SỞ LAO ĐỘNG - THƯƠNG BINH VÀ XÃ HỘI HÀ NỘI

TRƯỜNG TRUNG CẤP CÔNG NGHỆ VÀ DU LỊCH HÀ NỘI

GIÁO TRÌNH

MÔN ĐUN: VI ĐIỀU KHIỂN

NGHỀ: CÔNG NGHỆ KỸ THUẬT ĐIỆN – ĐIỆN TỬ

TRÌNH ĐỘ: TRUNG CẤP

(Ban hành kèm theo Quyết định số: 32/QĐ-CNDL ngày 28 tháng 02 năm 2023 của Hiệu trưởng Trường Trung cấp Công nghệ và Du lịch Hà Nội)

Hà Nội, năm 2023

LỜI NÓI ĐẦU

Cùng với các mô đun của ngành Công nghệ kỹ thuật điện - điện tử, mô đun Vi điều khiển là mô đun kỹ thuật chuyên ngành quan trọng của ngành điện tử, hiện nay mô đun được ứng dụng rộng rãi trong ngành kỹ thuật và các lĩnh vực điều khiển khác.

Mô đun được ứng dụng cho tất cả học viên ngành Công nghệ kỹ thuật điện - điện tử của trường ta. Bởi vậy để tạo điều kiện cho việc học tập và nghiên cứu mô đun của học viên được thuận lợi trong quá trình học tập. Bộ môn Điện - điện tử thuộc Khoa Kỹ thuậ Điện – Công nghệ trường Trung cấp Công nghệ và Du lịch Hà Nội tổ chức biên soạn tài liệu: " ***Vi điều khiển*** " làm bài giảng lưu hành hội bộ.

Trong quá trình biên soạn chắc chắn sẽ không tránh khỏi những thiếu sót, bởi vậy tôi mong nhận được sự thông cảm và góp ý chân thành của các bạn đồng nghiệp để cho giáo trình ngày càng hoàn thiện hơn.

Xin chân thành cảm ơn!

MỤC LỤC

BÀI 1: SƠ LƯỢC VỀ LỊCH SỬ VÀ HƯỚNG PHÁT TRIỂN CỦA VI ĐIỀU KHIỂN 1. Lịch sử phát triển

Bộ Vi xử lý có khả năng vượt bậc so với các hệ thống khác về khả năng tính toán, xử lý, và thay đổi chương trình linh hoạt theo mục đích người dùng, đặc biệt hiệu quả đối với các bài toán và hệ thống lớn.Tuy nhiên đối với các ứng dụng nhỏ, tầm tính toán không đòi hỏi khả năng tính toán lớn thì việc ứng dụng vi xử lý cần cân nhắc. Bởi vì hệ thống dù lớn hay nhỏ, nếu dùng vi xử lý thì cũng đòi hỏi các khối mạch điện giao tiếp phức tạp như nhau. Các khối này bao gồm bộ nhớ để chứa dữ liệu và chương trình thực hiện, các mạch điện giao tiếp ngoại vi để xuất nhập và điều khiển trở lại, các khối này cùng liên kết với vi xử lý thì mới thực hiện được công việc. Để kết nối các khối này đòi hỏi người thiết kế phải hiểu biết tinh tường về các thành phần vi xử lý, bộ nhớ, các thiết bị ngoại vi. Hệ thống được tạo ra khá phức tạp, chiếm nhiều không gian, mạch in phức tạp và vấn đề chính là trình độ người thiết kế. Kết quả là giá thành sản phẩm cuối cùng rất cao, không phù hợp để áp dụng cho các hệ thống nhỏ.

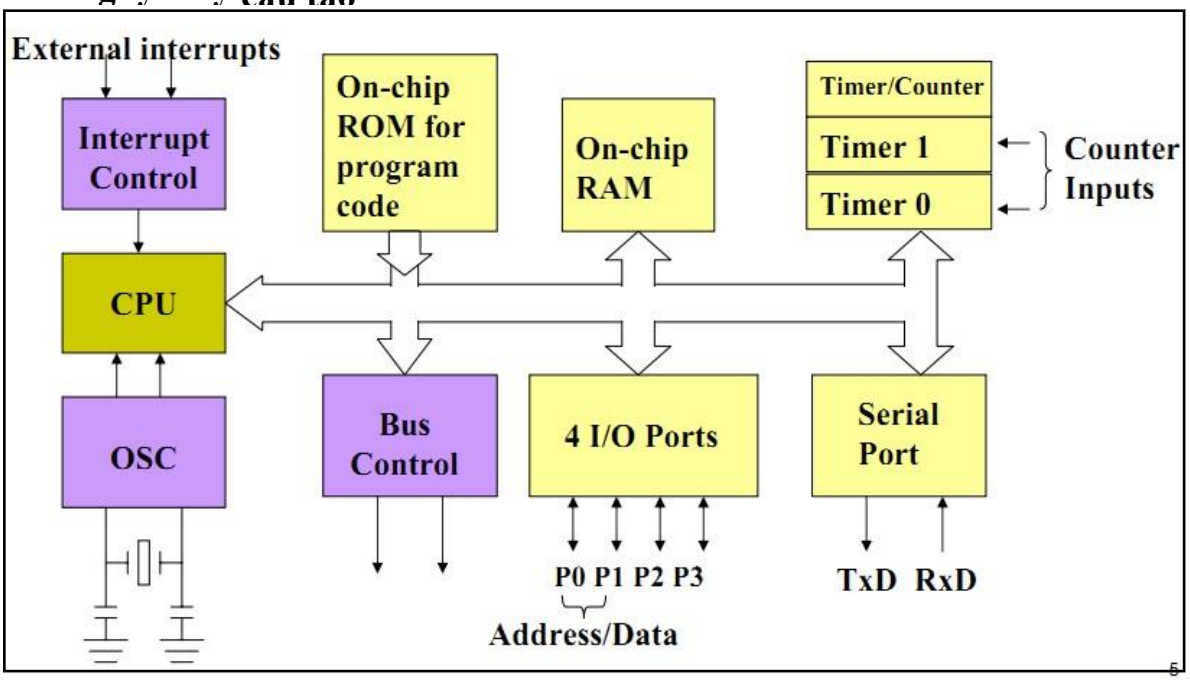
Vì một số nhược điểm trên nên các nhà chế tạo tích hợp một ít bộ nhớ và một số mạch giao tiếp ngoại vi cùng với vi xử lý vào một IC duy nhất được gọi là Microcontroller-Vi điều khiển. Vi điều khiển có khả năng tương tự như khả năng của vi xử lý, nhưng cấu trúc phần cứng dành cho người dùng đơn giản hơn nhiều. Vi điều khiển ra đời mang lại sự tiện lợi đối với người dùng, họ không cần nắm vững một khối lượng kiến thức quá lớn như người dùng vi xử lý, kết cấu mạch điện dành cho người dùng cũng trở nên đơn giản hơn nhiều và có khả năng giao tiếp trực tiếp với các thiết bị bên ngoài. Vi điều khiển tuy được xây dựng với phần cứng dành cho người sử dụng đơn giản hơn, nhưng thay vào lợi điểm này là khả năng xử lý bị giới hạn (tốc độ xử lý chậm hơn và khả năng tính toán ít hơn, dung lượng chương trình bị giới hạn). Thay vào đó, Vi điều khiển có giá thành rẻ hơn nhiều so với vi xử lý, việc sử dụng đơn giản, do đó nó được ứng dụng rộng rãi vào nhiều ứng dụng có chức năng đơn giản, không đòi hỏi tính toán phức tạp.

Vi điều khiển được ứng dụng trong các dây chuyền tự động loại nhỏ, các robot có chức năng đơn giản, trong máy giặt, ôtô v.v...

Năm 1976 Intel giới thiệu bộ vi điều khiển (microcontroller) 8748, một chip tương tự như các bộ vi xử lý và là chip đầu tiên trong họ MCS-48. Độ phức tạp, kích thước và khả năng của Vi điều khiển tăng thêm một bậc quan trọng vào năm 1980 khi intel tung ra chip 8051, bộ Vi điều khiển đầu tiên của họ MCS-51 và là chuẩn công nghệ cho nhiều họ Vi điều khiển được sản xuất sau này. Sau đó rất nhiều họ Vi điều khiển của nhiều nhà chế tạo khác nhau lần lượt được đưa ra thị trường với tính năng được cải tiến ngày càng mạnh.

2. Vi điều khiển

2.1. Nguyên lý cấu tạo



Điểm cần lưu ý về vi điều khiển là sơ đồ khối cấu tạo của nó. Cấu tạo một họ microcontroller chủ yếu dựa trên một kiểu tiêu chuẩn bao gồm các tính năng quan trọng nhất, nhiều chủng loại phù hợp với các lĩnh vực ứng dụng đặc biệt khác nhau, có thể kết hợp thêm thiết bị ngoại vi để tăng khả năng hoặc giảm nhỏ kích thước đến mức tối thiểu trong các ứng dụng chuyên biệt như: Kết nối bus, kết nối video hoặc điều khiển trực tiếp các cơ cấu hiển thị LCD...Với kiểu tiêu chuẩn cũng đủ dùng cho hầu hết các ứng dụng

2.2. Các kiểu cấu trúc bộ nhớ

Các bộ vi xử lý và vi điều khiển hiện nay chủ yếu được chế tạo theo 2 kiểu cấu trúc khác nhau: Cấu trúc Von Neumann và Harvard.

2.2.1. Cấu trúc Von Neumann

Trong cấu trúc Von Neumann chỉ có một vùng địa chỉ tuyến tính bao gồm tất cả dữ liệu và lệnh điều khiển, độ lớn của vùng địa chỉ phụ thuộc vào chiều dài của bộ đếm chương trình, nếu không trang bị thêm linh kiện phụ thì việc định địa chỉ bộ nhớ chương trình và bộ nhớ dữ liệu không độc lập với nhau. Trong cấu trúc này chỉ tồn tại một bus dữ liệu và một bus địa chỉ để đọc-ghi dữ liệu và đọc lệnh điều khiển chương trình và không có khả năng thực hiện song song (truy xuất đồng thời bộ nhớ dữ liệu và bộ nhớ chương trình)

2.2.2. Cấu trúc Harvard

Gồm hai vùng địa chỉ riêng biệt cho bộ nhớ dữ liệu và bộ nhớ chương trình nên có thể truy xuất song song dữ liệu và lệnh điều khiển, cấu trúc này đặc biệt thích hợp với các vi điều khiển 16 và 32 bít vì làm tăng tốc độ làm việc. Nếu chỉ có một hệ thống bus như thường thấy ở vi điều khiển 8 bít thì việc truy xuất bộ nhớ dữ liệu hoặc bộ nhớ chương trình sẽ được thực hiện thông qua các tín hiệu điều khiển, nếu không có yêu cầu ghi vào bộ nhớ chương trình thì cấu trúc này còn cho phép tăng tính an toàn của chương trình

3. Lĩnh vực và ứng dụng

- Hầu hết các thiết bị điện tử hiện nay đều sử dụng các chip vi điều khiển - Ứng dụng trong nhiều lĩnh vực khác nhau:

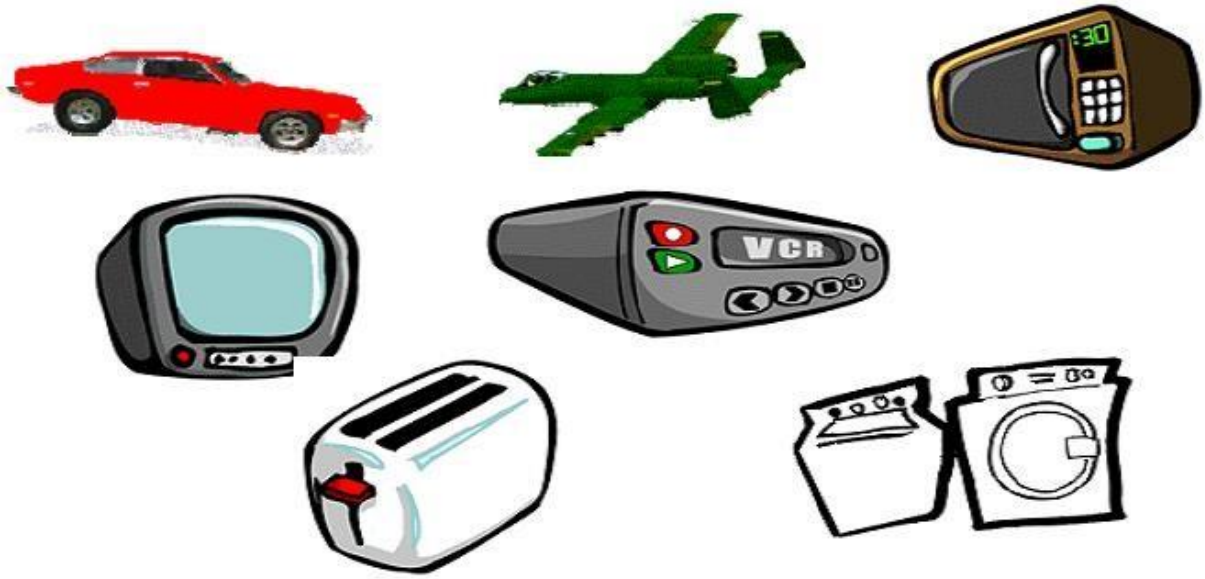
+ Điện thoại di động

+ Các thiết bị đo lường điện tử

+ Tivi, CD players, radio,… +
Bàn phím PC

+ Các hệ thống bảo vệ, báo cháy….

+ Các máy in



4. Hướng phát triển

Yêu cầu đặt ra cho vi điều khiển hiện nay là tăng lĩnh vực ứng dụng với tốc độ xử lý ngày càng nhanh và kích thước nhỏ gọn, công suất tiêu thụ thấp. Vấn đề đặt ra là liệu với vi điều khiển 8 bít có còn phù hợp hay không ? hoặc trong tương lai phải thay bằng các vi điều khiển 16/32 bít.

Khác với vi xử lý việc phát triển luôn kèm theo việc nâng cao khả năng tính toán bằng cách mở rộng hệ thống bus. Đối với vi điều khiển không nhất thiết phải như thế, một vi điều khiển 8 bít cũng đủ cho rất nhiều ứng dụng và vi điều khiển 16 bít là hoàn toàn quá dư thừa, trong trường hợp cần giảm giá thành, kích thước và công suất tiêu thụ thì vi điều khiển 4 bít là giải pháp tối ưu. Một vài ứng dụng cần vi điều khiển có nhiều khối ngoại vi, có ứng dụng lại cần ngoại vi tốc độ cao, hướng phát triển tương lai là tăng khả năng của CPU và khối ngoại vi

Một hướng đơn giản là tăng tần số xung đồng hồ để rút ngắn thời gian thực hiện chương trình, giảm thời gian biến đổi A/D và tăng tần số giới hạn của timer. Tuy nhiên các linh kiện bên ngoài cũng phải có khả năng làm việc ở tần số cao, khi tăng tần số đồng cũng làm tăng công suất tiêu thụ của vi điều khiển

Việc tối ưu hóa cấu trúc chương trình và bộ nhớ cũng góp phần nâng cao khả năng hệ thống. Trong các ứng dụng đa nhiệm, phương pháp phân đoạn và phân dãy hóa có một ý nghĩa rất lớn

BÀI 2: CẤU TRÚC HỌ VI ĐIỀU KHIỂN 8051

1. Tổng quan

Các thành viên của họ MCS-51 (Atmel) có các đặc điểm chung như sau:

+ Có 4/8/12/20 Kbyte bộ nhớ FLASH ROM bên trong để lưu chương trình. Nhờ vậy Vi điều khiển có khả năng nạp xoá chương trình bằng điện đến 10000 lần.

+ 128 Byte RAM nội

+ 4 Port xuất/nhập 8 bit

+ Từ 2 đến 3 bộ định thời 16-bit

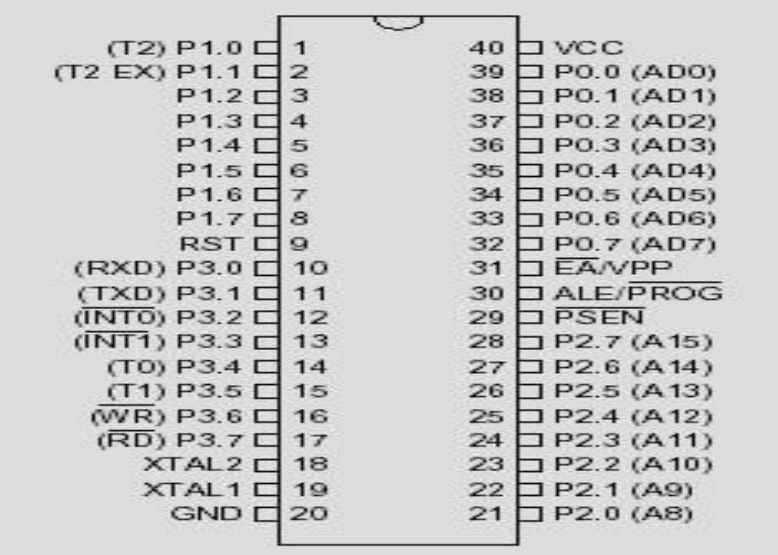
+ Có khả năng giao tiếp truyền dữ liệu nối tiếp

+ Có thể mở rộng không gian nhớ chương trình ngoài 64KByte (bộ nhớ ROM ngoại): *khi chương trình do người lập trình viết ra có dung lượng lớn hơn dung lượng bộ nhớ ROM nội, để lưu được chương trình này cần bộ nhớ ROM lớn hơn, cách giải quyết là kết nối Vi điều khiển với bộ nhớ ROM từ bên ngoài (hay còn gọi là ROM ngoại). Dung lượng bộ nhớ ROM ngoại lớn nhất mà Vi điều khiển có thể kết nối là 64Kbyte*

+ Có thể mở rộng không gian nhớ dữ liệu ngoài 64KByte (bộ nhớ RAM ngoại)

+ Bộ xử lí bit (thao tác trên các bit riêng rẽ) 210 bit có thể truy xuất đến từng bit

2. Sơ đồ chân



Hình 2.1. Sơ đồ chân vi điều khiển 8051

* Port 0 (P0)

Port 0 gồm 8 chân (từ chân 32 đến 39) có hai chức năng:

- Chức năng xuất/nhập :*các chân này được dùng để nhận tín hiệu từ bên ngoài vào để xử lí, hoặc dùng để xuất tín hiệu ra bên ngoài, chẳng hạn xuất tín hiệu để điều khiển led đơn sáng tắt.*

- Chức năng là bus dữ liệu và bus địa chỉ (AD7-AD0) : *8 chân này (hoặc Port 0) còn làm nhiệm vụ lấy dữ liệu từ ROM hoặc RAM ngoại (nếu có kết nối với bộ nhớ ngoài), đồng thời Port 0 còn được dùng để định địa chỉ của bộ nhớ ngoài.*

* Port 1 (P1)

- Port P1 gồm 8 chân (từ chân 1 đến chân 8), chỉ có chức năng làm các đường xuất/nhập, không có chức năng khác.

* Port 2 (P2)

Port 2 gồm 8 chân (từ chân 21 đến chân 28) có hai chức năng:

- Chức năng xuất/nhập

- Chức năng là bus địa chỉ cao (A8-A15): khi kết nối với bộ nhớ ngoài có dung lượng lớn,cần 2 byte để định địa chỉ của bộ nhớ, byte thấp do P0 đảm nhận, byte cao do P2 này đảm nhận. * Port 3 (P3)

Port 3 gồm 8 chân (từ chân 10 đến 17):

- Chức năng xuất/nhập

- Với mỗi chân có một chức năng riêng thứ hai như trong bảng sau

Bit	**Tên**	**Chức năng**
P3.0	RxD	Ngõ vào nhận dữ liệu nối tiếp
P3.1	TxD	Ngõ xuất dữ liệu nối tiếp
P3.2	INT0	Ngõ vào ngắt cứng thứ 0
P3.3	INT1	Ngõ vào ngắt cứng thứ 1
P3.4	T0	Ngõ vào của Timer/Counter thứ 0
P3.5	T1	Ngõ vào của Timer/Counter thứ 1
P3.6	WR	Ngõ điều khiển ghi dữ liệu lên bộ nhớ ngoài
P3.7	RD	Ngõ điều khiển đọc dữ liệu từ bộ nhớ bên ngoài
P1.0	T2	Ngõ vào của Timer/Counter thứ 2

P1.1	T2X	Ngõ Nạp lại/thu nhận của Timer/Counter thứ 2

* Chân cho phép bộ nhớ chương trình

- PSEN (program store enable) tín hiệu được xuất ra ở chân 29 dùng để truy xuất bộ nhớ chương trình ngoài. Chân này thường được nối với chân OE (output enable) của ROM ngoài.

- Khi vi điều khiển làm việc với bộ nhớ chương trình ngoài, chân này phát ra tín hiệu kích hoạt ở mức thấp và được kích hoạt 2 lần trong một chu kì máy

- Khi thực thi một chương trình ở ROM nội, chân này được duy trì ở mức logic không tích cực (logic 1)

(Không cần kết nối chân này khi không sử dụng đến)

* Chân cho phép chốt địa chỉ ALE

- Khi Vi điều khiển truy xuất bộ nhớ từ bên ngoài, port 0 vừa có chức năng là bus địa chỉ, vừa có chức năng là bus dữ liệu do đó phải tách các đường dữ liệu và địa chỉ. Tín hiệu ở chân ALE dùng làm tín hiệu điều khiển để giải đa hợp các đường địa chỉ và các đường dữ liệu khi kết nối chúng với IC chốt.

- Các xung tín hiệu ALE có tốc độ bằng 1/6 lần tần số dao động đưa vào Vi điều khiển, như vậy có thể dùng tín hiệu ở ngõ ra ALE làm xung clock cung cấp cho các phần khác của hệ thống.

- Ghi chú: khi không sử dụng có thể bỏ trống chân này

* Chân truy suất ROM ngoài (EA)

- Chân EA dùng để xác định chương trình thực hiện được lấy từ ROM nội hay ROM ngoại.

- Khi EA nối với logic 1(+5V) thì Vi điều khiển thực hiện chương trình lấy từ bộ nhớ nội

- Khi EA nối với logic 0(0V) thì Vi điều khiển thực hiện chương trình lấy từ bộ nhớ ngoại

* Chân Reset

- Ngõ vào RST ở chân 9 là ngõ vào Reset dùng để thiết lập trạng thái ban đầu cho vi điều khiển. Hệ thống sẽ được thiết lập lại các giá trị ban đầu nếu ngõ này ở mức 1 tối thiểu 2 chu kì máy.

* Các chân Xtal 1 và Xtal 2

- Hai chân này có vị trí chân là 18 và 19 được sử dụng để nhận nguồn xung clock từ bên ngoài để hoạt động, thường được ghép nối với thạch anh và các tụ để tạo nguồn xung clock ổn định.

* Chân cấp nguồn

- Chân số 40 là VCC cấp điện áp nguồn cho Vi điều khiển - Nguồn điện cấp là +5V±0.5.

- Chân số 20 nối GND(hay nối Mass)

3. Cấu trúc Port I/O

- Port 0 (Chân 32 đến 39): P0 (P0.0 đến P0.7)
- Port 1 (Chân 1 đến 8): P1 (P1.0 đến P1.7)
- Port 2 (Chân 21 đến 28): P2 (P2.0 đến P2.7)
- Port 3 (Chân 10 đến 17): P3 (P3.0 đến P3.7)

Mỗi cổng có 8 chân

- Đánh tên từ P0.X (X = 0, 1...7), P1.X, P2.X, P3.X
- Với P0.0 là bít 0 (LSB) của P0

  P0.7 là bít 7 (MSB) của P0
- 8 bít này cấu thành một byte
- Mỗi cổng có thể được dùng như input hay output

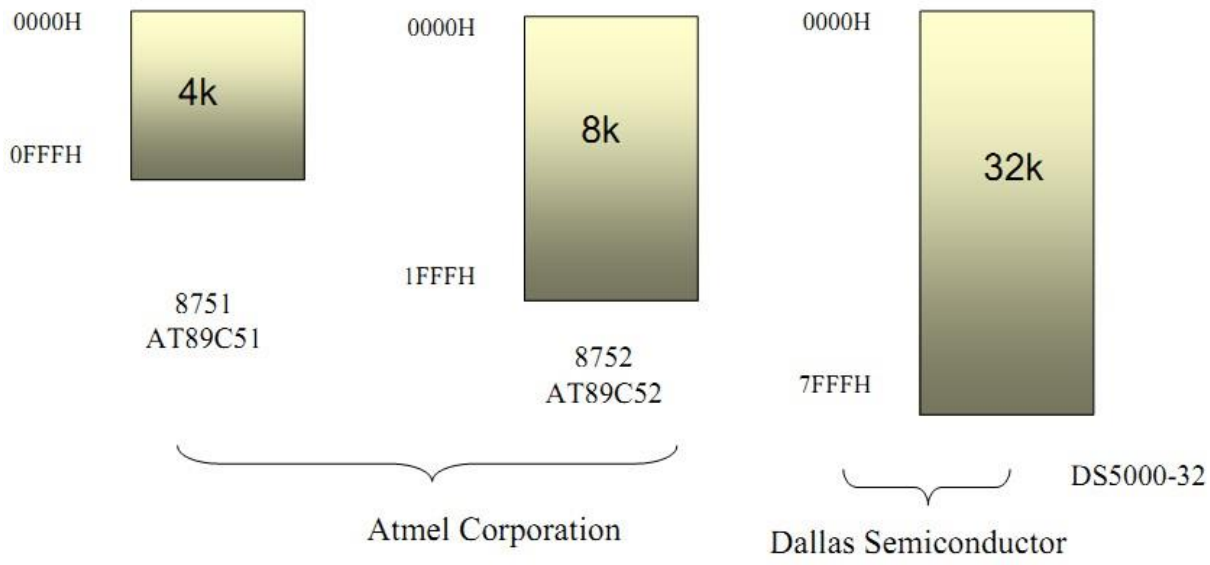
4. Tổ chức bộ nhớ

4.1. Bộ nhớ chương trình (ROM)

Bộ nhớ ROM dùng để lưu chương trình do người viết chương trình viết ra. Chương trình là tập hợp các câu lệnh thể hiện các thuật toán để giải quyết các công việc cụ thể, chương trình do người thiết kế viết trên máy vi tính, sau đó được đưa vào lưu trong ROM của vi điều khiển, khi hoạt động, vi điều khiển truy xuất từng câu lệnh trong ROM để thực hiện chương trình. ROM còn dùng để chứa số liệu các bảng, các tham số hệ thống, các số liệu cố định của hệ thống. Trong quá trình hoạt động nội dung ROM là cố định, không thể thay đổi, nội dung ROM chỉ thay đổi khi ROM ở chế độ xóa hoặc nạp chương trình (do các mạch điện riêng biệt thực hiện).

- Bộ nhớ ROM được tích hợp trong chip Vi điều khiển với dung lượng tùy vào chủng loại cần dùng, chẳng hạn đối với 89S52 là 8KByte, với 89S53 là 12KByte.

- Bộ nhớ bên trong Vi điều khiển 89Sxx là bộ nhớ Flash ROM cho phép xóa bộ nhớ ROM bằng điện và nạp vào chương trình mới cũng bằng điện và có thể nạp xóa nhiều lần



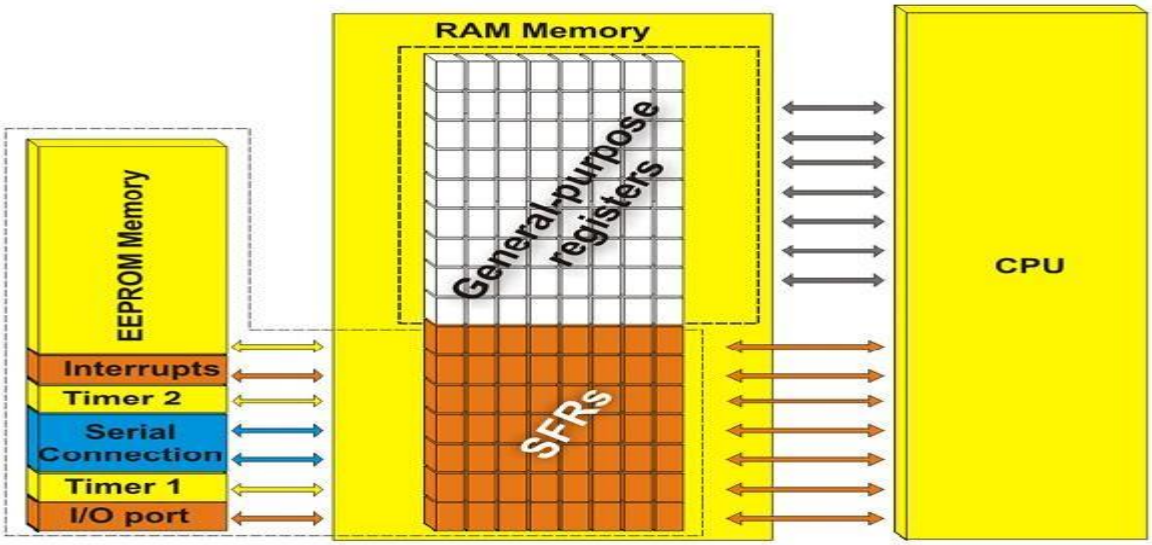
Hình 2.2. Tổ chức bộ nhớ ROM của 8051

Bộ nhớ ROM được định địa chỉ theo từng Byte, các byte được đánh địa chỉ theo số hex-số thập lục phân, bắt đầu từ địa chỉ 0000H, khi viết chương trình cần chú ý đến địa chỉ lớn nhất trên ROM, chương trình được lưu sẽ bị mất khi địa chỉ lưu vượt qua vùng này. Ví dụ: AT89S52 có 8KByte bộ nhớ ROM nội, địa chỉ lớn nhất là 1FFFH, nếu chương trình viết ra có dung lượng lớn hơn 8KByte các byte trong các địa chỉ lớn hơn 1FFFH sẽ bị mất.

- Ngoài ra Vi điều khiển còn có khả năng mở rộng bộ nhớ ROM với việc giao tiếp với bộ nhớ ROM bên ngoài lên đến 64KByte(địa chỉ từ 0000H đến FFFFH)

4.2. Bộ nhớ dữ liệu

Bộ nhớ RAM dùng làm môi trường xử lý thông tin, lưu trữ các kết quả trung gian và kết quả cuối cùng của các phép toán, xử lí thông tin. Nó cũng dùng để tổ chức các vùng đệm dữ liệu, trong các thao tác thu phát, chuyển đổi dữ liệu.



Hình 2.3. Giao tiếp bộ nhớ

* Vùng RAM đa năng

Mặc dù hình 2.4 trình bày vùng RAM đa năng có 80 byte đặt ở địa chỉ từ 30H đến 7FH, bên dưới vùng này từ địa chỉ 00H đến 2FH là vùng nhớ có thể được sử dụng tương tự (mặc dù vùng này có công dụng khác). Bất kỳ vị trí nhớ nào trong vùng RAM đa năng đều có thể được truy xuất tự do bằng cách sử dụng cách định địa chỉ trực tiếp hoặc gián tiếp. Ví dụ để đọc nội dung tại 5FH của RAM nội vào thanh ghi A có thể dùng lệnh sau MOV A, 5FH

Lệnh trên di chuyển một byte dữ liệu bằng cách dùng kiểu định địa chỉ trực tiếp để xác định vị trí nguồn (nghĩa là địa chỉ 5FH). Đích của dữ liệu được xác định rõ ràng trong mã lệnh là thanh ghi A

Vùng RAM đa năng còn có thể được truy xuất bằng cách dùng kiểu định địa chỉ gián tiếp qua các thanh ghi R0, R1. Ví dụ hai lệnh sau thực hiện cùng công việc như ở ví dụ trên.

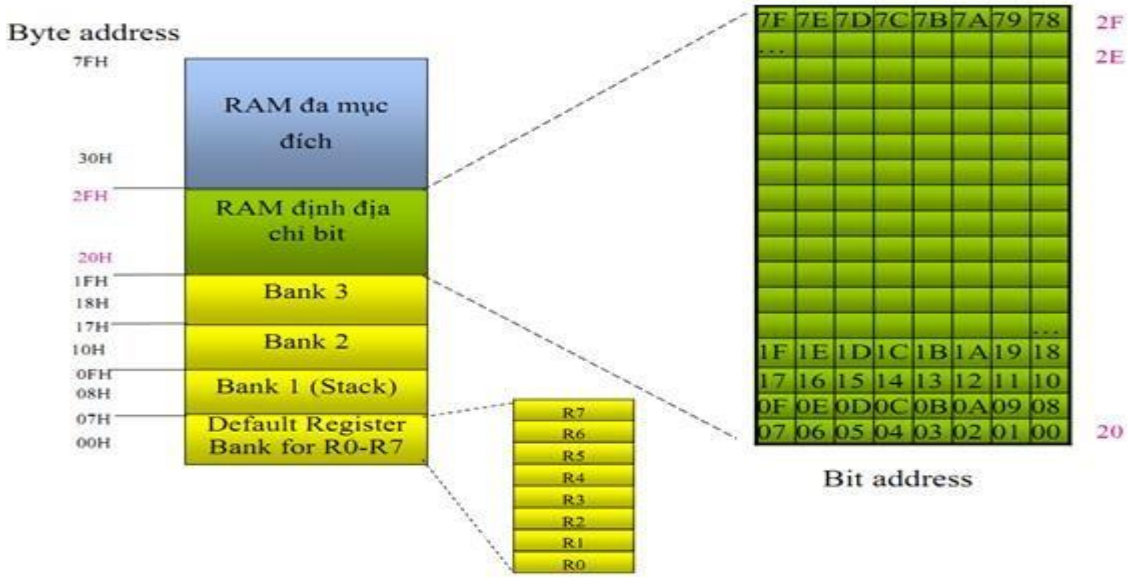
```
MOV R0,#5FH
MOV A,@R0
```

Lệnh đầu tiên sử dụng kiểu định địa chỉ tức thời di chuyển giá trị 5FH vào thanh ghi R0, lệnh tiếp theo sử dụng cách định địa chỉ gián tiếp di chuyển dữ liệu trỏ bởi R0 vào thanh ghi A

7F	Vùng RAM đa năng								FF									
									F0	F7	F6	F5	F4	F3	F2	F1	F0	B
									E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
									D0	D7	D6	D5	D4	D3	D2	-	D0	PSW
30									B8	-	-	-	BC	BB	BA	B9	B8	IP
2F	7F	7E	7D	7C	7B	7A	79	78										
2E	77	76	75	74	73	72	71	70	B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
2D	6F	6E	6D	6C	6B	6A	69	68										
2C	67	66	65	64	63	62	61	60	A8	AF	-	-	AC	AB	AA	A9	A8	IE
2B	5F	5E	5D	5C	5B	5A	59	58										
2A	57	56	55	54	53	52	51	50	A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
29	4F	4E	4D	4C	4B	4A	49	48										
28	47	46	45	44	43	42	41	40	99	Địa chỉ byte								SBUF
27	3F	3E	3D	3C	3B	3A	39	38	98	9F	9E	9D	9C	9B	9A	99	98	SCON
26	37	36	35	34	33	32	31	30										
25	2F	2E	2D	2C	2B	2A	29	28	90	97	96	95	94	93	92	91	90	P1
24	27	26	25	24	23	22	21	20										
23	1F	1E	1D	1C	1B	1A	19	18	8D	Địa chỉ byte								TH1
22	17	16	15	14	13	12	11	10	8C	Địa chỉ byte								TH0
21	0F	0E	0D	0C	0B	0A	09	08	8B	Địa chỉ byte								TL1
20	07	06	05	04	03	02	01	00	8A	Địa chỉ byte								TL0
1F	Bank 3								89	Địa chỉ byte								TMOD
18									88	8F	8E	8D	8C	8B	8A	89	88	TCON
17	Bank 2								87	Địa chỉ byte								PCON
10																		
0F	Bank 1								83	Địa chỉ byte								DPH
08									82	Địa chỉ byte								DPL
07	Bank 0 (mặc định) R0-R7								81	Địa chỉ byte								SP
00									80	87	86	85	84	83	82	81	80	P0

Hình 2.4. Cấu trúc RAM nội của 8051

* Vùng RAM địa chỉ bit



Hình 2.5. Vùng RAM định địa chỉ bit

Trên RAM nội có 210 ô nhớ bit được định địa chỉ và có thể truy xuất đến từng bit, các bit nhớ này cũng được định địa chỉ bằng các số thập lục phân- số Hex. Trong đó có 128 bit nằm trong các ô nhớ có địa chỉ byte từ 20H đến 2FH, các bit nhớ còn lại chứa trong nhóm thanh ghi có chức năng đặc biệt.

Mặc dù các bit nhớ và ô nhớ (byte) cùng được định bằng số Hex, tuy nhiên chúng sẽ được nhận dạng là địa chỉ bit hay địa chỉ byte thông qua các câu lệnh tương ứng dành cho các bit nhớ hoặc các ô nhớ này.

Ví dụ:

Mov 05H,#10111111B; >>> lệnh này thiết lập giá trị cho ***ô nhớ*** có địa chỉ là 05H

JB 05H,nhan01; >>> lệnh này liên quan đến trạng thái của ***bit nhớ*** có địa chỉ 05H

* Các dãy thanh ghi

Các thanh ghi này được định địa chỉ byte, một số được định thêm địa chỉ bit, có địa chỉ của các thanh ghi này nằm trong khoảng 80H đến FFH. Các thanh ghi đặc biệt này này được dùng để xác lập trạng thái hoạt động cần thiết cho Vi điều khiển.

5. Các thanh ghi chức năng đặc biệt

* Từ trạng thái chương trình PSW

BIT	ĐỊA CHỈ BIT	KÍ HIỆU	CHỨC NĂNG

PSW.7	D7H	C hoặc Cy	Cờ nhớ
PSW.6	D6H	AC	Cờ nhớ phụ
PSW.5	D5H	F0	Cờ 0 hay cờ Zero
PSW.4	D4H	RS1	Bit lựa chọn dãy thanh ghi
PSW.3	D3H	RS0	Bit lựa chọn dãy thanh ghi
PSW.2	D2H	0V	Cờ tràn với phép tính liên quan đến số nhị phân có dấu
PSW.1	D1H	-	Chưa được thiết kế để sử dụng
PSW.0	D0H	P	Cờ chẵn lẻ

Chức năng từng bit trong thanh trạng thái PSW - *Cờ nhớ C*: Cờ được sử dụng trong các lệnh toán học:

+ C=1 nếu phép toán cộng xảy ra tràn hoặc phép trừ có mượn + C=0 nếu phép toán cộng không tràn hoặc phép trừ không có mượn.

- *Cờ nhớ phụ AC:*

+ Cờ AC được dùng trong các phép toán cộng hai số BCD.

Khi cộng số BCD:

+ Nếu kết quả 4 bit lớn hơn 09H thì AC=1 +

Nếu kết quả 4 bit dưới 09H thì AC=0.

- *Cờ 0 hay cờ nhớ Z:*

+ Cờ Z = 0 khi thanh ghi A có giá trị khác 0

+ Cờ Z =1 khi A thanh ghi A có giá trị là 0 -

Các bit chọn bank thanh ghi:

Hai bit RS1 và RS2 dùng để xác lập bank thanh ghi được sử dụng, mặc định RS1=0 và RS2=0

RS1	**RS2**	**Bank thank ghi được sử dụng**
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

- *Cờ tràn OV*

+ Được sử dụng trong các phép toán cộng có dấu, với các phép toán cộng không dấu cờ tràn OV được bỏ qua, không cần quan tâm đến OV. Nếu:

Phép cộng hai số có dấu lớn hơn +127 thì OV=1

Hoặc phép trừ hai số có dấu nhỏ hơn -127 thì OV=1

Các trường hợp còn lại OV=0

- *Cờ chẵn lẻ*

+ Cờ chẵn lẻ P tự động được đặt bằng 1 hoặc 0 sao cho tổng số bit mang giá trị 1 trên thanh ghi A với cờ P luôn là một số chẵn. Cờ chẵn lẻ được dùng để xử lí dữ liệu trước khi truyền đi theo kiểu nối tiếp hoặc xử lí dữ liệu trước khi nhận vào theo kiểu nối tiếp (hạn chế lỗi phát sinh trong quá trình truyền).

* Thanh ghi B

Thanh ghi B ở địa chỉ F0H, được dùng với thanh ghi A để thực hiện các phép toán số học. Khi thực hiện lệnh chia với thanh ghi A, số dư được lưu trữ ở thanh ghi B. Ngoài ra thanh ghi B còn được dùng như một thanh ghi đệm có nhiều chức năng * Con trỏ Stack

Con trỏ stack SP nằm tại địa chỉ 81h và không cho phép định địa chỉ bit. SP dùng để chỉ đến đỉnh của stack. Stack là một dạng bộ nhớ lưu trữ dạng Lipo (Last in first out) thường dùng lưu trữ địa chỉ trả về khi gọi một chương trình con. Ngoài ra, stack còn dùng như bộ nhớ tạm để lưu lại và khôi phục các giá trị cần thiết.

Mặc định khi khởi động, giá trị của SP là 07h

* Con trỏ dữ liệu DPTR

Con trỏ dữ liệu DPTR là thanh ghi 16 bit duy nhất của Vi điều khiển được tạo thành từ hai thanh ghi DPL (byte thấp-địa chỉ byte 82H) và DPH (byte caođịa chỉ byte 83H). Hai thanh ghi DPL và DPH có thể truy xuất độc lập bởi người sử dụng. Con trỏ dữ liệu DPTR thường được sử dụng khi truy xuất dữ liệu từ bộ nhớ ROM hoặc bộ nhớ từ bên ngoài.

* Các thanh ghi Port

Các thanh ghi P0 tại địa chỉ 80h, P1 tại địa chỉ 90h, P2 tại địa chỉ A0h, P3 tại địa chỉ B0h là các thanh ghi chốt cho 4 port xuất/nhập (Port 0, 1, 2, 3). Tất cả các thanh ghi này đều cho phép định địa chỉ bit trong đó địa chỉ bit của P0 từ 80h – 87h, P1 từ 90h – 97h, P2 từ A0h – A7h và P3 từ B0h – B7h. Các địa chỉ bit này có thể thay thế bằng toán tử địa chỉ.

