BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC QUY NHƠN

TS. Đỗ Văn Cần (Chủ biên)

ThS. Bùi Văn Vũ

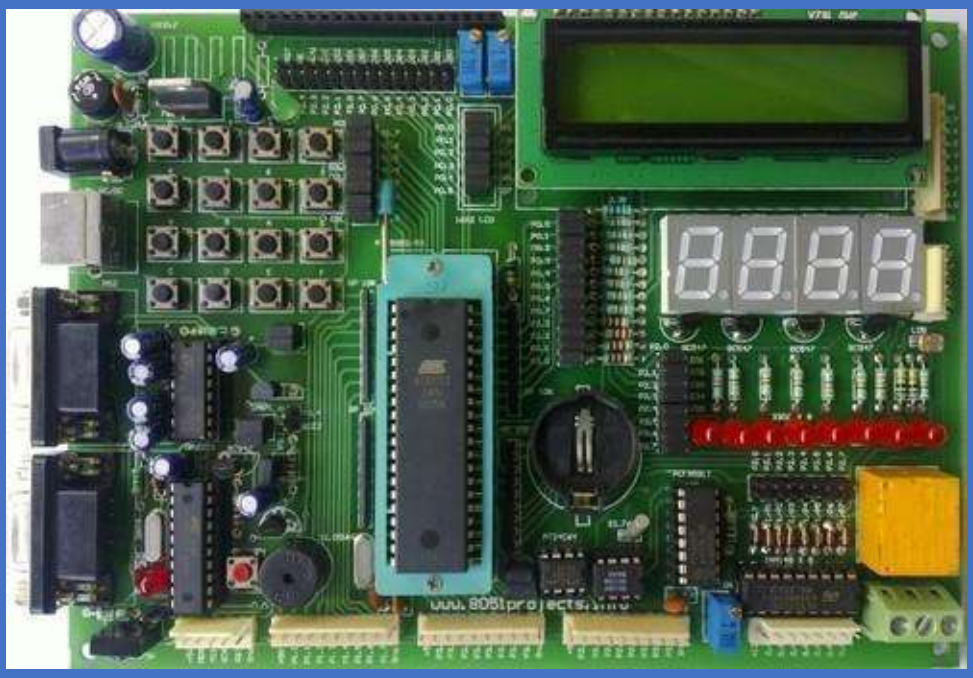
GIÁO TRÌNH

KỸ THUẬT VI ĐIỀU KHIỂN

Trình độ đại học ngành Kỹ thuật điện

Microcontroller

8051



Bình Định, 2/2022

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC QUY NHƠN

TS. Đỗ Văn Cần (Chủ biên)
ThS. Bùi Văn Vũ

GIÁO TRÌNH

KỸ THUẬT VI ĐIỀU KHIỂN

Trình độ đại học ngành Kỹ thuật điện

SỐ TÍN CHỈ: 2 (LÝ THUYẾT 30 TIẾT)

Bình Định, 02/2022

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG

LỜI NÓI ĐẦU

Vi điều khiển được hiểu như một máy tính được tích hợp trên một chip, nó thường được sử dụng để điều khiển các thiết bị điện tử. Vi điều khiển, là một hệ thống bao gồm một vi xử lý có hiệu suất đủ dùng và giá thành thấp (khác với các bộ vi xử lý đa năng dùng trong máy tính) kết hợp với các khối ngoại vi như bộ nhớ, các module vào/ra, các module biến đổi số sang tương tự và tương tự sang số,... Ở máy tính thì các module thường được xây dựng bởi các chip và mạch ngoài.

Ngày nay, kỹ thuật vi xử lý, vi điều khiển càng được sử dụng nhiều trong đời sống chúng ta, các thiết bị thông minh trong đời sống sinh hoạt, công nghiệp được sử dụng bởi các vi điều khiển ngày càng nhiều hơn. Lĩnh vực vi điện tử, hệ thống điều khiển, giám sát vừa và nhỏ được thực hiện bằng các vi xử lý 8051, PIC, AVR, ARM hay các board mạch Arduino.

Cuốn sách trang bị người đọc kiến thức về vi xử lý, vi điều khiển họ 8051 để người đọc có cái nhìn toàn diện về các họ vi điều khiển nói chung như: cấu trúc, tổ chức bộ nhớ, tập lệnh của các dòng chip lập trình.

Chương 1: Nêu tổng quan về vi xử lý, vi điều khiển;

Chương 2: Cung cấp chi tiết về cấu trúc họ vi điều khiển 8051;

Chương 3: Giới thiệu phương pháp lập trình cho vi điều khiển sử dụng ngôn ngữ lập trình bậc thấp và bậc cao;

Chương 4: Thực hiện các ứng dụng của 8051 với các thiết bị ngoại vi, cũng là để triển khai các kiến thức từ các chương trước.

Nội dung cuốn sách trang bị kiến thức ba lĩnh vực: điều khiển, công nghệ thông tin và tự động hoá. Từ đó, kỹ sư có thể thực hiện các ứng dụng điều khiển ở cấp độ vi mô nhỏ bởi 8051, ứng dụng ngôn ngữ lập trình hợp ngữ hoặc C để giải quyết các bài toán từ đơn giản đến phức tạp. Giáo trình được trình bày một cách chi tiết và dễ hiểu để giúp người mới bắt đầu học vi điều khiển dễ dàng xây dựng các chương trình ứng dụng.

Lần đầu xuất bản cuốn sách có những thiếu sót mong người đọc góp ý qua Email: dovancan@qnu.edu.vn để tác giả hoàn chỉnh hơn.

Xin trân thành cảm ơn!

Nhóm tác giả.

Chương 1. **TỔNG QUAN VỀ KỸ THUẬT VI ĐIỀU KHIỂN**

1.1. Các phép toán số học và chuyển đổi cơ số

1.1.1. Các phép toán số học

1.1.1.1. Các hệ đếm

Trong các hệ thống số thì hệ thập phân gần gũi với chúng ta nhất vì nó được sử dụng hằng ngày. Khi hiểu các đặc điểm của nó sẽ giúp chúng ta dễ hiểu hơn những hệ thống số khác. Hệ thập phân – hay còn gọi là hệ cơ số 10. Bao gồm 10 chữ số (ký hiệu) đó là 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Sử dụng những chữ số này giúp chúng ta có thể biểu thị được đại lượng bất kỳ.

Ví dụ 1.1: Số 435.568

$435.568 = 4x10^2 + 3x10^1 + 5x10^0 + 5x10^{-1} + 6x10^{-2} + 8x10^{-3}$

Trong hệ thống nhị phân (binary) chỉ có hai giá trị số là 0 và 1. Nhưng có thể biểu diễn bất kỳ đại lượng nào mà hệ thập phân và các hệ thống số khác có thể biểu diễn được, tuy nhiên phải dùng nhiều số nhị phân để biểu diễn một đại lượng nhất định. Tất cả các phát biểu về hệ thập phân đều có thể áp dụng được cho hệ nhị phân. Hệ nhị phân cũng là hệ thống số theo vị trí. Mỗi nhị phân đều có giá trị riêng, tức trọng số, là luỹ thừa của 2. Để biểu diễn một số nhị phân lẻ ta cũng dùng dấu chấm thập phân để phân biệt phần nguyên và phần lẻ. Ý nghĩa của một số nhị phân được mô tả như sau:

Bảng 1.1: Biểu diễn ý nghĩa của số nhị phân

2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	← Giá trị vị trí
= 8	= 4	= 2	= 1	.	= 1/2	= 1/4	= 1/8	
1	1	0	0	.	1	0	1	← Số nhị phân
MSB				Dấu chấm thập phân			LSB	

Để tìm giá trị thập phân tương đương ta chỉ việc tính tổng các tích giữa mỗi số (0 hay 1) với giá trị vị trí của nó.

Ví dụ 1.2: $1100.1012 = (1x\ 2^3) + (1x\ 2^2) + (0x2^1) + (0x2^0) + (1x2^{-1}) + (0x2^{-2}) + (1x\ 2^{-3})$

$= 8 + 4 + 0 + 0 + 0.5 + 0 + 0.125 = 12.125^{(10)}$

Số nhị phân có 8 bit được gọi là 1 byte, số nhị phân có 4 bit gọi là nipple (nửa

byte). Một nhóm các bit nhị phân nói chung được gọi một word (từ) nhưng thường dùng để chỉ số có 16 bit, số 32 bit gọi là doubleword, 64 bit gọi là quadword.

Để thuận tiện cho việc chuyển đổi số ta cần phải biết một số lũy thừa nguyên của chúng. Lũy thừa của $2^{10} = 1024$ được gọi tắt là 1K (đọc K hay kilo), trong ngôn ngữ nhị phân 1K là 1024. Những giá trị lớn hơn tiếp theo như:

$2^{11} = 2^1.2^{10} = 2K$

$2^{12} = 2^2.2^{10} = 4K$ (Kilo)

$2^{20} = 2^{10}.2^{10} = 1K.1K = 1M$ (Mega)

$2^{30} = 2^{10}.2^{20} = 1K.1M = 1G$ (Giga)

$2^{40} = 1T$ (Tiga)

Biểu đồ thời gian dùng để biểu diễn sự thay đổi theo thời gian của tín hiệu số, đặc biệt là biểu diễn hai hay nhiều tín hiệu số trong cùng một mạch điện hay một hệ thống. Cách đếm một số nhị phân được trình bày theo bảng sau:

Nếu sử dụng n bit hoặc n chữ số thì ta có thể đếm được 2^n số độc lập nhau

Ví dụ 1.3: Hai bit ta đếm được $2^2 = 4$ số. Ở bước đếm cuối cùng, tất cả các bit đều ở trạng thái 1 và bằng $2^n – 1$ trong hệ thập phân.

- Hệ thống số thập lục phân sử dụng cơ số 16, nghĩa là có 16 ký tự số. Hệ thập lục phân dùng các ký số từ 0 đến 9 cộng thêm 6 chữ A, B, C, D, E, F. Mỗi một ký số thập lục phân biểu diễn một nhóm 4 ký số nhị phân. Ý nghĩa của hệ thống số thập lục phân được mô tả bằng bảng sau:

Bảng 1.2: Biểu diễn ý nghĩa của số thập lục

16^3	16^2	16^1	16^0		16^{-1}	16^{-2}	16^{-3}
=4096	=256	=16	=1	.	=1/16	=1/256	=1/4096
MSD				Dấu chấm thập phân			LSD

- Mối quan hệ giữa các hệ thống thập lục phân, thập phân và nhị phân được trình bày bằng bảng sau:

Bảng 1.3: Quan hệ số thập phân, nhị phân và thập lục

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Khi đếm số thập lục phân, mỗi vị trí được tăng dần 1 đơn vị từ 0 cho đến F. khi đếm đến giá trị F, vòng đếm lại trở về 0 và vị trí ký số kế tiếp tăng lên 1. Trình tự đếm được minh họa như dưới đây: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13,...,1A, 1B, ..., 20, 21,.., 26, 27, 28, 29, 2A, 2B, 2D, 2E, 2F,..., 40, 41, 42 …., 6F8, 6F9, 6FA, 6FB, 6FC, 6FD, 6FE, 6FF, 700,….

1.1.1.2. Các phép toán

- Phép cộng:

Ở Ví dụ 1.4 ta cộng các LSD (7 và 4) cho ra 11, là B ở hệ hex. Không có số nhớ, cộng 5 và 3 cho ra 8 và công có nhớ giữa 8 và A, tính nhẩm A ra giá trị thập phân là 10. Do đó tổng là 18, số này lớn hơn 16 nên ta trừ cho 16 được 2, viết số 2 và nhớ 1 sang vị trí kế tiếp. Cộng số nhớ cho 4 và 3 ta được tổng là 8. Tiếp theo tổng của C và D xem như $12 + 13 = 25_{10}$, số này lớn hơn 16 nên trừ cho 16 còn 9, và nhớ 1 sang vị trí kế tiếp. Cộng số nhớ này với B và 3 được 15_{10} đổi sang số hex là F. Cộng số nhớ này với 4 và 4 được 8, và kết quả cuối cùng là 8F9. Dưới đây là một vài ví dụ về cộng hai số nhị phân (số thập phân tương đương trong dấu ngoặc).

Ví dụ 1.4:

57	48	4BC	110 (6)	1011 (11)	11.011 (3.375)
+	+	+	+	+	+
34	3A	43D	010 (2)	1101 (13)	10.110 (2.750)
8B	82	8F9	1000 (8)	11000 (24)	110.001 (6.125)

Phép cộng là phép toán số học quan trọng nhất trong hệ thống số. Đúng vậy, các phép trừ, nhân và chia được thực hiện ở hầu hết máy vi tính và máy tính bấm tay hiện đại nhất thực ra chỉ dùng phép cộng làm phép toán cơ bản của chúng.

- Phép trừ:

Số bị trừ	0	1	1	0
	-	-	-	-
Số trừ	0	1	0	1
Hiệu	0	0	1	1 1

số mượn (borrow)

Trong phép trừ nếu số bị trừ nhỏ hơn số trừ, cụ thể là khi 0 trừ 1, thì phải mượn

1 ở hàng cao kế tiếp là 2 ở hàng đang trừ và số mượn này phải trả lại cho hàng cao kế tiếp, tương tự như phép trừ của hai số thập phân.

- Phép nhân:

Tương tự như thập phân

```
  0   1   1   0          1001 | 11          1010 | 100
  x   x   x   x        - 011  | 0011      - 100  | 0010.1
  0   1   0   1        ------               -----
  -   -   -   -          0011                100
  0   1   0   0        -   11              - 100
                       ------              -----
                          0                   0
```

- Phép chia:

Phép chia một số nhị phân (số bị chia) cho một số khác (số chia) được thực hiện giống như phép chia số thập phân. Tiến trình thực tế còn đơn giản hơn do khi kiểm tra xem có bao nhiêu lần số chia " đi vào" số bị chia, chỉ có hai khả năng đó là 0 và 1. Quá trình chia được minh họa ở trên.

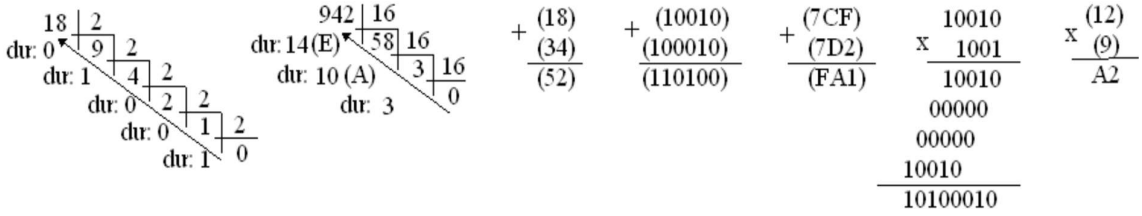
1.1.2. Chuyển đổi giữa các cơ số

Trong hệ tính toán các bộ vi xử lý thường ứng dụng 3 hệ tính toán: cơ số 10 (Decimal), cơ số 2 (Binary), cơ số 16 (Hexadecimal).

Chuyển từ cơ số 10 sang cơ số 2, 16:

Ví dụ 1.5 : 18_{10} → $???_2$

Sau đây là ví dụ chuyển đổi giữa các cơ số:



Sau khi chia xong được các số dư và tiến hành lấy các số dư từ dưới lên (theo chiều mũi tên). Vậy $18_{10} = 10010_2$, hoặc $942_{10} = 3AE_{16}$ Đối với các số chuyển đổi từ hệ Hexadecimal (16) sang hệ nhị phân (2) hoặc ngược lại, thực hiện theo nhóm 4 và tra theo **Error! Reference source not found.**.

Ví dụ 1.6: $A54F_{16}$ → $???_2$

A	5	4	F
1010	0101	0100	1111

Như vậy: $A54F_{16}$ → $1010\ 0101\ 0100\ 1111_2$

Ví dụ 1.7: Đổi số 765_{10} thành số thập lục phân.

Thực hiện phép chia, ta được:

$$\frac{765}{16} = 47 + \text{dư } 13 \leftarrow \text{LSD}$$

$$\frac{47}{16} = 2 + \text{dư } 15$$

$$\frac{2}{16} = 0 + \text{dư } 2 \leftarrow \text{MSD}$$

$$\Rightarrow 765_{10} = DF2_{16}$$

Cách đổi từ số thập lục phân sang số nhị phân cũng giống như đổi từ bát phân sang nhị phân, nghĩa là mỗi ký tự số thập lục phân được đổi sang giá trị nhị phân 4 bit tương đương.

Ví dụ 1.8: Đổi số 8D216 → $???_2$

$$8D2_{(16)} = 8\ \ D\ \ 2$$
$$= 100\ \ 1101\ \ 0010$$
$$= 100011010010_{(2)}$$

Để đổi từ số nhị phân sang thập lục phân ta làm ngược lại cách đổi từ thập lục phân sang nhị phân. Nghĩa là ta nhóm thành từng nhóm 4 bit, mỗi nhóm được đổi sang ký số thập lục phân tương đương. Số 0 có thể được thêm vào để hoàn chỉnh nhóm 4 bit.

Ví dụ 1.9: Đổi số 110011011012 thành số thập lục phân

$$11001101101_{(2)} = 110\ \ 0110\ \ 1101$$
$$= 6\ \ 6\ \ D$$
$$= 66D_{(16)}$$

1.2. Cấu trúc vi xử lý

Vào năm 1972, hãng Intel đưa ra bộ vi xử lí 8-bit đầu tiên với tên Intel-8008/8bit. Từ 1974 đến 1975, Intel chế tạo các bộ vi xử lí 8-bit 8080 và 8085A. Cũng vào khoảng thời gian này, một loạt các hãng khác trên thế giới cũng đã cho ra đời các

bộ vi xử lí tương tự như: 6800 của Motorola với 5000 tranzitor, Signetics 6520, 1801 của RCA, kế đến là 6502 của hãng MOS Technology và Z80 của hãng Zilog [1].

Năm 1978 xuất hiện Intel 8086 là loại bộ xi xử lý 16 bit với 29.000 tranzitor, Motorola 68000 tích hợp 70.000 tranzitor, APX 432 chứa 120.000 tranzitor.

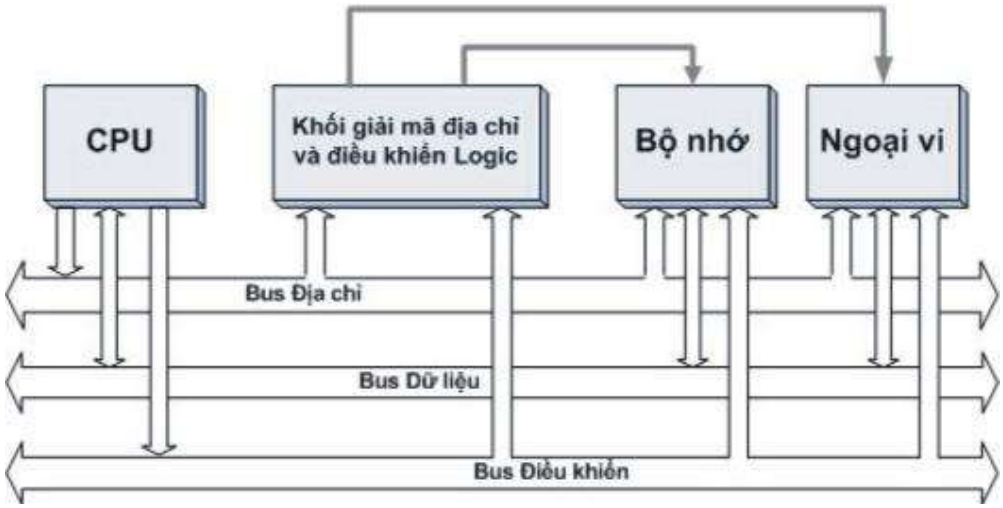
Bộ vi xử lý của Hewlet Pakard có khoảng 450.000 tranzitor. Từ năm 1974 đến 1984 số tranzitor tích hợp trong một chip tăng khoảng 100 lần.

Năm 1983, Intel đưa ra bộ vi xử lý 80286 dùng trong các máy vi tính họ AT (Advanced Technology). 80286 sử dụng I/O (Input/Output) 16 bit, 24 đường địa chỉ và không gian nhớ địa chỉ thực 16MB. Năm 1987, Intel đưa ra bộ vi xử lý 80386 32-bit. Năm 1989 xuất hiện bộ vi xử lý Intel 80486 là cải tiến của Intel 80386 với bộ nhớ ẩn và mạch tính phép toán đại số dấu phẩy động. Năm 1992, xuất hiện Intel 80586 còn gọi là Pentium 64 bit chứa 4 triệu tranzitor [1].

1.2.1. Định nghĩa vi xử lý – vi điều khiển

Từ các bộ vi xử lý ban đầu chỉ là các bộ xử lý trung tâm trong một hệ thống, không thể hoạt động nếu thiếu các bộ phận như RAM (Random Access Memory), ROM (Read-Only Memory), bo mạch chủ... các hãng đã phát triển các bộ vi xử lý này lên thành các bộ vi điều khiển để phục vụ các mục đích riêng biệt, khác nhau trong công nghiệp. Một bộ vi điều khiển là một hệ vi xử lý thật sự được tổ chức trong một chip (trong một vỏ IC) bao gồm một bộ vi xử lý (Microprocessor), bộ nhớ chương trình (ROM), bộ nhớ dữ liệu (RAM), tuy không bằng dung lượng RAM ở các máy vi tính nhưng đây không phải là một hạn chế vì các bộ vi điều khiển được thiết kế cho một mục đích hoàn toàn khác, ngoài ra trên chip còn có bộ xử lý số học - logic (ALU) cùng với các thanh ghi chức năng, các cổng vào/ra, cơ chế điều khiển ngắt, truyền tin nối tiếp, các bộ định thời... Hiện nay, các bộ vi điều khiển được sử dụng rất rộng rãi và ngày càng được chuẩn hóa để có thể sử dụng rộng rãi trong các ngành công nghiệp và nhiều thiết bị điện tử.

1.2.2. Cấu trúc vi điều khiển



Hình 1.1: Cấu trúc vi điều khiển

Cấu trúc cơ bản của một vi điều khiển bao gồm các thành phần chính sau:

+ CPU là bộ não trung tâm của vi điều khiển. CPU là thiết bị quản lý tất cả các hoạt động của hệ thống và thực hiện tất cả các thao tác trên dữ liệu như: nạp, giải mã và thực thi lệnh. CPU kết nối tất cả các thành phần của vi điều khiển thành một hệ thống duy nhất.

+ Memory(bộ nhớ): Trong vi điều khiển, bộ nhớ hoạt động giống như bộ vi xử lý. Bộ nhớ lưu trữ tất cả các chương trình và dữ liệu. Bộ nhớ của vi điều khiển là bộ nhớ ROM(EPROM, EEPROM) hoặc bộ nhớ RAM với dung lượng nhất định. Ngày nay còn có bộ nhớ flash lưu trữ mã nguồn chương trình.

+ Hệ thống bus và I/O sử dụng để giao tiếp hoặc điều khiển các thiết bị khác nhau như máy in, LCD, LED…

1.3. Bộ nhớ vi điều khiển

1.3.1. Bộ nhớ ROM

Bộ nhớ chỉ đọc hay ROM là loại bộ nhớ không khả biến dùng trong các máy tính hay hệ thống điều khiển, mà trong vận hành bình thường của hệ thống thì dữ liệu chỉ được đọc ra mà không được phép ghi vào.

ROM theo đúng nghĩa đối với các chip ở thế hệ đầu, cho phép chỉ đọc dữ liệu từ chúng và chỉ cho phép ghi dữ liệu một lần, gọi là nạp ROM.

Do công nghệ phát triển và nhu cầu thực tế, các thế hệ ROM sau được chế tạo với khả năng xoá được và nạp nhiều lần, mà việc thực hiện nạp ROM phải tuân theo quy trình đặc biệt đặc trưng. Nó cho ra khả năng mềm dẻo trong việc sửa đổi tính

năng vận hành của hệ thống điều khiển.

PROM (Programmable Read-Only Memory) hay Mask ROM: Được chế tạo bằng các mối nối (cầu chì - có thể làm đứt bằng mạch điện). Nó thuộc dạng WORM (Write-Once-Read-Many). Chương trình nằm trong PROM có thể lập trình được bằng những thiết bị đặc biệt. Loại ROM này chỉ có thể lập trình được một lần, và là rẻ nhất.

EPROM (Erasable Programmable Read-Only Memory): Được chế tạo bằng nguyên tắc phân cực tĩnh điện. Loại ROM này có thể bị xóa bằng tia cực tím và ghi lại thông qua thiết bị ghi EPROM.

EAROM (Electrically Alterable Read-Only Memory): Loại ROM này có thể thay đổi từng bit một lần. Tuy nhiên quá trình viết khá chậm và sử dụng điện thế không chuẩn. Việc viết lại EAROM không được thực hiện thường xuyên.

EEPROM (Electrically Erasable Programmable Read-Only Memory): Được tạo bằng công nghệ bán dẫn. Nội dung của ROM này có thể viết vào và xóa (bằng điện).

Một dạng phổ biến hiện hay dùng là Bộ nhớ flash, gọi đơn giản là Flash, được sử dụng với cả tư cách EEPROM lẫn trong ổ USB flash.

1.3.2. Bộ nhớ RAM

RAM là một loại bộ nhớ khả biến cho phép truy xuất đọc-ghi ngẫu nhiên đến bất kỳ vị trí nào trong bộ nhớ dựa theo địa chỉ bộ nhớ. Thông tin lưu trên RAM chỉ là tạm thời, chúng sẽ mất đi khi mất nguồn điện cung cấp.

RAM là bộ nhớ chính của máy tính và các hệ thống điều khiển, để lưu trữ các thông tin thay đổi đang sử dụng. Các hệ thống điều khiển còn sử dụng SRAM như làm một thiết bị lưu trữ thứ cấp (Secondary storage). Khi cần thiết thì bố trí một chân nhỏ làm nguồn điện phụ để duy trì dữ liệu trong RAM. RAM có một đặc tính là thời gian thực hiện thao tác đọc hoặc ghi đối với mỗi ô nhớ là như nhau, cho dù đang ở bất kỳ vị trí nào trong bộ nhớ. Mỗi ô nhớ của RAM đều có một địa chỉ. Thông thường, mỗi ô nhớ là một byte (8 bit), tuy nhiên hệ thống lại có thể đọc ra hay ghi vào nhiều byte (2, 4, 8 byte) một lúc.

RAM khác biệt với các thiết bị bộ nhớ tuần tự (sequential memory device) chẳng hạn như các băng từ, CD-RW, DVD-RW, ổ đĩa cứng, trong đó bắt buộc phải tìm đến sector và đọc/ghi cả khối dữ liệu ở đó để truy xuất. RAM là thuật ngữ phân biệt tương đối theo ý nghĩa sử dụng, với các chip nhớ truy xuất ngẫu nhiên là EEPROM (read-

only memory) cấm hoặc hạn chế chiều ghi và Bộ nhớ flash được phép đọc/ghi.

1.4. Một số dòng vi điều khiển thông dụng

1.4.1. Họ vi điều khiển 8051

Intel 8051 - là vi điều khiển đơn tinh thể (không nhầm với CPU) kiến trúc Harvard, lần đầu tiên được sản xuất bởi Intel năm 1980, để dùng trong các hệ thống nhúng. Trong những năm 1980 và đầu những năm 1990 đã rất nổi tiếng. Tuy nhiên hiện tại đã cũ và được thay thế bằng các thiết bị hiện đại hơn, với các lõi phối hợp 8051, được sản xuất bởi hơn 20 nhà sản xuất độc lập, như Atmel, Maxim IC (công ty con của Dallas Semiconductor), NXP Semiconductors (Philips Semiconductor trước đây), Winbond, Silicon Laboratories, Texas Instruments và Cypress Semiconductor. Tên gọi chính thức của họ vi điều khiển Intel 8051 — MCS 51.

Những vi điều khiển Intel 8051 được sản xuất với việc dùng công nghệ MOSFET, những phiên bản sau, chứa ký hiệu «C» trong tên, như 80C51, dùng công nghệ CMOS và yêu cầu công suất thấp, hơn các dòng MOSFET trước đây (điều này cho phép trang bị trong các thiết bị với nguồn nuôi là pin).

Hình 1.2: Hình ảnh chip họ 80C51

1.4.2. Họ vi điều khiển PIC

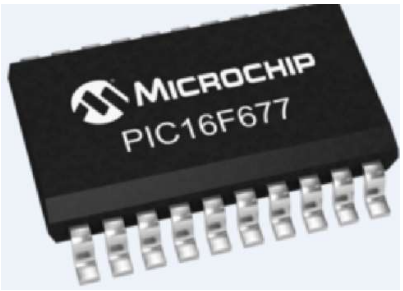
PIC là một họ vi điều khiển RISC được sản xuất bởi công ty Microchip Technology. Dòng PIC đầu tiên là PIC1650 được phát triển bởi Microelectronics Division thuộc General Instrument.

PIC bắt nguồn là chữ viết tắt của "Programmable Intelligent Computer" (Máy tính khả trình thông minh) là một sản phẩm của hãng General Instrument đặt cho dòng sản phẩm đầu tiên của họ là PIC1650. Lúc này, PIC1650 được dùng để giao tiếp với các thiết bị ngoại vi cho máy chủ 16 bit CP1600, vì vậy, người ta cũng gọi PIC với cái tên "Peripheral Interface Controller" (Bộ điều khiển giao tiếp ngoại vi).

CP1600 là một CPU tốt, nhưng lại kém về các hoạt động xuất nhập. Vì vậy PIC 8-bit được phát triển vào khoảng năm 1975 để hỗ trợ hoạt động xuất nhập cho CP1600. PIC sử dụng microcode đơn giản đặt trong ROM, và mặc dù, cụm từ RISC chưa được sử dụng thời bây giờ, nhưng PIC thực sự là một vi điều khiển với kiến trúc RISC, chạy một lệnh một chu kỳ máy (4 chu kỳ dao động).

Năm 1985 General Instrument bán bộ phận vi điện tử của họ, và chủ sở hữu mới hủy bỏ hầu hết các dự án - lúc đó đã quá lỗi thời. Tuy nhiên PIC được bổ sung EEPROM để tạo thành một bộ điều khiển vào ra khả trình. Ngày nay rất nhiều dòng PIC được xuất xưởng với hàng loạt các module ngoại vi tích hợp sẵn (như USART, PWM, ADC...), với bộ nhớ chương trình từ 512 Word đến 32K Word. Hiện nay, tại Việt Nam, đã có một cộng đồng nghiên cứu và phát triển PIC, dsPIC và PIC32.

PIC sử dụng tập lệnh RISC, với dòng PIC low-end (độ dài mã lệnh 12 bit, ví dụ: PIC12Cxxx) và mid-range (độ dài mã lệnh 14 bit, như: PIC16Fxxxx), tập lệnh bao gồm khoảng 35 lệnh, và 70 lệnh đối với các dòng PIC high-end (độ dài mã lệnh 16 bit, như: PIC18Fxxxx). Tập lệnh bao gồm các lệnh tính toán trên các thanh ghi, với các hằng số, hoặc các vị trí bộ nhớ, cũng như có các lệnh điều kiện, lệnh nhảy/gọi hàm, và các lệnh để quay trở về, nó cũng có các tính năng phần cứng khác như ngắt hoặc sleep (chế độ hoạt động tiết kiệm điện). Microchip cung cấp môi trường lập trình MPLAB, nó bao gồm phần mềm mô phỏng và trình dịch ASM.



Hình 1.3: Hình ảnh dòng PIC 16fxxx

Một số công ty khác xây dựng các trình dịch C, Basic, Pascal cho PIC. Microchip cũng bán trình dịch "C18" (cho dòng PIC high-end) và "C30" (cho dòng dsPIC30Fxxx). Họ cũng cung cấp các bản "student edition/demo" dành cho sinh viên hoặc người dùng thử, những version này không có chức năng tối ưu hoá code và có thời hạn sử dụng giới hạn. Những trình dịch mã nguồn mở cho C, Pascal, JAL, và Forth, cũng được cung cấp bởi PicForth.

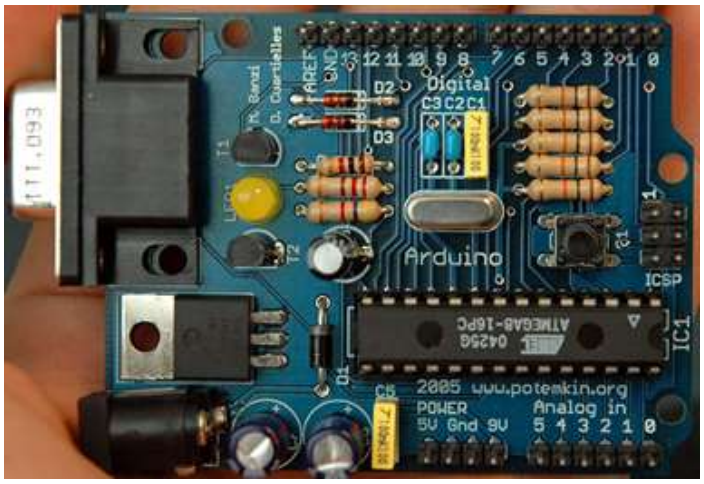
1.4.3. Board mạch Arduino

Arduino là một board mạch vi xử lý, nhằm xây dựng các ứng dụng tương tác với nhau hoặc với môi trường được thuận lợi hơn. Phần cứng bao gồm một board mạch nguồn mở được thiết kế trên nền tảng vi xử lý AVR Atmel 8bit, hoặc ARM Atmel 32-bit. Những Model hiện tại được trang bị gồm một cổng giao tiếp USB, 6 chân đầu vào analog, 14 chân I/O số tương thích với nhiều board mở rộng khác nhau.

Arduino được khởi động vào năm 2005 như là một dự án dành cho sinh viên trại Interaction Design Institute Ivrea (Viện thiết kế tương tác Ivrea) tại Ivrea, Italy.

Lý thuyết phần cứng được đóng góp bởi một sinh viên người Colombia tên là Hernando Barragan. Sau khi nền tảng Wiring hoàn thành, các nhà nghiên cứu đã làm việc với nhau để giúp nó nhẹ hơn, rẻ hơn, và khả dụng đối với cộng đồng mã nguồn mở. Trường này cuối cùng bị đóng cửa, vì vậy các nhà nghiên cứu, một trong số đó là David Cuarlielles, đã phổ biến ý tưởng này.

Một mạch Arduino bao gồm một vi điều khiển AVR với nhiều linh kiện bổ sung giúp dễ dàng lập trình và có thể mở rộng với các mạch khác. Một khía cạnh quan trọng của Arduino là các kết nối tiêu chuẩn của nó, cho phép người dùng kết nối với CPU của board với các module thêm vào có thể dễ dàng chuyển đổi, được gọi là shield. Vài shield truyền thông với board Arduino trực tiếp thông qua các chân khác nhau, nhưng nhiều shield được định địa chỉ thông qua serial bus I^2C, tức nhiều shield có thể được xếp chồng và sử dụng dưới dạng song song. Arduino chính thức thường sử dụng các dòng chip megaAVR, đặc biệt là ATmega8, ATmega168, ATmega328, ATmega1280, và ATmega2560. Một vài các bộ vi xử lý khác cũng được sử dụng bởi các mạch Aquino tương thích. Hầu hết các mạch gồm một bộ điều chỉnh tuyến tính 5V và một thạch anh dao động 16 MHz (hoặc bộ cộng hưởng ceramic trong một vài biến thể), mặc dù một vài thiết kế như LilyPad chạy tại 8 MHz và bỏ qua bộ điều chỉnh điện áp onboard do hạn chế về kích cỡ thiết bị. Một vi điều khiển Arduino cũng có thể được lập trình sẵn với một boot loader cho phép đơn giản là upload chương trình vào bộ nhớ flash on-chip, so với các thiết bị khác thường phải cần một bộ nạp bên ngoài. Điều này giúp cho việc sử dụng Arduino được trực tiếp hơn bằng cách cho phép sử dụng một máy tính gốc như là một bộ nạp chương trình.



Hình 1.4: Board mạch Adruino dùng Atmega8

Một mạch Arduino bao gồm một vi điều khiển AVR với nhiều linh kiện bổ sung giúp dễ dàng lập trình và có thể mở rộng với các mạch khác. Một khía cạnh quan trọng của Arduino là các kết nối tiêu chuẩn của nó, cho phép người dùng kết nối với CPU của board với các module thêm vào có thể dễ dàng chuyển đổi, được gọi là shield. Vài shield truyền thông với board Arduino trực tiếp thông qua các chân khác nhau, nhưng nhiều shield được định địa chỉ thông qua serial bus I^2C, tức nhiều shield có thể được xếp chồng và sử dụng dưới dạng song song.

Có nhiều biến thể như Arduino-compatible và Arduino-derived. Một vài trong số đó có chức năng tương đương với Arduino và có thể sử dụng để thay thế qua lại. Nhiều mở rộng cho Arduino được thực thiện bằng cách thêm vào các driver đầu ra, thường sử dụng trong các trường học để đơn giản hóa các cấu trúc của các 'con rệp' và các robot nhỏ. Những board khác thường tương đương về điện nhưng có thay đổi về hình dạng-đôi khi còn duy trì độ tương thích với các shield, đôi khi không. Vài biến thể sử dụng bộ vi xử lý hoàn toàn khác biệt, với các mức độ tương thích khác nhau.

1.5. Bài tập và câu hỏi cuối chương

Bài tập 1.1: Hãy chuyển đổi các số sau về dạng cơ số (2), (10), (16):

a) 111.101_2 → ……...$_{16}$; …….$_{10}$ b) 900_{10} → ……...$_{16}$; …….$_2$

Bài tập 1.2: Thực hiện phép toán sau đây:

a) $1011_2 + 30_{10}$= ……$_{16}$

b) $111_2 * C1_{16}$ = ……$_{10}$

c) 1234_{10} - 10_2 = ……$_{16}$

d) 111_{16} / 10_{16} = ……$_2$

Bài tập 1.3: Máy tính cơ bản có những thành phần nào? Nêu ý nghĩa và chức năng từng thành phần đó?

Tài liệu tham khảo chương 1

[1] Tống Văn On, ***Họ vi điều khiển 8051***, NXB Lao động Xã hội, Hà Nội, 2007.

[2] Ngô Diên Tập, ***Vi điều khiển với lập trình C***, NXB Khoa học Kỹ thuật, Hà Nội, 2006.

[3] Nguyễn Đình Phú, Trương Ngọc Anh, ***Giáo trình vi điều khiển***, NXB Đại học Sư phạm Kỹ thuật, 2006.

Chương 2. **HỌ VI ĐIỀU KHIỂN MCS-51**

2.1. Sơ đồ và chức năng các chân của họ 8051

2.1.1. Sơ đồ cấu trúc

8051 là một họ vi điều khiển do Intel phát triển và sản xuất. Một số nhà sản xuất được phép cung cấp các IC tương thích với các sản phẩm 8051 của Intel là Siemens, Advanced Micro Devices, Fujitsu, Philips, Atmel…

Các IC của họ 8051 có các đặc trưng chung như sau [2]:

- 4 port I/O x 8 bit;
- Giao tiếp nối tiếp;
- 64K không gian bộ nhớ chương trình mở rộng;
- 64K không gian bộ nhớ dữ liệu mở rộng;
- Một bộ xử lý luận lý (thao tác trên các bit đơn);
- 210 bit được địa chỉ hóa;
- Bộ nhân/chia 4 μs.

