UBND TỈNH LÂM ĐỒNG
TRƯỜNG CAO ĐẲNG ĐÀ LẠT

GIÁO TRÌNH

MÔN HỌC/MÔ ĐUN: Vi Điều Khiển
NGÀNH/NGHỀ: Điện Công Nghiệp
TRÌNH ĐỘ: Cao đẳng

Lâm Đồng, năm 2017

TUYÊN BỐ BẢN QUYỀN

LỜI GIỚI THIỆU

Để thực hiện biên soạn giáo trình đào tạo Nghề Điện công nghiệp ở trình độ CĐN và TCN, giáo trình Mô đun Vi điều khiển là một trong những giáo trình mô đun đào tạo chuyên ngành được biên soạn theo nội dung chương trình khung được Khoa Điện – Điện tử xây dựng và Trường Cao đẳng Nghề Đà Lạt ban hành dành cho hệ Cao Đẳng Nghề và Trung Cấp Nghề Điện công nghiệp.

Nội dung biên soạn ngắn gọn, dễ hiểu, tích hợp kiến thức và kỹ năng chặt chẽ với nhau.

Khi biên soạn, nhóm biên soạn đã cố gắng cập nhật những kiến thức mới có liên quan đến nội dung chương trình đào tạo và phù hợp với mục tiêu đào tạo, nội dung lý thuyết và thực hành được biên soạn gắn với nhu cầu thực tế trong sản xuất đồng thời có tính thực tiễn cao. Nội dung giáo trình được biên soạn với dung lượng thời gian đào tạo 60 giờ gồm có:

MĐ23-01: Sơ lược về lịch sử và hướng phát triển của vi điều khiển.

MĐ23-02: Cấu trúc họ vi điều khiển.

MĐ23-03: Ngôn ngữ lập trình C.

MĐ23-04: Xuất nhập Port.

MĐ23-05: Bộ định thời.

MĐ23-06: Ngắt.

MĐ23-07: Cổng nối tiếp.

Trong quá trình sử dụng giáo trình, tuỳ theo yêu cầu cũng như khoa học và công nghệ phát triển có thể điều chỉnh thời gian và bổ sung những kiên thức mới cho phù hợp.

Tuy nhiên, tùy theo điều kiện cơ sở vật chất và trang thiết bị, các trường có thể sử dụng cho phù hợp. Mặc dù đã cố gắng tổ chức biên soạn để đáp ứng được mục tiêu đào tạo nhưng không tránh được những khiếm khuyết. Rất mong nhận được đóng góp ý kiến của các thầy, cô giáo, bạn đọc để nhóm biên soạn sẽ hiệu chỉnh hoàn thiện hơn. Các ý kiến đóng góp xin gửi về Trường Cao đẳng Nghề Đà Lạt – Số 1 Hoàng Văn Thụ – Phường 4 – Tp.Đà lạt.

Đà Lạt, ngày 05 tháng 06 năm 2017

Tham gia biên soạn

1. Chủ biên: Ths.Nguyễn Mạnh Cường
2. Ths. Trương Duy Việt
3. Ks. Bùi Quang Sơn
4. Ks. Trương Đinh Minh Tân

MỤC LỤC

GIÁO TRÌNH MÔN HỌC/MÔ ĐUN

Tên mô đun: Vi điều khiển

Mã mô đun: MĐ23

I. Vị trí, tính chất của mô đun:

1. Vị trí: trước khi học mô đun này cần hoàn thành các môn học, mô đun cơ sở, đặc biệt là các môn học, mô đun: Tin học, Mạch điện, Điện tử cơ bản, Điện tử công suất

2. Tính chất: đây làm mô đun bắt buộc trong đào tạo chuyên môn nghề

II. Mục tiêu mô đun:

1. Về kiến thức: Cung cấp các khái niệm cơ bản, cách lập trình cơ bản cho họ vi điều khiển 8051.

2. Về kỹ năng:

- Vận hành được các thiết bị và dây chuyền sản xuất dùng vi điều khiển
- Xác định được các nguyên nhân gây ra hư hỏng sảy ra trong thực tế.
- Kiểm tra và viết được các chương trình điều khiển.

3. Về năng lực tự chủ và trách nhiệm:

- Có khả năng tự nghiên cứu, tự học, tham khảo tài liệu liên quan đến môn học để vận dụng vào hoạt động hoc tập.
- Vận dụng được các kiến thức tự nghiên cứu, học tập và kiến thức, kỹ năng đã được học để hoàn thiện các kỹ năng liên quan đến môn học một cách khoa học, đúng quy định.

Nội dung của môn học/mô đun:

STT	Tên các bài trong mô đun	Thời gian (giờ)			
		Tổng số	Lý thuyết	Thực hành, thí nghiệm, thảo luận, bài tập	Kiểm tra
1	**Bài mở đầu: Sơ lược về lịch sử và hướng phát triển của vi điều khiển**	2	2		
	1. Lịch sử phát triển 2. Vi điều khiển 3. Lĩnh vực và ứng dụng 4. Hướng phát triển				
2	**Bài 1:Cấu trúc họ vi điều khiển**	6	4	2	
	1. Tổng quan 2. Sơ đồ chân 3. Cấu trúc Port I/O				

	4. Tổ chức bộ nhớ 5. Các thanh ghi chức năng đặc biệt 6. Hoạt động Reset				
3	**Bài 2: Ngôn ngữ lập trình C**	8	4	4	
	1. Giới thiệu. 2. Các khái niệm cơ bản về ngôn ngữ lập trình C. 3. Các câu lệnh cơ bản. 4. Hàm trong C. 5. Hướng dẫn sử dụng Keil C.				
4	**Bài 3: Xuất nhập Port**	12	4	7	1
	1. Port của họ vi điều khiển 8051 2. Các ứng dụng Port của họ vi điều khiển 8051 3. Giao tiếp vi điều khiển với LCD				
5	**Bài 4: Bộ định thời**	12	6	5	1
	1. Mở đầu 2. Thanh ghi SFR của timer 3. Các chế độ làm việc của timer 4. Nguồn cung cấp xung cho Timer 5. Khởi động, dừng, điều khiển Timer 6. Khởi tạo và truy xuất thanh ghi Timer 7. Timer 2 của 8052				
6	**Bài 5: Ngắt**	8	4	3	
	1. Mở đầu 2. Tổ chức ngắt của 8051 3. Xử lý ngắt 4. Thiết kế chương trình dùng ngắt 5. Ngắt cổng nối tiếp 6. Các cổng ngắt ngoài 7. Đồ thị thời gian của ngắt				
7	**Bài 6: Cổng nối tiếp**	12	6	5	1
	1. Mở đầu 2. Thanh ghi điều khiển 3. Chế độ làm việc				

	4. Khởi tạo và truy suất thanh ghi PORT nối tiếp 5. Truyền thông đa xử lý 6. Tốc độ BAUD				
Cộng		**60**	**30**	**27**	**3**

BÀI MỞ ĐẦU: SƠ LƯỢC VỀ LỊCH SỬ VÀ HƯỚNG PHÁT TRIỂN CỦA VI ĐIỀU KHIỂN

Mã bài: MĐ23-01

Giới thiệu:

Trong những thập niên cuối thế kỷ XX, từ sự ra đời của công nghệ bán dẫn, kỹ thuật điện tử đã có sự phát triển vượt bậc. Các thiết bị điện tử sau đó đã được tích hợp với mật độ cao và rất cao trong các diện tích nhỏ,nhờ vậy các thiết bị nhỏ hơn và nhiều chức năng hơn. Các thiết bị điện tử ngày càng nhiều chức năng trong khi giá thành ngày càng rẻ hơn, chính vì vậy điện tử có mặt khắp nơi. Bước đột phát mới trong kỹ thuật điện tử là tạo ra một thiết bị điện tử mới là Vi điều khiển.

Một bộ vi điều khiển (microcontroller) được xem như là "một máy tính trong một chip" – nó là một mạch điện tích hợp trên một chip, có thể lập trình được, dùng để điều khiển hoạt động của một hệ thống.

Vi điều khiển được ứng dụng rất rộng rãi hiện nay. Đa số các lĩnh vực đều có thể ứng dụng vi điều khiển. Và đối với nền cơ khí tự động hoá bây giờ thì có lẽ nó đã gắn liền với vi xử lý. Vi điều khiển là một câu trúc siêu nhỏ, gồm các linh kiện điện tử có kích thước micro hoặc nano kết hợp với nhau, và được nối với các thiết bị bên ngoài qua các chân vi điều khiển. Vì vậy hiểu rõ cấu trúc của nó, ta sẽ hiểu được mình đang làm việc với cái gì? Và nó hoạt động như thế nào?

Mục tiêu:

- Trình bày được cấu trúc chung của vi điều khiển
- Phát biểu được các ứng dụng của vi điều khiển và hướng phát triển của vi điều khiển

Nội dung chính:

I. TỔNG QUAN

1. LỊCH SỬ PHÁT TRIỂN CỦA 8051

Vào năm 1981, hãng Intel giới thiệu một số bộ vi điều khiển có tên gọi là 8051. Bộ vi điều khiển này có 128 byte RAM, 4KB room trên chip, hai bộ định thời, một cổng nối tiếp và 4 cổng *(đều có độ rộng 8 bit)* vào ra, tất cả được đặt trên một chip. Lúc ấy, nó được coi là một " Hệ thống trên chip". 8051 là một bộ xử lý 8

bit, điều đó có nghĩa là CPU chỉ có thể làm việc với 8 bit dữ liệu tại một thời điểm. Dữ liệu lớn hơn 8 bit được chia ra thành các dữ liệu 8 bit để xử lý. 8051 có tất cả 4 cổng vào – ra *(I/O),* mỗi cổng rộng 8 bit. Mặc dù 8051 có thể có một ROM trên chip cực đại là 64KB, nhưng các nhà sản xuất lúc đó đã cho xuất xưởng chỉ với 4KB ROM trên chip.

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

Đặc tính	**Số lượng**
ROM trên chíp	4KB
RAM	128 byte
Bộ định thời	2
Các chân vào ra	32
Cổng nối tiếp	1
Nguồn ngắt	6

Các đặc tính của 8051 đầu tiên

2. BỘ VI ĐIỀU KHIỂN 8051

MCS-51 là họ vi điều khiển của hãng Intel. Vi mạch tổng quát của họ MCS-51 là chip 8051. Chip 8051 có một số đặc trưng cơ bản sau:

- Bộ nhớ chương trình bên trong: 4KB *(ROM).*
- Bộ nhớ dữ liệu bên trong: 128 byte *(RAM).*
- Bộ nhớ chương trình bên ngoài: 64 KB *(ROM)*
- Bộ nhớ dữ liệu bên ngoài: 64 KB I
- 4 port xuất nhập *(I/O port)* 8 bit.

- 2 bộ định thời 16 bit.
- Mạch giao tiếp nối tiếp.
- Bộ xử lý bit *(thao tác trên các bit riêng lẻ).*
- 210 vị trí nhớ được định địa chỉ, mỗi vị trí 1 bit.
- Nhân/Chia trong 4 µs.

3. CÁC THÀNH VIÊN KHÁC CỦA HỌ 8051

3.1 BỘ VI ĐIỀU KHIỂN 8031

8031 là một phiên bản khác của họ 8051. Chíp này thường được coi như là 8051 không có ROM trên chíp. Để có thể dùng được chip này cần phải bổ sung thêm ROM ngoài chứa chương trình cần thiết cho 8031. 8051 có chương trình được chứa ở ROM trên chip bị giới hạn đến 4KB, còn ROM ngoài của 8031 có thể lên đến 64 KB. Tuy nhiên, để có thể truy cập hết bộ nhớ ROM ngoài thì cần dùng thêm 2 cổng (*Port 0 và Port 2*), do vậy chỉ còn lại hai cổng (*Port 1 và Port 3*) để sử dụng. Nhằm khác phục vấn đề này, chúng ta có thể bổ sung thêm cổng vào/ra cho 8031.

3.2 BỘ VI ĐIỀU KHIỂN 8052

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

Đặc tính	**8051**	**8052**	**8031**
ROM trên chip	4KB	8KB	0K
RAM	128 byte	256 byte	128 byte
Bộ định thời	2	3	2
Chân vào ra	32	32	32
Cổng nối tiếp	1	1	1
Nguồn ngắt	6	8	6

So sánh các đặc tính của các thành viên họ 8051

Chúng ta có thể thấy 8051 là tập con của 8052. Do vậy tất cả mọi chương

trình viết cho 8051 đều có thể chạy trên 8052 nhưng điều ngược lại là không đúng.

3.3 BỘ VI ĐIỀU KHIỂN 8751

Chip 8751 chỉ có 4KB bộ nhớ UV-EROM trên chip. Để sử dụng chip này cần phải có thiết bị lập trình ROM và thiết bị xoá UV-EPROM. Do ROM trên chip 8751 là UV-EPROM nên cần mất 20 phút để xoá 8751 trước khi lập trình. Vì quá trình này mất quá nhiều thời gian nên nhiều nhà sản xuất đã cho ra phiên bản Flash ROM.

3.4 BỘ VI ĐIỀU KHIỂN AT8951 CỦA ATMEL CORPORATION

AT8951 là phiên bản 8051 có ROM trên chip là bộ nhớ Flash. Phiên bản này rất thích hợp cho các ứng dụng nhanh vì bộ nhớ Flash có thể được xoá trong vài giây. Dĩ nhiên, để dùng AT8951 cần phải có thiết bị lập trình PROM hỗ trợ bộ nhớ Flash nhưng không cần đến thiết bị xoá ROM vì bộ nhớ Flash sè được xoá bằng thiết bị lập trình PROM. Để tiện sử dụng, hiện nay hãng Atmel đang nghiên cứu một phiên bản của AT8951 có thể lập trình qua cổng COM của máy tính PC và như vậy sẽ không cần đến thiết bị lập trình PROM.

Ký hiệu	ROM	RAM	I/O	Timer	Ngắt thực tế	VCC	Số chân IC
AT89C51	4KB	128	32	2	5	5V	40
AT89LV51	4KB	128	32	2	5	3V	40
AT89C1051	1KB	64	15	1	3	3V	20
AT89C2051	2KB	128	15	2	5	3V	20
AT89C52	8KB	256	32	3	6	5V	40
AT89LV52	8KB	256	32	3	6	3V	40

3.5 BỘ VI ĐIỀU KHIỂN DS5000 CỦA DALLAS SEMICONDUCTOR

Bộ nhớ ROM trên chip của DS5000 là NV-RAM. DS5000 có khả năng nạp chương trình vào ROM trên chip trong khi nó vẫn ở trong hệ thống mà không cần phải lấy ra, Cách thực hiện là dùng cổng COM máy tính PC. Đây là một điểm

mạnh rất được ưa chuộng. Ngoài ra, NV-RAM còn có ưu việt là cho phép thay đổi nội dung RAM theo từng byte mà không cần phải xoá hết trước khi lập trình như bộ nhớ EPROM.

Ký hiệu	ROM	RAM	I/O	Timer	Ngắt thực tế	VCC	Số chân IC
DS5000-8	8KB	128	32	2	6	5V	40
DS5000-32	32KB	128	32	2	6	5V	40
DS5000T-8	8KB	128	32	2	6	5V	40
DS5000T-32	32KB	128	32	2	6	5V	40

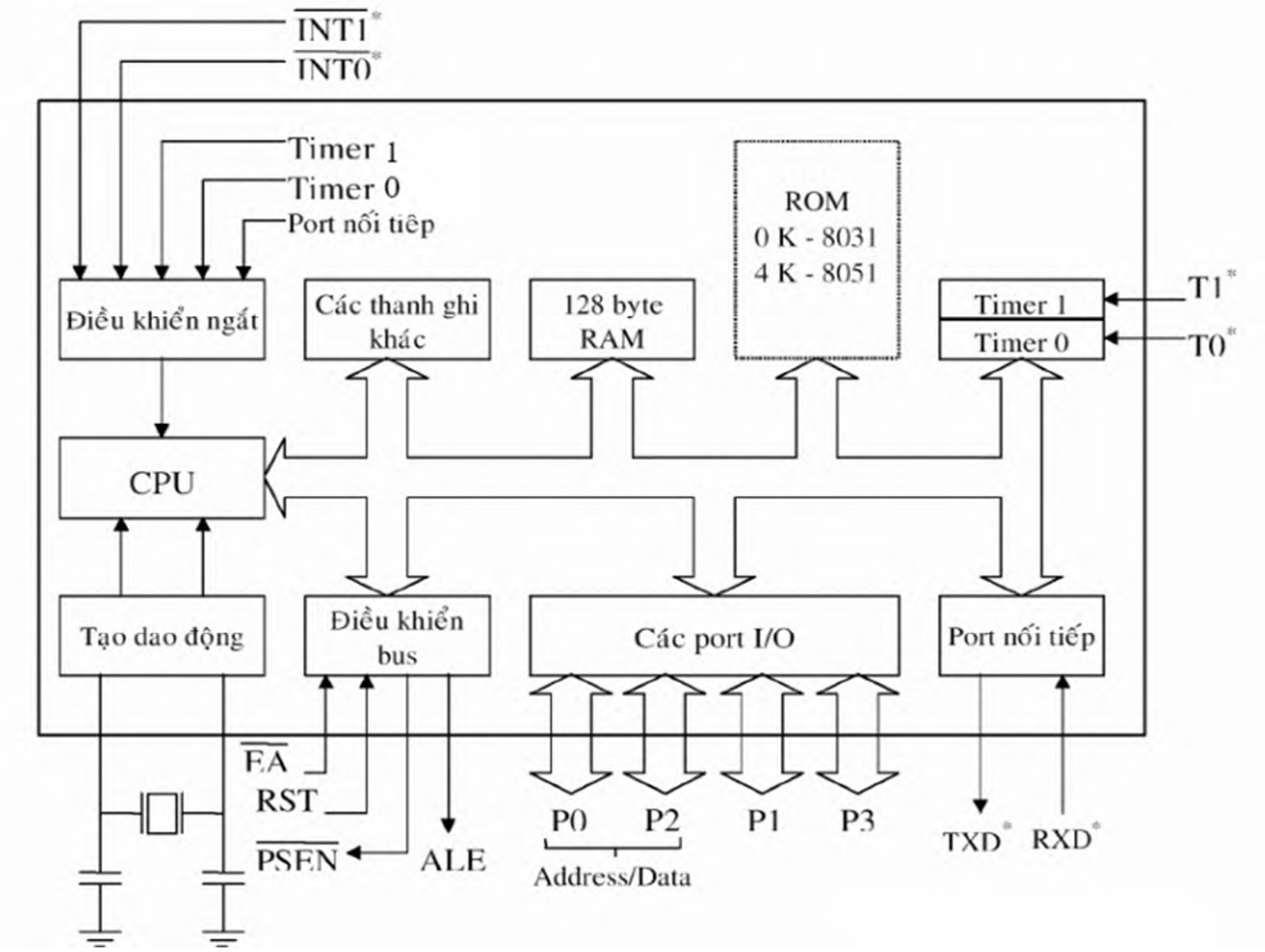
Điểm đặc biệt là các chip có chữ "T" theo sau ký hiệu "5000" có nghĩa là chip đó có thiết kế thêm một đồng hồ thời gian thực (*RTC: Real Time Clock*) bên trong. Lưu ý rằng, đồng hồ thời gian thực RTC hoàn toàn khác với bộ định thời Timer. RTC tạo và lưu giữ thời gian của ngày (*giờ/phút/giây*) và ngày tháng (*ngày/tháng/năm)* trên thực tế ngay cả khi không có nguồn cung cấp.

3.6 BỘ VI ĐIỀU KHIỂN P89V51XX CỦA PHILIPS CORPORATION

Đây là phiên bản cải tiến sử dụng CPU là bộ vi điều khiển 80C51 với nhiều tính năng vượt trội: dung lượng ROM/RAM trên chip rất lớn, 3 Timer 16 bit + 1 watch-dog Timer, 2 thanh ghi DPTR, 8 nguồn ngắt, PWM *(Pulse Width Modulator),* SPI *(Seria Periphearl Interface)* và đặc biệt là bộ nhớ chương trình trên chip có tính năng ISP *(In-System Programming)* và IAP *(In-Application Programming).*

II. SƠ ĐỒ CHÂN

1. SƠ ĐỒ KHỐI VÀ CHỨC NĂNG CÁC KHỐI CỦA CHIP 8051



Sơ đồ khối 8051

CPU *(Central Processing Unit)***:** Đơn vị xử lý trung tâm → tính toán và điều khiển quá trình hoạt động của hệ thống.

Tạo dao động: Mạch tạo dao động → tạo tín hiệu xung clock cung cấp cho các khối trong chip hoạt động.

Điều khiển ngắt: nhận tín hiệu ngắt từ bên ngoài (*INT0; INT1*), từ bộ định thời *(Timer 0; Timer 1)* và từ cổng nối tiếp *(Serial port),* lần lượt đưa các tín hiệu ngắt này đến CPU để xử lý.

Các thanh ghi khác: lưu trữ dữ liệu của các port xuất nhập, trạng thái làm việc của các khối trong chip trong suốt quá trình hoạt động của hệ thống.

RAM *(Random Accees Memory*): Bộ nhớ dữ liệu trong chip → lưu trữ các dữ liệu.

ROM *(Read Only Memory):* Bộ nhớ chương trình trong chip → lưu giữ chương trình hoạt động của chip.

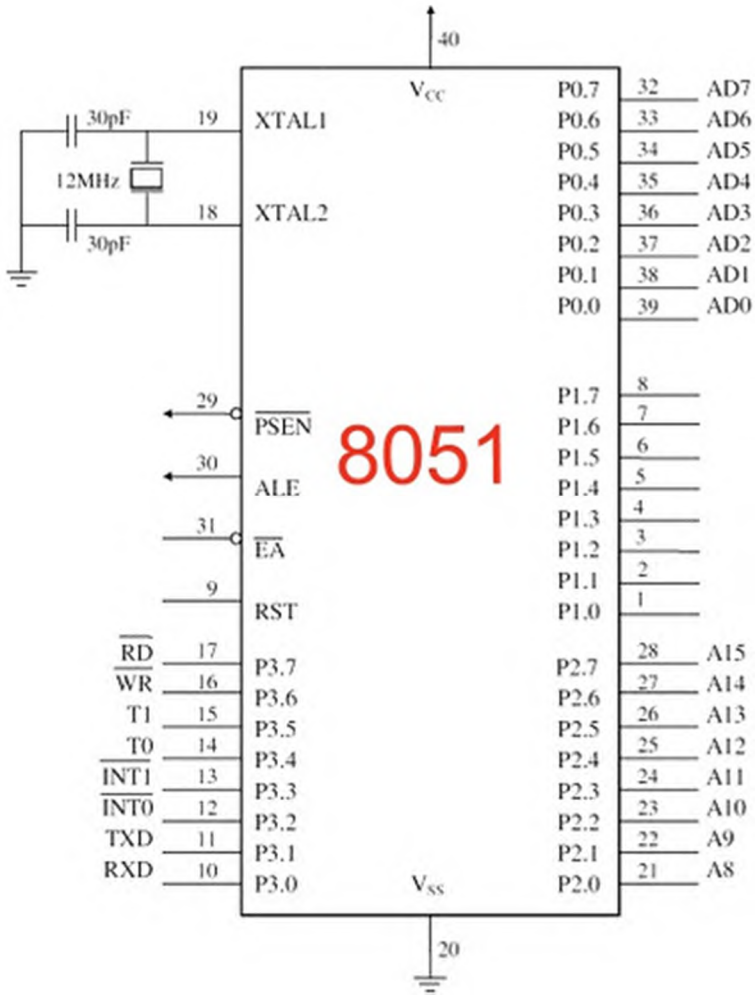
Các port I/O: Các port xuất/nhập → điều khiển việc xuất nhập dữ liệu dưới dạng song song giữa trong và ngoài chip thông qua các port P0; P1; P2; P3.

Port nối tiếp: điều khiển việc xuất nhập dữ liệu dưới dạng nối tiếp giữa trong và ngoài chip thông qua các chân TxD, RxD.

Timer 0, Timer 1: Bộ định thời 0,1 → dùng để định thời gian hoặc đếm sự kiện *(đếm xung)* thông qua các chân T0, T1.

Điều khiển bus: điều khiển hoạt động của hệ thống bus và việc di chuyển thông tin trện hệ thống bus.

2. SƠ ĐỒ CHÂN VÀ CHỨC NĂNG CÁC CHÂN CỦA CHIP 8051



Sơ đồ chân chip 8051

2.1 PORT 0

Port 0 *(P0.0 - P0.7)* có số chân từ 32 – 39.

Port có hai chức năng:

- Port xuất nhập dữ liệu *(P0.0 – P0.7)* → không sử dụng bộ nhớ ngoài.
- Bus địa chỉ byte thấp và bus dữ liệu đa hợp *(AD0 – AD7)* → có sử dụng bộ nhớ ngoài.

***Chú ý*:** Khi port 0 đóng vai trò là port xuất nhập dữ liệu thì phải sử dụng các điện trở kéo lên bên ngoài.

Ở chế độ mặc định (*khi reset)* thì các chân Port 0 *(P0.0 – P0.7)* được cấu hình là **port xuất** dữ liệu. Muốn các chân Port 0 làm **port nhập** dữ liệu thì cần lập trình lại, bằng cách ghi mức logic cao *(mức 1)* đến tất cả các bit của port trước khi bắt đầu nhập dữ liệu từ port.

Khi lập trình cho ROM trong chip thì Port 0 đóng vai trò là ngõ vào của dữ liệu (*D0 – D7).*

2.2 PORT 1

Port 1 *(P1.0 – P1.7*) có chân từ số 1 đến số 8.

Port 1 có một chức năng:

- Port xuất nhập dữ liệu *(P1.0 – P1.7)* → sử dụng hoặc không sử dụng bộ nhớ ngoài.

Ở chế độ mặc định *(khi reset)* thì các chân Port 1 *(P1.0 – P1.7)* được cấu hình là **port xuất** dữ liệu. Muốn các chân Port 1 làm **port nhập** dữ liệu thì cần phải lập trình lại, bằng cách ghi các mức logic cao (*mức 1)* đến tất cả các bit của port trước khi bắt đầu nhập dữ liệu từ port.

Khi lập trình cho ROM trong chip thì Port 1 đóng vai trò là ngõ vào của địa chỉ byte thấp (*A0 – A7).*

2.3 PORT 2

Port 2 *(P2.0 – P2.7)* có chân từ số 21 -28.

Port 2 có hai chức năng:

- Port xuất nhập dữ liệu *(P2.0 – P2.7)* → không sử dụng bộ nhớ ngoài.
- Bus địa chỉ byte cao *(A8 – A15)* → có sử dụng bộ nhớ ngoài.

Ở chế độ mặc định *(khi reset)* thì các chân Port 2 *(P2.0 – P2.7)* được cấu hình là **port xuất** dữ liệu. Muốn các chân Port 2 làm **port nhập** dữ liệu thì cần phải lập trình lại, bằng cách ghi các mức logic cao (*mức 1)* đến tất cả các bit của port trước khi bắt đầu nhập dữ liệu từ port.

Khi lập trình cho ROM trong chip thì Port 2 đóng vai trò là ngõ vào của địa chỉ byte cao (*A8 – A11)* và các tín hiệu điều khiển.

2.4 PORT 3

Port 3 *(P3.0 – P3.7)* có số chân từ 10 -17.

Port 3 có hai chức năng:

- Port xuất nhập dữ liệu *(P3.0 – P3.7).* → không sử dụng bộ nhớ ngoài hoặc các chức năng đặc biệt.
- Các tín hiệu điều khiển → có sử dụng bộ nhớ ngoài hoặc các chức năng đặc biệt.

Ở chế độ mặc định *(khi reset)* thì các chân Port 3 *(P3.0 – P3.7)* được cấu hình là **port xuất** dữ liệu. Muốn các chân Port 3 làm **port nhập** dữ liệu thì cần

phải lập trình lại, bằng cách ghi các mức logic cao (*mức 1)* đến tất cả các bit của port trước khi bắt đầu nhập dữ liệu từ port.

Khi lập trình cho ROM trong chip thì Port 3 đóng vai trò là ngõ vào của các tín hiệu điều khiển.

Chức năng của các chân Port 3:

Bit	Tên	Địa chỉ bit	Chức năng
P3.0	RxD	B0H	Chân nhận dữ liệu của port nối tiếp
P3.1	TxD	B1H	Chân phát dữ liệu của port nối tiếp
P3.2	INT0	B2H	Ngõ vào ngắt ngoài 0
P3.3	INT1	B3H	Ngõ vào ngắt ngoài 1
P3.4	T0	B4H	Ngõ vào của bộ định thời/đếm 0
P3.5	T1	B5H	Ngõ vào của bộ định thời/đếm 1
P3.6	WR	B6H	Điều khiển ghi vào RAM ngoài
P3.7	RD	B7H	Điều khiển đọc từ RAM ngoài

2.5 CHÂN CHO PHÉP BỘ NHỚ CHƯƠNG TRÌNH *(PSEN)*

PSEN *(Program Store Enable)* : chân cho phép bộ nhớ chương trình, chân số 29.

Chức năng:

- Là tín hiệu cho phép truy xuất *(đọc)* bộ nhớ chương trình *(ROM)* ngoài.
- Là tín hiệu xuất, tích cực mức thấp.

PSEN = 0 → trong thời gian CPU tìm – nạp lệnh từ ROM ngoài.

PSEN = 1 → CPU chỉ sử dụng ROM trong *(không sử dụng ROM ngoài)*.

Khi sử dụng bộ nhớ chương trình bên ngoài, chân PSEN thường được nối với chân OE của ROM ngoài để cho phép CPU đọc mã lệnh từ ROM ngoài.

2.6 CHÂN CHO PHÉP CHỐT ĐỊA CHỈ *(ALE)*

ALE (*Address Latch Enable):* cho phép chốt địa chỉ, chân số 30.

Chức năng:

- Là tín hiệu cho phép chốt địa chỉ để thực hiện việc giải đa hợp cho bus địa chỉ byte thấp và bus dữ liệu đa hợp *(AD0 - AD7).*
- Là tín hiệu xuất, tích cực mức cao.

ALE = 0 → trong thời gian bus AD0 – AD7 đóng vai trò là bus D0 –D7.

ALE = 1 → trong thời gian bus AD0 – AD7 đóng vai trò là bus A0 – A7.

Khi lập trình cho ROM trong chip thì chân ALE đóng vai trò là ngõ vào của xung lập trình.

***Chú ý*: $f_{ALE} = f_{OSC}/6$** → có thể dùng làm xung clock cho các mạch khác.

f_{ALE} (MHZ): tần số xung tại chân ALE.

f_{OSC} (MHZ): tần số dao động trên chip (*tần số thạch anh).*

Khi lệnh lấy dữ liệu từ RAM ngoài được thực hiện thì một xung ALE bị bỏ qua.

2.7 CHÂN TRUY XUẤT ROOM NGOÀI (*EA)*

EA (*External Access)*: truy xuất ngoài, chân số 31.

Chức năng:

- Là tín hiệu cho phép truy xuất (*sử dụng)* bộ nhớ chương trình *(ROM)* ngoài.
- Là tín hiệu nhập, tích cực mức thấp.

EA = 0 → Chip 8051 sử dụng chương trình của ROM ngoài.

EA = 1 → Chip 8051 sử dụng chương trình của ROM trong.

Khi lập trình cho ROM trong chip thì chân EA đóng vai trò là ngõ vào của điện áp lập trình.

***Chú ý*:** chân EA phải được nối lên V_{CC} *(nếu sử dụng chương trình của ROM trong)* hoặc nối xuống GND *(nếu sử dụng chương trình của ROM ngoài),* không bao giờ được phép bỏ trống chân này.

2.8 CHÂN RESET *(RST)*

RST (*reset):* thiết lập lại, chân số 9.

Chức năng:

- Là tín hiệu cho phép thiết lập (*đặt)* lại trạng thái ban đầu cho hệ thống.
- Là tín hiệu nhập, tích cực mức cao.

RST = 0 → chip hoạt động bình thường.

RST = 1 → Chip được thiết lập lại trạng thái ban đầu.

2.9 CHÂN XTAL1, XTAL2

XTAL (*Crystal)*: tinh thể thạch anh, chân số 18, 19.

Chức năng:

- Dùng để nối với thạch anh hoặc mạch dao động tạo xung clock bên ngoài, cung cấp tín hiệu xung clock cho chip hoạt động.
- XTAL1 → ngõ vào mạch tạo xung clock trong chip.
- XTAL2 → ngõ ra mạch tạo xung clock trong chip.

2.10 CHÂN VCC VÀ GND

Chân Vcc *(chân số 40)* được nối lên nguồn 5V.

Chân GND *(chân số 20)* được nối mass.

III. CẤU TRÚC PORT I/O

Khả năng fanout (số lượng tải đầu ra) của từng chân port chip 8051 là:

- Port 0: 8 tải TTL.
- Port 1: 4 tải TTL.
- Port 2: 4 tải TTL.
- Port 3: 4 tải TTL.

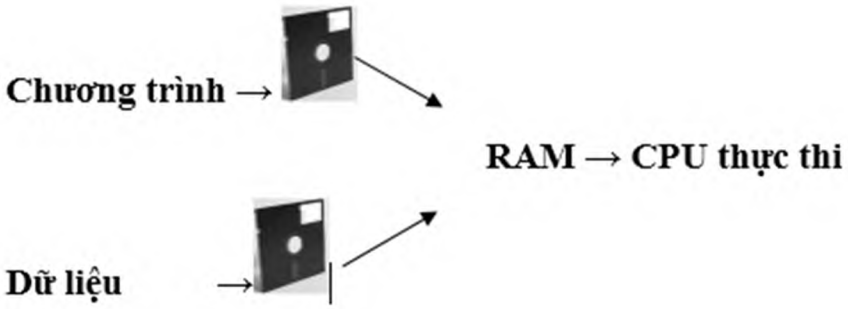
Chú ý:

- Khi Port 0 đóng vai trò là port xuất nhập thì sẽ không có điện trở kéo lên bên trong → do đó người dùng cần thêm vào điện trở kéo lên bên ngoài.
- Ở chế độ mặc định (*khi reset)* thì tất cả các chân của các port *(P0 –P3)* được cấu hình là port xuất dữ liệu.
- Muốn các chân của chip 8051 làm port nhập dữ liệu thì ta cần phải lập trình lại bằng cách ghi mức logic cao (*mức 1)* đến tất cả các bit (*các chân)* của port trước khi bắt đầu nhập dữ liệu từ port.
- Các chân trong cùng một port không nhất thiết phải có cùng kiểu cấu hình (*port xuất hoặc port nhập)*. Nghĩa là trong cùng một port có thể có chân dùng để nhập dữ liệu, có thể có chân dùng để xuất dữ liệu. Điều này tuỳ thuộc vào nhu cầu và mục đích của người lập trình.

***Chú ý*:** việc đọc dữ liệu của bất kỳ một port nào có thể cho ta hai giá trị khác nhau tuỳ thuộc vào lệnh mà ta sử dụng để đọc dữ liệu từ port. Xảy ra hiện tượng không mong muốn này là do quá trình đọc dữ liệu của chip 8051 gồm hai quá trình khác nhau: **quá trình đọc chân port và quá trình đọc bộ chốt.**

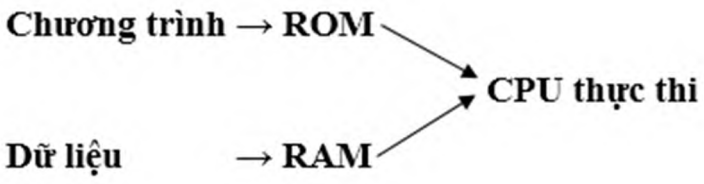
IV. TỔ CHỨC BỘ NHỚ

Bộ vi xử lý → có không gian bộ nhớ chung cho dữ liệu và chương trình.



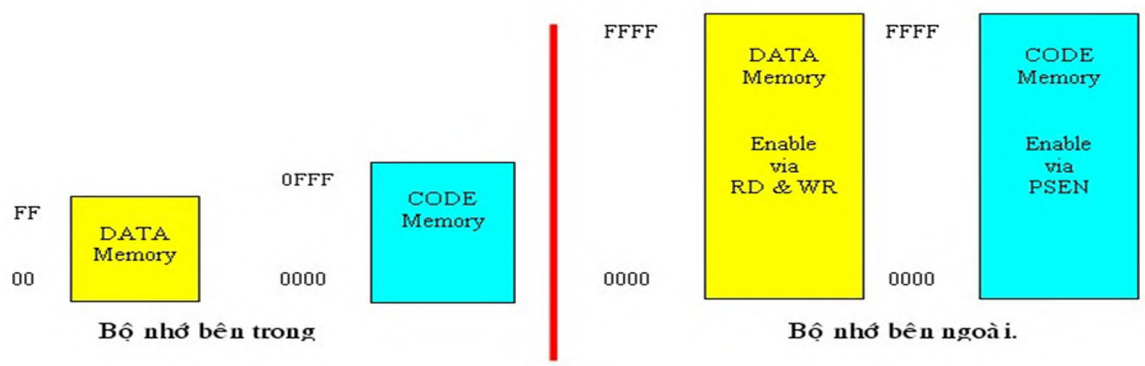
→ Chương trình và dữ liệu nằm chung trên RAM trước khi đưa vào CPU để thực thi.

Bộ vi điều khiển → có không gian bộ nhớ riêng cho dữ liệu và chương trình.

