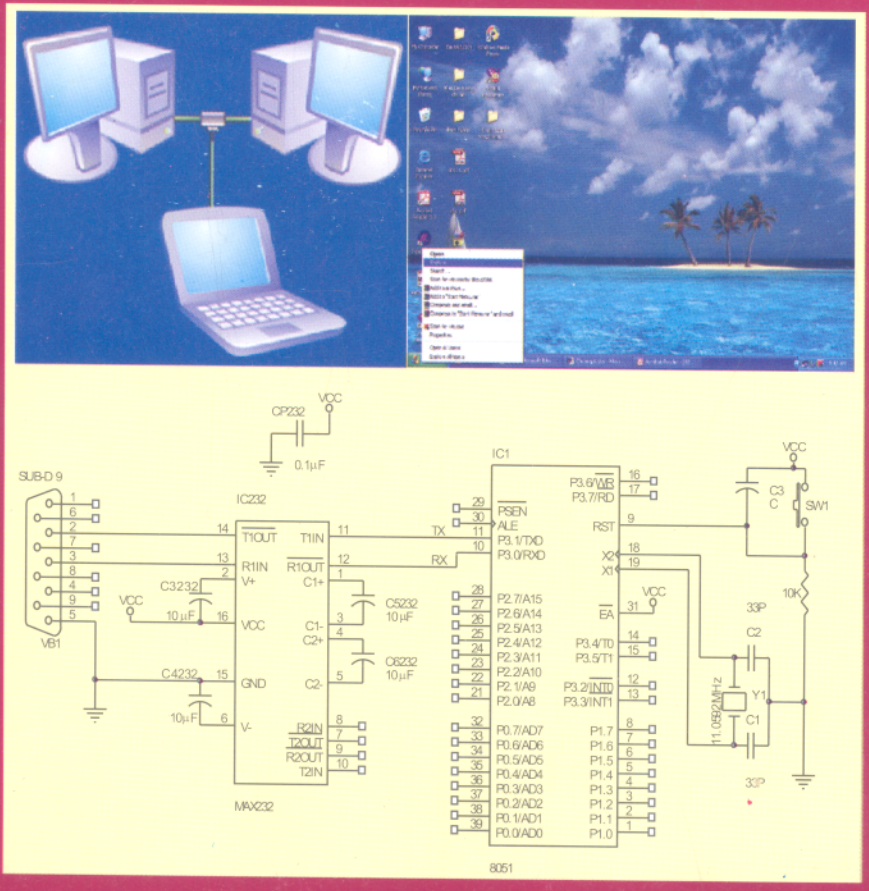
KIỀU XUÂN THỰC (Chủ biên)
VŨ THỊ THU HƯƠNG - VŨ TRUNG KIÊN

VI ĐIỀU KHIỂN

CẤU TRÚC - LẬP TRÌNH VÀ ỨNG DỤNG

(DÙNG CHO SINH VIÊN HỆ CAO ĐẲNG VÀ ĐẠI HỌC)



NHÀ XUẤT BẢN GIÁO DỤC

KIỀU XUÂN THỰC (Chủ biên)
VŨ THỊ THU HƯƠNG – VŨ TRUNG KIÊN

VI ĐIỀU KHIỂN

CẤU TRÚC – LẬP TRÌNH VÀ ỨNG DỤNG

(Dùng cho sinh viên hệ Cao đẳng và Đại học)

NHÀ XUẤT BẢN GIÁO DỤC

113–2008/CXB/72–175/GD Mã số: 7B718Y8 – DAI

LỜI NÓI ĐẦU

Sự ra đời của các bộ vi xử lý nói chung, các bộ vi điều khiển nói riêng đã tạo ra một bước ngoặt lớn trong việc thiết kế các hệ thống xử lý thông tin, đo lường điều khiển và truyền thông. Kết quả là đã tạo ra được những sản phẩm như máy ảnh số, máy chơi nhạc MP3, đầu đĩa DVD, các bộ biến tần, PLC,... ngày càng rẻ hơn, nhỏ gọn hơn, thông minh hơn và tiện dụng hơn. Cuốn sách "***Vi điều khiển – Cấu trúc, lập trình và ứng dụng***" được biên soạn nhằm giúp sinh viên Cao đẳng và Đại học các khối ngành Điện, Điện tử, Kỹ thuật máy tính tìm hiểu về cấu trúc, lập trình và ứng dụng các bộ vi điều khiển họ 8051 trong việc thiết kế các ứng dụng thực tế. Nội dung cuốn sách gồm 3 chương:

Chương 1: Giới thiệu qua các khái niệm về thông tin, biểu diễn thông tin trong máy tính; cấu trúc cơ bản và nguyên lý hoạt động của hệ vi xử lý; khái niệm về vi điều khiển, sự khác nhau giữa vi điều khiển và máy vi tính, đánh giá sơ lược về thiết kế ứng dụng vi điều khiển.

Chương 2: Giới thiệu về cấu trúc và hoạt động của vi điều khiển 89S52 – một chip phổ biến của họ 8051. Nội dung của chương bao gồm sơ đồ khối và chức năng các khối của 89S52; tập lệnh của họ 8051; hoạt động của các bộ đếm/định thời, cổng nối tiếp và xử lý ngắt.

Chương 3: Giới thiệu các bước cơ bản trong quá trình thiết kế ứng dụng có sử dụng vi điều khiển và hàng chục ví dụ cụ thể từ đơn giản (điều khiển LED nhấp nháy, LED 7 đoạn, động cơ bước...) đến phức tạp (bộ điều khiển mờ, robot dò đường với nhiều cấp tốc độ...). Đây là những ví dụ rất bổ ích với đầy đủ sơ đồ nguyên lý mạch điện và chương trình viết bằng ngôn ngữ C mà dựa vào đó bạn đọc có thể thiết kế, chế tạo được các ứng dụng thực tế hữu ích sử dụng họ vi điều khiển 8051 hay các họ vi điều khiển khác.

Mặc dù tài liệu này đã được sử dụng để giảng dạy nhiều năm tại Trường Đại học Công nghiệp Hà Nội nhưng cũng không tránh khỏi những sai sót. Chúng tôi rất mong nhận được những ý kiến đóng góp của bạn đọc để lần tái bản tới được hoàn thiện hơn.

Mọi ý kiến đóng góp xin gửi về:

- Công ty CP Sách Đại học - Dạy nghề, NXB Giáo dục, 25 Hàn Thuyên - Hà Nội, điện thoại: 04 8264974

- Bộ môn Điện tử công nghiệp, khoa Công nghệ kỹ thuật Điện tử, Trường Đại học Công nghiệp Hà Nội, Minh Khai - Từ Liêm - Hà Nội, điện thoại: 04 7655121.

CÁC TÁC GIẢ

Chương 1 GIỚI THIỆU CHUNG

1.1. CÁC HỆ ĐẾM VÀ BIỂU DIỄN THÔNG TIN TRONG MÁY TÍNH

1.1.1. Các hệ đếm

1. Hệ thập phân

Hàng ngày con người thường dùng hệ đếm cơ số mười (hệ thập phân) để biểu diễn các giá trị số. Trong hệ này, người ta dùng các số trong phạm vi từ 0 đến 9 để biểu diễn các giá trị. Hệ thập phân là một hệ đếm phụ thuộc vị trí, có nghĩa là mỗi chữ số gắn liền với một luỹ thừa 10 với số mũ phụ thuộc vào vị trí của con số đó, trong số được biểu diễn. Ví dụ số 1234 sẽ bằng 1 nghìn, 2 trăm, 3 chục và 4 đơn vị:

$1234 = 1.10^3 + 2.10^2 + 3.10^1 + 4.10^0$

Nhưng trong máy tính, các vi mạch chỉ có thể xử lý thông tin dưới dạng mã nhị phân nên chúng ta cần phải xem xét cách biểu diễn các số dưới dạng mã nhị phân như thế nào?

2. Hệ nhị phân

Trong hệ nhị phân (hay hệ hai), cơ số đếm là 2 nên chỉ sử dụng hai chữ số là 0 và 1 để biểu diễn các giá trị số. Hệ nhị phân cũng là một hệ đếm phụ thuộc vị trí, có nghĩa là mỗi chữ số gắn liền với một luỹ thừa 2 với số mũ phụ thuộc vào vị trí của con số đó, trong số được biểu diễn.

Ví dụ chuỗi số nhị phân 101011 dùng để biểu diễn số:

$101011 = 1.2^5 + 0.2^4 + 1.2^3 + 0.2^2 + 1.2^1 + 1.2^0 = 43$ trong hệ thập phân.

Bảng sau chỉ ra tương quan giữa số thập phân và số nhị phân:

Hệ thập phân	Hệ nhị phân
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
...	...

Mỗi chữ số 0 và 1 được gọi là một bit (viết tắt của binary digit – số nhị phân); một số ở hệ nhị phân gồm các bit được kết thúc bởi chữ B để phân biệt với các hệ khác. Một cụm bốn bit gọi là một nible, một cụm tám bit gọi là một byte, cụm mười sáu bit gọi là một word, cụm ba mươi hai bit gọi là một double word...

Bit đầu tiên bên trái được gọi là bit có ý nghĩa lớn nhất (MSB: most significant bit), còn bit cuối cùng ở bên phải được gọi là bit có ý nghĩa nhỏ nhất (LSB: least significant bit).

3. Hệ mười sáu

Ta thấy rằng một số biểu diễn ở hệ nhị phân thì rất dài và khó nhớ nên trong thực tế người ta thường sử dụng hệ mười sáu. Hệ mười sáu là hệ đếm cơ số 16 nên người ta sử dụng các số từ 0 đến 9 và các chữ cái từ A đến F để biểu diễn các số. Bảng sau chỉ ra tương quan giữa số hệ mười và hệ mười sáu:

Hệ mười	Hệ mười sáu
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10
17	11
...	...

1.1.2. Chuyển đổi giữa các hệ đếm

Vì con người quen sử dụng các số ở hệ thập phân trong khi đó các bộ vi xử lý chỉ thao tác với các số ở hệ nhị phân nên để đảm bảo việc giao tiếp giữa người và máy thì phải có phép chuyển đổi giữa hai hệ này.

**** Chuyển từ hệ nhị phân sang hệ thập phân***

Để đổi một số từ hệ nhị phân sang hệ thập phân ta áp dụng công thức sau:

$(b_n b_{n-1} \ldots b_1 b_0)\mathbf{B} = b_n \times 2^n + b_{n-1} \times 2^{n-1} + \ldots + b_1 \times 2^1 + b_0 \times 2^0$

Ví dụ :

$100011\mathbf{B} = 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 35$

**** Chuyển từ hệ thập phân sang hệ nhị phân***

Để chuyển từ hệ thập phân sang hệ nhị phân ta sử dụng phương pháp đơn giản sau: Lấy số cần chuyển chia cho 2 và ghi nhớ phần dư, tiếp theo lấy thương của phép chia trước đó chia cho 2 và ghi nhớ phần dư... cứ tiếp tục cho đến khi thương bằng 0. Kết quả của phép chuyển đổi, chính là dãy các số dư lấy theo thứ tự đảo ngược.

Ví dụ 1: Đổi số 25 sang hệ nhị phân.

25	1	25 chia 2 được 12 dư 1
12	0	12 chia 2 được 6 dư 0
6	0	6 chia 2 được 3 dư 0
3	1	3 chia 2 được 1 dư 1
1	1	1 chia 2 được 0 dư 1
0		

Vậy kết quả là ***11001B***.

Ví dụ 2: Đổi số 42 sang hệ nhị phân

42	0	42 chia 2 được 21 dư 0
21	1	21 chia 2 được 10 dư 1
10	0	10 chia 2 được 5 dư 0
5	1	5 chia 2 được 2 dư 1
2	0	2 chia 2 được 1 dư 0
1	1	1 chia 2 được 0 dư 1
0		

Vậy kết quả là ***101010B***.

1.1.3. Các phép toán số học với số hệ nhị phân

**** Phép cộng***

Phép cộng với các số hệ nhị phân được thực hiện tương tự như với các số hệ mười theo quy tắc sau:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$ nhớ 1

Ví dụ:

0010	100011
+ 1011	+ 011111
= 1101	= 1000010

**** Phép trừ***

Phép trừ với các số hệ nhị phân được thực hiện tương tự như với các số hệ mười.

Ví dụ:

1111	1000010
− 1011	− 01010
= 0100	= 0111000

Trong máy tính người ta thực hiện phép trừ bằng phép cộng với số bù hai.

**** Phép nhân, phép chia***

Phép nhân và phép chia hai số hệ nhị phân được thực hiện tương tự như nhân và chia hai số hệ mười.

Quy tắc thực hiện phép nhân các số nhị phân như sau:

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

1.1.4. Các mã thường dùng trong máy tính

**** Mã BCD***

Mã BCD thường được sử dụng để mã hoá các chữ số từ 0 đến 9 trong vi xử lý, PLC...

Chữ số của hệ 10	Mã BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Ví dụ mã BCD của 5 là 0101, của 9 là 1001.

Trong khi làm toán với các số BCD ta thường kết hợp hai số BCD tạo thành một byte, dạng biểu diễn này gọi là dạng BCD chuẩn hay BCD đóng gói (packed BCD). Ví dụ mã BCD đóng gói (BCD chuẩn) của 59 là 0101 1001B, của 62 là 0110 0010B.

Ngược lại, số BCD không gói là số dài một byte trong đó bốn bit cao bằng 0 còn bốn bit thấp là số BCD mã hóa số cần biểu diễn.

Ví dụ mã BCD không gói của 5 là 0000 0101B, của 4 là 0000 0100B.

**** Mã ASCII***

Mã ASCII (viết tắt của American Standard Code for Internationnal Interchange – Mã chuẩn của Mỹ dùng cho trao đổi thông tin) là bộ mã rất thông dụng trong máy tính và truyền thông. Trong bảng mã ASCII tiêu chuẩn người ta sử dụng 7 bit để mã hóa các ký tự thông dụng, như vậy bảng này sẽ có 128 ký tự ứng với các mã số từ 0 đến 127.

Trong bảng mã ASCII mở rộng, người ta bổ sung thêm 128 ký tự đặc biệt với mã từ 128 đến 255.

Ví dụ mã ASCII của 'A' là 65 (01000001B), của 'a' là 97 (01100001B).

1.2. MÁY TÍNH VÀ VI XỬ LÝ

1.2.1. Sơ lược về máy tính và vi xử lý

**** Thế hệ máy tính cơ khí***

Ý tưởng về một hệ thống tính toán đã có từ rất lâu, khoảng 500 năm Trước Công nguyên người Babylon đã chế tạo được máy tính đầu tiên có

tên là Abacus. Năm 1643, Blaise Pascal chế tạo thành công một máy tính tạo bởi các bánh răng trong đó số răng của bánh nọ gấp 10 lần số răng của bánh kia, nguyên lý này sau đó đã được sử dụng để tạo các đồng hồ đo quãng đường của motor, đồng hồ đo nước...

**** Thế hệ máy tính cơ điện – điện tử***

Năm 1889, Herman Holerith phát minh ra card đục lỗ dùng để lưu trữ dữ liệu, sau đó ông chế tạo thành công máy tính cơ khí được điều khiển bởi một môtơ điện, nó có thể thực hiện được các phép đếm, sắp xếp và so sánh thông tin lưu trong card đục lỗ.

Năm 1942, nhà phát minh người Đức Konrad Zure chế tạo ra máy tính điện tử Z3 dùng cho không quân Đức. Năm 1943, Alan Turing phát minh ra hệ thống máy tính điện tử có tên là Collossus được thiết kế từ các đèn điện tử chân không, đây là một máy tính chuyên dụng thực hiện theo một chương trình cố định để giải mã các bí mật quân sự của Đức Quốc Xã.

Máy tính điện tử đa dụng đầu tiên – hệ máy tính có thể lập trình được, được phát triển bởi trường Đại học Pensylvania có tên là ENIAC (Electronics Numerical Integrator and Calculator). Đây là một máy tính lớn chứa hơn 17.000 đèn điện tử, nặng khoảng 30 tấn và có thể thực hiện được 100.000 thao tác trong một giây. ENIAC được lập trình bằng cách nối lại mạch điện, công việc này được thực hiện bởi các công nhân và mất rất nhiều thời gian. Ngoài ra, việc bảo dưỡng cũng phải được thực hiện thường xuyên vì tuổi thọ của các đèn điện tử thấp.

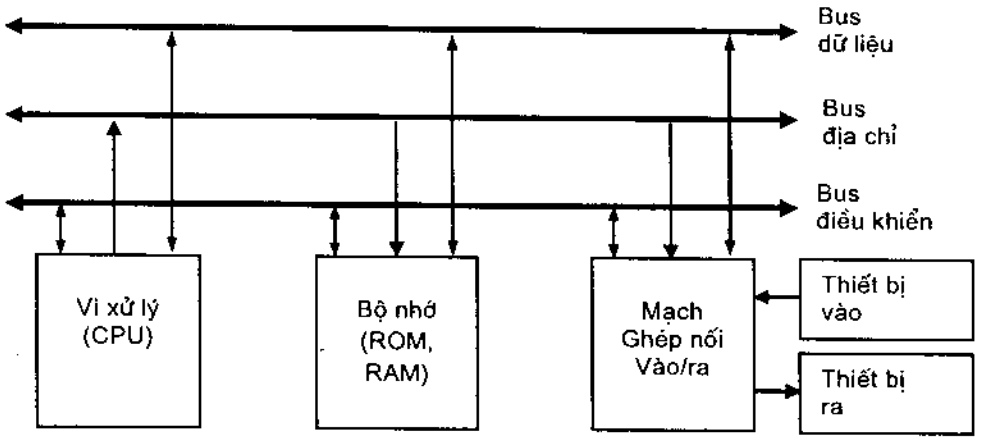
**** Thế hệ máy tính dùng vi xử lý***

Năm 1948, transistor được phát minh. Đến năm 1958, Jack Kilby phát minh ra mạch tổ hợp – đây là cơ sở để phát triển các vi mạch số. Năm 1971, Marcian T. Hoff – một kỹ sư của Intel đã thiết kế ra bộ vi xử lý 4004, mở đầu cho thời kỳ sử dụng vi xử lý trong máy tính. 4004 là bộ vi xử lý 4 bit, bên trong nó gồm 2300 transistor, có thể quản lý được bộ nhớ có 4096 (4k) ô nhớ, mỗi ô gồm 4 bit, tập lệnh của 4004 gồm 45 lệnh khác nhau, nó được chế tạo theo công nghệ MOSFET kênh p, có tốc độ xử lý là 50 KIPS (KIPS: viết tắt của kilo–instruction per second – nghìn lệnh/giây). 4004 được dùng để thiết kế các hệ thống video game, hệ thống điều khiển nhỏ dùng vi xử lý. Tiếp sau 4004, Intel và các hãng sản xuất khác liên tục tung ra thị trường những bộ vi xử lý với tốc độ, hiệu năng càng ngày càng cao.

1.2.2. Cấu trúc của máy tính dùng vi xử lý (hệ vi xử lý)

Vi xử lý là một thành phần cơ bản không thể thiếu của máy tính, ngoài ra để tạo ra một hệ hoàn chỉnh cần phải có các bộ phận khác như bộ nhớ, các thiết bị vào/ra như bàn phím, màn hình...

Hình 1.1 giới thiệu sơ đồ khối tổng quát của một máy tính sử dụng vi xử lý:



Hình 1.1. Sơ đồ khối cấu trúc của máy tính.

Trong sơ đồ này ta thấy một máy tính (hay hệ vi xử lý) bao gồm các khối chức năng sau:

+ Bộ vi xử lý (CPU);

+ Bộ nhớ bán dẫn (ROM, RAM);

+ Mạch ghép nối vào/ra;

+ Bus hệ thống để truyền thông tin giữa các khối, bus hệ thống gồm bus điều khiển, bus địa chỉ và bus dữ liệu.

Sau đây chúng ta sẽ tìm hiểu nguyên lý làm việc của máy tính bằng cách xem xét chức năng của các khối đó.

** Bộ vi xử lý*

Bộ vi xử lý (microprocessor) hay còn được gọi là CPU (viết tắt của Central Processing Unit – đơn vị xử lý trung tâm) đóng vai trò là bộ não của máy tính. Đây là một vi mạch số với mức độ tích hợp cực lớn (VLSI) bên trong nó bao gồm nhiều khối chức năng khác nhau như đơn vị số nguyên để thao tác tính toán với các số nguyên, đơn vị xử lý dấu phảy động để thực hiện các phép tính với số thực... Khi hoạt động nó đọc mã lệnh (mã lệnh được ghi dưới dạng chuỗi các bit 0, 1) từ bộ nhớ chương

trình, đưa vào trong vi xử lý để giải mã thành các vi lệnh, đây là những xung điều khiển, để điều khiển hoạt động của các đơn vị chức năng bên trong vi xử lý.

Các thông số quan trọng của một bộ vi xử lý gồm:

+ Tần số làm việc là tần số xung nhịp (clock) cung cấp cho vi xử lý, tần số này quyết định đến tốc độ làm việc của vi xử lý.

+ Độ rộng bus dữ liệu m là số đường dây dùng để truyền dữ liệu ký hiệu từ D_0 đến D_{m-1}. Các giá trị của m thường là 4, 8, 16, 32 và 64.

+ Độ rộng bus địa chỉ n quyết định đến dung lượng bộ nhớ cực đại mà vi xử lý có thể quản lý được. Một bộ vi xử lý có n đường địa chỉ từ A_0 đến A_{n-1} có thể quản lý được 2^n ô nhớ (mỗi ô nhớ thường là một byte). Các giá trị của n thường là 16, 20 và 32.

**** Bộ nhớ***

Bộ nhớ (hay còn gọi là bộ nhớ trong, bộ nhớ chính) được tạo từ các vi mạch nhớ ROM và RAM là nơi chứa các chương trình cần thực thi.

ROM (Read Only Memory – bộ nhớ chỉ đọc, nội dung bên trong của ROM không bị mất khi mất nguồn nuôi) dùng để chứa các chương trình điều khiển hệ thống như chương trình để kiểm tra các thiết bị mỗi khi bật nguồn, chương trình khởi động máy, các chương trình điều khiển trao đổi tin với các thiết bị ngoại vi như bàn phím, màn hình...

RAM (Random Access Memory – bộ nhớ truy cập ngẫu nhiên) là bộ nhớ có thể ghi/đọc được, có nghĩa là ta có thể đọc thông tin từ bộ nhớ, xóa thông tin cũ trong bộ nhớ hoặc ghi thông tin mới vào bộ nhớ; nội dung thông tin ghi trong bộ nhớ RAM sẽ bị mất khi mất nguồn cung cấp. RAM được dùng để lưu trữ mã lệnh, toán hạng và kết quả của chương trình khi nó đang được thực hiện. Trong máy tính bộ nhớ RAM là các module dạng thanh cắm trên bảng mạch chính của máy, trên mỗi module thường gắn nhiều vi mạch RAM và thường có dung lượng là 1, 2, 4, 8, 16, 32, 64, 128, 256 hoặc 512MB.

**** Mạch ghép nối vào/ra***

Mạch ghép nối vào/ra có nhiệm vụ tạo ra khả năng giao tiếp giữa hệ vi xử lý với thế giới bên ngoài. Các thiết bị vào/ra (hay còn gọi là thiết bị ngoại vi) bao gồm các thiết bị vào (bàn phím, chuột, máy quét...), thiết bị ra (màn hình, máy in, máy vẽ...), các thiết bị lưu trữ (còn gọi là bộ nhớ ngoài) dùng để lưu thông tin với khối lượng lớn như ổ đĩa cứng, ổ đĩa CD, ổ đĩa DVD...

Mạch ghép nối vào/ra có thể là một vi mạch cỡ nhỏ như 8255 để ghép nối song song, 8251 để ghép nối nối tiếp... Tuy nhiên trong các máy tính hiện nay mạch ghép nối vào/ra là những vi mạch cỡ lớn (VLSI) được gọi là chipset, ví dụ chipset 848P của Intel cho phép ghép nối giữa vi xử lý với cổng đồ hoạ tốc độ cao AGP 8x, với ổ đĩa cứng kiểu SATA, với hệ thống âm thanh 5.1, với các thiết bị vào/ra qua cổng USB 2.0...

**** Bus hệ thống***

Bus hệ thống là tập hợp tất cả các đường dây dùng để liên lạc giữa các khối chức năng trong hệ thống. Dựa vào chức năng của các đường dây người ta chia chúng làm 3 nhóm:

+ Bus điều khiển là các đường dây mang các tín hiệu điều khiển hoạt động hoặc phản ánh trạng thái của các khối như /RD (read – đọc bộ nhớ hoặc thiết bị vào), /WR (write – ghi dữ liệu vào bộ nhớ hoặc xuất dữ liệu ra thiết bị ra), INT (interrupt – ngắt vi xử lý để trao đổi dữ liệu)...

+ Bus dữ liệu là các đường dây mang số liệu mà vi xử lý đang trao đổi với bộ nhớ hoặc thiết bị vào/ra.

+ Bus địa chỉ mang thông tin về địa chỉ của ô nhớ hay một thiết bị vào/ra mà vi xử lý đang trao đổi tin. Thông tin về địa chỉ là do vi xử lý phát ra để chọn ra một ô nhớ hoặc một thiết bị vào/ra mà nó cần trao đổi tin.

1.3. VI ĐIỀU KHIỂN

Năm 1976, hãng Intel giới thiệu bộ vi điều khiển 8748 – mở đầu cho họ vi điều khiển MCS–48. 8748 là một vi mạch chứa hơn 17.000 transistor bao gồm một CPU, 1kbyte bộ nhớ EPROM, 64B RAM, một bộ đếm/định thời 8 bit và 27 chân vào/ra. 8748 và các vi điều khiển tiếp theo của nó trong họ MCS–48 đã được sử dụng phổ biến trong các ứng dụng hướng điều khiển như máy giặt, ôtô, các thiết bị ngoại vi của máy tính... Sau 8748, các bộ vi điều khiển mới liên tục được các hãng sản xuất như Intel, Atmel, Siemens... giới thiệu cho các ứng dụng nhúng.

Vi điều khiển (MCU – viết tắt của cụm từ 'Micro Control Unit') có thể được coi như một máy tính thu nhỏ trên một chip, nó có thể hoạt động với một vài linh kiện phụ trợ ở bên ngoài. Vi điều khiển khác với vi xử lý ở những điểm sau:

– Về cấu trúc: Vi xử lý là một CPU trên một chip còn vi điều khiển là một chip có chứa CPU, bộ nhớ, mạch vào/ra và các mạch đặc biệt khác

như bộ đếm/định thời, mạch biến đổi A/D, D/A... Như vậy, về cấu trúc thì vi điều khiển chính là một hệ vi xử lý thu nhỏ.

– Về ứng dụng: Các bộ vi xử lý chủ yếu được dùng làm CPU trong các máy tính còn các bộ vi điều khiển được dùng trong các ứng dụng hướng điều khiển.

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

Như trên đã nói có thể coi vi điều khiển như một máy tính thu nhỏ, tuy nhiên vi điều khiển khác máy tính ở chỗ:

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

– Về bộ nhớ: Máy tính là thiết bị đa dụng nên các chương trình ứng dụng thường được lưu ở các thiết bị lưu trữ ngoài như đĩa cứng, đĩa quang, ổ Flash. Khi cần thực thi, chương trình được nạp vào bộ nhớ RAM của máy tính và vi xử lý sẽ đọc mã lệnh từ bộ nhớ RAM để giải mã lệnh và thực thi. Như vậy, với máy tính thì RAM chính là bộ nhớ chương trình, còn ROM trong máy tính thường dùng để lưu các thông tin về cấu hình của máy và các chương trình vào ra cơ bản (BIOS). Điều này giải thích vì sao trong máy tính RAM có dung lượng lớn hơn ROM nhiều lần. Ngược lại, ở vi điều khiển thì chương trình được chứa trong ROM vì chúng là chương trình điều khiển ứng dụng, hầu như không thay đổi nội dung còn RAM được dùng để chứa số liệu tạm thời cho chương trình như trạng thái của các chân vào/ra, nội dung các biến được khai báo trong chương trình. Do đó ở vi điều khiển thì ROM có dung lượng lớn hơn RAM nhiều lần.

– Về hiệu năng: Do phải tích hợp nhiều thành phần trên một chíp duy nhất nên việc nâng cao hiệu năng cho vi điều khiển khó thực hiện hơn so với máy tính. Do đó với các ứng dụng đòi hỏi hiệu năng cao mà vi điều khiển không đáp ứng được như xử lý ảnh từ camera, quản trị cơ sở dữ liệu... thì người ta phải chọn giải pháp dùng máy tính, thường là máy tính

nhúng (có bộ vi xử lý, các vi mạch ROM, RAM, cổng I/O, đầu cắm thẻ nhớ, ổ đĩa... trên một bản mạch in; hệ điều hành, chương trình điều khiển... được cài đặt trên thẻ nhớ).

1.4. THIẾT KẾ HỆ THỐNG VỚI VI ĐIỀU KHIỂN – ƯU VÀ NHƯỢC ĐIỂM

Trước đây chúng ta đã quen với việc thiết kế các hệ thống số với hàng chục thậm chí hàng trăm vi mạch số. Việc chuyển sang thiết kế sử dụng vi điều khiển cho phép thực hiện những hệ thống tương đương với một vài linh kiện phụ trợ làm cho thời gian phát triển ngắn hơn, độ tin cậy của hệ thống cao hơn, công suất tiêu thụ thấp hơn. Tuy nhiên cũng phát sinh vấn đề mới – đó là tốc độ. Các giải pháp dựa trên vi điều khiển không thể cho tốc độ xử lý tín hiệu nhanh như các giải pháp sử dụng các linh kiện số rời rạc bởi hệ thống dựa trên vi điều khiển thường xuyên phải thực hiện chu trình "đọc – giải mã – thi hành lệnh", trong khi đó với các hệ thống dựa trên các linh kiện rời rạc thì tín hiệu chạy trực tiếp từ đầu ra của phần tử này tới đầu vào của phần tử kia với thời gian không đáng kể. Với hệ thống dựa trên vi điều khiển, muốn cải thiện được tốc độ, ngoài việc tối ưu hóa chương trình điều khiển thì phải cải tiến kiến trúc và tốc độ của vi điều khiển, kết quả là giá thành của hệ thống tăng mạnh. Tuy nhiên với đa số các ứng dụng thì tốc độ không phải là yêu cầu số 1 nên trong thực tế các thiết kế dựa trên vi điều khiển ngày càng trở nên phổ biến.

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

1.5.1. Vi điều khiển của Atmel

Atmel là một hãng cung cấp vi điều khiển lớn, sản phẩm vi điều khiển của Atmel gồm:

– Dòng vi điều khiển dựa trên kiến trúc 8051 của Intel như 83xx, 87xx, 89xx...

– Dòng vi điều khiển AT91CAP như AT91CAP7S250A, AT91CAP7S450A... với tần số hoạt động từ 80 đến 200MHz, 2 đến 4 kênh PWM, 10 kênh ADC 10 bit, ghép nối được với các module SDRAM ngoài.

– Dòng vi điều khiển AT91SAM 32–bit ARM – based với bộ nhớ chương trình có thể tới 2MB, tần số hoạt động đến 240MHz.

– Dòng AVR 8–bit kiến trúc RISC như AT90PWM1, ATmega128, ATmega16, ATmega32...

– Dòng AVR32 32–bit MCU/DSP như AVR 32 UC3A, AVR 32 UC3B... là những bộ vi điều khiển 32 bit có thêm các lệnh xử lý tín hiệu số để xử lý âm thanh, hình ảnh.

– Dòng FPSLIC như AT94K05L, AT94K10L, ATFS40... Đây là sự kết hợp vi điều khiển AVR với mảng cổng logic lập trình FPGA trên một chip rất phù hợp để tạo ra các hệ thống số trên một chip duy nhất.

– Dòng vi điều khiển 4 bit cho các ứng dụng đơn giản MARC4 như ATAM510, ATAR940...

1.5.2. Vi điều khiển của Microchip

– Dòng 8 bit như PIC10, PIC12, PIC14, PIC16, PIC18 với bộ nhớ kiểu Flash, OTP, ROM hoặc ROMless dung lượng từ 0,5 đến 256kbyte.

– Dòng 16 bit như PIC24F, PIC24H.

– Dòng xử lý tín hiệu số 16 bit như dsPIC30Fxxxx, dsPIC33FJxxxx.

1.5.3. Vi điều khiển của Cypress

Cypress nổi tiếng với dòng sản phẩm PSoC, đây là những vi mạch có tích hợp vi điều khiển, các linh kiện tương tự (các bộ khuếch đại, các bộ biến đổi A/D, D/A, các bộ lọc, các bộ so sánh...) và các linh kiện số (bộ định thời, bộ đếm, bộ tạo xung PWM, SPI, UART, I2C...) trên một chip duy nhất. Việc tích hợp hàng trăm khối chức năng cùng với một bộ vi điều khiển trên một chip cho phép giảm thời gian thiết kế, thu gọn kích thước sản phẩm, giảm công suất tiêu thụ và giảm giá thành sản phẩm.

1.5.4. Vi điều khiển của Hitachi

H8 là dòng vi điều khiển được phát triển bởi Hitachi, được sản xuất bởi Renesas Technology. H8 gồm các dòng sản phẩm H8/300, H8/300H, H8/500, H8S (vi điều khiển 16 bit) và H8SX (vi điều khiển 32 bit kiểu CISC). Các vi điều khiển họ H8 được sử dụng rộng rãi trong các sản phẩm dân dụng và công nghiệp như tivi, đầu ghi DVD, camera, PLC, biến tần...

1.5.5. Vi điều khiển của Motorola

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

1.5.6. Vi điều khiển của Maxim

Các sản phẩm vi điều khiển do Maxim cung cấp gồm:

– Vi điều khiển MAXQ 16 bit kiến trúc RISC như MAXQ3212, MAXQ2000.

– Các sản phẩm dựa trên kiến trúc 8051 của Intel như vi điều khiển tích hợp đồng hồ thời gian thực DS87C530, vi điều khiển tích hợp bộ biến đổi A/D 10 bit DS80CH11, vi điều khiển tích hợp giao tiếp mạng Ethernet DS80C400, DS80C430 (rất phù hợp thiết kế IP camera, các trạm đo/điều khiển phân tán AM như DS5250, DS2250, DS2252...).

Chương 2 VI ĐIỀU KHIỂN 8051

2.1. TỔNG QUAN VỀ HỌ VI ĐIỀU KHIỂN 8051

2.1.1. Tóm tắt về lịch sử của 8051

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

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

BẢNG 2.1. CÁC ĐẶC TÍNH CỦA 8051 ĐẦU TIÊN

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

2.1.2. Các thành viên khác của họ 8051

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

1. Bộ vi điều khiển 8052

Bộ vi điều khiển 8052 là một thành viên của họ 8051, 8052 có tất cả các đặc tính chuẩn của 8051 ngoài ra nó có thêm 128 byte RAM và một bộ định thời nữa (bảng 2.2).

BẢNG 2.2. SO SÁNH CÁC ĐẶC TÍNH CỦA CÁC THÀNH VIÊN HỌ 8051

Đặc tính	8051	8052	8031
ROM trên chip	4 kbyte	8 kbyte	không có
RAM	128 byte	256 byte	128 byte
Bộ định thời	2	3	2
Chân vào/ra	32	32	31
Cổng nối tiếp	1	1	1
Nguồn ngắt	6	6	1

Như thấy từ bảng 2.2 thì 8052 là mở rộng của 8051. Do vậy các chương trình viết cho 8051 đều chạy trên 8052 nhưng điều ngược lại là không đúng.

2. Bộ vi điều khiển 8031

Một thành viên khác nữa của 8051 là chíp 8031. Chíp này thường được coi như là 8051 không có ROM trên chíp. Để sử dụng chíp này ta phải bổ sung ROM ngoài cho nó, ROM ngoài phải chứa chương trình mà 8031 sẽ nạp và thực hiện. Với 8051, chương trình được chứa trong ROM trên chíp bị giới hạn bởi 4 kbyte, còn ROM ngoài được gắn vào 8031 thì có thể lớn đến 64 kbyte. Khi sử dụng ROM ngoài ta chỉ còn lại có hai cổng để sử dụng cho mục đích vào/ra, để giải quyết vấn đề này ta có thể mở rộng cổng vào/ra cho 8031 bằng cách sử dụng vi mạch PPI 8255.

2.1.3. Các phiên bản của 8051

Mặc dù 8051 là thành viên phổ biến nhất của họ 8051 nhưng chúng ta sẽ gặp rất nhiều phiên bản của nó với những tên gọi khác nhau tuỳ thuộc vào kiểu bộ nhớ chương trình, công nghệ chế tạo, tần số làm việc... Ví dụ phiên bản của 8051 với bộ nhớ UV – PROM được ký hiệu là 8751. Phiên bản Flash ROM cũng được bán bởi nhiều hãng khác nhau, chẳng hạn của Atmel với tên gọi là AT89C51 còn phiên bản NV – RAM của 8051 do Dalas Semiconductor cung cấp thì được gọi là DS5000. Ngoài ra còn có phiên bản OTP (lập trình được một lần) cũng được sản xuất bởi rất nhiều hãng.

1. Bộ vi điều khiển 8751

Chíp 8751 chỉ có 4kbyte bộ nhớ UV – EPROM trên chíp. Để sử dụng chíp này cần có bộ đốt PROM và bộ xoá UV – EPROM để xoá nội dung của bộ nhớ UV – EPROM bên trong 8751 trước khi ta có thể lập trình lại nó. Do ROM trên chíp đối với 8751 là UV – EPROM nên cần phải mất 20 phút để xoá 8751 trước khi nó có thể được lập trình trở lại. Vì điều này đã dẫn đến nhiều nhà sản xuất giới thiệu các phiên bản

Flash ROM và UV – RAM. Ngoài ra còn có nhiều phiên bản với các tốc độ khác nhau của 8751 từ nhiều hãng khác nhau.

2. Bộ vi điều khiển AT8951 từ Atmel Corporation

AT8951 là phiên bản 8051 có ROM trên chip ở dạng bộ nhớ Flash. Phiên bản này là lý tưởng đối với những phát triển nhanh vì bộ nhớ Flash có thể được xoá trong vài giây. Dùng AT89C51 để phát triển một hệ thống dựa trên bộ vi điều khiển yêu cầu một bộ đốt ROM hỗ trợ bộ nhớ Flash, không yêu cầu bộ xoá ROM. Hãng Atmel đã cho ra đời một phiên bản của AT 89C51 có thể được lập trình qua cổng truyền thông COM của máy tính IBM PC.

BẢNG 2.3. CÁC PHIÊN BẢN CỦA 8051 DO HÃNG ATMEL CUNG CẤP (FLASH ROM)

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

AT89C2051 là bộ vi điều khiển 8 bit được chế tạo theo công nghệ CMOS, có thể hoạt động được ở dải điện áp từ 2,7V đến 6V. Bộ vi điều khiển này được đóng gói DIP 20 chân, khá nhỏ gọn so với AT89S52 nhưng vẫn có đủ các tài nguyên thông dụng như:

– Bộ nhớ: 2 kbyte Flash có thể ghi/xoá được 1000 lần; 128 × 8–bit RAM;
– Có thể hoạt động ở tần số thạch anh lên tới 24MHz;
– 15 chân xuất nhập;
– 2 bộ Timer/Counter 16 bit;
– 6 nguồn ngắt;
– 1 PORT nối tiếp;
– 1 bộ so sánh (Analog Comparator).

AT89C4051 có sơ đồ chân và các tài nguyên giống AT89C2051, ngoại trừ bộ nhớ ROM có dung lượng lớn hơn (4kbyte).

AT89S52 là một bộ vi điều khiển thông dụng, giá rẻ có khá nhiều chức năng hay, đặc biệt là có tích hợp sẵn bộ nạp ISP trên chíp giúp người sử dụng có thể dễ dàng thực hiện các bài thí nghiệm với chi phí rất thấp.

Cũng có những phiên bản ký hiệu thể hiện kiểu đóng vỏ và tốc độ khác nhau của sản phẩm. Xem bảng 2.4, ví dụ, chữ "C" đứng trước số 51 trong AT89C51 – 12PC là ký hiệu cho CMOS, "12" ký hiệu cho

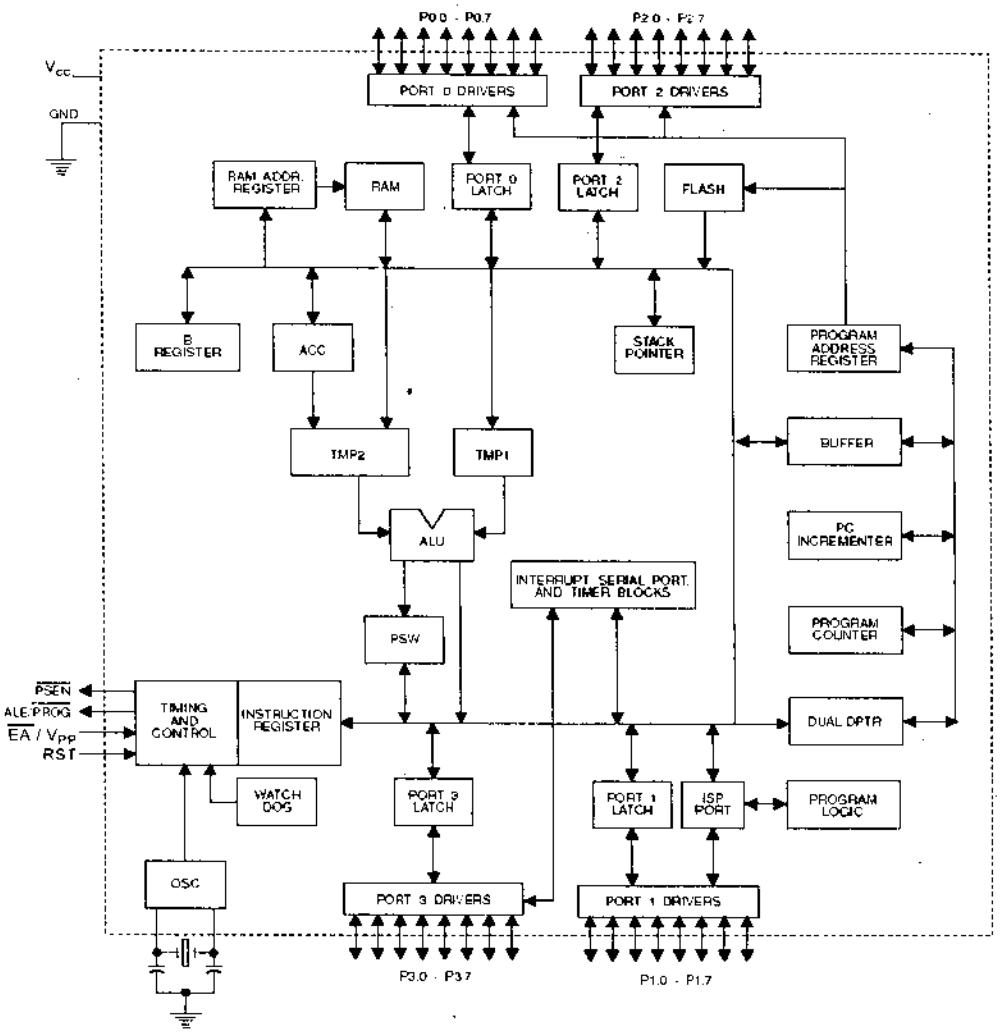
12MHz và "P" là kiểu đóng vỏ DIP và chữ "C" cuối cùng là ký hiệu cho thương mại (ngược với chữ "M" là quân sự). Thông thường AT89C51 – 12PC rất lý tưởng cho các dự án của học sinh, sinh viên.

BẢNG 2.4. CÁC PHIÊN BẢN 8051 VỚI TỐC ĐỘ KHÁC NHAU CỦA ATMEL

Mã linh kiện	Tốc độ	Số chân	Đóng vỏ	Mục đích
AT89C51–12PC	12 MHz	40	DIP	Thương mại

2.2. KIẾN TRÚC PHẦN CỨNG CỦA HỌ 8051

2.2.1. Sơ đồ khối và chức năng các khối của 8051/8052/AT89S52



Hình 2.1. Sơ đồ khối của bộ vi điều khiển AT89S52.

Bộ vi điều khiển AT89S52 gồm các khối chức năng chính sau đây:

• *CPU (Centralprocessing unit) bao gồm :*

– Thanh ghi tích lũy A;

– Thanh ghi tích lũy phụ B, dùng cho phép nhân và phép chia;

– Đơn vị logic học (ALU: Arithmetic Logical Unit);

– Thanh ghi từ trạng thái chương trình (PSW : Prorgam Status Word);

– Bốn băng thanh ghi;

– Con trỏ ngăn xếp.

• *Bộ nhớ chương trình (bộ nhớ ROM) gồm 8kbyte Flash.*

• *Bộ nhớ dữ liệu* (bộ nhớ RAM) gồm 256 byte.

• *Bộ UART* (Universal Ansynchronous Receiver and Transmitter) có chức năng truyền nhận nối tiếp, AT89S52 có thể giao tiếp với cổng nối tiếp của máy tính thông qua bộ UART.

• *3 bộ Timer/Counter* 16 bit thực hiện các chức năng định thời và đếm sự kiện.

• *WDM (Watch Dog Timer):* WDM được dùng để phục hồi lại hoạt động của CPU khi nó bị treo bởi một nguyên nhân nào đó. WDM ở AT89S52 gồm một bộ Timer 14 bit, 1 bộ Timer 7 bit, thanh ghi WDTPRG (WDT programable) điều khiển Timer 7 bit và một thanh ghi chức năng WDTRST (WDM register). Bình thường WDT không hoạt động (bị cấm), để cho phép WDT, các giá trị 1EH và E1H cần phải được ghi liên tiếp vào thanh ghi WDTRST. Timer 14 bit của WDT sẽ đếm tăng dần sau mỗi chu kỳ đồng hồ cho đến giá trị 16383 thì xảy ra tràn. Khi xảy ra tràn, chân Reset sẽ được đặt ở mức cao trong khoảng thời gian $98*T_{osc}$ ($T_{osc} = 1/F_{osc}$) và AT89S52 sẽ được reset. Khi WDT hoạt động, ngoại trừ Reset phần cứng và Reset do WDT tràn thì không có cách nào có thể cấm được WDT, vì vậy khi sử dụng WDT thì các đoạn mã của chương trình phải được đặt trong các khe thời gian giữa các lần WDT được khởi tạo lại.

Thanh ghi WDTPRG:

7	6	5	4	3	2	1	0
–	–	–	–	–	S2	S1	S0

Tuỳ theo các giá trị khác nhau được ghi vào S0, S1, S2 sẽ có số chu kỳ máy mà WDT sẽ đếm cho trong bảng 2.5 và thời gian tràn (time–out) của WDT trong bảng 2.6.

BẢNG 2.5. SỐ CHU KỲ MÁY WDT ĐẾM TÙY THEO GIÁ TRỊ CỦA S0, S1, S2

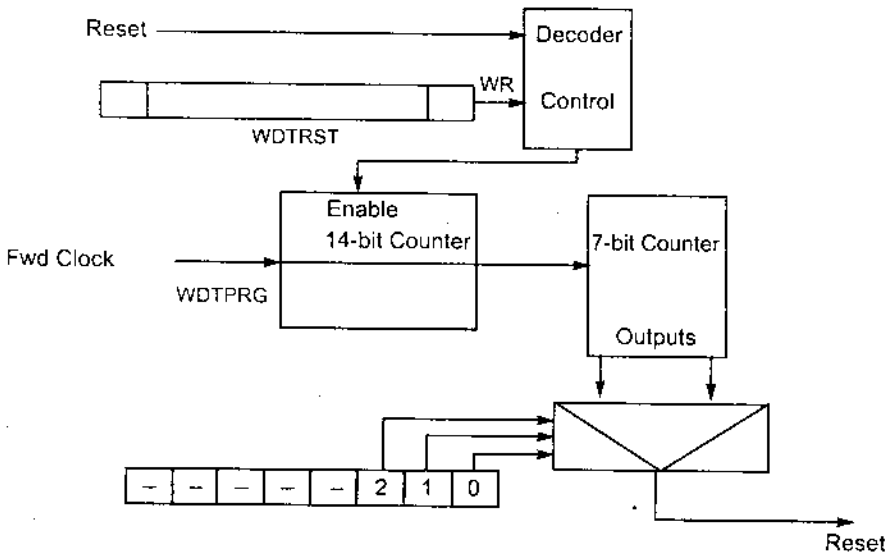
S2	S1	S0	Machine Cyle Count
0	0	0	2^{14}
0	0	1	2^{15}
0	1	0	2^{16}
0	1	1	2^{17}
1	0	0	2^{18}
1	0	1	2^{19}
1	1	0	2^{20}
1	1	1	2^{21}

BẢNG 2.6. THỜI GIAN TRÀN CỦA WDT

S2	S1	S0	F_{osc} = 12 MHz	F_{osc} = 16 MHz	F_{osc} = 20 MHz
0	0	0	16,38 ms	12,28 ms	9,82 ms
0	0	1	32,77 ms	24,57 ms	19,66 ms
0	1	0	65,54 ms	49,14 ms	39,32 ms
0	1	1	131,01 ms	98,28 ms	76,64 ms
1	0	0	262,14 ms	196,56 ms	157,28 ms
1	0	1	524,29 ms	393,12 ms	314,56 ms
1	1	0	1,45 s	788,24 ms	629,17 ms
1	1	1	2,10 s	1,57s	1,25 s

Machine cycle count: Số chu kỳ máy WDT đếm.

F_{osc}: Tần số dao động của thạch anh.



Hình 2.2. WDM trên AT89S52.

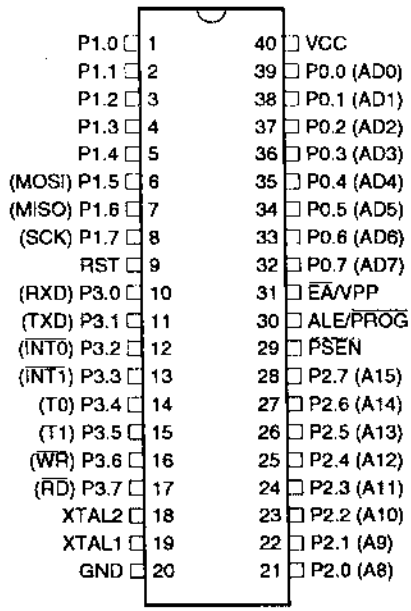
• *Khối điều khiển ngắt* với 2 nguồn ngắt ngoài và 4 nguồn ngắt trong.

• *Bộ lập trình* (ghi chương trình lên Flash ROM) cho phép người sử dụng có thể nạp các chương trình cho chip mà không cần các bộ nạp chuyên dụng.

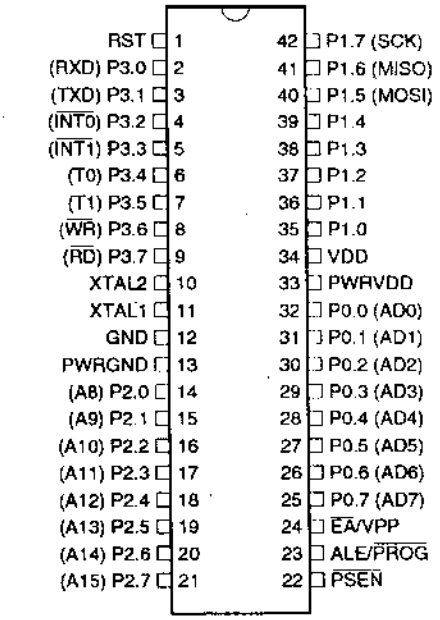
• *Bộ chia tần số* với hệ số chia là 12.

• *4 cổng xuất nhập* với 32 chân.

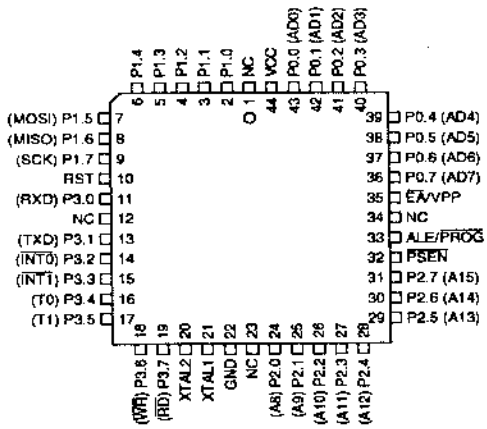
2.2.2. Sơ đồ chân, chức năng các chân của họ 8051



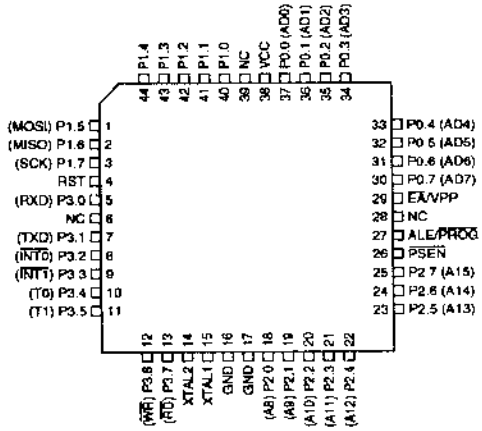
a) 89C51/89S51 DIP 40 chân;



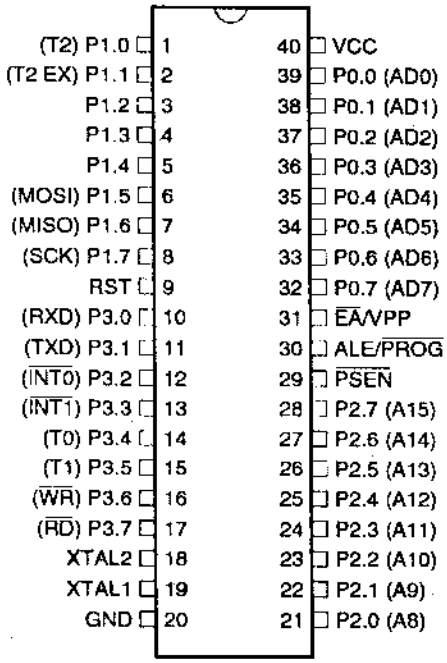
b) 89C51/89S51 kiểu DIP 42 chân;



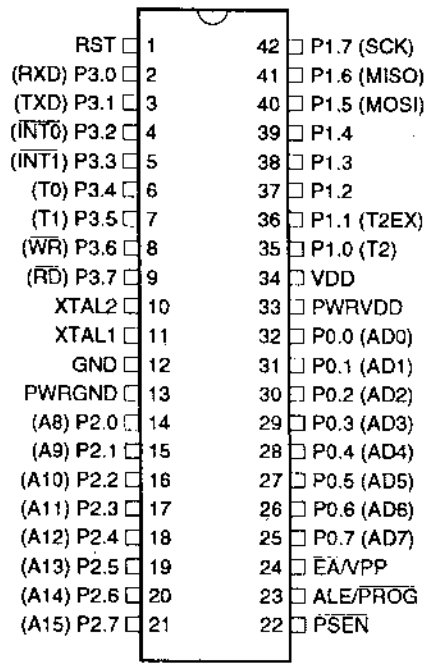
c) 89C51/89S51 kiểu PLCC;



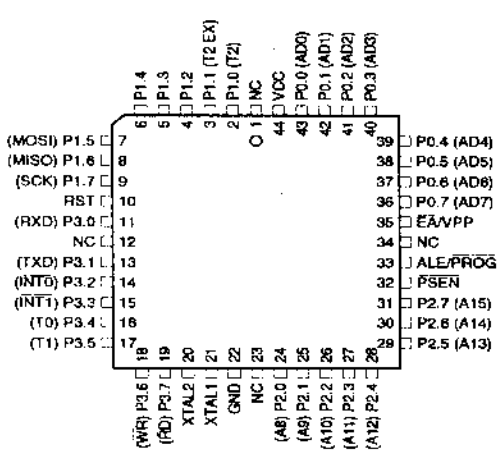
d) 89C51/89S51 kiểu PQFP/TQFP;



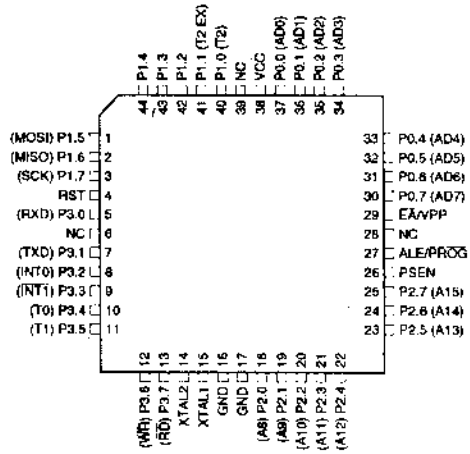
e) 89C52/89S52 kiểu DIP 40 chân;



f) 89C52/89S52 kiểu DIP 42 chân;



g) 89C52/89S52 kiểu PLCC;

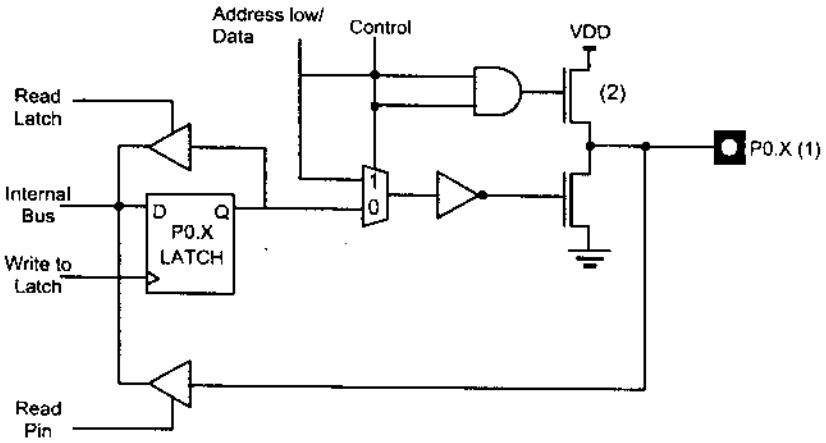


h) 89C52/89S52 kiểu PQFP/TQFP.

