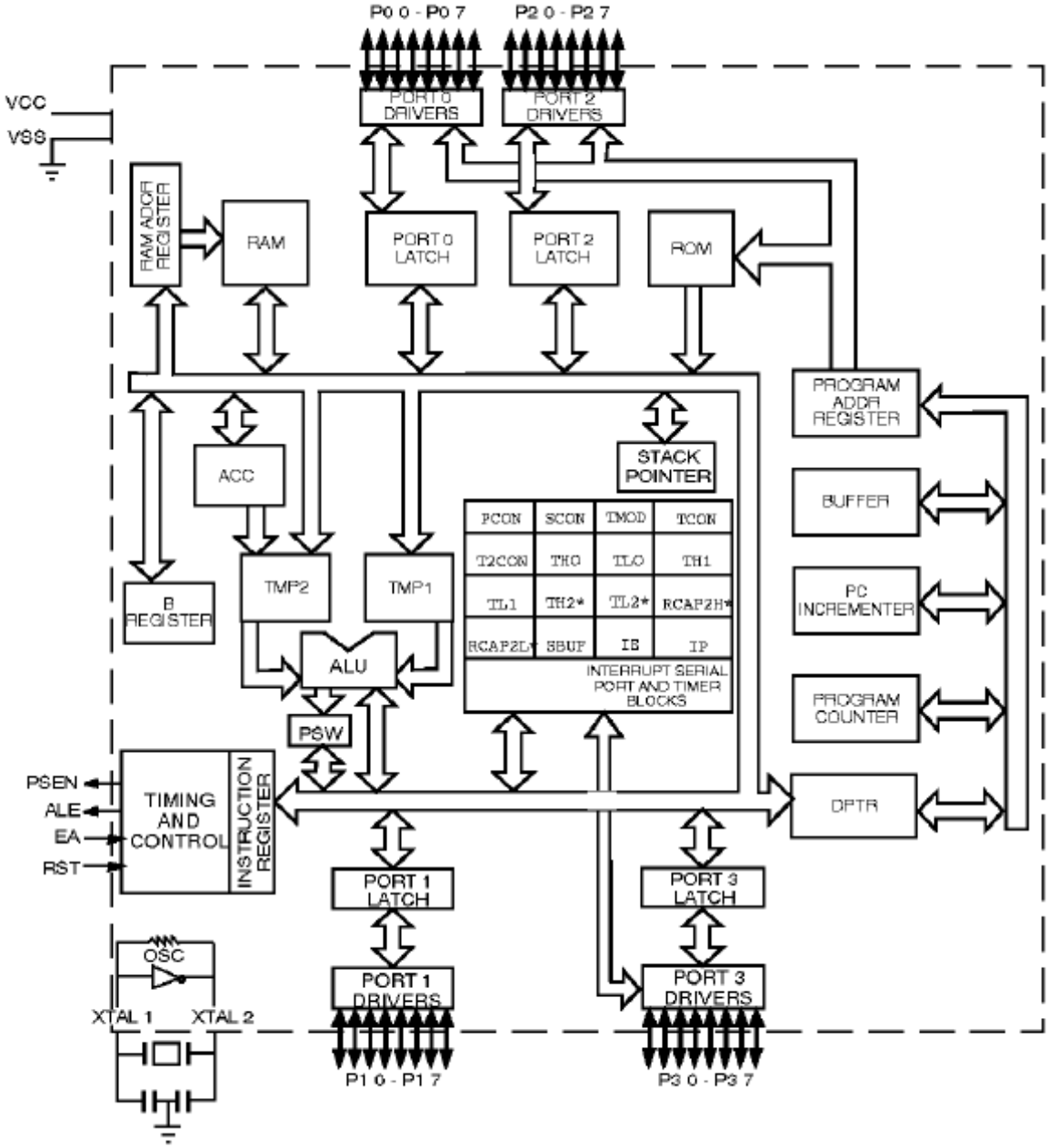
Ký hiệu	ROM	RAM	Chân I/O	Định thời /đếm	Ngắt	Vcc	Số chân
AT89C51	4 kbyte	128 byte	32	2	5	5V	40
AT89LV51	4 kbyte	128 byte	32	2	5	3V	40
AT89C1051	1 kbyte	64 byte	15	1	6	3V	20
AT89C2051	2 kbyte	128 byte	16	2	6	3V	20
AT89C52	8 kbyte	128 byte	32	3	6	5V	40
AT89LV52	8 kbyte	128 byte	32	3	6	3V	40

Hiện nay những người làm việc với vi điều khiển họ 8051 thường sử dụng AT89C51 và AT89C52 vì trên thị trường hai chip này rất phổ biến và giá thành rẻ. Vì vậy trong khuôn khổ tài liệu này tác giả đề cập đến các vấn đề đều ở trên chip AT89C51 và AT89C52.

2.2. SƠ ĐỒ KHỐI CỦA VI ĐIỀU KHIỂN AT89C51

Các khối cấu trúc trong vi điều khiển 8051 được thể hiện trong hình 2-1 bao gồm các khối chính như sau:

- Khối ALU đi kèm với các thanh ghi TMP1, TMP2 và thanh ghi trạng thái PSW.
- Bộ điều khiển logic (Timing and Control).
- Vùng nhớ RAM nội và vùng nhớ Flash ROM lưu trữ chương trình.



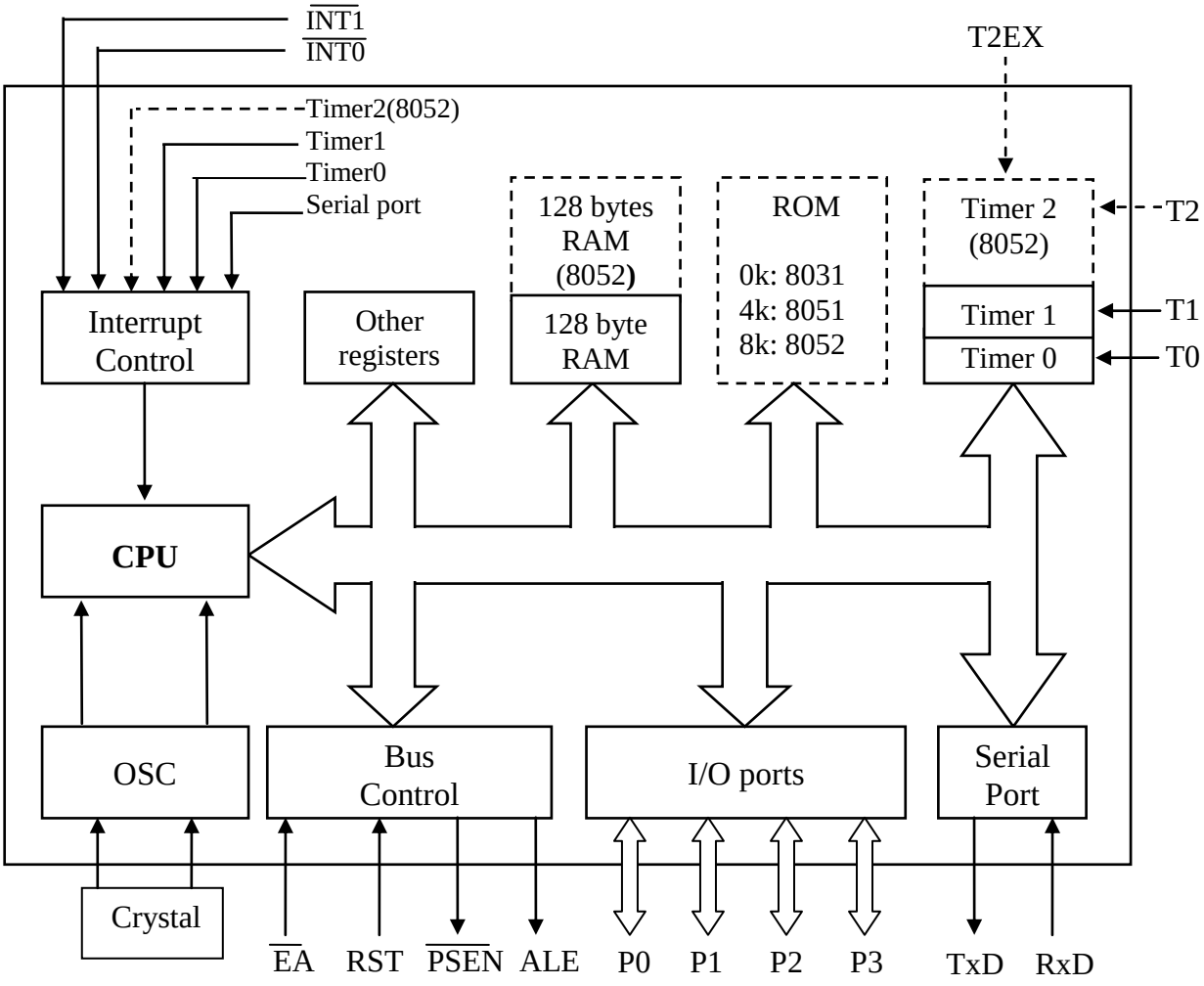
Ghi chú: Dấu (*) chỉ có ở AT89C52

Hình 2-1. Khối cấu trúc bên trong của vi điều khiển AT89C51

- Mạch tạo dao động nội kết hợp với tụ thạch anh bên ngoài để tạo dao động.
- Khối xử lý ngắt, truyền dữ liệu, khối Timer/Counter.
- Thanh ghi A, B, DPTR và 4 port: port0, port1, port2, port3 có chốt và đệm.
- Thanh ghi bộ đếm chương trình PC (Program Counter).
- Con trỏ dữ liệu DPTR (Data Pointer).
- Thanh ghi con trỏ ngăn xếp SP (Stack Pointer).
- Thanh ghi lệnh IR (Instruction Register).

- Ngoài ra còn có 1 số các thanh ghi hỗ trợ để quản lý địa chỉ bộ nhớ RAM nội cũng như các thanh ghi quản lý địa chỉ truy xuất bộ nhớ bên ngoài.

Giao tiếp các khối bên trong và các chân đưa ra ngoài được thể hiện trong hình 2-2.

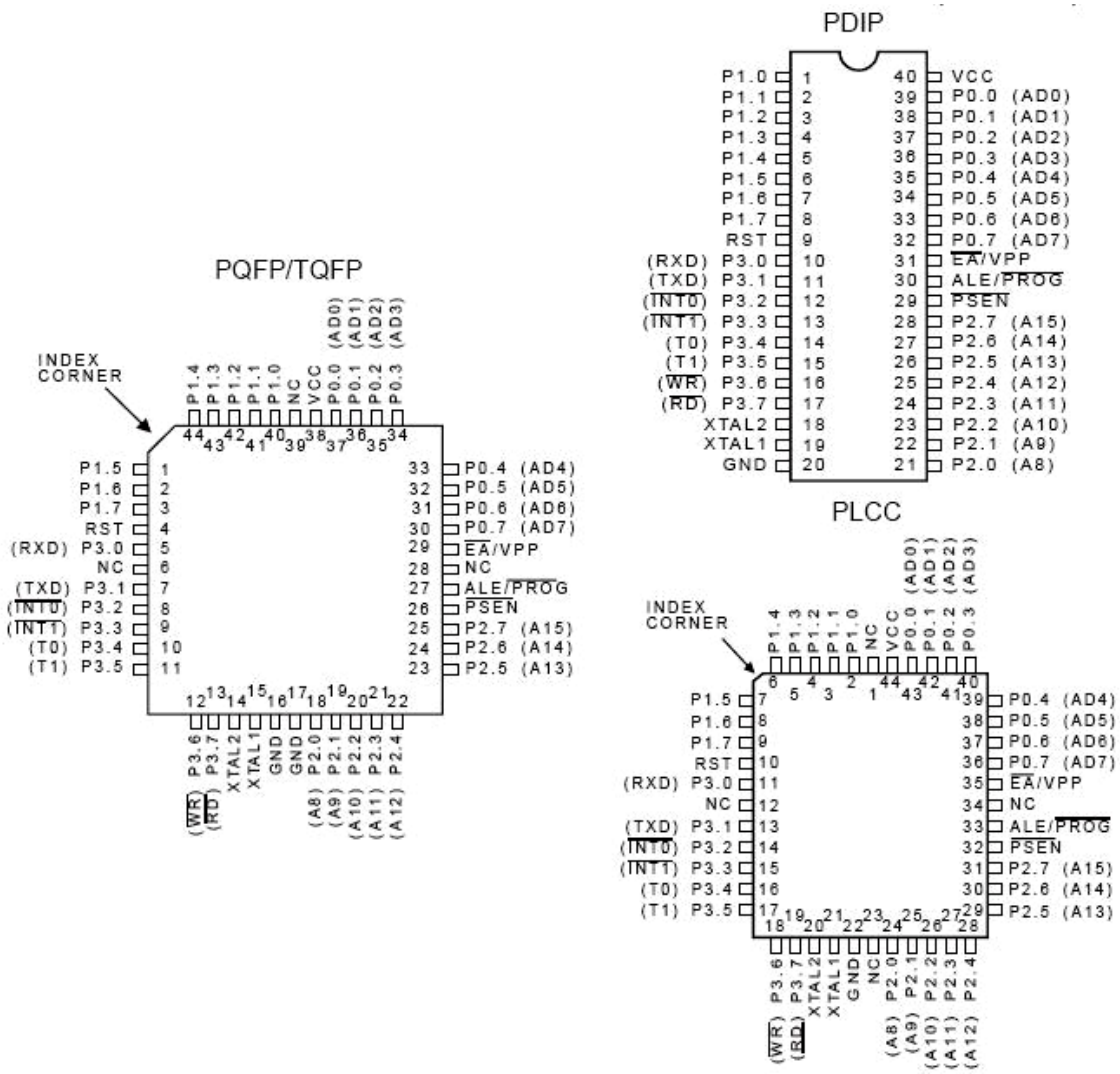


Hình 2-2. Sơ giao tiếp các khối trong vi điều khiển họ 8051

Chức năng của từng khối sẽ được đề cập trong các phần cụ thể trong các chương tiếp theo.

2.3. SƠ ĐỒ CHÂN VI ĐIỀU KHIỂN AT89C51

Vi điều khiển AT89C51 có tất cả 40 chân chức năng (hình 2-3). Trong đó có 24 chân có tác dụng kép (có nghĩa là 1 chân có 2 chức năng), mỗi chân có thể hoạt động như đường xuất nhập điều khiển IO (Input Output) hoặc là thành phần của các bus dữ liệu và bus địa chỉ để tải địa chỉ và dữ liệu khi giao tiếp với bộ nhớ ngoài.



Hình 2-3. Sơ đồ chân vi điều khiển AT89C51

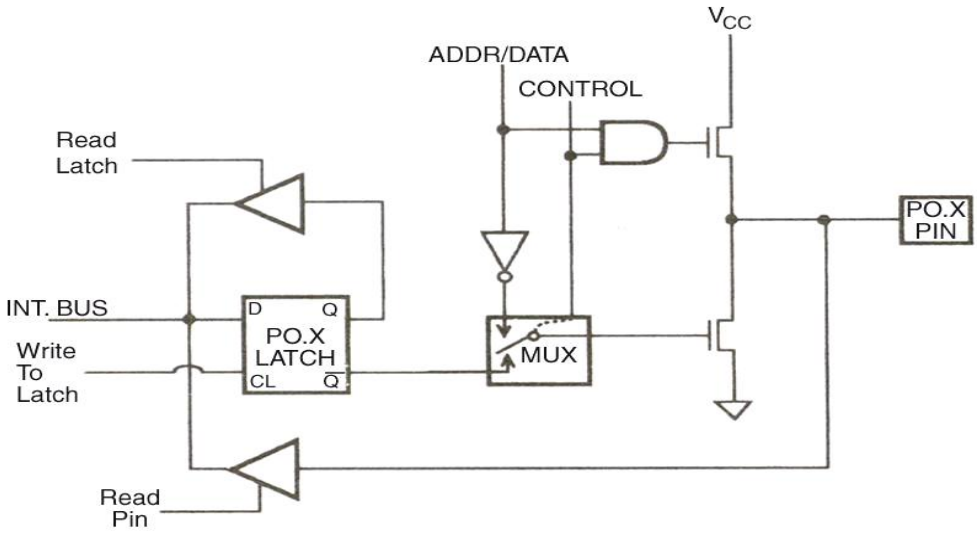
Sau đây là cấu hình các chân của AT89C51 (số thứ tự chân được nêu theo kiểu đóng vỏ PDIP).

2.3.1. Cổng P0

Cổng P0 là cổng có 2 chức năng với số thứ tự chân 32 – 39. Với cấu trúc các bit trên P0 (hình 2-4) thì thấy rằng P0 là cổng cực máng hở 8 bit xuất nhập hai chiều. Khi sử dụng chức năng IO thì các bit trên P0 nên gắn thêm điện trở khoảng 1kΩ kéo lên Vcc.

Trong các hệ thống điều khiển đơn giản sử dụng bộ nhớ bên trong không dùng bộ nhớ mở rộng bên ngoài thì P0 được dùng làm các đường điều khiển IO (Input-Output).

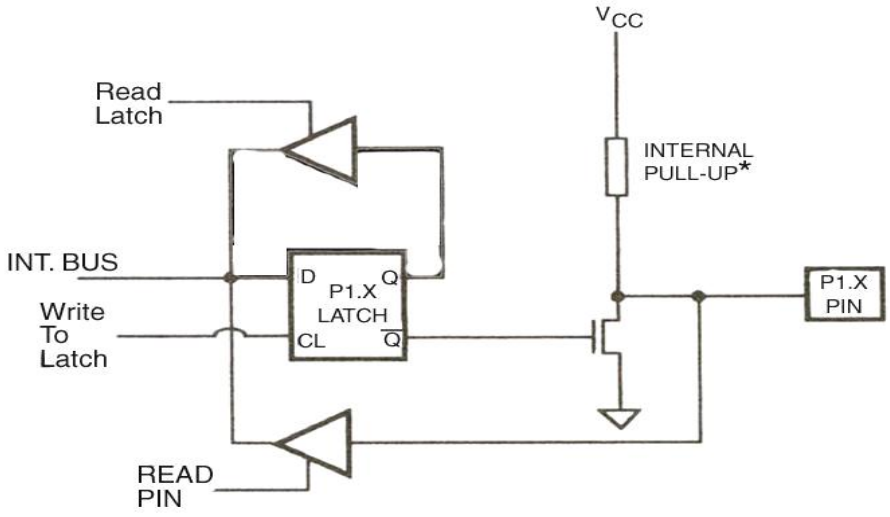
Trong các hệ thống điều khiển lớn sử dụng bộ nhớ mở rộng bên ngoài thì P0 có chức năng là bus địa chỉ và bus dữ liệu AD7 - AD0. (Address Data - địa chỉ dữ liệu)



Hình 2-4. Cấu trúc các bit trên cổng P0

2.3.2. Cổng P1

Cổng P1 với số thứ tự chân 1- 8 . P1 chỉ có một chức năng là dùng làm các đường điều khiển xuất nhập IO có các điện trở kéo lên bên trong (hình 2-5). P1 có thể xuất nhập theo bit hoặc byte. Đối với AT89C52 thì các bit P1.0 và P1.1 được dùng cho bộ Timer 2.



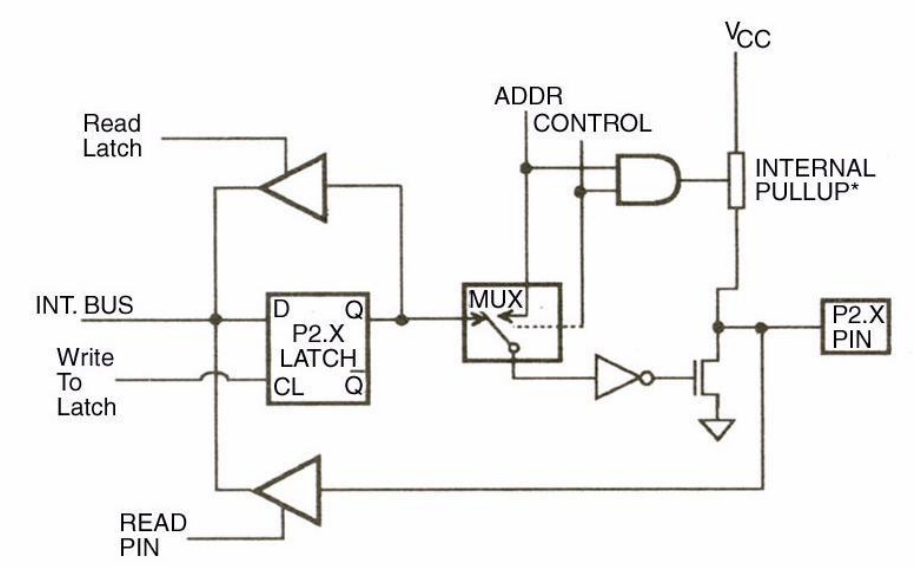
Hình 2-5. Cấu trúc các bit trên cổng P1 và P3

2.3.3. Cổng P2

Cổng P2 là cổng có 2 chức năng với số thứ tự chân 21 – 28. Cấu trúc các bit của P2 có các điện trở kéo lên (hình 2-6).

Trong các hệ thống điều khiển đơn giản sử dụng bộ nhớ bên trong không dùng bộ nhớ mở rộng bên ngoài thì P2 được dùng làm các đường điều khiển IO.

Trong các hệ thống điều khiển lớn sử dụng bộ nhớ mở rộng bên ngoài thì P2 có chức năng là bus địa chỉ cao A8 - A15.



Hình 2-6. Cấu trúc các bit trên P2

2.3.4. Cổng P3

Cổng P3 là cổng có 2 chức năng với số thứ tự chân 10 -17. Các chân của cổng này có nhiều chức năng, các công dụng chuyển đổi có liên hệ với các đặc tính đặc biệt của AT89C51 như ở bảng 2-4.

Bảng 2-4. Chức năng các chân trên P3

Bit	Tên	Chức năng chuyển đổi
P3.0	RxD	Ngõ vào nhận dữ liệu nối tiếp.
P3.1	TxD	Ngõ xuất dữ liệu nối tiếp.
P3.2	$\overline{\text{INT0}}$	Ngõ vào ngắt ngoài thứ 0.
P3.3	$\overline{\text{INT1}}$	Ngõ vào ngắt ngoài thứ 1.
P3.4	T0	Ngõ vào của Timer/Counter thứ 0.
P3.5	T1	Ngõ vào của Timer/Counter thứ 1.
P3.6	$\overline{\text{WR}}$	Tín hiệu điều khiển ghi dữ liệu lên bộ nhớ ngoài.
P3.7	$\overline{\text{RD}}$	Tín hiệu điều khiển đọc dữ liệu từ bộ nhớ ngoài.

Khi các mức logic 1 được ghi lên các bit của P3 thì các chân tương ứng được kéo lên mức cao bởi các điện trở bên trong (hình 2-5) và có thể sử dụng như là ngõ vào. Khi làm nhiệm vụ nhập thì thì các bit Port 3 được kéo xuống mức thấp do tác động của bên ngoài và sẽ cấp dòng cho các điện trở kéo lên bên trong.

2.3.5. Ngõ tín hiệu $\overline{\text{PSEN}}$ (Program Store Enable)

$\overline{\text{PSEN}}$ là tín hiệu ngõ ra ở chân 29 có tác dụng cho phép đọc bộ nhớ chương trình mở rộng thường nối đến chân OE (Output Enable hoặc RD) của EPROM cho phép đọc các byte mã lệnh.

Khi có giao tiếp với bộ nhớ chương trình bên ngoài thì mới dùng đến $\overline{\text{PSEN}}$, nếu không có giao tiếp thì chân $\overline{\text{PSEN}}$ bỏ trống.

$\overline{\text{PSEN}}$ ở mức thấp trong thời gian vi điều khiển AT89C51 lấy lệnh. Các mã lệnh của chương trình đọc từ EPROM qua bus dữ liệu và được chốt vào thanh ghi lệnh bên trong AT89C51 để giải mã lệnh. Khi AT89C51 thi hành chương trình trong EPROM nội thì $\overline{\text{PSEN}}$ ở mức cao.

2.3.6. Ngõ tín hiệu điều khiển ALE (Address Latch Enable)

Khi vi điều khiển AT89C51 truy xuất bộ nhớ bên ngoài, port 0 có chức năng là bus tải địa chỉ và bus dữ liệu AD7 – AD0 do đó phải tách các đường dữ liệu và địa chỉ. Tín hiệu ra ALE ở chân thứ 30 dùng làm tín hiệu điều khiển để giải đa hợp các đường địa chỉ và dữ liệu khi kết nối chúng với IC chốt.

Tín hiệu ra ở chân ALE là một xung trong khoảng thời gian port 0 đóng vai trò là địa chỉ thấp nên việc chốt địa chỉ được thực hiện 1 cách hoàn toàn tự động.

Các xung tín hiệu ALE có tốc độ bằng 1/6 lần tần số dao động của thạch anh gắn vào vi điều khiển và có thể dùng tín hiệu xung ngõ ra ALE làm xung clock cung cấp cho các phần khác của hệ thống.

Trong chế độ lập trình cho bộ nhớ nội của vi điều khiển thì chân ALE được dùng làm ngõ vào nhận xung lập trình từ bên ngoài để lập trình cho bộ nhớ Flash ROM trong AT89C51.

2.3.7. Ngõ tín hiệu $\overline{\text{EA}}$ (External Access)

Tín hiệu vào $\overline{\text{EA}}$ ở chân 31 phải nối lên mức 1 hoặc mức 0.

Nếu nối $\overline{\text{EA}}$ lên mức logic 1 (+5v) thì vi điều khiển sẽ thi hành chương trình từ bộ nhớ trong. Nếu nối $\overline{\text{EA}}$ với mức logic 0 (0V) thì vi điều khiển sẽ thi hành chương trình từ bộ nhớ ngoài.

2.3.8. Ngõ tín hiệu RST (Reset)

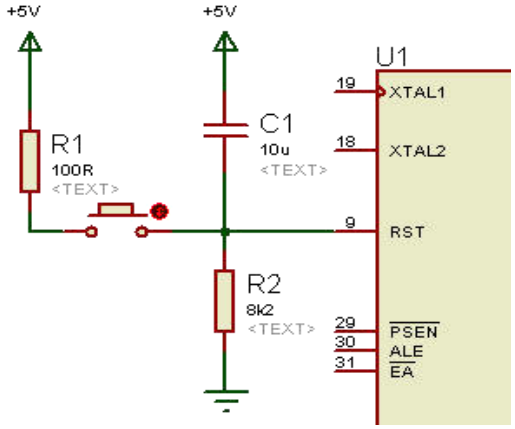
Thanh ghi quan trọng nhất là thanh ghi bộ đếm chương trình PC = 0000h.

Sau khi reset vi điều khiển luôn bắt đầu thực hiện chương trình tại địa chỉ 0000h của bộ nhớ chương trình nên các chương trình viết cho vi điều khiển luôn bắt đầu viết tại địa chỉ 0000h.

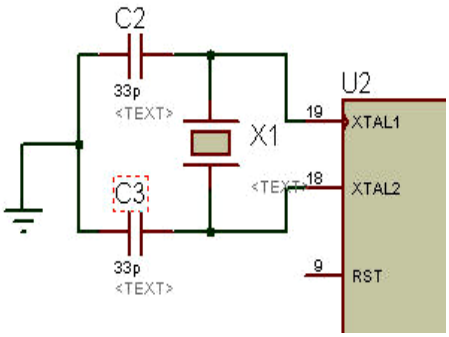
Bảng 2-5. Trạng thái của các thanh ghi trong AT89C51 sau khi reset hệ thống

Thanh ghi	Nội dung
Bộ đếm chương trình PC	0000h
Thanh ghi tích lũyA	00h
Thanh ghi B	00h
Thanh ghi trạng thái PSW	00h
Thanh ghi con trỏ SP	07h
DPTR	0000h
P0 đến P3	FFh
IP	xxx 0 0000 b
IE	0x0x 0000 b
Các thanh ghi định thời	00h
SCON SBUF	00h
PCON (HMOS)	00h
PCON (CMOS)	0xxx 0000 b

Để reset vi điều khiển thì chân RST được nối lên Vcc. Có thể reset vi điều khiển bằng một nút riêng hoặc reset khi cấp nguồn. Tuy nhiên để thuận tiện thì nên sử dụng mạch reset có cả hai tính năng (hình 2-7).



Hình 2-7. Mạch reset



Hình 2-8. Ngõ vào dao động

2.3.9. Các ngõ vào bộ dao động Xtal1, Xtal2

Bộ dao động được được tích hợp bên trong AT89C51, khi sử dụng AT89C51 người thiết kế chỉ cần kết nối thêm thạch anh và các tụ (hình 2-8). Tần số thạch anh thường sử dụng cho AT89C51 là 12Mhz ÷ 24Mhz. Tụ C2 và C3 dùng tụ gốm 30pF ± 10pF.

2.3.10. Chân cấp nguồn nuôi

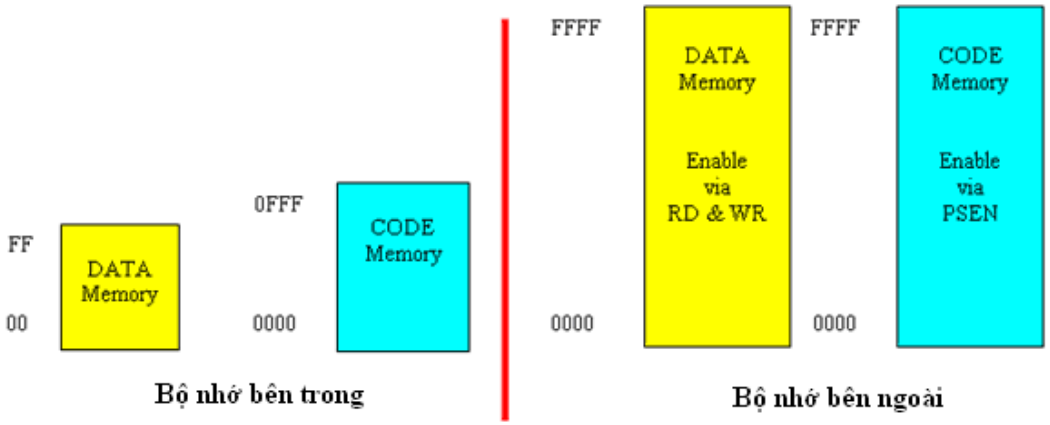
Chân 40 (Vcc) được nối lên nguồn 5V, chân 20 GND nối mass.

2.4. TỔ CHỨC BỘ NHỚ

Vi điều khiển AT89C51 có bộ nhớ nội bên trong và có thêm khả năng giao tiếp với bộ nhớ bên ngoài nếu bộ nhớ bên trong không đủ khả năng lưu trữ chương trình.

Bộ nhớ nội bên trong gồm có 2 loại bộ nhớ: bộ nhớ dữ liệu (Data Memory) và bộ chương trình (Code Memory). Bộ nhớ dữ liệu có 256 byte, bộ nhớ chương trình có dung lượng 4kbyte (89C52 có 8 kbyte, 89W55 có 16kbyte).

Bộ nhớ mở rộng bên ngoài cũng được chia ra làm 2 loại bộ nhớ: bộ nhớ dữ liệu và bộ nhớ chương trình. Khả năng giao tiếp là 64kbyte cho mỗi loại. Hình 2-9 minh họa khả năng giao tiếp bộ nhớ của vi điều khiển AT89C51.



Hình 2-9. Tóm tắt các vùng nhớ AT89C51

Bộ nhớ mở rộng bên ngoài và bộ nhớ chương trình bên trong không có gì đặc biệt vì chỉ có chức năng lưu trữ dữ liệu và mã chương trình nên không cần phải khảo sát. Bộ nhớ trong của AT89C51 được thể hiện trên hình 2-10.

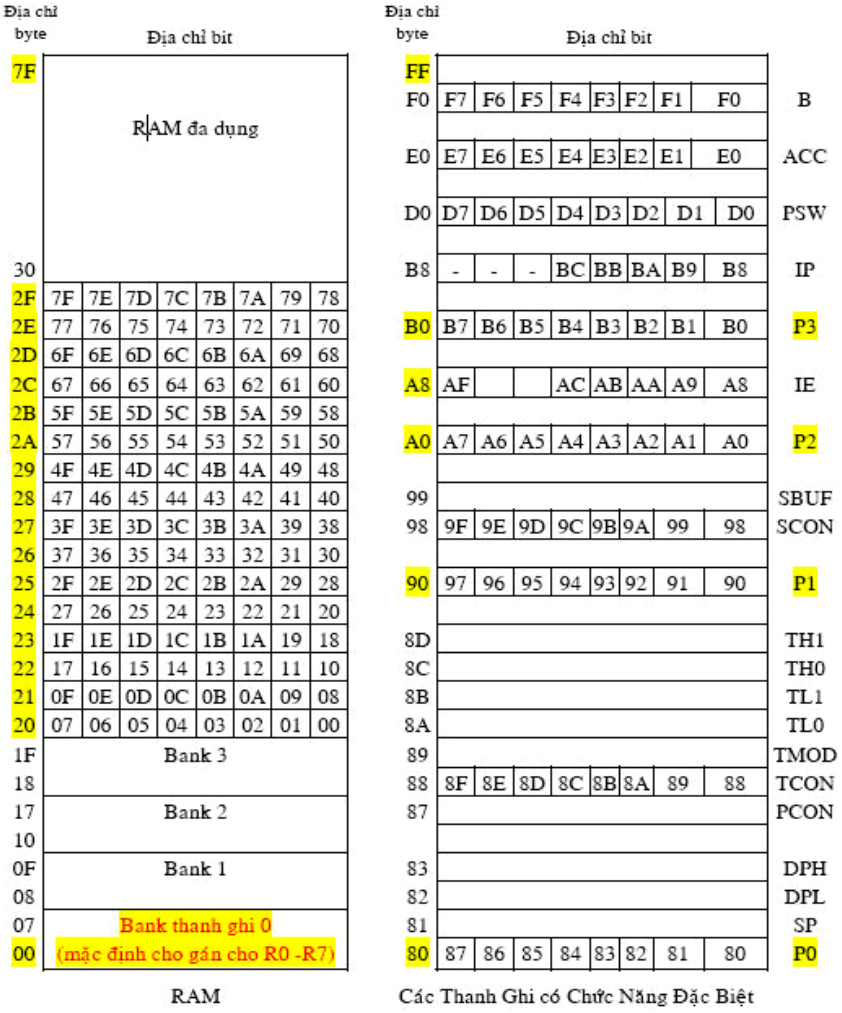
Bộ nhớ chương trình bên trong của vi điều khiển thuộc loại bộ nhớ Flash ROM cho phép xóa bằng xung điện và lập trình lại.

Bộ nhớ RAM nội bên trong là một bộ nhớ đặc biệt người sử dụng vi điều khiển cần phải nắm rõ các tổ chức và các chức năng đặc biệt của bộ nhớ này. Sơ đồ cấu trúc bên trong của bộ nhớ được trình bày như hình 2-10.

RAM bên trong AT89C51 được phân chia như sau:

- Các bank thanh ghi có địa chỉ từ 00h đến 1Fh.
- RAM địa chỉ hóa từng bit có địa chỉ từ 20h đến 2Fh.

- RAM đa dụng từ 30h đến 7Fh.
- Các thanh ghi chức năng đặc biệt từ 80h đến FFh.



Hình 2-10. Cấu trúc bộ nhớ trong của AT89C51

2.4.1. Các bank thanh ghi

32 byte thấp của bộ nhớ nội được dành cho 4 bank thanh ghi. Bộ lệnh AT89C51 hỗ trợ thêm 8 thanh ghi có tên là R0 đến R7 và theo mặc định sau khi reset hệ thống thì các thanh ghi R0 đến R7 được gán cho 8 ô nhớ có địa chỉ từ 00H đến 07H, khi đó bank 0 có 2 cách truy xuất bằng địa chỉ trực tiếp và bằng thanh ghi R.

Người lập trình dùng vùng nhớ 4 bank thanh ghi để lưu trữ dữ liệu phục vụ cho việc xử lý dữ liệu khi viết chương trình.

Chức năng chính của 4 bank thanh ghi này là nếu trong hệ thống có sử dụng nhiều chương trình thì chương trình thứ nhất có thể sử dụng hết các thanh ghi R0 đến R7 của bank0, khi chuyển sang chương trình thứ 2 để xử lý một công việc gì đó và vẫn sử dụng các thanh ghi R0 đến R7 để lưu trữ cho việc sử lý dữ liệu mà không làm ảnh

hưởng đến các dữ liệu R0 đến R7 trước đây và không cần phải thực hiện công việc cất dữ liệu thì cách nhanh nhất là gán nhóm thanh ghi R0 đến R7 cho bank1 là xong. Tương tự ta có thể mở thêm hai chương trình nữa và gán cho các bank 3 và 4.

2.4.2. RAM có thể truy xuất từng bit

Vi điều khiển AT89C51 có 210 ô nhớ bit có thể truy xuất từng bit, trong đó có 128 bit nằm ở các các ô nhớ byte có địa chỉ từ 20h đến 2Fh và các bit còn lại chứa trong nhóm thanh ghi có chức năng đặc biệt.

Các ô nhớ cho phép truy xuất từng bit và các lệnh xử lý bit là một thế mạnh của vi điều khiển. Các bit có thể được thiết lập, xóa, AND, OR bằng 1 lệnh duy nhất trong khi đó để xử lý các bit thì vi xử lý vẫn có thể xử lý được nhưng phải sử dụng rất nhiều lệnh để đạt được cùng một kết quả vì vi xử lý thường xử lý byte.

Các cổng cũng có thể truy xuất được từng bit.

Các ô nhớ bit cho phép truy xuất từng bit và cũng có thể truy xuất byte phụ thuộc vào lệnh được dùng là lệnh xử lý bit hay lệnh xử lý byte. Chú ý địa chỉ của ô nhớ byte và bit trùng nhau.

Người lập trình dùng vùng nhớ này để lưu trữ dữ liệu phục vụ cho việc xử lý dữ liệu byte hoặc bit. Các dữ liệu xử lý bit nên lưu vào vùng nhớ này.

Chú ý: Trong sơ đồ các ô nhớ nào mà chia ra làm 8 và có các con số bên trong là các ô nhớ vừa cho truy xuất byte và cả truy xuất bit. Những ô nhớ còn lại thì không thể truy xuất bit.

2.4.3. RAM đa dụng

Vùng nhớ RAM đa dụng gồm có 80 byte có địa chỉ từ 30h đến 7Fh – vùng nhớ này không có gì đặc biệt so với 2 vùng nhớ trên. Vùng nhớ bank thanh ghi 32 byte từ 00h đến 1Fh cũng có thể dùng làm vùng nhớ RAM đa dụng mặc dù các các ô nhớ này đã có chức năng như đã trình bày.

Mọi địa chỉ trong vùng RAM đa dụng đều có thể truy xuất tự do dùng kiểu địa chỉ trực tiếp hoặc gián tiếp.

Bộ nhớ ngăn xếp của vi điều khiển dùng bộ nhớ RAM nội nên dung lượng của bộ nhớ ngăn xếp nhỏ trong khi đó các bộ vi xử lý dùng bộ nhớ bên ngoài làm bộ nhớ ngăn xếp nên dung lượng tùy ý mở rộng.

2.4.4. Các thanh ghi có chức năng đặc biệt (SFR – Special Function Registers)

Các thanh ghi nội của AT89C51 được truy xuất ngầm định bởi bộ lệnh. Các thanh ghi trong AT89C51 được định dạng như một phần của RAM trên chip vì vậy mỗi thanh ghi sẽ có một địa chỉ (ngoại trừ thanh ghi bộ đếm chương trình và thanh ghi

lưu trữ mã lệnh vì các thanh ghi này đã có chức năng cố định). Cũng như các thanh ghi R0 đến R7. Vi điều khiển AT89C51 có 21 thanh ghi có chức năng đặc biệt nằm ở vùng trên của RAM nội có địa chỉ từ 80h đến FFh gồm các thanh ghi sau:

- Từ trạng thái chương trình PSW
- Thanh ghi B
- Con trỏ Stack
- Con trỏ dữ liệu DPTR
- Các thanh ghi Port
- Các thanh ghi định thời
- Các thanh ghi của Port nối tiếp
- Các thanh ghi ngắt
- Thanh ghi điều khiển nguồn

Trong phần này chỉ mang tính giới thiệu các phần tiếp theo sẽ trình bày chi tiết hơn về các thanh ghi này.

