TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHIỆP
KHOA ĐIỆN TỬ
BỘ MÔN KỸ THUẬT MÁY TÍNH

BÀI GIẢNG PHÁT CHO SINH VIÊN

(LƯU HÀNH NỘI BỘ)

Theo chương trình 150 TC thay 180 TC hoặc tương đương

Sử dụng cho năm học 2011 – 2012

Tên bài giảng: Vi xử lý – Vi điều khiển

Số tín chỉ: 03

BÀI GIẢNG PHÁT CHO SINH VIÊN

(LƯU HÀNH NỘI BỘ)

Theo chương trình 150 TC thay 180 TC hoặc tương đương

Sử dụng cho năm học 2011 – 2012

Tên bài giảng: Vi xử lý – Vi điều khiển

Số tín chỉ: 03

Thái Nguyên, ngày 01 tháng 07 năm 2011

Trưởng bộ môn | Trưởng khoa Điện Tử

Ths. Nguyễn Tuấn Linh | PGS. TS. Nguyễn Hữu Công

MỤC LỤC

ĐẠI HỌC THÁI NGUYÊN
TRƯỜNG ĐẠI HỌC
KỸ THUẬT CÔNG NGHIỆP

CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

CHƯƠNG TRÌNH GIÁO DỤC ĐẠI HỌC

NGÀNH ĐÀO TẠO: ĐIỆN – ĐIỆN TỬ, SPKT ĐIỆN – TIN, CƠ ĐIỆN TỬ
CHUYÊN NGÀNH: KHỐI NGÀNH ĐIỆN – ĐIỆN TỬ

ĐỀ CƯƠNG CHI TIẾT HỌC PHẦN:

VI XỬ LÝ – VI ĐIỀU KHIỂN

(Học phần bắt buộc)

1. Tên học phần: Vi xử lý – vi điều khiển.
2 . Số tín chỉ: 03; 3(3; 1,5; 6)/12
3. Trình độ cho sinh viên năm thứ: 3 (Điện, Điện tử, SPKT Điện, SPKT Tin) hoặc 4 (Cơ điện tử).
4. Phân bổ thời gian

- Lên lớp lý thuyết: 3 (tiết/tuần) x 12 (tuần) = 36 tiết.
- Thảo luận: 1,5 (tiết/tuần) x 12 (tuần) = 18 tiết.

5. Các học phần học trước

Kỹ thuật điện tử số.

6. Học phần thay thế, học phần tương đương

Vi xử lý – vi điều khiển (trong các chương trình 180 TC và 260 ĐVHT)

7. Mục tiêu của học phần

Sau khi học xong học phần sinh viên phải nắm được cấu trúc phần cứng của các bộ vi xử lý – vi điều khiển tiêu biểu: x86, 8051; Tổ chức bộ nhớ, tập lệnh, chế độ địa chỉ và lập trình cho chúng; Biết cách ghép nối với bộ nhớ và thiết bị ngoại vi; Biết khai thác khả năng ngắt và định thời. Có khả năng thiết kế và xây dựng modul (bao gồm cả phần cứng và phần mềm) sử dụng vi điều khiển cho bài toán cụ thể.

8. Mô tả vắn tắt nội dung học phần

Tổng quan về các hệ đếm và biểu diễn thông tin trong các hệ vi xử lý – vi điều khiển. Vi xử lý: Tổng quan về kiến trúc hệ vi xử lý; tổ chức phần cứng của CPU họ Intel 80x86, các chế độ đánh địa chỉ, tập lệnh, lập trình hợp ngữ (assembly) cho 80x86 với những bài toán đơn giản; một số vi mạch phụ trợ trong hệ vi xử lý. Vi điều khiển: Cấu trúc hệ vi điều khiển onchip MCS 8051; lập trình hợp ngữ cho vi điều khiển; hoạt động định thời, ngắt và truyền thông nối tiếp; giới thiệu một số họ vi xử lý thông dụng khác. Giới thiệu một số bài toán ứng dụng tiêu biểu.

9. Nhiệm vụ của sinh viên

1. Dự lớp ≥ 80 % tổng số thời lượng của học phần.
2. Chuẩn bị thảo luận.
3. Bài tập, Bài tập lớn (dài): Không

10. Tài liệu học tập

- **Sách, giáo trình chính:**

[1] Bài giảng ***"Vi xử lý – vi điều khiển"***

- **Sách tham khảo:**

[1] Văn Thế Minh, *Kỹ thuật vi xử lý*, NXB KHKT, 1997.

[2] Tống Văn On, *Họ vi điều khiển 8051*, NXB KH&KT, 2005.

[3] Nguyễn Tăng Cường, Phan Quốc Thắng, *Cấu trúc và lập trình họ vi điều khiển 8051*, NXB KH&KT, 2004.

[4] Michael Hordeski, *Personal Computer Interfaces*, Mc. Graw Hill, 1995.

[5] http://picat.dieukhien.net

11. Tiêu chuẩn đánh giá sinh viên và thang điểm

11.1. Các học phần lý thuyết

- **Tiêu chuẩn đánh giá**

1. Chuyên cần;
2. Thảo luận, bài tập;
3. Bài tập lớn (dài);
4. Kiểm tra giữa học phần;
5. Thi kết thúc học phần;
6. Khác.

- **Thang điểm**

- Điểm đánh giá bộ phận chấm theo thang điểm 10 với trọng số như sau:
  + Kiểm tra giữa học phần: 20 %.
  + Điểm thi kết thúc học phần: 80 %.

12. Nội dung chi tiết học phần và lịch trình giảng dạy

Người biên soạn: ThS. Nguyễn Tuấn Anh
ThS. Nguyễn Tuấn Linh
ThS. Nguyễn Văn Huy
Th.S Tăng Cẩm Nhung
Th.S Phùng Thị Thu Hiền
ThS. Nguyễn Tiến Duy

Tuần thứ	Nội dung	Tài liệu học tập, tham khảo	Hình thức học
1	**Chương I: Tổng quan về vi xử lý – vi điều khiển** **1.1. Giới thiệu chung về vi xử lý – vi điều khiển** 1.1.1. Tổng quan 1.1.2. Lịch sử phát triển của các bộ xử lý 1.1.3. Vi xử lý và vi điều khiển **1.2. Cấu trúc chung của hệ vi xử lý** 1.2.1. Khối xử lý trung tâm (CPU) 1.2.2. Bộ nhớ (Memory) 1.2.3. Khối phối ghép vào/ra (I/O) 1.2.4. Hệ thống bus **1.3. Định dạng dữ liệu và biểu diễn thông tin trong hệ vi xử lý – vi điều khiển** 1.3.1. Các hệ đếm 1.3.2. Biểu diễn số và ký tự 1.3.3. Các phép toán số học trên hệ đếm nhị phân	[1] - [4]	Giảng
2	**Chương II: Họ vi xử lý Intel 80x86** **2.1. Cấu trúc phần cứng của bộ vi xử lý 8086** 2.1.1. Tổng quan **2.1.2.** Cấu trúc bên trong và sự hoạt động **2.1.3.** Các chế độ địa chỉ	[1] - [4]	Giảng
3	**2.2. Tập lệnh** 2.2.1. Giới thiệu chung 2.2.2. Các nhóm lệnh **2.3. Biểu đồ thời gian ghi/đọc**	[1] - [4]	Giảng
4	**2.4. Lập trình hợp ngữ (Assembly) cho vi xử lý 80x86** 2.4.1. Giới thiệu chung về hợp ngữ 2.4.2. Cấu trúc của chương trình hợp ngữ	[1] - [4]	Giảng

	2.4.3. Các cấu trúc điều khiển cơ bản 2.4.4. Các bước khi lập trình 2.4.5. Các bài tập ví dụ		
5	**Thảo luận**		
6	**Chương III: Hệ vi điều khiển onchip MCS 8051** **3.1. Giới thiệu chung về vi điều khiển** 3.1.1. Giới thiệu chung 3.1.2. Khái niệm vi điều khiển 3.1.3. Cấu trúc chung của vi điều khiển 3.2. **Kiến trúc vi điều khiển 8051**	[1] - [4]	Giảng
7	**Kiến trúc vi điều khiển 8051 (tiếp)**	[1] - [4]	Giảng
8	**Kiểm tra giữa kỳ**		
9	3.3. Tập lệnh 8051 và lập trình hợp ngữ cho 8051 3.3.1. Tập lệnh 8051 3.3.2. Thành phần ngôn ngữ assembly	[1] - [4]	Giảng
10	3.4. Kiến trúc vi điều khiển 8051	[1] - [4]	Giảng
11	**Thảo luận**	[1] - [4]	Thảo luận
12	Kiến trúc vi điều khiển 8051 (tiếp)	[1] - [4]	Giảng
13	**Chương IV: Ứng dụng** Thảo luận	[1] - [4]	Thảo luận
14	**Chương V: Các hệ VĐK tiên tiến**	[1] - [4]	Giảng
15	Thảo luận	[1] - [4]	Thảo luận

CHƯƠNG 1.
TỔNG QUAN VỀ VI XỬ LÝ – VI ĐIỀU KHIỂN

Mục tiêu:

Giúp sinh viên hiểu về lịch sử ra đời của hệ vi xử lý – vi điều khiển; khái niệm, cấu tạo và nguyên lý của hệ vi xử lý – vi điều khiển; ôn lại kiến thức về các hệ thống số đếm.

Tóm tắt chương:

Chương chia làm 3 phần:

Giới thiệu chung về vi xử lý – vi điều khiển

Tổng quan

Lịch sử phát triển của các bộ xử lý

Vi xử lý và vi điều khiển

Cấu trúc chung của hệ vi xử lý

Khối xử lý trung tâm (CPU)

Bộ nhớ (Memory)

Khối phối ghép vào/ra (I/O)

Hệ thống bus

Định dạng dữ liệu và biểu diễn thông tin trong hệ vi xử lý – vi điều khiển

Các hệ đếm

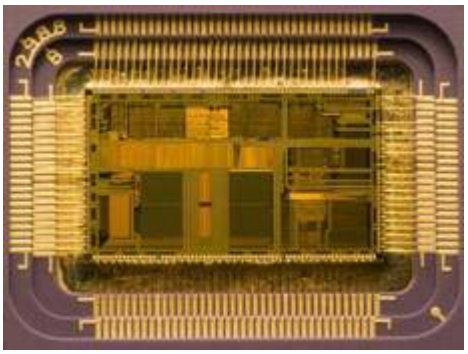
Biểu diễn số và ký tự

Các phép toán số học trên hệ đếm nhị phân

1.1 GIỚI THIỆU CHUNG VỀ VI XỬ LÝ – VI ĐIỀU KHIỂN

1.1.1 Tổng quan

Vi xử lý (viết tắt là **µP** hay **uP**), đôi khi còn được gọi là **bộ vi xử lý**, là một linh kiện điện tử được chế tạo từ các tranzito thu nhỏ tích hợp lên trên một vi mạch tích hợp đơn. Khối xử lý trung tâm (CPU) là một bộ vi xử lý được nhiều người biết đến nhưng ngoài ra nhiều thành phần khác trong máy tính cũng có bộ vi xử lý riêng của nó, ví dụ trên card màn hình (*video card*) chúng ta cũng có một bộ vi xử lý.



Hình 1-1.Bộ vi xử lý Intel 80486DX2

Trước khi xuất hiện các bộ vi xử lý, các CPU được xây dựng từ các mạch tích hợp cỡ nhỏ riêng biệt, mỗi mạch tích hợp chỉ chứa khoảng vào chục tranzito. Do đó, một CPU có thể là một bảng mạch gồm hàng ngàn hay hàng triệu vi mạch tích hợp. Ngày nay, công nghệ tích hợp đã phát triển, một CPU có thể tích hợp lên một hoặc vài vi mạch tích hợp cỡ lớn, mỗi vi mạch tích hợp cỡ lớn chứa hàng ngàn hoặc hàng triệu tranzito. Nhờ đó công suất tiêu thụ và giá thành của bộ vi xử lý đã giảm đáng kể.

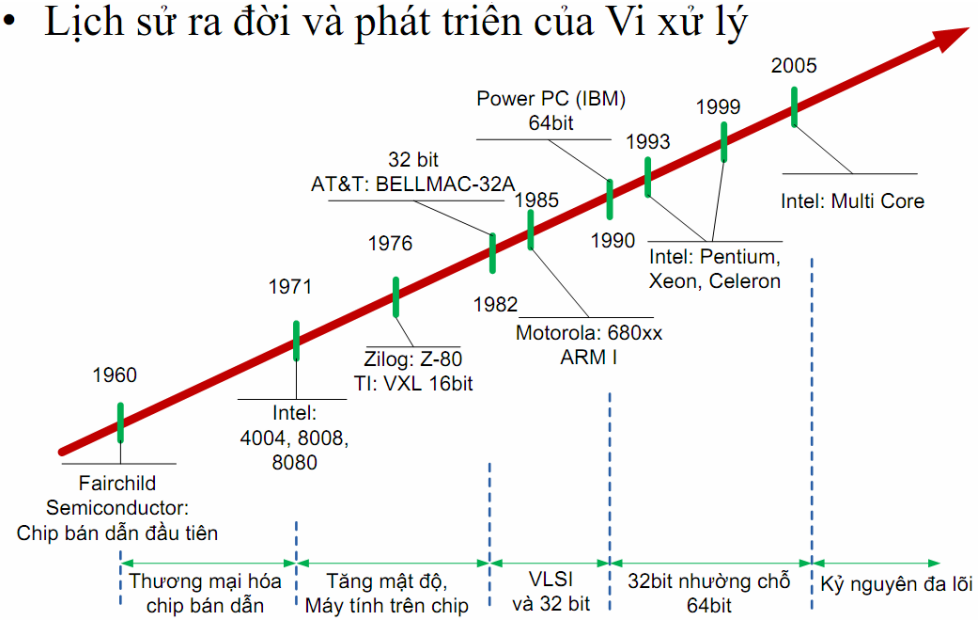
Vi điều khiển là một máy tính được tích hợp trên một chíp, nó thường được sử dụng để điều khiển các thiết bị điện tử. Vi điều khiển, thực chất, là một hệ thống bao gồm một vi xử lý có hiệu suất đủ dùng và giá thành thấp (khác với các bộ vi xử lý đa năng dùng trong máy tính) kết hợp với các khối ngoại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

Vi điều khiển thường được dùng để xây dựng các hệ thống nhúng. Nó xuất hiện khá nhiều trong các dụng cụ điện tử, thiết bị điện, máy giặt, lò vi sóng, điện thoại, đầu đọc DVD, thiết bị đa phương tiện, dây chuyền tự động, v.v.

Hầu hết các vi điều khiển ngày nay được xây dựng dựa trên kiến trúc Harvard, kiến trúc này định nghĩa bốn thành phần cần thiết của một hệ thống nhúng. Những thành phần này là lõi CPU, bộ nhớ chương trình (thông thường là ROM hoặc bộ nhớ Flash), bộ nhớ dữ liệu (RAM), một hoặc vài bộ định thời và các cổng vào/ra để giao tiếp với các thiết bị ngoại vi và các môi trường bên ngoài - tất cả các khối này được thiết kế trong một vi mạch tích hợp. Vi điều khiển khác với các bộ vi xử lý đa năng ở chỗ là nó có thể hoạt động chỉ với vài vi mạch hỗ trợ bên ngoài.

1.1.2 Lịch sử phát triển của các bộ xử lý

- Lịch sử ra đời và phát triển của Vi xử lý



Hình 1-2. Lịch sử phát triển của VXL

- Thế hệ 1 (1971 - 1973): vi xử lý 4 bit, đại diện là 4004, 4040, 8080 (Intel) hay IPM-16 (National Semiconductor).

+ Độ dài word thường là 4 bit (có thể lớn hơn).

+ Tốc độ 10 - 60 µs / lệnh với tần số xung nhịp 0.1 - 0.8 MHz. + Tập lệnh đơn giản và phải cần nhiều vi mạch phụ trợ.

- Thế hệ 2 (1974 - 1977): vi xử lý 8 bit, đại diện là 8080, 8085 (Intel) hay Z80 .

+ Tập lệnh phong phú hơn.

+ Địa chỉ có thể đến 64 KB. Một số bộ vi xử lý có thể phân biệt 256 địa chỉ cho thiết bị ngoại vi.

+ Sử dụng công nghệ NMOS hay CMOS.

+ Tốc độ 1 - 8 µs / lệnh với tần số xung nhịp 1 - 5 MHz

- Thế hệ 3 (1978 - 1982): vi xử lý 16 bit, đại diện là 68000/68010 (Motorola) hay 8086/ 80286/ 80386 (Intel)

+ Tập lệnh đa dạng với các lệnh nhân, chia và xử lý chuỗi.

+ Địa chỉ bộ nhớ có thể từ 1 - 16 MB và có thể phân biệt tới 64KB địa chỉ cho ngoại vi

+ Sử dụng công nghệ HMOS.

+ Tốc độ 0.1 - 1 µs / lệnh với tần số xung nhịp 5 - 10 MHz.

- Thế hệ 4: vi xử lý 32 bit 68020/68030/68040/68060 (Motorola) hay 80386/80486 (Intel) và vi xử lý 32 bit Pentium (Intel)

+ Bus địa chỉ 32 bit, phân biệt 4 GB bộ nhớ. + Có thể dùng thêm các bộ đồng xử lý (coprocessor). + Có khả năng làm việc với bộ nhớ ảo.

+ Có các cơ chế pipeline, bộ nhớ cache.

+ Sử dụng công nghệ HCMOS.

- Thế hệ 5: vi xử lý 64 bit

1.1.3 Vi xử lý và vi điều khiển

Khái niệm "vi xử lý" (microprocessor) và "vi điều khiển" (microcontroller).

Về cơ bản hai khái niệm này không khác nhau nhiều, "vi xử lý" là thuật ngữ chung dùng để đề cập đến kỹ thuật ứng dụng các công nghệ vi điện tử, công nghệ tích hợp và khả năng xử lý theo chương trình vào các lĩnh vực khác nhau. Vào những giai đoạn đầu trong quá trình phát triển của công nghệ vi xử lý, các chip (hay các vi xử lý) được chế tạo chỉ tích hợp những phần cứng thiết yếu như CPU cùng các mạch giao tiếp giữa CPU và các phần cứng khác. Trong giai đoạn này, các phần cứng khác (kể cả bộ nhớ) thường không được tích hợp trên chip mà phải ghép nối thêm bên ngoài. Các phần cứng này được gọi là các ngoại vi (Peripherals). Về sau, nhờ sự phát triển vượt bậc của công nghệ tích hợp, các ngoại vi cũng được tích hợp vào bên trong IC và người ta gọi các vi xử lýđã được tích hợp thêm các ngoại vi là các "vi điều khiển".

Vi xử lý có các khối chức năng cần thiết để lấy dữ liệu, xử lý dữ liệu và xuất dữ liệu ra ngoài sau khi đã xử lý. Và chức năng chính của Vi xử lý chính là xử lý dữ liệu, chẳng hạn như cộng, trừ, nhân, chia, so sánh.v.v... Vi xử lý không có khả năng giao tiếp trực tiếp với các thiết bị ngoại vi, nó chỉ có khả năng nhận và xử lý dữ liệu mà thôi.

Để vi xử lý hoạt động cần có chương trình kèm theo, các chương trình này điều khiển các mạch logic và từ đó vi xử lý xử lý các dữ liệu cần thiết theo yêu cầu. Chương trình là tập hợp các lệnh để xử lý dữ liệu thực hiện từng lệnh được lưu trữ trong bộ nhớ, công việc thực hành lệnh bao gồm: nhận lệnh từ bộ nhớ, giải mã lệnh và thực hiện lệnh sau khi đã giải mã.

Để thực hiện các công việc với các thiết bị cuối cùng, chẳng hạn điều khiển động cơ, hiển thị kí tự trên màn hình .... đòi hỏi phải kết hợp vi xử lý với các mạch điện giao tiếp với bên ngoài được gọi là các thiết bị I/O (nhập/xuất) hay còn gọi là các thiết bị ngoại vi. Bản thân các vi xử lý khi đứng một mình không có nhiều hiệu quả sử dụng, nhưng khi là một phần của một máy tính, thì hiệu quả ứng dụng của Vi xử lý là rất lớn. Vi xử lý kết hợp với các thiết bị khác được sử trong các hệ thống lớn, phức tạp đòi hỏi phải xử lý một lượng lớn các phép tính phức tạp, có tốc độ nhanh. Chẳng hạn như các hệ thống sản xuất tự động trong công nghiệp, các tổng đài điện thoại, hoặc ở các robot có khả năng hoạt động phức tạp v.v...

Bộ Vi xử lý có khả năng vượt bậc so với các hệ thống khác về khả năng tính toán, xử lý, và thay đổi chương trình linh hoạt theo mục đích người dùng, đặc biệt hiệu quả đối với các bài toán và hệ thống lớn. Tuy nhiên đối với các ứng dụng nhỏ, tầm tính toán không đòi hỏi khả năng tính toán lớn thì việc ứng dụng vi xử lý cần cân nhắc. Bởi vì hệ thống dù lớn hay nhỏ, nếu dùng vi xử lý thì cũng đòi hỏi các khối mạch điện giao tiếp phức tạp như nhau. Các khối này bao gồm bộ nhớ để chứa dữ liệu và chương trình thực hiện, các mạch điện giao tiếp ngoại vi để xuất nhập và điều khiển trở lại, các khối này cùng liên kết với vi xử lý thì mới thực hiện được

công việc. Để kết nối các khối này đòi hỏi người thiết kế phải hiểu biết tinh tường về các thành phần vi xử lý, bộ nhớ, các thiết bị ngoại vi. Hệ thống được tạo ra khá phức tạp, chiếm nhiều không gian, mạch in phức tạp và vấn đề chính là trình độ người thiết kế. Kết quả là giá thành sản phẩm cuối cùng rất cao, không phù hợp để áp dụng cho các hệ thống nhỏ.

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

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

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

Trong tài liệu này, ranh giới giữa hai khái niệm “vi xử lý” và “vi điều khiển” thực sự không cần phải phân biệt rõ ràng. Chúng tôi sẽ dùng thuật ngữ “vi xử lý” khi đề cập đến các khái niệm cơ bản của kỹ thuật vi xử lý nói chung và sẽ dùng thuật ngữ “vi điều khiển” khi đi sâu nghiên cứu một họ chip cụ thể.

1.1.4 Ứng dụng của Vi xử lý – vi điều khiển

Vi xử lý, chính là chip của các loại máy tính ngày nay, nên hẳn các bạn đã biết rất rõ nó có những ứng dụng gì. Ở đây, tôi chỉ nói đên ứng dụng của vi điều khiển. Vi điều khiển có thể dùng trong thiết kế các loại máy tính nhúng. Máy tính nhúng có trong hầu hết các thiết bị tự động, thông minh ngày nay. Chúng ta có thể dùng vi điều khiển để thiết kế bộ điều khiển cho các sản phẩm như:

❖ **Trong các sản phẩm dân dụng:**

- Nhà thông minh:
  - Cửa tự động
  - Khóa số
  - Tự động điều tiết ánh sáng thông minh (bật/tắt đèn theo thời gian, theo cường độ ánh sáng,...)
  - Điều khiển các thiết bị từ xa (qua điều khiển, qua tiếng vỗ tay,...)
  - Điều tiết hơi ẩm, điều tiết nhiệt độ, điều tiết không khí, gió
  - Hệ thống vệ sinh thông minh,...
- Trong quảng cáo:
  - Các loại biển quảng cáo nháy chữ
  - Quảng cáo ma trận LED (một màu, 3 màu, đa màu)
  - Điều khiển máy cuốn bạt quảng cáo,...
- Các máy móc dân dụng
  - Máy điều tiết độ ẩm cho vườn cây
  - Buồng ấp trứng gà/vịt
  - Đồng hồ số, đồng hồ số có điều khiển theo thời gian
- Các sản phẩm giải trí
  - Máy nghe nhạc
  - Máy chơi game
  - Đầu thu kỹ thuật số, đầu thu set-top-box,...

❖ **Trong các thiết bị y tế:**

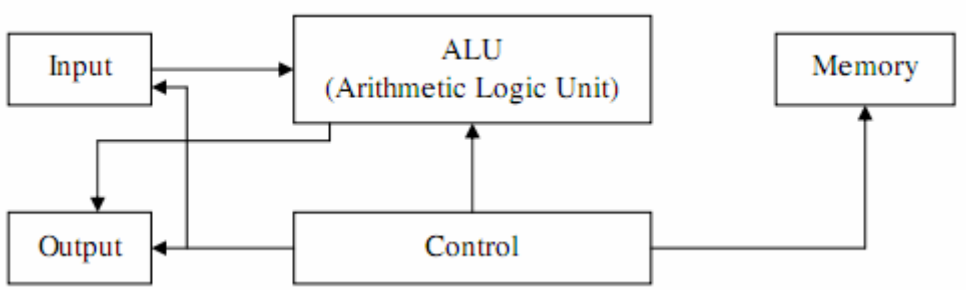
- Máy móc thiết bị hỗ trợ: máy đo nhịp tim, máy đo đường huyết, máy đo huyết áp, điện tim đồ, điện não đồ,…
- Máy cắt/mài kính
- Máy chụp chiếu (city, X-quang,...)

❖ **Các sản phẩm công nghiệp:**

- Điều khiển động cơ
- Điều khiển số (PID, mờ,...)
- Đo lường (đo điện áp, đo dòng điện, áp suất, nhiệt độ,...)
- Cân băng tải, cân toa xe, cân ô tô,...
- Máy cán thép: điều khiển động cơ máy cán, điều khiển máy quấn thép,..
- Làm bộ điều khiển trung tâm cho RoBot
- Ổn định tốc độ động cơ
- Đếm sản phẩm của 1 nhà máy, xí nghiệp,…
- Máy vận hành tự động (dạng CNC)
- ...

1.2 Cấu trúc chung của hệ vi xử lý

Sơ đồ khối một máy tính cổ điển



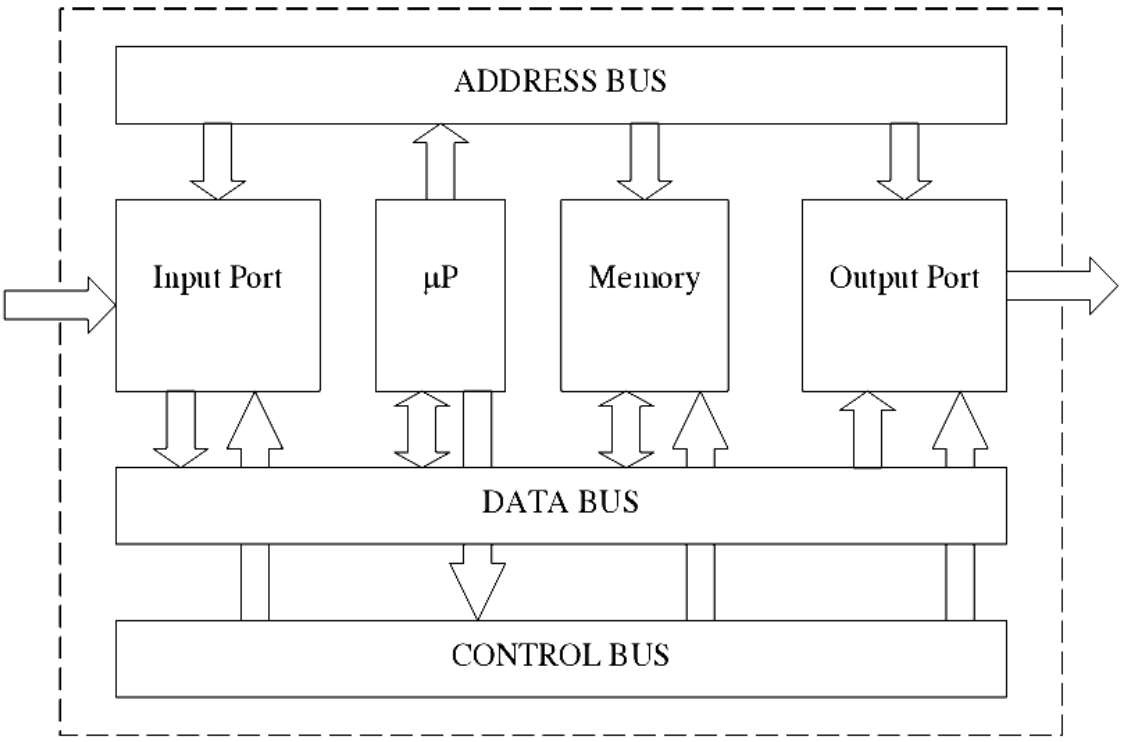
Hình 1-3. Sơ đồ khối một máy tính cổ điển

- ALU (đơn vị logic số học): thực hiện các bài toán cho máy tính bao gồm: +, *, /,-, phép toán logic, …
- Control (điều khiển): điều khiển, kiểm soát các đường dữ liệu giữa các thành phần của máy tính.
- Memory (bộ nhớ): lưu trữ chương trình hay các kết quả trung gian.
- Input (nhập), Output (Xuất): xuất nhập dữ liệu (còn gọi là thiết bị ngoại vi).

Về cơ bản kiến trúc của một vi xử lý gồm những phần cứng sau:

- Đơn vị xử lý trung tâm CPU (Central Processing Unit).
- Các bộ nhớ (Memories).
- Các cổng vào/ra (song song (Parallel I/O Ports), nối tiếp (Serial I/O Ports))
- Các bộ đếm/bộ định thời (Timers).
- Hệ thống BUS (Địa chỉ, dữ liệu, điều khiển)

Ngoài ra với mỗi loại vi điều khiển cụ thể còn có thể có thêm một số phần cứng khác như bộ biến đổi tương tự-số ADC, bộ biến đổi số-tương tự DAC, các mạch điều chế dạng sóng WG, điều chế độ rộng xung PWM…Bộ não của mỗi vi xử lý chính là CPU, các phần cứng khác chỉ là các cơ quan chấp hành dưới quyền của CPU. Mỗi cơ quan này đều có một cơ chế hoạt động nhất định mà CPU phải tuân theo khi giao tiếp với chúng.

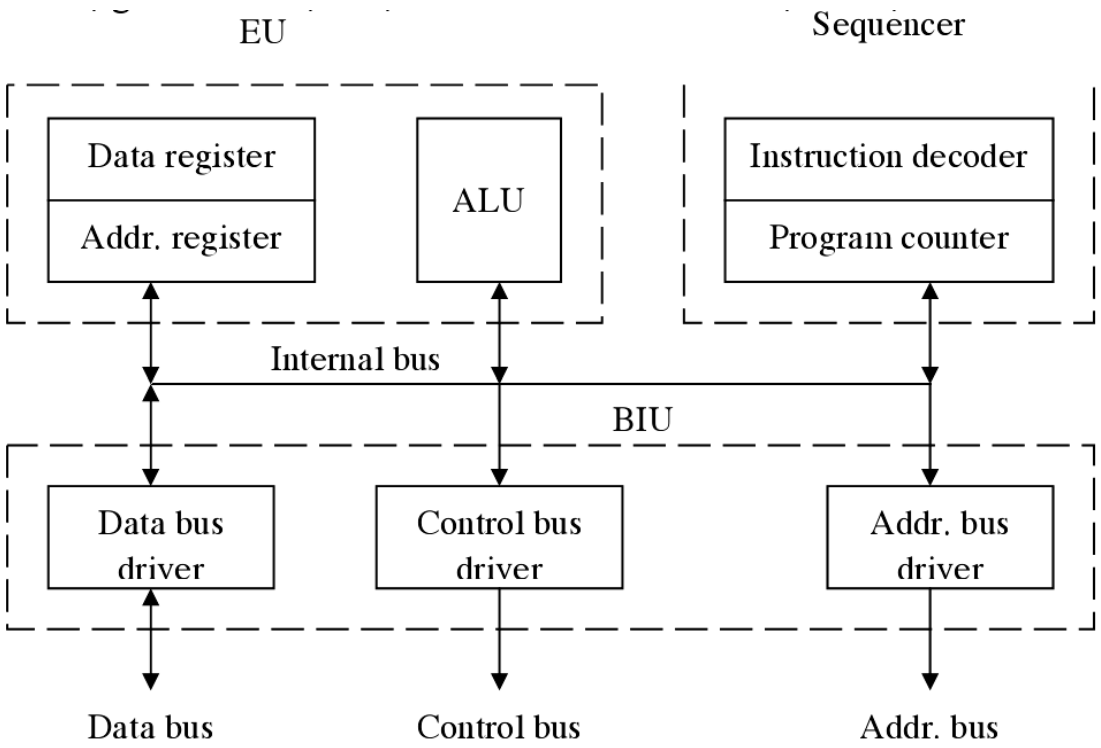


Hình 1-4. Sơ đồ khối hệ vi xử lý

Để có thể giao tiếp và điều khiển các cơ quan chấp hành (các ngoại vi), CPU sử dụng 03 loại tín hiệu cơ bản là tín hiệu địa chỉ (Address), tín hiệu dữ liệu (Data) và tín hiệu điều khiển (Control). Về mặt vật lý thì các tín hiệu này là các đường nhỏ dẫn điện nối từ CPU đến các ngoại vi hoặc thậm chí là giữa các ngoại vi với nhau. Tập hợp các đường tín hiệu có cùng chức năng gọi là các bus. Như vậy ta có các bus địa chỉ, bus dữ liệu và bus điều khiển.

1.2.1 Khối xử lý trung tâm (CPU)

CPU có cấu tạo gồm có đơn vị xử lý số học và lôgic (ALU), các thanh ghi, các khối lôgic và các mạch giao tiếp. Chức năng của CPU là tiến hành các thao tác tính toán xử lý, đưa ra các tín hiệu địa chỉ, dữ liệu và điều khiển nhằm thực hiện một nhiệm vụ nào đó do người lập trình đưa ra thông qua các lệnh (Instructions).



Hình 1-5. Khối xử lý trung tâm

1.2.2 Hệ thống bus

❖ Bus địa chỉ - Address bus

Là các đường tín hiệu song song 1 chiều nối từ CPU đến bộ nhớ

Độ rộng bus: là số các đường tín hiệu, có thể là 8, 18, 20, 24, 32 hay 64.

CPU gửi giá trị địa chỉ của ô nhớ cần truy nhập (đọc/ghi) trên các đường tín hiệu này.

1 CPU với n đường địa chỉ sẽ có thể địa chỉ hoá được 2n ô nhớ. Ví dụ, 1 Cpu có 16 đường địa chỉ có thể địa chỉ hoá được 216 hay 65,536 (64K) ô nhớ.

❖ Bus dữ liệu - Data bus

Độ rộng Bus: 4, 8, 16, 32 hay 64 bits

Là các đường tín hiệu song song 2 chiều, nhiều thiết bị khác nhau có thể được nối với bus dữ liệu; nhưng tại một thời điểm, chỉ có 1 thiết bị duy nhất có thể được phép đưa dữ liệu lên bus dữ liệu.

Bất kỳ thiết bị nào đọc kết nối đến bus dữ liệu phải có đầu ra ở dạng 3 trạng thái, sao cho nó có thể ở trạng thái treo (trở kháng cao) nếu không được sử dụng.

❖ Bus điều khiển - Control bus

Bao gồm 4 đến 10 đường tín hiệu song song.

CPU gửi tín hiệu ra bus điều khiển để cho phép các đầu ra của ô nhớ hay các cổng I/O đã được địa chỉ hoá. Các tín hiệu điều khiển thường là: đọc/ ghi bộ nhớ - memory read, memory write, đọc/ ghi cổng vào/ra - I/O read, I/O write.

Ví dụ, để đọc 1 byte dữ liệu từ ô nhớ sẽ cần đến các hoạt động sau:

CPU đưa ra địa chỉ của ô nhớ cần đọc lên bus địa chỉ.

CPU đưa ra tín hiệu đọc bộ nhớ - Memory Read trên bus điều khiển.

Tín hiệu điều khiển này sẽ cho phép thiết bị nhớ đã được địa chỉ hoá đưa byte dữ liệu lên bus dữ liệu. Byte dữ liệu từ ô nhớ sẽ được truyền tải qua bus dữ liệu đến CPU.

1.3 Định dạng dữ liệu và biểu diễn thông tin trong hệ vi xử lý – vi điều khiển

1.3.1 Các hệ đếm

- Hệ thập phân - Decimal
- Hệ nhị phân - Binary
- Hệ16 - Hexadecimal
- Mã BCD (standard BCD, gray code): (Binary Coded Decimal)

Trong thực tế, đối với một số ứng dụng như đếm tần, đo điện áp, … ngõ ra ở dạng số thập phân, ta dùng mã BCD. Mã BCD dùng 4 bit nhị phân để mã hoá cho một

số thập phân 0..9. Như vậy, các số hex A..F không tồn tại trong mã BCD.

Mã BCD gồm có 2 loại:

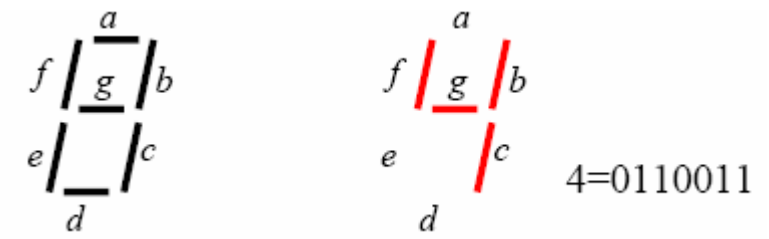
- Mã BCD không nén (unpacked): biểu diễn một số BCD bằng 8 bit nhị phân
- Mã BCD nén (packed): biểu diễn một số BCD bằng 4 bit nhị phân

VD: Số thập phân 5 2 9

Số BCD không nén 0000 0101b 0000 0010b 0000 1001b

Số BCD nén 0101b 0010b 1001b

- Mã hiển thị 7 đoạn (7-segment display code)



Hình 1-6.LED 7 thanh và cách mã hóa

- **Các mã hệ đếm thông dụng**

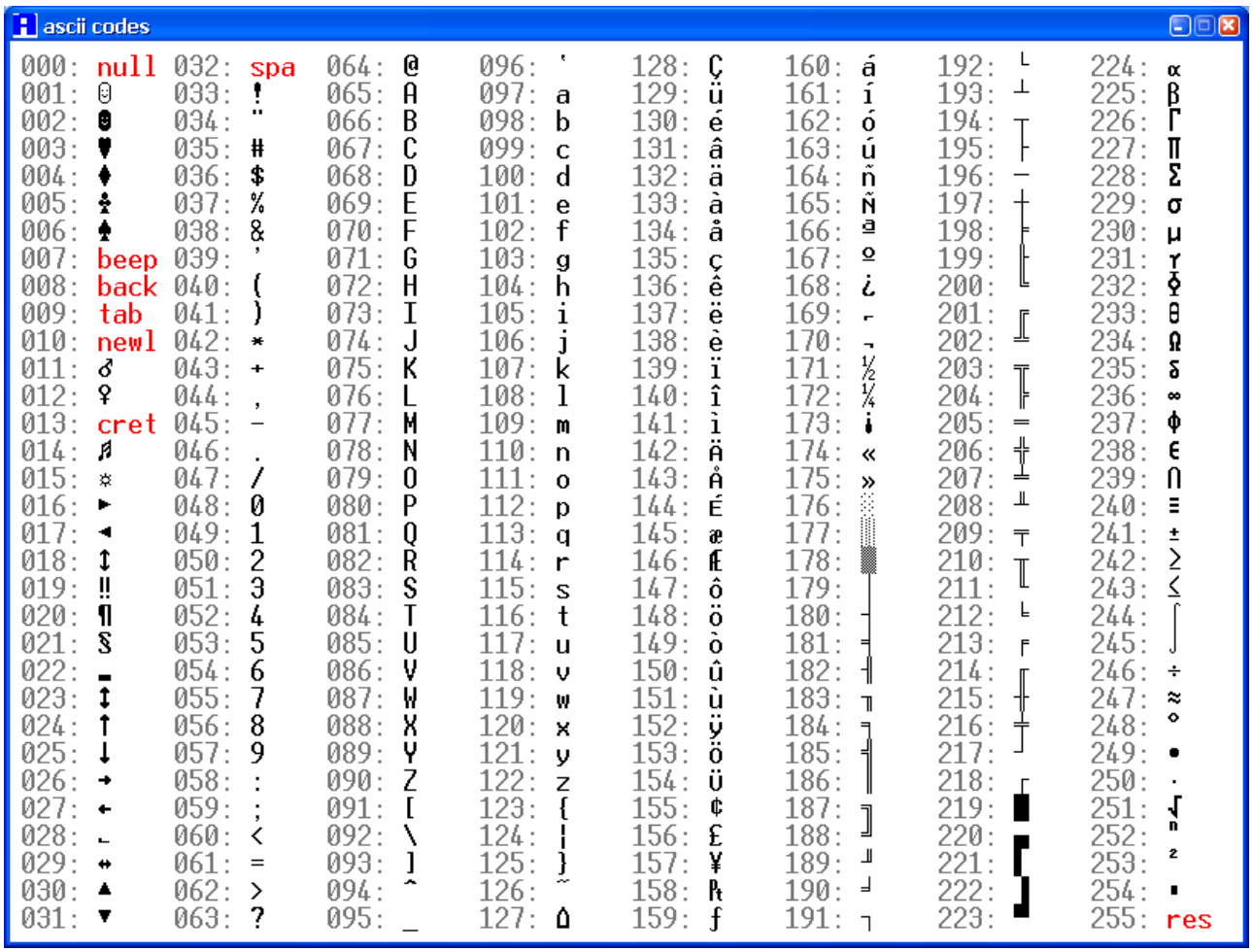
Hệ 10	Hệ 2	Hệ 8	Hệ 16	Binary-Coded Decimal		Gray Code	7-Segment	
				8421 BCD	EXCESS-3		abcdefg	Display
0	0000	0	0	0000	0011 0011	0000	111111	0
1	0001	1	1	0001	0011 0100	0001	011000	1
2	0010	2	2	0010	0011 0101	0011	110110	2
3	0011	3	3	0011	0011 0110	0010	111100	3
4	0100	4	4	0100	0011 0111	0110	011001	4
5	0101	5	5	0101	0011 1000	0111	101101	5
6	0110	6	6	0110	0011 1001	0101	101111	6
7	0111	7	7	0111	0011 1010	0100	111000	7
8	1000	10	8	1000	0011 1011	1100	111111	8
9	1001	11	9	1001	0011 1100	1101	111001	9
10	1010	12	A	0001 0000	0100 0011	1111	111110	A
11	1011	13	B	0001 0001	0100 0100	1110	001111	B
12	1100	14	C	0001 0010	0100 0101	1010	000110	C
13	1101	15	D	0001 0011	0100 0110	1011	011110	D
14	1110	16	E	0001 0100	0100 0111	1001	110111	E
15	1111	17	F	0001 0101	0100 1000	1000	100011	F

Bảng 1-1. Giá trị tương ứng giữa các hệ số

1.3.2 Mã ký tự - Alphanumeric CODE (ASCII, EBCDIC)

HEX	DEC	CHR	Ctrl	HEX	DEC	CHR	HEX	DEC	CHR	HEX	DEC	CHR
0	0	NUL	^@	20	32	SP	40	64	@	60	96	`
1	1	SOH	^A	21	33	!	41	65	A	61	97	a
2	2	STX	^B	22	34	"	42	66	B	62	98	b
3	3	ETX	^C	23	35	#	43	67	C	63	99	c
4	4	EOT	^D	24	36	$	44	68	D	64	100	d
5	5	ENQ	^E	25	37	%	45	69	E	65	101	e
6	6	ACK	^F	26	38	&	46	70	F	66	102	f
7	7	BEL	^G	27	39	'	47	71	G	67	103	g
8	8	BS	^H	28	40	(	48	72	H	68	104	h
9	9	HT	^I	29	41	)	49	73	I	69	105	i
0A	10	LF	^J	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	^K	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	^L	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	^M	2D	45	–	4D	77	M	6D	109	m
0E	14	SO	^N	2E	46	.	4E	78	N	6E	100	n
0F	15	SI	^O	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	^P	30	48	0	50	80	P	70	112	p
11	17	DC1	^Q	31	49	1	51	81	Q	71	113	q
12	18	DC2	^R	32	50	2	52	82	R	72	114	r
13	19	DC3	^S	33	51	3	53	83	S	73	115	s
14	20	DC4	^T	34	52	4	54	84	T	74	116	t
15	21	NAK	^U	35	53	5	55	85	U	75	117	u
16	22	SYN	^V	36	54	6	56	86	V	76	118	v
17	23	ETB	^W	37	55	7	57	87	W	77	119	w
18	24	CAN	^X	38	56	8	58	88	X	78	120	x
19	25	EM	^Y	39	57	9	59	89	Y	79	121	y
1A	26	SUB	^Z	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC		3B	59	;	5B	91	[	7B	123	{
1C	28	FS		3C	60	<	5C	92	\	7C	124	|
1D	29	GS		3D	61	=	5D	93	]	7D	125	}
1E	30	RS		3E	62	>	5E	94	^	7E	126	~
1F	31	US		3F	63	?	5F	95	_	7F	127	DEL

Hình 1-7. Bảng mã ASCII



Hình 1-8. Bảng mã ASCII có cả ký tự trong phần mở rộng

1.3.3 Các phép toán số học trên hệ đếm nhị phân

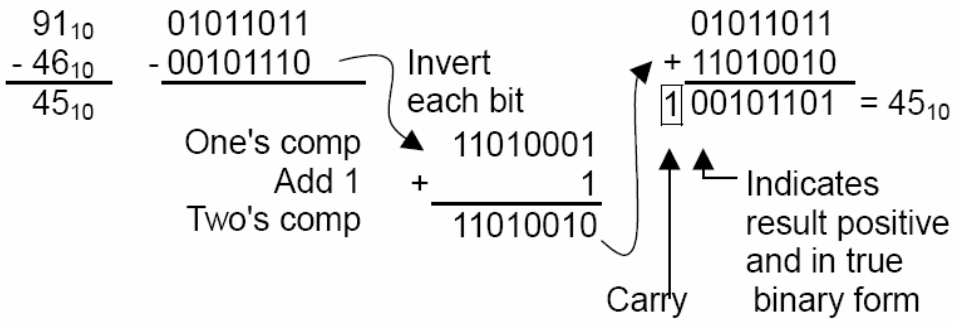
❖ Phép cộng nhị phân

Vào			Ra	
A	B	B_{IN}	D	B_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

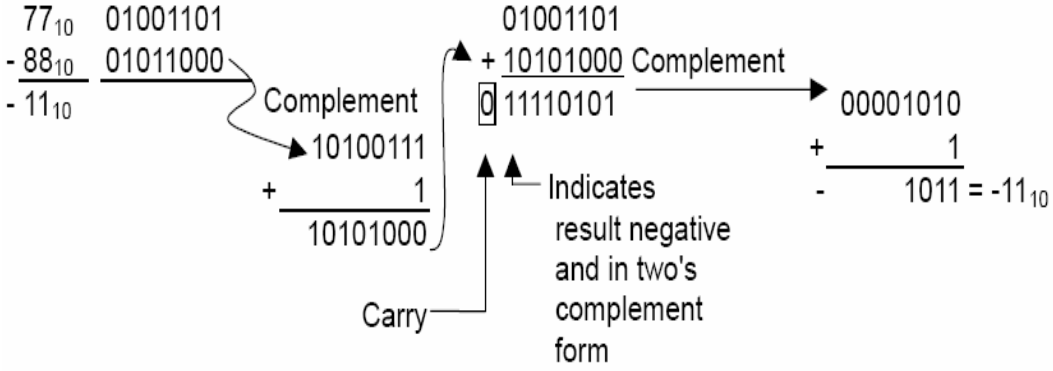
❖ Phép trừ nhị phân

Vào			Ra	
A	B	B_{IN}	D	B_{OUT}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Phép trừ nhị phân, chính là phép cộng nhị phân với số bù 2 của số trừ, trường hợp kết quả dương:



Trường hợp kết quả âm:



Phép nhân, phép chia, đề nghị sinh viên tự nghiên cứu.

CHƯƠNG 2.
HỌ VI XỬ LÝ INTEL 80x86

Mục tiêu:

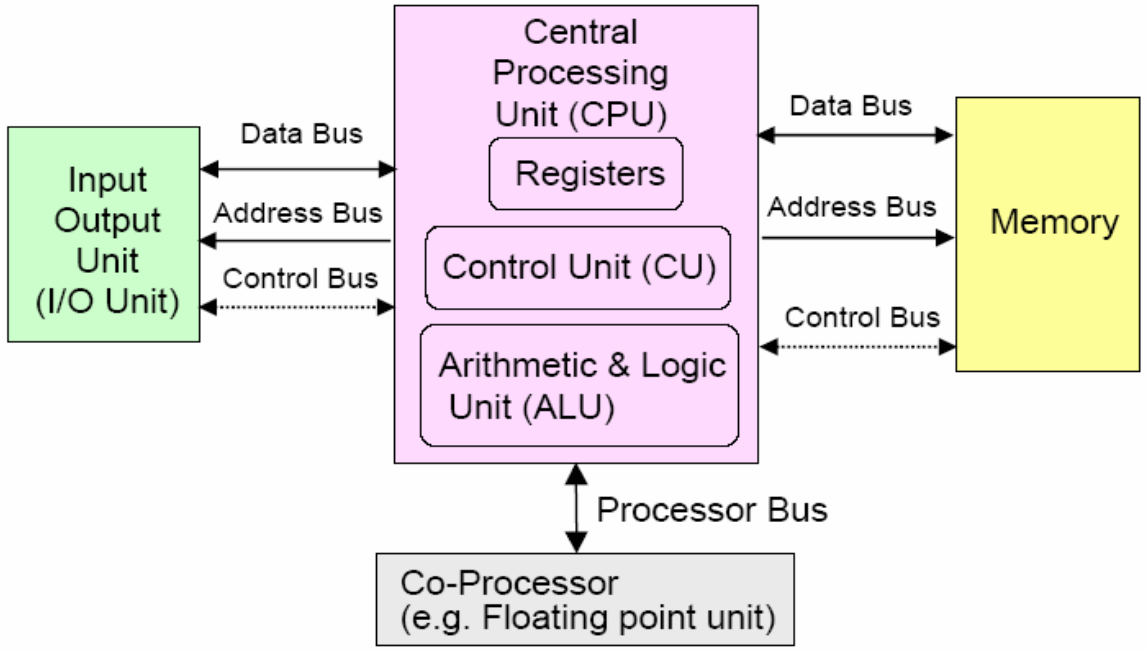
Hiểu được cấu trúc phần cứng của hệ vi xử lý; hiểu và vận dụng được các chế độ địa chỉ; nắm được tập lệnh và lập trình cho hệ vi xử lý 80x86

Tóm tắt chương:

- *Cấu trúc phần cứng của bộ vi xử lý 8086*
- *Chế độ địa chỉ*
- *Tập lệnh*
- *Các mạch phụ trợ*
- *Biểu đồ thời gian ghi/đọc*
- *Lập trình hợp ngữ (Assembly) cho vi xử lý 80x86*

2.1 Cấu trúc phần cứng của bộ vi xử lý 8086

2.1.1 Tổng quan

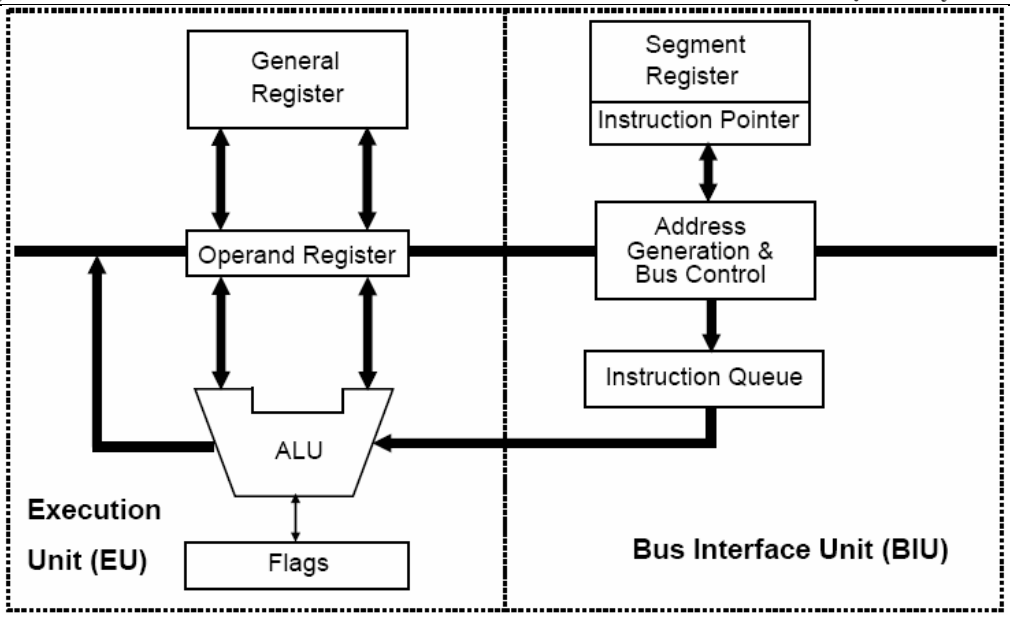


Hình 2-1.Tổng quan về phần cứng bộ xử lý

- **Control Unit** (CU) tạo ra tất cả các tín hiệu điều khiển trong CPU. Nó khởi tạo các thanh ghi khi mở nguồn, tạo ra các tín hiệu để lấy lệnh cho ALU. Khối điều khiển có thể được thực hiện hoàn toàn bởi phần cứng (điều khiển cứng, ví dụ như sử dụng một bộ đếm trạng thái và một mảng logic khả lập triình) hay kết hợp giữa các lệnh phần mềm (vi lệnh được lưu trữ trong CPU) và phần cứng (bộ điều khiển vi chương trình. Cả hai họ vi xử lý Intel 8086 và Motorola 68000 đều sử dụng các bộ điều khiển vi chương trình.
- **Registers** – là các bộ nhớ nhỏ, nhanh, thường được sử dụng để lưu dữ liệu và địa chỉ gắn với (tương ứng với) các mã lệnh của chương trình.
- **ALU** thực hiện các phép toán số học và logic

2.1.2 Cấu trúc bên trong và sự hoạt động

Trong sơ đồ khối "Hình 2-2.Sự hoạt động của CPU" ta thấy trong CPU 8086 có hai khối chính: *khối phối ghép bus* (bus interface unit, BIU) và *khối thực hiện lệnh* (execution unit, EU). Việc chia CPU thành hai phần đồng thời có liên hệ với nhau qua đệm lệnh làm tăng đáng kể tốc độ xử lý của CPU. Các bus bên trong CPU có nhiệm vụ chuyển tải tín hiệu của các khối khác. Trong số các bus có bus dữ liệu 16 bit của ALU, bus các tín hiệu điều khiển ở EU và bus trong của hệ thống ở BIU. Trước khi đi ra bus ngoài hoặc đi vào bus trong của bộ vi xử lý, các tín hiệu truyền trên bus thường được cho đi qua các bộ đệm để nâng cao tính tương thích cho nối ghép hoặc nâng cao khả năng phối ghép.



Hình 2-2.Sự hoạt động của CPU

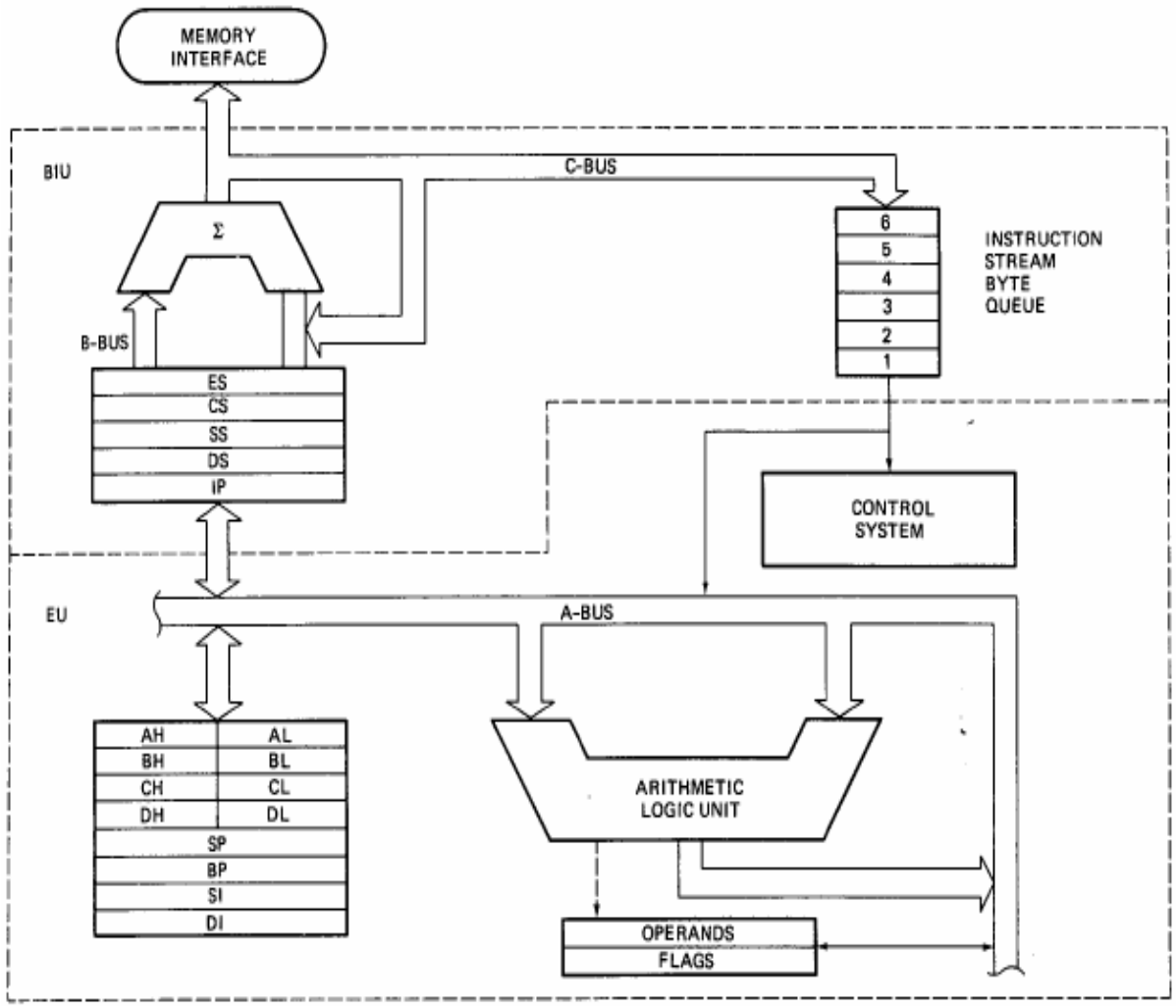
BIU có nhiệm vụ đưa ra địa chỉ, đọc mã lệnh từ bộ nhớ, đọc/ghi dữ liệu từ/vào cổng hoặc bộ nhớ. Bên trong BIU còn có bộ nhớ đệm lệnh (còn gọi là hàng đợi lệnh) dùng để chứa các lệnh đã đọc được nằm sẵn chờ EU xử lý. EU có nhiệm vụ cung cấp địa chỉ cho BIU để khối này đọc lệnh và dữ liệu, còn bản thân nó thì giải mã lệnh và thực hiện lệnh. Mã lệnh đọc vào từ bộ nhớ được đưa đến đầu vào của bộ giải mã (nằm trong khối điều khiển CU), các thông tin thu được từ đầu ra của bộ giải mã sẽ được đưa đến mạch tạo xung điều khiển để tạo ra các dãy xung khác nhau (tùy từng lệnh) điều khiển hoạt động của các bộ phận bên trong và bên ngoài CPU. Trong EU còn có khối tính toán số học và logic ALU dùng để thực hiện các thao tác khác nhau với các toán hạng của lệnh.

2.1.2.1 Sơ đồ khối bên trong của 8086

❖ **Đơn vị giao tiếp Bus (BIU)**

BIU bao gồm các thanh ghi đoạn (segment registers: CS, DS, SS, ES), con trỏ lệnh IP (instruction pointer) và bộ điều khiển logic bus (bus control logic, BCL). Đơn vị giao diện BIU còn có bộ nhớ đệm cho mã lệnh. Bộ nhớ này có chiều dài 4 byte (trong 8088) và 6 byte (trong 8086). Bộ nhớ đệm mã lệnh được nối với khối điều khển CB (control block) của đơn vị thực hiện lệnh EU. Bộ nhớ này lưu trữ tạm thời mã lệnh trong một dãy gọi là hàng đợi lệnh. Hàng đợi lệnh cho phép bộ vi xử lý có khả năng xử lý xen kẽ liên tục dòng mã lệnh (pipelining). Hoạt động của bộ CPU được chia làm ba giai đoạn: đọc mã lệnh (operation code fetching), giải mã lệnh (decording) và thực hiện lệnh (execution).

BIU đưa ra địa chỉ, đọc mã lệnh từ bộ nhớ, đọc/ghi dữ liệu từ các cổng vào hoặc bộ nhớ. Nói cách khác BIU chịu trách nhiệm đưa địa chỉ ra bus và trao đổi dữ liệu với bus.



Hình 2-3.Sơ đồ khối bên trong 8086

❖ Đơn vị xử lý lệnh (EU)

Trong EU có khối điều khiển (control unit, CU). Chính tại bên trong khối điều khiển này có mạch giải mã lệnh. Mã lệnh đọc vào từ bộ nhớ được đưa đến đầu vào của bộ giải mã, các thông tin thu được từ đầu ra của nó sẽ được đưa đến mạch tạo xung điều khiển, kết quả thu được là các dãy xung khác nhau tuỳ theo mã lệnh, để điều khiển hoạt động của các bộ phận bên trong và bên ngoài CPU.

Trong EU có khối số học và lôgic (arithmatic and logic unit, ALU) chuyên thực hiện các phép tính số học và logic mã toán tử của nó nằm trong các thanh ghi đa năng. Kết quả thường được đặt về thanh ghi AX.

Ngoài ra trong EU còn có các thanh ghi đa năng (registers: AX, BX, CX, DX, SP, BP, SI, DI), thanh ghi cờ FR (flag register).

Tóm lại, khi CPU hoạt động EU sẽ cung cấp thông tin về địa chỉ cho BIU để khối này đọc lệnh và dữ liệu, còn bản thân nó thì giải mã và thực hiện lệnh.

❖ Nhóm các thanh ghi

Vi xử lý 8086 có tất cả 14 thanh ghi nội. Các thanh ghi này có thể phân nhóm như sau:

- Thanh ghi dữ liệu (data register)
- Thanh ghi chỉ số và con trỏ (index & pointer register)
- Thanh ghi đoạn (segment register)
- Thanh ghi cờ

• Các thanh ghi dữ liệu

Các thanh ghi dữ liệu gồm có các thanh ghi 16 bit AX, BX, CX và DX trong đó nửa cao và nửa thấp của mỗi thanh ghi có thể định địa chỉ một cách độc lập. Các nửa thanh ghi này (8 bit) có tên là AH và AL, BH và BL, CH và CL, DH và DL.

Các thanh ghi này được sử dụng trong các phép toán số học và logic hay trong quá trình chuyển dữ liệu.

Trong đó :

AX (ACC – Accumulator): thanh ghi tích luỹ

BX (Base): thanh ghi cơ sở

CX (Count): đếm

DX (Data): thanh ghi dữ liệu

Ở "Bảng 2-1. Các thanh ghi" chỉ ra ứng dụng của các thanh ghi dữ liệu trong các phép toán như sau

Thanh ghi	Mục đích
AX	MUL, IMUL (toán hạng nguồn kích thước word) DIV, IDIV (toán hạng nguồn kích thước word) IN (nhập word) OUT (xuất word) CWD Các phép toán xử lý chuỗi (string)
AL	MUL, IMUL (toán hạng nguồn kích thước byte) DIV, IDIV (toán hạng nguồn kích thước byte) IN (nhập byte) OUT (xuất byte) XLAT AAA, AAD, AAM, AAS (các phép toán ASCII) CBW (đổi sang word) DAA, DAS (số thập phân)

Thanh ghi	Mục đích
	Các phép toán xử lý chuỗi (string)
AH	MUL, IMUL (toán hạng nguồn kích thước byte) DIV, IDIV (toán hạng nguồn kích thước byte) CBW (đổi sang word)
BX	XLAT
CX	LOOP, LOOPE, LOOPNE Các phép toán string với tiếp dầu ngữ REP
CL	RCR, RCL, ROR, ROL (quay với số đếm byte) SHR, SAR, SAL (dịch với số đếm byte)
CX	MUL, IMUL (toán hạng nguồn kích thước word) DIV, IDIV (toán hạng nguồn kích thước word)

Bảng 2-1. Các thanh ghi

- **Các thanh ghi chỉ số và con trỏ**

Bao gồm các thanh ghi 16 bit SP, BP, SI và DI, thường chứa các giá trị offset (độ lệch) cho các phần tử định địa chỉ trong một phân đoạn (segment). Chúng có thể được sử dụng trong các phép toán số học và logic. Hai thanh ghi con trỏ (SP – Stack Pointer và BP – Base Pointer) cho phép truy xuất dễ dàng đến các phần tử đang ở trong ngăn xếp (stack) hiện hành. Các thanh ghi chỉ số (SI – Source Index và DI – Destination Index) được dùng để truy xuất các phần tử trong các đoạn dữ liệu và đoạn thêm (extra segment). Thông thường, các thanh ghi con trỏ liên hệ đến đoạn stack hiện hành và các thanh ghi chỉ số liên hệ đến doạn dữ liệu hiện hành. SI và DI dùng trong các phép toán chuỗi.

- **Các thanh ghi đoạn**

Bao gồm các thanh ghi 16 bit CS (Code segment), DS (Data segment), SS (stack segment) và ES (extra segment), dùng để định địa chỉ vùng nhớ 1 MB bằng cách chia thành 16 đoạn 64 KB.

Tất cả các lệnh phải ở trong đoạn mã hiện hành, được định địa chỉ thông qua thanh ghi CS. Offset (độ lệch) của mã được xác định bằng thanh ghi IP. Dữ liệu chương trình thường được đặt ở đoạn dữ liệu, định vị thông qua thanh ghi DS. Stack định vị thông qua thanh ghi SS. Thanh ghi đoạn thêm có thể sử dụng để định địa chỉ các toán hạng, dữ liệu, bộ nhớ và các phần tử khác ngoài đoạn dữ liệu và stack hiện hành.

Do Bus địa chỉ của vi xử lý 8086 có kích thước là 20 bit, nhưng các thanh ghi con trỏ và thanh ghi chỉ số chỉ rộng 16 bit nên không thể định địa chỉ cho toàn bộ nhớ vật lý của máy tính là (2^{20}B = 1.048.576B = 1Mbyte). Vì vậy trong chế độ thực (real mode) bộ nhớ được chia làm nhiều đoạn để một thanh ghi con trỏ 16 bit

có thể quản lý được. Các thanh ghi đoạn 16 bit sẽ chỉ ra địa chỉ đầu của 4 đoạn trong bộ nhớ, dung lượng lớn nhất của mỗi đoạn nhớ sẽ dài 2^{16} = 64 Kbyte và tại một thời điểm nhất định bộ vi xử lý chỉ làm việc được với 4 đoạn nhớ 64Kbyte này. Việc thay đổi giá trị của các thanh ghi đoạn làm cho các đoạn có thể dịch chuyển linh hoạt trong không gian 1 Mbyte, vì vậy các đoạn có thể nằm cách nhau khi thông tin cần lưu trong chúng đòi hỏi dung lượng đủ 64 Kbyte hoặc cũng có thể nằm chồng nhau do có những đoạn không dùng hết độ dài 64 Kbyte và vì thế các đoạn khác có thể bắt đầu nối tiếp ngay sau đó. Địa chỉ của ô nhớ nằm ở đầu đoạn được ghi trong một thanh ghi đoạn 16 bit, địa chỉ này gọi là *địa chỉ cơ sở*. Mười sáu bit này tương ứng với các đường dây địa chỉ từ A4 đến A20. Như vậy giá trị vật lý của địa chỉ đoạn là giá trị trong thanh ghi đoạn dịch sang trái 4 vị trí. Điều này tương đương với phép nhân với 2^4 = 16. Địa chỉ của các ô nhớ khác nằm trong đoạn tính được bằng cách cộng thêm vào địa chỉ cơ sở một giá trị gọi là địa chỉ lệch hay độ lệch (offset), gọi như thế vì nó ứng với khoảng lệch của toạ độ một ô nhớ cụ thể nào đó so với ô đầu đoạn. Độ lệch này được xác định bởi các thanh ghi 16 bit khác đóng vai trò thanh ghi lệch (offset register). Nguyên tắc này dẫn đến công thức tính địa chỉ vật lý (physical address) từ địa chỉ đoạn (segment) trong thanh ghi đoạn và địa chỉ lệch (offset) trong thanh ghi con trỏ như sau:

Địa chỉ vật lý = Thanh ghi đoạn x 16 + Thanh ghi lệch

- **Thanh ghi cờ**

Các cờ chỉ thị tình trạng của bộ vi xử lý cũng như điều khiển sự hoạt động của nó.

Một thanh ghi cờ là 1 flip-flop mà nó chỉ thị một số tình trạng được tạo bởi việc thực thi 1 lệnh hay các hoạt động điều khiển cụ thể của EU. Thanh ghi cờ 16-bit trong EU có 9 cờ.

- *Các cờ điều kiện - conditional flags:* Có 6 cờ được gọi là cờ điều kiện. Chúng được lập hay xoá là bởi EU, dựa trên kết quả của các phép toán số học.
- *Cờ điều khiển - control flags* : 3 cờ còn lại trong thanh ghi cờ được sử dụng để điều khiển một số hoạt động của vi xử lý. Chúng được gọi là các cờ điều khiển.