Hình 2.3. Sơ đồ chân của 89C51/89S51, 89C52/89S52.

1. Port 0 (P0.0 – P0.7)

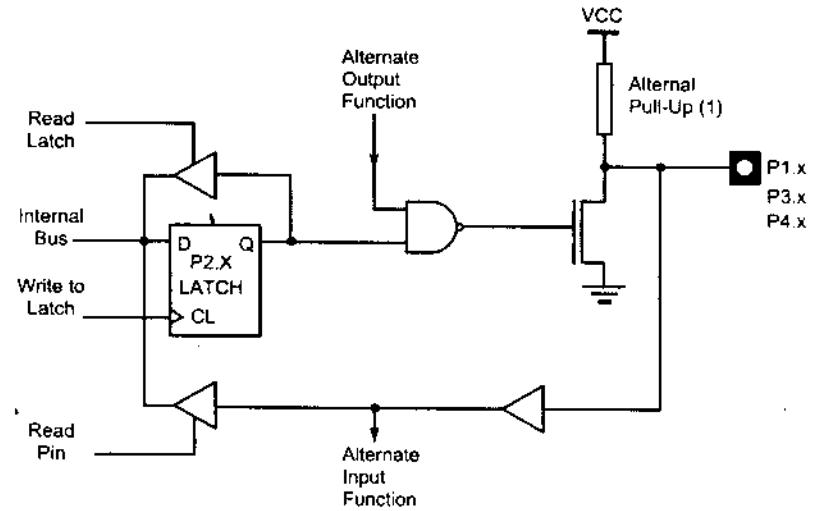
Port 0 gồm 8 chân, ngoài chức năng xuất nhập, Port 0 còn là bus đa hợp dữ liệu và địa chỉ (AD0–AD7), chức năng này sẽ được sử dụng khi 8051 giao tiếp với các thiết bị ngoài có kiến trúc Bus như các vi mạch nhớ, mạch PIO...



Hình 2.4. Cấu trúc của các chân trên Port 0.

2. Port 1 (P1.0 – P1.7)

Đối với 8051, chức năng duy nhất của Port 1 là chức năng xuất nhập, cũng như các Port khác, Port 1 có thể xuất nhập theo bit và theo byte.



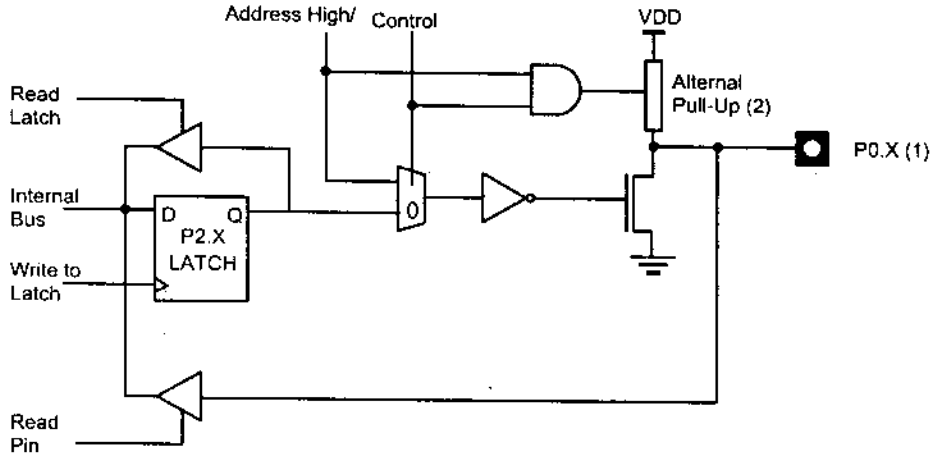
Hình 2.5. Cấu trúc của các chân trên Port 1 và Port 3.

Riêng dòng 89Sxx, ba chân P1.5, P1.6, P1.7 được dùng để nạp ROM theo chuẩn ISP; hai chân P1.0 và P1.1 được dùng cho bộ Timer 2. Sơ đồ kết nối AT89S52 với cổng song song để nạp chương trình (file mã – *.hex) từ máy tính được đưa ra trong phần Phụ lục.

3. Port 2

Port 2 ngoài chức năng là cổng vào/ra như port 0 và port 1 còn là byte cao của bus địa chỉ khi sử dụng bộ nhớ ngoài.

Cấu trúc các chân của port 2 được vẽ trên hình 2.6.



Hình 2.6. Cấu trúc của các chân trên Port 2.

4. Port 3

Mỗi chân trên Port 3 ngoài chức năng xuất nhập còn có một chức năng riêng, cụ thể như sau:

Bit	Tên	Chức năng
P3.0	RXD	Dữ liệu nhận cho Port nối tiếp.
P3.1	TXD	Dữ liệu truyền cho Port nối tiếp.
P3.2	INT0	Ngắt bên ngoài 0.
P3.3	INT1	Ngắt bên ngoài 1.
P3.4	T0	Ngõ vào của Timer/counter 0.
P3.5	T1	Ngõ vào của Timer/counter 1.
P3.6	/WR	Xung ghi bộ nhớ dữ liệu ngoài.
P3.7	/RD	Xung đọc bộ nhớ dữ liệu ngoài.

5. Chân /PSEN (Program Store Enable)

/PSEN là chân điều khiển đọc chương trình ở bộ nhớ ngoài, nó được nối với chân /OE để cho phép đọc các byte mã lệnh trên ROM ngoài. /PSEN sẽ ở mức thấp trong thời gian đọc mã lệnh. Mã lệnh được đọc từ bộ nhớ ngoài qua bus dữ liệu (port 0) thanh ghi lệnh để được giải mã.

Khi thực hiện chương trình trong ROM nội thì /PSEN ở mức cao.

6. Chân ALE (Address Latch Enable)

ALE là tín hiệu điều khiển chốt địa chỉ có tần số bằng 1/6 tần số dao động của vi điều khiển. Tín hiệu ALE được dùng để cho phép vi

mạch chốt bên ngoài như 74373, 74573 chốt byte địa chỉ thấp ra khỏi bus đa hợp địa chỉ/dữ liệu (Port 0).

7. Chân /EA (External Access)

Tín hiệu /EA cho phép chọn bộ nhớ chương trình là bộ nhớ trong hay ngoài vi điều khiển. Nếu /EA ở mức cao (nối với Vcc), thì vi điều khiển thi hành chương trình trong ROM nội. Nếu /EA ở mức thấp (nối GND) thì vi điều khiển thi hành chương trình từ bộ nhớ ngoài.

8. RST (Reset)

Ngõ vào RST trên chân 9 là ngõ reset của 8051. Khi tín hiệu này được đưa lên mức cao (trong ít nhất 2 chu kỳ máy), các thanh ghi trong bộ vi điều khiển được tải những giá trị thích hợp để khởi động hệ thống.

9. XTAL1, XTAL2

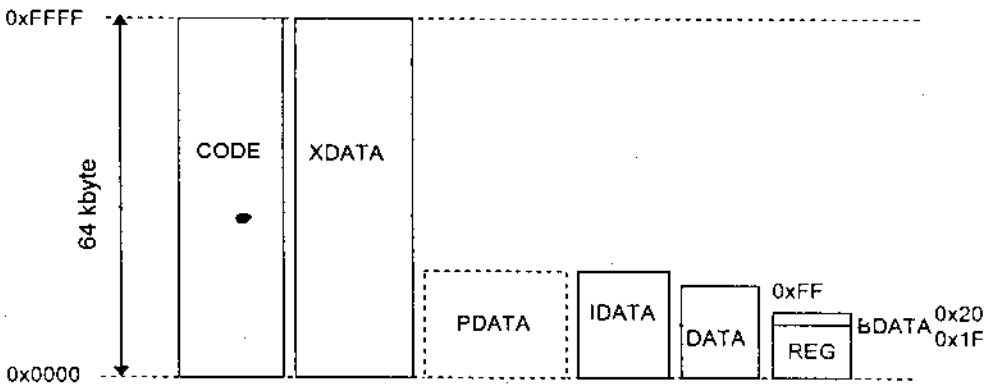
AT89S52 có một bộ dao động trên chip, nó thường được nối với bộ dao động thạch anh có tần số lớn nhất là 33MHz, thông thường là 12MHz.

10. Vcc, GND

AT89S52 dùng nguồn một chiều có dải điện áp từ 4V đến 5,5V được cấp qua chân 40 và 20.

2.2.3. Tổ chức bộ nhớ

AT89S52 có bộ nhớ theo cấu trúc Harvard – có những vùng cho bộ nhớ riêng biệt cho chương trình và dữ liệu. Như đã nói ở trên, cả bộ nhớ chương trình và dữ liệu có sẵn ở trên chip, tuy nhiên dung lượng của các bộ nhớ trên chip là hạn chế. Khi thiết kế các ứng dụng đòi hỏi bộ nhớ lớn người ta có thể dùng bộ nhớ ngoài với dung lượng lên tới 64 kbyte cho bộ nhớ chương trình và cho 64 kbyte bộ nhớ dữ liệu (hình 2.7).

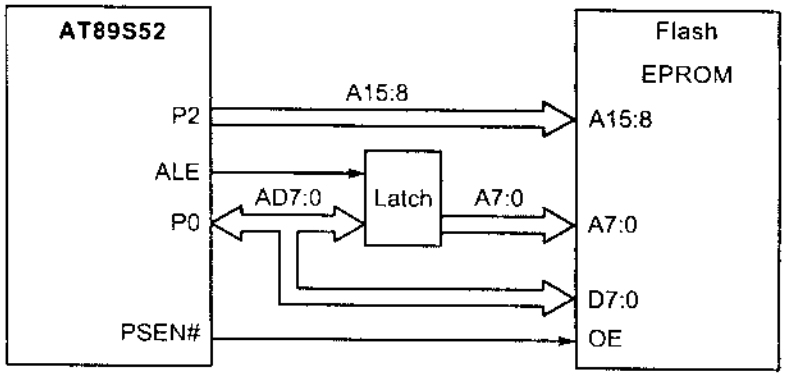


Hình 2.7. Tổ chức bộ nhớ của AT89S52.

1. Bộ nhớ chương trình

AT89S52 có 8 kbyte Flash ROM trên chip, khi chân /EA (chân số 31) được đặt ở mức logic cao (+5V), bộ vi điều khiển sẽ thực hiện chương trình trong bộ nhớ này bắt đầu từ địa chỉ 0000H. Số lần lập trình (ghi) cho bộ nhớ này là khoảng 1000 lần.

Khi chân /EA được đặt ở mức logic thấp, bộ vi điều khiển sẽ thực hiện chương trình ở bộ nhớ chương trình ngoài (EPPOM ngoài), tuy nhiên để có được điều này thì cần phải có một mạch phối ghép AT89S52 với Flash/EPROM như trong hình 2.8. Vi mạch chốt (Latch) sẽ tách riêng Bus đa hợp địa chỉ và dữ liệu AD0–AD7 trên port 0 của 8951, tuỳ theo dung lượng của EPROM sẽ có số đường địa chỉ tương ứng được dùng. Tín hiệu điều khiển đọc ROM là tín hiệu /PSEN.



Hình 2.8. Sơ đồ ghép nối AT89S52 với EPROM.

2. Bộ nhớ dữ liệu

AT89S52 có 256 byte RAM nội (bảng 2.7) được phân chia như sau:

* *Các bank thanh ghi có địa chỉ từ 00H đến 1FH.*

32 byte thấp của bộ nhớ nội được dành cho các bank thanh ghi. Bộ lệnh 8951 hỗ trợ 8 thanh ghi có tên là R0 – R7 và theo mặc định sau khi reset hệ thống, các thanh ghi này có các địa chỉ từ 00H – 07H.

Các lệnh dùng các thanh ghi R0 – R7 sẽ ngắn hơn và nhanh hơn so với các lệnh có chức năng tương ứng dùng kiểu địa chỉ trực tiếp. Các dữ liệu được dùng thường xuyên nên sử dụng một trong các thanh ghi này.

Do có 4 bank thanh ghi nên tại một thời điểm chỉ có một bank thanh ghi được truy xuất bởi các thanh ghi R0 – R7, để chuyển đổi việc truy xuất các bank thanh ghi ta phải thay đổi các bit chọn bank trong thanh ghi trạng thái.

** RAM địa chỉ hoá từng bit có địa chỉ từ 20H đến 2FH.*

AT89S52 có 128 bit có chứa các byte định địa chỉ theo bit từ 20H đến 2FH. Ý tưởng truy xuất từng bit bằng phần mềm là các đặc tính mạnh của các bộ vi điều khiển nói chung. Các bit có thể được đặt, xóa (AND, OR)... với 1 lệnh đơn.

** RAM đa dụng từ 30H đến FFH.*

** Các thanh ghi chức năng đặc biệt từ 80H đến FFH.*

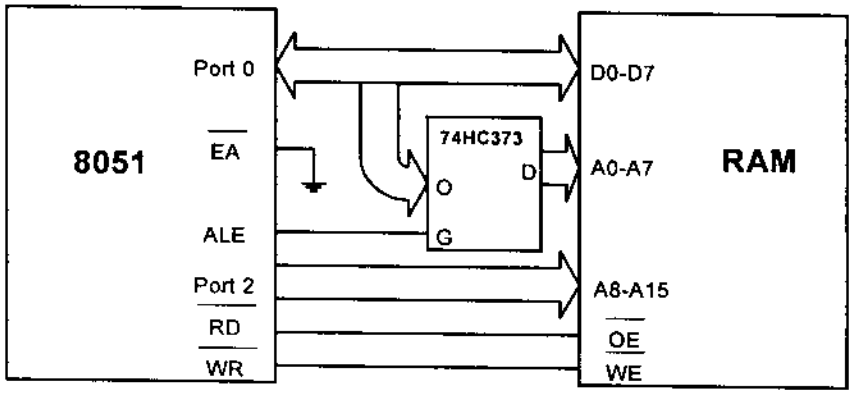
BẢNG 2.7. BỘ NHỚ DỮ LIỆU TRÊN CHIP CỦA AT89S52

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

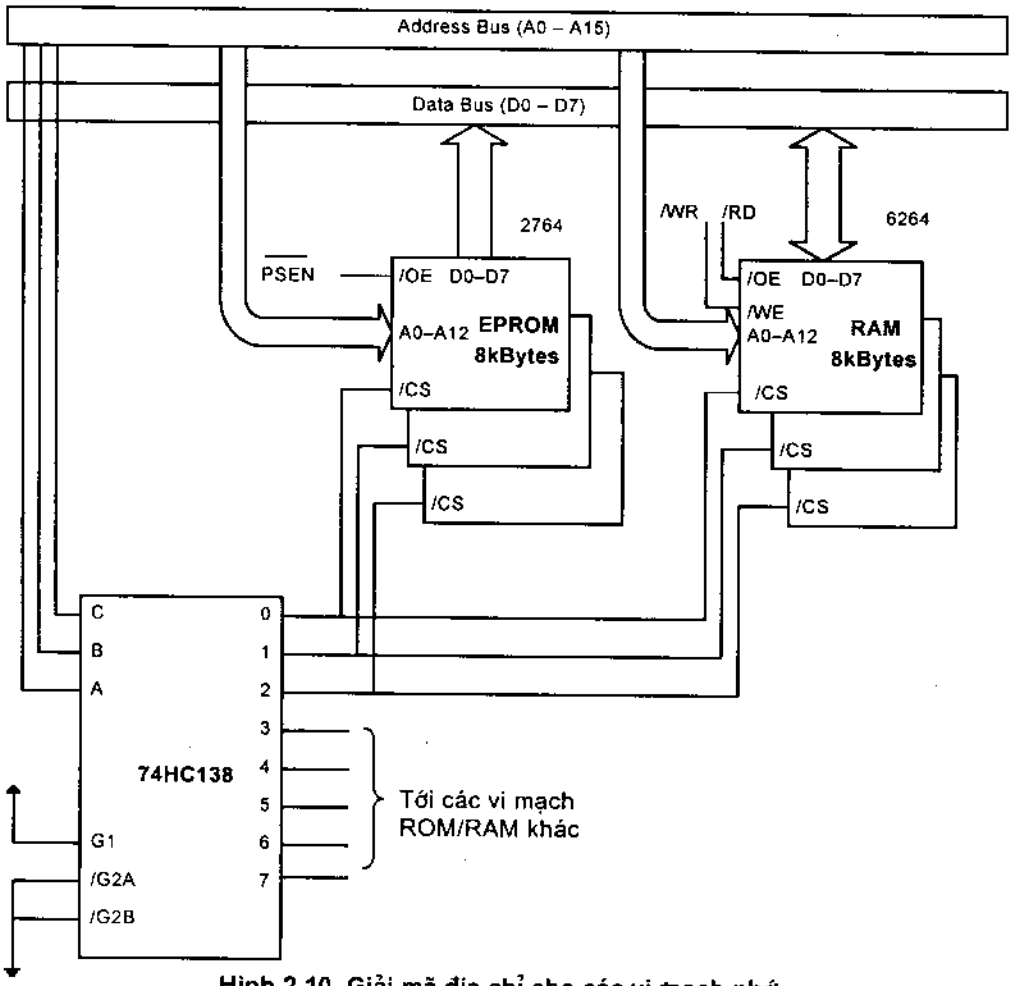
FF

80

Dành cho các thanh ghi đặc biệt (SFR)



Hình 2.9. Sơ đồ ghép nối 8051 với RAM ngoài.

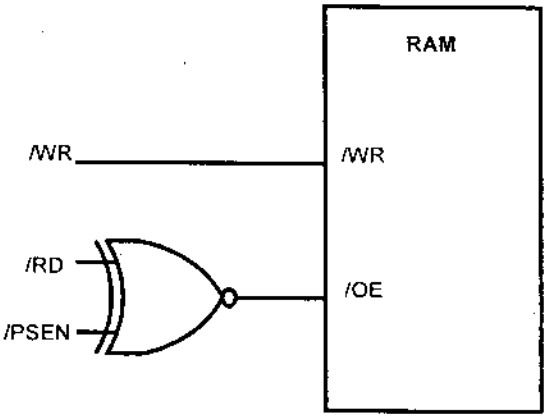


Hình 2.10. Giải mã địa chỉ cho các vi mạch nhớ.

Bộ nhớ dữ liệu ngoài là bộ nhớ RAM được đọc hoặc ghi bởi tín hiệu /RD và /WR. Các RAM có thể giao tiếp với AT89S52 tương tự cách thức như EPROM ngoại trừ chân /RD của AT89S52 nối với chân /OE (Output Enable) của RAM và chân /WR của AT89S52 nối với chân /WE của RAM (hình 2.9).

Nếu có nhiều vi mạch ROM và RAM cùng được ghép nối với AT89S52 thì có thể dùng thêm vi mạch giải mã 74LS138 (hình 2.10).

Như đã nói ở trên, bộ nhớ chương trình và bộ nhớ dữ liệu của AT89S52 có thể trùng địa chỉ, điều này cho phép người thiết kế có thể xây dựng một bộ nhớ dữ liệu chứa chương trình thực thi (bộ nhớ dữ liệu đọc như bộ nhớ chương trình) như hình 2.11.



Hình 2.11. Bộ nhớ dữ liệu đọc như bộ nhớ chương trình.

2.2.4. Các thanh ghi chức năng

1. Từ trạng thái chương trình (PSW: Program Status Word)

Từ trạng thái chương trình ở địa chỉ D0H được tóm tắt như sau:

Bit	Ký hiệu	Địa chỉ (bit)	Mô tả
PSW.7	CY	D7H	Cary Flag
PSW.6	AC	D6H	Auxiliary Cary Flag
PSW.5	F0	D5H	Flag 0
PSW.4	RS1	D4H	Register Bank Select 1
PSW.3	RS0	D3H	Register Bank Select 0
			00 = Bank 0; address 00H÷07H
			01 = Bank 1; address 08H÷0FH
			10 = Bank 2; address 10H÷17H
			11 = Bank 3; address 18H÷1FH
PSW.2	OV	D2H	Overlow Flag
PSW.1	–	D1H	Reserved
PSW.0	P	DOH	Even Parity Flag

BẢNG 2.8. ĐỊA CHỈ VÀ GIÁ TRỊ KHI RESET CỦA CÁC THANH GHI CHỨC NĂNG CỦA AT89S52

0F8H									0FFH
0F0H	B 00000000								0F7H
0E8H									0EFH
0E0H	ACC 00000000								0E7H
0D8H									0DFH
0D0H	PSW 00000000								0D7H
0C8H	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			0CFH
0C0H									0C7H
0B8H	IP XX000000								0BFH
0B0H	P3 11111111								0B7H
0A8H	IE 0X000000								0AFH
0A0H	P2 11111111		AUXR1 XXXXXXX0				WDTRST XXXXXXXX		0A7H
98H	SCON 00000000	SBUF XXXXXXXX							9FH
90H	P1 11111111								97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XXX00XX0		8FH
80H	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000		PCON 0XXX0000	87H

Chức năng từng bit của từ trạng thái chương trình:

– Cờ nhớ CY (Carry Flag): Cờ nhớ có tác dụng kép. Thông thường nó được dùng cho các lệnh toán học: C = 1 nếu phép toán cộng có sự tràn hoặc phép trừ có mượn và ngược lại C = 0 nếu phép toán cộng không tràn và phép trừ không có mượn.

– Cờ nhớ phụ AC (Auxiliary Carry Flag):

Khi cộng những giá trị BCD (Binary Code Decimal), cờ nhớ phụ AC được thiết lập nếu kết quả 4 bit thấp nằm trong phạm vi điều khiển 0AH – 0FH. Ngược lại AC = 0.

– Cờ 0 (Flag 0): Cờ 0 (F0) là 1 bit cờ đa dụng dùng cho các ứng dụng của người dùng.

– Bit chọn bank thanh ghi truy xuất RS0, RS1:

RS1 và RS0 quyết định dãy thanh ghi tích cực. Chúng được xóa sau khi reset hệ thống và được thay đổi bởi phần mềm khi cần thiết.

Tùy theo RS1, RS0 = 00, 01, 10, 11 sẽ được chọn Bank tích cực tương ứng là Bank0, Bank1, Bank2, Bank3.

RS1	RS0	BANK
0	0	0
0	1	1
1	0	2
1	1	3

– Cờ tràn OV (Over Flag):

Cờ tràn được thiết lập sau một hoạt động cộng hoặc trừ nếu có sự tràn toán học. Khi các số có dấu được cộng hoặc trừ với nhau, phần mềm có thể kiểm tra bit này để xác định xem kết quả có nằm trong tầm xác định không. Khi các số không có dấu được cộng, bit OV được bỏ qua. Các kết quả lớn hơn +127 hoặc nhỏ hơn –128 thì bit OV = 1.

– Bit Parity (P):

Bit tự động được thiết lập hay xóa ở mỗi chu kỳ máy để lập bit chẵn lẻ (Parity) chẵn với thanh ghi A. Sự đếm các bit 1 trong thanh ghi A cộng với bit chẵn lẻ luôn luôn chẵn. Ví dụ A chứa 10101101B thì bit P đặt lên 1 để tổng số bit 1 trong A và P là chẵn.

Bit chẵn lẻ thường được dùng trong sự kết hợp với những thủ tục của Port nối tiếp để tạo ra bit chẵn lẻ trước khi phát đi hoặc kiểm tra bit chẵn lẻ sau khi thu.

2. Thanh ghi B

Thanh ghi B ở địa chỉ F0H được dùng cùng với thanh ghi A trong các phép toán nhân chia. Lệnh MUL A B sẽ nhận những giá trị không dấu 8 bit trong hai thanh ghi A và B, rồi trả về kết quả 16 bit trong A (byte cao) và B (byte thấp). Lệnh DIV A B lấy A chia B, kết quả phần nguyên của phép chia đặt vào A, phần dư đặt vào B.

Thanh ghi B cũng có thể được dùng như một thanh ghi trung gian, thanh ghi này được định địa chỉ theo bit từ F0H đến F7H.

3. Con trỏ ngăn xếp SP (Stack Pointer)

Con trỏ ngăn xếp là một thanh ghi 8 bit ở địa chỉ 81H dùng để chứa địa chỉ của đỉnh ngăn xếp. Các lệnh thao tác với ngăn xếp bao gồm các lệnh cất dữ liệu vào ngăn xếp (PUSH) và lấy dữ liệu ra khỏi ngăn xếp (POP). Lệnh cất dữ liệu vào ngăn xếp sẽ làm tăng SP và lệnh lấy ra khỏi ngăn xếp sẽ làm giảm SP. Ngăn xếp của AT89S52 là 128 byte đầu của RAM nội.

4. Con trỏ dữ liệu DPTR (Data Pointer)

Con trỏ dữ liệu (DPTR) được dùng để truy xuất bộ nhớ ngoài. Con trỏ dữ liệu là một thanh ghi 16 bit ở địa chỉ 82H (DPL: byte thấp) và 83H (DPH: byte cao).

Ví dụ: 03 lệnh sau sẽ ghi 12H vào RAM ngoài ở địa chỉ 1200H:

```
MOV A,      #12H
MOV DPTR,   #1200H
MOV @DPTR,  A
```

5. Thanh ghi của các cổng (Port Register)

Các thanh ghi của các Port của AT89S52 bao gồm thanh ghi của Port 0 ở địa chỉ 80H, thanh ghi của Port 1 ở địa chỉ 90H, thanh ghi của Port 2 ở địa chỉ A0H, và thanh ghi của Port 3 ở địa chỉ B0H. Tất cả các thanh ghi của các Port này đều có thể truy xuất theo bit và theo byte.

6. Thanh ghi của các bộ định thời (Timer Register)

AT89S52 có chứa ba bộ đếm/định thời (Timer/Counter) 16 bit được dùng cho việc định thời hoặc đếm sự kiện. Bộ định thời 0 (Timer 0) ở địa chỉ 8AH (TLO: byte thấp) và 8CH (THO: byte cao); Timer 1 ở địa chỉ 8BH (TL1: byte thấp) và 8DH (TH1: byte cao) Timer 2 ở địa chỉ CCH (TL2: byte thấp) và CDH (TH2: byte cao). Các thanh ghi này không được định địa chỉ bit.

7. Thanh ghi của cổng nối tiếp (Serial Port Register)

AT89S52 chứa một Port nối tiếp phục vụ cho việc trao đổi thông tin với các thiết bị có khả năng giao tiếp nối tiếp như máy tính (qua cổng com), modem hoặc các bộ vi điều khiển khác có cùng chức năng. Thanh ghi đệm dữ liệu nối tiếp (SBUF) ở địa chỉ 99H làm cả hai nhiệm vụ truyền và nhận dữ liệu. Khi truyền dữ liệu được ghi lên SBUF, khi nhận dữ liệu được đọc từ SBUF. Thanh ghi này cũng không được định địa chỉ bit.

8. Thanh ghi ngắt (Interrupt Register)

AT89S52 có 6 nguồn ngắt, 2 mức ưu tiên. Các ngắt bị cấm sau khi bị reset hệ thống và sẽ được cho phép bằng việc thiết lập các bit của thanh ghi cho phép ngắt (IE) ở địa chỉ A8H, thứ tự ưu tiên của các ngắt được đặt bằng cách set các bit ở thanh ghi ưu tiên ngắt (IP) ở địa chỉ B8H. Cả hai thanh ghi này đều được định địa chỉ theo bit.

9. Thanh ghi điều khiển nguồn PCON (Power Control Register)

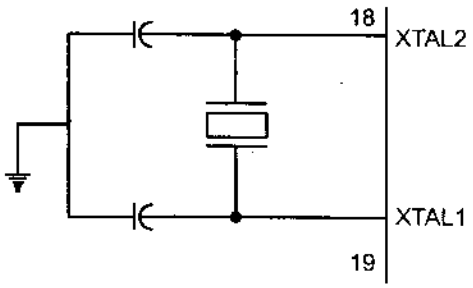
PCON gồm 8 bit nằm ở địa chỉ 87H. Thanh ghi này không được định địa chỉ theo bit, ý nghĩa của từ bit như sau:

Bit 7 (SMOD)	: Bit tăng tốc độ baud ở mode 1, 2, 3 của cổng nối tiếp.
Bit 6, 5, 4	: Không sử dụng.
Bit 3 (GF1)	: Bit cờ đa năng 1.
Bit 2 (GF0)	: Bit cờ đa năng 2.
Bit 1 (PD)	: Bit khởi tạo chế độ Power Down.
Bit 0 (IDL)	: Bit khởi tạo chế độ IDLE.

2.2.5. Mạch tạo dao động và Reset

1. Mạch tạo dao động

AT89S52 có một bộ chia tần số bên trong chip, bộ này sẽ cấp xung clock cho các khối trên chip từ nguồn dao động bên ngoài qua 2 chân XTAL1 và XTAL2 (hình 2.12).



Hình 2.12. Mạch tạo dao động.

Bộ chia tần số có thể hoạt động ở 2 chế độ:

Chế độ X1 (chế độ mặc định):

Ở chế độ này tần số thạch anh được chia 12 lần, nghĩa là nếu một lệnh thực hiện trong 1 chu kỳ máy và tần số thạch anh là 12MHz thì thời gian thực hiện lệnh đó sẽ là 1 (µs).

Chế độ X2:

Ở chế độ này tần số thạch anh được chia 6 lần, chế độ này được đặt bằng cách đặt các bit ở thanh ghi CLKCON0 và thanh ghi CLKCON1.

Thanh ghi CLKCON0:

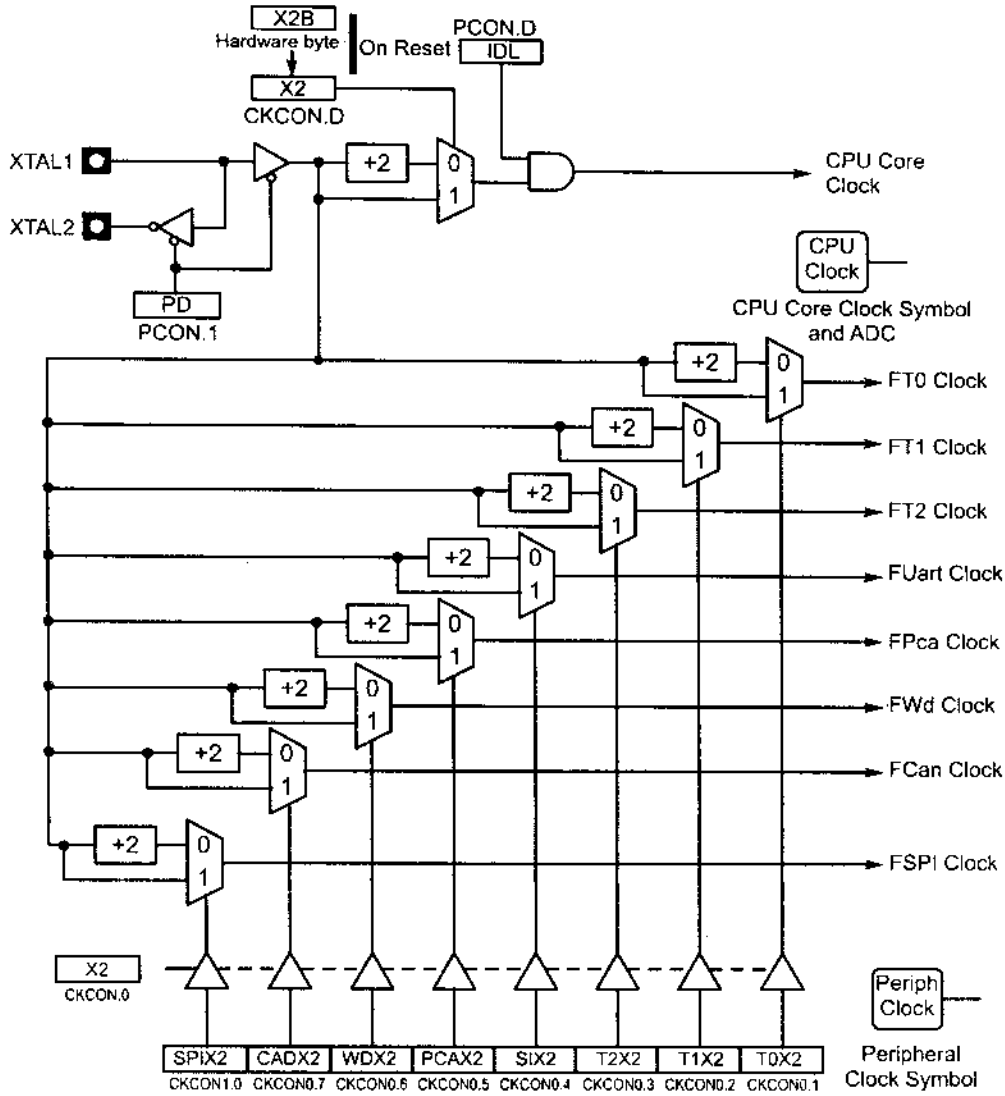
7	6	5	4	3	2	1	0
CANX2	WDX2	PCAX2	SIX2	T2X2	T1X2	T0X2	X2

Bit	Ký hiệu	Mô tả
CLKCON07	CANX2	
CLKCON06	WDX2	Cho phép đặt hệ số chia cho watchdog Timer: 0: hệ số chia là 6 1: hệ số chia là 12.
CLKCON05	PCAX2	
CLKCON04	SIX2	Cho phép đặt hệ số chia Port nối tiếp (mode 0 và mode 2): 0: hệ số chia là 6 1: hệ số chia là 12.
CLKCON03	T2X2	Cho phép đặt hệ số chia cho Timer 2: 0: hệ số chia là 6 1: hệ số chia là 12.
CLKCON02	T1X2	Cho phép đặt hệ số chia cho Timer 1: 0: hệ số chia là 6 1: hệ số chia là 12.
CLKCON01	T0X2	Cho phép đặt hệ số chia cho Timer 0: 0: hệ số chia là 6 1: hệ số chia là 12.
CLKCON00	X2	Cho phép đặt hệ số chia cho CPU: 0: hệ số chia là 12 (chế độ X1) 1: hệ số chia là 6 (chế độ X2).

Thanh ghi CLKCON1:

x	x	x	x	x	x	X	SPIX2

Bit	Ký hiệu	Mô tả
CLKCON10	SPIX2	Cho phép đặt hệ số chia cho tần số xung clock khi truy xuất ngoại vi: 0: hệ số chia là 12 (chế độ X1) 1: hệ số chia là 6 (chế độ X2)
	x	Không sử dụng.

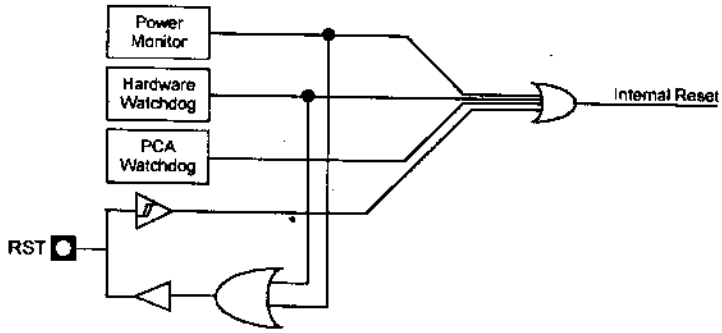


Hình 2.13. Mô tả hoạt động của bộ chia tần.

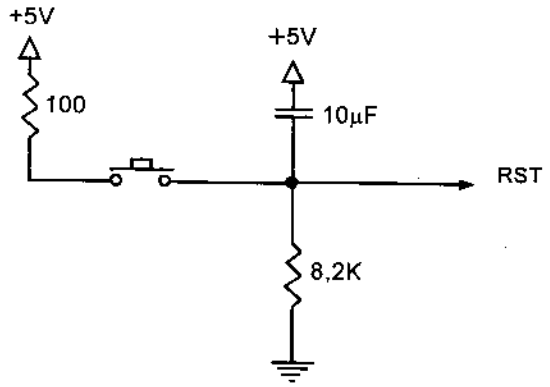
2. Mạch RESET

Có 4 cách để RESET AT89S52 lần lượt là: RESET khi cấp nguồn, RESET bởi WDT, RESET bằng phần mềm và RESET bằng mạch ngoài qua chân RST (hình 2.14).

Trong một hệ thống gồm nhiều vi mạch khả trình thì một mạch RESET tích hợp cả 2 cách RESET khi bật nguồn và RESET bởi mạch ngoài thường được sử dụng (hình 2.15).



Hình 2.14. Reset AT89S52.



Hình 2.15. Mạch reset AT89S52.

2.2.6. Lập trình cho vi điều khiển

Lập trình cho vi điều khiển họ 8051 trang bị bộ nhớ chương trình kiểu Flash cần điện áp 12V làm tín hiệu cho phép lập trình để tương thích với các bộ lập trình Flash hoặc EPROM chuyên dụng.

Các chân phục vụ việc lập trình cho 8051/8951 được cho trên hình 2.16, trong đó các tín hiệu địa chỉ, dữ liệu và điều khiển phải tuân thủ theo bảng 2.9. Thuật toán lập trình:

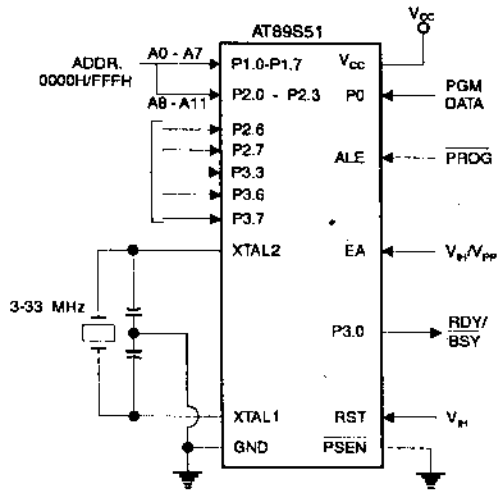
Bước 1: Đưa các địa chỉ lên các đường dây địa chỉ.

Bước 2: Đưa byte dữ liệu lên các đường dây dữ liệu.

Bước 3: Kích hoạt các đường dây điều khiển.

Bước 4: Nâng điện áp chân /EA lên 12V.

Bước 5: Hạ điện áp chân ALE/#PROG xuống mức thấp để ghi một byte vào Flash.



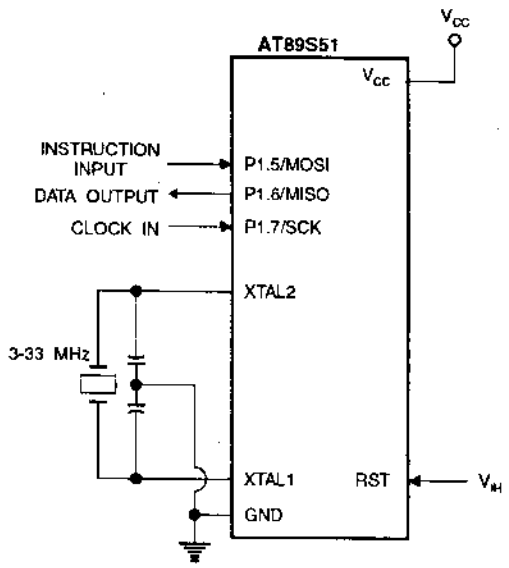
Hình 2.16. Mạch lập trình cho bộ nhớ Flash của AT89S51.

BẢNG 2.9. LẬP TRÌNH SONG SONG CHO FLASH

Mode	V_{CC}	RST	$\overline{PSEN}$	ALE/ $\overline{PROG}$	$\overline{EA}$/ V_{PP}	P2.6	P2.7	P3.3	P3.6	P3.7	P0.7-0 Data	P2.3-0 Address	P1.7-0 Address
Write Code Data	5V	H	L	(2)	12V	L	H	H	H	H	D_{IN}	A11-8	A7-0
Read Code Data	5V	H	L	H	H	L	L	L	H	H	D_{OUT}	A11-8	A7-0
Write Lock Bit 1	5V	H	L	(3)	12V	H	H	H	H	H	X	X	X
Write Lock Bit 2	5V	H	L	(3)	12V	H	H	H	L	L	X	X	X
Write Lock Bit 3	5V	H	L	(3)	12V	H	L	H	H	L	X	X	X
Read Lock Bits 1, 2, 3	5V	H	L	H	H	H	H	L	H	L	P0.2, P0.3, P0.4	X	X
Chip Erase	5V	H	L	(1)	12V	H	L	H	L	L	X	X	X
Read Atmel ID	5V	H	L	H	H	L	L	L	L	L	1EH	0000	00H
Read Device ID	5V	H	L	H	H	L	L	L	L	L	51H	0001	00H
Read Device ID	5V	H	L	H	H	L	L	L	L	L	06H	0010	00H

Notes:
1. Each $\overline{PROG}$ pulse is 200 ns - 500 ns for Chip Erase.
2. Each $\overline{PROG}$ pulse is 200 ns - 500 ns for Write Code Data.
3. Each $\overline{PROG}$ pulse is 200 ns - 500 ns for Write Lock Bits.
4. RDY/$\overline{BSY}$ signal is output on P3.0 during programming.

Với các bộ vi điều khiển có trang bị giao diện lập trình nối tiếp ISP như 89S51, 89S52... thì việc lập trình được thực hiện khi điện áp chân reset (RST) của vi điều khiển bằng Vcc như trong hình 2.17.



Hình 2.17. Mạch lập trình ISP.

BẢNG 2.10. LẬP TRÌNH ISP

Instruction	Instruction Format				Operation
	Byte 1	Byte 2	Byte 3	Byte 4	
Programming Enable	1010 1100	0101 0011	xxxx xxxx	xxxx xxxx 0110 1001 (Output on MISO)	Enable Serial Programming while RST is high
Chip Erase	1010 1100	100x xxxx	xxxx xxxx	xxxx xxxx	Chip Erase Flash memory array
Read Program Memory (Byte Mode)	0010 0000	xxxx A11 A10 A9 A8	A7 A6 A5 A4 A3 A2 A1 A0	D7 D6 D5 D4 D3 D2 D1 D0	Read data from Program memory in the byte mode
Write Program Memory (Byte Mode)	0100 0000	xxxx A11 A10 A9 A8	A7 A6 A5 A4 A3 A2 A1 A0	D7 D6 D5 D4 D3 D2 D1 D0	Write data to Program memory in the byte mode
Write Lock Bits[1]	1010 1100	1110 00 B1 B2	xxxx xxxx	xxxx xxxx	Write Lock bits. See Note (1).
Read Lock Bits	0010 0100	xxxx xxxx	xxxx xxxx	xx LB3 LB2 LB1 xx	Read back current status of the lock bits (a programmed lock bit reads back as a "1")
Read Signature Bytes	0010 1000	xxxx A11 A10 A9 A8	A7 xxx xxx0	Signature Byte	Read Signature Byte
Read Program Memory (Page Mode)	0011 0000	xxxx A11 A10 A9 A8	Byte 0	Byte 1... Byte 255	Read data from Program memory in the Page Mode (256 bytes)
Write Program Memory (Page Mode)	0101 0000	xxxx A11 A10 A9 A8	Byte 0	Byte 1... Byte 255	Write data to Program memory in the Page Mode (256 bytes)

Note: 1. B1 = 0, B2 = 0 → Mode 1, no lock protection
B1 = 0, B2 = 1 → Mode 2, lock bit 1 activated
B1 = 1, B2 = 0 → Mode 3, lock bit 2 activated
B1 = 1, B2 = 1 → Mode 4, lock bit 3 activated

} Each of the lock bit modes need to be activated sequentially before Mode 4 can be executed.

2.3. GIỚI THIỆU TẬP LỆNH CỦA 8051

Tập lệnh của 8051 được chia thành 5 nhóm như sau:

2.3.1. Nhóm lệnh số học

• **ADD A, nguồn; A = A + nguồn**

Chức năng: Cộng.

Mô tả: Lệnh ADD được dùng để cộng hai toán hạng, kết quả lưu vào thanh chứa A. Toán hạng đích luôn là thanh ghi A trong khi đó toán hạng nguồn có thể là một thanh ghi dữ liệu trực tiếp hoặc là ở trong bộ nhớ. Nếu có nhớ từ bit 7 hoặc bit 3 các cờ nhớ (CY) và cờ nhớ phụ (AC) được thiết lập. Khi cộng hai số nguyên không dấu, cờ nhớ được thiết lập bằng 1 khi phép toán bị tràn. Cờ tràn (OV) được thiết lập nếu có nhớ từ bit 6 nhưng không có nhớ từ bit 7 hoặc có nhớ từ bit 7 nhưng không có nhớ từ bit 6. Cờ tràn cho biết kết quả âm được tạo ra từ tổng của hai số nguyên dương, hoặc kết quả dương được tạo ra từ tổng hai số nguyên âm. Ngược lại các cờ trên bị xóa.

• **ADDC A, nguồn; A = A + nguồn + CY**

Chức năng: Cộng có nhớ.

Mô tả: Lệnh ADD được dùng để cộng đồng thời nội dung của hai toán hạng và cờ nhớ, kết quả lưu vào thanh chứa A. Toán hạng đích luôn là thanh ghi A trong khi đó toán hạng nguồn có thể là một thanh ghi dữ liệu trực tiếp hoặc là ở trong bộ nhớ. Nếu có nhớ từ bit 7 hoặc bit 3 các cờ nhớ (CY) và cờ nhớ phụ (AC) được thiết lập. Khi cộng hai số nguyên không dấu, cờ nhớ được thiết lập bằng 1 khi phép toán bị tràn. Cờ tràn (OV) được thiết lập nếu có nhớ từ bit 6 nhưng không có nhớ từ bit 7 hoặc có nhớ từ bit 7 nhưng không có nhớ từ bit 6. Cờ tràn cho biết kết quả âm được tạo ra từ tổng của hai số nguyên dương, hoặc kết quả dương được tạo ra từ tổng hai số nguyên âm. Ngược lại các cờ trên bị xóa.

• **SUBB A, nguồn; A = A – nguồn – CY**

Chức năng: Trừ có mượn.

Mô tả: Trong 8051 chỉ có một lệnh SUBB duy nhất. SUBB thực hiện trừ nội dung của thanh chứa A với toán hạng nguồn cùng với cờ nhớ và cất kết quả vào thanh chứa. Nếu có mượn cần đến cho bit 7 hoặc bit 3 các cờ nhớ (CY) và cờ nhớ phụ (AC) được thiết lập. Cờ tràn

(OV) được thiết lập nếu có mượn được cần đến cho bit 6 nhưng không phải cho bit 7 hoặc có mượn được cần đến cho bit 7 nhưng không phải cho bit 6. Khi trừ hai số nguyên không dấu, cờ tràn cho biết kết quả âm được tạo ra khi trừ một số dương cho một số âm hoặc kết quả dương được tạo ra khi trừ một số âm cho một số dương. Ngược lại các cờ trên bị xóa.

• **MUL AB**

Chức năng: Nhân.

Mô tả: MUL AB nhân các số nguyên không dấu 8 bit trong thanh ghi chứa A và thanh ghi B, kết quả là một số 16 bit có byte thấp đặt trong thanh chứa A và byte cao đặt trong thanh ghi B. Cờ nhớ luôn bằng 0, nếu kết quả lớn hơn 255, cờ tràn được thiết lập, ngược lại cờ này bị xóa. Ví dụ dưới đây trình bày phép nhân hai số 8 bit. Kết quả là dữ liệu 16 bit được đặt trong A và B.

```
MOV A, #9AH      ; Nạp vào A giá trị 9AH
MOV B, #35H      ; Nạp vào B giá trị 35H
MUL AB           ; 9AH*35H = 1FE2 với B = 1FH và A = E2H
```

• **DIV AB**

Chức năng: Chia A cho B.

Mô tả: 8051 cũng chỉ hỗ trợ phép chia hai số không dấu byte cho byte, tử số (số bị chia) phải ở trong thanh ghi A và mẫu số (số chia) phải ở trong thanh ghi B. Sau khi lệnh chia DIV được thực hiện thì thương số được đặt trong A, còn số dư được đặt trong B. Xét ví dụ dưới đây:

```
MOV A, #7AH      ; Nạp số bị chia vào A = 7AH
MOV B, #10H      ; Nạp số chia vào B = 10H
DIV  AB          ; A = 07 (thương số); B = 0AH (số dư)
```

Khi thực hiện DIV AB, CY = 0 và OV = 0 nếu tử số không phải là số 0. Nếu tử số là số 0 (B = 0) thì OV = 1 báo lỗi và CY = 0.

2.3.2. Nhóm lệnh logic

• **ANL đích, nguồn**

Chức năng: Thực hiện phép nhân logic (AND).

Mô tả: ANL thực hiện phép toán AND từng bit trên hai toán hạng đích và nguồn, đặt kết quả vào đích, các cờ không bị ảnh hưởng. Toán hạng đích thường là thanh ghi tổng (tích luỹ), toán hạng nguồn có thể

là thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. ANL thường được dùng để che (đặt về 0) những bit nhất định của một toán hạng. Cho biết kết quả của đoạn mã ở ví dụ dưới đây:

```
MOV A, #86H        ; Gán A = 86H
ANL A, #0FH        ; Thực hiện nhân logic A và 0FH (bây giờ A = 06)
```

Lời giải:

```
86H 1 0 0 0 0 1 1 0
0FH 0 0 0 0 1 1 1 1
06H 0 0 0 0 0 1 1 0          86H AND 0FH = 06H
```

- **ORL đích, nguồn**

Chức năng: Thực hiện phép cộng logic (OR).

Mô tả: ORL thực hiện phép toán OR từng bit trên hai toán hạng đích và nguồn, đặt kết quả vào đích, các cờ không bị ảnh hưởng. Toán hạng đích thường là thanh ghi tổng (tích luỹ) hoặc địa chỉ trực tiếp, toán hạng nguồn có thể là thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. ANL thường được dùng để thiết lập (đặt lên 1) những bit nhất định của một toán hạng. Cho biết kết quả của đoạn mã ở ví dụ dưới đây:

```
MOV A, #55H        ; A = 55H
ORL A, #0AH        ; A = 0AH
```

Lời giải:

```
55H  0 1 0 1 0 1 0 1
0AH  0 0 0 0 1 0 1 0
5FH  0 1 0 1 1 1 1 1          55H OR 0AH = 5FH
```

- **XRL đích, nguồn**

Chức năng: Thực hiện phép trừ logic (XOR).

Mô tả: XRL thực hiện phép toán XOR từng bit trên hai toán hạng đích và nguồn, đặt kết quả vào đích, các cờ không bị ảnh hưởng. Toán hạng đích thường là thanh ghi tổng (tích luỹ) hoặc địa chỉ trực tiếp, toán hạng nguồn có thể là thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. Lệnh XRL có thể được dùng để xoá nội dung của một thanh ghi bằng cách XOR nó với chính nó hoặc XRL cũng có thể được dùng để so sánh xem hai toán hạng có giá trị giống nhau không. Cho biết kết quả của đoạn mã ở ví dụ dưới đây:

a)

```
MOV A, #55H
XRL A, #79H
```

Lời giải:

```
55H 0 1 0 1 0 1 0 1
79H 0 1 1 1 1 0 0 1
2CH 0 0 1 0 1 1 0 0          55H XOR 79H = 2CH
```

b)

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

Lời giải:

```
54H 0 1 0 1 0 1 0 0
54H 0 1 0 1 0 1 0 0
00H 0 0 0 0 0 0 0 0          54H XOR 54H = 00H
```

• CPJ A

Chức năng: Lấy bù nội dung của thanh ghi A.

Mô tả: Nội dung của thanh ghi A được lấy bù logic (phép biến đổi các bit 0 thành các bit 1 và đổi các bit 1 sang bit 0). Các cờ không bị ảnh hưởng.

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

• CPL bit

Chức năng: Lấy bù bit.

Mô tả: Nội dung của bit chỉ ra trong lệnh được lấy bù logic (đổi các bit 0 thành các bit 1 và đổi các bit 1 sang bit 0). Các cờ không bị ảnh hưởng. CPL có thể được thao tác trên cờ nhớ và trên một bit bất kỳ được định địa chỉ bit.

• CJNE đích, nguồn, địa chỉ tương đối

Chức năng: So sánh và nhảy nếu không bằng nhau.