* Các thanh ghi định thời

Các cặp thanh ghi (TH0, TL0), (TH1, TL1) và (TH2, TL2) là các thanh ghi cho các bộ định thời 0, 1 và 2 trong đó bộ định thời 2 chỉ có trong 8032/8052.

* Các thanh ghi của Port nối tiếp

Thanh ghi port nối tiếp tại địa chỉ 99h thực chất bao gồm 2 thanh ghi: thanh ghi nhận và thanh ghi truyền. Nếu dữ liệu đưa tới SBUF thì đó là thanh ghi truyền, nếu dữ liệu được đọc từ SBUF thì đó là thanh ghi nhận. Các thanh ghi này không cho phép định địa chỉ bit.

* Các thanh ghi ngắt

- Thanh ghi IP tại địa chỉ B8h cho phép chọn mức ưu tiên ngắt khi có 2 ngắt xảy ra đồng thời. IP cho phép định địa chỉ bit từ B8h – BFh.

- Thanh ghi IE tại địa chỉ A8h cho phép hay cấm các ngắt. IE có địa chỉ bit từ A8h – Afh.

* Thanh ghi điều khiển nguồn

- Thanh ghi TMOD tại địa chỉ 89h dùng để chọn chế độ hoạt động cho các bộ định thời (0, 1) và không cho phép định địa chỉ bit.

- Thanh ghi TCON tại địa chỉ 88h điều khiển hoạt động của bộ định thời và ngắt. TCON có địa chỉ bit từ 88h – 8Fh.

- Thanh ghi T2CON tại địa chỉ C8h điều khiển hoạt động của bộ định thời 2. T2CON có địa chỉ bit từ C8h – CFh.

- Thanh ghi SCON tại địa chỉ 98h điều khiển hoạt động của port nối tiếp. SCON có địa chỉ bit từ 98h – 9Fh.

* Thanh ghi tích lũy (A hay Acc)

- Thanh ghi tích lũy là thanh ghi sử dụng nhiều nhất trong AT89C51, được ký hiệu trong câu lệnh là A. Ngoài ra, trong các lệnh xử lý bit, thanh ghi tích lũy được ký hiệu là Acc.

- Thanh ghi tích lũy có thể truy xuất trực tiếp thông qua địa chỉ E0h (byte) hay truy xuất từng bit thông qua địa chỉ bit từ E0h đến E7h.

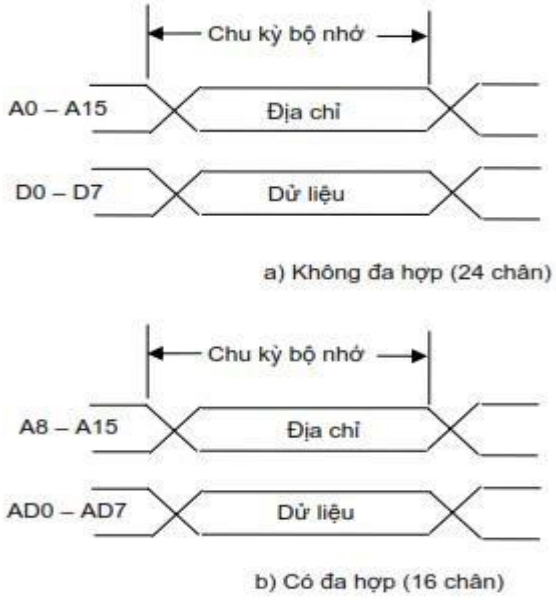
```
Ví dụ Mov A,#1
      Mov 0E0h,#1
Hay
      Setb Acc.4
      Setb 0E4h
```

6. Bộ nhớ ngoài

Các bộ vi điều khiển cần có khả năng mở rộng các tài nguyên trên chip (bộ nhớ, I/O...) để tránh hiện tượng cổ chai trong thiết kế. Cấu trúc của MCS-51 cho phép khả năng mở rộng không gian bộ nhớ chương trình đến 64K và không gian

bộ nhớ dữ liệu đến 64K ROM và RAM ngoài được thêm vào khi cần. Các IC giao tiếp ngoại vi cũng có thể được thêm vào để mở rộng khả năng xuất/nhập. Chúng trở thành một phần của không gian bộ nhớ dữ liệu ngoài bằng cách sử dụng cách định địa chỉ kiểu I/O ánh xạ bộ nhớ.

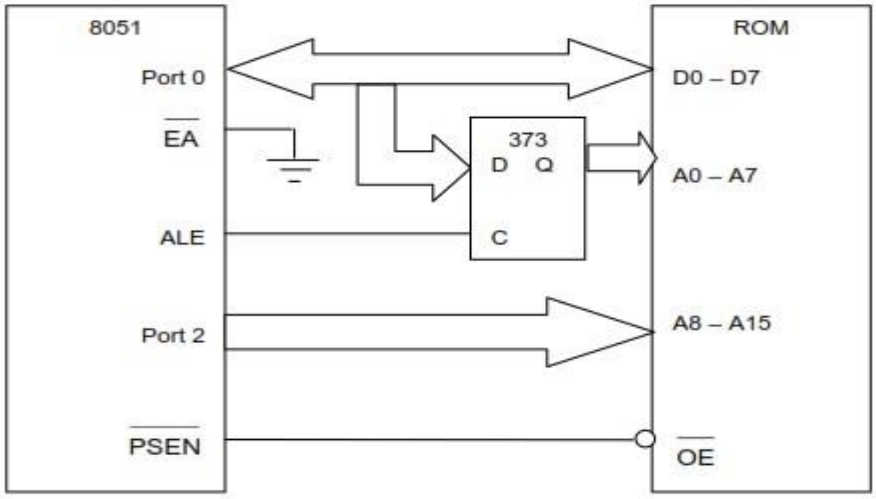
Khi bộ nhớ ngoài được sử dụng, port 0 không làm nhiệm vụ của port xuất/nhập, port này trở thành bus địa chỉ (A0..A7) và bus dữ liệu (D0..D7) đa hợp. Ngõ ra ALE chốt một byte thấp của địa chỉ ở thời điểm bắt đầu một chu kỳ bộ nhớ ngoài. Port 2 thường (nhưng không phải luôn luôn) được dùng làm byte cao của bus địa chỉ



Hình 2.6. Bus đa hợp địa chỉ/ dữ liệu

Sắp xếp không đa hợp sử dụng 16 đường địa chỉ và 8 đường dữ liệu tổng cộng 24 đường. Sắp xếp đa hợp kết hợp 8 đường của bus dũ liệu và byte thấp của bus địa chỉ thì chỉ cần 16 đường. Việc tiết kiệm các chân cho phép đóng gói họ MCS-51 trong một vỏ 40 chân. Sắp xếp đa hợp hoạt động như sau: Trong ½ chu kỳ đầu của chu kỳ bộ nhớ, byte thấp của địa chỉ được cung cấp bởi port 0 và được chốt nhờ tín hiệu ALE. Mạch chốt 74373 giữ cho byte thấp của địa chỉ ổn định trong cả chu kỳ bộ nhớ. Trong ½ sau của chu kỳ bộ nhớ, port 0 được sử dụng làm bus dữ liệu và dữ liệu sẽ được đọc hay ghi.

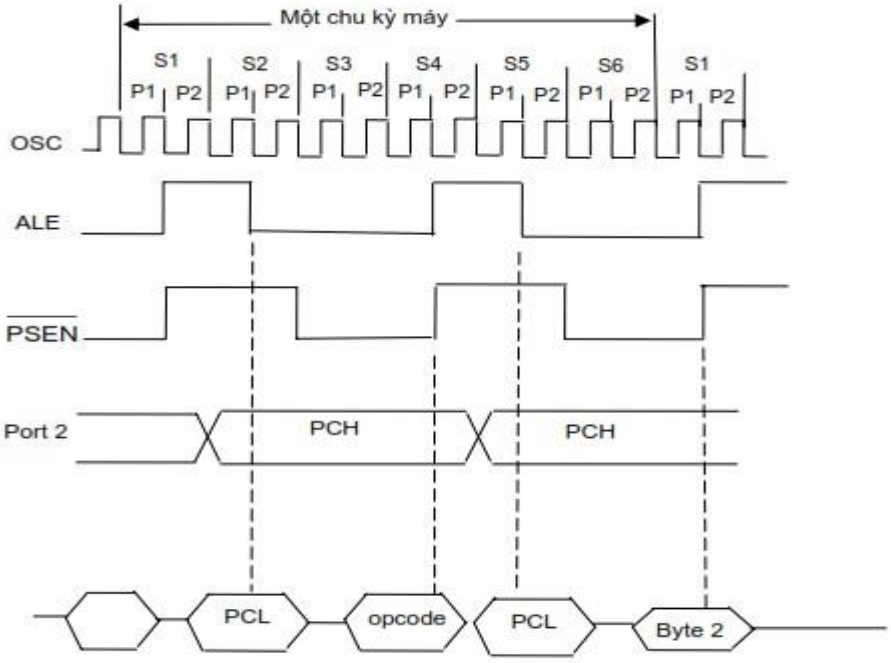
6.1 Truy xuất bộ nhớ chương trình ngoài Bộ nhớ chương trình ngoài là bộ nhớ chỉ đọc, được cho phép bởi tín hiệu PSEN.



Hình 2.7. Truy xuất ROM ngoài

Khi có một ROM ngoài được sử dụng, cả hai port 0 và port 2 đều không còn là các port xuất/nhập. Kết nối phần cứng với bộ ngoài được trình bày ở hình 2.7

Một chu kỳ máy của 8051 có 12 chu kỳ dao động. Nếu bộ dao động trên chip có tần số 12 MHz thì một chu kỳ máy dài 1□*s* . Trong 1 chu kỳ máy điển hình, ALE có hai xung và 2 byte của lệnh được đọc từ bộ nhớ chương trình (nếu lệnh chỉ có 1 byte, byte thứ hai bị loại bỏ). Giản đồ thời gian của chu kỳ máy này được gọi là chu kỳ tìm - nạp lệnh được trình bày ở hình 2.8



Hình 2.8. Chu kỳ tìm nạp lệnh ROM ngoài

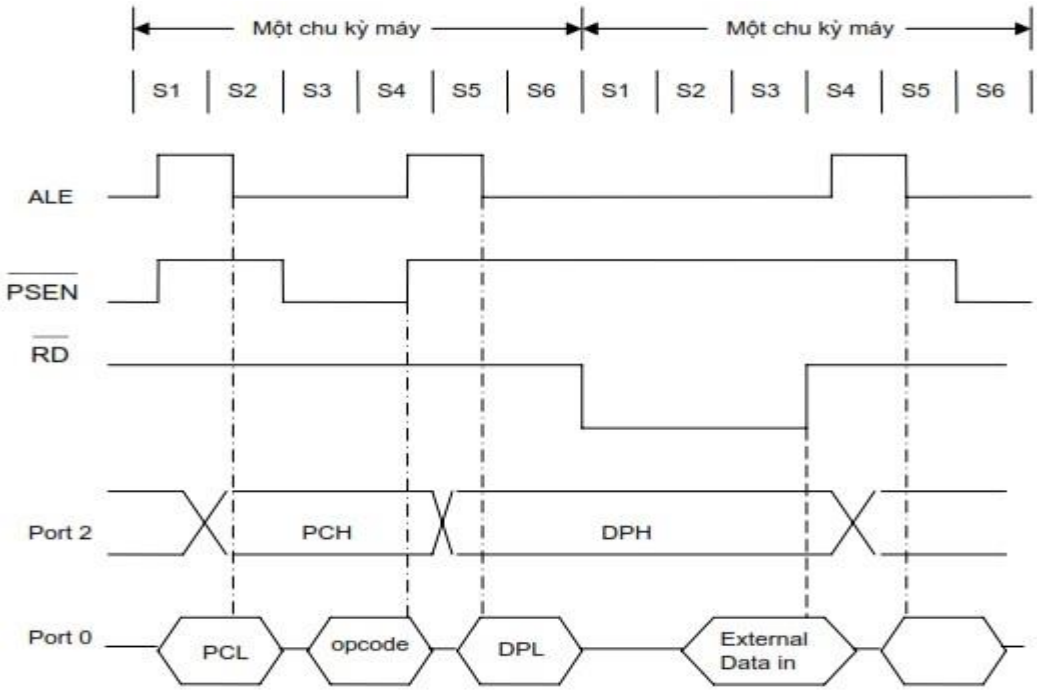
6.2 Truy xuất bộ nhớ dữ liệu ngoài

Bộ nhớ dữ liệu ngoài là bộ nhớ đọc-ghi được cho phép bởi các tín hiệu $\overline{RD}$

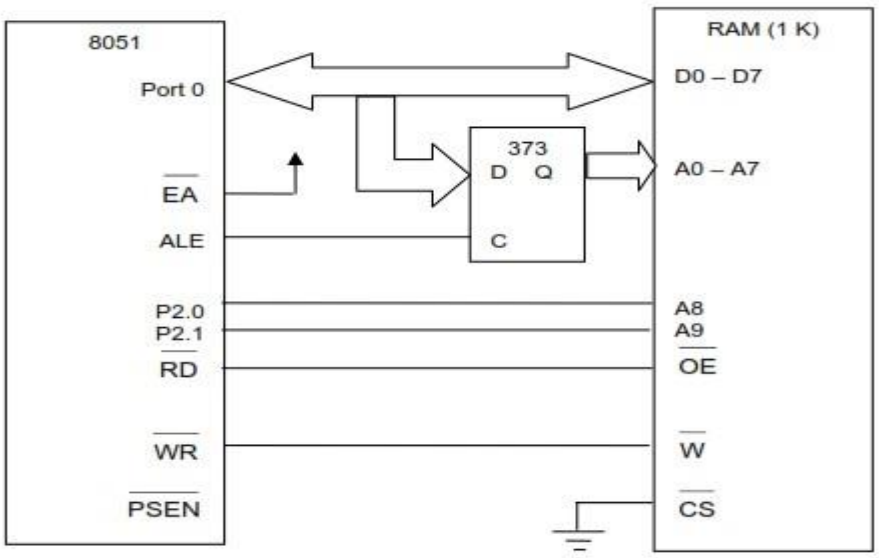
và *WR* ở các chân P3.7 và P3.6. Lệnh dùng để truy xuất bộ nhớ dữ liệu ngoài là MOVX, sử dụng hoặc con trỏ dữ liệu 16 bít DPTR hoặc R0, R1 làm thanh ghi chứa địa chỉ.

RAM có thể giao tiếp với 8051 theo cùng cách như ROM ngoại trừ đường $\overline{RD}$ nối với đường cho phép xuất $\overline{OE}$ của RAM và đường $\overline{WR}$ nối với đường $\overline{W}$ của RAM. Các kết nối với bus dữ liệu và bus địa chỉ giống như ROM bằng cách sử dụng port 0 và port 2 như ở phần trên dung lượng của RAM lên đến 64K Giản đồ thời gian của thao tác đọc dữ liệu ở bộ nhớ dữ liệu ngoài được trình bày ở hình 2.10 cho lệnh MOVX A,@DPTR. Lưu ý là cả hai xung ALE và PSEN được bỏ qua ở nơi mà xung $\overline{RD}$ cho phép đọc RAM, nếu lệnh MOVX và RAM ngoài không bao giờ được dùng, các xung ALE luôn có tần số bằng 1/6 tần số của mạch dao động

Giản đồ thời gian của chu kỳ ghi (lệnh MOVX @DPTR, A) cũng tương tự ngoại trừ các xung $\overline{WR}$ ở mức thấp và dữ liệu được xuất ra ở port 0 ($\overline{RD}$ vẫn ở mức cao)



Hình 2.9. Giản đồ thời gian lệnh MOVX



Hình 2.10. Giao tiếp với 1K RAM

Port 2 giảm bớt được chức năng làm nhiệm vụ cung cấp byte cao của địa chỉ trong các hệ thống tối thiểu hóa thành phần, hệ thống không dùng bộ nhớ chương trình ngoài và chỉ có một dung lượng nhỏ bộ nhớ dữ liệu ngoài. Các địa chỉ 8 bít có thể truy xuất bộ nhớ dữ liệu ngoài với cấu hình bộ nhớ nhỏ hướng trang. Nếu có nhiều hơn một trang 256 byte RAM, một vài bít từ port 2 hoặc một port khác có thể chọn một trang. VD: Với một RAM 1KB (4 trang 256) có thể được kết nối với 8051 như ở hình 2.10.

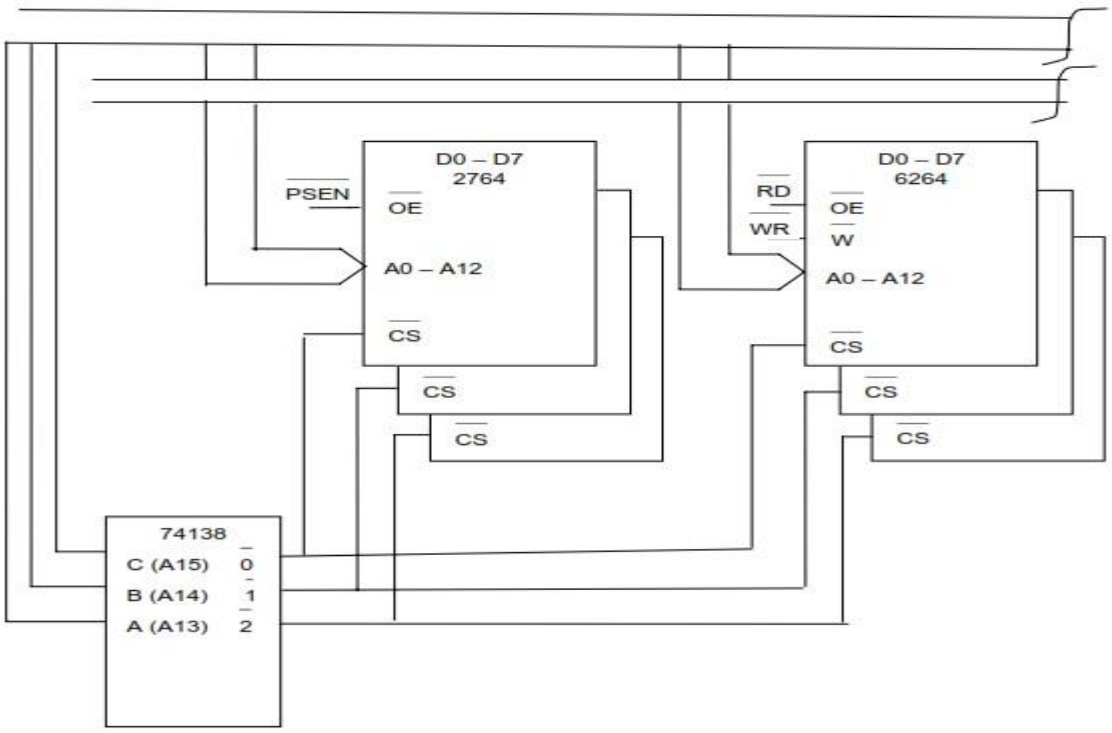
Các bít 0 và 1 của port 2 phải được khởi động để chọn một trang, sau đó dùng lệnh MOVX để đọc hoặc ghi trên trang này. Giả sử P2.0 = P2.1 = 0, các lệnh sau có thể dùng để đọc nội dung của RAM ngoài tại địa chỉ 0050H vào thanh ghi A

```
MOV R0,#50H
MOVX A, @R0
```

Để đọc ở địa chỉ cuối cùng của RAM là 03FFH thì phải chọn trang 3, nghĩa là phải set các bít P2.0 và P2.1 bằng 1 như chuỗi lệnh sau

```
SETB P2.0
SETB P2.1
MOV R0,#0FFH
MOVX A, @R0
```

Một đặc trưng của thiết kế này là các bít từ 2 đến 7 của port 2 không còn cần làm bít địa chỉ nữa, các bít còn lại này có thể sử dụng cho mục đích xuất/nhập **6.3. Giải mã địa chỉ**



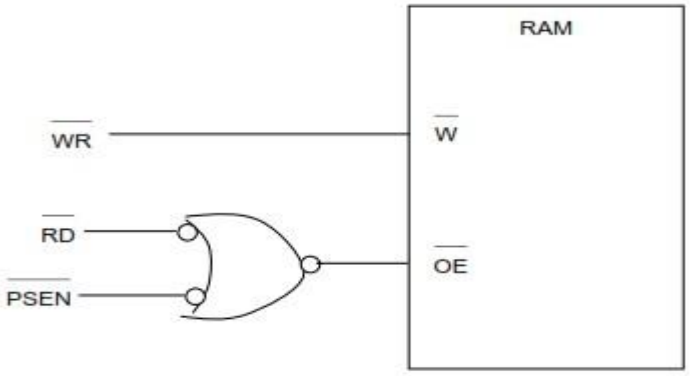
Hình 2.11. Giải mã địa chỉ

Nếu có nhiều ROM hoặc RAM giao tiếp với 8051 thì cần phải giải mã địa chỉ. Việc giải mã này cũng cần cho hầu hết các bộ vi xử lý. Ví dụ nếu các ROM và RAM 8KB được sử dụng, địa chỉ phải được giải mã để chọn các IC nhớ này trên các giới hạn 8K (0000H..1FFFH, 2000H..3FFFH....) Một IC giải mã điển hình là 74138 được dùng với các ngõ ra được nối với các ngõ vào chọn chíp $\overline{CS}$ của các IC nhớ như mô tả ở hình 2.11 cho một bộ nhớ có nhiều EPROM 2764 và RAM 6264. Cần lưu ý là do các đường cho phép riêng rẽ (PSEN cho bộ nhớ chương trình, $\overline{RD}$ và $\overline{WR}$ cho bộ nhớ dữ liệu) 8051 có thể quản lý không gian nhớ 64K cho bộ nhớ ROM và 64K cho bộ nhớ RAM

6.4 Các không gian nhớ chương trình và dữ liệu gối nhau

Vì bộ nhớ chương trình là bộ nhớ chỉ đọc, một tình huống khó xử được phát sinh trong quá tŕnh phát triển phần mềm cho 8051. Làm thế nào phần mềm được viết cho một hệ thống đích dể gỡ rối nếu phần mềm chỉ có thể được thực hiện từ không gian bộ nhớ chương trình chỉ đọc. Giải pháp tổng quát là cho các không gian bộ nhớ chương trình và dữ liệu ngoài gối lên nhau. Vì *PSEN* được dùng để đọc bộ nhớ chương trình và $\overline{RD}$ được dùng để đọc bộ nhớ dữ liệu, một RAM có thể chiếm không gian nhớ chương trình và dữ liệu bằng cách nối chân OE tới ngõ ra cổng AND có các ngõ vào là $\overline{PSEN}$ và $\overline{RD}$.

Mạch trình bày ở hình 2.12 cho phép IC RAM được ghi như là bộ nhớ dữ liệu và được đọc như là bộ nhớ chương trình hoặc dữ liệu. Vậy thì một chương trình có thể được nạp vào RAM bằng cách ghi vào RAM như là bộ nhớ dữ liệu và được thực hiện bằng cách truy xuất như là bộ đếm chương trình



Hình 2.12. Gối không gian nhớ ROM và RAM

7. Các cải tiến của 8032/8052

Các vi mạch 8032/8052 và các phiên bản CMOS có hai cải tiến so với 8031/8051. Một là có thêm 128 byte RAM trên chip từ địa chỉ 80H đến FFH, điều này không xung đột với các thanh ghi chức năng đặc biệt vì 128 byte RAM thêm vào chỉ có thể truy xuất bằng cách dùng kiểu định địa chỉ gián tiếp, xem lệnh sau

MOV A, 0F0H

Sẽ di chuyển nội dung của thanh ghi B vào thanh ghi A đối với họ MCS-51 còn chuỗi lệnh sau

MOV R0,#0F0H

MOV A, @R0

Ghi vào thanh ghi A nội dung tại địa chỉ 0F0H đối với 8032/8052, tổ chức bộ nhớ nội của 8032/8052 được trình bày ở hình 2.13

Hình 2.13. RAM nội trong 8032/8052

BẢNG 2.5 Các thanh ghi của timer 2

Thanh ghi	Địa chỉ	Địa chỉ bít	Mô tả
T2CON	C8H	Có	Điều khiển
RCAP2L	CAH	Không	Nhận byte thấp
RCAP2H	CBH	Không	Nhận byte cao
TL2	CCH	Không	Byte thấp timer 2
TH2	CDH	không	Byte cao timer 2

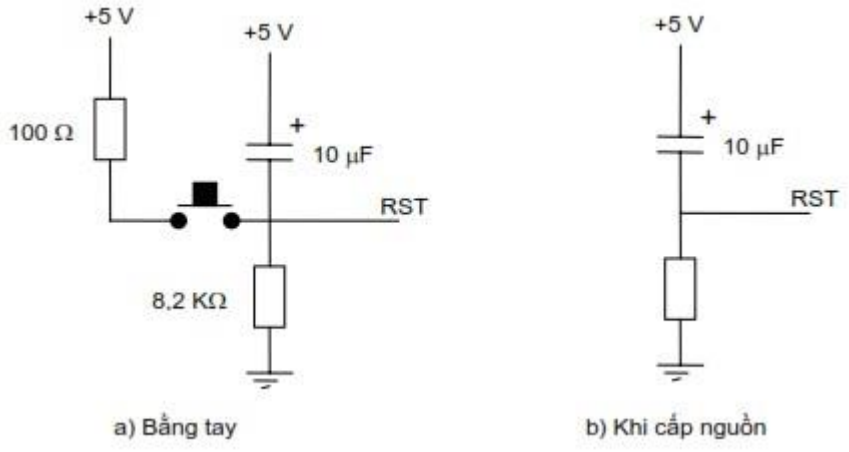
Cải tiến thứ hai là có thêm bộ định thời 16 bít, bộ timer 2 này được lập trình nhờ vào 5 thanh ghi chức năng đặc biệt trong bảng 2.5

8. Hoạt động Reset

8051 được reset bằng cách giữ chân RST ở mức cao tối thiểu 2 chu kỳ máy và sau đó chuyển về mức thấp. RST có thể được tác động bằng tay hoặc được tác động khi cấp nguồn bằng cách dùng một mạch RC như trình bày ở hình 2.14. Trạng thái của tất cả các thanh ghi sau khi reset hệ thống được tóm tắt ở bảng 2.6

BẢNG 2.6 Giá trị của các thanh ghi sau khi reset hệ thống

Thanh ghi	Nội dung
Bộ đếm chương trinh	0000H
Thanh ghi A	00H
Thanh ghi B	00H
PSW	00H
SP	07H
DPTR	0000H
Port 0..3	FFH
IP	xxx00000B (8031/8051) xx000000B (8032/8052)
IE	0xx00000B (8031/8051) 0x000000B (8032/8052)
Thanh ghi định thời	00H
SCON	00H
SBUF	00H
PCON (HMOS)	0xxxxxxxB
PCON (CMOS)	0xxx0000B



Hình 2.14. Sơ đồ mạch RESET

BÀI 3: TẬP LỆNH 8051

1. Mở đầu

- Vi điều khiển là một IC lập trình, vì vậy Vi điều khiển cần được lập trình trước khi sử dụng. Mỗi phần cứng nhất định phải có chương trình phù hợp kèm theo, do đó trước khi viết chương trình đòi hỏi người viết phải nắm bắt được cấu tạo phần cứng và các yêu cầu mà mạch điện cần thực hiện.

- Chương trình là tập hợp các lệnh được tổ chức theo một trình tự hợp lí để giải quyết các yêu cầu của người lập trình.Tập hợp tất cả các lệnh gọi là tập lệnh. Họ Vi điều khiển MSC-51 đều có chung một tập lệnh, các Vi điều khiển được cải tiến sau này thường ít thay đổi hoặc mở rộng tập lệnh mà chú trọng phát triển phần cứng.

- Lệnh của Vi điều khiển là các số nhị phân 8 bit hay còn gọi là mã máy. Các lệnh mang mã 00000000b đến 11111111b. Các mã lệnh này được đưa vào lưu trữ trong ROM, khi thực hiện chương trình Vi điều khiển đọc các mã lệnh này, giải mã, và thực hiện lệnh.

- Vì các lệnh của Vi điều khiển có dạng số nhị phân quá dài và khó nhớ, hơn nữa việc gỡ lỗi khi chương trình phát sinh lỗi rất phức tạp và khó khăn. Khó khăn này được giải quyết với sự hỗ trợ của máy vi tính, người viết chương trình có thể viết chương trình cho vi điều khiển bằng các ngôn ngữ lập trình cấp cao, sau khi việc viết chương trình được hoàn tất, các trình biên dịch sẽ chuyển các câu lệnh cấp cao thành mã máy một cách tự động. Các mã máy này sau đó được đưa (nạp) vào bộ nhớ ROM của Vi điều khiển, Vi điều khiển sẽ tìm đến đọc các lệnh từ ROM để thực hiện chương trình . Bản thân máy tính không thể thực hiện các mã máy này vì chúng không phù hợp với phần cứng máy tính, muốn thực hiện phải có các chương trình mô phỏng dành riêng.

- Chương trình cho Vi điều khiển có thể viết bằng C++,C,Visual Basic, hoặc băng các ngôn ngữ cấp cao khác. Tuy nhiên hợp ngữ Assembler được đa số người dùng Vi điều khiển sử dụng để lập trình, vì lí do này chúng tôi chọn Assembly để hướng dẫn viết chương trình cho Vi điều khiển. Assembly là một ngôn ngữ cấp thấp, trong đó mỗi câu lệnh chương trình tương ứng với một chỉ lệnh mà bộ xử lý có thể thực hiện được. Ưu điểm của hợp ngữ Assembly là: mã gọn,ít chiếm dung lượng bộ nhớ, hoạt động với tốc độ nhanh, và nó có hiệu suất tốt hơn so với các chương trình viết bằng ngôn ngữ bậc cao khác.

2. Các cách định địa chỉ

2.1. Định địa chỉ bằng thanh ghi

Chế độ định địa chỉ theo thanh ghi liên quan đến việc sử dụng các thanh ghi để lưu dữ liệu cần được thao tác và các toán hạng là một trong các thanh ghi Ri của các bank được chọn.

Ví dụ:

Mov A, R0 ; sao nội dung thanh ghi R0 vào thanh ghi A

Mov R2, A ; sao nội dung thanh ghi A vào thanh ghi R2

2.2. Định địa chỉ trực tiếp Ví

dụ:

Mov R0, 40h ; lưu nội dung của ngăn nhớ 40h của Ram vào R0

Mov 56h, A ; lưu nội dung thanh ghi A vào ngăn nhớ 56h của Ram

Ví dụ: Hai lệnh sau đều sao nội dung thanh ghi R4 vào A

Mov A, 4

Mov A, R4

2.3. Định địa chỉ gián tiếp

Đặc điểm nhận ra chế độ này là luôn có ký tự @ đứng trước toán hạng Ví dụ:

Mov A, @R0 ; chuyển nội dung của năng nhớ Ram có địa chỉ trong R0 vào A

2.4. Định địa chỉ tức thời

- Trong chế độ đánh địa chỉ này toán hạng nguồn là một hằng số
- Lưu ý rằng trước dữ liệu tức thời phải được đặt dấu (#) chế độ đánh địa chỉ này có thể được dùng để nạp thông tin vào bất kỳ thanh ghi nào kể cả thanh ghi con trở dữ liệu DPTR Ví dụ:

Mov A, #25h ; nạp giá trị 25h vào thanh ghi A

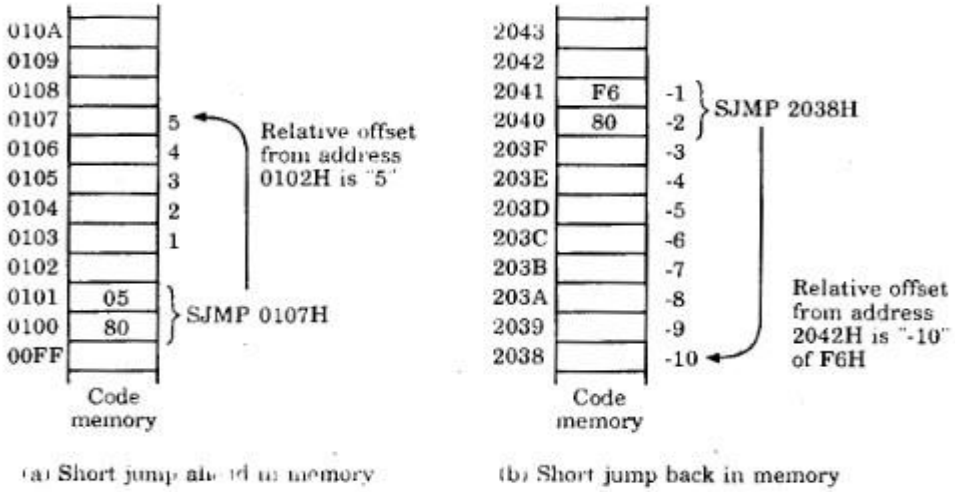
Mov R4, #62 ; nạp giá trị 62 thập phân vào R4

Mov DPTR, #4251h ; nạp 4521h vào con trỏ dữ liệu DPTR

2.5. Định địa chỉ tương đối

- Kiểu định địa chỉ tương đối chỉ được sử dụng cho các lệnh nhảy. Một địa chỉ tương đối (hay còn gọi là Offset) là một giá trị 8 bit có dấu. Giá trị này được cộng với bộ đếm chương trình để tạo ra địa chỉ của lệnh tiếp theo cần được thực thi. Do ta sử dụng một offset 8 bit có dấu, tầm nhảy được giới hạn là -128 byte

đến 127 byte. Byte địa chỉ tương đối là byte thêm vào tiếp theo byte opcode của lệnh



(a) Short jump ahead in memory (b) Short jump back in memory

Hình 3.1. Tính offset cho kiểu định địa chỉ tương đối

- Short jump ahead in memory: nhảy tới và ngắn

- Short jump back in memory: nhảy lùi và ngắn

- Relative offset from address 2042h is – 10(F6H): offset tương đối từ địa chỉ 2042h là -10(F6H)

- Nhờ vào phép cộng, bộ đếm chương trình được tăng đến địa chỉ theo sau lệnh nhảy; vậy thì địa chỉ mới liên quan đến lệnh kế tiếp, không liên quan đến địa chỉ của lệnh nhảy.

- Thông thường chi tiết này không liên quan đến người lập trình do bởi các đích nhảy thường được xác định bằng các nhãn và trình dịch hợp ngữ sẽ xác định offset tương đối tương ứng. Ví dụ nếu nhãn THERE đặt trước lệnh ở địa chỉ 1040H, lệnh:

SJMP THERE

- Ở trong bộ nhớ tại địa chỉ 1000h và 1001h, trình dịch hợp ngữ sẽ gán offset tương đối là 3EH cho byte 2 của lệnh (1002H + 3EH = 1040H)

- Định địa chỉ tương đối có điểm lợi là cung cấp cho ta mã không phụ thuộc vào vị trí (vì các địa chỉ tuyệt đối không được dùng), nhưng lại có điểm bất lợi là các đích nhảy bị giới hạn trong tầm.

2.6. Định địa chỉ tuyệt đối

- Cách định địa chỉ này chỉ áp dụng với lệnh ACALL và AJMP. Đây là các lệnh 2 byte cho phép chương trình nhảy trong phạm vi một trang 2K trong bộ nhớ chương trình bằng cách cung cấp 11 bít thấp của địa chỉ đích trong mã lệnh (A10-A8) và byte thứ 2 của mã lệnh (A7-A0).

- Năm bít cao của địa chỉ đích chính là 5 bít cao hiện đang chứa trong bộ đếm chương trình, do đó lệnh theo sau lệnh nhảy và đích đến của lệnh nhảy phải có vị trí trong cùng một trang 2K, bởi vì A15-A11 không thay đổi.

- Ưu điểm của phương pháp này là mã lệnh ngắn (2 byte) nhưng khuyết điểm là phạm vi nhảy có giới hạn và mã lệnh phụ thuộc vào vị trí

2.7. Định địa chỉ dài

Cách định địa chỉ này chỉ dùng cho lệnh LCALL và LJMP, đây là các lệnh 3 byte bao gồm địa chỉ đích đầy đủ 16 bít, đó là byte 2 và byte 3 trong mã lệnh. Ưu điểm là có thể nhảy đến một vị trí bất kỳ trong phạm vi 64K bộ nhớ chương trình, khuyết điểm là lệnh dài (3 byte) và phụ thuộc vị trí, điều này làm cho chương trình không thể được thực hiện tại địa chỉ khác. VD: Một chương trình có địa chỉ bắt đầu tại 2000H và trong chương trình có lệnh nhảy LJMP 2040H, thì chương trình không thể nào di chuyển đến 4000H vì lệnh nhảy này luôn nhảy đến 2040H, đây sẽ là môt vị trí sai nếu di chuyển chương trình đến 4000H.

2.8. Định địa chỉ theo chỉ số

- Chế độ định địa chỉ theo chỉ số được sử dụng rộng rãi trong việc truy cập các phần tử dữ liệu của bảng trong không gia ROM/RAM chương trình của 8051 trong dải 64Kb.

MovC A, @A + DPTR

MovX A, @A + DPTR

- Thanh ghi 16 bit DPTR là thanh ghi A được dùng để tạo ra địa chỉ của phần tử dữ liệu được lưu trong bộ nhớ (trong hoặc ngoài 8051)

- Thanh lệnh Mov bằng MovC/MovX do các phần tử dữ liệu được cất trong không gian mã (chương trình) của Flash ROM trong/ngoài chip của 8051. Trong lệnh này thì nội dung của A được bổ xung vào thanh ghi 16 bit DPTR để tạo ra địa chỉ 16 bit của dữ liệu cần thiết.

3. Các nhóm lệnh

3.1. Nhóm lệnh số học

3.1.1. Lệnh cộng dữ liệu trên thanh ghi A với dữ liệu trên thanh ghi Rn:

Cú pháp: ***Add A,Rn***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Cộng giá trị dữ liệu trên thanh ghi A với giá trị dữ liệu trên thanh ghi Rn, sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW

Ví dụ:

Mov A,#20H
Mov R1,#08H
Add A,R1

Kết quả : A có giá trị là 28H
R1 vẫn giữ nguyên giá trị là 08H
Cờ C = 0 Vidu2:

Mov A,#0E9H
Mov R6,#0BAH Add A,R6

Kết quả : A = #0A3h
R6 = #0Bah
Cờ C = 1

3.1.2. Lệnh cộng dữ liệu trên thanh ghi A với dữ liệu ở ô nhớ có địa chỉ direct:

Cú pháp: ***Add A,direct***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Cộng giá trị dữ liệu trên thanh ghi A với giá trị dữ liệu trên ô nhớ có địa chỉ direct, sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A.

Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW Ví dụ:

Mov 50h,#20H
Mov A,#0E8H
Add A,50H

Kết quả : A = #08H
50H = #20H C = 1

3.1.3. Lệnh cộng dữ liệu trên thanh ghi A với dữ liệu của ô nhớ có địa chỉ gián tiếp:

Cú pháp: ***Add A,@Ri***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Cộng giá trị dữ liệu trên thanh ghi A với giá trị dữ liệu của ô nhớ có địa chỉ bằng giá trị của thanh ghi Ri, sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW

Ví dụ:

```
AC  =  1     ;cờ C đang mang giá trị 1 Mov
        50H,#60H
        Mov      R2,#50H
        Mov       A,#01H
        Add       A,@R2
Kết quả  :    A = #61H
              R2 = #50H
              C = 0            ;cờ C mang giá trị 0
```

3.1.4. Lệnh cộng dữ liệu trên thanh ghi A với dữ liệu xác định:

Cú pháp: ***Add A,#data***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Cộng giá trị dữ liệu trên thanh ghi A với một giá trị xác định, sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW

Ví dụ:

```
      Mov      A,#05h
  Add        A,#06h
Kết quả :     A = #0Bh
              C = 0
```

3.1.5. Lệnh cộng dữ liệu trên thanh ghi A với dữ liệu trên thanh ghi Rn có số nhớ ở cờ C:

Cú pháp: ***AddC A,Rn***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Cộng giá trị dữ liệu trên thanh ghi A với giá trị dữ liệu trên thanh ghi Rn và cộng thêm giá trị của số nhớ trên cờ C, sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW

Ví dụ:

```
              C = 1
              Mov      A,#08h
```

Mov R1,#10h
Addc A,R1

Kết quả : A = #19h ;cộng cả cờ C
R1 = #10h
C =0

3.1.6. Lệnh cộng dữ liệu trên thanh ghi A với dữ liệu ở ô nhớ có địa chỉ direct và giá trị số nhớ ở cờ C:

Cú pháp: ***AddC A,direct***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Cộng giá trị dữ liệu trên thanh ghi A với giá trị dữ liệu của ô nhớ có địa chỉ direct và cộng thêm giá trị của số nhớ trên cờ C , sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW

Ví dụ: C = 0
Mov A,#0A5h
Mov 10h,#96h
Addc A,10h

Kết quả : A = #3Bh
10h = #96h
C =1

3.1.7. Lệnh cộng dữ liệu trên thanh ghi A với dữ liệu của ô nhớ có địa chỉ gián tiếp và số nhớ ở cờ C:

Cú pháp: ***AddC A,@Ri***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Cộng giá trị dữ liệu trên thanh ghi A với giá trị dữ liệu của ô nhớ có địa chỉ bằng giá trị của thanh ghi Ri và cộng thêm giá trị của số nhớ trên cờ C, sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW

Ví dụ:

C = 1
Mov A,#05h
Mov 50h,#10h
Mov R2,#50h

Addc a,@R2

Kết quả : A = #16h

C = 0

3.1.8. Lệnh cộng dữ liệu trên thanh ghi A với dữ liệu xác định và số nhớ ở cờ C:

Cú pháp: ***AddC A,#data***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Cộng giá trị dữ liệu trên thanh ghi A với giá trị xác định và cộng thêm giá trị của số nhớ trên cờ C, sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW

Ví dụ:

C = 1

Mov A,#05h

Addc A,#16h

Kết quả : A = #1Ch

C = 0

3.1.9. Lệnh trừ dữ liệu trên thanh ghi A với dữ liệu trên thanh ghi Rn và số nhớ ở cờ C:

Cú pháp: ***SubB A,Rn***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Trừ giá trị dữ liệu trên thanh ghi A với giá trị dữ liệu trên thanh ghi Rn và trừ cho giá trị nhớ trên cờ C, sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW Ví dụ: C = 1

Mov A,#0E5h

Mov R3,#9Fh

Subb A,R3

Kết quả : A = 45h

C = 0

3.1.10. Lệnh trừ dữ liệu trên thanh ghi A với dữ liệu ở ô nhớ có địa chỉ direct và số nhớ ở cờ C:

Cú pháp: ***SubB A,direct***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Trừ giá trị dữ liệu trên thanh ghi A với giá trị dữ liệu của ô nhớ có địa chỉ direct và trừ cho giá trị nhớ trên cờ C , sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW

Ví dụ:

```
C =  0
Mov   A,#0E5h
Mov    05h,#9Fh
Subb   A,05h
```

Kết quả : A = 46h

C = 0

3.1.11. Lệnh trừ dữ liệu trên thanh ghi A với dữ liệu của ô nhớ có địa chỉ gián tiếp và số nhớ ở cờ C:

Cú pháp: ***SubB A,@Ri***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Trừ giá trị dữ liệu trên thanh ghi A với giá trị dữ liệu của ô nhớ có địa chỉ bằng giá trị của thanh ghi Ri và trừ cho giá trị nhớ trên cờ C, sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW

Ví dụ:

```
C =  1
Mov   A,#0E5h
Mov   4Fh,#50h
Mov          R3,#4Fh
Subb   A,@R3
```

Kết quả : A = 94h

C = 0

3.1.12. Lệnh trừ dữ liệu trên thanh ghi A với dữ liệu xác định và số nhớ ở cờ C:

Cú pháp: ***SubB A,#data***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Trừ giá trị dữ liệu trên thanh ghi A với giá trị xác định và trừ thêm giá trị nhớ trên cờ C, sau khi thực hiện lệnh kết quả được lưu ở thanh ghi A. Lệnh này có ảnh hưởng đến thanh thanh trạng thái PSW

Ví dụ:

C = 0

Mov A,#05h

Subb A,#4Fh

Kết quả : A = 0B6h

C = 1

3.1.13.Lệnh tăng giá trị dữ liệu trên thanh ghi A lên 1 đơn vị:

Cú pháp: ***Inc A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Tăng giá trị dữ liệu lưu giữ trên thanh ghi A lên 1 đơn vị, không ảnh hưởng đến các cờ nhớ trên PSW Ví dụ: *Mov A,#05h*

Inc A

Kết quả : A = #06h

3.1.14.Lệnh tăng giá trị dữ liệu trên thanh ghi Rn lên 1 đơn vị:

Cú pháp: ***Inc Rn***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Tăng giá trị dữ liệu lưu giữ trên thanh ghi Rn lên 1 đơn vị, không ảnh hưởng đến các cờ nhớ trên PSW

Ví dụ:

Mov R7,#0Fh

Inc R7

Kết quả : R7 = #10h

3.1.15.Lệnh tăng giá trị dữ liệu ở ô nhớ có địa chỉ direct lên 1 đơn vị:

Cú pháp: ***Inc direct***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Tăng giá trị dữ liệu ở một ô nhớ có địa chỉ direct lên 1 đơn vị, không ảnh hưởng đến các cờ nhớ trên PSW

Ví dụ:

Mov 50h,#0FFh

Inc 50h

Kết quả : 50h = #00

3.1.16.Lệnh tăng giá trị dữ liệu ở ô nhớ có địa chỉ gián tiếp lên 1 đơn vị:

Cú pháp: ***Inc @Ri***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Tăng giá trị dữ liệu ở ô nhớ có địa chỉ bằng giá trị dữ liệu trên Ri lên 1 đơn vị, không ảnh hưởng đến các cờ nhớ trên PSW

Ví dụ:

Mov 0Fh,#05h

Mov R0,#0Fh

Inc @R0

Kết quả : R0 = #06h

0Fh = #05h

3.1.17.Lệnh tăng giá trị của con trỏ dữ liệu DPTR lên 1 đơn vị:

Cú pháp: ***Inc DPTR***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: Tăng giá trị dữ liệu của thanh ghi con trỏ dữ liệu DPTR lên 1 đơn vị, không ảnh hưởng đến các cờ nhớ trên PSW

Ví dụ:

Mov DPTR,#5Fh

Inc DPTR

Kết quả : DPTR = #060h

3.1.18.Lệnh giảm giá trị dữ liệu trên thanh ghi A xuống 1 đơn vị:

Cú pháp: ***Dec A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Giảm giá trị dữ liệu lưu giữ trên thanh ghi A xuống 1 đơn vị, không ảnh hưởng đến các cờ nhớ trên PSW

Ví dụ:

Mov A,#05h

Dec A

Kết quả : A = #04h

3.1.19.Lệnh giảm giá trị dữ liệu trên thanh ghi Rn xuống 1 đơn vị:

Cú pháp: ***Dec Rn***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Giảm giá trị dữ liệu lưu giữ trên thanh ghi Rn xuống 1 đơn vị, không ảnh hưởng đến các cờ nhớ trên PSW

Ví dụ:

```
Mov   R6,#0Fh
Dec    R6
```

Kết quả : R6 = #0Eh

3.1.20.Lệnh giảm giá trị dữ liệu ở ô nhớ có địa chỉ direct xuống 1 đơn vị:

Cú pháp: ***Dec direct***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Giảm giá trị dữ liệu ở ô nhớ có địa chỉ direct xuống 1 đơn vị, không ảnh hưởng đến các cờ nhớ trên PSW

Ví dụ:

```
Mov   7Fh,#0
Dec    7Fh
```

Kết quả : 7Fh = #0FFh

3.1.21.Lệnh giảm giá trị dữ liệu ở ô nhớ có địa chỉ gián tiếp xuống 1 đơn vị:

Cú pháp: ***Dec @Ri***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Giảm giá trị dữ liệu ở ô nhớ có địa chỉ bằng giá trị dữ liệu trên Ri xuống1 đơn vị, không ảnh hưởng đến các cờ nhớ trên PSW

Ví dụ:

```
Mov   60h,#05h
Mov    R1,#60h
Dec     @R1
```

Kết quả : R1 = #04h

60h = #05h

3.1.22.Lệnh nhân thanh ghi A với thanh ghi B:

Cú pháp: ***Mul AB***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 4 chu kì máy

Công dụng: Nhân hai dữ liệu là số nguyên không dấu ở thanh ghi A với thanh ghi B, kết quả là một dữ liệu 16 bit. Byte thấp của kết quả lưu ở thanh ghi A và byte cao của kết quả lưu ở thanh ghi B. Nếu tích số lớn hơn 255(0FFH), cờ tràn OV ở thanh trạng thái PSW được thiết lập lên 1, ngược lại nếu tích số nhỏ hơn 255(0FFH),cờ tràn OV được thiết lấp về 0. Cờ nhớ C luôn ở giá trị 0. Ví dụ:

Mov A,#0B9h

Mov B,#F7h

Mul AB

Kết quả : A = #7Fh

B = #0B2h

3.1.23.Lệnh chia thanh ghi A với thanh ghi B:

Cú pháp: ***Div AB***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 4 chu kì máy

Công dụng: Chia hai dữ liệu là số nguyên không dấu ở thanh ghi A với thanh ghi B, dữ liệu ở thanh ghi A là số chia còn ở thanh ghi B là số bị chia, kết quả là một dữ liệu 8 bit được lưu ở thanh ghi A.số dư lưu trữ trong thanh ghi B Cờ nhớ C luôn ở giá trị 0.

Cờ tràn OV được thiết lập giá trị 1 khi thanh ghi B mang giá trị là 00Hphép chia không thể thực hiện.

Ví dụ: *Mov A,#50h*

Mov B,#10h

DIV AB

Kết quả : A = #5h

B = #0h

3.1.24.Lệnh hiệu chỉnh thập phân nội dung của thanh ghi A đối với phép cộng:

Cú pháp: ***DA A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 4 chu kì máy

Công dụng: hiệu chỉnh dữ liệu là giá trị lưu giữ ở thanh ghi A từ số Hex (số nhị phân) thành số BCD (số thập phân viết dưới dạng nhị phân). Lí do có lệnh hiệu chỉnh này vì khi cộng hai giá trị là số BCD bằng các lệnh cộng, vi điều khiển chỉ hiểu hai số cộng là số nhị phân bình thường, kết quả sau lệnh cộng là một số nhị phân bình thường, không phải là một số BCD, vì vậy kết quả cần được hiệu chỉnh để dữ liệu cuối là một số BCD. Khi thực hiện lệnh, cờ nhớ C được xác lập lên 1 nếu phép cộng có kết quả vượt qua 99(số BCD). Kết quả cuối cùng, số BCD có hàng đơn vị nằm ở 4 bit thấp trên thanh ghi A, hàng chục ở 4 bit cao của thanh ghi A, hàng trăm là 1 nếu cờ C mang giá trị 1, là 0 nếu cờ C mang giá trị 0.

Ví dụ 1: Mov A,#10h

Kết quả : A = #10h

Ví dụ 2: Mov A,#0Eh

Kết quả : A = #14h

3.2. Nhóm lệnh logic

3.2.1. Lệnh And dữ liệu ở thanh ghi A với dữ liệu ở thanh ghi Rn:

Cú pháp: ***ANL A,Rn***

Lệnh này chiếm dung lượng bộ nhớ ROM là: 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic AND dữ liệu ở thanh ghi A với dữ liệu ở thanh ghi Rn, kết quả được lưu trữ ở thanh ghi A Ví dụ:

```
        mov     A,#0Fh
mov     R1,#0F0h
        ANL     A,R1
```

Kết quả : A = #0H

3.2.2. Lệnh And dữ liệu trên thanh ghi A với dữ liệu của ô nhớ có địa chỉ direct:

Cú pháp: ***ANL A,direct***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic AND dữ liệu ở thanh ghi A với dữ liệu ở ô nhớ có địa chỉ direct, kết quả được lưu trữ ở thanh ghi A Ví dụ:

```
        mov     A,#0FFh
mov     10h,#010h
        ANL     A,10h
```

Kết quả : A = #010h

3.2.3. Lệnh And dữ liệu trên thanh ghi A với dữ liệu của ô nhớ gián tiếp:

Cú pháp: ***ANL A,@Ri***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic AND dữ liệu ở thanh ghi A với dữ liệu của ô nhớ có địa chỉ bằng giá trị của thanh ghi Ri, kết quả được lưu trữ ở thanh ghi A

Ví dụ:

```
        mov     A,#0Fh
mov             70h,#0E1h
mov     R1,#070h
        ANL     A,@R1
```

Kết quả : A = #01h

3.2.4. Lệnh And dữ liệu trên thanh ghi A với dữ liệu xác định:

Cú pháp: ***ANL A,#data***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic AND dữ liệu ở thanh ghi A với dữ liệu cho trước, kết quả được lưu trữ ở thanh ghi A Ví dụ:

```
        mov     A,#0Eh
        ANL     A,#11h
```

Kết quả : A = #00

3.2.5. Lệnh And dữ liệu của ô nhớ có địa chỉ direct với dữ liệu trên thanh ghi A:

Cú pháp: ***ANL direct,A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic AND dữ liệu ở thanh ghi A với dữ liệu của ô nhớ có địa chỉ direct, kết quả được lưu trữ ở ô nhớ có địa chỉ direct. Ví dụ:

```
        mov     A,#08h
mov     R1,#0F7h
        ANL     R1,A
```

Kết quả : R1 = #0

3.2.6. Lệnh And dữ liệu trên ô nhớ có địa chỉ direct với dữ liệu xác định:

Cú pháp: ***ANL direct,#data***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic AND dữ liệu của ô nhớ có địa chỉ direct với dữ liệu cho trước, kết quả được lưu trữ ở ô nhớ có địa chỉ direct. Ví dụ:

```
mov     R1,#0F7h
ANL   R1,#1Fh
```

Kết quả : R1 = #017h

3.2.7. Lệnh OR dữ liệu ở thanh ghi A với dữ liệu ở thanh ghi Rn:

Cú pháp: ***ORL A,Rn***

Lệnh này chiếm dung lượng bộ nhớ ROM là: 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic OR dữ liệu ở thanh ghi A với dữ liệu ở thanh ghi Rn, kết quả được lưu trữ ở thanh ghi A

Ví dụ:

```
mov     A,#0Fh
mov     R1,#0F0h
ORL     A,R1
```

Kết quả : A = #0FFh

3.2.8. Lệnh OR dữ liệu trên thanh ghi A với dữ liệu của ô nhớ có địa chỉ direct:

Cú pháp: ***ORL A,direct***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic OR dữ liệu ở thanh ghi A với dữ liệu của ô nhớ có địa chỉ direct, kết quả được lưu trữ ở thanh ghi A

Ví dụ:

```
mov     A,#0Eh
mov     50h,#0F0h
ORL     A,50h
```

Kết quả : A = #0FEh

3.2.9. Lệnh OR dữ liệu trên thanh ghi A với dữ liệu của ô nhớ gián tiếp:

Cú pháp: ***ORL A,@Ri***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic OR dữ liệu ở thanh ghi A với dữ liệu của ô nhớ có địa chỉ bằng giá trị của thanh ghi Ri, kết quả được lưu trữ ở thanh ghi A

Ví dụ:

```
        mov         A,#18h
mov                 30h,#0F0h
mov     R1,#30h
        ORL     A,@R1
```

Kết quả : A = #0F8h

3.2.10. Lệnh OR dữ liệu trên thanh ghi A với dữ liệu xác định:

Cú pháp: ***ORL A,#data***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic OR dữ liệu ở thanh ghi A với dữ liệu cho trước, kết quả được lưu trữ ở thanh ghi A

Ví dụ:

```
        mov     A,#00h
        ORL     A,#10h
```

Kết quả : A = #010h

3.2.11. Lệnh OR dữ liệu của ô nhớ có địa chỉ direct với dữ liệu trên thanh ghi A:

Cú pháp: ***ORL direct,A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic OR dữ liệu ở thanh ghi A với dữ liệu của ô nhớ có địa chỉ direct, kết quả được lưu trữ ở ô nhớ có địa chỉ direct. Ví dụ:

```
        mov         A,#0Fh
mov     5Fh,#0F0h
        ORL     5Fh,A
```

Kết quả : 5Fh = #0FFh

3.2.12. Lệnh OR dữ liệu trên ô nhớ có địa chỉ direct với dữ liệu xác định:

Cú pháp: ***ORL direct,#data***

Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: thực hiện phép logic OR dữ liệu của ô nhớ có địa chỉ direct với dữ liệu cho trước, kết quả được lưu trữ ở ô nhớ có địa chỉ direct. Ví dụ:

```
        mov     60h,#0F0h
        ORL     60h,#1Fh
```

Kết quả : 60h = #0FFh

3.2.13. Lệnh EX-OR dữ liệu ở thanh ghi A với dữ liệu ở thanh ghi Rn:

Cú pháp: ***XRL A,Rn***

Lệnh này chiếm dung lượng bộ nhớ ROM là: 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic EX-OR dữ liệu ở thanh ghi A với dữ liệu ở thanh ghi Rn, kết quả được lưu trữ ở thanh ghi A Ví dụ:

```
        mov     A,#0F2h
mov     R3,#0E0h
        XRL     A,R3
```

Kết quả : A = #12h

3.2.14. Lệnh EX-OR dữ liệu trên thanh ghi A với dữ liệu của ô nhớ có địa chỉ direct:

Cú pháp: ***XRL A,direct***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic EX-OR dữ liệu ở thanh ghi A với dữ liệu của ô nhớ có địa chỉ direct, kết quả được lưu trữ ở thanh ghi A Ví dụ:

```
        mov     A,#012h
mov     10h,#0E0h
        XRL     A,10h
```

Kết quả : A = #0F2h

3.2.15. Lệnh EX-OR dữ liệu trên thanh ghi A với dữ liệu của ô nhớ gián tiếp:

Cú pháp: ***XRL A,@Ri***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic EX-OR dữ liệu ở thanh ghi A với dữ liệu của ô nhớ có địa chỉ bằng giá trị của thanh ghi Ri, kết quả được lưu trữ ở thanh ghi A

Ví dụ:

```
        mov     A,#08h
mov     10h,#0E9h
mov     R0,#10h
        XRL     A,@R0
```

Kết quả : A = #0E1h

3.2.16. Lệnh EX-OR dữ liệu trên thanh ghi A với dữ liệu xác định:

Cú pháp: ***XRL A,#data***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic EX-OR dữ liệu ở thanh ghi A với dữ liệu cho trước, kết quả được lưu trữ ở thanh ghi A Ví dụ:

```
mov   A,#12h
XRL   A,#12h
```

Kết quả : A = #0

3.2.17. Lệnh EX-OR dữ liệu của ô nhớ có địa chỉ direct với dữ liệu trên thanh ghi A:

Cú pháp: ***XRL direct,A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thực hiện phép logic EX-OR dữ liệu ở thanh ghi A với dữ liệu của ô nhớ có địa chỉ direct, kết quả được lưu trữ ở ô nhớ có địa chỉ direct. Ví dụ:

```
mov         A,#0F2h
mov   50h,#0E0h
XRL   50h,A
```

Kết quả : 50h = #12h

3.2.18. Lệnh EX-OR dữ liệu trên ô nhớ có địa chỉ direct với dữ liệu xác định:

Cú pháp: ***XRL direct,#data***

Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: thực hiện phép logic EX-OR dữ liệu của ô nhớ có địa chỉ direct với dữ liệu cho trước, kết quả được lưu trữ ở ô nhớ có địa chỉ direct.

Ví dụ:

```
mov         50h,#0E0h
XRL   50h,#01h
```

Kết quả : 50h = #0E1h

3.2.19. Lệnh bù giá trị dữ liệu trên thanh ghi A:

Cú pháp: ***CPL A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: lấy bù giá trị lưu giữ ở thanh ghi A, các bit có giá trị là 1 chuyển thành 0 và ngược lại các bit có giá trị là 0 chuyển thành 1.

Ví dụ: *Mov A,#01100111b ;(tương đương 67h)*
CPL A

Kết quả : A = #10011000b (tương đương 98h)

3.2.20. Lệnh xóa dữ liệu trên thanh ghi A:

Cú pháp: ***CLR A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: tất cả các bit của thanh ghi A đều được xác lập giá trị 0 .

Ví dụ:

Mov A,#01100111b
CLR A

Kết quả : A = #0

3.2.21. Lệnh xoay trái dữ liệu trên thanh ghi A:

Cú pháp: ***RL A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thanh ghi A gồm tám bit A7 A6 A5 A4 A3 A2 A1 A0. Khi thực hiện lệnh xoay trái ***RL A*** giá trị của các bit được chuyển trang bit ở bên trái nó, giá trị của bit A0 chuyển sang bit A1, giá trị của bit A1 chuyển sang bit A2, tương tự với các bit còn lại, và giá trị của bit A7 chuyển sang bit A0.

Ví dụ:

Mov A,#01001001b
RL A

Kết quả sau khi các lệnh được thực hiện A mang giá trị là 10010010b

	Giá trị thanh ghi A
Trước khi thực hiện lệnh xoay trái	0 1 0 0 1 0 0 1
Sau khi thực hiện lệnh xoay trái	1 0 0 1 0 0 1 0

3.2.22. Lệnh xoay trái dữ liệu trên thanh ghi A cùng với cờ nhớ C:

Cú pháp: ***RLC A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thanh ghi A gồm tám bit A7 A6 A5 A4 A3 A2 A1 A0. Khi thực hiện lệnh xoay trái A với cờ nhớ RLC A giá trị của các bit được chuyển trang bit ở bên trái nó, giá trị của bit A0 chuyển sang bit A1, giá trị của bit A1 chuyển sang bit A2, tương tự với các bit còn lại, và giá trị của bit A7 chuyển sang cờ nhớ C, giá trị ở cờ nhớ C chuyển sang bit A0

Ví dụ: giả sử cờ nhớ C đang mang giá trị 1 Mov

A,#11001001b

RLC A

Kết quả sau khi các lệnh được thực hiện A mang giá trị là 10010011b và C mang giá trị 1

	Cờ nhớ C	Giá trị thanh A
Trước khi thực hiện lệnh xoay trái với C	1	1 1 0 0 1 0 0 1
Sau khi thực hiện lệnh xoay trái với C	1	1 0 0 1 0 0 1 1

3.2.23. Lệnh xoay phải dữ liệu trên thanh ghi A:

Cú pháp: ***RR A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thanh ghi A gồm tám bit A7 A6 A5 A4 A3 A2 A1 A0. Khi thực hiện lệnh xoay phải ***RR A*** giá trị của các bit được chuyển trang bit ở bên phải nó, giá trị của bit A7 chuyển sang bit A6, giá trị của bit A6 chuyển sang bit A5, tương tự với các bit còn lại, và giá trị của bit A0 chuyển sang bit A7

Ví dụ:

Mov A,#01001001b

RL A

Kết quả sau khi các lệnh được thực hiện A mang giá trị là 10100100b

	Giá trị thanh A

Trước khi thực hiện lệnh xoay phải	0 1 0 0 1 0 0 1
Sau khi thực hiện lệnh xoay phải	1 0 1 0 0 1 0 0

3.2.24. Lệnh xoay phải dữ liệu trên thanh ghi A cùng với cờ nhớ C:

Cú pháp: ***RRC A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thanh ghi A gồm tám bit A7 A6 A5 A4 A3 A2 A1 A0. Khi thực hiện lệnh xoay phải A với cờ nhớ -***RRC A*** -giá trị của các bit được chuyển trang bit ở bên phải nó, giá trị của bit A7 chuyển sang bit A6, giá trị của bit A6 chuyển sang bit A5, tương tự với các bit còn lại, và giá trị của bit A0 chuyển sang cờ nhớ C, giá trị ở cờ nhớ C chuyển sang bit A7

Ví dụ: giả sử cờ nhớ C đang mang giá trị 1 Mov
A,#11001001b
RLC A

Kết quả sau khi các lệnh được thực hiện A mang giá trị là 11100100b và C mang giá trị 1

	Cờ nhớ C	Giá trị thanh A
Trước khi thực hiện lệnh xoay trái với C	1	1 1 0 0 1 0 0 1
Sau khi thực hiện lệnh xoay trái với C	1	1 1 1 0 0 1 0 0

3.2.25. Lệnh xoay 4 bit trên thanh ghi A:

Cú pháp: ***SWAP A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: hoán chuyển dữ liệu ở 4 bit thấp lên 4 bit cao và 4 bit cao xuống 4 bit thấp Ví dụ:

```
mov     A,#0E7h
SWAP    A
```

Kết quả : A = # 7Eh

3.3. Nhóm lệnh truyền dữ liệu

3.3.1. Lệnh chuyển dữ liệu từ một thanh ghi Rn vào thanh ghi A:

Cú pháp: ***Mov A,Rn***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Chuyển dữ liệu của thanh ghi Rn vào thanh ghi A, dữ liệu trên thanh ghi Rn không đổi

Ví dụ: Giả sử thanh ghi R5 mang dữ liệu với giá trị là 0A5H (10100101B)
Lệnh ***Mov A,R5***

Sau khi lệnh được thực hiện A mang dữ liệu giá trị A5H, Rn không đổi giá trị thanh ghi A trước khi thực hiện lệnh không cần quan tâm

3.3.2. Lệnh chuyển dữ liệu từ ô nhớ có địa chỉ direct vào thanh ghi A:

Cú pháp: ***Mov A,direct***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: chuyển dữ liệu của ô nhớ có địa chỉ bằng direct vào thanh ghi A.

Ví dụ: Giả sử thanh ghi có địa chỉ 33H mang dữ liệu với giá trị là 09H (00001001B)

Lệnh ***Mov A,33H***

Sau khi lệnh được thực hiện A mang dữ liệu giá trị 09H

3.3.3. Lệnh chuyển dữ liệu từ ô nhớ có địa chỉ gián tiếp vào thanh ghi A:

Cú pháp: ***Mov A,@Ri***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: chuyển dữ liệu của ô nhớ 'có địa chỉ bằng giá trị của thanh ghi Ri' vào thanh ghi A.

Ví dụ: Giả sử trước khi thực hiện lệnh ô nhớ có địa chỉ 33H mang dữ liệu với giá trị là 09H (00001001B) và thanh ghi R1 được thiết lập giá trị là 33H Lệnh ***Mov A,@R1***

Khi lệnh được thực hiện A nhận dữ liệu từ ô nhớ có vị trí bằng giá trị được thiết lập trong thanh ghi R1, tức là A nhận dữ liệu từ ô nhớ có địa chỉ là 33H, chú ý: trước đó ô nhớ 33H mang dữ liệu là 09H.

Sau khi lệnh được thực hiện A mang giá trị là 09H (00001001B)

3.3.4. Lệnh đưa dữ liệu vào thanh ghi A

Cú pháp: ***Mov A,#data***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thiết lập dữ liệu cho thanh ghi A

Ví dụ: Muốn thanh ghi A mang dữ liệu có giá trị là 56H ta thực hiện lệnh ***Mov A,#56H***

Sau khi lệnh được thực hiện A mang giá trị là 56H

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

Cú pháp: ***Mov Rn,A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: chuyển dữ liệu từ thanh ghi A vào thanh ghi Rn (n=0-7)

Ví dụ:

Mov A,#56H

Mov R1,A

Sau khi các lệnh được thực hiện R1 mang giá trị là 56H

3.3.6. Lệnh chuyển dữ liệu từ một ô nhớ có địa chỉ direct vào thanh ghi Rn

Cú pháp: ***Mov Rn,direct***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: chuyển dữ liệu của ô nhớ có địa chỉ direct vào thanh ghi Rn (n=0-7)

Ví dụ: giả sử ô nhớ 55H mang dữ liệu có giá trị là A3H

Mov R4,55H

Sau khi các lệnh được thực hiện R4 mang giá trị là A3H

3.3.7. Thiết đặt dữ liệu cho thanh ghi Rn

Cú pháp: ***Mov Rn,#data***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: thiết đặt dữ liệu cho thanh ghi Rn

Ví dụ: Muốn thanh ghi Rn mang dữ liệu có giá trị là 37H ta thực hiện lệnh

Mov A,#37H

Sau khi lệnh được thực hiện A mang giá trị là 37H

3.3.8. Lệnh chuyển dữ liệu từ thanh ghi A vào một ô nhớ có địa chỉ direct

Cú pháp: ***Mov direct,A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: chuyển dữ liệu từ thanh ghi A vào một ô nhớ có địa chỉ direct.

Ví dụ:

Mov A,#77H

Mov 69H,A

Sau khi các lệnh được thực hiện ô nhớ 69H mang giá trị là 77H (giá trị của các bit được thiết lập trong ô nhớ 69H là 01110111B)

3.3.9. Lệnh chuyển dữ liệu từ thanh ghi Rn vào một ô nhớ có địa chỉ direct

Cú pháp: ***Mov direct,Rn***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: chuyển dữ liệu từ thanh ghi Rn vào một ô nhớ có địa chỉ direct

Ví dụ:

Mov Rn,#78H

Mov 7AH,Rn

Sau khi các lệnh được thực hiện ô nhớ 7AH mang giá trị là 78H

3.3.10. Lệnh chuyển dữ liệu từ một ô nhớ có địa chỉ direct này vào một ô nhớ có địa chỉ direct khác

Cú pháp: ***Mov direct,direct***

Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: chuyển dữ liệu từ ô nhớ có địa chỉ direct này vào một ô nhớ có địa chỉ direct khác

Ví dụ:giả sử thanh ghi 20H mang dữ liệu có giá trị là FFH

Mov 22H,20H

Sau khi lệnh được thực hiện thanh ghi 22H mang giá trị là FFH

3.3.11. Lệnh đưa dữ liệu vào ô nhớ có địa chỉ direct

Cú pháp: ***Mov direct,#data***

Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: thiết lập dữ liệu cho ô nhớ có địa chỉ direct

Ví dụ:

Mov 52H,#43H

Sau khi các lệnh được thực hiện ô nhớ 52H mang giá trị là 43H

3.3.12. Lệnh chuyển dữ liệu từ một ô nhớ có địa chỉ gián tiếp vào ô nhớ có địa chỉ direct

Cú pháp: ***Mov direct,@Ri***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: Chuyển dữ liệu của ô nhớ có địa chỉ bằng giá trị của thanh ghi Ri vào ô nhớ có địa chỉ direct

Ví dụ:

```
Mov   30H,#46H
Mov   R0,#30H
Mov   23H, @R0
```

Sau khi các lệnh được thực hiện ô nhớ 23H mang giá trị là 46H

3.3.13. Lệnh chuyển dữ liệu từ thanh ghi A vào ô nhớ có địa chỉ gián tiếp

Cú pháp: ***Mov @Ri,A***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Chuyển dữ liệu của thanh ghi A vào ô nhớ có địa chỉ bằng giá trị của thanh ghi Ri

Ví dụ:

```
Mov   A,#33H
Mov   R1,#22H
Mov   @R0,A
```

Sau khi lệnh được thực hiện ô nhớ 22H mang giá trị là 33H

3.3.14. Lệnh chuyển dữ liệu từ một ô nhớ có địa chỉ direct vào ô nhớ có địa chỉ gián tiếp

Cú pháp: ***Mov @Ri,direct***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: Chuyển dữ liệu của ô nhớ có địa chỉ direct vào ô nhớ có địa chỉ bằng giá trị của thanh ghi Ri

Ví dụ:

```
Mov   4BH,#2AH
```

Mov R0,#2AH

Mov @R0,4BH

Sau khi lệnh được thực hiện ô nhớ 2AH mang giá trị là 2AH

3.3.15. Lệnh đưa dữ liệu vào ô nhớ có địa chỉ gián tiếp

Cú pháp: ***Mov @Ri,#data***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Thiết đặt dữ liệu cho ô nhớ có địa chỉ bằng giá trị của thanh ghi Ri

Ví dụ:

Mov R0,#3BH

Mov @R0,#27H

Sau khi lệnh được thực hiện ô nhớ 3BH mang giá trị là 27H

3.3.16. Lệnh đưa dữ liệu vào con trỏ dữ liệu DPTR

Cú pháp: ***Mov DPTR,#data16***

Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: Thiết đặt dữ liệu cho con trỏ dữ liệu với dữ liệu 16 bit, thực chất dữ liệu được lưu ở hai thanh ghi DPL (byte thấp-địa chỉ byte 82H) và DPH (byte cao-địa chỉ byte 83H). **Ví *dụ*: *Mov DPTR,#3A5FH***

Sau khi lệnh được thực hiện DPTR mang giá trị là 3A5FH

DPL mang giá trị 5FH và DPL mang giá trị 3AH

3.3.17. Lệnh trao đổi dữ liệu giữa ô nhớ có địa chỉ direct với thanh ghi A

Cú pháp: ***XCH A,direct***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Trao đổi dữ liệu của thanh ghi A với ô nhớ có địa chỉ direct, tức là sau khi thực hiện lệnh ô nhớ có địa chỉ direct mang dữ liệu của thanh ghi A trước đó và thanh ghi A mang dữ liệu của ô nhớ có địa chỉ direct.

Ví dụ:

Mov A,#0FAH

Mov 50H,#60H

XCH A,50H

Kết quả : A mang giá trị là 60H

50H mang giá trị là 0FAH

3.3.18. Lệnh trao đổi dữ liệu giữa thanh ghi Rn và thanh ghi A

Cú pháp: ***XCH A,Rn***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Trao đổi dữ liệu của thanh ghi A với thanh ghi Rn.

3.3.19. Lệnh trao đổi dữ liệu giữa thanh ghi có địa chỉ gián tiếp và thanh ghi A

Cú pháp: ***XCH A,@Ri***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Trao đổi dữ liệu của thanh ghi A với ô nhớ có địa chỉ bằng giá trị lưu giữ trong thanh ghi Ri

3.3.20. Lệnh trao đổi dữ liệu 4 bit giữa thanh ghi có địa chỉ gián tiếp và thanh ghi A

Cú pháp: ***XCHD A,@Ri***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 1 chu kì máy

Công dụng: Trao đổi dữ liệu của 4 bit thấp ở thanh ghi A với dữ liệu của 4 bit thấp ở ô nhớ có địa chỉ bằng giá trị lưu giữ trong thanh ghi Ri

3.3.21. Lệnh truy xuất dữ liệu từ ROM nội

Cú pháp: ***MovC A,@A+DPTR***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: Chuyển dữ liệu từ bộ nhớ ROM có địa chỉ bằng giá trị của A cộng với DPTR vào thanh ghi A

Các lệnh còn lại trong nhóm lệnh di chuyển

MovC A,@A+PC
MovC A,@i
MovX A,@DPTR
MovX A,@Ri
MovX @DPTR,A
PUSH direct
POP direct

3.4. Nhóm lênh Boolean (xử lý bít)

Qui ước: trong câu lệnh "***bit***" đại diện cho một địa chỉ của bit nhớ

3.4.1. Lệnh xoá cờ nhớ C

- Cú pháp: ***CLR C***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte
- Thời gian thực hiện: 1 chu kì máy
- Công dụng: Xóa cờ nhớ C - tức là đưa giá trị của cờ nhớ C về 0

3.4.2. Lệnh xoá bit

- Cú pháp: ***CLR bit***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte
- Thời gian thực hiện: 1 chu kì máy
- Công dụng: Xóa giá trị của bit nhớ có địa chỉ xác định - tức là đưa giá trị bit đó về 0

3.4.3. Lệnh thiết đặt cờ nhớ C

- Cú pháp: ***SetB C***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte
- Thời gian thực hiện: 1 chu kì máy
- Công dụng: thiết đặt cờ nhớ C - tức là đưa giá trị của cờ nhớ C lên 1

3.4.4. Lệnh thiết đặt giá trị cho bit nhớ

- Cú pháp: ***SetB bit***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte
- Thời gian thực hiện: 1 chu kì máy
- Công dụng: Thiết đặt giá trị bit nhớ có địa chỉ xác định - tức là đưa giá trị bit đó lên 1

3.4.5. Lệnh bù cờ nhớ C

- Cú pháp: ***CPL C***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte
- Thời gian thực hiện: 1 chu kì máy
- Công dụng: đổi giá trị của cờ nhớ C, nếu trước đó C có giá trị 0 chuyển thành 1, và ngược lại nếu trước đó C có giá trị 1 chuyển thành 0

3.4.6. Lệnh bù bit

- Cú pháp: ***CPL bit***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

- Thời gian thực hiện: 1 chu kì máy
- Công dụng: đổi giá trị của bit có địa chỉ xác định, nếu trước đó bit đó có giá trị 0 chuyển thành 1, và ngược lại nếu trước đó bit đó có giá trị 1 chuyển thành 0

3.4.7. Lệnh And cờ nhớ C với bit

- Cú pháp: ***ANL C,bit***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte
- Thời gian thực hiện: 1 chu kì máy
- Công dụng: Thực hiện phép And cờ nhớ C và bit có địa chỉ xác định, kết quả lưu ở C

3.4.8. Lệnh And cờ nhớ C với bit đã được lấy bù

- Cú pháp: ***ANL C,/bit***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte
- Thời gian thực hiện: 1 chu kì máy
- Công dụng: Thực hiện phép And cờ nhớ C và bit có địa chỉ xác định đã được lấy bù, kết quả lưu ở C

3.4.9. Lệnh OR cờ nhớ C với bit

- Cú pháp: ***ORL C,bit***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte
- Thời gian thực hiện: 2 chu kì máy
- Công dụng: Thực hiện phép And cờ nhớ C và bit có địa chỉ xác định, kết quả lưu ở C

3.4.10. Lệnh OR cờ nhớ C với bit đã được lấy bù

- Cú pháp: ***ORL C,/bit***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte
- Thời gian thực hiện: 2 chu kì máy
- Công dụng: Thực hiện phép And cờ nhớ C và bit có địa chỉ xác định đã được lấy bù, kết quả lưu ở C

3.4.11. Lệnh chuyển giá trị bit có địa chỉ xác định vào cờ nhớ C

- Cú pháp: ***Mov C,bit***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte
- Thời gian thực hiện: 1 chu kì máy

- Công dụng: Thực hiện chuyển giá trị của bit có địa chỉ xác định vào cờ nhớ C

3.4.12. Lệnh chuyển giá trị cờ nhớ C vào bit có địa chỉ xác định

- Cú pháp: ***Mov bit,C***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte
- Thời gian thực hiện: 2 chu kì máy
- Công dụng: Thực hiện chuyển giá trị của cờ nhớ C vào bit có địa chỉ xác định

3.5. Nhóm lệnh rẽ nhánh chương trình

Phần phụ chú:

Nhãn: Kí hiệu: ***rel***

- Nhãn là một chuỗi kí tự do người dùng tự đặt dùng để đánh dấu các đoạn chương trình, nhãn này biểu thị địa chỉ của lệnh khi được lưu trên ROM.

- Nhãn chỉ được bắt đầu bằng một kí tự chữ hoặc dấu "_" ,không được bắt đầu bằng số ,không có khoảng trắng và kết thúc bằng dấu hai chấm ":"

- Trong chương trình nhãn không được đặt trùng tên với nhau, và không được trùng với các từ khóa mà chương trình đã sử dụng.

Ví dụ : Các nhãn đúng X1: ;S_2: ;_5:s10: ;...
Các nhãn sai 1X: ; S_2 ;S 5: ;DW: ,LPT :...

Chương trình con: là những đoạn chương trình thực hiện một số lệnh nào đó và được viết ngoài chương trình chính, các chương trình con này được đặt tên bằng một nhãn và kết thúc bằng lệnh RET, chương trình con có thể gọi một chương trình con khác. Chương trình con được chương trình chính sử dụng khi cần thiết bằng các lệnh gọi chương trình con; khi có lệnh gọi chương trình con, Vi điều khiển chuyển về thực hiện các đoạn chương trình của chương trình con, sau khi thực hiện chương trình con Vi điều khiển tiếp tục trở về thực hiện các câu lệnh trong chương trình chính.

- Chương trình con giúp cho chương trình mạch lạc, dễ hiểu hơn, nếu trong chương trình chính có các đoạn chương trình được lặp đi lặp lại nhiều lần thì các đoạn chương trình đó thường được viết thành một chương trình con và truy xuất bằng một câu lệnh gọi chương trình con. Việc sử dụng chương trình con giúp cho việc tìm lỗi và chỉnh sửa chương trình dễ hơn, nếu chương trình chính sử dụng nhiều lần chương trình con, khi cần sửa đổi chỉ cần thay đổi các câu lệnh trong chương trình con.

- Chương trình con bắt đầu bằng một *nhãn* và kết thúc bằng lệnh *Reti*, chương trình con có thể đặt ở đầu hoặc cuối chương trình.

3.5.1. Lệnh gọi chương trình con dùng địa chỉ tuyệt đối

Cú pháp: ***ACall addr11***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: Khi lệnh được thực hiện, Vi điều khiển chuyển về thực hiện các câu lệnh của chương trình con bắt đầu từ địa chỉ ***addr11*** trên Rom, địa chỉ ***addr11*** có thể thay bằng nhãn bắt đầu của một chương trình con. Câu lệnh được thực hiện khi địa chỉ addr11 cách lệnh gọi không quá 2 KByte . Ví dụ: *ACall 45A6H*

3.5.2. Lệnh gọi chương trình con dùng địa chỉ tuyệt đối

Cú pháp: ***ACall addr16***

Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: Khi lệnh được thực hiện, Vi điều khiển chuyển về thực hiện các câu lệnh của chương trình con bắt đầu từ địa chỉ ***addr16*** trên Rom, địa chỉ ***addr16*** có thay bằng nhãn bắt đầu chương trình con. Câu lệnh có thể gọi chương trình con ở bất kì vị trí nào trên Rom vì khoảng cách từ lệnh gọi đến chương trình con là 64 KByte.

3.5.3. Lệnh kết thúc chương trình con

Cú pháp: ***Ret***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: Lệnh này dùng kết thúc chương trình con, khi gặp lệnh này Vi điều khiển quay về thực hiện lệnh ở chương trình chính.

3.5.4. Lệnh kết thúc chương trình con phục vụ ngắt

Cú pháp: ***Reti***

Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: Lệnh này dùng kết thúc chương trình con ngắt, khi gặp lệnh này Vi điều khiển quay về thực hiện lệnh ở chương trình chính.