Bit pos	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Func	x	x	x	x	OF	DF	IF	TF	SF	ZF	x	AF	x	PF	x	CF

- Carry Flag (CF)- set by carry out of MSB.
- Parity Flag (PF)- set if result has even parity.
- Auxiliary carry Flag (AF)- for BCD
- Zero Flag (ZF)- set if results = 0

- Sign Flag (SF) = MSB of result
- TF- single step trap flag
- IF- interrupt enable flag
- DF- string direction flag
- Overflow **Flag (OF)-** overflow flag

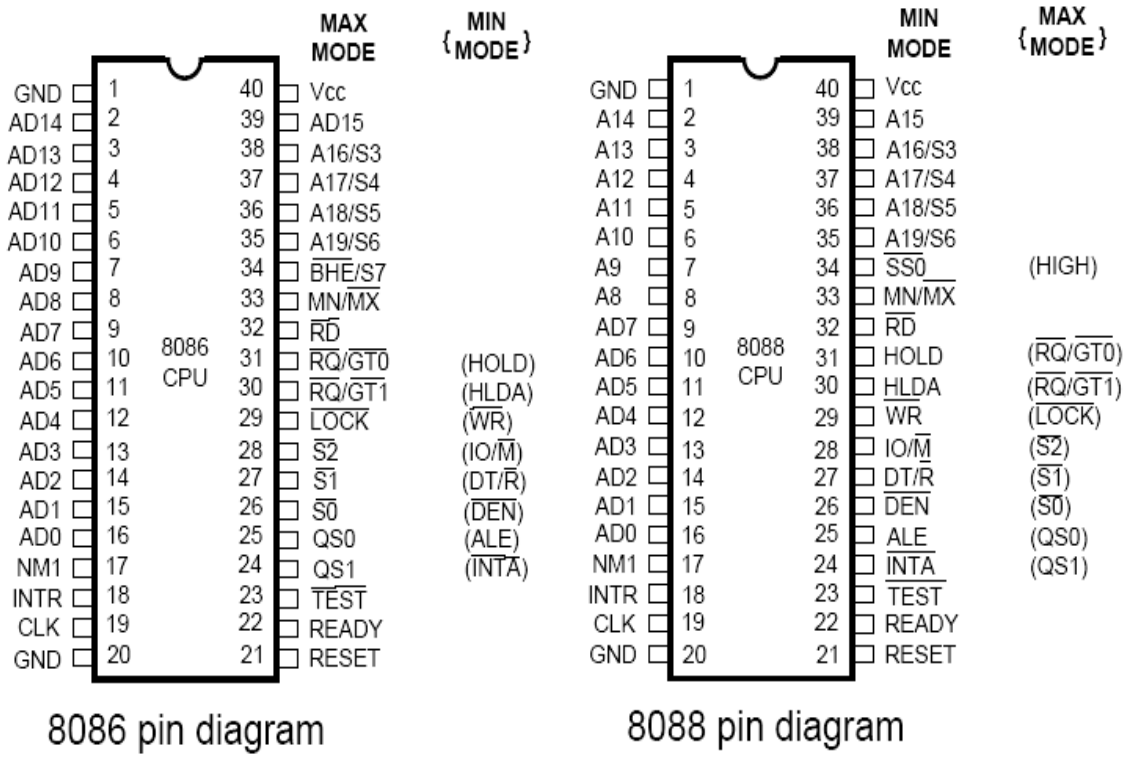
- **Các cờ điều kiện**
  - **Cờ nhớ - Carry flag (CF) –** Cờ này được đặt lên 1 khi tính toán một số không dấu bị tràn. Ví dụ khi cộng dạng byte: 255+1 (kết quả không nằm trong vùng 0..255). Khi không tràn, cờ này đặt bằng 0
  - **Cờ chẵn lẻ - parity flag (PF) –** Cờ PF=1 khi số lượng bit "1" trong kết quả là chẵn, PF=0 khi số lượng bit "1" là lẻ.
  - **Cờ nhớ phụ - auxiliary carry flag (AF)-** có ý nghĩa quan trọng đối với phép cộng và phép trừ các số BCD; AF=1 khi nhóm 4 bit thấp (không dấu) tràn. Chỉ được sử dụng với lệnh thao tác với số BCD.
  - **Cờ không - zero flag (ZF)-** chỉ thị rằng kết qủa của phép toán số học hay logic là bằng 0.
  - **Cờ dấu - sign flag (SF) -** chỉ thị dấu số học của kết quả sau 1 phép toán số học. Nếu số là âm (MSB=1) thì SF=1 và ngược lại SF=0 khi MSB=0
  - **Cờ tràn - overflow flag (OF)-** Cờ tràn OF=1 khi tính toán tràn số âm. Ví dụ khi tính bới 2 byte: 100+50 (kết quả ngoài khoảng -128..127)
- **Các cờ điều khiển**

Các cờ điều khiển được lập hay xoá thông qua các lệnh đặc biệt trong chương trình người dùng. Ba cờ điều khiển là:

- **Cờ bẫy - trap flag (TF)** – Khi cờ TF=1, CPU sẽ chờ ngắt từ thiết bị ngoài.
- **Cờ ngắt - interrupt flag (IF)** - được sử dụng để cho phép hay cấm ngắt của các chương trình;
- **Cờ hướng - direction flag (DF)** - được sử dụng với các lệnh chuỗi, mảng dữ liệu, nếu DF=0 thực thi theo hướng tiến, DF=1 thự thi theo hướng lùi.

Không có lệnh riêng để lập cờ TF.

2.1.3 Mô tả chức năng các chân



Hình 2-4. Sơ đồ chân 8086/8088

8088 và 8086 là gần tương tự như nhau, chỉ khác ở chỗ 8088 có 8bit dữ liệu còn 8086 có 16 bit dữ liệu ngoài.

Cả 2 bộ xử lý đều có:

- Độ rộng bus dữ liệu nội là 16 bit
- 20 đường địa chỉ (16 address/data + 4 address/status), cho phép địa chỉ hoá không gian bộ nhớ tối đa là 1Mbyte ở chế độ dồn kênh address/data pins (8088 only multiplexes 8 pins)
- 2 chế độ hoạt động (maximum và minimum mode)
- Cùng 1 tập lệnh

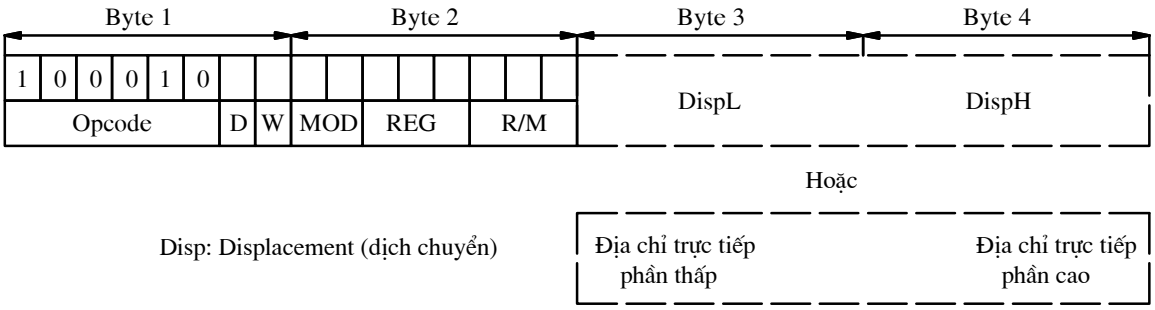
2.2 Chế độ địa chỉ

2.2.1 Khái niệm chế độ địa chỉ

Trước khi đi vào các chế độ địa chỉ của Vi xử lý 8086 ta nói qua về cách mã hoá lệnh trong vi xử lý 8086.

Lệnh của bộ vi xử lý được ghi bằng các ký tự dưới dạng gợi nhớ để người sử dụng dễ nhận biết. Đối với bản thân bộ vi xử lý thì lệnh cho nó được mã hoá dưới dạng các số 0 và 1 (còn gọi là mã máy) vì đó là dạng biểu diễn thông tin duy nhất mà máy có thể hiểu được. Vì lệnh cho bộ vi xử lý được cho dưới dạng mã nên sau khi nhận lệnh, bộ vi xử lý phải thực hiện giải mã lệnh rồi sau đó mới thực hiện lệnh

Một lệnh có thể có độ dài một vài byte tuỳ theo bộ vi xử lý. Đối với vi xử lý 8086 một lệnh có độ dài từ 1 đến 6 byte. Ta sẽ dùng lệnh **MOV** để giải thích cách ghi lệnh nói chung của 8086.



Dạng thức các byte mã lệnh của lệnh MOV

Từ đây ta thấy để mã hoá lệnh **MOV** cần ít nhất 2 byte. Trong đó 6 bit đầu dùng để chứa mã lệnh, 6 bit này luôn là 100010. đối với các thanh ghi đoạn thì điều này lại khác. Bit W dùng để chỉ ra rằng một byte (W=0) hoặc một từ (W=1) sẽ được chuyển đi. Trong thao tác chuyển dữ liệu, một toán hạng luôn bắt buộc phải là thanh ghi. Bộ vi xử lý sử dụng 2 hoặc 3 bit (REG) để mã hoá các thanh ghi trong CPU như sau:

Thanh ghi		Mã
W = 1	W = 0	
AX	AL	000
BX	BL	011
CX	CL	001
DX	DL	010
SP	AH	100
DI	BH	111
BP	CH	101
SI	DH	110

Thanh ghi đoạn	Mã
CS	01
DS	11
ES	00
SS	10

Bit D là hướng đi của dữ liệu. D = 1 thì dữ liệu đến thanh ghi, D = 0 thì dữ liệu đi ra từ thanh ghi.

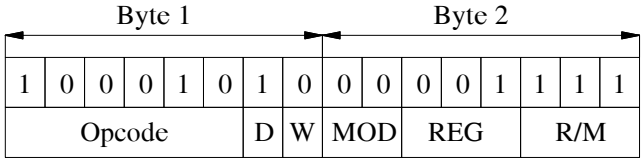
Hai bit MOD (chế độ) cùng với ba bit R/M (thanh ghi/bộ nhớ) tạo ra 5 bit dùng để chỉ ra chế độ địa chỉ cho các toán hạng của lệnh. Bảng 2.2 cho ta thấy cách mã hoá các chế độ địa chỉ.

MOD / R/M	00	01	10	11	
				W=0	W=1
000	[BX+SI]	[BX+SI]+d8	[BX+SI]+d16	AL	AX
001	[BX+DI]	[BX+DI]+d8	[BX+DI]+d16	CL	CX
010	[BP+SI]	[BP+SI]+d8	[BP+SI]+d16	DL	DX
011	[BP+DI]	[BP+DI]+d8	[BP+DI]+d16	BL	BX

MOD / R/M	00	01	10	11	
				W=0	W=1
100	[SI]	[SI]+d8	[SI]+d16	AH	SP
101	[DI]	[DI]+d8	[DI]+d16	CH	BP
110	D16(đ/c trực tiếp)	[BP]+d8	[BP]+d16	DH	SI
111	[BX]	[BX]+d8	[BX]+d16	BH	DI

Bảng 2-2.Phối hợp MOD và R/M để tạo ra các chế độ địa chỉ

Ví dụ 1: MOV CL, [BX]



Mã lệnh MOV: 100010

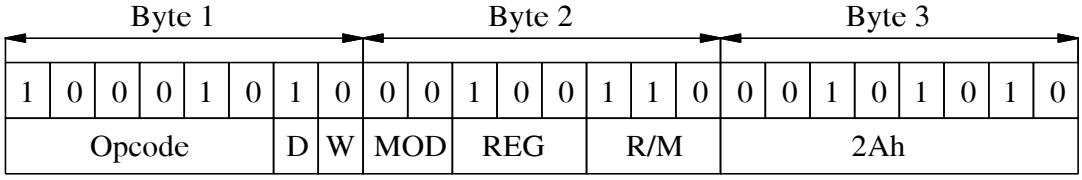
D = 1: Chuyển tới thanh ghi

W = 0: Chuyển 1 byte

MOD: ở chế độ 00 và R/M là 111

REG: 001 mã hoá CL

Ví dụ 2: MOV AH, 2Ah



Mã lệnh MOV: 100010

D = 1: Chuyển tới thanh ghi

W = 0: Chuyển 1 byte

MOD: ởy chế độ 00 và R/M là 110: Địa chỉ trực tiếp

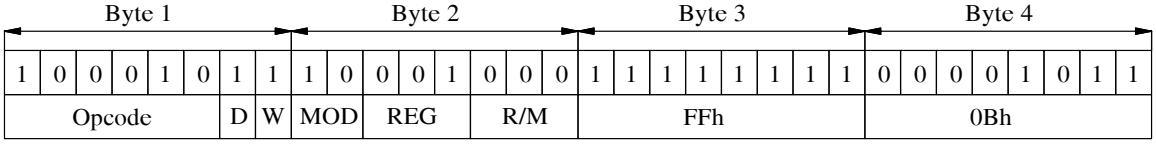
REG: 100 mã hoá AH

2Ah = 00101010 dữ liệu cần chuyển tới AH

Ví dụ 3: MOV CX, [BX][SI]+DATA

DATA là một biến trong bộ nhớ, đó là địa chỉ lệch và là một hằng (ví dụ như 0BFF).

Lệnh này sẽ sử dụng 4 byte tổ chức như sau:



Mã lệnh MOV: 100010

D = 1: Chuyển tới thanh ghi

W =1: Chuyển 1 Word

MOD: ở chế độ 10 (offset 16 bit) và R/M là 000 (sử dụng thanh ghi cơ sở BX và thanh ghi chỉ số SI).

REG: 001 mã hoá thanh ghi CX.

Như vậy trong ký hiệu nhị phân và hexa ta có.

Byte 1	Byte 2	Byte 3	Byte 4
10001011	10001000	11111111	00001011
8Bh	88h	FFh	0Bh

2.2.2 Các chế độ địa chỉ

Chế độ địa chỉ (addressing mode) là cách để CPU tìm thấy toán hạng cho các lệnh của nó khi hoạt động. Một bộ vi xử lý có thể có nhiều chế độ địa chỉ. Các chế độ địa chỉ này được xác định ngay từ khi chế tạo và không thể thay đổi được. Bộ vi xử lý 8086/8088 có 9 chế độ địa chỉ sau:

- Chế độ địa chỉ thanh ghi.
- Chế độ địa chỉ tức thì.
- Chế độ địa chỉ trực tiếp.
- Chế độ địa chỉ gián tiếp qua thanh ghi.
- Chế độ địa chỉ tương đối cơ sở.
- Chế độ địa chỉ tương đối chỉ số.
- Chế độ địa chỉ tương đối cơ sở chỉ số.
- Chế độ địa chỉ chuỗi (String) – mảng.
- Chế độ địa chỉ cổng (Port).
- Chế độ địa chỉ khác.

❖ CHẾ ĐỘ ĐỊA CHỈ THANH GHI

Trong chế độ địa chỉ này người ta sử dụng các thanh ghi có sẵn trong CPU như là các toán hạng để chứa dữ liệu cần thao tác, vì vậy khi thực hiện có thể đạt tốc độ truy nhập cao hơn so với các lệnh truy nhập đến bộ nhớ.

```
Ví dụ:
MOV   BX, DX      ;copy noi dung DX vao BX
ADD   AX, BX      ;AX=AX+BX
```

❖ CHẾ ĐỘ ĐỊA CHỈ TỨC THÌ

Trong chế độ này toán hạng đích là một thanh ghi hay một ô nhớ, còn toán hạng nguồn là một hằng số. Ta có thể dùng chế độ địa chỉ này để nạp dữ liệu cần thao tác vào bất kỳ thanh ghi nào (trừ thanh ghi đoạn và thanh ghi cờ) và bất kỳ ô nhớ nào trong đoạn dữ liệu DS.

Ví dụ:

```
MOV   CL, 100          ;chuyen 100 vao CL.
MOV   AX, 0BC8h        ;chuyen 0BC8h vao AX de roi
MOV   DS, AX           ;copy noi dung AX vao DS (vi
;khong duoc chuyen truc tiep vao thanh ghi doan).
MOV   [BX], 20   ;chuyen 20 vao o nho tai dia chi DS:BX.
```

❖ CHẾ ĐỘ ĐỊA CHỈ TRỰC TIẾP

Trong chế độ địa chỉ này một toán hạng chứa địa chỉ lệch của ô nhớ dùng chứa dữ liệu, còn toán hạng kia có thể là thanh ghi mà không được là ô nhớ.

Ví dụ:

```
MOV   AL, [0243H];chuyen noi dung o nho DS:0243 vao AL
MOV   [4320], CX ;chuyen noi dung CX vao hai o nho
                           ;lien tiep DS:4320 va DS:4321
```

❖ CHẾ ĐỘ ĐỊA CHỈ GIÁN TIẾP QUA THANH GHI

Trong chế độ địa chỉ này một toán hạng là một thanh ghi được sử dụng để chứa địa chỉ lệch của ô nhớ dữ liệu, còn toán hạng kia chỉ có thể là thanh ghi mà không được là ô nhớ. *Ví dụ:*

```
MOV   AL, [BX]    ;copy noi dung o nho co dia chi DS:BX
MOV   [SI], CL    ;copy noi dung CL vao o nho co dia ch
;DS:SI
MOV   [DI], AX    ;copy noi dung AX vao hai o nho lien
                        ;tiep co dia chi DS:DI va DS:(DI+1)
```

❖ CHẾ ĐỘ ĐỊA CHỈ TƯƠNG ĐỐI CƠ SỞ

Trong chế độ địa chỉ này các thanh ghi cơ sở như BX và BP và các hằng số biểu diễn các giá trị dịch chuyển được dùng để tính địa chỉ hiệu dụng của toán hạng trong các vùng nhớ DS và SS. *Ví dụ:*

```
MOV   CX, [BX]+10      ;copy noi dung hai o nho lien tiep
;co dia chi DS:BX+10 va DS:BX+11
;vao CX
MOV   CX, [BX+10]      ;cach viet khac cua lenh tren
MOV   CX, 10+[BX]      ;cach viet khac cua lenh tren
MOV   AL, [BP]+5 ;chuyen noi dung o nho co dia chi
;SS:BP+5 vao AL
```

Quan sát trên ta thấy: 10 và 5 là các dịch chuyển của các toán hạng tương ứng. BX+10, BP+5 gọi là địa chỉ hiệu dụng.

DS:BX+10, SS:BP+5 chính là địa chỉ logic ứng với địa chỉ vật lý.

❖ CHẾ ĐỘ ĐỊA CHỈ TƯƠNG ĐỐI CHỈ SỐ

Trong chế độ địa chỉ này các thanh ghi chỉ số như SI và DI và các hằng số biểu diễn các giá trị dịch chuyển được dùng để tính địa chỉ hiệu dụng của toán hạng trong các vùng nhớ DS. *Ví dụ*

```
MOV   CX, [SI]+10      ;copy noi dung hai o nho lien tiep
                  ;co dia chi DS:SI+10 va DS:SI+11 vao CX
MOV   CX, [SI +10]     ;cach viet khac cua lenh tren
MOV   CX, 10+[SI]      ;cach viet khac cua lenh tren
MOV   AL, [DI]+5 ;chuyen noi dung o nho co dia chi
;DS:DI+5 vao AL
```

❖ CHẾ ĐỘ ĐỊA CHỈ TƯƠNG ĐỐI CHỈ SỐ CƠ SỞ

Kết hợp hai chế độ địa chỉ chỉ số và cơ sở ta có chế độ địa chỉ chỉ số cơ sở. Trong chế độ này ta dùng cả hai thanh ghi cơ sở lẫn thanh ghi chỉ số để tính địa chỉ của toán hạng. Nếu ta dùng thêm cả thành phần biểu diễn sự dịch chuyển của địa chỉ thì ta có chế độ địa chỉ tổng hợp nhất: Chế độ địa chỉ tương đối chỉ số cơ sở.

```
Ví dụ:   MOV    BX, [BX]+[SI]+10 ;chuyen noi dung hai o nho
;lien tiep co dia chi DS:BX+SI+10 va DS:BX+SI+11 vao CX
MOV   AL, [BP+DI+5]    ;copy nội dung ô thứ: DS:BP+DI+5 vao AL
```

Các chế độ địa chỉ đã trình bày ở trên có thể tóm tắt lại trong bảng sau:

Chế độ địa chỉ	Toán hạng	Thanh ghi đoạn ngầm định
Thanh ghi	Reg	
Tức thì	Data	
Trực tiếp	[offset]	DS
Gián tiếp qua thanh ghi	[BX]	DS
	[SI]	DS
	[DI]	DS
Tương đối cơ sở	[BX]+Disp	DS
	[BP]+Disp	SS
Tương đối chỉ số	[DI]+Disp	DS
	[SI]+Disp	DS
Tương đối chỉ số cơ sở	[BX]+[DI]+Disp	DS
	[BX]+[SI]+Disp	DS
	[BP]+[DI]+Disp	SS
	[BP]+[SI]+Disp	SS

Bảng 2-3. Các chế độ địa chỉ

Chú ý: Reg: Thanh ghi, Data: Dữ liệu tức thì, Disp: Dịch chuyển.

❖ CHẾ ĐỘ ĐỊA CHỈ CHUỖI (STRING) – MẢNG

Một chuỗi (string) là một dãy các byte hoặc word liên tiếp trong bộ nhớ. Các lệnh thao tác với chuỗi không sử dụng bất kỳ một chế độ địa chỉ nào ở trên. Một chuỗi có thể có độ dài tối đa lên tới 64K-bytes (một segments). Chế độ địa chỉ chuỗi sử dụng các thanh ghi SI, DI, DS và ES. Với tất cả các lệnh thao tác chuỗi đều sử dụng SI để trỏ vào byte đầu tiên của chuỗi nguồn và DI trỏ vào byte đầu tiên của chuỗi đích.

```
Ví dụ: Giả sử: DS=1000h, ES=2000h, SI=10h, DI=20h)
MOVSB       ;Sao chép chuỗi từ 10010h đến 20020h
```

❖ CHẾ ĐỘ ĐỊA CHỈ CỔNG (PORT)

Trong họ vi xử lý 80x86 của Intel có không gian địa chỉ cho bộ nhớ và cổng vào/ra là tách biệt nhau. Không gian địa chỉ cổng có thể lên đến 65536 cổng (64K-ports).

Địa chỉ của một cổng có thể được xác định bởi một hằng giá trị kiểu byte (phạm vi = 0..255)

```
Ví dụ:
IN          AL, 40h     ;Đọc cổng – sao chép nội dung tại
;cổng có địa chỉ 40h và thanh ghi AL
OUT   80h, AL     ;Ghi cổng – gửi dữ liệu trong thanh
;ghi AL tới cổng có địa chỉ 80h
```

Địa chỉ của cổng cũng có thể được xác định gián tiếp qua thanh ghi (Khi này phạm vi tối đã sẽ là 65536 cổng).

Ví dụ:

```
IN    AL, DX           ;Đọc cổng có địa chỉ là nội dung của
;thanh ghi DX
OUT   DX, AX      ;Ghi một word trong AX tới cổng có địa
;chỉ là nội dung của thanh ghi DX.
```

2.3 Tập lệnh Assembly

2.3.1 Giới thiệu chung

Tập lệnh của họ vi xử lý 80x86 đảm bảo tương thích thế hệ sau với thế hệ trước. điều đó có nghĩa là các chương trình viết cho 8086 vẫn chạy được trên các bộ vi xử lý mới hơn mà không phải thay đổi (không đảm bảo thứ tự ngược lại). Tập lệnh của một bộ vi xử lý thường có rất nhiều lệnh (hàng trăm lệnh), vì thế mà việc tiếp cận và làm chủ chúng là trương đối khó khăn.

Có nhiều cách trình bày tập lệnh của bộ vi xử lý: Trình bày theo nhóm lệnh hoặc theo thứ tự abc. Để có thể nhanh chóng và dễ dàng sử dụng các lệnh cơ bản và lập trình được ngay, ta sẽ tiếp cận tập lệnh của bộ vi xử lý theo nhóm các thao tác cơ bản trong quá trình xử lý và điều khiển. Với mỗi thao tác nói trên, ta làm quen với một vài lệnh tiêu biểu (độc giả có thể tra cứu thêm các lệnh khác trong phần phụ lục). Các chức năng cơ bản của một bộ vi xử lý thường gồm:

- Nhóm các lệnh vận chuyển (sao chép) dữ liệu.
- Nhóm các lệnh tính toán số học.
- Nhóm các lệnh tính toán logic.
- Nhóm các lệnh dịch, quay toán hạng.
- Nhóm các lệnh nhảy (rẽ nhánh).

- Nhóm các lệnh lặp.
- Nhóm các lệnh điều khiển, đặc biệt khác.

2.3.2 Các nhóm lệnh

2.3.2.1 Nhóm các lệnh vận chuyển (sao chép) dữ liệu

1. MOV – MOV a byte or word (chuyển một byte hay từ)

Dạng lệnh: MOV Đích, Nguồn

Mô tả: Đích←Nguồn

Trong đó toán hạng đích và Nguồn có thể tìm được theo các chế độ địa chỉ khác nhau, nhưng phải có cùng độ dài và không được phép đồng thời là hai ô nhớ hoặc hai thanh ghi đoạn.

Các cờ bị thay đổi: không.

Ví dụ:

```
MOV   AL, AH      ;AL←AH
MOV   CX, 50      ;CX←50
MOV   DL, [SI]    ;DL←{DS:SI}
```

2. OUT – Output a byte or a work to a port.

Dạng lệnh: OUT Port, Acc

Mô tả: Acc→{Port}

Trong đó {port} là dữ liệu của cổng có địa chỉ port. Port là địa chỉ 8 bit của cổng, nó có thể là các giá trị trong khoảng 00...FFH. Như vậy có thể có các khả năng sau đây.

- Nếu Acc là AL thì dữ liệu 8 bit được đưa ra cổng Port.
- Nếu Acc là AX thì dữ liệu 16 bit được đưa ra cổng Port và Port + 1.

Có một cách khác để chứa địa chỉ cổng là thông qua thanh ghi DX. Khi dùng thanh ghi DX để chứa địa chỉ cổng ta có khả năng địa chỉ hoá cổng mềm dẻo hơn. Lúc này địa chỉ cổng nằm trong dải 0000H … FFFFH và viết lệnh theo dạng:

OUT DX, Acc

Các cờ bị thay đổi: không.

Ví dụ:

```
OUT   45H, AL     ;dua du lieu tu AL ra cong 45H
MOV   DX, 0 ;xoa DX
MOV   DX, 00FFH   ;nap dia chi cong vao DX
OUT   DX, AX      ;dua du lieu tu AX ra 00FFH
```

3. IN – Input data from a port (đọc dữ liệu từ cổng vào thanh ghi Acc).

Dạng lệnh: IN Acc, địa_chỉ_cổng

Lệnh IN truyền một byte hoặc một từ từ một cổng vào lần lượt tới thanh ghi AL hoặc AX. Địa chỉ của cổng có thể được xác định là một hằng tức thì kiểu byte cho phép truy nhập các cổng từ 0…255 hoặc thông qua một số đã được đưa ra trước đó trong thanh ghi DX mà cho phép truy nhập các cổng từ 0…65535.

Các cờ bị thay đổi: không.

Ví dụ:

```
IN    AL, 45H    ;doc mot byte tu mot cong duoc xac
;dinh trong che do tuc thi
IN    AX, 0046H  ;doc hai byte tu mot cong duoc xac
;dinh trong che do tuc thi
IN    AX, DX     ;doc mot tu tu mot cong dang bien
```

4. POP – Pop word from top of Stack (lấy lại 1 từ vào thanh ghi từ đỉnh ngăn xếp)

Dạng lệnh: POP Đích

Mô tả:

Đích←{SP}

SP←SP+2

Toán hạng đích đích có thể là các thanh ghi đa năng, thanh ghi đoạn (nhưng không được là thanh ghi đoạn mã CS) hoặc ô nhớ.

Các cờ bị thay đổi: không.

Ví dụ:

```
POP   DX    ;lay 2 byte tu dinh ngan xep dua vao DX
```

5. PUSH – Push word on the Stack (cất 1 từ vào ngăn xếp)

Dạng lệnh: PUSH Nguồn

Mô tả:

SP←SP-2

Nguồn→{SP}

Toán hạng đích đích có thể là các thanh ghi đa năng, thanh ghi đoạn(kể cả CS) hoặc ô nhớ.

Các cờ bị thay đổi: không.

Ví dụ:

```
PUSH   BX
;cat BX vao ngan xep tai vi tri do SP chi ra
```

2.3.2.2 Nhóm các lệnh tính toán số học

6. ADC – Add with Carry (cộng có nhớ)

Dạng lệnh: ADC Đích, Nguồn

Mô tả: Đích ← Đích + Nguồn + CF

Cộng hai toán hạng Đích và Nguồn với cờ CF kết quả lưu vào Đích.

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF.

Ví dụ:

```
ADC   AL, 74H     ;AL←AL+74+CF
ADC   CL, BL      ;CL←CL+BL+CF
ADC   DL, [SI]    ;DL←DL+(DS:SI)+CF
```

7. ADD – Add (cộng hai toán hạng)

Dạng lệnh: ADD Đích, Nguồn

Mô tả: Đích ← Đích + Nguồn

Cộng hai toán hạng đích và Nguồn kết quả lưu vào đích.

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF.

Ví dụ:

```
ADD   DX, CX      ;DX←DX+CX
ADD   AX, 400     ;AX←AX+400
```

8. DEC – Decrement (giảm byte hay word đi một giá trị)

Dạng lệnh: DEC Đích

DEC trừ toán hạng Đích đi 1. Toán hạng Đích có thể là byte hay word.

Các cờ bị thay đổi: AF, OF, PF, SF, ZF.

Ví dụ:

```
MOV   BX, 1200H  ;chuyen 1200H vao BX
DEC   BX         ;BX=11FFH
```

9. DIV – Division (chia không dấu)

Dạng lệnh: DIV Nguồn

Toán hạng Nguồn là số chia. Tuỳ theo độ dài toán hạng Nguồn ta có hai trường hợp bố trí phép chia.

- Nếu Nguồn là là số 8 bit: AX/Nguồn, thương để vào AL, số dư để vào AH
- Nếu Nguồn là số 16 bit: DXAX/Nguồn, thương để vào AX, số dư để vào DX

Nếu thương không phải là số nguyên nó được làm tròn theo số nguyên sát dưới. Nếu Nguồn bằng 0 hoặc thương thu được lớn hơn FFH hoặc FFFFH (tuỳ theo độ dài của toán hạng Nguồn) thì 8086 thực hiện lệnh ngắt INT 0.

Các cờ bị thay đổi: không.

Ví dụ:

```
MOV   AX, 0033H  ;chuyen 0033H vao AX
MOV BL, 25
DIV   BL         ;AL=02H va AH=01H
```

10. INC – Increment (tăng toán hạng lên 1)

Dạng lệnh: INC đích

Mô tả: Đích ← Đích + 1

Lệnh này tăng đích lên 1, tương đương với việc ADD đích, 1 nhưng chạy nhanh hơn.

Các cờ bị thay đổi: AF, OF, PF, SF, ZF.

Ví dụ:

```
INC   AL
INC   BX
```

11. MUL – Multiply unsigned byte or word (nhân số không dấu)

Dạng lệnh: MUL Nguồn

Thực hiện phép nhân không dấu với toán hạng Nguồn (ô nhớ hoặc thanh ghi) với thanh ghi tổng.

- Nếu Nguồn là số 8 bit: AL*Nguồn. Số bị nhân phải là số 8 bit đặt trong AL, sau khi nhân tích lưu vào AX
- Nếu Nguồn là số 16 bit: AX*Nguồn. Số bị nhân phải là số 16 bit đặt trong AX, sau khi nhân tích lưu vào DXAX.

Nếu byte cao (hoặc 16 bit cao) của 16 (hoặc 32) bit kết quả chứa 0 thì CF=OF=0.

Các cờ bị thay đổi: CF, OF.

Ví dụ:

```
MUL   CX    ;AXxCX → DXAX
MUL   BL    ;ALxBL → AX
```

12. NEG – Negation (lấy bù hai của một toán hạng, đảo dấu của một toán hạng).

Dạng lệnh: NEG Đích

Mô tả: Đích←0-Đích

NEG lấy 0 trừ cho đích (có thể là 1 byte hoặc 1 từ) và trả lại kết quả cho toán hạng đích, nếu ta lấy bù hai của -128 hoặc -32768 ta sẽ được kết quả không đổi nhưng OF=1 để báo là kết quả bị tràn vì số dương lớn nhất biểu diễn được là +127 và +32767.

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF

Ví dụ:

```
NEG   AL    ;AL←0-(AL)
```

13. SUB – Substract (trừ hai toán hạng)

Dạng lệnh: SUB Đích, Nguồn

Mô tả: Đích←Đích - Nguồn

Toán hạng đích vào Nguồn phải chứa cùng một loại dữ liệu và không được đồng thời là hai ô nhớ, cũng không được là thanh ghi đoạn.

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF.

Ví dụ:

```
SUB   AL, 78H     ;AL←AL-78H
SUB   BL, CL      ;BL←BL-CL
SUB   DL, [SI]    ;DL←DL-{DS:SI}
```

2.3.2.3 Nhóm các lệnh tính toán logic

14. AND (phép và logic)

Dạng lệnh: AND Đích, Nguồn

Mô tả: Đích ← Đích ^ Nguồn

Thực hiện phép và logic hai toán hạng và lưu kết quả vào toán hạng đích. Người ta thường sử dụng để che đi/giữ lại một vài bit nào đó của một toán hạng bằng cách nhân logic toán hạng đó với toán hạng tức thì có các bit 0/1 ở các vị trí cần che đi/giữ lại tương ứng.

Các cờ bị thay đổi: CF, OF, PF, SF, ZF.

Ví dụ:

```
AND   DX, CX      ;DX←DX AND CX theo tung bit
AND   AL, 0FH     ;che 4 bit cao cua AL
```

15. NOT – Logical Negation (phủ định logic)

Dạng lệnh: NOT Đích

NOT đảo các giá trị của các bit của toán hạng đích.

Các cờ bị thay đổi: không.

Ví dụ:

```
MOV   AL, 02H     ;AL=(0000 0010)B
NOT   AL          ;AL=(1111 1101)B
```

16. OR – Logic OR (phép hoặc logic)

Dạng lệnh: OR Đích, Nguồn

Mô tả: Đích = Đích ∨ Nguồn

Toán hạng Đích và Nguồn phải chứa dữ liệu cùng độ dài và không được phép đồng thời là hai ô nhớ và cũng không được là thanh ghi đoạn. Phép OR thường dùng để lập một vài bit nào đó của toán hạng bằng cách cộng logic toán hạng đó với các toán hạng tức thời có các bit 1 tại vị trí tương ứng cần thiết lập.

Các cờ bị thay đổi: CF, OF, PF, SF, ZF.

Ví dụ:

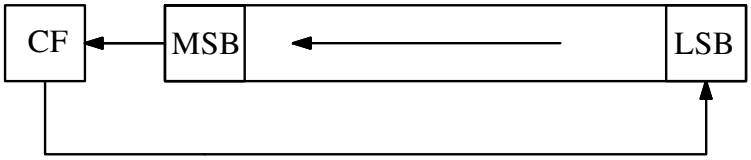
```
OR    AX, BX      ;AX←AX∨BX theo tung bit
OR    CL, 30H     ;lap bit b4 va b5 cua CL len 1
```

2.3.2.4 Nhóm các lệnh dịch, quay toán hạng

17. RCL – Rotate though CF to the Left (quay trái thông qua cờ nhớ)

Dạng lệnh: RCL Đích, CL

Mô tả:



Lệnh này để quay toán hạng sang trái thông qua cờ CF, CL phải được chứa sẵn số lần quay. Trong trường hợp quay 1 lần có thể viết RCL Đích, 1

Nếu số lần quay là 9 thì toán hạng không đổi vì cặp CF và toán hạng quay đúng một vòng (nếu toán hạng đích là 8 bit).

Sau lệnh RCL cờ CF mang giá trị cũ của MSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được xác định sau nhiều lần quay.

Các cờ bị thay đổi: CF, OF, SF, ZF, PF. *Ví dụ:*

```
MOV   CL, 3 ;so lan quay la 3
RCL   AL, CL
```

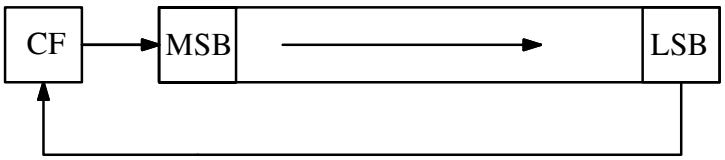
Trước khi thực hiện lệnh: AL = 01011110, CF = 0.

Sau khi thực hiện lệnh: AL = 11110001, CF = 0.

18. RCR – Rotate though CF to the Right (quay phải thông qua cờ nhớ)

Dạng lệnh: RCR Đích, CL

Mô tả:



Lệnh này để quay toán hạng sang phải thông qua cờ CF, CL phải được chứa sẵn số lần quay. Trong trường hợp quay 1 lần có thể viết RCR Đích, 1

Nếu số lần quay là 9 thì toán hạng không đổi vì cặp CF và toán hạng quay đúng một vòng (nếu toán hạng đích là 8 bit).

Sau lệnh RCR cờ CF mang giá trị cũ của LSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được xác định sau nhiều lần quay.

Các cờ bị thay đổi: CF, OF, SF, ZF, PF.

Ví dụ:

```
MOV   CL, 2 ;so lan quay la 2
RCR   AL, CL
```

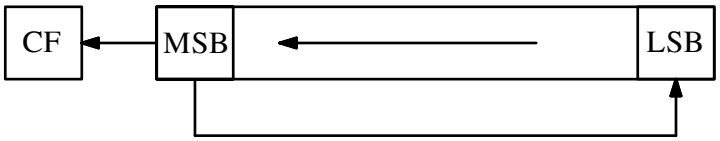
Trước khi thực hiện lệnh: AL = 11000010, CF = 1.

Sau khi thực hiện lệnh: AL = 01110000, CF = 1.

19. ROL – Rotate all bit to the Left (quay vòng sang trái).

Dạng lệnh: ROL Đích, CL.

Mô tả:



Lệnh này dùng để quay vòng toán hạng sang trái, MSB được đưa sang cờ CF và LSB. CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết ROL Đích, 1. Nếu số lần quay là 8 (CL=8) thì toán hạng không đổi vì toán hạng quay đúng một vòng (nếu toán hạng đích là 8 bit), còn nếu CL=4 thì 4 bit cao đổi chỗ cho 4 bit thấp.

Sau lệnh ROL cờ CF mang giá trị cũ của MSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được xác định sau nhiều lần quay. Lệnh này thường dùng để tạo cờ CF từ giá trị của MSB làm điều kiện cho lệnh nhảy có điều kiện.

Các cờ bị thay đổi: CF, OF, SF, ZF, PF.

Ví dụ:

```
MOV   CL, 2 ;so lan quay la 2
ROL   AL, CL
```

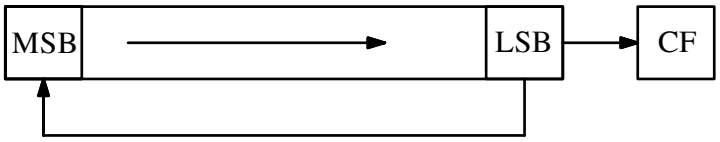
Trước khi thực hiện lệnh: AL = 11001100, CF = 1

Sau khi thực hiện lệnh: AL = 00110011, CF = 1

20. ROR – Rotate all bit to the Left (quay vòng sang phải).

Dạng lệnh: ROR Đích, CL

Mô tả:



Lệnh này dùng để quay vòng toán hạng sang phải, LSB được đưa sang cờ CF và MSB. CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết ROR Đích, 1. Nếu số lần quay là 8 (CL=8) thì toán hạng không đổi vì toán hạng quay đúng một vòng (nếu toán hạng đích là 8 bit), còn nếu CL=4 thì 4 bit cao đổi chỗ cho 4 bit thấp.

Sau lệnh ROR cờ CF mang giá trị cũ của LSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được

xác định sau nhiều lần quay. Lệnh này thường dùng để tạo cờ CF từ giá trị của LSB làm điều kiện cho lệnh nhảy có điều kiện.

Các cờ bị thay đổi: CF, OF, SF, ZF, PF.

Ví dụ:

```
MOV   CL, 2 ;so lan quay la 2
ROR   AL, CL
```

Trước khi thực hiện lệnh: AL = 11001100, CF = 0

Sau khi thực hiện lệnh: AL = 00110011, CF = 0

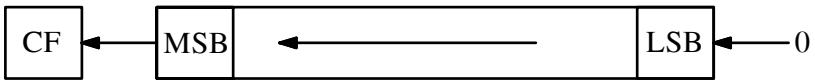
21. SAL/SHL - Shift Arithmetically Left (dịch trái số học)/Shift Logically Left (dịch trái logic).

Dạng lệnh:

SAL Đích, CL

SHL Đích, CL

Mô tả:



Hai lệnh này có tác dụng dịch trái số học toán hạng (còn gọi là dịch trái logic). Mỗi lần dịch MSB được đưa vào CF còn 0 được đưa vào LSB. CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết SAL Đích, 1

Sau lệnh SAL hoặc SHL cờ CF mang giá trị cũ của MSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được xác định sau nhiều lần quay. Lệnh này thường dùng để tạo cờ CF từ giá trị của MSB làm điều kiện cho lệnh nhảy có điều kiện.

Các cờ bị thay đổi: SF, ZF, CF, OF, PF.

Ví dụ:

```
MOV   CL, 2 ;so lan quay la 2
SAL   AL, CL
```

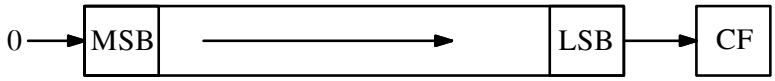
Trước khi thực hiện lệnh: AL = 11001100, CF = 0

Sau khi thực hiện lệnh: AL = 00110000, CF = 1

22. SHR – Shift logically Right (dịch phải logic)

Dạng lệnh: SHR Đích, CL

Mô tả:



Lệnh này có tác dụng dịch phải logic toán hạng. Mỗi lần dịch LSB được đưa vào CF còn 0 được đưa vào MSB. CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết SHR Đích, 1

Sau lệnh SHR cờ CF mang giá trị cũ của LSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được xác định sau nhiều lần quay. Lệnh này thường dùng để tạo cờ CF từ giá trị của LSB làm điều kiện cho lệnh nhảy có điều kiện.

Các cờ bị thay đổi: SF, ZF, CF, OF, PF.

Ví dụ:

```
MOV   CL, 2 ;so lan quay la 2
SHR   AL, CL
```

Trước khi thực hiện lệnh: AL = 11001100, CF = 1

Sau khi thực hiện lệnh: AL = 00110011, CF = 0

23. XOR – Exclusive OR (lệnh logic XOR (hoặc đảo)).

Dạng lệnh: XOR Đích, Nguồn

Mô tả: Đích←Đích⊕Nguồn.

Lệnh XOR thực hiện logic XOR (hoặc đảo) giữa hai toán hạng và kết quả được lưu vào trong đích, một bit kết quả được đặt bằng 1 nếu nếu các bit tương ứng hai toán hạng là đối nhau. Nếu toán hạng đích trùng toán hạng Nguồn thì kết quả bằng 0, do đó lệnh này còn được dùng để xoá thanh ghi về 0 kèm theo các cờ CF và OF cũng bị xoá.

Các cờ bị thay đổi: CF, OF, PF, SF, ZF.

Ví dụ:

```
XOR   AX, AX
XOR   BX, BX
MOV   AX, 5857H
MOV   BX, 58A8H
XOR   AX, BX
```

Trước khi thực hiện lệnh XOR	Sau khi thực hiện lệnh XOR
AX=5857H	AX=00FFH
BX=58A8H	BX=58A8H

2.3.2.5 Nhóm các lệnh so sánh

24. CMP – Compare (so sánh)

Dạng lệnh: CMP đích, Nguồn

CMP trừ toán hạng đích cho toán hạng Nguồn, chúng có thể là các byte hoặc các từ, nhưng không lưu trữ kết quả. Các toán hạng không bị thay đổi. Kết quả của

lệnh này dùng để cập nhật các cờ và có thể được dùng để làm điều kiện cho các lệnh nhảy có điều kiện tiếp theo.

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF.

Các cờ chính theo quan hệ đích và Nguồn khi so sánh hai số không dấu.

So sánh	CF	ZF
Đích = Nguồn	0	1
Đích > Nguồn	0	0
Đích < Nguồn	1	0

2.3.2.6 Nhóm các lệnh nhảy (rẽ nhánh)

25. JA/JNBE – Jump if Above/Jump if Not Below or Equal (nhảy nếu cao hơn/nhảy nếu không thấp hơn hoặc bằng).

Dạng lệnh:

JA NHAN

JNBE NHAN

Mô tả: IP←IP+dịch chuyển

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu CF + ZF = 0. Quan hệ cao hơn/thấp là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn hai số không dấu. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JA/JNBE. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
CMP   AX, 12ABH        ;so sanh AX voi 12ABH
JA    THOI             ;nhay den THOI neu AX cao hon 12ABH
```

26. JAE/JNB/JNC – Jump if Above or Equal/Jump if Not Below/Jump if No Carry (nhảy nếu lớn hơn hoặc bằng/nhảy nếu không thấp hơn/nhảy nếu không có nhớ).

Dạng lệnh:

JAE NHAN

JNB NHAN

JNC NHAN

Mô tả: IP←IP+dịch chuyển

Ba lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu CF = 0. Quan hệ cao hơn/thấp là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn hai số không dấu. NHAN phải nằm cách xa một khoảng

-128…+127 byte so với lệnh tiếp theo sau lệnh JAE/JNB/JNC. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
CMP   AL, 10H     ;so sanh AL voi 10H
JAE   THOI        ;nhay den THOI neu AL cao hon hoac bang 10H
```

27. **JB/JC/JNAE** – Jump if Below/Jump if Carry/Jump if Not Above or Equal (nhảy nếu thấp hơn/nhảy nếu có nhớ/nhảy nếu không cao hơn hoặc bằng).

Dạng lệnh:

JB NHAN

JC NHAN

JNAE NHAN

Mô tả: IP←IP+dịch chuyển

Ba lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu CF = 1. Quan hệ cao hơn/thấp là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn hai số không dấu. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JB/JC/JNAE. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
CMP   AL, 10H          ;so sanh AL voi 10H
JB    THOI             ;nhay den THOI neu AL thap hon 10H
```

28. **JBE/JNA** – Jump if Below or Equal/Jump if Not Above (nhảy nếu thấp hơn hoặc bằng/nhảy nếu không cao hơn).

Dạng lệnh:

JBE NHAN

JNA NHAN

Mô tả: IP←IP+dịch chuyển

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu

CF +ZF = 1. Quan hệ cao hơn/thấp là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn hai số không dấu. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JBE/JNA. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không. *Ví dụ:*

```
CMP   AL, 10H     ;so sanh AL voi 10H
JBE   THOI        ;nhay den THOI neu AL thap hon hoac
;bang 10H
```

29. JE/JZ – Jump if Equal/Jump if Zero (nhảy nếu bằng nhau/nhảy nếu kết quả bằng không)

Dạng lệnh:

JE NHAN

JZ NHAN

Mô tả: IP←IP+dịch chuyển

Lệnh trên biểu diễn thao tác nhảy có điều kiện tới NHAN nếu ZF = 1. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JE/JZ. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
SUB   AL, 10H          ;tru AL cho 10H
JE    THOI             ;nhay den THOI neu AL bang 10H
```

30. JMP – Unconditional Jump (lệnh nhảy không điều kiện).

JMP trao quyền điều khiển cho vùng mục tiêu một cách không điều kiện. Lệnh này có các chế độ giống như lệnh CALL và nó cũng phân biệt nhảy gần, nhảy xa.

Dạng lệnh: Sau đây là những cách viết lệnh không điều kiện.

JMP NHAN

Lệnh mới này bắt đầu địa chỉ ứng với NHAN. Chương trình sẽ căn cứ vào khoảng dịch giữa NHAN và lệnh nhảy để xác định xem nó là:

+ Nhảy ngắn: Trong trường hợp này NHAN phải nằm cách xa (dịch đi một khoảng).

-128…127 byte so với lệnh tiếp theo sau lệnh JMP. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển. Do đó

IP←IP+dịch chuyển

Đây là lệnh nhảy trực tiếp vì dịch chuyển để trực tiếp trong mã lệnh.

Để định hướng cho chương trình dịch làm việc nên viết lệnh dưới dạng:

JMP SHORT NHAN

+ Nhảy gần: Trong trường hợp này NHAN phải nằm cách xa (dịch đi một khoảng)

-32768…+32767 byte so với lệnh tiếp theo sau lệnh JMP. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển. Do đó

IP←IP+dịch chuyển

Đây là lệnh nhảy trực tiếp vì dịch chuyển để trực tiếp trong mã lệnh.

Để định hướng cho chương trình dịch làm việc nên viết lệnh dưới dạng:

JMP NEAR NHAN

+ Nhảy xa: Trong trường hợp này NHAN nằm ở đoạn mã khác so với lệnh tiếp theo sau lệnh JMP. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị địa chỉ nhảy đến (CS:IP của NHAN). Sau đó:

IP←IP của NHAN

CS←CS của NHAN

JMP BX

Đây là lệnh nhảy gần, trước đó BX phải chứa địa chỉ lệch của lệnh định nhảy đến trong đoạn CS. Khi thực hiện lệnh này thì IP←BX. Đây là lệnh nhảy gián tiếp vì địa chỉ lệch nằm trong thanh ghi. Để định hướng cho chương trình dịch làm việc ta nên viết lệnh dưới dạng:

JMP NEAR PTR BX

JMP [BX]

Đây là lệnh nhảy gần. IP mới được lấy từ nội dung 2 ô nhớ do BX và BX+1 chỉ ra trong đoạn DS (SI, DI có thể dùng thay chỗ của BX). Đây là lệnh nhảy gián tiếp vì địa chỉ lệch để trong ô nhớ. Để định hướng cho chương trình dịch làm việc ta nên viết lệnh dưới dạng:

JMP WORD PTR [BX]

Một biến dạng khác của lệnh trên thu được khi ta viết lệnh dưới dạng:

JMP DWORD PTR [BX]

Đây là lệnh nhảy xa. Địa chỉ nhảy đến ứng với CS:IP. Giá trị gán cho IP và CS được chứa trong 4 ô nhớ do BX và BX+1 (cho IP), BX+2 và BX+3 cho (CS) chỉ ra trong đoạn DS (SI, DI có thể sử dụng thay chỗ của BX)

Đây cũng là lệnh nhảy gián tiếp vì địa chỉ lệch và địa chỉ cơ sở nằm trong ô nhớ.

Các cờ bị thay đổi: không.

31. JNE/JNZ – Jump if Not Equal/Jump if Not Zero (nhảy nếu không bằng nhau/nhảy nếu kết quả không rỗng).

Dạng lệnh:

JNE NHAN

JNZ NHAN

Mô tả: IP←IP+dịch chuyển

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu ZF = 0. NHãN phải nằm cách xa (dịch đi một khoảng) -128…127 byte so với lệnh tiếp theo sau lệnh JNE/JNZ. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định độ dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
CMP   AL, 10H          ;so sanh AL voi 10H
JNE   THOI             ;nhay den THOI neu AL khac 10H
```

2.3.2.7 Nhóm các lệnh lặp

32. LOOP – Loop if CX is not 0 (lặp nếu CX ≠ 0)

Dạng lệnh: LOOP NHAN

Mô tả: Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ NHAN đến hết lệnh LOOP NHAN) cho đến khi số lần lặp CX=0. Điều này có nghĩa là trước khi vào vòng lặp ta phải đưa số lần lặp mong muốn vào CX, và sau mỗi lần lặp thì CX tự động giảm đi 1.

NHAN phải nằm cách xa (dịch đi một khoảng) tối đa -128 byte so với lệnh tiếp theo sau lệnh LOOP.

Các cờ bị thay đổi: không.

Ví dụ:

```
MOV   AL, 0 ;xoa AL
MOV   CX, 10      ;nap so lan lap vao CX
LAP:  INC   AL          ;tang AL len 1
LOOP  LAP         ;lap lai 10 lan, AL=10
```

33. LOOPE/LOOPZ – Loop while CX=0 or ZF=0 (lặp lại đoạn chương trình cho đến khi CX=0 hoặc ZF=0).

Dạng lệnh:

LOOPE NHAN

LOOPZ NHAN

Mô tả: Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ NHAN đến hết lệnh LOOPE NHAN hoặc LOOPZ NHAN) cho đến khi số lần lặp CX=0 hoặc cờ ZF=0. Điều này có nghĩa là trước khi vào vòng lặp ta phải đưa số lần lặp mong muốn vào CX, và sau mỗi lần lặp thì CX tự động giảm đi 1.

NHAN phải nằm cách xa (dịch đi một khoảng) tối đa -128 byte so với lệnh tiếp theo sau lệnh LOOPE/LOOPZ.

Các cờ bị thay đổi: không.

Ví dụ:

```
MOV   AL, AH      ;AL=AH
MOV   CX, 50      ;nap so lan lap vao CX
LAP:  INC   AL          ;tang AL
COMP  AL, 16      ;so sanh AL voi 16
LOOPE LAP   ;lap lai cho den khi AL≠16 hoac CX=0
```

34. LOOPNE/LOOPNZ – Loop while CX=0 or ZF=1 (lặp lại đoạn chương trình cho đến khi CX=0 hoặc ZF=1).

Dạng lệnh:

LOOPNE NHAN

LOOPNZ NHAN

Mô tả: Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ NHAN đến hết lệnh LOOPNE NHAN hoặc LOOPNZ NHAN) cho đến khi số lần lặp CX=0 hoặc cờ ZF=1. Điều này có nghĩa là trước khi vào vòng lặp ta phải đưa số lần lặp mong muốn vào CX, và sau mỗi lần lặp thì CX tự động giảm đi 1.

NHAN phải nằm cách xa (dịch đi một khoảng) tối đa -128 byte so với lệnh tiếp theo sau lệnh LOOPNE/LOOPNZ.

Các cờ bị thay đổi: không. *Ví dụ:*

```
MOV   AL, AH      ;AL=AH
MOV   CX, 50      ;nap so lan lap vao CX
LAP:  INC   AL          ;tang AL
COMP  AL, 16      ;so sanh AL voi 16
LOOPNE      LAP   ;lap lai cho den khi AL=16 hoac CX=0
```

2.3.2.8 Nhóm các lệnh điều khiển, đặc biệt khác

35. CALL – Call a procedure (gọi chương trình con)

Dạng lệnh: CALL Thủ_tục

Mô tả: Lệnh này dùng để chuyển hoạt động của vi xử lý từ chương trình chính (CTC) sang chương trình con (ctc). Nếu ctc nằm trong cùng một đoạn mã với CTC ta có gọi gần (near call). Nếu ctc và CTC nằm ở hai đoạn mã khác nhau ta có gọi xa (far call).

- Nếu gọi gần: Lưu vào Stack giá trị IP của địa chỉ trở về (vì CS không đổi) và các thao tác khi gọi ctc diễn ra như sau:
  + Nội dung thanh ghi SP giảm đi 2 byte, SP←SP – 2.
  + Nội dung thanh ghi IP được cất vào ngăn xếp (lưu địa chỉ trở về) {SP}←IP.
  + Địa chỉ lệch của ctc (lên tới ±32K) được lưu vào thanh ghi IP.
  + Khi gặp lệnh **RET** ở cuối ctc thì VXL lấy lại địa chỉ trở về IP từ Stack và tăng SP lên 2 byte.
- Nếu gọi xa: Lưu vào Stack giá trị IP và CS của địa chỉ trở về và các thao tác khi gọi ctc diễn ra như sau:
  + Nội dung thanh ghi SP giảm đi 2 byte, SP←SP – 2 và CS được lưu vào ngăn xếp.

+ Nội dung của CS được thay bằng địa chỉ đoạn của ctc được gọi.
+ Nội dung thanh ghi SP lại giảm đi 2 byte và IP được cất vào ngăn xếp.
+ Địa chỉ lệch của ctc được lưu vào thanh ghi IP.
+ Khi gặp lệnh **RET** ở cuối ctc thì VXL lấy lại địa chỉ trở về IP từ Stack và tăng SP lên 2 byte sau đó tiếp tục lấy lại CS và tăng SP lên 2 byte.

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF.

Ví dụ:

```
CALL  NEAR
CALL  FAR
```

36. INT – Interrupt (lệnh gọi ngắt)

Dạng lệnh: INT N (N=0…FFH)

Các thao tác của 8086 khi chạy lệnh: INT N

- Tạo địa chỉ mới của Stack, cất thanh ghi cờ vào Stack: SP←SP-2, {FR}→SP.
- Cấm các ngắt khác tác động vào vi xử lý, cho vi xử lý chạy ở chế độ từng lệnh: IF←0, TF←0.
- Tạo địa chỉ mới của Stack, cất địa chỉ đoạn của địa chỉ trở về vào Stack: SP←SP-2, SP←CS.
- Tạo địa chỉ mới của Stack, cất địa chỉ lệch của địa chỉ trở về vào Stack: SP←SP-2, SP←IP.
- Vi xử lý lấy lệnh tại địa chỉ mới, địa chỉ con trỏ ngắt được tính toán như sau:

{Nx4}→IP, {Nx4+2}→CS

Ví dụ: với N = 8 thì CS←{0022H} và IP←{0020H}

37. IRET – Interrupt Return (trở về CTC từ ctc phục vụ ngắt)

Dạng lệnh: IRET

Trở về chương trình chính từ chương trình con phục vụ ngắt. Trả lại quyền điều khiển cho chương trình tại vị trí xảy ra ngắt bằng cách lấy lại các giá trị thanh ghi IP, CS và các cờ từ vùng Stack.

- {SP}→IP, SP←SP+2
- {SP}→CS, SP←SP+2
- {SP}→FR, SP←SP+2

Các cờ bị thay đổi: tất cả các cờ (được phục hồi như trước khi diễn ra ngắt).

38. NOP – No Operation (CPU không làm gì)

Dạng lệnh: NOP

Lệnh này không thực hiện một công việc gì ngoài việc làm tăng nội dung của IP và tiêu tốn 3 chu kỳ đồng hồ. Nó thường được dùng để tính thời gian trễ trong các vòng trễ hoặc để chiếm chỗ các lệnh cần thêm vào chương trình sau này mà không làm ảnh hưởng dến độ dài chương trình.

Các cờ bị thay đổi: không.

39. **RET** – Return from Procedure to Calling Program (trở về chương trình chính từ chương trình con).

Dạng lệnh: **RET** hoặc **RET** N (N là số nguyên dương)

Mô tả: **RET** được đặt cuối ctc để vi xử lý lấy lại địa chỉ trở về, mà nó đã được tự động cất tại ngăn xếp khi có lệnh gọi ctc. Đặc biệt nếu dùng lệnh **RET** n thì sau khi đã lấy lại được địa chỉ trở về (chỉ có IP hoặc cả IP và CS) thì SP←SP+n (dùng để nhảy qua mà không lấy lại các thông số khác của chương trình còn lại trong ngăn xếp.

Các cờ bị thay đổi: không.

40. **STC** – Set the Carry Flag (lập cờ nhớ)

Dạng lệnh: STC

Mô tả: CF←1

STC thiết lập cờ nhớ bằng 1 và không ảnh hưởng đến các cờ khác.

Các cờ bị thay đổi: CF=1.

2.4 Lập trình hợp ngữ (Assembly) cho vi xử lý 80x86

Tham khảo "[12]"

2.4.1 Giới thiệu chung về hợp ngữ

Hợp ngữ (assembly language) là một ngôn ngữ cấp thấp dùng để viết các chương trình máy tính. Cách dùng các thuật nhớ (mnemonics) thân thiện để viết chương trình đã thay thế cách lập trình trực tiếp lên máy tính bằng mã máy dạng số (numeric machine code) - từng áp dụng cho những máy tính đầu tiên - vốn rất mệt nhọc, dễ gây lỗi và tốn nhiều thời giờ. Một chương trình viết bằng hợp ngữ sẽ được dịch sang ngôn ngữ máy bằng một tiện ích gọi là trình hợp dịch. Lưu ý rằng, trình hợp dịch khác hoàn toàn với trình biên dịch, vốn dùng để biên dịch các ngôn ngữ cấp cao sang các chỉ thị lệnh cấp thấp mà sau đó sẽ được trình hợp dịch chuyển đổi sang ngôn ngữ máy. Các chương trình hợp ngữ thường phụ thuộc chặt chẽ vào một kiến trúc máy tính xác định, nó khác với ngôn ngữ cấp cao thường độc lập đối với các nền tảng kiến trúc phần cứng. Nhiều trình hợp dịch phức tạp ngoài các tính năng cơ bản còn cung cấp thêm các cơ chế giúp cho việc viết chương trình, kiểm soát quá trình dịch cũng như việc gỡ rối được dễ dàng hơn. Hợp ngữ đã từng được dùng rộng rãi trong tất cả các khía cạnh lập trình, nhưng ngày nay nó có xu hướng chỉ được

dùng trong một số lãnh vực hẹp, chủ yếu để giao tiếp trực tiếp với phần cứng hoặc xử lý các vấn đề liên quan đến tốc độ cao điển hình như các trình điều khiển thiết bị, các hệ thống nhúng cấp thấp và các ứng dụng thời gian thực..

2.4.2 Các bước khi lập trình

Lập trình trên phần mềm emu8086

- Bước 1: Mở chương trình emu8086, chọn file \ new … Với các lựa chọn: New com template, new exe template, new bin template, new boot template.
- Bước 2: Viết mã nguồn
- Bước 3: dịch và gỡ rối (bấm F5)
- Bước 4: tạo file tự chạy: assembler \ Compile

Dịch, liên kết, chạy và chẩn lỗi chương trình từ dấu nhắc DOS:

Cần có các file: tasm.exe (dịch), tlink.exe (liên kết), td.exe (chẩn lỗi). Các bước như sau:

- B1. Thiết lập đường dẫn

  path = %path%;<đường dẫn đến thư mục chứa các file kể trên>

- B2. Biên dịch từ file .ASM sang file .OBJ

  Tasm <tên file chương trình>.ASM

- B3. Biên dịch từ file .OBJ sang file .EXE

  Tlink <tên file>.OBJ

- B4: chạy chương trình:

  <tên file>.EXE

- B5: chẩn lỗi (nếu cần thiết)

  Td <tên file>.EXE

  Để tự động hóa, ta có thể tạo file .BAT chứa các lệnh trên.

  Ví dụ:

  Tạo file RunASM.bat trong cùng thư mục với tập tin .ASM với nội dung như sau :

  tasm %1
  tlink %1
  %1

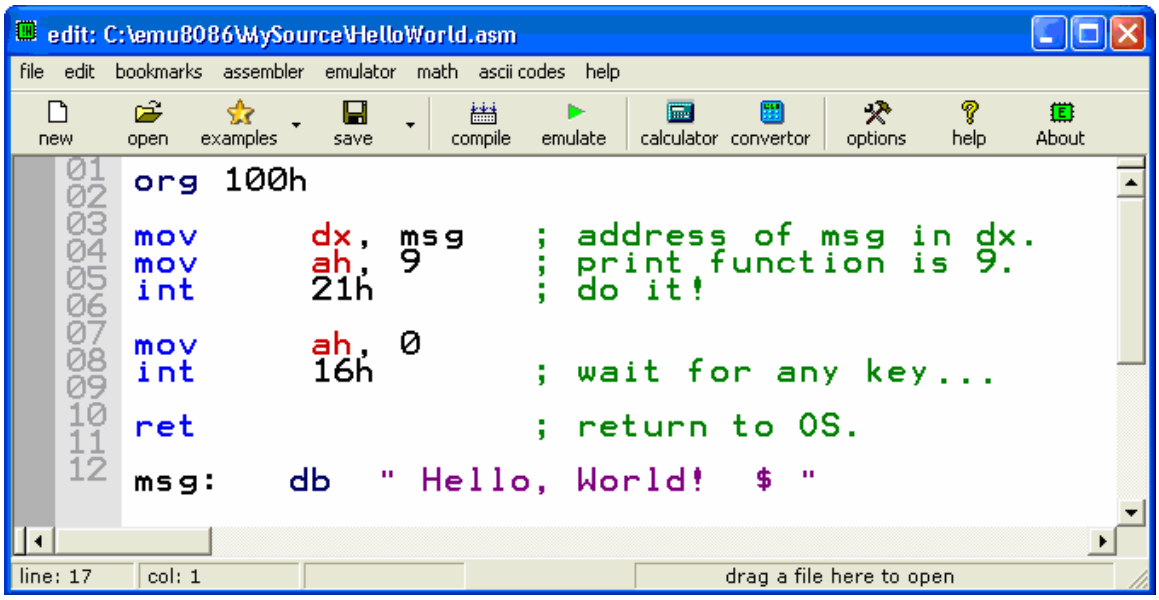
  (%1 là lấy tham số thứ nhất trong command line)

  Sau đó để biên dịch, liên kết và thực thi chương trình hello.ASM ta chỉ cần gõ :

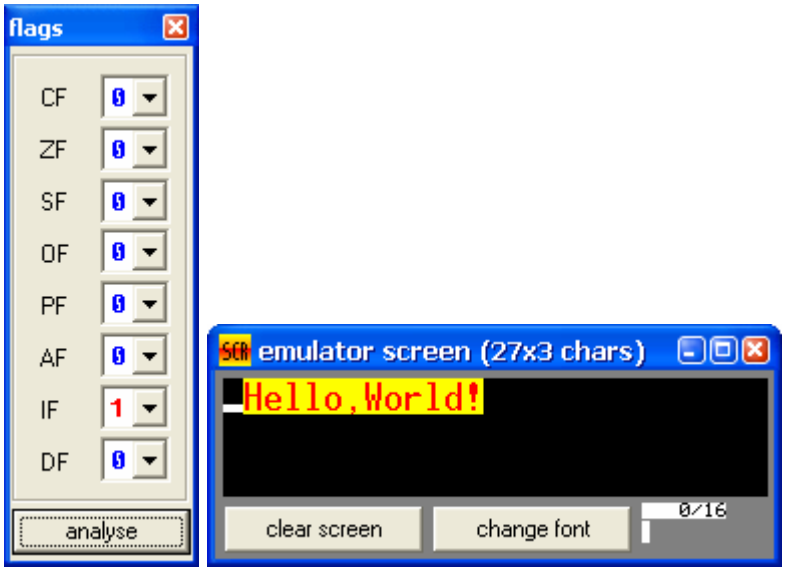
  RunASM hello

Chương trình emu8086:

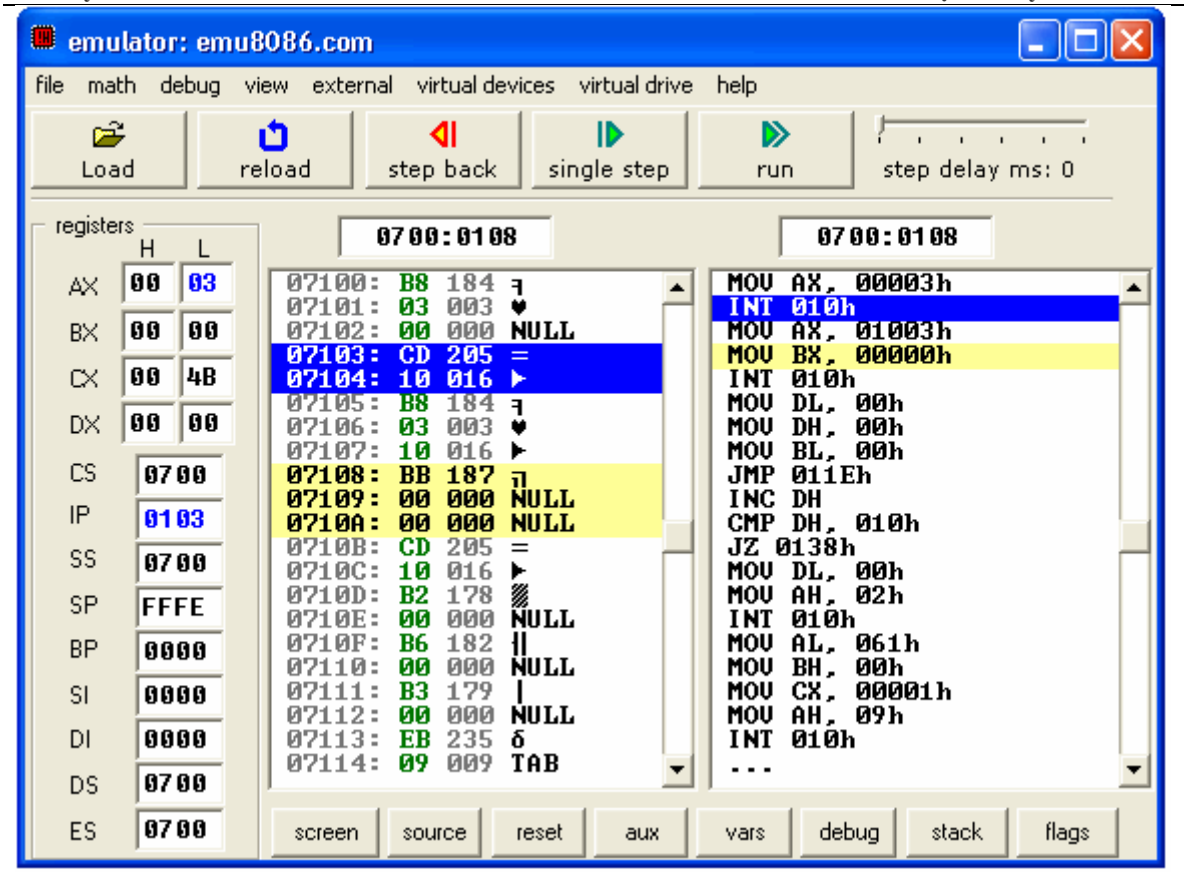
Chương trình emu8086 là chương trình lập trình mô phỏng cho 8086 (tương thích Intel và AMD) bao gồm bộ dịch ASM và giáo trình (tiếng anh) cho người mới bắt đầu. Chương trình có thể chạy hết hoặc chạy từng bước, ta có thể nhìn thấy các thanh ghi, bộ nhớ, stack, biến,…



Hình 2-5. Emu8086 - Môi trường soạn thảo



Hình 2-6. Emu8086 - Giá trị các cờ và màn hình hiển thị



Hình 2-7. Emu8086 - Màn hình Debug chương trình

2.4.3 Cấu trúc chung của chương trình hợp ngữ

2.4.3.1 Cấu trúc của một lệnh hợp ngữ

Tham khảo "[9]"

Một dòng lệnh trong chương trình hợp ngữ gồm có các trường sau:

Tên	Lệnh	Toán hạng	Chú thích
A:	Mov	AH, 10h	; Đưa giá trị 10h vào thanh ghi AH

Trường "tên" chứa nhãn, tên biến hay tên thủ tục. Các tên nhãn có thể chứa tối đa 31 ký tự, không chứa ký tự trắng (space) và không được bắt đầu bằng số. Các nhãn được kết thúc bằng dấu ':'.

Trường "lệnh" chứa các lệnh sẽ thực hiện. Các lệnh này có thể là các lệnh thật (MOV) hay các lệnh giả (PROC). Các lệnh thật sẽ được dịch ra mã máy.

Trường "toán hạng" chứa các toán hạng cần thiết cho lệnh (AH,10h).

Trường "chú thích" phải được bắt đầu bằng dấu ';'. Trường này chỉ dùng cho người lập trình để ghi các lời giải thích cho chương trình. Chương trình dịch sẽ bỏ qua các tất cả những gì nằm phía sau dấu ;

. Cấu trúc thông thường của một chương trình hợp ngữ dạng file *.exe

```
TITLE      Chương trình hợp ngữ
 .MODEL    Kiểu kích thước bộ nhớ ; Khai báo quy mô sử dụng bộ nhớ
 .STACK    Kích thước         ; Khai báo dung lượng đoạn stack
.DATA                         ; Khai báo đoạn dữ liệu
msg DB 'Hello$'
.CODE                         ; Khai báo đoạn mã main PROC
…
CALL  Subname                  ; Gọi chương trình con
…
main ENDP
Subname PROC                    ; Định nghĩa chương trình con
…
RET
Subname ENDP
END main
```

- **Quy mô sử dụng bộ nhớ:**

Loại	Mô tả
Tiny	Mã lệnh và dữ liệu nằm trong một đoạn
Small	Mã lệnh trong một đoạn, dữ liệu trong một đoạn
Medium	Mã lệnh không nằm trong một đoạn, dữ liệu trong một đoạn
Compact	Mã lệnh trong một đoạn, dữ liệu không nằm trong một đoạn
Large	Mã lệnh không nằm trong một đoạn, dữ liệu không nằm trong một đoạn và không có mảng nào lớn hơn 64KB
Huge	Mã lệnh không nằm trong một đoạn, dữ liệu không nằm trong một đoạn và các mảng có thể lớn hơn 64KB

Thông thường, các ứng dụng đơn giản chỉ đòi hỏi mã chương trình không quá 64 KB và dữ liệu cũng không lớn hơn 64 KB nên ta sử dụng ở dạng Small:

.MODEL SMALL

CPU 8086 có thể truy nhập tối đa 1MB bộ nhớ RAM. Dung lượng này là thừa để sử dụng cho bất kỳ loại máy tính nào.

Bản đồ bộ nhớ của máy tính IBM PC Địa chỉ vật lý của vùng nhớ (HEX)	**Giải thích vắn tắt**
00000 - 00400	Vector ngắt. Bộ mo phỏng sẽ load file này: **c:\emu8086\INT_VECT** tại địa chỉ vật lý 000000
00400 - 00500	Vùng thông tin hệ thống.
00500 - A0000	Một vùng nhớ tự do. Mỗi khối là 654,080 byte. Tại đây có thể load chương trình
A0000 - B1000	Vùng nhớ màn hình cho VGA, monochrome, và cho các bộ điều hợp khác
B1000 - B8000	Dự trữ

Bản đồ bộ nhớ của máy tính IBM PC Địa chỉ vật lý của vùng nhớ (HEX)	Giải thích vắn tắt
B8000 - C0000	32kb nhớ màn hình cho chế độ đồ họa màu (CGA). Bộ mô phỏng sử dụng vùng nhớ này để lưu 8 trang vùng nhớ màn hình. Màn hình mô phỏng có thể thay đổi kích thước, nên bộ nhớ tối thiểu được yêu cầu cho mỗi trang, mặc dù bộ mô phỏng luôn luôn sử dụng 1000h (4096 byte) cho mỗi trang (xem ngắt 10h, AH=05h)
C0000 - F4000	Dự trữ
F4000 - 10FFEF	ROM BIOS và mở rộng. Bộ mô phỏng tải file BIOS_ROM tại địa chỉ vật lý 0F4000h. Địa chỉ của bảng vector ngắt chỉ tới vùng nhớ này để tạo hàm ngắt mô phỏng.

Bảng vector ngắt (vùng nhớ từ 00000h đến 00400h)

Số hiệu ngắt (HEX)	Địa chỉ vector ngắt	Địa chỉ của chương trình con BIOS (address of BIOS sub-program)
00	00x4 = 00	F400:0170 – CPU tạo, lỗi chia
04	04x4 = 10	F400:0180 - CPU tạo, phát hiện INTO tràn
10	10x4 = 40	F400:0190 – Hàm video
11	11x4 = 44	F400:01D0 – Nhận danh sách thiết bị BIOS
12	12x4 = 48	F400:01A0 – Nhận kích thước bộ nhớ
13	13x4 = 4C	F400:01B0 - Các hàm về đĩa
15	15x4 = 54	F400:01E0 – Các hàm BIOS
16	16x4 = 58	F400:01C0 - Các hàm bàn phím
17	17x4 = 5C	F400:0400 – Máy in
19	19x4 = 64	FFFF:0000 – Khởi động lại
1A	1Ax4 = 68	F400:0160 – Hàm thời gian
1E	1Ex4 = 78	F400:AFC7 – vector tham số đĩa
20	20x4 = 80	F400:0150 – Hàm DOS: Kết thúc chương trình
21	21x4 = 84	F400:0200 – Các hàm của DOS
33	33x4 = CC	F400:0300 – Các hàm chuột
Các hàm khác	??x4 = ??	F400:0100 – Các ngắt mặc định

Vùng thông tin hệ thống (Bộ nhớ từ 00400h to 00500h)

Địa chỉ (HEX)	Kích thước	Giải thích
0040h:0010	WORD	Danh sách thiết bị BIOS Trường bit BIOS tìm thấy phần cứng được cài: bit(s) Giải thích 15-14 Số thiết bị song song 13 Dự trữ

Bản đồ bộ nhớ của máy tính IBM PC Địa chỉ vật lý của vùng nhớ (HEX)		Giải thích vắn tắt
		12 Cổng game được cài 11-9 Số thiết bị nối tiếp 8 Dự trữ 7-6 Số đĩa mềm (trừ 1): 00 Đĩa mềm đơn; 01 Hai dĩa mềm; 10 Ba đĩa mềm; 11 Bốn đĩa mềm; 5-4 Khởi tạo chế độ Video: 00 EGA,VGA,PGA, hoặc on-board video BIOS khác; 01 40x25 CGA màu. 10 80x25 CGA màu (Mô phỏng mặc định). 11 80x25 đen trắng. 3 Dữ trữ. 2 Chuột PS/2. 1 Bộ xử lý toán học; 0 Được cài khi khởi động từ đĩa mềm.
0040h:0013	WORD	kilobytes bắt đầu vùng nhớ liên tiếp tại địa chỉ 00000h từ này cũng được trả về AX bởi INT 12h giá trị này được đặt là 0280h (640KB)
0040h:004A	WORD	Số cột trên màn hình. Mặc định là 0032h (50 cột)
0040h:004E	WORD	Địa chỉ bắt đầu trang màn hình hiện hành trong bộ nhớ màn hình (sau 0B800:0000) Giá trị mặc định: 0000h
0040h:0050	8 WORD	Bao gồm vị trí hàng và cột cho con trỏ trong mỗi của tám trang nhớ màn hình. Giá trị mặc định: 00h (cho tất cả 8 từ (words)
0040h:0062	BYTE	Số trang màn hình hiện hành Mặc định: 00h (trang đầu tiên)
0040h:0084	BYTE	Hàng trên màn hình trừ 1 Giá trị mặc định: 13h (19+1=20 cột)

Bảng 2-4. Bản đồ bộ nhớ, địa chỉ ngắt của 8086

- **Khai báo kích thước stack:**

Khai báo stack dùng để dành ra một vùng nhớ dùng làm stack (chủ yếu phục vụ cho chương trình con), thông thường ta chọn khoảng 256 byte là đủ để sử dụng (nếu không khai báo thì chương trình dịch tự động cho kích thước stack là 1 KB):

.STACK 256

Khai báo đoạn dữ liệu:

Đoạn dữ liệu dùng để chứa các biến và hằng sử dụng trong chương trình.

- **Khai báo đoạn mã:**

Đoạn mã dùng chứa các mã lệnh của chương trình. Đoạn mã bắt đầu bằng một chương trình chính và có thể có các lệnh gọi chương trình con (CALL).

Một chương trình chính hay chương trình con bắt đầu bằng lệnh PROC và kết thúc bằng lệnh ENDP (đây là các lệnh giả của chương trình dịch). Trong chương trình con, ta sử dụng thêm lệnh RET để trả về địa chỉ lệnh trước khi gọi chương trình con.

Chương trình được kết thúc bằng lệnh END trong đó tên chương trình phía sau lệnh END sẽ xác định đó là chương trình chính. Nếu sau lệnh END không chỉ ra chương trình nào cả thì sẽ lấy chương trình con ở đàu đoạn mã làm chương trình chính.

Ví dụ: Chương trình sau in ra màn hình dòng chữ "Hello !"