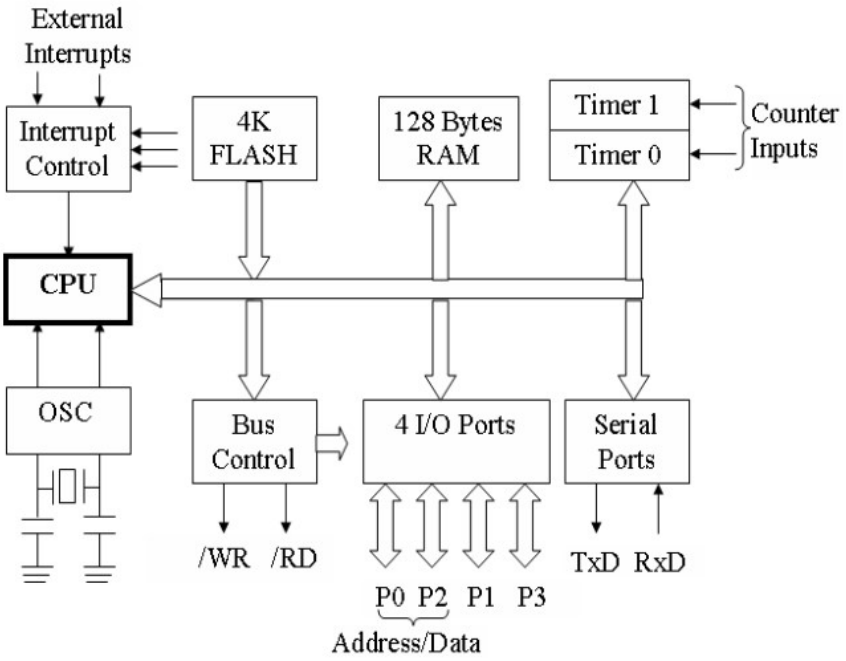
Ngoài ra, tùy theo số hiệu sản xuất mà chúng có những khác biệt về bộ nhớ và bộ định thời/bộ đếm như trong bảng 2.1 dưới đây:

Bảng 2.1: Các tham số RAM/ROM của họ 8051

Số hiệu sản xuất	Bộ nhớ chương trình trên chip	Bộ nhớ dữ liệu trên chip	Số bộ định thời (bộ đếm)
80x51	4K ROM	128 byte	2
87x51	4K EPROM	128 byte	2
89x51	4K FLASH	128 byte	2
80x32	0K	256 byte	3
80x52	8K ROM	256 byte	3
87x52	8K EPROM	256 byte	3
89x52	8K FLASH	256 byte	3

AT89C51 là một Microcomputer 8 bit, thuộc họ 8051 loại CMOS, có tốc độ cao và công suất thấp với bộ nhớ Flash có thể lập trình được. Nó được sản xuất với công nghệ bộ nhớ không bay hơi mật độ cao của hãng Atmel, và tương thích với chuẩn

công nghiệp của 8xC51 và 8xC52 về chân ra và bộ lệnh. Vì lý do đó, kể từ đây về sau ta sẽ dùng thuật ngữ "8051" (hoặc "89C51" là tương đương nhau).



Hình 2.1: Sơ đồ khối của AT89C51

2.1.2. Chức năng các chân

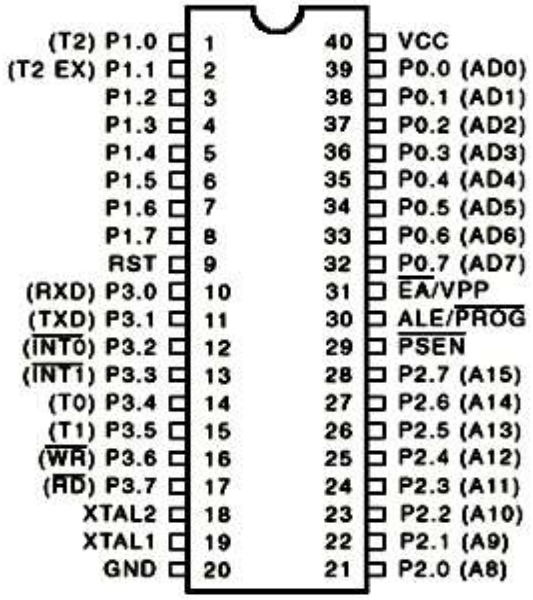
Như vậy, AT89C51 có tất cả 40 chân. Mỗi chân có chức năng như các đường I/O (xuất/nhập), trong đó 24 chân có công dụng kép: mỗi đường có thể hoạt động như một đường I/O hoặc như một đường điều khiển hoặc như thành phần của bus địa chỉ và bus dữ liệu [3]. Mô tả các chân họ 8051:

- *VCC* (chân 40): Chân cấp nguồn;
- *GND* (chân 20): Chân nối đất;
- *Port 0:* Port 0 là một port xuất/nhập song hướng cực máng hở 8 bit. Nếu được sử dụng như là một ngõ xuất thì mỗi chân có thể kéo 8 ngõ vào lên mức cao. Khi mức "1" được viết vào các chân của port 0, các chân này có thể được dùng như là các ngõ nhập tổng trở cao.

Port 0 cũng nhận các byte code (byte mã chương trình) khi lập trình Flash, và xuất ra các byte code khi kiểm tra chương trình. Cần có các điện trở pullup (5k-10k) bên ngoài khi thực hiện việc kiểm tra chương trình.

- *Port 1:* Port 1 là một port xuất/nhập song hướng 8 bit có các điện trở pullup bên trong. Các bộ đệm ngõ ra của port 1 có thể kéo hoặc cung cấp 4 ngõ nhập TTL. Khi mức "1" được viết vào các chân của port 1, chúng được kéo lên cao bởi các điện

trở pullup nội và có thể được dùng như là các ngõ nhập. Nếu đóng vai trò là các ngõ nhập, các chân của port 1 (được kéo xuống thấp qua các điện trở bên ngoài) sẽ cấp dòng I_{IL} do các điện trở pullup bên trong.



Hình 2.2: Sơ đồ chân của AT89C51

- *Port 2:* Port 2 là một port xuất/nhập song hướng 8 bit có các điện trở pullup bên trong. Các bộ đệm ngõ ra của port 2 có thể kéo hoặc cung cấp 4 ngõ vào TTL. Khi các mức "1" được viết vào các chân của port 2 thì chúng được kéo lên cao bởi các điện trở pullup nội và có thể được dùng như các ngõ vào. Khi được dùng như các ngõ vào, các chân của port 2 (được kéo xuống "0" qua các điện trở bên ngoài) sẽ cấp dòng I_{IL} do có các điện trở pullup bên trong.

- *Port 3:* Port 3 là một port xuất - nhập song hướng 8 bit có điện trở pullup nội bên trong. Các bộ đệm ngõ ra của port 3 có thể kéo hoặc cung cấp 4 ngõ vào TTL. Khi các mức "1" được viết vào các chân của port 3 thì chúng được kéo lên cao bởi các điện trở pullup nội và có thể được dùng như các ngõ vào. Khi được dùng như các ngõ vào, các chân của port 3 (được kéo xuống qua các điện trở bên ngoài) sẽ cấp dòng I_{IL} do có các điện trở pullup bên trong. Port 3 cũng cung cấp các chức năng của các đặc trưng đặc biệt như được liệt kê dưới đây:

Bảng 2.2: Các chức năng chuyển đổi trên Port 3

Chân	Tên	Các chức năng chuyển đổi
P3.0	RXD	Port nhập nối tiếp
P3.1	TXD	Port xuất nối tiếp

P3.2	$\overline{INT0}$	Ngắt 0 bên ngoài
P3.3	$\overline{INT1}$	Ngắt 1 bên ngoài
P3.4	T0	Ngõ vào Timer/Counter 0
P3.5	T1	Ngõ vào Timer/Counter 1
P3.6	$\overline{WR}$	Xung ghi bộ nhớ dữ liệu ngoài
P3.7	$\overline{RD}$	Xung đọc bộ nhớ dữ liệu ngoài

- *RST (chân 9):* Ngõ vào reset. Một mức cao trên chân này khoảng hai chu kỳ máy trong khi bộ dao động đang chạy sẽ reset toàn bộ hệ thống.

- *ALE/PROG:* ALE là một xung ngõ ra để chốt byte thấp của địa chỉ trong khi truy cập bộ nhớ ngoài. Chân này cũng là ngõ nhập xung lập trình (PROG) khi lập trình Flash.

Khi hoạt động bình thường, ALE được phát với một tỷ lệ không đổi là 1/6 tần số bộ dao động và có thể được dùng cho các mục đích timing và clocking bên ngoài. Tuy nhiên, lưu ý rằng một xung ALE sẽ bị bỏ qua mỗi khi truy cập bộ nhớ dữ liệu ngoài.

Nếu muốn hoạt động ALE có thể cấm được bằng cách set bit 0 của SFR tại địa chỉ 8Eh. Nếu bit này được set, ALE chỉ hoạt động được khi có một lệnh MOVX hoặc MOVC. Ngược lại, chân này được kéo lên cao bởi các điện trở pullup "nhẹ". Việc set bit cấm-ALE không có tác dụng khi bộ vi điều khiển đang ở chế độ thực thi bộ nhớ ngoài.

- *PSEN:* PSEN (Program Store Enable) là xung đọc bộ nhớ chương trình ngoài. Khi AT89C52 đang thi hành mã (code) từ bộ nhớ chương trình ngoài, $\overline{PSEN}$ được kích hoạt hai lần mỗi chu kỳ máy, nhưng hai hoạt động PSEN sẽ bị bỏ qua mỗi khi truy cập bộ nhớ dữ liệu ngoài.

- *EA/Vpp:* EA (External Access Enable) phải được nối với GND để cho phép thiết bị đọc code từ bộ nhớ chương trình ngoài có địa chỉ từ 0000H đến FFFFH. Tuy nhiên, lưu ý rằng nếu bit khoá “1” (lock-bit 1) được lập trình, EA sẽ được chốt bên trong khi reset.

EA phải được nối với Vcc khi thi hành chương trình bên trong. Chân này cũng nhận điện áp cho phép lập trình Vpp = 12V khi lập trình Flash (khi đó áp lập trình 12V được chọn).

- *XTAL1 và XTAL2:* XTAL1 và XTAL2 là hai ngõ vào và ra của một bộ khuếch đại dao động nghịch được cấu hình để dùng như một bộ dao động trên chip.

Không có yêu cầu nào về dao động của tín hiệu xung ngoài, vì ngõ nhập nối với mạch tạo xung nội là một flip-flop chia đôi, nhưng các chỉ định về thời gian high và low, các mức áp tối đa và tối thiểu phải được tuân theo.

2.2. Tổ chức bộ nhớ họ 8051

Họ 8051 có bộ nhớ theo cấu trúc Harvard: có những vùng nhớ riêng biệt cho chương trình và dữ liệu. Như đã nói ở trên, cả chương trình và dữ liệu có thể ở bên trong dù vậy chúng có thể được mở rộng bằng các thành phần bên ngoài lên đến tối đa 64 Kbytes bộ nhớ chương trình (ROM) và 64 Kbytes bộ nhớ dữ liệu (RAM).

Bộ nhớ bên trong bao gồm ROM và RAM trên chip, RAM trên chip bao gồm nhiều phần: phần lưu trữ đa dụng, phần lưu trữ địa chỉ hóa từng bit, các bank thanh ghi và các thanh ghi chức năng đặc biệt. Hai đặc tính cần lưu ý là:

- Các thanh ghi và các port xuất nhập đã được xếp trong bộ nhớ và có thể được truy xuất trực tiếp giống như các địa chỉ bộ nhớ khác.
- Ngăn xếp bên trong RAM nội nhỏ hơn so với RAM ngoài như trong các bộ vi xử lý khác.

2.2.1. Tổ chức bộ nhớ ROM

Nếu có nhiều EPROM hoặc nhiều RAM hoặc cả 2 giao tiếp với 8051 ta cần phải giải mã địa chỉ. Việc giải mã này cũng cần cho hầu hết các bộ vi xử xý khác.

Thí dụ nếu các RAM và ROM 8Kb được sử dụng, địa chỉ phải được giải mã để chọn các IC nhớ này trên các giới hạn 8K: 0000H – 1FFFH, 2000H – 3FFFH…

Một IC giải mã điển hình là 74HC138 được dùng với các ngõ ra được nối với các ngõ vào chọn chip /CS của các IC nhớ. Chúng được mô tả ở Hình 2.7, một bộ nhớ có EPROM 2764 (8K) và RAM 6264 (RAM). 8051 có thể quản lý không gian nhớ đến 64K bộ nhớ EPROM và 64K bộ nhớ RAM.

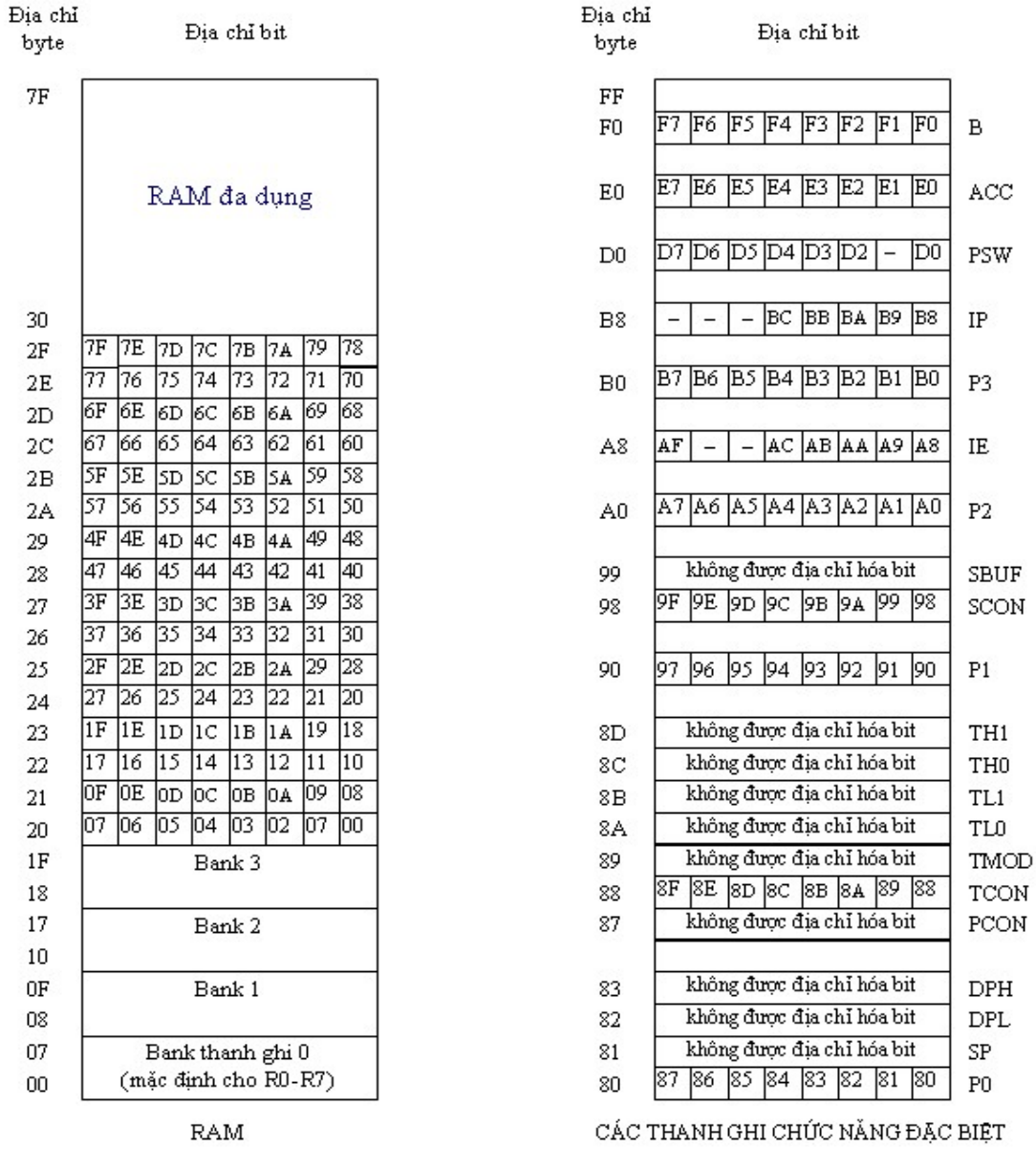
2.2.2. Tổ chức bộ nhớ RAM

Như ta thấy trên Hình 2.3, RAM bên trong 8051/8031 được phân chia giữa các bank thanh ghi (00H–1FH), RAM địa chỉ hóa từng bit (20H–2FH), RAM đa dụng (30H–7FH) và các thanh ghi chức năng đặc biệt (80H–FFH).

2.2.2.1. Vùng RAM đa dụng

Mặc dù trên hình 2.3 cho thấy 80 byte RAM đa dụng chiếm các địa chỉ từ 30H–7FH, 32 byte dưới cùng từ 00H đến 1FH cũng có thể được dùng với mục đích tương tự (mặc dù các địa chỉ này đã có mục đích khác).

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



Hình 2.3: Tóm tắt bộ nhớ dữ liệu trên chip

Ví dụ 2.1: Để đọc nội dung ở địa chỉ 5FH của RAM nội vào thanh ghi tích lũy, lệnh sau sẽ được dùng.

MOV A, 5FH

Lệnh này di chuyển một byte dữ liệu dùng cách đánh địa chỉ trực tiếp để xác định "địa chỉ nguồn" (5FH). Đích nhận dữ liệu được ngầm xác định trong mã lệnh là thanh ghi tích lũy A.

RAM bên trong cũng có thể được truy xuất dùng cách đánh địa chỉ gián tiếp qua R0 hay R1. So sánh hai lệnh sau thi hành cùng nhiệm vụ như lệnh đơn ở trên:

MOV R0, 5FH
MOV A, @R0

Lệnh đầu dùng địa chỉ tức thời để di chuyển giá trị 5FH vào thanh ghi R0, và lệnh thứ hai dùng địa chỉ trực tiếp để di chuyển dữ liệu "được trỏ bởi R0" vào thanh ghi tích lũy.

2.2.2.2. Vùng RAM địa chỉ hóa từng bit

8051/8031 chứa 210 bit được địa chỉ hóa, trong đó 128 bit là ở các địa chỉ byte 20H đến 2FH, và phần còn lại là trong các thanh ghi chức năng đặc biệt.

Ý tưởng truy xuất từng bit riêng rẽ bằng phần mềm là một đặc tính tiện lợi của vi điều khiển nói chung. Các bit có thể được đặt, xóa, AND, OR,... với một lệnh đơn. Đa số các vi xử lý đòi hỏi một chuỗi lệnh đọc-sửa-ghi để đạt được hiệu quả tương tự. Hơn nữa, các port I/O cũng được địa chỉ hóa từng bit làm đơn giản phần mềm xuất nhập từng bit.

Có 128 bit được địa chỉ hóa đa dụng ở các byte 20H đến 2FH. Các địa chỉ này được truy xuất như các byte hoặc như các bit phụ thuộc vào lệnh được dùng. Ví dụ, để bật bit 67H, ta dùng lệnh sau:

SETB 67H

Chú ý rằng "địa chỉ bit 67H" là bit có trọng số lớn nhất (MSB) ở "địa chỉ byte 2CH". Lệnh trên sẽ không tác động đến các bit khác ở địa chỉ này. Các vi xử lý khác sẽ phải thi hành nhiệm vụ tương tự như sau:

```
MOV    A, 2CH                ; Đọc cả byte
ORL    A, #10000000B         ; Set MSB
MOV    2CH,A                 ; Ghi lại cả byte
```

2.2.2.3. Các bank thanh ghi:

32 byte thấp nhất của bộ nhớ nội là dành cho các bank thanh ghi. Bộ lệnh của

8051/8031 hỗ trợ 8 thanh ghi (R0 đến R7) và theo mặc định (sau khi reset hệ thống) các thanh ghi này ở các địa chỉ 00H–07H. Lệnh sau đây sẽ đọc nội dung ở địa chỉ 05H vào thanh ghi tích lũy:

MOV A, R5

Đây là lệnh một byte dùng địa chỉ thanh ghi. Tất nhiên, thao tác tương tự có thể được thi hành bằng lệnh 2 byte dùng địa chỉ trực tiếp nằm trong byte thứ hai:

MOV A, 05H

Các lệnh dùng các thanh ghi R0 đến R7 thì sẽ ngắn hơn và nhanh hơn các lệnh tương ứng nhưng dùng địa chỉ trực tiếp. Các giá trị dữ liệu được dùng thường xuyên nên dùng một trong các thanh ghi này.

Bank thanh ghi tích cực có thể chuyển đổi bằng cách thay đổi các bit chọn bank thanh ghi trong từ trạng thái chương trình (PSW). Giả sử rằng bank thanh ghi 3 được tích cực, lệnh sau sẽ ghi nội dung của thanh ghi tích lũy vào địa chỉ 18H:

MOV R0, A

Ý tưởng dùng "các bank thanh ghi" cho phép "chuyển hướng" chương trình nhanh và hiệu quả (từng phần riêng rẽ của phần mềm sẽ có một bộ thanh ghi riêng không phụ thuộc vào các phần khác).

2.2.2.4. Các thanh ghi chức năng đặc biệt

Một vùng bộ nhớ trên chip được gọi là không gian thanh ghi chức năng đặc biệt (SFR) như được trình bày trong Hình 2.3.

Lưu ý rằng không phải tất cả các địa chỉ đều được sử dụng, và các địa chỉ không được sử dụng có thể không được cung cấp trên con chip. Các hành động đọc đến các địa chỉ này nói chung sẽ trả về các dữ liệu ngẫu nhiên, và các hành động viết sẽ có một hiệu ứng không xác định.

Các phần mềm người dùng không nên viết mức "1" đến những vị trí không được liệt kê này, vì chúng có thể được dùng trong các sản phẩm tương lai khi thêm vào các đặc trưng mới. Trong trường hợp này, các giá trị reset hoặc không tích cực của các bit mới sẽ luôn là "0".

Các thanh ghi nội của 8051/8031 được truy xuất ngầm định bởi bộ lệnh. Như lệnh "INC A" sẽ tăng nội dung của thanh ghi tích lũy A lên 1. Tác động này được ngầm định trong mã lệnh.

Các thanh ghi trong 8051/8031 được định dạng như một phần của RAM trên chip. Vì vậy mỗi thanh ghi sẽ có một địa chỉ (ngoại trừ thanh ghi đếm chương trình và thanh ghi lệnh vì các thanh ghi này hiếm khi bị tác động trực tiếp, nên không đặt chúng vào trong RAM trên chip).

Đó là lý do để 8051/8031 có nhiều thanh ghi như vậy. Cũng như R0 đến R7, có 21 thanh ghi chức năng đặc biệt (SFR: Special Function Register) ở vùng trên của RAM nội, từ địa chỉ 80H đến FFH. Chú ý rằng hầu hết 128 địa chỉ từ 80H đến FFH không được định nghĩa. Chỉ có 21 địa chỉ SFR là được định nghĩa.

Ngoại trừ thanh ghi tích lũy (A) có thể được truy xuất ngầm như đã nói, đa số các SFR được truy xuất dùng địa chỉ trực tiếp. Chú ý rằng một vài SFR có thể được địa chỉ hóa bit hoặc byte. Người thiết kế phải thận trọng khi truy xuất bit và byte. Ví dụ lệnh sau: SETB 0E0H sẽ set bit 0 trong thanh ghi tích lũy, các bit khác không đổi. Ta thấy rằng E0H đồng thời là địa chỉ byte của cả thanh ghi tích lũy và là địa chỉ bit của bit có trọng số nhỏ nhất trong thanh ghi tích lũy. Vì lệnh SETB chỉ tác động trên bit, nên chỉ có địa chỉ bit là có hiệu quả.

- Thanh ghi trạng thái chương trình: Từ trạng thái chương trình (PSW: Program Status Word) ở địa chỉ D0H chứa các bit trạng thái như Bảng 2.2.

Bảng 2.2: Thanh ghi trạng thái PSW

Bit	Ký hiệu	Địa chỉ	Ý nghĩa
PSW.7	CY	D7H	Cờ nhớ
PSW.6	AC	D6H	Cờ nhớ phụ
PSW.5	F0	D5H	Cờ 0
Bit	**Ký hiệu**	**Địa chỉ**	**Ý nghĩa**
PSW.4	RS1	D4H	Bit 1 chọn bank thanh ghi
PSW.3	RS0	D3H	Bit 0 chọn bank thanh ghi 00 = bank 0: Địa chỉ 00H–07H 01 = bank 1: Địa chỉ 08H–0FH 10 = bank 2: Địa chỉ 10H–17H 11 = bank 3: Địa chỉ 18H–1FH

PSW.2	OV	D2H	Cờ tràn
PSW.1	–	D1H	Dự trữ
PSW.0	P	D0H	Cờ parity chẵn/lẻ

Cờ nhớ: Cờ nhớ (CY) có công dụng kép. Thông thường nó được dùng cho các lệnh toán học; nó sẽ được bật lên "1" nếu có một số nhớ sinh ra bởi phép cộng hoặc có một số mượn bởi phép trừ. Chẳng hạn, nếu thanh ghi tích lũy chứa FFH, thì lệnh ADD A, #1 sẽ trả về thanh ghi tích lũy kết quả 00H và bật cờ nhớ trong PSW.

Cờ nhớ cũng có thể xem như một thanh ghi 1 bit cho các lệnh luận lý thi hành trên bit. Chẳng hạn, lệnh sau sẽ AND bit 25H với cờ nhớ và đặt kết quả trở vào cờ nhớ: ANL C, 25H

Cờ nhớ phụ: Khi cộng các số BCD, cờ nhớ phụ (AC) được bật lên "1" nếu kết quả của 4 bit thấp trong khoảng 0AH đến 0FH. Nếu các giá trị được cộng là số BCD, thì sau lệnh cộng cần có DA A (hiệu chỉnh thập phân thanh ghi tích lũy) để mang kết quả lớn hơn 9 vào nibble cao.

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

- Các bit chọn bank thanh ghi: Các bit chọn bank thanh ghi (RS0 và RS1) xác định bank thanh ghi được tích cực. Chúng được xóa sau khi reset hệ thống và được thay đổi bằng phần mềm nếu cần.

Khi chương trình được hợp dịch, các địa chỉ bit được thay thế cho các ký hiệu "RS1" và "RS0". Vậy, lệnh SETB RS1 sẽ giống như lệnh SETB 0D4H.

Cờ tràn: Cờ tràn (OV) được bật lên "1" sau một lệnh cộng hoặc trừ nếu có một phép toán bị tràn. 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 dấu được cộng, bit OV có thể được bỏ qua. Các kết quả lớn hơn +127 hoặc nhỏ hơn -128 sẽ bật bit OV.

Kết quả là một số có dấu 8EH được xem như -116, không phải là kết quả đúng (142), vì vậy, bit OV được bật.

- Thanh ghi B: Thanh ghi B ở địa chỉ F0H được dùng cùng với thanh ghi tích lũy A cho các phép toán nhân và chia. Lệnh MUL AB sẽ nhân các giá trị không dấu 8 bit trong A và B rồi trả về kết quả 16 bit trong A (byte thấp) và B (byte cao). Lệnh DIV AB sẽ chia A cho B rồi trả về kết quả nguyên trong A và phần dư trong B. Thanh

ghi B cũng có thể được xem như thanh ghi đệm đa dụng. Nó được địa chỉ hóa từng bit bằng các địa chi bit F0H đến F7H.

- Con trỏ ngăn xếp Stack pointer: Con trỏ ngăn xếp (SP) là một thanh ghi 8 bit ở địa chỉ 81H. Nó chứa địa chỉ của byte dữ liệu hiện hành trên đỉnh của ngăn xếp. Các lệnh trên ngăn xếp bao gồm các thao tác cất dữ liệu vào ngăn xếp và lấy dữ liệu ra khỏi ngăn xếp. Lệnh cất dữ liệu vào ngăn xếp sẽ làm tăng SP trước khi ghi dữ liệu, và lệnh lấy dữ liệu ra khỏi ngăn xếp sẽ đọc dữ liệu và giảm SP. Ngăn xếp của 8051/8031 được giữ trong RAM nội và được giới hạn các địa chỉ có thể truy xuất bằng địa chỉ gián tiếp.

Để khởi động lại SP với ngăn xếp bắt đầu tại 60H, các lệnh sau đây được dùng:

```
MOV    SP, #5FH
```

Trên 8051/8031 ngăn xếp bị giới hạn 32 byte vì địa chỉ cao nhất của RAM trên chip là 7FH. Sở dĩ dùng giá trị 5FH vì SP sẽ tăng lên 60H trước khi cất byte dữ liệu đầu tiên.

Người thiết kế có thể chọn không phải khởi động lại con trỏ ngăn xếp mà để nó lấy giá trị mặc định khi reset hệ thống. Giá trị mặc định đó là 07H và kết quả là ngăn đầu tiên để cất dữ liệu có địa chỉ là 08H. Nếu phần mềm ứng dụng không khởi động lại SP, bank thanh ghi 1 (có thể cả 2 và 3) sẽ không dùng được vì vùng RAM này đã được dùng làm ngăn xếp.

Ngăn xếp được truy xuất trực tiếp bằng các lệnh PUSH và POP để lưu trữ tạm thời và lấy lại dữ liệu, hoặc được truy xuất ngầm bằng các lệnh gọi chương trình con (ACALL, LCALL) và các lệnh trở về (RET, RETI) để cất và lấy lại bộ đếm chương trình.

- Con trỏ dữ liệu Data pointer: Con trỏ dữ liệu (DPTR) được dùng để truy xuất bộ nhớ ngoài là một thanh ghi 16 bit ở địa chỉ 82H (DPL: byte thấp) và 83H (DPH: byte cao). Ba lệnh sau sẽ ghi 55H vào RAM ngoài ở địa chỉ 1000H:

```
MOV    A, #55H
MOV    DPTR, #1000H
MOVX  @DPTR, A
```

Lệnh đầu tiên dùng địa chỉ tức thời để tải dữ liệu 55H vào thanh ghi tích lũy. Lệnh thứ hai cũng dùng địa chỉ tức thời, lần này để tải dữ liệu 16 bit 1000H vào con trỏ dữ liệu. Lệnh thứ ba dùng địa chỉ gián tiếp để di chuyển dữ liệu trong A (55H) đến RAM ngoài ở địa chỉ được chứa trong DPTR (1000H).

- Các thanh ghi port xuất nhập: Các port của 8051/8031 bao gồm Port 0 ở địa chỉ 80H, Port 1 ở địa chỉ 90H, Port 2 ở địa chỉ A0H và Port 3 ở địa chỉ B0H. Tất cả các port đều được địa chỉ hóa từng bit. Điều đó cung cấp một khả năng giao tiếp thuận lợi. Chẳng hạn nếu một motor được nối qua một cuộn dây có transistor lái đến bit 7 của Port 1, nó có thể được bật và tắt bằng một lệnh đơn:

SETB P1.7 ; bật motor

CLR P1.7 ; tắt motor

Các lệnh trên dùng dấu chấm để xác định một bit trong một byte. Trình hợp dịch sẽ thi hành sự chuyển đổi cần thiết, vì vậy hai lệnh sau đây là như nhau:

CLR P1.7

CLR 97H

Trong một trường hợp khác, xem xét giao tiếp đến một thiết bị với một bit trạng thái gọi là BUSY, được bật khi thiết bị đang bận và được xóa khi thiết bị đã sẵn sàng. Nếu BUSY được nối tới P1.5, vòng lặp sau sẽ được dùng để chờ thiết bị trở lại trạng thái sẵn sàng:

WAIT: JB P1.5, WAIT

Lệnh này có nghĩa là "nếu bit P1.5 được bật thì nhảy tới nhãn WAIT". Nói cách khác "nhảy trở lại và kiểm tra lần nữa".

- Các thanh ghi timer: 8051/8031 chứa hai bộ định thời/đếm (timer/counter) 16 bit được dùng cho việc định thời hoặc đếm sự kiện. Timer 0 ở địa chỉ 8AH (TL0: byte thấp) và 8CH (TH0: byte cao). Timer 1 ở địa chỉ 8BH (TL1: byte thấp) và 8DH (TH1: byte cao). Việc vận hành timer được thiết lập bởi thanh ghi Timer Mode (TMOD) ở địa chỉ 89H và thanh ghi điều khiển timer (TCON) ở địa chỉ 88H. Chỉ có TCON được địa chỉ hóa từng bit.

- Các thanh ghi port nối tiếp:

8051/8031 chứa một port nối tiếp trên chip dành cho việc trao đổi thông tin với các thiết bị nối tiếp như máy tính, modem hoặc cho việc giao tiếp với các IC khác có giao tiếp nối tiếp (các bộ chuyển đổi A/D, các thanh ghi dịch...). Một thanh ghi gọi là bộ đệm dữ liệu nối tiếp (SBUF) ở địa chỉ 99H sẽ giữ cả hai dữ liệu truyền và nhận. Khi truyền dữ liệu thì ghi lên SBUF, khi nhận dữ liệu thì đọc SBUF. Các chế độ vận hành khác nhau được lập trình qua thanh ghi điều khiển port nối tiếp (SCON) (được địa chỉ hóa từng bit) ở địa chỉ 98H.

- Các thanh ghi ngắt: 8051/8031 có cấu trúc 5 nguồn ngắt, 2 mức ưu tiên. Các ngắt bị cấm sau khi reset hệ thống và sẽ được cho phép bằng việc ghi lên thanh ghi cho phép ngắt (IE) ở địa chỉ A8H. Cả hai thanh ghi được địa chỉ hóa từng bit.

- Thanh ghi điều khiển công suất nguồn: Thanh ghi điều khiển công suất (PCON) ở địa chỉ 87H chứa nhiều bit điều khiển. Chúng được tóm tắt trong Bảng 2.3 sau:

Bảng 2.3: Chức năng các bit trong thanh ghi PCON

Bit	Ký hiệu	Ý nghĩa
7	SMOD	Nếu được SET tốc độ Baud sẽ tăng gấp đôi chế độ 1,2 và 3 của port nối tiếp.
6	-	Non
5	-	Non
4	-	Non
3	GF1	Cờ đa dụng 1
2	GF0	Cờ đa dụng 0
1	PD	SET kích hoạt giảm công suất nguồn, thoát khi Reset.
0	IDL	SET kích hoạt chế độ chờ, thoát khi có ngắt hay Reset hệ thống

- Bảo vệ bộ nhớ: Các bit khoá bộ nhớ chương trình. Vi điều khiển 89C51 có 3 bit khoá có thể bỏ không lập trình (U) hoặc được lập trình (P) để nhận các đặc trưng thêm vào được liệt kê trong bảng 2.4. dưới đây (với LB1, LB2, LB3 là các bit khóa tương ứng).

Bảng 2.4: Thanh ghi bảo vệ bộ nhớ

Chế độ	LB1	LB2	LB3	Kiểu bảo vệ
1	U	U	U	Không khoá chương trình
2	P	U	U	Các lệnh MOVC được thi hành từ bộ nhớ chương trình ngoài bị cấm khi lấy các byte mã từ bộ nhớ nội, /EA được lấy mẫu và được chốt lại khi reset và hơn nữa, việc lập trình bộ nhớ Flash là bị cấm.
3	P	P	U	Như chế độ 2 nhưng việc kiểm tra cũng bị cấm

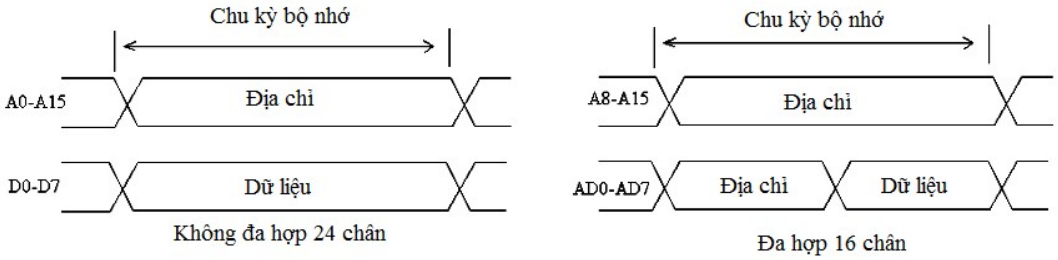
4	P	P	P	Như chế độ 3 nhưng việc thi hành ngoài cũng bị cấm.

Khi bit khoá "1" được lập trình, mức logic tại chân $\overline{EA}$ được lấy mẫu và chốt lại khi reset. Nếu thiết bị được bật nguồn mà không có reset, việc chốt sẽ được khởi tạo với một giá trị ngẫu nhiên cho đến khi được reset. Giá trị được chốt của $\overline{EA}$ phải bằng với mức logic hiện tại ở chân đó để cho thiết bị làm việc một cách chính xác.

2.2.3. Bộ nhớ ngoài

Các bộ vi điều khiển cần có khả năng mở rộng các tài nguyên trên chip MSC-51 có khả năng cho ta mở rộng bộ nhớ chương trình đến 64K và không gian bộ nhớ dữ liệu 64K. ROM và RAM ngoài được thêm vào khi cần.

Các IC giao tiếp ngoại vi cũng có thể được thêm vào để mở rộng khả năng xuất nhập chúng trở thành một phần của không gian bộ nhớ dữ liệu ngoài bằng cách sử dụng cách định địa chỉ kiểu I/O ánh xạ bộ nhớ. Khi bộ nhớ ngoài được sử dụng port 0 không làm nhiệm vụ của port xuất nhập mà port này trở thành port địa chỉ (A0- A7) và bus dữ liệu (D0 – D7) đa hợp. Ngõ ra ALE chốt byte thấp của địa chỉ ở thời điểm bắt đầu mỗi một chu kỳ bộ nhớ ngoài. Port 2 thường làm byte cao của bus địa chỉ. Một ý tưởng tổng quát được trình bày ở Hình 2.4.

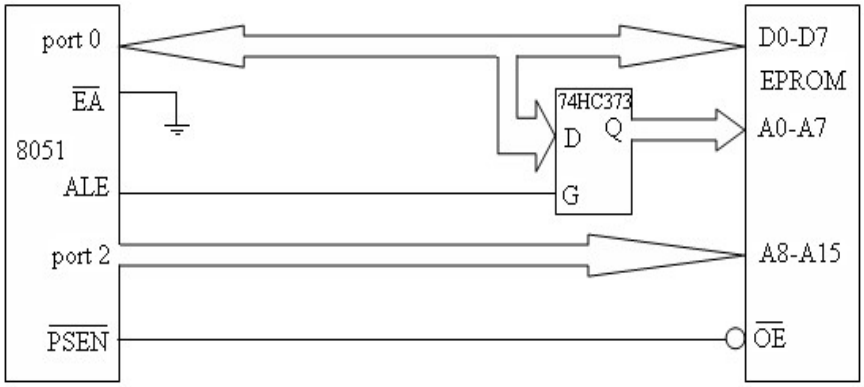


Hình 2.4: Đa hợp Bus địa chỉ (byte thấp) và bus dữ liệu đa hợp và không đa hợp

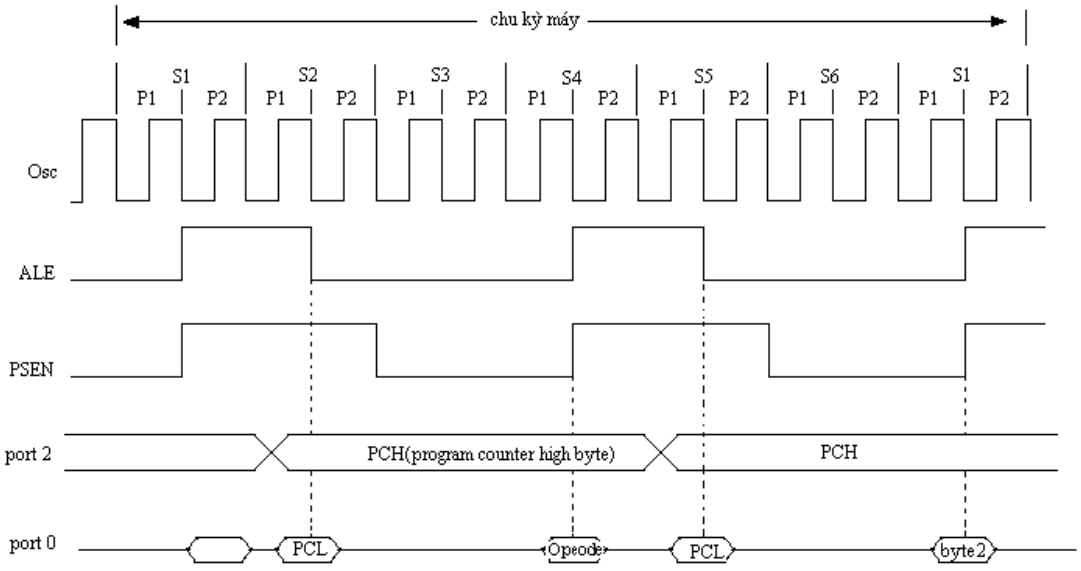
Sắp xếp không đa hợp sử dụng 16 đường địa chỉ và 8 đường dữ liệu tổng cộng 24 đường. Sắp xếp đa hợp 8 đường port 0 của bus dữ liệu và byte thấp của bus địa chỉ, do vậy ta chỉ cần 16 đường.