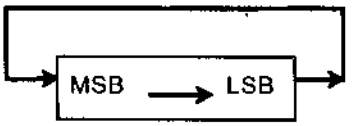
Mô tả: Lệnh CJNE so sánh hai toán hạng nguồn và đích và rẽ nhánh đến địa chỉ tương đối nếu hai toán hạng không bằng nhau. Ngoài ra cờ nhớ CY được lập nếu toán hạng đích nhỏ hơn toán hạng nguồn. Các toán hạng vẫn giữ nguyên không thay đổi.

• RR A

Chức năng: Quay các bit thanh ghi A sang phải.

Mô tả: Lệnh RR thực hiện quay các bit của thanh ghi A sang phải một vị trí, bit D0 rời từ vị trí bit thấp nhất và chuyển sang bit cao nhất D7. Các cờ không bị ảnh hưởng. Ví dụ:

```
MOV     A, #6AH   ; A = 01101010
RR      A         ; A = 00110101
RR      A         ; A = 10011010
RR      A         ; A = 01001101
```

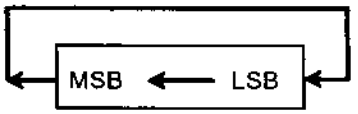


- **RL A**

Chức năng: Quay các bit thanh ghi A sang trái.

Mô tả: Lệnh RL thực hiện quay các bit của thanh ghi A sang trái một vị trí, bit D7 rời từ vị trí bit cao nhất và chuyển sang bit thấp nhất D0. Các cờ không bị ảnh hưởng. Ví dụ:

```
MOV     A, #6AH   ; A = 01101010
RL      A         ; A = 11010100
RL      A         ; A = 10101001
RL      A         ; A = 01010011
```

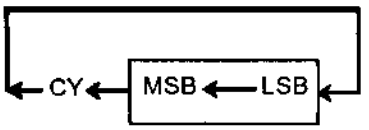


- **RLC A**

Chức năng: Quay trái qua cờ nhớ.

Mô tả: Lệnh RLC thực hiện quay các bit của thanh ghi A sang trái một vị trí, bit MSB vào cờ nhớ (CY), sau đó bit CY được chuyển vào bit LSB. Cờ nhớ CY tác động như là một bit bộ phận của thanh ghi A làm nó trở thành thanh ghi 9 bit. Ví dụ:

```
SET     C         ; C = 1
MOV     A, #71H   ; A = 01110001
RLC     A         ; A = 11100011    C = 0
RLC     A         ; A = 11000110    C = 1
RLC     A         ; A = 10001101    C = 1
```

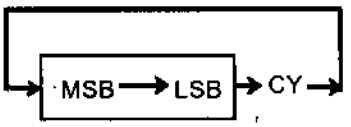


- **RRC A**

Chức năng: Quay phải qua cờ nhớ.

Mô tả: Lệnh RRC thực hiện quay các bit của thanh ghi A sang phải một vị trí, bit LSB vào cờ nhớ (CY), sau đó bit CY được chuyển vào bit MSB. Ví dụ:

```
SET     C         ; C = 1
MOV     A, #71H   ; A = 01110001
RRC     A         ; A = 10111000    C = 1
RRC     A         ; A = 11011100    C = 0
```



- SWAP A

Chức năng: Lệnh hoán đổi thanh ghi A.

Mô tả: SWAP hoán đổi 4 bit nửa phần cao của byte và 4 bit thấp của byte với nhau, SWAP chỉ hoạt động trên thanh ghi A. Các cờ không bị ảnh hưởng.

Trước khi thực hiện

D7	D6	D5	D4	D3	D2	D1	D0

Sau khi thực hiện

D3	D2	D1	D0	D7	D6	D5	D4

2.3.3. Nhóm lệnh dịch chuyển dữ liệu

1. Các lệnh dịch chuyển dữ liệu trong RAM nội

- **MOV đích, nguồn**

Chức năng: Di chuyển bit nguồn đến bit đích.

Mô tả: Nội dung của bit được chỉ ra bởi toán hạng nguồn được sao chép vào vị trí bit ở toán hạng thứ nhất. Một trong hai toán hạng phải là cờ nhớ, toán hạng còn lại là bit bất kỳ được định địa chỉ bit, bit nguồn không bị ảnh hưởng, các thanh ghi khác và các cờ không bị ảnh hưởng.

```
CLR P0.0        ; Bit P0.0 bị xóa về 0
SETB C          ; Bit cờ nhớ được thiết lập
MOV P0.1, C     ; Bit cờ chuyển vào P0.1 (bằng 1)
MOV C, P0.0     ; Bit cờ bị xóa về 0
```

- **XCH đích, nguồn**

Chức năng: Hoán chuyển nội dung của toán hạng đích và toán hạng nguồn.

Mô tả: Toán hạng đích phải là thanh ghi chứa A, lệnh XCH hoán chuyển nội dung của byte nguồn với thanh ghi chứa.

- **XCHD A, @Ri**

Chức năng: Tương tự như lệnh XCH, XCHD chỉ trao đổi nửa thấp của các byte với nhau.

Các lệnh trên chỉ trao đổi dữ liệu trong RAM nội, để cho phép dữ liệu được di chuyển giữa RAM nội với RAM ngoài ta phải sử dụng kiểu định địa chỉ gián tiếp.

2. Các lệnh dịch chuyển dữ liệu trong RAM ngoài

Có hai lệnh di chuyển dữ liệu dành cho việc đọc các bảng tìm kiếm trong bộ nhớ.

• **MOVC A, @A + DPTR**

Chức năng: Di chuyển bit nguồn đến bit đích.

Mô tả: MOVC nạp cho thanh chứa byte hằng số từ bộ nhớ chương trình. Địa chỉ của byte được nạp là tổng của giá trị 8 bit không dấu ban đầu chứa trong thanh chứa với nội dung của thanh ghi địa chỉ cơ sở (thanh ghi cơ sở có thể là thanh ghi con trỏ dữ liệu hoặc PC). Trong trường hợp sau, PC được tăng để chỉ đến địa chỉ của lệnh tiếp theo trước khi được cộng với nội dung của thanh ghi chứa; các thanh ghi địa chỉ cơ sở không bị thay đổi. Phép cộng bit thứ 16 do số nhớ từ 8 bit thấp có thể truyền qua các bit cao. Các cờ không bị ảnh hưởng.

• **MOVC** A, @A+PC

Cũng hoạt động tương tự như lệnh trên, ngoại trừ ở đây bộ đếm chương trình được dùng để chứa địa chỉ cơ sở và bảng được truy xuất nhờ vào một chương trình con. Trước tiên số của điểm nhập yêu cầu được nạp cho thanh chứa A, sau đó chương trình con được gọi. Chuỗi lệnh cho phép khởi động và gọi có thể là:

```
                MOV A, ENTRY_NUMBER
                CALL LOOK_UP
LOOK_UP:        INC A
                MOVC A, @A+PC
                RET
TABLE:          DB data, data, data,...
```

Bảng được định nghĩa sau lệnh RET trong chương trình, lệnh tăng được cần đến do PC trỏ tới lệnh RET khi lệnh MOVC được thực thi. Việc tăng nội dung thanh ghi chứa sẽ bỏ qua lệnh RET.

2.3.4. Nhóm lệnh xử lý bit

Bộ xử lý của 8051 chứa một bộ xử lý logic trên bit cho phép ta thực hiện các phép toán đơn bit. Do đó 8051 có khả năng truy cập đến từng bit một thay vì phải truy cập cả byte như ở các bộ vi xử lý. Các thanh ghi, RAM và các cổng cần được đánh địa chỉ theo bit vì có rất nhiều ứng dụng thì ta chỉ cần thay đổi giá trị của một bit chẳng hạn như là bật hoặc tắt một thiết bị. RAM nội chứa 128 bit và không gian SFR hỗ trợ thêm đến 128 bit được định địa chỉ. Tất cả các Port P0, P1, P2, P3 cũng đều được định địa chỉ bit. Các bit có thể được lập và được xoá chỉ bằng một lệnh. Dưới đây là bảng các lệnh dùng cho các phép tính một bit.

BẢNG 2.11. CÁC LỆNH MỘT BIT CỦA 8051

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

- **SETB bit**

Chức năng: Thiết lập bit.

Mô tả: SETB đặt bit được chỉ ra trong lệnh bằng 1, SETB có thể thao tác trên các cờ nhớ hoặc các bit được định địa chỉ bit. Ví dụ:

```
SETB C          ; Thiết lập cờ nhớ bằng 1
```

Bốn cổng I/O 8 bit là P0, P1, P2 và P3 của bộ vi điều khiển 8051 có thể truy cập toàn bộ 8 bit hoặc theo một bit bất kỳ mà không làm thay đổi các bit khác còn lại. Khi truy cập một cổng theo từng bit, chúng ta sử dụng cú pháp "SETB X, Y" với X là tên của cổng P0, P1, P2 hoặc P3, còn Y là vị trí các bit dữ liệu từ 0 đến 7. Ví dụ:

```
SETB P0.1       ; Thiết lập bit 1 của cổng P0
SETB TR0        ; Đặt TR0 = 1 để điều khiển bộ Timer 0 hoạt động
```

Lệnh trên khi được hợp dịch nó trở thành "SETB 81H" vì P0.1 có địa chỉ trong RAM là 81H.

```
SETB C          ; Thiết lập cờ nhớ bằng 1
```

- **CLR bit**

Chức năng: Xóa bit.

Mô tả: CLR xoá bit được chỉ ra trong lệnh về 0, CLR có thể thao tác trên các cờ nhớ hoặc các bit được định địa chỉ bit. Ví dụ:

```
CLR P0.1        ; xoá bit 1 của cổng P0
CLR TR0
```

- **CPL bit**

Chức năng: Lấy bù bit.

Mô tả: CPL được chỉ ra trong lệnh được lấy bù, một bit có giá trị 1 được đổi thành 0 và ngược lại. CPL có thể thao tác trên các cờ nhớ hoặc các bit được định địa chỉ bit. Ví dụ:

```
SETB P0.1       ; Bit 1 của cổng P0 được [illegible] thành 1
CPL P0.1        ; Đảo lại bit 1 của cổng P0 bằng 0
```

2.3.5. Nhóm lệnh rẽ nhánh

Phần này sẽ trình bày các lệnh chuyển điều khiển có trong hợp ngữ của 8051 như các lệnh sử dụng cho vòng lặp, các lệnh nhảy có và không có điều kiện, lệnh gọi chương trình con.

1. Các lệnh nhảy có điều kiện

- **JZ nhãn_đích**

Chức năng: Nhảy nếu A = 0.

Mô tả: Khi thực hiện lệnh JZ, nội dung của thanh ghi A được kiểm tra. Nếu nó bằng không thì nó nhảy đến địa chỉ đích, ngược lại thì tiếp tục với lệnh tiếp theo. Ví dụ xét đoạn mã sau:

```
MOV A, R0        ; Nạp giá trị của R0 vào     A
JZ LABLE         ; Nhảy đến LABLE nếu         A = 0
MOV A, R1        ; Nạp giá trị của R1 vào     A
JNZ LABLE        ; Nhảy đến LABLE nếu         A ≠ 0
LABLE:
```

- **JNC nhãn_đích**

Chức năng: Nhảy nếu không có nhớ (CY = 0).

Mô tả: Khi thực hiện lệnh JNC, bộ xử lý kiểm tra cờ nhớ nếu CY = 0 thì CPU bắt đầu nạp và thực hiện các lệnh từ địa chỉ của nhãn. Nếu cờ CY = 1, CPU sẽ chuyển đến địa chỉ đích, ngược lại thì tiếp tục với lệnh tiếp theo. Dưới đây là bảng các lệnh nhảy có điều kiện.

BẢNG 2.12. CÁC LỆNH NHẢY CÓ ĐIỀU KIỆN

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

Chú ý rằng tất cả các lệnh nhảy có điều kiện đều là các lệnh nhảy

ngắn, có nghĩa là địa chỉ của đích đều phải nằm trong khoảng –127 đến +127 byte của nội dung bộ đếm chương trình PC.

2. Các lệnh nhảy không điều kiện

- **LJMP nhãn_đích**

Chức năng: Nhảy dài.

Mô tả: Nhảy xa LJMP là một lệnh 3 byte trong đó byte đầu tiên là mã lệnh còn 2 byte còn lại là địa chỉ 16 bit của đích. Địa chỉ đích 2 byte có một phép nhảy đến bất kỳ vị trí nhớ nào trong khoảng 0000 – FFFH.

- **SJMP nhãn_đích**

Chức năng: Nhảy ngắn.

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

3. Các lệnh gọi

- **ACALL nhãn_lệnh**

Chức năng: Gọi đến địa chỉ tuyệt đối.

Mô tả: ACALL gọi không điều kiện một chương trình con đặt tại địa chỉ được chỉ ra trong lệnh. Do ACALL chỉ có 2 byte nên địa chỉ đích của chương trình con phải nằm trong khoảng 2kbyte địa chỉ vì chỉ có 11 bit của 2 byte được sử dụng cho địa chỉ. Không có sự khác biệt nào giữa ACALL và LCALL trong khái niệm cất bộ đếm chương trình vào ngăn xếp hay trong chức năng của lệnh trở về RET. Sự khác nhau duy nhất là địa chỉ đích của lệnh LCALL có thể nằm bất cứ đâu trong phạm vi 64 kbyte không gian địa chỉ của 8051, còn trong khi đó địa chỉ của lệnh ACALL phải nằm trong khoảng 2 byte. Trong nhiều thành viên của họ 8051 do các hãng cung cấp thì ROM trên chíp chỉ có 1 kbyte, việc sử dụng ACALL thay cho LCALL có thể tiết kiệm được một số byte bộ nhớ của không gian ROM chương trình.

- **LCALL nhãn_lệnh**

Chức năng: Gọi xa.

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

2.4. HOẠT ĐỘNG ĐỊNH THỜI

2.4.1. Giới thiệu

Các bộ định thời (Timer) được sử dụng rất rộng rãi trong các ứng dụng đo lường và điều khiển. Có thể coi một bộ định thời n bit là một bộ đếm n bit được tạo ra bởi n flip–flop mắc nối tiếp với nhau. Đầu vào của bộ định thời chính là đầu vào của flip–flop đầu tiên, đầu ra báo tràn (over flow) của bộ định thời phản ánh trạng thái tràn của nó. Đầu ra của các flip–flop phản ánh giá trị hiện thời của bộ đếm. Tuỳ thuộc vào ứng dụng, đầu vào bộ định thời có thể là nguồn xung lấy từ xung nhịp của vi điều khiển hoặc nguồn xung từ bên ngoài đưa đến. Vi điều khiển AT89S52 có ba bộ định thời 16 bit trong đó hai bộ Timer 0 và Timer 1 có bốn chế độ hoạt động, Timer 2 có ba chế độ hoạt động. Các bộ định thời được dùng để định khoảng thời gian (hẹn giờ), đếm sự kiện xảy ra bên ngoài bộ vi điều khiển hoặc tạo tốc độ baud cho cổng nối tiếp của vi điều khiển.

Trong các ứng dụng định khoảng thời gian, Timer được lập trình sao cho sẽ tràn sau một khoảng thời gian và thiết lập cờ tràn bằng 1. Cờ tràn được sử dụng bởi chương trình để thực hiện một hành động tương ứng như kiểm tra trạng thái của các ngõ vào hoặc gửi các sự kiện ra các ngõ ra.

Đếm sự kiện dùng để xác định số lần xảy ra của một sự kiện. Trong ứng dụng này người ta tìm cách quy các sự kiện thành sự chuyển mức từ 1 xuống 0 trên các chân T0 hoặc T1 hoặc T2 để dùng các Timer tương ứng đếm các sự kiện đó.

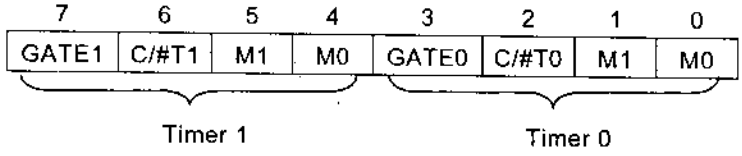
Dựa trên hai chức năng này, các bộ định thời còn có thể được lập trình để tạo xung nhịp, đo thời gian trôi qua giữa hai sự kiện (ví dụ đo độ rộng xung)...

2.4.2. Các thanh ghi của bộ định thời

1. Các thanh ghi của Timer 0 và Timer 1

Thanh ghi chế độ định thời (TMOD)

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



Hình 2.18. Thanh ghi TMOD.

Bit	Tên	Timer	Mô tả
7	GATE1	1	Bit mở cổng cho Timer 1, khi được đặt bằng 1 thì Timer 1 chỉ chạy khi chân INT1 ở mức cao. Nếu bit này được đặt là 0 thì hoạt động của Timer 1 không bị ảnh hưởng bởi mức logic trên chân INT1.
6	C/#T1	1	Bit chọn chế độ Counter/Timer của Timer 1 1 = bộ đếm sự kiện 0 = bộ định khoảng thời gian.
5	M1	1	Bit 1 chọn chế độ (mode) của Timer 1.
4	M0	1	Bit 0 chọn chế độ của Timer 1. 00: chế độ 0 – Timer 13 bit 01: chế độ 1 – Timer 16 bit 10: chế độ 2 – 8 bit tự động nạp lại 11: chế độ 3 – Tách Timer.
3	GATE0	0	Bit mở cổng cho Timer 0, khi được đặt bằng 1 thì Timer 0 chỉ chạy khi chân INT0 ở mức cao.
2	C/#T0	0	Bit chọn chế độ counter/timer của Timer 0.
1	M1	0	Bit 1 chọn chế độ của Timer 0.
0	M0	0	Bit 0 chọn chế độ của Timer 0.

Thanh ghi điều khiển Timer (TCON)

Thanh ghi TCON chứa các bit trạng thái và các bit điều khiển cho Timer 0 và Timer 1.

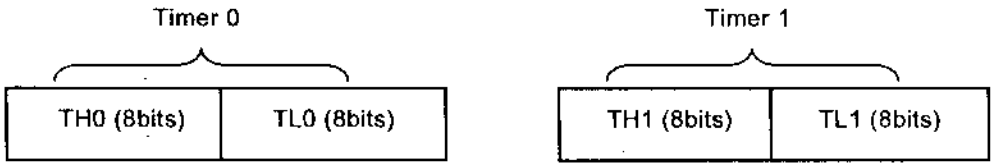
TCON.7	TCON.6	TCON.5	TCON.4	TCON.3	TCON.2	TCON.1	TCON.0
TF1	TR1	TF0	TR0	IT1	IE1	IT0	IE0

Hình 2.19. Thanh ghi TCON.

Bit	Ký hiệu	Địa chỉ	Mô tả
TCON.7	TF1	8FH	Cờ báo tràn của Timer 1, được đặt bởi phần cứng khi có tràn, được xóa bởi phần mềm hoặc bởi phần cứng khi bộ xử lý chỉ đến chương trình phục vụ ngắt.
TCON.6	TR1	8EH	Bit điều khiển Timer 1 hoạt động, được đặt/xóa bằng phần mềm để điều khiển cho Timer chạy/dừng.
TCON.5	TF0	8DH	Cờ báo tràn Timer 0.
TCON.4	TR0	8CH	Bit điều khiển Timer 0 hoạt động.
TCON.3	IT1	8BH	Cờ ngắt do Timer 1.
TCON.2	IE1	8AH	Cờ ngắt ngoài 1.
TCON.1	IT0	89H	Cờ ngắt do Timer 0.
TCON.0	IE0	88H	Cờ ngắt ngoài 0.

Các thanh ghi chứa giá trị của các bộ định thời

Các Timer 0 và Timer 1 đều là các Timer 16 bit, mỗi Timer có 2 thanh ghi 8 bit dùng để chứa giá trị khởi tạo hoặc giá trị hiện thời của các Timer. Cụ thể Timer 0 có TH0 và TL0; Timer 1 có TH1 và TL1. Cần lưu ý rằng các thanh ghi này không được định địa chỉ bit.



Hình 2.20. Các thanh ghi chứa giá trị của các bộ định thời.

2. Các thanh ghi của Timer 2

Thanh ghi T2CON

T2CON.7	T2CON.6	T2CON.5	T2CON.4	T2CON.3	T2CON.2	T2CON.1	T2CON.0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/#T2	CP/#RL2

Bit	Ký hiệu	Địa chỉ	Mô tả
T2CON.7	TF2	CFH	Cờ báo tràn của Timer 2, TF2 được đặt khi Timer 2 tràn và được xoá bằng phần mềm. TF2 không được thiết lập khi TCLK hoặc RCLK được đặt bằng 1.
T2CON.6	EXF2	CEH	Cờ ngắt ngoài của Timer 2, TXF2 = 1 khi xảy ra sự nạp lại hoặc thu nhận. EXF2 = 1 cũng gây ra ngắt do Timer 2 nếu như ngắt này được lập trình cho phép, EXF2 được xóa bởi phần mềm.
T2CON.5	RCLK	CDH	Bit chọn Timer cung cấp xung nhịp cho đường nhận của cổng nối tiếp. RCLK = 1 thì Timer 2 sẽ cung cấp tốc độ baud cho cổng nối tiếp (ở chế độ 1 và 3). RCLK = 0 thì Timer 1 sẽ cung cấp tốc độ baud cho cổng nối tiếp (ở chế độ 1 và 3).
T2CON.4	TCLK	CCH	Bit chọn Timer cung cấp xung nhịp cho đường truyền của cổng nối tiếp. TCLK = 1 thì Timer 2 sẽ cung cấp tốc độ baud cho cổng nối tiếp ở đường truyền. TCLK = 0 thì Timer 1 sẽ cung cấp tốc độ baud cho cổng nối tiếp ở đường truyền.
T2CON.3	EXEN2	CBH	Bit điều khiển hoạt động của Timer 2, khi EXEN2 = 1 việc nạp lại hoặc thu nhận (capture) diễn ra khi có sự chuyển trạng thái từ 1 sang 0 ở chân T2EX nếu T2 không sử dụng để cung cấp tốc độ baud cho cổng nối tiếp.
T2CON.2	TR2	CAH	Bit điều khiển hoạt động của Timer 2 (tương tự TR0, 1).
T2CON.1	C/#T2	C9H	Bit chọn chế độ đếm hoặc định thời của Timer 2 (tương tự C/#T0, 1).
T2CON.0	CP/#RL2	C8H	Bit chọn chế độ thu nhận hay nạp lại của Timer 2. Khi CP/#RL2C được thiết lập bằng 1, việc thu nhận được thực hiện khi có sườn xuống ở chân T2EX và bit EXEN2 được đặt là 1. Khi CP/#RL2C được đặt bằng 0, việc nạp lại được thực hiện khi hoặc là Timer 2 tràn hoặc là khi có sườn xuống ở chân T2EX và bit EXEN2 được đặt là 1. Nếu RCLK hoặc TCLK = 1, bit này được bỏ qua, Timer 2 tự nạp lại khi tràn.

Thanh ghi T2MOD

T2MOD có địa chỉ 0C9H, thanh ghi này không định địa chỉ bit.

Bit	Ký hiệu	Mô tả
T2MOD.7		Không sử dụng.
T2MOD.6		Không sử dụng.
T2MOD.5		Không sử dụng.
T2MOD.4		Không sử dụng.
T2MOD.3		Không sử dụng.
T2MOD.2		Không sử dụng.
T2MOD.1	T2OE	Cho phép đầu ra khi sử dụng timer 2 để tạo xung (chế độ tạo xung – clock out).
T2MOD.0	DCEN	Bit cho phép Timer 2 hoạt động như một bộ đếm tiến/lùi

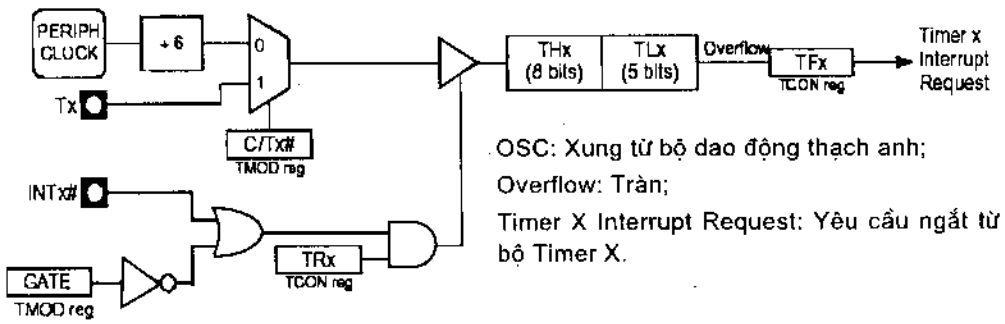
Thanh ghi TH2 và TL2, RCAP2H và RCAP2L.

Cũng giống như TH0, 1 và TL0, 1, TH2 và TL2 chứa giá trị đếm của Timer 2, tuy nhiên khác nhau là Timer 0, 1 có thể dùng THx để chứa giá trị nạp lại còn Timer 2 dùng RCAP2H và RCAP2L để chứa giá trị cần nạp lại.

2.4.3. Các chế độ của bộ định thời

1. Các chế độ của Timer 0 và Timer 1

Chế độ 0



Hình 2.21. Hoạt động của Timer 0 và Timer 1 ở chế độ 0.

Chế độ 0 là chế độ định thời 13 bit, chế độ này tương thích với các bộ vi điều khiển trước đó, trong các ứng dụng hiện nay chế độ này không còn thích hợp.

Trong chế độ này, bộ định thời dùng 13 bit (8 bit của TH và 5 bit cao của TL) để chứa giá trị đếm, 3 bit thấp của TL không được sử dụng.

Hình 2.21 mô tả hoạt động của các Timer ở chế độ 0: Nguồn xung clock được đưa tới Timer từ một trong cách phụ thuộc vào bit C/#T trong thanh ghi TMOD.

– Nếu C/#T = 1, xung clock sẽ được lấy từ bên ngoài qua chân Tx (T0, T1 hoặc T2).

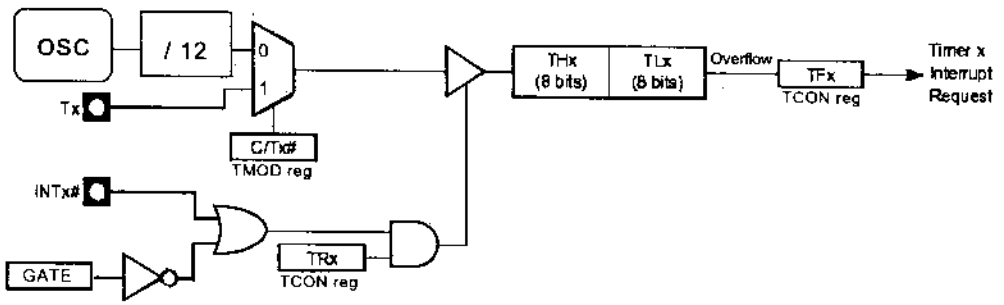
– Nếu C/#T = 0, xung clock sẽ được lấy từ bộ chia tần trong chip, tần số của xung ở đây là 1/12 tần số của bộ dao động thạch anh (F_{osc}).

Nguồn xung clock nói trên sẽ được điều khiển để đưa tới các Timer bằng các bit: TR, GATE và mức logic trên các chân INTx.

– Nếu TRx = 0, các Timer sẽ bị cấm mà không cần quan tâm tới GATE và mức logic trên các chân INTx (thể hiện bằng "cổng AND").

– Nếu TRx = 1, các Timer sẽ hoạt động khi hoặc là bit GATE = 0 hoặc là bit GATE = 1 và trên chân /INTx có mức logic 1.

Chế độ 1



Hình 2.22. Hoạt động của Timer 0 và Timer 1 ở chế độ 1.

Trong chế độ 1, bộ Timer dùng cả 2 thanh ghi TH và TL để chứa giá trị đếm, vì vậy chế độ này còn được gọi là chế độ định thời 16 bit. Bit MSB sẽ là bit D7 của TH còn bit LSB là D0 của TL.

Hình 2.22 mô tả hoạt động của các Timer ở chế độ 1: Nguồn xung clock được đưa tới Timer từ một trong cách phụ thuộc vào bit C/#T trong thanh ghi TMOD.

– Nếu C/#T = 1, xung clock sẽ được lấy từ bên ngoài qua chân Tx (T0, T1 hoặc T2).

– Nếu C/#T = 0, xung clock sẽ được lấy từ bộ chia tần trong chip, tần số của xung ở đây là 1/12 tần số của bộ dao động thạch anh (F_{osc}).

Nguồn xung clock nói trên sẽ được điều khiển để đưa tới các Timer bằng các bit: TR, GATE và mức logic trên các chân INTx.

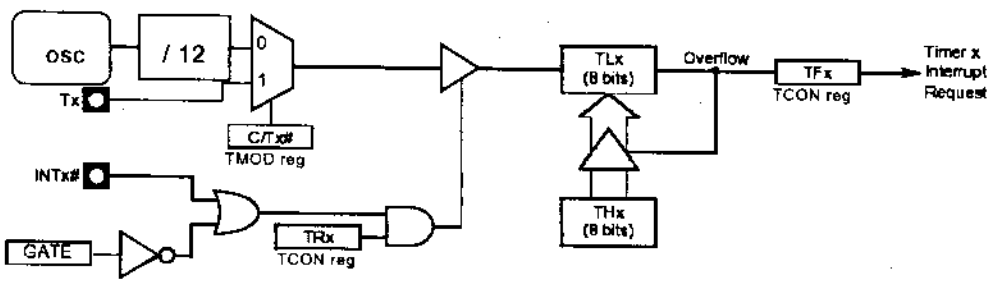
– Nếu TRx = 0, các Timer sẽ bị cấm mà không cần quan tâm tới GATE và mức logic trên các chân INTx (thể hiện bằng "cổng AND").

– Nếu TRx = 1, các Timer sẽ hoạt động khi hoặc là bit GATE = 0 hoặc là bit GATE = 1 và trên chân /INTx có mức logic 1.

Với chế độ 1, giá trị lớn nhất mà các Timer chứa được là 65.535 (tương ứng FFFFH), khi đếm quá giá trị này sẽ xảy ra tràn, khi cờ tràn

TF sẽ được đặt bằng 1. Sau khi xảy ra tràn, nếu muốn Timer tiếp tục đếm, chương trình phải có câu lệnh nạp lại giá trị khởi tạo sau khi đã dừng Timer bằng cách xoá bit TR.

Chế độ 2

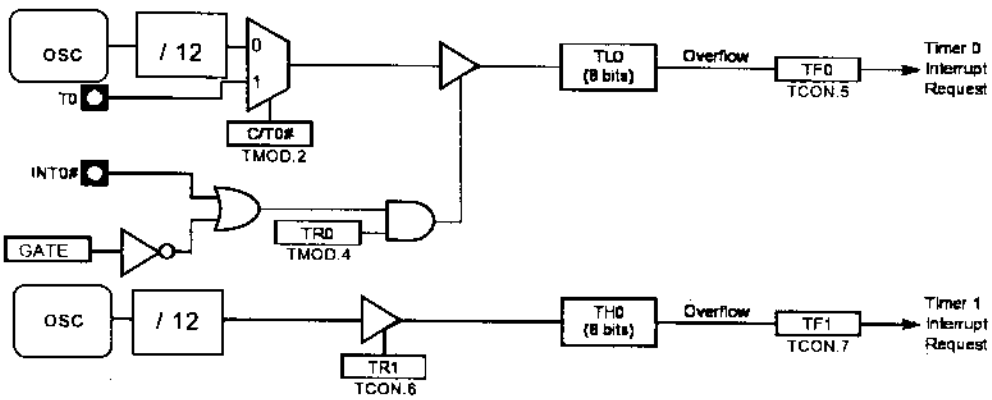


Hình 2.23. Hoạt động của Timer 0 và Timer 1 ở chế độ 2.

Trong chế độ 2, bộ Timer dùng TL để chứa giá trị đếm và TH để chứa giá trị nạp lại vì vậy chế độ này được gọi là chế độ tự nạp lại 8 bit. Sau khi đếm quá 255 sẽ xảy ra tràn, khi đó TF được đặt bằng 1 đồng thời giá trị của Timer tự động được nạp lại bằng nội dung của TH.

Với nguồn xung clock, cách điều khiển Timer ở chế độ 2 hoàn toàn giống chế độ 1 (hình 2.23).

Chế độ 3



Hình 2.24. Hoạt động của Timer 0 ở chế độ 3.

Trong chế độ 3, Timer 0 được tách thành 2 bộ Timer hoạt động độc lập (hình 2.24), chế độ này sẽ cung cấp cho bộ vi điều khiển thêm một Timer nữa.

Bộ Timer thứ nhất với nguồn xung clock được lấy từ bộ chia tần trên chip hoặc từ bộ tạo xung bên ngoài qua chân T0 tuỳ thuộc vào giá trị của bit C–/T0. Việc điều khiển hoạt động của bộ thứ nhất do bit GATE, bit TR0 và múc logic trên chân INT0 (giống chế độ 0, 1, 2).

Giá trị đếm của Timer được chứa trong TL0, khi xảy ra tràn, cờ TF0 được đặt bằng 1 và gây ra ngắt do Timer 0 (nếu được đặt).

Bộ Timer thứ hai với nguồn xung clock được lấy từ bộ chia tần trên chip. Việc điều khiển hoạt động của bộ thứ hai chỉ là việc đặt giá trị của bit TR0. Giá trị đếm của Timer được chứa trong TH0, khi xảy ra tràn, cờ TF1 được đặt bằng 1 và gây ra ngắt do Timer 1 (nếu được đặt).

Khi Timer 0 được tách thành 2 Timer 8 bit thì Timer 1 vẫn có thể hoạt động bình thường ở các chế độ 0, 1, 2, tuy nhiên khi xảy ra tràn cờ TF1 không được thiết lập bằng 1. Như vậy trong trường hợp này Timer 1 chỉ có thể sử dụng cho các ứng dụng không cần đến ngắt (TF1 = 1), chẳng hạn như tạo tốc độ baud cho port nối tiếp.

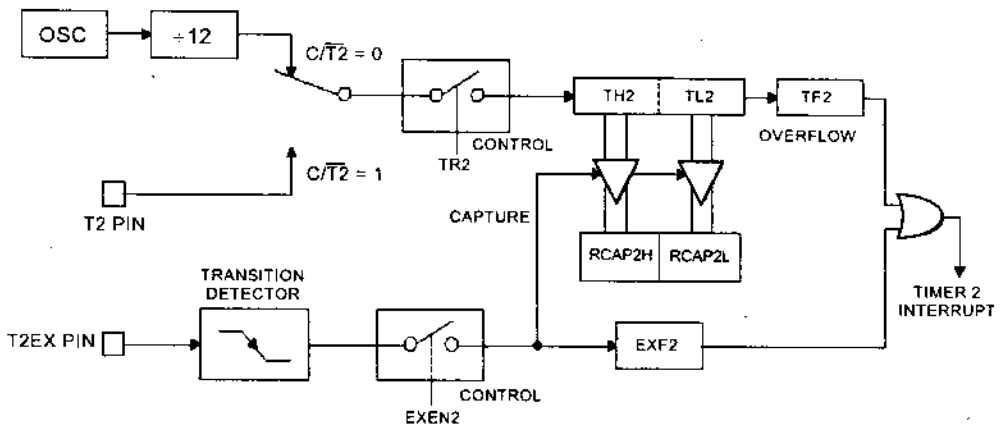
2. Các chế độ của Timer 2

Timer 2 có ba chế độ hoạt động là chế độ thu nhận (Captuer), chế độ tự nạp lại (Auto – reload) và chế độ cung cấp tốc độ baud cho cổng nối tiếp (Baud Rate Generator).

BẢNG 2.13. CÁC CHẾ ĐỘ HOẠT ĐỘNG CỦA TIMER 2

RCLK +TCLK	TR2	CP/#RL2	Chế độ
0	0	1	16–bit Auto–reload: 16 bit tự nạp lại.
0	1	1	16–bit Capture: 16 bit thu nhận.
1	X	1	Baud Rate Generator: cung cấp tốc độ baud.
X	X	0	(Off)

Chế độ thu nhận (capture)

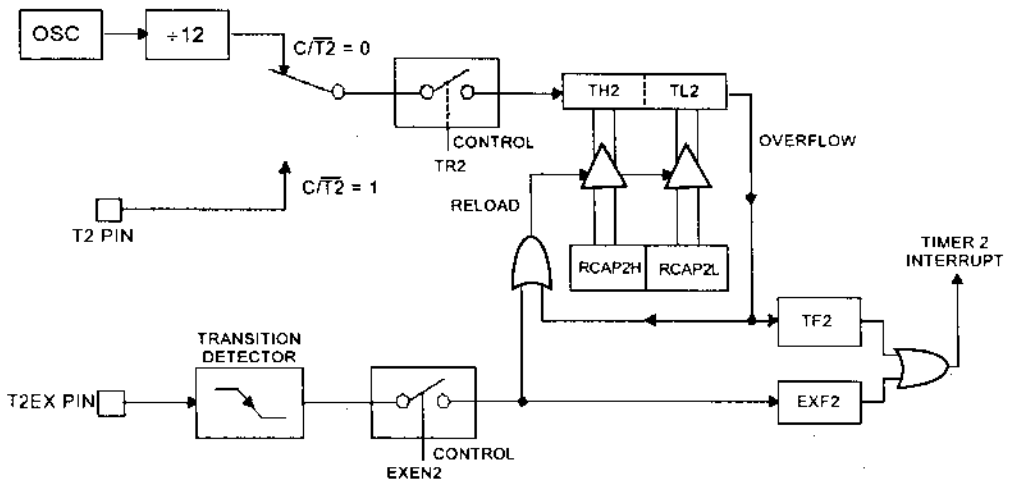


Hình 2.25. Timer 2 ở chế độ thu nhận (capture).

Khi CP/#RL2 = 1, chế độ thu nhận của Timer 2 được chọn bởi bit EXEN2. Xung clock cấp cho Timer 2 cũng được lấy từ một trong hai nguồn và được điều khiển bởi C/#T2 (tương tự Timer 0, Timer 1). Điều khiển sự hoạt động của Timer 2 cũng là bit TR2. Giá trị đếm của Timer được chứa trong TH2 và TL2 (Timer 2 hoạt động như một Timer 16 bit). Khi xảy ra tràn cờ tràn TF2 được đặt bằng 1.

Giá trị hiện thời của Timer 2 nằm trong TH2 và TL2 sẽ được chuyển tương ứng vào RCAP2H và RCAP2L khi bit EXEN2 được đặt bằng 1 và có sự chuyển mức từ 1 xuống 0 trên chân T2EX. Sự chuyển mức này kết hợp với việc đặt bit EXF2 bằng 1 sẽ gây ra ngắt ngoài do Timer 2.

Chế độ tự nạp lại

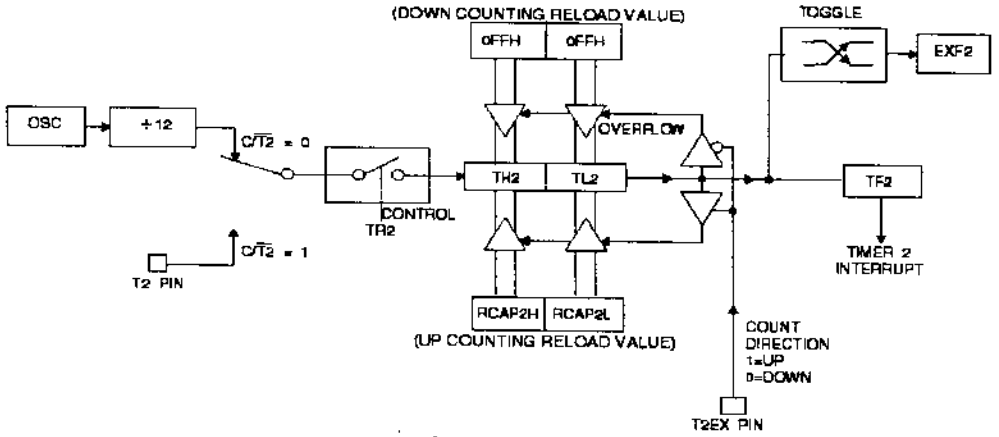


Hình 2.26. Timer 2 ở chế độ tự nạp lại (DCEN = 0).

Trong chế độ này, khi bit DCEN = 0 Timer 2 hoạt động như một Timer 16 bit tự nạp lại. Giá trị nạp lại được chứa trong RCAP2H và RCAP2L. Sự kiện nạp lại xảy ra khi:

– Hoặc là xảy ra tràn tức là có sự chuyển số đếm từ FFFFH sang 0.

– Hoặc là có sự chuyển mức từ 1 xuống 0 trên chân T2EX khi EXEN2 đã được đặt bằng 1. Sự chuyển mức này cũng đồng thời đặt EXF2 = 1.



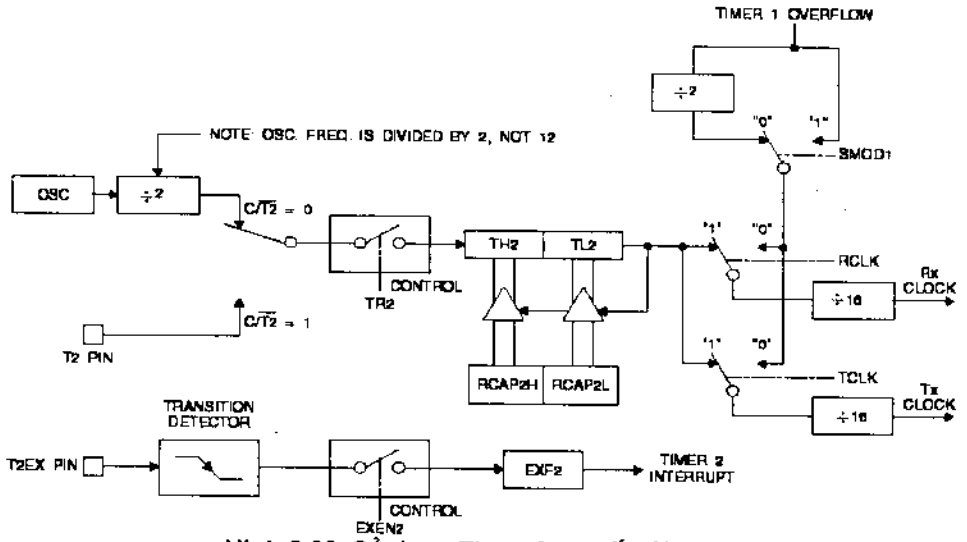
Hình 2.27. Timer 2 ở chế độ tự nạp lại (DCEN = 1).

Khi bit DCEN = 1, Timer 2 vẫn hoạt động như một Timer 16 bit tự nạp lại, có 2 cách tự nạp lại:

Cách thứ nhất: Khi chân T2EX được đặt ở mức logic 1, Timer 2 sẽ đếm tiến từ giá trị xuất phát cho đến khi có sự chuyển số đếm từ FFFFH sang 0 thì xảy ra tràn. Sự kiện tràn sẽ thiết lập cờ TF2 đồng thời nạp lại giá trị cho Timer, giá trị nạp lại được chứa trong RCAP2H và RCAP2L.

Cách thứ hai: Khi chân T2EX được đặt ở mức logic 0, Timer 2 sẽ đếm lùi từ giá trị xuất phát cho đến giá trị được đặt trong RCAP2H và RCAP2L thì xảy ra tràn. Sự kiện tràn sẽ thiết lập cờ TF2 đồng thời nạp lại giá trị cho Timer, giá trị nạp lại sẽ là FFFFH.

– Tạo tốc độ baud cho cổng nối tiếp.

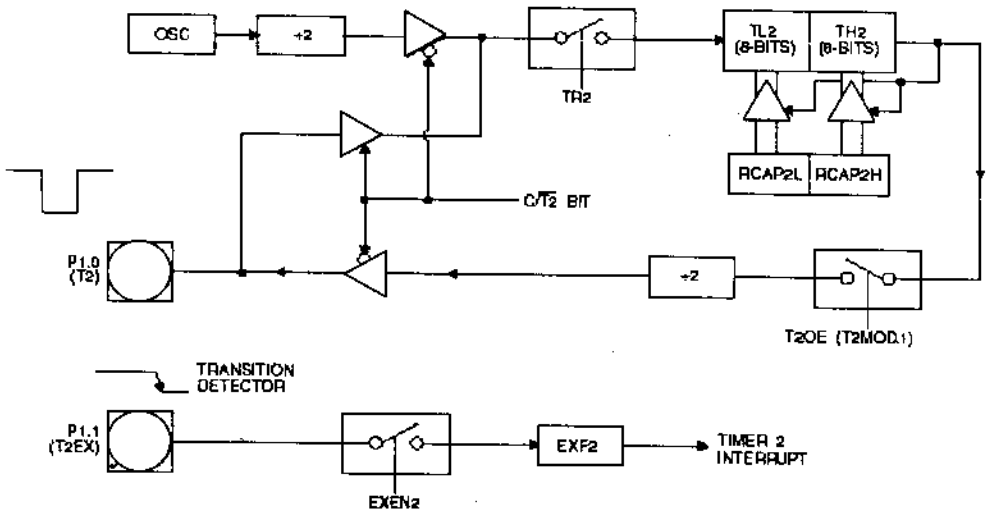


Hình 2.28. Sử dụng Timer 2 tạo tốc độ baud.

Timer 2 có thể được dùng để tạo tốc độ baud cho đường truyền và đường nhận của cổng nối tiếp tuỳ thuộc vào các bit TCLK và RCLK trong thanh ghi T2CON.

– Tạo xung vuông dùng Timer 2.

Có thể sử dụng Timer 2 để tạo xung vuông (tỷ số độ rộng xung 50%) ở chân P1.1 như trong hình 2.29. Khi đó bit C/#T2 phải được xóa, bit T2OE phải được thiết lập bằng 1, còn bit TR2 được dùng để điều khiển chạy/dừng Timer 2, tức là tạo/dừng xung vuông trên chân P1.1.



Hình 2.29. Timer 2 trong chế độ tạo xung.

Tần số xung vuông tạo ra trên chân P1.1 được cho bởi công thức:

$$F_{P1.1} = F_{OSC} / (4*(65.536 - [RCAP2H, PCAP2L]))$$

Trong đó [RCAP2H, PCAP2L] là nội dung của thanh ghi RCAP, được tính bằng công thức:

$$[RCAP2H, CAP2L] = RCAP2H*256 + PCAP2L$$

Với thạch anh 16MHz, tuỳ thuộc vào giá trị ghi vào [RCAP2H, PCAP2L] mà ta có thể tạo ra trên chân P1.1 xung vuông có tần số trong dải từ 61Hz đến 4MHz.

2.4.4. Nguồn xung clock

Nguồn xung clock cấp cho các bộ định thời có thể lấy từ ngoài qua các chân T0, T1, T2 lần lượt cho các bộ Timer 0, Timer 1, Timer 2 hoặc lấy từ bộ chia tần trên chip với tần số là 1/12 tần số của bộ dao động

thạch anh tuỳ thuộc vào bit C/#T. Nhìn chung với các ứng dụng định thời thì nguồn xung clock thường được lấy ngay trên chip còn với ứng dụng đếm thì nguồn xung được lấy từ ngoài qua các chân T0, T1, T2.

2.4.5. Làm việc với các bộ định thời

Trong phần này chúng ta sẽ xem xét một số ví dụ ứng dụng các bộ định thời.

Ví dụ 1: Tạo xung có tần số 10kHz trên chân P1.1, biết thạch anh được dùng có tần số 12MHz.

Trong ví dụ này tần số cần tạo là 10kHz trong khi xung từ bộ chia tần của AT89S52 cấp cho các bộ Timer là 1MHz, như vậy chúng ta cần có hệ số chia là 100. Với hệ số chia này thì chỉ cần các bộ Timer 8 bit là có thể đáp ứng được, dưới đây là chương trình cụ thể:

```
#include<reg52.h>
sbit F = P1^1;
void main(void)
{
 TMOD = 0X02;              //Timer 0 mode 2
 TH0 = -50;                //hệ số chia là 100
 TR0 = 1;
while(1)
{
while (!TF0);              //chờ cờ tràn
 TF0=0;                    //xoá cờ tràn
 F=~F;                     //đảo mức
}
}
```

Trong ví dụ trên, giá trị nạp cho TH0 là –50 trong khi thanh ghi này có thể chứa giá trị tối đa là –255 tương ứng với hệ số chia là 510. Như vậy xung có tần số nhỏ nhất mà một bộ Timer 8 bit có thể tạo được từ thạch anh 12MHz là 1/510MHz.

Ví dụ 2: Tạo xung có tần số 200Hz trên chân P1.1, biết thạch anh được dùng có tần số 12MHz.

Trong ví dụ này tần số cần tạo là 200Hz trong khi xung từ bộ chia tần của AT89S52 cấp cho các bộ Timer là 1MHz, như vậy chúng ta cần có hệ số chia là 5000. Với hệ số chia này thì chỉ cần các bộ Timer 16 bit là có thể đáp ứng được, dưới đây là chương trình cụ thể:

```
#include<reg52.h>
sbit F = P1^1;
void main(void)
{
 TMOD = 0X01;              //Timer 0 mode 1
while(1)
{
 TH0 = -2500/256;          //hệ số chia là 5000
 TL0 = -2500%256;
 TR0 = 1;
 while (!TF0);             //chờ cờ tràn
 TF0 = 0;                  //xoá cờ tràn
 TR0 = 0;                  //dừng Timer
 F = ~F;                   //đảo mức trên P1.1
}
}
```

Trong ví dụ trên, giá trị nạp cho TH0TL0 là –2500 trong khi thanh ghi này có thể chứa giá trị tối đa là –65535 tương ứng với hệ số chia là 131070. Như vậy xung có tần số nhỏ nhất mà một bộ Timer 8 bit có thể tạo được từ thạch anh 12MHz là 1/131070MHz. Lưu ý rằng trong chế độ 1 các Timer 0, 1 không tự nạp lại mà chương trình phải có câu lệnh nạp lại giá trị xuất phát khi Timer đã được dừng bởi lệnh TRx = 0.

Xung có tần số 200Hz có thể tạo bằng chế độ tự nạp lại của Timer 2, dưới đây là chương trình cụ thể:

```
#include<reg52.h>
sbit F = P1^1;
void main(void)
{
 T2MOD = 0X03;                 //DCEN = 1, Timer đếm tiến
 RCAP2H = TH2 = -2500/256;     //hệ số chia là 5000
 RCAP2L = TL2 = -2500%256;
 TR2=1;
while(1)
{
 while (!TF2);                 //chờ cờ tràn
 TF2=0;                        //xoá cờ tràn
 F=~F;                         //đảo mức trên P1.1
}
}
```

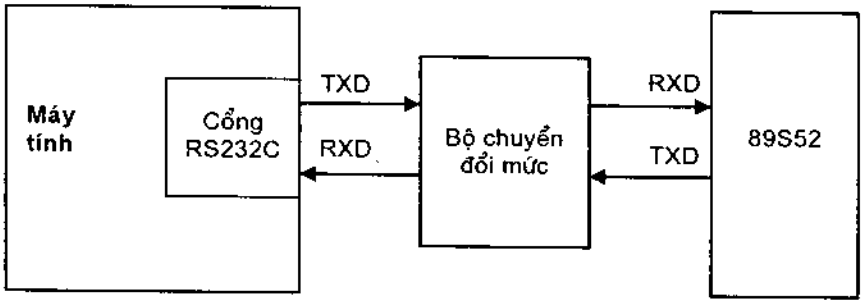
2.5. CỔNG NỐI TIẾP

2.5.1. Giới thiệu

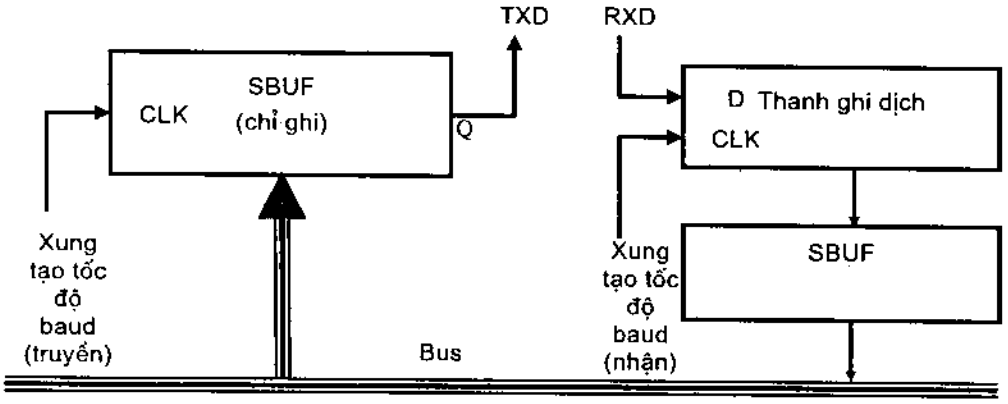
AT89S52 có một cổng nối tiếp trên chip có thể hoạt động ở nhiều chế độ khác nhau với các tốc độ khác nhau. Chức năng chủ yếu của cổng nối tiếp là thực hiện chuyển đổi song song sang nối tiếp với dữ

liệu xuất và chuyển đổi nối tiếp sang song song với dữ liệu nhập để có thể giao tiếp với máy tính (hình 2.30) qua cổng nối tiếp hoặc các thiết bị tương tự.

Cổng nối tiếp có thể hoạt động song công (full duplex : thu và phát đồng thời) và đệm lúc thu (receiver buffering) cho phép một ký tự sẽ được thu và được giữ trong khi ký tự thứ hai được nhận. Nếu CPU đọc ký tự thứ nhất trước khi ký tự thứ hai được thu đầy đủ thì dữ liệu sẽ không bị mất.



Hình 2.30. Mô tả hoạt động của cổng nối tiếp.



Hình 2.31. Sơ đồ khối cổng nối tiếp của 8051.

2.5.2. Các thanh ghi của cổng nối tiếp

Có hai thanh ghi chức năng đặc biệt cho phép phần mềm truy xuất đến cổng nối tiếp là SBUF và SCON.

Thanh ghi điều khiển cổng nối tiếp (SCON–Serial Controller).

Thanh ghi điều khiển cổng nối tiếp (SCON) ở địa chỉ 98H là thanh ghi có định địa chỉ bit, chứa các bit trạng thái và các bit điều khiển

liên quan tới cổng nối tiếp. Các bit điều khiển đặt chế độ hoạt động cho cổng nối tiếp, các bit trạng thái báo cáo kết thúc việc phát hoặc thu một ký tự. Các bit trạng thái có thể được kiểm tra bằng phần mềm hoặc có thể được lập trình để tạo ngắt.