3.5.5. Lệnh nhảy ngắn đến địa chỉ tuyệt đối

Cú pháp: ***AJMP addr11***

Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: Khi lệnh được thực hiện, Vi điều khiển chuyển về thực hiện các câu lệnh của chương trình bắt đầu tại địa chỉ ***addr11*** trên Rom, địa chỉ ***addr11*** có thể thay bằng nhãn . Câu lệnh chỉ được thực hiện khi vị trí lưu chương trình cần thực hiện cách lệnh gọi không quá 2 KByte

3.5.6. Lệnh nhảy dài đến địa chỉ tuyệt đối

Cú pháp: ***LJMP addr16***

Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte

Thời gian thực hiện: 2 chu kì máy

Công dụng: Khi lệnh được thực hiện, Vi điều khiển chuyển về thực hiện các câu lệnh của chương trình bắt đầu tại địa chỉ ***addr11*** trên Rom, địa chỉ ***addr11*** có thể thay bằng nhãn . Câu lệnh có thể gọi chương trình ở bất kì vị trí nào trên Rom vì khoảng cách từ lệnh gọi đến chương trình con là 64 KByte

3.5.7. Lệnh nhảy tương đối

- Cú pháp: ***SJMP rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte
- Thời gian thực hiện: 2 chu kì máy
- Công dụng: Khi lệnh được thực hiện, Vi điều khiển chuyển đến thực hiện các câu lệnh của chương trình được đánh dấu bằng nhãn. Câu lệnh chỉ được thực hiện địa chỉ của nhãn cách lệnh gọi không quá 128 Byte.(cả tới hoặc lùi)

3.5.8. Lệnh nhảy gián tiếp

- Cú pháp: ***JMP @A+DPTR***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte
- Thời gian thực hiện: 2 chu kì máy
- Công dụng: Khi lệnh được thực hiện, Vi điều khiển chuyển đến thực hiện các câu lệnh của chương trình có địa chỉ trên ROM bằng giá trị của A cộng với giá trị lưu giữ trên DPTR

3.5.9. Lệnh nhảy thuận với cờ Zero

- Cú pháp: ***JZ rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte - Thời gian thực hiện: 2 chu kì máy - Công dụng:

- Nếu cờ Zero có giá trị 1(tức thanh ghi A có giá trị 0), Vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt
- Nếu cờ Zero có giá trị 0(tức thanh ghi A có giá trị khác 0), Vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy)

3.5.10. Lệnh nhảy nghịch với cờ Zero

- Cú pháp: ***JNZ rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte
- Thời gian thực hiện: 2 chu kì máy - Công dụng:

+ Nếu cờ Zero có giá trị 0(tức thanh ghi A có giá trị khác 0), Vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt

+ Nếu cờ Zero có giá trị 1(tức thanh ghi A có giá trị 0), Vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy)

3.5.11. Lệnh nhảy thuận với cờ C

- Cú pháp: ***JC rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte - Thời gian thực hiện: 2 chu kì máy - Công dụng:

+ Nếu cờ C có giá trị 1, Vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt

+ Nếu cờ C có giá trị 0, Vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy)

3.5.12. Lệnh nhảy nghịch với cờ Zero

- Cú pháp: ***JNC rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte - Thời gian thực hiện: 2 chu kì máy - Công dụng:

+ Nếu cờ C có giá trị 0, Vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt

+ Nếu cờ C có giá trị 1, Vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy)

3.5.13. Lệnh nhảy thuận với giá trị của bit nhớ

- Cú pháp: ***JB bit,rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte - Thời gian thực hiện: 2 chu kì máy - Công dụng:

+ Nếu bit nhớ có giá trị 1, Vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt

+ Nếu bit nhớ có giá trị 0, Vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy)

3.5.14. Lệnh nhảy nghịch với giá trị của bit nhớ

- Cú pháp: ***JNC bit,rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte
- Thời gian thực hiện: 2 chu kì máy - Công dụng:

+ Nếu bit nhớ có giá trị 0, Vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt

+ Nếu bit nhớ có giá trị 1, Vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy)

3.5.15. Lệnh nhảy thuận với giá trị của bit nhớ và xóa bit

- Cú pháp: ***JBC bit,rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte - Thời gian thực hiện: 2 chu kì máy - Công dụng:

+ Nếu bit nhớ có giá trị 1, Vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt, đồng thời xóa giá trị chứa trong bit nhớ đó tức là đưa bit nhớ đó về giá trị 0

+ Nếu bit nhớ có giá trị 0, Vi điều khiển thực hiện lệnh kế tiếp (không thực hiện lệnh nhảy)

3.5.16. Lệnh nhảy có điều kiện(so sánh giá trị của thanh ghi A và Rn)

- Cú pháp: ***CJNE A,direct,rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte
- Thời gian thực hiện: 2 chu kì máy
- Công dụng: Vi điều khiển nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt nếu giá trị của thanh ghi A khác giá trị của ô nhớ có địa chỉ direct, nếu bằng nhau Vi điều khiển không nhảy và thực hiện lệnh kế Ảnh hưởng của lệnh đến cờ nhớ C:

+ Nếu giá trị của thanh ghi A ≥ giá trị của ô nhớ có địa chỉ direct thì bit C có giá trị 0

+ Nếu giá trị của thanh ghi A < giá trị của ô nhớ có địa chỉ direct thì bit C có giá trị 1

3.5.17. Lệnh nhảy có điều kiện(so sánh giá trị của thanh ghi A và dữ liệu cho trước)

- Cú pháp: ***CJNE A,#data,rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte
- Thời gian thực hiện: 2 chu kì máy
- Công dụng: Vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt, nếu giá trị của thanh ghi A khác giá trị dữ liệu cho trước, nếu bằng nhau Vi điều khiển không nhảy và thực hiện lệnh kế

Ảnh hưởng của lệnh đến cờ nhớ C:

+ Nếu giá trị của thanh ghi A ≥ giá trị dữ liệu cho trước thì bit C có giá trị 0

+ Nếu giá trị của thanh ghi A < giá trị dữ liệu cho trước thì bit C có giá trị 1

3.5.18. Lệnh nhảy có điều kiện(so sánh giá trị của thanh ghi Rn và dữ liệu cho trước)

- Cú pháp: ***CJNE Rn,#data,rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte
- Thời gian thực hiện: 2 chu kì máy
- Công dụng: Vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt nếu giá trị của thanh ghi Rn khác giá trị dữ liệu cho trước, nếu bằng nhau Vi điều khiển không nhảy và thực hiện lệnh kế.

Ảnh hưởng của lệnh đến cờ nhớ C:

+ Nếu giá trị của thanh ghi A ≥ giá trị dữ liệu cho trước thì bit C có giá trị 0

+ Nếu giá trị của thanh ghi A < giá trị dữ liệu cho trước thì bit C có giá trị 1

3.5.18. Lệnh nhảy có điều kiện(so sánh giá trị của ô nhớ có địa chỉ gián tiếp và dữ liệu cho trước)

- Cú pháp: ***CJNE @Ri,#data,rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte
- Thời gian thực hiện: 2 chu kì máy
- Công dụng: Vi điều khiển nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt nếu giá trị của ô nhớ có địa chỉ bằng giá trị của Ri khác giá trị dữ liệu cho trước, nếu bằng nhau Vi điều khiển không nhảy và thực hiện lệnh kế Ảnh hưởng của lệnh đến cờ nhớ C:

+ Nếu giá trị của ô nhớ có địa chỉ gián tiếp ≥ giá trị dữ liệu cho trước thì bit C có giá trị 0

+ Nếu giá trị của ô nhớ có địa chỉ gián tiếp < giá trị dữ liệu cho trước thì bit C có giá trị 1

3.5.19. Lệnh nhảy có điều kiện kết hợp với lệnh giảm trên thanh ghi Rn

- Cú pháp: ***DJNZ Rn,rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 2 Byte
- Thời gian thực hiện: 2 chu kì máy
- Công dụng: Giảm giá trị của thanh ghi Rn xuống 1 đơn vị, và

+ Nếu giá trị trong thanh ghi Rn khác 0, Vi điều khiển nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt.

+ Nếu giá trị trong thanh ghi Rn bằng 0, Vi điều khiển thực hiện lệnh kế tiếp

3.5.20. Lệnh nhảy có điều kiện kết hợp với lệnh giảm trên ô nhớ có địa chỉ direct

- Cú pháp: ***DJNZ direct,rel***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 3 Byte
- Thời gian thực hiện: 2 chu kì máy
- Công dụng: Giảm giá trị của ô nhớ có địa chỉ direct xuống 1 đơn vị

+ Nếu giá trị trong ô nhớ có địa chỉ direct khác 0, Vi điều khiển nhảy đến thực hiện chương trình tại địa chỉ mà nhãn được đặt.

+ Nếu giá trị trong ô nhớ có địa chỉ direct bằng 0, Vi điều khiển thực hiện lệnh kế tiếp

3.5.21. Lệnh delay 1 chu kì máy

- Cú pháp: ***NOP***
- Lệnh này chiếm dung lượng bộ nhớ ROM là 1 Byte
- Thời gian thực hiện: 1 chu kì máy
- Công dụng: delay trong 1 chu kì máy

BÀI 4: BỘ ĐỊNH THỜI

1. Mở đầu

- Nội dung của chương này khảo sát các bộ định thời (timer) của chíp 8051.

- Một bộ định thời là một chuỗi các Flip – Flop với mỗi Flip – Flop là một mạch chia 2, chuỗi này nhận một tín hiệu ngõ vào làm nguồn xung clock. Xung clock đặt vào Flip – Flop thứ nhất, Flip – Flop này chia đôi tần số xung clock. Ngõ ra của Flip – Flop thứ nhất trở thành nguồn xung clock choa Flip – Flop thứ hai, nguồn xung clock này cũng được chia cho 2,… Vì mỗi một tầng kế tiếp nhau đều chia cho 2 nên một bộ định thời có n tầng sẽ chia tần số xung clock ở ngõ vào của bộ này cho 2^n.

- Ngõ ra của tầng cuối cùng làm xung clock cho một Flip – Flop báo tràn bộ định thời hay còn gọi là cờ tràn (Overflow Flag), cờ tràn này được kiểm tra bởi phần mềm hoặc tạo ra một ngắt. Giá trị nhị phân trong các Flip – Flop của bộ định thời là số đếm của các xung clock từ khi bộ định thời bắt đầu đếm. Thí dụ một bộ định thời 16 bit sẽ đếm từ 0000h đến FFFFh. Cờ tràn được set bằng 1 khi xảy ra tràn số đếm từ FFFFh xuống 0000h.

- Mỗi một tầng là một FF-D kích khởi cạnh âm hoạt động như một mạch chia cho 2 do ta nối ngõ ra $\overline{Q}$ với ngõ vào FF-D cờ đơn giản là một mạch chốt D được set bằng 1 bởi tầng cuối của bộ định thời. Giản đồ thời gian cho thấy tầng thứ nhất (Q_0) chia 2 tần số xung clock, tầng thứ hai chia 4 tần số xung clock và … Số đếm (count) được ghi ở dạng thập phân và được kiểm tra dễ dàng bằng cách khảo sát trạng thái của 3 Flip – Flop. Thí dụ số đếm là 4 xuất hiện khi $Q_2 = 1$, $Q_1 = 0$ và $Q_0 = 0(4_{10} = 100_2)$. Khi số đếm tràn từ 111_2 xuống 000_2, ngõ ra Q_2 có cạnh

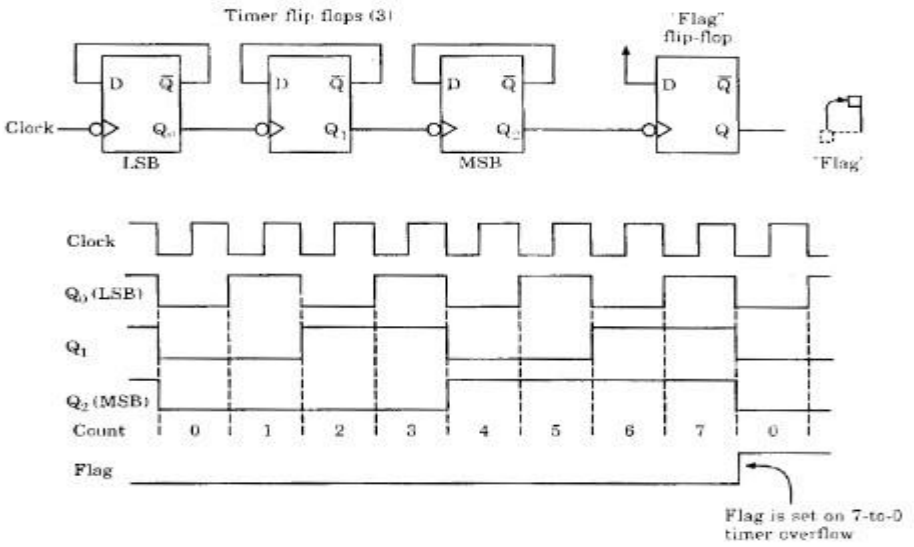
âm làm cho trạng thái của Flip – Flop cờ đổi từ 0 lên 1 (ngõ vào D của Flip – Flop này luôn luôn ở logic 1)

- Bộ định thời được sử dụng trong hầu hết các ứng dụng hướng điều khiển và 8051 với các bộ định thời trên chip không phải là trường hợp ngoại lệ. 8051 có hai bộ định thời 16 bit, mỗi bộ có bốn chế độ hoạt động. Bộ định thời thứ ba với ba chế độ hoạt động được thêm vào đối với chip 8052. Các bộ định thời được dùng để

+ Định thời trong một khoảng thời gian

+ Đếm sự kiện

+ Tạo tốc độ baud cho port nối tiếp của chip 8051



Hình 4.1. Một bộ định thời 3 bit

- Timer flipflops: Các Flip – Flop định thời

- Flag flipflop: Flip – Flop cờ

- Count: Số đếm

- Flag is set on 7 to 0 timer overflow: có được set khi có tràn bộ định thời (số đếm tràn từ 7 xuống 0)

- Với bộ định thời 16 bit, tầng cuối cung (tầng thứ 16) chia tần số xung clock ở ngõ vào của bộ định thời cho $2^{16} = 65536$

- Trong các ứng dụng định thời trong một khoảng thời gian, bộ định thời được lập trình sao cho sẽ tràn sau một khoảng thời gian quy định và set cờ tràn của bộ định thời bằng 1. Cờ tràn được sử dụng để đồng bộ chương trình nhằm thực hiện một công việc như là kiểm tra trạng thái của các ngõ nhập hoặc gởi dữ liệu đến các ngõ xuất. Các ứng dụng khác có thể sử dụng xung clock quy định của bộ định thời để đo khoảng thời gian giữa hai sự kiện (thí dụ đo độ rộng xung).

- Việc đếm sự kiện được dùng đê xác định số lần xuất hiện của một sự kiện hơn là đo thời gian giữa các sự kiện. Từ "sự kiện" là một kích thích bên ngoài cung cấp một chuyển trạng thái từ 1 xuống 0 tới một chân của chip 8051. Các bộ định thời cũng có thể cung cấp xung clock tốc độ baud cho port nối tiếp bên trong 8051.

- Các bộ định thời của 8051 được truy xuất bằng cách sử dụng 6 thanh ghi chức năng đặc biệt. Với bộ định thời thứ ba của chip 8052, ta có thêm 5 thanh ghi chức năng đặc biệt nữa để truy xuất bộ định thời này.

SFR của bộ định thời	Mục đích	Địa chỉ	Định địa chỉ bit
TCON	Điều khiển	88h	Có
TMOD	Chọn chế độ	89h	Không
TL0	Byte thấp của bộ định thời 0	8Ah	Không
TL1	Byte thấp của bộ định thời 1	8Bh	Không
TH0	Byte cao của bộ định thời 0	8Ch	Không
TH1	Byte cao của bộ định thời 1	8Dh	Không
T2CON	Điều khiển bộ định thời 2	C8h	Có
RCAP2L	Nhận byte thấp của bộ định thời 2	CAh	Không
RCAP2H	Nhận byte cao của bộ định thời 2	CBh	Không
TL2	Byte thấp của bộ định thời 2	CCh	Không
TH2	Byte cao của bộ định thời 2	CDh	Không

2. Thanh ghi SFR của timer

2.1. Thanh ghi chế độ TMOD

- Thanh ghi TMOD (timer mode register) chứa hai nhóm 4 bit dùng để thiết lập chế độ hoạt động cho bộ định thời 0 và bộ định thời 1. TMOD không được định địa chỉ từng bit và điều này cũng không cần thiết. Một cách tổng quát, TMOD được nạp một lần bởi phần mềm ở thời điểm bắt đầu của một chương trình để khởi động chế độ hoạt động của bộ định thời. Sau đó, bộ định thời có thể được dừng, bắt đầu. … bằng cách truy xuất các thanh ghi chức năng đặc biệt khác của bộ định thời.

Bit	Tên	Bộ định thời	Mô tả
7	GATE	1	Bit điều khiển công. Khi được set lên 1, bộ định thời chỉ hoạt động trong khi $\overline{INT1}$ ở mức cao

6	$\overline{C/T}$	1	Bit chọn chức năng đếm hoặc định thời 1 = đếm sự kiện 0 = định thời trong một khoảng thời gian
5	M1	1	Bit chọn chế độ thứ nhất
4	M0	1	Bit chọn chế độ thứ hai
3	GATE	0	Bit điều khiển cổng cho bộ định thời 0
2	$\overline{C/T}$	0	Bit chọn chức năng đếm hoặc định thời cho bộ định thời 0
1	M1	0	Bit chọn chế độ thứ nhất
0	M0	0	Bit chọn chế độ thứ hai

Thanh ghi chọn chế độ định thời

M1	M0	Chế độ	Mô tả
0	0	0	Chế độ định thời 13 bit
0	1	1	Chế độ định thời 16 bit
1	0	2	Chế độ tự động nạp lại 8 bit
1	1	3	Chế độ định thời chia xẻ

Hình 4.2. Các chế độ định thời

- Bộ định thời 0: TL0 là một bộ định thời 8 bit được điều khiển bởi các bit chọn chế độ của bộ định thời 0. TH0, tương tự TL0 chỉ khác là được điều khiển bởi các bit chọn chế độ của bộ định thời 1.

- Bộ định thời 1: dừng, không hoạt động

2.2. Thanh ghi điều khiển TCON

- Thanh ghi TCON chứa các bit điều khiển và trạng thái của bộ định thời 0 và bộ định thời 1. Bốn bit cao trong TCON (TCON.4 – TCON.7) được dùng để điều khiển các bộ định thời hoạt động hoặc ngưng (TR0, TR1) hoặc để báo các bộ định thời tràn (TF0, TF1). Các bit này được dùng rộng rãi trong các ví dụ.

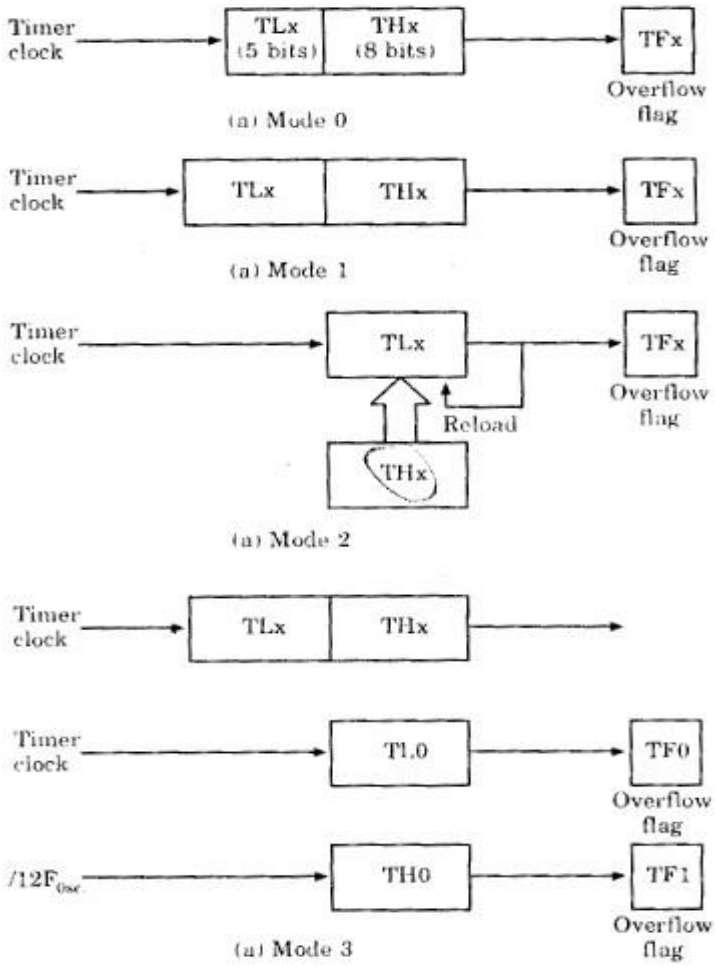
- Bốn bit thấp của TCON (TCON.0 – TCON.3) không dùng để điều khiển các bộ định thời, chúng được dùng để phát hiện và khởi động các ngắt ngoài.

Bit	Ký hiệu	Địa chỉ bit	Mô tả

TCON.7	TF1	8FH	Cớ tràn của bộ định thời 1. Cờ này được set bởi phần cứng khi có tràn, được xóa bởi phần mềm, hoặc bởi phần cứng khi bộ vi xử lý trở đến trình phục vụ ngắt
TCON.6	TR1	8EH	Bit điều khiển hoạt động của bộ định thời 1. Bit này được set hoặc được xóa bởi phần mềm để điều khiển bộ định thời hoạt động hay ngưng hoạt động
TCON.5	TF0	8DH	Cờ tràn của bộ định thời 0
TCON.4	TR0	8CH	Bit điều khiển hoạt động của bộ định thời 0
TCON.3	IE1	8BH	Cờ ngắt bên ngoài 1 (kích khởi cạnh). Cờ này được set bởi phần cứng khi có cạnh âm (xuống) xuất hiện ở chân INT1, được xóa bởi phần mềm, hoặc
			phần cứng khi CPU trở đến trình phục vụ ngắt
TCON.2	IT1	8AH	Cờ ngắt bên ngoài 1 (kích khởi cạnh hoặc mức). Cờ này được set hoặc xóa bởi phần mềm khi xảy ra ở cạnh âm (xuống) hoặc mức thấp tại chân ngắt ngoài
TCON.1	IE0	89H	Cờ ngắt ngoài 0 (kích khởi cạnh)
TCON.0	IT0	88H	Cờ ngắt ngoài 0 (kích khởi cạnh hoặc mức)

Hình 4.3. Thanh ghi điều khiển định thời TCON

3. Các chế độ làm việc



Hình 4.4. Các chế độ định thời

3.1. Chế độ Timer 13 bit (chế độ 0)

- Chế độ 0 là chế độ định thời 13 bit cung cấp khả năng tương thích với bộ vi điều khiển tiền nhiệm 8048. Chế độ này không được được dùng cho các thiết kế mới. Byte cao của bộ định thời THx được ghép cascade với 5 bit thấp của byte thấp bộ định thời TLx để tạo thành một bộ định thời 13 bit. Ba bit cao của TLx không sử dụng.

3.2. Chế độ Timer 16 bit (chế độ 1)

- Chế độ 1 là chế độ định thời 16 bit và có cấu hình giống chế độ định thời 13 bit, chỉ khác nhau ở chỗ bây giờ là bộ định thời 16 bit. Xung clock đặt vào các thanh ghi định thời cao và thấp kết hợp (TLx/THx). Khi có xung clock đến, bộ định thời đếm lên: 0000h, 0001h, 0002h,… Một tràn sẽ xuất hiện khi có sự chuyển số đếm từ FFFFh xuống 0000h. Sự kiện này sẽ set cờ tràn bằng 1 và bộ định thời tiếp tục đếm. Cờ tràn là bit TFx trong thanh ghi điều khiển định thời TCON, bit này được đọc hoặc ghi bởi phần mềm

- Bit có ý nghĩa lớn nhất (MSB) của giá trị trong các thanh ghi định thời là bit 7 của THx và bit có ý nghĩa thấp nhất (LSB) là bit 0 của TLx. Bit LSB thay

đổi trạng thái và chia 2 tần số xung clock định thời ở ngõ vào trong khi bit MSB thay đổi trạng thái và chia cho 65536 (tức là 2^{16}) tần số xung clock định thời ở ngõ vào. Các thanh ghi định thời (TLx/THx) có thể được đọc hoặc ghi ở một thời điểm bất kỳ bởi phần mềm

3.3. Chế độ tự nạp lại 8 bit (Chế độ 2)

- Chế độ 2 là chế độ tự nạp lại 8 bit. Byte thấp của bộ định thời (TLx) hoạt động định thời 8 bit trong khi byte cao của bộ định thời lưu giữ giá trị nạp lại. Khi số đếm tràn từ FFh xuống 00h, không chỉ cờ tràn của bộ định thời được set lên 1 mà giá trị trong THx còn được nạp vào TLx; việc đếm sẽ tiếp tục từ giá trị này cho đến khi xảy ra 1 tràn (FFh -> 00h) kế tiếp,… Chế độ này khá tiện lợi do bởi việc tràn bộ định thời xảy ra ở những khoảng thời gian xác định và tuần hoàn một khi các thanh ghi TMOD và THx đã được khởi động

3.4. Chế độ tách biệt Timer (Chế độ 3)

- Chế độ 3 có hoạt động khác nhau cho từng bộ định thời. Bộ định thời 0 ở chế độ 3 được chia thành 2 bộ định thời 8 bit hoạt động riêng rẽ TL0 và

TH0, mỗi bộ định thời sẽ set các cờ tràn tương ứng TF0 và TF1 khi xảy ra tràn

- Bộ định thời 1 không hoạt động ở chế độ 3 nhưng có thể được khởi động bằng cách chuyển bộ định thời này vào một trong các chế độ khác

- Giới hạn duy nhất là cờ tràn TF1 của bộ định thời 1 không bị ảnh hưởng bởi bộ định thời 1 khi bộ này xảy ra tràn vì TF1 được nối với bộ định thời 8 bit TH0.

- Chế độ 3 chủ yếu cung cấp thêm một bộ định thời 8 bit nữa, nghĩa là 8051 có thêm bộ định thời thứ 3. Khi bộ định thời 0 ở chế độ 3, bộ định thời 1 có thể hoạt động hoặc ngưng hoạt động bằng cách chuyển bộ này ra khỏi chế độ 3 hoặc vào chế độ 3. Bộ định thời 1 có thể được sử dụng bởi port nối tiếp (lúc này bộ định thời 1 làm nhiệm vụ của bộ tạo xung clock tốc độ baud) hoặc được sử dụng theo một cách nào đó nhưng không yêu cầu ngắt

(vì bộ đinh thời 1 lúc này không còn nối với TF1)

4. Nguồn cung cấp xung cho Timer

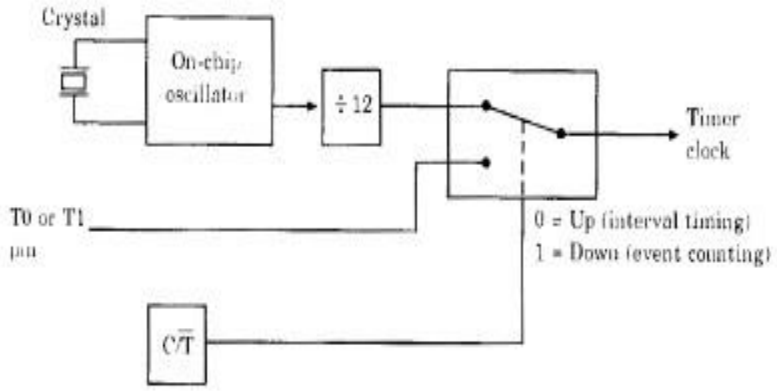
4.1. Chức năng định thời

- Nếu $C/\overline{T} = 0$, hoạt động định thời được chọn và nguồn xung clock của bộ định thời do mạch dao động bên trong chip tạo ra. Một mạch chia 12 tầng được thêm vào để giảm tần số xung clock đến một giá trị thích hợp với hầu hết các ứng dụng. Lúc này bộ định thời được dùng để định thời trong một khoảng thời gian. Các thanh ghi định thời (TLx/THx) đếm lên với tần số xung clock bằng 1/12 tần số của mạch dao động trên chip (định nghĩa thạch anh là 12MHz, tần số xung

clock là 1MHz). Bộ định thời sẽ tràn sau một số xung clock cố định phụ thuộc vào giá trị ban đầu nạp cho các thanh ghi định thời (TLx/THx).

4.2. Chức năng đếm sự kiện

- Nếu $C/\overline{T} = 1$, bộ định thời được cung cấp xung clock từ 1 nguồn tạo xung bên ngoài. Trong đa số các ứng dụng, nguồn xung clock này cung cấp cho bộ định thời một xung dựa trên việc xảy ra một sự kiện – bộ định thời bây giờ đếm sự kiện. Số các sự kiện được xác định trong phần mềm bằng cách đọc các thanh ghi định thời (TLx/THx), giá trị 16 bit trong các thanh ghi này tăng theo mỗi sự kiện. Hai chân của port 3 (P3.4 và P3.5) bây giờ trở thành ngõ vào xung clock cho các bộ định thời. Chân P3.4 làm ngõ vào xung clock cho bộ định thời 0 (ta còn gọi là chân T0 ở ngữ cảnh này), chân P3.5 hoặc T1 là ngõ vào xung clock cho bộ định thời 1



- Crytal: tinh thể thạch anh
- On-chip oscilator: bộ dao động bên trong chip
- T0 or T1 pin: Chân T0 hoặc T1

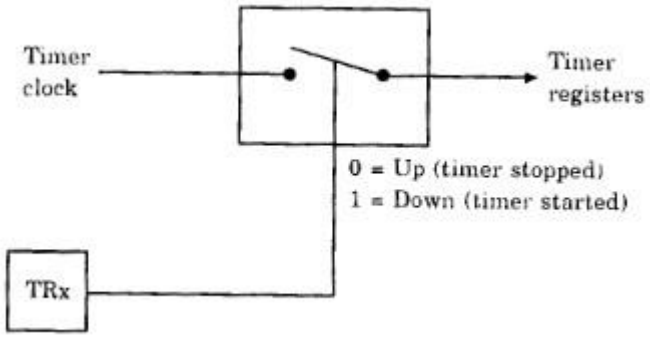
0 = Up (interval timing) : 0 = vị trí trên (đinh thời một khoảng thời gian)

1 = Down (even counting) : 1 = vị trí dưới (đếm sự kiện)

- Timer clock: xung clock định thời
- Trong các ứng dụng đếm sự kiện, các thanh ghi định thời tăng mỗi khi xảy ra chuyển trạng thái từ 1 xuống 0 ở ngõ vào Tx (T0 hoặc T1). Ngõ vào Tx được lấy mẫu trong suốt thời gian S5P2 của mỗi một chu kỳ máy, vậy thì khi ngõ vào ở mức cao trong một chu kỳ và mức thấp trong chu kỳ kế, số đếm được tăng. Giá trị mới xuất hiện trong các thanh ghi định thời trong suốt thời gian S3P1 của chu kỳ tiếp theo chu kỳ phát hiện sự chuyển trạng thái. Từ đó ta thấy phải mất 2 chu kỳ máy ($2\square s$) để nhận biết sự chuyển trạng thái từ 1 xuống 0, tần số cực đại của nguồn xung clock bên ngoài là 500KHz (với giả sử chip vi điều khiển hoạt động với thạch anh 12 MHz).

5. Khởi động, dừng, điều khiển Timer

- Cách đơn giản nhất để khởi động và dừng các bộ định thời là sử dụng bit điều khiển hoạt động TRx trong thanh ghi TCON. TRx được xóa khi thiết lập lại hệ thống, nghĩa là các bộ định thời ngưng hoạt động
- Do thanh ghi TCON là thanh ghi được định địa chỉ từng bit, ta dễ dàng khởi động và dừng các bộ định thời bằng chương trình



Hình 4.5. Bắt đầu và dừng các bộ định thời

- Timer clock: xung clock định thời
- Timer registers: các thanh ghi định thời

0 = Up (timer stopped): 0 = vị trí trên (bộ định thời dừng)

1 = Down (timer started): 1 = vị trí dưới (bộ định thời được khởi động)

Thí dụ bộ định thời 0 được khởi động bằng lệnh

SETB TR0

Và được điều khiển dừng bằng lệnh

CLR TR0

Trình dịch hợp ngữ sẽ thực hiện việc biến đổi ký hiệu "TR0" thành địa chỉ bit. Lệnh SETB TR0 đồng nghĩa với lệnh SETB 0Ch

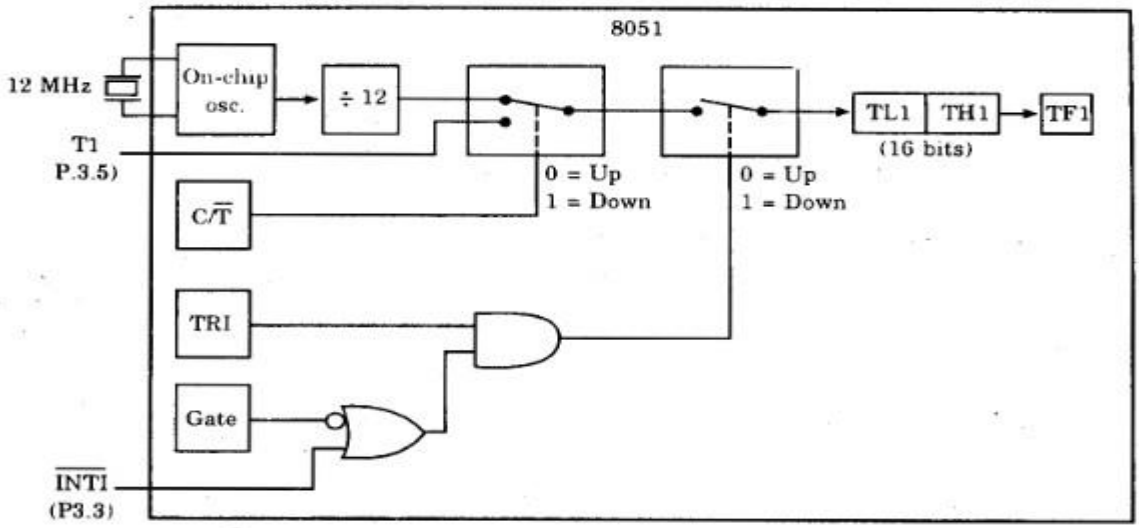
- Một phương pháp khác để điều khiển các bộ định thời là sử dụng bit GATE trong thanh ghi TMOD và ngõ vào $\overline{INTx}$. Bằng cách set bit GATE lên 1 ta cho phép bộ định thời được điều khiển bởi $\overline{INTx}$. Phương pháp này thường được dùng để đo độ rộng xung như được trình bày sau đầy

- Giả sử $\overline{INT0}$ ở mức thấp rồi chuyển sang mức cao trong một khoảng thời gian và ta đo khoảng thời gian này. Ta khởi động bộ định thời 0 ở chế độ 2, chế độ định thời 16 bit với TL0/TH0 = 0000h, GATE = 1 và TR0 = 1.

Khi $\overline{INT0}$ chuyển lên mức cao, bộ định thời được mở cổng và nhận xung

clock có tần số 1 MHz. Khi $\overline{INT0}$ chuyển xuống mức thấp, bộ định thời bị khóa cổng không nhận xung clock nữa và độ rộng xung tính bằng μs là số

đếm trong TL0/TH0 ($\overline{INT0}$ có thể được lập trình để tạo ra một ngắt khi chân này chuyển trở về mức thấp).



On – chip osc: mạch dao động bên trong chip

6. Khởi tạo và truy xuất thanh ghi Timer

6.1. Đọc thời gian đang hoạt động

- Trong một số ứng dụng ta cần phải đọc giá trị (nội dung) chứa trong các thanh ghi định thời đang hoạt động.

- Do ta phải đọc 2 thanh ghi định thời bằng 2 dòng lệnh liên tiếp (do không có lệnh đọc đồng thời cả hai thanh ghi định thời này), một sai pha (phase error) có thể xuất hiện nếu có tràn từ byte thấp chuyển sang byte cao giữa 2 lần đọc và do vậy ta không thể đọc đúng được giá trị cần đọc. Giải pháp đưa ra là trước tiên ta phải đọc byte cao, kế đến đọc byte thấp và rồi đọc byte cao lần nữa. Nếu byte cao thay đổi giá trị, ta lặp lại các thao tác đọc vừa nêu. Các lệnh sau đây đọc nội dung của các thanh ghi định thời

TL1/TH1, đưa vào các thanh ghi R6/R7 và giải quyết vấn đề vừa nêu.

```
        AGAIN:   MOV A, TH1
                 MOV R6, TL1
CJNE A, TH1, AGAIN   MOV R7, A
```

6.2. Thời gian ngắn và thời gian dài

- Tầm nào của các khoảng thời gian có thể định thời được? Vấn đề này được khảo sát bằng cách giả sử 8051 hoạt động với thạch anh nối với mạch dao động nội có tần số hoạt động là 12 MHz. Mạch dao động trên chip được chia cho 12 và

các xung clock cấp cho mạch định thời có tần số 1 MHz. Khoảng thời gian ngắn nhất có thể định thời được bị giới hạn không phải bởi tần số của xung clock định thời mà bởi phần mềm nghĩa là thời gian thực thi các lệnh tạo ra giới hạn đối với các khoảng thời gian định thời rất ngắn. Lệnh ngắn nhất của 8051 thực hiện trong một chu kỳ máy hay 1□*s* .

Khoảng thời gian	Kỹ thuật
□10	Điều chỉnh phần mềm
256	Bộ định thời 8 bit tự động nạp lại
65536	Bộ định thời 16 bit
Không giới hạn	Bộ định thời 16 bit + các vòng lặp

Hình 4.6. Khoảng thời gian định thời cực đại (□s)

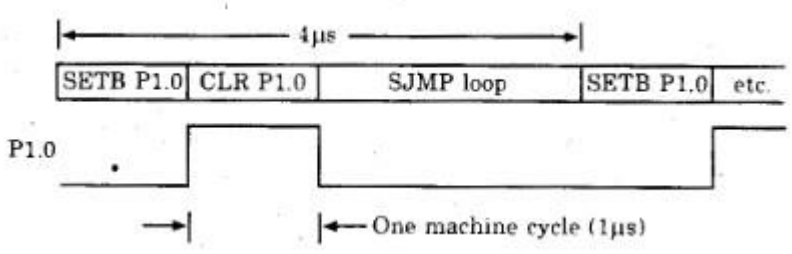
Ví dụ 1: Tạo dạng xung

Viết 1 chương trình tạo dạng xung tuần hoàn trên chân P1.0 có tần số cao có thể có được. Tần số và chu kỳ nhiệm vụ của dạng xung là bao nhiêu?

- Các khoảng thời gian rất ngắn (có nghĩa là tần số cao) có thể được lập trình mà không cần sử dụng đến các bộ định thời. Ví dụ chương trình sau

```
        ORG 8100H
LOOP:   SETB P1.0       ; 1 chu kỳ máy
        CLR P1.0        ; 1 chu kỳ máy
        SJMP LOOP       ; 2 chu kỳ máy
        END
```

- Chương trình trên tạo ra dạng xung trên chân P1.0, xung có chu kỳ là 4□*s* : thời gian mức cao là 1□*s* và thời gian mức thấp là 3□*s* trong một chu kỳ. Tần số của xung là 250 KHz và chu kỳ nhiệm vụ là 25%



- SJMP LOOP: vòng lặp SJMP

- One machine cycle (1□*s*): một chu kỳ máy (1□*s*)

- Lệnh SETB P1.0 thực tế không set bit 0 của port 1 lên 1 cho đến khi kết thúc lệnh này, trong thời gian S6P2, và các lệnh tiếp theo cũng tương tự. Chu kỳ

của tín hiệu ngõ ra có thể kéo dài thêm một ít bằng cách chèn các lênh NOP (không toán hạng) vào trong vòng lặp. Mỗi lệnh NOP được chèn thêm làm cho chu kỳ của tín hiệu ngõ ra được cộng thêm 1μs . Thí dụ nếu ta chèn 2 lệnh NOP sau lệnh SETB P1.0, chương trình sẽ tạo ở ngõ ra một xung vuông có chu kỳ 6μs và tần số là 166.7 KHz. Tuy vậy ta cần thấy rằng việc điều chỉnh phần mềm như trên sẽ trở nên cồng kềnh và vụng về, cách lựa chọn tốt nhất để trì hoãn vẫn là sử dụng bộ định thời

- Các khoảng thời gian dài vừa phải dễ dàng nhận được bằng cách sử dụng chế độ tự nạp lại 8 bit, chế độ 2. Do các khoảng thời gian cần được định thời được thiết lập bởi một số đếm 8 bit, khoảng thời gian dài nhất có thể có trước khi tràn là $2^8 = 256\mu s$

Ví dụ 2: Sóng vuông 10 KHz

Viết chuonwg trình tạo sóng vuông 10 KHz trên chân P1.0 bằng cách sử dụng bộ định thời 0

- Sóng vuông 10 KHz yêu cầu chu kỳ 100μs với thời gian mức cao là 50μs và thời gian mức thấp là 50μs . Do khoảng thời gian này nhỏ hơn 256μs nên chế độ 2 được sử dụng. Một tràn xảy ra sau mỗi 50μs yêu cầu một giá trị số đếm nhỏ hơn 00h một lượng +50 phải được nạp và nập lại cho

TL0, nghĩa là giá trị nạp cho TH0 là -50. Dưới đây là chương trình

```
        ORG 8100H
        MOV TMOD, #02H      ; chế độ tự nạp lại 8 bit
        MOV TH0, #-50H      ; TH0 chứa giá trị -50
        SETB TR0            ; bộ định thời hoạt động
LOOP:   JNB TF0, LOOP       ; chờ tràn
        CLR TF0             ; xóa cờ tràn
        CPL P1.0            ; đổi trạng thái bit P1.0
        SJMP LOOP           ; lặp lại
        END
```

- Chương trình trên sử dụng lệnh lấy bù bit CPL thay vì là lệnh SETB và CLR như ví dụ 1. Giữa hai thao tác lấy bù có một trì hoãn.

7. Timer 2 của 8052

Bit	Ký hiệu	Địa chỉ bit	Mô tả
T2CON.7	TF2	CFH	Cờ tràn của bộ định thời 2. (không được set khi TCLK hoặc RCLK = 1)

T2CON.6	EXF2	CEH	Cờ ngoài của bộ định thời 2. Cờ được set khi có sự thu nhận hoặc nạp lại xảy ra do bởi sự chuyển trạng thái 1 -> 0 ở chân T2EX và EXEN2 = 1; khi các ngắt do bộ định thời được phép, EXF2 = 1 làm cho CPU trỏ tới trình phục vụ ngắt. EXF2 được xóa bởi phần mềm.
T2CON.5	RCLK	CDH	Clock thu của bộ định thời 2. Khi được set, bộ định thời 2 cung cấp tốc độ baud (khi thu) cho port nối tiếp; bộ định thời 1 cung cấp tốc độ baud (khi phát)
T2CON.4	TCLK	CCH	Clock phát của bộ định thời 2. Khi được set, bộ định thời 2 cung cấp tốc độ baud phát; bộ định thời 1 cung cấp tốc độ baud thu
T2CON.3	EXEN2	CBH	Cho phép từ bên ngoài. Khi được set, việc thu nhận và nạp lại xuất hiện khi có sự chuyển trạng thái 1 -> 0 ở chân T2EX
T2CON.2	TR2	CAH	Bit điều khiển hoạt động bộ định thời 2. Bit này được set hay xóa bởi phần mềm để điều khiển bộ định thời 2 hoạt động hoặc ngưng
T2CON.1	$\overline{C/T2}$	C9H	Bit chọn chức năng đếm hoặc định thời của bộ định thời 2. 1 = đếm sự kiện; 0 = định thời một khoảng thời gian.
T2CON.0	$CP/\overline{RL2C}$	C8H	Cờ thu nhận/nạp lại của bộ định thời 2. Khi được set, việc thu nhận xuất hiện khi có sự chuyển trạng thái 1 -> 0 ở T2EX nếu
			EXEN2 = 1; khi được xóa, tự động nạp lại xuất hiện khi có tràn bộ định thời hoặc sự chuyển trạng thái ở T2EX nếu EXEN2 = 1; nếu RCLK hoặc TCLK = 1, bit này được bỏ qua.