Sắp xếp đa hợp có hoạt động như sau: trong ½ chu kỳ đầu của chu kỳ bộ nhớ, byte thấp của địa chỉ được cung cấp bởi port 0 và được chốt nhờ tín hiệu ALE. Mạch chốt 74HC373 giữ cho byte thấp của địa chỉ ổn định trong cả chu kỳ bộ nhớ. Trong ½ của chu kỳ bộ nhớ port 0 được sử dụng làm bus dữ liệu và dữ liệu được đọc hay ghi.

2.2.3.1. Truy xuất bộ nhớ chương trình ngoài



Hình 2.5: Truy xuất bộ nhớ chương trình ngoài



Hình 2.6: Giản đồ thời gian của chu kỳ tìm nạp ở bộ nhớ ngoài

Bộ nhớ chương trình ngoài là bộ nhớ chỉ đọc được cho phép bởi tín hiệu /PSEN. Khi có một EPROM ngoài được sử dụng cả hai port 0 và port 2 đều không còn là các port xuất nhập. Kết nối phần cứng với bộ nhớ ngoài EPROM được trình bày ở Hình 2.5.

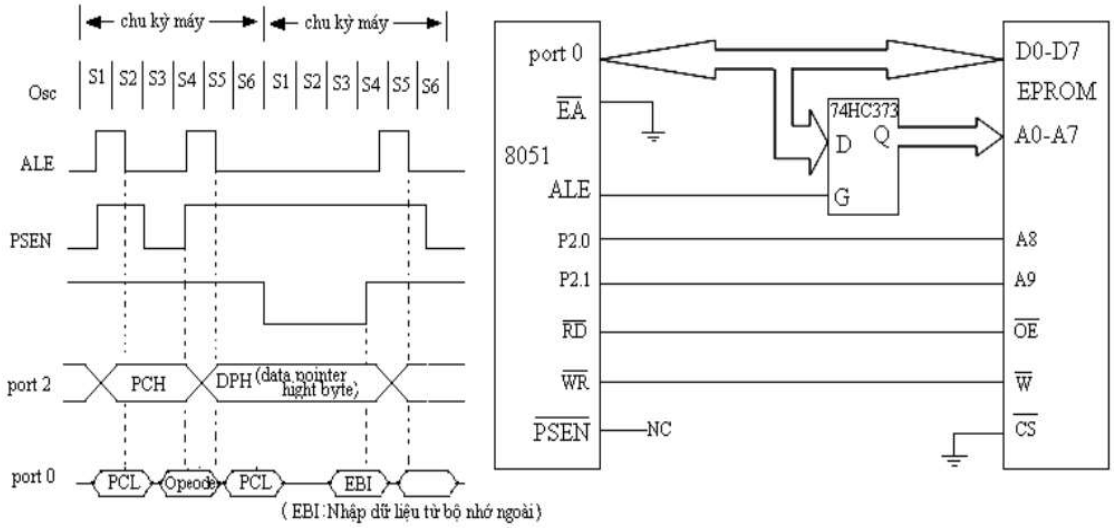
Chu kỳ máy của 8051 có 12 chu kỳ dao động. nếu bộ dao động trên chip có tần số 12MHz, một chu kỳ máy dài 1 μs. Trong một chu kỳ máy điển hình, ALE có 2 xung và 2 byte của lệnh được đọc từ bộ nhớ chương trình. Giản đồ thời gian của chu kỳ máy này được gọi là chu kỳ tìm nạp lệnh được trình bày ở Hình 2.6.

2.2.3.2. Truy xuất bộ nhớ dữ liệu ngoài

Bộ nhớ dữ liệu ngoài là bộ nhớ đọc/ ghi được cho phép bởi các tín hiệu /RD và /WR ở các chân P3.7 và P3.6. Lệnh dùng để truy xuất bộ nhớ dữ liệu ngoài là MOVX, sử dụng con trỏ dữ liệu 16 bit DPTR hoặc R0, R1 làm thanh ghi chứa địa chỉ.

RAM có thể giao tiếp với 89C51 theo cùng cách như EPROM ngoại trừ đường /RD nối với đường chéo xuất /OE của RAM và /WR nối với đường ghi W của RAM. Các kết nối của bus dữ liệu và bus địa chỉ giống như EPROM. Bằng cách sử dụng các port 0 và port 2 như ở phần trên, ta có một dung lượng RAM ngoài lên đến 64K được kết nối với 8051.

Giản đồ thời gian của thao tác "*đọc"* dữ liệu của bộ nhớ dữ liệu ngoài được trình bày ở Hình 2.6. Sử dụng lệnh MOVX A, @DPTR, lưu ý là cả 2 xung ALE và /PSEN được bỏ qua ở nơi mà xung /RD cho phép đọc RAM (nếu lệnh MOVX và RAM ngoài không bao giờ được dùng, các xung ALE luôn có tần số bằng 1/6 tần số của mạch dao động).



Hình 2.7: Giản đồ thời gian của lệnh MOVX, giao tiếp với 1K RAM

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

Port 2 giảm bớt được chức năng làm nhiệm vụ cung cấp byte cao của địa chỉ trong các hệ thống tối thiểu thành phần, hệ thống không dùng bộ nhớ chương trình ngoài và chỉ có một dung lượng nhỏ bộ nhớ dữ liệu ngoài. Các địa chỉ 8 bit có thể

truy xuất bộ nhớ dữ liệu ngoài với cấu hình bộ nhớ nhỏ hướng trang (page – oriented). Nếu có nhiều hơn một trang 64 byte RAM, một vài bit từ port 2 (hoặc một port khác) có thể chọn một trang. Như RAM 1 Kb (nghĩa là 4 trang 256 byte) ta có thể kết nối RAM này với 8051 như ở Hình 2.7.

Các bit 0 và 1 của port 2 phải được khởi động để chọn một trang, rồi lệnh MOVX được dùng để đọc hoặc ghi trên trang này.

Ví dụ 2.2: Giả sử P2.0 = P2.1 = 0, các lệnh sau có thể dùng đọc các nội dung của RAM ngoài ở địa chỉ 0050H vào thanh chứa A.

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

Để đọc ở địa chỉ cuối cùng của RAM này, 03FFH, trang 3 được chọn nghĩa là ta phải bật cho các bit P2.0 và P2.1 bằng một chuỗi lệnh sau được dùng:

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

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

2.3. Tập lệnh họ 8051

Tập lệnh họ MSC-51 được sự kiểm tra của các mode định vị và các lệnh của chúng có các Opcode 8 bit. Điều này cung cấp khả năng 2^8= 256 lệnh được thi hành và một lệnh không được định nghĩa. Vài lệnh có 1 hoặc 2 byte bởi dữ liệu hoặc địa chỉ thêm vào Opcode. Trong toàn bộ các lệnh có 139 lệnh một byte, 92 lệnh 2 byte và 24 lệnh 3 byte.

2.3.1. Quy ước đánh địa chỉ

Các chế độ định vị là một bộ phận thống nhất của tập lệnh. Chúng cho phép định rõ nguồn hoặc nơi gởi tới của dữ liệu ở các đường khác nhau tùy thuộc vào trạng thái của người lập trình. Họ 8051 có 8 chế độ định vị được dùng như sau:

Thanh ghi; Trực tiếp; Gián tiếp; Tức thời; Tương đối; Tuyệt đối; Dài; Định vị.

2.3.2. Nhóm lệnh di chuyển dữ liệu

Các lệnh dịch chuyển dữ liệu trong những vùng nhớ nội thực thi 1 hoặc 2 chu

kỳ máy. Mẫu lệnh MOV <destination>, <source> cho phép di chuyển dữ liệu bất kỳ 2 vùng nhớ nào của RAM nội hoặc các vùng nhớ của các thanh ghi chức năng đặc biệt mà không thông qua thanh ghi A.

MOV <đích>, <nguồn>

Vùng ngăn xếp của 8051 chỉ chứa 128 byte RAM nội, nếu con trỏ Ngăn xếp SP được tăng quá địa chỉ 7FH thì các byte được PUSH vào sẽ mất đi và các byte POP ra thì không biết rõ.

Các lệnh dịch chuyển bộ nhớ nội và bộ nhớ ngoại dùng sự định vị gián tiếp. Địa chỉ gián tiếp có thể dùng địa chỉ 1 byte (@ Ri) hoặc địa chỉ 2 byte (@ DPTR). Tất cả các lệnh dịch chuyển hoạt động trên toàn bộ nhớ ngoài thực thi trong 2 chu kỳ máy và dùng thanh ghi A làm toán hạng đích (DESTINATION).

Việc đọc và ghi RAM ngoài (RD và WR) chỉ tích cực trong suốt quá trình thực thi của lệnh MOVX, còn bình thường RD và WR không tích cực (mức 1).

Các bảng tìm kiếm:

Có 2 lệnh di chuyển dữ liệu dành cho việc đọc các bảng tìm kiếm trong bộ nhớ chương trình. Do bởi các lệnh này truy xuất bộ nhớ chương trình, các bảng tìm kiếm chỉ có thể được đọc và không được cập nhập. Mã gợi nhớ của lệnh là MOVC (move constan). MOVC sử dụng bộ đếm chương trình hoặc con trỏ dữ liệu làm thanh ghi nền và thanh chứa A chứa địa chỉ offset. Lệnh sau:

MOVC A, @A+DPTR

Có thể truy xuất một bảng 256 điểm nhập được đánh số từ 0 ÷ 255, số của điểm nhập yêu cầu được nạp cho thanh chứa A và con trỏ dữ liệu được khởi động để chứa địa chỉ đầu bảng, lệnh sau:

MOVC A, @A+PC

Cũng hoạt động tương tự, ngoại trừ ở đây bộ đếm chương trình được dùng để chứa địa chỉ nền và bảng được truy xuất nhờ 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, ....
```

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

2.3.3. Nhóm lệnh toán học

Các lệnh số học cũng được phân loại như dưới. Do có thể có 4 khả năng định địa chỉ, lệnh ADD A còn được viết dưới dạng.

ADD A, <src, byte>; cộng giá trị byte với nguồn src kết quả cất vào nguồn

SUBB A, <src, byte>; trừ giá trị nguồn src cho byte, kết quả cất vào nguồn

Ví dụ 2.3: Phép toán cộng

```
mov     a, #38h; Số thập phân của 38h là 56 và nhị phân là 0011 1000b
Add     a, #2fh ; Số thập phân 2f là 47 và nhị phân là 0010 1111b
38h+6fh =       67h     0011 1000b
                        + 0010 1111b
                        = 0110 0111b
```

```
INC <byte>    ; tăng thêm 1 đơn vị
INC   A       ; (A) (A) + 1
DEC <byte>    ; giảm đi 1 đơn vị
DEC   A       ; (A) (A) - 1
MULL AB       ; (A) LOW [(A) x (B)], có ảnh hưởng cờ OV
              ; (B) HIGH [(A) x (B)], cờ Cary được xóa.
DIV   AB      ; (A) Integer Result of [(A)/(B)], cờ OV
              ; (B) Remainder of [(A)/(B)], cờ Carry xóa
DA    A       ; Điều chỉnh thanh ghi A thành số BCD đúng trong phép
              ; cộng BCD (thường DA A đi kèm với ADD, ADDC)
```

Nếu 4 bit thấp (cao) có số lớn hơn 9 (không còn số BCD), lúc này nó được cộng thêm 6 (0110) vào 4 bit thấp (cao) và cờ phụ AC (cờ chính CY) sẽ bật lên "1".

Kết quả số thập phân đó là lấy trong r5, r6, r7.

- Nếu [(A3-A0)>9] và [(AC) = 1] (A3A0) (A3A0) + 6.

- Nếu [(A7-A4)>9] và [(C) = 1] (A7A4) (A7A4) + 6.

Tất cả các lệnh số học được thực thi trong một chu kỳ máy ngoại trừ lệnh INC DPTR được thực thi trong hai chu kỳ máy, các lệnh MUL AB và DIV thực thi trong 4 chu kỳ máy (lưu ý là 1 chu kỳ máy dài 1 μsec nếu 8051 hoạt động với xung clock 12 MHz).

Vi điều khiển 8051 cung cấp kiểu định địa chỉ linh hoạt cho không gian nhớ nội. Một vị trí bất kỳ đều có thể tăng hay giảm bằng cách dùng kiểu định địa chỉ trực tiếp mà không cần qua trung gian thanh chứa A. Lấy ví dụ nếu vị trí 7FH của RAM nội chứa giá trị 40H, lệnh:

```
INC    7FH
```

Tăng giá trị tại địa chỉ 7FH thành 41H và đặt kết quả tại 7FH, 8051 cũng có lệnh INC thao tác trên con trỏ dữ liệu 16 bit. Do con trỏ dữ liệu chứa 16 bit địa chỉ của bộ nhớ ngoài, việc tăng nội dung con trỏ này là một đặc trưng thường sử dụng. Tập lệnh 8051 không có lệnh giảm nội dung con trỏ dữ liệu mà ta phải dùng đến 1 chuỗi lệnh sau:

```
DEC    DPL                ; Giảm byte thấp của DPTR
MOV    R7, DPL            ; Cất vào R7
CJNE   R7, #0FFH, SKIP      ; So sánh với 0FFH
DEC    DPH                 ; Nếu bằng, giảm byte cao của DPTR
SKIP: ....                 ; Không bằng, bỏ qua
```

Các byte thấp và byte cao của DPTR phải được giảm riêng rẽ trong đó byte cao (DPH) chỉ được giảm 1 nếu byte thấp (DPL) tràn từ 00H qua FFH.

Cách cộng nhiều byte: khi có nhiều byte dữ liệu trong bộ nhớ SRAM, như địa chỉ 40h (chứa 7dh), 41h (chứa 0ebh), 42h (chứa 0c5h), 43h (chứa 5bh), 44h (chứa 30h) với đoạn lệnh sau:

```
Mov r0, #40h   ; Xác lập mã địa chỉ 40h của bộ nhớ RAM dùng thanh ghi R0
Mov r2, #5     ; Tạo số lần thực hiện phép cộng (5 lần)
Clr    a             ; Xóa sạch thanh ghi a
Mov    r7, a         ; Cất kết quả vào thanh ghi tạm r7
Lai:   add a, @r0    ; Cộng a với nội dung có ở địa chỉ trỏ bởi thanh ghi r0
   Jnc   ttuc        ; Nếu cờ CY còn là "0" thì nhảy đến nhãn ttuc
```

Inc r7 *; Nếu cờ CY = "1" thì tăng tăng r7 lên 1*

Ttuc: inc r0 *; Tăng r0 lên 1, chuyển qua địa chỉ tiếp theo*

Djnz r2, llai *; Nếu r2 giảm chưa = "0" thì nhảy đến nhãn llai*

Kết quả 5 lần cộng, kết quả byte thấp cho giữ trong thanh a kết quả byte cao lấy theo CY sẽ cất trong thanh ghi R7.

Các phép toán số học cho các con số có dấu +/-. Người ta dùng 8 bit với bit cao nhất MSB để làm bit dấu, 7 bit còn lại để ghi độ lớn dữ liệu. Bit D7 = "0" biểu thị dấu "+" D7 = "1" biểu thị dấu "–" như vậy dữ liệu lúc này + 127 đến - 127. Để biểu thị một số âm (-) có dấu D7 = "0" lúc này phần độ lớn được biểu thị bằng số lấy bù 2 của nó, việc chuyển phần độ lớn ra số bù 2 sẽ do trình biên dịch đảm nhiệm và theo trình tự sau:

Viết con số có dấu dạng 8 bit nhị phân đảo ngược các bit này và công (+) 1 thêm vào ta được số bù 2 của số đó.

Ví dụ 2.4: Biểu diễn con số -34h có số nhị phân là 00110100b lấy đảo lên là 1100 1011b + 1 = 1100 1100b hay cch. Chẳng hạn cộng 2 số -2 và -5:

Mov a, #-2 *; -2 là feh = 1111 1110b*

Mov r4, #-5 *; -5 là fbh = 1111 1001b*

Add a, r2 *; Kết quả f9h = 1111 1001b, OV = "0"*

Hoặc mov a, # +96 *; +96 là 60h = 0110 0000b*

Mov r1, #+70 *; +70 là 46h = 0010 0110b*

Add a,r1 *; Kết quả -90 là a6h = 1010 0110b, OV = "1"*

Kiểm tra cờ tràn OV bằng "0" là kết quả đúng, bằng "1" là kết quả sai.

2.3.4. Nhóm lệnh nhảy

Có nhiều lệnh để điều khiển lên chương trình bao gồm việc gọi hoặc trả lại từ chương trình con hoặc chia nhánh có điều kiện hay không có điều kiện.

Tất cả các lệnh rẽ nhánh đều không ảnh hưởng đến cờ. Ta có thể định nhãn cần nhảy tới mà không cần rõ địa chỉ, trình biên dịch sẽ đặt địa chỉ nơi cần nhảy tới vào đúng khẩu lệnh đã đưa ra. Có 3 biến thể của lệnh nhảy: SJMP, LJMP và AJMP (địa chỉ tương đối, dài và tuyệt đối).

Lệnh JMP @A+DPRT hỗ trợ các thao tác nhảy phụ thuộc vào trường hợp cụ thể cho các bảng nhảy. Địa chỉ đích được tính ở thời điểm thực thi lệnh là tổng của nội dung thanh ghi DPRT 16 bit với nội dung của thanh A. DPRT được nạp địa chỉ

của bảng nhảy và thanh chứa A đóng vai trò của một thanh ghi chỉ số.

Ví dụ 2.5: Nếu có 5 yêu cầu, 1 giá trị từ 0 đến 4 được nạp cho thanh chứa A và 1 nhảy trường hợp tương thích được thực hiện như sau:

```
Mov     dprt, #jump_table
Mov     a, index_number
Rl      a
Jmp     @a+dprt
```

8051 cung cấp cho ta nhiều lệnh nhảy có điều kiện. Tất cả các lệnh này xác định địa chỉ đích bằng kiểu định địa chỉ tương đối và cùng bị giới hạn ở khoảng cách nhảy từ -128 byte đến +127 byte kể từ lệnh theo sau lệnh nhảy có điều kiện. Tuy nhiên cần lưu ý là người lập trình sẽ xác định địa chỉ đích theo cung cách với các lệnh nhảy khác bằng cách dùng nhãn hay dùng số 16 bit. Trình dịch hợp ngữ sẽ thực hiện các việc còn lại. Các lệnh JZ và JNZ kiểm tra dữ liệu trong thanh chứa cho điều kiện này.

Lệnh DJNZ (giảm và nhảy nếu khác 0) dành cho điều khiển vòng lặp. Để thực thi một vòng lặp n lần, ta nạp một byte số đếm N cho một thanh ghi và kết thúc vòng lặp với DJNZ trỏ tới điểm bắt đầu vòng lặp, như N = 10:

```
MOV    R7, #10
LOOP   :        ; (Bắt đầu vòng lặp)
   ...
```

Kết thúc vòng lặp:

```
DJNZ   R7, LOOP
```

(tiếp tục)

Lệnh CJNE (so sánh và nhảy nếu không bằng) cũng dành cho việc điều khiển vòng lặp. Hai byte được xác định trong trường toán hạng của lệnh và việc nhảy chỉ được thực thi nếu 2 byte khác 0. Giả sử nếu một ký tự vừa được đọc vào thanh chứa A từ port nối tiếp và ta muốn nhảy đến một lệnh được nhận biết bởi nhãn TERMINATE nếu ký tự là "C" (03H), các dòng lệnh sau được sử dụng:

```
CJNE    A, #03H, SKIP
SJMP    TERMINATE
SKIP: ....
```

Vì thao tác nhảy chỉ xuất hiện nếu thanh chứa A chứa mã của "C", một nhãn SKIP được dùng để bỏ qua việc kết thúc lệnh nhảy ngoại trừ khi mã yêu cầu được

đọc. Lệnh trên còn được ứng dụng trong các phép so sánh lớn hơn hay nhỏ hơn 2 byte trong trường toán hạng là các số nguyên không dấu. Nếu byte nhỏ hơn byte thứ 2 cờ nhớ được SET lên 1. Nếu byte lớn hơn hoặc bằng byte thứ 2, cờ nhớ được xóa. Lấy ví dụ nếu ta muốn nhảy đến BIG trong khi giá trị trong thanh chứa A lớn hơn hoặc bằng 20H, các chỉ thị sau được dùng:

```
CJNE    A, #20H, $+3
JNC     BIG
```

Đích nhảy cho lệnh CJNE được xác định là $+3, dấu $ là một ký hiệu của trình dịch hợp ngữ biểu thị địa chỉ của lệnh hiện hành. Vì CJNE là lệnh 3 byte, $+3 là địa chỉ của lệnh tiếp theo, JNC. Mặt khác, lệnh CJNE theo sau bởi lệnh JNC không quan tâm đến kết quả so sánh. Mục đích duy nhất của việc so sánh là để bật hay xóa cờ nhớ và lệnh JNC quyết định nhảy hay không nhảy. Giả định này thể hiện trong đó 8051 tiếp cận với tình huống lập trình tổng quát một cách vụng về hơn so với hầu hết các bộ vi xử lý. Tuy nhiên, việc sử dụng "macro" cho phép ta có các chuỗi lệnh mạnh, chẳng hạn được cấu trúc và thực thi bằng cách dùng một mã gợi nhớ duy nhất.

Sau đây là sự tóm tắt từng hoạt động của lệnh nhảy.

```
JC      rel             ; Nhảy đến "rel" nếu cờ Carry C = 1.
JNC     rel             ; Nhảy đến "rel" nếu cờ Carry C = 0.
JB      bit, rel        ; Nhảy đến "rel" nếu (bit) = 1.
JNB     bit, rel        ; Nhảy đến "rel" nếu (bit) = 0.
JBC     bit, rel        ; Nhảy đến "rel" nếu bit = 1 và xóa bit.
ACALL   addr11          ; Lệnh gọi tuyệt đối trong page 2K;
LCALL   addr16          ; Lệnh gọi dài chương trình con trong 64K.
RET                     ; Kết thúc chương trình con trở về  chương trình chính.
RETI            ; Kết thúc thủ tục phục vụ ngắt quay về chương trình chính
AJMP    Addr11          ; Nhảy tuyệt đối không điều kiện trong 2K.
LJMP    Addr16 ; Nhảy dài không điều kiện trong 64K
SJMP    rel     ; Nhảy ngắn không điều kiện trong  (-128127) byte
JMP     @ A + DPTR ; Nhảy không điều kiện đến địa chỉ (A) + (DPTR)
JZ      rel     ; Nhảy đến A = 0. Thực hành lệnh kế nếu A khác 0.
JNZ     rel             ; Nhảy đến A 0. Thực hành lệnh kế nếu A = 0.
```

CJNE A, direct, rel ; So sánh và nhảy đến A direct thực hành lệnh kế tiếp:

DJNE Rn, rel ; Giảm Rn và nhảy nếu Rn 0.

DJNZ direct, rel ; Tương tự lệnh DJNZ Rn, rel.

Ví dụ 2.6: Sử dụng lệnh rẽ nhánh để tạo delay:

```
Delay:
Mov r5, #10
Vong1: mov      r6, #50
   Djnz r6,$
   Djnz r5, vong1
   Ret
```

2.3.5. Nhóm lệnh logic

Tất cả các lệnh logic sử dụng thanh ghi A như là một trong những toán hạng thực thi một chu kỳ máy, ngoài A ra mất 2 chu kỳ máy. Những hoạt động logic có thể được thực hiện trên bất kỳ byte nào trong vị trí nhớ dữ liệu nội mà không qua thanh ghi A.

Bao gồm các phép toán logic AND, OR, XOR, NOT trên các byte dữ liệu và thực hiện trên từng bit có cùng giá trị vị trí. Nếu thanh chứa A có giá trị 00110101B, lệnh AND logic sau:

ANL <dest - byte> <src - byte>

ORL <dest - byte> <src - byte>

XRL <dest - byte> <src - byte>

Các phép toán logic có thể được thực hiện trên một byte bất kỳ trong bộ nhớ dữ liệu nội mà không cần qua trung gian thanh chứa A. Lệnh XRL direct, #data giúp ta nhanh chóng và dễ dàng đảo mức logic các bit của port:

XRL P1, #0FFH

Thực hiện một thao tác đọc – sửa – ghi. 8 bit của port 1 được đọc, sau đó từng bit được XOR với các bit tương ứng của dữ liệu tức thời. Vì 8 bit của dữ liệu tức thời đều là “1”, kết quả là từng bit của port 1 được được lấy bù và kết quả này được ghi trở lại port 1.

Các lệnh quay (RL, RR) dịch thanh chứa A: Với lệnh quay trái RL A, bit có giá trị vị trí lớn nhất MSB được đưa vào vị trí có giá trị thấp nhất LSB. Với lệnh quay

phải RR A, bit có giá trị thấp nhất LSB được đưa vào vị trí thấp nhất LSB được đưa vào vị trí có giá trị lớn nhất MSB. Các lệnh RLC A và RRC A là các lệnh quay 9 bit sử dụng thanh chứa A và cờ nhớ CY trong thanh ghi PSW. Ví dụ nếu cờ nhớ CY chứa 1 và thanh chứa A chứa 00H thì lệnh RRC A sẽ cho kết quả là cờ nhớ CY chứa 0 và thanh chứa A có nội dung là 80H. Điều này có nghĩa là cờ nhớ CY được đưa đến ACC.7 và ACC.0 được đưa đến cờ nhớ.

Lệnh SWAP A trao đổi nửa thấp 4 bit với nửa cao 4 bit trong thanh chứa A với nhau. Lệnh này dùng trong các phép toán số BCD. Chẳng hạn thanh A chứa một số nhị phân nhỏ hơn $100_{(10)}$ ta biến đổi số nhị phân này thành BCD bằng dòng lệnh:

```
MOV    B, #10
DIV    AB
SWAP   A
ADD    A, B
```

Việc chia một số cho 10 trong 2 lệnh đầu tiên tạo ra số chục trong nửa thấp thanh A và số đơn vị trong thanh B. Lệnh SWAP và ADD di chuyển số chục đến nửa cao của thanh chứa A và số đơn vị vào nửa byte thấp của thanh chứa này.

```
CLR    A          ; (A) 0
CLR    C          ; (C) 0
CLR    Bit        ; (Bit) 0
RL     A          ; Quay vòng thanh ghi A qua trái 1 bit (An + 1)(An)
                    ; n = 06 (A0) (A7)
SWAP A   ; Đổi chỗ 4 bit thấp và 4 bit cao của A cho nhau (A3A0) (A7A4).
```

Ví dụ 2.7: Hãy nhập dữ liệu bên ngoài vào 89C51 qua Port P1, và dò xem nội dung có bằng 45h không. Nếu có thì gửi nội dung có trong địa chỉ 99h ra Port P2.

```
Mov    p2, #00h       ; Chọn p2 làm port xuất
Mov    p1, #0ffh      ; Chọn P1 làm port nhập
Mov    r3, #45h       ; Cất 45h vào thanh r3
Mov    a, p1   ;      ; Cho nhập dữ liệu ngoài qua port P1
Xrl    a, r3          ; So sánh 8 bit trong a với 8 bit trong r3
Jnz    exit_          ; Nếu A = 0 thì nhảy đến nhãn exit_
Mov    p2, 99h
Exit_: ...
```

Ví dụ 2.8: Chuyển mã ASCII ra dạng số BCD nén, trước hết cho chuyển về dạng không nén sau đó kết hợp các số BCD không nén thành BCD nén. Khi chúng ta gõ phím "4" (mã ASCII là 34h và phím "7" (mã ASCII là 37h) vậy từ 2 số 34h và 37h tạo ra số BCD dạng nén tức là tạo ra số 47h (0010 0111b).

```
Mov     a, #'4'         ; Gắn mã ASCII của số 4 cho thanh ghi a (a có số 34h)
Mov     r1, #'7'        ; Gắn mã ASCII của số 7 cho thanh ghi a (a có số 37h)
Anl     a, #0fh ; Lấy AND của a và 0fh để che 4 bit cao, giữ nguyên 4 bit thấp
Anl      r1, #0fh; Lấy AND của r1 và 0fh để che 4 bit cao, giữ nguyên 4 bit thấp
Swap a                  ; Lật 4 bit trong a lên nằm ở 4 bit cao, trong a có 40h
Orl     a, r1           ; Ghép 4 bit cao của a và 4 bit thấp của r1, tạo ra số BCD
;  dạng nén.
```

2.3.6. Các lệnh luận lý, bit

8051 chứa một bộ xử lí luận lý đầy đủ cho các hoạt động bit đơn, đây là một điểm mạnh của họ vi điều khiển MSC-51 mà các họ vi điều khiển khác không có.

RAM nội chứa 128 bit đơn vị và các vùng nhớ các thanh ghi chức năng đặc biệt cấp lên đến 128 đơn vị khác. Tất cả các đường Port là bit định vị, mỗi đường có thể được xử lí như Port đơn vị riêng biệt. Cách truy xuất các bit này không chỉ các lệnh rẽ nhánh không, mà là một danh mục đầy đủ các lệnh MOVE, SET, CLEAR, COMPLEMENT, OR, AND.

Toàn bộ sự truy xuất của bit dùng sự định vị trực tiếp với những địa chỉ từ 00H - 7FH trong 128 vùng nhớ thấp và 80H - FFH ở các vùng thanh ghi chức năng đặc biệt.

Bit Carry C trong thanh ghi PSW và được dùng như một sự tích lũy đơn bit của bộ xử lí luận lý. Bit Carry cũng là bit định vị và có địa chỉ trực tiếp vì nó nằm trong PSW. Hai lệnh CLR C và CLR CY đều có cùng tác dụng là xóa bit cờ Carry nhưng lệnh đầu mất 1 byte còn lệnh sau mất 2 byte.

Điều khiển đơn bit được dùng cho nhiều thiết bị xuất/ nhập, bao gồm xuất ra Rơle, động cơ, cuộn dây, các LED, mạch còi báo động, loa hoặc nhập từ các chuyển mạch hoặc các bộ chỉ thị trạng thái. Nếu có một mạch còi báo động nối với bit 7 của port 1, ta có thể tác động mạch còi bằng cách bật bit của port:

```
SETB    P1.7
```

Hoặc tắt còi bằng cách xóa bit của port:

CLR P1.7

Trình dịch hợp ngữ sẽ biến đổi ký hiệu P1.7 thành địa chỉ bit là 97H. Trình bày sau cho phép ta di chuyển một cờ vào một chân của port:

MOV C, FLAG

MOV P1.0, C

Trong ví dụ trên, FLAG là tên của 1 bit được định địa chỉ trong 128 vị trí thấp hoặc trong không gian SFR. Một đường xuất hoặc nhập (bit 0 của port 1) được bật hoặc xóa phụ thuộc vào bit cờ có giá trị "1" hoặc "0". Bit nhớ trong PSW được dùng như một thanh chứa đơn bit của bộ xử lý logic trên bit, các lệnh đơn bit liên quan đến bit nhớ ký hiệu C là các lệnh đặc biệt liên quan đến cờ nhớ (như CLR C). Bit nhớ cũng có một địa chỉ trực tiếp vì bit này được lưu trong thanh ghi PSW, thanh ghi này được định địa chỉ từng bit.

Các lệnh logic trên bit bao gồm cả lệnh ANL và ORL nhưng không bao gồm lệnh XRL. Nếu ta cần XOR 2 bit, Bit1 và Bit2, và kết quả cất trong cờ nhớ, các lệnh sau được sử dụng:

MOV C, Bit1

JNB Bit2, SKIP

CPL C

SKIP:…

Trước tiên Bit1 được nạp cho cờ nhớ. Nếu Bit2 bằng 0 thì C đã chứa kết quả (nghĩa là Bit1 $\oplus$ Bit2 = Bit1 nếu Bit2 = 0). Nếu bit2 = 1, C chứa kết quả là bù của cờ nhớ. Việc lấy bù C hoàn tất phép toán XOR.

Hoạt động của các lệnh luận lý được tóm tắt như sau:

```
CLR   C        ; Xóa cờ Carry xuống 0. Có ảnh hưởng cờ Carry.
CLR   BIT      ; Xóa bit xuống 0. Không ảnh hưởng cờ Carry
SET   C        ; Set cờ Carry lên 1. Có ảnh hưởng cờ Carry.
SET   BIT      ; Set bit lên 1. Không ảnh hưởng cờ Carry.
CPL   C        ; Đảo bit cờ Carry. Có ảnh hưởng cờ Carry.
CPL   BIT      ; Đảo bit. Không ảnh hưởng cờ Carry.
ANL   C, BIT    ; (C) (C) AND (BIT): Có ảnh hưởng cờ Carry.
ANL   C, /BIT  ; (C) (C) AND NOT (BIT): Không ảnh hưởng cờ Carry.
```

ORL C, BIT ; (C) (C) OR (BIT): Tác động cờ Carry.

ORL C, /BIT ; (C) (C) OR NOT (BIT): Tác động cờ Carry.

MOV C, BIT ; (C) (BIT) Cờ Carry bị tác động.

MOV BIT, C ; (BIT) (C) Không ảnh hưởng cờ Carry.

2.3.7. Các lệnh xen vào

NOP; Không hoạt động gì cả, chỉ tốn 1 byte và 1 chu kỳ máy. Ta dùng để delay những khoảng thời gian nhỏ.

2.4. Timer và Counter

2.4.1. Chức năng Timer

2.4.1.1. Chế độ và hoạt động thanh ghi

Trong 8051 có 4 chế độ Timer/counter có 2 bit C/T thanh ghi TMOD ở địa chỉ byte 89H quyết định chế độ Timer/counter.

GATE1	**C/T**	**M1**	**M0**	**GATE0**	**C/T**	**M1**	**M0**
Timer/Counter 1				Timer/Counter 0			

Bảng 2.5: Các chế độ chọn C/T quy định từ hai bit M1, M0

M1	**M0**	**Modul**	**Chế độ hoạt động**
0	0	0	Bộ định thời 13 bit (8 bit C/T và 5 bit đặt trước)
0	1	1	Bộ định thời 16 bit không đặt trước, không tự nạp lại
1	0	2	Bộ định thời 8 bit tự nạp lại
1	1	3	Chế độ định thời chia tách

C/T = 1 chế độ chọn là Counter, bit này C/T =0 chế độ là Timer

GATE1 = 1 hoạt động theo điều khiển ngắt ngoài (INT1 = 1)/ GATE1 = 0 theo điều khiển trong không phụ thuộc INT1.

GATE0 = 1 hoạt động theo điều khiển ngắt ngoài (INT0)/ GATE0 = 0 theo điều khiển trong không phụ thuộc INT0

MOV TMOD, #01H(0000 0001B); chỉ có bit M0 nhận 1 nên ta có bộ Timer 0 Modul 1 chế độ 16 bit không tự nạp lại.

Thanh ghi TCON có địa chỉ byte 88H

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Counter1/Timer1		Counter0/Timer0					

TF1 Cờ báo tràn TF1/TF0 = "1" khi xung đến thanh ghi TH1/TH0, TL1/TL0 từ 0000H đến FFFFH đối với chế độ 1 (16 bit) hoặc thanh ghi TL1/TL0 đếm từ 00H đến FFH đối với chế độ 2 (8bit). TR1, TR0 bắt đầu cho Timer/Counter hoạt động hay dừng.

2.4.1.2. Chế độ và nguồn xung clock định thời

Từng bộ định thời sẽ được đề cập đến trong mục này. Chip 8051 có 2 bộ định thời T0 và T1 nên ta có các thanh ghi TH0, TL0, TH1, TL1 và các cờ tràn TF0, TF1 tương ứng.

2.4.1.3. Chế độ định thời 16 bit (chế độ 1)

Xung clock đặt vào các thanh ghi định thời cao và thấp kết hợp (TLx/THx). Khi có xung clock đến, bộ định thời đếm lên: 0000H, 0001H, 0002H… một tràn sẽ xuất hiện khi có sự chuyển số đếm từ FFFFH xuống 0000H, sự kiện này sẽ bật cờ tràn bằng "1" và bộ định thời tiếp tục đếm. Cờ tràn là bit TFx nằm trong thanh ghi điều khiển định thời TCON, bit này được đọc hoặc ghi bởi phần mềm.

Bit có trọng số lớn nhất (MSB) của giá trị trong các thanh ghi định thời là bit 7 của THx và bit có trọng số thấp nhất (LSB) là bit 0 của TLx. Bit LSB thay đổi trạng thái và chia 2 tần số xung clock định thời ở ngõ vào trong khi bit MSB thay đổi trạng thái và chia cho 65536 (tức là 2^{16}) tần số xung clock định thời ở ngã vào. Các thanh ghi định thời (TLx/THx) có thể được đọc hoặc ghi ở 1 thời điểm bất kỳ bởi phần mềm.

2.4.1.4. Chế độ tự nạp lại 8 bit (chế độ 2)

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

2.4.1.5. Định thời một khoảng thời gian

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

2.4.1.6. Khởi tạo và truy xuất các thanh ghi định thời

Các bộ định thời thường được khởi động 1 lần ở thời điểm bắt đầu chương trình. TMOD là thanh ghi được khởi động trước tiên vì đây là thanh ghi thiết lập chế độ hoạt động. Dòng lệnh sau đây là khởi động bộ định thời 1 hoạt động chế độ 16 bit (chế độ 1) xung clock được cấp từ mạch dao động trên chip.