Bit	Ký hiệu	Địa chỉ	Mô tả
SCON.7	SM0	9FH	Serial Mode 0 – Bit 0 chọn chế độ cho cổng nối tiếp.
SCON.6	SM1	9EH	Serial Mode 1 – Bit 1 chọn chế độ cho cổng nối tiếp. SM0SM1 = 00: Cổng nối tiếp hoạt động ở chế độ 0. SM0SM1 = 01: Cổng nối tiếp hoạt động ở chế độ 1. SM0SM1 = 10: Cổng nối tiếp hoạt động ở chế độ 2. SM0SM1 = 11: Cổng nối tiếp hoạt động ở chế độ 3.
SCON.5	SM2	9DH	Serial Mode 2 – Bit 2 chọn chế độ cho cổng nối tiếp. Bit này cho phép truyền thông đa xử lý.
SCON.4	REN	9CH	Receive Enable – Bit cho phép thu, REN phải được đặt bằng 1 để cho phép nhận các ký tự.
SCON.3	TB8	9BH	Transmitted Bit 8 – Bit truyền thứ 9, sử dụng trong chế độ UART 9 bit.
SCON.2	RB8	9AH	Received Bit 8 – Bit nhận thứ 9, sử dụng trong chế độ UART 9 bit.
SCON.1	TI	99H	Transmitted Interrupt – Cờ ngắt truyền, TI được đặt bằng 1 bởi phần cứng khi kết thúc việc truyền 1 ký tự, TI được xoá bằng phần mềm.
SCON.0	RI	98H	Received Interrupt – Cờ ngắt truyền, RI được đặt bằng 1 bởi phần cứng khi kết thúc việc nhận 1 ký tự, RI được xoá bằng phần mềm.

Trước khi sử dụng cổng nối tiếp, phải khởi động SCON để chọn đúng chế độ.

Ví dụ câu lệnh SCON = 0×52 sẽ khởi động cổng nối tiếp cho chế độ 1 (SM0SM1 = 01), cho phép bộ thu (REN = 1) và đặt cờ ngắt phát (TI = 1) để chỉ bộ phát sẵn sàng hoạt động.

Thanh ghi đệm truyền nhận ở cổng nối tiếp (SBUF–Serial Buffer)

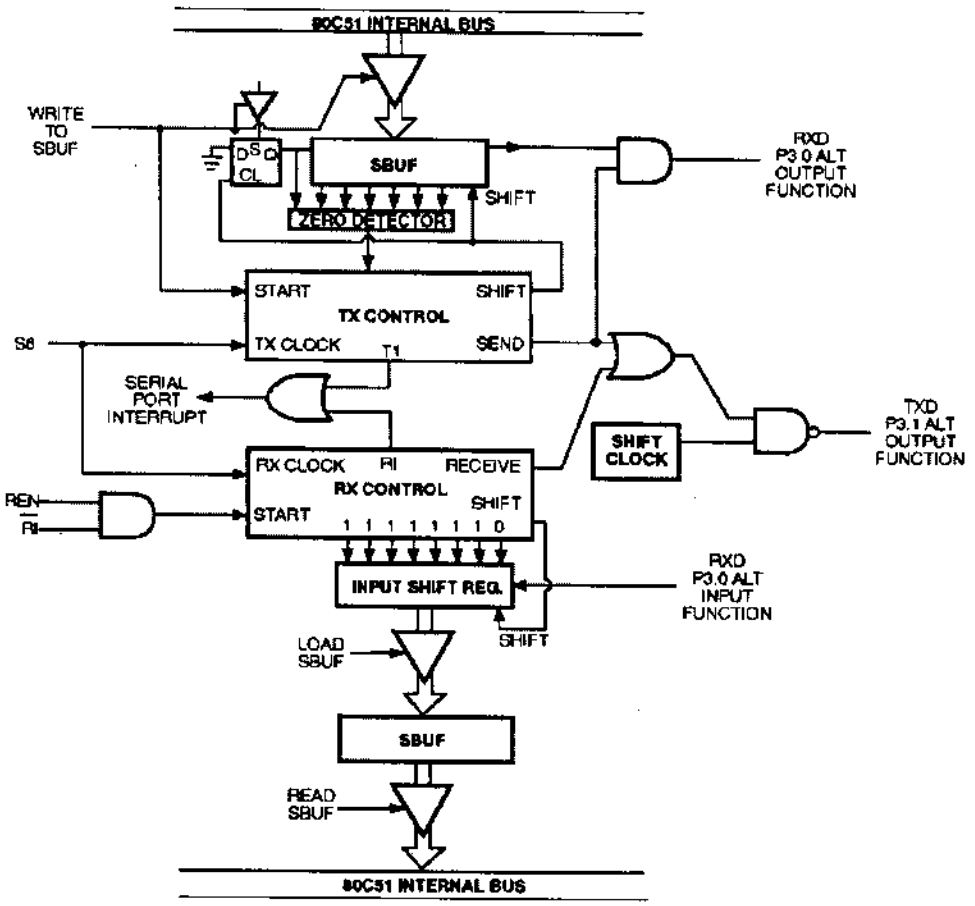
Thanh ghi này có chức năng đệm các ký tự khi chúng được nhận về từ cổng nối tiếp hoặc được truyền đi từ cổng nối tiếp, việc truyền nhận qua cổng nối tiếp thực chất là việc truy xuất thanh ghi này.

2.5.3. Các chế độ hoạt động

Chế độ 0

Chế độ 0 là chế độ mà cổng nối tiếp được dùng như một thanh ghi

dịch 8 bit. Dữ liệu được truyền/nhận nối tiếp trên chân RXD, chân TXD được dùng để phát xung clock dịch bit. Khi truyền/nhận các byte dữ liệu 8 bit, bit có giá trị thấp nhất (LSB) được truyền/nhận trước tiên và bit MSB được truyền/nhận sau cùng.

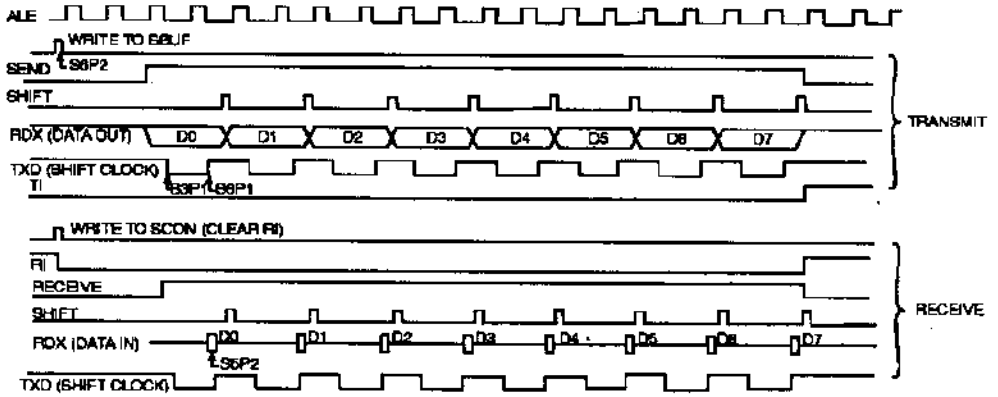


Hình 2.32. Hoạt động của cổng nối tiếp ở chế độ 0.

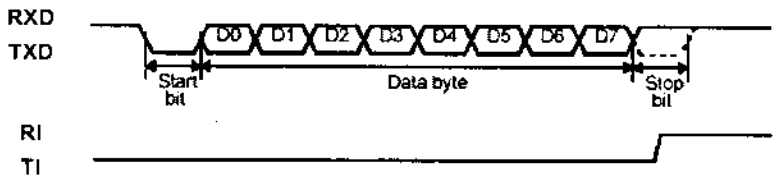
Việc truyền dữ liệu được bắt đầu bằng việc ghi 1 byte dữ liệu vào SBUF còn việc nhận dữ liệu được bắt đầu khi bit REN đã được đặt ở mức 1 và cờ thu RI = 0. Tốc độ baud ở chế độ 0 cố định bằng $F_{osc}/12$.

Chế độ 1

Trong chế độ 1, cổng nối tiếp hoạt động như một bộ UART 8 bit có tốc độ thay đổi. Dữ liệu được truyền nối tiếp trên chân TXD và nhận nối tiếp trên chân RXD, chế độ này cung cấp cho AT89S52 một công cụ giao tiếp với máy tính qua cổng COM.



Hình 2.33. Giản đồ truyền nhận dữ liệu ở chế độ 0.



Hình 2.34. Giản đồ truyền nhận dữ liệu ở chế độ 1.

Với chế độ 1, 1 khung truyền sẽ gồm 10 bit, ngoài 8 bit dữ liệu ra còn có 1 bit start (ở mức thấp) và 1 bit stop (ở mức cao), LSB cũng được truyền trước, MSM được truyền sau.

Tốc độ baud của cổng nối tiếp trong chế độ 1 có thể được cung cấp bởi Timer 1 hoặc Timer 2 hoặc đồng thời cả 2 bộ Timer nếu muốn tốc độ truyền và tốc độ nhận khác nhau (hình 2.35).

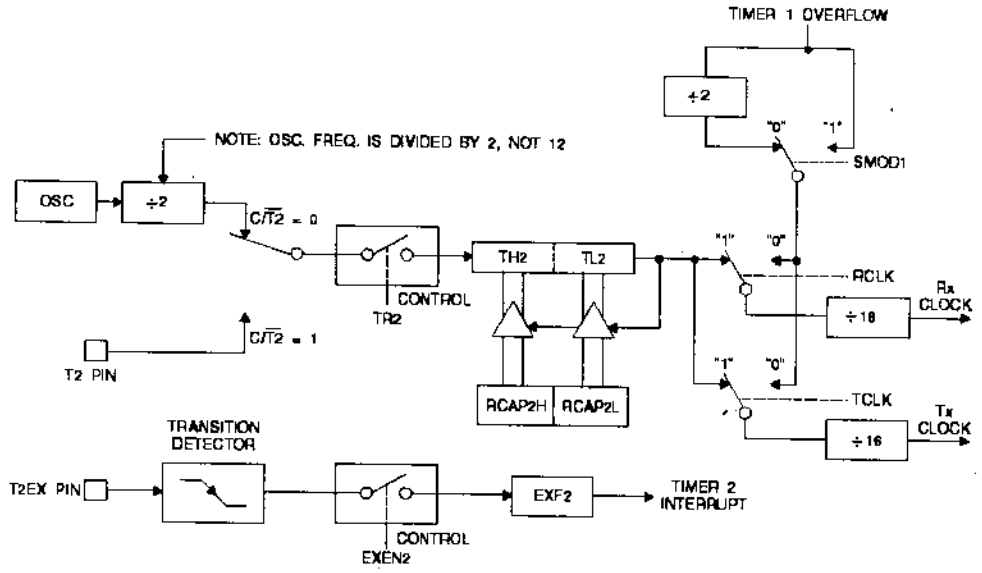
Khi sử dụng các bộ Timer cung cấp tốc độ baud cho cổng nối tiếp thì thạch anh có tần số 11,0592MHz được khuyến cáo nên dùng vì với tần số này sẽ tạo được các tốc độ baud chuẩn với sai số bằng 0.

Ví dụ muốn có tốc độ baud là 9600 thì cần có tốc độ tràn của Timer 1 là $f_1 = 9600 \times 32 = 307200$Hz. Nếu sử dụng thạch anh 11,0592MHz thì tần số của xung clock cấp cho Timer 1 sẽ là $f_2 = 11059200/12$Hz. Như vậy cần khởi tạo cho Timer 1 giá trị nhỏ hơn giá trị xảy ra tràn là $f_2/f_1 = 3$ nghĩa là Timer 1 sẽ được đặt ở chế độ 2 và giá trị nạp cho TH1 là –3.

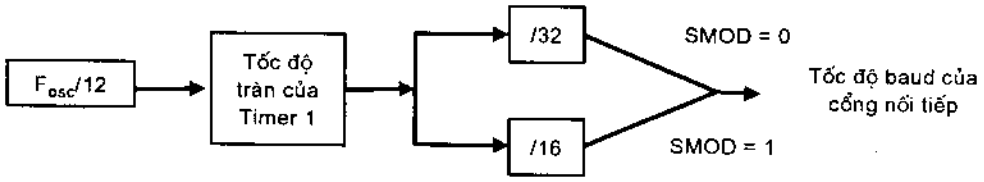
Trên hình 2.35, khi TCLK = RCLK = 1 thì tốc độ baud của cổng nối tiếp được cung cấp bởi Timer 2.

Khác với Timer 1, Timer 2 được cấp xung clock có tần số bằng 1/2 tần số của bộ dao động thạch anh, theo hình 2.37, giả sử cần tốc độ baud là 9600 thì giá trị nạp cho Timer 2 sẽ là $-(11059200/2)/(9600 \times 16) = -36$.

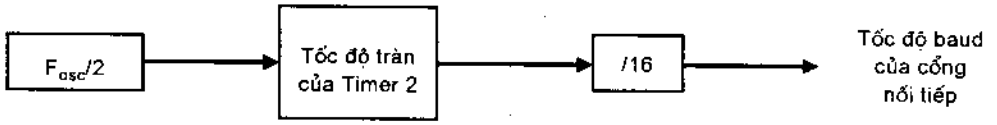
Khi cần có tốc độ baud khác nhau cho đường truyền và đường nhận thì có thể sử dụng cả 2 bộ Timer (hình 2.35). Nếu đặt TCLK = 1 và RCLK = 0 thì tốc độ baud của đường truyền sẽ được cung cấp bởi Timer 2, tốc độ baud của đường nhận sẽ được cung cấp bởi Timer 1. Nếu đặt TCLK = 0 và RCLK = 1 thì tốc độ baud của đường truyền sẽ được cung cấp bởi Timer 1, tốc độ baud của đường nhận sẽ được cung cấp bởi Timer 2.



Hình 2.35. Dùng Timer 1, 2 cung cấp tốc độ baud cho cổng nối tiếp.



Hình 2.36. Dùng Timer 1 cung cấp tốc độ baud cho cổng nối tiếp (TCLK = RCLK = 0).

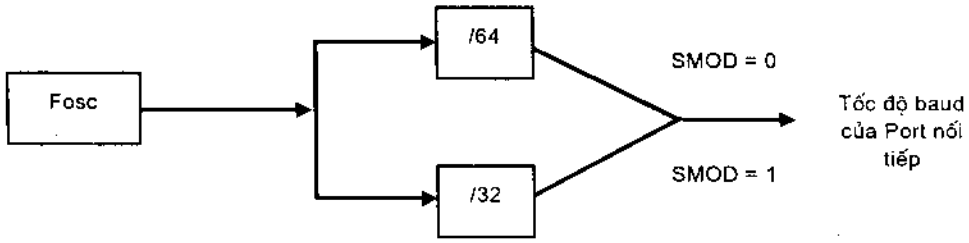


Hình 2.37. Dùng Timer 2 cung cấp tốc độ baud cho cổng nối tiếp.

Chế độ 2

Ở chế độ 2, cổng nối tiếp hoạt động như một bộ UART 9 bit, một khung truyền sẽ gồm 11 bit, trong đó bắt đầu là bit Start, tiếp theo là 8 bit dữ liệu, tiếp theo là bit dữ liệu thứ 9 (là bit TB8 nếu là khung truyền, là bit RB8 nếu là khung nhận), cuối cùng là bit Stop. Chế độ này thường được dùng khi cần chèn thêm bit kiểm tra chẵn lẻ vào trong khung truyền để giảm bit lỗi đường truyền.

Tốc độ baud trong chế độ 2 được tạo cố định như hình 2.38.



Hình 2.38. Tốc độ baud ở chế độ 2.

Chế độ 3

Chế độ 3 là sự kết hợp của chế độ 1 và chế độ 2, nghĩa là cổng nối tiếp hoạt động như 1 bộ UART 9 bit và tốc độ baud của UART là thay đổi giống như chế độ 1 (được cung cấp bởi Timer 1 và Timer 2).

2.5.4. Trao đổi dữ liệu qua cổng nối tiếp

Thao tác trao đổi dữ liệu qua cổng nối tiếp không đơn thuần chỉ là việc ghi/đọc dữ liệu như trao đổi dữ liệu trực tiếp qua các cổng mà còn bao gồm 3 thao tác chính như sau:

– Khởi tạo cổng nối tiếp:

+ Truy xuất SCON để đặt các thông số như chế độ hoạt động, cho phép thu...

+ Thiết lập hoặc xoá bit SMOD của thanh ghi PCON để đặt hệ số chia của tốc độ baud.

+ Truy xuất các thanh ghi của các bộ Timer 1 và Timer 2 để đặt tốc độ baud cho cổng nối tiếp (chỉ với chế độ 1 và 3).

– Kiểm tra cờ TI (khi truyền) và kiểm tra cờ RI (khi nhận).

– Ghi/đọc byte dữ liệu ở SBUF.

Chú ý : Nếu cổng nối tiếp hoạt động ở chế độ 2, 3 thì ngoài các thao tác trên còn có thao tác ghi/đọc bit dữ liệu thứ 9 vào/từ TB9/RB9.

Ví dụ 1: Truyền 1 byte qua Port nối tiếp.

```
#include<stdio.h>
#include<reg52.h>
char x;
void main(void)
{
 SCON = 0x52;              //Port nối tiếp chế độ 1,
                           // REN = TI = 1.
 TMOD = 0x20;              //Timer 1 mode 2
 TH1 = TL1 = -3;           //Tốc độ baud là 9600
 TR1 = 1;
 While ( !TI);             //chờ TI = 1
 TI=0;                     //xoá TI
 SBUF = x;                 //truyền byte dữ liệu trong biến x
}
```

Ví dụ 2: Nhận 1 byte qua Port nối tiếp.

```
#include<stdio.h>
#include<reg52.h>
char x;
void main(void)
{
 SCON = 0x52;              //Port nối tiếp chế độ 1,
                           //REN = TI = 1.
 TMOD = 0x20;              //Timer 1 mode 2
 TH1 = TL1 = -3;           //Tốc độ baud là 9600
 TR1 = 1;
 While (!RI) ;             //chờ RI = 1
 RI = 0 ;                  //xoá RI
 X = SBUF ;                //nhận byte dữ liệu chứa trong biến x
}
```

Trong hai ví dụ trên hệ số chia của tốc độ baud được mặc định là 32 do không có câu lệnh đặt SMOD = 1 (khi đó SMOD sẽ mặc định bằng 0).

Trong một chương trình khi có nhiều thao tác truyền/nhận qua cổng nối tiếp thì đoạn chương trình truyền hoặc nhận nên viết dưới dạng hàm, ví dụ:

```
void truyen(char x)
{
while(!TI);
TI = 0;
SBUF = x;
}
char nhan(void)
{
char C;
while(!RI);
TI = 0;
C = SBUF;
return C;
}
```

Trong thư viện của trình dịch C cũng có 2 hàm tương đương với hàm truyền và hàm nhận như trên, chúng là putchar () và _getkey ().

Ví dụ 3: Truyền nhận qua cổng nối tiếp với tốc độ baud khác nhau.

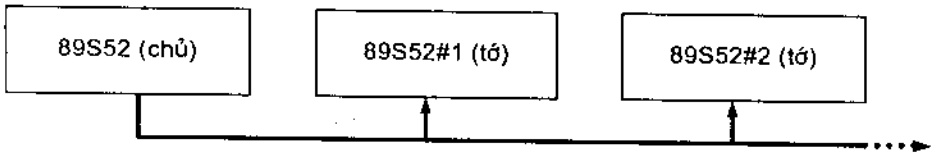
```
#include<stdio.h>
#include<reg52.h>
char x, M[10];
int i;
void main(void)
{
 SCON = 0x52;          //Port nối tiếp chế độ 1,
                       //REN = TI = 1.
 TMOD = 0x20;          //Timer 1 mode 2
 TH1 = TL1 = -6;       //Tốc độ baud là 4800
 TR1 = 1;
 T2CON = 0x20;
/*Timer 2 chế độ tự nạp lại, C-/T2 = 0;
TCLK = 0(Dùng Timer 1 tạo tốc độ baud cho đường truyền) ;
RCLK = 1 (Dùng Timer 2 tạo tốc độ baud cho đường nhận)*/
 RCAP2H = 0 ;
 RCAP2L = -36 ;        //Tốc độ baud là 9600
 TR2 = 1;
                       //Gửi 10 ký tự bắt đầu từ 'A'
x = 0x41;
for(i=1;i<=10,++i)
{
putchar(x)
++x;
}
                       //Nhận 10 ký tự lưu vào mảng M
for(i= 1;i<= 10,++i)
   M[i]=_getkey();
}
```

Ví dụ 4: Truyền thông qua cổng nối tiếp có kiểm tra chẵn lẻ.

```
#include<stdio.h>
#include<reg52.h>
char x;
bit i;
void main(void)
{
 SCON = 0xD2;          //Port nối tiếp chế độ 3,
                       //REN = TI = 1.
 TMOD = 0x20;          //Timer 1 mode 2
 TH1 = TL1 = -6;       //Tốc độ baud là 4800
 TR1 = 1;
                       //Truyền 1 khung có kèm theo bit kiểm tra chẵn.
 A = x;                //Đưa byte cần truyền vào thanh chứa
 I = P;                //Cập nhật bit kiểm tra chẵn
 TB8 = i;              //Đưa vào TB8 để truyền
 putchar(x);           //Truyền byte dữ liệu chứa trong x
```

Ví dụ 5: Truyền thông đa xử lý.

Chế độ 2 và chế độ 3 của cổng nối tiếp cung cấp cho bộ vi điều khiển khả năng truyền thông đa xử lý (hình 2.39).



Hình 2.39. Truyền thông đa xử lý.

Một bộ vi điều khiển chủ có thể truyền dữ liệu tới nhiều bộ vi điều khiển tớ bằng cách gửi các byte địa chỉ (các byte có RB8 = 1) mang mã của các bộ vi điều khiển tớ trước khi gửi các byte dữ liệu (các byte có RB8 = 0). Các bộ vi điều khiển tớ sẽ lập trình với bit SM2 = 1 (xem thanh ghi SCON) để phân biệt được đâu là byte địa chỉ, đâu là byte dữ liệu. Sau khi nhận được byte địa chỉ của mình (nghĩa là bộ vi điều khiển tớ tương ứng đã được bộ vi điều khiển chủ đánh địa chỉ tới), bộ vi điều khiển tớ sẽ xoá bit SM2 để có thể nhận các byte dữ liệu bình thường. Dưới đây là ví dụ cụ thể:

Chương trình viết cho bộ vi điều khiển chủ:

```
#include<stdio.h>
#include<reg52.h>
char x;
bit i;
void main(void)
{
  SCON = 0xF2;
/* Port nối tiếp chế độ 3,truyền đa xử lý
  SM2 = REN = TI = 1*/
 TMOD = 0x20;       //Timer 1 mode 2
 TH1 = TL1 = -6;    //Tốc độ baud là 4800
 TR1 = 1;
/*gửi đi địa chỉ của bộ vi điều khiển tớ thứ nhất*/
TB8 = 1 ;
putchar(0x01) ;     //mã của bộ vi điều khiển thứ nhất
/*gửi cho bộ vi điều khiển tớ thứ nhất 10 byte dữ liệu*/
X = 0x41;
for(i = 1 ;i<= 10,++i)
{
TB = 0;
putchar(x) ;
++x;
}
/*gửi đi địa chỉ của bộ vi điều khiển tớ thứ hai*/
TB8 = 1 ;
putchar(0x02) ;     //mã của bộ vi điều khiển thứ nhất
```

```
/*gửi cho cả hai bộ vi điều khiển 10 byte dữ liệu*/
x = 0x57;
for(i = 1;i<= 10,++i)
{
TB = 0;
putchar(x) ;
--x;
}
```

Chương trình viết cho bộ vi điều khiển tớ thứ nhất:

```
#include<stdio.h>
#include<reg52.h>
char x,y;
void main(void)
{
 SCON = 0xF2;          //Port nối tiếp chế độ 3,
                       //SM2 = REN = TI = 1.
 TMOD = 0x20;          //Timer 1 mode 2
 TH1 = TL1 = -6;       //Tốc độ baud là 4800
 TR1 = 1;
                       //Chờ nhận byte địa chỉ của mình
do
x = _getkey() ;
while(x = 0x01) ;
SM2 = 0;
do                     //Chỉ nhận các byte dữ liệu
y = _getkey();
while (RB8 = 0);
}
```

Chương trình viết cho bộ vi điều khiển tớ thứ hai:

```
#include<stdio.h>
#include<reg52.h>
char x,z;
bit i;
void main(void)
{
 SCON = 0xF2;          //Port nối tiếp chế độ 3,
                       //SM2 = REN = TI = 1.
 TMOD = 0x20;          //Timer 1 mode 2
 TH1 = TL1 = -6;       //Tốc độ baud là 4800
 TR1 = 1;
                       //Chờ nhận byte địa chỉ của mình
do
x = _getkey() ;
while(x = 0x02) ;
SM2 = 0;
do                     //Chỉ nhận các byte dữ liệu
y = _getkey();
while (RB8 = 0);
}
```

2.6. NGẮT VÀ XỬ LÝ NGẮT

2.6.1. Khái niệm sự cần thiết phải ngắt CPU

Trong thực tế người ta rất muốn tận dụng khả năng của CPU để làm thêm được nhiều công việc khác nữa, chỉ khi nào có cần trao đổi dữ liệu thì mới yêu cầu CPU tạm dừng công việc hiện tại để phục vụ việc trao đổi dữ liệu. Sau khi hoàn thành việc trao đổi dữ liệu thì CPU lại phải quay về để làm tiếp công việc hiện đang bị gián đoạn. Cách làm việc theo kiểu này gọi là *ngắt CPU (gián đoạn hoạt động của CPU)*. Một hệ thống sử dụng *ngắt* có thể đáp ứng rất nhanh các yêu cầu trao đổi dữ liệu trong khi vẫn có thể làm được các công việc khác.

2.6.2. Tổ chức ngắt ở AT89S52

AT89S52 có 6 nguồn ngắt:

– Ngắt ngoài đến từ chân #INT0.

– Ngắt ngoài đến từ chân #INT1.

– Ngắt do bộ Timer 0.

– Ngắt do bộ Timer 1.

– Ngắt do bộ Timer 2.

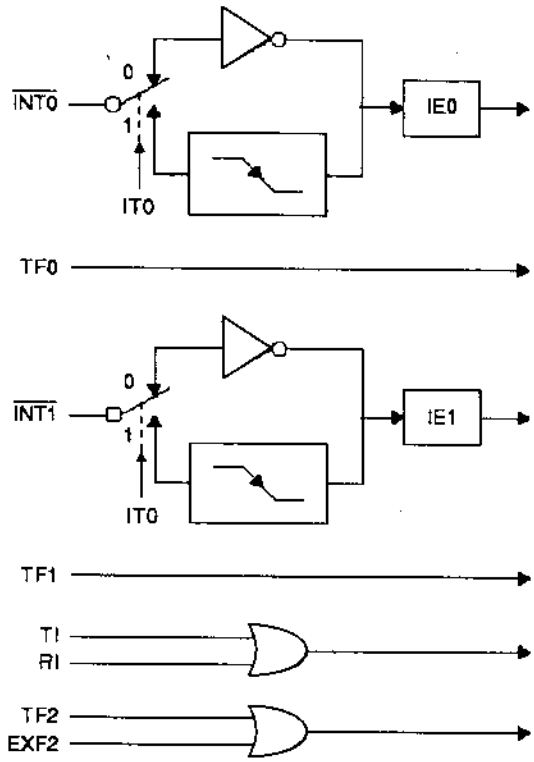
– Ngắt do Port nối tiếp.

6 nguồn ngắt này được xoá khi Reset và được đặt riêng bằng phần mềm bởi các bit trong các thanh ghi cho phép ngắt (IE), thanh ghi ưu tiên ngắt (IP).

* *Thanh ghi cho phép ngắt IE (Interrupt Enable):*

EA	–	ET2	ES	ET1	EX1	ET0	EX0

Bit	Ký hiệu	Địa chỉ bit	Mô tả (1: cho phép; 0: cấm)
IE.7	EA	AFH	Cho phép hoặc cấm toàn bộ.
IE.6	–	AEH	Không được định nghĩa.
IE.5	ET5	ADH	Cho phép ngắt từ Timer 2 (8052).
IE.4	ES	ACH	Cho phép ngắt Port nối tiếp.
IE.3	ET1	ABH	Cho phép ngắt từ Timer 1.
IE.2	EX1	AAH	Cho phép ngắt ngoài 1.
IE.1	ET0	A9H	Cho phép ngắt từ Timer 0.
IE.0	EX0	A8H	Cho phép ngắt ngoài 0.



Hình 2.40. Các nguồn ngắt của AT89S52.

** Thanh ghi ưu tiên ngắt IP:*

Mỗi nguồn ngắt được lập trình riêng để đặt vào một trong hai mức ưu tiên qua thanh ghi chức năng đặc biệt được địa chỉ bit IP (Interrupt priority: ưu tiên ngắt) ở địa chỉ B8H.

–	–	PT2	PS	PT1	PX1	PT0	PX0

Bit	Ký hiệu	Địa chỉ bit	Mô tả (1: mức cao; 0: mức thấp)
IP.7	–		Không được định nghĩa.
IP.6	–		Không được định nghĩa.
IP.5	PT2	BDH	Ưu tiên cho ngắt từ Timer 2.
IP.4	PS	BCH	Ưu tiên cho ngắt Port nối tiếp.
IP.3	PT1	BBH	Ưu tiên cho ngắt từ Timer 1.
IP.2	PX1	BAH	Ưu tiên cho ngắt ngoài 1.
IP.1	PT0	B9H	Ưu tiên cho ngắt từ Timer 0.
IP.0	PX0	B8H	Ưu tiên cho ngắt ngoài 0.

Các vector ngắt:

Khi một ngắt nào đó được chấp nhận, giá trị được nạp vào PC được gọi là Vector ngắt. Nó là địa chỉ bắt đầu của chương trình con phục vụ ngắt ISR (ISR viết tắt của Interrupt Service Routine – chương trình con phục vụ ngắt) tương ứng với nguồn tạo ngắt. Các vector ngắt được cho liệt kê như sau:

Ngắt	Cờ	Địa chỉ vector	Số hiệu
Reset hệ thống	RST	0000H	
Bên ngoài 0	IE0	0003H	0
Timer 0	TF0	000BH	1
Bên ngoài 1	IE1	0013H	2
Timer 1	TF1	001BH	3
Port nối tiếp	TI hoặc RI	0023H	4
Timer 2	TX2 hoặc EXF2	002BH	5

Khi chỉ đến một ngắt, cờ gây ngắt tự động bị xóa bởi phần cứng, ngoại trừ RI và TI phải được xoá bởi phần mềm.

2.6.3. Các thiết kế sử dụng ngắt

1. Ngắt do các bộ Timer

Các ngắt do các bộ Timer xảy ra do sự kiện tràn ở các Timer, khi đó các cờ tràn TFx sẽ được đặt bằng 1. Khi ISR được đáp ứng, các cờ TFx sẽ tự động được xoá bởi phần mềm.

Khuôn mẫu một chương trình có sử dụng ngắt viết bằng ngôn ngữ lập trình C như sau:

```
#include <stdio.h>
#include <reg52.h>
...
//Chương trình chính
void main (void)
{
//Khởi tạo các thanh ghi cho phép ngắt và ưu tiên ngắt
...
}
//Các chương trình con phục vụ ngắt (ISR)
void ngatT0(đối số)interrupt x
{
//Các câu lệnh trong ISR
}
```

Trong đó: ngatT0 là tên của ISR (tuỳ chọn), x là số hiệu ngắt tương ứng với vector ngắt.

Ví dụ 1: Tạo 2 xung khác nhau trên các chân P1.0 và P2.0.

```
#include <stdio.h>
#include <reg52.h>
sbit xung1 = P1^0;
sbit xung2 = P2^0;
                        //Chương trình chính
void main (void)
{
TMOD = 0x21;            //Timer 0 mode 1; Timer 1 mode 2
TH1 = TL1 = -100 ;      //xung có chu kỳ 200 µs
TR1 = 1;
/*Khởi tạo các thanh ghi cho phép ngắt và ưu tiên ngắt*/
IE = 0x8A;              //cho phép ngắt do Timer 0 và Timer 1
IP = 0;                 //mức ưu tiên bằng nhau
TF0 = 1;                //buộc ngắt do Timer 0
while(1);
}
                        //Các chương trình con phục vụ ngắt (ISR)
void ngatT0 (void) interrupt 1
{
TR0 = 0;
TH0 = -10000/256;       //xung có chu kỳ 20000 µs
TL0 = -10000%256;
xung2 = ~xung2;
TR0 = 1;
}
void ngatT1 (void) interrupt 3
{
xung1 = ~xung1;
}
```

Ví dụ 2: Tạo 3 xung khác nhau trên các chân P1.0, P1.1 và P1.2.

```
#include <stdio.h>
#include <reg52.h>
sbit xung1 = P1^0;
sbit xung2 = P1^1;
sbit xung3 = P1^2
                        //Chương trình chính
void main (void)
{
T2MOD = 0x02;           //DCEN = 0, Timer 2 đếm tiến
RCAP2H = -25000/256;    //xung có chu kỳ 50000 µs
RCAP2L = -25000%256;
TR2 =1;
TMOD = 0x21;            //Timer 0 mode 1; Timer 1 mode 2
TH1 = TL1 = -100;       //xung có chu kỳ 200 µs
TR1 = 1;
/*Khởi tạo các thanh ghi cho phép ngắt và ưu tiên ngắt*/
IE = 0xAA; /*cho phép ngắt do Timer 0 Timer 1 và Timer 2 */
IP = 0;                 //mức ưu tiên bằng nhau
TF0 = 1;                //buộc ngắt do Timer 0
while(1);
}
                        //Các chương trình con phục vụ ngắt (ISR)
```

```
void ngatT0 (void) interrupt 1
{
TR0 = 0;
TH0 = -10000/256;        //xung có chu kỳ 20000 µs
TL0 = -10000%256;
xung2 = ~xung2;
TR0 = 1;
}
void ngatT1 (void) interrupt 3
{
xung1 = ~xung1;
}
void ngatT2 (void) interrupt 5
{
xung3 = ~xung3;
}
```

2. Ngắt do cổng nối tiếp

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

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

Ví dụ 1: Viết chương trình dùng ngắt gửi đi 24 ký tự từ A đến Z trong bảng mã ASCII ra cổng nối tiếp của vi điều khiển.

```
#include <stdio.h>
#include <reg52.h>
char x;
                          //Chương trình chính
void main (void)
{
TMOD = 0x20;              //Timer 1 mode 1.
TH1 = TL1 = -24;          //tốc độ baud là 1200
TR1 = 1;
SCON = 0x52;              //UART8 bit, chế độ 1,TI = 1
x = 0x41 ;
IE = 0x90 ;               //cho phép ngắt do port nối tiếp
while(1) ;
}
void ngatnt(void) interrupt 4
{
if(x<='Z')
{
SBUF = x;
++x;
TI = 0;
}
}
```

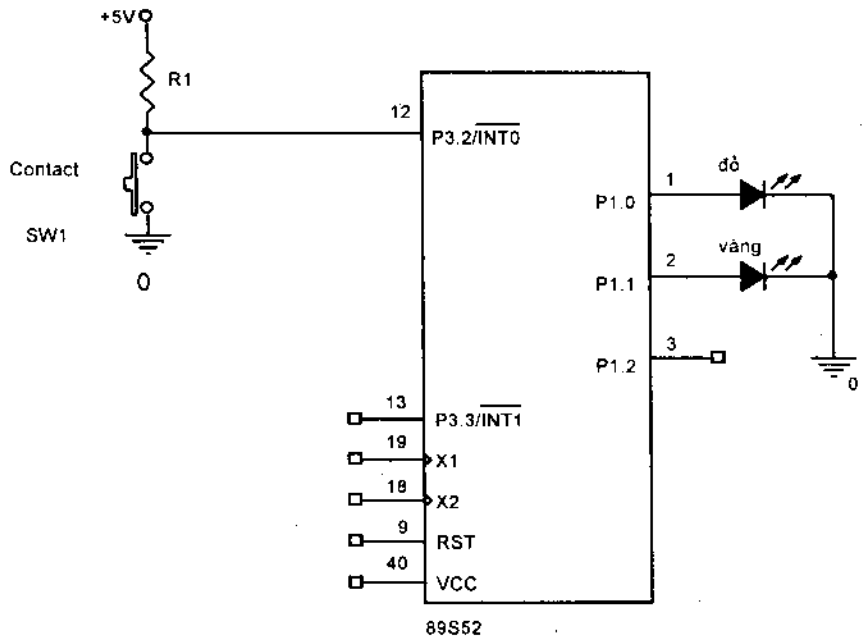
3. Ngắt ngoài

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

Các cờ tạo ngắt này là các bit IE0 và IE1 trong TCON. Khi quyền điều khiển đã chuyển đến ISR, cờ tạo ra ngắt chỉ được xóa nếu ngắt được tích cực bằng cạnh xuống, nếu ngắt được tích cực theo mức, thì nguồn yêu cầu ngắt bên ngoài sẽ điều khiển mức của cờ thay cho phần cứng.

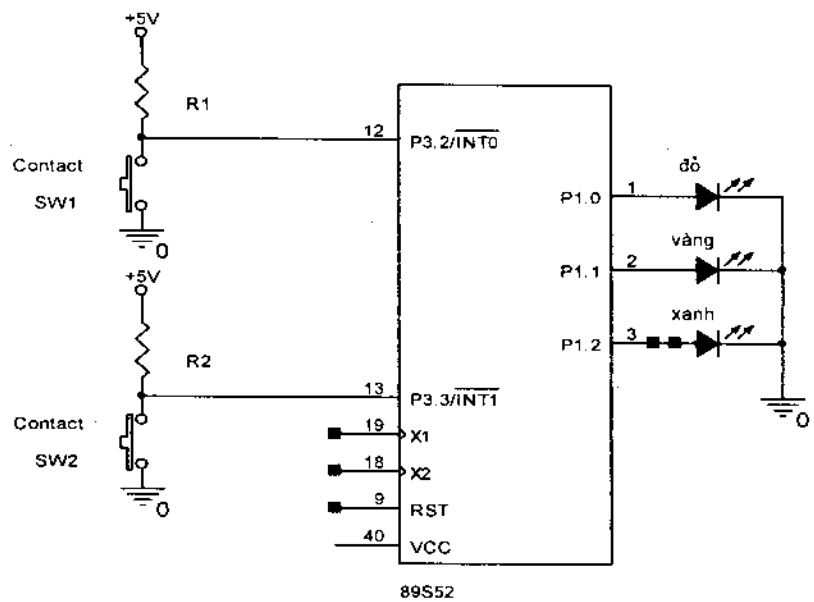
Cách thức tích cực ngắt được đặt bởi các bit ITx trong thanh ghi SCON, nếu Itx = 0, ngắt được tích cực bằng mức thấp, nếu Itx = 1, ngắt được tích cực bằng cạnh xuống (sườn âm). Nếu ngắt ngoài được tác động bằng cạnh xuống thì nguồn bên ngoài phải giữ chân /INTx ở mức cao tối thiểu trong một chu kỳ máy và giữ nó ở mức thấp trong một chu kỳ máy để đảm bảo cho CPU phát hiện được cạnh xuống. Nếu ngắt ngoài được tác động theo mức thì nguồn bên ngoài phải giữ tín hiệu yêu cầu tác động trên chân /INTx (ở mức thấp) cho đến khi ngắt được đáp ứng và không tác động nữa khi ISR đã được hoàn tất, nếu không một ngắt khác sẽ được lặp lại.

Ví dụ 1: Viết chương trình điều khiển LED vàng nhấp nháy liên tục, LED đỏ nhấp nháy 10 lần mỗi khi nhấn Contact.



Hình 2.41. Ví dụ 1.

```
#include <stdio.h>
#include <reg52.h>
int i;
sbit vang = P1^1;
sbit do = P1^0;
void delay(int t)
{
for(i = 1;i< =t;++i);
}
                              //Chương trình chính
void main (void)
{
IE = 0x81;                    //cho phép ngắt ngoài INT0.
                              //LED vàng nhấp nháy liên tục
while(1)
{
vang = 1;
delay(10000);
vang = 0;
delay(10000);
}
void ngat0(void) interrupt 0
{
for(i = 1;i< = 10;++i)
{
do = 1;
delay(10000);
do = 0;
delay(10000);
}
}
```



Hình 2.42. Ví dụ 2.

Ví dụ 2: Viết chương trình điều khiển LED vàng nhấp nháy liên tục, LED đỏ nhấp nháy 10 lần mỗi khi nhấn Contact 1, LED xanh nhấp nháy 5 lần mỗi khi nhấn Contact 2.

```
#include <stdio.h>
#include <reg52.h>
int i;
sbit xanh = P1^2;
sbit vang = P1^1;
sbit do = P1^0;
void delay(int t)
{
for(i=1;i<=t;++i);
}
                              //Chương trình chính
void main (void)
{
IE = 0x85;                    //cho phép ngắt ngoài INT0, INT1.
                              //LED vàng nhấp nháy liên tục
while(1)
{
vang = 1;
delay(10000);
vang = 0;
delay(10000);
}
void ngat0(void) interrupt 0
{
for(i = 1;i<= 10;++i)
{
do = 1;
delay(10000);
do = 0;
delay(10000);
}
}
void ngat1(void) interrupt 3
{
for(i = 1;i<= 5;++i)
{
xanh = 1;
delay(10000);
xanh = 0;
delay(10000);
}
}
```

Chương 3 THIẾT KẾ ỨNG DỤNG VỚI HỌ VI ĐIỀU KHIỂN 8051

3.1. QUY TRÌNH THIẾT KẾ ỨNG DỤNG VI ĐIỀU KHIỂN

Tương tự như thiết kế hệ thống sử dụng vi mạch số lập trình được như PAL, GAL, CPLD/FPGA..., việc thiết kế ứng dụng dùng vi điều khiển cũng có những bước cơ bản sau:

Bước 1: Phân tích bài toán.

Mục đích của việc phân tích bài toán là để xác định các yêu cầu kỹ thuật như yêu cầu về khả năng tính toán (tốc độ và độ chính xác); số đầu vào/ra số; số đầu vào/ra tương tự; số lượng bộ đếm/định thời; ước lượng dung lượng bộ nhớ chương trình, bộ nhớ dữ liệu cần thiết và các yêu cầu khác như các kênh PWM, truyền thông qua RS232, USB, CAN, LIN...

Bước 2: Lựa chọn vi điều khiển và các linh kiện ngoài.

Các tiêu chí chính lựa chọn bộ vi điều khiển là:

– Hiệu năng: bộ vi điều khiển 8 bit/16 bit hay 32 bit, tần số xung nhịp bao nhiêu MHz?

– Dung lượng bộ nhớ chương trình, bộ nhớ số liệu;

– Số lượng đầu vào/ra;

– Số lượng bộ đếm/định thời;

– Số lượng, số bit và tốc độ biến đổi của các bộ biến đổi A/D, D/A;

– Kiểu đóng vỏ (liên quan đến việc thiết kế mạch in);

– Công suất tiêu thụ, điều này đặc biệt quan trọng với các ứng dụng sử dụng pin, ắc quy;

– Kiểu bộ nhớ chương trình là ROM, OTP, EPROM hay Flash ROM. Trong quá trình tạo mẫu hoặc sản xuất với số lượng nhỏ người ta thường chọn vi điều khiển có bộ nhớ chương trình là Flash ROM hoặc NV–RAM, tuy nhiên khi sản suất hàng loạt để giảm giá thành sản phẩm người ta thường sử dụng vi điều khiển có bộ nhớ chương trình là ROM hoặc OTP.

Bước 3: Xây dựng sơ đồ nguyên lý.

Sau bước 2 ta đã lựa chọn được vi điều khiển và các linh kiện ngoài

phụ trợ, ta tiến hành thiết kế mạch nguyên lý cho ứng dụng, việc thiết kế có thể được thực hiện bằng các phần mềm hỗ trợ như ORCAD, PROTEL, ALTIUM...

Bước 4: Lập trình cho vi điều khiển.

Căn cứ vào yêu cầu của bài toán tiến hành lập lưu đồ thuật toán và viết chương trình điều khiển.

Bước 5: Biên dịch chương trình.

Sử dụng trình dịch để biên dịch tập tin nguồn (tập tin .asm hoặc .c hoặc .cpp) thành tập tin .hex theo chuẩn của Intel. Đối với vi điều khiển họ 8051 ta có thể dùng trình dịch Keil C51 của Keil để biên dịch các tập tin nguồn viết bằng ngôn ngữ C. Đối với vi điều khiển họ AVR ta có thể dùng AVR Studio hoặc CodeVisionAVR để soạn và biên dịch chương trình.

Bước 6: Chạy thử nghiệm trên máy tính.

Sử dụng các phần mềm hỗ trợ thiết kế ứng dụng có linh kiện lập trình như Altium, Proteus... để chạy mô phỏng ứng dụng sau khi đã nhập sơ đồ nguyên lý và tập tin .hex cho vi điều khiển.

Nếu kết quả chạy thử nghiệm trên máy tính không đáp ứng được các yêu cầu bài toán thì phải quay lại bước 4 để xem lại chương trình, nếu không phải do chương trình phải quay lại kiểm tra từ bước 3.

Bước 7: Thiết kế, làm mạch in và lắp ráp linh kiện trên mạch in.

Bước 8: Nạp chương trình vào vi điều khiển.

Việc nạp chương trình cho vi điều khiển thực chất là ghi các thông tin trong tập tin .hex vào bộ nhớ chương trình của vi điều khiển. Việc nạp chương trình vào vi điều khiển thường được thực hiện bằng bộ lập trình chuyên dụng (Programmer). Nếu vi điều khiển hỗ trợ khả năng lập trình trực tiếp trên hệ thống (ISP, SPI) thì ta có thể nạp chương trình cho vi điều khiển ngay trên bản mạch ứng dụng đã gắn vi điều khiển. Một cách đơn giản, để thực hiện việc nạp chương trình xuống vi điều khiển có hỗ trợ ISP như 89S51, 89S52 bạn đọc có thể sử dụng chương trình và mạch nạp qua cổng song song của máy tính trên trang web www.kmitl.ac.th/~kswichit/ISP-Pgm3v0/ISP-Pgm3v0.html (nối dây vào các chân chức năng ISP của vi điều khiển như trong hình 2.17).

Bước 9: Chạy thử nghiệm hệ thống.

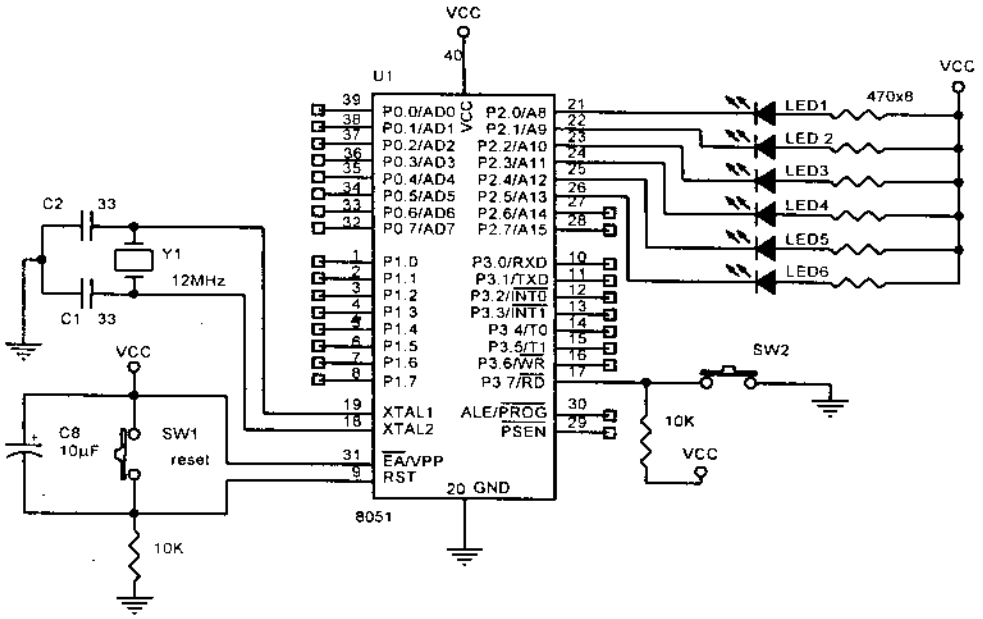
Việc chạy thử nghiệm hệ thống để kiểm tra xem hệ thống được thiết kế, lắp đặt có hoạt động và đạt các yêu cầu đề ra hay không.

3.2. CÁC VÍ DỤ MINH HOẠ

3.2.1. Xuất nhập với các cổng

1. Xuất nhập trực tiếp qua các cổng P0, P1, P2, P3

Đối với 8051, tất cả các cổng P0, P1, P2, P3 đều có chức năng xuất nhập theo bit và theo byte. Hình 3.1 minh hoạ mạch ghép nối giữa vi điều khiển 8051 với một nút nhấn và sáu LED.



Hình 3.1. Mạch điều khiển đèn LED.

*** Chương trình điều khiển các LED sáng tuần tự**

```
#include<reg51.h>
#include<stdio.h>
#include<intrins.h>
int j;
unsigned char x;          //x là một biến 8 bit
                          //thủ tục tạo trễ
void delay()
      {
      unsigned long int i;
      for (i = 1;i<= 5000;++i);
      }
void main (void)
      {
      while(1)
           {
```

```
            x = 0xfe;
            for (j = 1;j<= 5;++j)
            {
            P2 = x;
            delay();
            x =_crol_(x,1);/*quay x để các LED sáng tuần tự*/
            }
      }
}
```

*** Chương trình điều khiển các LED sáng tuần tự khi bấm vào nút nhấn SW2**

```
#include<reg51.h>
#include<stdio.h>
#include<intrins.h>
sbit contac=P3^7;/*khai báo biến kiểu bit phản ánh trạng
thái chân P3.7*/
sbit a;
unsigned char x;
int j;
void delay()
      {
      unsigned long int i;
      for (i = 1;i< =5000;++i);
      }
void main (void)
      {
      while(1)
      {
      x = 0xfe;
      contac = 1;
      a = contac;        //đọc trạng thái của nút nhấn
      if (a==0)
      for (j = 1;j< = 5;++j)
           {
           P2 = x;
           delay();
           x =_crol_(x,1);
           }
      P2 = 0;
      delay();
      P2 = 0xff;
      delay();
      }
      }
```

2. Truyền thông với máy tính qua cổng nối tiếp

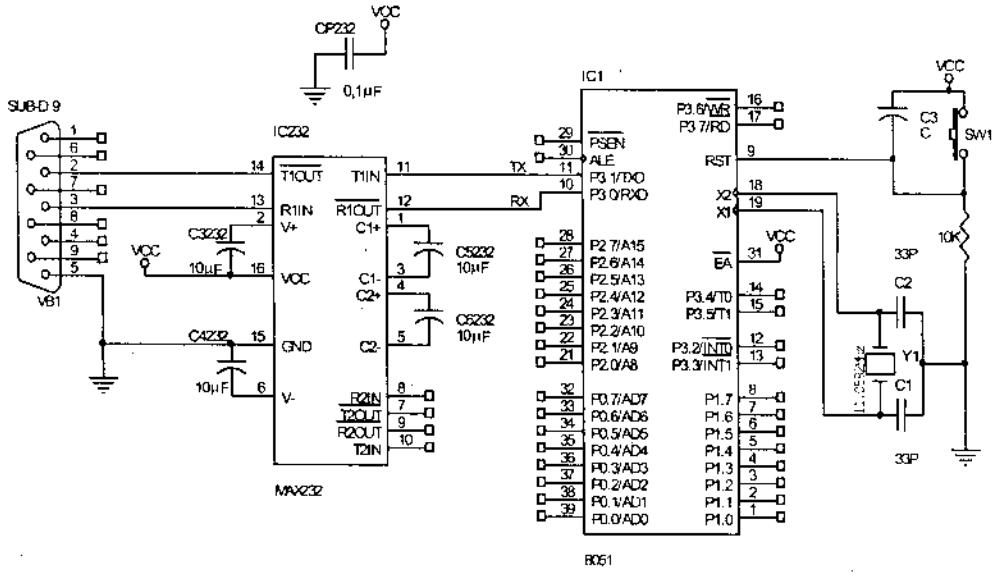
Như trong nội dung chương 2 đã khảo sát, bộ vi điều khiển AT89S52 có khả năng giao tiếp với thế giới bên ngoài qua cổng nối tiếp. Vấn đề trở

ngại duy nhất khi giao tiếp với máy tính là mức logic ở bộ vi điều khiển và ở cổng COM của máy tính khác nhau, cụ thể như sau:

BẢNG 3.1. SO SÁNH ĐIỆN ÁP CỦA CÁC MỨC LOGIC GIỮA RS232C VÀ TTL

Đối tượng	Mức logic	Mức điện áp tương ứng
Cổng COM (Mức RS232C)	1	–12V đến –3V
	0	+3V đến +12V
Vi điều khiển (Mức TTL)	1	+5V
	0	0V

Khắc phục vấn đề này, người ta sử dụng vi mạch MAX232 để chuyển đổi mức điện áp giữa hai chuẩn. Vi mạch này có chứa hai bộ chuyển đổi mức logic từ TTL sang RS232 và ngược lại. Hình 3.2 là mạch giao tiếp giữa vi điều khiển với máy tính qua cổng RS232C sử dụng vi mạch đổi mức MAX232.



Hình 3.2. Truyền thông qua cổng nối tiếp.

Sau đây là ví dụ về một chương trình nạp cho vi điều khiển để vi điều khiển truyền cho máy tính qua cổng nối tiếp các ký tự từ '0' tới '9'.

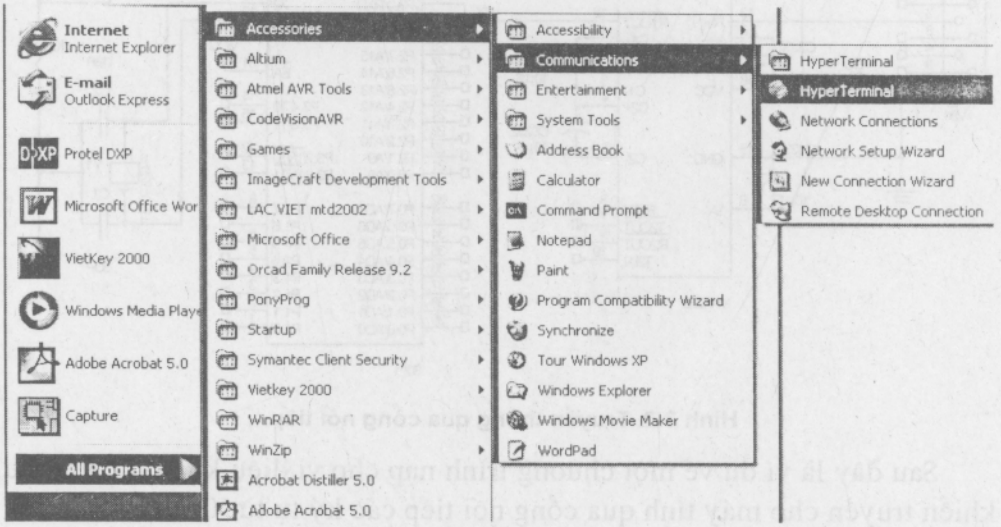
```
#include<stdio.h>
#include<reg52.h>
char x;
void main(void)
{
 SCON = 0x52; //Port nối tiếp chế độ 1, REN - TI = 1
 TMOD = 0x20; //Timer 1 mode 2
```

```
 TH1 = TL1 = -3;          //Tốc độ baud là 9600
 TR1 = 1;
 for(x = 0x30;x<0x39;x++)
  {
While ( !TI) ;           //chờ TI = 1
TI = 0 ;                 //xoá TI
SBUF = x ;               //truyền ký tự có mã ASCII trong biến x
  }
}
```

Ta biết rằng việc truyền một ký tự qua cổng nối tiếp thực chất là truyền mã ASCII của ký tự. Trong chương trình trên để gửi cho máy tính các ký tự từ '0' đến '9' ta phải truyền mã ASCII của chúng lần lượt từ 0x30 đến 0x39.