Hình 4.7. Thanh ghi điều khiển định thời T2CON

- Việc nạp lại xảy ra khi có tràn số đếm từ FFFFh xuống 0000h ở TL2/TH2 và thiết lập cờ TF2 bằng 1. Điều kiện này được xác định bởi phần mềm hoặc được lập trình để tạo ra một ngắt. Với cả 2 cách vừa nêu, TF2 phải được xóa bởi phần mềm trước khi cờ này được set lần nữa

- Bằng cách set bit EXEN2 trong thanh ghi T2CON bằng 1, việc nạp lại cũng xuất hiện khi có sự chuyển trạng thái 1 -> 0 của tín hiệu đặt vào chân T2EX (chân P1.1). Sự chuyển trạng thái 1 -> 0 ở T2EX cũng thiết lập bằng 1 cho một bit cờ mới trong bộ định thời 2: bit EXF2. Cùng vời TF2, bit EXF2 cũng được kiểm tra bởi phần mềm hoặc tạo ra một ngắt. EXF2 phải được xóa bởi phần mềm.

- Chế độ thu nhận

+ Khi CP/RL2 = 1, chế độ thu nhận được chọn. Bộ định thời 2 hoạt động như một bộ định thời 16 bit và thiết lập cờ TF2 bằng 1 khi xảy ra tràn số đếm từ FFFFh xuống 0000h ở TL2/TH2. Trạng thái của TF2 được kiểm tra bởi phần mềm hoặc tạo ra 1 ngắt.

+ Để cho phép chế độ này hoạt động, bit EXEN2 trong T2CON phải được set bằng 1. Nếu EXEN2 = 1, sự chuyển trạng thái 1 -> 0 ở T2EX (P1.1) sẽ đưa giá trị trong các thanh ghi định thời TL2/TH2 vào trong các thanh ghi RCAP2L và RCAP2H. Việc nạp giá trị này được điều khiển bởi xung clock. Cờ EXF2 trong T2CON cũng được set và như đã đề cập ở phần trên, được kiểm tra bởi phần mềm hoặc tạo ra một ngắt.

BÀI 5: CỔNG NỐI TIẾP

1. Mở đầu

- Bên trong chip 8051 có một port nối tiếp hoạt động ở một vài chế độ trên một tầm tần số rộng. Chức năng cơ bản của port nối tiếp là thực hiện việc chuyển đổi dữ liệu song song thành nối tiếp khi phát và chuyển đổi dữ liệu nối tiếp thành song song khi thu.

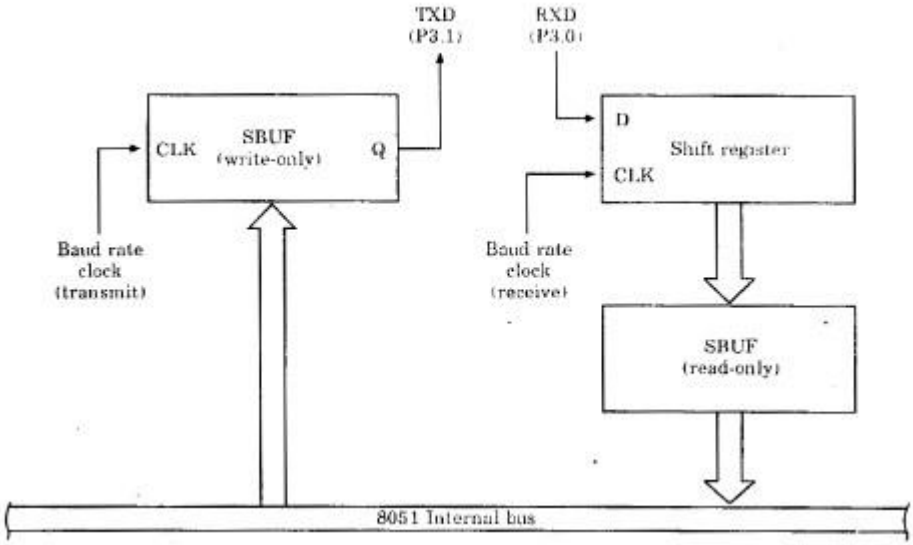
- Các mạch phần cứng bên ngoài truy xuất port nối tiếp thông qua các chân TxD (phát dữ liệu) và RxD (thu dữ liệu) đã được giới thiệu ở chương 2. Các chân này đa hợp với hai chân của port 3 là P3.1 (TxD) và P3.0 (RxD).

- Đặc trưng của port nối tiếp là hoạt động song công (full duplex) nghĩa là có khả năng thu và phát đồng thời. Ngoài ra port nối tiếp còn có một đặc trưng khác, việc đệm dữ liệu khi thu của port 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, dữ liệu sẽ không bị mất.

- Phần mềm sử dụng hai thanh ghi chức năng đặc biệt SBUF và SCON để truy xuất port nối tiếp. Bộ đệm của port nối tiếp SBUF có địa chỉ byte là 99H, trên thực tế bao gồm hai bộ đệm. Việc ghi lên SBUF sẽ nạp dữ liệu để phát và việc đọc SBUF sẽ truy xuất dữ liệu đã nhận được. Điều này có nghĩa là ta có hai thanh ghi riêng rẽ và phân biệt: thanh ghi phát (chỉ ghi) hay bộ đệm phát và thanh ghi thu (chỉ đọc) hay bộ đệm thu

- Thanh ghi điều khiển port nối tiếp SCON (có địa chỉ byte là 98H) là thanh ghi được định địa chỉ từng bit, thanh ghi này chứa các bit trạng thái và các bit điều khiển. Các bit điều khiển sẽ thiết lập chế độ hoạt động cho port nối tiếp còn các bit trạng thái chỉ ra sự kết thúc việc thu hoặc phát một ký tự. Các bit trạng thái được kiểm tra bởi phần mềm hoặc được lập trình để tạo ra ngắt.

- Tần số hoạt động của port nối tiếp, hay còn gọi là tốc độ baud (baud rate) có thể cố định hoặc thay đổi. Khi tốc độ baud thay đổi được sử dụng, bộ định thời 1 cung cấp xung clock tốc độ baud và ta phải lập trình sao cho phù hợp. Trên 8032/8052, bộ định thời 2 cũng có thể được lập trình để cung cấp xung clock tốc độ baud



Hình 5.1. Sơ đồ khối của port nối tiếp

Write only: chỉ ghi

Read only: chỉ đọc

Shift register: thanh ghi dịch bit

Baud rate clock (transmit): xung clock tốc độ baud (phát)

Baud rate clock (receive): xung clock tốc độ baud (thu)

8051 internal bus: bus nội của 8051

2. Thanh ghi điều khiển

- Chế độ hoạt động của port nối tiếp được thiết lập bằng cách ghi từ điều khiển lên thanh ghi chọn chế độ SCON của port nối tiếp ở địa chỉ byte 99H

- Trước khi sử dụng port nối tiếp, thanh ghi SCON phải được khởi động đúng chế độ yêu cầu như lệnh sau

MOV SCON, #01010010B

- Khởi động port nối tiếp ở chế độ 1 (SM0/SM1 = 0/1) cho phép thu (REN = 1) và set cờ ngắt phát bằng 1 (TI = 1) để chỉ ra rằng port nối tiếp sẵn sàng phát dữ liệu.

3. Chế độ làm việc

- Port nối tiếp của 8051 có 4 chế độ hoạt động, các chế độ được chọn bằng cách ghi 1 hoặc 0 cho các bit SM0 và SM1 trong thanh ghi SCON. Ba trong số các chế độ hoạt động cho phép truyền không đồng bộ (asynchronous), trong đó mỗi một ký tự được thu hoặc được phát sẽ cùng với một bit start và một bit stop tạo thành một khung (frame).

- Nếu đã làm quen với hoạt động của port nối tiếp theo chuẩn RS-232 trên một máy tính, ta sẽ tìm thấy được sự tương tự ở các chế độ này. Với chế độ thứ tư, port nối tiếp hoạt động như một thanh ghi dịch bit đơn giản. Mỗi một chế độ sẽ được đề cập tóm tắt sau đây.

Bit	Ký hiệu	Địa chỉ bit	Mô tả
SCON.7	SM0	9FH	Bit 0 chọn chế độ của port nối tiếp
SCON.6	SM1	9EH	Bit 1 chọn chế độ của port nối tiếp
SCON.5	SM2	9DH	Bit 2 chọn chế độ của port nối tiếp Bit này cho phép truyền thông đa xử lý ở chế độ 2 và 3, bit RI sẽ không được tích cực nếu bit thứ 9 nhận được là 0
SCON.4	REN	9CH	Cho phép thu. Bit này phải được set để nhận các ký tự

SCON.3	TB8	9BH	Bit phát 8. Bit thứ 9 được phát ở các chế độ 2 và 3; được set và xóa bởi phần mềm
SCON.2	RB8	9AH	Bit thu 8. Bit thứ 9 nhận được
SCON.1	TI	99H	Cờ ngắt phát. Cờ này được set ngay khi kết thúc việc phát một ký tự; được xóa bởi phần mềm
SCON.0	RI	98H	Cờ ngắt thu. Cờ này được set ngay khi kết thúc việc thu một ký tự; được xóa bởi phần mềm

Hình 5.2. Thanh ghi điều khiển port nối tiếp SCON

3.1. Thanh ghi dịch 8 bit

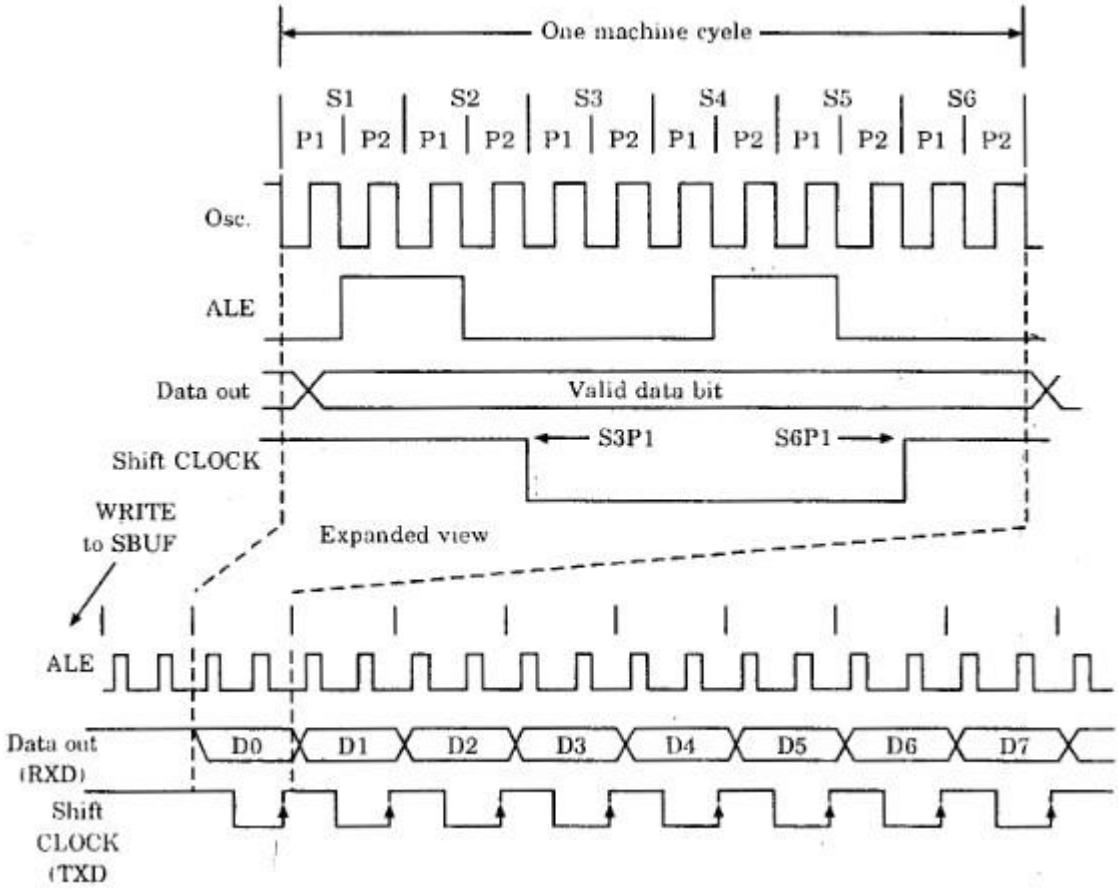
- Chế độ 0, được chọn bằng cách ghi giá trị 0 vào các bit SM0 và SM1 trong thanh ghi SCON, đặt port nối tiếp vào chế độ thanh ghi dịch 8 bit. Dữ liệu nối tiếp được thu và phát thông qua chân RxD, chân TxD xuất xung clock dịch bit. Khi phát và thu dữ liệu 8 bit, bit có ý nghĩa (giá trị vị trí) nhỏ nhất (bit LSB) được thu hoặc phát trước tiên. Tốc độ baud cố định và bằng 1/12 tần số của mạch dao động trên chip. Các thuật ngữ RxD và TxD bị sai lệch ý nghĩa trong chế độ này. Chân RxD được sử dụng cho cả thu và phát dữ liệu còn chân TxD được dùng làm chân xuất xung clock dịch bit.

SM0	SM1	Chế độ	Mô tả	Tốc độ baud
0	0	0	Thanh ghi dịch	Cố định (tần số dao động /12)
0	1	1	UART 8 bit	Thay đổi (thiết lập bởi bộ định thời)
1	0	2	UART 9 bit	Cố định (tần số dao động /12
				hoặc /64)
1	1	3	UART 9 bit	Thay đổi (thiết lập bởi bộ định thời)

Hình 5.3. Các chế độ của port nối tiếp

- Việc phát dữ liệu được khởi động bằng một lệnh ghi dữ liệu vào SBUF. Dữ liệu được dịch ra ngoài trên chân RxD (P3.0) với các xung clock dịch bit được gửi ra trên chân TxD (P3.1). Mỗi một bit hợp lệ được truyền đi trên đường RxD trong một chu kỳ máy.

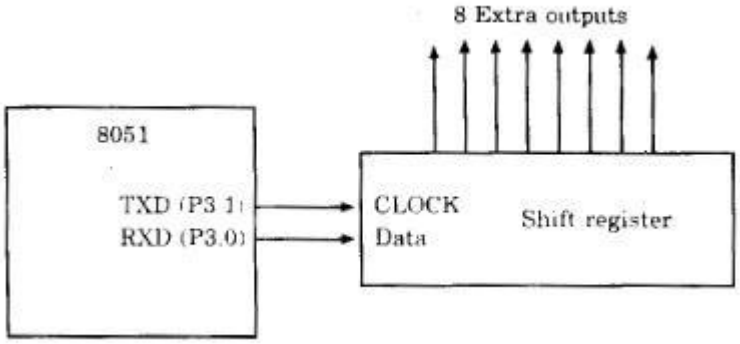
- Trong mỗi một chu kỳ máy, xung clock dịch bit đổi thành mức thấp ở S3P1 và trở lại mức cao ở S6P1.
- Việc thu dữ liệu được khởi động khi bit cho phép thu REN ở logic 1 và cờ ngắt thu RI ở logic 0. Qui luật tổng quát là ta phải set bit REN bằng 1 ở thời điểm bắt đầu chương trình để khởi động port nối tiếp và sau đó xóa bit RI để bắt đầu công việc thu dữ liệu.
- Khi bit RI được xóa, các xung clock dịch bit được xuất ra trên chân TxD, ta bắt đầu chu kỳ máy tiếp theo và dữ liệu được dịch vào chân RxD bởi xung clock dịch bit (hiển nhiên là các mạch ghép nối để cung cấp dữ liệu trên đường RxD được đồng bộ bởi xung clock dịch bit trên đường TxD)



Hình 5.4. Giản đồ thời gian phát dữ liệu ở chế độ 0

- One machene cycle: một chu kỳ máy
- Osc: xung clock của mạch dao động
- ALE: xung ALE
- Data out: dữ liệu xuất
- Valid data bit: bit dữ liệu hợp lệ
- Shift clock: xung clock dịch bit
- Write to SBUF: ghi vào SBUF

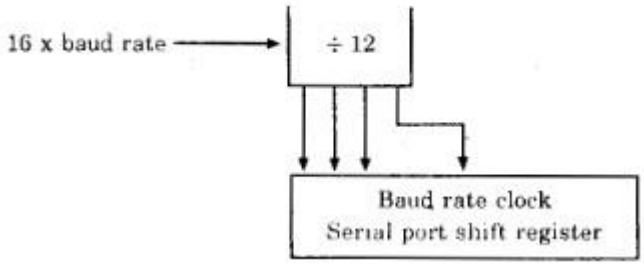
- Expanded view: quan sát được phóng đại
- Việc dịch dữ liệu vào port nối tiếp xảy ra ở cạnh dương (cạnh lên) của TxD
- Một ứng dụng khả thi của chế độ 0 (chế độ thanh ghi dịch bit) là mở rộng thêm các ngõ ra cho 8051. Một vi mạch thanh ghi dịch nối tiếp – song song có thể được nối với các chân TxD và RxD của 8051 để cung cấp thêm 8 đường xuất. Các thanh ghi dịch bit khác có thể ghép cascade với thanh ghi dịch bit đầu tiên để mở rộng thêm nữa.



- Shift register: thanh ghi dịch bit
- 8 extra outputs: 8 ngõ ra mở rộng

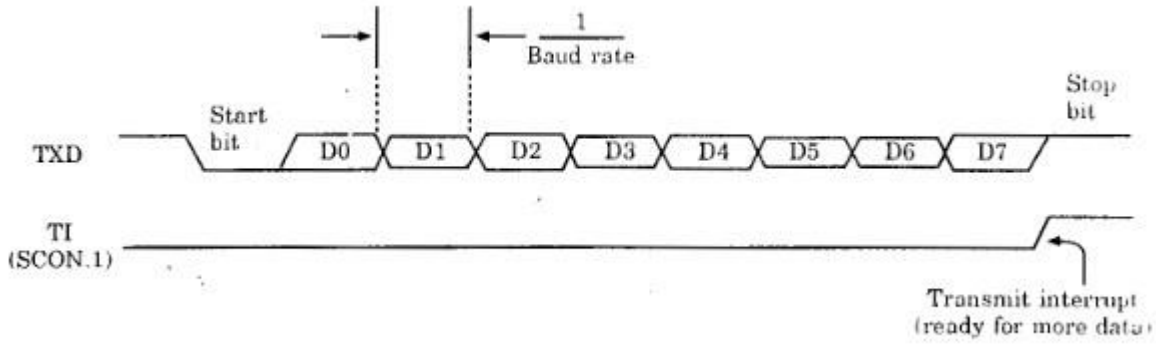
3.2. Chế độ UART 8 bit có tốc độ baud thay đổi

- Trong chế độ 1, port nối tiếp của 8051 hoạt động như một bộ thu phát không đồng bộ. UART 8 bit có tốc độ baud thay đổi. UART là một bộ thu và phát dữ liệu nối tiếp với mỗi một ký tự dữ liệu được đứng trước bởi một bit start (logic 0) và được đứng sau bởi một bit stop (logic 1). Thỉnh thoảng một bit chẵn lẻ được chèn giữa bit dữ liệu sau cùng và bit stop. Hoạt động chủ yếu của một UART là biến đổi dữ liệu phát từ song song thành nối tiếp và biến đổi dữ liệu thu từ nối tiếp thành song song.
- Như vậy ở chế độ 1 ta có 10 bit được thu trên chân RxD và 10 bit được phát trên chân TxD cho mỗi một ký tự dữ liệu, chúng bao gồm 1 bit start (luôn luôn là 0), 8 bit dữ liệu (bit LSB trước tiên) và 1 bit stop (luôn luôn là 1). Khi hoạt động thu, bit stop đưa đến bit RB8 của SCON. Với 8051 tốc độ baut được thiết lập bởi tốc độ tràn (Overflow rate) của bộ định thời 1 còn ở 8052 tốc độ baud được thiết lập bởi tốc độ tràn của bộ định thời 1 hoặc bộ định thời 2 hoặc tổ hợp của cả hai (một cho phát và một cho thu) - Việc cấp xung clock dịch bit và đồng bộ các thanh ghi dịch bit của port nối tiếp ở các chế độ 1,2 và 3 được thiết lập bởi một bộ đếm 16, ngõ ra của bộ đếm là xung clock tốc độ baud. Ngõ vào của bộ đếm vừa nêu được chọn bằng phần mềm và sẽ được trình bày như sau



Hình 5.5. Cấp xung clock cho port nối tiếp

- 16 x baud rate: 16 x tốc độ baud
- Baud rate clock: xung clock tốc độ baud
- Serial port shift register: thanh ghi dịch bit của port nối tiếp
- Việc phát được khởi động bằng cách ghi vào SBUF nhưng việc phát không thực sự bắt đầu cho đến lần tràn kế của bộ đếm 16, bộ đếm cung cấp tốc độ baud cho port nối tiếp. Dữ liệu được dịch bit để được xuất ra trên đường TxD sẽ bắt đầu bằng bit start, tiếp theo là 8 bit dữ liệu rồi đến bit stop. Thời gian của mỗi một bit là giá trị nghịch đảo của tốc độ baud, tốc độ baud có được bằng cách lập trình cho bộ định thời. Cờ ngắt phát TI được set bằng 1 ngay khi bit stop xuất hiện trên đường TxD



Hình 5.6. Set cờ TI của port nối tiếp

- Baud rate: tốc độ baud
- Start bit: bit start - Stop bit: bit stop
- Transmit interrupt: ngắt phát
- Việc nhận được khởi động bởi một chuyển trạng thái từ 1 xuống 0 trên đường RxD (bắt đầu bit start). Bộ đếm 16 ngay lập tức được xóa để gán các số đếm cho dòng bit đến chân RxD (bit kế tiếp đến khi bộ đếm tràn lần nữa). Dòng bit đến được lấy mẫu ở giữa 16 số đếm.
- Bộ thu bao gồm việc phát hiện bit start sai bằng cách yêu cầu 8 số đếm ở trạng thái 0 sau khi có sự chuyển trạng thái từ 1 xuống 0 lần đầu tiên. Nếu điều này không xảy ra, bộ thu được giả sử rằng đã nhận được nhiễu thay vì là nhận một bit hợp lệ. Bộ thu sẽ được thiết lập lại, quay về trạng thái nghĩ

và chờ sự chuyển trạng thái từ 1 xuống 0 kế. Giả sử một bit start hợp lệ được phát hiện, việc nhận ký tự sẽ tiếp tục. Bit start được bỏ qua và 8 bit dữ liệu được nhận tuần tự vào thanh ghi dịch bit của port nối tiếp. Khi cả 8 bit đã được nhận, các điều sau sẽ xảy ra là:

+ Bit thứ 9 (bit stop) được đưa đến bit RB8 trong thanh ghi SCON

+ 8 bit dữ liệu được nạp vào SBUF

+ Cờ ngắt thu RI được set

Tuy nhiên các điều trền chỉ xảy ra nếu các điều kiện sau tồn tại

+ RI = 0

+ SM2 = 1 và bit stop nhận được là bit 1 hoặc SM2 = 0

- Yêu cầu RI = 0 đảm bảo rằng phần mềm đã đọc ký tự trước (và xóa RI). Điều kiện thứ hai có vẻ phức tạp nhưng chỉ áp dụng trong chế độ truyền thông đa xử lý. Yêu cầu có nghĩa là không set RI bằng 1 trong chế độ truyền thông đa xử lý khi bit dữ liệu thứ 9 là 0.

3.3. UART 9 bit với tốc độ baud cố định

- Khi SM1 = 1 và SM0 = 0, port nối tiếp hoạt động ở chế độ 2, chế độ UART 9 bit có tốc độ baud cố định. 11 bit được thu hoặc phát cho việc thu phát một ký tự dữ liệu: bit start, 8 bit dữ liệu, bit dữ liệu thứ 9 lập trình được và bit stop. Khi phát, bit thứ 9 là bit bất kỳ được đặt vào bit TB8 trong thanh ghi SCON (có thể là bit chẵn lẻ). Khi thu bit thứ 9 nhận được sẽ đặt vào bit RB8. Tốc độ baud ở chế độ 2 bằng 1/32 hoặc bằng 1/64 tần số của mạch dao động trên chip

3.4. Chế độ UART với tốc độ baud cố định

- Chế độ 3, UART 9 bit có tốc độ thay đổi, tương tự như chế độ 2 ngoại trừ tốc độ baud được lập trình và được cung cấp bởi bộ định thời. Thật ra các chế độ 1, 2 và 3 đều tương tự nhau. Chúng khác nhau ở tốc độ baud (cố định ở chế độ 2 và thay đổi ở các chế độ 1 và 3) và ở số bit dữ liệu (8 bit ở chế độ 1 và 9 bit ở các chế độ 2 và 3)

4. Khởi tạo và truy suất thanh ghi PORT nối tiếp

4.1. Cho phép nhận

- Bít cho phép thu REN trong thanh ghi SCON phải được set bằng 1 bởi phần mềm để cho phép nhận các ký tự. Điều này thường được thực hiện ở đầu chương trình khi port nối tiếp, các bộ định thời… được khởi động và có thể được thực hiện theo 2 cách.

```
SETB REN
```

Set bit REN bằng 1 hoặc lệnh

```
MOV SCON, #xxx1xxxxB
```

Set bit REN bằng 1 và xóa hoặc set các bit khác trong SCON nếu cần.

4.2. Bít dữ liệu thứ 9

- Bit dữ liệu thứ 9 được phát ở các chế độ 2 và 3 phải được nạp cho bit TB8 bằng phần mềm. Bit dữ liệu thứ 9 thu được phải đặt vào bit RB8 của SCON. Phần mềm có thể yêu cầu hoặc không yêu cầu bit dữ liệu thứ 9 tùy vào các đặc tính của thiết bị nối tiếp mà với thiết bị này việc truyền dữ liệu được thiết lập (bit thứ 9 còn đóng vai trò quan trọng trong truyền thông đa xử lý).

4.3. Thêm vào bít chẵn - lẻ

- Bit thứ 9 thường được dùng làm bit chẵn lẻ cho một ký tự. Bit P trong từ trạng thái chương trình PSW được set hoặc xóa ở mỗi chu kỳ máy để thiết lập việc kiểm tra chẵn cho 8 bit chứa trong thanh chứa A.

- Thí dụ nếu việc truyền thông yêu cầu 8 bit dữ liệu cộng với một bit kiểm tra chẵn, các lệnh sau được dùng để phát đi 8 bit trong thanh chứa với bit kiểm tra chẵn được đưa vào bit thứ 9.

```
MOV C, P        ; đưa bit kiểm tra chẵn vào TB8
MOV TB8, C      ; bit này trở thành bit thứ 9
MOV SBUF, A     ; di chuyển 8 bit dữ liệu từ ACC đến SBUF
```

Nếu kiểm tra lẻ được yêu cầu, các lệnh trên phải được sửa đổi như sau

```
MOV C, P        ; đưa bit kiểm tra chẵn vào TB8
CPL C           ; biến đổi thành kiểm tra lẻ
MOV TB8, C      ; bit này trở thành bit thứ 9
MOV SBUF, A     ; di chuyển 8 bit dữ liệu từ ACC đến SBUF
```

- Dĩ nhiên việc sử dụng bit chẵn lẻ không bị giới hạn ở các chế độ 2 và chế độ 3. Trong chế độ 1, 8 bit dữ liệu được phát đi bao gồm 7 bit dữ liệu cộng với bit chẵn lẻ. Để phát đi một mã ASCII 7 bit cùng với bit kiểm tra chẵn (bit thứ 8) ta có thể sử dụng các dòng lệnh sau.

```
CLR ACC.7       ; đảm bảo bit MSB được xóa, kiểm tra chẵn
MOV C, P        ; sao chép bit P vào C
MOV ACC.7, C    ; đặt bit kiểm tra chẵn vào bit MSB
MOV SBUF, A     ; phát ký tự 7 bít dữ liệu cộng với bít kiểm tra chẵn
```

4.4. Các cờ ngắt

- Các cờ ngắt thu RI và ngắt phát TI trong thanh ghi SCON đóng một vai trò quan trọng trong việc truyền dữ liệu nối tiếp của 8051. Cả 2 bit đều được set bằng 1 bằng phần cứng nhưng phải được xóa bằng phần mềm.

- Điển hình là RI được set bằng 1 khi kết thúc việc nhận một ký tự và chỉ ra rằng bộ đệm thu đầy. Điều kiện này được kiểm tra bằng phần mềm hoặc được lập trình để tạo ra một ngắt

- Nếu phần mềm muốn nhập một ký tự từ một thiết bị ghép với port nối tiếp, phần mềm phải chờ cho đến khi RI được set bằng 1, kế đến phần mềm xóa RI và đọc ký tự từ SBUF. Điều này được thể hiện như sau

```
WAIT:   JNB RI, WAIT    ; Kiểm tra RI cho đến khi bằng 1
        CLR RI          ; xóa RI
        MOV A, SBUF     ; đọc ký tự
```

- Cờ TI được set bằng 1 khi kết thúc việc phát một ký tự và chỉ ra rằng bộ đệm phát rỗng. Nếu phần mềm muốn phát một ký tự đến một thiết bị ghép với port nối tiếp, phần mềm trước tiên phải kiểm tra để biết port nối tiếp đã sẵn sàng. Nói cách khác, nếu một ký tự trước đó đã được phát, phần mềm phải chờ việc phát kết thúc trước khi gửi tiếp ký tự kế

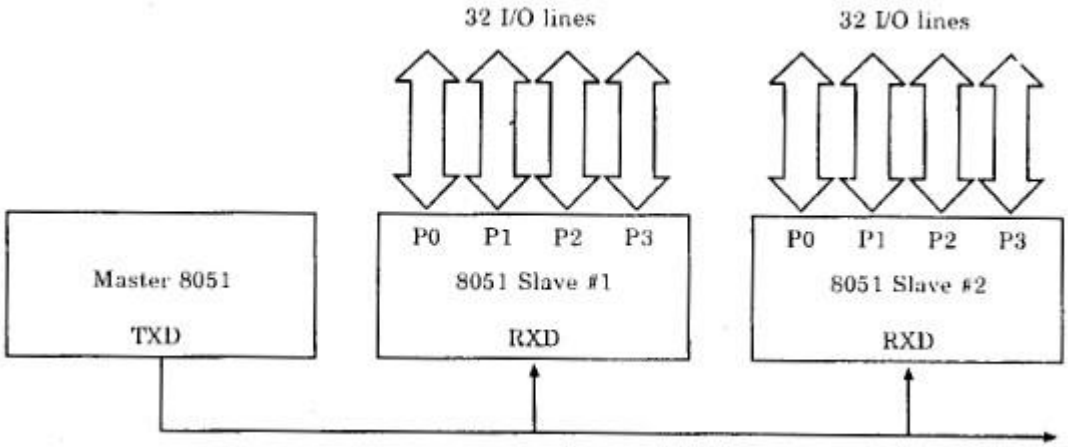
- Các lệnh sau đây phát một ký tự chưa trong thanh chứa

```
WAIT:   JNB TI, WAIT    ; kiểm tra TI cho đến khi bằng 1
        CLR TI          ; xóa TI
        MOV SBUF, A     ; phát ký tự
```

- Các chuỗi lệnh thu và phát ở trên là một phần của các chương trình con xuất nhập ký tự.

5. Truyền thông đa xử lý

- Các chế độ 2 và 3 là các chế độ dự phòng cho việc truyền thông đa xử lý. Trong các chế độ này, 9 bit dữ liệu được thu và bit thứ 9 được đưa đến RB8. Port có thể được lập trình sao cho khi bit stop được nhận, ngắt do port nối tiếp được tích cực chỉ nếu RB8 = 1. Đặc trưng này có được bằng cách set bit SM2 trong thanh ghi SCON bằng 1. Một ứng dụng cho điều này là một môi trường mạng sử dụng nhiều 8051 được sắp xếp theo mô hình chủ/tớ (master/slave).



Hình 5.7. Truyền thông đa xử lý

- Master 8051: 8051 chủ
- 8051 slave #1, #2: 8051 tớ 1, 2
- 32 I/O lines: 32 đường xuất/nhập
- Khi bộ xử lý chủ muốn truyền một khối dữ liệu đến một trong nhiều bộ xử lý tớ, trước tiên bộ xử lý chủ phát đi một byte địa chỉ nhận dạng bộ xử lý tớ đích. Một byte địa chỉ khác với một byte dữ liệu ở chỗ bit thứ 9 là 1 trong byte địa chỉ và là 0 trong byte dữ liệu. Một byte địa chỉ ngắt tất cả các bộ xử lý tớ để cho mỗi một bộ xử lý tớ có thể khảo sát byte nhận được để kiểm tra xem có phải là bộ xử lý tớ đang được định địa chỉ không. Bộ xử lý tớ được định địa chỉ sẽ xóa bit SM2 của mình và chuẩn bị nhận các byte dữ liệu theo sau. Các bộ xử lý tớ không được định địa chỉ có các bit SM2 của chúng được set bằng 1 và thực thi các công việc của riêng chúng, bỏ qua không nhận các byte dữ liệu. Các bộ xử lý này sẽ được ngắt lần nữa khi bộ xử lý chủ phát tiếp byte địa chỉ kế. Các sơ đồ cụ thể có thể được nêu ra sao cho một khi liên kết chủ tớ đã được thiết lập, bộ xử lý tớ cũng có thể phát đến bộ xử lý chủ.
- SM2 không ảnh hưởng đến chế độ 0 và trong chế độ 1 bit này có thể được dùng để kiểm tra sự hợp lệ của bit stop. Ở chế độ 1 thu, nếu SM2 = 1 ngắt thu sẽ không được tích cực trừ phi bit stop thu được là hợp lệ.

6. Tốc độ BAUD

- Tốc độ baud sẽ cố định trong các chế độ 0 và 2. Trong chế độ 0, tốc độ baud luôn luôn bằng tần số của mạch dao động trong chip chia cho 12. Thông thường người ta sử dụng một thạch anh bên ngoài chip cho mạch dao động này. Giả sử tần số của mạch dao động là 12 MHz, tốc độ baud của chế độ 0 là 1 MHz

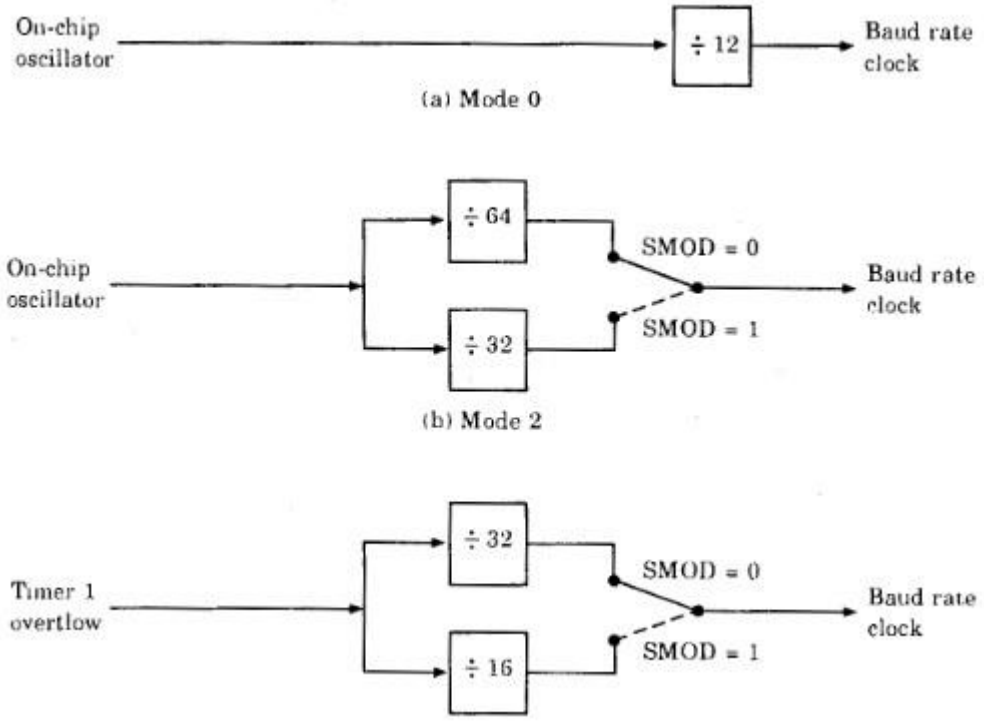
- Sau khi hệ thống được reset, tốc độ baud của chế độ 2 bằng tần số của mạch dao động chia cho 64. Tốc độ baud cũng bị ảnh hưởng bởi một bit trong thanh ghi điều khiển nguồn PCON. Bit 7 của PCON là bit SMOD và việc set bit này bằng 1 sẽ làm tăng tốc độ baud của các chế độ 1, 2 và 3 lên gấp đôi. Ở chế

độ 2, tốc độ baud có thể được nhân 2 từ giá trị mặc định là 1/64 tần số của mạch dao động (SMOD = 0) trở thành 1/32 tần số của mạch dao động (SMOD = 1).

- Vì thanh ghi PCON không được định địa chỉ từng bit, việc set bit SMOD lên 1 mà không làm thay đổi các bit khác của thanh ghi này được thực hiện bằng những dòng lệnh sau

```
MOV A, PCON     ; lấy giá trị hiện hành của PCON
SETB ACC.7      ; set bit 7 bằng 1 (SMOD)
MOV PCON, A     ; ghi giá trị mới vào PCON
```

- Các tốc độ baud của 8051 ở chế độ 1 và chế độ 3 được xác định bởi tốc độ tràn của bộ định thời 1. Vì bộ định thời hoạt động ở tần số tương đối cao, ta cần chia tốc độ tràn cho 32 (hoặc 16 nếu SMOD = 1) trước khi trở thành xung clock tốc độ baud cung cấp cho port nối tiếp. Tốc độ baud của 8052 ở các chế độ 1 và 3 được xác định bởi tốc độ tràn của bộ định thời 1 hoặc bộ định thời 2 hoặc cả hai.



Hình 5.8. Các nguồn xung clock cho port nối tiếp

BÀI 6: NGẮT

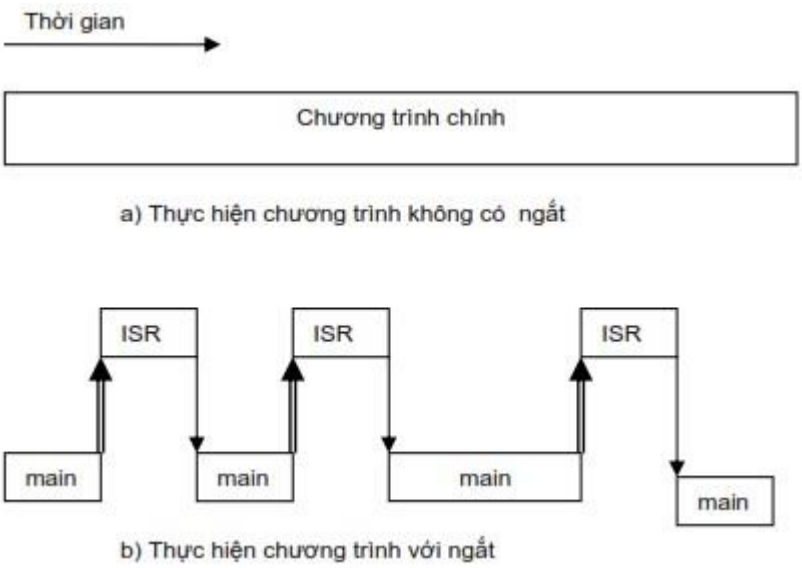
1. Mở đầu

- Ngắt là sự xuất hiện của một điều kiện, một sự kiện làm tạm dừng chương trình trong khi điều kiện này được phục vụ bởi một chương trình khác. Ngắt có một vai trò quan trọng trong thiết kế và thực hiện các ứng dụng của vi điều khiển. Chúng cho phép hệ thống đáp ứng không đồng bộ với một sự kiện và xử lý sự

kiện trong khi một chương trình khác đang hoạt động. Một hệ thống được điều khiển bằng ngắt tạo một ảo giác thực hiện đồng thời nhiều công việc cùng một lúc.. Dĩ nhiên, tại một thời điểm CPU không thể thực hiện nhiều hơn một lệnh nhưng nó có thể tạm dừng chương trình để thực hiện một chương trình khác và sau đó trở lại chương trình đầu tiên. Điểm khác là trong một hệ thống điều khiển bằng ngắt, các ngắt không xảy ra như là kết quả của một lệnh (như lệnh gọi chương trình con) mà là đáp ứng với một sự kiện xảy ra một cách không đồng bộ với chương trình chính có nghĩa là không biết trước chương trình chính sẽ bị ngắt lúc nào.