```
MOV   TMOD, #00010000B
```

Kết quả của lệnh này thiết lập M1 = 0 và M0 = 1 để ấn định chế độ 1, C/$\overline{T}$ = 0 và GATE = 0 để sử dụng xung clock trên chip, xóa các bit chọn chế độ của bộ định thời 0. Trong thực tế bộ định thời làm việc bắt đầu khi bit điều khiển hoạt động TR1 được bật bằng "1".

Trong trường hợp cần đếm số đếm ban đầu các thanh ghi định thời TL1, TH1 cũng phải được khởi động, cờ tràn = "1" khi xảy ra số đếm tràn từ FFFFH xuống 0000H, vì vậy một khoảng thời gian 100μs có thể định thời bằng cách khởi động TL1/TH1 chứa số đếm nhỏ hơn 0000H một lượng là 100 nghĩa là -100 hay FF9CH. Các lệnh sau thực hiện điều này:

```
MOV    TL1, #9CH
MOV    TH1, #0FFH
```

Kế đến bộ định thời bắt đầu hoạt động bằng cách thiết lập bit điều khiển hoạt động bằng 1 như sau:

```
SETB TR1
```

Cờ tràn được hoạt động thiết lập sau khoảng thời gian 100μs bằng cách sử dụng một lệnh rẽ nhánh và lặp lại chính lệnh này trong khi cờ tràn chưa được bật bằng "1":

WAIT: JNB TF1, WAIT

Khi bộ định thời tràn, ta cần dừng bộ định thời và xóa cờ tràn bằng phần mềm:

CLR TR1 ; Dừng bộ định thời

CLR TF1 ; Xóa cờ tràn

- Đọc bộ định thời: ta phải đọc 2 thanh ghi định thời bằng hai dòng lệnh liên tiếp, một sai lệch có thể xuất hiện nếu có tràn từ byte thấp chuyển sang byte cao giữa 2 lần đọc do vậy ta không đọc đúng được giá trị cần đọc. Giải pháp đưa ra là trước tiên ta đọc byte cao, kế đến byte thấp và rồi đọc byte cao lần nữa. Nếu byte cao thay đổi giá trị ta lặp lại các thao tác đọc vừa nêu. Như đọc thanh ghi TL1/TH1 đưa vào thanh ghi R6, R7 và giải quyết vấn đề vừa nêu:

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

2.4.2. Chức năng Counter

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

2.5. Truyền thông nối tiếp

2.5.1. Chức năng nối tiếp

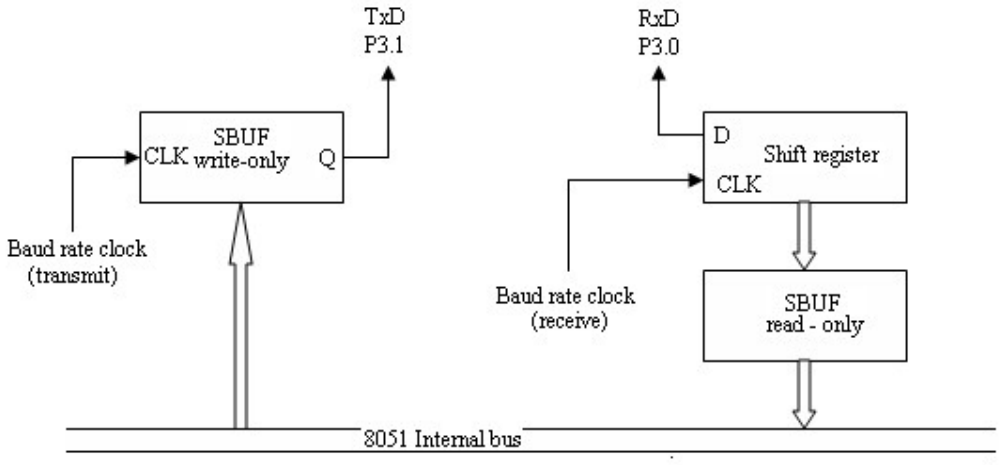
2.5.1.1. Giới thiệu

Chức năng của port nối tiếp là thực hiện việc chuyển đổi dữ liệu song song thành nối tiếp khi phát và chuyển đổi dữ liệu nối tiếp thành song song khi thu. Các mạch phần cứng bên ngoài truy xuất thông qua chân TxD (P3.1 phát dữ liệu) và RxD (P3.0 thu dữ liệu).

Đặc trưng của port nối tiếp là hoạt động song công nghĩa là có thể thu phát đồng

thời. Phần mềm sử dụng 2 thanh ghi chức năng đặc biệt SBUF và SCON để truy xuất port nối tiếp. Bộ đệm của port nối tiếp SBUF có địa chỉ 99H. Việc ghi lên SBUF sẽ nạp dữ liệu để phát và việc đọc SBUF sẽ truy xuất dữ liệu đã nhận (xem Hình 2.8).

Thanh ghi điều khiển port nối tiếp SCON (địa chỉ 98H là thanh ghi định địa chỉ bit) thanh ghi này chứa các bit trạng thái và các bit điều khiển. Các bit điều khiển sẽ thiết lập chế độ hoạt động cho port nối tiếp còn các bit trạng thái chỉ ra sự kết thúc việc thu hoặc phát một ký tự.



Hình 2.8: Sơ đồ khối của port nối tiếp

Do các chế độ truyền thông nối tiếp chuẩn RS232 ra đời rất lâu trước khi các dòng IC học TTL, nên sự tương thích điện áp đã không như mong muốn. Hầu hết các cổng RS232 (COM của PC) quy ước mức "0" có giá trị điện áp -25V ÷ -3V, mức "1" có giá trị điện áp +3V ÷ 25V nên khi mức điện áp rơi vào khoảng -3V ÷ +3V là không xác định để tương thích với các chế độ của vi điều khiển qua truyền thông với COM PC người ta sử dụng MAX232 để làm nhiệm vụ chuyển đổi tương thích cho 2 chế độ điện áp trên.

2.5.1.2. Chế độ port nối tiếp

Thanh ghi chọn chế độ SCON của port nối tiếp ở địa chỉ 99H, trước khi sử dụng port nối tiếp, thanh ghi SCON phải được khởi động đúng chế độ yêu cầu, mô tả như:

MOV SCON, #01010010B

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

Port nối tiếp có 4 chế độ hoạt động thụ thuộc vào SM0 và SM1.

Bảng 2.6: Tóm tắt thanh ghi SCON

Bit	Ký hiệu	Địa chỉ	Mô tả
SCON.7	SM0	9FH	Bit 0 chọn chế độ port nối tiếp
SCON.6	SM1	9EH	Bit 1 chọn chế độ port nối tiếp
SCON.5	SM2	9DH	Bit này cho phép truyền thông đa xử lý ở chế độ 2 và 3, bit RI sẽ không được tích cực nếu bit thứ 9 nhận được 0
SCON.4	REN	9CH	Cho phép thu, phải được bật để nhận ký tự
SCON.3	TB8	9BH	Bit phát 8. Bit thứ 9 được phát ở chế độ 2 và 3, bật và xóa bởi phần mềm
SCON.2	RB8	9AH	Bit thu 8. Bit thứ 9 nhận được
SCON.1	TI	99H	Cờ ngắt phát, cờ này được bật ngay khi kết thúc việc phát một ký tự, xóa bởi phần mềm
SCON.0	RI	98H	Cờ ngắt phát, cờ này được bật ngay khi kết thúc việc thu một ký tự, xóa bởi phần mềm

2.5.2. Hoạt động truy xuất thanh ghi

2.5.2.1. Thanh ghi 8 bit (chế độ 0)

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

Bằng cách ghi giá trị 0 vào SM0, SM1. Dữ liệu nối tiếp được thu và phát thông qua chân RxD, TxD xuất xung clock dịch bit.

Một ứng dụng khả thi của chế độ 0 là mở rộng thêm cho các ngõ ra cho 8051. Một vi mạch thanh ghi dịch nối tiếp – song song có thể được nối với các chân TxD và RxD của 8051 để cung cấp thêm 8 đường xuất.

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

2.5.2.2. UART 8- bit có tốc độ baud thay đổi (chế độ 1)

Trong chế độ 1 port nối tiếp hoạt động như một bộ thu phát không đồng bộ (UART) 8 bit có tốc độ baud thay đổi được UART là bộ thu phát dữ liệu nối tiếp với mỗi một ký tự dữ liệu được đứng trước bởi một bit Start (0) và đứng sau bởi một bit Stop (1). Như vậy chế độ 1 có 10 bit được thu trên chân RxD và 10 bit phát trên chân TxD cho mỗi 1 ký tự dữ liệu.

Khi hoạt động thu bit Stop đưa đến RB8 của SCON. Với 8051, tốc độ baud được thiết lập bởi tốc độ tràn. Bộ thu bao gồm việc phát hiện bit Start sai bằng cách yêu cầu 8 số đếm ở trạng thái 0 sau khi có sự chuyển trạng thái từ "1" xuống "0".

2.5.2.3. UART 9- bit có tốc độ baud cố định (chế độ 2)

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

2.5.3. Thiết kế chương trình

2.5.3.1. Cho phép thu

Bit cho phép thu REN trong thanh ghi SCON phải được bật bởi phần mềm để cho phép nhận các ký tự (REN = 1) vừa cho phép thu vừa cho phép truyền. Điều này thường được thực hiện ở đầu chương trình khi port nối tiếp, bộ định thời,…được khởi động và thực hiện theo 2 cách sau:

```
SETB   REN
```

Hoặc

```
MOV   SCON, #xxx1xxxxB
```

2.5.3.2. Các cờ ngắt

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

Nếu phần mềm muốn nhập một ký tự từ một thiết bị ghép với port nối tiếp phần mềm phải chờ cho tới khi RI được bật kế đến phần mềm xóa RI và đọc ký tự từ SBUF.

Thể hiện như sau:

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

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

```
MOV   SBUF, A            ; Phát ký tự
WAIT: JNB   TI, WAIT     ; Kiểm tra TI cho đến khi bằng 1
      CLR   TI           ; Xóa TI
```

Các chuỗi lệnh thu và phát ở trên là một phần của các chương trình con xuất nhập ký tự và chúng sẽ được mô tả chi tiết trong các nội dung khác.

2.5.3.3. Tốc độ của port nối tiếp

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

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

Vì thanh ghi PCON không được định nghĩa từng bit nên việc bật bit SMOD lên "1" mà không làm thay đổi các bit khác của thanh ghi này được thực hiện bằng những dòng lệnh sau:

```
MOV   A, PCON      ; Lấy giá trị hiện hành của PCON
SETB  ACC.7        ; Set bit 7 bằng 1 (SMOD)
MOV   PCON, A      ; Nạp giá trị mới vào PCON
```

Kỹ thuật thường dùng để tạo xung clock tốc độ baud là khởi động thanh ghi TMOD ở chế độ tự nạp lại 8 bit và đặt giá trị nạp lại thích hợp vào thanh ghi TH1 để có tốc độ tràn đúng, từ đó tạo ra tốc độ baud.

MOV TMOD, 0010xxxxB

Nếu một tần số dao động trên chip là 12MHz thì bộ định thời 1 cấp xung clock là 1 MHz. Do bộ định thời phải tràn ở tốc độ là 38.4 KHz, việc tràn cần xảy ra sau 26.04 xung clock và được làm tròn là 26. Vì bộ định thời đếm lên và tràn khi có số đếm từ FFH chuyển thành 00H, 26 số đếm nhỏ hơn 0 là giá trị nạp lại cần để nạp cho thanh ghi TH1. Giá trị này -26.

MOV TH1, #-26

Trình dịch hợp ngữ sẽ biến đổi cần thiết. Trong trường hợp này -26 sẽ biến đổi thành 0E6H vậy thì lệnh trên trở thành

MOV TH1, #0E6H

Ví dụ 2.9: Viết chương trình để khởi động port nối tiếp sao cho port này hoạt động như UART 8-bit với tốc độ 2400 baud. Sử dụng bộ định thời 1 để cung cấp xung clock cho tốc độ baud. Tốc độ baud được tính như trong bảng sau:

Bảng 2.7: Bảng tốc độ truyền thông

Tốc độ baud	TH1 (thập phân)	TH1 (Hexa)
9600	-3	FDH
4800	-6	FAH
2400	-12	F4H
1200	-24	F8H

Với trường hợp này ta khởi động 4 thanh ghi SMOD, TMOD, TCON và TH1. Các giá trị yêu cầu được tóm tắt sau:

Bảng 2.8: Các chế độ hoạt động thanh ghi

	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
SCON	0	1	0	1	0	0	0	0
	GATE	C/T	M1	M0	GATE	C/T	M1	M0
TMOD	0	0	1	0	0	0	0	0
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
TCON	0	1	0	0	0	0	0	0
TH1	1	1	1	0	0	1	1	0

Việc thiết lập SM0/SM1 =0/1 nhằm đặt port nối tiếp ở chế độ UART 8-bit, REN = “1” cho phép port nối tiếp thu các ký tự. Bật TI = “1” cho phép phát ký tự đầu tiên bằng cách chỉ ra rằng bộ đệm phát rỗng. Với thanh ghi TMOD việc thiết lập M1/M0 = 1/0 đặt bộ định thời vào chế độ tự nạp lại 8 bit. Việc bật TR = “1” trong TCON sẽ

khởi động bộ định thời 1 hoạt động.

Các bit khác được cho bằng "0" do chúng điều khiển các đặc trưng hoặc các chế độ không được sử dụng trong các chế độ này. Giá trị cần nạp cho TH1 là giá trị cung cấp tốc độ tràn 2400x32=76.8kHz. Giả sử tần số của mạch dao động trên chip là 12MHz bộ định thời 1 được cung cấp xung clock có tần số 1MHz và số xung clock cho mỗi lần tràn là 1000 /76.8 = 13.02 vậy giá trị nạp lại là -13 hoặc 0F3H.

Ví dụ 2.10: Chuỗi lệnh khởi động port nối tiếp và truyền ký tự YES ở tốc độ 9600 (chế độ 8 bit dữ liệu 1 bit START, 1 bit STOP như sau:

```
org     8100h
mov     scon, #50h      ; Port nối tiếp chế độ 1 (1 start + 8 bit dữ liệu + 1 stop)
mov     tmod, #20h      ; Bộ định thời 1 chế độ 2
mov     th1, #-3        ; Giá trị nạp lại để có 9600 baud
setb    tr1             ; Bộ định thời 1 hoạt động
again: mov      a, #"y"         ; Truyền ký tự y sử dụng bảng mã ASSCII
acall   truyen
mov     a, #"e"
acall   truyen
mov     a, #"s"
acall   truyen
sjmp    again
truyen:
mov     sbuf, a         ; Nạp ký tự từ a vào sbuf để truyền
here:   jnb ti, here    ; Chờ cho tới khi truyền xong ký tự
ret
end
```

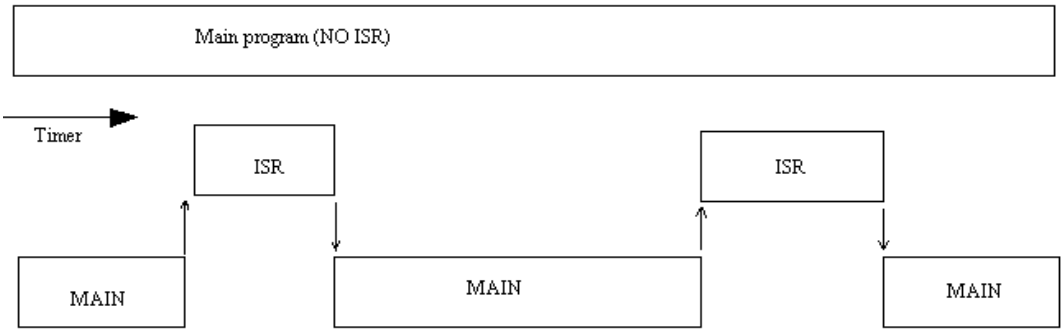
2.6. Hoạt động ngắt

2.6.1. Chức năng ngắt

Ngắt là sự kiện xảy ra của một điều kiện (một sự kiện) làm cho chương trình hiện hành bị tạm ngưng trong khi điều kiện được phục vụ bởi một trình khác. Dĩ nhiên CPU không thể thực thi nhiều hơn một lệnh ở một thời điểm nhưng có thể tạm ngưng một công việc này để thực thi một công việc khác rồi sau đó trở về thực thi tiếp chương trình đã tạm ngưng và vấn đề trên do điều khiển ngắt, việc ngắt nhằm đáp

ứng một sự kiện mà sự kiện này không đồng bộ với chương trình chính hay nói cách khác CPU không biết trước sẽ bị ngắt khi nào.

Chương trình xử lý ngắt gọi là trình phục vụ ngắt ISR hay một quản lý ngắt và mô tả như Hình 2.9 một ví dụ điển hình ngắt là nhập bằng tay sử dụng bàn phím khi buông tay tạo ra xung từ mức cao xuống mức thấp và chương trình chình được ngắt. ISR được thực thi để đọc mã phím, sau đó kết thúc bằng cách chuyển điều khiển trở về chương trình chính.



Hình 2.9: Sơ đồ sử dụng chương trình không ngắt và có ngắt

Họ 8051 có 5 nguyên nhân ngắt: 2 do bên ngoài, 2 do bộ định thời và 1 do port nối tiếp. Riêng dòng 8052 có thêm 1 ngắt do bộ định thời thứ 3. Khi thiết lập trạng thái ban đầu cho hệ thống (cấp nguồn, Reset) tất cả các ngắt đều bị vô hiệu hóa và sau đó chúng được cho phép riêng rẽ theo chương trình.

Khi xảy ra hai ngắt đồng thời hay một ngắt đang phục vụ xảy ra một ngắt khác thì ta có hai sơ đồ ngắt sơ đồ chuỗi vòng (cố định) và sơ đồ hai mức ưu tiên (do người lập trình).

2.6.2. Tổ chức ngắt

2.6.2.1. Cho phép và không cho phép ngắt

Mỗi nguyên nhân ngắt cho phép hay không cho phép ngắt thông qua thanh ghi chức năng đặc biệt định địa chỉ bit IE (0A8H), bit toàn cục không cho phép hoặc cho phép ngắt.

Hai bit sau đây phải bật bằng "1" để cho phép một ngắt nào đó, bit cho phép ngắt riêng rẽ là bit cho phép ngắt toàn cục. Chẳng hạn ngắt do bộ định thời 1:

```
SETB   ET1    ; Cho phép ngắt do bộ định thời 1
SETB   EA     ; Cho phép ngắt toàn cục
```

Hoặc bằng lệnh

MOV IE, #10001000B

Bảng 2.9: Địa chỉ cho phép ngắt

Bit	Ký hiệu	Địa chỉ	Mô tả
IE.7	EA	AFH	Cho phép 1/ không cho phép 0 toàn cục
IE.6	-	AEH	None
IE.5	ET2	ADH	Cho phép ngắt do bộ định thời 2
IE.4	ES	ACH	Cho phép ngắt do port nối tiếp
IE.3	ET1	ABH	Cho phép ngắt do bộ định thời 1
IE.2	EX1	AAH	Ngắt ngoài 1 (P3.3)
IE.1	ET0	A9H	Cho phép ngắt do bộ định thời 0
IE.0	EX0	A8H	Ngắt ngoài 0 (P3.2)

2.6.2.2. Ưu tiên ngắt

Mỗi một nguyên nhân ngắt được lập trình riêng để có một trong hai mức ưu tiên thông qua thanh ghi ưu tiên ngắt IP (interrup priority) có địa chỉ byte 0B8H.

Bảng 2.10: Địa chỉ ưu tiên ngắt

Bit	Ký hiệu	Địa chỉ bit	Mô tả (1 mức cao, 0 thấp)
IP.7	-	-	Không sử dụng
IP.6	-	-	Không sử dụng
IP.5	PT2	0BDH	Ưu tiên cho ngắt do bộ định thời 2
IP.4	PS	0BCH	Ưu tiên cho ngắt do port nối tiếp
IP.3	PT1	0BBH	Ưu tiên cho ngắt do bộ định thời 1
IP.2	PX1	0BAH	Ưu tiên ngắt ngoài 1
IP.1	PT0	0B9H	Ưu tiên cho ngắt do bộ định thời 0
IP.0	PX0	0B8H	Ưu tiên ngắt ngoài 0

Khi thiết lập hệ thống thanh ghi IP sẽ mặc định tất cả các mức ưu tiên mức thấp, ngắt có ưu tiên mức cao sẽ được phụ vụ trước hoặc dừng mức ngắt có mức ưu tiên thấp.

2.6.2.3. Xử lý ngắt

Khi có một ngắt xuất hiện và CPU chấp nhận chương trình chính bị ngắt. Các thao tác sau đây xảy ra:

- Hoàn tất việc thực thi hiện hành
- Bộ đếm chương trình PC được cất vào stack
- Trạng thái của ngắt hiện hành được lưu trữ lại
- Các ngắt được chặn lại ở mức ngắt

Bộ đếm chương trình PC được nạp địa chỉ vectơ của chương trình phục vụ ngắt ISR. ISR được thực thi và kết thúc khi gặp lệnh RETI, lệnh này lấy lại giá trị cũ của bộ đếm chương trình PC từ Stack và phục hồi trạng thái của ngắt cũ. Và thực thi chương trình chính ở nơi tạm dừng.

Bảng 2.11: Bảng thanh ghi các cờ ngắt

Ngắt	Cờ	Thanh ghi SFR và vị trí bit
Ngắt ngoài 0	IE0	TCON.1
Ngắt ngoài 1	IE1	TCON.3
Ngắt do bộ định thời 1	TF1	TCON.7
Ngắt do bộ định thời 0	TF0	TCON.5
Ngắt do port nối tiếp	TI	SCON.1
Ngắt do port nối tiếp	RI	SCON.0
Ngắt do bộ định thời 2	TF2	T2CON.7 (8052)
Ngắt do bộ định thời 2	EXF2	T2CON.6 (8052)

2.6.2.4. Các vectơ ngắt

Khi một ngắt được chấp nhận, giá trị được nạp cho bộ đếm chương trình PC được gọi là vectơ ngắt. Vectơ ngắt là địa chỉ bắt đầu trình phục vụ ngắt của nguyên nhân ngắt tương ứng.

vectơ reset hệ thống (RST ở địa chỉ 0000H) được chứa trong Bảng 2.12. Vì vậy cũng được xem như 1 ngắt: chương trình chính bị ngắt và PC được nạp giá trị mới.

Bảng 2.12: Vectơ ngắt trên 8051

Ngắt do	Cờ	Địa chỉ vectơ
Reset hệ thống	RST	0000H
Ngắt ngoài 0	IE0	0003H
Bộ định thời 0	TF0	000BH
Ngắt ngoài 1	IE1	0013H
Bộ định thời 1	TF1	001BH
Port nối tiếp	RI hoặc TI	0023H
Bộ định thời 2	TF2 hoặc EXF2	002BH

Khi một trình phục vụ ngắt được trỏ tới, cờ gây ra ngắt sẽ tự động bị xóa về 0 bởi phần cứng. Các ngoại lệ bao gồm cờ RI và TI đối với các ngắt do port nối tiếp; TF2 và EXF2 đối với các ngắt do bộ định thời 2. Các nguyên nhân ngắt thuộc hai ngoại lệ nêu trên do có hai khả năng tạo ra ngắt nên trong thực tế CPU không xóa cờ ngắt.

Các bit cờ này phải được kiểm tra trong ISR để xác định nguyên nhân ngắt và sau đó cờ gây ra ngắt được xóa bởi phần mềm. Thông thường sẽ có một rẽ nhánh chương trình đến công việc tương ứng tùy thuộc vào nguyên nhân ngắt. Vì các vectơ ngắt đặt ở đáy của bộ nhớ chương trình, lệnh đầu tiên của chương trình chính thường là một lệnh nhảy qua khỏi vùng nhớ chứa các vectơ ngắt chẳng hạn như lệnh

LJMP 0030H.

2.6.3. Thiết kế chương trình

Các lỗi thường xảy ra trong thiết kế hệ thống, thường liên quan đến các ngắt, vì chúng ta đang sử dụng các ngắt và được thực thi độc lập. Mỗi một chương trình bắt đầu ở địa chỉ 0000H với giả thiết là chương trình bắt đầu được thực thi sau khi hệ thống reset. Ý tưởng cuối cùng là các chương trình này phát triển cho các ứng dụng chính thức, chúng được thường trú trong ROM hoặc EPROM.

Khuôn mẫu:

```
ORG   0000H          ; Điểm nhập sau khi Reset
LJMP  MAIN
....                 ; Các điểm nhập của ISR
ORG   0030H          ; Điểm nhập của chương trình chính
MAIN:                ; Chương trình chính bắt đầu
```

```
...
```

Lệnh đầu tiên nhảy đến địa chỉ 0030H ngay trên các vectơ ngắt nơi các ISR bắt đầu, như được trình bày bảng vectơ ngắt.

Nếu chỉ có một nguyên nhân ngắt được dùng là ngắt do bộ định thời 0 thì khuôn mẫu trình bày dưới đây:

```
Org     0000h           ; reset
Ljmp    main
Org     000bh           ; điểm nhập của ngắt do bộ định thời 0
Toisr:                  ; bắt đầu isr cho bộ định thời 0
        Reti            ; trở về chương trình chính
Main:                   ; chương trình chính
```

Nếu có nhiều ngắt được sử dụng ta phải cẩn thận để bảo đảm các ISR được bắt đầu đúng vị trí và không tràn sang ISR kế. Vì chỉ có một ngắt được sử dụng trong ví dụ trên, chương trình chính có thể bắt đầu ngay sau lệnh RETI.

Các trình phục vụ ngắt kích thước lớn:

Nếu 1 trình phục vụ ngắt dài hơn 8 byte được cần đến, ta phải di chuyển chương trình này đến một nơi khác trong bộ nhớ chương trình hoặc ta có thể cho lấn qua điểm nhập của ISR kế tiếp. Điển hình ISR bắt đầu với một lệnh nhảy đến một vùng khác của bộ nhớ chương trình, ở đó ISR được trải rộng nếu cần. Nếu chỉ khảo sát bộ định thời, khuôn mẫu sau đây có thể được sử dụng:

```
Org     0000h           ; reset
Ljmp    main
Org     000bh           ; điểm nhập của ngắt do bộ định thời 0
Ljmp    t0isr
Org     0030h           ; phía trên các vectơ ngắt
Main:                   ; chương trình chính
        ...
        T0isr:          ; bắt đầu isr cho bộ định thời 0
        Reti            ; trở về chương trình chính
```

Ví dụ 2.11: Viết chương trình tạo sóng xung vuông 10KHz trên chân P1.0 bằng cách sử dụng bộ định thời 0.

Với tần số 10 KHz yêu cầu chu kỳ 100 µs. Với thời gian mức cao là 50 µs và thời gian mức thấp 50 µs. Do khoảng thời gian này nhỏ hơn 256 µs nên chế độ 2 sử dụng được. Một

tràn xảy ra sau 50 μs yêu cầu một giá trị số đếm nhỏ hơn 00H một lượng +50 phải được nạp và được nạp lại cho TL0, nghĩa là giá trị nạp cho TH0 là -50. Dưới đây là chương trình theo yêu cầu:

```
org     8100h
mov     tmod, #02h          ; Chế độ tự nạo lại 8 bit
mov     th0, # -50h         ; Th0 chứa giá trị -50
setb    tr0                 ; Bộ định thời hoạt động
loop: jnb     tf0, loop     ; Chờ tràn
      clr     tf0           ; Xóa cờ tràn
      cpl     p1.0          ; Đổi trạng thái p1.0 (bù)
      sjmp    loop          ; Lặp lại
      end
```

2.7. Bài tập và câu hỏi ôn tập cuối chương

Bài 2.1: Viết chương trình tạo sóng xung vuông 5KHz trên chân P1.2 bằng cách sử dụng bộ định thời 1.

Bài 2.2: Mạch còi được nối chân P3.7, tín hiệu đầu vào nối chân P1.1. Viết chương trình đọc mức logic chuyển mạch chân P1.1 và hụ còi trong 0,5s sau mỗi lần phát hiện sự chuyển trạng thái từ "1" xuống "0".

Bài 2.3: Viết chương trình tạo ra sóng vuông 1kHz trên P1.1 sử dụng định thời 1.

Tài liệu tham khảo chương 2

Tiếng Việt

[1] Ngô Diên Tập, ***Vi điều khiển với lập trình C***, NXB Khoa học Kỹ thuật, Hà Nội, 2006.

Tiếng Anh

[2] Muhammad Ali Mazidi, Janice Gillispie Mazidi, Rolin D. McKinlay, ***The 8051 Microcontroller and Embedded Systems Using Assembly and C***, Department of Computer Science and Information Engineering National Cheng Kung University, Taiwan, 2003.

[3] Dogan Lbrahim, ***Microcontroller projects in C for the 8051***, Oxford Auckland Boston Johannesburg Melbourne New Delhi, 2011.

Chương 3. **NGÔN NGỮ LẬP TRÌNH**

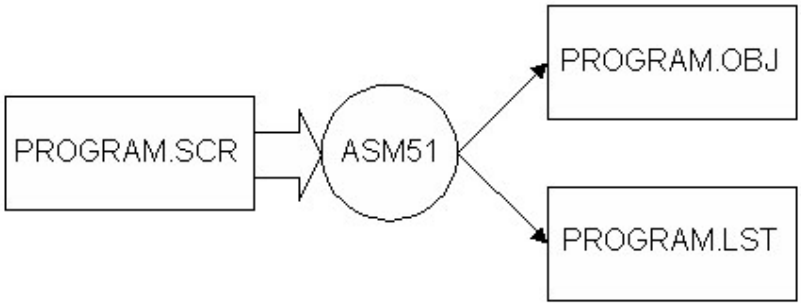
Trong kỹ thuật lập trình vi điều khiển nói chung, ngôn ngữ lập trình được sử dụng thường chia làm 2 loại: Ngôn ngữ bậc thấp (hợp ngữ) và Ngôn ngữ bậc cao (C/C++).

3.1. Lập trình hợp ngữ

ASM51 là tiêu biểu chuẩn biên dịch của họ MSC-51. ASM51 là trình biên dịch mạnh có tác dụng hữu hiệu trên hệ thống phát triển INTEL và họ IBM PC của máy vi tính.

3.1.1. Cấu trúc chương trình

ASM51 được hiện lên từ chỉ dẫn của hệ thống: ***ASM51*** *Source file (Assembly Control).* Trình biên dịch nhận một file nguồn với tư cách là ngõ nhập (PROGRAM.SCR) và chúng phát ra một file đối tượng (PROGRAM.OBJ) và file liệt kê (PROGRAM.LST). Vì hầu hết các trình biên dịch xem xét chương trình nguồn hai lần trong lúc thi hành sự dịch ngôn ngữ máy, nên chúng được mô tả qua hai Pass biên dịch là Pass1 và Pass2.



Hình 3.1: Trình biên dịch một chương trình nguồn

File Object chỉ chứa các byte nhị phân (00H - FFH) của chương trình ngôn ngữ máy. File Object Relocatable chứa một bảng ký hiệu và thông tin khác được yêu cầu bởi sự kết hợp và xác định đúng vị trí. File Listing chứa mã nguyên bảng ASCII (20H – 7FH) cho cả hai chương trình nguồn và các byte Hexadecimal trong chương trình ngôn ngữ máy.

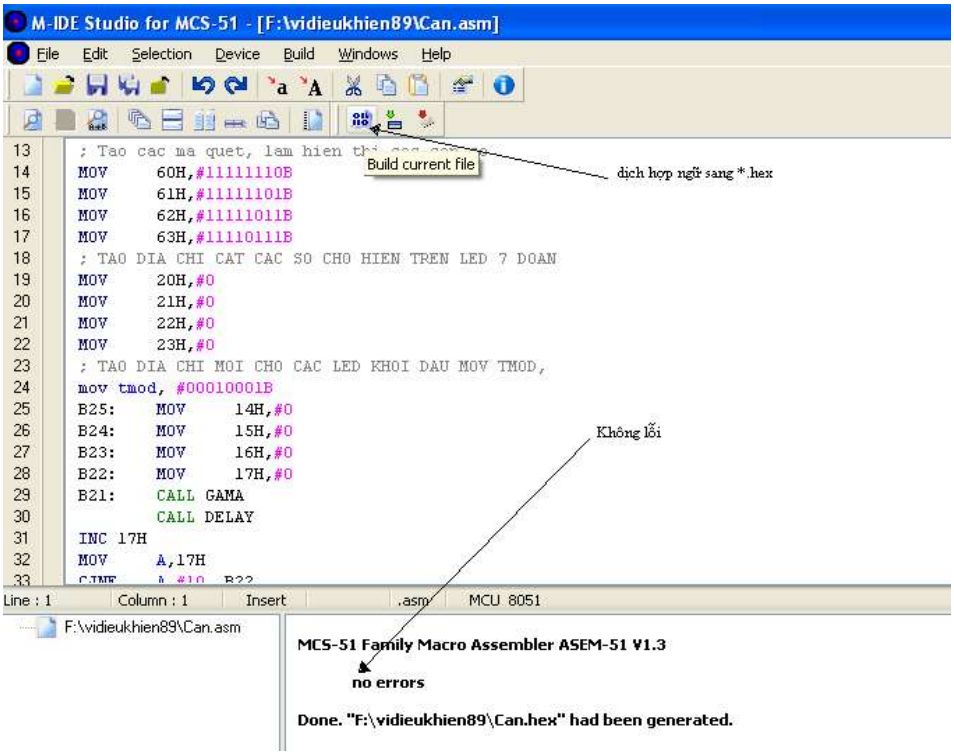
Một nhãn đặc trưng cho địa chỉ của lệnh (hoặc dữ liệu) theo sau nhãn. Khi các rẽ nhánh đến lệnh này, nhãn được dùng trong vùng toán hạng của nhánh (hoặc lệnh

nhảy).

3.1.2. Phần mềm MIDE

Sau đây là giao diện phần mềm dùng lập trình hợp ngữ cho họ 8051.

Hình 3.2: Cài đặt phần mềm MIDE-51



Hình 3.3: Cửa sổ trình hợp ngữ để lập trình file.asm

Có rất nhiều phần mềm cài đặt và dịch hợp có thể chuyển file C, ASM... thành file *.hex. Ở đây ta giới thiệu 1 phiên bản MIDE-51 dùng soạn thảo và hợp dịch sang file hex đồng thời có thể xem simulation trong bộ nhớ nội. Để cài đặt cần máy tính chạy trên nền Windown, nhắp đúp vào biểu tượng MIDE-51.exe như Hình 3.2.

Sau khi cài đặt xong, ta mở cửa sổ lập trình để lập trình hợp ngữ và biên dịch để sửa lỗi như Hình 3.3.

Phần mềm hợp ngữ này có ưu điểm như: cài đặt nhanh, bộ nhớ nhỏ, trình gỡ rối dễ hiểu, việc biên dịch và gỡ rối rõ ràng làm cho người dùng dễ dàng hiểu được cấu trúc chương trình.

3.2. Ngôn ngữ bậc cao

Ngôn ngữ bậc cao là các ngôn ngữ gần với ngôn ngữ con người hơn, do đó việc lập trình bằng các ngôn ngữ này trở nên dễ dàng và đơn giản hơn. Có thể kể đến một số ngôn ngữ lập trình bậc cao như C, Basic, Pascal… trong đó C là ngôn ngữ thông dụng hơn cả trong kỹ thuật vi điều khiển. Về bản chất, sử dụng các ngôn ngữ này thay cho ngôn ngữ bậc thấp là giảm tải cho lập trình viên trong việc nghiên cứu các tập lệnh và xây dựng các cấu trúc giải thuật.

3.2.1. Định dạng dữ liệu

Các kiểu dữ liệu trong C được liệt kê trong Bảng 3.1

Bảng 3.2: Bảng liệt kê các kiểu dữ liệu trong C

Kiểu	Số Byte	Khoảng giá trị
Char	1	-128 đến +127
Unsigned char	1	0 đến 255
Int	2	-32768 đến +32767
Unsigned int	2	0 đến 65535
Long	4	-2147483648 đến +2147483647
Unsigned long	4	0 đến 4294697295
Float	4	

* Khai báo biến:

- Cú pháp: Kiểu_dữ_liệu *Vùng_nhớ* Tên_biến Địa_chỉ;

Unsigned char data x;

- Khi khai báo biến có thể gán luôn cho biến giá trị ban đầu.

Thay vì:

unsigned char x;

x = 0;

Ta chỉ cần:

unsigned char x = 0;

- Có thể khai báo nhiều biến cùng kiểu một lúc.

unsigned int x,y,z,t;

3.2.1.1. Vùng nhớ

- Chỉ định vùng nhớ: từ khoá "Vùng_nhớ" cho phép người dùng có thể chỉ ra vùng nhớ sử dụng để lưu trữ các biến sử dụng trong chương trình. Các vùng nhớ có thể sử dụng là: CODE, DATA, BDATA, IDATA, PDATA, XDTA. Khi không khai báo vùng nhớ, trình dịch Keil C sẽ mặc định đó là vùng nhớ DATA.

Bảng 3.3: Quy ước vùng nhớ trong dữ liệu C

Vùng nhớ	Ý nghĩa
CODE	Bộ nhớ mã nguồn chương trình (ROM)
DATA	Bộ nhớ dữ liệu gồm 128 Byte thấp của RAM trong vi điều khiển (89x51)
BDATA	Bộ nhớ dữ liệu có thể định địa chỉ bit, nằm trong vùng nhớ DATA (20h – 27h)
IDATA	Bộ nhớ dữ liệu gồm 128 Byte cao của RAM trong vi điều khiển chỉ có ở một số dòng vi điều khiển sau này (**89x52**)
PDATA	Bộ nhớ dữ liệu ngoài gồm 256 Byte, được truy cập bởi địa chỉ đặt trên P0 (A0-A7)
XDATA	Bộ nhớ dữ liệu ngoài có dung lượng có thể lên đến 64 KB, được truy cập bởi địa chỉ đặt trên P0(A0-A7) và P2 (A8 – A15)

* Định nghĩa lại kiểu:

- Cú pháp: *typedef Kiểu_dữ_liệu Tên_biến;*
- Tên_biến sau này sẽ được sử dụng như một kiểu dữ liệu mới và có thể dùng để khai báo các biến khác:

typedef int m5[5];

Dùng tên m5 khai báo hai biến tên a và b có kiểu dữ liệu là mảng một chiều 5 phần tử:

m5 a,b;

Bảng 3.4: Kiểu dữ liệu định nghĩa trong Keil C

Kiểu	Số bit	Ngôn ngữ ASM
Bit	1	Setb 34H (từ 00h – 7Fh)
Sbit	1	Ví dụ setb p1^0, C… (80h – FFh)
Sfr	8	Gán byte MOV A, 67H
Sfr16	16	DPTR, TH0, TL0…

- Bit: Dùng để khai báo các biến có giá trị "0" hoặc "1" hay các biến logic trên vùng RAM của vi điều khiển. Khi khai báo biến kiểu bit trình dịch Keil C sẽ mặc định vùng nhớ sử dụng là BDATA.

- sbit, sfr, sfr16: Dùng để định nghĩa cho các thanh ghi chức năng hoặc các cổng trên vi điều khiển dùng để truy nhập các đoạn dữ liệu tương ứng 1 bit, 8 bit, 16 bit.

- Mảng: Mảng là một tập hợp nhiều phần tử cùng một kiểu giá trị và chung một tên. Các phần tử của mảng phân biệt với nhau bởi chỉ số hay số thứ tự của phần tử trong dãy phần tử. Mỗi phần tử có vai trò như một biến và lưu trữ được một giá trị độc lập với các phần tử khác của mảng. Mảng có thể là mảng một chiều hoặc mảng nhiều chiều.

* Khai báo:

- Cú pháp: *Tên_kiểu Vùng_nhớ Tên_mảng[số_phần_tử_mảng];*

Khi bỏ trống số phần tử mảng ta sẽ có mảng có số phần tử bất kì.

Unsigned int data a[5],b[2] [3]; // b=2x3=6 phần tử

Với khai báo trên ta sẽ có: mảng a là mảng một chiều 5 phần tử. Mảng b là mảng hai chiều, tổng số phần tử là 6.

Chỉ số của mảng bắt đầu từ số 0. Mảng có bao nhiêu chiều phải cung cấp đầy đủ bấy nhiêu chỉ số, như: phần tử mảng b[0] [1] là đúng. Ngược lại, khi viết b[0] là sai.

3.2.1.2. Con trỏ

Khi ta khai báo một biến, biến đó sẽ được cấp phát một khoảng nhớ bao gồm một số byte nhất định dùng để lưu trữ giá trị. Địa chỉ đầu tiên của khoảng nhớ đó chính là địa chỉ của biến được khai báo.

Con trỏ là một biến dùng để chứa địa chỉ mà không chứa giá trị, hay giá trị của con trỏ chính là địa chỉ khoảng nhớ mà nó trỏ tới.

Với các vùng nhớ cụ thể con trỏ, trỏ tới vùng nhớ đó chiếm dung lượng phụ thuộc vào độ lớn của vùng nhớ đó. Con trỏ tổng quát khi không xác định trước vùng nhớ sẽ có dung lượng lớn nhất vì vậy tốt nhất nên sử dụng con trỏ cụ thể.

Bảng 3.5: Quy ước con trỏ trong kiểu dữ liệu C

Loại con trỏ	Kích thước	Ngôn ngữ ASM
Con trỏ tổng quát	3 byte	
Con trỏ XDATA	2 byte	MOVX DPTR,#6000
Con trỏ CODE	2 byte	MOVC DPTR, #maled
Con trỏ DATA	1 byte	DPTR
Con trỏ IDATA	1 byte	MOVX A, @DPTR, 128 byte RAM
Con trỏ PDATA	1 byte	MOVX A, @DPTR, 256 byte RAM

* Khai báo biến con trỏ:

Cú pháp: *Kiểu_Dữ_liệu Vùng_nhớ *Tên_biến;*

*int *int_ptr;*

*long data *long_ptr;*

Khi không chỉ rõ vùng nhớ con trỏ sẽ được coi là con trỏ tổng quát.

3.2.1.3. Kiểu dữ liệu cấu trúc

Kiểu dữ liệu cấu trúc là một tập hợp các biến, các mảng và cả các kiểu cấu trúc khác được biểu thị bởi một tên duy nhất. Kiểu dữ liệu cấu trúc dùng để lưu trữ các giá trị, thông tin có liên quan đến nhau.

Định nghĩa và khai báo biến cấu trúc:

```
Định nghĩa: typedef struct {
                        Khai báo các biến thành phần;
                    } Tên_kiểu_cấu_trúc;
```

Khai báo: *Tên_kiểu_cấu_trúc Vùng_nhớ Tên_biến;*

```
typedef struct {
                    char day;
                    char month;
                    int year;
                } Date_type;
Date_type date,date_arr[5];
```

3.2.2. Phép toán trong ngôn ngữ C

Phép gán kí hiệu: "=".

* Cú pháp: *Biến_1 = Biến_2;*

Trong đó, *Biến_2* có thể là giá trị xác định cũng có thể là biến.

3.2.2.1. Phép toán số học

Bảng 3.6: Bảng quy ước các phép toán trong Keil C

Phép toán	Ý nghĩa	Ví dụ
+	Phép cộng	X=a+b
-	Phép trừ	X=a-b
*	Phép nhân	X=a*b
/	Phép chia lấy phần nguyên	X=a/b (a=9, b=2 → X=4)
%	Phép chia lấy phần dư	a%b (a=9, b=2 → X = 1)

3.2.2.2. Phép toán Logic

AND: &&

OR: ||

NOT: !

3.2.2.3. Các phép toán so sánh

Bảng 3.7: Bảng các phép toán so sánh trong Keil C

Phép toán	Ý nghĩa	Ví dụ	Asm
>	So sánh lớn hơn	a>b; 4>5 các giá trị 0	GT
>=	So sánh lớn hơn hoặc bằng	a>=b; 6>=2 các giá trị 1	GE
<	So sánh nhỏ hơn	a<b; 6<7 các giá trị 1	LT
<=	So sánh nhỏ hơn hoặc bằng	a<=b; 8<=5 các giá trị 0	LE
==	So sánh bằng nhau	a==b; 6==6 các giá trị 1	EQ
!=	So sánh khác nhau	a!=b; 9!=9 các giá trị 0	NE

3.2.2.4. Phép toán thao tác Bit

Bảng 3.8: Bảng các phép xử lý bit trong Keil C

Phép toán	Ý nghĩa	Ví dụ	Asm

&	Phép và (AND)	Bit_1 & Bit_2	ANL
|	Phép hoặc (OR)	Bit_1 | Bit_2	ORL
!	Phép đảo (NOT)	!Bit_1	CPL
^	Phép hoặc loại trừ (XOR)	Bit_1 ^ Bit_2	XRL
<<	Dịch trái	a<<3	RL
>>	Dịch phải	a>>4	RR
~	Lấy bù theo bit	~a	CPL

3.2.2.5. Phép toán kết hợp

Bảng 3.9: Bảng các phép toán kết hợp trong Keil C

Phép toán	Ví dụ	ASM
+=	a+=5 <=> a=a+5	INC
-=	a-=5 <=> a=a-5	DEC
*=	a*=5 <=> a=a*5	ADD
/=	a/=5 <=> a=a/5	DIV
%=	a%=5 <=> a=a%5	MOD

3.2.3. Cấu trúc chương trình C

3.2.3.1. Cấu trúc

- 1. Khai báo chỉ thị tiền xử lý
- 2. Khai báo các biến toàn cục
- 3. Khai báo nguyên mẫu các hàm
- 4. Xây dựng các hàm và chương trình chính

```
// Khai báo chỉ thị tiền xử lý
 #include<regx51.h>
 #include<string.h>
 #define Led1 P1_0
// Khai báo biến toàn cục
 unsigned char code led_arr[3];
 unsigned char data dem;
 unsigned int xdata x;
// Khai báo nguyên mẫu hàm
 void delay(unsigned int n);
```

```
bit kiemtra(unsigned int a);
void delay(unsigned int n)
 {
    // Khai báo biến cục bộ;
     // Mã chương trình trễ;
}

// Xây dựng các hàm và chương trình chính
void main()
{
     // Khai báo biến cụ bộ;
     // Mã chương trình chính;
}
Bit kiemtra(unsigned int a)
{
    // Khai báo biến cục bộ;
    // Mã chương trình kiểm tra biến a;
}
```

Chú ý: Hàm không khai báo nguyên mẫu phải được xây dựng trước hàm có lời gọi hàm đó. Ở trên do hàm "bit kiemtra(unsigned int a)" đã được khai báo nguyên mẫu hàm ở trước đó, nên có thể xây dựng hàm ở bất kì vị trí nào trong chương trình.

3.2.3.2. Chỉ thị tiền xử lý

Các chỉ thị tiền xử lý không phải là các lệnh của ngôn ngữ C mà là các lệnh giúp cho việc soạn thảo chương trình nguồn C trước khi biên dịch. Khi dịch một chương trình C thì không phải chính bản chương trình nguồn mà ta soạn thảo được dịch. Trước khi dịch, các lệnh tiền xử lý sẽ chỉnh lý bản gốc, sau đó bản chỉnh lý này sẽ được dịch. Có ba cách chỉnh lý được dùng là:

- Phép thay thế #define // Định nghĩa EQU trong ASM
- Phép chèn tệp #include // Gọi file khác cùng chạy
- Phép lựa chọn biên dịch #ifdef // Kết quả của các file cùng chạy

Các chỉ thị tiền xử lý giúp ta viết chương trình ngắn gọn hơn và tổ chức biên dịch, gỡ rối chương trình linh hoạt, hiệu quả hơn.

3.2.4. Các cấu trúc trong Keil C

3.2.4.1. Cú pháp if ... else ...