Chú ý: 128 ô nhớ có địa chỉ từ 80h đến FFh thì chỉ có 21 thanh ghi có chức năng đặc biệt được xác định các địa chỉ, còn các ô nhớ còn lại thì chưa thiết lập và trong tương lai sẽ được các nhà thiết kế vi điều khiển thiết lập thêm khi đó sẽ có các vi điều khiển thế hệ mới hơn.

Chương 3
TẬP LỆNH 8051

3.1. GIỚI THIỆU

Vi điều khiển hay vi xử lý là các IC lập trình, khi thiết kế hệ thống điều khiển có sử dụng vi xử lý hay vi điều khiển thì phần mạch điện là phần cứng, muốn hệ thống vận hành thì phải viết một chương trình điều khiển nạp vào bộ nhớ nội bên trong vi điều khiển hoặc bộ nhớ bên ngoài và gắn vào trong hệ thống để hệ thống vận hành và dĩ nhiên phải viết đúng thì hệ thống mới vận hành đúng. Chương trình gọi là phần mềm.

Phần mềm và phần cứng có quan hệ với nhau, người lập trình phải hiểu rõ hoạt động của phần cứng để viết chương trình.

Chương trình (Programe) là một tập hợp các lệnh được tổ chức theo một trình tự hợp lí để giải quyết đúng các yêu cầu của người lập trình. Người lập trình là người biết giải thuật để viết chương trình và sắp xếp đúng các lệnh theo giải thuật. Người lập trình phải biết chức năng của tất cả các lệnh của vi điều khiển để viết chương trình.

Tất cả các lệnh có thể có của một ngôn ngữ lập trình còn gọi là ***tập lệnh***.

Họ vi điều khiển 8051 đều có chung 1 tập lệnh, các vi điều khiển thế hệ sau chỉ phát triển nhiều về phần cứng còn lệnh thì ít mở rộng.

Tập lệnh họ 8051 có mã lệnh 8 bit nên có khả năng cung cấp 2^8= 256 lệnh.

Trong toàn bộ tập lệnh của vi điều khiển có 139 lệnh 1 byte, 92 lệnh 2 byte và 24 lệnh 3 byte.

Lệnh (Instruction) của vi điều khiển là một số nhị phân 8 bit (còn gọi là mã máy). 256 byte từ 0000 0000b đến 1111 1111b tương ứng với 256 lệnh khác nhau. Do lệnh được mã hóa thành dạng số nhị phân quá dài và khó nhớ nên các nhà lập trình đã xây dựng một ngôn ngữ lập trình Assembly cho dễ nhớ, điều này giúp cho việc lập trình được thực hiện một cách dễ dàng và nhanh chóng cũng như đọc hiểu và gỡ rối chương trình.

Khi viết chương trình bằng ngôn ngữ lập trình Assembly thì vi điều khiển sẽ không thực hiện được mà phải dùng chương trình biên dịch Assembler để chuyển đổi các lệnh viết bằng Assembly ra lệnh nhị phân tương ứng rồi nạp vào bộ nhớ khi đó vi điều khiển mới thực hiện được chương trình.

Ngôn ngữ lập trình Assembly do con người tạo ra, khi sử dụng ngôn ngữ Assembly để viết chương trình thì người lập trình vi điều khiển phải học hết tất cả các lệnh và viết đúng theo qui ước về cú pháp, trình tự sắp xếp dữ liệu để chương trình biên dịch có thể biên dịch đúng.

3.2. CÁC KIỂU ĐỊNH ĐỊA CHỈ BỘ NHỚ CỦA 8051

Vi điều khiển họ 8051 có 8 kiểu định địa chỉ như sau:

- Kiểu định địa chỉ dùng thanh ghi.
- Kiểu định địa chỉ trực tiếp.
- Kiểu định địa chỉ gián tiếp.
- Kiểu định địa chỉ tức thời.
- Kiểu định địa chỉ tương đối.
- Kiểu định địa chỉ tuyệt đối.
- Kiểu định địa chỉ dài.
- Kiểu định địa chỉ chỉ số.

3.2.1 . Kiểu định địa chỉ dùng thanh ghi (Register Addressing)

Kiểu này thường được dùng cho các lệnh xử lý dữ liệu mà dữ liệu luôn lưu trong các thanh ghi. Đối với vi điều khiển thì lệnh thuộc kiểu này mã hóa bằng 1 byte.

Ví dụ: Mov A,R1; sao chép nội dung thanh ghi R1 vào thanh ghi A

3.2.2 Kiểu định địa chỉ trực tiếp (Direct Addressing)

Kiểu này thường được dùng để truy xuất dữ liệu của bất kỳ ô nhớ nào trong 256 byte bộ nhớ RAM nội của vi điều khiển AT89C51. Các lệnh thuộc kiểu này thường mã hóa bằng 2 byte: byte thứ nhất là mã lệnh, byte thứ 2 là địa chỉ của ô nhớ:

Ví dụ: Mov A,05h; sao chép nội dung ô nhớ có địa chỉ 05h vào thanh ghi A

3.2.3. Định địa chỉ gián tiếp (Indirect Addressing)

Kiểu định địa chỉ gián tiếp được tượng trưng bởi ký hiệu @ và được đặt trước các thanh ghi R0, R1 hay DPTR. R0 và R1 có thể hoạt động như một thanh ghi con trỏ, nội dung của nó cho biết địa chỉ của một ô nhớ trong RAM nội mà dữ liệu sẽ ghi hoặc sẽ đọc. Còn DPTR dùng để truy xuất ô nhớ ngoại. Các lệnh thuộc dạng này mã hóa bằng 1 byte.

Ví dụ: Mov A,@R1; sao chép nội dung ô nhớ có địa chỉ trong thanh ghi R1 vào thanh ghi A

3.2.4. Định địa chỉ tức thời (Immediate Addressing)

Kiểu định địa chỉ tức thời được tượng trưng bởi ký hiệu # và được đặt trước một hằng số. Lệnh này thường dùng để nạp 1 giá trị là 1 hằng số ở byte thứ 2 (hoặc byte thứ 3) vào thanh ghi hoặc ô nhớ.

Ví dụ: Mov a,#30h; nạp dữ liệu là con số 30h vào thanh ghi A

3.2.5. Định địa chỉ tương đối (Relative Addressing)

Kiểu định địa chỉ tương đối chỉ sử dụng với những lệnh nhảy. Nơi nhảy đến có địa chỉ bằng địa chỉ đang lưu trong thanh ghi PC cộng với 1 giá trị 8 bit (còn gọi là giá trị lệch tương đối: relative offset) có giá trị từ – 128 đến +127 nên vi điều khiển có thể nhảy lùi (nếu số cộng với số âm) và nhảy tới (nếu số cộng với số dương). Lệnh này mã hóa bằng 2 byte, byte thứ 2 chính là giá trị lệch tương đối.

Nơi nhảy đến thường được xác định bởi nhãn (label) và trình biên dịch sẽ tính toán giá trị lệch.

Định vị tương đối có ưu điểm là mã lệnh cố định, nhưng nhược điểm là chỉ nhảy ngắn trong phạm vi -128 ÷127 byte (256byte), nếu nơi nhảy đến xa hơn thì lệnh này không đáp ứng được sẽ có lỗi.

Ví dụ: Sjmp X1; nhảy đến nhãn có tên là X1 nằm trong tầm vực 256 byte

3.2.6. Định địa chỉ tuyệt đối (Absolute Addressing)

Kiểu định địa chỉ tuyệt đối được dùng với các lệnh ACALL và AJMP. Các lệnh này có mã lệnh 2 byte cho phép phân chia bộ nhớ theo trang - mỗi trang có kích thước đúng bằng 2Kbyte so với giá trị chứa trong thanh ghi PC hiện hành. 11 bit địa chỉ A10÷A0 được thay thế cho 11 địa chỉ thấp trong thanh ghi PC nằm trong cấu trúc mã lệnh.

Định địa chỉ tuyệt đối có ưu điểm là mã lệnh ngắn (2 byte), nhưng nhược điểm là mã lệnh thay đổi và giới hạn phạm vi nơi nhảy đến, gọi đến không quá 2 kbyte.

Ví dụ: Ajmp X1 ;nhảy đến nhãn có tên là X1 nằm trong tầm vực 2 kbyte

3.2.7. Định địa chỉ dài (Long Addressing)

Kiểu định địa chỉ dài được dùng với lệnh LCALL và LJMP. Các lệnh này có mã lệnh 3 byte – trong đó có 2 byte (16bit) là địa chỉ của nơi đến. Cấu trúc mã lệnh là 3 byte.

Ưu điểm của định địa chỉ dài là có thể gọi 1 chương trình con hoặc có thể nhảy đến bất kỳ vùng nhớ nào trong vùng nhớ 64K, nhược điểm là các lệnh kiểu này dài 3 byte và phụ thuộc vào vị trí đến – điều này sẽ bất tiện bởi không thể dời toàn bộ mã lệnh của chương trình từ vùng nhớ này sang các vùng nhớ khác – có nghĩa là khi chương trình đã viết nơi đến tại địa chỉ 1000h thì sau khi dịch ra mã lệnh dạng số nhị phân thì sau đó nạp vào bộ nhớ thì địa chỉ bắt đầu phải đúng với địa chỉ đã viết là 1000h; nếu nạp ở vùng địa chỉ khác địa chỉ 1000h thì chương trình sẽ thực hiện sai.

Ví dụ: Ljmp X1; nhảy đến nhãn có tên là X1 nằm trong tầm vực 64 kbyte.

3.2.8. Định địa chỉ chỉ số (Index Addressing)

Kiểu định địa chỉ chỉ số "dùng một thanh ghi cơ bản: là bộ đếm chương trình PC hoặc bộ đếm dữ liệu DPTR" kết hợp với "một giá trị lệch (offset) còn gọi là giá trị tương đối (thường lưu trong thanh ghi)" để tạo ra 1 địa chỉ của ô nhớ cần truy xuất hoặc là địa chỉ của nơi nhảy đến.

Ví dụ: MOVX A, @A + DPTR; lấy dữ liệu trong ô nhớ có địa chỉ bằng DPTR + A

Khi khảo sát tập lệnh một cách chi tiết thì chức năng của các thanh ghi và các kiểu truy xuất này sẽ được trình bày rõ ràng hơn.

3.3. TẬP LỆNH

Để khảo sát tập lệnh thì phải thống nhất một số qui định về các từ ngữ kí hiệu trong tập lệnh thường được sử dụng như sau:

- Direct tượng trưng cho ô nhớ nội có địa chỉ trực tiếp (Direct).
- Rn tượng trưng cho các thanh ghi từ thanh ghi R0 đến thanh ghi R7.
- @Ri tượng trưng cho ô nhớ có địa chỉ lưu trong thanh ghi Ri và Ri chỉ có 2 thanh ghi là R0 và R1.
- Các lệnh thường xảy ra giữa các đối tượng sau:
    + Thanh ghi A.
    + Thanh ghi Rn.
    + Ô nhớ có địa chỉ direct.
    + Ô nhớ có địa chỉ lưu trong thanh ghi @Ri.
    + Dữ liệu 8 bit #data.

+ Addr11 là địa chỉ 11 bit từ A11 – A0: địa chỉ này phục vụ cho lệnh nhảy hoặc lệnh gọi chương trình con trong phạm vi 2 kbyte.

+ Addr16 là địa chỉ 16 bit từ A15 – A0: địa chỉ này phục vụ cho lệnh nhảy và lệnh gọi chương trình con ở xa trong phạm vi 64 kbyte – đó chính là địa chỉ nhảy đến, hoặc địa chỉ của chương trình con.

Khi viết chương trình người lập trình có thể thay thế địa chỉ bằng nhãn (label) để khỏi phải tính toán các địa chỉ cụ thể. Có thể nhiều nơi nhảy đến cùng một nhãn. Không được đặt các nhãn cùng tên.

Các lệnh có ảnh hưởng đến thanh ghi trạng thái thì có trình bày trong lệnh, còn các lệnh không đề cập đến thanh ghi trạng thái thì có nghĩa là nó không ảnh hưởng.

3.3.1. Nhóm lệnh di chuyển dữ liệu (8 bit)

Nhóm lệnh này dùng để sao chép dữ liệu giữa các thanh ghi, nạp dữ liệu vào các thanh ghi. Nhóm lệnh này xuất hiện nhiều nhất trong các chương trình khi lập trình vi điều khiển. Bảng 3-1 tóm tắt các lệnh thuộc nhóm này.

Bảng 3-1. Tóm tắt các lệnh di chuyển dữ liệu

Cú pháp	Kết quả	Định địa chỉ				Số chu kỳ máy
		Trực tiếp	Gián tiếp	Thanh ghi	Tức thời	
MOV A, <nguồn>	A = nguồn	x	x	x	x	1
MOV <đích>,A	<đích> = A	x	x	x		1
MOV <đích>,<nguồn>	<đích> = <nguồn>	x	x	x	x	2
MOV DPTR, #data16	DPTR = dữ liệu 16 bit				x	2
PUSH <nguồn>	Ngăn xếp = <nguồn>	x				2
POP <đích>	<đích> = ngăn xếp	x				2
XCH A, <byte>	Hoán đổi nội dung A và byte	x	x	x		1

Lệnh chuyển dữ liệu từ một thanh ghi vào thanh ghi A

Cú pháp: **Mov A, Rn**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Chuyển nội dung của thanh ghi Rn vào thanh ghi A, nội dung thanh ghi Rn vẫn giữ nguyên.

Lệnh chuyển dữ liệu từ ô nhớ trực tiếp vào thanh ghi A

Cú pháp: **Mov A, direct**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Chuyển nội dung của ô nhớ trong Ram nội có địa chỉ direct ở byte thứ hai vào thanh ghi A. Trực tiếp có nghĩa là địa chỉ của ô nhớ được ghi ở trong lệnh.

Lệnh chuyển dữ liệu từ ô nhớ gián tiếp vào thanh ghi A

Cú pháp: **MOV A,@Ri**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Chuyển nội dung ô nhớ trong Ram nội, có địa chỉ chứa trong thanh ghi Ri, vào thanh ghi A.

Lệnh nạp dữ liệu 8 bit vào thanh ghi A

Cú pháp: **MOV A, #data**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nạp dữ liệu 8 bit data (d0 đến d7) vào thanh ghi A.

Lệnh chuyển dữ liệu từ thanh ghi A vào thanh ghi Rn

Cú pháp: **Mov Rn, A**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Chuyển nội dung của thanh ghi A vào thanh ghi Rn.

Lệnh chuyển dữ liệu từ ô nhớ trực tiếp vào thanh ghi Rn

Cú pháp: **MOV Rn, direct**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Chuyển nội dung của ô nhớ trong Ram nội có địa chỉ direct vào thanh ghi Rn.

Lệnh chuyển tức thời dữ liệu 8 bit vào thanh ghi Rn

Cú pháp: **MOV Rn, #data**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nạp dữ liệu 8 bit data (d0 đến d7) vào thanh ghi Rn.

Lệnh chuyển dữ liệu từ thanh ghi A vào ô nhớ trực tiếp

Cú pháp: **MOV direct, A**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Chuyển nội dung của thanh ghi A vào ô nhớ trong Ram nội có địa chỉ direct.

Lệnh chuyển dữ liệu từ thanh ghi Rn vào ô nhớ trực tiếp

Cú pháp: **MOV direct, Rn**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Chuyển nội dung của thanh ghi Rn vào ô nhớ trong Ram nội có địa chỉ direct.

Lệnh chuyển dữ liệu từ ô nhớ trực tiếp vào ô nhớ trực tiếp

Cú pháp: **MOV direct, direct**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Chuyển nội dung của ô nhớ trong Ram nội có địa chỉ direct vào ô nhớ có địa chỉ trực direct.

Lệnh chuyển dữ liệu từ ô nhớ gián tiếp vào ô nhớ trực tiếp

Cú pháp: **MOV direct, @Ri**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Chuyển nội dung ô nhớ có địa chỉ chứa trong thanh ghi Ri vào ô nhớ có địa chỉ direct.

Lệnh chuyển dữ liệu vào ô nhớ trực tiếp

Cú pháp: **MOV direct, #data**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Nạp dữ liệu 8 bit (d0 đến d7) vào ô nhớ có địa chỉ direct.

Lệnh chuyển dữ liệu từ thanh ghi A vào ô nhớ gián tiếp

Cú pháp: **MOV direct, A**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Chuyển nội dung của thanh ghi A vào ô nhớ trong Ram nội có địa chỉ chứa trong thanh ghi Ri.

Lệnh chuyển dữ liệu từ ô nhớ trực tiếp vào ô nhớ gián tiếp

Cú pháp: **MOV @Ri, direct**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Chuyển nội dung ô nhớ có địa chỉ direct vào ô nhớ có địa chỉ chứa trong thanh ghi Ri.

Lệnh chuyển dữ liệu tức thời vào ô nhớ gián tiếp

Cú pháp: **MOV @Ri, #data**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nạp dữ liệu 8 bit (d0 đến d7) vào ô nhớ có địa chỉ chứa trong thanh ghi Ri.

Lệnh chuyển dữ liệu tức thời 16 bit vào thanh ghi con trỏ dữ liệu

Cú pháp: **MOV dptr, #data16**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Nạp dữ liệu 16 bit vào thanh ghi con trỏ dữ liệu dptr.

Lệnh chuyển dữ liệu từ ô nhớ có địa chỉ là Dptr + A vào thanh ghi A

Cú pháp: **MOVC A,@A+DPTR**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: chuyển nội dung của ô nhớ ngoài, có địa chỉ chứa bằng dptr cộng với giá trị chứa trong A, chuyển vào thanh ghi A.

Lệnh chuyển dữ liệu từ ô nhớ có địa chỉ là PC + A vào thanh ghi A

Cú pháp: **MOVC A,@A+PC**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: chuyển nội dung của ô nhớ trong bộ nhớ chương trình có địa chỉ chứa bằng PC cộng với giá trị chứa trong A được chuyển vào thanh ghi A.

Lệnh chuyển dữ liệu từ ô nhớ ngoài gián tiếp (8 bit địa chỉ) vào thanh ghi A

Cú pháp: **MOVX A, @Ri**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: chuyển nội dung ô nhớ trong bộ nhớ chương trình có địa chỉ chứa trong thanh ghi Ri vàothanh ghi A.

Lệnh chuyển dữ liệu từ ô nhớ ngoài gián tiếp (16 bit địa chỉ) vào thanh ghi A

Cú pháp: **MOVX A,@DPTR**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: chuyển nội dung của ô nhớ ngoài có địa chỉ chứa trong thanh ghi dptr vào thanh ghi A.

Lệnh chuyển dữ liệu từ thanh ghi A vào ô nhớ ngoài gián tiếp (8 bit địa chỉ)

Cú pháp: **MOVX @ Ri, A**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: chuyển nội dung của thanh ghi A ra ô nhớ ngoài có địa chỉ chứa trong thanh ghi Ri.

Lệnh chuyển dữ liệu từ thanh ghi A vào ô nhớ ngoài gián tiếp (16 bit địa chỉ)

Cú pháp: **MOVX @DPTR, A**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Chuyển nội dung của thanh ghi A ra ô nhớ ngoài có địa chỉ chứa trong thanh ghi dptr.

Lệnh cất nội dung ô nhớ trực tiếp vào ngăn xếp

Cú pháp: **PUSH direct**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: cất nội dung của ô nhớ có địa chỉ direct vào ô nhớ ngăn xếp. Con trỏ ngăn xếp SP tăng lên 1 trước khi lưu nội dung.

Lệnh lấy dữ liệu từ ngăn xếp trả về ô nhớ trực tiếp

Cú pháp: **POP direct**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: lấy nội dung của ô nhớ ngăn xếp trả cho ô nhớ có địa chỉ direct. Con trỏ ngăn xếp SP giảm 1 sau khi lấy dữ liệu ra.

Lệnh trao đổi dữ liệu giữa thanh ghi với thanh ghi A

Cú pháp: **XCH A,Rn**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Trao đổi nội dung của thanh ghi Rn với thanh ghi A.

Lệnh trao đổi dữ liệu giữa ô nhớ trực tiếp với thanh ghi A

Cú pháp: **XCH A,Direct**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Trao đổi nội dung của thanh ghi A với nội dung ô nhớ có địa chỉ direct.

Lệnh trao đổi dữ liệu giữa ô nhớ gián tiếp với thanh ghi A

Cú pháp: **XCH A,@Ri**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Trao đổi nội dung của ô nhớ có địa chỉ chứa trong thanh ghi Ri với thanh ghi A.

Lệnh trao đổi 4 bit dữ liệu giữa ô nhớ gián tiếp với thanh ghi A

Cú pháp: **XCHD A,@Ri**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Trao đổi dữ liệu 4 bit thấp của ô nhớ có địa chỉ chứa trong thanh ghi Ri với dữ liệu 4 bit thấp trong thang ghi A.

3.3.2. Nhóm lệnh số học

Nhóm lệnh số học thực hiện các phép toán số học thông thường như cộng, trừ, nhân, chia được liệt kê trong bảng 3-2.

Bảng 3-2. Tóm tắt các lệnh trong nhóm lệnh số học

Cú pháp	Kết quả	Định địa chỉ				Số chu kỳ máy
		Trực tiếp	Gián tiếp	Thanh ghi	Tức thời	
ADD A,<byte>	A = A + <byte>	x	x	x	x	1
ADDC A,<byte>	A = A + <byte >+ C	x	x	x	x	1
SUBB A, <byte>	A = A – <byte> - C	x	x	x	x	1
INC A	A = A + 1	Chỉ với thanh ghi A				1
INC <byte>	<byte> = <byte> + 1	x	x	x		1
INC DPTR	DPTR = DPTR + 1	Dữ liệu kiểu con trỏ				2
DEC A	A = A - 1	Chỉ với thanh ghi A				1
DEC <byte>	<byte> = <byte> - 1	x	x	x		1
MUL AB	B:A = B*A	Chỉ với thanh ghi A và B				4
DIV AB	A= int [A/B] B= mod [A/B]	Chỉ với thanh ghi A và B				4
DA A	Điều chỉnh thập phân	Chỉ với thanh ghi A				1

Lệnh cộng thanh ghi A với thanh ghi

Cú pháp: **ADD A,Rn**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: cộng nội dung thanh ghi A với nội dung thanh ghi Rn, kết quả lưu trong thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

Lệnh cộng nội dung ô nhớ trực tiếp vào thanh ghi A

Cú pháp: **ADD A, direct**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Cộng nội dung của ô nhớ có địa chỉ direct với nội dung thanh ghi A, kết quả chứa ở thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

Lệnh cộng nội dung ô nhớ gián tiếp vào thanh ghi A

Cú pháp: **ADD A,@Ri**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: cộng nội dung của ô nhớ có địa chỉ chứa trong thanh ghi Ri với thanh ghi A, kết qủa lưu trữ trong thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

Lệnh cộng dữ liệu tức thời 8 bit vào thanh ghi A

Cú pháp: **ADD A, #data**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Cộng dữ liệu 8 bit (d0 đến d7) với nội dung thanh ghi A, kết quả lưu trữ trong A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

Lệnh cộng thanh ghi A với thanh ghi có bit carry

Cú pháp: **ADDC A,Rn**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: cộng nội dung thanh ghi A với nội dung thanh ghi Rn với bit C, kết quả lưu trong thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

Lệnh cộng nội dung ô nhớ trực tiếp vào thanh ghi A có bit carry

Cú pháp: **ADDC A, direct**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Cộng nội dung của ô nhớ có địa direct nội dung thanh ghi A và bit C, kết quả chứa ở thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

Lệnh cộng nội dung ô nhớ gián tiếp vào thanh ghi A có bit carry

Cú pháp: **ADDC A,@Ri**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: cộng nội dung của ô nhớ có địa chỉ chứa trong thanh ghi Ri với thanh ghi A với bit C, kết quả lưu trữ trong thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

Lệnh cộng dữ liệu 8 bit vào thanh ghi A có bit carry

Cú pháp: **ADDC A, #data**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy..

Chức năng: Cộng dữ liệu 8 bit (d0 đến d7) với nội dung thanh ghi A và bit C, kết quả lưu trữ trong A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

Lệnh trừ thanh ghi A với thanh ghi

Cú pháp: **SUBB A,Rn**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Trừ nội dung thanh ghi A với nội dung thanh ghi Rn và trừ cho cờ Carry, kết quả lưu trong thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

Lệnh trừ nội dung thanh ghi A cho nội dung ô nhớ trực tiếp

Cú pháp: **SUBB A, direct**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Trừ nội dung thanh ghi A cho nội dung của ô nhớ có địa chỉ direct và trừ cho cờ Carry, kết quả chứa ở thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

Lệnh trừ nội dung thanh ghi A cho nội dung ô nhớ gián tiếp

Cú pháp: **SUBB A,@Ri**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Trừ nội dung của thanh ghi A cho dữ liệu của ô nhớ có địa chỉ chứa trong thanh ghi Ri và trừ cho cờ carry, kết qủa lưu trữ trong thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

Lệnh trừ nội dung thanh ghi A cho dữ liệu tức thời 8 bit

Cú pháp: **SUBB A, #data**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Trừ nội dung thanh ghi A cho dữ liệu 8 bit d0 đến d7 và trừ cho cờ carry, kết qủa lưu trữ trong A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

Lệnh tăng nội dung thanh ghi A

Cú pháp: **INC A**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Tăng nội dung thanh ghi A lên 1.

Lệnh tăng nội dung của thanh ghi

Cú pháp: **INC Rn**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Tăng nội dung thanh ghi Rn lên 1.

Lệnh tăng nội dung ô nhớ trực tiếp

Cú pháp: **INC direct**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Tăng nội dung của ô nhớ có địa chỉ trực tiếp ở byte thứ 2 lên 1.

Lệnh tăng nội dung ô nhớ gián tiếp

Cú pháp: **INC @Ri**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Tăng nội dung của ô nhớ có địa chỉ chứa trong Ri lên 1.

Lệnh tăng nội dung con trỏ dữ liệu Dptr

Cú pháp: **INC dptr**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Tăng nội dung của thanh ghi con trỏ dữ liệu dptr lên 1.

Lệnh giảm nội dung thanh ghi A

Cú pháp: **DEC A**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Giảm nội dung thanh ghi A xuống 1.

Lệnh giảm nội dung của thanh ghi

Cú pháp: **DEC Rn**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy..

Chức năng: Giảm nội dung thanh ghi Rn xuống 1.

Lệnh giảm nội dung ô nhớ trực tiếp

Cú pháp: **DEC direct**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Giảm nội dung của ô nhớ có địa chỉ direct ở byte thứ 2 xuống 1.

Lệnh giảm nội dung ô nhớ gián tiếp

Cú pháp: **DEC @Ri**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Giảm nội dung của ô nhớ có địa chỉ chứa trong Ri xuống 1.

Lệnh nhân thanh ghi A với thanh ghi B

Cú pháp: **MUL AB**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 4 chu kỳ máy.

Chức năng: Nội dung của thanh ghi A nhân với nội dung của thanh ghi B, kết qủa là một dữ liệu 16 bit, 8 bit thấp lưu trữ trong thanh ghi A, 8 bit cao lưu trữ trong thanh ghi B.

Lệnh chia thanh ghi A cho thanh ghi B

Cú pháp: **DIV AB**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 4 chu kỳ máy.

Chức năng: Nội dung của thanh ghi A chia cho nội dung của thanh ghi B, kết qủa của phép chia lưu trữ trong thanh ghi A, số dư lưu trữ trong thanh ghi B. Lệnh ảnh hưởng đến thanh ghi trạng thái: Bit CY và bit OV bị xóa về 0, nếu phép chia này mà dữ liệu trong thanh ghi B = 00h thì nội dung thanh ghi A không thay đổi, nội dung trong thanh ghi B không xác định và bit OV = 1, bit CY = 0.

Lệnh điều chỉnh thập phân nội dung thanh ghi A

Cú pháp: **DA A**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 4 chu kỳ máy.

Chức năng: Nếu 4 bit thấp A3A2A1A0>9 hoặc bit AC = 1thì A3A2A1A0 + 6, kết qủa lưu trở lại trong A. Nếu 4 bit cao A7A6A5A4>9 hoặc bit Cy = 1 thì A7A6A5A4 + 6, kết quả lưu trở lại thanh ghi A. Kết quả sau cùng trong thanh ghi A là số BCD.

3.3.3. Nhóm lệnh logic

Nhóm lệnh này thực hiện các hàm logic trên bit hoặc trên byte bao gồm các lệnh AND, OR, XOR, xoay …

Bảng 3-3. Tóm tắt các lệnh logic

Cú pháp	Kết quả	Định địa chỉ				Số chu kỳ máy
		Trực tiếp	Gián tiếp	Thanh ghi	Tức thời	
ANL A,<byte>	A = A AND <byte>	x	x	x	x	1
ANL <byte>,A	<byte> = <byte> AND A	x				1
ANL <byte>, # data	<byte> = <byte> AND #data	x				2
ORL A,<byte>	A = A OR <byte>	x	x	x	x	1
ORL <byte>,A	<byte> = <byte> OR A	x				1
ORL <byte>, # data	<byte> = <byte> OR #data	x				2
XRL A,<byte>	A = A XOR <byte>	x	x	x	x	1
XRL <byte>,A	<byte> = <byte> XOR A	x				1

XRL <byte>, # data	<byte> = <byte> XOR #data	x				2
CLR A	A = 00H	Chỉ với thanh ghi A				1
CPL A	A = NOT A	Chỉ với thanh ghi A				1
RL A	Xoay A sang trái 1 bit	Chỉ với thanh ghi A				1
RLC A	Xoay A sang trái 1 bit cùng với cờ CY	Chỉ với thanh ghi A				1
RR A	Xoay A sang phải 1 bit	Chỉ với thanh ghi A				1
RRC A	Xoay A sang phải 1 bit cùng với cờ CY	Chỉ với thanh ghi A				1
SWAP	Xoay thanh ghi A đi 4 bit	Chỉ với thanh ghi A				1

Lệnh and thanh ghi A với thanh ghi

Cú pháp: **ANL A,Rn (and logic)**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A and với nội dung thanh ghi Rn, kết quả lưu trữ trong thanh ghi A.

Lệnh and thanh ghi A với nội dung ô nhớ trực tiếp

Cú pháp: **ANL A, direct**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A and với nội dung của ô nhớ có địa chỉ direct, kết quả chứa ở thanh ghi A.

Lệnh and thanh ghi A với nội dung ô nhớ gián tiếp

Cú pháp: **ANL A, @Ri**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A and với ô nhớ có địa chỉ chứa trong thanh ghi Ri, kết quả lưu trữ trong thanh ghi A.

Lệnh and thanh ghi A với dữ liệu tức thời 8 bit

Cú pháp: **ANL A, #data**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung của thanh ghi A and với dữ liệu d0 đến d7, kết quả lưu trữ trong thanh ghi A.

Lệnh and nội dung ô nhớ trực tiếp với nội dung thanh ghi A

Cú pháp: **ANL direct, A**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung ô nhớ có địa chỉ direct and với nội dung của thanh ghi A, kết quả lưu trữ vào ô nhớ.

Lệnh and nội dung ô nhớ trực tiếp với dữ liệu tức thời 8 bit

Cú pháp: **ANL direct, #data**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Nội dung của ô nhớ có địa chỉ direct and với dữ liệu 8 bit, kết quả lưu trữ vào ô nhớ.

Lệnh or thanh ghi A với thanh ghi

Cú pháp: **ORL A, Rn**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A or với nội dung thanh ghi Rn, kết quả lưu trữ trong thanh ghi A.

Lệnh or thanh ghi A với nội dung ô nhớ trực tiếp

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A or với nội dung của ô nhớ có địa chỉ direct, kết qủa chứa ở thanh ghi A.

Lệnh or thanh ghi A với nội dung ô nhớ gián tiếp

Cú pháp: **ORL A, @Ri**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A or với ô nhớ có địa chỉ chứa trong thanh ghi Ri, kết quả lưu trữ trong thanh ghi A.

Lệnh or thanh ghi A với dữ liệu tức thời 8 bit

Cú pháp: **ORL A, #data**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung của thanh ghi A or với dữ liệu 8 bit (từ d0 đến d7), kết quả lưu trữ trong thanh ghi A

Lệnh or nội dung ô nhớ trực tiếp với nội dung thanh ghi A

Cú pháp: **ORL direct, A**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung ô nhớ có địa chỉ direct or với nội dung của thanh ghi A, kết qủa lưu trữ trong ô nhớ có địa chỉ direct.

Lệnh or nội dung ô nhớ trực tiếp với dữ liệu tức thời 8 bit

Cú pháp: **ORL direct, #data**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Nội dung của ô nhớ có địa chỉ direct or với dữ liệu 8 bit (từ d0 đến d7) ở byte thứ 3, kết quả lưu trữ trong ô nhớ.

Lệnh ex-or thanh ghi A với thanh ghi

Cú pháp: **XRL A, Rn**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A ex-or với nội dung thanh ghi Rn, kết quả lưu trữ trong thanh ghi A.

Lệnh ex-or thanh ghi A với nội dung ô nhớ trực tiếp

Cú pháp: **XRL A, direct**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A ex-or với nội dung của ô nhớ có địa chỉ direct, kết qủa chứa ở thanh ghi A.

Lệnh ex-or thanh ghi A với nội dung ô nhớ gián tiếp

Cú pháp: **XRL A, @Ri**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A ex-or với ô nhớ có địa chỉ chứa trong thanh ghi Ri, kết quả lưu trữ trong thanh ghi A.

Lệnh ex-or thanh ghi A với dữ liệu tức thời 8 bit

Cú pháp: **XRL A, #data**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung của thanh ghi A ex-or với dữ liệu 8bit, kết qủa lưu trữ trong thanh ghi A.

Lệnh ex-or nội dung ô nhớ trực tiếp với nội dung thanh ghi A :

Cú pháp: **XRL direct, A**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung ô nhớ có địa chỉ direct ex-or với nội dung của thanh ghi A, kết qủa lưu trữ vào ô nhớ.

Lệnh ex-or nội dung ô nhớ trực tiếp với dữ liệu tức thời 8 bit

Cú pháp: **XRL direct, #data**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Nội dung của ô nhớ có địa chỉ direct ex-or với 8 bit dữ liệu 8 bit, kết quả lưu trữ vào ô nhớ.

Lệnh xóa nội dung thanh ghi A

Cú pháp: **CLR A**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A bằng zero.

Lệnh bù nội dung thanh ghi A

Cú pháp: **CPL A**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A được lấy bù 1, kết quả chứa trong A.

Lệnh xoay trái nội dung thanh ghi A

Cú pháp: **RL A**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A được xoay trái 1 bit.

Lệnh xoay trái nội dung thanh ghi A và bit carry

Cú pháp: **RLC A**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A và bit C được xoay trái 1 bit.

Lệnh xoay phải nội dung thanh ghi A

Cú pháp: **RR A (rotate right)**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A xoay phải 1 bit ngược với lệnh RL A.

Lệnh xoay phải nội dung thanh ghi A và bit carry

Cú pháp: **RRC A**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung thanh ghi A và bit C được xoay phải 1 bit

Lệnh xoay thanh ghi A 4 bit

Cú pháp: **SWAP A**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: hoán chuyển 4 bit thấp và 4 bit cao trong thanh ghi A.

3.3.4. Nhóm lệnh chuyển quyền điều khiển

Nhóm lệnh này là nhóm lệnh chuyển quyền điều khiển có nghĩa là vi điều khiển đang thực hiện lệnh tại địa chỉ này thì có thể nhảy đến hoặc chuyển đến thực hiện lệnh tại một địa chỉ khác.

Trong nhóm này gồm có lệnh gọi chương trình con, lệnh kết thúc chương trình con trở về chương trình chính, lệnh nhảy không điều kiện và lệnh nhảy có điều kiện, lệnh trở về từ chương trình ngắt, lệnh lặp.

Các lệnh nhảy bao gồm lệnh nhảy tương đối, lệnh nhảy tuyệt đối, lệnh nhảy dài.

Các lệnh nhảy có điều kiện thì khi thỏa điều kiện thì lệnh sẽ nhảy còn nếu không thỏa mãn điều kiện thì sẽ thực hiện lệnh kế ngay sau lệnh nhảy. Ở đây chỉ trình bày điều kiện thỏa mãn còn điều kiện không thỏa mãn thì ta hiểu ngầm.

Bảng 3-4. Tóm tắt các lệnh chuyển quyền điều khiển

Cú pháp	Kết quả	Định địa chỉ				Số chu kỳ máy
		Trực tiếp	Gián tiếp	Thanh ghi	Tức thời	
JMP <nhãn>	Nhảy đến <nhãn> để thực hiện					2
JMP @A + DPTR	Nhảy đến vị trí A + DPTR					2
CALL <nhãn>	Gọi chương trình con <nhãn>					2
RET	Thoát khỏi chương trình con					2
RETI	Thoát khỏi chương trình ngắt					2
NOP	Không làm gì cả					1
JZ <nhãn>	Nhảy đến <nhãn> nếu A = 0	Chỉ với thanh ghi A				2
JZN <nhãn>	Nhảy đến <nhãn> nếu A ≠ 0	Chỉ với thanh ghi A				2
DJNZ <byte>, <nhãn>	Giảm và nhảy đến <nhãn> nếu <byte>≠ 0	x		x		2

CJNZ A,<byte>, <nhãn>	Nhảy đến <nhãn> nếu A=<byte>	x			X	2
CJNE <byte>, # data, <nhãn>	Nhảy đến <nhãn> nếu <byte>= #data		x	X		2

Lệnh gọi chương trình con dùng địa chỉ tuyệt đối

Cú pháp: **ACALL addr11**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy..

Chức năng: Khi lệnh này được thực hiện thì vi điều khiển sẽ thực hiện chương trình con tại địa chỉ addr11. Chương trình con không được cách lệnh gọi quá 2 kbyte. Addr11 của chương trình con có thể thay bằng nhãn (tên của chương trình con).

Chú ý: Trước khi nạp địa chỉ mới vào thanh ghi PC thì địa chỉ của lệnh kế trong chương trình chính được cất vào bộ nhớ ngăn xếp.

Lệnh gọi chương trình con dùng địa chỉ dài 16 bit

Cú pháp: **LCALL addr16**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Khi lệnh này được thực hiện thì vi điều khiển sẽ thực hiện chương trình con tại địa chỉ addr16. Lệnh này có thể gọi chương trình con ở bất kỳ vị trí nào cũng được trong vùng 64kbyte. Addr16 của chương trình con có thể thay bằng nhãn (tên của chương trình con). 16 bit địa chỉ A15 – A0 được nạp vào PC, vi điều khiển sẽ thực hiện chương trình con tại địa chỉ vừa nạp vào PC. Chú ý: Trước khi nạp địa chỉ vào thanh ghi PC thì địa chỉ của lệnh kế trong chương trình chính được cất vào bộ nhớ ngăn xếp.

Lệnh trở về từ chương trình con

Cú pháp: **RET**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Lệnh này sẽ kết thúc chương trình con, vi điều khiển sẽ trở lại chương trình chính để tiếp tục thực hiện chương trình.

