HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

*****

PHẠM HOÀNG DUY

BÀI GIẢNG

KỸ THUẬT VI XỬ LÝ

HÀ NỘI 06-2010

Lời nói đầu

Các bộ vi xử lý đóng vai trò quan trọng trong các hệ thống số và chúng được sử dụng trong rất nhiều ứng dụng như các hệ thống điều khiển, hệ thống thông tin liên lạc. Tài liệu này giới thiệu các khái niệm căn bản của hệ vi xử lý và tập trung trình bày vi xử lý Intel 8086 và ghép nối tiêu biểu để tạo nên hệ vi xử lý 8086, tiền than của các hệ vi xử lý x86 sau này. Các kiến thức thu nhận được từ việc xây dựng hệ vi xử lý 8086 cũng sẽ rất bổ ích cho việc phát triển các hệ vi xử lý phức tạp hơn cũng như các hệ thống nhúng.

Cấu trúc của tài liệu như sau.

Chương 1 giới thiệu các khái niệm tổng quan của hệ vi xử lý và các bộ phận căn bản cấu thành hệ vi xử lý nói chung. Chương này cũng tóm tắt quá trình phát triển và phân loại các bộ vi xử lý đến nay.

Chương 2 trình bày chi tiết về vi xử lý Intel 8086 bao gồm sơ đồ khối và cách tổ chức bộ nhớ. Ngoài ra, chương này giới thiệu tập lệnh x86 và quá trình thực hiện lệnh.

Chương 3 cung cấp các kiến thức căn bản để lập trình với vi xử lý 8086 bằng cách giới thiệu các cấu trúc chương trình và các cấu trúc rẽ nhánh và lặp tiêu biểu kết hợp với các ví dụ.

Chương 4 tập trung giới thiệu cách thức ghép nối vi xử lý 8086 với các thiết bị khác để tạo thành hệ vi xử lý căn bản. Chương này trình bày chu trình đọc/ghi của vi xử lý 8086. Đây là cơ sở để tiến hành ghép nối dữ liệu với các thiết bị khác như bộ nhớ hay các thiết bị vào/ra khác. Chương này giới thiệu cơ chế truyền thông nối tiếp và cách thức ghép nối với vi xử lý 8086.

Chương 5 cung cấp các kiến thức căn bản về các kỹ thuật trao đổi dữ liệu với các thiết bị ghép nối với hệ vi xử lý nói chung bao gồm vào/ra thăm dò (lập trình), vào/ra sử dụng ngắt và vào/ra trực tiếp bộ nhớ. Trong ba phương pháp, vào/ra trực tiếp bộ nhớ cho phép trao đổi khối lượng dữ liệu lớn với tốc độ cao và cần có vi mạch đặc biệt. Chương này cũng giới thiệu vi mạch trợ giúp cho các phương pháp vào ra như vi mạch điều khiển ngắt, vi mạch điều khiển vào ra trực tiếp bộ nhớ.

Chương 6 trình bày sơ bộ các khái niệm về các hệ vi điều khiển (hay hệ vi xử lý trên một vi mạch). Chương này còn cung cấp các thông tin căn bản về hệ vi điều khiển Intel 8051 và một số ứng dụng.

Chương 7, chương cuối cùng, giới thiệu một số bộ vi xử lý tiên tiến của Sun Microsystems và Intel dựa trên kiến trúc IA-32 và IA-64.

Tài liệu được biên soạn dựa trên tham khảo các tài liệu đặc biệt là cuốn "Kỹ thuật Vi xử lý" của tác giả Văn Thế Minh và dựa trên trao đổi kinh nghiệm giảng dạy với các đồng nghiệp và phản hồi của sinh viên tại Học viện Công nghệ Bưu chính Viễn thông. Tài liệu có thể được dùng làm tài liệu học tập cho sinh viên đại học, cao đẳng ngành công nghệ thông tin. Trong quá trình biên soạn, dù đã có nhiều cố gắng song không tránh khỏi thiếu sót, nhóm tác giả mong nhận được các góp ý cho các thiếu sót cũng như ý kiến cập nhật và hoàn thiện nội dung của tài liệu.

Hà nội, 06/2010

Tác giả

Mục lục

Chương I. Tổng quan về vi xử lý và hệ vi xử lý

I.1 Giới thiệu về vi xử lý

Một máy tính thông thường bao gồm các khối chức năng cơ bản như: khối xử lí trung tâm CPU (Central Processing Unit), bộ nhớ, và khối phối ghép với thiết bị ngoại vi (I/O, input/output). Tuỳ theo quy mô, độ phức tạp hiệu năng của các khối chức năng kể trên mà người ta phân các máy tính điện tử đã và đang sử dụng ra thành các loại sau:

Máy tính lớn (Mainframe) là loại máy tính được thiết kế để giải các bài toán lớn với tốc độ nhanh. Máy tính này thường làm việc với số liệu từ 64 bít hoặc lớn hơn nữa và được trang bị nhiều bộ xử lý tốc độ cao và bộ nhớ rất lớn. Chính vì vậy máy tính cũng lớn về kích thước vật lý. Chúng thường được dùng để tính toán điều khiển các hệ thống thiết bị dùng trong quân sự hoặc các hệ thống máy móc của chương trình nghiên cứu vũ trụ, để xử lý các thông tin trong ngành ngân hàng, ngành khí tượng, các công ty bảo hiểm. . . Loại máy lớn nhất trong các máy lớn được gọi là *supercomputer* (như loại máy Y-MP/832 của Cray).

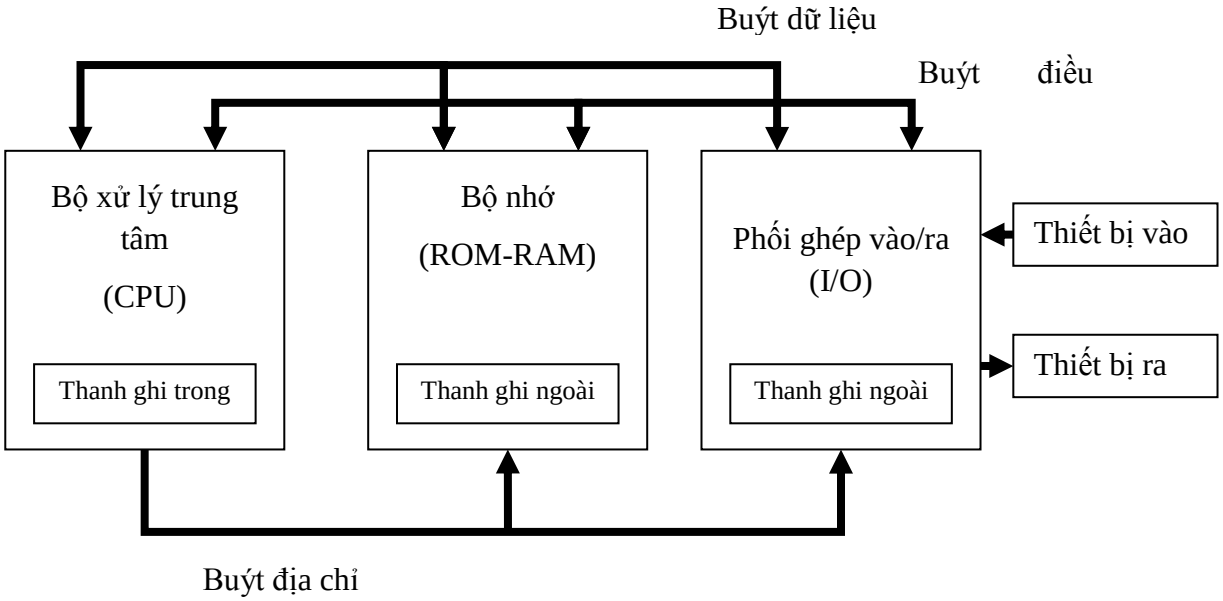
Máy tính con (Minicomputer) là một dạng thu nhỏ về kích thước cũng như về tính năng của máy tính lớn. Nó ra đời nhằm thoả mãn các nhu cầu sử dụng máy tính cho các ứng dụng vừa phải mà nếu dùng máy tính lớn vào đó thì sẽ gây lãng phí. Máy tính con thường được dùng cho các tính toán khoa học kỹ thuật, gia công dữ liệu quy mô nhỏ hay để điều khiển quy trình công nghệ. Tiêu biểu cho nhóm này là loại máy VAX 6360 của Digital Equipment Corporation và MV/8000II của Data genaral.

Máy vi tính (Microcomputer) là loại máy tính rất thông dụng hiện nay. Một máy vi tính có thể là một bộ vi điều khiển (microcontroller), một máy vi tính trong một vi mạch (one-chip microcomputer), và một hệ vi xử lí có khả năng làm việc với số liệu có độ dài 1 bít, 4 bít, 8 bít, 16 bít hoặc lớn hơn. Hiện nay một số máy vi tính có tính năng có thể so sánh được với máy tính con, làm việc với số liệu có độ dài từ là 32 bít (thậm chí là 64 bít). Ranh giới để phân chia giữa máy vi tính và máy tính con chính vì thế ngày càng không rõ nét.

Các bộ vi xử lý hiện có tên thị trường thường được xếp theo các họ phụ thuộc vào các nhà sản xuất và chúng rất đa dạng về chủng loại. Các nhà sản xuất vi xử lý nổi tiếng có thể kể tới Intel với các sản phẩm x86, Motorola với 680xx, Sun Microsystems với SPARC. Tính đến thời điểm hiện nay các chương trình viết cho tập lệnh x86 của Intel chiếm tỷ lệ áp đảo trong môi trường máy vi tính.

I.2 Hệ vi xử lý

Bộ vi xử lý là một thành phần rất cơ bản, không thiếu được để tạo nên máy vi tính. Trong thực tế bộ vi xử lý còn phải có thể kết hợp thêm với các bộ phận điện tử khác như bộ nhớ và bộ phối ghép vào/ra để tạo nên một *hệ vi xử lý* hoàn chỉnh. Cần lưu ý rằng, để chỉ một hệ thống có cấu trúc như trên, thuật ngữ "hệ vi xử lý" mang ý nghĩa tổng quát hơn so với thuật ngữ "máy vi tính", vì máy vi tính chỉ là một ứng dụng cụ thể cảu hệ vi xử lý. Hình I-1 giới thiệu sơ đồ khối tổng quát của một hệ vi xử lý.



Hình I-1. Sơ đồ khối của hệ vi xử lý

Trong sơ đồ này ta thấy rõ các khối chức năng chính của hệ vi xử lý gồm:

- Khối xử lý trung tâm (Central Processing unit, CPU)
- Bộ nhớ bán dẫn (ROM-RAM)
- Khối phối ghép với các thiết bị ngoại vi (Input/Output - I/O)
- Các buýt truyền thông tin.

Ba khối chức năng đầu liên hệ với nhau thông qua qập các đường day để truyền tín hiệu gọi chung là *Buýt hệ thống.* Buýt hệ thống bao gồm 3 buýt thành phần ứng với các tín hiệu địa chỉ, dữ liệu và điều khiển ta có *buýt địa chỉ, buýt dữ liệu và buýt điều khiển.*

CPU đóng vai trò chủ đạo trong hệ vi xử lý. Đây là một mạch vi điện tử có độ tích hợp rất cao. Khi hoạt động, CPU *đọc mã lệnh* được ghi dưới dạng các bít 0 và bít 1 từ bộ nhớ, sau đó sẽ *giải mã* các lệnh này thành các dãy xung điều khiển ứng với các thao tác trong lệnh để điều khiển các khối khác *thực hiện* từng bước các thao tác đó. Để làm được việc này bên trong CPU có thanh ghi dùng để chứa địa chỉ của lệnh sắp thực hiện gọi là *thanh ghi con trỏ lệnh (Instruction Pointer, IP)* hoặc *bộ đếm chương trình (Program Counter, PC),* một số thanh ghi đa năng khác cùng *bộ tính toán số học và lô-gíc (Arithmetic Logic Unit ALU)* để

thao tác với dữ liệu. Ngoài ra ở đây còn có các hệ thống mạch điện tử rất phức tạp để *giải mã lệnh* và từ đó tạo ra các xung điều khiển cho toàn hệ.

Bộ nhớ bán dẫn hay còn gọi là *bộ nhớ trong* là một bộ phận khác rất quan trọng của hệ vi xử lý. Tại đây (trong ROM) ta có thể chứa chương trình điều khiển hoạt động của toàn hệ để khi bật điện thì CPU có thể lấy lệnh từ đây để khởi động hệ thống. Một phần của chương trình điều khiển hệ thống, các chương trình ứng dụng, dữ liệu cùng các kết quả của chương trình thường được đặt trong RAM. Các dữ liệu và chương trình muốn lưu trữ lâu dài hoặc có dung lượng lớn sẽ được đặt trong bộ nhớ ngoài.

Khối phối ghép vào/ra (I/O) tạo ra khả năng giao tiếp giữa hệ vi xử lý với thế giới bên ngoài. Các thiết bị ngoại vi như bàn phím, chuột, màn hình, máy in, chuyển đổi số/tương tự (D/A Converter, DAC) và chuyển đổi tương tự/số (A/D Converter, ADC), ổ đĩa từ. . . đều liên hệ với bộ vi xử lý qua bộ phận này. Bộ phận phối ghép cụ thể giữa buýt hệ thống với thế giới bên ngoài thường được gọi là *cổng.* Như vậy ta sẽ có các *cổng vào* để lấy thông tin từ ngoài vào và các *cổng ra* để đưa thông tin từ trong ra. Tùy theo nhu cầu cụ thể của công việc, các mạch cổng này có thể được xây dựng từ các mạch lôgic đơn giản hoặc từ các vi mạch chuyên dụng lập trình được.

Buýt địa chỉ (address bus) thường có từ 16, 20, 24, 32 hay 64 đường dây song song chuyển tải thông tin của các bít địa chỉ. Khi đọc/ghi bộ nhớ CPU sẽ đưa ra trên buýt này địa chỉ của ô nhớ liên quan. *Khả năng phân biệt địa chỉ* (số lượng địa chỉ cho ô nhớ mà CPU có quản lý được) phụ thuộc vào số bít của buýt địa chỉ. Ví dụ nếu một CPU có số đường dây địa chỉ là N=16 thì nó có khả năng địa chỉ hóa được 2^N = 65536 =64 kilô ô nhớkhác nhau (1K= 2^{10} =1024). Khi đọc/ghi với cổng vào/ra CPU cũng đưa ra trên buýt địa chỉ các bít địa chỉ tương ứng của cổng. Trên sơ đồ khối ta dễ nhận ra tính *một chiều của buýt địa chỉ* qua một chiều của mũi tên. Chỉ có CPU mới có khả năng đưa ra địa chỉ trên buýt địa chỉ.

Buýt dữ liệu (data bus) thường có từ 8, 16, 20, 24, 32, 64 (hoặc hơn) đường dây tùy theo các bộ vi xử lý cụ thể. Số lượng đường dây này quyết định số bít dữ liệu mà CPU có khả năng xử lý cùng một lúc. Chiều mũi tên trên sus số liệu chỉ ra rằng đây là *buýt 2 chiều,* nghĩa là dữ liệu có thể truyền đi từ CPU (*dữ liệu ra)* hoặc truyền đến CPU (*dữ liệu vào).* Các phần tử có đầu ra nối thẳng với buýt dữ liệu đều phải được trang bị *đầu ra 3 tạng thái* để có thể ghép vào được và hoạt động bình thường với buýt này.

Buýt điều khiển (control bus) thường gồm hàng chục đường dây tín hiệu khác nhau. Mỗi tín hiệu điều khiển có *một chiều nhất định* vì khi hoạt động CPU đưa tín hiệu điều khiển tới các khối khác trong hệ. Đồng thời CPU cũng nhận tín hiệu điều khiển từ các khối đó để phối hợp hoạt động của toàn hệ. Các tín hiệu này trên hình vẽ được thể hiện bởi các đường có mũi tên 2 chiều, điều đó không phải là để chỉ tính hai chiều của một tín hiệu mà là tính hai chiều của cả một nhóm các tín hiệu.

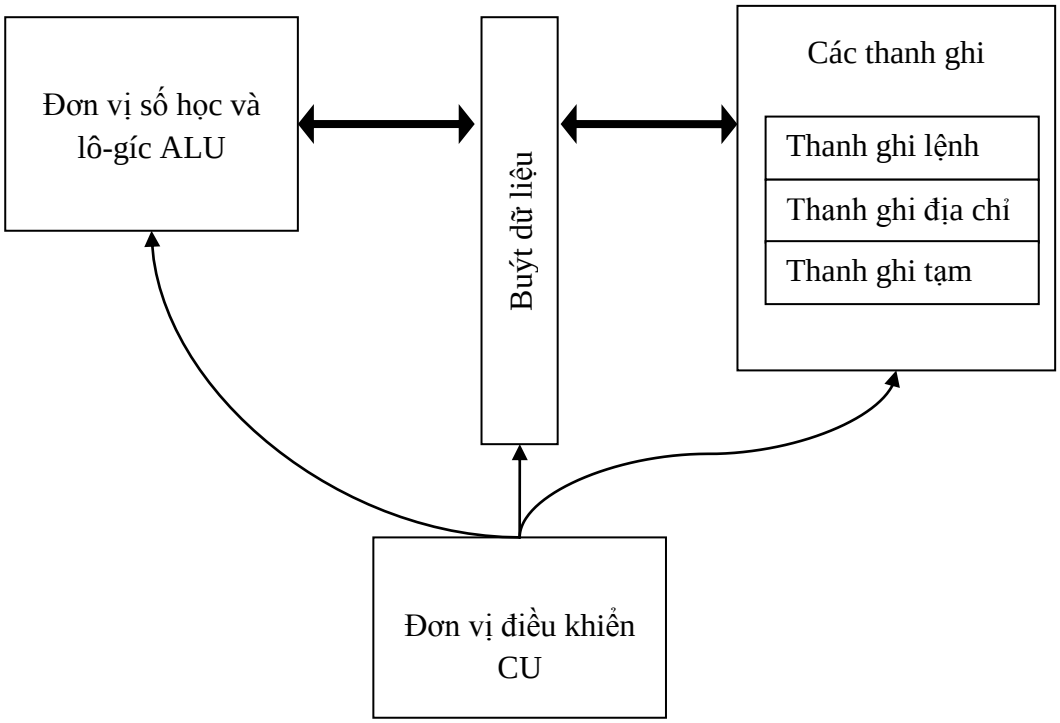
Mặt khác, hoạt động của hệ thống vi xử lý trên cũng có thể coi như là quá trình trao đổi dữ liệu giữa các thanh ghi bên trong. Về mặt chức năng mỗi khối trong hệ thống trên tương đương với *các thanh ghi trong* (nằm trong CPU) hoặc *các thanh ghi ngoài* (nằm rải rác trong bộ nhớ ROM, bộ nhớ RAM và trong khối phối ghép vào/ra). Hoạt động của toàn hệ thực chất

là sự phối hợp hoạt động của các thanh ghi trong và ngoài nói trên để thực hiện *sự biến đổi dữ liệu* hoặc *sự trao đổi dữ liệu* theo các yêu cầu đã định trước.

I.3 Các đặc điểm cấu trúc của vi xử lý

I.3.1 Cấu trúc căn bản

Như đã trình bày trong phần trên, vi xử lý chính là đơn vị xử lý trung tâm CPU của máy vi tính. Như vậy sức mạnh xử lý của máy vi tính được quyết định bởi năng lực của vi xử lý. Trên nguyên tắc, vi xử lý có thể được chia thành các đơn vị chức năng chính như trong Hình I-2.



Hình I-2. Sơ đồ khối chức năng vi xử lý

I.3.1.1 Các thanh ghi

Số lượng, kích cỡ và kiểu của các thanh ghi thay đổi từ vi xử lý này sang vi xử lý khác. Tuy nhiên, các thanh ghi này thực hiện các thao tác tương tự nhau. Cấu trúc các thanh ghi đóng vai trò quan trọng trong việc thiết kế kiến trúc của vi xử lý. Đồng thời, cấu trúc thanh ghi với một loại vi xử lý cụ thể cho biết mức độ thuận lợi và dễ dùng khi lập trình cho vi xử lý đó. Dưới đây là các thanh ghi cơ bản nhất:

i. Thanh ghi lệnh: lưu các lệnh. Sau khi nạp mã lệnh từ bộ nhớ, vi xử lý lưu mã lệnh trong thanh ghi lệnh. Giá trị trong thanh ghi này luôn được vi xử lý giải mã để xác định lệnh. Kích cỡ từ (word) của vi xử lý quyết định kích cỡ của thanh ghi này. Ví dụ, vi xử lý 32 bít thì sẽ có thanh ghi lệnh 32 bít.

ii. Bộ đếm chương trình: chứa địa chỉ của lệnh hay mã thực thi (op-code). Thông thường, thanh ghi này chứa địa chỉ của câu lệnh kế. Thanh ghi này có đặc điểm sau:
    1. Khi khởi động lại, địa chỉ của lệnh đầu tiên được thực hiện được nạp vào thanh ghi này.
    2. Để thực hiện lệnh, vi xử lý nạp nội dung của bộ đếm chương trình vào buýt địa chỉ và đọc ô nhớ ở địa chỉ đó. Giá trị của bộ đếm chương trình tự động tăng theo bộ lô-gíc trong của vi xử lý. Như vậy, vi xử lý thực hiện các lệnh tuần tự trừ phi chương trình có các lệnh làm thay đổi trật tự tính toán.
    3. Kích cỡ của bộ đếm chương trình phụ thuộc vào kích cỡ của buýt địa chỉ.
    4. Nhiều lệnh làm thay đổi nội dung của thanh ghi này so với trình tự thông thường. Khi đó, giá trị của thanh ghi được xác định thông qua địa chỉ chỉ định trong các lệnh này.

iii. Thanh ghi địa chỉ bộ nhớ: chứa địa chỉ của dữ liệu. Vi xử lý sử dụng các địa chỉ này như là các con trỏ trực tiếp tới bộ nhơ. Giá trị của các địa chỉ này chính là dữ liệu đang được trao đổi và xử lý.

iv. Thanh ghi dùng chung: còn được gọi là thanh ghi gộp (accumulator). Thanh ghi này thường là các thanh ghi 8 bít dùng thể lưu hầu hết các kết quả tính toán của đơn vị xử lý sô học và lô-gíc ALU. Thanh ghi này còn dùng để trao đổi dữ liệu với các thiết bị vào/ra.

I.3.1.2 Đơn vị xử lý số học và lô-gíc ALU

ALU thực hiện tất cả các thao tác xử lý dữ liệu bên trong vi xử lý như là các phép toán lô-gíc, số học. Kích cỡ thanh ghi ALU tương ứng với kích cỡ từ của vi xử lý. Vi xử lý 32 bít sẽ có ALU 32 bít. Một vài chức năng tiêu biểu của ALU:

1. Cộng nhị phân và các phép lô-gíc
2. Tính số bù một của dữ liệu
3. Dịch hoặc quay trái phải các thanh ghi dùng chung.

I.3.1.3 Đơn vị điều khiển CU

Chức năng chính của đơn vị điều khiển CU là đọc và giải mã các lệnh từ bộ nhớ chương trình. Để thực hiện lệnh, CU kích hoạt khối phù hợp trong ALU căn cứ vào mã lệnh (op-code) trong thanh ghi lệnh. Mã lệnh xác định thao tác để CU thực thi. CU thông dịch nội dung của thanh ghi lệnh và sau đó sinh ra một chuỗi các tín hiệu kích hoạt tương ứng với lệnh nhận được. Các tín hiệu này kích hoạt các khối chức năng phù hợp bên trong ALU.

CU sinh ra các tín hiệu điều khiển dẫn tới các thành phần khác của vi xử lý qua buýt điều khiển. Ngoài ra, CU cũng đáp ứng lại các tín hiệu điều khiển trên buýt điều khiển do các bộ phận khác gửi tới. Các tín hiệu này thay đổi theo từng loại vi xử lý. Một số tín hiệu điều khiển tiêu biểu như khởi động lại RESET, đọc ghi (R/W), tín hiệu ngắt (INT/IRQ), …

I.3.1.4 Thực hiện chương trình

Để chạy chương trình, vi xử lý thường lặp lại các bước sau để hoàn thành từng lệnh:

1. Nạp (Fetch). Vi xử lý nạp (đọc) lệnh từ bộ nhớ chính vào thanh ghi lệnh

2. Giải mã (Decode). Vi xử lý giải mã hay dịch lệnh nhờ đơn vị điều khiển CU. CU nhập nội dung của thanh ghi lệnh và giải mã để xác định kiểu lệnh.
3. Thực hiện (Execute). Vi xử lý thực hiện lệnh nhờ CU. Để hoàn thành nhiệm vụ, CU sinh ra một chuỗi các tín hiệu điều khiển tương ứng với lệnh.

Quá trình trên được lặp đi lặp lại cho đến câu lệnh cuối cùng của chương trình. Trong các vi xử lý tiên tiến quá trình thực hiện lệnh được cải tiến cho phép nhiều lệnh được thực hiện xen kẽ với nhau. Tức là, câu lệnh kế tiếp sẽ được thực hiện mà không cần chờ câu lệnh hiện thời kết thúc. Kỹ thuật trên được gọi là kỹ thuật đường ống (pipeline). Việc thực hiện xen kẽ cho phép nâng cao tốc độ thực hiện của vi xử lý và làm giảm thời gian chạy chương trình.

I.3.2 Kiến trúc RISC và CISC

Có hai kiển kiến trúc vi xử lý: máy tính với tập lệnh rút gọn (Reduced Instruction Set Computer-RISC) và máy tính với tập lệnh phức tạp (Complex Instruction Set Computer-CISC). Vi xử lý RISC nhấn mạnh tính đơn giản và hiệu quả. Các thiết kế RISC khởi đầu với tập lệnh thiết yếu và vừa đủ. RISC tăng tốc độ xử lý bằng cách giảm số chu kỳ đồng hồ trên một lệnh. Mục đích của RISC là tăng tốc độ hiệu dụng bằng cách chuyển việc thực hiện các thao tác không thường xuyên vào phần mềm còn các thao tác phổ biến do phần cứng thực hiện. Như vậy làm tăng hiệu năng của máy tính. Các đặc trưng căn bản của vi xử lý kiểu RISC:

1. Thiết kế vi xử lý RISC sử dụng điều khiển cứng (hardwired control) không hoặc rất ít sử dụng vi mã. Tất cả các lệnh RISC có định dạng cố định vì vậy việc sử dụng vi mã không cần thiết.
2. Vi xử lý RISC xử lý hầu hết các lệnh trong một chu kỳ.
3. Tập lệnh của vi xử lý RISC chủ yếu sử dụng các lệnh với thanh ghi, nạp và lưu. Tất cả các lệnh số học và lô-gíc sử dụng thanh ghi, còn các lệnh nạp và lưu dùng để truy nhập bộ nhớ.
4. Các lệnh có một định dạng cố định và ít chế độ địa chỉ.
5. Vi xử lý RISC có một số thanh ghi dùng chung.
6. Vi xử lý RISC xử lý một vài lệnh đồng thời và thường áp dụng kỹ thuật đường ống (pipeline).

Vi xử lý RISC thường phù hợp với các ứng dụng nhúng. Vi xử lý hay bộ điều khiển nhúng thường được nhúng trong hệ thống chủ. Nghĩa là, các thao tác của các bộ điều khiển này thường được che dấu khỏi hệ thống chủ. Ứng dụng điều khiển tiêu biểu cho ứng dụng nhúng là hệ thống tự động hóa văn phòng như máy in lade, máy đa chức năng. Vi xử lý RISC cũng rất phù hợp với các ứng dụng như xử lý ảnh, rô-bốt và đồ họa nhờ có mức tiêu thụ điện thấp, thực thi nhanh chóng.

Mặt khác, vi xử lý CISC bao gồm số lượng lớn các lệnh và nhiều chế độ địa chỉ mà nhiều kiểu rất ít được sử dụng. Với CISC hầu hết các lệnh đều có thể truy nhập bộ nhớ trong khi đó RISC chỉ có các lệnh nạp và lưu. Do tập lệnh phức tạp, CISC cần đơn vị điều khiển phức tạp và vi chương trình. Trong khi đó, RISC sử dụng bộ điều khiển kết nối cứng nên nhanh hơn. Kiến trúc CISC khó triển khai kỹ thuật đường ống.

Ưu điểm của CISC là các chương trình phức tạp có thể chỉ cần vài lệnh với vài chu trình nạp còn RISC cần một số lượng lớn các lệnh để thực hiện cùng nhiệm vụ. Tuy nhiên, RISC có thể cải thiện hiệu năng đáng kể nhờ xung nhịp nhanh hơn, kỹ thuật đường ống và tối ưu hóa quá trình biên dịch. Hiện nay, các vi xử lý CISC sử dụng phương pháp lai, với các lệnh đơn giản CISC sử dụng cách tiếp cận của RISC để thực thi xen kẽ (kỹ thuật đường ống) với các câu lệnh phức tạp sử dụng các vi chương trình để đảm bảo tính tương thích.

I.3.3 Các đặc điểm

Từ cấu trúc căn bản của vi xử lý, có thể rút ra các đặc điểm cấu trúc như sau:

- Tốc độ xung nhịp. Vi xử lý là thiết bị số nên sử dụng tín hiệu xung nhịp (clock) để đồng bộ các hoạt động của mình. Tốc độ xung nhịp càng lớn vi xử lý chạy càng nhanh.
- Khối lượng dữ liệu xử lý được: thể hiện qua kích cỡ các thanh ghi dữ liệu. Với kích cỡ thanh ghi dữ liệu là 32 bít, vi xử lý có khả năng đọc/ghi 4 byte cho mỗi thao tác với bộ nhớ.
- Dung lượng bộ nhớ trực tiếp: thể hiện qua dung lượng thanh ghi địa chỉ. Với dung lượng 32 bít, vi xử lý có thể quản lý trực tiếp 4GB bộ nhớ.
- Năng lực tính toán: được quyết định bởi năng lực của bộ số học và lô-gíc. Bên cạnh các thao tác số học thông thường cần có các đơn vị chức năng phục vụ các yêu cầu chuyên biệt khác như đơn vị xử lý dấu phẩy động cho các tinh toán số thực.
- Khả năng thực hiện lệnh: thể hiện năng lực và độ phức tạp của đơn vị điều khiển. Đơn vị này có thể cho phép quá trình xử lý tuần tự đơn giản hay phức tạp như xen kẽ các lệnh nhằm nâng cao hiệu năng của vi xử lý trên chu kỳ lệnh. Các thiết kế phức tạp cho phép đơn vị điều khiển thực hiện nhiều lệnh trong một chu trình.

I.4 Lịch sử phát triển và phân loại các bộ vi xử lý

Phần này giới thiệu quá trính phát triển của các bộ vi xử lý qua các giai đoạn từ năm 1971 tập chung chủ yếu vào các sản phẩm của hãng Intel do đây là một trong những hãng sản xuất vi xử lý hàng đầu đồng thời cũng là hãng triển khai nhiều công nghệ mới giúp nâng cao hiệu năng của vi xử lý đặc biệt trong lĩnh vực máy vi tính.

I.4.1 Giai đoạn 1971-1973

Năm 1971, trong khi phát triển các vi mạch dùng cho máy tính cầm tay, Intel đã cho ra đời bộ vi xử lý đầu tiên là 4004 (4 bít) của Rockwell International, IPM-16 (16 bít) của National Semiconductor.

Đặc điểm chung của các vi xử lý thế hệ này là:

- Độ dài từ thường là 4 bít (cũng có thể dài hơn)
- Công nghệ chế tạo PMOS với đặc điểm mật độ phần tử nhỏ, tốc độ thấp, giá thành rẻ và có khả năng đưa ra dòng tải nhỏ.
- Tốc độ thực hiện lệnh: 10-16μs/lệnh với tần số đồng hồ f_{clk} = 0, 1- 0, 8 MHz.

- Tập lệnh đơn giản phải cần nhiều mạch phụ trợ mới tạo nên một hệ vi xử lý hoàn chỉnh.

I.4.2 Giai đoạn 1974-1977

Các bộ vi xử lý đại diện trong thế hệ này là các vi xử lý 8 bít 6502 của MOS Technology, 6800 và 6809 của Motorola, 8080 và 8085 của Intel và đặc biệt là bộ vi xử lý Z80 của Zilog. Các bộ vi xử lý này có tập lệnh phong phú hơn và thường có khả năng phân biệt địa chỉ bộ nhớ với dung lượng đến 64KB. Có một số bộ vi xử lý còn có khả năng phân biệt được 256 địa chỉ cho các thiết bị ngoại vi (họ Intel và Zilog). Chúng đã được sử dụng rộng rãi trong công nghiệp và nhất là để tạo ra các máy tính 8 bít nổi tiếng một thời như Apple II và Commodore 64. Tất cả các bộ vi xử lý thời kì này đều được sản xuất bằng công nghệ NMOS (Với mật độ điện tủ trên một đơn vị diện tích cao hơn so với công nghệ PMOS) hoặc CMOS (tiết kiệm điện năng tiêu thụ) cho phép đạt được tốc độ từ 1-8 μs/lệnh với tần số đồng hồ f_{clk} = 1-5 MHz.

I.4.3 Giai đoạn 1978-1982

Các bộ vi xử lý trong thế hệ này có đại diện là các bộ vi xử lý 16 bít 8086/80186/80286 của Intel hoặc 86000/86010 của Motorola. Một điều tiến bộ hơn hẳn so với các bộ vi xử lý 8 bít thế hệ trước là các bộ vi xử lý 16 bít có tập lệnh đa dạng với các lệnh nhân, lệnh chia và các lệnh thao tác với chuỗi kí tự. Khả năng phân biệt địa chỉ cho bộ nhớ hoặc cho thiết bị ngoại vi của các vi xử lý thế hệ này cũng lớn hơn (từ 1MB đến 16 MB cho bộ nhớ và tới 64 K địa chỉ cho thiết bị ngoại vi đối với họ Intel). Đây là các bộ vi xử lý được dùng trong các máy IBM PC, PC/XT, PC/AT và các máy Macintosh của Apple. Phần lớn các bộ vi xử lý trong thế hệ này đều được sản xuất bằng công nghệ HMOS và cho phép đạt được tốc độ từ 0, 1-1μs/lệnh với tần số đồng hồ f_{clk} =5-10 MHz.

I.4.4 Giai đoạn 1983-1999

Các bộ vi xử lý đại diện trong thế hệ này là các vi xử lý 32 bít 80386/80486 và 64 bít Pentium của Intel gồm có Pentium Pro với thiết kế bộ đệm trên cùng vi mạch xử lý, Pentium MMX với các mở rộng cho đa phương tiện, Pentium II, Pentium III. Song song với các hệ vi xử lí của hãng Intel, hãng Motorola cũng đưa ra các vi xử lý 32 bít 68020/68030/68040 và các vi xử lí 64 bít 68060/64. Đặc điểm của các bộ vi xử lý có số lượng transistor rất lớn (từ vài 3 triệu đến trên 50 triệu transistor. Phần lớn các bộ vi xử lí mới thực hiện nhiều hơn 1 lệnh trong một chu kỳ, và tích hợp đơn vị xử lí dấu phẩy động FPU (Floating-Point Unit). Chúng có các thanh ghi dùng chung 16-32 bít. Nhiều loại có phân biệt các tệp thanh nghi 32-bít (register file) cho đơn vị nguyên IU (interger unit) và tệp thanh ghi 32- bít cho FPU. Chúng có bộ nhớ đệm bên trong mức 1 với dung lượng lên tới 64 KB. Đa số bộ nhớ đệm mức 1 được phân đôi: dùng cho lệnh (Instruction cache-Icache) và dùng cho dữ liệu (Data cache-Dcache). Các bộ vi xử lí công nghệ cao hiện nay (advanced microprocessors) đã thoả mãn các yêu cầu chế tạo các máy tính lớn (mainframes) và các siêu máy tính (supercomputers). Các vi xử lí thời này có buýt địa chỉ đều là 32 bít (phân biệt 4 GB bộ nhớ) và có khả năng làm việc với *bộ nhớ ảo*. Người ta cũng áp dụng các cơ chế hoặc các cấu trúc đã được sử dụng

trong các máy tính lớn vào các bộ vi xử lí: cơ chế *xử lý xen kẽ liên tục dòng mã lệnh(pipeline), bộ nhớ đệm (cache), bộ nhớ ảo.* Các bộ vi xử lý này đều có *bộ quản lý bộ nhớ (*Memory Management Unit-MMU). Chính nhờ các cải tiến đó mà các bộ vi xử lý thế hệ này có khả năng cạnh tranh được với các máy tính nhỏ trong rất nhiều lĩnh vực ứng dụng. Phần lớn các bộ vi xử lý thế hệ này đều được sản xuất bằng công nghệ HCMOS.

Bên cạnh các bộ vi xử lý vạn năng truyền thống thường được dùng để xây dựng các *máy tính với tập lệnh phức tạp (Complex Instruction Set Computer,* CISC) đã nói ở trên, trong thời gian này cũng xuất hiện các bộ vi xử lý cải tiến dùng để xây dựng các *máy tính với tập lệnh rút gọn (Reduced Instruction Set Computer,* RISC) với nhiều tính năng có thể so sánh với các máy tính lớn ở các thế hệ trước. Đó là các bộ vi xử lý Alpha của Digital, PowerPC của tổ hợp hãng Apple- Motorola- IBM. . . Sự ra đời của các vi xử lý loại RISC chính là sự bắt đầu cho một thế hệ khác trong lịch sử phát triển của các thế hệ vi xử lý.

I.4.5 Giai đoạn 2000-2006

Các vi xử lý Intel trong thời gian này thể hiện quan điểm nâng cao hiệu năng của bộ vi xử lý và hệ thống máy tính bằng việc nâng cao xung nhịp. Phiên bản Intel Pentium 4 đã tăng xung nhịp từ 1,5 GHz năm 2000 tới 3GHz vào năm 2002. Vi kiến trúc tiêu biểu cho các vi xử lý này là Netburst với khả năng nâng cao xung nhịp gấp 4 lần xung nhịp của hệ thống. Ngoài ra, Intel giới thiệu công nghệ siêu phân luồng tăng hiệu năng cho hệ thống đa nhiệm và đa luồng. Về lô-gíc, các chương trình phần mềm có thể sử dụng 2 bộ vi xử lý trên 1 bộ vi xử lý vật lý.

Việc nâng cao xung nhịp nhanh chóng đẩy các bộ vi xử lý tới ngưỡng vật lý về điện và nhiệt năng tỏa ra. Thực tế cho thấy đây không phải là phương pháp hiệu quả để tăng hiệu năng của hệ thống. Hãng AMD, một trong những đối thủ cạnh tranh trực tiếp của Intel, nhấn mạnh việc tăng hiệu năng qua việc nâng cao tốc độ thực hiện các lệnh trong một chu kỳ máy. AMD là một trong những hãng đầu tiên tích hợp nhiều bộ giải mã và bộ điều khiển bộ nhớ vào bên trong đơn vị xử lý trung tâm CPU, bộ nhớ đệm mức 1 lớn tới 128KB. Các bộ vi xử lý Athlon 64, Opteron là bộ vi xử lý tiêu biểu của AMD, có tốc độ xung nhịp thấp hơn như hiệu năng thì không hề thua kém Intel. Đặc biệt về tiêu thụ điện và mức tỏa nhiệt thì tốt hơn hẳn Intel nhờ có các công nghệ kiểm soát tiêu thụ điện.

Trong giai đoạn này cũng chứng kiến sự bùng nổ về việc phát triển bộ vi xử lý cho các máy tính xách tay. Yêu cầu rất quan trọng với thiết bị này là hiệu năng xử lý đủ mạnh nhưng mức tiêu thụ điện phải đủ thấp để máy tính có thể hoạt động lâu dài bằng pin. Các bộ vi xử lý di động của Intel Pentium Mobile đã triển khai các giải pháp dung hòa hai yêu cầu trên bằng các nâng cao khả năng xử lý lệnh trên 1 chu kỳ xung nhịp, nâng cao bộ đệm mức 2 lên 1MB, kiểm soát xung nhịp vi xử lý (Speedstep) theo yêu cầu của ứng dụng. Bộ vi xử lý di động đầu tiên hoạt động ở tần số 1,6GHz có thể giảm xuống tới 200MHz khi rỗi có hiệu năng ngang ngửa với Pentium 4 ở tần số trên 2GHz.

Một sự kiện quan trọng trong giai đoạn này là sự ra đời của các bộ vi xử lý 2 nhân cho các máy vi tính. Các hệ thống đa xử lý trước kia chỉ có trong môi trường máy chủ hoặc máy trạm hiệu năng cao. Năm 2005 Intel đưa ra vi xử lý đa nhân đầu tiên Pentium D với hai vi xử

lý riêng biệt trên cùng một vi mạch. Ngay sau đó, AMD cũng đưa ra vi xử lý đa nhân của mình Athlon×2. Thực tế cho thấy thiết kế của AMD mang lại hiệu năng tốt hơn so với Intel.

I.4.6 Giai đoạn 2007-nay

Giai đoạn này tiếp tục chứng kiến sự gia tăng số nhân bên trong bộ vi xử lý giữa các hãng sản xuất vi xử lý như Intel và AMD. Ngoài ra các yêu cầu về tiêu thụ điện và tỏa nhiệt của bộ vi xử lý cũng được quan tâm hơn. Intel cải tiến thiết kế vi kiến trúc nhân (Core micro-architecture) thay thế Netburst và đưa ra thế hệ bộ vi xử lý hai nhân mới Core-2. Bộ vi xử lý này khắc phục các điểm yếu của thế hệ trước đó đặc biệt về tương quan giữa hiệu năng và mức tiêu thụ điện. Năm 2006 chứng kiến sự kiện mới Intel đưa ra các bộ vi xử lý với bốn nhân cho môi trường máy chủ Intel Xeon Quadcore 5355 và máy vi tính Intel Core-2 Extreme QX6700. Việc kết hợp với công nghệ siêu phân luồng trong các bộ vi xử lý Core i7 của Intel cho phép nâng số vi xử lý lô-gíc lên tới 8 cho các các chương trình ứng dụng.

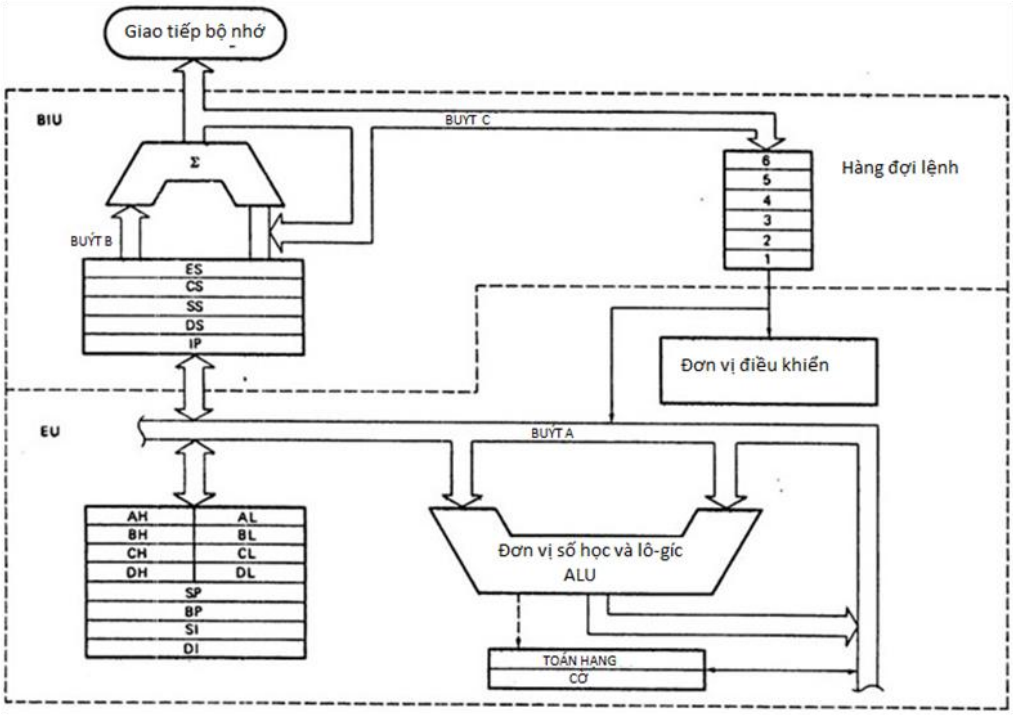
Bên cạnh các bộ vi xử lý cho máy PC và máy chủ, các hãng sản xuất vi xử lý cũng phát triển các dòng vi xử lý nhúng cho các thiết bị tính toán cá nhân. Ưu thế của các vi xử lý nhúng so với vi xử lý kể trên là mức tiêu thụ điện năng, năng lực xử lý và chi phí. Intel cung cấp các vi xử lý nhúng Atom có khả năng xử lý bằng một nửa Pentium M ở cùng xung nhịp với mức tiêu thụ điện khoảng 3W. Ngoài vi xử lý Intel Atom, trên thị trường còn có vi xử lý ARM do hãng Acon phát triển, VIA Nano của hãng VIA. . .

Chương II. Bộ vi xử lý Intel 8086

II.1 Cấu trúc bên trong của 8086/8088

Intel 8086 là bộ vi xử lý 16 bít đầu tiên của Intel và là vi xử lý đầu tiên hỗ trợ tập lệnh x86. Ngoài ra Intel cũng giới thiệu 8088 tương thích với 8086 nhưng độ rộng buýt dữ liệu bằng một nửa (8 bít). Vi xử lý được sử dụng trong nhiều lĩnh vực khác nhau, nhất là trong các máy IBM PC/XT. Các bộ vi xử lý thuộc họ này sẽ còn được sử dụng rộng rãi trong thời gian tới do tính kế thừa của các sản phẩm trong họ x86. Các chương trình viết cho 8086/8088 vẫn có thể chạy trên các hệ thống tiên tiến sau này.

II.1.1 Sơ đồ khối



Hình II-1. Sơ đồ khối 8086

Trong sơ đồ khối, vi xử lý 8086 có hai khối chính BIU và EU. Về chi tiết, vi xử lý này bao gồm các đơn vị điều khiển, số học và lô-gíc, hàng đợi lệnh và tập các thanh ghi. Chi tiết các khối và đơn vị chức năng này được trình bày trong phần sau.

II.1.2 Các đơn vị chức năng: BIU, EU, các thanh ghi và buýt trong

II.1.2.1 Đơn vị giao tiếp buýt và thực thi EU

Theo sơ đồ khối trên Hình II-1 CPU 8086 có 2 khối chính: *khối phối ghép buýt* BIU (Bus Interface Unit) *và khối thực hiện lệnh* EU (Execution Unit). Việc chia CPU ra thành 2 phần làm việc đồng thời có liên hệ với nhau qua đệm lệnh làm tăng đáng kể tốc độ xử lý của CPU. Các buýt bên trong CPU có nhiệm vụ chuyển tải tín hiệu giữa các khối. Trong số các buýt đó có buýt dữ liệu 16 bít của ALU, buýt các tín hiệu điều khiển ở EU và buýt trong của hệ thống ở BIU. Trước khi đi ra buýt ngoài hoặc đi vào buýt trong của bộ vi xử lý, các tín hiệu truyền trên buýt thường được cho đi qua các bộ đệm để nâng cao tính tương thích cho nối ghép hoặc nâng cao phối ghép.

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

EU bao gồm một *đơn vị điều khiển,* khối 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 là ta thu được các dãy xung khác nhau trên kênh điều khiển (tuỳ theo mã lệnh) để điều khiển hoạt động của các bộ phận bên trong và bên ngoài CPU. Ngoài ra, EU còn có *khối số học và lôgic* (Arithmetic and Logic Unit ALU) dùng để thực hiện các thao tác khác nhau với các toán hạng của lệnh. 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 đọc lệnh và giải mã lệnh.

Trong BIU còn có một *bộ nhớ đệm lệnh* với dung lượng 6 byte dùng để chứa các mã lệnh để chờ EU xử lý (bộ đệm lệnh này còn được gọi là *hàng đợi lệnh).*

II.1.2.2 Các thanh ghi

II.1.2.2.a Các thanh ghi đoạn

Thông thường bộ nhớ của chương trình máy tính được chia làm các đoạn phục vụ các chức năng khác nhau như đoạn chứa các câu lệnh, chứa dữ liệu. Trong thực tế bộ vi xử lý 8086 cung cấp các các thanh ghi 16 bít liên quan đến địa chỉ đầu của các đoạn kể trên và chúng được gọi là các thanh ghi đoạn (Segment Registers) cụ thể:

- Thanh ghi đoạn mã CS (Code-Segment),
- Thanh ghi đoạn dữ liệu DS (Data sement).
- Thanh ghi đoạn ngăn xếp SS (Stack segment)
- Thanh ghi đoạn dữ liệu phụ ES (Extra segment).

Các thanh ghi đoạn 16 bít này chỉ ra địa chỉ đầu của bốn đoạn trong bộ nhớ, dung lượng lớn nhất của mỗi đoạn nhớ này là 64 Kbyte và tại một thời điểm nhất định bộ vi xử lý chỉ làm việc được với bốn đoạn nhớ 64 Kbyte này. Để xác định chính xác vị trí một ô nhớ của chương trình các thanh ghi đoạn sẽ phải phối hợp với các thanh ghi đặc biệt khác còn gọi là các thanh ghi lệch hay phân đoạn (offset register). Chi tiết được trình bày ở phần II.1.3.

II.1.2.2.b Các thanh ghi đa năng

Trong khối EU có bốn thanh ghi đa năng 16 bít AX, BX, CX, DX. Điều đặc biệt là khi cần chứa các dữ liệu 8 bít thì mỗi thanh ghi có thể tách ra thành hai thanh ghi 8 bít cao và thấp để làm việc độc lập, đó là các tập thanh ghi AH và AL, BH và BL, CH và CL, DH và DL (trong đó H chỉ phần cao, L chỉ phần thấp). Mỗi thanh ghi có thể dùng một cách vạn năng để chứa các tập dữ liệu khác nhau nhưng cũng có công việc đặc biệt nhất định chỉ thao tác với một vài thanh ghi nào đó. Chính vì vậy các thanh ghi thường được gán cho những cái tên có ý nghĩa. Cụ thể:

- AX (accumulator): thanh chứa. Các kết qủa của các thao tác thường được chứa ở đây (kết quả của phép nhân, chia). Nếu kết quả là 8 bít thì thanh ghi AL được coi là thanh ghi tích luỹ.
- BX (base): thanh ghi cơ sở thường chứa địa chỉ cơ sở của một bảng dùng trong lệnh XLAT.
- CX (count): bộ đếm. CX thường được dùng để chứa số lần lặp trong trường hợp các lệnh LOOP (lặp), còn CL thường cho ta số lần dịch hoặc quay trong các lệnh dịch hoặc quay thanh ghi.
- DX (data): thanh ghi dữ liệu DX cùng BX tham gia các thao tác của phép nhân hoặc chia các số 16 bít. DX thường dùng để chứa địa chỉ của các cổng trong các lệnh vào/ ra dữ liệu trực tiếp.

II.1.2.2.c Các thanh ghi con trỏ và chỉ số

Trong 8088 còn có ba thanh ghi con trỏ và hai thanh ghi chỉ số 16 bít. Các thanh ghi này (trừ IP) đều có thể được dùng như các thanh ghi đa năng, nhưng ứng dụng chính của mỗi thanh ghi là chúng được ngầm định như là thanh ghi lệch cho các đoạn tương ứng. Cụ thể**:**

- IP: con trỏ lệnh (Instruction Pointer). IP luôn trỏ vào lệnh tiếp theo sẽ được thực hiện nằm trong đoạn mã CS. Địa chỉ đầy đủ của lệnh tiếp theo này ứng với CS:IP và được xác định theo cách đã nói ở trên.
- BP: con trỏ cơ sở (Base Pointer). BP luôn trỏ vào một dữ liệu nằm trong đoạn ngăn xếp SS. Địa chỉ đầy đủ của một phần tử trong đoạn ngăn xếp ứng với SS:BP và được xác định theo cách đã nói ở trên.
- SP: con trỏ ngăn xếp (Stack Pointer). SP luôn trỏ vào đỉnh hiện thời của ngăn xếp nằm trong đoạn ngăn xếp SS. Địa chỉ đỉnh ngăn xếp ứng với SS:SP và được xác định theo cách đã nói ở trên.
- SI: chỉ số gốc hay nguồn (Source Index). SI chỉ vào dữ liệu trong đoạn dữ liệu DS mà địa chỉ cụ thể đầy đủ ứng với DS:SI và được xác định theo cách đã nói ở trên.
- DI: chỉ số đích (Destination Index). DI chỉ vào dữ liệu trong đoạn dữ liệu DS mà địa chỉ cụ thể đầy đủ ứng với DS:DI và được xác định theo cách đã nói ở trên.

Riêng trong các lệnh thao tác với dữ liệu kiểu chuỗi thì cặp ES:DI luôn ứng với địa chỉ của phần tử thuộc chuỗi đích còn cặp DS:SI ứng với địa chỉ của phần tử thuộc chuỗi gốc.

II.1.2.2.d Thanh ghi cờ FR (Flag Register)

Đây là thanh ghi khá đặc biệt trong CPU, mỗi bít của nó được dùng để phản ánh một trạng thái nhất định của kết quả phép toán do ALU thực hiện hoặc một trạng thái hoạt động của EU. Dựa vào các cờ này người lập trình có thể có các lệnh thích hợp tiếp theo cho bộ vi xử lý (các lệnh nhảy có điều kiện). Thanh ghi cờ gồm 16 bít nhưng người ta chỉ dùng hết 9 bít của nó để làm các bít cờ như hình vẽ dưới đây.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	U	U	U	0F	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

Hình II-2. Thanh ghi cờ

- U không sử dụng.
- C hoặc CF (Carry Flag): cờ nhớ. CF = 1 khi có nhớ hoặc mượn từ bít có nghĩa lớn nhất MSB (Most Significant Bit).
- P hoặc PF (Parity Flag): cờ parity. PF phản ánh tính chẵn lẻ của tổng số bít 1 có trong kết quả. Cờ PF =1 khi tổng số bít 1 trong kết quả là chẵn (even parity).
- A hoặc AF (Auxiliary Carry Flag): cờ nhớ phụ rất có ý nghĩa khi ta làm việc với các số BCD (Binary Coded Decimal). AF = 1 khi có nhớ hoặc mượn từ một số BCD thấp (4 bít thấp) sang một số BCD cao (4 bít cao).
- Z hoặc ZF (Zero Flag): cờ rỗng. ZF =1 khi kết quả = 0.
- S hoặc SF (sign flag): cờ dấu. SF = 1 khi kết quả âm.
- O hoặc OF (Overflow Flag): cò tràn. OF = 1 khi kết quả là một số bù 2 vượt qua ngoài giới hạn biểu diễn dành cho nó.

Trên đây là 6 bít cờ trạng thái phản ánh các trạng thái khác nhau của kết sau một thao tác nào đó, trong đó 5 bít cờ đầu thuộc byte thấp của thanh cờ là các cờ giống như của bộ vi xử lý 8 bít 8085 của Intel. Chúng được lặp hoặc xoá tuỳ theo các điều kiện cụ thể sau các thao tác của ALU. Ngoài ra, bộ vi xử lý 8086/8088 còn có các cờ điều khiển sau đây (các cờ này được lập hoặc xoá bằng các lệnh riêng):

- T hoặc TF (Trap Flag): cờ bẫy. TF = 1 thì CPU làm việc ở chế độ chạy từng lệnh (chế độ này dùng khi cần tìm lỗi trong một chương trình).
- I hoặc IF (Interrupt Enable Flag): cờ cho phép ngắt. IF = 1 thì CPU cho phép các yêu cầu ngắt (che được) được tác động.
- D hoặc DF (Direction Flag): cờ hướng. DF = 1 khi CPU làm việc với chuỗi ký tự theo thứ tự từ phải sang trái (vì vậy D chính là cờ lùi)

II.1.3 Phân đoạn bộ nhớ của 8086/8088

Khối BIU đưa ra trên buýt địa chỉ 20 bít địa chỉ, như vậy 8086/8088 có khả năng phân biệt ra được 2^{20} = 1.048.576 = 1M ô nhớ hay 1Mbyte, vì các bộ nhớ thường tổ chức theo byte. Nói cách khác: không gian địa chỉ của 8088 là 1Mbyte. Trong không gian 1Mbyte bộ nhớ cần được chia thành các vùng khác nhau (điều này rất có lợi khi làm việc ở chế độ nhiều người sử dụng hoặc đa nhiệm) dành riêng để:

- Chứa mã chương trình.
- Chứa dữ liệu và kết quả không gian của chương trình.
- Tạo ra một vùng nhớ đặc biệt gọi là ngăn xếp (stack) dùng vào việc quản lý các thông số của bộ vi xử lý khi gọi chương trình con hoặc trở về từ chương trình con.

Trong thực tế bộ vi xử lý 8086/8088 có các thanh ghi 16 bít liên quan đến địa chỉ đầu của các vùng (các đoạn) kể trên và chúng được gọi là các thanh ghi đoạn (Segment Registers). Đó là thanh ghi đoạn mã CS (Code-Segment), thanh ghi đoạn dữ liệu DS (Data Sement), thanh ghi đoạn ngăn xếp SS (Stack Segment) và thanh ghi đoạn dữ liệu phụ ES (Extra Segment). Các thanh ghi đoạn 16 bít này chỉ ra địa chỉ đầu của bốn đoạn trong bộ nhớ, dung lượng lớn nhất của mỗi đoạn nhớ này là 64 Kbyte và tại một thời điểm nhất định bộ vi xử lý chỉ làm việc được với bốn đoạn nhớ 64 Kbyte này. Việc thay đổi giá trị của các thanh ghi đoạn làm cho các đoạn có thể dịch chuyển linh hoạt trong phạm vi không gian 1 Mbyte. Vì vậy các đoạn này có thể nằm cách nhau khi thông tin cần lưu đòi hỏi dung lượng đủ 64 Kbyte hoặc cũng có thể nằm trùm nhau do có những đoạn không cần dùng hết đoạn dài 64 Kbyte và vì vậy những đoạn khác có thể bắt đầu nối tiếp ngay sau đó. Điều này cũng cho phép ta truy nhập vào bất kỳ đoạn nhớ (64 Kbyte) nào nằm trong toàn bộ không gian 1 MByte.

Nội dung các thanh ghi đoạn sẽ xác định địa chỉ của ô nhớ nằm ở đầu đoạn. Địa chỉ này còn gọi là địa chỉ cơ sở. 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), do nó ứng với khoảng lệch địa chỉ của một ô nhớ cụ thể nào đó so với ô đầu đoạn. Độ lệch này được xác định bởi các thanh ghi 16 bít khác đóng vai trò thanh ghi lệch (offset register) mà ta sẽ được trình bày sau. Cụ thể, để xác định địa chỉ vật lý 20 bít của một ô nhớ nào đó trong một đoạn bất kỳ. CPU 8086/8088 phải dùng đến 2 thanh ghi 16 bít: một thanh ghi để chứa địa chỉ cơ sở, còn thanh kia chứa độ lệch. Từ nội dung của cặp thanh ghi đó tạo ra địa chỉ vật lý theo công thức sau:

Địachỉvậtlý=Thanh_ghi_đoạn×16+Thanh_ghi_lệch

Việc dùng 2 thanh ghi để ghi nhớ thông tin về địa chỉ thực chất để tạo ra một loại địa chỉ gọi là địa chỉ logic và được ký hiệu như sau:

Thanh_ghi_đoạn: Thanh_ghi_lệch hay **segment: offset**

Địa chỉ kiểu **segment: offset** là logic vì nó tồn tại dưới dạng giá trị của các thanh ghi cụ thể bên trông CPU và ghi cần thiết truy cập ô nhớ nào đó thì nó phải được đổi ra địa chỉ vật

lý để rồi được đưa lên buýt địa chỉ. Việc chuyển đổi này do một bộ tạo địa chỉ thực hiện (phần tử $\sum$ trên Hình II-1).

Ví dụ: cặp CS:IP sẽ chỉ ra địa chỉ của lệnh sắp thực hiện trong đoạn mã. Tại một thời điểm nào đó ta có CS = F00H và IP = FFF0H thì

CS:IP~F000Hx16 + FFF0H = F000H + FFF0H = FFFF0H

Do tổ chức như vậy nên dẫn đến tính đa trị của các thanh ghi đoạn và thanh ghi lệch trong địa chỉ logic ứng với một địa chỉ vật lý. Từ một địa chỉ vật lý ta có thể tạo ra các giá trị khác nhau của thanh ghi đoạn và thanh ghi lệch

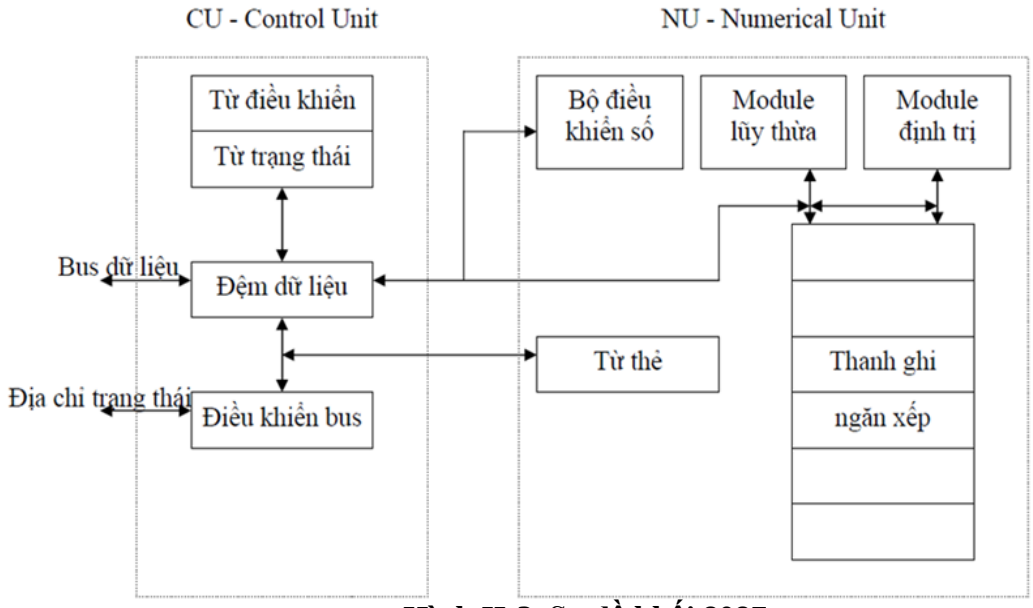
Ví dụ: Địa chỉ vật lý 12345H có thể được tạo ra từ các giá trị:

Thanh ghi đoạn	Thanh ghi lệch
1000H	2345H
1200H	0345H
1004H	2305H

II.2 Bộ đồng xử lý toán học 8087

Như được trình bày trong phần trước, 8086/8088 không có các thao tác với số thực. Để làm việc này, hệ vi xử lý cần có các bộ đồng xử lý toán học 80x87 hỗ trợ CPU trong việc tính toán các biểu thức dùng dấu chấm động như cộng, trừ, nhân, chia các số dấu chấm động, căn thức, logarit, … Chúng cho phép xử lý các phép toán này nhanh hơn nhiều so với 8086/8088.

8087 gồm một đơn vị điều khiển (CU – Control Unit) dùng để điều khiển buýt và một đơn vị số học (NU – Numerical Unit) để thực hiện các phép toán dấu chấm động trong các mạch tính lũy thừa (exponent module) và mạch tính phần định trị (mantissa module). Khác với 8086, thay vì dùng các thanh ghi rời rạc là một ngăn xếp thanh ghi.



Hình II-3. Sơ đồ khối 8087

Đơn vị điều khiển nhận và giải mã lệnh, đọc và ghi các toán hạng, chạy các lệnh điều khiển riêng của 8087. Do đó, CU có thể đồng bộ với CPU trong khi NU đang thực hiện các công việc tính toán. CU bao gồm bộ điều khiển buýt, bộ đệm dữ liệu và hàng lệnh.

Ngăn xếp thanh ghi có tất cả 8 thanh ghi từ R0 ÷ R7, mỗi thanh ghi dài 80 bít trong đó bít 79 là bít dấu, bít 64 ÷ 78 dùng cho số mũ và phần còn lại là phần định trị. Dữ liệu truyền giữa các thanh ghi này được thực hiện rất nhanh do 8087 có độ rộng buýt dữ liệu là 84 bít và không cần phải biến đổi định dạng. Ngay sau khi khởi động lại PC, bộ đồng xử lý kiểm tra xem nó có được nối với PC hay không và sẽ điều chỉnh độ dài của hàng lệnh cho phù hợp với CPU (nếu dùng 8086 thì độ dài là 6 byte).

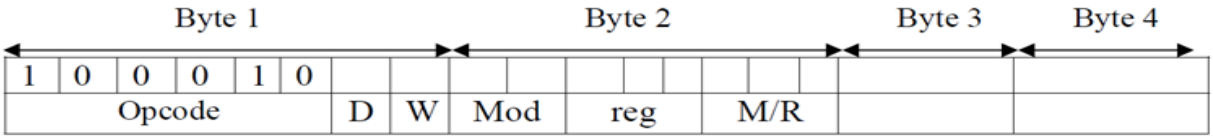
II.3 Tập lệnh của 8086/8088

II.3.1 Khái niệm lệnh, mã hoá lệnh và quá trình thực hiện lệnh

Lệnh của bộ vi xử lý được ghi bằng các ký tự dưới dạng gợi nhớ (memonic) để người sử dụng để 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 hiểu được. Vì lệnh do bộ vi xử lý được cho dưới dạng mã nên sau khi nhận lệnh, bộ vi xử lý phải thực hiện việc giải mã lệnh rồi sau đó mới thực hiện lênh.

Một lệnh có thể có độ dài một vài byte tuỳ theo bộ vi xử lý. Số lượng các bít *n* dùng để mã hóa vi lệnh (opcode) cho biết số lượng tối đa các lệnh (2^n) có trong bộ vi xử lý. Với 1 byte bộ vi xử lý có thể mã hoá được tối đa 256 lệnh. Trong thực tế việc ghi lệnh không phải hoàn toàn đơn giản như vậy. Việc mã hoá lệnh cho bộ vi xử lý là rất phức tạp và bị chi phối bởi nhiều yếu tố khác nữa. Đối với bộ vi xử lý 8086/8088 một lệnh có thể có độ dài từ 1 đến 6 byte. Ta sẽ chỉ lấy trường hợp lệnh MOV để giải thích cách ghi lệnh nói chung của 8086/8088.

Lệnh MOV *đích, gốc* dùng để chuyển dữ liệu giữa thanh ghi và ô nhớ. Chỉ nguyên với các thanh ghi của 8086/8088, nếu ta lần lượt đặt các thanh ghi vào các vị trí toán hạng đích và toán hạng gốc ta thấy đã phải cần tới rất nhiều mã lệnh khác nhau để mã hoá tổ hợp các này.



Hình vẽ trên biểu diễn dạng thức các byte dùng để mã hoá lệnh MOV. Từ đây ta thấy rằng để mã hoá lệnh MOV ta phải cần ít nhất là 2 byte, trong đó 6 bít của byte đầu dùng để chứa mã lệnh. Đối với các lệnh MOV. Bít W dùng để chỉ ra rằng 1 byte (W = 0) hoặc 1 từ (W = 1) sẽ được chuyển. Trong các thao tác chuyển dữ liệu, một toán hạng luôn bắt buộc phải là thanh ghi. Bộ vi xử lý dùng 2 hoặc 3 bít để mã hoá các thanh ghi trong CPU như sau:

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

Bít D dùng để chỉ hướng đi của dữ liệu. D = 1 thì dữ liệu đi đến thanh ghi cho bởi bít của REG. 2 bít MOD (chế độ) cùng với 3 bít M/R (bộ nhớ/thanh ghi) tạo ra 5 bít dùng để chỉ ra chế độ địa chỉ cho các toán hạng của lệnh.

Bảng dưới đây cho ta thấy cách mã hoá các chế độ địa chỉ (cách tìm ra các toán hạng bằng các bít này).

MOD / R/M	00	01	10	11	
				W = 1	W = 0
000	[BX]+[SI]	[BX]+[SI]+addr8	[BX]+[SI]+addr16	AX	AL
001	[BX]+[DI]	[BX]+[DI]+addr8	[BX]+[DI]+addr16	CX	CL
010	[BP]+[SI]	[BP]+[SI] +addr8	[BP]+[SI] +addr16	DX	DL
011	[BP]+[DI]	[BP]+[DI] +addr8	[BP]+[DI] +addr16	BX	BL
100	[SI]	[SI] +addr8	[SI] +addr16	SP	AH
101	[DI]	[DI] +addr8	[DI] +addr16	BP	CH
110	addr16	[BP] +addr8	[BP] +addr16	SI	DH
111	[BX]	[BX] +addr8	[BX] +addr16	DI	BH

Ghi chú:

- addr8, addr16 tương ứng với địa chỉ 8 và 16 bít
- Các giá trị cho trong các cột 2, 3, 4 (ứng với MOD =00, 01, 10) là các địa chỉ hiệu dụng (EA) sẽ được cộng với DS để tạo ra địa chỉ vật lý (riêng BP phải được cộng với SP)

II.3.2 Các chế độ địa chỉ của 8086/8088

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 ra bộ bi xử lý và sau này không thể thay đổi được. Bộ vi xử lý 8088 và cả họ 80x86 nói chung đều có 7 chế độ địa chỉ sau:

1. Chế độ địa chỉ thanh gi (register addressing mode).
2. Chế độ địa chỉ tức thì (immediate addressing mode).
3. Chế độ địa chỉ trực tiếp (direct addressing mode).
4. Chế độ địa chỉ gián tiếp qua thanh ghi (register indirect addressing mode).
5. Chế độ địa chỉ tương đối cơ sở (based indexed relative addressing mode).

6. Chế độ địa chỉ tương đối chỉ số (indexed relative addressing mode).
7. Chế độ địa chỉ tương đối chỉ số cơ sở (based indexed relative addressing mode).

II.3.2.1 Chế độ địa chỉ thanh ghi

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

Ví dụ II-1

```
MOV BX, DX  ; chuyển nội dung DX vào BX.
MOV DS, AX  ; chuyển nội dung AX vào DX
ADD AL, DL  ; cộng nội dung AL và DL rồi đưa vào
```

II.3.2.2 Chế độ địa chỉ tức thì

Trong chế độ địa chỉ này, toán hạng đích là một thanh ghi hay một ô nhớ, còn toán hạng nguồn là một hằng số và vị trí của toán hạng này ở ngay sau mã lệnh. Chế độ địa chỉ này có thể được dùng để nạp dữ liệu cần thao tác vào bất kỳ thanh ghi nào (ngoại trừ các thanh ghi đoạn và thanh cờ) hoặc vào bất kỳ ô nhớ nào trong đoạn dữ liệu DS.

Ví dụ II-2

```
MOV CL, 100       ; chuyển 100 vào CL.
MOV AX, 0FF0H     ; chuyển 0FF0H vào AX để rồi đưa
MOV DS, AX        ; vào DS (vì không thể chuyểntrực tiếp vào thanh ghi đoạn)
MOV [BX], 10      ; chỉ DS:BX.
```

II.3.2.3 Chế độ địa chỉ trực tiếp

Trong chế độ địa chỉ này một toán hạng chứa địa chỉ lệnh của ô nhớ dùng chứa dữ liệu còn toán hạng kia chỉ có thể là thanh ghi mà không được là ô nhớ. Nếu so sánh với chế độ địa chỉ tức thì ta thấy ở đây ngay sau mã lệnh không phải là toán hạng mà là địa chỉ lệch của toán hạng. Xét về phương diện địa chỉ thì đó là địa chỉ trực tiếp.

Ví dụ II-3

```
MOV AL, [1234H]   ;chuyển ô nhớ DS:1234 vào AL.
MOV [4320H], CX   ;chuyển CX vào 2 ô nhớ liên tiếp DS:4320 và DS:4321
```

II.3.2.4 Chế độ 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 chứa dữ liệu, còn toán hạng kia chỉ có thể là thanh ghi mà không được là ô nhớ (8086/8088 không cho phép quy chiếu bộ nhớ 2 lần đối với một lệnh).

Ví dụ II-4

```
MOV AL, [BX]        ; chuyển ô nhớ có địa chỉ DS:BX vào AL.
MOV [SI], CL        ; chuyển CL vào ô nhớ có địa chỉ DS:SI.
MOV [DI], AX        ; chuyển AX vào 2 ô nhớ liên tiếp tại DS:DI và DS: (DI + 1).
```

II.3.2.5 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 (*displacement values)* được dùng để tính địa chỉ hiệu dụng của toán hạng trong các vùng nhớ DS và SS. Sự có mặt của các giá trị dịch chuyển xác định tính tương đối của địa chỉ so với địa chỉ cơ sở.

Ví dụ II-5

```
MOV CX, [BX] +10     ; chuyển 2 ô nhớ liên tiếp có địa chỉ DS: [BX + 10] và
                     ; DS: [BX + 10] vào CX.
MOV CX, [BX+10]      ; một cách viết khác của lệnh trên.
MOV CX, 10 [BX]      ; một cách viết khác của lệnh đầu.
MOV AL, [BP] +5      ; chuyển ô nhớ SS: [BP+5] vào AL.
ADD AL, Table [BX]   ; cộng AL với ô nhớ do BX chỉ ra trong bảng table
                     ; (bảng này nằm trong DS), kết quả dựa vào AL.
```

Trong ví dụ trên:

- 10 và 5 là các giá trị cụ thể cho biết mức dịch chuyển của các toán hạng. Table là tên mảng biểu diễn kiểu dịch chuyển của mảng (phần tử đầu tiên) so với địa chỉ đầu của đoạn dữ liệu DS.
- [BX + 10] hoặc [BP+5] gọi là địa chỉ hiệu dụng *(*Effective Address *EA.* theo cách gọi của Intel).
- DS: [BX + 10] hoặc SS: [BP+5] chính là logic tương ứng với một địa chỉ vật lý.
- Theo cách định nghĩa này thì địa chỉ hiệu dụng của một phần tử thứ BX nào đó (kể từ 0) trong mảng Table [BX] thuộc đoạn DS là EA = Table+BX và của phần tử đầu tiên là EA = Table.

II.3.2.6 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 địa chỉ này ta dùng cả 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 phức tạp nhất: chế độ địa chỉ tương đối chỉ số cơ sở. Ta có thể thấy chế độ địa chỉ này rất phù hợp cho việc địa chỉ hoá các mảng hai chiều.

Ví dụ II-6

```
MOV AX, [ BX ] [SI ]+8      ;chuyển 2 ô nhớ liên tiếp có địa chỉ
                            ; DS:[BX+SI+8] và DS:[BX+SI+9] vào AX
MOV AX, [BX+SI+8]           ; một cách viết khác của lệnh trên
```

MOV CL, [BP+DI+5] ; chuyển ô nhớ SS:[BP+DI+5] vào CL.

II.3.2.7 Tổng kết các chế độ địa chỉ

Các chế địa chỉ đã trình bày ở trên có thể tóm tắt lại trong Bảng II-1.

Bảng II-1. Tóm tắt các chế độ địa chỉ

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] [SI] [DI]	DS DS DS
Tương đối cơ sở	[BX]+disp [BP] +DISP	DS SS
Tương đối chỉ số	[DI]+Disp [SI]+ DISP	DS DS
Tương đối chỉ số cơ sở	[BX]+[DI]+DISP [BX]+[SI]+DISP [BP]+[DI]+DISP [BP]+[SI]+DISP	DS DS SS SS

II.3.2.8 Phương pháp bỏ ngầm định thanh ghi đoạn (segment override)

Như trong các phần trước đã nói, các thanh ghi đoạn và thanh ghi lệch được ngầm định đi kèm với nhau từng cặp dùng để địa chỉ hoá các toán hạng trong các vùng khác nhau của bộ nhớ. Bảng II-2 chỉ ra các cặp đôi ngầm định của cácthan ghi đoạn và thanh ghi lệch thường dung. Vì tính ngầm định này nên trong các lệnh ta chỉ cần viết các thanh ghi lệch là đủ cơ sở để tính ra được đia chỉ của toán hạng.

Tuy nhiên, ngoài các tổ hợp ngầm định đã kể, 8086/8088 còn cho phép ta làm việc với các tổ hợp ngầm định đã kể, 8086/8088 còn cho phép ta làm việc với các tổ hợp khác của các thanh ghi đoạn và thanh ghi lệch. Muốn loại vỏ các tổ hợp ngầm định nói trên, trong khi viết lệnh ta phải ghi rõ thanh ghi đoạn sẽ dùng để tính địa chỉ và kèm thêm dấu 2 chấm trước thanh ghi lệch.

Bảng II-2. Các cặp thanh ghi đoạn và thanh ghi lệch ngầm định

Thanh ghi đoạn	CS	DS	ES	SS
Thanh ghi lệch	IP	SI, DI, BX	DI	SP, BP

Ví dụ:

Nếu ta muốn thay đổi, không lấy toán hạng trong đoạn dữ liệu DS, mà lại lấy toán hạng trong đoạn dữ liệu phụ ES để đưa vào AL, thì ta phải viết lại lệnh trên thành

MOV AL, ES:[BX]

Trong đó ta đã dùng ES: để loại bỏ thanh ghi đoạn ngầm định DS và để chỉ rõ thanh ghi đoạn mới dùng trong lệnh này bây giờ là ES.

II.3.3 Tập lệnh của 8086/8088

Bộ xử lý 8086 có tập lệnh gồm 111 lệnh, chiều dài của lệnh từ 1 byte đến vài byte. Tập lệnh 8086 hỗ trợ các nhóm thao tác căn bản như dưới đây.

II.3.3.1 Các lệnh trao đổi dữ liệu.

Các câu lệnh trong nhóm cho phép trao đổi dữ liệu giữa thanh ghi và ô nhớ hay giữa thiết bị vào/ra với ô nhớ hoặc thanh ghi. Kích cỡ dữ liệu cho phép với các câu lệnh này là byte (8 bít) hoặc word (16 bít). Như vậy các câu lệnh trao đổi dữ liệu giúp nạp dữ liệu cần thiết cho các thao tác tính toán của vi xử lý. Ngoài ra các lệnh này cho phép lưu các kết quả tính toán ra bộ nhớ hoặc các thiết bị ngoại vi.

Bảng II-3. Các lệnh trao đổi dữ liệu

Mã gợi nhớ	Chức năng
MOV	Di chuyển byte hay word giữa thanh ghi và ô nhớ
IN, OUT	Đọc, ghi một byte hay word giữa cổng và ô nhớ
LEA	Nạp địa chỉ hiệu dụng
PUSH, POP	Nạp vào, lấy ra một word trong ngăn xếp.
XCHG	Hoán đổi byte hay word

II.3.3.1.a MOV – Chuyển 1 byte hay word

Viết lệnh: MOV Đích, Gốc.

Mô tả: Đích ← Gốc

Trong đó toán hạng đích và gốc 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 2 ô nhớ hoặc 2 thanh ghi đoạn.

Lệnh này không tác động đến các cờ.

Ví dụ:

```
MOV AL, 74H          ; AL ← 74
MOV CL, BL           ; CL ← BL
MOV DL, [SI]         ; DL ← [DS:SI]
MOV AL, Table [BX]   ; AL ←[DS:(Table+BX)]
```

II.3.3.1.b LEA - Nạp địa chỉ hiệu dụng vào thanh ghi

Viết lệnh: *LEA Đích, Gốc*

Trong đó:

+ Đích thường là một trong các thanh ghi: BX, CX, DX, BP, SI, DI.

+ Gốc là tên biến trong đoạn DS được chỉ rõ trong lệnh hoặc ô nhớ cụ thể.

Mô tả: Đích ← Địa chỉ lệch của Gốc, hoặc

Đích ← Địa chỉ hiệu dụng của Gốc

Đây là lệnh để tính địa chỉ lệch của biến hoặc địa chỉ của ô nhớ chọn làm gốc rồi nạp vào thanh ghi đã chọn.

Lệnh này không tác động đến các cờ.

Ví dụ:

```
LEA DX, MSG          ; nạp địa chỉ lệch của bản tin MSG vào DX.
LEA CX, [BX] [DI]    ; nạp vào CX địa chỉ hiệu dụng
                     ; do BX và DI chỉ ra: EA =BX+DI
```

II.3.3.1.c IN- Đọc dữ liệu từ cổng vào thanh ghi ACC.

Viết lệnh: IN ACC, Port

Mô tả: ACC <- [Port]

Trong đó [Port] là dữ liệu của cổng có địa chỉ là Port. Port là địa chỉ 8 bít của cổng, nó có thể có các giá trị trong khoảng 00H…FFH. Như vậy ta có thể có các khả năng sau:

+Nếu ACC là AL thì dữ liệu 8 bít được đưa vào từ cổng Port.

+Nếu ACC là AX thì dữ liệu 16 bít được đưa vào từ cổng Port và cổng Port+1.

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

IN ACC, DX

Trong đó DX phải được gắn từ trước giá trị ứng với địa chỉ cổng. Lệnh này không tác động đến các cờ.

II.3.3.1.d OUT - Ghi dữ liệu từ Acc ra cổng)

Viết lệnh: *OUT Port, Acc*

Mô tả: *Acc → [port]*

Trong đó [port]là dữ liệu của cổng có địa chỉ là Port. Port là địa chỉ 8 bít của cổng, nó có thể có các giá trị trong khoảng 00H. . . FFH. Như vậy ta có thể có các khả năng sau:

+ Nếu Acc là AL thì dữ liệu 8 bít được đưa ra cổng port.

+ Nếu Acc là AX thì dữ liệu 16 bít được đưa ra cổng port và cổng port +1.

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

OUT DX, Acc

Trong đó DX phải được gán từ trước giá trị ứng với địa chỉ cổng. Lệnh này không tác động đến các cờ.

II.3.3.2 Các lệnh tính toán số học và lô gíc.

Đây là các nhóm lệnh thực hiện các tính toán chủ yếu của vi xử lý 8086/8088.

Bảng II-4. Các lệnh số học và lô gíc

Mã gợi nhớ	Chức năng
NOT	Đảo (bù một) byte hay word
AND	Phép và byte hoặc word
OR	Phép hoặc byte hoặc word
XOR	Phép hoặc loại trừ byte hoặc word
SHL, SHR	Dịch trái, dịch phải lôgíc byte hay word. Số bước 1 hoặc do CL xác định
SAL, SAR	Dịch trái, dịch phải số học byte hay word. Số bước 1 hoặc do CL xác định
ROL, ROR	Quay trái, quay phải byte hay word. Số bước 1 hoặc do CL xác định
ADD, SUB	Cộng trừ byte hoặc word
ADC, SBB	Cộng trừ byte hoặc word có nhớ
INC, DEC	Tăng, giảm
NEG	Đảo byte hoặc word (bù 2)
CMP	So sánh hai byte hoặc word
MUL, DIV	Nhân, chia byte hoặc word không dấu
IMUL, IDIV	Nhân chia byte hoặc word có dấu

II.3.3.2.a ADD-Cộng 2 toán hạng

Viết lệnh: ADD Đích, Gốc.

Mô tả: Đích ← Đích + Gốc.

Trong đó toán hạng đích và gốc có thể tìm được theo các chế độ địa chỉ khác nhau. Nhưng phải chứa dữ liệu có cùng độ dài và không được phép đồng thời là 2 ô nhớ và cũng không được là thanh ghi đoạn. Có thể tham khảo các ví dụ của lệnh ADC.

Cập nhật: AF, CF, PF, SF, ZP

II.3.3.2.b MUL - Nhân số không dấu

Viết lệnh: MUL Gốc

Trong đó toán hạng Gốc là số nhân và có thể tìm được theo các chế độ địa chỉ khác nhau.

Mô tả: tuỳ theo độ dài của toán hạng Gốc ta có 2 trường hợp tổ chức phép nhân, chỗ để ngầm định cho số bị nhân và kết quả:

Nếu Gốc là số 8 bít: AL × Gốc,

số bị nhân phải là số 8 bít đặt trong AL.

sau khi nhân: AX ← tích,

Nếu Gốc là số 16 bít: AX × Gốc,

số bị nhân phải là số 16 bít đặt trong AX.

sau khi nhân: DXAX ← tích.

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

Như vậy các cờ CF và OF sẽ báo cho ta biết có thể bỏ đi bao nhiêu số 0 trong kết quả. Ví dụ: Nếu ta cần nhân một số 8 bít với một số 16 bít, ta để số 16 bít tại Gốc và số 8 bít ở AL. Số 8 bít này ở AL cần phải được mở rộng sang AH bằng cách gán AH=0 để làm cho số bị nhân nằm trong AX. Sau cùng chỉ việc dùng lệnh MUL Gốc và kết quả có trong cặp DXAX.

Cập nhật: CF, OF.

Không xác định: AF, PF, SF, ZP.

*II.3.3.2.c DIV – **C**hia 2 số không có dấu*

*Viết lệnh: **DIV Gốc***

Trong đó toán hạng Gốc là số chia và có thể tìm được theo các chế độ địa chỉ khác nhau.

Mô tả: tuỳ theo độ dài của toán hạng gốc ta có 2 trường hợp bố trí phép chia. Các chỗ để ngầm định cho số bị chia và kết quả:

- Nếu Gốc là số 8 bít: AX/Gốc. Số bị chia phải là số không dấu 16 bít đặt trong AX.
- Nếu Gốc là số 16 bít: DXAX/Gốc. Số bị chia phải là số không dấu 32 bít đặt trong cặp thanh ghi DXAX.
- Nếu thương không phải là số nguyên nó được làm tròn theo số nguyên sát đuôi.
- Nếu Gốc = 0 hoặc thương thu được lớn hơn FFH hoặc FFFFH (tuỳ theo độ dài của toán hạng Gốc) thì 8088 thực hiện lệnh ngắt INT 0.

Không xác định: AF, CF, OF, PF, SF, ZP.

II.3.3.2.d CMP- So sánh 2 byte hay 2 word

Viết lệnh: **CMP Đích, Gốc.**

Mô tả: Đích – Gốc.

Trong đó toán hạng đích và gốc có thể tìm được theo các chế độ địa chỉ khác nhau. Nhưng phải chứa dữ liệu có cùng độ dài và không được phép đồng thời là 2 ô nhớ.

Lệnh này chỉ tạo các cờ, không lưu kết quả so sánh, sau khi so sánh các toán hạng không bị thay đổi. Lệnh này thường được dùng để tạo cờ cho các lệnh nhảy có điều kiện (nhảy theo cờ).

Các cờ chính theo quan hệ đích và gốc khi so sánh 2 số không dấu:

CF ZF

Đích = Gốc 0 1

Đích > Gốc 0 1

Đích > Gốc 1 0

Cập nhật: AF, CF, OF, PF, SF, ZP.

II.3.3.2.e AND - Phép và 2 toán hạng

Viết lệnh: AND Đích, Gốc

Mô tả: Đích - Đích, Gốc.

Trong đó toán hạng đích và gốc có thể tìm được theo các chế độ địa chỉ khác nhau. Nhưng phải chứa dữ liệu cùng độ dài và không được phép đồng thời là 2 ô nhớ và cũng không được là thanh ghi đoạn. Phép AND thường dùng để che đi/giữ lại một vài bít nào đó của một toán hạng bằng cách nhân logic toán hạng đó với toán hạng tức thì có các bít 0/1 ở các chỗ cần che đi/giữ nguyên tương ứng (toán hạng tức thì lúc này còn được gọi là mặt nạ).

Xoá: CF, OF.

Cập nhật: PF, SF, ZP, PF chỉ có nghĩa khi toán hạng là 8 bít.

Không xác định: AF.

Ví dụ:

```
AND AL, BL      ;AL, AL BL theo từng bít.
AND BL, 0FH     ;che 4 bít cao của BL.
```

II.3.3.3 Điều khiển, rẽ nhánh và lặp.

Các câu lệnh thuộc nhóm này cho phép thay đổi trật tự thực hiện các câu lệnh bên trong chương trình. Một số câu lệnh tiêu biểu được liệt kê trong bảng dưới đây.

Bảng II-5. Các lệnh rẽ nhánh và lặp tiêu biểu

Mã gợi nhớ	Chức năng
JMP	Nhảy không điều kiện
JA (JNBE)	Nhảy nếu lớn hơn
JAE (JNB)	Nhảy nếu lớn hơn hoặc bằng
JB (JNAE)	Nhảy nếu bé hơn
JBE (JNA)	Nhảy nếu bé hơn hoặc bằng
JE (JZ)	Nhảy nêu bằng
JC, JNC	Nhảy nếu cờ nhớ đặt, xóa
JO, JNO	Nhảy nếu cờ tràn đặt, xóa
JS, JNS	Nhảy nếu cờ dấu đặt, xóa
LOOP	Lặp không điều kiện, số lần lặp do CX xác định
LOOPE (LOOPZ)	Lặp nếu bằng (cờ không) hoặc số lần lặp do CX xác định
LOOPNE (LOOPNZ)	Lặp nếu không bằng (cờ không xóa) hoặc số lần lặp do CX xác định
CALL, RET	Gọi hàm, trở về từ hàm con
INT	Ngắt mềm
IRET	Quay trở về từ đoạn chương trình ngắt

II.3.3.3.a JMP - Nhảy (vô điều kiện) đến một đích nào đó

Lệnh này khiến cho bộ vi xử lý 8086/8088 bắt đầu thực hiện một lệnh mới tại địa chỉ được mô tả trong lệnh. Lệnh này phân biệt nhảy xa và nhảy gần theo vị trí của câu lệnh mới. Tuỳ thuộc vào độ dài của bước nhảy chúng ta phân biệt các kiểu lệnh nhảy gần và nhảy xa với độ dài lệnh khác nhau. Lệnh nhảy đến nhãn ngắn *shortlabel* là lệnh nhảy tương đối. Nơi đến phải nằm trong phạm vi từ -128 đến +127 so với vị trí của lệnh nhảy. Toán hạng nguồn trong lệnh chỉ là byte độ dời để cộng thêm vào thanh ghi IP. Byte độ dời này được mở rộng dấu trước khi cộng vào thanh ghi IP.

- *Ví dụ :*

JMP SHORT 18h

JMP 0F008h

JMP DWORD PTR [3000h]

Lệnh này không tác động đến các cờ.

II.3.3.3.b LOOP -Lặp lại đoạn chương trình do nhãn chỉ ra cho đến khi CX=0

Viết lệnh: LOOP NHÃN

Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ nhãn 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 thanh ghi CX và sau mỗi lần thực hiện lệnh LOOP NHAN thì đồng thời CX tự động giảm đi một (CX ←CX-1).

Lệnh này không tác động đến các cờ.

II.3.3.4 Điều khiển vi xử lý.

Các câu lệnh này tác động lên thanh ghi cờ là thay đổi trạng thái hoạt động của vi xử lý.

Bảng II-6. Các lệnh điều khiển vi xử lý tiêu biểu

Mã gợi nhớ	*Chức năng*
STC, CLC, CMC	Lập, xóa cờ nhớ
STD, CLD	Lập xóa cờ hướng
STI, CLI	Lập xóa cờ cho phép ngắt
PUSHF, POPF	Nạp vào, lấy ra thanh ghi cờ tới/từ ngăn xếp
NOP	Không làm gì cả
WAIT	Chờ tín hiệu TEST
HLT	Treo vi xử lý

II.4 Ngắt và xử lý ngắt trong 8086/8088

II.4.1 Sự cần thiết phải ngắt CPU

Ngắt là việc tạm dừng việc chương trình đang chạy để CPU có thể chạy một chương trình khác nhằm xử lý một yêu cầu do bên ngoài đưa tới CPU như yêu cầu vào/ra hoặc do chính yêu cầu của bên trong CPU như lỗi trong khi tính toán. Trong cách tổ chức trao đổi dữ liệu thông qua việc thăm dò trạng thái sẵn sàng của thiết bị ngoại vi, trước khi tiến hành bất kỳ một cuộc trao đổi dữ liệu nào CPU phải dành toàn bộ thời gian vào việc xác định trạng thái sẵn sàng làm việc của thiết bị ngoại vi. Để tận dụng khả năng của CPU để làm thêm được nhiều công việc khác nữa, chỉ khi nào có yêu cầu trao đổi dữ liệu thì mới yêu cầu CPU tạm dừng công việc hiện tại để phục vụ việc trao đổi dữ liệu. Sau khi hoàn thành việc trao đổi dữliệu thì CPU lại phải quay về để làm tiếp công việc hiện đang bị gián đoạn.

Khi nghiên cứu các tín hiệu của CPU8086/ 8088, vi mạch này có các chân tín hiệu cho các yêu cầu ngắt che được INTR và không che được NMI, chính các chân này sẽ được sử dụng vào việc đưa các yêu cầu ngắt từ bên ngoài đến CPU.

II.4.2 Các loại ngắt trong hệ 8088

Trong hệ vi xử lý 8088 có thể xếp các nguyên nhân gây ra ngắt CPU vào 3 nhóm như sau:

- Nhóm các ngắt cứng: đó là các yêu cầu ngắt CPU do các tín hiệu đến từ các chân INTR và NMI. Ngắt cứng INTR là yêu cầu ngắt che được. Các lệnh CLI và STI có

ảnh hưởng trực tiếp tới trạng thái của cờ IF trong bộ vi xử lý, tức là ảnh hưởng tới việc CPU có nhận biết yêu cầu ngắt tại chân này hay không. Yêu cầu ngắt tại chân INTR có thể có kiểu ngắt N nằm trong khoảng 0-FFH. Kiểu ngắt này phải được đưa vào buýt dữ liệu để CPU có thể đọc được khi có xung trong chu kỳ trả lời chấp nhận ngắt.

- Nhóm các ngắt mềm: khi CPU thực hiện các lệnh ngắt dạng INT N, trong đó N là số hiệu (kiểu) ngắt nằm trong khoảng 00-FFH (0-255).
- Nhóm các hiện tượng ngoại lệ: đó là các ngắt do các lỗi nảy sinh trong quá trình hoạt động của CPU như phép chia cho 0, xảy ra tràn khi tính toán.

Yêu cầu ngắt sẽ được CPU kiểm tra thường xuyên tại chu kỳ đồng hồ cuối cùng của mỗi lệnh. Bảng II-1 trình bày một cách đơn giản để đưa được số hiệu ngắt N vào buýt dữ liệu trong khi cũng tạo ra yêu cầu ngắt đưa vào chân INTR của bộ vi xử lý 8086/8088.

Giả thiết trong một thời điểm nhất định chỉ có một yêu cầu ngắt IRi được tác động và sẽ có xung yêu cầu ngắt đến CPU. Tín hiệu IRi được đồng thời đưa qua mạch khuếch đại đệm để tạo ra số hiệu ngắt tương ứng, số hiệu ngắt này sẽ được CPU đọc vào khi nó đưa ra tín hiệu trả lời.

Bảng II-7 Quan hệ giữa IRi và số hiệu ngắt N tương ứng.

IR6	IR5	IR4	IR3	IR2	IR1	IR0	N
1	1	1	1	1	1	0	FEH (254)
1	1	1	1	1	0	1	FDH (253)
1	1	1	1	0	1	1	FBH (251)
1	1	1	0	1	1	1	F7H (247)
1	1	0	1	1	1	1	EFH (239)
1	0	1	1	1	1	1	DFH (223)
0	1	1	1	1	1	1	BFH (191)

II.4.3 Đáp ứng của CPU khi có yêu cầu ngắt

Khi có yêu cầu ngắt kiểu N đến CPU và nếu yêu cầu đó được phép, CPU thực hiện các công việc sau:

1. SP ← SP-2, [SP] ← FR, trong đó [SP] là ô nhớ do SP chỉ ra.
   (chỉ ra đỉnh mới của ngắn xếp, cất thanh ghi cờ vào đỉnh ngăn xếp)
2. IF ← 0, TF ← 0.
   (cấm các ngắt khác tác động vào CPU, cho CPU chạy ở chế độ bình thường)
3. SP ← SP-2, [SP] ← CS.
   (chỉ ra đỉnh mới của ngăn xếp, cất phần địa chỉ đoạn của địa chỉ trở về vào đỉnh ngăn xếp)
4. SP ← SP-2, [SP] ← IP

(chỉ ra đỉnh mới của ngăn xếp, cất phần địa chỉ lệch của địa chỉ trờ về vào đỉnh ngăn xếp)

5. [N*4] → IP, [N*4+2] → CS
   (lấy lệnh tại địa chỉ mới của chương trình con phục vụ ngắt kiểu N tương ứng trong bảng vectơ ngắt)
6. Tại cuối chương trình phục vụ ngắt, khi gặp lệnh IRET
   [SP] → IP, SP ← SP+2

   [SP] → CS, SP ← SP+2

   [SP] → FR, SP ← SP+2

   (bộ vi xử lý quay lại chương trình chính tại địa chỉ trở về và với giá trị cũ của thanh ghi cờ được lấy ra từ ngăn xếp).

Về mặt cấu trúc chương trình, khi có ngắt xảy ra thì chương trình chính tạm dừng việc thực hiện và lưu các thanh ghi cần thiết như thanh ghi cờ. Sau đó con trỏ lệnh của CPU sẽ được trỏ tới đoạn mã của chương trình con phục vụ ngắt. Khi chương trình con phục vụ ngắt kết thúc, CPU khôi phục lại trạng thái các thanh ghi của chương trình chính và đặt con trỏ lệnh về vị trí bị ngừng khi phục vụ ngắt. Dưới đây là danh sách một số kiểu ngắt đặc biệt được xếp vào đầu dãy ngắt mềm INT N như sau:

+ INT 0: Ngắt mềm do phép chia cho số 0 gây ra,

+ INT1: Ngắt mềm để chạy từng lệnh ứng với trường hợp cờ TF=1,

+ INT2: Ngắt cứng do tín hiệu tích cực tại chân NMI gây ra,

+ INT3: Ngắt mềm để đặt điểm dừng của chương trình tại một địc chỉ nào đó

+ IN T4: (Hoặc lệnh INTO): ngắt mềm ứng với trường hợp cờ tràn OF=1.

Các kiểu ngắt khác còn lại thì được dành cho nhà sản xuất và cho người sử dụng định nghĩa:

+ INT 5-INT 1FH; dành riêng cho Intel trong các bộ vi xử lý cao cấp khác,

+ INT 20H-INT FFH: dành cho người sử dụng.

Các kiểu ngắt N trong INT N đều tương ứng với các địa chỉ xác định của chương trình con phục vụ ngắt mà ta có thể tra được trong bảng các vectơ ngắt. Intel quy định bảng này nằm trong RAM bắt đầu từ đạic hỉ 00000H và dài 1 KB (vì 8086/8088 có tất cả 256 kiểu ngắt, mỗi kiểu ngắt ứng với 1 vectơ ngắt, 1 vectơ ngắt cần 4 byte để chứa địa chỉ đầy đủ cho CS:IP của chương trình con phục vụ ngắt).

Bảng II-8. Bảng vectơ ngắt của 8086/8088 tại 1KB RAM đầu tiên

03FEH-03FFH	CS của chương trình con phục vụ ngắt INT FFH
03FCH-03FDH	IP của chương trình con phục vụ ngắt INT FFH
0082H-0083H	CS của chương trình con phục vụ ngắt INT 20H
0080H-0081H	IP của chương trình con phục vụ ngắt INT 20H
000AH-000BH	CS của chương trình con phục vụ ngắt INT 2
0008H-0009H	IP của chương trình con phục vụ ngắt INT 2
0006H-0007H	CS của chương trình con phục vụ ngắt INT 1
0004H-0005H	IP của chương trình con phục vụ ngắt INT 1
0002H-0003H	CS của chương trình con phục vụ ngắt INT 0
0000H-0001H	IP của chương trình con phục vụ ngắt INT 0

II.4.4 Xử lý ưu tiên khi ngắt

Có một vấn đề rất thực tế đặt ra là nếu tại cùng một thời điểm có nhiều yêu cầu ngắt thuộc các loại ngắt khác nhau cùng đòi hỏi CPU phục vụ thì CPU sẽ phải có cơ chế để xử lý các yêu cầu ngắt này. Cơ chế phổ biến là chia các ngắt theo mức ưu tiên. CPU 8086/8088 có khả năng phân biệt các mức ưu tiên khác nhau cho các loại ngắt (theo thứ tự từ cao xuống thấp) như sau:

+ ngắt trong: INT 0 (phép chia cho 0), INT N, INTO . . . cao nhất

+ ngắt không che được NMI

+ ngắt che được INTR

+ ngắt để chạy từng lệng INT 1 . . . thấp nhất

Theo thứ tự ưu tiên ngầm định trong việc xử lý ngắt của CPU 8086/8088 thì INT 0 có mức ưu tiên cao hơn INTR, vì vậy đầu tiên CPU sẽ thực hiện chương trình phục vụ ngắt INT 0 để đáp ứng với lỗi đặc biệt cho phép chua cho 0 gây ra và cờ IF bị xóa về 0. Yêu cầu ngắt INTR sẽ tự động bị cấm cho tới khi chương trình phục vụ ngắt INT 0 được hoàn tất và trở về nhờ IRET, cờ IF cũ được trả lại. Tiếp theo đó CPU sẽ đáp ứng yêu cầu ngắt INTR bằng cách thực hiện chương trình phục vụ ngắt dành cho INTR.

Chương III. Lập trình hợp ngữ với 8086/8088

III.1 Giới thiệu khung của chương trình hợp ngữ

III.1.1 Cú pháp của chương trình hợp ngữ

Một chương trình hợp ngữ bao gồm các dòng lệnh, một dòng lệnh có thể là một lệnh thật dưới dạng ký hiệu (symbolic), mà đôi khi còn được gọi là dạng gợi nhớ (mnemonic) của bộ vi xử lý, hoặc một hướng dẫn cho chương trình dịch (assembler directive). Lệnh gợi nhớ sẽ được dịch ra mã máy còn hướng dẫn cho chương trình dịch thì không được dịch vì nó chỉ có tác dụng chỉ dẫn riêng thực hiện công việc. Các dòng lệnh này có thể được viết bằng chữ hoa hoặc chữ thường và chúng sẽ được coi là tương đương vì đối với dòng lệnh chương trình dịch không phân biệt kiểu chữ.

Một dòng lệnh của chương trình hợp ngữ có thể có những trường sau (không nhất thiết phải có đủ hết tất cả các trường):

Tên Mã lệnh Các toán dạng Chú giải

Một ví dụ dòng lệnh gợi nhớ:

TIEP: MOV AH, [BX] [SI] ; nạp vào AH ô nhớ có địa chỉ DS: (BX+SI)

Trong ví dụ trên, tại trường tên ta có nhãn TIEP, tại trường mã lệnh ta có lệnh MOV, tại trường toán hạng ta có các thanh ghi AH, BX và SI và phần chú giải gồm có các dòng

; nạp vào AH ô nhớ có địa chỉ DS: (BX+SI)

Một ví dụ khác là các dòng lệnh với các hướng dẫn cho chương trình dịch:

MAIN PROC

và

MAIN ENDP

Trong ví dụ này, ở trường tên ta có tên thủ tục là MAIN, ở trường mã lệnh ta có các lệnh giả PROC và ENDP. Đây là các lệnh giả dùng để bắt đầu và kết thúc một thủ tục có tên là MAIN.

a) Trường tên

Trường tên chứa các nhãn, tên biến hoặc tên thủ tục. Các tên và nhãn này sẽ được chương trình dịch gán bằng các địa chỉ cụ thể của ô nhớ. Tên và nhẫn có thể có độ dài 1. . 31 ký tự, không được chứa dấu cách và không được bắt đầu bằng số. Các ký tự đặc biệt khác có thể dùng trong tên là ?. @_$%. Nếu dấu chấm ('. ') được dùng thì nó phải được đặt ở vị trí đầu tiên của tên. Một nhãn thường kết thúc bằng dấu hai chấm (:).

b) Trường mã lệnh

Trong trường mã lệnh nói chung sẽ có các lệnh thật hoặc lệnh giả. Đối với các lệnh thật thì trường này chứa các mã lệnh gợi nhớ. Mã lệnh này sẽ được chương trình dịch dịch ra mã máy. Đối với các hướng dẫn chương trình dịch thì trường này chứa các lệnh giả và sẽ không được dịch ra mã máy.

c) Trường toán hạng

Đối với một lệnh thì trường này chứa các toán hạng của lệnh. Tùy theo từng loại lệnh mà ta có thể có 0, 1 hoặc 2 toán hạng trong một lệnh. Trong trường hợp các lệnh với 1 toán hạng thông thường ta có toán hạng là đích hoặc gốc, còn trong trường hợp lệnh với 2 toán hạng thì ta có 1 toán hạng là đích và 1 toán hạng là gốc.

Đối với hướng dẫn chương trình dịch thì trường này chứa các thông tin khác nhau liên quan đến các lệnh giả của hướng dẫn.

d) Trường chú giải

Lời giải thích ở trường chú giải phải được bắt đầu bằng dấu chấm phẩy (;). Trường chú giải này được dành riêng cho người lập trình để ghi các lời giải thích cho các lệnh của chương trình với mục đích giúp cho người đọc chương trình dễ hiểu các thao tác của chương trình hơn. Thông thường lời chú giải cần phải mang đủ thông tin để giải thích về thao tác của lệnh trong hoàn cảnh cụ thể và như thế thì mới có ích cho người đọc.

III.1.2 Dữ liệu cho chương trình

Dữ liệu của một chương trình hợp ngữ là rất đa dạng. Các dữ liệu có thể được cho dưới dạng số hệ hai, hệ mười, hệ mười sáu hoặc dưới dạng ký tự. Khi cung cấp số liệu cho chương trình, số cho ở hệ nào phải được kèm đuôi của hệ đó (trừ hệ mười thì không cần vì là trường hợp ngầm định của assembler). Riêng đối với số hệ mười sáu nếu số đó bắt đầu bằng các chữ (a. f hoặc A. . F) thì ta phải thêm 0 ở trước để chương trình dịch có thể hiểu được đó là một số hệ mười sáu chứ không phải là một tên hoặc một nhãn.

Ví dụ các số viết đúng:

```
0011B          ; Số hệ hai.
1234           ; Số hệ mười
0ABBAH         ; Số hệ mười sáu
1EF1H          ; Số hệ mười sáu.
```

Nếu dữ liệu là ký tự hoặc chuỗi ký tự thì chúng phải được đóng trong cặp dấu trích dẫn đơn hoặc kép, thí dụ 'A' hay "abcd". Chương trình dịch sẽ dịch ký tự ra mã ASCII tương ứng của nó. Vì vậy trong khi cung cấp dữ liệu kiểu ký tự cho chương trình ta có thể dùng bản thân ký tự được đóng trong dấu trích dẫn hoặc mã ASCII của nó. Ví dụ, ta có thể sử dụng liệu ký tự là "0" hoặc mã ASCII tương ứng là 30H, ta có thể dùng '$' hoặc 26H hoặc 34. . .