```
.model      small
.stack      100h
.data
      s     DB    "Hello !$" ; khai báo xâu kí tự cần in
.code
            mov   AX,@data   ; lấy địa chỉ data segment ghi vào DS
            mov   DS,AX      ; Vì model small, đây cũng là địa chỉ
; segment của xâu s. ; xuất chuỗi:
            mov   DX, OFFSET s     ; lấy địa chỉ offset ghi vào DX
            mov   AH , 9
            int   21h              ; gọi hàm 9, ngắt 21h để in
            mov   AH, 4Ch          ; Thoát khỏi chương trình
            int   21h
end
```

Lưu ý:

- Mọi chương trình đều phải có đoạn **CODE** thoát khỏi chương trình, nếu không chương trình sẽ không dừng khi hết chương trình của mình.

2.4.3.2 Khung chương trình dịch ra .exe

Các tập tin .EXE và .COM

DOS chỉ có thể thi hành được các tập tin dạng .COM và .EXE. Tập tin .COM thường dùng để xây dựng cho các chương trình nhỏ còn .EXE dùng cho các chương trình lớn.

Tập tin .EXE

- Nằm trong nhiều đoạn khác nhau, kích thước thông thường lớn hơn 64 KB.
- Có thể gọi được các chương trình con dạng near hay far.
- Tập tin .EXE chứa một header ở đầu tập tin để chứa các thông tin điều khiển cho tập tin.

```
data segment
    ; add your data here!
    pkey db "press any key to exit ...$"
ends
stack segment
    dw   128  dup(0)
ends
CODE segment
start:
; set segment registers:
    MOV ax, data
    MOV ds, ax
    MOV es, ax

            ; add your CODE here

    lea dx, pkey
    MOV ah, 9
    int 21h        ; output string at ds:dx

    ; wait for any key....
    MOV ah, 1
    int 21h

    MOV ax, 4c00h ; exit to operating system.
    int 21h
ends
END start ; set entry point and stop the assembler.
```

2.4.3.3 Khung chương trình dịch ra .com

- Tập tin .COM chỉ có một đoạn nên kích thước tối đa của một tập tin loại này là 64 KB.
- Tập tin .COM được nạp vào bộ nhớ và thực thi nhanh hơn tập tin .EXE nhưng chỉ áp dụng được cho các chương trình nhỏ.
- Chỉ có thể gọi các chương trình con dạng near.

Khi thực hiện tập tin .COM, DOS định vị bộ nhớ và tạo vùng nhớ dài 256 byte ở vị trí 0000h, vùng này gọi là PSP (Program Segment Prefix), nó sẽ chứa các thông tin cần thiết cho DOS. Sau đó, các mã lệnh trong tập tin sẽ được nạp vào sau PSP ở vị trí 100h và đưa giá trị 0 vào stack. Như vậy, kích thước tối đa thực sự của tập tin .COM là 64 KB – 256 byte PSP – 2 byte stack.

Tất cả các thanh ghi đoạn đều chỉ đến PSP và thanh ghi con trỏ lệnh IP chỉ đến 100h, thanh ghi SP có giá trị 0FFFEh.

```
; You may customize this and other start-up templates;
; The location of this template is
;c:\emu8086\inc\0_com_template.txt
CSEG    SEGMENT     ; code segment starts here.
org 100h
; add your CODE here
ret
```

❖ Khai báo dữ liệu

Khi khai báo dữ liệu trong chương trình, nếu sử dụng số nhị phân, ta phải dùng thêm chữ B ở cuối, nếu sử dụng số thập lục phân thì phải dùng chữ H ở cuối. Chú ý rằng đối với số thập lục phân, nếu bắt đầu bằng chữ A..F thì phải thêm vào số 0 ở phía trước.

Ví dụ:

```
1011b  ; Số nhị phân
1011  ; Số thập phân
1011d  ; Số thập phân
1011h  ; Số thập lục phân
```

❖ Khai báo hằng, biến

Cú pháp:

<tên biến> D<Kiểu DL> <giá trị khởi tạo>

hoặc

<tên biến> D<Kiểu DL> <số phần tử> dup(<giá trị khởi tạo>)

Các kiểu dữ liệu: B (1 byte), W (2 bytes), D (4 bytes)

Nếu không khởi tạo, dùng dấu hỏi "**?**"

Ví dụ:

Khai báo trong C	Khai báo biến trong hợp ngữ
char ch;	ch DB ?
char ch = 'a';	ch DB 'a'
char ch = 5;	ch DB 5
Char s[]="\nhello world!"	s DB 10,13,"hello world!$"
int i=100;	i DW 100
long L;	L DD ?
char a[] = {1,2,3};	a DB 1,2,3
char a[100];	a DB 100 dup(?)
char a[100][50];	a DB 100 dup(50 dup(?))

Hằng số:

Khai báo hằng số trong chương trình hợp ngữ bằng lệnh EQU.

Ví dụ:

```
A1    EQU   02, 11
A2    EQU   19, 81
```

❖ Toán tử trong hợp ngữ

Toán tử số học

Toán tử	Cú pháp	Mô tả
+	+bt	Số dương
-	-bt	Số âm
*	bt1*bt2	Nhân
/	bt1/bt2	Chia
mod	bt1 mod bt2	Lấy phần dư
+	bt1 + bt2	Cộng
-	bt1 – bt2	Trừ
shl	bt shl n	Dịch trái n bit
shr	bt shr n	Dịch phải n bit

Trong đó bt, bt1, bt2 là các biểu thức hằng, n là số nguyên.

Toán tử logic: Bao gồm các toán tử AND, OR, NOT, XOR

Toán tử quan hệ: Các toán tử quan hệ so sánh 2 biểu thức, cho giá trị true (1) nếu điều kiện thoả và false (0) nếu không thoả.

Toán tử	Cú pháp	Mô tả
EQ	bt1 EQ bt2	Bằng
NE	bt1 NE bt2	Không bằng
LT	bt1 LT bt2	Nhỏ hơn
LE	bt1 LE bt2	Nhỏ hơn hay bằng
GT	bt1 GT bt2	Lớn hơn
GE	bt1 GE bt2	Lớn hơn hay bằng

Toán tử cung cấp thông tin:

- Toán tử SEG: SEG bt ; Toán tử SEG xác định địa chỉ đoạn của biểu thức bt. bt có thể là biến, nhãn, hay các toán hạng bộ nhớ.

- Toán tử OFFSET: OFFSET bt ;Toán tử OFFSET xác định địa chỉ offset của biểu thức bt. bt có thể là biến, nhãn, hay các toán hạng bộ nhớ.

VD: MOV AX,SEG A ; Nạp địa chỉ đoạn và địa chỉ offset
MOV DS,AX ; của biến A vào cặp thanh ghi
MOV AX,OFFSET A ; DS:AX

- Toán tử chỉ số [ ]: (index operator) Toán tử chỉ số thường dùng với toán hạng trực tiếp và gián tiếp.

- Toán tử (:) (segment override operator) Segment:bt ; Toán tử : quy định cách tính địa chỉ đối với segment được chỉ. Segment là các thanh ghi đoạn CS, DS, ES, SS.

Chú ý rằng khi sử dụng toán tử : kết hợp với toán tử [ ] thì segment: phải đặt ngoài toán tử [ ].

VD: Cách viết [CS:BX] là sai, ta phải viết CS:[BX]

- Toán tử TYPE:

TYPE bt ;Trả về giá trị biểu thị dạng của biểu thức bt.

Nếu bt là biến thì sẽ trả về 1 nếu biến có kiểu byte, 2 nếu biến có kiểu word, 4 nếu biến có kiểu double word. Nếu bt là nhãn thì trả về 0FFFFh nếu bt là near và 0FFFEh nếu bt là far. Nếu bt là hằng thì trả về 0.

- Toán tử LENGTH:

LENGTH bt ;Trả về số đơn vị bộ nhớ cấp cho biến bt

- Toán tử SIZE:

SIZE bt ;Trả về tổng số các byte cung cấp cho biến bt

```
VD:   A DD 100 DUP(?)
MOV AX,LENGTH A ; AX = 100
MOV AX,SIZE A  ; AX = 400
```

Các toán tử thuộc tính:

- Toán tử PTR:

Loai PTR bt ; Toán tử này cho phép thay đổi dạng của biểu thức bt.

Nếu bt là biến hay toán hạng bộ nhớ thì Loai là byte, word hay dword. Nếu bt là nhãn thì Loai là near hay far.

```
      VD:   A DW 100 DUP(?)
            B DD ?
```

```
MOV AH,BYTE PTR A  ; Đưa byte đầu tiên trong mảng A vào ;thanh ghi AH
MOV AX,WORD PTR B  ; Đưa 2 byte thấp trong biến B vào thanh ;ghi AX
```

- Toán tử HIGH, LOW:

HIGH bt

LOW bt

Cho giá trị của byte cao và thấp của biểu thức bt, bt phải là một hằng.

```
VD:    A EQU 1234h
 MOV AH,HIGH A    ; AH ← 12h
 MOV AH,LOW A    ; AH ← 34h
```

❖ Chương trình con

Chương trình con (PROC) là một phần của mã nguồn mà có thể gọi chúng trong chương trình của bạn để làm một vài nhiệm vụ nhất định nào đó. Chương trình con làm cho chương trình có cấu trúc hơn và dễ hiểu hơn. Thông thường, chương trình con trở lại ngay sau điểm đã gọi nó.

Cấu trúc một chương trình con như sau:

```
TÊN PROC

      ; đây là mã lệnh của chương trình con

RET
TÊN ENDP
```

TÊN là tên của chương trình con, tên phải giống nhau ở trên và dưới của chương trình con, đó là cách để kiểm tra điểm kết thúc của chương trình con.

Hầu như chắc chắn, bạn đã biết rằng lệnh RET được sử dụng để trở về hệ điều hành. Lệnh tương tự cũng được sử dụng để trở về từ chương trình con (thực sự, OS coi chương trình của chúng ta như một chương trình con đặc biệt)

PROC và ENDP là các định hướng chương trình dịch, nên chúng không được dịch ra mã máy. Chương trình dịch nhớ địa chỉ của chương trình con.

Lệnh CALL được sử dụng để gọi chương trình con

Đây là một ví dụ:

```
ORG 100h
     CALL   ta
     MOV    AX, 2
RET        ; Trở về OS
ta PROC
     MOV    BX, 5
     RET    ; Trở về sau điểm đã gọi.
ta     ENDP
END
```

Ví dụ trên gọi chương trình con **ta**, để thực hiện lệnh "MOV BX, 5" , và trở về sau lệnh gọi nó "MOV AX, 2"

Có vài cách để truyền tham số cho chương trình con, cách đơn giản nhất là sử dụng các thanh ghi, dưới đây là một ví dụ khác về cách gọi chương trình con và cách truyền tham số cho nó qua thanh ghi **AL** và **BL**, nhân hai tham số với nhau và trả kết quả về trong thanh ghi **AX**:

```
ORG    100h
MOV    AL, 1
MOV    BL, 2
     CALL   m2
     CALL   m2
     CALL   m2
     CALL   m2
RET              ; Trở về HĐH
m2     PROC
     MUL    BL             ; AX = AL * BL.
     RET                   ; Trở về sau điểm gọi nó.
m2     ENDP
END
```

Trong ví dụ trên, giá trị của thnh ghi **AL** được cập nhật mỗi lần chương trình con được gọi, thanh ghi **BL** không thay đổi, nên thuật toán trên là tính 2^4, kết quả lưu trong AX là **16** (hay 10h)

Dưới đây là một ví dụ khác, sử dụng chương trình con để in xâu *"PICAT.dieukhien.net"* :

```
ORG    100h
LEA    SI, tbao_tw  ; Lấy địa chỉ của msg vào SI.
CALL   In_Xau
RET                 ; trở về hệ điều hành.
;================================================
; Chương trình này in 1 xâu, xâu phải kết thúc
; bằng ký tự null (phải có 0 cuối xâu)
; địa chỉ của xâu phải được đặt trong thanh ghi SI:

In_Xau     PROC
next_char:
    CMP  b.[SI], 0    ; kiểm tra nếu = 0 thì dừng
    JE   stop         ;
    MOV  AL, [SI]     ; lấy ký tự tiếp theo.
    MOV  AH, 0Eh      ; số hiệu in ký tự.
    INT  10h          ; sử dụng ngắt để in ký tự trong AL.
    ADD  SI, 1        ; Tăng con trỏ cần in lên 1.
    JMP  next_char    ; trở lại, in ký tự tiếp.
stop:
RET                 ; trở về sau điểm gọi.
print_me     ENDP
; ================================================
tbao_tw DB 'PICAT.dieukhien.net',0; xâu kết thúc: null.
END
```

Tiếp đầu ngữ "**b.**" trước [SI] nghĩa là so sánh byte, không phải từ. Nếu bạn cần so sánh từ, bạn dùng tiếp đầu ngữ "**w.**" thay thế vào. Khi một toán hạng đã nằm trong thanh ghi, nó không yêu cầu nữa.

❖ Lệnh bó (Macro)

Macro tương tự như chương trình con nhưng không thực sự là chương trình con. Macro nhìn có vẻ như chương trình con, nhưng chúng chỉ tồn tại cho đến khi chương trình được dịch, sau khi chương trình được dịch tất cả các macro được thay thế bằng lệnh thực sự. Nếu bạn khai báo một macro và không bao giờ sử dụng chúng trong mã nguồn, chương trình dịch sẽ bỏ qua nó.

Khai báo:

```
name    MACRO  [tham số,...]

            <Lệnh>
ENDM
```

Không như chương trình con, macro phải khai báo bên trên đoạn mã nguồn gọi nó, ví dụ:

```
MyMacro    MACRO  p1, p2, p3
     MOV AX, p1
     MOV BX, p2
     MOV CX, p3
ENDM
```

```
ORG 100h
     MyMacro 1, 2, 3
     MyMacro 4, 5, DX
RET
```

Đoạn mã nguồn trên sẽ được mở rộng thành:

```
MOV   AX,   0001h
MOV   BX,   0002h
MOV   CX,   0003h
MOV   AX,   0004h
MOV   BX,   0005h
MOV   CX,   DX
```

Vài điều thực sự quan trọng về Macro và chương trình con:

- Khi muốn sử dụng một chương trình con, bạn phải sử dụng từ khóa CALL, ví dụ:

```
Call  TA_Proc
```

- Khi bạn sử dụng một Macro, bạn chỉ cần gõ tên của chúng, ví dụ:

```
Ta_Macr
```

- Chương trình con được định vị tại một địa chỉ cụ thể trong bộ nhớ, và nếu bạn sử dụng 100 lần chương trình con đó, CPU chỉ chuyển điều khiển đến vùng nhớ của chương trình con đó thôi. Điều khiển sẽ trở lại chương trình khi gặp lệnh RET. Stack được sử dụng để giữ địa chỉ trở về. Lệnh CALL chỉ tốn hết 3 byte, nên kích thước của chương trình thực thi nhỏ, không quan trọng việc gọi chương trình con bao nhiêu lần.
- Macro mở rộng trực tiếp các lệnh của nó vào mã nguồn, nếu macro mở rộng 100 lần (gọi 100 lần) sẽ làm cho chương trình thực thi lớn hơn rất nhiều, càng lớn khi macro được gọi càng nhiều.
- Bạn phải sử dụng Stack hoặc bất kỳ thanh ghi nào để truyền tham số cho chương trình con
- Để truyền tham số cho maco, bạn chỉ cần gõ chúng sau tên của macro khi gọi, ví dụ:

```
TA_mac 1, 2, 3
```

- Để đánh dấu kết thúc macro, chỉ cần từ khóa ENDM là đủ

- Để đánh dấu kết thúc chương trình con bạn cần phải đánh tên của chương trình con trước từ khóa ENDP

Macro được mở rộng trực tiếp trong mã nguồn của bạn, vì thế nếu bạn có nhiều nhãn giống nhau trong khai báo macro bạn có thể nhận thông báo lỗi "Khai báo trùng lặp" khi macro được sử dụng 2 lần hoặc nhiều hơn. Để loại bỏ lỗi này, bạn dùng từ khóa LOCAL để khai báo rằng nhãn sau nó là nhãn cục bộ, nhãn cục bộ có thể là biến, nhãn, hoặc chương trình con.

Ví dụ:

```
MyMacro2    MACRO
     LOCAL label1, label2
     CMP  AX, 2
     JE label1
     CMP  AX, 3
     JE label2
     label1: INC  AX
     label2: ADD  AX, 2
ENDM
ORG 100h
MyMacro2
MyMacro2
RET
```

Nếu bạn có kế hoạch sử dụng macro nhiều lần, một ý hay là nên đặt tất cả các macro trong một file. Và đặt file đó trong thư mục INC và sử dụng chỉ thị INCLUDE <Tên-file> để có thể sử dụng macro đó.

2.4.4 Các cấu trúc điều khiển cơ bản

2.3.4.1 Cấu trúc tuần tự

Cấu trúc tuần tự là cấu trúc đơn giản nhất. Trong cấu trúc tuần tự, các lệnh được sắp xếp tuần tự, lệnh này tiếp theo lệnh kia, mỗi lệnh một dòng.

```
Lệnh 1
Lệnh 2
…
Lệnh n
```

VD: Cộng 2 giá trị của thanh ghi BX và CX, rồi nhân đôi kết quả, kết quả cuối cùng chứa trong AX

```
MOV AX,BX
ADD AX,CX ; Cộng BX với CX
SHL AX,1  ; Nhân đôi
```

2.3.4.2 Cấu trúc IF – THEN, IF – THEN – ELSE

IF <Điều kiện> THEN <Công việc>

IF <Điều kiện> THEN <Công việc1> ELSE <Công việc2>

VD: Gán BX = |AX|

```
CMP AX,0  ; AX > 0?
JNL DUONG ; AX dương
NEG AX  ; Nếu AX < 0 thì đảo dấu
DUONG: MOV BX,AX
NEXT:
```

VD: Gán CL giá trị bit dấu của AX

```
CMP AX,0  ; AX > 0?
JNS AM  ; AX âm
MOV CL,1  ; CL = 1 (AX dương)
JMP NEXT
AM: MOV CL,0  ; CL = 0 (AX âm)
NEXT:
```

2.3.4.3 Cấu trúc CASE

```
CASE <Biểu thức>
     Giá trị 1: Công việc 1
     Giá trị 2: Công việc 2
     …
     Giá trị n: Công việc n
END
```

VD: Nếu AX > 0 thì BH = 0, nếu AX < 0 thì BH = 1. Ngược lại BH = 2

```
CMP AX,0
JL AM
JE KHONG
G DUONG
DUONG: MOV BH,0
     JMP NEXT
AM:  MOV BH,1
     JMP NEXT
KHONG: MOV BH,2
     NEXT:
```

2.3.4.4 Cấu trúc FOR

FOR <Số lần lặp> DO <Công việc>

VD: Cho vùng nhớ M dài 200 bytes trong đoạn dữ liệu, chương trình đếm số chữ A trong vùng nhớ M như sau:

```
           MOV CX,200  ; Đếm 200 bytes
           MOV BX,OFFSET M ; Lấy địa chỉ vùng nhớ
           XOR AX,AX  ; AX = 0
NEXT: CMP BYTE PTR [BX],'A'; So sánh với chữ A
            JNZ ChuA   ; Nếu không phải là chữ A thì tiếp
           INC AX   ; tục, ngược lại thì tăng AX
ChuA:  INC BX
           LOOP  NEXT
```

2.3.4.5 Cấu trúc lặp WHILE

WHILE <Điều kiện> DO <Công việc>

(TRONG KHI <Điều kiện = True> LÀM <Công việc>)

VD: Chương trình đọc vùng nhớ bắt đầu tại địa chỉ 1000h vào thanh ghi AH, đến khi gặp ký tự '$' thì thoát:

```
 MOV BX,1000h
CONT: CMP AH,'$'
      JZ NEXT
MOV AH,DS:[BX]
 JMP CONT
NEXT:
```

2.3.4.6 Cấu trúc lặp REPEAT

REPEAT <Công việc> UNTIL <Điều kiện>

(LẶP <Công việc> ĐẾN KHI <Điều kiện=True thì dừng>)

VD: Chương trình đọc vùng nhớ bắt đầu tại địa chỉ 1000h vào thanh ghi AH, đến khi gặp ký tự '$' thì thoát:

```
MOV BX,1000h
CONT: MOV AH,DS:[BX]
      CMP AH,'$'
      JZ NEXT
      JMP CONT
NEXT:
```

Ví dụ:

```
org 100h
mov bx, 0  ; total step counter.
mov cx, 5
k1: add bx, 1
    mov al, '1'
    mov ah, 0eh
    int 10h
    push cx
    mov cx, 5
      k2: add bx, 1
      mov al, '2'
      mov ah, 0eh
      int 10h
      push cx
         mov cx, 5
         k3: add bx, 1
         mov al, '3'
         mov ah, 0eh
         int 10h
         loop k3    ; internal in internal loop.
      pop  cx
      loop  k2      ; internal loop.
    pop cx
loop k1             ; external loop.
Ret
```

2.4.5 Ngắt trong Assembly

Ngắt có thể hiểu là số của các hàm. Các hàm hày làm cho việc lập trình đơn giản hơn, thay vì viết mã nguồn để in ra ký tự bạn có thể đơn giản là gọi ngắt và nó sẽ tự làm mọi việc cho bạn. Cũng có các hàm ngắt (chương trình con ngắt) làm việc với ổ đĩa và các phần cứng khác. Chúng ta gọi là ngắt mềm.

Ngắt cũng có thể được gọi từ các phần cứng. ở đây chúng ta chỉ đề cập đến ngắt mềm.

Để tạo ngắt mềm, có cách là chúng ta gọi lệnh INT, cấu trúc rất đơn giản:

```
INT <giá trị>
```

Trong đó <giá trị> có thể là các số từ 0 đến 255 (hoặc 0..0FFh). Chúng ta thường dùng số hệ 16.

Bạn có thể hiểu là chỉ có 256 hàm ngắt, điều đó là không đúng. Mỗi hàm ngắt có thể có số hiệu ngắt. Với mỗi số hiệu ngắt, ta lại có một chương trình con ngắt riêng.

Để chỉ ra số hiệu ngắt, thanh ghi AH phải được thiết lập trước khi gọi ngắt.

Mỗi ngắt có thể có tối đa 256 số hiệu ngắt (nên chúng ta có 256*256=65536)

Ngoài ra, chúng ta có thể dùng các thanh ghi khác để truyền tham số cho ngắt.

Ví dụ sau in ra một ký tự ra màn hình:

```
mov ah, 2
mov dl, 'a'
int 21h
```

Trong đó, chúng ta dùng hàm ngắt thứ 21h (INT 21h), số hiệu ngắt là 2 (ah=2), và tham số cần in đượ truyền vào thanh ghi dl (dl='a').

Dưới đây là một số số hiệu ngắt thông dụng:

Ngắt 21h:	
AH	**Ý nghĩa**
1	Đọc 1 ký tự từ bàn phím, KQ lưu trong AL, nếu chưa bấm, chờ bằng được

2	In 1 ký tự ra màn hình, DL=Ký tự cần in, sau khi in: AL=DL
	mov ah, 2
	mov dl, 'a'
	int 21h

6	- Nhập, Nếu DL=255: ZF=1,AL=0 nếu không có phím nào được bấm, trái lại: ZF=0, AL=Ký tự đã bấm, xóa bộ đệm bàn phím. - In ký tự, nếu DL=0..254: DL=ký tự cần in, in xong: AL=DL,
9	In một xâu ký tự, được trỏ bởi DX, xâu ký tự phải kết thúc bằng '$' org 100h mov dx, offset msg mov ah, 9 int 21h

Ngắt 21h:	
AH	**Ý nghĩa**
	ret msg db "hello world $"

10	Nhập một xâu ký tự vào: DS:DX, Byte đầu tiên là kích thước bộ đệm, Byte thứ 2 là ký tự thực tế đã nhập. Hàm này không thêm '$' vào cuối xâu.
	Để in được xâu, cần thêm ký tự '$' vào cuối, và bắt đầu in từ địa chỉ DS:DX+2
	Ví dụ:
	org 100h
	mov dx, offset buffer
	mov ah, 0ah
	int 21h
	jmp print
	buffer db 10,?, 10 dup(' ')
	print:
	xor bx, bx
	mov bl, buffer[1]
	mov buffer[bx+2], '$'
	mov dx, offset buffer + 2
	mov ah, 9
	int 21h
	ret

2.4.6 Các ví dụ

Ví dụ 1. Hello word đơn giản (COM file)

```
; Viết ra màn hình dòng chữ "hello, world!"
; Su dung .com
name "hi"
org 100h
JMP start       ; jump over string declaration
     msg     db      "hello, world!", 0Dh,0Ah, 24h
start:  lea     dx, msg  ; load effective address of
                         ;msg into dx.
        MOV     ah, 09h  ; print function is 9.
        int     21h      ; do it!

        MOV     ah, 0
        int     16h      ; wait for any key any....
RET ; return to operating system.
```

Ví dụ 2. Hello Word (EXE file)

```
; a tiny example of multi segment executable file.
; data is stored in a separate segment, segment registers must be
set correctly.
name "testexe"
data segment
    msg  db "hello, world!", 0dh,0ah, '$'
ends
stack segment
    db 30 dup(0)
ends
CODE segment
start:
 ; set segment registers:
        MOV     ax, data
        MOV     ds, ax
        MOV     es, ax
 ; print "hello, world!":
        lea     dx, msg
        MOV     ah, 09h
        int     21h
 ; wait for any key...
        MOV     ah, 0
        int     16h
 ; return control to os:
        MOV     ah, 4ch
        int     21h
ends
        END start  ; set entry point and stop the assembler.
```

Ví dụ 3. Tính: Tổng, hiệu, tích, thương:

```
org 100h
    mov cl,8
    mov dl,3
    Call Tong
    Call hieu
    Call tich
    Call thuong
ret
Tong proc
    mov al,cl
    add al,dl
    ret
Hieu proc
    mov al,cl
    sub al,dl
    ret
Tich proc
    mov al,cl
    mul dl
    ret
Thuong proc
    mov al,cl
    div dl
    ret
end
```

Ví dụ 4. In một số nhị phân ra màn hình:

```
name "add-sub"
org 100h
MOV al, 5       ; bin=00000101b
MOV bl, 10      ; hex=0ah or bin=00001010b
; 5 + 10 = 15 (decimal) or hex=0fh or bin=00001111b
add bl, al
; 15 - 1 = 14 (decimal) or hex=0eh or bin=00001110b
sub bl, 1
; print result in binary:
MOV cx, 8
print: MOV ah, 2   ; print function.
       MOV dl, '0'
       test bl, 10000000b  ; test first bit.
       jz zero
       MOV dl, '1'
zero:  int 21h
       shl bl, 1
```

```
loop print
; print binary suffix:
MOV dl, 'b'
int 21h
; wait for any key press:
MOV ah, 0
int 16h
ret
```

Ví dụ 5. In một số hệ 10 ra màn hình:

```
name "Print Decimal function, tuananhvxl@gmail.com"
Enter Macro
    mov ah,2
        mov dl, 0ah ; new line.
        int 21h
      mov dl, 0dh ; carrige return.
      int 21h
endm
org 100h ; directive make tiny com file.
         ; print result in decimal:
mov al, 123
call Print_dec8AL
     Enter
mov al, 45
call Print_dec8AL
         ; wait for any key press:
    mov ah, 0
    int 16h
ret

Print_dec8AL proc
cmp al, 0
jne Print_dec8AL_r
    push ax
    mov dl, '0'
    mov ah, 2
    int 21h
    pop ax
    ret
Print_dec8AL_r:
    pusha
    mov ah, 0
    cmp ax, 0
    je pn_done
        mov dl, 10
        div dl
        call Print_dec8AL_r
        mov dl, ah
```

```
        add dl, 30h
        mov ah, 2h
        int 21h
    jmp pn_done
pn_done:
    popa
    ret
endp
```

Ví dụ 6. In xâu ra màn hình:

```
name "Print_String"

Print_String macro str
    mov dx, offset str
    mov ah, 9
    int 21h
endm

org 100h
     Print_String Thongbao1
     Print_String Thongbao2
ret

Thongbao1 db "Xin chao", 0Dh,0Ah, "$"
Thongbao2 db "Cac ban", 0Dh,0Ah, "$"
```

Ví dụ 7. Nhập một số hệ 10, nhân với 2 rồi in ra màn hình

```
name "Input Number [tuananhktmt@gmail.com]"
Enter Macro
    pusha
        mov ah,2
        mov dl, 0ah ; new line.
        int 21h
        mov dl, 0dh ; carrige return.
        int 21h
    popa
endm

org     100h

; Nhap - start:
mov dx, offset msg
mov ah, 9
int 21h

;-----------------------------------
xor cl,cl
```

```
wait_for_key: ; Cho` bam phim:
    mov ah, 1
    int 21h
  cmp al,0Dh;| Bam ENTER thi ket thuc:
  jz  exit  ;|
; Tinh CL=CL*10+AL
  sub al,'0'
   push ax
   xor ah,ah
     mov al,cl ;|
     mov dl,10 ;| CL=CL*10
     mul dl    ;|
     mov cl,al ;|
   pop ax
  add cl,al  ; CL=CL+AL (0..9)
jmp wait_for_key
exit:

    mov AX,0
    mov DL,2
    mov AL, CL
    Mul DL
    Enter
    call Print_dec8AL

    mov     ah, 0
    int     16h
ret

;-----------------------
Print_dec8AL proc
cmp al, 0
jne Print_dec8AL_r
    push ax
    mov al, '0'
    mov ah, 0eh
    int 10h
    pop ax
    ret
Print_dec8AL_r:
    pusha
    mov ah, 0
    cmp ax, 0
    je pn_done
    mov dl, 10
    div dl
    call Print_dec8AL_r
    mov al, ah
```

```
    add al, 30h
    mov ah, 0eh
    int 10h
    jmp pn_done
pn_done:
    popa
    ret
endp
;-----------------------------------------------
msg  db "Moi ngai nhap vao 1 so 8 bit:", 0Dh,0Ah
     db "N=$"
end
```

Ví dụ 8. Cộng 2 mảng dài 4 byte

```
name "add-2 array"
org 100h
jmp start
    vec1 db 1, 2, 5, 6
    vec2 db 3, 5, 6, 1
    vec3 db ?, ?, ?, ?
start:
    lea si, vec1
    lea bx, vec2
    lea di, vec3
 mov cx, 4
 sum:
    mov al, [si]
    add al, [bx]
    mov [di], al
       inc si
       inc bx
       inc di
    loop sum
ret
```

Ví dụ 9. **Nhập một chuỗi ký tự và chuyển chữ thường thành chữ hoa**

```
.MODEL SMALL
.STACK 100h
.DATA
 m1   DB 81
           DB ?
           DB 81 DUP(?)
      m2   DB 'Chuoi da doi:$'
.CODE
main PROC
 MOV AX,@DATA
 MOV DS,AX  ; Khoi dong thanh ghi DS
 MOV ES,AX
```

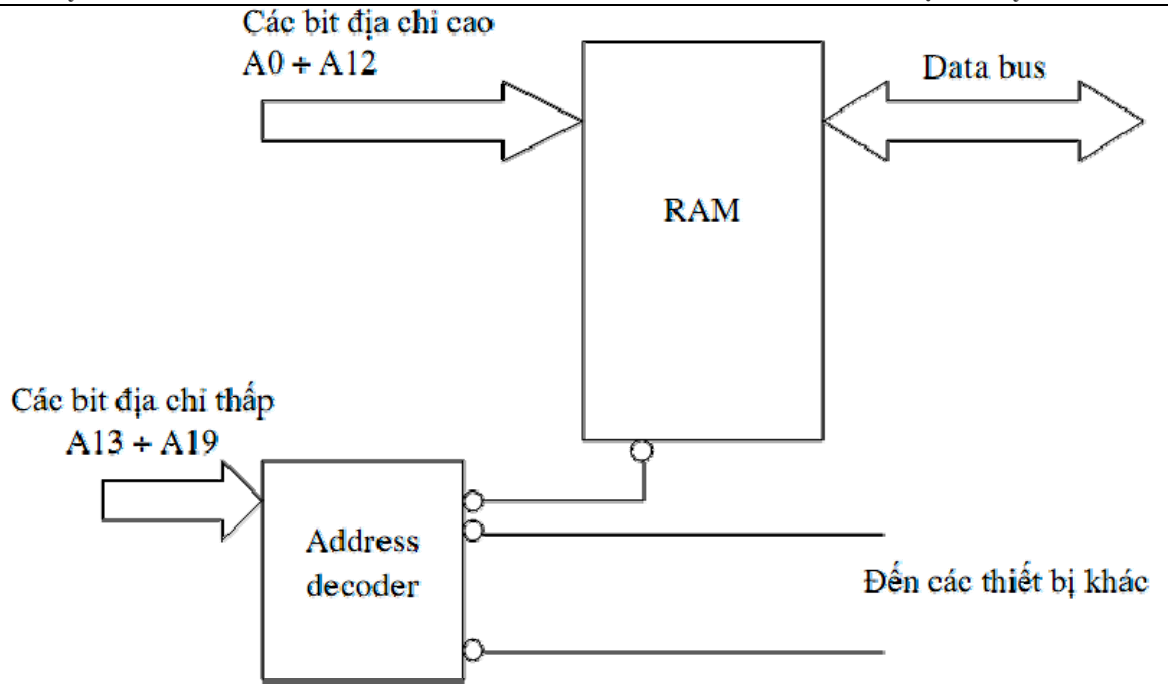
```
 LEA DX,m1
 MOV AH,0Ah  ; Nhap chuoi
 INT 21h
 LEA SI,m1
 ADD SI,2
 MOV DI,SI
Next: LODSB  ; Lay ky tu
 CMP AL,0Dh  ; Neu la ky tu Enter thi ket thuc
 JE quit
 CMP AL,'a'  ; Neu ky tu nhap khong phai la ky tu thuong tu ‘a’
toi ‘z’ thi bo qua
JB cont
CMP AL,'z'
JA cont
     SUB AL,20h  ; Chuyen ky tu thuong thanh ky tu hoa
 STOSB   ; Luu ky tu
 DEC DI   ; Neu la ky tu thuong thi dung lenh STOSB nen DI tang
len 1 ta phai giam DI
cont: INC DI  ;
     JMP next
quit: MOV AL,'$'
STOSB
 MOV AX,02h  ; Xoa man hinh
 INT 10h
 LEA DX,m2
 MOV AH,09h
 INT 21h
 LEA DX,m1+2
MOV AH,09h
 INT 21h
 MOV AH,4Ch
 INT 21h
main ENDP
END main
```

2.5 Ghép nối bộ nhớ và thiết bi ngoại vi

2.5.1 Ghép nối bộ nhớ

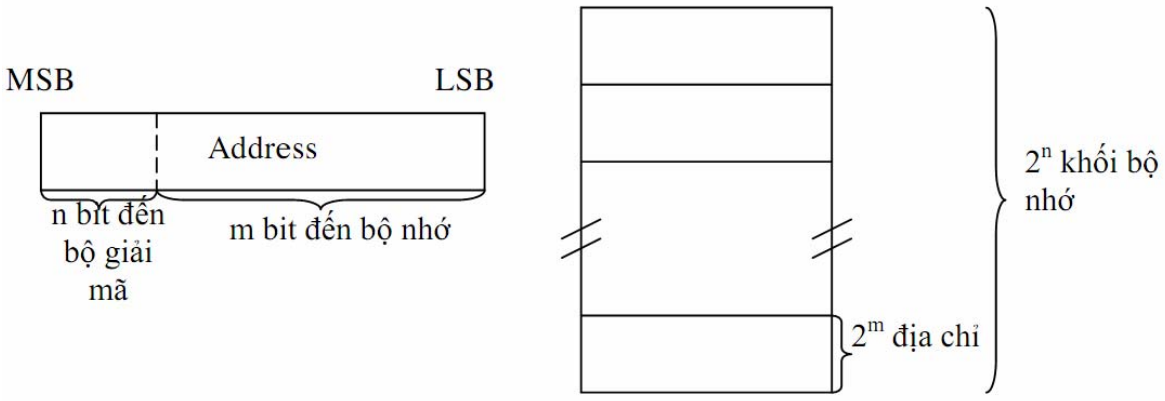
2.5.1.1 Giao tiếp bus cơ bản

- Các bit địa chỉ thấp (giả sử 13 đường A0 ÷ A12) nối trực tiếp đến chip bộ nhớ (giả sử RAM có dung lượng 8K × 8)
- Các bit địa chỉ cao (giả sử A13 ÷ A19) nối với bộ giải mã địa chỉ (address định mỗi chip bộ nhớ thuộc vùng địa chỉ nào. Tập hợp các vùng này theo bảng gọi là bảng bộ nhớ (memory map).



Hình 2-8. Giao tiếp bus cơ bản

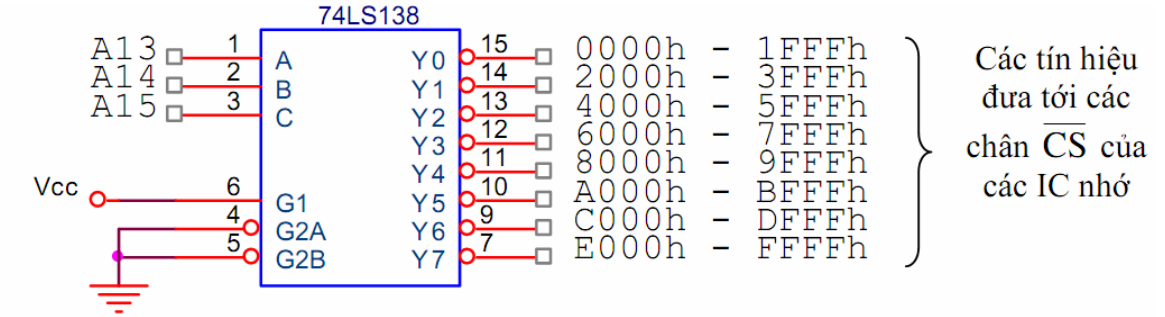
Quan hệ giữa giải mã địa chỉ và bảng bộ nhớ:



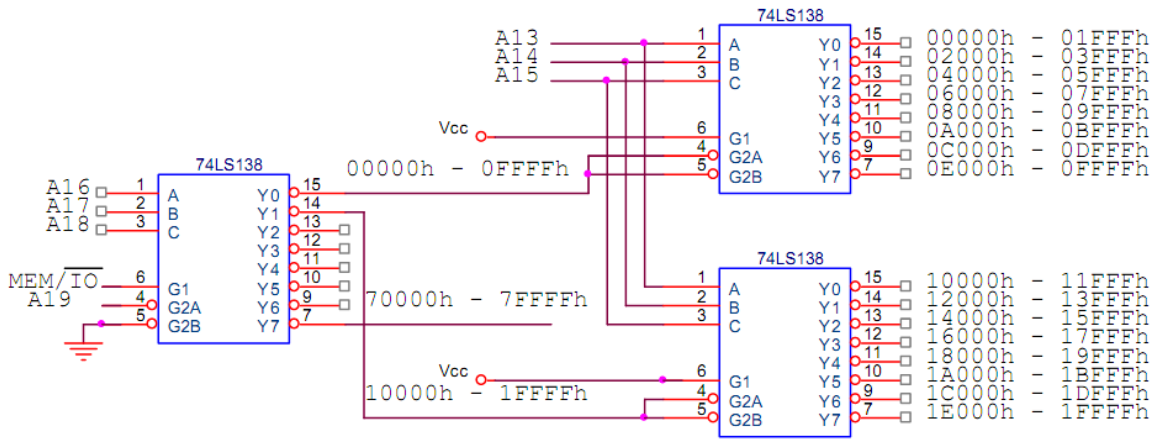
Hình 2-9. Quan hệ giữa giải mã địa chỉ và bộ nhớ

2.5.2 Giải mã địa chỉ

2.5.2.1 Dùng 74LS138



Dùng nhiều 74LS138



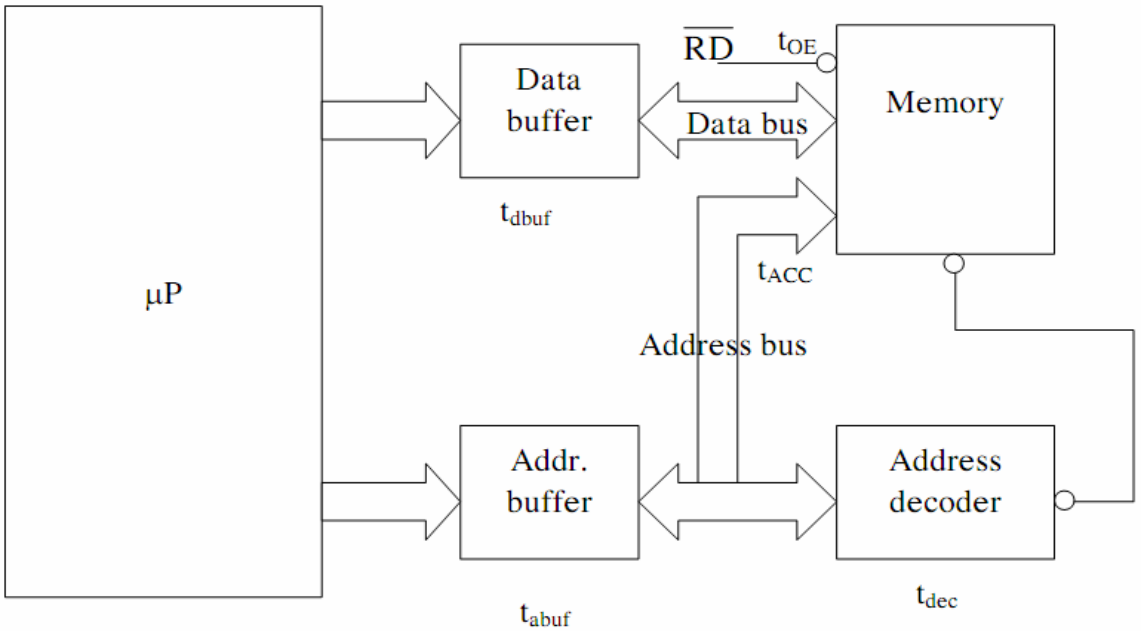
Hình 2-10. Mắc nối tầng nhiều 74LS138

2.5.2.2 Định thời bộ nhớ

- **Thời gian truy xuất (access time):**

- Với chu kỳ đọc: thời gian truy xuất là thời gian tính từ lúc địa chỉ mới xuất hiện ở bộ nhớ cho đến khi có dữ liệu đúng ở ngõ ra của bộ nhớ.
- Với chu kỳ ghi: thời gian truy xuất là thời gian tính từ lúc địa chỉ mới xuất hiện ở bộ nhớ cho đến khi dữ liệu đã đưa vào bộ nhớ.

- **Thời gian chu kỳ (cycle time):** là thời gian từ lúc bắt đầu chu kỳ bộ nhớ đến khi bắt đầu chu kỳ kế tiếp. Ngoài ra, µP có thể sử dụng thêm một số trạng thái chờ khi đọc bộ nhớ.



t_{dbuf}: thời gian trễ ở bộ đệm dữ liệu (data buffer)
t_{abuf}: thời gian trễ ở bộ đệm địa chỉ (address buffer)
t_{OE}: thời gian đáp ứng của bộ nhớ với tín hiệu cho phép ngõ ra (ouput enable)
t_{CS}: thời gian bộ nhớ truy xuất từ Chip Select
t_{ACC}: thời gian bộ nhớ truy xuất từ địa chỉ, thông thường $t_{ACC} = t_{cs}$
t_{dec}: thời gian trễ ở bộ giải mã (decoder)

Hình 2-11. Ghép nối VXL với bộ nhớ

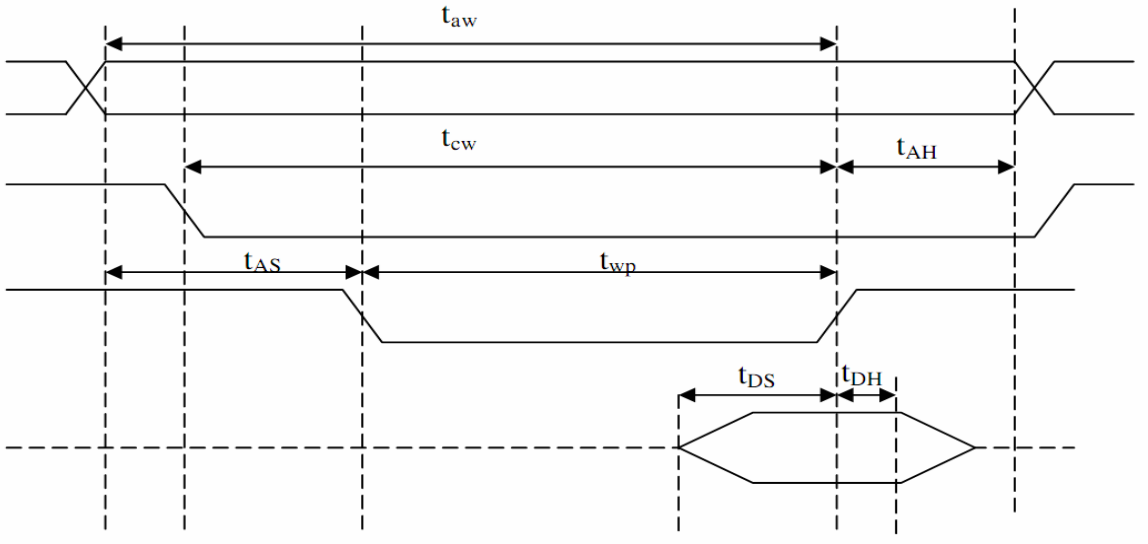
♦ **Định thời đọc bộ nhớ:**

Thời gian truy xuất tổng cộng của hệ thống bộ nhớ chính là tổng thời gian trễ trong các bộ đệm và thời gian truy xuất (access time) bộ nhớ.

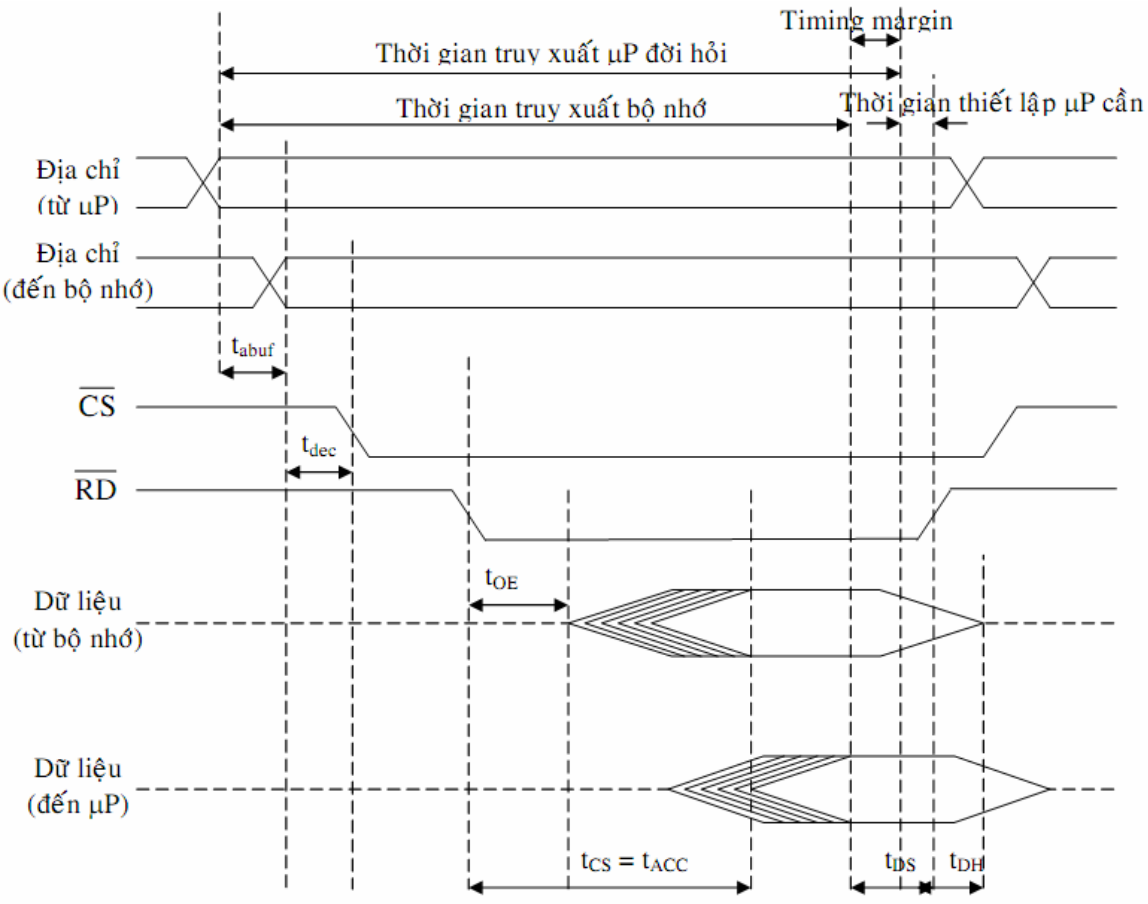
Hiệu giữa thời gian truy xuất cần thiết bởi µP với thời gian truy xuất thật sự của bộ nhớ gọi là biên định thời (timing margin).

tDS (Data Setup): thời gian thiết lập dữ liệu cung cấp bởi hệ thống bộ nhớ

tDH (Data Hold): thời gian giữ dữ liệu cung cấp bởi hệ thống bộ nhớ



Hình 2-12. Định thời ghi bộ nhớ



Hình 2-13. Định thời đọc bộ nhớ

t_{aw}: thời gian truy xuất ghi (access write)

t_{wp}: độ rộng xung ghi tối thiểu (write pulse)

t_{AS}: thời gian địa chỉ hợp lệ trước khi WR = 0

Thông thường, ta không quan tâm đến địa chỉ cho đến khi xác nhận CS nên thường $t_{cw} = t_{aw}$.

2.5.3 Ghép nối thiết bị ngoại vi

2.5.4 Các kiểu giao tiếp vào / ra

Thiết bị ngoại vi có địa chỉ tách rời với bộ nhớ

Trong cách giao tiếp này, bộ nhớ dùng toàn bộ không gian 1 MB. Các thiết bị ngoại vi sẽ có một không gian 64 KB cho mỗi loại cổng. Trong kiểu giao tiếp này, ta phải dùng tín hiệu IO/M và các lệnh trao đổi dữ liệu thích hợp.

Bộ nhớ: IO/M = 0, dùng lệnh MOV

Ngoại vi: IO/M = 1, dùng lệnh IN (nhập) hay OUT (xuất)

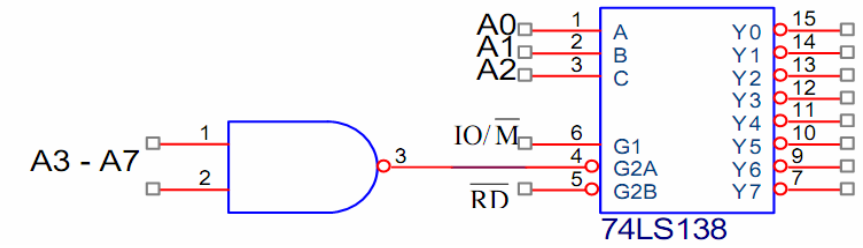
Thiết bị ngoại vi và bộ nhớ có chung không gian địa chỉ

Trong kiểu giao tiếp này, thiết bị ngoại vi sẽ chiếm một vùng nào đó trong không gian địa chỉ 1 MB và ta chỉ dùng lệnh MOV để thực hiện trao đổi dữ liệu.

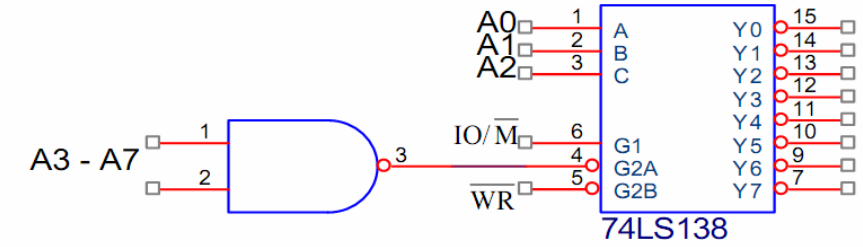
2.5.5 Giải mã địa chỉ cho thiết bị vào / ra

Việc giải mã địa chỉ cho thiết bị ngoại vi cũng tương tự với việc giải mã địa chỉ cho bộ nhớ. Thông thường, các cổng có địa chỉ 8 bit A0 – A7. Tuy nhiên, trong một số hệ vi xử lý, các cổng sẽ có địa chỉ 16 bit.

Ta có thể dùng mạch NAND để tạo tín hiệu chọn cổng nhưng mạch này chỉ có thể giải mã cho 1 cổng. Trong trường hợp cần nhiều tín hiệu chọn cổng, ta có thể dùng bộ giải mã 74LS138 để giải mã cho 8 cổng khác nhau.



a. Giải mã cho cổng vào



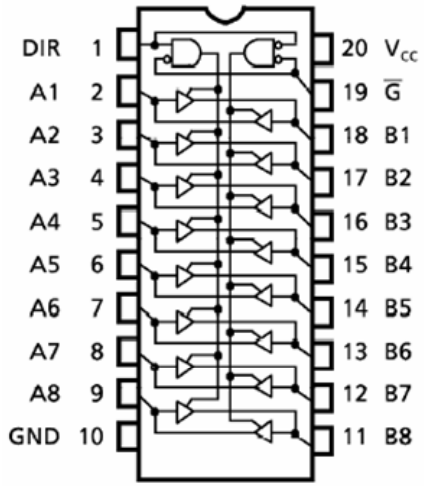
b. Giải mã cho cổng ra

Hình 2-14. Giải mã cho các cổng

2.5.6 Các mạch cổng đơn giản

Các mạch cổng có thể được xây dựng từ các mạch chốt 8 bit (74LS373: kích theo mức, 74LS374: kích theo cạnh), các mạch đệm 8 bit (74LS245). Chúng được dùng trong các giao tiếp đơn giản để µP và ngoại vi hoạt động tương thích với nhau.

Vi mạch đệm 74LS245:



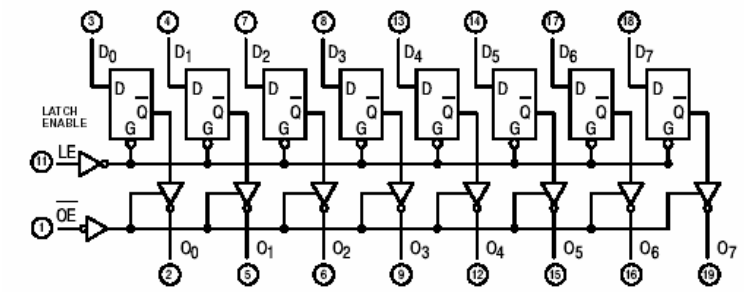
Inputs		Function		Outputs
$\overline{G}$	DIR	A bus	B bus	
L	L	Output	Input	A = B
L	H	Input	Output	B = A
H	X	High Impedance		Z

Hình 2-15. Vi mạch 74LS245

Vi mạch 74LS245 cho tín hiệu vào ra 2 chiều dùng để đệm số liệu trong máy tính PC/XT (VXL 8086). Vi mạch này có 2 đường điều khiển chính, tín hiệu /G là tín hiệu cho phép vi mạch hoạt động, khi /G ở mức cao, các chân dữ liệu của vi mạch ở trạng thai trở kháng cao.

Tín hiệu DIR xác định chiều truyền dữ iệu. DIR = 1 dữ liêu được truyền từ A sang B, ngược lại, khi DIR = 0 dữ liệu được truyền từ B sang A

Vi mạch chốt 74LS373:



LS373

D_n	LE	$\overline{OE}$	O_n
H	H	L	H
L	H	L	L
X	L	L	Q_0
X	X	H	Z*

H = HIGH Voltage Level
L = LOW Voltage Level
X = Immaterial
Z = High Impedance

Hình 2-16. Vi mạch chốt 74LS373

Vi mạch bao gồm các vi mạch chốt và các vi mạch cổng 3 trạng thái. Vi mạch này thường được dùng để chốt địa chỉ trong máy PC/XT và chốt dữ liêu trong các ứng dụng ghép nối máy tính. Có 2 đường tín hiệu điều khiển là /OE và LE. Tín hiệu /OE là tín hiệu cho phép hoạt động của vi mạch. Khi /OE ở mức cao, các cổng của vi mạch ở trạng thái trở kháng cao. Tín hiệu LE là tín hiệu cho phép chốt, tín hiệu này tích cực ở mức dương. Đối với 74LS373, khi LE ở mức cao, tín hiệu đưa vào từ cổng D được đưa ra cổng Q. Khi LE chuyển sang mức thấp, tín hiệu ở cổng Q được chốt lại.

2.6 Câu hỏi và bài tập

Bài 1. Viết CT nhập vào 1 ký tự, xuất ra ký tự đó

Ví dụ:

Moi ban nhap 1 ky tu: b
Ky tu vừa nhập: b

Bài 2. Viết chương trình xuất ra màn hình một số dòng.

Ví dụ:

De chay duoc 1 CT hop ngu ban can thuc hien cac buoc sau:
Dich file ASM thanh file OBJ
Lien ket file OBJ thanh file EXE
Chay file EXE

Bài 3. Viết CT nhập vào 1 ký tự, xuất ra ký tự liền trước và liền sau.

Ví dụ:

Moi ban nhap 1 ky tu: b
Ky tu lien truoc: a
Ky tu lien sau: c

Bài 4. Viết CT nhập vào 1 ký tự thường. In ra ký tự Hoa

Ví dụ:

Moi ban nhap 1 ky tu: b
Ky tu Hoa: B

Bài 5. Viết CT nhập vào 1 ký tự hoa. In ra ký tự thường

Ví dụ:

Moi ban nhap 1 ky tu: B
Ky tu thường: b

Bài 6. Viết chương trình nhập vào 2 số nguyên dương x1, x2 ($1 \leq x2 < x1 < 9$). Xuất ra kết quả các phép tính: x1-1, x1 +2, x1+x2, x1-x2

Ví dụ:

x1 = 5
x2 = 3
x1 – 1 = 4
x1 + 1 = 6
x1 + x2 = 8
x1 – x2 = 7

Mở rộng

1. Tự tìm hiểu xem hàm nào trong ngắt 21h dùng để nhập một xâu kí tự ? Ngoài ngắt 21h, còn ngắt nào có thể dùng để nhập xuất từ bàn phím ? (dùng NortonGuide hoặc TechHelp).
2. Viết chương trình nhập tên và in ra màn hình câu "Hello " + tên đã nhập.
3. Tìm hiểu xem tại sao không có lệnh **MOV** x1, x2 (x1,x2 là hai biến trong bộ nhớ)
4. Hai lệnh "INC AX" và "ADD AX, 1" khác nhau chỗ nào ?

Hướng dẫn

Bài 1. Để nhập 1 một ký tự sử dụng hàm 1 của ngắt 21h, để xuất, sử dụng hàm 2.

Ví dụ:

```
MOV AH,1
int 21h          ; kết quả trong AL

MOV DL,AL        ; kí tự cần xuất trong DL
MOV AH,2
int 21h
```

Bài 2. Cặp kí tự xuống dòng là 10,13. Có thể khai báo nhiều xâu kí tự hoặc chung một xâu.

Ví dụ:

```
Msg3  DB 10,13,9,"1. Dich file ASM thanh file OBJ.$"
Msg4  DB 10,13,9,"2. Lien ket file OBJ thanh file EXE.$"
```

Hoặc