Chú ý: lệnh này sẽ lấy địa chỉ của lệnh kế đã lưu trong bộ nhớ ngăn xếp (khi thực hiện lệnh gọi) trả lại cho thanh ghi PC để tiếp tục thực hiện chương trình chính. *Khi viết chương trình con thì phải luôn luôn kết thúc bằng lệnh RET.*

Lệnh trở về từ chương trình con phục vụ ngắt

Cú pháp: **RETI**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Lệnh này sẽ kết thúc chương trình phục vụ ngắt, vi điều khiển sẽ trở lại chương trình chính để tiếp tục thực hiện chương trình.

Lệnh nhảy dùng địa chỉ tuyệt đối

Cú pháp: **AJMP addr11**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Ý nghĩa của lệnh: vi điều khiển sẽ nhảy đến đến địa chỉ addr11 để thực hiện chương trình tại đó. Addr11 có thể thay thế bằng nhãn. Nhãn hay địa chỉ nhảy đến không quá 2 kbyte. 11 bit địa chỉ A10 – A0 được nạp vào PC, các bit cao của PC không thay đổi, vi điều khiển sẽ nhảy đến thực hiện lệnh tại địa chỉ PC mới vừa nạp.

Lệnh này khác với lệnh gọi chương trình con là không cất địa chỉ trở về. Nơi nhảy đến không quá 2 kbyte so với lệnh nhảy.

Lệnh nhảy dùng địa chỉ 16 bit

Cú pháp: **LJMP addr16**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: vi điều khiển sẽ nhảy đến địa chỉ addr16 để thực hiện chương trình tại đó. Nơi nhảy đến tùy ý nằm trong vùng 64 kbyte.

Lệnh nhảy tương đối

Cú pháp: **SJMP rel**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: vi điều khiển sẽ nhảy đến lệnh có địa chỉ tương đối (rel) để thực hiện tiếp. Có thể thay thế rel bằng nhãn. Lệnh này chỉ nhảy trong tầm vực 256 byte: có thể nhảy tới 128 byte và có thể nhảy lùi 128 byte. Khi tầm vực nhảy xa hơn ta nên dùng lệnh AJMP hay LJMP.

Chú ý: rel (relative: tương đối) các lệnh có xuất hiện "rel" đều liên quan đến lệnh nhảy: nơi nhảy đến được tính bằng cách lấy nội dung của PC cộng với số lượng byte của các lệnh nằm giữa lệnh nhảy và nơi nhảy đến. Chúng ta không cần quan tâm đến điều này vì chương trình biên dịch của máy tính sẽ tính giúp chúng ta.

Lệnh nhảy gián tiếp

Cú pháp: **JMP @A + DPTR**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: lệnh sẽ nhảy đến nơi có địa chỉ bằng nội dung của A cộng với dptr để tiếp tục thực hiện chương trình tại đó.

Lệnh nhảy nếu cờ Z =1 (nội dung thanh ghi A bằng 0)

Cú pháp: **JZ rel (jump zero)**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: nếu bit Z = 1 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel (thỏa điều kiện), nếu Z = 0 thì vi điều khiển sẽ tiếp tục thực hiện lệnh kế (không thỏa điều kiện).

Lệnh nhảy nếu cờ Z = 0 (nội dung thanh ghi A khác 0)

Cú pháp: **JNZ rel**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: nếu Z = 0 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

Lệnh nhảy nếu bit carry = 1

Cú pháp: **JC rel**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: nếu bit carry CY = 1 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

Lệnh nhảy nếu bit carry = 0

Cú pháp: **JNC rel**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: nếu bit carry CY = 0 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

Lệnh nhảy nếu bit = 1

Cú pháp: **JB bit, rel**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: nếu nội dung của bit có địa chỉ bit (được xác định bởi byte thứ 2) bằng 1 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

Lệnh nhảy nếu bit = 0

Cú pháp: **JNB bit, rel**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: nếu nội dung của bit có địa chỉ bit (được xác định bởi byte thứ 2) bằng 0 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ bằng rel.

Lệnh nhảy nếu bit = 1 và xóa bit

Cú pháp: **JBC bit, rel**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: nếu bit được xác định bởi byte thứ 2 bằng 1 thì bit này được xóa về 0 và vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

Lệnh so sánh ô nhớ trực tiếp với nội dung thanh ghi A

Cú pháp: **CJNE A, direct, rel (compare jump if not equal)**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: lệnh này ảnh hưởng đến cờ CY và thực hiện việc nhảy như sau:

- Nếu nội dung của A ≥ nội dung của ô nhớ có địa chỉ direct thì bit CY = 0.
- Nếu nội dung của A < nội dung của ô nhớ có địa chỉ direct thì bit CY = 1.
- Nếu nội dung của A khác nội dung của ô nhớ có địa chỉ direct thì lệnh sẽ nhảy đến và thực hiện lệnh tại địa chỉ rel.
- Nếu nội dung của A bằng nội dung của ô nhớ có địa chỉ direct thì không nhảy và làm lệnh kế.

Lệnh so sánh dữ liệu tức thời với nội dung thanh ghi A

Cú pháp: **CJNE A, #data, rel**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: lệnh này ảnh hưởng đến cờ C và thực hiện việc nhảy như sau:

- Nếu nội dung của A ≥ data 8 bit thì bit C = 0.
- Nếu nội dung của A < data 8 bit thì bit C = 1.
- Nếu nội dung của A khác data 8 bit thì lệnh sẽ nhảy đến thực hiện lệnh tại địa chỉ rel.
- Nếu nội dung của A bằng data 8 bit thì không nhảy và làm lệnh kế.

Lệnh so sánh dữ liệu tức thời với nội dung thanh ghi

Cú pháp: **CJNE Rn, #data, rel**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: lệnh này ảnh hưởng đến cờ C và thực hiện việc nhảy như sau:

- Nếu nội dung của thanh ghi Rn ≥ data 8 bit thì bit C = 0.
- Nếu nội dung của thanh ghi Rn < data 8 bit thì bit C = 1.
- Nếu nội dung của thanh ghi Rn khác data 8 bit thì lệnh sẽ nhảy đến thực hiện lệnh tại địa chỉ rel.
- Nếu nội dung của thanh ghi Rn bằng data 8 bit thì không nhảy và làm lệnh kế.

Lệnh so sánh dữ liệu tức thời với dữ liệu gián tiếp

Cú pháp: **CJNE @Ri, #data, rel**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: lệnh này ảnh hưởng đến cờ C và thực hiện việc nhảy như sau:

- Nếu nội dung của ô nhớ có địa chỉ lưu trong thanh ghi Ri ≥ data 8 bit thì bit C =0.
- Nếu nội dung của ô nhớ có địa chỉ lưu trong thanh ghi Ri < data 8 bit thì bit C =1.
- Nếu nội dung của ô nhớ có địa chỉ lưu trong thanh ghi Ri khác data 8 bit thì lệnh sẽ nhảy đến thực hiện lệnh tại địa chỉ rel.
- Nếu nội dung của ô nhớ có địa chỉ lưu trong thanh ghi Ri bằng data 8 bit thì không nhảy và làm lệnh kế.

Lệnh giảm thanh ghi và nhảy

Cú pháp: **DJNZ Rn, rel (decrement and jump if not zero)**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Nội dung của thanh ghi Rn giảm đi 1 và nếu kết quả trong thanh ghi Rn sau khi giảm khác 0 thì vi điều khiển sẽ thực hiện chương trình tại địa chỉ rel, nếu kết quả bằng 0 thì vi điều khiển sẽ tiếp tục thực hiện lệnh kế.

Lệnh giảm ô nhớ trực tiếp và nhảy

Cú pháp: **DJNZ direct, rel**

Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Nếu nội dung của ô nhớ có địa chỉ direct giảm đi 1 và nếu kết quả sau khi giảm khác 0 thì vi điều khiển sẽ thực hiện chương trình tại địa chỉ rel, ngược lại nếu kết quả bằng 0 thì vi điều khiển sẽ tiếp tục thực hiện lệnh kế.

Lệnh Nop

Cú pháp: **NOP**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Nội dung của PC tăng lên 1 và tiếp tục thực hiện lệnh tiếp theo.

3.3.5. Nhóm lệnh xử lý bit

Bảng 3-5. Tóm tắt các lệnh xử lý bit

Cú pháp	Kết quả	Số chu kỳ máy.
ANL C,bit	C = C AND bit	2
ANL C,/bit	C = C AND (NOT bit)	2
ORL C,bit	C = C OR bit	2
ORL C,/bit	C = C OR (NOT bit)	2
MOV C,bit	C = bit	1
MOV bit,C	bit = C	2
CLR C	C = 0	1
CLR bit	bit = 0	1
SETB C	C = 1	1
SETB bit	bit = 1	1
CPL C	C = NOT C	1
CPL bit	bit = NOT bit	1
JC rel	Nhảy nếu C = 1	2
JNC rel	Nhảy nếu C = 0	2
JB bit,rel	Nhảy nếu bit = 1	2
JNB bit,rel	Jump nếu bit = 0	2
JBC bit,rel	Nhảy nếu bit = 1 và đồng thời xóa bit	2

Lệnh xóa bit carry

Cú pháp: **CLR C**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Xóa bit C về 0.

Lệnh xóa bit

Cú pháp: **CLR bit**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Xóa bit có địa chỉ được xác định bởi byte thứ 2 về 0.

Lệnh thiết lập bit carry

Cú pháp: **SETB C**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Thiết lập bit C = 1.

Lệnh thiết lập bit

Cú pháp: **SETB bit**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Thiết lập bit có địa chỉ được xác định bởi byte thứ 2 lên 1.

Lệnh bù bit carry

Cú pháp: **CPL C**

Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Bù bit carry, nếu trước đó C = 1 thì C = 0, nếu C = 0 thì C = 1.

Lệnh bù bit

Cú pháp: **CPL bit**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Bù bit có địa chỉ xác định bởi byte thứ 2, nếu trước đó bit này = 0 thì kết quả bit này bằng 1 và ngược lại nếu trước đó bằng 1 thì nó sẽ bằng 0.

Lệnh and bit carry với bit

Cú pháp: **ANL C, bit**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Bit C and với bit có địa chỉ được xác định bởi byte thứ 2, kết quả chứa ở bit C.

Lệnh and bit carry với bù bit

Cú pháp: **ANL C, /bit**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Bit C and với bù bit có địa chỉ được xác định bởi byte thứ 2, kết quả chứa ở bit C.

Lệnh or bit carry với bit

Cú pháp: **ORL C, bit**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Bit C or với bit có địa chỉ được xác định bởi byte thứ 2, kết quả chứa ở bit C.

Lệnh or bit carry với bù bit

Cú pháp: **ORL C, /bit**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Bit C or với bù bit có địa chỉ được xác định bởi byte thứ 2, kết quả chứa ở bit C.

Lệnh di chuyển bit vào bit carry

Cú pháp: **MOV C, bit**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.

Chức năng: Bit có địa chỉ xác định bởi byte thứ 2 được chuyển vào bit C.

Lệnh di chuyển bit carry vào bit

Cú pháp: **MOV bit, C**

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Chức năng: Bit C chuyển vào bit có địa chỉ được xác định bởi byte thức.

Chương 4
BỘ ĐỊNH THỜI/ĐẾM

4.1. GIỚI THIỆU

Các bộ định thời/đếm (Timer/Counter) trong vi điều khiển được sử dụng rộng rãi trong các ứng dụng đo lường và điều khiển. Các Timer hay Counter chỉ là một và chính là bộ đếm có chức năng đếm xung.

Trong vi điều khiển 8051 có 2 Timer/Counter 0 và Timer/Counter 1, còn 8052 thì có 3 Timer/Counter 0, Timer/Counter 1 và Timer/Counter 3.

Nếu sử dụng ở chế độ Timer thì thời gian định thời nhân với chu kỳ của mỗi xung sẽ tạo ra lượng thời gian cần thiết, ở chế độ Timer vi điều khiển thường đếm xung nội lấy từ mạch dao động bên trong vi điều khiển có chu kỳ ổn định. Chế độ Timer dùng để định thời gian chính xác để điều khiển các thiết bị theo thời gian. Trong ứng dụng định khoảng thời gian thì Timer được lập trình sao cho sẽ tràn sau một khoảng thời gian và thiết lập cờ tràn bằng 1. Cờ tràn được dùng bởi chương trình để thực hiện một hành động tương ứng như kiểm tra trạng thái các ngõ vào ra hoặc gửi các sự kiện đến các ngõ ra.

Nếu sử dụng ở chế độ Counter thì ta chỉ cần quan tâm đến số lượng xung đếm được, không cần quan tâm đến chu kỳ của xung đếm. Chế độ Counter thường đếm xung nhận từ bên ngoài đưa đến ngõ vào T0 đối với Timer/Counter thứ 0 và ngõ vào T1 đối với Timer/Counter thứ 1. Đếm xung từ bên ngoài còn gọi là đếm sự kiện. Một ứng dụng cho chế độ Counter là có thể sử dụng vi điều khiển làm các mạch đếm sản phẩm.

Đến đây ta có thể xem Timer hay Counter là 1 và chú ý rằng tại mỗi một thời điểm ta chỉ sử dụng một trong 2 chức năng hoặc là Timer hoặc là Counter.

Các Timer/Counter của vi điều khiển sử dụng 16 flip flop nên ta gọi là Timer/ Counter 16 bit và số lượng xung mà Timer/Counter có thể đếm được tính theo số nhị phân bắt đầu từ 0000 0000 0000 0000b đến 1111 1111 1111 1111b, nếu viết theo số hexa thì bắt đầu từ 0000h đến FFFFh và nếu tính theo giá trị thập phân thì bắt đầu từ 0 đến 65535.

Khi đạt đến giá trị cực đại và nếu có thêm 1 xung nữa thì bộ đếm sẽ bị tràn, khi bị tràn thì giá trị đếm sẽ tự động về 0 (giống như mạch đếm nhị phân 4 bit khi đếm lên 1111 và nếu có 1 xung nữa thì giá trị đếm về 0000) và cờ tràn của Timer/Counter lên 1 để báo hiệu Timer/Counter đã bị tràn (Trước khi đếm thì phải xoá cờ tràn).

Người lập trình sử dụng trạng thái cờ tràn lên 1 để rẽ nhánh hoạt chấm dứt thời gian cần thiết đã định để chuyển sang làm một công việc khác. Và khi cờ tràn lên 1 sẽ tạo ra ngắt cũng để rẽ nhánh chương trình để thực hiện một chương trình khác.

Các giá trị đếm được của Timer/Counter 0 thì lưu trong 2 thanh ghi TH0 và TL0 – mỗi thanh ghi 8 bit kết hợp lại thành 16 bit.

Tương tự, các giá trị đếm được của Timer/Counter 1 thì lưu trong 2 thanh ghi TH1 và TL1 – mỗi thanh ghi 8 bit kết hợp lại thành 16 bit.

Ngoài các thanh ghi lưu trữ số xung đếm vừa giới thiệu thì còn có 2 thanh ghi hỗ trợ kèm theo: thanh ghi TMOD và thanh ghi TCON dùng để thiết lập nhiều chế độ hoạt động khác nhau cho Timer/Counter để đáp ứng được sự đa dạng các yêu cầu ứng dụng trong thực tế.

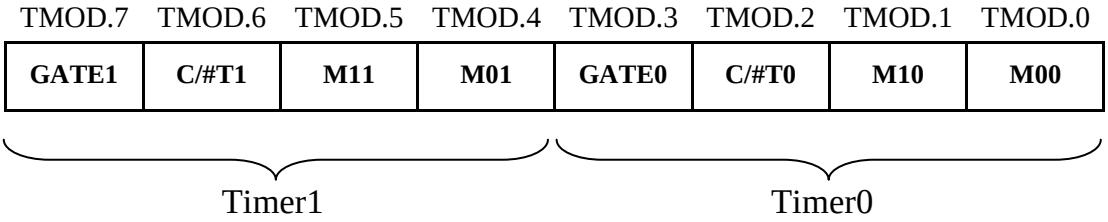
4.2. CÁC THANH GHI LIÊN QUAN ĐẾN TIMER/COUNTER

Bảng 4-1 liệt kê tên, chức năng, địa chỉ của các thanh ghi liên quan đến các Timer/Counter của vi điều khiển AT89C51.

Bảng 4-1. Các thanh ghi liên quan đến các Timer/Counter của vi điều khiển AT89C51

Tên	Chức năng	Địa chỉ	Cho phép truy xuất bit
TCON	Điều khiển	88h	Có
TMOD	Chọn chế độ làm việc	89h	Không
TL0	Byte thấp của Timer 0	8Ah	Không
TL1	Byte thấp của Timer 1	8Bh	Không
TH0	Byte cao của Timer 0	8Ch	Không
TH1	Byte cao của Timer 1	8Dh	Không

4.2.1. Thanh ghi chọn chế độ làm việc cho Timer/Counter (Thanh ghi TMOD)



Hình 4-1. Thanh ghi TMOD

Thanh ghi TMOD chứa hai nhóm 4 bit dùng để thiết lập chế độ làm việc cho Timer 0 và Timer 1

Bảng 4-2. Bảng mô tả hoạt động thanh ghi TMOD

Bit	Tên	Mô tả
7	Gate1	Nếu Gate1=1 thì Timer1 chỉ làm việc khi INT1=1 Nếu Gate1=0 thì hoạt động của Timer1 không bị ảnh hưởng bởi mức logic trên chân INT1

6	C/#T1	Nếu C/#T1=0 Timer/Counter1 là bộ định thời (Timer) Nếu C/#T1=1 Timer/Counter1 là bộ đếm (Counter)
5	M11	Chọn chế độ cho Timer/Counter1 00: Chế độ 0 – Chế độ 13 bit 01: Chế độ 1 – Chế độ 16 bit 10: Chế độ 2 – Chế độ 8 bit tự động nạp lại
4	M01	
3	Gate0	Nếu Gate0=1 thì Timer0 chỉ làm việc khi INT0=1 Nếu Gate0=0 thì hoạt động của Timer0 không bị ảnh hưởng bởi mức logic trên chân INT0
2	C/#T0	Nếu C/#T0=0 Timer/Counter0 là bộ định thời Nếu C/#T0=1 Timer/Counter0 là bộ đếm
1	M10	Các bit chọn chế độ cho Timer/Counter0 00: Chế độ 0 – Chế độ 13 bit 01: Chế độ 1 – Chế độ 16 bit 10: Chế độ 2 – Chế độ 8 bit tự động nạp lại 11: Chế độ 3 – Timer 0 được tách ra làm hai timer 8 bit gồm có: TL0 là Counter/Timer 8 bit TH0 là Timer 8 bit
0	M00	

Ghi chú: Giá trị thanh ghi khi Reset là 00000000b

4.2.2. Thanh ghi điều khiển Timer (Thanh ghi TCON)

TCON.7	TCON.6	TCON.5	TCON.4	TCON.3	TCON.2	TCON.1	TCON.0
TF1	**TR1**	**TF0**	**TR0**	**IE1**	**IT1**	**IE0**	**IT0**

Hình 4-2. Thanh ghi TCON

Chức năng các bit trong thanh ghi TCON được mô tả trong bảng 4-3.

Bảng 4-3. Bảng mô tả hoạt động thanh ghi TCON

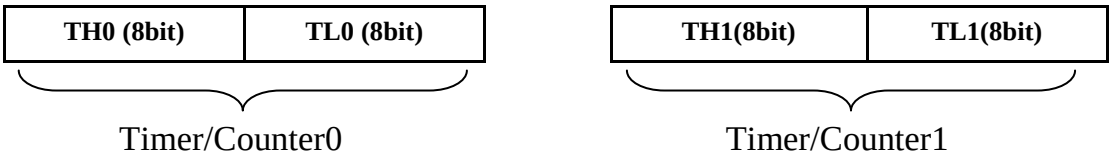
Bit	Tên	Mô tả
7	TF1	Cờ báo tràn của Timer1. Được thiết lập bởi phần cứng khi có tràn Timer1, được xóa bởi phần mềm hoặc bởi phần cứng khi bộ xử lý chỉ đến chương trình phục vụ ngắt.
6	TR1	Bit điều khiển Timer1 hoạt động. Được thiết lập/xóa bằng phần mềm để điều khiển Timer1 chạy/dừng. TR1=0 thì Timer1 không được cấp xung (Dừng) TR1=1 thì Timer1 được cấp xung (Chạy)

5	TF0	Cờ báo tràn của Timer0 (Hoạt động tương tự TF1)
4	TR0	Bit điều khiển Timer0 hoạt động. (Hoạt động tương tự TR1)
3	IE1	Cờ báo ngắt ngoài 1. Khi có ngắt xảy ra ở ngõ vào INT1 (cạnh xuống hoặc mức thấp) thì cờ IE1 tác động lên mức 1. Khi vi điều khiển thực hiện chương trình con phục vụ ngắt INT1 thì tự động xóa luôn cờ báo ngắt IE1
2	IT1	Bit điều khiển cho phép ngắt INT1 tác động bằng mức hay bằng cạnh. IT1 = 0 thì ngắt ngoài INT1 tác động bằng mức thấp. IT1 = 1 thì ngắt ngoài INT1 tác động bằng cạnh xuống.
1	IE0	Cờ báo ngắt ngoài 0 (Hoạt động giống IE1)
0	IT0	Bit điều khiển ngắt 0 (Hoạt động giống IT1)

Ghi chú: Giá trị thanh ghi khi Reset là 0000000b

4.2.3. Các thanh ghi chứa giá trị của các bộ Timer/Counter

Các Timer/Counter đều là 16 bit, mỗi Timer/Counter có 2 thanh ghi 8 bit để chứa các giá trị khởi tạo hoặc các giá trị hiện thời của các Timer/Counter (Hình 4-3). Cụ thể Timer/Counter0 có TH0 và TL0, Timer/Counter1 có TH1 và TL1. Các thanh ghi này không được định địa chỉ bit.



Hình 4-3. Các thanh ghi chứa giá trị của bộ Timer/Counter

4.3. CÁC CHẾ ĐỘ HOẠT ĐỘNG CỦA BỘ ĐỊNH THỜI

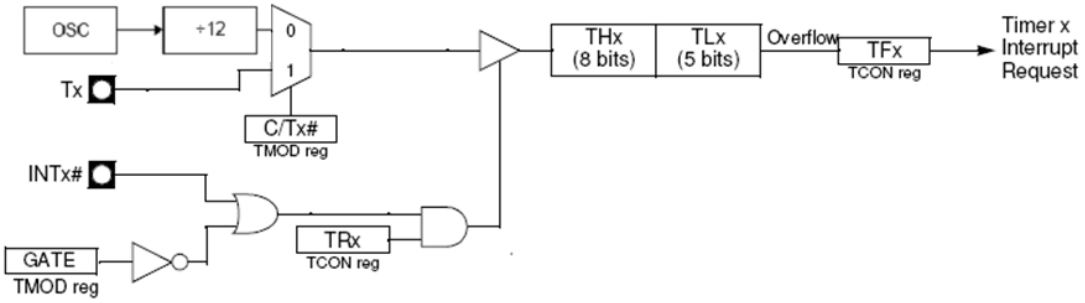
Vi điều khiển 8051 có 2 Timer là Timer0 và Timer1. Ta dùng ký hiệu TLx và THx để chỉ 2 thanh ghi byte thấp và byte cao của Timer0 hoặc Timer1. Như đã trình bày ở trên các Timer có 4 kiểu hoạt động, phần này ta sẽ khảo sát chi tiết các kiểu hoạt động của Timer.

4.3.1. Chế độ định thời 13 bit (Chế độ 0 – Mode 0)

Chế độ này tương thích với các bộ vi điều khiển trước đó, trong các ứng dụng hiện nay thì chế độ này ít sử dụng.

Ở chế độ này 8 bit cao sử dụng hết 8 bit của thanh ghi THx, 5 bit còn lại chỉ sử dụng 5 bit trọng số thấp của thanh ghi TLx, còn 3 bit cao của TLx không dùng đến (hình 4-4).

Ở chế độ này giá trị đếm lớn nhất là 8191 tức đếm từ 000h đến 1FFFh và nếu có thêm một xung nữa thì bộ đếm sẽ tràn và làm cho cờ tràn lên 1.

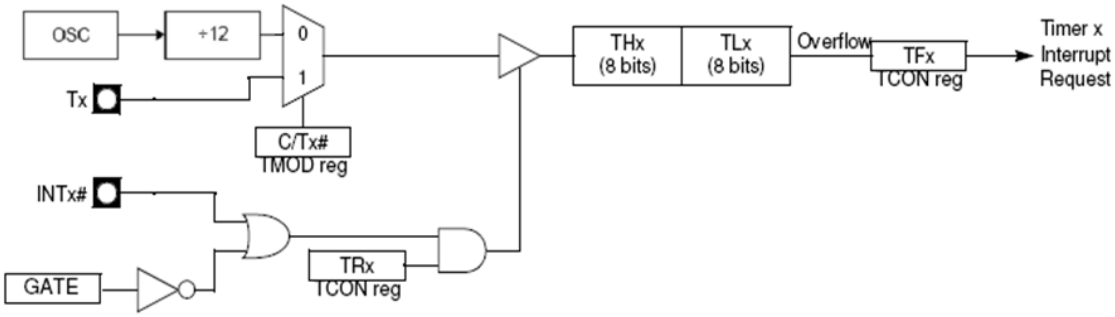


Hình 4-4. Timer/Counter 0 và 1 ở chế độ 0

4.3.2. Chế độ định thời 16 bit (Chế độ 1 – Mode 1)

Cấu hình chế độ này giống chế độ 0, chỉ khác bây giờ Timer là 16 bit và sử dụng hết tất cả các bit của hai thanh ghi THx và TLx (Hình 4-5). Giá trị đếm lớn nhất trong chế độ này là 65535 tức là từ 0000h đến FFFFh. Cờ tràn tương ứng sẽ bằng 1 nếu như có sự chuyển số đếm từ FFFFh xuống 0000h và Timer tiếp tục đếm.. Cờ tràn là bit TFx trong thanh ghi điều khiển Timer TCON, bit này được đọc hoặc ghi bằng phần mềm.

Các thanh ghi giá trị định thời THx và TLx có thể được đọc hoặc ghi ở một thời điểm bất kỳ bằng phần mềm.

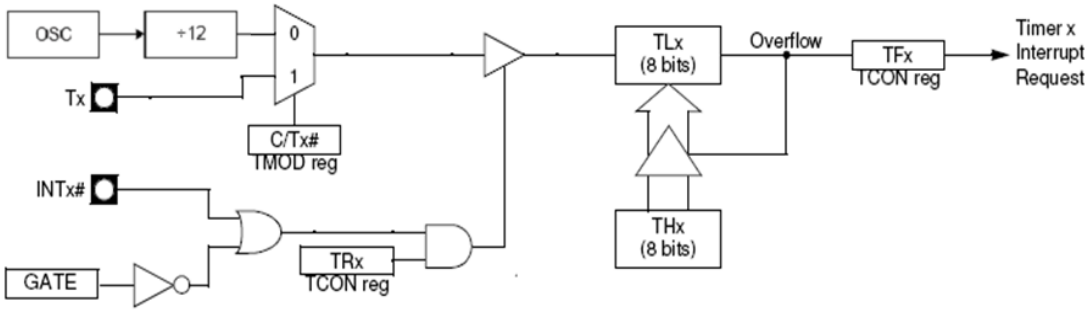


Hình 4-5. Timer/Counter 0 và 1 ở chế độ 1

4.3.3. Chế độ tự động nạp lại (Chế độ 2 – Mode 2)

Byte thấp của Timer (TLx) chứa giá trị định thời 8 bit, byte cao của Timer (THx) lưu giữ giá trị nạp lại. Khi số đếm tràn từ FFh xuống 00h thì cờ tràn của Timer được thiết lập lên 1 và đồng thời giá trị trong thanh ghi THx sẽ được nạp vào thanh ghi

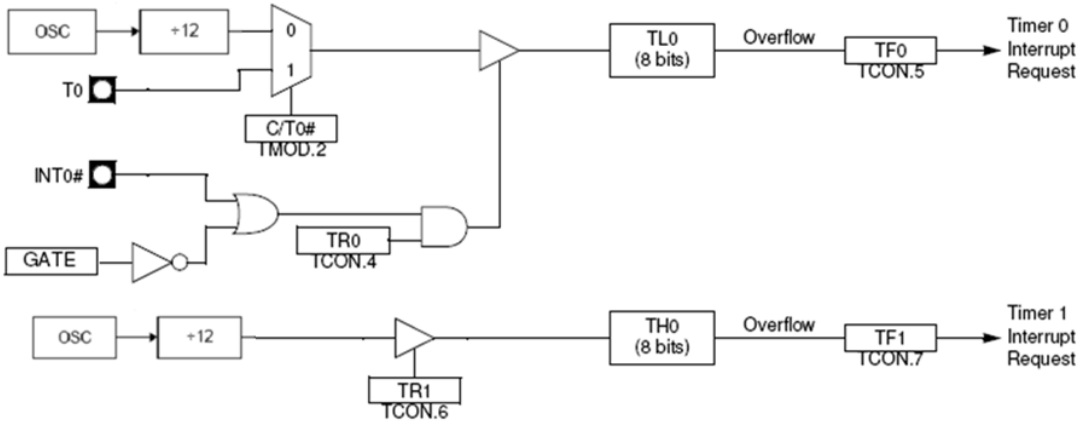
TLx, việc đếm sẽ bắt đầu từ giá trị này và cho đến khi có tràn tiếp theo và quá trình được lặp lại (Hình 4-6). Chế độ này khá tiện lợi bởi việc tràn Timer xảy ra ở những khoảng thời gian xác định và tuần hoàn một khi thanh ghi TMOD và THx được khởi động.



Hình 4-6. Timer/Counter 0 và 1 ở chế độ 2

4.3.4. Chế độ tách Timer (Chế độ 3 – Mode 3)

Chế độ 3 là chế độ định thời tách Timer và từng Timer hoạt động khác nhau. Timer 0 ở chế độ 3 được chia làm 2 bộ 8 bit hoạt động riêng rẽ là TL0 và TH0, mỗi Timer sẽ thiết lập các cờ tràn tương ứng là TF0 và TF1 khi xảy ra tràn (Hình 4-7).



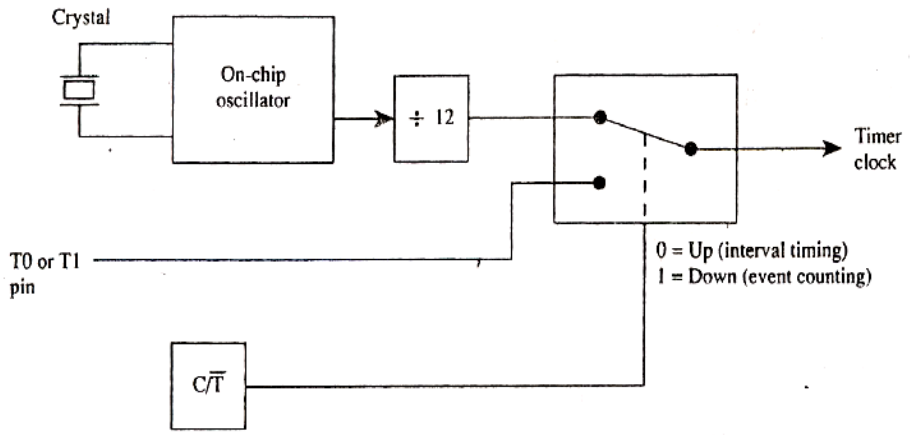
Hình 4-7. Timer/Counter 0 ở chế độ 3

Timer 1 không hoạt động ở chế độ 3 nhưng có thể được khởi động bằng cách chuyển Timer này vào một trong các chế độ khác và không thể báo cờ tràn vì TF1 đã được sử dụng để báo tràn cho TH0.

Khi Timer 0 hoạt động ở Mode 3 sẽ cung cấp thêm 1 Timer 8 bit thứ ba. Khi Timer0 ở mode 3, Timer 1 có thể hoạt động như là một bộ dao động thiết lập tốc độ Baud phục vụ cho cổng nối tiếp để truyền và nhận dữ liệu, hoặc nó có thể dùng trong các ứng dụng mà không sử dụng chế độ báo tràn và báo ngắt.

4.4. NGUỒN XUNG CUNG CẤP CHO TIMER/COUNTER

Timer/Counter có thể đếm xung từ hai nguồn: nếu Timer/Counter sử dụng ở chế độ định thời (Timer) thì sẽ đếm xung bên trong (xung nội) đã biết tần số, nếu Timer/Counter sử dụng ở chế độ đếm (Counter) thì đếm xung từ bên ngoài (hình 4-8)**.** Bit C/#T trong TMOD cho phép chọn chế độ Timer hay Counter khi khởi tạo ở thanh ghi TMOD.



Hình 4-8. Nguồn xung clock

4.4.1. Hoạt động định thời

Nếu bit C/#T = 0 thì Timer hoạt động đếm nội xung liên tục lấy từ dao động trên Chip. Tần số ngõ vào tụ thạch anh được đưa qua một mạch chia 12 để giảm tần số phù hợp với các ứng dụng. Nếu dùng thạch anh 12MHz thì sau khi qua bộ chia 12 tần số đưa đến bộ đếm Timer là 1MHz.

Timer sẽ sinh ra tràn khi nó đã đếm đủ số xung tương ứng, thời gian qui định phụ thuộc vào giá trị khởi tạo được nạp vào các thanh ghi THx và TLx.

4.4.2. Đếm sự kiện

Nếu bit C/#T = 1 thì Timer hoạt động đếm xung đến từ bên ngoài và chu kỳ của mỗi xung do nguồn tạo tín hiệu bên ngoài quyết định. Hoạt động này thường dùng để đếm các sự kiện. Số lượng các sự kiện được lưu trữ trong thanh ghi của các Timer.

Nguồn xung clock bên ngoài đưa vào chân P3.4 (T0) là ngõ nhập xung clock của Timer0 và P3.5 (T1) là ngõ nhập xung clock của Timer1.

Trong các ứng dụng đếm xung từ bên ngoài các thanh ghi Timer sẽ tăng giá trị đếm khi xung ngõ vào Tx chuyển trạng thái từ 1 sang 0 (tác động xung clock cạnh xuống). Ngõ vào nhận xung bên ngoài được lấy mẫu trong suốt khoảng thời gian của mỗi chu kỳ máy., do đó khi xung ở mức cao (1) trong một chu kỳ này và chuyển sang

mức thấp (0) trong một chu kỳ kế thì bộ đếm tăng lên một. Để nhận ra sự chuyển đổi từ 1 sang 0 phải mất 2 chu kỳ máy., nên tần số xung bên ngoài lớn nhất là 500KHz nếu hệ thống vi điều khiển sử dụng dao động thạch anh 12 MHz.

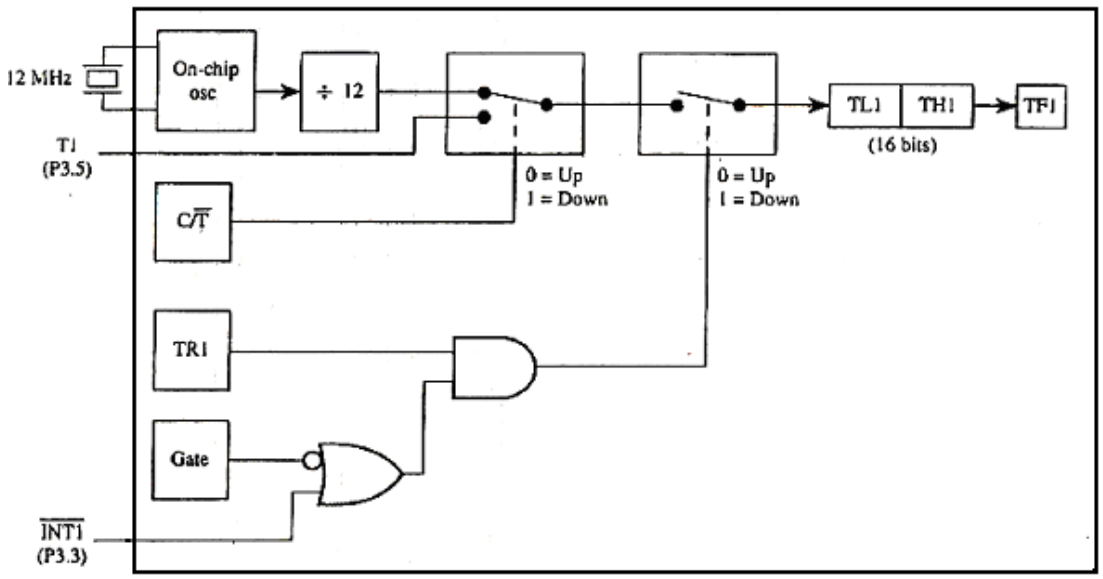
4.5. KHỞI ĐỘNG, DỪNG VÀ ĐIỀU KHIỂN CÁC BỘ TIMER/COUNTER

Bit TRx trong thanh ghi TCON được điều khiển bởi phần mềm để cho phép các Timer/Counter bắt đầu quá trình đếm hoặc ngừng đếm.

Để Timer/Counter bắt đầu hoạt động thì phải thiết lập bit TRx lên bằng 1. Để các Timer/Counter ngừng hoạt động thì phải xóa bit TRx xuống bằng 0.

Bit TRx bị xóa khi Reset hệ thống, do đó ở chế độ mặc định khi mở máy các Timer bị cấm.

Toàn bộ chức năng khởi động, dừng và điều khiển Timer/Counter được trình bày trên hình 4-9.



Hình 4-9. Điều khiển hoạt động của Timer/Counter

Một phương pháp khác để điều khiển các Timer là dùng bit GATE trong thanh ghi TMOD và ngõ nhập bên ngoài INTx. Phương pháp này thường được dùng để đo các độ rộng xung. Giả sử xung cần đo độ rộng đưa vào chân INT0, ta phải khởi tạo Timer 0 hoạt động ở mode 1 là mode Timer 16 bit với giá trị khởi tạo ban đầu là TL0/TH0 = 0000h, bit GATE = 1, bit TR0 = 1. Khi xung đưa đến ngõ vào INT0 = 1 thì "cổng được mở" để cho xung nội có tần số 1MHz vào Timer0. Quá trình Timer 0 đếm xung nội sẽ dừng lại cho đến khi xung đưa đến ngõ vào INT0 xuống mức 0. Thời gian đếm được của Timer 0 chính là độ rộng xung cần đo.

4.6. KHỞI TẠO VÀ TRUY XUẤT CÁC THANH GHI TIMER/COUNTER

Các Timer/Counter thường được khởi tạo 1 lần ở đầu chương trình để thiết lập chế độ hoạt động phục vụ cho các ứng dụng điều khiển liên quan đến định thời hay đếm xung ngoại. Tùy thuộc vào yêu cầu điều khiển cụ thể mà ta điều khiển các Timer bắt đầu đếm, ngừng hay khởi động đếm lại từ đầu …

Thanh ghi TMOD là thanh ghi đầu tiên cần phải khởi tạo để thiết lập mode hoạt động cho các Timer/Counter. Ví dụ khởi động cho Timer0 hoạt động ở mode 1 (mode Timer 16 bit) và hoạt động định thời đếm xung nội bên trong thì ta khởi tạo bằng cách nạp giá trị 00000001b vào thanh ghi TMOD. Trong lệnh này M1 = 0, M0 = 1 để vào mode 1 và C/#T = 0, GATE = 0 để cho phép đếm xung nội bên trong đồng thời xóa các bit mode của Timer 1. Sau lệnh trên Timer0 vẫn chưa đếm và Timer0 chỉ đếm khi bit điều khiển chạy TR0=1.

Nếu ta không thiết lập các giá trị bắt đầu đếm cho các thanh ghi TLx/THx thì Timer sẽ bắt đầu đếm từ 0000h lên và khi chuyển trạng thái từ FFFFh sang 0000h sẽ sinh ra tràn làm cho bit TFx = 1 rồi tiếp tục đếm từ 0000h lên tiếp . . .

Nếu ta thiết lập giá trị bắt đầu đếm cho TLx/THx khác 0000h, thì Timer sẽ bắt đầu đếm từ giá trị thiết lập đó lên nhưng khi chuyển trạng thái từ FFFFh sang 0000h thì Timer lại đếm từ 0000h lên.