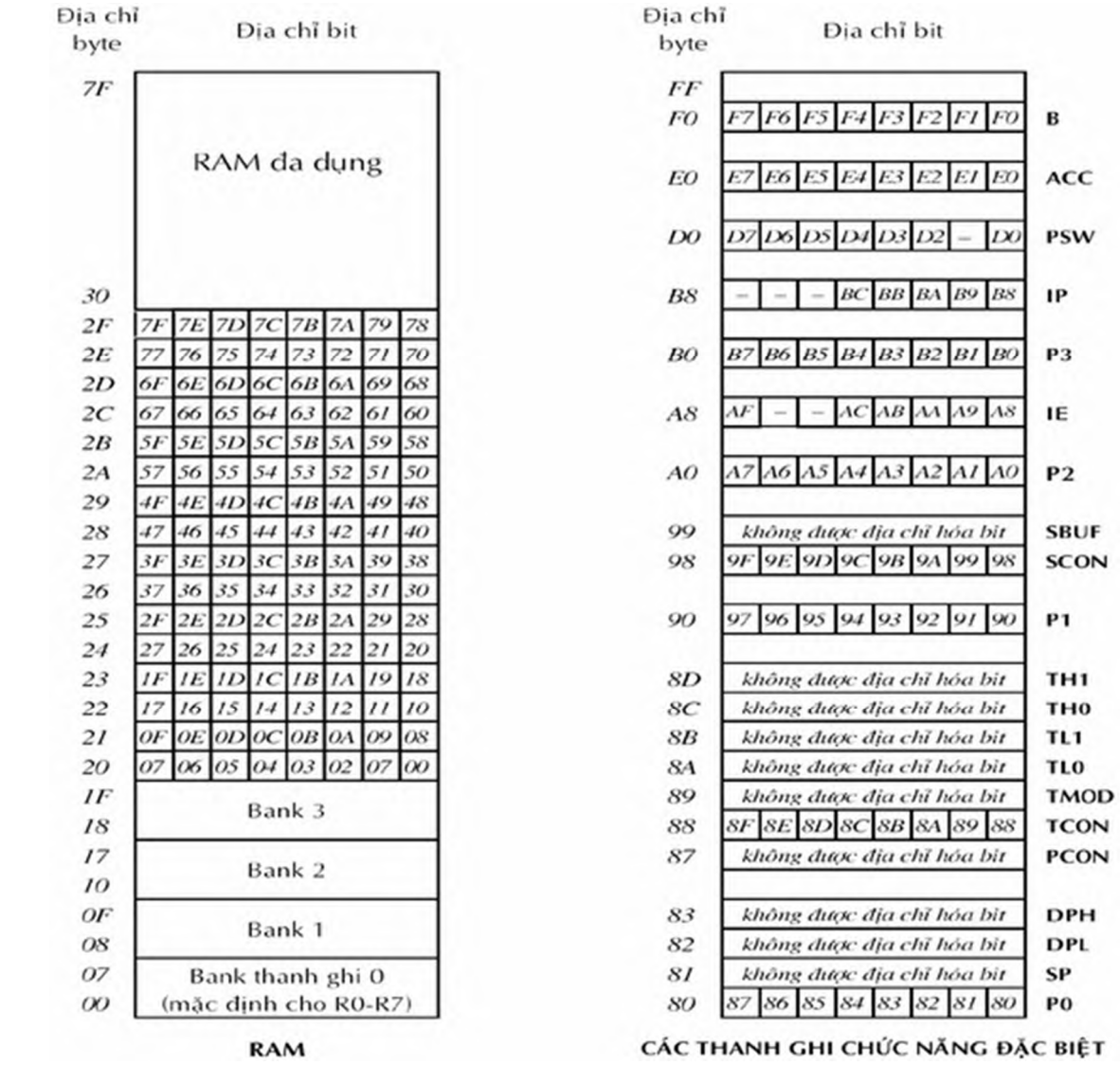


→ Chương trình và dữ liệu nằm trên ROM và RAM trước khi đưa vào CPU để thực thi.

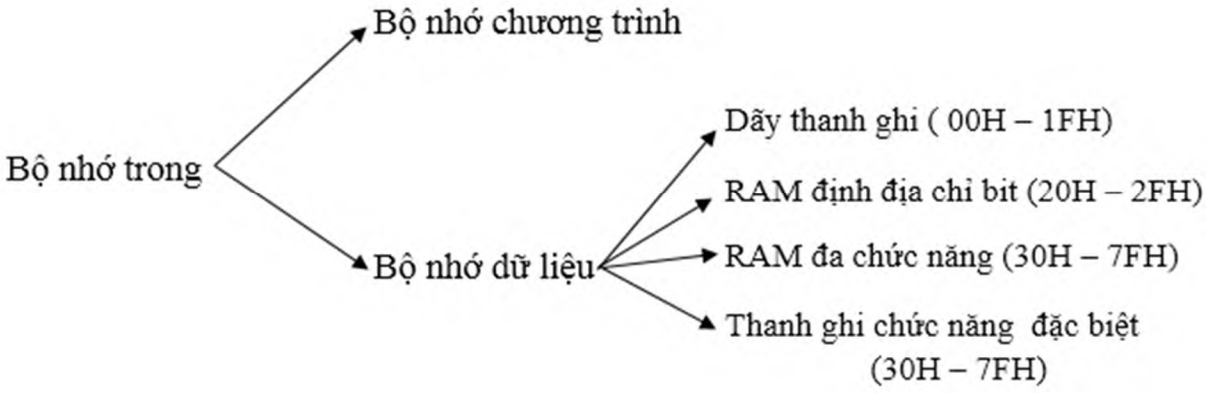
Tổ chức bộ nhớ chip 8051:



Không gian bộ nhớ chip 8051



Bộ nhớ dữ liệu trên chip 8501



Bộ nhớ chương trình (ROM):

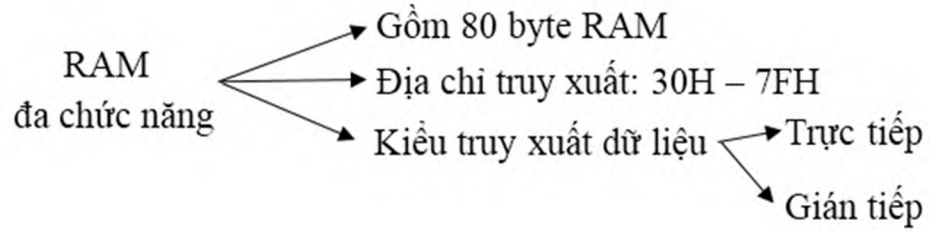
- Dùng để lưu trữ chương trình điều khiển cho chip 8051 hoạt động.

- Chip 8051 có 4 KB ROM trong, địa chỉ truy xuất: 000H – FFFH.

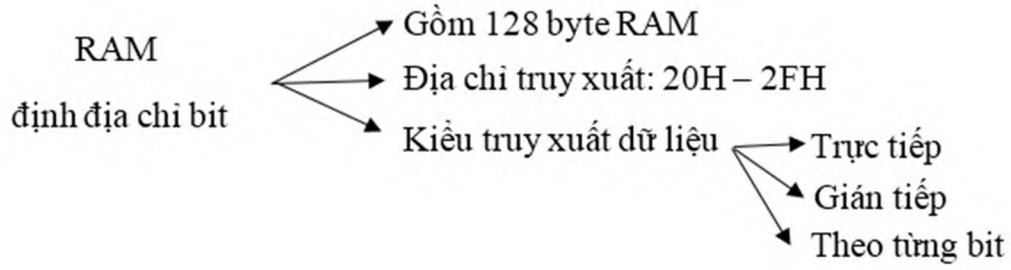
Bộ nhớ dữ liệu (RAM):

- Dùng để lưu trữ các dữ liệu và tham số.
- Chip 8051 có 128 byte RAM trong, địa chỉ truy xuất: 00H – 7FH.
- RAM trong của chip 8051 được chia ra:

RAM đa chức năng:

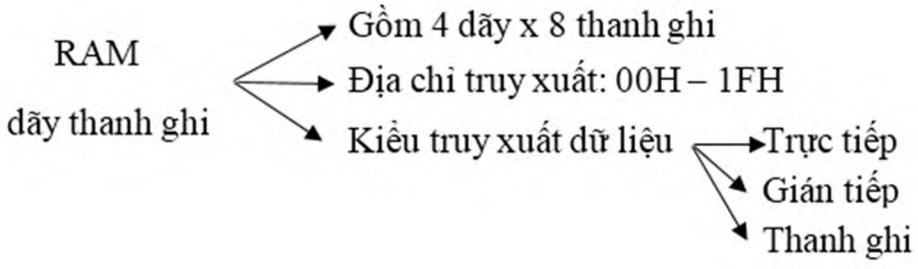


RAM định địa chỉ bit:



***Chú ý*:** Nếu trong chương trình không sử dụng các bit trong vùng RAM định địa chỉ bit này, ta có thể sử dụng vùng nhớ 20H – 2FH cho các mục đích khác. Ngược lại, ta phải viết chương trình cẩn thận khi sử dụng vùng nhớ 20H – 2FH vì nếu sơ suất ta có thể ghi dữ liệu đè lên các bit đã được sử dụng.

Các dãy thanh ghi: Cho phép truy xuất dữ liệu nhanh, lệnh truy xuất đơn giản và ngắn gọn.



Bảng số liệu dưới đây minh hoạ địa chỉ của các ô nhớ trong một dãy và các ký hiệu thanh ghi R0 – R7 được gán cho từng ô nhớ trong dãy tích cực:

	Dãy 0	**Dãy 1**	**Dãy 2**	**Dãy 3**

R0	00H	08H	10H	18H
R1	01H	09H	11H	19H
R2	02H	0AH	12H	1AH
R3	03H	0BH	13H	1BH
R4	04H	0CH	14H	1CH
R5	05H	0DH	15H	1DH
R6	06H	OEH	16H	1EH
R7	07H	0FH	17H	1FH

Địa chỉ của các thanh ghi (R0 – R7) tương ứng với các dãy thanh ghi tích cực

Chú ý:

- Ở chế độ mặc định thì dãy thanh ghi tích cực *(đang được sử dụng)* là dãy 0 và các thanh ghi lần lượt trong dãy có tên là **R0 – R7.** Có thể thay đổi dãy tích cực bằng cách thay đổi các bit chọn dãy thanh ghi RS1 và RS0 trong thanh ghi PSW.
- Nếu chương trình của ta chỉ sử dụng dãy thanh ghi đầu tiên (*dãy 0)* thì ta có thể sử dụng vùng nhớ 08H – 1FH cho các mục đích khác. Nhưng nếu trong chương trình có sử dụng các dãy thanh ghi (*dãy 1,2 hoặc 3)* thì phải cẩn thận khi sử dụng vùng nhớ từ 1FH trở xuống vì nếu sơ suất ta có thể ghi dữ liệu đè lên các thanh ghi R0 –R7.

V. CÁC THANH GHI CHỨC NĂNG ĐẶC BIỆT

1. THANH GHI TỪ PSW

BIT	SYMBOL	ADDRESS	DESCRIPTION
PSW.7	CY	D7H	Cary Flag
PSW.6	AC	D6H	Auxiliary Cary Flag
PSW.5	F0	D5H	Flag 0
PSW4	RS1	D4H	Register Bank Select 1

PSW.3	RS0	D3H	Register Bank Select 0
			00=Bank 0; address 00H÷07H
			01=Bank 1; address 08H÷0FH
			10=Bank 2; address 10H÷17H
			11=Bank 3; address 18H÷1FH
PSW.2	OV	D2H	Overlow Flag
PSW.1	-	D1H	Reserved
PSW.0	P	DOH	Even Parity Flag

Bảng thể hiện chức năng các bit thanh ghi PSW

PSW: Program Status Word → từ trạng thái chương trình.

Địa chỉ byte: D0H.

Địa chỉ bit: D0H –D7H.

Công dụng: cho biết trạng thái làm việc của CPU, mỗi bit của PSW có một chức năng *(thể hiện tại bảng chức năng các bit).*

1.1 Cờ CY (Cary Flag)

Là cờ nhớ → báo có nhớ/mượn tại bit 7.

- CY = 0: nếu không có nhớ từ bit 7 hoặc không có mượn cho bit 7.
- CY = 1: nếu có nhớ từ bit 7 hoặc có mượn cho bit 7.

Cờ AC (Auxiliary Carry)

Là cờ nhớ phụ → báo có nhớ/mượn tại bit 3.

- AC = 0: nếu không có nhớ từ bit 3 hoặc không có mượn cho bit 3.
- AC = 1: nếu có nhớ từ bit 3 hoặc có mượn cho bit 3.

1.2 **CỜ F0 (Flag 0*)***

Là cờ zero → có nhiều mục đích dành cho các ứng dụng khác nhau của người lập trình.

1.3 **BIT RS0, RS1 (Register Select*)***

Là bit chọn dãy thanh ghi → cho phép xác định dãy thanh ghi tích cực *(hay dãy thanh ghi mà các thanh ghi có tên là R0 – R7).*

RS 1	RS 0	DÃY THANH GHI	R0 – R7
0	0	Dãy 0	00H – 07H
0	1	Dãy 1	08H -0FH
1	0	Dãy 2	10H -17H
1	1	Dãy 3	18H – 1FH

1.4 CỜ OV (*Overflow)*

Là cờ tràn → báo kết quả tính toán của phép toán số học (*phép toán có dấu)* có nằm trong khoảng từ -128 đến +127 hay không.

- OV = 0: nếu -128 ≤ kết quả ≤ +127.
- OV = 1: nếu kết quả < -128 hoặc kết quả > +127. Nói cách khác: đối với **phép cộng** thì OV = 1 nếu có nhớ từ bit 7 nhưng không có nhớ từ bit 6 hoặc nếu có nhớ từ bit 6 nhưng không có nhớ từ bit 7. Đối với **phép trừ** thì OV =1 nếu có mượn cho bit 7 nhưng không có mượn cho bit 6 hoặc nếu có mượn bit 6 nhưng không có mượn bit 7.

1.5 CỜ P (*Parity)*

Là cờ chẵn lẻ → báo số chữ số 1 trong thanh ghi A là số chẵn hay số lẻ (*chip 8051 sử dụng chế độ parity chẵn).*

- P = 0: nếu số chữ số 1 trong thanh ghi A là số chẵn (*parity chẵn).*
- P = 1: nếu số chữ số 1 trong thanh ghi A là số lẻ (*parity chẵn).*

2. THANH GHI A

Địa chỉ byte: E0H.

Địa chỉ bit: E0H – E7H.

Công dụng: chứa dữ liệu của các phép toán mà vi điều khiển xử lý.

3. THANH GHI B

Địa chỉ byte: F0H.

Địa chỉ bit: F0H – F7H.

Công dụng: sử dụng kết hợp với thanh ghi A trong phép nhân và chia hai số 8 bit.

Phép nhân 2 số 8 bit không dấu → kết quả là số 16 bit.

- Byte cao → chứa vào thanh ghi B.
- Byte thấp → chứa vào thanh ghi A.

Phép chia 2 số 8 bit → thương số và số dư là số 8 bit.

- Thương số → chứa vào thanh ghi A.
- Số dư → chứa vào thanh ghi B.

4. CON TRỎ STACK (Stack Pointer)

Địa chỉ byte: 81H.

Địa chỉ bit: không định địa chỉ bit.

Công dụng: chứa địa chỉ của dữ liệu hiện đang ở đỉnh ngăn xếp.

- Ngăn xếp là vùng nhớ dùng để lưu trữ tạm thời các dữ liệu.
- Đối với chip 8051 thì vùng nhớ được dùng để làm ngăn xếp được giữ trong RAM nội.
- Để sử dụng ngăn xếp thì ta phải khởi động thanh ghi SP (*nghĩa là nạp giá trị cho thanh ghi SP)* → vùng nhớ của ngăn xếp *có địa chỉ bắt đầu*: **(SP) +1** và *địa chỉ kết thúc*: **7FH.**
- Nếu không khởi động SP → vùng nhớ của ngăn xếp *có địa chỉ bắt đầu*: **08H** *và địa chỉ kết thúc:* **7FH** (*chế độ mặc định).*

***Chú ý*:** Trong trường hợp không khởi động SP (*chế độ mặc định)* thì dãy thanh ghi 1 (*và có thể là dãy 2 và dãy 3)* sẽ không còn hợp lệ vì khi đó vùng nhớ này đã được sử dụng để làm ngăn xếp. Điều này có nghĩa là nếu ta sử dụng các dãy thanh ghi này và lưu trữ dữ liệu vào đó thì có khả năng sẽ bị mất do tác động cất dữ liệu vào ngăn xếp của các lệnh.

5. CON TRỎ DỮ LIỆU DPTR (Data Pointer Register)

Đây là thanh con trỏ dữ liệu.

Địa chỉ byte: 83H và 82H.

Địa chỉ bit: không định địa chỉ bit.

Công dụng: là thanh ghi 16 bit *(DPH + DPL),* chứa địa chỉ của ô nhớ cần truy xuất thuộc ROM *(trong/ngoài)* và RAM ngoài.

6. THANH GHI PORT XUẤT NHẬP

Port: cổng xuất nhập.

Địa chỉ byte: 80H *(P0),* 90H *(P1),* A0H *(P2),* B0H *(P3).*

Địa chỉ bit: 80H – 87H *(P0),* 90H -97H *(P1),* A0H – A7H *(P2),* B0H – B7H *(P3).*

Công dụng: chứa dữ liệu hiện tại trên các port xuất nhập.

Chú ý:

- Trong trường hợp phần cứng có sử dụng ROM hoặc RAM bên ngoài thì ta không thể sử dụng Port 0 và Port 2 để xuất dữ liệu. Vì khi đó chip 8051 sẽ sử dụng hai port này để xác định địa chỉ và dữ liệu cho bộ nhớ ngoài. Khi đó, ta chỉ có thể sử dụng Port 1 và Port 3 để xuất nhập dữ liệu.
- Ở chế độ mặc định (*khi reset)* thì tất cả các chân của port (P0 – P3) được cấu hình là **port xuất** dữ liệu. Muốn các chân port của chip 8051 làm **port nhập** dữ liệu thì ta cần lập trình lại, bằng cách ghi mức logic cao (*mức 1)* đến tất cả các bit (*các chân)* của port trước khi bắt đầu nhập dữ liệu từ port.

7. CÁC THANH GHI ĐỊNH THỜI

7.1 THANH GHI TIMER 0

Timer 0: bộ định thời 0.

Địa chỉ byte: 8CH và 8AH.

Địa chỉ bit: không định địa chỉ bit.

Công dung: là thanh ghi bộ định thời 16 bit *(TH0 + TL0),* chứa giá trị hiện tại của bộ định thời 0.

7.2 THANH GHI TIMER 1

Timer 1: bộ định thời 1.

Địa chỉ byte: 8DH và 8BH.

Địa chỉ bit: không định thời địa chỉ bit.

Công dụng: là thành ghi bộ định thời 16 bit *(TH1 + TL1),* chứa giá trị hiện tại của bộ định thời 1.

7.3 THANH GHI TMOD

Timer Mode: Chế độ bộ định thời.

Địa chỉ byte: 89H.

Địa chỉ bit: không định địa chỉ bit.

Công dụng: qui định chế độ hoạt động cho cả hai bộ định thời T0 và T1.

7.4 THANH GHI TCON

Timer Control: điều khiển bộ định thời.

Địa chỉ byte: 88H.

Địa chỉ bit: 88H – 8FH.

Công dụng: báo trạng thái và điều khiển quá trình hoạt động của hai bộ định thời T0 và T1.

8. CÁC THANH GHI PORT NỐI TIẾP

8.1 THANH GHI SBUF

Serial Buffer: bộ đệm port nối tiếp.

Địa chỉ byte: 99H.

Địa chỉ bit: không định địa chỉ bit.

Công dụng: chứa dữ liệu cần truyền ra bên ngoài hay dữ liệu nhận được từ bên ngoài dưới dạng nối tiếp.

8.2 THANH GHI SCON

Serial Control: điều khiển port nối tiếp.

Địa chỉ byte: 98H.

Địa chỉ bit: 98H – 9FH.

Công dụng: báo trạng thái và điều khiển quá trình hoạt động của port nối tiếp.

9. CÁC THANH GHI NGẮT

9.1 THANH GHI IE

Interrupt Enable: cho phép ngắt.

Địa chỉ byte: A8H.

Địa chỉ bit: A8H – AFH.

Công dụng: cho phép hoặc không cho phép các ngắt hoạt động *(từng ngắt*

riêng lẻ hoặc tất cả các ngắt).

9.2 THANH GHI IP

Interrupt Priority: ưu tiên ngắt.

Địa chỉ byte: B8H.

Địa chỉ bit: B8H – BCH.

Công dụng: thiết lập mức ưu tiên cho các ngắt *(ưu tiên thấp hoặc ưu tiên cao).*

10. THANH GHI ĐIỀU KHIỂN NGUỒN (Thanh ghi PCON).

Power Control: điều khiển nguồn.

Địa chỉ byte: 87H.

Địa chỉ bit: không định địa chỉ bit.

Công dụng: điều khiển tốc độ baud của port nối tiếp, điều khiển chế độ nguồn giảm và chế độ nghỉ của chip.

- Bit SMOD *(Serial Mode)* → cho phép tăng gấp đôi tốc độ truyền dữ liệu nối tiếp (*tốc độ baud)* khi SMOD =1.
- Bit GF1, GF0 (*General Function)* → cho phép người lập trình dùng với mục đích riêng.
- Bit PD (*Power Down)* → dùng để qui định chế độ nguồn giảm.
- Bit IDL (*Idle)* → dùng để qui định chế độ nghỉ.

Ở chế độ nguồn giảm *(PD = 1):*

- ✓ Mạch dao động trên chip ngừng hoạt động.
- ✓ Mọi chức năng ngừng hoạt động.
- ✓ Nội dung của RAM trên chip được duy trì.
- ✓ Các chân port duy trì mức logic của chúng.
- ✓ ALE = 0, PSEN = 0, V_{cc} = 2V.
- ✓ Thoát khỏi hệ thống bằng cách reset hệ thống.

***Chú ý*:** Hệ thống phải phục hồi V_{cc} = 5V trước khi thoát khỏi chế độ nguồn giảm.

Ở chế độ nghỉ (*IDL = 1):*

- ✓ Mạch dao động trên chip ngừng hoạt động
- ✓ Các chức năng ngắt, định thời và port nối tiếp còn hoạt động.
- ✓ Nội dung của RAM trên chip được duy trì.
- ✓ Các chân port duy trì mức logic của chúng.
- ✓ ALE =1, PSEN = 1, V_{cc} = 5V.
- ✓ Thoát khỏi chế độ bằng cách reset hoặc bằng cách cho phép ngắt.

VI. BỘ NHỚ NGOÀI

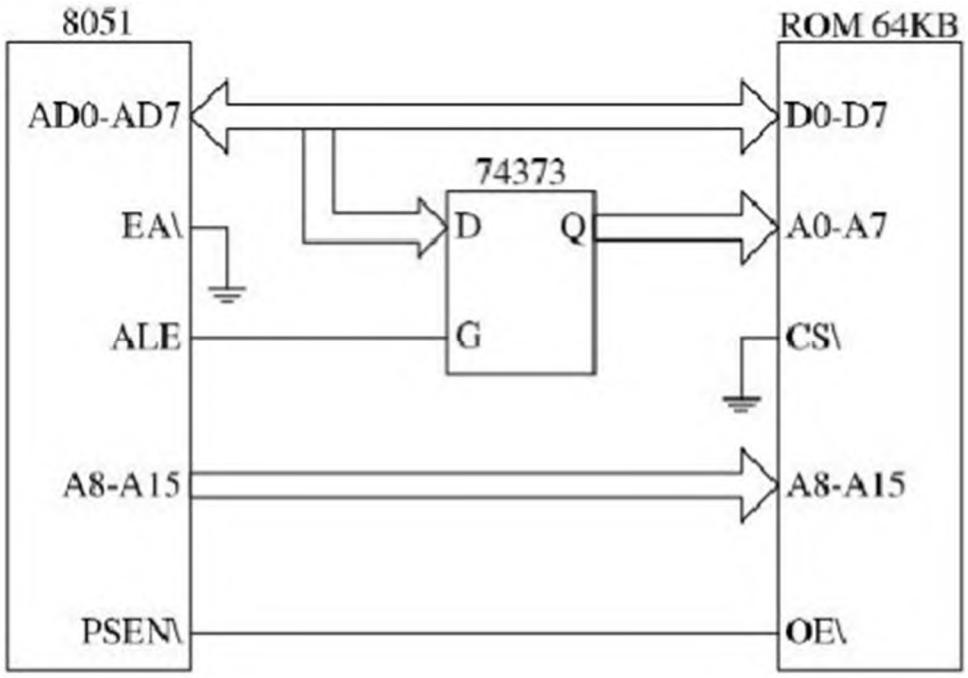
Chip 8051 cho ta khả năng mở rộng:

- Không gian bộ nhớ chương trình lên đến 64 KB.
- Không gian bộ nhớ dữ liệu lên đến 64 KB.

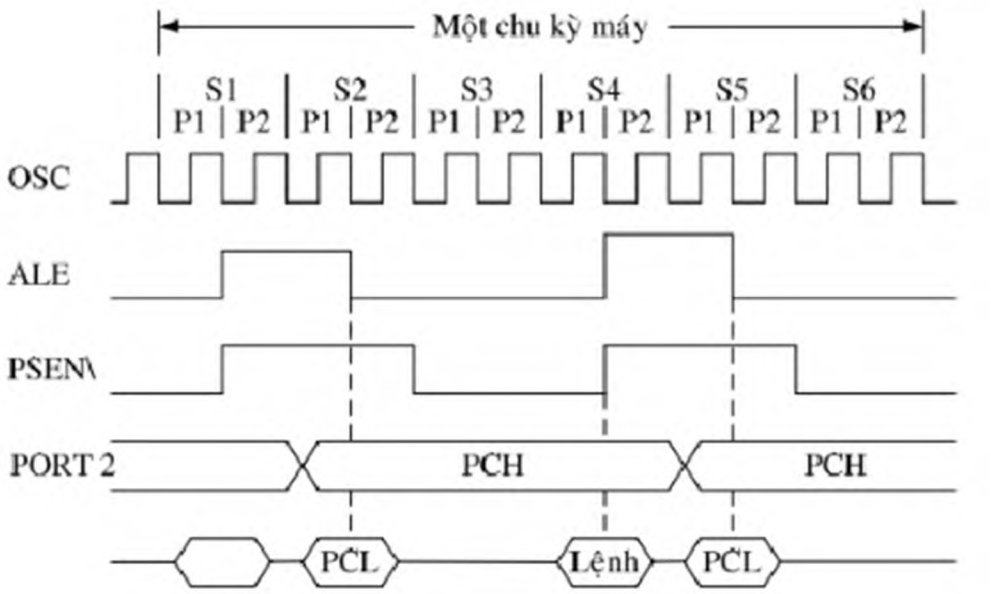
Khi sử dụng bộ nhớ ngoài:

- Port 0 → bus địa chỉ byte thấp và bus dữ liệu đa hợp (*AD0 – AD7).*
- Port 2 → bus địa chỉ byte cao (*A8 – A15).*
- Port 3 → các tín hiệu điều khiển (*WR, RD).*

1. KẾT NỐI VÀ TRUY XUẤT BỘ NHỚ CHƯƠNG TRÌNH NGOÀI

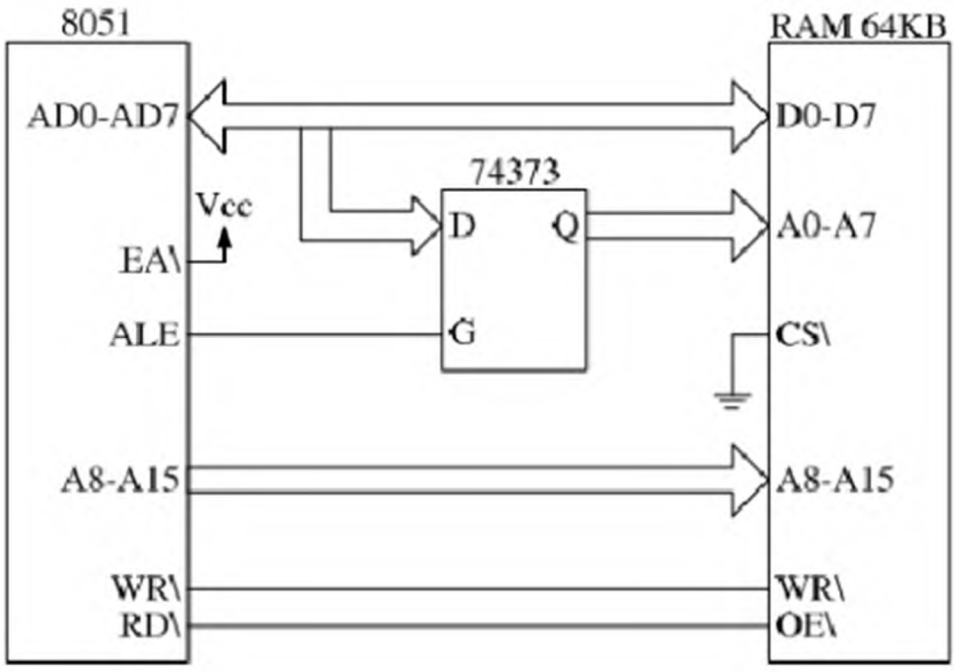


Truy xuất bộ nhớ chương trình ngoài



Giản đồ thời gian của chu kỳ tìm nạp lệnh ở bộ nhớ ngoài

2. KẾT NỐI VÀ TRUY XUẤT BỘ NHỚ DỮ LIỆU NGOÀI



Truy xuất bộ nhớ dữ liệu ngoài

3. GIẢI MÃ ĐỊA CHỈ

Trường hợp RAM và ROM được kết hợp từ nhiều bộ nhớ có dung lượng nhỏ hoặc cả hai giao tiếp với chip 8051 thì ta cần phải giải mã địa chỉ. Việc giải mã địa chỉ này cũng cần cho hầu hết các bộ vi xử lý.

Ví dụ: nếu các ROM và RAM có dung lượng 8 KB được sử dụng thì tầm địa chỉ mà chip 8051 quản lý (*0000H – FFFFH)* cần phải được giải mã thành từng đoạn 8KB để chip có thể chọn từng IC nhớ trên các giới hạn 8KB tương ứng: IC1: 0000H – 1FFFH; IC2: 2000H – 3FFFH…

IC chuyên dùng cho việc tạo tín hiệu giải mã là 74HC138, các ngõ ra của IC này lần lượt được nối với các ngõ vào chọn chip CS tương ứng của các IC nhớ để cho phép IC nhớ hoạt động (*tại một thời điểm chỉ có 1 IC nhớ được phép hoạt động).* Cần lưu ý là do các đường cho phép IC nhớ hoạt động riêng lẻ cho từng loại (*PSEN cho bộ nhớ chương trình, RD và WR cho bộ nhớ dữ liệu)* nên 8051 có thể quản lý không gian nhớ lên đến 64KB cho ROM và 64 KB cho RAM.

VII. HOẠT ĐỘNG RESET

Chân RST = 0 → chip 8051 hoạt động bình thường.

Chân RST bằng 1 → chip 8051 được reset.

Chú ý:

- Để hoàn tất quá trình reset thì chân RST phải ở mức cao tối thiểu là 2 chu kỳ máy và sau đó chuyển xuống mức thấp.
- Nội dung của RAM trong chip không bị ảnh hưởng bởi hoạt động reset.
- Sau khi reset, việc thực thi chương trình luôn bắt đầu ở vị trí đầu tiên trong bộ nhớ chương trình: Địa chỉ 0000H.
- Trạng thái của các thanh ghi sau khi reset hệ thống:
  - ✓ Bộ đếm chương trình (*PC)* 0000H
  - ✓ Thanh ghi A 00H
  - ✓ Thanh ghi B 00H
  - ✓ Thanh ghi PSW 00H
  - ✓ Thanh ghi SP 07H
  - ✓ Thanh ghi DPTR 0000H

Giới thiệu:

Trong những thập niên cuối thế kỉ XX, từ sự ra đời của công nghệ bán dẫn, kỹ thuật điện tử đã có sự phát triển vượt bậc. Các thiết bị điện tử sau đó đã được tích hợp với mật độ cao và rất cao trong các diện tích nhỏ, nhờ vậy các thiết bị nhỏ hơn và nhiều chức năng hơn. Các thiết bị điện tử ngày càng nhiều chức năng trong khi giá thành ngày càng rẻ hơn, chính vì vậy điện tử có mặt khắp nơi. Bước đột phát mới trong kỹ thuật điện tử là tạo ra một bộ Vi điều khiển.

Vi điều khiển(Microcontroller) có khả năng tính toán, xử lý, và thay đổi chương trình linh hoạt theo mục đích người dùng, đặc biệt hiệu quả đối với các bài toán và hệ thống, nhưng cấu trúc phần cứng dành cho người dùng đơn giản. Vi điều khiển ra đời mang lại sự tiện lợi đối với người dùng, họ không cần nắm vững một khối lượng kiến thức quá lớn như người dùng vi xử lý, kết cấu mạch điện dành cho người dùng cũng trở nên đơn giản hơn nhiều và có khả năng giao tiếp trực tiếp với các thiết bị bên ngoài.

Vi điều khiển tuy được xây dựng với phần cứng dành cho người sử dụng đơn giản hơn, nhưng thay vào lợi điểm này là khả năng xử lý bị giới hạn. Vì Vi điều khiển có giá thành rẻ hơn nhiều so với vi xử lý, việc sử dụng đơn giản nên nó được ứng dụng rộng rãi vào nhiều ứng dụng có chức năng đơn giản, không đòi hỏi tính toán phức tạp. Do đó, để nắm được hoạt động của các hệ thống dùng vi điều khiển ta phải tìm hiểu cấu trúc của họ vi điều khiển 8051.

Mục tiêu:

Mô tả được cấu trúc họ vi điều khiển chuẩn công nghiệp

- Thực hiện truy xuất bộ nhớ dữ liệu, bộ nhớ chương trình đúng qui trình kỹ thuật
- Thực hiện đúng kỹ thuật phương pháp mở rộng bộ nhớ ngoài.
- Trình bày được nguyên lý hoạt động của mạch reset

Nội dung chính:

I. TỔNG QUAN

4. LỊCH SỬ PHÁT TRIỂN CỦA 8051

Vào năm 1981, hãng Intel giới thiệu một số bộ vi điều khiển có tên gọi là 8051. Bộ vi điều khiển này có 128 byte RAM, 4KB room trên chip, hai bộ định thời, một cổng nối tiếp và 4 cổng *(đều có độ rộng 8 bit)* vào ra, tất cả được đặt trên một chip. Lúc ấy, nó được coi là một " Hệ thống trên chip". 8051 là một bộ xử lý 8

bit, điều đó có nghĩa là CPU chỉ có thể làm việc với 8 bit dữ liệu tại một thời điểm. Dữ liệu lớn hơn 8 bit được chia ra thành các dữ liệu 8 bit để xử lý. 8051 có tất cả 4 cổng vào – ra *(I/O),* mỗi cổng rộng 8 bit. Mặc dù 8051 có thể có một ROM trên chip cực đại là 64KB, nhưng các nhà sản xuất lúc đó đã cho xuất xưởng chỉ với 4KB ROM trên chip.

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

Đặc tính	**Số lượng**
ROM trên chíp	4KB
RAM	128 byte
Bộ định thời	2
Các chân vào ra	32
Cổng nối tiếp	1
Nguồn ngắt	6

Các đặc tính của 8051 đầu tiên

5. BỘ VI ĐIỀU KHIỂN 8051

MCS-51 là họ vi điều khiển của hãng Intel. Vi mạch tổng quát của họ MCS-51 là chip 8051. Chip 8051 có một số đặc trưng cơ bản sau:

- Bộ nhớ chương trình bên trong: 4KB *(ROM).*
- Bộ nhớ dữ liệu bên trong: 128 byte *(RAM).*
- Bộ nhớ chương trình bên ngoài: 64 KB *(ROM)*
- Bộ nhớ dữ liệu bên ngoài: 64 KB I
- 4 port xuất nhập *(I/O port)* 8 bit.

- 2 bộ định thời 16 bit.
- Mạch giao tiếp nối tiếp.
- Bộ xử lý bit *(thao tác trên các bit riêng lẻ).*
- 210 vị trí nhớ được định địa chỉ, mỗi vị trí 1 bit.
- Nhân/Chia trong 4 µs.

6. CÁC THÀNH VIÊN KHÁC CỦA HỌ 8051

3.6 BỘ VI ĐIỀU KHIỂN 8031

8031 là một phiên bản khác của họ 8051. Chíp này thường được coi như là 8051 không có ROM trên chíp. Để có thể dùng được chip này cần phải bổ sung thêm ROM ngoài chứa chương trình cần thiết cho 8031. 8051 có chương trình được chứa ở ROM trên chip bị giới hạn đến 4KB, còn ROM ngoài của 8031 có thể lên đến 64 KB. Tuy nhiên, để có thể truy cập hết bộ nhớ ROM ngoài thì cần dùng thêm 2 cổng (*Port 0 và Port 2*), do vậy chỉ còn lại hai cổng (*Port 1 và Port 3*) để sử dụng. Nhằm khác phục vấn đề này, chúng ta có thể bổ sung thêm cổng vào/ra cho 8031.

3.7 BỘ VI ĐIỀU KHIỂN 8052

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

Đặc tính	**8051**	**8052**	**8031**
ROM trên chip	4KB	8KB	0K
RAM	128 byte	256 byte	128 byte
Bộ định thời	2	3	2
Chân vào ra	32	32	32
Cổng nối tiếp	1	1	1
Nguồn ngắt	6	8	6

So sánh các đặc tính của các thành viên họ 8051

Chúng ta có thể thấy 8051 là tập con của 8052. Do vậy tất cả mọi chương

trình viết cho 8051 đều có thể chạy trên 8052 nhưng điều ngược lại là không đúng.

3.8 BỘ VI ĐIỀU KHIỂN 8751

Chip 8751 chỉ có 4KB bộ nhớ UV-EROM trên chip. Để sử dụng chip này cần phải có thiết bị lập trình ROM và thiết bị xoá UV-EPROM. Do ROM trên chip 8751 là UV-EPROM nên cần mất 20 phút để xoá 8751 trước khi lập trình. Vì quá trình này mất quá nhiều thời gian nên nhiều nhà sản xuất đã cho ra phiên bản Flash ROM.

3.9 BỘ VI ĐIỀU KHIỂN AT8951 CỦA ATMEL CORPORATION

AT8951 là phiên bản 8051 có ROM trên chip là bộ nhớ Flash. Phiên bản này rất thích hợp cho các ứng dụng nhanh vì bộ nhớ Flash có thể được xoá trong vài giây. Dĩ nhiên, để dùng AT8951 cần phải có thiết bị lập trình PROM hỗ trợ bộ nhớ Flash nhưng không cần đến thiết bị xoá ROM vì bộ nhớ Flash sè được xoá bằng thiết bị lập trình PROM. Để tiện sử dụng, hiện nay hãng Atmel đang nghiên cứu một phiên bản của AT8951 có thể lập trình qua cổng COM của máy tính PC và như vậy sẽ không cần đến thiết bị lập trình PROM.

Ký hiệu	ROM	RAM	I/O	Timer	Ngắt thực tế	VCC	Số chân IC
AT89C51	4KB	128	32	2	5	5V	40
AT89LV51	4KB	128	32	2	5	3V	40
AT89C1051	1KB	64	15	1	3	3V	20
AT89C2051	2KB	128	15	2	5	3V	20
AT89C52	8KB	256	32	3	6	5V	40
AT89LV52	8KB	256	32	3	6	3V	40

3.10 BỘ VI ĐIỀU KHIỂN DS5000 CỦA DALLAS SEMICONDUCTOR

Bộ nhớ ROM trên chip của DS5000 là NV-RAM. DS5000 có khả năng nạp chương trình vào ROM trên chip trong khi nó vẫn ở trong hệ thống mà không cần phải lấy ra, Cách thực hiện là dùng cổng COM máy tính PC. Đây là một điểm

mạnh rất được ưa chuộng. Ngoài ra, NV-RAM còn có ưu việt là cho phép thay đổi nội dung RAM theo từng byte mà không cần phải xoá hết trước khi lập trình như bộ nhớ EPROM.