- Chương trình xử lý ngắt được gọi là chương trình phục vụ ngắt (Interrupt service routine) viết tắt là ISR hay quản lý ngắt. ISR hoạt động để đáp ứng một ngắt và thường thực hiện một thao tác vào hoặc ra đến một thiết bị. Khi xảy ra một ngắt thì chương trình chính tạm thời dừng lại và rẽ nhánh đến ISR. ISR thực hiện các thao tác cần thiết và kết thúc với lệnh trở về từ ngắt và chương trình chính lại tiếp tục từ nơi tạm dừng. Như vậy có thể nói chương trình chính hoạt động ở mức cơ sở và các ISR hoạt động ở mức ngắt cũng có dùng các thuật ngữ:
"phía trước" (foreground) để chỉ mức cơ sở và "phía sau" (backgrround) để chỉ mức ngắt, trong hình 6.1a trình bày hoạt động của một chương trình không có ngắt và 6.1b là hoạt động của chương trình chính ở mức cơ sở có ngắt và các ngắt hoạt động ở mức ngắt.

- Một ví dụ điển hình về ngắt là việc nhập dữ liệu bằng tay dùng bàn phím. Hãy khảo sát một ứng dụng về lò vi sóng: Chương trình chính điều khiển phần tử tạo năng lượng vi sóng để nấu ăn, nhưng trong khi đang nấu hệ thống cần phải đáp ứng việc nhập bằng tay trên cửa lò ví dụ tăng hoặc giảm thời gian nấu. Khi người sử dụng thả nút nhấn, một ngắt được tạo ra (có thể là một tín hiệu chuyển từ mức cao xuống mức thấp) và chương trình chính bị dừng lại, chương trình ISR hoạt động đọc các mã của bàn phím và thay đổi quá trình nấu tương ứng sau đó chấm dứt bằng cách chuyển điều khiển về cho chương tfnh chính, chương trình chính lại tiếp tục từ nơi bị ngắt. Một điểm quan trọng trong ví dụ này là việc nhập bằng tay xảy ra một cách không đồng bộ có nghĩa là không biết trước hoặc không được điều khiển bằng phần mềm đang chạy trong hệ thống. Đó chính là đặc điểm của ngắt



Hình 6.1: Thực hiện chương trình

2. Tổ chức ngắt của 8051

- 8051 có năm nguồn tín hiệu ngắt: 2 ngắt ngoài, 2 ngắt định thời và 1 ngắt cổng nối tiếp. 8052 có thêm ngắt thứ sáu của timer thứ ba. Trạng thái mặc định của các ngắt là không hoạt động sau khi reset hệ thống và chuyển sang hoạt động từng ngắt riêng rẽ bằng phần mềm.
- Trong trường hợp có hai hoặc nhiều ngắt xuất hiện đồng thời hoặc một ngắt xảy ra trong khi một ngắt khác đang được phục vụ. Có hai sơ đồ sắp xếp ưu tiên các ngắt đó là: Chuỗi pooling và ưu tiên hai cấp, thứ tự theo chuỗi pooling thì cố định nhưng sơ đồ ưu tiên hai cấp thì lập trình được. Sau đây là phương pháp cho phép và không cho phép sự hoạt động của các ngắt.

Cho phép và không cho phép các ngắt

- Mỗi một tín hiệu ngắt được cho phép hoặc không cho phép bởi địa chỉ bít trong thanh ghi chức năng đặc biệt IE (Interrupt enable) tại địa chỉ 0A8H, có một bít cho phép toàn cục, bít này khi bị xóa sẽ ngăn tất cả các ngắt. (bảng 6.1).

Để cho phép một ngắt cần phải set hai bít: Một bít cho phép riêng và bít cho phép toàn cục. VD Ngắt timer 1 được cho phép như sau: SETB ET1

```
SETB EA
```

Hoặc

```
MOV IE,#10001000B
```

Mặc dù hai cách trên có cùng kết quả sau khi reset hệ thống nhưng kết quả sẽ khác nhau nếu IE được ghi ở giữa chương trình trong khi đang chạy. Cách thứ nhất không ảnh hưởng đến 5 bít còn lại trong thanh ghi IE còn cách thứ hai sẽ xóa các bít còn lại khác. Tốt nhất nên dùng cách thứ hai tại vị trí bắt đầu chương trình

(nghĩa là khi bắt đầu mở máy hoặc reset hệ thống) nên dùng lệnh SETB và CLR trong khi chương trình đang chạy để tránh ảnh hươngr các bít khác trong thanh ghi IE.

Mức ưu tiên

- Mỗi ngắt được lập trình ở một trong hai mức ưu tiên bằng thanh ghi IP (Interrupt priority) tại địa chỉ 0B8H (bảng 6.2) Thanh ghi IP tự động xóa sau khi reset hệ thống để đặt các ngắt ở mức ưu tiên thấp

- BẢNG 6.1 Thanh ghi IE

BÍT	Ký hiệu	Địa chỉ bít	Mô tả (1=cho phép, 0=không cho phép)
IE.7	EA	AFH	Cho phép toàn cục
IE.6	-	AEH	Không dùng
IE.5	ET2	ADH	Cho phép ngắt timer 2 (8052)
IE.4	ES	ACH	Cho phép ngắt cổng nối tiếp
IE.3	ET1	ABH	Cho phép ngắt timer 1
IE.2	EX1	AAH	Cho phép ngắt 1 ngoài
IE.1	ET0	A9H	Cho phép ngắt timer 0
IE.0	EX0	A8H	Cho phép ngắt 0 ngoài

BẢNG 6.2 Thanh ghi IP

BÍT	Ký hiệu	Địa chỉ bít	Mô tả (1=mức cao, 0=mức thấp)
IP.7	-	-	Không dùng
IP.6	-	-	Không dùng
IP.5	PT2	0BDH	Ưu tiên ngắt timer 2 (8052)
IP.4	PS	0BCH	Ưu tiên ngắt cổng nối tiếp
IP.3	PT1	0BBH	Ưu tiên ngắt timer 1
IP.2	PX1	0BAH	Ưu tiên ngắt 1 ngoài
IP.1	PT0	0B9H	Ưu tiên ngắt timer 0
IP.0	PX0	0B8H	Ưu tiên ngắt 0 ngoài

- Khái niệm "ưu tiên" cho phép một ISR bị dừng bởi một ngắt khác nếu ngắt mới xuất hiện này có mức ưu tiên cao hơn ngắt đang được phục vụ, điều này phù hợp với 8051 vì chỉ có hai mức ưu tiên, nếu một ISR ưu tiên thấp đang chạy nhưng lại xảy ra một ngắt ưu tiên cao thì ISR sẽ bị dừng. Một ISR ưu tiên không thể bị dừng.

- Chương trình chính hoạt động ở mức ưu tiên cơ sở và không liên hệ với một ngắt bất kỳ nào nên luôn bị dừng khi xảy ra ngắt. Nếu hai ngắt có mức ưu tiên khác nhau cùng xảy ra thì ngắt có mức ưu tiên cao sẻ được phục vụ trước.

Chuỗi pooling

- Nếu đồng thời xuất hiện hai ngắt có cùng mức ưu tiên thì ngắt được phục vụ trước được xác định theo thứ tự chuỗi pooling: Ngắt 0 ngoài, ngắt timer 0, ngắt 1 ngoài, ngắt timer 1, ngắt cổng nối tiếp, ngắt timer 2.

- Hình 6.2 trình bày năm nguồn tín hiệu ngắt cùng cơ chế cho phép toàn cục và riêng rẽ, chuỗi pooling và các mức ưu tiên, trạng thái của tất cả các nguồn tín hiệu ngắt có thể thông qua các bít cờ trong thanh ghi chức năng đặc biệt. Dĩ nhiên, nếu một ngắt nào đó không được cho phép thì ngắt tương ứng không được tạo ra nhưng phần mềm vẫn có thể kiểm tra cờ ngắt. Các ví dụ về timer và cổng nối tiếp trong hai bài trước đã sử dụng các cờ ngắt mà thực tế không dùng các ngắt.

- Một ngắt cổng nối tiếp là kết quả từ phép OR của ngắt thu (RI) với ngắt phát (TI). Tương tự, ngắt timer 2 được tạo ra bởi cờ tràn TF2 hoặc với cờ nhập bên ngoài EXF2. Khả năng tạo ngắt của các bít cờ được tóm tắt trong bảng 6.3

3. Xử lý ngắt

- Khi một ngắt xuất hiện được CPU nhận ra, chương trình chính sẽ dừng lại và kế tiếp là các thao tác như sau:

+ Thực hiện hoàn tất lệnh hiện hành

+ Lưu nội dung thanh ghi PC vào ngăn xếp

+ Lưu trạng thái ngắt hiện hành

+ Các ngắt được chặn lại tại mức ngắt

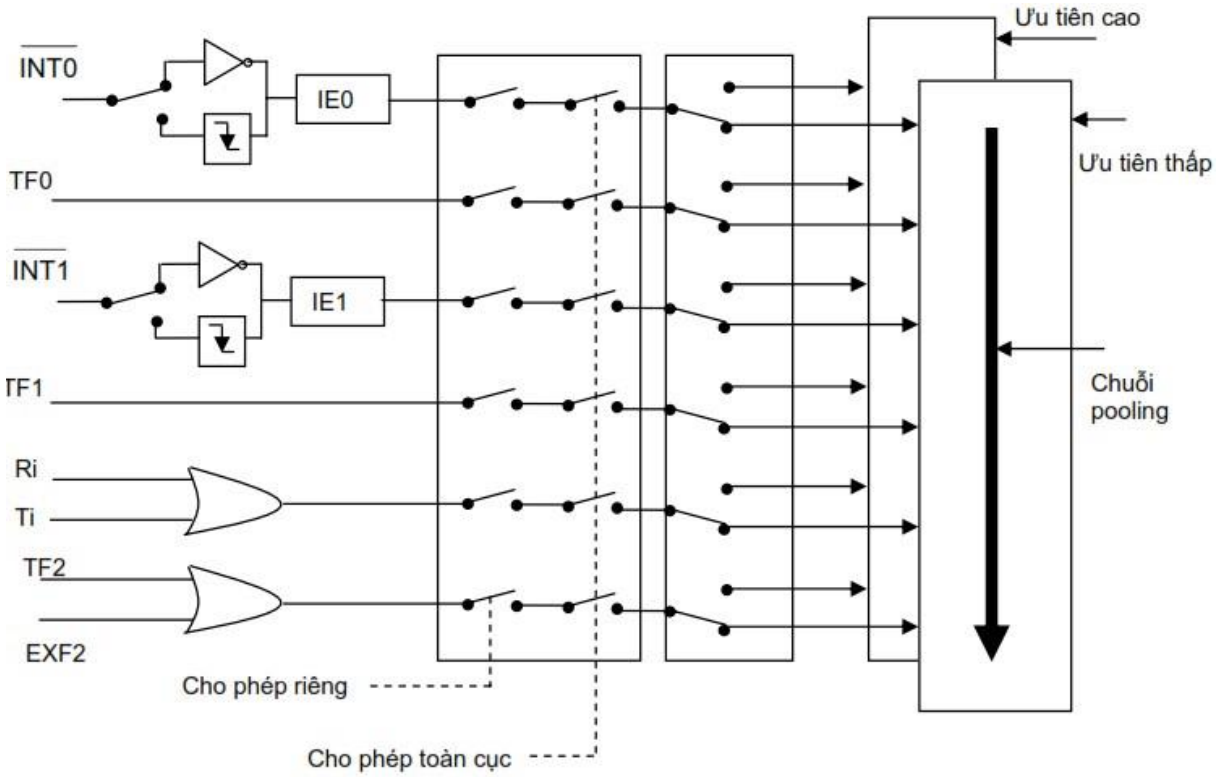
+ Nạp địa chỉ vectơ của ISR vào PC

+ Thực hiện ISR

- Chương trình ISR hoạt động và thực hiện các thao tác tương ứng với ngắt. Sau đó, kết thúc khi gặp lệnh RETI (return from interrupt), lệnh này lấy lại giá trị của PC từ ngăn xếp và phục hồi trạng thái ngắt cũ, chương trình tiếp tục chạy từ nơi tạm dừng.

BẢNG 6.3 Các bít cờ ngắt

Ngắt	Cờ	Thanh ghi SFR và vị trí bít
0 ngoài	IE0	TCON.1
1 ngoài	IE1	TCON.3
timer 1	TF1	TCON.7
timer 0	TF0	TCON.5
cổng nối tiếp	Ti	SCON.1
cổng nối tiếp	Ri	SCON.0
timer 2	TF2	T2CON.7 (8052)
timer 2	EXF2	T2CON.6 (8052)



Hình 6.2. Cấu trúc ngắt 8051

Các vec tơ ngắt

- Khi một ngắt được chấp nhận, giá trị nạp vào bộ đếm chương trình PC được gọi là véc tơ ngắt. Đây là địa chỉ bắt đầu của ISR đối với ngắt tương ứng. Các véc tơ ngắt được cho trong bảng 6.4.

- Véc tơ reset hệ thống (RST tại địa chỉ 0000H) cũng được cho trong bảng nên nó cũng giống như một ngắt, nó dừng chương trình chính và nạp vào PC một giá trị mới. Khi trỏ đến một ngắt, cờ gây ra ngắt sẽ bị xóa tự động bị xóa bởi phần cứng ngoại trừ Ri và Ti đối với ngắt cổng nối tiếp và TF2, EXF2 đối với ngắt timer 2. Do có hai nguyên nhân tạo ngắt cho các ngắt này nên thật là không thực tế nếu CPU xóa cờ ngắt. Các bít này phải được kiểm tra trong ISR để xác định nguyên nhân tạo ngắt và sau đó cờ ngắt được xóa bằng phần mềm, thường có một sự rẽ nhánh đến các thao tác tương ứng phụ thuộc vào nguồn tạo ra ngắt.

Vì các véc tơ ngắt được đặt phía dưới đáy của bộ nhớ chương trình nên lệnh đầu tiên của chương trình chính thường là lệnh nhảy qua vùng này ví dụ lệnh LJMP 0030H.

4. Thiết kế chương trình dùng ngắt

- Trong các bài trước đây đã không dùng đến ngắt mà dùng nhiều các vòng lặp để kiểm tra cờ tràn của timer TF0, TF1 hoặc TF2 hoặc các cờ thu phát của cổng nối tiếp Ti hoặc Ri, vấn đề của phương pháp này là thời gian hoạt động của CPU hoàn toàn được dùng vào việc chờ các cờ này được set, điều này hoàn toàn

không thích hợp với các ứng dụng hướng điều khiển mà trong đó yêu cầu vi điều khiển phải đồng thời tương tác với nhiều thiết bị vào ra.

- Một ví dụ trong phần này sẽ minh họa các phương pháp thực tế viết phần mềm cho các ứng dụng hướng điều khiển, thành phần chính là các ngắt. Mặc dù các ví dụ này không cần thiết lớn hơn nhưng chúng sẽ phức tạp hơn, điều này được nhận ra bằng cách tiến hành từng bước. Một số rắc rối xảy ra trong khi thiết kế hệ thống là do các ngắt.

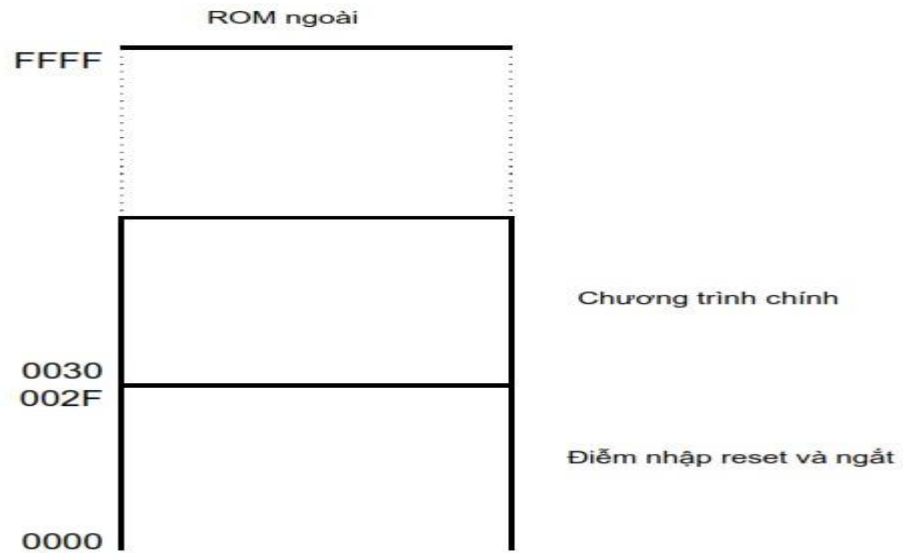
- Các chương trình ví dụ sau đây được bắt đầu tại địa chỉ 0000H và chương trình sẽ bắt đầu thực hiện khi reset hệ thống, các chương trnh này phát triển cho các ứng dụng thực tế và được lưu vào ROM hoặc EPROM. Khuôn mẫu đề nghị cho một chương trình có sử dụng ngắt như sau:

```
ORG 0000H ; reset điểm nhập
LJMP MAIN
.
.
.                ; các điểm nhập ISR
ORG 0030H ; điểm nhập chương trình chính
MAIN: …….. ; bắt đầu chương trình chính
```

- Lệnh đầu tiên nhảy đến địa chỉ 0030H ngay phía trên các vị trí của véc tơ ngắt và cũng là nơi bắt đầu các ISR như ở hnh 6.3, chương trnh chính bắ t đầu tại địa chỉ 0030H

BẢNG 6.4 Các véc tơ ngắt

Ngắt	Cờ	Địa chỉ véc tơ
Reset hệ thống	RST	0000H
0 bên ngoài	IE0	0003H
Timer 0	TF0	000BH
1 bên ngoài	IE1	0013H
Timer 1	TF1	001BH
Cổng nối tiếp	Ri hoặc Ti	0023H
Timer 2	TF2 hoặc EXF2	002BH



Hình 6.3. Tổ chức ngắt trong bộ nhớ

Chương trình phục vụ ngắt kích thước nhỏ

Các chương trnh phục vụ ngắt phải bắt đầu gần phía dưới đáy của bộ nhớ chương trình tại các địa chỉ cho trong bảng 6.4. Mặc dù giữa các điểm nhập chỉ có 8 byte nhưng thường cũng đủ để thực hiện các thao tác cần thiết và trở lại chương trình chính từ ISR

Nếu chỉ dùng một tín hiệu ngắt ví dụ timer 0 thì có thể áp dụng chương trình sau đây

```
        ORG 0000H ; reset    LJMP
        MAIN
        ORG 000BH ; điểm nhập timer 0  T0ISR:
…… ; bắt đầu ISR của timer 0    ……..
        ……
        RETI  ; trở về chương trình chính  MAIN:
        …….. ; chương trình chính    ……..
```

Nếu dùng nhiều ngắt thì phải cẩn thận để bảo đảm rằng chúng bắt đầu tại các vị trí đúng (bảng 6.4) và không đè lên các ISR kế tiếp vì trong ví dụ trên chỉ dùng có một ngắt nên chương trình chính có thể bắt đầu ngay phía dưới lệnh RETI

Chương trình phục vụ ngắt kích thước lớn

Nếu một ISR có kích thước nhiều hơn 8 byte thì có thể chuyển nó đến vị trí khác trong bộ nhớ chương trình hoặc cho lấn sang điểm nhập của ngắt kế tiếp. Điển hình là ISR bắt đầu bằng một lệnh nhảy đến nơi khác trong bộ nhớ chương trình và tại đó ISR có thể trải rộng ra. Hãy xem ví dụ sau chỉ dùng ngắt timer 0

```
ORG 0000H ; reset
        LJMP MAIN
```

```
        ORG 000BH ; điểm nhập timer 0    LJMP
        T0ISR
        ORG 0030H; ; trên vùng véc tơ ngắt
MAIN: ……
        ……
T0ISR: …….. ; ISR của timer 0    ……..
        ……..
        RETI  ; trở về chương trình chính
```

Để cho đơn giản các chương trình chỉ làm một việc tại thời điểm bắt đầu, chương trình chính sẽ khởi tạo timer, cổng nối tiếp và các thanh ghi ngắt tương ứng và sau đó không làm gì hết. Toàn bộ công việc được thực hiện trong ISR,

sau các dòng lệnh khởi tạo chương trình chính chỉ còn dùng lệnh sau đây HERE:

```
        SJMP HERE
```

Khi một ngắt xảy ra, chương trình chính tạm thời dừng lại trong khi ISR đang hoạt động, lệnh RETI ở cuối ISR trả quyền điều khiển về cho chương trình chính và chương trình lại tiếp tục không làm gì cả.

Điều này cũng không có gì lạ, trong nhiều ứng dụng hướng điều khiển phần lớn công việc thường được thực hiện trong chương trình phục vụ ngắt

Ví dụ 6-1: **Tạo xung vuông bằng ngắt timer**

Viết chương trình dùng timer 0 và các ngắt để tạo xung vuông 1 KHz tại chân P1.0

Các ngắt timer xảy ra khi các thanh ghi THx/TLx tràn và cờ TFx được set, chương trình như sau

```
        ORG 0  ; reset
        LJMP MAIN  ; nhảy qua vùng véc tơ ngắt
        ORG 000BH  ; véc tơ ngắt timer 0
T0ISR:  CPL P1.0  ; đảo port bít
        RETI
        ORG 0030H
MAIN:   MOV TMOD,#02H ; timer 0 mode 2  MOV
        TH0,#-50 ; delay 50 □s
        SETB TR0  ; khởi động timer
        MOV IE,#82H ; cho phép ngắt timer 0
        SJMP $  ; không làm gì cả
```

Đây là một chương trình đầy đủ có thể nạp vào EPROM và cài đặt vào UNIKIT để chạy thử. Ngay sau khi reset bộ đếm chương trình sẽ được nạp giá trị 0000H, lệnh đầu tiên được thực hiện là LJMP MAIN và chương trình nhảy qua ISR của timer đến địa chỉ 0030H trong bộ nhớ chương trình, ba lệnh tiếp theo khởi tạo timer 0 ở chế độ tự động nạp lại 8 bít và tràn sau mỗi 50μs . Lệnh

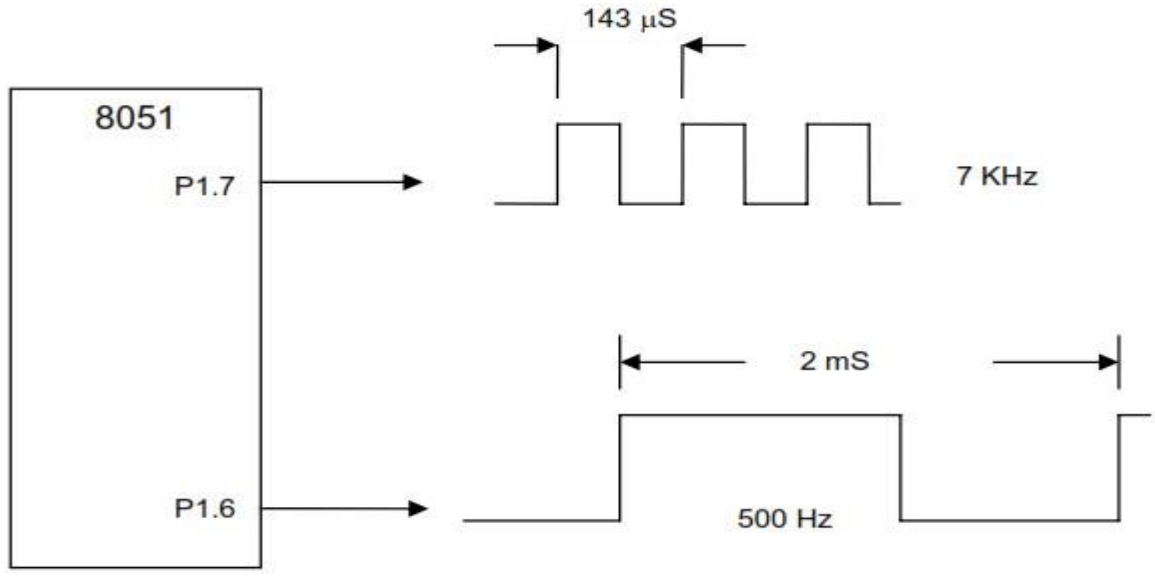
MOV IE,#82H cho phép ngắt timer 0 có nghĩa là mỗi lần tràn của timer sẽ tạo ra một ngắt.

Dĩ nhiên lần tràn đầu tiên sẽ không xảy ra sau 50μs do chương trình chính rơi vào vòng lặp "không làm gì cả". Cứ sau mỗi 50μs một ngắt sẽ xuất hiện, chương trình chính bị dừng và chương trình T0ISR hoạt động đảo trạng thái của port bít và sau đó trở về chương trình chính thực hiện tiếp vòng lặp "không làm gì cả" và chờ một ngắt khác sau 50μs tiếp theo. Lưu ý là cờ timer TF0 không bị xóa bằng phần mềm, khi các ngắt được cho phép TF0 sẽ bị xóa tự động bởi phần cứng ngay khi CPU trỏ đến ngắt. Địa chỉ trở về trong chương trình chính là vị trí của lệnh SJMP, địa chỉ này được CPU cất vào trong ngăn xếp trước khi trỏ đến véc tơ ngắt và được lấy lại từ ngăn xếp khi thực hiện lệnh RETI ở cuối chương trình ngắt. Do thanh ghi SP đã không được khởi tạo nên địa chỉ mặc định sau khi reset của ngăn xếp là 07H, lệnh PUSH sẽ cất địa chỉ trở về trong RAM nội tại 08H (PCL) và 09H (PCH)

Ví dụ 6-2: **Tạo hai xung bằng ngắt**

Viết chương trình sử dụng ngắt tạo đồng thời hai xung vuông 7 KHz và 500 Hz tại port P1.7 và P1.6

Cấu hình phần cứng và dạng sóng ra được trình bày ở hình 6.4



Hình 6.4. Dạng sóng

Việc kết hợp các ngõ ra là cực kỳ khó khăn đối với các hệ thống không được điều khiển bằng ngắt. Timer 0 hoạt động ở mode 2 cung cấp xung 7 KHz và timer 1 hoạt động ở mode 1 cung cấp xung 500 Hz. Vì xung 500 Hz có thời gian mức cao là 1 mS và mức thấp cũng là 1 mS nên không thể dùng mode 2, chương trình như sau ORG 0

```
        LJMP MAIN
        ORG 0BH  ; địa chỉ véc tơ timer 0    JMP
        T0ISR
        ORG 1BH  ; địa chỉ véc tơ timer 1
        LJMP T1ISR
        ORG 30H
MAIN:   MOV TMOD,#12H ; timer 1=mode 1; timer 0=mode 2
        MOV TH0,#-71 ; tạo xung 7 KHz    SETB
        TR0
        SETB TF1  ; tạo ngắt timer 1
        MOV IE,#8AH ; cho phép cả hai ngắt timer    SJMP
        $
T0ISR:  CPL P1.7
        RETI
T1ISR:  CLR TR1
        MOV TH1,#HIGH (-1000) ; 1 mS mức cao
        MOV TL1,#LOW (-1000) ; và 1 mS mức thấp
        SETB TR1
        CPL P1.6
        RETI
```

Chương trình chính và các ISR được đặt phía trên vùng dành cho các véc tơ ngắt và reset hệ thống, cả hai dạng sóng được tạo ra bởi lệnh CPL. Tuy nhiên, cách tạo thời gian trì hoãn thì có khác nhau.

Vì cặp thanh ghi TL1/TH1 phải được nạp lại sau mỗi lần tràn (có nghĩa là sau mỗi ngắt) nên ISR của timer 1 hoạt động như sau: (a) dừng timer, (b) nạp lại TL1/TH1, (c) khởi động timer và sau đó (d) đảo port bít. Một điểm lưu ý là TL1/TH1 không được khởi tạo tại vị trí bắt đầu của chương trình chính giống như TH0 vì TL1/TH1 phải được khởi tạo lại sau mỗi lần tràn. TF1 được set trong chương trình chính bằng phần mềm để tạo ra ngắt ban đầu ngay khi các ngắt được cho phép để bắt đầu dạng sóng 500 Hz.

ISR của timer 0 chỉ có nhiệm vụ đơn giản là đảo port bít và sau đó trở về chương trình chính, SJMP $ là dạng viết tắt của HERE: SJMP HERE

5. Ngắt cổng nối tiếp

Các ngắt của cổng nối tiếp xảy ra khi một trong hai cờ ngắt phát Ti hoặc cờ ngắt thu Ri được set. Ngắt phát xuất hiện khi quá trình phát của kí tự trước đó được viết vào SBUF hoàn tất, một ngắt thu xuất hiện khi một kí tự đã được nhận đầy đủ và đang chờ đọc trong SBUF

Các ngắt cổng nối tiếp khác với các ngắt của timer, cờ tạo ra ngắt cổng nối tiếp không được xóa bằng phần cứng khi CPU trỏ đến véc tơ ngắt lí do là có hai nguồn tạo ra một ngắt cổng nối tiếp đó là Ti hoặc Ri. Nguồn tạo ra ngắt phải được xác định trong ISR và cờ ngắt được xóa bằng phần mềm. Trở lại với các ngắt timer, cờ ngắt được xóa bằng phần cứng khi CPU trỏ đến ISR

Ví dụ 6-3: **Xuất kư tự bằng ngắt**

Viết chương trình dùng ngắt xuất liên tục bảng mã ASCII (bao gồm cả kí tự điều khiển) đến một thiết bị cuối nối với cổng nối tiếp của 8051

Có 128 mã 7 bít trong bảng mã ASCII bao gồm 95 mã đồ họa (20H đến 7EH) và 33 mã điều khiển (00H đến 1FH và 7FH) chương trình sau đây được thực hiện ngay sau khi reset hệ thống

```
        ORG 0
        LJMP MAIN
        ORG 0023H  ; điểm nhập ngắt cổng nối tiếp
        LJMP SPISR
        ORG 0030H
MAIN:   MOV TMOD,#20H ; timer 1 mode 2
        MOV TH1,#-26 ; nạp giá trị 1200 baud
        SETB TR1  ; khởi động timer
        MOV SCON,#42H ; mode 1, set Ti để tạo ngắt đầu tiên
                      ; gởi kí tự thứ nhất
        MOV A,#20H ; gởi kí tự trắng đầu tiên
        MOV IE,#90H ; cho phép ngắt cổng nối tiếp
        SJMP $  ; không làm gì cả
SPISR:  CJNE A,#7FH,SKIP
        MOV A,#20H
SKIP:   MOV SBUF, A
        INC A
        CLR Ti  ; xóa cờ ngắt
```

RETI

Sau khi nhảy đến MAIN tại địa chỉ 003H, ba lệnh đầu tiên khởi tạo timer 1 để cung cấp xung đồng hồ 1200 baud cho cổng nối tiếp. MOV SCON,#42H khởi tạo cổng nối tiếp hoạt động ở mode 1 (8 bít UART) và set Ti để gây ra một ngắt ngay khi các ngắt được cho phép. Sau đó mã ASCII đầu tiên (20H) được nạp vào A và cho phép các ngắt, cuối cùng phần thân của chương trình chính là vòng lặp "không làm gì cả" (SJMP $)

Chương trình phục vụ ngắt cổng nối tiếp làm tất cả công việc mỗi khi chương trình chính thiết lập các điều kiện ban đầu. Hai lệnh đầu tiên kiểm tra bộ tích lũy và nếu mã ASCII có giá trị là 7FH (mã cuối cùng được gởi đi là 7EH) thì bộ tích lũy được reset về 20H, sau đó mã ASCII được gởi đến bộ đệm cổng nối tiếp (MOV SBUF, A) mã được tăng lên (INC A), cờ ngắt phát bị xóa (CLR Ti) và kết thúc ISR (RETI), điều khiển được trả về chương trình chính và thực hiện tiếp lệnh SJMP $ cho đến khi Ti được set khi kết thúc quá trình xuất kí tự tiếp theo. Nếu so sánh tốc độ của CPU với tốc độ truyền kí tự sẽ thấy rằng SJMP $ được thực hiện với phần trăm rất lớn về thời gian trong chương trình, tỉ lệ phần trăm này là bao nhiêu ? Với tốc độ 1200 baud thì mỗi bít truyền đi cần 1/1200 = 0,833 mS, 8 bít dữ liệu cộng với 1 bít start và 1 bít stop cần 8,33 mS hoặc 8330μs , thời gian thực hiện dài nhất của SPISR là tổng các chu kỳ của từng lệnh nhân với 1μs (giả sử tần số thạch anh là 12 MHz), trong trường hợp này là 8μs . Như vậy, với mỗi 8333μs cần thiết để truyền một kí tự thì chỉ cần 8μs cho SPISR, lệnh SJMP thực hiện với tỉ lệ (8325/8333) x 100 =

99,90 % thời gian. Do dùng ngắt nên lệnh SJMP có thể được thay thế bằng các lệnh khác để thực hiện các công việc cần thiết khác của ứng dụng. Các ngắt vẫn xảy ra sau mỗi 8333μs và kí tự vẫn được xuất ra cổng nối tiếp như nêu trên.

6. Các ngắt ngoài

Các ngắt ngoài xảy ra tại mức thấp hoặc cạnh âm ở chân $\overline{INT0}$ hoặc $\overline{INT1}$ của 8051, đây là các chân đa năng của port 3: Bít P3.2 (chân 12) và bít P3.3 (chân 13).

Các cờ tạo ra các ngắt này là bít IE.0 và IE.1 trong thanh ghi TCON, cờ tạo ra ngắt bị xóa bởi phần cứng khi CPU trỏ đến ISR nếu ngắt là loại tác động cạnh, còn đối với ngắt tác động bằng mức thì nguồn tạo ngắt bên ngoài sẽ điều khiển mức của cờ ngắt.

Việc chọn lựa ngắt tác động mức thấp hoặc tác động cạnh âm được được lập trình thông qua bít IT0 và IT1 trong thanh ghi TCON. VD: Nếu IT1 = 0 thì

ngắt 1 ngoài được kích bởi mức thấp tại chân $\overline{INT1}$ và nếu IT1 = 1 thì ngắt này

được kích bằng cạnh âm. Trong chế độ này nếu các mẫu tại chân *INT*1 ở mức cao trong một chu kỳ và ở mức thấp trong các chu kỳ kế tiếp thì cờ IE1 trong TCON được set và sau đó cờ này sẽ yêu cầu một ngắt.

Vì các chân ngắt ngoài được lấy mẫu mỗi chu kỳ máy một lần nên ngõ vào này phải được duy trì ít nhất trong 12 chu kỳ dao động để bảo đảm việc lấy mẫu là thích hợp. Nếu là loại tác động cạnh thì nguồn ngoài phải giữ ở mức cao ít nhất một chu kỳ và ở mức thấp ít nhất một chu kỳ hoặc hơn để bảo đảm nhận ra được sự chuyển mức. IE0 và IE1 được xóa tự động khi CPU trỏ đến ngắt.

Nếu ngắt ngoài là loại tác động mức thì nguồn ngoài phải duy trì mức tác động cho đến khi ngắt yêu cầu thực sự được tạo ra. Sau đó phải trở về mức không tác động trước khi ISR hoàn tất hoặc trước khi một ngắt khác được tạo ra. Thông thường một thao tác trong ISR làm cho nguồn tạo ngắt trả tín hiệu ngắt trở về trạng thái không tác động.

Ví dụ: điều khiển lò sưởi

Sử dụng ngắt thiết kế chương trình điều khiển lò sưởi ổn định tại nhiệt độ $20^0C \pm 1^0C$.

- Giả sử rơ le tắt/mở lò sưởi được đặt P1.7

  P1.7 = 1(lò sưởi mở)

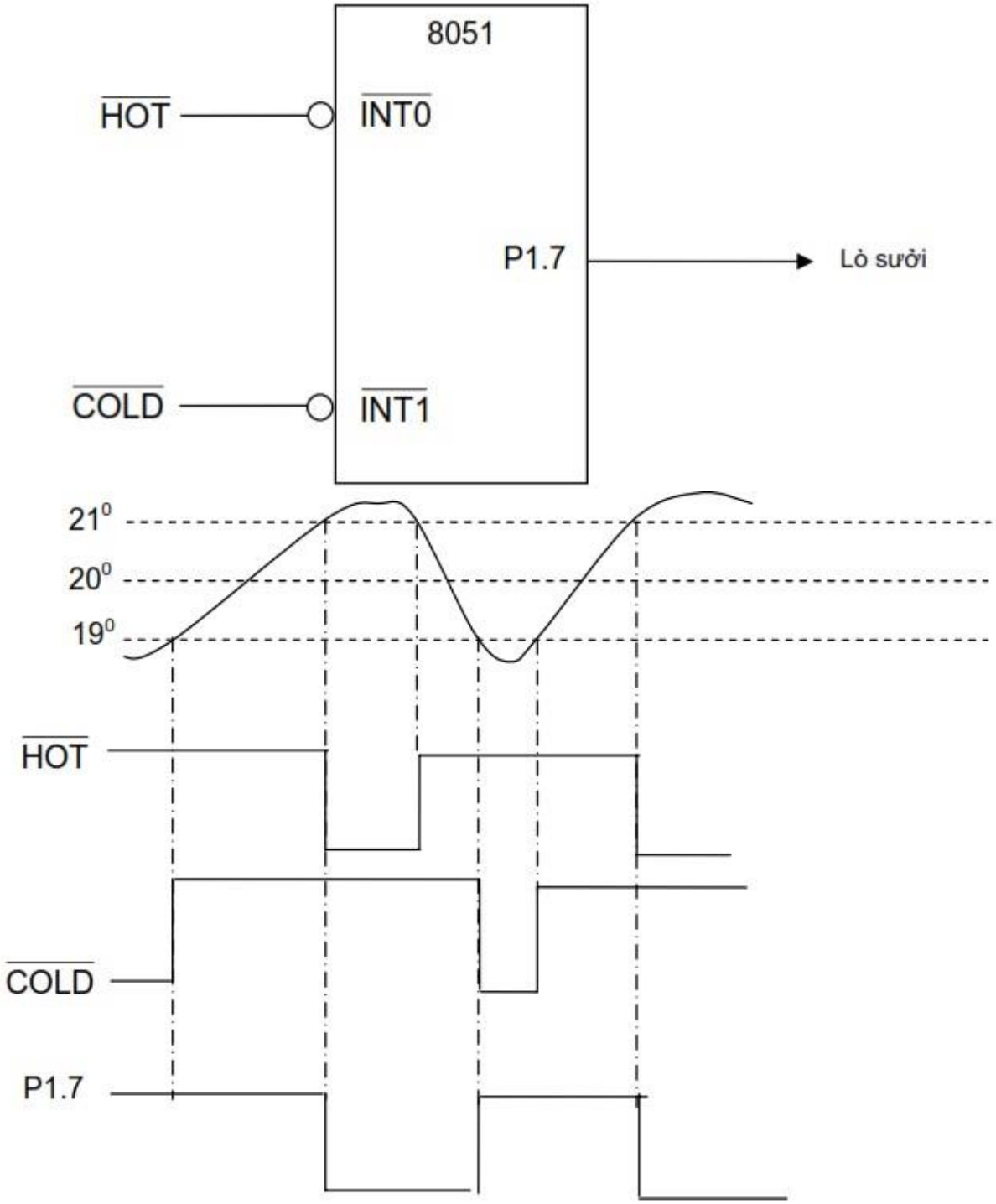
  P1.7 = 0(lò sưởi tắt)

- Các cảm biến nhiệt độ được nối đến $\overline{INT0}$ và $\overline{INT1}$ và tạo ra các tín hiệu $\overline{HOT}$ và $\overline{COLD}$ theo thứ tự

  $\overline{HOT}$ = 0 nếu $T > 21^0C$

  $\overline{COLD}$ = 0 nếu $T < 19^0C$

Lò sưởi được mở nếu $T < 19^0C$ và tắt khi $T > 21^0C$, cấu trúc phần cứng và đồ thị thời gian được trình bày ở hình 6.5



Hình 6.5. Sơ đồ lò sưởi và đồ thị thời gian

```
        ORG 0
        LJMP MAIN
EX0ISR: CLR P1.7  ; véc tơ ngắt 0 ngoài tại 0003H; lò sưởi tắt
        RETI
        ORG 13H  ; địa chỉ véc tơ 1 ngoài, lò sưởi mở
EX1ISR: SETB P1.7
        RETI
        ORG 30H
```

```
MAIN:   MOV IE,#85H ; cho phép các ngắt ngoài
        SETB IT0  ; tác động cạnh âm    SETB
        IT1
        SETB P1.7  ; mở lò sưởi
        JB P3.2,SKIP ; nếu T> 21    CLR
        P1.7  ; thì tắt lò sưởi SKIP:
        SJMP $  ; không làm gì cả
```

Ba dòng lệnh đầu tiên trong chương trình chính cho phép các ngắt ngoài và làm cho được lấy mẫu (JB P3.2,SKIP) nếu ngõ vào này ở mức cao, nhiệt độ chưa cao hơn 21cả hai *INT*0 và *INT*1 trở thành loại tác động cạnh âm. Vì trạng thái hiện tại của các ngõ vào *HOT* (P3.2) và *COLD* (P3.1) không biết trước nên cần ba dòng lệnh tiếp theo để tắt hoặc mở lò sưởi tùy theo nhiệt độ. Trước tiên, lò được mở (SETB P1.7) và ngõ vào *HOT* được lấy mẫu (JB P3.2, SKIP) nếu ngõ vào này ở mức cao, nhiệt độ chưa cao hơn 21^0C nên lệnh kế tiếp bị bỏ qua và là vẫn tiếp tục được mở.