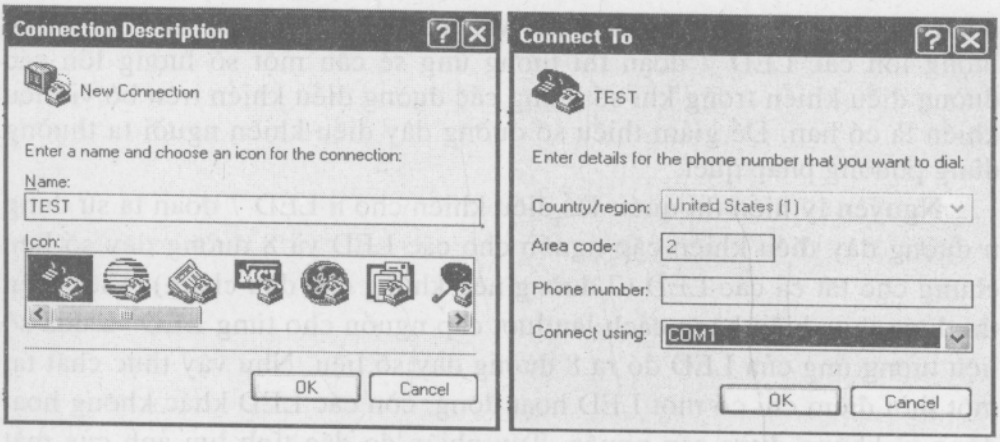
Để kiểm tra xem máy tính có nhận được các ký tự mà vi điều khiển truyền tới chưa, ta phải cho máy tính thi hành chương trình nhận số liệu qua cổng nối tiếp. Chương trình này có thể viết bằng ngôn ngữ lập trình Basis, Pascal, C, C++... Trong Windows có cung cấp sẵn cho chúng ta một công cụ truyền tin qua cổng nối tiếp là Hyper Terminal. Cách mở chương trình này là:

Từ menu Start → Programs → Acessories → Communications → chọn Hyper Terminal (hình 3.3).



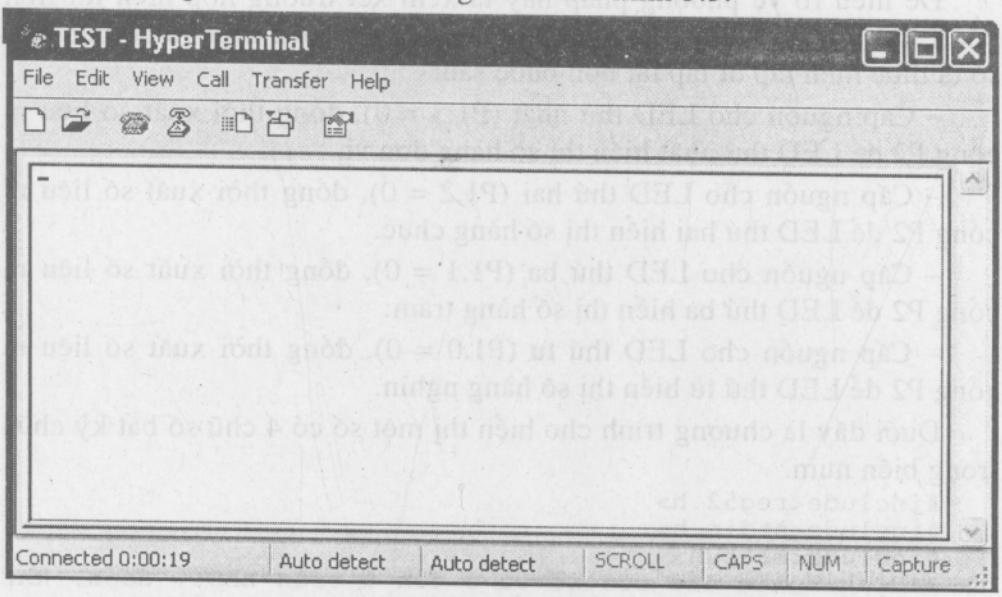
Hình 3.3. Cách mở Hyper Terminal trên Windows.

Sau đó nhập tên kết nối, chọn cổng nối tiếp và thiết lập các thông số cho cổng nối tiếp. Lưu ý rằng các thông số này phải giống như đã thiết lập cho cổng nối tiếp của vi điều khiển.



Hình 3.4. Chọn cổng nối tiếp và thiết lập các thông số.

Cuối cùng ta có cửa sổ chương trình như hình 3.5.



Hình 3.5. Chương trình Hyper Terminal.

Tại dấu nhắc con trỏ sẽ hiển thị các ký tự nhận được từ cổng nối tiếp. Với chương trình đã viết cho vi điều khiển như ở trên ta sẽ thu được chuỗi '0123456789' trên màn hình.

3.2.2. Hiển thị bằng LED 7 đoạn

Để hiển thị trên một đèn LED 7 đoạn thì cần tám đường điều khiển (bảy đường cho 7 đoạn và một đường cho dấu chấm) khi điều khiển trực tiếp hoặc cần 4 đường dây điều khiển khi dùng vi mạch giải mã BCD–7

đoạn 7447/7448. Với cả hai cách vừa nêu nếu cần điều khiển một số lượng lớn các LED 7 đoạn thì tương ứng sẽ cần một số lượng lớn các đường điều khiển trong khi số lượng các đường điều khiển trên bộ vi điều khiển là có hạn. Để giảm thiểu số đường dây điều khiển người ta thường dùng phương pháp quét.

Nguyên lý hiển thị quét: Để điều khiển cho n LED 7 đoạn ta sử dụng n đường dây điều khiển cấp nguồn cho các LED và 8 đường dây số liệu chung cho tất cả các LED (7 đường nếu không cần dấu chấm). Việc hiển thị được thực hiện bằng cách lần lượt cấp nguồn cho từng LED và đưa số liệu tương ứng của LED đó ra 8 đường dây số liệu. Như vậy thực chất tại một thời điểm chỉ có một LED hoạt động, còn các LED khác không hoạt động vì không được cấp nguồn. Tuy nhiên do đặc tính lưu ảnh của mắt người mà ta nhìn thấy tất cả các LED đều hoạt động.

Để hiểu rõ về phương pháp này ta xem xét trường hợp hiển thị trên bốn LED 7 đoạn như trong hình 3.6, để hiển thị được một số có bốn chữ số ta thực hiện lặp đi lặp lại bốn bước sau:

– Cấp nguồn cho LED thứ nhất (P1.3 = 0), đồng thời xuất số liệu ra cổng P2 để LED thứ nhất hiển thị số hàng đơn vị.

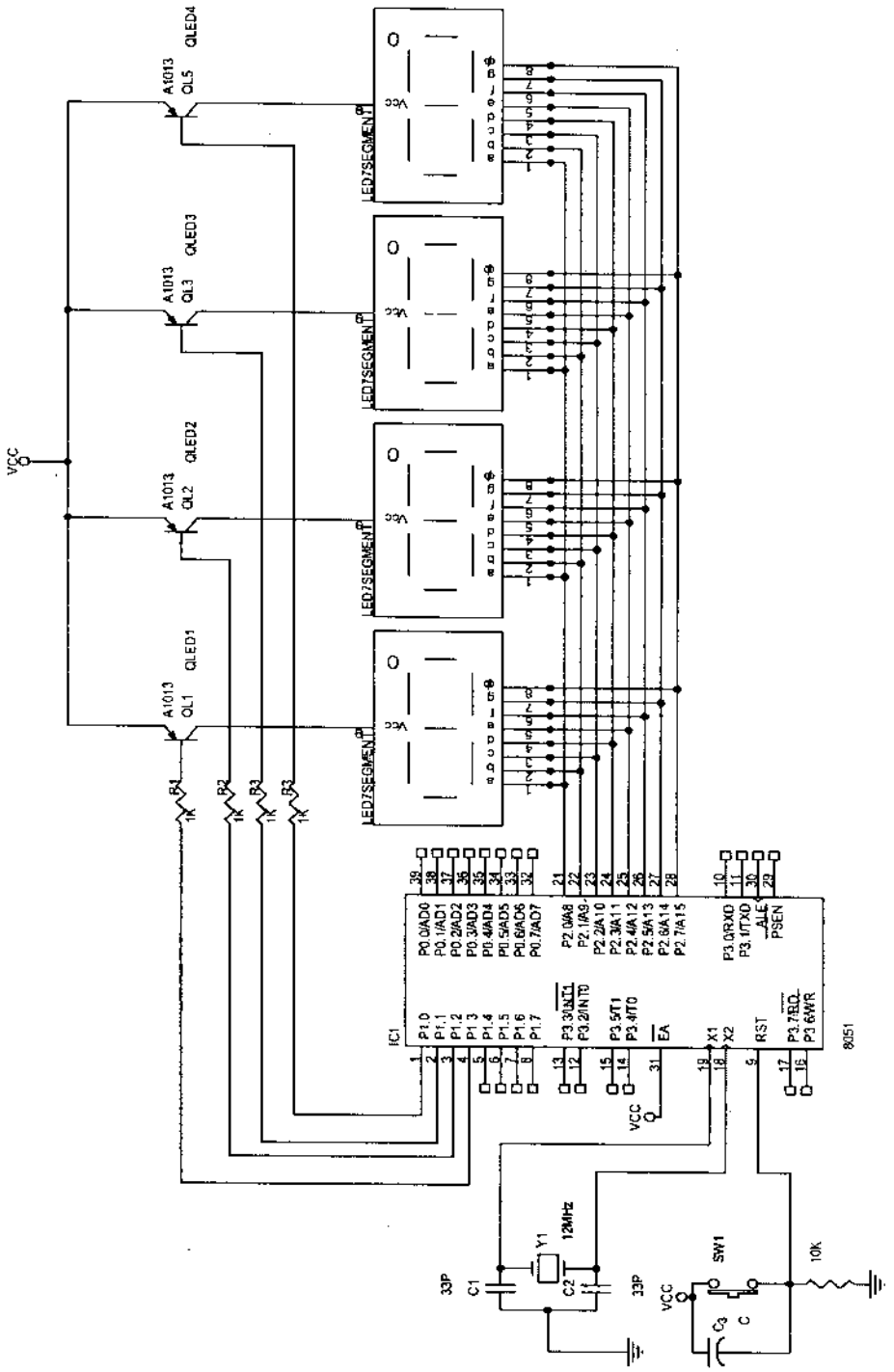
– Cấp nguồn cho LED thứ hai (P1.2 = 0), đồng thời xuất số liệu ra cổng P2 để LED thứ hai hiển thị số hàng chục.

– Cấp nguồn cho LED thứ ba (P1.1 = 0), đồng thời xuất số liệu ra cổng P2 để LED thứ ba hiển thị số hàng trăm.

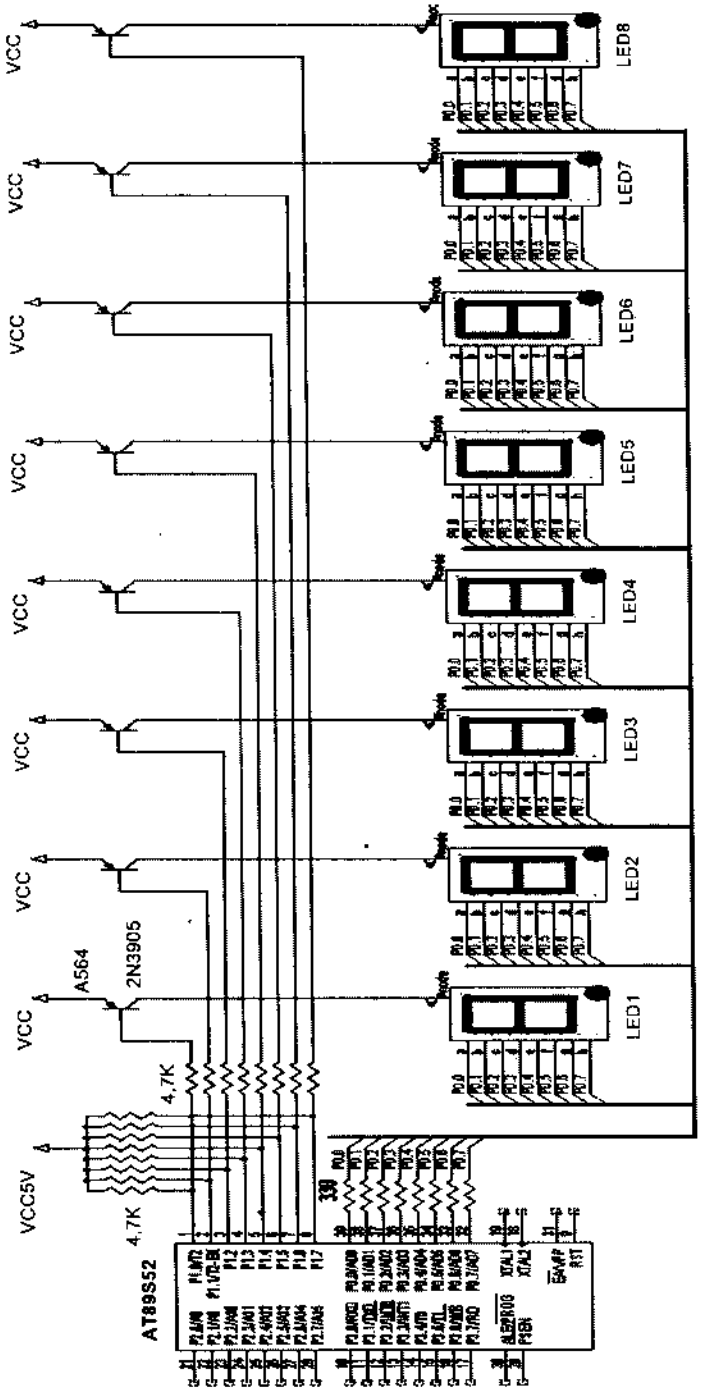
– Cấp nguồn cho LED thứ tư (P1.0 = 0), đồng thời xuất số liệu ra cổng P2 để LED thứ tư hiển thị số hàng nghìn.

Dưới đây là chương trình cho hiển thị một số có 4 chữ số bất kỳ chứa trong biến num.

```
#include<reg52.h>
#include<stdio.h>
#include<math.h>
/*Định nghĩa các con số từ 0 đến 9 trên LED 7 đoạn, khi
cần hiển thị số i ta xuất phần tử M[i] ra bus số liệu*/
unsigned char M[10]={0XC0,0XF9,0XA4,0XB0,0X99,
0X92,0X82,0XF8,0X80,0X90};
unsigned char donvi,chuc,tram,nghin,num;
void delay (void)
      {
      TMOD   = 0x02;
      TH0    = TL0 = -100;
      TR0    = 1;
      while(!TF0);
      TF0    = 0;
      TR0    = 0;
      }
```



Hình 3.6a. Mạch hiển thị 4 LED 7 đoạn.



Hình 3.6b. Mạch hiển thị 8 LED 7 đoạn.

```
void main (void)
{
                        //giả sử số cần hiển thị chứa trong biến num
nghin = num/1000;   //tính số hàng nghìn của num
tram =(num%1000)/100;//tính số hàng trăm của num
chuc =((num%1000)%100)/10;/*tính số hàng chục của num*/
donvi =((num%1000)%100)%10;/*tính số hàng đơn vị của num*/
while(1)
{
P1= 0XF7;           //P1.3 = 0,LED hàng đơn vị được cấp nguồn
P2 = M[donvi];
delay();
P1 = 0XFB;          //P1.2 = 0,LED hàng chục được cấp nguồn
P2 = M[chuc];
delay();
P1 = 0XFD;          //P1.1 = 0,LED hàng trăm được cấp nguồn
P2 = M[tram];
delay();
P1 = 0XFE;          //P1.0 = 0,LED hàng nghìn được cấp nguồn
P2 = M[nghin];
delay();
}
}
```

Giả sử cần hiển thị bằng n LED 7 đoạn ta cần dùng n đường dây điều khiển cấp nguồn cho n LED và 8 đường dây số liệu chung cho các LED, tổng số đường dây điều khiển sẽ là n+8. Trong khi đó nếu điều khiển trực tiếp n LED 7 đoạn ta phải dùng tới 8*n đường dây điều khiển.

Giải phương trình: 8*n = n + 8

ta có n = 8/7, có nghĩa là với n $\geq$ 2 đèn LED 7 đoạn thì phương pháp quét sẽ dùng ít đường dây điều khiển hơn phương pháp điều khiển trực tiếp, n càng lớn thì càng tiết kiệm được nhiều số đường dây điều khiển. Trên hình 3.6b là ví dụ điều khiển hiển thị 8 LED 7 đoạn cần dùng 16 đường dây số liệu.

3.2.3. Hiển thị bằng ma trận LED 8x8

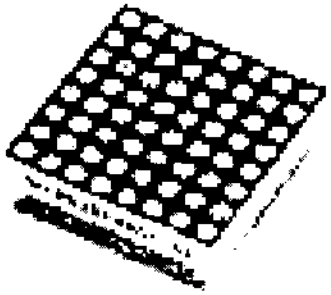
Ma trận LED 8x8 là linh kiện hiển thị được dùng rất rộng rãi trong thực tế. Có hai loại ma trận LED 8x8 là ma trận LED đơn sắc và ma trận LED đa sắc.

Một ma trận LED 8x8 đơn sắc bao gồm 64 LED được bố trí thành 8 hàng x 8 cột, trong đó các anốt của tám LED trong cùng một hàng được nối với nhau để tạo thành một đường dây hàng và các katốt của tám LED trong cùng một cột được nối với nhau để tạo thành một đường dây cột. Như vậy một ma trận LED 8x8 đơn sắc có tám đường dây hàng và tám đường dây cột, muốn một LED trong ma trận sáng ta cần cấp nguồn cho LED vào đường dây hàng và đường dây cột tương ứng với LED đó.

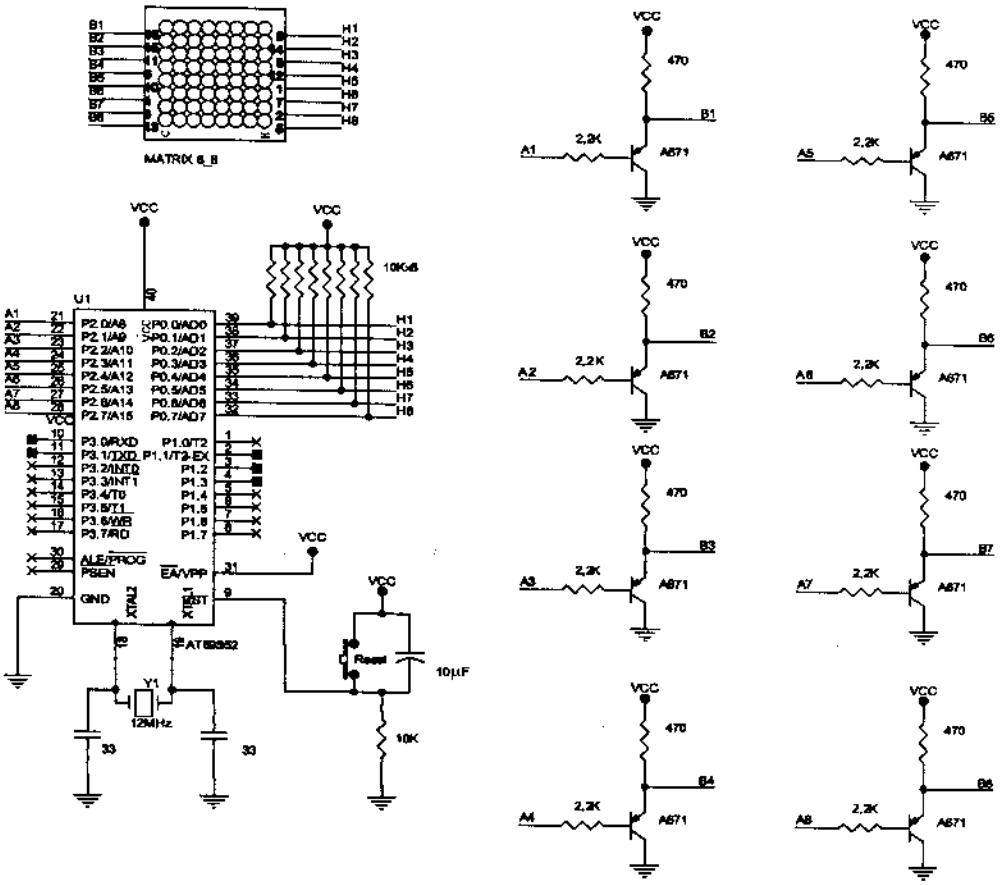
Một ma trận LED 8x8 đa sắc bao gồm 64 điểm sáng được bố trí thành 8 hàng x 8 cột, trong đó mỗi điểm sáng có thể gồm một LED màu xanh lục + một LED màu đỏ hoặc một LED màu xanh lục + một LED

màu xanh lơ + một LED màu đỏ hoặc một LED màu xanh lục + một LED màu xanh lơ + hai LED màu đỏ.

Với loại ma trận mà mỗi điểm sáng gồm một LED màu xanh lục + một LED màu đỏ thì điểm sáng hiển thị màu xanh nếu LED đỏ tắt, màu đỏ nếu LED xanh tắt, màu vàng nếu cả hai LED sáng và tắt nếu cả hai LED cùng tắt. Trong cùng một hàng, các katốt của các LED màu xanh được nối với nhau để tạo thành một đường dây hàng thứ nhất và các katốt của các LED màu đỏ được nối với nhau để tạo thành một đường dây hàng thứ hai. Các anốt của mười sáu LED trong cùng một cột được nối với nhau để tạo thành một đường dây cột. Như vậy muốn một LED trong ma trận sáng ta cần cấp nguồn cho LED vào đường dây hàng và đường dây cột tương ứng với LED đó.



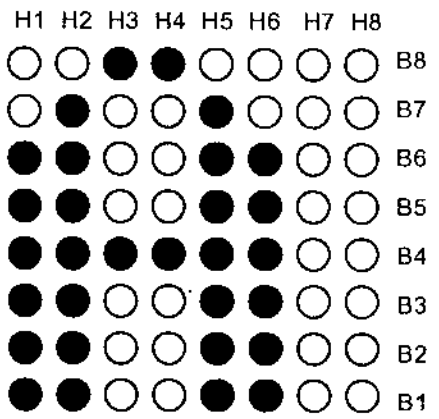
Hình 3.7. Ma trận LED 8x8.



Hình 3.8. Mạch điều khiển một ma trận LED 8x8 đơn sắc.

Ví dụ 1: Viết chương trình điều khiển để ma trận LED sáng chữ A như sau:

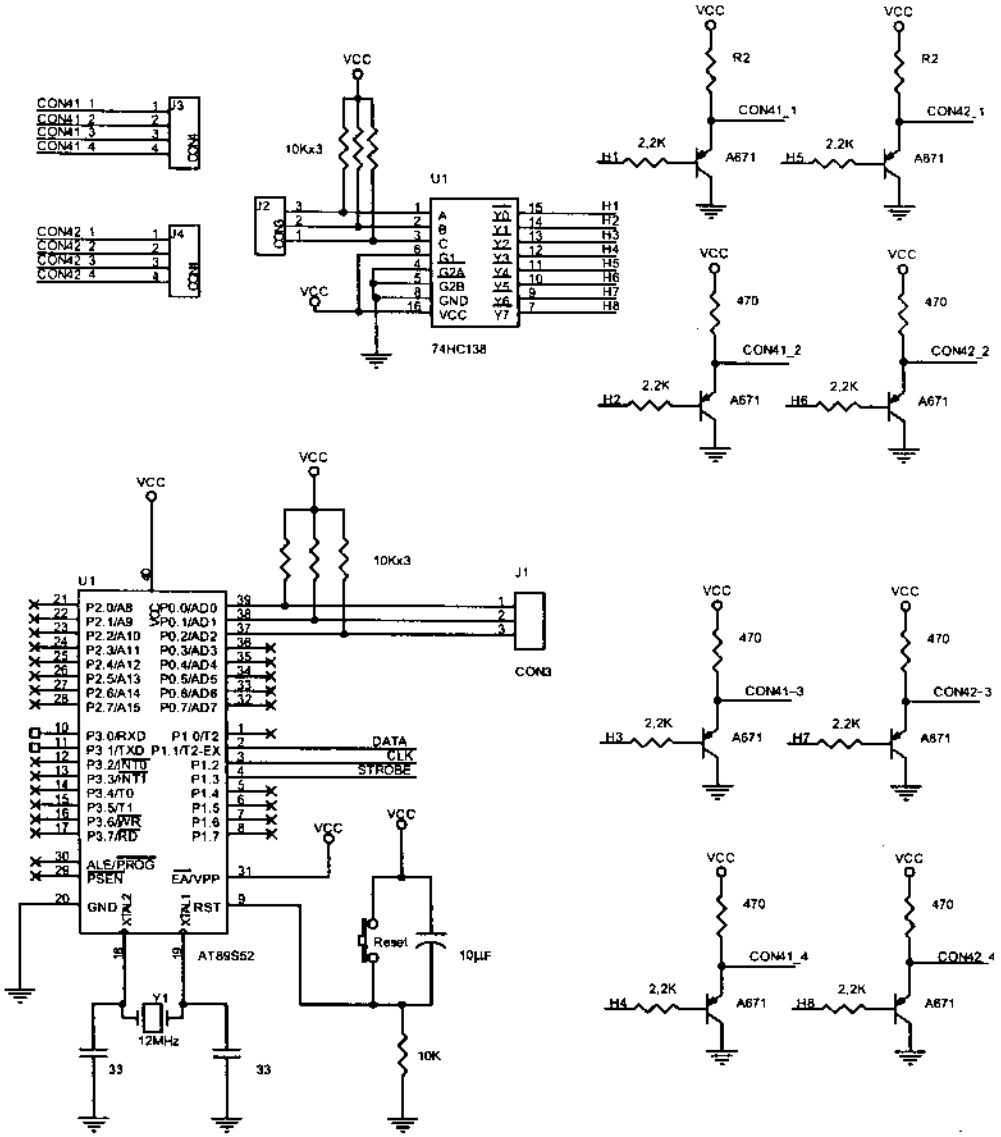
```
#include<reg51.h>
#include<stdio.h>
#include<instring.h>
void delay(unsigned int t)
{
unsigned int i;
for(i = 1;i< = t;++i);
}
int m,n,t,k,j;
unsigned char manghang [8] =
{0,  0x3F,  0x7F,  0x88,  0x88,
0x7F, 0x3F, 0};
unsigned char mangcot[8] =
{0xFE, 0xFD, 0xFB, 0xF7, 0XEF,
0xDF, 0XBF, 0X7F};
void main()
{
while (1)
{
for(n = 0;n< = 7;++n)
{
P2 = manghang[n];
P0 = mangcot[n];
delay(20);
}
}
}
```



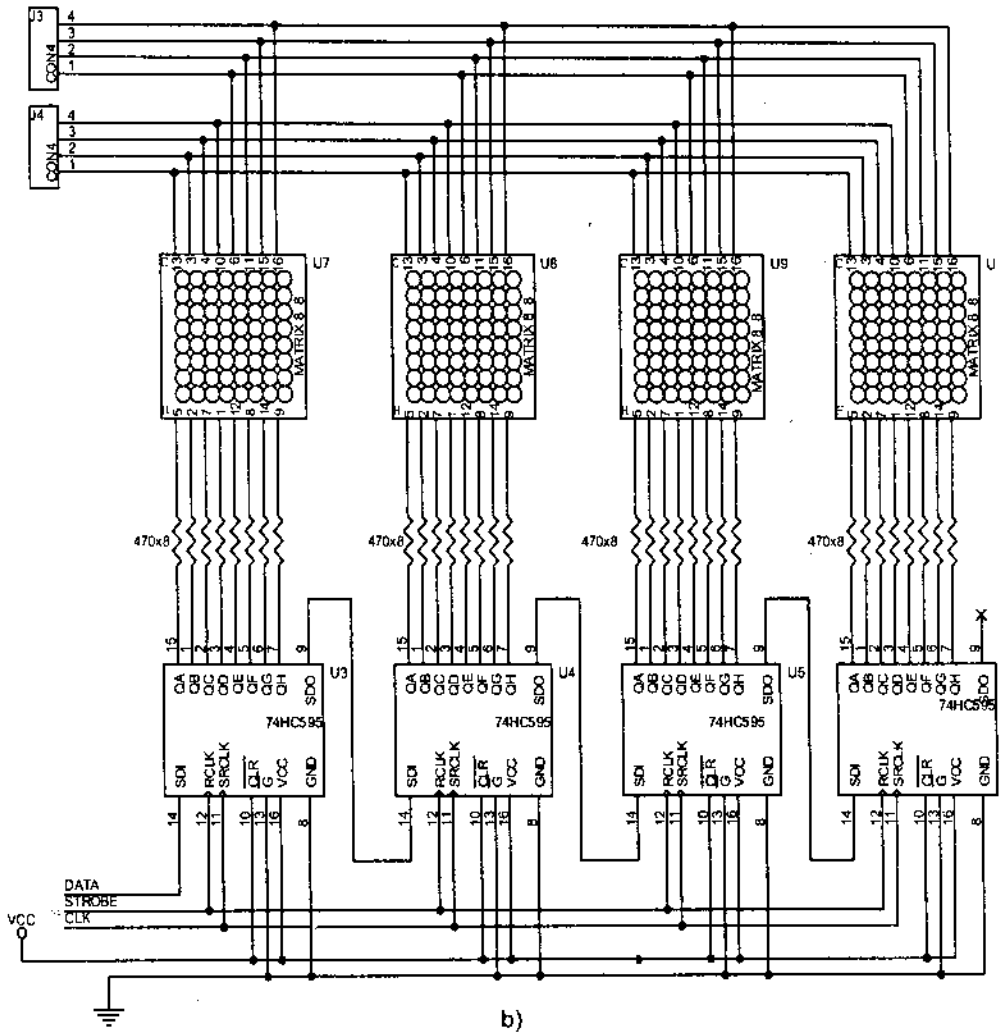
Ví dụ 2: Viết chương trình điều khiển để ma trận LED sáng chữ A chạy từ trái sang phải.

```
#include<REG51.H>
#include<STDIO.H>
#include<INTRINS.H>
void delay(unsigned int t)
{
unsigned int i;
for(i = 1;i< = t;++i);
}
int m,n,t,k,j;
unsigned char manghang[16]=
{0,0,0,0,0,0,0,0,0x3F,0x7F,0x88,0x88,0x7F,0x3F,0,0};
unsigned char mangcot[10]= {0xFE,0xFD,0xFB,0xF7,0XEF,0xDF,0XBF,0X7F};
void main()
{
while (1)
{
k = 8;
for (t = 0;t< = 7;++t)
{
for (j = 1;j< = 200;++j)// quét chữ A 200 lần
for(n = 0;n< = 7;++n)
{
P2 = manghang[n+k];
P0 = mangcot[n];
delay(20);
}
P2 = 0 ;
delay(100);
k--;
}
}
}
```

Trong thực tế thường yêu cầu hiển thị thông tin trên bảng gồm nhiều ma trận LED ghép lại. Nếu ta dùng vi điều khiển ghép trực tiếp với các ma trận LED thì đòi hỏi rất nhiều chân vào/ra vì theo như hình 3.8 thì để điều khiển một ma trận phải cần tới 16 chân vào/ra của vi điều khiển. Để khắc phục vấn đề này ta có thể dùng các thanh ghi dịch biến đổi dữ liệu nối tiếp – song song kết hợp với vi mạch giải mã 3:8 để chọn cột (hoặc chọn hàng). Hình 3.9 là ví dụ về một mạch hiển thị gồm bốn ma trận LED 8x8 đơn sắc.



a)



Hình 3.9. Mạch hiển thị sử dụng bốn ma trận LED 8x8 đơn sắc.

Ví dụ 3:

```
#include<REG51.H>
#include<STDIO.H>
#include<INTRINS.H>
sbit DATA=P1^1;      //dữ liệu nối tiếp
sbit SCK=P1^2;       //clock dịch bit
sbit STROBE=P1^3;    //chốt
                     //mảng cột
unsigned char code array_colum[8]= {0x00, 0x01, 0x02,
0x03, 0x04, 0x05, 0x06, 0x07};
                     //mảng hàng
unsigned char code ARRAYLINE[105]={
```

```
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x7F,0xFF,0x88,0x88,0xFF,0x7F,0x00,  // A
    0xFF,0xFF,0x91,0x91,0xFF,0x6E,0x00,  // B
    0x7E,0xFF,0x81,0x81,0xC3,0x42,0x00,  // C
    0xFF,0xFF,0x81,0x81,0x7E,0x3C,0x00,  // D
    0xFF,0xFF,0x99,0x99,0x99,0x99,0x00,  // E
    0xFF,0xFF,0x90,0x90,0x90,0x90,0x00,  // F
    0x7E,0xFF,0x81,0x89,0xEF,0x6F,0x00,  // G
    0xFF,0xFF,0x10,0x10,0xFF,0xFF,0x00,  // H
    0x81,0x81,0xFF,0xFF,0x81,0x81,0x00,  // I
    0x83,0x81,0xFF,0xFE,0x80,0x80,0x00,  // J
    0xFF,0xFF,0x38,0x6C,0xC6,0x83,0x00,  // K
    0xFF,0xFF,0x01,0x01,0x01,0x00,0x00,  // L
    0xFF,0x7F,0x20,0x20,0x7F,0xFF,0x00,  // M
    0xFF,0x7F,0x10,0x08,0xFE,0xFF,0x00,  // N
  //0x7E,0xFF,0x81,0x81,0xFF,0x7E,0x00   // O
    };
    unsigned char code arrayA[7]= {0x7F, 0xFF, 0x88, 0x88,
0xFF, 0x7F, 0x00}; // A
    unsigned char code arrayB[7]= {0xFF, 0xFF, 0x91, 0x91,
0xFF, 0x6E, 0x00}; // B
    unsigned char code arrayC[7]= {0x7E, 0xFF, 0x81, 0x81,
0xC3, 0x42, 0x00}; // C
    unsigned char code arrayD[7]= {0xFF, 0xFF, 0x81, 0x81,
0x7E, 0x3C, 0x00}; // D
    unsigned char code arrayE[7] = {0xFF, 0xFF, 0x99,
0x99, 0x99, 0x99, 0x00}; // E
    unsigned char code arrayF [7]= {0xFF, 0xFF, 0x90,
0x90, 0x90, 0x90, 0x00}; // F
    unsigned char code arrayG[7]= {0x7E, 0xFF, 0x81, 0x89,
0xEF, 0x6F,0x00}; // G
    unsigned char code arrayH[7]= {0xFF, 0xFF, 0x10, 0x10,
0xFF, 0xFF, 0x00}; // H
    unsigned char code arrayI[7]= {0x81, 0x81, 0xFF, 0xFF,
0x81, 0x81, 0x00}; // I
    unsigned char code arrayJ[7]= {0x83, 0x81, 0xFF, 0xFE,
0x80, 0x80, 0x00}; // J
    unsigned char code arrayK[7]= {0xFF, 0xFF, 0x38, 0x6C,
0xC6, 0x83, 0x00}; // K
    unsigned char code arrayL[7]= {0xFF, 0xFF, 0x01, 0x01,
0x01,0x00,0x00}; // L
    unsigned char code arrayM[7]= {0xFF, 0x7F, 0x20, 0x20,
0x7F, 0xFF,0x00}; // M
    unsigned char code arrayN[7]= {0xFF, 0x7F, 0x10, 0x08,
0xFE,0xFF,0x00}; // N
    unsigned char code arrayO[7]= {0x7E, 0xFF, 0x81, 0x81,
0xFF, 0x7E, 0x00}; // O
    unsigned char code arrayP[7]= {0xFF, 0xFF, 0x88, 0x88,
0xF8, 0x70, 0x00}; // P
    unsigned char code arrayQ[7]= {0x7E, 0xFF, 0x85, 0x83,
0xFF, 0x7F, 0x01}; // Q
```

```
    unsigned char code arrayR[7]= {0xFF, 0xFF, 0x8C, 0x8E,
0xFB, 0x71, 0x00}; // R
    unsigned char code arrayS[7]= {0x62, 0xF1, 0xB9, 0x9D,
0x8F, 0x46, 0x00}; // S
    unsigned char code arrayT[7]= {0x80, 0x80, 0xFF, 0xFF,
0x80, 0x80, 0x00}; // T
    unsigned char code arrayU[7]= {0xFE, 0xFF, 0x01, 0x01,
0xFF, 0xFE, 0x00}; // U
    unsigned char code arrayV[7]= {0xFC, 0xFE, 0x03, 0x03,
0xFE, 0xFC, 0x00}; // V
    unsigned char code arrayW[7]= {0xFF, 0xFE, 0x04, 0x04,
0xFE, 0xFF, 0x00}; // W
    unsigned char code arrayX[7]= {0xC3, 0x66, 0x18, 0x18,
0x66, 0xC3, 0x00}; // X
    unsigned char code arrayY[7]= {0xE0, 0x30, 0x1F, 0x1F,
0x30, 0xE0, 0x00}; // Y
    unsigned char code arrayZ[7]= {0xC3, 0xCF, 0x99, 0xB1,
0xE3, 0xC3, 0x00}; // Z
    unsigned char code arraySP[7]= {0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00}; // space
  //Hàm chuyển dữ liệu song song sang nối tiếp
  void put_rows(unsigned char DATA_OUT)
  {
  unsigned char i,tempH;
  for(i = 0;i<8;i++)
  {
  tempH = DATA_OUT;
  tempH = tempH&0x80;
  if(tempH == 0x80)
  DATA = 1;
  else
  DATA = 0;
  DATA_OUT * = 2;
  SCK = 1;
  nop_();
  _nop_();
  SCK = 0;
  }
  }
  void delay(unsigned int t)
  {
  unsigned int i;
  for(i = 1;i <= t;++i);
  }
  //xóa ma trận sau mỗi lần quét
  void clear()
  {
  put_rows(0X00);
     put_rows(0X00);
   put_rows(0X00);
```

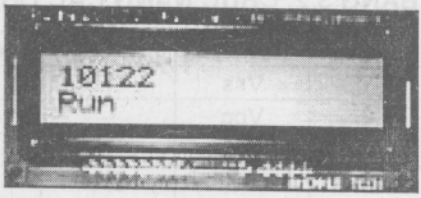
```
    put_rows(0X00);
}
unsigned int n,m,z;
void main(void)
{
while(1)
{
z = 0;
while(105-z)
   {
   for(m = 1;m< = 7;m++)
       {
      for(n = z;n< = 7+z;++n)
          {
          if (n>=105)                    //ma trận 4
              put_rows(0X00);
          else
              put_rows(ARRAYLINE[n]);
          if ((n> = 202)||(n<8))         //ma trận 3
              put_rows(0X00);
          else
              put_rows(ARRAYLINE[n-8]);
          if((n> = 209)||(n<16))         //ma trận 2
              put_rows(0X00);
          else
              put_rows(ARRAYLINE[n-16]);
          if((n>= 216)||(n<24))          //ma trận 1
              put_rows(0X00);
          else
              put_rows(ARRAYLINE[n-24]);
          STROBE=1;
          _nop_();
          _nop_();
          STROBE = 0;
          P0 = array_colum[n-z];         //quét cột
          delay(300);
          clear();                       //xóa
         }
       }
   ++z;
   }
  }
}
```

3.2.4. Ghép nối với LCD hiển thị ký tự

1. Phân loại LCD

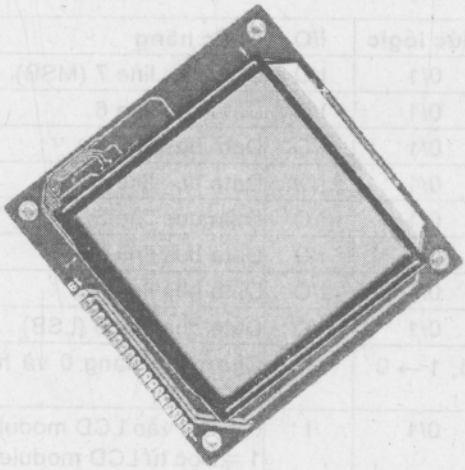
Có thể chia các module LCD làm hai loại chính là:

– Loại hiển thị ký tự (character LCD) gồm có các kích cỡ 16x1 (16 ký tự x 1 dòng); 16x2 (16 ký tự x 2 dòng); 16x4 (16 ký tự x 4 dòng); 20x1 (20 ký tự x 1 dòng); 20x2 (20 ký tự x 2 dòng); 20x4 (16 ký tự x 4 dòng); 40x1 (40 ký tự x 1 dòng); 40x2 (40 ký tự x 2 dòng) và 40x4 (40 ký tự x 4 dòng). Mỗi ký tự được tạo bởi một ma trận các điểm sáng kích thước 5x7 hoặc 5x10 điểm ảnh.



Hình 3.10. Module LCD hiển thị ký tự 16x2.

– Loại hiển thị đồ hoạ (graphic LCD) đen trắng hoặc màu, gồm có các kích cỡ 1,47 inch (128x128 điểm ảnh); 1,8 inch (128x160 điểm ảnh); 2 inch (176x220 điểm ảnh); 2,2 inch (240x320 điểm ảnh); 2,4 inch (240x320 điểm ảnh); 3,5 inch (320x240 điểm ảnh); 4,3 inch (480x272 điểm ảnh); 7 inch (800x480 điểm ảnh); 8 inch (800x600 điểm ảnh)... được dùng trong điện thoại di động, máy ảnh số, camera, máy hiện sóng số...



Hình 3.11. Module LCD hiển thị graphic đơn sắc 160x160.

2. Ý nghĩa các chân và mã lệnh điều khiển LCD hiển thị ký tự

Hầu hết các module LCD hiển thị ký tự được thiết kế dựa trên bộ điều khiển HD44780 của Hitachi nên chúng có tập lệnh và các chân tương thích nhau. Bảng 3.2 cho ta ý nghĩa các chân của các module LCD có tối đa 80 ký tự còn bảng 3.3 cho ta biết ý nghĩa các chân của các module LCD lớn hơn 80 ký tự.

BẢNG 3.2. CHỨC NĂNG CÁC CHÂN CỦA MODULE LCD CÓ TỐI ĐA 80 KÝ TỰ

Chân số	Ký hiệu	Mức logic	I/O	Chức năng
1	Vss	–	–	Nguồn cung cấp (GND)
2	Vcc	–	–	Nguồn cung cấp (+5V)
3	Vee	–	I	Điện áp vào để điều chỉnh độ tương phản
4	RS	0/1	I	Lựa chọn thanh ghi 0 = Thanh ghi lệnh 1 = Thanh ghi dữ liệu
5	R/W	0/1	I	0 = Ghi vào LCD module 1 = Đọc từ LCD module
6	E	1, 1 → 0	I	Tín hiệu cho phép
7	DB0	0/1	I/O	Data bus line 0 (LSB)
8	DB1	0/1	I/O	Data bus line 1
9	DB2	0/1	I/O	Data bus line 2
10	DB3	0/1	I/O	Data bus line 3
11	DB4	0/1	I/O	Data bus line 4
12	DB5	0/1	I/O	Data bus line 5
13	DB6	0/1	I/O	Data bus line 6
14	DB7	0/1	I/O	Data bus line 7 (MSB)

BẢNG 3.3. CHỨC NĂNG CÁC CHÂN CỦA MODULE LCD HƠN 80 KÝ TỰ

Chân số	Ký hiệu	Mức logic	I/O	Chức năng
1	DB7	0/1	I/O	Data bus line 7 (MSB)
2	DB6	0/1	I/O	Data bus line 6
3	DB5	0/1	I/O	Data bus line 5
4	DB4	0/1	I/O	Data bus line 4
5	DB3	0/1	I/O	Data bus line 3
6	DB2	0/1	I/O	Data bus line 2
7	DB1	0/1	I/O	Data bus line 1
8	DB0	0/1	I/O	Data bus line 0 (LSB)
9	E1	1, 1 → 0	I	Cho phép hàng 0 và hàng 1 (bộ điều khiển thứ nhất)
10	R/W	0/1	I	0 = Ghi vào LCD module 1 = Đọc từ LCD module
11	RS	0/1	I	Lựa chọn thanh ghi 0 = Thanh ghi lệnh 1 = Thanh ghi dữ liệu
12	Vee	–	I	Điện áp vào để điều chỉnh độ tương phản
13	Vss	–	–	Power supply (GND)
14	Vcc	–	–	Power supply (+5V)
15	E2	1, 1 → 0	I	Cho phép hàng 2 và hàng 3 (bộ điều khiển thứ hai)
16	NC			Không dùng

BẢNG 3.4. MÃ LỆNH CỦA HD44780

Lệnh	Mã lệnh										Mô tả	Thời gian thi hành
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Xóa màn hình	0	0	0	0	0	0	0	0	0	1	Xóa màn hình, đưa con trỏ về vị trí đầu (address 0).	1,64 ms
Đưa con trỏ về vị trí đầu	0	0	0	0	0	0	0	0	1	x	Đưa con trỏ về vị trí đầu (address 0).	1,64 ms
Thiết lập chế độ	0	0	0	0	0	0	0	1	I/D	S	Thiết lập hướng dịch con trỏ (I/D), dịch hiển thị (S).	40µs
Bật/tắt hiển thị	0	0	0	0	0	0	1	D	C	B	Bật/Tắt hiển thị, con trỏ; bật/tắt chế độ nhấp nháy của con trỏ.	40µs
Dịch con trỏ/hiển thị	0	0	0	0	0	1	S/C	R/L	*	*	Thiết lập chiều dịch chuyển của con trỏ và hiển thị.	40µs
Thiết lập chức năng	0	0	0	0	1	DL	N	F	*	*	Thiết lập độ dài của dữ liệu, số dòng và font chữ.	40µs
Thiết lập địa chỉ CGRAM	0	0	0	1	CGRAM address						Thiết lập địa chỉ CGRAM.	40µs
Thiết lập địa chỉ DDRAM	0	0	1	DDRAM address							Thiết lập địa chỉ DDRAM	40µs
Đọc cờ báo bận và địa chỉ CGRAM /DDRAM	0	1	BF	CGRAM/DDRAM address							Đọc cờ báo bận và địa chỉ của CGRAM hoặc DDRAM (tuỳ vào lệnh trước đó)	0µs
Ghi CGRAM/ DDRAM	1	0	write data								Ghi dữ liệu vào CGRAM hoặc DDRAM.	40µs
Đọc CGRAM/ DDRAM	1	1	read data								Đọc dữ liệu từ CGRAM hoặc DDRAM.	40µs

Chú ý:

– DDRAM = Display Data RAM – chứa dữ liệu cần hiển thị (mã ASCII của các ký tự); địa chỉ của DDRAM tương ứng với vị trí con trỏ trên màn hình.

– CGRAM = Character Generator RAM – RAM bộ tạo ký tự

– Các bit viết tắt trong mã lệnh:

Tên bit	Mô tả	
I/D	0 = Decrement cursor position	1 = Increment cursor position
S	0 = Không dịch chuyển hiển thị	1 = Dịch chuyển hiển thị
D	0 = Tắt hiển thị	1 = Bật hiển thị
C	0 = Tắt con trỏ	1 = Bật con trỏ
B	0 = Con trỏ không nhấp nháy	1 = Con trỏ nhấp nháy
S/C	0 = Di chuyển con trỏ	1 = Dịch chuyển hiển thị
R/L	0 = Dịch trái	1 = Dịch phải
DL	0 = Chế độ 4–bit dữ liệu	1 = Chế độ 4–bit dữ liệu
N	1 dòng	2 dòng
F	0 = Font chữ 5x7	1 = Font chữ 5x10
BF	0 = Không bận	1 = Đang bận

3. Nguyên tắc hiển thị ký tự trên LCD

Một chương trình hiển thị ký tự trên LCD sẽ đi theo bốn bước sau:

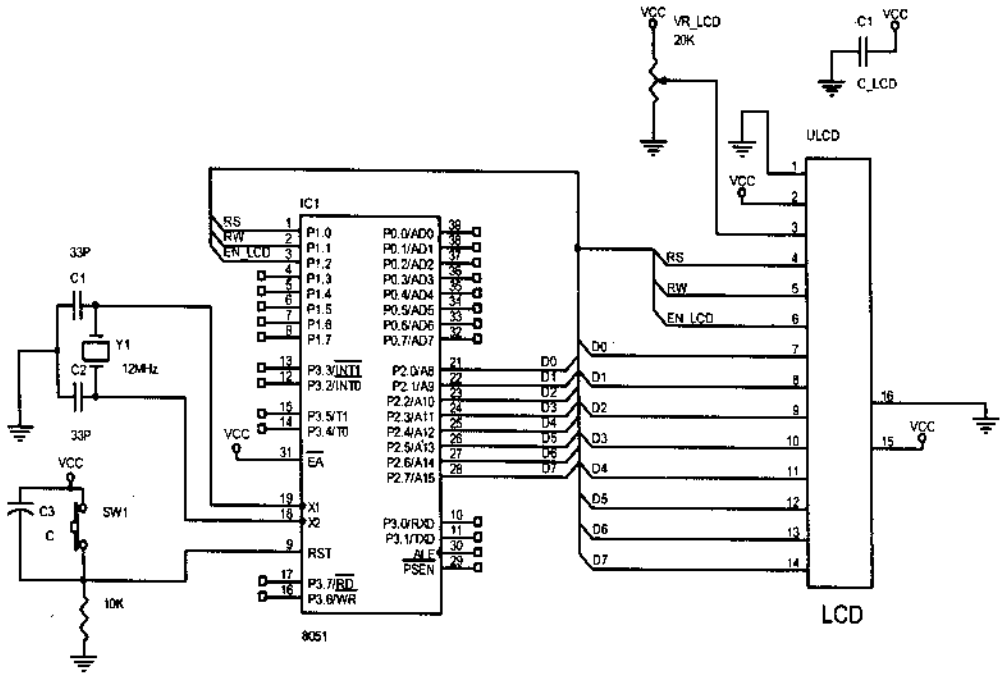
1. Xoá toàn bộ màn hình.
2. Đặt chế độ hiển thị.
3. Đặt vị trí con trỏ (nơi bắt đầu của ký tự hiển thị).
4. Hiển thị ký tự.

Chú ý:

+ Các bước 3, 4 có thể lặp lại nhiều lần nếu cần hiển thị nhiều ký tự.

+ Mỗi khi thực hiện ghi lệnh hoặc ghi dữ liệu hiển thị lên LCD đều phải kiểm tra cờ bận (xem hàm `busy_flag` trong chương trình). Tuy nhiên có một số loại LCD không cho phép kiểm tra cờ bận, vì vậy bộ vi điều khiển cần phải chủ động phân phối thời gian khi ra lệnh cho LCD (ví dụ sau khi xoá màn hình thì sau khoảng 2ms mới ra lệnh khác vì thời gian để LCD xoá màn hình là 1,64ms).

+ Chế độ hiển thị mặc định sẽ là hiển thị dịch, vị trí con trỏ mặc định sẽ là đầu dòng thứ nhất.



Hình 3.12. Mạch giao tiếp với màn hình tinh thể lỏng LCD 16x2.

Để điều khiển hoạt động của LCD nên sử dụng Port 2 hoặc Port 1 cho việc xuất nhập dữ liệu, các chân tạo tín hiệu điều khiển RS, RW, EN_LCD có thể chọn tùy ý trong các chân của các Port còn lại. Hình 3.12 là ví dụ về mạch ghép nối giữa vi điều khiển 8951 với module LCD 16x2. Port 2 của vi điều khiển được nối tới bus dữ liệu của LCD, các chân P1.0, P1.1 và P1.2 của Port 1 dùng để tạo các tín hiệu điều khiển LCD. Chương trình dưới đây sẽ hiển thị các ký tự trên LCD.

```
#include<reg52.h>
#include<stdio.h>
sbit RS = P1^0;
sbit RW = P1^1;
sbit EN = P1^2;
char x;
void delay30ms(void)
      {
      TMOD = 0X10;
      TH1 = 35535/256;
      TL1 = 35535%256;
      TR1 = 1;
      while(!TF1);
      TR1 = TF1 = 0;
      }
```

```
void delay(unsigned long int t)
      {
      unsigned long int i;
      for(i= 0; i< t; ++i);
      }
                              //Kiểm tra cờ bận của LCD
void busy_flag (void)
      {
      P2 = 0xff;
      RS = 0;
      RW = 1;
      do
           {
           EN = 1;
           delay(10);
           EN = 0;
           x = P2;
           x = x&0x80;
           }
      while (x!= 0x80);
      }
                              //ghi lệnh ra LCD
void write_command (unsigned char LCD_command)
      {
      busy_flag();
      P2 = LCD_command;
      RS = 0;                 //chọn thanh ghi lệnh
      RW = 0;
      EN = 1;
      delay(50);
      EN = 0;
      delay(50);
      }
                              //ghi dữ liệu cần hiển thị ra LCD
void write_data (unsigned char LCD_data)
      {
      busy_flag();
      P2 = LCD_data; /*Ghi mot ky tu ra LCD*/
      RS = 1;                 //chọn thanh ghi dữ liệu
      RW = 0;
EN = 1;
      delay(50);
      EN = 0;
```

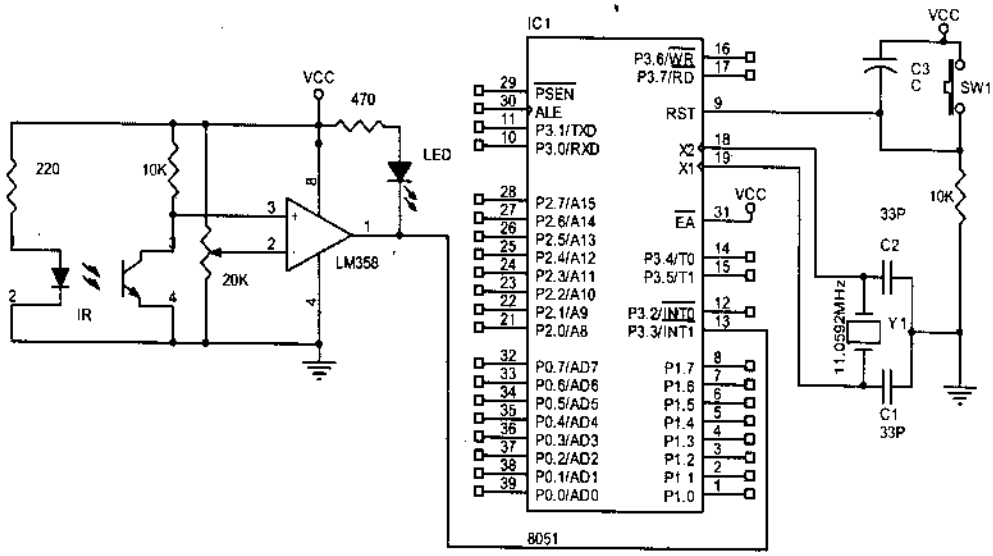
```
        delay(50);
        }
void write_string(char *s)
        {
        while(*s)
             {
             write_data(*s); /*Ghi mot chuoi ky tu ra LCD*/
             s++;
             }
        }
void init(void)          //khởi tạo LCD
        {
        write_command(0x03);
        write_command(0x38);
        write_command(0x06);
        write_command(0x0e);
        }
void main(void)
        {
        delay30ms();
        init();
        while(1)
             {
             write_command(0x01);
             write_command(0x80);
             write_string(" DH CONG NGHIEP ");
             write_command(0xc0);
             write_string(" KHOA DIEN TU ");
             }
        }
```

3.2.5. Bộ đếm sản phẩm sử dụng ngắt ngoài

Cảm biến hồng ngoại (IR Sensor) được dùng khá phổ biến trong các ứng dụng thực tế. Tuỳ từng ứng dụng khác nhau người ta có thể dùng các kỹ thuật khuếch đại, điều chế điện, điều chế quang... để nâng cao độ nhạy, độ tin cậy... của cảm biến. Trong khuôn khổ chương trình này chúng tôi chỉ đưa ra một mạch điện đơn giản, rẻ tiền để có thể thí nghiệm làm quen được với loại cảm biến thông dụng này.