Ký hiệu	ROM	RAM	I/O	Timer	Ngắt thực tế	VCC	Số chân IC
DS5000-8	8KB	128	32	2	6	5V	40
DS5000-32	32KB	128	32	2	6	5V	40
DS5000T-8	8KB	128	32	2	6	5V	40
DS5000T-32	32KB	128	32	2	6	5V	40

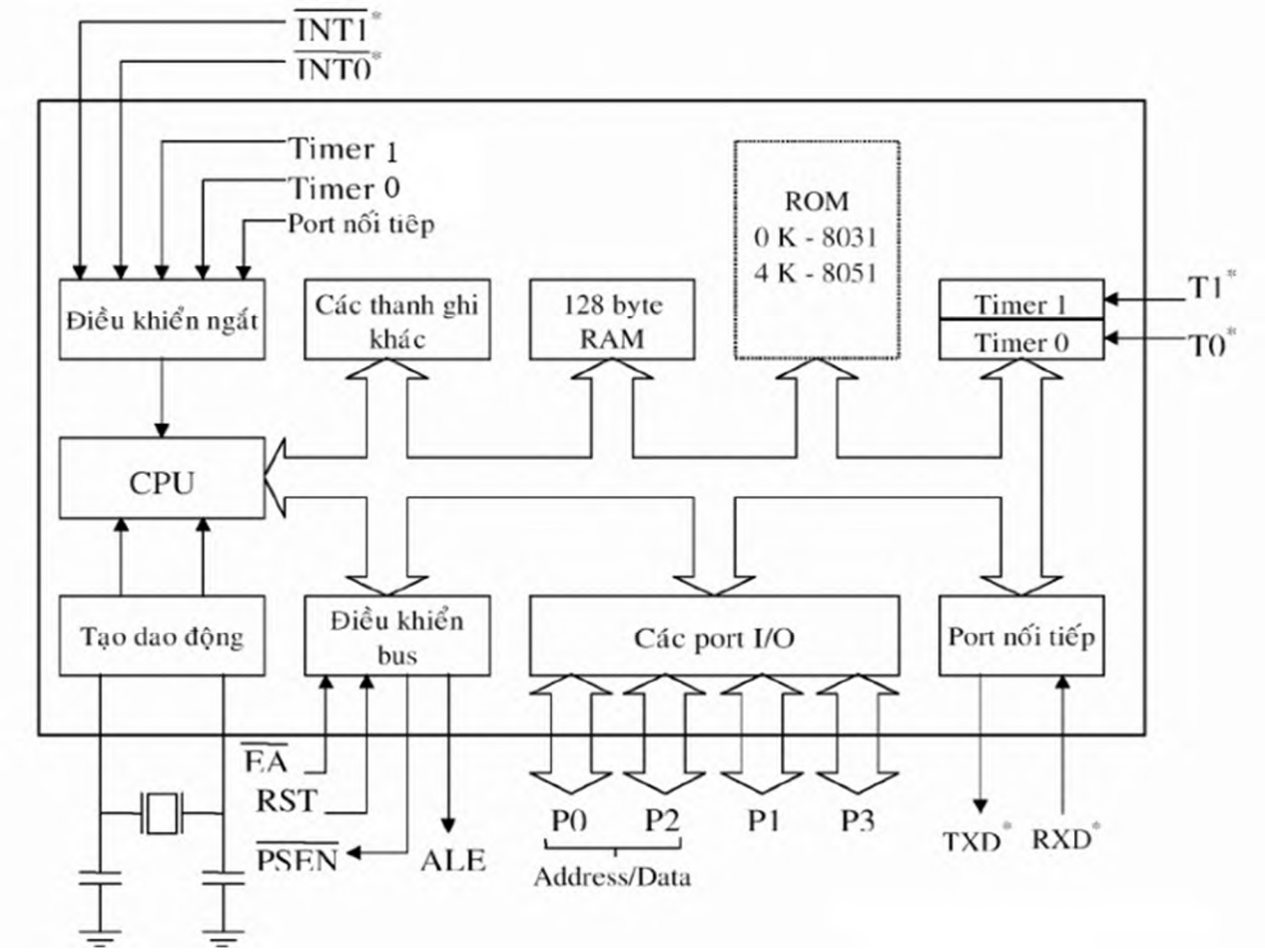
Điểm đặc biệt là các chip có chữ "T" theo sau ký hiệu "5000" có nghĩa là chip đó có thiết kế thêm một đồng hồ thời gian thực (*RTC: Real Time Clock*) bên trong. Lưu ý rằng, đồng hồ thời gian thực RTC hoàn toàn khác với bộ định thời Timer. RTC tạo và lưu giữ thời gian của ngày (*giờ/phút/giây*) và ngày tháng (*ngày/tháng/năm)* trên thực tế ngay cả khi không có nguồn cung cấp.

3.6 BỘ VI ĐIỀU KHIỂN P89V51XX CỦA PHILIPS CORPORATION

Đây là phiên bản cải tiến sử dụng CPU là bộ vi điều khiển 80C51 với nhiều tính năng vượt trội: dung lượng ROM/RAM trên chip rất lớn, 3 Timer 16 bit + 1 watch-dog Timer, 2 thanh ghi DPTR, 8 nguồn ngắt, PWM *(Pulse Width Modulator),* SPI *(Seria Periphearl Interface)* và đặc biệt là bộ nhớ chương trình trên chip có tính năng ISP *(In-System Programming)* và IAP *(In-Application Programming).*

II. SƠ ĐỒ CHÂN

3. SƠ ĐỒ KHỐI VÀ CHỨC NĂNG CÁC KHỐI CỦA CHIP 8051



Sơ đồ khối 8051

CPU *(Central Processing Unit)***:** Đơn vị xử lý trung tâm → tính toán và điều khiển quá trình hoạt động của hệ thống.

Tạo dao động: Mạch tạo dao động → tạo tín hiệu xung clock cung cấp cho các khối trong chip hoạt động.

Điều khiển ngắt: nhận tín hiệu ngắt từ bên ngoài (*INT0; INT1*), từ bộ định thời *(Timer 0; Timer 1)* và từ cổng nối tiếp *(Serial port),* lần lượt đưa các tín hiệu ngắt này đến CPU để xử lý.

Các thanh ghi khác: lưu trữ dữ liệu của các port xuất nhập, trạng thái làm việc của các khối trong chip trong suốt quá trình hoạt động của hệ thống.

RAM *(Random Accees Memory*): Bộ nhớ dữ liệu trong chip → lưu trữ các dữ liệu.

ROM *(Read Only Memory):* Bộ nhớ chương trình trong chip → lưu giữ chương trình hoạt động của chip.

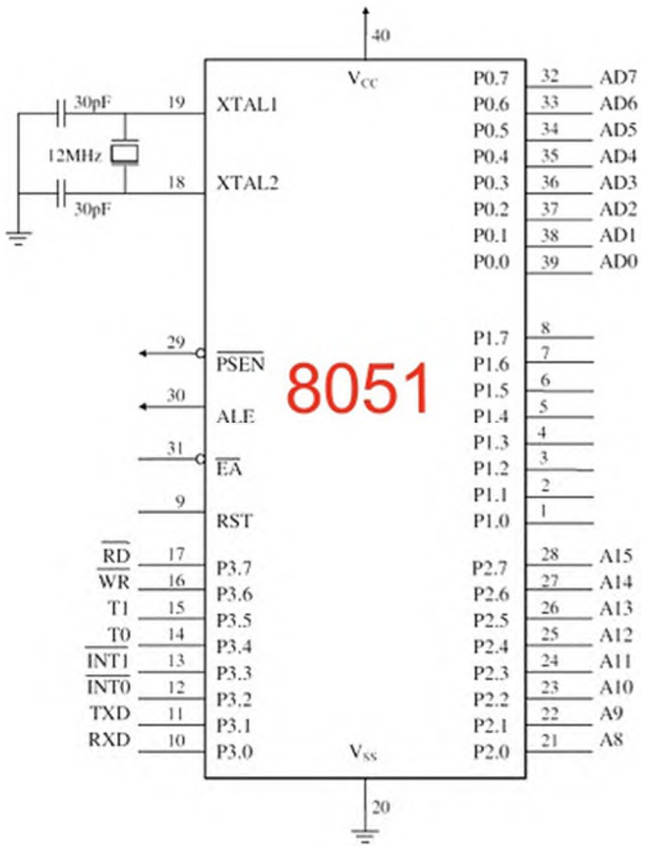
Các port I/O: Các port xuất/nhập → điều khiển việc xuất nhập dữ liệu dưới dạng song song giữa trong và ngoài chip thông qua các port P0; P1; P2; P3.

Port nối tiếp: điều khiển việc xuất nhập dữ liệu dưới dạng nối tiếp giữa trong và ngoài chip thông qua các chân TxD, RxD.

Timer 0, Timer 1: Bộ định thời 0,1 → dùng để định thời gian hoặc đếm sự kiện *(đếm xung)* thông qua các chân T0, T1.

Điều khiển bus: điều khiển hoạt động của hệ thống bus và việc di chuyển thông tin trện hệ thống bus.

4. SƠ ĐỒ CHÂN VÀ CHỨC NĂNG CÁC CHÂN CỦA CHIP 8051



Sơ đồ chân chip 8051

4.1 PORT 0

Port 0 *(P0.0 - P0.7)* có số chân từ 32 – 39.

Port có hai chức năng:

- Port xuất nhập dữ liệu *(P0.0 – P0.7)* → không sử dụng bộ nhớ ngoài.

- Bus địa chỉ byte thấp và bus dữ liệu đa hợp *(AD0 – AD7)* → có sử dụng bộ nhớ ngoài.

***Chú ý*:** Khi port 0 đóng vai trò là port xuất nhập dữ liệu thì phải sử dụng các điện trở kéo lên bên ngoài.

Ở chế độ mặc định (*khi reset)* thì các chân Port 0 *(P0.0 – P0.7)* được cấu hình là **port xuất** dữ liệu. Muốn các chân Port 0 làm **port nhập** dữ liệu thì cần lập trình lại, bằng cách ghi mức logic cao *(mức 1)* đến tất cả các bit của port trước khi bắt đầu nhập dữ liệu từ port.

Khi lập trình cho ROM trong chip thì Port 0 đóng vai trò là ngõ vào của dữ liệu (*D0 – D7).*

4.2 PORT 1

Port 1 *(P1.0 – P1.7*) có chân từ số 1 đến số 8.

Port 1 có một chức năng:

- Port xuất nhập dữ liệu *(P1.0 – P1.7)* → sử dụng hoặc không sử dụng bộ nhớ ngoài.

Ở chế độ mặc định *(khi reset)* thì các chân Port 1 *(P1.0 – P1.7)* được cấu hình là **port xuất** dữ liệu. Muốn các chân Port 1 làm **port nhập** dữ liệu thì cần phải lập trình lại, bằng cách ghi các mức logic cao (*mức 1)* đến tất cả các bit của port trước khi bắt đầu nhập dữ liệu từ port.

Khi lập trình cho ROM trong chip thì Port 1 đóng vai trò là ngõ vào của địa chỉ byte thấp (*A0 – A7).*

4.3 PORT 2

Port 2 *(P2.0 – P2.7)* có chân từ số 21 -28.

Port 2 có hai chức năng:

- Port xuất nhập dữ liệu *(P2.0 – P2.7)* → không sử dụng bộ nhớ ngoài.
- Bus địa chỉ byte cao *(A8 – A15)* → có sử dụng bộ nhớ ngoài.

Ở chế độ mặc định *(khi reset)* thì các chân Port 2 *(P2.0 – P2.7)* được cấu hình là **port xuất** dữ liệu. Muốn các chân Port 2 làm **port nhập** dữ liệu thì cần phải lập trình lại, bằng cách ghi các mức logic cao (*mức 1)* đến tất cả các bit của port trước khi bắt đầu nhập dữ liệu từ port.

Khi lập trình cho ROM trong chip thì Port 2 đóng vai trò là ngõ vào của địa chỉ byte cao (*A8 – A11)* và các tín hiệu điều khiển.

4.4 PORT 3

Port 3 *(P3.0 – P3.7)* có số chân từ 10 -17.

Port 3 có hai chức năng:

- Port xuất nhập dữ liệu *(P3.0 – P3.7).* → không sử dụng bộ nhớ ngoài hoặc các chức năng đặc biệt.
- Các tín hiệu điều khiển → có sử dụng bộ nhớ ngoài hoặc các chức năng đặc biệt.

Ở chế độ mặc định *(khi reset)* thì các chân Port 3 *(P3.0 – P3.7)* được cấu hình là **port xuất** dữ liệu. Muốn các chân Port 3 làm **port nhập** dữ liệu thì cần phải lập trình lại, bằng cách ghi các mức logic cao (*mức 1)* đến tất cả các bit của port trước khi bắt đầu nhập dữ liệu từ port.

Khi lập trình cho ROM trong chip thì Port 3 đóng vai trò là ngõ vào của các tín hiệu điều khiển.

Chức năng của các chân Port 3:

Bit	Tên	Địa chỉ bit	Chức năng
P3.0	RxD	B0H	Chân nhận dữ liệu của port nối tiếp
P3.1	TxD	B1H	Chân phát dữ liệu của port nối tiếp
P3.2	INT0	B2H	Ngõ vào ngắt ngoài 0
P3.3	INT1	B3H	Ngõ vào ngắt ngoài 1
P3.4	T0	B4H	Ngõ vào của bộ định thời/đếm 0
P3.5	T1	B5H	Ngõ vào của bộ định thời/đếm 1
P3.6	WR	B6H	Điều khiển ghi vào RAM ngoài
P3.7	RD	B7H	Điều khiển đọc từ RAM ngoài

4.5 CHÂN CHO PHÉP BỘ NHỚ CHƯƠNG TRÌNH *(PSEN)*

PSEN *(Program Store Enable)* : chân cho phép bộ nhớ chương trình, chân số 29.

Chức năng:

- Là tín hiệu cho phép truy xuất *(đọc)* bộ nhớ chương trình *(ROM)* ngoài.
- Là tín hiệu xuất, tích cực mức thấp.

PSEN = 0 → trong thời gian CPU tìm – nạp lệnh từ ROM ngoài.

PSEN = 1 → CPU chỉ sử dụng ROM trong *(không sử dụng ROM ngoài).*

Khi sử dụng bộ nhớ chương trình bên ngoài, chân PSEN thường được nối với chân OE của ROM ngoài để cho phép CPU đọc mã lệnh từ ROM ngoài.

4.6 CHÂN CHO PHÉP CHỐT ĐỊA CHỈ *(ALE)*

ALE (*Address Latch Enable):* cho phép chốt địa chỉ, chân số 30.

Chức năng:

- Là tín hiệu cho phép chốt địa chỉ để thực hiện việc giải đa hợp cho bus địa chỉ byte thấp và bus dữ liệu đa hợp *(AD0 - AD7).*
- Là tín hiệu xuất, tích cực mức cao.

ALE = 0 → trong thời gian bus AD0 – AD7 đóng vai trò là bus D0 –D7.

ALE = 1 → trong thời gian bus AD0 – AD7 đóng vai trò là bus A0 – A7.

Khi lập trình cho ROM trong chip thì chân ALE đóng vai trò là ngõ vào của xung lập trình.

***Chú ý*: $f_{ALE} = f_{OSC}/6$** → có thể dùng làm xung clock cho các mạch khác.

f_{ALE} (MHZ): tần số xung tại chân ALE.

f_{OSC} (MHZ): tần số dao động trên chip (*tần số thạch anh)*.

Khi lệnh lấy dữ liệu từ RAM ngoài được thực hiện thì một xung ALE bị bỏ qua.

4.7 CHÂN TRUY XUẤT ROOM NGOÀI (*EA)*

EA (*External Access)*: truy xuất ngoài, chân số 31.

Chức năng:

- Là tín hiệu cho phép truy xuất (*sử dụng)* bộ nhớ chương trình *(ROM)* ngoài.
- Là tín hiệu nhập, tích cực mức thấp.

EA = 0 → Chip 8051 sử dụng chương trình của ROM ngoài.

EA = 1 → Chip 8051 sử dụng chương trình của ROM trong.

Khi lập trình cho ROM trong chip thì chân EA đóng vai trò là ngõ vào của điện áp lập trình.

***Chú ý*:** chân EA phải được nối lên V_{CC} *(nếu sử dụng chương trình của ROM trong)* hoặc nối xuống GND *(nếu sử dụng chương trình của ROM ngoài),* không bao giờ được phép bỏ trống chân này.

4.8 CHÂN RESET *(RST)*

RST (*reset):* thiết lập lại, chân số 9.

Chức năng:

- Là tín hiệu cho phép thiết lập (*đặt)* lại trạng thái ban đầu cho hệ thống.
- Là tín hiệu nhập, tích cực mức cao.

RST = 0 → chip hoạt động bình thường.

RST = 1 → Chip được thiết lập lại trạng thái ban đầu.

4.9 CHÂN XTAL1, XTAL2

XTAL (*Crystal)*: tinh thể thạch anh, chân số 18, 19.

Chức năng:

- Dùng để nối với thạch anh hoặc mạch dao động tạo xung clock bên ngoài, cung cấp tín hiệu xung clock cho chip hoạt động.
- XTAL1 → ngõ vào mạch tạo xung clock trong chip.
- XTAL2 → ngõ ra mạch tạo xung clock trong chip.

4.10 CHÂN VCC VÀ GND

Chân Vcc *(chân số 40)* được nối lên nguồn 5V.

Chân GND *(chân số 20)* được nối mass.

III. CẤU TRÚC PORT I/O

Khả năng fanout (số lượng tải đầu ra) của từng chân port chip 8051 là:

- Port 0: 8 tải TTL.
- Port 1: 4 tải TTL.
- Port 2: 4 tải TTL.
- Port 3: 4 tải TTL.

Chú ý:

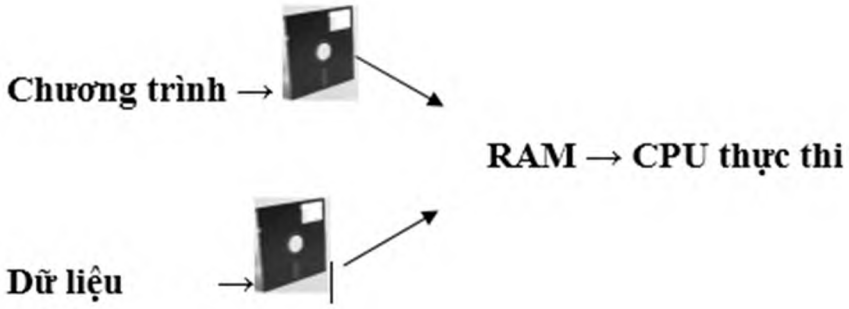
- Khi Port 0 đóng vai trò là port xuất nhập thì sẽ không có điện trở kéo lên bên trong → do đó người dùng cần thêm vào điện trở kéo lên bên ngoài.
- Ở chế độ mặc định (*khi reset)* thì tất cả các chân của các port *(P0 –P3)* được cấu hình là port xuất dữ liệu.
- Muốn các chân của chip 8051 làm port nhập dữ liệu thì ta cần phải lập trình lại bằng cách ghi mức logic cao (*mức 1)* đến tất cả các bit (*các chân)* của port trước khi bắt đầu nhập dữ liệu từ port.

- Các chân trong cùng một port không nhất thiết phải có cùng kiểu cấu hình (*port xuất hoặc port nhập).* Nghĩa là trong cùng một port có thể có chân dùng để nhập dữ liệu, có thể có chân dùng để xuất dữ liệu. Điều này tuỳ thuộc vào nhu cầu và mục đích của người lập trình.

***Chú ý*:** việc đọc dữ liệu của bất kỳ một port nào có thể cho ta hai giá trị khác nhau tuỳ thuộc vào lệnh mà ta sử dụng để đọc dữ liệu từ port. Xảy ra hiện tượng không mong muốn này là do quá trình đọc dữ liệu của chip 8051 gồm hai quá trình khác nhau: **quá trình đọc chân port và quá trình đọc bộ chốt.**

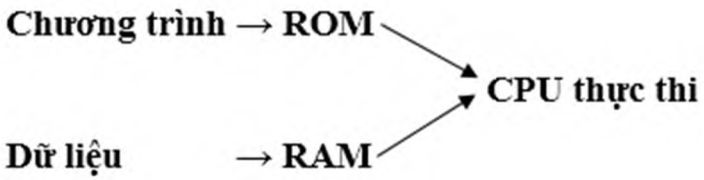
IV. TỔ CHỨC BỘ NHỚ

Bộ vi xử lý → có không gian bộ nhớ chung cho dữ liệu và chương trình.



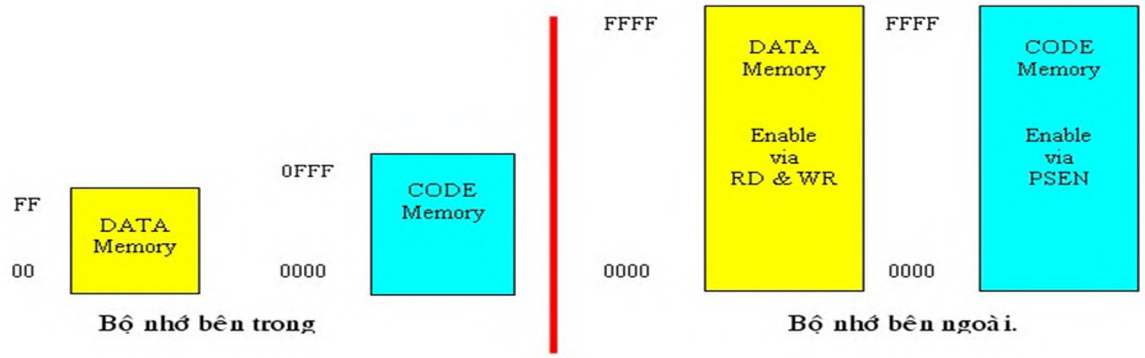
→ Chương trình và dữ liệu nằm chung trên RAM trước khi đưa vào CPU để thực thi.

Bộ vi điều khiển → có không gian bộ nhớ riêng cho dữ liệu và chương trình.

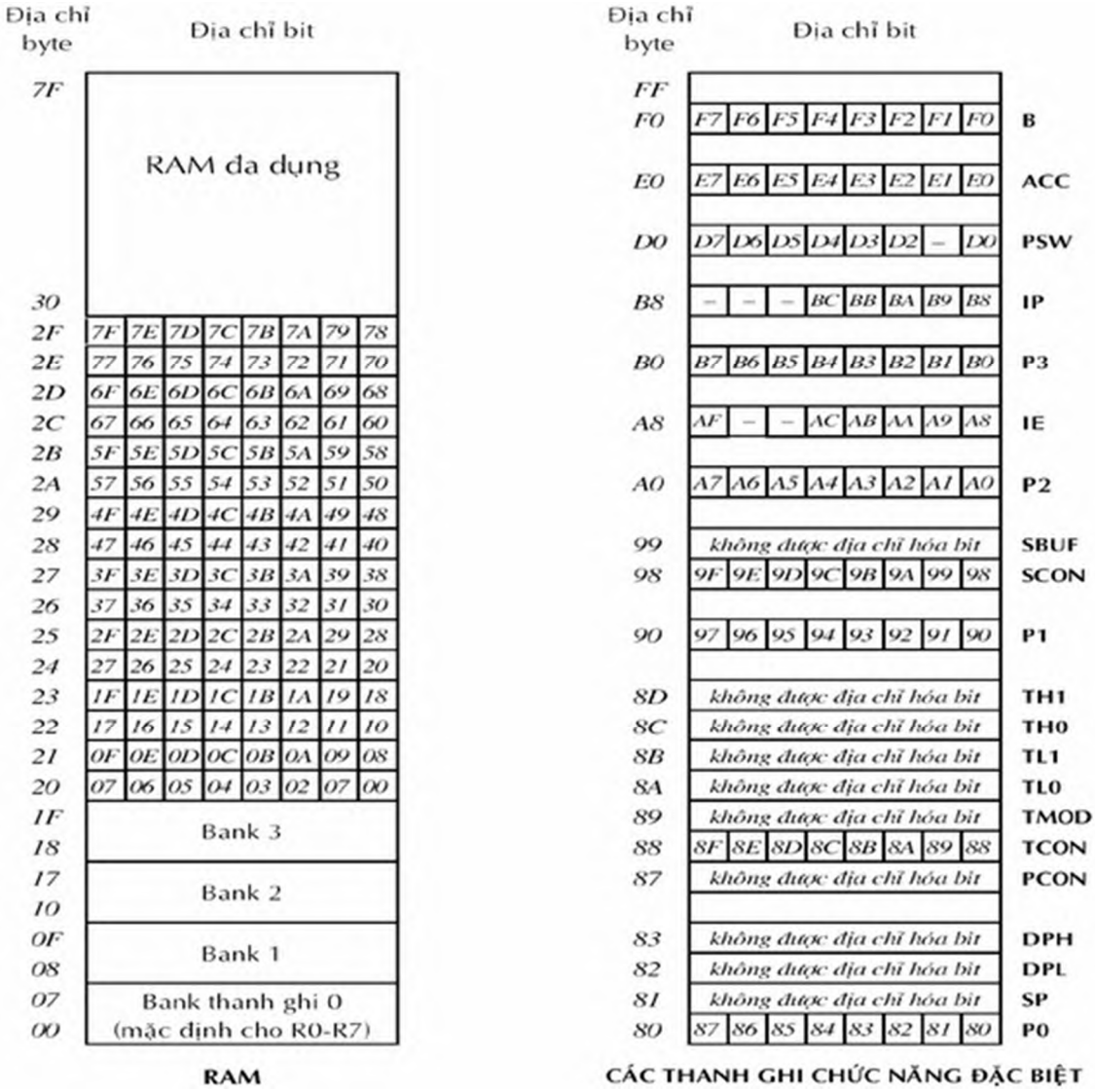


→ Chương trình và dữ liệu nằm trên ROM và RAM trước khi đưa vào CPU để thực thi.

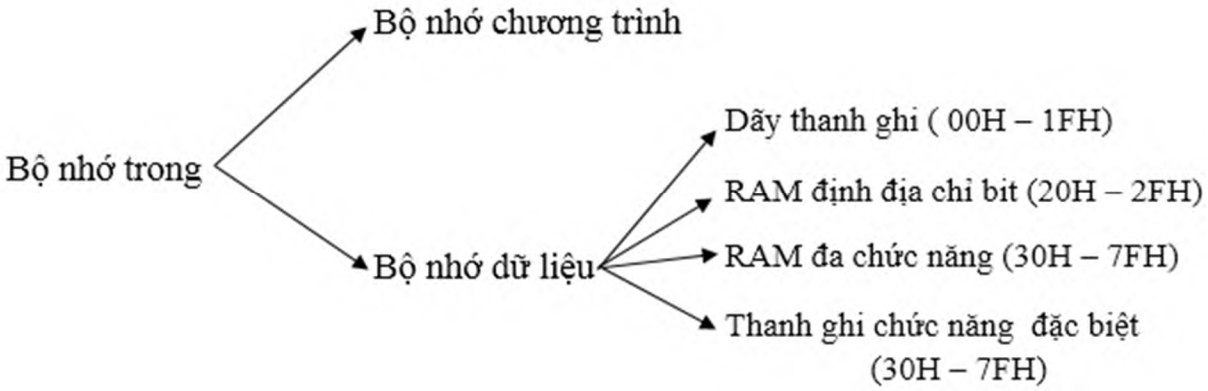
Tổ chức bộ nhớ chip 8051:



Không gian bộ nhớ chip 8051



Bộ nhớ dữ liệu trên chip 8501



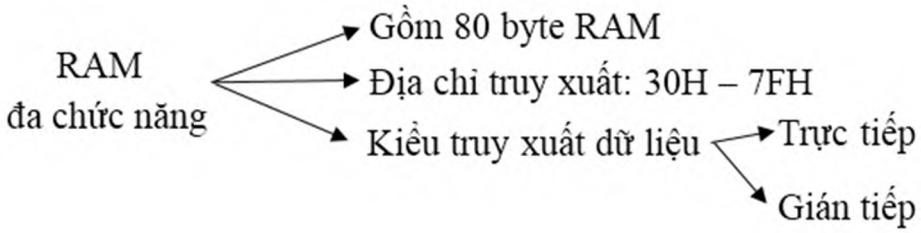
Bộ nhớ chương trình (ROM):

- Dùng để lưu trữ chương trình điều khiển cho chip 8051 hoạt động.
- Chip 8051 có 4 KB ROM trong, địa chỉ truy xuất: 000H – FFFH.

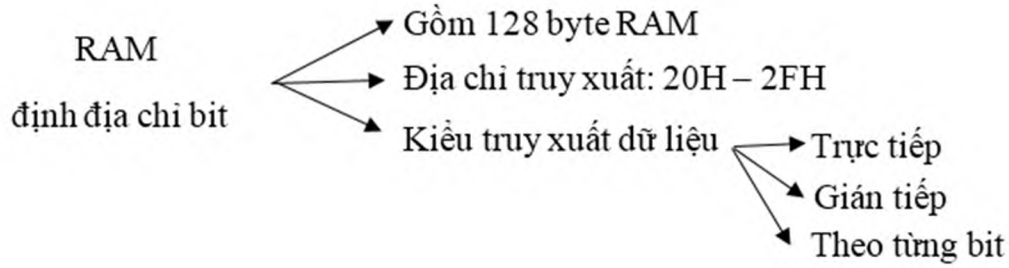
Bộ nhớ dữ liệu (RAM):

- Dùng để lưu trữ các dữ liệu và tham số.
- Chip 8051 có 128 byte RAM trong, địa chỉ truy xuất: 00H – 7FH.
- RAM trong của chip 8051 được chia ra:

RAM đa chức năng:



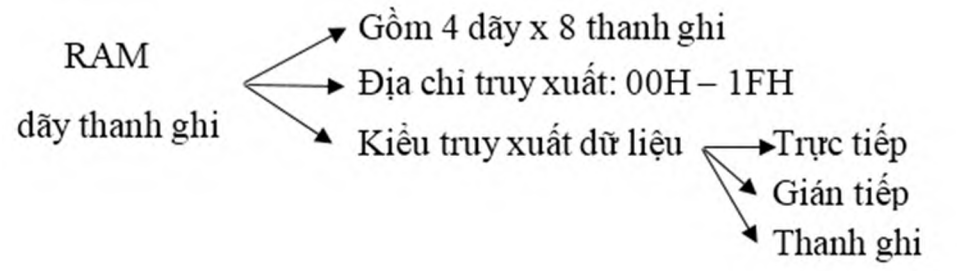
RAM định địa chỉ bit:



***Chú ý*:** Nếu trong chương trình không sử dụng các bit trong vùng RAM định địa chỉ bit này, ta có thể sử dụng vùng nhớ 20H – 2FH cho các mục đích khác. Ngược lại, ta phải viết chương trình cẩn thận khi sử dụng vùng nhớ 20H – 2FH vì nếu sơ suất ta có thể ghi dữ liệu đè lên các bit đã được sử dụng.

Các dãy thanh ghi: Cho phép truy xuất dữ liệu nhanh, lệnh truy xuất đơn

giản và ngắn gọn.



Bảng số liệu dưới đây minh hoạ địa chỉ của các ô nhớ trong một dãy và các ký hiệu thanh ghi R0 – R7 được gán cho từng ô nhớ trong dãy tích cực:

	Dãy 0	**Dãy 1**	**Dãy 2**	**Dãy 3**
R0	00H	08H	10H	18H
R1	01H	09H	11H	19H
R2	02H	0AH	12H	1AH
R3	03H	0BH	13H	1BH
R4	04H	0CH	14H	1CH
R5	05H	0DH	15H	1DH
R6	06H	OEH	16H	1EH
R7	07H	0FH	17H	1FH

Địa chỉ của các thanh ghi (R0 – R7) tương ứng với các dãy thanh ghi tích cực

Chú ý:

- Ở chế độ mặc định thì dãy thanh ghi tích cực *(đang được sử dụng)* là dãy 0 và các thanh ghi lần lượt trong dãy có tên là **R0 – R7.** Có thể thay đổi dãy tích cực bằng cách thay đổi các bit chọn dãy thanh ghi RS1 và RS0 trong thanh ghi PSW.
- Nếu chương trình của ta chỉ sử dụng dãy thanh ghi đầu tiên (*dãy 0)* thì ta có thể sử dụng vùng nhớ 08H – 1FH cho các mục đích khác. Nhưng nếu trong chương trình có sử dụng các dãy thanh ghi (*dãy 1,2 hoặc 3)*

thì phải cẩn thận khi sử dụng vùng nhớ từ 1FH trở xuống vì nếu sơ suất ta có thể ghi dữ liệu đè lên các thanh ghi R0 –R7.

V. CÁC THANH GHI CHỨC NĂNG ĐẶC BIỆT

11.THANH GHI TỪ PSW

BIT	SYMBOL	ADDRESS	DESCRIPTION
PSW.7	CY	D7H	Cary Flag
PSW.6	AC	D6H	Auxiliary Cary Flag
PSW.5	F0	D5H	Flag 0
PSW4	RS1	D4H	Register Bank Select 1
PSW.3	RS0	D3H	Register Bank Select 0
			00=Bank 0; address 00H÷07H
			01=Bank 1; address 08H÷0FH
			10=Bank 2; address 10H÷17H
			11=Bank 3; address 18H÷1FH
PSW.2	OV	D2H	Overlow Flag
PSW.1	-	D1H	Reserved
PSW.0	P	DOH	Even Parity Flag

Bảng thể hiện chức năng các bit thanh ghi PSW

PSW: Program Status Word → từ trạng thái chương trình.

Địa chỉ byte: D0H.

Địa chỉ bit: D0H –D7H.

Công dụng: cho biết trạng thái làm việc của CPU, mỗi bit của PSW có một chức năng *(thể hiện tại bảng chức năng các bit).*

11.1 Cờ CY (Cary Flag)

Là cờ nhớ → báo có nhớ/mượn tại bit 7.

- CY = 0: nếu không có nhớ từ bit 7 hoặc không có mượn cho bit 7.
- CY = 1: nếu có nhớ từ bit 7 hoặc có mượn cho bit 7.

Cờ AC (Auxiliary Carry)

Là cờ nhớ phụ → báo có nhớ/mượn tại bit 3.

- AC = 0: nếu không có nhớ từ bit 3 hoặc không có mượn cho bit 3.
- AC = 1: nếu có nhớ từ bit 3 hoặc có mượn cho bit 3.

11.2 CỜ F0 (Flag 0*)*

Là cờ zero → có nhiều mục đích dành cho các ứng dụng khác nhau của người lập trình.

11.3 BIT RS0, RS1 (Register Select*)*

Là bit chọn dãy thanh ghi → cho phép xác định dãy thanh ghi tích cực *(hay dãy thanh ghi mà các thanh ghi có tên là R0 – R7).*

RS 1	RS 0	DÃY THANH GHI	R0 – R7
0	0	Dãy 0	00H – 07H
0	1	Dãy 1	08H -0FH
1	0	Dãy 2	10H -17H
1	1	Dãy 3	18H – 1FH

11.4 CỜ OV (*Overflow)*

Là cờ tràn → báo kết quả tính toán của phép toán số học (*phép toán có dấu)* có nằm trong khoảng từ -128 đến +127 hay không.

- OV = 0: nếu -128 ≤ kết quả ≤ +127.
- OV = 1: nếu kết quả < -128 hoặc kết quả > +127. Nói cách khác: đối với **phép cộng** thì OV = 1 nếu có nhớ từ bit 7 nhưng không có nhớ từ bit 6 hoặc nếu có nhớ từ bit 6 nhưng không có nhớ từ bit 7. Đối với **phép trừ** thì OV =1 nếu có mượn cho bit 7 nhưng không có mượn cho bit 6 hoặc nếu có mượn bit 6 nhưng không có mượn bit 7.

11.5 CỜ P (*Parity)*

Là cờ chẵn lẻ → báo số chữ số 1 trong thanh ghi A là số chẵn hay số lẻ (*chip 8051 sử dụng chế độ parity chẵn).*

- P = 0: nếu số chữ số 1 trong thanh ghi A là số chẵn (*parity chẵn).*
- P = 1: nếu số chữ số 1 trong thanh ghi A là số lẻ (*parity chẵn).*

12.THANH GHI A

Địa chỉ byte: E0H.

Địa chỉ bit: E0H – E7H.

Công dụng: chứa dữ liệu của các phép toán mà vi điều khiển xử lý.

13.THANH GHI B

Địa chỉ byte: F0H.

Địa chỉ bit: F0H – F7H.

Công dụng: sử dụng kết hợp với thanh ghi A trong phép nhân và chia hai số 8 bit.

Phép nhân 2 số 8 bit không dấu → kết quả là số 16 bit.

- Byte cao → chứa vào thanh ghi B.
- Byte thấp → chứa vào thanh ghi A.

Phép chia 2 số 8 bit → thương số và số dư là số 8 bit.

- Thương số → chứa vào thanh ghi A.
- Số dư → chứa vào thanh ghi B.

14.CON TRỎ STACK (Stack Pointer)

Địa chỉ byte: 81H.

Địa chỉ bit: không định địa chỉ bit.

Công dụng: chứa địa chỉ của dữ liệu hiện đang ở đỉnh ngăn xếp.

- Ngăn xếp là vùng nhớ dùng để lưu trữ tạm thời các dữ liệu.
- Đối với chip 8051 thì vùng nhớ được dùng để làm ngăn xếp được giữ trong RAM nội.
- Để sử dụng ngăn xếp thì ta phải khởi động thanh ghi SP (*nghĩa là nạp giá trị cho thanh ghi SP)* → vùng nhớ của ngăn xếp *có địa chỉ bắt đầu*: **(SP) +1** và *địa chỉ kết thúc*: **7FH.**
- Nếu không khởi động SP → vùng nhớ của ngăn xếp *có địa chỉ bắt đầu*: **08H** *và địa chỉ kết thúc:* **7FH** (*chế độ mặc định).*

***Chú ý*:** Trong trường hợp không khởi động SP (*chế độ mặc định)* thì dãy thanh ghi 1 (*và có thể là dãy 2 và dãy 3)* sẽ không còn hợp lệ vì khi đó vùng nhớ này đã được sử dụng để làm ngăn xếp. Điều này có nghĩa là nếu ta sử dụng các dãy

thanh ghi này và lưu trữ dữ liệu vào đó thì có khả năng sẽ bị mất do tác động cất dữ liệu vào ngăn xếp của các lệnh.

15.CON TRỎ DỮ LIỆU DPTR (Data Pointer Register)

Đây là thanh con trỏ dữ liệu.

Địa chỉ byte: 83H và 82H.

Địa chỉ bit: không định địa chỉ bit.

Công dụng: là thanh ghi 16 bit *(DPH + DPL),* chứa địa chỉ của ô nhớ cần truy xuất thuộc ROM *(trong/ngoài)* và RAM ngoài.

16.THANH GHI PORT XUẤT NHẬP

Port: cổng xuất nhập.

Địa chỉ byte: 80H *(P0),* 90H *(P1),* A0H *(P2),* B0H *(P3).*

Địa chỉ bit: 80H – 87H *(P0),* 90H -97H *(P1),* A0H – A7H *(P2),* B0H – B7H *(P3).*

Công dụng: chứa dữ liệu hiện tại trên các port xuất nhập.

Chú ý:

- Trong trường hợp phần cứng có sử dụng ROM hoặc RAM bên ngoài thì ta không thể sử dụng Port 0 và Port 2 để xuất dữ liệu. Vì khi đó chip 8051 sẽ sử dụng hai port này để xác định địa chỉ và dữ liệu cho bộ nhớ ngoài. Khi đó, ta chỉ có thể sử dụng Port 1 và Port 3 để xuất nhập dữ liệu.
- Ở chế độ mặc định (*khi reset)* thì tất cả các chân của port (P0 – P3) được cấu hình là **port xuất** dữ liệu. Muốn các chân port của chip 8051 làm **port nhập** dữ liệu thì ta cần lập trình lại, bằng cách ghi mức logic cao (*mức 1)* đến tất cả các bit (*các chân)* của port trước khi bắt đầu nhập dữ liệu từ port.