```
if ([biểu thức 1] [toán tử so sánh] [biểu thức 2]) { // Biểu thức điều kiện
    [câu lệnh 1]
} else {
    [câu lệnh 2]
}
```

Nếu biểu thức điều kiện trả về giá trị TRUE, *[câu lệnh 1]* sẽ được thực hiện, ngược lại, *[câu lệnh 2]* sẽ được thực hiện.

3.2.4.2. Cấu trúc switch / case

Giống như *if, switch / case* cũng là một dạng lệnh nếu thì, nhưng nó được thiết kế chuyên biệt để chúng ta xử lý giá trị trên một biến chuyên biệt.

Ví dụ 3.1: Xây dựng cấu trúc Switch

```
switch (action) {
    case "callMyMom":       // Gọi điện cho mẹ của tôi
    break;
    case "callMyDad":       // Gọi điện cho ba của tôi
    break;
    default:                // Mặc định là không làm gì cả
                            // Chúng ta có thể có default: hoặc không
}
```

* Cú pháp:

```
switch (var) {
  case label:          // Đoạn lệnh
  break;
  case label:          // Đoạn lệnh
  break;
 /*
  case... more and more
 */
  default:             // Statements
}
```

3.2.4.3. Cấu trúc for

Hàm for có chức năng làm một vòng lặp. Hãy hiểu một cách đơn giản, nó làm đi làm lại một công việc có một tính chất chung nào đó. Chẳng hạn, chúng ta bật tắt

một con LED thì dùng digitalWrite xuất HIGH delay rồi lại LOW rồi lại delay. Nhưng nếu chúng ta muốn làm nhiều hơn 1 con LED thì đoạn code của chúng ta sẽ dài ra. Với 1 con led, chúng ta lập trình như sau:

```
digitalWrite(led1,HIGH);
delay(1000);
digitalWrite(led1,LOW);
delay(1000);
```

Hàm for chúng ta chỉ cần một đoạn code ngắn như sau:

```
void setup(){
    Serial.begin(9600);
    int i;
    for (i = 1;i< = 10;i=i+1) {
      Serial.println(i);
    }
}
void loop(){
}
```

3.2.4.4. Cấu trúc while

Giống như for, cũng có vài khái niệm mà chúng ta cần nắm. While là một vòng lặp không biết trước số lần lặp, nó dựa vào điều kiện, điều kiện còn đúng thì còn chạy.

* Cú pháp:

```
while (<điều kiện>) {
   // Các đoạn lệnh;
}
```

Ví dụ 3.2: Chương trình sử dụng cấu trúc while

```
int day = 1;
int nam = 2014;          // Năm 2014
while (day < 365) {      // Chừng nào day < 365 thì
                         // còn chạy (<=364). Khi day // == 365 thì hết 1 năm...
    day + = 1;
    delay(60*60*24); // Một ngày có 24 giờ, mỗi giờ
       // có 60 phút, mỗi phút có 60 giây
 }
 nam + = 1;    // ... bây giờ đã là một năm mới !
```

```
// Chúc mừng năm mới:)
```

3.2.4.5. Cấu trúc break

Break là một lệnh có chức năng dừng ngay lập tức một vòng lặp *(do, for, while)* chứa nó trong đó. Khi dừng vòng lặp, tất cả những lệnh phía sau *break* và ở trong vòng lặp chịu ảnh hưởng của nó sẽ bị bỏ qua.

Ví dụ 3.3: Chương trình sử dụng cấu trúc while/ break

```
int a = 0;
while (true) {
  if (a == 5) break;
  a = a + 1;
}
// a = 5
 while (true) {
    while (true) {
        a++;
        if (a > 5) break;
    }
    a++;
    if (a > 100) break;
 }
// a = 101
```

3.2.4.6. Cấu trúc continue

continue là một lệnh có chức năng bỏ qua một chu kì lặp trong một vòng lặp *(for, do, while)* chứa nó trong đó. Khi gọi lệnh *continue*, những lệnh sau nó và ở trong cùng vòng lặp với nó sẽ bị bỏ qua để thực hiện những chu kì lặp kế tiếp.

Ví dụ 3.4: Chương trình sử dụng cấu trúc contrinue

```
int a = 0;
int i = 0;
while (i < 10) {
    i = i + 1;
    continue;
    a = 1;
}
// a vẫn bằng 0
```

3.2.4.7. Cấu trúc return

Cấu trúc "*return*" có nhiệm vụ trả về một giá trị (cùng kiểu dữ liệu với hàm) mà nó được gọi.

* Cú pháp:

```
return;
return value;          // cả 2 đều đúng
```

Thông số value: bất kỳ giá trị hoặc một đối tượng.

Ví dụ 3.5: Chương trình sử dụng cấu trúc return

```
// Hàm kiểm tra giá trị của cảm biến có hơn một ngưỡng nào đó hay không
int checkSensor(){
    if (analogRead(0) > 400) {
        return 1;
    else{
        return 0;
    }
```

3.2.4.8. Cấu trúc goto

Nó có nhiệm vụ tạm dừng chương trình rồi chuyển đến một nhãn đã được định trước, sau đó lại chạy tiếp chương trình.

* Cú pháp:

- *label:* // Khai báo một nhãn có tên là label.
- *goto label;* /* Chạy đến nhãn label rồi sau đó thực hiện tiếp những đoạn chương trình sau nhãn đó. Không nên dùng lệnh goto trong chương trình hay bất cứ chương trình nào sử dụng ngôn ngữ C nếu không chắc. Nhưng nếu sử dụng.*/

Vậy nó hữu ích khi nào, đó là lúc chúng ta đang dùng nhiều vòng lặp quá và muốn thoát khỏi nó một cách nhanh chóng.

Ví dụ 3.6: Chương trình sử dụng cấu trúc goto

```
For(byte r = 0; r < 255; r++){
    for(byte g = 255; g > -1; g--){
        for(byte b = 0; b < 255; b++){
            if (analogRead(0) > 250){ goto bailout;}
        }
    }
```

}

bailout:

3.2.5. Hàm trong Keil C

3.2.5.1. Hàm trễ

```
Void delay(long time)
{
    while(time--);      // gọi hàm delay (số đếm ví dụ 25000)
}
```

3.2.5.2. Phép gán

Kiểu dữ liệu Nhãn = địa chỉ vi điều khiển

Port vi điều khiển ký hiệu *Px^y*.

Như: *sbit LED = P3^0; // Gán nhãn LED ở địa chỉ P3.0 hay LED bit p3.0*

Với phép tiền định nghĩa *define nhãn p1_0 // p1.0 = nhãn*

Vùng RAM vi điều khiển ký hiệu 0xYY

Như: *sfr maled = 0x90 ; // Gán mã led ở địa chỉ 90H hay maled equ 90H*

3.2.5.3. Khai báo hàm

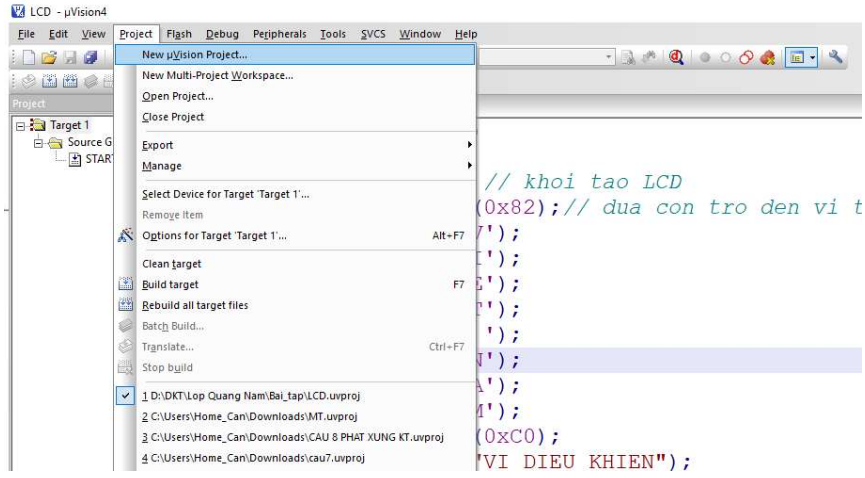
Void tên hàm(khai báo biến) nếu không có khai báo biến thì mặc định là (void)

Gọi hàm bằng lệnh tên hàm();

kiemtraphim(); // Call kiểm tra phím

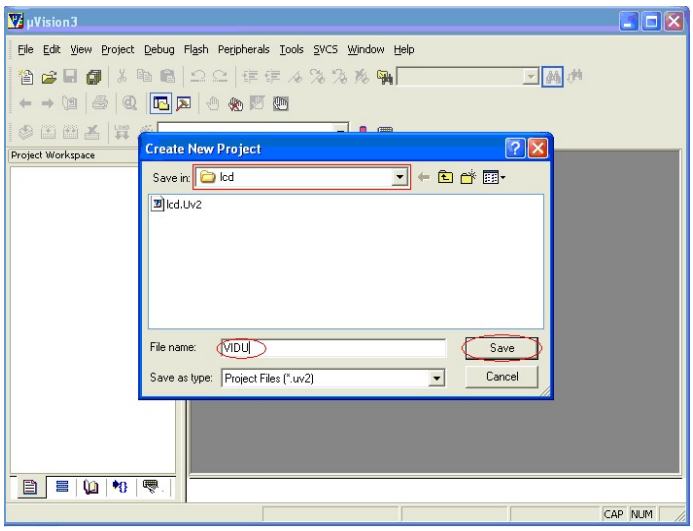
3.2.6. Phần mềm Keil C

3.2.6.1. Khởi tạo cho Project.



Hình 3.4: Giao diện phần mềm Keil C

Khởi động phần mềm Keil C và tạo một dự án bằng cách chọn Menu Project/ New Project trên giao diện. Hộp thoại create new project hiện ra như Hình 3.4:

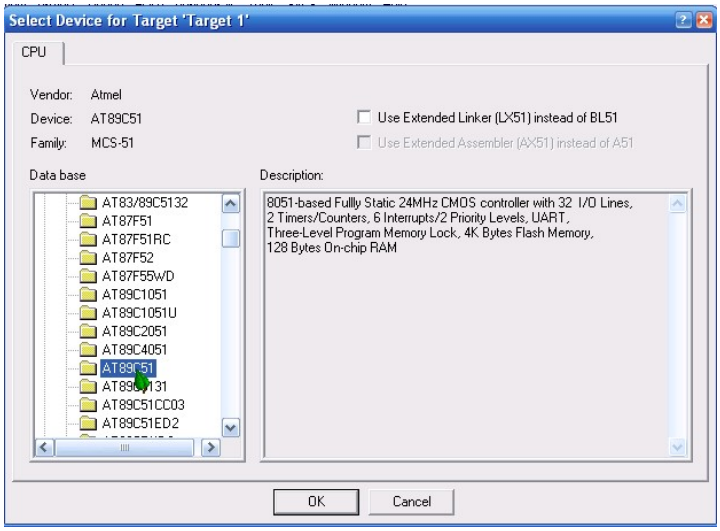


Hình 3.5: Đặt tên dự án cho Keil C

Đánh tên và chuyển đến thư mục chúng ta lưu project và chúng ta tạo mỗi một thư mục cho một project rồi chọn lưu (save)**.** Chú ý quy định đặt tên file giống như trong C/C++, thư mục cũng không sử dụng tiếng việt có dấu.

Hộp thoại sau hiện ra là lựa chọn dòng chip cần lập trình ứng dụng, ở đây lựa chọn họ 8051 là AT89xx của hãng Atmel.

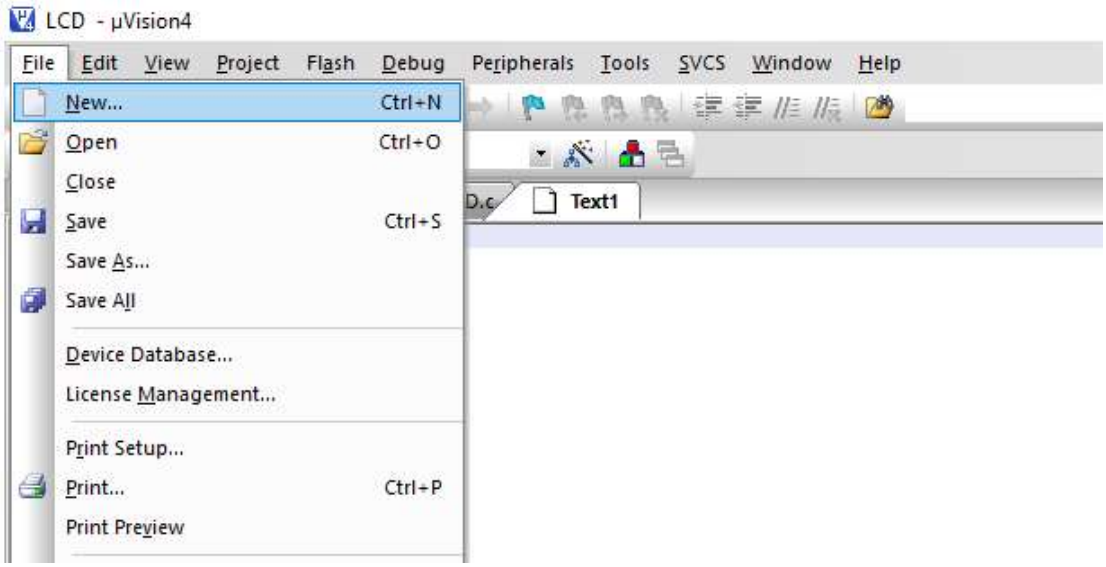
Trong này có hàng loạt các hãng điện tử sản xuất 8051. Chúng ta lập trình cho dòng chip nào thì chọn loại chip đó. Ở đây ta lập trình cho AT89C51 của hãng Atmel nên ta chọn như trên Hình 3.6.



Hình 3.6: Lựa chọn Vi điều khiển cần lập trình

Khi chọn chip thì ngay lập tức hiện ra bảng một số tính năng của chip chúng ta có thể nhìn thấy: 8051 based fully static 24Mhz …. nhập OK, chọn câu trả lời NO khi được hỏi "copy standard 8051 startup code to project and add file to project" vì nếu chọn YES chỉ làm cho file lập trình của chúng ta thêm nặng.

Để tạo một file lập trình có tên.c các chúng ta chọn file/new hoặc ấn ctrl+N như Hình 3.7.



Hình 3.7: Tạo cửa sổ lập trình

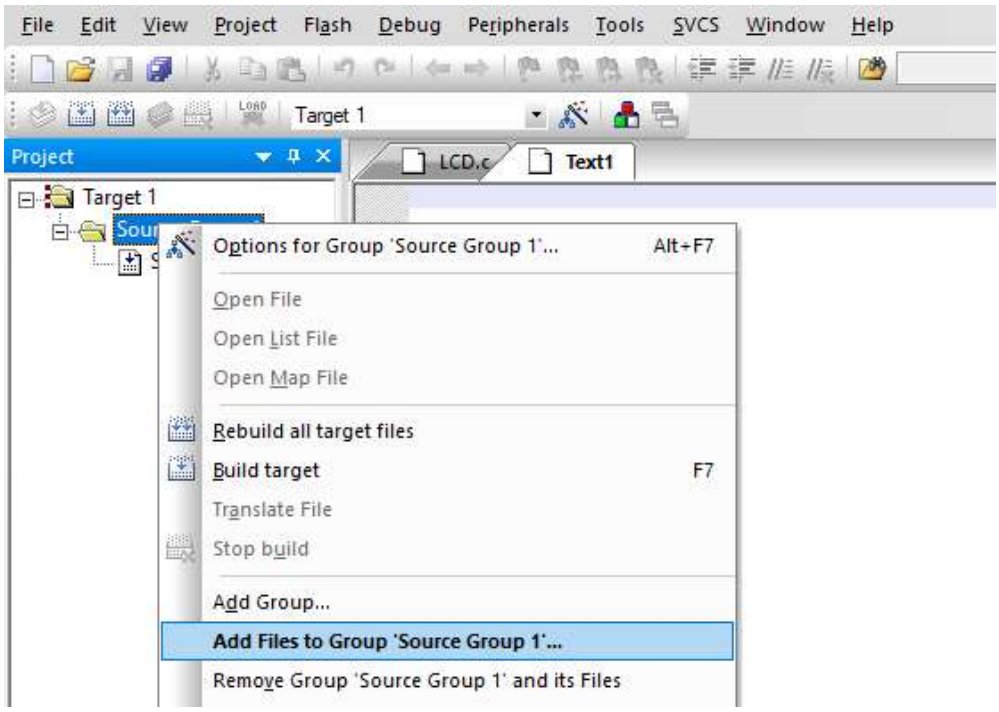
Cửa sổ text1 hiện ra. Tiếp theo chúng ta chọn File/save As hoặc Ctrl+S,

Trong ô bên trái màn hình, cửa sổ PROJECT WORKSPACE, chúng ta mở rộng cái target 1 ra như Hình 3.8.

Nhấp chuột phải nên SOURCE GROUP, chọn Add file to Group "Source Group 1" hộp thoại hiện ra chọn file.C mà chúng ta vừa SAVE rồi nhấn Add một lần rồi nhấn Close. Nếu chúng ta nhấn Add 2 lần phần mềm sẽ thông báo là file đã add, chúng ta chỉ việc OK rồi nhấn Close.

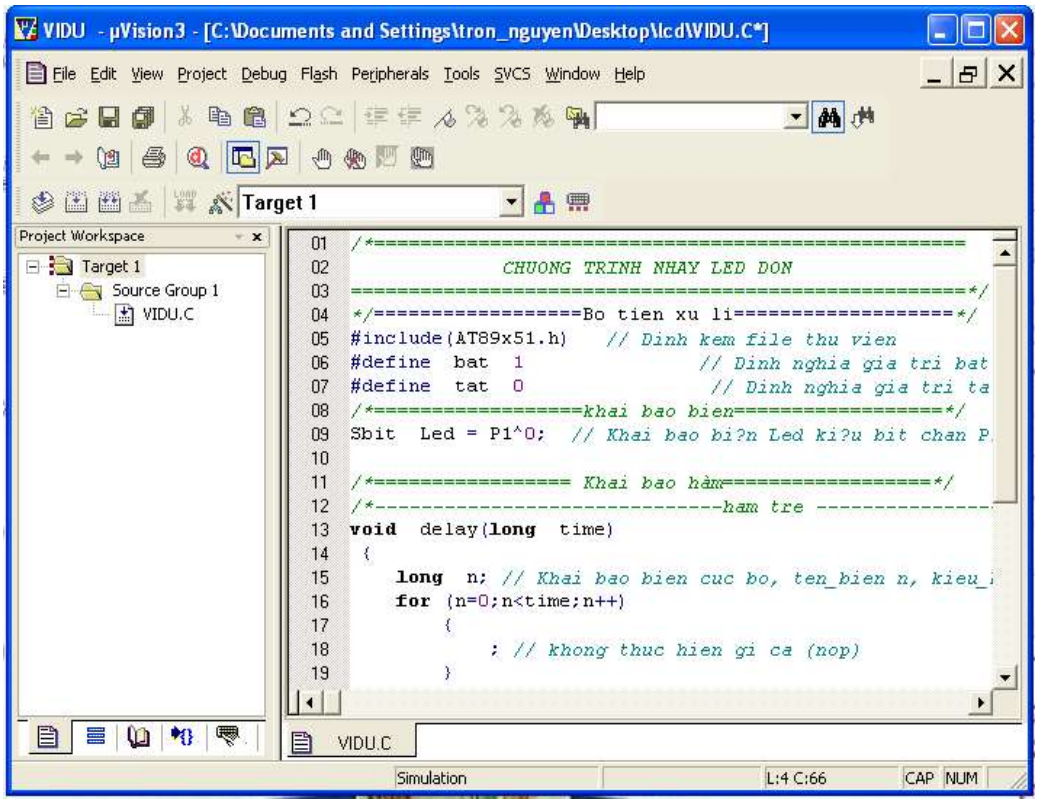
Bây giờ trong hình nhìn thấy trong Source Group 1 có file VIDU.C. Chúng ta nhấp chuột phải vào vùng soạn thảo file VIDU.C để thêm file thư viện. Chọn Insert "#include<REGX51.H>".

Phần cuối cùng của công việc khởi tạo là chúng ta viết lời giải thích cho dự án của mình. Phần này rất cần thiết vì nó để người khác hiểu mình làm gì trong project này và khi mình cần sử dụng lại code đọc lại còn biết nó là cái gì.



Hình 3.8: Đóng gói nhiều chương trình C trong một dự án

3.2.6.2. Soạn thảo chương trình.



Hình 3.9: Một chương trình lập trình C cho vi điều khiển

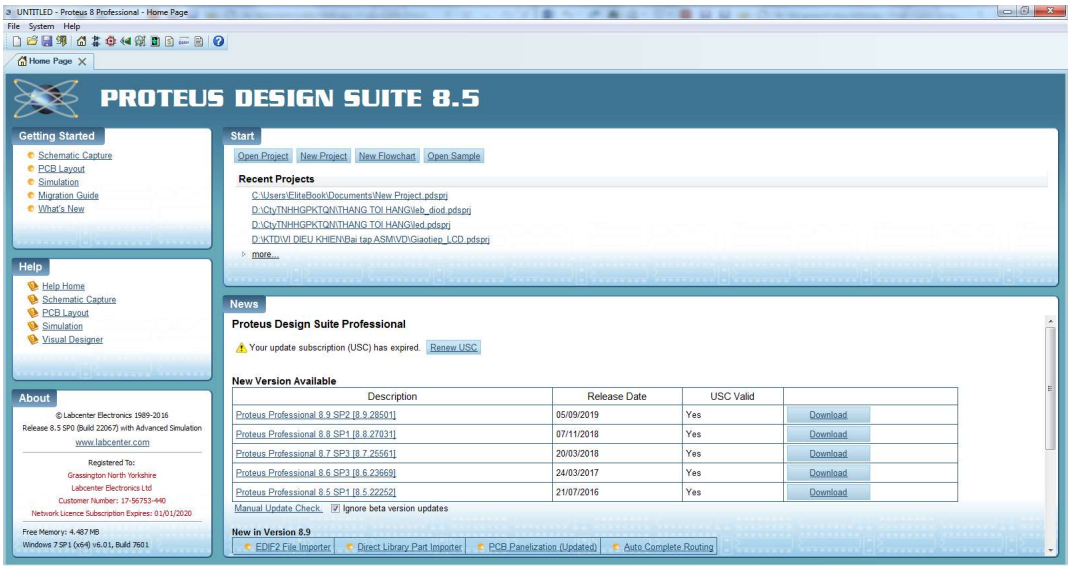
Chúng ta viết thử một chương trình làm ví dụ. Chú ý khi viết xong mỗi dòng

lệnh, ta nên giải thích ý nghĩa của dòng lệnh đó trong chương trình để dễ dàng cho việc đọc hiểu, kiểm tra, sửa lỗi và tối ưu chương trình như được thể hiện trong hình 3.11.

3.3. Biên dịch và nạp

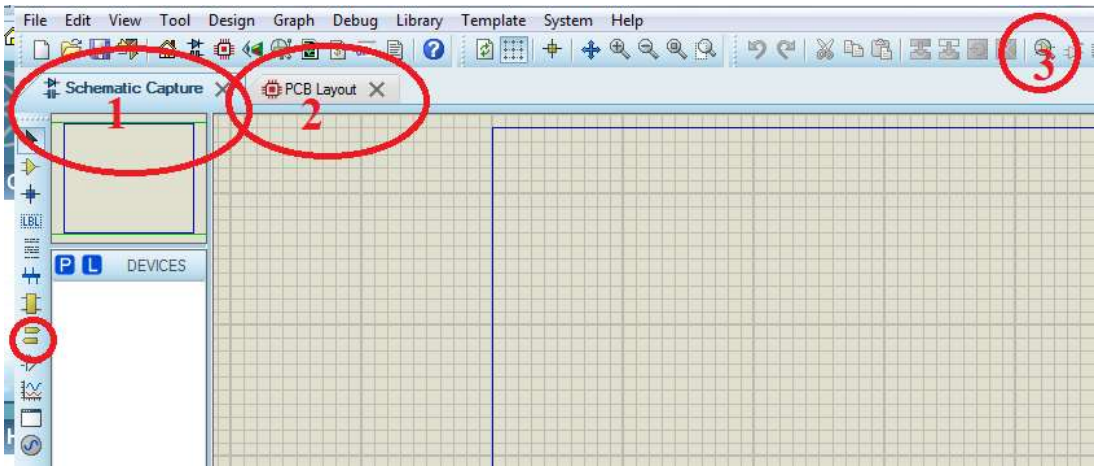
3.3.1. Mô phỏng chương trình

Khởi tạo chương trình Protues từ thanh Menu Start hoặc biểu tượng ngoài màn hình nền. Tạo dự án mới từ New Project, chọn đường dẫn đặt tên.



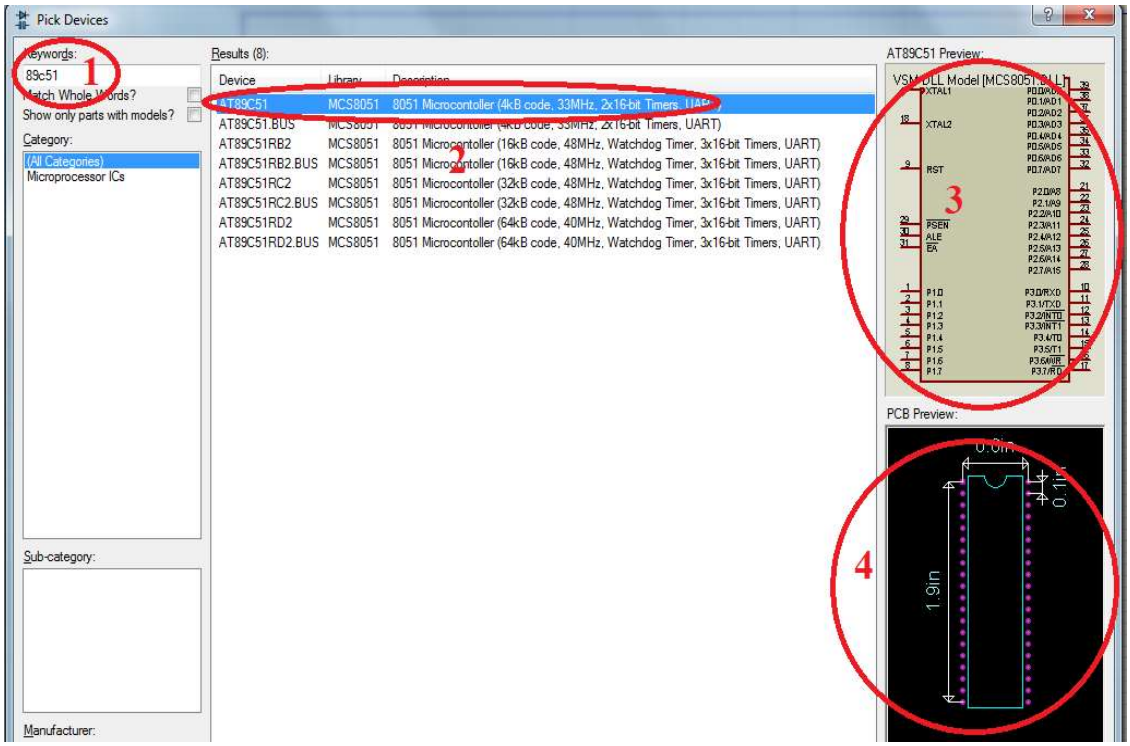
Hình 3.10: Giao diện Protues cho mạch nguyên lý và mạch in

Sau khi đặt tên và lựa chọn đường dẫn, bấm tiếp tục chương trình phần mềm sẽ hỏi lựa chọn mạch nguyên lý với các chức năng giao diện mặc định (DEFAULT) và tương tự mạch in PCB cũng chọn mặc định (DEFAULT), nếu chúng ta không chọn mặc định thì màn hình sẽ cho ra trang trắng, việc thiết lập trang nguyên lý và mạch in phải được thực hiện bởi người dùng.



Hình 3.11: Giao diện mạch nguyên lý

Vị trí tương ứng trên giao diện Hình 3.11 vùng 1, là giao diện thiết kế mạch nguyên lý, vùng 2 là thiết kế mạch in PCB, vùng 3 là nơi lấy linh kiện cho mạch nguyên lý cần thiết kế và vùng còn lại là chọn nguồn dương Power và đất Ground.



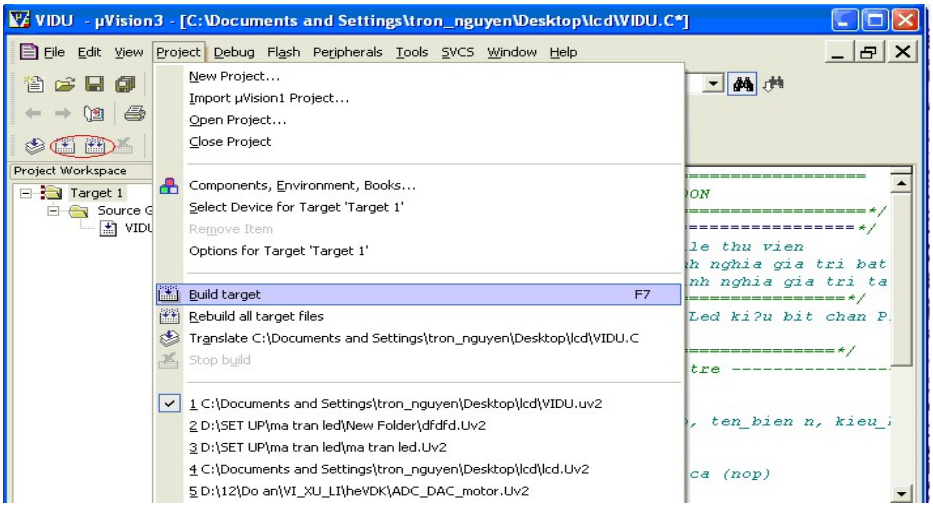
Hình 3.12: Đặc tính linh kiện được lựa chọn

Trên Hình 3.12 vùng 1 là sử dụng phím tắt để lựa chọn nhanh linh kiện so với tìm kiếm trên thanh công cụ dọc menu, vùng 2 là vùng lựa chọn linh kiện phù hợp đồng thời xuất hiện sơ đồ mạch nguyên lý và mạch in của linh kiện đó tương ứng với vùng 3, và vùng 4.

Sau khi nạp file hex đã lập trình và biên dịch, click đúp vào vi điều khiển để chọn file hex cần nạp vào chip;

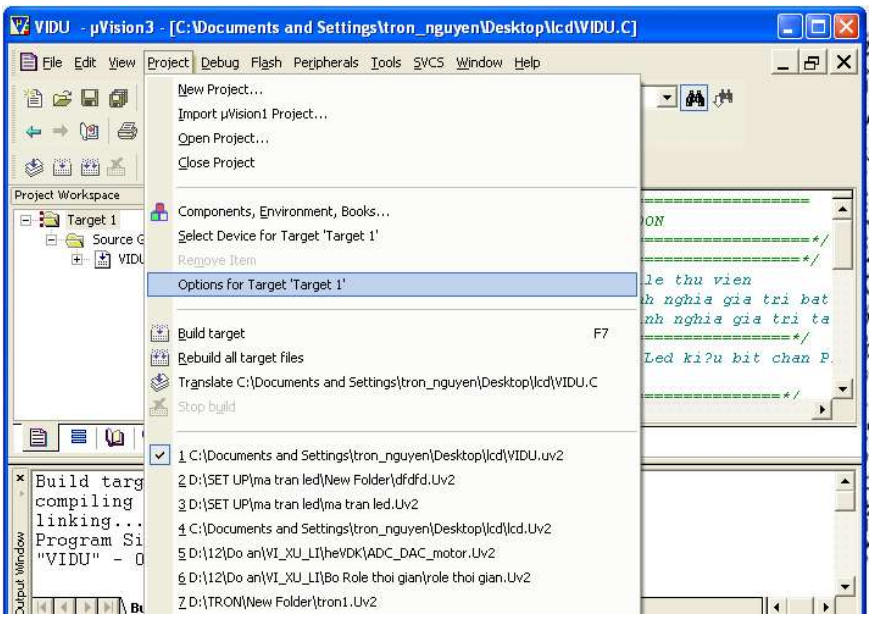
3.3.2. Công cụ biên dịch

Sau khi soạn thảo xong nhấn Ctrl+S để lưu. Sau khi lưu chúng ta biên dịch chương trình bằng cách ấn phím F7 hoặc chọn Build target là biểu tượng ngay trên cửa sổ Workspace, như trên hình:



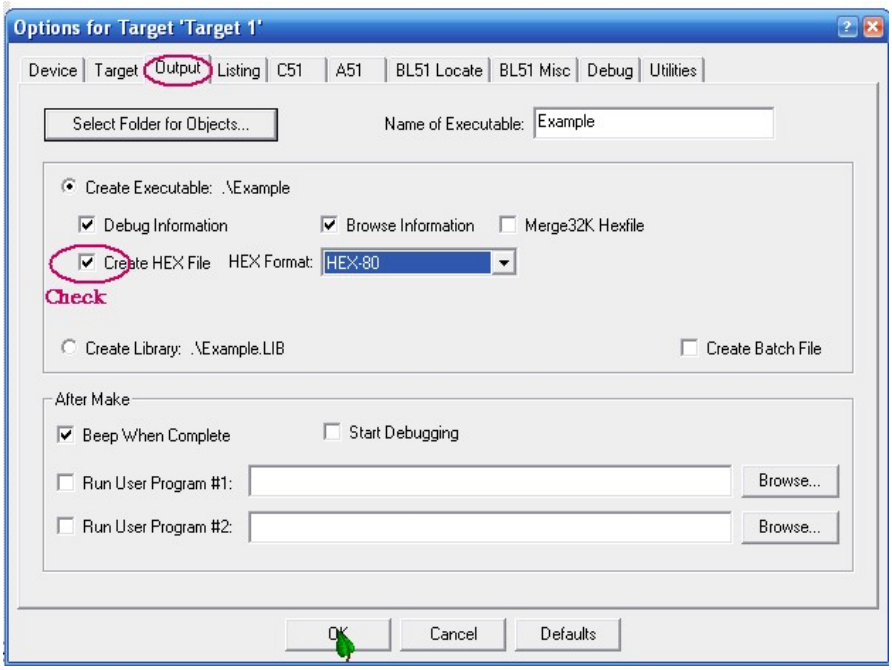
Hình 3.13: Biên dịch chương trình Keil C

Để biên dịch chương trình thành file HEX các chúng ta chọn: Project/option for ‘target 1’ như hình:



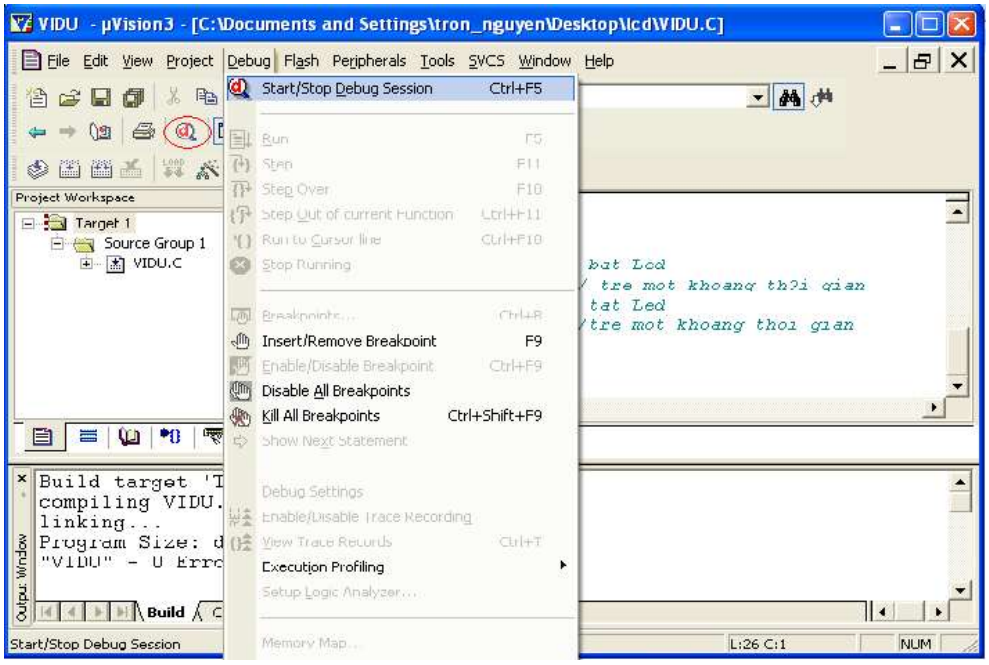
Hình 3.14: Đặt thuộc tính tạo file hex

Trong hộp thoại hiện ra, hãy check vào Creat HEX File như chỉ dẫn: Chọn thẻ menu Target và nhập lại tần số thạch anh là 12Mhz.



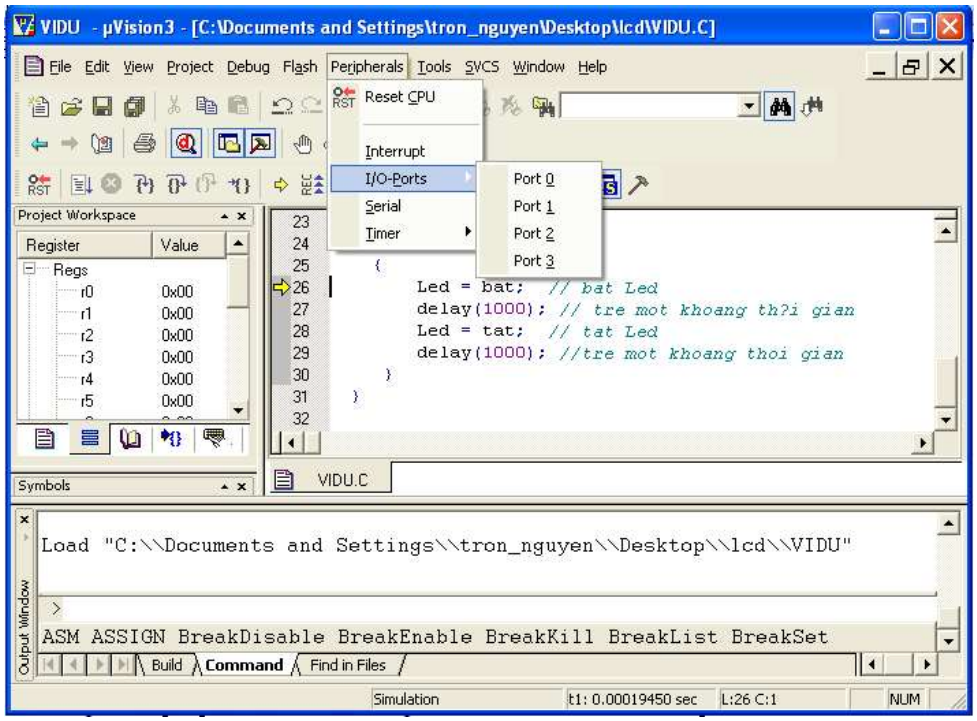
Hình 3.15: Tạo file HEX cho chương trình

Để mô phỏng, chúng ta chọn Debug/Start/stop debug session hoặc ấn Ctrl+F5, hoặc nhấn vào Icon chữ D màu đỏ trong cái kính lúp trên thanh công cụ.



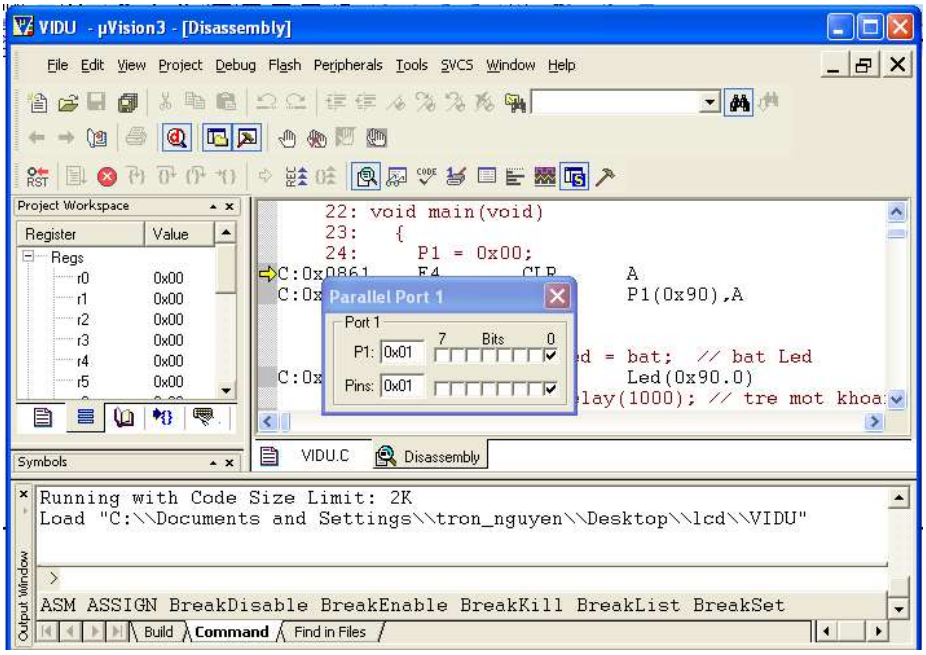
Hình 3.16: Chạy mô phỏng chương trình

Để hiển thị các cổng, các thanh ghi chúng ta chọn trong Peripherals.



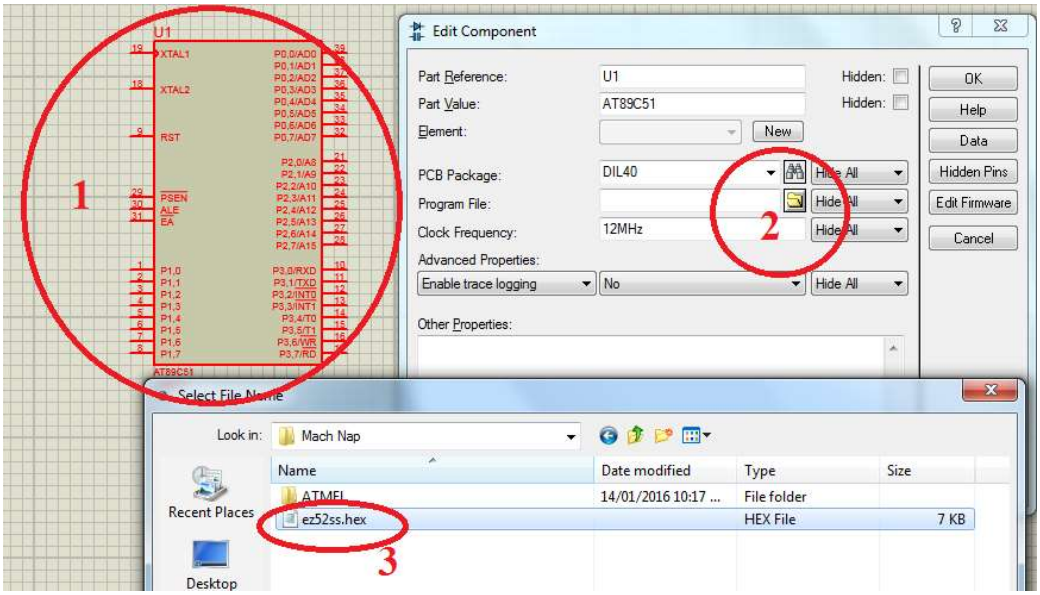
Hình 3.17: Kiểm tra trạng thái các thanh ghi trong vi điều khiển

Ta thấy 1 cửa sổ nhỏ Parallel Port xuất hiện. Đó là cái để mô phỏng cho một cổng của AT89C51, dấu tick tương đương chân ở mức cao (5V), không tick chân ở mức thấp 0V. Trong menu Peripherals còn có các ngoại vi khác như timer, interrupt, serial.



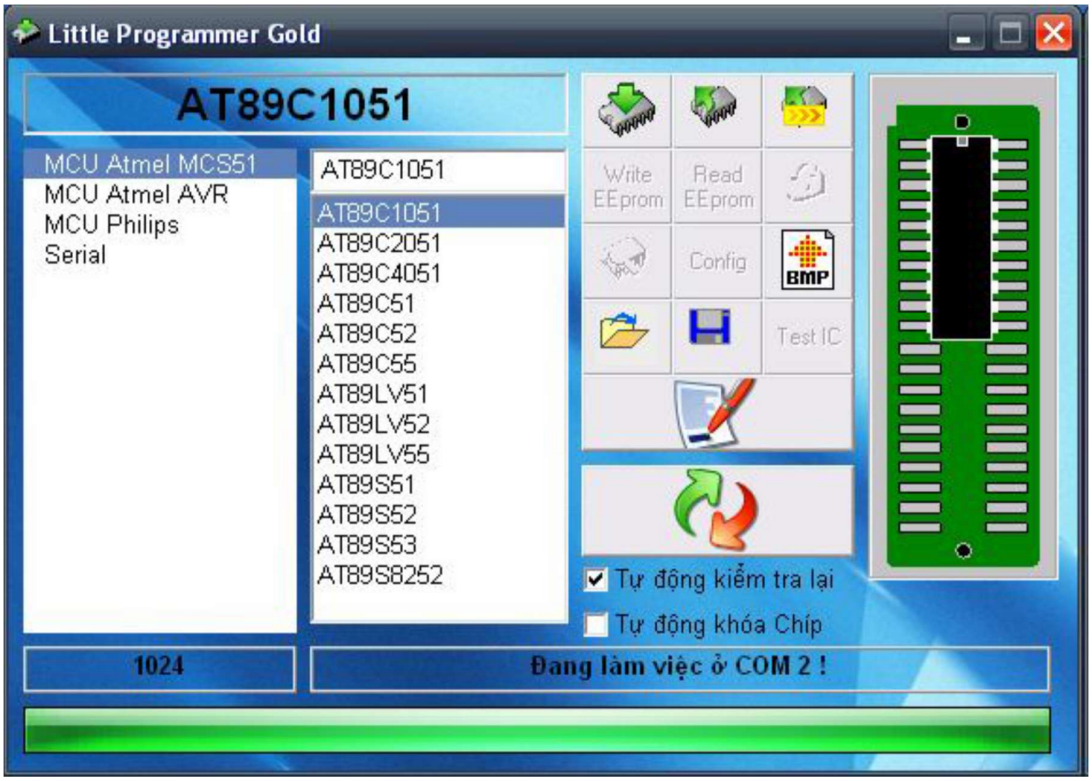
Hình 3.18: Kết quả gỡ rối chương trình

Để chạy chương trình chúng ta ấn chuột phải vào màn hình soạn thảo, rồi ấn F11, mỗi lần ấn sẽ chạy một lệnh, khi debug nếu chúng ta chờ hàm delay lâu quá 1000 lần lặp các chúng ta nhấn ctrl+F11 để bỏ qua hàm hoặc ấn F10 để chạy từng dòng lệnh.



Hình 3.19: Nạp file hex vào mô phỏng

Công cụ hỗ trợ nạp lên chip Little Programmer USB & COM



Hình 3.20: Lựa chọn chip cho mạch nạp

3.4. Bài tập và câu hỏi cuối chương

Câu 3.1: Viết chương trình 4 Led (nối vào các chân P1) sáng lần lượt cách nhau 0,5s. Chỉ có 1 LED sáng tại một thời điểm?

Câu 3.2: Viết chương trình nhấn nút Start (chân P1.7) động cơ (chân P2.0) chạy, nhấn nút Stop (chân P1.6) động cơ dừng?

Câu 3.3: Viết chương trình đếm sản phẩm sử dụng Counter 0, chế độ 16 bit, chân cảm biến vào P2.0 = "0" khi có sản phẩm và ngược lại. Khi số lượng đếm đủ 10 sản phẩm một đèn Led chân P3 được sáng lên đến 8 Led tương ứng 80 sản phẩm hệ thống quay trở lại từ đầu?

Tài liệu tham khảo chương 3

Tiếng Việt

[1] Ngô Diên Tập, ***Vi điều khiển với lập trình C***, Nhà xuất bản Khoa học Kỹ thuật, 2006.

Tiếng Anh

[2] Muhammad Ali Mazidi, Janice Gillispie Mazidi, Rolin D. McKinlay, ***The 8051 Microcontroller and Embedded Systems Using Assembly and C***, Department of Computer Science and Information Engineering National Cheng Kung University, Taiwan, 2003.

[3] Dogan Lbrahim, ***Microcontroller projects in C for the 8051***, Oxford Auckland Boston Johannesburg Melbourne New Delhi, 2011.

Chương 4. **THIẾT KẾ GIAO TIẾP NGOẠI VI**

4.1. Kết nối vào ra cơ bản

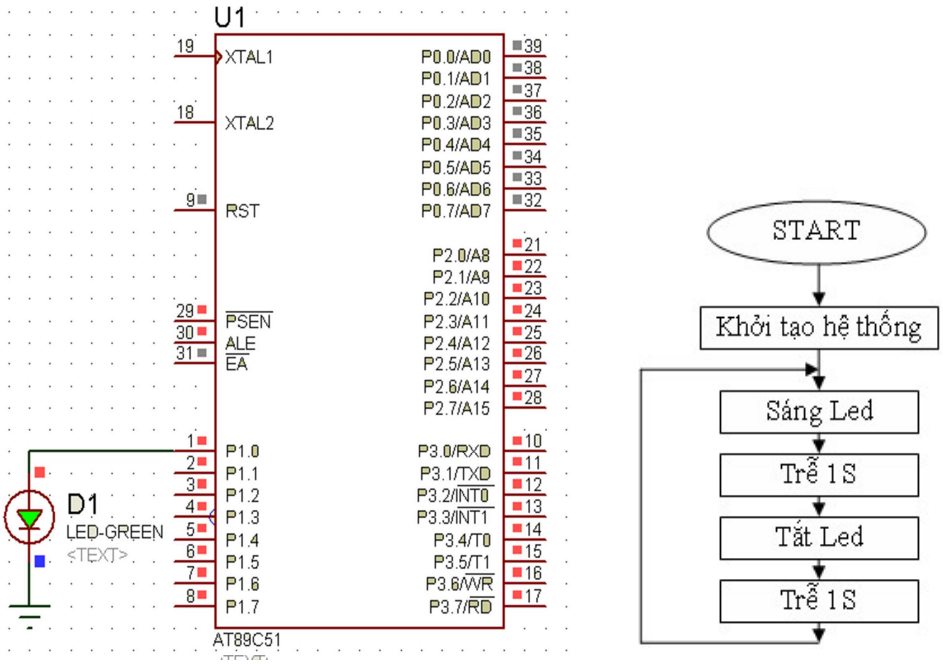
Một vài ứng dụng thực tế cần công suất rất lớn nên ta cần phải sử dụng khuếch đại Hình 4.1. Để đáp ứng được các dòng điện lớn, người ta thường dùng các transistor lưỡng cực công suất lớn như D718, B688, A671, D613... dòng điện cực đại cho phép có thể lên đến 10A.

4.1.1. Kết nối ra đơn bít

Thực hiện ghép nối đơn bit bằng cách cho giao tiếp họ 8051 với đèn LED ở chế độ xuất tín hiệu như trong ví dụ sau:

Ví dụ 4.1: Chương trình hợp ngữ điều khiển port 0 sáng tắt:

```
                                ; Chương trình chính
ORG    0000H                    ; Khai báo địa chỉ bắt đầu của chương trình
       MOV   P0, #00H           ; Nạp 00 vào port 0 để tắt 8 led
PORT_0: LCALL DELAY             ; Gọi chương trình con DELAY
       INC    P0                ; Tăng nội dung P0 lên 1
       LCALL DELAY              ; Gọi chương trình con DELAY
SJMP   PORT_0
                                ; Chương trình con DELAY
DELAY:      MOV   R6, #0FFH     ; Nạp hằng số FF vào thanh ghi R6
       DE2: MOV R7, #0FFH       ; Nạp hằng số FF vào thanh ghi R7
       DE1: DJNZ R7, DE1        ; Giảm R7 đi 1 đơn vị và nhảy đến DE1 khi
R7≠0)
            DJNZ R6, DE2 ; Giảm R6 đi 1 đơn vị và nhảy đến DE2 khi R6≠0)
RET                             ; Thoát khỏi chương trình con
END
; Chương trình con DELAY dùng timer
DELAY:      CLR    TF0          ; Xóa cờ ngắt của Timer 0
       MOV    TL0, # 00H       ; Nạp 0 vào TL0
       MOV    TH0, # 00H       ; Nạp 0 vào TH0
       MOV    TMOD, # 01       ; Khởi tạo timer T0 mode 1 bộ đếm 16 bit
       SETB   TR0              ; Cho phép timer 0 bắt đầu đếm xung
DE1:   JBN    TF0, DE1         ; Kiểm tra cờ tràn
RET.
```



Hình 4.1: Sơ đồ đấu nối Led đơn với họ 8051 và thuật toán xuất đơn bit

Ví dụ 4.2: Ghép nối LED đơn với chân P1.0 của vi điều khiển, viết chương trình điều khiển LED nhấp nháy với thời gian trễ là 1s.

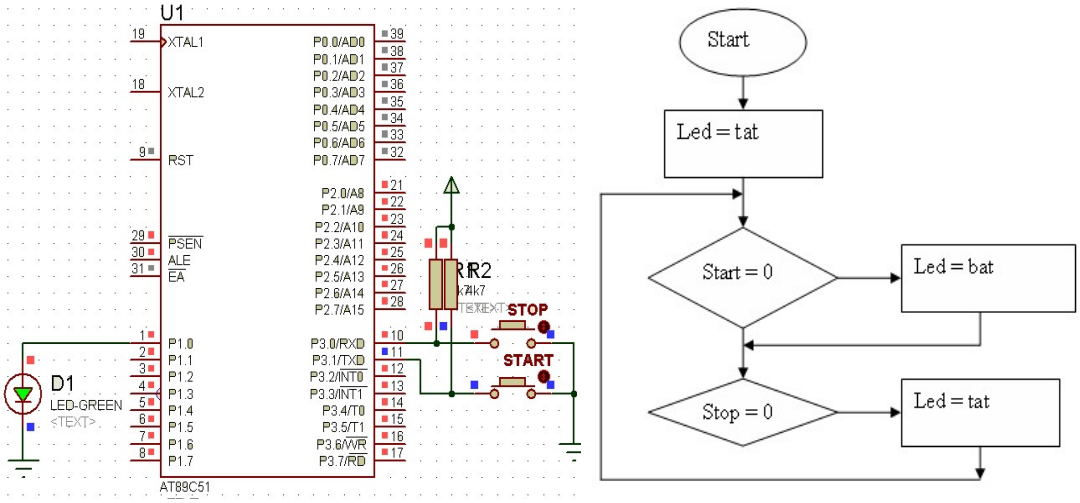
+ Chương trình C điều khiển:

```
*/========== Tiền xử lý ====================*/
#include<AT89x51.h> // Đính kèm file thư viện
#define bat 1            // Định nghĩa bật LED
#define tat 0            // Định nghĩa tắt LED
/*==========Khai báo biến ================*/
sbit Led = P1^0;        // Khai báo biến Led kiểu sbit ở gán chân P1.0
/*============ Khai báo hàm ================*/
/*-----------Hàm trễ ---------------------------------*/
void delay(long time)
 {
       while(time--);
 }
/*-----------------------------Chương trình chính --------------------*/
void main(void)
        {
              while(1)
                {
                        Led = bat;              // Bật Led
                        delay(25000); // Trễ 1s
```

```
                    Led = tat;              // Tắt Led
                    delay(25000); // Trễ 1s
                }
            }
```

4.1.2. Kết nối vào đơn bít

Tương tự chúng ta thực hiện kết nối nhập đơn bit nút nhấn Start, Stop như trong Hình 4.2. Cụ thể, một đèn Led được nối với chân P1.0 của vi điều khiển. Một công tắc Start nối với chân P3.0 và công tắc Stop nối với chân P3.1 của vi điều khiển. Hãy viết chương trình điều khiển để khi bật công tắc Start thì Led sáng, khi bật công tắc Stop thì Led tắt.



Hình 4.2: Sơ đồ nối họ 8051 kết hợp xuất nhập đơn bit

Ví dụ 4.3: Chương trình điều khiển kết nối vào đơn bit:

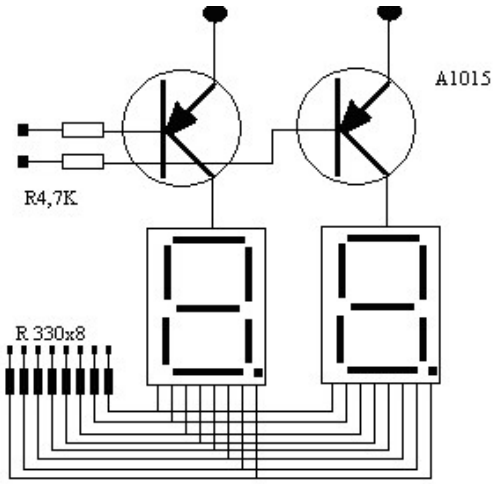
```
#include<AT89x51.h>      // Đính kèm file thư viện
#define bat 1            // Định nghĩa giá trị bật đèn Led
#define tat 0            // Định nghĩa giá trị tắt đèn Led
sbit Led = P1^0;         // Khai báo biến Led kiểu bit chân P1.0
sbit STOP = P3^0;        // Công tắc STOP để tắt Led
sbit START = P3^1;               // Công tắc START để tắt Led
void main(void)
{
        Led = tat;               // Ban đầu tắt Led
       while(1){
          if((START==0)&&(STOP= = 1)) {
             Led = bat;
```

```
            }
            if((START= = 1)&&(STOP==0)){
                Led = tat;
            }
        }
    }
```

4.2. Giao tiếp mã quét

4.2.1. Hiển thị LED 7 đoạn

Trong đó các đường dữ liệu được nối chung lại với nhau và nối tiếp với 8 điện trở hạn dòng, giá trị điện trở này tuỳ thuộc vào điện áp: 5V-330Ω, 12V-1kΩ, 24V-2,2kΩ. Như vậy tại một thời điểm chỉ có một Led được sáng và luân phiên nhau, với tốc độ luân phiên nhanh thì mắt người không phân biệt được chớp tắt của từng Led ở tần số lớn hơn 24Hz, lúc đó xem như cả hai Led cùng sáng đồng thời.



Hình 4.3: Sơ đồ nguyên lý nối 2 Led 7 đoạn đếm số

Bảng 4.1: Mã hình Led 7 đoạn

Các số hiển thị	P1.0 g	P1.1 f	P1.2 e	P1.3 d	P1.4 c	P1.5 b	P1.6 a	dp	số nạp hex mov P1,#
0	1	0	0	0	0	0	0	1	81
1	1	1	1	1	0	0	1	1	cf
2	0	1	0	0	1	0	0	1	92
3	0	1	1	0	0	0	0	1	86

Các số hiển thị	P1.0 g	P1.1 f	P1.2 e	P1.3 d	P1.4 c	P1.5 b	P1.6 a	dp	số nạp hex mov P1,#
4	0	0	1	1	0	0	1	1	cc
5	0	0	1	0	0	1	0	1	a4
6	0	0	0	0	0	1	0	1	a0
7	1	1	1	1	0	0	0	1	8f
8	0	0	0	0	0	0	0	1	80
9	0	0	1	0	0	0	0	1	84

Một trường hợp khác cho việc kết nối Led 7 đoạn sử dụng công tắc điều khiển

Gạt switch 2 lên ON để kích hoạt Port 3 là các nút nhấn.

Gạt switch 4 lên ON để kích hoạt Led 7 đoạn.

Port 2 dùng để chọn Led 7 đoạn nào sẽ được sáng và Port 0 là dữ liệu cho Led 7 đoạn đó. Với kết nối phần cứng như vậy, để hiển thị số 5 thì dữ liệu xuất ra sẽ là ***0x6D*** (0110 1101). Tương tự, giá trị cho các số từ 0 đến 9 sẽ là:

unsigned char Led7_data[10] =
{0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};

Để quét Led, sơ đồ nguyên lý sẽ được mắc như **Error! Reference source not found.**. Port 2 gọi là port dữ liệu, dữ liệu này sẽ được nối với tất cả các Led. Port 1 dùng để chọn Led, cực E của transitor được nối với chân mass của Led 7 đoạn. Port 1 bằng 1 tại bit nào thì transitor tương ứng sẽ dẫn. Giả sử Port 1 = 0x80, transitor T4 dẫn, dòng data sẽ đi qua Led 7 đoạn nối với T4 xuống đất, và Led này sẽ sáng, các Led còn lại thì không. Khi số lượng Led càng nhiều, ta phải tính toán để tìm ra số t_0 hợp lý. Thông thường, khoảng thời gian tối đa giữa Led đầu tiên và Led cuối cùng trong khi quét Led phải nhỏ hơn 1/50 giây (50Hz). Để quét 8 Led 7 đoạn trong trường hợp này, ta sẽ dùng 1 buffer có 8 phần tử, rồi định kì xuất từng phần tử ra Led 7 đoạn tương ứng. Việc thay đổi dữ liệu trong buffer này sẽ do các hàm được cung cấp cho người dùng sử dụng cập nhật. Ta sẽ dùng ngắt timer để định kì gọi hàm scan_Led7(), hàm này có chức năng xuất 1 giá trị trong buffer ra Led tương ứng, sau mỗi lần gọi, index sẽ tăng lên 1, đến khi bằng 8 sẽ quay trở về 0.

Viết chương trình để module này có thể dùng lại cho các ứng dụng khác, ta sẽ

hiện thực các hàm dưới đây:

void init_Led7(): Khởi tạo các thông số ban đầu

void clear_Led7(): Xoá dữ liệu đang hiển thị trên Led 7 đoạn

void set_position(unsigned int pos): Thiết lập vị trí xuất dữ liệu, vị trí 0 là Led ngoài cùng bên trái.

unsigned int get_position(): Truy xuất vị trí đang xuất dữ liệu.

void put_number(unsigned int num): Xuất giá trị num (là số unsigned int) ra Led 7 đoạn từ vị trí hiện tại.

void put_string(char strNum[]): Xuất giá trị strNum (là giá trị char) ra Led 7 đoạn từ vị trí hiện tại.

void scan_Led7(): Xuất dữ liệu từ buffer ra Led 7 đoạn.

Các hàm *interface* này được khai báo trong file *Led7.h. File Led7.c* sẽ thực hiện các hàm này.

```
Hàm init_Led7()
void init_Led7()
{
    P0 = 0x00;              // Port dữ liệu
    P2 = 0x00;              // Port lựa chọn quét
    position = 0;
    Led7_index = 0;
    is_valid_data = 1;
}
```

***** Giải thích:

- P0: Port để xuất dữ liệu cho Led 7 đoạn, khởi tạo 0x00 tức là không có Led nào trong Led 7 đoạn sáng.

- P2: Port để chọn Led 7 đoạn nào trong 8 Led sẽ nhận data từ Port 0, tích cực mức "1", P2 = 0x01 (0000 000***1***) tức là Led 7 đoạn ngoài cùng bên trái sẽ sáng, P2 = 0x20 (0000 00***1***0) tức là Led thứ hai tính từ trái sẽ sáng.

- Position: Là vị trí bắt đầu hiển thị giá trị, chẳng hạn người dùng muốn hiển thị số 123 từ vị trí thứ hai tính từ trái thì phải thiết lập position là "1" (position = 0 là Led ngoài cùng bên trái). Biến position này người dùng phải dùng hàm set_positon thì mới thay đổi được.

- Led7_index: vị trí sẽ xuất data, biến này được dùng trong hàm scan_Led, người dùng không can thiệp vào biến này được.

- is_valid_data: biến dùng trong hàm scan_Led, biến này để tạm dừng việc quét Led trong khi đang thay đổi buffer hiển thị, người dùng cũng không can thiệp được vào biến này.

Hàm clear_Led7()

```
void clear_Led7()
{
    Led7_buffer[0] = 0x00;
    Led7_buffer[1] = 0x00;
    Led7_buffer[2] = 0x00;
    Led7_buffer[3] = 0x00;
    Led7_buffer[4] = 0x00;
    Led7_buffer[5] = 0x00;
    Led7_buffer[6] = 0x00;
    Led7_buffer[7] = 0x00;
}
```

Hàm này chỉ đơn giản là xoá *buffer* 8 phần tử, lúc đó toàn bộ các Led sẽ tắt. Buffer này dùng để chứa dữ liệu của 8 Led 7 đoạn, dữ liệu của từng Led sẽ được định kì đưa ra Led tương ứng.

Hàm set_position(unsigned int pos)

```
void set_position(unsigned int pos) // Thiết lập vị trí đầu ra
{
    position = pos;
}
```

Hàm này sẽ lấy thông số từ người dùng để cập nhật cho biến position.

Hàm get_position()

```
unsigned int get_position()          // Lấy vị trí hiện tại
{
    return position;
}
```

Hàm này trả về vị trí đang xuất dữ liệu.

Hàm put_Number(unsigned int num)

```
void put_number(unsigned int num)
{
```

```
    int i;
    is_valid_data = 0;                  // Disable scan Led
    for(i=position; i<8; i++)
    {
        if(i>=0)
        {
            Led7_buffer[i] = Led7_data[num % 10];
        }
        num = num / 10;
        if(num == 0)
            break;
    }
    position = i;
    is_valid_data = 1;                  // Enable scan Led
}
```

Hàm này dùng để cắt từng chữ số của số ***num*** để bỏ vào buffer tương ứng. Vòng lập for dùng để hiện thực tác vụ này, chữ số hàng đơn vị của num sẽ được bỏ vào buffer vị trí ***position***.

Hàm put_string(char strNum[])

Hàm này cũng gần giống với hàm put_Number, chỉ khác đối số là kiểu char[]. Việc thao tác trên dữ liệu kiểu char[] cần phải include thêm thư viện ***string.h***

```
Hàm scan_Led7
void scan_Led7()
{
    if(is_valid_data)
    {
        P2 = 0x00;                          // Remove noise
        P0 = Led7_buffer[Led7_index];       // Data for the next Led
        P2 = 1 << Led7_index;                       // Enable data
        Led7_index++;                               // Update new index
        if(Led7_index == 8)
            Led7_index = 0;
    }
}
```

Đây là hàm dùng để quét Led, và sẽ được ngắt timer gọi. Mục đích của hàm này

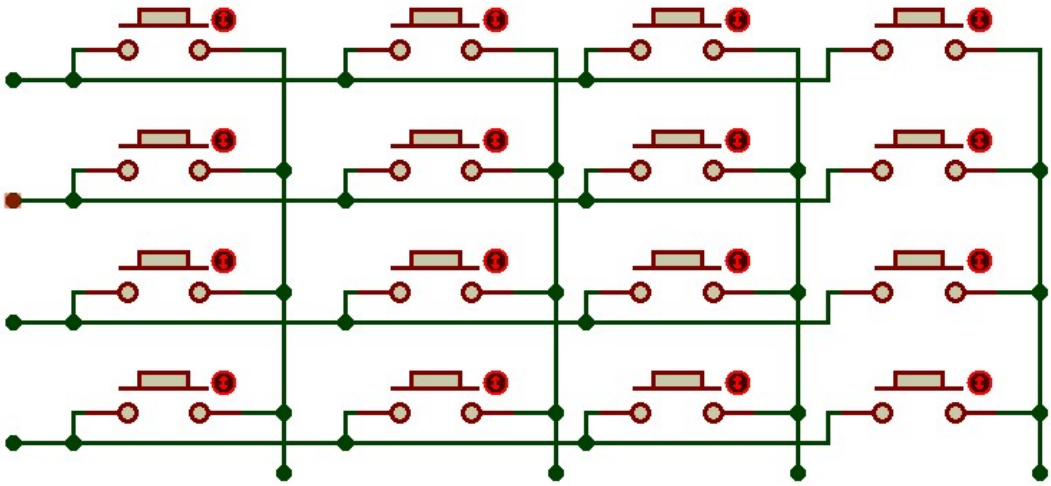
là xuất dữ liệu ra Led ở vị trí tiếp theo. Port 2 dùng để chọn Led, và để tránh hiện tượng bóng mờ khi quét Led, chúng ta phải gán nó về 0x00 trước. Led7_index là biến chỉ vị trí của Led hiện tại chuẩn bị nhận dữ liệu.

Hàm scan_Led7() được gọi trong ngắt timer0. Với các hàm đã thiết kế ở trên, để xuất số 1234 tại vị trí thứ 2 tính từ trái sang ta viết trong hàm main() như sau:

```
void main(){
    init_main();
    init_timer0();
    init_Led7();
    set_position(1);
    put_Number(1234);
    while(1){};}
```

4.2.2. Giao tiếp ma trận

4.2.2.1. Giao tiếp phím ma trận



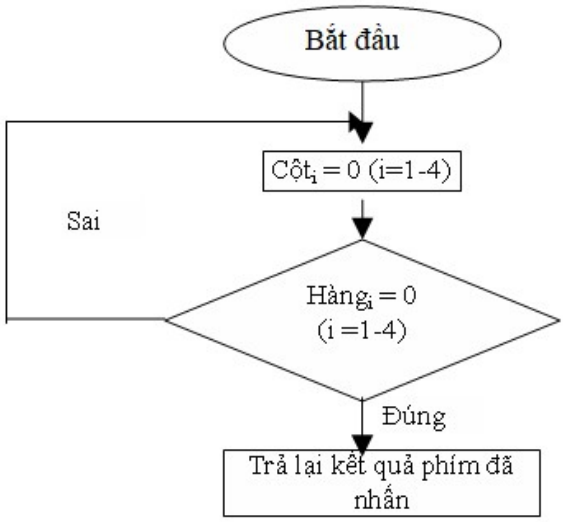
Hình 4.4: Sơ đồ ghép nối phím ma trận với vi điều khiển

Kết nối bàn phím cho phép người sử dụng có thể nhập dữ liệu và thông qua các phím chức năng điều khiển hoạt động của vi điều khiển. Để quản lý được các phím của bàn phím mà không làm ảnh hưởng nhiều tới quá trình thực hiện tính toán của vi điều khiển chính, người ta sử dụng riêng một vi điều khiển loại nhỏ để quản lý bàn phím. Bàn phím được xây dựng theo kiểu ma trận, gồm 4 hàng x 5 cột.

- 4 hàng được lấy từ các chân P1.0 - P1.3 của vi điều khiển
- 5 cột được lấy từ các chân P1.4 - P1.7 của vi điều khiển và cột cuối cùng nối đất.

Tổ chức các phím gồm các phím từ 0 -9 và từ A- F làm thành phần nhập dữ liệu. Các phím chức năng F1, F2 là phím lựa chọn các lệnh trên menu, phím BACK SPACE là phím xóa, giúp chỉnh sửa khi nhập liệu nhầm. Phím ENTER để kích hoạt lệnh hiện hành, tùy theo ngữ cảnh, được hiển thị trên LCD. Sơ đồ nguyên lý của bàn phím như hình:

Thuật toán đọc bàn phím được xây dựng trên Hình 4.5



Hình 4.5: Thuật toán giao tiếp bàn phím với 8051

4.2.2.2. Giao tiếp Led ma trận

LED matrix 8x8: là một bảng 64 điểm LED bố trí theo kiểu ma trận 8 hàng x 8 cột, tại mỗi điểm 2 LED xanh và đỏ đấu chung Anôt, mỗi hàng gồm 8 điểm cũng đấu chung Anôt với nhau. Dựa trên nguyên tắc như quét màn hình, ta có thể thực hiện việc hiển thị ma trận đèn bằng cách quét theo hàng và quét theo cột. Mỗi Led trên ma trận LED có thể coi như một điểm ảnh. Địa chỉ của mỗi điểm ảnh này được xác định đồng thời bởi mạch giải mã hàng và giải mã cột, điểm ảnh này sẽ được xác định trạng thái nhờ dữ liệu đưa ra từ bộ vi điều khiển 89C51.

Như vậy tại mỗi thời điểm chỉ có trạng thái của một điểm ảnh được xác định. Tuy nhiên khi xác định địa chỉ và trạng thái của điểm ảnh tiếp theo thì các điểm ảnh còn lại sẽ chuyển về trạng thái tắt. Vì thế để hiển thị được toàn bộ hình ảnh của ma trận đèn, ta có thể quét ma trận nhiều lần với tốc độ quét rất lớn, lớn hơn nhiều lần thời gian kịp tắt của đèn. Mắt người chỉ nhận biết được tối đa 24 hình/s do đó nếu tốc độ quét rất lớn thì sẽ không nhận ra được sự thay đổi nhỏ của đèn mà sẽ thấy được toàn bộ hình ảnh cần hiển thị.

Thực hiện từng cột, đầu tiên chọn cột 1, đưa điện áp cột 1 về 0. Sau đó chọn và quét lần lượt các hàng 1, 2, 3, 4, 5, 6, 7, 8 như sau:

- Đèn 1 tắt → Điện áp đưa vào hàng 1 là 0V.
- Đèn 2 tắt → Điện áp đưa vào hàng 2 là 0V.
- Đèn 3 sáng → Điện áp đưa vào hàng 3 là 5V.
- Đèn 4 sáng → Điện áp đưa vào hàng 4 là 5V.
- Đèn 5 sáng → Điện áp đưa vào hàng 5 là 5V.
- Đèn 6 sáng → Điện áp đưa vào hàng 6 là 5V.
- Đèn 7 sáng → Điện áp đưa vào hàng 7 là 5V.
- Đèn 8 sáng → Điện áp đưa vào hàng 8 là 5V.

Chọn cột 2, nối đất. Sau đó quét lần lượt các hàng 1, 2, 3, 4, 5, 6, 7, 8.

- Đèn 1 tắt → Điện áp đưa vào hàng 1 là 0V.
- Đèn 2 sáng → Điện áp đưa vào hàng 2 là 5V
- Đèn 3 tắt → Điện áp đưa vào hàng 3 là 0V
- Đèn 4 sáng → Điện áp đưa vào hàng 4 là 5V
- Đèn 5 tắt → Điện áp đưa vào hàng 5 là 0V
- Đèn 6 tắt → Điện áp đưa vào hàng 6 là 0V
- Đèn 7 tắt → Điện áp đưa vào hàng 7 là 0V
- Đèn 8 tắt → Điện áp đưa vào hàng 8 là 0V

Tiếp tục quét với các cột từ 3 đến 8 bằng cách như trên, sau đó chuyển sang quét đèn LED thứ hai và thứ ba một cách tương tự.

Để mắt người nhận biết được toàn bộ hình ảnh của ma trận ta phải tiến hành quét nhiều lần. Do mắt người không phân biệt được sự thay đổi ảnh nếu ảnh đó được quét với tốc độ 24 hình/s trở lên, nên nếu ta quét ảnh với tốc độ lớn hơn hoặc bằng 24 hình/s thì ảnh sẽ chạy liên tục và không bị giật.

Nguyên lý điều khiển ma trận Led: Ma trận Led hai màu được cấu tạo gồm những điểm sáng, mỗi điểm sáng có hai bóng đèn đỏ và xanh lá bên trong. Khi cả 2 đèn này cùng sáng cho ta cảm giác màu vàng.

Viết chương trình: Chương trình được tổ chức chức ba module: MAIN để chứa file main.c, TIMER để gọi hàm quét ma trận Led và Led ma trận để chứa các hàm liên quan đến ma trận Led.

Các hàm về ma trận Led như sau:

void init_Led_matrix(); *// Khởi tạo ma trận Led*

```
void scan_Led_matrix(); // Quét ma trận Led, hàm này được gọi trong timer.
void update_display_Led_matrix();                    // Cập nhật buffer hiển thị.
Hàm init_Led_matrix()
void init_Led_matrix()
{
    P0 = 0x00;                                  // Red data
    P2 = 0x00;                                  // Green data
    index_Led_matrix = 0;              // Index use to scan Led
    alphabet_index = 208;// In the initial, Led matrix doesn't display anything
}
```

Trong đó:

- *P0*: Port để xuất dữ liệu có màu đỏ cho ma trận Led.
- *P2*: Port để xuất dữ liệu có màu xanh cho ma trận Led.
- *index_Led_matrix*: biến dùng để quét dữ liệu hiển thị trên ma trận Led, ta dùng biến này để lấy dữ liệu trong hai buffer (mỗi buffer 8 phần tử) để hiển thị và chọn cột.
- *alphabet_index*: biến dùng để lấy dữ liệu cần hiển thị, bỏ vào hai buffer đỏ và xanh. Mảng dữ liệu này thường khá lớn và được khai báo trong file *table_Led_matrix.h*.

Hàm *scan_Led_matrix():* Hàm này được gọi trong timer, có nhiệm vụ xuất buffer hiển thị ra ma trận Led, định kì xuất từng phần tử của buffer ra cột tương ứng.