```
Msg34    DB 10,13,9,"1. Dich file ASM thanh file OBJ."
         DB 10,13,9,"2. Lien ket file OBJ thanh file EXE.$"
```

Bài 3, 4. Kí tự hoa và kí tự thường của cùng một chữ cái tiếng Anh cách nhau 20h. Do đó, để chuyển đổi chữ hoa thành chữ thường và ngược lại, chỉ cần dùng lệnh ADD, SUB.

Bài 5. Để chuyển đổi các kí tự '0' – '9' thành số 0 – 9 chỉ cần thực hiện phép trừ đi 48 (mã của '0'). Sau khi thực hiện phép tính, chuyển đổi thành kí tự và in ra màn hình (có thể dùng biểu diễn Hex).

(Trang này nên bỏ trống)

CHƯƠNG 3.
HỌ VI ĐIỀU KHIỂN 8051

Mục tiêu:

Giúp sinh viên hiểu được cấu trúc phần cứng, sơ đồ chân và các mạch phụ trợ của họ vi điều khiển 8051; nắm được và biết cách vận dụng các chế độ địa chỉ trong lập trình; nắm được tập lệnh và phương pháp lập trình cho họ vi điều khiển 8051.

Tóm tắt học phần:

Cấu trúc phần cứng và tổ chức bộ nhớ

Giới thiệu chung
Sơ đồ cấu trúc
Mô tả chức năng các chân
Hoạt động Reset
Tổ chức bộ nhớ

Các chế độ định địa chỉ

Tập lệnh

Lập trình hợp ngữ (Assembly) cho vi điều khiển 8051

Trình dịch hợp ngữ
Cổng vào/ra và lập trình
Bộ đếm/định thời và lập trình
Lập trình ngắt

3.1 Giới thiệu chung

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

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

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

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

điều khiển của nhiều nhà chế tạo khác nhau lần lượt được đưa ra thị trường với tính năng được cải tiến ngày càng mạnh.

3.1.1 Ứng dụng của vi điều khiển

Về cơ bản, vi điều khiển rất đơn giản. Chúng chỉ bao gồm tối thiểu một số thành phần sau:

- Một bộ vi xử lý tối giản được sử dụng như bộ não của hệ thống
- Tùy theo công nghệ của mỗi hãng sản xuất, có thể có thêm bộ nhớ, các chân nhập/xuất tín hiệu, bộ đếm, bộ định thời, các bộ chuyển đổi tương tự/số (A/D), …
- Tất cả chúng được đặt trong một vỏ chíp tiêu chuẩn.
- Một phần mềm đơn giản có thể điều khiển được toàn bộ hoạt động của vi điều khiển và có thể dễ dàng cho người sử dụng nắm bắt.

Dựa trên nguyên tắc cơ bản trên, rất nhiều họ vi điều khiển đã được phát triển và ứng dụng một cách thầm lặng nhưng mạnh mẽ vào mọi mặt của đời sống của con người. Một số ứng dụng cơ bản thành công có thể kể ra sau đây:

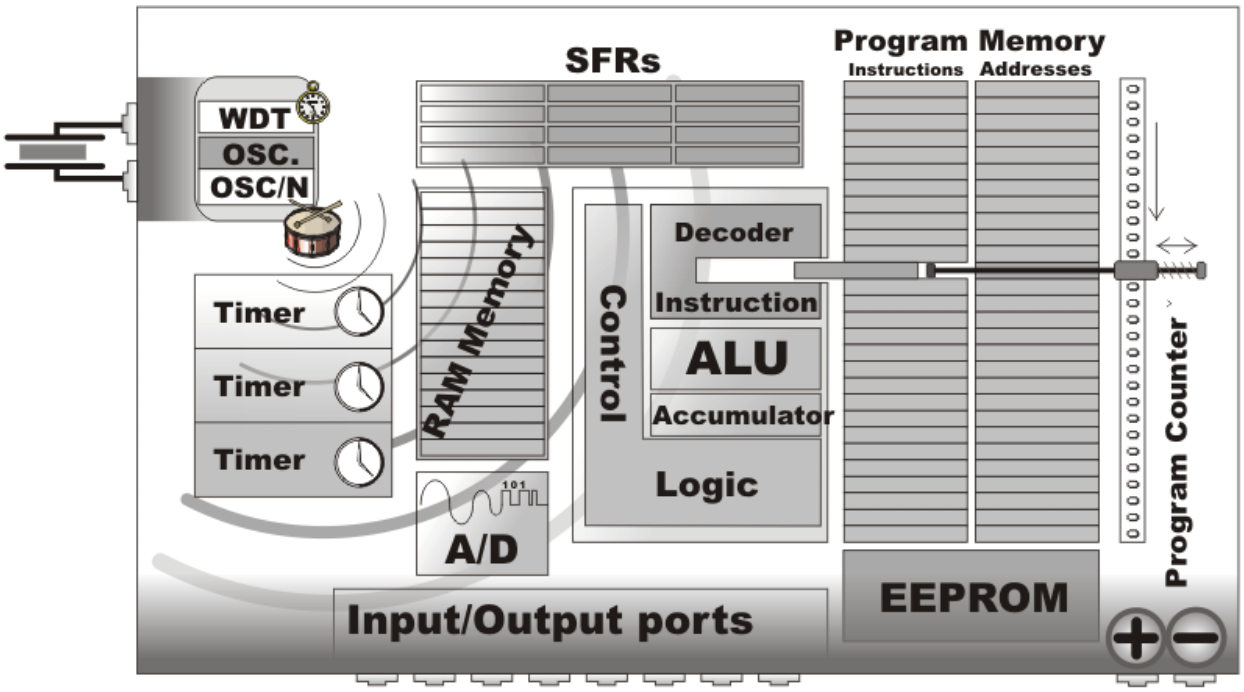
- Những thành phần điện tử được nhúng vào vi điều khiển có thể trực tiếp hoặc qua các thiết bị vào ra (công tắc, nút bấm, cảm biến, LCD, rơ le, …) điều khiển rất nhiều thiết bị và hệ thống như thiết bị tự động trong công nghiệp, điều khiển nhiệt độ, dòng điện, động cơ, …
- Giá thành rất thấp khiến cho chúng được nhúng vào rất nhiều thiết bị thông minh trong đời sống con người như ti vi, máy giặt, điều hòa nhiệt độ, máy nghe nhạc, …

3.1.2 Hoạt động của vi điều khiển.

Mặc dù đã có rất nhiều họ vi điều khiển được phát triển cũng như nhiều chương trình điều khiển tạo ra cho chúng, nhưng tất cả chúng vẫn có một số điểm chung cơ bản. Do đó nếu ta hiểu cặn kẽ một họ thì việc tìm hiểu thêm một họ vi điều khiển mới là hoàn toàn đơn giản. Một kịch bản chung cho hoạt động của một vi điều khiển như sau:

1. Khi không có nguồn điện cung cấp, vi điều khiển chỉ là một con chip có chương trình nạp sẵn vào trong đó và không có hoạt động gì xảy ra.
2. Khi có nguồn điện, mọi hoạt động bắt đầu được xảy ra với tốc độ cao. Đơn vị điều khiển logic có nhiệm vụ điều khiển tất cả mọi hoạt động. Nó khóa tất cả các mạch khác, trừ mạch giao động thạch anh. Sau mini giây đầu tiên tất cả đã sẵn sàng hoạt động.

3. Điện áp nguồn nuôi đạt đến giá trị tối đa của nó và tần số giao động trở nên ổn định. Các bit của các thanh ghi SFR cho biết trạng thái của tất cả các mạch trong vi điều khiển. Toàn bộ vi điều khiển hoạt động theo chu kỳ của chuỗi xung chính.
4. Thanh ghi bộ đếm chương trình (Program Counter) được xóa về 0. Câu lệnh từ địa chỉ này được gửi tới bộ giải mã lệnh sau đó được thực thi ngay lập tức.
5. Giá trị trong thanh ghi PC được tăng lên 1 và toàn bộ quá trình được lặp lại vài … triệu lần trong một giây.



Hình 3-1. Cấu trúc chung họ VĐK

3.1.3 Cấu trúc chung của vi điều khiển

Như ta thấy, tất cả các hoạt động trong các vi điều khiển được thực hiện ở tốc độ cao và khá đơn giản, nhưng vi điều khiển chính nó sẽ không được thật sự hữu ích nếu không có mạch đặc biệt làm cho nó hoàn thiện. Có một số mạch cụ thể sau đây.

❖ Read Only Memory (ROM)

Read Only Memory (ROM) là một loại bộ nhớ được sử dụng để lưu vĩnh viễn các chương trình được thực thi. Kích cỡ của chương trình có thể được viết phụ thuộc vào kích cỡ của bộ nhớ này. ROM có thể được tích hợp trong vi điều khiển hay thêm vào như là một chip gắn bên ngoài, tùy thuộc vào loại vi điều khiển. Cả hai tùy chọn có một số nhược điểm. Nếu ROM được thêm vào như là một chip bên ngoài, các vi điều khiển là rẻ hơn và các chương trình có thể tồn tại lâu hơn đáng kể. Nhưng đồng thời, làm giảm số lượng các chân vào/ra để vi điều khiển sử dụng với mục đích khác.

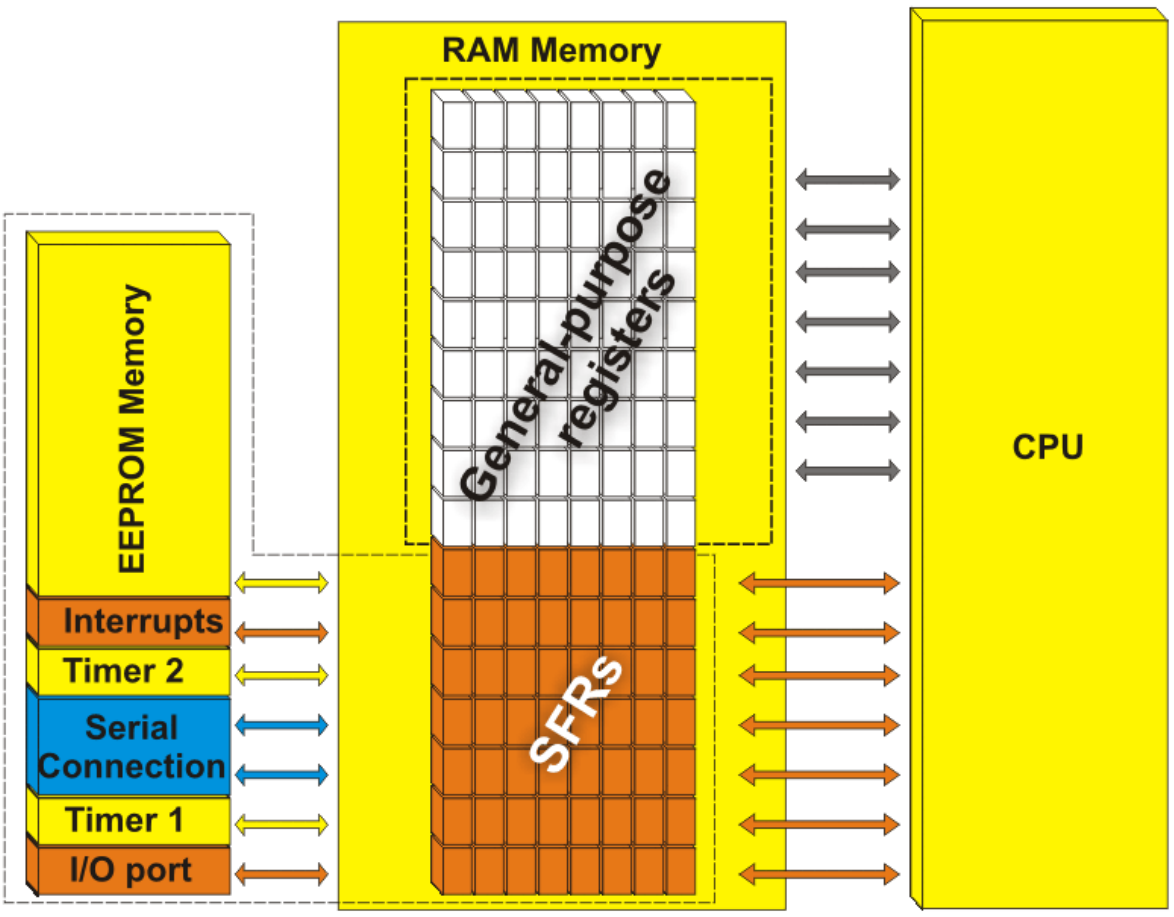
ROM nội thường là nhỏ hơn và đắt tiền hơn, nhưng lá ghim thêm có sẵn để kết nối với môi trường ngoại vi. Kích thước của dãy ROM từ 512B đến 64KB

❖ Random Access Memory (RAM)

Random Access Memory (RAM) là một loại bộ nhớ sử dụng cho các dữ liệu lưu trữ tạm thời và kết quả trung gian được tạo ra và được sử dụng trong quá trình hoạt động của bộ vi điều khiển. Nội dung của bộ nhớ này bị xóa một khi nguồn cung cấp bị tắt.

❖ Electrically Erasable Programmable ROM (EEPROM)

EEPROM là một kiểu đặc biệt của bộ nhớ chỉ có ở một số loại vi điều khiển. Nội dung của nó có thể được thay đổi trong quá trình thực hiện chương trình (tương tự như RAM), nhưng vẫn còn lưu giữ vĩnh viễn, ngay cả sau khi mất điện (tương tự như ROM). Nó thường được dùng để lưu trữ các giá trị được tạo ra và được sử dụng trong quá trình hoạt động (như các giá trị hiệu chuẩn, mã, các giá trị để đếm, v.v..), mà cần phải được lưu sau khi nguồn cung cấp ngắt. Một bất lợi của bộ nhớ này là quá trình ghi vào là tương đối chậm.



Hình 3-2 Giao tiếp bộ nhớ

❖ Các thanh ghi chức năng đặc biệt (SFR)

Thanh ghi chức năng đặc biệt (Special Function Registers) là một phần của bộ nhớ RAM. Mục đích của chúng được định trước bởi nhà sản xuất và không thể thay đổi được. Các bit của chúng được liên kết vật lý tới các mạch trong vi điều khiển như bộ chuyển đổi A/D, modul truyền thông nối tiếp,… Mỗi sự thay đổi trạng thái của các bit sẽ tác động tới hoạt động của vi điều khiển hoặc các vi mạch.

❖ Bộ đếm chương trình (PC:Program Counter)

Bộ đếm chương trình chứa địa chỉ chỉ đến ô nhớ chứa câu lệnh tiếp theo sẽ được kích hoạt. Sau mỗi khi thực hiện lệnh, giá trị của bộ đếm được tăng lên 1. Vì lý do đó nên chương trình chỉ thực hiện được được từng lệnh trong một thời điểm.

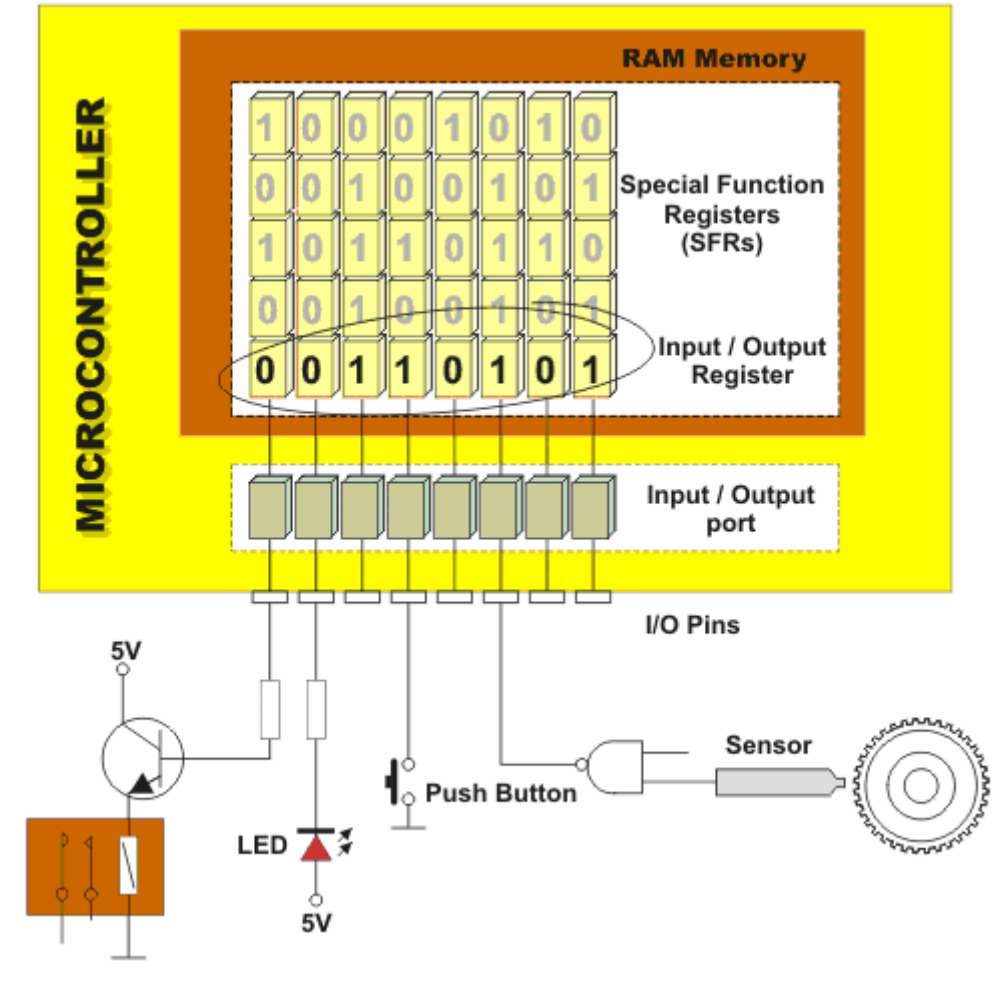
❖ Central Processor Unit (CPU)

Đây là một đơn vị có nhiệm vụ điều khiển và giám sát tất cả các hoạt động bên trong vi điều khiển và người sử dụng không thể tác động vào hoạt động của nó. Nó bao gồm một số đơn vị con nhỏ hơn, trong đó quan trọng nhất là:

- ***Instruction decoder*** is a part of the electronics which recognizes program instructions and runs other circuits on the basis of that. The abilities of this circuit are expressed in the "instruction set" which is different for each microcontroller family.
- ***Bộ giải mã lệnh*** có nhiệm vụ nhận dạng câu lệnh và điều khiển các mạch khác theo lệnh đã giải mã. Việc giải mã đươpcj thực hiện nhờ có tập lệnh “instruction set”. Mỗi họ vi điều khiển thường có các tập lệnh khác nhau.
- ***Arithmetical Logical Unit (ALU)*** Thực thi tất cả các thao tác tính toán số học và logic.
- ***Thanh ghi tích lũy (Accumulator)*** là một thanh ghi SFR liên quan mật thiết với hoạt động của ALU. Nó lưu trữ tất cả các dữ liệu cho quá trình tính toán và lưu giá trị kết quả để chuẩn bị cho các tính toán tiếp theo. Một trong các thanh ghi SFR khác được gọi là thanh ghi trạng thái (Status Register) cho biết trạng thái của các giá trị lưu trong thanh ghi tích lũy.

❖ Các cổng vào/ra (I/O Ports)

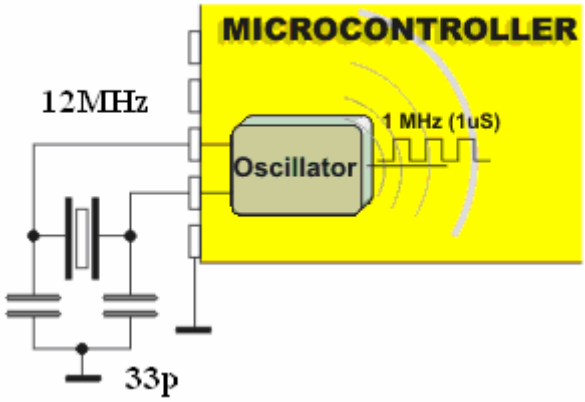
Để vi điều khiển có thể hoạt động hữu ích, nó cần có sự kết nối với các thiết bị ngoại vi. Mỗi vi điều khiển sẽ có một hoặc một số thanh ghi (được gọi là cổng) được kết nối với các chân của vi điều khiển.



Hình 3-3. Vào ra với thiết bị ngoại vi

Chúng được gọi là cổng vào/ra (I/O port) bởi vì chúng có thể thay đổi chức năng, chiều vào/ra theo yêu cầu của người dùng.

❖ Bộ dao động (Oscillator)



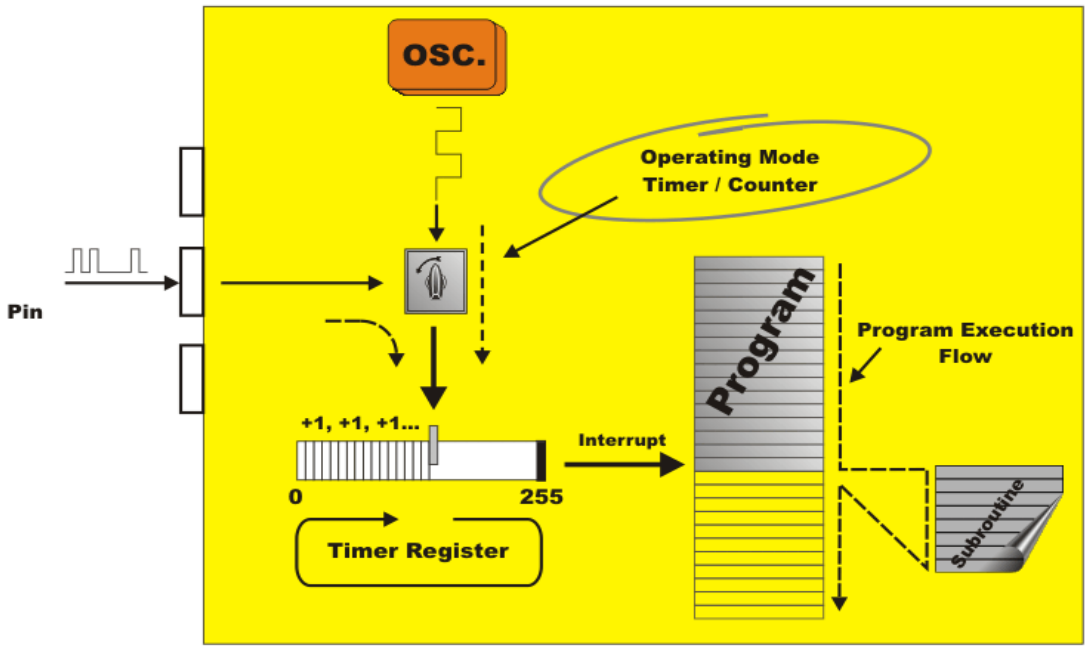
Hình 3-4 ghép nối bộ dao động

Bộ dao động đóng vai trò nhạc trưởng làm nhiệm vụ đồng bộ hóa hoạt động của tất cả các mạch bên trong vi điều khiển. Nó thường được tạo bởi thạch anh hoặc gốm để ổn định tần số. Các lệnh không được thực thi theo tốc độ của bộ dao động mà thường

chậm hơn, bởi vì mỗi câu lệnh được thực hiện qua nhiều bước. Mỗi loại vi điều khiển cần số chu kỳ khác nhau để thực hiện lệnh.

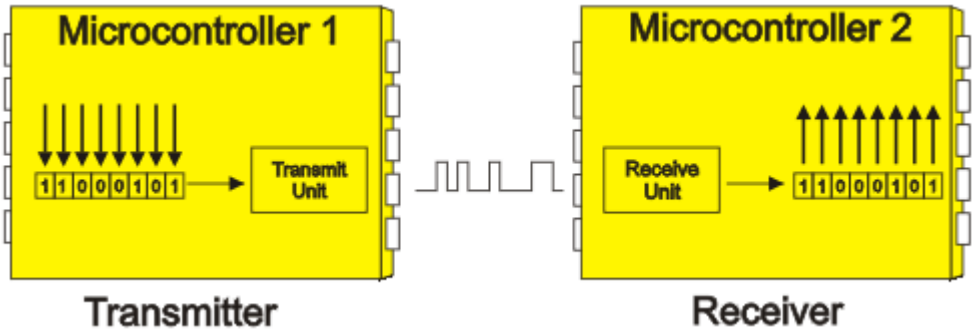
❖ Bộ định thời/đếm (Timers/Counters)

Hầu hết các chương trình sử dụng các bộ định thời trong hoạt động của mình. Chúng thường là các thanh ghi SFR 8 hoặc 16 bit, sau mỗi xung dao động clock, giá trị của chúng được tăng lên. Ngay khi thanh ghi tràn, một ngắt sẽ được phát sinh.



Hình 3-5. Bộ định thời/đếm

❖ Truyền thông nối tiếp



Hình 3-6. Truyền nhận nối tiếp

Kết nối song song giữa vi điều khiển và thiết bị ngoại vi được thực hiện qua các cổng vào/ra là giải pháp lý tưởng với khoảng cách ngắn trong vài mét. Tuy nhiên khi cần truyền thông giữa các thiết bị ở khoảng cách xa thì không thể dùng kết nối song song, vì vậy truyền thông nối tiếp là giải pháp tốt nhất.

Ngày nay, hầu hết các vi điều khiển có một số bộ điều khiển truyền thông nối tiếp như một trang bị tiêu chuẩn. Chúng được sử dụng phụ thuộc vào nhiều yếu tố khác nhau như:

- Bao nhiêu thiết bị vi điều khiển muốn trao đổi dữ liệu
- Tốc độ trao đổi dữ liệu
- Khoảng cách truyền
- Truyền/nhận dữ liệu đồng thời hay không?

❖ Chương trình

Không giống như các mạch tích hợp, chỉ cần kết nối các thành phần với nhau và bật nguồn, vi điều khiển cần phải lập trình trước. Để viết một chương trình cho vi điều khiển, có một vài ngôn ngữ lập trình bậc thấp có thể sử dụng như Assembly, C hay Basic. Viết một chương trình bao gồm việc viết các câu lệnh đơn giản theo một thứ tự để chúng có thể thực thi. Có rất nhiều phần mềm chạy trên môi trường Windows cho phép xây dựng các chương trình hoàn chỉnh cho các họ vi điều khiển

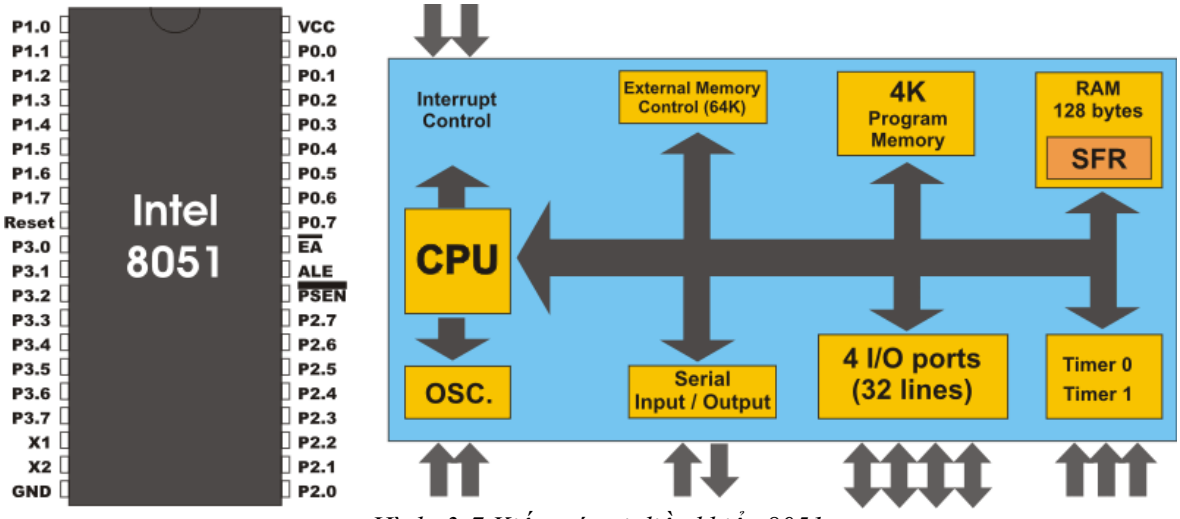
3.2 Kiến trúc vi điều khiển 8051

3.2.1 Chuẩn 8051

Họ vi điều khiển MCS-51 do Intel sản xuất đầu tiên vào năm 1980 là các IC thiết kế cho các ứng dụng hướng điều khiển. Các IC này chính là một hệ thống vi xử lý hoàn chỉnh bao gồm các các thành phần của hệ vi xử lý: CPU, bộ nhớ, các mạch giao tiếp, điều khiển ngắt.

MCS-51 là họ vi điều khiển sử dụng cơ chế CISC (Complex Instruction Set Computer), có độ dài và thời gian thực thi của các lệnh khác nhau. Tập lệnh cung cấp cho MCS-51 có các lệnh dùng cho điều khiển xuất/nhập tác động đến từng bit. MCS-51 bao gồm nhiều vi điều khiển khác nhau, bộ vi điều khiển đầu tiên là 8051 có 4KB ROM, 128 byte RAM và 8031, không có ROM nội, phải sử dụng bộ nhớ ngoài. Sau này, các nhà sản xuất khác như Siemens, Fujitsu, … cũng được cấp phép làm nhà cung cấp thứ hai.

MCS-51 bao gồm nhiều phiên bản khác nhau, mỗi phiên bản sau tăng thêm một số thanh ghi điều khiển hoạt động của MCS-51.

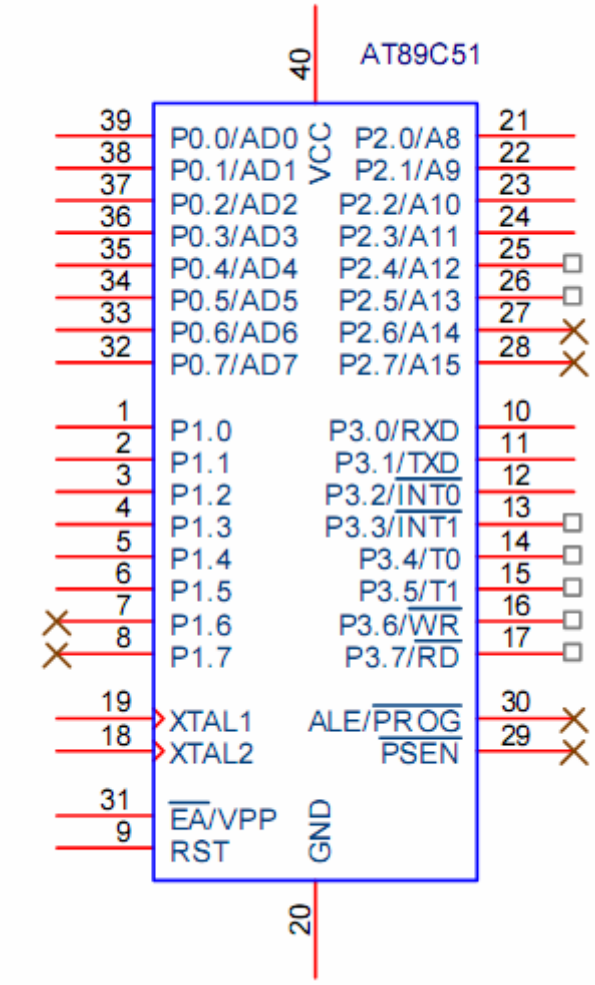


Hình 3-7.Kiến trúc vi điều khiển 8051

AT89C51 là vi điều khiển do Atmel sản xuất, chế tạo theo công nghệ CMOS có các đặc tính như sau:

- 4 KB PEROM (Flash Programmable and Erasable Read Only Memory), có khả năng tới 1000 chu kỳ ghi xoá
- Tần số hoạt động từ: 0Hz đến 24 MHz
- 3 mức khóa bộ nhớ lập trình
- 128 Byte RAM nội.
- 4 Port xuất /nhập I/O 8 bit.
- 2 bộ Timer/counter 16 Bit.
- 6 nguồn ngắt.
- Giao tiếp nối tiếp điều khiển bằng phần cứng.
- 64 KB vùng nhớ mã ngoài
- 64 KB vùng nhớ dữ liệu ngoài.
- Cho phép xử lý bit.
- 210 vị trí nhớ có thể định vị bit.
- 4 chu kỳ máy (4 µs đối với thạch anh 12MHz) cho hoạt động nhân hoặc chia.
- Có các chế độ nghỉ (Low-power Idle) và chế độ nguồn giảm (Power-down).
- Ngoài ra, một số IC khác của họ MCS-51 có thêm bộ định thời thứ 3 và 256 byte RAM nội.

3.2.2 Chân vi điều khiển 8051



Hình 3-8. Sơ đồ chân VĐK AT89C51

Chip AT89C51 có các tín hiệu điều khiển cần phải lưu ý như sau:

- Tín hiệu vào **/EA** trên chân 31 thường đặt lên mức cao (+5V) hoặc mức thấp (GND). Nếu ở mức cao, 8951 thi hành chương trình từ ROM nội trong khoảng địa chỉ thấp (4K hoặc tối đa 8k đối với 89C52). Nếu ở mức thấp, chương trình được thi hành từ bộ nhớ mở rộng (tối đa đến 64Kbyte). Ngoài ra người ta còn dùng /EA làm chân cấp điện áp 12V khi lập trình EEPROM trong 8051.

❖ Chân PSEN (Program store enable):

PSEN là chân tín hiệu ra trên chân 29. Nó là tín hiệu điều khiển cho phép chương trình mở rộng, PSEN thường được nối đến chân /OE (Output Enable) của một EPROM hoặc ROM để cho phép đọc các bytes mã lệnh.

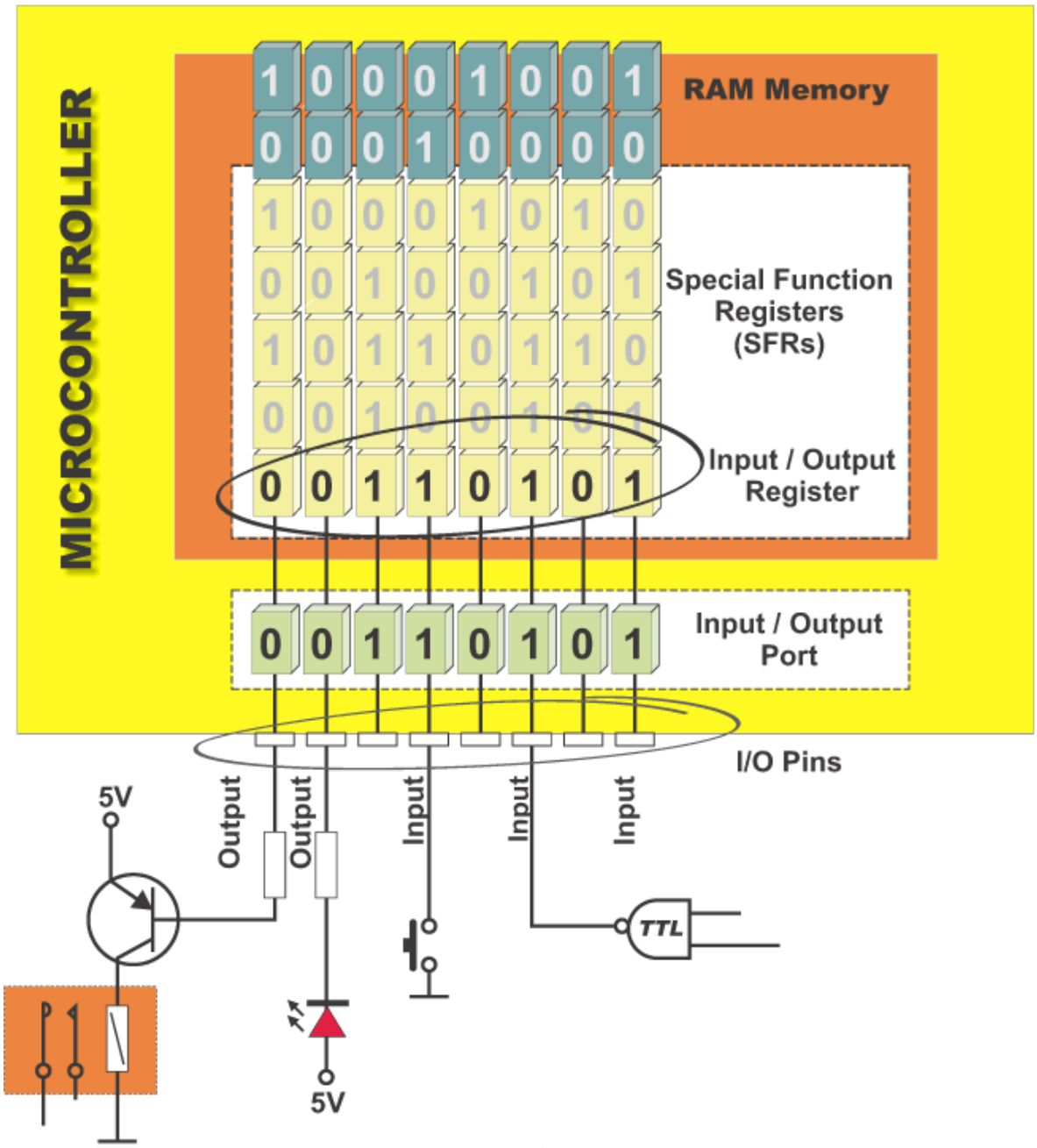
Hãy nhớ rằng : bình thường chân /PSEN sẽ được thả trống (No Connect).Chỉ khi nào cho /EA ở mức thấp thì lúc đó: /PSEN sẽ ở mức thấp trong thời gian lấy lệnh. Các mã nhị phân của chương trình được lấy từ EPROM qua bus dữ liệu và được chốt vào thanh ghi lệnh của 8951 để giải mã lệnh. /PSEN ở mức thụ động (mức cao) nếu thi hành chương trình trong ROM nội của 8951.

❖ CÁC CHÂN NGUỒN:

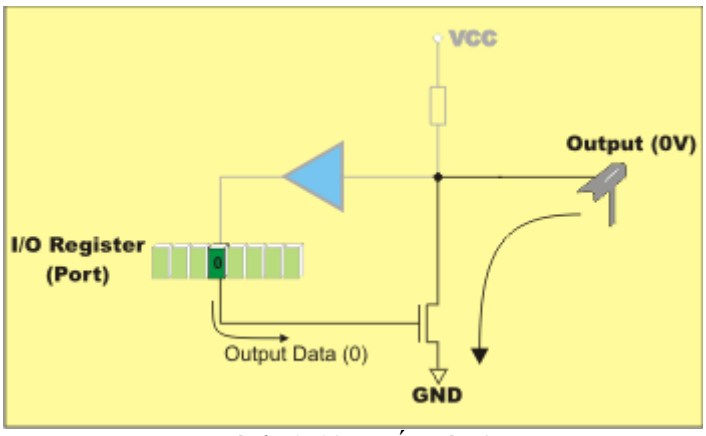
AT89C51 hoạt động ở nguồn đơn +5V. Vcc được nối vào chân 40, và Vss (GND) được nối vào chân 20.

3.2.3 Cổng vào/ra

Tất cả các vi điều khiển 8051 đều có 4 cổng vào/ra 8 bit có thể thiết lập như cổng vào hoặc ra. Như vậy có tất cả 32 chân I/O cho phép vi điều khiển có thể kết nối với các thiết bị ngoại vi.



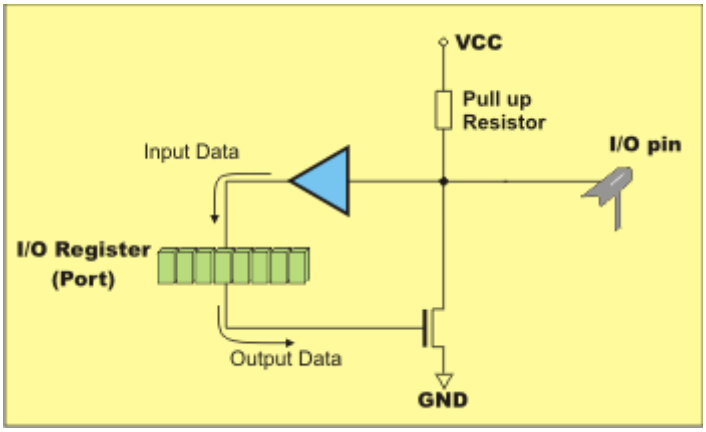
Hình 3-9. Cổng vào/ra



Hình 3-10. Xuất mức 0

❖ Chân vào/ra (I/O)

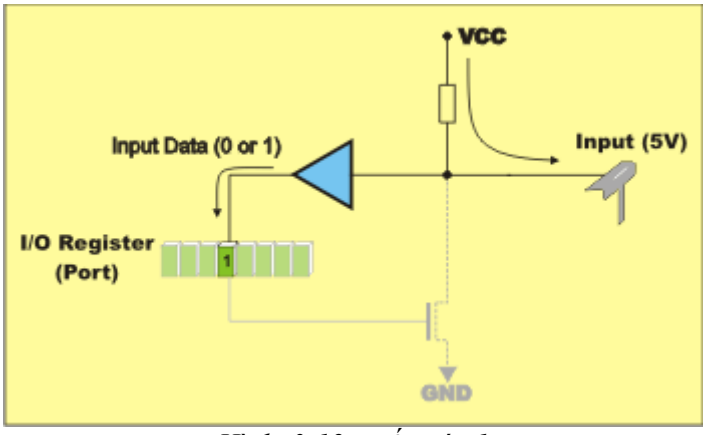
Hình trên mô tả sơ đồ đơn giản của mạch bên trong các chân vi điều khiển trừ cổng P0 là không có điện trở kéo lên (pull-up).



Hình 3-11. Trở treo nội tại chân

❖ Chân ra

Một mức logic 0 đặt vào bit của thanh ghi P làm cho transistor mở, nối chân tương ứng với đất.



Hình 3-12. xuất mức 1

❖ Chân vào

Một bit 1 đặt vào một bit của thanh ghi cổng, transistor đóng và chân tương ứng được nối với nguồn Vcc qua trở kéo lên.

❖ Port 0

Port 0 là port có 2 chức năng ở các chân 32 – 39 của AT89C51:

- Chức năng I/O (xuất/nhập): dùng cho các thiết kế nhỏ. Tuy nhiên, khi dùng chức năng này thì Port 0 phải dùng thêm các điện trở kéo lên (pull-up), giá trị của điện trở phụ thuộc vào thành phần kết nối với Port.
- Khi dùng làm ngõ vào, Port 0 phải được set mức logic 1 trước đó.
- Chức năng địa chỉ / dữ liệu đa hợp: khi dùng các thiết kế lớn, đòi hỏi phải sử dụng bộ nhớ ngoài thì Port 0 vừa là bus dữ liệu (8 bit) vừa là bus địa chỉ (8 bit thấp).

Ngoài ra khi lập trình cho AT89C51, Port 0 còn dùng để nhận mã khi lập trình và xuất mã khi kiểm tra (quá trình kiểm tra đòi hỏi phải có điện trở kéo lên).

❖ Port 1:

Port1 (chân 1 – 8) chỉ có một chức năng là I/O, không dùng cho mục đích khác (chỉ trong 8032/8052/8952 thì dùng thêm P1.0 và P1.1 cho bộ định thời thứ 3). Tại Port 1 đã có điện trở kéo lên nên không cần thêm điện trở ngoài.

Port 1 có khả năng kéo được 4 ngõ TTL và còn dùng làm 8 bit địa chỉ thấp trong quá trình lập trình hay kiểm tra.

Khi dùng làm ngõ vào, Port 1 phải được set mức logic 1 trước đó.

❖ Port 2:

Port 2 (chân 21 – 28) là port có 2 chức năng:

- Chức năng I/O (xuất / nhập)
- Chức năng địa chỉ: dùng làm 8 bit địa chỉ cao khi cần bộ nhớ ngoài có địa chỉ 16 bit. Khi đó, Port 2 không được dùng cho mục đích I/O.
- Khi dùng làm ngõ vào, Port 2 phải được set mức logic 1 trước đó.
- Khi lập trình, Port 2 dùng làm 8 bit địa chỉ cao hay một số tín hiệu điều khiển.

❖ Port 3:

Port 3 (chân 10 – 17) là port có 2 chức năng:

- Chức năng I/O. Khi dùng làm ngõ vào, Port 3 phải được set mức logic 1 trước đó.
- Chức năng khác: mô tả như sau:

Bit	Tên	Chức năng
P3.0	RxD	Ngõ vào port nối tiếp
P3.1	TxD	Ngõ ra port nối tiếp
P3.2	INT0	Ngắt ngoài 0
P3.3	INT1	Ngắt ngoài 1
P3.4	T0	Ngõ vào của bộ định thời 0
P3.5	T1	Ngõ vào của bộ định thời 1
P3.6	WR	Tín hiệu điều khiển ghi dữ liệu lên bộ nhớ ngoài.
P3.7	RD	Tín hiệu điều khiển đọc từ bộ nhớ dữ liệu ngoài.

Bảng 3-1. Chức năng các chân của Port 3

❖ Các chân nguồn:

Chân 40: VCC = 5V± 20%
Chân 20: GND

❖ /PSEN (Program Store Enable):

/PSEN (chân 29) cho phép đọc bộ nhớ chương trình mở rộng đối với các ứng dụng sử dụng ROM ngoài, thường được nối đến chân /OC (Output Control) của ROM để đọc các byte mã lệnh. /PSEN sẽ ở mức logic 0 trong thời gian AT89C51 lấy lệnh.Trong quá trình này, / PSEN sẽ tích cực 2 lần trong 1 chu kỳ máy.

Mã lệnh của chương trình được đọc từ ROM thông qua bus dữ liệu (Port0) và bus địa chỉ (Port0 + Port2).

Khi 8051 thi hành chương trình trong ROM nội, PSEN sẽ ở mức logic 1.

❖ ALE/ PROG (Address Latch Enable / Program):

ALE/ PROG (chân 30) cho phép tách các đường địa chỉ và dữ liệu tại Port 0 khi truy xuất bộ nhớ ngoài. ALE thường nối với chân Clock của IC chốt (74373, 74573). Các xung tín hiệu ALE có tốc độ bằng 1/6 lần tần số dao động trên chip và có thể được dùng làm tín hiệu clock cho các phần khác của hệ thống. Xung này có thể cấm bằng cách set bit 0 của SFR tại địa chỉ 8Eh lên 1. Khi đó, ALE chỉ có tác dụng khi dùng lệnh MOVX hay MOVC. Ngoài ra, chân này còn được dùng làm ngõ vào xung lập trình cho ROM nội (/PROG**)**.

❖ EA /VPP (External Access) :

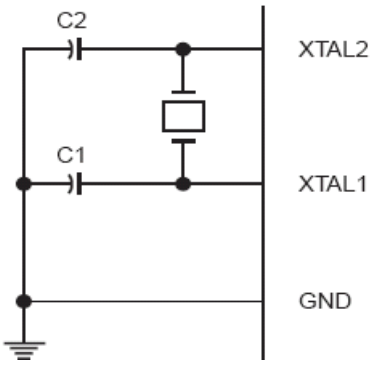
EA (chân 31) dùng để cho phép thực thi chương trình từ ROM ngoài. Khi nối chân 31 với Vcc, AT89C51 sẽ thực thi chương trình từ ROM nội (tối đa 8KB), ngược lại thì thực thi từ ROM ngoài (tối đa 64KB).
Ngoài ra, chân /EA được lấy làm chân cấp nguồn 12V khi lập trình cho ROM.

❖ RST (Reset):

RST (chân 9) cho phép reset AT89C51 khi ngõ vào tín hiệu đưa lên mức 1 trong ít nhất là 2 chu kỳ máy.

❖ X1, X2:

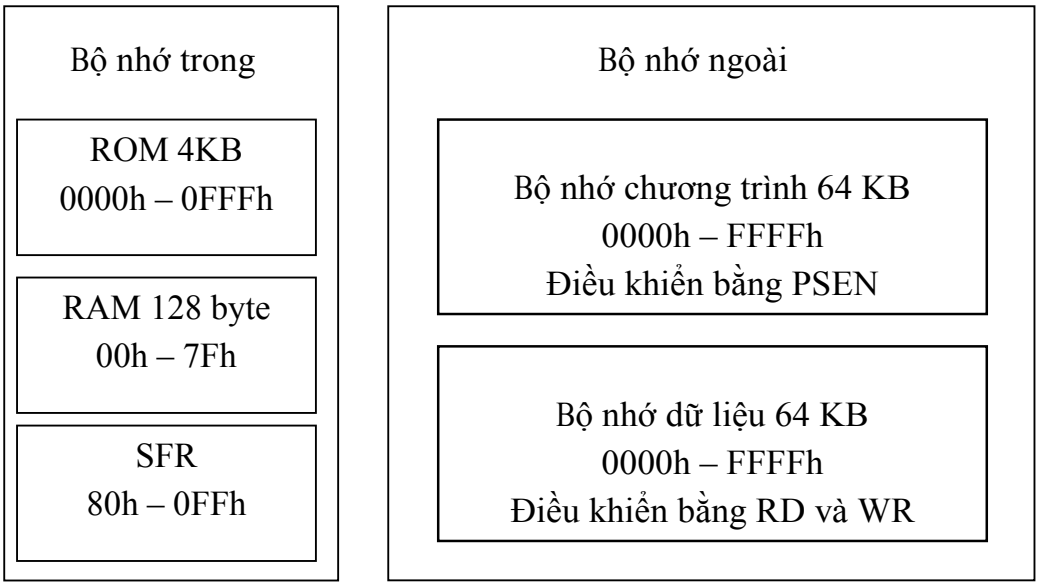
Ngõ vào và ngõ ra bộ dao động, khi sử dụng có thể chỉ cần kết nối thêm thạch anh và các tụ như hình vẽ trong sơ đồ. Tần số thạch anh thường sử dụng cho AT89C51 là 12Mhz.



Giá trị C_1, C_2 = 30 pF ± 10 pF

Hình 3-13 – Sơ đồ kết nối thạch anh

3.2.4 Tổ chức bộ nhớ 8051



Hình 3-14. Các vùng nhớ trong AT89C51

Bộ nhớ của họ MCS-51 có thể chia thành 2 phần: bộ nhớ trong và bộ nhớ ngoài. Bộ nhớ trong bao gồm 4 KB ROM và 128 byte RAM (256 byte trong 8052). Các byte RAM có địa chỉ từ 00h – 7Fh và các thanh ghi chức năng đặc biệt (SFR) có địa chỉ từ 80h – 0FFh có thể truy xuất trực tiếp. Đối với 8052, 128 byte RAM cao (địa chỉ từ 80h – 0FFh) không thể truy xuất trực tiếp mà chỉ có thể truy xuất gián tiếp (xem thêm trong phần tập lệnh).

Bộ nhớ ngoài bao gồm bộ nhớ chương trình (điều khiển đọc bằng tín hiệu PSEN) và bộ nhớ dữ liệu (điều khiển bằng tín hiệu RD hay WR để cho phép đọc hay ghi dữ liệu). Do số đường địa chỉ của MCS-51 là 16 bit (Port 0 chứa 8 bit thấp và Port 2 chứa 8 bit cao) nên bộ nhớ ngoài có thể giải mã tối đa là 64KB.

3.2.4.1 Tổ chức bộ nhớ trong

Bộ nhớ trong của MCS-51 gồm ROM và RAM. RAM bao gồm nhiều vùng có mục đích khác nhau: vùng RAM đa dụng (địa chỉ byte từ 30h – 7Fh và có thêm vùng 80h – 0FFh ứng với 8052), vùng có thể địa chỉ hóa từng bit (địa chỉ byte từ 20h – 2Fh, gồm 128 bit được định địa chỉ bit từ 00h – 7Fh), các bank thanh ghi (từ 00h – 1Fh) và các thanh ghi chức năng đặc biệt (từ 80h – 0FFh).

❖ Các thanh ghi chức năng đặc biệt (SFR – Special Function Registers):

Địa chỉ byte	**Có thể định địa chỉ bit**	**Không định địa chỉ bit**						
F8h								
F0h	B							
E8h								
E0h	ACC							
D8h								
D0h	PSW							
C8h	(T2CON)		(RCAP2L)	(RCAP2H)	(TL2)	(TH2)		
C0h								
B8h	IP	SADEN						
B0h	P3							
A8h	IE	SADDR						
A0h	P2							
98h	SCON	SBUF	BRL	BDRCON				
90h	P1							
88h	TCON	TMOD	TL0	TH0	TL1	TH1	AUXR	CKCON
80h	P0	SP	DPL	DPH				PCON

Bảng 3-2. Các thanh ghi chức năng đặc biệt

Các thanh ghi có thể định địa chỉ bit sẽ có địa chỉ bit bắt đầu và địa chỉ byte trùng nhau. Ví dụ như: thanh ghi P0 có địa chỉ byte là 80h và có địa chỉ bit bắt đầu từ 80h (ứng với P0.0) đến 87h (ứng với P0.7). Chức năng các thanh ghi này sẽ mô tả trong phần sau.

❖ RAM nội:

chia thành các vùng phân biệt: vùng RAM đa dụng (30h – 7Fh), vùng RAM có thể định địa chỉ bit (20h – 2Fh) và các bank thanh ghi (00h – 1Fh).

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

Bảng 3-3. Địa chỉ RAM nội 8051

❖ RAM đa dụng:

RAM đa dụng có 80 byte từ địa chỉ 30h – 7Fh có thể truy xuất mỗi lần 8 bit bằng cách dùng chế độ địa chỉ trực tiếp hay gián tiếp.

Các vùng địa chỉ thấp từ 00h – 2Fh cũng có thể sử dụng cho mục đich như trên ngoài các chức năng đề cập như phần sau.

❖ RAM có thể định địa chỉ bit:

Vùng địa chỉ từ 20h – 2Fh gồm 16 byte (= 128 bit) có thể thực hiện giống như vùng RAM đa dụng (mỗi lần 8 bit) hay thực hiện truy xuất mỗi lần 1 bit bằng các lệnh xử lý bit. Vùng RAM này có các địa chỉ bit bắt đầu tại giá trị 00h và kết thúc tại 7Fh.

Như vậy, địa chỉ bắt đầu 20h (gồm 8 bit) có địa chỉ bit từ 00h – 07h; địa chỉ kết thúc 2Fh có địa chỉ bit từ 78h – Fh.

❖ Các bank thanh ghi:

Vùng địa chỉ từ 00h – 1Fh được chia thành 4 bank thanh ghi: bank 0 từ 00h-07h, bank 1 từ 08h – 0Fh, bank 2 từ 10h – 17h và bank 3 từ 18h – 1Fh. Các bank thanh ghi này được đại diện bằng các thanh ghi từ R0 đến R7. Sau khi khởi động hệ thống thì bank thanh ghi được sử dụng là bank 0.

Do có 4 bank thanh ghi nên tại một thời điểm chỉ có một bank thanh ghi được truy xuất bởi các thanh ghi R0 đến R7. Việc thay đổi bank thanh ghi có thể thực hiện thông qua thanh ghi từ trạng thái chương trình (PSW). Các bank thanh ghi này cũng có thể truy xuất bình thường như vùng RAM đa dụng đã nói ở trên.

3.2.4.2 Tổ chức bộ nhớ ngoài

MCS-51 có bộ nhớ theo cấu trúc Harvard: phân biệt bộ nhớ chương trình và dữ liệu. Chương trình và dữ liệu có thể chứa bên trong nhưng vẫn có thể kết nối với 64KB chương trình và 64KB dữ liệu. Bộ nhớ chương trình được truy xuất thông qua chân PSEN còn bộ nhớ dữ liệu đươc truy xuất thông qua chân WR hay RD.

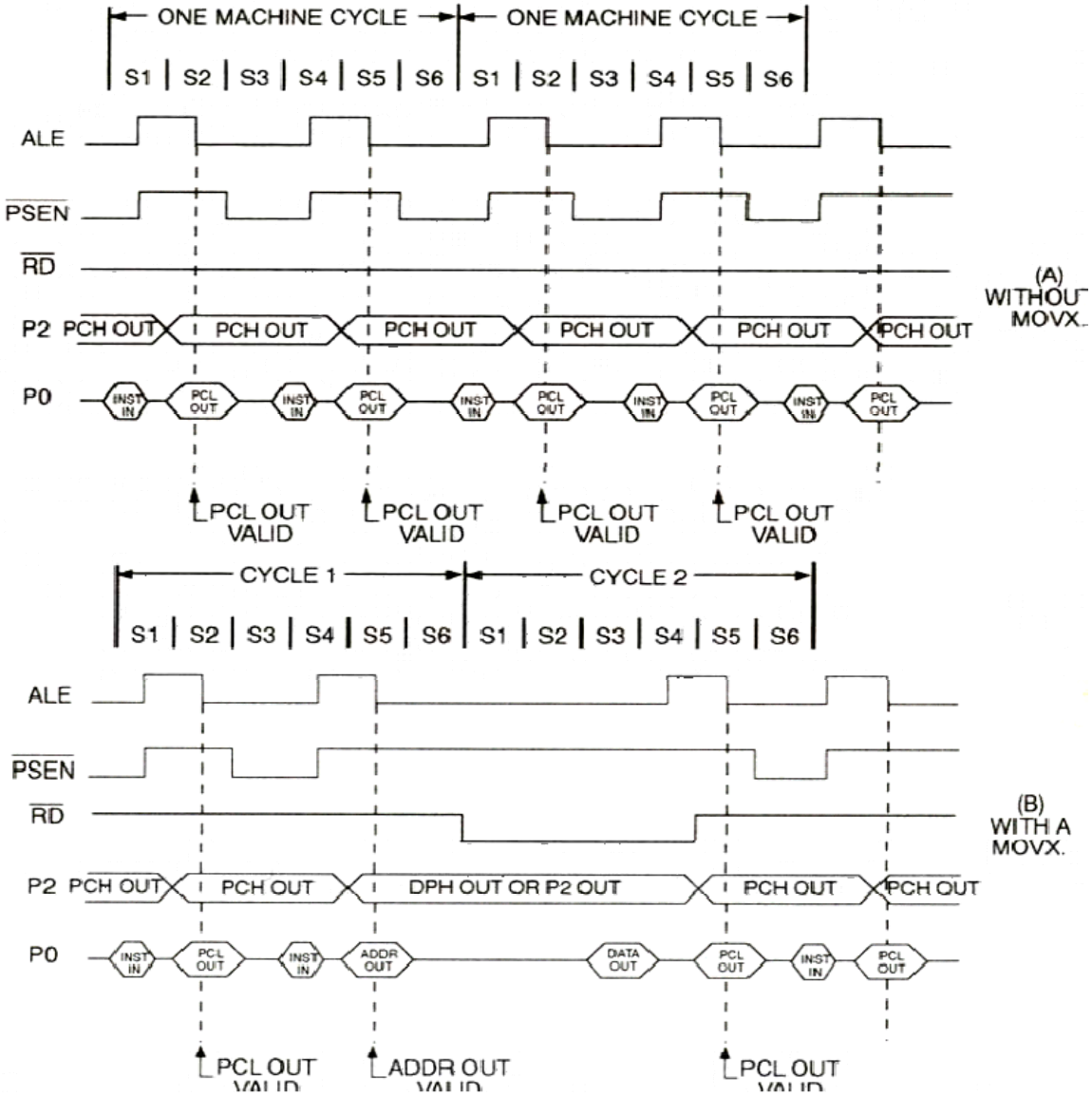
Lưu ý rằng việc truy xuất bộ nhớ chương trình luôn luôn sử dụng địa chỉ 16 bit còn bộ nhớ dữ liệu có thể là 8 bit hay 16 bit tuỳ theo câu lệnh sử dụng. Khi dùng bộ nhớ dữ liệu 8 bit thì có thể dùng Port 2 như là Port I/O thông thường còn khi dùng ở chế độ 16 bit thì Port 2 chỉ dùng làm các bit địa chỉ cao.

Port 0 được dùng làm địa chỉ thấp/ dữ liệu đa hợp. Tín hiệu /ALE để tách byte địa chỉ và đưa vào bộ chốt ngoài.

Trong chu kỳ ghi, byte dữ liệu sẽ tồn tại ở Port 0 vừa trước khi /WR tích cực và được giữ cho đến khi /WR không tích cực.Trong chu kỳ đọc, byte nhận được chấp nhận vừa trước khi /RD không tích cực.

Bộ nhớ chương trình ngoài được xử lý 1 trong 2 điều kiện sau:

- Tín hiệu /EA tích cực (= 0).
- Giá trị của bộ đếm chương trình (PC – Program Counter) lớn hơn kích thước bộ nhớ.



PCH: Program Counter High – PCL: Program Counter Low
DPH: Data Pointer High – DPL: Data Pointer Low

Hình 3-15. Thực thi bộ nhớ chương trình ngoài

❖ Bộ nhớ chương trình ngoài:

Quá trình thực thi lệnh khi dùng bộ nhớ chương trình ngoài có thể mô tả như "Hình 3-15. Thực thi bộ nhớ chương trình ngoài". Trong quá trình này, Port 0 và Port 2 không còn là các Port xuất nhập mà chứa địa chỉ và dữ liệu. Sơ đồ kết nối với bộ nhớ chương trình ngoài mô tả như "Hình 3-14. Các vùng nhớ trong AT89C51".

Trong một chu kỳ máy, tín hiệu ALE tích cực 2 lần. Lần thứ nhất cho phép 74HC573 mở cổng chốt địa chỉ byte thấp, khi /ALE xuống 0 thì byte thấp và byte cao của bộ đếm chương trình đều có nhưng ROM chưa xuất vì PSEN chưa tích cực, khi tín hiệu ALE lên 1 trở lại thì Port 0 đã có dữ liệu là mã lệnh. ALE tích cực lần thứ hai được giải thích tương tự và byte 2 được đọc từ bộ nhớ chương trình. Nếu lệnh đang thực thi là lệnh 1 byte thì CPU chỉ đọc Opcode, còn byte thứ hai bỏ qua.

❖ Bộ nhớ dữ liệu ngoài:

Bộ nhớ dữ liệu ngoài được truy xuất bằng lệnh MOVX thông qua các thanh ghi xác định địa chỉ DPTR (16 bit) hay R0, R1 (8 bit).

Quá trình thực hiện đọc hay ghi dữ liệu được cho phép bằng tín hiệu RD hay WR (chân P3.7 và P3.6).

❖ Bộ nhớ chương trình và dữ liệu dùng chung:

Trong các ứng dụng phát triển phần mềm xây dựng dựa trên AT89C51, ROM sẽ được lập trình nhiều lần nên dễ làm hư hỏng ROM. Một giải pháp đặt ra là sử dụng RAM để chứa các chương trình tạm thời. Khi đó, RAM vừa là bộ nhớ chương trình vừa là bộ nhớ dữ liệu. Yêu cầu này có thể thực hiện bằng cách kết hợp chân RD và chân PSEN thông qua cổng AND. Khi thực hiện đọc mà lệnh, chân /PSEN tích cực cho phép đọc từ RAM và khi đọc dữ liệu, chân RD sẽ tích cực.

❖ Giải mã địa chỉ

Trong các ứng dụng dựa trên AT89C51, ngoài giao tiếp bộ nhớ dữ liệu, vi điều khiển còn thực hiện giao tiếp với các thiết bị khác như bàn phím, led, động cơ, … Các thiết bị này có thể giao tiếp trực tiếp thông qua các Port. Tuy nhiên, khi số lượng các thiết bị lớn, các Port sẽ không đủ để thực hiện điều khiển. Giải pháp đưa ra là xem các thiết bị này giống như bộ nhớ dữ liệu. Khi đó, cần phải thực hiện quá trình giải mã địa chỉ để phân biệt các thiết bị ngoại vi khác nhau. Quá trình giải mã địa chỉ thường được thực hiện thông qua các IC giải mã như 74139 (2 -> 4), 74138 (3 -> 8), 74154 (4 -> 16). Ngõ ra của các IC giải mã sẽ được đưa tới chân chọn chip của RAM hay bộ đệm khi điều khiển ngoại vi.

3.2.5 Các thanh ghi chức năng đặc biệt (SFRs - Special Function Registers)

❖ Thanh ghi tích luỹ (Accumulator)

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

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

VD: Câu lệnh:

```
MOV   A,#1
MOV   0E0h,#1
```

có cùng kết quả. Hay:

```
SETB ACC.4
SETB 0E4h
```

cũng tương tự.

❖ Thanh ghi B

Thanh ghi B dùng cho các phép toán nhân, chia và có thể dùng như một

thanh ghi tạm, chứa các kết quả trung gian.

Thanh ghi B có địa chỉ byte F0h và địa chỉ bit từ F0h – F7h có thể truy xuất giống như thanh ghi A.

❖ Thanh ghi từ trạng thái chương trình (PSW - Program Status Word)

Thanh ghi từ trạng thái chương trình PSW nằm tại địa chỉ D0h và có các địa chỉ bit từ D0h – D7h, bao gồm 7 bit (1 bit không sử dụng) có các chức năng như sau:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Chức năng	CY	AC	F0	RS1	RS0	OV	F1	P

- CY (Carry): cờ nhớ, thường được dùng cho các lệnh toán học không dấu (C = 1 khi có nhớ trong phép cộng hay mượn trong phép trừ)
- AC (Auxiliary Carry): cờ nhớ phụ (thường dùng cho các phép toán BCD).
- F0 (Flag 0): được sử dụng tuỳ theo yêu cầu của người sử dụng.
- RS1, RS0: dùng để chọn bank thanh ghi sử dụng. Khi reset hệ thống, bank 0 sẽ được sử dụng.

RS1	RS0	Bank thanh ghi
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

- OV (Overflow): cờ tràn. Cờ OV = 1 khi có hiện tượng tràn số học xảy ra (dùng cho số nguyên có dấu).
- F1 (Flag 1): được sử dụng tuỳ theo yêu cầu của người sử dụng.
- P (Parity): kiểm tra parity (lẻ). Cờ P = 1 khi tổng số bit 1 trong thanh ghi A là số lẻ (nghĩa là tổng số bit 1 của thanh ghi A cộng thêm cờ P là số chẵn). Ví dụ như: A = 10101010b có tổng cộng 4 bit 1 nên P = 0. Cờ P thường được dùng để kiểm tra lỗi truyền dữ liệu.

❖ Thanh ghi con trỏ stack (SP – Stack Pointer)

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

Đối với AT89C51, stack được chứa trong RAM nội (128 byte đối với 8031/8051 hay 256 byte đối với 8032/8052). Mặc định khi khởi động, giá trị của SP là 07h, nghĩa là stack bắt đầu từ địa chỉ 08h (do hoạt động lưu giá trị vào stack yêu cầu phải tăng nội dung thanh ghi SP trước khi lưu). Như vậy, nếu không gán giá trị cho thanh ghi SP thì không được sử dụng các bank thanh ghi 1, 2, 3 vì có thể làm sai dữ liệu. Đối với các ứng dụng thông thường không cần dùng nhiều đến stack, có thể

không cần khởi động SP mà dùng giá trị mặc định là 07h. Tuy nhiên, nếu cần, ta có thể xác định lại vùng stack cho MCS-51.

❖ Con trỏ dữ liệu DPTR (Data Pointer)

Con trỏ dữ liệu DPTR là thanh ghi 16 bit bao gồm 2 thanh ghi 8 bit: DPH (High) nằm tại địa chỉ 83h và DPL (Low) nằm tại địa chỉ 82h. Các thanh ghi này không cho phép định địa chỉ bit. DPTR được dùng khi truy xuất đến bộ nhớ có địa chỉ 16 bit.

❖ Các thanh ghi port

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

Ví dụ như: 2 lệnh sau là tương đương:

```
SETB P0.0
SETB 80h
```

❖ Thanh ghi port nối tiếp (SBUF - Serial Data Buffer)

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

❖ Các thanh ghi định thời (Timer Register)

Các cặp thanh ghi (TH0, TL0), (TH1, TL1) và (TH2, TL2) là các thanh ghi dùng cho các bộ định thời 0, 1 và 2 trong đó bộ định thời 2 chỉ có trong 8032/8052. Ngoài ra, đối với họ 8032/8052 còn có thêm cặp thanh ghi (RCAP2L, RCAP2H) sử dụng cho bộ định thời 2 (sẽ thảo luận trong phần hoạt động định thời).

❖ Các thanh ghi điều khiển

Bao gồm các thanh ghi IP (Interrupt Priority), IE (Interrupt Enable), TMOD (Timer Mode), TCON (Timer Control), T2CON (Timer 2 Control), SCON (Serial port control) và PCON (Power control).

- Thanh ghi IP tại địa chỉ B8h cho phép chọn mức ưu tiên ngắt khi có 2 ngắt xảy ra đông thời. IP cho phép định địa chỉ bit từ B8h – BFh.
- Thanh ghi IE tại địa chỉ A8h cho phép hay cấm các ngắt. IE có địa chỉ bit từ A8h – AFh.
- Thanh ghi TMOD tại địa chỉ 89h dùng để chọn chế độ hoạt động cho các bộ định thời (0, 1) và không cho phép định địa chỉ bit.

- Thanh ghi TCON tại địa chỉ 88h điều khiển hoạt động của bộ định thời và ngắt. TCON có địa chỉ bit từ 88h – 8Fh.
- Thanh ghi T2CON tại địa chỉ C8h điều khiển hoạt động của bộ định thời 2. T2CON có địa chỉ bit từ C8h – CFh.
- Thanh ghi SCON tại địa chỉ 98h điều khiển hoạt động của port nối tiếp. SCON có địa chỉ bit từ 98h – 9Fh.

Các thanh ghi đã nói ở trên sẽ được thảo luận thêm ở các phần sau.

❖ Thanh ghi điều khiển nguồn PCON

Thanh ghi PCON tại địa chỉ 87h không cho phép định địa chỉ bit bao gồm các bit như sau:

Bit	7	6	5	4	3	2	1	0
Chức năng	SMOD1	SMOD0	-	POF	GF1	GF0	PD	IDL

SMOD1 (Serial Mode 1): = 1 cho phép tăng gấp đôi tốc độ port nối tiếp trong chế độ 1, 2 và 3.

SMOD0 (Serial Mode 0): cho phép chọn bit SM0 hay FE trong thanh ghi SCON (= 1 chọn bit FE).

POF (Power-off Flag): dùng để nhận dạng loại reset. POF = 1 khi mở nguồn. Do đó, để xác định loại reset, cần phải xoá bit POF trước đó.

GF1, GF0 (General purpose Flag): các bit cờ dành cho người sử dụng.

PD (Power Down): được xoá bằng phần cứng khi hoạt động reset xảy ra. Khi bit PD = 1 thì vi điều khiển sẽ chuyển sang chế độ nguồn giảm. Trong chế độ này:

- Chỉ có thể thoát khỏi chế độ nguồn giảm bằng cách reset.
- Nội dung RAM và mức logic trên các port được duy trì.
- Mạch dao động bên trong và các chức năng khác ngừng hoạt động.
- Chân ALE và PSEN ở mức thấp.
- Yêu cầu Vcc phải có điện áp ít nhất là 2V và phục hồi Vcc = 5V ít nhất 10 chu kỳ trước khi chân RESET xuống mức thấp lần nữa.

IDL (Idle): được xoá bằng phần cứng khi hoạt động reset hay có ngắt xảy ra. Khi bit IDL = 1 thì vi điều khiển sẽ chuyển sang chế độ nghỉ. Trong chế độ này:

- Chỉ có thể thoát khỏi chế độ nguồn giảm bằng cách reset hay có ngắt xảy ra.
- Trạng thái hiện hành của vi điều khiển được duy trì và nội dung các thanh ghi không đổi.
- Mạch dao động bên trong không gởi được tín hiệu đến CPU.
- Chân ALE và PSEN ở mức cao.

Lưu ý rằng các bit điều khiển PD và IDL có tác dụng chính trong tất cả các IC họ MSC-51 nhưng chỉ có thể thực hiện được trong các phiên bản CMOS.

3.2.6 Bộ đếm và bộ định thời

Định thời là sự hoạt động để kiểm soát thời gian thực thi các câu lệnh trong quá trình xử lý của vi điều khiển.

8051 có hai bộ định thời/ bộ đếm. Chúng có thể được dùng như các bộ định thời để tạo một bộ trễ thời gian hoặc như các bộ đếm để đếm các sự kiện xảy ra bên ngoài bộ VĐK. Các timer này đều là timer 16bit, giá trị đếm được tính từ 0 đến 216 (đếm từ 0 đến 65535).

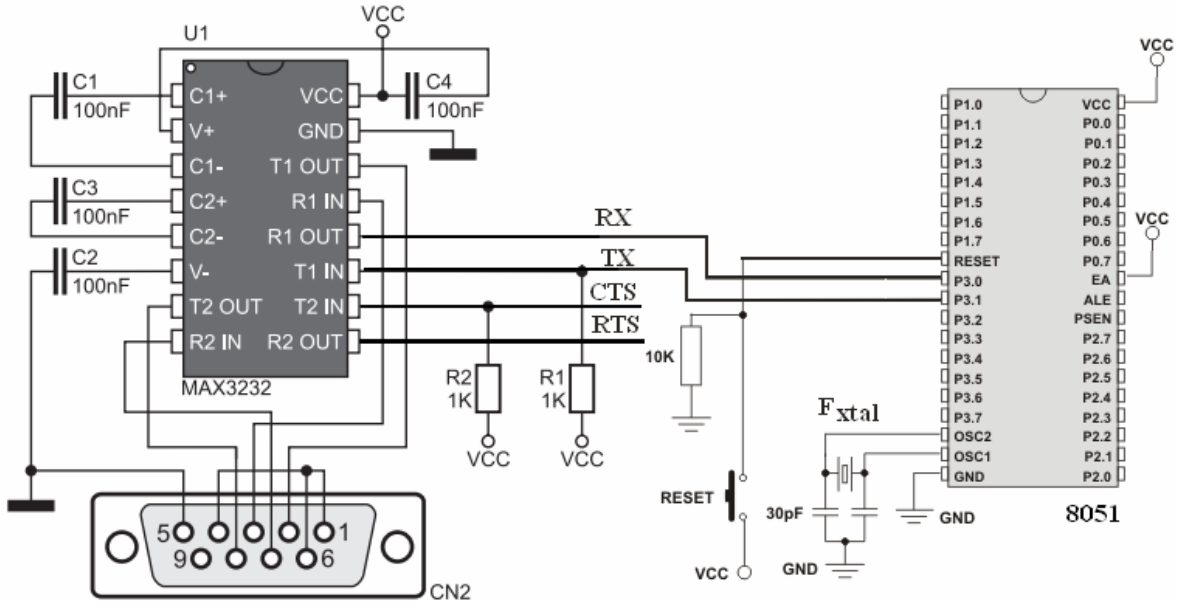
Hai timer có nguyên lý hoạt động hoàn toàn giống nhau và độc lập. Sau khi cho phép chạy, mỗi khi có thêm một xung tại đầu vào đếm, giá trị của timer sẽ tự động được tăng lên 1 đơn vị, cứ như vậy cho đến khi giá trị tăng lên vượt quá giá trị 65535 mà thanh ghi đếm có thể biểu diễn thì giá trị đếm lại được đưa trở về giá trị 0

Việc cho timer chạy/dừng được thực hiện bởi các bit TR trong thanh ghi TCON (đánh địa chỉ đến từng bit).

Các timer có thể hoạt động theo nhiều chế độ, được quy định bởi các bit trong thanh ghi TMOD.

3.2.7 Truyền thông không đồng bộ (UART)

8051 có 1 cổng UART làm việc ở chuẩn TTL, mặc định sau khi khởi động tất các cổng của 8051 dều làm việc ở chế độ vào ra số, vì thế để có thể sử dụng UART cần phải cấu hình cho cổng này làm việc thông qua các thanh ghi điều khiển và ghép nối tương thích với chuẩn rs232.



Hình 3-16 - Ghép nối RS232 với 8051

Cổng nối tiếp trong 8051 chủ yếu được dùng trong các ứng dụng có yêu cầu truyền thông với máy tính, hoặc với một vi điều khiển khác. Liên quan đến cổng nối

tiếp chủ yếu có 2 thanh ghi: SCON và SBUF. Ngoài ra, một thanh ghi khác là thanh ghi PCON (không đánh địa chỉ bit) có bit 7 tên là SMOD quy định tốc độ truyền của cổng nối tiếp có gấp đôi lên (SMOD = 1) hay không (SMOD = 0).

Dữ liệu được truyền nhận nối tiếp thông qua hai chân cổng P3.0(RxD) và P3.1(TxD).

3.2.8 Ngắt vi điều khiển 8051

8051 hỗ trợ 5 loại ngắt, mỗi ngắt có một vector ngắt riêng, đó là một địa chỉ cố định nằm trong bộ nhớ chương trình. Khi xảy ra ngắt CPU sẽ tự động nhảy đến thực hiện lệnh thuộc địa chỉ này.

Liên quan đến ngắt chủ yếu có hai thanh ghi là thanh ghi IE và thanh ghi IP.

Thanh ghi IE là thanh ghi đánh địa chỉ bit, do đó có thể dùng các lệnh tác động bit để tác động riêng rẽ lên từng bit mà không làm ảnh hưởng đến giá trị các bit khác. Để cho phép một ngắt, bit tương ứng với ngắt đó và bit EA phải được đặt bằng 1.

3.3 Lập trình hợp ngữ cho 8051

Lập trình cho vi điều khiển cũng tương tự như lập trình cho máy tính, bản chất là ta 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. Và tất cả những lệnh mà vi điều khiển có thể hiểu được gọ là tập lệnh. Các vi điều khiển tương thích với 8051 có 255 lệnh.

3.3.1 Các chế độ địa chỉ

a) Địa chỉ tức thời

Trong chế độ đánh địa chỉ này toán hạng nguồn là một hằng số. Và như tên gọi của nó thì khi một lệnh được hợp dịch toán hạng đi tức thi ngay sau mã lệnh. Lưu ý rằng trước dữ liệu tức thời phải được đặt dấu (#) chế độ đánh địa chỉ này có thể được dùng để nạp thông tin vào bất kỳ thanh ghi nào kể cả thanh ghi con trỏ dữ liệu DPTR. Ví dụ:

```
MOV   A, # 25H   ; Nạp giá trị 25H vào thanh ghi A
MOV   R4, #62    ; Nạp giá trị  62 thập phân vào R4
MOV   DPTR, #4521H     ; Nạp 4512H vào con trỏ dữ liệu DPTR
```

b) Địa chỉ theo thanh ghi

Chế độ đánh địa chỉ theo thanh ghi liên quan đến việc sử dụng các thanh ghi để lưu dữ liệu cần được thao tác và các các toán hạng là 1 trong các thanh ghi Ri của các bank được chọn. Ví dụ :

```
MOV   A,  R0      ; Sao nội dung thanh ghi R0 vào thanh ghi A
MOV   R2, A       ; Sao nội  dung thanh ghi  A vào thanh ghi R2
```

c) Địa chỉ trực tiếp

Bộ nhớ RAM được gán các địa chỉ từ 00 đến FFH và được phân chia như sau:

1. Các ngăn nhớ từ 00 đến 1FH được gán cho các băng thanh ghi và ngăn xếp.
2. Các ngăn nhớ từ 20H đến 2FH được dành cho không gian đánh địa chỉ theo bit để lưu các dữ liệu 1 bit.
3. Các ngăn nhớ từ 30H đến 7FH là không gian để lưu dữ liệu có kích thước 1byte.

Toán hạng là tên hoặc địa chỉ của các thanh ghi trong vùng RAM thấp (0-127) và vùng chứa các thanh ghi chức năng đặc biệt SFR.

Ví dụ :

```
MOV   R0, 40H; Lưu nội dung của ngăn nhớ 40H của RAM vào R0
MOV   56H, A; Lưu nội dung thanh ghi A vào ngăn nhớ 56H của RAM
```

Các ngăn nhớ dành cho băng ghi được truy cập bằng thanh ghi theo các tên gọi của chúng là R0 - R7. Nên các thanh ghi có thể được truy cập theo hai cách sau:

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

```
MOV   A, 4
MOV   A, R4
```

d) Địa chỉ gián tiếp

Trong chế độ này, một thanh ghi được sử dụng như một con trỏ đến dữ liệu. Toán hạng có thể nằm trong cả vùng RAM thấp và cao, hoặc RAM ngoài, không dùng cho vùng SFR. Địa chỉ của toán hạng chứa trong thanh ghi con trỏ (R0 hoặc R1 với RAM trong, DPTR đối với RAM ngoài). Đặc điểm nhân ra chế độ này là luôn có ký tự @ đứng trước toán hạng.

Ví dụ:

```
MOV   A, @ R0     ; Chuyển nội dung của ngăn nhớ RAM có
                  ;địa chỉ trong RO và A
```

e) Địa chỉ chỉ số

Chế độ đánh địa chỉ theo chỉ số được sử dụng rộng rãi trongviệc truy cập các phân tử dữ liệu của bảng trong không gian ROM/RAM chương trình của 8051 trong dải 64KB.

Lệnh được dùng cho mục đích này là

```
"MovC A, @ A + DPTR" và
"MovX A, @ A + DPTR".
```

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

Thay lệnh ***Mov*** bằng ***MovC/MovX*** do các phần tử dữ liệu được cất trong không gian mã (chương trình) của Flash ROM trong/ngoài chip của 8051. Trong lệnh này

thì nội dung của A được bổ xung vào thanh ghi 16 bit DPTR để tạo ra địa chỉ 16 bit của dữ liệu cần thiết

3.3.2 Tập lệnh trong 8051

❖ Phân loại tập lệnh

Tùy thuộc vào cách và chức năng của mỗi lệnh, có thể chia ra thành 5 nhóm lệnh như sau:

- Các lệnh toán học
- Các lệnh điều khiển chương trình
- Các lệnh vận chuyển dữ liệu
- Các lệnh logic
- Các lệnh thao tác bit

Cấu trúc chung của mỗi lệnh:

```
Mã_lệnh Toán_hạng1, Toán_hạng2, Toán_hạng3
```

Trong đó:

- Mã_lệnh: Tên gợi nhớ cho chức năng của lệnh. (VD như add cho addition)
- Toán_hạng1, Toán_hạng2, Toán_hạng3: Là các toán hạnh của lệnh, tùy thuộc vào mỗi lệnh số toán hạng có thể không có, có 1, 2 hoặc 3.

VD:

- RET (Kết thúc chương trình con) Lệnh này không có toán hạng
- JZ TEMP (Chuyển con trỏ chương trình đến vị trí TEMP) Chỉ có 1 toán hạng
- ADD A, R3; (A = A + R3) Có 2 toán hạng
- CJNE A, #20, LOOP (So sánh A với 20, nếu không bằng thì chuyển con trỏ chương trình đến nhã LOOP) Có 3 toán hạng

❖ Các ký hiệu sử dụng mô tả lệnh

Ký hiệu	Mổ tả
A:	Thanh ghi chứa (Accumulator).
B:	Thanh ghi B.
Ri:	Thanh ghi R0 hoặc R1 của bất kỳ băng thanh ghi nào trong 4 băng thanh ghi trong RAM.
Rn:	Rn: bất kỳ thanh ghi nào của bất kỳ băng thanh ghi nào trong 4 băng thanh ghi trong RAM.

Ký hiệu	Mổ tả
Dptr:	thanh ghi con trỏ dữ liệu (có độ rộng 16bit được kết hợp từ 2 thanh ghi 8 bit là DPH và DPL).
Direct:	Direct: là một biến 8 bit(hay chính là ô nhớ) bất kỳ trong RAM (trừ 32 thanh ghi Rn ở đầu RAM).
#data:	một hằng số 8 bit bất kỳ.
#data16:	một hằng số 16 bit bất kỳ
<rel>:	địa chỉ bất kỳ nằm trong khoảng [PC-128 ; PC+127]
<addr11>:	địa chỉ bất kỳ nằm trong khoảng 0 – 2Kbyte tính từ địa chỉ của lệnh tiếp theo.
<addr16>:	địa chỉ bất kỳ trong không gian 64K (áp dụng cho cả không gian nhớ chương trình và không gian nhớ dữ liệu).
<bit>:	bit bất kỳ có thể đánh địa chỉ được (không dùng cho các bit không đánh được địa chỉ).

Bảng 3-4. ký hiệu sử dụng mô tả lệnh

❖ Các lệnh toán học

Các ký hiệu dùng trong việc mô tả tập lệnh

Thực hiện các phép tính cơ bản như +, -, *, /, … Kết quả sau khi thực hiện lệnh được lưu vào toán hạng đầu tiên trong lệnh

Các lệnh toán học như: ADD, ADDC, SUBB, INC, DEC, MUL, DA. Ví dụ1 :

```
MOV   A, # 0F5H  ; A = F5H
MOV   A, # 0BH   ; A = F5 + 0B = 00
```

Sau phép cộng, thanh ghi A (đích) chứa 00 và các cờ sẽ như sau:

CY = 1 vì có phép nhớ từ D7

PF = 1 vì số các số 1 là 0 (một số chẵn) cờ PF được đặt lên 1.

AC = 1 vì có phép nhớ từ D3 sang D4. Ví dụ 2:

```
MOV   A, #47H          ; A = 47H là toán hạng BCD đầu tiên
MOV   B, #25H          ; B = 25H  là toán hạng BCD thứ hai
ADD   A, B             ; Cộng các số hex (nhị phân) A = 6CH
DA    A                ; Điều chỉnh cho phép cộng BCD (A = 72H)
```

Sau khi chương trình được thực hiện thanh ghi A sẽ chứa 72h (47 + 25 = 72).

Ví dụ 3: thực hiện phép nhân

```
MOV   A, #25H          ; Nạp vào A giá trị 25H
MOV   B, 65H           ; Nạp vào B giá trị 65H
MUL   AB               ; 25H*65H = E99 với B = 0EH và A = 99H
```

Các lệnh số học xem chi tiết trong phần phụ lục

❖ Các lệnh logic

Thực hiện các phép toán logic, các lệnh bao gồm:

ANL: phép toán "and " logic

ORL: phép toán "or " logic

XRL: phép toán "xor " logic

CLR: phép toán "và " logic

CPL: phép toán bù

RL: phép quay bit sang trái

RR: phép quay bit sang phải

RLC: : phép quay trái có nhớ

RRC: phép quay phải có nhớ

SWAP: lệnh trao đổi thanh ghi

Ví dụ 1:

```
MOV   A, #35H          ; Gán A = 35H
ANL   A, #0FH          ; Thực hiện phép "và"  A với 0FH
```

Kết quả: A=05h

Ví dụ 2:

```
MOV   A, #04           ; A = 04
ORL   A, #68H    ; A = 6C
```

Ví dụ 3:

```
MOV   A, #54H    ; A= 54H
XRL   A, #78H    ; A=2CH
```

Ví dụ 4:

```
MOV        A, #55H
CPL        A           ;kết qủa thanh ghi A là AAH
```

Ví dụ 5: các lệnh quay

```
RR:   MOV   A, #36H          ; A = 0011 0110
      RR    A                ; A = 0001 1011
      RR    A                ; A = 1000 1101
      RR    A                ; A = 1100 0110
      RR    A                ; A = 0110 0011
RRC: MOV   A #26H           ; A = 0010 0110
      RRC   A                ; A = 0001 0011  CY = 0
      RRC   A                ; A = 0000 1001  CY = 1
      RCC   A                ; A = 1000 0100  CY = 1
```

Ví dụ 6:

```
MOV        A, #72H    ; A = 72H
SWAP       A          ; A = 27H
```

Các lệnh số học xem chi tiết trong phần phụ lục

❖ Các lệnh vận chuyển dữ liệu

Di chuyển dữ liệu từ ô nhớ này đến ô nhứ khác, hoặc giữa hai thanh ghi, thanh ghi ô nhớ.

Các lệnh vận chuyển dữ liệu bao gồm:
MOV: chuyển dữ liệu giữa thanh ghi với thanh ghi, thanh ghi với ô nhớ, một hằng số đến thanh ghi, một hằng số đến ô nhớ, và ngược lại
MOVC: Sao chép mã nguồn (dữ liệu đã được đặt trong vùng mã nguồn)

STT	Cú pháp		Mô tả	Số byte	Số chu kỳ
	Mã lệnh	Toán hạng			
1	MOV	A,Rn	Copy giá trị của toán hạng bên phải cho vào toán hạng bên trái (các toán hạng đều là 8bit)	1	1
2	MOV	A,direct		2	1
3	MOV	A,@Ri		1	1
4	MOV	A,#data		2	1
5	MOV	Rn,A		1	1
6	MOV	Rn,direct		2	2
7	MOV	Rn,#data		2	1
8	MOV	Direct,A		2	1
9	MOV	Direct,Rn		2	2
10	MOV	Direct,direct		3	2
11	MOV	Direct,@Ri		2	2
12	MOV	Direct,#data		3	2
13	MOV	@Ri,A		1	1
14	MOV	@Ri,direct		2	1
15	MOV	@Ri,#data		2	1
16	MOV	Dptr,#data16	Đưa giá trị 16bit vào thanh ghi DPTR	3	2
17	MOVC	A,@A+dptr	Đọc giá trị bộ nhớ chương trình tại địa chỉ = A + DPTR, cất kết quả vào A	1	2
18	MOVC	A,@A+PC	Đọc giá trị bộ nhớ chương trình tại địa chỉ = A + PC, cất kết quả vào A	1	2
19	MOVX	A,@Ri	Đọc vào A giá trị của bộ nhớ ngoài tại địa chỉ = Ri	1	2
20	MOVX	A,@dptr	Đọc vào A giá trị của bộ nhớ ngoài tại địa chỉ = DPTR	1	2
21	MOVX	@dptr,A	Ghi giá trị của A vào bộ nhớ ngoài tại địa chỉ = DPTR	1	2

STT	Cú pháp		Mô tả	Số byte	Số chu kỳ
	Mã lệnh	Toán hạng			
22	MOVX	@dptr,A	Ghi giá trị của A vào bộ nhớ ngoài tại địa chỉ = DPTR	2	2
23	PUSH	Direct	Cất nội dung của biến trong RAM vào đỉnh ngăn xếp	2	2
24	POP	Direct	Lấy byte ở đỉnh ngăn xếp cho vào biến trong RAM	2	2
25	XCH	A,Rn	Hoán đổi giá trị của A và giá trị còn lại	1	1
26	XCH	A,direct		2	1
27	XCH	A,@Ri		1	1
28	XCHD	A,@Ri	Hoán đổi 4 bit thấp giữaA và một ô nhớ trong Ram tại địa chỉ = Ri	1	1

Bảng 3-5. Các lệnh vận chuyển dữ liệu

❖ Các lệnh thao tác bit và đọc cổng: ***Các lệnh thao tác bit***:

Lệnh	Chức năng
SETB bit	Thiết lập bit (bit bằng 1)
CLR bit	Xoá bit về không (bit = 0)
CPL bit	Bù bit (bit = NOT bit)
JB bit, đích	Nhảy về đích nếu bit = 1
JNB bit, đích	Nhảy về đích nếu bit = 0
JBC bit, đích	Nhảy về đích nếu bit = 1 và sau đó xoá bit
Lệnh	chức năng
SETB C	Thực hiện (tạo) CY = 1
CLR C	Xoá bit nhớ CY = 0
CPL C	Bù bit nhớ
MOV b, C	Sao chép trạng thái bit nhớ vào vị trí bit b = CY
MOV C, b	Sao chép bit b vào trạng thái bit nhớ CY = b
JNC đích	Nhảy tới đích nếu CY = 0
JC đích	Nhảy tới đích nếu CY = 1
ANL C. bit	Thực hiện phép AND với bit b và lưu vào CY
ANL C./ bit	Thực hiện phép AND với bit đảo và lưu vào CY
ORL C. bit	Thực hiện phép OR với bit và lưu vào CY
ORL C./ bit	Thực hiện phép OR với bit đảo và lưu vào CY

Bảng 3-6. Các lệnh thao tác bit và đọc cổng

Các lệnh thao tác bit xem chi tiết trong phần phụ lục

Ví dụ: viết chương trình để lưu các bit P1.2 vào vị trí bit 06 và trạng thái P1.3 vào vị trí bit 07

```
      CLR   06          ;Xoá địa chỉ bit 06
      CLR   07          ; Xoá địa chỉ bit 07
      JNB   P1.2, OVER ;Kiểm tra bit P1.2 nhảy về OVER nếu P1.2 = 0
      SETB  06          ; Nếu P1.2 thì thiết lập vị trí bit 06 = 0
OVER: JNB   P1.3, NEXT ;Kiểm tra bit P1.3 nhảy về NEXT nếu nó = 0
      SETB  07          ;Nếu P1.3 = 1thì thiết lập vị trí bit 07 = 1
NEXT: ....
```

Lệnh đọc cổng

Trong việc đọc cổng thì một số lệnh đọc trạng thái của các chân cổng, còn một số lệnh khác thì đọc một số trạng thái của chốt cổng trong. Do vậy, khi đọc các cổng thì có hai khả năng:

1. Đọc trạng thái của cổng vào.

Lệnh	Ví dụ	Mô tả
MOV A, PX	MOV A, P2	Chuyển dữ liêuj ở chân P2 vào ACC
JNB PX.Y, ...	JNB P2.1, đích	Nhảy tới đích nếu, chân P2.1 = 0
JB PX.Y,	JB P1.3, đích	Nhảy đích nếu, chân P1.3 = 1
MOV C, PX.Y	MOV C, P2.4	Sao trạng thái chân P2.4 vào CY

Bảng 3-7. Lệnh đọc cổng

2. Đọc chốt trong của cổng ra.

Lệnh	Ví dụ
ANL PX	ANL P1, A
ORL PX	ORL P2, A
XRL PX	XRL P0, A
JBC PX.Y, đích	JBC P1.1, đích
CPL PX	CPL P1.2
INC PX	INC P1
DEC PX	DEC P2
DJN2 PX.Y, đích	DJN2 P1, đích
MOV PX.Y, C	MOV P1.2, C
CLR PX.Y	CLR P2.3
SETB PX.Y	SETB P2.3

Bảng 3-8. Đọc chốt trong của cổng ra

❖ Các lệnh điều khiển chương trình (rẽ nhánh)

Nhóm lệnh điều khiển chương trình có thể chia thành 2 loại:

1. Nhảy vô điều kiện
2. Nhảy có điều kiện:

Nhảy vô điều kiện: Chuyển con trỏ chương trình đến vị trí khác

Lệnh	Hoạt động
JZ	Nhảy nếu A = 0
JNZ	Nhảy nếu A ≠ 0

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

Bảng 3-9. Nhảy vô điều kiện

Ví dụ: Hãy tìm tổng của các giá trị 79H, F5H và E2H. Đặt vào trong các thanh ghi R0 (byte thấp) và R5 (byte cao).

```
MOV   A, #0       ; Xoá thanh ghi A = 0
MOV   R5, A       ; Xoá R5
ADD   A #79H      ; Cộng 79H vào A (A = 0 + 79H = 79H)
JNC   N-1         ; Nếu không có nhớ cộng kế tiếp
INC   R5          ; Nếu CY = 1, tăng R5

N-1:  ADD   A, #0F5H    ; Cộng F5H vào A (A = 79H + F5H = 6EH)
                        ;và CY = 1
      JNC   N-2         ; Nhảy nếu CY = 0
      INC   R5          ; Nếu CY = 1 tăng R5 (R5 = 1)
N-2:  ADD   A, #0E2H    ; Cộng E2H vào A (A = 6E + E2 = 50)
                        ; và CY = 1
      JNC   OVER        ; Nhảy nếu CY = 0
      INC   R5          ; Nếu CY = 1 tăng R5
 OVER:MOV   R0, A       ; Bây giờ R0 = 50H và R5 = 02
```

Nhảy có điều kiện: Chỉ chuyển con trỏ chương trình đến vị trí khác từ vị trí hiện thời nếu thả mãn điều kiện. Trong 8051 có hai lệnh nhảy không điều kiện đó là: LJMP - nhảy xa và SJMP - nhảy gần.

- Nhảy xa LJMP: Nhảy xa LJMP là một lệnh 3 byte trong đó byte đầu tiên là mã lệnh còn hai byte còn lại là địa chỉ 16 bit của đích. Địa chỉ đích 02 byte có phép một phép nhảy đến bất kỳ vị trí nhớ nào trong khoảng 0000 - FFFFH.
- Nhảy gần SJMP: Trong 2 byte này thì byte đầu tiên là mã lệnh và byte thứ hai là chỉ tương đối của địa chỉ đích. Đích chỉ tương đối trong phạm vi 00 - FFH được chia thành các lệnh nhảy tới và nhảy lùi: Nghĩa là -128 đến +127 byte của bộ nhớ tương đối so với địa chỉ hiện thời của bộ đếm chương trình. Nếu là lệnh nhảy tới thì địa chỉ đích có thể nằm trong khoảng 127 byte từ giá trị hiện thời của bộ đếm chương trình. Nếu địa chỉ đích ở phía sau thì nó có thể nằm trong khoảng -128 byte từ giá trị hiện hành của PC.

Các lệnh gọi: Một lệnh chuyển điều khiển khác là lệnh CALL được dùng để gọi một chương trình con. Các chương trình con thường được sử dụng để thực thi các công việc cần phải được thực hiện thường xuyên. Điều này làm cho chương trình

trở nên có cấu trúc hơn ngoài việc tiết kiệm được thêm không gian bộ nhớ. Trong 8051 có 2 lệnh để gọi đó là: Gọi xa CALL và gọi tuyệt đối ACALL

- Lệnh gọi xa LCALL: Trong lệnh 3 byte này thì byte đầu tiên là mã lệnh, còn hai byte sau được dùng cho địa chỉ của chương trình con đích.
- Lệnh gọi tuyệt đối ACALL (Absolute call): Lệnh ACALL là lệnh 2 byte khác với lệnh LCALL dài 3 byte. Do ACALL chỉ có 2 byte nên địa chỉ đích của chương trình con phải nằm trong khoảng 2k byte địa chỉ vì chỉ có 11bit của 2 byte được sử dụng cho địa chỉ.

3.3.3 Cấu trúc chung chương trình hợp ngữ cho 8051

a) Các thành phần cơ bản của ngôn ngữ Assembly:

- Lables: Nhãn – đánh dấu cho một đoạn lệnh
- Orders: Lệnh
- Directives: Định hướng chương trình dịch
- Comments: Các lời chú thích

Một dòng lệnh trong chương trình hợp ngữ gồm có các trường sau:

```
Tên   Lệnh       Toán hạng  Chú thích
A:    Mov        AH, 10h    ; Đưa giá trị 10h vào thanh ghi AH
```

Để có thể dịch thành file mã máy dạng HEX-Code trước khi download vào Chip thì một chương trình assembly phải tuân thủ các nguyên tắc sau:

- Mỗi dòng lệnh không vượt quá 255 ký tự
- Mỗi dòng lệnh phải bắt đầu bằng 1 ký tự, nhãn, lệnh hoặc chỉ thị định hướng chương trình dịch
- Mọi thứ sau dấu ";" được xem là lời giải thích và chương trình dịch sẽ bỏ qua.
- Các thành phần của mỗi dòng lệnh cách biệt nhau ít nhất bằng một dấu cách.

b) Khai báo trong lập trình hợp ngữ cho 8051

- **Khai báo biến**

Ten_bien DB Gia_Tri_Khoi_Tao

DB là một chỉ lệnh dữ liệu được sử dụng rộng rãi nhất trong hợp ngữ. Nó được dùng để định nghĩa dữ liệu 8 bit. Khi DB được dùng để định nghĩa byte dữ liệu thì các số có thể ở dạng thập phân, nhị phân, Hex hoặc ở dạng thức ASCII. Đối với dữ liệu thập phân thì cần đặt chữ "D" sau số thập phân, đối với số nhị phân thì đặt chữ "B" và đối với dữ liệu dạng Hex thì cần đặt chữ "H".

Khi dữ liệu có kích thước là 2byte sử dụng: **DW** để khai báo biến kiểu nguyên

Ví dụ

```
DATA1:     DB    2D               ; Số thập phân
DATA2:     DB    00110101B        ; Số nhị phân (35 ở dạng Hex)
DATA3:     DB    39H              ; Số dạng Hex
DATA4 DB    "Ky thuat may tinh"   ; Các ký tự ASCII
```

- **Khai báo hằng**

Ten_Hang EQU Gia_tri

Được dùng để định nghĩa một hằng số mà không chiếm ngăn nhớ nào. Chỉ lệnh EQU không dành chỗ cất cho dữ liệu nhưng nó gắn một giá trị hằng số với nhãn dữ liệu sao cho khi nhãn xuất hiện trong chương trình giá trị hằng số của nó sẽ được thay thế đối với nhãn

Ví dụ:

```
COUNT EQU  25
MOV   R3, #count          ; Khi thực hiện lện "MOV R3, #COUNT"
                          ;thì thanh ghi R3 sẽ được nạp giá trị 25
```

- **Các toán tử**

Ký hiệu	Thực hiện	Ví dụ	Kết quả
+	Cộng	10+5	15
-	Trừ	25-17	8
*	Nhân	7*4	28
/	Chia nguyên	7/4	1
MOD	Chia lấy dư	7 MOD 4	3
SHR	Dịch phải	1000B SHR 2	0010B
SHL	Dịch trái	1010B SHL 2	101000B
NOT	Đảo	NOT 1	1111111111111110B
AND	And bit	1101B AND 0101B	0101B
OR	Or bit	1101B OR 0101B	1101B
XOR	Xor	1101B XOR 0101B	1000B
LOW	Lấy byte thấp	LOW(0AADDH)	0DDH
HIGH	Lấy byte cao	HIGH(0AADDH)	0AAH
EQ, =	So sánh bằng	7 EQ 4 or 7=4	0 (false)
NE,<>	SS Không bằng	7 NE 4 or 7<>4	0FFFFH (true)
GT, >	SS lớn hơn	7 GT 4 or 7>4	0FFFFH (true)
GE, >=	SS nhỏ hơn hoặc bằng	7 GE 4 or 7>=4	0FFFFH (true)
LT, <	SS nhỏ hơn	7 LT 4 or 7<4	0 (false)
LE,<=	SS nhỏ hơn hoặc bằng	7 LE 4 or 7<=4	0 (false)

Bảng 3-10. Các toán tử

- **Tên**

Thay vì phải nhớ tên từng thanh ghi, hay từng bit, ta có thể gán cho nó một cái nhã gợi nhớ tương ưngs với chức năng của nó, assembly hỗ trợ việc đặt tên theo quy tắc sau:

- Tên được tổ hợp từ các ký tự (A-Z, a-z), các số (0-9), các ký tự đặc biệt ("?" Và "_") và không phâ biệt chữ cái và chữ thường.

- Độ dài tên tối đa là 255 ký tự, nhưng chỉ 32 ký tự đầu được dùng để phân biệt
- Tên phải bắt đầu bằng ký tự.
- Không được trùng với các từ khóa sau:

A	AB	ACALL	ADD	JZ	LCALL	LE	LJMP
ADDC	AJMP	AND	ANL	LOW	LT	MOD	MOV
AR0	AR1	AR2	AR3	MOVC	MOVX	MUL	NE
AR4	AR5	AR6	AR7	NOP	NOT	OR	ORG
BIT	BSEG	C	CALL	ORL	PC	POP	PUSH
CJNE	CLR	CODE	CPL	R0	R1	R2	R3
CSEG	DA	DATA	DB	R4	R5	R6	R7
DBIT	DEC	DIV	DJNZ	RET	RETI	RL	RLC
DPTR	DS	DSEG	DW	RR	RRC	SET	SETB
END	EQ	EQU	GE	SHL	SHR	SJMP	SUBB
GT	HIGH	IDATA	INC	SWAP	USING	XCH	XCHD
ISEG	JB	JBC	JC	XDATA	XOR	XRL	XSEG
JMP	JNB	JNC	JNZ	JZ	LCALL	LE	LJMP
LOW	LT	MOD	MOV				

c) Cấu trúc một chương trình hợp ngữ

```
ORG (Vị trí bắt đầu con trỏ chương trình )
…..
<đoạn chương trình chính>
…..
<các chương trình con>
…..
END.(Kết thúc chương trình)
```

Ví dụ:

```
ORG 00H ;(con trỏ chương trình bắt đầu từ 00h)
LJMP MAIN ; nhảy tới vị trí có nhãn là MAIN)
; (vị trí bắt đầu chương trình chính MAIN):
ORG 0030H
MAIN:
MOV R1,#10 ;(nạp cho R1 giá trị là 10).
LAP1:
DJNZ R1,LAP1
END              ; (Kết thúc chương trình.)
```

Con trỏ: vị trí mà vi điều khiển bắt đầu thực thi tại đó. Thường khi bắt đầu con trỏ có địa chỉ thấp nhất là 00h, tuy nhiên người lập trình cũng có thể quy định cho nó làm việc tại một vị trí bất kỳ

Ví dụ:

```
ORG 00H          ; Bắt đầu tại vị trí 00h
ORG 0030H        ; Bắt đầu tại vị trí 0030h
```

Chương trình con:

```
Nhãn:
 ..................
 Các câu lệnh
 .....
RET
```

Ví dụ:

```
ORG 00H
LJMP MAIN
ORG 0030H
MAIN:
MOV R1,#10
LCALL LAP1       ;gọi chương trình con
LAP1:
DJNZ R1,LAP1
RET              ; kết thúc chương trình con
END
```

3.4 Bộ đếm và bộ định thời

8051 có hai bộ định thời là Timer 0 và Timer1, ở phần này chúng ta bàn về các thanh ghi của chúng và sau đó trình bày cách lập trình chúng như thế nào để tạo ra các độ trễ thời gian.

❖ Các thanh ghi cơ sở của bộ định thời.

Cả hai bộ định thời Timer 0 và Timer 1 đều có độ dài 16 bít được truy cập như hai thanh ghi tách biệt byte thấp và byte cao. Chúng ta sẽ bàn riêng về từng thanh ghi.

❖ Các thanh ghi của bộ Timer 0.

Thanh ghi 16 bít của bộ Timer 0 được truy cập như byte thấp và byte cao. Thanh ghi byte thấp được gọi là TL0 (Timer 0 bow byte) và thanh ghi byte cao là TH0 (Timer 0 High byte). Các thanh ghi này có thể được truy cập như mọi thanh ghi khác chẳng hạn như A, B, R0, R1, R2 v.v... Ví dụ, lệnh "MOV TL0, #4FH" là chuyển giá trị 4FH vào TL0, byte thấp của bộ định thời 0. Các thanh ghi này cũng có thể được đọc như các thanh ghi khác. Ví dụ "MOV R5, TH0" là lưu byte cao TH0 của Timer 0 vào R5.

TH0								TL0							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Hình 3-17. Các thanh ghi của bộ Timer 0

❖ Các thanh ghi của bộ Timer 1.

Bộ định thời gian Timer 1 cũng dài 16 bít và thanh ghi 16 bít của nó được chia ra thành hai byte là TL1 và TH1. Các thanh ghi này được truy cập và đọc giống như các thanh ghi của bộ Timer 0 ở trên.

TH1								TL1							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

Hình 3-18. Các thanh ghi của bộ Timer 1

❖ Thanh ghi TMOD (chế độ của bộ định thời).

Cả hai bộ định thời Timer 0 và Timer 1 đều dùng chung một thanh ghi được gọi là IMOD để thiết lập các chế độ làm việc khác nhau của bộ định thời. Thanh ghi TMOD là thanh ghi 8 bít gồm có 4 bít thấp được thiết lập dành cho bộ Timer 0 và 4 bít cao dành cho Timer 1. Trong đó hai bít thấp của chúng dùng để thiết lập chế độ của bộ định thời, còn 2 bít cao dùng để xác định phép toán. Các phép toán này sẽ được bàn dưới đây.

TMOD Register

MSB							LSB
GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer1				Timer0			

Hình 3-19. Timer TMOD

❖ Các bít M1, M0:

Là các bít chế độ của các bộ Timer 0 và Timer 1. Chúng chọn chế độ của các bộ định thời: 0, 1, 2 và 3. Chế độ 0 là một bộ định thời 13, chế độ 1 là một bộ định thời 16 bít và chế độ 2 là bộ định thời 8 bít. Chúng ta chỉ tập chung vào các chế độ thường được sử dụng rộng rãi nhất là chế độ 1 và 2. Chúng ta sẽ sớm khám phá ra các đặc tính củ các chế độ này sau khi khám phần còn lại của thanh ghi TMOD. Các chế độ được thiết lập theo trạng thái của M1 và M0 như sau:

M1	M0	Chế độ	Chế độ hoạt động
0	0	0	Bộ định thời 13 bít gồm 8 bít là bộ định thời/ bộ đếm 5 bít đặt trước
0	1	1	Bộ định thời 16 bít (không có đặt trước)
1	0	2	Bộ định thời 8 bít tự nạp lại
1	1	3	Chế độ bộ định thời chia tách

Bảng 3-11. Chế độ hoạt động của Timer/Counter

❖ C/ T (đồng hồ/ bộ định thời).

Bít này trong thanh ghi TMOD được dùng để quyết định xem bộ định thời được dùng như một máy tạo độ trễ hay bộ đếm sự kiện. Nếu bít C/T = 0 thì nó được dùng như một bộ định thời tạo độ chễ thời gian. Nguồn đồng hồ cho chế độ trễ thời gian là

tần số thạch anh của 8051. ở phần này chỉ bàn về lựa chọn này, công dụng của bộ định thời như bộ đếm sự kiện thì sẽ được bàn ở phần kế tiếp.

Ví dụ : Hãy cho biết chế độ nào và bộ định thời nào đối với các trường hợp sau:

a) MOV TMOD, #01H b) MOV TMOD, #20H c) MOV TMOD, #12H

Lời giải: Chúng ta chuyển đổi giá trị từ số Hex sang nhị phân và đối chiếu với từng bít trong thanh ghi TMOD ta có:

a) `TMOD = 0000 0001`, chế độ 1 của bộ định thời Timer 0 được chọn.
b) `TMOD = 0010 0000`, chế độ 1 của bộ định thời Timer 1 được chọn.
c) `TMOD = 0001 0010`, chế độ 1 của bộ định thời Timer 0 và chế độ 1 của Timer 1 được chọn.

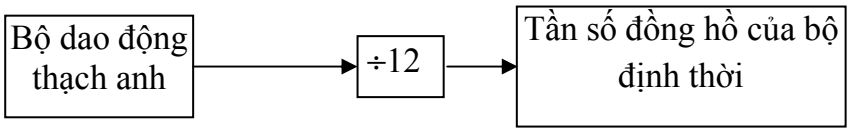
❖ Nguồn xung đồng hồ cho bộ định thời:

Như chúng ta biết, mỗi bộ định thời cần một xung đồng hồ để giữ nhịp. Vậy nguồn xung đồng hồ cho các bộ định thời trên 8051 lấy ở đâu? Nếu C/T = 0 thì tần số thạch anh đi liền với 8051 được làm nguồn cho đồng hồ của bộ định thời. Điều đó có nghĩa là độ lớn của tần số thạch anh đi kèm với 8051 quyết định tốc độ nhịp của các bộ định thời trên 8051. Tần số của bộ định thời luôn bằng 1/12 tần số của thạch anh gắn với 8051.

Ví dụ:

Hãy tìm tần số đồng bộ và chu kỳ của bộ định thời cho các hệ dựa trên 8051 với các tần số thạch anh sau:

a) 12MHz
b) 16MHz
c) 11,0592MHz



Lời giải:

a) $\frac{1}{12}\times 12\text{MHz} = 1\text{MHz}$ vμ $T = \frac{1}{1/1\text{MHz}} = 1\mu\text{s}$

b) $\frac{1}{12}\times 16\text{MHz} = 1{,}111\text{Mz}$ vμ $T = \frac{1}{1{,}333\text{MHz}} = 0{,}75\mu\text{s}$

c) $\frac{1}{12}\times 11{,}0592\text{MHz} = 921{,}6\text{kHz}$ vμ $T = \frac{1}{0{,}9216\text{MHz}} = 1{,}085\mu\text{s}$

Mặc dù các hệ thống dựa trên 8051 khác với tần số thạch anh từ 10 đến 40MHz, song ta chỉ tập chung vào tần số thạch anh 11,0592MHz. Lý do đằng sau một số lẻ như vậy là phải làm việc với tần suất baud đối với truyền thông nối tiếp của 8051. Tần số XTAL = 11,0592MHz cho phép hệ 8051 truyền thông với IBM PC mà không có lỗi.

❖ Bít cổng GATE.

Một bít khác của thanh ghi TMOD là bít cổng GATE. Để ý trên thanh ghi TMOD ta thấy cả hai bộ định thời Timer0 và Timer1 đều có bít GATE. Vậy bít GATE dùng để làm gì? Mỗi bộ định thời thực hiện điểm khởi động và dừng. Một số bộ định thời thực hiện điều này bằng phần mềm, một số khác bằng phần cứng và một số khác vừa bằng phần cứng vừa bằng phần mềm. Các bộ định thời trên 8051 có cả hai. Việc khởi động và dừng bộ định thời được khởi động bằng phần mềm bởi các bít khởi động bộ định thời TR là TR0 và TR1. Điều này có được nhờ các lệnh "SETB TR1" và "CLR TR1" đối với bộ Timer1 và "SETB TR0" và "CLR TR0" đối với bộ Timer0. Lệnh SETB khởi động bộ định thời và lệnh CLR dùng để dừng nó. Các lệnh này khởi động và dừng các bộ định thời khi bít GATE = 0 trong thanh ghi TMOD. Khởi động và ngừng bộ định thời bằng phần cứng từ nguồn ngoài bằng cách đặt bít GATE = 1 trong thanh ghi TMOD. Tuy nhiên, để tránh sự lẫn lộn ngay từ bây giờ ta đặt GATE = 0 có nghĩa là không cần khởi động và dừng các bộ định thời bằng phần cứng từ bên ngoài. Để sử dụng phần mềm để khởi động và dừng các bộ định thời phần mềm để khởi động và dừng các bộ định thời khi GATE = 0. Chúng ta chỉ cần các lệnh "SETB TRx" và "CLR TRx".

Ví dụ:

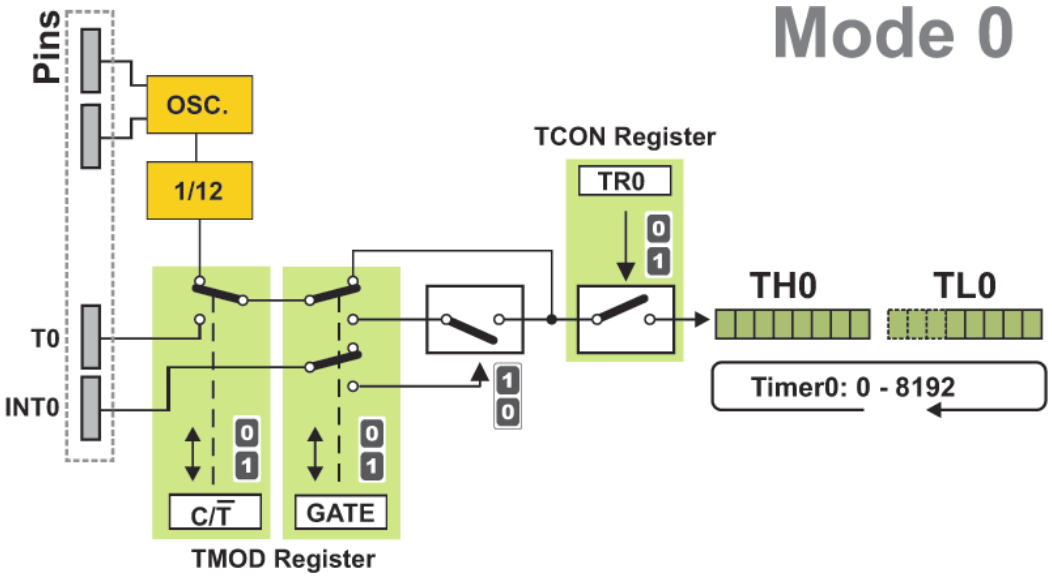
Tìm giá trị cho TMOD nếu ta muốn lập trình bộ Timer0 ở chế độ 2 sử dụng thạch anh XTAL 8051 làm nguồn đồng hồ và sử dụng các lệnh để khởi động và dừng bộ định thời.

Lời giải:

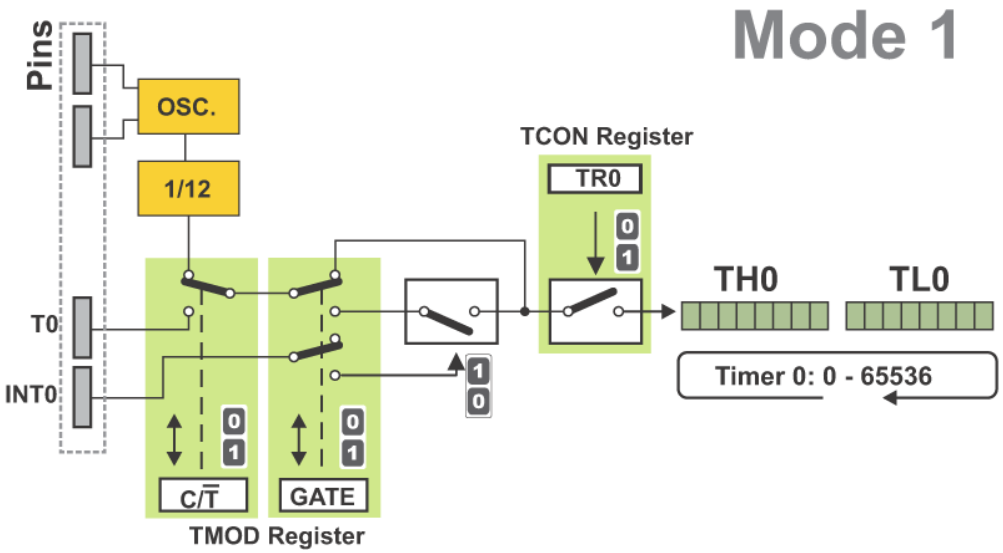
TMOD = 0000 0010: Bộ định thời Timer0, chế độ 2 C/T = 0 dùng nguồn XTAL GATE = 0 để dùng phần mềm trong để khởi động và dừng bộ định thời.

❖ **Các chế độ của bộ đếm/định thời (Timer Mode)**

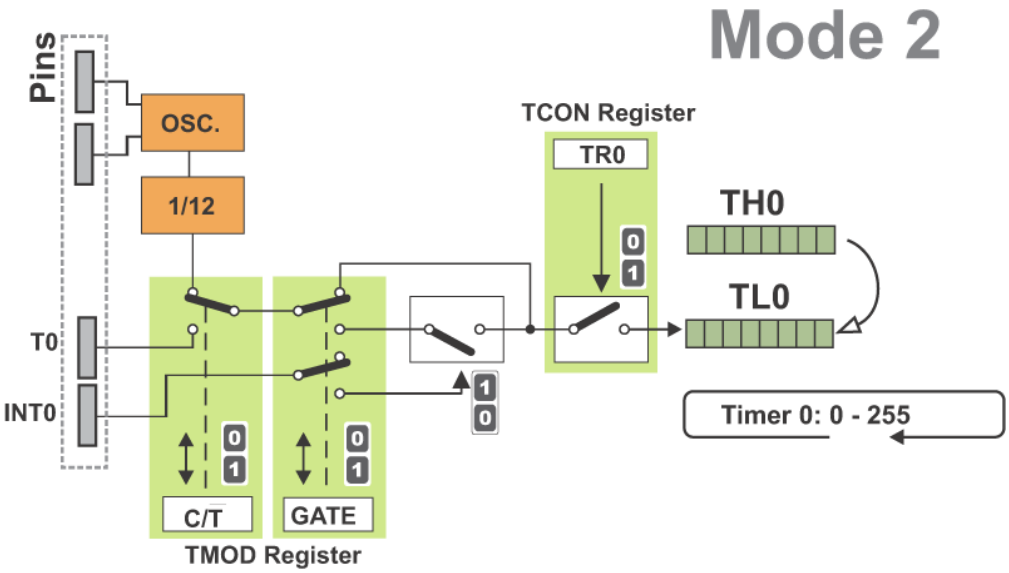
Như vậy, bây giờ chúng ta đã có hiểu biết cơ bản về vai trò của thanh ghi TMOD, chúng ta sẽ xét chế độ của bộ định thời và cách chúng được lập trình như thế nào để tạo ra một độ trễ thời gian. Do chế độ 1 và chế độ 2 được sử dụng rộng rãi nên ta đi xét chi tiết từng chế độ một.



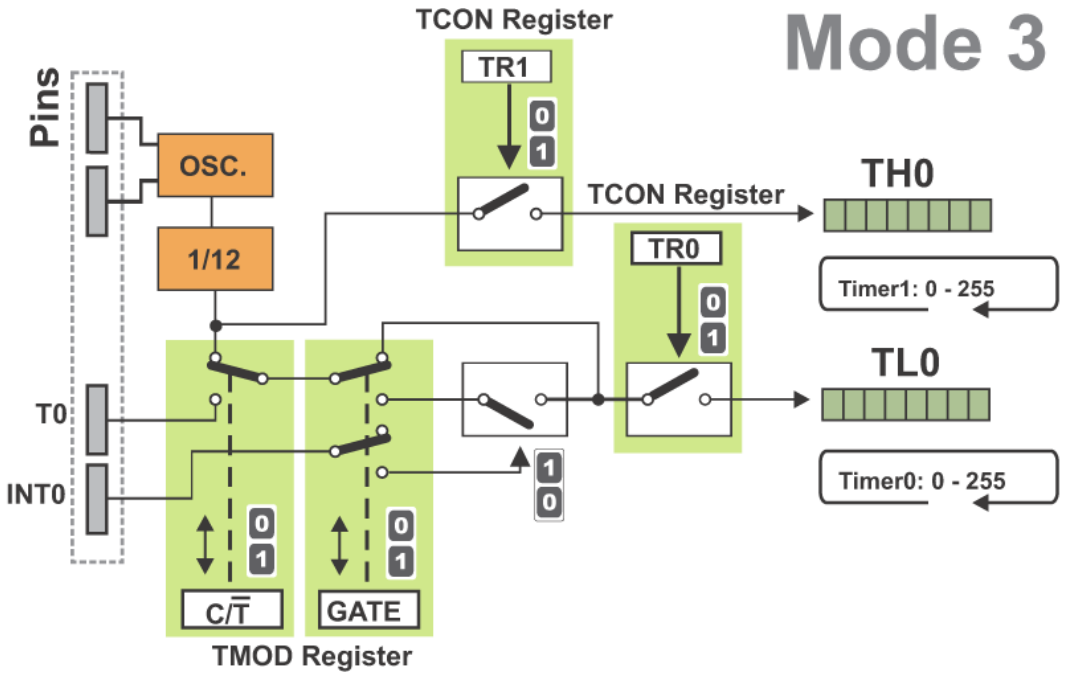
Hình 3-20. Timer 0 – Mode 0



Hình 3-21. Timer 0 – Mode 1



Hình 3-22. Timer 0 – Mode 2



Hình 3-23. Timer 0 – Mode 3

❖ Ngắt timer.

Các ngắt timer có địa chỉ Vector ngắt là 000BH (timer 0) và 001BH (timer 1). Ngắt timer xẩy ra khi các thanh ghi timer (TLx ITHx) tràn và set cờ báo tràn (TFx) lên 1. Các cờ timer (TFx) không bị xóa bằng phần mềm. Khi cho phép các ngắt, TFx tự động bị xóa bằng phần cứng khi CPU chuyển đến ngắt.

Ví dụ 1:

Trong chương trình dưới đây ta tạo ra một sóng vuông với độ đầy xung 50% (cùng tỷ lệ giữa phần cao và phần thấp) trên chân P1.5. Bộ định thời Timer0 được dùng để tạo độ trễ thời gian. Hãy phân tích chương trình này.

```
      MOV   TMOD, #01         ; Sử dụng Timer0 và chế độ 1(16 bít)
 HERE:      MOV   TL0, #0F2H        ; TL0 = F2H, byte thấp
            MOV   TH0, #0FFH        ; TH0 = FFH, byte cao
            CPL   P1.5              ; Sử dụng chân P1.5
            ACALL DELAY
            SJMP  HERE              ; Nạp lại TH, TL
 ;_____________                       delay using timer0.
 DELAY:
SETB  TR0               ; Khởi động bộ định thời Timer0
 AGAIN:     JNB   TF0, AGAIN        ; Hiển thị cờ bộ định thời cho đến
                                    ; khi nó vượt  qua FFFFH.
            CLR   TR0               ; Dừng bộ Timer
            CLR   TF0               ; Xoá cờ bộ định thời 0
            RET
```

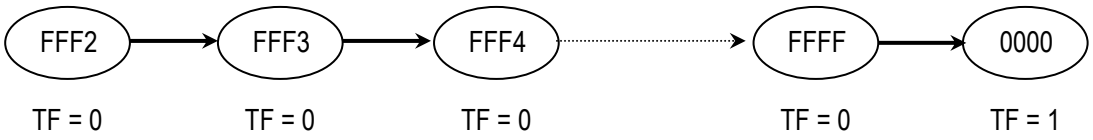
Lời giải:

Trong chương trình trên đây chú ý các bước sau:

1. TMOD được nạp.
2. Giá trị FFF2H được nạp và TH0 - TL0
3. Chân P1.5 được chọn dùng cho phần cao thấp của xung.

4. Chương trình con DELAY dùng bộ định thời được gọi.
5. Trong chương trình con DELAY bộ định thời Timer0 được khởi động bởi lệnh "SETB TR0"
6. Bộ Timer0 đếm lên với mỗi xung đồng hồ được cấp bởi máy phát thạch anh. Khi bộ định thời đếm tăng qua các trạng thái FFF3, FFF4 ... cho đến khi đạt giá trị FFFFH. Và một xung nữa là nó quay về không và bật cờ bộ định thời TF0 = 1. Tại thời điểm này thì lệnh JNB hạn xuống.
7. Bộ Timer0 được dùng bởi lệnh "CLR TR0". Chương trình con DELAY kết thúc và quá trình được lặp lại.

Lưu ý rằng để lặp lại quá trình trên ta phải nạp lại các thanh ghi TH và TL và khởi động lại bộ định thời với giả thiết tần số XTAL = 11, 0592MHz.



Ví dụ 2.

Chương trình dưới đây tạo ra một sóng vuông trên chân P2.5 liên tục bằng việc sử dụng bộ Timer1 để tạo ra độ trễ thời gian. Hãy tìm tần số của sóng vuông nếu tần số XTAL = 11.0592MHz. Trong tính toán không đưa vào tổng phí của các lệnh vòng lặp:

```
            MOV   TMOD, #01H       ; Chọn Timer0, chế độ 1 (16 bít)
 HERE:      MOV   TL1, #34H        ; Đặt byte thấp TL1 = 34H
            MOV   TH0, #76H        ; Đặt byte cao TH1 = 76H
                                   ; (giá trị bộ định thời là 7634H)
            SETB  TR1              ; Khởi động bộ Timer1
AGAIN:      JNB   TF1, BACK        ; ở lại cho đến khi
                                   ;bộ định thời đếm qua 0
            CLR   TR1              ; Dừng bộ định thời.
            CPL   P1.5             ; Bù chân P1.5 để nhận Hi, L0
            CLR   TF               ; Xoá cờ bộ định thời
            SJMP  AGAIN            ; Nạp lại bộ định thời do chế độ 1
                                   ; không tự động nạp lại .
```

Lời giải:

Trong chương trình trên đây ta lưu ý đến đích của SJMP. Ở chế độ 1 chương trình phải nạp lại thanh ghi. TH và TL mỗi lần nếu ta muốn có sóng dạng liên tục. Dưới đây là kết quả tính toán:

Vì FFFFH - 7634H = 89CBH + 1 = 89CCH và 90CCH = 35276 là số lần đếm xung đồng hồ, độ trễ là 35276 × 1.085μs = 38274ms và tần số là **Error! Objects cannot be created from editing field codes.**

Cũng để ý rằng phần cao và phần thấp của xung sóng vuông là bằng nhau.

Trong tính toán trên đây là chưa kể đến tổng phí các lệnh vòng lặp.

Bài tập:

Hãy kiểm tra chương trình sau và tìm độ trễ thời gian theo giây, không tính đến tổng phí các lệnh trong vòng lặp.

```
            MOV   TMOD, #10H ; Chọn bộ Timer1, chế độ 1 (16 bít)
 AGAIN:     MOV   R3, #200         ; Chọn bộ đếm độ giữ chậm lớn
```

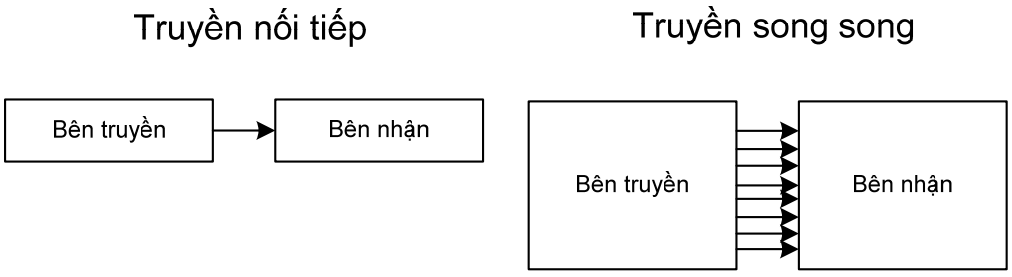
```
          MOV   TL1, #08          ; Nạp byte thấp TL1 = 08
          MOV   TH1, #08          ; Nạp byte cao TH1 = 01
          SETB  TR1               ; Khởi động Timer1
BACK:     JNB   TF1, BACK  ; giữ nguyên cho đến khi
                           ;bộ định thời quay về 0
          CLR   TR1               ; Dừng bộ định thời.
          CLR   TF1               ; Xoá cờ bộ định thời TF1
          DJNZ  R3, AGAIN         ; Nếu R3 không bằng
                                  ;không thì nạp lại bộ định thời.
```

3.5 Truyền thông nối tiếp

Các máy tính truyền dữ liệu theo hai cách: Song song và nối tiếp. Trong truyền dữ liệu song song thường cần 8 hoặc nhiều đường dây dẫn để truyền dữ liệu đến một thiết bị chỉ cách xa vài bước. Ví dụ của truyền dữ liệu song song là các máy in và các ổ cứng, mỗi thiết bị sử dụng một đường cáp với nhiều dây dẫn. Mặc dù trong các trường hợp như vậy thì nhiều dữ liệu được truyền đi trong một khoảng thời gian ngắn bằng cách dùng nhiều dây dẫn song song nhưng khoảng cách thì không thể lớn được. Để truyền dưdx liệu đi xa thì phải sử dụng phương pháp truyền nối tiếp. Trong truyền thông nối tiếp dữ liệu được gửi đi từng bít một so với truyền song song thì một hoặc nhiều byte được truyền đi cùng một lúc. Truyền thông nối tiếp của 8051 là chủ đề của chương này. 8051 đã được cài sẵn khả năng truyền thông nối tiếp, do vậy có thể truyền nhánh dữ liệu với chỉ một số ít dây dẫn.

Các cơ sở của truyền thông nối tiếp.

Khi một bộ vi xử lý truyền thông với thế giới bên ngoài thì nó cấp dữ liệu dưới dạng từng khúc 8 bít (byte) một. Trong một số trường hợp chẳng hạn như các máy in thì thông tin đơn giản được lấy từ đường bus dữ liệu 8 bít và được gửi đi tới bus dữ liệu 8 bít của máy in. Điều này có thể làm việc chỉ khi đường cáp bus không quá dài vì các đường cáp dài làm suy giảm thậm chí làm méo tín hiệu. Ngoài ra, đường dữ liệu 8 bít giá thường đắt. Vì những lý do này, việc truyền thông nối tiếp được dùng để truyền dữ liệu giữa hai hệ thống ở cách xa nhau hàng trăm đến hàng triệu dặm. "Hình 3-24. Truyền thông" là sơ đồ truyền nối tiếp so với sơ đồ truyền song song.



Hình 3-24. Truyền thông

Thực tế là trong truyền thông nối tiếp là một đường dữ liệu duy nhất được dùng thay cho một đường dữ liệu 8 bít của truyền thông song song làm cho nó không chỉ

rẻ hơn rất nhiều mà nó còn mở ra khả năng để hai máy tính ở cách xa nhau có truyền thông qua đường thoại.

Đối với truyền thông nối tiếp thì để làm được các byte dữ liệu phải được chuyển đổi thành các bít nối tiếp sử dụng thanh ghi giao dịch vào - song song - ra - nối tiếp. Sau đó nó có thể được truyền quan một đường dữ liệu đơn. Điều này cũng có nghĩa là ở đầu thu cũng phải có một thanh ghi vào - nối tiếp - ra - song song để nhận dữ liệu nối tiếp và sau đó gói chúng thành từng byte một. Tất nhiên, nếu dữ liệu được truyền qua đường thoại thì nó phải được chuyển đổi từ các số 0 và 1 sang âm thanh ở dạng sóng hình sin. Việc chuyển đổi này thực thi bởi một thiết bị có tên gọi là Modem là chữ viết tắt của "Modulator/ demodulator" (điều chế/ giải điều chế).

Khi cự ly truyền ngắn thì tín hiệu số có thể được truyền như nói ở trên, một dây dẫn đơn giản và không cần điều chế. Đây là cách các bàn PC và IBM truyền dữ liệu đến bo mạch mẹ. Tuy nhiên, để truyền dữ liệu đi xa dùng các đường truyền chẳng hạn như đường thoại thì việc truyền thông dữ liệu nối tiếp yêu cầu một modem để điều chế (chuyển các số 0 và 1 về tín hiệu âm thanh) và sau đó giải điều chế

Trong RS232 thì mức 1 được biểu diễn bởi - 3v đến 25v trong khi đó mức 0 thì ứng với điện áp + 3v đến +25v làm cho điện áp - 3v đến + 3v là không xác định. Vì lý do này để kết nối một RS232 bất kỳ đến một hệ vi điều khiển thì ta phải sử dụng các bộ biến đổi điện áp như MAX232 để chuyển đổi các mức lô-gíc TTL về mức điện áp RS232 và ngược lại. Kết nối RS232 đến MAX232 được như "Hình 3-16 - Ghép nối RS232 với 8051".

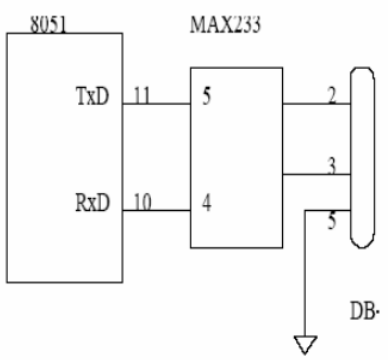
8051 có hai chân được dùng chuyên cho truyền và nhận dữ liệu nối tiếp. Hai chân này được gọi là TxD và RxD và là một phần của cổng P3 (đó là P3.0 và P3.1). chân 11 của 8051 là P3.1 được gán cho TxD và chân 10 (P3.0) được dùng cho RxD. Các chân này tương thích với mức lô-gích TTL. Do vậy chúng đòi hỏi một bộ điều khiển đường truyền để chúng tương thích với RS232. Một bộ điều khiển như vậy là chíp MAX232.

Trong phần này chúng ta sẽ nghiên cứu về các thanh ghi truyền thông nối tiếp của 8051 và cách lập trình chúng để truyền và nhận dữ liệu nối tiếp. Vì các máy tính IBM PC và tương thích được sử dụng rất rộng rãi để truyền thông với các hệ dựa trên 8051, do vậy ta chủ yếu tập trung vào truyền thông nối tiếp của 8051 với cổng COM của PC. Để cho phép truyền dữ liệu giữa máy tín PC và hệ thống 8051 mà không có bất kỳ lỗi nào thì chúng ta phải biết chắc rằng tốc độ baud của hệ 8051 phải phù hợp với tốc độ baud của công COM máy tính PC được cho trong "Bảng 3-12. Một số giá trị thường dùng trong truyền thông nối tiếp".

Tham khảo "[1]"

Các thanh ghi cần nghiên cứu: **SCON, SBUF, TMOD, TH1, TL1, ...**

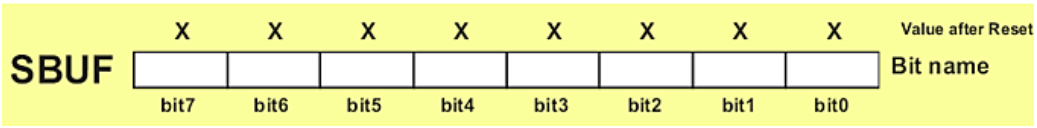
8051 có 1 cổng UART làm việc ở chuẩn TTL, mặc định sau khi khởi động tất các cổng của 8051 đều làm việc ở chế độ vào ra số, vì thế để có thể sử dụng UART cần phải cấu hình cho cổng này làm việc thông qua các thanh ghi điều khiển và ghép nối tương thích với chuẩn rs232.



Hình 3-25. Ghép nối RS232 với 8051

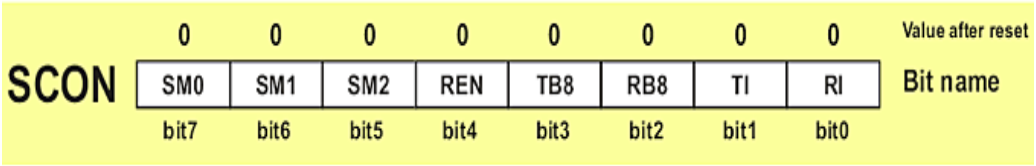
Các thanh ghi điều khiển trong chế dộ UART:

a) **SBUF**: Vùng đệm truyền thông dữ liệu ra/vào cổng nối tiếp.



- Việc truyền dữ liệu tương ứng với việc nạp cho SBUF một giá trị
- Dữ liệu nhận từ RxD cũng được lưu vào SBUF

b) **SCON**: Thanh ghi điều khiển hoạt động cổng nối tiếp



Trong đó:

Bit	Mô tả
SM0	**Lựa chọn mode làm việc**
SM1	
SM2	
REN	= 1: Cho phép nhận = 0: Chỉ truyền
TB8	(=1) Bit truyền thông thứ 8, được sử dụng khi truyền thông ở chế độ 9 bit
RB8	(=1) Bit truyền thông thứ 8, hệ thống sẽ tự đặt nó =1 nếu phát hiện khung truyền là 9bit

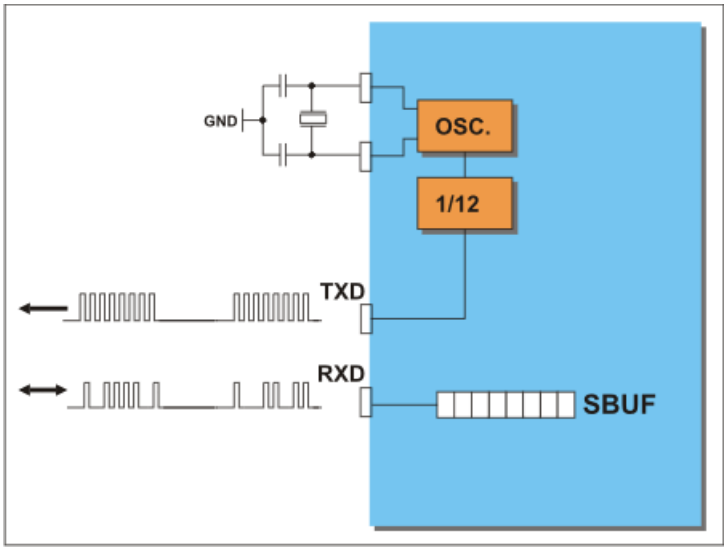
TI	Cờ ngắt truyền. Khi một byte trong SBUF được truyền thành công thì TI=1. Trước khi truyền byte khác bit này cần phải được xóa bằng phần mềm
RI	Cờ ngắt nhận, Khi nhần thành công 1 byte vào SBUF thì RI=1. Sau khi đọc SUBF, RI cần phaiđược xóa bằng phần mềm

Lựa chọn mode làm việc

SM0	SM1	Mode	Description	Baud Rate
0	0	0	Thanh ghi dịch 8 bit	1/12 tần số clock
0	1	1	8-bit UART	Cấu hình qua timer1
1	0	2	9-bit UART	1/32 tần số clock (hoặc 1/64)
1	1	3	9-bit UART	Cấu hình qua timer 1

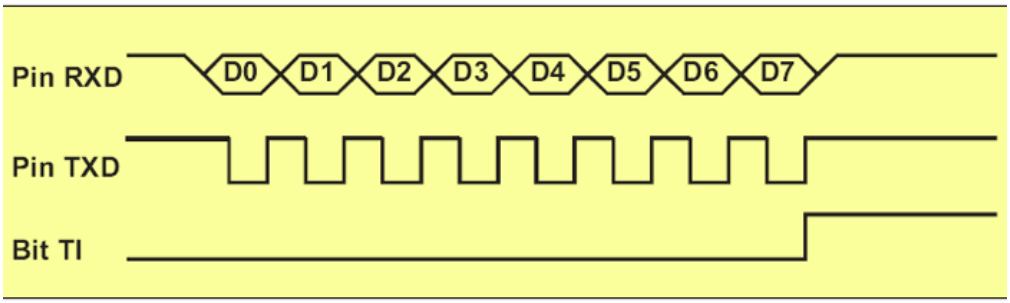
❖ Mode 0

Đây là chế độ thanh ghi dịch 8 bit, không có bit start/stop, ở chế độ này RxD là chân truyền nhận, còn TxD phát xung đồng bộ.



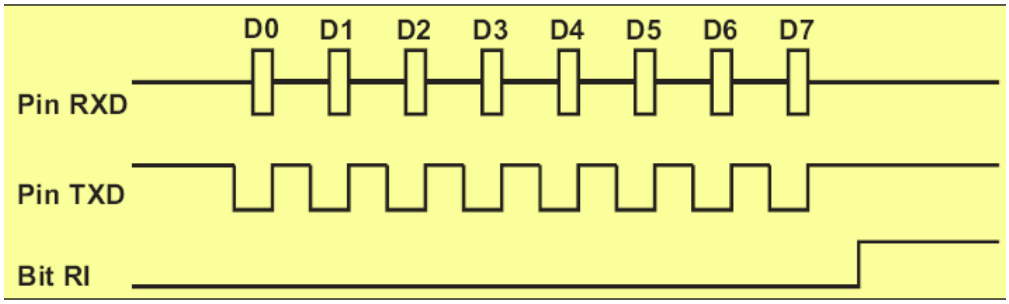
Hình 3-26. Truyền thông nối tiếp – Mode 0

- Quá trình truyền bắt đầu khi ghi giá trị vào SBUF, kết thúc được báo qua TI



Hình 3-27. Giản đồ thời gian truyền nối tiếp – Mode 0

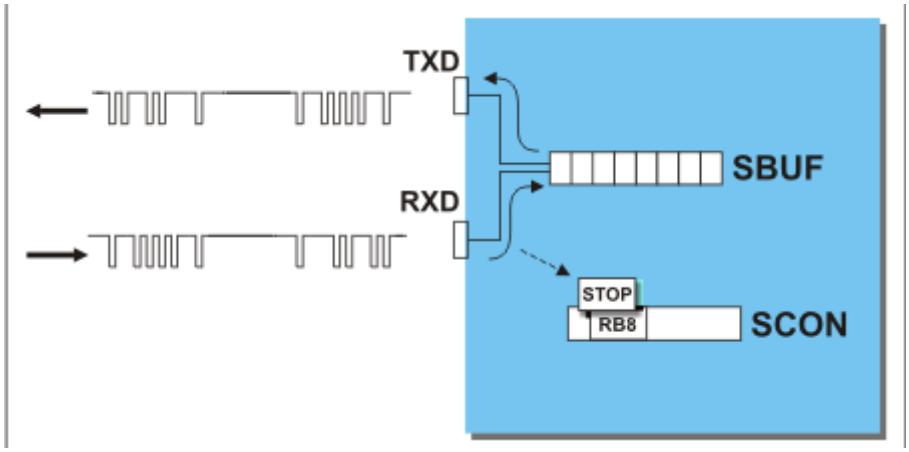
- Quá trình nhận tự động bởi hệ thống và kết thúc khi RI=1



Hình 3-28. Giản đồ thời gian nhận nối tiếp – Mode 0

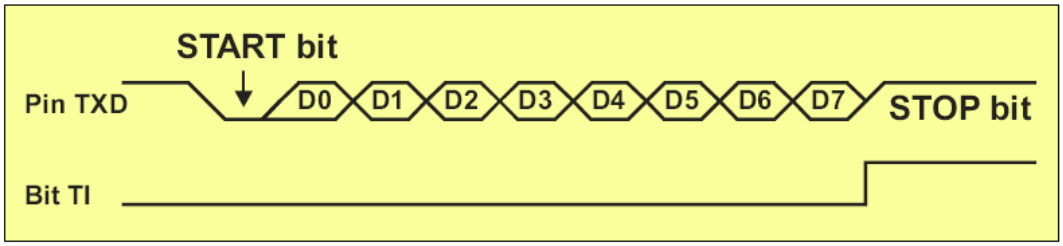
❖ Mode 1

Truyền thông bất đồng bộ với frame truyền 10 bit, gồm 1 start, 8 bit dữ liệu và 1 stop. TxD thực hiện truyền, RxD nhận dữ liệu, tốc độ truyền cài đặt qua Timer 1



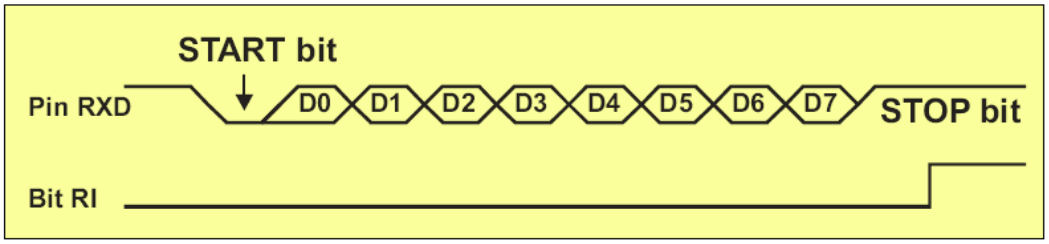
Hình 3-29. Truyền nhận nối tiếp – Mode 1

- Quá trình truyền:



Hình 3-30. Giản đồ thời gian truyền nối tiếp – Mode 1

- Quá trình nhận

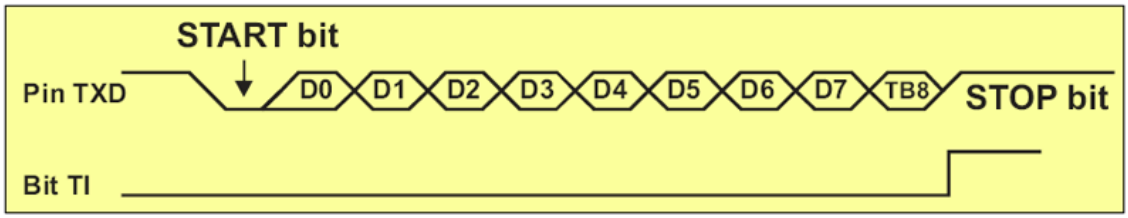


Hình 3-31. Giản đồ thời gian nhận nối tiếp – Mode 1

❖ Mode 2

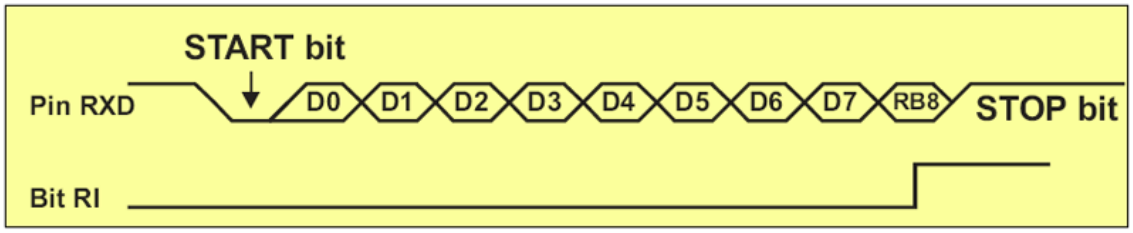
Truyền thông bất đồng bộ với frame truyền 11 bit, gồm 1 start, 8 bit dữ liệu, 1 bit lập trình được(nêu truyền là TB8, nhận là RB8) và 1 bit stop. TxD thực hiện truyền, RxD nhận dữ liệu, tốc độ truyền cài đặt qua Timer 1. Bit thứ 9 thường được dùng là bit phát hiện lỗi party.

- Quá trình truyền



Hình 3-32. Giản đồ thời gian truyền nối tiếp – Mode 2

- Quá trình nhận:



Hình 3-33. Giản đồ thời gian nhận nối tiếp – Mode 2

❖ Mode 3

Mode 3 tương tự mode 2 về mọi mặt ngoại trừ tốc độ baud

❖ Tốc độ Baud

Trong một số mode hoạt động của cổng nối tiếp thì tốc độ baud phụ thuộc vào timer 1. Để cài đặt cần qua các bước sau:

- Cho phép timer 1 hoạt động và cho phép ngắt tràn timer 1
- Cấu hình cho timer 1 làm việc ở chế độ tự nạp lại

Công thức tính:

$$Baud\_Rate = \frac{2^{SMOD} \times F_{Xtal}}{6^{(1-SPD)} \times 12 \times 32 \times [256 - (BRL)]}$$

$$(BRL) = 256 - \frac{2^{SMOD} \times F_{Xtal}}{6^{(1-SPD)} \times 12 \times 32 \times Baud\_Rate}$$

- Đặt giá trị cho thanh ghi TH1 tùy thuộc vào tốc độ mong muốn theo bảng dưới

Baud Rate	Tần số thạch anh					Bit SMOD
	11.0592	12	14.7456	16	20	
150	40 h	30 h	00 h			0
300	A0 h	98 h	80 h	75 h	52 h	0
600	D0 h	CC h	C0 h	BB h	A9 h	0
1200	E8 h	E6 h	E0 h	DE h	D5 h	0
2400	F4 h	F3 h	F0 h	EF h	EA h	0
4800		F3 h	EF h	EF h		1
4800	FA h		F8 h		F5 h	0
9600	FD h		FC h			0
9600					F5 h	1
19200	FD h		FC h			1
38400			FE h			1
76800			FF h			1

Bảng 3-12. Một số giá trị thường dùng trong truyền thông nối tiếp

❖ Một số ví dụ và bài tập:

Ví dụ 1:

Giả sử tần số XTAL = 11.0592MHz cho chương trình dưới đây, hãy phát biểu a) chương trình này làm gì? b) hãy tính toán tần số được Timer1 sử dụng để đặt tốc độ baud? và c) hãy tìm tốc độ baud truyền dữ liệu.