Để Timer luôn bắt đầu đếm từ giá trị ta gán thì ta có thể lập trình chờ sau mỗi lần tràn ta sẽ xóa cờ TFx và gán lại giá trị cho TLx/THx để Timer luôn luôn bắt đầu đếm từ giá trị khởi gán lên.

Đặc biệt nếu bộ định thời hoạt động trong phạm vi nhỏ hơn 256µs thì ta nên dùng Timer ở mode 2 (tự động nạp 8 bit). Sau khi khởi tạo giá trị đầu cho thanh ghi THx, và TLx, khi thiết lập bit TRx thì Timer sẽ bắt đầu đếm từ giá trị đã gán trong TLX và khi tràn từ FFh sang 00h trong TLx, thì cờ tràn TFx tự động được thiết lập, đồng thời giá trị trong THx tự động nạp sang cho TLx và Timer bắt đầu đếm từ giá trị khởi gán này lên. Nói cách khác, sau mỗi lần tràn ta không cần khởi gán lại cho các thanh ghi Timer mà chúng vẫn đếm được lại từ giá trị đã gán.

4.7. TIMER/COUNTER 2 CỦA 8052

Họ vi điều khiển 8052 có 3 Timer/Counter: Timer/Counter0, Timer/Counter1 và Timer/Counter2. Các Timer/Counter0 và Timer/Counter1 có các thanh ghi và hoạt động giống như họ 8051. Ở đây chỉ trình bày thêm phần hoạt động của Timer/Counter2.

Các thanh ghi của Timer/Counter2 bao gồm thanh ghi TL2, TH2, thanh ghi điều khiển T2CON, thanh ghi RCAP2L và RCAP2H.

4.7.1. Thanh ghi T2CON

T2CON.7 T2CON.6 T2CON.5 T2CON.4 T2CON.3 T2CON.2 T2CON.1 T2CON.0

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/#T2	CP/#RL2

Hình 4-10. Thanh ghi T2CON

Thanh ghi T2CON là thanh ghi điều khiển Timer/Counter2. Thanh ghi này cho phép truy xuất từng bit. Các bit trong thanh ghi và hoạt động của nó được mô tả trong hình 4-10 và bảng 4-4

Bảng 4-4. Bảng mô tả hoạt động thanh ghi T2CON

Bit	Tên	Mô tả
7	TF2	Cờ tràn Timer2: hoạt động giống như các Timer trên (TF2 sẽ không được thiết lập lên mức 1 nếu bit TCLK hoặc RCLK ở mức 1). TF2 chỉ có thể xóa bằng phần mềm.
6	EXF2	Cờ ngắt ngoài của Timer2: Chỉ được thiết lập khi xảy ra sự thu nhận hoặc nạp lại dữ liệu bởi sự chuyển trạng thái từ 1 sang 0 ở ngõ vào T2EX và EXEN2 = 1. Khi cho phép Timer2 ngắt, EXF2=1 thì CPU sẽ thực hiện hiện chương trình con phục vụ ngắt Timer2. EXF2 chỉ có thể xóa bằng phần mềm.
5	RCLK	Bit clock thu của Timer 2. Khi RCLK=1 thì Timer2 cung cấp tốc độ Baud cho port nối tiếp để nhận dữ liệu về (Khi thu) và Timer T1 sẽ cung cấp tốc độ Baud cho port nối tiếp để phát dữ liệu đi (Khi phát).
4	TCLK	Bit clock phát của Timer 2. Khi TCLK=1 thì Timer2 cung cấp tốc độ Baud cho port nối tiếp để phát dữ liệu đi và Timer1 sẽ cung cấp tốc độ Baud cho port nối tiếp để nhận dữ liệu về.
3	EXEN2	Bit điều khiển cho phép tác động từ bên ngoài. Khi EXEN2 = 1thì hoạt động thu nhận và nạp lại của Timer2 chỉ xảy ra khi ngõ vào T2EX có sự chuyển trạng thái từ 1 sang 0.
2	TR2	Bit điều khiển Timer 1 đếm / ngừng đếm: TR2 = 1 thì Timer2 được phép đếm xung. (Chạy) TR2 = 0 thì time2 không được phép đếm xung (Ngừng). Dùng phần mềm điều khiển bit TR2 để cho phép Timer 2 đếm hay ngừng đếm.

1	C/#T2	Bit lựa chọn Counter hay Timer: C/#T2 = 1 Timer/Counter 2 là bộ đếm C/#T2 = 0 Timer/Counter 2 là bộ định thời
0	CP/#RL2	Cờ thu nhận/nạp lại dữ liệu của Timer T2. Khi bit này = 1 thì thu nhận chỉ xảy ra khi có sự chuyển trạng thái từ 1 sang 0 ở ngõ vào T2EX và EXEN2=1. Khi bit này = 0 thì quá trình tự động nạp lại khi Timer T2 tràn hoặc khi có sự chuyển trạng thái ở ngõ vào T2EX và bit EXEN2 = 1. Nếu bit RCLK hoặc TCLK = 1 thì bit này xem như bỏ.

Ghi chú: Giá trị thanh ghi khi Reset là 00000000b

4.7.2.Thanh ghi T2MOD

T2MOD.7 T2MOD.6 T2MOD.5 T2MOD.4 T2MOD.3 T2MOD.2 T2MOD.1 T2MOD.0

-	-	-	-	-	-	**T20E**	**DCEN**

Hình 4-11. Thanh ghi T2MOD

Thanh ghi T2MOD là thanh ghi không cho phép truy xuất từng bit. Thanh ghi này đượ mô tả trên hình 4-11 và bảng 4-5

Bảng 4-5. Bảng mô tả hoạt động thanh ghi T2MOD

Bit	Tên	Mô tả
7	-	Không sử dụng
6	-	Không sử dụng
5	-	Không sử dụng
4	-	Không sử dụng
3	-	Không sử dụng
2	-	Không sử dụng
1	T2OE	Cho phép đầu ra khi sử dụng Timer 2 để tạo xung (Chế độ tạo xung – Clock out)
0	DCEN	Bit cho phép Timer 2 hoạt động như một bộ đếm tiến lùi

Ghi chú: Giá trị thanh ghi khi Reset là xxxxxx00b

4.7.3. Thanh ghi TH2 và TL2; RCAP2H và RCAP2L

Cũng giống như các thanh ghi TH0, TL0, TH1, TL1 thanh ghi TH2 và TL2 chứa giá trị đếm của Timer2, tuy nhiên khác nhau là Timer0 và Timer1 dùng các thanh ghi THx để chứa giá trị nạp lại trong chế độ tự động nạp lại còn Timer2 dùng RCAP2H và RCAP2L để chứa giá trị cần nạp lại.

4.7.4. Các chế độ hoạt động của Timer/Counter2

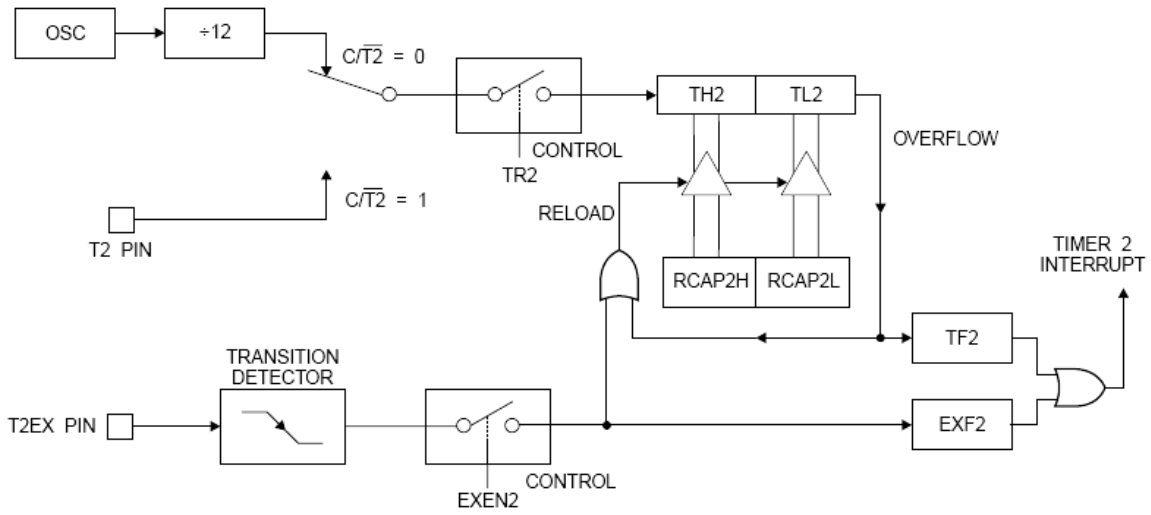
Timer/Counter2 có thể dùng để định thời hoặc dùng như bộ đếm để đếm xung ngoài đưa đến ngõ vào T2 chính là chân P1.0.

Timer/Counter2 có 3 chế độ hoạt động: Tự động nạp lại, thu nhận và thiết lập tốc độ Baud để phục vụ cho truyền dữ liệu. Cấu hình các bit để chọn chế độ hoạt động được trình bày trong bảng 4-6.

Bảng 4-6. Bảng thiết lập chế độ hoạt động của Timer/Counter2

RCLK + TCLK	CP/#RL2	TR2	Chế độ
0	0	1	16 bit tự động nạp lại
0	1	1	16 bit thu nhận
1	x	1	Cung cấp tốc độ Baud
X	x	0	Off

a. Chế độ tự động nạp lại



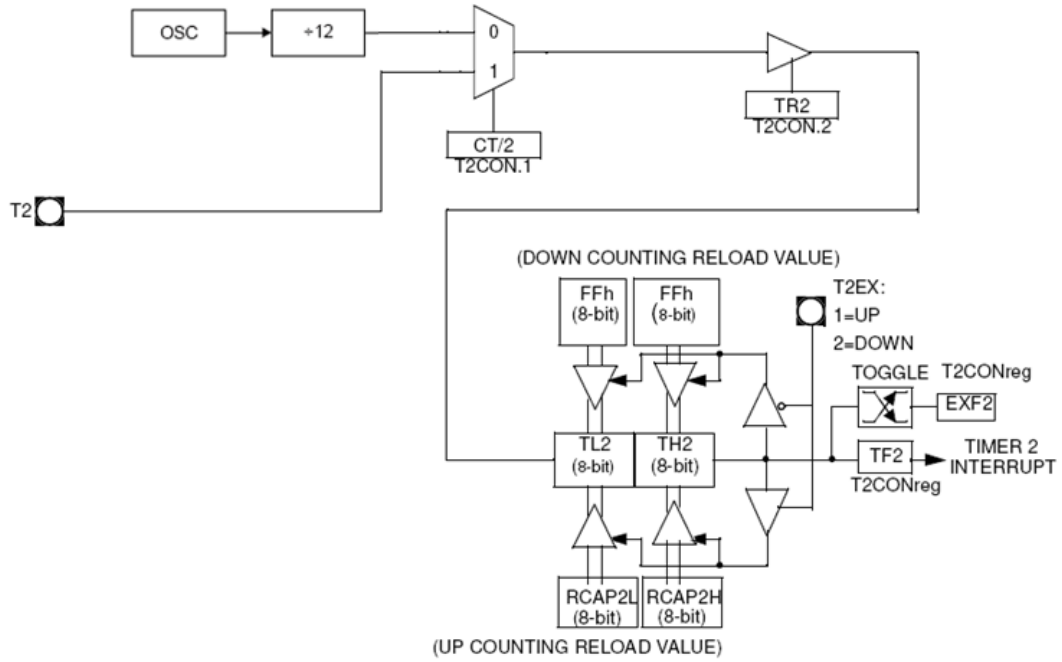
Hình 4-12. Timer2 ở chế độ tự động nạp lại (DCEN=0)

- Nếu DCEN=0 thì Timer 2 hoạt động như một Timer 16 bit tự động nạp lại. Giá trị nạp lại được chứa trong RCAP2H và RCAP2L (Hình 4-12). Sự kiện nạp lại xảy ra khi:

+ Xảy ra tràn tức là chuyển số đếm từ FFFFh sang 0000h.

+ Có sự chuyển từ mức 1 xuống mức 0 (cạnh xuống) trên chân T2EX khi EXEN2 đã được thiết lập bằng 1. Sự chuyển mức này cũng đồng thời thiết lập

EXF2=1. Tương tự như cờ TF2 thì cờ EXF2 cũng có thể được kiểm tra bằng phần mềm hoặc tạo ngắt. Bit EXF2 phải xóa bằng phần mềm.



Hình 4-13. Timer2 ở chế độ tự động nạp lại (DCEN=1)

- Nếu DCEN=1 thì Timer 2 vẫn hoạt động như một Timer 16 bit tự động nạp lại nhưng có hai cách tự động nạp lại:

+ Khi T2EX=1 thì Timer 2 sẽ đếm tiến từ giá trị xuất phát cho đến khi có sự kiện chuyển số đếm từ FFFFh sang 0000h thì xảy ra tràn. Sự kiện tràn sẽ thiết lập cờ TF2 đồng thời nạp giá trị chứa trong RCAP2H và RCAP2L vào Timer2.

+ Khi T2EX=0 thì Timer 2 sẽ đếm lùi từ giá trị xuất phát cho đến giá trị chứa trong RCAP2H và RCAP2L thì xảy ra tràn. Sự kiện tràn sẽ thiết lập cờ TF2 đồng thời nạp giá trị FFFFh vào Timer (Hình 4-13).

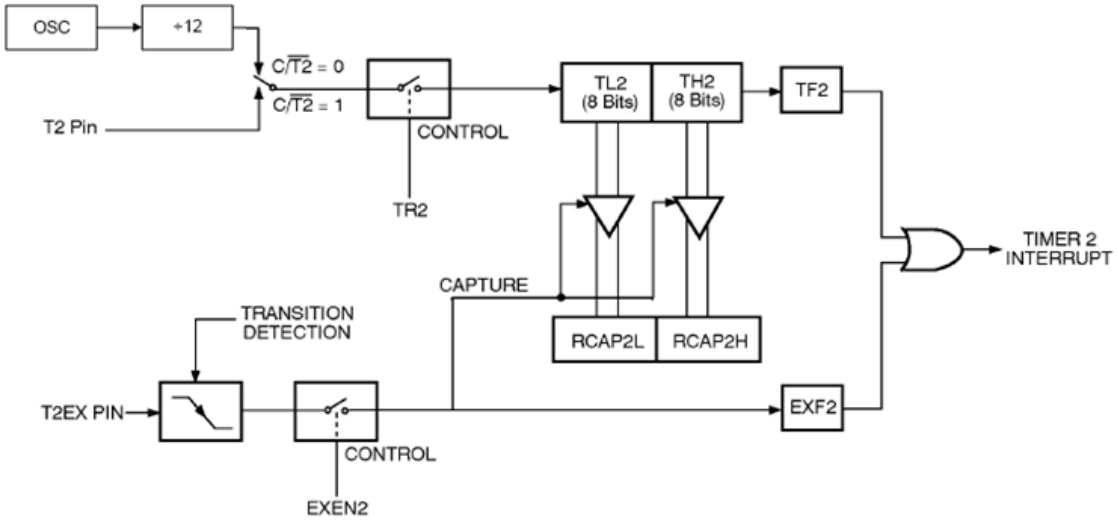
b. Chế độ thu nhận

Khi CP/#RL2=1 thì Timer 2 hoạt động ở chế độ thu nhận. Khi đó Timer 2 hoạt động bình thường như một Timer/Counter 16 bit, thanh ghi TL2, TH2 sẽ lưu trữ xung đếm và nếu có sự chuyển trạng thái từ FFFFh sang 0000h thì sẽ sinh ra tràn và làm cho cờ tràn TF2=1. Bit cờ tràn có thể kiểm tra bằng phần mềm hay có thể tạo ra ngắt.

Để cho phép chế độ thu nhận hoạt động thì làm cho bit EXEN2 = 1. Nếu bit EXEN2 = 1 và khi có sự chuyển trạng trái từ 1 sang 0 ở ngõ vào T2EX thì chế độ thu nhận sẽ xảy ra: giá trị đếm được trong thanh ghi TL2, TH2 sẽ được chuyển sang 2

thanh ghi RCAP2L và RCAP2H. Cờ EXF2 cũng được chuyển lên mức 1 để báo hiệu quá trình thu nhận đã xảy ra, cờ EXF2 có thể kiểm tra bằng phần mềm hoặc tạo ngắt.

Hoạt động thu nhận dữ liệu của Timer T2 được trình bày ở hình 4-14.



Hình 4-14. Timer2 ở chế độ thu nhận

c. Chế độ cung cấp tốc độ Baud

Timer 2 có thể được dùng vào việc tạo tốc độ Baud cho đường truyền và đường nhận dữ liệu cho cổng nối tiếp tùy thuộc vào bit TCLK và RCLK trong thanh ghi T2CON. Chế độ này được đề cập cụ thể trong chương 5.

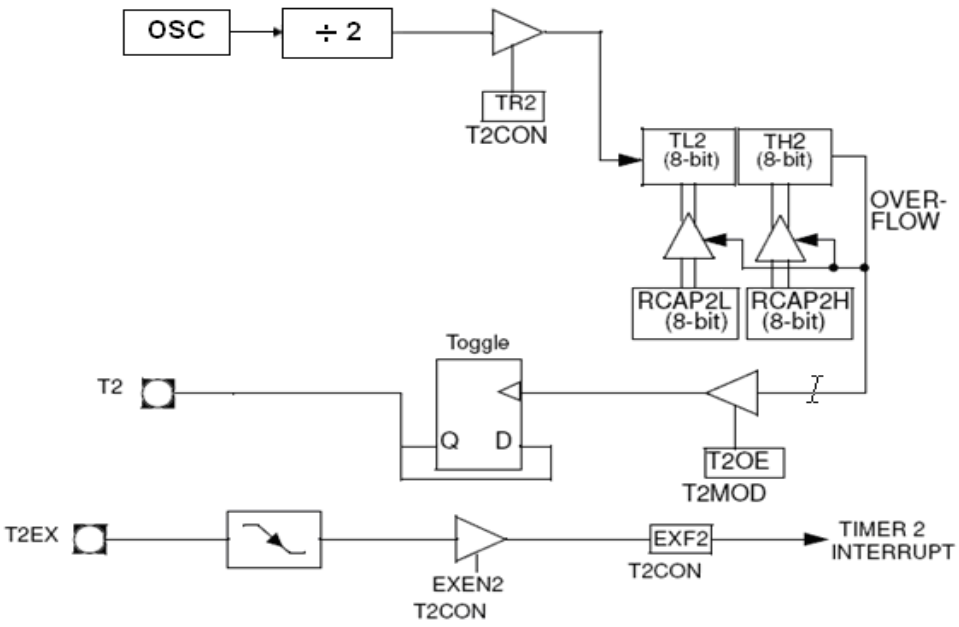
4.7.5. Sử dụng Timer 2 để tạo xung vuông trên chân P1.1

Có thể sử dụng Timer 2 để tạo xung vuông (Có tỷ số độ rộng xung 50%) ở chân P1.1 được minh họa trong hình 4-15. Khi đó bit C/#T2 phải được xóa, bit T2OE phải được thiết lập còn bit TR2 được dùng điều khiển chạy/dừng Timer 2 tức là tạo/dừng xung vuông trên P1.0.

Tần số xung tạo ra được tính bằng công thức

$$F_{P1.0}=F_{OSC} / (4 * (65536 - (RCAP2H*256 + RCAP2L)))$$

Nếu sử dụng thạch anh có tần số 12MHz thì tùy thuộc vào giá trị nạp vào thanh ghi RCAP2L và RCAP2H mà có thể tạo ra xung vuông trên chân P1.0 có tần số từ 46Hz đến 3MHz.



Hình 4-15. Timer2 ở chế độ tạo xung vuông trên P1.0

Chương 5

CỔNG NỐI TIẾP

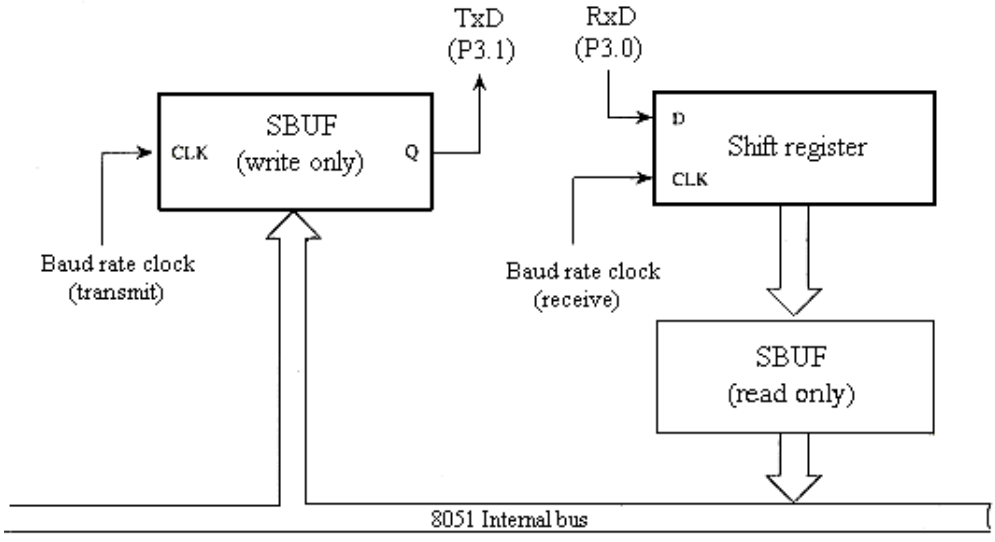
5.1. GIỚI THIỆU

Truyền dữ liệu nối tiếp của 8051 có thể hoạt động ở nhiều mode riêng biệt trong phạm vi cho phép của tần số. Chức năng cơ bản của cổng nối tiếp là dữ liệu dạng song song được chuyển thành nối tiếp để truyền đi và dữ liệu nhận về dạng nối tiếp được chuyển thành song song.

Chân TxD (P3.1) là ngõ xuất dữ liệu đi và chân RxD (P3.0) là ngõ nhận dữ liệu về.

Đặc trưng của truyền dữ liệu nối tiếp là hoạt động song công có nghĩa là có thể thực hiện truyền và nhận dữ liệu cùng một lúc. Ngoài ra cổng nối tiếp còn có một đặc trưng khác, việc đệm dữ liệu khi thu của cổng này cho phép một ký tự được nhận và lưu giữ trong bộ đệm thu trong khi ký tự tiếp theo được nhận vào. Nếu CPU đọc ký tự thứ nhất trước khi ký tự thứ hai được nhận đầy đủ thì dữ liệu không bị mất.

Hai thanh ghi chức năng đặc biệt phục vụ cho truyền dữ liệu là thanh ghi đệm SBUF và thanh ghi điều khiển SCON. Thanh ghi đệm SBUF nằm ở địa chỉ 99H có 2 chức năng: nếu vi điều khiển ghi dữ liệu lên thanh ghi SBUF thì dữ liệu đó sẽ được truyền đi, nếu hệ thống khác gửi dữ liệu đến thì sẽ được lưu vào thanh ghi đệm SBUF. Sơ đồ khối của hệ thống truyền dữ liệu như hình 5-1.



Hình 5-1. Sơ đồ khối của cổng nối tiếp

Thanh ghi điều khiển truyền dữ liệu SCON nằm ở địa chỉ 98H là thanh ghi cho phép truy xuất bit bao gồm các bit trạng thái và các bit điều khiển. Các bit điều khiển

dùng để thiết lập nhiều mode hoạt động truyền dữ liệu khác nhau, còn các bit trạng thái cho biết thời điểm kết thúc khi truyền xong một kí tự hoặc nhận xong một kí tự. Các bit trạng thái có thể được kiểm tra trong chương trình hoặc có thể lập trình để sinh ra ngắt.

Tần số hoạt động của truyền dữ liệu nối tiếp còn gọi tốc độ Baud (số lượng bit dữ liệu được truyền đi trong một giây) có thể hoạt động cố định (sử dụng dao động trên chip) hoặc có thể thay đổi. Khi cần tốc độ Baud thay đổi thì phải sử dụng Timer 1 để tạo tốc độ Baud. Trên 8052 bộ Timer2 cũng có thể được lập trình để tạo tốc độ Baud.

5.2. THANH GHI ĐIỀU KHIỂN TRUYỀN DỮ LIỆU NỐI TIẾP SCON

SCON.7	SCON.6	SCON.5	SCON.4	SCON.3	SCON.2	SCON.1	SCON.0
SM0	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**

Hình 5-2. Thanh ghi SCON

Thanh ghi SCON sẽ thiết lập các mode hoạt động truyền dữ liệu khác nhau cho 8051. Bảng 5-1 tóm tắt thanh ghi điều khiển cổng nối tiếp SCON.

Bảng 5-1. Thanh ghi điều khiển cổng nối tiếp

Bit	Ký hiệu	Mô tả hoạt động
7	SM0	Bit chọn mode truyền nối tiếp: bit thứ 0.
6	SM1	Bit chọn mode truyền nối tiếp: bit thứ 1.
5	SM2	Bit cho phép truyền kết nối nhiều vi xử lý ở mode 2 và 3; RI sẽ không tích cực nếu bit thứ 9 đã thu vào là 0.
4	REN	Bit cho phép nhận kí tự. REN = 1 sẽ cho phép nhận kí tự.
3	TB8	Dùng lưu bit 9 để tryền đi khi hoạt động ở mode 2 và 3. TB8 bằng 0 hay 1 là do người lập trình thiết lập.
2	RB8	Dùng để lưu bit 9 nhận về khi hoạt động ở mode 2 và 3.
1	TI	Cờ báo hiệu này lên mức 1 khi truyền xong 1 kí tự và xóa bởi người lập trình để sẵn sàng truyền kí tự tiếp theo.
0	RI	Cờ báo hiệu này lên mức 1 khi nhận xong 1 kí tự và xóa bởi người lập trình để sẵn sàng nhận kí tự dữ liệu tiếp theo.

Ghi chú: Giá trị thanh ghi khi Reset là 0000000b

5.3. CÁC MODE TRUYỀN DỮ LIỆU NỐI TIẾP

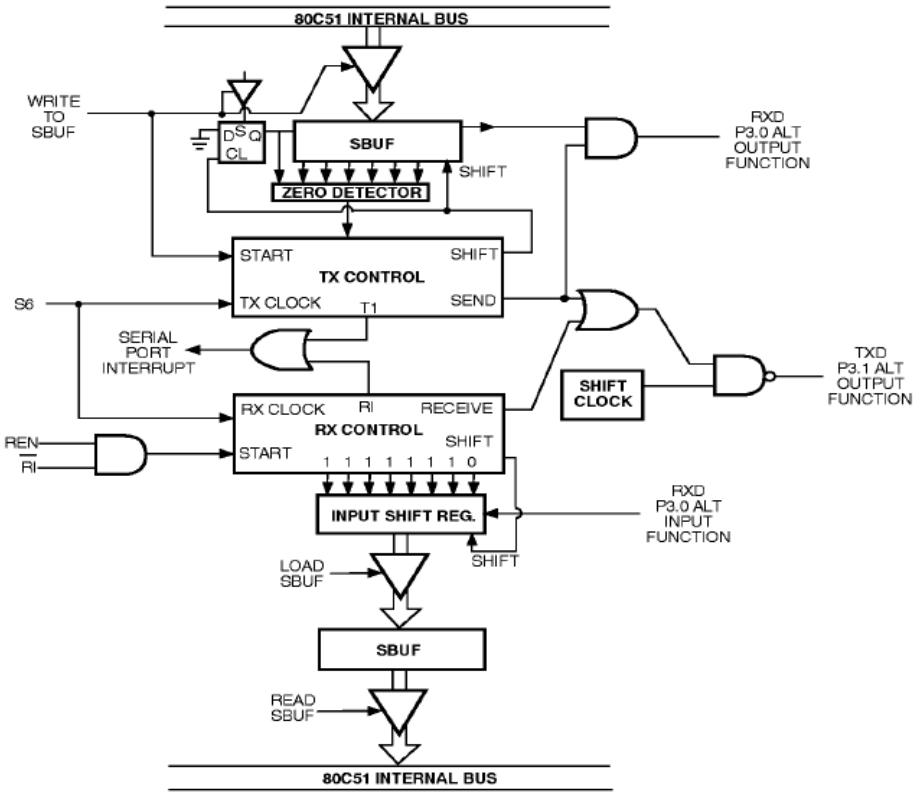
Trước khi truyền dữ liệu thì thanh ghi SCON phải được khởi tạo đúng mode. Ví dụ để khởi tạo truyền dữ liệu mode 1 thì 2 bit: SM0 SM1 = 01, bit cho phép thu: REN =1, và cờ ngắt truyền TI = 0 để sẵn sàng truyền (TI=1 chỉ thực hiện bởi phần cứng, TI=0 thực hiện bởi phần mềm).

Truyền dữ liệu nối tiếp của 8051 có 4 mode hoạt động tùy thuộc theo 4 trạng thái của 2 bit SM0 SM1 được liệt kê ở bảng 5.2. Ba trong bốn mode cho phép truyền bất đồng bộ với mỗi kí tự thu hoặc phát sẽ được kết hợp với bit start hoặc bit stop.

Bảng 5-2. Các mode truyền dữ liệu

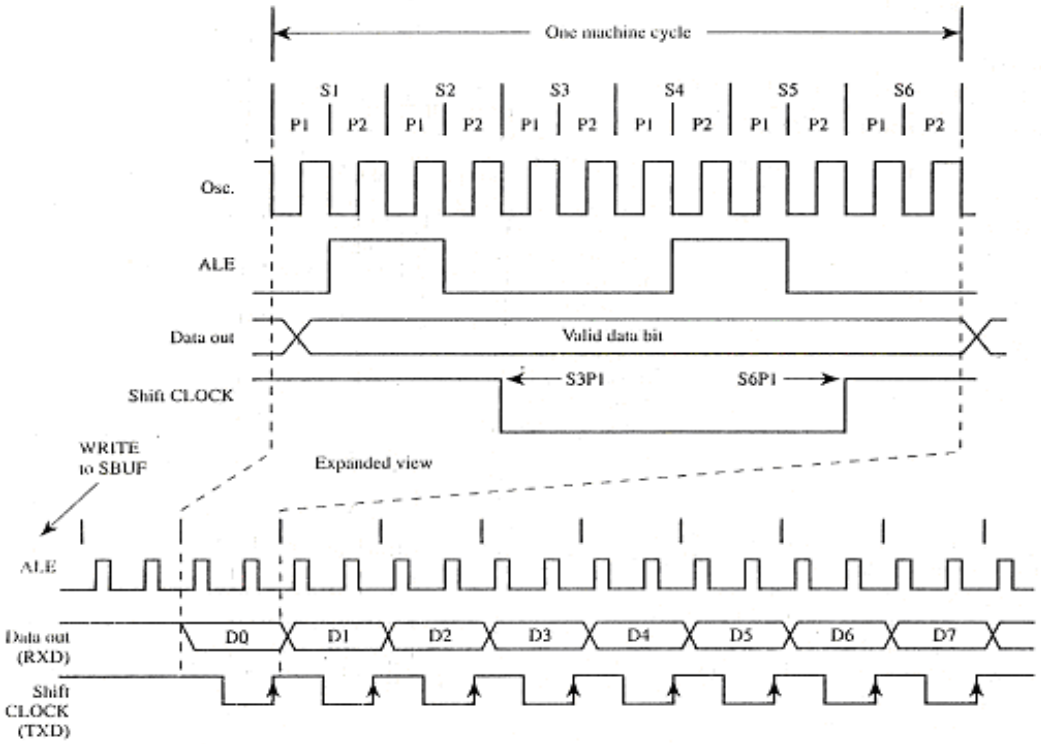
SM0	SM1	Mode	Mô tả	Tốc độ Baud
0	0	0	Thanh ghi dịch	Cố định (tần số f/12).
0	1	1	UART 8 bit	Thay đổi (được thiết lập bởiTimer).
1	0	2	UART 9 bit	Cố định (tần số f/32 hoặc f/64)
1	1	3	UART 9 bit	Thay đổi (được thiết lập bởi Timer).

5.3.1. Chế độ thanh ghi dịch (Mode 0)



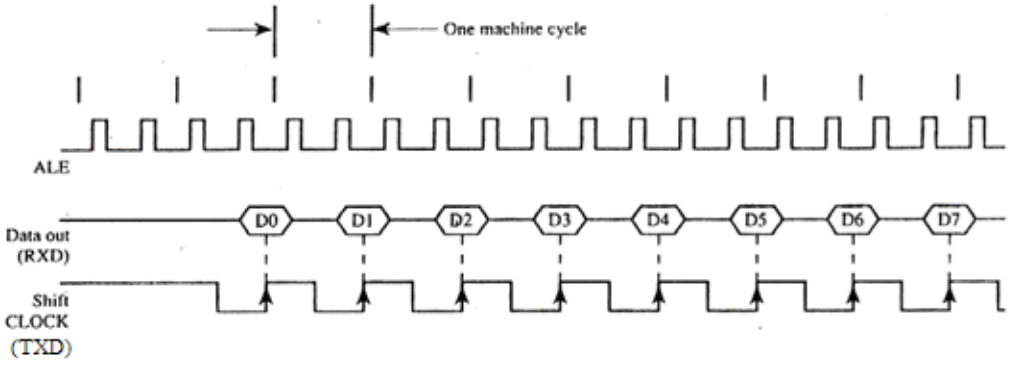
Hình 5-3. Cổng nối tiếp khi thu phát dữ liệu ở mode 0

Để định cấu hình cho truyền dữ liệu nối tiếp ở mode 0 thì 2 bit SM0 SM1 = 00. Dữ liệu nối tiếp nhận vào và dữ liệu truyền đi đều thông qua chân RxD, còn chân TxD thì dùng để dịch chuyển xung clock (Hình 5-3). 8 bit dữ liệu để truyền đi hoặc nhận về thì luôn bắt đầu với bit có trọng số nhỏ nhất LSB. Tốc độ Baud được thiết lập cố định ở tần số bằng 1/12 tần số dao động thạch anh trên chip.



Hình 5-4. Giản đồ thời gian phát dữ liệu ở mode 0

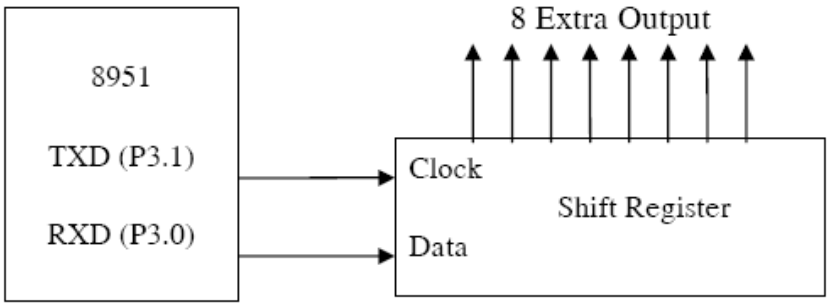
Khi thực hiện lệnh ghi dữ liệu lên thanh ghi SBUF thì quá trình truyền dữ liệu bắt đầu. Dữ liệu được dịch ra ngoài thông qua chân RxD cùng với các xung nhịp cũng được gửi ra ngoài thông qua chân TxD. Mỗi bit truyền đi chỉ có xuất hiện trên chân RxD trong khoảng thời gian một chu kỳ máy. Trong khoảng thời gian của mỗi chu kỳ máy., tín hiệu xung clock xuống mức thấp tại thời điểm S3P1 và lên mức cao tại thời điểm S6P1 trong giản đồ thời gian hình 5-4.



Hình 5-5. Giản đồ thời gian thu dữ liệu ở mode 0

Quá trình nhận được khởi động khi bit cho phép nhận REN = 1 và cờ nhận RI = 0. Nguyên tắc chung là khởi tạo bit REN = 1 ở đầu chương trình để khởi động truyền dữ liệu, và xóa bit RI để sẵn sàng nhận dữ liệu vào. Khi bit RI bị xóa, các xung clock sẽ xuất ra bên ngoài thông qua chân TxD, bắt đầu chu kỳ máy. kế tiếp thì dữ liệu từ bên ngoài sẽ được dịch vào bên trong thông qua chân RxD (hình 5-5).

Một ứng dụng cụ thể sử dụng mode 0 là dùng để mở rộng thêm số lượng ngõ ra cho 8051 với cách thức thực hiện như sau: một thanh ghi dịch từ nối tiếp thành song song được nối đến các đường TxD và RxD của 8051 để mở rộng thêm 8 đường ra như hình 5-6. Nếu dùng thêm nhiều thanh ghi dịch mắc nối tiếp vào thanh ghi dịch đầu tiên sẽ mở rộng được nhiều ngõ ra.



Hình 5-6. Một ứng dụng mode 0 để tăng thêm ngõ ra bằng thanh ghi dịch.

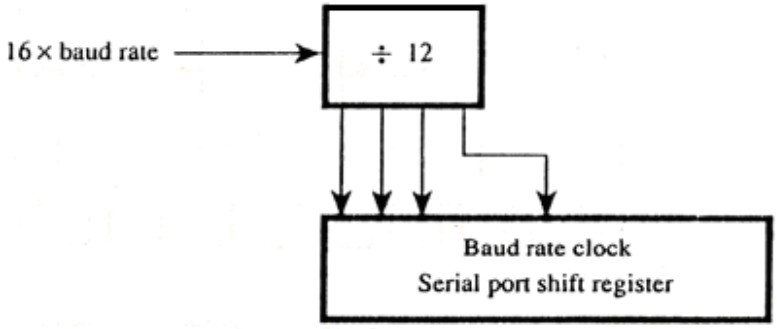
5.3.2. Thu phát bất đồng bộ 8 bit với tốc độ Baud thay đổi (Mode 1)

Trong mode này, truyền dữ liệu nối tiếp hoạt động bất đồng bộ UART (Universal Asynchronous Receiver Transmitter) 8 bit có tốc độ Baud thay đổi được. UART là bộ thu và phát dữ liệu nối tiếp với mỗi ký tự dữ liệu luôn bắt đầu bằng 1 bit start (ở mức 0) và kết thúc bằng 1 bit stop (ở mức 1), bit parity đôi khi được ghép vào giữa bit dữ liệu sau cùng và bit stop.

Trong mode này, 10 bit dữ liệu sẽ phát đi ở chân TxD và nếu nhận thì sẽ nhận ở chân RxD. 10 bit đó bao gồm: 1 bit start, 8 bit data (LSB là bit đầu tiên), và 1 bit stop. Đối với hoạt động nhận dữ liệu thì bit Stop được đưa vào bit RB8 trong thanh ghi SCON.

Trong 8051, tốc độ Baud được thiết lập bởi tốc độ tràn của Timer1. Đối với họ 8052 có 3 Timer thì tốc độ Baud có thể thiết lập bởi tốc độ tràn của Timer1 hoặc Timer2 hoặc cả 2 Timer1 và Timer2: một Timer cho máy phát và 1 Timer cho máy thu.

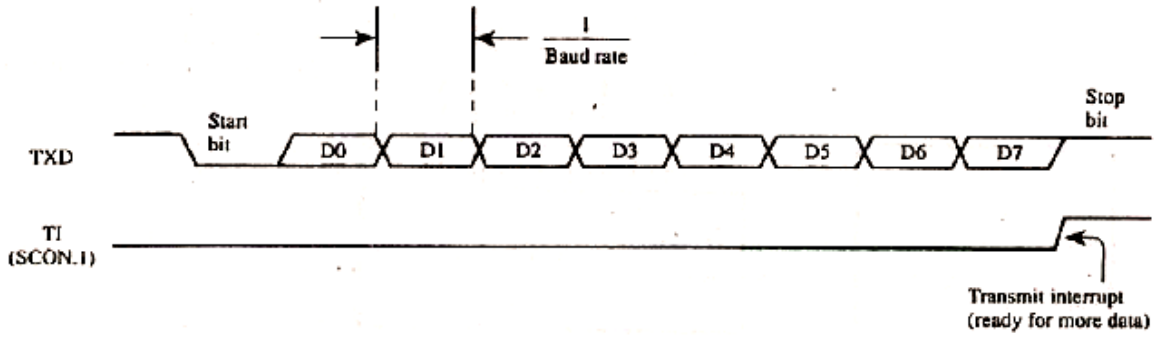
Nguồn cung cấp xung clock để đồng bộ các thanh ghi truyền dữ liệu nối tiếp hoạt động ở mode 1, 2, 3 được thiết lập bởi bộ đếm 16 như hình 5-7, ngõ ra của bộ đếm là xung clock tạo tốc độ Baud. Xung ngõ vào của bộ đếm có thể lập trình bằng phần mềm.