III.1.2.1 Biến và hằng

Biến trong chương trình hợp ngữ có vai trò như nó có ở ngôn ngữ bậc cao. Một biến phải được định kiểu dữ liệu là kiểu byte hay kiểu từ và sẽ được chương trình dịch gán cho

một địa chỉ nhất định trong bộ nhớ. Để định nghĩa các kiểu dữ liệu khác nhau ta thường dùng các lệnh giả sau:

DB (define byte)	: định nghĩa biến kiểu byte
DW (define word)	: định nghĩa biến kiểu từ
DD (define double word)	: định nghĩa biến kiểu từ kép

a) Biến byte

Biến kiểu byte sẽ chiếm 1 byte trong bộ nhớ. Hướng dẫn chương trình dịch để định nghĩa biến kiểu byte có dạng tổng quát như sau:

```
Tên     DB     giá_ trị_khởi_đầu
```

Ví dụ:

```
B1      DB     4
```

Ví dụ trên định nghĩa biến byte có tên là B1 và dành 1 byte trong bộ nhớ cho nó để chứa giá trị khởi đầu bằng 4.

Nếu trong lệnh trên ta dùng dấu? thay vào vị trí của số 4 thì biến B1 sẽ được dành chỗ trong bộ nhớ nhưng không được gán giá trị khởi đầu. Cụ thể dòng lệnh giả:

```
B2      DB     ?
```

chỉ định nghĩa 1 biến byte có tên là B2 và dành cho nó một byte trong bộ nhớ.

Một trường hợp đặc biệt của biến byte là biến ký tự. Ta có thể có định nghĩa biến kỳ tự như sau:

```
C1      DB     ' $'
C2      DB     34
```

b) Biến từ

Biến từ cũng được định nghĩa theo cách giống như biến byte. Hướng dẫn chương trình dịch để định nghĩa biến từ có dạng như sau:

```
Tên     DB     giá_ trị_khởi_đầu
```

Ví dụ:

```
W1      DW     40
```

Ví dụ trên định nghĩa biến từ có tên là W1 và dành 2 byte trong bộ nhớ cho nó để chứa giá trị khởi đầu bằng 40.

Chúng ta cũng có thể sử dụng dấu? chỉ để định nghĩa và dành 2 byte trong bộ nhớ cho biến từ W2 mà không gán giá trị đầu cho nó bằng dòng lệnh sau:

```
W2      DW     ?
```

c) Biến mảng

Biến mảng là biến hình thành từ một dãy liên tiếp các phần tử cùng loại byte hoặc từ, khi định nghĩa biến mảng ta gán tên cho một dãy liên tiếp các byte hay từ trong bộ nhớ cùng với các giá trị ban đầu tương ứng.

Ví dụ:

```
M1    DB    4, 5, 6, 7, 8, 9
```

Ví dụ trên định nghĩa biến mảng có tên là M1 gồm 6 byte và dành chỗ cho nó trong bộ nhớ từ địa chỉ ứng với M1 để chứa các giá trị khởi đầu bằng 4, 5, 6, 7, 8, 9. Phần tử đầu tỏng mảng là 4 và có địa chỉ trùng với địa chỉ của M1, phần tử thứ hai là 5 và có địa chỉ M1+1. . .

Khi chúng ta muốn khởi đầu các phần tử của mảng với cùng một giá trị chúng ta có thể dùng thêm toán tử DUP trong lệnh.

Ví dụ:

```
M2    DB    100    DUP (0)
M3    DB    100    DUP (?)
```

Ví dụ trên định nghĩa một biến mảng tên là M2 gồm 100 byte, dành chỗ trong bộ nhớ cho nó để chứa 100 giá trị khởi đầu bằng 0 và biến mảng khác tên là M3 gồm 100byte, dành sẵn chỗ cho nó trong bộ nhớ để chứa 100 giá trị nhưng chưa được khởi đầu.

Toán tử DUP có thể lồng nhau để định nghĩa ra 1 mảng.

Ví dụ: dòng lệnh

```
M4    DB    4, 3, 2, 2 DUP(1, 2 DUP(5), 6)
```

Sẽ định nghĩa ra một mảng M4 tương đương với lệnh sau:

```
M4    DB    4, 3, 2, 1, 5, 5, 6, 1, 5, 5, 6
```

Một điều cần chú ý nữa là đối với các bộ vi xử lý của Intel, nếu ta có một từ đặt trong bộ nhớ thì byte thấp của nó sẽ được đặt vào ô nhớ có địa chỉ thấp, byte cao sẽ được đặt vào ô nhớ có địa chỉ cao. Cách lưu giữ số liệu kiểu này cũng còn có thể thấy ở các máy VAX của Digital hoặc của một số hãng khác và thường gọi là 'quy ước đầu bé' (little endian, byte thấp được cất tại địa chỉ thấp). Cũng nên nói thêm ở đây là các bộ vi xử lý của motorola lại có cách cất số liệu theo thứ tự ngược lại hay còn được gọi là 'quy ước đầu to' (big endian byte cao được cất tại địa chỉ thấp).

Ví dụ: Sau khi định nghĩa biến từ có tên là WORDA như sau:

```
WORDA    DW    0FFEEH
```

Thì ở trong bộ nhớ thấp (EEH) sẽ được để tại địa chỉ WORDA còn byte cao (FFH) sẽ được để tại địa chỉ tiếp theo, tức là tại WORDA+1

d) Biến kiểu xâu kí tự

Biến kiểu xâu kí tự là một trường hợp đặc biệt của biến mảng, trong đó các phần tử của mảng là các kí tự. Một xâu kí tự có thể được định nghĩa bằng các kí tự hoặc bằng mã ASCII của các kí tự đó. Các ví dụ sau đều là các lệnh đúng và đều định nghĩa cùng một xâu kí tự nhưng gắn nó cho các tên khác nhau:

```
STR1  DB    'string'
STR2  DB    73h, 74h, 72h, 69h, 6Eh, 67h
```

```
STR3  DB     73h, 74h, 'x' 'i', 6Eh, 67h
```

e) Hằng có tên

Các hằng trong chương trình hợp ngữ thường được gán tên để làm cho chương trình trở nên dễ đọc hơn. Hằng có thể là kiểu số hay kiểu ký tự. Việc gán tên cho hằng được thực hiện nhờ lệnh giả EQU như sau:

```
CR     EQU   0Dh    ;CR là carriage return
LE     EQU   0Ah    ;LF là line feed
```

Trong ví dụ trên lệnh giả EQU gán giá trị số 13 (mã ASCII của kí tự trở về đầu dòng) cho tên CR và 10 (mã ASCII của ký tựu thêm dòng mới) cho tên LF.

Hằng cũng có thể là một chuỗi ký tự. trong ví dụ dưới đây sau khi đã gán một chuỗi ký tự cho một tên:

```
CHAO EQU   'Hello'
```

ta có thể sử dụng hằng này để định nghĩa một biến mảng khác.

```
MSG  DB    CHAO, '$'
```

Vì lệnh giả EQU không dành chỗ của bộ nhớ cho tên của hằng nên ta có thể đặt nó khá tự do tại những chỗ thích hợp bên trong chương trình. Tuy nhiên trong thực tế người ta thường đặt các định nghĩa này trong đoạn dữ liệu.

III.1.2.2 Khung của một chương trình hợp ngữ

Một chương trình mã máy trong bộ nhớ thường bao gồm các vùng nhớ khác nhau để chứa mã lệnh, chứa dữ liệu của chương trình và một vùng nhớ khác được dùng làm ngăn xếp phục vụ hoạt động của chương trình. Chương trình viết bằng hợp ngữ cũng phải có cấu trúc tương tự để khi được dịch nó sẽ tạo ra mã tương ứng với chương trình mã máy nói trên. Để tạo ra sườn của một chương trình hợp ngữ chúng ta sẽ sử dụng cách định nghĩa đơn giản đối với mô hình bộ nhớ dành cho chương trình và đối với các thanh ghi đoạn.

III.1.2.2.a Khai báo quy mô sử dụng bộ nhớ

Kích thước của bộ nhớ dành cho đoạn mã và đoạn dữ liệu trong một chương trình được xác định nhờ hướng dẫn chương trình dịch MODEL như sau (hướng dẫn này phải được đặt trước các hướng dẫn khác trong chương trình hợp ngữ, nhưng sau hướng dẫn về loại CPU):

```
. MODEL     Kiểu_ kích_thước_bộ_nhớ
```

Có nhiều Kiểu_ kích_thước_bộ_nhớ cho các chương trình với đòi hỏi dung lượng bộ nhớ khác nhau. Đối với ta thông thường các ứng dụng đòi hỏi mã chương trình dài nhất cũng chỉ cần chứa trong một đoạn (64KB), dữ liệu cho chương trình nhiều nhất cũng chỉ cần chứa trong một đoạn, thích hợp nhất nên chọn Kiểu_ kích_thước_bộ_nhớ là Small (nhỏ) hoặc nếu như tất cả mã và dữ liệu có thể gói trọn được trong một đoạn thì có thể chọn Tiny (hẹp):

```
. Model     Small
```

hoặc

```
. Model     Tiny
```

Ngoài Kiểu_ kích_thước_bộ_nhớ nhỏ hoặc hẹp nói trên, tuỳ theo nhu cầu cụ thể MASM còn cho phép sử dụng các Kiểu_ kích_thước_bộ_nhớ khác như liệt kê trong Bảng III-1.

Bảng III-1. Các kiểu kích thước bộ nhớ cho chương trình hợp ngữ

Kiểu kích thước	Mô tả
Tiny (Hẹp)	Mã lệnh và dữ liệu gói gọn trong một đoạn
Small (Nhỏ)	Mã lệnh gói gọn trong một đoạn, dữ liệu nằm trong một đoạn.
Medium (Trung bình)	Mã lệnh không gói gọn trong một đoạn, dữ liệu nằm trong một đoạn.
Compact(Gọn)	Mã lệnh không gói gọn trong một đoạn, dữ liệu không gói gọn trong một đoạn.
Large (lớn)	Mã lệnh không gói gọn trong một đoạn, dữ liệu không gói gọn trong một đoạn, không có mảng nào lớn hơn 64KB.
Huge (Đồ sộ)	Mã lệnh không gói gọn trong một đoạn, dữ liệu không gói gọn trong một đoạn, các mảng có thể lớn hơn 64KB

III.1.2.2.b Khai báo đoạn ngăn xếp

Việc khai báo đoạn ngăn xếp là để dành ra một vùng nhớ đủ lớn dùng làm ngăn xếp phục vụ cho hoạt động của chương trình khi có chương trình con. Việc khai báo được thực hiện nhờ hướng dẫn chương trình dịch như sau.

. Stack Kích_thước

Kích_thước sẽ quyết định số byte dành cho ngăn xếp. Nếu ta không khai Kích_thước thì chương trình dịch sẽ tự động gán cho Kích_thước giá trị 1 KB, đây là kích thước ngăn xếp quá lớn đối với một ứng dụng thông thường. Trong thực tế các bài toán của ta thông thường với 100-256 byte là đủ để làm ngăn xếp và ta có thể khai báo kích thước như sau:

```
. Stack          100
```

Khai báo đoạn dữ liệu

Đoạn dữ liệu chứa toàn bộ các định nghĩa cho các biến của chương trình. Các hằng cũng nên được định nghĩa ở đây để đảm bảo tính hệ thống mặc dù ta có thể để chúng ở trong chương trình như đã nói ở phần trên.

Việc khai báo đoạn dữ liệu được thực hiện nhờ hướng dẫn chương trình dịch DATA, việc khai báo và hằng được thực hiện tiếp ngay sau đó bằng các lệnh thích hợp. Điều này được minh hoạ trong các thí dụ đơn giản sau:

```
. Data
        MSG   DB          'helo!$'
```

```
CR      DB          13
LF      EQU         10
```

III.1.2.2.c Khai báo đoạn mã

Đoạn mã chứa mã lệnh của chương trình. Việc khai báo đoạn mã được thực hiện nhờ hướng dẫn chương trình dịch. CODE như sau:

```
. CODE
```

Bên trong đoạn mã, các dòng lệnh phải được tổ chức một cách hợp lý, đúng ngữ pháp dưới dạng một chương trình chính (CTC) và nếu cần thiết thì kèm theo các chương trình con (ctc). Các chương trình con sẽ được gọi ra bằng các lệnh CALL có mặt bên trong chương trình chính.

Một thủ tục được định nghĩa nhờ các lệnh giả PROC và ENDP. Lệnh giả PROC để bắt đầu một thủ tục còn lệnh giả ENDP được dùng để kết thúc nó. Như vậy một chương trình chính có thể được định nghĩa bằng các lệnh giả PROC và ENDP theo mẫu sau:

```
Tên_CTC         Proc

; Các lệnh của thân chương trình chính

CALL Tên_ ctc; gọi ctc

Tên_CTC         Endp
```

Giống như chương trình chính con cũng được định nghĩa dưới dạng một thủ tục nhờ các lệnh giả PROC và ENDP theo mẫu sau:

```
Tên_ctc  Proc

; các lệnh thân chương trình con

RET

Tên_ctc Endp
```

Trong các chương trình nói trên, ngoài các lệnh giả có tính nghi thức bắt buộc ta cần chú ý đến sự bố trí của lệnh gọi (CALL) trong chương trình chính và lệnh về (RET) trong chương trình con.

III.1.2.2.d Khung của chương trình hợp ngữ để dịch ra chương trình. EXE

Từ các khai báo các đoạn của chương trình đã nói ở trên ta có thể xây dựng một khung tổng quát cho các chương trình hợp ngữ với kiểu kích thước bộ nhớ nhỏ. Sau đây là một khung cho chương trình hợp ngữ để rồi sau khi được dịch (assembled), nối (linked) trên máy IBM PC sẽ tạo ra một tệp chương trình chạy được ngay (executable) với đuôi. EXE.

```
. Model small
. Stack 100
. Data
        ; các định nghĩa cho biến và hằng để tại đây
```

```
. Code
MAIN Proc
        ; Khởi đầu cho DS
         MOV AX, @Data
         MOV DS, AX
        ; Các lệnh của chương trình chính để tại đây
        ; Trở về DOS dùng hàm 4CH của INT 21H
         MOV AH, 4CH
         INT 21 H
MAIN Endp
        ; các chương trình con (nếu có)        để tại đây
        END MAIN
```

Trong khung chương trình trên, tại dòng cuối cùng của chương trình ta dùng hướng dẫn chương trình dịch END và tiếp theo là MAIN để kết thúc toàn bộ chương trình. Ta có nhận xét rằng MAIN là tên của chương trình chính nhưng quan trọng hơn và về thực chất thì nó là nơi bắt đầu các lệnh của chương trình trong đoạn mã.

Khi một chương. EXE được nạp vào bộ nhớ. Hệ điều hành DOS sẽ tạo ra một mảng gồm 256 byte của cái gọi là *đoạn mào đầu chương trình* (Program Segment Prefix - PSP) dùng để chứa các thông tin liên quan đến chương trình và các thanh ghi DS và ES. Do vậy DS và ES không chứa giá trị địa chỉ của các đoạn dữ liệu cho chương trình của chúng ta. Để chương trình có thể chạy đúng ta phải có các lệnh sau để khởi đầu cho thanh ghi DS (hoặc ES nếu cần):

```
MOV AX, @Data
MOV DS, AX
```

Trong đó @Data là tên của đoạn dữ liệu. Data định nghĩa bởi hướng dẫn chương trình dịch sẽ dịch tên @Data thành giá trị số của đoạn dữ liệu. Ta phải dùng thanh ghi AX làm trung gian cho việc khởi đầu DS như trên là do bộ vi xử lý 8086/8088, Vì những lí do kỹ thuật, không cho phép chuyển giá trị số (chế độ địa chỉ tức thì) vào các thanh ghi đoạn. Thanh ghi AX cũng có thể được thay thế bằng các thanh ghi khác.

Sau đây là ví dụ của một chương trình hợp ngữ được viết để dịch ra chương trình với đuôi. EXE. khi cho chạy, chương trình này sẽ hiện lên màn hình lời chào 'Hello' nằm giữa hai dòng trống cách đều các dòng mang dấu nhắc của DOS.

Ví dụ III-1. Chương trình Hello. EXE

```
. Model Small
. Stack 100
. Data
        CRLF        DB      13, 10, ' $ '
        MSG         DB      ' Hello!$ '
. Code
MAIN Proc
                ; khởi đầu thanh ghi DS
        MOV AX, @Data
```

```
        MOV DS, AX
            ; về đầu dòng mới dùng hàm 9 của INT 21H
        MOV AH, 9
        LEA DX, CRLF
        INT 21H
            ; hiện thị lời chào dùng hàm 9 của INT 21H
        MOV AH, 9
        LEA DX, MSG
        INT 21H
            ; về đầu dòng mới dùng hàm 9 của INT 21H
        MOV AH, 9
        LEA DX, CFLF
        INT 21H
            ; trở về DOS dùng hàm 9 của INT 21H
        MOV AH, 4CH
        INT 21H
MAIN Endp
        END MAIN
```

Trong ví dụ trên chúng ta đã sử dụng các dịch vụ có sẵn (các hàm 9 và 4CH) của ngắt INT 21H của DOS trên máy IBM PC để hiện thị xâu ký tự và trở về DOS một cách thuận lợi.

III.1.2.2.e Khung của chương trình hợp ngữ để dịch ra chương trình. COM

Nhìn vào khung chương trình hợp ngữ để dịch ra tệp chương trình đuôi. EXE ta thấy có mặt đầy đủ các đoạn. Trên máy tính IBM PC ngoài tệp chương trình với đuôi. EXE. Chúng ta còn có khả năng dịch chương trình hợp ngữ có kết cấu thích hợp ra một loại tệp chương trình chạy được kiểu khác với đuôi. COM. Đây là một chương trình ngắn gọn và đơn giản hơn nhiều so với tệp chương trình đuôi. EXE, trong đó các đoạn mã, đoạn dữ liệu và đoạn ngăn xếp được gộp lại trong một đoạn duy nhất là đoạn mã. Như vậy nếu ta có các ứng dụng mà dữ liệu và mã chương trình không yêu cầu nhiều về không gian của bộ nhớ, ta có thể ghép luôn cả dữ liệu, mã chương trình và ngăn xếp chung vào trong cùng một đoạn mã rồi tạo ra tệp. COM. Với việc tạo ra tệp này còn tiết kiệm được cả không gian nhớ khi phải lưu trữ nó trên ổ đĩa. Để có thể dịch được ra chương trình đuôi. COM thì chương trình nguồn hợp ngữ phải được kết cấu sao cho thích hợp với mục đích này.

Sau đây là khung của một chương trình hợp ngữ để dịch được ra tệp chương trình đuôi .COM.

Ví dụ III-2. Khung chương trình. COM

```
. Model Tiny
. Code
        ORG 100h
START: JMP CONTINUE
;       các định nghĩa cho biến và hằng để tại đây
CONTINUE:
MAIN Proc
;       các lệnh của chương trình chính để tại đây
 INT 20H                ; Trở về DOS
MAIN Endp
;       các chương trình con (nếu có) để tại đây
 END START
```

So sánh khung này với khung cho chương trình. EXE ta thấy trong khung không có khai báo đoạn ngăn xếp và đoạn dữ liệu, còn khai báo quy mô sử dụng nhớ là kiểu Tiny. Ở ngay đầu đoạn mã là lệnh giả ORG (origin: điểm xuất phát) lệnh JMP (nhảy). Lệnh giả ORH 100H dùng để gán địa chỉ bắt đầu cho chương trình tại 100H trong đoạn mã, chừa lại vùng nhớ với dung lượng 256 byte (từ địa chỉ 0 đến địa chỉ 255) cho đoạn mào đầu chương trình (PSP).

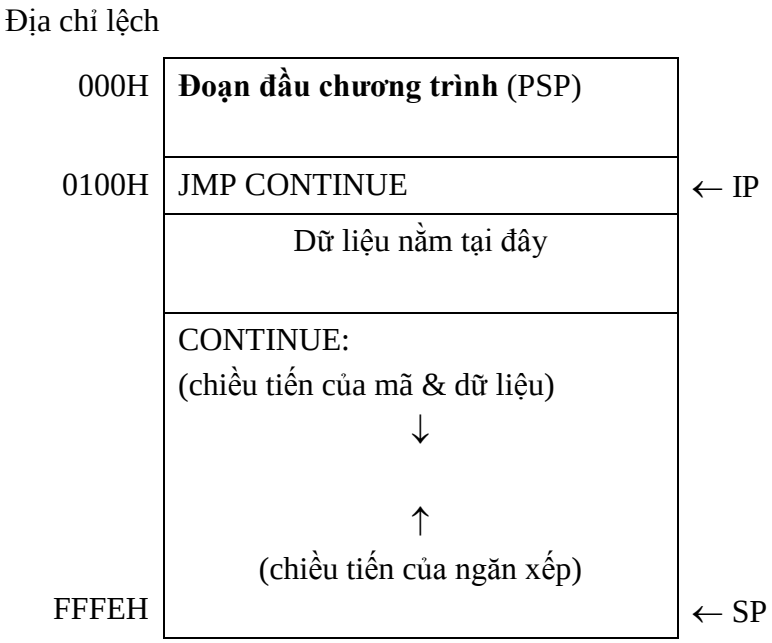
Lệnh JMP sau nhãn START dùng để nhảy qua phần bộ nhớ dành cho việc định nghĩa và khai báo dữ liệu (về nguyên tắc, dữ liệu có thể được đặt ở đầu hoặc ở cuối đoạn mã, nhưng ở đây ta đặt nó ở đầu đoạn mã để có thể áp dụng các định nghĩa đơn giản đã nói). Đích của lệnh nhảy là phần đầu của chương trình chính. Hình III-1 biểu diễn việc một chương trình kiểu. COM được nạp vào và sắp xếp trong một đoạn mã của bộ nhớ ra sao.

Theo Hình III-1 ta thấy một chương trình. COM cũng được nạp vào bộ nhớ sau vùng PSP như chương trình đuôi. EXE. Ngăn xếp cho chương trình. COM được xếp đặt tại cuối đoạn mã, đỉnh của ngăn xếp lúc ban đầu là ô nhớ có địa chỉ là FFFEH.

Trong trường hợp chương trình kiểu. COM này chúng ta sẽ bị các hạn chế

– Dung lượng nhớ cực đại của một đoạn là 64KB, tức là ta phải luôn chắc chắn được rằng các chương trình ứng dụng phải có số lượng byte của mã máy và dữ liệu cho chương trình không lớn lắm.

– Chương trình cũng chỉ được phép sử dụng ngăn xếp một cách hạn chế (nếu không điều này có thể làm cho đỉnh của nó trong khi hoạt động dâng lên nhiều về phía địa chỉ thấp của đoạn).



Hình III-1. Tệp chương trình. COM trong bộ nhớ

Tóm lại chúng ta phải chắc chắn đảm bảo không thể xảy ra hiện tượng trùm vào nhau của các thông tin tại vùng mã lệnh hoặc dữ liệu. Khi kết thúc chương trình kiểu. COM, để trở về DOS ta dùng ngắt INT 20H của DOS để làm cho chương trình gọn hơn. Tất nhiên ta cũng có thể dùng hàm 4CH của ngắt INT 21H như đã dùng trong chương trình để dịch ra tệp. EXE.

Để kết thúc toàn bộ chương trình ta dùng hướng dẫn chương chính dịch END đi kèm theo nhãn START tương ứng với địa chỉ lệnh đầu tiên của chương trình trong đoạn mã.

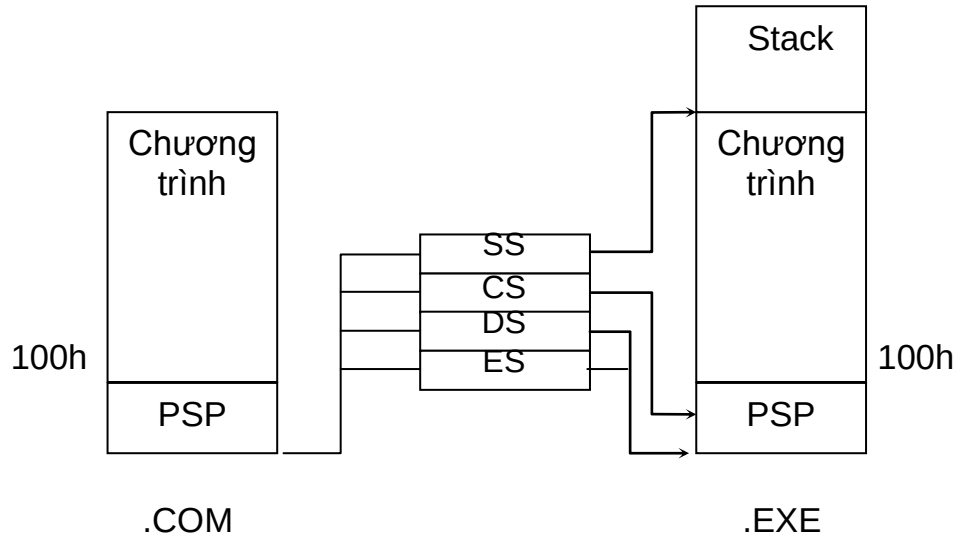
Sau đây là ví dụ của một chương trình hợp ngữ để dịch ra tệp chương trình chạy được với đuôi. COM.

Ví dụ III-3. Chương trình Helo. COM

```
. Model Tiny
. Code
ORG 100H
START:      IMP CONTINUE
CRLF        DB    13, 10, '$'
MSG         DB    !Hello! $'
CONTINUE:
MAIN Proc
; về đầu dòng mới dùng hàm 9 của INT 21H
MOV AH, 9
LEA DX, CRLF
INT 21H
```

```
        ; hiện thị lời chào
                MOV AH, 9
                LEA DX, CRLF
                INT 21H
        ; trở về DOS
                INT 20H
MAIN Endp
                END START
```

Trong Ví dụ III-3 ta không cần đến các thao tác khởi đầu cho thanh ghi DS, như ta đã phải làm trong Ví dụ III-1, vì trong chương trình. COM không có đoạn dữ liệu nằm riêng rẽ.



Hình III-2. Môđun chương trình. COM và. EXE trong bộ nhớ.

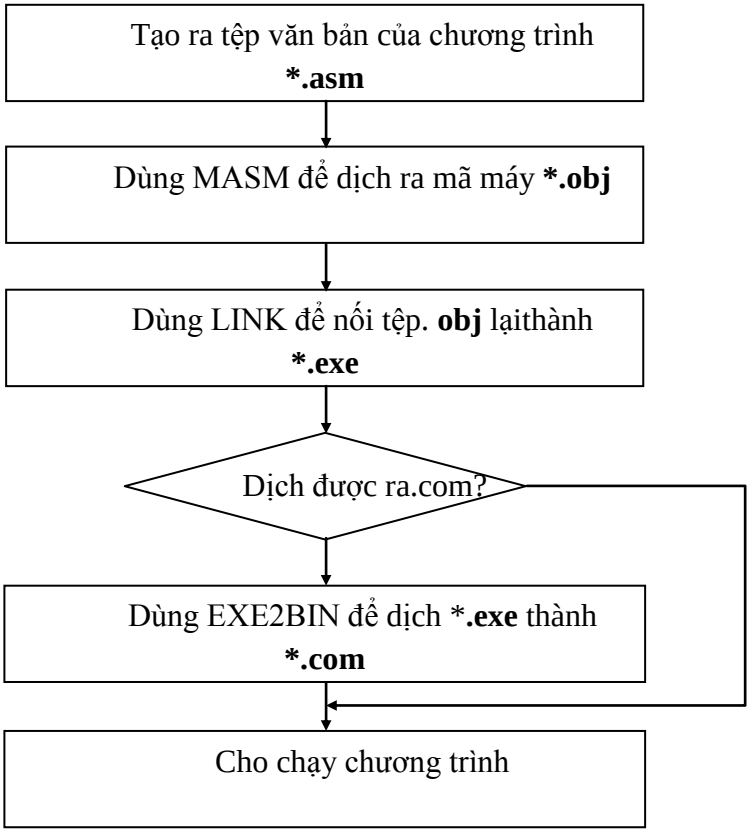
Cuối cùng để kết thúc phần nói về các chương trình kiểu. COM và. EXE ta đưa ra hình ảnh của các chương trình này khi chúng được tải vào trong bộ nhớ để có thể tiện so sánh (Hình III-2).

III.2 Cách tạo và chạy chương trình hợp ngữ

Như đã nói trong phần trước, máy IBM PC là phương tiện lý tưởng để chúng ta tạo ra và thử nghiệm các chương trình hợp ngữ 8086/88. Các bước để làm công việc này có thể liệt kê ra như sau:

1. Dùng các phần mềm soạn thảo văn bản (SK, NCedit. . .) để tạo ra một tệp văn bản chương trình gốc bằng hợp ngữ. Tệp này phải được gán đuôi. ASM.
2. Dùng chương trình dịch MASM để dịch tệp. ASM ra mã máy dưới dạng tệp. OBJ. Nếu trong bước này nếu trong chương trình có lỗi cú pháp thì ta phải quay lại bước 1 để sửa lại chương trình gốc.

3. Dùng chương trình LINK để nối một hay nhiều tệp OBJ lại với nhau thành một tệp chương trình chạy được với đuôi. EXE.
4. Nếu chương trình gốc viết ra là để dịch ra kiểu. COM thì ta phải dùng chương trình EXE2BIN (đọc là EXEtoBIN) của DOS để dịch tiếp tệp. EXE ra tệp chương trình chạy được với đuôi. COM.
5. Cho chạy chương trình vừa dịch



Hình III-3. Các bước để tạo ra và chạy chương trình hợp ngữ

III.3 Các cấu trúc lập trình cơ bản

Ngày nay, trong khi tiến hành việc thiết kế hệ thống người ta thường dùng phương pháp *thiết kế từ trên xuống dưới*. Bản chất của phương pháp thiết kế này là đầu tiên ta chia chương trình tổng thể thành các khối chức năng nhỏ hơn, các khối chức năng nhỏ này lại được chia tiếp thành các khối chức năng nhỏ hơn nữa, việc phân chia chức năng phải làm cho đến khi mỗi khối nhỏ này trở thành các khối chức năng đơn giản và dễ thực hiện.

Trong khi thực hiện các khối chức năng thành phần, thông thường người ta sử dụng các cấu trúc lập trình cơ bản để thực hiện các nhiệm cụ của khối đó. Điều này làm cho các chương trình viết ra trở thành có *cấu trúc* với các ưu điểm chính là dễ phát triển, dễ hiệu chỉnh hoặc cải tiến và dễ lập tài liệu.

Để giải quyết các công việc khác nhau thông thường trong khi viết chương trình ta chỉ cần đến 3 cấu trúc lập trình cơ bản sau:

+ Cấu trúc tuần tự.

+ Cấu trúc lựa chọn (IF-THEN-ELSE) và

+ Cấu trúc lặp (WHILE. DO).

Thay đổi các cấu trúc này một chút ít, ta có thể tạo thêm 4 cấu trúc khác cũng rất có tác dụng trong khi viết chương trình:

+ cấu trúc chọn kiểu IF-THEN

+ cấu trúc chọn kiểu CASE,

+ cấu trúc lặp kiểu REPEAT-UNTIL và

+ cấu trúc lặp kiểu FOR-DO.

Đặc điểm chung của tất cả các cấu trúc lập trình cơ bản là *tính cấu trúc* chi có một lối vào cấu trúc và một lối ra để ra khỏi cấu trúc đó.

III.3.1.1 Cấu trúc tuần tự

Cấu trúc tuấn tự là một cấu trúc thông dụng và đơn giản nhất. Trong cấu trúc này các lệnh được sắp xếp tuần tự, lệnh này kế tiếp lệnh kia. Sau khi thực hiện xong lệnh cuối cùng của cấu trúc thì công việc phải làm cũng được hoàn tất.

Ngữ pháp:

Lệnh 1

Lệnh 2

Lệnh n

Bài tập III-1

Các thanh ghi CX và BX chứa các giá trị của biến c và b. Hãy tính giá trị của biểu thức a = 2×(c+b) và chứa kết quả trong thanh ghi AX.

Giải

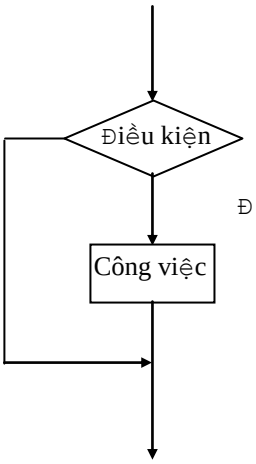
Ta có thể thực hiện công việc trên bằng mẫu chương trình sau:

```
XOR AX, AX      ;tổng tại AX lúc đầu là 0.
ADD AX, BX      ;cộng thêm b.
ADD AX, CX      ;cộng thêm c.
SHL AX, l       ;nhân đôi kết quả trong AX.
RA:             ;lối ra của cấu trúc.
```

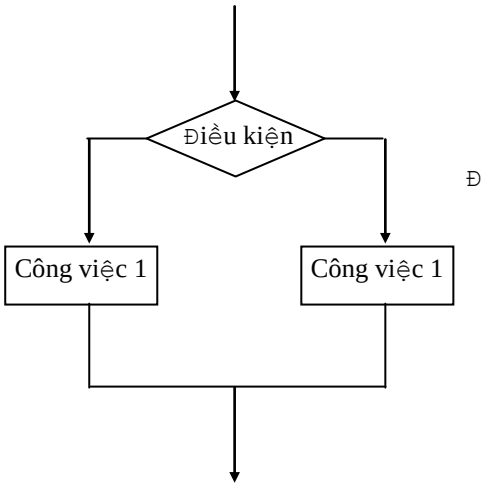
III.3.1.2 Cấu trúc IF - THEN

IF **Điều kiện** THEN **công việc.**

Từ ngữ pháp của cấu trúc IF-THEN ta thấy nếu thoả mãn Điều kiện thì Công việc được thực hiện nếu không Công việc sẽ bị bỏ qua. Điều này tương đương với việc dùng lệnh nhảy có điều kiện để bỏ qua một thao tác náo đó trong chương trình hợp ngữ.



Hình III-6.Cấu trúc IF-THEN



Hình III-5 Cấu trúc IF-THEN-ELSE

Bài tập III-2.

Gán cho BX giá trị tuyệt đối của AX.

Giải

Để thực hiện phép gán BX ← |AX| ta có thể dùng các lệnh sau:

```
        CMP AX, 0       ;   AX<0?
        JNL GAN         ;   không, gán luôn.
        NEG AX          ;   đúng. đào dấu, rồi
GAN:    MOV BX, AX      ;   lối ra của cấu trúc.
```

III.3.1.3 Cấu trúc IF - THEN - ELSE

IF **ĐiềuKiện** THEN **CôngViệc1** ELSE **CôngViệc2**

Từ ngữ pháp của cấu trúc IF-THEN-ELSE ta thấy nếu thoả mãn Điều kiện thì Côngviệc1 được thực hiện nếu không thì Côngviệc2 được thực hiện. Điều này tương đương với việc dùng lệnh nhảy có điều kiện và không điều kiện để nhảy đến các nhãn nào đó trong chương hợp ngữ.

Bài tập III-3.

Gán cho CL giá trị bít dấu của AX.

Giải

Ta có thể thực hiện các công việc trên bằng mẩu chương trình sau:

```
CMP AX, 0        ;     AX>0?.
JNS DG           ;     đúng.
MOV CL, l        ;     sai, cho CL ← 1 rồi
JMP RA           ;     di ra.
DG: XOR CL, CL   ;     cho CL ← 0.
RA:              ; lối ra của cấu trúc.
```

III.3.1.4 Cấu trúc CASE

CASE **Biểuthức**

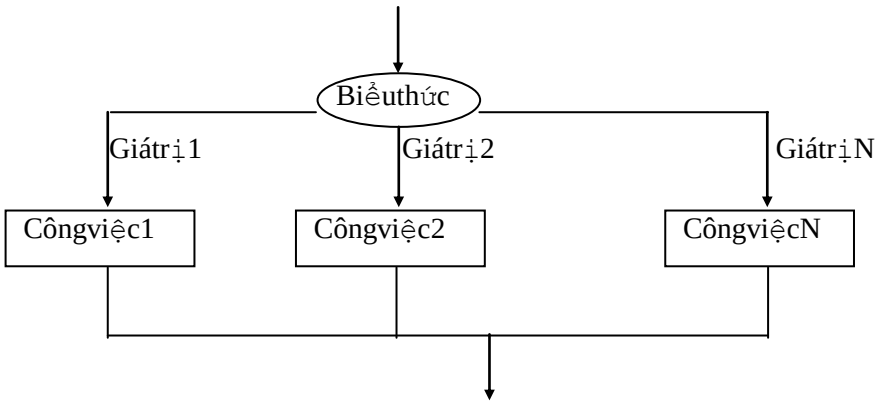
Giátrị1: Côngviệc1

Giátrị2: Côngviệc2

. . .

GiátrịN: CôngviệcN

END CASE



Hình III-7. Cấu trúc lệnh CASE

Từ ngữ pháp của cấu trúc ta thấy nếu Biểuthức có Giátrị1 thì Côngviệc1 được thực hiện. nếu Biểuthức có Giátrị2 thì Côngviệc2 được thực hiện và cứ tiếp tục cho đến CôngviệcN. Điều này tương đương với việc dùng các lệnh nhảy có điều kiện và nhảy không điều kiện để nhảy các nhãn nào đó trong chương trình hợp ngữ. Cấu trúc CASE có thể thực hiện bằng các cấu trúc lựa chọn lồng nhau.

Bài tập III-4.

Dùng CX để biểu hiện các giá trị khác nhau của AX theo quy tắc sau:

AX < 0 thì CX =-1

AX = 0 thì CX =0

AX > 0 thì CX =1

Giải

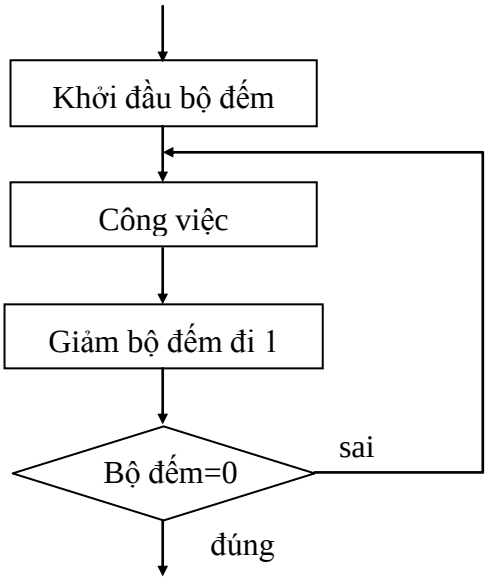
Ta có thể thực hiện các công việc trên bằng mẫu chương trình sau:

```
            CMP AX, 0       ; Kiểm tra dấu của AX.
            JL AM   ; AX<0.
            JE KHONG        ; AX =0.
            JG DUONG        ; AX > 0.
AM:         MOV CX, -1
            JMP RA
DUONG: MOV CX, 1
            JMP RA
KHONG: XOR CX. CX
RA:                         ; lối ra của cấu trúc.
```

III.3.1.5 Cấu trúc lặp FOR - DO

FOR **Số lần lặp** DO **Công việc**

Từ ngữ pháp cuả cấu trúc FOR - DO ta thấy ở đây Công việc được thực hiện lặp đi lặp lại tất cả Số lần lặp lại. Điều này hoàn toàn tương đương với việc dùng lệnh LOOP trong hợp ngữ để lặp lại CX lần một Công việc nào đó, trước đó ta phải gán Số lần lặp cho thanh ghi CX.



Hình III-8. Cấu trúc lặp FOR - DO.

Bài tập III-5

Hiển thị một dòng kí tự '$' trên màn hình.

Giải

Một dòng màn hình trên máy IBM PC chứa được nhiều nhất là 80 kí tự.

Ta sẽ sử dụng hàm 2 của ngắt 21H để hiển thị 1 kí tự. Ta phải lặp lại công việc này 80 lần cả thảy bằng lệnh LOOP. Muốn dùng lện này, ngay từ đầu ta phải nạp vào thanh ghi CX số lần hiển thị, nội dung của CX được tự động giảm đi1 do tác động của lệnh LOOP.

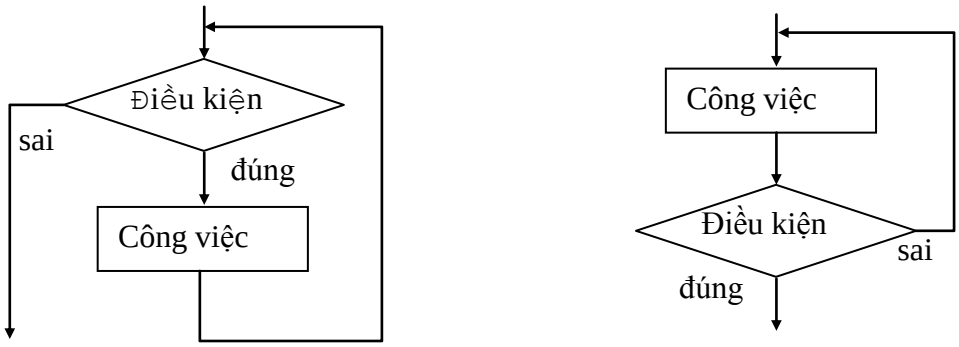
Sau đây là mẩu chương trình thực hiện các công việc trên:

```
        MOV CX, 80      ; số lần hiện thị trong CX
        MOV AH, 2       ; AH chứa số hiệu hàm hiện thị,
        MOV DL, '$'     ; DL chứa kí tự cần hiện thị,
HIEN:   INT 21H         ; hiện thị
        LOOP HIEN       ; cả một dòng kí tự.
RA:                     ; lối ra của cấu trúc.
```

III.3.1.6 Cấu trúc lặp WHILE - DO

WHILE **Điều kiện** DO **Công việc**

Từ ngữ pháp của cấu trúc **WHILE - DO** ta thấy: Điều kiện được kiểm tra đầu tiên. Công việc được lặp đi lặp lại chừng nào Điều kiện còn đúng. Điều này trong hợp ngữ hoàn toàn tương đương với việc dùng lệnh CMP để kiểm tra Điều kiện và sau đó dùng lệnh nhảy có điều kiện để thoát khỏi vòng lặp.



Hình III-9. Cấu trúc *WHILE - DO* và REPEAT-UNTIL

Bài tập III-6

Đếm số ký tự đọc được từ bàn phím, khi gặp ký tự CR thì thôi.

Giải

Ta có thể thực hiện công việc trên bằng mẩu chương trình sau:

```
        XOR CX, CX      ; tổng số ký tự đọc được lúc đầu là 0
        MOV AH, 1       ; hàm đọc ký tự từ bàn phím.
TIEP:   INT 21H         ; đọc 1 ký ự, Al chứa mã ký tự.
        CMP AL, 13      ; đọc được CR?
        JE RA           ; đúng, ra.
        INC CX          ; sai, thêm 1 ký tự vào tổng.
RA:                     ; lối ra của cấu trúc.
```

III.3.1.7 Cấu trúc lặp REPEAT - UNTIL

REPEAT **Công việc** UMTIL **Điều kiện**

Từ ngữ pháp của cấu trúc **REPEAT - UNTIL** ta thấy: Công việc được thực hiện đầu tiên. Điều đó có nghĩa là công việc được thực hiện ít nhất một lần. Điều kiện được kiểm tra sau đó. Công việc được lặp đi lặp lại cho tới Điều kiện được thoả mãn. Điều này trong hợp ngữ hoàn toàn tương đương với việc dùng lệnh CMP để kiểm tra Điềukiện và sau đó dùng lệnh nhảy có điều kiện để thoát khỏi vòng lặp.

Bài tập III-7

Đọc ký tự từ bàn phím cho tới khi gặp '$' thì thôi.

Giải

Ví dụ này chỉ làm một phần công việc của ví dụ trước. Tại đây ta chỉ phải đọc các ký tự đọc được.

Ta có thể tực hiện công việc trên bằng mẩu chương trình sau:

```
        MOV Ah, 1    ; hàm đọc ký tự bàn phím.
TIEP:   INT 21H      ; đọc 1 ký tự.
        CMP AL, '$'  ; đọc được đôla ?
RA:                  ; lối ra của cấu trúc.
```

III.4 Giới thiệu một số chương trình cụ thể

Trong phần này ta sẽ xét một số chương trình cho các ứng dụng cụ thể, thông qua các ví dụ này ta có thể học được các lệnh, cách lập chương trình cùng với cách tổ chức dữ liệu để giải quyết các bài toán cụ thể. Một số chương trình liên quan đến các vấn đề khác chưa được đề cập đến từ trước đến nay có thể được nêu ra ở những chương tương ứng sau chương này.

Trước khi giới thiệu các ví dụ ta hệ thống lại một vài hàm của các loại ngắt có trong máy IBM PC với hệ điều hành MS DOS hay chưa được dùng trong các ví dụ đã nêu trước đây và sau này.

Bảng III-2. Một số dịch vụ ngắt DOS

Ngắt INT 20H dành riêng để kêt thúc chương trình loại. COM
Hàm 1 của ngắt INT 21H: đọc 1 ký tự từ bàn phím
Vào: AH = 1
Ra: AL = mã ASCH của ký tự cần hiện thị
Al = 0 khi ký tự gõ vào là từ các phím chức năng
Hàm 2 của ngắt INT 21H: hiện 1 ký tự lên màn hình
Vào: AH = 2
DL = mã ASCH của ký tự cần hiện thị.
Hàm 9 của ngắt INT 21H: hiện chuỗi ký tự với $ ở cuối lên màn hình
Vào: AH = 9
DX = địa chỉ lệch của chuỗi ký tự cần hiện thị.
Hàm 4CH của ngắt INT 21H: kết thúc chương trình loại. EXE
Vào: AH = 4CH

III.4.1 Ví dụ 1

Trong phần đầu của chương trình hợp ngữ ta có giới thiệu một chương trình hiện lời chào băng tiếng Anh "Hello". Bây giờ ta phải thêm một lời chào bằng tiếng Việt không dấu "Chao ban" nằm cách lời chào "Hello" trước đây một số dòng nhất định nào đó.

Giải

Ta cũng vẫn sử dụng phương pháp đã được dùng ở chương trình mẫu trước đây để hiện thị lời chào 'tây', hiện các dòng giãn cách và hiện lời chào 'ta'. Trong ví dụ này ta cũng bỏ bớt đi các dòng cách ở đầu và cuối để chương trình đỡ rườm rà.

```
. Model Small
. Stack 100
. Data
        CRLF            DB      13, 10, '$'
        Chao tay        DB      'hello!$'
        ChaoTa          DB      'Chao ban!$'
. Code
MAIN Proc
        MOV AX, @ Data  ; khởi đầu thanh ghi DS
        MOV DS, AX
; hiện thị lời chào dùng hàm 9 của INT 21H
        MOV AH, 9
        LEA DX, ChaoTay
        INT 21H
; cách 5 dòng dùng hàm 9 của INT 21H
        LEA DX, CELF
        MOV CX, 6       ;CX chứa số dòng cách +1
LAP:    INT 21H
        LOOP LAP
; hiện thị lời chào dùng hàm 9 của INT 21H
        LEA DX, ChaoTa
        INT 21H
; trở về DOS dùng hàm 4 CH của INT 21H
        MOV AH, 4CH
        INT 21H
MAIN Endp
        END MAIN
```

Trong chương trình trên ta đã dùng thanh ghi CX để chứa số dòng phải giãn cách. Với cách làm này mỗi khi muốn thay đổi số dòng dãn cách giữa 2 lời chào ta và lời chào tây, ta phải gắn giá trị khác cho thanh ghi CX.

III.4.2 Ví dụ 2

Trên cơ sở ví dụ trước, ta phải viết chương trình sao cho số dòng giãn cách có thể thay đổi được ngay trong khi chạy chương trình.

Giải

Muốn có số dòng cách thay đổi được theo ý muốn giữa 2 lời chào ta và tây khi chạy chương trình mà không phải thay giá trị mới cho thanh ghi CX ngay trong chương trình như ở ví dụ trước, ta cần dùng thêm 1 biến mới để chứa số dòng cách và viết chương trình sao cho mỗi khi cho chạy thì chương trình có thêm phần đối thoại để người sử dụng có thể thay đổi giá trị của số dòng giãn cách đó.

Sau đây là chương trình thực hiện công việc trên:

```
. Model Small
. Stack 100
. Data
        CRLF        DB    13, 10, '$'
        ChaoTay     DB    'Hello!S'
        ChaoTa      DB    'Chao ban!S'
        Thongbao    DB    'go vao so dong cach:S'
        SoCRLF      DB    ?
. Code
MAIN Proc
        MOV AX, @Data      ; khởi đầu thanh ghi DS
        MOV DS, AX
                ; hiện thông báo dùng hàm 9 của INT 21H
MOV AH, 9
LEA DX, Thongbao
INT 21H
        ; đọc số dòng cách dùng hàm 1 của INT 21H
MOV AH, 1
INT 21H                    ; đọc số dòng cách
AND AL, OFH                ; đổi ra hệ hai
MOV SoCRLE, AL             ; cất đi
        ; cách 1 dòng dùng hàm 9 của INT 21H
MOV AH, 9
LEA DX, CRLF
INT 21H
        ; hiển thị lời chào dùng hàm 9 của INT 21H
MOV AH, 9
LEA DX, ChaoTay
INT 21H
LEA DX, CFLF
```

```
        XOR CX, CX
        MOV CL, SoCRLE          ; CX chứa số dòng cách
LAP:    INT 21H
        LOOP LAP
            ; hiện thị lời chào dùng hàm 9 của INT 21H
        LEA DX, ChaoTa
        INT 21H
            ; trở về DOS dùng hàm 4CH của INT 21H
        MOV AH, 4CH
        INT 21H
MAIN    Endp
        END MAIN
```

Trong ví dụ trên có một điều cần chú ý là khi đọc một ký tự từ bàm phím (trong trường hợp cụ thể này thì đó là số dòng cách) ta sẽ thu được trong thanh ghi AL mã ASCII của ký tự (số) đã gõ. Để sử dụng nó trong trường hợp cụ thể như một giá trị số và cất nó tại biến SoCRLF, ta phải biến đổi mã ASCII này thành hệ số hai. Để đối mã ASCII của một số ra trị số hoặc ngược lại ta cần nhớ rằng giữa giá trị số và mã ASCII của số đó có một khoảng cách là 30H. Ví dụ số 9 có mã ASCII là 39H (có thể được viết là "9"), tương tự số 0 có mã ASCII là 30H (có thể được viết là "0"). Như vậy việc biến đổi mã ASCII (giả thiết đã có sẵn trong AL) ra giá trị số có thể thực hiện được bằng một trong các lệnh sau:

+ SUB AL, 30H
+ AND AL, 0FH

Tương tự như vậy, việc biến đổi ngược lại từ số hệ hai (thường giả thiết đã có sẵn trong thanh ghi DL) ra mã ASCII (để đưa ra hiện lên mãn hình) có thể làm được bằng một trong các lệnh sau:

+ ADD DL, 30H
+ OR DL, 30H

III.4.3 Ví dụ 3

Đọc từ bàn phím một số hệ hai (dài nhất là 16 bít), kết quả đọc được để tại thanh ghi BX. Sau đó hiện nội dung thanh ghi BX ra màn hình.

Giải

Công việc của bài này thực chất gồm hai phần, một phần đầu ta phải đọc được số hệ hai và cất nó tại BX, trong phần tiếp theo ta phải đưa được nội dung của thanh ghi BX ra màn hình.

Sau đây là chương trình thực hiện công việc trên:

```
. Model Small
. Stack 100
. Data
```

```
        TBao DB 'Go vao 1 so he hai (max 16 bít, '
             DB 'CR de thoi):$'
. Code
MAIN proc
        MOV AX, @ Data
        MOV DS, AX
        MOV AH, 9          ; hiện thị thông báo
        LEA DX, TBao
        INT 21H
        XOR   BX, BX       ; BX chứa kết quả, lúc đầu là 0
        MOV AH, 1          ; hàm đọc 1 số từ bàn phím
TIEP: INT 21H
        CMP AL, 13         ; CR?
        JF THOIDOC ; đúng, thôi đọc
        AND AL, OFH        ; không, đổi mã ASCII ra số
        SHL BX, 1          ; dịch trái BX 1 bít để lấy chỗ
        OR BL, AL          ; chèn bít vừa đọc vào kết quả
        JMP TIEP           ; đọc tiếp một ký tự
THOIDOC:MOV CX, 16         ; CX chứa số bít của BX
        MOV AH, 2          ; hàm hiện ký tự
        HIEN:XOR DL, DL    ; xoá DL để chuẩn bị đổi
        ROL BX, 1          ; đưa bít MSB của BX sang CF
        ADC DL, 30H; đổi giá trị bít đó ra ASCII
        INT 21H            ; hiển thị 1 bít của BX
        LOOP HIEN          ; lặp lại cho đến hết
        MOV AH, 4CH        ; trở về DOS
        INT 21H
MAIN Endp
        END MAIN
```

Chương trình hợp ngữ cho công việc đã nêu được hình thành từ 2 phần, một phần với chức năng đọc và một phần với chức năng hiện thị.

Thuật toán cho phần đọc: đọc một ký tự số, chuyển mã ASCII ra số rồi chèn số đọc được vào BX theo thứ tự từ phải qua trái, lặp lại công việc trên các số khác.

Thuật toán cho phần hiện thị ngược lại so với phần đọc: lấy ra 1 bít của số đó trong BX theo thứ tự từ trái qua phải, đổi số đó ra mã ASCII rồi cho hiện thị nó ra màn hình, lặp lại công việc trên cho các số khác.

Các thuật toán của 2 phần trên về cơ bản có thể ứng dụng được cho trường hợp phải đọc và hiện thị số hệ mười sáu hoặc hệ mười.

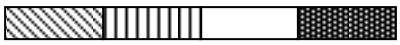
Một số nhận xét có thể rút ra khi đọc chương trình trên:

- Lệnh xóa thanh ghi BX là rất cần thiết để sau này khi gõ vào các bít của nó ta không nhất thiết phải gõ đủ 16 bít mà vẫn xác định được giá trị của thanh ghi này.
- Trong chương trình này ta đã dùng lệnh ROL để quay tròn thanh ghi BX, vì vậy sau khi quay và hiện thị tất cả 16 bít của BX ta vẫn bảo toàn được giá trị của thanh ghi BX lúc đầu. Để so sánh, nếu ở phần trên thay vì lệnh quay ROL ta dùng lệnh dịch SHL thì ta vẫn hiện thị được đúng thanh ghi BX, nhưng sau khi hiện thị xong thì quá trị nguyên thủy của thanh ghi BX, nhưng sau khi hiện thị xong thì giá trị nguyên thuỷ của thanh ghi BX bị mất do quá trình dịch gây nên.
- Trong chương trình này ta đã dùng lệnh cộng có nhớ ADC một cách rất hiệu dụng để lấy ra 1 bít của thanh ghi BX từ giá trị của cờ CF và đổi luôn được nó ra mã ASCII cần thiết cho việc hiện thị.

III.4.4 Ví dụ 4

Trong thanh ghi BX có sẵn 4 số hệ mười sáu, mỗi số được biểu diễn bằng 1 ô màu:

Hãy lập trình để biến đổi thanh ghi BX thành:



(Ví dụ: nếu như lúc đầu thanh ghi BX chứa giá trị 1234H thì sau khi biến đổi, BX sẽ chứa giá trị 3241H. v. v. . .)

Giải

Thực chất đây là kiểu bài toán cụ thể này, sau khi xem xét dạng thức của thanh ghi BX trước và sau khi biến đổi, ta thấy có thể thu được kết quả môt cách rất đơn giản bằng cách quay trái thanh ghi BX nguyên gốc đi 4 bít rồi sau đó quay tiếp thanh ghi BH đi 4 bít.

Sau đây là chương trình thực hiện công việc trên.

```
. Model Small
. Stack 100
. Code
MAIN Proc
        MOV CL, 4
        ROL BX, CL          ; quay BX đi 4 bít
        MOV CL, 4
        ROR BH, CL  ; tráo 4 bít thấp và cao của BH
        MOV AH, 4CH         ; trở về DOS
        INT 21H
MAIN Endp
```

```
        END MAIN
```

III.4.5 Ví dụ 5

Có một chuỗi ký tự thường trong bộ nhớ. Hãy tạo ra một chuỗi ký tự chữ hoa từ chuỗi trên rồi cất chuỗi đó trong bộ nhớ.

Giải:

Ví dụ này và ví dụ trước khi khác nhau chút ít trong việc xử lý các ký tự của chuỗi, vì vậy phần trên các lệnh có tính chất chuẩn bị trước và sau các thao tác với chuỗi có thể coi là như nhau. Để giải bài toán này có thể ứng dụng các lệnh LODSB và STOSB với chuỗi đã cho. Thuật toán là:

- Lấy từng ký tự của chuỗi gốc (cũ) bằng lệnh LODSB,
- Biến đổi thành chữ hoa bằng cách trừ đi 20H,
- Cất ký tự đã biến đổi vào chuỗi đích (mới) bằng lệnh STOSB.

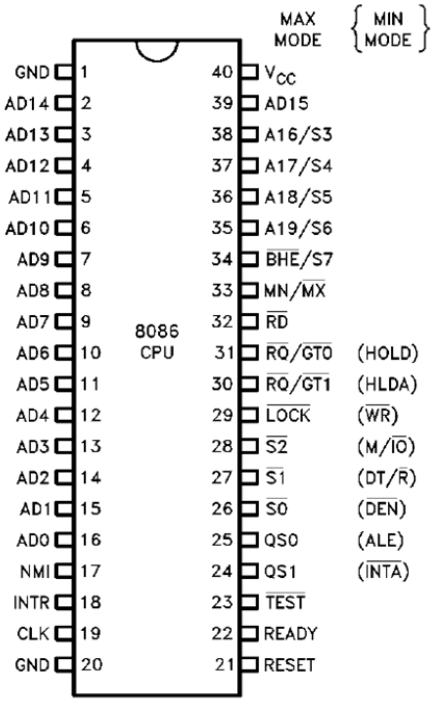
Sau đây là cách tổ chức dữ liệu và chương trình cho bài toán trên với độ dài chuỗi là 8 byte. Để minh hoạ một cách thao tác khác so với cách ở ví dụ trước trong ví dụ này là dùng cách thao tác lùi đối với chuỗi ký tự.

```
. Model Small
. Stack 100
. Data
        Str1    DB 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'
        Tbao    DB 'chuỗi đã được đổi: ', 10, 13
                DB '$'
. Code
MAIN Proc
        MOV   AX, @Data     ; khởi đầu đầu cho DS và ES
        MOV   DS, AX
        MOV   ES, AX
        LEA   SI, Str1+7    ; SI chỉ vào cuối chuỗi cũ
        LEA   DI, Str2+7    ; DI chỉ vào cuối chuỗi mới
        STD                 ; định hướng lùi
        MOV   CX, 8         ; CX chứa số byte phải đổi
LAP:    LODSB               ; lấy 1 ký tự của chuỗi cũ
        SUB AL, 20H ; đổi thành chữ hoa
        STOSB               ; cất vào chuỗi mới
        LOOP LAP            ; làm cho đến hết
        LEA DX, Tbao        ; chuẩn bị hiện chuỗi mới
        MOV   AH, 9
        INT   21H
        MOV   AH, 4CH       ; về DOS
        INT   21H
MAIN Endp
        END MAIN
```

Chương IV. Phối ghép vi xử lý với bộ nhớ và các thiết bị vào/ra

IV.1 Các tín hiệu của vi xử lý và các mạch phụ trợ

IV.1.1 Các tín hiệu của 8086/8088



Hình IV-1. Tín hiệu 8086

Hình vẽ trên cho chúng ta thấy tính hiệu của 8086. Các tín hiệu của 8086 và 8088 chỉ khác ở số lượng kênh dữ liệu. 8086 có 16 đường dữ liệu trong khi 8088 chỉ có 8 đường dữ liệu. Dưới đây trình bày ý nghĩa các tín hiệu của 8086,

Chức năng các tín hiệu tại các chân cụ thể như sau:

- ADO – AD15 [I;O: tín hiệu vào và ra]: Các chân dồn kênh cho các tín hiệu buýt dữ liệu và buýt địa chỉ. Xung ALE sẽ báo cho mạnh ngoài biết khi nào trên các đường đó có tín hiệu dữ liệu (ALE = 0) hoặc địa chỉ (ALE = 1). Các chân này ở trạng thái trở kháng cao khi μP chấp nhận treo.
- A16/S3, A17/S4, A18/S5, A19/S6 [O]: Các chân dồn kênh của địa chỉ phần cao và trạng thái. Địa chỉ A16 - A19 được truyền trên các chân đó khi ALE = 1 còn khi AEL = 0 thì trên các chân đó có các tín hiệu trạng thái S3 - S6. Các chân này ở trạng thái trở kháng cao khi μP chấp nhận treo. Việc kết hợp S_3 và S_4 để biểu diễn truy nhập các thanh ghi như trong bảng dưới đây.

Bảng IV-1. Các bít trạng thái và việc truy nhập các thanh ghi đoạn.

S4	S3	Truy nhập đến
0	0	Đoạn dữ liệu phụ
0	1	Đoạn ngăn xếp
1	0	Đoạn mã hoặc không đoạn nào
1	1	Đoạn dữ liệu

- $\overline{RD}$ [O]: Xung cho phép đọc. Khi $\overline{RD}$ = 0 thì buýt dữ liệu sẵn sàng nhận số liệu từi bộ nhớ hoặc thiết bị ngoại vi. Chân $\overline{RD}$ ở trạng thái trở kháng cao khi μP chấp nhận treo.
- READY [I]: Tín hiệu báo cho CPU biết tình trạng sẵn sàng của thiết bị ngoại vi hay bộ nhớ. Khi READY = 1 thì CPU thực ghi/đọc mà không cần chèn thêm các chu kỳ đợi. Ngược lại khi thiết bị ngoại vi hay bộ nhớ có tốc độ hoạt động chậm, chúng có thể đưa tín hiệu READY = 0 để báo cho CPU biết. Lúc này CPU tự kéo dài thời gian thực hiện lệnh ghi/đọc bằng cách chèn thêm các chu kỳ đợi.
- INTR [I]: Tín hiệu yêu cầu ngắt che được. Khi có yêu cầu ngắt mà cờ cho phép ngắt IF = 1 thì CPU kết thúc lệnh đang làm dở, sau đó nó đi vào chu kỳ chấp nhận ngắt và đưa ra bên ngoài tín hiệu INTA = 0.
- $\overline{TEST}$ [I]: Tín hiệu tại chân này được kiểm tra bởi lệnh WAIT. Khi CPU thực hiện lệnh WAIT mà lúc đó tín hiệu $\overline{TEST}$ =1, nó sẽ chờ cho đến khi tín hiệu $\overline{TEST}$ = 0 thì mới thực hiện lệnh tiếp theo.
- NMI [I]: Tín hiệu yêu cầu ngắt không che được. Tín hiệu này không bị khống chế bởi cờ IF và nó sẽ được CPU nhận biết bằng các tác động của sườn lên của xung yêu cầu ngắt. Nhận được yêu cầu này CPU kết thúc lệnh đang làm dở, sau đó nó chuyển sang thực hiện chương trình phục vụ ngắt kiểu INT2.
- RESET [I]: tín hiệu khởi động lại 8086/8088. khi RESET = 1 kéo dài ít nhất trong thời gian 4 chu kỳ đồng hồ thì 8086/8088 bị buộc phải khởi động lại: nó xoá các thanh ghi DS, ES, SS, IP và FR về 0 và bắt đầu thực hiện chương trình tại địa chỉ CS:IP=FFFF:0000H (chú ý cờ IF← 0 để cấm các yêu cầu ngắt khác tác động vào CPU và cờ TF←0 để bộ vi xử lý không -bị đặt trong chế độ chạy từng lệnh).
- CLK [I]: Tín hiệu đồng hồ (xung nhịp). Xung nhịp có độ rỗng là 77% và cung cấp nhịp làm việc cho CPU.
- Vcc [I]: Chân nguồn. Tại đây CPU được cung cấp 5V±10%. 340mA
- GND [O]: Hai chân nguồn để nối với điểm 0V của nguồn nuôi.
- MN/MX [I]: Chân điều khiển hoạt động của CPU theo chế độ MIN/MAX. Do 8086/8088 có thể làm việc ở 2 chế độ khác nhau nên có một số chân tín hiệu phụ thuộc vào các chế độ đó.
  a) Chế độ MIN (Chân MN/MX cần được nối thẳng vào +5V mà không qua điện trở)
     - Trong chế độ MIN tất cả các tín hiệu điều khiển liên quan đến các thiết bị ngoại vi truyền thống và bộ nhớ giống như trong hệ 8085 đều có sẵn trong

8086/8088. Vì vậy việc phối ghép với các thiết bị đó sẽ rất dễ dàng và chính vì tận dụng được các phối ghép ngoại vi sẵn nên có thể giảm giá thành hệ thống.

- IO/ $\overline{M}$ [O]: Tín hiệu này phân biệt trong thời điểm đã định phần tử nào trong các thiết bị vào/ra (IO) hoặc bộ nhớ (M) được chọn làm việc với CPU. Trên buýt địa chỉ lúc đó sẽ có các địa chỉ tương ứng của các thiết bị đó. Chân này ở trạng thái trở kháng cao khi μP chấp nhận treo.
- $\overline{WR}$ [O]: Xung cho phép ghi. Khi CPU đưa ra $\overline{WR}$ =0 thì trên buýt dữ liệu các dữ liệu đã ổn định và chúng sẽ được ghi vào bộ nhớ hoặc thiết bị ngoại vi tại thời điểm $\overline{WR}$ = 1. Chân $\overline{WR}$ ở trạng thái trở kháng cao khi μP chấp nhận treo.
- INTA [O]: Tín hiệu báo cho các mạch bên ngoài biết CPU chấp nhận yêu cầu ngắt INTR. Lúc này CPU đưa ra INTA = 0 để báo là nó đang chờ mạch ngoài đưa vào số hiệu ngắt (kiểu ngắt) trên buýt dữ liệu.
- ALE [O]: Xung cho phép chốt địa chỉ. Khi ALE = 1 có nghĩa là trên buýt dồn kênh AD có các địa chỉ của thiết bị vào/ra hay của ô nhớ. ALE không bao giờ bị thả nối (trong trạng thái trở kháng cao) khi CPU bị treo thì ALE = 0.
- DT/ $\overline{R}$ [O]: Tín hiệu điều khiển các đệm 2 chiều của buýt dữ liệu để chọn chiều chuyển của vận dữ liệu trên buýt D. Chân này ở trạng thái trở kháng cao khi μP chấp nhận treo.
- $\overline{DEN}$ [O]: Tín hiệu báo cho bên ngoài biết là lúc này trên buýt dồn kênh AD có dữ liệu ổn định. Chân này ở trạng thái trở kháng cao khi μP chấp nhận treo.
- HOLD [I]: Tín hiệu yêu cầu treo CPU để mạch ngoài thực hiện việc trao đổi dữ liệu với bộ nhớ bằng cách truy nhập trực tiếp. Khi HOLD = 1. CPU 8086 sẽ tự tách ra hệ thống bằng cách treo tất cả các buýt A, buýt D, buýt C của nó (các buýt ở trạng thái trở kháng cao) để bộ điều khiển DMA (DMA contrroller, DMAC) có thể lấy được quyền điều khiển hệ thống để làm các công việc trao đổi dữ liệu.

Bảng IV-2. Các chu kỳ của buýt qua các tín hiệu $\overline{SS0}$, IO/ $\overline{M}$, DT/ $\overline{R}$

IO/ $\overline{M}$	DT/ $\overline{R}$	$\overline{SS0}$	Chu kỳ điều khiển của buýt
0	0	0	Đọc mã lệnh
0	0	1	Đọc bộ nhớ
0	1	0	Ghi bộ nhớ
0	1	1	Buýt rỗĩ (nghĩ)
1	0	0	Chấp nhận yêu cầu ngắt
1	0	1	Đọc thiết bị ngoại vi
1	1	0	Ghi thiết bị ngoại vi
1	1	1	Dừng (halt)

Bít S6 = 0 liên tục, bít S5 phản kánh giá trị bít IF của thanh ghi cờ. Hai bít S3 và S4 phối hợp với nhau để chỉ ra việc truy nhập các thanh ghi đoạn

- HLDA [O]: Tín hiệu báo cho bên ngoài biết yêu cầu treo CPU để dùng các buýt đã được chấp nhận, và CPU 8086/8088 đã treo các buýt A, buýt D và một số tín hiệu của buýt C.
- $\overline{SSO}$ [O]: Tín hiệu trạng thái được dùng kết hợp với IO/M và DT/$\overline{R}$ để giải mã các chu kỳ hoạt động của buýt.

b) Chế độ MAX (Chân MN/MX nối đất)

- Trong chế độ MAX một số tín hiệu điều khiển cần thiết được tạo ra trên cơ sở các tín hiệu trạng thái nhờ dùng thêm ở bên ngoài một mạch điều khiển buýt 8288. Chế độ MAX được sử dụng khi trong hệ thống có mặt bộ đồng xử lý toán học 8087.
- $\overline{S2}$ $\overline{S1}$ và $\overline{S0}$[O]: Các chân trạng thái dùng trong chế độ MAX để ghép với mạch điều khiển buýt 8288. Các tín hiệu này được 8288 dùng để tạo ra các tín hiệu điều khiển trong các chu kỳ hoạt động của buýt. Các tín hiệu điều khiển đó được chỉ ra trong bảng dưới đây.

Bảng IV-3. Các tín hiệu điều khiển của 8288.

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Chu kỳ điều khiển của buýt	Tín hiệu
0	0	0	Chấp nhận yêu cầu ngắt	INTA
0	0	1	Đọc thiết bị ngoại vi	IORC
0	1	0	Ghi thiết bị ngoại vi	IOWC, $\overline{AIOWC}$
0	1	1	Dừng (halt)	Không
1	0	0	Đọc mã lệnh	MRDC
1	0	1	Đọc bộ nhớ	MRDC
1	1	0	Ghi bộ nhớ	MWTC, $\overline{AMWC}$
1	1	1	Buýt rỗi (nghĩ)	Không

- $\overline{RQ}/\overline{GT0}$ và $\overline{RQ}/\overline{GT1}$ [I/O]: Các tín hiệu yêu cầu dùng buýt của các bộ xử lý khác hoặc thông báo chấp nhận treo của CPU để cho các bộ vi xử lý khác dùng buýt. $\overline{RQ}/\overline{GT0}$ có mức ưu tiên hơn $\overline{RQ}/\overline{GT1}$.
- $\overline{LOCK}$ [O]: Tín hiệu do CPU đưa ra để cấm các bộ xử lý khác trong hệ thống dùng buýt trong khi nó đang thi hành một lệnh nào đó đặt sau tiếp đầu LOCK.
- QS0 và QS1 [O]: Tín hiệu thông báo các trạng thái khác nhau của đệm lệnh (hàng đợi lệnh). Bảng IV-4 cho biết các trạng thái của đệm lệnh được mã hoá bằng các tín hiệu trên.

  Trong hệ vi xử lý với sự có mặt của bộ đồng hồ xử lý toán học 8087, các tín hiệu này được mạch 8087 dùng để đồng bộ quá trình hoạt động của nó với bộ vi xử lý 8086/8088.

Bảng IV-4. Các trạng thái của lệnh đệm

QS1	QS0	Trạng thái lệnh đệm
0	0	Không hoạt động
0	1	Đọc byte mã lệnh đầu tiên từ đệm lệnh
1	0	Đọc lệnh rỗng
1	1	Đọc byte tiếp theo từ đệm lệnh

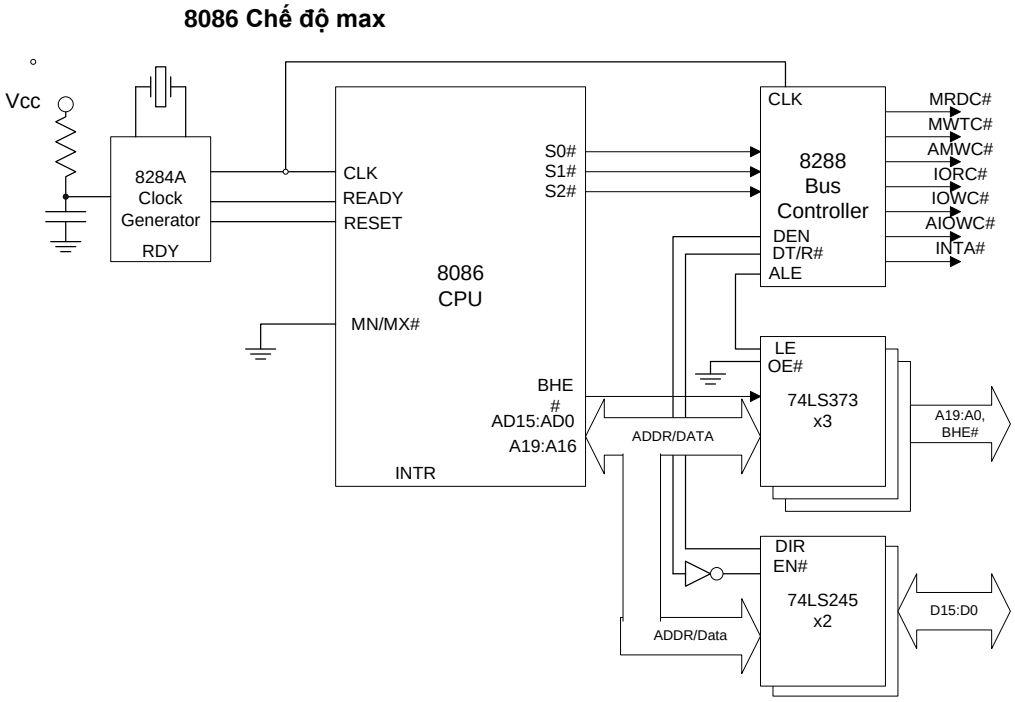
IV.1.2 Phân kênh để tách thông tin và việc đệm cho các buýt

Để giảm bớt khó khăn về mặt công nghệ do việc phải chế tạo nhiều chân cho các tín hiệu của vi mạch CPU, người ta đã tìm cách hạn chế số chân của vi mạch bằng cách dồn kênh nhiều tín hiệu trên cùng một chân. Ví dụ các chân $AD_0 – AD_{16}$ của 8086 được dồn kênh để có thể đưa ra bên ngoài các thông tin về địa chỉ và dữ liệu. Khi nhận được các tín hiệu đó ở bên ngoài vi mạch, ta phải tiến hành tách các tín hiệu để tái tạo lại các tín hiệu gốc cho các buýt độc lập (buýt địa chỉ và buýt dữ liệu). Việc này thực hiện bằng cách sử dụng các vi mạch chức năng thích hợp ở bên ngoài (thông thường thì đó là các mạch chốt). Ta cũng phải làm tương tự như vậy đối với các chân dồn địa chỉ/trạng thái. Để hỗ trợ cho việc tách thông tin này, CPU đưa ra thêm xung ALE sao cho khi ALE ở mức cao sẽ có tác dụng báo cho bên ngoài biết lúc này thông tin về địa chỉ tại các chân dồn kênh có giá trị. Xung ALE được dùng để mở các mạch chốt và tách được các thông tin về địa chỉ bị dồn kênh.

Muốn nâng cao tải của các buýt để đảm nhận việc nuôi các mạch bên ngoài. Các tín hiệu ra và vào CPU cần phải được khuếch đại thông qua các mạch đệm một chiều hoặc hai chiều với các đầu ra thường hoặc đầu ra 3 trạng thái.

Hình IV-3 cho thấy một ví dụ đơn giản các tổ chức việc tách tín hiệu địa chỉ từ các tín hiệu dồn kênh chỉ/dữ liệu hoặc địa chỉ/điều khiển bằng các mạch chốt 74LS373 và việc sử dụng các bộ khuếch đại đệm 74LS244 và 74LS245 cho các tín hiệu của bộ vi xử lý 8086/8088 làm việc ở chế độ MAX.

Hình IV-3 cho thấy sự có mặt của các mạch phụ trợ như: bộ điều khiển buýt 8288, bộ tạo ra xung đồng hồ 8284. Ngoài ra còn có thể có các mạch phụ trợ khác như: bộ phối ghép ngoại vi song song 8255, bộ điều khiển truy nhập trực tiếp vào bộ nhớ 8237 và bộ điều khiển ngắt ưu tiên 8259.



Hình IV-3. Buýt hệ thống 8086 có đệm

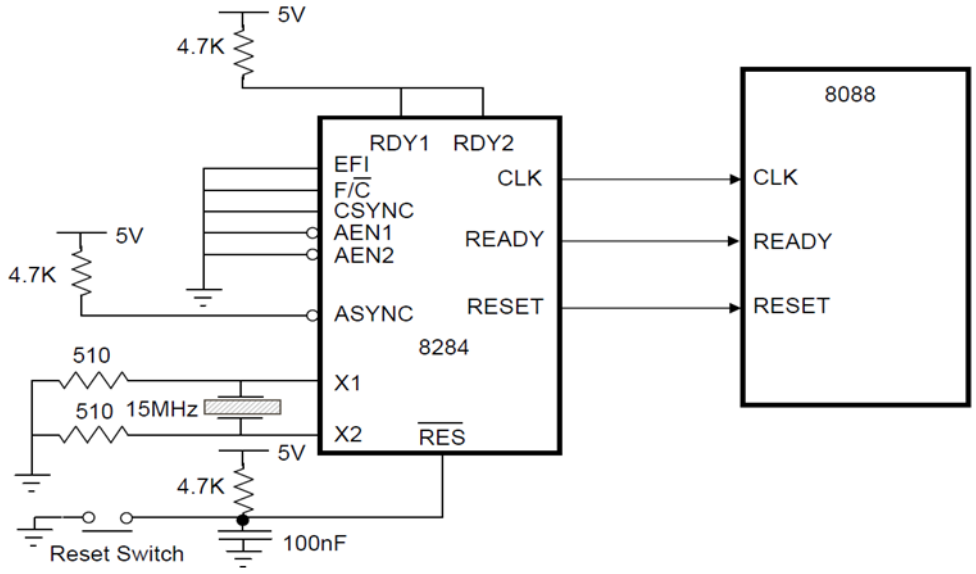
IV.1.3 Mạch tạo xung nhịp 8284.

Cho dù làm việc trong chế độ MIN hay MAX, CPU 8086/8088 luôn cần xung nhịp (xung đồng hồ) từ mạch tạo xung nhip 8284. Mạch tạo xung nhịp không những cung cấp xung nhịp với tần số thích hợp cho toàn hệ mà nó còn có ảnh hưởng tới việc đồng bộ tín hiệu RESET và tín hiệu READY của CPU (Hình IV-4). Ý nghĩa của các tín hiệu

- $\overline{AEN1}$, $\overline{AEN2}$: Tín hiệu cho phép chọn đầu vào tương ứng RDY1, RDY2 làm tín hiệu báo tình trạng sẵn sàng của bộ nhớ hoặc thiết bị ngoại vi.
- RDY1, RDY2: cùng với $\overline{AEN1}$, $\overline{AEN2}$ dùng để tạo ra các chu kỳ đợi ở CPU.
- $\overline{ASYNC}$: Chọn đồng bộ hai tầng hoặc đồng bộ một tầng cho tín hiệu RDY1, RDY2. Trong chế độ đồng bộ một tầng ($\overline{ASYNC}$ = 1) tín hiệu RDY có ảnh hưởng đến tín hiệu READY tới sườn xuống của xung đồng hồ tiếp theo. Còn trong chế độ đồng bộ hai tầng ($\overline{ASYNC}$ = 0) tín hiệu RDY chỉ có ảnh hưởng đến tín hiệu READY khi có sườn xuống của xung đồng hồ tiếp theo.
- READY: Nối đến đầu READY của CPU. Tín hiệu này được đồng bộ với các tín hiệu RDY1, RDY2.
- X1, X2: Nối với hai chân của thạch anh với tần số f_x, thạch anh này là một bộ phận của một mạch dao động bên trong 8284 có nhiệm vụ tạo xung chuẩn dùng làm tín hiệu đồng hồ cho toàn hệ thống.

- F/$\overline{C}$: Dùng để chọn nguồn tín hiệu chuẩn cho 8284. Khi chân này ở mức cao thì xung đồng hồ bên ngoài sẽ được dùng làm xung nhịp cho 8284, ngược lại thì xung đồng hồ của mạch dao động bên trong dùng thạch anh sẽ được chọn để làm xung nhịp.
- EFI: lối vào cho xung từ bộ dao động ngoại.
- CLK: Xung nhịp $f_{CLK}=f_x/3$với độ rỗng 77% nối đến chân của CLK của 8086/8088.
- PCLK: Xung nhịp $f_{CLK}=f_x/6$với độ rỗng 50% dành cho thiết bị ngoại vi.
- OSC: Xung nhịp đã được khuếch đại có tần số bằng f_x của bộ dao động.
- $\overline{RES}$: Chân khởi động, nối với mạch RC để 8284 để tự khởi động khi bật nguồn.
- RESET: Nối vào RESETcủa 8086/8088 và là tín hiệu khởi động lại cho toàn hệ
- CSYNC: Lối vào cho xung đồng bộ chung khi trong hệ thống có các 8284 dùng dao động ngoài tại chân này (Hình IV-4)

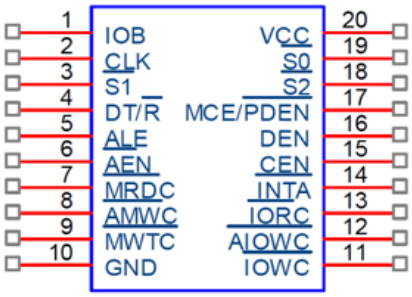
Hình IV-4 biểu diễn các đường nối tín hiệu chính của 8086/8088 và 8284. Mạch 8284 nhận được xung khởi động từ bên ngoài thông qua mạch RC khi có nguồn hoặc xung khởi động lại khi bấm công tắc K. Từ xung này 8284 có nhiệm vụ đưa ra xung khởi động đồng bộ cho CPU cùng với tất cả các thành phần khác của hệ thống.



Hình IV-4. Nối 8284 với 8086/8088

IV.1.4 Mạch điều khiển buýt 8288

Như đã giới thiệu ở phần trước, vi mạch 8288 là mạch điều khiển buýt, nó lấy một số tín hiệu điều khiển của CPU và cung cấp tất cả các tín hiệu điều khiển cần thiết cho hệ vi xử lý khi CPU 8086 làm việc ở chế độ MAX. Sơ đồ chân và các tín hiệu của 8288.



Hình IV-5. Bộ điều khiển buýt 8288

Các tín hiệu chính của 8288 bao gồm:

- $\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$ [I, I, I]: là các tín hiệu trạng thái lấy thẳng từ CPU. Tuỳ theo các tín hiệu này mà mạch 8288 sẽ tạo ra các tín hiệu điều khiển khác nhau tại các chân ra của nó để điều khiển hoạt động của các thiết bị nối với CPU. Bảng 5. 3 mô tả các tín hiệu vào và ra đó.
- CLK [I]: Đây là đầu vào nối với xung đồng hồ hệ thống (từ mạch 8284) và dùng để đồng bộ toàn bộ các xung điều khiển đi ra từ mạch 8288.
- CEN [I]: Là tín hiệu đầu vào để cho phép đưa ra tín hiệu DEN và các tín hiệu điều khiển khác của 8288.
- IOB [I]: tín hiệu để điều khiển mạch 8288 làm việc ở các chế độ buýt khác nhau. Khi IOB = 1 8288 làm việc ở chế độ buýt vào/ra, khi IOB = 0 mạch 8288 làm việc ở chế độ buýt hệ thống (như trong các máy IBM PC).
- $\overline{MRDC}$ [O]: tín hiệu điều khiển đọc bộ nhớ. Nó kích hoạt bộ nhớ đưa dữ liệu ra buýt.
- $\overline{MWTC}$ [O] $\overline{AMWC}$ [O]: là các tín hiệu điều khiển ghi bộ nhớ hoặc ghi bộ nhớ kéo dài.

  Đó thực chất là các tín hiệu giống như $\overline{MEMW}$, nhưng $\overline{AMWC}$ (advanced memory write command) hoạt động sớm lên một chút để tạo ra khả năng cho các bộ nhớ chậm có thêm được thời gian ghi.
- $\overline{IORC}$ [O]: tín hiệu điều khiển đọc thiết bị ngoại vi. Nó kích hoạt các thiết bị được chọn để các thiết bị này đưa dữ liệu ra buýt.
- $\overline{IOWC}$ [O] $\overline{AIOWC}$ [O]: là các tín hiệu điều khiển đọc thiết bị ngoại vi hoặc đọc thiết bị ngoại vi kéo dài. Đó thực chất là các tín hiệu giống như $\overline{IOW}$, nhưng $\overline{AIOWC}$ (advanced I/O write command) hoạt động sớm lên một chút để tạo ra khả năng cho các bộ nhớ chậm có thêm được thời gian ghi.
- $\overline{INTA}$ [O]: là đầu ra để thông báo là CPU chấp nhận yêu cầu ngắt của thiết bị ngoại vi và lúc này các thiết bị ngoại vi phải đưa ra số hiệu ngắt ra buýt để CPU đọc.
- DT/$\overline{R}$ [O]: là tín hiệu để điều khiển hướng đi của dữ liệu trong hệ vào hay ra so với CPU (DT/$\overline{R}$ = 0: CPU đọc dữ liệu, DT/$\overline{R}$ = 1 CPU ghi dữ liệu).

Trong các máy IBM PC thì tín hiệu này được nối đến các chân DIR của mạch đệm 2 chiều 74LS245 để điều khiển dữ liệu đi từ CPU đến buýt hệ thống khi ghi hoặc ngược lại, từ buýt hệ thống đến CPU khi đọc.

- DEN [O]: đây là tín hiệu để điều khiển buýt dữ liệu trở thành buýt cục bộ hay buýt hệ thống.
  Trong các máy IBM PC thì tín hiệu này được sử dụng cùng với tín hiệu của mạch điều khiển ngắt PIC 8259 để tạo ra tín hiệu điều khiển cực G của mạch đệm 2 chiều 74LS245.
- MCE/ $\overline{PDEN}$ [O]: đây là tín hiệu dùng để định chế độ làm việc cho mạch điều khiển ngắt PIC 8259 để nó làm việc ở chế độ chủ.
- ALE [O]: đây là tín hiệu cho phép chốt địa chỉ tại các chân dồn kênh địa chỉ - dữ liệu AD0 - AD7, tín hiệu này thường được nối với chân G của mạch 74LS373 để điều khiển mạch này chốt lấy địa chỉ.

IV.1.5 Biểu đồ thời gian của các lệnh ghi/đọc

Hình IV-6 và Hình IV-7 là các biểu đồ thời gian đã được đơn giản hoá của các tín hiệu cơ bản trong CPU 8086/8088 cho các lệnh ghi/đọc bộ nhớ hoặc thiết bị ngoại vi.

Một chu kỳ ghi/đọc bình thường (còn gọi là chu kỳ buýt) của CPU kéo dài 4 chu kỳ đồng hồ. Các chu kỳ đồng hồ được đánh dấu là T1, T2, T3 và T4. Nếu CPU làm việc với tần số đồng hồ 5MHz thì một chu kỳ đồng hồ kéo dài T=200ns và một chu kỳ buýt kéo dài 4*T=800ns.

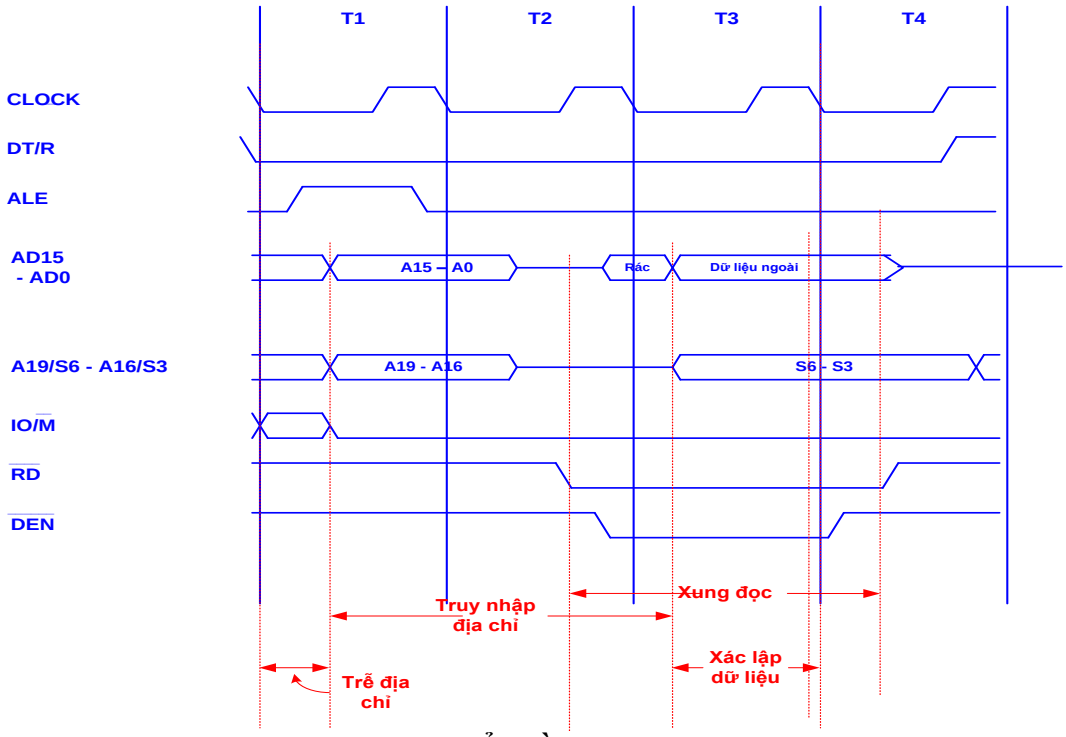
Chúng ta mô tả tóm tắt các hiện tượng xảy ra trong một chu kỳ T nói trên.

Chu kỳ T1:

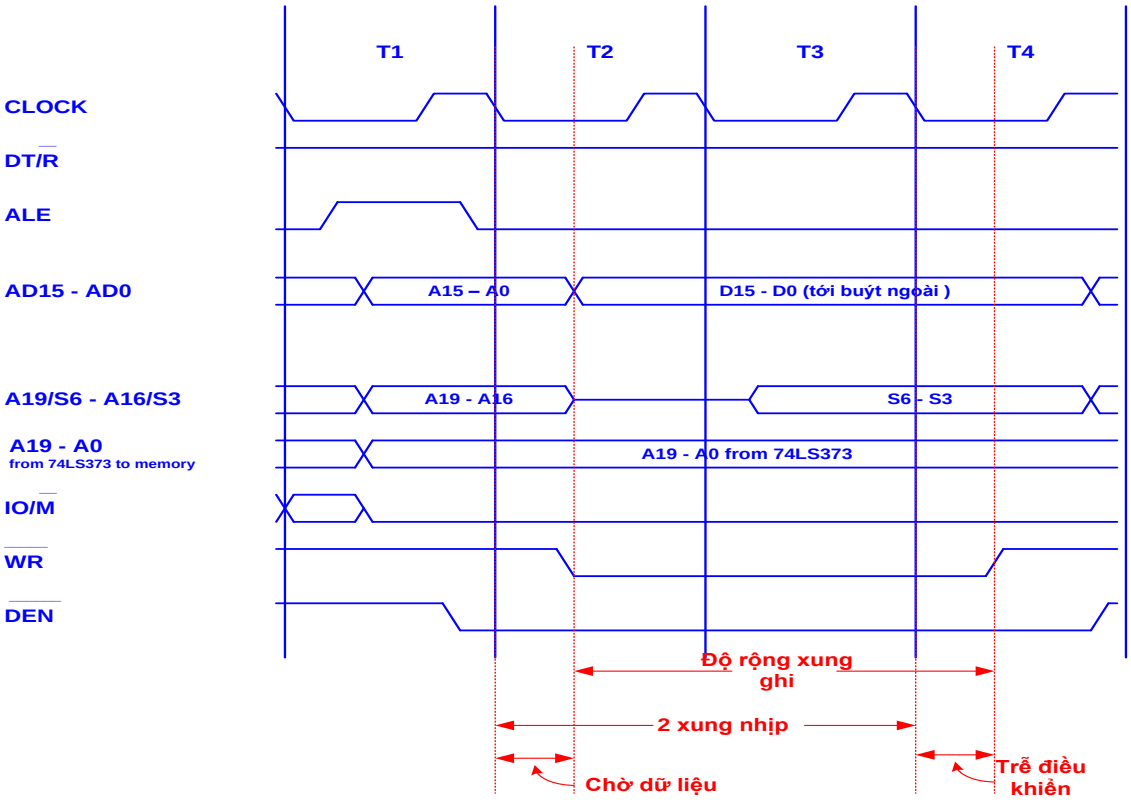
Trong chu kỳ này địa chỉ của bộ nhớ hay thiết bị ngoại vi được đưa ra trên các đường địa chỉ, hoặc địa chỉ/dữ liệu và địa chỉ/ trạng thái. Các tín hiệu điều khiển ALE, DT/ $\overline{R}$,IO/ $\overline{M}$ cũng được đưa ra để giúp việc hoàn tất việc giữ thông tin địa chỉ này.

Chu kỳ T2:

Trong chu này CPU đưa ra các tín hiệu điều khiển $\overline{RD}$ hoặc $\overline{WR}$, $\overline{DEN}$ và tín hiệu dữ liệu trên D0 - D7 nếu là lệnh ghi. $\overline{DEN}$ thường dùng để mở các bộ đệm của buýt dữ liệu nếu như chúng được dùng trong hệ. Tại cuối kỳ T2 (và giữa mỗi chu kỳ T của T_w, nếu có) CPU lấy mẫu tín hiệu READY để xử lý trong chu kỳ tiếp theo khi nó phải làm việc với bộ nhớ hoặc thiết bị ngoại vi chậm.



Hình IV-6. Biểu đồ đọc đơn giản hóa



Hình IV-7. Biểu đồ ghi đơn giản hóa

Chu kỳ T3:

Trong chu kỳ này CPU dành thời giờ cho bộ nhớ hay thiết bị ngoại vi khi nhập dữ liệu. Nếu là chu kỳ đọc dữ liệu thì tại cuối T3 CPU sẽ lấy mẫu tín hiệu của buýt dữ liệu.

Nếu tại cuối chu kỳ đồng hồ T2 (hoặc giữa mỗi chu kỳ T của T_w) mà CPU phát hiện ra tín hiệu READY=0 (do bộ nhớ hay thiết bị ngoại vi đưa đến) thì CPU tự xen vào sau T3 một vài chu kỳ T để tạo chu kỳ đợi $T_w = n*T$ nhằm kéo dài thời gian thực hiện lệnh, tạo điều kiện cho bộ nhớ hoặc thiết bị ngoại vi có đủ thời gian hoàn tất việc ghi/đọc dữ liệu.

Chu kỳ T4:

Trong chu kỳ này các tín hiệu trên buýt được đưa về trạng thái bị động để chuẩn bị cho chu kỳ buýt mới. Tín hiệu $\overline{WR}$ trong khi chuyển trạng thái từ 0 lên 1 sẽ kích hoạt động quá trình đưa vào bộ nhớ hay thiết bị ngoại vi.

Trên các biểu đồ đọc ghi cũng biểu diễn các thông số quan trọng về mặt thời gian liên quan đến tốc độ hoạt động tối thiểu cần thiết của các bộ nhớ hoặc thiết bị ngoại vi nếu chúng muốn làm việc với CPU 5MHz.

Trong biểu đồ thời gian đọc (Hình IV-6) ta thấy việc truy nhập bộ nhớ kéo dài trong khoảng thời gian từ T1 - T3 (gần 3 chu kỳ đồng hồ 3*T = 600 ms). Trong tổng số thời gian này phải tính đến thời gian trễ khi chuyển địa chỉ $t_{trễ\ địa\ chỉ}$ = 110ns, thời gian giữ của dữ liệu khi đọc $t_{giữR}$ = 30 ns và thời gian trễ do việc truyền tín hiệu qua các mạch đệm nhiều nhất là $t_{trễ\ đệm}$ = 40ns. Như vậy các bộ nhớ nối với 8086/8088 - 5MHz cần phải có thời gian truy nhập nhỏ hơn:

$$3*T - t_{trễ\ địa\ chỉ} - t_{giữR} - t_{trễ\ đệm} = 600 - 110 - 30 - 40 = 420ns.$$

Mặt khác với CPU 8086/8088 5MHz thì độ rộng xung đọc là T_{RD} = 325ns, đó là thời gian đủ dài để cho bộ nhớ với thời gian truy nhập cỡ 420ns làm việc.

Trong biểu đồ thời gian ghi (Hình IV-7) ta thấy phải có một thời gian giữ dữ liệu tối thiểu để ghi $t_{giữW}$ = 88ns sau khi $\overline{WR}$ đột biến từ 0 lên 1. trong thực tế thời gian này gần như bằng 0 đối với bộ nhớ thông dụng. Độ dài của xung ghi đối với CPU 8086/8088 - 5MHz là t_{WR} = 340ns cũng là phù hợp với các bộ nhớ với thời gian truy nhập cỡ 450ns.

IV.2 Phối ghép vi xử lý với bộ nhớ

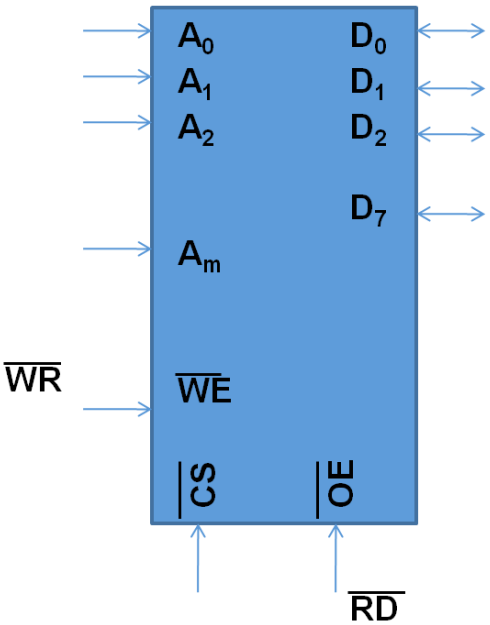
IV.2.1 Giới thiệu bộ nhớ

Các bộ nhớ bán dẫn thường dùng với bộ vi xử lý bao gồm:

- Bộ nhớ cố định ROM (Read Only Memory, bộ nhớ có nội dung ghi sẵn chỉ để đọc ra). Thông tin ghi trong mạch không bị mất khi mất nguồn điện nuôi cho mạch.
- Bộ nhớ bán cố định EPROM (Erraseble Programmable ROM là bộ nhớ ROM có thể lập trình được bằng xung điện và xoá được bằng tia cực tím).

– Bộ nhớ không cố định RAM (Random Access Memory, bộ nhớ truy nhập ngẫu nhiên) thông tin ghi trong mạch bị mất khi mất nguồn điện nuôi cho mạch. Trong các bộ nhớ RAM ta còn phân biệt ra loại RAM tĩnh (Static RAM hay SRAM, trong đó mỗi phần tử nhỏ là một mạch lật hay trạng thái ổn định) và loại RAM động (Dynamic RAM hay DRAM, trong đó mỗi phần tử nhớ là một tụ điện rất nhỏ được chế tạo bằng công nghệ MOS).

Một bộ nhớ thường được chế tạo nên từ nhiều vi mạch nhớ. Một vi mạch nhớ thường có dạng cấu trúc tiêu biểu như hình sau



Hình IV-8. Vi mạch nhớ khái quát

Theo sơ đồ khối này ta thấy một l vi mạnh nhớ có các nhóm tín hiệu sau:

a) Nhóm tín hiệu địa chỉ:

Các tín hiệu địa chỉ có tác dụng chọn ra một ô nhớ. Các ô nhớ có độ dài khác nhau (còn gọi là từ nhớ) tuỳ theo nhà sản xuất: 1, 4, 8, bít. Số đường tín hiệu địa chỉ có liên quan đến dung lượng của mạch nhớ. Với một mạch nhớ có m bít địa chỉ thì dung lượng của mạnh nhớ đó là 2^m từ nhớ. Ví dụ, với m = 10 ta có dung lượng mạch nhớ là 1K ô nhớ (1 kilô = 2^{10} = 1024) và với m=20 ta có dung lượng mạch nhớ là 1M ô nhớ (1 Mêga = 2^{20} = 1048576).

b) Nhóm tín hiệu dữ liệu:

Các tín hiệu dữ liệu thường là đầu ra đối với mạch ROM hoặc đầu vào/ra dữ liệu chung (hai chiều) đối với mạch RAM. Ngoài ra có mạch nhớ RAM với đầu ra và đầu vào dữ liệu riêng biệt. Các mạch nhớ thường có đầu ra dữ liệu kiểu 3 trạng thái. Số đường dây dữ liệu quyết định độ dài từ nhớ của mạch nhớ. Thông thường người ta hay nói rõ dung lượng và độ dài từ nhớ cùng một lúc. Ví dụ mạch nhớ dung lượng 1 Kx8 (tức là 1KB) hoặc 16Kx4. . .

c) Nhóm tín hiệu chọn vi mạch (chọn vỏ):

Các tín hiệu chọn vi mạch là $\overline{CS}$ (chip select) hoặc $\overline{CE}$ (Chip enable) thường được dùng để tạo ra vi mạch nhớ cụ thể để ghi/đọc. Tín hiệu chọn vi mạch ở các

mạch RAM thường la $\overline{CS}$, còn ở mạch ROM thường là $\overline{CE}$. Các tín hiệu chọn vi mạch thường được nối với đầu ra của bộ giải mã địa chỉ. Khi một mạnh nhớ không được chọn thì buýt dữ liệu của nó bị treo (ở trạng thái trở kháng cao)

d) Nhóm tín hiệu điều khiển:
Tín hiệu điều khiển cần có trong tất cả các mạch nhớ. Các mạch nhớ ROM thường có một đầu vào điều khiển $\overline{OE}$ (output enable) để cho phép dữ liệu được đưa ra buýt. Một mặt mạch nhớ không được mở bởi $\overline{OE}$ thì buýt dữ liệu của nó bị treo. Một mạch nhớ RAM nếu chỉ có một tín hiệu điều khiển thì thường đó là $\overline{R}/\overline{W}$ để điều khiển quá trình ghi/đọc. Nếu mạch nhớ RAM có hai tín hiệu điều khiển đó thường là $\overline{WE}$ (write enable) để điều khiển ghi và $\overline{OE}$ để điều khiển đọc. Hai tín hiệu này phải loại trừ lẫn nhau (ngược pha) để điều khiển việc ghi/đọc mạch nhớ.

Một thông số đặc trưng khác của bộ nhớ là thời gian truy nhập t_{ac} được định nghĩa như là thời gian kể từ khi có xung địa chỉ trên buýt địa chỉ cho đến khi có dữ liệu ra ổn định trên buýt dữ liệu. Thời gian thâm nhậm bộ nhớ phụ thuộc rất nhiều vào công nghệ chế tạo nên nó. Các bộ nhớ làm bằng công nghệ lưỡng cực có thời gian truy nhập nhỏ (10 - 30ns) còn các bộ nhớ làm bằng công nghệ MOS có thời gian truy nhập lớn hơn nhiều (cỡ 150ms hoặc hơn nữa).

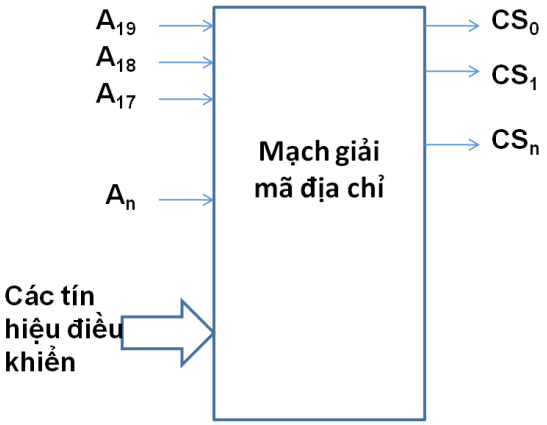
IV.2.2 Giải mã địa chỉ cho bộ nhớ

IV.2.2.1 Giới thiệu

Mỗi mạch nhớ nối ghép với CPU cần phải được CPU tham chiếu chính xác khi thực hiện các thao tác ghi/đọc. Điều đó có nghĩa là mỗi mạch nhớ phải được gán cho một vùng riêng biệt có địa chỉ xác định nằm trong không gian địa chỉ tổng thể của bộ nhớ. Việc gán địa chỉ cụ thể cho mạch nhớ được thực hiện nhờ một xung chọn vi mạch lấy từ mạch giải mã địa chỉ. Việc phân định không gian địa chỉ tổng thể thành các cùng nhớ khác nhau để thực hiện những chức năng nhất định gọi là phân vùng bộ nhớ. Ví dụ, đối với CPU 8086/8088 thì không gian địa chỉ tổng thể dành cho bộ nhớ là 1MB, trong đó vùng nhớ dung lượng 1 KB kể từ địa chỉ thấp nhất 00000H nhất thiết phải được dành cho RAM (vì đây là bảng gồm 256 vectơ ngắt của 8086/8088), tại còn vùng nhớ có chứa địa chỉ FFFF0H thì lại nhất thiết phải dành cho ROM hay EPROM (vì FFFF0H là địa chỉ bắt đầu của đoạn mã khởi động của CPU).

Về nguyên tắc một bộ giải mã địa chỉ khái quát thường có cấu tạo như trên Hình IV-9 dưới đây. Đầu vào của bộ giải mã là các tín hiệu địa chỉ và tín hiệu điều khiển. Các tín hiệu địa chỉ gồm các bít địa chỉ có quan hệ nhất định với các tín hiệu chọn vỏ ở đầu ra. Thường là các tín hiệu địa chỉ tương ứng với dải địa chỉ cấp cho vi mạch nhớ sẽ sinh ra tín hiệu chọn vỏ tương ứng. Tín hiệu điều khiển thường là tín hiệu IO/ $\overline{M}$ dùng để phân biệt đối tượng mà CPU chọn làm việc là bộ nhớ hay thiết bị vào/ra. Mạch giải mã là một trong những khâu tăng

thêm trễ thời gian của tín hiệu từ CPU tới bộ nhớ hoặc thiết bị ngoại vi. Tuỳ theo quy mô của mạch giải mã mà ta có thể có ở đầu ra một hay nhiều tín hiệu chọn vỏ.



Hình IV-9. Mạch giải mã địa chỉ tổng quát

Giải mã đầy đủ cho một mạch nhớ đòi hỏi ta phải đưa đến đầu vào của mạch giải mã các tín hiệu địa chỉ sao cho tín hiệu ở đầu ra của nó chỉ chọn riêng mạch nhớ đã định. Trong trường hợp này ta phải dùng tổ hợp đầy đủ của các đầu vào địa chỉ tương ứng để chọn được mạch nhớ, vì xung nhận được từ mạch giải mã ngoài việc chọn mạch nhớ ở vùng đã định sẽ có thể chọn ra các mạch nhớ ở các vùng nhớ khác nữa. Nói cách khác, từ một tổ hợp tín hiệu địa chỉ, bộ giải mã sẽ chỉ sinh ra một tín hiệu chọn vỏ duy nhất ứng với không gian địa chỉ cấp cho vi mạch nhớ.