```
MOV   A, PCON     ; Sao nội dung thanh ghi PCON vào thanh ghi ACC
SETB  ACC.7       ; Đặt D7 = 0
MOV   PCON, A     ; Đặt SMOD = 1 để tăng gấp đôi tần
                  ;số baud với tần số XTAL cố định
MOV   TMOD, #20H ; Chọn bộ Timer1, chế độ 2, tự động nạp lại
      MOV   TH1, - 3         ; Chọn tốc độ baud 19200
```

```
                          ;(57600/3=19200) vì SMOD = 1
      MOV   SCON, #50H ; Đóng khung dữ liệu gồm 8 bít
                       ;dữ liệu, 1 Stop và cho phép RI.
      SETB  TR1        ; Khởi động Timer1
      MOV   A, #'B'    ; Truyền ký tự B
A_1:  CLR   TI         ; Khẳng định TI = 0
      MOV   SBUF, A    ;Truyền nó
 H_1: JNB   TI, H_1    ; Chờ ở đây cho đến khi bít cuối được gửi đi
      SJMP  A_1        ; Tiếp tục gửi "B"
```

Lời giải:

a) Chương trình này truyền liên tục mã ASCII của chữ B (ở dạng nhị phân là 0100 0010)

b) Với tần số XTAL = 11.0592MHz và SMOD = 1 trong chương trình trên ta có: 11.0592MHz/12 = 921.6kHz là tần số chu trình máy, 921.6kHz/16 = 57.6kHz là tần số được Timer1 sử dụng để đặt tốc độ baud

c) 57.6kHz/3 = 19.200 là tốc độ cần tìm

Ví dụ 2:

Tìm giá trị TH1 (ở dạng thập phân và hex) để đạt tốc độ baud cho các trường hợp sau.

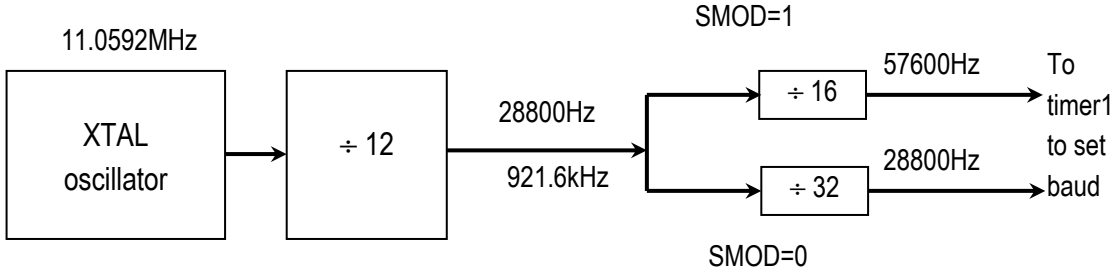
a) 9600 b) 4800 nếu SMOD = 1 và tần số XTAL = 11.0592MHz

Lời giải:

Với tần số XTAL = 11.0592MHz và SMOD = 1 ta có tần số cấp cho Timer1 là 57.6kHz.

a) 57.600/9600 = 6 do vậy TH1 = - 6 hay TH1 = FAH

b) 57.600/4800 = 12 do vậy TH1 = - 12 hay TH1 = F4H



Bài tập:

Hãy tìm tốc độ baud nếu TH1 = -2, SMOD = 1 và tần số XTAL = 11.0592MHz. Tốc độ này có được hỗ trợ bởi các máy tính IBM PC và tương thích không?

3.6 Xử lý ngắt

Một ngắt là một sự kiện bên trong hoặc bên ngoài làm ngắt bộ vi điều khiển để báo cho nó biết rằng thiết bị cần dịch vụ của nó. Trong chương này ta tìm hiểu khái niệm ngắt và lập trình ngắt.

Một bộ vi điều khiển có thể phục vụ một vài thiết bị, có hai cách để thực hiện điều này đó là sử dụng các ngắt và thăm dò (polling). Trong phương pháp sử dụng các ngắt thì mỗi khi có một thiết bị bất kỳ cần đến dịch vụ của nó thì nó bao cho bộ

vi điều khiển bằng cách gửi một tín hiệu ngắt. Khi nhận được tín hiệu ngắt thì bộ vi điều khiển ngắt tất cả những gì nó đang thực hiện để chuyển sang phục vụ thiết bị. Chương trình đi cùng với ngắt được gọi là trình dịch vụ ngắt ISR (Interrupt Service Routine) hay còn gọi là trình quản lý ngắt (Interrupt handler). Còn trong phương pháp thăm dò thì bộ vi điều khiển hiển thị liên tục tình trạng của một thiết bị đã cho và điều kiện thoả mãn thì nó phục vụ thiết bị. Sau đó nó chuyển sang hiển thị tình trạng của thiết bị kế tiếp cho đến khi tất cả đều được phục vụ. Mặc dù phương pháp thăm dò có thể hiển thị tình trạng của một vài thiết bị và phục vụ mỗi thiết bị khi các điều kiện nhất định được thoả mãn nhưng nó không tận dụng hết cộng dụng của bộ vi điều khiển. Điểm mạnh của phương pháp ngắt là bộ vi điều khiển có thể phục vụ được rất nhiều thiết bị (tất nhiên là không tại cùng một thời điểm). Mỗi thiết bị có thể nhận được sự chú ý của bộ vi điều khiển dựa trên mức ưu tiên được gán cho nó. Đối với phương pháp thăm dò thì không thể gán mức ưu tiên cho các thiết bị vì nó kiểm tra tất cả mọi thiết bị theo kiểu hơi vòng. Quan trọng hơn là trong phương pháp ngắt thì bộ vi điều khiển cũng còn có thể che hoặc làm lơ một yêu cầu dịch vụ của thiết bị. Điều này lại một lần nữa không thể thực hiện được trong phương pháp thăm dò. Lý do quan trọng nhất là phương pháp ngắt được ưu chuộng nhất là vì phương pháp thăm dò làm lãng phí thời gian của bộ vi điều khiển bằng cách hỏi dò từng thiết bị kể cả khi chúng không cần đến dịch vụ.

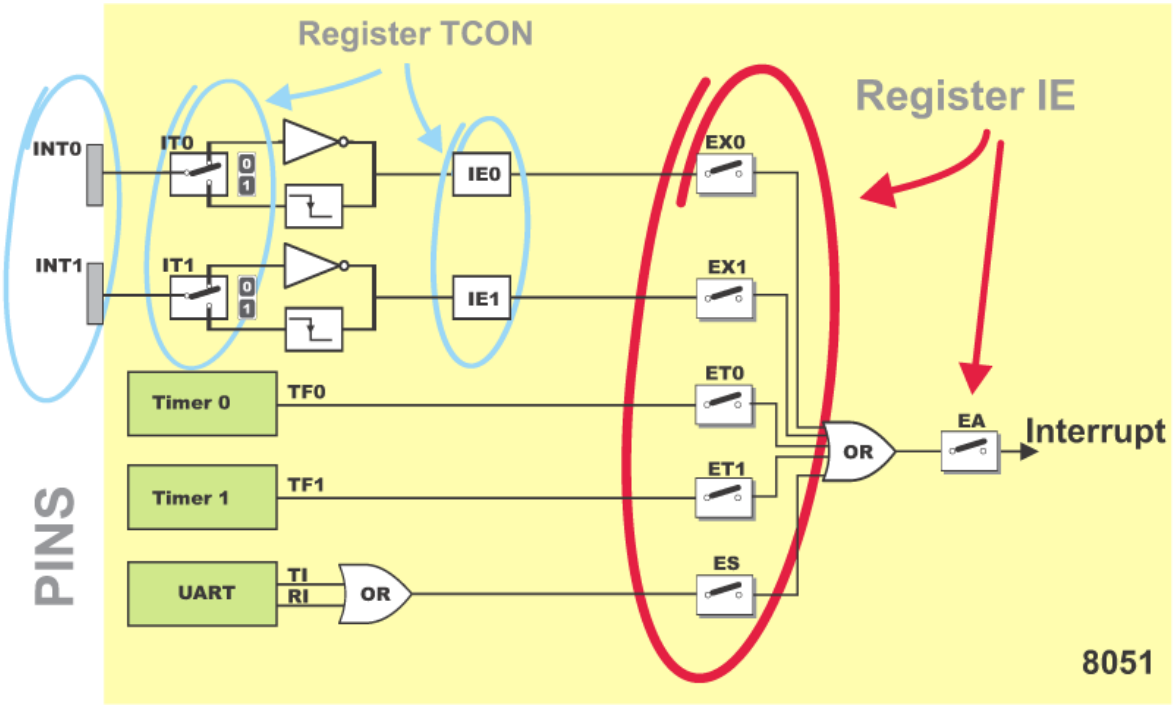
Ví dụ trong các bộ định thời, ta đã dùng lệnh "JNB TF, đích" và đợi cho đến khi bộ định thời quay trở về 0. Trong ví dụ đó, trong khi chờ đợi thì ta có thể làm việc được gì khác có ích hơn, chẳng hạn như khi sử dụng phương pháp ngắt thì bộ vi điều khiển có thể đi làm các việc khác và khi cờ TF bật lên nó sẽ ngắt bộ vi điều khiển cho dù nó đang làm bất kỳ điều gì.

Trình phục vụ ngắt.

Đối với mỗi ngắt thì phải có một trình phục vụ ngắt ISR hay trình quản lý ngắt. khi một ngắt được gọi thì bộ vi điều khiển phục vụ ngắt. Khi một ngắt được gọi thì bộ vi điều khiển chạy trình phục vụ ngắt. Đối với mỗi ngắt thì có một vị trí cố định trong bộ nhớ để giữ địa chỉ ISR của nó. Nhóm các vị trí nhớ được dành riêng để gửi các địa chỉ của các ISR được gọi là bảng véc tơ ngắt, xem "Hình 3-35. Bảng vector ngắt và ví dụ"

8051 hỗ trợ 5 loại ngắt, có thể cho phép hoặc cấm ngắt với từng loại thông qua thanh ghi điều khiển ngắt IE, hoặc có thể cấm tất cả các ngắt thông qua bit EA.

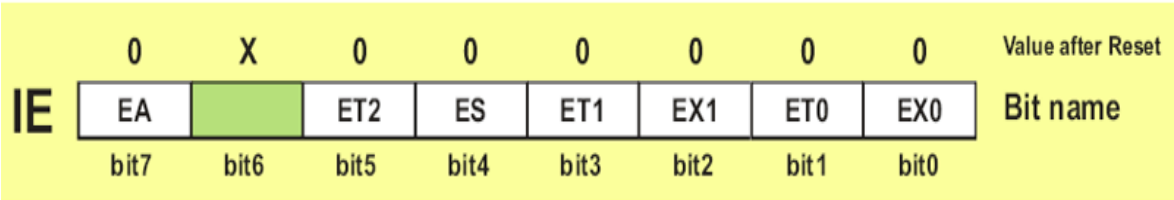
Các tín hiệu điều khiển ngắt có thể được mô tả như hình dưới



Hình 3-34. Các tín hiệu điều khiển ngắt

Ở hình trên chỉ có 1 điểm chú ý đó là hai tín hiệu IT0 và IT1, hai bit này lựa chọn nguyên nhân ngắt cho 2 ngắt ngoài INTR0 và INTR1. Nếu =1 thì ngắt tại sườn âm, =0 ngắt tại sườn dương

Thanh ghi điều khiển ngắt IE



Trong đó:

Bit	Mô tả
EA	Cho phép/cấm ngắt toàn cục = 0: Cấm tất cả các ngắt = 1: Cho phép các ngắt
ES	= 0: Cấm ngắt truyền thông nối tiếp = 1: Cho phép ngắt truyền thông nối tiếp
ET1	= 0: Cấm ngắt Timer 1 = 1: Cho phép ngắt Timer 1
EX1	= 0: Cấm ngắt ngoại vi INT0 = 1: Cho phép ngắt ngoại vị INT0
ET0	= 0: Cấm ngắt Timer 0 = 1: Cho phép ngắt timer 0
EX0	= 0: Cấm ngắt ngoại vi INT1 = 1: Cho phép

❖ **Các bước khi thực hiện một ngắt.**

Khi kích hoạt một ngắt bộ vi điều khiển đi qua các bước sau:

1. Nó kết thúc lệnh đang thực hiện và lưu địa chỉ của lệnh kế tiếp (PC) vào ngăn xếp.
2. Nó cũng lưu tình trạng hiện tại của tất cả các ngắt vào bên trong (nghĩa là không lưu vào ngăn xếp).
3. Nó nhảy đến một vị trí cố định trong bộ nhớ được gọi là bảng véc tơ ngắt nới lưu giữ địa chỉ của một trình phục vụ ngắt.
4. Bộ vi điều khiển nhận địa chỉ ISR từ bảng véc tơ ngắt và nhảy tới đó. Nó bắt đầu thực hiện trình phục vụ ngắt cho đến lệnh cuối cùng của ISR là RETI (trở về từ ngắt).
5. Khi thực hiện lệnh RETI bộ vi điều khiển quay trở về nơi nó đã bị ngắt. Trước hết nó nhận địa chỉ của bộ đếm chương trình PC từ ngăn xếp bằng cách kéo hai byte trên đỉnh của ngăn xếp vào PC. Sau đó bắt đầu thực hiện các lệnh từ địa chỉ đó.

Lưu ý ở bước 5 đến vai trò nhạy cảm của ngăn xếp, vì lý do này mà chúng ta phải cẩn thận khi thao tác các nội dung của ngăn xếp trong ISR. Đặc biệt trong ISR cũng như bất kỳ chương trình con CALL nào số lần đẩy vào ngăn xếp (Push) và số lần lấy ra từ nó (Pop) phải bằng nhau.

❖ **Lập trình ngắt**

Khi có một ngắt, chương trình chính sẽ bị dừng, con trỏ chương trình ngay lập tức được chuyển đến một địa chỉ quy định sẵn trong bản vector ngắt như hình dưới:

```
ORG 0
rom_start: LJMP
main_code
ORG 13H
int1_vec: LJMP
int1_isr
ORG 30H
main_code:      ;bla bla
; ....
int1_isr:       ;bla bla
```

Interrupt	ROM Location	Pin
Reset	0000H	9
INT0	0003H	P3.2
TF0	000BH	
INT1	0013H	P3.3
TF1	001BH	
S0	0023H	

Hình 3-35. Bảng vector ngắt và ví dụ

Một số ví dụ và bài tập:

Ví dụ 1:

Hãy chỉ ra những lệnh để a) cho phép ngắt nối tiếp ngắt Timer0 và ngắt phần cứng ngoài 1 (EX1) và b) cấm (che) ngắt Timer0 sau đó c) trình bày cách cấm tất cả mọi ngắt chỉ bằng một lệnh duy nhất.

Lời giải:

a) MOV IE, #10010110B ; Cho phép ngắt nối tiếp, cho phép ngắt Timer0 và cho phép ngắt phần cứng ngoài.

Vì IE là thanh ghi có thể đánh địa chỉ theo bít nên ta có thể sử dụng các lệnh sau đây để truy cập đến các bít riêng rẽ của thanh ghi:

```
SETB IE.7      ; EA = 1, Cho phép tất cả mọi ngắt
SETB IE.4      ; Cho phép ngắt nối tiếp
SETB IE.1      ; Cho phép ngắt Timer1
SETB IE.2      ; Cho phép ngắt phần cứng ngoài 1
```

(tất cả những lệnh này tương đương với lệnh "MOV IE, #10010110B" trên đây).

b) CLR IE.1 ; Xoá (che) ngắt Timer0

c) CLR IE.7 ; Cấm tất cả mọi ngắt.

Ví dụ 2:

Hãy viết chương trình nhân liên tục dữ liệu 8 bít ở cổng P0 và gửi nó đến cổng P1 trong khi nó cùng lúc tạo ra một sóng vuông chu kỳ 200us trên chân P2.1. Hãy sử dụng bộ Timer0 để tạo ra sóng vuông, tần số của 8051 là XTAL = 11.0592MHz.

Lời giải:

Ta sử dụng bộ Timer0 ở chế độ 2 (tự động nạp lại) giá trị nạp cho TH0 là 100/1.085us = 92.

```
; - - Khi khởi tạo vào chương trình main tránh dùng không gian.
; Địa chỉ dành cho bảng véc tơ ngắt.
          ORG   0000H
          CPL   P2.1            ; Nhảy đến bảng véc tơ ngắt.
; - - Trình ISR dành cho Timer0 để tạo ra sóng vuông.
          ORG   0030H      ; Ngay sau địa chỉ bảng véc-tơ ngắt
MAIN:          TMOD, #02H ; Chọn bộ Timer0, chế độ 2 tự nạp lại
          MOV   P0, #0FFH  ; Lấy P0 làm cổng vào nhận dữ liệu
          MOV   TH0, # - 92      ; Đặt TH0 = A4H cho – 92
          MOV   IE, #82H         ; IE = 1000 0010 cho phép Timer0
          SETB  TR0              ; Khởi động bộ Timer0
BACK:          MOV   A, P0       ; Nhận dữ liệu vào từ cổng P0
          MOV   P1, A            ; Chuyển dữ liệu đến cổng P1
          SJMP  BACK             ; Tiếp tục nhận và chuyển dữ liệu
                                 ; Chừng nào bị ngắt bởi TF0
          END
```

Trong ví dụ 2 trình phục vụ ngắt ISR ngắn nên nó có thể đặt vừa vào không gian địa chỉ dành cho ngắt Timer0 trong bảng véc tơ ngắt. Tất nhiên không phải lúc nào cũng làm được như vậy. Xét ví dụ 3 dưới đây.

Ví dụ 3:

Hãy viết lại chương trình ở ví dụ 2 để tạo sóng vuông với mức cao kéo dài 1085us và mức thấp dài 15us với giả thiết tần số XTAL = 11.0592MHz. Hãy sử dụng bộ định thời Timer1.

Lời giải:

Vì 1085us là 1000x1085us nên ta cần sử dụng chế độ 1 của bộ định thời Timer1.

```
;Khi khởi tạo tránh sử dụng không gian dành cho bảng véc tơ ngắt.
           ORG   0000H
           LJMP  MAIN             ; Chuyển đến bảng véc tơ ngắt.
; - - Trình ISR đối với Timer1 để tạo ra xung vuông
           OR6   001BH            ; Địa chỉ ngắt của Timer1
                                  ; trong bảng véc tơ ngắt
           LJMP  ISR_T1           ; Nhảy đến ISR

 ; - - Bắt đầu các chương trình chính MAIN.
           ORG   0030H      ; Sau bảng véc tơ ngắt
MAIN:      MOV   TMOD, #10H ; Chọn Timer1 chế độ 1
           MOV   P0, #0FFH  ; Chọn cổng P0 làm đầu vào nhận dữ liệu
           MOV   TL1, #018H ; Đặt TL1 = 18 byte thấp của - 1000
           MOV   TH1, #0FCH ; Đặt TH1 = FC byte cao của  - 1000
           MOV   IE, #88H   ; IE = 10001000 cho phép ngắt Timer1
           SETB  TR1        ; Khởi động bộ Timer1
 BACK:     MOV   A, P0      ; Nhận dữ liệu đầu vào ở cổng P0
           MOV   P1, A      ; Chuyển dữ liệu đến P1
           SJMP  BACK       ; Tiếp tục nhận và chuyển dữ liệu

 ; - - Trình ISR của Timer1 phải được nạp lại vì ở chế độ 1
 ISR_T1:         CLR   TR1   ; Dừng bộ Timer1
           CLR   P2.1        ; P2.1 = 0 bắt đầu xung mức thấp
           MOV   R2, #4      ; 2 chu kỳ máy MC (Machine Cycle)
 HERE:           DJNZ  R2, HERE   ; 4  2 MC = 8 MC
           MOV   TL1, #18H         ; Nạp lại byte thấp giá trị 2 MC
           MOV   TH1, #0FCH        ; Nạp lại byte cao giá trị 2 MC
           SETB  TR1               ; Khởi động Timer1    1 MC
           SETB  P2.1              ; P2.1 = 1 bật P2.1 trở lại cao
           RETI                    ; Trở về chương trình chính
           END
```

Lưu ý rằng phần xung mức thấp được tạo ra bởi 14 chu kỳ mức MC và mỗi MC = 1.085us và 14 x 1.085us = 15.19us.

Bài tập:

Viết một chương trình để tạo ra một sóng vuông tần số 50Hz trên chân P1.2. Giả sử XTAL = 11.0592MHz.

❖ **Thứ tự ưu tiên ngắt**

Khi có hai hay nhiều ngắt cùng lúc xảy ra, hoặc một ngắt đang thực hiện thì mô ngắt khác yêu cầu thì ngắt nào có độ ưu tiên hơn sẽ được ưu tiên xử lý.

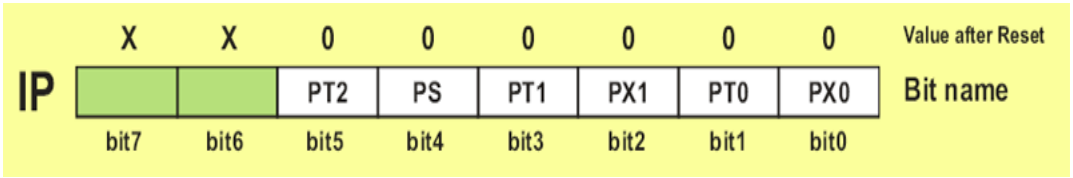
Có 3 cấp độ ưu tiên ngắt trong 8051

- Ngắt reset là ngắt có mức ưu tiên cao nhất, khi reset xảy ra tất cả các ngắt khác và chương trình đều bị dừng và vi điều khiển trở về chế độ khởi động ban đầu.
- Ngắt mức 1, chỉ có reset mới có thể cấm ngắt này
- Ngắt mức 0, các ngắt mức 1 và reset có thể cấm ngắt này.

Việc đăt chọn mức ưu tiên ngắt là 1 hoặc 0 thông qua thanh ghi IP. Việc xử lý ưu tiên ngắt của 8051 như sau:

- Nếu 1 có độ ưu tiên cao hơn một ngắt đang được xử lý xuất hiện thì, ngắt có ưu tiên thấp ngay lập tức bị dừng để ngắt kia được thực hiện
- Nếu 2 ngắt cùng yêu cầu vào 1 hời điểm thì ngắt có mức ưu tiên hơn sẽ được xử lý trước
- Nếu 2 ngắt có cùng mức ưu tiên cùng yêu cầu vào 1 thời điểm thì thứ tự được chọn như sau:
  - INTR 0
  - Timer 0
  - INTR 1
  - Timer 1
  - UART

Thanh ghi IP



Trong đó: Các bit từ 0 đến 5 đặt mức ngắt là 0 hoặc 1 cho các ngắt tương ứng như sau:

- PS: UART
- PT1: Timer 1
- PX1: INTR 1
- PT0: Timer 0
- PX0: INTR 0

3.7 Câu hỏi và bài tập cuối chương

Câu 1. Nêu các bước cấu hình cho timer 0 mode 1 sử dụng ngắt

Câu 2. Nêu các bước cấu hình cho timer 1 mode 1 sử dụng ngắt

Câu 3. Nêu các bước cấu hình cho counter 0 mode 1 sử dụng ngắt

Câu 4. Nêu các bước cấu hình cho counter 1 mode 2 sử dụng ngắt

Câu 5. Nêu các bước khởi tạo truyền thông nối tiếp

Câu 6. Nêu các bước khởi tạo ngắt ngoài 0 theo mức thấp

Câu 7. Nêu các bước khởi tạo ngắt ngoài 0 theo sườn xuống

Câu 8. Nêu các bước khởi tạo ngắt ngoài 1 theo mức thấp

Câu 9. Nêu các bước khởi tạo ngắt ngoài 1 theo sườn xuống

Câu 10. Tính giá trị TH, TL cho Timer 0, tràn sau mỗi 60µs, biết tần số thạch anh là 16Mhz

Câu 11. Tính giá trị TH, TL cho Timer 1, tràn sau mỗi 90µs, biết tần số thạch anh là 12Mhz

Câu 12. Tính giá trị TH, TL cho Timer 1, tràn sau mỗi 60ms, biết tần số thạch anh là 20Mhz

Câu 13. Tính giá trị TH, TL cho Timer 1, tràn sau mỗi 550µs, biết tần số thạch anh là 11.0592Mhz

Câu 14. Cho tần số thạch anh Fxtal= 8MHz, baud=9600bps, tính giá trị TH1

Câu 15. Cho tần số thạch anh Fxtal=10MHz, baud=9600bps, tính giá trị TH1

Câu 16. Cho tần số thạch anh Fxtal= 8MHz, baud=19200bps, tính giá trị TH1

Câu 17. Cho tần số thạch anh Fxtal=10MHz, baud=19200bps, tính giá trị TH1

Câu 18. Cho tần số thạch anh Fxtal= 8MHz, baud=19200bps (cấu hình nhân đôi tốc độ baud), tính giá trị TH1

Câu 19. Cho tần số thạch anh Fxtal=10MHz, baud=19200bps(cấu hình nhân đôi tốc độ baud), tính giá trị TH1

Câu 20. Viết chương trình mỗi khi bấm và giữ phím thì đèn LED nhấp nháy. Biết phím bấm tích cực mức 0, ghép vào chân P0.0, LED mắc cực dương vào P2.0, cực âm qua trở 280Ω xuống GND

Câu 21. Viết chương trình mỗi khi bấm và giữ phím thì đèn LED nhấp nháy. Biết phím bấm tích cực mức 0, ghép vào chân P0.1, LED mắc cực dương vào P2.1, cực âm qua trở 280Ω xuống GND

Câu 22. Viết chương trình liên tục nhấp nháy đèn LED, nếu bấm và giữ phím thì ngừng nhấp nháy LED. Biết phím bấm tích cực mức 0, ghép vào chân P0.0, LED mắc cực dương vào P2.3, cực âm qua trở 280Ω xuống GND

Câu 23. Viết chương trình liên tục nhấp nháy đèn LED, nếu bấm và giữ phím thì ngừng nhấp nháy LED. Biết phím bấm tích cực mức 0, ghép vào chân P1.0, LED mắc cực dương vào P2.5, cực âm qua trở 280Ω xuống GND

Câu 24. Viết chương trình con ngắt và khởi tạo ngắt Timer 0, mode 1, với tần số tràn là 200KHz, biết tần số thạch anh F_{xtal}=8MHz

Câu 25. Viết chương trình con ngắt và khởi tạo ngắt Timer 0, mode 1, với tần số tràn là 400KHz, biết tần số thạch anh F_{xtal}=11.0592MHz

Câu 26. Viết chương trình con ngắt và khởi tạo ngắt Timer 0, mode 2, với chu kỳ tràn là T=200µs, biết tần số thạch anh F_{xtal}=11.0592MHz

Câu 27. Viết chương trình con ngắt và khởi tạo ngắt Timer 1, mode 1, với tần số tràn là 200KHz, biết tần số thạch anh F_{xtal}=8MHz

Câu 28. Viết chương trình con ngắt và khởi tạo ngắt Timer 1, mode 1, với tần số tràn là 400KHz, biết tần số thạch anh F_{xtal}=11.0592MHz

Câu 29. Viết chương trình con ngắt và khởi tạo ngắt Timer 1, mode 2, với chu kỳ tràn là T=255µs, biết tần số thạch anh F_{xtal}=8MHz

Câu 30. Viết chương trình con ngắt và khởi tạo ngắt Timer 1, mode 2, với chu kỳ tràn là T=200µs, biết tần số thạch anh F_{xtal}=11.0592MHz

Câu 31. Viết đoạn lệnh khởi tạo truyền thông nối tiếp biết tần số thạch anh là 8MHz, tốc độ baud=9600bps.

Câu 32. Viết đoạn lệnh khởi tạo truyền thông nối tiếp biết tần số thạch anh là 16MHz, tốc độ baud=19200bps.

Câu 33. Viết đoạn lệnh khởi tạo truyền thông nối tiếp biết tần số thạch anh là 20MHz, tốc độ baud=19200bps.

Câu 34. Thiết kế và viết chương trình con đọc ma trận 2x2 nút bấm (nút bấm được đánh số từ 1 đến n), kết quả trả về là số thứ tự nút bấm, nếu không có nút nào được bấm, trả về 0. Biết nút bấm được ghép hàng vào P1, cột vào P2.

Câu 35. Thiết kế và lập trình hiển thị số 1234 ở 4 LED 7 thanh. Biết 4 LED là chung âm, mắc chung BUS dữ liệu (a..h).

Câu 36. Viết chương trình truyền liên tục tên mình lên máy tính qua đường RS232, với tốc độ baud = 9600bps

Câu 37. Hãy lập trình cho 8051 để nhận các byte dữ liệu nối tiếp và đặt chúng vào cổng P1. Đặt tốc độ baud là 4800bps, 8 bit dữ liệu và 1 bit Stop.

Câu 38. Hãy lập trình cho 8051 để nhận các byte dữ liệu nối tiếp và đặt chúng vào cổng P2. Đặt tốc độ baud là 9600bps, 8 bit dữ liệu và 1 bit Stop.

Câu 39. Viết chương trình truyền thông với máy tính, nếu máy tính gửi ký tự 'a' thì 8051 gửi trả về ký tự 'b', nếu máy tính gửi ký tự 'b' thì 8051 gửi trả về ký tự 'c',…

Câu 40. Viết chương trình truyền thông với máy tính nếu máy tính gửi xuống chữ 'Ten' thì 8051 gửi trả về tên mình (thí sinh).

Câu 41. Hãy viết chương trình nhận liên tục dữ liệu 8 bit ở cổng P0 và gửi nó đến cổng P1 trong khi nó cùng lúc tạo ra một sóng vuông chu kỳ 200µs trên chân P2.1. Hãy sử dụng bộ Timer0 để tạo ra sóng vuông, tần số của 8051 là F_{XTAL} =11.0592MHz.

Câu 42. Hãy viết chương trình nhận liên tục dữ liệu 8 bit ở cổng P0 và gửi nó đến cổng P1 trong khi nó cùng lúc tạo ra một sóng vuông với mức cao kéo dài

1085µs và mức thấp dài 15µs với giả thiết tần số F_{XTAL} = 11.0592MHz. Hãy sử dụng bộ định thời Timer1.

Câu 43. Hãy viết chương trình trong đó 8051 đọc dữ liệu từ cổng P1 và ghi nó tới cổng P2 liên tục trong khi đưa một bản sao dữ liệu tới cổng COM nối tiếp để thực hiện truyền nối tiếp giả thiết tần số XTAL là 11.0592MHz và tốc độ baud là 9600bps.

Câu 44. Hãy viết chương trình trong đó 8051 nhận dữ liệu từ cổng P1 và gửi liên tục đến cổng P2 trong khi đó dữ liệu đi vào từ cổng nối tiếp COM được gửi đến cổng P0. Biết tần số F_{XTAL}=11.0592MHz và tốc độ baud 9600bps.

Câu 45. Hãy viết một chương trình để thực hiện các công việc sau:

a. Nhận dữ liệu nối tiếp và gửi nó đến cổng P0.
b. Đọc dữ liệu từ cổng P1, truyền nối tiếp và sao chép đến cổng P2.
c. Sử dụng Timer0 tạo sóng vuông tần số 5kHz trên chân P0.1

giả thiết tần số XTAL = 11.0592MHz và tốc độ baud 4800.

(Trang này nên bỏ trống)

CHƯƠNG 4. ỨNG DỤNG

Mục tiêu

Giúp sinh viên học tập và thực hành theo các ví dụ mẫu, nhằm nâng cao trình độ lập trình của sinh viên.

Tóm tắt:

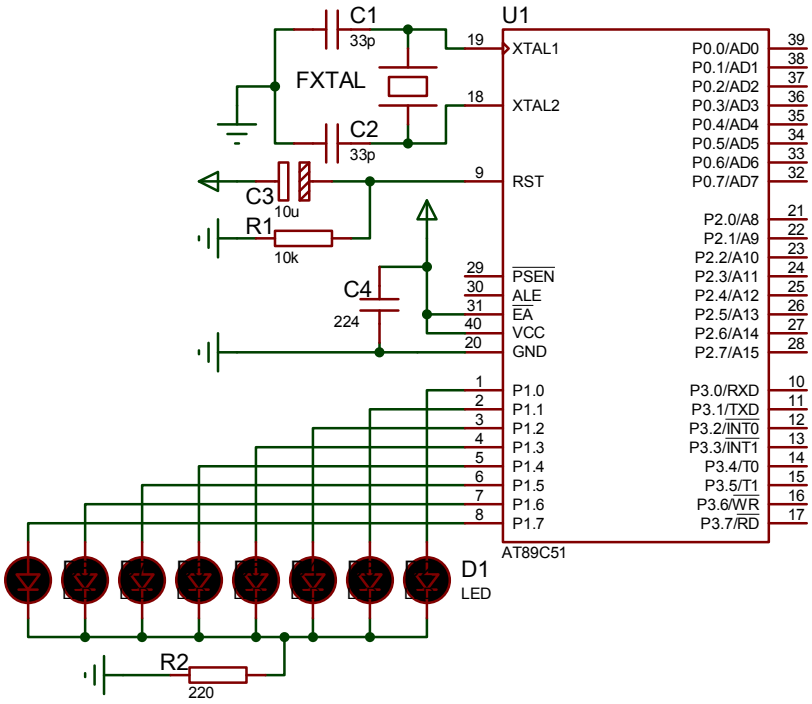
Ứng dụng các vi điều khiển đó để lập trình giao tiếp với thế giới thực thông qua các ví dụ:

- *Ghép nối vi điều khiển với hiển thị 7 thanh*
- *Ghép nối vi điều khiển với màn hình LCD*
- *Ghép nối vi điều khiển với bàn phím*
- *Ghép nối vi điều khiển với các bộ chuyển đổi ADC và DAC*
- *Ghép nối vi điều khiển với step motor*
- *...*

4.1 Nhấp nháy dãy LED đơn

Mục đích của ví dụ này không phải là để chứng minh hoạt động của đèn LED, nó dùng để biểu thị sự hoạt động của các vi điều khiển.

Sơ đồ chung của nhóm ví vụ với LED chúng ta dùng sơ đồ như sau:

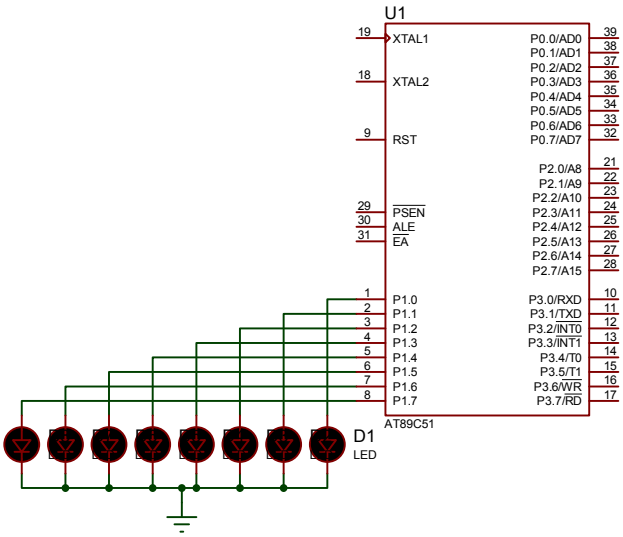


Hình 4-1. Mạch nhấp nháy LED đơn

Sơ đồ này là sơ đồ nguyên lý thực khi thiết kế mạch để chạy mạch in. Nhưng trong mô phỏng, chúng ta chỉ đơn giản là mô phỏng nguyên lý hoạt động của mạch, nên một số linh kiện có thể bỏ qua, vì chúng đã được phần mềm mô phỏng Proteus đặt mặc định rồi.

Mặc định, chúng ta sử dụng Fxtal=12MHz.

Cụ thể, sơ đồ mô phỏng chỉ cần như sau:



Hình 4-2. Mạch nhấp nháy LED đơn trong mô phỏng.

Một số hàm mẫu:

Trong tất cả các ví dụ về LED, chúng ta đều sử dụng một chương trình con (CTC) tạo trễ, thường đặt tên là Delay. Chương trình con Delay được viết bằng cách tạo ra nhiều vòng lặp lồng nhau, nhằm tiêu tốn thời gian. Khoảng thời gian bao nhiêu được tính dựa theo tần số thạch anh (Fxtal), số vòng lặp, số lần gọi CTC.

Chương trình con Delay có thời gian cố định:

```
Delay:mov R7, #10
   DL:mov R6,#255
     DL1:mov R5,#255
        DL2:djnz r5,dl2
     djnz R6,DL1
   DJNZ R7, DL
ret
```

Mã nguồn 4-1. Delay

Giá trị nạp vào R7, R6, R5 có thể được thay đổi tùy thời gan yêu cầu.

Chương trình con Delay có thời gian thay đổi tùy lúc gọi:

```
DelayX macro Tdelay
    local DL1,DL2,DL3
   push 7
   push 6
   push 5
          mov    R7,#Tdelay
      DL1:mov    R6,#100
      DL2:mov    R5,#100
      DL3:djnz   R5, Dl3
         djnz   R6, Dl2
         djnz   R7, Dl1
   pop 5
   pop 6
   pop 7
endm
```

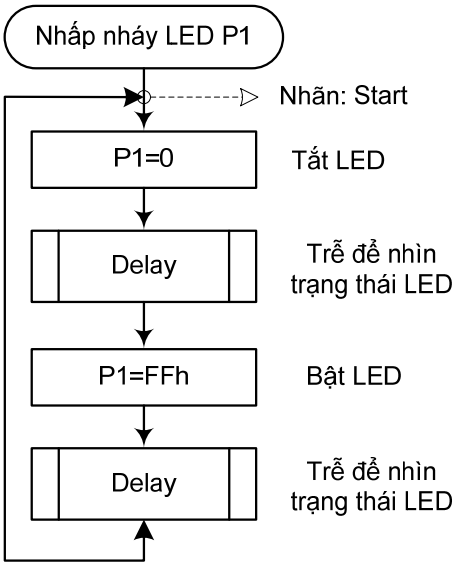
Mã nguồn 4-2. DelayX

Ý tưởng là viết CTC trong một MACRO, truyền tham số thời gian trễ vào tham số tại lúc gọi hàm. Như vậy, có thể với mỗi lần gọi hàm khác nhau, chương trình con Delay sẽ được truyền thời gian khác nhau, và khoảng thời gian trễ là như nhau.

Nhấp nháy cả cổng P1:

Muốn LED nhấp nháy trên cổng P1, chúng ta tắt LED, trễ rồi bật LED, trễ.

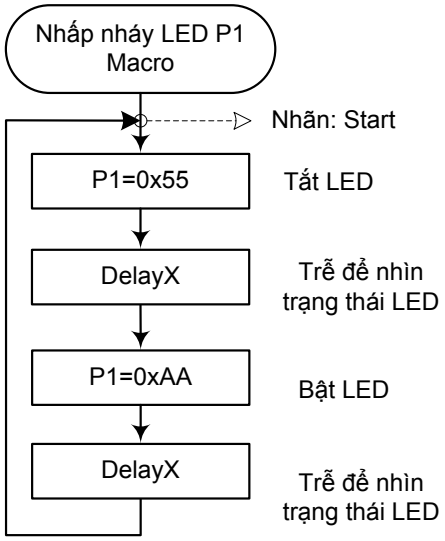
Sơ đồ như *"Hình 4-1. Mạch nhấp nháy LED đơn"*, sơ đồ thuật toán và mã nguồn như hình dưới đây:



Hình 4-3. Thuật toán: Nhấp nháy P1

```
org 0
start:
   mov P1,#0x00
      call delay
   mov P1,#0xff
      call delay
jmp start
delay:mov R7, #10
   DL:mov R6,#255
     DL1:
          mov R5,#255
        DL2:djnz r5,dl2
      djnz R6,DL1
   DJNZ R7, DL
ret
end
```

Mã nguồn 4-3. Nhấp nháy cổng P1



Hình 4-4. Thuật toán: Nhấp nháy P1-Macro

```
org 0
jmp main
  // Khai báo DelayX tại đây
  // Xem: "Mã nguồn 4-2. DelayX"
  sbit   L0=   P2.0
main:
   mov P1, #0x55
   cpl L0
      delay 10
   mov P1, #0xaa
   cpl L0
      delay 10
jmp main
end
```

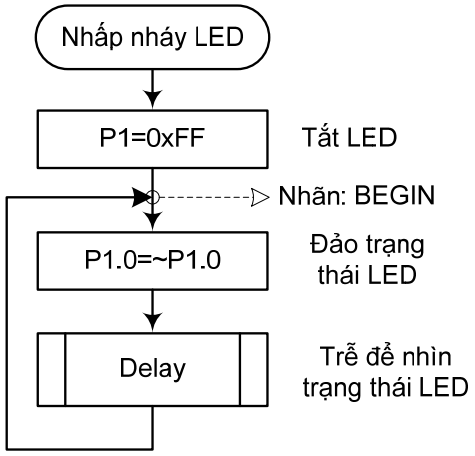
Mã nguồn 4-4. Nhấp nháy cổng P1và đảo trạng thái P2.0

Trong ví dụ trên, có thêm phần đảo LED, xem kỹ đảo LED ở "Hình 4-5. Thuật toán: Nhấp nháy P1.0"

Nhấp nháy một LED đơn:

Đơn giản chỉ cần kích hoạt đèn LED nhấp nháy để nhìn thấy được sự hoạt động, trong mỗi lần thay đổi trạng thái của LED, cần tạo một khoảng thời gian trễ để có thể quan sát thấy trạng thái. Trong ví dụ này, thời gian trễ được cung cấp bằng cách thực

hiện một chương trình con trễ, được gọi là Delay. Nó là 3 vòng lặp lồng nhau sử dụng thanh ghi R0, R1 và R2. Sau khi trở về từ các chương trình con, trạng thái của chân được đảo ngược và các thủ tục tương tự được lặp đi lặp lại ...




```
ORG 0
    JMP BEGIN ;Reset vector
ORG 100H
;Khởi tạo trạng thái:
  MOV P1,#0FFh
BEGIN:
  CPL P1.0 ;Đảo trạng thái P1.0
  LCALL   Delay   ;Time delay
  SJMP    BEGIN
Delay:
  MOV R2,#20  ;500 ms time delay
F02: MOV     R1,#50  ;25 ms
F01: MOV     R0,#230
     DJNZ    R0,$
     DJNZ    R1,F01
     DJNZ    R2,F02
Ret
END    ;End of program
```

Hình 4-5. Thuật toán: Nhấp nháy P1.0 *Mã nguồn 4-5. Nhấp nháy P1.0*

4.2 Timer

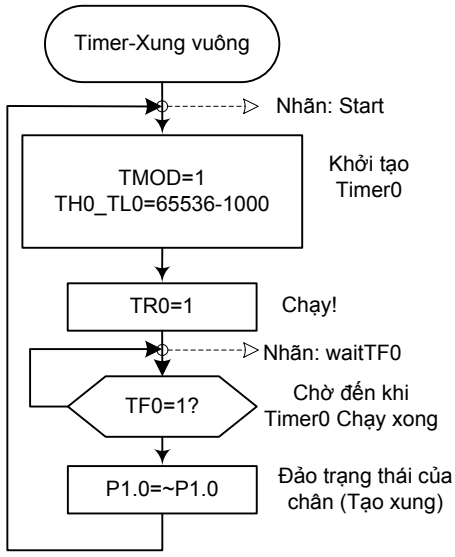
Chương trình dưới đây minh họa một ví dụ đơn giản nhất về Timer, lấy ví dụ là Timer 0 (vì trong chế độ cơ bản, Timer 0 và Timer 1 là như nhau).

Sơ đồ nguyên lý, vẫn lấy sơ đồ trong "Hình 4-1. Mạch nhấp nháy LED đơn".

Thuật toán lập trình và mã nguồn có thể thực hiện như các ví dụ dưới đây.

♦ **Timer – Bài toán 1:**

Liên tục phát xung vuông có chu kỳ là 2ms ra chân P1.0




```
CSEG AT 0
JMP     Start  ; Reset vector
ORG     100H
Start:
   MOV     TMOD,#0x01
   MOV          TH0, #HIGH(-1000);1ms
   MOV          TL0, #LOW(-1000)
  SETB    TR0  ; Cho TIMER chay

  waitTF0: jnb TF0, waitTF0

   CPL          P1.0
   JMP          Start
END
```

Hình 4-6. Thuật toán: TIMER0 *Mã nguồn 4-6. Timer0 tạo xung PWM*

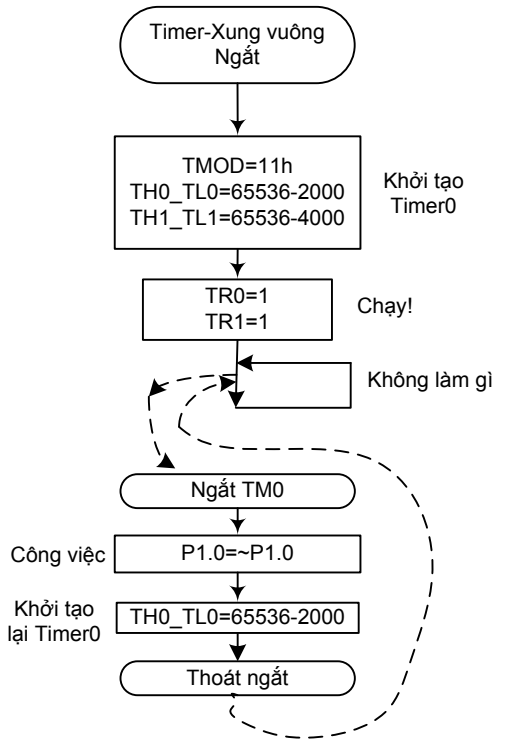
♦ **Timer – Bài toán 2:**

Sử dụng Timer0 và Timer1, Timer0 dùng để phát xung vuông có chu kỳ 4ms tại chân P1.0, Timer 1 phát xung vuông 8ms ở chân P1.7.

Hình vẫn dùng ở: "Hình 4-1. Mạch nhấp nháy LED đơn"

Phân tích:

- Với yêu cầu đề bài như trên, ta sử dụng ngắt timer0 và ngắt timer1. Chương trình chính khởi tạo timer, khởi tạo ngắt rồi không làm gì nữa.
- Cả hai CTC ngắt có vai trò như nhau. Trong mỗi ngắt, thực hiện nhiệm vụ là lật trạng thái chân (để tạo xung vuông), và khởi tạo lại giá trị timer tương ứng.



Hình 4-7. Thuật toán: Ngắt Timer 0 và Timer1

(Thuật toán TM0 và TM1 là tương đương)

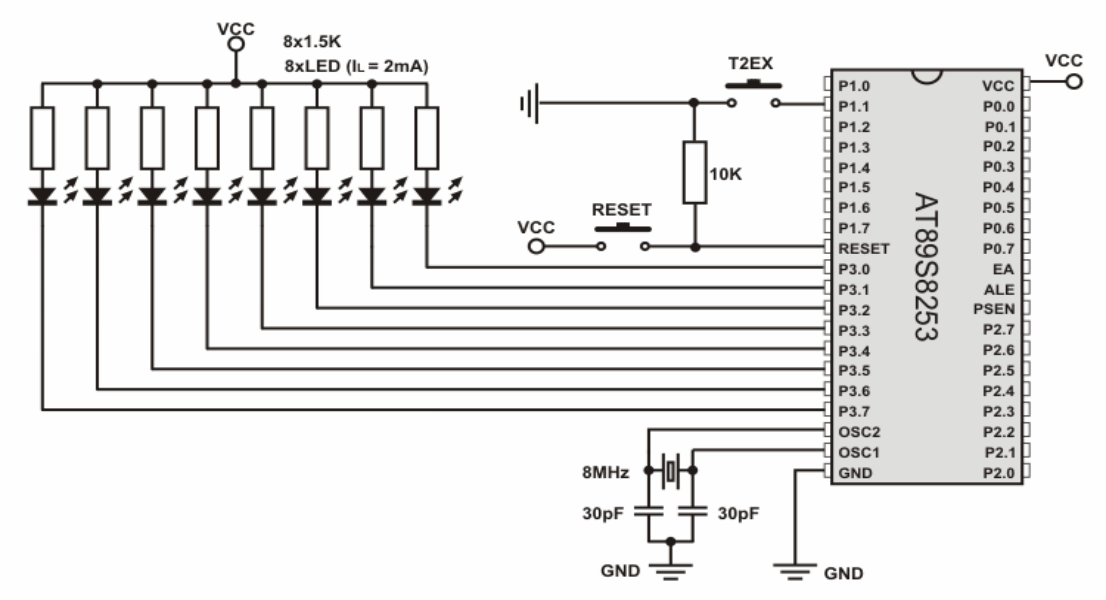
```
CSEG    AT    0
JMP     Start  ; Reset vector
   ORG 0BH
     JMP TM0_PWM ;Vector ngắt TM0
    ORG     01BH
     JMP TM1_PWM ;Vector ngắt TM1
    ORG     100H
Start:
   MOV   TMOD,#0x11
   MOV     TH0,#HIGH(-2000)
   MOV     TL0,#LOW(-2000)
   MOV     TH1,#HIGH(-4000)
   MOV     TL1,#LOW(-4000)
   MOV IE,#08AH ; Interrupt enabled

        SETB TR0  ; Cho TIMER0 chay
        SETB TR1  ; Cho TIMER1 chay
        JMP     $
TM0_PWM:
      CPL   P1.0
      MOV   TH0,#HIGH(-2000)
      MOV   TL0,#LOW(-2000)
    RETI ; RETurn from Interrupt
TM1_PWM:
      CPL   P1.7
      MOV   TH1,#HIGH(-4000)
      MOV   TL1,#LOW(-4000)
    RETI ; RETurn from Interrupt
END ; End of program
```

Mã nguồn 4-7. Timer0 và Timer1 tạo xung PWM dùng ngắt.

4.3 Sử dụng Timer T2

Ví dụ này mô tả việc cấu hình Timer T2 (chỉ có trong các dòng VĐK 8052) sử dụng để hoạt động ở chế độ tự động cập nhật. Trong trường hợp này, đèn LED được kết nối với cổng P3 trong khi các nút nhấn được sử dụng để bắt buộc thiết lập lại bộ đếm thời gian (T2EX) được kết nối với chân P1.1.

Khi kết thúc giờ đếm, ngắt Timer tràn được kích hoạt và chương trình con TIM2_ISR được thi hành, sau đó, quay thanh ghi A rồi đưa ra cổng P3. Cuối cùng, xóa ngắt và trở về nơi đã gọi nó.



Hình 4-8. Sử dụng Timer 2

Nếu bấm T2EX, bộ đếm thời gian tạm thời đặt lại. Nút bấm này reset lại timer, trong khi nút nhấn RESET sẽ Reset lại vi điều khiển.

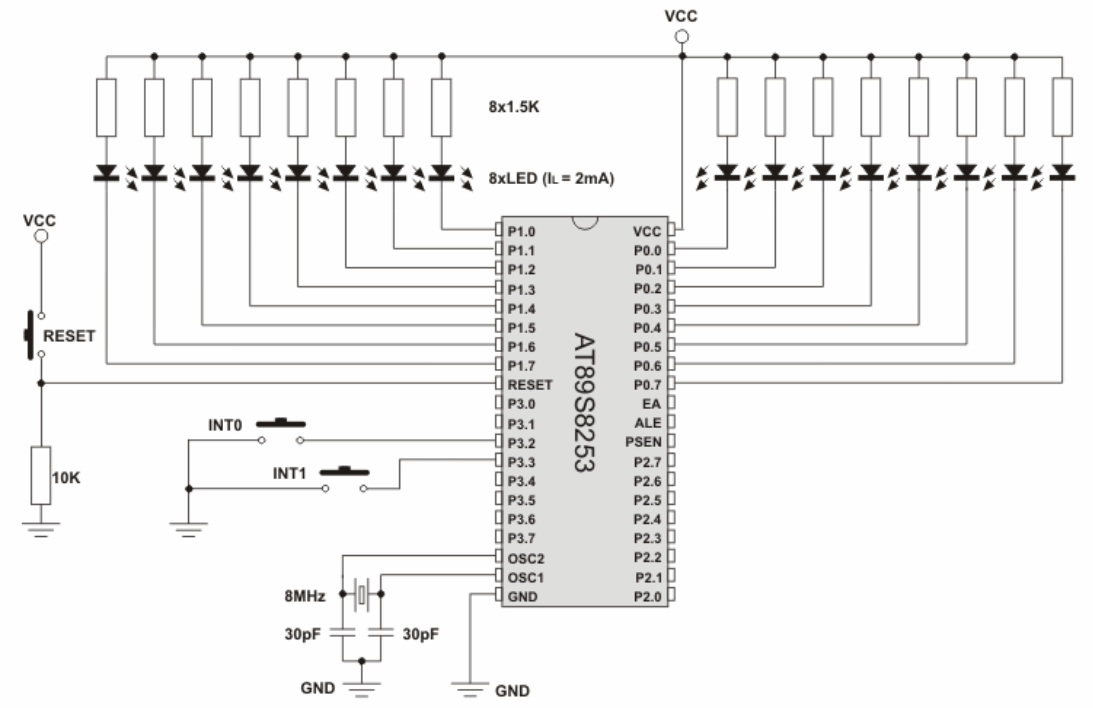
```
CSEG     AT       0
JMP      XRESET              ; Reset vector
         ORG      02BH        ; Vector ngắt của TM2
         JMP      TIM2_ISR
ORG      100H
XRESET:
         MOV      A,#0FFH
         MOV      P3,#0FFH
         MOV      RCAP2L,#0FH ; TM2, 16-bit tự nạp lại
         MOV      RCAP2L,#01H
         CLR      CAP2        ; Cho phép 16-bit tự nạp lại
         SETB     EXEN2       ; Khởi tạo nút bấm
         SETB     TR2         ; Cho TM2 chạy
         MOV      IE,#0A0H    ; Cho phép ngắt TM2
         CLR      C
```

```
LOOP1:  SJMP     LOOP1        ; Chạy tại chỗ

TIM2_ISR:
      RRC A       ; Quay A qua cờ C
      MOV       P3,A  ; Xuất A ra cổng P3
      CLR       TF2   ; Xóa cờ ngắt
         CLR       EXF2  ; Xóa cờ ngắt
RETI     ; Kết thúc ngắt
END  ; Kết thúc chương trình
```

Mã nguồn 4-8. Sử dụng Timer 2

4.4 Dùng ngắt ngoài.

Dưới đây là một ví dụ khác của sự thực thi ngắt. Một ngắt ngoài được tạo ra khi một logic không (0) xảy ra trên chân P3.2 hoặc chân P3.3. Tùy thuộc vào đó là đầu vào hoạt động nào, một trong hai công việc sẽ được thực thi:



Hình 4-9. Lập trình 2 ngắt ngoài

Một logic số mức không (0) trên P3.2 khởi tạo sự thực thi ngắt Isr_Int0, vì thế số tăng dần trong R0 được sao chép ra cổng P0. Logic số mức không trên P3.3 khởi tạo sự thực thi chương trình con ngắt Isr_Int1, số tăng dần trong R1 được sao chép sang P1.

Trong ngắn hạn, mỗi lần bấm vào các nút nhấn INT0 và INT1 sẽ được tính và ngay lập tức được hiển thị ở định dạng nhị phân trên cổng thích hợp

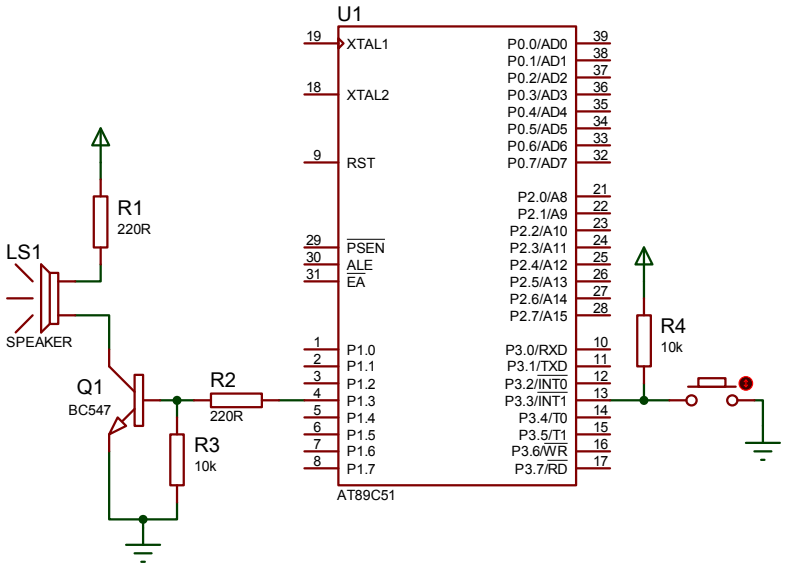
```
        CSEG     AT     0
        JMP      XRESET; Reset vector
        ORG      003H   ; Vector ngắt INT0
        JMP      Isr_Int0
        ORG 013H        ; Vector ngắt INT1
        JMP      Isr_Int1
ORG      100H
XRESET:
  MOV TCON,#00000101B  ; Ngắt INT0 (P3.2)và ngắt INT1 (P3.3) xảy
```

```
                        ;  ra khi có thay đổi mức từ 1 xuống 0
  MOV       IE,#10000101B    ; Cho phép ngắt
  MOV       R0,#00H    ; Khởi tạo
  MOV       R1,#00H    ;
  MOV       P0,#00H    ;
  MOV       P1,#00H    ;
LOOP:   SJMP      LOOP ;Chạy tại chỗ
Isr_Int0:
        INC R0     ; Tăng R0 rồi xuất ra P0
        MOV P0,R0
   RETI
Isr_Int1:
        INC R1     ; Tăng R1 rồi xuất ra P1
        MOV P1,R1
   RETI
END ; End of program
```

Mã nguồn 4-9. Lập trình 2 ngắt ngoài

4.5 Lập trình ngắt ngoài theo sườn xuống.

Phát hiện nếu có sườn xuống tại chân ngắt ngoài 1 (INT1, P3.3) thì sinh ngắt, Trong CTC ngắt, bật loa kêu một lúc rồi tắt.



Hình 4-10. Lập trình ngắt ngoài – bật loa

```
ORG     0000H
LJMP    MAIN
     ORG     0013H           ;INT1 ISR
          MOV   R7, #50 ; Thời gian loa kêu
          Speak:
               SETB    P1.3            ;Bật loa
               MOV     R3,#255
               BACK:   DJNZ    R3,BACK ;Trễ 1 chút
               CLR     P1.3            ;Tắt loa
               MOV     R3,#255
               BACK1:   DJNZ    R3,BACK1 ;Trễ 1 chút
```