Trên hình 3.13 bộ khuếch đại thuật toán làm việc như một bộ so sánh, biến trở 20k có chức năng điều khiển điện áp ngưỡng so sánh. Mỗi khi có một vật đi qua tức là có sự che/không che khuất tia hồng ngoại từ

diode phát sáng photo–transistor dẫn đến có sự chuyển mức từ 1 sang 0 ở đầu ra của bộ khuếch đại thuật toán gây ra ngắt ngoài INT1.



Hình 3.13. Đếm sản phẩm dùng cảm biến hồng ngoại.

Chương trình dưới đây sẽ minh hoạ quá trình đếm mỗi khi có vật đi qua khe hẹp giữa diode phát sáng và photo–transistor và hiển thị giá trị đếm trên 4 đèn LED 7 đoạn (tất nhiên cần bổ sung phần LED 7 đoạn vào hình 3.13 như đã làm ở mục 3.2.2).

```
#include<reg52.h>
#include<stdio.h>
#include<math.h>
unsigned char M[10]={0XC0,0XF9,0XA4,0XB0,0X99,
                     0X92,0X82,0XF8,0X80,0X90};
unsigned char donvi,chuc,tram,nghin;
long int num,j;
void tre (unsigned long int t)
      {
      unsigned long int i;
      for (i=1;i<=t;++i);
      }
void hienthi(void)
      {
      nghin = num/1000;
      tram = (num%1000)/100;
      chuc = ((num%1000)%100)/10;
      donvi= ((num%1000)%100)%10;
      for (j = 1;j<= 100;++j)
            {
            P1 = 0XF7;
            P2 = M[donvi];
```

```
            tre(100);
            P1 = 0XFB;
            P2 = M[chuc];
            tre(100);
            P1 = 0XFD;
            P2 = M[tram];
            tre(100);
            P1 = 0XFE; ;
            P2 = M[nghin];
            tre(100);
            P1 = 0XFF;
            }
      }
void main(void)
      {
      IE = 0x84 ; //cho phép ngắt ngoài INT1
      IT1 = 1;    //ngắt bằng sườn âm
      num = 0;
      while(1)
      {
      hienthi();
      }
      }
void ngatngoai_1(void) interrupt 2
      {
      ++num ;/*số sản phẩm đếm được chứa trong biến num*/
      }
```

3.2.6. Tạo xung vuông, hẹn giờ và đo tần số sử dụng timer

Ví dụ 1: Tạo xung có tần số 50kHz trên chân P1.0 của 8951, biết thạch anh được sử dụng có tần số dao động là 24MHz.

Trong ví dụ này tần số cần tạo là 50kHz trong khi xung từ bộ chia tần của AT89S51 cấp cho các bộ Timer là 2MHz, như vậy chúng ta cần có hệ số chia là 40. Với hệ số chia này thì chỉ cần các bộ Timer 8 bit là có thể đáp ứng được. Dưới đây là chương trình cụ thể sử dụng Timer 0 ở mode 2:

```
#include<reg51.h>
sbit F = P1^0;
void main(void)
{
      TMOD = 0X02;  //Timer 0 mode 2
      TH0 = -20;    //hệ số chia là 40
      TR0 = 1;
      while(1)
      {
      while (!TF0); //chờ cờ tràn
      TF0 = 0;      //xoá cờ tràn
      F = ~F;       //đảo mức
      }
}
```

Ví dụ 2: Tạo hai xung đồng thời có tần số 10kHz và 50Hz trên chân P1.0 và P1.1, biết thạch anh được sử dụng có tần số 12MHz.

Xung từ bộ chia tần của AT89S52 cấp cho các bộ timer là 1MHz, để tạo hai xung đồng thời ta cần sử dụng hai bộ Timer:

– Timer 1 tạo xung có tần số 10kHz, cần có hệ số chia là 100. Với hệ số chia này thì bộ Timer 1 làm việc ở chế độ 2 tự nạp lại.

– Timer 0 tạo xung có tần số 50Hz, cần có hệ số chia là 20000, với hệ số chia lớn hơn 256 thì chỉ cần bộ Timer 0 làm việc ở chế độ 1.

Dưới đây là chương trình cụ thể:

```
#include<stdio.h>
#include<reg52.h>
sbit xung1 = P1^0;
sbit xung2 = P1^1;
void main (void)
      {
            TMOD = 0X21;     //Timer 0 mode 1, Timer 1 mode 2
            TH1 = TL1= -50;  //hệ số chia là 100, tức f = 10kHz
            TR1 = 1;         //cho Timer 1 chạy
            IE = 0X8A;       //cho phép ngắt do Timer 1 và Timer 0
            IP = 0;          //mức ưu tiên bằng nhau
            TF0 = 1;         //buộc ngắt do Timer 0;
            while(1);        //chờ ngắt
      }
void ngatT0 (void) interrupt 1
      {
            TR0 = 0;dừng timer 0 để nạp giá trị đầu
            TH0 = -10000/256;
            TL0 = -10000%256;//hệ số chia là 20000,tức f = 50Hz
            xung2 = ~xung2;//lật mức logic chân P1.1
            TR0 = 1;         //cho timer 0 chạy
      }
void ngatT1 (void) interrupt 3
      {
      xung1 = ~xung1;    //lật mức logic chân P1.0
      }
```

Trong ví dụ trên để tạo tần số 50Hz, giá trị nạp cho timer 0 là –10000 (TH0 chứa byte cao của –10000, TL0 chứa byte thấp của –10000), trong khi thanh ghi này có thể chứa giá trị tối đa là –65535 tương ứng với hệ số chia là 131070. Như vậy xung có tần số nhỏ nhất mà một bộ Timer 16 bit có thể tạo được từ thạch anh 12MHz là 1/131070MHz, tức xấp xỉ 8Hz, muốn tạo ra xung có tần số nhỏ hơn ta phải sử dụng kết hợp nhiều Timer.

Lưu ý rằng trong chế độ 1 các Timer 0, 1 không tự nạp lại được, do đó trong chương trình phải có câu lệnh nạp lại giá trị xuất phát sau khi cho timer dừng lại bởi lệnh TRx = 0.

Ví dụ 3: Tạo xung có tần số 200Hz bằng chế độ tự nạp lại của Timer 2, tần số dao động thạch anh là 12MHz.

```
#include<reg52.h>
#include<stdio.h>
sbit F = P1^1;
void main(void)
      {
      T2MOD = 0X03;          //DCEN = 1, Timer đếm tiến
      RCAP2H = TH2= -2500/256;//hệ số chia là 5000
      RCAP2L = TL2= -2500%256;
      TR2 = 1;
           while(1)
           {
           while (!TF2);      //chờ cờ tràn
           TF2 = 0;           //xoá cờ tràn
           F = ~F;            //đảo mức logic chân P1.1
      }
}
```

Ví dụ 4: Viết chương trình để chân P1.0 của vi điều khiển 89S52 chuyển từ 0 lên 1 sau 0,25 giây kể từ khi bật nguồn hoặc reset. Cho thạch anh có tần số dao động là 24MHz.

Tần số dao động của thạch anh là 24MHz nên tần số xung nhịp cung cấp cho các bộ Timer là 24/2 = 12MHz, tức chu kỳ là 0,0000005 giây. Để tạo khoảng thời gian trễ là 1 giây thì Timer phải đếm 0,25/0,0000005 = 50000 xung. Muốn vậy ta sử dụng Timer 0 hoạt động định thời chế độ 1, giá trị đầu là –50000.

```
#include<stdio.h>
#include<reg52.h>
sbit output = P1^0;
void main (void)
{
      output = 0 ;
      TMOD = 0X01;        //Timer 0 mode 1
      TH0 = -50000/256;   //nạp giá trị đầu -50000 cho Timer 0
      TL0 = -50000%256;
      TR0 = 1;
      while (!TF0);       //chờ tràn
      TR0 = 0;
      output = 1 ;
}
```

Ví dụ 5: Đo tần số của xung ngoài đưa đến chân T0 hoặc T1 của vi điều khiển.

Để đo tần số ta cần phải đếm được số xung trong một khoảng thời gian chính xác. Muốn vậy ta phải sử dụng hai bộ Timer của vi điều khiển:

– Bộ Timer thứ nhất hoạt động ở chế độ đếm dùng để đếm số xung của tín hiệu cần đo tần số trong một khoảng thời gian xác định (do Timer thứ hai tạo ra).

– Bộ Timer thứ hai có nhiệm vụ tạo khoảng thời gian chính xác cho bộ Timer thứ nhất làm việc (hẹn giờ).

Chương trình sau được viết để đo tần số của xung đưa vào ở chân T0 của vi điều khiển. Timer 1 được dùng để tạo khoảng thời gian đếm là 0,1 giây còn Timer 0 có nhiệm vụ đếm xung trong khoảng thời gian 0,1 giây do Timer 1 tạo ra. Kết quả đo tính bằng Hz và được lưu trong biến f.

```
#include<stdio.h>
#include<reg52.h>
#include<math.h>
unsigned long int dem, n;
double f;
void main (void)
      {
      TMOD = 0X25;
      /*Timer 0 làm bộ đếm, chế độ 1 đếm xung ở chân T0;
      Timer 1 là định thời chế độ 2*/
      IE = 0x88;
      IP = 0x08;
      TH0 = TL0 = 0;// xóa Timer 0
      TH1 = TL1 = -100;// nạp giá trị đầu cho Timer 1
      dem = 0;
      while (1)
           {
           TR1 = TR0 = 1;
           while (dem<1000);/* chờ hết khoảng thời gian
           0,1 giây*/
           n = TH0*256 + TL0;/* đổi giá trị đếm được
           thành số thập phân*/
           f = n*10; /*đơn vị Hz, nhân 10 vì chỉ đếm trong 0.1s*/
           }
      }
void ngatT1 (void) interrupt 3
      {
      dem++;
      TF1 = 0;//xóa cờ ngắt TF1 để Timer 1 tiếp tục đếm
      TR1 = 1;
      }
```

Trong ví dụ này, thời gian đếm là 0,1 giây nên ta có thể đo được tần số từ 10Hz đến 655350Hz (tương ứng với giá trị đếm trong Timer 0 từ 1 đến 65535). Muốn đo tần số thấp hơn ta phải tăng thời gian đếm lên và ngược lại, muốn đo tần số cao hơn ta phải giảm thời gian đếm đi. Thời gian đếm có thể được điều chỉnh bằng hai cách:

– Thay đổi số lần tràn của Timer 1, tức là thay đổi giá trị '1000' trong mệnh đề điều kiện (dem<1000).

– Thay đổi tốc độ tràn của Timer 1, tức là thay đổi giá trị nạp ban đầu cho TH1, TL1.

3.2.7. Tạo xung PWM

Kỹ thuật điều chế độ rộng xung (PWM) được sử dụng rất rộng rãi trong điều khiển, nó có ưu điểm là phạm vi điều chỉnh rộng, khả năng điều chỉnh gần như vô cấp. Xung PWM có thể dùng để điều chỉnh tốc độ động cơ điện một chiều, điều chỉnh công suất lò nhiệt, điều chỉnh điện áp trong các mạch ổn áp... Trong phần này chúng ta sẽ tìm hiểu phương pháp sử dụng các bộ timer của 8051 để tạo ra xung có tỷ số độ rộng mong muốn.

Ví dụ 1: Tạo xung tần số 1kHz ở chân P2.0 của 8952, tỷ số độ rộng xung 30%. Biết thạch anh có tần số dao động là 12MHz.

Trong ví dụ 1 mục 3.2.6, cứ sau 20 xung nhịp cấp cho Timer thì đầu ra P1.0 lại lật mức, kết quả là xung tạo ra ở P1.0 có tỷ số độ rộng xung là 50%.

Trong ví dụ này cần tạo ra xung 1kHz với tỷ số độ rộng xung 30% nên ta có:

– Hệ số chia tần số là 1MHz/1kHz = 1000, giá trị này lớn hơn 256 nên ta phải sử dụng timer 16 bit.

– Để xung ra có tỷ số độ rộng xung là 30% thì cứ 300 xung nhịp của Timer đầu ra P2.0 ở mức 1 tiếp đến 700 xung nhịp tiếp theo đầu ra P2.0 ở mức 0...

```
#include<reg52.h>
sbit F = P2^0;
void main(void)
{
TMOD = 0X01; //Timer 0 mode 1
while(1)
      {
      F = 1;
      TR0 = 0;//dừng Timer 0
      TH0 = -300/256;//300 xung đầu tiên
      TL0 = -300%256;
      TR0 = 1;
      while (!TF0);//chờ cờ tràn
```

```
        TF0 = 0;//xoá cờ tràn
        F = 0;
        TR0 = 0;//dừng Timer 0
        TH0 = -700/256;//700 xung tiếp theo
        TL0 = -700%256;
        TR0 = 1;
        while (!TF0);//chờ cờ tràn
        TF0 = 0;//xoá cờ tràn
}
}
```

Ví dụ 2: Tạo xung tần số 1kHz ở chân P2.0 của 8952, biết xung thứ nhất có tỷ số độ rộng xung 25%, xung thứ hai có tỷ số độ rộng xung 50%, xung thứ ba có tỷ số độ rộng xung 75%, xung thứ tư có tỷ số độ rộng xung 25%... Cho thạch anh có tần số dao động là 12MHz.

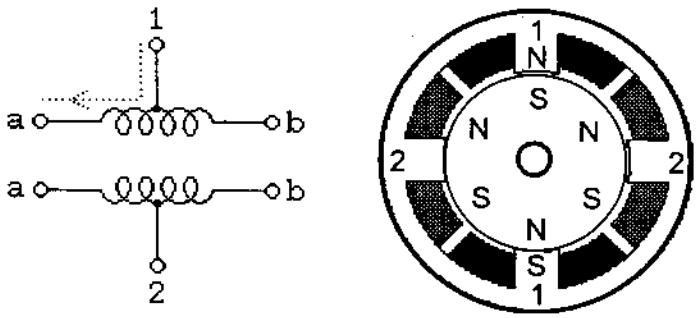
```
#include<reg52.h>
#include<math.h>
sbit F = P2^0;
void taoxung(float tyso) ;
void main(void)
{
      TMOD = 0X01; //Timer 0 mode 1 16 bit
      while(1)
      {
      taoxung(0.25) ;
      taoxung(0.50) ;
      taoxung(0.75) ;
      }
}
void taoxung(float tyso)
{
      F = 1;
      TR0 = 0;
      TH0 = -(unsigned int) (tyso*1000)/256;
      TL0 = -(unsigned int) (tyso*1000)%256;
      TR0 = 1;
      while (!TF0);
      TF0 = 0;
      F = 0;
      TR0 = 0;
      TH0 = -(unsigned int) ((1-tyso)*1000)/256;
      TL0 = -(unsigned int) ((1-tyso)*1000)%256;
      TR0 = 1;
      while (!TF0);
      TF0 = 0;
}
```

3.2.8. Điều khiển động cơ bước

Động cơ bước là một cơ cấu chấp hành rất thông dụng trong các ứng dụng điều khiển vị trí chính xác, ta thường gặp động cơ bước để dịch

chuyển đầu từ của ổ đĩa mềm, dịch chuyển giấy trong máy in kim và máy in phun, dịch chuyển mũi khoan trong máy khoan lỗ mạch in, máy gia công cơ khí...

Có nhiều loại động cơ bước được sử dụng trong thực tế, tuy nhiên trong phần này chúng ta chỉ tìm hiểu về điều khiển động cơ bước đơn cực. Hình 3.14 minh hoạ một động cơ bước đơn cực và các cuộn dây của nó.



Hình 3.14. Động cơ bước đơn cực và các cuộn dây của nó.

– Điều khiển một pha: cấp điện lần lượt cho từng cuộn dây của động cơ bước để làm quay rôto của nó như trong bảng 3.5. Trong bảng này ký hiệu '1' có nghĩa là cuộn dây được cấp điện. Với một động cơ 200 bước thì với phương pháp này các bước của môtơ sẽ từ : 0^0; $1,8^0$; $3,6^0$;... $258,2^0$; động cơ quay 200 bước/vòng.

BẢNG 3.5. ĐIỀU KHIỂN MỘT PHA

Bước thứ	Cuộn 1a	Cuộn 2a	Cuộn 1b	Cuộn 2b
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	1	0	0	0
6	0	1	0	0
7	0	0	1	0
8	0	0	0	1

– Điều khiển hai pha: cấp điện lần lượt cho từng cặp cuộn dây của động cơ bước để làm quay rôto của nó như trong bảng 3.6. Với một động cơ 200 bước thì với phương pháp này các bước của môtơ sẽ từ : 0^0; $1,8^0$; $3,6^0$;... $258,2^0$; động cơ quay 200 bước/vòng.

BẢNG 3.6. ĐIỀU KHIỂN HAI PHA

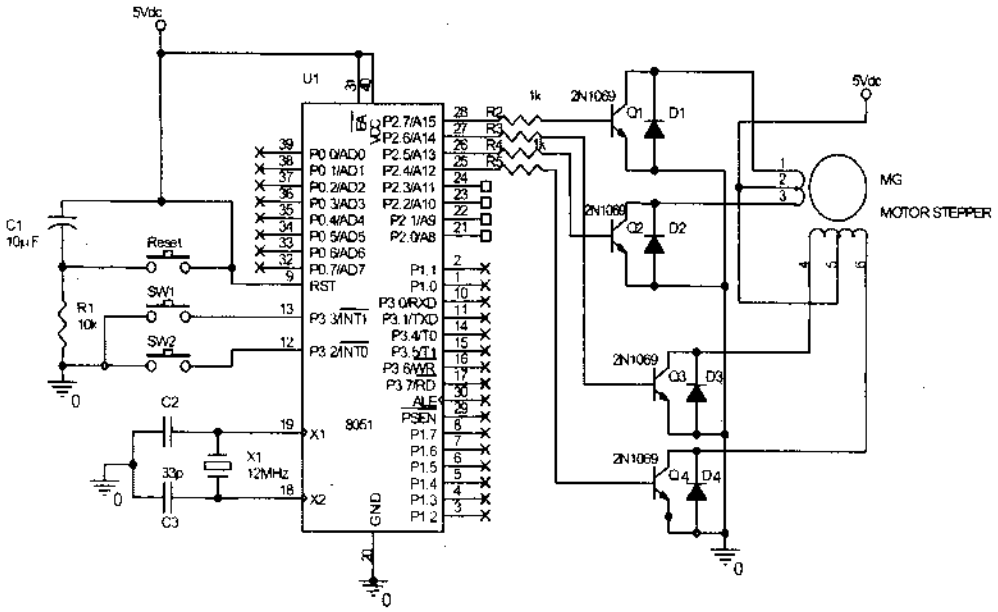
Bước thứ	Cuộn 1a	Cuộn 2a	Cuộn 1b	Cuộn 2b
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1
5	1	1	0	0
6	0	1	1	0
7	0	0	1	1
8	1	0	0	1

– Điều khiển hỗn hợp một pha và hai pha (chế độ nửa bước): cấp điện cho các cuộn dây của động cơ bước như trong bảng 3.7. Với một động cơ 200 bước được điều khiển theo phương pháp này sẽ quay 400 bước/vòng, các bước của môtơ sẽ là : 0^0; $0,9^0$; $1,8^0$; $2,7^0$; $3,6^0$; $4,5^0$;... $359,1^0$.

BẢNG 3.7. ĐIỀU KHIỂN HỖN HỢP MỘT PHA VÀ HAI PHA

Bước thứ	Cuộn 1a	Cuộn 2a	Cuộn 1b	Cuộn 2b
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

Sau đây là ví dụ về mạch điện và chương trình điều khiển động cơ bước quay thuận và quay ngược theo phương pháp điều khiển một pha và hai pha.

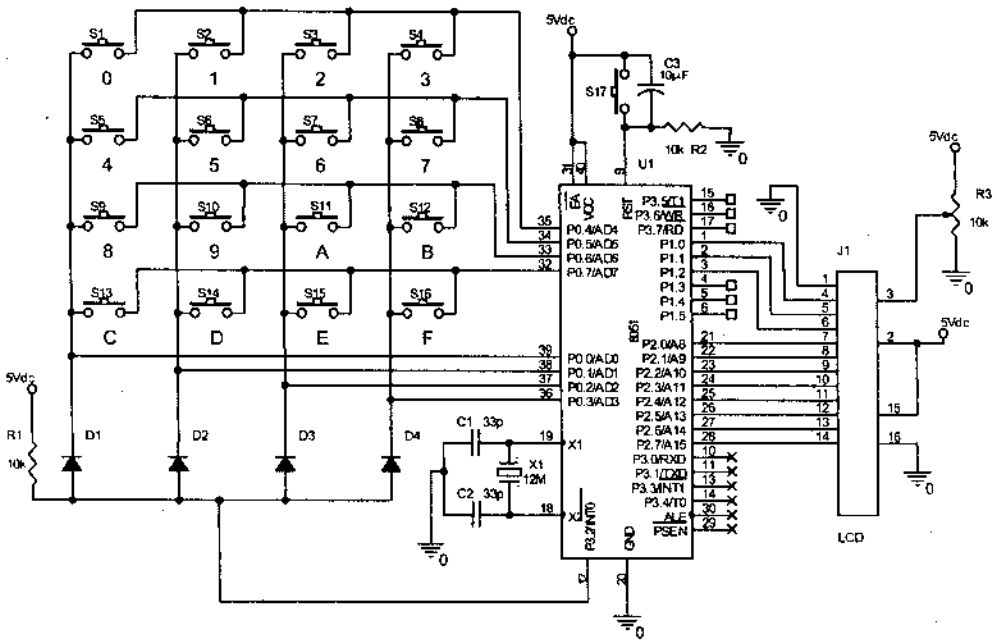


Hình 3.15. Mạch điều khiển động cơ bước.

```
#include<reg52.h>
#include<stdio.h>
#include<intrins.h>
int j;
void delay(long int t)
      {
      long int i;
      for (i = 1;i<= t;++i);
      }
void dkmotor_thuan(unsigned char b,int n,int t)
      {
      for (j = 1;j<= n;++j)
            {
            P2 = b;
            delay(t);
            b = _cror_(b,1);
            }
      }
void dkmotor_nguoc(unsigned char b,int n,int t)
      {
      for (j = 1;j<= n;++j)
            {
            P2 = b;
            delay(t);
            b = _crol_(b,1);
            }
      }
void main (void)
      {
```

```
while(1)
      {
      /* motor quay 8 bước theo chiều kim đồng hồ
      ở chế độ 1 pha*/
      dkmotor_thuan(0x11,8,5); //P2 = 00010001B;
      /* motor quay 8 bước theo chiều kim đồng hồ
      ở chế độ 2 pha*/
      dkmotor_thuan(0x33,8,5); //P2 = 00110011B;
      /* motor quay 8 bước theo chiều ngược kim đồng
      hồ ở chế độ 1 pha*/
      dkmotor_nguoc(0x88,8,5); //P2 = 10001000B;
      /* motor quay 8 bước theo chiều ngược kim đồng
      hồ ở chế độ 2 pha*/
      dkmotor_nguoc(0x99,8,5); //P2 = 10011001B;
      }
      }
```

3.2.9. Giao tiếp với bàn phím 16 phím



Hình 3.16. Mạch giao tiếp bàn phím 16 phím và LCD 16x2.

Chương trình sau sẽ đọc phím và hiển thị số hiệu phím được bấm trên màn hình LCD (các phím tương ứng với các số từ 0 đến 15):

```
#include<reg51.h>
#include<string.h>
#include<math.h>
#include<intrins.h>
#include<stdlib.h>
```

```
sbit  RS = P1^0;
sbit  RW = P1^1;
sbit  EN = P1^2;

sbit   hang0 = P0^4;
sbit   hang1 = P0^5;
sbit   hang2 = P0^6;
sbit   hang3 = P0^7;
char   str;
void  delay(unsigned long int n)
     {
     int i;
     for(i = 0;i<n;i++);
     }

void  delay30ms(void)
     {
     TMOD = 0X10;
     TH1 = 35535/256;
     TL1 = 35535%256;
     TR1 = 1;
     while(!TF1);
     TR1 = TF1 = 0;
     }
void delay10ms(void)
     {
     TMOD = 0x01;
     TH0 = 55535/256;
     TL0 = 55535%256;
     TR0 = 1;
     while(!TF0);
     TR0 = TF0 = 1;
     }
void write_command (unsigned char LCD_command)
     {
     delay10ms();
     P2 = LCD_command;
     RS = 0;
     RW = 0;
     EN = 1;
     delay(50);
     EN = 0;
     delay(50);
     }
```

```
void write_data (unsigned char LCD_data)
     {
     delay10ms();
     P2 = LCD_data;
     RS = 1;
     RW = 0;
     EN = 1;
     delay(50);
     EN = 0;
     delay(50);
     }
void write_string(char *s)
     {
     while(*s)
     {
     write_data(*s); /*Ghi mot chuoi ky tu ra LCD*/
     s++;
     }
     }
void init(void)
     {
     write_command(0x03);
     write_command(0x38);
     write_command(0x06);
     write_command(0x0e);
     }
   unsigned char i,j,x;
   sbit m_hang[4];
void   main(void)
{
   init();
   while(1)
         {
         m_hang[0] = hang0;
         m_hang[1] = hang1;
         m_hang[2] = hang0;
         m_hang[3] = hang3;
         write_command(0x01);
         write_command(0x80);//lcd_locate(0,0);
         write_string("   Phim so:  ");
//đặt giá trị 0 cho từng cột, kiểm tra từng hàng
         x = 0xFE;
         for (i = 0;i< = 3;++i)
         {
```

```
                P0 = x;
                x = _crol_(x,1);
                for (j = 0;j< = 3;++j)
                {
                        if(m_hang[j] == 0)
                        itoa(j*4+i,&str);
                        write_string(&str);
                }
        }
}
```

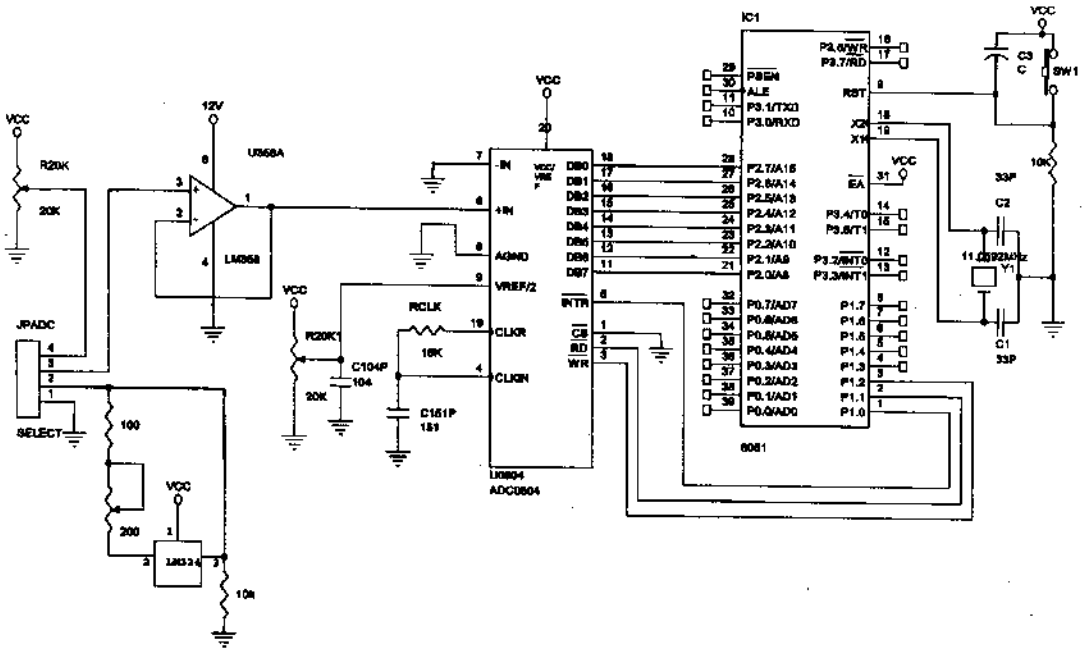
3.2.10. Giao tiếp với bộ biến đổi tương tự – số (ADC)

Các bộ vi xử lý và vi điều khiển nói chung chỉ có thể làm việc với các tín hiệu số, cầu nối giữa chúng và các thiết bị tương tự chính là các bộ DAC và ADC. Trên hình 3.17 là một ứng dụng đo điện áp tương tự, điện áp được đo đưa đến chân V_{in} (chân số 6) của ADC0804, chân V_{ref} dùng để chỉnh dải điện áp đầu vào, đầu ra của ADC0804 là tín hiệu số biến đổi từ 0 đến 255. Điện áp đưa vào ADC được lấy từ bộ cảm biến nhiệt LM334 (khi đo nhiệt độ) hoặc từ biến trở 20k (khi đo điện áp một chiều).

Từ ứng dụng đơn giản này có thể phát triển thành các ứng dụng đo lường các đại lượng vật lý phong phú hơn nếu kết hợp thêm các bộ cảm biến như cảm biến nhiệt, cảm biến độ ẩm, cảm biến cường độ sáng...

Chương trình minh hoạ việc đo điện áp một chiều như sau:

```
#include<stdio.h>
#include<reg52.h>
#include<math.h>
unsigned char x;
sbit INTR_ADC = P1^0;
sbit RD_ADC = P1^1;
sbit WR_ADC = P1^2;
int t;
float volt;
void main (void)
{
P2 = 0xFF;
while(1)
        {
        RD_ADC = 0;
        WR_ADC = 0;
        for (t = 1;t< = 3;++t);
        WR_ADC = 1;
        while (INTR_ADC); //chờ biến đổi xong
        x = P2;    /*đọc giá trị điện áp biến đổi được*/
        volt = (float)(x*5)/255;// giá trị điện áp đầu vào
        }
}
```



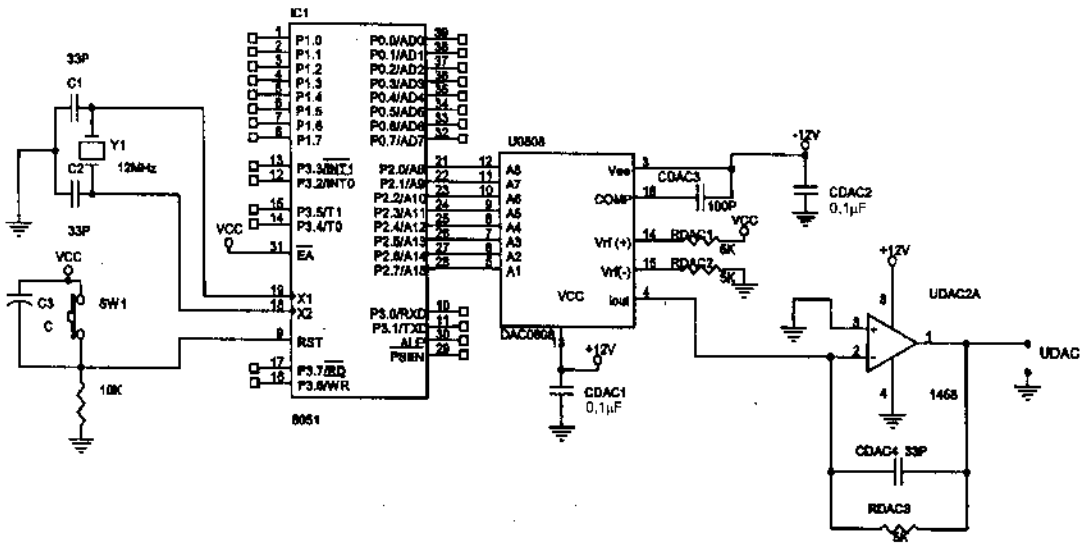
Hình 3.17. Mạch ghép nối với ADC0804.

3.2.11. Giao tiếp với bộ biến đổi số – tương tự (DAC)

Hình 3.18 là mạch ghép nối giữa bộ biến đổi số – tương tự 8 bit DAC0800 và vi điều khiển AT89S52. Chương trình dưới đây sẽ tạo điện áp hình răng cưa ở đầu ra của DAC.

```
#include<stdio.h>
#include<math.h>
#include<reg52.h>
unsigned char value;
void delay (void)
{
      TMOD = 0X10;
      TH1 = (-9207)/256;
      TL1 = (-9207)%256;
      TR1 = 1;
      while(!TF1);
      TF1 = 0;
      TR1 = 0;
}
void main (void)
{
      value = 0X00;
      while(1)
      {
      while(value<0XFF)
      {
```

```
            P2 = value;
            value = value+5;
            delay();
        }
        while(value>0)
        {
            P2 = value;
            value = value-5;
            delay();
        }
        }
}
```



Hình 3.18 Mạch ghép nối với DAC0808.

3.2.12. Giao tiếp với vi mạch 8255A

8255A là vi mạch vào/ra song song khả trình có cùng kiến trúc BUS với họ 8051 (BUS INTEL). Vấn đề chính khi giao tiếp với các vi mạch có kiến trúc bus như 8255A chính là phải tách riêng phần địa chỉ và dữ liệu trên các chân đa hợp AD0–AD7.

Trên hình 3.19 là sơ đồ ghép nối 8255A với AT89S52, vi mạch 74LS573 với chức năng chốt sẽ cho phép tách riêng phần địa chỉ trên các chân đa hợp dữ liệu và địa chỉ, vi mạch 74LS138 có chức năng giải mã (trong trường hợp có nhiều vi mạch khả trình cùng được ghép với AT89S52 thì vi mạch này là cần thiết).

Theo sơ đồ ghép nối trên thì địa chỉ của AT89S52 dành cho 8255A sẽ là:

– Cổng A (PA): 0000H

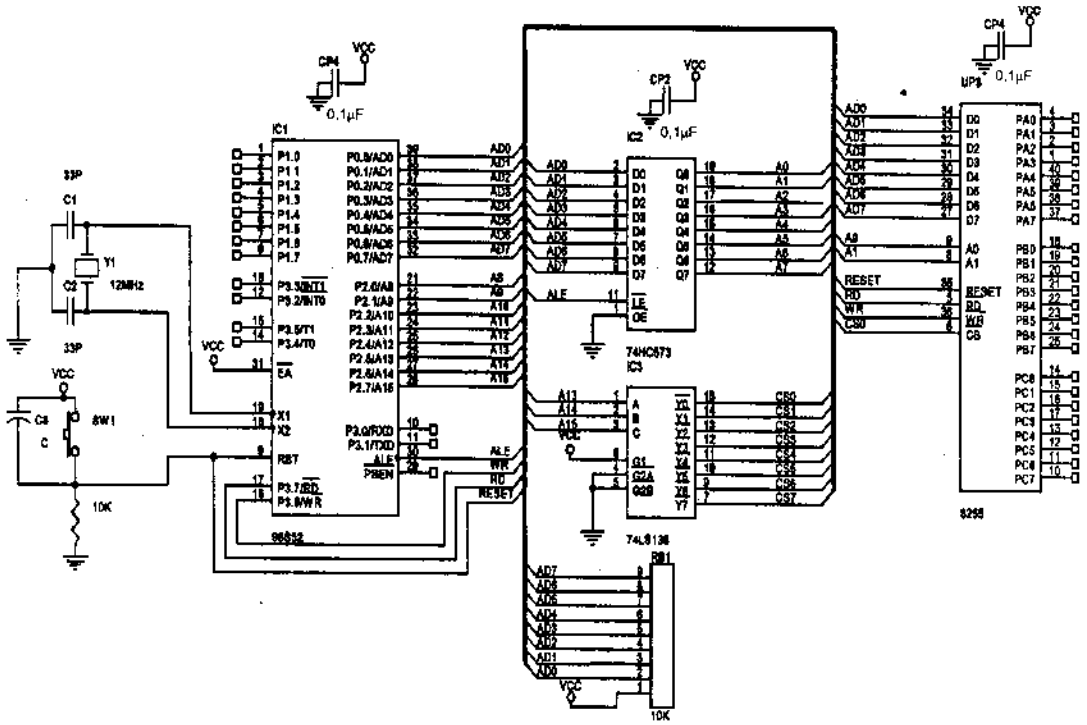
– Cổng B (PB): 0001H

– Cổng C (PC): 0002H

– Thanh ghi điều khiển (CREG): 0003H

Chương trình dưới đây sẽ đọc số liệu ở cổng C rồi xuất ra cổng A, xuất giá trị FF ra cổng B của 8255A.

```
#include <stdio.h>
#include <reg52.h>
//khai báo địa chỉ của 8255A
xdata unsigned char PA     _at_     0x0000;
xdata unsigned char PB     _at_     0x0001;
xdata unsigned char PC     _at_     0x0002;
xdata unsigned char CREG   _at_     0x0003;
char x,y;
void main(void)
{
      CREG = 0x89 ;   //ghi từ điều khiển
      x = PC ;        //đọc cổng C
      PA = x ;        //xuất ra cổng A giá trị đọc được  từ cổng C
      y = 0xff;
      PB = y;         //xuất(ghi) ra cổng A giá trị FF(H)
}
```



Hình 3.19. Mở rộng cổng vào ra với vi mạch 8255A.

3.2.13. Xử lý đa nhiệm – thời gian thực với RTX51

Đối với một chương trình thông thường, các dòng lệnh được thực thi tuần tự, dòng lệnh ở chương trình chính (main) sẽ được thực thi trước, các chương trình con được thực thi khi chương trình chính gọi đến nó. Xét chương trình:

```
void main (void)
{
      while (1)           /* vòng lặp vô tận */
      {
           do_something();/* thực hiện một tác vụ nào đó */
      }
}
```

Với chương trình trên, tại một thời điểm, CPU chỉ thực hiện một tác vụ "do_something", nếu như ứng dụng yêu cầu cần có nhiều tác vụ cần xử lý cùng một lúc thì khuôn mẫu chương trình này không thể đáp ứng.

Nhược điểm của chương trình viết theo khuôn mẫu trên cũng có thể được khắc phục bằng cách đưa nhiều tác vụ vào trong một vòng lặp, ví dụ:

```
void main (void)
{
int counter = 0;
while (1)                 /* vòng lặp vô tận–super loop */
    {
    check_serial_io();    /* Kiểm tra vào ra */
    process_serial_cmds ();/* vào ra dữ liệu */
    check_kbd_io();       /* Kiểm tra keyboard */
    process_kbd_cmds();   /* Đọc mã quét */
    adjust_ctrlr_parms();/* Điều chỉnh tham số bộ điều
                            khiển */
      counter++;          /* đếm số lần thực hiện */
      }
}
```

Chương trình viết theo mô hình này rõ ràng có thể thực hiện được nhiều tác vụ tuy nhiên do các tác vụ được thực hiện tuần tự mà không có sự phân chia thời gian hợp lý nên có thể có tình trạng khi tác vụ này cần thực hiện thì CPU lại đang thực hiện tác vụ kia, chẳng hạn khi có dữ liệu đưa tới, lẽ ra phải thực hiện ở process_serial_cmds () thì CPU lại đang thực hiện ở tác vụ check_kbd_io (). Nói cách khác hệ thống không đáp ứng được yêu cầu của hệ thống thời gian thực.

Để phát triển các ứng dụng thời gian thực cần phải xây dựng một hệ điều hành thời gian thực (RTOS – Realtime Operating System). Hãng Keil có cung cấp một số hệ điều hành thời gian thực tích hợp trong trình dịch C để phục vụ cho việc phát triển các ứng dụng thời gian thực như

RTX51, RTX51 Tiny, ARTX166, RTX166 Tiny... Hệ điều hành thời gian thực được thiết kế để giải quyết hai vấn đề cơ bản của các chương trình nhúng là:

– Đa nhiệm: nhiều tác vụ được thực hiện đồng thời.

– Điều khiển thời gian thực: các thao tác phải được thực thi trong một khoảng thời gian xác định.

Đối với các thiết kế sử dụng vi điều khiển họ 8051 có:

– RTX51 là hệ điều hành cho các ứng dụng dựa trên 8051, kích thước 8kB, có thể định nghĩa tới 256 tác vụ với 4 mức ưu tiên. RTX51 sử dụng Timer 0, 1 và 2 để tạo Timer_tick.

– RTX51 Tiny là hệ điều hành thu gọn cho các ứng dụng dựa trên 8051, kích thước 900 byte, có thể định nghĩa tới 16 tác vụ với một mức ưu tiên. RTX51 Tiny sử dụng Timer 0 để tạo Timer_tick.

Chương trình viết theo khuôn mẫu của RTX51 hoặc RTX51 Tiny sẽ không có chương trình chính mà chỉ có các *task*, trong đó *task 0* có chức năng khởi tạo cho các *task* khác.

```
void check_serial_io_task (void) _task_ 1
{
/* task 1 Kiểm tra vào ra */
}

void process_serial_cmds_task (void) _task_ 2
{
/* task 2 điều khiển vào ra dữ liệu*/
}

void check_kbd_io_task (void) _task_ 3
{
/* task 3 kiểm tra keyboard */
}

void process_kbd_cmds_task (void) _task_ 4
{
/* đọc mã quét */
}
void startup_task (void) _task_ 0
{
os_create_task (1);  /* Khởi tạo serial_io Task */
os_create_task (2);  /* Khởi tạo serial_cmds Task */
os_create_task (3);  /* Khởi tạo kbd_io Task */
os_create_task (4);  /* Khởi tạo kbd_cmds Task */
os_delete_task (0);  /* Xoá Startup Task */
}
```

Sau đây là một số hàm cơ bản của RTX51 Tiny:

- Hàm os_create_task

Cú pháp	`#include <rtx51tny.h>` `char os_create_task (unsigned char task_id);`
Mô tả	Hàm ***os_create_task*** cho phép khởi tạo một task có số hiệu là task_id. RTX51 tiny cho phép định nghĩa 16 task với các số hiệu từ 0 đến 15.
Giá trị trả về	Hàm **os_create_task** trả về giá trị 0 nếu task_id tương ứng được khởi tạo thành công, ngược lại hàm sẽ trả về giá trị 1.

Ví dụ	`#include <rtx51tny.h>`
	`#include <stdio.h>`
	
	`void new_task (void) _task_ 2`
	`{`
	`.`
	`.`
	`}`
	`void tst_os_create_task (void) _task_ 0`
	`{`
	`.`
	`.`
	`.`
	`if (os_create_task (2))`
	`{`
	`printf ("Couldn't start task 2\n");`
	`}`
	`.`
	`.`
	`}`

- Hàm os_delete_task

Cú pháp	`#include <rtx51tny.h>` `char os_delete_task (unsigned char task_id);`
Mô tả	Hàm **os_delete_task** cho phép dừng một tác vụ có số hiệu là task_id đang thực hiện. Tác vụ có số hiệu task_id sẽ bị xoá khỏi danh sách các tác vụ mà RTX51 đang luân phiên thực hiện.
Giá trị trả về	Hàm **os_delete_task** trả về giá trị 0 nếu tác vụ có số hiệu là task_id được dừng và xóa thành công, trả về giá trị 1 nếu tác vụ có số hiệu là task_id không tồn tại hoặc chưa được dừng.

Ví dụ	`#include <rtx51tny.h>`
	`#include <stdio.h>`
	`void tst_os_delete_task (void) _task_ 0`
	`{`
	`.`
	`.`
	`if (os_delete_task (2))`
	`{`
	`printf ("Couldn't stop task 2\n");`
	`}`
	`.`
	`}`

– Hàm os_wait

Cú pháp	`#include <rtx51tny.h>`
	`char os_wait (`
	`  unsigned char event_sel,`
	`  unsigned char ticks,`
	`  unsigned int dummy);`

Mô tả	Hàm **os_wait dừng** task hiện thời và đợi một sự kiện nào đó. Biến **event_sel** xác định một hoặc một số các sự kiện. Biến **event_sel** có thể là một hoặc kết hợp của các sự kiện sau: K_IVL: Đợi một khoảng thời gian giữa hai sự kiện (khoảng thời gian này được xác định bởi biến **ticks**). K_SIG: Đợi cờ của một task khác. K_TMO: Đợi một khoảng thời gian nghỉ (time–out), khoảng thời gian này được xác định bởi biến **ticks**. Những sự kiện nói trên có thể kết hợp theo cách OR logic, ví dụ **K_TMO | K_SIG** nghĩa là đợi cờ của một task khác hoặc một khoảng thời gian nghỉ. Biến **ticks** xác định số timer _tick cho các sự kiện **K_IVL** hoặc **K_TMO.** Biến **dummy** hỗ trợ cho RTX51 Full, với RTX51 Tiny biến này được gán bằng 0.

Ví dụ	`#include <rtx51tny.h>`
	`#include <stdio.h>  /* for printf */`
	
	`void tst_os_wait (void) _task_ 9`
	`{`
	`while (1)`
	`  {`
	`  char event;`
	`  event = os_wait (K_SIG + K_TMO, 50, 0);`
	`  switch (event)`
	`  {`
	`  default:  /* this never happens */`
	`  break;`
	`case TMO_EVENT:  /* time-out */`
	`  /* 50 ticks time-out */`
	`  break;`
	`case SIG_EVENT:  /* signal received */`
	`  break;`
	`  }`
	`  }`
	`}`

- Hàm os_switch_task

Cú pháp	`#include <rtx51tny.h>` `char os_switch_task (void);`
Mô tả	Hàm ***os_switch_task*** cho phép dừng tác vụ để tác vụ khác được thực hiện.
Giá trị trả về	Không

Ví dụ	`#include <rtx51tny.h>`
	`#include <stdio.h>`
	`void long_job (void) _task_ 1`
	`{`
	`float f1, f2;`
	`f1 = 0.0;`
	`while (1)`
	`{`
	`f2 = log (f1);`
	`f1 += 0.0001;`
	`os_switch_task (); // run other tasks`
	`}`
	`}`

RTX51 Tiny sử dụng Timer 0 để tạo ra các sự kiện ngắt định kỳ, các ngắt này được gọi là các *tick* và khoảng thời gian định kỳ xảy ra các tick được gọi là *timer_tick*. Thời gian thực hiện luân phiên các tác vụ ở RTX51 được đo lường bằng đơn vị là timer_tick. Mặc định, một timer_tick bằng 10.000 chu kỳ máy, nghĩa là nếu dùng thạch anh 12MHz thì một timer_tick là 0,01 giây. Tất nhiên, giá trị này có thể thay đổi được bằng việc mở và sửa giá trị trong tập tin CONF_TNY.A51.

Sau đây là hai ví dụ minh hoạ ứng dụng của RTX51 Tiny:

Ví dụ 1: Viết chương trình điều khiển đồng thời 5 đèn LED nối vào các chân P3.4, P3.5, P3.6 và P3.7 để 5 LED này nhấp nháy với chu kỳ khác nhau.

```
#include<reg52.h>
#include<rtx51tny.h>
#define on 0
#define off 1
sbit led1 = P3^4;
sbit led2 = P3^5;
sbit led3 = P3^6;
sbit led4 = P3^7;
void delay(unsigned int n)
{
```

```
        int i;
        for(i = 0;i<n;i++)
        {
        TMOD = 0X01;
        TH0 = 15535/256;
        TL0 = 15535%256;
        TR0 = 1;
        while(!TF0);
        TF0 = TR0 = 0;
        }
}

void nhay1(void) _task_  1
{
        while(1)
        {
        led1 = on;delay(30);
        led1 = off;delay(30);
        }
}

void nhay2(void)  _task_  2
{
        while(1)
        {
        led2 = on;delay(40);
        led2 = off;delay(40);
        }
}
void nhay3(void)  _task_  3
{
        while(1)
        {
        led3 = on;delay(50);
        led3 = off;delay(50);
        }
}

void nhay4(void)  _task_  4
{
        while(1)
        {
        led4 = on;delay(60);
        led4 = off;delay(60);
        }
}
void nhay(void)  _task_  0
{
        os_create_task (1);
        os_create_task (2);
        os_create_task (3);
        os_create_task (4);
        os_delete_task (0);
}
```

Ví dụ 2: Bạn đọc tự tìm hiểu ứng dụng của chương trình này.

```
#include<reg51.h>
#include<rtx51tny.h>
#include<intrins.h>
#defineMODE115//15 phút
#defineMODE230//30 phút
#defineMODE360//60 phút
#defineMODE420//20 phút

sbit D4 = P1^7;
sbit D3 = P1^6;
sbit D2 = P1^5;
sbit D1 = P1^4;
sbit Loa  = P1^3;
sbit Key  = P3^2;
sbit Role = P3^7;
unsigned char MODE;
int d;
struct time
{
unsigned char hour;
unsigned char minute;
unsigned char second;
};
struct time timer;
void delay(void)
{
int i;
for(i = 0;i<7;i++);
}
void common (void) _task_ 1
{
while(1)
{
switch(MODE)
{
case 1:
      D1 = 0;
      D2 = D3 = D4 = 1;
      Role = 0;
      if(timer.minute == MODE1)
      {
           Role = 1;
           D1 = 1;
           MODE = 0;
      }
break;
case 2:
      D2 = 0;
      D1 = D3 = D4 = 1;
      Role = 0;
      if(timer.minute == MODE2)
      {
           Role = 1;
```

```
            D2 = 1;
            MODE = 0;
        }
break;
case 3:
        D3 = 0;
        D2 = D1 = D4 = 1;
        Role = 0;
        if(timer.minute == MODE3)
        {
            Role = 1;
            D3 = 1;
            MODE = 0;
        }
break;
case 4:
        D4 = 0;
        D2 = D3 = D1 = 1;
        if(timer.minute == MODE4)
        {
            Role = ~Role;
            timer.hour = 0;
            timer.minute = 0;
            timer.second = 0;
        }
break;
default:
        D2 = D3 = D1 = D4 = 1;
        timer.hour = 0;
        timer.minute = 0;
        timer.second = 0;
break;
}
}
}
void clock(void) _task_ 4
{
while(1)
{
if(MODE == 4)
{
        D4 = 0;
        D2 = D3 = D1 = 1;
        if(timer.minute == MODE4)
        {
            Role = ~Role;
            timer.hour = 0;
            timer.minute = 0;
            timer.second = 0;
        }
}
}
}

void Buzzer (void) _task_ 2
```

```
{
while(1)
{
      Loa = 0;
      TR1 = 1;
      os_wait (K_TMO, 1, 0); //100ms
      TR1 = 0;
      Loa = 1;
      os_delete_task (2);
}
}
void key(void) _task_ 3
{
while(1)
{
if(!Key)
{
      MODE++;
      if(MODE == 6)MODE = 1;//out of range work
      Role = 1;
      timer.hour = 0;
      timer.minute = 0;
      timer.second = 0;
      for(d = 0;d<100;d++)
      {
           Loa = ~Loa;
           delay();
      }
      Loa = 1;os_wait (K_TMO, 20, 0); //200ms
      while(!Key);

}
}
}
void inctime (void) _task_ 4
{
while(1)
{
if((MODE>0)&&(MODE<5)) //MODE is 1 to 4 range
{
os_wait (K_TMO, 100, 0); //1 s
timer.second++;
if(timer.second == 60)
{
      timer.second = 0;
      timer.minute++;
      if(timer.minute == 60)
      {
           timer.minute = 0;
           timer.hour++;
           if(timer.hour == 24)
           {
                timer.hour = 0;
                timer.minute = 0;
                timer.second = 0;
```

```
            }
        }
}
}
else
{
Role = 1;
timer.hour = 0;
timer.minute = 0;
timer.second = 0;
}
}
}
void startup_task (void) _task_ 0
{
P1 = 0xFF;
Loa = 1;
Role = 1;/*
TMOD| = 0x20;
TH1 = TL1 = -200;
TR1 = 0;
ET1 = 1;
EA = 1;*/
MODE = 0;
timer.hour = 0;
timer.minute = 0;
timer.second = 0;
os_create_task (1);
os_create_task (2);
os_create_task (3);
os_create_task (4);
os_delete_task (0);
}
```

3.2.14. Thiết kế bộ điều khiển nhiệt độ ứng dụng logic mờ

1. Phân tích bài toán

Bài toán đặt ra là thiết kế một bộ điều khiển hoạt động dựa trên cơ sở logic mờ để điều khiển nhiệt độ cho một lò nhiệt kiểu điện trở với dải nhiệt độ khoảng từ 30^0C đến 130^0C. Để thực hiện được thì hệ thống điều khiển này phải có các khối chức năng sau:

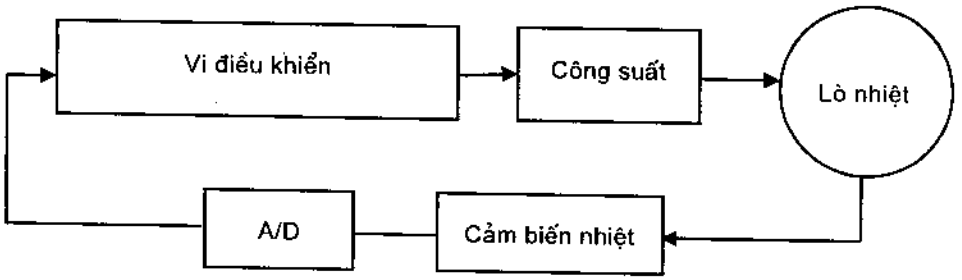
– Cảm biến nhiệt độ: dùng để đo nhiệt độ hiện tại trong lò nhiệt.

– Khối biến đổi A/D: có nhiệm vụ số hoá tín hiệu ra của cảm biến nhiệt để đưa vào bộ vi điều khiển – là nơi cài đặt thuật toán điều khiển mờ.

– Khối vi điều khiển (MCU): có nhiệm vụ thực hiện các chức năng mờ hóa, hợp thành và giải mờ.

– Khối công suất: có đầu vào là đầu ra của khối giải mờ (từ vi điều khiển), có nhiệm vụ điều khiển việc cấp nguồn động lực cho lò nhiệt, qua đó làm thay đổi nhiệt độ trong lò.

Sơ đồ khối của hệ thống được minh hoạ trong hình 3.20.



Hình 3.20. Sơ đồ khối hệ thống điều khiển nhiệt độ.