```
void scan_Led_matrix()
{
    P1 = 0;      // Remove noise
    P0 = red_buff[index_Led_matrix];
    P2 = green_buff[index_Led_matrix];
    P1 = 1 << index_Led_matrix;
    index_Led_matrix = (index_Led_matrix + 1) % 8; // Next column
}
```

Hàm *update_display_Led_matrix():* Hàm này dùng để thay đổi nội dung hiển thị, các ứng dụng của người dùng chủ yếu là thay đổi code ở hàm này.

```
void update_display_Led_matrix()
{
    char i;
    for(i = 0; i < 8; i++)
```

```
    {
        red_buff[i]=alphabet_upcase_Led_matrix[i+alphabet_index ];
        green_buff[i]=alphabet_upcase_Led_matrix[i+alphabet_index];
    }
}
```

Hàm main chỉ việc thay đổi *alphabet_index* (dữ liệu của chữ cái kế tiếp) rồi gọi hàm *update_display_Led_matrix(),* định kì 1s sau đó thay đổi chữ kế tiếp:

```
while(1)
{
    alphabet_index = (alphabet_index + 8) % 216;
    update_display_Led_matrix();
    delay_ms(1000);
}
```

Viết chương trình chữ chạy trên ma trận Led, cung cấp giao diện giúp người dùng có thể thay đổi dễ dàng. Nguyên lý chạy chữ trên ma trận Led. Để chạy một dòng chữ qua ma trận Led, chúng ta sẽ có một buffer lưu toàn bộ dòng chữ đó. Buffer này thường là một mảng các byte. Chương trình sẽ định kì cắt một phần trong buffer này đổ dữ liệu vào buffer nhỏ hơn dùng để quét Led. Chương trình quét Led sẽ hiển thị ra Led ma trận.

Tại thời điểm T1, dữ liệu đổ vào buffer quét Led là chữ "H". Tại thời điểm T2, một phần chữ H và E được đổ vào buffer này, và tại thời điểm T3 là chữ "E". Nếu khoảng thời gian giữa các thời điểm là nhỏ, chúng ta sẽ thấy hiệu ứng dòng chữ chạy qua ma trận Led.

Để sinh ra được buffer chứa toàn bộ dòng chữ, chúng ta phải xử lý dữ liệu đã lưu sẵn (tạm gọi là font chữ), ghép nối chúng sao cho đẹp mắt. Chẳng hạn muốn hiện chữ "HELLO WORLD", chúng ta phải làm như sau:

Lấy font của chữ "H", bỏ những cột trống ở đầu và cuối, phần còn lại bỏ vào buffer.

Lấy font của chữ "E", bỏ những cột trống ở đầu và cuối, thêm một byte 0x00 vào buffer (tạo một nét rời giữa H và E) rồi bỏ dữ liệu của E vào.

Tương tự, hết chữ "O", chúng ta thêm khoảng ba byte 0x00.

Tuỳ vào tài nguyên của hệ thống, chúng ta định nghĩa ra độ dài tối đa của buffer này. Trong quá trình sinh ra buffer, chương trình sẽ cập nhật độ dài hiện tại và sẽ

dừng việc ghép chữ nếu độ dài là quá mức cho phép.

Viết chương trình: Chương trình sẽ gồm ba nhóm là *TIMER*, *LED MATRIX* và *MAIN*. Các nhóm *TIMER* và *LED MATRIX* được dùng lại. Trong file *Led_matrix.h* ta khai báo thêm một hàm cung cấp cho người dùng:

void set_message(char strMsg[]);

Hàm này cho phép người dùng thay đổi nội dung chữ chạy qua ma trận Led. Như gọi *set_message ("HELLO WORLD")*.

Để xử lý chuỗi trong KeilC, chúng ta chèn thêm hai thư viện sau ở đầu file *Led_matrix.c:*

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

Dữ liệu của buffer lớn gồm có 100 cột, biến *total_length* trong ví dụ sau dùng để lưu độ dài hiện tại của mảng buffer này:

```
unsigned char data_buff[100];
int total_length;
```

Mảng ***alphabet_upcase_Led_matrix*** là dữ liệu font của các chữ cái in hoa. Trong chương trình này chỉ hiển thị được các chữ cái in hoa, không bao gồm các chữ cái thường và các kí tự số.

4.3. Giao tiếp dữ liệu

4.3.1. Giao tiếp LCD

LCD là dạng màn hình tinh thể lỏng dùng để hiển thị các dữ liệu từ vi điều khiển lên giao diện người dùng. Hình 4.6 là mô tả sơ đồ giao tiếp họ 8051 với LCD2408.

Để có thể hiển thị một cách thuận tiện, linh hoạt các thông số của hệ thống đồng thời đảm bảo được tính mỹ thuật, ta chọn màn hình tinh thể lỏng LCD 2408. Đây là loại màn tinh thể lỏng gồm có 8 dòng, mỗi dòng có thể hiển thị 24 ký tự, rất tiện cho người sử dụng trong khi làm việc với 8051 ở chế độ monitor. LCD 2408 có 4 chân điều khiển và 8 chân dữ liệu, 4 chân điều khiển là *RS, R/W, E1, E2*. Chân *E1, E2*: được gọi là chân "*Enable*". Chân này cho phép gửi dữ liệu vào LCD hay không.

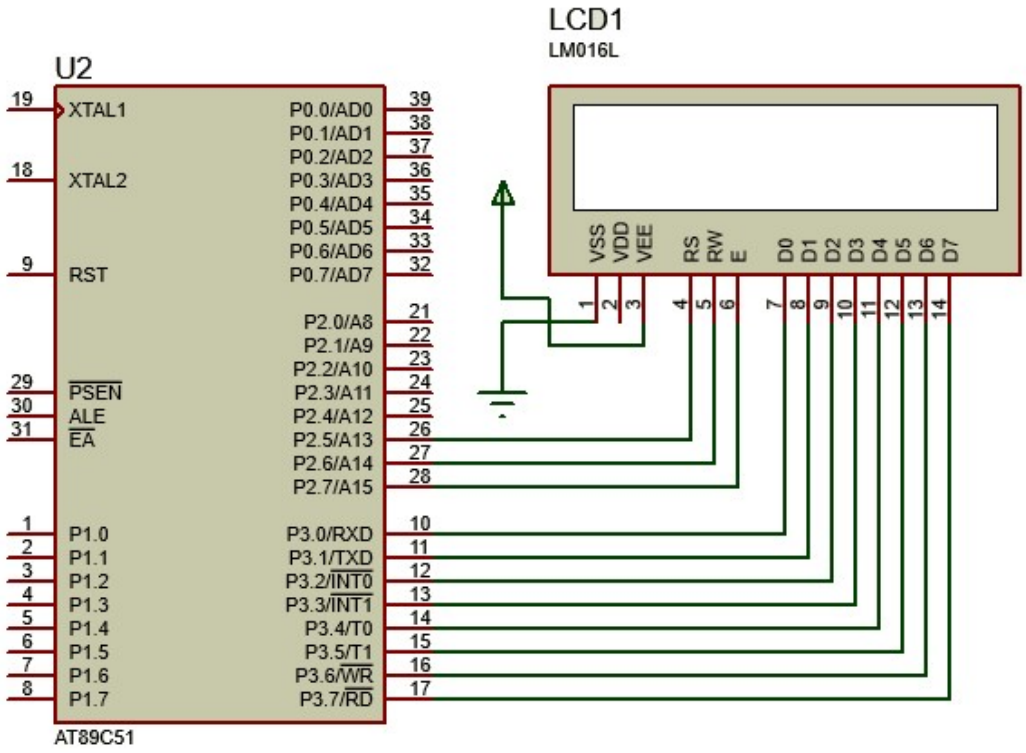
Chân 15 và 16 là *A* và *K* của LCD.

E1 cho phép làm việc với 4 dòng trên của màn hình. *E2* là chân cho phép làm việc với 4 dòng dưới. Để có thể gửi dữ liệu vào LCD, đầu tiên chân này phải được set lên "1". Sau khi thực hiện xong các lệnh, chân này phải set xuống "0" để báo cho

biết rằng LCD đã thực hiện lệnh và đang chờ lệnh tiếp theo.

Chân *RS*: là chân "*Register Select*". Khi chân này ở mức "0", LCD sẽ biết rằng các dữ liệu truyền đến nó dùng để điều khiển như các lệnh xóa màn hình, đặt vị trí con trỏ,.... Nếu *RS* ở mức "1" các dữ liệu truyền đến LCD được nó hiểu là các dữ liệu dạng ký tự cần hiển thị.

Chân *R/W*: là chân "*Read/Write*". Để có thể ghi dữ liệu lên LCD, chân này phải ở mức "0". Còn để đọc dữ liệu từ LCD thì chân này phải ở mức "1". Tuy nhiên trong LCD chỉ có một lệnh đọc dữ liệu từ LCD, đó chính là lệnh lấy trạng thái của LCD để báo cho biết nó đang bận hay không. Chính vì vậy chân này hầu như chỉ ở mức tín hiệu "0". *DB0 - DB7*: 8 chân dữ liệu của LCD.



Hình 4.6: Sơ đồ ghép nối LCD với vi điều khiển 1

Mỗi lần thực hiện một lệnh, LCD phải mất một khoảng thời gian để hoàn tất việc này. Chính vì vậy khi ra lệnh cho LCD, ta phải trễ một khoảng thời gian. Sau đó mới được thực hiện lệnh tiếp theo. Tuy nhiên phương pháp trễ không được ổn định và chính xác khi tần số thạch anh thay đổi, còn một phương pháp nữa là phương pháp kiểm tra LCD đã sẵn sàng nhận dữ liệu hay chưa bằng cách kiểm tra bit có trọng số cao nhất - D7 - của thanh ghi lệnh (lưu ý là thanh ghi lệnh, tức khi RS = 0), khi bit này xuống 0 báo hiệu LCD đã sẵn sàng nhận lệnh tiếp theo. Phương pháp này có ưu

điểm là làm việc ổn định, đồng bộ và không phải mất thời gian trễ dư ra không cần thiết. Để cho LCD có thể hoạt động, đầu tiên ta phải khởi tạo LCD, báo cho nó biết số hàng, số ký tự hiển thị trên một hàng. Các dữ liệu điều khiển lần lượt được chuyển vào Data Bus của LCD là 38H, 0EH, và 06H. Sau các lệnh khởi tạo LCD này ta mới có thể hiển thị ký tự lên trên màn hình LCD. Việc xóa màn hình, đưa con trỏ về góc trên bên trái có thể thực hiện được bằng cách chuyển dữ liệu điều khiển 01H vào Data Bus của LCD.

Bảng 4.2: Mã lệnh điều khiển của LCD 2408

Mã (Hexa)	Lệnh đến thanh ghi của LCD
1	Xóa màn hình hiển thị
2	Trở về đầu dòng
4	Dịch con trỏ sang trái
5	Dịch con trỏ sang phải
6	Dịch hiển thị sang phải
7	Dịch hiển thị sang trái
8	Tắt con trỏ, tắt hiển thị
A	Tắt hiển thị, bật con trỏ
C	Bật hiển thị, tắt con trỏ
E	Bật hiển thị, nhấp nháy con trỏ
F	Tắt hiển thị, nhấp nháy con trỏ
10	Dịch vị trí con trỏ sang trái
14	Dịch vị trí con trỏ sang phải
18	Dịch toàn bộ hiển thị sang trái
1C	Dịch toàn bộ hiển thị sang phải
80	Đưa con trỏ về đầu dòng thứ nhất (*)
C0	Đưa con trỏ về đầu dòng thứ hai (*)
38	Hai dòng và ma trận 5x7

(*) Địa chỉ của các dòng cụ thể của LCD 2408 như sau :

- Dòng 1 0080H Dòng 2 0098H
- Dòng 3 00B8H Dòng 4 00D8H

LCD 2408 được phân làm 2 nửa màn hình và chân E1, E2 được dùng để chọn làm việc với các màn hình theo thứ tự 1, 2. Mức tích cực của các chân này là mức "1". Cụ thể, ta muốn in kí tự 'A' lên dòng 2 của nửa màn hình 2 (dưới) ở LCD, ta thực hiện:

- Đặt E1 = 0, E2 = 1
- Chuyển sang thanh ghi lệnh: đặt RS = 0
- Gửi mã lệnh ra LCD: mov LCD_DATA_PORT, #DONG2
- Chuyển sang thanh ghi dữ liệu: đặt RS = 1
- In kí tự 'A': mov LCD_DATA_PORT, #'A'

Gửi mã lệnh hoặc dữ liệu đến LCD có kiểm tra cờ bận.

Đoạn chương trình trên đây đã chỉ ra cách gửi các lệnh đến LCD mà không có kiểm tra cờ bận (Busy Flag). Lưu ý rằng chúng ta phải đặt một độ trễ lớn trong quá Trình xuất dữ liệu hoặc lệnh ra LCD. Tuy nhiên, một cách tốt hơn là hiển thị cờ bận trước khi xuất một lệnh hoặc dữ liệu tới LCD. Dưới đây là một chương trình như vậy.

; Kiểm tra cờ bận trước khi gửi dữ liệu, lệnh ra LCD

; Đặt P1 là cổng dữ liệu

; Đặt P2.0 nối tới chân RS

; Đặt P2.1 nối tới chân R/W

; Đặt P2.2 nối tới chân E

Lưu ý rằng trong chương trình kiểm tra cờ bận D7 của thanh ghi lệnh. Để đọc thanh ghi lệnh ta phải đặt RS = 0, R/W = 1 và xung “cao - xuống - thấp” cho bit E để chọn thanh ghi lệnh. Sau khi đọc thanh ghi lệnh, nếu bit D7 (cờ bận) ở mức cao thì LCD bận và không có thông tin (lệnh) nào được xuất đến nó chỉ khi nào D7 = 0 mới có thể gửi dữ liệu hoặc lệnh đến LCD. Lưu ý phương pháp này không sử dụng độ trễ thời gian vì ta đang kiểm tra cờ bận trước khi xuất lệnh hoặc dữ liệu lên LCD.

Lưu ý rằng trong chương trình cờ bận D7 của thanh ghi lệnh, để đọc thanh ghi lệnh ta phải đặt RS = 0, R/W = 1 và xung “cao - xuống - thấp” cho bit E để chọn thanh ghi lệnh. Sau khi đọc thanh ghi lệnh, nếu bit D7 (cờ bận) ở mức cao thì LCD bận và không có thông tin (lệnh) nào được xuất đến nó chỉ khi nào D7 = 0 mới có thể gửi dữ liệu hoặc lệnh đến LCD. Trong LCD ta có thể đặt dữ liệu vào bất cứ chỗ nào dưới đây là các vị trí địa chỉ và cách chúng được truy cập.

Bảng 4.3: Bảng dữ liệu của LCD

RS	E/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A	A	A	A	A	A	A

Khi AAAAAAA = 0000000 đến 0100111 cho dòng lệnh 1 và AAAAAAA = 1100111 cho dòng lệnh 2.

Bảng 4.4: Đánh địa chỉ cho LCD

	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Dòng 1 (min)	1	0	0	0	0	0	0	0
Dòng 1 (max)	1	0	1	0	0	1	1	1
Dòng 2 (min)	1	1	0	0	0	0	0	0
Dòng 2 (max)	1	1	1	0	0	1	1	1

Dải địa chỉ cao có thể là 0100111 cho LCD, 40 ký tự trong khi đối với CLD 20 ký tự chỉ đến 010011 (19 thập phân = 10011 nhị phân). Để ý rằng dải trên 0100111 (nhị phân) = 39 (thập phân) ứng với vị trí 0 đến 39 cho LCD kích thước 40 × 2.

Từ những điều nói ở trên đây ta có thể nhận được các địa chỉ của vị trí con trỏ có các kích thước LCD khác nhau (Bảng 4.5).

Bảng 4.5: Danh sách liệt kê chi tiết các lệnh của LCD

16 × 2 LCD	80	81	82	83	84	85	86	Through	8F
	C0	C0	C2	C3	C4	C5	C6	Through	CF
20 × 1 LCD	80	81	82	83	Through 93				
20 × 2 LCD	80	81	82	83	Through 93				
	C0	C0	C2	C3	Through D3				
20 × 4 LCD	80	81	82	83	Through 93				
	C0	C0	C2	C3	Through D3				
	94	95	96	97	Through A7				
	D4	D5	D6	D7	Through E7				
20 × 2 LCD	80	81	82	83	Through A7				
	C0	C0	C2	C3	Through E7				

4.3.2. Giao tiếp ADC

ADC 0804 là bộ chuyển đổi tương tự sang số thuộc họ ADC8xx của hãng National Semiconductor. Nó cũng được nhiều hãng khác sản xuất, nó làm việc với điện áp +5V và có độ phân giải là 8 bit. Ngoài độ phân giải thì thời gian chuyển đổi cũng là một yếu tố quan trọng khác khi đánh giá một bộ ADC. Thời gian chuyển đổi được định nghĩa như là thời gian mà bộ ADC cần để chuyển một đầu vào tương tự

thành là đầu ra số nhị phân. Trong ADC0804 thời gian chuyển đổi thay đổi phụ thuộc vào tần số xung nhịp được cấp tới chân CLK và CLK IN nhưng không thể nhanh hơn 110μs. Các chân của ADC0804 được mô tả như sau [2]:

- Chân $\overline{CS}$ - chọn chip: Là một đầu vào tích cực mức thấp được sử dụng để kích hoạt chip ADC0804. Để truy cập ADC0804 thì chân này phải ở mức thấp.

- Chân $\overline{RD}$ (đọc): Đây là một tín hiệu đầu vào được tích cực mức thấp. Các bộ ADC chuyển đổi đầu vào tương tự thành số nhị phân tương đương với nó và giữ nó trong một thanh ghi trong. $\overline{RD}$ được sử dụng để nhận dữ liệu được chuyển đổi ở đầu ra của ADC0804. Khi CS = 0 nếu một xung "cao - xuống - thấp" được áp đến chân $\overline{RD}$ thì đầu ra số 8 bit được xuất hiện ở các chân dữ liệu D0 - D7. Chân $\overline{RD}$ cũng được coi như cho phép đầu ra.

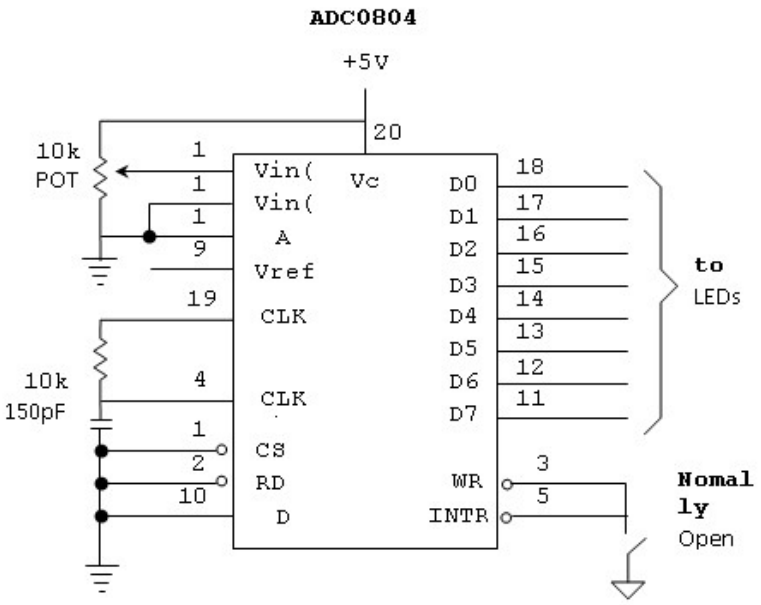
- Chân ghi $\overline{WR}$ (thực ra tên chính xác là "Bắt đầu chuyển đổi"). Đây là chân đầu vào tích cực mức thấp được dùng để báo cho ADC0804 bắt đầu quá trình chuyển đổi. Nếu CS = 0 khi $\overline{WR}$ tạo ra xung "cao - xuống - thấp" thì bộ ADC0804 bắt đầu chuyển đổi giá trị đầu vào tương tự V_{in} về số nhị phân 8 bit. Lượng thời gian cần thiết để chuyển đổi thay đổi phụ thuộc vào tần số đưa đến chân CLK IN và CLK R. Khi việc chuyển đổi dữ liệu được hoàn tất thì chân INTR được ép xuống thấp bởi ADC0804.

- Chân ngắt $\overline{INTR}$ (ngắt hay gọi chính xác hơn là "kết thúc chuyển đổi').

- Chân CLK IN là một chân đầu vào được nối tới một nguồn xung nhịp ngoài, khi xung nhịp ngoài được sử dụng để tạo ra thời gian. Tuy nhiên ADC0804 cũng có một bộ xung nhịp bên trong. Để sử dụng bộ xung nhịp bên trong của ADC804 thì các chân CLK IN và CLK R được nối tới một tụ điện và một điện trở như trên Hình 4.7. Trong trường hợp này tần số xung nhịp được xác định bằng biểu thức:

$$f = \frac{1}{1{,}1RC}$$

Giá trị tiêu biểu của các đại lượng trên là R = 10kΩ và C = 150pF và tần số nhận được là f = 606kHz và thời gian chuyển đổi sẽ mất là 110μs.



Hình 4.7: Sơ đồ nguyên lý ADC0804 ở chế độ chạy tự do.

Đây là chân đầu ra tích cực mức thấp. Bình thường nó ở trạng thái cao và khi việc chuyển đổi hoàn tất thì nó xuống thấp để báo cho CPU biết là dữ liệu được chuyển đổi sẵn sàng để lấy đi. Sau khi $\overline{\text{INTR}}$ xuống thấp, ta đặt CS = 0 và gửi một xung "cao xuống - thấp" tới chân $\overline{\text{RD}}$ lấy dữ liệu ra của ADC804.

- Chân V_{in} (+) và V_{in} (-): Đây là các đầu vào tương tự vi sai mà $V_{in} = V_{in}$ (+) - V_{in} (-). Thông thường V_{in} (-) được nối xuống đất và V_{in} (+) được dùng như đầu vào tương tự được chuyển đổi về dạng số.

- Chân V_{CC}: Đây là chân nguồn nuôi +5V, nó cũng được dùng như điện áp tham chiếu khi đầu vào $V_{ref/2}$ (chân 9) để hở.

- Chân $V_{ref/2}$: Chân 9 là một điện áp đầu vào được dùng cho điện áp tham chiếu. Nếu chân này hở (không được nối) thì điện áp đầu vào tương tự cho ADC0804 nằm trong dải 0 đến +5V (giống như chân V_{CC}). Tuy nhiên, có nhiều ứng dụng mà đầu vào tương tự áp đến V_{in} cần phải khác ngoài dải 0 đến 5V. Chân $V_{ref/2}$ được dùng để thực thi các điện áp đầu vào khác ngoài dải 0 – 5V, nếu dải đầu vào tương tự cần phải là 0 đến 4v thì $V_{ref/2}$ được nối với +2V. Biểu diễn dải điện áp V_{in} đối với các đầu vào $V_{ref/2}$ khác nhau:

Bảng 4.6: Điện áp $V_{ref/2}$ liên hệ với dải V_{in}

Vref/ 2(V)	Vin(V)	Step Size (mV)
Hở *	0 đến 5	5/256 = 19.53
2.0	0 đến 4	4/255 = 15.62

Vref/ 2(V)	Vin(V)	Step Size (mV)
1.5	0 đến 3	3/256 = 11.71
1.28	0 đến 2.56	2.56/256 = 10
1.0	0 đến 2	2/256 = 7.81
0.5	0 đến 1	1/256 = 3.90

Ghi chú: V_{CC} = 5V

Khi $V_{ref}/2$ hở thì đo được ở đó khoảng 2,5V; kích thước bước k (độ phân dải) là sự thay đổi nhỏ nhất mà ADC có thể phân biệt được.

- Các chân dữ liệu D0 - D7: Các chân dữ liệu D0 - D7 (D7 là bit cao nhất MSB và D0 là bit thấp nhất LSB) là các chân đầu ra dữ liệu số. Đây là những chân được đệm ba trạng thái và dữ liệu được chuyển đổi chỉ được truy cập khi chân CS = 0 và chân $\overline{RD}$ bị đưa xuống thấp. Để tính điện áp đầu ra ta có thể sử dụng công thức sau:

$$D_{out} = \frac{V_{in}}{k}$$

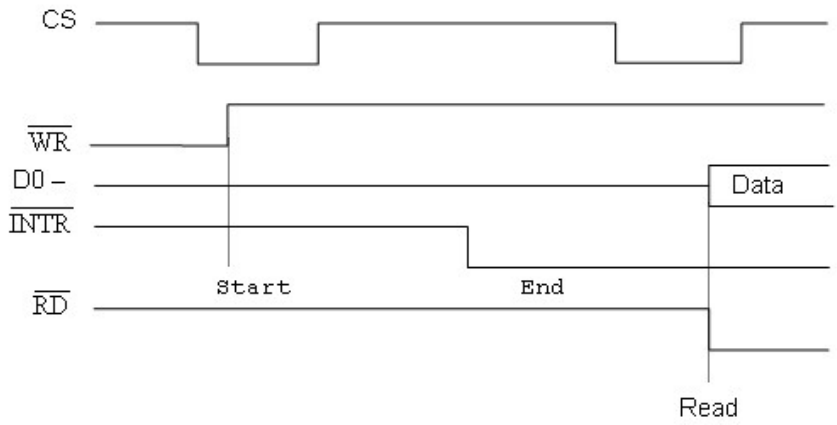
Với D_{out} là đầu ra dữ liệu số (dạng thập phân). V_{in} là điện áp đầu vào tương tự và độ phân dải là sự thay đổi nhỏ nhất được tính như là ($2 \times V_{ref}/2$) chia cho 256 đối với ADC 8 bit.

- Chân mass tương tự và chân mass số: Đây là những chân đầu vào nối đất chung cho cả tín hiệu số và tương tự. Đất tương tự được nối tới đất của chân V_{in}, còn đất số được nối tới đất của chân V_{cc}. Lý do mà ADC phải có hai chân nối đất là để cách ly tín hiệu tương tự V_{in} từ các điện áp ký sinh nhằm mục đích tạo ra việc chuyển mạch số được chính xác. Trong phần trình bày của chúng ta thì các chân này được nối chung với một đất. Tuy nhiên, trong thực tế thu đo dữ liệu các chân đất này được nối tách biệt. Từ những điều trên ta kết luận rằng các bước cần phải thực hiện khi chuyển đổi dữ liệu bởi ADC0804 là:

  • Bật CS = 0 và gửi một xung thấp lên cao tới chân $\overline{WR}$ để bắt đầu chuyển đổi.

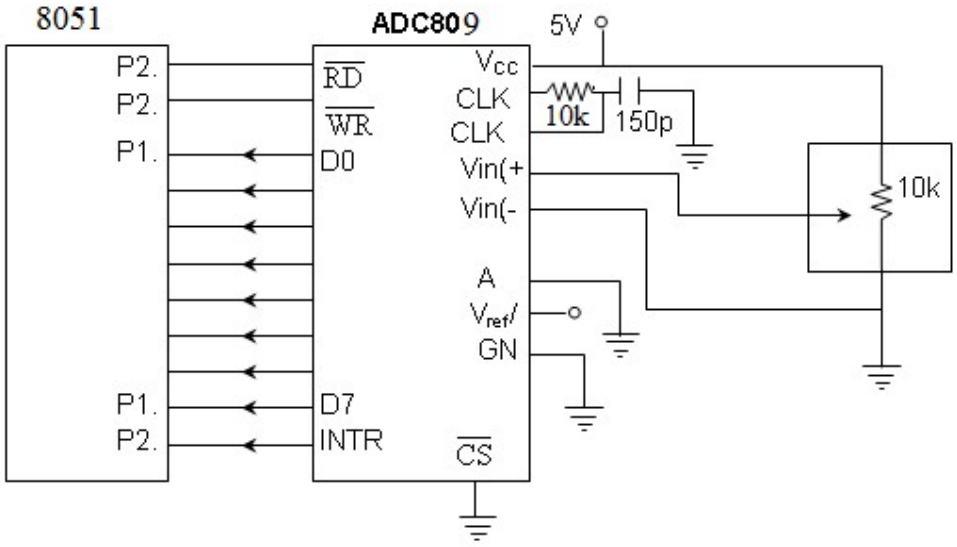
  • Duy trì hiển thị chân $\overline{INTR}$. Nếu $\overline{INTR}$ xuống thấp thì việc chuyển đổi được hoàn tất và ta có thể sang bước kế tiếp. Nếu $\overline{INTR}$ ở mức cao và tiếp tục thăm dò cho đến khi nó xuống thấp.

  • Sau khi chân $\overline{INTR}$ xuống thấp, ta bật CS = 0 và gửi một xung "cao - xuống - thấp" đến chân $\overline{RD}$ để lấy dữ liệu ra khỏi chip ADC0804. Phân chia thời gian cho quá trình này được trình bày trên Hình 4.8.



Hình 4.8: Giản đồ thời gian hoạt động các chân ADC 0809

Tương tự như ADC0804, IC chuyển đổi ADC tích hợp đến 8 kênh đầu vào là ADC0809. Vi mạch này chuyển đổi tín hiệu từ tương tự sang số 8 bit, được chế tạo theo công nghệ CMOS. Bộ chuyển đổi tương tự số này sử dụng phương pháp chuyển đổi xấp xỉ. Sai số của phép chuyển đổi là một bit. Sau đây sơ đồ kết nối họ 8051 với ADC0809.



Hình 4.9: Ghép nối ADC0809 với họ 8051

Nguồn nuôi 5V, dải tín hiệu lối vào tương tự 5V khi nguồn nuôi là +5V. Có thể mở rộng thang đo bằng các giải pháp kỹ thuật cho từng mạch cụ thể. Dễ dàng giao tiếp với vi điều khiển vì đầu ra có bộ đệm 3 trạng thái nên có thể ghép trực tiếp vào kênh dữ liệu của hệ vi điều khiển.

Một số chức năng của ADC0809:

- Tổng sai số chưa chỉnh là 1/2 LSB;
- Thời gian chuyển đổi 100µs;

- Tần số xung clock 10kHz – 1028 kHz;
- Đảm bảo sai số tuyến tính trong dải nhiệt độ từ -40^0C 85^0C;
- IN0 – IN7 là 8 chân đầu vào tương tự;
- A, B, C là các chân tín hiệu chọn kênh;
- Các chân D0 - D7 là các đầu ra số ;
- ALE là chân tín hiệu cho phép chốt số liệu đầu vào ;
- Start là chân cấp xung cho phép bắt đầu chuyển đổi ;
- Clk là đầu vào cung cấp xung clock cho ADC;
- Ref(+) là chân điện áp vào chuẩn +5V;
- Ref(-) là chân điện áp vào chuẩn 0;
- OE là chân cho phép xuất dữ liệu đã chuyển đổi ra DataBus;
- EOC là chân cho biết quá trình chuyển đổi đã kết thúc;
- Vcc là chân nguồn cung cấp.

Ví dụ 4.4: Chương trình giao tiếp ADC với 8051.