```
            DJNZ R7, Speak
      RETI                    ;
;Chương trình chính:
ORG     30H
MAIN:   SETB    TCON.2   ;Cho ngắt cạnh xuống
      MOV   IE,#10000100B  ;Cho phép ngắt ngoài
      HERE: SJMP    HERE      ;Chạy tại chỗ.
;Nếu có ngắt thì thực hiện ngắt, thực hiện xong lại về đây
END
```

Mã nguồn 4-10. Lập trình ngắt ngoài – bật loa

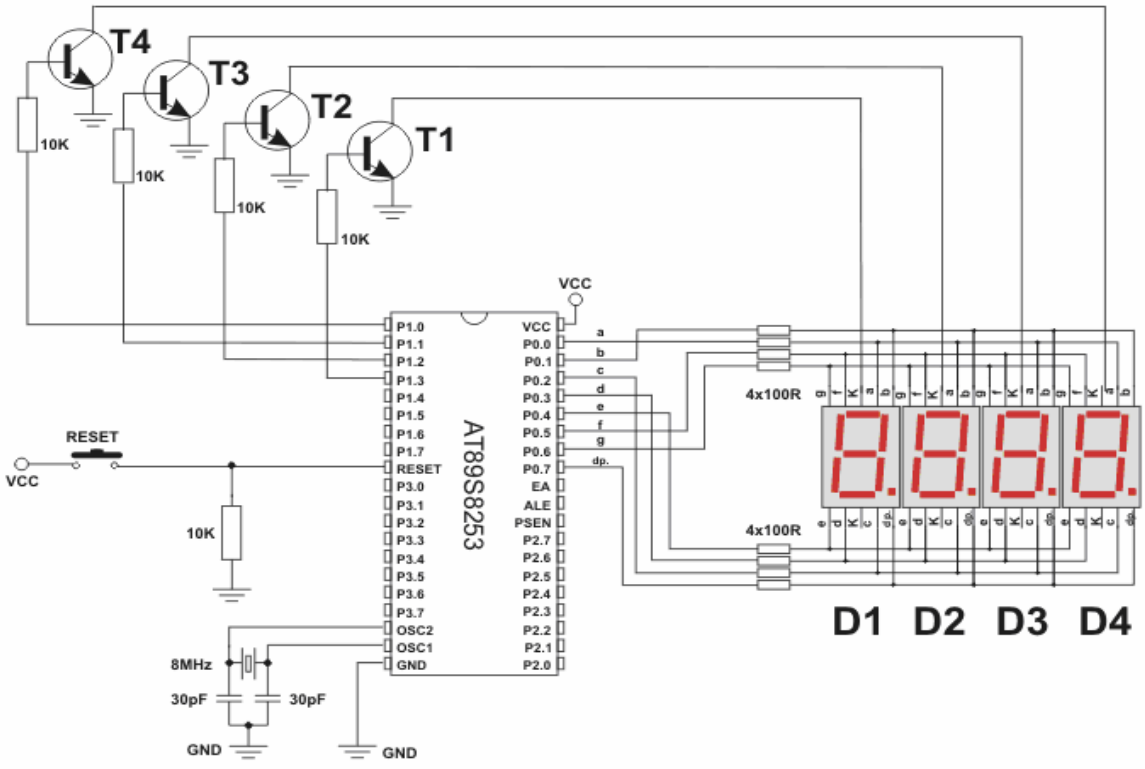
4.6 Sử dụng LED 7 thanh

Các ví dụ sau đây mô tả việc sử dụng đèn LED hiển thị 7 thanh. Để tiết kiệm chân I/O, bốn LED hiển thị được kết nối để hoạt động ở chế độ multiplex. Nó có nghĩa là tất cả các đoạn có cùng tên được kết nối với một cổng ra, và chỉ có một màn hình LED hoạt động tại một thời điểm, ta gọi là quét LED.

Các tranzistor và LED trên màn hình này luân phiên sáng trong khoảng thời gian ngắn, do đó làm cho ta tưởng rằng tất cả các chữ số đang hiển thị đồng thời.

4.6.1 Hiển thị số trên 1 LED 7 thanh

Chương trình này là một loại bài tập "Khởi động" trước khi làm việc thực tế. Mục đích của ví dụ này là để hiển thị trên màn hình bất cứ điều gì đó. Ta sử dụng 1 LED trong màn hình để hiển thị các số từ 0-9 thông qua mặt nạ số trong chương trình con hiển thị.



Hình 4-11. Hiển thị LED 7 thanh

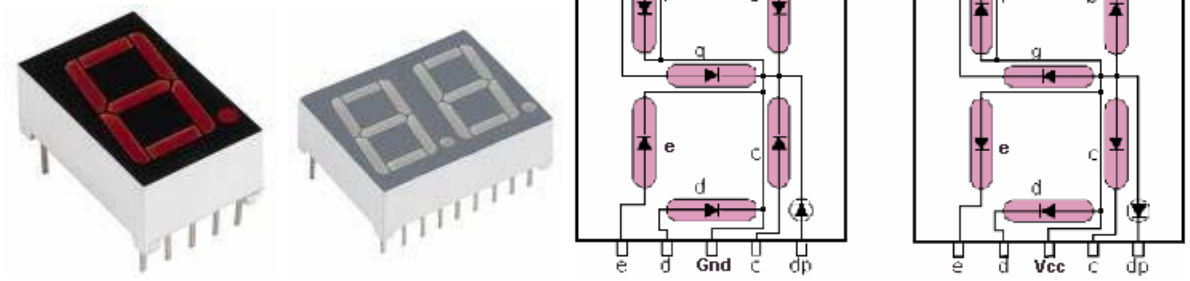
```
;*****************************************************************
;* PROGRAM NAME : 7Seg1.ASM
;* DESCRIPTION: Program displays number "3" on 7-segment LED
display
;*****************************************************************
          CSEG     AT     0
          JMP      XRESET ; Reset vector
          ORG      100H
XRESET:
          MOV      P1,#0   ; Tắt LED
          MOV      P3,#20h ; Chọn LED D4 để hiển thị
LOOP:
          MOV      A,#03   ; Hiển thị số 3
          LCALL    Disp    ; Thông qua mặt nạ trong CTC Disp
          MOV      P1,A
          SJMP     LOOP
Disp:   ;CTC hiển thị số
          INC      A
          MOVC     A,@A+PC
    RET
          DB       3FH   ; Mặt nạ số 0
          DB       06H   ; Mặt nạ số 1
          DB       5BH   ; Mặt nạ số 2
          DB       4FH   ; Mặt nạ số 3
          DB       66H  ; Mặt nạ số 4
          DB       6DH  ; Mặt nạ số 5
          DB       7DH  ; Mặt nạ số 6
          DB       07H  ; Mặt nạ số 7
          DB       7FH  ; Mặt nạ số 8
          DB       6FH  ; Mặt nạ số 9
END    ; Kết thúc chương trình
```

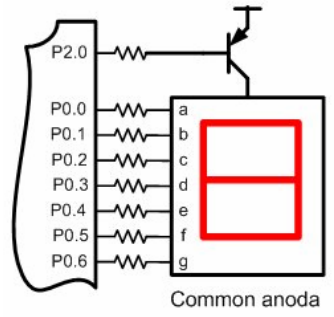
Mã nguồn 4-11. Hiển thị LED 7 thanh -1

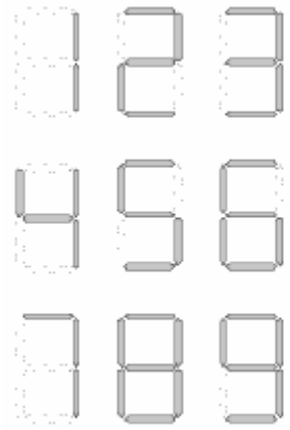
4.6.2 Hiển thị trên nhiều LED 7 thanh

Led 7 thanh được ứng dụng khá phổ biến khi cần hiển thị số tự nhiên hoặc vài chữ cái nhất định. Led 7 thanh có thể có kích thước lớn nhỏ khác nhau, màu sắc khác nhau nhưng về hình dáng cơ bản như hình dưới đây

Led 7 thanh bao gồm nhiều led tích hợp bên trong, các led được nối chung nhau 1 chân. Trong thực tế có 2 loại led 7 thanh là led 7 thanh A-nốt chung và led 7 thanh Ka-tốt chung. Led loại A-nốt chung, các led sẽ có chung nhau chân nguồn (chân dương), chân còn lại của led nào được nối đất thì led đó sẽ sáng. Led loại Ka-tốt chung, các led sẽ nối chung nhau chân đất (chân âm), chân còn lại của led nào được nối nguồn thì led đó sẽ sáng.







Digit Shown	Illuminated Segment (1 = illumination)						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	0	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Hình 4-12.Sơ đồ chân LED 7 thanh

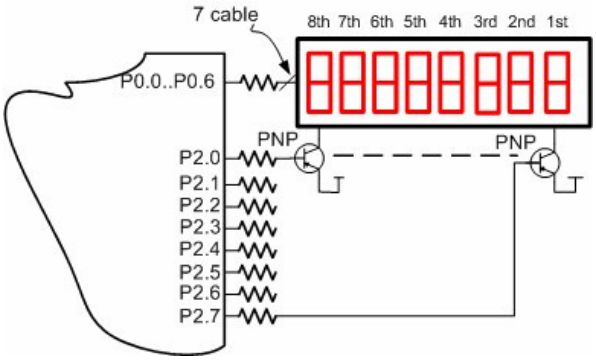
Ví dụ:

```
org 0h
start:
      mov P0,#11111100b; Cấp 0V cho thanh led a và b
      clr P2.0 ; Cấp 5V cho led 7 thanh
      call delay ; Gọi hàm trễ

      mov P0,#11011011b; Cấp 0V cho thanh led c,f
      clr P2.0 ; Cấp 5V cho led
      call delay ; Gọi hàm trễ

      mov P0,#10110000b; Cấp 0V to a,b,c,d,g
      clr P2.0 ; Cấp 5V cho led

      call delay ; Gọi hàm trễ
      sjmp start ; Trở về đầu chương trình
```

```
;===========================================
;subroutine delay created to rise delay time
;===========================================
      delay: mov R1,#255
      del1: mov R2,#255
      del2: djnz R2,del2
      djnz R1,del1
      ret
      end
```

Mã nguồn 4-12. Hiển thị LED 7 thanh - 2

Ví dụ điều khiển nhiều LED 7 thanh:



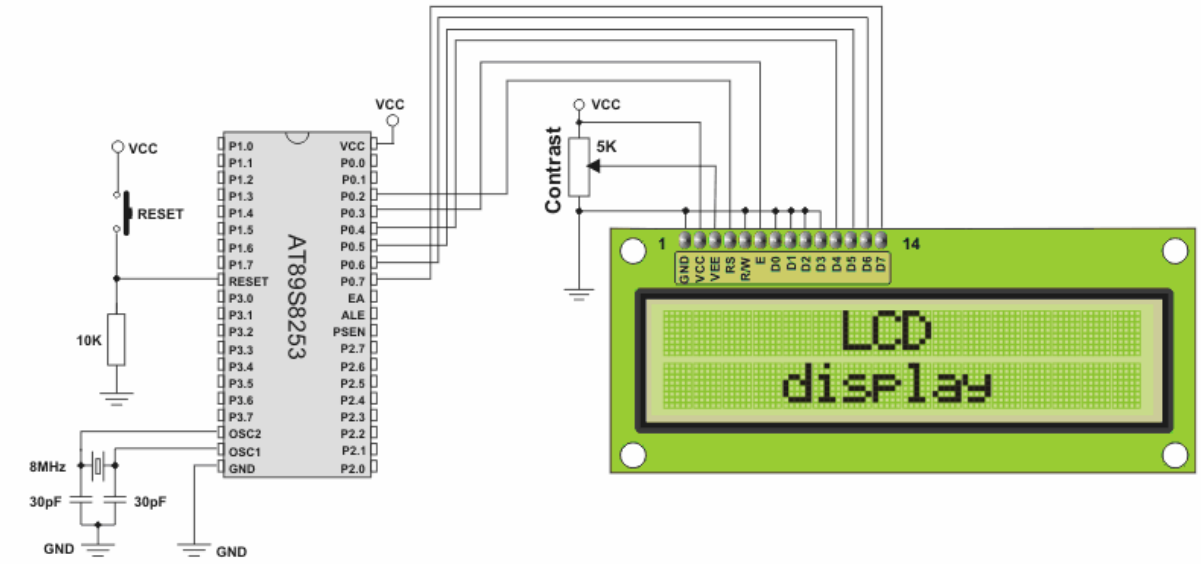

```
org 0h
start:
      mov   dptr, #word      ;để con trỏ dữ liệu vào đầu bảng
      mov   R6,   #8    ; số led cần hiển thị, 8 led
      mov   R1,   #01111111b; khởi đầu ở led 8
Again:
      clr   A ; xóa thanh ghi acc
      movc  A,    @A+dptr ; đưa số đầu tiên ở bảng vào acc
      inc   dptr        ; tăng vị trí con trỏ
      mov   P0,   A     ; đưa mã cần hiển thị ra P0
      mov   A,    R1    ; thứ tự led cần hiển thị
      mov   P2,   A     ; hiển thị led
      rr    A           ; dịch vi trí led cần hiển thị
      mov   R1,   A     ; lưu vào thanh ghi R1
      call  delay       ; gọi hàm trễ
      mov   P0,   #11111111b; xóa
      djnz  R6,   Again       ; lặp lại 8 lần
sjmp start                      ; trở về vị trí ban đầu
delay: mov R1,#255
      del1: mov R2,#255
      del2: djnz R2,del2
      djnz R1,del1
ret
word: DB 00111111b, 01000111b, 00001000b, 00000011b
      DB 01000110b, 01000000b, 01001000b, 00111111b
end
```

Mã nguồn 4-13. Hiển thị trên nhiều LED 7 thanh

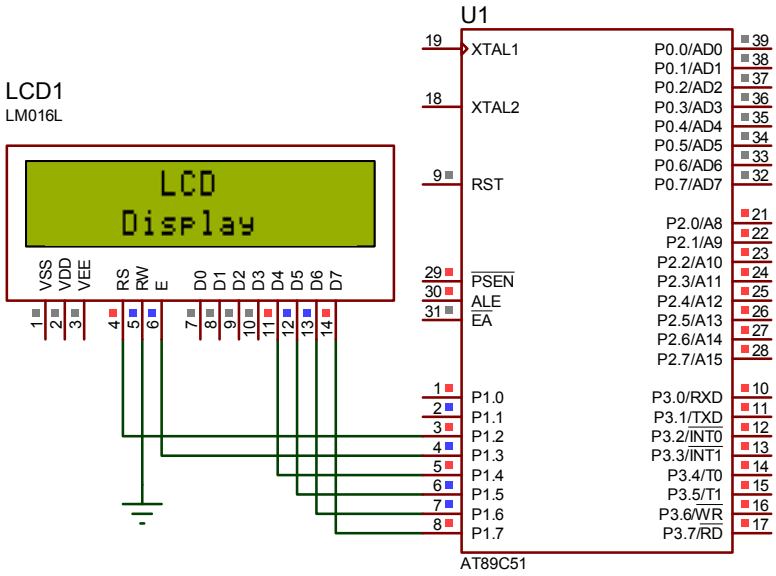
4.7 Thông báo bằng văn bản trên màn hình LCD

Ví dụ này sử dụng loại LCD phổ biến nhất để hiển thị văn bản trong hai dòng với 16 ký tự mỗi dòng. Để tiết kiệm chân IO của vi điều khiển, chỉ có 4 chân được sử dụng cho giao tiếp dữ liệu. Bằng cách này, mỗi byte được truyền đi theo hai bước: đầu tiên là 4 bit cao sau đó là 4 bit thấp.

LCD cần phải được khởi tạo tại đầu chương trình (trước khi sử dụng các tính năng ghi đọc LCD). Bên cạnh đó, các phần của chương trình lặp đi lặp lại trong chương trình tạo ra một chương trình con đặc biệt. Tất cả điều này có vẻ rất phức tạp, nhưng toàn bộ chương trình về cơ bản thực hiện một số hoạt động đơn giản và hiển thị dòng chữ **"LCD display"**.



Hình 4-13. Sơ đồ hiển thị LCD thực



Hình 4-14. Sơ đồ hiển thị LCD mô phỏng

```
;*****************************************************************
;* PROGRAM NAME : Lcd.ASM
;* DESCRIPRTION : Program for testing LCD display. 4-bit
communication
;* is used. Program does not check BUSY flag but uses program delay
;* between 2 commands. PORT1 is used for connection
;* to the microcontroller.
;*****************************************************************

Start_address      EQU    0000h
          CSEG     AT     0
          ORG      Start_address
          JMP      Inic
LCD_Disp MACRO TS
     MOV      A,#TS              ; Display character ' '.
     CALL     LCD_putc
ENDM
          ORG      Start_address+100h
          MOV      IE,#00      ; All interrupts are disabled
Inic:     CALL     LCD_inic    ; Initialize LCD
;************************************************
;* MAIN PROGRAM
;************************************************
START:    MOV      A,#80h ; Hiển thị tại dòng 1 cột 1
     CALL     LCD_status
     LCD_Disp ' '
     LCD_Disp ' '
     LCD_Disp ' '
     LCD_Disp ' '
     LCD_Disp ' '
     LCD_Disp ' '
     LCD_Disp 'L'
     LCD_Disp 'C'
     LCD_Disp 'D'
     MOV      A,#0c0h   ; Hiển thị tại dòng 2 cột 1
     CALL     LCD_status
     LCD_Disp ' '
     LCD_Disp ' '
     LCD_Disp ' '
     LCD_Disp ' '
     LCD_Disp 'D'
     LCD_Disp 'i'
     LCD_Disp 's'
     LCD_Disp 'p'
     LCD_Disp 'l'
     LCD_Disp 'a'
     LCD_Disp 'y'
```

```
      MOV       R0,#20d   ; Chờ một tí(20x10ms)
      CALL      Delay_10ms
      MOV       DPTR,#LCD_DB  ; Xóa màn hình
      MOV       A,#6d
      CALL      LCD_inic_status
      MOV       R0,#10d    ; Chờ một tí 10x10ms)
      CALL      Delay_10ms
JMP       START

;*******************************************
;* Chương trình con tạo trễ (T= r0 x 10ms)
;*******************************************
Delay_10ms: MOV   R5,00h    ;T.gian trễ ~ 1+(1+(1+2*r7+2)*r6+2)*r5
         MOV   R6,#100d         ; (nếu r7>10)
         MOV   R7,#100d         ; 2*r5*r6*r7
         DJNZ  R7,$
         DJNZ  R6,$-4
         DJNZ  R5,$-6
    RET
;*************************************************************
; Chương trình con khởi tạo:
;*************************************************************

LCD_enable      BIT    P1.3    ; Bit for activating pin E on LCD.
LCD_read_write BIT    P1.1    ; Bit for activating pin RW on LCD.
LCD_reg_select BIT    P1.2    ; Bit for activating pin RS on LCD.
LCD_port        SET    P1      ; Port for connection to LCD.
Busy          BIT    P1.7    ; Port pin on which Busy flag appears.

LCD_Start_I_red  EQU   00h    ; Address of the first message
character
                      ; in the first line of LCD display.
LCD_Start_II_red EQU   40h    ; Address of the first message
character
                      ; in the second line of LCD display.

LCD_DB:          DB     00111100b    ; 0 -8b, 2/1 lines, 5x10/5x7
format
           DB     00101100b    ; 1 -4b, 2/1 lines, 5x10/5x7 format
           DB     00011000b    ; 2 -Display/cursor shift,
right/left
           DB     00001100b    ; 3 -Display ON, cursor OFF,
                               ;cursor blink off
           DB     00000110b    ; 4 -Increment mode, display shift
off
           DB     00000010b    ; 5 -Display/cursor home
           DB     00000001b    ; 6 -Clear display
```

```
          DB     00001000b    ; 7 -Display OFF, cursor OFF,
                              ;   cursor blink off
LCD_inic:
     MOV    DPTR,#LCD_DB
     MOV    A,#00d           ; Triple initialization in 8-bit
     CALL   LCD_inic_status_8  ; mode is performed at the beginning
     MOV    A,#00d             ; (in case of slow increment of
     CALL   LCD_inic_status_8; the power supply is on
     MOV    A,#00d
     lcall LCD_inic_status_8

     MOV    A,#1d            ; Change from 8-bit into
     CALL   LCD_inic_status_8 ; 4-bit mode
     MOV    A,#1d
     CALL   LCD_inic_status
     MOV    A,#3d   ; As from this point the program executes in
                    ;4-bit mode
     CALL   LCD_inic_status
     MOV    A,#6d
     CALL   LCD_inic_status
     MOV    A,#4d
     CALL   LCD_inic_status
RET

;*****************************************
LCD_inic_status_8:
     PUSH   B
     MOVC   A,@A+DPTR
     CLR    LCD_reg_select    ; RS=0 - Write command
     CLR    LCD_read_write    ; R/W=0 - Write data on LCD

     MOV    B,LCD_port  ; Lower 4 bits from LCD port are memorized
     ORL    B,#11110000b
     ORL    A,#00001111b
     ANL    A,B
     MOV    LCD_port,A        ; Data is moved from A to LCD port
     SETB   LCD_enable        ; high-to-low transition signal
                         ; is generated on the LCD's EN pin
     CLR    LCD_enable
     MOV    B,#255d     ; Time delay in case of improper reset
     DJNZ   B,$         ; during initialization
     DJNZ B,$
     DJNZ B,$
     POP B
     RET
;*****************************************************
LCD_inic_status:
     MOVC   A,@A+DPTR
```

```
      CALL  LCD_status
RET
;*********************************************************
;* SUBROUTINE: LCD_status
;* DESCRIPTION: Subroutine for defining LCD status.
;*********************************************************
LCD_status:    PUSH  B
      MOV   B,#255d
      DJNZ  B,$
      DJNZ  B,$
      DJNZ  B,$
      CLR   LCD_reg_select   ; RS=0: Command is sent to LCD
      CALL  LCD_port_out
      SWAP  A               ; Nibles are swapped in accumulator
      DJNZ  B,$
      DJNZ  B,$
      DJNZ  B,$
      CLR   LCD_reg_select   ; RS=0: Command is sent to LCD
      CALL  LCD_port_out
      POP   B
RET
;*********************************************************
;* SUBROUTINE: LCD_putc
;* DESCRIPTION: Sending character to be displayed on LCD.
;*********************************************************
LCD_putc:      PUSH  B
      MOV   B,#255d
      DJNZ  B,$
      SETB  LCD_reg_select   ; RS=1: Character is sent to LCD
      CALL  LCD_port_out
      SWAP  A               ; Nibles are swapped in accumulator
      DJNZ  B,$
      SETB  LCD_reg_select   ; RS=1: Character is sent to LCD
      CALL  LCD_port_out
      POP   B
RET
;*************************************************
;* SUBROUTINE: LCD_port_out
;* DESCRIPTION: Sending commands or characters on LCD display
;*************************************************
LCD_port_out:  PUSH  ACC
      PUSH  B
      MOV   B,LCD_port
      ORL   B,#11110000b
      ORL   A,#00001111b
      ANL   A,B
      MOV   LCD_port,A        ; Data is copied from A to LCD port
      SETB  LCD_enable        ; high-to-low transition signal
```

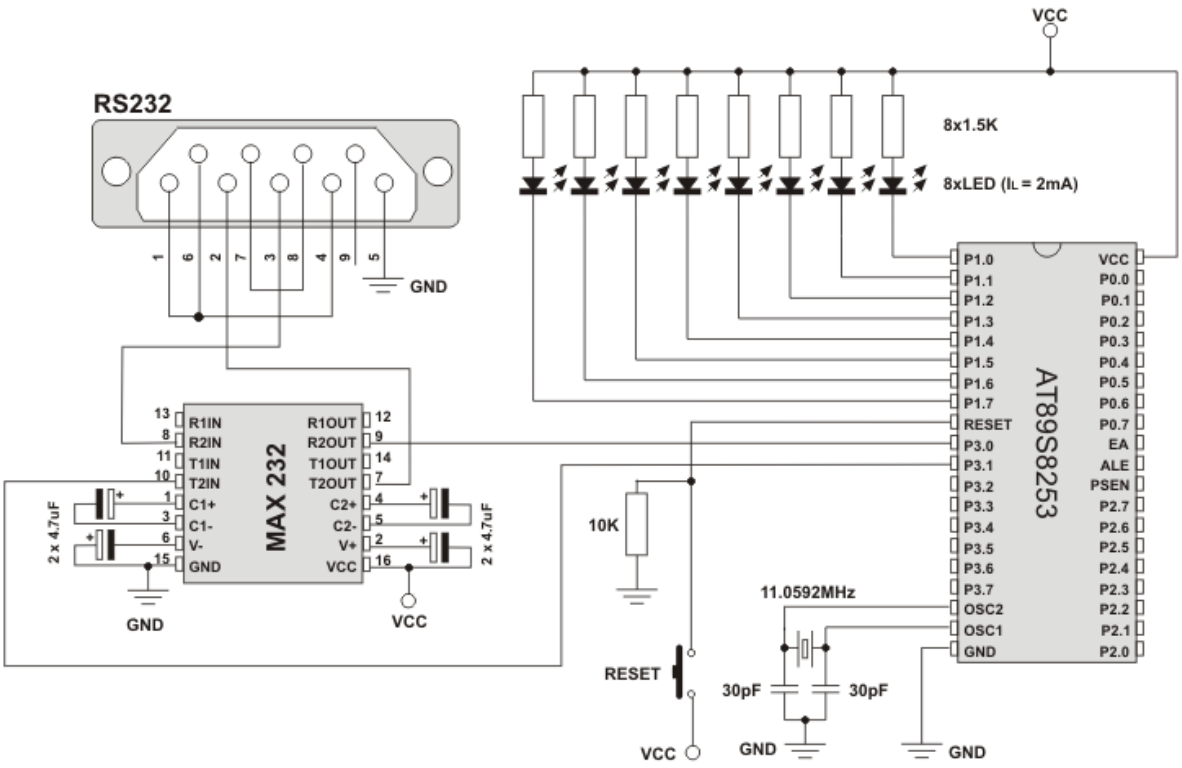
```
                        ; is generated on the LCD's EN pin
      CLR   LCD_enable
      POP   B
      POP   ACC
RET
END      ; End of program
```

Mã nguồn 4-14. Hiển thị LCD

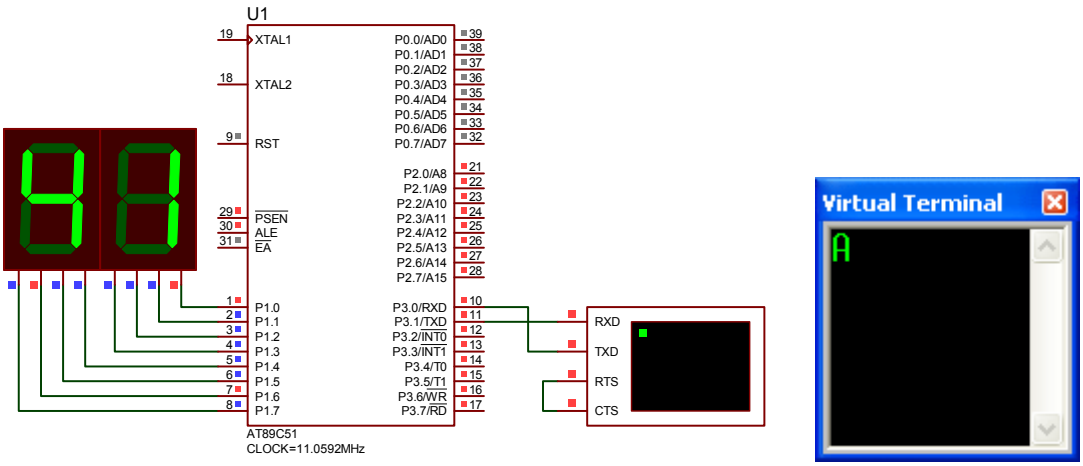
4.8 Nhận dữ liệu qua UART

Truyền thông giữa VĐK với máy tính, cần có bộ chuẩn hóa dữ liệu. Tín hiệu điện áp từ PC tại cổng COM là từ ±12v đến ±3v, trong khi đó, VĐK có chuẩn TTL 0v/5v. Trong thực tế, bộ chuẩn hóa thường dùng MAX232 như "Hình 4-15. Ghép nối VĐK với máy tính".

Ví dụ này cho thấy làm thế nào để nhận được thông điệp gửi từ PC. Timer T1 tạo tốc độ baud. Thạch anh 11,0592 MHz để tạo tốc độ baud là 9600 bps không có lỗi. Mỗi dữ liệu nhận được ngay lập tức được chuyển ra P1.



Hình 4-15. Ghép nối VĐK với máy tính



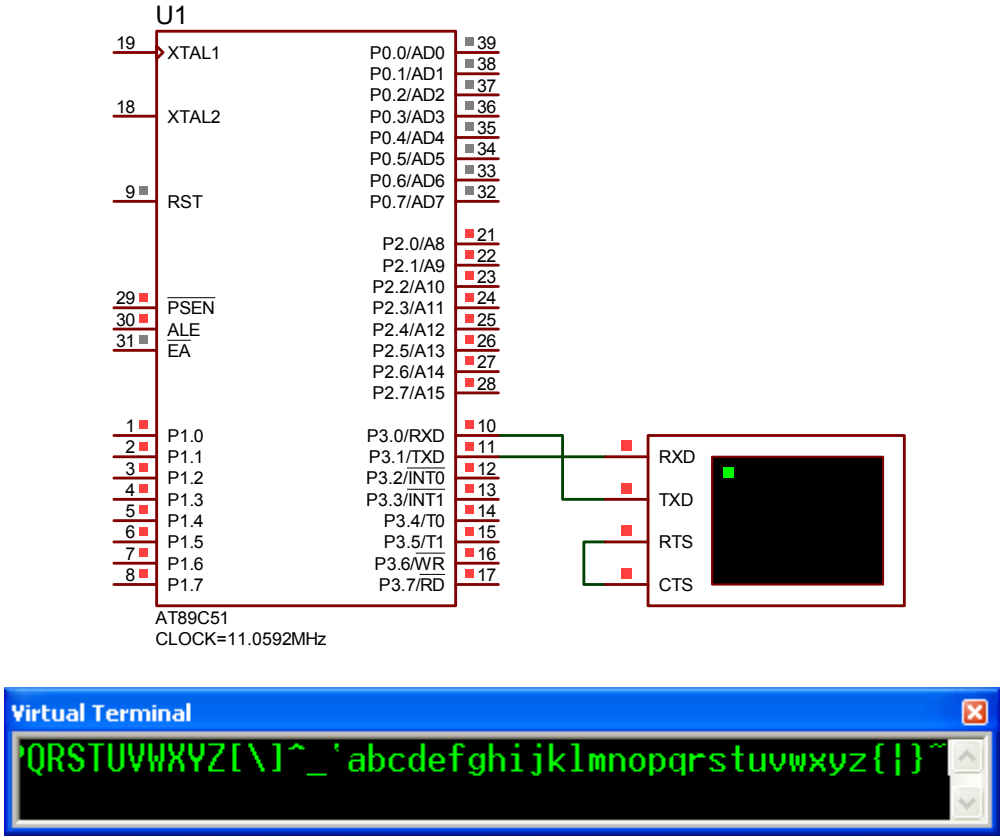
Hình 4-16. Nhận dữ liệu nối tiếp – mô phỏng

```
;*************************************************************
;* PROGRAM NAME : UartR.ASM
;* DESCRIPTION:Nhận dữ liệu từ UART, truyền thẳng xuống P1
;*************************************************
          CSEG      AT      0
          JMP       XRESET        ; Reset vector
          ORG       023H          ; Vector ngắt nối tiếp
          JMP       IR_SER

 ORG       100H
XRESET:   MOV       IE,#00            ; All interrupts are disabled
          MOV       TMOD,#20H         ; Timer1 in mode2
          MOV       TH1,#0FDH   ; 9600 baud rate at the frequency of
                                ; 11.0592MHz
          MOV       SCON,#50H         ; Receiving enabled, 8-bit UART
          MOV       IE,#10010000B     ; UART interrupt enabled
          CLR       TI                ; Clear transmit flag
          CLR       RI                ; Clear receive flag
          SETB      TR1               ; Start Timer1
LOOP:     SJMP      LOOP              ; Remain here

IR_SER:   JNB       RI,OUTPUT         ; If any data is received,
                                      ; move it to the port
          MOV       A,SBUF            ; P1
          MOV       P1,A
          CLR       RI                ; Clear receive flag
OUTPUT:   RETI

          END                         ; End of program
```

Mã nguồn 4-15. Nhận dữ liệu nối tiếp

4.9 Truyền dữ liệu qua UART

Chương trình này mô tả cách sử dụng UART để truyền dữ liệu. Một dãy số (0-255) được truyền đến máy PC ở tốc độ truyền 9600 baud. MAX 232 được sử dụng như là một bộ điều áp.



Hình 4-17. Truyền dữ liệu nối tiếp – mô phỏng

```
;*****************************************************************
;* PROGRAM NAME : UartS.ASM
;* DESCRIPTION: Sends values 65-127 to PC.
;*****************************************************************
         CSEG    AT    0
         JMP     XRESET          ; Reset vector
ORG     100H
XRESET:  MOV     IE,#00      ; All interrupts are disabled
         MOV     TMOD,#20H   ; Timer1 in mode 2
         MOV     TH1,#0FDH   ; 9600 baud rate at the frequency of
         MOV     TL1,#0FDH   ; 11.0592MHz
         MOV     SCON,#40H   ; 8-bit UART
         CLR     TI          ; Clear transmit bit
         CLR     RI          ; Clear receive flag
         MOV     R3,#'A'     ; Reset counter from A (65)
         SETB    TR1         ; Start Timer 1
START:   MOV     SBUF,R3     ; Move number from counter to a PC
LOOP1:   JNB     TI,LOOP1   ; Wait here until byte transmission is
                            ; complete
         CLR     TI          ; Clear transmit bit
       INC     R3           ; Increment the counter value by 1
         CJNE    R3,#127,START    ; Send until R3=127

LOOP:    SJMP    LOOP        ; Remain here
END                        ; End of program
```

Mã nguồn 4-16. Truyền dữ liệu nối tiếp

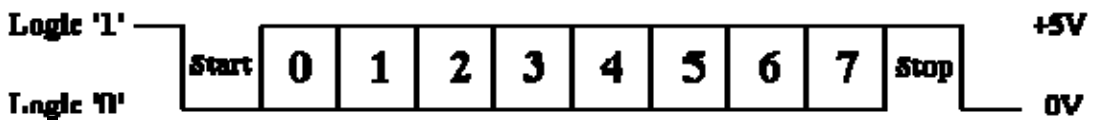
4.10 Chương trình con phục vụ truyền thông nối tiếp

```
Serial_Init: ;Khởi tạo:
        ;Set timer 1 mode to 8-bit Auto-Reload
        mov TMOD,#20H
        ;Enable reception
        ;Set Serial port mode to 8-bit UART
        mov SCON,#50H
        ;Set baudrate to 9600 at 11.0592MHz
        mov TH1,#0FDH
        mov TL1,#0FDH
        ;Start Timer
        setb TR1
        ret
 Serial_Send: ; truyền nội dung thanh ghi A ra UART
        ;wait for last data to be
        ;sent completely
        jnb TI,Serial_Send
        ;clear the transmit interrupt flag
        clr TI
        ;Then move the data to send in SBUF
        mov SBUF,A
        ret

Serial_Read:  ; nhận từ UART về thanh ghi A
        ;Wait for Receive interrupt flag
        jnb RI,Serial_Read
        ;If falg is set then clear it
        clr RI
        ;Then read data from SBUF
        mov A,SBUF
        ret
```

Mã nguồn 4-17. Các CTC Truyền-Nhận dữ liệu nối tiếp

4.11 Truyền thông UART cho 8051 bằng phần mềm

Để thực hiện thành công UART đầu tiên chúng ta cần phải biết giao thức truyền thông UART.



Sơ đồ trên cho thấy dạng sóng của một frame truyền. Đầu tiên là bit bắt đầu "START", sau đó 8-bit dữ liệu và một bit "STOP" cuối. Có một công thức bí mật để tính toán thời gian trì hoãn là có baudrate chính xác giữa các bit.

Dưới đây là một phần mềm triển khai UART, trong đó có thể được sử dụng ở chương trình C cũng như ASM. Nó được viết cho phần mềm Keil. Nhưng với một vài thay đổi nhỏ bạn có thể dùng nó trong chương trình của bạn.

```
?SU?PUTC SEGMENT CODE
?SU?GETC SEGMENT CODE

PUBLIC _putc
PUBLIC getc

txd_pin EQU     P3.1            ;Transmit on this pin
rxd_pin EQU     P3.0            ;Receive on this pin

;Formula to calculate the bit time delay constant
;This constant is calculated as: (((crystal/baud)/12) - 5) / 2
;crystal is the frequency of crystal in Hz
;baud is required baudrate
;Please try to keep baudrate below 9600
;to get best results :)

BITTIM  EQU     45;             (((11059200/9600)/12) - 5) / 2

;-----------------------------------------
;To send data serially
;For C programs
;Protype definition:
;               void putc(unsigned char);
;Usage:
;               putc(data);
;Return:
;               This function returns nothing
;
;For Assembly Programs:
;
;Usage:
;       data to be send has to be moved to R7
;       for example:
;               mov R7,#'a'
;               lcall _putc
;-----------------------------------------
RSEG ?SU?PUTC
_putc:
        push ACC
        Push PSW
        mov a,r7
        CLR txd_pin                     ;Drop line for start bit
        MOV R0,#BITTIM          ;Wait full bit-time
```

```
        DJNZ R0,$                       ;For START bit
        MOV R1,#8                       ;Send 8 bits
putc1:
        RRC A                           ;Move next bit into carry
        MOV txd_pin,C           ;Write next bit
        MOV R0,#BITTIM          ;Wait full bit-time
        DJNZ R0,$                       ;For DATA bit
        DJNZ R1,putc1           ;write 8 bits
        SETB txd_pin            ;Set line high
        RRC A                           ;Restore ACC contents
        MOV R0,#BITTIM          ;Wait full bit-time
        DJNZ R0,$                       ;For STOP bit
        POP PSW
        pop ACC
        RET

;-------------------------------------------
;To receive data Serially
;If you want to use this routine in your
;C program then define function prototype
; as:
;       unsigned char getc(void);
;
;       Usage:
;               data = getc();
;       Return value:
;               Returns data received
;If you are using it in assembly program
;       Usage:
;               lcall getc
;       Return:
;               data received is stored in R7
;-------------------------------------------

RSEG ?SU?GETC
getc:
        Push ACC
        Push PSW
        JB rxd_pin,$            ;Wait for start bit
        MOV R0,#BITTIM/2        ;Wait 1/2 bit-time
        DJNZ R0,$                       ;To sample in middle
        JB rxd_pin,getc         ;Insure valid
        MOV R1,#8                       ;Read 8 bits
getc1:
        MOV R0,#BITTIM          ;Wait full bit-time
        DJNZ R0,$                       ;For DATA bit
        MOV C,rxd_pin           ;Read bit
        RRC A                           ;Shift it into ACC
```

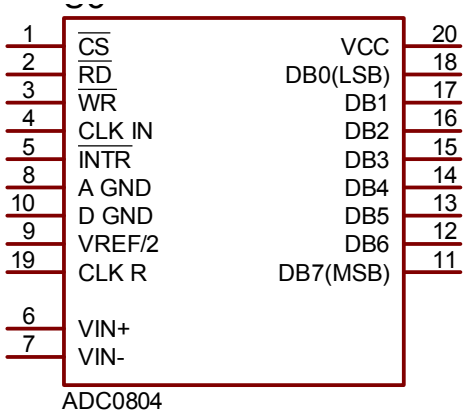
```
        DJNZ R1,getc1           ;read 8 bits
        mov r7,a
        POP PSW
        pop ACC
        RET                     ;go home
```

Mã nguồn 4-18. CTC truyền thông nối tiếp bằng phần mềm

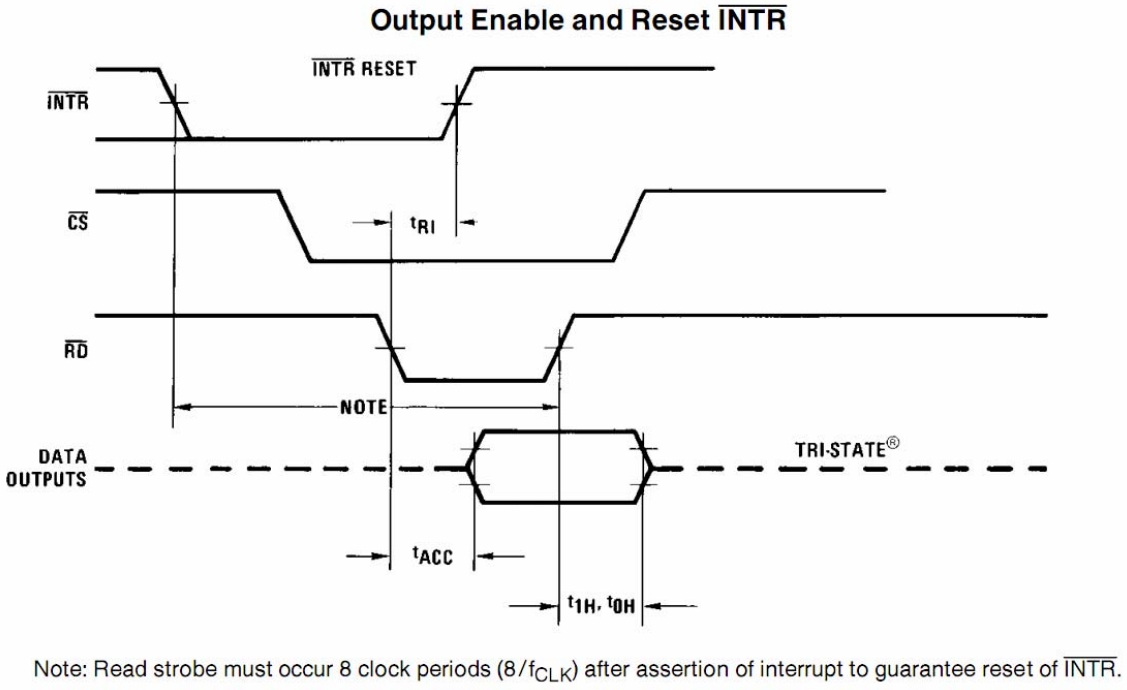
4.12 Ghép nối 8051 với ADC0804, chuyển đổi ADC 8-bit

Chuyển đổi ADC là chuyển đổi từ tương tự sang số, chúng ta có đầu vào là điện áp tương tự có dải từ 0v..5v, đầu ra sau khi chuyển đổi là số: 0..255, tương ứng, tỷ lệ với đầu vào.

Trước khi xây dựng mạch và lập trình, ta cần nghiên cứu đôi chút về IC ADC0804

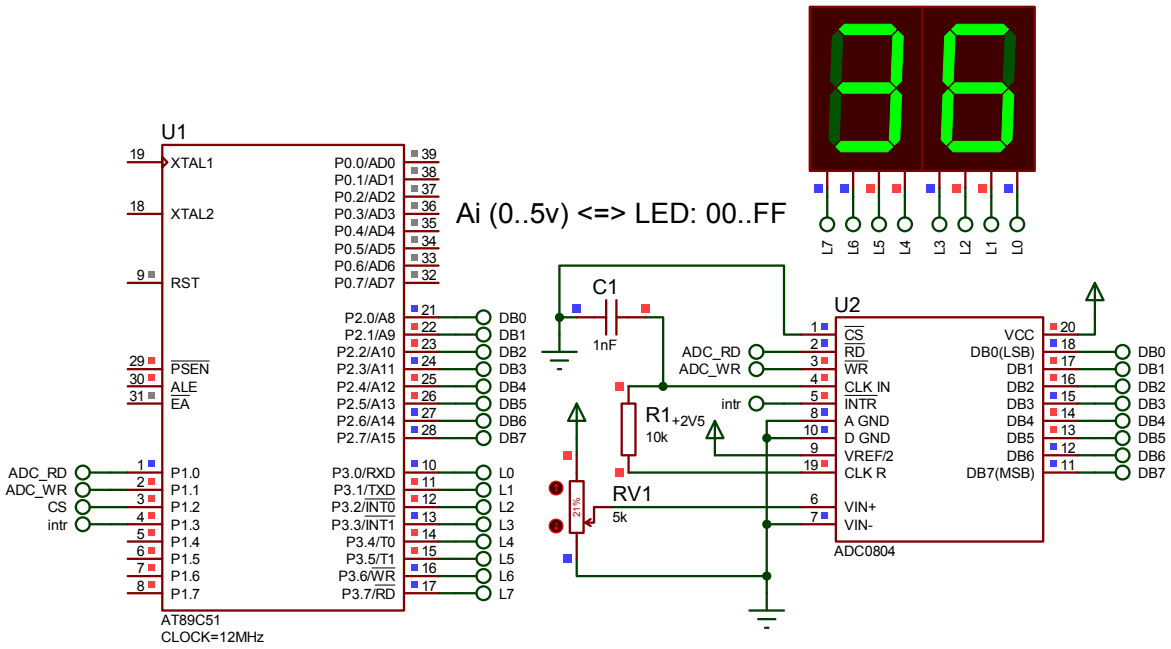


Hình 4-18. Sơ đồ chân ADC0804



Hình 4-19. Giản đồ thời gian đọc ADC

Cứ theo như giản đồ thời gian ta làm, vậy thì sơ đồ ghép nối chỉ cần 3 tín hiệu điều khiển và một cổng đọc ADC. Ta có mạch nguyên lý như sau:

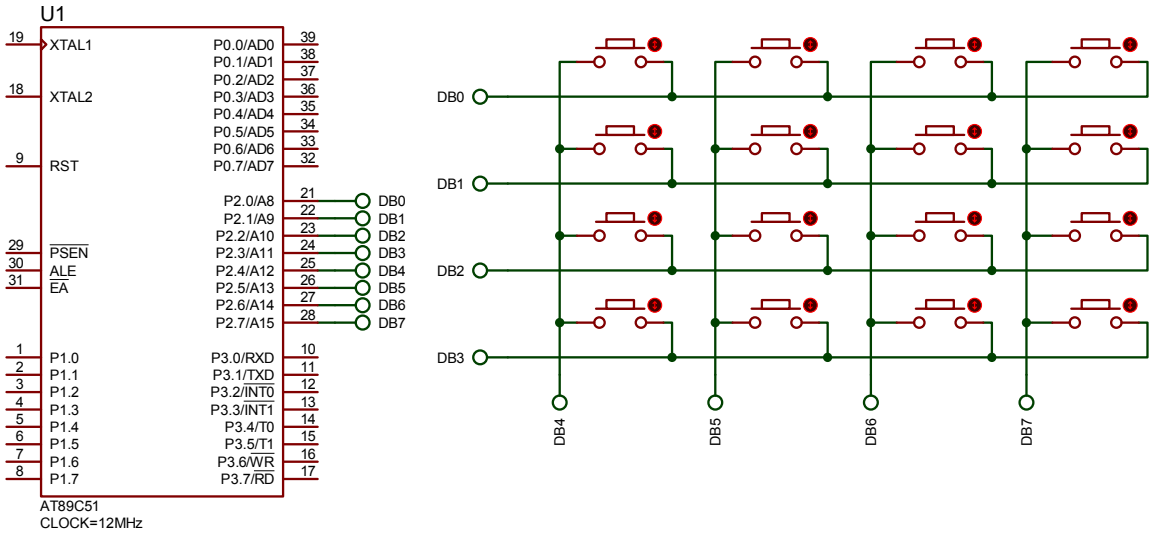


Hình 4-20. Mạch nguyên lý mô phỏng chuyển đổi ADC0804

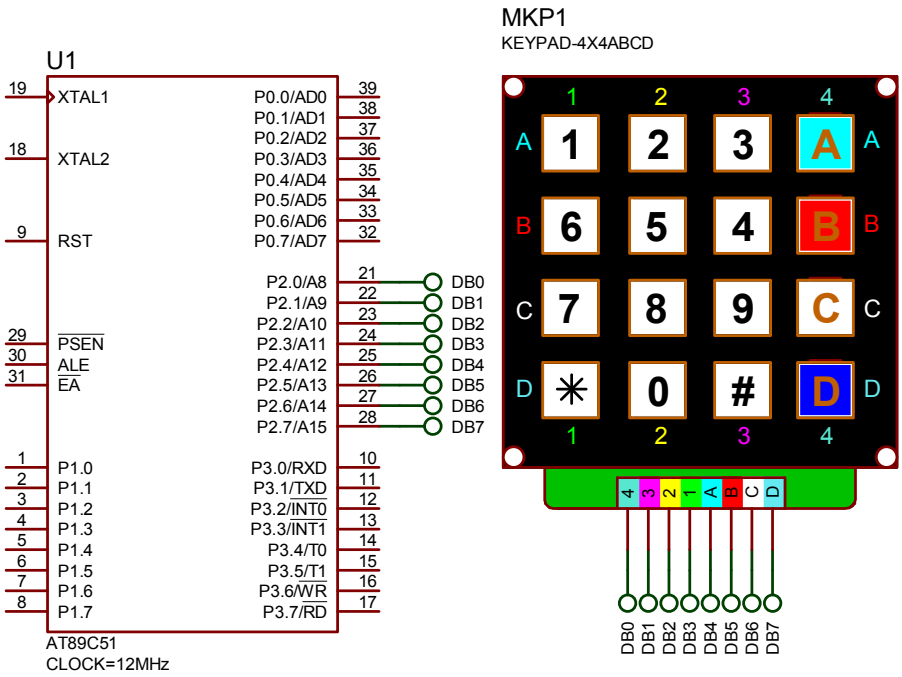
```
ORG 0h
JMP   MAIN
ADC_RD  EQU P1.0
ADC_WR  EQU P1.1
INTR    EQU P1.3
ADC_DAT EQU P2
LED7 EQU P3
        ;Khai báo chương trình con DelayX ở đây
        ; tham khảo "Mã nguồn 4-2. DelayX"
ORG     30H
MAIN:
     ACALL   TACT_LayMau
     mov LED7,A
SJMP    MAIN
TACT_LayMau:
     CLR ADC_WR        ; Tao xung tu cao xuong thap
                       ;tai chan ADC_WR(Tuc W/R cua ADC)
     DelayX 1
     SETB ADC_WR  ; Cho phep ADC0804 bat dau qua trinh
                       ; chuyen doi tu tuong tu sang so
     JB INTR, $    ; Doi cho qua trinh chuyen doi xong(100us)
     CLR ADC_RD    ; Dua xung muc thap toi chan RD -
                       ; cho phep doc du lieu tu ADC(Xuat ra D0..D7)
     DelayX 1
     MOV A,ADC_DAT ; Dua du lieu 8bit tu ADC_DAT den thanh ghi A
     setb   ADC_RD
   RET
END
```

Mã nguồn 4-19. Chuyển đổi ADC (VĐK-ADC0804)

4.13 Ghép nối vi điều khiển với bàn phím



Hình 4-21.Cách ghép nối bàn phím trong mô phỏng- phím đơn ghép lại



Hình 4-22.Cách ghép nối bàn phím trong mô phỏng – dùng module bàn phím

Thuật toán đọc phím bấm:

- Khởi tạo: Cho Cổng P2=0xFF
- Lần lượt cho hàng = 0.
- Với mỗi hàng, lần lượt kiểm tra cột, nếu cột nào = 0 → phím tương ứng với hàng và cột đó được bấm.
- Với mỗi phím được bấm, lưu lại kết quả (để làm gì sau đó, nếu cần)

```
ORG 0
JMP MAIN
      KQ      EQU 0
            COL1 EQU P2.3
            COL2 EQU P2.2
            COL3 EQU P2.1
            COL4 EQU P2.0
      ROW_A EQU P2.4
      ROW_B EQU P2.5
      ROW_C EQU P2.6
      ROW_D EQU P2.7
MAIN:
      MOV P2,#0FFh
      CLR ROW_A
            ADB0:JB COL1, ADB1
               MOV KQ,#1     // Phim 1 bam
            ADB1:JB COL2, ADB2
               MOV KQ,#2     // Phim 2 bam
            ADB2:JB COL3, ADB3
               MOV KQ,#3     //Phim 3 bam
            ADB3:JB COL4, AFINISH
               MOV KQ,#'A'    //Phim A bam
            AFINISH:
      SETB ROW_A

      CLR ROW_B
            BDB0:JB COL1, BDB1
               MOV KQ,#6   // Phim 6 bam
            BDB1:JB COL2, BDB2
               MOV KQ,#5   // Phim 5 bam
            BDB2:JB COL3, BDB3
               MOV KQ,#4   //Phim 4 bam
            BDB3:JB COL4, BFINISH
               MOV KQ,#'B'   //Phim 4 bam
            BFINISH:
      SETB ROW_B

      CLR ROW_C
            CDB0:JB COL1, CDB1
               MOV KQ,#7     // Phim 7 bam
            CDB1:JB COL2, CDB2
               MOV KQ,#8     // Phim 8 bam
            CDB2:JB COL3, CDB3
               MOV KQ,#9     //Phim 9 bam
            CDB3:JB COL4, CFINISH
               MOV KQ,#'C'        //Phim C bam
            CFINISH:
      SETB ROW_C
```

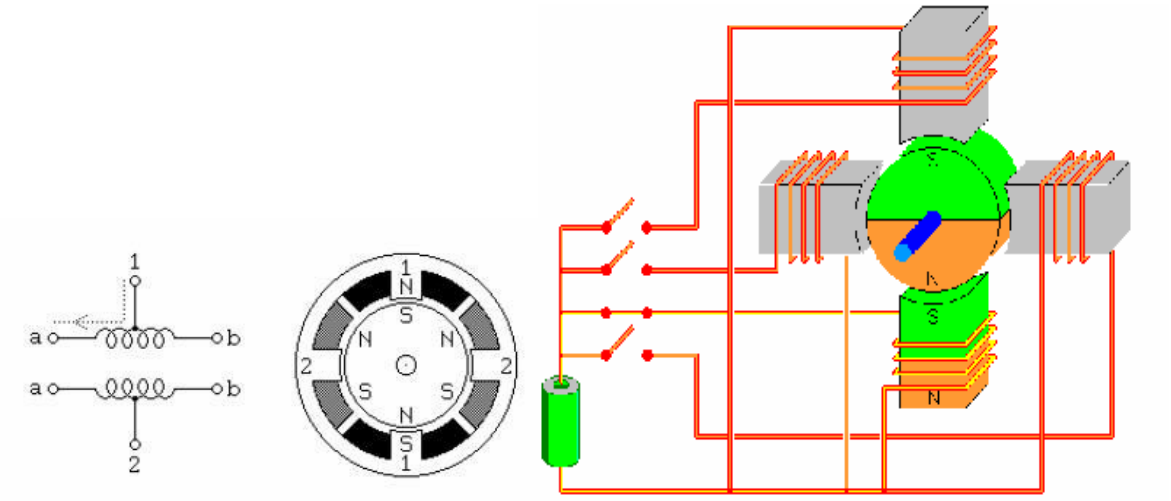
```
      CLR ROW_D
            DDB0:JB COL1, DDB1
                MOV KQ,#'*'    // Phim * bam
            DDB1:JB COL2, DDB2
                MOV KQ,#0    // Phim 0 bam
            DDB2:JB COL3, DDB3
                MOV KQ,#'#'    //Phim # bam
            DDB3:JB COL4, DFINISH
                MOV KQ,#'D'    //Phim D bam
            DFINISH:
      SETB ROW_D

      MOV P3,KQ // xử lý kết quả

   JMP MAIN
END
```

Mã nguồn 4-20. Đọc ma trận phím

4.14 Ghép nối vi điều khiển với step motor

Bài toán thực hiện việc điều khiển động cơ bước quay, thay đổi tốc độ, đảo chiều, dừng động cơ. Chương trình sử dụng 4 đầu tạo xung vào động cơ để làm thay đổi trạng thái của động cơ bước.

Thường các cuộn dây của động cơ bước được xác định theo màu dây, tuy nhiên đối với một động cơ bất kỳ, ta có thể dùng đồng hồ để xác định dây như hình vẽ, ở đây trình bày cách xác định động cơ có 5, 6 đầu dây.



Hình 4-23. Cấu tạo động cơ bước

1. dùng đồng hồ để xác định đầu chung (common) dùng đồng hồ để ở thang đo trở, đo trở giữa các cặp dây, đầu chung là đầu có trở giữa nó và các đầu khác bằng ½ điện trở các đầu khác với nhau.

Khi biết được thứ tự các cuộn dây, ta kích xung theo thứ tự đó động cơ sẽ chạy.

Ví dụ một đoạn chương trình sau, giả sử 4 đầu của động cơ bước đấu vào 4 bit: P1.0 – P1.3 của 8051.

```
ORG 0H
         MOV R3, #00000011B
 MOV  A,     R3
BACK: MOV P1,A
 RL  A      ;Quay thanh ghi A
 ACALL      DELAY
      SJMP  BACK
DELAY:
      MOV   R1,   #50
H1:   MOV   R2 ,  #255
H2:   DJNZ R2,    H2
      DJNZ R1,    H1
      RET
END
```

Mã nguồn 4-21. Điều khiển động cơ bước

Ví dụ mở rộng 1:

Chương trình đo nhiệt độ dùng LM35DZ, ADC0804, và thiết lập ; nhiệt độ cảnh báo bằng bàn phím máy tính:

```
LCD_DATA        EQU     P2
LCD_RS          BIT     P0.0
LCD_RW          BIT     P0.1
LCD_E           BIT     P0.2
ADC_DATA        EQU     P1
ADC_RD          BIT     P0.4
ADC_WR          BIT     P0.5
ADC_INTR        BIT     P0.6
KB_CLK          BIT     P3.2
KB_DATA         BIT     P0.3
WARN            BIT     P0.7

ORG 0000H
LJMP MAIN
ORG 0003H
LJMP EX0_ISR

ORG 0030H
MAIN:
LCALL CONFIG
;Thiet lap cac thong so ban dau
MAIN1: LCALL READ_ADC ;Doc ADC
LCALL CONVERT ;Chuyen doi
LCALL COMPARE ;So sanh va hien thi
LCALL DELAY_500MS ;Cho 0,1s
LJMP MAIN1
;_____________________________________
CONFIG: ;CTC thiet lap cac thong so
MOV A,#38H ;K.D LCD
LCALL WRCMD
MOV A,#0CH ;Display ON, Cursor OFF
LCALL WRCMD
MOV A,#06H ;LCD tu dong dich phai
LCALL WRCMD
MOV A,#01H ;Ghi loi chao
LCALL WRCMD
MOV DPTR,#CHAO1
LCALL OUT_STRING_LINE1
MOV DPTR,#CHAO2
LCALL OUT_STRING_LINE2
LCALL DELAY_2S
MOV DPTR,#CHAO3
LCALL OUT_STRING_LINE1
MOV DPTR,#CHAO4
LCALL OUT_STRING_LINE2
LCALL DELAY_2S
SETB WARN ;Tat den canh bao
CLR F0 ;F0=0: chuc, =1: dvi
MOV 41H,#'4'
;Dat nhiet do canh bao ban dau
MOV 42H,#'0'
MOV IE,#81H ;Cho phep ngat ngoai 0
RET
;_____________________________________
;CTC doc ADC ;
;Du lieu doc duoc chua trong 30H ;
;--------------------------------------;
READ_ADC:
MOV ADC_DATA,#0FFH ;De doc ADC chinh xac
SETB ADC_INTR ;nhan t.hieu canh xuong
CLR ADC_WR ;Tao canh len
SETB ADC_WR
JNB ADC_INTR,$ ;Cho chuyen doi xong
CLR ADC_RD ;Cho phep doc ADC
MOV A,ADC_DATA ;Doc du lieu tu ADC
MOV 30H,A ;Luu vào 30H
RET
;_____________________________________
;CTC chuyen ma doc duoc tu ADC chua ;
;trong 30H thanh nhiet do chua trong ;
;31H(chuc), 32H(Don vi), 33H(phan tram) ;ma
ASCII ;
;--------------------------------------
CONVERT:
MOV A,30H ;Lay ma doc duoc tu ADC
MOV B,#4 ;Do phan giai la 0,4oC
MUL AB
MOV R7,B ;Nhiet do dat trong R7-R6
MOV R6,A
MOV B,#10
LCALL DIV16_8
MOV A,B
ADD A,#30H
MOV 33H,A ;Thap phan
MOV B,#10
LCALL DIV16_8
MOV A,B
ADD A,#30H
MOV 32H,A ;Don vi
MOV B,#10
LCALL DIV16_8
MOV A,B
ADD A,#30H
MOV 31H,A ;Chuc
RET
;_____________________________________
;CTC chia 1 so 16-bit cho 1 so 8-bit ;
;So bi chia: R7-R6 ;
;So chia: B ;
;Thuong so: R7-R6 ;
;So du B ;
;--------------------------------------;
DIV16_8:
        CLR A
        MOV R2,#16
DIV1: CLR C
        LCALL RLC_R7R6
        ;Xoay trai R7_R6 qua co C
        RLC A
        CJNE A,B,NOT_EQ
        LJMP LOW1
NOT_EQ: JNC LOW1
        SJMP GIAM

LOW1:   SUBB A,B
        XCH A,R6
        ORL A,#01H
        XCH A,R6
        GIAM: DJNZ R2,DIV1
        MOV B,A ;So du chua trong B
```

```
RET
;_____________________________________
RLC_R7R6:
;CTC xoay trai so 16 bit R7_R6 qua co C
        PUSH ACC
        MOV A,R6
        RLC A
        MOV R6,A
        MOV A,R7
        RLC A
        MOV R7,A
        POP ACC
RET



;_____________________________________
;CTC so sanh nhiet do hien thi và ;
;nhiet do dat ;
;Neu lon hon hoac bang nhiet do dat ;
;thi canh bao ;
;-------------------------------------;
COMPARE:
   MOV A,#01H
   LCALL WRCMD
   MOV A,31H
   CJNE A,41H,KHAC
   MOV A,32H
   CJNE A,42H,KHAC
   LJMP CANHBAO ;Neu bang nhau thi canh bao
   KHAC:
   JNC CANHBAO ;Neu lon hon thi canh bao
   LJMP HIENTHI
   CANHBAO:
   CLR WARN ;Bat den canh bao
   MOV DPTR,#ST3
   LCALL OUT_STRING_LINE1
   MOV A,#0C1H
   LCALL WRCMD
   LCALL DISPLAY_TEMP
   MOV A,#' '
   LCALL WRTXT
   MOV A,#'>'
   LCALL WRTXT
   MOV A,#' '
   LCALL WRTXT
   LCALL DP_WARN_TEMP
   LJMP THOAT
   HIENTHI:
   SETB WARN ;Tat den canh bao
   MOV DPTR,#ST1
   LCALL OUT_STRING_LINE1
   MOV A,#08AH
   LCALL WRCMD
   LCALL DISPLAY_TEMP
   MOV DPTR,#ST2
   LCALL OUT_STRING_LINE2
   MOV A,#0C8H
   LCALL WRCMD
   LCALL DP_WARN_TEMP
THOAT: RET
DISPLAY_TEMP: ;CTC hien thi nhiet do
        MOV A,31H
        LCALL WRTXT
        MOV A,32H
        LCALL WRTXT
        MOV A,#','
        LCALL WRTXT
        MOV A,33H
        LCALL WRTXT
        MOV A,#'o'
        LCALL WRTXT
        MOV A,#'C'
        LCALL WRTXT
RET
DP_WARN_TEMP:
        MOV A,41H
        LCALL WRTXT
        MOV A,42H
        LCALL WRTXT
        MOV A,#'o'
        LCALL WRTXT
        MOV A,#'C'
        LCALL WRTXT
RET
;_____________________________________
;CTC xuat mot chuoi ra LCD ;
;Con tro DPTR chi toi chuoi can xuat ;
;-------------------------------------;
OUT_STRING:
        MOV R4,#0
        OUTST1: MOV A,R4
        MOVC A,@A+DPTR
        LCALL WRTXT
        INC R4
        CJNE R4,#16,OUTST1
RET
OUT_STRING_LINE1:
        MOV A,#80H
        LCALL WRCMD
        LCALL OUT_STRING
RET
OUT_STRING_LINE2:
        MOV A,#0C0H
        LCALL WRCMD
        LCALL OUT_STRING
RET
WRCMD: ;CTC ghi lenh ra LCD
        CLR LCD_RW
        SETB LCD_E
        CLR LCD_RS
        MOV LCD_DATA,A
        NOP
        CLR LCD_E
        LCALL READY
RET
WRTXT: ;CTC ghi ki tu ra LCD
        CLR LCD_RW
        SETB LCD_E
        SETB LCD_RS
        MOV LCD_DATA,A
        NOP
        CLR LCD_E
        LCALL READY
RET
READY: ;CTC cho LCD
        PUSH ACC
        OK: CLR LCD_E
        CLR LCD_RS
        SETB LCD_RW
        MOV LCD_DATA,#0FFH
        SETB LCD_E
        MOV A,LCD_DATA
        JB ACC.7,OK
```

```
        CLR LCD_RW
        POP ACC
RET
DELAY_500MS: ;CTC delay 0,5s
        MOV R7,#250
        DELAY1: MOV R6,#200
        DELAY2: MOV R5,#5
        DJNZ R5,$
        DJNZ R6,DELAY2
        DJNZ R7,DELAY1
RET
DELAY_2S: ;CTC delay 2s
        MOV R7,#250
        DELAY3: MOV R6,#200
        DELAY4: MOV R5,#20
        DJNZ R5,$
        LOOP: JB KB_CLK,$ ;Lay 8 bit Data
        MOV C,KB_DATA
        RRC A
        JNB KB_CLK,$
        DJNZ R3,LOOP

        MOV R3,#24
        SKIP: JB KB_CLK,$
        JNB KB_CLK,$
        DJNZ R3,SKIP

        MOV 40H,A
        MOV R4,#0
        LOOP1: MOV DPTR,#SCAN MOV A,R4
        MOVC A,@A+DPTR
        CJNE A,40H,LOOP2
        MOV DPTR,#ASCII
        MOV A,R4
        MOVC A,@A+DPTR
        JB F0,DVI ;Neu F0=1 ghi hang d.vi
CHUC:
        MOV 41H,A ;neu F0=0 nho hang chuc
        SETB F0
        LJMP EXIT
        DJNZ R6,DELAY4
        DJNZ R7,DELAY3
RET
;_____________________________________
;CTC ngat ngoai 0 ;
;Doc tu ban phim P/S2 ;
;Va h.thi ra LCD ;
;--------------------------------------;
EX0_ISR:
        CLR EX0
        PUSH ACC
        MOV A,#00H
        MOV R3,#8

        JNB KB_CLK,$ ;Bo qua bit Start

         DVI: MOV 42H,A
         CLR F0
         LJMP EXIT
 LOOP2: INC R4
         CJNE R4,#20,LOOP1 ; 20 lan?
 EXIT: CLR C
         POP ACC
         SETB EX0 ;Bat co ngat
 RETI
 CHAO1: DB ' Duc, Q.Toan '
 CHAO2: DB 'H.Thuong, Nguyen'
 CHAO3: DB 'CT do n.do so V1'
 CHAO4: DB ' Xin chao! '
 ST1: DB 'Nhiet do: '
 ST2: DB 'C.bao: '
 ST3: DB ' Canh bao! '
 SCAN: DB
 45H,16H,1EH,26H,25H,2EH,36H,3DH,3EH,46H,70
 H,69H,72H,7AH,6BH,73H,74H,6CH,75H,7DH
 ASCII: DB '01234567890123456789'