Giải mã địa chỉ thiếu hay giải mã rút gọn thì ta chỉ dùng một nhóm trong số các tín hiệu địa chỉ để sinh ra tín hiệu chọn vỏ cho mạch nhớ. Như vậy, từ một tổ hợp các tín hiệu địa chỉ có thể sinh ra nhiều tín hiệu chọn vỏ khác nhau. Vì sử dụng ít tín hiệu hơn nên mạch giải mã thiếu cần ít linh kiện hơn nhưng lại làm mất tính đơn trị của xung chọn thu được ở đầu ra.

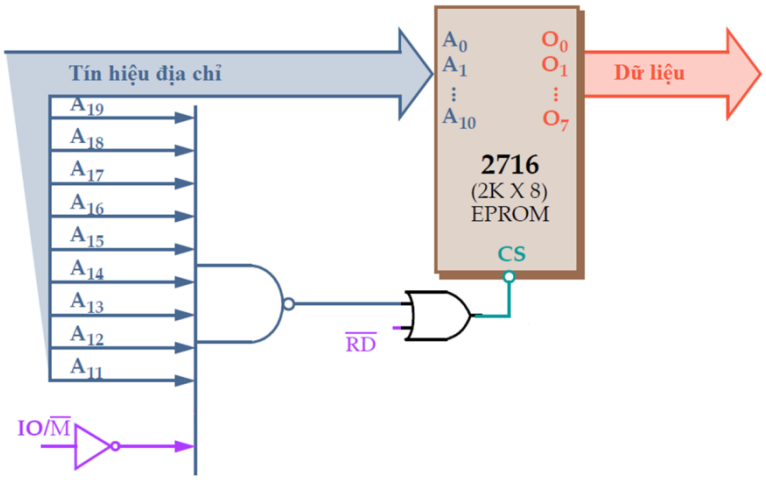
Ví dụ: Chíp nhớ C có dung lượng 10000H ô nhớ và được gán cho dải địa chỉ từ 00000H-0FFFFH. Để sinh ra tín hiệu chọn vỏ cho C ta có thể sử dụng duy nhất tín hiệu địa chỉ A_{16} ở mức thấp (A_{16}=0) hoặc cả bốn tín hiệu A_{16}-A_{19} ở mức thấp (A_{16}=. . . =A_{19}=0). Với trường hợp thứ nhất ta có giải mã thiếu do A_{16}=0 có thể do các yêu cầu truy nhập tới dải địa chỉ 20000H-2FFFFH.

Thông thường khi thiết kế mạch giải mã người ta hay tính đôi ra một chút để dự phòng, sao cho sau này nếu có sự thay đổi do phải tăng thêm dung lượng của bộ nhớ thì vẫn có thể sử dụng được mạch giải mã đã được thiết kế. Nói cách khác, hệ thống có thể mở rộng thêm không gian nhớ bằng các bổ sung thêm các vi mạch nhớ. Phần dưới đây sẽ xem xét một số phương pháp thực hiện mạch giải mã địa chỉ bộ nhớ.

IV.2.2.2 Thực hiện mạch giải mã bằng các mạch lô-gíc đơn giản

Các mạch lô-gíc đơn giản bao gồm các mạch AND, OR, NOT hay kết hợp như NAND, NOR. Bằng các mạch kiểu này ta có thể xây dựng được mạch giải mã địa chỉ đơn giản với số đầu ra hạn chế. Các mạch lô-gíc làm nhiệm vụ tổ hợp các tín hiệu địa chỉ và điều khiển

đọc/ghi bộ nhớ sao cho với một tổ hợp địa chỉ cho trước sẽ sinh ra tín hiệu chọn vỏ tương ứng.



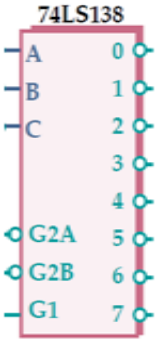
Hình IV-10. Mạch giải mã dùng mạch lô-gíc

Hình IV-10 giới thiệu mạch giải mã cho mạch EPROM 2716 có dung lượng 2K ô nhớ mỗi ô chứa 8 bít, làm việc trong dải địa chỉ FF800H-FFFFFH. Do mạch nhớ có dung lượng 2K tương ứng với dải địa chỉ 0FFH-7FFH (tương ứng với A_0. . . A_{10}). Như vậy, số lượng các tín hiệu địa chỉ dùng sinh ra tín hiệu kích hoạt chíp nhớ này là A_{11}-A_{19}. Với dải địa chỉ cho trước FF800H-FFFFFH thì tổ hợp A_{11}=. . . =A_{19}=1 sẽ sinh ra tín hiệu chọn vỏ cho EPROM 2716. Bên cạnh đó, ta cần phối hợp với các tín hiệu điều khiển IO/ $\overline{M}$ và RD (ở mức thấp) để tạo ra tín hiệu chọn vỏ.

Như trong hình vẽ, các tín hiệu địa chỉ và tín hiệu đảo của IO/ $\overline{M}$ được liên kết trực tiếp với nhau bằng phép lô-gíc AND rồi đảo. Do tính chất của mạch AND kết quả tổ hợp là duy nhất. Đầu ra sẽ chỉ bằng 1 khi tất cả đầu vào bằng 1. Đầu ra của mạch NAND được OR với RD (mức thấp) để sinh ra tín hiệu chọn vỏ (kích hoạt). Tương tự, do tính chất của mạch OR đầu ra sẽ chỉ bằng 0 nếu tất cả các đầu vào bằng 0 nên tín hiệu chọn vỏ là tín hiệu duy nhất được sinh ra ứng với thao tác truy nhập tới dải địa chỉ FF800H-FFFFFH. Như vậy, mạch giải mã trên là mạch giải mã đầy đủ.

IV.2.2.3 Thực hiện bộ giải mã dùng mạch giải mã tích hợp

Khi ta muốn có nhiều đầu ra chọn vỏ từ bộ giải mã mà vẫn dùng các mạch logic đơn giản thì thiết kế sẽ trở nên rất cồng kềnh do số lượng các mạch tăng lên. Trong trường hợp như vậy ta thường sử dụng các mạch giải mã tích hợp có sẵn. Một trong các mạch giải mã hay được sử dụng là 74LS138 cho phép giải mã 3 tín hiệu đầu vào thành 8 tín hiệu đầu ra như trong hình dưới đây.



Inputs						Output							
Enable			Select										
G2A	G2B	G1	C	B	A	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	0

Hình IV-11. 74LS138 và bảng trạng thái

Giả sử chúng ta cần xây dựng mạch giải mã cho không gian nhớ 256KB tương ứng với dải địa chỉ F8000H-FFFFFH trong đó mỗi mạch nhớ 2732 có dung lượng 4K×8. Từ dải địa chỉ được gán và dung lượng của từng mạch nhớ, có thể thấy rằng từ các tín hiệu địa chỉ A_{13} tới A_{19} cần phải sinh ra 8 tín hiệu kích hoạt các vi mạch nhớ ứng với 8 dải địa chỉ như bảng dưới đây:

Bảng IV-5. Dải tín hiệu của các mạch nhớ 2732

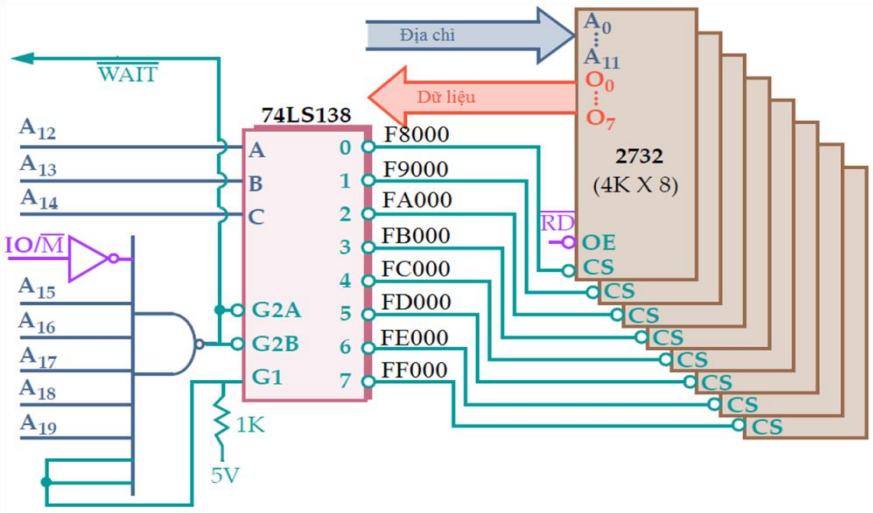
Địa chỉ	A_{19}-A_{16}	A_{15}	A_{14}	A_{13}	A_{12}
F8	1111	1	0	0	0
F9	1111	1	0	0	1
FA	1111	1	0	1	0
FB	1111	1	0	1	1
FC	1111	1	1	0	0
FD	1111	1	1	0	1
FE	1111	1	1	1	0
FF	1111	1	1	1	1

Qua bảng trên, ta thấy chỉ có các tín hiệu A_{12}-A_{14} là thay đổi còn A_{15}-A_{19} bằng 1 và không đổi. Như vậy ta có thể sử dụng mạch giải mã 74LS138 để sinh ra các tín hiệu chọn vỏ cho các mạch nhớ như hình sau:

Trong hình vẽ dưới đây, các tín hiệu A_{12}-A_{14} được nối trực tiếp vào tín hiệu đầu vào (A-C) của 74LS138. Các tín hiệu địa chỉ còn lại A_{15}-A_{19} và các tín hiệu điều khiển IO/ $\overline{M}$ được nối vào tín hiệu điều khiển của 74LS138 (G2A, G2B). Tín hiệu G1 luôn ở mức lô-gíc 1. Các đầu ra của 74LS138 được nối lần lượt với các mạch nhớ ứng với dải địa chỉ gán trước.

Tại thí dụ này ta thấy mạch giải mã có sẵn 74LS138 có số lượng đầu vào địa chỉ và đầu vào cho phép hạn chế. Nếu ta có số lượng đầu vào cho địa chỉ lớn mà ta lại phải giải mã đầy đủ để thực hiện bộ giải mã đã hoàn chỉnh ta vẫn phải dùng thêm các mạch logic phụ. Đây cũng là lý do để người ta thay thế các bộ giải mã kiểu này bằng các bộ giải mã dùng PROM

hoặc PLA (programable logic array) với ưu điểm chính là chúng có rất nhiều đầu vào cho các bít địa chỉ và vì thế rất thích hợp trong các hệ vi xử lý sau này với không gian địa chỉ lớn



Hình IV-12. Giải mã sử dụng 74LS138

IV.2.2.4 Thực hiện bộ giải mã dùng PROM

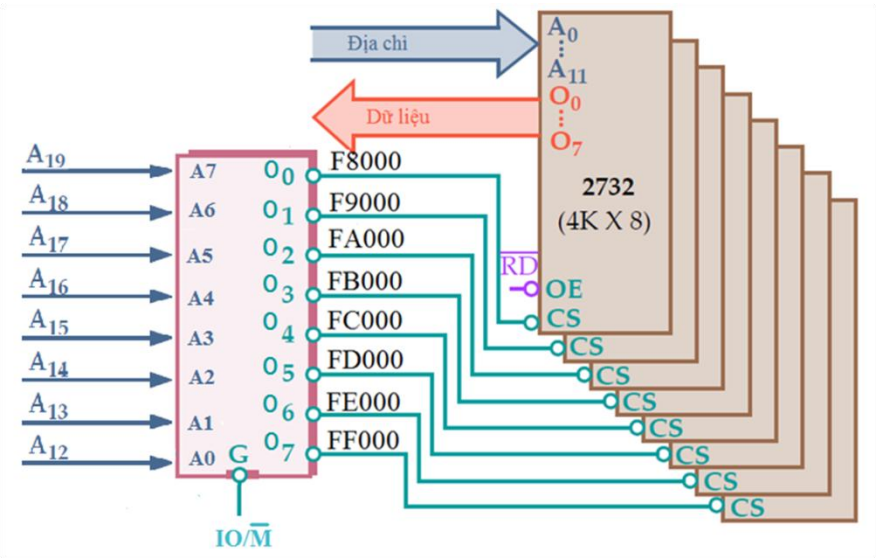
Việc sử dụng bộ nhớ ROM làm bộ giải mã lợi dụng số lượng lớn các tín hiệu địa chỉ đầu vào, điều khiển và dữ liệu ra của mạch nhớ ROM. Với mỗi tổ hợp tín hiệu địa chỉ và điều khiển đầu vào, mạch nhớ ROM sẽ sinh ra một nhóm tín hiệu trên kênh dữ liệu. Trạng thái của các tín hiệu dữ liệu này tùy thuộc vào giá trị được lưu vào trong ROM trước đó. Nếu các tín hiệu này loại trừ lẫn nhau thì các tín hiệu dữ liệu có thể được dùng làm các tín hiệu chọn vi mạch nhớ.

Để trình bày ứng dụng của PROM trong việc thực hiện các bộ giải mã ta lấy lại ví dụ phân vùng bộ nhớ cho ROM trong phần trước. Tại đây ta dùng mạch PROM 256 byte để làm bộ giải mã. Trong bảng dưới đây là mẫu các bít để ghi vào PROM cho trường hợp ứng dụng cụ thể này.

Bảng IV-6. Mẫu dữ liệu ghi vào ROM

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	$\_O_0$	$\_O_1$	$\_O_2$	$\_O_3$	$\_O_4$	$\_O_5$	$\_O_6$	$\_O_7$
1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1
1	1	1	1	1	0	0	1	1	0	1	1	1	1	1	1
1	1	1	1	1	0	1	0	1	1	0	1	1	1	1	1
1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	0	0	1	1	1	1	0	1	1	1
1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1
1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Theo bảng trên, trong dải địa chỉ từ F8H-FFH của ROM ta ghi 8 giá trị sao cho tín hiệu dữ liệu đầu ra chỉ có duy nhất một tín hiệu mức thấp còn tất cả các tín hiệu còn lại đều ở mức cao. Ngoài 8 ô nhớ này, tất cả các ô nhớ khác của ROM đều được điền giá trị FFH.



Hình IV-13. Giải mã dùng ROM

Mạch giải mã cho bộ nhớ PROM được thể hiện trên hình trên so với cách thực hiện bộ giải mã bằng 74LS138 chúng ta không phải dùng đến các mạch phụ điều này làm giảm đáng kể kích thước vật lý của bộ giải mã. Ngoài ra ta có thể dễ dàng thay đổi địa chỉ của các mạch nhớ bằng cách thay đổi vị trí và giá trị dữ liệu trong mạch nhớ giải mã ROM.

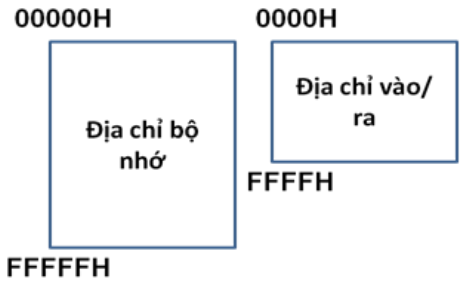
IV.3 Phối ghép vi xử lý với thiết bị vào ra

IV.3.1 Giới thiệu về thiết bị vào/ra

Đối với 8086/8088 (hay họ 80x86 nói chung) có 2 cách phối ghép CPU với các thiết bị ngoại vi (các cổng vào/ra, I/O):

a) Thiết bị vào/ra có không gian địa chỉ tách biệt (Hình IV-14)
Trong cách phối ghép này, bộ nhớ được dùng toàn bộ không gian 1MB mà CPU dành cho nó. Các thiết bị ngoại vi (các cổng) sẽ được dành riêng một không gian 64KB cho mỗi loại cổng vào hoặc ra. Để phân biệt các thao tác truy nhập, ta phải dùng tín hiệu IO/ $\overline{M}$ =1, và các lệnh trao đổi dữ liệu một cách thích hợp cho mỗi không gian đó. Với các thiết bị này cần sử dụng các câu lệnh IN, OUT để trao đổi dữ liệu.

b) Thiết bị vào/ra và bộ nhớ có chung không gian địa chỉ
Trong cách phối ghép này, bộ nhớ và thiết bị ngoại vi cùng chia nhau không gian địa chỉ 1MB mà CPU 8086/8088 có khả năng địa chỉ hóa. Các thiết bị ngoại vi sẽ chiếm một vùng nào đó trong không gian 1MB, phần còn lại là của bộ nhớ. Tất nhiên trong trường hợp này ta dùng chung tín hiệu IO/ $\overline{M}$ =0 và lệnh trao đổi dữ liệu kiểu lệnh MOV cho cả bộ nhớ và thiết bị ngoại vi



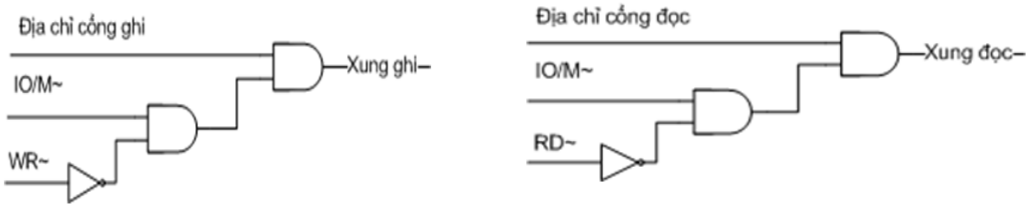
Hình IV-14. Không gian nhớ của thiết bị vào/ra và bộ nhớ chính

IV.3.2 Giải mã địa chỉ thiết bị vào ra

IV.3.2.1 Giới thiệu

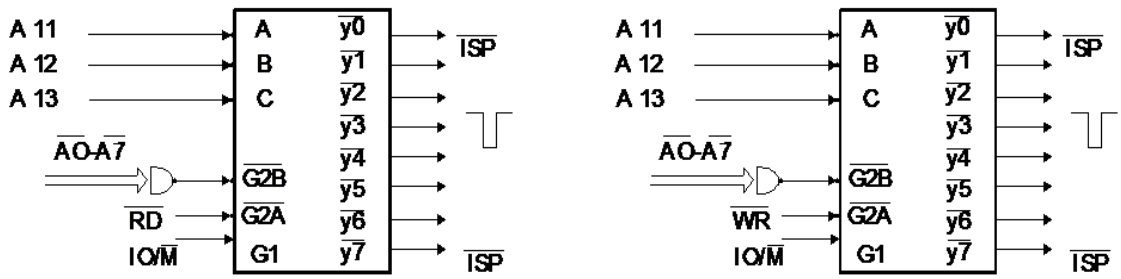
Việc giải mã địa chỉ cho thiết bị vào/ra cũng gần giống như giải mã địa chỉ cho mạch nhớ. Ta sẽ nhấn mạnh ở đây việc giải mã địa chỉ cho các cổng. Thông thường các cổng có địa chỉ 8 bít tại A0-A7, trong một số hệ vi xử lý khác (như các máy IBM PC) các cổng có 16 bít tại A0 - A15. Tuỳ theo độ dài của toán hạng trong lệnh là 8 hay 16 bít ta có 1 cổng 8 bít có địa chỉ liên nhau để tạo nên từ với độ dài tương ứng. Trong thực tế ít có hệ sử dụng hết 256 cổng vào/ra khác nhau nên ta chỉ xét ở đây các bộ giải mã địa chỉ 8 bít A0-A7 và mạch giải mã thông dụng có sẵn (như 74LS138) để tạo ra các xung chọn thiết bị.

Các mạch giải mã đơn giản có thể tạo được từ mạch lô-gíc đơn giản như sau:



Hình IV-15. Giải mã thiết bị dùng cổng lô-gíc

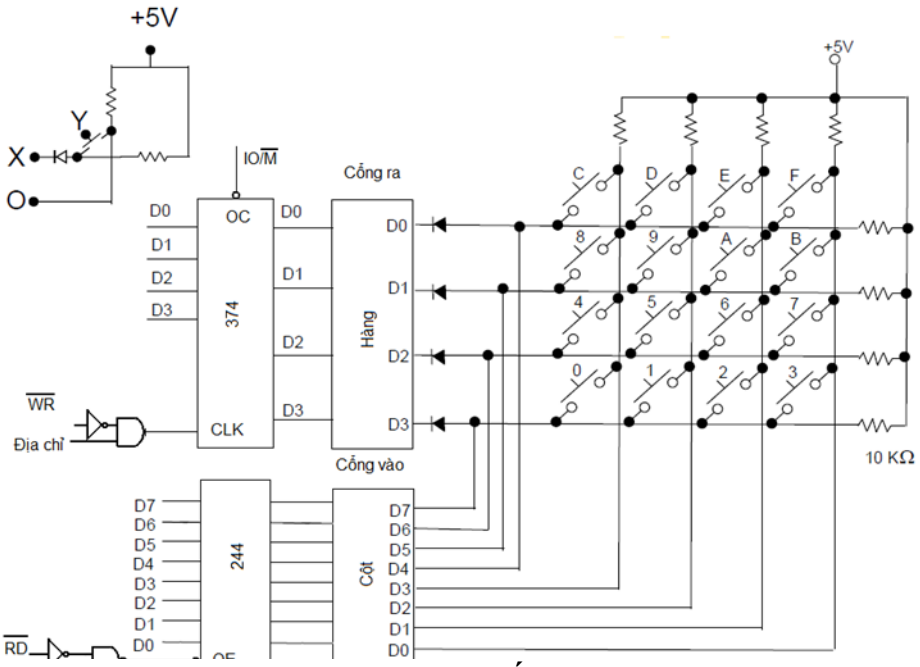
Trong trường hợp cần nhiều xung chọn ở đầu ra cho các cổng vào/ra có địa chỉ liên tiếp, ta có thể dùng các mạch giải mã có sẵn kiểu 74LS138. Như trên hình dưới đây trình bày 2 mạch tương tự nhau dùng 74LS138 để giải mã địa chỉ cho 8 cổng vào và 8 cổng ra. Trên cơ sở mạch này ta cũng có thể phối hợp với cả hai tín hiệu đọc và ghi để tạo ra tín hiệu chọn cho việc đọc/ghi từng cổng vào/ra ra cụ thể.



Hình IV-16. Giải mã địa chỉ cổng dùng 74LS138

IV.3.2.2 Các mạch cổng đơn giản

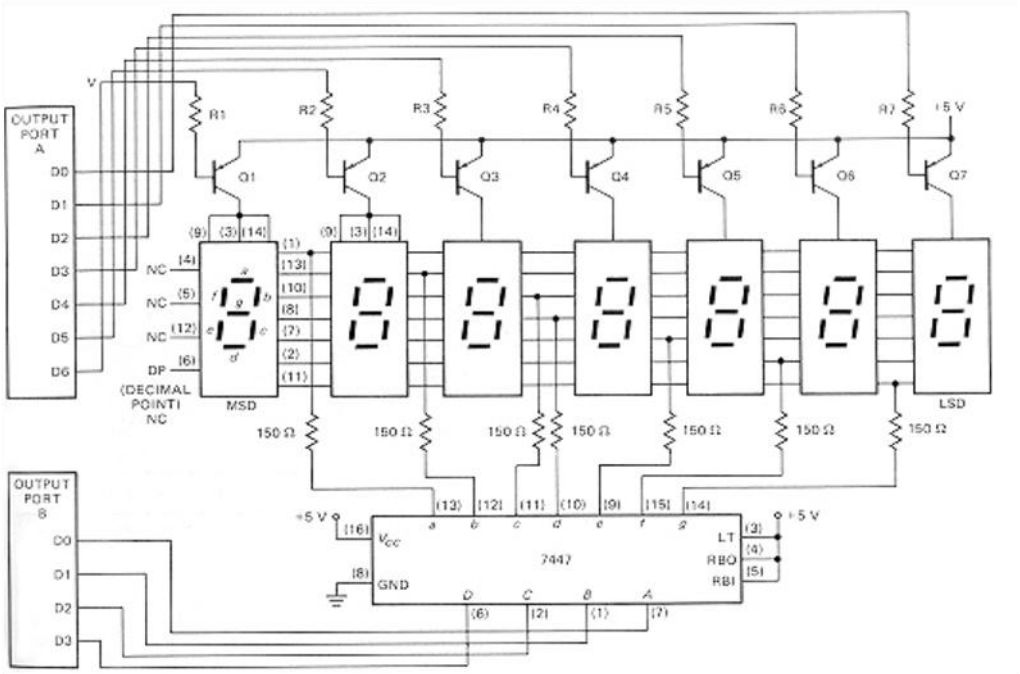
Trong thực tế có rất nhiều vi mạch tổ hợp cỡ vừa có thể được dùng làm cổng phối ghép với bộ vi xử lý để vào/ra dữ liệu. Các mạch này thường được cấu tạo từ các mạch chốt 8 bít có đầu ra 3 trạng thái (74LS373: kích theo mức; 74LS374: kích theo sườn), các mạch khuếch đại đệm 2 chiều 8 bít đầu ra 3 trạng thái (74LS245). Chúng được dùng trong các phối ghép đơn giản để làm cho CPU và thiết bị ngoại vi hoạt động tương thích với nhau, ví dụ như để đệm buýt hoặc các mạch cổng để tạo ra các tín hiệu móc nối. . . Dưới đây là một số ví dụ



Hình IV-17. Ghép nối với bàn phím

Hình IV-17 biểu diễn ghép nối giữa 8086 với bàn phím 16 số dạng tiếp điểm. Vi mạch 74LS374 được dùng để điều khiển các tín hiệu hàng và 74LS244 dùng để điều khiển các tín hiệu cột của bàn phím. Nguyên tắc hoạt động của một phím như sau. Nếu tín hiệu X ở mức cao (lô-gíc 1) thì đi-ốt sẽ khóa lại, vậy nên tiếp điểm Y có đóng xuống hay không thì tại đầu O ta luôn thu được điện áp 5V (không có dòng điện). Nếu tín hiệu X ở mức thấp (lô-gíc 0), thì đi-ốt mở và khi tiếp điểm Y đóng xuống tại đầu O ta thu được điện áp 0V. Bằng cách quét tuần tự các hàng và đọc trên các cột ta sẽ xác định được phím bấm. Giả sử tín hiệu địa chỉ giải mã vi mạch đệm cổng 374 là 0AH còn 244 là 0BH, đoạn mã sau đây cho phép xác định phím C có được bấm hay không:

```
Hang    EQU 0AH
Cot     EQU 0BH
        MOV AL,11111110b    ; Chỉ có D0=0
        OUT Hang, AL
Ktra:   IN AL, Cot          ; Đọc tín hiệu cột
        AND AL,00001000b    ; Giữ lại bít D3 ứng với phím C
        JNZ Ktra            ; Không bấm
        ...                 ; Phím C được bấm
```



Hình IV-18. Ghép nối hiển thị số

Hình IV-18 biểu diễn một mạch hiển thị số sử dụng vi mạch 7447 và. LED bảy đoạn. 7447 cho phép điều khiển các LED bảy đoạn bằng cách giải mã số BCD tại đầu vào (A-D) và sinh ra các tín hiệu kích hoạt các thanh led của LED bảy đoạn (a-g). Để tiết kiệm chi phí, 7447 được dùng chung cho cả 7 LED bảy đoạn. Việc kích hoạt LED bảy đoạn được điều khiển thông qua cổng A và các transitor Q_1-Q_7, dữ liệu số cần hiển thị được gửi qua cổng B. Đoạn mã sau đây dùng để kiểm tra hệ thống LED bằng cách hiển thị trên cả 7 LED số 8. Chú ý rằng để bật 1 LED_i ta cần đưa tín hiệu dữ liệu D_i=0 trên cổng 0AH tới transitor Q_i tương ứng.

```
DK_LED  EQU 0AH          ; Cổng điều khiển LED
DL_LED  EQU 0BH          ; Cổng dữ liệu hiển thị
        MOV AL,FFH       ; Tắt tất cả các LED
        OUT DK_LED, AL
        MOV CX,16        ; Trễ
Tre:    NOP
        LOOP Tre
        MOV AL,8         ; Đưa số 8 ra 7447
        OUT DL_LED,AL
        XOR AL,AL        ; Đặt AL=0
        OUT DK_LED,AL    ; Bật tất cả các LED
```

IV.4 Giới thiệu một số vi mạch hỗ trợ vào ra

Để thực hiện trao đổi dữ liệu vào/ra, vi xử lý có thể sử dụng một số vi mạch chuyên dụng cho phép trao đổi dữ liệu kiểu song song như Intel 8255A hỗ trợ 3 cổng dữ liệu 8 bít

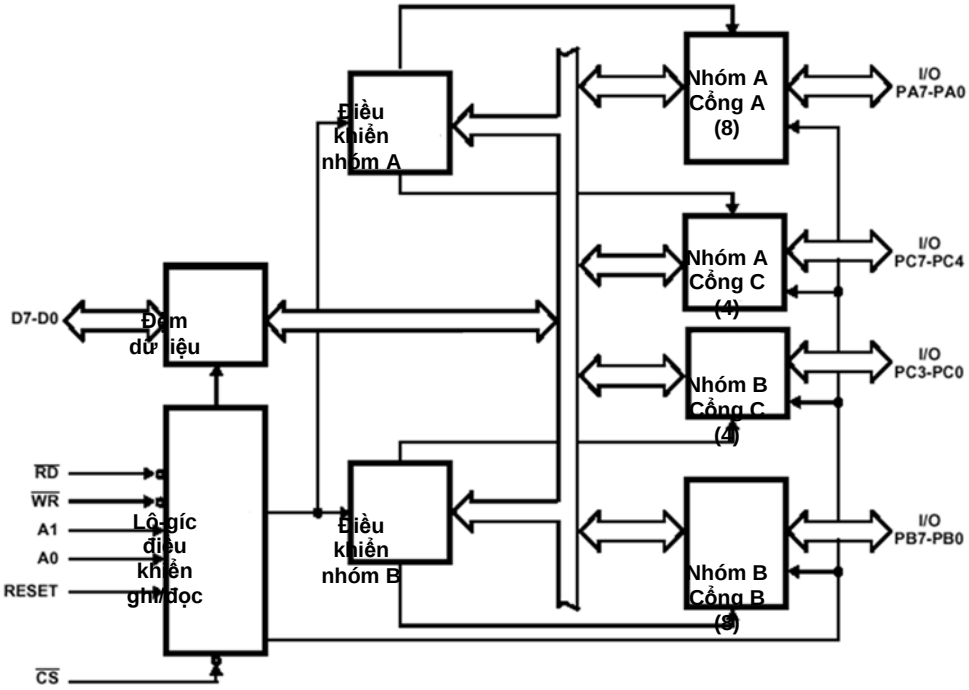
hay trao đổi dữ liệu nối tiếp như Intel 8251 hay 8250. Hai phương pháp đều có ưu và nhược điểm riêng. Với việc trao đổi dữ liệu song song, kênh dữ liệu có thể ghép nối trực tiếp với các thiết bị bên ngoài mà không cần phải thực hiện việc biến đổi tín hiệu, tốc độ trao đổi dữ liệu lớn, tuy nhiên khoảng cách ghép nối các thiết bị ngắn. Ghép nối nối tiếp cho phép mở rộng cự ly ghép nối thiết bị, tuy nhiên tốc độ trao đổi dữ liệu hạn chế và phải thực hiện việc biến đổi dữ liệu trên kênh dữ liệu song song của hệ vi xử lý thành chuỗi các tín hiệu nối tiếp.

Phần dưới đây giới thiệu ghép nối dữ liệu song song sử dụng vi mạch Intel 8255A và truyền thông nối tiếp sử dụng vi mạch Intel 8251A.

IV.4.1 Ghép nối song song dùng 8255A

IV.4.1.1 Giới thiệu 8255A

Vi mạch 8255A là thiết bị giao tiếp ngoại vi lập trình được (Programmable Peripheral Interface-PPI) dùng cho hệ thống máy tính Intel. Thiết bị có thể được lập trình mà không cần thiết bị logic ngoài để giao tiếp với thiết bị ngoại vi. Dưới đây là sơ đồ khối Hình IV-19.



Hình IV-19. Sơ đồ khối 8255A

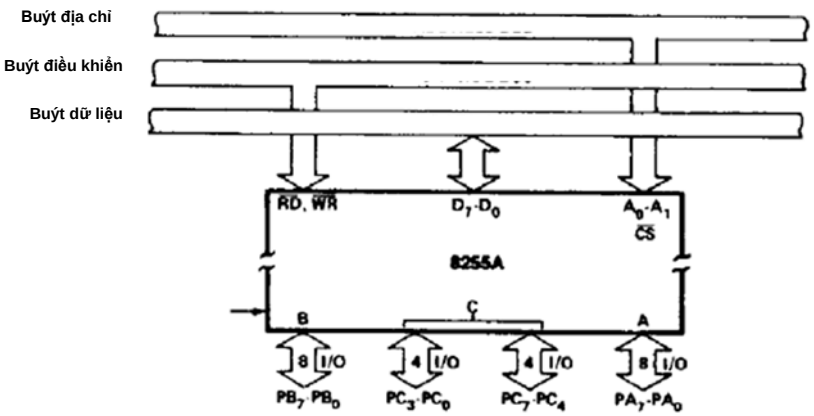
Các tín hiệu của 8255A có ý nghĩa như sau :

CS:	Chọn chíp (mức thấp)	PA_7-PA_0:	Cổng A
RD:	Đọc (mức thấp)	PB_7-PB_0:	Cổng B
WR:	Ghi (mức thấp)	PC_7-PC_0:	Cổng C
A_0A_1:	Chọn cổng	D_7-D_0:	Dữ liệu

Vi mạch 8255A cung cấp 3 cổng vào/ra A,B, và C có độ rộng 8 bít, chia làm 2 nhóm A, B. Các cổng này có thể được lập trình để làm việc trong ba chế độ:

a) *Chế độ 0: Vào/ra cơ sở*:

Chế độ này cung cấp thao tác vào/ra đơn giản cho từng cổng, trên các cổng không có tín hiệu kết nối. Các cổng A, B và C có thể được chia thành 2 cổng 8 bít (A,B) và 2 cổng 4 bít (C thấp PC_0-PC_3, C cao PC_4-PC_7). Bất kỳ cổng nào có thể dùng làm cổng vào/ra.



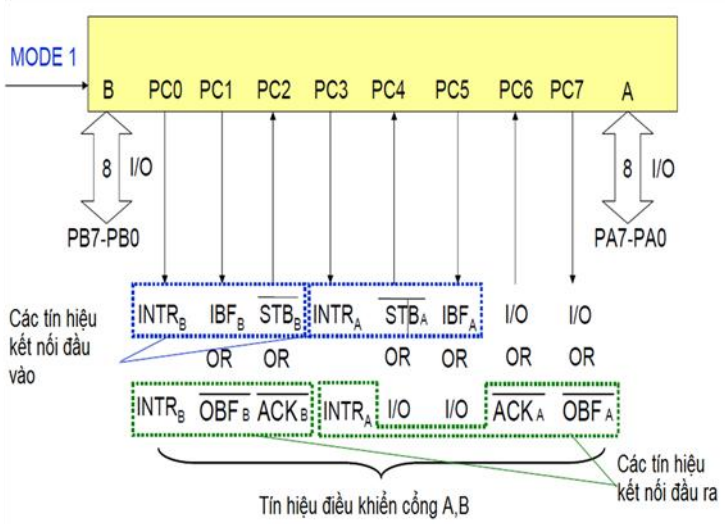
Hình IV-20. Các chế độ 0

b) Chế độ 1: Vào/ra thăm dò

Chế độ này chỉ được cung cấp trên hai cổng A,B, mỗi cổng có kênh dữ liệu là 8 bít và 4 tín hiệu điều khiển lấy từ cổng C. Dữ liệu trên kênh có thể là vào hay ra. Các nhóm tín hiệu điều khiển vào/ra như sau:

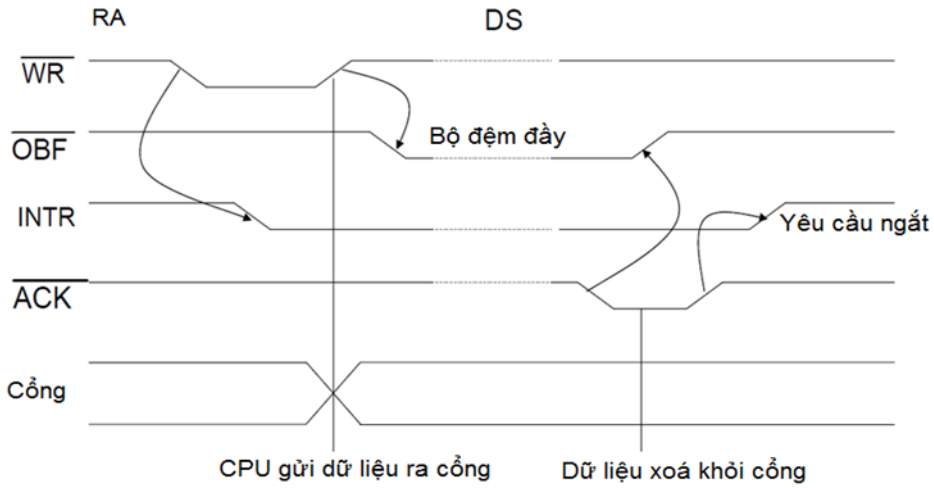
Đầu vào	**Đầu ra**
STB: Kiểm tra đầu vào (mức thấp)	OBF: Dữ liệu ra sẵn sàng (mức thấp)
IBF: Dữ liệu sẵn sàng (mức cao)	ACK: Nhận xong dữ liệu (mức thấp)
INTR: Báo ngắt CPU (mức cao)	INTR: Báo ngắt CPU (mức cao)

Các tín hiệu điều khiển của hai cổng A và B lấy từ cổng C như sau:



Hình IV-21. Ghép nối các tín hiệu điều khiển ở chế độ 1

Các tín hiệu điều khiển ra này biến đổi như hình vẽ dưới đây

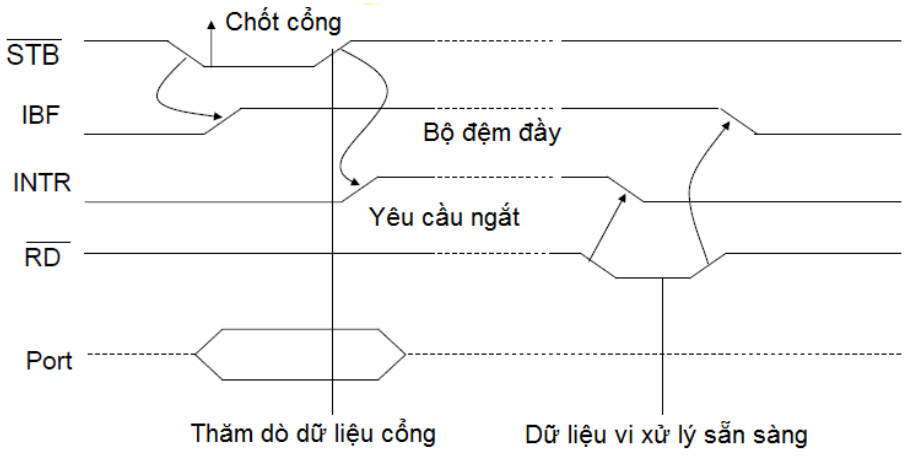


Hình IV-22. Biểu đồ thời gian tín hiệu ra

Với cổng A các tín hiệu điều khiển hoạt động như sau:

- $\overline{OBFA}$ (Đệm ra của PA đầy). Tín hiệu báo cho thiết bị ngoại vi biết CPU đã ghi dữ liệu vào cổng để chuẩn bị đưa ra. Tín hiệu này thường được nối với $\overline{STB}$ của thiết bị nhận.
- $\overline{ACKA}$ (Trả lời đã nhận được dữ liệu). Đây là tín hiệu của thiết bị ngoại vi cho biết là nó đã nhận được dữ liệu từ PA của 8255A.
- INTRA (Yêu cầu ngắt từ PA). Đây là kết quả thu được từ quan hệ giữa các tín hiệu khác của 8255A trong quá trình đối thoại với thiết bị ngoại vi, nó được dùng để phản ánh yêu cầu ngắt của PA tới CPU (xem biểu đồ quan hệ giữa các tín hiệu trong hình 5.38).
- INTEA là tín hiệu của một mạch lật bên trong 8255A để cho phép/cấm yêu cầu ngắt INTRA của PA. INTEA được lập/xoá thông qua bit PC6 của PC.

Các tín hiệu điều khiển vào thay đổi như hình vẽ dưới đây

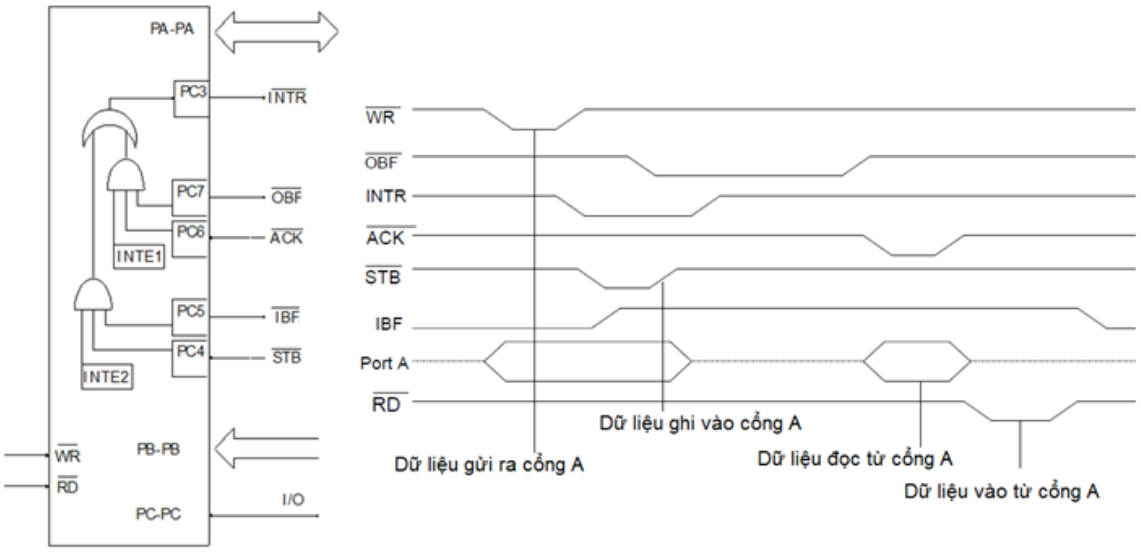


Hình IV-23. Biểu đồ thời gian tín hiệu vào

- $\overline{STB}$ (Cho phép chốt dữ liệu): Khi dữ liệu đã sẵn sàng để được đọc vào bằng PA, thiết bị ngoại vi phải dùng $\overline{STB}$ để báo cho 8255A biết để chốt dữ liệu.
- IBF (Đệm vào đầy): Sau khi 8255A chốt được dữ liệu do thiết bị ngoại vi đưa đến nó đưa ra tín hiệu IBF để báo cho thiết bị ngoại vi biết là đã chốt xong.
- INTR : Tín hiệu để báo cho CPU biết là đã có dữ liệu sẵn sàng để đọc từ PA. Đây là kết quả thu được từ quan hệ giữa các tín hiệu khác của 8255A trong quá trình đối thoại với thiết bị ngoại vi

c) Chế độ 2: Vào/ra hai chiều

Chế độ này chỉ áp dụng được cho cổng A và tất cả các tín hiệu của cổng C được dùng làm tín hiệu kết nối như trong Hình IV-24. Các tín hiệu kết nối biến đổi tuỳ thuộc theo dữ liệu được gửi ra hay đọc về từ cổng A.



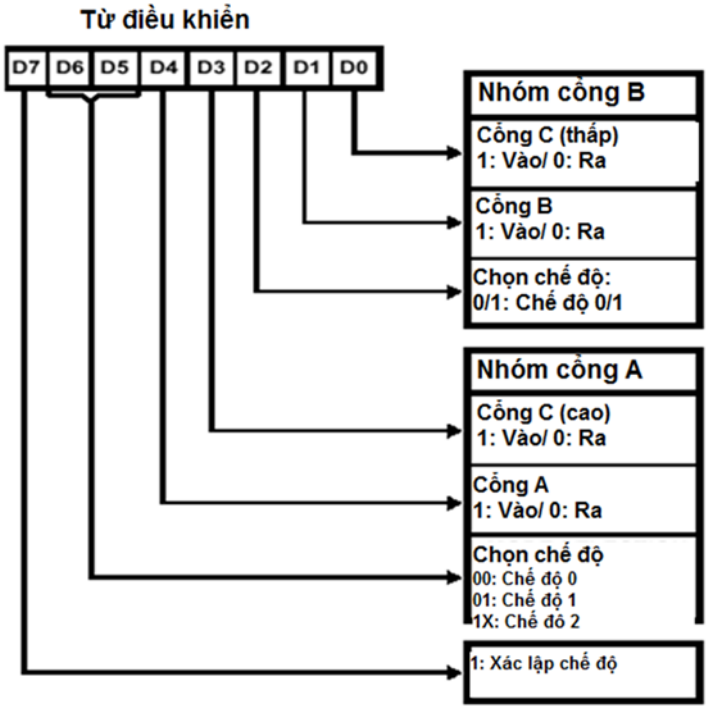
Hình IV-24. Các tín hiệu kết nối hai chiều và biểu đồ thời gian

IV.4.1.2 Lập trình 8255A

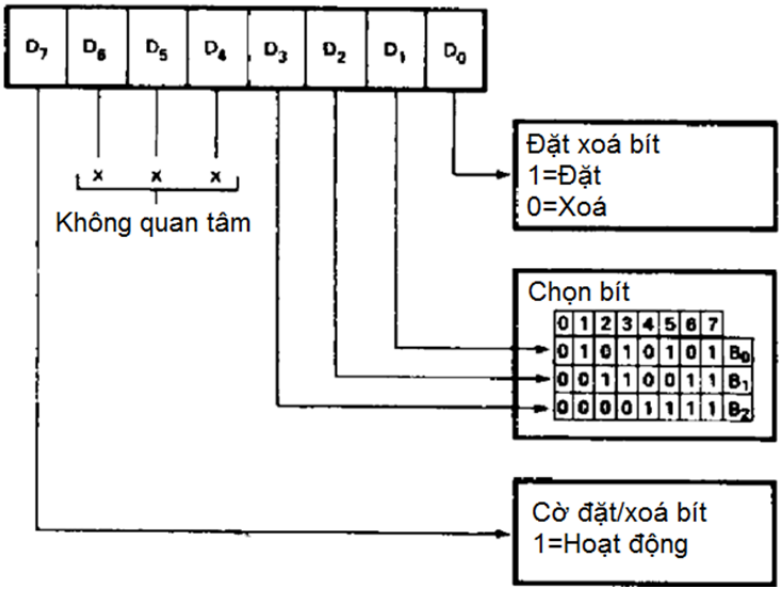
Các thanh ghi của 8255A được xác định qua tính hiệu địa chỉ A_0A_1 như sau

A1	A0	Thanh ghi
x	x	Không sử dụng
0	0	Cổng A (PA)
0	1	Cổng B (PB)
1	0	Cổng C (PC)
1	1	Điều khiển

Ý nghĩa các bít của thanh ghi điều khiển chế độ hoạt động như trong Hình IV-25. Chú ý khi này bít có nghĩa lớn nhất của thanh ghi điều khiển nhận giá trị 1. Thanh ghi này cũng được dùng để xác lập trạng thái của các tín hiệu điều khiển trên cổng C khi 8255A hoạt động ở chế độ 1 hoặc 2.



Hình IV-25. Thanh ghi điều khiển chế độ



Hình IV-26. Đặt xoá các tín hiệu điều khiển trên cổng C

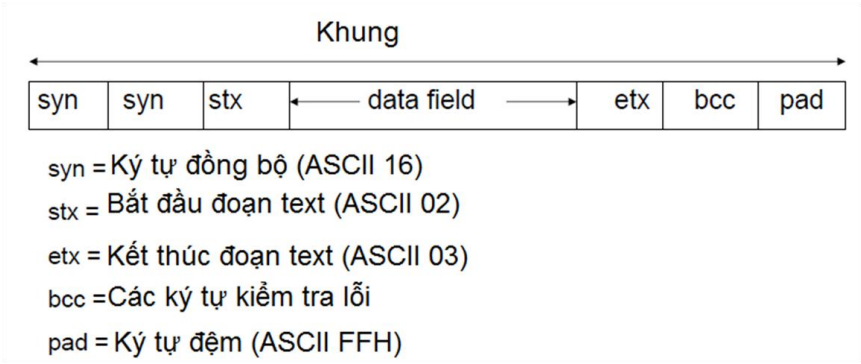
IV.4.2 Truyền thông nối tiếp dùng 8251

IV.4.2.1 Giới thiệu truyền thông nối tiếp

Việc truyền thông tin giữa các bộ phận nằm gần nhau trong hệ thống vi xử lý có thể thực hiện thông qua buýt song song mở rộng hoặc qua các mạch phối ghép song song trong đó các byte hoặc được truyền đi trên một tập các đường dẫn bằng mạch in hoặc dây cáp trong trường hợp cần phải truyền thông tin giữa các thiết bị ở cách xa nhau, ta không thể dùng cả

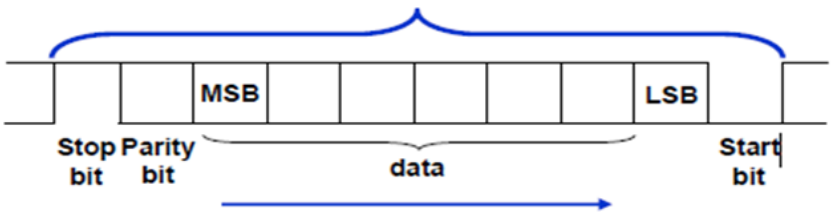
tập các đường dây như trên mà phải có cách truyền khác để đảm bảo chất lượng tín hiệu cũng như tiết kiệm được số đường dây dẫn cần thiết. Từ yêu cầu trên ra đời phương pháp truyền thông tin nối tiếp, tín hiệu được truyền đi liên tiếp từng bít trên một đường dây (như đường điện thoại chẳng hạn). Ở đầu thu tín hiệu nối tiếp sẽ được biến đổi ngược lại để tái tạo hiệu dạng song song thích hợp cho việc xử lý tiếp theo. Trong thực tế có 2 phương pháp truyền thông tin kiểu nối tiếp: đồng bộ và không đồng bộ.

Trong phương pháp truyền đồng bộ, dữ liệu được truyền theo từng khối với một tốc độ xác định. Khối dữ liệu trước khi được truyền đi sẽ được bổ sung thêm các phần tử đặc biệt ở đầu và ở cuối tạo thành khung. Các phần tử này dùng để đánh dấu điểm bắt đầu của khối dữ liệu hay các thông tin giúp phát hiện lỗi trong quá trình truyền. Hình IV-27 biểu diễn cấu trúc khung dữ liệu để truyền đồng bộ. Đây thực chất là cách điều khiển hướng ký tự vì các ký tự đặc biệt được dùng để đánh dấu các phần khác nhau trong khung.



Hình IV-27. Cấu trúc khung đồng bộ

Trong cách truyền thông dị bộ, dữ liệu được truyền đi theo từng ký tự riêng biệt. Độ dài ký tự có thể thay đổi từ 5 đến 8 bít. Ký tự cần truyền đi được gắn thêm 1 bít đánh dấu ở đầu để báo bắt đầu kí tự (Start bit) và một hoặc hai bít báo kết thúc kí tự (Stop bit), và một bít kiểm tra tính toàn vẹn dữ liệu (Parity bit). Vì mỗi kí tự được nhận dạng riêng biệt nên nó có thể được truyền đi vào bất kì lúc nào. Giữa các kí tự truyền đi có thể có các khoảng cách về thời gian. Dạng thức của dữ liệu truyền đi theo phương pháp dị bộ được thể hiện trên hình dưới đây.



Hình IV-28. Cấu trúc dữ liệu truyền dị bộ

Tốc độ truyền dữ liệu theo phương pháp nối tiếp được đo bằng bít/chu kỳ. Ngoài ra người ta cũng hay dùng đơn vị baud. Đó là giá trị nghịch đạo của thời gian giữa các lần thay đổi mức tín hiệu, với dữ liệu chỉ có hai mức (0 và 1) và mỗi thay đổi mức tín hiệu chỉ mã hoá một bít thì tốc độ baud bằng tốc độ bít/s. Trong các phương pháp mã hóa khác, người ta có

thể mã hóa nhiều hơn một bít thông tin trên một trạng thái tín hiệu. Các giá trị tốc độ truyền thường gặp trong thực tế là 2400, 4800, 9. 600. . .

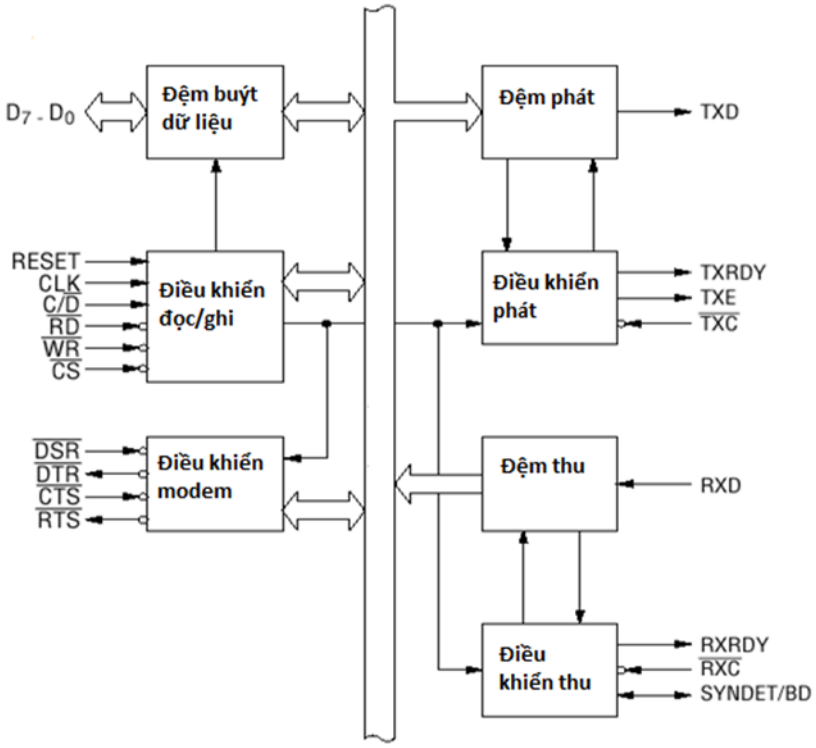
Để tạo điều kiện dễ dàng cho việc phối ghép đường truyền nối tiếp với hệ vi xử lý và để giảm tối đa các mạch phụ thêm ở ngoài. Người ta đã chế tạo ra các vi mạch tổ hợp cỡ lớn lập trình có khả năng hoàn thành các công việc cần thiết trong khi phối ghép đó là các mạch thu phát dị bộ vạn năng (N8250/16450 của National Semiconductor universal asynchronous receiver - transmitter USART) và mạch thu phát đồng bộ - dị bỏ vạn năng 8251A của Intel (universal synchronous - asynchronous receiver - transmitter USART).

Với các mạch phối ghép như trên, việc truyền tin dị bộ chẳng hạn sẽ được thực hiện nhờ 1USART ở đầu phát và 1 USART khác ở đầu thu. Khi có kí tự để phát 8251A tạo ra khung cho kí tự bằng cách gắn thêm vào mã kí tự các bít start, parity và stop và gưi liên tiếp từng bít ra đường truyền. Bên phía thu, 1 8521A khác sẽ nhận kí tự tháo bỏ khung, kiểm tra parity, rồi chuyển sang dạng song song để CPU đọc.

IV.4.2.2 Mạch USART 8251A

IV.4.2.2.a Sơ đồ khối và tín hiệu

Trong phần này ta sẽ giới thiệu mạch 8251A, đó là mạch USART có thể dùng cho hai kiểu truyền thông tin nối tiếp đồng bộ. Sơ đồ khối của mạch 8251A của Intel được biểu diễn trên hình dưới đây.



Hình IV-29. Sơ đồ khối 8251A

Các tín hiệu của mạch 8251A hầu hết là giống tín hiệu của 8086/8088. Chân chọn vỏ của 8251A phải được nối với đầu ra của một mạch giải mã địa chỉ để đặt mạch 8251A vào một địa chỉ cơ bản nào đó. Các tín hiệu cần được giải thích thêm gồm:

+ CLK [I]: Chân nối đến xung đồng hồ của hệ thống.

+ TxRDY [0]: Tín hiệu báo đệm giữ rỗng (sẵn sàng nhận ký tự mới từ CPU)

+ RxRDY [0]: Tín hiệu báo đệm thu đầy (có ký hiệu nằm chờ CPU đọc vào)

+ TxEMPTY [0]: Tín hiệu báo cả đệm thu và đệm phát đều rỗng.

+ C/D [I]: CPU thao tác với thanh ghi lệnh / thanh ghi dữ liệu của 8251A, khi C/D=1 thì thanh ghi lệnh được chọn làm việc. Chân này thường được nối với A0 của buýt địa chỉ để cùng với các tín hiệu WR và RD chọn ra 4 thanh ghi bên trong 8251A.

+ RxC [I] và TxC [I]: Xung đồng hồ cung cấp cho các thanh ghi dịch của phần thu và phần phát. Thường 2 thanh này nối chung để phần thu và phần phát làm việc với cùng tầng số nhịp. Tần số của các khung đồng hồ đưa đến chân RxC và TxC được chọn sao cho là bội số (cụ thể là gấp 1, 16 hoặc 64) của tốc độ thu hay tốc độ phát theo yêu cầu.

+ $\overline{DSR}$ $\overline{DTR}$ là hai cặp tín hiệu yêu cầu thiết bị modem sẵn sàng và trả lời của modem với tín hiệu yêu cầu.

+ $\overline{RTS}$ $\overline{CTS}$ là cặp tín hiệu yêu cầu modem sẵn sàng phát và đáp ứng của modem với tín hiệu yêu cầu.

+ SYNDET/BRKDET [O]: khi 8251A làm việc ở chế độ không đồng bộ, nếu RxD = 0 kéo dài hơn thời gian của 2 ký tự thì chân này có mức cao để báo là việc truyền hoặc đường truyền bị gián đoạn. Khi 8251A làm việc ở chế độ đồng bộ, nếu phần thu tìm thấy ký tự đồng bộ rong bản tin thu được thì chân này có mức cao.

Đệm ở phần phát của mạch 8251A là loại đệm kép, bao gồm đệm giữ và đệm phát. Trong khi 1 ký tự đang được chuyển đi ở đệm phát thì một ký tự khác có thể đưa từ CPU sang đệm giữ. Các tín hiệu TxRDY và TxEMPTY sẽ cho biết trạng thái của các đệm này khi mạch 8251A hoạt động. Khi đệm ở phần thu đầy thì sẽ có tin shiệu RxRDY = 1. Nếu cho đến khi phần thu nhận được ký tự mới mà CPU không kịp thời đọc được ký tự cũ sẽ bị mất do bị đè bởi ký tự mới nhận được. Hiện tượng này gọi là thu đè.

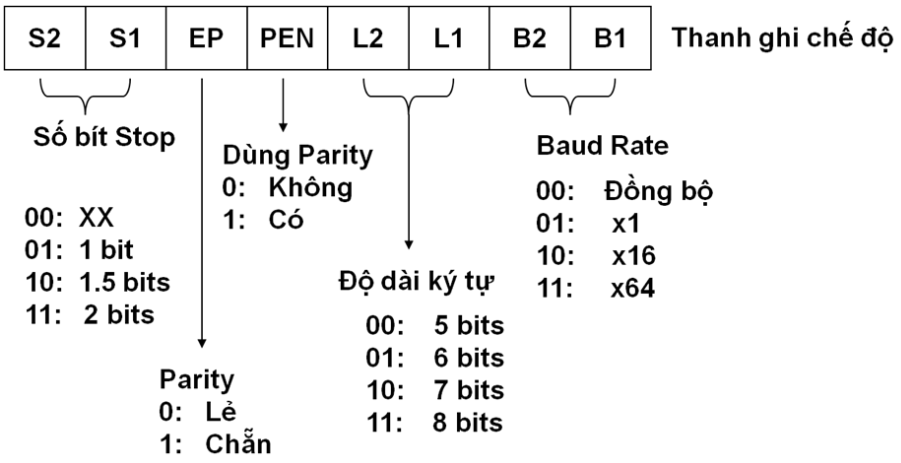
IV.4.2.2.b Các thanh ghi bên trong của 8251A

Như đã nói ở trên chân C/D (giải sử nó được nối vào A_0 của buýt địa chỉ) cùng các tín hiệu WR và RD sẽ chọn ra 4 thanh ghi bên trong của mạch USART, thanh ghi đệm dữ liệu thu, thanh ghi đệm dữ liệu phát, thanh ghi trạng thái và thanh ghi điều khiển (Bảng IV-7).

Bảng IV-7. Các thanh ghi bên trong của 8251A

A_0	$\overline{RD}$	$\overline{WR}$	Chọn ra
0	0	1	Thanh ghi đệm dữ liệu thu
0	1	0	Thanh ghi đệm dữ liệu phát
1	0	1	Thanh ghi trạng thái
1	1	0	Thanh ghi điều khiển

Thanh ghi chế độ:



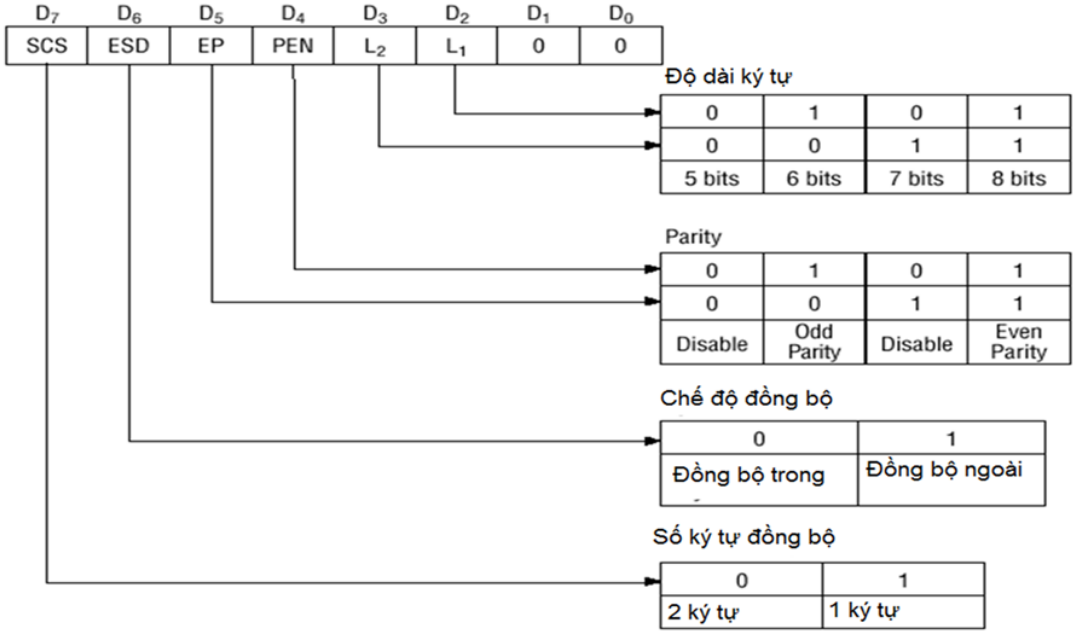
Hình IV-30. Thanh ghi chế độ (dị bộ)

Trong từ chế độ, đối với ký tự cần truyền ta có thể chọn số bít (kiểu mã) của ký tự, số bít stop và tốc độ truyền. Nếu có sẵn tầng số xung đồng hồ cho phần thu hoặc phần phát (giả sử là F_{dk}) và ta muốn truyền (thu/phát) dữ liệu vưới tốc độ X baud, ta phải chọn hệ số nhân tốc độ truyền k sao cho thỏa mãn biểu thức.

F_{dk} = X. K, trong đó X là các tốc độ truyền tiêu chuẩn.

Ví dụ: nếu ta có tần số xung đồng hồ phát là 19. 200Hz và ta muốn truyền dữ liệu với tốc độ 1. 200 baud thì ta phải ghi từ chế độ có 2 bít cuối là 10 để chọn được hệ số nhân tốc độ truyền là 16, vì 1200 x 16 = 19. 200. Với việc dùng tần số đồng hồ cho phần thu/ phát cao hơn so với tốc độ truyền ta sẽ giảm được lỗi khi truyền thông tin.

Hình dưới đây giới thiệu các giá trị của thanh ghi lệnh khi hoạt động ở chế độ truyền đồng bộ. Ở chế độ này ta không phải quan tâm tới tốc độ phát, thay vào đó ta cần xác định số lượng ký tự đồng bộ và độ dài của các ký tự truyền đi.



Hình IV-31. Thanh ghi chế độ (đồng bộ)

Thanh ghi lệnh:

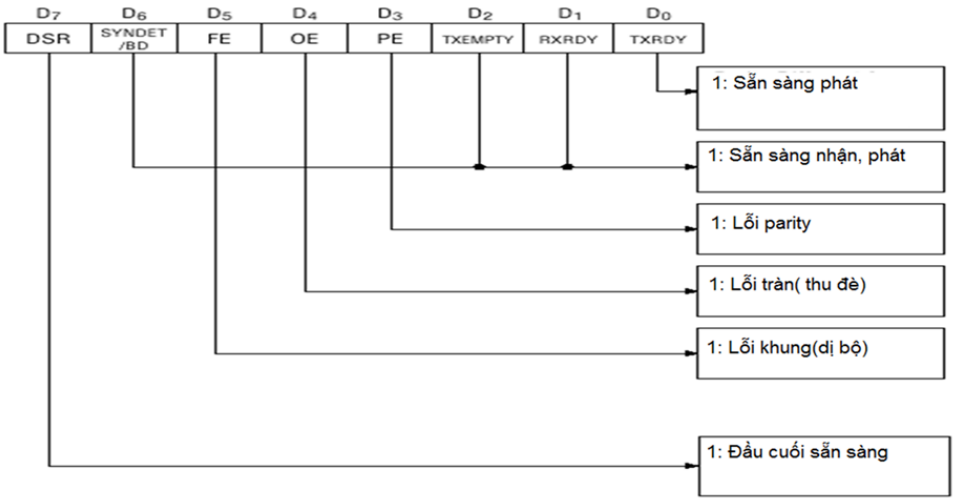
EH	IR	RTS	ER	SBRK	RxE	DTR	TxE	Thanh ghi lệnh

TxE: Cho phép truyền(=1)
DTR: Đầu cuối dữ liệu sẵn sàng(=1)
RxE: Cho phép nhận (=1)
SBPRK:Gửi ký tự gián đoạn(1)
ER: Xóa cờ(=1)
RTS: Yêu cầu gửi (=1)
IR: Khởi động lại(=1)
EH: Tìm ký tự đồng bộ(=1)

Hình IV-32. Cấu trúc thanh ghi lệnh

Thanh ghi trạng thái

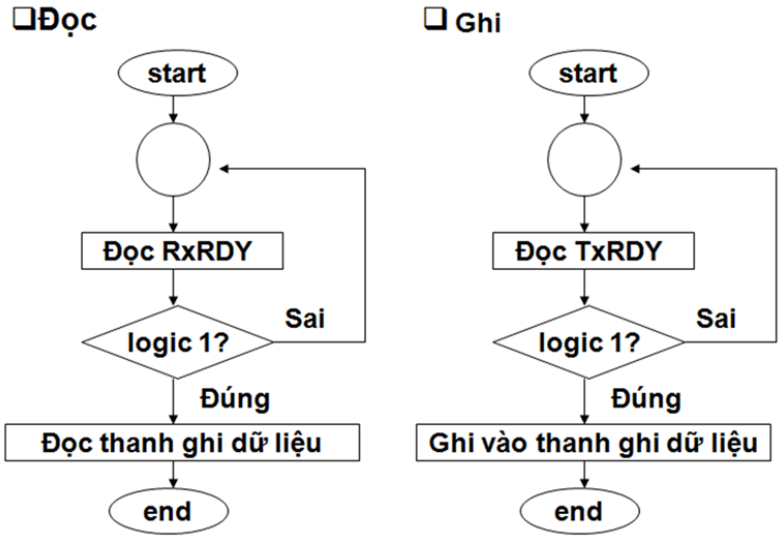
Giá trị trên các bít thanh ghi này cho ta biết tình trạng hoạt động của 8251A



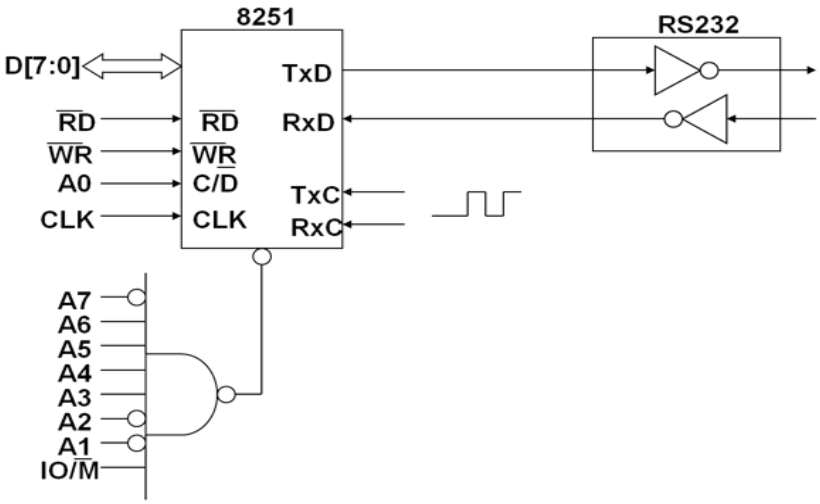
Hình IV-33. Cấu trúc thanh ghi trạng thái

IV.4.2.2.c Lập trình 8251A

Để lập trình cho 8251A trước tiên ta cần xác lập chế độ hoạt động bằng cách tính giá trị của thanh ghi chế độ và gửi ra cổng điều khiển. Để gửi hoặc nhận dữ liệu ta cần liên tục kiểm tra trạng thái của 8251A theo lưu đồ đọc/ghi đơn giản sau:



Hình IV-34. Lưu đồ đọc/ghi đơn giản



Hình IV-35. Ghép nối 8251A

Hình IV-35 giới thiệu mạch giải mã địa chỉ cho 8251A và ghép nối tín hiệu nối tiếp theo chuẩn RS232. Vi mạch 8251A hoạt động ở 2 cổng 78H và 79H (0111 100x). Lưu đồ đọc có thể được triển khai như sau:

```
DK      EQU 79H                 ; Thanh ghi điều khiển
TThai   EQU 79H                 : Thanh ghi trạng thái
DL      EQU 78H                 ;Thanh ghi dữ liệu
khoitao: MOV AL, 11001111b      ; Xác lập chế độ 8251A dị bộ 2 bít stop,
        OUT DK,AL               ; 8 bít dữ liệu không chẵn lẻ, tốc độ x64
Ktra:   IN AL, TThai            ; Kiểm tra trạng thái
        AND AL,02H              ; Bít 2 thanh ghi trạng thái RxRDY
        JNZ Ktra
DocDL: IN AL, DL
        ....                    ; Xử lý dữ liệu
```

Chương V. Tổng quan về các phương pháp vào ra dữ liệu

V.1 Giới thiệu

Kỹ thuật trao đổi dữ liệu giữa máy vi tính và các thiết bị ngoại vi được gọi là vào/ra hay I/O (Input/Output). Thiết bị liên lạc với máy vi tính qua các giao tiếp vào/ra. Người dùng có thể nhập chương trình và dữ liệu bằng các dùng bàn phím và chạy các chương trình để lấy kết quả. Như vậy, các thiết bị vào/ra kết nối tới máy vi tính cung cấp cách thức liên lạc tiện lợi với thế giới bên ngoài. Các thiết bị vào/ra phổ biến gồm có bàn phím, màn hình, máy in và ổ đĩa cứng.

Đặc tính của các thiết bị vào/ra thường khác với đặc tính của máy vi tính. Chẳng hạn như tốc độ của các thiết bị thường chậm hơn máy vi tính, độ dài từ (word) và định dạng dữ liệu cũng khác nhau giữa thiết bị và máy tính. Để hai bên có thể liên lạc được với nhau cần có các mạch giao tiếp giữa thiết bị vào/ra và máy tính. Giao tiếp cung cấp trao đổi dữ liệu vào/ra qua buýt vào/ra. Buýt này thông thường chuyển tải 3 loại tín hiệu: địa chỉ thiết bị, dữ liệu và lệnh.

Có ba phương pháp trao đổi dữ liệu giữa máy vi tính và các thiết bị vào/ra: vào/ra lập trình (programmed I/O) hay thăm dò, vào/ra bằng ngắt và truy nhập trực tiếp bộ nhớ (Direct Memory Access DMA). Dùng vào/ra thăm dò, vi xử lý chạy một chương trình thực hiện toàn bộ các trao đổi dữ liệu giữa vi xử lý và âccs thiết bị bên ngoài. Đặc tính chủ yếu của phương pháp này là thiết bị thực hiện các chức năng được chỉ định bởi chương trình bên tron bộ nhớ của vi xử lý. Nói cách khác, vi xử lý điều khiển hoàn toàn các trao đổi dữ liệu.

Với vào/ra bằng ngắt, thiết bị có thể bắt vi xử lý dừng việc thực hiện chương trình hiện thời để thiết bị có thết chạy chương trình khác gọi là chương trình phục vụ ngắt. Chương trình này đáp ứng yêu cầu của thiết bị. Sau khi kết thúc chương trình này, câu lệnh trở về từ ngắt để trả lại quyền điều khiển cho chương trình bị ngắt.

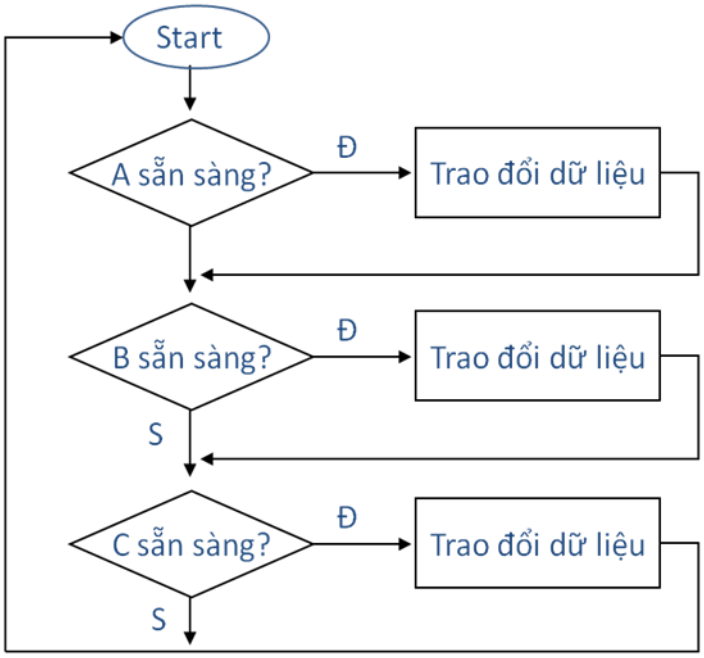
Truy nhập bộ nhớ trực tiếp là kỹ thuật vào/ra mà trong đó dữ liệu có thể được trao đổi giữa bộ nhớ của máy tính với thiết bị như ổ cứng mà không cần sự can thiệp của vi xử lý. Thông thường, phương pháp này cần sử dụng vi mạch đặc biệt gọi là vi mạch DMA.

Trong máy tính sử dụng hệ điều hành, người dùng thường làm việc với thiết bị vào/ra ảo. Người dùng không phải quan tâm tới các đặc tính của thiết bị. Thay vào đó, người dùng thực hiện trao đổi dữ liệu thông qua các dịch vụ vảo/ra do hệ điều hành cung cấp. Về căn bản, hệ điều hành đóng vai trò giao tiếp giữa chương trình người dùng và phần cứng thiết bị. Hệ điều hành hỗ trợ tạo nhiều các thiết bị lô-gíc hay thiết bị vào/ra ảo và cho phép người dùng liên lạc trực tiếp với các thiết bị này. Chương trình người dùng hoàn toàn không biết được việc ánh xạ giữa thiết bị ảo và thiết bị vật lý. Như vậy, khi thiết bị ảo gán cho thiết bị vật lý khác thì không phải thay đổi chương trình người dùng.

V.2 Vào/ra bằng phương pháp thăm dò

Vấn đề điều khiển vào/ra dữ liệu sẽ trở nên đơn giản nếu thiết bị ngoại nếu lúc nào cũng sẵn sàng để làm việc với CPU. Ví dụ, bộ phận do nhiệt độ số (như là một thiết bị vào) lắp sẵn trong một hệ thống điều khiển lúc nào cũng có thể cung cấp số đo về nhiệt độ của đối tượng cần điều chỉnh, còn một bộ đèn LED 7 nét (như là một thiết bị ra) dùng để chỉ thị một giá trị nào đó của một đại lượng vật lý nhất định trong hệ thống nói trên thì lúc nào cũng có thể biểu hiện thông tin đó. Như vậy khi CPU muốn có thông tin về nhiệt độ của hệ thống thì nó chỉ việc đọc cổng phối ghép với bộ đo nhiệt độ, và nếu CPU muốn biểu diễn thông tin vừa đọc được trên đèn LED thì nó chỉ việc đưa tín hiệu điều khiển tới đó mà không cần phải kiểm tra xem các thiết bị này có đang sẵn sàng làm việc hay không.

Tuy nhiên trong thực tế không phải lúc nào CPU cũng làm việc với các đối tượng "liên tục sẵn sàng" như trên. Thông thường khi CPU muốn làm việc với một đối tượng nào đó, trước tiên nó phải kiểm tra xem thiết bị đó có đang ở trạng thái sẵn sàng làm việc hay không; nếu có thì nó mới thực hiện vào việc trao đổi dữ liệu. Như vậy, nếu làm việc theo phương thức thăm dò thì thông thường CPU chia sẻ thời gian hoạt động cho việc trao đổi dữ liệu và việc kiểm tra trạng thái sẵn sàng của thiết bị ngoại vi thông qua các tín hiệu móc nối (handshake signal).



Hình V-1. Vào/ra lập trình với nhiều thiết bị

Các mạch kết nối trong Hình IV-17 và Hình IV-18 là các ví dụ tiêu biểu cho phương pháp vào/ra lập trình. Với bàn phím, CPU liên tục kiểm tra trạng thái các phím và nếu có phím được bấm CPU sẽ đọc thông tin trên cổng vào để xác định phím nào được bấm. Với bộ hiển thị LED, CPU liên tục đưa dữ liệu ra các cổng ra để thiết bị có thể hiển thị các thông tin.

Khi số lượng các thiết bị vào/ra tăng lên thì thời gian dành cho việc xác định trạng thái của thiết bị vào/ra cũng tăng lên nhanh chóng như trong Hình V-1. CPU kiểm tra lần lượt các

thiết bị để phát hiện trạng thái sẵn sàng trao đổi dữ liệu của từng thiết bị và thực hiện các lệnh trao đổi dữ liệu. Các thiết bị được kiểm tra thăm dò theo trật tự ngẫu nhiên hoặc theo mức độ ưu tiên của các thiết bị. Cách thức này dù đơn giản song có nhược điểm là thời gian quét trạng thái của các thiết bị chiếm tỷ trọng rất đáng kể trong suốt quá trình vào/ra nhất là khi các thiết bị chưa có dữ liệu để trao đổi.

V.3 Vào/ra bằng ngắt

V.3.1 Giới thiệu

Nhược điểm của vào/ra thăm dò là máy tính cần kiểm tra bít trạng thái bằng cách chờ đáp ứng của thiết bị vào/ra. Với các thiết bị chậm, việc chờ làm giảm khả năng xử lý dữ liệu khác của máy tính. Kỹ thuật ngắt cho phép giải quyết vấn đề này.

Với vào/ra bằng ngắt, thiết bị khởi xướng việc trao đổi vào/ra. Thiết bị được nối với chân tín hiệu ngắt (INT) trên vi mạch của vi xử lý. Khi thiết bị cần trao đổi dữ liệu, thiết bị sinh ra tín hiệu ngắt. Máy tính sẽ hoàn thành câu lệnh hiện thời và lưu nội dung của bộ đếm chương trình và các thanh ghi trạng thái. Sau đó, máy tính tự động nạp địa chỉ của chương trình phục vụ ngắt vào thanh ghi đếm chương trình. Chương trình này thường do người dùng viết và máy tính thực hiện chương trình này để trao đổi dữ liệu với thiết bị. Câu lệnh cuối của chương trình này khôi phục thanh ghi đếm chương trình bị dừng và thanh ghi trạng thái của vi xử lý.

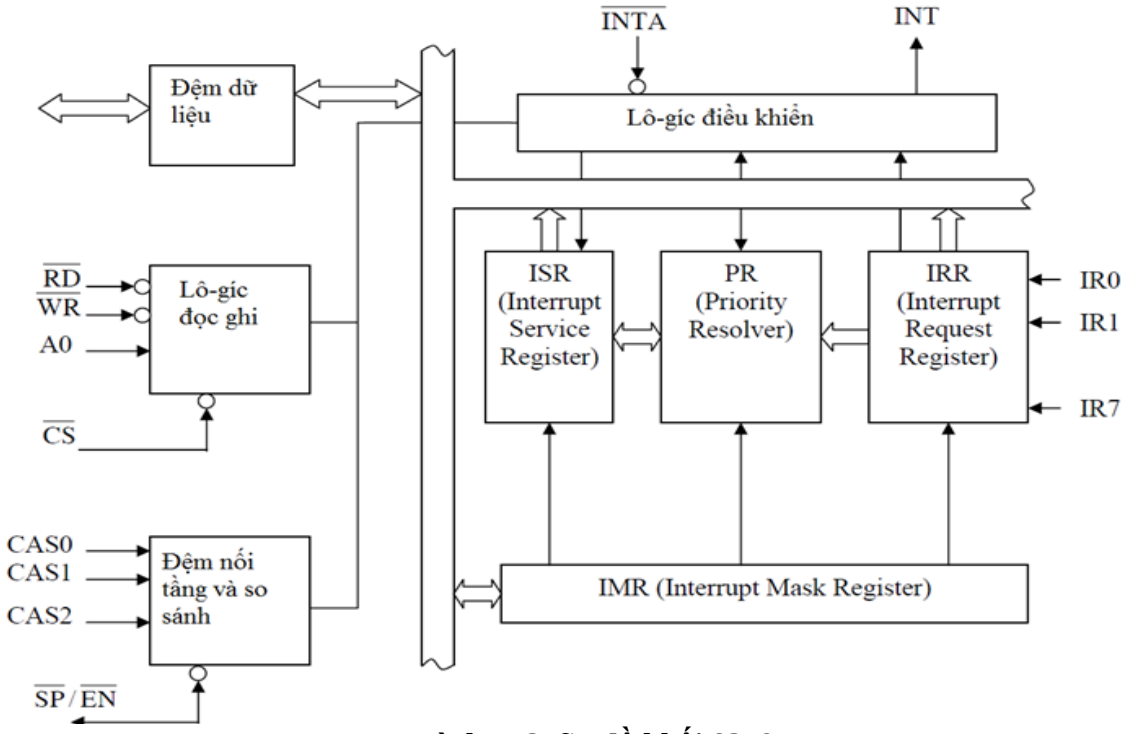
Vi xử lý thường cung cấp một hay nhiều tín hiệu ngắt trên vi mạch. Như vậy, để xử lý các yêu cầu ngắt từ nhiều thiết bị cần có cơ chế đặc biệt. Thường có các cách sau: thăm dò và quay vòng. Thăm dò sử dụng phần mềm chung cho tất cả các thiết bị vì vậy làm giảm tốc độ đáp ứng ngắt. Khi có tín hiệu ngắt phần mềm thăm dò kiểm tra trạng thái của các thiết bị theo thứ tự ưu tiên bắt đầu với thiết bị được ưu tiên cao nhất. Khi xác định được thiết bị yêu cầu trao đổi dữ liệu, phần mềm thăm dò chuyển quyền điều khiển cho phần mềm phục vụ ngắt.

V.3.2 Bộ xử lý ngắt ưu tiên 8259

Trong trường hợp có nhiều yêu cầu ngắt che được từ bên ngoài phải phục vụ máy tính thường dùng vi mạch có sẵn 8259A để giải quyết vấn đề ưu tiên. Vi mạch 8259A được gọi là mạch điều khiển ngắt lập trình được (Programmable Interrupt Controller, PIC). Đó là một vi mạch cỡ lớn có thể xử lý trước được 8 yêu cầu ngắt với các mức ưu tiên khác nhau để tạo ra một yêu cầu ngắt đưa đến đầu vào INTR (yêu cầu ngắt che được của CPU 8086/8088. Nếu nối tầng 1 mạch 8259A chủ với 8 mạch 8259A thợ ta có thể nâng tổng số các yêu cầu ngắt với các mức ưu tiên khác nhau lên thành 64.

V.3.2.1 Các khối chức năng chính của 8259A

Sơ đồ khối của 8259A được trình bày trong hình vẽ dưới đây



Hình V-2. Sơ đồ khối 8259

- Thanh ghi IRR: ghi nhớ các yêu cầu ngắt có tại đầu vào IRi.
- Thanh ghi ISR: ghi nhớ các yêu cầu ngắt đang được phục vụ trong số các yêu cầu ngắt IRi.
- Thanh ghi IMR: ghi nhớ mặt nạ ngắt đối với các yêu cầu ngắt IRi.
- Logic điều khiển: khối này có nhiệm vụ gửi yêu cầu ngắt tới INTR của 8086/8088 khi có tín hiệu tại các chân IRi và nhận trả lời chấp nhận yêu cầu ngắt INTA từ CPU để rồi điều khiển việc đưa ra kiểu ngắt trên buýt dữ liệu.
- Đệm buýt dữ liệu: dùng để phối ghép 8259A với buýt dữ liệu của CPU
- Logic điều khiển ghi/đọc: dùng cho việc ghi các từ điều khiển và đọc các từ trạng thái của 8259A.
- Khối đệm nối tầng và so sánh: ghi nhớ và so sánh số hiệu của các mạch 8259A có mặt trong hệ vi xử lý.

V.3.2.2 Các tín hiệu của 8259A

Một số tín hiệu trong mạch 8259 có tên giống như các tín hiệu tiêu chuẩn của hệ vi xử lý 8086/8080. Ta có thể thấy rõ và hiểu được ý nghĩa của chúng ngay trên Hình V-2. Ngoài các tín hiệu này ra, còn có một số tín hiệu đặc biệt khác của 8259A cần phải giới thiệu thêm gồm:

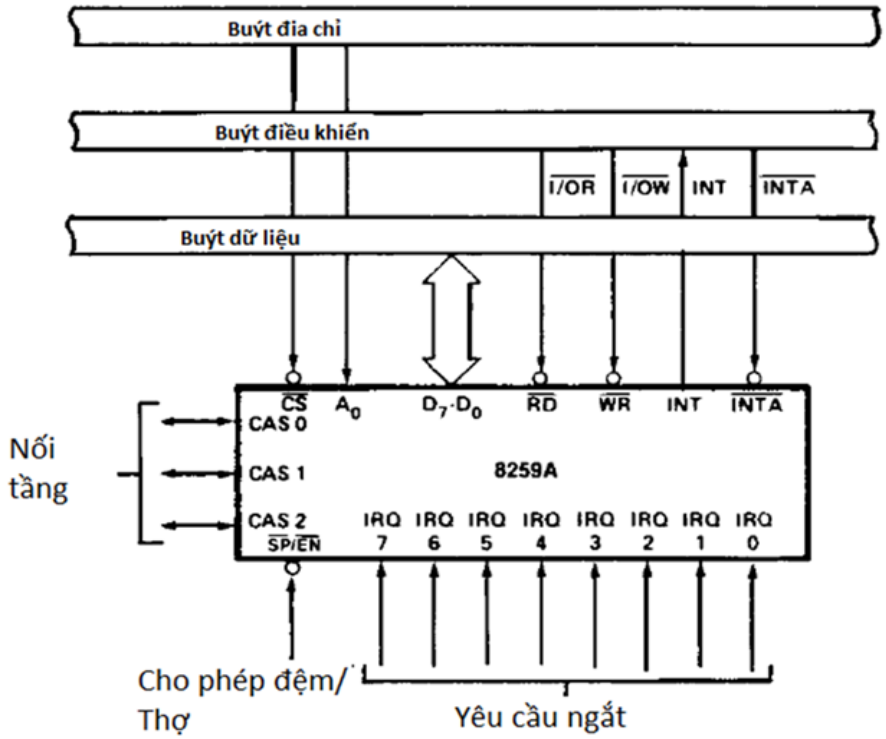
+ CAS_0-CAS_2 [I, O]: là các đầu vào đối với các mạch 8259A thợ hoặc các đầu ra của mạch 8259A chủ dùng khi cần nối tầng để tăng thêm các yêu cầu ngắt cần xử lý.

+ $\overline{SP}/\overline{EN}$ [I, O]: khi 8259A làm việc ở chế độ không có đệm buýt dữ liệu thì đây là tín hiệu vào dùng lập trình để biến mạch 8259A thành mạch thợ ($\overline{SP}=0$) hoặc chủ ($\overline{SP}=1$); khi 8259A làm việc trong hệ vi xử lý ở chế độ có đệm buýt dữ liệu thì chân này là tín hiệu ra $\overline{EN}$ dùng mở đệm buýt dữ liệu để 8086/8088 và 8259A thông vào buýt dữ liệu hệ thống. Lúc này việc định nghĩa mạch 8259A là chủ hoặc thợ phải thực hiện thông qua từ điều khiển đầu ICW4.

+ INT [O]: tín hiệu yêu cầu ngắt đến chân INTR của CPU 8086/8088.

+ $\overline{INTA}$ [I]: nối với tín hiệu báo chấp nhận ngắt $\overline{INTA}$ của CPU.

Hình vẽ dưới đây thể hiện ghép nối 8259A với hệ thống buýt của 8086/8088. Nếu hệ vi xử lý 8086/8088 làm việc ở chế độ MAX thường ta phải dùng mạch điều khiển buýt 8288 và các đệm buýt để cung cấp các tín hiệu thích hợp cho buýt hệ thống. Mạch 8259A phải làm việc ở chế độ có đệm để nối được với buýt hệ thống này.



Hình V-3. Ghép nối 8259 với buýt 8086/8088

V.3.2.3 Lập trình cho PIC 8259A

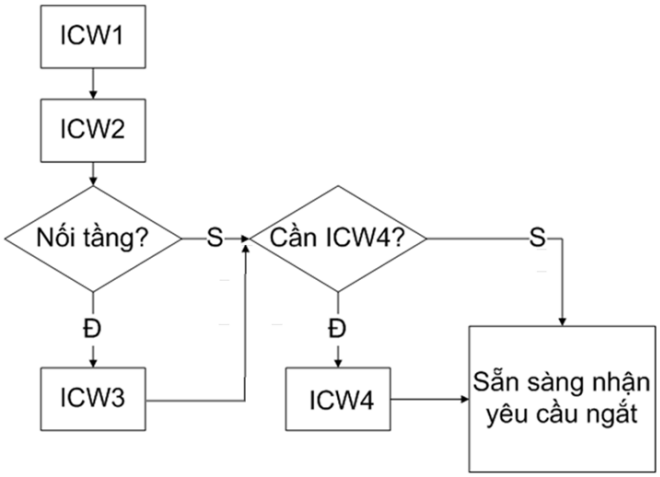
Để mạch PIC 8259A có thể hoạt động được theo yêu cầu, sau khi bật nguồn cấp điện PIC cần phải được lập trình bằng cách ghi vào các thanh ghi (tương đương với các cổng) bên trong *các từ điều khiển khởi đầu (ICW) và tiếp sau đó là các từ điều khiển hoạt động (OCW).*

Các từ điều khiển khởi đầu dùng để tạo nên các kiểu làm việc cơ bản cho PIC, còn các từ điều khiển hoạt động sẽ quyết định cách thức làm việc cụ thể của PIC. Từ điều khiển hoạt động sẽ được ghi khi ta muốn thay đổi hoạt động của PIC.

Các từ điều khiển nói trên sẽ được giới thiệu cụ thể dưới đây.

V.3.2.3.a Các từ điều khiển khởi đầu ICW

PIC 8259A có tất cả 4 từ điều khiển khởi đầu là ICW1 - ICW4. Trong khi lập trình cho PIC không phải lúc nào ta cũng cần dùng cả 4 từ điều khiển khởi đầu những có lúc ta chỉ cần ghi vào đó 2 hay 3 từ là đủ. Hình dưới đây thể hiện thứ tự ghi và điều kiện để ghi các điều khiển ICW vào 8259A.

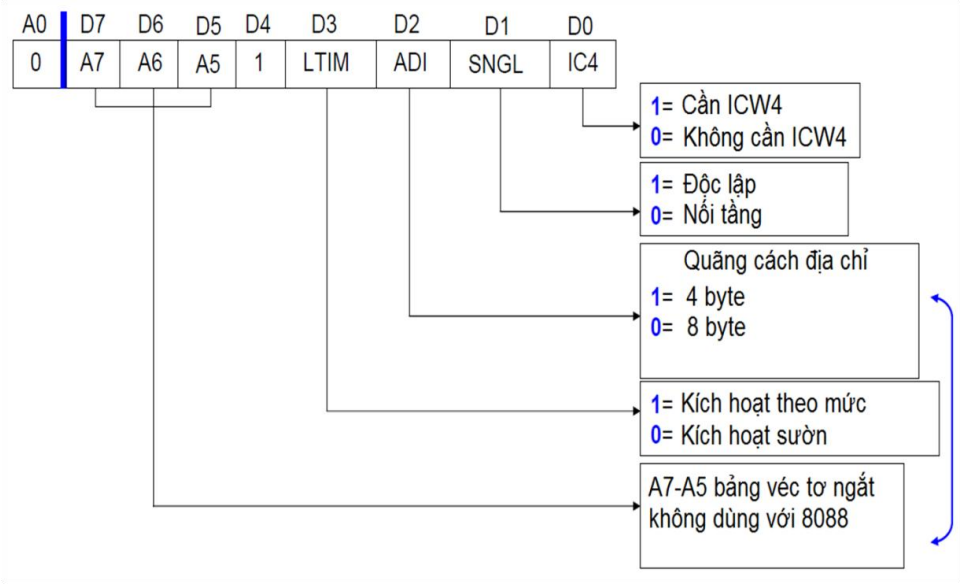


Hình V-4. Trình tự sử dụng các thanh ghi khởi đầu

Để xác định các thanh ghi bên trong ta cần sử dụng tín hiệu địa chỉ A_0 và thứ tự ghi để ghi dữ liệu cho các từ điều khiển. Ví dụ $A_0 = 0$ là dấu hiệu để nhận biết rằng ICW1 được đưa vào thanh ghi có địa chỉ chẵn trong PIC, còn khi $A_0 = 1$ thì các từ điều khiển khởi đầu ICW2, ICW3, ICW4 sẽ được đưa vào các thanh ghi có địa chỉ lẻ trong mạch PIC.

- **ICW1**

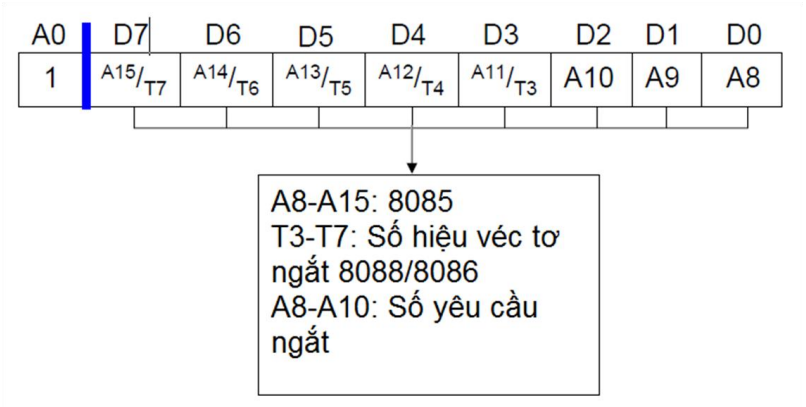
Bít D_0 của ICW1 quyết định 8259A sẽ được nối với họ vi xử lý nào. Để làm việc với hệ 16-32bít (8088 hoặc họ x86) thì ICW nhất thiết phải có ICW4 = 0 (và như vậy các bít của ICW4 sẽ bị xóa về 0). Các bít còn lại của ICW1 định nghĩa cách thức tác động của xung yêu cầu ngắt (tác động theo sườn hay theo mức) tại các chân yêu cầu ngắt IR của mạch 8259A và việc bố trí các mạch 8259A khác trong hệ làm việc đơn lẻ hay theo chế độ nối tầng.



Hình V-5. Dạng thức của ICW1

- **ICW2**

Từ điều khiển khởi đầu này cho phép chọn kiểu ngắt (số hiệu ngắt) ứng với các bít T3-T7 cho các đầu vào yêu cầu ngắt. Các bít T0-T2 được 8259A tự động gán giá trị tùy theo đầu vào yêu cầu ngắt cụ thể IRi. Ví dụ nếu ta muốn các đầu vào của mạch 8259A có kiểu ngắt là 40-47H ta chỉ cần ghi 40H vào các bít T3-T7. Nếu làm như vậy thì IR0 sẽ có kiểu ngắt là 40H, IR1 sẽ có kiểu ngắt là 41H. . .



Hình V-6. Dạng thức của ICW2

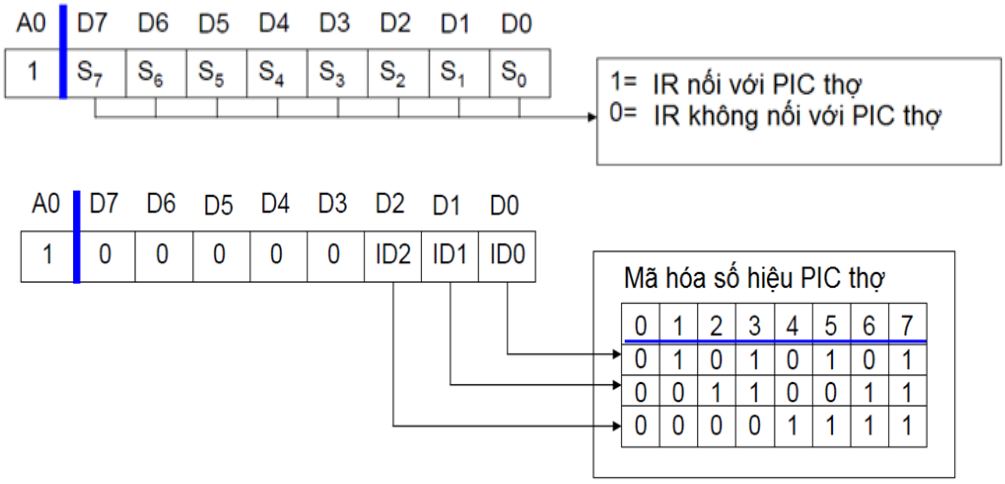
- **ICW3**

Từ điều khiển khởi đầu này chỉ dùng đến khi bít SNGL thuộc từ điều khiển khởi đầu ICW1 có giá trị 0, nghĩa là trong hệ có các mạch 8259A làm việc ở chế độ nối tầng. Chính vì vậy tồn tại 2 loại ICW3: 1 cho mạch 8259A chủ và 1 cho mạch 8259A thợ.

ICW3 cho mạch chủ: dùng để chỉ ra đầu vào yêu cầu ngắt IRi nào của nó có tín hiệu INT của mạch thợ nối vào.

ICW3 cho mạch thợ: dùng làm phương tiện để các mạch này được nhận biết. Vì vậy từ điều khiển khởi đâùu này phải chứa mã số i ứng với đầu vào Iri của mạch chủ mà

mạch thợ đã cho nối vào. Mạch thợ sẽ so sánh mã số này với mã số nhận được ở CAS_2-CAS_0. Nếu bằng nhau thì số hiệu ngắt sẽ được đưa ra buýt khi có INTA.



Hình V-7. Dạng thức của ICW3 cho mạch chủ và thợ

Ví dụ: Trong một hệ vi xử lý ta có một mạch 8259A chủ và 2 mạch 8259A thợ nối vào chân IR1 của mạch chủ. Tìm giá trị phải gán cho các từ điều khiển khởi đầu ICW ?

Giải: Như trên đã nói, để các mạch này làm việc được vói nhau ta sẽ phải gán các từ điều khiển khởi đầu như sau: ICW3 = 03H cho mạch chủ. ICW3 = 00H cho mạch thợo thứ nhất và ICW3 = 01H cho mạch thợ thứ hai.

- **ICW4**

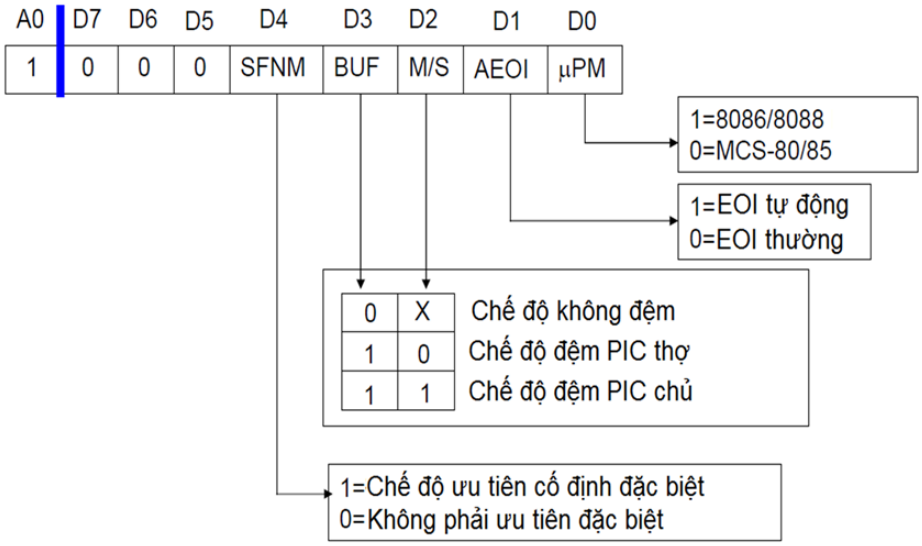
Từ điều khiển khởi đầu này chỉ dùng đến khi trong từ điều khiển ICW1 có IC4 = 1 (cần thêm ICW4)

Bít µPM cho ta khả năng chọn loại vi xử lý để làm việc với 8259A. Bít µPM = 1 cho phép các bộ vi xử lý từ 8086/88 hoặc cao hơn làm việc vưới 8259A

Bít SFNM = 1 cho phép chọn chế độ ưu tiên cố định đặc biệt. Trong chế độ này yêu cầu ngắt với mức ưu tiên cao nhất hiện thời từ một mạch thợ làm việc theo kiểu nối tầng sẽ được mạch chủ nhận biết ngay cả khi mạch chủ còn đang phải phục vụ một yêu cầu ngắt ở mạch thợ khác nhưng với mức ưu tiên thấp hơn. Sau khi các yêu cầu ngắt được phục vụ xong thì chương trình phục vụ ngắt phải có lệnh kết thúc yêu cầu ngắt (EOI) đặt trước lệnh trở về (IRET) đưa đến cho mạch 8259A chủ.

Khi bít SFNM = 0 thì chế độ ưu tiên cố định được chọn (IR0: mức ưu tiên cao nhất. IR7: mức ưu tiên thấp nhất) thực ra đối vưới mạch 8259A không dùng đến ICW1 thì chế độ này đã được chọn như là ngầm định. Trong chế độ ưu tiên cố định tại một thời điểm chỉ có một yêu cầu ngắt i được phục vụ (bít IRi = 1) lúc này ấtt cả các yêu cầu khác vưới ứmc ưu tiên cao hươn có thể ngắt yêu cầu khác với mức ưu tiên thấp hơn.

.



Hình V-8. Dạng thức của ICW4

Bít BUF cho phép định nghĩa mạch 8259A để làm việc với CPU trong trường hợp có đệm hoặc không có đệm nối với buýt hệ thống. Khi làm việc ở chế độ có đệm (BUF = 1). Bít M/S = 1/0 cho phép ta chọn mạch 8259A để làm việc ở chế độ chủ/ thợ. SP/EN trở thành đầu ra cho phép mở đệm để PIC và CPU thông vưới buýt hệ thống.

Bít AEOI = 1 cho phép chọn cách kết thcú yêu cầu ngắt tự động. Khi AEOI = 1 thì 8259A tự động xóa ISRi = 0 khi xung INTA cuối cùng chuyển lên mức cao mà không làm thay đổi thứ tự ưu tiên. Ngược lại. Khi ta chọn cách kết thúc yêu cầu ngắt thường (AEOI = 0) thì chương trình phục vụ ngắt phải có thêm lệnh EOI đặt trước lệnh IRET để kết thúc cho 8259A.

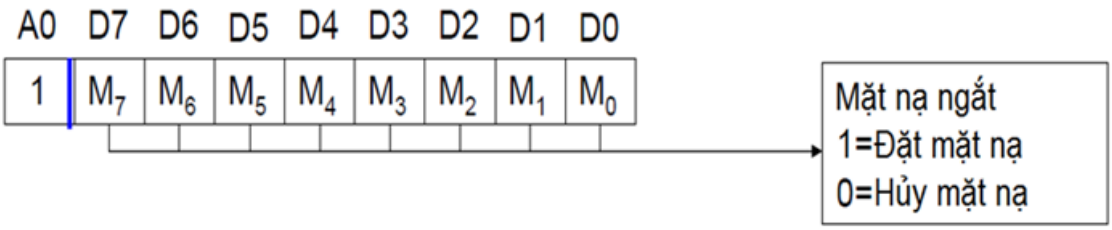
V.3.2.3.b Các từ điều khiển hoạt động OCW

Các từ điều khiển hoạt động OCW sẽ quyết định mạch 8259A sẽ hoạt động như thế nào sau khi nó đã được khởi đầu bằng các từ điều khiển ICW. Tất cả các từ điều khiển này sẽ được ghi vào các thanh ghi trong PIC khi A0 = 0, trừ OCW1 được ghi khi A0 = 1.

- **OCW1**

OCW1 dùng để ghi giá trị của các bít mặt nạ vào thanh ghi mặt nạ ngắt IMR. Khi một bít mặt nạ nào đó của được lập thì yêu cầu ngắt tương ứng với mặt nạ đó sẽ không được 8259A nhận biết nữa (bị che). Từ điều khiển này phải được đưa đến 8259A ngay sau khi ghi các ICW vào 8259A.