17.CÁC THANH GHI ĐỊNH THỜI

7.1 THANH GHI TIMER 0

Timer 0: bộ định thời 0.

Địa chỉ byte: 8CH và 8AH.

Địa chỉ bit: không định địa chỉ bit.

Công dung: là thanh ghi bộ định thời 16 bit *(TH0 + TL0)*, chứa giá trị hiện tại của bộ định thời 0.

7.2 THANH GHI TIMER 1

Timer 1: bộ định thời 1.

Địa chỉ byte: 8DH và 8BH.

Địa chỉ bit: không định thời địa chỉ bit.

Công dụng: là thành ghi bộ định thời 16 bit *(TH1 + TL1)*, chứa giá trị hiện tại của bộ định thời 1.

7.3 THANH GHI TMOD

Timer Mode: Chế độ bộ định thời.

Địa chỉ byte: 89H.

Địa chỉ bit: không định địa chỉ bit.

Công dụng: qui định chế độ hoạt động cho cả hai bộ định thời T0 và T1.

7.4 THANH GHI TCON

Timer Control: điều khiển bộ định thời.

Địa chỉ byte: 88H.

Địa chỉ bit: 88H – 8FH.

Công dụng: báo trạng thái và điều khiển quá trình hoạt động của hai bộ định thời T0 và T1.

18.CÁC THANH GHI PORT NỐI TIẾP

8.1 THANH GHI SBUF

Serial Buffer: bộ đệm port nối tiếp.

Địa chỉ byte: 99H.

Địa chỉ bit: không định địa chỉ bit.

Công dụng: chứa dữ liệu cần truyền ra bên ngoài hay dữ liệu nhận được từ bên ngoài dưới dạng nối tiếp.

8.2 THANH GHI SCON

Serial Control: điều khiển port nối tiếp.

Địa chỉ byte: 98H.

Địa chỉ bit: 98H – 9FH.

Công dụng: báo trạng thái và điều khiển quá trình hoạt động của port nối tiếp.

19.CÁC THANH GHI NGẮT

9.1 THANH GHI IE

Interrupt Enable: cho phép ngắt.

Địa chỉ byte: A8H.

Địa chỉ bit: A8H – AFH.

Công dụng: cho phép hoặc không cho phép các ngắt hoạt động *(từng ngắt riêng lẻ hoặc tất cả các ngắt).*

9.2 THANH GHI IP

Interrupt Priority: ưu tiên ngắt.

Địa chỉ byte: B8H.

Địa chỉ bit: B8H – BCH.

Công dụng: thiết lập mức ưu tiên cho các ngắt *(ưu tiên thấp hoặc ưu tiên cao).*

20.THANH GHI ĐIỀU KHIỂN NGUỒN (Thanh ghi PCON).

Power Control: điều khiển nguồn.

Địa chỉ byte: 87H.

Địa chỉ bit: không định địa chỉ bit.

Công dụng: điều khiển tốc độ baud của port nối tiếp, điều khiển chế độ nguồn giảm và chế độ nghỉ của chip.

- Bit SMOD *(Serial Mode)* → cho phép tăng gấp đôi tốc độ truyền dữ liệu nối tiếp (*tốc độ baud)* khi SMOD =1.
- Bit GF1, GF0 (*General Function)* → cho phép người lập trình dùng với mục đích riêng.
- Bit PD (*Power Down)* → dùng để qui định chế độ nguồn giảm.

- Bit IDL (*Idle)* → dùng để qui định chế độ nghỉ.

Ở chế độ nguồn giảm *(PD = 1):*

- ✓ Mạch dao động trên chip ngừng hoạt động.
- ✓ Mọi chức năng ngừng hoạt động.
- ✓ Nội dung của RAM trên chip được duy trì.
- ✓ Các chân port duy trì mức logic của chúng.
- ✓ ALE = 0, PSEN = 0, V_{cc} = 2V.
- ✓ Thoát khỏi hệ thống bằng cách reset hệ thống.

***Chú ý*:** Hệ thống phải phục hồi V_{cc} = 5V trước khi thoát khỏi chế độ nguồn giảm.

Ở chế độ nghỉ (*IDL = 1):*

- ✓ Mạch dao động trên chip ngừng hoạt động
- ✓ Các chức năng ngắt, định thời và port nối tiếp còn hoạt động.
- ✓ Nội dung của RAM trên chip được duy trì.
- ✓ Các chân port duy trì mức logic của chúng.
- ✓ ALE =1, PSEN = 1, V_{cc} = 5V.
- ✓ Thoát khỏi chế độ bằng cách reset hoặc bằng cách cho phép ngắt.

VI. BỘ NHỚ NGOÀI

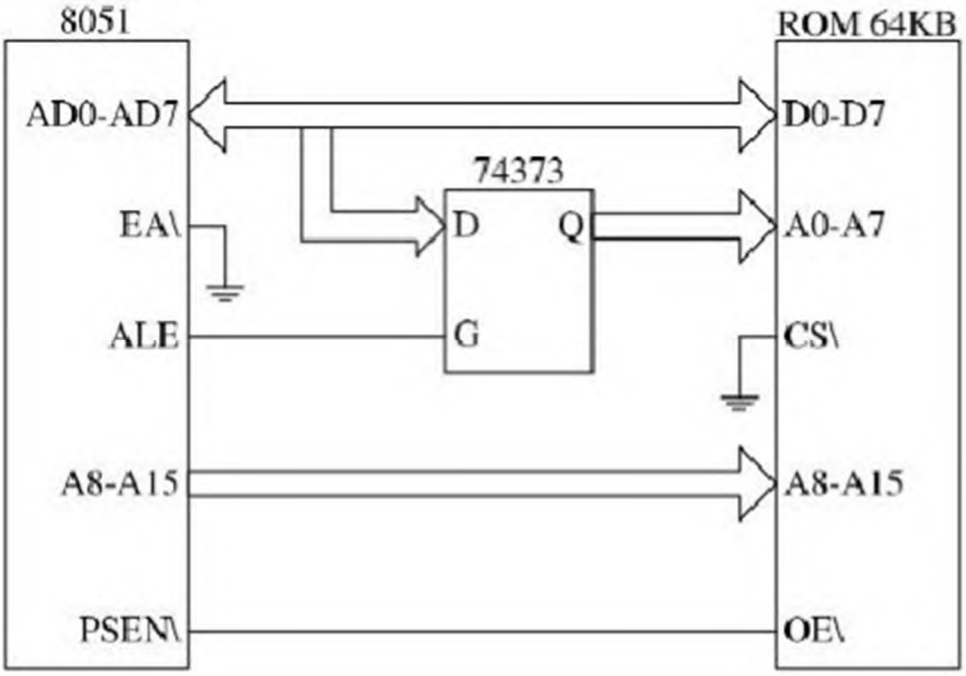
Chip 8051 cho ta khả năng mở rộng:

- Không gian bộ nhớ chương trình lên đến 64 KB.
- Không gian bộ nhớ dữ liệu lên đến 64 KB.

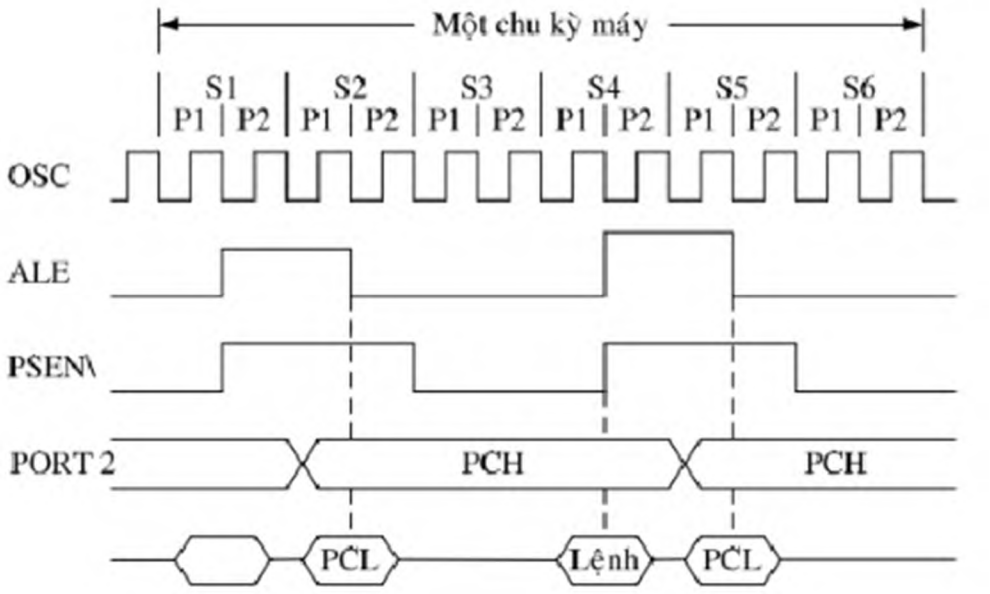
Khi sử dụng bộ nhớ ngoài:

- Port 0 → bus địa chỉ byte thấp và bus dữ liệu đa hợp (*AD0 – AD7).*
- Port 2 → bus địa chỉ byte cao (*A8 – A15).*
- Port 3 → các tín hiệu điều khiển (*WR, RD).*

4. KẾT NỐI VÀ TRUY XUẤT BỘ NHỚ CHƯƠNG TRÌNH NGOÀI

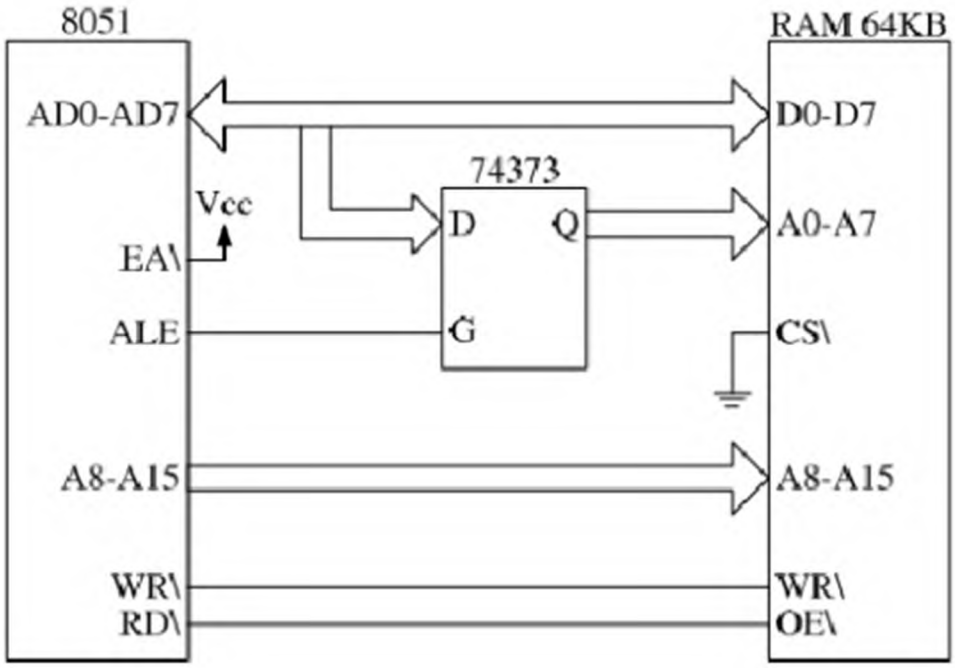


Truy xuất bộ nhớ chương trình ngoài



Giản đồ thời gian của chu kỳ tìm nạp lệnh ở bộ nhớ ngoài

5. KẾT NỐI VÀ TRUY XUẤT BỘ NHỚ DỮ LIỆU NGOÀI



Truy xuất bộ nhớ dữ liệu ngoài

6. GIẢI MÃ ĐỊA CHỈ

Trường hợp RAM và ROM được kết hợp từ nhiều bộ nhớ có dung lượng nhỏ hoặc cả hai giao tiếp với chip 8051 thì ta cần phải giải mã địa chỉ. Việc giải mã địa chỉ này cũng cần cho hầu hết các bộ vi xử lý.

Ví dụ: nếu các ROM và RAM có dung lượng 8 KB được sử dụng thì tầm địa chỉ mà chip 8051 quản lý (*0000H – FFFFH)* cần phải được giải mã thành từng đoạn 8KB để chip có thể chọn từng IC nhớ trên các giới hạn 8KB tương ứng: IC1: 0000H – 1FFFH; IC2: 2000H – 3FFFH…

IC chuyên dùng cho việc tạo tín hiệu giải mã là 74HC138, các ngõ ra của IC này lần lượt được nối với các ngõ vào chọn chip CS tương ứng của các IC nhớ để cho phép IC nhớ hoạt động (*tại một thời điểm chỉ có 1 IC nhớ được phép hoạt động).* Cần lưu ý là do các đường cho phép IC nhớ hoạt động riêng lẻ cho từng loại (*PSEN cho bộ nhớ chương trình, RD và WR cho bộ nhớ dữ liệu)* nên 8051 có thể quản lý không gian nhớ lên đến 64KB cho ROM và 64 KB cho RAM.

VII. HOẠT ĐỘNG RESET

Chân RST = 0 → chip 8051 hoạt động bình thường.

Chân RST bằng 1 → chip 8051 được reset.

Chú ý:

- Để hoàn tất quá trình reset thì chân RST phải ở mức cao tối thiểu là 2 chu kỳ máy và sau đó chuyển xuống mức thấp.
- Nội dung của RAM trong chip không bị ảnh hưởng bởi hoạt động reset.
- Sau khi reset, việc thực thi chương trình luôn bắt đầu ở vị trí đầu tiên trong bộ nhớ chương trình: Địa chỉ 0000H.
- Trạng thái của các thanh ghi sau khi reset hệ thống:
  - ✓ Bộ đếm chương trình (*PC)* 0000H
  - ✓ Thanh ghi A 00H
  - ✓ Thanh ghi B 00H
  - ✓ Thanh ghi PSW 00H
  - ✓ Thanh ghi SP 07H
  - ✓ Thanh ghi DPTR 0000H

BÀI 2: NGÔN NGỮ LẬP TRÌNH C
Mã bài: MĐ23-03

Giới thiệu:

Với kiến thức về cấu trúc bên trong của vi điều khiển và để điều khiển hoạt động khối mạch điện giao tiếp phức tạp của hệ thống. Các khối này bao gồm bộ nhớ để chứa dữ liệu và chương trình thực hiện, các mạch điện giao tiếp ngoại vi để xuất nhập và điều khiển trở lại, các khối này cùng liên kết với vi điều khiển thì mới thực hiện được công việc. Để kết nối các khối này đòi hỏi người thiết kế phải hiểu biết tinh tường về các thành phần vi điều khiển, bộ nhớ, các thiết bị ngoại vi. Sau khi kết nối vi điều khiển với các thiết bị ngoại vi, ta phải tìm hiểu về cách lập trình để điều khiển hoạt động hệ thống.

Chương này giới thiệu cách thức lập trình trên Keil'C cũng như giải thích hoạt động của các lệnh sử dụng cho họ MCS-51

Mục tiêu:

- Trình bày được sự cần thiết và cơ chế hoạt động của ngôn ngữ C theo nội dung đã học.
- Trình bày được cấu trúc chung của ngôn ngữ C theo nội dung đã học.

- Thực hiện viiết chương trình tổ chức lớn bằng cách phân chia thành các mô đun chương trình đúng qui trình kỹ thuật.
- Viết được chương trình điều khiển theo yêu cầu.

Nội dung chính:

I. MỞ ĐẦU

Ở chương này chúng ta sẽ khảo sát tập lệnh của 8051.

Vi điều khiển hay vi xử lý là các IC lập trình, khi bạn đã thiết kế hệ thống điều khiển có sử dụng vi xử lý hay vi điều khiển, ví dụ như hệ thống điều khiển đèn giao thông cho một ngã tư gồm các đèn Xanh, Vàng, Đỏ và các led 7 đoạn để hiển thị thời gian thì đó mới chỉ là phần cứng, muốn hệ thống vận hành thì bạn phải viết một chương trình điều khiển nạp vào bộ nhớ nội bên trong vi điều khiển hoặc bộ nhớ bên ngoài và gắn vào trong hệ thống để hệ thống vận hành. Dĩ nhiên, bạn phải viết đúng thì hệ thống mới vận hành đúng. Chương trình bạn viết gọi là phần mềm.

Phần mềm và phần cứng có quan hệ với nhau, người lập trình phải hiểu rõ hoạt động của phần cứng để viết chương trình. Ở phần này sẽ trình bày chi tiết về tập lệnh của vi điều khiển giúp bạn hiểu rõ từng lệnh để có thể lập trình.

Chương trình là một tập lệnh được tổ chức theo một trình tự hợp lý để giải quyết các yêu cầu của người lập trình.

Người lập trình là người biết giải thuật để viết chương trình và sắp xếp đúng các lệnh theo giải thuật. Người lập trình phải biết chức năng tất cả các lệnh của vi điều khiển để viết chương trình

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

Trong chương này chúng ta sẽ tìm hiểu các lệnh cơ bản để có thể lập trình của ngôn ngữ C. Ngôn ngữ lập trình C là một ngôn ngữ lập trình được sử dụng phổ biến, là ngôn ngữ tạo mã hiệu quả, các phần tử lập trình có cấu trúc, và một tập hợp phong phú các toán tử.

Ngôn ngữ C là một ngôn ngữ lập trình thuận tiện và hiệu quả, nhiều ứng dụng có thể được giải quyết dễ dàng hơn và hiệu quả hơn bằng ngôn ngữ C so với các ngôn ngữ chuyên biệt khác.

II. CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ C

1. CÁC KIỂU DỮ LIỆU CỦA BIẾN

Trong chương trình thường khai báo biến để lưu dữ liệu và xử lý dữ liệu, tuỳ thuộc vào loại dữ liệu mà ta phải chọn loại dữ liệu cho phù hợp. Các biến của vi xử lý bao gồm bit, byte, word và long word tương ứng với dữ liệu 1 bit, 8 bit, 16 bit và 32 bit. Các kiểu dữ liệu cơ bản như sau:

STT	Kiểu dữ liệu	Số bit	Giới hạn
1	bit	1	0÷1
2	signed char	8	-128÷127
3	unsigned char	8	0÷255
4	signed int	16	-32768÷32767
5	unsigned int	16	0÷65535
6	signed long	32	-2147483648 to 2147483647
7	unsigned long	32	0 to 4294967295
8	float	32	±1.175494E-38 to ±3.402823E+38
9	pointer	24/16/8	
10	sbit	1	0÷1
11	sfr	8	0÷255
12	sfr16	16	0÷65535

Ví dụ: *Khai báo các biến*

bit TT; //khai báo biến trạng thái thuộc kiểu dữ liệu bit.

unsigned char dem; //khai báo biến đếm *(dem)* thuộc kiểu kí tự không dấu 8 bit.

2. CÁC TOÁN TỬ

Các toán tử là thành phần quan trọng trong lập trình, để lập trình thì chúng ta cần phải hiểu rõ chức năng của các loại toán tử.

Các toán tử trong ngôn ngữ C bao gồm:

STT	Toán tử	Chức năng	Ví dụ
1	+	Toán tử cộng	
2	+=	Toán tử cộng và gán	x+=y tương đương với x=x+y
3	&=	Toán tử and và gán	x&=y tương đương với x=x&y
4	&	Toán tử địa chỉ	
5	&	Toán tử and	
6	^=	Toán tử ex-or và gán	x^=y tương đương với x=x^y
7	^	Toán tử ex-or	
8	|=	Toán tử or và gán	x|=y tương đương với x=x|y
9	|	Toán tử or nhiều đại lượng với nhau thành 1	Ví dụ: or nhiều bit trong 1 byte với nhau.
10	--	Giảm	
11	/=	Toán tử chia và gán	x/=y tương đương với x=x/y
12	/	Toán tử chia	
13	==	Toán tử bằng dùng để so sánh	
14	>	Toán tử lớn hơn	
15	>=	Toán tử lớn hơn hay bằng	
16	++	Tăng	

17	*	Toán tử truy xuất gián tiếp, đi trước con trỏ	
18	!=	Toán tử không bằng	
19	<<=	Toán tử dịch trái và gán	x<<=y tương đương với x=x<<y
20	<	Toán tử nhỏ hơn	
21	<<	Toán tử dịch trái	
22	<=	Toán tử nhỏ hơn hay bằng	
23	&&	Toán tử and	
24	!	Toán tử phủ định (not)	
25	||	Toán tử or	
26	%=	Toán tử chia lấy số dư và gán	x%=y tương đương với x=%y
27	%	Toán tử module	
28	*=	Toán tử nhân và gán	x*=y tương đương với x=x*y
29	*	Toán tử nhân	
30	~	Toán tử bù 1	
31	>>=	Toán tử dịch phải và gán	x>>=y tương đương với x=x>>y
32	>>	Toán tử dịch phải	
33	->	Toán tử con trỏ cấu trúc	
34	-=	Toán tử trừ và gán	x-=y tương đương với x=x-y

35	-	Toán tử trừ	
36	Sizeof	Xác định kích thước theo byte của toán tử	

2.1 TOÁN TỬ GÁN *(=)*

Dùng để gán một giá trị nào đó cho một biến.

Ví dụ: A = 5;

→ Gán biến A bằng 5.

Ví dụ: A = 2+(B=5);

→ Gán biến B bằng 5 rồi cộng với 2 và gán cho biến A, kết quả B = 5 và A = 7.

2.2 TOÁN TỬ SỐ HỌC *(+ , - , * , /)*

Có 5 toán tử để thực hiện các phép toán cộng, trừ, nhân, chia và chia lấy dư.

Ví dụ: A = 24; B = A%5;

→ Gán A = 24, B gán số dư của A chia cho 5, kết quả B = 4.

Ví dụ:
```
A = 123;    X = A%10   //X = 3
A = A/10;              //A = 12
Y = A%10;              //Y = 2
Z = A/10;              //Z = 1
```

→ Ví dụ này hướng dẫn tách từng con số đơn vị, chục, tram gán cho 3 biến X, Y, Z.

Ví dụ:
```
A = 1234;   X=A%10    //X = 4
A = A/10;             //A = 123
Y = A%10;             //Y = 3
A = A/10;             //A = 12
Z = A%10;             //Z = 2
V = A/10;             //V = 1
```

→ Ví dụ này hướng dẫn tách từng con số đơn vị, chục, trăm, ngàn gán cho 4 biến X, Y, Z, V.

2.3 TOÁN TỬ GÁN PHỨC HỢP *(+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)*

Tổng quát cho toán tử gán phức hợp: ***biến* += *giá_trị*** tương đương ***biến* = *biến* + *giá_ trị.***

Ví dụ:
```
A+=5; tương đương với A = A+5;
A/=5;  tương đương với A = A/5;
B*=X+1  tương đương B = B*(X+1);
```

2.4 TOÁN TỬ TĂNG VÀ GIẢM *(++,--)*

Ví dụ:

A = 5; → Gán A bằng 5.

A++; → Tương đương với A = A+1 hay A+=; Kết quả A =6.

Ví dụ:

B = 3;→ Gán B bằng 3.

A=++B; → Kết quả A = 4, B =4

Ví dụ:

B = 3;→ Gán B bằng 3.

A = B++; → Kết quả A bằng B và bằng 3, B tăng lên 1 bằng 4.

→ Sự khác nhau là "++" đặt trước thì tính trước rồi mới gán, đặt sau thì gán trước rồi mới tính.

2.5 TOÁN TỬ QUAN HỆ *(==, !=,> , <, >=, <=)*

Các toán tử quan hệ dùng để so sánh các biểu thức, kết quả so sánh là đúng hoặc sai.

Các toán tử tương ứng là bằng, khác, lớn hơn, nhỏ hơn, lớn hơn hay bằng, nhỏ hơn hay bằng.

Ví dụ:

```
If(X<100)    X+=1
```

→ Lệnh trên kiểm tra nếu X còn nhỏ hơn 100 thì tăng X lên 1.

2.6 TOÁN TỬ LOGIC *(!,&&,||)*

Các toán tử trên tương đương là NOT, AND và OR.

Ví dụ:

!true sẽ cho kết quả là false.

((5==5)&&(6>4)) and hai điều kiện lại với nhau và kết quả là true.

2.7 TOÁN TỬ XỬ LÝ BIT *(&, |, ^,~, <<, >>)*

Các toán tử xử lý bit với bit, các toán tử trên tương đương là AND, OR, XOR, NOT,SHL (*dịch trái),* SHR *(dịch phải).*

Ví dụ:

```
A = 0x77;    //gán A=0111 0111B
B = 0xC9;    //gán B=1100 1001B
X = A&B;    //X bằng A and với B, kết quả X = 0100 0001B = 0x41
Y = A|B;     //Y bằng A or với B, kết quả Y = 1111 1111B = 0xFF
```

Z = A^B; //Z bằng A xor với B, kết quả Z = 1011 1110B = 0xBE

W = ~A; //W bằng not A, kết quả W = 1000 1000B = 0x88

Ví dụ:

A = 0x01; //gán A = 0000 0001B

A = (A<<1);//dịch A sang trái 1 bit, kết quả A = 0000 0010B

A = (A<<1);//dịch A sang trái 1 bit, kết quả A = 0000 0100B

→ Khi dịch trái thì bit bên trái mất, bit 0 thêm vào bên phải.

Ví dụ:

A = 0x81; //gán A = 1000 0001B

A = (A>>1);//dịch A sang phải 1 bit, kết quả A = 0100 0000B

→ Khi dịch phải thì bit bên phải mất, bit 0 thêm vào bên trái.

Ví dụ:

A = 0x00; //gán A = 0000 0000B

A = (A<<1) +0x01;//dịch A sang trái 1 bit rồi cộng với 1, kết quả A = 0000 0001B

A = (A<<1) +0x01;//dịch A sang trái 1 bit rồi cộng với 1, kết quả A = 0000 0011B

→ Khi dịch trái và cộng 1 thì có chức năng đẩy số 1 thêm vào bên phải, với dữ liệu 8 bit thì sau 8 lần sẽ lấp đầy bit 1.

Ví dụ:

A = 0x00; //gán A = 0000 0000B

A = (A>>1) +0x80; //dịch A sang phải 1 bit rồi cộng với 1000 0000, kết quả A = 1000 0000B.

A = (A>>1) +0x80; //dịch A sang phải 1 bit rồi cộng với 1000 0000, kết quả A = 1100 0000B

→ Khi dịch phải và cộng với 0x80 thì có chức năng đẩy số 1 thêm vào bên trái, với dữ liệu 8 bit thì sau 8 lần sẽ lấp đầy bit 1.

Ví dụ:

unsigned char X,Y; //X,Y là biến 8 bit

unsigned int A; //A là biến 16 bit

A = 0x1234;

X = A; //gán X=0x34 là gán byte thấp

Y = A>>8 //dịch A sang phải 8 bit rồi gán Y, kết quả Y=0x12 có nghĩa là gán byte cao.

2.8 TOÁN TỬ LẤY KÍCH THƯỚC CHUỖI THEO BYTE 0

Ví dụ:

A = sizeof (charac); //kết quả là A sẽ chứa số byte của chuỗi charac

3. CÁC LỆNH C CƠ BẢN

Thành phần quan trọng thứ 3 trong lập trình C là các lệnh của ngôn ngữ C, phần tiếp theo sẽ là khảo sát các lệnh cơ bản.

3.1 LỆNH IF VÀ ELSE

Chức năng: kiểm tra điều kiện nếu thoả mãn thì thực hiện.

Cú pháp:

```
if (điều_kiện)
        {       lệnh a1;
                lệnh a2;
                …
        }
else
        {       lệnh b1;
                lệnh b2;
                …
        }
```

Ví dụ:

```
if (x==50)
                x=1;
else
                x=x+1;
```

3.2. LỆNH LẶP WHILE

Chức năng: lặp lại một thao tác với một số lần nhất định hoặc khi còn thoả một điều kiện nào đó.

Cú pháp:

```
while(điều_kiện)
         {      lệnh 1;
                lệnh 2;
                …
          }
```

Ví dụ:

```
while (x>0)
```

{x=x-1;}

3.3 LỆNH LẶP DO WHILE

Chức năng: làm các lệnh trong dấu ngoặc và thoát điều kiện theo sau lệnh while không đúng.

Cú pháp:

```
do
    {   lệnh 1;
        lệnh 2;
        …
    }
while (điều_kiện)
```

Ví dụ:

```
do
    {x=x+10;}
While (x<100)
```

3.4 LỆNH LẶP FOR

Chức năng: làm các lệnh trong dấu ngoặc một số lần nhất định.

Cú pháp:

```
for (giá_trị_bắt_đầu;điều_kiện_kết_thúc;tăng_giá_trị)
    {   lệnh 1;
        lệnh 2;
        …
    }
```

Ví dụ:

```
for(int n = 0; n < 100; n++)
    {x=x+10;}
```

3.5 LỆNH RẼ NHÁNH BREAK

Chức năng: dùng để thoát khỏi vòng lặp cho dù điều kiện để kết thúc chưa thoả mãn.

3.6 LỆNH CONTINUE

Chức năng: dùng để kết thúc phần còn lại của vòng lặp để làm lần lặp tiếp theo.

3.7 LỆNH RẼ NHÁNH GOTO

Chức năng: dùng để nhảy đến nhãn trong chương trình.

3.8 LỆNH SWITH CASE

Chức năng: thực hiện một công việc tuỳ thuộc vào hàm.

Cú pháp:

```
switch (cmd)
    {
        case constan1:      lệnh a1;
                            lệnh a2;
                            …
                            break;
        case constant2:     lệnh b1;
                            lệnh b2;
                            …
                            break;
        default:            lệnh c1;
                            lệnh c2;
                            …
                            break;
    }
```

Ví dụ:

```
switch (cmd)
    {
        Case 0:     printf(“cmd 0”);
                    break;
        case 1:             printf(“cmd 1”);
                    break;
        default:    print(“bad cmd”);
                    break;
    }
```

4. CÁC THÀNH PHẦN CỦA CHƯƠNG TRÌNH C

4.1 Chỉ Dẫn Tiền Xử Lý

Có hai chỉ dẫn là **#define** và **#include**.

- Chỉ dẫn **#define** có chức năng thay thế 1 đoạn chuỗi bằng 1 địa chỉ đặc biệt nào đó.
- Chỉ dẫn **#include** có chức năng chèn vào một đoạn lệnh từ 1 file khác vào trong chương trình.

4.2 Khai Báo

Khai báo biến bao gồm tên và thuộc tính, khai báo hàm, khai báo hằng. Khai báo biến toàn cục nằm bên ngoài hàm, khai báo biến cục bộ thì nằm bên trong hàm.

4.3 Định Nghĩa Giá Trị Cho Biến

Dùng để thiết lập giá trị cho 1 biến hoặc 1 hàm.

4.4 Biểu Thức

Là sự kết hợp của toán tử và tác tố để cho ra một kết quả duy nhất.

4.5 Hàm

Một hàm bao gồm các khai báo biến, định nghĩa giá trị, các biểu thức và các lệnh để thực hiện 1 chức năng đặc biệt nào đó.

BÀI 3: XUẤT NHẬP PORT
Mã bài: MĐ23-04

Giới thiệu:

Để lập trình được Vi điều khiển điều cơ bản nhất là truy xuất I/O các chân của Vi điều khiển hay gọi tắt là việc xuất nhập Port.

Mỗi Port của Vi điều khiển gồm 8 chân, mỗi chân tương ứng với 1 bít trong thanh ghi điều khiển như vậy mỗi Port là 1 byte. Họ vi điều khiển 8051 có 4 Port tương ứng với 32 chân được bố trí theo sơ đồ chân, phần cứng của Vi điều khiển.

Mục tiêu:

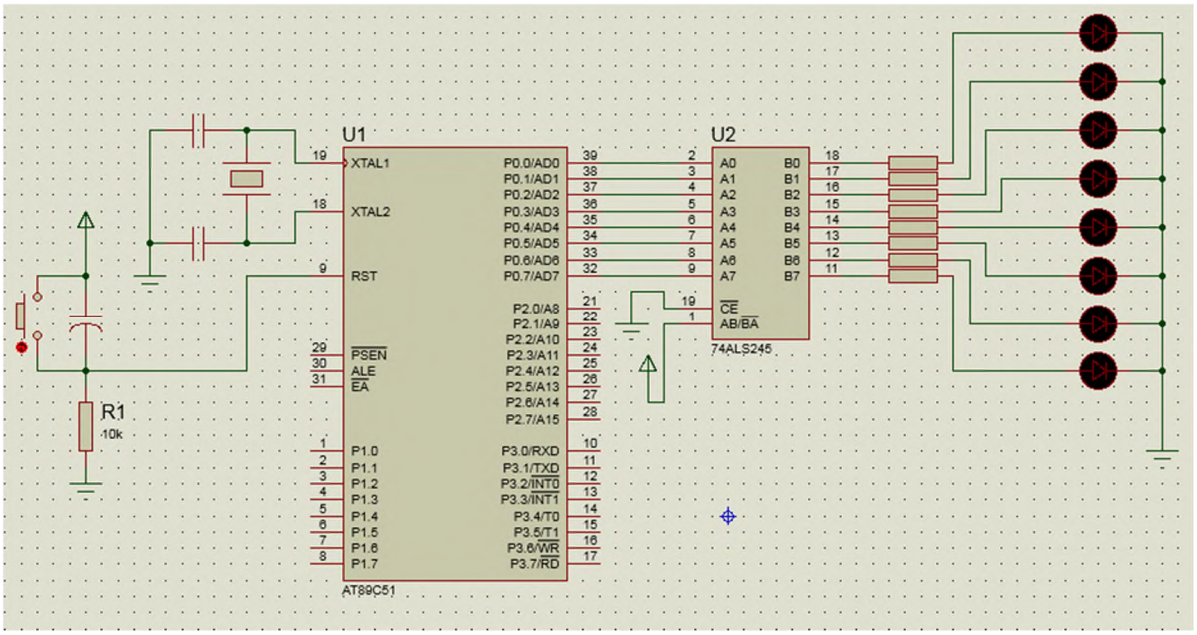
- Trình bày được sự cần thiết và cơ chế hoạt động của ngôn ngữ C theo nội dung đã học.
- Trình bày được cấu trúc chung của ngôn ngữ C theo nội dung đã học.
- Thực hiện viiết chương trình tổ chức lớn bằng cách phân chia thành các mô đun chương trình đúng qui trình kỹ thuật.
- Viết được chương trình điều khiển theo yêu cầu.

Nội dung chính:

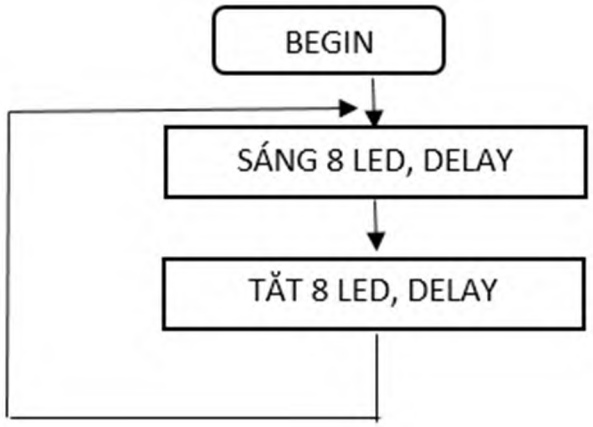
I. LED ĐƠN

Bài tập 1: Viết chương trình điều khiển 8 led đơn sáng tắt *(nhấp nháy).*

- ***Sơ đồ mạch***



- ***Lưu đồ***



- ***Chương trình***

```
#include<AT89X51.H>
void   delay(unsigned      int      x)
{   unsigned int      y;
    for(y=0;y<x;y++)      {}
}
void main    ()
{   while(1)
        {       p0      =       0x00; delay(5000);
                p0      =       0xFF; delay(5000);
```

```
    }
}
```

- ***Giải thích chương trình:***

```
#include <AT89X51.H>
```

File "AT89X51.H" lưu trong thư mục INC của phần mềm KEIL, file này chứa các tên của các thành phần vi điều khiển AT89S51. Trong file "AT89X51.H" đã định sẵn địa chỉ cho Port 0. Khi có thêm dòng lệnh này, chúng ta sẽ không phải khai báo địa chỉ cho Port 0 như ở bài tập 1 nữa, các thành phần khác cũng được đã được khai báo tương tự (*xem thêm ở phần FILE THƯ VIỆN CHO HỌ 89S51).*

***Giải thích chương trình con delay*:**

```
void delay(unsigned int x)
{ unsigned int y;
  for(y=0;y<x;y++) {}
}
```

Chương trình được đặt tên là "***delay(unsigned int x)"***, Khi gọi hàm delay này thì phải trao cho hàm này một thông số để tiến hành, thông số đó sẽ được gán cho biến x, "**int y"** là khai báo biến y kiểu số nguyên 16 bit.

Lệnh: for(y=0;y<x;y++) {}

→ Vòng lặp for thực hiện các lệnh trong dấu {} với x lần từ 0 đến x-1. Ví dụ khi gọi hàm delay(5000) thì x = 5000 và vòng for thực hiện 5000 lần tạo ra một lượng thời gian cần thiết. Muốn delay kéo dài hay ngắn thì chúng ta thay đổi thông số delay.

Giải thích chương trình chính:

```
void main ()
{ while(1)
    { p0 = 0x00; delay(5000);
      p0 = 0xFF; delay(5000);
    }
```

```
}
```

Chương trình chính bắt đầu bằng từ khoá "**void main ()".**

Trong chương trình chính có sử dụng vòng lặp **while(1)** có cú pháp như sau:

```
While(điều_kiện)
        {       lệnh 1;
                lệnh 2;
                …
        }
```

Khi "điều_kiện" là true thì các lệnh trong vòng lặp được thực hiện, sai thì không thực hiện.

Với lệnh **while(1)** có nghĩa là **điều_kiện** luôn đúng nên các lệnh trong vòng lặp được thực hiện lặp lại.

Lệnh **p0 = 0x00** là gán trị số hex 00h cho port 0 để điều khiển 8 led tắt, gọi chương trình delay để thực hiện thời gian trễ.

Lệnh **p0 = 0xFF** là gán trị số hex FFh cho port 0 để điều khiển 8 led bật, gọi chương trình delay để thực hiện thời gian trễ.

Bài 2: Viết chương trình 0 điều khiển 8 led đơn sáng tắt 5 lần rồi sáng luôn.

- ***Chương trình:***

Cách 1:

```
#include <AT89X51.H>
signed char i;
void   delay(unsigned     int     x)
{   unsigned int     y;
    for(y=0;y<x;y++)      {;}
}
void main    ()
{   for(i=0;i<5;i++)
        {       P0     =      0x00; delay(5000);
                P0     =      0xFF; delay(5000);
        }
    while(1)
```

```
        {;}
}
```

Giải thích chương trình:

```
for(i=0;i<5;i++)
```

Ta cho vòng lặp for thực hiện 5 chớp tắt 5 lần, sau đó thực hiện lệnh **while(1):** không làm gì cả, nhảy tại chỗ.