Hình 5-7. Cung cấp xung cho truyền dữ liệu nối tiếp.

Khi có một lệnh ghi dữ liệu lên thanh ghi SBUF thì quá trình truyền dữ liệu bắt đầu nhưng nó chưa truyền mà chờ cho đến khi bộ chia 16 (cung cấp tốc độ Baud cho truyền dữ liệu nối tiếp) bị tràn. Dữ liệu được xuất ra trên chân TxD bắt đầu với bit start theo sau là 8 bit data và sau cùng là bit stop. Các cờ phát TI được nâng lên mức 1 cùng lúc với thời điểm xuất hiện bit Stop trên chân TxD như hình 5-8.

Quá trình nhận dữ liệu được khởi động khi có sự chuyển đổi từ mức 1 sang mức 0 ở ngõ vào RxD. Bộ đếm 4 bit được reset ngay lập tức để sắp xếp bit dữ liệu đang đến từ ngõ vào RxD. Mỗi bit dữ liệu đến được lấy mẫu ở trạng thái đếm thứ 8 trong một chu kỳ 16 trạng thái của bộ đếm 4 bit.



Hình 5-8. Cờ báo phát xong dữ liệu TI

Khi có sự chuyển trạng thái từ 1 xuống 0 ở ngõ vào RxD của bộ thu thì trạng thái 0 này phải tồn tại trong 8 trạng thái liên tục của bộ đếm 4 bit. Nếu trường hợp này không đúng thì bộ thu xem như bị tác động bởi tín hiệu nhiễu. Bộ thu sẽ reset và trở về trạng thái nghỉ và chờ sự chuyển trạng thái tiếp theo.

Giả sử việc kiểm tra bit start là hợp lệ thì bit start sẽ được bỏ qua và 8 bit dữ liệu được nhận vào thanh ghi dịch nối tiếp.

Khi tất cả 8 bit được ghi vào thanh ghi dịch thì 3 công việc sau sẽ được thực hiện tiếp theo:

- Bit thứ 9 (bit stop) được dịch vào bit RB8 trong SCON.
- 8 bit data được nạp vào thanh ghi SBUF.

- Cờ ngắt nhận RI =1.

Tuy nhiên, 3 công việc trên chỉ xảy ra nếu hai điều kiện sau tồn tại:

• RI = 0

• SM2 = 1 và bit Stop nhận được = 1 hoặc SM2 = 0.

5.3.3. Thu phát bất đồng bộ 9 bit có tốc độ Baud cố định (Mode 2)

Khi SM0 SM1 = 10 thì truyền dữ liệu hoạt động ở mode 2 có tốc độ Baud cố định. Có 11 bit được phát hoặc thu: 1 bit Start, 8 bit data, 1 bit data thứ 9 được lập trình và 1 bit Stop. Khi phát thì bit thứ 9 được đặt vào TB8 của SCON (có thể bit parity). Khi thu thì bit thứ 9 được đặt vào bit RB8 của thanh ghi SCON. Tốc độ Baud trong mode 2 bằng 1/32 hoặc 1/64 tần số dao động thạch anh.

5.3.4. Thu phát bất đồng bộ 9 bit có tốc độ Baud thay đổi (Mode 3)

Khi SM0 SM1 = 11 thì truyền dữ liệu hoạt động ở mode 3 là mode UART 9 bit có tốc độ Baud thay đổi. Mode 3 tương tự mode 2 ngoại trừ tốc độ Baud được lập trình và được cung cấp bởi Timer. Các mode 1, mode 2 và mode 3 rất giống nhau, những điểm khác nhau là ở tốc độ Baud (mode 2 cố định, mode 1 và mode 3 thay đổi) và số bit dữ liệu (mode 1 có 8 bit, mode 2 và mode 3 có 9 bit data).

5.4. KHỞI TẠO VÀ TRUY XUẤT CÁC THANH GHI TRUYỀN DỮ LIỆU

5.4.1. Bit cho phép thu (Receive Enable).

Để cho phép thu dữ liệu thì chương trình phải làm cho bit REN = 1 và điều này được thực hiện ở đầu chương trình.

5.4.2. Bit dữ liệu thứ 9.

Bit dữ liệu thứ 9 được phát trong mode 2 và mode 3 phải được nạp vào bit TB8 bằng phần mềm có nghĩa là người lập trình phải thực hiện công việc này trước khi truyền dữ liệu đi, còn bit dữ liệu thứ 9 của dữ liệu thu được thì tự động đặt vào trong bit RB8.

Phần mềm có thể hoặc không đòi hỏi bit dữ liệu thứ 9 tham gia vào quá trình truyền dữ liệu tùy thuộc vào đặc tính của các thiết bị nối tiếp kết nối với nhau thiết lập ra qui định. Bit dữ liệu thứ 9 đóng vai trò quan trọng trong truyền thông nhiều vi xử lý.

5.4.3. Bit kiểm tra chẵn lẻ Parity

Bit thứ 9 thường được dùng là bit kiểm tra chẵn lẻ. Ở mỗi chu kỳ máy., bit P trong thanh ghi trạng thái PSW bằng 1 hay bằng 0 tùy thuộc vào quá trình kiểm tra chẵn 8 bit dữ liệu chứa trong thang ghi A.

Ví dụ nếu hệ thống truyền dữ liệu yêu cầu 8 bit data cộng thêm 1 bit kiểm tra chẵn, thì các lệnh sau đây sẽ phát 8 bit trong thanh ghi A cộng với bit kiểm tra chẵn được cộng vào bit thứ 9.

Trong mode 1 ta vẫn có thể sử dụng bit kiểm tra chẵn lẻ như sau: 8 bit data được phát trong mode 1 có thể bao gồm 7 bit dữ liệu, và 1 bit kiểm tra chẵn lẻ. Để phát một mã ASCII 7 bit với 1 bit kiểm tra chẵn vào 8 bit.

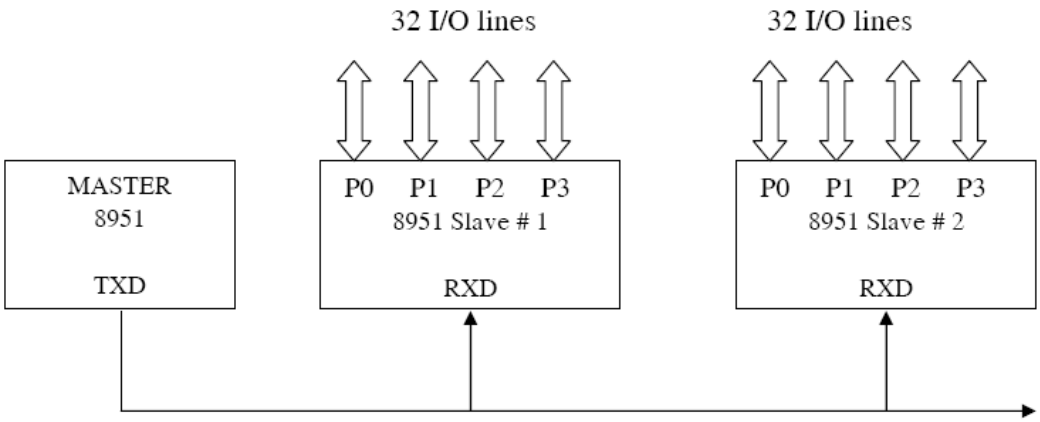
5.4.4. Các cờ ngắt

Cờ ngắt nhận RI và phát TI trong thanh ghi SCON đóng một vai trò quan trọng trong truyền dữ liệu của 8051. Cả hai bit đều được thiết lập bởi phần cứng nhưng phải xóa bởi phần mềm.

Cờ RI được thiết lập ở mức 1 khi kết thúc quá trình nhận đầy đủ 1 kí tự và cho biết thanh ghi đệm thu đã đầy. Trạng thái của cờ RI có thể kiểm tra bằng phần mềm hoặc có thể lập trình để sinh ra ngắt. Nếu muốn nhận một ký tự từ một thiết bị đã được kết nối đến Port nối tiếp, thì chương trình phải chờ cho đến khi cờ RI = 1, sau đó xóa cờ RI và đọc ký tự từ thanh ghi SBUF.

Cờ TI lên mức 1 cho biết đã phát xong ký tự và cho biết thanh ghi đệm SBUF đã rỗng. Nếu muốn gởi 1 ký tự đến một thiết bị đã được kết nối đến cổng nối tiếp thì trước tiên phải kiểm tra xem cổng nối tiếp đã sẵn sàng chưa. Nếu ký tự trước đang được gửi đi, thì phải chờ cho đến khi kết thúc quá trình gửi.

5.5. TRUYỀN THÔNG ĐA XỬ LÝ



Hình 5-9. Kết nối nhiều vi xử lý.

Mode 2 và mode 3 có một chức năng đặc biệt cho việc truyền thông đa xử lý. Ở các mode 2 và 3, 9 bit dữ liệu được thu và bit thứ 9 được lưu vào bit RB8. Truyền dữ liệu có thể lập trình sao cho khi thu được bit Stop thì ngắt của truyền dữ liệu nối tiếp tác động chỉ khi bit RB8 =1. Cấu trúc này được phép bởi bằng cách thiết lập SM2 = 1

trong thanh ghi SCON. Mode này được ứng dụng trong mạng sử dụng nhiều 8051 được tổ chức theo cấu hình máy chủ và máy tớ (hình 5-9).

Trong cấu hình kết nối ở trên thì mỗi một vi xử lý tớ sẽ có một địa chỉ duy nhất do chúng ta qui định.

Khi bộ xử lý chủ muốn phát một khối dữ liệu đến một trong các bộ xử lý tớ thì trước tiên vi xử lý chủ phải gửi ra 1 byte địa chỉ để nhận diện bộ xử lý tớ muốn kết nối.

Byte địa chỉ được phân biệt với byte dữ liệu bởi bit thứ 9: trong byte địa chỉ thì bit thứ 9 bằng 1 và trong byte dữ liệu thì bit thứ 9 bằng 0.

Các vi xử lý tớ sau khi nhận được byte địa chỉ sẽ biết được vi xử lý chủ muốn giao tiếp tớ nào. Khi có vi xử lý tớ được phép thì nó sẽ xóa bit SM2 để bắt đầu nhận các byte dữ liệu tiếp theo. Còn các vi xử lý không được phép thì vẫn giữ nguyên bit SM2=1 để không nhận các byte dữ liệu truyền giữa vi xử lý chủ và vi xử lý tớ đang được phép. Vi xử lý tớ sau khi kết nối với vi xử lý chủ xong thì phải làm cho bit SM2=1 để sẵn sàng kết nối cho những lần tiếp theo.

Sau khi thực hiện xong việc trao đổi dữ liệu thì vi xử lý muốn truy xuất một vi xử lý khác thì phải tạo ra một địa chỉ mới và vi xử lý tớ tương ứng với địa chỉ đó được phép và hoạt động giống như vừa trình bày.

5.6. TỐC ĐỘ BAUD CỦA CỔNG NỐI TIẾP

Truyền dữ liệu nối tiếp nếu hoạt động ở mode 0 và mode 2 thì có tốc độ truyền cố định. Trong mode 0 thì tốc độ truyền bằng 1/12 tần số dao động trên Chip. Nếu sử dụng thạch anh 12 MHz thì tốc độ truyền của mode 0 là 1MHz như hình 5-10a.

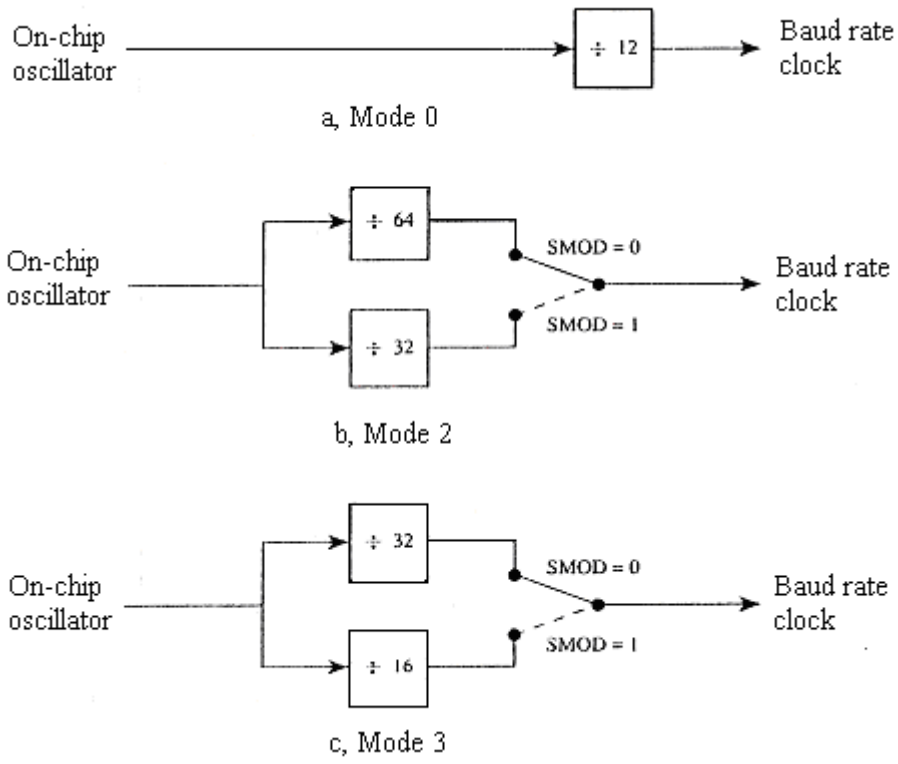
Trong thanh ghi PCON có một bit SMOD có chức năng làm tăng gấp đôi tốc độ Baud, mặc nhiên sau khi reset hệ thống thì bit SMOD = 0 thì các mode truyền dữ liệu hoạt động với tốc độ qui định, khi bit SMOD = 1 thì tốc độ tăng gấp đôi.

Ví dụ trong mode 2, tốc độ truyền có thể tăng gấp đôi từ giá trị mặc định 1/64 tần số dao động trên Chip (SMOD = 0) lên đến 1/32 tần số dao động trên Chip (ứng với SMOD =1) như hình 5-10b.

Các tốc độ Baud trong mode 1 và mode 3 của 8051 được xác định bởi tốc độ tràn của Timer 1. Bởi vì Timer hoạt động ở tần số tương đối cao nên phải chia cho 32 khi bit SMOD = 0 và chia cho 16 nếu SMOD = 1 trước khi cung cấp xung clock để thiết lập tốc độ Baud cho port nối tiếp. Tốc độ Baud ở mode 1 và 3 của 8051 được xác định bởi tốc độ tràn của Timer 1 hoặc Timer 2, hoặc cả 2 như hình 5-10c.

Muốn có tốc độ Baud thì ta khởi tạo thanh ghi TMOD ở mode tự động nạp 8 bit (mode 2) và đặt giá trị nạp lại vào thanh ghi TH1 của Timer 1 để tạo ra tốc độ tràn

chính xác để thiết lập tốc độ Baud. Thanh ghi TMOD được khởi tạo để thiết lập tốc độ Baud bằng cách nạp giá trị 0010xxxxb vào nó.



Hình 5-10. Thiết lập tốc độ Baud

Một cách khác để tạo tốc độ Baud là nhận tín hiệu xung clock từ bên ngoài đưa đến ngõ vào T1. Công thức chung để xác định tốc độ Baud trong mode 1 và mode 3 là:

BAUD RATE = TIMER 1 OVERFLOW RATE ÷ 32

Ví dụ: Truyền dữ liệu cần tốc độ Baud là 1200 thì ta tính toán như sau:

Tốc độ tràn của Timer 1 bằng 1200 x 32 = 38,4KHz. Nếu hệ thống sử dụng thạch anh 12 MHz thì xung cung cấp cho Timer1 đếm có tần số là 1 MHz hay 1000KHz. Vậy để đạt tốc độ tràn 38,4 KHz thì ta tính được số lượng xung đếm cho mỗi chu kỳ tràn là 1000 KHZ/38,4 KHz = 26,4 xung (làm tròn bằng 26).

Do các Timer đếm lên và thời điểm tràn xảy ra khi chuyển trạng thái đếm từ FFh → 00h nên ta phải nạp giá trị bắt đầu từ (256 – 26 = 230) để từ giá trị này Timer 1 đếm lên 26 xung nữa thì sinh ra tràn. Giá trị 230 được nạp vào thanh ghi TH1 để tự động nạp lại cho thanh ghi TL1 khi tràn.

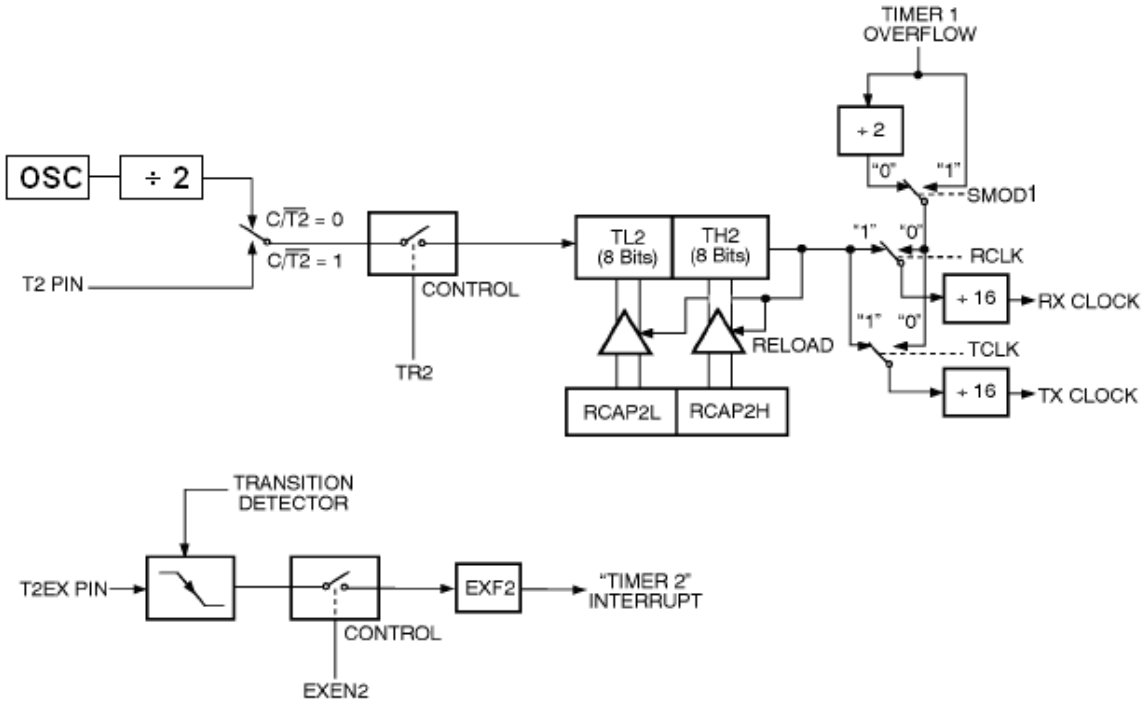
Bảng tóm 5-3 tóm tắt giá trị nạp vào TH1 ứng với tốc độ Baud được tạo ra khi dùng các thạch anh khác nhau.

Bảng 5-3. Tóm tắt tốc độ Baud và giá trị nạp cho TH1

F_{OSC} / Baud rate	11,059 MHz	12MHz	14,7456 MHz	16MHz	20MHz	SMOD
1200	E8h	E6h	E0h	DEh	D5h	0
2400	F4h	F3h	F0h	EFh	EAh	0
4800		F3h	EFh			1
4800	FAh		F8h		F5h	0
9600	FDh		FCh			0
9600					F5h	1
19200	FDh		FCh			1

Việc sử dụng truyền dữ liệu ở tốc độ Baud nào tùy thuộc vào yêu cầu thực tế. Tốc độ càng cao thì dữ liệu truyền càng nhanh. Khi truyền nhiều dữ liệu thì ngoài tốc độ qui định thống nhất giữa 2 hệ thống kết nối với nhau còn phải quan tâm đến tốc độ xử lý dữ liệu nhận về và lấy dữ liệu để gởi đi để không bị mất dữ liệu trong quá trình truyền và nhận. Một trong những giải pháp để kiểm tra xem dữ liệu có bị mất hay không thì phải sử dụng thủ tục bắt tay.

5.7. SỬ DỤNG TIMER 2 TẠO TỐC ĐỘ BAUD



Hình 5-11. Timer 2 trong chức năng tạo tốc độ Baud

Timer 2 có thể được dùng tạo tốc độ Baud cho đường truyền và đường nhận của cổng nối tiếp tùy thuộc vào các bit TCLK và RCLK trong thanh ghi T2CON. Sử dụng Timer 2 để tạo tốc độ Baud được minh họa trong hình 5-11. Tốc độ Baud phụ thuộc vào giá trị nạp trong thanh ghi RCAP2H và RCAP2L được liệt kê trong bảng 5-4 (Theo thứ tự RCAP2H - RCAP2L).

Bảng 5-4. Tóm tắt tốc độ Baud và giá trị nạp cho RCAP2H - RCAP2L

F_{OSC} / Baud rate	6MHz	11,059MHz	12MHz	16MHz
1200	FFh-64h	FEh-E0h	FEh-C8h	FEh-5Fh
2400	FFh-B2h	FFh-70h	FFh-64h	FFh-30h
4800	FFh-D9h	FFh-B8h	FFh-B2h	FFh-98h
9600		FFh-DCh	FFh-D9h	FFh-CCh
19200		FFh-EEh		FFh-E6h

Chương 6

HOẠT ĐỘNG NGẮT

6.1. GIỚI THIỆU NGẮT CỦA VI ĐIỀU KHIỂN 8051

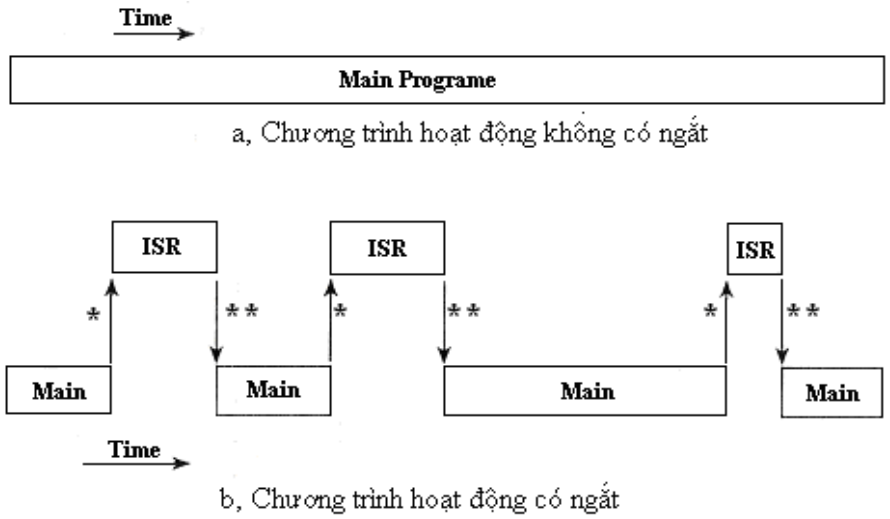
Ngắt (Interrupt) sử dụng trong vi xử lý hay vi điều khiển hoạt động như sau: vi xử lý hay vi điều khiển luôn thực hiện một chương trình mà ta thường gọi là chương trình chính, khi có một sự tác động từ bên ngoài bằng phần cứng hay sự tác động bên trong làm cho vi xử lý ngừng thực hiện chương trình chính để thực hiện một chương trình khác (còn gọi là chương trình phục vụ ngắt ISR – Interrupt Service Rountine) và sau khi thực hiện xong vi xử lý trở lại thực hiện tiếp chương trình chính. Quá trình làm gián đoạn vi xử lý thực hiện chương trình chính xem như là ngắt.

Các sự tác động làm ngừng chương trình chính gọi là các nguồn ngắt, trong vi điều khiển khi Timer/Counter đếm tràn cũng sẽ tạo ra ngắt. Ngắt đóng một vai trò quan trọng trong lập trình điều khiển.

Khi sử dụng ngắt sẽ cho phép vi xử lý hay vi điều khiển đáp ứng nhiều sự kiện quan trọng và giải quyết sự kiện đó trong khi chương trình khác đang thực thi. Ví dụ: Vi điều khiển đang thực hiện chương trình chính thì có dữ liệu từ hệ thống khác gửi đến thì vi điều khiển ngừng chương trình chính để thực hiện chương trình phục vụ ngắt nhận dữ liệu xong rồi trở lại tiếp tục thực hiện chương trình chính, hoặc có một tín hiệu báo ngắt từ bên ngoài thì vi điều khiển sẽ ngừng thực hiện chương trình chính để thực hiện chương trình ngắt rồi tiếp tục thực hiện chương trình chính.

Ta có thể sử dụng ngắt để yêu cầu vi điều khiển thực hiện nhiều chương trình cùng một lúc có nghĩa là các chương trình được thực hiện xoay vòng.

Ta có thể minh họa quá trình thực hiện 1 chương trình trong trường hợp có ngắt và không có ngắt như hình 6-1.



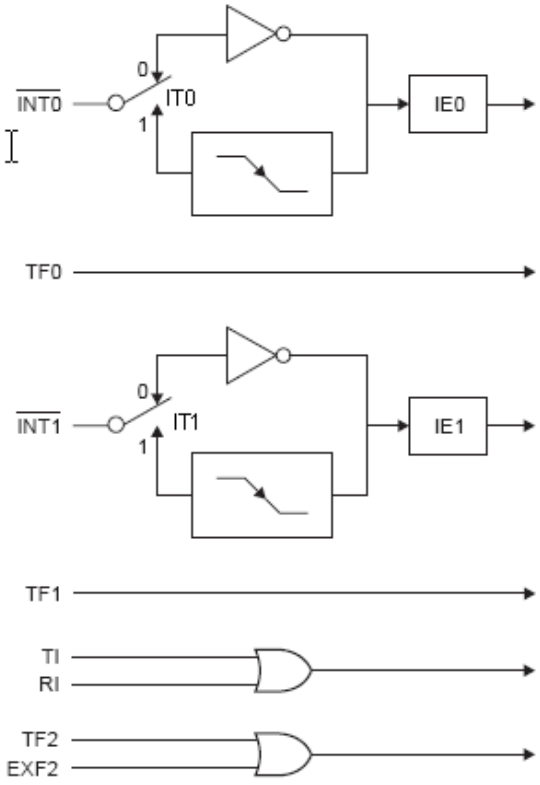
Hình 6-1 Vi điều khiển thực hiện chương trình chính khi không có ngắt và khi có ngắt.

Trong đó: Ký hiệu * cho biết vi điều khiển ngừng chương trình chính để thực thi chương trình con phục vụ ngắt ISR. Còn ký hiệu ** cho biết vi điều khiển quay trở lại thực hiện tiếp chương trình chính sau khi thực hiện xong chương trinh con phục vụ ngắt ISR.

6.2. TỔ CHỨC NGẮT CỦA 8051

Vi điều khiển 8051 có 5 nguồn ngắt: 2 ngắt ngoài, 2 ngắt Timer và một ngắt Port nối tiếp. Vi điều khiển 8052 có thêm một nguồn ngắt là của Timer2 (hình 6-2). Mặc nhiên khi vi điều khiển bị reset thì tất cả các ngắt sẽ mất tác dụng và được cho phép bởi phần mềm.

Trong trường hợp có hai hoặc nhiều nguồn ngắt tác động đồng thời hoặc vi điều khiển đang phục vụ ngắt thì xuất hiện một ngắt khác, thì sẽ có hai cách giải quyết là kiểm tra liên tiếp và sử dụng chế độ ưu tiên.



Hình 6-2. Các nguồn ngắt của họ 8052

6.2.1. Cho phép / cấm ngắt.

Khi ta cho phép ngắt và khi ngắt tác động thì vi điều khiển sẽ ngừng chương trình chính để thực hiện chương trình con phục vụ ngắt, còn khi không cho phép thì dù có sự tác động đến ngắt thì vi điều khiển vẫn tiếp tục thực hiện chương trình chính – không thực hiện chương trình phục vụ ngắt.

Trong vi điều khiển có thanh ghi IE (Interrupt Enable) ở tại địa chỉ 0A8h có chức năng cho phép / cấm ngắt. Ta sử dụng thanh ghi này để cho phép hay không cho phép đối với từng nguồn ngắt và cho toàn bộ các nguồn ngắt.

IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0
EA	**_**	**ET2**	**ES**	**ET1**	**EX1**	**ET0**	**EX0**

Hình 6-3. Thanh ghi IE

Hoạt động của từng bit trong thanh ghi cho phép ngắt IE được tóm tắt trong bảng 6-1

Bảng 6-1. Tóm tắt chức năng các bit của thanh ghi IE.

Bit	Kí hiệu	Chức năng (Enable = 1; Disable = 0)
IE.7	EA	Cho phép toàn bộ hoặc cấm toàn bộ các nguồn ngắt.
IE.6	-	Chưa dùng đến
IE.5	ET2	Cho phép ngắt Timer2 (8052).
IE.4	ES	Cho phép ngắt cổng nối tiếp.
IE.3	ET1	Cho phép ngắt Timer1.
IE.2	EX1	Cho phép ngắt ngoài External1 (INT1).
IE.1	ET0	Cho phép ngắt Timer0.
IE.0	EX0	Cho phép ngắt ngoài External0 (INT0).

Ghi chú: Giá trị thanh ghi khi Reset là 0x00000b

Trong thanh ghi IE có bit IE.6 chưa dùng đến, bit IE.7 là bit cho phép/cấm ngắt toàn bộ các nguồn ngắt. Khi bit IE.7= 0 thì cấm hết tất cả các nguồn ngắt, khi bit IE.7=1 thì cho phép tất cả các nguồn ngắt nhưng còn phụ thuộc vào từng bit điều khiển ngắt của từng nguồn ngắt. Thanh ghi này cho phép truy xuất từng bit.

6.2.2. Ưu tiên ngắt

Khi có nhiều nguồn ngắt tác động cùng lúc thì ngắt nào quan trọng cần thực hiện trước và ngắt nào không quan trọng thì thực hiện sau giống như các công việc mà ta giải quyết hằng ngày. Ngắt cũng được thiết kế có sự sắp xếp thứ tự ưu tiên từ thấp đến cao để người lập trình sắp xếp các nguồn ngắt theo yêu cầu công việc mà mình xử lý.

IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
_	**_**	**PT2**	**PS**	**PT1**	**PX1**	**PT0**	**PX0**

Hình 6-4. Thanh ghi IP

Bảng 6-2. Tóm tắt chức năng các bit của thanh ghi IP

Bit	Kí hiệu	Chức năng
IP.7	-	Chưa sử dụng
IP.6	-	Chưa sử dụng
IP.5	PT2	Ưu tiên cho sự ngắt Timer 2 (8052).
IP.4	PS	Ưu tiên cho sự ngắt Port nối tiếp.
IP.3	PT1	Ưu tiên cho sự ngắt Timer 1.
IP.2	PX1	Ưu tiên cho sự ngắt ngoài External 1.
IP.1	PT0	Ưu tiên cho sự ngắt Timer 0.
IP.0	PX0	Ưu tiên cho sự ngắt ngoài External 0.

Ghi chú: Giá trị thanh ghi khi Reset là 0x00000b

Khi reset hệ thống thì thanh ghi ưu tiên ngắt IP bị xóa và tất cả các ngắt ở mức ưu tiên thấp nhất.

Ngắt trong 8051 có mức ưu tiên thấp và mức ưu tiên cao. Nếu vi điều khiển đang thực hiện chương trình con phục vụ ngắt có mức ưu tiên thấp và có một yêu cầu ngắt với mức ưu tiên cao hơn xuất hiện thì vi điều khiển phải ngừng thực hiện chương trình con phục vụ ngắt có mức ưu tiên thấp để thực hiện chương trình con phục vụ ngắt mới có ưu tiên cao hơn.

Ngược lại nếu vi điều khiển đang thực hiện chương trình con phục vụ ngắt có mức ưu tiên cao hơn và có yêu cầu ngắt với mức ưu tiên thấp hơn xuất hiện thì vi điều khiển vẫn tiếp tục thực hiện cho đến khi thực hiện xong chương trình phục vụ ngắt có ưu tiên cao hơn rồi mới thực hiện chương trình phục vụ ngắt có ưu tiên thấp đang yêu cầu.

Chương trình chính mà vi điều khiển luôn thực hiện trong một hệ thống thì ở mức thấp nhất, không có liên kết với yêu cầu ngắt nào, luôn luôn bị ngắt bất chấp ngắt ở mức ưu tiên cao hay thấp. Nếu có 2 yêu cầu ngắt với các ưu tiên khác nhau xuất hiện đồng thời thì yêu cầu ngắt có mức ưu tiên cao hơn sẽ được phục vụ trước.

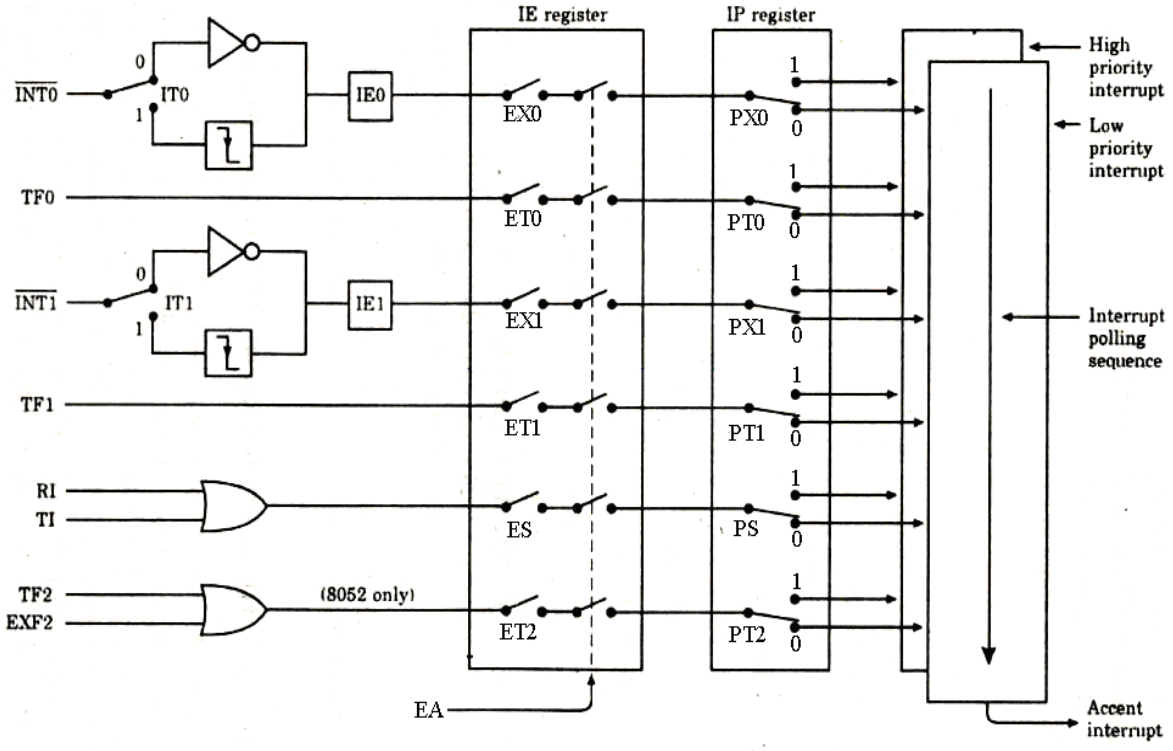
6.2.3. Kiểm tra vòng quét liên tiếp.

Nếu 2 yêu cầu ngắt có cùng mức ưu tiên xuất hiện đồng thời thì vòng quét kiểm tra liên tiếp sẽ xác định yêu cầu ngắt nào sẽ được phục vụ trước tiên. Vòng quét kiểm tra liên tiếp theo thứ tự ưu tiên từ trên xuống là: ngắt ngoài thứ 0 (INT0), ngắt Timer0, ngắt ngoài thứ 1 (INT1), ngắt Timer1, ngắt truyền dữ liệu nối tiếp (Serial Port), ngắt Timer 2. Hình 6-5 sẽ minh họa cho trình tự trên.

Có 6 nguồn ngắt của 8052 và tác dụng của các thanh ghi IE hoạt động như một contact On/Off còn thanh ghi IP hoạt động như một contact chuyển mạch giữa 2 vị trí để lựa chọn 1 trong 2.

Trước tiên ta xét thanh ghi IE: bit cho phép ngắt toàn cục (Global Enable) nếu được phép sẽ đóng toàn bộ các contact và tùy thuộc vào bit cho phép của từng nguồn ngắt có được phép hay không và chúng hoạt động cũng giống như một contact: nếu được phép thì đóng mạch và tín hiệu yêu cầu ngắt sẽ đưa vào bên trong để xử lý, nếu không được phép thì contact hở mạch nên tín hiệu yêu cầu ngắt sẽ không đưa vào bên trong và không được xử lý.

Tiếp theo là thanh ghi IP: tín hiệu sau khi ra khỏi thanh ghi IE thì đưa đến thanh ghi IP để sắp xếp ưu tiên cho các nguồn ngắt. Có 2 mức độ ưu tiên: mức ưu tiên cao và mức ưu tiên thấp. Nếu các nguồn nào có ưu tiên cao thì contact chuyển mạch sẽ đưa tín hiệu yêu cầu ngắt đó đến vòng kiểm tra có ưu tiên cao, nếu các nguồn nào có ưu tiên thấp thì contact chuyển mạch sẽ đưa tín hiệu yêu cầu ngắt đó đến vòng kiểm tra có ưu tiên thấp.



Hình 6-5. Cấu trúc ngắt của 8051

Vòng kiểm tra ngắt ưu tiên cao sẽ được thực hiện trước và sẽ kiểm tra theo thứ tự từ trên xuống và khi gặp yêu cầu ngắt nào thì yêu cầu ngắt đó sẽ được thực hiện. Sau đó tiếp tục thực hiện cho vòng kiểm tra ưu tiên ngắt có mức ưu tiên thấp hơn.

Ngắt do cổng nối tiếp là kết quả OR của cờ ngắt khi thi RI và cờ ngắt khi phát TI, ngắt do bộ Timer2 do cờ tràn TF2 hoặc do cờ từ bên ngoài EXF2. Các bit cờ tạo ra các ngắt được tóm tắt ở bảng 6-3

Bảng 6-3. Các cờ ngắt

Ngắt	Cờ	Thanh ghi SFR và vị trí bit
Ngắt do bên ngoài (Ngắt ngoài 0)	IE0	TCON.1
Ngắt do bên ngoài (Ngắt ngoài 1)	IE1	TCON.3
Ngắt do Timer1	TF1	TCON.7
Ngắt do Timer0	TF0	TCON.5
Ngắt do cổng nối tiếp khi phát	TI	SCON.1
Ngắt do cổng nối tiếp khi thu	RI	SCON.0
Ngắt do Timer2	TF2	T2CON.7 (8052)
Ngắt do Timer2	EXF2	T2CON.6 (8052)

6.3. XỬ LÝ NGẮT

Khi tín hiệu yêu cầu ngắt xuất hiện và được chấp nhận bởi CPU thì CPU thực hiện các công việc sau.