**** Cảm biến nhiệt:***

Với dải nhiệt độ từ 30^{0}C đến 130^{0}C, ta chọn sử dụng cảm biến nhiệt độ bán dẫn thông dụng là vi mạch LM35 của hãng National Semiconductor.

Vi mạch cảm biến nhiệt LM35 có các đặc điểm sau:

– Chuẩn hoá theo thang đo nhiệt độ Cesius;

– Đầu ra tuyến tính 10mV/^{0}C;

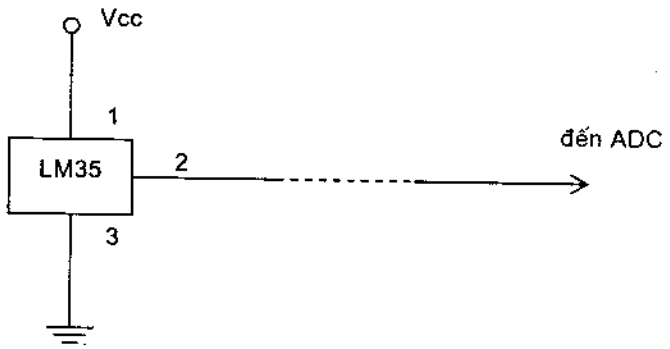
– Dải nhiệt độ đo được từ –55^{0}C đến +150^{0}C tuỳ theo kiểu đóng vỏ;

– Điện áp làm việc từ 4 đến 30V;

– Dòng tiêu thụ rất nhỏ cỡ 60μA, nên nhiệt tự tỏa rất nhỏ hầu như không ảnh hưởng đến kết quả đo.

Sai số nhỏ, chỉ khoảng 0,5^{0}C.

Sơ đồ mạch cảm biến được cho ở hình 3.21.



Hình 3.21. Sơ đồ mạch cảm biến nhiệt độ LM35.

**** Mạch biến đổi A/D:***

Mạch biến đổi A/D có nhiệm vụ biến đổi điện áp tương tự từ cảm biến nhiệt thành tín hiệu số để gửi đến vi điều khiển. Để lựa chọn sử dụng các vi mạch A/D người ta thường căn cứ vào 2 thông số chính là:

– Độ phân giải hay số bit đầu ra, số bit đầu ra của ADC quyết định đến độ chính xác của phép biến đổi, số bit càng lớn thì sai số lượng tử càng nhỏ.

– Tốc độ biến đổi được tính bằng số mẫu hoàn thành trong một giây (samples/s), nó tỷ lệ nghịch với thời gian để hoàn thiện việc biến đổi một mẫu.

Trong ứng dụng điều khiển nhiệt độ, nhiệt độ là một đại lượng biến đổi chậm nên ta chọn vi mạch A/D thông dụng trên thị trường là ADC0804. ADC0804 là bộ biến đổi A/D 8 bit đủ đáp ứng được yêu cầu về độ chính xác đặt ra. ADC0804 có các đặc tính sau:

– Đầu ra được đệm bằng các cổng 3 trạng thái nên có thể ghép trực tiếp vào bus dữ liệu mà không cần mạch đệm dữ liệu ở bên ngoài;

– Thời gian biến đổi cỡ 100μs/mẫu;

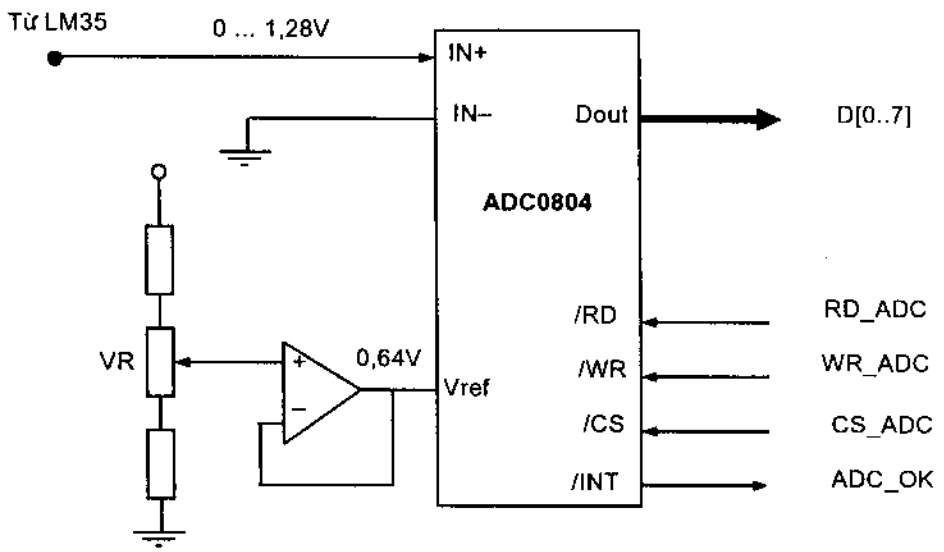
– Không cần hiệu chỉnh điểm 0;

– Khi điện áp nguồn nuôi là 5V thì tín hiệu đầu vào cực đại là 5V;

– Tất cả các tín hiệu đều tương thích mức TTL;

– Dòng tiêu thụ nhỏ, cỡ 1,9mA.

Hình 3.22 mô tả mạch A/D ứng dụng vi mạch ADC0804 để đo nhiệt độ.



Hình 3.22. Sơ đồ mạch biến đổi A/D.

Mạch này gồm hai phần:

– Phần tương tự: Với dải nhiệt độ từ 0 đến 128°C thì điện áp ra của LM35 là từ 0 đến 1,28V, điện áp này được đưa đến lối vào IN+, còn lối vào IN– được nối đất. Với điện áp vào cực đại là 1,28V thì điện áp chuẩn V_{ref} được đặt là 0,64V, việc điều chỉnh V_{ref} được thực hiện bởi biến trở vi chỉnh VR. Khi đó nếu nhiệt độ là 0^0C thì điện áp IN+ là 0V, đầu ra D_{out} là 00000000, vi điều khiển nhận được giá trị là 0; khi nhiệt độ là 128^0C thì điện áp IN+ là 1,28V, đầu ra D_{out} là 11111111, vi điều khiển nhận được giá trị tương ứng là 255.

– Phần giao tiếp với vi điều khiển: để thực hiện giao tiếp với vi điều khiển cần các tín hiệu sau:

+ Tín hiệu chọn vi mạch biến đổi tương tự/số CS_ADC, tích cực thấp, được đưa từ mạch giải mã địa chỉ tới. Khi CS_ADC = 0 thì ADC0804 được chọn, chuẩn bị cho việc đọc số liệu từ ADC.

+ Tín hiệu ra lệnh cho ADC0804 bắt đầu biến đổi WR_ADC, tích cực thấp, được điều khiển trực tiếp bởi vi điều khiển, khi WR_ADC chuyển từ 1 xuống 0 thì ADC bắt đầu quá trình biến đổi.

+ Tín hiệu báo kết thúc biến đổi ADC_OK, tích cực thấp, được nối với chân INTR (interrupt) của ADC0804, mỗi khi ADC0804 hoàn thành việc biến đổi cho một mẫu nó báo hiệu cho bên ngoài bằng một mức logic 0 ở chân này. Tín hiệu này được gửi tới vi điều khiển để báo cho vi điều khiển biết ADC đã biến đổi xong; để không phải hỏi vòng thì tín hiệu này được đưa tới một đầu vào ngắt ngoài của vi điều khiển, mỗi khi ADC biến đổi xong sẽ gây ra một ngắt vi điều khiển, vi điều khiển sẽ chuyển sang chương trình con phục vụ ngắt để đọc số liệu từ ADC.

+ Tín hiệu đọc số liệu RD_ADC, tích cực thấp, sau khi nhận biết ADC đã biến đổi xong bởi mức logic 0 ở ADC_OK, vi điều khiển thực hiện việc lấy số liệu bằng cách chuyển tín hiệu RD_ADC xuống mức thấp, tín hiệu này được nối với chân /RD của ADC0804, lúc này các cổng đệm 3 trạng thái ở đầu ra D_{out} của ADC0804 mở, kết quả biến đổi được đưa lên bus dữ liệu.

**** Khối vi điều khiển:***

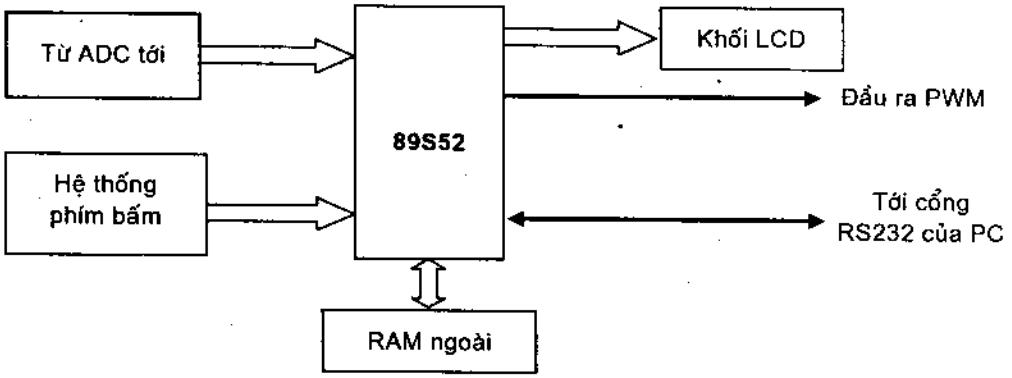
Vi điều khiển là nơi thực hiện thuật toán điều khiển, bao gồm các các chức năng: mờ hoá, hợp thành (hay còn gọi suy diễn và hợp mờ) và giải mờ. Ngoài ra để có thể giao tiếp được với người sử dụng thì ta cần phải bổ sung:

– Màn hình hiển thị LCD để hiển thị các thông tin như nhiệt độ hiện tại trong lò (nhiệt độ thực tế), nhiệt độ do người sử dụng đặt (nhiệt độ mong muốn).

– Hệ thống các phím bấm chức năng để người sử dụng có thể nhập giá trị nhiệt độ đặt trực tiếp.

– Cổng giao tiếp với PC để từ PC người sử dụng có thể giám sát hoạt động của hệ thống.

Sơ đồ khối của khối vi điều khiển được cho ở hình 3.23.



Hình 3.23. Sơ đồ khối của khối vi điều khiển.

Trong khối CPU chúng tôi có sử dụng các linh kiện sau:

+ Vi điều khiển 89S52.

+ Vi mạch RAM ngoài 62256 để chứa các mảng số liệu phục vụ cho quá trình suy diễn, hợp mờ và giải mờ (lượng RAM nội của 89S52 không đủ chứa các mảng này).

+ Vi mạch vào/ra lập trình được 8255 để mở rộng số đầu vào/ra của vi điều khiển.

+ Vi mạch MAX232 làm nhiệm vụ ghép nối vi điều khiển với PC qua cổng RS232.

+ Bộ hiển thị LCD 16x2 để hiển thị nhiệt độ đo và nhiệt độ đặt.

**** Khối công suất:***

Khối công suất có nhiệm vụ nhận xung điều khiển PWM từ vi điều khiển đến để điều khiển công suất nguồn cung cấp cho lò nhiệt.

Công suất đặt vào lò được tính theo công thức:

$$P = I^2 . R \qquad (3.1)$$

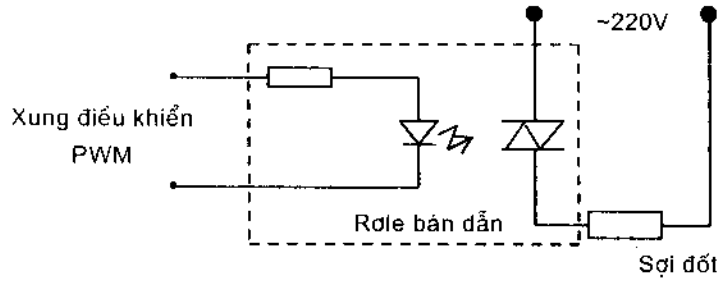
Trong đó: I – dòng điện chạy qua sợi đốt;

R – điện trở của sợi đốt.

Vậy có hai phương án thay đổi công suất P:

– Điều chỉnh về phía tiêu thụ: thay đổi điện trở của sợi đốt R, phương án này ít được sử dụng bởi tính không liên tục và hạn chế phạm vi điều khiển.

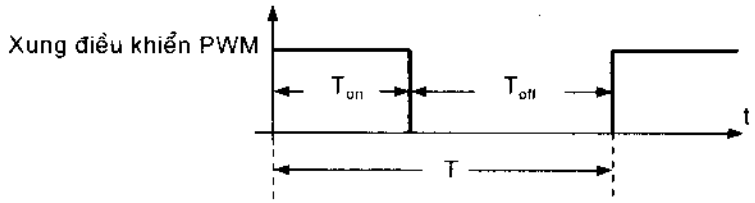
– Điều chỉnh về phía cung cấp tức làm thay đổi cường độ dòng điện chạy qua sợi đốt bằng cách thay đổi điện áp cấp cho sợi đốt. Muốn vậy ta điều khiển đóng/cắt liên tục nguồn cung cấp cho sợi đốt, qua đó làm thay đổi điện áp trung bình trên sợi đốt. Trong ví dụ này sử dụng phương pháp điều khiển công suất lò nhiệt bằng phương pháp điều chế độ rộng xung điều khiển rơle bán dẫn, sơ đồ mạch điện được cho ở hình 3.24.



Hình 3.24. Mạch điều khiển công suất lò nhiệt dùng rơle bán dẫn.

Nguyên lý điều khiển:

Xung điều khiển PWM do vi điều khiển tạo ra có dạng như trong hình 3.25. Trong đó chu kỳ xung T được giữ cố định, $T = T_{on} + T_{off}$ = hằng số, còn thời gian T_{on} thay đổi. Khi T_{on} thay đổi từ 0 đến T thì độ rộng xung $\mu = \frac{T_{on}}{T_{off}} x100\%$ thay đổi từ 0% đến 100%.



Hình 3.25. Xung điều khiển PWM.

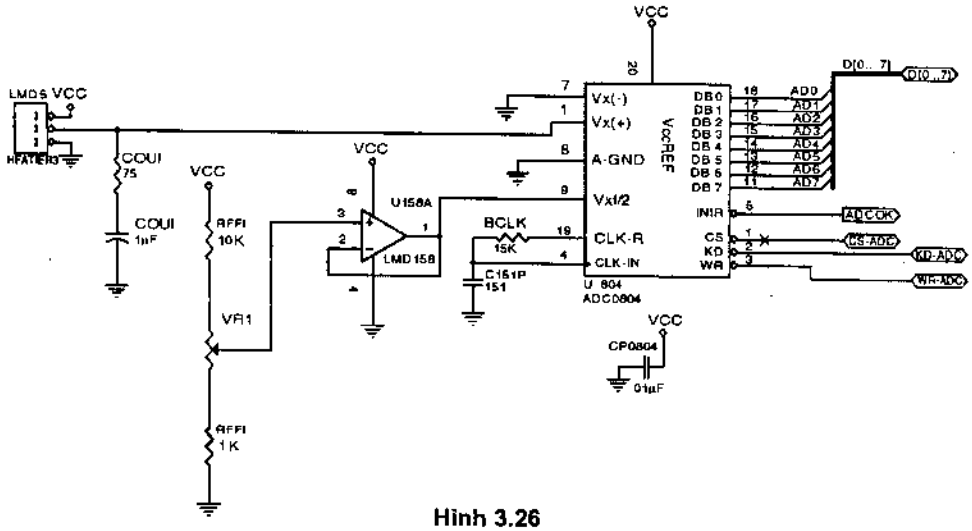
Khi $\mu = 0\%$, tức là $T_{on} = 0$, $T_{off} = T$ thì rơle không hoạt động, sợi đốt không được cấp điện hoàn toàn, công suất P = 0.

Khi $\mu = 100\%$, tức là $T_{on} = T$, $T_{off} = 0$ thì rơle đóng liên tục, sợi đốt được cấp nguồn liên tục, công suất trên sợi đốt là cực đại $P = P_{max}$.

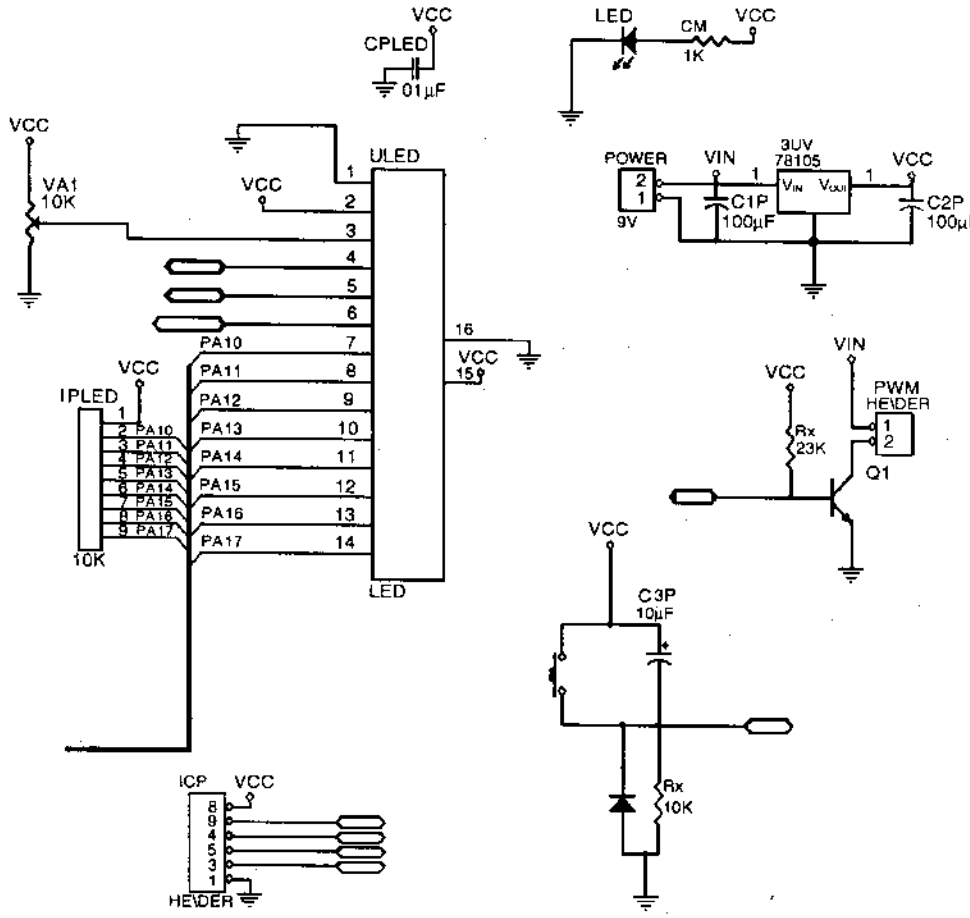
Như vậy khi thay đổi tỷ số độ rộng xung μ ta sẽ điều khiển được công suất lò nhiệt từ 0 đến giá trị cực đại của nó.

2. Sơ đồ nguyên lý bộ điều khiển nhiệt độ

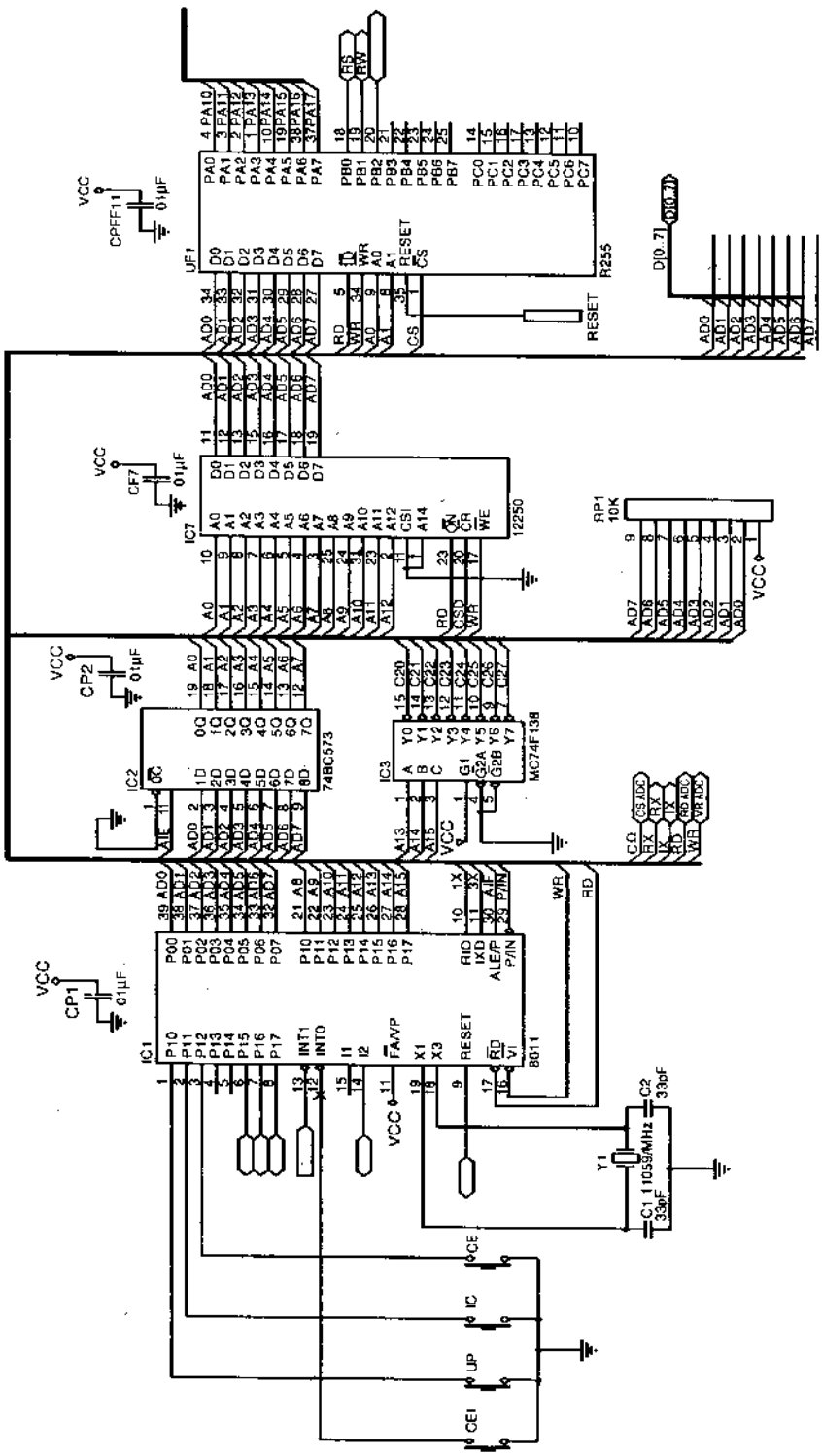
Từ các phân tích trên ta có thể xây dựng được sơ đồ nguyên lý của bộ điều khiển nhiệt độ ứng dụng logic mờ như hình 3.26, 3.27, 3.28.



Hình 3.26



Hình 3.27


VCC
CPFF11
0.1µF
UF1
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
RS
RW
WR
RESET
CS
R255
D[0..7]
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
RD
CF7
IC7
12250
RP1
10K
CP2
IC2
74BC573
IC3
MC74F138
CP1
IC1
8011
INT1
INT0
FA/VP
X1
X3
ALE/P
P/IN
C2
33pF
Y1
11.0592MHz
C1
33pF
CE
IC
UP
CEI

Hình 3.28

3. Thiết kế chương trình điều khiển

Chương trình điều khiển được nạp xuống vi điều khiển 89S52 để thực hiện các chức năng:

– Mờ hoá;

– Hợp thành (suy diễn và hợp mờ);

– Giải mờ.

Cơ sở và phương pháp thực hiện các chức năng này như sau:

*** Mờ hoá**

Ta biết rằng bộ điều khiển mờ có một đầu vào và một đầu ra:

+ Đầu vào DT là sai lệch nhiệt độ giữa nhiệt độ thực tế đo được ở lò nhiệt với nhiệt độ do người sử dụng đặt trước:

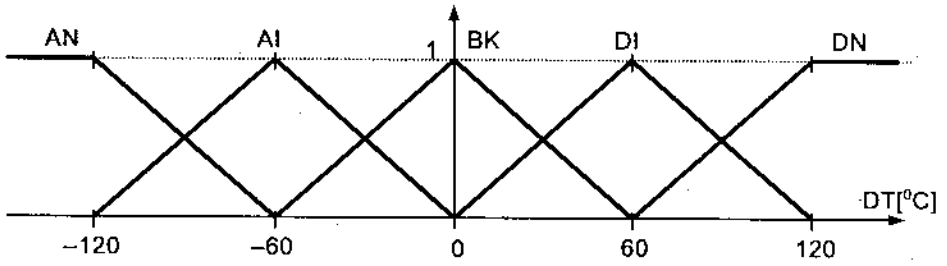
$$DT = T_{current} - T_{set} \quad (3.2)$$

trong đó $T_{current}$ là nhiệt độ thực tế đo được ở lò nhiệt (từ ADC gửi đến) và T_{set} là nhiệt độ do người sử dụng đặt (đặt bởi các phím bấm hoặc bởi phần mềm giám sát trên PC).

+ Đầu ra tỷ số độ rộng xung μ là tỷ số độ rộng xung của xung điều khiển, được dùng để đưa tới mạch công suất điều khiển việc cấp nguồn cho lò nhiệt.

Trong ví dụ này, việc mờ hoá đầu vào DT và đầu ra μ được thực hiện như sau:

+ Đầu vào sai lệch nhiệt độ DT được xác định thông qua 5 giá trị ngôn ngữ là: âm nhiều, âm ít, bằng không, dương ít và dương nhiều. Miền giá trị vật lý của các biến ngôn ngữ này là từ -128^0C đến $+128^0$C. Mỗi giá trị ngôn ngữ này được mô tả bằng một tập mờ có hàm thuộc được chọn như trên hình 3.29.



Hình 3.29. Mô tả các giá trị ngôn ngữ của DT bằng các tập mờ.

Trong đó:

AN – âm nhiều;

AI – âm ít;

BK – không;

DI – dương ít;

DN – dương nhiều.

Quá trình mờ hoá đầu vào DT là ánh xạ từ giá trị rõ DT thành véctơ $\underline{\mu}$

$$DT \mapsto \underline{\mu} = \begin{bmatrix} \mu_{AN}(DT) \\ \mu_{AI}(DT) \\ \mu_{BK}(DT) \\ \mu_{DI}(DT) \\ \mu_{DN}(DT) \end{bmatrix}$$

Với các giá trị $\mu_{AN}(DT)$, $\mu_{AI}(DT)$, $\mu_{BK}(DT)$, $\mu_{DI}(DT)$, $\mu_{DN}(DT)$ được xác định theo các công thức (3.3), (3.4), (3.5), (3.6), (3.7).

$$\mu_{AN}(DT) = \begin{cases} \dfrac{-DT-60}{60} & \text{nếu } -120 < DT < -60 \\ 1 & \text{nếu } DT < -120 \end{cases} \quad (3.3)$$

$$\mu_{AI}(DT) = \begin{cases} \dfrac{DT+120}{60} & \text{nếu } -120 < DT < -60 \\ \dfrac{-DT}{60} & \text{nếu } -60 < DT < 0 \\ 0 & \text{nếu } DT < -120 \text{ hoặc } > 0 \end{cases} \quad (3.4)$$

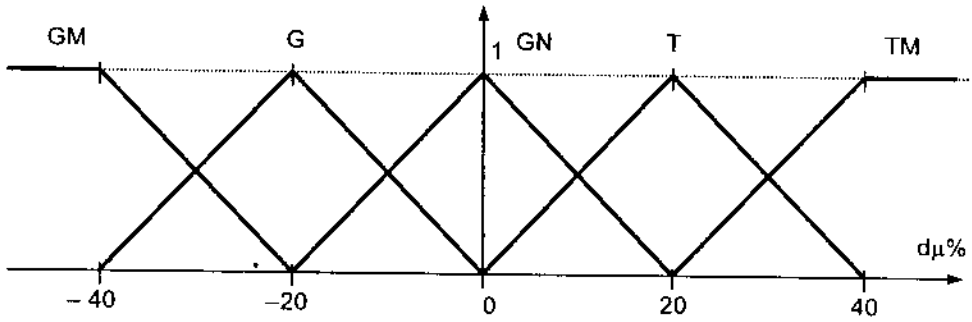
$$\mu_{BK}(DT) = \begin{cases} \dfrac{DT+60}{60} & \text{nếu } -60 < DT < 0 \\ \dfrac{60-DT}{60} & \text{nếu } 0 < DT < 60 \\ 0 & \text{nếu } DT < -60 \text{ hoặc } > 60 \end{cases} \quad (3.5)$$

$$\mu_{DI}(DT) = \begin{cases} \dfrac{DT}{60} & \text{nếu } 0 < DT < 60 \\ \dfrac{120-DT}{60} & \text{nếu } 60 < DT < 120 \\ 0 & \text{nếu } DT < 0 \text{ hoặc } > 120 \end{cases} \quad (3.6)$$

$$\mu_{DN}(DT) = \begin{cases} \dfrac{DT-60}{60} & \text{nếu } 60 < DT < 120 \\ 1 & \text{nếu } DT > 120 \end{cases} \qquad (3.7)$$

Trong chương trình điều khiển, việc mờ hoá giá trị rõ DT được thực hiện thông qua năm hàm dựa theo năm công thức (3.3), (3.4), (3.5), (3.6), (3.7); các tập mờ này được rời rạc hoá tại 256 điểm ứng với các giá trị nguyên DT từ -127^0C đến 128^0C.

+ Đầu ra tỷ số độ rộng xung μ được xác định bằng cách cộng thêm một lượng dμ vào giá trị hiện tại của nó, trong đó giá trị ban đầu mặc định của μ được chọn là 50%. dμ được xác định thông qua 5 giá trị ngôn ngữ là: giảm mạnh, giảm, giữ nguyên, tăng và tăng nhiều. Miền giá trị vật lý của các biến ngôn ngữ này là từ –50% đến +50%. Mỗi giá trị ngôn ngữ này được mô tả bằng một tập mờ có hàm thuộc được chọn như trên hình 3.30.



Hình 3.30. Mô tả các giá trị ngôn ngữ của dμ bằng các tập mờ.

Trong đó:

GM – giảm mạnh;
G – giảm;
GN – giữ nguyên;
T – tăng;
TM – tăng mạnh.

Quá trình mờ hoá đầu ra dμ là ánh xạ từ giá trị rõ dμ thành véctơ $\underline{\mu}$:

$$d\mu \mapsto \underline{\mu} = \begin{bmatrix} \mu_{GM}(d\mu) \\ \mu_{G}(d\mu) \\ \mu_{GN}(d\mu) \\ \mu_{T}(d\mu) \\ \mu_{TM}(d\mu) \end{bmatrix} \qquad (3.8)$$

Với các giá trị $\mu_{GM}(d\mu)$, $\mu_G(d\mu)$, $\mu_{GN}(d\mu)$, $\mu_T(d\mu)$, $\mu_{TM}(d\mu)$ được xác định theo các công thức (3.9), (3.10), (3.11), (3.12), (3.13).

$$\mu_{GM}(d\mu) = \begin{cases} 1 & \text{nếu } d < -40 \\ \dfrac{-20-dm}{20} & \text{nếu } -40 < d < -20 \\ 0 & \text{nếu } d > -20 \end{cases} \quad (3.9)$$

$$\mu_G(d\mu) = \begin{cases} \dfrac{dm+40}{20} & \text{nếu } -40 < d < -20 \\ \dfrac{-dm}{20} & \text{nếu } -20 < d < 0 \\ 0 & \text{nếu } d < -40 \text{ hoặc } > 0 \end{cases} \quad (3.10)$$

$$\mu_{GN}(d\mu) = \begin{cases} \dfrac{dm+20}{20} & \text{nếu } -20 < d < 0 \\ \dfrac{20-dm}{20} & \text{nếu } 0 < d < 20 \\ 0 & \text{nếu } d < -20 \text{ hoặc } > 20 \end{cases} \quad (3.11)$$

$$\mu_T(d\mu) = \begin{cases} \dfrac{dm}{20} & \text{nếu } 0 < d < 20 \\ \dfrac{40-dm}{20} & \text{nếu } 20 < d < 40 \\ 0 & \text{nếu } d < 0 \text{ hoặc } > 40 \end{cases} \quad (3.12)$$

$$\mu_{TM}(d\mu) = \begin{cases} \dfrac{dm-20}{20} & \text{nếu } 20 < d < 40 \\ 1 & \text{nếu } d > 40 \end{cases} \quad (3.13)$$

Với dμ nguyên từ –50 đến +50, các hàm thuộc này được lưu ở bộ nhớ RAM ngoài dưới dạng các mảng một chiều 101 phần tử mà giá trị mỗi phần tử được khởi tạo dựa theo các công thức từ (3.9) đến (3.13).

$$GM = \{\mu_{GM}(-50); \mu_{GM}(-49); ...; \mu_{GM}(50)\}$$

$$G = \{\mu_G(-50); \mu_G(-49); ...; \mu_G(50)\}$$

$$GN = \{\mu_{GN}(-50); \mu_{GN}(-49);...; \mu_{GN}(50)\}$$

$$T = \{\mu_{T}(-50); \mu_{T}(-49);...; \mu_{T}(50)\}$$

$$TM = \{\mu_{TM}(-50); \mu_{TM}(-49);...; \mu_{TM}(50)\}$$

*** Hợp thành (suy diễn và hợp mờ)**

– Xây dựng các luật điều khiển:

Với việc mờ hoá đầu ra và đầu vào của bộ điều khiển như trên, ta đặt ra cho bộ điều khiển 5 luật điều khiển sau:

R1: Nếu DT = AN thì dμ = TM

R2: Nếu DT = AI thì dμ = T

R3: Nếu DT = BK thì dμ = GN

R4: Nếu DT = DI thì dμ = G

R5: Nếu DT = DN thì dμ = GM

Gọi H1 là độ thuộc của DT vào AN, có nghĩa là $H1 = \mu_{AN}(DT)$

H2 là độ thuộc của DT vào AI, có nghĩa là $H2 = \mu_{AI}(DT)$

H3 là độ thuộc của DT vào BK, có nghĩa là $H3 = \mu_{BK}(DT)$

H4 là độ thuộc của DT vào DI, có nghĩa là $H4 = \mu_{DI}(DT)$

H5 là độ thuộc của DT vào DN, có nghĩa là $H5 = \mu_{DN}(DT)$

Chọn luật hợp thành là luật max–min thì ta có:

+ Đầu ra của luật điều khiển R1:

$$R1 = \{\min(H1; \mu_{TM}(-50)); \min(H1; \mu_{TM}(-49));....; \min(H1; \mu_{TM}(50))\}$$

hay $R1[i] = \min(H1; TM[i])$ với $i \in [0,100]$ (3.14)

+ Đầu ra của luật điều khiển R2:

$$R2 = \{\min(H2; \mu_{T}(-50)); \min(H2; \mu_{T}(-49));....; \min(H2; \mu_{T}(50))\}$$

hay $R2[i] = \min(H2; T[i])$ với $i \in [0,100]$ (3.15)

+ Đầu ra của luật điều khiển R3:

$$R3 = \{\min(H3; \mu_{GN}(-50)); \min(H3; \mu_{GN}(-49));....; \min(H3; \mu_{GN}(50))\}$$

hay $R3[i] = \min(H3; GN[i])$ với $i \in [0,100]$ (3.16)

+ Đầu ra của luật điều khiển R4:

$$R4 = \{\min(H4; \mu_G(-50)); \min(H4; \mu_G(-49));....; \min(H4; \mu_G(50))\}$$

hay $R4[i] = \min(H4;G[i])$ với $i \in [0,100]$ (3.17)

+ Đầu ra của luật điều khiển R5:

$$R5 = \{\min(H5; \mu_{GM}(-50)); \min(H5; \mu_{GM}(-49));....; \min(H5; \mu_{GM}(50))\}$$

hay $R1[i] = \min(H5; GM[i])$ với $i \in [0,100]$ (3.18)

Trong chương trình điều khiển sau khi tính các độ thuộc H1, H2, H3, H4, H5 của DT, dựa vào các công thức (3.14), (3.15), (3.16), (3.17) và (3.18) ta hoàn toàn có thể thiết lập được giá trị cho các phần tử của các mảng R1, R2, R3, R4 và R5.

– Hợp mờ:

Chọn luật hợp thành là luật max – min, ta có đầu ra R của thiết bị hợp thành được xác định theo công thức sau:

$R[i] = \max\{ R1[i], R2[i], R3[i], R4[i], R5[i]\}$ với $i \in [0,100]$ (3.19)

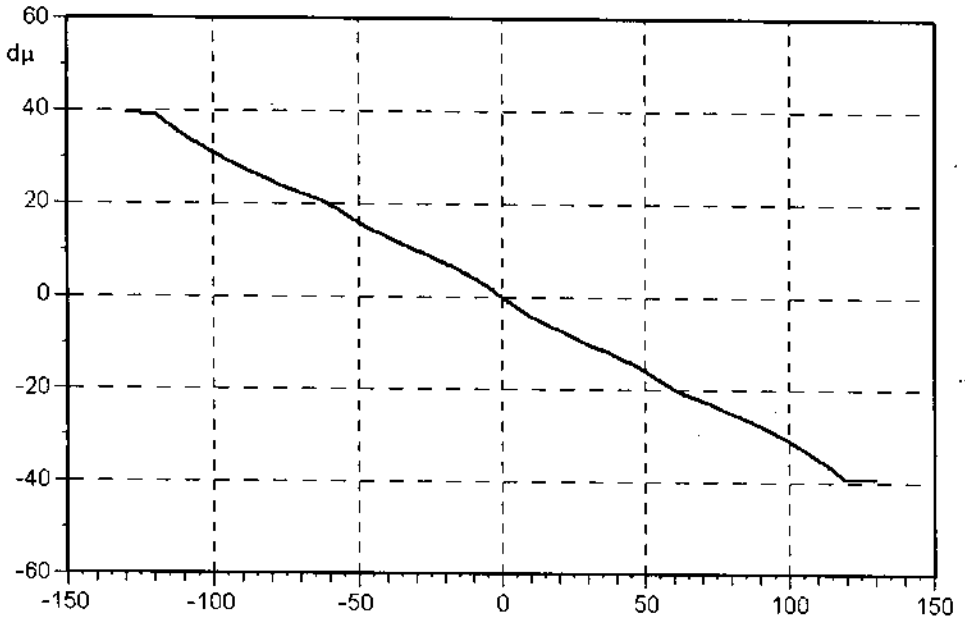
*** Giải mờ**

Để thực hiện giải mờ ta chọn phương pháp giải mờ thông dụng là phương pháp điểm trọng tâm. Vì miền giá trị của dμ là miền rời rạc nên ta có thể thay phép tính tích phân như trường hợp miền giá trị liên tục bằng phép cộng. Kết quả ta thu được giá trị rõ của dμ là:

$$d\mu = \frac{\sum_{i=0}^{100} i.R[i]}{\sum_{i=0}^{100} R[i]} \quad (3.20)$$

Giá trị rõ dμ này được sử dụng để tạo ra xung vuông có tỷ số độ rộng đúng bằng μ + dμ (ban đầu μ được gán một giá trị mặc định là 50%, sau mỗi vòng lặp nó được cộng thêm dμ) ở một chân I/O của vi điều khiển, đó chính là đầu ra PWM của bộ điều khiển.

Với phương thức mờ hoá, suy diễn và giải mờ như đã trình bày, dùng phần mềm FuzzyTech để mô phỏng bộ điều khiển mờ ta thu được đặc tuyến quan hệ vào – ra trong hình 3.31.



Hình 3.31. Đặc tuyến truyền đạt của bộ điều khiển.

Sau đây là chương trình cụ thể được viết để thực hiện toàn bộ các công việc nêu trên:

```
#include<stdio.h>
#include<reg51.h>
#include<string.h>
#include<math.h>
#include<INTRINS.H>
unsigned int x = 0xFF;
unsigned char SetTemp = 30,CurrentTemp;
int dt,dm,muy = 50;
xdata unsigned char CREG _at_ 0x2003;
xdata unsigned char DATA _at_ 0x2000;/* cổng A của 8255
nối với bus dữ liệu của LCD*/
xdata unsigned char PORTB _at_ 0x2001;
xdata unsigned char PORTC _at_ 0x2002;
unsigned char controllcd;
xdata unsigned char ADC _at_ 0x4000;
void LCDSetup(void);
void ClearDisplay(void);
void ExeLCD(void);
void gotoxy(unsigned char x,unsigned char y);
void BeginOne(void);
void BeginTwo(void);
sbit PWM = 0xB4;
```

```
sbit ADC_OK = 0xB3;
unsigned int
khong[101],nho[101],trung_binh[101],lon[101],rat_lon[101];
unsigned int
R1[101],R2[101],R3[101],R4[101],R5[101],R[101];
int count = 0;
bit pwmflag;
unsigned int dCycle = 0,dCycleC = 0xFFFF;
void delay(long int Wait);
char putchar(char c);
char sendchar(char c);
void setting(void);
int am_nhieu(int error);
int am_it(int error);
int bang_khong(int error);
int duong_it(int error);
int duong_nhieu(int error);
void inference(void);
void refuzzy(void);
int min(int x0,int y0);
int max2(int x0,int y0);
int max5(int x1,int x2,int x3,int x4,int x5);
unsigned char getadc(void);
void request(void);
float dtemp;
void main(void)
{
setting();
LCDSetup();
while(1)
{
CurrentTemp = getadc();
dt = SetTemp-CurrentTemp;
inference();
refuzzy();
muy = muy + dm;
if(muy>99)
{
muy = 99;
}
if(muy<1)
{
muy = 1;
}
dtemp = ((float) (muy*65535))/100;
dCycle = (unsigned int)(dtemp);
dCycleC = ~dCycle;
ClearDisplay();
BeginOne();
printf("Set:%bu C\n",SetTemp);
```

```
printf("Current:%bu C",CurrentTemp);
sendchar(CurrentTemp);//gui nhiet do len PC
request();
delay(20000);
}
}
unsigned char getadc(void)
{
      unsigned char temp;
      ADC = 0xFF;
      while(!ADC_OK);
      while(ADC_OK);
      temp = ADC;
      return(temp);
}
// mờ hóa đầu vào
int am_nhieu(int error)
{
           if(error<= -120)
           return(100);
      if((error>-120)&&(error<-60))
           return((int)(5*(-60-error)/3));
      if(count>= -60)
           return(0);
}
int am_it(int error)
{
      if(error<= -120)
           return(0);
      if((error>-120)&&(error<=-60))
           return((int)(5*(error+120)/3));
      if((error>-60)&&(error<0))
           return((int)(5*(-error)/3));
      if(count>= 0)
           return(0);
}
int bang_khong(int error)
{
      if(error<=-60)
           return(0);
      if((error>-60)&&(error<= 0))
           return((int)(5*(error+60)/3));
      if((error>0)&&(error<60))
           return((int)(5*(60-error)/3));
      if(error>= 60)
           return(0);
}
int duong_it(int error)
{
if(error<=0)
```

```
return(0);
if((error>0)&&(error<=60))
return((int)(5*(error)/3));
if((error>60)&&(error<120))
return((int)(5*(120-error)/3));
if(error>=120)
return(0);
}
int duong_nhieu(int error)
{
if(error<=60)
return(0);
if((error>60)&&(error<120))
return((int)(5*(error-60)/3));
if(count>=120)
return(100);
}
void inference(void)
{
int temp1,temp2,temp3,temp4,temp5;
// suy diễn
      emp1 = am_nhieu(dt);
      emp2 = am_it(dt);
      emp3 = bang_khong(dt);
      emp4 = duong_it(dt);
      emp5 = duong_nhieu(dt);
      for(count = 0;count<100;count++)
      {
            R1[count] = min(temp1,rat_lon[count]);
            R2[count] = min(temp2,lon[count]);
            R3[count] = min(temp3,trung_binh[count]);
            R4[count] = min(temp4,nho[count]);
            R5[count] = min(temp5,khong[count]);
      }
// hợp mờ
      for(count = 0;count<100;count++)
      {
      R[count]= max5(R1[count], R2[count], R3[count],
R4[count], R5[count]);
      }
}

// giải mờ
void refuzzy(void)
{
      long int temp1 = 0,temp2 = 0;
      for(count = 0;count<100;count++)
      {
            temp1+ = count*R[count];
            temp2+ = R[count];
```

```
        }
        dm =(int) ((float) temp1/temp2 - 50);
}
void delay(long int Wait)
{
unsigned int Count;
for (Count = 0;Count < Wait;Count++);
}

int max2(int x0,int y0)
{
        if(x0 < y0) return(y0);
        if(x0 > = y0) return(x0);
}
int max5(int x1,int x2,int x3,int x4,int x5)
{
        int temp1,temp2,temp3;
        temp1 = max2(x1,x2);
        temp2 = max2(x3,x4);
        temp3 = max2(temp1,temp2);
        return(max2(temp3,x5));
}
int min(int x0,int y0)
{
        if(x0<y0) return(x0);
        if(x0> = y0) return(y0);
}

// tạo xung PWM
void timer0 (void) interrupt 1 using 1
{
if (pwmflag == 0)
{
        PWM = 1;
        TH0 = dCycle/256;
        TL0 = dCycle%256;
        pwmflag = 1;
}
else
{
        PWM = 0;
        TH0 = dCycleC/256;
        TL0 = dCycleC%256;
        pwmflag = 0;
}
}
void Ext0 (void) interrupt 0 using 3
{
unsigned char temp;
EA = PWM = 0;
```

```
//nhap so lieu tu ban phim
ClearDisplay();
BeginOne();
printf("Set:%bu C\n",SetTemp);
printf("Set Up Down OK");
do
{
temp = P1 & 0x07;
delay(3000);
if(temp == 0x06)
{
if(SetTemp < 130)
{
SetTemp++;
}
// khống chế SetTemp không vượt quá 130
if(SetTemp>130)
{
SetTemp = 130;
}
ClearDisplay();
BeginOne();
printf("Set:%bu C\n",SetTemp);
printf("Set Up Down OK");
}
if(temp == 0x05)
{
if(SetTemp > 30)
{
SetTemp--;
}
// khống chế SetTemp không nhỏ hơn 30
if(SetTemp < 30)
{
SetTemp = 30;
}
ClearDisplay();
BeginOne();
printf("Set:%bu C\n",SetTemp);
printf("Set Up Down OK");
}
}
while(temp! = 0x03);
EA = 1;
}
void request(void)
{
if(RI == 1)
{
      RI = 0;
```

```
      SetTemp = SBUF;
}
}
void serialport(void) interrupt 4 using 2
{
//nhận giá trị SetTemp từ PC
if(RI == 1)
{
      RI = 0;
      SetTemp = SBUF;
}
else
{
TI = 0;
SBUF = CurrentTemp;
}
}
void setting(void)
{
      PWM = 0;// xóa dầu ra điều khiển rơle
      SCON = 0x52;//khởi tạo cổng nối tiếp
      TMOD = 0x21;
      TH1 = TL1 = -3;
      TR1 = 1;
      EA = ET0 = PT0 = EX0 = 1;
      TH0 = 0;
      TL0 = 0xFF;
      TR0 = 1;
      CREG = 0x80;

//mờ hoá đầu ra
for(count = 0;count < 100;count++)
      {
      if(count < = 10)
      {
            khong[count] = 100;
            nhỏ[count] = trung_binh[count] = lon[count] =
rat_lon[count] = 0;
      }
            if((count > 10)&&(count < 30))
      {
            khong[count]= 5*(30-count);
            nho[count]= 5*(count-10);
trung_binh[count]= lon[count]=rat_lon[count]=0;
      }
      if((count> = 30)&&(count<50))
      {
            khong[count] = 0;
            nho[count] = 5*(50-count);
            trung_binh[count] = 5*(count -30);
```

```
            lon[count] = rat_lon[count] = 0;
        }
        if((count > = 50)&&(count < 70))
        {
            khong[count] = 0;
            nho[count] = 0;
            trung_binh[count]= 5*(70-count);
            lon[count]=5*(count-50);
            rat_lon[count]=0;
        }
        if((count > = 70)&&(count < 90))
        {
            khong[count] = 0;
            nho[count] = 0;
            trung_binh[count] = 0;
            lon[count] = 5*(90-count);
            rat_lon[count] = 5*(count-70);
        }
        if(count> = 90)
        {
            khong[count] = 0;
            nho[count] = 0;
            trung_binh[count] = 0;
            lon[count] = 0;
            rat_lon[count] = 100;
        }
}

}
void ExeLCD(void)
{
// xóa RS = 0;
controllcd = controllcd & 0xFE;
PORTB = controllcd;
// thiết lập ENABLE = 1;
controllcd = controllcd | 0x04;
PORTB = controllcd;
delay(30);
// xóa ENABLE = 0;
controllcd = controllcd & 0xFB;
PORTB = controllcd;
// thiết lập RS = 1;
controllcd = controllcd | 0x01;
PORTB = controllcd;
}
// đua con trỏ về đầu dòng 1
void BeginOne(void)
{
DATA = 0x80;
ExeLCD();
```

```
}
// đưa con trỏ về đầu dòng 2
void BeginTwo(void)
{
DATA = 0xC0;
ExeLCD();
}
// khởi tạo LCD
void LCDSetup(void)
{
CREG = 0x80;
PORTB = DATA = controllcd = 0xFF;
controllcd = controllcd & 0xFD;
PORTB = controllcd;
// RW = 0;
DATA = 0x3C;
ExeLCD();
DATA = 0x0D;
ExeLCD();
DATA = 0x06;
ExeLCD();
DATA = 0x02;
ExeLCD();
}
// xóa màn hình, đưa con trỏ về đầu dòng 1
void ClearDisplay(void)
{
DATA = 0x01;
ExeLCD();
BeginOne();
}
char putchar (char c)
{
 if (c ! = '\n')
 {
 DATA = c;
// thiết lập ENABLE=1;
controllcd = controllcd | 0x04;
PORTB = controllcd;
 _nop_();
 _nop_();
// xóa ENABLE;
controllcd = controllcd & 0xFB;
PORTB = controllcd;
 }
 if (c == '\n')
 {
  BeginTwo();
 }
return(1);
```

```
}
// gửi ký tự c ra cổng UART
char sendchar(char c)
{
while(!TI);// chờ TI=1
TI = 0; // xóa TI
SBUF = c;
return(c);
}
```

3.2.15. Thiết kế bộ mô hình Robot chạy bám đường (Line Tracking Robot)

1. Phân tích bài toán

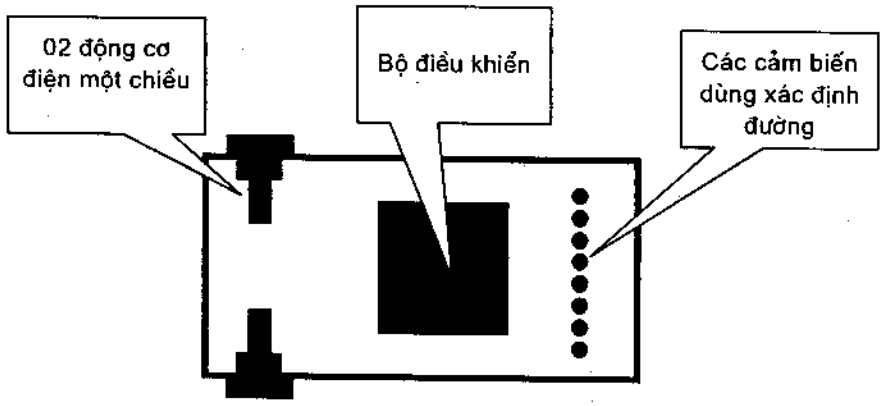
Kỹ thuật dò đường (Line Tracking) là một trong những kỹ thuật được áp dụng rộng rãi nhất trong các cuộc thi sáng tạo Robot (Robocon) dành cho sinh viên các trường kỹ thuật. Kỹ thuật này giúp Robot có thể di chuyển theo các đường có sẵn trên sân với độ tin cậy khá cao. Mô hình Robot chạy bám đường gồm các phần chính sau (hình 3.32):

– 02 động cơ DC có công suất thích hợp;

– 02 bộ điều khiển các động cơ DC (Motor Driver) có chức năng đảo chiều, tăng giảm tốc độ bằng kỹ thuật PWM;

– Từ 4 đến 8 cảm biến quang cho phép nhận biết đường đi (line) và nền sân (background);

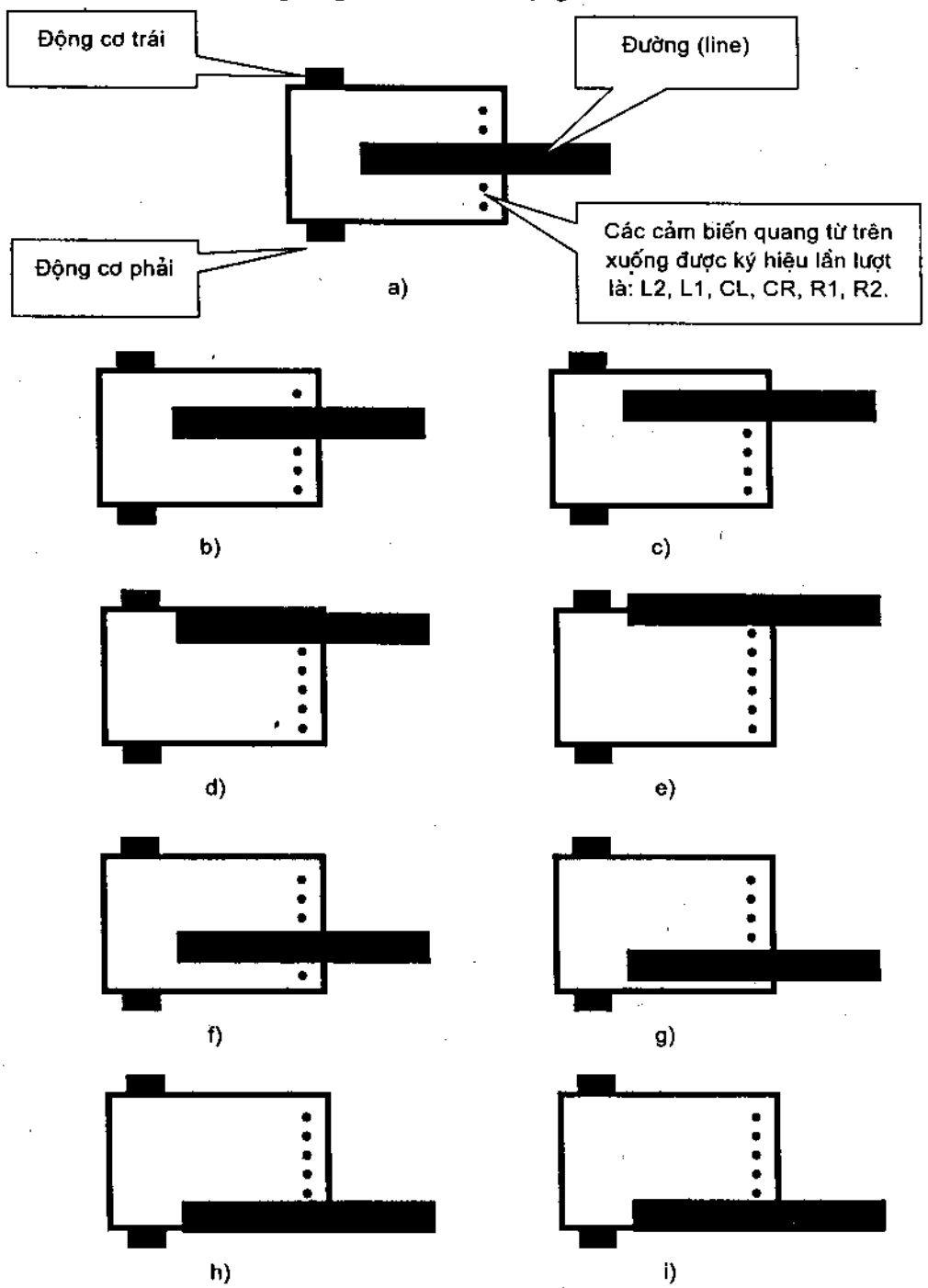
– 01 bộ vi điều khiển để thực hiện các thuật toán dò đường.