Cách 2:

```
#include <AT89X51.H>
signed char i;
void   delay(unsigned     int     x)
{      unsigned     int     y;
       for(y=0;y<x;y++)  {}
}
void main    ()
{      i=0;
       while(i<5)
              {      P0     =      0x00; delay(5000);
                     P0     =      0xFF; delay(5000);
                     i++;
              }
       while(1)
              {;}
}
```

Ở chương trình này, chúng ta cho biến "i = 0", thực hiện vòng lặp while chớp tắt với điều kiện là biến "i" nhỏ hơn 5, đến khi biến "i = 5" thì thực hiện lệnh **while(1)** để chương trình không làm gì cả, nhảy tại chỗ.

Bài 3: Viết chương trình điều khiển 8 led đơn sáng tắt dần từ phải sang trái.

- ***Chương trình*:**

```
#include <AT89X51.H>
signed char i;
void delay(unsigned int x)
{ unsigned int y;
  for(y=0;y<x;y++) {}
}
void main ()
{ P0 = 0x00; delay(5000);
  while(1)
    { for(i=0;i<8;i++)
        { P0=(P0<<1)+1; delay(5000);}
      for(i=0;i<8;i++)
        { P0=(P0<<1); delay(5000);}
    }
}
```

Giải thích chương trình:

Mở đầu chương trình chính, lệnh "**P0 = 0x00"** làm Port 0 bằng 00 để tắt 8 led, delay để nhìn thấy.

Vòng lặp for thực hiện 8 lần: mỗi lần thực hiện lệnh: "**P0=(P0<<1)+1"** sẽ dịch dữ liệu 8 bit trong port 0 sang trái 1 bit và cộng với 1.

Dữ liệu của 8 lần dịch trái:

i	Trước khi dịch	Sau khi dịch	Sau khi cộng 1
0	0000 0000	0000 0000	0000 0001
1	0000 0001	0000 0010	0000 0011
2	0000 0011	0000 0110	0000 0111

3	0000 0111	0000 1110	0000 1111
4	0000 1111	0001 1110	0001 1111
5	0001 1111	0011 1110	0011 1111
6	0011 1111	0111 1110	0111 1111
7	0111 1111	1111 1110	1111 1111

Sau mỗi lần dịch, sử dụng delay để có thể nhìn thấy.

Tương tự, để tắt 8 led, chúng ta thực hiện lệnh dịch trái 8 lần.

Bài 4: Viết chương trình điều khiển 8 led đơn sáng tắt dần từ phải sang trái, trái sang phải

- ***Chương trình***

```
#include <AT89X51.H>
signed char i;
void delay(unsigned int x)
{ unsigned int y;
  for(y=0;y<x;y++) {;}
}
void main ()
{ P0 = 0x00; delay(5000);
  while(1)
    { for(i=0;i<8;i++)
        { P0=(P0<<1)+1; delay(5000);}
      for(i=0;i<8;i++)
        { P0=(P0<<1); delay(5000);}
      for(i=0;i<8;i++)
        { P0=(P0>>1)+0x80; delay(5000);}
      for(i=0;i<8;i++)
```

```
                    {       P0=(P0>>1);             delay(5000);}
            }
}
```

Giải thích chương trình:

Lệnh xoay từ trái sang phải là "**P0=(P0>>1)+0x80",** lệnh này làm Port 0 xoay từ trái sang phải, đồng thời cộng với 0x80 để đẩy số 1 vào từ bên trái.

Dữ liệu lần lượt của 8 lần dịch phải:

i	**Trước khi dịch**	**Sau khi dịch**	**Sau khi cộng 0x80 (0x80 ↔ 1000 0000)**
0	0000 0000	0000 0000	1000 0000
1	1000 0000	0100 0000	1100 0000
2	1100 0000	0110 0000	1110 0000
3	1110 0000	0111 0000	1111 0000
4	1111 0000	0111 1000	1111 1000
5	1111 1000	0111 1100	1111 1100
6	1111 1100	0111 1110	1111 1110
7	1111 1110	0111 1111	1111 1111

Sau mỗi lần dịch, thực hiện delay để có thể nhìn thấy.

Bài 5: Viết chương trình điều khiển 16 led đơn sáng tắt dần từ phải sang trái, trái sang phải.

- ***Chương trình:***

Cách 1:

```
#include <AT89X51.H>
signed char i;
```

```
void delay(unsigned int x)
{ unsigned int y;
  for(y=0;y<x;y++) {}
}
void main ()
{ P0 = 0x00; P1 = 0x00;                delay(5000);
  while(1)
    {
      for(i=0;i<8;i++)                 //P0 sang dan
        { P0=(P0<<1)+0x01;             delay(5000);}
      for(i=0;i<8;i++)                 //P1 sang dan
        { P1=(P1<<1)+0x01;             delay(5000);}
      for(i=0;i<8;i++)                 //P0 tat dan
        { P0=(P0<<1);                  delay(5000);}
      for(i=0;i<8;i++);                //P1 tat dan
        { P1=(P1<<1);                  delay(5000);}
      for(i=0;i<8;i++)                 //P1 sang dan tsp
        { P1=(P1>>1)+0x80;             delay(5000);}
      for(i=0;i<8;i++)                 //P0 sang dan tsp
        { P0=(P0>>1)+0x80;             delay(5000);}
      for(i=0;i<8;i++)                 //P1 tat dan tsp
        { P1=(P1>>1);                  delay(5000);}
      for(i=0;i<8;i++)                 //P0 tat dan tsp
        { P0=(P0>>1);                  delay(5000);}
    }
}
```

Cách 2:

```
#include <AT89X51.H>
signed char i;
unsigned int Z;
void delay(unsigned int x)
{ unsigned int y;
  for(y=0;y<x;y++) {}
}
void XUAT2PORT()
{ P0 = Z; P1 = (Z>>8); }
void main()
{ Z = 0x0000; XUAT2PORT(); delay(5000);
  while(1)
  { for(i=0;i<16;i++)
      { Z = (Z<<1)+0x01; XUAT2PORT();
  delay(5000);}
    for(i=0;i<16;i++)
      { Z = (Z<<1); XUAT2PORT();
  delay(5000);}
    for(i=0;i<16;i++)
      { Z = (Z>>1)+0x8000; XUAT2PORT();
  delay(5000);}
    for(i=0;i<16;i++)
      { Z = (Z>>1); XUAT2PORT();
  delay(5000);}
  }
}
```

II. LED ĐƠN – GIAO TIẾP NÚT NHẤN

Nút nhấn, bàn phím để giao tiếp giữa con người và mạch điện tử để điều

khiển, ví dụ: bàn phím máy tính, bàn phím điện thoại…máy giặt tự động có bàn phím chỉnh chế độ, chọn mực nước.

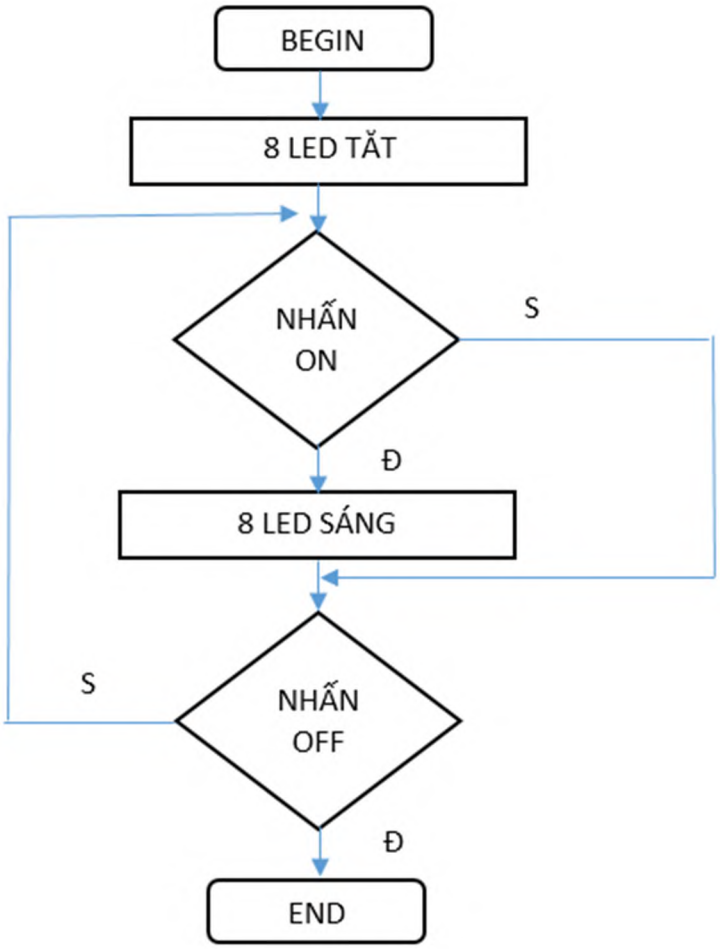
Có hai dạng giao tiếp vi điều khiển với bàn phím, nút nhấn:

- Hệ thống ít phím: điều khiển động cơ 3 pha: gồm các nút Start; Stop,Inv…
- Hệ thống nhiều phím: bàn phím máy tính, bàn phím điện thoại…

Trong giáo trình này, chúng ta chỉ tìm hiểu những bài lập trình đơn giản sử dụng hệ thống ít phím.

Bài 1: Viết chương trình điều khiển 8 led đơn bằng 2 nút nhấn. Khi cấp điện thì 8 led tắt, khi nhấn ON thì 8 led sáng, khi nhấn OFF thì 8 led tắt.

- ***Lưu đồ:***



- ***Chương trình:***

```
#include <AT89X51.H>
#define       ON    P3_0
```

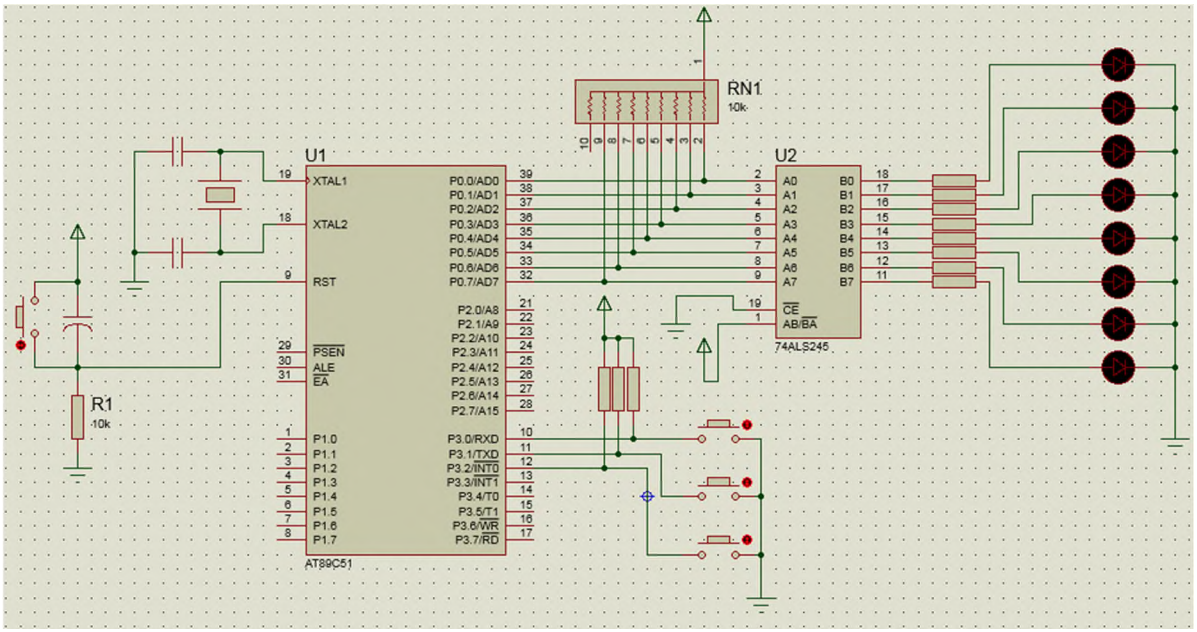
```
#define       OFF   P3_1
void main ()
{       P0 = 0x00;
        while(1)
                {       if(ON==0)   {P0    = 0xFF;}
                        if(OFF=0)   {P0    = 0x00;}
                }
}
```

- ***Giải thích chương trình:***

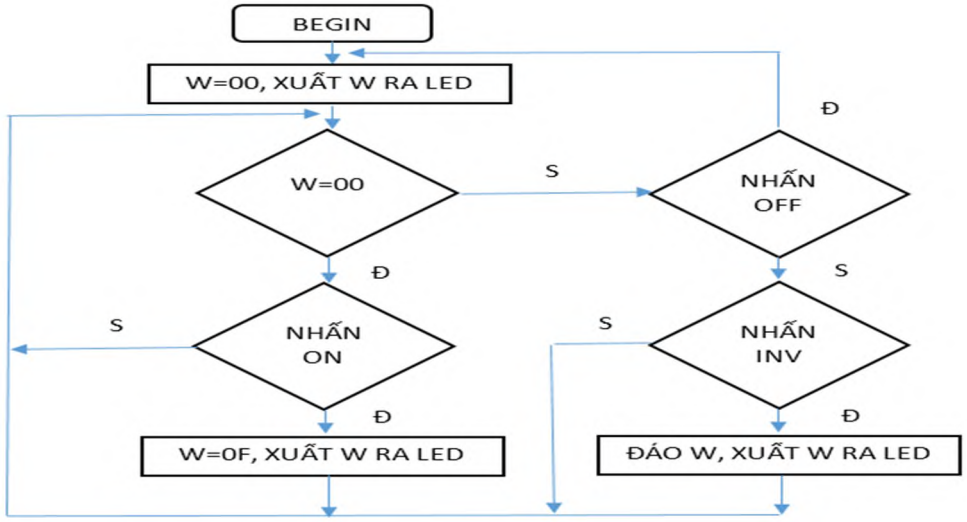
Chương trình này chúng ta sử dụng thêm lệnh **#define** (xem thêm ở mục **CÁC THÀNH PHẦN CỦA CHƯƠNG TRÌNH C → CHỈ DẪN TIỀN XỬ LÝ).**

Bài 2: Viết chương trình điều khiển 8 led đơn bằng 3 nút nhấn ON, OFF, INV. Khi cấp điện thì 8 led tắt. Khi nhấn ON thì 4 led phải sáng, 4 led trái tắt. Khi nhấn INV thì 4 led sáng thành tắt, 4 led tắt thành sáng. Khi nhấn OFF thì 8 led tắt.

- ***Sơ đồ mạch***



- ***Lưu đồ***



- ***Chương trình:***

```
#include <AT89X51.H>
#define       ON    P3_0
#define       OFF   P3_1
#define       INV   P3_2
unsigned char W;
void main ()
{       W= 0x00;    P0 = W;
        while(1)
                {       if(W == 0x00)
                                {if(ON == 0)        {W = 0x0F; P0 = W;}}
                        else
                                {if(OFF == 0)       {W = 0x00; P0 = W;}
                                 if(INV == 0)       {W = ~W;   P0 = W;}
                                }
                }
}
```

- ***Giải thích chương trình:***

Lệnh if thứ nhất kiểm tra xem W = 00 hay không? Nếu W = 00 thì mới kiểm

tra có nhấn phím ON không. Nếu nhấn ON thì gán biến W = 0F và xuất ra port làm 4 led thấp sáng, 4 led cao tắt. Nếu W khác 00 tức là đã nhấn ON thì không cần kiểm tra nhấn ON nữa, chúng ta tiến hành kiểm tra xem có nhấn OFF không?

Nếu có nhấn OFF thì gán biến W bằng 00 và xuất ra port làm led tắt. Nếu không nhấn OFF thì kiểm tra xem có nhấn phím đảo chiều INV không? Nếu có nhấn INV thì đảo giá trị của W (*sử dụng lệnh* ***~W***) rồi xuất ra port làm 4 led tắt thành sáng, 4 led sáng thành tắt. Nếu không nhấn INV thì quay lại kiểm tra xem có nhấn OFF không.

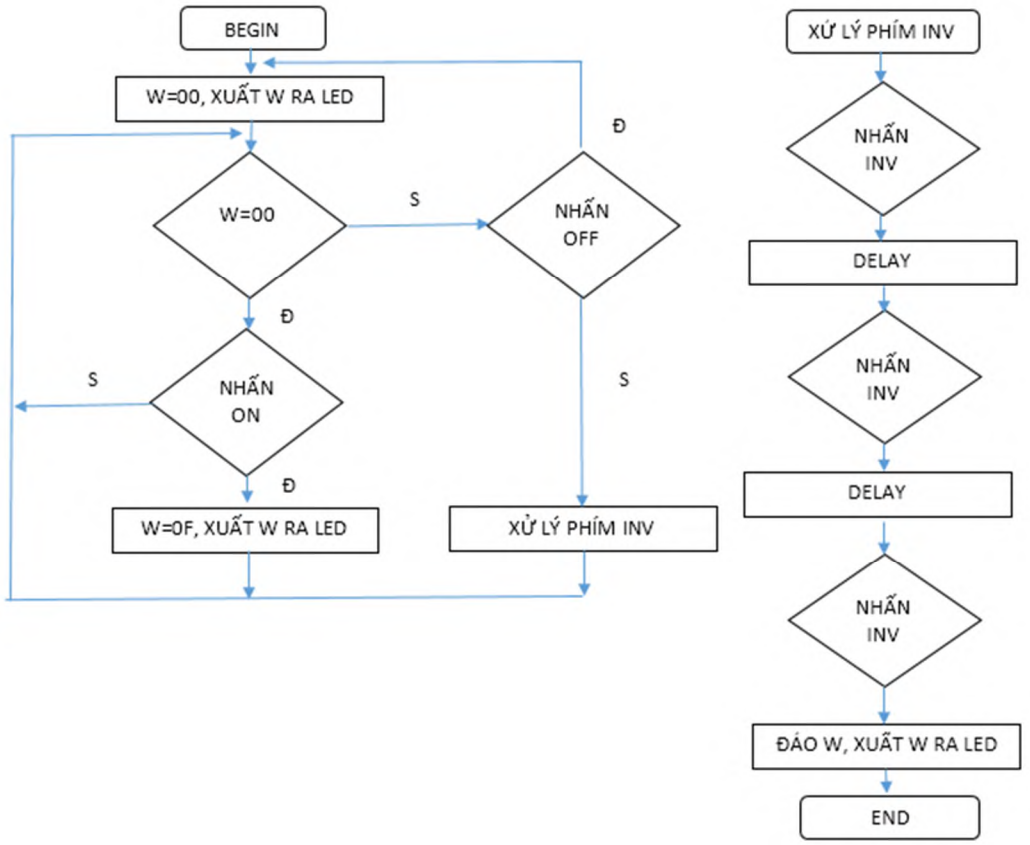
Trong chương trình này chạy thực tế thì do tốc độ của vi điều khiển quá nhanh, khi nhấn đảo chiều thì do thời gian nhấn phím dài nên vi điều khiển thực hiện trạng thái đảo led liên tục, ta sẽ nhìn thấy 8 led sáng luôn cho đến khi buông phím, hoặc nếu ta nhấn nhanh thì trạng thái đảo của led không thể xác định rõ ràng.

Để điều khiển chính xác thì phải chông dội phím nhấn, rồi xử lý chức năng và kiểm tra buông phím.

Trong bài, chỉ có phím INV gây ra ảnh hưởng nên lưu đồ và chương trình phải có thêm phần chống dội và chờ buông phím INV.

Bài 3: Viết chương trình điều khiển 8 led đơn bằng 3 nút nhấn ON, OFF, INV. Khi cấp điện thì 8 led tắt. Khi nhấn ON thì 4 led phải sáng, 4 led trái tắt. Khi nhấn INV thì 4 led sáng thành tắt, 4 led tắt thành sáng. Khi nhấn OFF thì 8 led tắt. Có sử dụng thêm chương trình chống dội cho phím INV.

- ***Lưu đồ***



- ***Chương trình***

```
#include <AT89X51.H>
#define       ON    P3_0
#define       OFF   P3_1
#define       INV   P3_2
unsigned char W;
void delay(unsigned int x)
{       unsigned int y;
              for(y=0;y<x;y++)  {}
}
void CHONGDOI_INV()
{       if(INV == 0)
              {       delay(5000);
```

```
                    if( INV == 0)
                        {delay(5000);
                        if( INV == 0)
                            { W = ~ W; P0 = W;
                              do {;}
                              while( INV == 0);}
                        }       }}
void main ()
{       W= 0x00;    P0 = W;
        while(1)
            {       if(W == 0x00)
                        {if(ON == 0)        {W = 0x0F; P0 = W;}}
                    else
                        {if(OFF == 0)       {W = 0x00; P0 = W;}
                         CHONGDOI_INV ();
                        }
            }
}
```

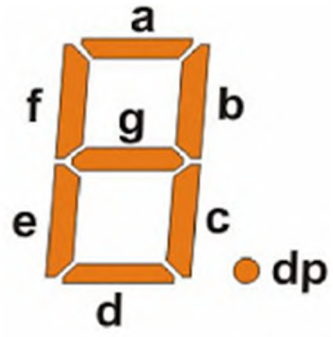
- ***Giải thích chương trình:***

Chương trình có chương trình con có tên gọi CHONGDOI_INV, ở chương trình con này sẽ kiểm tra xem có nhấn phím INV không, nếu có nhấn thì gọi hàm delay, sau đó kiểm tra lần 2 xem còn nhấn INV hay không, nếu còn thì thực hiện delay lần 2 và kiểm tra lần 3 còn nhấn phím INV không. Nếu còn thì xử lý đảo led, xuất ra port, sau đó thực hiện vòng lặp do while để chờ nhả phím thì thoát. Nếu không nhấn INV thì thoát ngay lần đầu hoặc sau delay mà không còn nhấn thì vẫn thoát.

III. LED 7 ĐOẠN

1.1 GIỚI THIỆU VỀ LED 7 ĐOẠN

Chức năng led 7 đoạn là hiển thị số thập phân.



Có 2 loại led 7 đoạn, loại anode chung và cathode chung, trong bộ thí nghiệm dùng led anode chung, mức 0 làm đoạn tương ứng của led sáng, mức 1 làm đoạn tương ứng của led tắt.

Các phương pháp điều khiển led 7 đoạn:

- Điều khiển led 7 đoạn trực tiếp: mỗi port chỉ điều khiển 1 led 7 đoạn.
- Điều khiển led 7 đoạn bằng phương pháp quét: mỗi port có thể điều khiển nhiều led 7 đoạn.

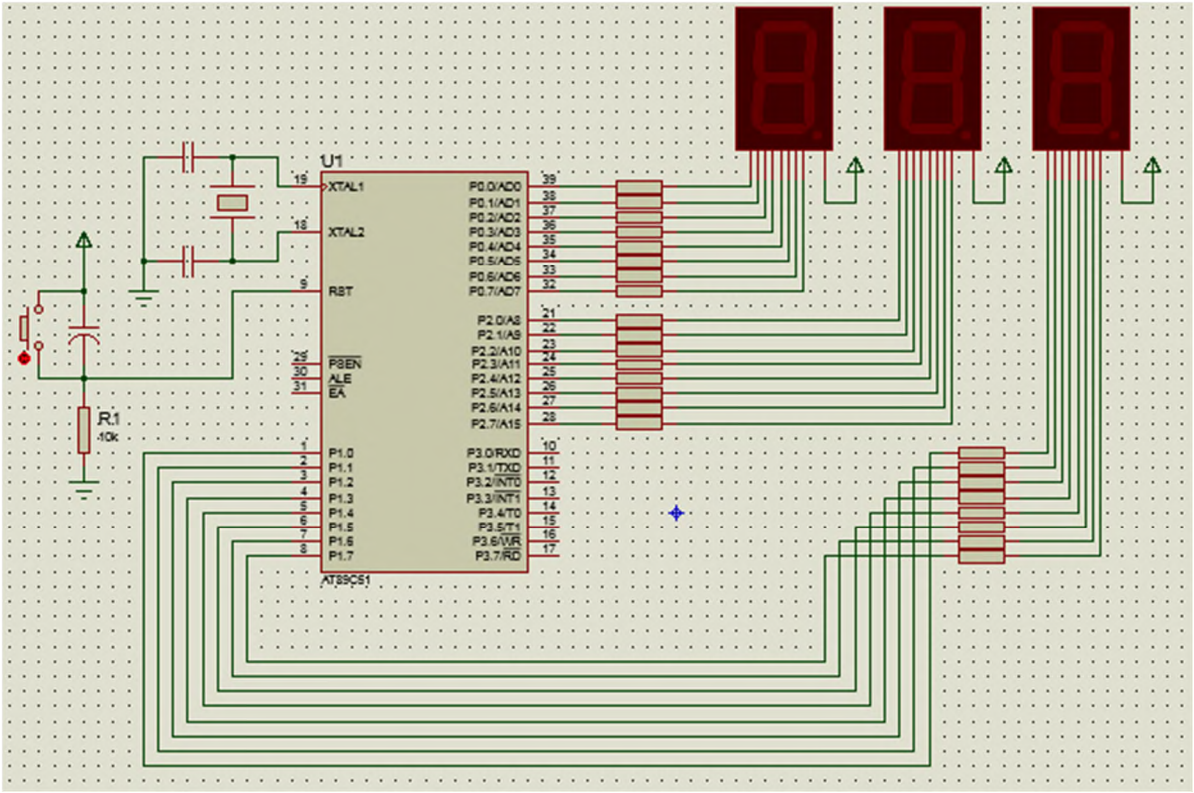
Bảng mã 7 đoạn của các số từ 0 đến 9 như sau:

Số hiển thị trên led 7 đoạn	Mã hiển thị led 7 đoạn dạng nhị phân	Mã hiển thị led 7 đoạn dạng thập lục phân
	h g f e d c b a	
0	1 1 0 0 0 0 0 0	C0
1	1 1 1 1 1 0 0 1	F9
2	1 0 1 0 0 1 0 0	A4
3	1 0 1 1 0 0 0 0	B0
4	1 0 0 1 1 0 0 1	99
5	1 0 0 1 0 0 1 0	92
6	1 1 0 0 0 0 1 0	82
7	1 1 1 1 1 0 0 0	F8
8	1 0 0 0 0 0 0 0	80
9	1 0 0 1 0 0 0 0	90
A	1 0 0 0 1 0 0 0	88
B	1 0 0 0 0 0 1 1	83
C	1 1 0 0 0 1 1 0	C6
D	1 0 1 0 0 0 0 1	A1
E	1 0 0 0 0 1 1 0	86
F	1 0 0 0 1 1 1 0	8E
-	1 0 1 1 1 1 1 1	BF

1.2 ĐIỀU KHIỂN LED 7 ĐOẠN TRỰC TIẾP

Bài tập 1: Viết chương trình điều khiển 2 led 7 đoạn hiển thị 2 số 0 và 1.

- ***Sơ đồ mạch***

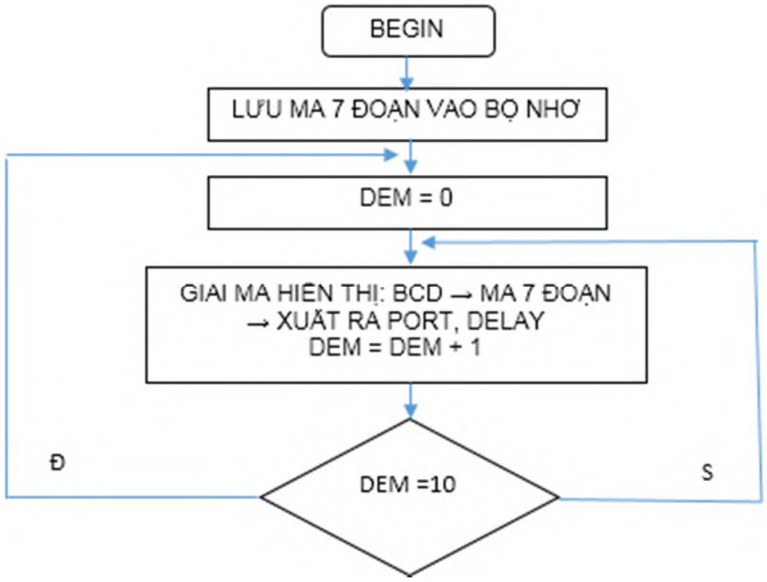


- ***Chương trình***

```
#include <AT89X51.H>
void main ()
{
        while(1)
                {       P0=0xF9;     P1=0xC0;}
}
```

Bài tập 2: Viết chương trình điều khiển 1 led 7 đoạn đếm từ 0 đến 9 dùng port 0.

- ***Lưu đồ***



- ***Giải thích lưu đồ***

B1: 10 mã 7 đoạn của 10 số thập phân từ 0 đến 9 được lưu vào bộ nhớ.

B2: Cho biến **DEM** bắt đầu từ 0.

B3: Chuyển đổi giá trị của biến **DEM** dạng số nhị phân thành số BCD hàng đơn vị và hàng chục, chuyển mã BCD thành mã 7 đoạn rồi gởi ra port điều khiển led sáng đúng số thập phân.

B4: Thực hiện delay để trì hoãn sau đó tăng biến **DEM** lên 1.

B5: So sánh biến **DEM** xem bằng 10 hay chưa? Nếu chưa bằng 10 thì quay bước 3, nếu bằng 10 thì quay lại bước 2.

- ***Chương trình***

```
#include <AT89X51.H>
unsigned char
MA7D[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
signed char DEM;
void delay(unsigned int x)
{       unsigned int y;
            for(y=0;y<x;y++)  {}
}
void main ()
```

```
{
        while(1)
                {
                        for(DEM=0;DEM<10;DEM++)
                                {P0 = MA7D[DEM]; delay(20000);}
                }
}
```

- ***Giải thích chương trình:***

Sau phần khai báo thư viện là khai báo mảng 10 ký tự của 10 mã 7 đoạn từ 0 đến 9, tên của mảng là **MA7D**, kiểu ký tự không dấu.

Tiếp theo là chương trình khai báo biến **DEM** kiểu số nguyên có dấu.

Chương trình con delay với tham số truyền là 20000.

Chương trình chính cho vòng lặp for với biến **DEM** chạy từ 0 đến 10, mỗi lần tăng lên 1 đơn vị và lấy mã 7 đoạn trong mảng **MA7D** với chỉ số là biến **DEM.** Nếu biến **DEM** bắt đầu là 0 thì lấy phần tử thứ 0 của mảng là **0xc0**, sau đó lấy biến **DEM** tăng lên, lúc này lấy phần tử thứ 1 của mảng là **0xF9**, tương tự cho đến khi bằng 9 thì lấy phần tử thứ 9 của mảng là **0x90**. Các phần tử của mảng sẽ được gán cho Port 0 để điều khiển led sáng lần lượt từ 0 đến 9.

Bài tập 3: Viết chương trình điều khiển 1 led 7 đoạn đếm từ 9 về 0 dùng Port 0.

- ***Gợi ý: hiệu chỉnh vòng lặp for:***

```
{
        for(DEM=9;DEM>-1;DEM--)
                        {P0=MA7D[DEM];        delay(20000);}
}
```

Bài tập 4: Viết chương trình điều khiển 2 led 7 đoạn đếm từ 00 đến 99.

- ***Lưu đồ: xây dựng lưu đồ tương tự như bài tập 2 (Chú ý: so sánh biến DEM với 100)***

- ***Chương trình:***

```
#include <AT89X51.H>
unsigned char
MA7D[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
signed char DEM,CHUC,DONVI;
void delay(unsigned int x)
{       unsigned int y;
            for(y=0;y<x;y++)  {}
}
void main ()
{
      while(1)
      {
      for(DEM=0;DEM<100;DEM++)
            {     DONVI = DEM%10;     CHUC = DEM/10;
                  P0 = MA7D[DONVI];   P1 = MA7D[CHUC];
                  delay(10000);
            }
      }
}
```

- ***Giải thích chương trình:***

Vòng lặp for cho biến **DEM** tăng từ giá trị 0 lên đến nhỏ hơn 100, mỗi lần tăng 1 rồi thực hiện các lệnh:

Lệnh **DONVI=DEM%10** sẽ lấy giá trị biến **DEM** chia cho 10 và gán số dư cho biến **DONVI.** Lệnh **CHUC=DEM/10** sẽ lấy giá trị biến **DEM** chia cho 10 và gán kết quả cho biến **CHUC.** Ví dụ: biến **DEM** =15 thì sau khi thực hiện 2 lệnh chia ta sẽ có biến **DONVI = 5** và **CHUC=1.**

Hai lệnh trên có chức năng tách hàng chục và hàng đơn vị của biến DEM thành 2 số độc lập để giải mã 7 đoạn.

Hai lệnh tiếp theo tiến hành giải mã 7 đoạn và gán cho 2 port để hiển thị trên 2 led.

Bài tập 5: Viết chương trình điều khiển 2 led 7 đoạn đếm từ 00 đến 25.

Bài tập 6: Viết chương trình điều khiển 2 led 7 đoạn đếm từ 05 đến 25.

Bài tập 7: Viết chương trình điều khiển 2 led 7 đoạn đếm từ 99 đến 00.

Bài tập 8: Viết chương trình điều khiển 3 led 7 đoạn đếm từ 000 đến 199.

- ***Lưu đồ: xây dựng lưu đồ tương tự như bài tập 2 (Chú ý: so sánh biến DEM với 200)***
- ***Chương trình:***

```
#include<AT89X51.H>
unsigned char
MA7D[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
signed char DEM,DEMT,TRAM,CHUC,DONVI;
void delay(unsigned int x)
{       unsigned int y;
            for(y=0;y<x;y++)  {}
}
void GIAIMA_HIENTHI ()
      {
            DEMT=DEM;
            DONVI=DEMT%10;      DEMT=DEMT/10;
            CHUC=DEMT%10;       TRAM=DEMT/10;
            P0=MA7D[DONVI];     P1=MA7D[CHUC];
      P2=MA7D[TRAM];
      }
void main ()
{
      while(1)
      {
```

```
        for(DEM=0;DEM<200;DEM++)
                {GIAIMA_HIENTHI(DEM);    delay(20000); }
        }
}
```

- ***Giải thích chương trình:***

Trong bài, biến DEM chạy từ 0 đến 199, giá trị của biến DEM gồm hàng đơn vị, hàng chục, hàng trăm nên phải chia để lấy hàng đơn vị, hàng chục, hàng trăm.

Bài tập 9: Viết chương trình điều khiển 3 led 7 đoạn đếm từ 255 đến 000.

Bài tập 10: Viết chương trình điều khiển 3 led 7 đoạn đếm từ 000 đến 150 rồi đếm xuống 000.

(Gợi ý: Cho vòng lặp for đếm lên sau đó thực hiện vòng lặp for cho đếm xuống.)

Bài tập 11: Viết chương trình điều khiển 3 led 7 đoạn đếm từ 000 đến 999.

1.3 ĐIỀU KHIỂN LED 7 ĐOẠN BẰNG PHƯƠNG PHÁP QUÉT

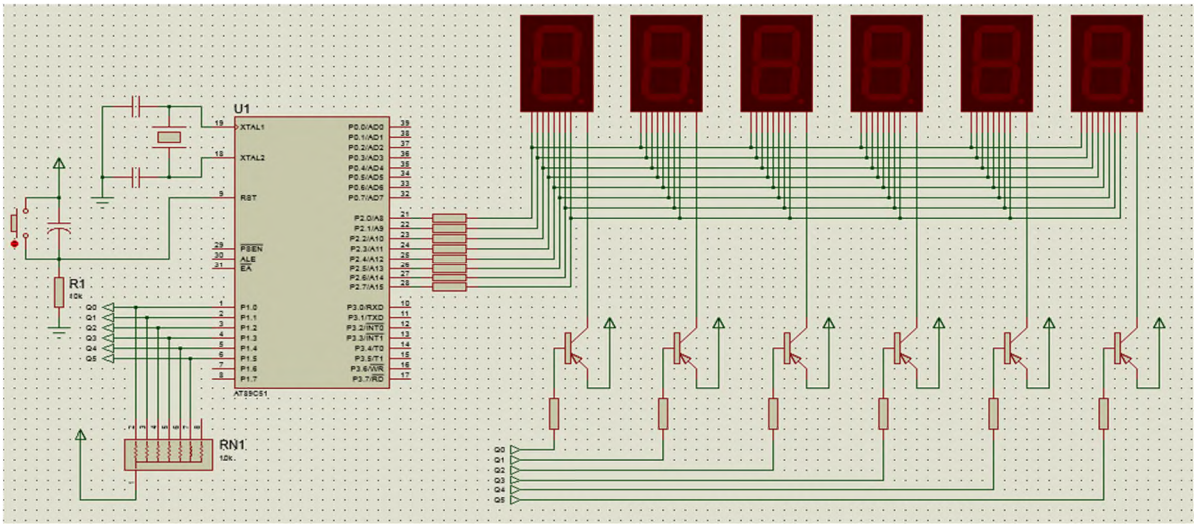
Khi số led 7 đoạn cần nhiều hơn ví dụ như 8 led thì ta không thể kết nối trực tiếp 1 Port điều khiển 1 led vì không đủ port. Có nhiều phương pháp mở rộng để điều khiển được nhiều led, trong tài liệu này sẽ trình bày phương pháp quét 8 led dùng 2 port.

Để điều khiển 8 led 7 đoạn cần dùng 16 đường điều khiển: 8 đường điều khiển 7 đoạn a, b, c, d, e, f, g, dp và 8 đường điều khiển đóng ngắt transistor.

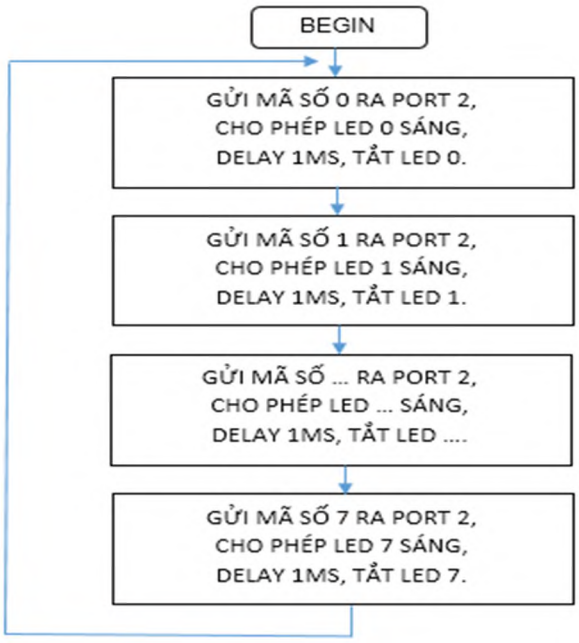
Tại mỗi thời điểm ta chỉ cho 1 transistor dẫn và 7 transistor còn lại tắt, dữ liệu gởi ra sẽ sáng trên led tương ứng với transitor dẫn. Sau đó cho 1 transitor khác dẫn và gửi dữ liệu hiển thị cho led đó, quá trình điều khiển này diễn ra lần lượt cho đến khi hết 8 led. Do tốc độ quét nhanh hơn đáp ứng của mắt nên bạn không nhìn thấy led tắt mà sẽ thấy 8 led sáng 1 lúc.

Bài tập 1: Viết chương trình điều khiển hiển thị từ số 0 đến số 7 trên 8 led dùng phương pháp quét. Sử dụng Port 1 điều khiển 8 transistor đóng ngắt và Port 2 điều khiển 8 led.