Ta cũng có thể đọc lại IMR để xác định tình trạng mặt nạ ngắt hiện tại (xem trong thời điểm hiện tại yêu cầu ngắt nào bị che)



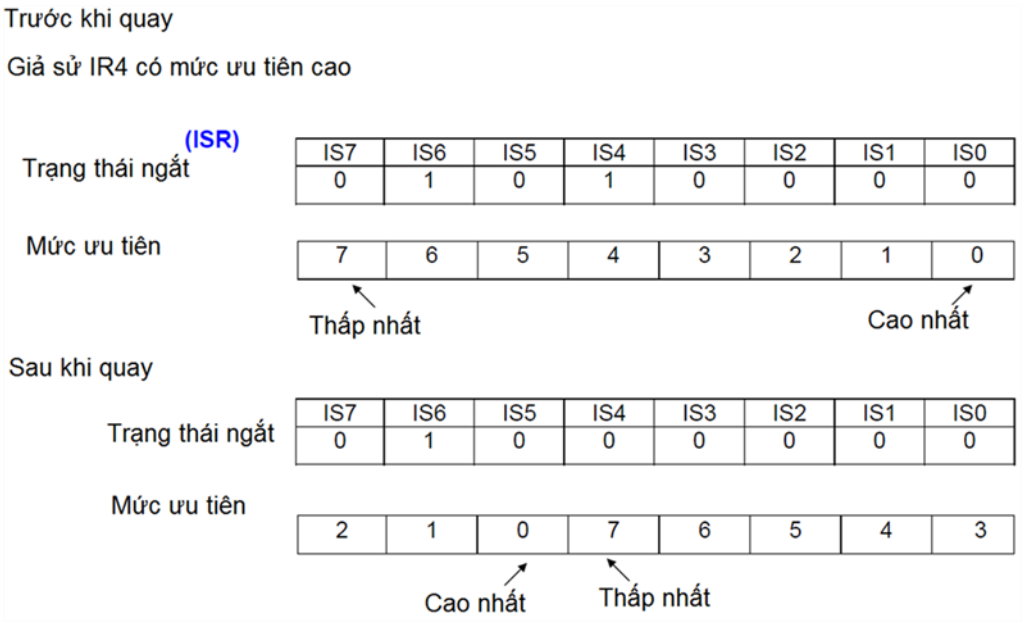
Hình V-9. OCW1 Trạng thái yêu cầu ngắt

OCW2

Các bít R., SL, và EOI phối hợp vưới nhau cho phép chọn ra các cách thức kết thúc yêu cầu ngắt khác nhau. Một vài cách thcú yêu cầu ngắt còn tác động tới các yêu cầu ngắt được chỉ đích danh với mức ưu tiên được giải mã hóa của 3 bít L_2, L_1, L_0.

Trước khi nói về các cách kết thúc yêu cầu ngắt cho các chế độ ta cần mở dấu ngoặc ở đây để giới thiệu các chế độ làm việc của 8259A.

- Chế độ ưu tiên cố định:
  Đây là chế độ làm việc ngầm định của 8259A sau khi nó đã được nạp các từ điều khiển khởi đầu. Trong chế độ này, các đầu vào IR7-IRO được gán cho các mức ưu tiên cố định: IRO được gán cho mức ưu tiên cao nhất còn IR7 mức ưu tiên thấp nhất. Mức ưu tiên này được giữ không thay đổi cho đến khi ghi mạch 8259A bị lập trình khác đi do OCW2.
  Trong chế độ ưu tiên cố định tại một thời điểm chỉ có một yêu cầu ngắt i được phục vụ (bít ISRi = 1) lúc này tất cả ác yêu cầu khác với mức ưu tiên thấp hơn đều bị cấm, tất cả các yêu cầu khác vưới mức ưu tiên thấp hơn đều có thể ngắt yêu cầu khác với mức ưu tiên thấp hơn.
- Chế độ quay mức ưu tiên (ưu tiên luân phiên) tự động:
  Ở chế độ này sau khi một yêu cầu ngắt được phục vụ xong, 8259A sẽ xoá bít tương ứng của nó trong thanh ghi ISR và gán cho đầu vào của nó mức ưu tiên thấp nhất để tạo điều kiện cho các yêu cầu ngắt khác có cơ hội được phục vụ.
- Chế độ quay (đổi) mức ưu tiên chỉ đích danh:
  Ở chế độ này ta cần chỉ rõ (đích danh) đầu vào IRi nào, với i=$L_2L_1L_0$, được gán mức ưu tiên thấp nhất, đầu vào IR_{i+1} sẽ được tự động gán mức ưu tiên cao nhất.

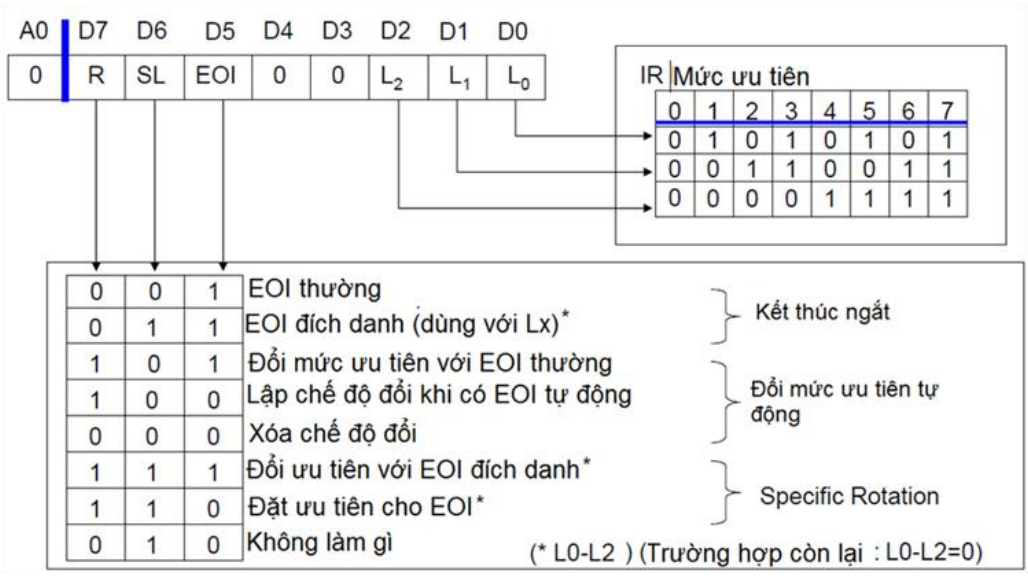


Hình V-10. Trạng thái ngắt và chế độ quay mức ưu tiên

Trở lại các vấn đề liên quan đến OCW, ta sẽ nói rõ việc các bít R, SL và EOI phối hợp với nhau như thế nào để tạo ra các lệnh quy định các cách thức kết thúc yêu cầu ngắt cho các chế độ làm việc khác nhau đã nói đến ở phần trên.

1. Kết thúc yêu cầu ngắt thường: chương trình còn phục vụ ngắt phải có lệnh EOI đặt trước lệnh trở về IRET cho 8259A. Vi mạch này sẽ xác định yêu cầu ngắt IRi vừa được phục vụ và xoá bít ISRi tương ứng của nó để tạo điều kiện cho chính yêu cầu ngắt này hoặc các ngắt khác có mức ưu tiên thấp hơn có thể được tác động.
2. Kết thúc yêu cầu ngắt thường: chương trình con phục vụ ngắt phải có lệnh EOI chỉ đích danh đặt trước lệnh trở về IRET cho 8259A. 8259A xoá đích danh bít ISRi, với i=$L_2L_1L_0$ để tạo điều kiện cho chính yêu cầu ngắt này hoặc các ngắt khác có mức ưu tiên thấp hơn có thể được tác động.
3. Quay (đổi) mức ưu tiên khi kết thúc yêu cầu ngắt thường: chương trình con phục vụ ngắt phải có lệnh EOI đặt trước lệnh trở về IRET cho 8259A. 8259A sẽ xác định yêu cầu ngắt thứ i vừa được phục vụ. Xóa bít ISRi tương ứng và gán luôn mức ưu tiên thấp nhất cho đầu vào IR, này còn đầu vào IR_{i+1} sẽ được gán mức ưu tiên cao nhất.

   Có thể theo dõi cách thức hoạt động của mạch 8259A trong chế độ quay (đổi) mức ưu tiên khi kết thúc yêu cầu ngắt thường thông qua ví dụ minh họa trình bày trên Hình V-10.



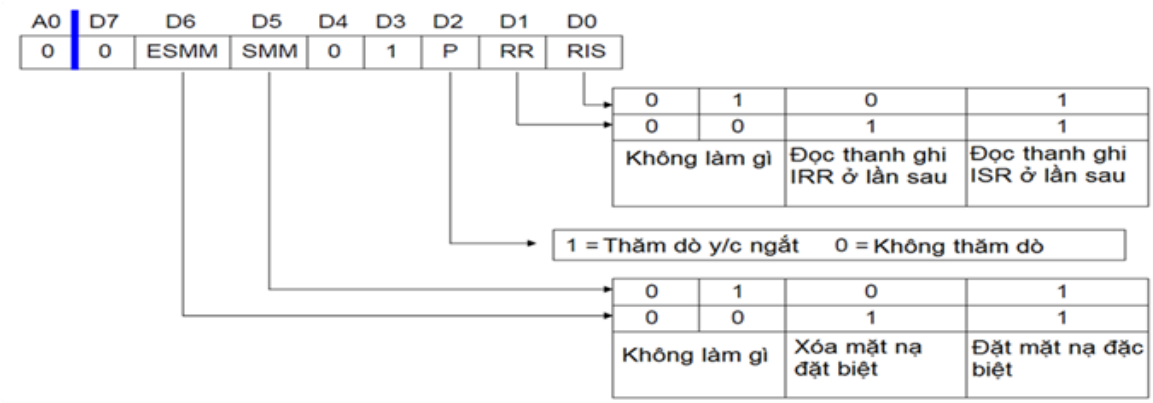
Hình V-11. OCW2 xác định xử lý các yêu cầu ngắt

4. Quay (đổi) mức ưu tiên trong chế độ kết thúc yêu cầu ngắt tự động: chỉ cần một lần đưa lệnh chọn chế độ đổi mức ưu tiên khi kết thúc yêu cầu ngắt tự động. Có thể chọn chế độ này bằng lệnh lập "chế độ quay khi có EOI tự động". Từ đó trở đi 8259A sẽ đổi mức ưu tiên mỗi khi kết thúc ngắt tự động theo cách tương tự như ở mục 3. Muốn bỏ chế độ này ta có thể dùng lệnh xóa "chế độ quay khi có EOI tự động".
5. Quay (đổi) mức ưu tiên khi kết thúc yêu cầu ngắt đích danh: chương trình còn phục vụ ngắt phải có lệnh EOI đích danh cho 8259A đặt trước lệnh trở về IRET. Mạch 8259A sẽ xóa bít ISRi của yêu cầu ngắt tương ứng và gán luôn mức ưu tiên thấp nhất cho đầu vào IRi, với i = L_2 L_1 L_0.
6. Lập mức ưu tiên: chế độ này cho phép thay đổi mức ưu tiên cố định hoặc mức ưu tiên gán trước đó bằng cách gán mức ưu tiên thấp nhất cho yêu cầu ngắt IRi chỉ đích danh với tổ hợp i = L_2 L_1 L_0. Yêu cầu ngắt IR_{i+1} sẽ được gán mức ưu tiên cao nhất.

OCW3

Từ điều khiển hoạt động sau khi được ghi vào 8259A cho phép:

- Chọn các ra thanh ghi để đọc
- Thăm dò trạng thái yêu cầu ngắt bằng cách trạng thái của đầu vào yêu cầu ngắt Iri với mức ưu tiên cao nấht cùng mã của đầu vào đó và.
- Thao tác với mặt nạ đặc biệt.



Hình V-12. OCW3

Các thanh ghi IRR và ISR có thể đọc được sau khi nạp vào 8259A từ điều khiển OCW3 với bít RR = 1: bít RIS = 0 sẽ cho phép đọc IRR. Bít RIS = 1 sẽ cho phép đọc ISR. Dạng thức của các thanh ghi này biểu diễn trên hình dưới đây.

IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
D7	D6	D5	D4	D3	D2	D1	D0

- 0 = Có yêu cầu ngắt
- 1 = Không có yêu cầu ngắt

IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
D7	D6	D5	D4	D3	D2	D1	D0

- 0 = Yêu cầu ngắt IRi không được phục vụ
- 1 = Yêu cầu ngắt IRi đang được phục vụ

Hình V-13. Thanh ghi IRR và ISR

Bằng việc đưa vào 8259A từ điều khiển OCW3 với bít P = 1 ta có thể đọc được trên buýt dữ liệu ở lần đọc tiếp ngay sau đó từ thăm dò, trong đó có các thông tin về yêu cầu ngắt với mức ưu tiên cao nhất đang hoạt động và mã tương ứng với yêu cầu ngắt ấy theo dạng thức được biểu diễn trên dưới đây.

D7	D6	D5	D4	D3	D2	D1	D0
1: có ngắt	X	x	X	x	Số hiệu yêu cầu ngắt		

Hình V-14. Dạng thức từ thăm dò trạng thái

Có thể gọi đây là chế độ thăm dò yêu cầu ngắt và chế độ này thường được ứng dụng trong trường hợp có nhiều chương phục vụ ngắt giống nhau cho một yêu cầu ngắt và việc chọn chương trình nào để sử dụng là trách nhiệm của người lập trình.

Tóm lại, muốn dùng chế độ thăm dò của 8259A để xác định yêu cầu ngắt hiện tại ta cần làm các thao tác lần lượt như sau:

- Cấm các yêu cầu ngắt bằng lệnh CLI
- Ghi từ lệnh OCW3 với bít P = 1
- Đọc từ thăm dò trạng thái yêu cầu ngắt trên buýt dữ liệu.

Bít ESMM = 1 cho phép 8259A thao tác với chế độ mặt nạ đặc biệt. Bít SMM = 1 cho phép lập chế độ mặt nạ đặc biệt. Chế độ mặt nạ đặc biệt được dùng để thay đổi thứ tự ưu tiên ngay bên trong chương trình con phục vụ ngắt. Ví dụ trong trường hợp có một yếu cầu ngắt cấm (bị che bởi chương trình phục vụ ngắt với từ lệnh OCW1 mà ta lại muốn cho phép các yêu cầu ngắt với mức ưu tiên thấp hơn so với yêu cầu ngắt bị cấm đó được tác động thì ta sẽ dùng chế độ mặt nạ đặc biệt. Một khi đã được lập, chế độ mặt nạ đặc biệt sẽ tồn tại cho tới khi bị xóa bằng cách ghi vào 8259A một từ lệnh OCW3 khác vúoi bít SMM = 0. Mặt nạ đặc biệt không ảnh hưởng tới các yêu cầu ngắt vuới mức ưu tiên cao hơn)

V.3.2.3.c Hoạt động của 8086/8088 với 8259A

Cuối cùng để có cái nhìn một cách có hệ thống về hoạt động của hệ vi xử lý với CPU 8086/8088 và PIC 8259A khi có yêu cầu ngắt, ta tóm lượt hoạt động của chúng như sau:

1 Khi có yêu cầu ngắt từ thiết bị ngoại vi tác động vào một trong các chân IR của PIC. 8259A sẽ đưa INT = 1 đến chân INTR của 8086/8088.
2 8086/8088 đưa ra xung INTA đầu đến 8259A
3 8259A dùng xung INTA đầu như là thông báo để nó hoàn tất các xử lý nội bộ cần thiết, kể cả xử lý ưu tiên nếu như có nhiều yêu cầu ngắt cùng xảy ra.
4 8086/8088 đưa ra xung INTA thứ hai đến 8259A
5 Xung INTA thứ hai khiến 8259A đưa ra buýt dữ liệu 1 byte chứa thông tin về số hiệu ngắt của yêu cầu ngắt vừa được nhận biết.
6 8086/8088 dùng số hiệu ngắt để tính ra địa chỉ ngắt của vectơ ngắt tương ứng.
7 8086/8088 cất FR, xóa các cờ IF và TF và cất địa chỉ trở về CS:IP vào ngăn xếp.
8 8086/8088 lấy địa chỉ CS:IP của chương trình phục vụ ngắt từ bảng vectơ ngắt và thực hiện chương trình đó.

V.4 Vào/ra bằng truy nhập trực tiếp bộ nhớ (Direct memory Access)

V.4.1 Khái niệm về phương pháp truy nhập trực tiếp vào bộ nhớ

Trong các cách điều khiển việc trao đổi dữ liệu giữa thiết bị ngoại vi và hệ vi xử lý bằng cách thăm dò trạng thái sẵn sàng của thiết bị ngoại vi hay bằng cách ngắt bộ vi xử lý đã trình bày ở các chương trước, dữ liệu thường được chuyển từ bộ nhớ qua bộ vi xử lý để rồi từ đó ghi vào thiết bị ngoại vi hoặc ngược lại, từ thiết bị ngoại vi nó được đọc vào bộ vi xử lý để rồi từ đó được chuyển đến bộ nhớ. Vì thế tốc độ trao đổi dữ liệu phụ thuộc rất nhiều vào tốc độ thực hiện của các lệnh MOV, IN và OUT của bộ vi xử lý và do đó việc trao đổi dữ liệu không thể tiến hành nhanh đươc.

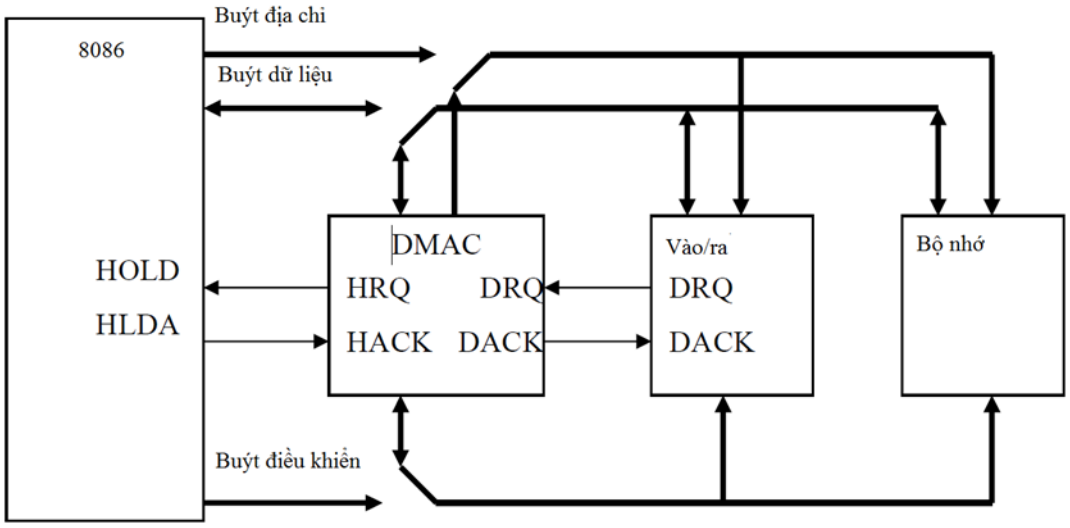
Trong thực tế có những khi ta cần trao đổi dữ liệu thật nhanh với thiết bị ngoại vi: như khi cần đưa dữ liệu hiện thị ra màn hình hoặc trao đổi dữ liệu với bộ điều khiển đĩa. Trong các trường hợp đó ta cần có khả năng ghi /đọc dữ liệu trực tiếp với bộ nhớ thì mới đáp ứng được yêu cầu về tốc độ trao đổi dữ liệu. Để làm được điều này các hệ vi xử lý nói chung đều phải dùng thêm mạch chuyên dụng để điều khiển việc truy nhập trực tiếp vào bộ nhớ DMAC (Direct Memory Access Controller)

Ví dụ dưới đây minh họa điều này. Trong khi một mạch DMAC như 8237A của Inter có thể điều khiển việc chuyển một byte trong một mảng dữ liệu từ bộ nhớ ra thiết bị ngoại vi chỉ hết 4 chu kỳ đồng hồ thì bộ vi xử lý 8086/8088 phải làm hết cỡ 4 chu kỳ:

```
                        ; số chu kỳ đồng hồ
LAP:    MOV AL, (SI )   ;10
        OUT PORT, AL    ;10
        INC SI          ; 2
        LOOP LAP        ; 17
                        ; CỘNG:39 chu kỳ
```

Để hỗ trợ cho việc trao đổi dữ liệu với thiết bị ngoại vi bằng cách truy nhập trực tiếp vào bộ nhớ. CPU thường có tín hiệu yêu cầu treo HOLD để mỗi khi thiết bị cần dùng buýt cho việc trao đổi dữ liệu với bộ nhớ thì thông qua chân này mà báo cho CPU biết. Đến lượt CPU, khi nhận được yêu cầu treo thì nó tự treo lên (tự tách ra khỏi hệ thống bằng cách đưa các bít vào trạng thái trở kháng cao) và đưa xung HLDA ra ngoài để thông báo CPU cho phép sử dụng buýt.

Sơ đồ khối của một hệ vi xử lý có khả năng trao đổi dữ liệu theo kiểu DMA được thể hiện trên hình dưới đây.



Hình V-15. Hệ vi xử lý với DMAC

Ta nhận thấy trong hệ thống này, khi CPU tự tách ra khỏi hệ thống bằng cách tự treo (ứng với vị trí hiện thời của các công tắc chuyển mạch), DMAC phải chịu trách nhiệm điều

khiển toàn bộ hoạt động trao đổi dữ liệu của hệ thống. Như vậy, DMAC phải có khả năng tạo ra được các tín hiệu điều khiển cần thiết giống như các tín hiệu của CPU và bản thân nó phải là một thiết bị lập trình được. Quá trình hoạt động của hệ thống trên có thể được tóm tắt như sau:

Khi thiết bị ngoại vi có yêu cầu trao đổi dữ liệu kiểu DMA với bộ nhớ, nó đưa yêu cầu DREQ=1 đến DMAC, DMAC sẽ đưa yêu cầu treo HRQ=1 đến chân HOLD của CPU. Nhận được yêu cầu treo, CPU sẽ treo các buýt của mình và trả lời chấp nhận treo qua tín hiệu HLDA=1 đến chân HACK của DMAC, DMAC sẽ thông báo cho thiết bị ngoại vi thông qua tín hiệu DACK=1 là nó cho phép thiết bị ngoại vi trao đổi dữ liệu kiểu DMA. khi quá trình DMA kết thúc thì DMAC đưa ra tín hiệu HRQ=0.

V.4.2 Các phương pháp trao đổi dữ liệu

Trong thực tế tồn tại 3 kiểu trao đổi dữ liệu bằng cách truy nhập trực tiếp vào bộ nhớ như sau:

- Treo CPU một khoảng thời gian để trao đổi cả mảng dữ liệu.
- Treo CPU để trao đổi từng byte.
- Tận dụng thời gian không dùng buýt để trao đổi dữ liệu.

V.4.2.1 Trao đổi cả một mảng dữ liệu

Trong chế độ này CPU bị treo trong suốt quá trình trao đổi mảng dữ liệu. Chế độ này được dùng khi ta có nhu cầu trao đổi dữ liệu với ổ đĩa hoặc đưa dữ liệu ra hiển thị. Các bước để chuyển một mảng dữ liệu từ bộ nhớ ra thiết bị ngoại vi:

1. CPU phải ghi từ điều khiển và từ chế độ làm việc vào DMAC để quy định cách thức làm việc, địa chỉ đầu của mảng nhớ, độ dài của mảng nhớ, . . .
2. Khi thiết bị ngoại vi có yêu cầu trao đổi dữ liệu, nó đưa DREQ =1 đến DMAC.
3. DMAC đưa ra tín hiệu HRQ đến chân HOLD của CPU để yêu cầu treo CPU. Tín hiệu HOLD phải ở mức cao cho đến hết quá trình trao đổi dữ liệu.
4. Nhận được yêu cầu treo, CPU kết thúc chu kỳ buýt hiện tại, sau đó nó treo các buýt của mình và đưa ra tín hiệu HLDA báo cho DMAC được toàn quyền sử dụng buýt.
5. DMAC đưa ra xung DACK để báo cho thiết bị ngoại vi biết là có thể bắt đầu trao đổi dữ liệu.
6. DMAC bắt đầu chuyển dữ liệu từ bộ nhớ ra thiết bị ngoại vi bằng cách đưa địa chỉ của byte đầu ra buýt địa chỉ và đưa ra tín hiệu $\overline{\text{M}}$EMR=O để đọc một byte từ bộ nhớ ra buýt dữ liệu. tiếp đó DMAC đưa ra tín $\overline{\text{hiệu}}$ IOW =0 để ghi đưa dữ liệu ra thiết bị ngoại vi. DMAC sau đó giảm bộ đếm số byte còn phải chuyển, cập nhật địa chỉ của byte cần đọc tiếp, và lặp lại cá c động tác trên cho tới khi hết số đếm (TC).
7. Quá trình DMA kết thúc, DMAC cho ra tín hiệu HRQ=0 để báo cho CPU biết để CPU dành lại quyền điều khiển hệ thống.

V.4.2.2 Treo CPU để trao đổi từng byte.

Trong cách trao đổi dữ liệu này CPU không bị treo lâu dài trong một lần nhưng thỉnh thoảng lại bị treo trong khoảng thời gian rất ngắn đủ để trao đổi 1 byte dữ liệu (CPU bị lấy

mất một số chu kỳ đồng hồ). Do bị lấy đi một số chu kỳ đồng hồ như vậy lên tốc độ thực hiện một công việc nào đó của CPU chỉ bị suy giảm chứ không dừng lại. Cách hoạt động cũng tương tự như phần trước, chỉ có điều mỗi lần DMAC yêu cầu treo CPU thi chỉ có một byte được trao đổi.

V.4.2.3 Tận dụng thời gian CPU không dùng buýt để trao đổi dữ liệu.

Trong cách trao đổi dữ liệu này, ta phải có các logic phụ bên ngoài cần thiết để phát hiện ra các chu kỳ xử lý nội bộ của CPU (không dùng đến buýt ngoài) và tận dụng các chu kỳ đó vào việc trao đổi dữ liệu giữa thiết bị ngoại vi với bộ nhớ. Trong cách làm này thì DMAC và CPU luân phiên nhau sử dụng buýt và việc truy nhập trực tiếp bộ nhớ kiểu này không ảnh hưởng gì tới hoạt động bình thường của CPU.

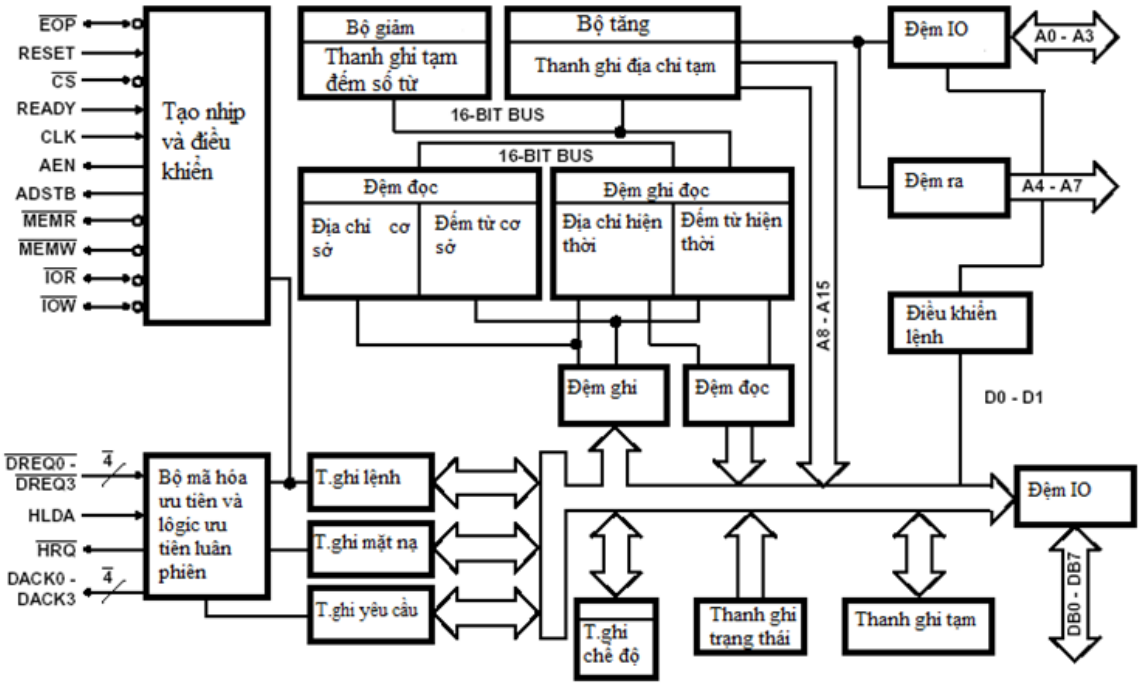
V.4.3 Bộ điều khiển truy nhập trực tiếp vào bộ nhớ Intel 8237A

V.4.3.1 Giới thiệu

DMAC 8237A có thể thực hiện truyền dữ liệu theo 3 kiểu: kiểu đọc (từ bộ nhớ ra thiết bị ngoại vi), kiểu ghi (từ thiết bị ngoại vi đến bộ nhớ) và kiểu kiểm tra.

Trong chế độ truyền kiểu đọc thì dữ liệu được đọc từ bộ nhớ rồi đưa ra thiết bị ngoại vi. Trong chế độ truyền kiểu ghi thì dữ liệu được đọc từ thiết bị ngoại vi rồi đưa vào bộ nhớ. Khi 8237A làm việc ở chế độ kiểm tra thì tuy địa chỉ được đưa đến bộ nhớ nhưng DMAC không tạo ra các xung điều khiển để tiến hành các thao tác ghi/đọc bộ nhớ hay thiết bị ngoại vi.

Ngoài ra mạch 8237A còn hỗ trợ việc trao đổi dữ liệu giữa các vùng khác nhau của bộ nhớ và cũng chỉ riêng trong chế độ làm việc này, dữ liệu cần trao đổi mới phải di qua DMAC nhưng với tốc độ cao hơn khi đi qua CPU nhưng với tốc độ cao hơn khi đi qua CPU (trong trường hợp này ta có thể đọc được dữ liệu đó trong thanh ghi tạm). Sơ đồ khối cấu trúc bên trong của mạch 8237A -5 được thể hiện trên hình dưới đây.



Hình V-16. Sơ đồ khối 8237A

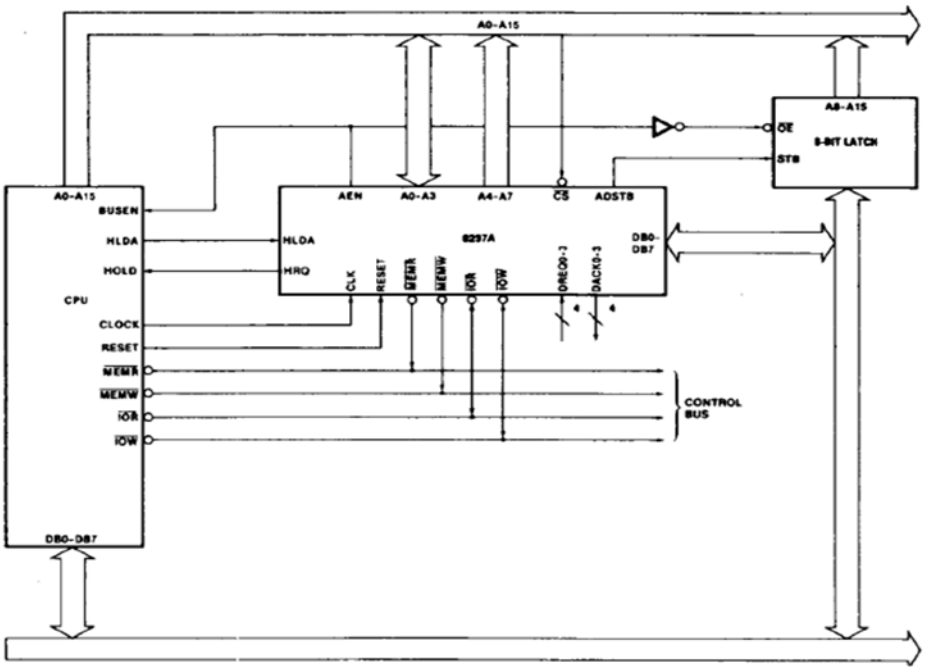
Mạch DMAC 8237A chứa 4 kênh trao đổi dữ liệu DMA với mức ưu tiên lập trình được. DMAC 8237A có tốc độ truyền 1 MB/s cho mỗi kênh, một kênh có thể truyền môt mảng có độ dài 64KB.

V.4.3.2 Các tín hiệu của 8237A -5

- CLK[I]:tín hiệu đồng hồ của mạch. để mạch có thể làm việc tốt với hệ 8086/8088 thì tín hiệu CLK của hệ thống thường được đảo trước khi đưa vào CLK của 8237A.
- CS [I]: tín hiệu chọn vỏ 8237A chân này thường được nối với đầu ra của bộ giải mã địa chỉ. bộ giải mã địa chỉ này không cần dùng đến đầu vào IO/M vì bản thân DMAC đã được cung cấp các xung điều khiển mới của hên thống.
- RESET[I]:tín hiệu nối với tín hiệu khởi động của hệ thống. Khi mạch 8237A được khởi động riêng thanh ghi mặt nạ được lập còn các bộ phận sau bị xóa:
  - Thanhghi lệnh
  - Thanh ghi trạng thái
  - Thanh ghi yêu cầu DMA
  - Thanh ghi tạm thời
  - Mạch lật byte đầu /byte cuối (First/Last)
- READY[I]:tín hiệu sẵn sàng, nối với READY của hệ thống để gây ra các chu kỳ đợi đối với các thiết bị ngoại vi và các bộ nhớ chậm.
- HLDA [I]:tín hiệu báo chấp nhân yêu cầu treo từ CPU
- $DREQ_0$-DREQ3[I]:các tín hiệu yêu cầu treo từ thiết bị ngoại vi. Cực tính của các tín hiệu này có thể lập trình được. Sau khi khởi động các tín hiệunày được định nghĩa là các tín hiệu kích hoạt mức cao.
- DB_0-DB_7[I, O]:tín hiệu hai chiều nối đến buýt địa chỉ và buýt dữ lliệu của hệ thống các tín hiệu này được dùng khi lập trình cho DMAC và khi DMAC hoạt động các chân này chứa 8 bít địa chỉ cao A_8-A_{15} của mảng nhớ dữ liệu cần chuyển. Trong chế độ chuyển dữ liệu giữa các vùng của bộ nhớ tại các chân này có các dữ liệu được chuyển.
- IOR[I, O]VÀ IOW[I, O]: là các chân tín hiệu hai chiều dùng trong khi lập trình cho DMAC và trong các chu kỳ đọc và ghi.
- EOP[I, O]: là tín hiệu hai chiều dùng để yêu cầu DMAC kết thúc quá trình DMA. Khi là đầu ra nó được dùng để báo cho bên ngoài biết một kênh nào đó đã chuyển xong số byte theo yêu cầu, luc này nó thường dùng như một yêu cầu ngắt để CPU xử lý việc kết thúc quá trình DMA.
- A_0-A_3[I, O]:là các tín hiệu hai chiều dùng để chọn các thanh ghi trong 8237A khi lập trình và khi đọc (đầu vào), hoặc để chuyển 4 bít địa chỉ thấp nhất của địa chỉ mảng nhớ cần chuyển (đầu ra).
- A_4-A_7[O]:các chân để chứa 4 bít địa chỉ phần cao trong byte địa chỉ thấp của địa chỉ mảng nhớ cần chuyển.

- HRQ[O]:tín hiệu yêu cầu treo đến CPU. Tín hiệu này thường được đồng bộ với tín hiệu CLK của hệ thống rồi được đưa đến chân HOLD của 8086/8088.
- $DACK_0$-$DACK_3$[0]: là các tín hiệu trả lời các yêu cầu DMA cho các kênh. Các tín hiệu này có thể được lập trình để hoạt động theo mức thấp hoặc mức cao. Sau khi khởi động, các tín hiệu này được định nghĩa là các xung tích cực thấp.
- AEN[0]: tín hiệu cho phép mạch nối vào DB_0-DB_7 chốt lấy địa chỉ của vùng nhớ cần trao đổi theo kiểu DMA. Tín hiệu này cũng cho phép cấm các mạch đệm buýt địa chỉ và dữ liệu hoặc mạch tạo tín hiệu điều khiển của CPU nối vào các buýt tương ứng khi DMAC hoạt động.
- ADSTB[0]: xung cho phép chốt các bít địa chỉ phần cao A8-A15 có mặt trên DB0-DB7.
- MEMR[0] và MEMW[0]: là các chân tín hiệu do DMAC tạo ra và dùng khi đọc/ghi bộ nhớ trong khi hoạt động.

Hình vẽ dưới đây minh họa cách ghép nối các tín hiệu của 8237A với buýt hệ thống.



Hình V-17. Ghép nối 8237 với buýt hệ vi xử lý

V.4.3.3 Các thanh ghi bên trong của DMAC 8237A

Các thanh ghi bên trong DMAC 8237A được CPU 8086/8088 chọn để làm việc nhờ các bít địa chỉ thấp A0-A3. Bảng dưới đây chỉ ra cách thức chọn ra các thanh ghi đó.

Bảng V-1. Địa chỉ các thanh ghi 8237A

Bit địa chỉ				Địa chỉ	Chọn chức năng	R/W?
A3	A2	A1	A0			
0	0	0	0	X0	Thanh ghi địa chỉ bộ nhớ kênh 0	R/W
0	0	0	1	X1	Thanh ghi đếm từ kênh 0	R/W
0	0	1	0	X2	Thanh ghi địa chỉ bộ nhớ kênh 1	R/W
0	0	1	1	X3	Thanh ghi đếm từ kênh 1	R/W
0	1	0	0	X4	Thanh ghi địa chỉ bộ nhớ kênh 2	R/W
0	1	0	1	X5	Thanh ghi đếm từ kênh 2	R/W
0	1	1	0	X6	Thanh ghi địa chỉ bộ nhớ kênh 3	R/W
0	1	1	1	X7	Thanh ghi đếm từ kênh 3	R/W
1	0	0	0	X8	Thanh ghi trạng thái / lệnh	R/W
1	0	0	1	X9	Thanh ghi yêu cầu	W
1	0	1	0	XA	Thanh ghi mặt nạ cho một kênh	W
1	0	1	1	XB	Thanh ghi chế độ	W
1	1	0	0	XC	Xóa flip-flop đầu/cuối	W
1	1	0	1	XD	Xóa toàn bộ các thanh ghi / đọc thanh ghi tạm	W/R
1	1	1	0	XE	Xóa thanh ghi mặt nạ	W
1	1	1	1	XF	Thanh ghi mặt nạ	W

Các bảng dưới đây cho biết các thanh ghi trên theo các quan điểm ứng dụng khác nhau để dễ tra cứu địa chỉ cho chúng khi lập trình với DMAC 8237A.

Bảng V-2. Địa chỉ các thanh ghi trong dùng cho các kênh

Kênh	$\overline{IOR}$	$\overline{IOW}$	A3	A2	A1	A0	Thanh ghi	R/W?
0	1	0	0	0	0	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	0	0	0	Địa chỉ hiện hành	R
	1	0	0	0	0	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	0	0	1	Bộ đếm hiện hành	R
1	1	0	0	0	1	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	0	1	0	Địa chỉ hiện hành	R
	1	0	0	0	1	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	0	1	1	Bộ đếm hiện hành	R
2	1	0	0	1	0	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	1	0	0	Địa chỉ hiện hành	R
	1	0	0	1	0	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	1	0	1	Bộ đếm hiện hành	R
3	1	0	0	1	1	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	1	1	0	Địa chỉ hiện hành	R
	1	0	0	1	1	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	1	1	1	Bộ đếm hiện hành	R

Bảng V-3. Các thanh ghi điều khiển và trạng thái

$\overline{IOR}$	$\overline{IOW}$	A3	A2	A1	A0	Thanh ghi
1	0	1	0	0	0	Ghi thanh ghi lệnh
0	1	1	0	0	0	Đọc thanh ghi trạng thái
1	0	1	0	0	1	Ghi thanh ghi yêu cầu
1	0	1	0	1	0	Ghi thanh ghi mặt nạ
1	0	1	0	1	1	Ghi thanh ghi chế độ
1	0	1	1	0	0	Xóa flip-flop đầu/cuối
1	0	1	1	0	1	Xóa tất cả các thanh ghi nội
0	1	1	1	0	1	
1	0	1	1	1	0	Địa chỉ cơ sở và địa chỉ hiện hành
0	1	1	1	1	0	Địa chỉ hiện hành
1	0	1	1	1	1	Bộ đếm cơ sở và bộ đếm hiện hành
0	1	1	1	1	1	Bộ đếm hiện hành

V.4.3.3.a Thanh ghi địa chỉ hiện thời:

Đây là thanh ghi 16 bít dùng để chứa địa chỉ của vùng nhớ phải chuyển. Mỗi kênh có riêng thanh ghi này để chứa địa chỉ. Khi 1 byte được truyền đi. Các thanh ghi này tự động tăng hay giảm tuỳ theo trước nó được lập trình như thế nào.

V.4.3.3.b Thanh ghi số đếm hiện thời:

Thanh ghi 16 bít này dùng để chứa số byte mà kênh phải truyền(nhiều nhất là 16KB). Mỗi kênh có thanh ghi số byte của mình. Các thanh ghi này được ghi bằng số đếm nhỏ nhất hơn 1 so với số byte thực chuyển.

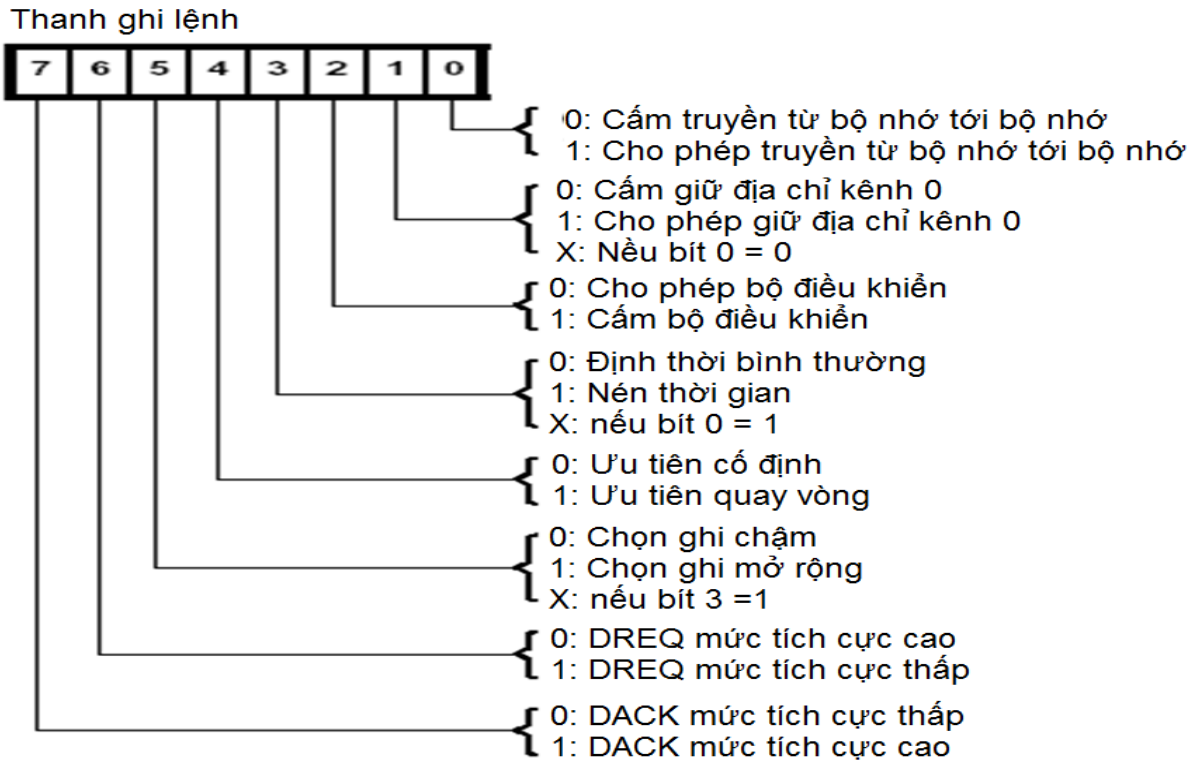
V.4.3.3.c Thanh ghi địa chỉ cơ sở và thanh ghi số đếm cơ sở:

Các thanh ghi này được dùng để chứa địa chỉ và số đếm cho mỗi kênh khi chế độ tự động khởi đầu được sử dụng.

Trong chế độ này một quá trình DMA kết thúc thì các thanh ghi địa chỉ hiện thời và số đếm hiện thời được nạp lại giá trị cũ lấy từ thanh ghi địa chỉ cơ sở và thanh ghi số đếm cơ sở. Khi các thanh ghi địa chỉ hiện thời và số đếm hiện thời được lập trình thì các thanh ghi địa chỉ cơ sở và thanh ghi số đếm cơ sở cũng được lập trình bất kể chế độ tự khởi đầu có được sử dụng hay không.

V.4.3.3.d Thanh ghi lệnh:

Thanh ghi này dùng để lập trình cho DMAC. Nó bị xoá khi khởi động hoặc khi ta sử dụng lệnh xoá toàn bộ các thanh ghi. Dạng thức của thanh ghi lệnh như sau.



Các bít của thanh ghi này quyết định các phương thức làm việc khác nhau của 8237A. Ta sẽ giải thích sau đây ý nghĩa của các bít.

Bít D0 cho phép DMAC dùng kênh 0 và kênh 1 để chuyển dữ liệu giữa 2 vùng nhớ. Địa chỉ của byte dữ liệu ở vùng đích được chứa trong thanh ghi địa chỉ của kênh 1. Số byte chuyển được đặt trong thanh ghi đếm của kênh 1. Byte cần chuyển lúc đầu được đọc từ vùng gốc vào thanh ghi tạm để rồi từ đó nó được gửi đến vùng đích trong bước tiếp theo (hoạt động như lệnh MOVSB nhưng với tốc độ cao).

Bít D1=1 dùng để cho phép kênh 0 giữ nguyên địa chỉ trong chế độ truyền giữ liệu giữa 2 vùng nhớ. Điều này khiến cho toàn bộ các ô nhớ vùng đích được nạp cùng một byte dữ liệu.

Bít D2 cho phép DMAC hoạt động hay không.

Bít D3 quyết định byte cần chuyển được truyền với 4 hay 2 chu kì đồng hồ.

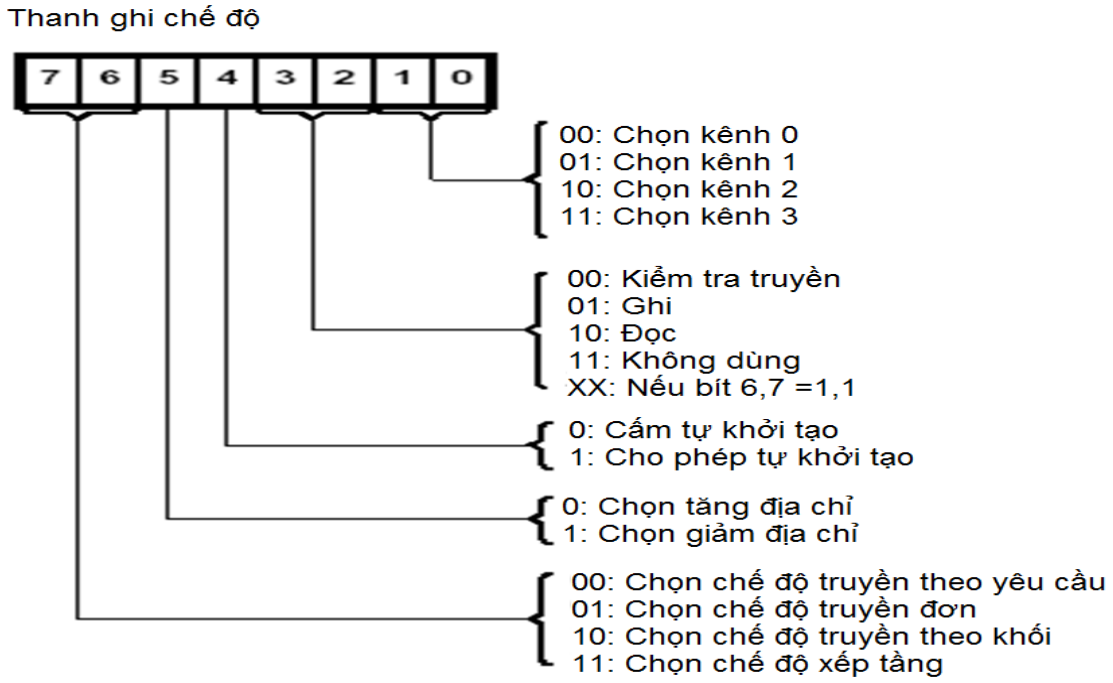
Bít D4 cho phép chọn chế độ ưu tiên cố định (kênh 0 có mức ưu tiên cao nhất. Kênh 3 có mức ưu tiên thấp nhất) hoặc chế độ ưu tiên luân phiên (kênh 0 lúc đầu có mức ưu tiên cao nhất. Sau khi kênh này được chọn để chuyển dữ liệu thì nó được nhận mức ưu tiên thấp nhất. Kênh 1 lại trở thành kênh có mức ưu tiên cao nhất)

Bít D5 cho phép chọn thời gian ghibình thường hay kéo dài cho tiết bị ngoại vi chậm.

Các bít D6 và D7 cho phép chọn cực tính tích cực của các xung DRQ0-DRQ4 và DACK0- DACK4.

V.4.3.3.e Thanh ghi chế độ:

Dùng đặt chế độ làm việc cho các kênh của DMAC. Mỗi kênh của DMAC có một thanh ghi chế độ riêng. Dạng thức của thanh ghi chế độ được biểu diễn như sau:



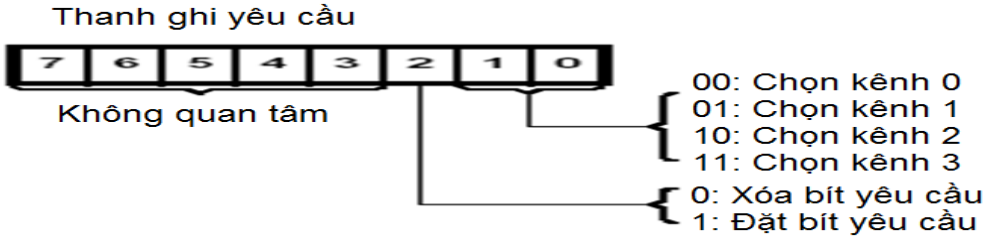
Trong chế độ DMA theo yêu cầu. DMAC tiến hành chuyển dữ liệu cho đến khi có tín hiệu EOP từ bên ngoài hoặc cho đến khi không còn yêu cầu DMA nữa (DREQ trở nên không tích cực)

Trong chế độ DMA chuyển từng byte, chừng nào vẫn còn yêu cầu DMA (DREQ vẫn là tích cực) thì DMAC đưa ra HRQ=0 trong thời gian 1 chu kì buýt sau mỗi lần chuyển sang 1 byte. Sau đó nó lại đưa ra HRQ=1. Cứ như vậy DMAC và CPU luân phiên nhau sử dụng buýt cho đến khi đếm hết (TC).

Trong chế độ DMA chuyển cả mảng, cả một mảng gồm một số byte bằng nội dung bộ đếm được chuyển liền một lúc. Chân yêu cầu chuyển dữ liệu DREQ không cần phải giữ được ở mức tích cực suốt trong quá trình chuyển. Chế độ nối tầng được dùng khi có nhiều bộ DMAC được dùng trong hệ thống để mở rộng số kênh có thể yêu cầu DMA.

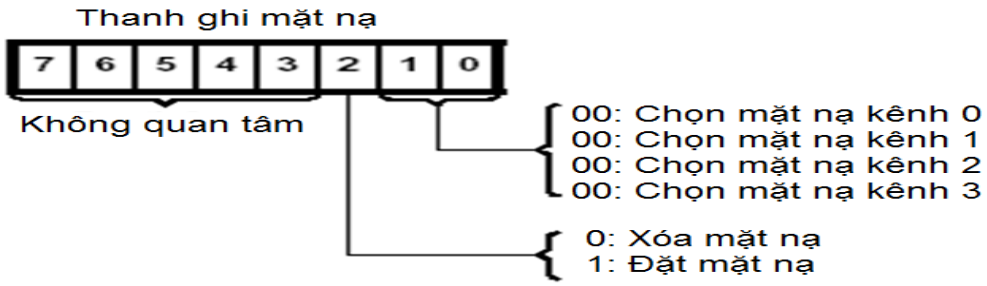
V.4.3.3.f Thanh ghi yêu cầu:

Thanh ghi này dùng để yêu cầu DMA có thể được thiết lập/ xoá theo ý muốn bằng chương trình. Điều này rất có lợi khi ta muốn chuyển dữ liệu giữa các vùng khác nhau của bộ nhớ lúc này các kênh liên quan phải được lập trình ở chế độ chuyển cả mảng. Dạng thức của thanh ghi yêu cầu như sau:



V.4.3.3.g Thanh ghi mặt nạ riêng cho từng kênh:

Bằng thanh ghi này ta có thể lập trình để cấm (cho Bít mặt nạ tương ứng = 1) thay cho phép hoạt động (cho Bít mặt nạ tương ứng = 0) đối với từng kênh một.



V.4.3.3.h Thanh ghi mặt nạ tổng hợp:

Với thanh ghi này ta có thể lập trình để cấm (Bít mặt nạ tương ứng = 1) thay cho phép hoạt động (Bít mặt nạ tương ứng = 0) đối với từng kênh chỉ bằng một lệnh.

V.4.3.3.i Thanh ghi trạng thái:

Thanh ghi này cho phép xác định trạng thái của các kênh trong DMAC. Kênh nào đã truyền xong (đạt số đếm TC), kênh nào đang có yêu cầu DMA để trao đổi dữ liệu. Khi một kênh nào đó đạt TC. Kênh đó sẽ tự động bị cấm. Cấu trúc thanh ghi trạng thái như sau:

D7	D6	D5	D4	D3	D2	D1	D0

D7=1: Kênh 0 có yêu cầu
D6=1: Kênh 1 có yêu cầu
D5=1: Kênh 2 có yêu cầu
D4=1: Kênh 3 có yêu cầu

D0=1: Kênh 0 đạt số đếm
D1=1: Kênh 1 đạt số đếm
D2=1: Kênh 2 đạt số đếm
D3=1: Kênh 3 đạt số đếm

V.4.3.4 Các lệnh đặc biệt cho DMAC 8237A

Có 3 lệnh đặc biệt để điều khiển hoạt động của DMAC 8237A. Các lệnh này chỉ thực hiện bằng các lệnh OUT với các địa chỉ cổng xác định thì theo thanh ghi mà không cần đến giá trị cụ thể của thanh ghi AL.

1. Lệnh xóa mạch lật byte đầu/byte cuối (First/Lát, F/L): F/L là một mạch lật bên trong DMAC bít để chỉ ra byte nào trong các thanh ghi 16bít (thanh ghi địa chỉ hoặc thanh ghi số đếm được chọn làm việc. Nếu F/L=1 thi số đó là MSB, còn nếu F/L=O) thì số đó là LSB. Mạch lật F/L tự động thay đổi trạng thái khi ta ghi /đọc các thanh ghi đó. khi khởi động xong thì F/L=O
2. Lệnh xoá toàn bộ các thanh ghi: lệnh này có tác động như thao tác khởi động. Tất cả các thanh ghi đều bị xoá riêng thanh ghi mặt nạ tổng hợp thì được lập để cấm các yêu cầu trao đổi dữ liệu.
3. Lệnh xoá thanh ghi mặt nạ tổng hợp: Lệnh này cho phép các kênh của DMAC bắt đầu yêu cầu trao đổi dữ liệu.

V.4.3.5 Lập trình cho các thanh ghi địa chỉ và thanh ghi số đếm:

Việc lập trình cho các thanh ghi địa chỉ và thanh ghi số đếm được thực hiện riêng cho mỗi kênh. cần phải định trước giá trị logic của F/L để thao tác chính xác được với LSB và MSB của các thanh ghi trên. ngoài ra còn phải cấm các yêu cầu DMA của các kênh trong khi lập trình cho chúng. Có thể tuân theo các bước sau đây để lập trình cho DMAC 8237A:

+ xoá mặt lật F/L

+cấm các yêu cầu của các kênh

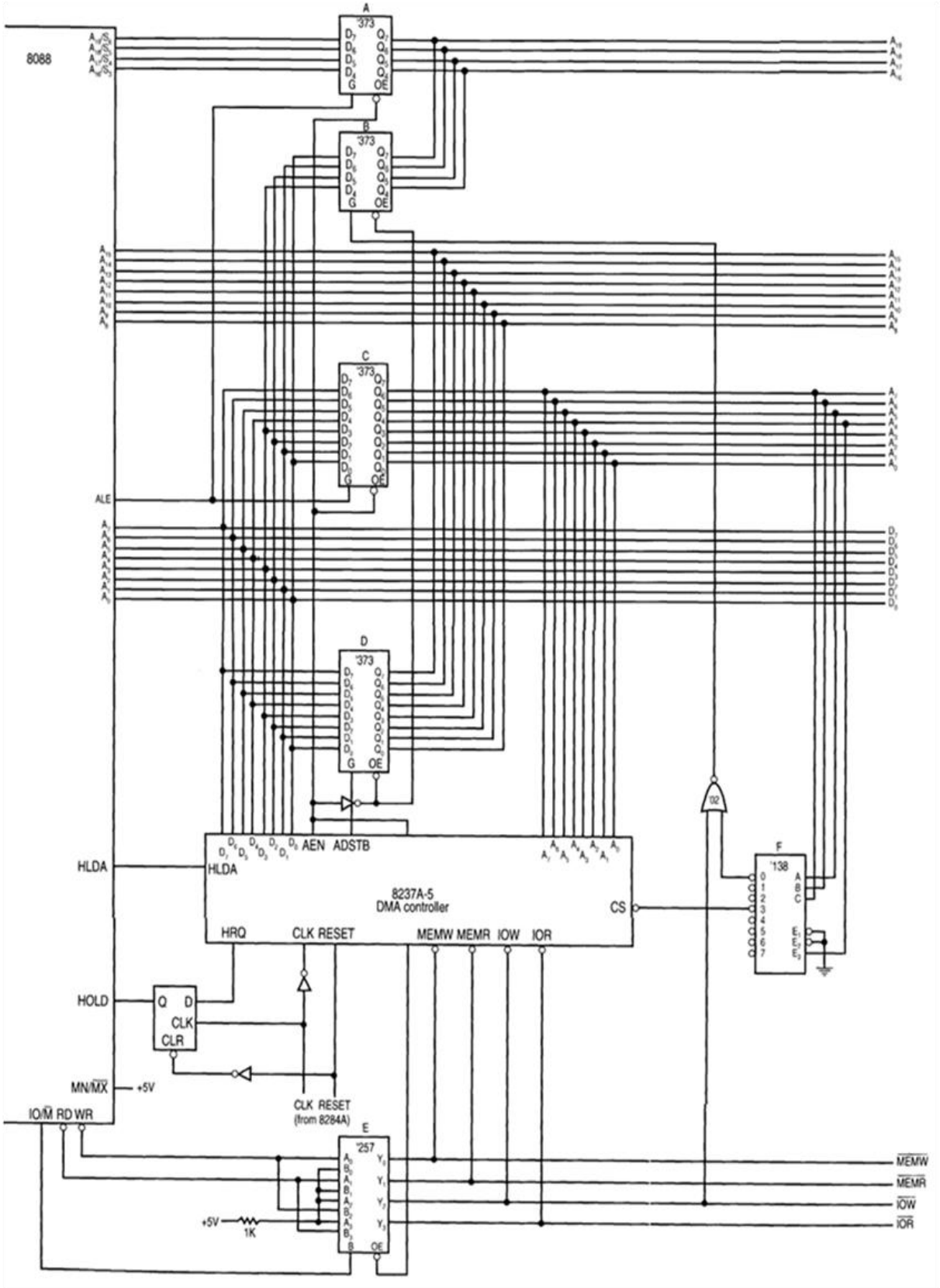
+ghi LSB rồi MSB của thanh ghi địa chỉ

+ghi LSB rồi MSB của thanh ghi số đếm

Dưới đây là một đoạn mã cho 8237A có địa chỉ cơ sở 70H và được ghép với vi xử lý 8088 như trong Hình V-18.

```
ChotB    EQU 010H          ; Địa chỉ mạch chốt B
FL       EQU 07CH          ; Địa chỉ mạch lật
C0       EQU 070H          ; Địa chỉ kênh 0
C1       EQU 072H          ; Địa chỉ kênh 1
Dem_C1   EQU 073H          ; Địa chỉ kênh 0
CheDo    EQU 07BH          ; Địa chỉ thanh ghi chế độ
Lenh     EQU 078H          ; Địa chỉ thanh ghi lệnh
MatNa    EQU 07FH          ; Địa chỉ thanh ghi mặt nạ
YeuCau   EQU 079H          ; Địa chỉ thanh ghi yêu cầu
TThai    EQU 078H          ; Địa chỉ thanh ghi trạng thái
SoByte   DW 0100H          ; Số byte cần chuyển
A16_19   DB 01H            ; 4 bít địa chỉ cao
Nguon    DW 00000H         ; Địa chỉ nguồn
Dich     DW 04000H         ; Địa chỉ đích
;
MOV      AL,A16_19
OUT      ChotB, AL         ; Gửi địa chỉ cao ra mạch chốt
OUT      FL, AL            ; Xóa mạch lật
MOV      AX, Nguon         ;Địa chỉ nguồn ra kênh 0
OUT      C0,AL
MOV      AL, AH
OUT      C0, AL
MOV      AX, Dich          ; Địa chỉ đích ra kênh 1
OUT      C1, AL
MOV      AL, AH
OUT      C1, AL
DEC      SoByte
```

```
        MOV     AX, SoByte
        OUT     Dem_C1, AL          ; số byte cần chuyển vào bộ đếm kênh 1
        MOV     AL, AH
        OUT     Dem_C1, AL
        MOV     AL, 088H            ; Chế độ kênh 0
        OUT     CheDo, AL
        MOV     AL, 085H            ; Chế độ kênh 1
        OUT     CheDo, AL
        MOV     AL,1                ; Chuyển mảng
        OUT     Lenh, AL
        MOV     AL, 0CH             ; Bỏ mặt nạ kênh 0,1
        OUT     MatNa, AL
        MOV     AL,4                ; Kênh 0 yêu cầu DMA
        OUT     YeuCau, AL
LAP:    IN      AL,TThai
        TEST    AL,2                ; Kiểm tra bộ đếm kênh 1 xong?
        JZ      LAP
```



Hình V-18. Ghép nối 8237A với 8088 ở chế độ MIN

Chương VI. Các bộ vi điều khiển

VI.1 Giới thiệu về vi điều khiển và các hệ nhúng

VI.1.1 Giới thiệu

Hệ vi điều khiển là một máy tính trong đó các vi mạch cần thiết được bố trí trên một vi mạch duy nhất. Tất cả các máy tính điều có một số điểm chung như sau:

- Đơn vị xử lý trung tâm (CPU) thực hiện các chương trình
- Bộ nhớ truy nhập ngẫu nhiên RAM chứa dữ liệu thay đổi
- Bộ nhớ chỉ đọc ROM chức các chương trình
- Các thiết bị vào/ra để liên lạc với thế giới bên ngoài như bàn phím, màn hình …

Hệ vi điều khiển có thể được mô tả bằng các đặc trưng khác. Nếu một máy tính có các đặc điểm chung như thế thì chúng có thể coi như là hệ vi điều khiển. Hệ vi điều khiển có thể:

- được nhúng bên trong các thiết bị khác (thường là các sản phẩm tiêu dùng) để kiểm soát các chức năng hay hoạt động của sản phẩm. Hệ vi điều khiển cũng được coi như bộ điều khiển nhúng;
- chỉ dùng cho một nhiệm vụ và chạy một chương trình xác định. Chương trình này thường được lưu trong ROM và không thay đổi;
- là thiết bị tiêu thụ điện thấp. Bộ vi điều khiển sử dụng pin có thể tiêu thụ chỉ 50 mA.

Bộ vi điều khiển có thể nhận đầu vào từ thiết bị và điều khiển thiết bị này bằng cách gửi các tín hiệu tới các bộ phận khác nhau trong thiết bị được điều khiển. Bộ vi điều khiển thường nhỏ và chi phí thấp. Bộ xử lý được dùng trong một bộ vi điều khiển có thể thay đổi rất nhiều. Trong nhiều sản phẩm như lò vi sóng, yêu cầu về CPU khá thấp và sức ép về giá thành lại lớn nên các nhà sản xuất lựa chọn các vi mạch vi điều khiển chuyên dụng. Đó là các thiết bị CPU nhúng, giá rẻ, tiêu thụ điện thấp. Các vi mạch Motorola 6811 và Intel 8051 là các ví dụ tiêu biểu. Các vi mạch vi điều khiển cấp thấp thường có sẵn 1KB ROM và 20 B RAM trên vi mạch cùng với 8 tín hiệu vào/ra.

VI.1.2 Các kiểu vi điều khiển

Họ vi điều khiển chủ yếu là 8 bít do kích cỡ từ này rất phổ biến với phần lớn các công việc mà các thiết bị này cần phải thực hiện. Độ dài từ 8 bít được coi là đủ cho hầu hết các ứng dụng và có lợi thế giao tiếp với các vi mạch nhớ cũng như lô-gíc hiện có. Cấu trúc dữ liệu ASCII nối tiếp cũng được bố trí theo byte nên việc truyền thông với các thiết bị vi điều khiển dễ dàng tương thích và thuận tiện. Do các dạng ứng dụng với vi điều khiển có thể thay đổi rất lớn, hầu hết các nhà sản xuất cung cấp họ các thiết bị vi điều khiển mà khả năng mỗi thành

viên phù hợp với yêu cầu chế tạo. Điều này tránh tình trạng thiết bị vi điều khiển quá phức tạp và tốn kém để đáp ứng tất cả các dạng ứng dụng, đồng thời, hạn chế việc một số phần của vi điều khiển hoàn toàn không được sử dụng khi chạy ứng dụng. Họ vi điều khiển sẽ có tập lệnh con chung, tuy nhiên các thành viên trong họ có thể khác nhau về số lượng, kiểu, bộ nhớ, các cổng v. v. Như vậy nhà sản xuất có thể chế tạo các thiết bị với chi phí hiệu quả phù hợp với các yêu cầu sản xuất cụ thể.

Việc mở rộng bộ nhớ có thể sử dụng các vi mạch ROM/RAM bên ngoài vi điều khiển. Mộ số vi điều khiển không tích hợp sẵn ROM cũng như EPROM hay EEROM. Một số chức năng bổ sung khác có thể được tích hợp vào vi mạch của bộ vi điều khiển như chuyển đổi tương tự số (Analogue-to-Digital Converter ADC). Một số vi điều khiển khác có số lượng tín hiệu ít hơn để giảm thiểu chi phí. Bảng dưới đây liệt kê đặc tính của một số vi điều khiển.

Bảng VI-1. Đặc tính một số vi điều khiển

Mô-đen	Tín hiệu: Vào/ra	RAM (byte)	ROM (Byte)	Độ rộng từ (bít)	Tính năng khác
Intel 8051	40:32	64	1K	8	Bộ nhớ mở rộng 8K
Motorola 68HC11	52:40	256	8K	8	Cổng nối tiếp ; chuyển đổi tương tự số
Zilog Z8820	44:40	272	8K	8	Bộ nhớ mở rộng 128K ; cổng nối tiếp
Intel 8096	68:40	232	8K	16	Bộ nhớ mở rộng 64K ; chuyển đổi tương tự số ;cổng nối tiếp; điều biến xung

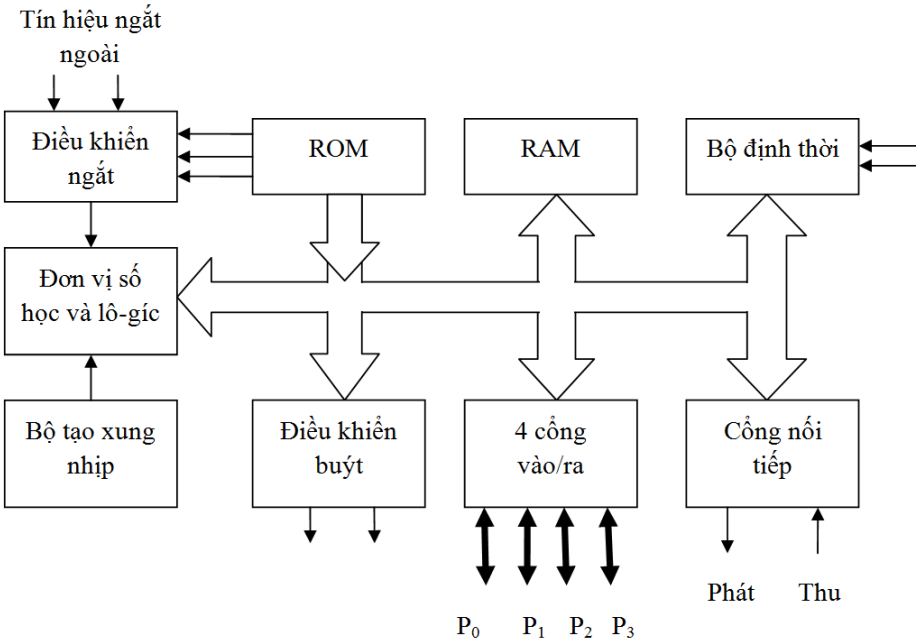
VI.2 Họ vi điều khiển Intel 8051

Vi điều khiển 8051 lần đầu tiên được Intel giới thiệu vào năm 1981. Đây là bộ vi điều khiển 8 bít với 128 byte RAM và 4KB ROM, một cổng nối tiếp và 4 cổng 8 bít trên một vi mạch đơn lẻ. Dòng vi điều khiển này trở nên phổ biến sau khi Intel cho phép các nhà sản xuất khác được chế tạo vi điều khiển tương thích với 8051. Đến nay vi điều khiển 8051 thực ra bao gồm họ vi điều khiển ký hiệu từ 8031 tới 8751 được sản xuất bằng công nghệ NMOS và CMOS với nhiều kiểu đóng gói khác nhau. Phiên bản nâng cao của 8051 là 8052 cũng có các biến thể khác nhau. Các biến thể này nhằm đắp ứng các yêu cầu ứng dụng khác nhau của các nhà phát triển.

Bảng VI-2. Thông số của một số vi điều khiển họ 8051

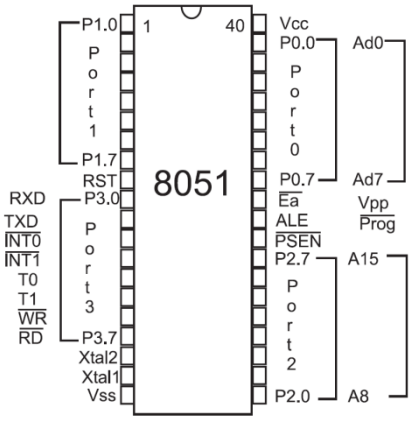
Tính năng	8051	8052	8031
ROM	4K	8K	-
RAM (Byte)	128	256	128
Bộ định thời	2	3	2
Tín hiệu vào/ra	32	32	32
Cổng nối tiếp	1	1	1
Nguồn ngắt	6	8	6

VI.2.1 Sơ đồ khối



Hình VI-1. Sơ đồ khối 8051

Hình VI-1 cho thấy khối chức năng đặc trưng cho vi điều khiển đó là: ROM và RAM, các cổng vào/ra, bộ định thời và kênh thông tin nối tiếp. Hình VI-2 cho biết sơ đồ tín hiệu của 8051, ý nghĩa các tín hiệu được giải thích trong Bảng VI-3.



Hình VI-2. Sơ đồ chân tín hiệu 8051

8051 hỗ trợ 4 cổng vào/ra trong một số biến thể các cổng này đều có thể hoạt động ở cả hai chế độ vào và ra. Cổng truyền thông nối tiếp thường xử lý dữ liệu 8 bít cho phép gửi và nhận song song. Cổng nối tiếp có 4 chế độ hoạt động. Chế độ 0, chân TxD sử dụng như tín hiệu xung nhịp cố định ở mức 1/12 xung nhịp của vi điều khiển còn chân RxD dùng để thu và phát. Chế độ 1 là chế độ giao tiếp UART với 1 bít stop. Chế độ 2 giống chế độ 1 nhưng thêm bít chẵn lẻ. Chế độ 3 giống chế độ 2 nhưng cho phép lập trình tốc độ tín hiệu.

Bộ nhớ ROM trong vi mạch có thể là loại EPROM lập trình bằng điện. Bộ nhớ ngoài có thể truy nhập thông qua tín hiệu truy nhập EA=0. Việc truy nhập ROM ngoài được thực hiện thông qua tín hiệu PSEN ở mức thấp để kích hoạt vi mạch nhớ ROM.

Bảng VI-3. Ý nghĩa tín hiệu 8051

Tín hiệu	Ý nghĩa
P0. 0-P0. 7	Tín hiệu dữ liệu cổng P0
P1. 0-P1. 7	Tín hiệu dữ liệu cổng P1
P2. 0-P2. 7	Tín hiệu dữ liệu cổng P2
P3. 0-P3. 7	Tín hiệu dữ liệu cổng P3
A8-A15	Tín hiệu địa chỉ
Xtal1-2	Tín hiệu xung nhịp
RxD	Tín hiệu thu truyền thông nối tiếp
TxD	Tín hiệu phát truyền thông nối tiếp
INT0-1	Tín hiệu ngắt 0-1 (mức thấp)
RD	Đọc dữ liệu bộ nhớ ngoài
WR	Tín hiệu ghi dữ liệu bộ nhớ ngoài
EA	Tín hiệu truy nhập bộ nhớ chương trình ngoài EA=0 dùng ROM ngoài EA=1 dùng ROM trong
ALE	Tín hiệu chốt địa chỉ trên P0 ALE=1 Trên nhóm cổng P0 là tín hiệu địa chỉ ALE=0 Trên nhóm cổng P0 là tín hiệu dữ liệu
PSEN	Tín hiệu cho phép lưu chương trình dùng đọc bộ nhớ chương trình bên ngoài
RST	Khởi động lại

Các tín hiệu ngắt của 8051 có thể chia thành 2 loại bên trong và bên ngoài khởi xướng. Khi ngắt diễn ra, chương trình đang chạy sẽ bị dừng và chương trình phục vụ ngắt được kích hoạt. Khi kết thúc, vi điều khiển sẽ quay trở lại chương trình bị dừng như chưa có gì xảy ra. Ngắt xảy ra đồng thời được xử lý theo độ ưu tiên.

Bộ định thời hay bộ đếm là chuỗi mạch lật thay đổi trạng thái theo từng tín hiệu vào/ra. Hai bộ đếm T0, T1 có thể được lập trình chia 256, 8192 hay 65536 và sinh ra các tín hiệu ngắt khi kết thúc. Tín hiệu này có thể được phát hiện thông qua phần mềm.

VI.2.2 Các thanh ghi

Thanh ghi đếm chương trình (PC) và con trỏ dữ liệu (DPTR) là các thanh ghi 16 bít cho phép xác định vị trí 1 ô nhớ. Bộ nhớ chương trình nằm trong dải 0000-FFFFh trong đó 0000-0FFFh là không gian nhớ chương trình bên trong vi điều khiển. Con trỏ dữ liệu chia thành hai phần thấp (8 bít) và cao (8 bít)

Thanh ghi A và B là các thanh ghi dùng chung dùng cho các thao tác tính toán của đơn vị xử lý của 8051. Thanh ghi A là thanh ghi gộp (accumulator) dùng trong các thao tác số học và lô-gíc. Thanh ghi này cũng dùng để trao đổi dữ liệu với bộ nhớ ngoài. Thanh ghi B thường

dùng kèm với thanh ghi A trong các thao tác nhân chia. Ngoài ra, 8051 còn 32 thanh ghi khác nằm trong bộ nhớ RAM trong chia thành bốn băng, B0-B3, gồm 8 thanh ghi R0-R7.

Cờ là các thanh ghi 1 bít cho biết trạng thái của một số lệnh và được gộp vào thanh ghi từ trạng thái chương trình (Program Status Word PSW). 8051 có các cờ nhớ C, phụ AC, tràn OV và chẵn lẻ P. Các cờ người dùng F0 và GF0-1. Các cờ người dùng có thể tùy biến theo yêu cầu người viết chương trình như lưu các sự kiện.

Con trỏ ngăn xếp SP là thanh ghi 8 bít lưu vị trí đỉnh ngăn xếp trong bộ nhớ RAM trong của 8051.

Các thanh ghi chức năng đặc biệt nằm trong bộ nhớ RAM trong từ địa chỉ 00-7Fh. Các thanh ghi này có thể được đặt tên riêng trong một mã lệnh và tham chiếu qua địa chỉ. Ví dụ thanh ghi A còn được tham chiếu qua địa chỉ 0E0h.

VI.2.3 Tập lệnh

8051 hỗ trợ các chế độ địa chỉ sau:

1. Chế độ địa chỉ trực tiếp: dữ liệu dành cho lệnh là một phần trong mã lệnh. Từ gợi nhớ cho chế độ này là dấu #. Ví dụ MOV A, #100.
2. Chế độ địa chỉ thanh ghi: thanh ghi lưu giá trị dữ liệu.
3. Chế độ địa chỉ trực tiếp: địa chỉ ô nhớ là một phần của câu lệnh
4. Chế độ địa chỉ gián tiếp: giá trị thanh ghi cho biết địa chỉ của dữ liệu. Từ gợi nhớ là @. Ví dụ MOV A, @R0 ; Nạp dữ liệu tại ô nhớ có giá trị R0 vào thanh ghi A.

Tập lệnh 8051 hỗ trợ các thao tác di chuyển dữ liệu, các thao tác lô-gíc, các phép toán số học và các câu lệnh nhảy và gọi hàm.

Ví dụ VI-1

Đoạn chương trình 8051

```
Nhan: INC 3Ch          ; Tăng giá trị ô nhớ 3Ch lên 1
      MOV A, #2Ah      ; A=2Ah
      XRL A, 3Ch       ; XOR A với giá trị tại ô nhớ 3Ch
      JNZ Nhan         ; Nhảy tới Nhan nếu kết quả XOR khác 0
      NOP              ;không làm gì cả
```

VI.3 Giới thiệu một số ứng dụng tiêu biểu của vi điều khiển

Việc chuyển đổi tín hiệu tương tự sang số và ngược lại thường gặp khi ta muốn kết nối máy tính với thế giới tương tự. Trong phần này giới thiệu sử dụng bộ vi điều khiển kết nối với bộ chuyển đổi tương tự số (A/D) và ngược lại (D/A). Thông thường các bộ chuyển đổi cho phép kết nối thông qua kênh dữ liệu 8 bít, ba trạng thái và cho phép điều khiển thôngqua các tín hiệu đọc/ghi, chọn chíp.

VI.3.1 Chuyển đổi số tương tự (D/A)

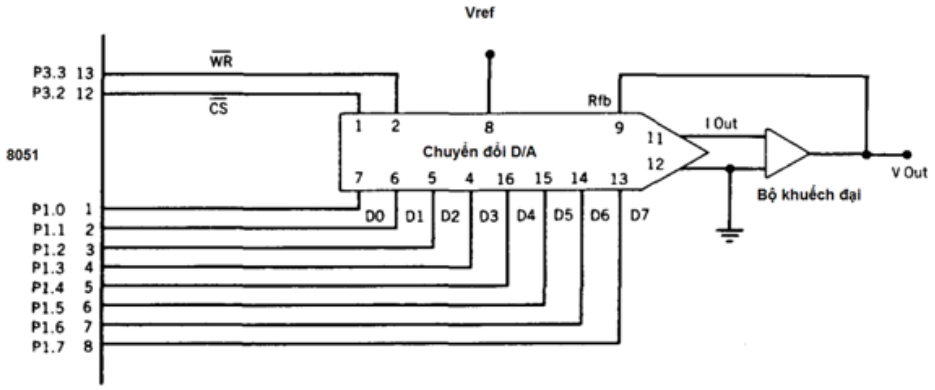
Hình VI-3 giới thiệu kết nối giữa vi điều khiển 8051 và bộ chuyển đổi D/A khái quát. Bộ chuyển đổi D/A có đặc điểm sau:

$V_{out} = -V_{ref} \times$(byte đầu vào/100H) và $V_{ref} = \pm 10V$

Thời gian chuyển đổi 5µs

Trình tự điều khiển ~CS rồi ~WR.

Cổng 1 được nối với các tín hiệu dữ liệu của bộ chuyển đổi còn cổng 3 dùng để điều khiển. Trong ví dụ này, thiết bị tạo ra sóng hình sin với chu kỳ 1000Hz và có thể thay đổi theo chương trình. V_{ref} đặt bằng -10V dạng tín hiệu đầu ra thay đổi từ 0V tới +9, 96V. Chương trình dùng bảng tra cứu để sinh ra biên độ sóng sin. Chu kỳ được thiết lập căn cứ vào khoảng thời gian truyền dữ liệu cho bộ chuyển đổi. Với *S* điểm lấy mẫu, chu kỳ ngắt nhất T_{min}= 5×S µs tần số tối đa f_{max} = 200. 000/S.



Hình VI-3. Ghép nối bộ chuyển đổi D/A với 8051

Với sóng có tần số 1000Hz, cần số lượng mẫu là 200. Tuy nhiên thực tế chạy chương trình cho thấy thời gian để tạo ra một mẫu cần 6µs và thời gian để chuyển sang mẫu kế tiếp mất hơn 2µs. Như vậy thực tế chỉ cho phép số lượng mẫu là 166.

Ví dụ VI-2. Chương trình chuyển đổi D/A

```
            . org 0000h
daconv:     clr p3, 2           ; Chọn chíp
            mov dptr, #bang     : lấy địa chỉ cơ sở bảng
repeat:     mov r1, #0A6h       ; Khởi tạo R1 = 166
next:       mov a, r1           ; Lấy địa chỉ offset của bảng
            movc a, @a+dptr     ;Lấy giá trị mẫu
            mov p1, a           ; Gửi mẫu ra cổng 1
            clr p3, 3
            setb p3, 3
            djnz r1, next
            sjmp repeat
```

; Bảng chuyển đổi sử dụng hàm cosin để tính giá trị biên bộ của tín hiệu tại đầu ra. 83 giá

```
;trị đầu thể hiện biên độ từ cực đại tới nhỏ hơn 0, 83 giá trị còn lại từ 0 tới cực đại. Với 83
; mẫu cho nửa chu kỳ giá trị góc của hàm cosin thay đổi 2, 17 độ cho các mẫu kế tiếp.
bang:   . db 00h          ;
        . db ffh          ; s1:FF×cos(0)
        . db feh          ; s2:7FH+FF×cos(2, 17)
        . db feh          ; s3:7FH+FF×cos(2, 17×2)
        . db 81h          ; s42:7FH+FF×cos(88, 9)
        . . . . . .
        . db 00h          ; s84:7FH+FF×cos(180)
        . . . . . .
        . db feh          ; s166:7FH+FF×cos(2, 17)
```

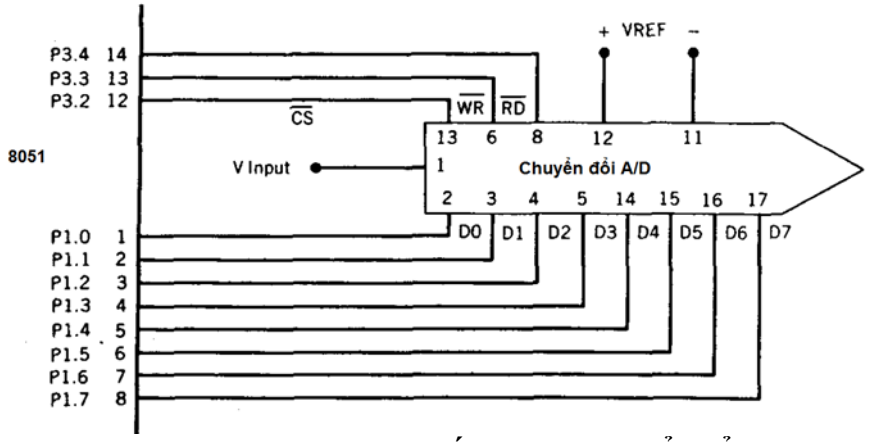
VI.3.2 Chuyển đổi tương tự số (A/D)

Hình VI-4 sử dụng bộ chuyển đổi tương tự số 8 bít có các đặc tính sau:

Tín hiệu lấy mẫu: Vin = Vref-, dữ liệu =00h ; Vin=Vref+, dữ liệu = FFh

Thời gian lấy mẫu: 1 μs

Trình tự điều khiển: CS, WR rồi RD (ở mức tích cực thấp). Trong hình vẽ, cổng 1 của 8051 nối với kênh dữ liệu của bộ chuyển đổi còn cổng 3 nối với các tín hiệu điều khiển.



Hình VI-4. Ghép nối 8051 và chuyển đổi A/D

Ví dụ VI-3. Chương trình chuyển đổi A/D

Đoạn chương trình sau số hóa các tín hiện Vref với chu kỳ 100μs và lưu kết quả vào trong bộ nhớ RAM 4000h:43E7h.

```
        . equ begin, 4000h   ;Địa chỉ bắt đầu
        . equ delay, 74h     ;trễ 87μs
        . equ end1, 43h      ;Địa chỉ kết thúc byte cao
        . equ end2, e8h      ;Địa chỉ kết thúc byte thấp
adconv: mov dptr, #begin
        clr p3, 2            ; Gửi ~CS tới bộ A/D
next:   clr p3, 3            ; Tạo xung ~WR tới bộ A/D
        setb p3, 3           ;
        clr p3, 4            ;Tạo xung ~RD
```

```
        mov a, p1           ;Đọc dữ liệu từ A/D
        setb p3, 4          ;Kết thúc đọc
        mov @dptr, a        ;Lưu vào RAM
        inc dptr            ;Tăng con trỏ RAM lên 1
        mov a, dph          ;Kiểm tra kết thúc
        cjne a, #end1, wait
        mov a, dpl
        cjne a, #end2, wait
        sjmp done           ; Kết thúc khi tới vị trí cuối cùng
wait:   mov r1, #delay      ;Trễ 87μs
here:   djnz r1, here
        sjmp next
done:   sjmp done
        . end
```

Chương VII. Giới thiệu một số vi xử lý tiên tiến

VII.1 Các vi xử lý tiên tiến dựa trên kiến trúc Intel IA-32

VII.1.1 Giới thiệu IA-32

IA-32 là kiến trúc 32 bít do hãng sản xuất Intel phát triển lần đầu tiên được giới thiệu trên bộ vi xử lý Intel386. Kiến trúc IA-32 hỗ trợ ba chế độ hoạt động: chế độ bảo vệ (protected mode), chế độ thực (real mode) và chế độ quản lý hệ thống SMM (System Management Mode). Các chế độ hoạt động quyết định các lệnh và các chức năng mà chương trình có thể truy nhập:

- Chế độ bảo vệ: là chế độ căn bản của bộ xử lý. Chế độ này cho phép chạy các phần mềm 8086 trong môi trường đa nhiệm và bảo vệ. Chế độ này còn được gọi là chế độ 8086 ảo.
- Chế độ địa chỉ thực: Ché độ này cung cấp môi trường lập trình 8086 với một số tính năng mở rộng như chuyển sang chế độ bảo vệ. Để bộ xử lý hoạt động ở chế độ này thông thường phải khởi động lại bộ xử lý.
- Chế độ quản lý hệ thống - SMM: Chế độ này cung cấp cho hệ điều hành các cơ chế trong suốt phục vụ nhiệm vụ cụ thể như quản lý năng lượng hay bảo mật hệ thống. Chế độ này được kích hoạt thông qua tín hiệu SMM hoặc tín hiệu này nhận được từ bộ điều khiển ngắt tiên tiến.

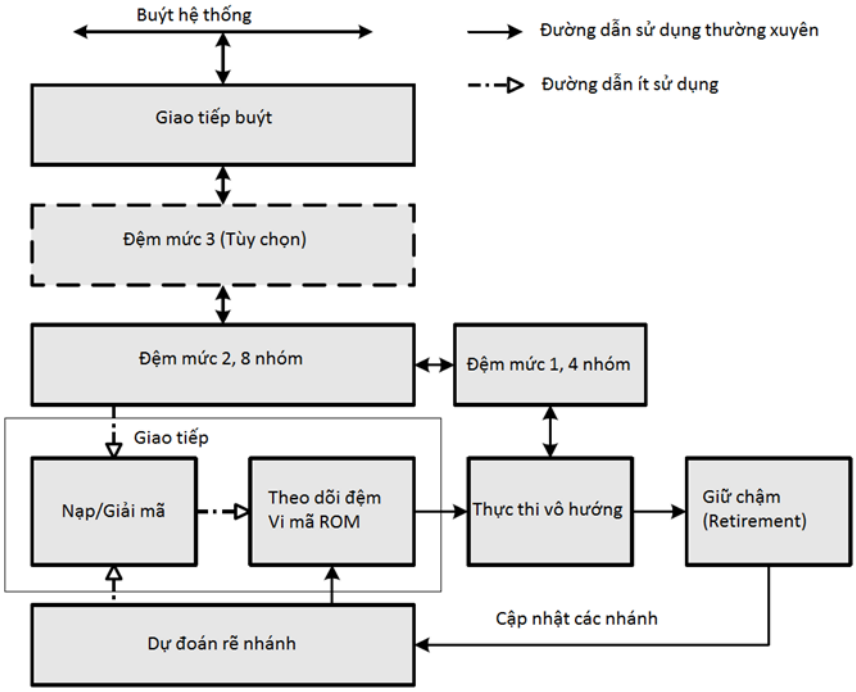
  Trong chế độ này bộ xử lý chuyển qua lại các không gian địa chỉ riêng biệt trong khi lưu lại ngữ cảnh căn bản của các chương trình đang chạy. Các đoạn mã SMM có thể được thực hiện hoàn toàn trong suốt. Ngay khi quay trở lại từ chế độ SMM, bộ xử lý được khôi phục lại trạng thái giống như trước khi ngắt SMM xảy ra.

Bất kỳ chương trình chạy trên bộ xử lý IA-32 được cung cấp các tài nguyên để thực hiện lệnh, lưu đoạn mã, dữ liệu và các thông tin trạng thái. Các tài nguyên này tạo lập nên môi trường thực thi cho chương trìnhh:

- Không gian địa chỉ: bất cứ chương trình nào đều có thể đánh địa chỉ không gian nhớ tuyến tính tới 2^{32} byte hay 4GB và không gian địa chỉ vật lý có thể lên tới 2^{36} khi sử dụng cách đánh địa chỉ mở rộng.
- Các thanh ghi thực thi căn bản: bao gồm 8 thanh ghi dùng chung, sáu thanh ghi đoạn, thanh ghi cờ và con trỏ lệnh EIP.
- Các thanh ghi đấu phẩy động x87FPU: bao gồm 8 thanh ghi dữ liệu, thanh ghi điều khiển, thanh ghi trạng thái, thanh ghi lệnh, thanh ghi con trỏ toán hạng, thẻ và mã lệnh. Các thanh ghi này cho phép thực hiện các phép toán với độ chính xác kép mở rộng hay với số nguyên 8 byte.

- Các thanh ghi MMX: bao gồm 8 thanh ghi hỗ trợ cơ chế thực hiện 1 lệnh và nhiều dữ liệu với các thao tác các số nguyên (1 byte, 2 byte hay 4 byte) được xếp vào gói 64 bít.
- Các thanh ghi XMM: hỗ trợ các thao tác số nguyên và số thực được xếp vào các gói 128 bít.

Các vi xử lý thế hệ sau hỗ trợ IA-32 áp dụng các tính năng thực thi lệnh tiên tiến cho phép thực hiện được nhiều hơn 1 lệnh trong 1 chu trình lệnh như kỹ thuật đường ống, siêu vô hướng, hay siêu phân luồng. Các thế hệ Pentium đầu tiên sử dụng các vi kiến trúc siêu vô hướng cho phép thực hiện 3 lệnh trong một chu kỳ xung nhịp với các siêu đường ống 12 đoạn và cơ chế thực thi vô hướng (out-of-order execution). Vi kiến trúc Netburst trong Hình VII-1. Vi kiến trúc Netburst tăng cường tính năng kiến trúc Pentium thế hệ đầu bằng việc tăng cường năng lực của đơn vị xử lý, năng cao hiệu năng của bộ đệm tích hợp, mở rộng giao tiếp buýt. Các vi xử lý IA-32 thế hệ mới còn hỗ trợ cơ chế đa nhân (multi-core) bên cạnh kiến trúc siêu phân luồng cho phép chạy nhiều ứng dụng đồng thời. Việc kết hợp hai kiến trúc làm cho các chương trình ứng dụng có thể sử dụng 4 bộ vi xử lý lô-gíc trên 2 bô vi xử lý vật lý. Bên cạnh đó, bộ xử lý thế hệ mới hỗ trợ công nghệ ảo hóa cho phép nhiều hệ điều hành và ứng dụng chạy trên các máy ảo khác nhau cùng chia sẻ hệ thống phần cứng.



Hình VII-1. Vi kiến trúc Netburst

Kiến trúc IA-32 cung cấp các chức năng hỗ trợ hệ điều hành hay các phần mềm hệ thống. Với các thao tác vào/ra ở chế độ bảo vệ, các thao tác này bị hạn chế thông qua:

- Cờ đặc quyền IOPL (I/O privilege level) và trạng thái của quyền vào/ra trong phân đoạn trạng thái chương trình TSS (Task state segment)
- Cơ chế bảo vệ đoạn và trang bộ nhớ.

Về mô hình bộ nhớ, các chương trình không truy nhập trực tiếp vào bộ nhớ vật lý. Thay vào đó, các chương trình có thể sử dụng các mô hình truy nhập:

1. Tuyến tính: Chương trình coi bộ nhớ như một chuỗi liên tiếp các byte. Đoạn mã, dữ liệu và ngăn xếp đều nằm trong không gian địa chỉ này.
2. Phân đoạn: Bộ nhớ được chia thành các không gian khác nhau được gọi là đoạn. Thông thường dữ liệu, đoạn mã, ngăn xếp sử dụng các đoạn khác nhau. Bộ xử lý hỗ trợ IA-32 có thể cung cấp 16. 383 đoạn với các kích cỡ khác nhau, kích cỡ lớn nhất của 1 đoạn là 4GB.
3. Địa chỉ thực: đây là mô hình bộ nhớ của 8086.
4. Phân trang và bộ nhớ ảo: khi này bộ nhớ chương trình được chia thành các trang ánh xạ vào bộ nhớ ảo. Sau đó, bộ nhớ ảo được ánh xạ vào bộ nhớ thực. Nếu hệ điều hành sử dụng phân trang, cơ chế ánh xạ hoàn toàn trong suốt đối với chương trình ứng dụng

VII.1.2 Các vi xử lý hỗ trợ IA-32

Với ưu thế của công nghệ và thiết kế vi kiến trúc mới, mỗi một thế hệ vi xử lý IA-32 mới đều vượt ngưỡng tốc độ (tần số hoạt động) và năng lực thực hiện của các vi xử lý thế hệ trước. Bảng dưới đây liệt kê các vi xử lý IA-32 thế hệ đầu không có bộ đệm tích hợp trong vi xử lý (GP-thanh ghi dùng chung; FPU-thanh ghi dấu phẩy động).

Bảng VII-1. Vi xử lý hỗ trợ IA-32 thế hệ đầu

Vi xử lý	Năm sản xuất	Tần số (MHz)	Số thanh ghi	Buýt dữ liệu mở rộng	Bộ nhớ tối đa	Bộ đệm
80386DX	1985	20	32GP	32	4GB	
Intel 486DX	1989	25	32GP 80 FPU	32	4GB	8KB L1
Pentium	1993	60	32GP 80 FPU	64	4GB	16KB L1
Pentium Pro	1995	200	32GP 80 FPU	64	64GB	16KB L1 256-512KB L2
Pentium II	1997	266	32GP 80 FPU 64 MMX	64	64GB	32KB L1 256-512KB L2
Pentium III	1999	500	32GP 80 FPU 64 MMX 128 XMM	64	64GB	32KB L1 512KB L2

Bảng VII-2. Vi xử lý IA-32 thế hệ sau

Vi xử lý	Năm sản xuất	Vi kiến trúc	Tần số (GHz)	Số thanh ghi	Băng thông buýt hệ thống	Bộ nhớ tối đa	Bộ đệm
Pentium 4	2000	Netburst	1, 5	32 GP 80 FPU 64 MMX 128 XMM	3, 2GB/s	64GB	8KB L1 256KB L2
Pentium 4	2002	Netburst, Siêu phân luồng	3, 06	32 GP 80 FPU 64 MMX 128 XMM	4, 2GB/s	64GB	8KB L1 256KB L2
Pentium M	2003	Pentium M	1, 6	32 GP 80 FPU 64 MMX 128 XMM	3, 2GB/s	64GB	64KB L1 1MB L2
Pentium 4 Extreme	2005	Netburst, Siêu phân luồng	3, 73	32 GP 80 FPU 64 MMX 128 XMM	8, 5GB/s	64GB	16KB L1 2MB L2
Core Duo	2006	Pentium M, Lõi kép	2, 16	32 GP 80 FPU 64 MMX 128 XMM	5, 3 GB/s	4GB	64KB L1 2MB L2
Atom Z5xx	2008	Atom, Ảo hóa	1, 86	32 GP 80 FPU 64 MMX 128 XMM	4, 2GB/s	4GB	56KB L1 512KB L2

VII.2 Các vi xử lý tiên tiến dựa trên kiến trúc Intel IA-64

Kiến trúc Intel IA-64 bổ sung không gian địa chỉ chương trình 64 bít hỗ trợ không gian nhớ vật lý tới 40 bít và chế độ IA-32e so với kiến trúc IA-32 trước đó. Kiến trúc IA-64 đảm bảo tính tương thích ngược cho phép chạy các chương trình viết cho kiến trúc IA-32. Các chế độ mới của IA-64 bao gồm:

- Chế độ tương thích: cho phép chạy các ứng dụng 16 và 32 bít mà không phải biên dịch lại. Chế độ này tương tự như chế độ bảo vệ trong IA-32. Các ứng dụng chỉ truy nhập được 4GB đầu trong không gian nhớ tuyến tính. Tuy nhiên, ứng dụng có thể sử dụng không gian nhớ lớn hơn với chế độ mở rộng địa chỉ vật lý.
- Chế độ 64 bít. Cho phép chương trình truy nhập không gian nhớ tuyến tính 64 bít. Chế độ này mở rộng số lượng các thanh ghi dùng chung và thanh ghi XMM từ 8 lên 16. Các thanh ghi dùng chung có kích cỡ 64 bít.

  Chế độ 64 được kích hoạt trên cơ sở đoạn mã. Kích cỡ mặc định cho địa chỉ là 64 bít còn toán hạng 32 bít. Kích cỡ của toán hạng có thể thay đổi theo từng lệnh sử dụng tiền tố

REX. Điều này giúp cho các câu lệnh cũ có thể chuyển sang chế độ 64 bít thanh ghi và địa chỉ.

Bảng VII-3. Vi xử lý hỗ trợ IA-64

Vi xử lý	Năm sản xuất	Vi kiến trúc	Tần số (GHz)	Số thanh ghi	Băng thông buýt hệ thống	Bộ nhớ tối đa	Bộ đệm
Xeon	2004	Netburst, Siêu phân luồng, IA-64	3, 6	32, 64 GP 80 FPU 64 MMX 128 XMM	6, 4GB/s	64GB	16KB L1 1MB L2
Xeon	2005	Netburst, Siêu phân luồng, IA-64	3, 03	32, 64 GP 80 FPU 64 MMX 128 XMM	5, 3GB/s	1024GB	16KB L1 1MB L2 8MB L3
Pentium 4 Extreme	2005	Netburst, Siêu phân luồng, IA-64	3, 73	32 GP 80 FPU 64 MMX 128 XMM	8, 5GB/s	64GB	16KB L1 2MB L2
Dual-Core Xeon	2005	Netburst, Siêu phân luồng, Đa nhân, IA-64	3	32, 64 GP 80 FPU 64 MMX 128 XMM	6, 4GB/s	64GB	16KB L1 2MB L2 (Tổng 4MB)
Pentium 4 672	2005	Netburst, Siêu phân luồng, IA-64, Ảo hóa, Đa nhân	3, 8	32, 64 GP 80 FPU 64 MMX 128 XMM	6, 4GB/s	64GB	16KB L1 2MB L2
Core 2 Extreme X6800	2006	Netburst, Siêu phân luồng, IA-64, Ảo hóa, Đa nhân	2, 93	32, 64 GP 80 FPU 64 MMX 128 XMM	8, 5GB/s	64GB	64KB L1 4MB L2
Xeon 7140	2006	Netburst, Siêu phân luồng, IA-64, Ảo hóa, Đa nhân	3, 40	32, 64 GP 80 FPU 64 MMX 128 XMM	12, 8 GB/s	64GB	64KB L1 1MB L2 (tổng 2MB) 16MB L3
Xeon 5472	2007	Netburst, Siêu phân luồng, IA-64, Ảo hóa, Đa nhân (4 nhân)	3, 00	32, 64 GP 80 FPU 64 MMX 128 XMM	12, 8 GB/s	256GB	64KB L1 6MB L2 (Tổng 12MB)
Atom	2008	Atom, IA-64, Ảo hóa, Đa nhân (4 nhân)	1, 60	32, 64 GP 80 FPU 64 MMX 128 XMM	12, 8 GB/s	64GB	56KB L1 512KB L2 (Tổng 1MB)
Core i7	2008	Netburst, Siêu phân luồng, IA-64, Ảo hóa, Đa nhân (4 nhân)	3, 20	32, 64 GP 80 FPU 64 MMX 128 XMM	6, 4 GT/s	64GB	64KB L1 256KB L2 8MB L3

VII.3 Các vi xử lý tiên tiến của Sun Microsystems

Sun Microsystems hỗ trợ thiết kế bộ xử lý có thể mở rộng SPARC (Scalable Processor Architecture). Kiến trúc này chịu ảnh hưởng của máy tính Berkeley RISC I. Tập lệnh và tổ chức các thanh ghi của bộ xử lý SPARC rất giống với Berkeley RISC. SPARC cho phép triển khai từ các ứng dụng nhúng cho tới các máy chủ rất lớn, tất cả đều dùng chung một tập lệnh căn bản. Hiện nay, bộ xử lý SPARC thường được sử dụng rộng rãi trong môi trường máy chủ, trạm ;àm việc sử dụng hệ điều hành SUN, Unix và Linux.

Bộ xử lý SPARC thường có tới 128 thanh ghi dùng chung. Tại bất cứ thời điểm nào, phần mềm có thể sử dụng tức thì 32 thanh ghi bao gồm 8 thanh ghi toàn cục, 24 thanh ghi ngăn xếp. Các thanh ghi ngăn xếp có thể tạo thành cửa số thanh ghi (register window) tối đa 32 cửa số cho phép tối ưu các thao tác gọi hàm và trở về. Mỗi cửa số có 8 thanh ghi cục bộ và dùng chung 8 thanh ghi với cửa sổ kề. Các thanh ghi chia sẻ được dùng để truyền các tham số và giá trị trả về cho các hàm còn thanh ghi cục bộ dùng để lưu các giá trị cục bộ giữa các lời gọi hàm.

Hầu hết các lệnh xử lý của SPARC chỉ sử dụng các toán hạng thanh ghi. Các lệnh nạp và lưu chuyên dùng để trao đổi dữ liệu giữa các thanh ghi và bộ nhớ. Ngoài chế độ địa chỉ thanh ghi, SPARC chỉ sử dụng chế độ địa chỉ dịch chuyển. Trong chế độ này, địa chỉ hiệu dụng của toán hạng được dịch chuyển 1 đoạn tương ứng với giá trị của thanh ghi. Để thực hiện câu lệnh nạp hoặc ghi, quá trình thực hiện lệnh sẽ cần thêm 1 giai đoạn để tính địa chỉ ô nhớ.

Vi xử lý hỗ trợ SPARC 32 bít phiên bản 8 cho phép sử dụng 16 thanh ghi dấu phẩy động với độ chính xác kép, hoặc 32 thanh ghi với độ chính xác đơn. Các cặp chẵn-lẻ của các thanh ghi độ chính xác kép có thể kết hợp với nhau để nâng độ chính xác lên gấp đôi mức 4. SPARC 64 bít phiên bản 9, xuất hiện vào năm 1993, có thêm 16 thanh ghi độ chính xác kép nhưng các thanh ghi mới này không tách thành các thanh ghi có độ chính xác đơn được.

Bảng dưới đây liệt kê một số tính năng của các vi xử lý sử dụng SPARC.

Bảng VII-4. Tính năng một số vi xử lý SPARC

Tên	Tần số MHz	Năm sản xuất	Số luồng x Số nhân	Số chân tín hiệu	Đệm dữ liệu L1 (k)	Đệm lệnh L1 (k)	Đệm L2 (k)
UltraSPARC IIs (Blackbird)	250–400	1997	1×1	521	16	16	1024 or 4096
UltraSPARC IIs (Sapphire-Black)	360–480	1999	1×1	521	16	16	1024–8192
UltraSPARC IIi (Sabre)	270–360	1997	1×1	587	16	16	256–2048
UltraSPARC IIi (Sapphire-Red)	333–480	1998	1×1	587	16	16	2048
UltraSPARC IIe (Hummingbird)	400–500	1999	1×1	370	16	16	256
UltraSPARC IIi (IIe+) (Phantom)	550–650	2000	1×1	370	16	16	512
UltraSPARC III (Cheetah)	600	2001	1×1	1368	64	32	8192
UltraSPARC III Cu (Cheetah+)	1002–1200	2001	1×1	1368	64	32	8192
UltraSPARC IIIi (Jalapeño)	1064–1593	2003	1×1	959	64	32	1024
UltraSPARC IV (Jaguar)	1050–1350	2004	1×2	1368	64	32	16384
UltraSPARC IV+ (Panther)	1500–2100	2005	1×2	1368	64	64	2048
UltraSPARC T1 (Niagara)	1000–1400	2005	4×8	1933	8	16	3072
UltraSPARC T2 (Niagara 2)	1000–1600	2007	8×8	1831	8	16	4096
UltraSPARC T2 Plus (Victoria Falls)	1200–1600	2008	8×8	1831	8	16	4096

Tài liệu tham khảo

1. Crisp J. *Introduction to microprocessors and microcontrollers*, Newnes 2004
2. David Calcutt, Fred Cowan,Hassan Parchizadeh, 8051 Microcontrollers An Applications-Based Introduction, Newnes 2004
3. Douglas V. Hall. *Microprocessor and Interfacing- programming and hardware*, 2nd edition. McGraw Hill. 1997.
4. Hari BalaKrishnan & Samel Madden. *The lecture notes on Computer Systems Engineering*, Open Courses Ware. Massachusets Institute of Technology.
5. Hồ Khánh Lâm, *Kỹ thuật vi xử lý*, NXB Bưu điện 2005
6. Intel Corp. Intel® 64 and IA-32 Architectures Software Developer's Manual
7. Rafiquzzaman M. *Microprocessor theory and applications with 68000/68020 and Pentium*, John Wiley&Sons 2008
8. Văn Thế Minh. *Kỹ thuật vi xử lý*. NXB Giáo dục 1999.