- Nếu CPU đang thực hiện lệnh thì phải chờ thực hiện xong lệnh đang thực hiện.
- Giá trị của bộ đếm chương trình PC được cất giữ vào Stack (chính là địa chỉ của lệnh tiếp theo trong chương trình chính).
- Trạng thái ngắt hiện hành được lưu vào bên trong.
- Các yêu cầu ngắt khác sẽ bị ngăn lại ở mức ngắt.
- Địa chỉ của chương trình phục vục ngắt tương ứng sẽ được nạp vào bộ đếm chương trình PC.
- Thực hiện chương trình phục vụ ngắt ISR.

Chú ý: chương trình con phục vụ ngắt không được làm mất hoặc làm sai địa chỉ của PC đã lưu trong ngăn xếp nếu điều này xảy ra thì khi trở lại chương trình chính CPU sẽ không thực hiện tiếp công việc của chương trình chính và chúng ta cũng không xác định CPU đang làm gì và ở đâu. Khi đó chúng ta mất quyền kiểm soát vi xử lý.

Một điều cần phải chú ý nữa là trong lập trình chúng ta không được nhảy từ chương trình con sang chương trình chính để thực hiện tiếp chương trình vì làm như vậy sau nhiều lần thực hiện thì bộ nhớ ngăn xếp sẽ bị tràn và ghi đè lên các dữ liệu khác làm sai chương trình. Trong trường hợp này chương trình chúng ta thực hiện đúng một vài lần và sau đó thì sai.

6.3.1. Các vectơ ngắt (Interrupt Vectors)

Như đã trình bày ở trên, khi có một yêu cầu ngắt xảy ra thì sau khi cất giá trị địa chỉ trong PC vào ngăn xếp thì địa chỉ của chương trình con phục vụ ngắt tương ứng

còn gọi bởi vectơ địa chỉ ngắt sẽ được nạp vào thanh ghi PC, địa chỉ này là cố định và do nhà chế tạo vi điều khiển qui định. Các chương trình ngắt phải bắt đầu viết đúng tại địa chỉ quy định đó. Các vectơ địa chỉ ngắt được cho trong bảng 6.4.

Bảng 6-4. Tóm tắt vector địa chỉ ngắt.

Interrupt	Flag	Vectors Address
System Reset	RST	0000h
External 0	IE 0	0003h
Timer 0	TF 0	000Bh
External 1	IE 1	0013h
Timer 1	TF1	001Bh
Serial Port	RI hoặc TI	0023h
Timer 2	TF 2 hoặc EXF2	002Bh

Vectơ reset hệ thống bắt đầu tại địa chỉ 0000h khi reset vi điều khiển thì thanh ghi PC = 0000h và chương trình chính luôn bắt đầu tại địa chỉ này.

Khi sử dụng yêu cầu ngắt nào thì chương trình con phục vụ ngắt phải viết đúng tại địa chỉ tương ứng. Ví dụ sử dụng ngắt Timer T0 thì chương trình ngắt phải viết tại địa chỉ 000Bh

Do khoảng vùng nhớ giữa các vector địa chỉ của các nguồn ngắt chỉ có vài ô nhớ ví dụ như vector địa chỉ ngắt của ngắt INT0 tại 0003hvà vector địa chỉ ngắt của ngắt T0 tại 000Bh chỉ cách nhau có 8 ô nhớ. Nếu chương trình phục vụ ngắt của ngắt INT0 có kích thước lớn hơn 8 byte thì nó sẽ đụng đến vùng nhớ của ngắt T0. Cách giải quyết tốt nhất là ngay tại địa chỉ 0003h viết lệnh nhảy đến một vùng nhớ khác rộng hơn. Còn nếu các ngắt T0 và các ngắt khác không sử dụng thì có thể viết chương trình tại đó cũng được.

Chương trình chính luôn bắt đầu tại địa chỉ 0000h sau khi reset hệ thống, nếu trong chương trình có sử dụng ngắt thì ta phải dùng lệnh nhảy tại địa chỉ 0000h để nhảy đến một vùng nhớ khác rộng hơn không bị giới hạn để viết tiếp.

6.3.2. Ngắt do các bộ Timer

Các ngắt do các bộ Timer xảy ra do sự kiện tràn các bộ Timer, khi đó các cờ tràn TFx sẽ được thiết lập bằng 1. Khi chương trình phục vụ ngắt được đáp ứng thì các cờ tràn sẽ được tự động xóa bằng phần cứng. Riêng TF2 chỉ có thể xóa bằng phần mềm.

6.3.3. Ngắt do cổng nối tiếp

Ngắt do cổng nối tiếp xảy ra khi cờ ngắt phát TI hoặc cờ ngắt thu RI được thiết lập lên 1. Ngắt phát xảy ra khi bộ đệm truyền rỗng, ngắt thu xảy ra khi một ký tự được nhận xong và đang đợi trong SBUF để được đọc (Bộ đệm truyền đã đầy).

Các ngắt do cổng nối tiếp khác với các ngắt do Timer. Cờ gây ra ngắt cổng nối tiếp không bị xóa bằng phần cứng khi CPU chuyển tới chương trình phụ vụ ngắt do có hai nguồn ngắt do cổng nối tiếp TI và RI, nguồn ngắt phải được xác định trong chương trình phục vụ ngắt và cờ tạo ngắt sẽ được xóa bằng phần mềm.

6.3.4. Ngắt ngoài

Các ngắt ngoài xảy ra khi có một mức thấp hoặc cạnh xuống trên chân INT0 hoặc INT1 của bộ vi điều khiển.

Các cờ tạo ngắt này là các bit IE0 và IE1 trong thanh ghi IE. Khi quyền điều khiển đã chuyển đến chương trình phụ vụ ngắt cờ tạo ngắt chỉ được xóa nếu ngắt được tích cực ở cạnh xuống. Nếu ngắt được thực hiện theo mức thì nguồn ngắt bên ngoài sẽ được điều khiển mức của cờ thay cho phần cứng.

Cách tích cực ngắt được đặt bởi bit ITx. Nếu ITx=0 ngắt được tích cực bằng mức thấp, nếu ITx=1 ngắt được tích cực bằng cạnh xuống (sườn âm).Nếu ngắt ngoài được tác động bằng cạnh xuống thì nguồn bên ngoài phải giữ trên chân INTx ở mức cao ít nhất trong một chu kỳ máy. và mức thấp cũng ít nhất trong một chu kỳ máy. để CPU phát hiện được cạnh xuống. Nếu ngắt ngoài tác động bằng mức thấp thì phải giữ INTx mức thấp cho đến khi ngắt được đáp ứng và không tác động nữa khi chương trình phục vụ ngắt đã được hoàn tất, nếu không một ngắt khác sẽ được lặp lại.

Chương 7

TRÌNH DỊCH KEIL C51 CHO HỌ VI ĐIỀU KHIỂN 8051

7.1. GIỚI THIỆU

Nhiều người lập trình cho họ vi điều khiển 8051 vẫn thường làm việc với hợp ngữ. Do bản chất hợp ngữ là ngôn ngữ mã gợi nhớ và khi dịch sẽ ánh xạ 1 – 1 sang ngôn ngữ máy, việc lập trình với hợp ngữ thường gặp khó khăn, nhất là các chương trình lớn cũng như với các phiên bản nâng cao của họ 8051. Chương trình viết bằng hợp ngữ thường khó đọc và khó bảo trì hơn so với chương trình viết bằng hợp ngôn ngữ lập trình cấp cao như C chẳng hạn.

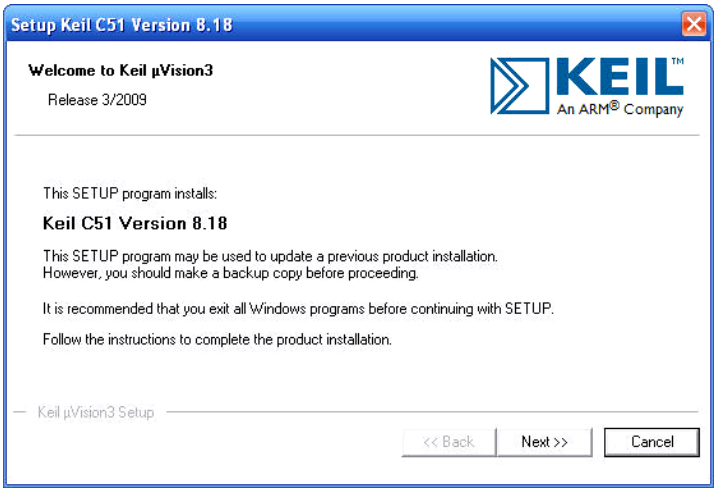
Ngôn ngữ C cho phép người thiết kế hệ thống viết chương trình tương đồng nhiều hơn với suy nghĩ của con người so với chương trình viết bằng hợp ngữ. Người thiết kế hệ thống có thể dành nhiều thời gian hơn vào việc thiết kế giải thuật của hệ thống thay vì phải tập trung vào việc thực hiện riêng rẽ từng giải thuật. Điều này làm giảm một cách đáng kể thời gian phát triển hệ thống và giảm thời gin gỡ rối vì chương trình sẽ dễ hiểu hơn.

Bằng cách sử dụng ngôn ngữ C người lập trình không nhất thiết phải thật am tường cấu trúc vi điều khiển điều này giúp người lập trình tập trung đầu tư nhiều thời gian vào ý tưởng và giải thuật. Ngoài ra thì chương trình viết bằng ngôn ngữ C sẽ dễ dàng sử dụng lại với các hệ thống khác so với chương trình viết bằng hợp ngữ.

Một trong những phần mềm hỗ trợ biên dịch C cho họ 8051 là Keil C. Đây là trình biên dịch C cho họ 8051 phổ biến nhất hiện nay.

7.2. HƯỚNG DẪN CÀI ĐẶT PHẦN MỀM KEIL C51 VERSION 8.18

Bước 1: Click vào file ***c51v818.exe*** trong CD đi kèm hoặc trong Folder chứa phần mềm cài đặt Keil C51, cửa sổ giao diện Setup Keil C51 (hình 7-1) xuất hiện.

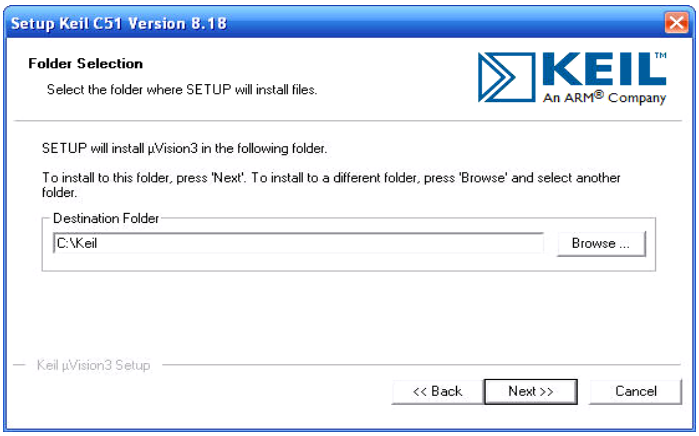


Hình 7-1

Bước 2: Chọn ***Next***, cửa sổ hình 7-2 sẽ xuất hiện

Hình 7-2

Bước 3: Click vào ***I agree to all the items of the preceding License Agreement,*** sau đó chọn ***Next***, cửa sổ giao diện hình 7-3 sẽ xuất hiện

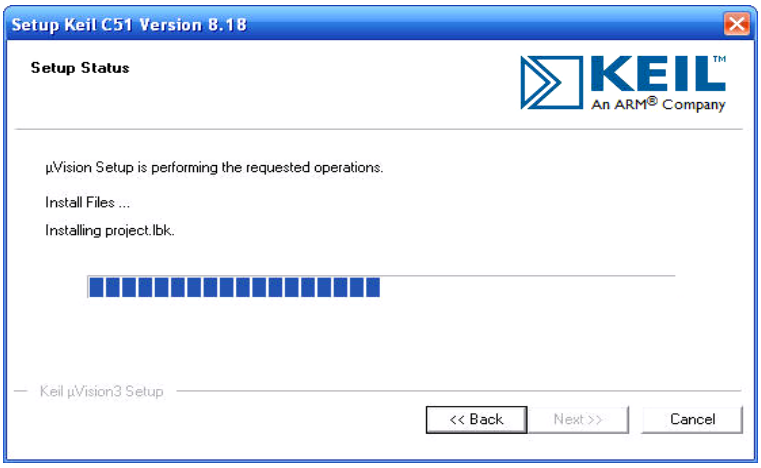


Hình 7-3

Bước 4: Nên để ***mặc định*** và chọn ***Next.*** Cửa sổ hình 7-4 xuất hiện để điền thông tin cá nhân. Người cài đặt phải điền đầy đủ các thông tin cá nhân như ví dụ trong hình 7-4

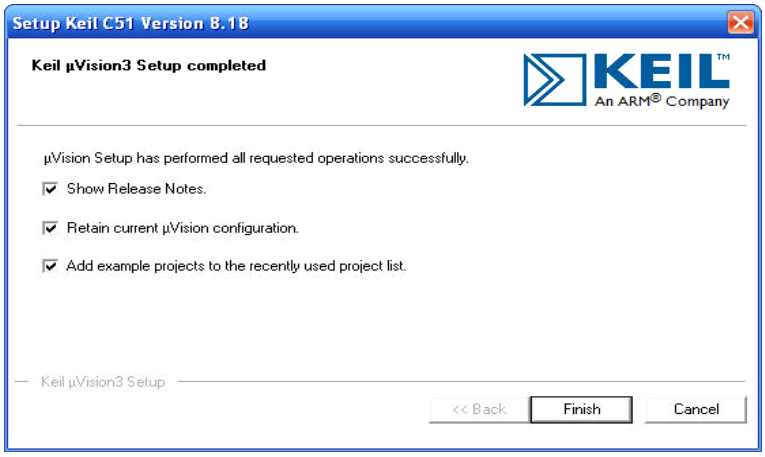
Hình 7-4

Bước 5: Chọn ***Next*** để bắt đầu cài đặt. Quá trình cài đặt hiển thị trên cửa sổ hình 7-5



Hình 7-5

Bước 6: Khi cài đặt xong, màn hình sau xuất hiện và nhấn ***Finish*** để hoàn tất quá trình cài đặt.



Hình 7-6

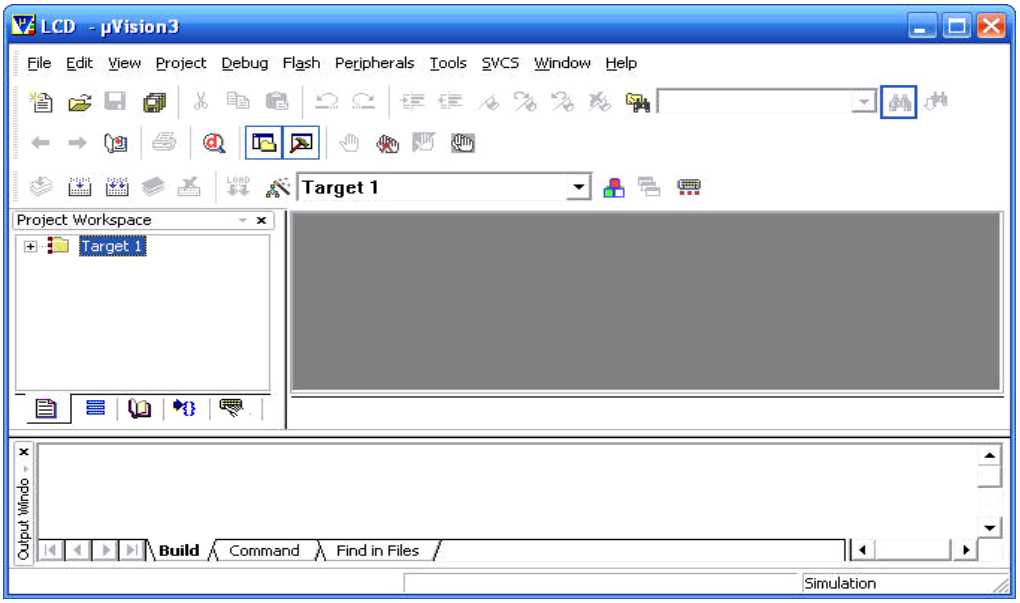
Sau khi cài đặt xong thì sẽ có trang web giới thiệu về phiên bản này của Keil C xuất hiện để người dùng tìm hiểu thông tin về phiên bản.

7.3. HƯỚNG DẪN SỬ DỤNG PHẦN MỀM KEIL C51

7.3.1. Tạo project mới

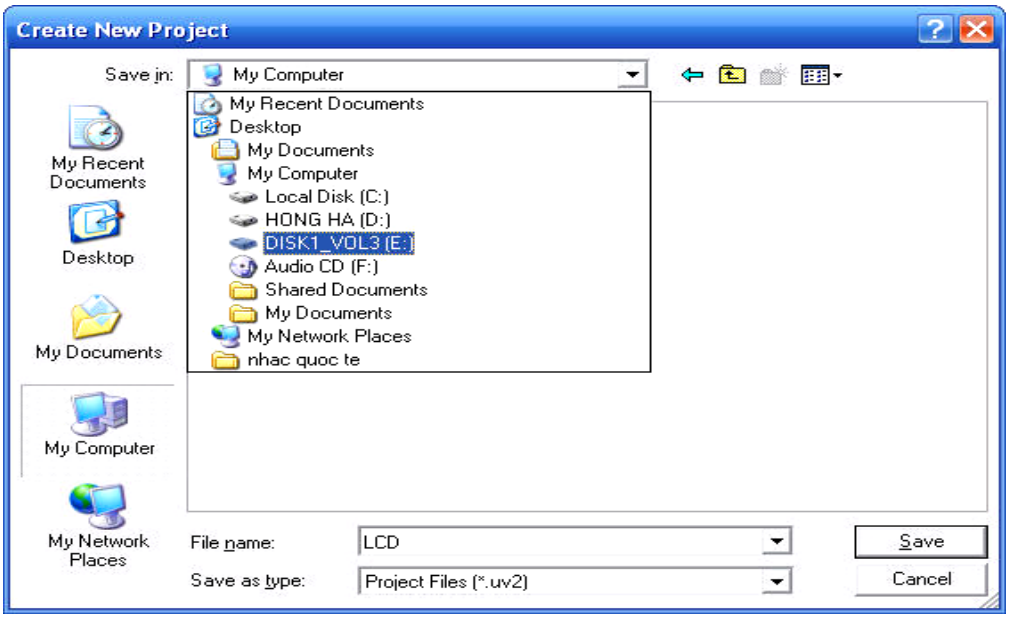
Bước 1: Chạy chương trình Keil C bằng cách double click vào biểu tượng 

trên màn hình hoặc vào ***Start\ Programs\Keil uVision3***. Màn hình làm việc của Keil C (hình 7-7) xuất hiện



Hình 7-7

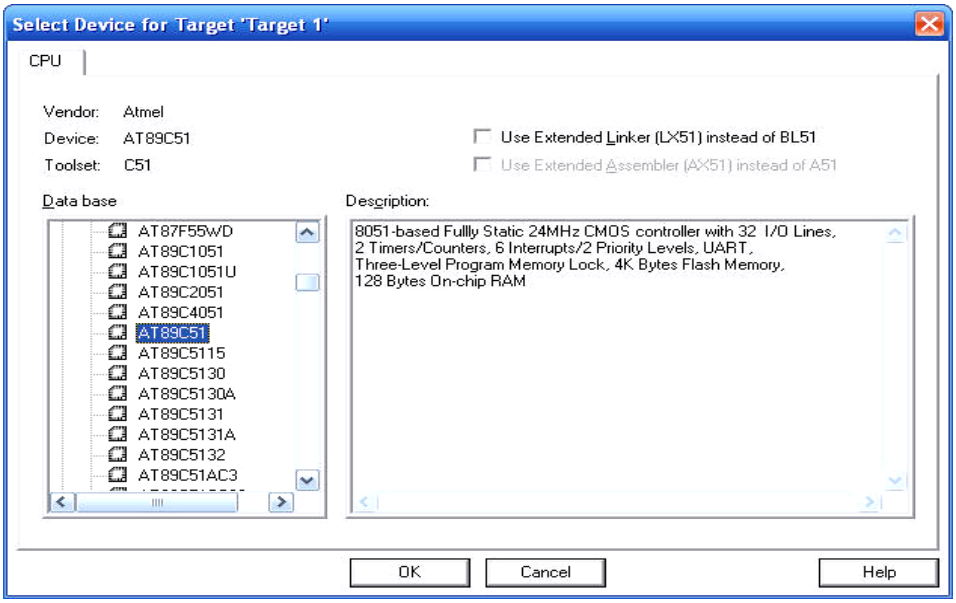
Bước 2: Vào ***Project\ New µVision Project*** cửa sổ Creat New Project xuất hiện.



Hình 7-8

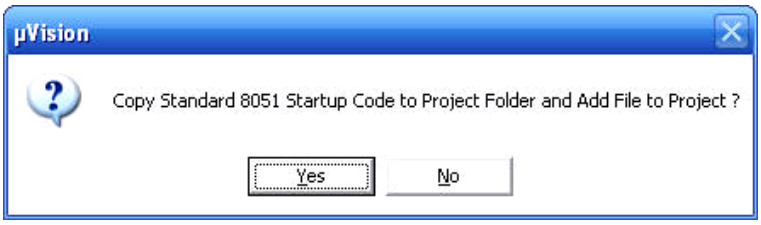
Bước 3: Gõ tên Project trong mục ***File name***, chọn Folder để lưu Project trong mục ***Save in*** và click vào nút Save (Hình 7-8). Cửa sổ ***Select Device for Target*** xuất hiện để chọn vi điều khiển (Hình 7-9).

Chú ý: Nên tạo một Folder cùng tên với Project và lưu Project vào đó để dễ quản lý bởi đi kèm Project có nhiều File được tạo ra.



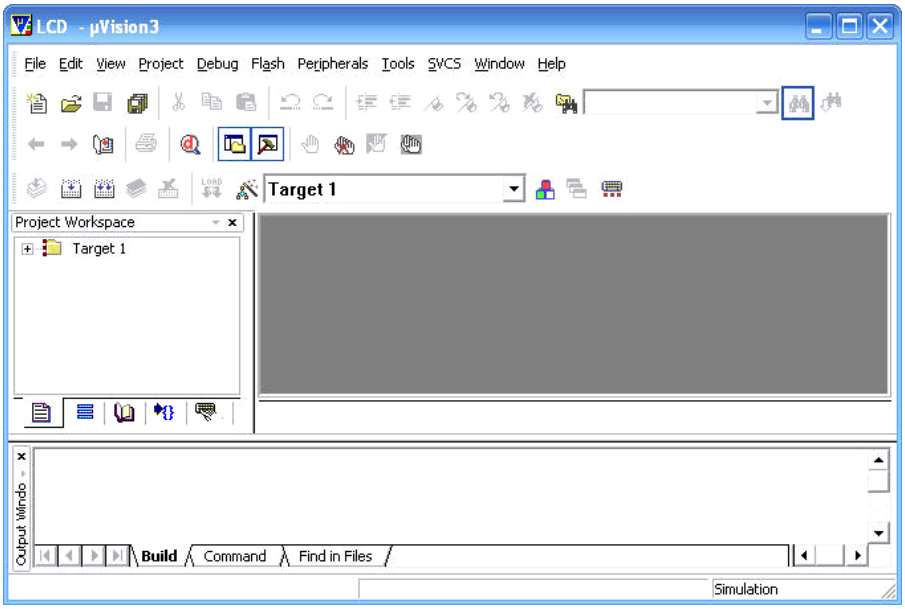
Hình 7-9

Bước 4: Chọn đúng tên linh kiện cần lập trình. Ví dụ lập trình cho AT89C51 chọn Atmel\AT89C51 và click nút OK.(Hình 7-9). Cửa sổ giao diện hình 7-10 xuất hiện



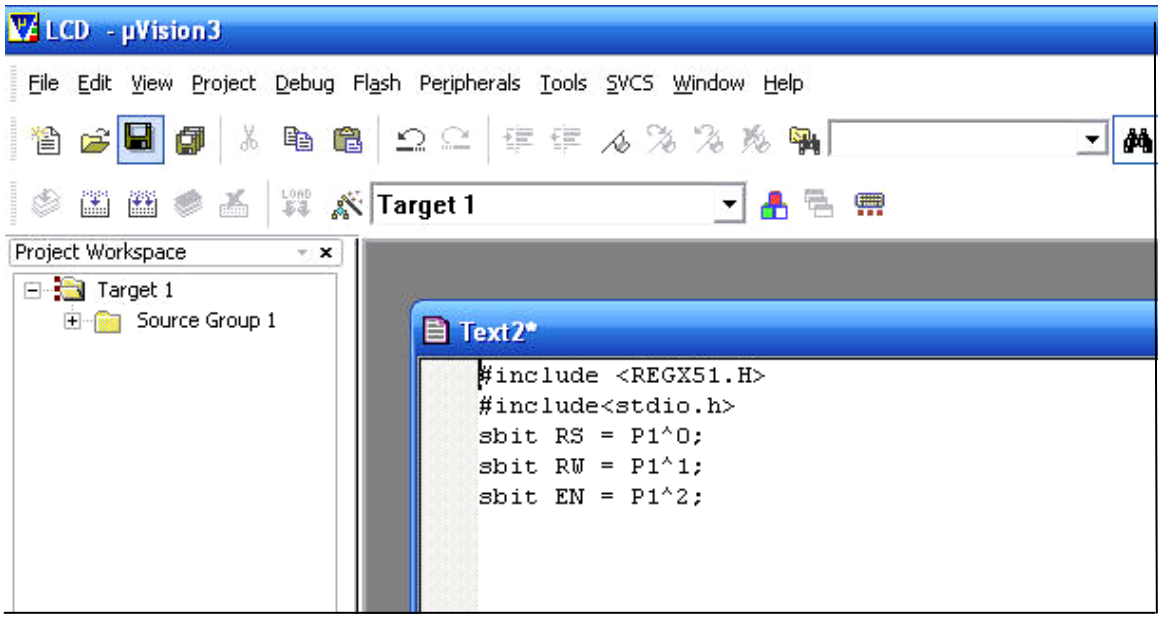
Hình 7-10

Bước 5: Click nút OK (hình 7-10) màn hình Keil C có giao diện như hình 7-11



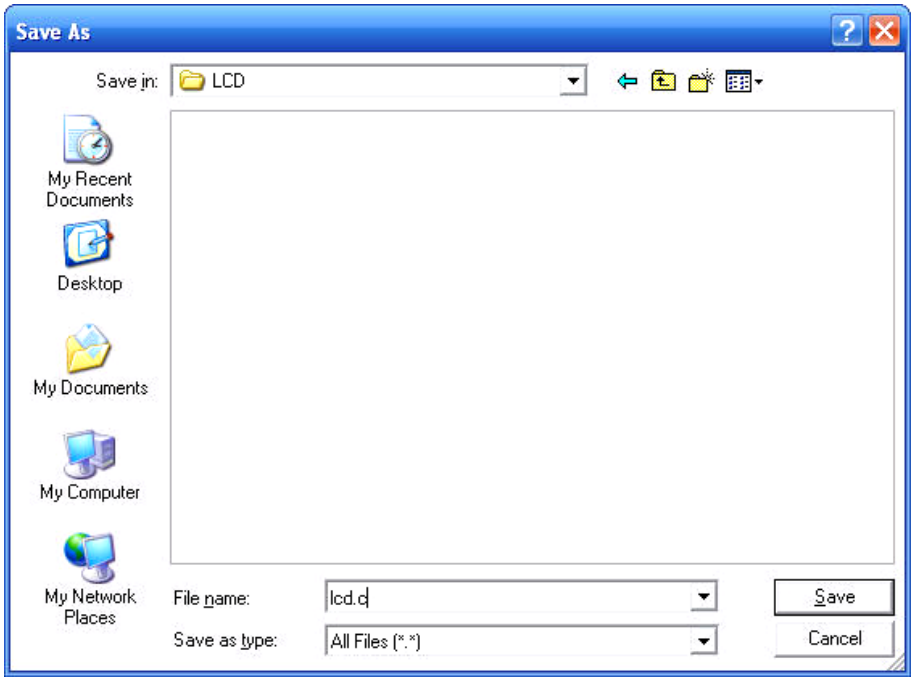
Hình 7-11

Bước 6: Tạo file nguồn bằng cách vào ***File\ New*** cửa sổ Text xuất hiện (Hình 7-12)



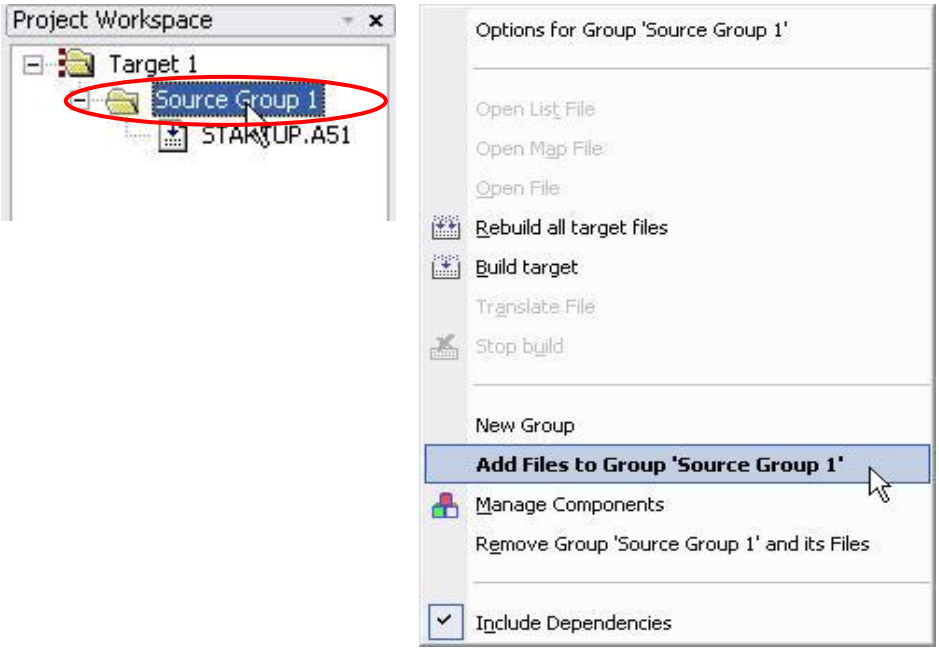
Hình 7-12

Bước 7: Soạn thảo chương trình trong khung Text sau đó vào ***File/ Save*** để lưu tên file trong mục ***File Name*** ở định dạng đuôi .c (Hình 7-13). File này nên lưu trong cùng Folder với Project



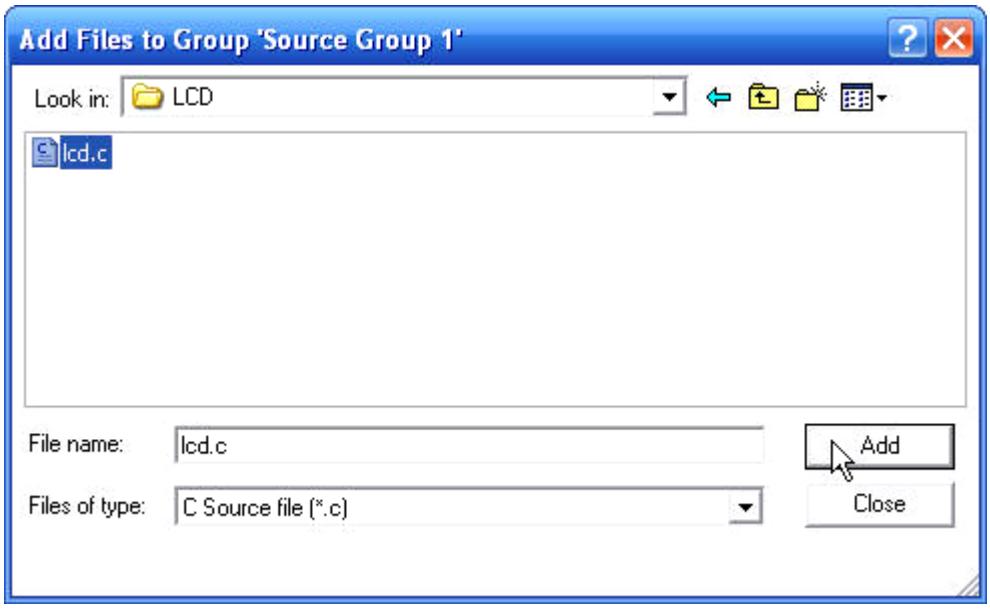
Hình 7-13

Bước 8: Right Click vào ***Source Group 1*** trong cửa sổ ***Project Workspace***, giao diện hình 7-14 xuất hiện



Hình 7-14

Bước 9: Click vào ***Add File to Group 'Source Group'*** giao diện hình 7-15 xuất hiện.

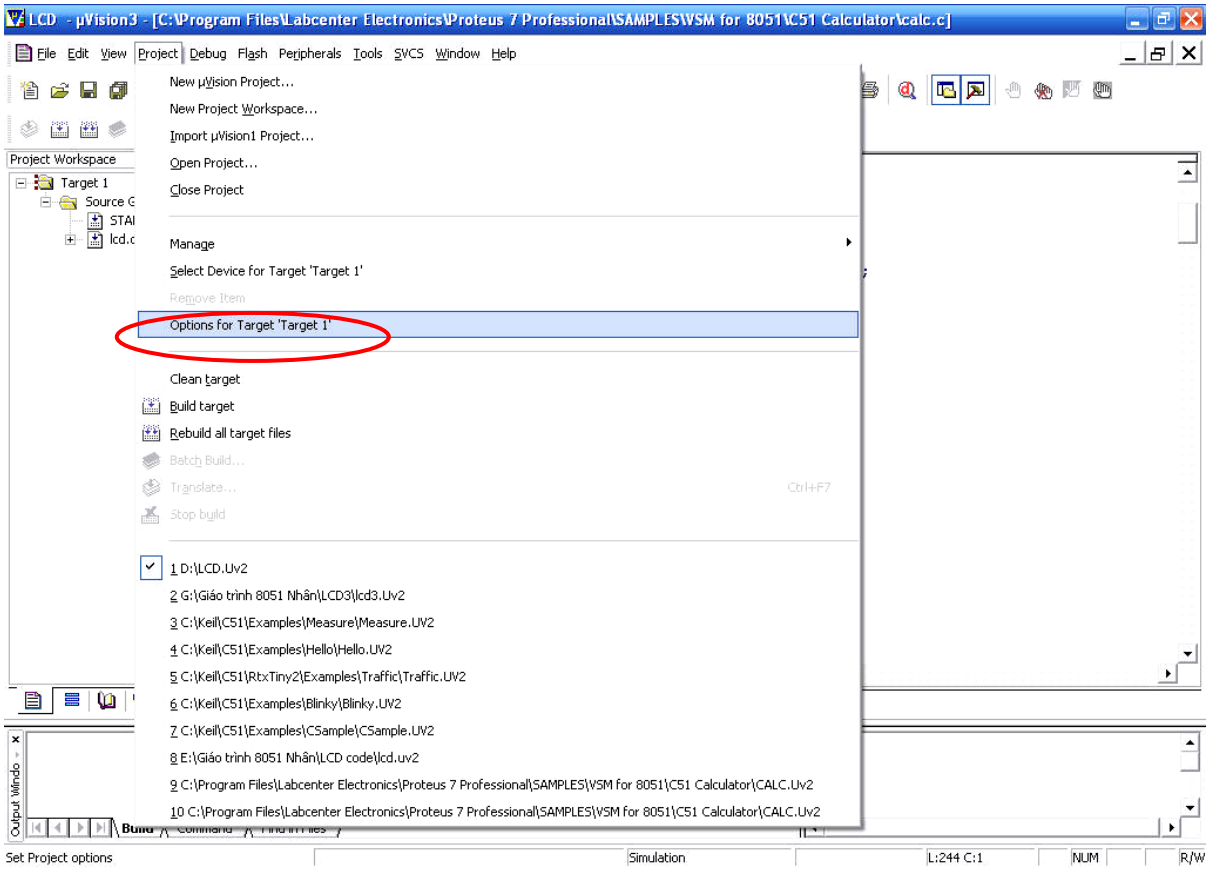


Hình 7-15

Bước 10: Chọn đúng File nguồn đã soạn thảo và click vào nút ***Add*** sau đó click vào nút ***Close*** để kết thúc.

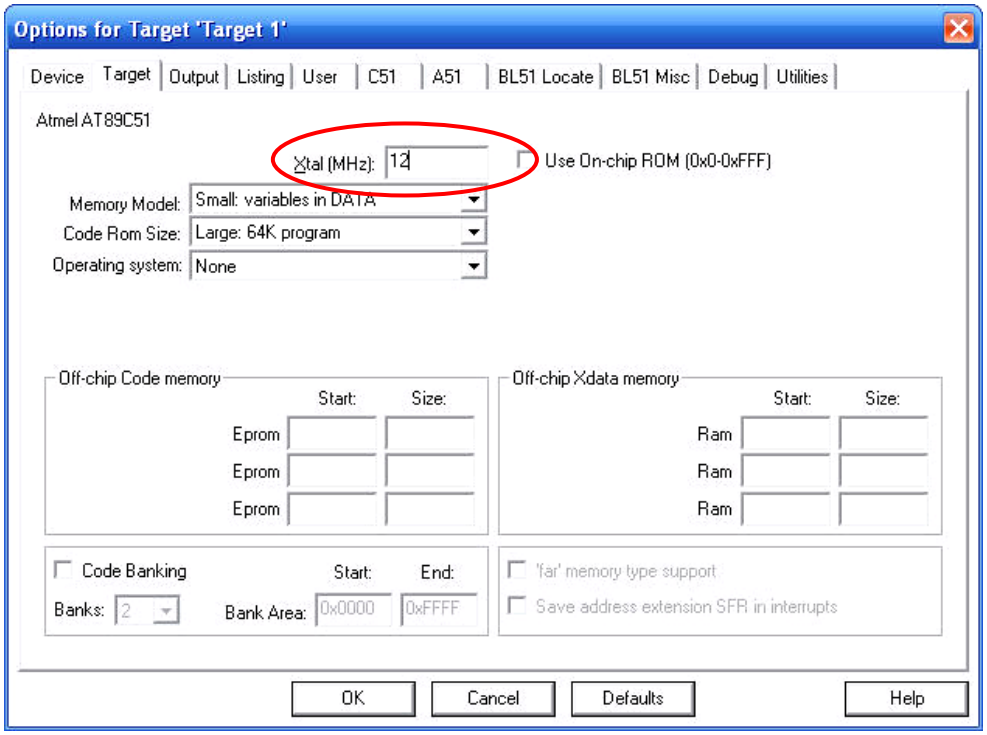
7.3.2. Thiết lập cấu hình tạo File hex

Bước 1: Vào menu ***Project*** chọn ***Option for Target 'Target 1'*** (hình 7-16)



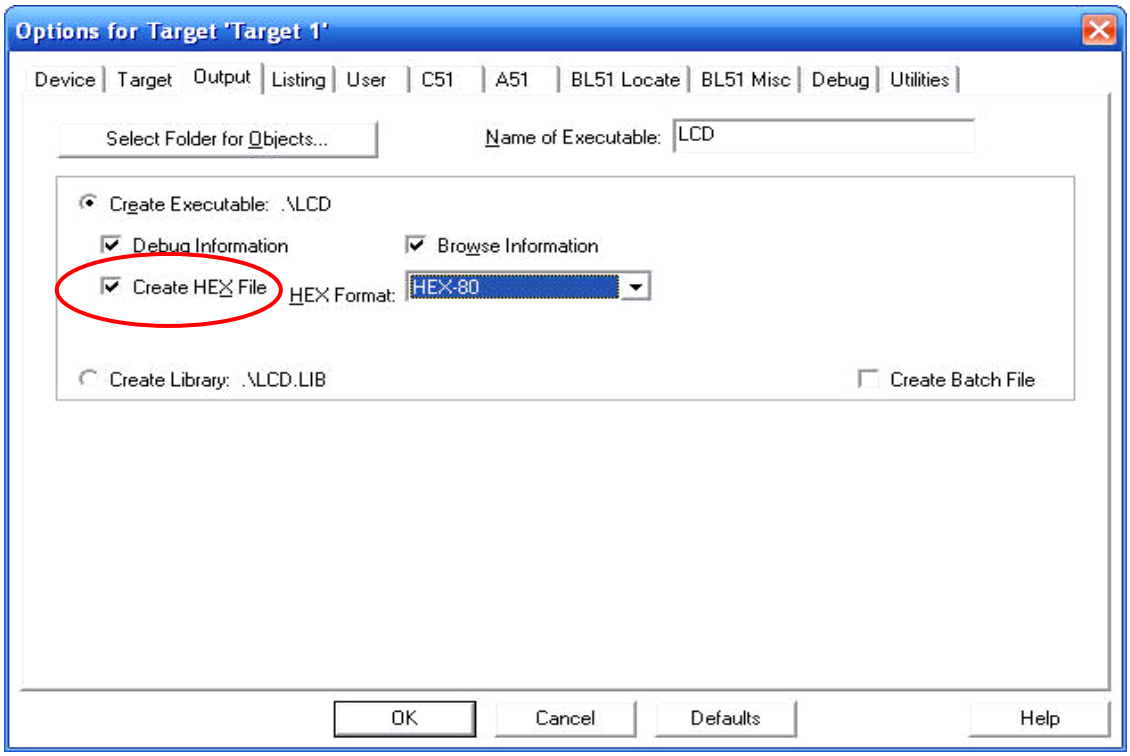
Hình 7-16

Bước 2: Chọn thẻ ***Target*** và điền giá trị thạch anh sử dụng vào ô ***Xtal*** (Hình 7-17)



Hình 7-17

Bước 3: Chọn thẻ ***Output*** và tích vào mục ***Creat HEX file***



Hình 7-18

Bước 4: Biên dịch và tạo File hex bằng cách chọn ***Project/ Build target.*** Các thông báo về quá trình biên dịch sẽ hiển thị trong cửa sổ ***Output Window*** ở phia dưới màn hình làm việc.