- ***Sơ đồ mạch***



- ***Lưu đồ***



- ***Chương trình***

```
#include <AT89X51.H>
void delay(unsigned int x)
{       unsigned int y;
                for(y=0;y<x;y++)  {;}
}
void main ()
{
```

```
        while(1)
                {
                        P2=0xC0;    P1_0=0;      delay(200);  P1_0=1;
                        P2=0xF9;    P1_1=0;      delay(200);  P1_1=1;
                        P2=0xA4;    P1_2=0;      delay(200);  P1_2=1;
                        P2=0xB0;    P1_3=0;      delay(200);  P1_3=1;
                        P2=0x99;    P1_4=0;      delay(200);  P1_4=1;
                        P2=0x92;    P1_5=0;      delay(200);  P1_5=1;
                        P2=0x82;    P1_6=0;      delay(200);  P1_6=1;
                        P2=0xF8;    P1_7=0;      delay(200);  P1_7=1;
                }
}
```

- ***Giải thích chương trình:***

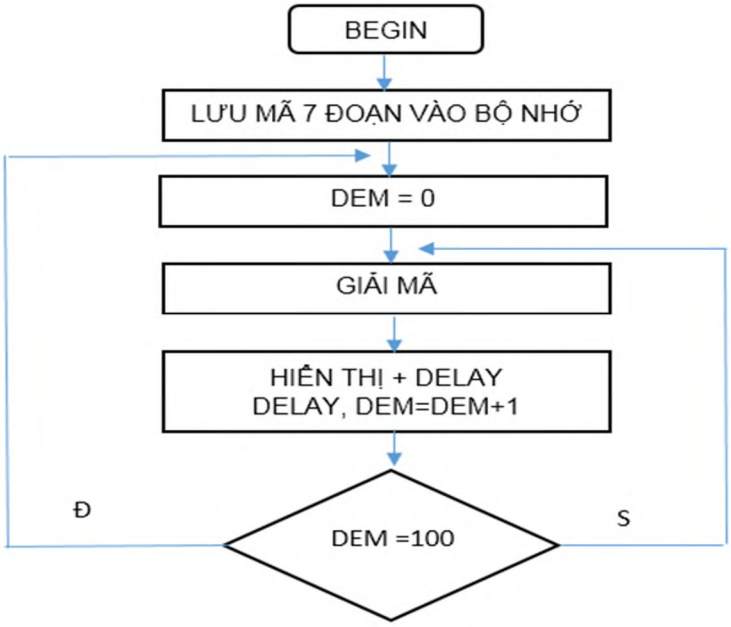
Lệnh **P2=0xC0** có chức năng gán mã số 0 cho Port 2.

Lệnh **P1_0=0** có chức năng làm bit thứ 0 của Port 1 xuống mức 0 để transistor cấp dòng cho led, lệnh **delay(200)** trì hoãn 1 ms sau đó tắt transitor để tắt led.

Lặp lại cho led tiếp theo, thời gian delay nhỏ khoảng 1ms thì ta sẽ thấy 8 led sáng, delay lớn hơn thì led sẽ sáng chập chờn hoặc từng led sáng.

Bài tập 2: Viết chương trình đếm từ 00 đến 99 hiển thị trên 2 led 7 đoạn bằng phương pháp quét.

- ***Lưu đồ***



Chương trình

```
#include <AT89X51.H>
unsigned char
MA7D[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
unsigned char DEM,CHUC,DONVI,TAM;
unsigned char MCHUC,MDONVI;
void delay(unsigned int x)
{       unsigned int y;
            for(y=0;y<x;y++)  {}
}
void HIENTHI_DELAY()
{       for(TAM=0;TAM<100;TAM++)
                    P2=MDONVI;      P1_0=0;      delay(200); P1_0=1;
                    P2=MCHUC;       P1_1=0;      delay(200);  P1_1=1;
}
void GIAIMA()
{       DONVI=DEM%10;       CHUC=DEM/10;
        MDONVI=MA7D[DONVI];    MCHUC=MA7D[CHUC];
```

```
}
void main()
{
        while(1)
        {
        for(DEM=0;DEM<100;DEM++)
                {       GIAIMA();
                        HIENTHI_DELAY();
                }
        }
}
```

BÀI 4: BỘ ĐINH THỜI
Mã bài: MĐ23-05

Giới thiệu:

Trong quá trình viết chương trình dùng vi điều khiển để vận hành một hệ thống thì việc khởi tạo và đọc đúng bộ định thời là một vấn đề quan trọng.

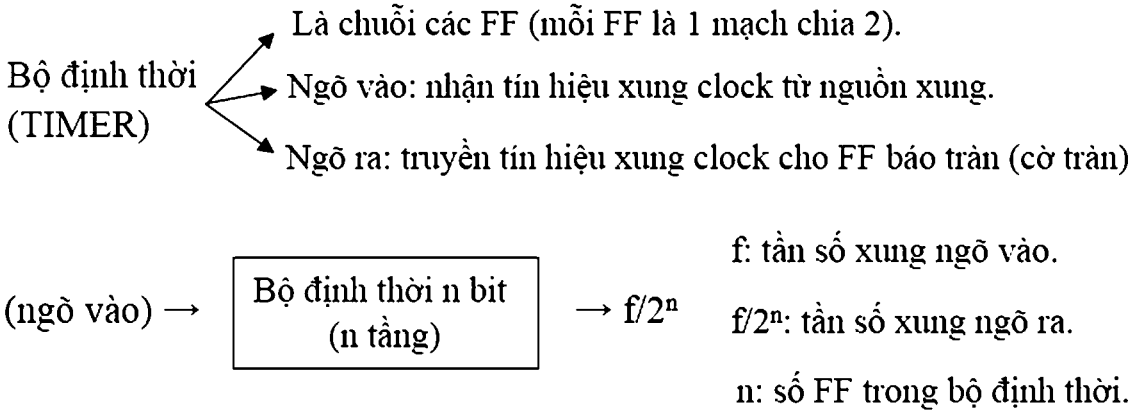
Việc hiểu và sử dụng đúng chức năng của bộ timer là một yêu cầu cấp thiết trong việc lập trình cho vi điều khiển. Trong bài 4, chúng ta đi vào tìm hiểu cách thức khởi tạo và sử dụng bộ đinh thời của vi điều khiển.

Mục tiêu:

- Trình bày cấu tạo và các chế độ làm việc của bộ định thời 8051 theo nội dung đã học
- Thực hiện khởi tạo bộ nhớ đúng yêu cầu kỹ thuật
- Thực hiện đọc bộ định thời trong khi hoạt động đúng yêu cầu kỹ thuật
- Thực hiện lập trình điều khiển dùng bộ định thời đúng yêu cầu kỹ thuật

Nội dung chính:

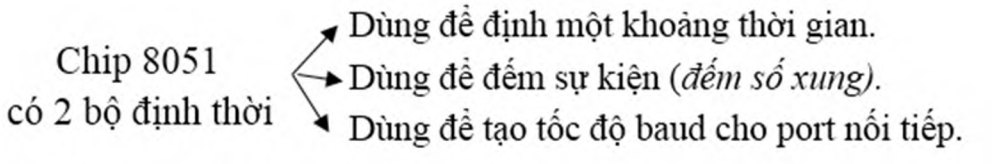
I. *MỞ ĐẦU*



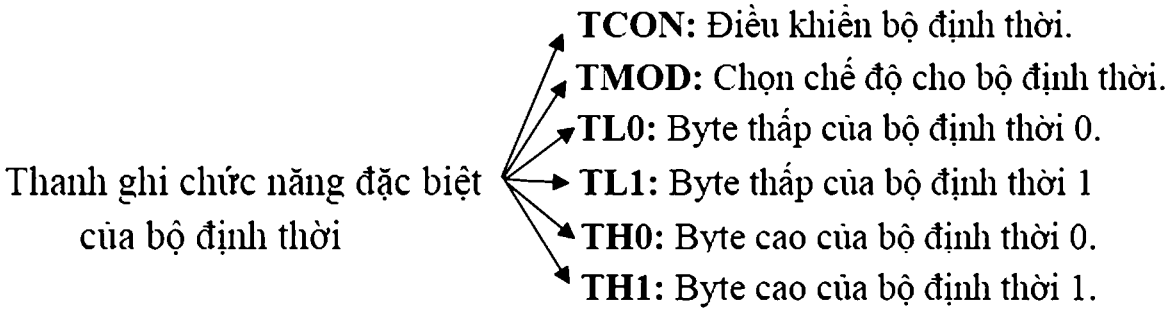
- Tần số: tần số xung ngõ ra bằng tần số xung ngõ vào chia cho 2^n.
- Giá trị: giá trị nhị phân trong các FF của bộ định thời là số đếm của các xung clock tại ngõ vào từ khi bộ định thời bắt đầu đếm.
- Tràn: xảy ra hiện tượng tràn (*cờ tràn=1)* khi số đếm chuyển từ giá trị lớn nhất xuống giá trị nhỏ nhất của bộ định thời.

Ví dụ: Bộ định thời 16bit (*chứa 16 FF bên trong).*

- ✓ Tần số: $f_{OUT} = f_{IN} / 2^{16} = f_{IN} / 65536$
- ✓ Giá trị: số đếm này trong khoảng 0 (*0000H)* → 65535 (*FFFFH).*
- ✓ Tràn: cờ tràn bằng 1 khi số đếm từ FFFFH chuyển xuống 0000H.



- **Ứng dụng định thời gian(*TIMER*):** bộ định thời được lập trình sao cho sẽ tràn sau một khoảng thời gian đã qui định và khi đó cờ tràn của bộ định thời sẽ bằng 1.
- **Ứng dụng đếm sự kiện (*COUNTER):*** để xác định số lần xuất hiện của một kích thích ngoài tới một chân của chip 8051 (*kích thích là sự chuyển trạng thái từ 1 xuống 0).*
- **Ứng dụng tạo tốc độ baud cho port nối tiếp**



II. THANH GHI SFR CỦA TIMER

1. THANH GHI CHẾ ĐỘ TMOD

Thanh ghi TMOD (*Timer Mode Register)* chứa các bit dùng để thiết lập chế độ hoạt động cho bộ định thời 0 và bộ định thời 1.

Thanh ghi TMOD được nạp một giá trị một lần tại thời điểm bắt đầu của chương trình để qui định chế độ hoạt động của các bộ định thời.

Cấu trúc thanh ghi TMOD:

GATE	C/T	M1	M0	GATE	C/T	M1	M0
TIMER 1				TIMER 0			

Với **TIMER 0:**

- M0: Bit chọn chế độ hoạt động cho bộ định thời.
- M1: Bit chọn chế độ hoạt động cho bộ định thời.
- C/T: Bit chọn chức năng đếm hoặc định thời.
  - ✓ C/T=1: Bộ định thời là bộ đếm *(Counter).*
  - ✓ C/T=0: Bộ định thời là bộ định khoảng thời gian *(Timer).*
- GATE: Bit điều khiển cổng.
  - ✓ GATE=0: Bộ định thời hoạt động khi bit TR0=1 *(điều khiển bằng phần mềm).*
  - ✓ GATE=1: Bộ định thời hoạt độn khi chân INT0=1 (*điều khiển bằng phần cứng).*

Với **TIMER 1:**

- M0: Bit chọn chế độ hoạt động cho bộ định thời.
- M1: Bit chọn chế độ hoạt động cho bộ định thời.
- C/T: Bit chọn chức năng đếm hoặc định thời.

- ✓ C/T=1: Bộ định thời là bộ đếm *(Counter).*
- ✓ C/T=0: Bộ định thời là bộ định khoảng thời gian *(Timer).*

- GATE: Bit điều khiển cổng.
  - ✓ GATE=0: Bộ định thời hoạt động khi bit TR1=1 *(điều khiển bằng phần mềm).*
  - ✓ GATE=1: Bộ định thời hoạt độn khi chân INT1=1 (*điều khiển bằng phần cứng).*

Các chế độ hoạt động của bộ định thời:

M1	M0	Chế độ	Mô tả
0	0	0	Chế độ định thời 13 bit
0	1	1	Chế độ định thời 16 bit
1	0	2	Chế độ định thời 8 bit tự động nạp lại
1	1	3	Chế độ chia sẻ định thời: • **TIMER 0**: được tách ra làm 2 timer 8 bit. ✓ Timer 8 bit TL0 được điều khiển bởi các bit của T0. ✓ Timer 8 bit TH0 được điều khiển bởi các bit của T1. • **TIMER 1:** không được hoạt động ở chế độ này.

2. THANH GHI ĐIỀU KHIỂN TCON

Thanh ghi TCON (*Time Control Register)* chứa các bit dùng để điều khiển và báo trạng thái của bộ định thời 0 và bộ định thời 1.

Cấu trúc thanh ghi TCON:

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Hoạt động của thanh ghi TCON được tóm tắt trong bảng dưới đây:

Bit	Ký	Chức năng

	hiệu	
7	TF1	Cờ tràn Timer 1.
6	TR1	Bit điều khiển Timer 1 đếm/ngừng đếm: • TR1=1 thì timer được phép đếm khi có xung. • TR1=0 thì timer không được phép đếm (ngừng).
5	TF0	Cờ tràn Timer 0.
4	TR0	Bit điều khiển Timer 0 đếm/ngừng đếm: • TR0=1 thì timer được phép đếm khi có xung. • TR0=0 thì timer không được phép đếm (ngừng).
3	IE1	Cờ báo ngắt INT1. IE1=1 khi tín hiệu ngắt xuất hiện tại chân INT1.
2	IT1	Bit lựa chọn ngắt INT1 tác động bằng mức hay cạnh: • IT1=0 thì ngắt INT1 tác động bằng mức. • IT1=1 thì ngắt INT1 tác động bằng cạnh xuống.
1	IE0	Cờ báo ngắt INT0. IE0=1 khi tín hiệu ngắt xuất hiện tại chân INT0.
0	IT0	Bit lựa chọn ngắt INT0 tác động bằng mức hay cạnh: • IT0=0 thì ngắt INT1 tác động bằng mức. • IT0=1 thì ngắt INT1 tác động bằng cạnh xuống.

Khi Timer bị tràn thì cờ tràn TFx =1, ta có thể xoá xoá cờ ngắt bằng phần mềm hoặc tự động xoá khi vi điều khiển thực hiện chương trình con phục vụ ngắt Timer.

Khi có ngắt xảy ra ở ngõ vào INTx sẽ làm cờ IEx lên mức 1.

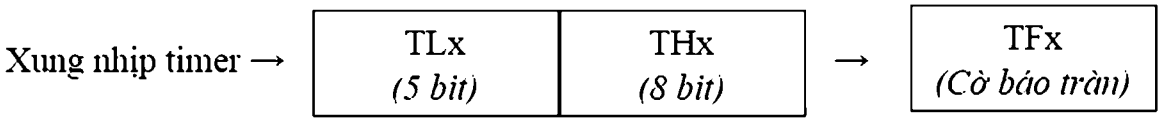
Khi vi điều khiển thực hiện chương trình con phục vụ ngắt INTx thì tự động xoá luôn cờ báo ngắt IEx.

***Chú ý:* "x"** lả 0 hoặc 1 nói chung cho cả T0 và T1 hoặc INT0 và INT1.

III. CÁC CHẾ ĐỘ LÀM VIỆC

1. CHẾ ĐỘ TIMER 13 BIT

Ở chế độ này 8 bit cao sử dụng hết 8 bit của thanh ghi THx, 5 bit còn lại chỉ sử dụng 5 bit trong số thấp của thanh ghi TLx để tạo bộ định thời, còn 3 bit cao của thanh ghi TLx không sử dụng.



Ở chế độ này thì giá trị đếm lớn nhất là 2^{13} = 8192 tức là đếm từ "0 0000 0000 0000B" đến "1 1111 1111B" (*0000H đến 1FFFH)*.

Khi có xung clock, bộ định thời bắt đầu đếm lên từ giá trị chứa trong THx/TLx.

Xảy ra tràn *(cờ tràn TFx=1)* khi số đếm chuyển từ 1FFFH sang 0000H và đếm sẽ tiếp tục đếm lên từ giá trị 0000H.

2. CHẾ ĐỘ TIMER 16 BIT

Chế độ này sử dụng thanh ghi THx và TLx để tạo ra bộ định thời.

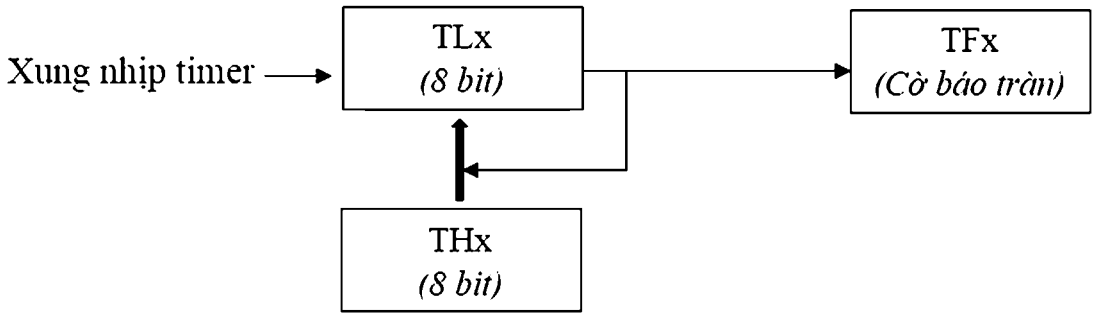
Ở chế độ này thì giá trị đếm lớn nhất là 2^{16} = 65535 tức là đếm từ "0000 0000 0000 0000B" đến "1111 1111 1111 1111B" (*0000H đến FFFFH).*

Khi có xung clock, bộ định thời bắt đầu đếm lên từ giá trị chứa trong THx/TLx.

Xảy ra tràn *(cờ tràn TFx=1)* khi số đếm chuyển từ FFFFH sang 0000H và đếm sẽ tiếp tục đếm lên từ giá trị 0000H.

3. CHẾ ĐỘ TỰ NẠP LẠI 8 BIT

Ở chế độ này byte thấp TLx của timer làm việc như một timer 8 bit trong khi đó byte cao THx của timer dùng để lưu trữ giá trị để nạp lại cho thanh ghi TLx.



Số đếm ở chế độ này từ "0000 0000B" đến "1111 1111" (*00H đến FFH).*

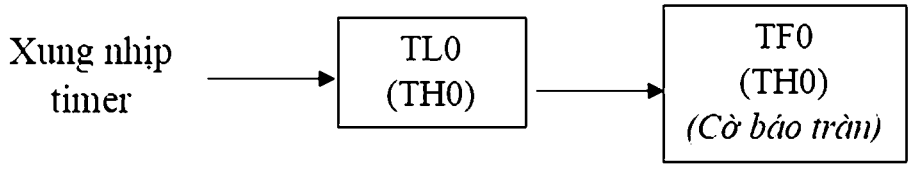
Khi có xung clock, bộ định thời bắt đầu đếm từ giá trị chứa trong TLx (*THx không thay đổi giá trị).*

Xảy ra tràn (*cờ tràn TFx=1)* khi số đếm chuyển từ FFH sang 00H, đồng thời giá trị trong THx sẽ được nạp vào TLx và việc đếm sẽ tiếp tục đếm lên từ giá trị chứa trong thanh ghi TLx *(giá trị này bằng với giá trị của THx).*

4. CHẾ ĐỘ ĐỊNH THỜI CHIA SẺ

Ở chế độ này:

- Bộ định thời 0 được chia ra:
  - ✓ Bộ định thời 8 bit thứ I:
    - ➢ Sử dụng thanh ghi TL0 để tạo ra bộ định thời.
    - ➢ Số đếm: 00H – FFH.
    - ➢ Thanh ghi TL0 chứa giá trị của bộ định thời.
    - ➢ Khi có xung clock, bộ định thời bắt đầu đếm lên từ giá trị chứa trong TL0.
    - ➢ Xảy ra tràn *(cờ tràn TF0=0)* khi số đếm chuyển từ FFH sang 00H và việc đếm sẽ tiếp tục bắt đầu đếm lên từ giá trị 00H.
  - ✓ Bộ định thời 8 bit thứ II:
    - ➢ Sử dụng thanh ghi TH0 để tạo ra bộ định thời.
    - ➢ Số đếm: 00H – FFH.
    - ➢ Thanh ghi TH0 chứa giá trị của bộ định thời.
    - ➢ Khi có xung clock, bộ định thời bắt đầu đếm lên từ giá trị chứa trong TH0.
    - ➢ Xảy ra tràn *(cờ tràn TF1=0)* khi số đếm chuyển từ FFH sang 00H và việc đếm sẽ tiếp tục bắt đầu đếm lên từ giá trị 00H.
- Bộ định thời 1: không hoạt động ở chế độ này.

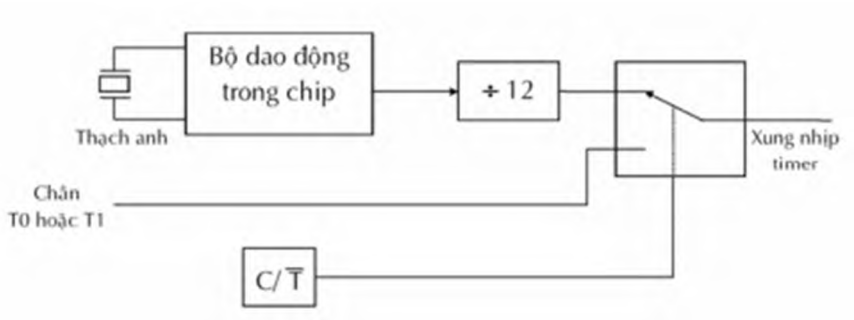


IV. NGUỒN CẤP XUNG CHO TIMER

Nguồn xung bộ định thời được tạo ra từ:

- Mạch dao động trên chip → dùng cho tính năng định thời gian.
- Xung kích thích bên ngoài → dùng cho tính năng đếm sự kiện.

1. TRƯỜNG HỢP ĐỊNH THỜI GIAN



Nếu C/T=0 thì:

- Bộ định thời được dùng để định thời gian *(Timer).*
- Nguồn xung clock định thời được lấy từ mạch dao động trên chip.

Chú ý:

- Tần số xung clock cung cấp cho bộ định thời bằng 1/12 tần số của mạch dao động trên chip 8051.
- Thời gian định thời là khoảng thời gian được tính từ lúc bộ định thời bắt đầu đếm lên *(từ giá trị chứa trong các thanh ghi THx/TLx)* cho đến lúc bộ định thời bắt đầu tràn (*thời gian này phụ thuộc vào giá trị ban đầu được nạp cho các thanh ghi THx và TLx).*

2. CHỨC NĂNG ĐẾM SỰ KIỆN

Nếu C/T=1 thì:

- Bộ định thời được dùng để đếm sự kiện *(Counter).*
- Nguồn xung clock định thời được lấy từ xung kích thích tại hai chân T0 và T1 của chip 8051.

Chú ý:

- Tần số kích thích tối đa cho phép tại chân T0 và T1:

$$f_{T0,T1(MAX)} = f_{TIMER} / 2$$

f_{TIMER} : tần số xung clock định thời.

$f_{T0,T1(MAX)}$: tần số kích thích tối đa cho phép tại T0 và T1

V. *KHỞI ĐỘNG, DỪNG, ĐIỀU KHIỂN TIMER*

- Cách 1: *(thường dùng để định thời gian).*

Điều kiện sử dụng: bit GATE=0

(*Phương pháp điều khiển bằng phần mềm)*

→ Bộ định thời x chạy khi bit TRx =1.

→ Bộ định thời x dừng khi bit TRx=0.

- Cách 2: *(thường dùng để đo độ rộng xung tại chân INTx)*

Điều kiện sử dụng: bit GATE=1 và bit TRx=1

(*Phương pháp điều khiển bằng phần cứng)*

→ Bộ định thời x chạy khi chân INTx=1.

→ Bộ định thời x chạy khi chân INTx=0.

VI. *KHỞI TẠO VÀ TRUY XUẤT THANH GHI TIMER*

1. KHỞI TẠO BỘ ĐỊNH THỜI

Trước khi các bộ định thời hoạt động cần phải:

- Qui định chế độ của bộ định thời → Thanh ghi TMOD.
- Quy định điểm bắt đầu đếm của bộ định thời *(khoảng thời gian định thời)*

→ Thanh ghi THx/TLx.

2. TRUY XUẤT GIÁ TRỊ CỦA BỘ ĐỊNH THỜI ĐANG HOẠT ĐỘNG

Trong các ứng dụng thực tế, ta cần phải đọc giá trị *(nội dung)* chứa trong các thanh ghi định thời THx/TLx trong khi bộ định thời vẫn đang hoạt động. Do giá trị của bộ định thời được chứa trong cả hai thanh ghi THx và TLx nên ta phải đọc hai thanh ghi này bằng hai dòng lệnh liên tiếp nhau. Một sự sai pha có thể xuất hiện nếu có sự tràn từ byte thấp chuyển sang byte cao giữa hai lần đọc và do vậy ta không thể đọc đúng giá trị cần đọc.

Giải pháp đưa ra là trước tiên ta phải đọc byte cao, kế đến đọc byte thấp và rồi đọc byte thấp thêm lần nửa. Nếu byte cao thay đổi giá trị, ta lặp lại thao tác đọc vừa nêu.

3. CÁC KHOẢNG THỜI GIAN ĐỊNH THỜI

Khảo sát trước hợp 8051 dùng thạch anh 12MHZ:

- Khoảng thời gian định thời ngắn nhất (**µs): 1**
- Khoảng thời gian định thời dài nhất (**µs):**
  - ✓ ≈ 10 → Dùng các lệnh.
  - ✓ ≤ 256 → Dùng bộ định thời 8 bit tự động nạp lại.
  - ✓ ≤ 65536 → Dùng bộ định thời 16 bit.
  - ✓ Không giới hạn → Dùng bộ định thời 16 bit + các vòng lặp.

VII. TIMER 2 CỦA 8052

Các thanh ghi của timer/couter T2 bao gồm: thanh ghi TL2, TH2, thanh ghi T2CON, thanh ghi T2MOD, thanh ghi PCAP2L và RCAP2H.

Timer/Counter T2 có thể dùng để định thời timer hoặc dùng như bộ đếm counter để đếm xung ngoài đưa đến ngõ vào P1.0.

Timer/Counter T2 có 3 kiểu hoạt động: đếm tự động nạp lại, chế độ Capture và thiết lập tốc độ baud để phục vụ cho truyền dữ liệu.

1. KHẢO SÁT THANH GHI T2MOD

Cấu trúc thanh ghi T2MOD:

						T2OE	DCEN

BIT 7 BIT 0

Bit DCEN *(Down Counter Enable):* khi bằng 0 thì chỉ đếm lên, khi bằng 1 thì T2 được định cấu hình như bộ đếm lên/xuống.

Bit T2OE: bit cho phép T2 xuất.

2. KHẢO SÁT THANH GHI T2CON

Cấu trúc thanh ghi T2CON:

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	$C/\overline{T2}$	$CP/\overline{RL2}$

BIT 7 BIT 0

Chức năng của thanh ghi điều khiển

T2CON:

Bit	Ký hiệu	Chức năng
7	TF2	Cờ tràn Timer 2
6	EXF2	Cờ báo có xung ngoài đưa đến T2XE của T2: lên 1 khi xảy ra ở chế độ capture hoặc nạp lại dữ liệu chuyển trạng thái từ 1 sang 0 ở ngõ vào T2EX và EXEN2=1. Khi cho phép T2 ngắt, EXF2=1 thì CPU sẽ thực hiện chương trình con phục vụ ngắt T2, bit EXF2 phải xoá bằng phần mềm. Bit EXF2 không tạo ra ngắt ở chế độ đếm lên/xuống.
5	RCLK	Bit cho phép nhận lựa chọn xung clock để nhận dữ liệu nối tiếp: • Khi bit RCLK=1 thì T2 tạo tốc độ baud cho port nối tiếp để nhận dữ liệu còn timer T1 sẽ cung cấp tốc độ boud cho port nối tiếp để phát dữ liệu đi. • Khi bit RCLK=0 thì T1 sẽ tạo tốc độ baud để nhận dữ liệu.
4	TCLK	Bit cho phép nhận lựa chọn xung clock để phát dữ liệu nối tiếp: • Khi bit TCLK=1 thì T2 tạo tốc độ baud cho port nối tiếp để nhận dữ liệu về. • Khi bit TCLK=0 thì T1 sẽ tạo tốc độ baud để nhận dữ liệu.
3	EXEN2	Bit cho phép tác động bên ngoài: • EXEN2=1 thì cho phép xuất xung báo hiệu ở ngõ ra T2EX khi hoạt động ở chế độ capture hoặc chế độ nạp lại với điều kiện là T2 không được dùng để tạo tốc độ baud. • Khi EXEN2=0 thì không cho phép xuất tín hiệu ra

		ngoài.
2	TR2	Bit điều khiển Timer1 đếm/ngừng đếm: • TR2=1 thì timer 1 được phép đếm xung. • TR2=0 thì timer 1 không được phép đếm xung (ngừng).
1	C/$\overline{T2}$	Bit lựa chọn counter hay timer: (*tích cực cạnh xuống)* • C/$\overline{T2}$=1: đếm xung từ bên ngoài đưa đến ngõ vào T2. • C/$\overline{T2}$=0: định thời đếm xung nội bên trong.
0	CP/$\overline{RL2}$	Bit lựa chọn chế độ capture hay tự động nạp lại: • CP/RL2=1: thì hoạt động Capture và khi kết thúc chuyển sang trạng thái từ 1 sang 0 ở ngõ ra T2EX khi bit EXEN2=1. • CP/$\overline{RL2}$=0: thì hoạt động đếm và tự động nạp lại và khi tràn hoặc khi ngõ vào T2EX chuyển trạng thái từ 1 sang 0 nếu bit EXEN2=1. Khi bit RCLK hoặc TCLK=1 thì bit này xem như bỏ.

3. CHẾ ĐỘ CAPTURE

Khi CP/$\overline{RL2}$=1 thì timer hoạt động ở chế độ capture.

Có 2 lựa chọn tuỳ thuộc vào bit EXEN2 trong thanh ghi T2CON.

Nếu bit EXEN2=0 thì T2 là timer hay counter 16 bit và khi đếm tràn thì làm cờ tràn lên 1 và phát sinh ngắt nếu cho phép.

Nếu bit EXEN2=1 thì T2 hoạt động đếm bình thường nhưng nếu có chuyển trạng thái của xung bên ngoài đưa đến ngõ T2EX (*cạnh xuống)* thì giá trị đếm của cặp thanh ghi TH2 và TL2 sẽ nạp sang cặp thanh ghi RCAP2H và PCAP2L tương ứng, đồng thời làm bit EXF2 trong T2CON lên 1, cờ EXF2 có thể tạo yêu cầu ngắt giống TF2.

4. KHẢO SÁT CHẾ ĐỘ ĐẾM TỰ ĐỘNG NẠP LẠI

Khi CP/$\overline{RL2}$=0 thì timer hoạt động ở chế độ đếm tự động nạp lại. Ở chế độ này thì T2 có thể lập trình để đếm lên hoặc xuống phụ thuộc vào bit DCEN.

Khi reset thì bit DCEN bằng 0 nên chế độ đếm mặc nhiên là đếm lên. Khi lập trình làm bit DCEN bằng 1 thì hoạt động đếm lên hoặc đếm xuống phụ thuộc vào giá trị của bit ở ngõ vào T2EX.

VIII. CÁC THANH GHI, CÁC BIT CỦA TIMER TRONG NGÔN NGỮ KEIL-C

Trong ngôn ngữ lập trình Keil-C đã định nghĩa sẵn các tên thanh ghi và bit của các thanh ghi. Bạn cần phải biết các tên thanh ghi và các bit để lập trình.

Thư viện đã định nghĩa các thanh ghi và các bit timer/counter bao gồm:

TCON Bit Registers	**TMOD bit Values**
sbit IT0 = 0x88;	#define T0_M0_ 0x01
sbit IE0 = 0x89;	#define T0_M1_ 0x02
sbit IT1 = 0x8A;	#define T0_CT_ 0x04
sbit IE1 = 0x8B;	#define T0_GATE_ 0x08
sbit TR0 = 0x8C;	#define T1_M0_ 0x10
sbit TF0 = 0x8D;	#define T1_M1_ 0x20
sbit TR1 = 0x8E;	#define T1_CT_ 0x40
sbit TF1 = 0x8F;	#define T1_GATE_ 0x80
	#define T1_MASK_ 0xF0
	#define T0_MASK_ 0x0F

Thanh ghi TCON và các bit có địa chỉ xác định, ví dụ "sbit TF1 = 0x8F" là bit tràn của Timer 1 có địa chỉ là 0x8F. Thanh ghi TMOD không cho phép truy xuất bit, mỗi bit có 1 giá trị tương ứng, ví dụ: bit "T0_CT" được gán có giá trị là 0x04 tương ứng với chế độ được chọn là counter, mặc nhiên không chọn thì hoạt động ở chế độ timer.

IX. LUYỆN TẬP

Bài tập 1: Thiết lập chế độ hoạt động cho Timer 0: là counter, chế độ 1.

Giải:

TMOD = T0_CT + T0_M0_;

Với lệnh này thì gán giá trị cho thanh TMOD bằng :

0x04 + 0x01 = 0x05 (0000 0101).

Bài tập 2: Thiết lập chế độ hoạt động cho Timer 0: là timer, chế độ 2.

Giải:

TMOD = T0_M1_;

Với lệnh này thì gán giá trị cho thanh TMOD bằng:

0x02 (0000 0010).

Bài tập 3: Thiết lập chế độ hoạt động cho Timer 1: là timer, chế độ 1; Timer 0: là counter, chế độ 1.

Giải:

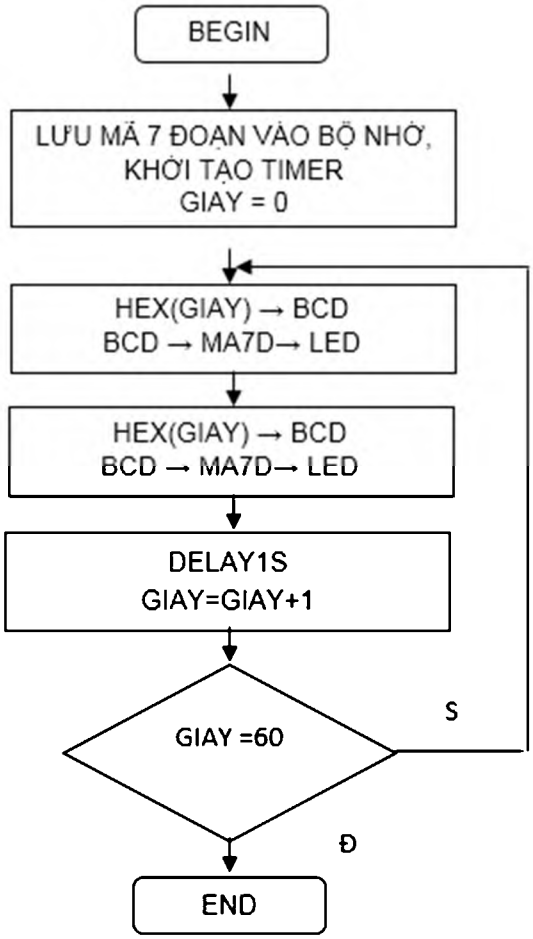
TMOD = T1_M0_+ T0_CT_ + T0_M0_;

Với lệnh này thì gán giá trị cho thanh TMOD bằng:

0x10 + 0x40 + 0x10 = 0x15 (0001 0101).

Bài tập 4: Viết chương trình điều khiển 2 led 7 đoạn đếm GIÂY (từ 00 đến 59) sử dụng Timer 0 để định thời gian.

- ***Lưu đồ***



- ***Chương trình:***

```
#include<AT89X51.H>
unsigned char
MA7D[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
signed int GIAY,CHUC,DONVI;
void DELAY_TIMER0()
{       int BDN;
        TR0 = 1;
        for (BDN=0;BDN<20;BDN++)
                {       do      {}
                        while(TF0==0);
                        TF0=0;TH0=0x3C;TL0=0xB0;
                }
```

```
            TR0=0;
}
void main()
{
      TMOD=T0_M0_;  TH0=0x3C; TL0=0xB0;
      while(1)
                  {
                        for(GIAY=0;GIAY<60;GIAY++)
                        { CHUC=GIAY/10;       DONVI=GIAY%10;
                        P0=MA7D[DONVI];      P1=MA7D[CHUC];
                        DELAY_TIMER0();
                  }
                  }
}
```

- ***Giải thích chương trình:***

Chương trình chính có chức năng khởi tạo Timer 0 hoạt động định thời chế độ 1, giá trị bắt đầu đếm cho Timer là 3CB0H *(tương ứng với 15536)* gán cho cặp thanh ghi TH0 và TL0 để đếm 50000 xung có chu kỳ 1µs, thạch anh sử dụng là 12MHz.

Cho biến **GIAY** chạy từ 0 đến 59, tiến hành tách hàng chục và hàng đơn vị để giải mã và gởi ra 2 port để hiển thị trên 2 led 7 đoạn.

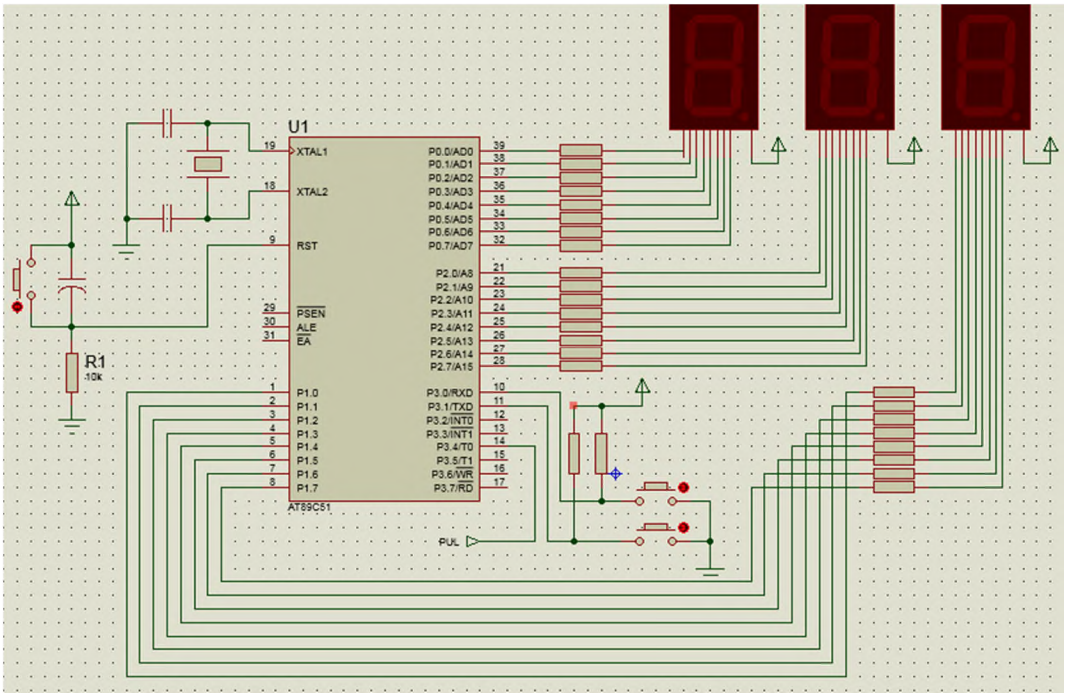
Sau đó gọi chương trình con delay 1s dùng Timer.