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

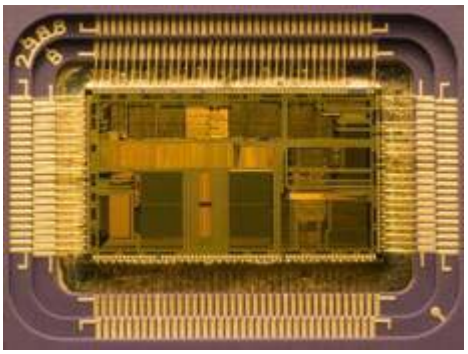
Biểu diễn số và ký tự

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.

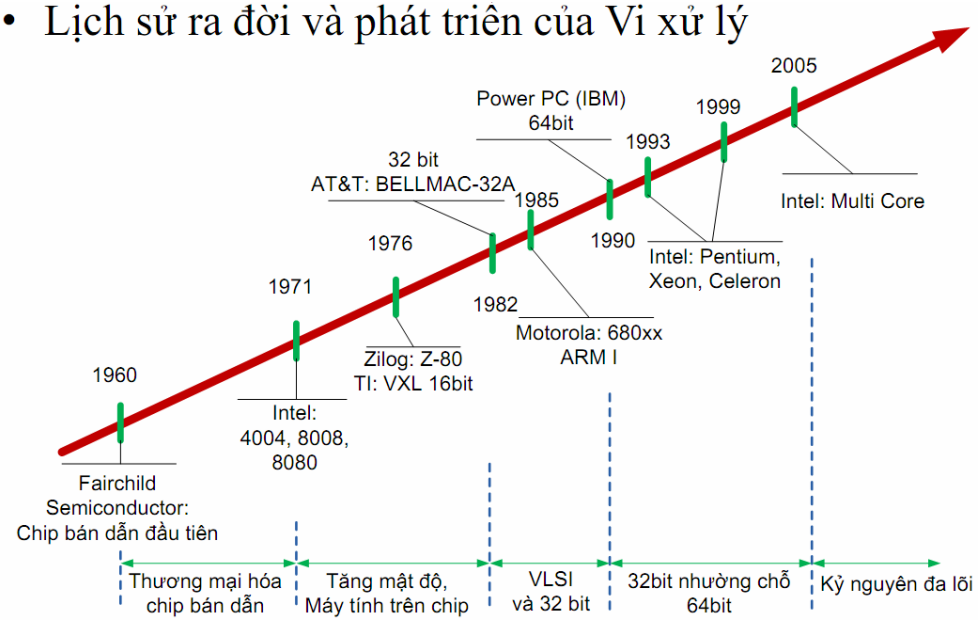
Vi điều khiển là một máy tính được tích hợp trên một chíp, nó thường được sử dụng để điều khiển các thiết bị điện tử. Vi điều khiển, thực chất, là một hệ thống bao gồm một vi xử lý có hiệu suất đủ dùng và giá thành thấp (khác với các bộ vi xử lý đa năng dùng trong máy tính) kết hợp với các khối ngoại 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:**

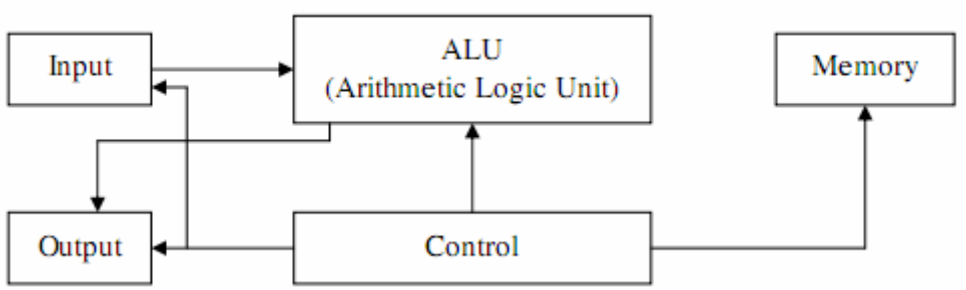
- Máy móc thiết bị hỗ trợ: máy đo nhịp tim, máy đo đường huyết, máy đo huyết áp, điện tim đồ, điện não đồ,…
- Máy cắt/mài kính
- Máy chụp chiếu (city, X-quang,...)

❖ **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



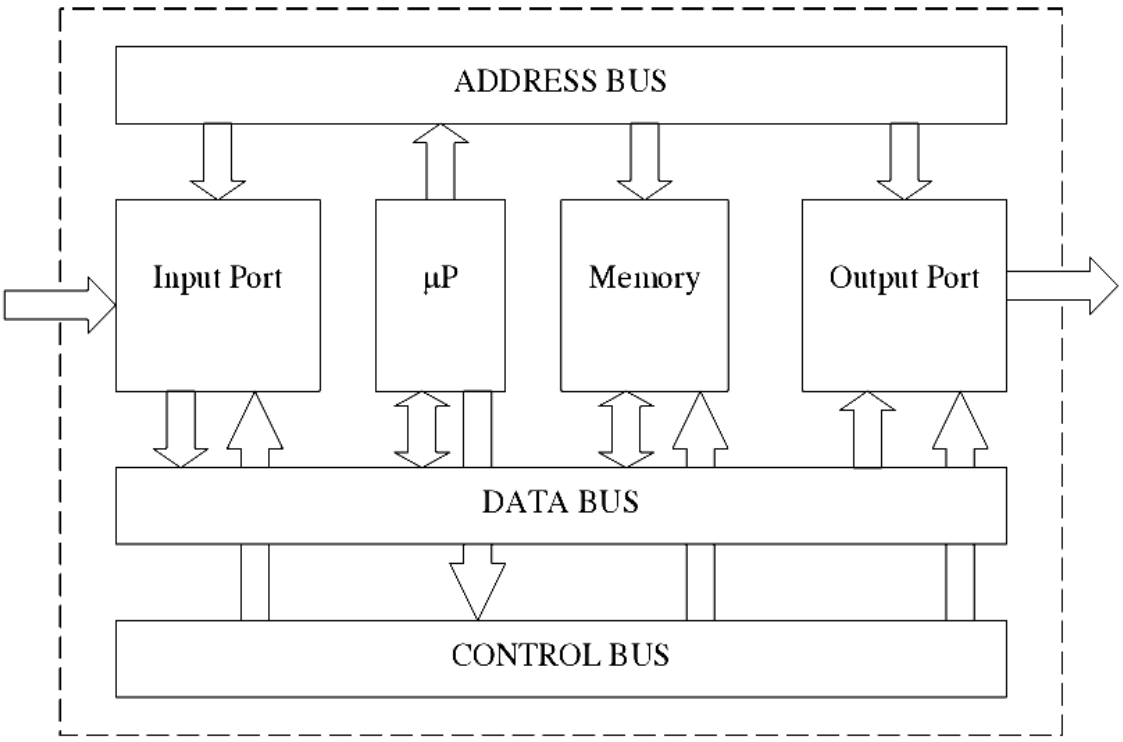
Hình 1-3. Sơ đồ khối một máy tính cổ điển

- 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.

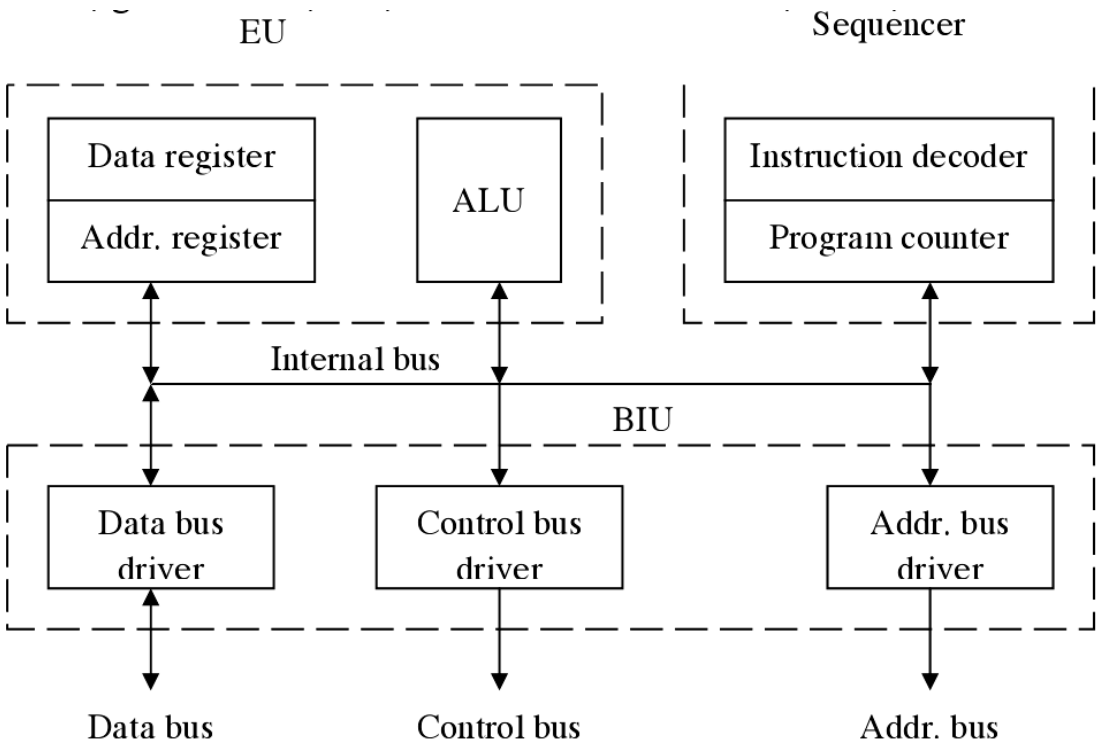


Hình 1-4. Sơ đồ khối hệ vi xử lý

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:

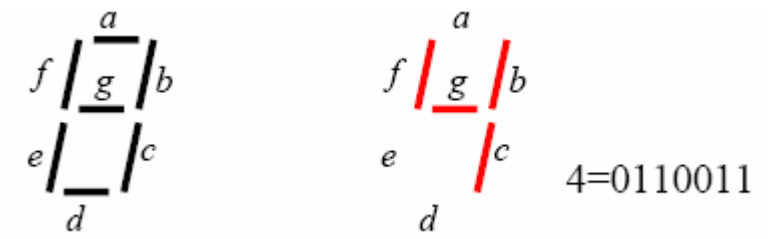
- Mã BCD không nén (unpacked): biểu diễn một số BCD bằng 8 bit nhị phân
- Mã BCD nén (packed): biểu diễn một số BCD bằng 4 bit nhị phân

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**

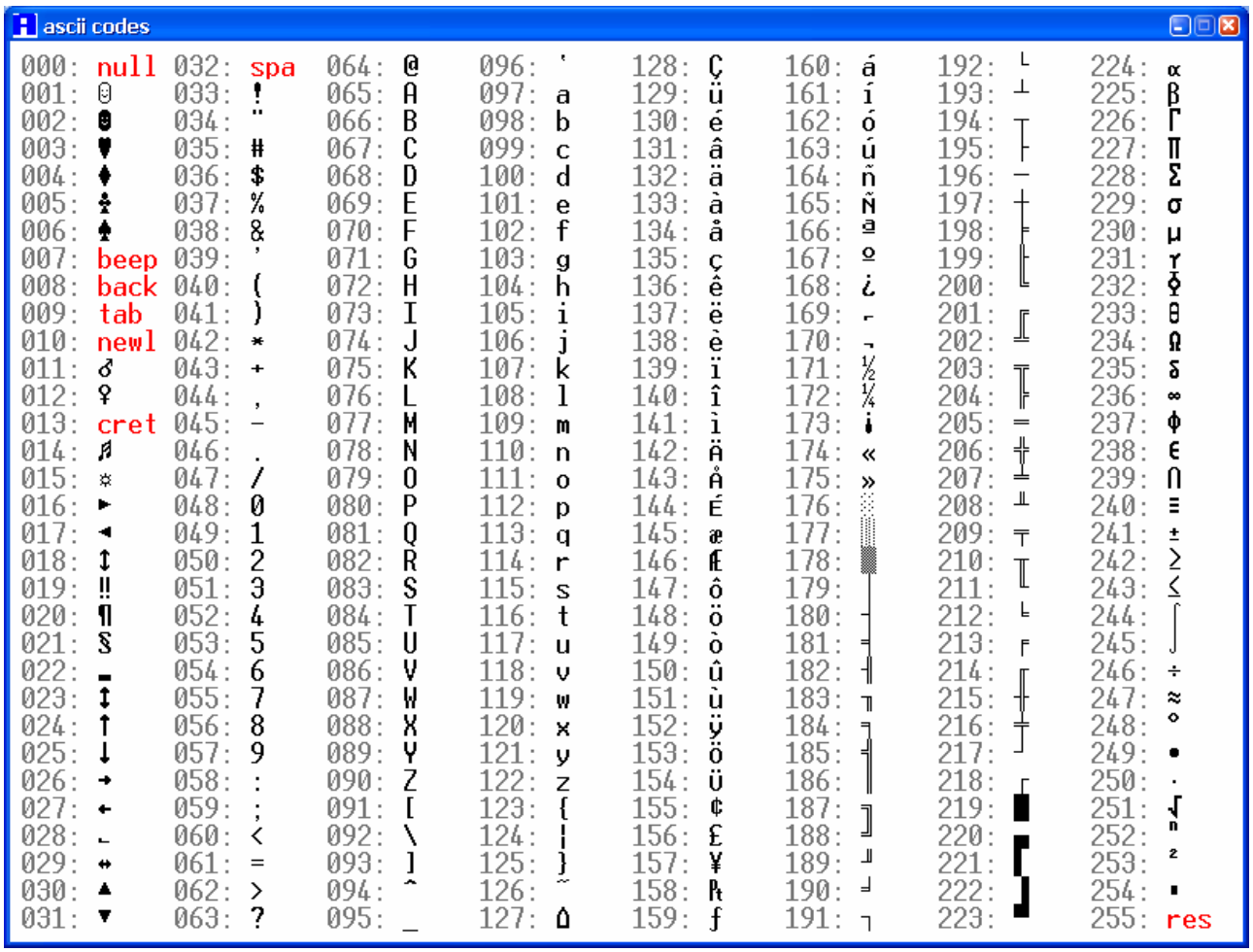
Hệ 10	Hệ 2	Hệ 8	Hệ 16	Binary-Coded Decimal		Gray Code	7-Segment	
				8421 BCD	EXCESS-3		abcdefg	Display
0	0000	0	0	0000	0011 0011	0000	111111	0
1	0001	1	1	0001	0011 0100	0001	011000	1
2	0010	2	2	0010	0011 0101	0011	110110	2
3	0011	3	3	0011	0011 0110	0010	111100	3
4	0100	4	4	0100	0011 0111	0110	011001	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

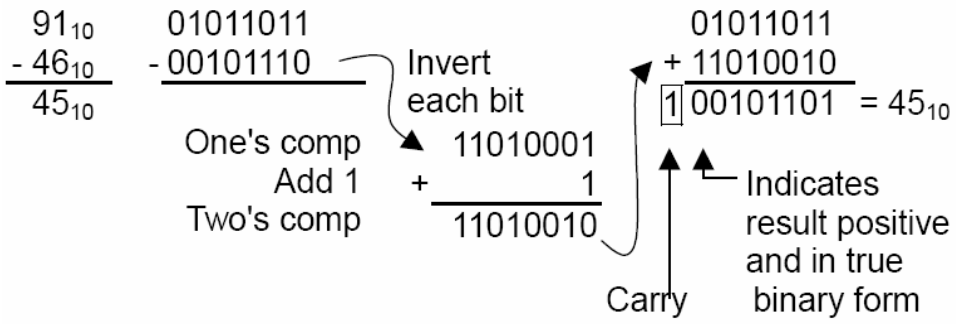
❖ Phép cộng nhị phân

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

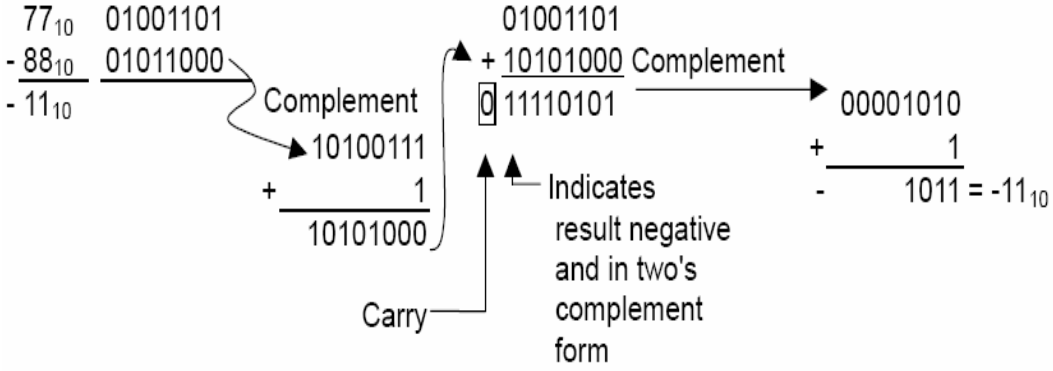
❖ Phép trừ nhị phân

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:

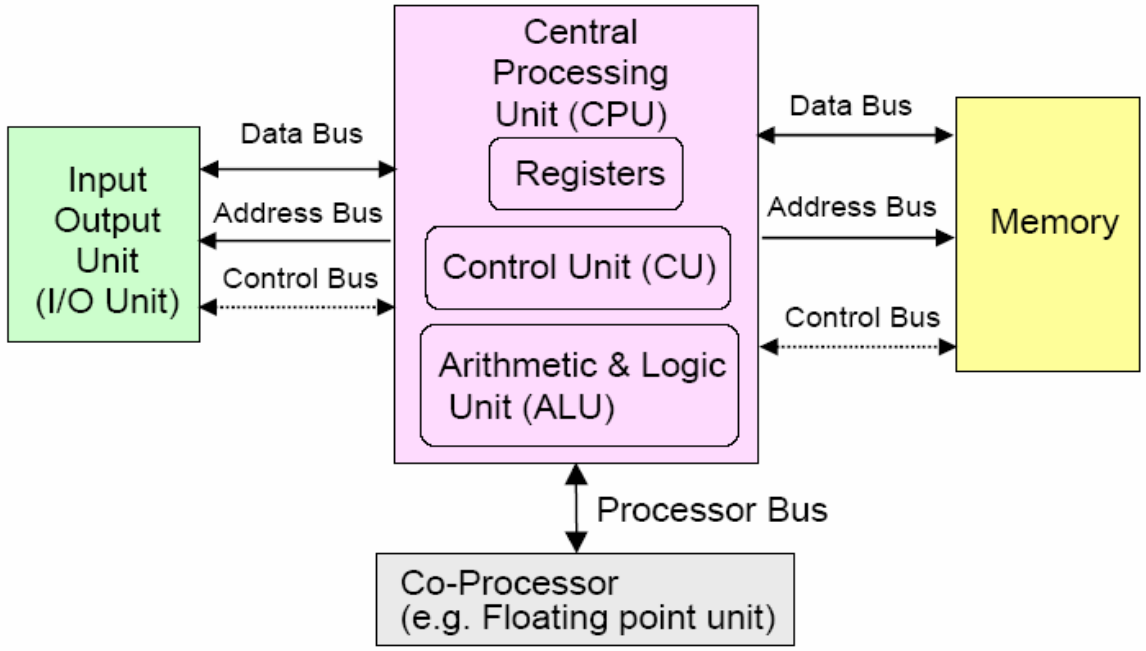
Hiểu được cấu trúc phần cứng của hệ vi xử lý; hiểu và vận dụng được các chế độ địa chỉ; nắm được tập lệnh và lập trình cho hệ vi xử lý 80x86

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

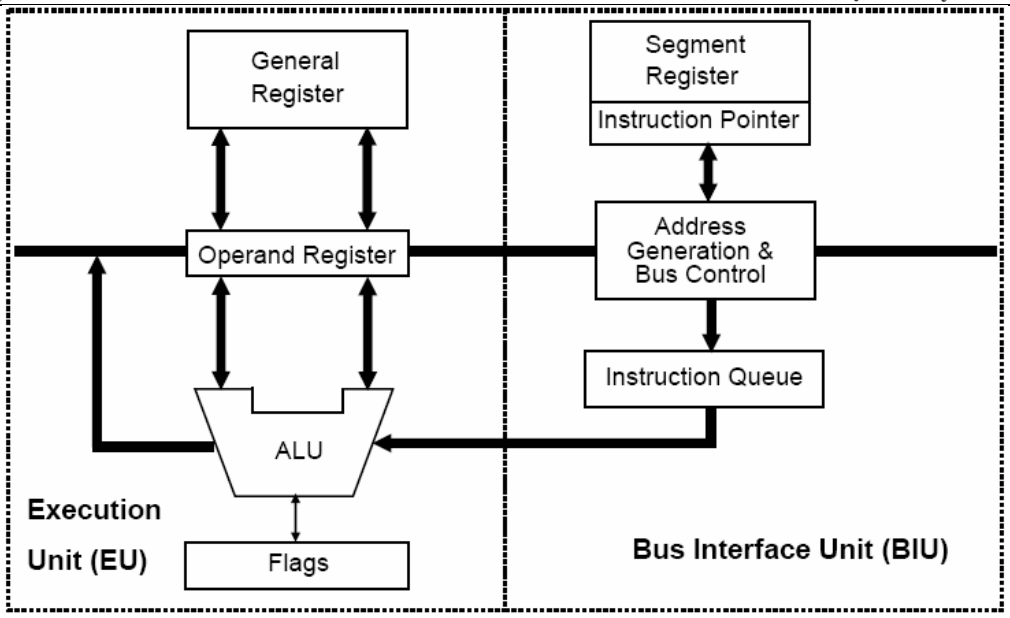


Hình 2-1.Tổng quan về phần cứng bộ xử lý

- **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

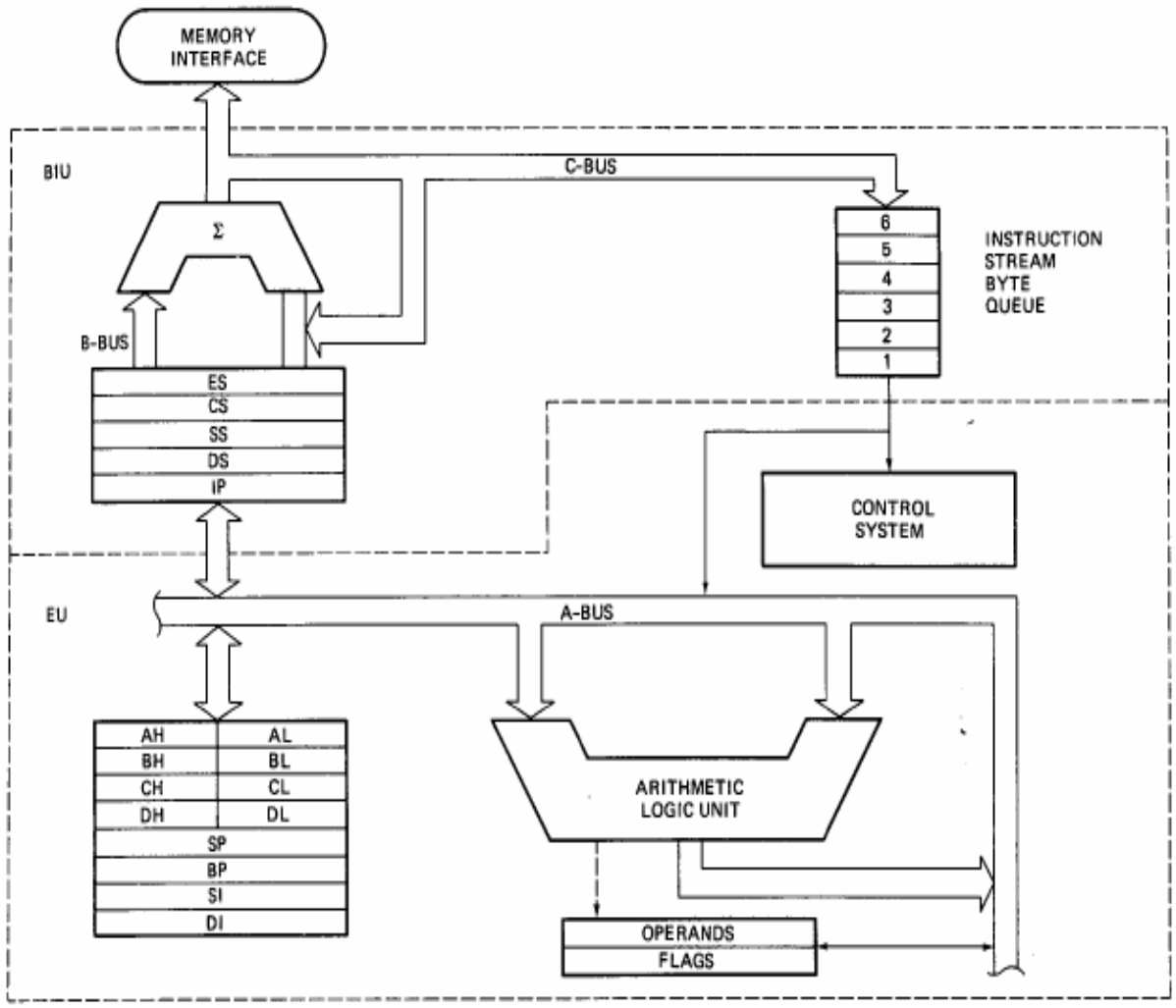
BIU có nhiệm vụ đưa ra địa chỉ, đọc mã lệnh từ bộ nhớ, đọc/ghi dữ liệu từ/vào cổng hoặc bộ nhớ. Bên trong BIU còn có bộ nhớ đệm lệnh (còn gọi là hàng đợi lệnh) dùng để chứa các lệnh đã đọc được nằm sẵn chờ EU xử lý. EU có nhiệm vụ cung cấp địa chỉ cho BIU để khối này đọc lệnh và dữ liệu, còn bản thân 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

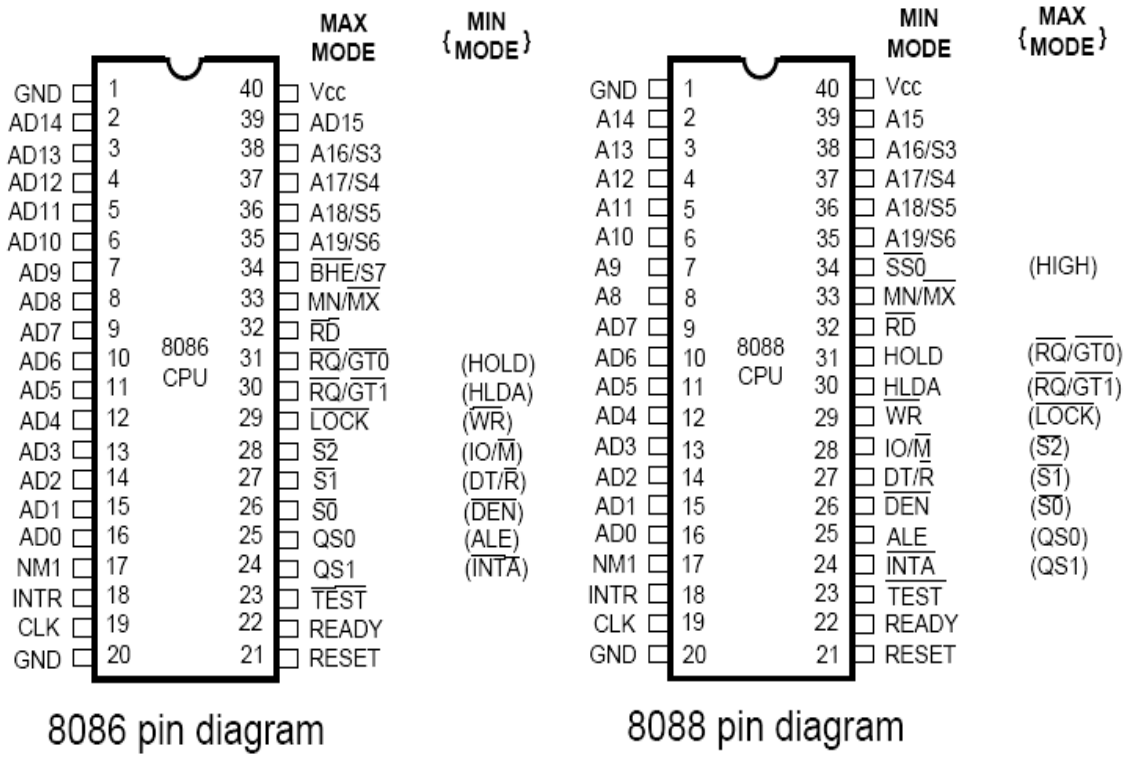
- **Các cờ điều kiện**
  - **Cờ nhớ - Carry flag (CF) –** Cờ này được đặt lên 1 khi tính toán một số không dấu bị tràn. Ví dụ khi cộng dạng byte: 255+1 (kết quả không nằm trong vùng 0..255). Khi không tràn, cờ này đặt bằng 0
  - **Cờ chẵn lẻ - parity flag (PF) –** Cờ PF=1 khi số lượng bit "1" trong kết 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

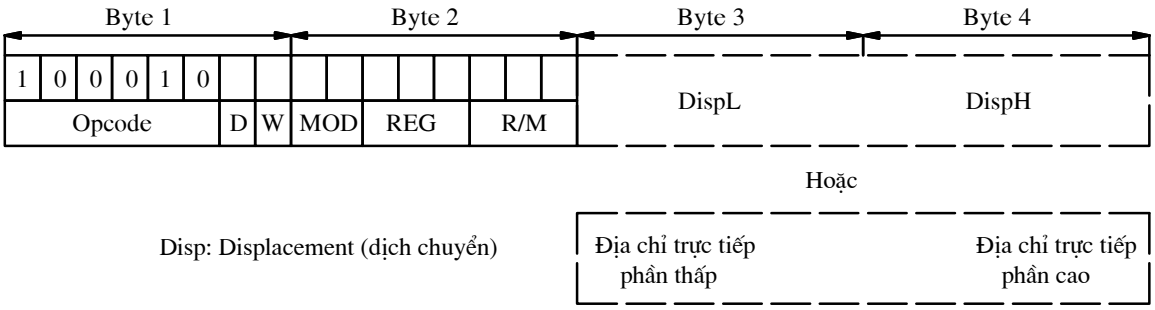
2.2 Chế độ địa chỉ

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.

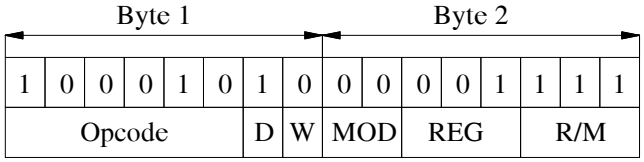
Hai bit MOD (chế độ) cùng với ba bit R/M (thanh ghi/bộ nhớ) tạo ra 5 bit dùng để chỉ ra chế độ địa chỉ cho các toán hạng của lệnh. Bảng 2.2 cho ta thấy cách mã hoá các chế độ địa chỉ.

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

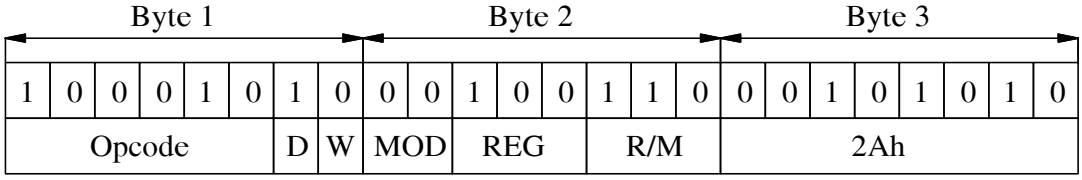
D = 1: Chuyển tới thanh ghi

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

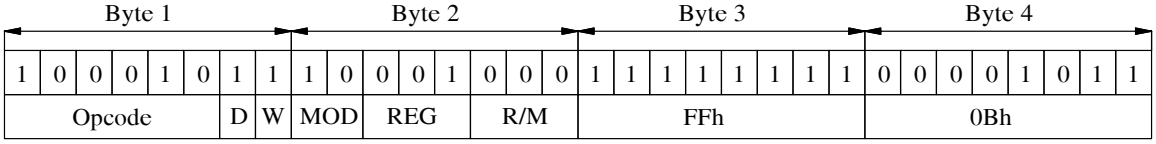
REG: 100 mã hoá AH

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:

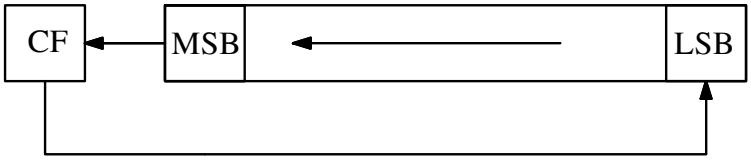
```
OR    AX, BX      ;AX←AX∨BX theo tung bit
OR    CL, 30H     ;lap bit b4 va b5 cua CL len 1
```

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
```

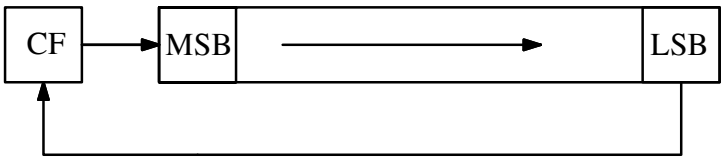
Trước khi thực hiện lệnh: AL = 01011110, CF = 0.

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
```

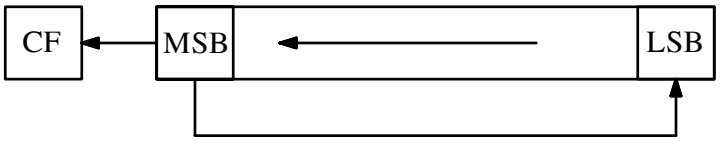
Trước khi thực hiện lệnh: AL = 11000010, CF = 1.

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
```

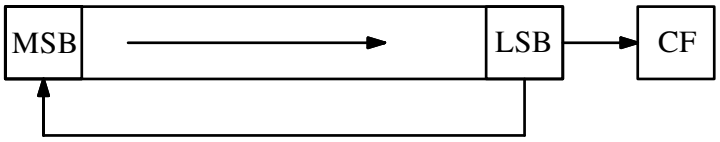
Trước khi thực hiện lệnh: AL = 11001100, CF = 1

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

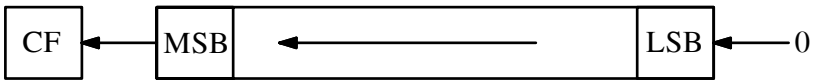
21. SAL/SHL - Shift Arithmetically Left (dịch trái số học)/Shift Logically Left (dịch trái logic).

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
```

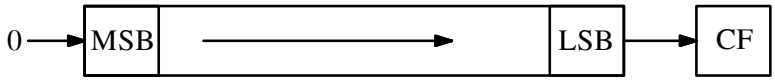
Trước khi thực hiện lệnh: AL = 11001100, CF = 0

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

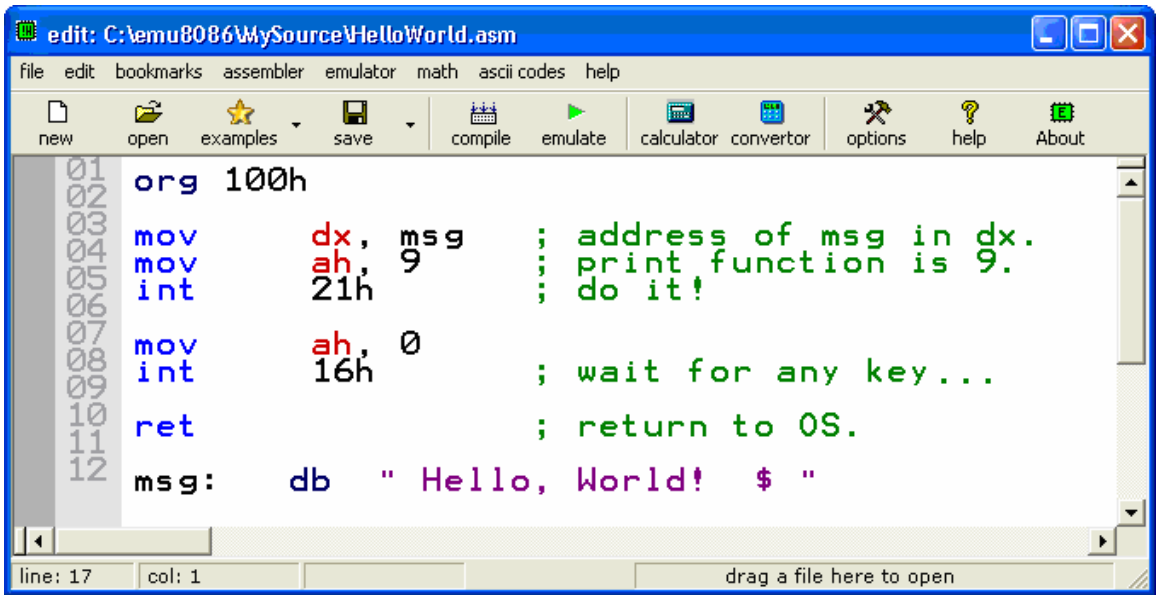
  (%1 là lấy tham số thứ nhất trong command line)

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

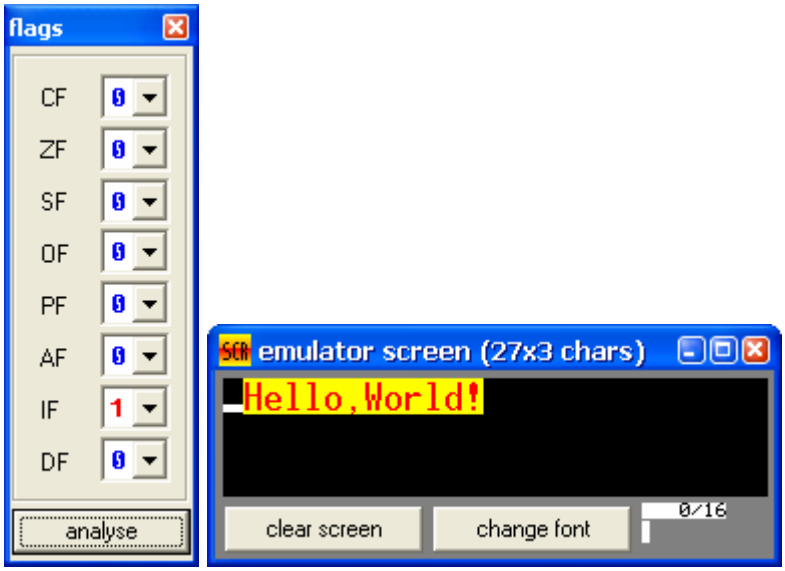
  RunASM hello

Chương trình emu8086:

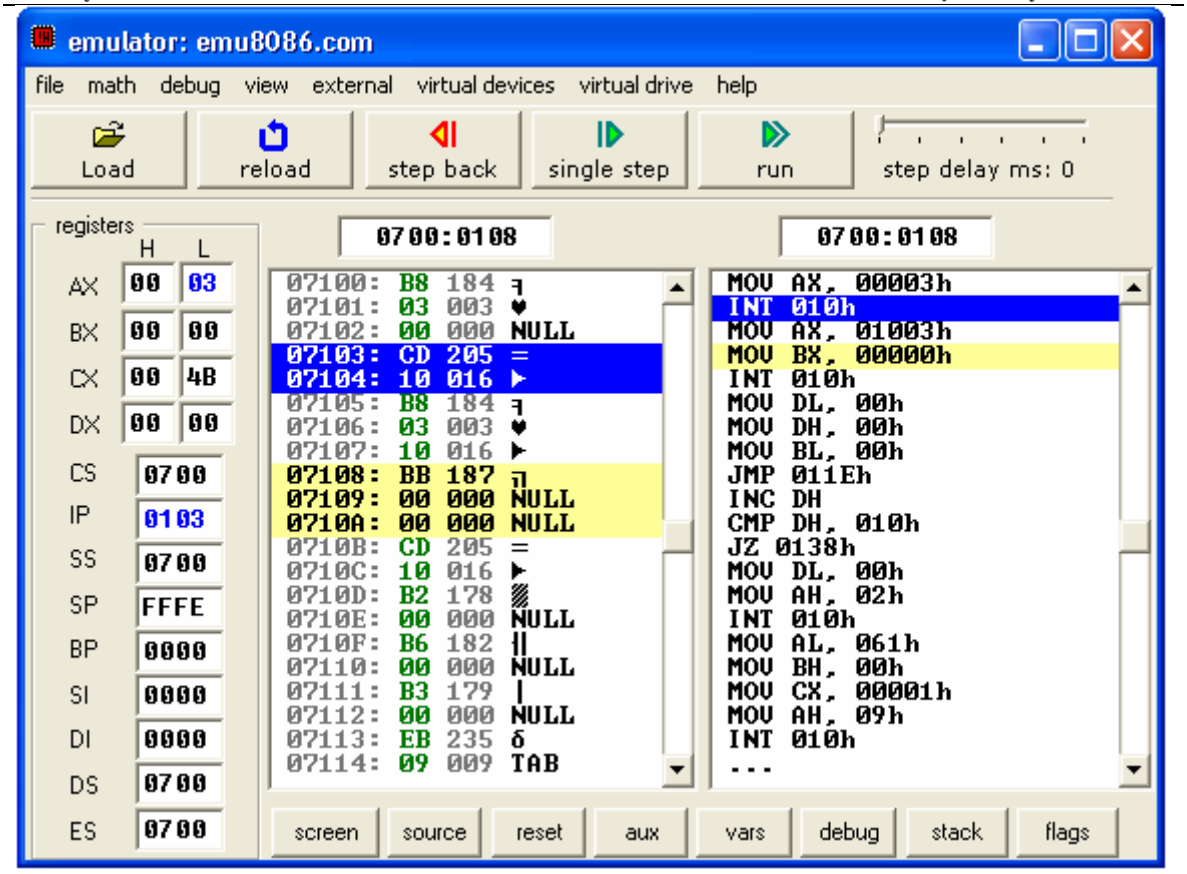
Chương trình emu8086 là chương trình lập trình mô phỏng cho 8086 (tương thích Intel và AMD) bao gồm bộ dịch ASM và giáo trình (tiếng anh) cho người mới bắt đầu. Chương trình có thể chạy hết hoặc chạy từng bước, ta có thể nhìn thấy các thanh ghi, bộ nhớ, stack, biến,…



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
```

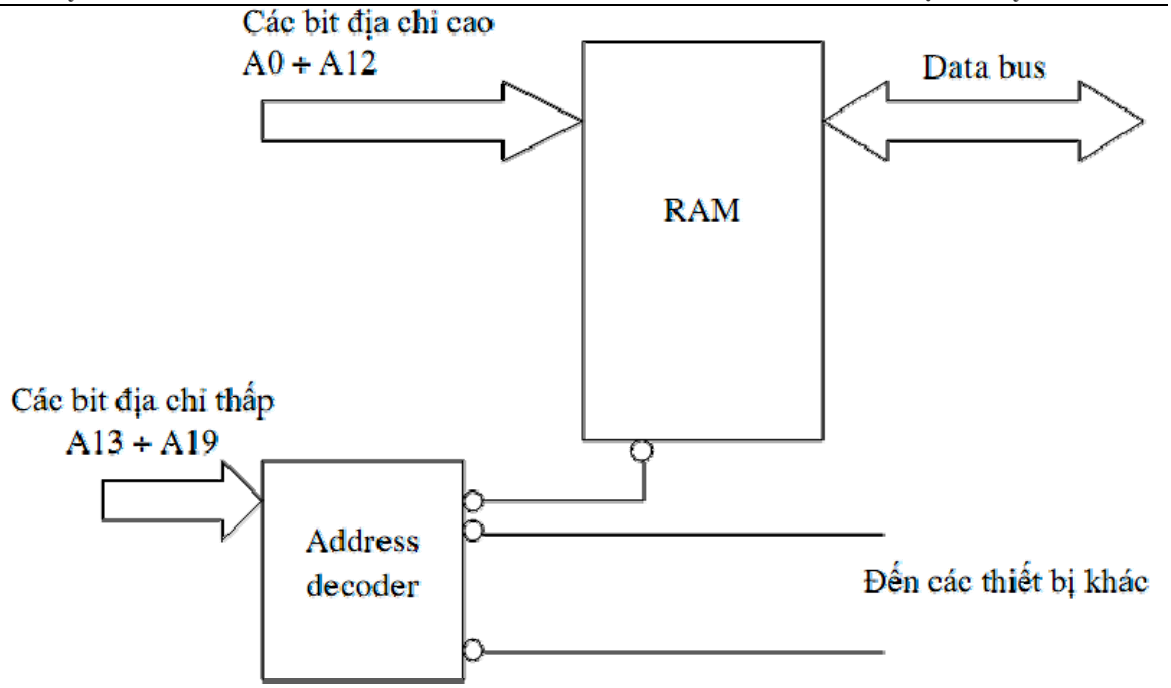
```
 LEA DX,m1
 MOV AH,0Ah  ; Nhap chuoi
 INT 21h
 LEA SI,m1
 ADD SI,2
 MOV DI,SI
Next: LODSB  ; Lay ky tu
 CMP AL,0Dh  ; Neu la ky tu Enter thi ket thuc
 JE quit
 CMP AL,'a'  ; Neu ky tu nhap khong phai la ky tu thuong tu ‘a’
toi ‘z’ thi bo qua
JB cont
CMP AL,'z'
JA cont
     SUB AL,20h  ; Chuyen ky 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

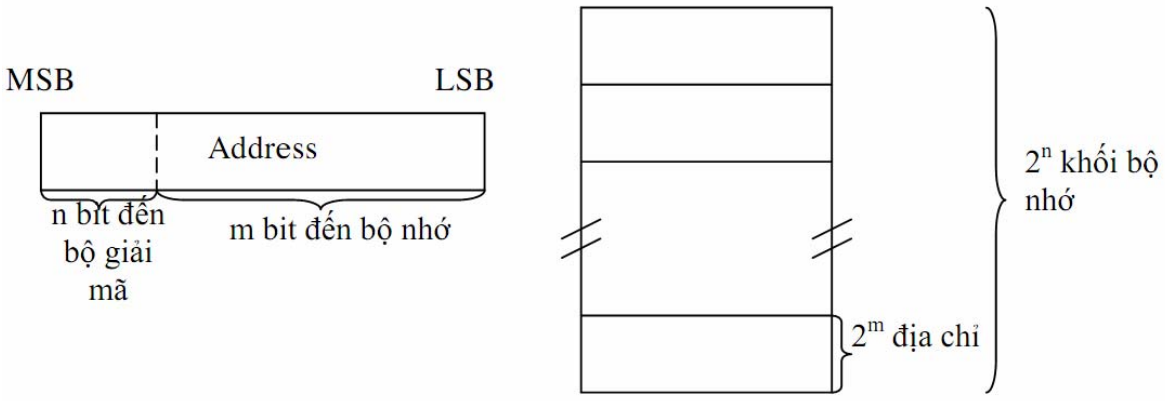
2.5.1.1 Giao tiếp bus cơ bản

- 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

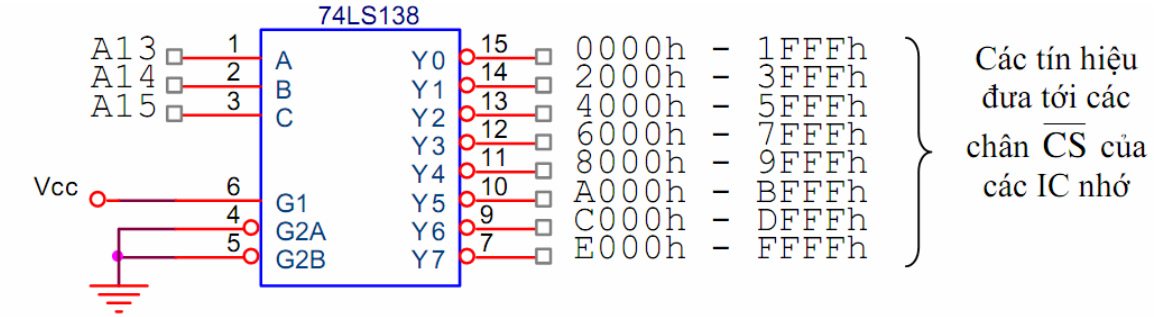
Quan hệ giữa giải mã địa chỉ và bảng bộ nhớ:



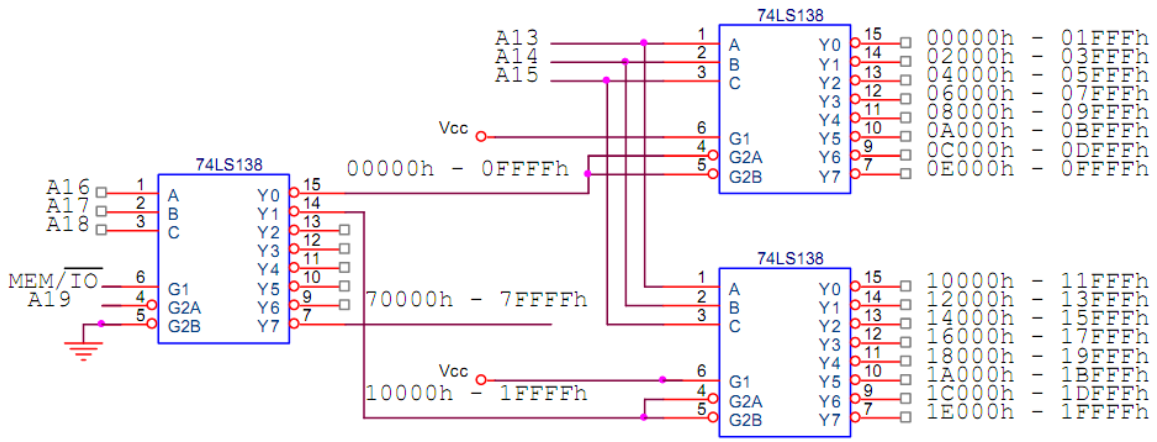
Hình 2-9. Quan hệ giữa giải mã địa chỉ và bộ nhớ

2.5.2 Giải mã địa chỉ

2.5.2.1 Dùng 74LS138



Dùng nhiều 74LS138



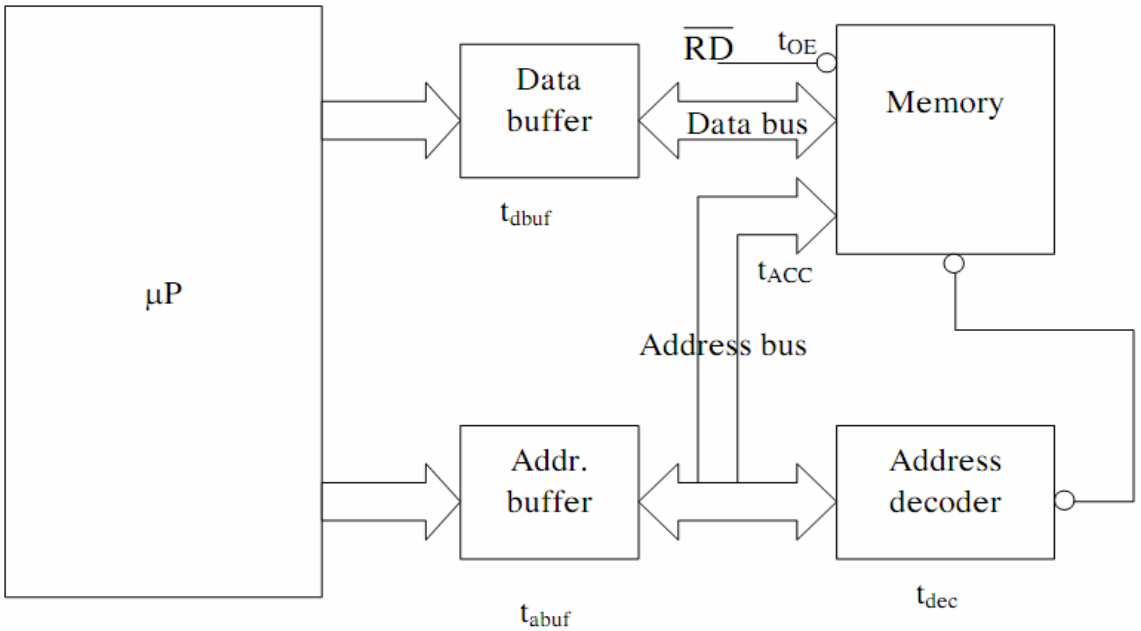
Hình 2-10. Mắc nối tầng nhiều 74LS138

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

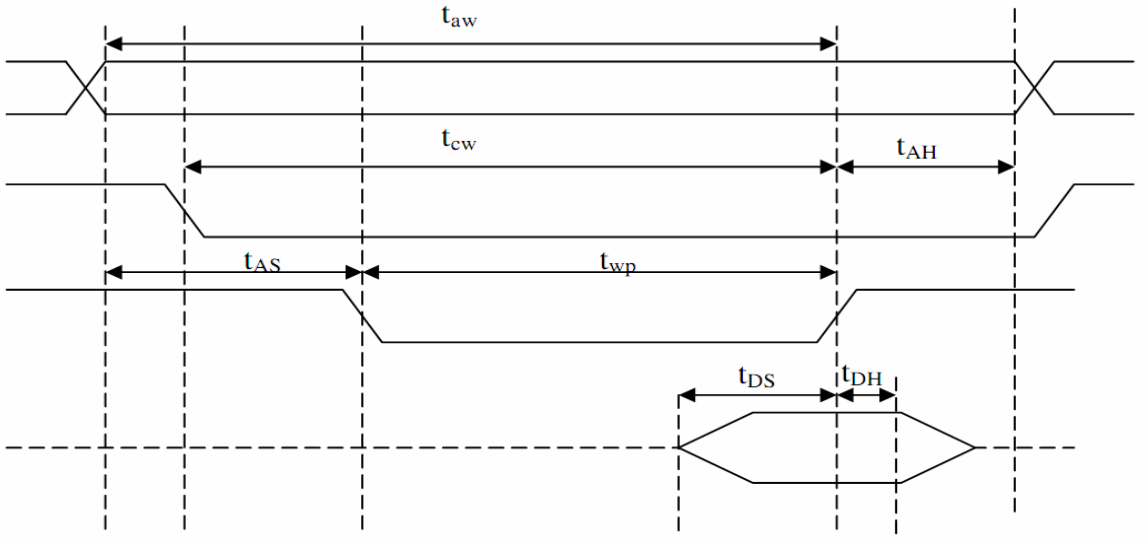
♦ **Định thời đọc bộ nhớ:**

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.

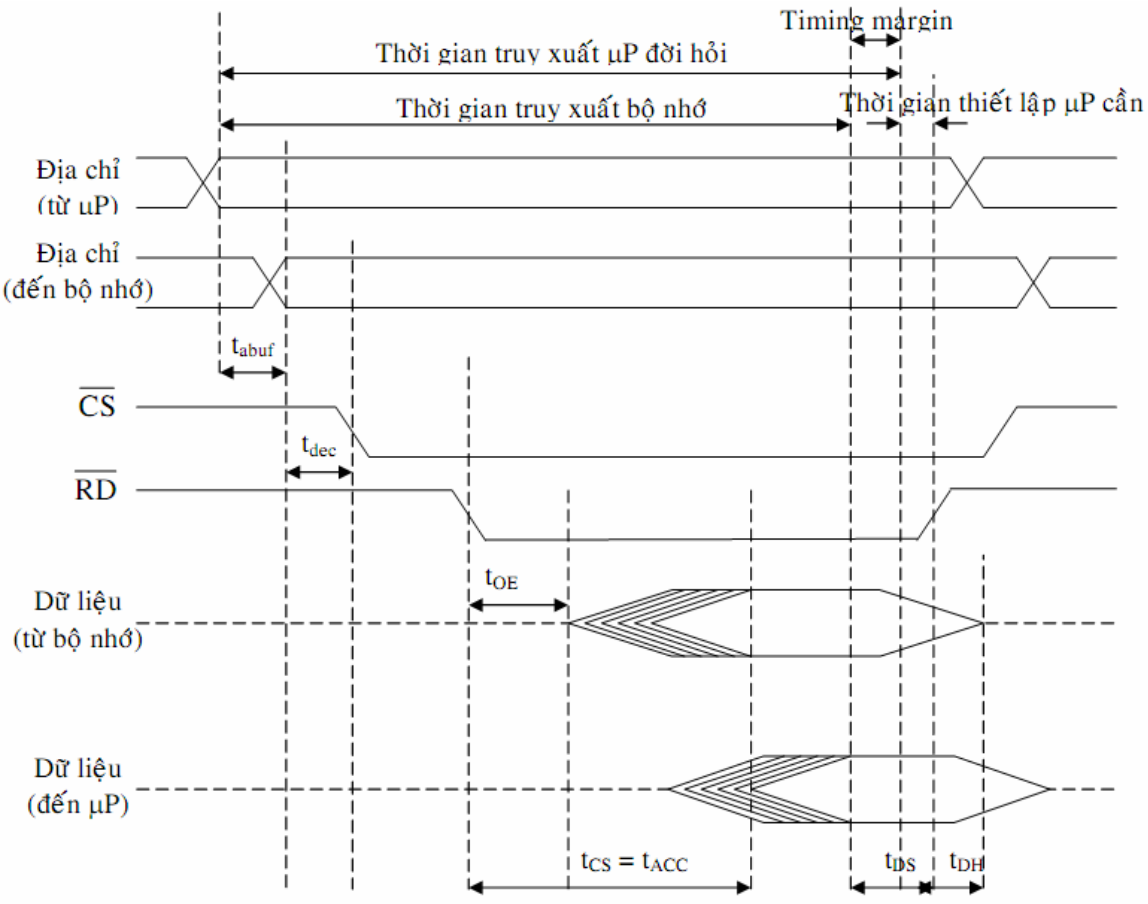
Hiệu giữa thời gian truy xuất cần thiết bởi µP với thời gian truy xuất thật sự của bộ nhớ gọi là biên định thời (timing margin).

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)

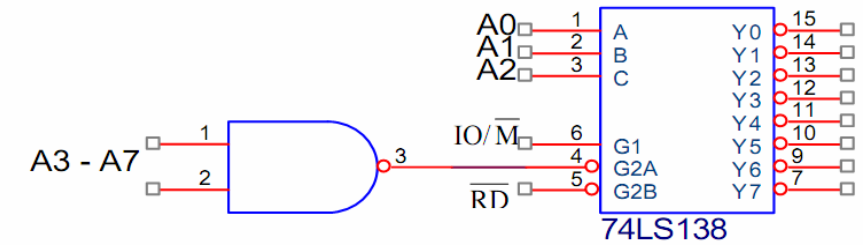
Thiết bị ngoại vi và bộ nhớ có chung không gian địa chỉ

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.

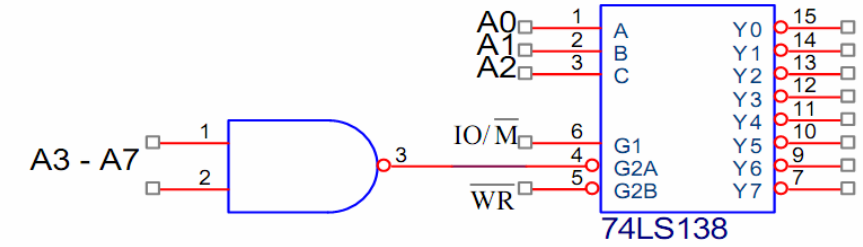
2.5.5 Giải mã địa chỉ cho thiết bị vào / ra

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



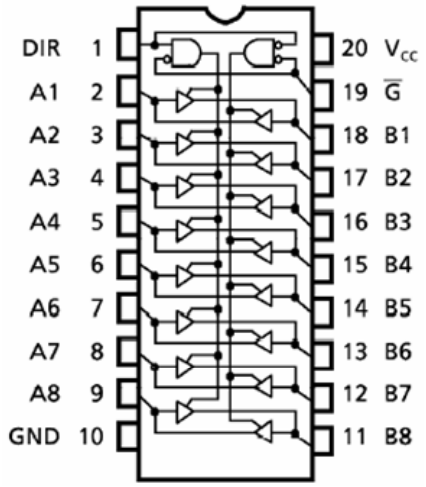
b. Giải mã cho cổng ra

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:



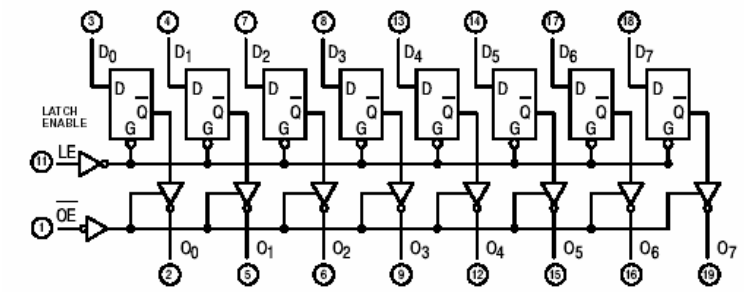
Inputs		Function		Outputs
$\overline{G}$	DIR	A bus	B bus	
L	L	Output	Input	A = B
L	H	Input	Output	B = A
H	X	High Impedance		Z

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:

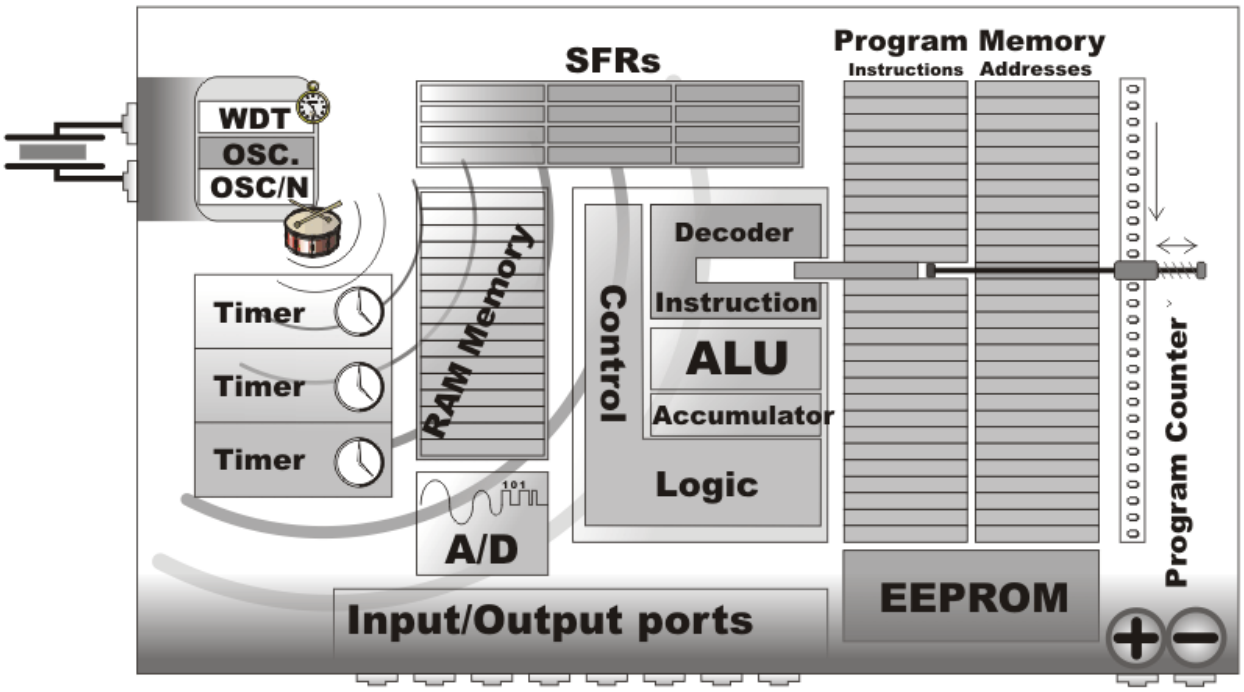
- Những thành phần điện tử được nhúng vào vi điều khiển có thể trực tiếp hoặc qua các thiết bị vào ra (công tắc, nút bấm, cảm biến, LCD, rơ le, …) điều khiển rất nhiều thiết bị và hệ thống như thiết bị tự động trong công nghiệp, điều khiển nhiệt độ, dòng điện, động cơ, …
- Giá thành rất thấp khiến cho 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.

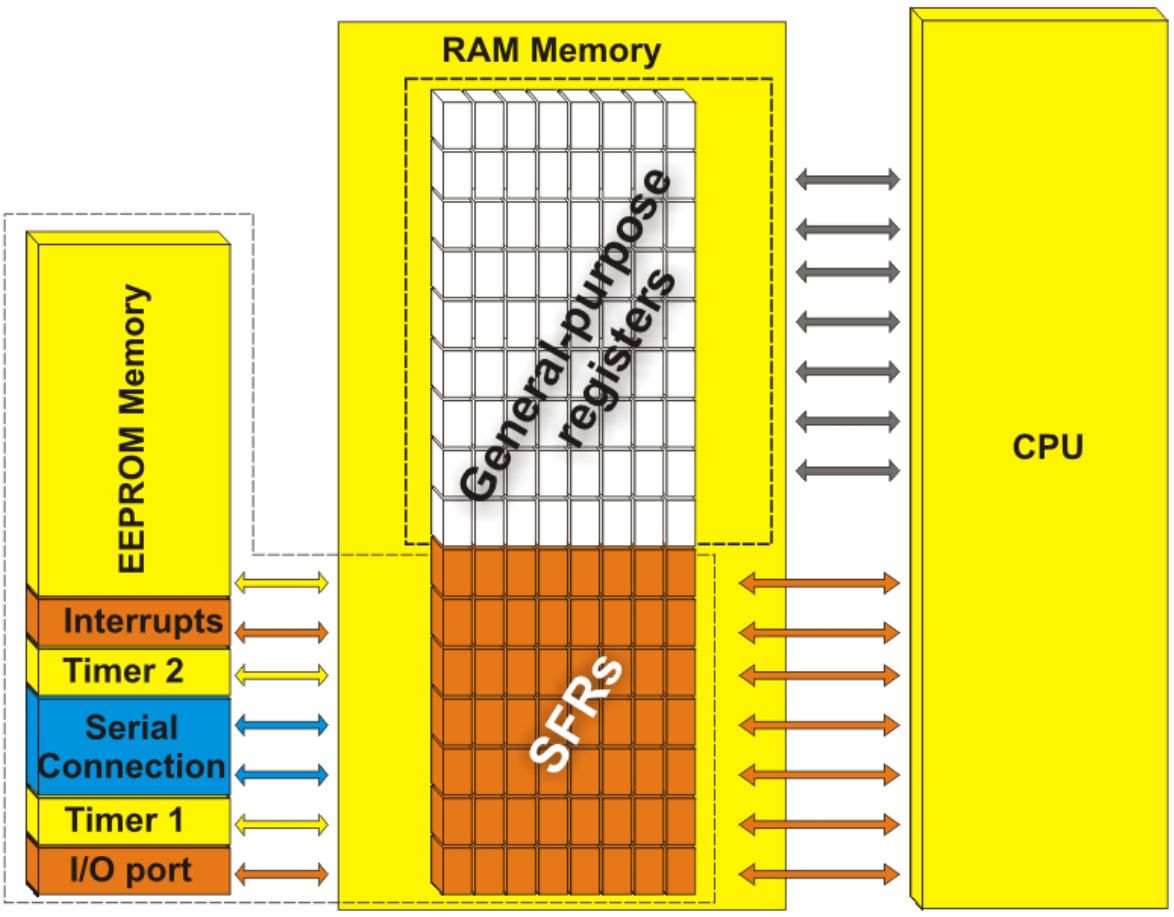
ROM nội thường là nhỏ hơn và đắt tiền hơn, nhưng lá ghim thêm có sẵn để kết nối với môi trường ngoại vi. Kích thước của dãy ROM từ 512B đến 64KB

❖ 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.

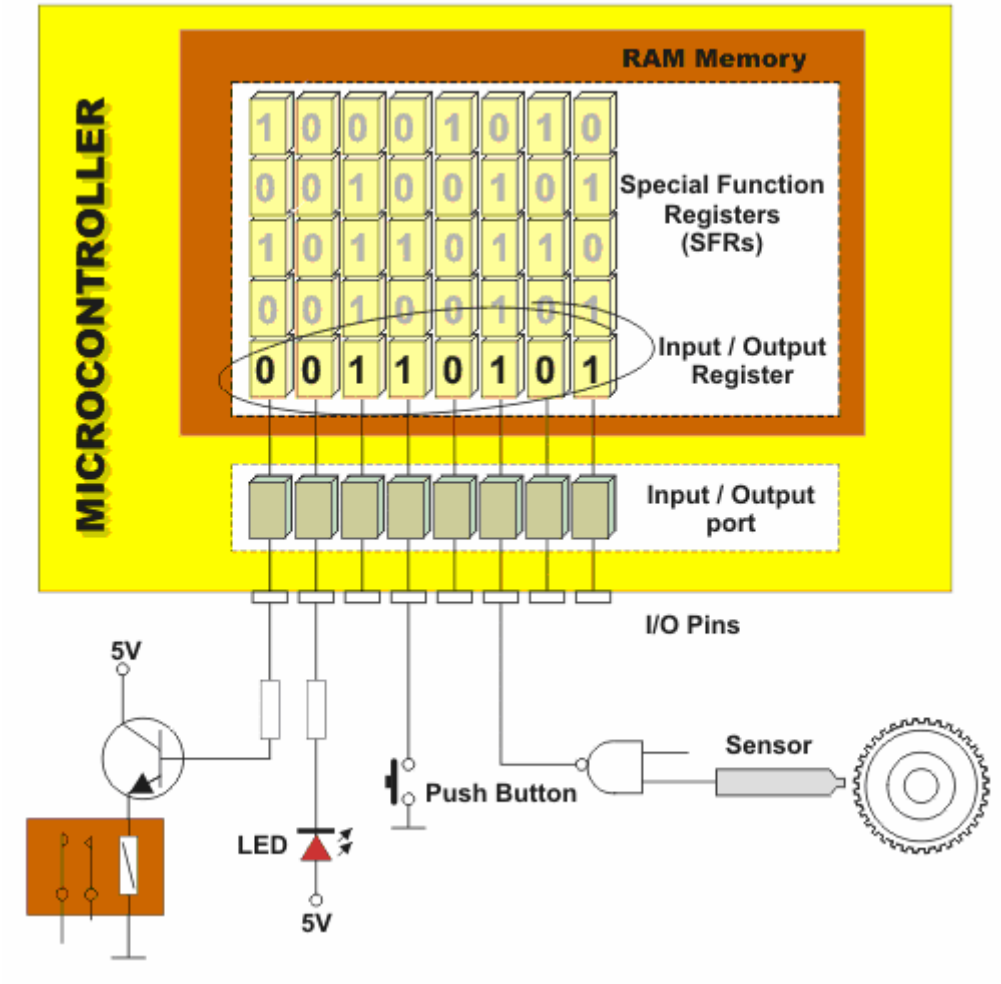
❖ Central Processor Unit (CPU)

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)

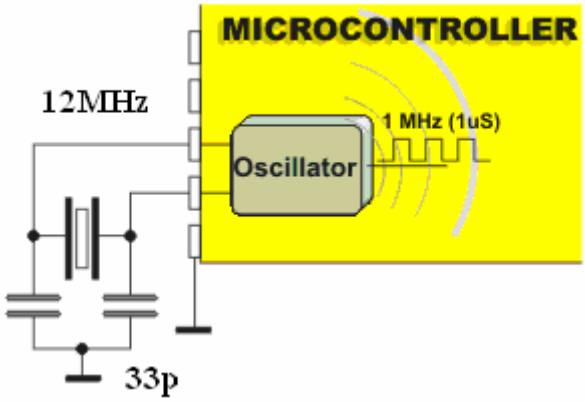
Để vi điều khiển có thể hoạt động hữu ích, nó cần có sự kết nối với các thiết bị ngoại vi. Mỗi vi điều khiển sẽ có một hoặc một số thanh ghi (được gọi là cổng) được kết nối với các chân của vi điều khiển.



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)



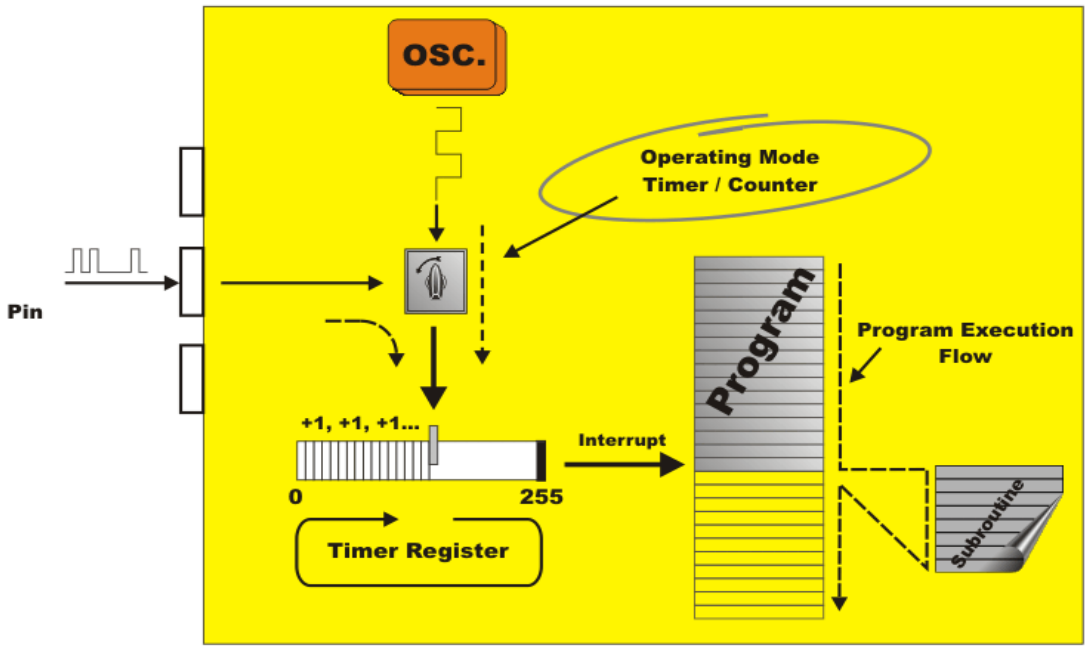
Hình 3-4 ghép nối bộ dao động

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.

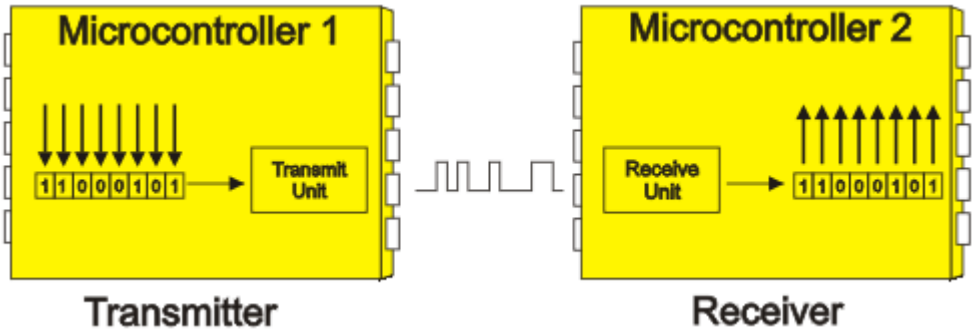
❖ Bộ định thời/đếm (Timers/Counters)

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

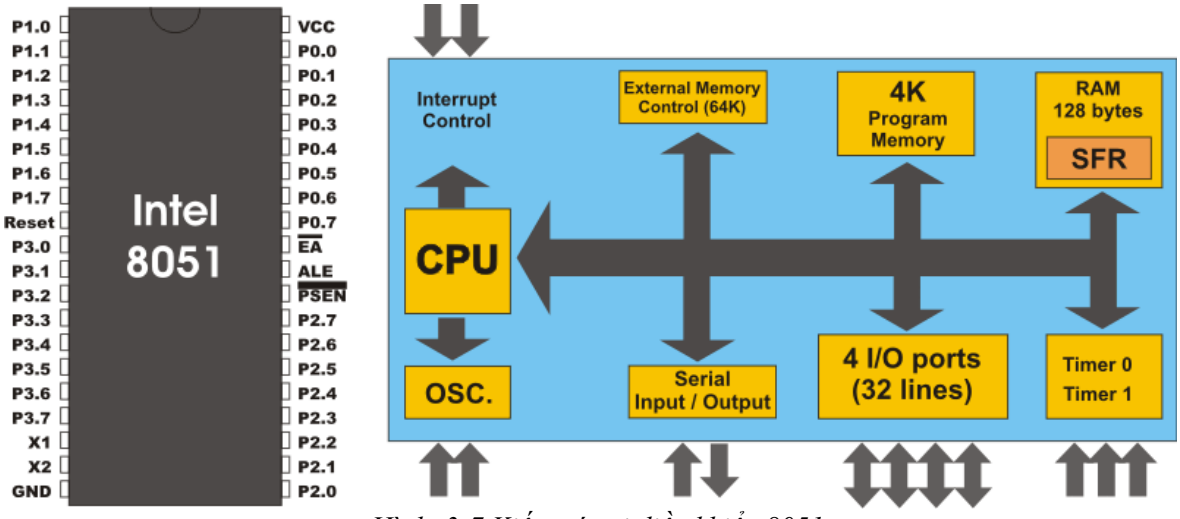
3.2 Kiến trúc vi điều khiển 8051

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.

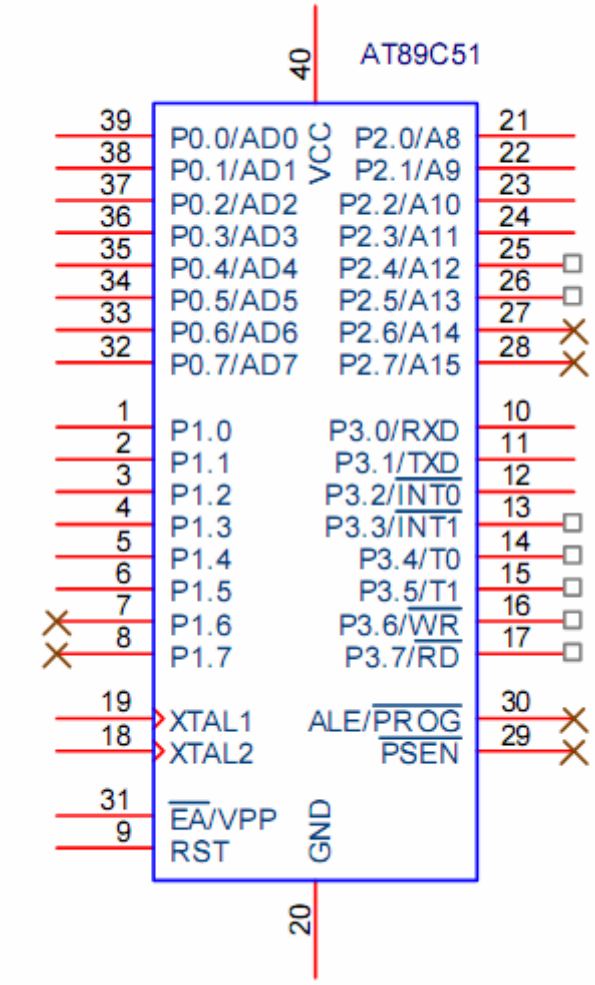


Hình 3-7.Kiến trúc vi điều khiển 8051

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.

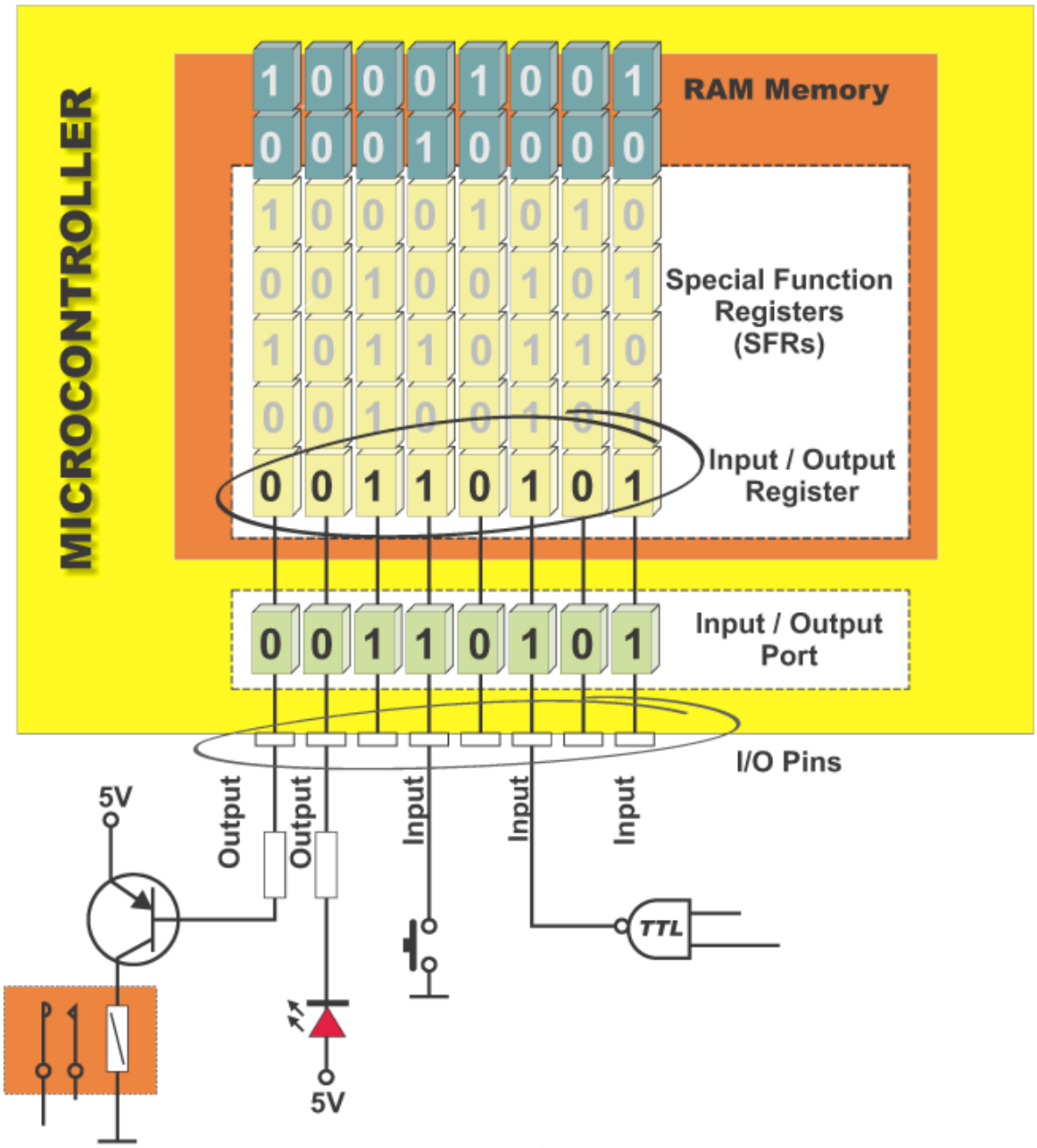
Hãy nhớ rằng : bình thường chân /PSEN sẽ được thả trống (No Connect).Chỉ khi nào cho /EA ở mức thấp thì lúc đó: /PSEN sẽ ở mức thấp trong thời gian lấy lệnh. Các mã nhị phân của chương trình được lấy từ EPROM qua bus dữ liệu và được chốt vào thanh ghi lệnh của 8951 để giải mã lệnh. /PSEN ở mức thụ động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

❖ CÁC CHÂN NGUỒN:

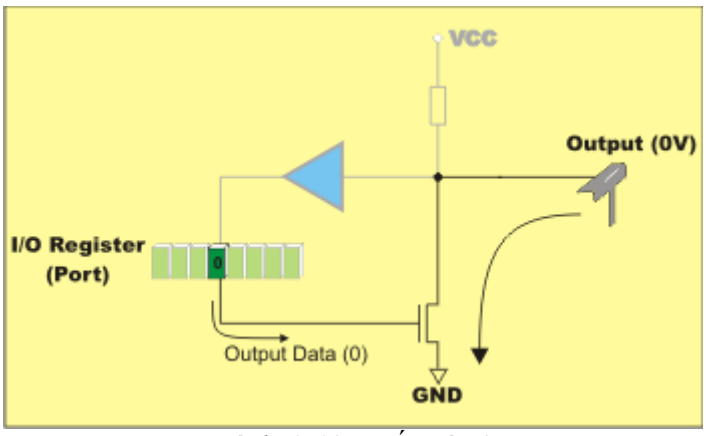
AT89C51 hoạt động ở nguồn đơn +5V. Vcc được nối vào chân 40, và Vss (GND) được nối vào chân 20.

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.



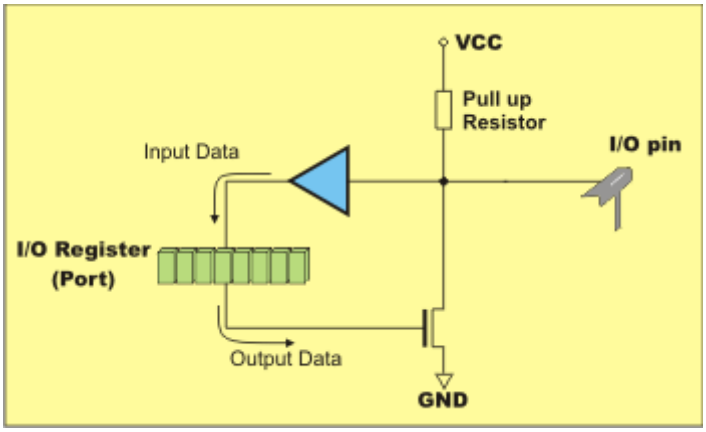
Hình 3-9. Cổng vào/ra



Hình 3-10. Xuất mức 0

❖ Chân vào/ra (I/O)

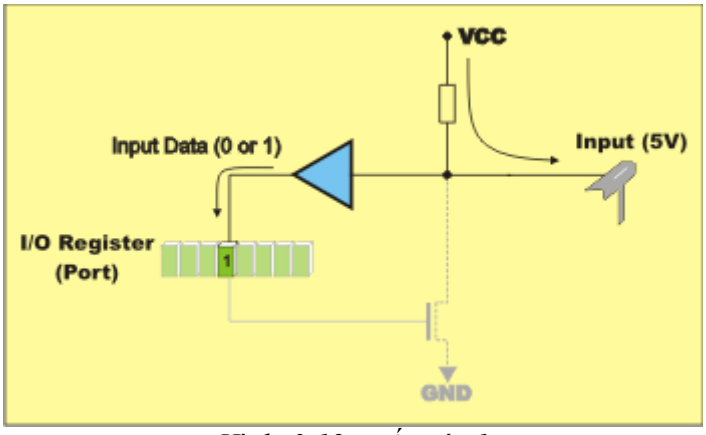
Hình trên mô tả sơ đồ đơn giản của mạch bên trong các chân vi điều khiển trừ cổng P0 là không có điện trở kéo lên (pull-up).



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) :

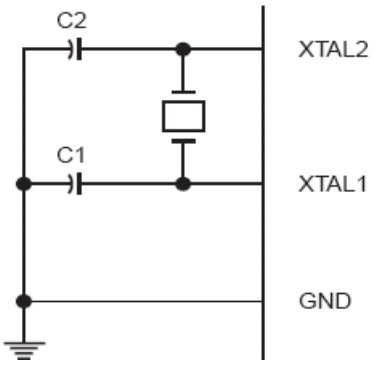
EA (chân 31) dùng để cho phép thực thi chương trình từ ROM ngoài. Khi nối chân 31 với Vcc, AT89C51 sẽ thực thi chương trình từ ROM nội (tối đa 8KB), ngược lại thì thực thi từ ROM ngoài (tối đa 64KB).
Ngoài ra, chân /EA được lấy làm chân cấp nguồn 12V khi lập trình cho ROM.

❖ 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:

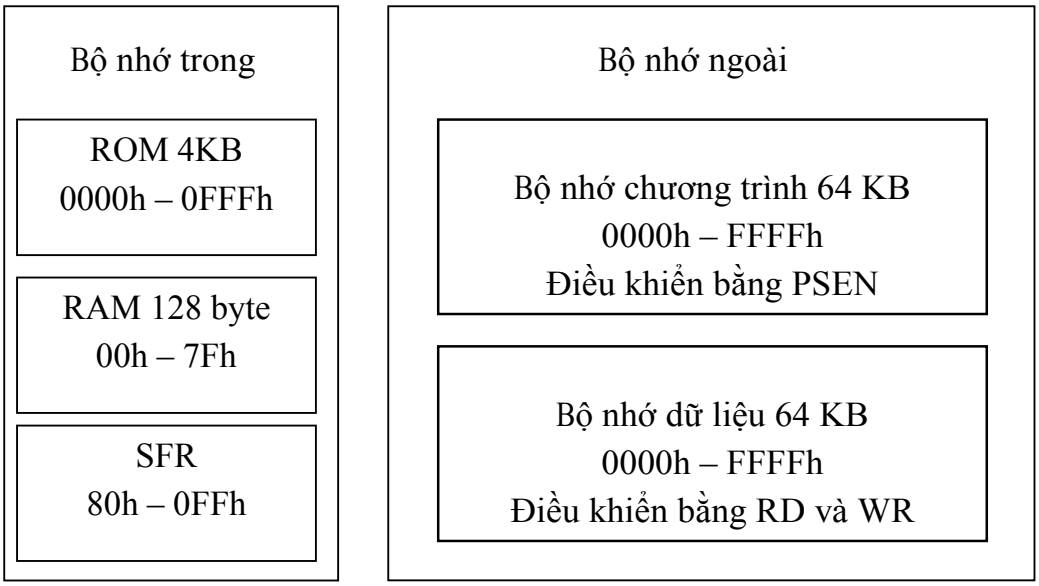
Ngõ vào và ngõ ra bộ dao động, khi sử dụng có thể chỉ cần kết nối thêm thạch anh và các tụ như hình vẽ trong sơ đồ. Tần số thạch anh thường sử dụng cho AT89C51 là 12Mhz.



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.

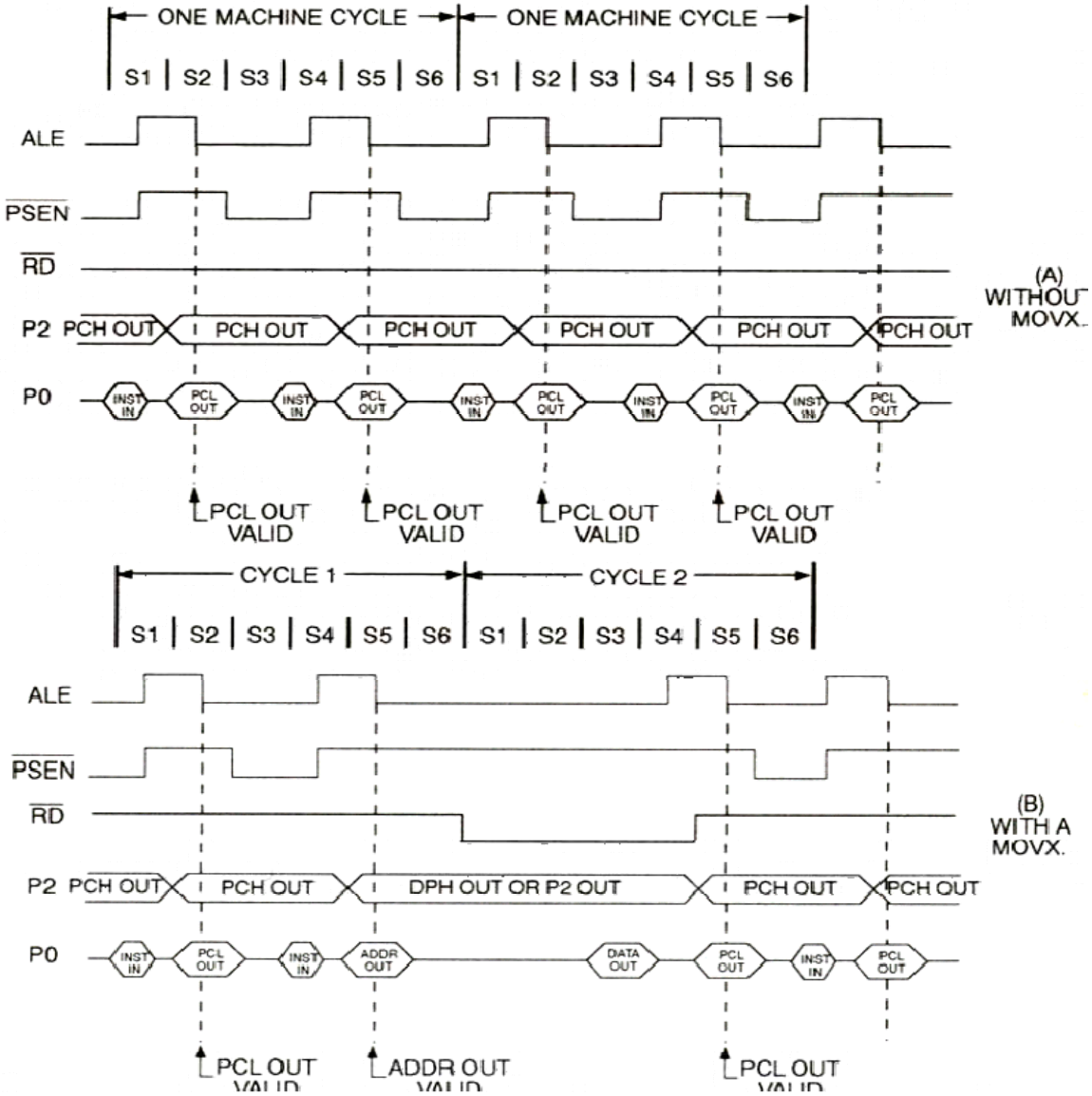
Lưu ý rằng việc truy xuất bộ nhớ chương trình luôn luôn sử dụng địa chỉ 16 bit còn bộ nhớ dữ liệu có thể là 8 bit hay 16 bit tuỳ theo câu lệnh sử dụng. Khi dùng bộ nhớ dữ liệu 8 bit thì có thể dùng Port 2 như là Port I/O thông thường còn khi dùng ở chế độ 16 bit thì Port 2 chỉ dùng làm các bit địa chỉ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

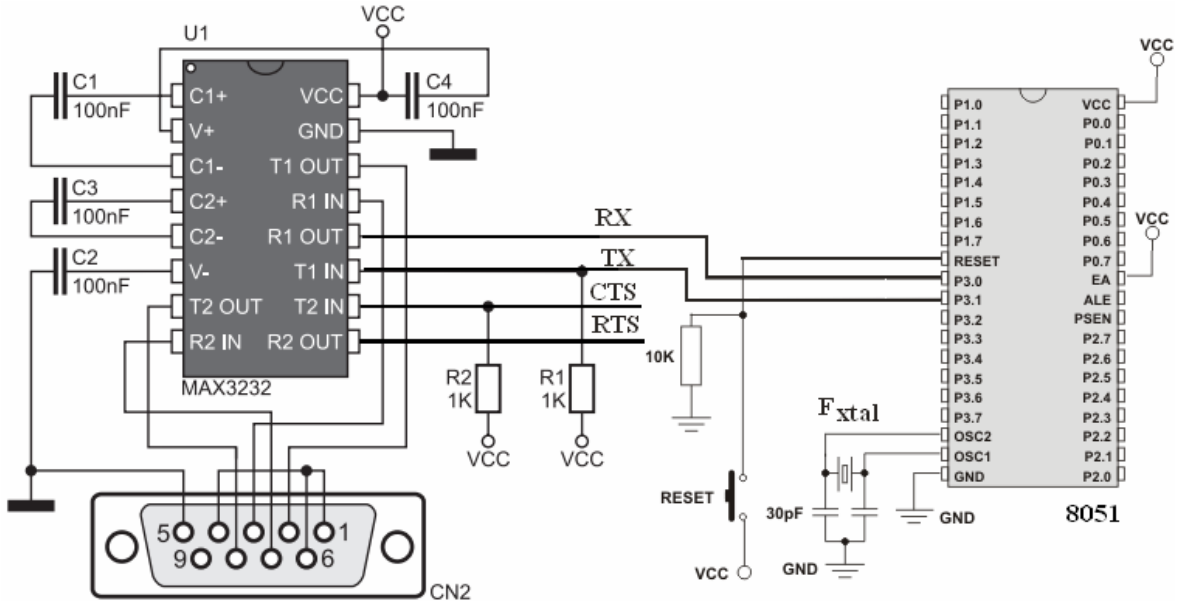
Hai timer có nguyên lý hoạt động hoàn toàn giống nhau và độc lập. Sau khi cho phép chạy, mỗi khi có thêm một xung tại đầu vào đếm, giá trị của timer sẽ tự động được tăng lên 1 đơn vị, cứ như vậy cho đến khi giá trị tăng lên vượt quá giá trị 65535 mà thanh ghi đếm có thể biểu diễn thì giá trị đếm lại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

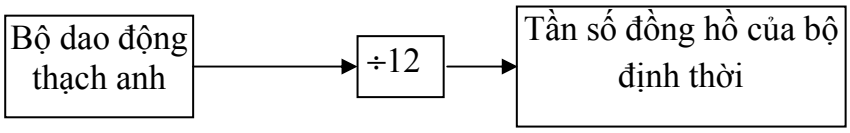
❖ Nguồn xung đồng hồ cho bộ định thời:

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:

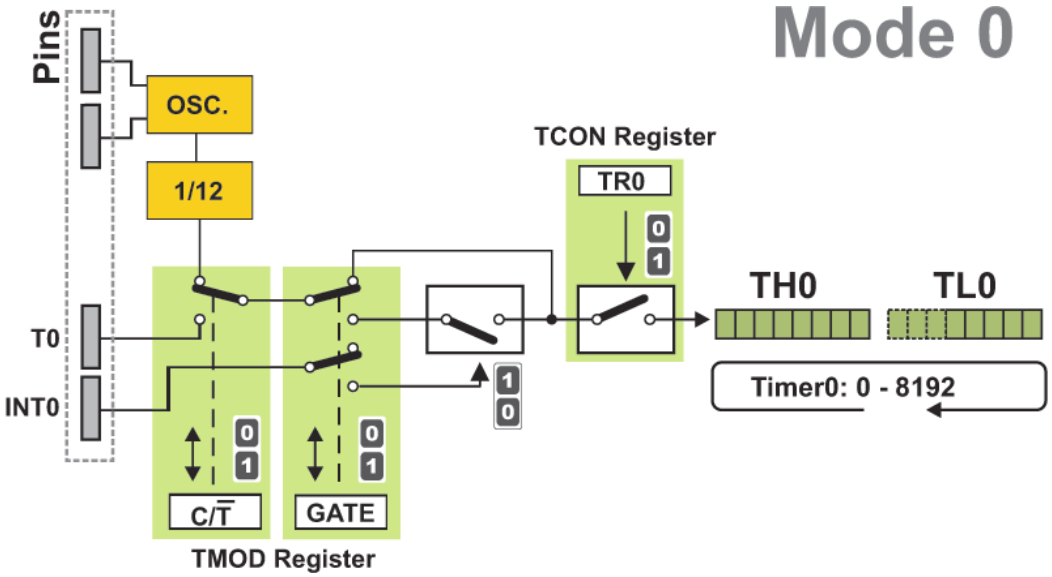
Tìm giá trị cho TMOD nếu ta muốn lập trình bộ Timer0 ở chế độ 2 sử dụng thạch anh XTAL 8051 làm nguồn đồng hồ và sử dụng các lệnh để khởi động và dừng bộ định thời.

Lời giải:

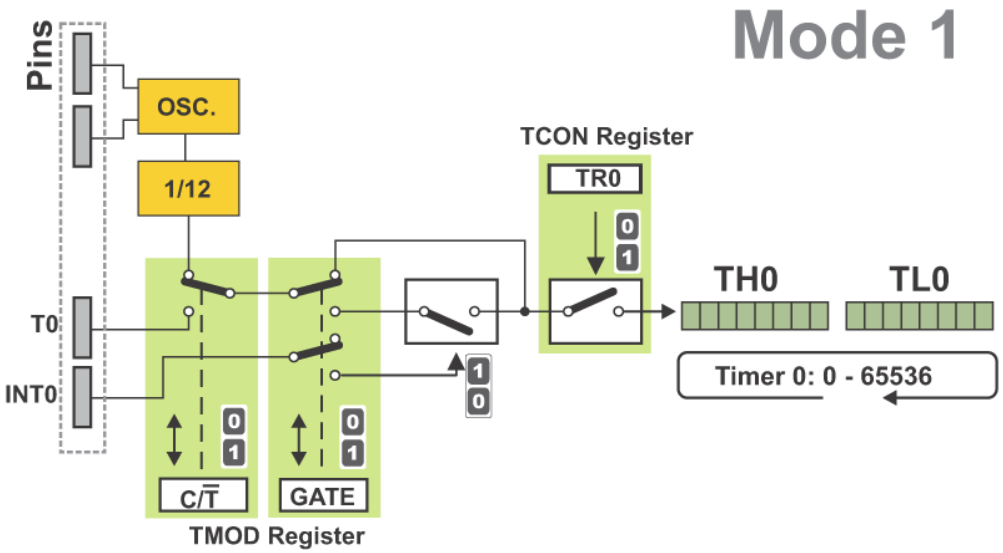
TMOD = 0000 0010: Bộ định thời Timer0, chế độ 2 C/T = 0 dùng nguồn XTAL GATE = 0 để dùng phần mềm trong để khởi động và dừng bộ định thời.

❖ **Các chế độ của bộ đếm/định thời (Timer Mode)**

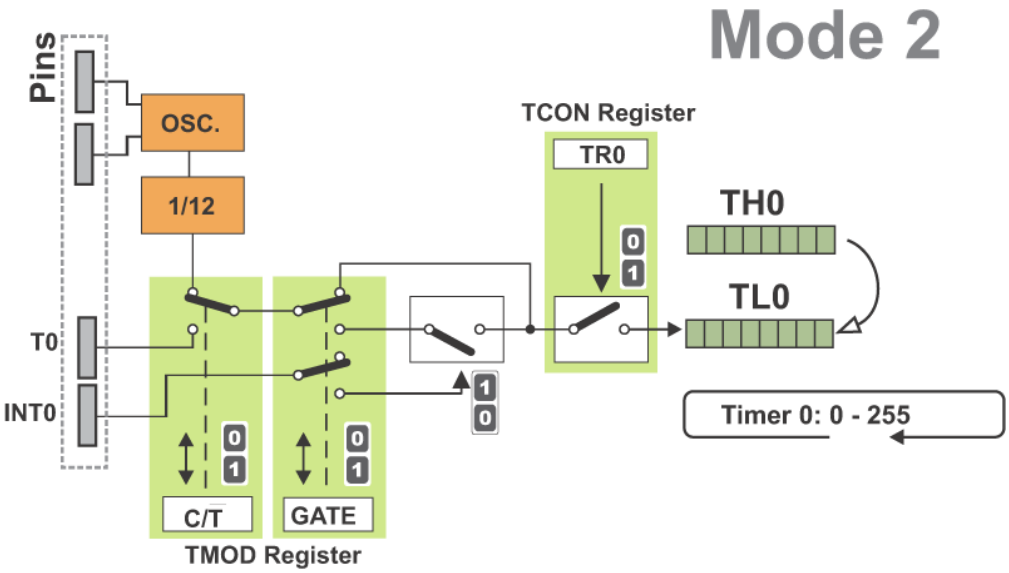
Như vậy, bây giờ chúng ta đã có hiểu biết cơ bản về vai trò của thanh ghi TMOD, chúng ta sẽ xét chế độ của bộ định thời và cách chúng được lập trình như thế nào để tạo ra một độ trễ thời gian. Do chế độ 1 và chế độ 2 được sử dụng rộng rãi nên ta đi xét chi tiết từng chế độ một.