7.4. MẪU CHƯƠNG TRÌNH VIẾT TRÊN KEIL C51

Thông thường khi viết chương trình C cho vi điều khiển phải theo khuôn mẫu như sau:

// Khai báo các thư viện

```
#include <reg51.h>
#include <stdio.h>
```

// Khai báo các biến số, hằng số, cấu trúc, macro, chương trình con.

```
int x, i, k;
array[10];
sbit led1 = P1^0;
#define on = 1
#define off = 0
```

// Các chương trình con

```
void sub (int i)
{
Nội dung chương trình con
}
```

// Chương trình chính

```
void main ()
{
Nội dung chương trình chính
}
```

//Chương trình phục vụ ngắt

```
void ngat0 (void) intrerrupt 0
{
Nội dung chương trình phục vụ ngắt
}
```

Tùy theo yêu cầu đặt ra mà chương trình có thể thiếu một vài phần ở trên nhưng bắt buộc phải có chương trình chính (hàm main).

7.5. CÁC KIẾN THỨC CƠ BẢN VỀ C

Trình dịch Keil C hỗ trợ các lệnh trong C chuẩn. Sau đây sẽ nhắc lại một số kiến thức cơ bản về ngôn ngữ lập trình C, các kiến thức này được sử dụng nhiều trong quá trình lập trình cho vi điều khiển.

7.5.1. Các kiểu dữ liệu

Kiểu	Dung lượng (đơn vị là bit)	Phạm vi giá trị
bit	1	0 và 1
char	8	-128 tới 127
unsigned char	8	0 tới 255
signed char	8	-128 tới 127
int	16	-32.768 tới 32.767
unsigned int	16	0 tới 65.535
signed int	16	Giống như kiểu int
short int	8	-128 tới 127
unsigned short int	8	0 tới 255
signed short int	8	Giống như kiểu short int
long int	32	-2.177.783.678 tới 2.177.783.677
signed long int	32	Giống như kiểu long int
unsigned long int	32	0 tới 7.297.967.295
float	32	6 con số thập phân

7.5.2. Các toán tử trên bit

Toán tử	Mô tả
&	Mỗi vị trí của bit trả về kết quả là 1 nếu bit tại vị trí tương ứng của hai toán hạng đều là 1.
|	Mỗi vị trí của bit trả về kết quả là 1 nếu bit tại vị trí tương ứng của một trong hai toán hạng là 1.
~	Đảo ngược giá trị các bit của toán hạng (1 thành 0 và ngược lại).
^	Mỗi vị trí của bit trả về kết quả là 1 nếu bit tại vị trí tương ứng của một trong hai toán hạng là 1 chứ không phải cả hai cùng là 1.
<<	Phép dịch trái nhị phân
>>	Phép dịch phải nhị phân

7.5.3.Các toán tử số học

Các toán tử một ngôi	Chức năng	Các toán tử hai ngôi	Chức năng
-	Lấy đối số	+	Cộng
++	Tăng một giá trị	-	Trừ
--	Giảm một giá trị	*	Nhân
		%	Chia lấy phần dư
		/	Chia lấy phần nguyên
		^	Lấy số mũ

7.5.4. Các toán tử logic

Toán tử	Ý nghĩa
&&	AND: trả về kết quả là **đúng** khi cả 2 toán hạng đều **đúng**
||	OR: trả về kết quả là **đúng** khi chỉ cần một trong hai toán hạng **đúng**
!	NOT: Chuyển đổi giá trị của toán hạng duy nhất từ **đúng** thành **sai** và ngược lại.

7.5.5. Các toán tử quan hệ

Toán tử	Ý nghĩa
>	Lớn hơn
>=	Lớn hơn hoặc bằng
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
==	Bằng
!=	Không bằng

7.5.6. Cấu trúc điều khiển, rẽ nhánh

*a. Cấu trúc **while***

```
while (biểu thức điều kiện)
{
     Các câu lệnh;
}
```

Nếu biểu thức điều kiện **đúng**(giá trị khác 0), chương trình sẽ thực hiện các câu lệnh bên trong vòng lặp **while**

*b. Cấu trúc **do … while***

Cú pháp tổng quát của vòng lặp **do ... while** như sau:

```
do
{
     câu_lệnh;
}
while (điều_kiện);
```

Cặp dấu ngoặc {} là không cần thiết khi chỉ có một câu lệnh hiện diện trong vòng lặp, nhưng việc sử dụng dấu ngoặc {} là một thói quen tốt. Vòng lặp **do ... while** lặp đến khi **điều_kiện** mang giá trị **sai (0)**. Trong vòng lặp **do ... while**, ***câu_lệnh*** (khối các câu lệnh) sẽ được thực thi trước, và sau đó **điều_kiện** được kiểm tra. Nếu điều kiện là **đúng**, chương trình sẽ quay lại thực hiện lệnh **do**. Nếu điều kiện là **sai**, chương trình chuyển đến thực hiện lệnh nằm sau vòng lặp.

*c. Cấu trúc **for***

```
for(biểu thức 1; biểu thức 2; biểu thức 3)
{
Các câu lệnh;
}
```

Biểu thức 1 là khởi tạo giá trị cho biến điều khiển bằng một lệnh gán trước khi thực hiện vòng lặp, lệnh này chỉ thực hiện một lần

Biểu thức 2 là một biểu thức quan hệ, xác định điều kiện thoát cho vòng lặp.

Biểu thức 3 xác định biến điều khiển sẽ bị thay đổi như thế nào sau mỗi lần vòng lặp được lặp lại (thường là tăng hoặc giảm giá trị của biến điều khiển).

Quá trình thực hiện như sau: thực hiện biểu thức 1 (chỉ một lần đầu), kiểm tra biểu thức 2, thực hiện các câu lệnh trong thân vòng lặp, thực hiện biểu thức 3.

Vòng lặp for sẽ tiếp tục được thực hiện chừng nào mà biểu thức điều kiện còn **đúng.** Khi biểu thức điều kiện là **sai** chương trình sẽ thoát ra khỏi vòng lặp **for.**

d. Cấu trúc ***if, if … else***

```
if (biểu thức điều kiện)
{
     Các câu lệnh 1;
}
else
{
     Các câu lệnh 2;
}
```

Nếu biểu thức điều kiện đúng thì chương trình thực hiện các câu lệnh 1, nếu không thì chương trình sẽ thực hiện các câu lệnh 2.

e. Câu lệnh ***switch***

Câu lệnh **switch** cho phép ta đưa ra quyết định có nhiều cách lựa chọn, nó kiểm tra giá trị của một biểu thức trên một danh sách các hằng số nguyên hoặc kí tự. Khi nó tìm thấy một giá trị trong danh sách trùng với giá trị của biểu thức điều kiện, các câu lệnh gắn với giá trị đó sẽ được thực hiện. Cú pháp tổng quát của lệnh **switch** như sau:

```
switch (biểu_thức)
{    case hằng_1:
          chuỗi_câu_lệnh1;
          break;
     case hằng_2:
          chuỗi_câu_lệnh2;
          break;
     case hằng_3:
          chuỗi_câu_lệnh3;
          break;
     default:
          chuỗi_câu_lệnh4;
}
```

Ở đó, **switch**, **case** và **default** là các từ khoá, chuỗi_câu_lệnh có thể là lệnh đơn hoặc lệnh ghép và không cần đặt trong cặp dấu ngoặc. Biểu_thức theo sau từ khóa **switch** phải được đặt trong dấu ngoặc ( ), và toàn bộ phần thân của lệnh switch phải được đặt trong cặp ngoặc nhọn { }. Kiểu dữ liệu kết quả của biểu_thức và kiểu dữ liệu của các hằng theo sau từ khoá case phải đồng nhất. Chú ý hằng số sau **case** chỉ có thể là một **hằng số nguyên** hoặc **hằng ký tự**. Nó cũng có thể là các **hằng biểu thức** –

những biểu thức không chứa bất kỳ một biến nào. Tất cả các giá trị của case phải khác nhau.

Trong câu lệnh **switch**, biểu thức được xác định giá trị, giá trị của nó được so sánh với từng giá trị gắn với từng case theo thứ tự đã chỉ ra. Nếu một giá trị trong một case trùng với giá trị của biểu thức, các lệnh gắn với case đó sẽ được thực hiện. Lệnh **break** cho phép thoát ra khỏi **switch**. Nếu không dùng lệnh **break**, các câu lệnh gắn với **case** bên dưới sẽ được thực hiện không kể giá trị của nó có trùng với giá trị của biểu thức điều kiện hay không. Chương trình cứ tiếp tục thực hiện như vậy cho đến khi gặp một lệnh **break**. Chính vì thế, lệnh **break** được coi là lệnh quan trọng nhất khi dùng **switch.**

Các câu lệnh gắn với **default** sẽ được thực hiện nếu không có **case** nào thỏa mãn. Lệnh **default** là tùy chọn. Nếu không có lệnh **default** và không có **case** nào thỏa mãn, không có hành động nào được thực hiện. Có thể thay đổi thứ tự của **case** và **default.**

7.6. CÁC HÀM THƯ VIỆN TRÌNH DỊCH KEIL C51 HỖ TRỢ

7.6.1. Hàm ngắt

Hàm ngắt là các hàm được dùng với hoạt động ngắt của vi điều khiển, nghĩa là các hàm này sẽ được bộ đếm chương trình PC trỏ tới khi xảy ra ngắt.

Cách khai báo :

```
void  [tên hàm ngắt] (void) interrupt [số hiệu ngắt]
```

Tương ứng với 6 nguồn ngắt trong 8052 thì có các hàm ngắt sau:

```
void  ext0 (void) interrupt 0 //ngắt ngoài 0
      {
      Các câu lệnh;
      }
void  timer0 (void) interrupt 1 //ngắt timer0
      {
      Các câu lệnh;
      }
void ext1 (void) interrupt 2 //ngắt ngoài 1
      {
      Các câu lệnh;
      }
void  timer1 (void) interrupt 3 //ngắt timer1
      {
      Các câu lệnh;
      }
void serialport (void) interrupt 4 /*ngắt cổng nối tiếp */
```

```
        {
        Các câu lệnh;
        }
    void  timer2 (void) interrupt 5 //ngắt timer2
        {
        Các câu lệnh;
        }
```

Chú ý:

- timer0,ext0… là các tên do người lập trình đặt cho các hàm trên, có thể thay đổi nhưng 0, 1, 2, 3, 4, 5 là số hiệu ngắt thì không thể thay đổi
- Đối với 8051 không có hàm ngắt 5, còn các hàm ngắt khác giống như 8052

Ví dụ chương trình có dùng ngắt

```
    #include<reg52.h>
    sbit xung = P1^0;
    void timer1 (void ) interrupt 3 // ngắt timer1 số
hiệu ngắt là 3
    {
    xung =~xung ; // Đảo bít tạo xung
    }
    void main()
    {
    IE=0xFF;
    TMOD=0x20; //Timer1 chế độ 8bit tự động nạp lại
    TH1 = TL1 = -100; // giá trị nạp lại
    TR1  = 1;
    while(1); //không làm gì cả
    }
```

7.6.2. Các hàm thao tác với bộ nhớ đệm

a, Hàm **memccpy**

Cú pháp:

```
    #include <string.h>
    void *memccpy(void *dest,    // buffer đích
    void *scr,      //buffer  nguồn
    char  c,        //ký tự đánh dấu việc kết thúc copy
    int len );       // số byte lớn nhất được coppy
```

Chức năng: copy các bytes từ bộ đệm này tới bộ đệm khác

Ví dụ:

```
#include <string.h>
#include <stdio.h> /* for printf */
void tst_memccpy (void)
{
```

```
  char src1 [100] = "Copy this string to dst1";
  char dst1 [100];
  void *c;
  c = memccpy (dst1, src1, 'g', sizeof (dst1));
  if (c == NULL)
  printf ("'g' was not found in the src buffer\n");
  else
  printf ("characters copied up to 'g'\n");
}
```

b, Hàm **memchr**

Cú pháp:

```
     void  *memchr  (void *buf,   // buffer
     char c,   // byte cần tìm
     int len ;  // số byte lớn nhất sẽ tìm
     )
```

Chức năng: tìm ký tự c trong số len byte đấu tiên của bộ đệm buf

Ví dụ:

```
#include <string.h>
#include <stdio.h> /* for printf */
void tst_memchr (void)
{
  static char src1 [100] = "Search this string from the
start";
  void *c;
  c = memchr (src1, 'g', sizeof (src1));
  if (c == NULL)
    printf ("'g' was not found in the buffer\n");
  else
    printf ("found 'g' in the buffer\n");
}
```

c, Hàm **mmcmp**

Cú pháp:

```
     #include <string.h>
     char memcmp (void *buffer1 ,// buffer thứ nhất
     void *buffer2,          //buffer thứ hai
     int len );
```

Chức năng:

So sánh 2 bộ đệm len byte đầu và trả về giá trị như sau :

Giá trị trả về	Ý nghĩa
<0	buffer1<buffer2
=0	buffer1=buffer2
>0	buffer1>buffer2

Ví dụ :

```
#include <string.h>
#include <stdio.h>
void tst_memcmp ( void )
{
static char hexchars1[] = "0123456789ABCDEF";
static char hexchars2[] = "0123456789abcdef";
char i;
i = memcmp (hexchars ,hecchars2, 16);
if (i<0)
printf("hexchars < hexchars 2\n");
else if (i>0)
printf("hexchars > hexchars 2\n");
else
printf("hexcahars = hexchar2");
}
```

d, Hàm **memcpy**

Cú pháp:

```
#include <string.h>
void *memcpy   (void *dest, // buffer đích
void *src,                  // buffer nguồn
int len );                  // số byte copy
)
```

Chức năng:

Copy len byte từ src tới dest .

Ví dụ:

```
#include <string.h>
#include <stdio.h> /* for printf */
void tst_memchr (void)
{
 static char src1 [100] = "Search this string from
the start";
void *c;
 c = memchr (src1, 'g', sizeof (src1));
if (c == NULL)
printf ("'g' was not found in the buffer\n");
else
printf ("found 'g' in the buffer\n");
}
```

e, Hàm **memmove**

Cú pháp:

```
#include <string.h>
```

```
void *memcpy   (void *dest, //buffer đích
void *src,              //buffer nguồn
int len );              //số byte được di chuyển
)
```

Chức năng:Di chuyển len byte từ src tới dest.

Vi dụ :

```
#include <string.h>
#include <stdio.h>
void tst_memcpy  (void)
{
Static char buf [100] = "this is line 1"
"this is line 1";
"this is line 2";
"this is line 3";
printf ("buf befor   =  %S\n",buf);
memmove (&buf [0], &buf [16], 32);
printf ("buf after = %S\n", buf);
}
```

f, Hàm **memset**

Cú pháp:

```
#include <string.h>
void *memset   (void  * buf,
char c,
int len);
```

Chức năng: Thay len byte đầu tiên trong buf bởi ký tự c.

Ví dụ :

```
#include <string.h>
#include <stdio.h>
void tst_memset (void)
{
char buf [10];
memset   (buf, '\0', sizeof (buf));
}
```

7.6.3. Các hàm định bộ nhớ

a, Hàm **calloc**

Cú pháp:

```
#include <stdio.h>
void *calloc (unsigned int num , //số phần tử của mảng
unsigned int len);              // chiều dài của mảng
```

Chức năng:

Cấp phát bộ nhớ cho num mảng, mỗi mảng gồm len phần tử bắt đầu từ phần tử 0.

Ví dụ :

```
#include <stdio.h>
```

```
#include<string.h>
void tst_calloc (void)
{
int xdata   *p;   /* trỏ tới mảng gồm 100 phần tử */
p = calloc (100,sizeof(int));
if(p==null)
printf("error allocating array\n");
else
printf("array adrress is %p\n",(void * ) p);
}
```

b, Hàm **malloc**

Cú pháp:

```
#include <stdlib.h>
void *malloc (unsigned int size);
```

Mô tả:

Định phần một khối nhờ có kích thước size byte trong không gian nhớ . Giá trị trả về của hàm là già trị của con trỏ nếu như bộ nhớ còn đủ kích thước dành cho khối nhớ yêu cầu, ngược lại giá trị trả về sẽ là 0.

Ví dụ :

```
#include <stdio.h>
#include<string.h>
void tst_malloc  (void)
{
unsigned char xdata *d;
p = malloc (1000);
if (p==null)
printf(" khong co bo nho \n ");
else
printf(" khoi nho da duoc dinh phan\n ");
}
```

c, Hàm **realloc**

Cú pháp:

```
#include <stdio.h>
void *realloc (void xdata *p,   //block cần thay đổi
unsigned int size);     //kích thước mới của block
```

Mô tả :

Thay đổi kích thước của khối nhớ . Biến con trỏ p sẽ trỏ tới khối nhớ cần thay đổi, biến size sẽ là kích thước mới của khối nhớ .

Ví dụ :

```
#include <stdio.h>
#include<string.h>
void tst_realloc (void)
{
```

```
void xdata *p;
void xdata *new_p;
p = malloc   (100);
if (p !=null)
     {
     new_p = realloc (p,200);
     if  (new_p!= null)  p = new_p;
     else printf (reallocation failed\n);
     }
}
```

d, Hàm **_getkey**

Cú pháp:

```
#include<stdio.h>
char _getkey (void);
```

Mô tả :

Chờ nhận một ký tự được gửi đến từ port nối tiếp .

Ví dụ :

```
do
{
x=_getkey();
}
while(x!= 0x41);     // chờ nhận ký tự A
```

e, Hàm **printf**

Cú pháp:

```
#include<stdio.h>
int printf (const char *fmtstr   /* định dạng chuỗi
ký tự*/
[, arguments]…);   // Các biến
```

Mô tả :

Định dạng chuỗi ký tự và ghi (xuất) ra đối tượng được chỉ định trong thủ tục putchar

Ví dụ :

```
#include <stdio.h>
char putchar (char c)
{
if (c ! ='\n')
     {
     line1();
     DATA = c ;
     _nop_();
     _nop_();
     }
if(c =='\n')
     {
```

```
        line1();
        DATA = c;
        _nop_();
        _nop_();
        }
    }
    void main (void)
    {
    char a;
    int b;
    long c;
    unsigned char x;
    unsigned int y;
    unsigned long z;
    float f,g;
    char buf [] ="test string";
    char *p =buf;
    a=1;
    b=12365;
    c=0x7fffffff;
    x=a;
    y=54321;
    z=0x4A6F6E00;
    f=10.0;
    g=22.95;
    printf("char %bd int %d long %ld\n",a,b,c);
    printf("uchar %bu unit %u ulong %lu\n",x,y,z);
    printf("xchar %bx xint %x xlong %lx\n",x,y,z);
    printf("string %s  is at adrress % p\n",buf,p);
    printf("%f != %g\n",f ,g);
    printf("%f * != %*g\n",8,f,8,g);
    while(1);
    }
```

f, Hàm **putchar**

Cú pháp:

```
    #include <stdio.h>
    char putchar (char c);
```

Mô tả :

Truyền ký tự c qua port nối tiếp

Ví dụ :

```
    #include<stdio.h>
    …
    void tst_putchar void()
```

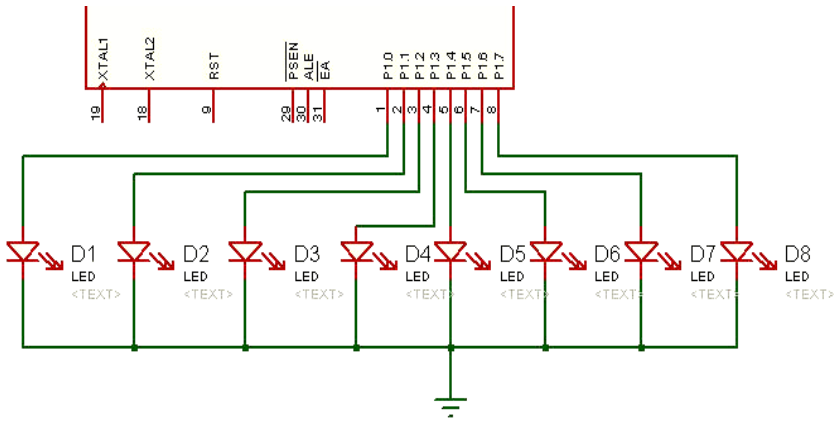
```
{
unsigned char i;
for(i=0x20;i<0x7f;i++)
      putchar(i);
}
```

7.7. MỘT SỐ BÀI TẬP ỨNG DỤNG VIẾT TRÊN TRÌNH BIÊN DỊCH KEIL C51

7.7.1. Điều khiển LED đơn

Để điều khiển LED đơn tạo ra các hiệu ứng thì phải gữ trạng thái của LED một thời gian tương đối dài (ít nhất 0.3s) vì vậy phải có các khoảng thời gian trễ để duy trì trạng thái LED. Các bit vào ra của vi điều khiển chỉ cung cấp dòng nhỏ nên độ sáng các LED không cao nên thực tế điều khiển LED đơn cần có transistor để tăng dòng qua LED. Khi lập trình phải căn cứ phần cứng xem LED sáng khi xung ở chân điều khiển ở mức cao hay thấp. Sau đây là một số ví dụ điều khiển LED đơn tạo các hiệu ứng đơn giản.

Ví dụ 1: Điều khiển led từ LED1 đến LED8 trong sơ đồ hình 7-19 nhấp nháy liên tục với chu kỳ 1s



Hình 7-19

```
#include <REGX51.H>
// Hàm tạo trễ
void delay(unsigned int time)
{
      while(time--)
      {
            int k = 100;
            while(k--);
      }
}
```

```
// Chương trình chính
void main()
{
     while (1)            // Vòng lặp vô tận
     {
          P1=0x00;        // Tắt tất cả các LED
          delay(500);     // Trễ 0.5s
          P1=0xFF;        // Sáng tất cả các LED
          delay (500);    // Trễ 0.5s
     }
}
```

Chương trình trên có thể viết ngắn gọn hơn, trong vòng lặp while(1) chỉ cần khối lệnh sau

```
          {
          P1= ~P1;
          Delay (500);
          }
```

Ví dụ 2: Điều khiển một LED sáng chạy từ LED1 đến LED8 trong sơ đồ hình 7-19 với chu kỳ 4s một vòng.

```
#include <REGX51.H>
#include <INTRINS.H>
void delay(unsigned int time)
{
     while(time--)
     {
          int k = 100;
          while(k--);
     }
}
void main()
{    P1=0x01;
     while (1)
     {
          delay(500);
          P1=_crol_(P1,1); // Xoay P1 sang trái 1 bit
     }
}
```

Ví dụ 3: Điều khiển LED sáng dần từ LED1 đến LED8 trong sơ đồ hình 7-19 với chu kỳ 4s một vòng.

```
#include <REGX51.H>
void delay(unsigned int time)
{
	while(time--)
	{
		int k = 100;
		while(k--);
	}
}
void main()
{
	while (1)
	{
	P1=_crol_(P1,1);
	P1=P1|0x01;
	delay(500);
	if (P1==0xFF)
		{
		P1=0x00;
		delay(500);
		}
	}
}
```

7.7.2. Giao tiếp với LED 7 thanh

Giao tiếp giữa vi điều khiển và LED 7 thanh thường sử dụng trong các ứng dụng hiển thị số như bộ đếm, đồng hồ thời gian … Vi điều khiển kết nối với LED 7 thanh thường sử dụng các phương pháp hiển thị sau:

- Điều khiển trực tiếp từng thanh
- Quét từng số
- Dùng mạch chốt
- Dùng thanh ghi dịch

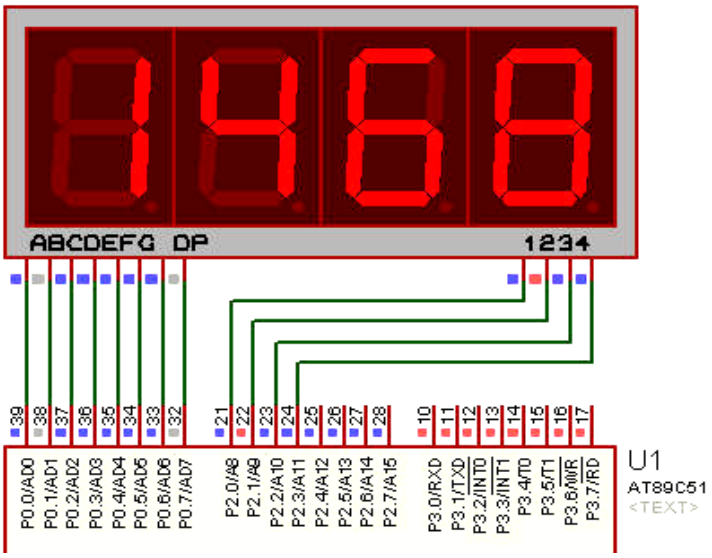
Phương pháp điều khiển trực tiếp từng thanh tốn cổng giao tiếp nên chỉ dùng trong trường hợp hiển thị trên 1 LED 7 thanh, phương pháp dùng thanh ghi dịch phù hợp với các vi điều khiển có chuẩn giao tiếp SPI. Đối với họ 8051 thì nên dùng phương pháp quét hoặc dùng mạch chốt.

a, Phương pháp quét

Đối với các ứng dụng từ 4 số trở xuống thì thường dùng phương pháp quét vì vừa tiết kiệm cổng và các ngoại vi khác mà vẫn đảm bảo độ sáng. Nguyên lý quét LED 7 thanh như sau: Muốn điều khiển n LED 7 đoạn thì dùng n đường dây cung cấp nguồn và 8 đường dây dữ liệu chung cho tất cả các LED. Tại mỗi thời điểm chỉ cho hiển thị một số (Cấp nguồn cho LED tương ứng) và đưa dữ liệu tương ứng của LED đó lên 8 đường dây dữ liệu. Các số lần lượt được hiển thị với tần số trên 10 lần mỗi giây, do hiện tượng lưu ảnh trên võng mạc nên có cảm giác các LED hiển thị liên tục. Sau đây là các ví dụ lập trình hiển thị LED 7 thanh bằng phương pháp quét.

Ví dụ1: Hiển thị một số có 4 chữ số bất kỳ chứa trong biến x ra 4 LED 7 thanh như trong hình 7-20 (Trong thực tế thì dòng điện qua mỗi LED 7 thanh được điều khiển qua 1 transistor để đủ dòng cấp cho LED).

```
#include <REGX51.H>
#include <stdio.h>
#include <math.h>
// Khai báo mảng mã các số từ 0-9 hiển thị trên LED 7 thanh
unsigned char mang[10]={0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92,
0x82, 0xF8, 0x80, 0x90};
unsigned int donvi, chuc, tram, ngin, x;
```

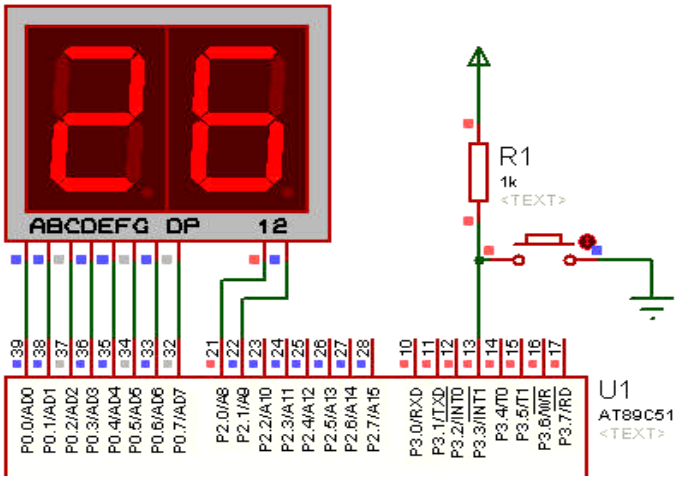


Hình 7-20

```
// Hàm tạo trễ dùng Timer0
void delay (void)
{
      TMOD=0x02;
      TH0=TL0=-100;
      TR0=1;
      while (!TF0);
```

```
      TF0=0;
      TR0=0;
}
void main (void)
{
      x=55*23;              // Giá trị của x theo phép tính
      ngin=x/1000;                // Tính số hàng ngìn
      tram=(x%1000)/100;          // Tính số hàng trăm
      chuc=((x%1000)%100)/10;     // Tính số hàng chục
      donvi=((x%1000)%100)%10;// Tính số hàng đơn vị
      while (1)
      {
            P2=0x08;              // Hiển thị số hàng đơn vị
            P0= mang[donvi];
            delay();
            P2=0x04;              // Hiển thị số hàng chục
            P0= mang[chuc];
            delay();
            P2=0x02;              // Hiển thị số hàng trăm
            P0= mang[tram];
            delay();
            P2=0x01;              // Hiển thị số hàng ngìn
            P0= mang[ngin];
            delay();
       }
}
```

Ví dụ 2: Đếm số lần ấn nút và hiển thị trên 2 LED 7 thanh theo mạch điện hình 7-21



Hình 7-21

```
#include <REGX51.H>
#include <stdio.h>
#include <math.h>
unsigned char ma_led7[10]={0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92,
0x82, 0xF8, 0x80, 0x90};
unsigned char donvi, chuc, x=0;
void delay (void)
{
     TMOD=0x02;
     TH0=TL0=-100;
     TR0=1;
     while (!TF0);
     TF0=0;
     TR0=0;
}
void main (void)
{    IE=0x84;  //Cho phép ngắt ngoài 1
     IT1=1;    // Ngắt bằng sườn âm
     while (1)
     {
          chuc=x/10;
          donvi=x%10;
          P2=0x02;
          P0=ma_led7[donvi];
          delay();
          P2=0x01;
          P0=ma_led7[chuc];
          delay();
          if (x==100) x=0;
      }
}
// Hàm đếm (mỗi lần xảy ra ngắt giá trị x tăng thêm 1)
void ngat(void) interrupt 2
{
     x++;
}
```

b, Phương pháp dùng mạch chốt

Đối với các ứng dụng sử dụng nhiều LED 7 đoạn nếu dùng phương pháp quét thì không đảm bảo độ sáng các LED do đó người ta thường sử dụng các mạch chốt để

giao tiếp với LED 7 đoạn. Khi dùng mạch chốt thì độ sáng LED không phụ thuộc số lượng LED.

Trong thực tế thường dùng vi mạch 74LS374 để làm mạch chốt. Để hiển thị lên LED 7 đoạn thì các LED có chung đường dữ liệu, dữ liệu trên mỗi LED 7 đoạn được lưu giữ nhờ mạch chốt. Hiển thị LED 7 đoạn nào thì đưa dữ liệu LED đó lên đường dữ liệu sau đó tác động sườn dương lên ở chân clock 74LS374 để chốt dữ liệu, sau khi đã chốt thì dữ liệu trên LED không bị ảnh hưởng bởi dữ liệu trên đường dây dữ liệu. Sau đây là ví dụ úng dụng phương pháp chốt để hiển thị LED 7 đoạn.

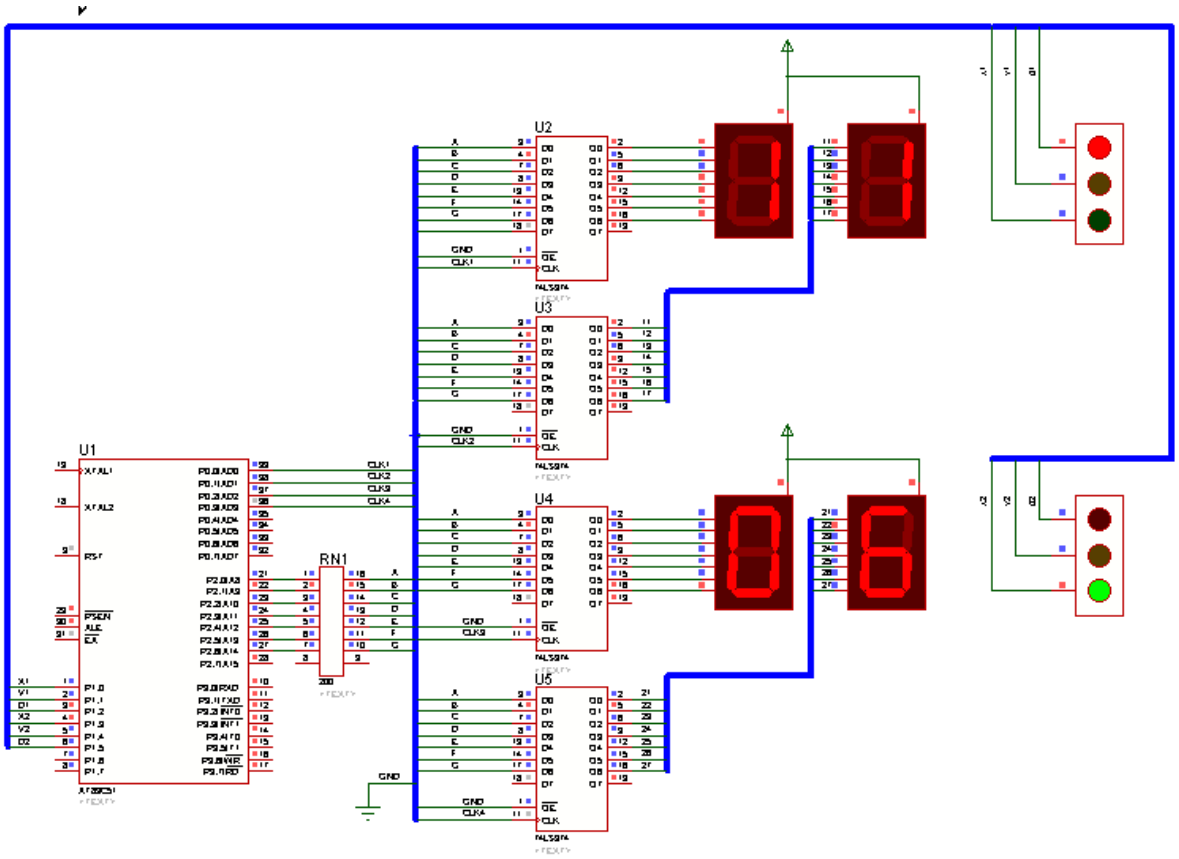
Ví dụ: Điều khiển đèn giao thông có hiển thị đếm lùi thời gian theo sơ đồ mạch điện hình 7-22 với thời gian cho các tuyến đường như sau:

Tuyến 1:

Đỏ 1 (20s) | Xanh 1 (20s) | Vàng 1(5s)

Tuyến 2:

Xanh 2(15s) | Vàng 2 (5s) | Đỏ 2 (25s)



Hình 7-22. Mạch điện điều khiển đèn giao thông

```
#include <REGX51.H>
// Định nghĩa các bit điều khiển các đèn
sbit x1=P1^0;
sbit v1=P1^1;
sbit d1=P1^2;
sbit x2=P1^3;
sbit v2=P1^4;
sbit d2=P1^5;
// Nạp giá trị thời gian sáng của các đèn
unsigned char  t_do1=20, t_xanh1=20, t_do2=25,
               t_xanh2=15, t_vang=5;
// Khai báo Macro điều khiển đèn
#define do1    {d1=x2=1; x1=v1=d2=v2=0;}
#define vang2  {d1=v2=1; x1=v1=d2=x2=0;}
#define do2    {x1=d2=1; d1=v1=x2=v2=0;}
#define vang1  {v1=d2=1; x1=d1=x2=v2=0;}
// Tạo mã LED 7 thanh để hiển thị
ma_led[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
// Hàm tạo trễ khoảng thời gian 1s
void delay()
{
int i,j;
for(i=0;i<500;i++)
     for(j=0;j<300;j++);
}
// Hàm hiển thị đếm lùi thời gian
void hien_thi(unsigned char t1, unsigned char t2)
{
while(t1>0&t2>0)
     {
     P2=ma_led[t1/10];
     P0=0x01;            // Hiển thị hàng chục của tuyến 1
     P2=ma_led[t1%10];
     P0=0x02;            // Hiển thị hàng đơn vị của tuyến 1
     P2=ma_led[t2/10];
     P0=0x04;            // Hiển thị hàng chục của tuyến 2
     P2=ma_led[t2%10];
     P0=0x08;            // Hiển thị hàng đơn vị của tuyến 2
     delay();
     t1--;
     t2--;
```

```
      }
}
void main()
{
P0=P1=0x00;
while (1)
      {
      do1;                              // Đèn đỏ 1 và xanh 2 sáng
      hien_thi(t_do1, t_xanh2);
      vang2;                            // Đèn đỏ 1 và vàng 2 sáng
      hien_thi(t_vang, t_vang);
      do2;                              // Đèn đỏ 2 và xanh 1 sáng
      hien_thi(t_xanh1,t_do2);
      vang1;                            // Đèn đỏ 2 và vàng 1 sáng
      hien_thi(t_vang,t_vang);
      }
}
```

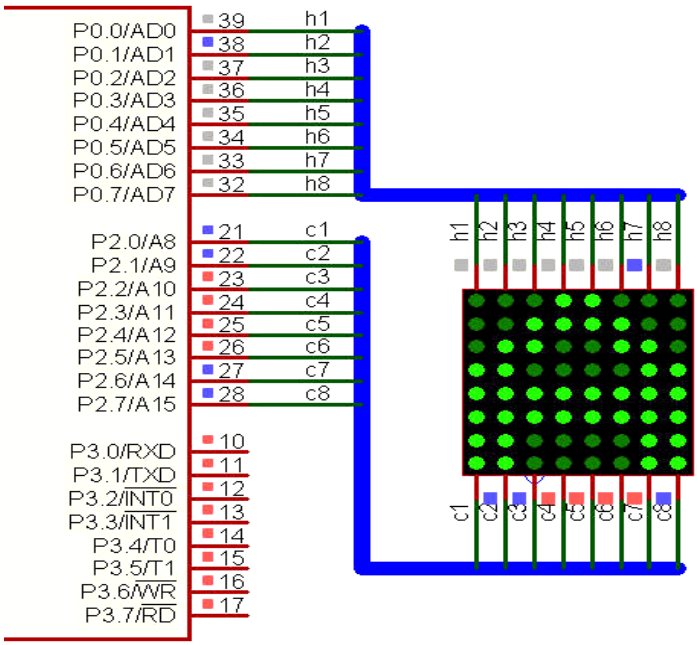
7.7.3. Giao tiếp với LED ma trận

LED ma trận thường được sử dụng để hiển thị chuỗi lý tự, các hình ảnh động, chạy các dòng thông báo… Khi hiển thị trên LED ma trận thì thông thường sử dụng phương pháp quét hàng(hoặc cột). Đầu tiên phải thống kê các LED sáng trên mỗi hàng và tạo mã từng hàng, tại mỗi thời điểm chỉ có một hàng được cấp điện và các LED sáng trên hàng đó được điều khiển bởi cột có mã tương ứng nhập vào, thực hiện việc quét nhanh để mắt không có cảm giác các hàng sáng rời rạc thì trên LED ma trận sẽ hiển thị ký tự hoặc hình ảnh mong muốn.