Chương trình con delay dùng timer có tên là **"DELAY TIMER0()"** thực hiện các lệnh:

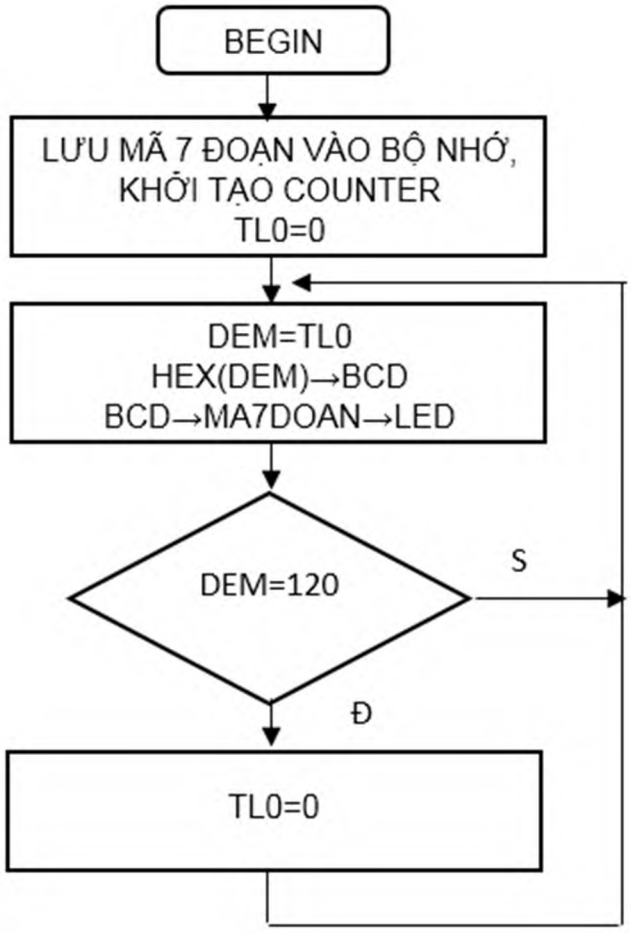
Gán TR0 bằng 1 để Timer bắt đầu đếm, cho biến đếm ngắt chạy từ 0 đến 19 là 20 lần, mỗi lần làm lệnh **do while(TF==0)** – lệnh này kiểm tra cờ tràn TF0 của Timer 0, nếu chưa tràn thì tiếp tục chờ, khi tràn thì tiến hành xoá cờ tràn bằng lệnh **TF0=0** và khởi gán giá trị xuất phát để đếm 50000 xung tiếp theo, cho đến khi BDN bằng 20 thì kết thúc, ngừng Timer và thời gian delay được 1 giây, trở lại chương trình chính để thực hiện tiếp.

Bài tập 5: Viết chương trình dùng vi điều khiển đếm xung ngoại dùng Timer 0, kết quả đếm hiển thị trên 3 led 7 đoạn kết nối trực tiếp với 3 port, giới hạn đếm từ 0 đến 120, khi bằng 120 thì quay về 0.

- ***Sơ đồ mạch***



- ***Lưu đồ***



- ***Chương trình***

```
#include <AT89X51.H>
unsigned char
MA7D[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
int KQD;
unsigned char DVI,CHUC,TRAM;
void GIAIMA_HIENTHI()
{
        DVI=KQD%10;             KQD=KQD/10;
        CHUC=KQD%10; TRAM=KQD/10;
        P0=MA7D[DVI];
        P1=MA7D[CHUC];
        P2=MA7D[TRAM];
```

```
}
void main()
{
        TMOD=0x05;       TR0=1;
        while(1)
                {       KQD=TL0;  GIAIMA_HIENTHI();
                        if(TL0==120)       {TL0=0;}
                }
}
```

BÀI 5: NGẮT
Mã bài: MĐ23-06

Giới thiệu:

Lập trình cho vi điều khiển bằng cách gia lệnh cho vi điều khiển thực hiện 1 danh sách các lệnh cơ bản được sắp xếp theo một trình tự nào đó để có thể hoàn thành một nhiệm vụ đề ra. Việc dừng chương trình đang thực thi để phục vụ cho một chương trình khác khi xảy ra một sự kiện. Chương trình xử lý sự kiện ngắt gọi là chương trình phục vụ ngắt (ISR- Interrupt Service Routine).

Mục tiêu:

- Trình bày tác dung thực tế của một hệ thống được điều khiển bằng tín hiệu ngắt theo nội dung đã học
- Thực hiện tổ chức ngắt và cơ chế thực hiện chương trình phục vụ ngắt của 8051 đúng yêu cầu kỹ thuật.
- Thực hiện tổ chức ngăt đạt yêu cầu kỹ thuật.

Nội dung chính:

I. MỞ ĐẦU

CPU CHỈ THỰC THI ĐƯỢC 1 LỆNH TẠI MỘT THỜI ĐIỂM

Ngắt *(interrupt)* là việc xảy ra một điều kiện *(một sự kiện)* làm cho chương trình đang thực thi *(chương trình chính)* bị tạm dừng để quay sang thực thi một chương trình khác *(chương trình xử lý ngắt)* rồi sau đó quay trở lại để thực thi tiếp chương trình đang bị tạm dừng. Các ngắt đóng vai trò quan trọng trong việc thiết kế và hiện thực các ứng dụng của bộ vi điều khiển. Các ngắt cho phép hệ thống đáp ứng một sự kiện theo cách không đồng bộ và xử lý sự kiện trong khi một chương

trình khác đang thực thi. Một hệ thống được điều khiển bởi ngắt cho ta ảo tưởng nhiều công việc đang được vi xử lý thực hiện đồng thời.

CPU dĩ nhiên không thể thực thi nhiều hơn một lệnh tại một thời điểm nhưng CPU có thể tạm ngưng việc thực thi một chương trình để thực thi một chương trình khác sau đó quay lại thực thi tiếp tục chương trình đang bị tạm ngưng, điều này tương tự như việc CPU rời khỏi chương trình chính để thực thi chương trình con bị gọi để rồi sau đó quay trở về chương trình chính.

Chúng ta phân biệt "***ngắt"*** và "***gọi chương trình con thế nào" ?***

- ***Giống nhau:***

  Khi xảy ra điều kiện tương ứng thì CPU sẽ tạm dừng chương trình chính đang thực thi để thực thi một chương trình khác *(chương trình con/chương trình xử lý ngắt)* rồi sau đó *(sau khi xử lý xong chương trình con/chương trình xử lý ngắt)* thì CPU sẽ quay về để thực thi tiếp chương trình chính đang bị tạm dừng.

- ***Khác nhau:***

	Ngắt	**Chương trình con**
Thời điểm xảy ra sự kiện	Không biết trước *(hay xảy ra không đồng bộ với chương trình chính).*	Biết trước *(hay xảy ra đồng bộ với chương trình chính).*
Nguyên nhân dẫn đến sự kiện	Do các tín hiệu điều khiển từ Timer, Serial port, và bên ngoài chip.	Do lệnh gọi chương trình con.

Chương trình xử lý ngắt *(tức là chương trình mà CPU phải thực hiện khi có một ngắt xảy ra)* được gọi là trình phục vụ ngắt ISR *(Interrupt Service Routine)* hay trình quản lý ngắt. ISR được thực thi nhằm đáp ứng một ngắt và trong trường hợp tổng quát thực hiện việc xuất nhập đối với một thiết bị. Khi một ngắt xuất hiện, việc thực thi chương trình chính tạm thời bị dừng lại và CPU thực thi việc rẽ nhánh đến trình phục vụ ngắt ISR. CPU sẽ thực thi ISR để thực hiện một công việc, sau đó quay về thực hiện chương trình chính.

Một ví dụ điển hình về ngắt là nhập thông số điều khiển sử dụng bàn phím. Chúng ta hãy cùng khảo sát một ứng dụng của ngắt trên lò viba. Chương trình chính có thể điều khiển thành phần công suất của lò để thực hiện **việc nấu nướng**. Tuy nhiên trong khi đang nấu, hệ thống phải đáp ứng việc **nhập số liệu** bằng tay

(chẳng hạn chúng ta muốn yêu cầu rút ngắn hay kéo dài thời gian nấu), điều này có thể xảy ra tại bất cứ thời điểm nào trong quá trình nấu.

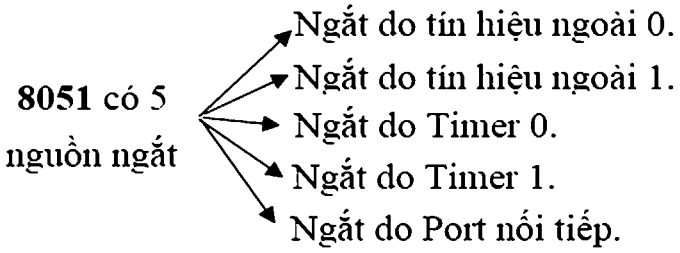
Trường hợp chúng ta không sử dụng ngắt: Như ta đã biết, một hệ thống chỉ có thể thực thi một công việc tại một thời điểm. Cho nên khi hệ thống đang thực thi việc nấu nướng thì nó không thể đáp ứng nhập số liệu và ngược lại. Vì thế trong trường hợp này hệ thống phải thực hiện xong việc nấu nướng rồi mới thực hiện tiếp công việc đáp ứng nhập số liệu *(điều này vô lý, vì khi đã nấu nướng xong thì cần gì phải điều chỉnh thời gian nữa)* hoặc ngược lại hệ thống phải thực hiện cho xong việc đáp ứng nhập số liệu rồi mới thực hiện việc nấu nướng *(điều này cũng vô lý vì không thể biết trước được việc nhập số liệu xảy ra lúc nào, cho nên quá trình hệ thống chờ đợi việc nhập số liệu là vô nghĩa).*

Trường hợp chúng ta sử dụng ngắt: Ta nhận thấy việc nấu nướng là việc diễn ra liên tục từ đầu đến cuối, còn việc đáp ứng nhập số liệu chỉ xảy ra khi ta nhấn bàn phím *(không xác định được thời điểm xảy ra)*. Vì thế ta để chương trình chính điều khiển thành phần công suất của lò để thực hiện **việc nấu nướng,** còn việc đáp ứng nhập số liệu sẽ do **ngắt** điều khiển. Bình thường thì lò thực hiện công việc nấu nướng như đã xác định, khi người sử dụng nhấn bàn phím thì một tín hiệu ngắt được tạo ra và chương trình chính sẽ tạm thời dừng lại. ISR được thực thi để đọc mã phím và thay đổi các điều kiện nấu nướng tương ứng, sau đó kết thúc bằng cách chuyển điều khiển trở về chương trình chính. Chương trình chính được thực thi tiếp từ nơi tạm dừng.

Điều quan trong trong ví dụ chúng ta xét đến bên trên là việc nhập bàn phím xuất hiện không đồng bộ, nghĩa là xuất hiện ở các khoảng thời gian không báo trước hoặc được điều khiển bởi phần phần mềm đang thực thi trong hệ thống. **Đó là một ngắt.**

II. TỔ CHỨC NGẮT CỦA 8051

1. CÁC NGUỒN NGẮT



Chú ý:

- Khi ta reset hệ thống thì tất cả các ngắt đều bị cấm hoạt động.
- Các nguồn ngắt này được cho phép hoặc cấm hoạt động bằng lệnh do người lập trình thiết lập cho từng ngắt.
- Việc xử lý các ngắt được thực hiện qua hai sơ đồ:
  - ✓ Sơ đồ ưu tiên ngắt → có thể thay đổi được và do người lập trình thiết lập.
  - ✓ Sơ đồ chuỗi vòng → cố định không thay đổi được.

Hai sơ đồ này giúp CPU giải quyết các vấn đề liên quan đến ngắt như: hai hay nhiều ngắt xảy ra đồng thời hoặc một ngắt xảy ra khi một ngắt khác đang được thực thi.

Các cờ ngắt của chip 8051:

Loại ngắt	Cờ ngắt	Vị trí của bit trong các thanh ghi
Ngắt ngoài 0	IE0	TCON.1
Ngắt ngoài 1	IE1	TCON.3
Ngắt Timer 0	TF1	TCON.7
Ngắt Timer 1	TF0	TCON.5
Ngắt port nối tiếp	RI	SCON.0
Ngắt port nối tiếp	TI	SCON.1

Chú ý:

- Một ngắt xảy ra thì cờ ngắt tương ứng sẽ được **set bằng 1.**
- Khi IRS của ngắt được thực thi thì cờ ngắt tương ứng sẽ tự động ***bị xoá về 0 bằng phần cứng (****ngoại trừ cờ ngắt RI và TI phải được xoá về 0 bằng phần mềm).*
- Đối với ngắt ngoài sẽ có hai cách kích hoạt để tạo ra một tín hiệu ngắt: ngắt ngoài kích hoạt khi có **mức thấp** và ngắt ngoài kích hoạt khi có **cạnh âm** tại chân INT0 hoặc INT1.

2. QUY ĐỊNH VIỆC CHỌN LOẠI KÍCH HOẠT CHO NGẮT NGOÀI

Việc lựa chọn loại kích hoạt cho các ngắt ngoài, thuộc loại kích hoạt cạnh hay thuộc loại kích hoạt mức, thì được lập trình thông qua các bit IT0 và IT1 của thanh ghi TCON.

IT0=0 → Ngắt ngoài 0 được kích khởi bởi việc phát hiện **mức thấp** tại chân INT0.

IT0=1 → Ngắt ngoài 0 được kích khởi bởi việc phát hiện **cạnh âm** tại chân INT0**.**

IT1=0 → Ngắt ngoài 1 được kích khởi bởi việc phát hiện **mức thấp** tại chân INT1.

IT1=1 → Ngắt ngoài 1 được kích khởi bởi việc phát hiện **cạnh âm** tại chân INT1.

3. THANH GHI CHO PHÉP NGẮT *(IE)*

Thanh ghi cho phép ngắt *(IE: Interrupt Enable):* chứa các bit dùng để cho phép hoặc cấm các hoạt động.

Cấu trúc thanh ghi IE:

EA	-	ET2	ES	ET1	EX1	ET0	EX0

Hoạt động của thanh ghi IE được tóm tắt ở bảng dưới đây:

Bit	Ký hiệu	Chức năng
7	EA	Cho phép hoặc cấm toàn bộ các nguồn ngắt.
6	-	Chưa dùng đến.
5	ET2	Cho phép ngắt Timer 2 *(sử dụng ở 8052).*
4	ES	Cho phép ngắt port nối tiếp.
3	ET1	Cho phép ngắt Timer 1.
2	EX1	Cho phép ngắt ngoài 1.
1	ET0	Cho phép ngắt Timer 0.
0	EX0	Cho phép ngắt ngoài 0.

Chú ý: Cho phép khi bit=1; Cấm khi bit=0.

Hai điều kiện để một ngắt được phép hoạt động là:

- Bit EA=1.
- Bit ngắt tương ứng =1.

4. THANH GHI ƯU TIÊN NGẮT*(IP)*

Khái niệm ưu tiên ngắt giúp 8051 giải quyết vấn đề hai tín hiệu ngắt xuất hiện đồng thời và vấn đề một tín hiệu ngắt xuất hiện trong khi một ngắt khác đang được thực thi.

Thanh ghi ưu tiên ngắt *(IP: Interrupt Priority):* chứa các bit dùng để thiết lập mức độ ưu tiên *(mức cao hay mức thấp)* cho từng ngắt riêng rẽ.

Cấu trúc thanh ghi IP:

-	-	PT2	PS	PT1	PX1	PT0	PX0

Hoạt động của thanh ghi IP được tóm tắt ở bảng dưới đây:

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

Khi hệ thống được thiết lập lại trạng thái ban đầu thì tất cả các ngắt đều sẽ được mặc định ở mức ưu tiên thấp. Ý tưởng các mức ưu tiên cho phép một trình phục vụ ngắt được tạm dừng bởi một ngắt khác nếu ngắt mới này có mức ưu tiên cao hơn mức ưu tiên của ngắt hiện đang được phục vụ. Điều này hoàn toàn hợp lý với 8051 vì ta chỉ có hai mức ưu tiên. Nếu có ngắt có mức ưu tiên cao xuất hiện,

trình phục vụ ngắt cho ngắt có ưu tiên thấp phải tạm dừng (nghĩa là bị ngắt). Ta không thể tạm dừng một chương trình phục vụ ngắt có mức ưu tiên cao.

Chương trình chính do được thực thi ở mức nền và không kết hợp với một ngắt nào nên luôn luôn bị ngắt bởi các ngắt cho dù các ngắt có mức ưu tiên thấp hay mức ưu tiên cao. Nếu có hai ngắt với mức ưu tiên khác nhau xuất hiện đồng thời thì ngắt có mức ưu tiên cao sẽ được phục vụ trước.

5. THỨ TỰ CHUỖI VÒNG NGẮT

Khái niệm chuỗi vòng giúp 8051 giải quyết được vấn đề hai hay nhiều tín hiệu ngắt có mức ưu tiên giống nhau xuất hiện cùng lúc.

Chuỗi vòng này là *(được sắp xếp theo thứ tự từ thấp đến cao):*

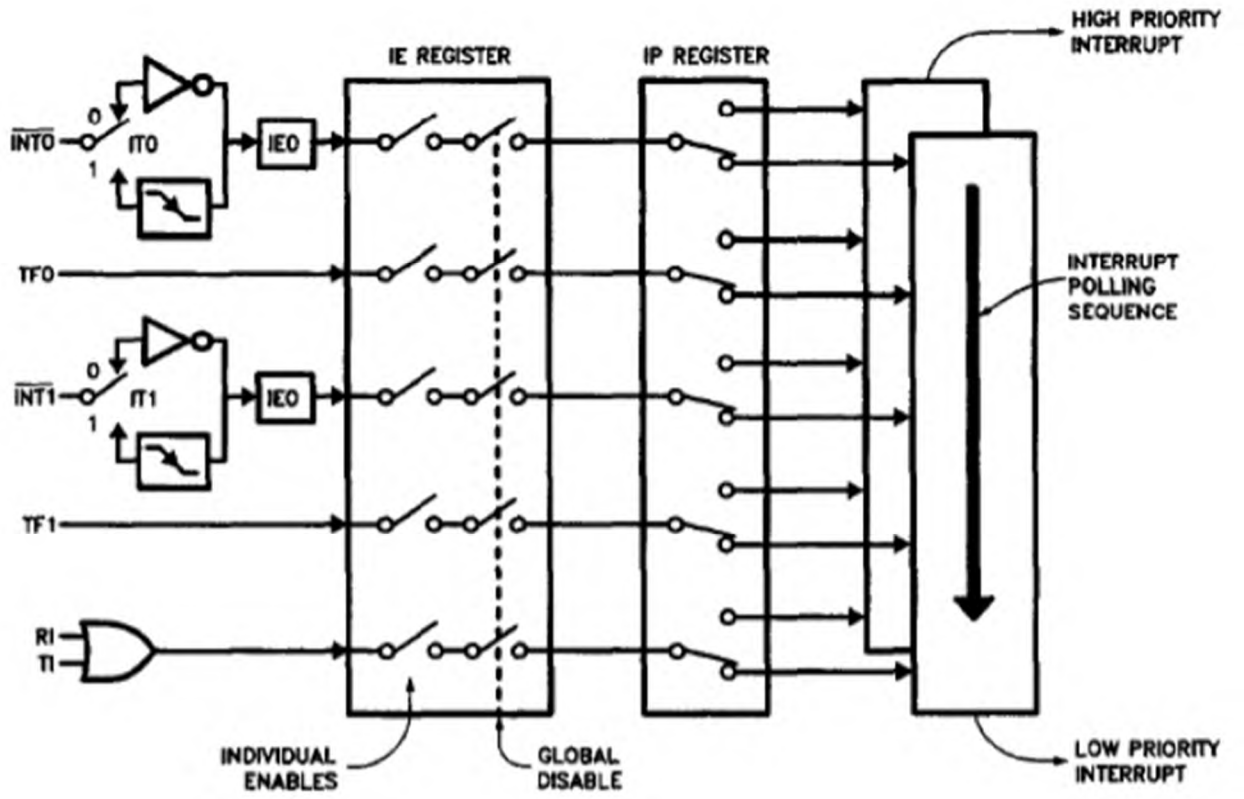
Ngắt ngoài 0 → Ngắt Timer 0 → Ngắt ngoài 1→

Ngắt Timer 1 → Ngắt Port nối tiếp → Ngắt Timer 2 (chỉ có ở 8052)

Hình dưới đây sẽ minh hoạ nguyên nhân ngắt, cơ chế cho phép riêng lẻ và toàn cục, chuỗi vòng và các mức ưu tiên. Trạng thái của tất cả các nguyên nhân ngắt được thể hiện thông qua các bit cờ tương ứng trong các thanh ghi chức năng đặc biệt có liên quan. Dĩ nhiên một ngắt nào đó không được phép, nguyên nhân ngắt tương ứng không thể tạo ra một ngắt nhưng phần mềm vẫn có thể kiểm tra cờ ngắt đó. Lấy ví dụ, bộ định thời và port nối tiếp ở các bài trước sử dụng các cờ ngắt một cách rộng rãi dù không có ngắt tương ứng xảy ra, nghĩa là không sử dụng các ngắt.

Ngắt do port nối tiếp là kết quả OR của cờ ngắt khi thu *(RI – cờ ngắt thu)* và cờ ngắt khi phát *(TI – cờ ngắt phát).*

Cấu trúc ngắt của 8051:



Trong đó:

- IE Register: thanh ghi IE.
- IP Register: thanh ghi IP.
- High Priority Interrupt: ngắt ưu tiên cao.
- Low Priority Interrupt: ngắt ưu tiên thấp.
- Interrupt Polling Interrupt: thứ tự chuỗi vòng ngắt.
- Interrupt enable: cho phép ngắt.
- Global enable: cho phép toàn cục.

III. XỬ LÝ NGẮT VÀ CÁC VECTOR NGẮT

1. QUY TRÌNH XỬ LÝ NGẮT

Các thao tác sẽ xảy ra khi có một ngắt xuất hiện và nó được CPU chấp nhận:

- Hoàn tất thực thi lệnh tại thời điểm đó và dừng chương trình chính.
- Giá trị thanh ghi PC được lưu vào stack *(bộ nhớ ngăn xếp)*.
- Trạng thái ngắt tại thời điểm đó được lưu giữ lại.
- Các yêu cầu ngắt khác sẽ bị ngăn lại.
- Địa chỉ của IRS của ngắt tương ứng được nạp vào thanh ghi PC.
- IRS của ngắt tương ứng được thực thi.

- Sau khi IRS thực thi xong thì giá trị trong stack *(của PC cũ)* được phục hồi vào thanh ghi PC.
- Trạng thái các ngắt được phục hồi lại.
- Chương trình chính tiếp tục tại chỗ bị tạm dừng.

2. CÁC VECTOR NGẮT

Khi một ngắt được chấp nhận, giá trị được nạp cho bộ đếm chương trình PC được gọi là vector ngắt. Vector ngắt là địa chỉ bắt đầu của chương trình phục vụ ngắt *(IRS)* của ngắt tương ứng.

Vector reset hệ thống cũng được xem như là một ngắt: chương trình chính bị ngắt và bộ đếm chương trình PC được nạp giá trị mới.

Khi một trình phục vụ ngắt được trỏ đến, cờ gây ra ngắt sẽ tự động bị xoá về 0 bởi phần cứng. Các ngoại lệ bao gồm cờ RI và TI đối với ngắt do port nối tiếp, các nguyên nhân ngắt thuộc loại này có khả năng tạo ra ngắt nên trong thực tế CPU không xoá cờ ngắt.

Bảng qui định địa chỉ bắt đầu của các IRS *(bảng vector ngắt):*

Loại ngắt	Cờ ngắt	Địa chỉ vector ngắt
Reset hệ thống	RST	0000H
Ngắt ngoài 0	IE0	0003H
Ngắt ngoài Timer 0	IT0	000BH
Ngắt ngoài 1	IE1	0013H
Ngắt Timer 1	IT1	001BH
Ngắt port nối tiếp	RI hoặc TI	0023H
Ngắt Timer 2 *(dành cho 8052)*	TF2 hoặc EXF2	0002BH

IV. THIẾT KẾ CHƯƠNG TRÌNH SỬ DỤNG NGẮT

1. TỔNG QUAN

Khi thiết kế các chương trình không sử dụng ngắt thì ta sẽ gặp phải những trường hợp CPU hoàn toàn tiêu phí thời gian vào việc chờ đợi các tác nhân cần thiết xảy ra *(Ví dụ như: sự tràn cờ của TF0, TF1; Việc thu xong một dữ liệu và cờ*

RI =1, việc phát xong một ký tự và cờ TI =1…) để sau đó mới tiếp tục thực hiện công việc.

⇨ *Điều này không thích hợp cho các ứng dụng điều khiển đòi hỏi phải tác động qua lại với nhiều thiết bị cùng lúc.*

Để giải quyết điều này ta cần thiết kế **các chương trình có sử dụng đến ngắt.**

⇨ *Điều này giúp CPU không tốn thời gian để chờ đợi tác nhân mà chỉ khi nào tác nhân xảy đến thì CPU mới thực hiện việc xử lý tác nhân đó, khoảng thời gian tác nhân không xảy ra thì CPU sẽ làm việc khác.*

1. THIẾT KẾ CÁC CHƯƠNG TRÌNH ISR KÍCH THƯỚC NHỎ

Điều kiện: ***khi các IRS có kích thước không quá 8 byte.***

⇨ IRS phải được viết trong phạm vi điểm nhập tương ứng của nó trong bộ nhớ chương trình *(xem lại phần tổ chức bộ nhớ khi sử dụng ngắt).*

Chú ý:

- Nếu chỉ có một nguyên nhân ngắt được sử dụng thì ISR của nó có thể được viết tràn sang điểm nhập của các ISR khác *(nghĩa là ISR có kích thước lớn hơn 8 byte, nhưng phải nhỏ hơn 64 byte).*
- Nếu có nhiều nguyên nhân ngắt được sử dụng thì ta cần phải cẩn thận để đảm bảo cho các ISR được bắt đầu đúng vị trí nhưng không tràn qua ISR kế *nghĩa là ISR có kích thước không quá 8 byte).*

2. THIẾT KẾ CÁC CHƯƠNG TRÌNH ISR CÓ KÍCH THƯỚC LỚN

Điều kiện: ***khi ISR có kích thước vượt qua 8 byte.***

⇨ Khi các ISR không thể viết vào điểm nhập tương ứng của nó trong bộ nhớ chương trình *(vì kích thước điểm nhập chỉ có 8 byte)* → ta phải chuyển ISR này đến một nơi khác trong bộ nhớ chương trình hoặc có thể viết lấn qua điểm nhập của ISR kế tiếp *(nếu ISR đó không sử dụng).*

2.1 XỬ LÝ NGẮT PORT NỐI TIẾP

Các ngắt do port nối tiếp xuất hiện khi cờ phát ngắt TI hoặc cờ phát thu RI được set bằng 1. Một ngắt phát xuất hiện khi việc phát một ký tự đã ghi vào SBUF hoàn tất. Một ngắt thu xuất hiện khi một ký tự được thu nhận đầy đủ và đang trong SBUF để chờ được đọc. Như vậy ngắt xảy ra khi bộ đệm phát SBUF rỗng, còn ngắt thu xảy ra khi bộ đệm SBUF đầy.

Các ngắt do port nối tiếp có khác với các ngắt do bộ định thời. Cờ ngắt gây ra ở port nối tiếp không được xoá bởi phần cứng khi CPU trỏ đến trình phục vụ ngắt. Lý do là vì ta có hai nguyên nhân tạo ra ngắt ở port nối tiếp, cụ thể là hai ngắt

tạo ra bởi hai cờ TI và RI. Nguyên nhân ngắt phải được xác định trong trình phục vụ ngắt và cờ tạo ra ngắt phải được xoá bằng phần mềm. Cần nhắc lại là với các ngắt do bộ định thời, cờ ngắt tạo ta được xoá bởi phần cứng khi CPU trỏ tới trình phục vụ ngắt.

2.2 XỬ LÝ NGẮT NGOÀI

Ngắt ngoài xảy ra khi có mức thấp hoặc có cạnh âm tác động lên chân INT0 hoặc chân INT1 của 8051. Khi một ngắt ngoài được tạo ra, cờ tạo ra ngắt *(IE0 và IE1)* được xoá bởi phần cứng khi CPU trỏ đến trình phục vụ ngắt nếu ngắt thuộc loại tác động cạnh âm, còn nếu ngắt thuộc loại tác động mức thấp thì nguyên nhân ngắt ngoài sẽ điều khiển mức của cờ thay vì là phần cứng trên chip.

3. KHAI BÁO NGẮT CỦA 8051 TRONG LẬP TRÌNH KEIL-C

Khai báo chương trình con phục vụ ngắt trong Keil giống như khai báo chương trình con nhưng thêm phần thứ tự ngắt và khai báo sử dụng thanh ghi cho chương trình con phục vụ ngắt sử dụng, cú pháp như sau:

Void tên_chương_trình_ngắt() **interrupt n using m**

Trong đó "tên_chương_trình_ngắt()" do chúng ta tự đặt và nên đặt đúng với ngắt đang dùng, **"interrupt n"** với n là số thứ tự các nguồn ngắt như bảng dưới đây:

Thứ tự *interrupt n*	**Mô tả**	**Địa chỉ**
0	Ngắt ngoài 0	0003H
1	Ngắt Timer 0	000BH
2	Ngắt ngoài 1	0013H
3	Ngắt Timer 1	001BH
4	Ngắt Port nối tiếp	0023H
5	Ngắt Timer 2	002BH

Giá trị n của ngắt dùng để tính toán ra địa chỉ của vector ngắt theo công thức:

8*n + 3

Khai báo "**using m"** dùng để sử dụng bank thanh ghi thứ m từ 0 đến 3.

Ví dụ: Khai báo ngắt Timer 0 như sau:

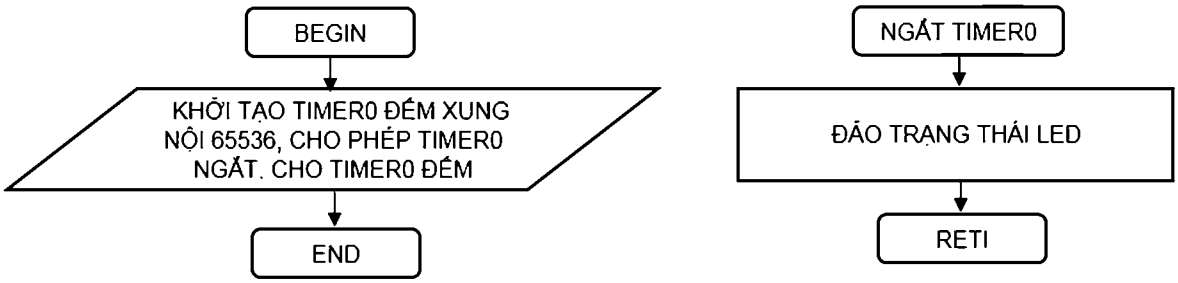
Void timer0_interrupt90 interrupt 1 using 0

Hằng số n = 1 thì vector địa chỉ là: 8*n+3 = 8*1+3 = 11 = 000BH, sử dụng bank thanh ghi thứ 0.

V. *LUYỆN TẬP*

Bài tập 1: Viết chương trình điều khiển 8 led đơn, sử dụng ngắt để tạo delay 65536µs.

- ***Lưu đồ***



- ***Chương trình***

```
#include <AT89X51.H>
void timer0_interrupt()interrupt 1 using 0
{
        P0=~P0;
}
void main()
{       TMOD=T0_M0_; TR0=1;       EA=1;       ET0=1;
        while(1)
                {}
}
```

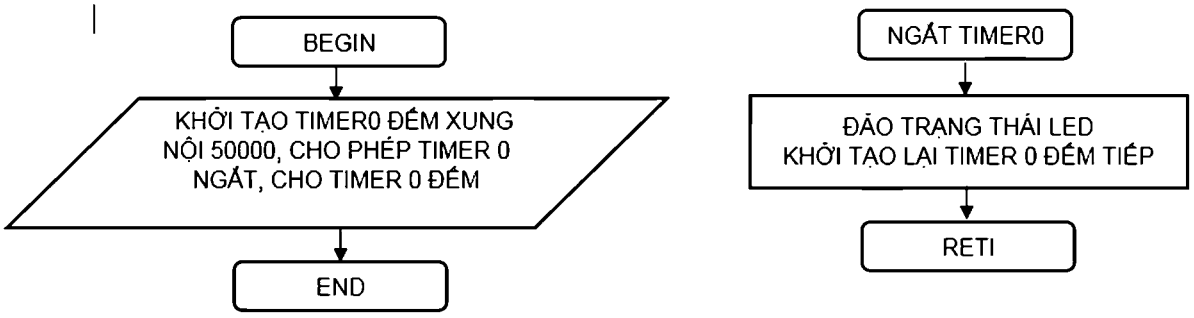
- ***Giải thích chương trình:***

Chương trình chính: có chức năng khai báo sử dụng Timer T0, cho phép Timer 0 ngắt, cho timer bắt đầu đếm. Vòng lặp while không làm gì cả, chỉ chờ ngắt xảy ra.

Chương trình phục vụ ngắt Timer 0: có chức năng khi ngắt xảy ra thì đảo dữ liệu Port 0, thời gian ngắt xảy ra khi Timer 0 đếm được 65536 xung. Nếu sử dụng thạch anh có tần số 12MHz thì sau khi qua bộ chia cho 12 bên trong mạch dao động thì tần số còn lại là 1MHz, chu kỳ cho mỗi xung là 1µs.

Bài tập 2: Viết chương trình điều khiển 8 led đơn sử dụng ngắt để tạo delay 50000µs.

- ***Lưu đồ***



- ***Chương trình***

```
#include <AT89X51.H>
unsigned int DEM;
void timer0_interrupt()interrupt 1 using 0
{
        P0=~P0;
        TL0=DEM; TH0=DEM>>8;
}
void main()
{       TMOD=T0_M0_; TR0=1;             EA=1;       ET0=1;
        DEM=15536;              TL0=DEM; TH0=DEM>>8;
        while(1)
                {}
}
```

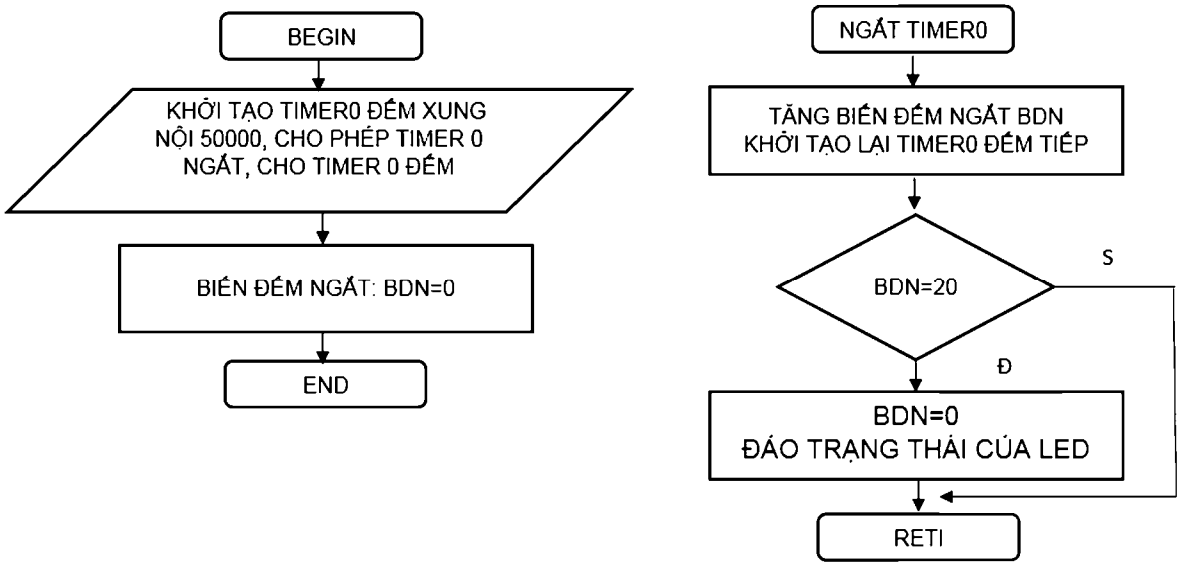
- ***Giải thích chương trình***

Chương trình chính: ngoài các khai báo giống bài trước thì thêm phần gán giá trị cho biến **DEM** bằng 15536 *(do yêu cầu của bài là tạo delay 50000 nên timer sẽ đếm 50000 xung và sau khi đếm xong sẽ phải báo ngắt, nên giá trị xuất phát của timer là:65536 – 50000=15536 chính là giá trị khởi tạo của biến DEM, giá trị này được gán cho 2 thanh ghi TH0 và TL0).*

Chương trình phục vụ ngắt Timer0: có chức năng khi ngắt xảy ra thì đảo dữ liệu Port 0, đồng thời khởi tạo lại giá trị 15536 cho 2 thanh ghi để bắt đầu chu kỳ đếm tiếp theo.

Bài tập 3: Viết chương trình điều khiển 8 led đơn sáng nhấp nháy sử dụng ngắt để tạo delay 1s.