```
; Đặt P2.6 = WR (bắt đầu chuyển đổi cần một xung thấp lên cao)
; Đặt chân P2.7 = 0 khi kết thúc chuyển đổi
; Đặt P2.5 = RD (xung cao - xuống - thấp sẽ đọc dữ liệu từ ADC)
; P1.0 – P1.7 của ADC 804
MOV   P1, # 0FFH                ; Chọn P1 là cổng đầu vào
BACK:
      CLR   P2.6                ; Đặt WR = 0
      SETB        P2.6          ; Đặt WR = "1" để bắt đầu chuyển đổi
HERE:
      JB    P2.7, HERE          ; Chờ cho P2.7 to để kết thúc chuyển đổi
      CLR   P2.5                ; Kết thúc chuyển đổi, cho phép đọc RD
      MOV   A, P1               ; Đọc dữ liệu vào thanh ghi A
      ACALL       CONVERSION          ; Chuyển đổi số Hex ra mã
ASCII
      ACALL       DATA-DISPLAY  ; Hiển thị dữ liệu
      SETB  P2.5                ; Đưa RD = "1" để cho lần đọc sau.
SJMP  BACK
```

Xét trường hợp nối một LM35 tới một ADC0804 có độ phân dải 8 bit với tối đa 256 bước (2^8) và LM35 (hoặc ML34) tạo điện áp 10mV cho mỗi sự thay đổi nhiệt độ 1^0C nên ta có thể tạo điều kiện V_{in} của ADC0804 tạo ra một V_{out} = 2560mV (2,56V) cho đầu ra đầu thang đo. Do vậy, nhằm tạo ra V_{out} đầy thang 2,56V cho ADC0804 ta cần đặt điện áp $V_{ref}/2$ = 1,28V. Điều này làm cho V_{out} của ADC0804 đáp ứng trực tiếp với nhiệt độ được hiển thị trên LM35 (Bảng 4.7). Các giá trị của $V_{ref}/2$ được cho

ở **Error! Reference source not found.**.

Bảng 4.7: Giá trị điện áp tương ứng nhiệt độ

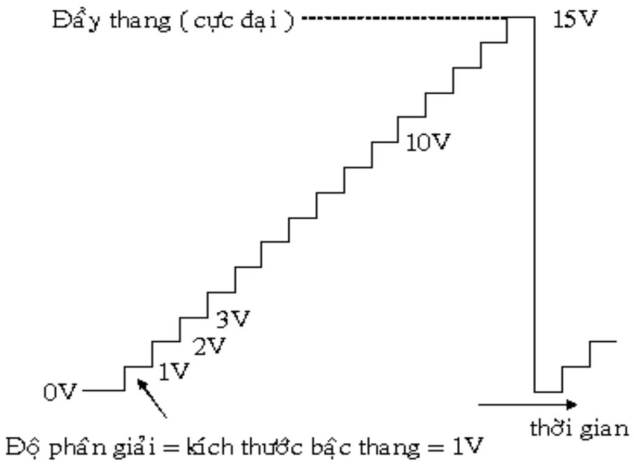
Nhiệt độ (^{0}C)	V_{in} (mV)	V_{out} (D7 – D0)
0	0	0000 0000
1	10	0000 0001
2	20	0000 0010
3	30	0000 0011
10	100	0000 1010
30	300	0001 1110

Hoạt động của ADC0809: Đầu tiên ta phát tín hiệu vào 3 chân A, B, C để chọn cổng vào tương tự. Để bắt đầu cho ADC0809 hoạt động, ta phát xung vào chân Start. Tiếp tục phát xung ALE để chốt dữ liệu tương tự đầu vào. Sau khi quá trình chuyển đổi tương tự – số đã diễn ra xong. ADC0809 sẽ tự phát ra một xung trên chân EOC để báo cho biết đã kết thúc quá trình chuyển đổi. Để dữ liệu được đưa ra các chân D0 - D7, ta phát một xung vào chân OE của ADC0809, bây giờ có thể đọc dữ liệu được.

4.3.3. Giao tiếp DAC

4.3.3.1. Độ phân giải

Độ phân giải (resolution) của bộ biến đổi DAC được định nghĩa là thay đổi nhỏ nhất có thể xảy ra ở đầu ra tương tự bởi kết quả của một thay đổi ở đầu vào số. Độ phân giải của DAC phụ thuộc vào số bit, do đó các nhà chế tạo thường ấn định độ phân giải của DAC ở dạng số bit. DAC 10 bit có độ phân giải tốt hơn DAC 8 bit. DAC có càng nhiều bit thì độ phân giải càng tốt hơn. Độ phân giải luôn bằng trọng số của LSB. Còn gọi là kích thước bậc thang (step size), vì đó là khoảng thay đổi của V_{out} khi giá trị của đầu vào số thay đổi từ bước này sang bước khác.



Hình 4.10: Cấu trúc thang điện áp của bộ DAC

Dạng sóng bậc thang có 16 mức với 16 thạng thái đầu vào nhưng chỉ có 15 bậc giữa mức 0 và mức cực đại. Với DAC có N bit thì tổng số mức khác nhau sẽ là 2^N, và tổng số bậc sẽ là $2^N - 1$. Do đó độ phân giải bằng với hệ số tỷ lệ trong mối quan hệ giữa đầu vào và đầu ra của DAC.

Đầu ra tương tự = K x đầu vào số

Với K là mức điện thế (hoặc cường độ dòng điện) ở mỗi bậc.

Như vậy ta có công thức tính độ phân giải như sau:

Độ phân giải:

$$\gamma = \frac{A_{fs}}{2^N - 1} \tag{4.1}$$

Với A_{fs} là đầu ra cực đại (đầy thang)

N là số bit.

Nếu tính theo phần trăm ta có công thức như sau:

$$\gamma\% = \frac{K}{A_{fs}} \tag{4.2}$$

Ta có :

$$\gamma\% = \frac{1V}{5V}.100\% = 6,67\% \tag{4.3}$$

Ví dụ 4.5: Một ADC 10 bit có kích thước bậc thang = 10mV. Hãy xác định điện thế đầu ra cực đại (đầy thang) và tỷ lệ % độ phân giải.

Bài giải:

DAC có 10 bit nên ta có số bậc là $2^{10} - 1 = 1023$ bậc

Với mỗi bậc là 10mV nên đầu ra cực đại sẽ là 10mVx1023 = 10.23V

$$\gamma\% = \frac{10mV}{10,23mV}.100\% = 0,1\% \quad (4.4)$$

Từ ví dụ trên cho thấy tỷ lệ phần trăm độ phân giải giảm đi khi số bit đầu vào tăng lên. Do đó ta còn tính được % độ phân giải theo công thức:

$$\gamma\% = \frac{A_{fs}}{2^{N-1}}.100\% \quad (4.5)$$

Với mã đầu vào nhị phân N bit ta có tổng số bậc là $2^N - 1$ bậc.

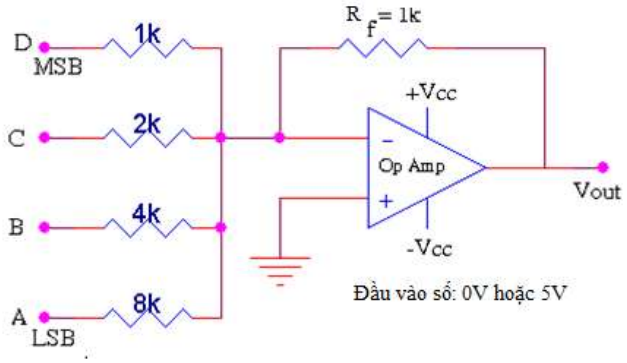
4.3.3.2. Độ chính xác

Có nhiều cách đánh giá độ chính xác, hai cách thông dụng nhất là sai số toàn thang (full scale error) và sai số tuyến tính (linearity error) thường được biểu diễn ở dạng phần trăm đầu ra cực đại (đầy thang) của bộ chuyển đổi.

Sai số toàn thang là khoảng lệch tối đa ở đầu ra DAC so với giá trị dự kiến (lý tưởng), được biểu diễn ở dạng phần trăm.

Sai số tuyến tính là khoảng lệch tối đa ở kích thước bậc thang so với kích thước bậc thang lý tưởng. Điều quan trọng của một DAC là độ chính xác và độ phân giải phải tương thích với nhau.

Sơ đồ mạch của một mạch DAC 4 bit dùng điện trở và bộ khuếch đại đảo, bốn đầu vào A, B, C, D có giá trị giả định lần lượt là 0V và 5V.



Hình 4.11: Sơ đồ cấu trúc DAC

Bộ khuếch đại thuật toán (Operational Amplifier – OA) được dùng làm bộ cộng đảo cho tổng trọng số của bốn mức điện thế vào. Ta thấy các điện trở đầu vào giảm

dần 1/2 lần điện trở trước nó. Nghĩa là đầu vào D (MSB) có RIN = 1k, vì vậy bộ khuếch đại cộng chuyển ngay mức điện thế tại D đi mà không làm suy giảm (vì R_f = 1k). Đầu vào C có R = 2k, suy giảm đi 1/2, tương tự đầu vào B suy giảm 1/4 và đầu vào A giảm 1/8. Do đó đầu ra bộ khuếch đại được tính bởi biểu thức:

$$V_{out} = -\frac{1}{2}(V_D + \frac{1}{2}V_C + \frac{1}{4}V_B + \frac{1}{8}V_A) \quad (4.6)$$

Dấu âm (-) biểu thị bộ khuếch đại cộng ở đây là khuếch đại cộng đảo. Dấu âm này chúng ta không cần quan tâm.

Độ phân giải của mạch DAC Hình 4.11 bằng với trọng số của LSB, nghĩa là bằng V_{out} nhân với 5V = 0.625V.

Ví dụ 4.6: Độ phân giải DAC

a. Xác định trọng số của mỗi bit đầu vào ở trên

b. Thay đổi Rf thành 500Ω. Xác định đầu ra cực đại đầy thang.

Bài giải:

a. MSB chuyển đi với mức khuếch đại = 1 nên trọng số của nó ở đầu ra là 5V.

Tương tự như vậy ta tính được các trọng số của các bit đầu vào như sau:

- MSB # 5V
- MSB thứ 2 # 2.5V (giảm đi 1/2)
- MSB thứ 3 # 1.25V (giảm đi 1/4)
- MSB thứ 4 (LSB) # 0.625V (giảm đi 1/8)

b. Nếu R_f = 500Ω giảm theo thừa số 2, nên mỗi trọng số đầu vào sẽ nhỏ hơn 2 lần so với giá trị tính ở trên. Do đó đầu ra cực đại (đầy thang) sẽ giảm theo cùng thừa số, còn lại: - 9.375/2 = - 4.6875V

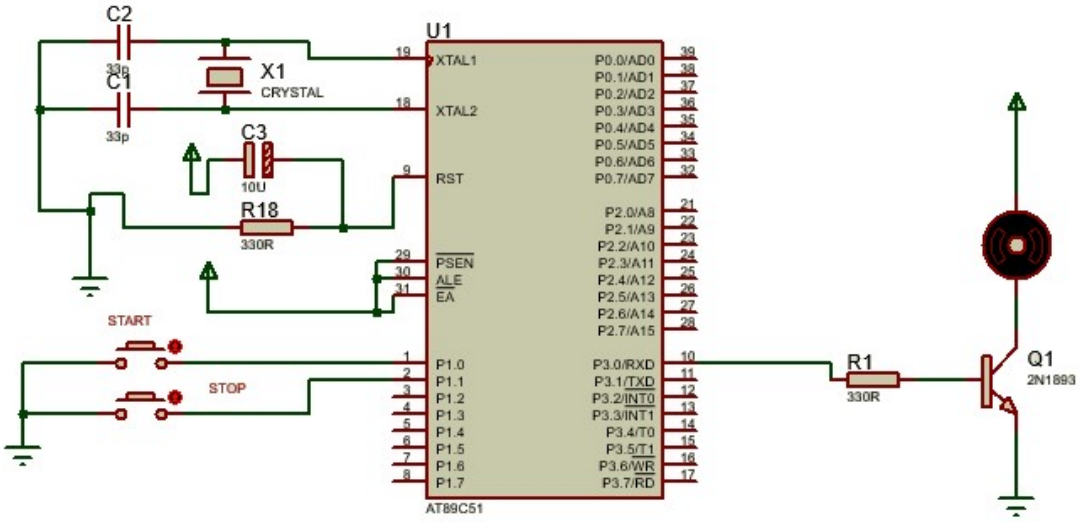
4.4. Kết nối với linh kiện điện tử công suất

4.4.1. Giao tiếp Transistor

Thực hiện việc chuyển các xung điện thành các bước quay mịn của motor. Do có sự tương ứng 1-1 giữa xung điện điều khiển và bước quay của motor nên động cơ bước có độ chính xác dịch chuyển cao. Vì vậy động cơ bước thường được sử dụng trong các thiết bị toạ độ chính xác như cánh tay robot, máy in, máy vẽ, thiết bị khoa học... Nguyên lý hoạt động và cấu tạo của động cơ bước được trình bày như sau:

Error! Reference source not found. là động cơ bước loại nam châm vĩnh cửu. Rôto làm bằng nam châm vĩnh cửu còn các cuộn dây được quấn trên stato. Bước góc

của Rôto là 90^0. Động cơ có 4 cực được đặt ở vị trí đối diện nhau từng cặp một, gồm X, X và Y, Y.



Hình 4.12: Sơ đồ ghép nối họ 8051 với transistor

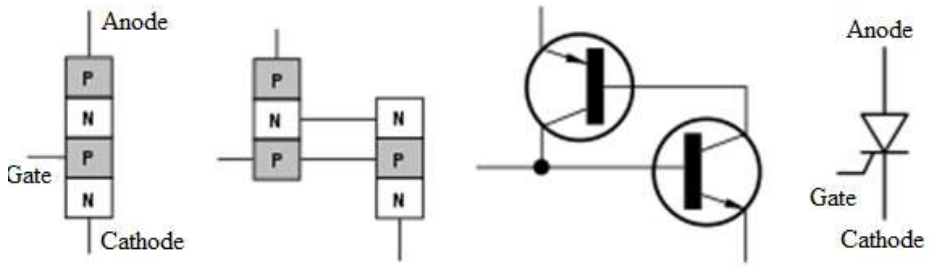
Tuỳ thuộc vào chiều dòng điện chạy qua các cuộn dây mà từ trường của stato sẽ có một hướng nhất định. Do đó Rôto sẽ được định vị chính xác.

Việc thay đổi thứ tự chiều dòng điện chạy qua các cuộn dây theo một trình tự nhất định sẽ điều khiển được động cơ chạy thuận hay chạy ngược như mong muốn. Và tốc độ của động cơ có thể được điều chỉnh bằng khoảng thời gian trễ giữa hai lần thay đổi thứ tự chiều dòng điện.

Trong mạch nguyên lý ở **Error! Reference source not found.** có một Header 6 đầu ra để điều khiển động cơ bước (Stepper motor). Loại động cơ bước được lắp đặt trên hình là động cơ bước có điện áp nguồn cấp là 5V và độ phân giải là 200 bước/vòng, tức là với mỗi nhịp điều khiển, motor bước sẽ quay một góc = 3600/200 = $1,8^0$. vậy 4 bước Rôto sẽ quay được 4x1.8 = $7,2^0$.

4.4.2. Giao tiếp Thyristor và Triac

Thay vì chỉ có 3 lớp bán dẫn (hai mối nối) thì người ta chế tạo linh kiện có 4 lớp bán dẫn (3 mối nối).

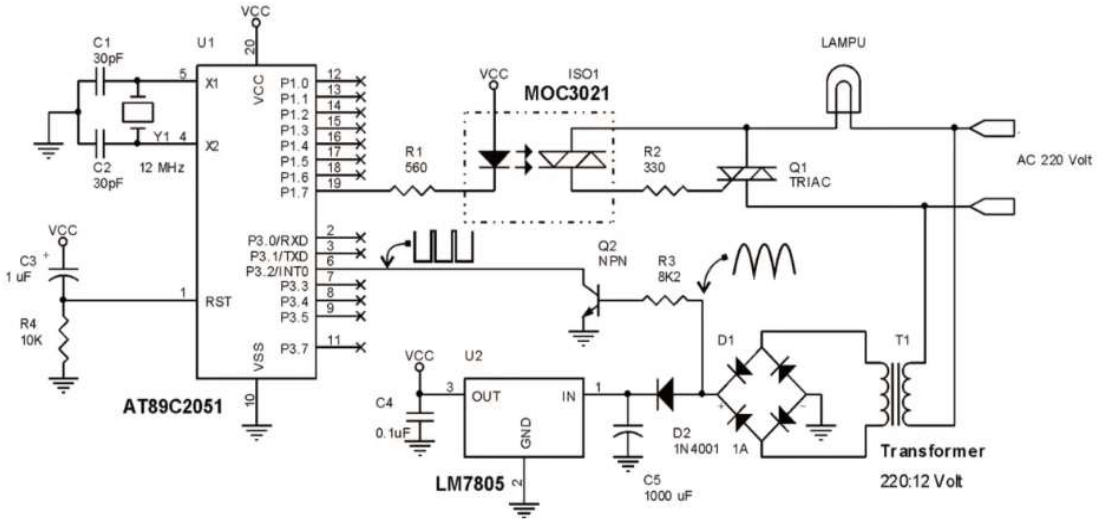


Hình 4.13: Cấu trúc của linh kiện thyristor

Trong Hình 4.13 cho thấy 4 lớp bán dẫn xem như là 2 transistor PNP và NPN kết hợp nhau thành một cấu trúc mới. Cấu trúc bán dẫn này có tên là SCR (thyristor) vì nó có khả năng khởi dẫn ở một thời gian, điều khiển được quá trình nắn dòng. SCR giống như hai diod nối tiếp, có 3 cực Anode-A, Cathode-K và cực Gate-G.

Phân tích về dòng cho thấy khi A - K có điện áp thuận, chưa có dòng qua SCR. Ở thời điểm t_{on} có xung trên G thì NPN dẫn, lúc này PNP kích dẫn và dòng chảy từ A qua K do đó SCR dẫn tạo ra dòng I(scr) qua SCR.

Sơ đồ thiết kế Kết nối triac với 8051 trong đó P1.7 là xung được phát ra cấp cho việc mở thyristor hoặc triac nhằm điều chỉnh điện áp công suất đầu ra.



Hình 4.14: Sơ đồ ghép nối họ 8051 với triac

4.4.3. Giao tiếp IGBT

PWM viết tắt của từ Pulse Width Modulation. PWM được sử dụng nhiều trong hệ thống điều khiển tự động ngày nay. Nó được ứng dụng trong điều khiển tốc độ động cơ, độ sáng tối của Led, màn hình LCD, pha màu cho bảng quang báo, sử dụng trong các thuật toán điều khiển vận tốc cho Robot như PI, PD, PID …

Hiểu đơn giản PWM hoạt động như một công tắc đóng mở rất nhiều lần trong một khoảng thời gian. Nếu tần số đóng mở càng nhanh thì điện áp cấp trung bình càng lớn.

Ngoài lĩnh vực điều khiển hay ổn định tải thì PWM nó còn tham gia và điều chế các mạch nguồn như là: boot, buck, nghịch lưu 1 pha và 3 pha… chúng ta còn gặp nhiều trong thực tế và các mạch điện điều khiển. Điều đặc biệt là PWM chuyên dùng để điều khiển các phần tử điện tử công suất có đường đặc tính là tuyến tính khi có sẵn một nguồn một chiều cố định. Như vậy PWM nó được ứng dụng rất nhiều trong các thiết bị điện, điện tử. Điều mà lĩnh vực điện, điện tử dễ dàng nhận ra là PWM chính nhân tố mà các đội Robocon sử dụng để điều khiển động cơ hay ổn định tốc độ động cơ. Trong bài viết này chúng ta sẽ tìm hiểu phương pháp dùng 8051 để tạo xung có độ rộng mong muốn trong điều khiển động cơ điện.

- Tạo một tần số chuẩn từ 0,1ms đến 10ms dùng các bộ timer của vi điều khiển.
- Thay đổi độ rộng xung bằng cách thay đổi tỷ số xung mức "1" và mức "0" của tần số chuẩn đó.

Với độ rộng xung thay đổi chúng ta có thể thay đổi độ sáng tối của đèn Led, tốc độ động cơ. Dưới đây là một ứng dụng của PWM trong việc điều khiển động cơ điện một chiều

4.5. Kết nối truyền thông

4.5.1. Kết nối RS232

Xây dựng ứng dụng kết nối giữa board 89Cxx và máy tính thông qua cổng COM.

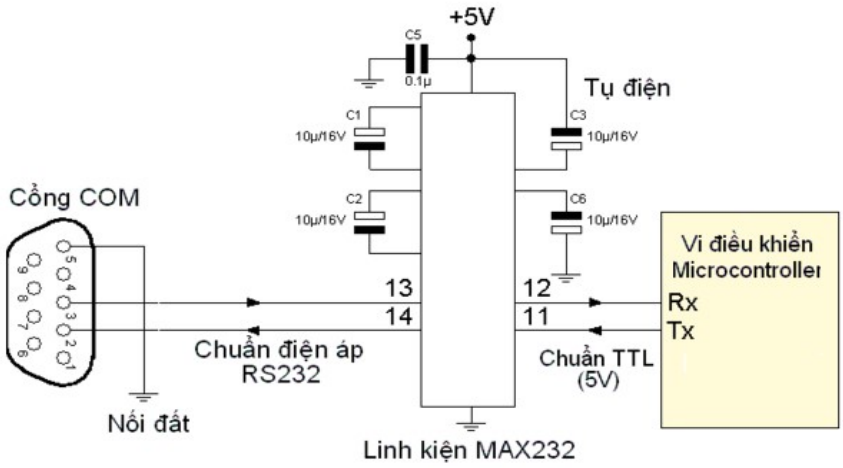
Nguyên lý kết nối UART: UART RS232 là chuẩn kết nối khá phổ biến và được hỗ trợ ở hầu hết các dòng vi điều khiển vì khoảng cách xa và chi phí thấp. Dòng 8051 hỗ trợ một kênh giao tiếp UART. Dữ liệu được truyền đi trên chân TX gồm bit 1 start (mức 0), data và 1 bit stop (mức 1).

UART là phương thức truyền nhận bất đồng bộ, nghĩa là bên nhận và bên phát không cần phải có chung tốc độ xung clock (xung clock của vi điều khiển khác xung clock của máy tính). Khi đó bên truyền muốn truyền dữ liệu sẽ gửi *start bit* (bit 0) để báo cho bên thu biết để bắt đầu nhận dữ liệu và khi truyền xong dữ liệu thì *stop bit* (bit 1) sẽ được gửi để báo cho bên thu biết kết thúc quá trình truyền.

Khi có start bit thì cả hai bên sẽ dùng chung một xung clock (có thể sai khác một ít) với độ rộng một tín hiệu (0 hoặc 1) được quy định bởi "baud rate", "baud rate"

= 9600bps nghĩa là độ rộng của tín hiệu 0 (hoặc 1) là 1/9600 = 104ms và khi phát thì bên phát sẽ dùng "baud rate" chính xác (9600bps) còn bên thu có thể dùng "baud rate" sai lệch một ít (9800bps chẳng hạn). Truyền bất đồng bộ sẽ truyền theo từng frame và mỗi frame có cấu trúc như sau: *Stop bit--B7--B6-- B5-- B4-- B3-- B2-- B1-- B0-- Start bit.*

Ngoài ra trong frame truyền có thể có thêm bit odd parity (bit lẻ) hoặc even parity (bit chẵn) để kiểm tra lỗi trong quá trình truyền. Bit parity này có đặc điểm nếu sử dụng odd parity thì số các bit 1 + odd parity bit sẽ ra một số lẻ còn nếu sử dụng even parity thì số các bit 1 + even parity bit sẽ ra một số chẵn.



Hình 4.15: Giao tiếp UART

Dữ liệu truyền tới vi điều khiển được lưu trong thanh ghi SBUF và khi vi điều khiển nhận đủ dữ liệu, cờ ngắt RI sẽ bật lên.

Cổng nối tiếp được sử dụng để truyền dữ liệu hai chiều giữa máy tính và ngoại vi, có các ưu điểm sau:

- Khoảng cách truyền xa hơn truyền song song.
- Số dây kết nối ít.
- Có thể truyền không dây dùng hồng ngoại.
- Có thể ghép nối với vi điều khiển hay PLC (Programmable Logic Device).
- Cho phép nối mạng.
- Có thể tháo lắp thiết bị trong lúc máy tính đang làm việc.
- Có thể cung cấp nguồn cho các mạch điện đơn giản

Tín hiệu truyền theo chuẩn RS-232 của EIA (Electronics Industry Associations). Chuẩn RS-232 quy định mức logic "1" ứng với điện áp từ -3V đến -25V (mark), mức logic 0 ứng với điện áp từ 3V đến 25V (space) và có khả năng cung cấp dòng từ 10

mA đến 20 mA. Ngoài ra, tất cả các ngõ ra đều có đặc tính chống chập mạch. Chuẩn RS-232 cho phép truyền tín hiệu với tốc độ đến 20.000 bps nhưng nếu cáp truyền đủ ngắn có thể lên đến 115.200 bps. Các phương thức nối giữa DTE và DCE:

- Đơn công (simplex connection): dữ liệu chỉ được truyền theo một hướng.
- Bán song công (half-duplex): dữ liệu truyền theo hai hướng, nhưng mỗi thời điểm chỉ được truyền theo một hướng.
- Song công (full-duplex): số liệu được truyền đồng thời theo 2 hướng.
- Định dạng của khung truyền dữ liệu theo chuẩn RS-232 như sau:

	Start	D0	D1	D2	D3	D4	D5	D6	D7	P	Stop
	0										1

Khi không truyền dữ liệu, đường truyền sẽ ở trạng thái mark (điện áp -10V). Khi bắt đầu truyền, DTE sẽ đưa ra xung Start (space: 10V) và sau đó lần lượt truyền từ D0 đến D7 và Parity, cuối cùng là xung Stop (mark: -10V) để khôi phục trạng thái đường truyền. Dạng tín hiệu truyền mô tả như sau (truyền ký tự A).

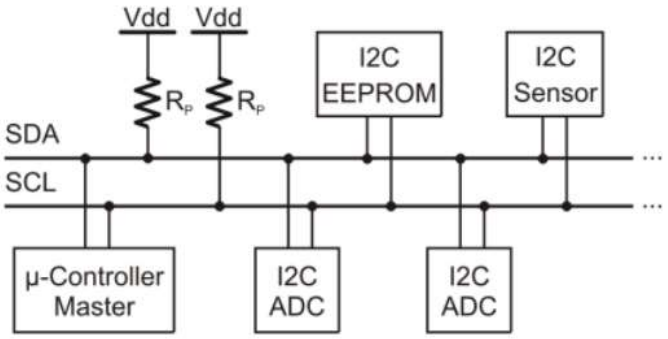
4.5.2. Giao tiếp I2C

I2C là một chuẩn giao tiếp theo mô hình master/slave được phát triển vào năm 1982 bởi hãng Philips cho việc giao tiếp ngoại vi giữa các vi điều khiển, IC trong khoảng cách ngắn. Chuẩn giao tiếp I2C được sử dụng rất phổ biến trong các thiết bị điện tử bởi đặc tính dễ thực hiện với giao tiếp giữa một hoặc nhiều thiết bị làm master với một hoặc hoặc nhiều thiết bị làm slave. Các hãng sản xuất IC ngày nay đều hỗ trợ giao tiếp I2C trên các IC có ứng dụng cần giao tiếp với các IC hay các vi điều khiển khác.

Giao tiếp I2C chỉ sử dụng 2 dây cho việc giao tiếp giữa 128 thiết bị với việc định địa chỉ 7 bit và 1024 thiết bị với việc định địa chỉ 10 bit. Khi đó, mỗi thiết bị trong mô hình master/slave sẽ có một địa chỉ cố định duy nhất và master sẽ lựa chọn thiết bị nào cần giao tiếp.

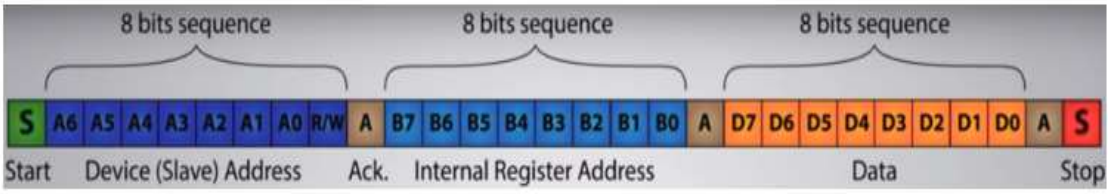
I2C sử dụng 2 dây giao tiếp là SCL và SDA. Dây SCL (viết tắt của Serial Clock Data) là dây truyền xung clock phát từ thiết bị master đồng bộ với việc truyền dữ liệu. Dây SDA (Serial Data) là dây truyền dữ liệu.

Mạch vật lý I2C là mạch cực thu hở, do đó để mạng I2C có thể hoạt động được, cần tối thiểu 2 cặp điện trở pull-up như trên hình, thông thường các giá trị điện trở 4k7 hoặc 1k2 được sử dụng, tùy thuộc vào tốc độ truyền và khoảng cách truyền.



Hình 4.16: Mô hình giao tiếp I2C

Truyền nhận bit trong I2C: Các dữ liệu được truyền trong I2C dạng các chuỗi 8 bit. Sau khi bit Start đầu tiên được truyền, 8 bit tiếp theo chứa thông tin về địa chỉ của thiết bị sẽ được truyền tiếp. Sau đó, một bit ACK sẽ được truyền để xác nhận việc truyền bit thành công. Sau khi truyền bit ACK, 8 bit tiếp theo chứa thông tin địa chỉ thanh ghi nội của thiết bị slave sẽ được truyền. Sau khi hoàn tất việc định địa chỉ với 8 bit này, một bit ACK nữa sẽ được truyền để xác nhận. Sau đó, 8 bit Data sẽ được truyền tiếp theo. Cuối cùng, quá trình truyền kết thúc với một bit ACK và 1 bit Stop xác nhận việc truyền dữ liệu kết thúc.



Hình 4.17: Các bit truyền, nhận trong giao tiếp I2C.

Xây dựng ứng dụng giao tiếp với DS1307 để lấy dữ liệu ngày tháng năm, giờ phút giây.

Các đặc điểm của DS1307: Real time clock đếm giờ, phút, giây, tháng, ngày của tháng, ngày của tuần, năm kể cả năm nhuận (đến năm 2100).

56 byte Ram để lưu trữ dữ liệu, nhưng dữ liệu không bị mất khi tắt nguồn.

Sử dụng 2 dây tín hiệu để truyền dữ liệu theo giao thức I2C.

Có thể lập trình được để xuất tín hiệu xung vuông.

Tự động phát hiện ra nguồn cung cấp bị lỗi (ngắt nguồn) và chuyển qua mạch bảo vệ sử dụng nguồn pin dự trữ.

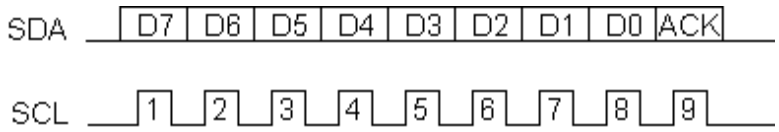
Dựa vào giản đồ trên ta hiện thực hàm để Start I2C như sau:

```
void start_I2C()
{
    SCL = 1;
    SDA = 1;
    nop();
    SDA = 0;
    SCL = 0;
    nop();
}
```

Tần số xung clock chuẩn của giao thức I2C là 100KHz. Khi truyền ở tốc độ cao có thể hoạt động ở clock 1MHz. Tuy nhiên chúng ta nên delay vài µs để đảm bảo dữ liệu được truyền đầy đủ và chính xác. Hàm để Stop I2C được hiện thực như sau:

```
void stop_I2C()
{
    SCL = 1;
    SDA = 0;
    nop(); nop();
    SDA = 1;
}
```

Truyền một byte dữ liệu: Khi truyền một byte dữ liệu, bit có trọng số cao nhất (bit 7) sẽ được truyền trước. Khi bit cuối cùng (bit 0) được truyền, sẽ có thêm bit ACK báo hiệu kết thúc một byte dữ liệu.



Dữu liêu được truyền từ “master” xuống “slave”.

Ở chế độ này, master sẽ gửi 8 bit dữ liệu, sau khi nhận xong 8 bit này, slave sẽ tự động gửi lại một bit ACK và master phải tạo ra thêm một clock để nhận bit ACK này.

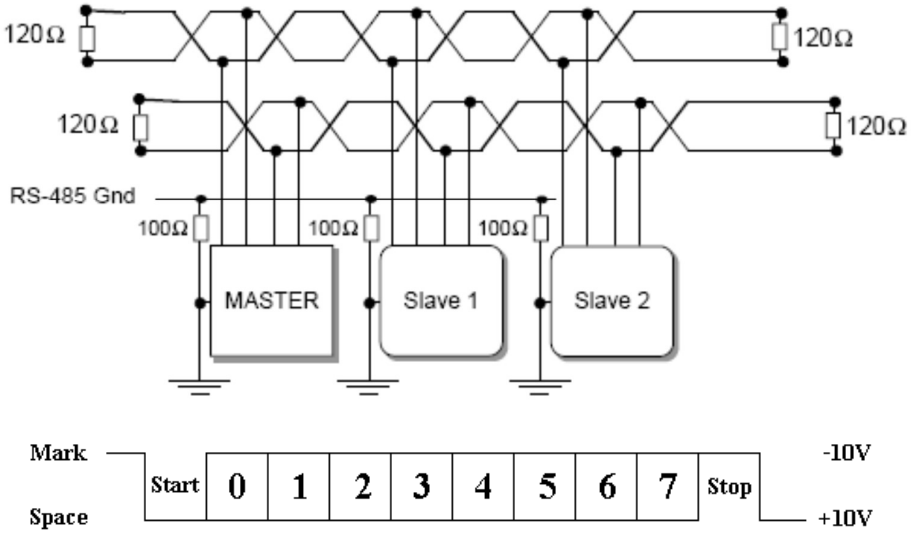
```
// Write I2C
void write_I2C(unsigned char data2send)
{
    int i;
    for (i=0;i<8;i++)
```

```
    {
        SDA = (data2send & 0x80) ? 1:0;
        SCL = 1;
        nop();
        SCL=0;
        data2send<< = 1;
        nop();
    }
// Clock to receive ACK from slave
    SCL = 1;
    nop();nop();
    SCL = 0;
}
```

Truyền từ slave lên master: Ở chế độ này, master sẽ nhận vào 8 bit dữ liệu, và sau khi nhận xong, master phải gửi một bit ACK xuống slave. Trong quá trình đọc một chuỗi byte từ slave, master gửi bit ACK. Đối với byte cuối cùng, master sẽ gửi bit NO ACK và sau đó gửi tín hiệu stop. Hàm đọc một byte sau đây có tham số là ACK_Bit, dùng để phân biệt ACK và NACK.

4.5.3. Truyền thông RS485

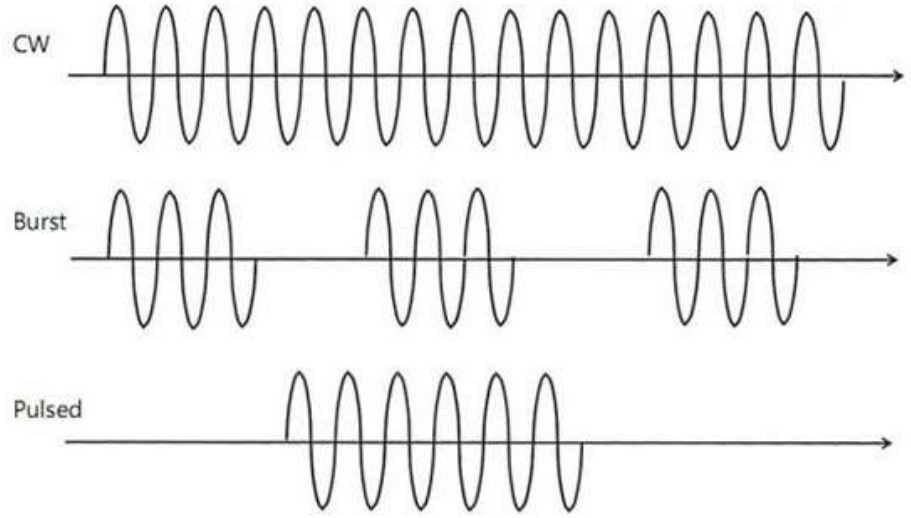
Chuẩn RS232 dùng đường truyền không cân bằng vì các tín hiệu lấy chuẩn là GND chung nên dễ bị ảnh hưởng của nhiễu làm tốc độ và khoảng cách truyền bị giới hạn. Khi muốn tăng khoảng cách truyền, môt phương pháp có thể sử dụng là dùng 2 dây truyền vi dây, vì lúc này hai dây có cùng đặc tính nên sẽ loại trừ được nhiễu chung. Hai chuẩn được sử dụng là RS422 và RS485 nhưng thông thường sử dụng RS485. Điện áp vi sai yêu cầu phải lớn hơn 200mV. Nếu V_{AB} > 200mV thì tương ứng với logic "1" và V_{AB} < -200mV tương ứng với logic 0. Chuẩn RS485 sử dụng hai điện trở kết thúc là 120Ω tại hai đầu xa nhất của đường truyền và sử dụng dây xoắn đôi.



Hình 4.18: Giải mã truyền thông RS

4.5.4. Truyền thông IRF (RF)

RF là từ viết tắt của chữ tiếng anh Radio Frequency, nói đơn giản đây là sóng vô tuyến có tần số từ 30kHz đến 300 GHz.



Hình 4.19: Dạng sóng RF

RF 433Mhz: Đây là tín hiệu sóng vô tuyến có tần số 433Mhz (Radio Frequency 433Mhz). Hz là đơn vị đo của tần số hay chu kỳ sóng, như vậy 433Mhz có nghĩa là 433.000.000 chu kỳ trong mỗi giây. Trong hình trên, kênh liên lạc (Communication Channel) chính là RF 433Mhz.

Tần số 433Mhz nằm trong băng sóng UHF (Untra Hight Frequency). Thường anten là loại đẳng hướng (tức bức xạ điện từ hướng ra mọi hướng trong không gian).

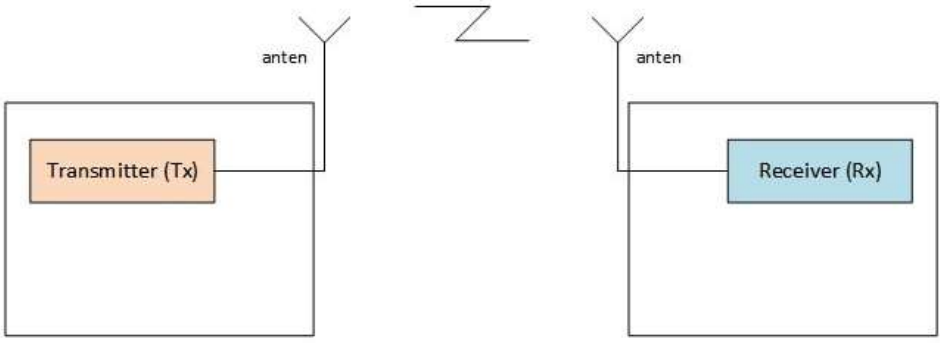
Đó là lý do vì sao máy thu sóng RF chỉ thu được một phần tín hiệu từ máy phát. "Một phần" ý nói một phần của công suất từ mát phát, còn nội dung thông tin nguyên vẹn.

Các đặc tính quan trọng: RF 433Mhz (hay 315Mhz) nằm trong miền tần số sóng điện từ UHF nên thường dùng để truyền tín hiệu trong môi trường không khí. Loại sóng này này cũng tuân theo các định luật phản xạ, khúc xạ, giao thoa của sóng điện từ và còn có khả năng đâm xuyên (xuyên tường, xuyên vật cản không phải là kim loại). Cự ly truyền phụ thuộc vào nhiều yếu tố: như tần số truyền (tần số càng thấp truyền càng xa); độ ẩm không khí hoặc tính đồng nhất của môi trường truyền; phân cực sóng; công suất phát (dBm), độ nhạy máy thu (dBm) v.v…

Các khối chức năng trong một hệ thống liên lạc sóng vô tuyến RF:

a) Simplex

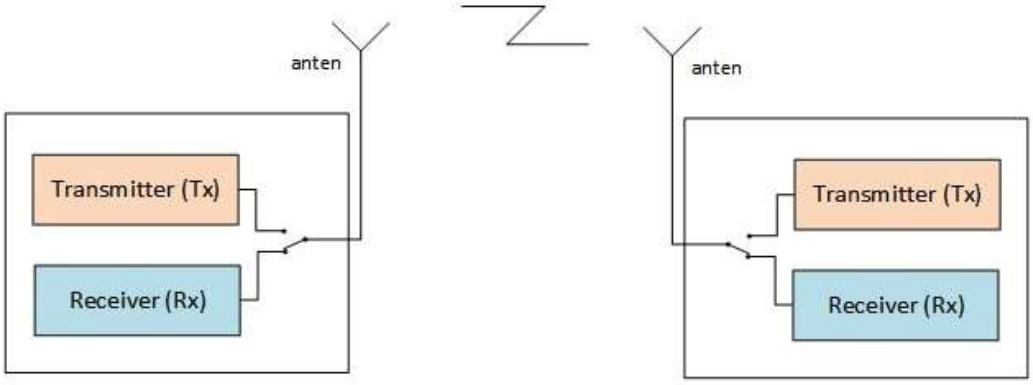
Kiểu liên lạc đơn công, đây chính là kiểu liên lạc của các remote xe hơi, remote cửa cuốn vv… Cái remote chứa mạch phát tín hiệu chỉ phát RF, còn xe hơi hay cái motor chứa mạch chỉ thu RF (Rx-Receiver).



Hình 4.20: Phương thức truyền sóng simplex – đơn công

b) Half-duplex

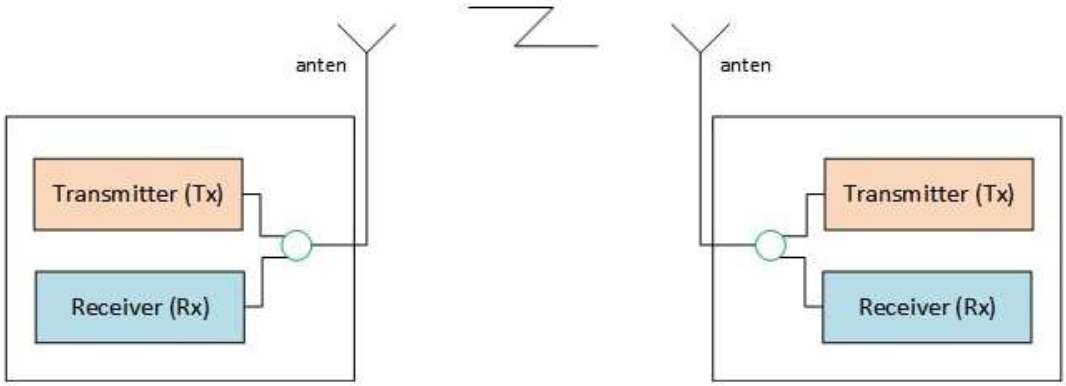
Với phương thức half-duplex thì mỗi bên đều có khối phát Tx và khối thu Rx và đều cần thêm cái chuyển mạch (nút bấm để nói trên bộ đàm):



Hình 4.21: Phương thức haff duplex – bán song công

c) Full-duplex

Với phương thức full-duplex (song công) thì mạch điện bên trong hơi phức tạp hơn. Cần có bộ điều hướng anten gọi là circulator. Bộ này có chức năng vừa đưa tín hiệu của bộ phát Tx lên anten để bức xạ ra không gian vừa lấy tín hiệu thu về và đưa vào bộ thu Rx mà không lẫn vào nhau (không nhiễu nội bộ):



Hình 4.22: Phương thức truyền sóng Full Duplex – Song công

4.6. Bài tập và câu hỏi cuối chương

Bài tập 4.1: Viết chương trình hiển thị số 24 trên LED 7 đoạn và LCD.

Bài tập 4.2: Viết chương trình quét phím 4x4 kết nối Port 0.

Bài tập 4.3: Viết chương trình PWM tạo sóng dạng Sin có tần số 50Hz.

Bài tập 4.4: Viết chương trình giao tiếp máy tính điều khiển 4 kênh thiết bị qua relay kết nối 8051.

Tài liệu tham khảo chương 4

Tiếng Việt

[1] Nguyễn Tăng Cường, Phan Quốc Thắng, ***Cấu trúc và lập trình họ 8051***, Nhà xuất

bản Khoa học Kỹ thuật, 2004.

Tiếng Anh

[2] Muhammad Ali Mazidi, Janice Gillispie Mazidi, Rolin D. McKinlay, ***The 8051 Microcontroller and Embedded Systems Using Assembly and C***, Department of Computer Science and Information Engineering National Cheng Kung University, Taiwan, 2003.

[3] Dogan Lbrahim, ***Microcontroller projects in C for the 8051***, Oxford Auckland Boston Johannesburg Melbourne New Delhi, 2011.

TÀI LIỆU THAM KHẢO

[1] H. Đ. H. Tống Văn On, *Vi điều khiển 8051*, NXB Lao động, 2000.

[2] Muhammad Ali Mazidi Janice Gillispie Mazidi Rolin D. McKinlay, "*The 8051 Microcontroller and Embedded Systems Using Assembly and C*," in Dept. of Computer Science and Information Engineering National, Dept. of Computer Science and Information Engineering National, 2005.

[3] Dogan Lbrahim, *Microcontroller Projects in C for the 8051*, vol. 0 7506 464, no. 9, 2000.

[4] Atmel, *ATmega328, in 8-bit AVR Microcontrollers*, 2016.

[5] P. M. Tuấn, *Arduino cho người mới bắt đầu*, Cộng đồng arduino Việt Nam; http://arduino.vn/, 2020.

[6] Trương Đình NHơn, Phạm Quang Huy, *Vi điều khiển và ứng dụng arduino dành cho người tự học*, NXB Thanh Niên, 2018, p. 479.

[7] Microchip, *User's Guide with MPLAB* ***Editor and MPLAB SIM Simulator***, Microchip Technology Inc., 2009.