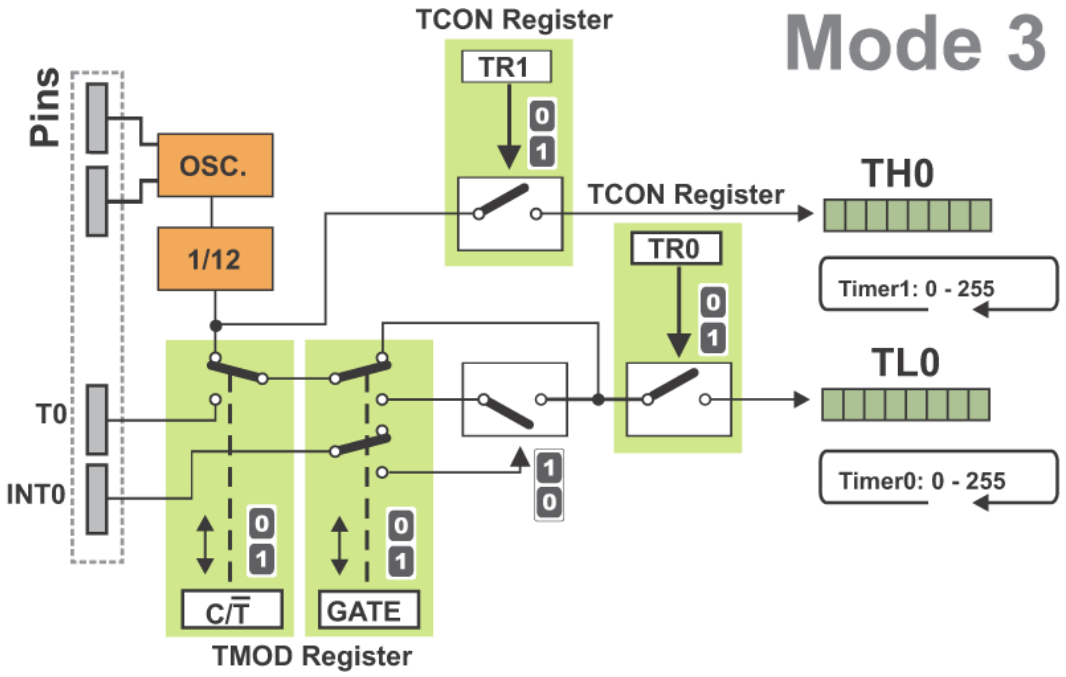
Hình 3-20. Timer 0 – Mode 0



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
```

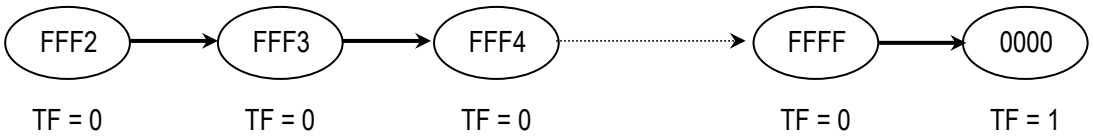
Lời giải:

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
```

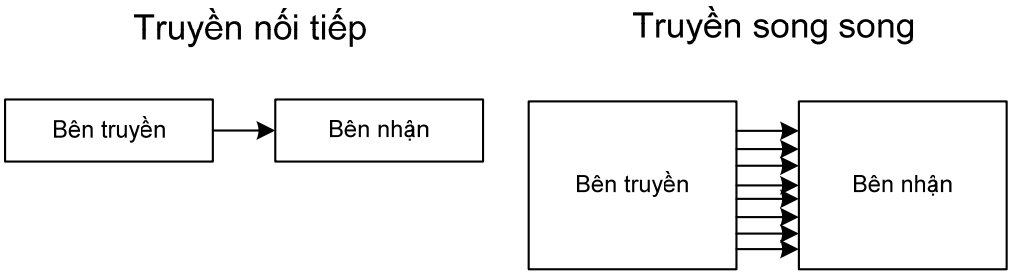
```
          MOV   TL1, #08          ; Nạp byte thấp TL1 = 08
          MOV   TH1, #08          ; Nạp byte cao TH1 = 01
          SETB  TR1               ; Khởi động Timer1
BACK:     JNB   TF1, BACK  ; giữ nguyên cho đến khi
                           ;bộ định thời quay về 0
          CLR   TR1               ; 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".

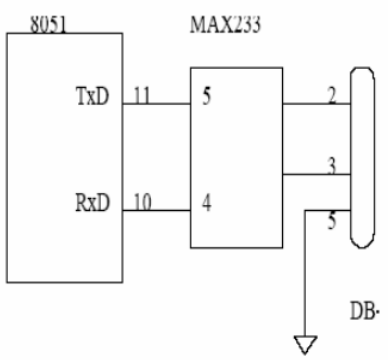
8051 có hai chân được dùng chuyên cho truyền và nhận dữ liệu nối tiếp. Hai chân này được gọi là TxD và RxD và là một phần của cổng P3 (đó là P3.0 và P3.1). chân 11 của 8051 là P3.1 được gán cho TxD và chân 10 (P3.0) được dùng cho RxD. Các chân này tương thích với mức lô-gích TTL. Do vậy chúng đòi hỏi 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, ...**

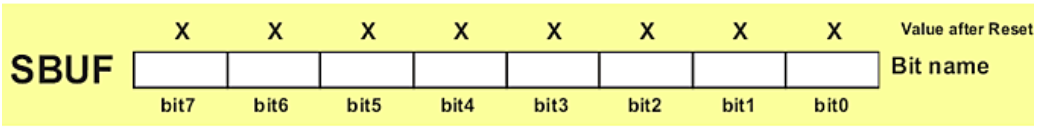
8051 có 1 cổng UART làm việc ở chuẩn TTL, mặc định sau khi khởi động tất các cổng của 8051 đều làm việc ở chế độ vào ra số, vì thế để có thể sử dụng UART cần phải cấu hình cho cổng này làm việc thông qua các thanh ghi điều khiển và ghép nối tương thích với chuẩn rs232.



Hình 3-25. Ghép nối RS232 với 8051

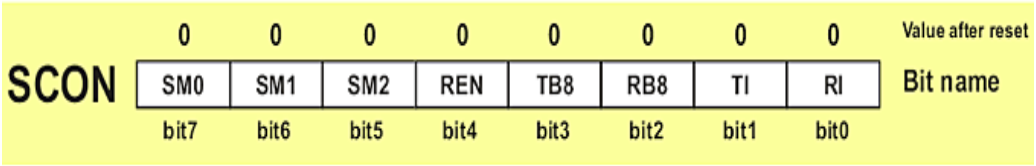
Các thanh ghi điều khiển trong chế dộ UART:

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

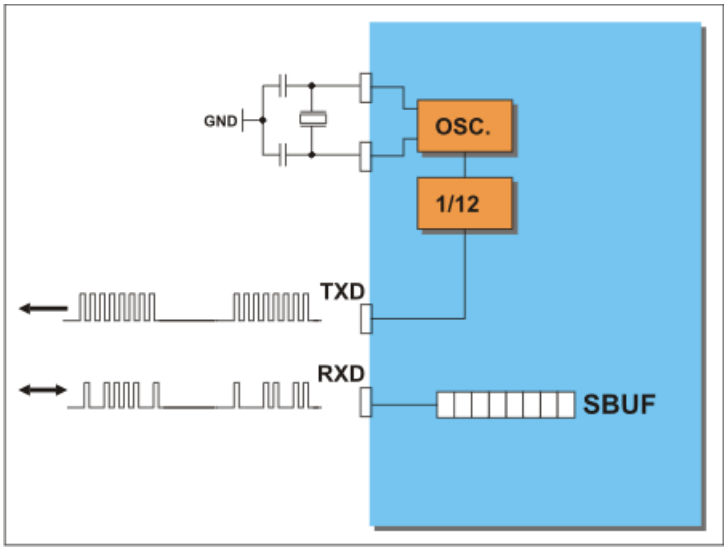
TI	Cờ ngắt truyền. Khi một byte trong SBUF được truyền thành công thì TI=1. Trước khi truyền byte khác bit này cần phải được xóa bằng phần mềm
RI	Cờ ngắt nhận, Khi nhần thành công 1 byte vào SBUF thì RI=1. Sau khi đọc SUBF, RI cần phaiđược xóa bằng phần mềm

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

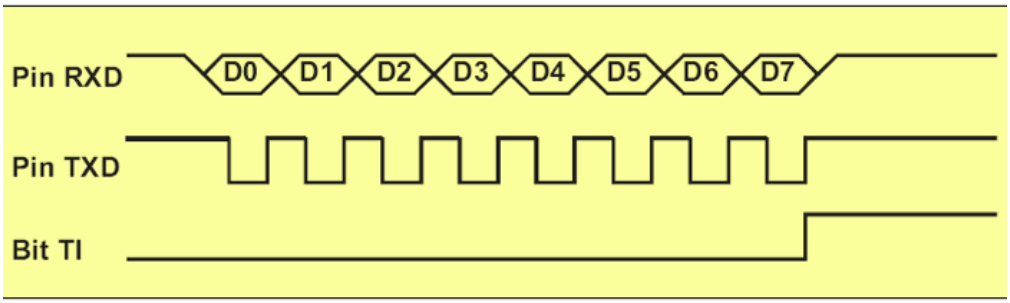
❖ Mode 0

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.



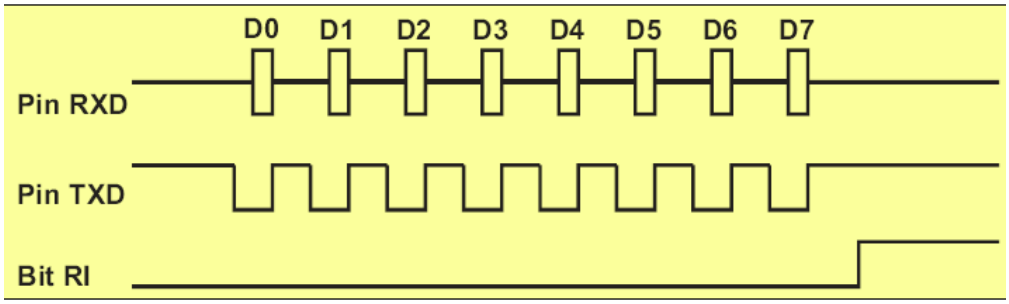
Hình 3-26. Truyền thông nối tiếp – Mode 0

- 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

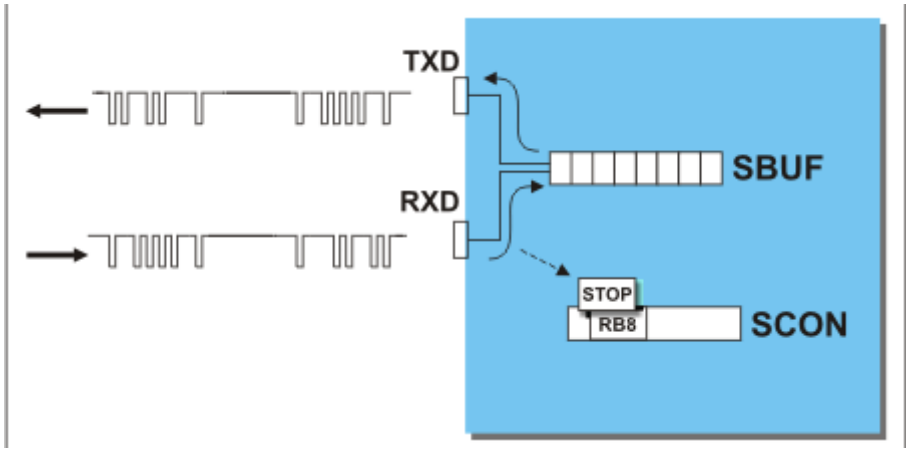
- Quá trình nhận tự động bởi hệ thống và kết thúc khi RI=1



Hình 3-28. Giản đồ thời gian nhận nối tiếp – Mode 0

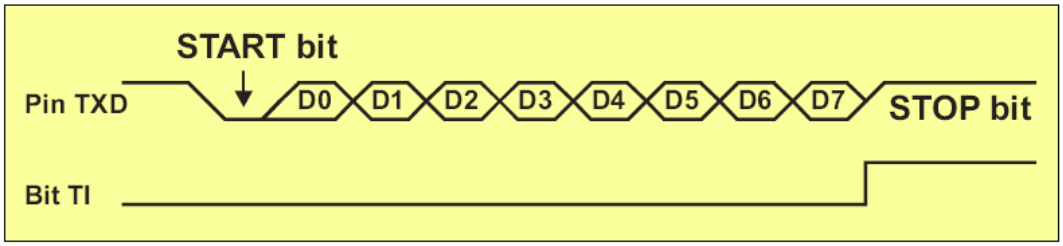
❖ Mode 1

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



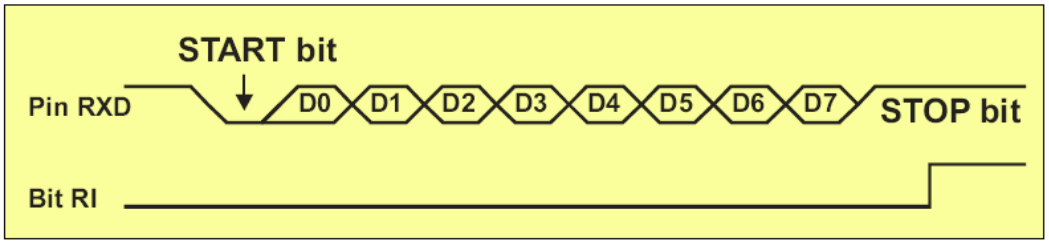
Hình 3-29. Truyền nhận nối tiếp – Mode 1

- Quá trình truyền:



Hình 3-30. Giản đồ thời gian truyền nối tiếp – Mode 1

- Quá trình nhận

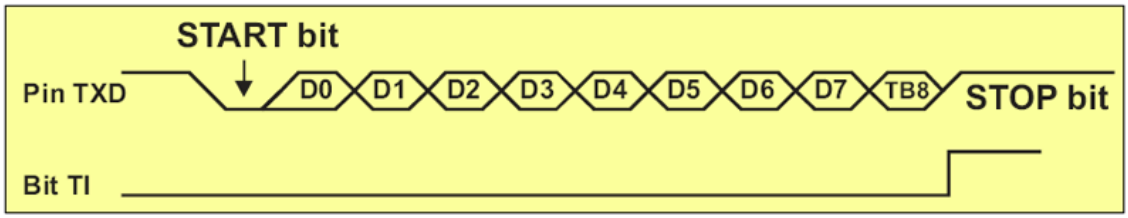


Hình 3-31. Giản đồ thời gian nhận nối tiếp – Mode 1

❖ Mode 2

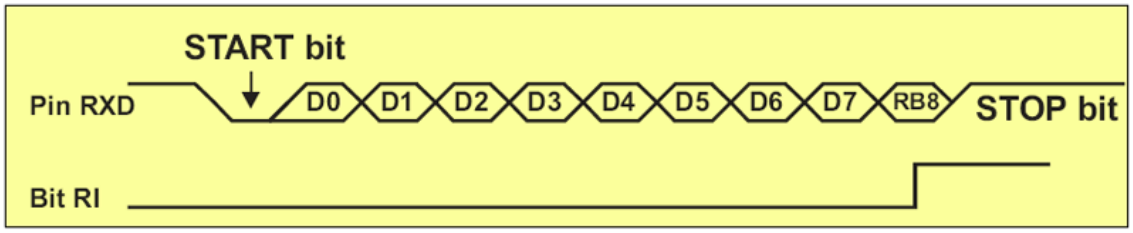
Truyền thông bất đồng bộ với frame truyền 11 bit, gồm 1 start, 8 bit dữ liệu, 1 bit lập trình được(nêu truyền là TB8, nhận là RB8) và 1 bit stop. TxD thực hiện truyền, RxD nhận dữ liệu, tốc độ truyền cài đặt qua Timer 1. Bit thứ 9 thường được dùng là bit phát hiện lỗi party.

- 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.

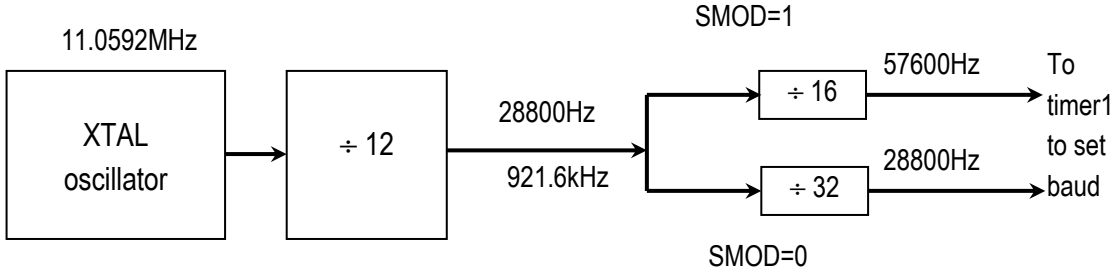
a) 9600 b) 4800 nếu SMOD = 1 và tần số XTAL = 11.0592MHz

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.

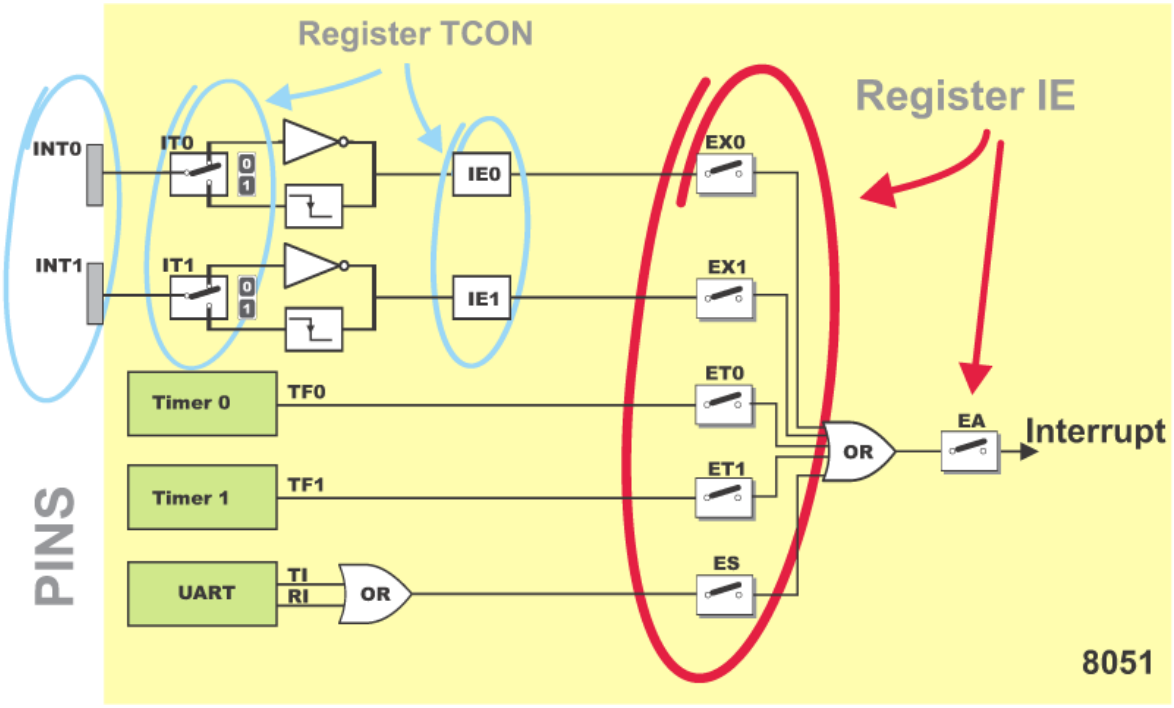
Ví dụ trong các bộ định thời, ta đã dùng lệnh "JNB TF, đích" và đợi cho đến khi bộ định thời quay trở về 0. Trong ví dụ đó, trong khi chờ đợi thì ta có thể làm việc được gì khác có ích hơn, chẳng hạn như khi sử dụng phương pháp ngắt thì bộ vi điều khiển có thể đi làm các việc khác và khi cờ TF bật lên 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.

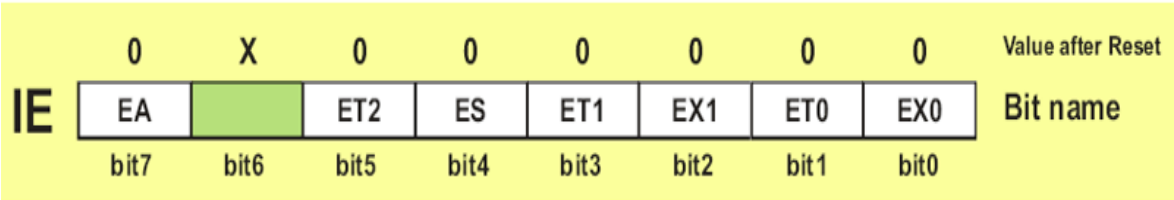
Các tín hiệu điều khiển ngắt có thể được mô tả như hình dưới



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.

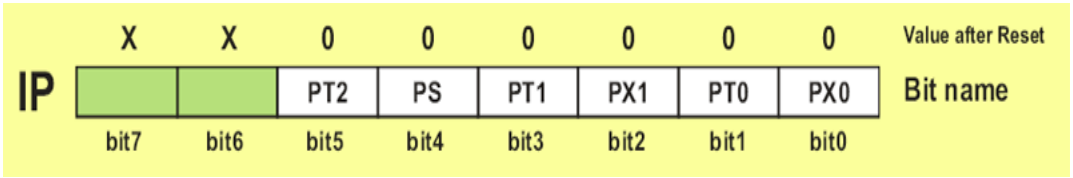
Có 3 cấp độ ưu tiên ngắt trong 8051

- 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:

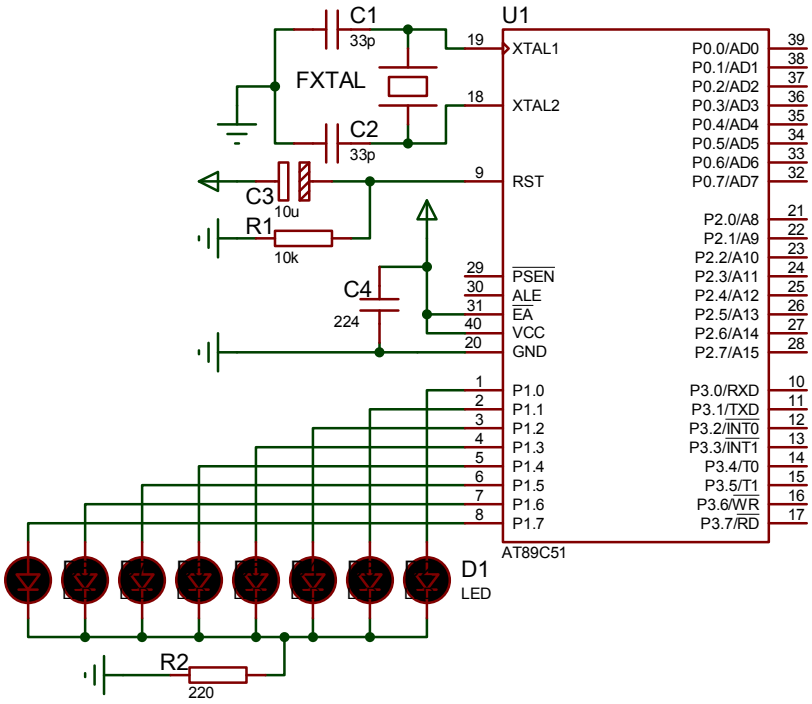
Ứng dụng các vi điều khiển đó để lập trình giao tiếp với thế giới thực thông qua các ví dụ:

- *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:

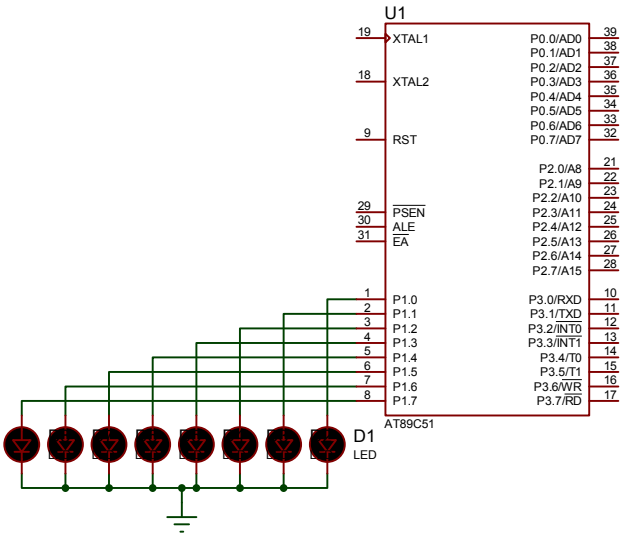


Hình 4-1. Mạch nhấp nháy LED đơn

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
```

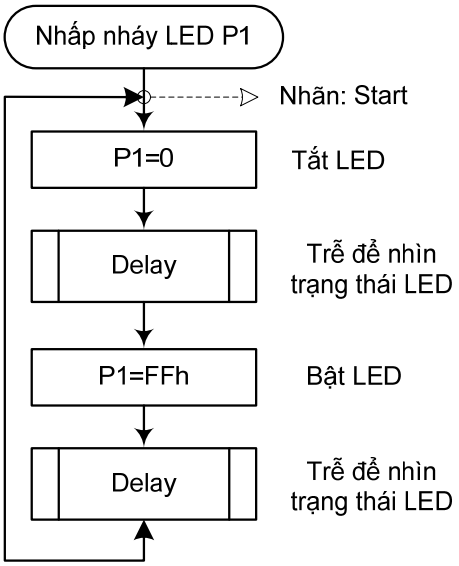
Mã nguồn 4-2. DelayX

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.

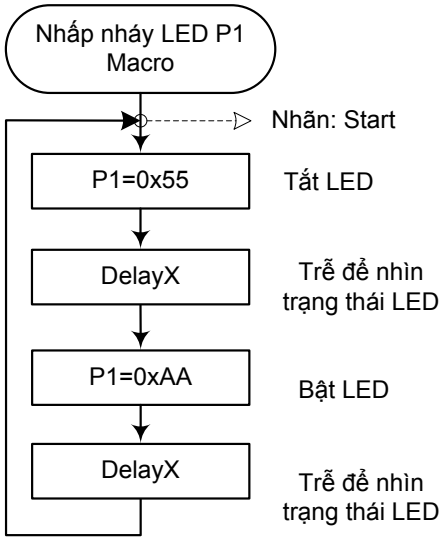
Sơ đồ như *"Hình 4-1. Mạch nhấp nháy LED đơn"*, sơ đồ thuật toán và mã nguồn như hình dưới đây:



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
```

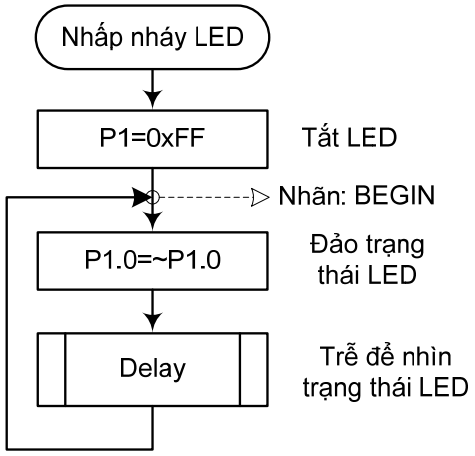
Mã nguồn 4-4. Nhấp nháy cổng P1và đảo trạng thái P2.0

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

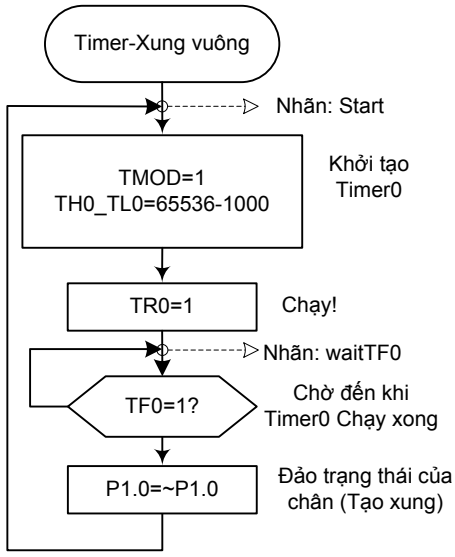
Chương trình dưới đây minh họa một ví dụ đơn giản nhất về Timer, lấy ví dụ là Timer 0 (vì trong chế độ cơ bản, Timer 0 và Timer 1 là như nhau).

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*

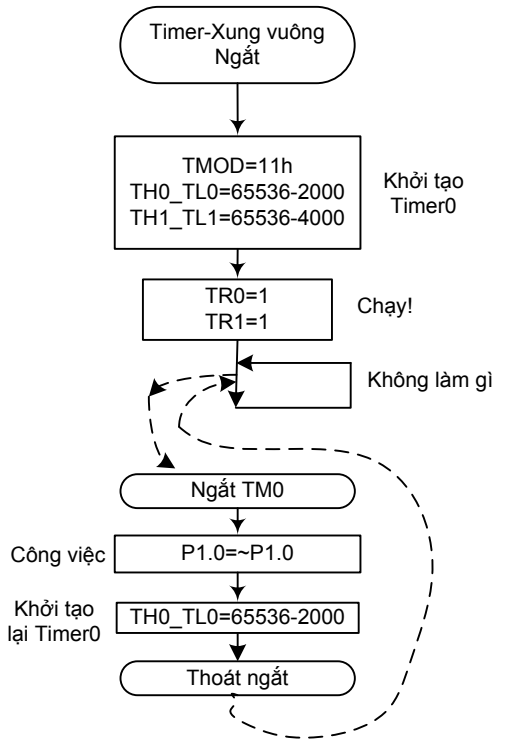
♦ **Timer – Bài toán 2:**

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)

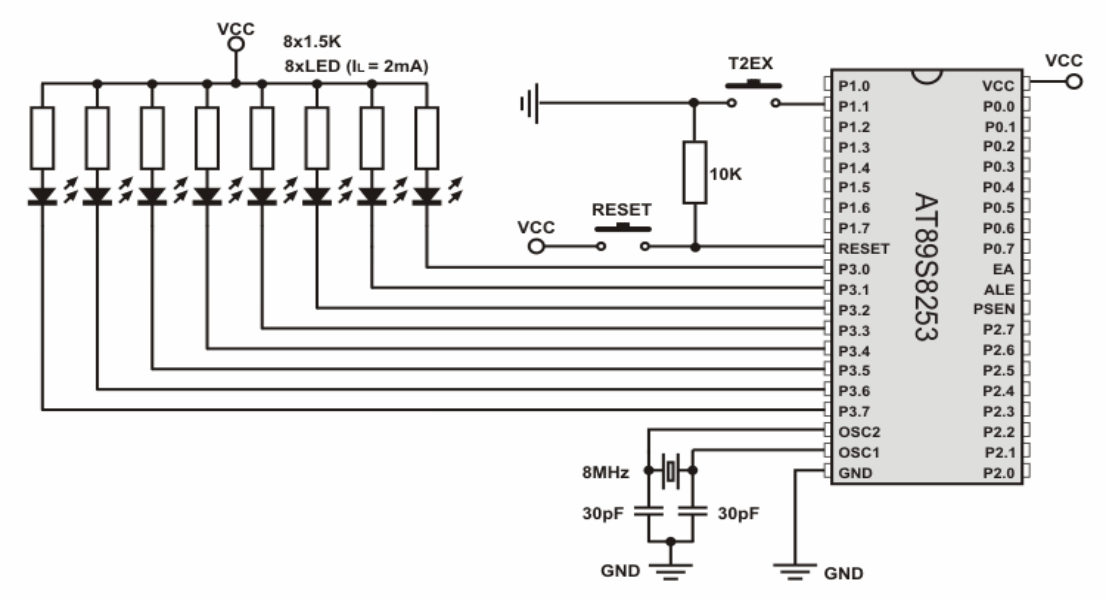
```
CSEG    AT    0
JMP     Start  ; Reset vector
   ORG 0BH
     JMP TM0_PWM ;Vector ngắt TM0
    ORG     01BH
     JMP TM1_PWM ;Vector ngắt TM1
    ORG     100H
Start:
   MOV   TMOD,#0x11
   MOV     TH0,#HIGH(-2000)
   MOV     TL0,#LOW(-2000)
   MOV     TH1,#HIGH(-4000)
   MOV     TL1,#LOW(-4000)
   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.

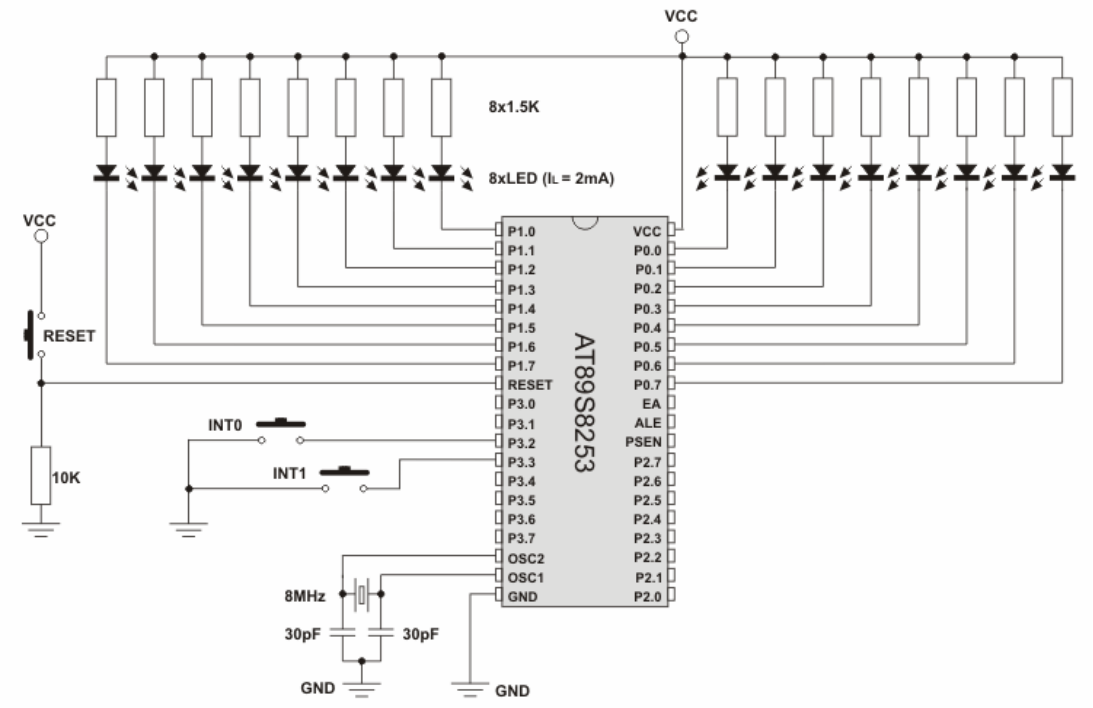
```
CSEG     AT       0
JMP      XRESET              ; Reset vector
         ORG      02BH        ; Vector ngắt của TM2
         JMP      TIM2_ISR
ORG      100H
XRESET:
         MOV      A,#0FFH
         MOV      P3,#0FFH
         MOV      RCAP2L,#0FH ; TM2, 16-bit tự nạp lại
         MOV      RCAP2L,#01H
         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

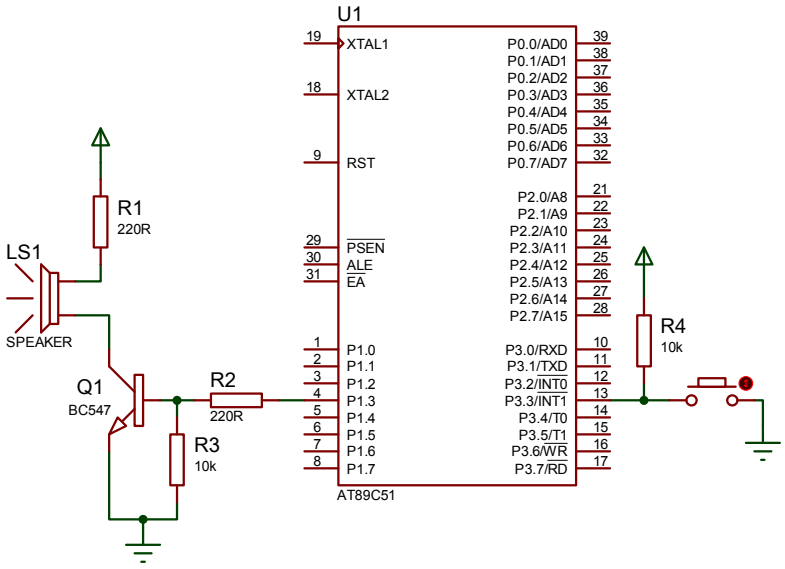
```
        CSEG     AT     0
        JMP      XRESET; Reset vector
        ORG      003H   ; Vector ngắt INT0
        JMP      Isr_Int0
        ORG 013H        ; Vector ngắt INT1
        JMP      Isr_Int1
ORG      100H
XRESET:
  MOV TCON,#00000101B  ; Ngắt INT0 (P3.2)và ngắt INT1 (P3.3) xảy
```

```
                        ;  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

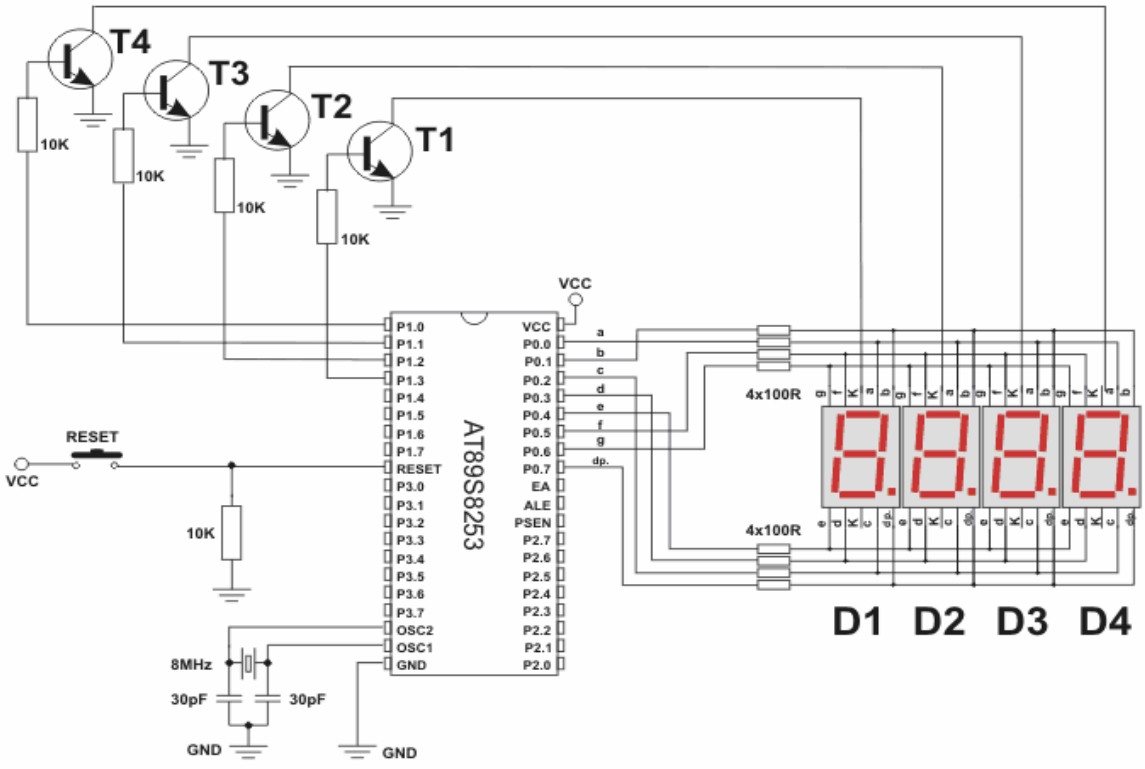
4.6 Sử dụng LED 7 thanh

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

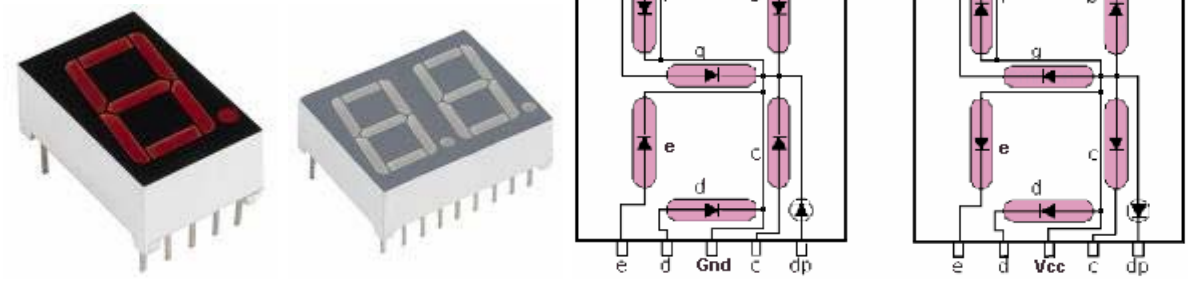
```
;*****************************************************************
;* PROGRAM NAME : 7Seg1.ASM
;* DESCRIPTION: Program displays number "3" on 7-segment LED
display
;*****************************************************************
          CSEG     AT     0
          JMP      XRESET ; Reset vector
          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
```

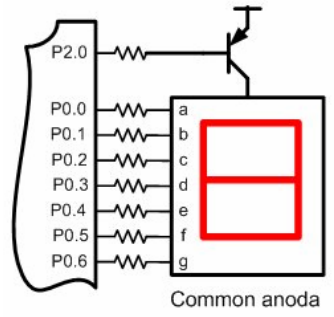
Mã nguồn 4-11. Hiển thị LED 7 thanh -1

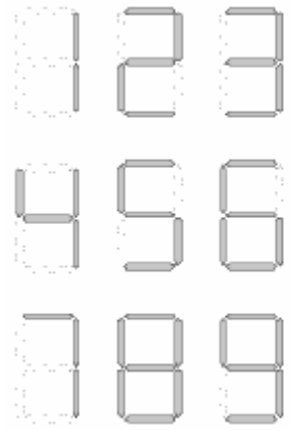
4.6.2 Hiển thị trên nhiều LED 7 thanh

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

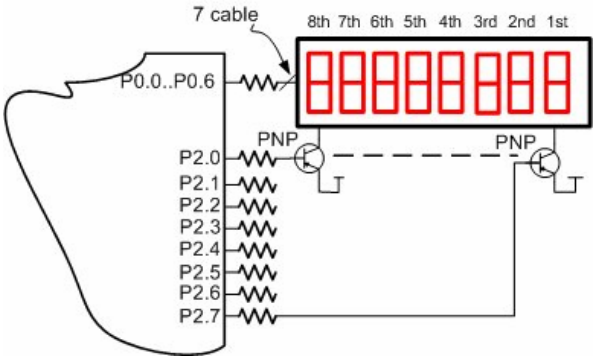
Ví dụ:

```
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:



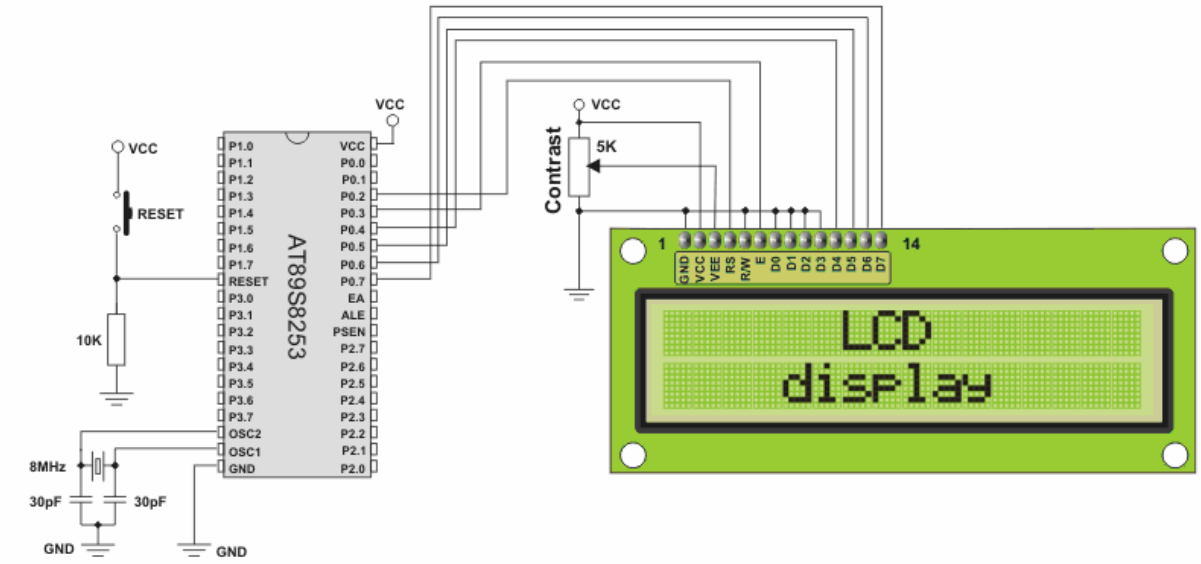

```
org 0h
start:
      mov   dptr, #word      ;để con trỏ dữ liệu vào đầu bảng
      mov   R6,   #8    ; số led cần hiển thị, 8 led
      mov   R1,   #01111111b; khởi đầu ở led 8
Again:
      clr   A ; xóa thanh ghi acc
      movc  A,    @A+dptr ; đưa số đầu tiên ở bảng vào acc
      inc   dptr        ; 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

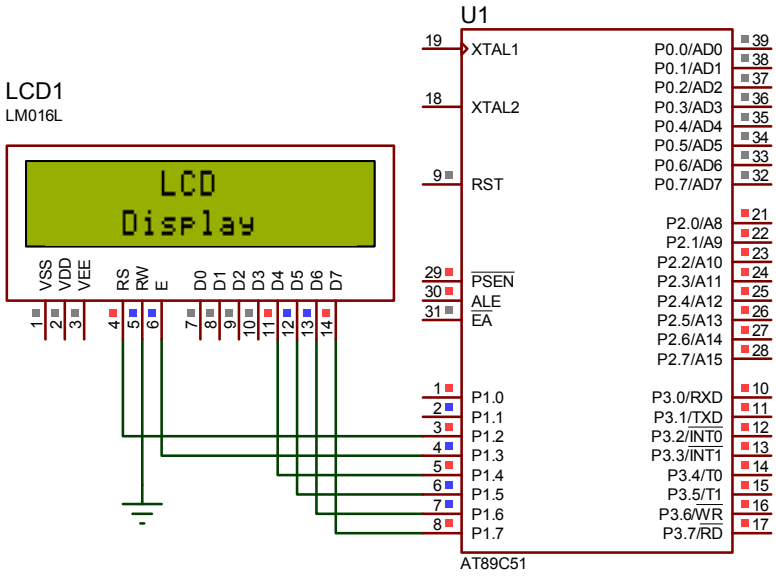
4.7 Thông báo bằng văn bản trên màn hình LCD

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
```

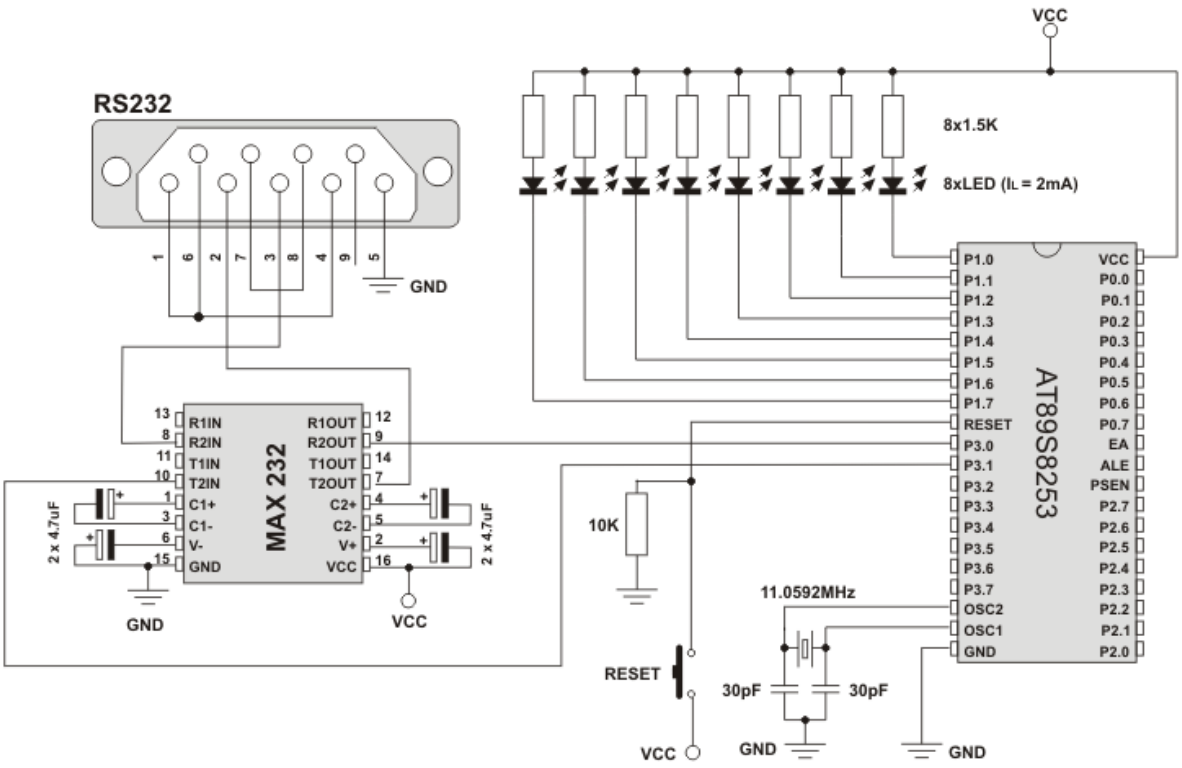
```
                        ; is generated on the LCD's EN pin
      CLR   LCD_enable
      POP   B
      POP   ACC
RET
END      ; End of program
```

Mã nguồn 4-14. Hiển thị LCD

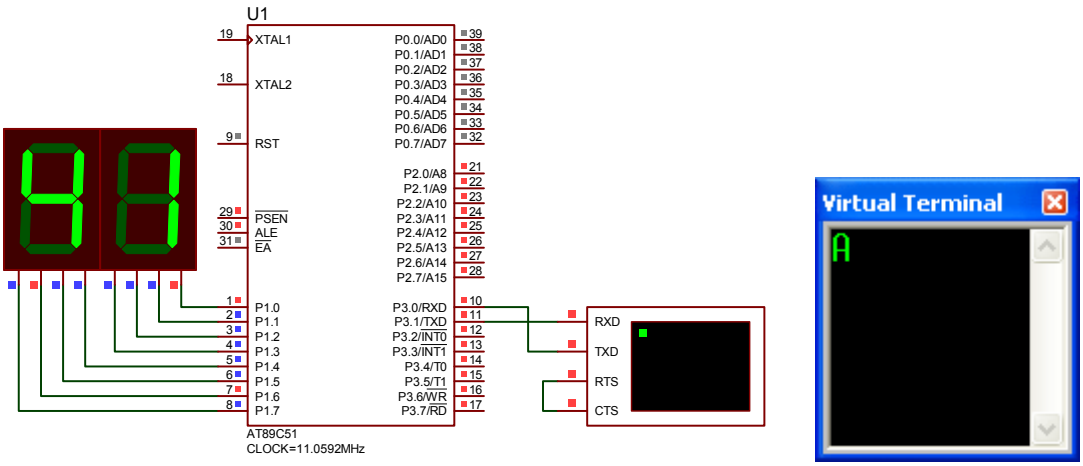
4.8 Nhận dữ liệu qua UART

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



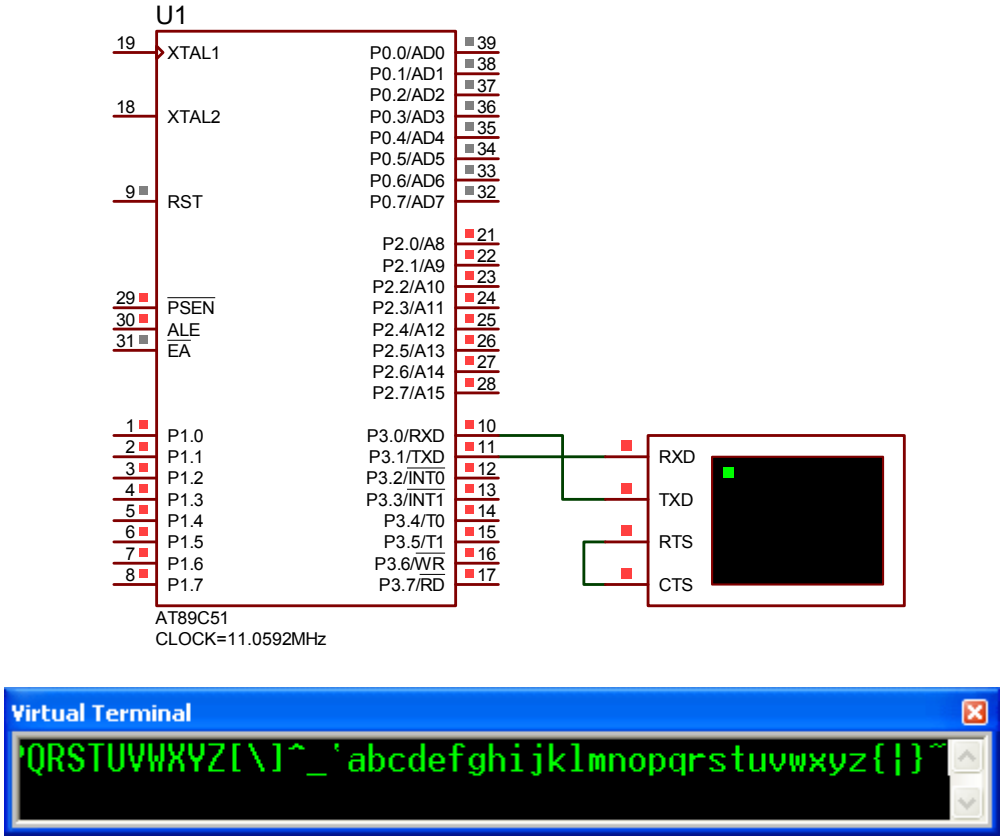
Hình 4-16. Nhận dữ liệu nối tiếp – mô phỏng

```
;*************************************************************
;* 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

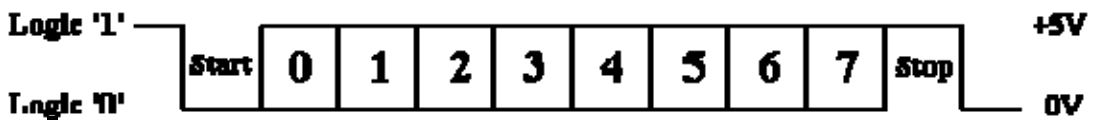
4.10 Chương trình con phục vụ truyền thông nối tiếp

```
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
```

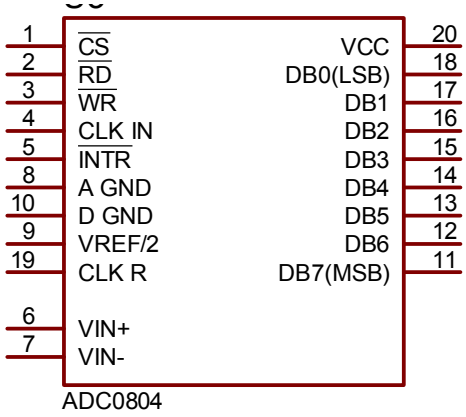
```
        DJNZ R1,getc1           ;read 8 bits
        mov r7,a
        POP PSW
        pop ACC
        RET                     ;go home
```

Mã nguồn 4-18. CTC truyền thông nối tiếp bằng phần mềm

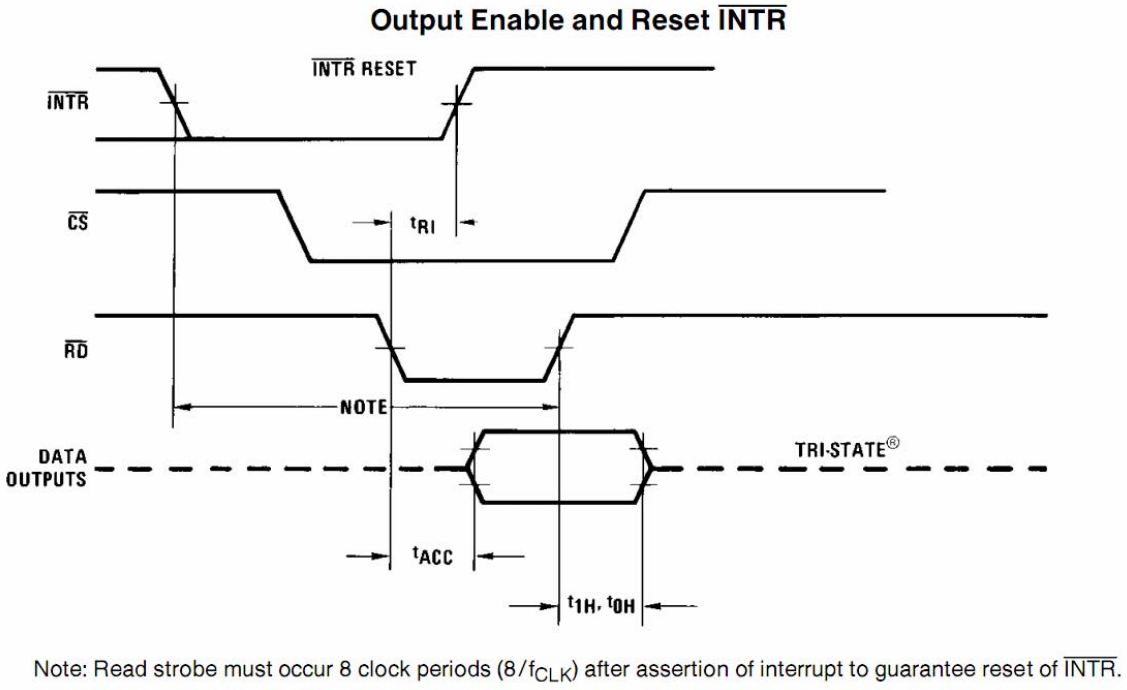
4.12 Ghép nối 8051 với ADC0804, chuyển đổi ADC 8-bit

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

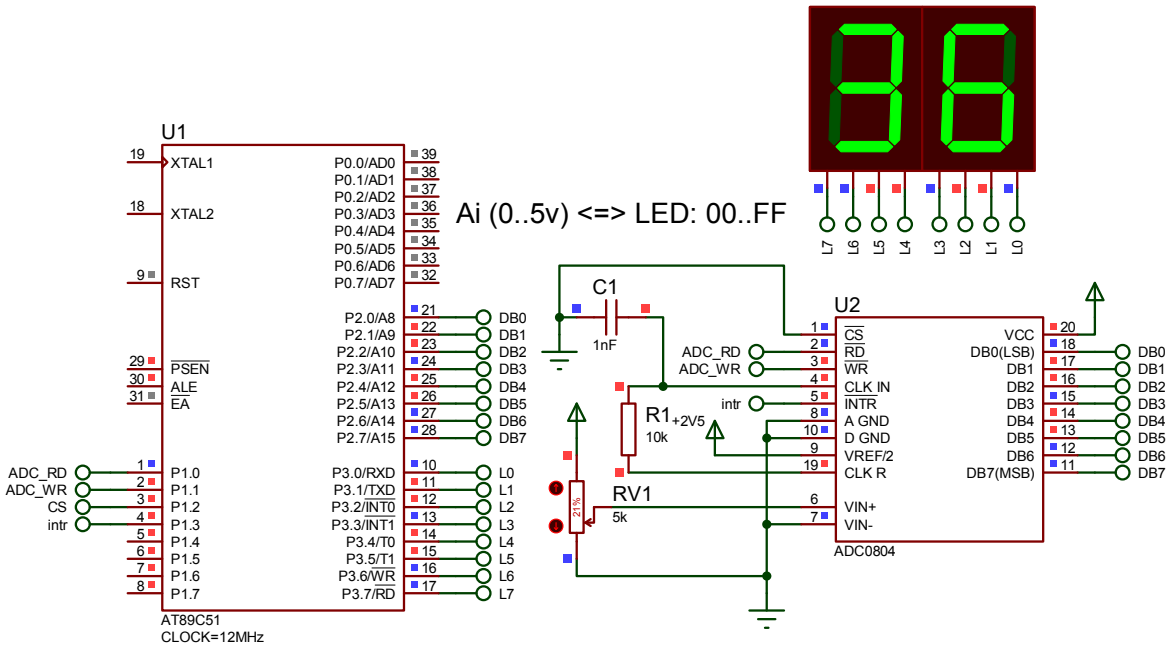


Hình 4-18. Sơ đồ chân ADC0804



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:

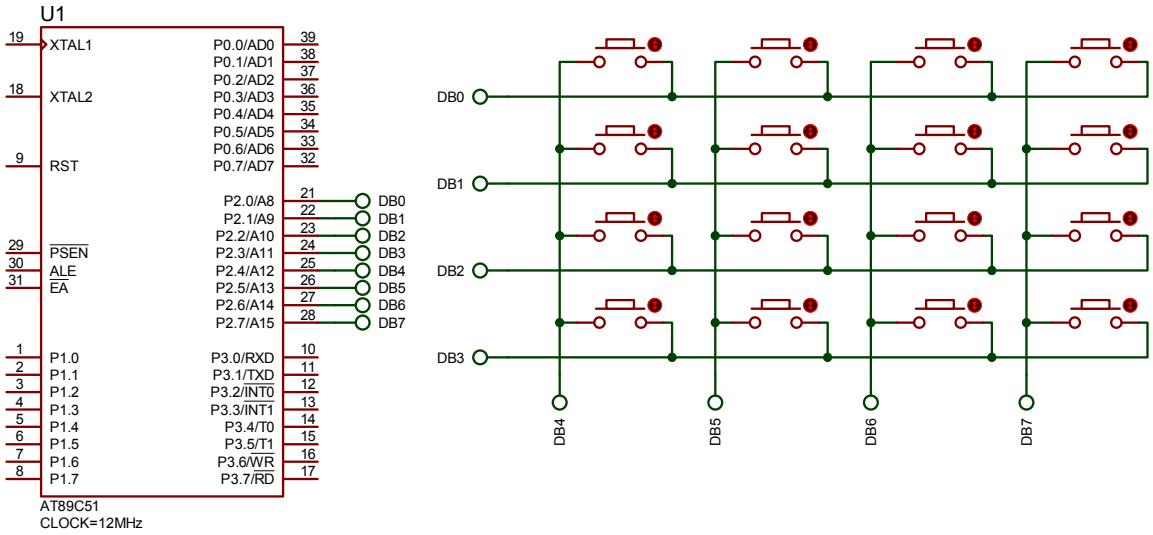


Hình 4-20. Mạch nguyên lý mô phỏng chuyển đổi ADC0804

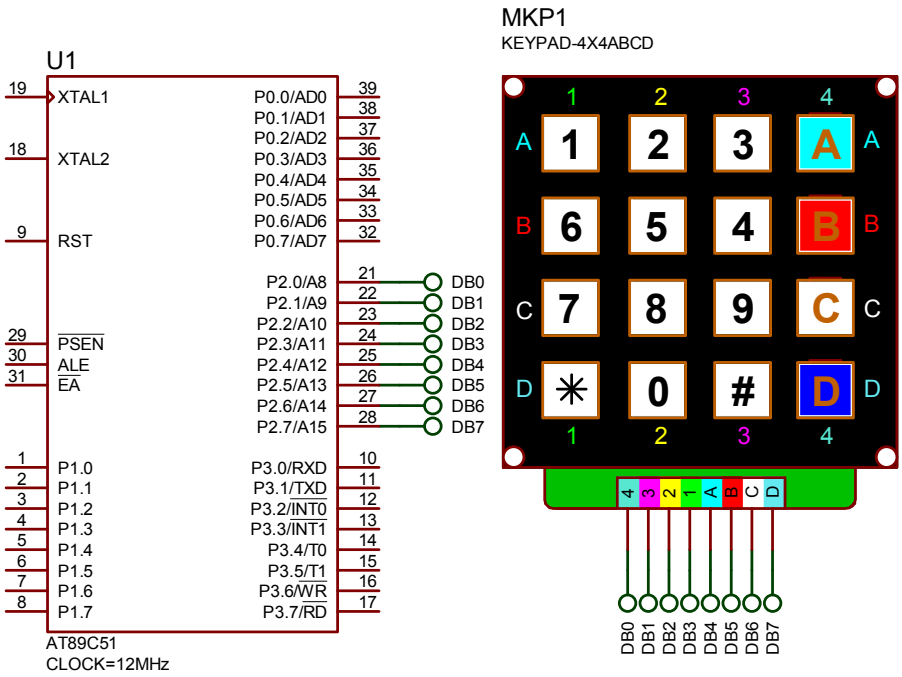
```
ORG 0h
JMP   MAIN
ADC_RD  EQU P1.0
ADC_WR  EQU P1.1
INTR    EQU P1.3
ADC_DAT EQU P2
LED7 EQU P3
        ;Khai báo chương trình con DelayX ở đây
        ; tham khảo "Mã nguồn 4-2. DelayX"
ORG     30H
MAIN:
     ACALL   TACT_LayMau
     mov LED7,A
SJMP    MAIN
TACT_LayMau:
     CLR ADC_WR        ; 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
```

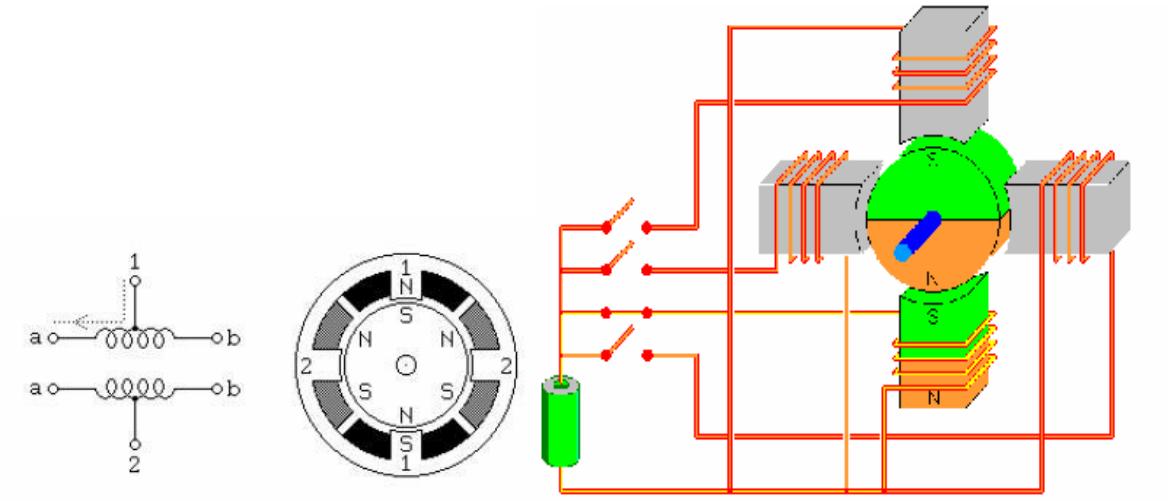
```
      CLR ROW_D
            DDB0:JB COL1, DDB1
                MOV KQ,#'*'    // Phim * bam
            DDB1:JB COL2, DDB2
                MOV KQ,#0    // Phim 0 bam
            DDB2:JB COL3, DDB3
                MOV KQ,#'#'    //Phim # bam
            DDB3:JB COL4, DFINISH
                MOV KQ,#'D'    //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)

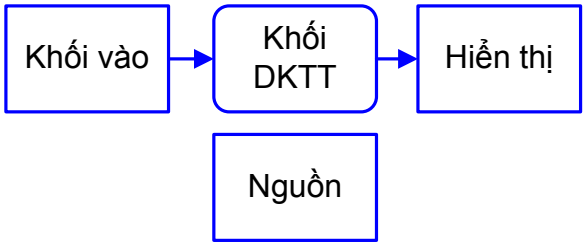
Biết cảm biến áp suất có tín hiệu ra trong khoảng 0..100mV tương ứng với áp suất từ 0..3000 atmosphe.

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.

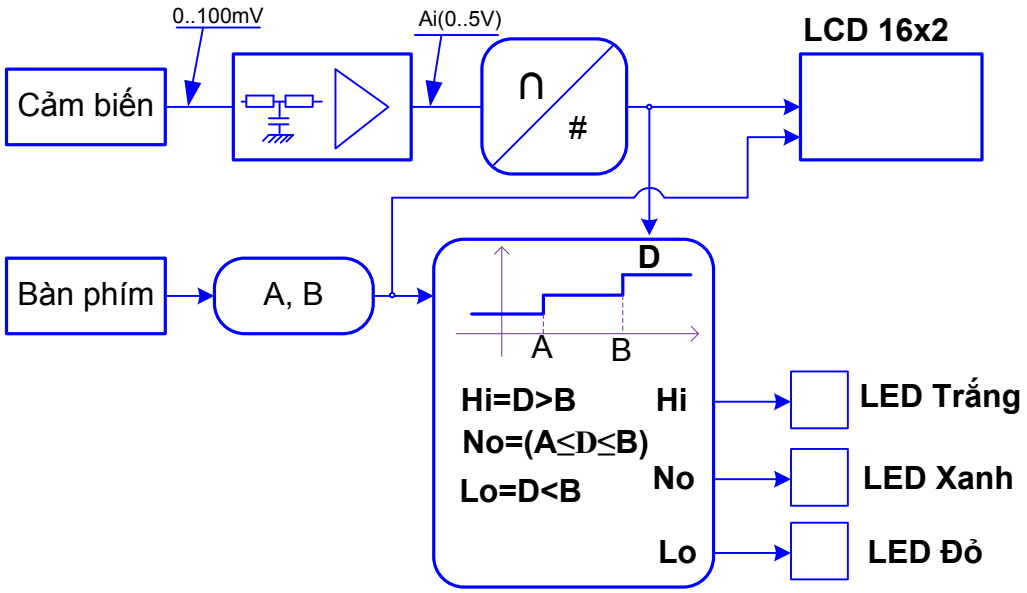
c. **Khối Điều khiển trung tâm (DKTT):** Có chức năng xử lý tín hiệu từ khối vào để đưa ra hiển thị và cảnh báo ở khối hiển thị.

- 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.

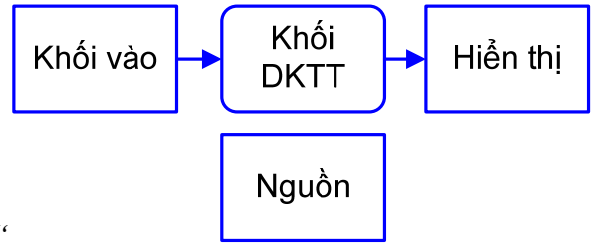
- *Thiết kế sơ đồ tương tác (thuật toán nhúng) toàn hệ thống:*



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/,…)*

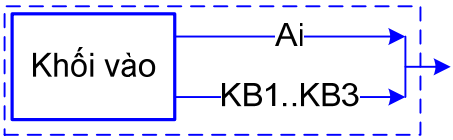
Thiết kế nguyên lý chi tiết cho từng khối theo



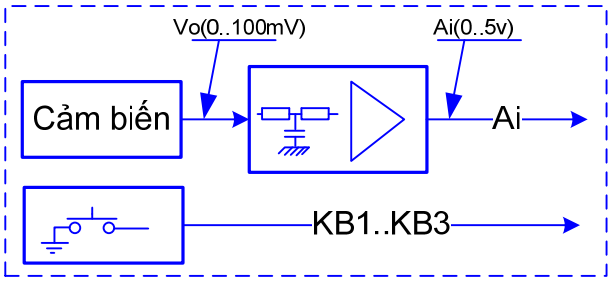
"

❖ Hình 4-24. Sơ đồ khối tổng thể toàn bộ hệ thống":

❖ Khối Vào: được yêu cầu như sau:

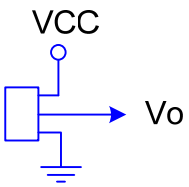


Hình 4-26.

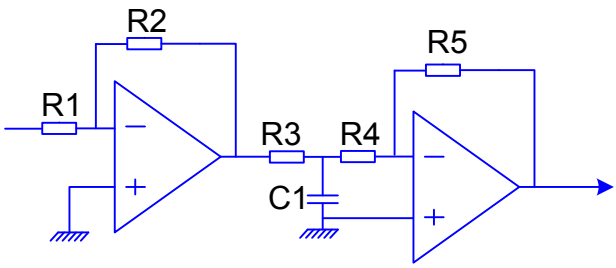


Hình 4-27.

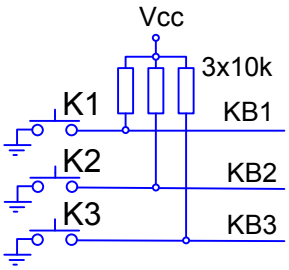
Khối vào sẽ bao gồm: Khối cảm biến, khối lọc và khuếch đại, khối bàn phím. Chi tiết được thiết kế như sau:



Hình 4-28. Mạch cảm biến



Hình 4-29. Khuếch đại và lọc nhiễu



Hình 4-30. Bàn phím

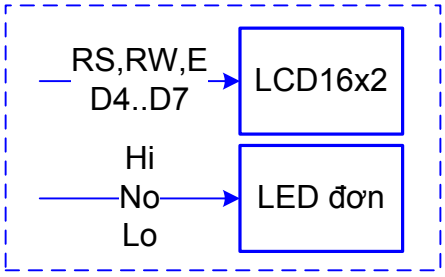
Đặc tả khối:

- 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:

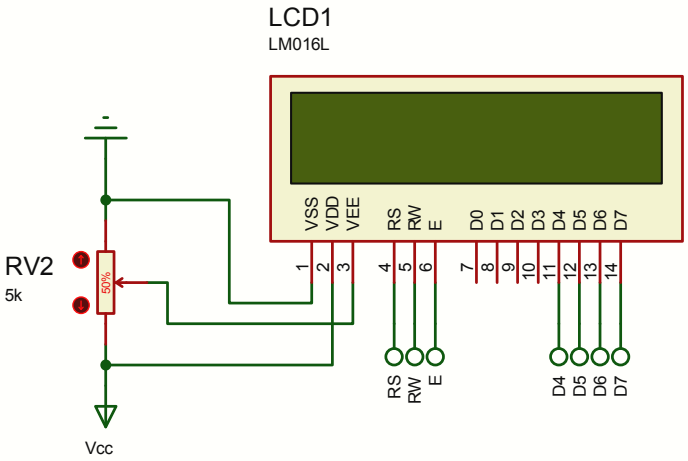
Hình 4-31.



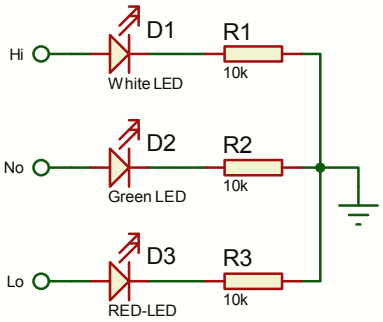
Hình 4-32.

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:

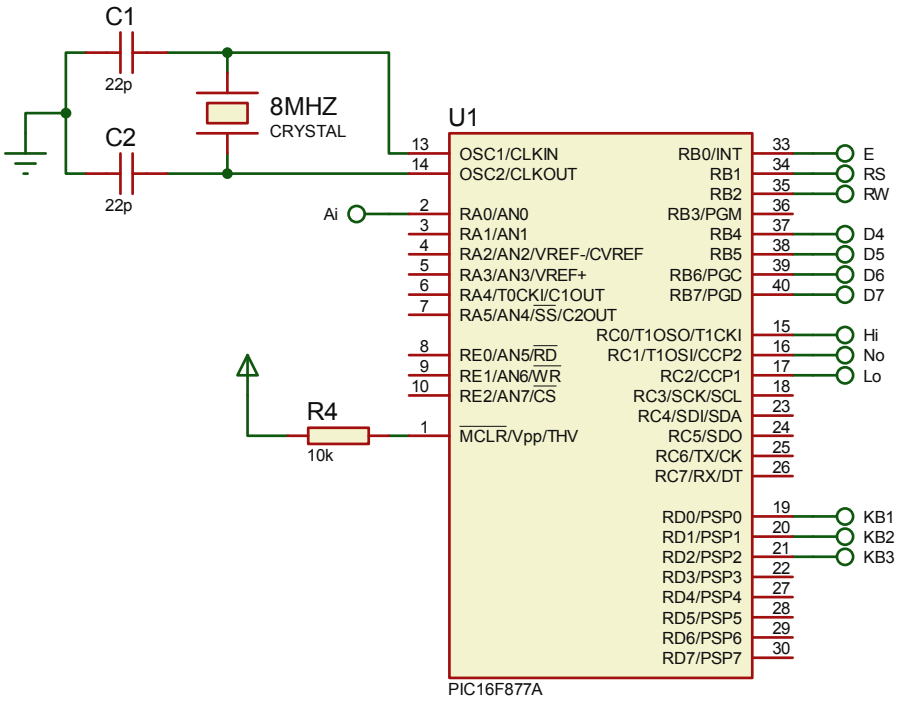


Hình 4-33. Hiển thị LCD



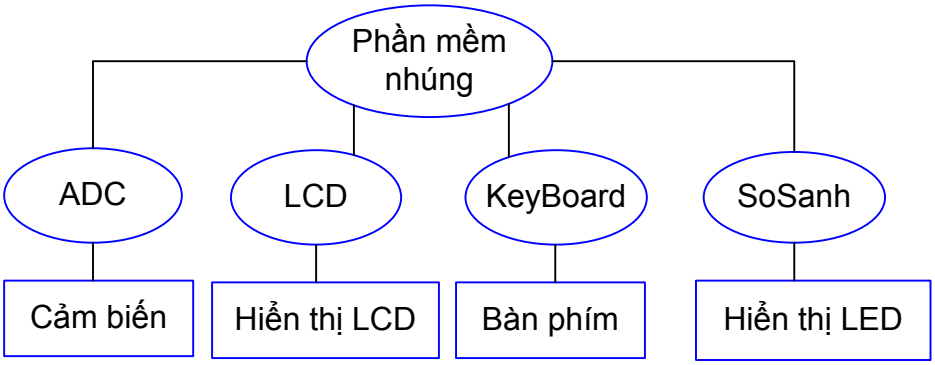
Hình 4-34. Bàn phím

❖ 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

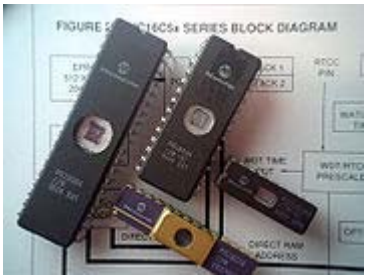
AVRs hiện cung cấp một loạt các tính năng:

- 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

ĐỀ CƯƠNG CHI TIẾT MÔN HỌC

Tên môn học: Kỹ thuật vi xử lý.

ĐỀ CƯƠNG CHI TIẾT MÔN HỌC

1. **Tên môn học:** Kỹ thuật vi xử lý.
2. **Phân bố thời gian:**

   45 tiết (3 trình).
3. **Môn tiên quyết:**

Kỹ thuật vi xử lý là môn học quan trọng trong việc nghiên cứu phần cứng máy tính. Các kiến thức môn học cần có để phục vụ cho môn học này bao gồm:

- Kỹ thuật điện tử số (2 mức: 0, 1, bộ nhớ).
- Kiến trúc máy tính.

4. **Đối tượng học:**

Sinh viên ngành Kỹ thuật máy tính, Điều khiển tự động, Điện tử viễn thông, Kỹ thuật điện tử, Công nghệ thông tin.

5. **Mô tả môn học:**

Trang bị cho sinh viên những kiến thức cơ bản nhất có tính chất hệ thống liên quan đến kỹ thuật VXL. Trang bị cho sinh viên khả năng tư duy trong nghiên cứu, tiếp cận với các hệ VXL tiên tiến, hiện đại hơn. Ngoài ra học phần còn giúp cho sinh viên dễ dàng hơn trong việc xây dựng các chương trình điều khiển thiết bị ghép nối với máy tính.

Nội dụng cụ thể bao gồm các phần cơ bản sau:

- Khái niệm, cấu trúc và nguyên lý hoạt động của một hệ VXL.
- Bộ VXL 8088/8086.
- Các ghép nối cơ bản của 8088/8086 với thiết bị ngoại vi.
- Các phương thức điều khiển vào ra dữ liệu trong kỹ thuật VXL.

6. **Nhiệm vụ của sinh viên:**

Sinh viên phải tham gia đủ trên 80% giờ trên lớp. Phải hoàn thành đầy đủ bài tập và các bài thực hành trong chương trình.

7. **Nội dung giảng dậy**

Chương 1 Hệ vi xử lý

1.1. Vi xử lý là gì?
1.2. Các thế hệ của bộ vi xử lý

4.1.4. Mạch điều khiển bus 8288
4.1.5. Biểu đồ thời gian của các lệnh đọc/ghi
4.2. Phối ghép 8088 với bộ nhớ
4.2.1. Bộ nhớ bán dẫn
4.2.2. Giải mã địa chỉ cho bộ nhớ
4.2.3. Phối ghép 8088 với bộ nhớ
4.3. Phối ghép 8088 với thiết bị ngoại vi
4.3.1. Các kiểu phối ghép vào ra
4.3.2. Giải mã địa chỉ cho thiết vị vào/ra
4.3.3. Các mạch cổng đơn giản
4.3.4. Mạch phối ghép vào/ra song song lập trình được PPI 8255

Chương 5 Vào ra dữ liệu bằng cách thăm dò

5.1. Giới thiệu chung về các phương pháp điều khiển vào/ra dữ liệu
5.2. Vào/ra dữ liệu bằng cách thăm dò trạng thái sẵn sàng của thiết bị ngoại vi

Chương 6 Ngắt và xử lý ngắt trong hệ vi xử lý 8088

6.1. Sự cần thiết phải ngắt CPU
6.2. Ngắt trong vi xử lý 8088
6.2.1. Các loại ngắt trong hệ 8088
6.2.2. Đáp ứng của CPU khi có yêu cầu ngắt
6.2.3. Xử lý ưu tiên ngắt
6.2.4. Mạch điều khiển ngắt ưu tiên PPI 8259A

Chương 7 Vào ra dữ liệu bằng DMA

7.1. Nguyên tắc của việc trao đổi dữ liệu với thiết bị ngoại vi bằng cách thâm nhập trực tiếp vào bộ nhớ (DMA)
7.2. DMAC 8237-5 trong hệ vi xử lý 8088
7.2.1. Tín hiệu HOLD và HLDA trong CPU 8088
7.2.2. Mạch DMAC 8237-5 của Intel

8. Nội dung các bài thực hành (gồm có 6 bài thực hành - 5tiết/bài)

Bài 1: Làm quen với trình dich hợp ngữ ASSEMBLER và cách gọi ngắt trong ASM

Mục đích:

- Biết cách chuyển một chương trình hợp ngữ (Assembler) ra dạng mã máy, qua đó giúp sinh viên hiểu rõ hơn cơ chế hoạt động của các lệnh, cách trao đổi số liệu.
- Biết cách nhập chương trình vào bộ nhớ và chạy chương trình
- Hiểu, biết cách gọi và truyền tham số khi gọi ngắt (Chủ yếu là ngát 21h của DOS, với các hàm hiển thị và nhập ký tự. Nhằm phục vụ cho việc kiểm tra kết qua cho những bài thí nghiệm sau)

Bài 2: Trao đổi dữ liệu và thực hiện các phép tính số học.

Mục đích:

- Giúp sinh viên làm quen với việc nạp và trao đổi dữ liệu giữa các thanh ghi, giữa thanh ghi với bộ nhớ và các phép tính số học cơ bản. Qua đó thấy được ý nghĩa và tác dụng của các thanh ghi và bộ nhớ trong quá trình lưu trữ và xử lý số liệu.

Bài 3: So sánh và kiểm tra dữ liệu.

Mục đích:

- Giúp sinh viên làm quen với các cấu trúc lập trình cơ bản bằng hợp ngữ.
- Biết cách so sánh và kiểm tra dữ liệu thông qua các cờ.

Bài 4: Các thao tác Logic.

Mục đích:

- Giúp sinh viên làm quen với các cấu trúc lập trình cơ bản bằng hợp ngữ.
- Giúp sinh viên khảo sát các thao tác đại số BOOLEAN theo phương thức mà chúng tác động trên các thanh ghi và dữ liệu trong bộ nhớ.

Bài 5: Chương trình con và truyền tham số.

Mục đích:

- Giúp sinh viên làm quen với các cấu trúc lập trình cơ bản bằng hợp ngữ.
- Giúp sinh viên làm quen với cấu trúc lập trình theo chương trình con, truyền tham số cho chương trình con.

Bài 6: Vào ra dữ liệu với các cổng.

Mục đích:

- Giúp sinh viên làm quen với các kiểu giao tiếp giữa vi xử lý với các thiết bị ngoài.
- Biết cách lập trình đặt các chế độ làm việc cho các cổng giao tiếp của bộ điều khiển ghép nối.

9. Tài liệu tham khảo:

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

[2] Đỗ Xuân Thụ & Hồ Khánh Lâm, *Kỹ thuật vi xử lý và máy vi tính, ...*

[3] Đỗ Xuân Tiến, *Kỹ thuật lập trình điều khiển hệ thống, ...*

[4] Lê Văn Doanh & Phạm Khắc Chương, *Kỹ thuật vi điều khiển, ...*

[5] Biên Dịch: Nguyễn Minh San - Hoàng Đức Hải, *Cẩm nang lập trình hệ thống, ...*

[6] Nguyễn Đình Việt, *Kiến trúc máy tính, ...*

[7] Trần Quang Vinh, *Cấu trúc máy vi tính*, NXB Giáo Dục, 1998.

[8] Ytha Yu & Charles Marut, *Lập trình hợp ngữ (Assembly) và máy vi tính IBM-PC*, NXB Giáo Dục, 1996.

[9] PTS. Nguyễn Quang Tấn, Vũ Thanh Hiền, *Lập trình với Hợp Ngữ*, NXB Thống Kê, 1997.

[10] Trần Bá Thái, *Điều khiển và ghép nối các thiết bị ngoại vi*, NXB thống kê, 1987.

[11] Computer Organization and Assembly Language Programming For IBM PC and Compatibles Michael Thorne - The Benjamin-Cummings Publishing Company, Inc. 1991.

[12] Microprocessors and microcomputer-based system design Mohamed Rafiquzzaman - CRC Press, 1995.

NỘI DUNG CHI TIẾT

Chương 1
HỆ VI XỬ LÝ

1.1. Vi xử lý là gì?

Ngày nay xu hướng số hoá mọi dạng tín hiệu càng được khẳng định rõ nét trong nhiều lĩnh vực: Điện tử, tin học, viễn thông, công nghệ thông tin, kỹ thuật điều khiển tự động ... vì tín hiệu số có cấu trúc đơn giản, dễ tính toán, xử lý và gia công ...

Việc xử lý, tính toán, điều khiển được thực hiện chủ yếu trên các máy tính PC (Hay hệ vi xử lý nói chung). Các hệ vi xử lý này thường được ghép nối và giao tiếp với nhiều thiết bị ngoại vi khác nhau. Mỗi thiết bị làm việc ở môi trường khác nhau cũng như chức năng, nhiệm vụ khác nhau.

Môi trường của thiết bị có thể là:

- Điện, điện tử.
- Cơ, cơ điện.
- Quang điện tử, ...

Chức năng, nhiệm vụ của thiết bị như:

- Thông tin vô tuyến, hữu tuyến.
- Kỹ thuật viễn thông.
- Robốt, máy công cụ, dây truyền sản xuất tự động.

Các hệ thống làm nhiệm vụ xử lý và điều khiển nói chung luôn có một thành phần làm nhiệm vụ xử lý được chế tạo bằng công nghệ vi điện tử với độ tích hợp cao và rất cao, chúng thường được gọi là các bộ vi xử lý (MicroProcessor). Các bộ vi xử lý hoạt động (làm việc) theo chương trình, dùng để tính toán và điều khiển mọi hoạt động của hệ thống.

Việc xây dựng các chương trình điều khiển các thiết bị cho chúng làm việc chính xác, đồng bộ là rất phức tạp. Các hệ thống càng thông minh thì vai trò của bộ vi xử lý càng quan trọng.

1.2. Các thế hệ của bộ vi xử lý

1.2.1. Thế hệ 1 (1971 đến 1973)

Năm 1971 Intel cho ra đời bộ vi xử lý (VXL) 4004 (dùng cho các máy tính cầm tay) được chế tạo bằng công nghệ PMOS. Đây là bộ VXL 4 bit dữ liệu, 12 bit địa chỉ và có 2250 Transitor.

Tiếp theo, Intel cho ra đời bộ VXL 4040 là bộ VXL được cải tiến từ VXL 4004. Trong thời gian này, Intel tiếp tục cho ra đời bộ VXL 8008 là bộ VXL 8 bit dữ liệu.

- **Đặc điểm của các bộ vi xử lý trong khoảng thời gian này là:**
  - Tốc độ thực hiện: 10 ÷ 60 (μs/lệnh).
  - Tần số đồng hồ: fCLK = 0,1 ÷ 0,8 MHz.
  - Cần nhiều mạch phụ trợ để tạo nên một hệ vi xử lý hoàn chỉnh.

1.2.2. Thế hệ 2 (1974 đến 1977)

Trong thời gian này Intel lần lượt cho ra đời bộ VXL 8080, 8085. Motorola có các bộ VXL 6800, 6809. Zilog có bộ VXL Z80 và Signetics có bộ VXL 6520.

- **Đặc điểm của các bộ vi xử lý trong khoảng thời gian này là:**
  - Tập lệnh phong phú hơn.
  - Là các bộ vi xử lý 8 bit dữ liệu.
  - Khả năng phân biệt địa chỉ bộ nhớ lên tới 64 KB (16 bit địa chỉ).
  - Khả năng phân biệt địa cổng là 256 cổng cho thiết bị ngoại vi (sử dụng 8 bit để đánh địa chỉ cho các cổng).
  - Tốc độ 1 ÷ 8 (μs/lệnh).
  - Tần số đồng hồ: fCLK = 1 ÷ 5 MHz.
- **Ứng dụng:**
  - Điều khiển các hệ thống trong công nghiệp.
  - Chế tạo các máy tính 8 bit như Apple II.

1.2.3. Thế hệ 3 (1978 đến 1982)

Trong khoảng thời gian này Intel lần lượt cho ra đời các bộ VXL 8086, 8088, 80186, 80286. Motorola có các bộ VXL 68000, 68010

- **Đặc điểm của các bộ vi xử lý trong khoảng thời gian này là:**
  - Là các bộ vi xử lý 16 bit dữ liệu.
  - Tập lệnh đầy đủ hơn.
  - Khả năng phân biệt địa chỉ bộ nhớ từ 1 MB đến 16 MB.
  - Khả năng phân biệt địa chỉ cổng là 64 K cổng cho thiết bị ngoại vi (đối với các bộ VXL của Intel).
  - Tốc độ 0,1 ÷ 1 (μs/lệnh).
  - Tần số đồng hồ: fCLK = 5 ÷ 10 MHz.
- **Ứng dụng:**

Chế tạo các máy tính IBM PC, PC/XT, PC/AT và máy tính Macintosh của Apple.

1.2.4. Thế hệ 4 (1983 đến nay)

Trong thời gian này Intel thể hiện sức mạnh vượt trội các hãng khác trong việc chế tạo bộ vi xử lý. Intel liện tục cho ra đời các bộ VXL 80386, 80486 là các bộ VXL 32 bit dữ liệu, có bên trong đơn vị quản lý bộ nhớ (MMU) cho phép chạy trong chế độ bộ nhớ ảo và đa nhiệm. Tiếp theo là các bộ VXL Pentium, Pentium II, Pentium III, Pentium IV là các bộ VXL 64 bit dữ liệu.

Motorola có các bộ VXL 68020, 68030, 68040, 68060.

- **Đặc điểm của các bộ vi xử lý trong khoảng thời gian này là:**
  - 32 bit địa chỉ, nên trong chế độ thực thì khả năng phân biệt địa chỉ bộ nhớ là 4 GB. Trong chế độ bộ nhớ ảo thì chúng có khả năng quản lý không gian nhớ lên tới 64 TB (Teta Byte).
  - Cơ chế xử lý xen kẽ dòng mã lệnh (Pipline).
  - Bộ nhớ ẩn (Cache).
  - Có bộ quản lý bộ nhớ (MMU), bộ đồng xử lý toán học được tích hợp bên trong.
  - Tốc độ 6 ÷ 112 (triệu lệnh/μs).

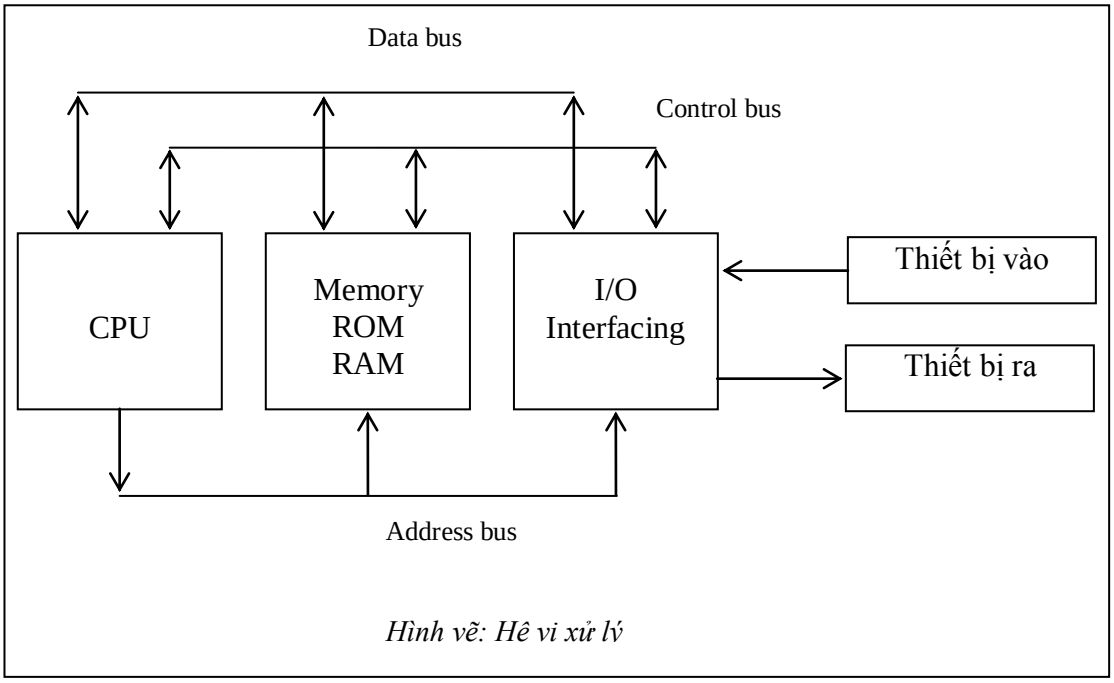
- Tần số đồng hồ: fCLK = 10 ÷ 100 MHz và cao hơn nữa.

Ứng dụng:

- Chế tạo các máy tính có tốc độ cao, các máy chủ đáp ứng cho các xử lý lớn như thống kê hàng ngày tại các ngân hàng, khí tượng thuỷ văn, mô phỏng các quá trình, lĩnh vực quân sự ...
- Các máy tính hiện nay.

1.3. Giới thiệu cấu trúc của hệ vi xử lý

Chúng ta đã được tìm hiểu qua về sự ra đời và vị trí của các bộ vi xử lý. Bộ VXL là thành phần cơ bản (trái tim) của máy tính, nó được kết hợp với các bộ phận mạch điện tử khác như bộ nhớ (bộ nhớ bán dẫn), bộ phối ghép vào ra để tạo nên hệ vi xử lý nói chung mà máy tính là một trường hợp ứng dụng của thể của hệ vi xử lý.



Hình vẽ: Hệ vi xử lý

- CPU (Central Processing Unit) - Bộ xử lý trung tâm.
- Memory: Bộ nhớ bán dẫn.
- I/O (Input/Output): Khối phối ghép với thiết bị ngoại vi.
- Bus là tập các đường dây truyền thông tin, tín hiệu gồm:
  - Data bus.
  - Control bus.
  - Address bus.

1.3.1. CPU - Bộ xử lý trung tâm

Là mạch điện tử có độ tích hợp cao (là trái tim của hệ vi xử lý). Ngày nay thường là các vi mạch có độ tích hợp VLSI.

Chức năng:

- Điều khiển mọi hoạt động của hệ vi xử lý (hoạt động tính toán).
- Thực hiện lệnh, xử lý dữ liệu.

Nguyên tắc hoạt động

Hoạt động theo chương trình nằm trong bộ nhớ. Nó nhận lần lượt nhận từng lệnh từ bộ nhớ, các lệnh được ghi dưới dạng các bit 0, 1 sau đó giải mã lệnh thành các xung điều khiển tương ứng các thao tác của lệnh để điều khiển các khối chức năng thực hiện các thao tác đó. Quá trình thực hiện trên bao gồm cả trao đổi dữ liệu với bộ nhớ. Để thực hiện được như trên, bên trong CPU có thanh ghi lưu địa chỉ của lệnh chuẩn bị được thực hiện, gọi là thanh ghi con trỏ lệnh (Instruction Pointer - PC), hay còn được gọi là bộ đếm chương trình (Program Counter - PC).

❖ **Các thành phần cơ bản của bộ vi xử lý:**

- *Đơn vị điều khiển (Control Unit - CU)*: điều khiển hoạt động chính của CPU và các thành phần khác của hệ theo chương trình đã định (dãy các lệnh) bằng các xung điều khiển.
- *Đơn vị số học và logic (Arithmetic and Logic Unit - ALU)*: thực hiện chức năng xử lý dữ liệu (tính toán) như cộng, trừ, nhân, chia, NOT, AND, OR ...
- *Tập thanh ghi (Registers Set)*: là các ngăn nhớ đặc biệt nằm ngay trong CPU để tăng tốc độ trao đổi dữ liệu. Một số thanh ghi lưu trữ thông tin tạm thời phục vụ cho việc thực hiện chương trình.
- *Bus bên trong (Internal Bus)*: Hệ thống bus trong CPU là tập các đường dây làm nhiệm vụ kết nối, vận chuyển thông tin (tín hiệu) giữa các thành phần với nhau.

1.3.2. Bộ nhớ bán dẫn (ROM, RAM)

Là bộ phận quan trọng trong hệ vi xử lý. Nó có nhiệm vụ lưu trữ chương trình và dữ liệu. Bộ nhớ trong bao gồm bộ nhớ chính và bộ nhớ ẩn (cache L1, L2). Khi khởi động máy, chương trình điều khiển được chứa trong ROM sẽ điều khiển hoạt động toàn hệ. Các chương trình ứng dụng, một phần chương trình điều khiển, kết quả chạy chương trình được để ở RAM. Ngoài ra còn có bộ nhớ ngoài (ổ đĩa từ, quang ...) lưu trữ lâu dài chương trình và dữ liệu, là các thiết bị ngoại vi.

1.3.3. Hệ thống vào ra (I/O)

❖ **Chức năng**

Giao tiếp, trao đổi thông tin giữa hệ vi xử lý với thế giới bên ngoài.

❖ **Các thành phần cơ bản**

- Thiết bị ngoại vi: Bàn phím, màn hình, chuột, máy in, ổ đĩa từ, ổ đĩa quang ... các bộ chuyển đổi ADC, DAC ..., chuyển đổi thông tin dưới dạng nào đó thành dạng phù hợp với máy tính và ngược lại rồi liên hệ với máy tính thông qua khối phối ghép vào/ra.
- Mạch phối ghép vào/ra dùng ghép nối thiết bị ngoại vi với hệ vi xử lý (máy tính). Trong mạch phối ghép vào/ra có bộ phận phối ghép cụ thể giữa hệ thống bus với thế giới bên ngoài gọi là các cổng vào ra (I/O port). Mỗi cổng có địa chỉ xác định. I/O port vào: nhận thông tin từ bên ngoài vào hệ thống, I/O port ra: đưa thông tin từ hệ ra thế giới bên ngoài.

1.3.4. Liên hệ giữa các khối

Hệ thống bus là tập các đường dây dùng để kết nối, trao đổi thông tin từ các phần mạch này tới các thành phần khác (các khối) trong phạm vi một máy tính (1 hệ vi xử lý).

Độ rộng bus là số bit thông tin được vận chuyển đồng thời trong một chu kỳ bus.

- **Bus địa chỉ**

Theo sự phát triển các bộ vi xử lý, độ rộng bus địa chỉ tăng từ 16, 20, 24 và 32 bit. Bus địa chỉ dùng để vận chuyển địa chỉ từ CPU đến bộ nhớ hay mạch phối ghép vào/ra để tìm ra ngăn hay nhớ cổng vào/ra cần trao đổi dữ liệu.

- Khả năng phân biệt địa chỉ của CPU phụ thuộc độ rộng bus địa chỉ.
- Bus địa chỉ gồm $A_{n-1} \div A_0$ (n bit) ➡ có thể quản lý được 2n địa chỉ.
- Độ rộng bus địa chỉ cho biết khả năng phân biệt và quản lý không gian nhớ.

- **Bus dữ liệu**

Độ rộng bus dữ liệu thường là 8, 16, 32 và 64 tuỳ theo các bộ vi xử lý. Ngày nay các bộ vi xử lý thường làm việc với bus dữ liệu có độ rộng 64 bit, thậm chí là 128 bit. Độ rộng bus dữ liệu quyết định số bit dữ liệu mà CPU có khả năng nhận hay gửi (đọc/ghi) hay xử lý cùng lúc.

Bus dữ liệu là bus 2 chiều, dữ liệu có thể được truyền từ CPU đến bộ nhớ hay cổng vào/ra hoặc ngược lại.

- **Bus điều khiển**

Độ rộng bus điều khiển thường nhỏ hơn độ rộng bus địa chỉ và bus dữ liệu. Mỗi tín hiệu điều khiển có một chiều nhất định. CPU có thể gửi các tín hiệu điều khiển tới các khối đồng thời nó cũng nhận tín hiệu điều khiển từ các khối gửi đến. Trong chừng mực nào đó có thể coi bus điều khiển là 2 chiều. Tính 2 chiều không phải của một tín hiệu điều khiển cụ thể nào mà là của một nhóm tín hiệu.

- Các tín hiệu phát ra từ CPU: MEMR (tín hiệu điều khiển đọc bộ nhớ), MEMW (tín hiệu điều khiển ghi bộ nhớ), IOR (tín hiệu điều khiển đọc cổng vào ra), IOW (tín hiệu điều khiển ghi cổng vào ra).
- Tín hiệu điều khiển ngắt: INTR.
- Tín hiệu điều khiển chuyển nhượng bus (HOLD, HLDA).
- Clock (CLK): Xung nhịp phát ra từ bộ dao động cấp cho CPU và các thành phần khác để hệ thống hoạt động đồng bộ.

Chương 2
BỘ VI XỬ LÝ 8088 CỦA INTEL

2.1. Giới thiệu hoạt động của bộ vi xử lý 8088

2.1.1. Giới thiệu chung

Bộ vi xử lý 8088 thuộc họ vi xử lý của Intel.

Điển hình: 8085 là bộ vi xử lý 8 bit.
8086 là bộ vi xử lý 16 bit hoàn chỉnh.
8088 là bộ vi xử lý 16 bit trong/ 8 bit ngoài.

- **Các đặc tính kỹ thuật chủ yếu:**
  - Số thanh ghi: 14 thanhg ghi 16 bit.
  - Bus địa chỉ: 20 bit.
  - Bus dữ liệu: 16 (8086) và 8 (8088).
  - Tập lệnh: 115 (là số lệnh được công bố trong nhiều tài liệu).
  - Tốc độ chuẩn: 4.77 MHz.
  - Số chân của bộ vi xử lý: 40.

Bộ vi xử lý 8086 của Intel được phát triển từ năm 1978 vào đưa vào thị trường từ năm 1980. Đây là bộ vi xử lý 16 bit bán hoàn chỉnh, các thanh ghi bên trong là 16 bit và nó xử lý 16 bit dữ liệu cùng một lúc, 8086 liên hệ với kênh số liệu bên ngoài bằng bus dữ liệu 16 bit và bus địa chỉ là 20 bit.

Bộ vi xử lý 8088 ra đời sau 8086, nó có cấu trúc bên trong và tập lệnh hoàn toàn giống của 8086, chỉ khác 8086 ở kênh truyền dữ liệu với bên ngoài. 8088 sử dụng kênh dữ liệu 8 bit nên việc truyền dữ liệu giữa các thanh ghi trong nó với bộ nhớ chậm hơn so với 8086. Đây là một bước lùi về kỹ thuật nhưng đem lại nhiều lợi ích về kinh tế. Tại thời điểm lịch sử đó, bus DataMaster 8 bit đang được sử dụng rộng rãi trên thị trường, nhiều loại card mở rộng và các chip hỗ trợ có sẵn trên thị trường là loại 8 bit nên giá thấp hơn loại 16 bit tương ứng. Việc sử dụng bus dữ liệu 8 bit giúp cho người sử dụng khi nâng cấp máy có thể tận dụng được các card 8 bit trong các máy cũ cũng như chỉ phải mua mới với giá thấp. Đồng thời cũng giúp cho các nhà sản xuất các máy tương thích với IBM PC có thêm nhiều cơ hội lựa chọn sản phẩm của các hãng khác nhau. Chiến lược phát triển có tính kế thừa như vậy đã góp phần làm cho khách hàng của IBM ngày một gia tăng.

- **Chọn bộ vi xử lý 8088 để nghiên cứu vì:**
  - Tập lệnh chung cho các bộ vi xử lý nói trên.
  - Tính phức tạp vừa phải, phù hợp với những người mới tìm hiểu.

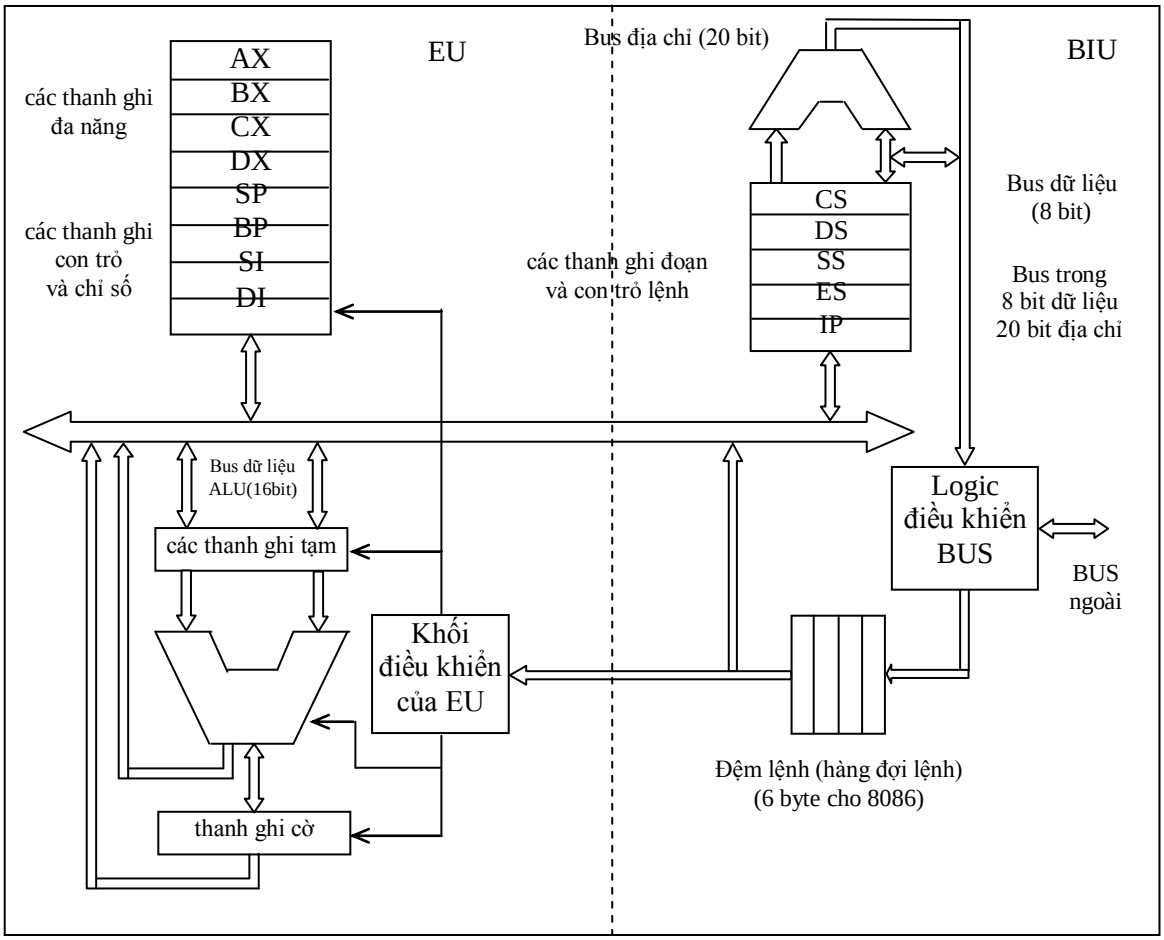
2.1.2. Cấu trúc và hoạt động của bộ VXL 8088

Sự hoạt động của bộ vi xử lý 8088/8086 thực sự là việc thực hiện lặp đi lặp lại 3 thao tác chính là lấy lệnh (fetch), giải mã lệnh (decode) và thực hiện (execute). Sơ đồ khối của bộ vi xử lý 8088/8086 như hình vẽ với 2 đơn vị chính:

- EU (Execution Unit): Đơn vị (khối) thực hiện lệnh.
- BIU (Bus Interface Unit): Đơn vị giao tiếp bus (khối tương thích bus) để điều khiển bus hiệu quả hơn.

- **EU**

Đơn vị EU của 8088 và 8086 giống nhau, bao gồm ALU, thanh ghi cờ, các thanh ghi đệm và các thanh ghi đa năng. Các bus dữ liệu bên trong của EU đều là 16 bit. EU không nối trực tiếp với bên ngoài, nó nhận lệnh từ hàng đợi lệnh bên trong BUI. Nếu là lệnh cần truy nhập bộ nhớ hoặc cổng vào/ra (I/O port - thiết bị ngoại vi) thì EU yêu cầu BIU lấy hoặc gửi dữ liệu. Tất cả các địa chỉ mà EU thao tác đều là 16 bit, khi gửi sang BIU thì BIU sẽ thực hiện tính toán để tạo địa chỉ vật lý 20 bit và phát ra các chân địa chỉ của chip.

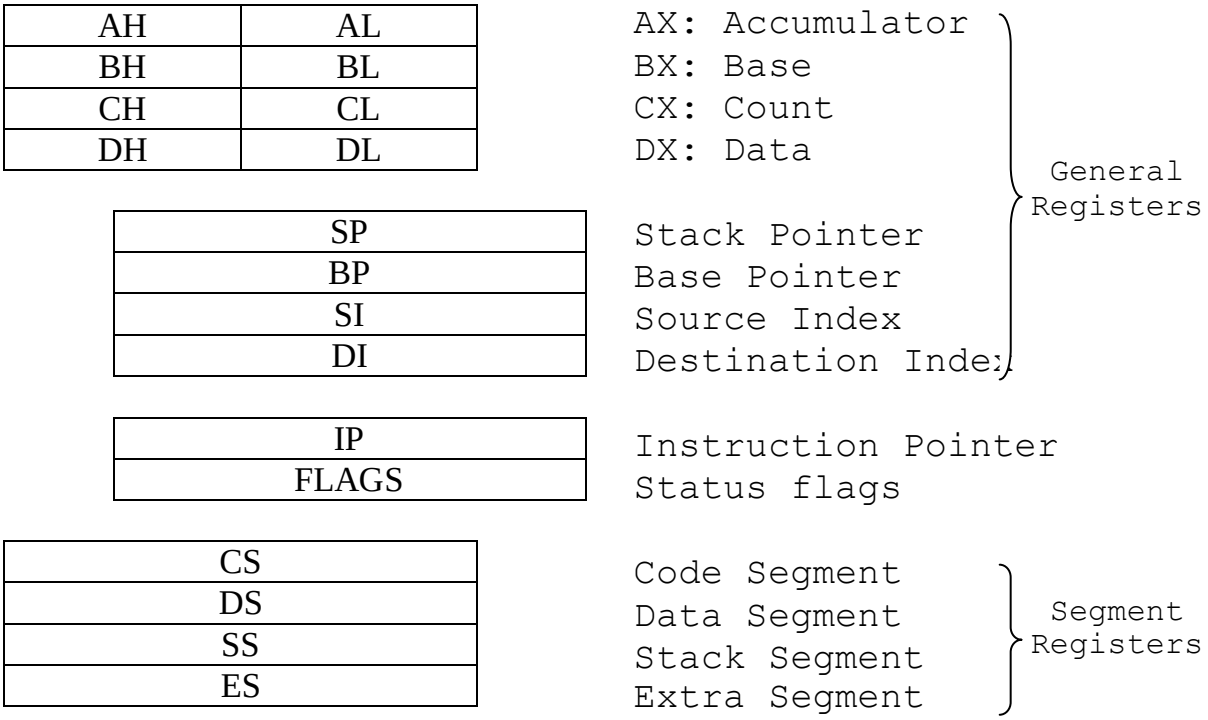


Các thanh ghi trong

Có 8 thanh ghi, là thành phần nhớ có tốc độ truy nhập rất cao. Bao gồm các thanh ghi đa năng Ax, Bx, Cx, Dx. Mỗi thanh ghi 16 bit có thể được phân chia thành 2 thanh ghi 8 bit làm việc độc lập nhau.

- AX (Accumulator, Acc): Thanh chứa, chứa tạm thời dữ liệu (toán hạng, kết quả phép toán như nhân, chia và được coi là Acc). AX có thể được phân chia thành AH (Phần cao) và AL (Phần thấp).
- BX (Base): Thanh ghi cơ sở, thường được dùng để chứa địa chỉ cơ sở cần truy nhập trong lệnh XLAT. BX có thể được phân chia thành BH (Phần cao) và BL (Phần thấp).
- CX (Count): Bộ đếm, thường được dùng để đếm số lần lặp của công việc (số lần lặp trong các vòng lặp). CX có thể được phân chia thành CH (Phần cao) và CL (Phần thấp). CX thường được dùng để chứa số lần lặp trong các lệnh LOOP, còn CL thường chứa số lần dịch hoặc quay trong các lệnh dịch quay thanh ghi.

- DX (Data): Thanh ghi dữ liệu, sử dụng để chứa toán hạng, kết quả. DX cùng AX dùng để chứa toán hạng hoặc kết quả của các phép nhân, chia số 16 bit. DX còn chứa địa chỉ cổng trong các lệnh vào ra trực tiếp (IN, OUT). DX có thể được phân chia thành DH (Phần cao) và DL (Phần thấp).



Hình vẽ: Bộ vi xử lý 8088 và tập thanh ghi

Các thanh ghi con trỏ, chỉ số

Các thanh ghi SP, BP là các thanh ghi con trỏ không tách rời.

- SP (Stack Pointer): Thanh ghi con trỏ ngăn xếp, trỏ vào đỉnh hiện thời của ngăn xếp nằm trong đoạn ngăn xếp SS (Nó luôn kết hợp với thanh ghi SS). Ta có địa chỉ logic SS: SP. Sau mỗi thao tác cất một word vào stack (thao tác Push) thì SP tự động giảm 2 đơn vị, còn sau thao tác lấy một word ra khỏi stack (thao tác Pop), SP được tự động tăng 2 đơn vị.
- BP (Base stack Pointer): Con trỏ cơ sở, luôn trỏ vào một dữ liệu cụ thể nằm trong đoạn ngăn xếp SS. Ta có địa chỉ logic SS: BP.
- SI (Source Index): Thanh ghi chỉ số nguồn (hay nguồn), chỉ vào dữ liệu nằm trong đoạn DS. Ta có địa chỉ logic DS: SI.
- DI (Destination Index): Thanh ghi chỉ số đích, chỉ dữ liệu trong đoạn DS. Ta có địa chỉ logic DS: DI.

Ta có các cặp SP, BP đi với SS và SI, DI đi với DS.

Trong các lệnh thao tác với dữ liệu kiểu chuỗi thì cặp ES:DI luôn ứng với địa chỉ của phần tử thuộc chuỗi đích còn cặp DS:SI ứng với địa chỉ của phần tử thuộc chuỗi nguồn.

Khối ALU

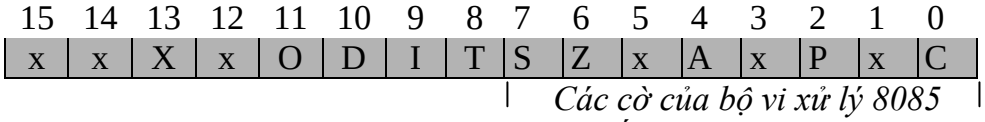
Làm nhiệm vụ thực hiện các lệnh số học và logic.

- Số học: +, -, *, /, so sánh, đảo dấu.

- Logic: NOT, AND, OR, XOR.

Thanh ghi cờ:

Đây là thanh ghi 16 bit, mỗi bit được sử dụng để thể hiện một trạng thái của bộ vi xử lý tại một thời điểm nhất định trong quá trình thực hiện chương trình (dãy các câu lệnh), nhưng chỉ dùng 9 bit đối với bộ vi xử lý 8088/8086. Mỗi bit đó được gọi là một cờ (flag). Giá trị của mỗi cờ được biểu diễn bằng các ký hiệu gợi nhớ như cách biểu diễn của chương trình Debug của DOS.



(x: Không được định nghĩa (don't care), với 8 bit thấp là các cờ của bộ vi xử lý 8085)

Hình vẽ: Sơ đồ thanh ghi cờ của 8088/86

- Cờ trạng thái: Biểu diễn trạng thái phép toán vừa thực hiện.
  - CF (Carry Flag): Cờ nhớ (cờ tràn).

CF = 1 (CY - CarrY) khi kết quả phép toán vượt (tràn) khuôn khổ biểu diễn (có nhớ (mượn) lên trên (từ) MSB), CF = 0 (NC - No Carry) trong trường hợp ngược lại. Cờ này thường được sử dụng khi thực hiện các phép cộng, trừ các số nhiều byte.

Ví dụ:

```
    1011 0001
+   0110 1001
  10001 1010
```

  - PF (Parity Flag): Cờ chẵn lẻ.

PF = 1 (PE - Parity Even) khi số bit 1 trong kết quả phép toán (hay các phép vận chuyển dữ liệu) là chẵn, PF = 0 (PO - Parity Old) trong trường hợp ngược lại.

  - AF (Carry Flag): Cờ nhớ phụ (tràn phụ - cờ bổ trợ).

Cờ này có ý nghĩa khi ta làm việc với số BCD. AF = 1 (AC - Auxiliary Carry) khi có nhớ hoặc mượn từ một số BCD thấp (4 bit thấp) sang một số BCD cao (4 bit cao) và AF = 0 (NA - No Auxiliary carry) trong trường hợp ngược lại.

  - SF (Sign Flag): Cờ dấu. Trong bộ vi xử lý 8088/8086 các số âm được biểu diễn dưới dạng số bù 2, nên phải dùng cờ SF để chỉ thị dấu của kết quả.

SF = 1 (NG - NeGative), khi kết quả phép toán là một số âm, SF = 0 (PL- PLus) trong trường hợp ngược lại.

  - ZF (Zero Flag): Cờ rỗng.

ZF = 1 (ZR - ZeRo) khi kết quả phép toán = 0, ZF = 0 (NZ-Non Zero) trong trường hợp ngược lại.

  - OF (Overflow Flag): Cờ tràn.

OF = 1 (OV-OVerflow) khi kết quả là số bù 2 vượt khuôn khổ biểu diễn (tràn số học, hay nói cách khác: khi cộng hai số cùng dấu mà kết quả là một số trái dấu thì OF = 1), OF = 0 (NV-Non oVerflow) trong trường hợp ngược lại (cờ này làm việc với số có dấu).

- Cờ điều khiển

Cờ trạng thái phụ thuộc kết quả phép toán, còn với cờ điều khiển ta có thể thiết lập nhờ lệnh.

- IF (Interrupt Flag): Cờ ngắt.

IF = 1 (EI-Enable Interrup), CPU cho phép ngắt, IF = 0 (DI-Disable Interrup) CPU không cho phép ngắt (cấm) các loại ngắt che được (Maskable)..

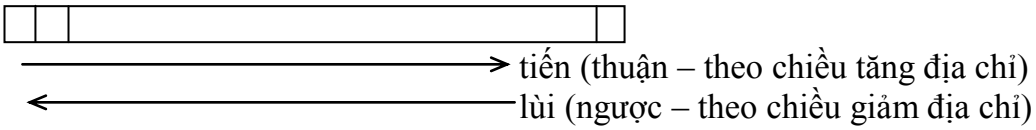
- TF (Trap Flag): Cờ bẫy.

TF = 1 CPU làm việc trong chế độ chạy từng lệnh, thường dùng để gỡ rối chương trình (debug). Sau khi thực hiện xong mỗi lệnh, bộ vi xử lý sẽ phát ra một lệnh ngắt (INT) để có kiểm tra chương trình.

- DF (Direction Flag): Cờ hướng.

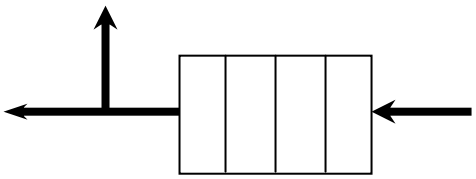
Điều khiển hướng xử lý đối với thao tác chuỗi. DF = 1 (DN-DowN) thì các lệnh vận chuyển dữ liệu hay xử lý chuỗi sẽ thao tác lùi từ phải đến trái (địa chỉ cao đến địa chỉ thấp). DF=0 (UP) trong trường hợp ngược lại (thao tác các phần tử từ địa chỉ thấp đến địa chỉ cao).

Cờ này thường được lập bởi chương trình của người sử dụng khi có các lệnh thao tác chuỗi.



BIU

- Hàng đợi: Là tập thanh ghi



Với 8086 hàng đợi lệnh là 6 byte, 8088 hàng đợi lệnh là 4 byte nên chứa được tối đa là 4 lệnh. CPU chứa tập thanh ghi theo kiểu LIFO.

Cơ chế:

- Đọc lệnh (lấy lệnh)
- Giải mã lệnh
- Thực hiện

- Thanh ghi IP: Thanh ghi con trỏ lệnh, trỏ vào lệnh tiếp theo chuẩn bị được thực hiện nằm trong đoạn CS (CS: IP). Sau khi đọc 1 byte, IP tự động tăng thêm 1. Như vậy thực tế thì cặp CS:IP mới là con trỏ lệnh vì nó chứa địa chỉ đầy đủ của một lệnh trong bộ nhớ.
- Thanh ghi đoạn:
  - CS (Code Segment): Thanh ghi đoạn mã, là thanh ghi địa chỉ đoạn mã lệnh, chứa địa chỉ cơ sở (địa chỉ đoạn) của chương trình đang thực hiện.
  - DS (Data Segment): Thanh ghi đoạn dữ liệu, là thanh ghi địa chỉ đoạn dữ liệu, chứa địa chỉ đoạn của vùng dữ liệu mà chương trình đang thực hiện sử dụng. Vùng này thường chứa các biến của chương trình.
  - SS (Stack Segment): Thanh ghi đoạn ngăn xếp, là thanh ghi địa chỉ đoạn bộ nhớ ngăn xếp (stack) của chương trình đang chạy.

- ES (Extra Segment): Thanh ghi đoạn dữ liệu phụ, là thanh ghi địa chỉ đoạn dữ liệu bổ sung mà chương trình đang thực hiện sử dụng. Vùng này cũng thường chứa các biến của chương trình.

8088 có 20 bit địa chỉ, trong khi đó các thanh ghi quản lý bộ nhớ là 16 bit. Để xác định một ngăn nhớ có địa chỉ 20 bit, dùng 2 thanh ghi 16 bit để xác định địa chỉ.

Địa chỉ luôn nằm trong một thanh ghi gọi là địa chỉ đoạn.

Địa chỉ lệch (offset) trong một thanh ghi khác (Ax, Bx, Cx, ...).

Địa chỉ logic: DS:SI
DS:DI
DS:XX

Địa vật lý (phải là số 20 bit) được xác định như sau:

Địa vật lý = địa chỉ đoạn*16 + địa chỉ lệch

Ví dụ: DS = 4000h, SI = 3F4Dh

Địa chỉ logic của ngăn nhớ: 4000:3F4D

Địa chỉ vật lí của ngăn nhớ: DS * 16 + SI = 40000 + 3F4D = 43F4Dh

Tại một thời điểm CPU quản lý được 4 đoạn nhớ (DS, SS, CS, ES) , nó có thể đồng thời truy nhập 4 đoạn đó.

Bộ nhớ được chia thành nhiều đoạn. Mỗi đoạn có kích thước tối đa là 64 KB. Vậy ta có 16 đoạn tách rời và địa chỉ lệch thay đổi từ 0000 H đến FFFFh.

Nếu địa chỉ đoạn thay đổi từ 0000h đến FFFFh thì có 64 Kđoạn. Khi này các đoạn bao trùm lên nhau. 2 đoạn kề nhau sẽ các nhau 16 Byte (Paragraph).

Khi khởi động máy (hoặc Reset) CS được nạp giá trị F000h, IP được nạp giá trị FFF0h. Địa chỉ này thuộc đoạn cuối, nơi đặt ROM khởi động.

Địa chỉ vật lý = F000h*16 + FFF0h = F0000h + FFF0h = FFFF0h

2.2. Chế độ địa chỉ của 8088

2.2.1. Chế độ địa chỉ thanh ghi

Dùng thanh ghi như là các toán hạng chứa dữ liệu cần thao tác, tốc độ thực hiện lệnh rất nhanh vì CPU không mất thời gian tìm kiếm dữ liệu.

Ví dụ:

```
MOV  AL, BL      ; AL:= BL
MOV  DS, BX      ; DS:= BX
ADD  AL, DL      ; AL:= AL + DL
```

2.2.2. Chế độ địa chỉ tức thì

Toán hạng đích là thanh ghi, hoặc ô nhớ. Toán hạng nguồn là 1 giá trị cụ thể (hằng số).

Ví dụ:

```
MOV  AL, 0Dh     ; AL:=0Dh
```

Lệnh này thường được dùng để nạp dữ liệu cần thao tác vào thanh ghi nào đó.

2.2.3. Chế độ địa chỉ tực tiếp

Một toán hạng là địa chỉ offset của ô nhớ chứa giá trị cần thao tác. Toán hạng kia là thanh ghi (không được là ô nhớ).

Ví dụ:

```
MOV  AL, [04FCh] ; đưa vào thanh ghi AL nội dung ô nhớ có địa chỉ logic
                 ; DS:04FCh.
```

2.2.4. Chế độ địa chỉ gián tiếp qua thanh ghi

Một toán hạng là thanh ghi chứa địa chỉ offset của ngăn nhớ chứa giá trị cần thao tác. Toán hạng kia là thanh ghi (không được là ngăn nhớ).

Ví dụ:

```
MOV  [BX], AL    ; Đưa nội dung thanh ghi AL vào ngăn nhớ có địa chỉ
                 ; offset nằm trong thanh ghi BX.
```

2.2.5. Chế độ địa chỉ tương đối cơ sở

Thanh ghi BX, BP và các hằng số là giá trị dịch chuyển để xác định ngăn nhớ trong DS, SS chứa giá trị cần thao tác.

Ví dụ:

```
MOV  [BX + N], CL      ; Đưa nội dung thanh ghi CL vào ngăn nhớ có địa
chỉ
                       ; offset BX + N (DS:BX + N)
MOV  AL, [BP + N]; Đưa nội ngăn nhớ có địa chỉ offset BP + N vào thanh
                 ; ghi AL (SS:BP + N)
```

2.2.6. Chế độ địa chỉ tương đối chỉ số

Thanh ghi DI, SI và các hằng số là giá trị dịch chuyển để xác định ngăn nhớ chứa giá trị trong DS cần thao tác.

Ví dụ:

```
MOV  [SI + N], AH ; Đưa nội dung thanh ghi AH vào ngăn nhớ có địa chỉ
                  ; offset tại SI + N (DS:SI + N).
MOV  CL, [DI + N] ; Đưa nội dung ngăn nhớ có địa chỉ offset tại DI + N
                  ; (DS:DI + N) vào thanh ghi CL.
```

2.2.7. Chế độ địa chỉ tương đối chỉ số cơ sở

Chế độ địa chỉ này là sự kết hợp cả 2 chế độ địa chỉ trên. Dùng cả thanh ghi cơ sở và thanh ghi chỉ số để tính địa chỉ toán hạng.

Ví dụ:

```
MOV  [BX + SI + N], AX  ; Đưa nội dung thanh ghi AX vào ngăn nhớ có địa
                        ; chỉ offset tại BX + SI + N (DS:BX + SI + N).
MOV  CL, [BP + DI + N]  ; Đưa nội dung ngăn nhớ có địa chỉ offset tại
                        ; BP + DI + N (DS:BP + DI + N) vào th.ghi CL.
```

2.3. Mô tả tập lệnh của 8088

Tập lệnh của bộ vi xử lý 8088 nói chung được chia thành 6 nhóm, với 5 nhóm thao tác dữ liệu và 1 nhóm đặc biệt (gồm các chỉ thị để điều khiển). Một lệnh thường có cấu trúc như sau:

Mã lệnh đích, nguồn

2.3.1. Nhóm lệnh chuyển dữ liệu

Nhóm này thực hiện vận chuyển dữ liệu (sao chép - copy) từ nơi này đến nơi khác.

MOV đích, nguồn (Move a Word or Byte)

Lệnh thực hiện chuyển dữ liệu từ nguồn tới đích (đích<=nguồn), dữ liệu có thể là một byte hoặc một word (từ). Các toán hạgn có thể được tìm thấy qua các chế độ địa chỉ khác nhau. Lệnh này không tác động đến các cờ.

LDS đích, nguồn (Load Register and DS with Word from Memory)

Lệnh thực hiện nạp một word từ bộ nhớ vào thanh ghi cho trong lệnh và một word tiếp theo vào thanh ghi DS (đích <= nguồn, DS <= nguồn+2).

Ứng dụng: thường nạp địa chỉ đầu của vùng nhớ chứa chuỗi nguồn vào SI và DS trước khi dùng đến lệnh thao tác chuỗi. Lệnh này không tác động đến các cờ.

Ví dụ:

```
LDS   SI, Str      ; Lệnh nạp vào SI nội dung 2 ô nhớ Str và Str+1,
                   ; rồi nạp vào DS nội dung 2 ô nhớ Str+2 và Str+3,
                   ; nằm trong DS.
```

LEA đích, nguồn (Load Effective Address)

Lệnh thực hiện nạp địa chỉ hiệu dụng vào thanh ghi. Toán hạng 'đích' thường là một trong các thanh ghi: BX, CX, DX, BP, SI, DI. Toán hạng 'nguồn' là tên biến trong đoạn DS được chỉ ra trong lệnh hoặc ô nhớ cụ thể.

đích<=địa chỉ lệch của nguồn (đích<=@nguồn), đích <= địa chỉ hiệu dụng của nguồn.

Lệnh này không tác động đến các cờ.

Ví dụ:

```
LEA   DX, Str      ; Lệnh nạp địa chỉ offset của Str vào DX
LEA   CX, [BX]     ; Lệnh nạp địa chỉ hiệu dụng (EA-Effective Address)
                   ; EA = BX
LEA   CX, [BX][DI]       ; Lệnh nạp địa chỉ hiệu dụng EA = BX + DI
```

LES đích, nguồn

Lệnh này giống lệnh LDS nhưng 2 byte tiếp theo được nạp vào thanh ghi ES.

Ứng dụng: thường nạp vào DI và ES địa chỉ đầu của vùng nhớ chứa chuỗi trước khi thực hiện các lệnh thao tác chuỗi. Lệnh này không tác động đến các cờ.

Ví dụ:

```
LES   DI, Str      ; Lệnh nạp địa chỉ offset của Str vào DX
```

IN Acc, Port (Input data from a port)

Đọc dữ liệu từ cổng vào/ra vào thanh ghi Acc (Acc <= {Port}). Với {Port} là dữ liệu của cổng có địa chỉ là Port (là địa chỉ 8 bit = 00h..FFh).

Acc là AL => đọc 8 bit từ cổng Port, là AX => đọc 16 bit từ cổng Port và Port+1.

Thường dùng DX để chứa địa chỉ cổng nên có thể có địa chỉ từ 0000h..FFFFh

Viết lệnh: IN Acc, DX

Lệnh này không tác động đến các cờ.

Ví dụ:

```
MOV  DX, 07F8h
IN     AL, DX        ; Đọc 1 byte từ cổng có địa chỉ 07F8h
```

OUT Acc, Port (Output a byte or a word to a port)

Đưa dữ liệu từ thanh ghi Acc đến cổng vào/ra.

Acc => Port.

Lệnh này thao tác ngược lại với lệnh IN, tính chất hoàn toàn tương tự. Lệnh này không tác động đến các cờ.

STC (Set the Carry fag): Lập cờ nhớ: CF <= 1

Lệnh này không tác động đến các cờ khác.

STD (Set the Direcion fag): Lập cờ hướng: DF <= 1

Lệnh này định hướng thao tác cho các lệnh làm việc với chuỗi theo chiều lùi (<=). Các thanh ghi liên quan: SI, DI sẽ được tự động giảm khi làm việc xong với 1 phần tử của chuỗi. Lệnh này không tác động đến các cờ.

STC (Set the Interrupt fag): Lập cờ cho phép ngắt: IF <= 1

Lệnh lập IF để cho các yêu cầu ngắt tác động vào chân INTR được CPU nhận biết. IF=1, INTR=1 => CPU bị ngắt, cất thanh ghi cờ, địa chỉ trở về vào Stack rồi thực hiện chương trình con phục vụ ngắt. Lệnh này không tác động đến các cờ khác.

CLC (Clear the Carry flag): Xoá cờ nhớ: CF <= 0.

Lệnh này không tác động đến các cờ khác.

CLD (Clear the Direction flag): Xoá cờ hướng: DF <= 0.

Lệnh định hướng thao tác cho các lệnh làm việc với chuỗi theo chiều tiến (=>). Các thanh ghi liên quan: SI, DI sẽ tự động tăng 1 khi làm việc xong với một phần tử dữ liệu của chuỗi. Lệnh này không tác động đến các cờ khác.

CLI (Clear the Interrupt flag): Xoá cờ cho phép ngắt: IF <= 0.

Khi thực hiện lệnh này, các ngắt che được sẽ bị che. Lệnh này không tác động đến các cờ khác.

CMC (Complement the Carry flag): Đảo cờ nhớ: CF $\overline{\text{<=}}$ CF

Lệnh chỉ cập nhật CF, không tác động đến các cờ khác.

PUSH nguồn

Push Word on the Stack: Cất 1 từ vào ngăn xếp

- SP <= SP-2
- {SP} <= nguồn

Toán hạng nguồn tìm được theo các chế độ địa chỉ khác nhau. Lệnh này thường đi cặp với lệnh POP. Lệnh này không tác động đến các cờ.

POP đích

Pop Word from top of Stack: Lấy 1 word từ đỉnh ngăn xếp vào thanh ghi

- đích <= {SP}
- SP <= SP+2

Toán hạng đích tìm được theo các chế độ địa chỉ (không được là thanh ghi đoạn mã: CS). Dữ liệu để tại ngăn xếp không thay đổi. SS không thay đổi. Lệnh này không tác động đến các cờ.

POPF

Pop Word from top of Stack to Flag register: Lấy 1 word từ đỉnh ngăn xếp vào thanh ghi cờ.

- RF <= {SP}
- SP <= SP+2

Dữ liệu để tại ngăn xếp không thay đổi. SS không thay đổi. Lệnh này không tác động đến các cờ.

2.3.2. Nhóm lệnh số học

ADC đích, nguồn (Add with carry: cộng 2 toán hạng có nhớ)

đích <= đích+nguồn+CF

Các toán hạng đích, nguồn tìm được theo các chế độ địa chỉ, phải chứa dữ liệu cùng độ dài (cùng kiểu). Không được là 2 ô nhớ và không được là thanh ghi đoạn. Điều này áp dụng hầu hết cho tất cả các lệnh số học có cú pháp tương tự.

Lệnh cập nhật các cờ: AF, CF, OF, PF, SF, ZF.

+ ADD đích, nguồn (Add: cộng 2 toán hạng có nhớ)

đích <= đích+nguồn

Các toán hạng đích, nguồn tìm được theo các chế độ địa chỉ, phải chứa dữ liệu cùng độ dài (cùng kiểu). Không được là 2 ô nhớ và không được là thanh ghi đoạn. (Tính chất giống với lệnh ADC).

Lệnh cập nhật các cờ: AF, CF, OF, PF, SF, ZF.

SBB đích, nguồn (Substract with Borrow: Trừ có mượn)

đích <= đích-nguồn-CF

Các toán hạng đích, nguồn tìm được theo các chế độ địa chỉ, phải chứa dữ liệu cùng độ dài (cùng kiểu). Không được là 2 ô nhớ và không được là thanh ghi đoạn. (Tính chất giống với lệnh ADC).

Lệnh cập nhật các cờ: AF, CF, OF, PF, SF, ZF.

AF, PF chỉ liên quan đến 8 bit thấp

SUB đích, nguồn (Substract: Trừ 2 toán hạng)

đích <= đích-nguồn

Các toán hạng đích, nguồn tìm được theo các chế độ địa chỉ, phải chứa dữ liệu cùng độ dài (cùng kiểu). Không được là 2 ô nhớ và không được là thanh ghi đoạn. (Tính chất giống với lệnh ADC).

Lệnh cập nhật các cờ: AF, CF, OF, PF, SF, ZF.

AF, PF chỉ liên quan đến 8 bit thấp

Chú ý: Các ví dụ cho các lệnh ADC, ADD, SBB, SUB có thể tham khảo trong tài liệu Kỹ thuật Vi xử lý - Văn Thế Minh hoặc phụ lục của tài liệu này.

MUL nguồn (Multiply Unsigned Byte or Word)

Nhân số không dấu. Toán hạng 'nguồn' là số nhân, tìm được theo các chế độ địa chỉ. Tuỳ theo độ dài (kích thước) của toán hạng 'nguồn' mà ta có các trường hợp sau:

- 'nguồn' là 8 bit (1 byte): AX <= AL*nguồn, số bị nhân phải là số 8 bit để trong thanh ghi AL.
- 'nguồn' là 16 bit (2 byte): DXAX <= AX*nguồn, số bị nhân phải là số 16 bit để trong thanh ghi AX.

Nếu byte cao (hoặc 16 bit cao) của 16 (hoặc 32) bit kết quả chứa 0 thì CF=OF=0. Vậy CF, OF cho ta biết có thể bỏ đi bao nhiêu bit 0 trong kết quả của lệnh nhân này.

Lệnh cập nhật: CF, OF

Không xác định: AF, PF, SF, ZF

DIV nguồn (Unsigned Divide)

Chia số không dấu. Toán hạng 'nguồn' là số chia, tìm được theo các chế độ địa chỉ. Tuỳ theo độ dài (kích thước) của toán hạng 'nguồn' mà ta có các trường hợp sau:

- 'nguồn' là 8 bit (1 byte): AL <= lấy nguyên(AX/nguồn), AH <= lấy dư(AX/nguồn). Số bị chia phải là số 16 bit để trong thanh ghi AX.
- 'nguồn' là 16 bit (2 byte):AX <= lấy nguyên(DXAX/nguồn), DX <= lấy dư(DXAX/nguồn). Số bị chia phải là số 32 bit để theo thứ tự trong cặp thanh ghi DXAX.

Phần nguyên được lấy làm tròn xuống theo số nguyên sát dưới.

Nếu 'nguồn' = 0 hoặc thương lớn hơn FFh (hoặc FFFFh tuỳ theo độ dài toán hạng 'nguồn') thì 8088/8086 thực hiện ngắt INT 0.

Không xác định các cờ: AF, CF, OF, PF, SF, ZF.

IMUL nguồn (Integer Multiplication (Multiply signed Number))

Nhân số có dấu. Toán hạng 'nguồn' là số nhân, có thể tìm được theo các chế độ địa chỉ. Tuỳ theo độ dài (kích thước) toán hạng 'nguồn' mà ta có các trường hợp sau:

- 'nguồn' là 8 bit (1 byte): AX <= AL*nguồn, số bị nhân phải là số 8 bit có dấu để trong thanh ghi AL.
- 'nguồn' là 16 bit (2 byte): DXAX <= AX*nguồn, số bị nhân phải là số 16 bit có dấu để trong thanh ghi AX.

Nếu tích thu được nhỏ, không lấp đầy được hết các chỗ dành cho nó thì các bit không dùng đến được thay bằng bit dấu.

Nếu byte cao (hoặc 16 bit cao) của 16 bit (hoặc 32 bit) kết quả chỉ chứa các giá trị của dấu thì CF=OF=0.

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

Vậy CF, OF: cho ta biết kết quả có độ dài thực chất là bao nhiêu.

Lệnh cập nhật: CF, OF.

Không xác định: AF, PF, SF, ZF.

IDIV nguồn (Integer Division (Signed Divide))

Chia số có dấu. Toán hạng 'nguồn' là số chia, có thể tìm được theo các chế độ địa chỉ. Lệnh này dùng để chia các số nguyên có dấu. Chỗ để ngầm định của số chia, số bị chia, thương và phần dư giống như của lệnh DIV đã giới thiệu trên. Chỉ khác là:

- AL chứa thương (số có dấu), AH chứa phần dư (số có dấu). Dấu của số dư trùng với dấu của số bị chia (AX hoặc DXAX)
- Nếu 'nguồn' = 0 hoặc thương nằm ngoài -128..127 (hoặc ngoài -32768..32767 tuỳ độ dài (kích thước) của toán hạng 'nguồn') thì 8088/8086 thực hiện lệnh INT 0.

NEG đích (Negate a Operand (From its 2's Complement))

Đảo dấu một toán hạng (lấy bù 2 của một toán hạng). Toán hạng đích có thể tìm được theo các chế độ địa chỉ.

đích <= 0-đích = not(đích+1)

Lệnh cập nhật: AF, CF, OF, PF, SF, ZF.

INC đích (Increment Destination register or Memory)

Tăng toán hạng đích lên 1. Toán hạng đích có thể tìm được theo các chế độ địa chỉ.

đích <= đích+1

Nếu đích = FFh (hoặc FFFFh) thì đích-1 = 00h (hoặc 0000h) mà không ảnh hưởng cờ CF. Lệnh này tương đương ADD đích,1.

Lệnh cập nhật: AF, OF, PF, SF, ZF

Không xác định CF

DEC đích (Decrement Destination register or Memory)

Giảm toán hạng đích đi 1. Toán hạng đích có thể tìm được theo các chế độ địa chỉ.

đích <= đích-1

Nếu đích = 00h (hoặc 0000h) thì đích-1 = FFh (hoặc FFFFh) mà không ảnh hưởng cờ CF. Lệnh này tương đương ADD đích,1.

Lệnh cập nhật: AF, OF, PF, SF, ZF

Không xác định CF

XCHG đích, nguồn (Exchange 2 Operands)

Hoán đổi nội dung 2 toán hạng.

đích <=> nguồn

Các toán hạng đích, nguồn tìm được theo các chế độ địa chỉ, phải chứa dữ liệu có cùng độ dài (kích thước-kiểu). Không được đồng thời là 2 ô nhớ, không được là thanh ghi đoạn. Sau khi thực hiện XCHG thì toán hạng này chứa nội dung cũ của toán hạng kia và ngược lại.

Lệnh này không tác động đến các cờ.

Lệnh này thuộc nhóm lệnh vận chuyển dữ liệu.

2.3.3. Nhóm lệnh logic, dịch và quay

- **Các lệnh logic**

NOT đích (Invert Each bit of an Operand (From its 1's Complement))

Lệnh lấy bù 1 của một toán hạng (đảo bit của một toán hạng). Toán hạng đích tìm được theo các chế độ địa chỉ.

đích <= not(đích)

Lệnh này không tác động đến các cờ.

AND đích, nguồn (And Corresponding bits of Two Operand)

Và 2 toán hạng.

đích <= đích∧nguồn

Các toán hạng đích, nguồn tìm được theo các chế độ địa chỉ, phải chứa dữ liệu cùng độ dài (kích thước), không được là 2 ô nhớ, không được là thanh ghi đoạn.

Ứng dụng: Thường dùng lệnh AND để 'che' đi (giữ lại) một số bit nào đó của toán hạng bằng cách nhân logic (AND) toán hạng đó với một toán hạng tức thì có các bit 0, 1 ở các vị trí cần 'che' (giữ lại). Toán hạng tức thì lúc này gọi là mặt nạ.

Lệnh xoá cờ CF, OF. Cập nhật: PF, SF, ZF (PF chỉ có nghĩa khi toán hạng 8 bit), không xác định cờ AF.

OR đích, nguồn (Logically Or Corresponding bits of Two Operands)

Hoặc 2 toán hạng.

đích <= đích∨nguồn

Các toán hạng đích, nguồn tìm được theo các chế độ địa chỉ, phải chứa dữ liệu cùng độ dài (kích thước), không được là 2 ô nhớ, không được là thanh ghi đoạn.

Ứng dụng: Thường dùng lệnh Or để lập một số bit nào đó của toán hạng bằng cách cộng logic (Or) toán hạng đó với một toán hạng tức thì có các bit 0, 1 ở các vị trí cần lập.

Lệnh xoá cờ CF, OF. Cập nhật: PF, SF, ZF (PF chỉ có nghĩa khi toán hạng 8 bit), không xác định cờ AF.

XOR đích, nguồn (Exclusive Or Corresponding bits of Two Operands)

Hoặc loại trừ 2 toán hạng.

đích <= đích⊕nguồn

Các toán hạng đích, nguồn tìm được theo các chế độ địa chỉ, phải chứa dữ liệu cùng độ dài (kích thước), không được là 2 ô nhớ, không được là thanh ghi đoạn. Theo tính chất của

phép hoặc loại trừ, nếu toán hạng đích trùng toán hạng gốc thì kết quả bằng không (kết quả = 0) và các cờ CF, OF cũng bị xoá.

Ứng dụng: Lệnh thường được dùng xoá về 0 môt thanh ghi nào đó.

Lệnh xoá cờ CF, OF. Cập nhật: PF, SF, ZF (PF chỉ có nghĩa khi toán hạng 8 bit), không xác định cờ AF.

- **Các lệnh dịch, quay**

SAL đích, CL (Shift Arithmetically Left): Dịch trái số học.

SHL đích, CL (Shift (Logically) Left): Dịch trái logic.

Toán hạng đích tìm được theo các chế độ địa chỉ. Hai lệnh này có cùng tác động là dịch trái số học (còn gọi là dịch trái logic). Mỗi lần dịch MSB được đưa qua cờ CF và giá trị 0 được đưa vào LSB, thao tác như vậy được gọi là dịch logic. CL phải chứa sẵn số lần dịch mong muốn.

Trong trường hợp muốn dịch 1 lần, ta viết lệnh trực tiếp: *SAL đích, 1*

Ứng dụng: Mỗi lần dịch, tương đương việc nhân toán hạng với 2 của số không dấu. Vậy nếu muốn nhân một số không dấu với 2i thì ta dịch trái số bị nhân đi i lần. Nói chung lệnh này chạy nhanh hơn lệnh MUL.

Cờ OF <= 1 nếu khi dịch một lần mà MSB thay đổi. Không xác định (không đúng) sau nhiều lần dịch.

CF <= MSB sau mỗi lần dịch, vì vậy lệnh này còn dùng để tạo cờ CF từ giá trị của MSB làm điều kiện cho các lệnh nhảy có điều kiện.

Cập nhật: SF, PF, ZF (PF chỉ có nghĩa khi toán hạng là 8 bit)

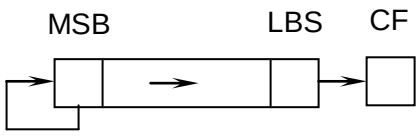
Không xác định AF.

SAR đích, CL (Shift Arithmetically Right): Dịch phải số học.

Toán hạng đích tìm được theo các chế độ địa chỉ. Lệnh có tác dụng dịch phải số học toán hạng. Sau mỗi lần dịch MSB được giữ lại, LSB được đưa vào cờ CF. CL chứa sẵn số lần dịch.

Trong trường hợp muốn dịch 1 lần, ta viết lệnh trực tiếp: *SAR đích, 1*

Ứng dụng: Sau mỗi lần dịch, tương đương việc chia toán hạng với 2 của số có dấu. Vậy nếu muốn chia một số có dấu với 2i thì ta dịch phải số bị chia đi i lần (vì vậy gọi là dịch phải số học). Nói chung lệnh này chạy nhanh hơn lệnh IDIV.



Cờ OF <= 1 nếu khi dịch một lần mà LSB thay đổi. Không xác định (không đúng) sau nhiều lần dịch.

CF <= LSB sau mỗi lần dịch, vì vậy lệnh này còn dùng để tạo cờ CF từ giá trị của LSB làm điều kiện cho các lệnh nhảy có điều kiện.

Cập nhật: SF, PF, ZF (PF chỉ có nghĩa khi toán hạng là 8 bit).

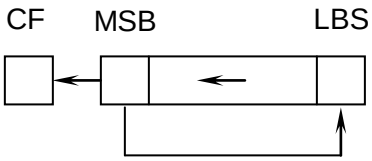
Không xác định AF.

SHR đích, CL (Shift (Logically) Right): Dịch phải logic.

Toán hạng đích được tìm theo các chế độ địa chỉ. Lệnh có tác động giống các lệnh SAL, SHL nhưng theo chiều ngược lại.

ROL đích, CL (Rotate All Bits to the Left): Quay vòng sang trái.

Toán hạng đích tìm được theo các chế độ địa chỉ. Lệnh có tác dụng quay vòng toán hạng sang trái. MSB được đưa qua cờ CF và LSB. CL phải chứa sẵn số lần quay.



Trong trường hợp muốn quay 1 lần, ta viết lệnh trực tiếp: *ROL đích, 1*

Ta thấy, nếu CL=8 và toán hạng đích là 8 bit thì kết quả không bị thay đổi vì toán hạng quay tròn đúng 1 vòng, còn nếu CL=4 thì 2 nibble của toán hạng bị đổi chỗ cho nhau.

Cờ OF <= 1 nếu khi quay một lần mà MSB thay đổi. Không xác định (không đúng) sau nhiều lần quay.

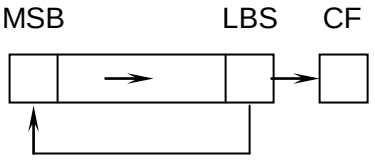
CF <= MSB sau mỗi lần quay, vì vậy lệnh này còn dùng để tạo cờ CF từ giá trị của MSB làm điều kiện cho các lệnh nhảy có điều kiện.

Cập nhật: CF, OF (chỉ 2 cờ này bị ảnh hưởng).

Không xác định AF.

ROR đích, CL (Rotate All Bits to the Left): Quay vòng sang phải.

Toán hạng đích tìm được theo các chế độ địa chỉ. Lệnh có tác dụng quay vòng toán hạng sang phải. LSB được đưa qua cờ CF và MSB. CL phải chứa sẵn số lần quay (tác động của lệnh này giống lệnh ROL nhưng theo chiều ngược lại).



RCL đích, CL (Rotate though CF to the Left): Quay trái qua cờ CF.

Lệnh này có thể quay toán hạng sang trái thông qua cờ CF, CL phải chứa sẵn số lần quay.

Trong trường hợp muốn quay 1 lần, ta viết lệnh trực tiếp: *RCL đích, 1*

Ta thấy, nếu CL=9 và toán hạng đích là 8 bit thì kết quả không bị thay đổi vì CF cùng với toán hạng thanh ghi (8 bit) quay đúng một vòng.

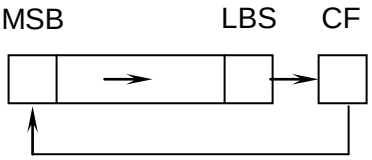
Cờ OF <= 1 nếu khi quay một lần mà LSB thay đổi. Không xác định (không đúng) sau nhiều lần quay.

CF <= MSB sau mỗi lần quay.

Cập nhật: CF, OF (chỉ 2 cờ này bị ảnh hưởng).

RCR đích, CL (Rotate though CF to the Right): Quay phải qua cờ CF.

Lệnh này có thể quay toán hạng sang phải thông qua cờ CF, CL phải chứa sẵn số lần quay.



Trong trường hợp muốn quay 1 lần, ta viết lệnh trực tiếp: *RCR đích, 1*

Ta thấy, nếu CL=9 và toán hạng đích là 8 bit thì kết quả không bị thay đổi vì CF cùng với toán hạng thanh ghi (8 bit) quay đúng một vòng.

Cờ OF <= 1 nếu khi quay một lần mà MSB thay đổi. Không xác định (không đúng) sau nhiều lần quay.

CF <= LSB sau mỗi lần quay.

Cập nhật: CF, OF (chỉ 2 cờ này bị ảnh hưởng).

2.3.4. Nhóm lệnh so sánh

CMP đích, gốc (Compare Byte or Word): So sánh 2 byte hay 2 từ (word).

đích-gốc;

Toán hạng	CF	ZF
đích = gốc	0	1
đích > gốc	0	0
đích < gốc	1	0

Toán hạng đích, gốc tìm được theo các chế độ địa chỉ, phải chứa dữ liệu có cùng độ dài (kích thước). Không được đồng thời là 2 ô nhớ, không được là thanh ghi đoạn.

Lệnh chỉ tạo cờ, không lưu kết quả so sánh và không làm thay đổi giá trị các toán hạng. Lệnh này thường dùng kèm với các lệnh nhảy có điều kiện (nhảy theo cờ).

Khi so sánh 2 số không dấu, ta có quan hệ như bảng bên.

Lệnh cập nhật: AF, CF, OF, PF, SF, ZF.

TEST đích, gốc (And Operands to Update Flag): Và 2 toán hạng để tạo cờ.

đích∧gốc;

Toán hạng đích, gốc tìm được theo các chế độ địa chỉ, phải chứa dữ liệu có cùng độ dài (kích thước). Không được đồng thời là 2 ô nhớ, không được là thanh ghi đoạn.

Lệnh chỉ tạo cờ, không lưu kết quả. Sau lệnh này giá trị các toán hạng không bị thay đổi. Các cờ được tạo làm điều kiện cho các lệnh nhảy có điều kiện.

Lệnh xoá cờ CF, OF. Cập nhật: PF, SF, ZF (PF chỉ liên quan đến 8 bit thấp), không xác định cờ AF.

2.3.5. Nhóm lệnh rẽ nhánh (nhảy), lặp

Việc sử dụng nhóm các lệnh nhảy có thể làm thay đổi tính tuần tự của các câu lệnh hợp ngữ.

- **Lệnh nhảy không điều kiện**

JMP Nhãn (Uncondition Jump to Specified Destination) Nhảy (vô điều kiện) đến đích Nhãn

Lệnh mới bắt đầu tại địa chỉ ứng với nhãn 'Nhãn'. Lệnh Jmp có thể nhảy lên (về phía địa chỉ thấp) hoặc nhảy xuống (về phía địa chỉ cao) và có thể nhảy xa được tối đa 1/2 đoạn (64Kbyte)

Lệnh này không tác động đến các cờ.

- **Lệnh nhảy có điều kiện (nhảy khi thoả mãn một số điều kiện)**

Nhắc lại một số từ tiếng Anh

A: Above (trên)	Z: Zero (không)	S: Signed
B: Below (dưới)	G: Greater (lớn hơn)	PO: Parity Old
N: Not (phủ định – không)	L: Less (nhỏ hơn)	PE: Parity Even
E: Equal (bằng)	P: Parity	C: Carry (nhớ)
O: Overflow		

JA/JNBE	(CF+ZF = 0, so sánh không dấu)
JAE/JNB/JNC	(CF = 0, so sánh không dấu)
JB/JC/JNAE	(CF = 1, so sánh không dấu)
JCXZ	(CX = 0, không so sánh)
JE/JZ	(ZF = 1, so sánh không dấu)
JG/JNLE	((SF⊕OF)+ZF = 0, so sánh có dấu)
JGE/JNL	((SF⊕OF) = 0, so sánh có dấu)
JL/JNGE	((SF⊕OF) = 1, so sánh có dấu)
JLE/JNG	((SF⊕OF)+ZF = 1, so sánh có dấu)
JNE/JNZ	(ZF = 0, so sánh không dấu)
JNO	(OF = 0)
JNP/JPO	(PF = 0)
JNS	(SF = 0)
JO	(OF = 1)
JP/JPE	(PE = 1)
JS	(SF = 1)

(Chú ý: Các lệnh nhảy trên không tác động đến các cờ)

- **Các lệnh lặp**

LOOP NHAN (Jump to Specified Label if CX<>0 after autodecrement)

Lặp lại đoạn chương trình do NHAN chỉ ra cho đến khi CX=0. Lệnh này dùng để thực hiện lặp lại đoạn chương trình trong khoảng từ NHAN đến hết lệnh LOOP NHAN cho đến khi số lần lặp CX=0. Số lần lặp CX phải được nạp sẵn từ trước. Sau mỗi lần thực hiện lệnh LOOP NHAN thì CX tự động giảm 1 (CX <= CX-1).

Lệnh này không tác động đến các cờ.

LOOPE/LOOPZ NHAN (Loop while CX<>0 and ZF=1)

Lệnh thực hiện lặp đoạn chương trình do NHAN chỉ ra cho đến khi CX=0 hoặc ZF=0. Số lần lặp CX phải được nạp từ trước. Sau mỗi lần lặp thì CX tự động giảm 1 (CX <= CX-1).

Lệnh này không tác động đến các cờ.

LOOPNE/LOOPNZ NHAN (Loop while CX<>0 and ZF=0)

Lệnh thực hiện lặp đoạn chương trình do NHAN chỉ ra cho đến khi CX=0 hoặc ZF=1. Số lần lặp CX phải được nạp từ trước. Sau mỗi lần lặp thì CX tự động giảm 1 (CX <= CX-1).

Lệnh này không tác động đến các cờ.

REP (Repeat string instruction until CX=0)

Thực hiện lặp lại lệnh đứng sau đó cho đến khi CX=0. Đây là 'tiếp đầu ngữ' dùng để viết trước các lệnh thao tác với chuỗi mà ta muốn lặp lại một số lần. Số lần lặp CX phải được nạp từ trước. Sau mỗi lần lặp thì CX tự động giảm 1 (CX <= CX-1).

REPE/REPZ (Repeat string instruction until CX=0 or ZF=0)

Lặp lại lệnh viết sau đó cho đến khi CX=0 hoặc ZF=0

REPNE/REPNZ (Repeat string instruction until CX=0 or ZF=1)

Lặp lại lệnh viết sau đó cho đến khi CX=0 hoặc ZF=1

2.3.6. Nhóm các lệnh đặc biệt

CALL TEN_ctc

Lệnh gọi chương trình con có tên là TEN_ctc trong cùng đoạn mã với chương trình chính. Chương trình này phải nằm trong giới hạn dịch chuyển –32Kbyte..32Kbyte-1 so với lệnh tiếp theo ngay sau lệnh CALL. Sau khi cất IP vào Stack, IP <= IP+Dịch chuyển. Nếu chương trình con nằm ở đoạn mã khác thì trong chương trình hợp ngữ TEN_ctc phải được khai báo là một chương trình con ở xa.

TEN_ctc Proc Far

HTL (Hall Processing): Dừng

Khi gặp lệnh này các hoạt động của 8088 bị tạm dừng (bước vào trạng thái dừng) cho đến khi có tác động vào chân INTR, NMI hoặc Reset của nó.

INT N (Interrupt Program Execution): Ngắt, gián đoạn chương trình đang được thực hiện.

(N=00h..FFh, gọi là số hiệu ngắt)

Khi gặp lệnh này, 8088 thực hiện các thao tác sau:

- SP <= SP-2, {SP} <= FR
- IF <= 0 (cấm ngắt), TF <= 0 (chạy suốt)
- SP <= SP-2, {SP} <= CS
- SP <= SP-2, {SP} <= IP
- IP <= {N*4}, CS <= {N*4+2}

Ví dụ:

Với N=5 thì:

CS <= {0016h}
IP <= {0014h}

INTO (Interrupt on Overflow): Ngắt nếu có tràn.

Nếu OF=1 thì lệnh ngắt công việc đang làm của bộ vi xử lý và thực hiện lệnh ngắt INT 4.

IRET(Interrupt Return)

Trở về chương trình chính (CTC) từ chương trình con phục vụ ngắt (ISR – Interrup Service Routin).

Tại cuối ISR phải có lệnh IRET để bộ vi xử lý tự động lấy lại địa chỉ trở về CTC và lấy lại thanh ghi cờ.

RET(Return from Procedure to Calling Program): Trở về CTC từ ctc

Viết lệnh: RET hoặc RET n (n: nguyên, dương)

Lệnh RET được đặt tại cuối chương trình con để bộ vi xử lý biết tự động lấy lại địa chỉ trở về chương trình chính.

Nếu dùng lệnh RET n thì sau khi lấy lại địa chỉ trở về (chỉ có IP hoặc có cả IP và CS) thì SP <= SP+n (dùng để nhảy qua, không muốn lấy lại các thông số khác của chương trình còn lại trong stack).

Lệnh này không tác động đến các cờ.

WAIT (Wait for TEST or INTR signal): Chờ tín hiệu từ chân TEST hoặc chân INTR

Lệnh đưa bộ vi xử lý vào trạng thái nghỉ cho đến khi có tín hiệu mức thấp tác động vào chân TEST hoặc tín hiệu mức cao tác động vào chân INTR. Nếu có yêu cầu ngắt và yêu cầu ngắt này được chấp nhận khi 8088 đang ở trạng thái nghỉ thì sau khi thực hiện xong ISR tương ứng, 8088 lại trở lại trạng thái nghỉ.

NOP (No Operation): CPU không làm gì cả

Lệnh này không làm gì, nó chỉ tăng giá trị của thanh ghi IP và tiêu tốn 3 chu kỳ đồng hồ (xung clock).

Lệnh này không tác động đến các cờ

Ứng dụng: Thường được tính thời gian trong các vòng trễ chính xác.

Chương 3
LẬP TRÌNH BẰNG HỢP NGỮ CHO 8088 TRÊN MÁY TÍNH IBM PC VÀ CÁC MÁY TƯƠNG THÍCH IBM PC

3.1. Giới thiệu chung

Sau khi đã giới thiệu một số lệnh cơ bản của bộ vi xử lý 8088 => ta sẽ dùng các lệnh đó để lập trình dùng hợp ngữ trên các máy tính IBM PC (hoặc các máy tương thích máy IBM PC). Vì loại máy tính này có cấu trúc tiêu biểu của một hệ vi xử lý, hơn nữa ta cũng có thể sử dụng nhiều chức năng sẵn có cho chương trình thông qua các dịch vụ (các chương trình con phục vụ ngắt) của các ngắt của DOS và của BIOS. Có thể sử dụng chương trình dịch hợp ngữ MASM 5.10 (Macro Assembler phiên bản 5.10) của Microsoft với các định nghĩa đoạn đơn giản và chế độ bộ nhớ nhỏ. Ngoài ra ta cũng có thể sử dụng chương trình dịch hợp ngữ TASM 2.0 (Turbo Assembler phiên bản 2.0) của Borland International để thực hiện dịch chương trình của chúng ta.

Ngôn ngữ assembly (hợp ngữ)

Các chương trình thực hiện chuyển đổi chương trình của người sử dụng được viết bằng một ngôn ngữ nào đó sang một ngôn ngữ khác được gọi là chương trình dịch (translate). Ngôn ngữ được sử dụng để viết chương trình nguồn được gọi là ngôn ngữ nguồn còn ngôn ngữ của chương trình mà do chương trình nguồn chuyển sang được gọi là ngôn ngữ đích.

Người ta đã phân chương trình dịch làm 2 loại dựa trên mối quan hệ giữa ngôn ngữ nguồn và ngôn ngữ đích như sau:

- Khi ngôn ngữ nguồn về căn bản là một sự biểu diễn bằng ký hiệu cho một ngôn ngữ máy bằng số thì chương trình dịch được gọi là ***assembler*** và ngôn ngữ nguồn được gọi là ngôn ngữ ***assembly (hợp ngữ)***.
- Khi ngôn ngữ nguồn là một ngôn ngữ bậc cao như Pascal, C, . . . và ngôn ngữ đích là ngôn ngữ máy hoặc là một biểu diễn bằng ký hiệu cho một ngôn ngữ như vậy thì chương trình dịch được gọi là ***compiler***.

Ngôn ngữ assembly thuần khiết là ngôn ngữ mà trong đó mỗi lệnh (chỉ thị) của nó khi được dịch sinh ra đúng một chỉ thị máy, điều đó có nghĩa là có sự tương ứng 1-1 giữa các lệnh máy và các lệnh trong ngôn ngữ ***assembly***. Nếu mỗi dòng trong chương trình assembly chứa một chỉ thị assmebly và mỗi word trong bộ nhớ chứa một lệnh máy thì chương trình dài n dòng sẽ sinh ra một chương trình ngôn ngữ máy dài n word .

Sử dụng ngôn ngữ assmebly để lập trình dễ hơn sử dụng ngôn ngữ máy (dạng số, là dãy các bit) rất nhiều. Việc sử dụng tên và địa chỉ bằng ký hiệu thay cho số nhị phân (hoặc hệ 8, 10, 16) tạo nên sự khác biệt lớn. Mọi người dễ dàng có thể nhớ được các ký hiệu (symbol) viết tắt cho lệnh cộng (add), trừ (substract), nhân (multiply) và chia (divide) là ADD, SUB, MUL, DIV nhưng ít ai có thể nhớ được các lệnh máy cho các phép toán đó dưới dạng số, ví dụ là: 24576, 57344, 28672 và 29184 (trừ khi làm việc quá nhiều với chúng mà tự nhiên nhớ được). Người lập trình bằng ngôn ngữ assembly chỉ cần nhớ các tên bằng ký hiệu gợi nhớ ADD, SUB, MUL, DIV, . . . vì chúng sẽ được assembler dịch ra các lệnh máy. Tuy nhiên nếu ai muốn lập trình bằng ngôn ngữ máy thì họ cần phải nhớ mã lệnh dưới dạng số (hoặc liên tục tra cứu).

Đối với địa chỉ, cũng rút ra các nhận xét tương tự. Người lập trình bằng ngôn ngữ asembly có thể đặt tên bằng ký hiệu gợi nhớ cho các ô nhớ và giao cho assembly phải cung

cấp đúng địa chỉ bằng số, trong khi đó người lập trình bằng ngôn ngữ máy luôn luôn phải làm việc với các giá trị bằng số của các địa chỉ.

Vì vậy mà từ khi có ngôn ngữ assembly ra đời cho đến nay, không còn ai viết chương trình bằng ngôn ngữ máy nữa.

Ngoài sự tương ứng (ánh xạ) 1-1 của các lệnh assembly vào các lệnh máy, ngôn ngữ assembly còn có một tính chất khác nữa làm cho nó khác hẳn các ngôn ngữ lập trình bậc cao, đó là người lập trình bằng ngôn ngữ assembly có thể truy cập tới tất cả các đặc điểm trong máy tính vật lý. Ví dụ, nếu có một bit báo tràn số (Overflow bit) thì chương trình bằng ngôn ngữ assembly có thể truy cập và kiểm tra trực tiếp bit này, trong khi đó chương trình bằng ngôn ngữ bậc cao (Pascal, C …) không thể làm được việc đó.

Một sự khác biệt lớn và quan trong nữa giữa chương trình assembly và chương trình bằng ngôn ngữ bậc cao là chương trình bằng ngôn ngữ assembly chỉ có thể chạy được trên một họ máy, trong khi đó chương trình được viết bằng ngôn ngữ bậc cao nói chung có thể chạy được trên nhiều họ máy, đây chính là một ưu điểm lớn của ngôn ngữ bậc cao so với ngôn ngữ assembly.

Nói chung, tất cả các việc có thể thực hiện được bằng ngôn ngữ máy đều có thể thực hiện được bằng ngôn ngữ assembly, tuy nhiên ngôn ngữ bậc cao không làm được như vậy một cách hiệu quả.

Khi xây dựng các ứng dụng lớn, thông thường người ta chọn ngôn ngữ bậc cao vì nó hướng tới thuật toán giải quyết vấn đề (Ngôn ngữ hướng bài toán - problem-oriented language) mà không chọn ngôn ngữ assembly vì khi đó người lập trình phải chú ý tới các chi tiết nhỏ nhặt khi lập trình. Ngược lại, khi xây dựng các chương trình nhỏ thực hiện các thao tác can thiệp sâu vào phần cứng máy tính thì người ta thường chọn ngôn ngữ assembly vì tính tối ưu, hiệu quả và khả năng mạnh mẽ của nó.

3.2. Giới thiệu khung chương trình

Với bất kỳ ngôn ngữ nào, khi ta lập trình bằng ngôn ngữ đó ta cũng phải tuân thủ chương trình viết đúng cú pháp, quy định khung chương trình. Từ đó chương trình mới được dịch ra mã máy, rồi mới tạo ra các chương trình chạy được (phần mở rộng: ***.EXE** hoặc ***.COM**).

Một chương trình hợp ngữ bao gồm các dòng lệnh, mỗi lệnh được viết trên một dòng

- Một dòng lệnh có thể là lệnh thật dưới dạng gợi nhớ (nmenomic) hay dạng ký hiệu (symbolic) của bộ vi xử lý.
- Hoặc hướng dẫn chương trình dịch (Essembler directive).

Lệnh thật dưới dạng gợi nhớ sẽ được dịch ra mã máy còn hướng dẫn chương trình dịch thì không, nó chỉ có tác dụng chỉ dẫn cho chương trình dịch thực hiện công việc trong quá trình dịch.

Lệnh có thể được viết dưới dạng chữ hoa hay chữ thường đều được, chúng được cho là tương đương vì đối với các dòng lệnh, chương trình dịch không phân biệt kiểu chữ.

3.2.1. Cấu trúc của một lệnh hợp ngữ

Một dòng lệnh của chương trình hợp ngữ (assembly) có cấu trúc như sau:

Tên(Nhãn)	Mã lệnh	Các toán hạng	Giải thích

Ví dụ:

LAP: Mov CL, AH ;Số lần lặp được đặt trong thanh ghi CL

- LAP là nhãn
- Mov là mã lệnh
- CL, AH là các toán hạng
- Và trường giải thích bắt đầu bằng dấu chấm phẩy (;)

Main Proc

- Main là tên
- Proc là mã của lệnh giả hay hướng dẫn chương trình dịch (dùng để bắt đầu chương trình hoặc bắt đầu chương trình con)

Một lệnh không nhất thiết phải có đầy đủ các trường như trên. Tuỳ từng công việc cụ thể mà lệnh có thể khuyết một hoặc một số trường nào đó.

Trường tên (Nhãn)

Trường này chứa nhãn, tên biến, tên hằng hoặc tên thủ tục của chương trình. Tên và nhãn sẽ được chương trình dịch gán bằng các địa chỉ cụ thể của ô nhớ.

Quy tắc đặt tên (cũng khá giống như quy tắc đặt tên trong ngôn ngữ Pascal)

- Dùng các ký tự thuộc bộ chữ cái (không phân biệt chữ hoa, chữ thường).
- Không được bắt đầu bằng chữ số, không được chứa dấu cách.
- Độ dài: 1..21 ký tự.
- Có thể sử dụng các ký tự đặc biệt như: ?, ., _, @, $, %.
- Trong trường hợp nếu dùng dấu chấm (.), thì nó phải được đặt ở vị trí đầu tiên của tên hoặc nhãn.
- Nhãn thường kết thúc bằng dấu hai chấm (;).

Trường mã lệnh

Trường này gồm mã các lệnh thật hoặc giả (hướng dẫn chương trình dịch):

- Lệnh thật: lệnh dạng gợi nhớ (nmemonic) của bộ vi xử lý. Lệnh này sẽ được chương trình dịch dịch ra mã máy.
- Hướng dẫn chương trình dịch thì không được dịch.

Trường các toán hạng

Trường này là dữ liệu cho các thao tác, tuỳ từng lệnh cụ thể mà có thể có 2, 1 hoặc không có toán hạng nào.

Ví dụ:

```
Mov     al, al   ; Lệnh này có 2 toán hạng
Rol     bx, cl   ; Lệnh này có 2 toán hạng
Not     bl       ; Lệnh này có 1 toán hạng
Ret     n        ; Lệnh này có 1 toán hạng
Ret              ; Lệnh này không có toán hạng nào
Sti              ; Lệnh này không có toán hạng nào
Nop              ; Lệnh này không có toán hạng nào
```

Với hướng dẫn chương trình dịch, trường này chứa các thông tin khác nhau liên quan đến các lệnh giả của hương dẫn.

Trường giải thích

Trường này được bắt đầu bằng dấu chầm phẩy (;), sau đó là dòng giải thich. Chương trình dịch sẽ bỏ qua không dịch trường này.

Lệnh tuy được viết dưới dạng gợi nhớ của bộ vi xử lí, tuy nhiên chúng ta luôn nên có trường này. Lời giải thích cần sát nghĩa của công việc thực hiện (không nên giải thích ý nghĩa của câu lệnh).

3.2.2. Dữ liệu cho chương trình hợp ngữ

Dữ liệu cho (của) một chương trình hợp ngữ có thể ở dạng hệ 2, hệ 10, hệ 16 hoặc dạng ký tự. Đối với chương trình Debug (dùng để tìm lỗi cho các chương trình hợp ngữ) thì dữ liệu bằng số được ngầm định ở dạng hệ 16. Còn đối với assembly thì dữ liệu bằng số được ngầm định ở hệ 10. Khi cung cấp dữ liệu cho chương trình, số cho ở hệ nào thì phải kèm hậu tố của hệ đó (trừ hệ 10 - ngầm định). Riêng đối với hệ 16, nếu số bắt đầu bằng chữ cái (a..f hoặc A..F) thì phải thêm số 0 ở trước để chương trình dịch không nhầm với một tên hoặc nhãn nào đó. (B-Binary: Hệ 2; D-Decimal: Hệ 10; H-Hexa: Hệ 16).

Ví dụ:

```
1001b   ; Số ở hệ 2
100     ; Số ở hệ 10
0ah     ; Số ở hệ 16
```

Nếu dữ liệu cho dưới dạng ký tự thì phải bao đóng (đặt) ký tự trong cặp dấu nháy đơn.

Ví dụ:

```
'a'     ; Ký tự a
'abcd'  ; Chuỗi ký tự
```

Với kiểu ký tự, ngoài cách trên ta còn có thể dùng mã ASCII của ksy tự đó.

Ví dụ:

```
'0'     ; Ký tự 0
30h     ; Mã ASCII của ký tự 0
```

Với 2 cách viết trong ví dụ trên là như nhau đối với chương trình dịch assembler.

3.2.3. Biến và hằng

Một biến bất kỳ được sử dụng trong chương trình hợp ngữ phải được định nghĩa, chương trình dịch sẽ gán cho biến đó một địa chỉ xác định trong bộ nhớ.

Biến đơn

Một biến đơn trong chương trình hợp ngữ được định nghĩa theo mẫu sau:

Tên_biến Kiểu Giá trị khởi tạo

- Tên: do người sử dụng tự đặt theo quy tắc đặt tên.
- Kiểu: là kích thước (phạm vi) biểu diễn của biến. Có các kiểu sau:
  - DB (Define Byte): Kiểu byte (1 byte).
  - DW (Define Word): Kiểu word (2 byte).
  - DD (Define Double Word): Kiểu double word (4 byte).

- DF (Define Farword): Kiểu farword (6 byte), chỉ dùng với bộ vi xử lý 80386 trở lên.
- DQ (Define Quadword): Kiểu Quadword (8 byte).
- DT (Define Ten byte): Kiểu Ten byte (10 byte).

Trong một biến có kích tước lớn hơn 1 byte thì byte cao ở địa chỉ cao, byte thấp ở địa chỉ thấp (theo quy ước Big-endian của Intel).

Ví dụ:

```
Ab      db      4       ; Định nghĩa một biến có tên là Ab, kích thước 1 byte và
                        ; được khởi tạo giá trị bằng 4.
Ab1     db      ?       ; Định nghĩa một biến có tên là Ab1, kích thước 1 byte và
                        ; được chưa được khởi tạo giá trị.
Ab2     dw      100h    ; Định nghĩa một biến có tên là Ab2, kích thước 2 byte và
                        ; được được khởi tạo giá trị bằng 100h = 256.
Ab3     dw      ?       ; Định nghĩa một biến có tên là Ab3, kích thước 2 byte và
                        ; được chưa được khởi tạo giá trị.
```

Biến mảng

Một biến mảng trong chương trình hợp ngữ được định nghĩa theo mẫu sau:

Tên_biến	Kiểu	Các giá trị khởi tạo

- Tên: do người sử dụng tự đặt theo quy tắc đặt tên.
- Kiểu: là kích thước (phạm vi) biểu diễn của biến như đã biết.

Biến mảng là biến hình thành từ một dãy liên tiếp các phần tử cùng kiêu. Khi định nghĩa biến mảng ta gán tên cho một dãy liên tiếp các phần tử có cùng độ dài (kích thước) trong bộ nhớ cùng với các giá trị ban đầu tương ứng.

Ví dụ:

```
Ar      db      1, 3, 2, 4
```

Định nghĩa một biến có tên là Ar, gồm 4 phần tử, mỗi phần tử có kích thước 1 byte (gồm 4 byte được dành chỗ cho nó trong bộ nhớ từ địa chỉ ứng với Ar để chứa các giá trị khởi đầu là: 1, 2, 3 và 4). Phần tử đầu tiên của mảng có địa chỉ trùng với địa chỉ của Ar và có giá trị là 1, phần tử thứ 2 có địa chỉ là Ar+1 và có giá trị là 2, …

Ta có thể dùng toán tử DUP để khởi đầu giá trị các phần tử của mảng với cùng một giá trị.

Ví dụ:

```
Ar1     dw      100DUP(5)
```

Định nghĩa một biến có tên là Ar1, gồm 100 phần tử, mỗi phần tử có kích thước 2 byte (gồm 200 byte được dành chỗ cho nó trong bộ nhớ từ địa chỉ ứng với Ar1 để chứa với cùng một giá trị khởi đầu cho mỗi 2 byte (word) là 5). Phần tử đầu tiên của mảng có địa chỉ trùng với địa chỉ của Ar1, các phần tử tiếp theo có địa chỉ Ar1+2, Ar1+4, Ar1+6, …

```
Ar2     dd      20DUP(?)
```

Định nghĩa một biến có tên là Ar2, gồm 20 phần tử, mỗi phần tử có kích thước 4 byte (gồm 80 byte được dành chỗ cho nó trong bộ nhớ từ địa chỉ ứng với Ar2 và chưa được khởi

đầu giá trị). Phần tử đầu tiên của mảng có địa chỉ trùng với địa chỉ của Ar, các phần tử tiếp theo có địa chỉ Ar2+4, Ar2+8, Ar2+12, …

Đặc biệt ta có thể dùng toán tử DUP lồng nhau để khởi đầu giá trị các phần tử của mảng.

Ví dụ:

```
Ar3     db      2, 2, 2DUP(1, 3DUP(5), 4)
```

Định nghĩa một biến có tên là Ar3, gồm 12 phần tử, mỗi phần tử có kích thước 1 byte (gồm 12 byte được dành chỗ cho nó trong bộ nhớ từ địa chỉ ứng với Ar3 để chứa các giá trị khởi đầu cho mỗi byte). Phần tử đầu tiên của mảng có địa chỉ trùng với địa chỉ của Ar3, các phần tử tiếp theo có địa chỉ Ar3+1, Ar3+2, …

Dãy thứ tự giá trị các phần tử là: 2, 2, 1, 5, 5, 5, 4, 1, 5, 5, 5, 4.

- **Biến xâu**

Biến kiểu xâu ký tự là trường hợp dặc biệt của biến mảng mà các phần tử của mảng là ký tự. Một xâu ký tự có thể được định nghĩa bằng các ký tự, xâu ký tự hoặc bằng mã ASCII của các ký tự.

Chúng ta có thể định nghĩa biến xâu ký tự theo các dòng ví dụ sau, chúng là tương đương nhau:

Ví dụ:

```
Str1    db      'Co non'
Str2    db      'C', 'o', ' ', 'n', 'o', 'n'
Str3    db      'C', 'o', ' ', 'non'
Str4    db      43h, 6fh, 32h, 6eh, 6fh, 6eh
Str5    db      43h, 'o', 32h, 'n', 6fh, 6eh
```

- **Hằng**

Trong chương trình hợp ngữ, các giá trị không đổi thường được gán tên làm cho chương trình rõ ràng, dễ đọc hơn - gọi là các hằng. Hằng trong chương trình có thể là kiểu số hoặc kiểu ký tự. Việc gán tên cho hằng được thực hiện nhờ lệnh giả EQU (Equate) theo mẫu sau:

```
Tên_hằng   Equ   Giá trị khởi tạo
```

Ví dụ:

```
Cr      Equ     0dh     ; Carriage return
Lf      Equ     0ah     ; Line feed
Pa      Equ     3f8h
Clause  Equ     'Co non xanh tan chan troi'
Str     db      Clause,'$'
Str1    db      Clause, Cr, Lf, '$'
```

3.2.4. Khung của một chương trình hợp ngữ

Một chương trình mã máy được nạp vào bộ nhớ thường bao gồm các vùng nhớ khác nhau:

- Vùng dữ liệu: Dùng để chứa các biến, kết quả trung gian hay kết quả khi chạy chương trình.
- Vùng mã lệnh: Dùng để chứa mã lệnh của chương trình.
- Vùng ngăn xếp: Dùng để phục vụ cho các hoạt động của chương trình như gọi chương trình con, trở về chương trình chính từ chương trình con.

Một chương trình hợp ngữ cũng có cấu trúc như vậy, để khi được dịch nó sẽ tạo ra mã tương ứng với chương trình mã máy nói trên (có cấu trúc giống như vậy). Chúng ta sẽ khai báo quy mô sử dụng bộ nhớ đối với các vùng nhớ đó để sử dụng một cách phù hợp, tiết kiệm, hiệu quả và đúng với cấu trúc chương trình.

Khai báo quy mô sử dụng bộ nhớ

Khai báo này xác định kích thước cho đoạn mã và dữ liệu của chương trình.

Sử dụng hướng dẫn chương trình dịch .Model đặt trước các hướng dẫn khác trong chương trình theo mẫu như sau:

```
.Model    Kích_thước
```

Ví dụ:

```
.Model  Small
.Model  Tiny
```

Có các kiểu Kích_thước bộ nhớ cho chương trình hợp ngữ như sau:

- Tiny (hẹp): Mã lệnh và dữ liệu nằm gọn trong một đoạn.
- Small (nhỏ): Mã lệnh trong một đoạn, dữ liệu trong một đoạn.
- Medium (Trung bình): Mã lệnh hơn một đoạn, dữ liệu trong một đoạn.
- Compact (Gọn): Mã lệnh trong một đoạn, dữ liệu hơn một đoạn.
- Large (Lớn), Huge (Rất lớn - khổng lồ): Mã lệnh và dữ liệu hơn một đoạn. Các mảng có thể lớn hơn 64Kbyte.

Khai báo đoạn ngăn xếp

Ngăn xếp là vùng nhớ phục vụ cho các hoạt động của chương trình khi gọi cheơng trình con và trở về chương trình chính từ chương trình con. Tuỳ theo cấu trúcvà quy mô của chương trình mà ta khai báo kích thước của đoạn này. Việc khai báo được thực hiện nhờ hướng dẫn chương trình dịch .Stack theo mẫu sau:

```
.Stack Kích_thước
```

Ví dụ:

```
.Stack   100
.Stack   100h   ; 256
```

Chú ý: Nếu ta không khai báo kích thước của đoạn này thì chương trình dịch sẽ tự động gán giá trị 1 Kbyte cho vùng ngăn xếp này. Đây là kích thước quá lớn đối với một ứng dụng thông thường. Nói chung ta nên chọn là 100 hoặc 100h là đủ.

Khai báo đoạn dữ liệu

Phần này để định nghĩa các biến của chương trình.

Hằng cũng nên định nghĩa ở đây để đảm bảo sự thống nhất (mặc dù ta có thể định nghĩa hằng ở chỗ khác, lý do là lệnh giả EQU không cấp phát bộ nhớ cho hằng (tên hằng không

tương ứng với một địa chỉ nào) nên ta có thể định nghĩa hằng tự do thoải mái trong chương trình.

Việc khai báo đoạn dữ liệu được thực hiện nhờ hướng dẫn chương trình dịch **.Data**. Định nghĩa các biến, mảng và hằng được thực hiện tiếp ngay sau đó bằng các lệnh giả thích hợp, của thể như sau:

Ví dụ:

```
.Data
    Chao    db    'Xin chao ban!','$'
    Crlf    db    0dh, 0ah, '$'
    Pa      Equ   300h
```

- **Khai báo đoạn mã lệnh**

Phần này chứa toàn bộ mã lệnh của chương trình. Việc khai báo đoạn mã được thực hiện nhờ hướng dẫn chương trình dịch .Code như sau:

```
.Code
  Tên_CTC   Proc
      Các lệnh            ; Các lệnh của thanh chương trình chính
      Call  Tên_ctc       ; Gọi chương trình con
  Tên_CTC   Endp
```

Tổng quát: Một thủ tục được định nghĩa nhờ cặp thủ tục 'Proc - Endp ', chương trình chính cũng là một thủ tục được định nghĩa như trên. Lệnh giả Proc dùng để báo bắt đầu một thủ tục và lệnh giả Endp dùng để báo kết thúc thủ tục đó. Một chương trình con cũng được định nghĩa dưới dạng một thủ tục nhờ các lệnh giả 'Proc - Endp ' như sau:

```
Tên_ctc   Proc
    Các lệnh của chương trình con ở đây
    Ret
Tên_ctc   Endp
```

Chú ý: Trong chương trình con, tại cuối chương trình có lệnh Ret là lệnh trở về chương trình chính từ chương trình con.

Để kết thúc toàn bộ chương trình, ta dùng hướng dẫn chương trình dịch End như sau:

```
End Tên_CTC
```

- **Khung của chương trình hợp ngữ để dịch ra chương trình *.exe**

```
.Model    Small
.Stack    100
.Data
    ;Định nghĩa các biến, mảng, hằng ở đây
.Code
Tên_CTC   Proc
  ;Khởi tạo đoạn dữ liệu
  Mov     ax, @Data
  Mov     ds, ax
  Mov     es, ax          ; Nếu cần
  ; Các lệnh của chương trình chính
```

```
    ; Trở về DOS dùng hàm 4ch của ngắt 21h
    Mov     ah, 4ch
    Int     21h
Tên_CTC  Endp
     ;Các chương trình con nếu có được định nghĩa ở đây
End  Tên_CTC
```

Khi chương trình ***.exe** được nạp vào bộ nhớ, DOS sẽ lập ra một một mảng gọi là đoạn mào đầu chương trình (Program Segment Prefix - PSP) gồm 256 byte dùng để chứa các thông tin liên quan đến chương trình và cả DOS, được gắn vào đầu chương trình. DOS sử dụng các thông tin này để giúp chạy chương trình, PSP được DOS khởi tạo cho mọi chương trình dù chúng được viết bằng ngôn ngữ nào. Do ngay khi chương trình được nạp vào bộ nhớ, DOS cũng đưa các thông số liên quan đến chương trình vào các thanh ghi DS và ES (cụ thể là DS và ES trỏ vào đầu của PSP) mà không chứa giá trị địa chỉ của các thanh ghi đoạn dữ liệu của chương trình. Để chương trình chạy đúng, ta phải khởi đầu cho các thanh ghi DS và ES nhờ các lệnh:

```
Mov     ax, @Data
Mov     ds, ax
Mov     es, ax
```

Với 8088/8086 và một số bộ vi xử lý khác thuộc họ 80x86 của Intel, vì lý do kỹ thuật mà chúng không cho phép chuyển giá trị số (chế độ địa chỉ trực tiếp) vào các thanh ghi đoạn nên ta phải dùng thanh ghi **ax** làm trung gian. Thanh ghi **ax** cũng có thể thay thế bằng các thanh ghi đa năng khác.

@Data là tên của đoạn dữ liệu, **.Data** ddịnh nghĩa bởi hướng dẫn chương trình dịch => chương trình dịch sẽ dịch tên **@Data** thành giá trị địa chỉ của đoạn dữ liệu.

Chương trình ví dụ1.asm để dịch ra *,exe, thực hiện xuất một dòng ký tự lên màn hình. Dòng ký tự ở đây là lời chào bất kỳ được hiện giữa 2 dòng trống:

```
.Model      Small
.Stack      100
.Data
      Chao  db      'Chuc ban da thanh cong voi chuong trinh dau tay$'
      Crlf  db      0dh, 0ah, '$'
.Code
Vidu1       Proc
      Mov   ax, @Data     ; Lấy địa chỉ của đoạn dữ liệu
      Mov   ds, ax        ; Khởi tạo đoạn dữ liệu
      Mov   es, ax
      ; Về đầu dòng mới dùng hàm 9 của ngắt 21h để "hiển thị" cặp ký tự
      ; xuống dòng (lf: line feed) và về đầu dòng (cr: carriage return)
      Mov   ah, 9
      Lea   dx, crlf
      Int   21h
      ; Hiển thị lời chào dùng hàm 9 của ngắt 21h
      Mov   ah, 9
      Lea   dx, Chao
      Int   21h
      ; Về đầu dòng mới dùng hàm 9 của ngắt 21h
```

```
        Mov   ah, 9
        Lea   dx, crlf
        Int   21h
        ; Về DOS dùng hàm 4ch của ngắt 21h
        Mov   ah, 4ch
        Int   21h
Vidu1   Endp
End Vidu1
```

- **Khung của chương trình hợp ngữ để dịch ra chương trình *.com**

Với khung chương trình hợp ngữ để dịch ra tệp chương trình chạy được ***.exe** thì có mặt đầy đủ các đoạn. Ngoài tệp chương trình chạy được có phần mở rộng ***.exe** ra ta còn có khả năng dịch chương trình hợp ngữ có kết cấu (cấu trúc) thích hợp ra một loại chương trình chạy được kiểu khác với phần mở rộng ***.com**. Đây là chương trình ngắn gọn và đơn giản hơn nhiều so với tệp chương trình ***.exe** mà trong đó các đoạn: đoạn mã, đoạn dữ liệu và đoạn ngăn xếp của chương trình được gói gọn trong một đoạn (64Kbyte) duy nhất là đoạn mã. Với những ứng dụng mà dữ liệu và mã lệnh của chương trình không yêu cầu nhiều về không gian nhớ thì ta có thể ghép luôn chúng chung vào cùng một đoạn mã rồi tạo ra tệp ***.com**. Việc tạo ra tệp này không chỉ tiết kiệm được thời gian và bộ nhớ khi cho chạy chương trình mà còn tiết kiệm cả không gian nhớ khi phải lưu trữ chúng trên bộ nhớ ngoài (đĩa từ).

Để có thể tạo ra được chương trình với phần mở rộng ***.com** thì chương trình nguồn hợp ngữ phải có kết cấu thích hợp, một ví dụ như sau:

```
.Model      Tiny
.Code
        ORG  100h
Start: Jmp   Continue
        ; Định nghĩa các biến, mảng, hằng ở đây
Continue:
Tên_CTC   Proc
        ; Các lệnh của chương trình chính
        ; Trở về DOS dùng ngắt 20h
        Int   20h
Tên_CTC   Endp
        ; Các chương trình con nếu có được định nghĩa ở đây
End  Start
```

Nhìn vào khung chương trình hợp ngữ để dịch ra chương trình .com ta thấy không có khai báo đoạn ngăn xếp và đoạn dữ liệu, khai báo quy mô sử dụng bộ nhớ là Tiny (tuy nhiên có thể sử dụng quy mô bộ nhớ là Small). ở đầu đoạn mã có lệnh giả Org (Origin: điểm xuất phát) và lệnh Jmp (nhảy). Lệnh Org 100h dùng để gán địa chỉ bắt đầu cho chương trình là 100h trong đoạn mã, bỏ qua vùng nhớ kích thước 100h (256 byte) cho đoạn mào đầu (PSP) từ địa chỉ 0 đến địa chỉ 255.

Lệnh Jmp dùng để nhảy qua phần bộ nhớ dành cho việc định nghĩa các dữ liệu (về nguyên tắc, dữ liệu có thể được đặt ở đầu hoặc cuối đoạn mã nhưng ở đây, nó được đặt ở đầu để có thể áp dụng các định nghĩa đơn giản đã nói). Đích của lệnh nhảy là phần bắt đầu của chương trình chính.

Đặc điểm của chương trình *.com

Vì dung lượng nhớ cực đại của một đoạn là 64Kbyte, nên ta phải chắc chắn rằng chương trình của ta có số lượng byte của mã lệnh và dữ liệu là không lớn (không vượt quá giới hạn cho phép của một đoạn, nếu không nó sẽ làm cho cả nhóm nở ra về phía địa chỉ cao của đoạn).

Chương trình phải sử dụng ngăn xếp một cách hạn chế, nếu không nó sẽ làm cho đỉnh ngăn xếp dâng lên về phía địa chỉ thấp của đoạn khi hoạt động.Chúng ta phải đảm bảo rằng không thể xảy ra hiện tượng chùm lên nhau của các thông tin tại vùng ngăn xếp và thông tin tại vùng mã lệnh và dữ liệu.

Khi kết thúc chương trình kiểu ***.com**, để trở về DOS ta dùng ngắt 20h của DOS để làm cho chương trình gọn hơn. Mặc dù ta vẫn có thể dùng hàm 4ch của ngắt 21h để trở về DOS như đã dùng trong chương trình để dịch ra ***.exe**.

Khi kết thúc toàn bộ chương trình ta dùng hướng dẫn chương trình dịch END kèm theo nhãn Start. Nhãn Start tương ứng địa chỉ lệnh đầu tiên của chương trình trong đoạn mã.

Chúng ta có thể viết lại chương trình trong ví dụ trước (để dịch ra ***.exe**) thực hiện việc xuất một xâu ký tự lên màn hình theo khung chương trình để dịch ra ***.com**:

```
.Model      Tiny
.Code
      ORG   100h
Start: Jmp  Continue
      Chao  db      'Xin chào . . .$'
      Crlf  db      0dh, 0ah,'$'
Continue:
Main        Proc
      Mov   ah, 9           ; Về đầu dòng mới dùng hàm 9 của ngắt 21h
      Lea   dx, Crlf
      Int   21h             ; Hiển thị lời chào
      Mov   ah, 9
      Lea   dx, Chao
      Int   21h
      Mov   ah, 9           ; Về đầu dòng mới dùng hàm 9 của ngắt 21h
      Lea   dx, Crlf
      Int   21h
      Int   20h             ; Trở về DOS dùng ngắt 20h
Main        Endp
      ; Các chương trình con nếu có được định nghĩa ở đây
End  Start
```

3.3. Cách tạo và cho chạy một chương trình hợp ngữ

Các bước thực hiện việc tạo ra và cho chạy một chương trình hợp ngữ như sau:

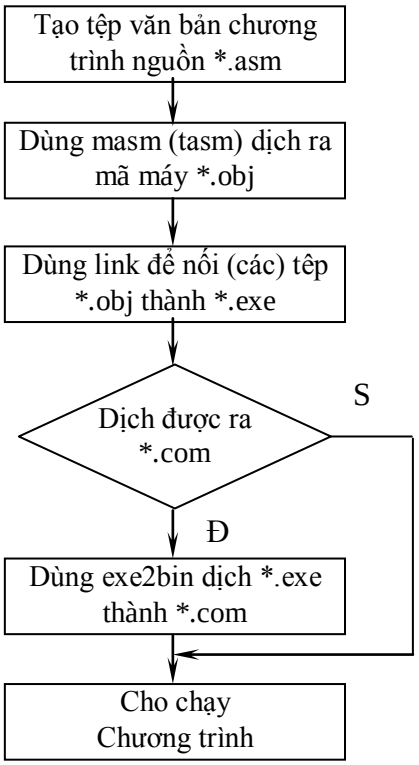
(1) Soạn thảo văn bản chương trình nguồn (dùng các phần mềm soạn thảo như: SK, NCedit, Bked, Turbo Pascal, . . .), nên dùng NCedit cho đơn giản. Tệp chương trình nguồn này phải được gán phần mở rộng là ***.asm**.

(2) Dùng chương trình dịch MASM (hoặc TASM) để dịch tệp ***.asm** ra mã máy dưới dạng ***.obj**. Nếu trong bước này trong chương trình nguồn có lỗi cú pháp thì ta quay lại bước (1) để sửa lại chương trình nguồn.

(3) Dùng chương trình LINK để nối một hay nhiều tệp ***.obj** lại với nhau thành một chương trình chạy được ***.exe**.

(4) Nếu chương trình viết ra để dịch ra kiểu chương trình ***.com** thì ta phải dùng chương trình EXE2BIN của DOS để dịch tiếp tệp ***.exe** ra tệp chương trình ***.com**.

(5) Cho chạy chương trình vừa dịch.



3.4. Các cấu trúc lập trình cơ bản trong assembly

Thông thường trong thực tế, người ta thường phân tích bài toán và thiết kế chương trình (hệ thống nói chung) bằng phương pháp thiết kế từ trên xuống (top-down) tương ứng với kỹ thuật lập trình có cấu trúc.

Nội dung của phương pháp là chia bài toán tổng thể (hay chương trình đang thiết kế) thành những bài toán nhỏ hơn (có thể là các khối chức năng). Các bài toán nhỏ này lại được chia thành các bài toán nhỏ hơn nữa cho đến khi mỗi bài toán nhỏ này trở thành những bài toán đơn giản, dễ thực hiện.

Việc lập trình giải quyết bài toán nhỏ để tạo thành khối chức năng thành phần người ta thường sử dụng các cấu trúc lập trình cơ bản để thực hiện nhiệm vụ các khối đó. Với cách tiến hành như vậy làm cho chương trình viết ra trở thành "có cấu trúc", mang theo những ưu điểm là rõ ràng, dễ phát triển, dễ hiệu chỉnh hoặc cải tiến và nâng cấp.

Khi phân tích và viết chương trình để giải quyết các công việc khác nhau ta có các cấu trúc lập trình cơ bản sau:

- Cấu trúc tuần tự
- Cấu trúc lựa chọn:
    - if dk then s1 [else s2]
    - case V of . . . [else]
- Cấu trúc lặp:
    - while dk do s
    - repeat s1, s2, . . ., sn until dk
    - for index=v1 to v2 do s

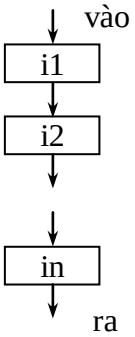
Các cấu trúc lập trình cơ bản trên đều có một đặc điểm là "tính cấu trúc". Chỉ có một lối vào cấu trúc và một lối ra cấu trúc đó.

Chúng ta đã được làm quen với các cấu trúc lập trình cơ bản khi viết chương trình trên ngôn ngữ bậc cao. Công việc của chúng ta giờ đây là làm thế nào để thực hiện các cấu trúc lập trình này bằng hợp ngữ (assembly). Chúng ta sẽ lần lượt nghiên cứu cách mô tả các cấu trúc lập trình cơ bản:

3.4.1. Cấu trúc tuần tự

Đây là cấu trúc lập trình thông dụng và đơn giản nhất. Trong cấu trúc này các công việc (các lệnh) được tiến hành (thực hiện) tuần tự, lệnh này là tiếp theo của lệnh kia. Lệnh cuối cùng thực hiện thì sẽ hoàn tất công việc của khối chức năng và ra khỏi cấu trúc.

Mô tả:




```
i1    ; lệnh 1
i2    ; lệnh 2
. . .
in    ; lệnh n
```

Mở rộng: Các lệnh i1, i2, …, in cũng có thể là một cấu trúc bất kỳ

Ví dụ:

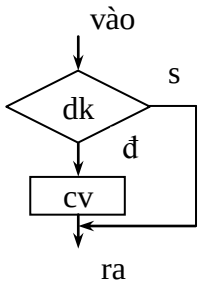
Tính biểu thức b^2 - 4ac, với: al chứa a, bl chứa b và cl chứa c:

```
Mul     cl          ; tính ax <= al*cl
Mov     cl, 2       ; số lần dịch toán hạng ax
Shl     ax, cl      ; ax*4
Mov     ax, cx      ; lưu kết quả sang cx
Mov     al, bl      ; nạp biến b vào al
Mul     bl          ; ax <= al*bl
Sub     ax, cx      ; ax <= ax-cx, ax chứa kết quả của biểu thức b2 – 4ac.
```

3.4.2. Cấu trúc lựa chọn

if dk then cv

Nếu điều kiện (dk) cho giá trị đúng thì thực hiện cv (công việc). Ngược lại công việc bị bỏ qua. Để thực hiện điều này, ta sử dụng cặp lệnh so sánh (cmp) và lệnh nhẩy có điều kiện để nhẩy qua một số lệnh (công việc nào đó) trong chương trình hợp ngữ.

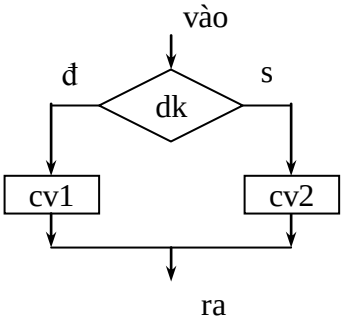


Ví dụ: Nhập một ký tự từ bàn phím, kiểm tra xem nếu ký tự đó không là ký tự điều khiển thì hiển thị lên màn hình ở đầu dòng tiếp theo, ngược lại (là ký tự điều khiển) thì không làm gì và ra khỏi cấu trúc.

```
    Mov     ah, 1       ; Nhập 1 ký tự
    Int     21h         ; bằng hàm 2 của ngắt 21h
    Mov     bl, al      ; Cất mã ASCII của ký tự nhận được
    Lea     dx, crlf            ; Xuống dòng bằng cách hiển thi xâu
    Mov     ah, 9       ; có các ký tự CR và LF
    Int     21h         ; bằng hàm 9 của ngắt 21h
    Cmp     bl, 20h     ; Kiểm tra ký tự nhận được
    Jb      Ra          ; Là ký tự điều khiển => không làm gì,
    Mov     dl, bl      ; không là ký tự điều khiển thì
    Mov     ah, 2       ; hiển thị ký tự đó
    Int     21h         ; bằng hàm 2 của ngắt 21h
Ra:                     ; Ra khỏi cấu trúc
```

if dk then cv1 else cv2

Nếu điều kiện (dk) cho giá trị đúng thì thực hiện cv1 (công việc 1). Ngược lại, thực hiện cv2 (công việc 2) qua. Để thực hiện điều này, ta sử dụng cặp lệnh so sánh (cmp) và lệnh nhẩy có điều kiện để nhẩy qua một số lệnh (công việc nào đó) trong chương trình hợp ngữ.



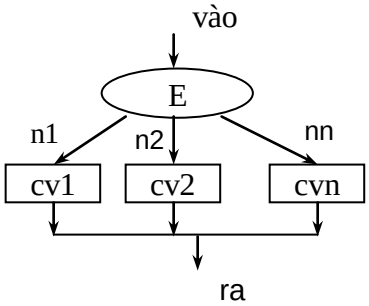
Ví dụ: Nhập một ký tự từ bàn phím, kiểm tra xem nếu ký tự đó không là ký tự điều khiển thì hiển thị lên màn hình ở đầu dòng tiếp theo, ngược lại (là ký tự điều khiển) thì hiển thị một thông báo.

```
    Mov     ah, 1       ; Nhập 1 ký tự
    Int     21h         ; bằng hàm 2 của ngắt 21h
    Mov     bl, al      ; Cất mã ASCII của ký tự nhận được
    Lea     dx, crlf            ; Xuống dòng bằng cách hiển thi xâu
    Mov     ah, 9       ; có các ký tự CR và LF
    Int     21h         ; bằng hàm 9 của ngắt 21h
    Cmp     bl, 20h     ; Kiểm tra ký tự nhận được
    Jb      Dkhien      ; Là ký tự điều khiển => hiên thị thông báo (Dkhien),
    Mov     dl, bl      ; không là ký tự điều khiển thì
    Mov     ah, 2       ; hiển thị ký tự đó
    Int     21h         ; bằng hàm 2 của ngắt 21h
    Jmp     Ra          ; Xong công việc thì ra khỏi cấu trúc
Dkhien:
    Lea     dx, mesg    ; mesg là biến xâu chứa dòng thông báo
    Mov     ah, 9       ; Dùng hàm 9
    Int     21h         ; của ngắt 21h
Ra:                     ; Ra khỏi cấu trúc
```

- **Cấu trúc Case**

Ví dụ: Nhập một ký tự từ bàn phím, kiểm tra xem:

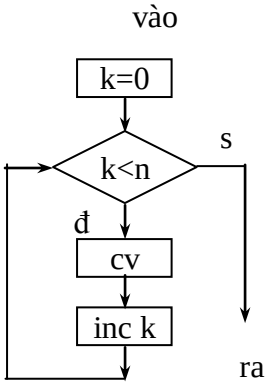
- nếu ký tự đó là ký tự ‘1’, thì hiển thị thông báo 1.
- nếu ký tự đó là ký tự ‘2’, thì hiển thị thông báo 2.
- nếu ký tự đó là ký tự ‘3’, thì hiển thị thông báo 3.
- nếu không là các ký tự trên: hiển thị thông báo 4.




```
    Mov     ah, 1       ; Nhập 1 ký tự
    Int     21h         ; bằng hàm 2 của ngắt 21h
    Mov     bl, al      ; Cất mã ASCII của ký tự nhận được
    Lea     dx, crlf            ; Xuống dòng bằng cách hiển thi xâu
    Mov     ah, 9       ; có các ký tự CR và LF
    Int     21h         ; bằng hàm 9 của ngắt 21h
    Cmp     bl, '1'     ; Kiểm tra ký tự nhận được xem có bằng ký tự '1'
    Je      Tb1         ; bằng, thì hiển thị thông báo 1. Không, kiểm tra tiếp
    Cmp     bl, '2'     ; Kiểm tra ký tự nhận được xem có bằng ký tự '2'
    Je      Tb2         ; bằng, thì hiển thị thông báo 2. Không, kiểm tra tiếp
    Cmp     bl, '3'     ; Kiểm tra ký tự nhận được xem có bằng ký tự '3'
    Je      Tb3         ; bằng, thì hiển thị thông báo 3. Không, kiểm tra tiếp
    Jmp     Tb4         ; Nếu không bằng các ký tự trên thì hiển thị thông báo 4
```

```
Tb1:
    ; Hiển thị thông báo 1
    jmp Ra              ; hiển thị xong thì ra khỏi cấu trúc
Tb2:
    ; Hiển thị thông báo 2
    Jmp Ra              ; hiển thị xong thì ra khỏi cấu trúc
Tb3:
    ; Hiển thị thông báo 3
    Jmp Ra              ; hiển thị xong thì ra khỏi cấu trúc
Tb4:
    ; Hiển thị thông báo 4
Ra:                     ; Ra khỏi cấu trúc
```

3.4.3. Cấu trúc lặp



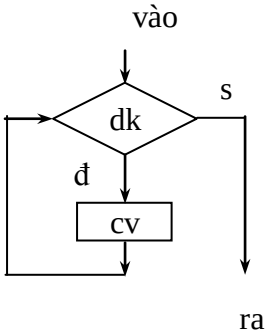
Cấu trúc For ... do

Đây là vòng lặp với số lần lặp n biết trước. Ban đầu biến chỉ số lần lặp k được gán bằng không, chừng nào nó còn nhỏ hơn n thì thực hiện lặp lại công việc (cv), k được tăng 1 sau mỗi lần thực hiện cv.

Trong sơ đồ bên, ta có thể đặt biến chỉ số k=n và kiểm tra xem sau mỗi lần lặp (thực hiện cv) thì k>0? Sẽ còn lặp khi biểu thức so sánh này là đúng, tất nhiên k sẽ được giảm 1 sau mỗi lần lặp. Điều này hoàn toàn phù hợp với sự làm việc của lệnh LOOP mà chúng ta đã biết.

Ví dụ: Hiển thị lên màn hình 80 dấu * trên một dòng

```
    Mov     ah, 2       ; Dùng hàm 2 của ngắt 21h
    Mov     dl, '*'     ; để hiển thị ký tự
    Mov     cl, 80
for:
    Int     21h
    Loop    for
```



Cấu trúc While ... do

Đây là vòng lặp với số lần lặp không biết trước. Chừng nào biểu thức điều kiện còn đúng thì thực hiện lặp lại công việc (cv), để đảm bảo cho tính dừng của giải thuật thì công việc (cv) phải có sự tác động đến dk dưới hình thức nào đó.

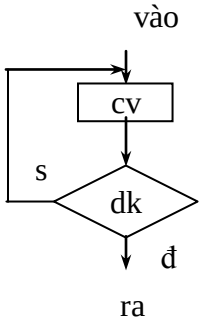
Nhìn vào sơ đồ khối của vòng lặp này ta thấy rất giống với cấu trúc của vòng lặp for - to tuy nhiên ta không thể dùng lệnh LOOP để điều khiển cho vòng lặp này vì lệnh LOOP lặp lại đoạn chương trình do NHAN chỉ ra với số nguyên lần được xác định trước trong thanh ghi CX. Ta sẽ sử dụng các lệnh nhảy có điều kiện để điều cho vòng lặp này.

Ví dụ: Tính tổng ax = 1+4+7+… Cộng cho đến khi ax>100.

```
    Xor     ax, ax      ; Ban đầu tổng tích luỹ bằng 0
    Mov     bx, 1       ; Đặt phần tử đầu tiên vào thanh ghi bx
while:
```

```
        Cmp     ax, 100         ; Kiểm tra điều kiện (yêu cầu) của bài toán
        Ja      End_while       ; ax>100 (đúng), ra khỏi cấu trúc (có thể dùng jnbe)
        Add     ax, bx          ; ngược lại, cộng tiếp
        Add     bx, 3           ; chuyển lên phần tử tiếp theo của dãy
        Jmp     while           ; Sau khi cập nhật ax, kiểm tra lại điều kiện của bài toán
End_while:                      ; Ra khỏi cấu trúc
```

- **Cấu trúc Repeat … until**



Đây là vòng lặp với số lần lặp không biết trước. Thực hiện lặp lại ccông việc (cv) cho đến khi biểu thức điều kiện (dk) đúng. Để đảm bảo cho tính dừng của giải thuật thì công việc (cv) phải có sự tác động đến dk dưới hình thức nào đó.

Thực tế, trong nhiều trường hợp cấu trúc while - do và cấu trúc repeat - until có thể thay thế cho nhau được. Sự khác nhau ở chỗ: với cấu trúc repeat - until thì công việc được thực hiện ít nhất 1 lần còn trong cấu trúc while - do thì công việc có thể không được thực hiện lần nào. Ta sẽ sử dụng các lệnh nhảy có điều kiện để điều cho vòng lặp này.

Ví dụ: Tính tổng ax = 1+4+7+ . . . + (3*(n-1)+1), cộng cho đến khi số hạng trong dãy trên >100, số hạng này không được cộng vào ax.