 END
```

Mã nguồn 4-22. Chương trình đo nhiệt độ

Ví dụ mở rộng 2:

Thiết kế hệ thống hiển thị và cảnh báo áp suất nước trong bình nén, với 03 mức thấp, trung bình và cao (ngưỡng do người dùng tự đặt)

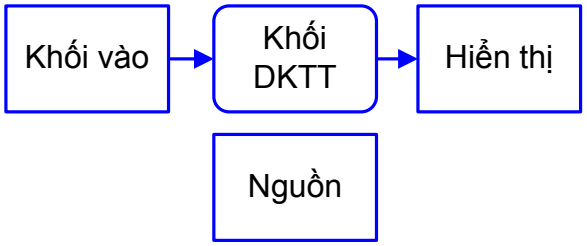
Biết cảm biến áp suất có tín hiệu ra trong khoảng 0..100mV tương ứng với áp suất từ 0..3000 atmosphe.

Yêu cầu thiết kế theo các bước sau:

- Thiết kế sơ đồ khối tổng thể toàn hệ thống
- Đặc tả mỗi khối:
  - Chức năng, nhiệm vụ của khối
  - Số lượng và chuẩn tín hiệu điện áp vào/ra
- Thiết kế sơ đồ tương tác (thuật toán nhúng) toàn hệ thống
- Thiết kế sơ đồ nguyên lý
- Đặc tả sơ đồ nguyên lý
  - Chức năng, nhiệm vụ của (nhóm) linh kiện
  - Chuẩn giao tiếp (chuẩn truyền thông (nếu có chuẩn), lúc bình thường/lúc hoạt động,…)
- Lập trình
  - Sơ đồ Call graph
  - Sơ đồ khối (của mỗi chức năng trong sơ đồ call graph, nếu cần)
  - Viết mã nguồn

Đáp Án:

- *Thiết kế sơ đồ khối tổng thể toàn hệ thống:*



Hình 4-24. Sơ đồ khối tổng thể toàn bộ hệ thống

- *Đặc tả mỗi khối:*
  - *Chức năng, nhiệm vụ, số lượng và chuẩn tín hiệu điện áp vào/ra của khối:*

a. **Khối vào**: Gồm cảm biến áp suất và các phím bấm. Cảm biến có chức năng thu nhận giá trị áp suất. Phím bấm dùng để nhập giá trị tham số từ bàn phím.
   - Cảm biến áp suất có đầu vào là áp suất thuộc khoảng 0..3000 atmosphe, đầu ra là 0..100mV. Tín hiệu này sẽ được lọc nhiễu, khuếch đại tỷ lệ lên 0..5v trước khi đưa vào khối điều khiển trung tâm.
   - Phím bấm: gồm 3 phím có đầu ra chuẩn điện áp TTL, bình thường là mức 1, khi được bấm sẽ về mức 0.
   - Vậy, khối vào, đưa ra 1 tín hiệu Analog (0..5v), và 3 tín hiệu số cho 3 nút bấm.

b. **Khối hiển thị:** Hiển thị trạng thái của hệ thống lên LCD16x2 và 3 LED đơn. LCD dùng để liên tục hiển thị trạng thái của hệ thống như: Giá trị áp suất, giá trị ngưỡng

cảnh báo mức áp suất, dùng để hiển thị giá trị tham số, hiển thị giá trị nút bấm khi nhập tham số,… 3 đèn LED gồm LED Đỏ, LED xanh, LED Trắng.

- LCD sẽ dùng chế độ 4bit, nên cần 7 đường hiển thị số cho LCD: RS, RW, E, D4..D7
- LED đơn dùng để báo trạng thái ngưỡng áp suất, các màu khác nhau để nhìn từ xa, sử dụng LED siêu sáng. Cần 03 tín hiệu số trực tiếp từ khối điều khiển trung tâm.
- Vậy, đầu vào khối hiển thị là: 10 tín hiệu số.

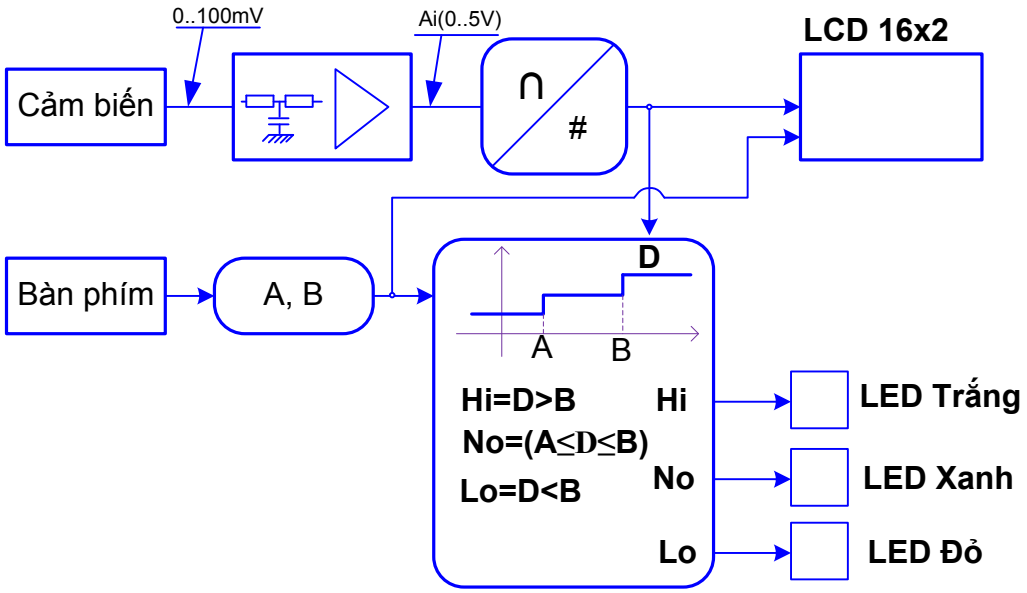
c. **Khối Điều khiển trung tâm (DKTT):** Có chức năng xử lý tín hiệu từ khối vào để đưa ra hiển thị và cảnh báo ở khối hiển thị.

- Đọc tín hiệu tương tự từ khối vào (0..5v), chuyển đổi ADC sang số (0..1023) để xử lý.
- Đọc giá trị nút bấm để thay đổi tham số (Ngưỡng thấp và Ngưỡng cao) cảnh báo.
- So sánh giá trị ADC thực tế và giá trị ngưỡng để quyết định trạng thái cần cảnh báo lên LED đơn, và hiển thị giá trị áp suất, các tham số lên LCD 16x2

d. **Khối nguồn:** Hệ thống sử dụng nguồn 5v, và ±12v cho bộ khuếch đại trong khối đầu vào. Nên bộ nguồn cần thiết phải thiết kế là:

- Vào: 220VAC
- Ra: -12v, GND, +5v, +12v có ổn áp.
- Dự tính toàn bộ hệ thống tiêu thụ công suất thấp. Nên bộ nguồn sẽ chỉ cần dòng điện tối đa 1A cho mỗi đầu ra.

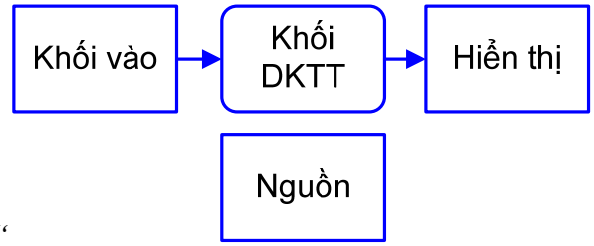
- *Thiết kế sơ đồ tương tác (thuật toán nhúng) toàn hệ thống:*



Hình 4-25. Sơ đồ tương tác hệ thống cảnh báo áp suất

- *Thiết kế sơ đồ nguyên lý*
- *Đặc tả sơ đồ nguyên lý*
  - *Chức năng, nhiệm vụ của (nhóm) linh kiện*
  - *Chuẩn giao tiếp (chuẩn truyền thông (nếu có chuẩn), lúc bình thường/lúc hoạt động/,…)*

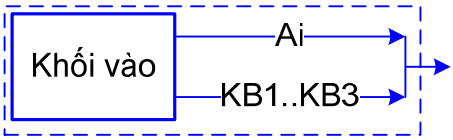
Thiết kế nguyên lý chi tiết cho từng khối theo



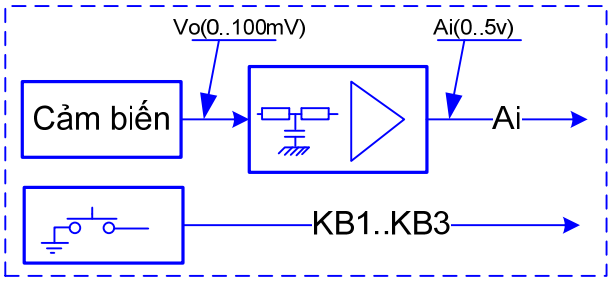
"

❖ Hình 4-24. Sơ đồ khối tổng thể toàn bộ hệ thống":

❖ Khối Vào: được yêu cầu như sau:

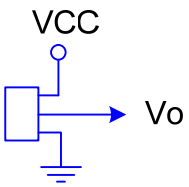


Hình 4-26.

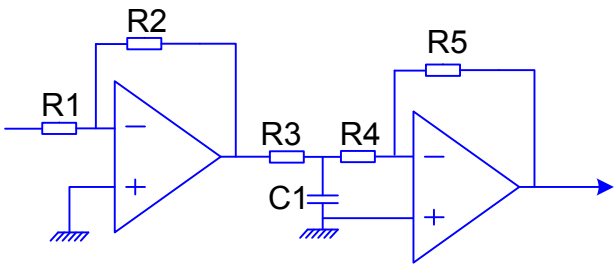


Hình 4-27.

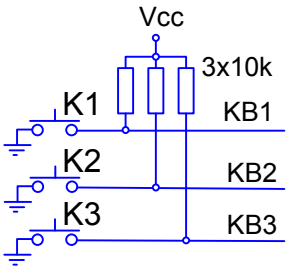
Khối vào sẽ bao gồm: Khối cảm biến, khối lọc và khuếch đại, khối bàn phím. Chi tiết được thiết kế như sau:



Hình 4-28. Mạch cảm biến



Hình 4-29. Khuếch đại và lọc nhiễu



Hình 4-30. Bàn phím

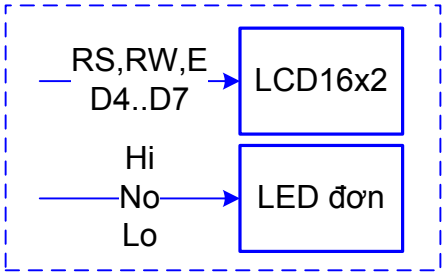
Đặc tả khối:

- Khối cảm biến, đầu vào là áp suất đo của đối tượng, đầu ra là điện áp, có dải từ 0..100mV
- Khối khuếch đại và lọc nhiễu, nhận tín hiệu từ khối cảm biến (0..100mv) xuất ra tín hiệu 0..5v tuyến tính với tín hiệu vào. Như vậy, hệ số khuếch đại là 5v/100mV = k = 50 lần.
  Từ đó, ta tính được như sau: **Error! Objects cannot be created from editing field codes.**, ta chọn linh kiện: R1=R2=R3=R4=10K, R5=50K
  Khuếch đại, dùng LM324, nguồn đối xứng ±12v

- Khối bàn phím, có 3 phím, bình thường thì KB1..KB3 có mức 1, phím nào được bấm sẽ có mức 0.

❖ Khối hiển thị:

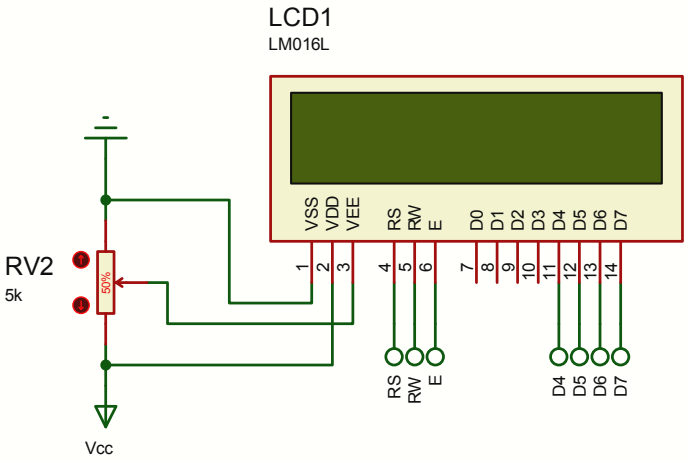
Hình 4-31.



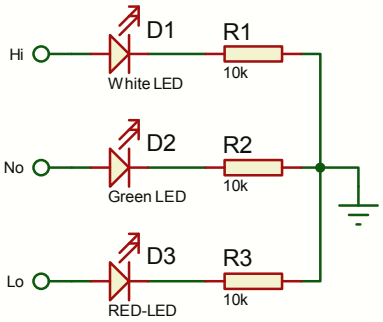
Hình 4-32.

Khối hiển thị được cho như "Hình 4-31", ta triển khai sâu hơn như "Hình 4-33. Hiển thị LCD".

Thiết kế chi tiết các khối như sau:

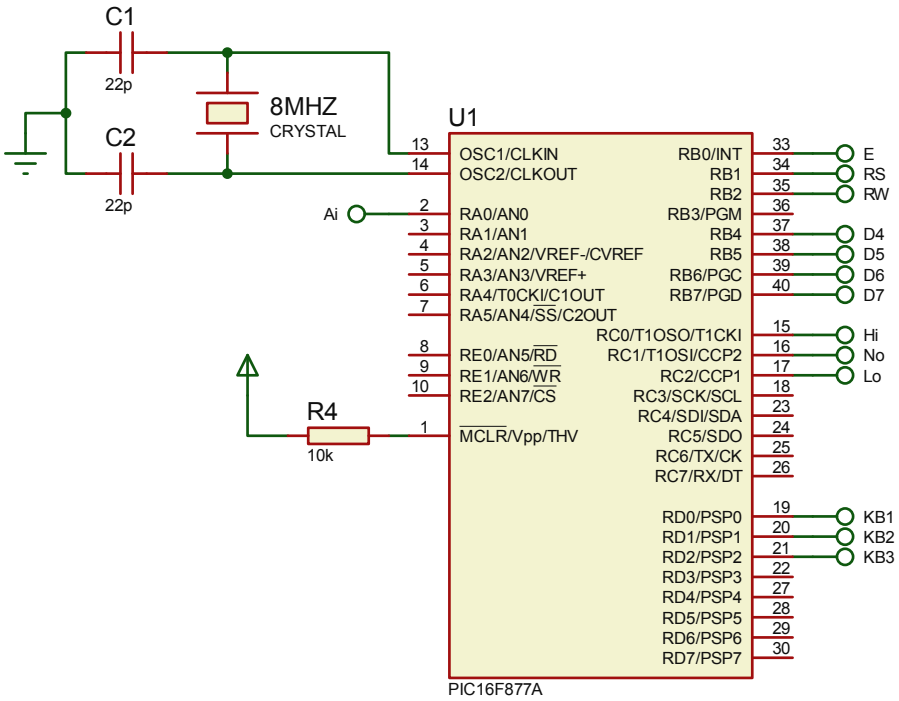


Hình 4-33. Hiển thị LCD



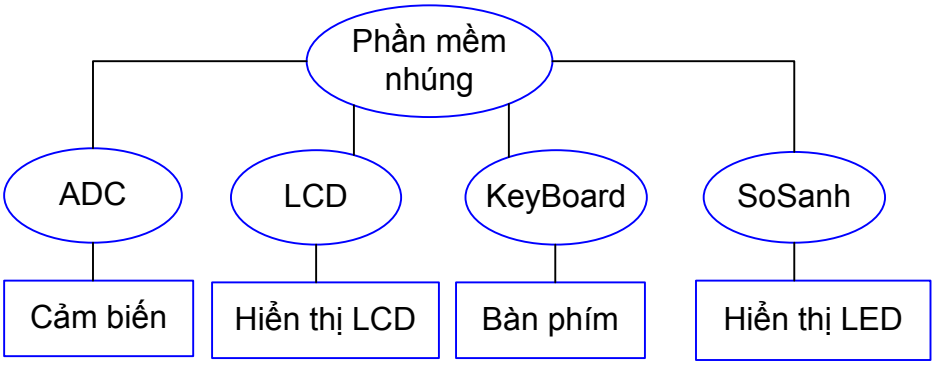
Hình 4-34. Bàn phím

❖ Khối điều khiển trung tâm



Hình 4-35. Khối điều khiển trung tâm

- *Lập trình*
  - *Sơ đồ Call graph*
  - *Sơ đồ khối (của mỗi chức năng trong sơ đồ call graph, nếu cần)*
  - *Viết mã nguồn*



Hình 4-36. Sơ đồ Callgraph

Mã nguồn: Vì dự án này rất lớn, và khi thiết kế lại chọn PIC, nên ngôn ngữ lập trình dùng ngôn ngữ C cho PIC là hợp lý nhất.

```
#include "ADC-LCD-LED.h"
#define use_portb_lcd true
#include <LCD.C>
#define KB1 !input(PIN_D0)
#define KB2 !input(PIN_D1)
#define KB3 !input(PIN_D2)
#define T 50
int16 D,Hi,No,Lo,A,B;
int8 cnt;
void init_main();
```

```
void KeyBoard(){
   if(KB1){Mode=(++Mode)%3;delay_ms(T);}
   if(Mode==1){
      if(KB2){A++;delay_ms(T);}
      if(KB3){A--;delay_ms(T);}
   }
   if(Mode==2){
      if(KB2){B++;delay_ms(T);}
      if(KB3){B--;delay_ms(T);}
}   }
void main(){
   init_main();
   while(1){
      D=read_adc();
      lcd_gotoxy(1,1); printf(lcd_putc,"Ap
Suat=%4LU.Mode=%u\nA=%4u.B=%4u",D,Mode,A,B);
      Hi_LED=No_LED=Lo_LED=0;
      if(D<A)Lo_LED=1;
      else if(D>B)Hi_LED=1;
         else No_LED=1;
      KeyBoard();
      delay_ms(100);
}   }
void init_main(){
   setup_adc_ports(AN0);
   setup_adc(ADC_CLOCK_INTERNAL);
      set_adc_channel(0);
   lcd_init();
}
```

Mã nguồn 4-23. Lập trình cho VĐK tiên tiến

(Trang này nên bỏ trống)

CHƯƠNG 5.
CÁC HỆ VI ĐIỀU KHIỂN TIÊN TIẾN

Mục tiêu

Giúp sinh viên biết về các hệ vi điều khiển hiện đại và phổ biến trong thực tế sản xuất; và ứng dụng cơ bản của chúng.

Tóm tắt:

Tìm hiểu về các vi điều khiển hiện đại họ AVR, họ PIC và ARM

5.1 Atmel AVR

Hình 5-1 - Atmel AVR ATmega8 PDIP

AVR là một kiến trúc Harvard sửa đổi 8-bit RISC đơn chip vi điều khiển (μC) đã được phát triển bởi Atmel vào năm 1996. Các AVR là một trong những họ vi điều khiển đầu tiên sử dụng on-chip bộ nhớ flash để lưu trữ chương trình, trái với One-Time Programmable ROM, EPROM hoặc EEPROM được sử dụng bởi vi điều khiển khác vào lúc đó.

5.1.1 Lịch sử họ AVR

Người ta tin vào kiến trúc AVR cơ bản đã được hình thành bởi hai sinh viên tại Viện Công nghệ Na Uy (thứ n) Alf-Egil Bogen và Vegard Wollan.

Các AVR MCU bản gốc đã được phát triển tại một ngôi nhà ASIC thuộc địa phương ở Trondheim, Na Uy, nơi mà hai thành viên sáng lập của Atmel Na Uy đã làm việc như sinh viên. Nó được biết đến như một μRISC (Micro RISC). Khi công nghệ đã được bán cho Atmel, kiến trúc nội bộ đã được phát triển thêm bởi Alf và Vegard tại Atmel Na Uy, một công ty con của Atmel thành lập bởi hai kiến trúc sư.

Atmel AVR nói rằng các tên không phải là một từ viết tắt và không phải là bất cứ điều gì đặc biệt. Những người sáng tạo AVR không có câu trả lời dứt khoát về thuật ngữ viết tắt "AVR".

Lưu ý rằng việc sử dụng "AVR" trong bài viết này thường đề cập đến 8-bit RISC dòng vi điều khiển Atmel AVR.

Trong số những thành viên đầu tiên của dòng AVR là AT90S8515, đóng vỏ trong gói 40-pin DIP có chân ra giống như một vi điều khiển 8051, bao gồm địa chỉ BUS multiplexed bên ngoài và dữ liệu. Tín hiệu RESET đã đổi ngược, 8051 RESET mức cao, AVR RESET mức thấp), nhưng khác với đó, chân ra là giống hệt nhau.

5.1.2 Tổng quan về thiết bị

AVR là một kiến trúc máy Modified Harvard với chương trình và dữ liệu được lưu trữ trong các hệ thống bộ nhớ vật lý riêng biệt xuất hiện trong không gian địa chỉ khác nhau, nhưng có khả năng đọc ghi dữ liệu từ bộ nhớ bằng cách sử dụng lệnh đặc biệt.

Cơ bản về họ AVR
AVRs thường phân thành bốn nhóm rộng:

- TinyAVR - chuỗi Attiny
  - 0,5-8 kBbộ nhớ chương trình
  - Đóng vỏ 6-32-chân
  - Tập ngoại vi hữu hạn
- MegaAVR - chuỗi Atmega
  - 4-256 kB bộ nhớ chương trình
  - Đóng vỏ 28-100-chân
  - Tập lệnh mở rộng (Lệnh nhân và lệnh cho quản lý bộ nhớ lớn hơn).
  - Mở rộng hơn về thiết bị ngoại vi
- XMEGA - chuỗi Atxmega
  - 16-384 kB bộ nhớ chương trình.
  - Đóng vỏ 44-64-100-chân (A4, A3, A1)
  - Mở rộng các tính năng hiệu suất, chẳng hạn như DMA, "Sự kiện hệ thống", và hỗ trợ mật mã.
  - Thiết bị ngoại vi được mở rộng với DACs
- Ứng dụng cụ thể AVR
  - megaAVRs với các tính năng đặc biệt không tìm thấy trên các thành viên khác của gia đình AVR, chẳng hạn như bộ điều khiển LCD, USB, điều khiển, nâng cao PWM, CAN v.v..
  - Atmel At94k FPSLIC (Field Programmable System Level Circuit), một lõi trên AVR với một FPGA. FPSLIC sử dụng SRAM cho mã chương trình AVR, không giống như tất cả các AVRs khác. Một phần do sự khác biệt tốc độ tương đối giữa SRAM và flash, lõi AVR trong FPSLIC có thể chạy lên đến 50 MHz.

5.1.3 Kiến trúc thiết bị

Flash, EEPROM, và SRAM tất cả được tích hợp vào một chip duy nhất, loại bỏ sự cần thiết của bộ nhớ ngoài trong hầu hết các ứng dụng. Một số thiết bị có BUS mở rộng song song để cho phép thêm dữ liệu bổ sung (hoặc mã) bộ nhớ, hoặc bộ nhớ ánh xạ thiết bị. Tất cả các thiết bị có giao tiếp nối tiếp, mà có thể được sử dụng để kết nối EEPROMs nối tiếp chip flash.

5.1.4 Program Memory (Flash)

Mã lệnh chương trình được lưu trữ trong bộ nhớ Flash chống xóa (non-volatile Flash). Mặc dù họ là 8-bit MCUs, mỗi lệnh mất 1 hoặc 2 từ 16-bit.

Kích cỡ của bộ nhớ chương trình thường được chỉ định trong việc đặt tên của thiết bị chính (ví dụ, dòng ATmega64x có 64 kB của Flash, tuy nhiên ATmega32x chỉ có 32kB).

5.1.5 EEPROM

Hầu như tất cả các vi điều khiển AVR đều có Electrically Erasable Programmable Read Only Memory (EEPROM) để lưu “nửa vĩnh viễn” dữ liệu lưu

trữ. Cũng giống như bộ nhớ Flash, EEPROM có thể duy trì nội dung của nó khi được gỡ bỏ.

Trong hầu hết các biến thể của kiến trúc AVR, bộ nhớ EEPROM nội bộ này không phải là ánh xạ vào không gian địa chỉ bộ nhớ của MCU. Nó chỉ có thể được truy cập cùng một cách như là thiết bị ngoại vi bên ngoài, thanh ghi sử dụng con trỏ đặc biệt và đọc / ghi hướng dẫn mà làm cho truy cập EEPROM chậm hơn nhiều so với RAM nội bộ khác.

Tuy nhiên, một số thiết bị trong dòng SecureAVR (AT90SC) sử dụng một bản đồ EEPROM đặc biệt đến các dữ liệu hoặc bộ nhớ chương trình tùy thuộc vào cấu hình. Dòng XMEGA cũng cho phép EEPROM ánh xạ vào không gian địa chỉ dữ liệu.

Kể từ khi số lượng các lần ghi EEPROM không phải là không giới hạn - Atmel chỉ được 100.000 chu kỳ ghi.

5.1.6 Chương trình thực thi

Atmel's AVRs có hai giai đoạn, thiết kế kiểu đường ống (pipeline) duy nhất. Điều này có nghĩa là chỉ lệnh kế tiếp là được lấy khi lệnh này đang thực hiện. Hầu hết các lệnh chỉ mất một hoặc hai chu kỳ đồng hồ, làm cho AVRs tương đối nhanh trong số vi điều khiển 8-bit.

Họ AVR của bộ vi xử lý được thiết kế với sự thực hiện hiệu quả của mã C.

5.1.7 Tập lệnh

Tập lệnh AVR hơn là trực giao với hầu hết các vi điều khiển tám-bit, đặc biệt là 8051 và vi điều khiển PIC với AVR mà ngày nay đang cạnh tranh. Tuy nhiên, nó không phải là hoàn toàn bình thường:

- Con trỏ ghi X, Y, và Z có khả năng đánh địa chỉ khác với nhau.
- Vị trí thanh ghi R0 đến R15 có khả năng đánh địa chỉ khác hơn vị trí thnah ghi R16 đến R31.
- I / O port 0-31 có khả năng đánh địa chỉ khác so với I / O ports 32-63.
- CLR ảnh hưởng đến các cờ, trong khi SER không, ngay cả khi chúng được lệnh bổ sung. CLR xóa tất cả các bit về không và SER đặt chúng lên một.
- Truy cập dữ liệu chỉ đọc được lưu trong bộ nhớ chương trình (flash) yêu cầu lệnh đặc biệt LPM.

Ngoài ra, một số chip-sự khác biệt cụ thể ảnh hưởng đến các thế hệ mã. Mã con trỏ (bao gồm cả các địa chỉ trở lại stack) là hai byte trên chip lên đến 128 KBytes bộ nhớ flash, nhưng ba byte trên chip lớn hơn, không phải tất cả các chip có số nhân phần cứng; chip với hơn 8 Kbytes flash có nhánh và gọi lệnh với khoảng rộng hơn; v.v. .

Lập trình cho nó bằng cách sử dụng lập trình C (hoặc thậm chí Ada) trình biên dịch khá đơn giản. GCC đã bao gồm hỗ trợ AVR từ khá lâu, và hỗ trợ được sử dụng

rộng rãi. Trong thực tế, Atmel gạ gẫm đầu vào từ các nhà phát triển chính của trình biên dịch cho vi điều khiển nhỏ, để tích hợp tính năng cho các tập lệnh hữu dụng nhất trong một trình biên dịch cho các ngôn ngữ cấp cao.

5.1.8 Tốc độ MCU

Dòng AVR bình thường có thể hỗ trợ tốc độ đồng hồ 0-20 MHz, với một số thiết bị đạt 32 MHz. Hỗ trợ hoạt động thấp hơn thường đòi hỏi một tốc độ giảm. Tất cả gần đây (Tiny và Mega, nhưng không phải 90S) AVRs tích hợp oscillator-chip, loại bỏ sự cần thiết của đồng hồ bên ngoài hoặc mạch dao động. Một số AVRs cũng có một prescaler đồng hồ hệ thống, có thể chia xuống đồng hồ của hệ thống lên đến 1024. Prescaler này có thể được cấu hình lại bằng phần mềm trong thời gian chạy, cho phép tối ưu hóa tốc độ đồng hồ.

Vì tất cả các hoạt động (trừ literals) trên thanh ghi R0 - R31 là đơn chu kỳ, các AVR có thể đạt được lên đến 1MIPS mỗi MHz. Tải và lưu trữ vào / ra bộ nhớ mất 2 chu kỳ, phân nhánh phải mất 3 chu kỳ.

5.1.9 Những đặc tính

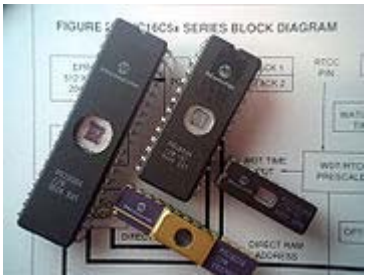
AVRs hiện cung cấp một loạt các tính năng:

- Máy đa chức năng, Bi-directional General Purpose I / O port với cấu hình, built-in pull-up resistors
- Nhiều nội Oscillators, bao gồm cả RC oscillator mà không có bộ phận bên ngoài
- Nội, lệnh Self-Programmable Flash Memory lên đến 256 KB (384 KB trên XMega)
    - In-System Programmable sử dụng nối tiếp / song song hạ thế độc quyền hoặc các giao diện JTAG
    - Tùy chọn khởi động với bảo vệ Lock Bits độc lập.
- On-chip gỡ lỗi (OCD) hỗ trợ thông qua JTAG hoặc debugWIRE trên hầu hết các thiết bị
    - tín hiệu JTAG (TMS, TDI, TDO, và TCK) là multiplexed ngày GPIOs. Những Pin có thể được cấu hình với chức năng như JTAG hoặc GPIO tùy thuộc vào thiết lập của một vài cầu chì (FUSES), có thể được lập trình thông qua ISP hoặc HVSP. Theo mặc định, AVRs với JTAG đi kèm với giao diện JTAG bật.
    - debugWIRE sử dụng chân /RESET như một kênh giao tiếp hai hướng để truy cập vào mạch debug-chip. Đó là hiện nay trên các thiết bị với số lượng chân ít, vì nó chỉ cần một chân.
- Internal Data EEPROM lên đến 4 kB
- Internal SRAM lên đến 8 kB (32 kB trên XMega)
- Ngoài 64KB dữ liệu trên các mô hình không gian nhất định, bao gồm cả Mega8515 và Mega162.
    - Trong một số thành viên của loạt XMEGA, dữ liệu không gian bên ngoài đã được tăng cường để hỗ trợ cả hai SRAM và SDRAM. Đồng

thời, các dữ liệu địa chỉ, các chế độ đã được mở rộng cho phép lên đến 16MB bộ nhớ của dữ liệu được đề cập trực tiếp.
    - AVR thường không hỗ trợ thực thi mã từ bộ nhớ bên ngoài. Một số ASSP bằng cách sử dụng mã AVR làm bộ nhớ hỗ trợ chương trình bên ngoài.
- 8-Bit và 16-Bit Timers
    - PWM đầu ra (thời gian chết máy phát điện trên một số thiết bị)
    - vào capture
- So sánh Analog
    - 10 hoặc 12-Bit A / D Converters, với multiplex lên đến 16 kênh
    - 12-bit D / A Converters
- Một loạt các giao tiếp nối tiếp, bao gồm cả
    - I²C tương thích Two-Wire Interface (TWI)
    - Thiết bị ngoại vi Synchronous / Asynchronous Serial (UART / USART) (được sử dụng với RS-232, RS-485, và nhiều hơn nữa)
    - Thiết bị giao diện Serial Bus (SPI)
    - Universal Serial Interface (USI) cho 2 hoặc 3 dây truyền thông đồng bộ nối tiếp.
- Brownout Detection
- Watchdog Timer (WDT)
- Nhiều chế độ tiết kiệm điện (Power-Saving Sleep)
- Điều khiển ánh sáng và điều khiển động cơ (cụ thể là PWM) điều khiển mô hình
- Hỗ trợ CAN Controller
- Hỗ trợ USB Controller
    - USB – Full speed (12 Mbit / s) điều khiển phần cứng & Hub với AVR nhúng.
    - Cũng sẵn sàng tự do với tốc độ thấp (1,5 Mbit / s) (HID) bitbanging EMULATIONS phần mềm
- Hỗ trợ Ethernet Controller
- Hỗ trợ LCD Controller
- Hoạt động ở mức điện áp thấp, có thể xuống đến 1.8v (đến 0.7v với loại hỗ trợ chuyển đổi DC-DC)
- Thiết bị picoPower
- bộ điều khiển DMA và truyền thông "Sự kiện hệ thống" ngoại vi.
- Mã hóa và giải mã nhanh, hỗ trợ cho AES và DES

5.2 Vi điều khiển PIC

PIC 1655A



Các dòng PIC khác

Hình 5-2 – Các dòng PIC

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

PIC bắt nguồn là chữ viết tắt của "Programmable Intelligent Computer" (Máy tính khả trình thông minh) là một sản phẩm của hãng General Instruments đặt cho dòng sản phẩm đầu tiên của họ là PIC1650. Lúc này, PIC1650 được dùng để giao tiếp với các thiết bị ngoại vi cho máy chủ 16bit CP1600, vì vậy, người ta cũng gọi PIC với cái tên "Peripheral Interface Controller" (Bộ điều khiển giao tiếp ngoại vi). CP1600 là một CPU tốt, nhưng lại kém về các hoạt động xuất nhập, và vì vậy PIC 8-bit được phát triển vào khoảng năm 1975 để hỗ trợ hoạt động xuất nhập cho CP1600. PIC sử dụng microcode đơn giản đặt trong ROM, và mặc dù, cụm từ RISC chưa được sử dụng thời bây giờ, nhưng PIC thực sự là một vi điều khiển với kiến trúc RISC, chạy một lệnh một chu kỳ máy (4 chu kỳ của bộ dao động).

Năm 1985 General Instruments bán bộ phận vi điện tử của họ, và chủ sở hữu mới hủy bỏ hầu hết các dự án - lúc đó đã quá lỗi thời. Tuy nhiên PIC được bổ sung EEPROM để tạo thành 1 bộ điều khiển vào ra khả trình. Ngày nay rất nhiều dòng PIC được xuất xưởng với hàng loạt các module ngoại vi tích hợp sẵn (như USART, PWM, ADC...), với bộ nhớ chương trình từ 512 Word đến 32K Word.

❖ Lập trình cho PIC

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

Một số công ty khác xây dựng các trình dịch C, Basic, Pascal cho PIC. Microchip cũng bán trình dịch "C18" (cho dòng PIC high-end) và "C30" (cho dòng

dsPIC30Fxxx). Họ cũng cung cấp các bản "student edition/demo" dành cho sinh viên hoặc người dùng thử, những version này không có chức năng tối ưu hoá code và có thời hạn sử dụng giới hạn. Những trình dịch mã nguồn mở cho C, Pascal, JAL, và Forth, cũng được cung cấp bởi PicForth.

GPUTILS là một kho mã nguồn mở các công cụ, được cung cấp theo công ước về bản quyền của GNU General Public License. GPUTILS bao gồm các trình dịch, trình liên kết, chạy trên nền Linux, Mac OS X, OS/2 và Microsoft Windows. GPSIM cũng là một trình mô phỏng dành cho **vi điều khiển PIC** thiết kế ứng với từng module phần cứng, cho phép giả lập các thiết bị đặc biệt được kết nối với PIC, ví dụ như LCD, LED...

❖ Một vài đặc tính

Hiện nay có khá nhiều dòng PIC và có rất nhiều khác biệt về phần cứng, nhưng chúng ta có thể điểm qua một vài nét như sau:

- 8/16 bit CPU, xây dựng theo kiến trúc Harvard có sửa đổi
- Flash và ROM có thể tuỳ chọn từ 256 byte đến 256 Kbyte
- Các cổng Xuất/Nhập (I/O ports) (mức logic thường từ 0V đến 5.5V, ứng với logic 0 và logic 1)
- 8/16 Bit Timer
- Công nghệ Nanowatt
- Các chuẩn Giao Tiếp Ngoại Vi Nối Tiếp Đồng bộ/Không đồng bộ USART, AUSART, EUSARTs
- Bộ chuyển đổi ADC Analog-to-digital converters, 10/12 bit
- Bộ so sánh điện áp (Voltage Comparators)
- Các module Capture/Compare/PWM
- LCD
- MSSP Peripheral dùng cho các giao tiếp I²C, SPI, và I²S
- Bộ nhớ nội EEPROM - có thể ghi/xoá lên tới 1 triệu lần
- Module Điều khiển động cơ, đọc encoder
- Hỗ trợ giao tiếp USB
- Hỗ trợ điều khiển Ethernet
- Hỗ trợ giao tiếp CAN
- Hỗ trợ giao tiếp LIN
- Hỗ trợ giao tiếp IrDA
- Một số dòng có tích hợp bộ RF (PIC16F639, và rfPIC)
- KEELOQ Mã hoá và giải mã
- DSP những tính năng xử lý tín hiệu số (dsPIC)

❖ Họ vi điều khiển PIC 8/16-bit

Các link này được lấy từ trang chủ www.microchip.com, tuy nhiên hiện nay trang này đang rất thường bị chết, có thể do lượng truy cập quá nhiều, và các đường dẫn luôn thay đổi, vì vậy, có thể link sẽ bị chết.

Vi điều khiển 8-bit

- PIC10, PIC12, PIC14, PIC16, PIC17, PIC18

Vi điều khiển 16-bit:

- PIC24

Bộ điều khiển xử lý tín hiệu số 16-bit (dsPIC):

- dsPIC30
- dsPIC33F

Bộ điều khiển xử lý tín hiệu số 32-bit (PIC32):

- PIC32

5.3 ARM

❖ Cấu trúc ARM

Cấu trúc ARM (viết tắt từ tên gốc là Acorn RISC Machine) là một loại cấu trúc vi xử lý 32-bit kiểu RISC được sử dụng rộng rãi trong các thiết kế nhúng. Do có đặc điểm tiết kiệm năng lượng, các bộ CPU ARM chiếm ưu thế trong các sản phẩm điện tử di động, mà với các sản phẩm này việc tiêu tán công suất thấp là một mục tiêu thiết kế quan trọng hàng đầu.

Ngày nay, hơn 75% CPU nhúng 32-bit là thuộc họ ARM, điều này khiến ARM trở thành cấu trúc 32-bit được sản xuất nhiều nhất trên thế giới. CPU ARM được tìm thấy khắp nơi trong các sản phẩm thương mại điện tử, từ thiết bị cầm tay (PDA, điện thoại di động, máy đa phương tiện, máy trò chơi cầm tay, và máy tính cầm tay) cho đến các thiết bị ngoại vi máy tính (ổ đĩa cứng, bộ định tuyến để bàn.) Một nhánh nổi tiếng của họ ARM là các vi xử lý Xscale của Intel.

Trụ sở chính của công ty ARM tại Cambridge Anh)

Một bộ vi xử lý Conexant được dùng chủ yếu trong các bộ định tuyến

❖ Lịch sử phát triển

Việc thiết kế ARM được bắt đầu từ năm 1983 trong một dự án phát triển của công ty máy tính Acorn.

Nhóm thiết kế, dẫn đầu bởi Roger Wilson và Steve Furber, bắt đầu phát triển một bộ vi xử lý có nhiều điểm tương đồng với Kỹ thuật MOS 6502 tiên tiến. Acorn đã từng sản xuất nhiều máy tính dựa trên 6502, vì vậy việc tạo ra một chip như vậy là một bước tiến đáng kể của công ty này.

Nhóm thiết kế hoàn thành việc phát triển mẫu gọi là ARM1 vào năm 1985, và vào năm sau, nhóm hoàn thành sản phẩm ‘’thực’’ gọi là ARM2. ARM2 có tuyến dữ liệu 32-bit, không gian địa chỉ 26-bit tức cho phép quản lý đến 64 Mbyte địa chỉ và 16 thanh ghi 32-bit. Một trong những thanh ghi này đóng vai trò là bộ đếm chương trình với 6 bit cao nhất và 2 bit thấp nhất lưu giữ các cờ trạng thái của bộ vi xử lý. Có thể nói ARM2 là bộ vi xử lý 32-bit khả dụng đơn giản nhất trên thế giới, với chỉ gồm 30.000 transistor (so với bộ vi xử lý lâu hơn bốn năm của Motorola là 68000

với khoảng 68.000 transistor). Sự đơn giản như vậy có được nhờ ARM không có vi chương trình (mà chiếm khoảng ¼ đến 1/3 trong 68000) và cũng giống như hầu hết các CPU vào thời đó, không hề chứa cache. Sự đơn giản này đưa đến đặc điểm tiêu thụ công suất thấp của ARM, mà lại có tính năng tốt hơn cả 286. Thế hệ sau, ARM3, được tạo ra với 4KB cache và có chức năng được cải thiện tốt hơn nữa.

Vào những năm cuối thập niên 80, hãng máy tính Apple Computer bắt đầu hợp tác với Acorn để phát triển các thế hệ lõi ARM mới . Công việc này trở nên quan trọng đến nỗi Acorn nâng nhóm thiết kế trở thành một công ty mới gọi là Advanced RISC Machines. Vì lý do đó bạn thường được giải thích ARM là chữ viết tắt của Advanced RISC Machines thay vì Acorn RISC Machine. Advanced RISC Machines trở thành công ty ARM Limited khi công ty này được đưa ra sàn chứng khoán London và NASDAQ năm 1998.

Kết quả sự hợp tác này là ARM6. Mẫu đầu tiên được công bố vào năm 1991 và Apple đã sử dụng bộ vi xử lý ARM 610 dựa trên ARM6 làm cơ sở cho PDA hiệu Apple Newton. Vào năm 1994, Acorn dùng ARM 610 làm CPU trong các máy vi tính RiscPC của họ.

Trải qua nhiều thế hệ nhưng lõi ARM gần như không thay đổi kích thước. ARM2 có 30.000 transistors trong khi ARM6 chỉ tăng lên đến 35.000. Ý tưởng của nhà sản xuất lõi ARM là sao cho người sử dụng có thể ghép lõi ARM với một số bộ phận tùy chọn nào đó để tạo ra một CPU hoàn chỉnh, một loại CPU mà có thể tạo ra trên những nhà máy sản xuất bán dẫn cũ và vẫn tiếp tục tạo ra được sản phẩm với nhiều tính năng mà giá thành vẫn thấp.

Thế hệ thành công nhất có lẽ là ARM7TDMI với hàng trăm triệu lõi được sử dụng trong các máy điện thoại di động, hệ thống video game cầm tay, và Sega Dreamcast. Trong khi công ty ARM chỉ tập trung vào việc bán lõi IP, cũng có một số giấy phép tạo ra bộ vi điều khiển dựa trên lõi này.

Dreamcast đưa ra bộ vi xử lý SH4 mà chỉ mượn một số ý tưởng từ ARM (tiêu tán công suất thấp, tập lệnh gọn …) nhưng phần còn lại thì khác với ARM. Dreamcast cũng tạo ra một chip xử lý âm thanh được thiết kế bởi Yamaha với lõi ARM7. Bên cạnh đó, Gameboy Advance của Nintendo, dùng ARM7TDMI ở tần số 16,78 MHz.

Hãng DEC cũng bán giấy phép về lõi cấu trúc ARM (đôi khi chúng ta có thể bị nhầm lẫn vì họ cũng sản xuất ra DEC Alpha) và sản xuất ra thế hệ Strong ARM. Hoạt động ở tần số 233 MHz mà CPU này chỉ tiêu tốn khoảng 1 watt công suất (những đời sau còn tiêu tốn ít công suất hơn nữa). Sau những kiện tụng, Intel cũng được chấp nhận sản xuất ARM và Intel đã nắm lấy cơ hội này để bổ sung vào thế hệ già cỗi i960 của họ bằng Strong ARM. Từ đó, Intel đã phát triển cho chính họ một sản phẩm chức năng cao gọi tên là Xscale.

❖ Các dạng lõi

Họ	Lõi	Đặc tính	Cache (I/D)/MMU	MIPS điển hình @ MHz	Ứng dụng
ARM7TDMI	ARM7TDMI (-S)	3-tầng pipeline	không	15 MIPS @ 16.8 MHz	Game Boy Advance, Nintendo DS, iPod
	ARM710T	MMU		36 MIPS @ 40 MHz	Psion 5 series
	ARM720T		8KB unified, MMU	60 MIPS @ 59.8 MHz	
	ARM740T		MPU		
	ARM7EJ-S	Jazelle DBX	không		
ARM9TDMI	ARM9TDMI	5-tầng pipeline	không		
	ARM920T		16KB/16KB, MMU	200 MIPS @ 180 MHz	GP32,GP2X (lõi đầu tiên), Tapwave Zodiac (Motorola i.MX1)
	ARM922T		8KB/8KB, MMU		
	ARM940T		4KB/4KB, MPU		GP2X (lõi thứ hai)
ARM9E	ARM946E-S		thay đổi được, tightly coupled memories, MPU		Nintendo DS, Nokia N-Gage, Conexant 802.11 chips
	ARM966E-S		không có cache, TCMs		ST Micro STR91xF, includes Ethernet [1]
	ARM968E-S		không có cache, TCMs		
	ARM926EJ-S	Jazelle DBX	thay đổi được, TCMs, MMU	220 MIPS @ 200 MHz	Điện thoại di động: Sony Ericsson (K, W series),Siemens and Benq (đời x65 và đời mới hơn)
	ARM996HS	Clockless processor	không caches, TCMs, MPU		
ARM10E	ARM1020E	(VFP)	32KB/32KB, MMU		
	ARM1022E	(VFP)	16KB/16KB, MMU		
	ARM1026EJ-S	Jazelle DBX	variable, MMU or MPU		
ARM11	ARM1136J (F)-S	SIMD, Jazelle DBX, (VFP)	variable, MMU		
	ARM1156T2 (F)-S	SIMD, Thumb-2, (VFP)	thay đổi được, MPU		
	ARM1176JZ	SIMD, Jazelle	thay đổi được,		

Họ	Lõi	Đặc tính	Cache (I/D)/MMU	MIPS điển hình @ MHz	Ứng dụng
	(F)-S	DBX, (VFP)	MMU+TrustZone		
	ARM11 MPCore	1-4 core SMP, SIMD, Jazelle DBX, (VFP)	thay đổi được, MMU		
Cortex	Cortex-A8	Application profile, NEON, Jazelle RCT, Thumb-2	variable (L1+L2), MMU+TrustZone	lên đến 2000 (2.0 DMIPS/MHz in speed from 600 MHz to greater than 1 GHz)	Texas Instruments OMAP3
	Cortex-R4	Embedded profile	variable cache, MMU optional	600 DMIPS	Broadcom là một hãng sử dụng
	Cortex-M3	Microcontroller profile	no cache, (MPU)	120 DMIPS @ 100MHz	Luminary Micro[2] microcontroller family
XScale	80200/IOP310 /IOP315	I/O Processor			
	80219				
	IOP321				Iyonix
	IOP33x				
	PXA210 /PXA250	Applications processor			Zaurus SL-5600
	PXA255		32KB/32KB, MMU	400 BogoMips @400 MHz	Gumstix
	PXA26x				
	PXA27x			800 MIPS @ 624 MHz	HTC Universal, Zaurus SL-C1000
	PXA800(E)F				
	Monahans			1000 MIPS @ 1.25 GHz	
	PXA900				Blackberry 8700
	IXC1100	Control Plane Processor			
	IXP2400 /IXP2800				
	IXP2850				
	IXP2325 /IXP2350				
	IXP42x				NSLU2
	IXP460				

Họ	Lõi	Đặc tính	Cache (I/D)/MMU	MIPS điển hình @ MHz	Ứng dụng
	/IXP465				

❖ Các lưu ý về thiết kế

Để đạt được một thiết kế gọn, đơn giản và nhanh, các nhà thiết kế ARM xây dựng nó theo kiểu nối cứng không có vi chương trình, giống với bộ vi xử lý 8-bit 6502 đã từng được dùng trong các máy vi tính trước đó của hãng Acorn.

Cấu trúc ARM bao gồm các đặc tính của RISC như sau:

- Cấu trúc nạp/lưu trữ.
- Không cho phép truy xuất bộ nhớ không thẳng hàng (bây giờ đã cho phép trong lõi Arm v6)
- Tập lệnh trực giao
- File thanh ghi lớn gồm 16 x 32-bit
- Chiều dài mã máy cố định là 32 bit để dễ giải mã và thực hiện pipeline, để đạt được điều này phải chấp nhận giảm mật độ mã máy.
- Hầu hết các lệnh đều thực hiện trong vòng một chu kỳ đơn.

So với các bộ vi xử lý cùng thời như Intel 80286 và Motorola 68020, trong ARM có một số tính chất khá độc đáo như sau:

- Hầu hết tất cả các lệnh đều cho phép thực thi có điều kiện, điều này làm giảm việc phải viết các tiêu đề rẽ nhánh cũng như bù cho việc không có một bộ dự đoán rẽ nhánh.
- Trong các lệnh số học, để chỉ ra điều kiện thực hiện, người lập trình chỉ cần sửa mã điều kiện
- Có một thanh ghi dịch đóng thùng 32-bit mà có thể sử dụng với chức năng hoàn hảo với hầu hết các lệnh số học và việc tính toán địa chỉ.
- Có các kiểu định địa chỉ theo chỉ số rất mạnh
- Có hệ thống con thực hiện ngắt hai mức ưu tiên đơn giản nhưng rất nhanh, kèm theo cho phép chuyển từng nhóm thanh ghi.

Tài liệu tham khảo

[1]. Tống Văn On, Hoàng Đức Hải, ***Họ vi điều khiển 8051,*** NXB Lao động xã hội, năm 2001

[2]. Nguyễn Tăng Cường, ***Cấu trúc và lập trình họ vi điều khiển 8051,*** NXB Khoa học và kỹ thuật, năm 2004

[3]. Nguyễn Minh Tuấn, ***Giáo trình hợp ngữ - Chương 1***, ĐHKHTN, 2002

[4]. Randal Hyde, ***The art of assembly language programming – Chapter 1.***

[5]. Norton Guide

[6]. Dan Rollins, TechHelp v.6.0

[7]. http://picat.dieukhien.net

[8]. wapedia. http://wapedia.mobi/vi/Hợp_ngữ

[9]. http://www.emu8086.com/

[10]. http://www.daniweb.com/code/

[11]. http://www.freewebs.com/maheshwankhede/adcdac.html

[12]. http://wapedia.mobi/vi

PHỤ LỤC A: Tập lệnh trong 8051

Lệnh số học

STT	Cú pháp		Mô tả	Số byte	Số chu kỳ
	Mã lệnh	Toán hạng			
1	ADD	A, Rn	A = A + Rn	1	1
	ADD	A,direct	A = A + direct	2	1
	ADD	A,@Ri	A = A + @Ri	1	1
	ADD	A,#data	A = A + #data	2	1
	ADDC	A,Rn	A = A + Rn + C	1	1
6	ADDC	A,direct	A = A + direct + C	2	1
7	ADDC	A,@Ri	A = A + @Ri + C	1	1
8	ADDC	A,#data	A = A + #data + C	2	1
9	SUBB	A,Rn	A = A – Rn – C	1	1
10	SUBB	A,direct	A = A – direct – C	2	1
11	SUBB	A,@Ri	A = A – @Ri – C	1	1
12	SUBB	A,#data	A = A – #data – C	2	1
13	INC	A	A = A + 1	1	1
14	INC	Rn	Rn = Rn + 1	1	1
15	INC	Direct	direct = direct + 1	2	1
16	INC	@Ri	@Ri = @Ri + 1	1	1
17	DEC	A	A = A – 1	1	1
18	DEC	Rn	Rn = Rn – 1	1	1
19	DEC	Direct	direct = direct – 1	2	1
20	DEC	@Ri	@Ri = @Ri – 1	1	1
21	INC	Dptr	dptr = dptr + 1	1	2
22	MUL	AB	B:A = A*B	1	4
23	DIV	AB	A/B = A(thương) + B (dư)	1	4
24	DA	A	Hiệu chỉnh thập phân số liệu trong thanh ghi A	1	1

Lệnh logic

STT	Cú pháp		Mô tả	Số byte	Số chu kỳ
	Mã lệnh	Toán hạng			
1	ANL	A,Rn	A = (A)and(Rn)	1	1
2		A,direct	A = (A)and(direct)	2	1
3		A,@Ri	A = (A)and(@Ri)	1	1
4		A,#data	A = (A)and(#data)	2	1
5		direct,A	direct = (direct)and(A)	2	1

STT	Cú pháp		Mô tả	Số byte	Số chu kỳ
	Mã lệnh	Toán hạng			
6		Direct,#data	direct = (direct)and(#data)	3	2
7	ORL	A,Rn	A = (A)or(Rn)	1	1
8		A,direct	A = (A)or(direct)	2	1
9		A,@Ri	A = (A)or(@Ri)	1	1
10		A,#data	A = (A)or(#data)	2	1
11		direct,A	direct = (direct)or(A)	2	1
12		Direct,#data	direct = (direct)or(#data)	3	2
13	XRL	A,Rn	A = (A)xor(Rn)	1	1
14		A,direct	A = (A)xor(direct)	2	1
15		A,@Ri	A = (A)xor(@Ri)	1	1
16		A,#data	A = (A)xor(#data)	2	1
17		direct,A	direct = (direct)xor(A)	2	1
18		Direct,#data	direct = (direct)xor(#data)	3	2
19	CLR	A	A = 0	1	1
20	CPL	A	A = not(A)	1	1
21	RL	A	Quay trái A	1	1
22	RLC	A	Quay trái A qua cờ C	1	1
23	RR	A	Quay phải A	1	1
24	RRC	A	Quay phải A qua cờ C	1	1
25	SWAP	A	Hoán đổi 2 nửa của A	1	1

Các lệnh Bit

STT	Cú pháp		Mô tả	Số byte	Số chu kỳ
	Mã lệnh	Toán hạng			
1	CLR	C	Xóa cờ C về 0	1	1
2	CLR	Bit	Xóa bit về 0	2	1
3	SETB	C	Đặt cờ C = 1	1	12
4	SETB	Bit	Đặt bit = 1	2	1
5	CPL	C	Đảo giá trị của cờ C	1	1
6	CPL	Bit	Đảo giá trị của bit	2	1
7	ANL	C,bit	C = (C)and(bit)	2	2
8	ANL	C,/bit	C = (C)and(đảo của bit)	2	2
9	ORL	C,bit	C = (C)or(bit)	2	2
10	ORL	C,/bit	C = (C)or(đảo của bit)	2	2
11	MOV	C,bit	C = bit	2	1

STT	Cú pháp		Mô tả	Số byte	Số chu kỳ
	Mã lệnh	Toán hạng			
12	MOV	Bit,C	Bit = C	2	2
13	JC	<rel>	nhảy đến nhãn <rel> nếu C = 1	2	2
14	JNC	Bit, <rel>	nhảy đến nhãn <rel> nếu bit= 1	3	2
15	JB	Bit, <rel>	nhảy đến nhãn <rel> nếu bit= 1	3	2
16	JNB	Bit, <rel>	nhảy đến nhãn <rel> nếu bit= 0	3	2
17	JBC	Bit, <rel>	nhảy đến nhãn <rel> nếu bit = 1 và sau đó xóa luôn bit về 0	3	2

Các lệnh điều khiển chương trình

STT	Cú pháp		Mô tả	Số byte	Số chu kỳ
	Mã lệnh	Toán hạng			
1	ACALL	<addr11>	gọi chương trình con (nằm trong phạm vi 2k mem)	3	2
2	LCALL	<addr16>	gọi chương trình con (trong pham vi 64k mem)	3	2
3	RET		trở về từ chương trình con	1	2
4	RETI		trở về từ chương trình phục vụ ngắt	1	2
5	AJMP	<addr11>	nhảy đến nhãn (trong phạm vi 2k mem)	2	2
6	LJMP	<addr16>	nhảy đến nhãn (trong phạm vi 64 mem)	3	2
7	SJMP	<rel>	nhảy đến nhãn	2	2
8	JMP	@A+DPTR	nhảy đến địa chỉ = A+DPTR	1	2
9	JZ	<rel>	nhảy đến nhãn nếu A = 0	2	2
10	JNZ	<rel>	nhảy đến nhãn nếu A #0	2	2
11	CJNE	A,direct,<rel>	So sánh và nhảy đến nhãn nếu A # direct	3	2
12	CJNE	A,#data,<rel>	So sánh và nhảy đến nhãn nếu A#data	3	2
13		Rn,#data,<rel>	So sánh và nhảy đến	3	2

STT	Cú pháp		Mô tả	Số byte	Số chu kỳ
	Mã lệnh	Toán hạng			
			nhãn nếu Rn#data		
14		@Ri,#data,<rel>	So sánh và nhảy đến nhãn nếu byte có địa chỉ = Ri có nội dung khác với data	3	2
15	DJNZ	Rn,<rel>	Giảm Rn đi 1 và nhảy đến nhãn nếu chưa giảm về 0	2	2
16	DJNZ	direct,<rel>	Giảm direct đi 1 và nhảy đến nhãn nếu chưa giảm về 0	3	2
17	NOP		Không làm gì cả	1	1

PHỤ LỤC B: Chi tiết các thanh ghi chức năng trong 8051

❖ Bản tóm tắt chức năng các thanh ghi

1. Thanh ghi IE:

IE: Interrupt Enable, cho phép ngắt: thanh ghi này cho phép/cấm các ngắt hoạt động

EA	--	ET2	ES	ET1	EX1	ET0	EX0

2. Thanh ghi TCON: TCON Register - TCON (S:88h)

TCON: Timer/Counter Control Register: thanh ghi điều khiển bộ đếm/bộ định thời

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

3. Thanh ghi TMOD: TMOD Register - TMOD (S: 89h)

TMOD: Timer/Counter 0 and 1 Modes: thanh ghi đặt chế độ cho Timer/Counter 0 và 1

GATE1	C/T1#	M11	M01	GATE0	C/T0#	M10	M00

4. Thanh ghi T2CON: T2CON Register - T2CON (S:C8h)

Thanh ghi điều khiển Timer/Counter 2

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

5. Thanh ghi T2MOD: T2MOD Register - T2MOD (S:C9h)

Thanh ghi điều khiển Timer/Counter 2

-	-	-	-	-	-	T2OE	DCEN

6. Thanh ghi SCON

SCON: Serial Controller: thanh ghi cấu hình truyền thông nối tiếp.

FE/SM0	SM1	SM2	REN	TB8	RB8	TI	RI

7. Thanh ghi PCON: PCON Register

PCON - Power Control Register (87h): Thanh ghi điều khiển nguồn

SMOD1	SMOD0	-	POF	GF1	GF0	PD	IDL

❖ Diễn giải ý nghĩa các thanh ghi:

1. Thanh ghi phục vụ lập trình ngắt:

Thanh ghi **IE**: Interrupt Enable: Cho phép ngắt:

EA	--	ET2	ES	ET1	EX1	ET0	EX0

Nếu lập bít =1 thì cho phép ngắt, đặt bằng 0 thì cấm ngắt

- **EA**: Enable All : cho phép ngắt tất cả, phải đặt bít này bằng 1, thì mọi ngắt khác mới được phép hoạt động. Muốn ngắt nào hoạt động thì cho phép ngắt đó theo các bit dưới đây.
- **ET2**: Enable Timer 2: cho phép Timer 2 hoạt động
- **ES**: Enable Serial: Cho phép ngắt nối tiếp.
- **ET1**: Enable Timer 1: cho phép Timer 1 hoạt động
- **EX1**: Enable eXternal: cho phép ngắt ngoài 1
- **ET0**: Enable Timer 0: cho phép Timer 0 hoạt động
- **EX0**: Enable eXternal: cho phép ngắt ngoài 0

2. Thanh ghi TCON: TCON Register - TCON (S:88h)

Timer/Counter Control Register.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Bit thứ	Ký hiệu	Ý nghĩa
7	**TF1**	**Cờ tràn Timer 1** Được xóa bởi phần cứng khi vi xử lý nhảy đến chương trình con ngắt Lập bởi phần cứng khi Timer / Counter 1 tràn
6	**TR1**	**Bit điều khiển chạy Timer 1** Xóa để cấm chạy timer/counter 1. Lập để chạy timer/counter 1.
5	**TF0**	**Cờ tràn Timer 0** Xóa bằng phần cứng khi chạy chương trình con ngắt Lập bằng phần cứng khi thanh ghi timer/counter tràn
4	**TR0**	**Bit điều khiển chạy Timer 0** Xóa để cấm chạy timer/counter 0. Lập để chạy timer/counter 0.
3	**IE1**	**Cờ cạnh ngắt 1** Xóa bằng phần cứng khi ngắt vi xử lý, nếu đặt ngắt cạnh (sườn) Lập bằng phần cứng khi ngắt ngoài được phát hiện tại chân INT1
2	**IT1**	**Bit điều khiển loại ngắt ngoài 1** Xóa: Ngắt theo mức thấp cho ngắt ngoài 1 (INT1) Lập : Ngắt theo cạnh xuống (sườn xuống) cho ngắt ngoài 1 (INT1)
1	**IE0**	**Cờ cạnh ngắt 0** Xóa bằng phần cứng khi ngắt vi xử lý, nếu đặt ngắt cạnh (sườn) Lập bằng phần cứng khi ngắt ngoài được phát hiện tại chân INT0
0	**IT0**	**Bit điều khiển loại ngắt ngoài 0** Xóa: Ngắt theo mức thấp cho ngắt ngoài 1 (INT0) Lập : Ngắt theo cạnh xuống (sườn xuống) cho ngắt ngoài 1 (INT0)

Giá trị sau khi reset = 0000 0000b

Mode: chế độ

Timer : Bộ định thời, Counter: Bộ đếm

3. Thanh ghi TMOD: TMOD Register - TMOD (S: 89h)

TMOD - Timer/Counter 0 and 1 Modes.

GATE1	C/T1#	M11	M01	GATE0	C/T0#	M10	M00

Bit thứ	Ký hiệu	Ý nghĩa
7	GATE1	Bit điều khiển cổng Timer 1 Xóa để cho phép timer 1 mỗi khi bit TR1 lập Lập để cho phép timer 1 chỉ khi chân INT1# ở mức 1 và bit TR1 được lập
6	C/T1#	Bit lựa chọn Counter/Timer 1 Xóa: Timer 1 hoạt động: Bộ định thời, đếm tăng theo xung nhịp hệ thống Lập: Counter hoạt động: Bộ đếm, đếm tăng theo sườn xuống của chân T1.
5	M11	Các bit chọn chế độ cho Timer 1
4	M01	M11 M01 Chế độ hoạt động 0 0 Mode 0 8-bit Timer/Counter (TH1) với 5-bit(TL1) 0 1 Mode 1 16-bit Timer/Counter 1 0 Mode 2 8-bit Timer/Counter (TL1) tự nạp lại giá trị từ TH1 mỗi khi Timer/Counter tràn 1 1 Mode 3 Timer 1 dừng
3	GATE0	Bit điều khiển cổng Timer 0 Xóa để cho phép timer 0 mỗi khi bit TR0 lập Lập để cho phép timer 0 chỉ khi chân INT0# ở mức 1 và bit TR0 được lập
2	C/T0#	Bit lựa chọn Counter/Timer 0 Xóa: Timer 0 hoạt động: Bộ định thời, đếm tăng theo xung nhịp hệ thống Lập: Counter 0 hoạt động: Bộ đếm, đếm tăng theo sườn xuống của chân T0.
1	M10	Các bit chọn chế độ cho Timer 0
0	M00	M10 M00 Chế độ hoạt động 0 0 Mode 0 8-bit Timer/Counter (TH0) với 5-bit(TL0) 0 1 Mode 1 16-bit Timer/Counter 1 0 Mode 2 8-bit Timer/Counter (TL0) tự nạp lại giá trị từ TH0 mỗi khi Timer/Counter tràn 1 1 Mode 3 TL0 là bộ Timer/Counter 8 bit. TH0 là bộ Timer 8-bit sử dụng Timer các bit TR0 và TF0

Giá trị sau khi reset = 0000 0000b

Khi Timer 0 ở Mode 3, Timer 1 có thể được bật hoặc tắt bằng cách chuyển nó ra khỏi hoặc vào Mode 3, hoặc có thể vẫn dùng để tạo tốc độ Baud (đọc là bơ-u-d) cho cổng truyền thông nối tiếp, hoặc trong thực tế, trong một số ứng dụng không dùng ngắt.

4. Thanh ghi T2CON: T2CON Register - T2CON (S:C8h)

Thanh ghi điều khiển Timer 2

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

Bit thứ	Ký hiệu	Ý nghĩa
7	TF2	Cờ tràn Timer 2 TF2 không được lập nếu RCLK=1 hoặc TCLK=1 Phải xóa bằng phần mềm Lập bằng phần cứng khi Timer 2 tràn
6	EXF2	Cờ ngắt ngoài Timer 2 Lập khi có cạnh xuống tại chân T2EX, nếu EXEN2=1 Nếu ngắt Timer 2 hoạt động, mức 1 tại bit này, chương trình sẽ gọi ngắt Phải xóa bằng phần mềm
5	RCLK	Bit clock nhận Xóa để sử dụng cờ tràn Timer 1 làm xung clock nhận cho truyền thông nối tiếp trong Mode 1 hoặc 3 Lập để sử dụng cờ tràn Timer 2 làm xung clock nhận cho truyền thông nối tiếp trong Mode 1 hoặc 3
4	TCLK	Bit clock truyền Xóa để sử dụng cờ tràn Timer 1 làm xung clock phát cho truyền thông nối tiếp trong Mode 1 hoặc 3 Lập để sử dụng cờ tràn Timer 2 làm xung clock phát cho truyền thông nối tiếp trong Mode 1 hoặc 3
3	EXEN2	Bit cho phép ngắt ngoài Timer 2 Xóa để bỏ qua sự kiện tại chân T2EX cho hoạt động Timer 2 Lập sẽ xảy ra ngắt ngoài tại chân T2EX khi có sườn xuống, nếu Timer 2 không sử dụngcho truyền thông nối tiếp.
2	TR2	Bit điều khiển chạy Timer 2 Xóa : cấm Timer 2 chạy Lập: chạy Timer 2
1	C/T2#	Bit lựa chọn Timer/Counter 2 Xóa: Timer (nguồn xung hệ thống: Fosc) Lập: Counter (đầu vào từ chân T2)
0	CP/RL2#	Bit capture/Reload của Timer 2 Nếu RCLK=1 hoặc TCLK=1, CP/RL2# bị bỏ qua và Timer sẽ chạy ở chế độ tự nạp lại mỗi khi tràn Xóa: để tự nạp lại khi Timer 2 tràn hoặc khi có sườn xuống ở chân T2EX nếu EXEN2=1. Lập để bắt giữ (capture) khi có cạnh xuống ở chân T2EX nếu EXEN2=1

Giá trị sau khi reset = 0000 0000b

5. Thanh ghi T2MOD: T2MOD Register - T2MOD (S:C9h)

Thanh ghi chọn chế độ cho Timer 2

-	-	-	-	-	-	T2OE	DCEN

Bit thứ	Ký hiệu	Ý nghĩa
7	-	Dự trữ
6	-	Dự trữ
5	-	Dự trữ
4	-	Dự trữ
3	-	Dự trữ
2	-	Dự trữ
1	T2OE	Bit cho phép xuất Timer 2 Xóa để lập trình P1.0/T2 như đầu vào clock hoặc cổng I/O Lập để lập trình P1.0/T2 như đầu ra clock
0	DCEN	Bit cho phép đếm lùi Xóa: cấm Timer 2 đếm tăng/giảm Lập: cho phép Timer 2 đếm tăng/giảm

Giá trị sau khi reset = xxxx xx00b

6. Thanh ghi SCON

FE/SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Bit thứ	Ký hiệu	Ý nghĩa
7	FE	FE: Framing Error bit: bit báo truyền thông lỗi. (SMOD0=1) Xóa để reset trạng thái lỗi, không được xóa bởi một bit STOP đúng Lập bởi phần cứng khi phát hiện lỗi bit STOP không đúng SMOD0 phải được lập để cho phép truy cập đến bit FE
	SM0	Chế độ truyền thông nối tiếp
6	SM1	SM0 SM1 Mode Ý nghĩa Tốc độ Baud 0 0 0 Thanh ghi dịch $F_{CPU\ PERIPH}/6$ 0 1 1 8-bit UART Có thể thay đổi 1 0 2 9-bit UART $F_{CPU\ PERIPH}/32$ hoặc /16 1 1 3 39-bit UART Có thể thay đổi
5	SM2	Bit Mode 2 cổng nối tiếp / Bit cho phép truyền thông đa vi xử lý Xóa để cấm chức năng truyền thông đa vi xử lý Lập để cho phép chế độ truyền thông đa vi xử lý trong Mode 2 và 3, và thậm chí Mode 1. Bit này phải xóa nếu dùng Mode 0
4	REN	Bit cho phép nhận (Reception Enable bit) Xóa: cấm nhận nối tiếp Lập: cho phép nhận nối tiếp
3	TB8	Phát bit 8 / bit thứ 9 để truyền thông trong Mode 2 và 3 Xóa: truyền bit logic 0 trong bit thứ 9

		Lập: truyền bit logic 1 trong bit thứ 9
2	RB8	Nhận bit 8 / nhận bit thứ 9 trong mode 2 và 3 Xóa bằng phần cứng nếu bit thứ 9 nhận được là logic 0 Lập bằng phần cứng nếu bit thứ 9 nhận được là logic 1 Trong mode1, nếu SM2=0, RB8 là bit STOP nhận được. Trong mode 0 RB8 không sử dụng
1	TI	Cờ ngắt truyền Xóa để chấp nhận ngắt Lập bằng phần cứng tại thời gian cuối cùng của bit thứ 8 trong mode 0 hoặc bắt đầu của bit STOP trong các mode khác
0	RI	Cờ ngắt nhận Xóa để chấp nhận ngắt Lập bằng phần cứng tại thời gian cuối cùng của bit thứ 8 trong mode 0. Xem hình dưới để biết thêm trong các mode khác

Giá trị sau khi reset = 0000 0000b

7. Thanh ghi PCON: PCON Register

PCON - Power Control Register (87h)

SMOD1	SMOD0	-	POF	GF1	GF0	PD	IDL

Bit thứ	Ký hiệu	Ý nghĩa
7	SMOD1	Chế độ cổng nối tiếp, bit 1 cho USART Lập để lựa chọn tốc độ Baud gấp đôi trong mode 1,2 hoặc 3
6	SMOD0	Chế độ cổng nối tiếp, bit 0 cho USART Xóa để lựa chọn bit SM0 trong thanh ghi SCON Lập để lựa chọn bit FE trong thanh ghi SCON
5	—	Dự trữ
4	POF	Cờ tắt nguồn: Power Off Flag Xóa để nhận biết kiểu reset lần kế tiếp Lập bằng phần cứng khi VCC tăng từ 0 lên điện áp bình thường. cũng có thể đặt bằng phần mềm
3	GF1	Cờ mục đích chung, lập, xóa tùy lập trình viên
2	GF0	Cờ mục đích chung, lập, xóa tùy lập trình viên
1	PD	Bit chế độ nguồn giảm Xóa bằng phần cứng khi xảy ra reset Lập để vào chế độ nguồn giảm
0	**IDL**	Bit chế độ IDL Xóa bằng phần cứng khi xảy ra ngắt hoặc reset Lập để vào chế độ nghỉ (IDLE)

Giá trị sau khi reset = 00X1 0000b

Không thể định địa chỉ bit

PHỤ LỤC C: Ngắt

❖ **INT 21h / AH=1** - read character from standard input, with echo, result is stored in **AL**.

if there is no character in the keyboard buffer, the function waits until any key is pressed.

example:

```
mov ah, 1
int 21h
```

❖ **INT 21h / AH=2 - write character to standard output.**

entry: **DL** = character to write, after execution **AL = DL**.

example:

```
mov ah, 2
mov dl, 'a'
int 21h
```

❖ **INT 21h / AH=5 - output character to printer.**

entry: **DL** = character to print, after execution **AL = DL**.

example:

```
mov ah, 5
mov dl, 'a'
int 21h
```

❖ **INT 21h / AH=6 - direct console input or output.**

parameters for output: **DL** = 0..254 (ascii code)

parameters for input: **DL** = 255

for output returns: AL = DL

for input returns: **ZF** set if no character available and **AL = 00h**, **ZF** clear if character available.

AL = character read; buffer is cleared.

example:

```
mov ah, 6
mov dl, 'a'
int 21h      ; output character.

mov ah, 6
mov dl, 255
int 21h      ; get character from keyboard buffer (if any) or set ZF=1.
```

❖ **INT 21h / AH=7 - character input without echo to AL.**

if there is no character in the keyboard buffer, the function waits until any key is pressed.

example:

```
mov ah, 7
int 21h
```

❖ **INT 21h / AH=9 - output of a string at DS:DX. String must be terminated by '$'.**

example:

```
org 100h
mov dx, offset msg
mov ah, 9
int 21h
ret
msg db "hello world $"
```

❖ **INT 21h / AH=0Ah - input of a string to DS:DX.**

fist byte is buffer size, second byte is number of chars actually read. this function does **not** add '$' in the end of string. to print using **INT 21h** / **AH=9** you must set dollar character at the end of it and start printing from address **DS:DX + 2**.

example:

```
org 100h
mov dx, offset buffer
mov ah, 0ah
int 21h
jmp print
buffer db 10,?, 10 dup(' ')
print:
xor bx, bx
mov bl, buffer[1]
mov buffer[bx+2], '$'
mov dx, offset buffer + 2
mov ah, 9
int 21h
ret
```

the function does not allow to enter more characters than the specified buffer size.

see also **int21.asm** in c:\emu8086\examples

Danh mục hình ảnh

Danh mục mã nguồn

Danh mục bảng

Chỉ mục