Khi cần hiển thị LED ma trận với số dòng số cột lớn thì sử dụng thêm các mạch ghi dịch.

Ví dụ 1: Hiển thị chữ A trên LED ma trận 8x8 như trên hình 7-23.
Khi muốn hiển thị chữ A như yêu cầu trong hình 7-23 thì quá trình quét như sau

- Hàng 1: Cột 4 và 5 sáng
- Hàng 2: Cột 3, 4, 5, 6 sáng
- Hàng 3: Cột 2, 3, 6, 7 sáng
- Hàng 4: Cột1, 2, 7, 8 sáng
- Hàng 5 và hàng 6: Tất cả các cột sáng
- Hàng 7 và hàng 8: Cột 1, 2, 7, 8 sáng



Hình 7-23

```
#include <REGX51.H>
#include <stdio.h>
// Hàm tạo trễ
void delay1(void)
{
      TMOD=0x02;
      TH0=TL0=-200;
      TR0=1;
      while (!TF0);
      TF0=0;
      TR0=0;
}
unsigned char n;
// Mã quét hàng
unsigned char ma_hang [8]= {0xFE, 0xFD, 0xFB, 0xF7, 0xEF,
0xDF, 0xBF, 0x7F};
// Mã hiển thị cột
unsigned char ma_cot [8]= {0x18, 0x3C, 0x66, 0xC3, 0xFF, 0xFF,
0xC3, 0xC3};
void main ()
{
      while (1)
            for (n=0; n<=7; n++)      // Thực hiện quét
            {
```

```
                P0=ma_hang[n];
                P2=ma_cot[n];
                delay1();
           }
}
```

Ví dụ 2: Hiển thị chữ A trên LED ma trận 8x8 như trên hình 7-23 sau đó chữ A chạy từ phải qua trái

```
#include <REGX51.H>
#include <stdio.h>
#include <intrins.h>
void delay1()
{
     TMOD=0x02;
     TH0=TL0=-200;
     TR0=1;
     while (!TF0);
     TF0=0;
     TR0=0;
}
// Tạo mã quét hàng
unsigned char ma_hang [8]= {0xFE, 0xFD, 0xFB, 0xF7, 0xEF,
0xDF, 0xBF, 0x7F};
// Tạo mã cột
unsigned char ma_cot [8]= {0x18, 0x3C, 0x66, 0xC3, 0xFF, 0xFF,
0xC3, 0xC3};
unsigned char k, m, n;
void main ()
{
     while (1)
     for (k=0; k<=8; k++)
          for (m=0; m<=100; m++)   // Quét 100 lần
               for (n=0; n<=7; n++)
               {
                    P0=ma_hang[n];
                    P2=ma_cot[n];
                    P2=P2>>k;      // Chạy chữ sang trái
                    delay1();
               }
}
```

7.7.4. Giao tiếp với Text LCD 16x2

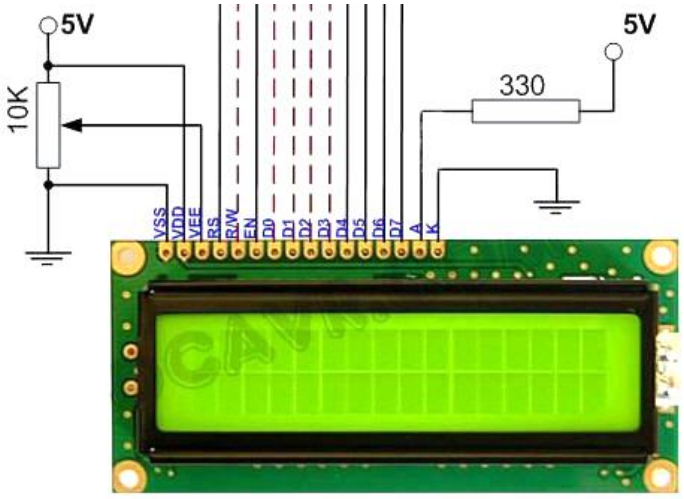
Thiết bị hiển thị LCD (Liquid Crystal Display) được sử dụng trong rất nhiều các ứngdụng của VĐK. LCD có rất nhiều ưu điểm so với các dạng hiển thị khác: Nó có khả năng hiển thị kí tự đa dạng, trực quan (chữ, số và kí tự đồ họa), dễ dàng đưa vào mạch ứng dụng theo nhiều giao thức giao tiếp khác nhau, tốn rất ít tài nguyên hệ thống và giá thành rẻ …

Text LCD là các loại màn hình tinh thể lỏng nhỏ dùng để hiển thị các dòng chữ hoặc số trong bảng mã ASCII. Không giống các loại LCD lớn, Text LCD được chia sẵn thành từng ô và ứng với mỗi ô chỉ có thể hiển thị một ký tự ASCII. Cũng vì lý do chỉ hiện thị được ký tự ASCII nên loại LCD này được gọi là Text LCD (để phân biệt với Graphic LCD có thể hiển thị hình ảnh). Mỗi ô của Text LCD bao gồm các "chấm" tinh thể lỏng, việc kết hợp "ẩn" và "hiện" các chấm này sẽ tạo thành một ký tự cần hiển thị. Trong các Text LCD, các mẫu ký tự được định nghĩa sẵn. Kích thước của Text LCD được định nghĩa bằng số ký tự có thể hiển thị trên 1 dòng và tổng số dòng mà LCD có. Ví dụ LCD 16x2 là loại có 2 dòng và mỗi dòng có thể hiển thị tối đa 16 ký tự. Một số kích thước Text LCD thông thường gồm 16x1, 16x2, 16x4, 20x2, 20x4…

Trong phạm vi giáo trình này chỉ giới thiệu loại giao tiếp song song, cụ thể là LCD 16x2 điều khiển bởi chip HD44780 của hãng Hitachi.

HD44780 là bộ điều khiển cho các Text LCD dạng ma trận điểm (dot-matrix), chip này có thể được dùng cho các LCD có 1 hoặc 2 dòng hiển thị. HD44780U có 2 mode giao tiếp là 4 bit và 8 bit. Nó chứa sẵn 208 ký tự mẫu kích thước font 5x8 và 32 ký tự mẫu font 5x10 (tổng cộng là 240 ký tự mẫu khác nhau).

Sơ đồ chân và chức năng các chân được thể hiện trên hình 7-24 và bảng 7-1



Hình 7-24. Các chân trên Text LCD 16*2

Bảng 7-1. Chức năng các chân trên Text LCD 16x2

Chân số	Tên	Chức năng
1	V_{SS}	Chân nối đất cho LCD, khi thiết kế mạch ta nối chân này với GND
2	V_{DD}	Chân cấp nguồn cho LCD, khi thiết kế mạch ta nối chân này với VCC=5V
3	V_{EE}	Chân này dùng để điều chỉnh độ tương phản của LCD
4	RS	Chân chọn thanh ghi (Register Select). Nối chân RS với logic "0" (GND) hoặc logic "1" (VCC) để chọn thanh ghi. + Logic "0": Bus DB0-DB7 sẽ nối với thanh ghi lệnh IR của LCD (ở chế độ ghi - write) hoặc nối với bộ đếm địa chỉ của LCD (ở chế độ đọc - read) + Logic "1": Bus DB0-DB7 sẽ nối với thanh ghi dữ liệu DR bên trong LCD.
5	R/W	Chân chọn chế độ đọc/ghi (Read/Write). Nối chân R/W với logic "0" để LCD hoạt động ở chế độ ghi, hoặc nối với logic "1" để LCD ở chế độ đọc.
6	EN	Chân cho phép (Enable). Sau khi các tín hiệu được đặt lên bus DB0-DB7, các lệnh chỉ được chấp nhận khi có 1 xung cho phép của chân E. + Ở chế độ ghi: Dữ liệu ở bus sẽ được LCD chuyển vào(chấp nhận) thanh ghi bên trong nó khi phát hiện một xung (high-to-low transition) của tín hiệu chân E. + Ở chế độ đọc: Dữ liệu sẽ được LCD xuất ra DB0-DB7 khi phát hiện cạnh lên (lowto- high transition) ở chân E và được LCD giữ ở bus đến khi nào chân E xuống mức thấp.
7 đến 14	DB0-DB7	Tám đường của bus dữ liệu dùng để trao đổi thông tin với vi điều khiển . Có 2 chế độ sử dụng 8 đường bus này : + Chế độ 8 bit: Dữ liệu được truyền trên cả 8 đường, với bit MSB là bit DB7. + Chế độ 4 bit: Dữ liệu được truyền trên 4 đường từ DB4 tới DB7, bit MSB là DB7
15	A	Anốt và Katốt của LED chiếu sáng LCD trong bóng tối
16	K	

Các thanh ghi trong Text LCD 16*2 như sau:

- Thanh ghi IR: Để điều khiển LCD, người dùng phải "ra lệnh" thông qua tám đường bus DB0-DB7. Mỗi lệnh được nhà sản xuất LCD đánh địa chỉ rõ ràng. Người dùng chỉ việc cung cấp địa chỉ lệnh bằng cách nạp vào thanh ghi IR. Nghĩa là, khi ta nạp vào thanh ghi IR một chuỗi 8 bit, chíp HD44780 sẽ tra bảng mã lệnh tại địa chỉ mà IR cung cấp và thực hiện lệnh đó.

- **Thanh ghi DR**: Thanh ghi DR dùng để chứa dữ liệu 8 bit để ghi vào vùng RAM DDRAM hoặc CGRAM (ở chế độ ghi) hoặc dùng để chứa dữ liệu từ 2 vùng RAM này gởi ra cho vi điều khiển (ở chế độ đọc). Nghĩa là, khi vi điều khiển ghi thông tin vào DR, mạch nội bên trong chíp sẽ tự động ghi thông tin này vào DDRAM hoặcCGRAM. Hoặc khi thông tin về địa chỉ được ghi vào IR, dữ liệu ở địa chỉ này trong vùng RAM nội của HD44780 sẽ được chuyển ra DR để truyền cho vi điều khiển. Bằng cách điều khiển chân RS và R/W chúng ta có thể chuyển qua lại giữ 2 thanh ghi này khi giao tiếp với vi điều khiển . Bảng 7-2 tóm tắt lại các thiết lập đối với hai chân RS và R/W theo mục đích giao tiếp.

Bảng 7-2. Thiết lập các chế độ bằng chân RS và R/W

RS	R/W	Khi sử dụng
0	0	Ghi vào thanh ghi IR để ra lệnh cho LCD
0	1	Đọc cờ bận ở DB7 và giá trị của bộ đếm địa chỉ ở DB0-DB6
1	0	Ghi vào thanh ghi DR
1	1	Đọc dữ liệu từ DR

- **Cờ báo bận BF (Busy Flag).** Khi thực hiện các hoạt động bên trong chíp, mạch nội bên trong cần một khoảng thời gian để hoàn tất. Khi đang thực thi các hoạt động bên trong chip như thế, LCD bỏ qua mọi giao tiếp với bên ngoài và bật cờ BF (thông qua chân DB7 khi có thiết lập RS=0, R/W=1) lên để báo cho vi điều khiển biết nó đang bận và khi xong việc nó sẽ xóa cờ BF lại mức 0.

- **Bộ đếm địa chỉ AC (Address Counter).** Thanh ghi IR không trực tiếp kết nối với vùng RAM (DDRAM và CGRAM) mà thông qua bộ đếm địa chỉ AC. Bộ đếm này lại nối với 2 vùng RAM theo kiểu rẽ nhánh. Khi một địa chỉ lệnh được nạp vào thanh ghi IR, thông tin được nối trực tiếp cho 2 vùng RAM nhưng việc chọn lựa vùng RAM tương tác đã được bao hàm trong mã lệnh.

Sau khi ghi vào (đọc từ) RAM, bộ đếm AC tự động tăng lên (giảm đi) 1 đơn vị và nội dung của AC được xuất ra cho MCU thông qua DB0-DB6 khi có thiết lập RS=0 và R/W=1 (xem bảng tóm tắt RS - R/W).

Lưu ý: Thời gian cập nhật AC không được tính vào thời gian thực thi lệnh mà được cập nhật sau khi cờ BF lên mức cao (not busy), cho nên khi lập trình hiển thị phải delay một khoảng thời gian khoảng 4uS- 5uS (ngay sau khi BF=1) trước khi nạp dữ liệu mới.

- **Vùng RAM hiển thị DDRAM (Display Data RAM).** Đây là vùng RAM dùng để hiển thị, nghĩa là ứng với một địa chỉ của RAM là một ô kí tự trên màn hình

và khi ghi vào vùng RAM này một mã 8 bit thì LCD sẽ hiển thị tại vị trí tương ứng trên màn hình một kí tự có mã 8 bit mà ta đã cung cấp.

- Vùng RAM chứa kí tự đồ họa CGRAM (Character Generator RAM). Nhà sản xuất dành vùng có địa chỉ byte cao là 0000 để người dùng có thể tạo các mẫu kí tự đồ họa riêng. Tuy nhiên dung lượng vùng này rất hạn chế: Ta chỉ có thể tạo 8 kí tự loại 5x8 điểm ảnh, hoặc 4 kí tự loại 5x10 điểm ảnh.

Để lập trình hiển thị trên LCD 16*2 thì ngoài việc nắm sơ đồ chân, các thanh ghi thì còn phải nắm được mã lệnh của nó. Bảng 7-3 thống kê tập lệnh điều khiển Text LCD 16*7 theo chuẩn của HD44780

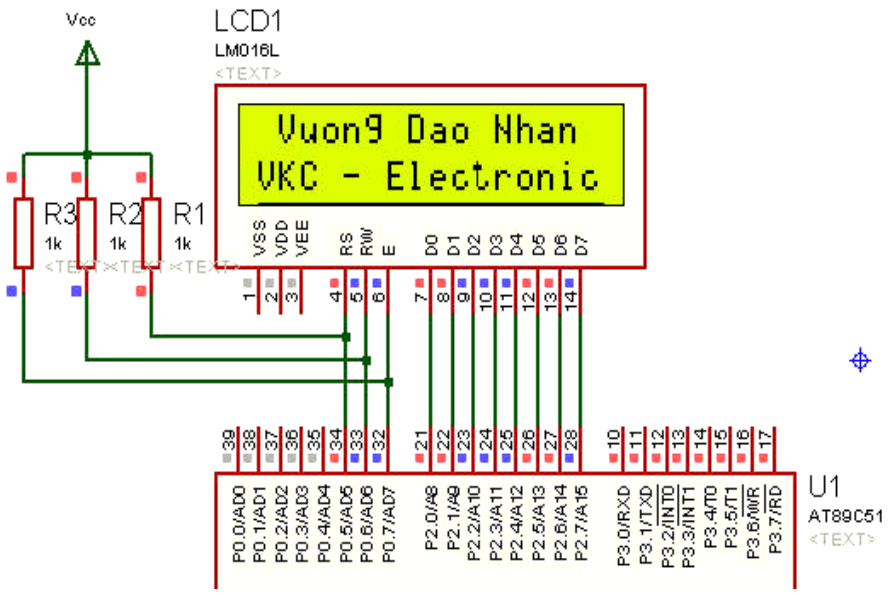
Bảng 7-3. Các lệnh điều khiển LCD

Tên lệnh	Hoạt động	Thời gian thực hiện
Clear Display	**Mã lệnh:** DBx: DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 0 0 0 0 0 0 0 1 Lệnh Clear Display (xóa hiển thị) sẽ ghi một khoảng trống-blank (mã hiển kí tự 20h) vào tất cả ô nhớ trong DDRAM. Nghĩa là xóa hiển thị, con trỏ dời về góc trái hàng đầu tiên.	1.52µs
Return home	**Mã lệnh:** DBx: DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 0 0 0 0 0 0 1 * Lệnh Return home (Con trỏ về dòng trên cùng bên trái) . Nội dung của DDRAM không thay đổi.	1.52µs
Entry mode set	**Mã lệnh:** DBx DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 0 0 0 0 0 1 [I/D] [S] **I/D:** Tăng (I/D=1) hoặc giảm (I/D=0) bộ đếm địa chỉ hiển thị AC 1 đơn vị mỗi khi có hành động ghi hoặc đọc vùng DDRAM. Vị trí con trỏ cũng di chuyển theo sự tăng giảm này. **S:** Khi S=1 toàn bộ nội dung hiển thị bị dịch sang phải (I/D=0) hoặc sang trái (I/D=1) mỗi khi có hành động ghi vùng DDRAM. Khi S=0: không dịch nội dung hiển thị. Nội dung hiển thị không dịch khi đọc DDRAM hoặc đọc/ghi vùng CGRAM.	37µs
Display on/off control	**Mã lệnh:** DBx: DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 0 0 0 0 1 [D] [C] [B] **D:** Hiển thị màn hình khi D=1 và ngược lại. Khi tắt hiển thị, nội dung DDRAM không thay đổi.	37µs

<table>
<tr><td></td><td>C: Hiển thị con trỏ khi C=1 và ngược lại.
B: Nhấp nháy kí tự tại vị trí con trỏ khi B=1 và ngược lại.</td><td></td></tr>
<tr><td>Cursor or display shift</td><td>Mã lệnh: DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
DBx = 0 0 0 1 [S/C] [R/L] * *
Lệnh Cursor or display shift dịch chuyển con trỏ hay dữ liệu hiển thị sang trái mà không cần hành động ghi/đọc dữ liệu. Chi tiết sử dụng xem bảng bên dưới:
<table>
<tr><th>S/C</th><th>R/L</th><th>Hoạt động</th></tr>
<tr><td>0</td><td>0</td><td>Dịch vị trí con trỏ sang trái</td></tr>
<tr><td>0</td><td>1</td><td>Dịch vị trí con trỏ sang phải</td></tr>
<tr><td>1</td><td>0</td><td>Dịch toàn bộ nội dung hiển thị sang trái, con trỏ cũng dịch theo.</td></tr>
<tr><td>1</td><td>1</td><td>Dịch toàn bộ nội dung hiển thị sang phải, con trỏ cũng dịch theo.</td></tr>
</table></td><td>37µs</td></tr>
<tr><td>Function set</td><td>Mã lệnh: DBx: DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
DBx = 0 0 1 [DL] [N] [F] * *
DL: Khi DL=1, LCD giao tiếp với vi điều khiểnbằng giao thức 8 bit (từ bit DB7 đến DB0). Ngược lại, giao thức giao tiếp là 4 bit (từ bit DB7 đến bit DB0). Khi chọn giao thức 4 bit, dữ liệu được truyền/nhận 2 lần liên tiếp. với 4 bit cao gởi/nhận trước, 4 bit thấp gởi/nhận sau.
N: Thiết lập số hàng hiển thị. Khi N=0: hiển thị 1 hàng, N=1: hiển thị 2 hàng.
F: Thiết lập kiểu kí tự. Khi F=0: kiểu kí tự 5x8 điểm ảnh, F=1: kiểu kí tự 5x10 điểm ảnh.
* Chú ý:
- Chỉ thực hiện thay đổi Function set ở đầu chương trình. Và sau khi được thực thi 1 lần, lệnh thay đổi Function set không được LCD chấp nhận nữa ngoại trừ thiết lập chuyển đổi giao thức giao tiếp.
- Không thể hiển thị kiểu kí tự 5x10 điểm ảnh ở kiểu hiển thị 2 hàng</td><td>37µs</td></tr>
<tr><td>Set CGRAM address</td><td>Mã lệnh DB: DB7DB6 DB5 DB4 DB3 DB2 DB1 DB0
DBx = 0 1 [ACG][ACG][ACG][ACG][ACG][ACG]
Lệnh này ghi vào AC địa chỉ của CGRAM. Kí hiệu [ACG] chỉ 1 bit của chuỗi dữ liệu
6 bit. Ngay sau lệnh này là lệnh đọc/ghi dữ liệu từ CGRAM tại</td><td>37µs</td></tr>
</table>

	địa chỉ đã được chỉ định.	
Set DDRAM address	**Mã lệnh:** DBx: DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 1 [AD] [AD] [AD] [AD] [AD] [AD] [AD] Lệnh này ghi vào AC địa chỉ của DDRAM, dùng khi cần thiết lập tọa độ hiển thị mong muốn. Ngay sau lệnh này là lệnh đọc/ghi dữ liệu từ DDRAM tại địa chỉ đã được chỉ định. Khi ở chế độ hiển thị 1 hàng: địa chỉ có thể từ 00h đến 4Fh. Khi ở chế độ hiển thị 2 hàng, địa chỉ từ 00h đến 27h cho hàng thứ nhất, và từ 40h đến 67h cho hàng thứ 2.	37µs
Read BF and address	**Mã lệnh:** DBx: DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = [BF] [AC] [AC] [AC] [AC] [AC] [AC] [AC] (RS=0, R/W=1) Như đã đề cập trước đây, khi cờ BF bật, LCD đang làm việc và lệnh tiếp theo (nếu có) sẽ bị bỏ qua nếu cờ BF chưa về mức thấp. Cho nên, khi lập trình điều khiển phải kiểm tra cờ BF trước khi ghi dữ liệu vào LCD. Khi đọc cờ BF, giá trị của AC cũng được xuất ra các bit [AC]. Nó là địa chỉ của CG hay DDRAM là tùy thuộc vào lệnh trước đó	0
Write data to CG or DDRAM	**Mã lệnh:** DBx: DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = [Write data] (RS=1, R/W=0) Khi thiết lập RS=1, R/W=0, dữ liệu cần ghi được đưa vào các chân DBx từ mạch ngoài sẽ được LCD chuyển vào trong LCD tại địa chỉ được xác định từ lệnh ghi địa chỉ trước đó (lệnh ghi địa chỉ cũng xác định luôn vùng RAM cần ghi) Sau khi ghi, bộ đếm địa chỉ AC tự động tăng/giảm 1 tùy theo thiết lập Entry mode. Lưu ý là thời gian cập nhật AC không tính vào thời gian thực thi lệnh.	37µs
Read data from CG or DDRAM	**Mã lệnh:** DBx: DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = [Read data] (RS=1, R/W=1) Khi thiết lập RS=1, R/W=1,dữ liệu từ CG/DDRAM được chuyển ra MPU thông qua các chân DBx (địa chỉ và vùng RAM đã được xác định bằng lệnh ghi địa chỉ trước đó). Sau khi đọc, AC tự động tăng/giảm 1 tùy theo thiết lập Entry mode, tuy nhiên nội dung hiển thị không bị dịch bất chấp chế độ Entry mode.	37µs

Ví dụ: Thực hiện giao tiếp AT89C51 với LCD 16x2 theo chuẩn Hitachi HD44780 để hiển thị chuỗi ký tự và giá trị của biến số x (hình 7-25)



Hình 7-25

```
#include <REGX51.H>
// Định nghĩa Port truyền dữ liệu ra LCD
#define lcd_data    P2
// Định nghĩa các bit điều khiển LCD
sbit lcd_RS = P0^5;
sbit lcd_RW = P0^6;
sbit lcd_EN = P0^7;
sbit busy_flag = P2^7;
// Khai báo các Macro thực hiện các chế độ của LCD
#define read_mode   {lcd_RS = 0; lcd_RW = 1;}
#define write_mode  {lcd_RS = 0; lcd_RW = 0;}
#define write_data  {lcd_RS = 1; lcd_RW = 0;}
#define latch_data  {lcd_EN = 1; lcd_EN = 0;}
// Khai báo các Macro lệnh LCD
#define lcd_clear         0x01
#define lcd_home          0x02
#define lcd_off           0x08
#define lcd_display_on    0x0C
#define lcd_blink_cursor 0x0E
#define lcd_font_8bits    0x38
#define lcd_font_4bits    0x28
#define lcd_move_right    0x1C
#define lcd_move_left     0x18
```

```
// Hàm kiểm tra cờ bận (Busy flag)
void wait_ready(void)
{
     read_mode;               // Đọc trạng thái LCD
     busy_flag = 1;
     while(busy_flag)
     {
          lcd_EN = 0;
          lcd_EN = 1;
     }
}

//Hàm tạo trễ
void delay(unsigned int time)
{
     while(time--)
     {
          int k = 100;
          while(k--);
     }
}
//Hàm ghi lệnh điều khiển lên LCD
void lcd_command(unsigned char command)
{
     wait_ready();       // Chờ LCD sẵn sàng
     write_mode;         // LCD ở chế độ ghi dữ liệu
     lcd_data = command; // Đưa lệnh điều khiển lên LCD
     latch_data;         // Chốt dữ liệu
}
//Hàm ghi một ký tự lên LCD
void write_char(unsigned char ch)
     {
          wait_ready();
          write_data;
          lcd_data = ch;
          latch_data;
     }
// Hàm ghi chuỗi ký tự lên LCD
void write_string(char *str)
     {
          while(*str!='\0')   // Thực hiện đến kết thúc chuỗi
```

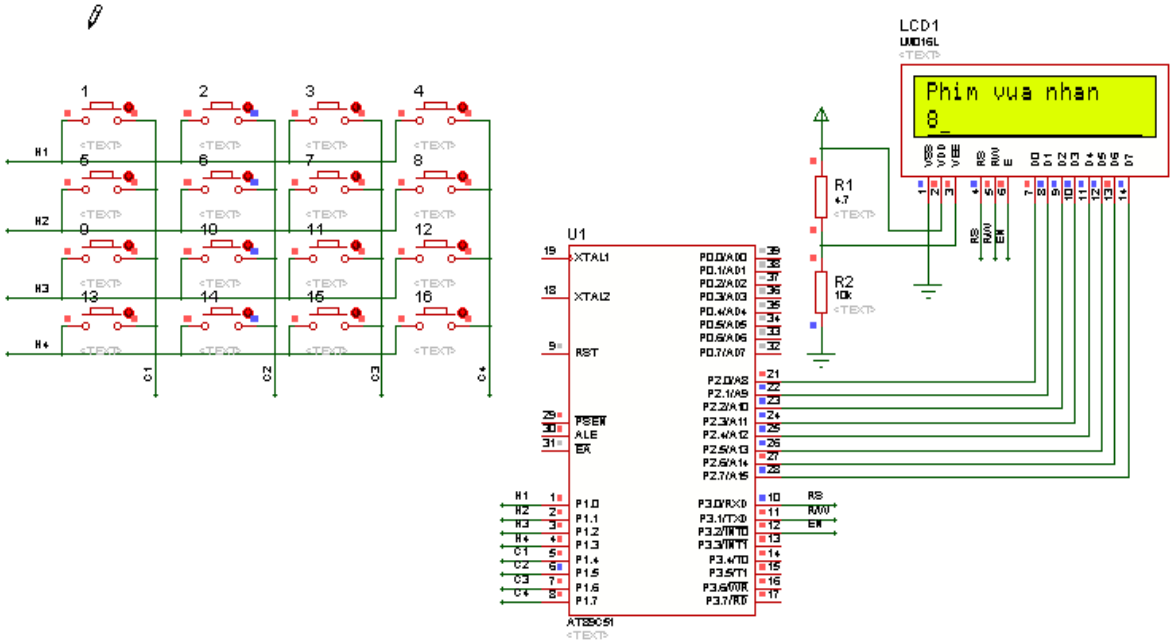
```
          {
               write_char(*str);
               str++;
               delay(150);
          }
     }
// Hàm đưa con trỏ đến vị trí cột x dòng y
//Cot x=0 đến 15, dong y=0 đến 1
void gotoxy_lcd(unsigned char x,unsigned char y)
{
     if(y==0)       lcd_command(x+0x80);
     else if(y==1)  lcd_command(x+0xC0);
}
// Hàm khởi tạo chế độ hoạt động LCD
void lcd_init()
{
     lcd_command(lcd_font_8bits);
     lcd_command(lcd_clear);
     lcd_command(lcd_display_on);
     lcd_command(lcd_blink_cursor);
}
//Hàm hiển thị giá trị chứa trong biến số lên LCD
void write_number(unsigned int number)
{
     unsigned int  temp;                 // Biến trung gian
     unsigned char buffer_number[4];
     buffer_number[0]=number/1000;       //Số hàng ngìn
     temp=number%1000;
     buffer_number[1]=temp/100;          //Số hàng trăm
     temp=temp%100;
     buffer_number[2]=temp/10;           //Số hàng chục
     buffer_number[3]=temp%10;           //Số hàng đơn vị
     for(temp=0;temp<4;temp++)           // Ghi dãy 4 số
     write_char(buffer_number[temp]+0x30);
}
unsigned char str1[16]= "Vuong Dao Nhan";
unsigned char str2[16]= "VKC - Electronic";
int x=34*256;
void main(void)
{
     lcd_init();
```

```
        gotoXY_lcd(1,0);
        write_string(str1);       // Hiển thị chuỗi ký tự str1
        gotoXY_lcd(0,1);
        write_string(str2);       // Hiển thị chuỗi ký tự str2
        lcd_command(lcd_clear);
        write_number(x);          // Hiển thị giá trị x
        delay(500);
}
```

7.7.5 Giao tiếp với ma trận phím 4x4

Nguyên lí quét phím: Các hàng lần lượt được đưa xuống mức thấp (0V). Khi phím nào được nhấn thì cột tương ứng sẽ xuống mức thấp và căn cứ hàng đang quét và cột được xuống mức thấp để xem phím nào được nhấn.

Ví dụ: Hiển thị phím được nhấn trong ma trận phím 4x4 lên LCD 16x2 (Hình 7-26)



Hình 7-26

```
        #include<AT89x51.h>
        #include<string.h>
        #define H1 P1_0
        #define H2 P1_1
        #define H3 P1_2
        #define H4 P1_3
        #define LCDdata P2;.
        sbit BF = 0xA7;
        sbit RS = P3^0;
        sbit RW = P3^1;
        sbit EN = P3^2;
unsigned char t=0;
```

```
void wait(void)
{
     RS=0;
     RW=1;
     LCDdata=0xff;
     while(BF)
          {
          EN=0;
          EN=1;
          }
}
void LCDcontrol(unsigned char x)
{
     RS=0;
     RW=0;
     LCDdata=x;
     EN=1;
     EN=0;
     wait();
 }
void LCDinit(void)
{
     LCDcontrol(0x38);
     LCDcontrol(0x0e);
     LCDcontrol(0x01);
}
void LCDwrite(unsigned char c)
 {
     RS=1;
     RW=0;
     LCDdata=c;
     EN=1;
     EN=0;
     wait();
}
void LCDwrites(unsigned char *s)
 {   unsigned char data lens,count;
     lens=strlen(s);
     for(count=0;count<lens;count++)
     LCDwrite(*(s+count));
 }
```

```
void gotoXY_lcd(unsigned char x,unsigned char y)
{
     if(y==0)       LCDcontrol(x+0x80);
     else if(y==1)  LCDcontrol(x+0xC0);
}

void  delay(long  time)
{
     while(time--);
}

unsigned char Phim(void)
{
     unsigned char phimnhan, dem, chot;
     phimnhan=0;
     chot=128;
     dem=4;
     while(dem--)
     {
          P1=~chot;
          if(!H1)
          {
               delay(500);
               phimnhan=dem+1;
               delay(500);
          }
          else if(!H2)
          {
               delay(500);
               phimnhan=dem+5;
               delay(500);
          }
          else if(!H3)
          {
               delay(500);
               phimnhan=dem+9;
               delay(500);
          }
          else if(!H4)
```

```
            {
                  delay(500);
                  phimnhan=dem+13;
                  delay(500);
            }
      chot>>=1;
}
return phimnhan;
}

void viewphim(unsigned char phimnhan)
{
      if(phimnhan<10)
      {
            LCDwrite(phimnhan+48);
      }
      else
      {
            LCDwrite(phimnhan/10+48);
            LCDwrite(phimnhan%10+48);
      }
      if (t==10)t=0;
}
      void main(void)
      {
            unsigned char phimnhan=0;
            int a[]={0,1,2,3,4,5,6,7,8,9};
            LCDinit();
            LCDwrites("Moi nhan phim");
            LCDcontrol(0xC0);
            while(1)
            {     phimnhan=Phim();
                  if(phimnhan)
                  {
                        LCDcontrol(0x01);
                        LCDwrites("Phim vua nhan");
                        gotoXY_lcd(a[t],1);
                        viewphim(phimnhan);
                  }
            }
      }
```

7.7.6. Tạo xung

Ví dụ 1: Tạo xung có tần số 10KHz trên chân P1.1, thạch anh được dùng có tần số 12MHz.

Trong ví dụ này thì sau 50µs (tương ứng 50 chu kỳ máy.) mức điện áp đầu ra tại chân P1.1 đảo mức một lần. Sau 50 chu kỳ máy. thì đảo mức nên chỉ cần Timer 8 bit là đáp ứng được.

```
#include<reg51.h>
sbit xung=P1^1;
void main (void)
{
     TMOD=0x02;      //Timer0 ở mode 2
     TH0=-50;        //Hệ số chia là 100
     TR0=1;          //Cho Timer0 chạy
     while(1)        //Vòng lặp vô hạn
     {
          while(!TF0);         // Chờ tràn Timer0
          TF0=0;               // Xóa cờ tràn
          xung=~xung;          // Đảo mức điện áp chân P1.1
     }
}
```

Ví dụ 2: Tạo xung có tần số 200Hz trên chân P1.1, thạch anh được dùng có tần số 12MHz.

Trong ví dụ này thì sau 2500µs (tương ứng 2500 chu kỳ máy.) mức điện áp đầu ra tại chân P1.1 đảo mức một lần. Sau 2500 chu kỳ máy. thì đảo mức nên cần Timer 16 bit mới đáp ứng được.

```
#include<reg51.h>
sbit xung=P1^1;
void main (void)
{
     TMOD=0x01;      //Timer0 ở mode 1
     While(1)
     {
     TH0=-2500/256; //Hệ số chia là 5000
     TL0=-2500%256
     TR0=1;          //Cho Timer0 chạy
     while(!TF0);    // Chờ tràn Timer0
     TF0=0;          // Xóa cờ tràn
     TR0=0;          // Dừng Timer0
```

```
      xung=~xung      // Đảo mức điện áp chân P1.1
      }
}
```

Ví dụ 3: Tạo hai xung khác nhau trên các chân P1.0 và P1.1

```
#include<reg51.h>
#include<stdio.h>
sbit xung1=P1^0;
sbit xung2=P1^1;
void main (void)
{
      TMOD=0x21;      // Timer0 mode1, Timer1 mode2
      TH1=TL1=-100;   // Xung có chu kỳ 200µs
      TR1=1;
// Khởi tạo các thanh ghi ngắt và mức ưu tiên ngắt
      IE=0x8A;        // Cho phép ngắt do Timer0 và Timer1
      IP=0;           // Mức ưu tiên ngắt bằng nhau
      TF0=1;          // Buộc ngắt do Timer1
      while (1);
```

Ví dụ 4: Tạo xung PWM: Tạo xung có độ rộng thay đổi theo thứ tự 25%; 50% và 75% trên chân P1.1.

```
#include <REGX51.H>
sbit F=P1^0;
void  rong_xung (float tyso);
void main(void)
{
      TMOD=0x01;      // Timer0 chế độ16 bit
      while (1)
      {
            rong_xung(0.25);
            rong_xung(0.5);
            rong_xung(0.75);
      }
}
void rong_xung(float tyso)
{
      F=1;
      TR0=0;
      TH0=-(unsigned int) (tyso*1000)/256;
```

```
      TL0=-(unsigned int) (tyso*1000)%256;
      TR0=1;
      while(!TF0);
      TF0=0;
      F=0;
      TR0=0;
      TH0=-(unsigned int) ((1-tyso)*1000)/256;
      TL0=-(unsigned int) ((1-tyso)*1000)%256;
      TR0=1;
      while(!TF0);
      TF0=0;
}
```

7.7.7. Truyền thông qua cổng nối tiếp

Ví dụ 1: Truyền một byte qua cổng nối tiếp

```
#include<reg51.h>
#include<stdio.h>
char x;
void main (void)
{
      SCON=0x52;           // Cổng nối tiếp ở mode 1
      TMOD=0x20;           // Timer1 ở mode 2
      TH1 = TL1 = -3;      // Tốc độ Baud là 9600, dùng thạch
                           anh 11,0592MHz
      TR1=1;
      while(!TI);     // Chờ TI=1
      TI=0;           // Xóa TI
      SBUF=x;         // Truyền byte dữ liệu trong biến x
}
```

Ví dụ 2: Nhận một byte qua cổng nối tiếp

```
#include<reg51.h>
#include<stdio.h>
char x;
void main (void)
{
      SCON=0x52;           // Cổng nối tiếp ở mode 1
      TMOD=0x20;           // Timer1 ở mode 2
      TH1 = TL1 = -3;      // Tốc độ Baud là 9600, dùng thạch
                           anh 11,0592MHz
```

```
    TR1=1;
    while(!RI);     // Chờ RI=1
    RI=0;           // Xóa RI
    x=SBUF;         // Nhận byte dữ liệu trong biến x
}
```

Trong 2 ví dụ trên hện số chia của tốc độ Baud được mặc định là 32 do không có câu lệnh đặt SMOD=1 (Khi đó SMOD sẽ mặc định bằng 0).

Trong một chương trình khi có nhiều thao tác truyền nhận thì đoạn chương trình truyền hoặc nhận nên viết dưới dạng hàm như sau

```
void truyen (char x)
{
    while (!TI);
    TI=0;
    SBUF=x;
}
void nhan (void)
{
    char c;
    while (!RI);
    RI=0;
    c=SBUF;
    return c; // Giá trị trả về là c
}
```

TÀI LIỆU THAM KHẢO

[1] Nguyễn Tăng Cường, Phan Quốc Thắng (2004), *Cấu trúc và lập trình vi điều khiển*, Nhà xuất bản Khoa học Kỹ thuật, Hà Nội.

[2] Nguyễn Mạnh Giang (2009), *Cấu trúc - Lập trình - Ghép nối và ứng dụng vi điều khiển,* Nhà xuất bản Giáo dục, Hà Nội.

[3] Tống Văn On (2005), *Thiết kế hệ thống với họ 8051*, Nhà xuất bản Phương Đông, TP Hồ Chí Minh.

[4] Tống Văn On, Hoàng Đức Hải (2008), *Họ vi điều khiển 8051*, Nhà xuất bản Lao động – Xã hội, TP Hồ Chí Minh.

[5] Ngô Diên Tập (2005), *Vi điều khiển với lập trình C*, Nhà xuất bản Khoa học Kỹ thuật, Hà Nội.

[6] Kiều Xuân Thực (Chủ biên) (2005), *Vi điều khiển, cấu trúc - lập trình và ứng dụng*, Nhà xuất bản Giáo dục, Hà Nội.

[7] Atmel 8051 Microcontrollers Hardware Manual on Website www.atmel.com

[8] Datasheet 89C51; 89C52 on Website www.atmel.com

[9] www.keil.com