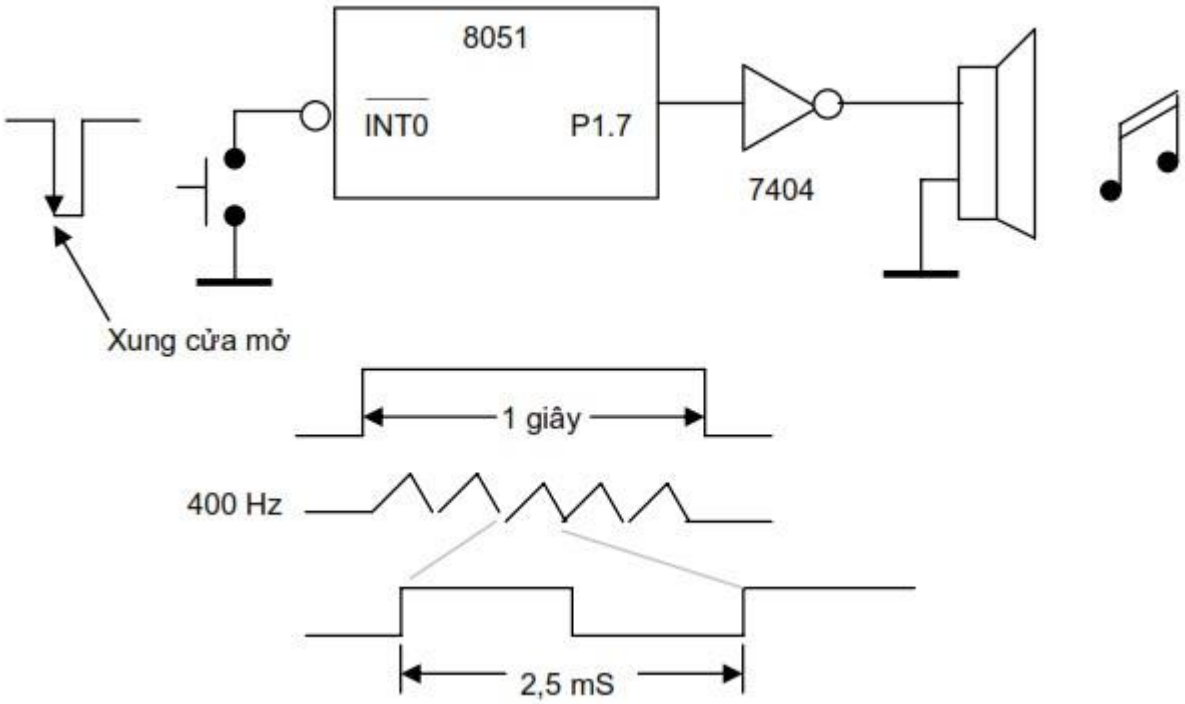
Ngược lai, nếu ngõ vào *HOT* ở mức thấp (T> 21^0C) thì lệnh kế tiếp CLR P1.7 được thực hiện để tắt lò trước khi đi vào vòng lặp "không làm gì cả".

Lưu ý là chỉ dẫn ORG 0003H không cần thiết phải hiện diện ngay trên nhãn EX0ISR do lệnh LJMP MAIN dài 3 byte nên EX0ISR chắc chắn được bắt đầu tại địa chỉ 0003H, điểm nhập của ngắt 0 ngoài.

Ví dụ 6-5: **Hệ thống báo động**

Dùng ngắt để thiết kế một hệ thống báo động tạo ra âm thanh 400 Hz trong

1 giây (nhờ 1 loa nối vào chân P1.7) mỗi khi cảm biến đặt ở cửa (nối đến chân



*INT*0) tạo ra một sườn xuống.

Hình 6.6. Điều khiển loa bằng ngắt

Hướng giải quyết là dùng 3 ngắt: Ngắt 0 ngoài (cảm biến cửa), ngắt timer 0 (âm thanh 400 Hz) và ngắt timer 1 (định thời 1 giây). Sơ đồ mạch và đồ thị thời gian trình bày ở hình 6.6

```
        ORG 0
        LJMP MAIN
        LJMP EX0ISR
        ORG 000BH  ; véc tơ timer 0
        LJMP T0ISR
        ORG 001BH  ; véc tơ timer 1
        LJMP T1ISR
        ORG 0030H
MAIN:   SETB IT0  ; tác động cạnh âm
        MOV TMOD,#11H ; chế độ định thời 16 bít
        MOV IE,#81H ; cho phép EX0
        SJMP $
```

```
EX0ISR:  MOV R7,#20  ; 20x5000□s = 1s
         SETB TF0  ; tạo ngắt timer 0
         SETB TF1  ; tạo ngắt timer 1
         SETB ET0  ; tạo âm thanh trong 1s
         SETB ET1
         RETI
T0ISR:   CLR TR0
         DJNZ R7,SKIP ; nếu chưa đủ 20 lần, thoát
         CLR ET0  ; nếu đủ, kết thúc âm thanh
         CLR ET1
         LJMP EXIT
SKIP:    MOV TH0,#HIGH(-50000) ; trì hoãn 0,05s
         MOV TL0,#LOW(-50000)
         SETB TR0
EXIT:    RETI
T1ISR:   CLR TR1
         MOV TH1,#HIGH(1250)
         MOV Tl1,#LOW(-1250)
         CPL P1.7
         SETB TR1
         RETI
         END
```

Chương trình trên gồm 5 phần phân biệt: Vị trí các véc tơ ngắt, chương trình chính và 3 chương trình phục vụ ngắt. Các vị trí véc tơ ngắt chứa các lệnh LJMP để chuyển điều khiển đến các ISR tương ứng. Chương trình chính bắt đầu tại địa chi3 0030H chỉ gồm 4 lệnh. Lệnh SETB IT0 cho phép ngõ vào ngắt nối với cảm biến cửa được kích bởi cạnh âm.

Lệnh MOV TMOD,#11H xác định chế độ hoạt động của cả hai timer là chế độ định thời 16 bít, chỉ có ngắt 0 ngoài được bắt đầu (MOV IE,#81H) khi cửa mở là điều kiện cần phải có trước khi một ngắt nào đó được chấp nhận.

Lệnh cuối cùng SJMP $ đưa chương trình chính vào vòng lặp "không làm gì cả" khi trạng thái mở cửa được phát hiện (cạnh âm tại $\overline{INT0}$) thì ngắt 0 ngoài sẽ được tạo ra.

Chương trình phục vụ ngắt EX0ISR bắt đầu bằng việc nạp hằng số 20 cho R7 rồi set cờ tràn của cả hai timer để buộc các ngắt định thời xuất hiện. Tuy nhiên,

các ngắt timer chỉ xuất hiện khi các bít tương ứng trong thanh ghi IE cho phép. Hai lệnh kế tiếp SETB ET0 và SETB ET1 cho phép các ngắt bộ định thời, cuối cung EX0ISR trở về chương trình chính bằng lệnh RETI.

Timer tạo ra khoảng thời gian trì hoãn 1s và timer 1 tạo ra âm thanh 400 Hz. Sau khi chương trình EX0ISR kết thúc, các ngắt timer được lập tức tạo ra và được thực hiện sau khi thực hiện 1 lệnh SJMP $. Do tác dụng của chuỗi pooling nên ngắt của timer 0 được phục vụ trước tiên. Khoảng thời gian 1s được tạo ra bằng cách lập trình để lặp lại 20 lần khoảng thời gian định thời 50000μs , thanh ghi R7 hoạt động như một bộ đếm. Chương trình phục vụ ngắt

T0ISR hoạt động như sau: Trước tiên, timer 0 được điều khiển dừng và thanh ghi R7 bị giảm 1, tiếp theo TH0/TL0 được nạp lại giá trị -50000, timer 0 chạy trở lại và ngắt được kết thúc. Ở lần ngắt thứ 20, R7 được giảm xuống 0 (đã trôi qua 1s), các ngắt của cả hai timer bị ngăn (CLR ET0, CLR ET1) và ngắt kết thúc. Không còn ngắt do bộ định thời tạo ra nữa cho đến khi phát hiện cửa mở một lần nữa. Âm thanh 400 Hz được lập trình bằng cách sử dụng ngắt timer 1, tần số 400 Hz tương đương chu kỳ là 2500μs với 1250μs mức cao và 1250μs mức thấp. Chương trình T1ISR chỉ đơn giản nạp -1250 cho TH1/TL1, đảo port bít để điều khiển loa và kết thúc.

7. Đồ thị thời gian của ngắt

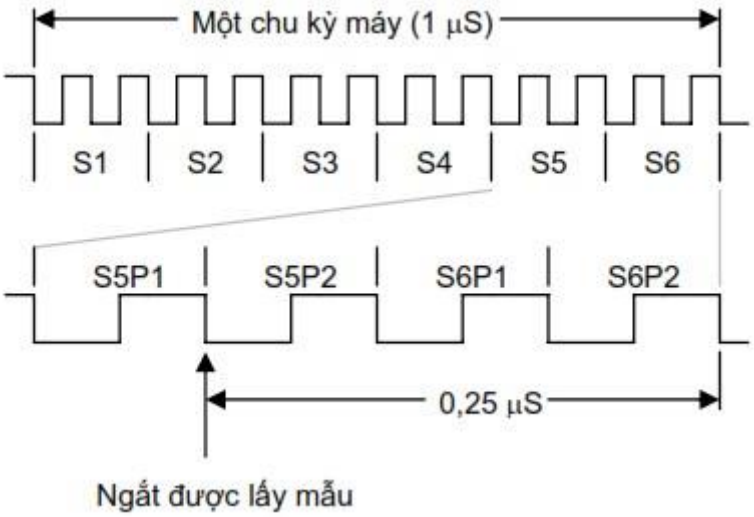
Các ngắt được lấy mẫu và được chốt ở S5P2 của mỗi chu kỳ máy (hình 6.7), chúng được xoay vòng đến chu kỳ máy kế tiếp và nếu có một điều kiện ngắt tồn tại, ngắt được chấp nhận nếu:

a) Không có ngắt nào khác có ưu tiên bằng hay cao hơn đang xảy ra.

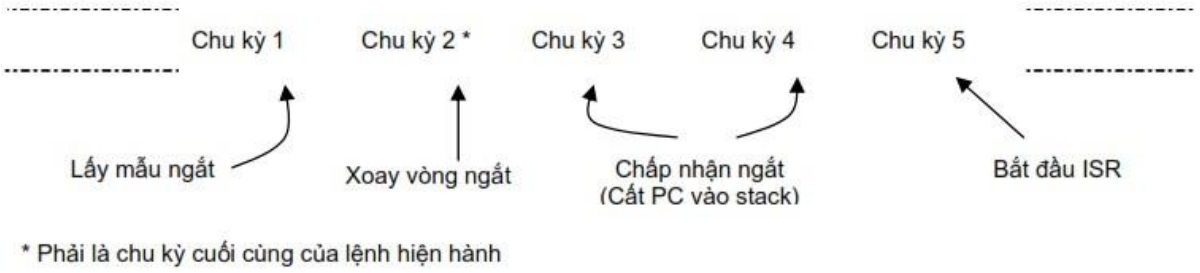
b) Chu kỳ xoay vòng là chu kỳ cuối của một lệnh.

c) Lệnh hiện hành không phải là lệnh RETI hoặc lệnh truy xuất đến thanh ghi IE hoặc IP. Trong suốt hai chu kỳ kế tiếp, CPU cất nội dung của PC vào ngăn xếp và nạp cho PC địa chỉ véc tơ ngắt, chương trình phục vụ ngắt được bắt đầu

Điều kiện lệnh hiện hành không phải là lệnh RETI nhằm bảo đảm rằng có ít nhất một lệnh được thực hiện sau mỗi một ISR



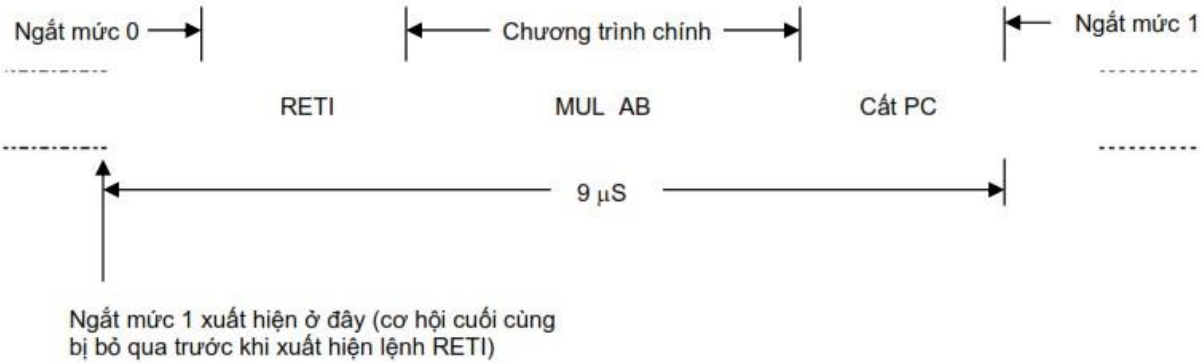
Hình 6.7. Lấy mẫu các ngắt tại S5P2



Hình 6.8. Quá trình thực hiện ngắt

Thời gian từ lúc có một điều kiện ngắt xuất hiện đến khi ISR bắt đầu được gọi là interrupt latency và rất quan trọng trong một số các ứng dụng điều khiển.

Với thạch anh 12 MHz, interrupt latency có thể ngắn khoảng 3,25□*s* đối với 8051. Một hệ thống dùng 8051 với ngắt ưu tiên cao sẽ có interrupt latency xấu nhất là 9,25□*s* (giả sử ngắt ưu tiên cao luôn luôn được phép). Điều này xảy ra nếu điều kiện ngắt xuất hiện ngay trước khi có lệnh RETI của ISR mưc 0 được theo sau bởi một lệnh nhân (hình 6.9)



Hình 6.9. Interrupt latency

BÀI 7: PHẦN MỀM HỢP NGỮ

1. Mở đầu

Hợp ngữ là ngôn ngữ máy tính có vị trí giữa ngôn ngữ cấp thấp là ngôn ngữ máy và các ngôn ngữ cấp cao như Pascal, C... Các ngôn ngữ ngày sử dụng từ và các phát biểu rất dễ hiểu đối với con người những vẫn còn một khoảng cách khá xa với ngôn ngữ tự nhiên trong cuộc sống, ngôn ngữ máy là ngôn ngữ nhị phân của máy tính. Một chương tfnh ngôn ngữ máy là một chuỗi các byte nhị phân biểu diễn các lệnh mà máy tính có thể thực hiện. Hợp ngữ thay thế các mã nhị phân của ngôn ngữ máy bằng các dạng gợi nhớ (mnemonics) tạo thuận lợi cho việc lập trình. Vd: một lệnh cộng ngôn ngữ máy có mã nhị phân là 10110011 được thay bằng lệnh gợi nhớ là ADD, việc lập trình bằng dạng gợi nhớ đuợc ưa chuộng hơn bằng mã nhị phân. Dĩ nhiên điều này không phải là toàn bộ câu chuyện. Các lệnh thao tác với dữ liệu và vị trí của dữ liệu thì được xác định bởi cách định địa chỉ khác nhau được nhúng vào trong mã nhị phân của lệnh hợp ngữ, do đó có thể có nhiều kiểu lệnh ADD phụ thuộc vào cái gì được cộng. Quy luật xác định các kiểu lệnh này được tập trung ở chủ đề lập trình bằng hợp ngữ. Một chương trình hợp ngữ không thể thực hiện được bởi máy tính, mỗi khi viết xong chương trình này phải được dịch ra ngôn ngữ máy. Trong ví dụ ở trên lệnh gợi nhớ ADD phải được dịch ra mã nhị phân 10110011. Tùy theo mức độ phức tạp của môi trường lập trình, quá trình dịch này có thể gồm một hoặc nhiều bước trước khi cho ra kết quả một chương trình mã máy có thể thực hiện được. Ít nhất phải cần đến trình dịch hợp ngữ để dịch các lệnh gợi nhớ thành mã nhị phân của ngôn ngữ máy, bước tiếp theo có thể phải cần đến một chương trình liên kết để kết nối các phần của chương trình từ các file khác nhau và đặt vào vị trí thích hợp trong bộ nhớ để chương trình có thể chạy được. Chương trình viết bằng hợp ngữ thường sử dụng các nhãn, dạng gợi nhớ... trong đó mỗi câu lệnh tương ứng với một mã máy, chương trình hợp ngữ được goị là mã nguồn hoặc mã kí hiệu và không thể chạy được bằng máy tính. Chương trình ngôn ngữ máy bao gồm các mã nhị phân tượng trưng cho các lệnh điều khiển máy tính, chương trình này được gọi là mã đối tượng và thực hiện được bằng máy tính. Trình dịch hợp ngữ là chương trình dùng để dịch chương trình viết bằng hợp ngữ (nhãn, gợi nhớ...) sang chương trình ngôn ngữ máy (mã nhị phân, địa chỉ, dữ liệu...) Chương trình ngôn ngữ máy (mã đối tượng) có thể ở dạng tuyệt đối hoặc dạng có thể định vị lại. Trong trường hợp sau phải cần đến một sự kết nối để xác lập địa chỉ tuyệt đối cho việc chạy chương trình. Trình liên kết là chương trình dùng để kết nối các chương trình đối tượng có thể định vị lại (modul) và tạo ra một chương trình đối tượng tuyệt đối thực hiện được bằng máy tính, trình liên kết thường được gọi là trình liên kết / định vị bao hàm 2 chức năng: kết nối các modul chương trình (link) và thiết lập địa chỉ để chạy chương trình (locate). Một đoạn (segment) là một đơn vị bộ nhớ mã lệnh hoặc dữ liệu, một đoạn có thể là tuyệt đối hoặc định vị lại. Một đoạn định vị lại có một tên, kiểu và các

thuộc tính khác cho phép trình liên kết nối nó với các đoạn thành phần khác nếu cần và để định vị đoạn một cách đúng đắn. Một đoạn tuyệt đối thì không có tên và không thể kết nối với các đoạn khác. Một modul chương trình bao gồm một hoặc nhiều đoạn thành phần. Tên modul được người dùng đặt. Các định nghĩa modul xác định phạm vi của các kí hiệu nội bộ. Một file đối tượng gồm một hoặc nhiều modul, trong nhiều trường hợp một modul được xem như là một file. Một chương trình bao gồm một modul tuyệt đối đó là sự kết hợp tất cả các đoạn tuyệt đối và các định vị lại từ tất cả các modul ngõ vào. Một chương trình chỉ chứa các lệnh mã nhị phân (với địa chỉ và dữ liệu) và máy tính hiểu được chương trình này.

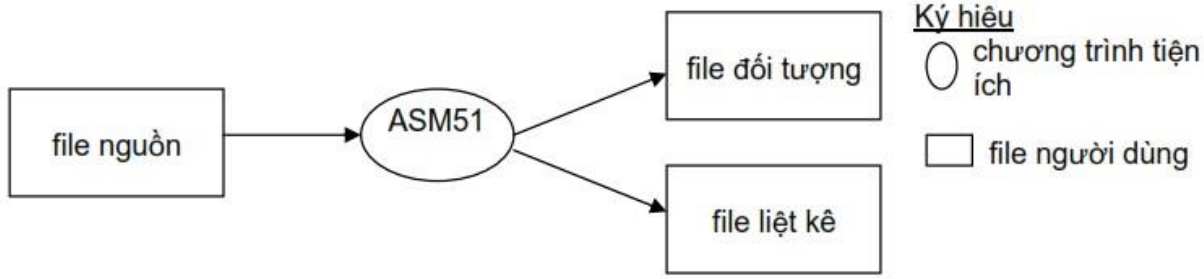
2. Hoạt động của Assembler

Có nhiều chương trình Assembler và các chương trình hỗ trợ khác phục vụ quá trình phát triển các ứng dụng dùng vi điều khiển, ASM51 là chương trình dịch hợp ngữ chuẩn của Intel đối với họ MCS-51, mặc dù có nhiều tính năng đã được chuẩn hóa nhưng cũng có một số tính năng không có trong trình dịch hợp ngữ của các công ty khác. ASM51 là một trình dịch mạnh, nó chạy được trên các hệ thống Intel và cả trên họ máy tính IBM PC, do CPU của máy tính này không phải là 8051 nên ASM51 được gọi là phần mềm hợp ngữ chéo. Chương trình nguồn của 8051 có thể được viết trên máy tính (dùng một trình xử lí văn bản bất kỳ) và được dịch ra file đối tượng và file liệt kê (listing file) bằng ASM51 nhưng không chạy trực tiếp được bằng máy tính cá nhân vì CPU của máy không phải là 8051 nên máy sẽ không hiểu mã lệnh trong file đối tượng, để thực hiện chương trình này trên máy tính phải có mô hình phần cứng hoặc phần mềm mô phỏng 8051, khả năng thứ ba là nạp file đối tượng vào hệ thống 8051 để thực hiện.

ASM51 được gọi từ dấu nhắc hệ thống bằng lệnh sau:

ASM51 source_file (assembler_controls)

File nguồn (source_file) được hợp dịch và các điều khiển hợp dịch xác định hiệu ứng (các điều khiển hợp dịch là tùy chọn).



Hình 7.1. Kết quả hợp dịch file nguồn

Trình dịch hợp ngữ xem file nguồn như là đầu vào và tạo ra file đối tượng và file liệt kê là kết quả đầu ra như mô tả ơ hình 7.1. Hầu hết các trình dịch hợp ngữ khi dịch sang mã máy tính đều quét file nguồn hai lần nên chúng được gọi là trình

dịch 2 bước. Trình dịch hợp ngữ dùng một bộ đếm vị trí biểu diễn địa chỉ của các lệnh và các nhãn, hoạt động của từng bước được mô tả như sau:

Bước 1 Trong khoảng thời gian của bước 1, file nguồn được quét từng dòng lệnh một và một bảng kí hiệu được thành lập. Giá trị mặc định của bộ đếm vị trí bằng 0 hoặc được xác định bởi chỉ dẫn ORG (set origin). Khi quét file nguồn nội dung bộ đếm vị trí sẽ tăng lên theo độ dài của mỗi lệnh. Các chỉ dẫn định nghĩa dữ liệu (DB hoặc DW) tăng bộ đếm vị trí theo số byte dành riêng. Mỗi khi tìm thấy một nhãn tại vị trí bắt đầu của dòng lệnh, nó sẽ được đặt vào trong bảng kí hiệu tương ứng với giá trị hiện hành của bộ đếm vị trí. Các kí hiệu được định nghĩa bằng chỉ dẫn EQU (equate) được đặt vào trong bảng tương ứng với giá trị sau EQU. Bảng kí hiệu được lưu trữ và được dùng trong bước 2.

Bước 2. File đối tượng và file liệt kê được tạo ra trong bước 2. Các lệnh dạng gợi nhớ được chuyển thành mã máy và đưa vào các file xuất. Các toán hạng được tính giá trị và đặt sau mã máy. Nơi các kí hiệu xuất hiện trong vùng toán hạng, giá trị của chúng được lấy lại từ bảng kí hiệu (được tạo ra trong bước 1) và được dùng để tính dữ liệu hoặc địa chỉ của lệnh. Do thực hiện 2 bước nên chương trình nguồn có thể dùng các tham chiếu thuận có nghĩa là có thể dùng 1 kí hiệu trước khi nó được định nghĩa, điều này thường xảy ra với những lệnh nhảy về phía trước. File đối tượng tuyệt đối chỉ chứa các byte mã máy nhị phân (00H-FFH) trong khi các file đối tượng định vị lại còn chứa bảng kí hiệu và các thông tin cần thiết cho quá trình liên kết và định vị lại. File liệt kê chứa mã văn bản ASCII (20H – 7EH) cho cả 2 chương trình nguồn và các byte hexa trong chương trình ngôn ngữ máy. File đối tượng là file liệt kê được phân biệt bằng cách cho hiển thị từng file trên màn hình máy vi tính bằng lệnh TYPE của MSDOS, file liệt kê sẽ hiển thị một cách rõ ràng từng dòng bao gồm địa chỉ, mã lệnh và có thể là dữ liệu tiếp theo sau là phát biểu chương trình từ file nguồn; trong khi đó file đối tượng sẽ hiển thị những kí hiệu kỳ lạ do nó chỉ chứa mã nhị phân của 8051.

3. Cấu trúc chương trình dữ liệu

Một chương trình hợp ngữ bao gồm các thành phần sau:

+ Các lệnh máy (Machine instructions)
+ Các chỉ dẫn hợp dịch (Assembler directives)
+ Các điều khiển hợp dịch (Assembler controls)
+ Các chú thích (Comments)

Lệnh máy là mã dạng gợi nhớ của các lệnh thực hiện được. Vd: ANL, chỉ dẫn hợp dịch là các lệnh của phần mềm assembler dùng để xác định cấu trúc của chương trình, các kí hiệu, dữ liệu, hằng số .... Vd: ORG, các điều khiển hợp dịch xác định chế độ cho assembler và dòng hợp dịch trực tiếp, vd: $ TITLE, các ghi chú hỗ trợ cho việc đọc chương trình được dễ hiểu hơn qua việc giải thích mục

đích và hoạt động của các dòng lệnh. Mỗi dòng lệnh hợp ngữ được chia thành nhiều vùng cách nhau bằng các khoảng trắng hoặc các kí tự tab, dạng thức tổng quát như sau:

[label:] mnemonic [operand] [, operand] [...] [; comment]

Trong đó chỉ có vùng mã gợi nhớ (mnemonic) là bắt buộc phải có, nhiều phần mềm hợp ngữ cần có vùng nhãn, nếu có thì vùng này phải bắt đầu ở bên trái trong cột 1 và các vùng tiếp theo phải cách nhau bằng kí tự space hoặc tab. Đối với ASM51 vùng nhãn không cần phải bắt đầu trong cột 1 và vùng mã gợi nhớ cũng không cần ở cùng 1 dòng với vùng nhãn, nhưng phải ở cùng dòng với vùng toán dạng. Các vùng được mô tả như sau:

Vùng nhãn Một nhãn đại diện cho địa chỉ của lệnh (hoặc dữ liệu) theo sau, khi cần nhảy đến một lệnh thì nhãn của lệnh được dùng trong vùng toán hạng của lệnh nhảy. Ví dụ: SJMP SKIP Thuật ngữ "nhãn" luôn được dùng để biểu diễn một địa chỉ còn thuật ngữ "kí hiệu" thì tổng quát hơn, nhãn là 1 loại kí hiệu và được kết thúc bằng dấu hai chấm (:). Các kí hiệu được gán giá trị hoặc thuộc tính thông qua các chỉ dẫn như: EQU, SEGMENT, BIT, DATA... Các kí hiệu có thể là địa chỉ, dữ liệu, tên các đoạn hoặc các cấu trúc khác do người lập trình đặt ra. Các kí hiệu không kết thúc bằng dấu 2 chấm, trong ví dụ sau đây PAR là kí hiệu và START là 1 nhãn.

```
PAR EQU 500 ;  "PAR" là kí hiệu biểu diễn giá trị 500
START  :  MOV  A, # OFFH ; "START" là 1 nhãn biểu diễn địa chỉ của lệnh MOV
```

Kí hiệu hoặc nhãn phải bắt đầu bằng một kí tự, dấu hỏi hoặc dấu gạch ngang dưới và theo sau có thể là kí tự, kí số, dấu hỏi, gạch dưới và độ dài tối đa là 31 kí tự, không phân biệt chữ in hay chữ thường nhưng không được trùng với các từ dành riêng (mã gợi nhớ, các toán tử, các chỉ dẫn hoặc các kí hiệu đã định nghĩa trước).

Vùng gợi nhớ

Các lệnh gợi nhớ hoặc các chỉ dẫn hợp dịch được đặt trong vùng gợi nhớ theo sau vùng nhãn, các lệnh gợi nhớ như: ADD, MOV, DIV hoặc INC, các chỉ dẫn như: ORG, EQU hoặc DB.

Vùng toán hạng Theo sau vùng gợi nhớ là vùng toán hạng, vùng này chứa địa chỉ hoặc dữ liệu được sử dụng bởi lệnh. Một nhãn có thể dùng để điều khiển địa chỉ của dữ liệu hoặc một kí hiệu được dùng để biểu diễn cho dữ liệu, vùng toán hạng phụ thuộc theo toán tử. Một vài toán tử không có toán hạng, Vd: lệnh RET trong khi các lệnh khác lại cần nhiều toán hạng cách nhau bởi dấu phẩy.

Vùng ghi chú Là vùng cuối cùng của một dòng lệnh hợp ngữ nhằm mục đích làm rõ nghĩa của chương trình, các chi chủ phải bắt đầu bằng dấu chấm phẩy, ghi chú có thể chứa cả 1 dòng nếu nó được bắt đầu bằng dấu phẩy. Các chương trình con hoặc 1 đoạn chương trình lớn thường được bắt đầu bằng 1 khối ghi chú gồm nhiều dòng ghi chú nhằm giải thích mục đích của đoạn chương trình theo sau.

Các kí hiệu của hợp ngữ đặc biệt Các kí hiệu đặc biệt được dùng cho chế độ địa chỉ thanh ghi, các kí hiệu này bao gồm A, R0 đến R7, DPTR, PC, C và AB, dấu $ có thể được dùng để thay cho giá trị hiện tại của bộ đếm vị trí, như ví dụ sau:

```
SETB C
INC DPTR
JNB Ti, $
```

Lệnh cuối cùng dùng bộ đếm vị trí của ASM51 để tránh việc phải dùng một nhãn, việc này có thể viết như sau: HERE: JNB Ti, HERE

Địa chỉ gián tiếp Đối với một số lệnh, vùng toán hạng có thể là một thanh ghi mà nội dung của nó là địa chỉ của dữ liệu, dấu @ cho biết đó là địa chỉ gián tiếp và các thanh ghi sử dụng chỉ có thể là R0 và R1, DPTR hoặc PC tùy theo lệnh. Ví dụ:

```
ADD A, @ R0
MOVC A, @ A + PC
```

Lệnh đầu tiên lấy một byte dữ liệu trong RAM nội tại địa chỉ là nội dung của R0, lệnh thứ hai lấy một byte dữ liệu từ bộ nhớ chương trình bên ngoài tại địa chỉ là tổng giữa nội dung của A với bộ đếm chương trình. Lưu ý là giá trị của bộ đếm chương trình khi cổng là địa chỉ của lệnh sau lệnh MOVC, trong hai lệnh trên giá trị lấy từ bộ nhớ sẽ được nạp vào thanh ghi A.

Dữ liệu tức thời

Các lệnh dùng cách địa chỉ tức thời sẽ cung cấp dữ liệu trong vùng toán hạng và là một thành phần của lệnh, đứng trước dữ liệu tức thời là dấu #. Ví dụ

```
CONSTANT    EQU 100
            MOV A, #0FEH
            ORL 40H, #CONSTANT
```

Tất cả các thao tác với dữ liệu tức thời (ngoại trừ MOV DPTR, #data) đều dùng dữ liệu 8 bit. Dữ liệu tức thời được tính toán như một hằng số 16 bit và sau đó byte thấp được sử dụng, tất cả các bít của byte phải giống nhau (00H hoặc FFH) hoặc một thông báo bởi "value will not fit in a byte" sẽ xuất hiện. Ví dụ:

Cú pháp các lệnh sau đây là đúng. MOV

```
A, # 0FF00H
```

MOV A, # 00FFH

MOV A, # 0FE00H

MOV A, # 01FFH

Nếu dùng số thập phân có dấu thí các giá trị từ -256 đến +256 có thể được sử dụng. Ví dụ hai lệnh sau đây là tương đương và hợp lệ.

MOV A, # -256

MOV A, # 0FF00H

Địa chỉ dữ liệu

Có nhiều lệnh truy xuất bộ nhớ bằng cách dùng địa chỉ trực tiếp và cần một địa chỉ RAM nội (00H đến 7FH) hoặc một địa chỉ SFR (80H đến 0FFH) đặt trong vùng toán hạng. Các kí hiệu định nghĩa sẵn có thể được dùng để biểu thị địa chỉ SFR. Ví dụ:

MOV A, 45 H

MOV A, SBUF

Địa chỉ bít

Một trong những đặc điểm của 8051 là khả năng truy xuất bít mà không cần đến kỹ thuật mặt nạ trên byte, các lệnh này đưa ra địa chỉ bít trong RAM nội (00H đến 7FH) hoặc một địa chỉ bít trong vùng SFR (80H đến 0FFH). Có 3 phương pháp để xác định một địa chỉ bít trong một lệnh: (a) địa chỉ được cho biết một cách rõ ràng (b) dùng dấu chấm giữa địa chỉ byte với vị trí bít và (c) dùng các kí hiệu có sẵn của hợp ngữ. Ví dụ:

SETB 0E7H ; địa chỉ rõ ràng

SETB ACC.7 ; dấu chấm

JNB Ti, $; Ti là kí hiệu có sẵn

JNB 99H, $; giống như trên

Địa chỉ của lệnh

Địa chỉ của lệnh được đặt trong vùng toán hạng của các lệnh nhảy bao gồm lệnh nhảy tương đối (SJMP và các lệnh nhảy có điều kiện) các lệnh nhảy và lệnh gọi tuyệt đối (ACALL, AJMP) và các lệnh gọi là lệnh nhảy dài (LJMP, LCALL). Địa chỉ lệnh thường được biểu diễn bằng một nhãn, ví dụ:

HERE: …. SJMP

HERE

ASM51 sẽ xác định giá trị đúng của địa chỉ và đưa vào trong lệnh một offset 8 bít có dấu, địa chỉ trang 11 bít hoặc địa chỉ dài 16 bít tương ứng. ***Lệnh nhảy và lệnh gọi tổng quát***

Phần mềm ASM51 cho phép người lập trình sử dụng dạng lệnh tổng quát JMP và CALL. Trong đó "JMP" thay cho SJMP, AJMP hoặc LJMP và "CALL" thay cho ACALL hoặc LCALL.

Trình hợp dịch sẽ biến đổi dạng tổng quát thành các lệnh thích hợp theo một số quy luật đơn giản: Sang dạng địa chỉ gần (chỉ đối với JMP) nếu không có tham chiếu thuận và đích nhảy trong vòng -128 đến +127 vị trí hoặc sang dạng tuyệt đối nếu không có tham chiếu thuận và lệnh theo sau JMP hoặc CALL ở trong cùng một khối 2K với lệnh đích nếu không thể dùng dạng gần thì sẽ biến đối thành dạng địa chỉ dài.

Việc biến đổi không nhất thiết chọn lựa cách lập trình tốt nhất. Ví dụ lệnh nhảy thuận đến đích chỉ cách một vài lệnh kế tiếp thì dùng tổng quát JMP được chuyển thành LJMP mặc dù nếu chọn SJMP thì tốt hơn. Hãy xem một đoạn chương trình dùng ba lệnh nhảy tổng quát sau đây: lệnh nhảy thứ nhất (dạng thứ 3) được dịch thành SJMP vì đích đến ở trước lệnh nhảy (có nghĩa là không có tham chiếu thuận) và giá trị offset trong khoảng -128 đến +127, Chỉ dẫn ORG trong dạng 4 tạo một khoảng cách 200 vị trí giữa nhãn START và lệnh nhảy thứ hai nên việc biến đối ở dạng thứ 5 là AJMP vì offset lớn hơn phạm vi của SJMP, và cũng lưu ý là địa chỉ của lệnh theo sau lệnh nhảy thứ hai (12FCH) và địa chỉ của START ở trong cùng một trang 2k (100 H – 17FFH).

Lệnh nhảy thứ ba được dịch thành LJMP vì đích nhảy (FINISH) chưa được định nghĩa khi dịch lệnh JMP (có nghĩa là tham chiếu thuận được sử dụng).

4. Tính biểu thức trong khi hợp dịch

Các giá trị và hằng số trong vùng toán hạng có thể được biểu diễn theo 3 cách: (a) cho biết rõ ràng (VD: 0EFH), (b) dùng kí hiệu có sẵn (VD: ACC) hoặc (c) dùng biểu thức (VD: 2+3). Việc dùng biểu thức là một đặc điểm mạnh khi tạo chương trình bằng hợp ngữ giúp dễ đọc và linh động hơn. Khi một biểu thức được áp dụng thì assembler sẽ tính toán giá trị của nó và chèn vào mã lệnh. Tất cả các biểu thức được tính toán trên cơ sở phép tính số học 16 bít. Tuy nhiên, kết quả 8 bít hoặc 16 bít sẽ được chèn vào câu lệnh tùy theo yêu cầu. Ví dụ: Các dòng lệnh sau đây là tương đương.

```
MOV DPTR, # 04FFH + 3
MOV  DPTR, # 0502H ; toàn bộ 16 bít được dùng.
```

Tuy nhiên, nếu áp dụng cùng biểu thức trên cho câu lệnh MOV A, # data th sẽ xuất hiện lỗi "value will not fit in a byte" , khái quát về các quy luật tính toán biểu thức như sau:

Cơ số

Cơ số của một hằng số được Intel quy định như sau: Số nhị phân được kết thúc bằng chữ "B", số bát phân là "O" hoặc "Q", "H" đối với số thập lục phân và "D" hoặc không có gì đối với số thập phân. Các lệnh sau đây là tương đương.

```
MOV A, # 15
MOV A, # 1111B
MOV A, # 0FH
MOV A, # 17Q
MOV A, #15D
```

Lưu ý: Phải có một kí số trước kí tự số hexa để phân biệt chúng với nhãn ("0A5H")

Chuỗi kí tự

Chuỗi dùng 1 hoặc 2 kí tự có thể dùng làm toán hạng trong biểu thức, trình dịch hợp ngữ sẽ chuyển mã ASCII sang số nhị phân tương đương, các hằng kí tự được bao bởi dấu móc đơn. Ví dụ: CJNE A, # 'Q'

```
AGAIN    SUBB A, # 'Q'  ; chuyển ASCII sang binary
         MOV DPTR, # 'AB'
         MOV DPTR, # 4142H ; giống lệnh phía trên.
```

Toán tử số học Các toán tử số học gồm có:

+ phép cộng
- phép trừ
* phép nhân
/ phép chia

MOV phép modulo (số dư của phép chia)

Ví dụ hai lệnh sau đây tương đương nhau: MOV

```
A, # 10 + 10 H
MOV A, # 1AH
```

Hai lệnh sau cũng tương đương nhau: MOV

```
A, # 25 MOD 7
MOV A, # 4
```

Vì toán tử MOD có thể bị hiểu là một kí tự nên cần phải phân cách toán tử này với toán hạng ít nhất một kí tự space hoặc Tab, hoặc các toán hạng phải được bao trong ngoặc đơn, áp dụng tương tự đối với các toán tử khác có cú pháp bằng kí tự chữ.

Toán tử logic Các toán tử logic gồm có

- OR phép OR
- AND phép AND
- XOR phép EXOR
- NOT phép đảo

Các phép trên xử lý từng bít của toán hạng, toán tử và toán hạng phải cách nhau bởi kí tự space hoặc Tab. Ví dụ: hai lệnh sau đây là tương đương: MOV A, # '9' AND 0FH

```
MOV A, #  9
```

Toán tử NOT chỉ có một toán hạng, ba lệnh MOV sau đây tương đương nhau:

```
THERE  EQU 3
MINUS-THERE EQU -3
MOV  A, # (NOT THERE) +1
MOV  A, # MINUS-THERE
MOV  A, # 11111101B
```

Các toán tử đặc biệt

Các toán tử đặc biệt gồm có

SHR dịch phải

SHL dịch trái

HIGH byte cao

LOW byte thấp

( ) được tính trước tiên

Hai lệnh say đây là tương đương nhau MOV

```
A, # HIGH   1234 H
MOV A, # 12 H
```

Toán tử quan hệ

Khi thao tác quan hệ được áp dụng giữa hai toán hạng, kết quả chỉ có hai giá trị là sai (0000H) hoặc đúng (FFFFH), các toán tử này gồm:

EQ = bằng nhau

NE <> khác nhau

LT < nhỏ hơn

LE <= nhỏ hơn hoặc bằng

GT > lớn hơn

GE >= lớn hơn hoặc bằng

Lưu ý: Là mỗi toán tử có hai cách biểu diễn (VD: "EQ" hoặc "=") các ví dụ quan hệ sau cho kết quả là "đúng".

```
MOV  A, # 5 = 5
MOV A, # 5 NE 4
MOV A, # 'X' LT 'Z'
MOV A, # 'X' = 'X'
MOV A, # 5 > 0
MOV A, # 100 GE 50
```

Tất cả các lệnh hợp ngữ trên tương đương lệnh sau

```
MOV A, # Ø FFH
```

Mặc dù việc tính toán các biểu thức cho ra kết quả 16 bít (0FFFFH) nhưng trong các ví dụ trên chỉ có byte thấp được sử dụng đó là do lệnh MOV byte. Kết quả không được xem là quá lớn trong trường hợp này bởi vì đối với số có dấu giá tr ị 16 bít FFFFH và giá trị 8 bít FFH cùng đều bằng -1.

Các ví dụ về biểu thức

Sau đây là các ví dụ về biểu thức và giá trị kết quả

Biểu thức	*Kết quả*
'B' – 'A'	0001H
8/3	0002H
155MOD2	0001H
4 * 4	0010H
8 AND 7	0000H
NOT 1	FFFEH
'A' SHL 8	4100H
LOW 65535	00FFH
(8+1)*2	0012H
5 EQ 4	0000H
'A' LT 'B'	FFFH
3 < = 3	FFFH

Một ví dụ thực tế trình bày cách khởi tạo timer như sau: Đặt -500 vào các thanh ghi TH1 và TL1 của timer 1 sử dụng các toán tử HIGH và LOW.

```
VALUE EQU – 500
MOV TH1, # HIGH VALUE
MOV TL1, # LOW VALUE
```

Trình hợp dịch biến đổi -500 giá trị 16 bít (FEOCH) sau đó toán tử HIGH và LOW sẽ đưa byte cao (FEH) và byte thấp (0CH) tương ứng với mỗi lệnh MOV.

Thứ tự ưu tiên

Thứ tự ưu tiên của các toán tử được liệt kê từ cao nhất đến thấp nhất.