Hình 3.32. Mô hình Robot chạy bám đường.

Thuật toán dò đường dựa trên trạng thái của các cảm biến quang để ra quyết định điều khiển 02 động cơ sao cho Robot luôn chạy trên đường (hình 3.33).

Với 06 cảm biến quang thì sẽ có 09 trạng thái của Robot:



Hình 3.33. Các trạng thái của Robot khi chạy trên đường.

– Trạng thái 1 (hình 3.33a): Robot đang chạy đúng đường, trong trường hợp này ta điều khiển để 2 động cơ chạy cùng một tốc độ.

– Trạng thái 2 (hình 3.33b): Robot chạy hơi lệch sang trái, trong trường hợp này ta điều khiển để động cơ phải chạy nhanh hơn động cơ trái một chút (chẳng hạn độ rộng xung của động cơ phải là 100% còn động cơ trái là 90%).

– Trạng thái 3 (hình 3.33c): Robot chạy lệch sang trái, trong trường hợp này ta đã điều khiển để động cơ phải chạy nhanh hơn động cơ trái (chẳng hạn độ rộng xung của động cơ phải là 100% còn động cơ trái là 70%).

– Trạng thái 4 (hình 3.33d): Robot chạy lệch nhiều sang trái, trong trường hợp này ta đã điều khiển để động cơ phải chạy nhanh hơn nhiều so với động cơ trái (chẳng hạn độ rộng xung của động cơ phải là 100% còn động cơ trái là 50%).

– Trạng thái 5 (hình 3.33e): Robot chạy lệch hẳn sang trái, trong trường hợp này để robot có thể quay sang trái nhanh và bám vào đường có thể cho dừng động cơ trái trong khi vẫn động cơ phải quay với tốc độ vừa phải (chẳng hạn 50%) hoặc cho động cơ phải quay tiến còn động cơ trái quay lùi tuỳ thuộc vào đặc tính cơ học của Robot tương ứng.

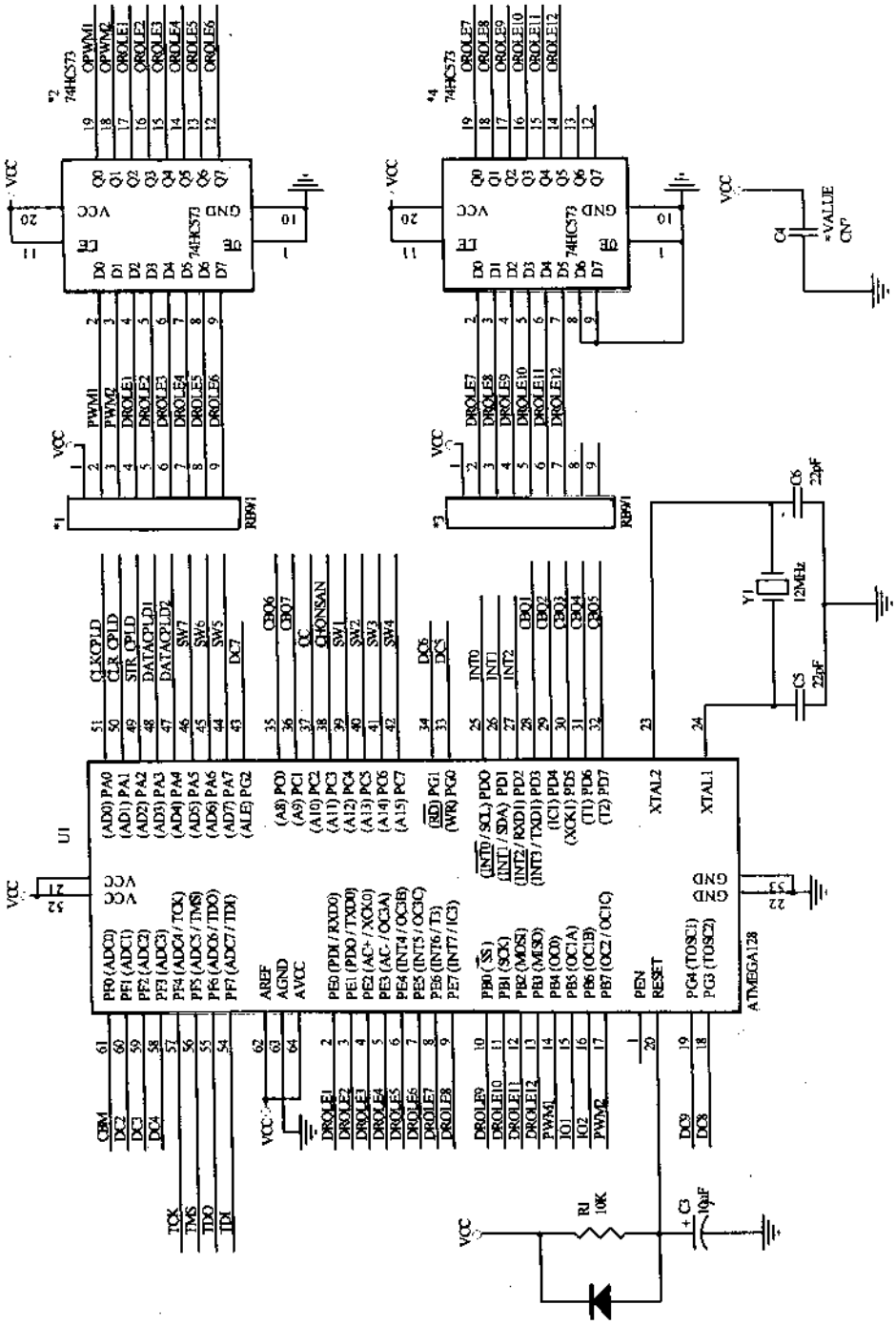
Các trạng thái 6, 7, 8, 9 là các trạng thái Robot lệch sang phải (hình 3.33f, g, h, i), giải thuật điều khiển tương tự.

2. Sơ đồ nguyên lý mạch điều khiển

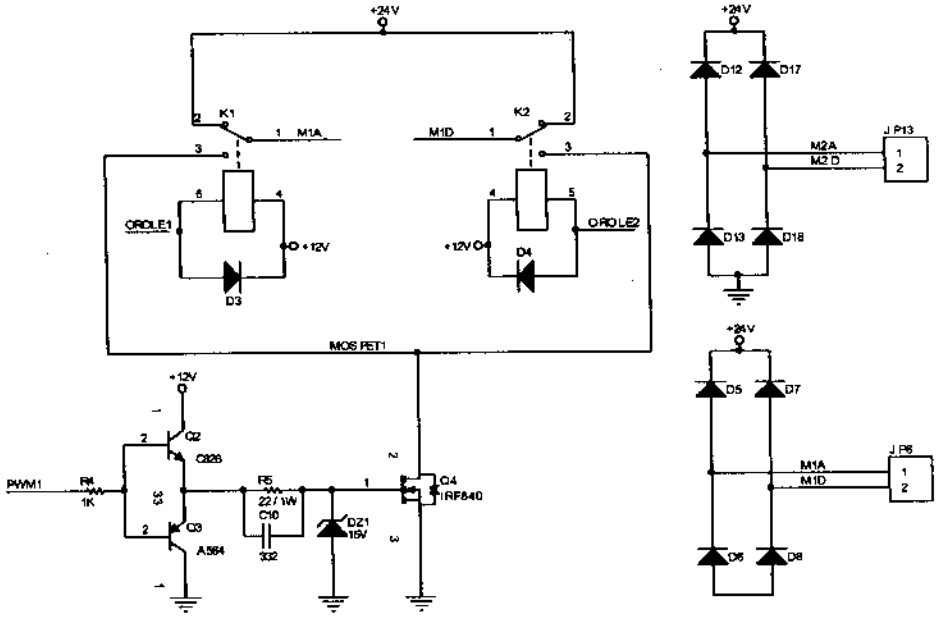
Hình 3.34 là mạch điều khiển trung tâm. Đối với các ứng dụng có sự tham gia của các hệ điện cơ thì dòng vi điều khiển AVR tỏ ra có ưu thế hơn vi điều khiển 8051 nhờ khả năng chống nhiễu tốt. Mạch điện hình 3.34 sử dụng bộ vi điều khiển Atmega128L – bộ vi điều khiển có 6 cổng vào/ra, dung lượng bộ nhớ Flash lên tới 128kB – rất thích hợp khi sử dụng cho Robot.

Hình 3.35 là mạch điều khiển động cơ. Trong mạch này động cơ được đảo chiều quay bằng cách dùng 02 Relay 12VDC để đảo cực tính nguồn một chiều cung cấp cho 2 động cơ, động cơ được tăng/giảm tốc độ bằng cách tăng/giảm tỷ số độ rộng xung đưa đến đèn công suất IRF840.

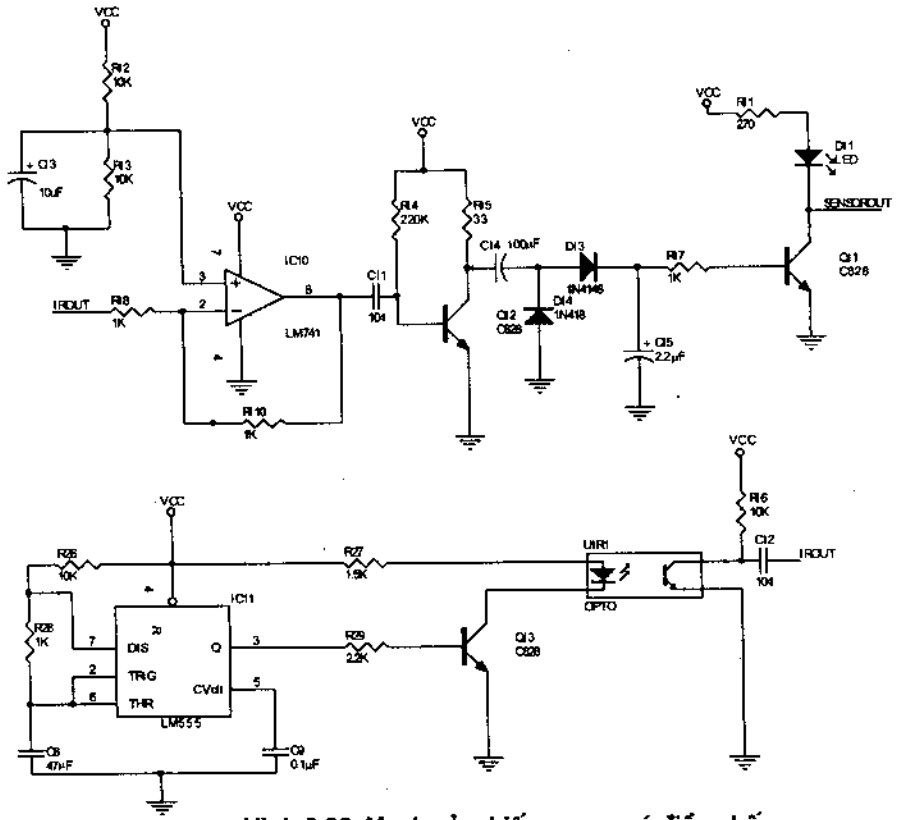
Hình 3.36 là mạch cảm biến quang có điều chế để có thể hạn chế được nhiễu do ánh sáng lạ chiếu vào phototransistor và nâng cao độ nhạy của cảm biến. Với 6 cảm biến quang L2, L1, CL, CR, R1, R2 ta phải lắp ráp 6 mạch điện như sơ đồ này.



Hình 3.34. Bộ điều khiển trung tâm.



Hình 3.35. Mạch điều khiển động cơ.



Hình 3.36. Mạch cảm biến quang có điều chế.

2. *Chương trình điều khiển*

Chương trình 1: Điều khiển robot chạy bám đường

```
#include <mega128.h>
#include <delay.h>
#include <math.h>
//định nghĩa các đầu vào của các cảm biến quang
#define    LL2     PINC.6
#define    LL1     PINC.5
#define    CL      PINC.4
#define    CR      PINC.3
#define    RR1     PINC.2
#define    RR2     PINC.1
//định nghĩa các chân điều khiển 2 động cơ
#define DROLE1   PORTE.0////M////
#define DROLE2   PORTE.1////S////
#define DROLE3   PORTE.2////A////
#define DROLE4   PORTE.3////U////

//định nghĩa các cấp tốc độ
//tốc độ của động cơ bên phải
#define   R11     0
#define    R12     32 /* tỷ số độ rộng xung là 32/256 = 12,5% */
#define   R13     64
#define   R14     128
#define   R15     256
//*******************************
//tốc độ của động cơ bên trái
#define   L11     0
#define   L12     32
#define   L13     64
#define   L14     128
#define   L15     256
// định nghĩa label
#define   go      0
#define   back    1

//định nghĩa các trạng thái của robot
/* 1 là cảm biến tương ứng ở trên vạch; 0 là ở ngoài vạch */
```

```
#define   L4 0B00000000
#define   L3 0B01000000
#define   L2 0B01100000
#define   L1 0B00110000 //tương ứng hình 3.33.b
#define   TT 0B00011000 //tương ứng hình 3.33.a
#define   R1 0B00001100 //tương ứng hình 3.33.f
#define   R2 0B00000110
#define   R3 0B00000010
#define   R4 0B00000000

//************************************
//chương trình con điều khiển động cơ bên trái
void runL (unsigned char power,unsigned char dir)
{
OCR0 = power; // biến power chứa tỷ số độ rộng xung PWM
if(dir == go) //chiều tiến
{
DROLE1 = 1;
DROLE2 = 0;
}
else if (dir == back) //chiều lùi
 {
DROLE1 = 0;
DROLE2 = 1;
 }
}
//************************************
//chương trình con dừng (khoá) động cơ trái
void stopL(void)
{
DROLE1 = 1;
DROLE2 = 1;
}
//************************************
//chương trình con điều khiển động cơ bên trái
void runR(unsigned char power,unsigned char dir)
{
OCR2 = power;
if(dir == go)
{
DROLE3 = 1;
```

```
DROLE4 = 0;
}
else if (dir == back)
 {
DROLE3 = 0;
DROLE4 = 1;
 }
}
//************************************
//chương trình con dừng (khoá) động cơ phải
void stop_R(void)
{
DROLE3 = 1;
DROLE4 = 1;
}
//************************************
//chương trình con dừng toàn bộ robot
void stop(void)
{
OCR0 = OCR2 = 0;
stop_L();
stop_R();
}
//************************************
//chương trình con điều khiển chạy bám đường
void tracking (void)
{
unsigned char temp1,temp2,INPUT;
INPUT = PINC; //đọc các cảm biến
INPUT = INPUT&0x81; //bỏ đi 2 chân không dùng
switch (INPUT)
{
 case TT: //robot chạy thẳng
 {
 runR(R14,go);
 runL(L14,go);
 temp1 = 0;
 temp2 = 0;
 break;
 }
 case L1: //robot chạy hơi lệch sang trái
 {
```

```
runR(R14,go);
runL(L13,go);
temp1 = 0;
temp2 = 0;
 break;
}
case L2: //robot chạy lệch sang trái
{
runR(R14,go);
runL(L12,go);
temp1 = 1;
temp2 = 0;
break;
}
case L3: //robot chạy lệch nhiều sang trái
{
runR(R14,go);
runL(L11,go);
temp1 = 0;//nhớ trạng thái lệch sang trái
temp2 = 0;
break;
}
case R1:  //robot chạy hơi lệch sang phải
{
runR(R13,go);
runL(L14,go);
temp1 = 0;
temp2 = 0;
break;
}
case R2:  //robot chạy lệch sang phải
{
runR(R12,go);
runL(L14,go);
temp1 = 0;
temp2 = 0;
break;
}
case R3: //robot chạy lệch nhiều sang phải
{
runR(R11,go);
runL(L14,go);
```

```
temp1 = 0;
temp2 = 1;  //nhớ trạng thái lệch phải
break;
}
//nếu lệch hẳn sang trái hoặc sang phải
case (L4|R4):
{
if (temp1 == 1)    //lệch trái
 {
 runR(R12,go);
 runL(L12,back);
 }
if (temp2 == 1)    //lệch phải
 {
 runR(R12,back);
 runL(L12,go);
 }
break;
 }
}
}
}
//chương trình chính
void main (void)
{
//khởi tạo chiều vào/ra của các port
PORTA = 0xFF;
DDRA = 0x00;
PORTC = 0xFF;
DDRC = 0x00;
//khởi tạo timer 0 ở chế độ PWM
ASSR = 0x00;
TCCR0 = 0x6C;
TCNT0 = 0x00;
OCR0 = 0x00;
//khởi tạo timer 2 ở chế độ PWM
TCCR2 = 0x6B;
TCNT2 = 0x00;
OCR2 = 0x00;
while(1)
tracking();
}
```

Chương trình 2: Điều khiển robot chạy bám đường với các tốc độ, các đoạn đường khác nhau.

```
#include <mega128.h>
#include <delay.h>
#include <math.h>
//định nghĩa các đầu vào của các cảm biến quang
#define   LL2       PINC.6
#define   LL1       PINC.5
#define   CL        PINC.4
#define   CR        PINC.3
#define   RR1       PINC.2
#define   RR2       PINC.1
// định nghĩa các chân điều khiển 2 động cơ
#define   DROLE1    PORTE.0////M////
#define   DROLE2    PORTE.1////S////
#define   DROLE3    PORTE.2////A////
#define   DROLE4    PORTE.3////U////

//định nghĩa các cấp tốc độ
//cấp 1, tốc độ lớn nhất
//tốc độ của động cơ bên phải
#define   R11       0
#define   R12       50
#define   R13       150
#define   R14       200
#define   R15       256
//tốc độ của động cơ bên trái
#define   L11       0
#define   L12       50
#define   L13       150
#define   L14       200
#define   L15       256
//*******************************
//cấp 2, tốc độ trung bình
//tốc độ của động cơ bên phải
#define   R21       0
#define   R22       50
#define   R23       100
#define   R24       140
#define   R25       180
//tốc độ của động cơ bên trái
#define   L21       0
```

```
#define   L22     50
#define   L23     100
#define   L24     140
#define   L25     180
//************************************
//cấp 3, tốc độ nhỏ nhất
//tốc độ của động cơ bên phải
#define   R31     0
#define   R32     20
#define   R33     50
#define   R34     80
#define   R35     120
//tốc độ của động cơ bên trái
#define   L31     0
#define   L32     20
#define   L33     50
#define   L34     80
#define   L35     120

//định nghĩa label
#define   go      0
#define   back    1

//định nghĩa các trạng thái của robot
/*1 là cảm biến tương ứng ở trên vạch; 0 là cảm biến ở
ngoài vạch*/
#define   L4 0B00000000
#define   L3 0B01000000
#define   L2 0B01100000
#define   L1 0B00110000 //tương ứng hình 3.33.b
#define   TT 0B00011000 //tương ứng hình 3.33.a
#define   R1 0B00001100 //tương ứng hình 3.33.f
#define   R2 0B00000110
#define   R3 0B00000010
#define   R4 0B00000000
unsigned int counter0; //biến đếm của encoder
//*******************************************
//chương trình con điều khiển động cơ bên trái
void runL (unsigned char power,unsigned char dir)
{
OCR0 = power; //biến power chứa tỷ lệ xung PWM
if(dir == go) //chiều tiến
```

```
{
DROLE1 = 1;
DROLE2 = 0;
}
else if (dir == back) //chiều lùi
 {
DROLE1 = 0;
DROLE2 = 1;
 }
}
//************************************
//chương trình con dừng (khoá) động cơ trái
void stopL(void)
{
DROLE1 = 1;
DROLE2 = 1;
}
//************************************
//chương trình con điều khiển động cơ bên trái
void runR(unsigned char power,unsigned char dir)
{
OCR2 = power;
if(dir == go)
{
DROLE3 = 1;
DROLE4 = 0;
}
else if (dir == back)
 {
DROLE3 = 0;
DROLE4 = 1;
 }
}
//************************************
//chương trình con dừng (khoá) động cơ phải
void stop_R(void)
{
DROLE3 = 1;
DROLE4 = 1;
}
//************************************
//chương trình con dừng toàn bộ robot
```

```
void stop(void)
{
OCR0 = OCR2 = 0;
stop_L();
stop_R();
}
//*************************************
//chương trình con điều khiển chạy bám đường
//dò đường tốc độ cao nhất
void tracking1 (unsigned int cm)
{
unsigned char temp1,temp2,INPUT;
counter0 = 0;
do
{
INPUT = PINC; //đọc các cảm biến
INPUT = INPUT&0x81; //bỏ đi 2 chân không dùng
switch (INPUT)
{
 case TT: //robot chạy thẳng
 {
 runR(R14,go);
 runL(L14,go);
 temp1 = 0;
 temp2 = 0;
 break;
 }
 case L1: //robot chạy hơi lệch sang trái
 {
 runR(R14,go);
 runL(L13,go);
 temp1 = 0;
 temp2 = 0;
  break;
 }
 case L2: //robot chạy lệch sang trái
 {
 runR(R14,go);
 runL(L12,go);
 temp1 = 1;
 temp2 = 0;
 break;
```

```
}
case L3: //robot chạy lệch nhiều sang trái
{
runR(R14,go);
runL(L11,go);
temp1 = 0;//nhớ trạng thái lệch sang trái
temp2 = 0;
break;
}
case R1: //robot chạy hơi lệch sang phải
{
runR(R13,go);
runL(L14,go);
temp1 = 0;
temp2 = 0;
break;
}
case R2: //robot chạy lệch sang phải
{
runR(R12,go);
runL(L14,go);
temp1 = 0;
temp2 = 0;
break;
}
case R3: //robot chạy lệch nhiều sang phải
{
runR(R11,go);
runL(L14,go);
temp1 = 0;
temp2 = 1; //nhớ trạng thái lệch phải
break;
}
//nếu lệch hẳn sang trái hoặc sang phải
case (L4||R4):
{
if (temp1 == 1) //lệch trái
 {
 runR(R12,go);
 runL(L12,back);
 }
if (temp2 == 1) //lệch phải
```

```
  {
  runR(R12,back);
  runL(L12,go);
  }
 break;
  }
}
}
}
while((100*cm)>=(counter0*18));
/*encoder 100 xung/vòng;bánh tỳ có chu vi là 18cm*/
stop();
}
//*****************************************
//dò đường tốc độ trung bình
void tracking2 (unsigned int cm)
{
unsigned char temp1,temp2,INPUT;
counter0 = 0;
do
{
INPUT = PINC; //đọc các cảm biến
INPUT = INPUT&0x81; //bỏ đi 2 chân không dùng
switch (INPUT)
{
 case TT: //robot chạy thẳng
 {
 runR(R24,go);
 runL(L24,go);
 temp1 = 0;
 temp2 = 0;
 break;
 }
 case L1: //robot chạy hơi lệch sang trái
 {
 runR(R24,go);
 runL(L23,go);
 temp1 = 0;
 temp2 = 0;
  break;
 }
 case L2: //robot chạy lệch sang trái
```

```
{
runR(R24,go);
runL(L22,go);
temp1 = 1;
temp2 = 0;
break;
}
case L3: //robot chạy lệch nhiều sang trái
{
runR(R24,go);
runL(L21,go);
temp1 = 0;//nhớ trạng thái lệch sang trái
temp2 = 0;
break;
}
case R1:  //robot chạy hơi lệch sang phải
{
runR(R23,go);
runL(L24,go);
temp1 = 0;
temp2 = 0;
break;
}
case R2:  //robot chạy lệch sang phải
{
runR(R22,go);
runL(L24,go);
temp1 = 0;
temp2 = 0;
break;
}
case R3: //robot chạy lệch nhiều sang phải
{
runR(R21,go);
runL(L24,go);
temp1 = 0;
temp2 = 1;  //nhớ trạng thái lệch phải
break;
}
//nếu lệch hẳn sang trái hoặc sang phải
case (L4||R4):
{
```

```
 if (temp1 == 1) //lệch trái
  {
  runR(R22,go);
  runL(L22,back);
  }
 if (temp2 == 1) //lệch phải
  {
  runR(R22,back);
  runL(L22,go);
  }
 break;
  }
}
}
}
while((100*cm)>=(counter0*18));
/*encoder 100 xung/vòng;bánh tỳ có chu vi là 18cm*/
stop();
}
//*****************************************
//dò đường tốc độ thấp
void tracking3 (unsigned int cm)
{
unsigned char temp1,temp2,INPUT;
counter0 = 0;
do
{
INPUT = PINC; //đọc các cảm biến
INPUT = INPUT&0x81; //bỏ đi 2 chân không dùng
switch (INPUT)
{
 case TT: //robot chạy thẳng
 {
 runR(R34,go);
 runL(L34,go);
 temp1 = 0;
 temp2 = 0;
 break;
 }
 case L1: //robot chạy hơi lệch sang trái
 {
 runR(R34,go);
```

```
runL(L33,go);
temp1 = 0;
temp2 = 0;
 break;
}
case L2: //robot chạy lệch sang trái
{
runR(R34,go);
runL(L32,go);
temp1 = 1;
temp2 = 0;
break;
}
case L3: //robot chạy lệch nhiều sang trái
{
runR(R34,go);
runL(L31,go);
temp1 = 0;//nhớ trạng thái lệch sang trái
temp2 = 0;
break;
}
case R1:  //robot chạy hơi lệch sang phải
{
runR(R33,go);
runL(L34,go);
temp1 = 0;
temp2 = 0;
break;
}
case R2:  //robot chạy lệch sang phải
{
runR(R32,go);
runL(L34,go);
temp1 = 0;
temp2 = 0;
break;
}
case R3: //robot chạy lệch nhiều sang phải
{
runR(R31,go);
runL(L34,go);
temp1 = 0;
```

```
  temp2 = 1;  //nhớ trạng thái lệch phải
  break;
  }
  //nếu lệch hẳn sang trái hoặc sang phải
  case (L4||R4):
  {
  if (temp1 == 1)  //lệch trái
   {
   runR(R32,go);
   runL(L32,back);
   }
  if (temp2 == 1)  //lệch phải
   {
   runR(R32,back);
   runL(L32,go);
   }
  break;
   }
}
}
}
while((100*cm)>=(counter0*18));
/*encoder 100 xung/vòng;bánh tỳ có chu vi là 18cm*/
stop();
}
//*************************************************
//dùng ngắt ngoài 0 để đếm xung từ encoder
interrupt [EXT_INT0] void ext_int0_isr(void)
{
counter0++;
}
//chương trình chính
void main (void)
{
//khởi tạo chiều vào/ra của các port
PORTA = 0xFF;
DDRA = 0x00;
PORTC = 0xFF;
DDRC = 0x00;
//khởi tạo timer 0 ở chế độ PWM
ASSR = 0x00;
TCCR0 = 0x6C;
```

```
TCNT0 = 0x00;
OCR0 = 0x00;
//khởi tạo timer 2 ở chế độ PWM
TCCR2 = 0x6B;
TCNT2 = 0x00;
OCR2 = 0x00;
//khởi tạo ngắt ngoài 0
EICRA = 0x2A;
EICRB = 0x00;
EIMSK = 0x07;
EIFR = 0x07;
TIMSK = 0x00;
ETIMSK = 0x00;
ACSR = 0x80;
SFIOR = 0x00;
#asm("sei")
//chạy khởi động 20cm bằng tốc độ chậm nhất
tracking3(20);
//tăng dần tốc độ
tracking2(30);
tracking2(100);
//giảm dần tốc độ
tracking2(30);
tracking3(20);
stop();
//tổng cộng robot đã đi một đoạn đường dài 2m
}
```

Phụ lục

TRÌNH DỊCH C CHO HỌ VI ĐIỀU KHIỂN 8051

1. GIỚI THIỆU

Có thể nói trình dịch C là một công cụ vô cùng hữu hiệu cho người lập trình với vi điều khiển. Ngoài sự ngắn gọn, dễ viết, dễ hiểu, dễ kiểm soát lỗi, dễ bảo dưỡng của một chương trình viết bằng ngôn ngữ C so với một chương trình viết bằng hợp ngữ ra, trình dịch C còn cung cấp cho người lập trình những công cụ mà trình dịch ASSEMBLER không thể có, có thể kể ra đây một số công cụ nổi bật:

• Trình dịch C cho phép người sử dụng có thể viết hệ điều hành thời gian thực (RTOS–Real Time Operating Systems) trên họ vi điều khiển 8051.

• Trình dịch C cung cấp một thư viện phong phú gồm khá nhiều hàm giúp người sử dụng có thể dễ dàng làm các tác vụ trên bộ vi điều khiển.

• Trình dịch C cung cấp một trình debug khá hiệu quả, cho phép người lập trình có thể kiểm soát và sửa các lỗi trên các chương trình.

2. KHUNG MỘT CHƯƠNG TRÌNH VIẾT CHO VI ĐIỀU KHIỂN

```
//khai báo các thư viện
#include <stdio.h>
#include <reg51.h>
//khai báo các biến số, hằng số,cấu trúc,chương trình con.
xdata unsigned char CREG _at_ 0x0003;
int x;
char m[10];
//chương trình con có tên là delay, 1 đối số
void delay(long int t);
//chương trình chính
void main (void)
      {
      SCON = 0x52;
      TMOD = 0x20;
      TH1 = TL1= -3;
      TR1 = 1;
      .
      .
      .
      }
```

```
/*điểm đặt chương trình con*/
void delay(long int t)
      {
      unsigned int i;
      for(i = 1;i< =t;++i);
      }
```

Chú ý: Trình dịch C phân biệt chữ hoa và chữ thường.

2.1. Hằng số

Một hằng số thông thường được định nghĩa bởi từ khoá Const:

```
const unsigned char tens[] = { 1, 10, 100, 1000 };
```

Hằng số trong ROM được định nghĩa bởi từ khoá code:

```
unsigned char code coolant_temp = 0x02 ;
unsigned char code look_up_table[5]  { '1','2','3','4''};
unsigned int code pressure = 4 ;
```

Một mảng các giá trị nằm trong ROM có thể được định nghĩa như sau:

```
unsigned char code ma_hoa_file_anh[] =
{
0x08,0x08,
0x00,0x00,0x00,0x09,0x41,0x80,0xC0,0xFF,
0x00,0x00,0x13,0x1A,0x26,0x33,0x80,0xFF,
0x00,0x00,0x00,0x09,0x41,0x80,0x66,0x66,
0x00,0x00,0x00,0x05,0x4A,0x46,0x40,0x40,
0x00,0x00,0x00,0x08,0x43,0x43,0x3D,0x3A,
0x00,0x00,0x00,0x00,0x2D,0x4D,0x56,0x4D,
0x00,0x00,0x00,0x00,0x21,0x56,0x6C,0x6F
};
```

2.2. Biến

Một số kiểu biến được dùng trong ngôn ngữ lập trình C như sau:

Loại biến	Độ dài	Khoảng giá trị
bit	1-bit	0 - 1
signed char	8-bit	0 - +/- 127
unsigned char	8-bit	0 - 255
signed int	16-bit	0 - +/-32768
unsigned int	16-bit	0 - 65535
signed long int	32-bit	0 - +/- 2.147483648x109
unsigned long int	32-bit	0 - 4.29496795x109
float	32-bit	+/-1.176E-38 đến +/-3.4E+38
pointer	24/16/8bit	

Ngoài các biến trên, trong trình dịch C còn một số loại biến đặc biệt, *ví dụ:*

```
#define MOTOR_ON 1;
#define MOTOR_OFF 0;
sbit Co_tran = RI;
sbit Motor_Control = P1^1; /*Biến Motor_Control được
                             gán cho chân P1.1. */
sbit Motor_State = P1^2;
Motor = MOTOR_ON;
.
.
Motor = MOTOR_OFF;
```

Các biến có chỉ định địa chỉ, ví dụ:

```
pdata unsigned char PA _at_ 0x90;/* từ khoá pdata dùng
cho địa chỉ 8 bit*/
xdata unsigned char PA _at_ 0x9000;/*từ khoá xdata dùng
cho địa chỉ 16 bit*/
```

** Chuyển đổi giữa các loại biến:*

Chúng ta cùng xem xét ví dụ sau:

```
unsigned char x = 10;
unsigned char y = 5;
unsigned char z ;
z = x * y ;
```

Kết quả là z = 50

```
unsigned char x = 10;
unsigned char y = 50;
unsigned char z ;
z = x * y ;
```

Kết quả là z = 244 vì 500 = 0x1F4, do z được khai báo kiểu `unsigned char` nên nó chỉ là 8 bit cuối có giá trị là 0xF4 = 244.

Đoạn lệnh dưới đây sẽ chuyển đổi giữa 2 kiểu biến:

```
unsigned char x ;
unsigned char y ;
unsigned int z ;
z = (unsigned int) x * (unsigned int) y ;
```

2.3. Các định danh phần cứng

Các định danh phần cứng là tên của các thanh ghi, các bit chức năng có trên phần cứng của 8052/8952 đã được định nghĩa trong tập tin reg52.h, các bit chức năng có trên phần cứng của 8051/8951 đã được định

nghĩa trong tập tin reg51.h... Khi lập trình chúng ta chỉ cần nhớ tên của các định danh này, cụ thể chúng gồm có:

Các định danh là các thanh ghi:

```
sfr P0 = 0x80;/*từ khoá (sfr-special function register )
                dùng để định nghĩa các thanh ghi*/
sfr P1       = 0x90;
sfr P2       = 0xA0;
sfr P3       = 0xB0;
sfr PSW      = 0xD0;
sfr ACC      = 0xE0;
sfr B        = 0xF0;
sfr SP       = 0x81;
sfr DPL      = 0x82;
sfr DPH      = 0x83;
sfr PCON     = 0x87;
sfr TCON     = 0x88;
sfr TMOD     = 0x89;
sfr TL0      = 0x8A;
sfr TL1      = 0x8B;
sfr TH0      = 0x8C;
sfr TH1      = 0x8D;
sfr IE       = 0xA8;
sfr IP       = 0xB8;
sfr SCON     = 0x98;
sfr SBUF     = 0x99;
sfr T2CON    = 0xC8;
sfr RCAP2L   = 0xCA;
sfr RCAP2H   = 0xCB;
sfr TL2      = 0xCC;
sfr TH2      = 0xCD;
```

Các định danh là các bit:

```
sbit CY      = PSW^7;
sbit AC      = PSW^6;
sbit F0      = PSW^5;
sbit RS1     = PSW^4;
sbit RS0     = PSW^3;
sbit OV      = PSW^2;
sbit P       = PSW^0;
sbit TF1     = TCON^7;
sbit TR1     = TCON^6;
sbit TF0     = TCON^5;
sbit TR0     = TCON^4;
sbit IE1     = TCON^3;
```

```
sbit IT1     = TCON^2;
sbit IE0     = TCON^1;
sbit IT0     = TCON^0;
sbit EA      = IE^7;
sbit ET2     = IE^5;
sbit ES      = IE^4;
sbit ET1     = IE^3;
sbit EX1     = IE^2;
sbit ET0     = IE^1;
sbit EX0     = IE^0;
sbit PT2     = IP^5;
sbit PS      = IP^4;
sbit PT1     = IP^3;
sbit PX1     = IP^2;
sbit PT0     = IP^1;
sbit PX0     = IP^0;
sbit RD      = P3^7;
sbit WR      = P3^6;
sbit T1      = P3^5;
sbit T0      = P3^4;
sbit INT1    = P3^3;
sbit INT0    = P3^2;
sbit TXD     = P3^1;
sbit RXD     = P3^0;
sbit SM0     = SCON^7;
sbit SM1     = SCON^6;
sbit SM2     = SCON^5;
sbit REN     = SCON^4;
sbit TB8     = SCON^3;
sbit RB8     = SCON^2;
sbit TI      = SCON^1;
sbit RI      = SCON^0;
sbit T2EX    = P1^1;
sbit T2      = P1^0;
sbit TF2     = T2CON^7;
sbit EXF2    = T2CON^6;
sbit RCLK    = T2CON^5;
sbit TCLK    = T2CON^4;
sbit EXEN2   = T2CON^3;
sbit TR2     = T2CON^2;
sbit C_T2    = T2CON^1;
sbit CP_RL2  = T2CON^0;
```

Chú ý: Các định danh phần cứng được viết hoa.

2.4. Từ khoá (keywords)

Ngoài các từ khoá trong ngôn ngữ lập trình C ra trình dịch Keil C còn có một số từ khoá sau đây:

_at_	far	sbit
alien	idata	sfr
bdata	interrupt	sfr16
bit	large	small
code	pdata	_task_
compact	_priority_	using
data	reentrant	xdata

3. CÁC HÀM

Trong trình dịch C, hàm có thể do người lập trình xây dựng và cũng có thể là các hàm đã có sẵn trong thư viện của trình dịch, chúng ta sẽ xem xét cả hai loại này.

3.1. Xây dựng hàm

3.1.1. Khai báo hàm

Cách khai báo một hàm:

```
[giá trị trả về] [tên hàm(các đối số)];
```

Ví dụ:

```
int cong(int a,int b); //khai báo một hàm tên "cong" với
2 đối số là a,b
//hàm trả về kiểu int
void delay(void); //hàm không đối cũng không trả về giá trị
```

Ví dụ: Chương trình có dùng hàm

```
#include<reg52.h>
void delay(int d);
void delay(int d)
{
long int i;
for(i = 0;i<d;i++);
}
void main()
{
while(1)
{
P1 = 0x00;
delay(6000);//gọi hàm với đối d = 6000
P1 = 0xFF;
delay(6000);
}
}
```

3.1.2. Hàm ngắt trong trình dịch C

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

Cách khai báo :

```
void [tên hàm ngắt](void) interrupt [kiểu ngắt];
```

AT89S52 có một số nguồn ngắt tương ứng với một số hàm ngắt sau:

```
void timer0 (void) interrupt 1 //ngắt timer0
{
các câu lệnh;
}
void timer1 (void) interrupt 3 //ngắt timer1
{
các câu lệnh;
}
void ext0 (void) interrupt 0 //ngắt ngoài 0
{
các câu lệnh;
}
void ext1 (void) interrupt 2 //ngắt ngoài 1
{
các câu lệnh;
}
void serialport (void) interrupt 4 /*ngắt port nối tiếp*/
{
các câu lệnh;
}
void timer2 (void) interrupt 5 //ngắt timer2
{
các câu lệnh;
}
```

Chú ý: timer0, ext0... là các tên do người lập trình đặt cho các hàm trên, có thể thay đổi nhưng 0, 1, 2... 5 là số hiệu ngắt thì không thể thay đổi (xem chương 2).

Ví dụ : Chương trình có dùng ngắt

```
#include<reg52.h>
sbit freq = P1^0;
void timer1 (void) interrupt 3 //ngắt timer1 số hiệu
ngắt= 3
{
freq = ~freq; //đảo bit, tạo tần số cỡ 10kHz trên P1.0
}
void main()
{
```

```
    TMOD = 0x22; //chú ý viết hoa các thanh ghi và tên định
danh của 8051
    //0x22 là số 22h trong C
    TH1 = TL1 = -50; //giá trị nạp lại
    TR1 = 1;
    while(1); //không làm gì cả
    }
```

3.2. Các hàm có sẵn trong thư viện của trình dịch C

Trình dịch C có sẵn một số hàm, trong số những hàm này phần lớn cũng có trong ngôn ngữ C thông thường. Trong khuôn khổ của giáo trình này, chúng tôi chỉ liệt kê các hàm có trong thư viện của trình dịch C và mô tả chi tiết các hàm đặc trưng cho trình dịch, các hàm khác có thể tham khảo trong các tài liệu về ngôn ngữ lập trình C.

**** Các hàm thao tác với bộ nhớ đệm:***

memccpy

Cú pháp:

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

Chức năng: Copy các bytes từ bộ đệm này tới bộ đệm khác.

Ví dụ:

```
#include <string.h>
#include <stdio.h>
void tst_memccpy (void) {
 static char src1 [100] = "Copy this string to dst1";
 static char dst1 [100];
void *c;
c = memccpy (dst1, src1, 'g', sizeof (dst1));
if (c == NULL)
  printf ("'g' was not found in the src buffer\n");
else
  printf ("characters copied up to 'g'\n");
{
```

memchr

Cú pháp:

```
void *memchr (void *buf, /* buffer     */
       char c,   /* byte cần tìm   */
```

```
        int len);  /* số byte lớn nhất sẽ tìm */
```

Chức năng: Tìm ký tự c trong số len bytes đầu tiên của bộ đệm buf

Ví dụ:

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

memcmp

Cú pháp:

```
#include <string.h>
char memcmp (void *buf1,  /* buffer thứ nhất       */
        void *buf2,  /* buffer thứ hai       */
        int len);   /* số byte lớn nhất dùng để so sánh */
```

Chức năng:

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

Giá trị trả về	Ý nghĩa
< 0	*buf1 < buf2*
= 0	*buf1 = buf2*
> 0	*buf1 > buf2*

Ví dụ:

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

```
else
 printf ("hexchars == hexchars2\n");
```

memcpy

Cú pháp:

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

Chức năng:

Copy len byte từ src tới dest.

Ví dụ:

```
#include <string.h>
#include <stdio.h>
void tst_memcpy (void) {
 static char src1 [100] =
 "Copy this string to dst1";
 static char dst1 [100];
 char *p;
 p = memcpy (dst1, src1, sizeof (dst1));
 printf ("dst = \"%s\"\n", p);
}
```

memmove

Cú pháp:

```
#include <string.h>
void *memmove (void *dest,  /* buffer đích  */
        void *src,  /* buffer nguồn */
        int len);   /* số bytes được di chuyển */
```

Chức năng:

Di chuyển len byte từ src tới dest.

Ví dụ:

```
#include <string.h>
#include <stdio.h>
void tst_memmove (void) {
 static char buf [] = "This is line 1 "
            "This is line 2 "
            "This is line 3 ";
 printf ("buf before = %s\n", buf);
 memmove (&buf [0], &buf [16], 32);
 printf ("buf after = %s\n", buf);
}
```

memset

Cú pháp:

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

Chức năng:

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

Ví dụ:

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

void tst_memset (void) {
 char buf [10];

 memset (buf, '\0', sizeof (buf));
         /* fill buffer with null characters */

}
```

**** Các hàm chuyển đổi và phân lớp ký tự***

(Các hàm này bạn đọc có thể tham khảo chi tiết trong các tài liệu về ngôn ngữ lập trình C).

Tên hàm	
isalnum	isspace
isalpha	isupper
iscntrl	isxdigit
isdigit	toascii
isgraph	toint
islower	tolower
isprint	_tolower
ispunct	toupper
	_toupper

**** Các hàm chuyển đổi dữ liệu***

(Các hàm này bạn đọc có thể tham khảo chi tiết trong các tài liệu về ngôn ngữ lập trình C)

Tên hàm	
abs	cabs
atof / atof517	labs
atoi	strtod/ strtod517
atol	strtol
	strtoul

**** Các hàm toán học***

(Các hàm này bạn đọc có thể tham khảo chi tiết trong các tài liệu về ngôn ngữ lập trình C)

Tên hàm	
acos / acos517	rand
asin / asin517	sin / sin517
atan / atan517	sinh
atan2	srand
ceil	sqrt / sqrt517
cos / cos517	tan / tan517
cosh	tanh
exp / exp517	_chkfloat_
fabs	_crol_
floor	_cror_
fmod	_irol_
log / log517	_iror_
log10 / log10517	_lrol_
modf	_lror_
pow	

**** Các hàm định bộ nhớ***

calloc

Cú pháp:

```
#include <stdlib.h>
void *calloc (unsigned int num,  /*số phần tử của mảng */
       unsigned int len);  /*chiều dài của mảng */
```

Chức năng:

Cấp phát bộ nhớ cho num mảng, mỗi mảng gồm len phần tử bắt đầu từ phần tử 0.

Ví dụ:

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

void tst_calloc (void)
 {
 int xdata *p;      /* trỏ tới mảng gồm 100 phần tử */
 p = calloc (100, sizeof (int));
 if (p == NULL)
 printf ("Error allocating array\n");
 else
 printf ("Array address is %p\n", (void *) p);
```

malloc

Cú pháp:

```
#include <stdlib.h>
void *malloc (unsigned int size);
```

Mô tả:

Định phần một khối nhớ có kích thước *size* byte trong không gian nhớ. Giá trị trả về của hàm là giá trị của con trỏ nếu như bộ nhớ còn đủ kích thước dành cho khối nhớ yêu cầu, ngược lại giá trị trả về sẽ là 0.

Ví dụ:

```
#include <stdlib.h>
#include <stdio.h>
void tst_malloc (void)
{
 unsigned char xdata *p;
 p = malloc (1000);       /* định phần 1000 bytes */
 if (p == NULL)
 printf ("khong du bo nho\n");
 else
 printf ("khoi nho da duoc dinh phan\n");
}
```

realloc

Cú pháp:

```
#include <stdlib.h>
void *realloc (void xdata *p,  /* block cần thay đổi*/
        unsigned int size);/* kích thước mới của block */
```

Mô tả:

Thay đổi kích thước của khối nhớ. Biến con trỏ p sẽ trỏ tới khối nhớ cần thay đổi, biến size sẽ là kích thước mới của khối nhớ.

Ví dụ:

```
#include <stdlib.h>
#include <stdio.h>
void tst_realloc (void)
{
 void xdata *p;
 void xdata *new_p;
 p = malloc (100);
 if (p != NULL)
 {
  new_p = realloc (p, 200);
  if (new_p != NULL) p = new_p;
  else printf ("Reallocation failed\n");
 }
}
```

_getkey

Cú pháp:

```
#include <stdio.h>
char _getkey (void);
```

Mô tả:

Chờ nhận một ký tự được gửi đến từ Port nối tiếp.

Ví dụ:

```
do
{
x = _getkey();
}
while(x!= 0x41); //chờ nhận ký tự A
```

printf

Cú pháp:

```
#include <stdio.h>
int printf (const char *fmtstr  /* định dạng chuỗi ký tự*/
      [, arguments]…);          /* các biến */
```

Mô tả:

Định dạng chuỗi ký tự và ghi (xuất) ra đối tượng được chỉ định trong thủ tục putchar

Ví dụ:

```
#include <stdio.h>
char putchar (char c)
```

```
{
 if (c != '\n')
{
  line1();
  DATA = c;
 _nop_();
 _nop_();
 }
 if (c == '\n')
 {
  line1();
  DATA = c;
 _nop_();
 _nop_();
 }
}
void main (void)
{
 char a;
 int b;
 long c;
 unsigned char x;
 unsigned int y;
 unsigned long z;
 float f,g;
 char buf [] = "Test String";
 char *p = buf;
 a = 1;
 b = 12365;
 c = 0x7FFFFFFF;
 x = 'A';
 y = 54321;
 z = 0x4A6F6E00;
 f = 10.0;
 g = 22.95;
 printf ("char %bd int %d long %ld\n",a,b,c);
 printf ("Uchar %bu Uint %u Ulong %lu\n",x,y,z);
 printf ("xchar %bx xint %x xlong %lx\n",x,y,z);
 printf ("String %s is at address %p\n",buf,p);
 printf ("%f != %g\n", f, g);
 printf ("%*f != %*g\n", 8, f, 8, g);
 while(1);
}
```

putchar

Cú pháp:

```
#include <stdio.h>
char putchar (char c);
```

Mô tả:

Truyền ký tự c qua port nối tiếp.

Ví dụ:

```
#include <stdio.h>
...
void tst_putchar (void) {
 unsigned char i;
 for (i = 0x20; i < 0x7F; i++)
  putchar (i);
}
```

4. CÁC CẤU TRÚC ĐIỀU KHIỂN, RẼ NHÁNH

Trong khuôn khổ giáo trình này, chúng tôi cũng chỉ đề cập tới một số các cấu trúc cơ bản, các cấu trúc khác trong ngôn ngữ lập trình C cũng hoàn toàn có giá trị trong trình dịch C (ngoại trừ kiểu file).

**** Cấu trúc While***

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

Nếu biểu thức điều kiện đúng, chương trình sẽ thực hiện các câu lệnh bên trong thân của vòng lặp while.

Ví dụ:

```
#include<reg52.h>
void main()
{
while(1)
{
while(P1 == 0xFE)
{
P2 = 0x02;
}
}
}
```

**** Cấu trúc IF, IF... ELSE***

```
if(biểu thức điều kiện)
{
các câu lệnh;
```

```
}
else
{
các câu lệnh;
}
```

Ví dụ:

```
#include<reg52.h>
void main()
{
while(1)
{
if(P1 == 1)
{
P2 = 0x02;
}
}
}
```

*** *Cấu trúc SWITCH***

switch (biểu thức nguyên):

```
{
case n1:
các câu lệnh;
case n2:
các câu lệnh;
case nk:
các câu lệnh;
default:
các câu lệnh;
}
```

Cấu trúc switch cho phép ta căn cứ vào giá trị của biểu thức nguyên để chọn một trong nhiều cách thực hiện.

Ví dụ:

```
#include<reg52.h>
unsigned char i; //khai báo biến kiểu byte
void main()
{
while(1)
{
i = P1; //đọc cổng P1
switch(i):
{
case 0:
P3 = 1;
case 4:
P3 = 7;
case 6:
```

```
P3 = 3;
}
}
}
```

**** Cấu trúc FOR***

for (biểu thức1; biểu thức2; biểu thức3)

```
{
các câu lệnh;
}
```

Ví dụ:

```
#include<reg52.h>
void delay(void)
{
long int i;
for (i  = 0;i < 6000;i++);
}
void main()
{
while(1)
{
P1 = 0x00;
delay();
P1 = 0xFF;
delay();
}
}
```

**** Cấu trúc Do... While***

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

Chương trình sau khi thực hiện các câu lệnh sẽ kiểm tra điều kiện nếu đúng sẽ thực hiện tiếp.

Ví dụ:

```
sbit contac = P1^1;
sbit led_do = P1^2;
bit x;
...
do
{
x = contac;
led = ~led;
}
while(x!= 1);
```

TÀI LIỆU THAM KHẢO

1. Tống Văn On, Hoàng Đức Hải, 2005; **Họ vi điều khiển 8051**; NXB Lao động – Xã hội.
2. Nguyễn Tăng Cường, Phan Quốc Thắng, 2004; **Cấu trúc và lập trình vi điều khiển**; NXB Khoa học và Kỹ thuật.
3. Nguyễn Linh Giang, Lê Văn Thái, Kiều Xuân Thực, 2007; **Giáo trình Kỹ thuật lập trình C**; NXB Giáo dục.
4. Ngô Diên Tập, Vũ Trung Kiên, Phạm Xuân Khánh, Kiều Xuân Thực, 2007; **Giáo trình Vi xử lý và cấu trúc máy tính**; NXB Giáo dục.
5. www.keil.com/c51
6. www.atmel.com/atmel/acrobat/doc1919.pdf

MỤC LỤC

Phụ lục

TRÌNH DỊCH C CHO HỌ VI ĐIỀU KHIỂN 8051

Chịu trách nhiệm xuất bản:
Chủ tịch HĐQT kiêm Tổng Giám đốc NGÔ TRẦN ÁI
Phó Tổng Giám đốc kiêm Tổng biên tập NGUYỄN QUÝ THAO

Tổ chức bản thảo và chịu trách nhiệm nội dung:
Chủ tịch HĐQT kiêm Giám đốc Công ty CP Sách ĐH–DN
TRẦN NHẬT TÂN

Biên tập và sửa bản in:
DƯƠNG VĂN BẰNG
Trình bày bìa:
HỒNG NHUNG
Chế bản:
ĐINH XUÂN DŨNG

VI ĐIỀU KHIỂN
CẤU TRÚC – LẬP TRÌNH VÀ ỨNG DỤNG
Mã số : 7B718Y8 – DAI

In 1.000 cuốn (QĐ : 14), khổ 16 x 24cm. In tại Công ty CP In – TM Hà Tây.
Địa chỉ : Số 15, đường Quang Trung, TP. Hà Đông.
Số ĐKKH xuất bản : 113 – 2008/CXB/72 – 175/GD.
In xong và nộp lưu chiểu tháng 3 năm 2008.

CÔNG TY CỔ PHẦN SÁCH ĐẠI HỌC - DẠY NGHỀ
HEVOBCO
25 HÀN THUYÊN – HÀ NỘI
Website : www.hevobco.com.vn

VƯƠNG MIỆN KIM CƯƠNG
CHẤT LƯỢNG QUỐC TẾ

TÌM ĐỌC SÁCH THAM KHẢO KỸ THUẬT
CỦA NHÀ XUẤT BẢN GIÁO DỤC

(Bộ sách giáo trình dùng cho các trường đào tạo hệ cao đẳng và đại học)

1. Xử lý số tín hiệu	*PGS. TS. Nguyễn Quốc Trung*
2. Vi xử lý và cấu trúc máy tính	*TS. Ngô Diên Tập*
3. Lý thuyết điều khiển tự động	*PGS. TS. Phan Xuân Minh*
4. Linh kiện điện tử	*TS. Nguyễn Viết Nguyên*
5. Kỹ thuật xung số	*PGS. TS. Đặng Văn Chuyết*
6. Kỹ thuật mạch điện tử	*PGS. TS. Đặng Văn Chuyết*
7. Kỹ thuật lập trình C	*TS. Nguyễn Linh Giang*
8. Thiết bị điều khiển khả trình - PLC	*ThS. Phạm Xuân Khánh*
9. Vi điều khiển - Cấu trúc - Lập trình và ứng dụng	*ThS. Kiều Xuân Thực*

Bạn đọc có thể mua tại các Công ty Sách - Thiết bị trường học ở các địa phương hoặc các Cửa hàng của Nhà xuất bản Giáo dục :

Tại Hà Nội : 25 Hàn Thuyên ; 187B Giảng Võ ; 232 Tây Sơn ; 23 Tràng Tiền ;
Tại Đà Nẵng : Số 15 Nguyễn Chí Thanh ; Số 62 Nguyễn Chí Thanh ;
Tại Thành phố Hồ Chí Minh : Cửa hàng 451B - 453, Hai Bà Trưng Quận 3 ;
240 Trần Bình Trọng – Quận 5.
Tại Thành phố Cần Thơ : Số 5/5, đường 30/4 ;
Website : www.nxbgd.com.vn

8 934980 813232

Giá: 23.500đ