- ***Lưu đồ***



- ***Chương trình***

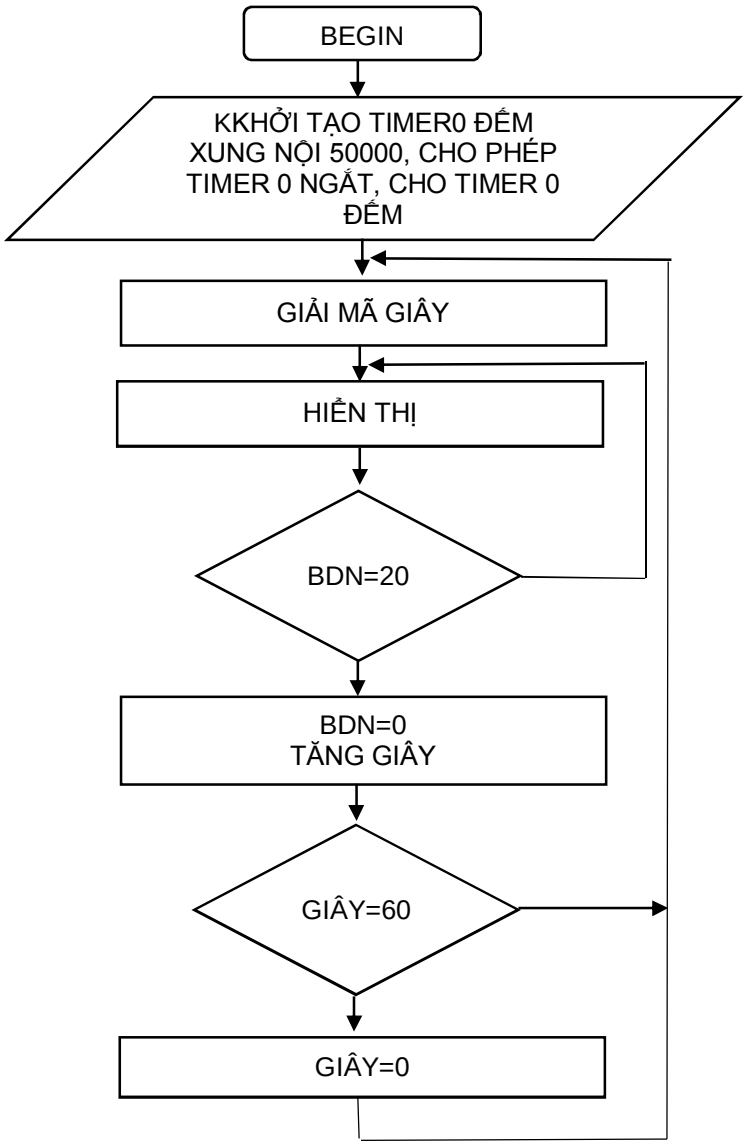
```
#include <AT89X51.H>
unsigned int DEM;
int     BDN;
void timer0_interrupt()interrupt 1 using 0
{
        BDN++;      TL0=DEM; TH0=DEM>>8;
        if(BDN==20)
                {       BDN=0;      P0=~P0;}
}
void main()
{       TMOD=T0_M0_; TR0=1;             EA=1;       ET0=1;
        DEM=15536;              TL0=DEM; TH0=DEM>>8;
        BDN=0;
        while(1)
                {}
}
```

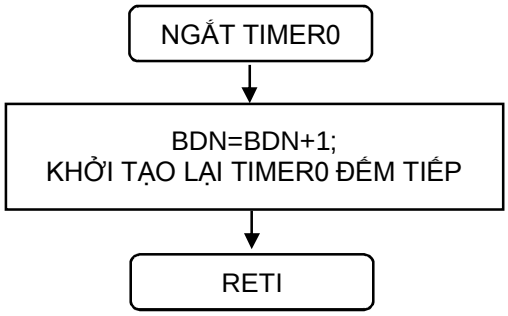
- ***Giải thích chương trình:***

Chương trình phục vụ ngắt Timer0: khi ngắt xảy ra sau mỗi thời gian 50ms *(tương ứng với 50000 xung)* thì tăng BDN; khởi tạo lại giá trị 15536 cho 2 thanh ghi để bắt đầu chu kỳ đếm 50ms tiếp theo. Tiếp theo là kiểm tra BDN xem bằng 20 hay chưa, nếu chưa thì thoát, nếu đúng thì đảo dữ liệu Port 0. Thời gian sáng bằng thời gian tắt bằng 20 lần của 50ms tường đương 1s.

Bài tập 4: Viết chương trình điều khiển 2 led 7 đoạn đếm giây chính xác sử dụng timer 0 để báo ngắt.

- ***Lưu đồ***





Chương trình:

```
#include <AT89X51.H>
unsigned char
MA7D[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
unsigned int DEM;
unsigned char BDN,GIAY,X,Y;
unsigned char MADONVIS,MACHUCS;
void timer0_interrupt()interrupt 1 using 0
{
        BDN++;      TL0=DEM; TH0=DEM>>8;
```

```
}
void delay(unsigned int x)
{unsigned int y;
        for (y=0;y<x;y++)
                {}
}
void HIENTHI()
{
        P2=MADONVIS;  P1_0=0;       delay(200);  P1_0=1;
        P2=MACHUCS;           P1_1=0;       delay(200);  P1_1=1;
}
void GIAIMA()
{
        X=GIAY%10;       Y=GIAY/10;
        MADONVIS=MA7D[X];
        MACHUCS=MA7D[Y];
}
void main()
{       TMOD=T0_M0_;  TR0=1;              EA=1;         ET0=1;
        DEM=15536;              TL0=DEM;  TH0=DEM>>8;
        BDN=0;                  GIAY=0;
        while(1)
                {
                        GIAIMA();
                        if(BDN<20) HIENTHI();
                        else{BDN=BDN-20;
                                        GIAY++;
                                        if(GIAY==60)GIAY=0;}
                }
}
```

- ***Giải thích chương trình:***

Chương trình phục vụ ngắt Timer 0: khi ngắt xảy ra sau thời gian mỗi 50ms thì tăng BDN, khởi tạo lại giá trị 15536 cho 2 thanh ghi để bắt đầu chu kỳ đếm 50ms tiếp theo.

Chương trình chính sau khi tiến hành các lệnh khởi tạo thì giải mã và kiểm tra BDN nếu còn nhỏ hơn 20 thì cho hiển thị, nếu bằng hoặc lớn hơn 20 thì trừ BDN cho 20, tiến hành tăng giây và xử lý giây.

BÀI 6: CỔNG NỐI TIẾP
Mã bài: MĐ23-07

Giới thiệu:

Để truyền thông với máy tính, hoặc với một vi điều khiển khác, trong kết nối này dữ liệu được chuyển 1 bit trong một đơn vị thời gian. Truyền thông nối tiếp chỉ yêu cầu một đường dây cho dữ liệu, đường thứ hai cho nối đất và có thể là đường thứ ba cho xung đồng hồ để đồng bộ. Đồng bộ và không đồng bộ là hai phương thức khác nhau trong việc truyền dữ liệu. Trong truyền đồng bộ thì bộ truyền và bộ thu được đồng bộ hóa qua một đồng hồ bên ngoài, trong khi với truyền không đồng bộ thì bộ truyền và bộ thu được đồng bộ hóa bởi một tín hiệu đặc biệt trên bộ truyền trung gian.

Mục tiêu:

- Trình bày cấu tạo và các chế độ làm việc của cổng truyền thông nối tiếp theo nội dung đã học
- Thực hiện cổng truyền thông nối tiếp đúng yêu cầu kỹ thuật.
- Thực hiện thu phát dữ liệu nối tiếp bằng 8051 đạt yêu cầu kỹ thuật.

Nội dung chính:

I. MỞ ĐẦU

Máy tính truyền dữ liệu theo hai phương pháp: truyền dữ liệu song song và truyền dữ liệu nối tiếp.

- Truyền song song: Sử dụng nhiều dây dẫn để truyền dữ liệu giữa các thiết bị có khoảng cách gần nhau (*khoảng vài mét).* Phương pháp này cho phép truyền dữ liệu với tốc độ cao nhờ sử dụng nhiều dây dẫn để truyền dữ liệu đồng thời nên tại một thời điểm có thể truyền được nhiều bit thông tin nhưng khoảng cách truyền bị hạn chế.
- Truyền nối tiếp: Sử dụng một dây dẫn để truyền dữ liệu *(một dây phát đi và một dây thu về)* giữa các thiết bị có khoảng cách xa nhau *(khoảng vài trăm mét)*. Phương pháp này sẽ truyền dữ liệu với tốc độ chậm hơn so với phương pháp truyền song song vì chỉ sử dụng một dây dẫn để truyền dữ liệu nên tại một thời điểm chỉ có thể truyền được một bit thông tin, nhưng khoảng cách truyền thì không bị hạn chế như ở phương pháp truyền song song.

Trên vi điều khiển, có nhiều kiểu truyền dữ liệu phổ biến như:

- Truyền dữ liệu nối tiếp đồng bộ và bất đồng bộ.
- Truyền dữ liệu giữa các vi điều khiển với các thiết bị ngoại vi.
- Truyền dữ liệu 2 dây.

Vi điều khiển 8051 chỉ có một Port nối tiếp, ở bài học này chúng ta sẽ khảo sát nguyên lý hoạt động truyền dữ liệu đồng bộ và không đồng bộ.

- Ở kiểu truyền dữ liệu nối tiếp đồng bộ thì có 1 đường dữ liệu và 1 đường xung clock, thiết bị nào cấp xung clock thì thiết bị đó đóng vai trò và chủ, thiết bị nhận xung clock đóng vai trò là tớ, tốc độ truyền dữ liệu phụ thuộc vào tần số xung clock. Ví dụ: tần số xung clock là 1MHZ thì tốc độ truyền dữ liệu là 1MBPS – 1M baud.
- Ở kiểu truyền dữ liệu nối tiếp không đồng bộ thì có một đường phát dữ liệu và một đường nhận dữ liệu, không còn tín hiệu xung clock nên gọi là không đồng bộ. Để truyền được dữ liệu thì cả bên phát và bên nhận phải tự tạo xung clock có cùng tần số và thường gọi là tốc độ truyền dữ liệu *(baud).* Ví dụ: 2400baud, 4800baud…2400 có nghĩa là truyền 2400 bit trên 1 giây.

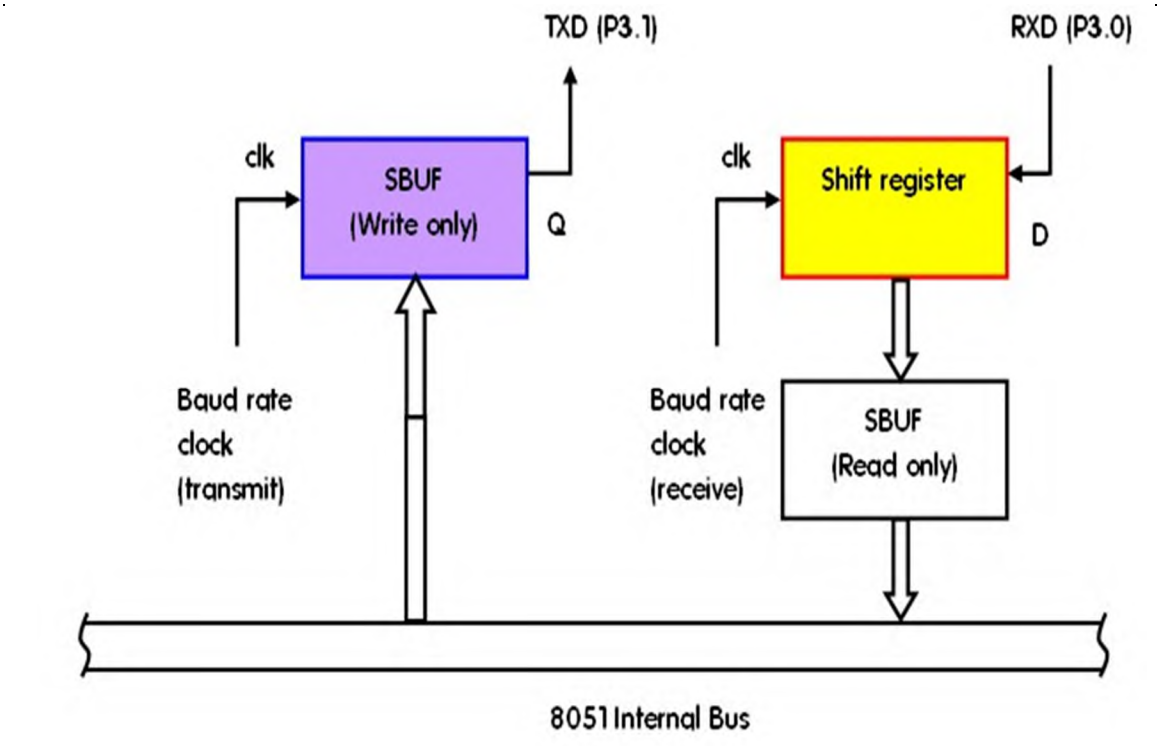
II. THANH GHI ĐỆM PORT NỐI TIẾP (SBUF)

Thanh ghi SBUF *(Serial Buffer Register):* được dùng để lưu giữ liệu cần phát đi và dữ liệu đã nhận được. Việc ghi dữ liệu vào thanh ghi SBUF sẽ nạp dữ liệu để phát đi và việc đọc dữ liệu từ thanh ghi SBUF sẽ truy xuất dữ liệu đã thu được.

Thanh ghi SBUF bao gồm 2 thanh ghi:

- Thanh ghi phát (bộ đệm phát): dùng để lưu giữ dữ liệu cần phát đi.
- Thanh ghi thu (bộ đệm thu): dùng để lưu giữ dữ liệu đã nhận được.

Cấu trúc thanh ghi SBUF:



Trong đó:

- Wrirte only: chỉ ghi.
- Read only: chỉ đọc.
- Shift register: thanh ghi dịch bit.
- Baud rate clock (transmit): xung clock tốc độ baud *(phát).*
- Baud rate clock (receive): xung clock tốc độ baud (*thu).*

III. THANH GHI ĐIỀU KHIỂN PORT NỐI TIẾP (SCON)

Thanh ghi SCON (Serial Control Register): chứa các bit dùng để điều khiển chế độ hoạt động và báo trạng thái port nối tiếp.

Cấu trúc thanh ghi SCON:

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Hoạt động của thanh ghi SCON được tóm tắt trong bảng dưới đây:

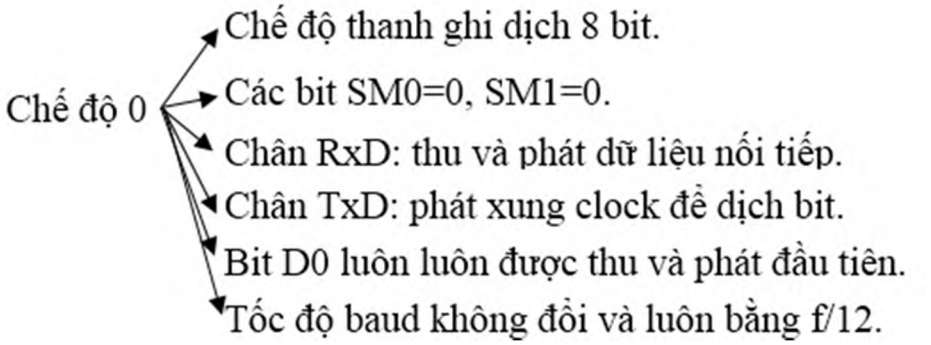
Bit	**Ký hiệu**	**Chức năng**
7	SM0	Bit 0 chọn chế độ của port nối tiếp.
6	SM1	Bit 1 chọn chế độ của port nối tiếp.
5	SM2	Bit 2 chọn chế độ của port nối tiếp. Bit này cho phép truyền thông đa xử lý ở chế độ 2 và 3. Bit RI sẽ không được tích cực nếu bit thứ 9 nhận được là 0.
4	REN	Cho phép thu. Bit này phải được set để nhận các dữ liệu.
3	TB8	Bit thứ 9 được phát *(chế độ 2 và 3).*
2	RB8	Bit thứ 9 nhận được *(chế độ 2 và 3).*
1	TI	Cờ ngắt phát: TI=1 ngay khi kết thức việc phát một dữ liệu, TI được xoá bởi phần mềm.
0	RI	Cờ ngắt thu: RI=1 ngay khi kết thúc việc thu một dữ liệu, RI được xoá bởi phần mềm.

IV. CÁC CHẾ ĐỘ LÀM VIỆC

Các chế độ làm việc của port nối tiếp:

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

1. THANH GHI DỊCH 8 BIT *(CHẾ ĐỘ 0)*



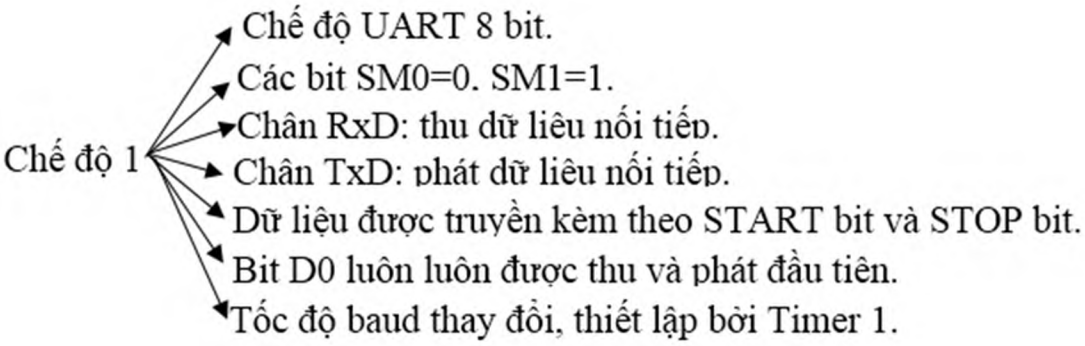
Quá trình phát dữ liệu: Ghi dữ liệu cần phát vào SBUF → Việc phát dữ liệu bắt đầu: Dữ liệu từ SBUF được dịch ra ngoài chân RxD đồng thời với các xung clock dịch bit được gởi ra chân TxD *(mỗi bit được truyền đi trên chân RxD trong 1 chu kỳ máy).*

Quá trình thu dữ liệu: Set bit cho phép thu *(REN=1)* → Xoá cờ ngắt thu *(RI=0)* → Việc thu dữ liệu bắt đầu: các xung clock dịch bit gởi ra chân TxD và dữ liệu từ thiết bị bên ngoài được dịch vào chân RxD bởi các xung clock dịch bit này *(việc dich dữ liệu vào chân RxD xảy ra ở cạnh lên của xung clock dịch bit).*

2. CHẾ ĐỘ UART 8 BIT CÓ TỐC ĐỘ BAUD THAY ĐỔI (*CHẾ ĐỘ 1)*

Trong chế độ 1, port nối tiếp của 8051 hoạt động như một bộ thu phát không đồng bộ 8 bit có tốc độ baud thay đổi.

UART là một bộ thu phát dữ liệu nối tiếp với mỗi ký tự dữ liệu được đứng trước bởi một bit START *(logic 0)* và được đứng sau bởi một bit STOP *(logic 1).* Thỉnh thoảng, một bit chẵn lẻ *(Parity bit)* được chèn vào giữa bit dữ liệu sau cùng và bit STOP. Hoạt động chủ yếu của UART là biến đổi dữ liệu phát từ song song thành nối tiếp và biến đổi dữ liệu thu nối tiếp thành song song.



Quá trình phát dữ liệu: Ghi dữ liệu cần phát vào SBUF → Việc phát dữ liệu bắt đầu: dữ liệu từ SBUF được dịch ra chân TxD *(theo thứ tự: Start bit → 8 bit data (D0…D7) → Stop bit)* → Cờ TI=1.

- ✓ Tốc độ baud: do người lập trình thiết lập và được qui định bởi tốc độ tràn của Timer 1.
- ✓ Thời gian 1 bit trên đường truyền: bằng nghịch đảo của tốc độ baud.
- ✓ Cờ ngắt phát TI=1: khi bit Stop xuất hiện trên chân TxD.

Quá trình thu dữ liệu: Một sự chuyển trạng thái từ mức 1 xuống mức 0 tại chân RxD *(xuất hiện bit Start)* → Việc thu dữ liệu bắt đầu: 8 bit dữ liệu được dịch vào trong SBUF *(theo thứ tự từ D0…D7)* → Stop bit *(bit thứ 9)* được đưa vào bit RB8 *(thuộc thanh ghi SCON)* → Cờ ngắt RI =1.

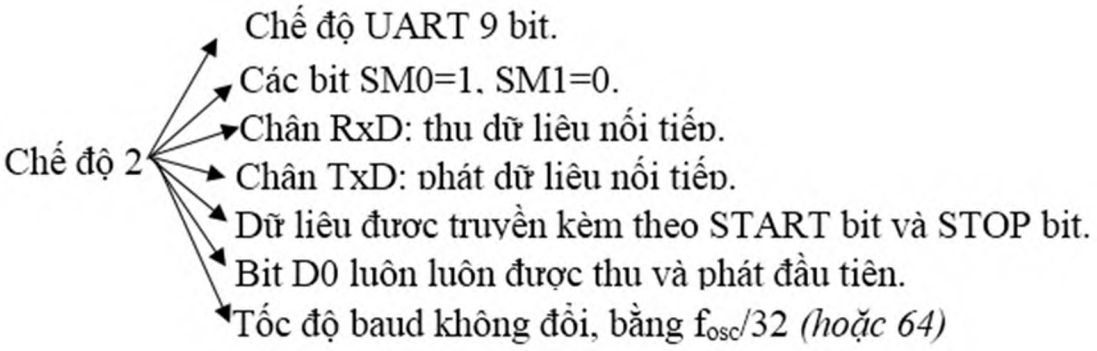
- ✓ Tốc độ baud: do người lập trình thiết lập và được qui định bởi tốc độ tràn của Timer 1.
- ✓ Hai điều kiện bắt buộc để thực hiện quá trình thu dữ liệu như trên:
  - ➢ RI =0 → Điều này nghĩa là chip 8051 đã đọc xong dữ liệu trước đó và xoá cờ RI.
  - ➢ SM=1 và Stop bit =1 hoặc SM2 =0 → chỉ áp dụng trong chế độ truyền thông đa xử lý. Yêu cầu này nghĩa là không set cờ RI

bằng 1 trong chế độ truyền thông đa xử lý khi bit dữ liệu thứ 9 là 0.

- Cờ ngắt thu RI=1: khi 8 bit dữ liệu đã được nạp vào SBUF.

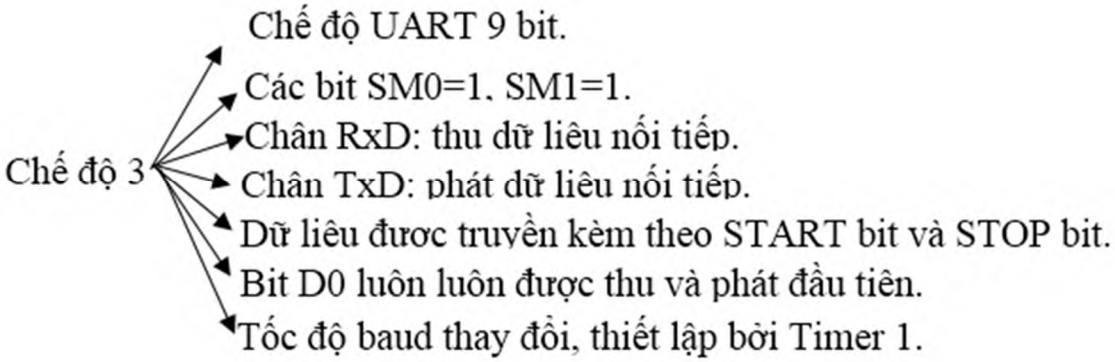
3. UART 9 BIT VỚI TỐC ĐỘ BAUD CỐ ĐỊNH (*Chế độ 2)*

Chế độ này tương tự như UART 8 bit, chỉ khác số bit dữ liệu là 9 bit).



4. UART 9 BIT VỚI TỐC ĐỘ BAUD THAY ĐỔI *(Chế độ 3)*

Chế độ này tương tự như UART 9 bit, chỉ khác tốc độ baud có thể thay đổi.



V. *KHỞI TẠO VÀ TRUY SUẤT THANH GHI PORT NỐI TIẾP*

1. BIT CHO PHÉP THU *(NHẬN)* DỮ LIỆU *(REN)*

Công dụng: dùng để cho phép *(hoặc không cho phép)* nhận các ký tự dữ liệu.

- REN=1: cho phép nhận dữ liệu.
- REN=0: không cho phép nhận dữ liệu.

2. BIT DỮ LIỆU THỨ 9

Công dụng: tuỳ thuộc vào đặc tính kỹ thuật của thiết bị nối tiếp mà có thể yêu cầu hoặc không yêu cầu bit dữ liệu thứ 9.

Khi phát dữ liệu: bit dữ liệu thứ 9 phải được nạp vào bit TB8 của SCON trước khi phát đi.

Khi thu dữ liệu: bit dữ liệu thứ 9 sẽ được nạp vào bit RB8 của SCON sau khi thu xong.

3. BIT KIỂM TRA CHẴN LẺ

Công dụng: trong chip 8051 thì bit Parity được dùng để thiết lập việc kiểm tra chẵn cho 8 bit dữ liệu chứa trong thanh ghi A *(thường dùng để kiểm tra lỗi khi truyền dữ liệu).*

- P=1 : số lượng bit 1 trong thanh ghi A là số lẻ.
- P=0: số lượng bit 1 trong thanh ghi A là số chẵn hoặc số lượng bit 1 trong thanh ghi A và bit P là một số chẵn.

Ở chế độ 1*(UART 8 bit)* thì bit chẵn/lẻ do chip 8051 tạo ra có thể được thêm vào tại bit thứ 8 *(vị trí D7)* và khi đó ta chỉ có thể truyền dữ liệu chỉ có 7 bit.

Ở chế độ 2,3 (*UART 9 bit)* thì bit chẵn/lẻ do chip 8051 tạo ra có thể thêm vào tại bit thứ 9 *(nghĩa là thêm vào bit TB8 của SCON)* và khi đó ta có thể truyền dữ liệu có 8 bit.

4. CÁC CỜ NGẮT CỦA PORT NỐI TIẾP

Có hai cờ ngắt của Port nối tiếp:

- RI: cờ ngắt thu. RI=1 khi kết thúc việc thu dữ liệu và chỉ ra rằng bộ đệm thu đã đầy.
- TI: cờ ngắt phát. TI=1 khi kết thúc việc phát dữ liệu và chỉ ra rằng bộ đệm phát đã rỗng.

Thông qua việc kiểm tra cở ngắt TI ta có thể biết được chip 8051 đã sẵn sàng để truyền 1 byte dữ liệu hay chưa. Cần chú ý rằng, cờ TI được đặt *(TI=1)* khi 8051 đã hoàn tất việc truyền 1 byte dữ liệu, còn được xoá *(TI=0)* thì phải do người lập trình thực hiện bằng lệnh. Cũng nên nhớ, nếu ghi 1 byte vào thanh ghi SBUF trước khi cờ TI được đặt thì sẽ có nguy cơ bị mất phần dữ liệu trước đó chưa kịp truyền đi.

Thông qua việc kiểm tra cờ ngắt RI ta có thể biết được chip 8051 đã sẵn sàng để nhận một byte dữ liệu hay chưa. Cần chú ý rằng, cờ RI được đặt *(RI=1)* khi 8051 đã hoàn tất việc nhận 1 byte dữ liệu, còn được xoá *(RI=0)* thì phải do người lập trình thực hiện bằng lệnh. Cũng nên nhớ, nếu không tiến hành cất nội dung của thanh ghi SBUF vào nơi an toàn thì sẽ có nguy cơ bị mất dữ liệu vừa nhận được do dữ liệu tiếp theo được chuyển vào.

VI. TRUYỀN THÔNG ĐA XỬ LÝ

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

Khi bộ vi xử lý chủ *(master)* muốn truyền đi một khối dữ liệu đến một trong nhiều bộ vi xử lý tớ *(slave)*, trước tiên bộ vi xử lý chủ phát đi một byte địa chỉ nhận dạng bộ vi xử lý tớ đích. Một byte địa chỉ khác với một byte dữ liệu ở chỗ bi thứ 9 là 1 *(đối với byte địa chỉ)* và là 0 *(đối với byte dữ liệu)*. Một byte địa chỉ ngắt tất cả các bộ vi xử lý tớ để cho mỗi một bộ vi xử lý tớ có thể khảo sát byte nhận được để kiểm tra xem có phải là bộ vi xử lý tớ được định địa chỉ không. Bộ vi xử lý tớ được định địa chỉ sẽ xoá bit SM2 của mình và chuẩn bị nhận các byte dữ liệu theo sau. Các bộ vi xử lý tớ không được định địa chỉ có các bit SM2 được set lên bằng 1 và thực thi các công việc của riêng chúng, bỏ qua không nhận byte dữ liệu. Các bộ vi xử lý này sẽ được ngắt một lần nữa khi bộ vi xử lý chủ phát tiếp byte địa chỉ kế.

SM2 không ảnh hưởng đến chế độ 0, và trong chế độ 1 thì bit này có thể được dùng để kiểm tra sự hợp lệ của bit Stop. Ở chế độ 1 thu, nếu SM2=1, ngắt thu sẽ không được tích cực trừ khi bit Stop thu được là hợp lệ.

VII. TỐC ĐỘ BAUD

1. TỐC ĐỘ BAUD CHO CHẾ ĐỘ 0

Baud rate = f_{osc} /12

- Baud rate: tốc độ baud.
- f_{osc} : tần số mạch dao động trên chip.

2. TỐC ĐỘ BAUD CHO CHẾ ĐỘ 1,3

Với SMOD=1:

Baud rate = Timer 1 overflow rate / 16

- Timer 1 overflow rate: thời gian thiết lập tràn của Timer 1.

Với SMOD=0:

Baud rate = Timer 1 overflow rate / 32

- Timer 1 overflow rate: thời gian thiết lập tràn của Timer 1

3. TỐC ĐỘ BAUD CHO CHẾ ĐỘ 2

Với SMOD=1:

Baud rate = f_{osc} / 32

- f_{osc} : tần số mạch dao động trên chip.

Với SMOD=0:

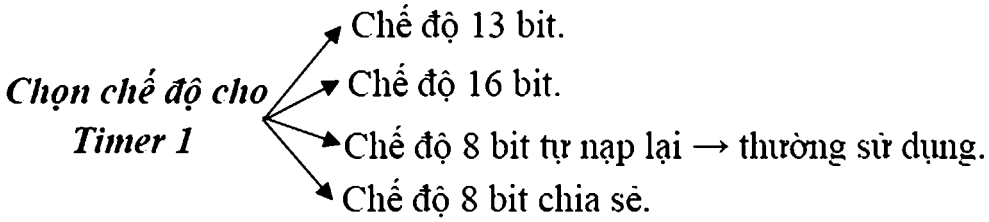
Baud rate = f_{osc} / 64

- f_{osc} : tần số mạch dao động trên chip.

Chú ý: Sau khi reset thì bit SMOD = 0 *(chế độ mặc định).*

4. SỬ DỤNG TIMER 1 LÀM XUNG CLOCK TỐC ĐỘ BAUD CHO PORT NỐI TIẾP

Kỹ thuật tạo xung clock tốc độ baud bằng Timer 1:



Nạp giá trị thích hợp vào thanh ghi TH1 để có tốc độ tràn đúng → tạo tốc độ baud cho port nối tiếp.

Về chọn chế độ: thường dùng Timer 1 ở chế độ 8 bit tự động nạp lại. Các tốc độ baud rất chậm có thể dùng chế độ 16 bit.

Bảng tóm tắt tốc độ baud ứng với hai loại thạch anh 12MHz và 11,059MHz:

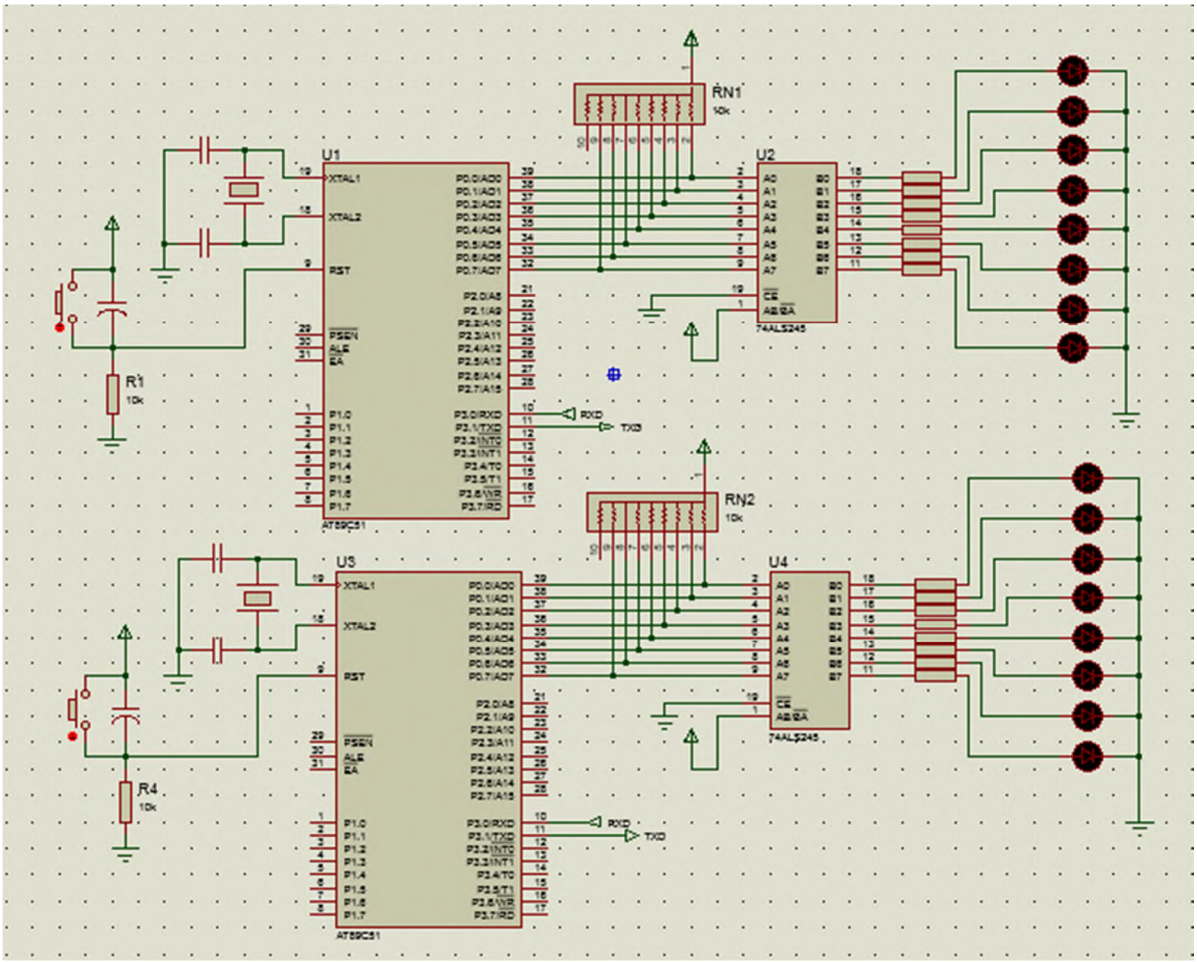
Tốc độ baud	Tần số thạch anh	SMOD	Giá trị nạp cho TH1	Tốc độ baud thực tế	Sai số
9600	**12MHz**	**1**	**- 7 (F9H)**	**8923**	**7%**
2400	**12MHz**	**0**	**-13 (F3H)**	**2404**	**0,16%**
1200	**12MHz**	**0**	**-26 (E6H)**	**1202**	**~0%**

19200	11,059MHz	1	-3 (FDH)	19200	0%
9600	11,059MHz	0	-3 (FDH)	9600	0%
2400	11,059MHz	0	-12 (F4H)	2400	0%
1200	11,059MHz	0	-24 (E8H)	1200	0%

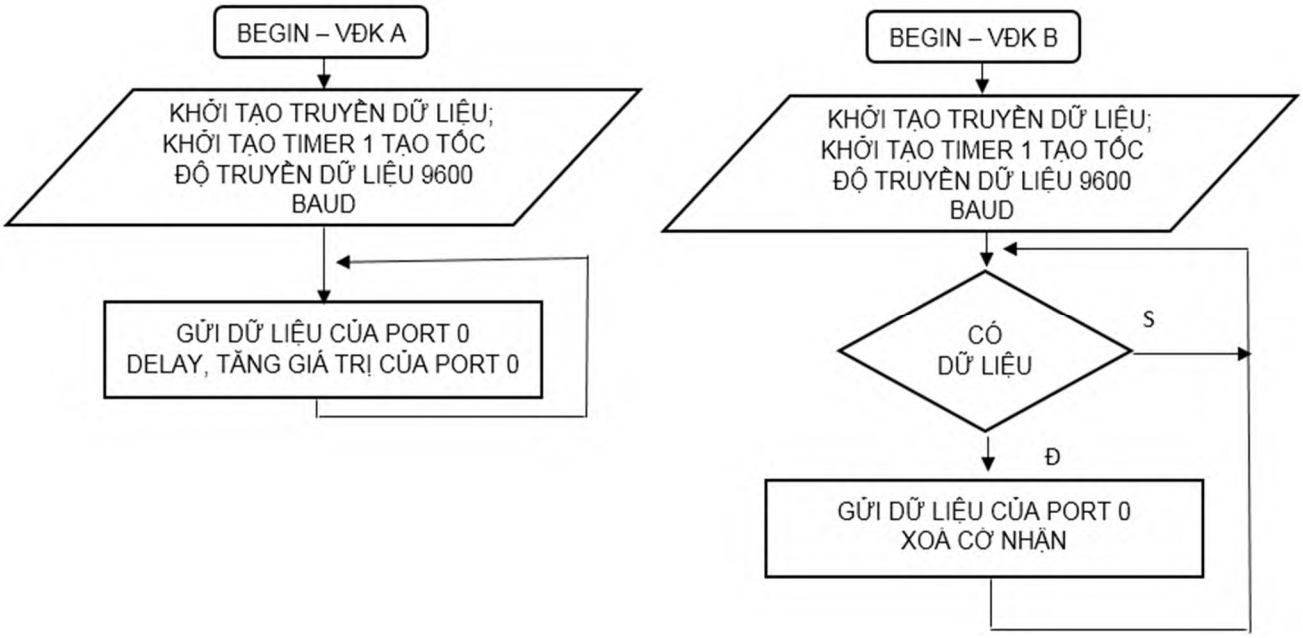
VIII. LUYỆN TẬP

Bài tập : Một hệ thống dùng hai vi điều khiển AT89S52 truyền dữ liệu nối tiếp UART. Vi điều khiển A dùng Port 0 điều khiển 8 led đơn đếm nhị phân đồng thời gửi dữ liệu của Port 0 sang vi điều khiển B để gửi ra Port 0. Tốc độ truyền dữ liệu là 9600baud, dùng thạch anh 11,059MHz. Viết chương trình đáp ứng yêu cầu bài toán.

- ***Sơ đồ mạch***



- ***Lưu đồ***



Chương trình vi điều khiển A:

```
#include <AT89X51.H>
void delay(unsigned int x)
{       unsigned int y;
            for(y=0;y<x;y++)  {;}
}
void main()
{       SCON=0x50;
        TMOD=T1_M1_;  TH1=-3;      TR1=1;
        P0=0x00;
        while(1)
            {       SBUF=P0;
                    do{;}          while(TI==0);
                    TI=0; delay(10000);       P0++;
            }
}
```

- ***Chương trình vi điều khiển B***

```
#include <AT89X51.H>
void main()
{       SCON=0x50;
        TMOD=T1_M1_;  TH1=-3;      TR1=1;
        while(1)
                {
                        do{}
                        while(RI==0);
                        P0=SBUF;   RI=0;
                }
}
```

- ***Giải thích chương trình***

Ở VĐK A: Chương trình chính có chức năng khởi tạo truyền dữ liệu chế độ 1, cho phép nhận dữ liệu, xoá các cờ phát và nhận ("**SCON=0x50**;". Thiết lập Timer 1 hoạt động ở chế độ 2 để tạo tốc độ tràn phục vụ cho việc truyền dữ liệu. Lệnh khởi tạo giá trị đếm 3 xung là tràn để thiết lập tốc độ 9600baud và cho phép Timer 1 hoạt động ("**TMOD=T1_M1_; TH1=-3; TR1=1**;")

Lệnh kiểm tra TI bằng vòng lặp do while, nếu TI lên 1 thì thoát, tiến hành xoá TI, delay và tăng giá trị Port 0 rồi lặp lại.

Ở VĐK B: Chương trình chính ở VĐK B có chức năng khởi tạo truyền dữ liệu và Timer giống ở VĐK A. Sau khi khởi tạo xong thì tiến hành kiểm tra cờ nhận dữ liệu RI có bằng 1 hay không? Nếu bằng 0 thì không có dữ liệu gởi đến, nếu bằng 1 thì đã có dữ liệu, tiến hành xoá cờ nhận để báo hiệu cho lần sau, tiến hành nhận dữ liệu rồi gởi ra Port 0 và quay trở lại kiểm tra để nhận byte tiếp theo.