( )
HIGH LOW
* / MOD SHL SHR
+ -
EQ NE LT LE GT GE = < > < <= > > =
NOT
AND
OR
XOR

Khi thực hiện các phép tính có cùng mức ưu tiên thì thứ tự tính là từ trái sang phải

Biểu thức	*Giá trị*
HIGH (‘A’ SHL 8)	0041H
HIGH ‘A’ SHL 8	0000H
NOT ‘A’ – 1	FFBFH
‘A’ OR ‘A’ SHL 8	4141H

Các chỉ dẫn

Các chỉ dẫn là các lệnh của phần mềm hợp ngữ, chúng không phải là các mã lệnh của vi điều khiển. Tuy chúng cũng được đặt trong vùng gợi nhớ của chương trình với ngoại lệ DB và DW không ảnh hưởng trực tiếp lên nội dung của bộ nhớ. ASM51 cung cấp một vài loại chỉ dẫn như sau:

- Các chỉ dẫn điều khiển trạng thái (ORG, END, USING)
- Chỉ dẫn định nghĩa kí hiệu (SEGMENT, EQU, SET, DATA, IDATA, XDATA, BIT, CODE)
- Khởi tạo vùng nhớ (DS, DBIT, DB, DW)
- Liên kết chương trình (PUBLIC, EXTRN, NAME)
- Chọn vùng nhớ đoạn (RSEG, CSEG, DSEG, ISEG, BSEG, XSEG)
- Điều khiển trạng thái

ORG (set origin)

Dạng thức của chỉ dẫn ORG

ORG biểu thức

Chỉ dẫn ORG thay đổi bộ đếm vị trí để xác định điểm bắt đầu của các dòng lệnh tiếp theo sau, không cho phép đặt nhãn tại chỉ dẫn này.

```
ORG 100H    ; đặt bộ đếm vị trí về 100H
ORG ($+1000H); AND 0F00H  ; đặt bộ đếm về khối 4K kế tiếp.
```

Chỉ dẫn ORG sử dụng được với mọi vùng nhớ đoạn. Nếu đoạn hiện hành là tuyệt đối thì giá trị sẽ là một địa chỉ tuyệt đối trong đoạn hiện hành. Nếu đoạn định vị được tác động thì giá trị biểu thức ORG được xem như là offset từ địa chỉ cơ sở của vị trí hiện hành trong đoạn.

END

Cú pháp của chỉ dẫn END

END

Chỉ dẫn END nên đặt ở vị trí cuối của file nguồn, không cho phép dùng nhãn và trnh dịch sẽ không xử lý những gì theo sau lệnh END.

USING

Cú pháp

USING biểu thức

Chỉ dẫn này báo cho ASM51 biết về dãy thanh ghi đang hoạt động, sau chỉ dẫn này các thanh ghi được ký hiệu là AR0 đến AR7 và các ký hiệu này sẽ được biến đổi thành địa chỉ trực tiếp tương ứng trong dãy thanh ghi hoạt động.

```
USING 3
PUSH AR7
USING 1
PUSH AR7
```

Lệnh PUSH đầu tiên sẽ đẩy dữ liệu vào 1FH (R7 trong dãy thanh ghi 3) lệnh PUSH thứ hai đẩy dữ liệu vào 0FH (R7 trong dãy thanh ghi 1).

Lưu ý: USING không thực hiện việc chuyển đổi giữa các dãy thanh ghi mà nó chỉ báo dãy thanh ghi hoạt động cho ASM51, việc chuyển đổi thanh ghi chỉ được thực hiện bằng các lệnh của 8051

```
MOV PSW, # 00011000B ; chọn dãy thanh ghi 3   USING
3   ;
PUSH AR7   ; dịch thành
PUSH 1FH
MOV PSW, # 00001000B ; chọn dãy thanh ghi 1
```

```
PUSH AR7   ; dịch thành PUSH 0FH
```

Định nghĩa kí hiệu

Chỉ dẫn này tạo ra các kí hiệu để biểu diễn vùng nhớ đoạn, thanh ghi, số và địa chỉ, không được dùng nhãn đứng trước chỉ dẫn này. Các kí hiệu được định nghĩa bởi chỉ dẫn này không được định nghĩa trước và sau đó cũng không được định nghĩa lại, ngoại trừ chỉ dẫn SET. Các chỉ dẫn định nghĩa kí hiệu được mô tả như sau:

Segment

Cú pháp kí hiệu SEGMENT kiểu đoạn

Kí hiệu là tên của đoạn định vị lại. Trong việc dùng các đoạn, ASM51 là trình dịch phức tạp hơn các trình dịch hợp ngữ thông thường khác, các trình này chỉ hỗ trợ kiểu đoạn "code" và "data". Tuy nhiên, ASM51 có định nghĩa thêm các kiểu đoạn khác, dưới đây là các kiểu đoạn được định nghĩa cho 8051.

- CODE (đoạn mã)
- XDATA (vùng nhớ dữ liệu ngoài)
- DATA (vùng nhớ dữ liệu bên trong, truy xuất bằng địa chỉ trực tiếp 00H 7FH)
- IDATA (toàn bộ vùng nhớ trong truy xuất bằng địa chỉ gián tiếp 00H-7FH, 00H – FFH đối với 8052)
- BIT (vùng nhớ bít, là các byte từ 20H – 2FH của bộ nhớ dữ liệu trong)

Ví dụ:

```
EPROM SEGMENT CODE
```

Khai báo EPROM là một SEGMENT kiểu CODE, lưu ý là câu này chỉ đơn giản khai báo EPROM là gì, để thực sự bắt đầu sử dụng SEGMENT này thì phải dùng chỉ dẫn RSEG. ***EQU (equate)***

Cú pháp Ký hiệu EQU biểu thức

Chỉ dẫn này gán một giá trị số cho ký hiệu, ví dụ

```
 N27 EQU 27       ; gán giá trị 27 cho N27
HERE EQU $        ; gán giá trị của bộ đếm
                  ; vị trí cho HERE
OR   EQU 0DH ; đặt OR bằng 0DH
MESSAGE DB 'This is a message'
 LENGTH EQU $-MESSAGE ; "LENGTH" là độ dài của
```

“MASSAGE”

Các chỉ dẫn định nghĩa kí hiệu khác

Chỉ dẫn SET tương tự như EQU chỉ khác là sau đó có thể định nghĩa lại kí hiệu bằng một chỉ dẫn SET khác. Các chỉ dẫn DATA, IDATA, XDATA, BIT và CODE sẽ gán địa chỉ của kiểu đoạn tương ứng cho kí hiệu, các chỉ dẫn này không phải là duy nhất, một kết quả tương tự có thể nhận được bằng cách dùng chỉ dẫn EQU. Tuy nhiên, nếu được dùng chung sẽ cung cấp một khả năng kiểm tra loại bằng ASM51. Xem ví dụ sau: FLAG 1 EQU 05H

```
FLAG 2 BIT 05H
SETB FLAG 1
SETB FLAG 2
MOV FLAG 1, # 0
MOV FLAG 2, # 0
```

Việc sử dụng FLAG 2 trong dòng lệnh cuối sẽ xuất hiện thông báo lỗi “ data segment address expected” bởi ASM 51 vài FLAG 2 được định nghĩa là một địa chỉ bít do chỉ dẫn bit nên nó có thể được dùng trong lệnh set bit nhưng không thể dùng trong lệnh MOV byte. Mặc dù FLAG 1 có cùng giá trị 05H được định nghĩa bởi EQU và không có một không gian địa chỉ tương ứng, đây không phải là ưu điểm của EQU mà nói khác hơn là khuyết điểm, bằng cách định nghĩa các kí hiệu địa chỉ tương ứng với vùng nhớ xác định thông qua các chỉ dẫn BIT, DATA, XDATA … người lập trình sẽ tận dụng được ưu điểm của ASM 51 nhờ khả năng kiểm tra và tránh được các rắc rối khi dùng sai kí hiệu.

Khởi tạo và dành trước vùng nhớ

Các chỉ dẫn này sẽ khởi tạo và dành trước vùng nhớ có độ dài là word, byte hoặc bít. Vùng nhớ dành trước bắt đầu tại vị trí được xác định bởi giá trị hiện hành của bộ đếm vị trí trong đoạn tác động hiện hành, có thể đặt nhãn phía trước các chỉ dẫn này, các chỉ dẫn này là:

DS (define storage)

Cú pháp

```
[ nhãn:] DS biểu thức
```

Vùng nhớ dành trước của chỉ dẫn này là byte, nó có thể được dùng cho mọi kiểu đoạn ngoại trừ BIT, biểu thức phải là biểu thức thời gian dịch hợp lệ không có tham chiếu thuận và không định vị lại hoặc tham chiếu ngoài. Khi nhận thấy phát biểu DS trong chương trình, bộ đếm vị trí của đoạn hiện hành được tăng lên bởi giá trị của biểu thức, tổng giữa giá trị của bộ đếm vị trí với giá trị của biểu thức không được vượt quá giới hạn của không gian địa chỉ hiện hành. Các phát biểu sau đây tạo ra một vùng đệm 40 byte trong đoạn dữ liệu nội: DSEG AT 30H ; đặt trong đoạn dữ liệu (tuyệt đối, bên trong) LENGTH EQU 40

```
BUFFER:  DS LENGTH ; dành trước 40 byte.
```

Nhãn BUFFER biểu diễn địa chỉ vị trí đầu tiêu của vùng nhớ dành trước. Trong ví dụ này địa chỉ đầu tiên là 30H bởi vì AT 30H được định nghĩa bởi DSEG. Vùng nhớ này có thể bị xóa bởi các lệnh sau: MOV

```
        R7 , # LENGTH
        MOV R0, # BUFFER
Loop:   MOV @ R0, # 0
        DJNZ R7, Loop. (tiếp tục)
```

Để tạo một vùng đệm 1000 byte trong RAM ngoài bắt đầu tại 4000H, các chỉ dẫn sau đây được sử dụng:

```
XSTART    EQU 4000H
XLENGTH   EQU 1000
              XSEG AT XSTART   XBUFFER
  DS XLENGTH
```

Có thể xóa vùng đệm này bằng các lệnh sau đây. MOV

```
  DPTR, # XBUFFER
Loop:CLR A
    MOVX @DPTR, A
    INC  DPTR
    MOV A, DPL
    CJNE A, # Low (XBUFFER + XLENGTH + 1), Loop
    MOV A, DPH CJNE A, # HIGH (XBUFFER + XLENGTH + 1), Loop
```

(tiếp tục)

Ví dụ trên minh họa các sử dụng các toán tử của ASM 51 và các biểu thức thời gian dịch hợp lệ; vì không có lệnh so sánh giữa con trỏ dữ liệu với một giá trị bất tức thời nên cần phải so sánh hai lần, một với byte cao và một với byte thấp của DPTR, hơn thế nữa lệnh CJNE chỉ làm việc với thanh ghi A hoặc một thanh ghi nên các byte của DPTR phải được chuyển vào A trước lệnh CJNE, vòng lập chỉ kết thúc khi DPTR bằng XBUFFER + XLENGTH + 1 (phải cộng thêm 1 vì DPTR được tăng lênh sau lệnh MOVX). ***DBIT***

Cú pháp: [nhãn:] DBIT biểu thức

Chỉ dẫn này dành trước một vùng nhớ có đơn vị là BIT và chỉ được dùng được trong đoạn BIT. Biểu thức phải là biểu thức thời gian dịch hợp lệ không có tham chiếu thuận. Khi gặp phát biểu DBIT trong chương trình, bộ đếm vị trí của đoạn bít hiện hành được tăng lên bởi giá trị của biểu thức. Lưu ý rằng trong đoạn

BIT, đơn vị cơ bản của bộ đếm vị trí là bít, không phải là byte, các chỉ dẫn sau đây tạo ra ba cờ trong một đoạn BIT tuyệt đối.

```
BSEG  ; đoạn BIT (tuyệt đối)
KBFLAG: DBIT 1 ; trạng thái bàn phím
PRFLAG: DBIT 1 ; trạng thái máy in
DKFLAG: DBIT 1 ; trạng thái đĩa
```

Vì trong ví dụ trên không có định nghĩa địa chỉ cho BSEG, địa chỉ của các cờ được định nghĩa bởi DBIT có thể được xác định (nếu người ta muốn làm như thế) bằng cách kiểm tra bảng kí hiệu trong các file . LST hoặc .M51. Nếu các định nghĩa ở trên là lần áp dụng đầu tiên của BSEG thì KBFLAG được đặt tại vị trí 00H (bít 0 của địa chỉ byte 20H). Nếu các bít khác được định nghĩa trước bằng BSEG thì sau đó các bít ở trên được đặt sau bít cuối cùng đã định nghĩa. ***DB*** (define byte)

Cú pháp: [nhãn:] DB biểu thức [, biểu thức] [….]

Chỉ dẫn DB khởi tạo vùng nhớ mã với giá trị byte vì thực tế chỉ dẫn này đặt hằng số dữ liệu trong vùng nhớ mã, một đoạn mã phải được tác động. Danh sách biểu thức là một chuỗi gồm một hoặc nhiều giá trị byte (mỗi byte có thể là một biểu thức được cách nhau bởi dấu phẩy) Chỉ dẫn DB cho phép dùng chuỗi kí tự (được bao bằng dấu móc đơn) dài hơn 2 kí tự và không phải là một phần của biểu thức. Mỗi kí tự trong chuỗi được biến đổi thành mã ASCII tương ứng. Nếu một nhãn được sử dụng nó sẽ được gán địa chỉ của byte đầu tiên. Ví dụ:

```
CSEG AT 0100H
SQUARES DB 0,1,4,9,16,25 ; bình phương của các số
MESSAGE: DB  'Login', 0  ; kí tự kết thúc chuỗi NULL  Khi hợp dịch
```

kết quả dạng hexa được gán cho bộ nhớ chương trình bên ngoài.

Địa chỉ	*Nội dung*
0100	00
0101	01
0102	04
0103	09
0104	10
0105	19
0106	4C
0107	6F

0108	67
0109	69
010A	6E
010B	3A
010C	00

DW (define word)

Cú pháp: [nhãn:] DW biểu thức [, biểu thức] […]

Tương tự như DB chỉ khác là chỉ dẫn này dùng đến 2 vị trí nhớ (16 bít) gán cho mỗi một đề mục dữ liệu

```
CSEG AT 200H
DW $, ‘A’, 1234H, 2 , ‘RC’
```

Kết quả ở dạng thập lục phân như sau

Địa chỉ	*Nội dung*
0200	02
0201	00
0202	00
0203	41
0204	12
0205	34
0206	00
0207	02
0208	42
0209	43

Liên kết chương trình

Các chỉ dẫn liên kết chương trình cho phép các modul (file) đã được hợp dịch riêng có thể liên kết với nhau bằng cách cho phép một tham chiếu trên modul và bằng cách đặt tên các modul, một modul được xem như là một file

(trong thực tế một modul có thể gồm nhiều file).

PUBLIC

Cú pháp: PUBLIC [, kí hiệu] […]

Chỉ dẫn PUBLIC cho phép một danh sách các modul xác định được biết và sử dụng bên ngoài modul hợp dịch hiện hành. Một kí hiệu được khai báo

PUBLIC phải được định nghĩa trong modul hiện hành qua việc khai báo PUBLIC cho phép modul này được tham chiếu trong các modul khác. Vd: PUBLIC INCHAR, OUTCHAR, INLINE, OUTSTR

EXTRN

Cú pháp: EXTRN kiểu đoạn kí hiệu [,kí hiệu] […]

Chỉ dẫn EXTRN liệt kê các kí hiệu được tham chiếu trong modul hiện hành, các kí hiệu này được định nghĩa trong các modul khác. Danh sách các kí hiệu bên ngoài phải có một kiểu đoạn kèm theo với mỗi kí hiệu trong danh sách (Các kiểu đoạn là CODE, XDATA, DATA, IDATA, BIT và NUMBER. NUMBER là một kí hiệu không kiểu được định nghĩa bởi EQU) kiểu đoạn cho biết phương pháp mà một kí hiệu được sử dụng thông tin này là quan trọng trong thời gian liên kết để bảo đảm các kí hiệu được sử dụng một cách thích hợp trong các modul khác. Các chỉ dẫn PUBLIC và EXTRN hoạt động chung với nhau. Hãy xem hai file sau đây MAIN.SRC và MESSAGES.SRC, các chương trình con HELLO và
GOODBYE được định nghĩa trong modul MESSAGES, nhưng có thể được các modul khác sử dụng nhờ chỉ dẫn PUBLIC, các chương trình con này được gọi trong modul MAIN mặc dù chúng không được định nghĩa trong đó. Chỉ dẫn EXTRN khai báo các kí hiệu được định nghĩa trong các modul khác.

```
MAIN.SRC
                    EXTRN       CODE (HELLO, GOOD_BYE)
                    CALL  HELLO
                        CALL    GOOD_BYE
                    END

MESSAGES.SRC
                    PUBLIC      HELLO, GOOD_BYE
                    ---
HELLO:              (bắt đầu chương trình con)
                    ---
                    RET.
GOOD_BYE:  (bắt đầu chương trình con)
                    ---
                    RET
                    ---
                        END
```

Các file MAIN.SRC và MESSAGES.SRC đều không phải là một chương trình hoàn chỉnh chúng phải được hợp dịch riêng và sau đó liên kết với nhau để tạo thành chương trình thực hiện được. Trong quá trình liên kết các tham chiếu bên ngoài được tính với địa chỉ đúng và chèn vào đích của các lệnh gọi.

NAME

Cú pháp: NAME tên modul

Tên modul áp dụng cùng một quy tắc như tên của kí hiệu, nếu không cho biết tên, modul sẽ lấy tên file (không có đường dẫn và phần mở rộng). Nếu không có chỉ dẫn NAME thì một chương trình sẽ bao gồm 1 modul cho mỗi file.

Khái niệm modul là một vấn đề đối với các chương trình nhỏ kể cả các chương trình kích thước trung bình. Vd: Bao gồm vài file hoàn chỉnh với các segment có thể định vị lại. Tuy nhiên, đối với các chương trình lớn (vài ngàn dòng lệnh hoặc nhiều hơn) nên chia nhỏ vấn đề thành nhiều modul trong đó mỗi modul có thể gồm nhiều file chứa các chương trình có chung một mục đích.

Các chỉ dẫn chọn segment

Khi trình dịch hợp ngữ gặp một chỉ dẫn nó sẽ hướng mã lệnh hoặc dữ liệu vào segment đã chọn cho đến khi một segment khác được chọn bởi chỉ dẫn chọn segment. Chỉ dẫn này có thể chọn một segment định vị lại đã được chọn trước hoặc tạo ra và chọn các segment tuyệt đối. ***RSEG*** (Relocatable segment)

Cú pháp: RSEG tên segment Trong đó "tên segment" là tên của segment định vị lại đã được định nghĩa trước chỉ dẫn SEGMENT, chỉ dẫn RSEG sẽ hướng mã lệnh và dữ liệu theo sau vào segment có tên kèm theo cho đến khi gặp 1 chỉ dẫn chọn segment khác. ***Chọn các segment tuyệt đối***

RSEG dùng để chọn một segment định vị lại còn segment tuyệt đối được chọn bằng các chỉ dẫn sau:

CSEG (AT địa chỉ)

DSEG (AT địa chỉ)

ISEG (AT địa chỉ)

BSEG (AT địa chỉ)

XSEG (ATđịa chỉ)

Các chỉ dẫn này chọn một segment tuyệt đối trong vùng mã lệnh, vùng dữ liệu trong, vùng dữ liệu trong gián tiếp, vùng bít, hoặc vùng dữ liệu ngoài. Nếu một segment tuyệt đối được cung cấp (bằng thông báo "AT địa chỉ") thì trình dịch hợp ngữ sẽ kết thúc segment tuyệt đối cuối cùng nếu có kiểu segment xác định và tạo ra một segment tuyệt đối khác bắt đầu tại địa chỉ trên. Nếu không xác định địa chỉ tuyệt đối thì segment tuyệt đối cuối cùng của kiểu segment đã xác định vẫn tiếp tục. Nếu không có segment tuyệt đối nào thuộc kiểu này đã được chọn trước đó và địa chỉ tuyệt đối được bỏ qua thì một segment mới sẽ được tạo ra và bắt đầu tại vị trí 0. Không cho phép các tham chiếu thuận và địa chỉ bắt đầu phải là tuyệt đối. Mỗi segment có 1 bộ đếm vị trí riêng và bộ đếm này luôn được đặt

về 0 khi khởi tạo, segment mặc định là một segment mã lệnh tuyệt đối vì vậy trạng thái ban đầu của assembler là vị trí 0000H trong segment mã lệnh địa chỉ tuyệt đối. Khi một segment khác được chọn lần đầu tiên bộ đếm vị trí của segment trước còn duy trì giá trị tác động cuối cùng, khi segment trước đó được chọn lại, bộ đếm vị trí sẽ nhận giá trị tác động sau cùng này. Chỉ dẫn ORG có thể được dùng để thay đổi bộ đếm vị trí trong segment đã được chọn đang hiện hành. Hình 7.2 là một ví dụ về việc định nghĩa, khởi tạo các segment tuyệt đối và định vị lại

```
LOC  OBJ     LINE     SOURCE
               1      ONCHIP    SEGMENT   DATA ; một segment dữ
                                          ; liệu định vị lại
               2      EPROM     SEGMENT   CODE ; một segment mã
                                          ; lệnh định vị lại
               3
               4      BSEG      AT70H     ; bắt đầu segment
                                          ; BIT tuyệt đối
0070           5      FLAG1:    DBIT  1
0071           6      FLAG2:    DBIT  1
               7
               8      RSEG      ONCHIP          ; bắt đầu segment
                                          ; dữ liệu định vị lại
0000           9      TOTAL:    DS    1
0001           10     COUNT:    DS    1
0002           11     SUM16:    DS    2
               12
               13     RSEG      EPROM           : bắt đầu segment
                                          ;mã lệnh định vị lại
0000 750000    14     BEGIN:    MOV   TOTAL, # 0;
               15     (tiếp tục chương trình)
               16     END
```

Hình 7.2. Định nghĩa và khởi tạo segment tuyệt đối và định vị lại

Hai dòng đầu tiên mô tả các kí hiệu ONCHIP và EPROM là các segment loại DATA (RAM trong) và CODE. Dòng 4 bắt đầu segment tuyệt đối BIT khởi tạo tại địa chỉ bit 70H(bít 0 của byte 2EH), sau đó FLAG1 và FLAG2 được tạo ra bởi các nhãn với địa chỉ bit tương ứng là 70H và 71H. Chỉ dẫn RSEG ở dụng 8 bắt đầu segment định vị lại ONCHIP trong vùng RAM trong. TOTAL và COUNT là các nhãn tương ứng với các vị trí byte. SUM16 là một nhãn tương ứng với một vị trí word. Chỉ dẫn RSEG kế tiếp ở dòng 13 bắt đầu segment định vị lại EPROM, trong vùng nhớ chương trình. Nhãn BEGIN là địa chỉ của lệnh đầu tiên trong thể hiện này của EPROM. Lưu ý là không thể xác định được địa chỉ của các nhãn TOTAL, COUNT, và BEGIN vì các nhãn này nằm trong segment định vị lại File đối tượng phải được xử lý bằng trình liên kết / định vị với địa chỉ đầu được xác định các đoạn ONCHIP và EPROM, file liệt kê. M51 được tạo ra bởi tính liên kết sẽ cho biết địa chỉ tuyệt đối của nhãn này. Tuy nhiên, FLAG1 và FLAG2 luôn

luôn tương ứng với các địa chỉ bít 70H và 71H vì chúng được định nghĩa trong đoạn tuyết đối BIT.

5. Các điều khiển Assembler

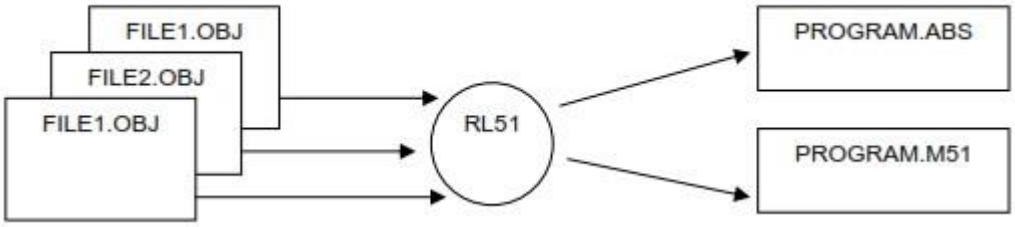
Các điều khiển hợp dịch tạo khuôn dạng cho file liệt kê và file đối tượng bằng cách điều phối các hoạt động của ASM51, thường các điều khiển hợp dịch ảnh hưởng đến hình thức của file liệt kê nhưng không ảnh hưởng đến chương trình, chúng có thể được đưa vào dòng lệnh khi hợp dịch chương trình hoặc đưa vào trong file nguồn. các điều khiển hợp dịch trong file nguồn phải được bắt đầu bằng dấu $ và nằm trong cột thứ nhất. Có hai loại điều khiển hợp dịch: Loại cơ bản và loại tổng quát. Các điều khiển cơ bản có thể được đặt trong dòng lệnh hoặc tại vị trí bắt đầu trong file nguồn, chỉ có các điều khiển cơ bản mới có thể đặt trước một điều khiển cơ bản khác. Các điều khiển tổng quát có thể được đặt ở bất kỳ nơi nào trong file nguồn. Hình 7.3 trình bày các điều khiển hợp dịch được hỗ trợ bởi ASM51.

TÊN	P/G	Mục đích	Viết tắt	Ý nghĩa
DATA (date)	P	DATE ()	DA	Đặt chuỗi trong header (9 ký tự)
DEBUG	P	NODEBUG	DB	xuất thông tin ký hiệu debug cho file đối tượng
NODEBUG	P	NODEBUG	NODB	không đặt thông tin ký hiệu debug trong file đối tượng
ERRORDRINT	P	NOERRORPRINT	EP	Chỉ định 1 file nhãn các thông báo lỗi cùng với file liệt kê
NOERRORPRINT	P	NOERRORPRINT	NOEP	Các thông báo lỗi chỉ được in trong file liệt kê
GEN	G	GENONLY	GE	Tạo một danh sách đầy đủ các quá trình mở rộng macro bao gồm các lời gọi macro trong file liệt kê
GENONLY	G	GENONLY	GO	Chỉ liệt kê mã nguồn được mở rộng đầy đủ giống như các dòng được tạo ra bởi lời gọi macro torng file nguồn.
NOGEN	G	GENONLY	NOGE	Chỉ liệt kê file nguồn gốc trong file liệt kê.
INCLUDE	G	không áp dụng	IC	Xác định fiel bao gồm trong chương Trình.
LIST	G	LIST	LI	In các dòng mã nguồn liên tiếp nhau trong file liệt kê.
NOLIST	G	LIST	NOLI	Không in các dòng mã nguồn liên tiếp nhau trong file liệt kê.
MACRO	P	MACRO (50)	MR	Đánh giá và mở rộng tất cả các lời gọi macro, cấp phát phần trăm bộ nhớ còn trống cho việc xử lý macro.
NOMACRO	P	MACRO (50)	NOMR	Khống đánh giá các lời gọi macro.
M0051	P	M0051	MO	Nhận biết các SFR đã định nghĩa của 8051.
NOM0051	P	M0051	NOMO	Không nhận biết các SFR đã định nghĩa của 8051.
OBJECT [(file)]	P	OBJECT (source.Obj)	OJ	Xác định file nhân mã đối tượng.
NOOBJECT	P	OBJECT (source.Obj)	NOOJ	Không tạo ra file đối tượng.
PAGING	P	PAGING	PI	File liên kết tách thành nhiều trang, mỗi trang có một header.
NOPAGING	P	PAGING	NOPI	File liên kết tách thành nhiều trang.
PAGELENGTH (n)	P	PAGELENGTH (60)	PL	Thiết lập số dòng tối đa cho một trang của file liệt kê (10 – 65535).
PAGEWIDTH (n)	P	PAGEWIDTH (120)	PW	Thiết lập số ký tự tối đa trên một dòng (72 – 132).
PRINT [(file)]	P	PRINT (source.LST)	PR	Xác định file nhận liệt kê nguồn.
NOPRINT	P	PRINT (source.LST)	NOPR	Không tạo ra file liệt kê.
SAVE	G	không áp dụng	SA	Lưu các thiết lập điều khiển hiện hành cho LIST và GEN.
RESTORE	G	không áp dụng	RS	Phục hồi các thiết lập điều khiển từ ngăn xếp SAVE.
REGISTERBANK(rb..)	P	REGISTERBANK (0)	RB	Xác định một hoặc nhiều dây thanh ghi được dùng trong chương trình.
NOREGISTERBANK	P	REGISTERBANK (0)	NORB	Cho biết dây thanh ghi không dùng.
SYMBOLS	P	SYMBOLS	SB	Tạo một bảng định dạng cho tất cả các ký hiệu dùng trong chương trình.
NOSYMBOLS	P	SYMBOLS	NOSB	Không tạo bảng ký hiệu.
TITLE (string)	G	TITLE ()	TT	Đặt một chuỗi vào tất cả các header trong (60 ký tự max).
WORKFILES (path)	P	giống như source	WF	Chỉ đường dẫn cho file tạm.
XREF	P	NOXREF	XR	Tạo một danh sách tham chiếu chéo của tất cả các ký hiệu.
NOXREF	P	NOXREF	NOXR	Không tạo danh sách tham chiếu chéo.

Hình 7.3. Các điều khiển hợp dịch được hỗ trợ bởi ASM51

6. Hoạt động liên kết

Khi phát triển các chương trình ứng dụng lớn thông thường nên chia các công việc thành các chương trình con hoặc các modul trong đó chứa các đoạn mã lệnh (các chương trình con thường được viết riêng), phương pháp này gọi là lập trình theo modul. Các modul thường là loại định vị lại có nghĩa là chúng không được xác định một địa chỉ cụ thể trong vùng nhớ chương trình hoặc dữ liệu nên phải cần một chương trình liên kết / định vị để kết nối các modul thành một modul đối tượng tuyệt đối có thể chạy được



*Hình 7.4. Hoạt động liên k*ết

RL51 của Intel là một chương trình liên kết / định vị điển hình nó nhân các modul định vị lại làm ngõ vào và sau đó tạo ra chương trình mã máy chạy được và một file liệt kê chứa một bản đồ bộ nhớ và bảng kí hiệu.

Khi các modul định vị lại được kết nối, tất cả các giá trị của kí hiệu ngoài sẽ được tính toán và chèn giá trị tính được vào file kết quả, chương trình RL51 được khởi động như sau:

RL51 input-list [to output-file] [location-controls]

Input-list là danh sách các modul đối tượng định vị lại (các file) được phân cách bởi dấu phẩy, output-file là tên của modul đối tượng tuyệt đối được tạo ra, nếu không cung cấp tên cho modul này thì tên mặc định được chọn là tên của modul vào đầu tiên không có phần mở rộng, các điều khiển vị trí thiết lập các địa chỉ bắt đầu cho các segment đã được đặt tên.

Giả sử có ba modul hoặc file (MAIN.OBJ, MESSAGES.OBJ và SUBROUTINES.OBJ) chúng được kết nối để tạo ra chương trình EXAMPLE, mỗi modul nói trên chứa hai segment định vị lại là EPROM thuộc loại CODE và ONCHIP thuộc loại DATA. Giả sử đoạn mã lệnh được thực hiện tại địa chỉ 4000H và đoạn dữ liệu bắt đầu tại địa chỉ 30H (trong RAM nội), các dòng lệnh được sử dụng như sau:

RL51 MAIN.OBJ, MESSAGES.OBJ, SUBROUTINES.OBJ TO

EXAMPLE & CODE (EPROM(4000H) DATA (ONCHIP (30H))

Kí tự & cho biết là dòng lệnh cn tiếp tục.

Nếu chương trình bắt đầu tại nhãn START và đây là dòng lệnh đầu tiên trong modul MAIN thì sự thực thiện sẽ bắt đầu tại 4000H, nếu modul MAIN không được kết nối trước tiên hoặc nếu nhãn START không phải là vị trí bắt đầu của MAIN thì điểm nhập của chương trình có thể được xác định bằng cách kiểm tra bảng kí hiệu trong file liệt kê EXAMPLE.M51 được tạo ra bởi RL51. Mặc định file EXAMPLE.M51 chỉ chứa bản đồ liên kết, nếu cần một bảng kí hiệu thì mỗi file nguồn phải dùng điều khiển $DEBUG

7. Macro

Khả năng xử lý macro MPL (macro processing facility) của ASM51 là 2 khả năng thay thế chuỗi. Macro cho phép thường xuyên sử dụng các đoạn mã lệnh đã được định nghĩa một lần bằng một dạng ghi nhớ đơn giản và được áp dụng bất cứ nơi đâu trong chương trình bằng cách chèn dạng mã gợi nhớ này vào. Phương pháp lập trình là một mở rộng của các kỹ thuật đã bàn trước đây. Macro có thể được định nghĩa bất cứ nơi đâu trong chương trình nguồn và sau đó được sử dụng giống như một lệnh bất kỳ khác. Cú pháp định nghĩa macro như sau: % * DEFINE (call – pattern) (macro – body)

Mỗi một lần được định nghĩa, mẫu gọi (call-pattern) macro giống như một lệnh gợi nhớ, nó có thể được dùng như một lệnh hợp ngữ bằng cách đặt vào vùng gợi nhớ của chương trình. Các macro được phân biệt với các lệnh hợp ngữ bằng dấu % đặt ở phía trước. Khi chương trình nguồn được hợp dịch, mọi thứ trong thân macro được thay thế bằng mẫu gọi macro. Các macro cung cấp một phương tiện đơn giản để thay thế các khối lệnh hợp ngữ phức tạp bằng một mã gợi nhớ đơn giản dễ hiểu. Ví dụ định nghĩa macro sau đây được đặt tại phần mở đầu của file nguồn.

```
% * DEFINE        PUSH – DPTR)
                  (PUSH   DPH
                  PUSH    DPL)
```

Sau đó phát biểu

% PUSH – DPTR

Sẽ xuất hiện trong file liệt kê như sau

```
PUSH    DPH
PUSH    DPL
```

Ví dụ trên là một macro điển hình vì các lệnh về ngăn xếp của 8051 chỉ làm việc với địa chỉ trực tiếp nên việc chuyển dữ liệu vào DPTR phải cần đến hai lệnh

PUSH; một macro tương tự có thể được tạo ra để POP dữ liệu từ DPTR. Những ưu điểm khác khi dùng macro.

+ Chương trình nguồn dễ đọc hơn vì lệnh macro dễ nhận ra hoạt động hơn là các lệnh hợp ngữ tương đương.

+ Chương trình nguồn viết ngắn gọn.

+ Giảm được lỗi

Giúp lập trình viên không cần phải bận tâm đến các chi tiết nhỏ nhặt. Hai ưu điểm sau cùng có quan hệ với nhau, mỗi khi macro đã được viết và sửa lỗi xong thì chúng được sử dụng mà không sợ gây ra lỗi. Trong ví dụ PUSH – DPTR, ne71u dùng lệnh PUSH và POP thay vì dùng macro push và pop lập trình viên có thể vô tư đảo thứ tự của các lệnh PUSH và POP (byte cao hoặc byte thấp được cất vào trước), điều này sẽ tạo ra lỗi. Khi dùng macro các chi tiết này chỉ cần quan tâm một lần và khi macro đã được viết xong thì sau này khi dùng đến sẽ không lo xảy ra lỗi như trên.

Do việc thay thế macro dựa trên nguyên tắc từng kí tự một nên khi định nghĩa macro phải thật cẩn thận với các kí hiệu xuống dòng, tab… để bảo đảm sự chanh lề phù hợp giữa phát biểu macro với các lệnh hợp ngữ còn lại trong chương trình, đôi khi cần đến các phép thử và sửa lại.

Có nhiều đặc tính cải tiến khả năng xử lý của ASM51 đó là cho phép truyền thông số, các nhãn cục bộ, thao tác lập, điều khiển luồng hợp dịch…

Truyền thông số

Một macro với các thông số được truyền từ chương trình chính có dạng thức được sửa đổi như sau:

% * DEFINE (macro-name (parameter-list) (macro-body)

Ví dụ nếu macro sau đây được định nghĩa

```
% * DEFINE (CMPA # (VALUE))
(CJNE A, # % VALUE, $+3)  Thì
```

lời gọi macro như sau

```
% CMPA # (20H)
```

Sẽ được mở rộng thành lệnh sau trong file liệt kê

```
CJNE A, # 20H, $ +3
```

Mặc dù 8051 không có lệnh so sánh bộ tích lũy nhưng cũng có thể dễ dàng tạo ra bằng cách dùng lệnh CJNE với “$+3” (lệnh kế tiếp) là đích đến của lệnh nhảy có điều kiện; lệnh gọi nhớ CMPA # giúp mọi người dễ nhớ, bên cạnh đó việc dùng macro còn giúp lập trình viên không cần nhớ đến các chi tiết chú thích như “$+3”.

Hãy xem một ví dụ khác và thật là tốt nếu 8051 có các lệnh sau đây:

JUMP IF ACCUMULATOR CREATER THAN X

JUMP IF ACCUMULATOR CREATER THAN OR EQUAL TO X

JUMP IF ACCUMULATOR LESS THAN X

JUMP IF ACCUMULATROR LESS THAN OR EQUAL TO X Các thao tác này có thể được tạo ra bằng cách dùng CJNE và tiếp theo sau là JC hoặc JNC, nhưng các chi tiết này cũng phức tạp. VD để nhảy đến nhãn GREATER-THAN khi nội dung bộ tích lũy lớn hơn mã ASCII của kí hiệu "Z"
(5AH) có thể thực hiện bằng chuỗi lệnh sau:

```
CJNE  A, 5BH, $+3
JNC   CREATER-THAN
```

Lệnh CJNE lấy nội dung của A trừ cho 5BH ("Z" +1) và bit carry sẽ set và clear thích thợp. Lệnh CJNE đặt C = 1 khi nội dung của A từ 00H đến 5AH (5AH-5BH < 0, nên C = 1, nhưng 5BH – 5AH = 0 và C = 0). Việc nhảy đến nhãn CREATER – THAN nhưng không có carry sẽ được thực hiện với các giá trị của A là 5BH, 5CH, 5DH … cho đến FFH. Công việc sẽ đơn giản hơn nếu tạo ra một lệnh gợi nhớ tương tự bằng cách định nghĩa macro, dưới đây là macro
" jump if greater than".

```
% * DEFINE          ( JGT (VALUE, LABEL )
                    ( CJNE A, # % VALUE + 1, $ + 3    ; JGT    JNC  %
                    LABEL   )
```

Để thử lại nếu A chứa một mã ASCII lớn hơn "Z" như đã nói, thì macro sẽ được gọi như sau:

```
% JGT  ('Z', GEARTER-THAN)
ASM51 sẽ biến dòng lệnh này thành
CJNE   A, # 5BH, $ + 3      ; JGT
JNC    GREATER-THAN
```

Bằng cách sử dụng macro lập trình viên sẽ tránh được những chi tiết rắc rối dễ sinh ra lỗi.

Các nhãn cục bộ

Các nhãn cục bộ có thể dùng trong macro với dạng thức sau đây

% * DEFINE (macro-name [(parameter-list)])

[LOCAL list-of-local-labels] (macro-body) Ví dụ:

```
% * DEFINE  (DEC – DPTR)   LOCAL SKIP
        (DEC DPL ; giảm DPTR
        MOV A, DPL
        CJNE A, # 0FFH, % SKIP
        DEC  DPH
% SKIP:       )
```

Cách gọi macro %

DEC – DPTR

và ASM51 sẽ mở rộng thành

```
DEC  DPL
MOV  A, DPL
CJNE  A, # 0FFH, SKIP ØØ
DEC   DPH  SKIP ØØ:
```

Lưu ý là nhãn cục bộ không được xung đột với nhãn cùng tên ở nơi khác trong chương trình nguồn do ASM51 nối thêm một mã số vào mã cục bộ khi macro được mở rộng và những lần sử dụng tiếp theo nhãn cục bộ cùng tên lại nhận thêm các mã số khác. Macro ở trên có một ảnh hưởng phụ; vì thanh ghi A được dùng làm nơi lưu trữ tạm thời cho DPL nên nếu macro được áp dụng trong một đoạn mã có dùng thanh ghi A cho mục đích khác thì giá trị của A sẽ bị mất và gây lỗi cho chương trình, để khắc phục khi định nghĩa macro nên cất A vào ngăn xếp như trình bày sau đây.

```
% * DEFINE  (DEC-DPTR)  LOCAL SKIP       (PUSH
                ACC
                DEC DPL      ; giảm DPTR
                MOV   A, DPL
                CJNE   A, #0FFH, % SKIP
                DEC   DPH
  % SKIP:         POP   ACC       )
```

Thao tác lặp

Sau đây là một trong vài macro có sẵn

% REPEAT (expression) (text)

Ví dụ điền 100 lệnh NOP vào một vùng trong bộ nhớ

% REPEAT (100) (NOP)

Các thao tác điều khiển luồng

Việc hợp dịch có điều kiện các đoạn mã lệnh được cung cấp bởi định nghĩa macro điều khiển của ASM51, cú pháp như sau:

% IF (expression) THEN (balanced text)

[balanced text] Ví dụ:

```
  INTERNAL        EQU 1 ; 1 = 8051 serial I/O drivers
                  .
                  % IF (INTERNAL) THEN
                  (INCHAR: ; 9/8051 drivers
                              ;
                  OUTCHAR: ;
                              ) ELSE
```

INCHAR: , ; 8051 dirvers

,

OUTCHAR: ,

,

,

)

Trong ví dụ này, ký hiệu INTERNAL được cho giá trị 1 để chọn các chương trình còn về cổng nối tiếp của 8051 hoặc bằng 0 để chọn các chương trình con cho một UART bên ngoài, trong trường hợp này là 8251. Macro IF làm cho 8051 hợp dịch chương trình còn này và bỏ qua các chương trình con khác. Các chương trình INCHAR và OUTCHAR được sử dụng ở mọi nơi trong chương trình mà không cần quan tâm đến cấu hình phần cứng chỉ cần chương trình được hợp dịch với giá trị đúng cho INTERNAL thì chương trình con tương ứng sẽ được thực hiện.

TÀI LIỆU THAM KHẢO

1. Đề cương môđun/môn học nghề Sửa chữa thiết bị điện tử công nghiệp", *Dự án Giáo dục kỹ thuật và Dạy nghề (VTEP), Tổng cục Dạy Nghề, Hà Nội, 2003*
2. Microprocessor and IC families - *Walter H. Buchbaum. Sc.D*
3. Mikrocompute Lehrbuch - *HPI Fachbuchreihen Pflaum Verlag Munchen*
4. 8051 Development Boad, *Rev 5 - Paul Stoffregen*
5. Họ vi điều khiển - *Tống văn On - Đại học Bách khoa TP.HCM - 2005*