```
        Xor     ax, ax          ; Ban đầu tổng tích luỹ bằng 0
        Mov     bx, 1           ; Đặt phần tử đầu tiên vào thanh ghi bx
repeat:
        Add     ax, bx          ; Cộng vào tổng tích luỹ
        Add     bx, 3           ; Chuyển lên phần tử tiếp theo của dãy
        Cmp     bx, 100         ; Kiểm tra điều kiện (yêu cầu) của bài toán
        Jbe     repeat          ; nếu chưa thoả mãn thì cộng tiếp số hạng tiếp theo
        ; Ra khỏi cấu trúc
```

3.5. Truyền tham số

Khi xây dựng các ứng dụng cụ thể, ta cần quan tâm tới việc truyền tham số giữa chương trình chính cho chương trình con hoặc giữa các modul chương trình với nhau. Với assembly, người ta thường dùng các cách truyền sau:

- Truyền tham số qua thanh ghi (truyền tham trị)
- Truyền tham số qua ô nhớ (biến)
- Truyền tham số qua ô nhớ có địa chỉ trong một thanh ghi nào đó (tham biến)
- Truyền tham số qua ngăn xếp (stack).

3.6. Một số ngắt của DOS và của BIOS

Khi xây dựng các ứng dụng bằng hợp ngữ (assemby), thường là các công việc cho phép can thiệp xâu vào phần cứng máy tính, các thao tác cấp thấp nhất của các thành phần trong máy tính. Để các thao tác trên có thể đạt hiệu quả cao, ta nên sử dụng các dịch cụ của BIOS và của DOS. Đây là các ngắt làm việc với độ tin cậy rất cao và có sẵn (mặc dù ta vẫn có thể tạo ra các ngắt riêng để thực hiện các công việc tương tự nhưng sự ngắn gọn, tính tối ưu và độ tin cậy thì khó có thể sánh với các ngắt của BIOS và của DOS).

- **Các ngắt của BIOS**

Số hiệu ngắt	Hàm	Công dụng	Tham số vào	Tham số ra
10h	0	Chọn chế độ hiển thị cho màn hình.	ah=0; al=chế độ (VGA, 16 mau>: al=3)	Không
	1	Thay đổi kích thước con trỏ, phải chọn dòng quét bắt đầu và kết thúc của con trỏ.	ah=1; 4 bit thấp của ch=dòng quét đầu; 4 bit thấp của cl=dòng quét cuối	Không
	2	Dịch chuyển con trỏ (vị trí).	ah=2; bh=số trang; dh=hàng; dl=cột	Không
	3	Xác định vị trí và kích thước hiện thời của con trỏ.	ah=3; bh=số trang;	ch=dòng quét đầu; cl=dòng quét cuối; dh=dòng; dl=cột
	5	Chọn trang hiển thị.	ah=5; al=số trang; dh=dòng; dl=cột	Không
	6	Cuốn màn hình hay cửa sổ lên một số dòng xác định.	ah=6; al=số dòng cuốn (al=0 thì cuốn cả màn hình hay cửa sổ); bh=thuộc tính của các dòng trống; (ch,cl)=(dòng,cột) góc trên trái của cửa sổ; (dh,dl)=(dòng,cột) góc dưới phải của cửa sổ	Không
	7	Cuốn màn hình hay cửa sổ xuống một số dòng xác định.	ah=6; al=số dòng cuốn (al=0 thì cuốn cả màn hình hay cửa sổ); bh=thuộc tính của các dòng trống; (ch,cl)=(dòng,cột) góc trên trái của cửa sổ; (dh,dl)=(dòng,cột) góc dưới phải của cửa sổ	Không

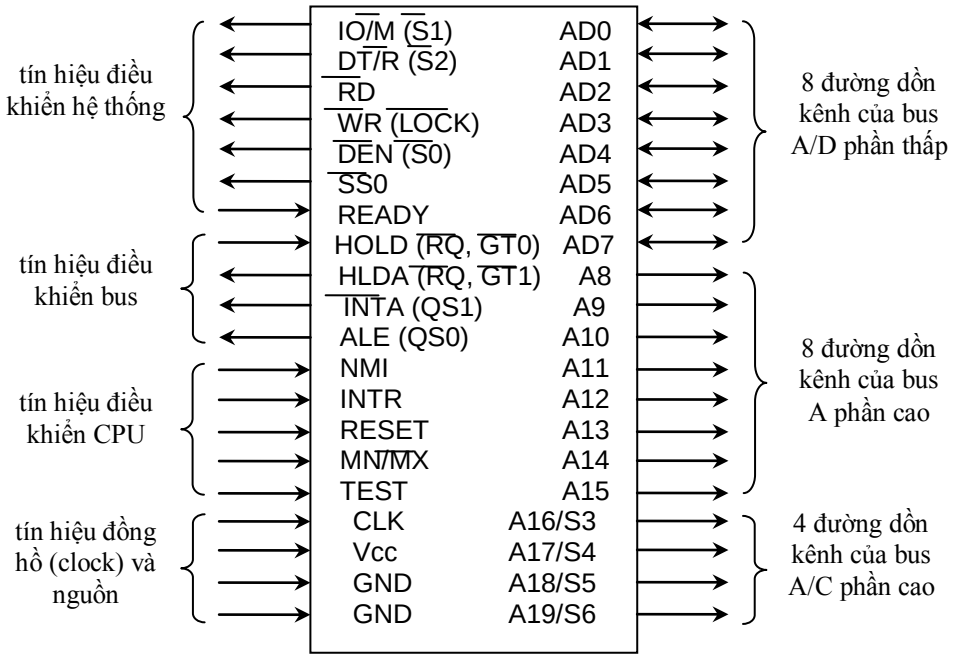
- **Các ngắt của DOS**

Số hiệu ngắt	Hàm	Công dụng	Tham số vào	Tham số ra
21h	0	Kết thúc việc thi hành một chương trình	ah=0; CS=địa chỉ đoạn của đoạn mào đầu (PSP)	Không
	1	Vào một ký tự từ bàn phím (đợi đọc 1 ký tự từ thiết bị vào chuẩn (nếu chưa có), sau đó đưa ký tự tới thiết bị ra chuẩn và đưa mã ASCII của ký tự vào al	ah=1	al=mã ASCII của ký tự
	2	Hiển thị lên màn hình (đưa ký tự có mã ASCII trong dl tới thiết bị ra chuẩn)	ah=2; dl=mã ASCII của ký tự cần (đưa ra) hiển thị	Không
	9	Hiển thị chuỗi (đưa chuỗi ký tự tới thiết bị ra chuẩn)	ah=9; ds:dx=trỏ tới chuỗi ký tự kết thúc bằng '$'	Không

Chương 4
GHÉP 8088 VỚI BỘ NHỚ VÀ TỔ CHỨC VÀO RA DỮ LIỆU

4.1. Giới thiệu tín hiệu chân của 8088 và các mạch phụ trợ

4.1.1. Bảy nhóm tín hiệu



Hình vẽ: Các tín hiệu của 8088 ở chế độ Min (và Max)

S6=0 liên tục, S5 phản ánh cờ IF.
S3, S4 cùng phối hợp để chỉ ra việc truy nhập các thanh ghi đoạn

S4	S3	Truy nhập đến các đoạn
0	0	Đoạn dữ liệu phụ (ES:)
0	1	Đoạn ngăn xếp (SS:)
1	0	Đoạn mã hoặc không đoạn nào
1	1	Đoạn dữ liệu

Bảng Các bit trạng thái và việc truy nhập đến các thanh ghi đoạn

(Hình vẽ: Đóng vỏ DIP 40 chân của 8088/86)

- AD7 - AD0 [I/O]: Các chân dồn kênh cho tín hiệu phần thấp của bus địa chỉ và dữ liệu. Khi xung ALE=0 => báo cho mạch ngoài biết trên đường đó (các chân) có tín hiệu dữ liệu (ALE: Address Latch Enable). Khi xung ALE=1 => báo cho mạch ngoài biết trên đường đó (các chân) có tín hiệu địa chỉ. Các chân này ở trạng thái trở kháng cao khi 8088 chấp nhận treo (Hold).
- A15-A8 [O]: Là các bit phần cao của bus địa chỉ. Các chân này ở trạng thái trở kháng cao khi 8088 chấp nhận treo.
- A16/S3, S17/S4, A18/S5, A19/S6 [O]: Là các chân dồn kênh của địa chỉ phần cao của tín hiệu trạng thái. (A: Address, S: Status). Khi ALE=0: Tại các chân này là tín hiệu trạng thái: S6-S3. Khi ALE=1: Tại các chân này là tín hiệu địa chỉ. Các chân này ở trạng thái trở kháng cao khi 8088 chấp nhận treo.

- RD [O]: Tín hiệu điều khiển đọc ("Xung cho phép đọc"). Khi RD=0 thì bus dữ liệu sẵn sàng nhận dữ liệu từ bộ nhớ hoặc thiết bị ngoại vi. Chân RD ở trạng thái trở kháng cao khi 8088 chấp nhận treo.
- READY [I]: Tín hiệu báo cho CPU biết tính trạng (trạng thái) sẵn sàng của thiết bị ngoại vi hoặc của bộ nhớ. Khi READY=1 => CPU thực hiện ghi/đọc mà không cần xen thêm các chu kỳ đợi. Ngược lại khi TBNV hay bộ nhớ có tốc độ chậm => chúng có thể đưa ra tín hiệu READY=0 để báo cho CPU chờ. Khi này CPU tự kéo dài thời gian thực hiện ghi/đọc bằng cách xen thêm các chu kỳ đợi.
- INTR [I]: Đây là chân tiếp nhận tín hiệu yêu cầu ngắt che được. Khi có yêu cầu ngắt tác động đến chân này mà cờ cho phép ngắt IF=1 thì CPU kết thúc lệnh đang thi hành dở (kết thúc chu kỳ lệnh), sau đó nó đi vào chu kỳ chấp nhận ngắt và đưa ra tí hiệu INTA=0 tại chân INTA (24).
- TEST [I]: Tín hiệu tại chân này được kiểm tra bằng lệnh WAIT. Khi CPU thực hiện lệnh WAIT mà khi đó TEST=1 thì CPU sẽ chờ cho đến khi TEST=0 thì mới thực hiện lệnh tiếp theo.
- NMI [I]: Tín hiệu yêu cầu ngắt không che được. Tín hiệu này không bị khống chế bởi cờ IF và nó sẽ được CPU nhận biết tại dườn dương của xung yêu cầu ngắt. Nhận được yêu cầu này CPU kết thúc lệnh đang làm dở, sau đó chuyển sang thực hiện chương trình con phục vụ ngắt INT2 (ISR: Interrupt Service Routine).
- RESET [I]: Tín hiệu Reset lại 8088 (Trong chừng mực nào đó có thể coi tín hiệu này là tín hiệu yêu cầu ngắt không che được). Khi tín hiệu RESET=1 và kéo dài ít nhất 4 chu kỳ đồng hồ (4 xung clock) thì 8088 bị buộc phải khởi động lại, nó xoá các thanh ghi: DS, ES, SS, IP và FR về 0 và bắt đầu thực hiệu chương trình tại địa chỉ CS:IP =FFFF:0000 (như khi khởi động, IF<=0 để cấm các ngắt, TF<=0 để 8088 không bị đặt trong chế độ chạy từng lệnh => chạy suốt).
- CLK [I]: Tín hiệu xung đồng hồ (xung nhịp). Xung nhịp có độ rỗng 77% và cung cấp nhịp làm việc cho CPU (và các mạch khác của hệ thống).
- Vcc [I]: Chân nguồn. Nguồn cung cấp cho CPU là +5V±10%, 340mA.
- GND [O]: 2 chân nguồn nối với 0V của nguồn nuôi.
- MN/MX [I]: Chân điều khiển hoạt động của CPU theo chế độ Min/Max (8088 có thể làm việc ở 2 chế độ khác nhau nên có một số chân tín hiệu phụ thuộc vào chế độ làm việc đó).

Chế độ MIN (chân MN/MX cần được nối thẳng vào +5V mà không qua điện trở)

Khi 8088 ở chế độ Min, tất cả các tín hiệu điều khiển liên quan đến các thiết bị ngoại vi truyền thống và bộ nhớ giống như trong hệ 8085, đều có sẵn bên trong 8088 cho nên việc phối ghép với các thiết bị ngoại vi và bộ nhớ sẽ rất dễ dàng. Vì vậy có thể tận dụng được các phối ghép ngoại vi có sẵn => giảm giá thành hệ thống.

- IO/M [O]: Tín hiệu này phân biệt tại một thời điểm cụ thể nào đó phần tử nào trong các thiết bị vào/ra (I/O) hoặc bộ nhớ (M: Memory) được chọn để trao đổi dữ liệu với CPU. Trên Address bus lúc đó sẽ có các địa chỉ tương ứng thiết bị. Chân này ở trạng thái trở kháng cao khi CPU chấp nhận treo.
- WR [O]: Xung cho phép (giống như RD). Khi CPU đưa ra tín hiệu WR=0 thì dữ liệu đã ổn định và chúng sẽ được ghi vào bộ nhớ hoặc thiết bị vào/ra tại thời điểm chuyển mức WR=1. Chân WR sẽ ở trạng thái trở kháng cao khi 8088 chấp nhận treo.

- INTA [O]: Tín hiệu điều khiển báo cho mạch bên ngoài biết CPU đã chấp nhận yêu cầu ngắt (INTR). Lúc này CPU đưa ra tín hiệu INTA=0 để báo cho TBNV biết nó đang chờ mạch ngoài đưa lên Data bus số hiệu ngắt (kiểu ngắt).
- ALE [O]: Xung cho phép chốt địa chỉ (Address Latch Enable). Khi ALE=1 cso nghĩa trên bus dồn kênh A/D là địa chỉ của thiết bị ngoại vi hay bộ nhớ. ALE không bao giờ bị thả nổi (trạng thái trở kháng cao), khi CPU chấp nhận treo thì ALE=0.
- DT/R [O]: Tín hiệu điều khiển các đệm bus 2 chiều của Data bus để chọn chiều vận chuyển dữ liệu trên bus D. Chân này ở trạng thái trở kháng cao khi 8088 chấp nhạn treo.
- DEN [O]: Tín hiệu báo cho bên ngoài biết khi này trên bus dồn kênh A/D có dữ liệu ổn định. Chân này ở trạng thái trở kháng cao kho 8088 chấp nhận treo.
- HOLD [I]: Tín hiệu yêu cầu treo CPU để mạch ngoài thực hiện trao đổi dữ liệu với bộ nhớ bằng cách truy nhập trực tiếp bộ nhớ (DMA – Direct Memory Access). Khi HOLD=1, 8088 sẽ tự tách khỏi hệ thống bằng cách treo các bus A, D, C của nó (các bus ở trạng thái trở kháng cao) để bộ điều khiển DMA là DMAC (DMA Controler) có thể lấy được quyền điều khiển hệ thống để thực hiện công việc trao đổi dữ liệu.
- HLDA [O]: Tín hiệu báo cho bên ngoài biết yêu cầu treo CPU đã được CPU chấp nhận và CPU đã treo các bus A, D và một số tín hiệu điều khiển trên bus C.
- SS0 [O]: Đây là tín hiệu trạng thái. Tín hiệu này giống như S0 trong chế độ Max và dùng kết hợp với IO/M, DT/R để giải mã các chu kỳ hoạt động của bus.

IO/M	DT/R	SS0	Chu kỳ điều khiển của bus
0	0	0	Đọc mã lệnh
0	0	1	Đọc bộ nhớ
0	1	0	Ghi bộ nhớ
0	1	1	Bus rỗi (nghỉ)
1	0	0	Chấp nhận yêu cầu ngắt
1	0	1	Đọc thiết bị ngoại vi
1	1	0	Ghi thiết bị ngoại vi
1	1	1	Dừng (Halt)

Bảng quan hệ một số tín hiệu điều khiển

Chế độ MAX (chân MN/MX cần được nối thẳng vào 0V)

Khi 8088 làm việc ở chế độ Max, một số tín hiệu điều khiển cần được tạo ra trên cơ sở các tín hiệu trạng thái nhờ dùng thêm ở bên ngoài một mạch điều khiển bus 8288. Chế độ Max được sử dụng trong hệ thống có mặt bộ đồng xử lý toán học 8087.

- S2, S1 và S0 [O]: Là các tín hiệu trạng thái dùng trong chế độ Max để ghép nối với mạch điều khiển bus 8288. Các tín hiệu này được 8288 dùng để tạo ra các tín hiệu điều khiển trong các chu kỳ hoạt động của bus. Các tín hiệu điều khiển đó như trong bảng sau:

S2	S1	S0	Chu kỳ điều khiển của bus	Tín hiệu
0	0	0	Chấp nhận yêu cầu ngắt	INTA
0	0	1	Đọc thiết bị ngoại vi	IORC
0	1	0	Ghi thiết bị ngoại vi	IOWC,AIOWC
0	1	1	Dừng (Halt)	Không
1	0	0	Đọc mã lệnh	MRDC
1	0	1	Đọc bộ nhớ	MRDC
1	1	0	Ghi bộ nhớ	MWTC, AMWC
1	1	1	Bus rỗi (nghỉ)	Không

- RQ/GT0 và RQ/GT1 [O]: Là các tín hiệu yêu cầu dùng bus của các bộ vi xử lý khác trong hệ thống hoặc thông báo chấp nhận treo của CPU để cho phép các bộ vi xử lý khác trong hệ thống dùng bus. RQ/GT0 có các mức ưu tiên cao hơn RQ/GT1.
- LOCK [O]: Tín hiệu do CPU đưa ra để cấm các bộ vi xử lý khác trong hệ thống dùng bus trong khi nó đang thi hành lệnh nào đó đặt sau tiếp đầu LOCK.
- QS0 và QS1 [O]: Tín hiệu thông báo trạng thái khác nhau của đệm lệnh (hàng đợi lệnh). Trong các hệ vi xử lý có mặt bộ đồng xử lý toán học 8087 thì các tín hiệu này dùng để đồng bộ quá trình hoạt động của nó với 8088.

QS1	QS0	Trạng thái đệm lệnh
0	0	Không hoạt động
0	1	Đọc byte mã lệnh đầu tiên từ đệm lệnh
1	0	Đệm lệnh rỗng
1	1	Đọc byte tiếp theo từ đệm lệnh

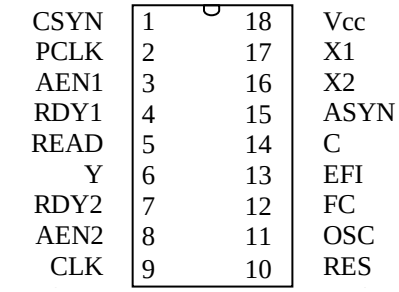
4.1.2. Phân kênh để tách thông tin và đệm bus

Để giảm bớt số chân cho các tín hiệu (khó khăn về công nghệ cũng như khi sử dụng) của CPU, người ta thường thực hiện bằng cách dồn kệnh nhiều tín hiệu trên một chân của CPU (ví dụ như 8 đường dồn kênh của bus A, D phần thấp của 8088). Khi nhận được tín hiệu ở bên ngoài, ta phải tiến hành tách thông tin (dữ liệu hoặc địa chỉ). Việc này được thực hiện bằng các vi mạch chuyên dung có chức năng thích hợp (thường thì đó là các mạch chốt – latch). Để hỗ trợ cho việc tách thông tin, CPU đưa ra tín hiệu ALE sao cho khi ALE=1 (mức cao) => báo cho bên ngoài biết tại các chân dồn kênh là thông tin về địa chỉ và khi ALE=0 => thì tại các chân đó là tín hiệu dữ liệu. Xung ALE được dùng để mở các mạch chốt và tách được các thông tin về địa chỉ bị dồn kênh. Để nâng cao khả năng tải của các bus (chống suy giảm tín hiệu do đảm nhận nhiều việc nuôi các mạch bên ngoài) các tín hiệu vào/ra CPU phải được khuếch đại thông qua các mạch đệm 1 chiều hoặc 2 chiều với các đầu ra: thường hoặc 3 trạng thái.

Các mạch: 74LS373 chốt, 74LS244 khuếch đại đệm 1 chiều, 74LS245 khuếch đại đệm 2 chiều.

4.1.3. Mạch tạo xung nhịp 8284

Dù cho 8088 làm việc ở chế độ Max hay chế độ Min thì nó vẫn luôn cần xung nhịp (xung clock) từ mạch tạo xung nhịp 8284. Mạch tạo xung clock ngoài việc cung

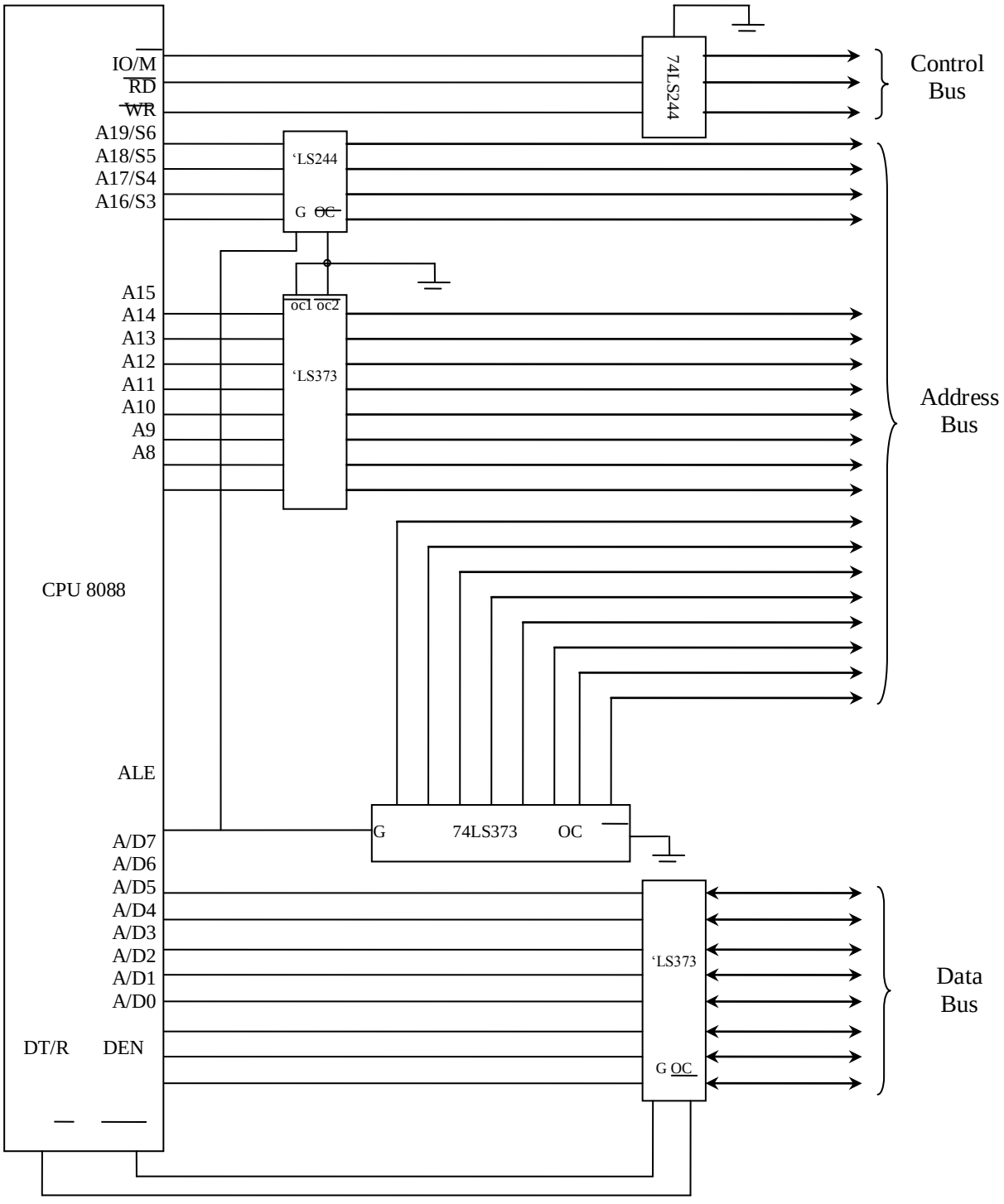


Hình vẽ:Mạch 8284

cấp xung clock cho 8088, nó còn cung cấp xung nhịp có tần số thích hợp cho toàn hệ và nó còn đồng bộ tín hiệu RESET và tín hiệu READY của CPU.

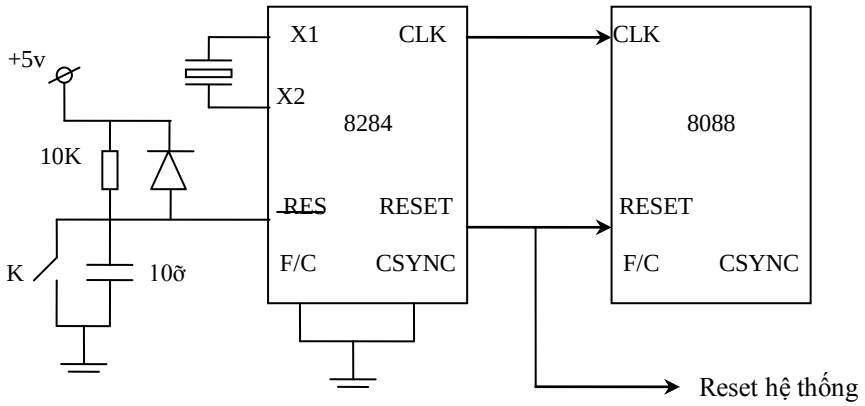
Các tín hiệu

- AEN1, AEN2 (Address ENable) [I]: Tín hiệu cho phép chọn đầu vào tương ứng RDY1, RDY2 làm tín hiệu báo trạng thái sẵn sàng của bộ nhớ hoặc thiết bị ngoại vi.
- RDY1, RDy2 (Bus Ready): Các tín hiệu này cùng với AEN1, AEN2 dùng để tạo ra các chu kỳ đợi (Tw) ở CPU.



Hình vẽ: Lược đồ máy IBM PC/XT

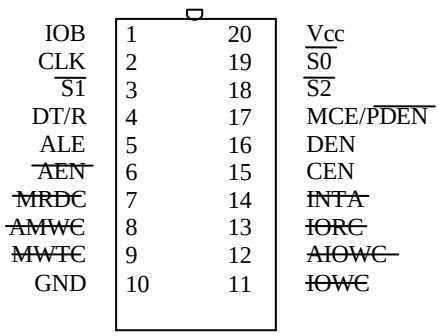
- ASYNC (Ready Synchronisation Setect) [I]: Chọn đồng bộ hai tầng hoặc đồng bộ một tầng cho tín hiệu RDY1, RDY2. Trong chế độ đồng bộ một tầng (ASYNC=1) tín hiệu RDY có ảnh hưởng tới các tín hiệu READY tới tận sườn âm của xung đồng hồ tiếp theo. Còn trong chế độ đồng bộ hai tầng (ASYNC = 0), tín hiệu RDY chỉ có ảnh hưởng đến tín hiệu READY khi có sườn xuống (sườn âm) của xung đồng hồ tiếp theo.
- READY [O]: Nối đến đầu vào READY của 8088. Tín hiệu này được đồng bộ với các tín hiệu RDY1, RDY2.
- X1, X2 (Crystal) [I]: Nối với 2 chân của thạch anh với tần số fx. Thạch anh này là một bộ phận của mạch dao động bên trong 8284 có nhiệm vụ tạo xung chuẩn làm tín hiệu đồng bộ cho toàn bộ hệ thống.
- F/C (Frequency/Crystal) [I]: Dùng để chọn nguồn tín hiệu chuẩn cho 8284. Khi chân này ở mức cao thì xung đồng hồ bên ngoài sẽ được dùng làm xung nhịp cho 8284, ngược lai thì xung đồng hồ của mạch dao động bên trong sẽ được chọn làm xung nhịp.
- EFI (External Frequency Input) [I]: Lối vào cho xung từ bộ dao động ngoài.
- CLK [O]: Xung nhịp, fclk = fx/3, với độ rỗng 77% nối đến chân CLK của 8088.
- PCLK (Peripheral Clock) [O]: Xung nhip fpclk = fx/6, với độ rỗng 50% dành cho thiết bị ngoại vi.
- OSC (OSC Input) [O]: Xung nhịp có tần số fx đã được khuếch đại.
- RES (Reset Input) [I]: Chân khởi động, nối với mạch RC để 8284 có thể tự khởi động khi bật nguồn.
- RESET (Reset Output) [O]: Nối vào Reset của 8088, là tín hiệu khởi động lại (Reset) cho toàn hệ thống.
- CSYNC (Clock Synchronisation) [O]: Lối vào cho xung đồng bộ chung khi trong hệ thống có các 8284 dùng dao động ngoài tai chân EFI. Khi dùng mạch dao động trong thì phải nối đất chân này.



Hình vẽ: 8284 nối với 8088

4.1.4. M
ạch điều khiển bus 8288

Vi mạch 8288 là mạch điều khiển bus, nó nhận một số tín hiệu điều khiển từ CPU (8088) và cung



cấp tất cả các tín hiệu điều khiển cần thiết cho hệ vi xử lý khi CPU 8088 làm việc ở chế độ MAX.

Trong đó có một số tín hiệu mang tên:

AEN:	Address ENable
CEN:	Command ENable
IOB:	Input/Output Bus mode
MRDC:	Memory ReaD Command
MWTC:	Memory WriTe Command
AMWC:	Advanced MWTC
IORC:	Input/Output Read Command
AIOWC:	Advanced IOWC
DT/R:	Data transmit/Receive
DEN:	Data Enable.
MCE/PDEN:	Master Cascade Enable/Peripheral Data Enable

Các tín hiệu của 8288 gồm:

- S2, S1, S0 [I] (Status): Là tín hiệu trạng thái lấy trực tiếp từ CPU. Tuỳ theo các tín hiệu này mà 8288 sẽ tạo ra các tín hiệu điều khiển khác nhau tại các chân ra của nó để điều khiển sự hoạt động của các thiết bị nối với CPU (Bảng tín hiệu trạng thái Si của 8088 ở chế độ MAX, bao gồm cả vào và ra của các tín hiệu).
- CLK [I] (Clock): Nối với xung đồng hồ của hệ thống (từ 8284) và dùng để đồng bộ các xung điều khiển đi ra từ mạch 8284.
- ANE [I] (Address Enable): Đây là tín hiệu vào, sau khoảng thời gian trễ cỡ 150 ms thì sẽ kích hoạt các tín hiệu điều khiển đầu ra của 8288.
- CEN [I] (Command Enable): Tín hiệu vào để cho phép đea ra tín hiệu DEN và các tín hiệu điều khiển khác của 8288.
- IOB [I] (Input/Output bus mode): Tín hiệu để điều khiển mạch 8284 làm việc ở các chế độ bus khác nhau.
  - Khi IOB = 1 thì 8288 làm việc ở chế độ bus vào/ra.
  - Khi IOB = 0 thì 8288 làm việc ở chế độ bus hệ thống.

(Như trong các máy IBM PC)

- MRDC [O] (Memory Read Command): Là tín hiệu điều khiển đoc bộ nhớ, nó sẽ kích hoạt bộ nhớ đưa dữ liệu ra bus.
- MWTC, AMWC [O] (Memory Write Command – Advanced MWTC): Là các tín hiệu ghi bộ nhớ hoặc ghi bộ nhớ kéo dài. Chúng giống như MEMW, nhưng AMWC hoạt động sớm lên một chút để tạo ra khả năng cho các bộ nhớ chậm có thêm thời gian.
- IORC [O] (I/O Read Command): Là các tín hiệu điều khiển đọc thiết bị ngoại vi. Nó kích hoạt các thiết bị ngoại vi được chọn để các thiết bị này đưa dữ liệu ra bus.
- IOWC, AIOWC [O] (I/O Write Command, Advanced IOWC): Là các tín hiệu điều khiển ghi thiết bị ngoại vi hoặc ghi thiết bi ngoại vi kéo dài. Chúng là các tín hiệu giống như IOW, nhưng AIOWC hoạt động sớm lên một chút để cho các thiết bị ngoại vi chậm được kéo dài thêm thời gian ghi.

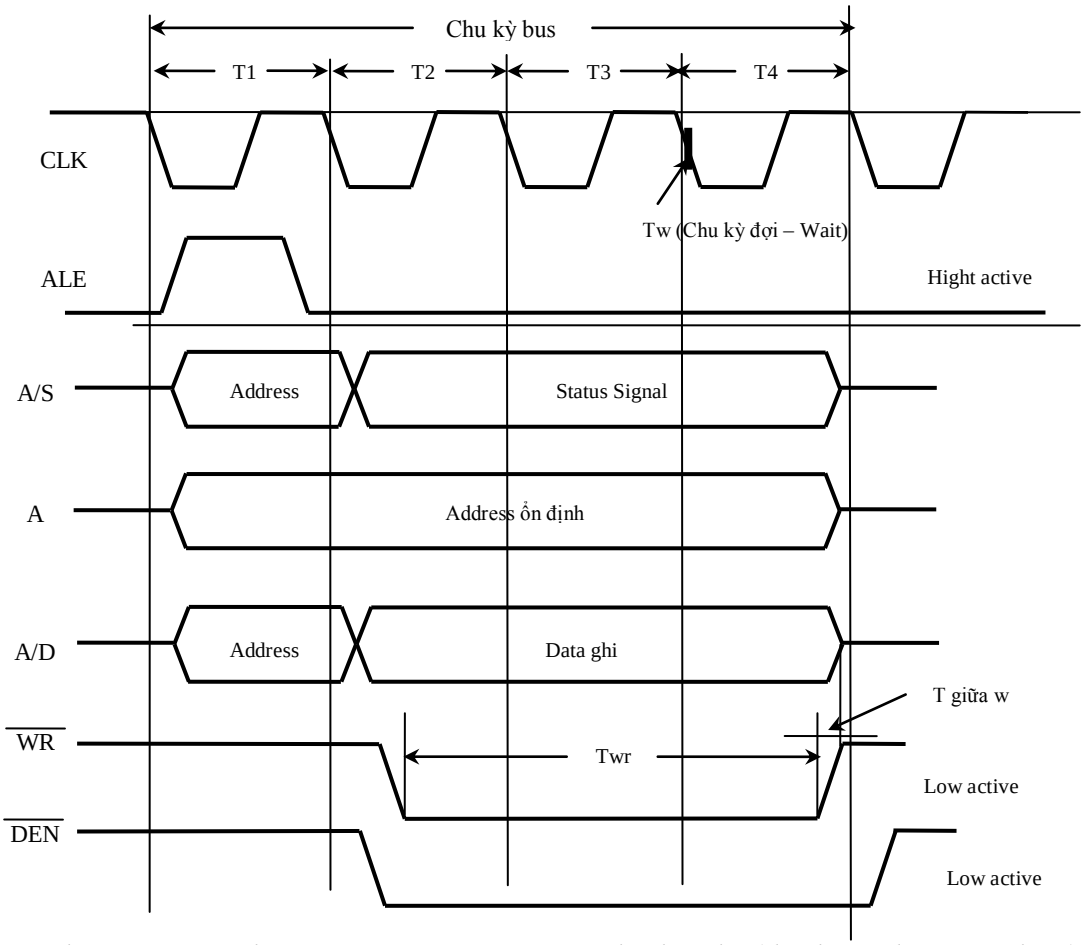
- INTA [O] (Interrupt Acknowledge): Là đầu ra để thông báo cho thiết bị ngoại vi biết là CPU đã chấp nhận yêu cầu ngắt (yêu cầu của thiết bị ngoại vi). Và khi này thiết bị ngoại vi phải đưa số hiệu ngắt ra bus dữ liệu để CPU đọc lấy.
- DT/R [O] (Data Transmit/Receive): Là tín hiệu để điều khiển chiều của dữ liệu trong hệ thống là vào hay ra so với CPU.
  - DT/R = 1: CPU đọc dữ liệu.
  - DT/R = 0: CPU ghi dữ liệu.

  Trong các máy IBM PC thì tín hiệu này được nối đến chân DIR của mạch đệm 2 chiều 74LS245 để điều khiển hướng đi của dữ liệu.
- DEN [O] (Data Enable): Là tín hiệu để điều khiển dữ liệu trở thành bus cục bộ hay bus hệ thống. Trong các máy IBM PC thì tín hiệu này được sử dụng cùng với tín hiệu của mạch điều khiển ngắt PIC 8259 để tạo ra tín hiệu điều khiển cực G của mạch đệm 2 chiều 74LS245.
- MCE/PDEN [O] (Master Cascade Enable/Peripheral Data ENable): Đây là tín hiệu dùng để đặt chế độ làm việc cho mạch điều khiển ngắt PIC 8259 để nó làm việc ở chế độ chủ (Master).
- ALE [O] (Address Latch Enable): Là tín hiệu báo cho phép chốt địa chỉ có tại các chân dồn kênh A/D (A/D0 – A/D7), tín hiệu này thường nối với chân G của mạch chốt 74LS373 để điều khiển mạch mày chốt lấy địa chỉ.

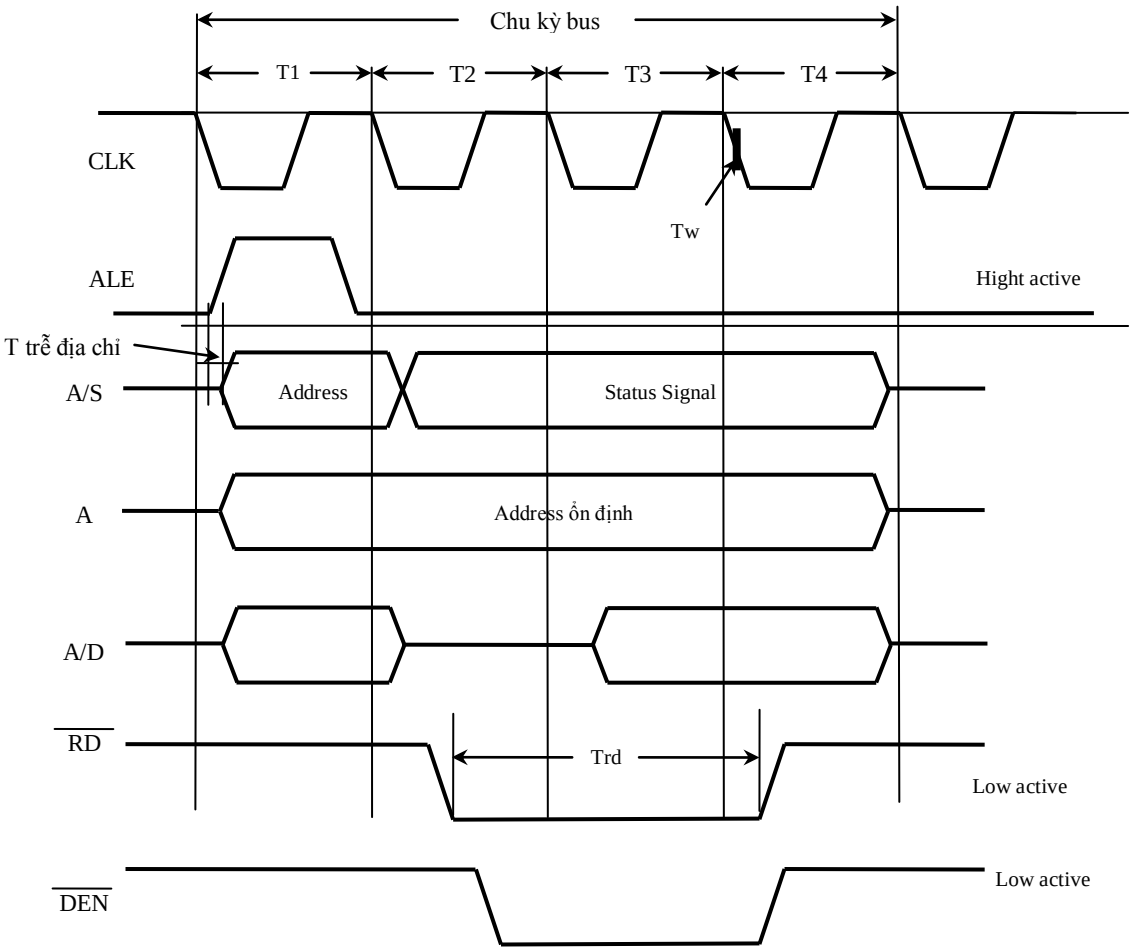
4.1.5. Biểu đồ thời gian của các lệnh đọc/ghi

Chu kỳ bus là một chu kỳ vận chuyển dữ liệu hoàn thành, điều này còn phụ thuộc vào số lượng các tín hiệu điều khiển (của CPU điều khiển hay DMAC, ...). Một chu kỳ bus thường tốn khoảng 4 chu kỳ xung clock. Cụ thể những hiện tượng xảy ra trong một chu kỳ bus (đã được đơn giản hoá) như sau:

- T1: Trong chu kỳ này tín hiệu địa chỉ của bộ nhớ hay thiết bị ngoại vi được đưa ra các chân địa chỉ, các chân dồn kênh A/D, A/S. Các tín hiệu điều khiển như ALE, DT/R, IO/M cũng được đưa ra để giúp cho việc chốt địa chỉ này (20 bit địa chỉ).
- T2: Trong chu kỳ này, CPU 8088 đưa ra các tín hiệu điều khiển RD, hoặc WR, DEN (Data Enable) và các tín hiệu dữ liệu: D7 – D0 (nếu là lệnh ghi). Tín hiệu DEN thường dùng để mở các bộ đệm của bus dữ liệu (nếu các mạch đệm này có mặt trong hệ thống). Tại cuối chu kỳ T2 (và giữa mỗi chu kỳ Ti cả Tw nếu có), CPU lây mẫu tín hiệu READY để xử lý trong các chu kỳ tiếp theo khi nó phải làm việc với bộ nhớ hoặc thiết bị ngoại vi hoạt động chậm.
- T3: Trong chu kỳ T3, CPU dành thời gian cho bộ nhớ hay thiết bị ngoại vi truy nhập dữ liệu. Nếu là chu kỳ đọc dữ liệu thì tại cuối T3, CPU sẽ lấy mẫu tín hiệu của bus dữ liệu. Nếu tại cuối chu kỳ T2 (hoặc giữa mỗi chu kỳ Ti của Tw) mà Cpu phát hiện ra tín hiệu READY = 0 (do bộ nhớ hoặc thiết bị ngoại vi đưa đên) thf CPU tự xen vào sau T3 một vai chu kỳ T để tạo chu kỳ đợik Tw (Tw (wait) = n*T) nhằm kéo dài thời gian thực hiện lệnh, tạo điều kiện cho bộ nhớ hoặc thiết bị ngoại vi chậm có đủ thời gian hoàn thành việc ghi/đọc dữ liệu.
- T4: Trong chu kỳ này các tín hiệu điều khiển được đưa về trạng thái không tích cực để chuẩn bị cho chu kỳ bus mới. Tín hiệu điều khiển WR trong khi chuyển trạng thái từ 0 => 1 sẽ kích hoạt việc ghi dữ liệu vào bộ nhớ hay thiết bị ngoại vi.



Hình vẽ: Các tín hiệu của 8088 trong một chu kỳ ghi (đã được đơn giản hoá)



Hình vẽ: Các tín hiệu của 8088 trong một chu kỳ đọc (đã được đơn giản hoá)

Ví dụ:

Xét với CPU 8088 làm việc ở tần số = 5MHz, mỗi Ti kéo dài 200 ns (nano giây).

Theo hình vẽ chu kỳ đọc bộ nhớ: việc truy nhập bộ nhớ kéo dài trong khoảng từ T1 đến T3 mất khoảng 600 ns.

Ttrễ địa chỉ = 110 ns, Tgiữ R = 30 ns (thời gian giữ của dữ liệu khi đọc)

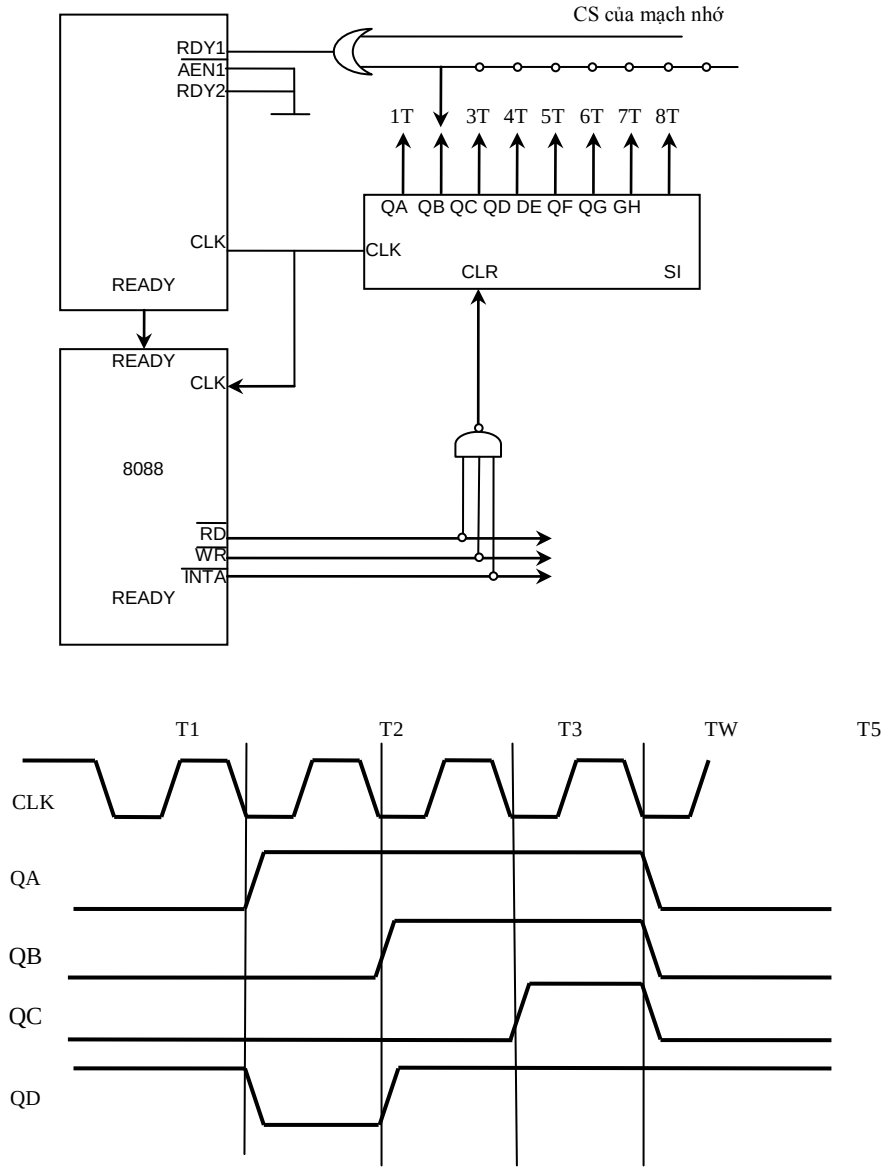
Ttrễ đệm = 40 ns (trễ khi truyền dữ liệu qua các mạch đệm)

==> thời gian thâm nhập = 3*T - Ttrễ địa chỉ - Tgiữ R - Ttrễ đệm = 420 ns

==> Bộ nhớ nối với 8088/5MHz cần có thời gian thâm nhập ≤ 420 ns thì hiệu quả (8088 không phải xen thêm các chu kỳ đợi).

Hơn nữa, với CPU 8088 thì TRD (độ rộng xung đọc) = 325 ns, là khoảng thời gian đủ dài để cho bộ nhớ với thời gian thâm nhập 420 ns.

Trong hình vẽ chu kỳ ghi dữ liệu: Cần có thời gian giữ dữ liệu tối thiểu Tgiữ W = 88 ns sau khi WR chuyển từ 0 --> 1. Thực tế thời gian này xấp xỉ bằng 0 đối với các bộ nhớ thông dụng. Độ dài xung ghi đối với 8088/5 MHz là TWR = 340 ns cũng phù hợp với các bộ nhớ có thời gian thâm nhập cỡ 420 ns.



Hình vẽ: Mạch tạo 0 .. 7 trạng thái đợi và biểu đồ thời gian

4.2. Phối ghép 8088 với bộ nhớ

4.2.1. Bộ nhớ bán dẫn

Các vi mạch nhớ thường dùng với các hệ vi xử lý gồm:

- ROM (Read Only Memory – Bộ nhớ cố định): Bộ nhớ loại này thường có nội dung được ghi săn từ khi sản xuất và chỉ có thể đọc ra nên chúng được gọi là bộ nhớ cố định. Loại này còn được gọi là ROM mặt nạ vì thông tin trong ROM được ghi thông qua một mặt nạ. Khi mất nguồn nuôi cung cấp cho vi mạch thì thông tin vẫn còn (nội dung trong RM không bị mất đi).
  - PROM (Programmable ROM): Loại này ra ROM trắng (chưa ghi thôngo tin) sau khi sản xuất. Người sử dụng có thể ghi thông tin vào ROM theo ý mình một lần duy nhất bằng máy nạp ROM (máy ghi ROM chuyên dụng).
  - EPROM (Erasable ROM): Loại ROM này có thể ghi ("lập trình") bằng xung điện và xoá bằng tia cực tím (UV – Untra Violete), từ một máy nạp ROM.

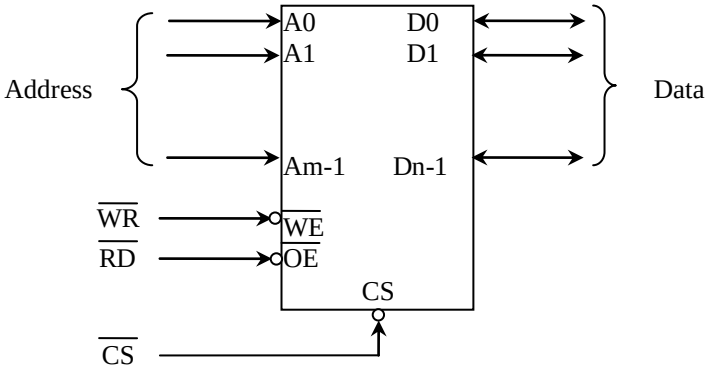
- EEPROM (Electric EPROM): Giống như EPROM, nhưng việc ghi/xoá ROM có thể thực hiện ngay trong mạch làm việc mà không đòi hỏi phải thông qua máy nạp ROM.

- RAM (Random Access Memory – Bộ nhớ ghi/đọc): Đặc trưng của loại bộ nhớ này là thông tin sẽ bị mất đi khi mất nguồn nuôi cấp cho vi mạch.
  - SRAM (Static RAM – RAM tĩnh): Mỗi phần tử nhớ của loại này được cấu tạo từ mạch lật (flip – flop) nên có đặc điểm là tác động nhanh, thông tin ổn định. Kèm theo đó là tốn nhiều transistor cho một đơn vị nhớ nên đắt tiền. Chúng thường được dùng vào những thành phần nhớ quan trọng như thanh ghi, cache, ...
  - DRAM (Dynamic RAM): Mỗi phần tử nhớ của loại này được cấu tạo từ một tụ điện nhỏ, được chế tạo bằng công nghệ PMOS. Loại này có đặc điểm là tác động chậm, thông tin không ổn định vì có sự dò điện tích giữa 2 bản tụ (1: tụ được tích điện, 0: tụ không được tích điện) nên cần các mạch phụ trợ để bù lại lượng điện tích bị dò – gọi là các mạch làm tươi. Loại này rẻ hơn SRAM và có thể sản xuất vi mạch nhớ với dụng lượng lớn.

Một bộ nhớ (hay một modul nhớ nói chung) được cấu tạo (tạo nên) từ nhiều vi mạch nhớ ghép lại. Mỗi vi mạch nhớ thường có cấu tạo như sau:

$A_{m-1} – A_0$: m bit địa chỉ

$D_{n-1} – D_0$: n bit dữ liệu



Dung lượng tính theo bit: 2^m*n (bits).

Nếu n = 8 thì dung lượng là: 2^m (bytes)

WR [I] (WRite): Tín hiệu điều khiển ghi, với ROM thường là WE: Write Enale

RD [I] (ReaD): Tín hiệu điều khiển đọc, với ROM thường là OE: Output Enale

CS [I] (Chip Select): Tín hiệu chọn chip, với ROM thường là CE: Chip Enale

- Nhóm tín hiệu địa chỉ ($A_{m-1} – A_0$): Dùng để chọn ra ô nhớ cụ thể được ghi/đọc. Các ô nhớ có độ dài khác nhau tuỳ từng loại vi mạch nhớ, từng nhà sản xuất: 1, 4, 8, ... bit. Số lượng các đường dây địa chỉ xác định dung lượng của vi mạch nhớ.

  Dung lượng = 2^m (ô nhớ).

  Nếu m = 10 thì dung lượng là: 2^{10} = 1 Kilo ô nhớ (kilo = 1024)

  Nếu m = 20 thì dung lượng là: 2^{20} = 1 Mega ô nhớ (kilo = 1024*1024 = 1048576)

- Nhóm tín hiệu dữ liệu ($D_{n-1} – D_0$): Các đường tín hiệu này là đầu ra đối với vi mạch ROM và là vào/ra đối với vi mạch RAM, cũng có thể đặt riêng nhóm đường vào và ra của các tín hiệu này, khi nối vào bus dữ liệu: là nối chung vì các mạch nhớ thường có đầu ra dữ liệu 3 trạng thái. Số đường dữ liệu (n) xác định độ dài của ô nhớ (ngăn nhớ – từ nhớ). Thường ghi rõ 1Kx8 hoặc 16Kx4 hoặc 1Kx1.

- Tín hiệu chọn vi mạch: Người ta thường dùng ký hiệu CS (Chip Select) cho các vi mạch RAM và CE (Chip Enable) cho các vi mạch ROM. Tín hiệu này dùng để chọn ra (cho phép) vi mạch nhớ nào trong bộ nhớ làm việc thì mới tiến hành ghi/đọc ô nhớ của vi mạch đó. Tín hiệu này thường được nối với đầu ra của mạch giải mã địa chỉ. Khi một số vi mạch nhớ có tín hiệu này ở trạng thái không tích cực thì bus dữ liệu của nó (Am-1 – A0) bị treo (trạng thái trở kháng cao).
- Nhóm tín hiệu điều khiển: Các tín hiệu điều khiển đọc RD và WR (OE và WE đối với ROM) luôn ngược pha nhau để cho phép dữ liệu từ bus dữ liệu được đi vào bi mạch hoặc ngược lại từ vi mạch nhớ ra bus dữ liệu. Cũng có những RAM chỉ có 1 đường tín hiệu điều khiển cho cả ghi và đọc: R/W. Khi này: R/W=1: đọc, R/W=1: ghi.

Một đặc trưng quan trọng của vi mạch nhớ là thời gian thâm nhập tac (taccess). Có thể định nghĩa rằng thời gian thâm nhập của một vi mạch nhớ là khoảng thời gian kể từ khi có (xung) địa chỉ trên bus địa chỉ cho đến khi dữ liệu ra ổn định trên bus dữ liệu. Thông số này phụ thuộc vào công nghệ chế tạo vi mạch nhớ.

Với công nghệ lưỡng cực, có thể sản xuất các vi mạch nhớ có tac = 10 – 30 ns.

Với công nghệ MOS, có thể sản xuất các vi mạch nhớ có tac lớn hơn, cỡ 150 ns hoặc hơn nữa.

Bộ nhớ EPROM

Các vi mạch EPROM thông dụng tồn tại dưới nhiều kiểu khác nhau. Hệ 27xxx có một số loại sau:

2708 (1Kx8); 27256 (32Kx8);

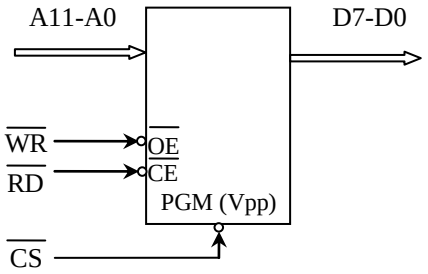
2732 (2Kx8); 27512 (64Kx8);

2764 (8Kx8); 27128 (16Kx8);

Với tac = 250 – 450 ns tùy từng loại cụ thể.

Ví dụ: Xét vi mạch nhớ EPROM 2716 có tac = 450 ns. Vậy để ghép với bộ vi xử lý 8088 – 5 MHz cần có thêm các chu kỳ đợi (Tw). Ngược lại mạch 2716-1 lại có tac = 250 ns nên không cần có thêm các chu kỳ đợi.

Chú ý: Trong chế độ duy trì công suất tiêu thụ của mạch giảm được tới 75% so với công suất tiêu thụ khi nó ở chế độ tích cực.



Bộ nhớ EPROM 2716 (2Kx8)

x: Don't care

Chân (pins) / Chế độ (Mode)	CE/PGM	OE	Vpp [V]	Vcc [V]	D7–D0
Đọc	0	0	+5	+5	Dout
Duy trì	1	x	+5	+5	HZ
Ghi	Xung ghi kéo dài 50 ns	1	+25	+5	Din
Kiểm tra ghi	0	0	+25	+5	Dout
Cấm ghi	0	1	+25	+5	HZ

HZ: Trạng thái trở kháng cao.

A10-A0: 11 bit địa chỉ.

D7-D0: 8 bit dữ liệu.

OE (Output Enable): Cho phép đưa dữ liệu ra.

CE/PGM: Chọn chip/điều khiển ghi.

Vpp: Điện áp ghi.

- **Bộ nhớ SRAM (RAM tĩnh)**

Bộ nhớ được cấu tạo từ các phần tử nhớ là các mạch lật (flip – flop) nên nó sẽ lưu giữ thông tin một cách ổn định chừng nào nó còn được cấp điện (nguồn nuôi). Các bộ nhớ SRAM và các bộ nhớ EPROM có dung lượng bằng nhau thường có cách tổ chức (bố trí) các chân giống nhau để dễ dàng thay thế cho nhau trong quá trình phát triển hệ thống.

Trong thực tế đã tồn tại các mạch nhớ SRAM với dung lượng:

32Kx8 (62256 LP – 10), tac = 100ns, công nghệ CMOS.

SRAM được chế tạo theo công nghệ lưỡng cực (8KB – 128 KB) có thời gian trao đổi dữ liệu (thâm nhập): tac = 15 ns.

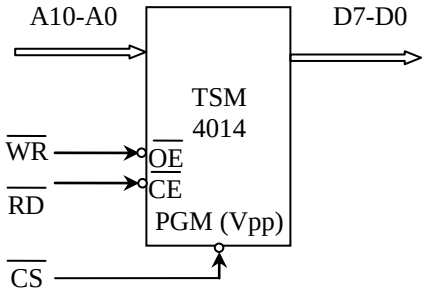
Ví dụ: Xét vi mạch nhớ SRAM TMS 4014 (2Kx8), tac = 250 ns.

A10-A0 : 11 bit địa chỉ

D7-D0 : 8 bit dữ liệu

OE/WE : Cho phép đọc/cho phép ghi.

CS : Chọn chip.

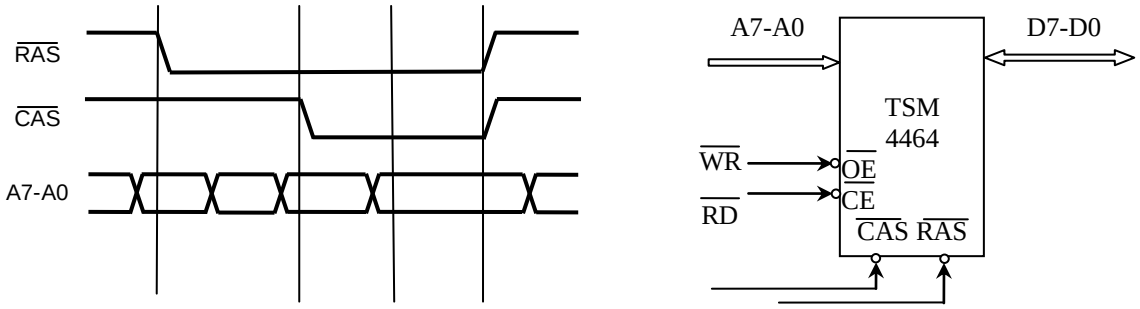


- **Bộ nhớ DRAM (RAM động)**

Bộ nhớ DRAM được cấu tạo từ các phần tử nhớ là các tụ điện. Nó lưu trữ thông tin bằng cách nạp tụ (1) hay không nạp (0) điên tích trên tụ. Vì có sự dò điện tích trên 2 bản cực tụ điện mà thông tin lưu trữ trong bộ nhớ loại này không ổn định. Vì vậy DRAM cần được làm tươi (refresh) bằng cách ghi hay đọc lại DRAM theo chu kỳ sau mỗi quãng thời gian khoảng 15,6 μs. Các mạch nhớ SRAM cần có thêm các mạch logic phụ để đảm bảo điều khiển việc làm tươi nên việc phối ghép nó với bộ vi xử lý là rất phức tạp. Nhưng DRAM có ưu điểm là có thể chế tạo được một số lượng rất lớn các phần tử nhớ trên một đơn vị diện tích (vì mỗi phần tử nhớ cần rất ít transistor – 1). Và vì vậy các vi mạch nhớ này cũng cần rất nhiều chân địa chỉ. Để giảm bớt số chân địa chỉ trên vi mạch nhớ (để dễ chế tạo mạch in và lắp ráp) người ta thường chia địa chỉ thành 2 nhóm là địa chỉ hàng và địa chỉ cột và dồn kênh chúng trên các chân địa chỉ. Vì vậy các phần tử nhớ của bộ nhớ được tổ chức theo ma trận mà mỗi ô nhớ là một phần tử của ma trận nhớ. Việc dồn kênh địa chỉ trên các chân địa chỉ cần thêm các tín hiệu báo thời điểm nào là địa chỉ hàng và thời điểm nào là địa chỉ cột trên các chân địa chỉ, các tín hiệu đó là:

RAS (Row Access) : Cho phép chốt địa chỉ hàng.

CAS (Column Access): Cho phép chốt địa chỉ cột.



Hình vẽ: DRAM TMS 4464 (64Kx8)

Các mạch nhớ DRAM thường được chế tạo với độ dài 1 hoặc 4 bit trên một ngăn nhớ. Thực tế đã có các mạch nhớ 1Mx1, 4Mx1, 16Mx1 và chúng được xây dựng thành các modul nhớ kiểu SIMM (Single Inline Memory Modul) hay SIP (Single Inline Package) dùng trong các máy 80268, 80386, ...

4.2.2. Giải mã địa chỉ cho bộ nhớ

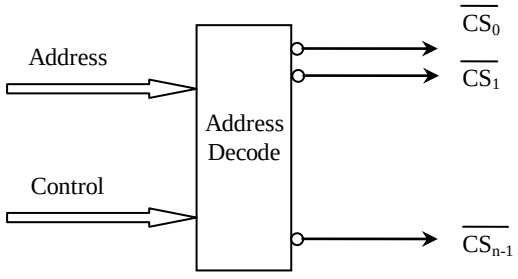
Mỗi vi mạch nhớ hay một modul nhớ khi ghép với bộ vi xử lý cần phải được bộ vi xử lý tham chiếu tới một cách chính xác khi thực hiện các thao tác ghi/đọc. Có nghĩa là mỗi "modul" nhớ cần được gán cho một vùng không gian riêng biệt cụ thể (trong không gian quản lý chung của bọ vi xử lý), có địa chỉ xác định. Việc gán địa chỉ cụ thể cho modul nhớ được thực hiện nhờ một xung chọn chip từ mạch giải mã địa chỉ. Việc phân chia không gian tổng thể của bộ vi xử lý thành các vùng khác nhau dành cho các mục đích khác nhau gọi là phân vùng bộ nhớ.

Ví dụ:

Với 8088 vì có 20 bit địa chỉ ($A_{19} - A_0$) nên không gian tổng thể dành cho bộ nhớ là $2^{20} = 2^{10}*2^{10} = 1$ MB (1 byte/ngăn nhớ). Trong không gian tổng thể đó:

1) Vùng không gian 1 KB đầu, kể từ địa chỉ thấp nhất: 00000h – là phải được dành cho RAM vì tại đây phải có chỗ dành cho bảng vector ngắt gồm 256 (ngắt)*4 (byte)= 1KB).
2) Vùng nhớ có địa chỉ FFFF0h thì nhất thiết phải được dành cho ROM hoặc EPROM – chứa chương trình khởi động hệ thống. Vì sau khi bật nguồn, CS sẽ mạng giá trị FFFF0h (là địa chỉ khởi động của CPU).

Bộ giải mã địa chỉ để tạo ra xung chọn chip thường có cấu tạo như hình vẽ bên:



Hình vẽ: Sơ đồ khối giải mã địa chỉ cho modul nhớ

Đầu vào bộ (mạch) giải mã là các tín hiệu địa chỉ và các tín hiệu điều khiển khác nữa (nếu cần thiết). Các tín hiệu địa chỉ gồm các bit địa chỉ có quan hệ nhất định với các tín hiệu chọn chip ở đầu ra. Tín hiệu điều khiển thường là tín hiệu IO/M dùng để phân biệt đối tượng mà bộ vi xử lý chọn làm việc là bộ nhớ hay thiết bị vào/ra (I/O device). Mạch giải mã là khâu gây ra việc trễ thời gian từ bộ vi xử lý đên thiết bvị ngoại vi mà trong khi chọn mạch nhớ/thiết bị ngoại vi ta cần tính đến. Tuỳ theo quy mô (yêu cầu) của mạch giải mã mà ở đầu ra ta có thể có nhiều tín hiệu chọn chip (CS).

Giải mã đầy đủ cho một modul nhớ yêu cầu ta phải đưa đến đầu vào của mạch giải mã các tín hiệu địa chỉ sao cho tín hiệu đầu ra của nó chỉ chọn riêng mạch nhớ đã định trước.

Trong trường hợp này ta phải dùng tổ hợp đầy đủ cả các đầu và địa chỉ tương ứng để chọn mạch nhớ. Nếu ta bỏ bớt đi một tín bit địa chỉ nào đó thì đó là việc giải mã thiếu cho modul nhớ, vì xung chọn chip ở đầu ra mạch giải mã ngoài việc chọn mạch nhớ ở vùng đã định sẽ có thể chọn ra các mạch nhớ ở vùng khác nữa. Vậy, việc giải mã thiếu thì có thể tiết kiệm được linh kiện khi xây dựng mạch giải mã nhưng lại không đảm bảo tính đơn trị theo ý nghĩa của việc giải mã.

Trong thực tế, thông thường khi thiết kế mạch giải mã địa chỉ cho modul nhớ người ta thường tính dôi ra một chút để dự phòng sự phát triển tăng thêm dung lượng của bộ nhớ mà vẫn có thể tận dụng (sử dụng) được mạch giải mã sẵn có.

- **Xây dựng mạch giải mã bằng các mạch NAND (các mạch logic đơn giản)**

Bằng các gates logic kiểu NAND, ta có thể xây dựng được mạch giải mã địa chỉ đơn giản với số lượng tín hiệu chọn chip ở đầu ra là hạn chế. Ta phải đưa đến đầu vào của mạch cửa NAND nhiều lối vào một tổ hợp thích hợp của các bit địa chỉ để nhận được ở đầu ra của nó tín hiệu chọn chip cho modul nhớ.

Ví dụ: Cho modul nhớ SRAM có dung lượng 32 KB (32Kx8). Xây dựng mạch giải mã để ghép modul trên với 8088 tại địa chỉ 08000h.

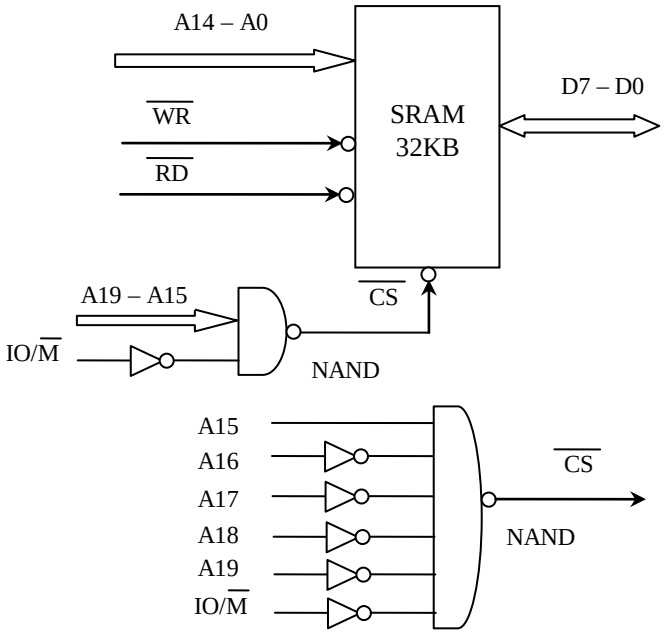
Chú ý: Một địa chỉ bắt đầu cho một modul nhớ cần đảm bảo sao cho có giá trị các bit địa chỉ đưa vào modul nhớ để chọn ra ô nhớ cần tham chiếu phải bằng 0, chúng sẽ "chạy" lần lượt tới 1 để chọn hết đủ tất cả các ô nhớ.

Giải:

Modul nhớ 32 KB cần có 15 bit địa chỉ để chọn các ô nhớ.

Không gian địa chỉ của 8088 gồm 20 bit địa chỉ: A19 – A0.

Nên phải sử dụng các bit địa chỉ thấp: A14 – A0 để đưa vào modul nhớ, còn lại 5 bit địa chỉ phần cao và các tín hiệu điều khiển IO/M sẽ đi vào mạch giải mã. Vậy ta có sơ đồ như sau:



Triển khai địa chỉ của modul nhớ ta có:

0000 1000 0000 0000 0000=08000h

0000 1000 0000 0000 0001=08001h

…

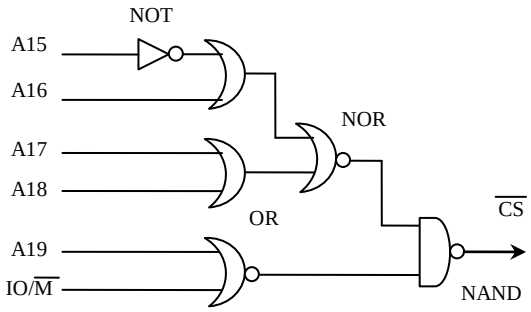
0000 1111 1111 1111 1111=0FFFFh

Phần cố định Phần thay đổi

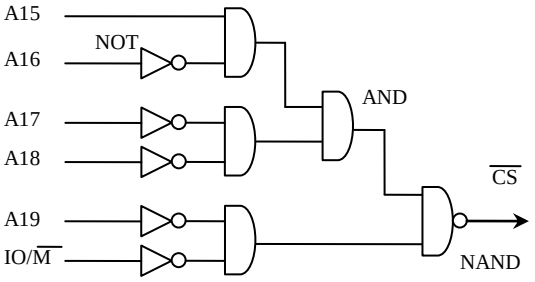
Các tín hiệu điều khiển vào mạch NAND nhiều đầu vào phải đồng thời bằng 1 thì đầu ra CS = 0.

Vậy ta có sơ đồ như bên:

Hoặc ta có thể sử dụng các mạch logic đơn giản khác để xây dựng mạch giải mã sau cho có nhiều đầu ra CS = 0 như sau:



<u>*Kết luận:*</u> Để thực hiện mạch giải mã kiểu này, có nhiều cách sắp xếp các phần tử logic (gates) để đưa ra được tín hiệu CS = 0 ở đầu ra. Tuy nhiên người thiết kế cần phân tích hàm đầu vào và tín hiệu đầu ra và tối giản các phần tử để sao cho số lượng các gates logic sử dụng trong mạch giải mã là ít nhất (tối ưu mạch giải mã). Ví dụ, mạch giải mã trên có thể thay thế bằng mạch giải mã bên:



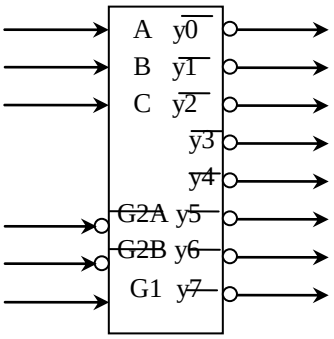
Mạch này cũng thực hiện đúng yêu cầu như mạch giải mã trên, tuy nhiên tốn nhiều phần tử gates logic hơn nên chưa tốt.

<u>*Giải thích:*</u> Trong mạch giải mã cho modul nhớ này, xung chọn chip (CS) sẽ có tác động (CS = 0 – tích cực thấp) khi ta truy xuất bộ nhớ (ghi/đọc) tại địa chỉ nằm trong phạm vi 08000h – 0FFFFh, 5 bit địa chỉ phần cao (A19 – A15) phối hợp cùng tín hiệu điều khiển IO/M (= 0 để chỉ ra sẵn sàng truy xuất bộ nhớ) để tạo ra xung chọn chip cho modul nhớ. Mỗi ô nhớ cụ thể trong 32 KB của modul nhớ sẽ do các bit thấp còn lại (A14 – A0) của bus địa chỉ chọn ra. Để kiểm chứng nhanh điều này ta thấy bit địa chỉ A15 để chọn ra vùng nhớ 32 KB, bit A16 để chọn ra vùng nhớ 64 KB, ... các vùng nhớ này nằm rải rác nhau trong không gian nhớ 1 MB.

▯ Xây dựng mạch giải mã bằng mạch giải mã chuyên dụng 74LS138

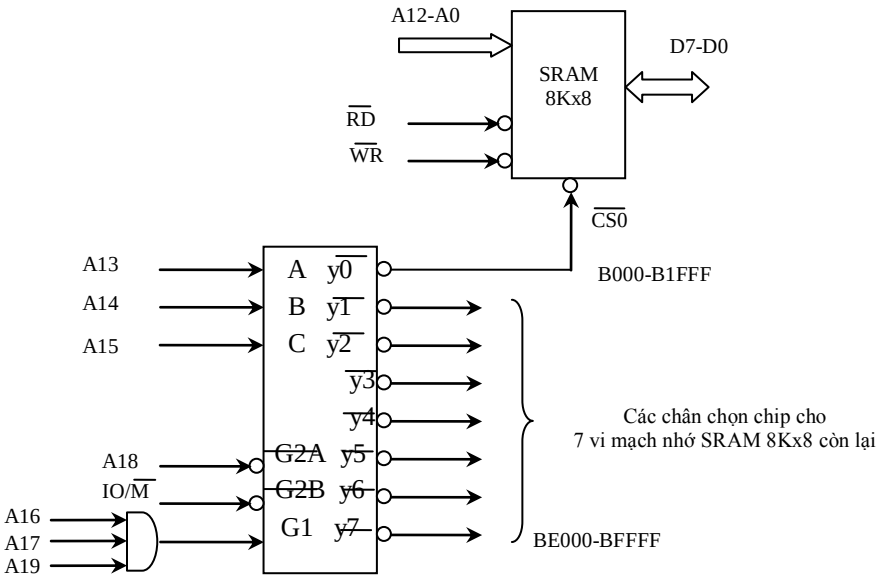
Khi ta muốn có nhiều đầu ta chọn chip (CS) ở đầu ra mạch giải mã mà vẫn dùng các mạch logic đơn giản thì thiết kế sẽ trở nên rất cồng kềnh do số lượng các gates tăng lên. Trong trường hợp đó người ta thường sử dụng mạch giải mã chuyên dụng có sẵn. Một trong các mạch giải mã kiểu đó hay được sử dụng là 74LS138. Đây là mạch giải mã 3 – 8 (vào 3, ra 8).

Sơ đồ khối của 74LS138 và bảng trạng thái



Các đầu vào						Các đầu ra							
chọn			Cho phép										
C	B	A	G2B	G2A	G1	y0	y1	y2	y3	y4	y5	y6	y7
x	x	x	1	x	x	1	1	1	1	1	1	1	1
x	x	x	x	1	x	1	1	1	1	1	1	1	1
x	x	x	x	x	0	1	1	1	1	1	1	1	1
0	0	0	0	0	1	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	1	0	0	0	1	1	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1	1	0	1	1	1	1
1	0	0	0	0	1	1	1	1	1	0	1	1	1
1	0	1	0	0	1	1	1	1	1	1	0	1	1
1	1	0	0	0	1	1	1	1	1	1	1	0	1
1	1	1	0	0	1	1	1	1	1	1	1	1	0

Ví dụ: Xây dựng mạch giải mã cho một vùng nhớ 64 KB bắt đầu từ địa chỉ B0000h đến BFFFFh (vùng này có chứa vùng RAM màn hình) cho các vi mạch nhớ SRAM 8Kx8. Vậy cần 8 vi mạch đã cho ghép lại.



Hình vẽ: Mạch giải mã cho modul nhớ 64 KB dùng 74LS138

Giải: Modul nhớ 64 KB (gồm 8 vi mạch SRAM 8Kx8 ghép lại) cần 16 đường địa chỉ để chọn ra các ô nhớ riêng trong đó, còn 4 đường địa chỉ sẽ được tổ hợp cùng tín hiệu điều khiển IO/M để chọn cho vi mạch 74LS138. Sơ đồ như trên

Trong ví dụ này ta thấy bit địa chỉ A13 có thể chọn ra các vùng nhớ 8K (bằng dung lượng vi mạch) nằm rải rác trong không gian nhớ của 8088. Vì vậy ta có thể dùng nó như đầu vào chọn A của 74LS138, cùng với các bit địa chỉ A14, A15 tại các chân B và C ta sẽ chọn ta được 8 vùng nhớ liền nhau (64 KB = 8x8KB). Vấn đề còn lại định vị vùng 64 KB này vào địa chỉ B0000h của không gian nhớ 1 MB. Điều này có thể thực hiện một cách dễ dàng bằng cách dùng tổ hợp 4 bit địa chỉ phần cao còn lại: A19...A16 = 1011. Thấy A18 = 0, đưa vào đầu vào cho phép G2A, tín hiệu IO/M ở mức thấp ==> đưa vào G2B, còn lại A16, A17, A19 = 1 ==> đưa qua mạch AND 3 đầu vào để có đầu ra = 1 duy nhất gửi đến G1 của 74LS138.

- **Xây dựng mạch giải mã bằng PROM**

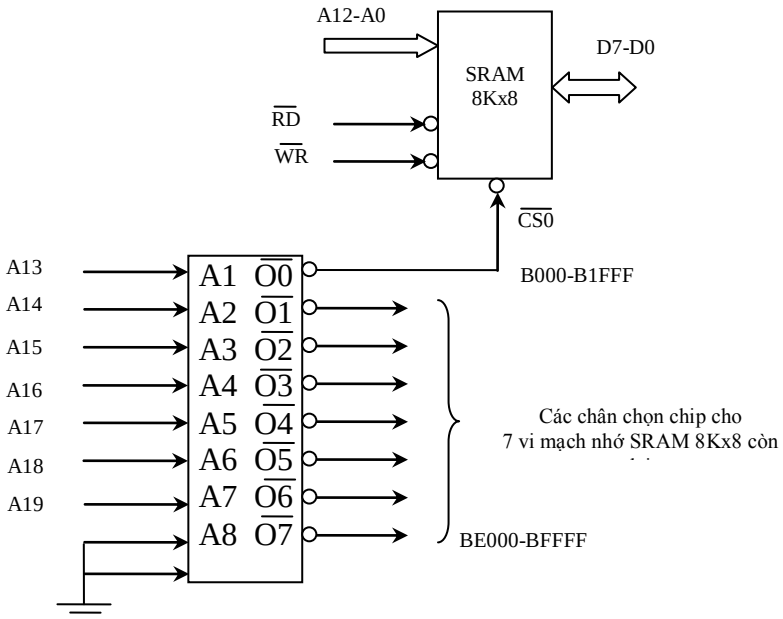
Trong ví dụ trước ta thấy mạch giải mã chuyên dụng sẵn có 74LS138 có số lượng đầu vào địa chỉ và đầu vào cho phép là hạn chế. Nếu ta có số lượng đầu vào cho địa chỉ là lớn mà lại phải giải mã đầy đủ thì để xây dựng bộ giải mã hoàn chỉnh ta vẫn phải dùng thêm các gates logic phụ. Đó chính là lý do để người ta thay thế các bộ giải mã kiểu này bằng các bộ giải mã bằng PROM hoặc PLA (Programmable Logic Array – mảng logic lập trình được), với ưu điểm là chúng có rất nhiều các bit đầu vào cho địa chỉ và vì vậy rất thích hợp trong các hệ vi xử lý có bộ vi xử lý tiên tiến – có không gian địa chỉ rất lớn.

Ta lấy lại ví dụ trong phần trước, có nghĩa là xây dựng mạch giải mã dùng PROM cho vùng nhớ 64 KB (8KBx8). Ta sử dụng loại PROM TPB28L42 với dung lượng 512 byte để làm bộ giải mã. Ta có bảng mẫu các bit để ghi vào PROM TPB28L42 cho ví dụ cụ thể này như sau:

Các đầu vào										Các đầu ra							
G	A8	A7	A6	A5	A4	A3	A2	A1	A0	O0	O1	O2	O3	O4	O5	O6	O7
0	0	0	1	0	1	1	0	0	0	0	1	1	1	1	1	1	1
0	0	0	1	0	1	1	0	0	1	1	0	1	1	1	1	1	1
0	0	0	1	0	1	1	0	1	0	1	1	0	1	1	1	1	1
0	0	0	1	0	1	1	0	1	1	1	1	1	0	1	1	1	1
0	0	0	1	0	1	1	1	0	0	1	1	1	1	0	1	1	1
0	0	0	1	0	1	1	1	0	1	1	1	1	1	1	0	1	1
0	0	0	1	0	1	1	1	1	0	1	1	1	1	1	1	0	1
0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	Các địa chỉ khác									1	1	1	1	1	1	1	1

Bảng mẫu các bit ghi PROM theo yêu cầu ví dụ

Theo bảng mẫu các bit ta thấy để thực hiện giải mã cho bộ nhớ theo yêu cầu trong ví dụ trước, ta mới chỉ dùng hết 8 byte đầu tiên trong tổng số 512 byte của PROM TPB28L42 (tương ứng để chọn ra 8 vùng nhớ 8KB). Các ô nhớ còn lại của PROM vì thế mà đều chứa cùng một giá trị như nhau là FFh (không nên là 00h để khỏi xảy ra chọn nhầm một vùng nhớ (1 địa chỉ) nào đó).



Hình vẽ: Mạch giải mã cho modul nhớ 64 KB dùng PROM

So với cách thực hiện dùng 74LS138, ta thấy không phải dùng đến các gates logic phụ, điều đó làm giảm đáng kể kích thước của bộ giải mã.

Trong các máy vi tính cá nhân thương phẩm ngày nay, việc phân vùng và giải mã địa chỉ cho bộ nhớ đã được thực hiện hoàn chỉnh. Việc nâng cấp, bổ xung các modul (DRAM) vào hệ thống cũng được các nhà sản xuất tính toán từ trước để tạo thuận lợi tối đa cho người sử dụng. Ngày nay khi mở vỏ hộp máy tính (case), trên mainboard ta thường thấy 2-3 khe cắm DRAM có sẵn, cho phép ta bổ xung hoặc sử dụng bất kỳ khe cắm nào trong số đó vì trên mainboard đã có sẵn mạch giải mã cho các modul RAM cắm vào.

4.2.3. Phối ghép 8088 với bộ nhớ

Sau khi đã biết một số phương pháp giải mã cho bộ nhớ, ta sẽ thực hiện phối ghép 8088 – 5 MHz với bộ nhớ. Có thể nói rằng nếu không có sự chênh lệch về tốc độ thâm nhập bộ nhớ và tốc độ của CPU thì việc phối ghép CPU với bộ nhớ chỉ đơn giản là giải mã cho mạch nhớ. Phần lớn các trường hợp, điều này có thể đúng với các mạch nhớ RAM và các mạch nhớ EPROPM có thời gian thâm nhập ≤ 250 ns, cách phối ghép CPU với các mạch này về cơ bản là giống nhau. Đối với các mạch nhớ ROM và EPROM có thời gian thâm nhập lớn (ví dụ như: 2716, 2732, ... với tac = 450 ns) thì khi phối ghép với CPU 8088 – 5MHz ta cần tính toán thận trọng hơn.

Trong phần đầu chương, tại ví dụ khi xét CPU 8088 – 5MHz thì mỗi Ti kéo dài 200 ns. Theo biểu đồ thời gian đọc bộ nhớ (đã đơn giản hoá) thì việc đọc bộ nhớ kéo dài từ T1 đến T3 mất 600 ns.

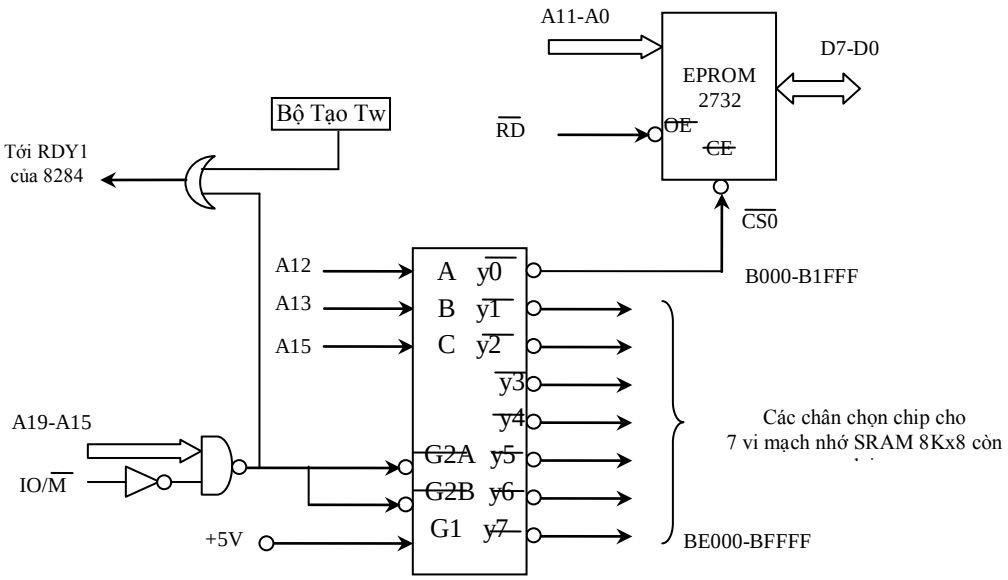
Ttrễ địa chỉ = 110 ns, Tgiữ R = 30 ns (thời gian giữ của dữ liệu khi đọc)

Ttrễ đệm = 40 ns (trễ khi truyền dữ liệu qua các mạch đệm)

*==> thời gian thâm nhập = 3*T - Ttrễ địa chỉ - Tgiữ R - Ttrễ đệm = 420 ns*

Và kết luận: Bộ nhớ ghép với 8088 – 5MHz cần có thời gian thâm nhậơ bộ nhớ (tac) ≤ 420 ns thì hiệu quả, ngược lại nếu tac ≥ 420 ns thì 8088 phải xen thêm các chu kỳ đợi (Tw).

Vậy nếu muốn ghép EPROM 2732 có tac = 450 ns vào không gian nhớ của CPU 8088 – 5MHz thì phải có cách báo cho CPU xen thêm chu kỳ đợi Tw. Sau đây là sơ đồ mạch phối ghép EPROM 2732 có thêm mạch NAND để tạo tín hiệu cho phép mạch giải mã và tín hiệu yêu cầu đợi để đưa đến chân RDY1 của 8284. Như vậy, mỗi khi 8088 đọc EPROM 2832 thì một chu kỳ đợi sẽ được xen thêm.



Hình vẽ: Sơ đồ ghép nối EPOM 2732 với 8088

Việc đặt modul nhớ này vào địa chỉ nào trong không của 8088 là do việc thực hiện mạch giải mã địa chỉ để tạo ra tín hiệu CS. Nó được quyết định bởi các bit từ A19 – A15 và tín hiệu điều khiển IO/M.

Việc phối ghép SRAM với 8088 thường đơn giản hơn so với ROM (nói chung) có tốc độ thâm nhập thấp (tac lớn) vì SRAM có tac nhỏ nên không cần có mạch xen thêm các chu kỳ đợi Tw.

Kiểm tra Parity để phát hiện lỗi trong bộ nhớ RAM

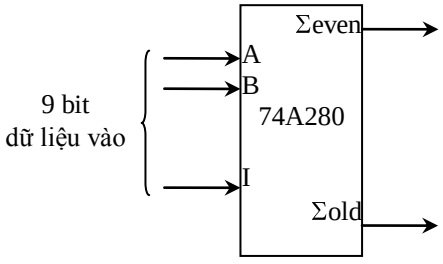
Bộ nhớ bán dẫn DRAM dùng cho các máy tính cá nhân ngày nay có giá ngày càng giảm nên người ta đưa vào sử dụng các bộ nhớ DRAM với dung lượng ngày càng lớn và vì vậy việc kiểm tra parity để phát hiện lỗi trong thiết bị nhớ là hết sức cần thiết. Cụ thể, người ta dùng thêm 1 bit (bit thứ 9) để làm bit kiểm tra parity cho 8 bit dữ liệu – người ta so sánh bit parity khi ghi và khi đọc để phát hiện lỗi. Thực tế có thể dùng một trong hai kiểu parity là parity chẵn (even parity) và parity lẻ (odd parity).

Parity chẵn: Khi tổng số bit 1 trong 8 bit dữ liệu là chẵn thì ghi vào bit thứ 9 số 0, ngược lại thì ghi 1 (bit parity = xor (8 bit dữ liệu)).

Parity lẻ: Khi tổng số bit 1 trong 8 bit dữ liệu là lẻ thfi ghi vào bit thứ 9 số 0, ngược lại thì ghi 1 (bit parity = xor (8 bit dữ liệu) xor1).

Mặc dù công việc trên có thể thực hiện bằng chương trình tuy nhiên nếu thực hiện bằng phần cứng thì cho kết quả nhanh hơn rất nhiều. Trong thực tế, người ta đã chế tạo ra các vi mạch chuyên dụng thực hiện công việc trên khi kiểm tra parity nhằm xác định lỗi của bộ nhớ khi hoạt động (ghi/đọc). Một trong những vi mạch loại này là mạch tạo parity và phát hiện lỗi 74AS280.

Số bit 1 trong các đầu vào: A-I	Đầu ra	
	Σeven	Σold
0, 2, 4, 6, 8	1	0
1, 3, 5, 7, 9	0	1



Σeven = xor(A-I); Σold = (xor(A-I))xor1

Hình vẽ: Sơ đồ khối và bảng trạng thái của 74AS280

Dưới đây là sơ đồ tổng thể của một bộ nhớ SRAM trong thực tế có sử dụng vi mạch 74AS280 để tạo và kiểm tra parity. Nếu có sai parity do có lỗi tại bộ nhớ trong khi ghi/đọc thì lỗi này được dùng để tạo ra yêu cầu ngắt không che được NMI gửi đến CPU. Trong sơ đồ có sử dụng thêm 4 mạch nhớ 4044 (là loại có dung lượng 4Kx1) để ghi các bit parity. Các vi mạch nhớ này được giải mã (chọn) thông qua ẵ mạch giải mã 74LS139 (mạch giải mã này gần giống 74LS138, nó được tách làm 2 nửa làm việc độc lập nhau). Trong sơ đồ này, ta sử dụng parity chẵn, tức là khi số bit 1 tại các đầu vào từ A đến H của 74LS280A là chẵn (I nối đất = 0) thì ta ghi 1 vào bộ nhớ parity, ngược lại ta ghi 0. Khi đọc dữ liệu, bit parity được đưa đến đầu vào I của 74LS280B, như vậy nếu các bit đầu đọc được tại các chân từ A đến H (dữ liệu) không đổi thì đầu ra Σ even của 74LS280B = 0 nên không xuất hiện yêu cầu ngắt NMI.

Với sơ đồ này ta chỉ có thể phát hiện sai tại 1 bit nào đó (là đặc điểm của parity) của byte nhớ (Nếu sai tại 2 bit thì parity không phát hiện được).

Trong thực tế tồn tại những mạch phức tạp hơn cho phép phát hiện và sửa sai ghi thêm vào byte dữ liệu nên có khả năng phát hiện sai tại 2 bit và tự động sửa sai được tại 1 bit lỗi (nếu chỉ lỗi tại 1 bit).

Việc ghép bộ vi xử lý 8088 – 5MHz (vi xử lý nói chung) với DRAN là phức tạp hơn vì cần có thêm các mạch dồn kênh địa chỉ và mạch làm tươi (refresh). Thông thường, để cho thông tin lưu trên DRAM là chính xác (được bảo toàn) thì ta phải làm tươi DRAM định kỳ sau mỗi 15,6 μs/lần.

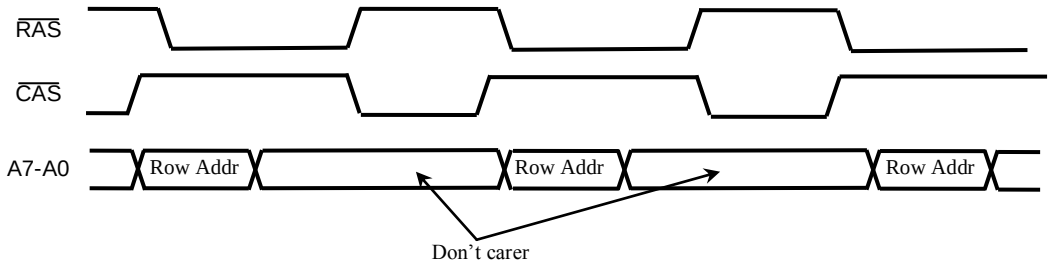
Các mạch DRAM thường có 2 chế độ làm tươi:

Làm tươi cả mảng: một mảng các phần tử nhớ gồm 128 đến 256 (thậm chí 512) hàng được chọn ra để làm tươi cứ 2 – 4 ms (hoặc 8 ms) trên một lần.

Làm tươi từng hàng: với tốc độ sao cho đảm bảo mỗi phần tử nhớ đều được làm tươi trong giới hạn 15,6 μs/lần.

Việc làm tươi được thực hiện bằng cách ghi hoặc đọc một loạt các ô nhớ. Số lương bit được làm tươi đồng thời phụ thuộc vào tổ chức bên trong của mỗi mạch nhớ. Một "chu kỳ làm tươi đặc biệt" là một chu kỳ khi việc làm tươi là công việc nội bộ của DRAMvà được hoàn tắt trong khi các thành phần (bộ phận) khác của bộ nhớ vẫn đang làm việc (trong suốt đói với CPU và hệ thống). Điều này được thực hiện bằng việc đưa ra các địa chỉ hàng và xung RAS nhằm chọn ra một hàng các phần tử cần được làm tươi (RAS only refresh). Xung RAS sẽ khiến cho dòng định chọn làm tươi được đọc và ghi lại. Địa chỉ làm tươi được lấy từ một bộ đếm 7,8 (hoặc 9) bit tuỳ theo kích thước của mạch nhớ cần làm tươi. Nội dung của bộ đếm này được tăng lên sau mỗi chu kỳ làm tươi sao cho tất cả các dòng đều được làm tươi trong thời gian đã định. Kiểu làm tươi này còn được gọi là làm tươi ẩn vì trong lúc làm tươi một vùng DRAM nào đó thì bộ vi xử lý vẫn ghie/đọc ở vùng nhớ khác.

Đối với DRAM, nói chung các nhà chế tạo đã cấu trúc mạch sao cho thời gian làm tươi một phần tử nhớ là 15,6 μs/lần.

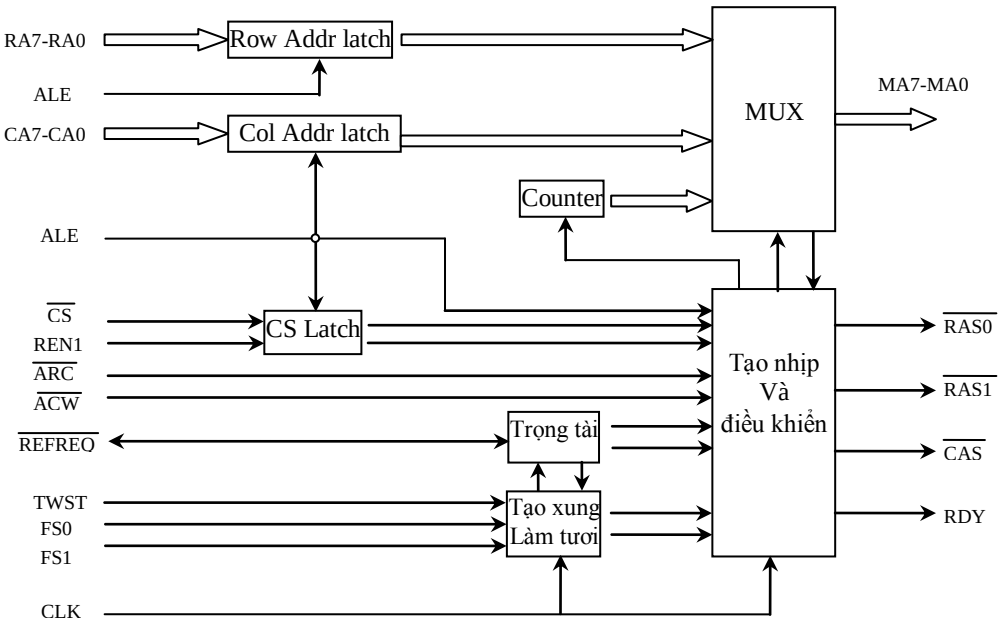


Hình vẽ: Sơ đồ khối bộ nhớ SRAM có kiểm tra parity

Khi phối ghép DRAM với bộ nhớ ta cần thực hiện:

- Dồn kênh 2 loại địa chỉ cho mỗi mạch nhớ và cung cấp xung cho phép chốt 2 loại địa chỉ trên là RAS và CAS.
- Cung cấp các tín hiệu điều khiển ghi/đọc bộ nhớ.
- Làm tươi mỗi hàng trong thời gian đảm bảo yêu cầu không mất mát thông tin.
- Đảm bảo không có tranh chấp trong hoạt động bình thường của vi xử lý với công việc làm tươi.

Để đơn giản các công việc khi phối ghép DRAM với vi xử lý, các nhà sản xuất vi mạch nhớ DRAM đã cung cấp các "bộ điều khiển DRAM" thực hiện các chức năng của mạch dồn kênh cho địa chỉ hàng, cột và mạch làm tươi mà không ảnh hưởng nhiều đến sự hoạt động bình thường của hệ vi xử lý.



Hình vẽ: Bộ điều khiển DRAM TMS 4500A

Các chân tín hiệu của bộ điều khiển DRAM TMS 4500A:

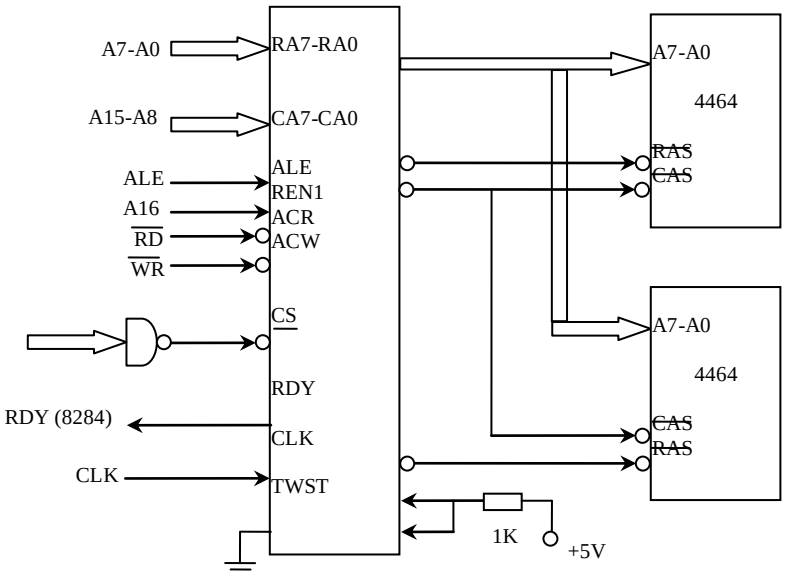
- RA7 – RA0 (Row Address) [I]: Địa chỉ hàng, thường được nối với Address bus tại các chân địa chỉ phần thấp: A7 – A0.

- CA7 – CA0 (Column Address) [I]: Địa chỉ cột, thường được nối với Address bus tại các chân địa chỉ phần cao: A15 – A8.
- MA7 – MA0 (Memory Address) [O]: Địa chỉ cho vi mạch nhớ, được nối trực tiếp với DRAM tại các chân địa chỉ: A7 – A0.
- ALE (Address Latch Enable) [I]: Tín hiệu cho phép chốt địa chỉ hàng, địa chỉ cột, REN và CS.
- CS (Chip Select) [I]: Xung chọn vi mạch để bắt đầu việc ghi/đọc DRAM, được nối qua bộ điều khiển trong thời điểm sườn xuống (âm) của xung ALE.
- REN1 [I]: Chọn một trong 2 khối nhớ DRAM nối với bộ điều khiển:
  - REN1 = 1 thì chọn RAS0
  - REN1 = 0 thì chọn RAS1
- ACR [I]: Sườn lên của xung này sẽ kết thúc việc đọc. Thường được nối với RD trong chế độ MIN của 8088.
- ACW [I]: Sườn lên của xung này kết thúc việc ghi. Thường được nối với WR trong chế độ MIN của 8088.
- CLK [I]: Đầu vào xung đồng hồ (nối với CLK của 8088).
- REFREQ [I/O]: Khi tín hiệu này là vào: để điều khiển việc bắt đầu chu kỳ làm tươi. Còn khi là ra: để báo cho bên ngoài biết bộ nhớ đang được làm tươi.
- RAS1, RAS0 [O]: Các chân cho phép chốt địa chỉ hàng cho các khối nhớ DRAM.
- CAS [O]: Chân cho phép chốt địa chỉ cột DRAM.
- RDY [O]: RDY = 0 khi DRAM đang được làm tươi, nên chưa sẵn sàng (chân này được nối với RDY của 8088).
- TWST (Timming/Wait Strap) [I]: Chọn trạng thái chờ và các thông số thời gian khác. Khi TWST = 1 thì bộ vi xử lý phải xen thêm một số trạng thái chờ (1 Tw) mỗi khi ghi/đọc bộ nhớ DRAM.
- FS1, FS0 [I]: Cùng với TWST để chọn chế độ làm việc theo như bảng sau:

Bảng:

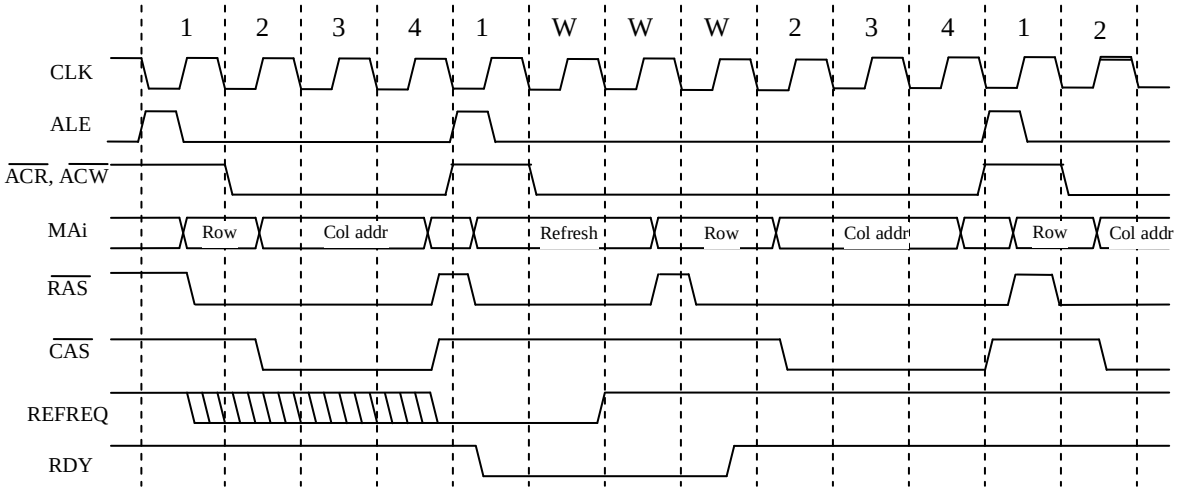
TWST	FS1	FS0	Số trạng thái chờ	Chu kỳ làm tươi	Tần số xung làm tươi (KHz)	Số nhịp cho 1 lần làm tươi
0	0	0	0	Ngoài	REFREQ	4
0	0	1	0	Tclkx31	64-95	3
0	1	0	0	Tclkx46	64-85	3
0	1	1	0	Tclkx61	64-82	4
1	0	0	1	Tclkx46	64-85	3
1	0	1	1	Tclkx61	64-80	4
1	1	0	1	Tclkx76	64-77	4
1	1	1	1	Tclkx91	64-88	4

Hoạt động của TMS 4500A:



Ví dụ: Ví dụ về một ứng dụng của bộ điều khiển DRAM TMS 4500A trong (để điều khiển) mạch nhớ gồm 4x4464 để tạo nên bộ nhớ 128 KB bắt đầu tại địa chỉ 00000h.

Trong sơ đồ, việc ghép nối chân REN1 với đường địa chỉ A16 để chọn ra 2 vùng nhớ 64 KB và CS nối với mạch NAND để đặt 2 mảng nhớ vào địa chỉ đã định. Các chân TWST, FST1, FST0 được đặt tương ứng là 0, 1 và 1 để không có trạng thái chờ và việc làm tươi được tiến hành sau 61 chu kỳ đồng hồ và kéo dài trong 4 nhịp của xung làm tươi. Với tần số fclk = 5 MHz thì nhịp làm tươi cho mỗi hàng là 12,2 μs (≈giá trị yêu cầu là 15,6 μs. Trong khi bộ điều khiển làm tươi đang làm tươi bộ nhớ thì RDY = 0 và CPU bị đưa vào trạng thái chờ.



Hình vẽ: Biểu đồ thời gian của bộ điều khiển DRAM TMS 4500A

4.3. Phối ghép 8088 với thiết bị ngoại vi

4.3.1. Các kiểu phối ghép vào ra

Đối với bộ vi xử lý 8088 của Intel (hay Intel 80x86 Family) có 2 cách phối ghép CPU với thiết bị ngoại vi (các cổng vào/ra – I/O ports), đó là:

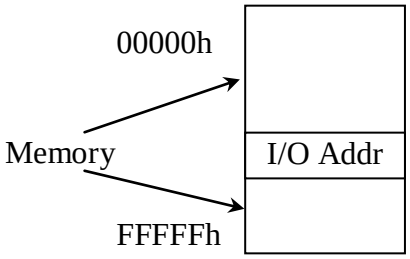
- Thiết bị ngoại vi có không gian địa chỉ chung với bộ nhớ và
- Thiết bi ngoại vi có không địa chỉ tách biệt với bộ nhớ.
- Kiểm tra Parity để phát hiện lỗi trong bộ nhớ RAM

Thiết bị ngoại vi (I/O) có không gian địa chỉ tách biệt

Trong (với) cách phối ghép này, bộ nhớ được dùng hoàn toàn không gian !MB địa chỉ mà CPU dành cho nó. Các thiết bị ngoại vi (ports) sẽ được dành riêng một không gian 64 K cổng cho mỗi loại cổng vào hoặc ra. Tất nhiên ta phải dùng tín hiệu IO/M và các lệnh vận chuyển dữ liệu một cách thích hợp cho mỗi không gian nói trên.

	Không gian bộ nhớ	**Không gian cổng vào/ra**	
		Thiết bị vào	**Thiết bị ra**
	00000h 1MB FFFFFh	0000h 64K FFFFh	0000h 64K FFFFh
Kiểu lệnh	Mov	In	Out
Tín hiệu điều khiển	IO/M = 0	IO/M = 1	IO/M = 1

Không gian địa chỉ cho bộ nhớ và thiết bị ngoại vi



Thiết bị ngoại vi (I/O) có không địa chỉ chung với bộ nhớ

Kiểu lệnh vận chuyển dữ liệu: Mov

Tín hiệu điều khiển: IO/M = 0

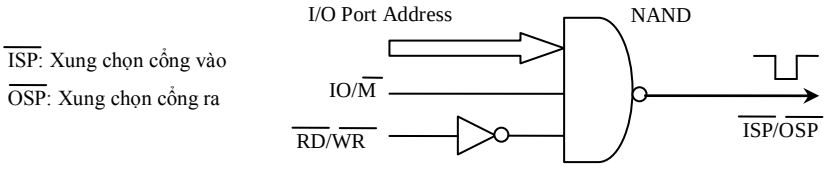
Trong cách phối ghép này, bộ nhớ và thiết bị ngoại vi cùng nhau chia sẻ không gian địa chỉ 1 MB mà CPU 8088 có khả năng địa chỉ hoá (quản lý). Các thiết bị ngoại vi sẽ chiếm một vùng địa chỉ nào đó trong không gian 1 MB, phần còn lại sẽ là của bộ nhớ. Tất nhiên trong trường hợp này ta dùng tín hiệu điều khiển IO/M = 0 và lệnh vận chuyển dữ liệu là MOV cho cả bộ nhớ và thiết bị ngoại vi – vì CPU hiểu địa chỉ là địa chỉ của bộ nhớ mà không hề biết rằng tại địa chỉ đó có thiết bị ngoại vi.

4.3.2. Giải mã địa chỉ cho thiết vị vào/ra

Việc giải mã địa chỉ cho thiết bị vào/ra (I/O) cũng gần giống như giải mã địa chỉ cho bộ nhớ. ở đây, nhấn mạnh việc giải mã địa chỉ cho các cổng. Thông thường các cổng có địa chỉ là 8 bits (A7 – A0), trong một số hệ vi xử lý khác (như máy IBM PC hoặc các máy tính tương thích IBM PC) thì các cổng có địa chỉ 16 bits (A15 – A0, nhưng vì với 16 bits => có 64K cổng mà thực tế ít có máy sử dụng hết không gian này => thường chỉ dùng 12 bits). Tuỳ theo độ dài toán hạng trong lệnh là 8 hay 16 bits mà ta có 1 cổng 8 bits hay 2 cổng 8 bits có địa chỉ liền nhau (tại mỗi địa chỉ cổng là một thanh ghi đệm/chốt 8 bit. Điều này không còn đúng đối với các hệ vi xử lý có bộ vi xử lý 32 bits hoặc cao hơn, mỗi cổng có thể có số bits cực đại bằng số bits của bus dữ liệu). để tạo nên word có độ dài tương ứng. Trong thực tế ít có hệ vi xử lý nào sử dụng hết 256 địa chỉ cổng I/O khác nhau nên để đơn giản ta

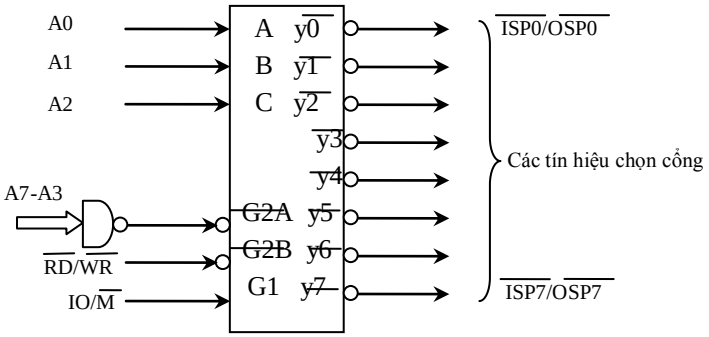
chỉ xét các bộ giải mã địa chỉ 8 bits (A7 – A0) và các mạch giải mã thông dụng sẵn có (như 74LS138 chẳng hạn) để tạo ra các xung chọn cho thiết bị ngoại vi.

Các mạch giải mã đơn giản có thể được tạo từ các mạch NAND và một số gates logic phụ như hình vẽ.



Hình vẽ: Mạch giải mã địa chỉ cho I/O dùng NAND

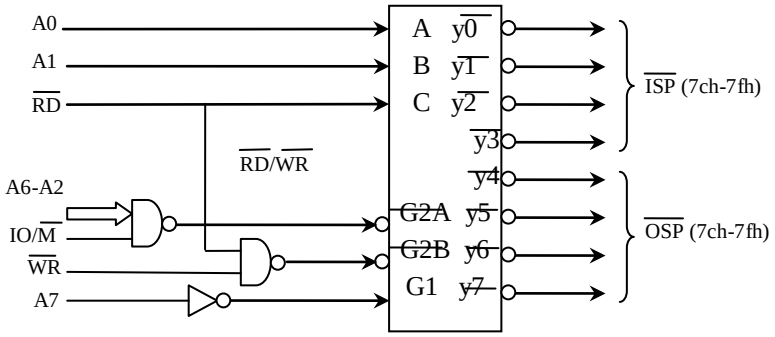
Trong trường hợp cần nhiều xung chọn thiết bị ở đầu ra mạch giải mã cho các cổng có địa chỉ liên tiếp, ta có thể sử dụng mạch giải mã có sẵn 74LS138 nhs sau:



Hình vẽ: Mạch giải mã địa chỉ cho I/O dùng 74LS138

Mạch giải mã này có thể giải mã cho 8 cổng vào (ISP) hoặc 8 cổng ra (OSP) tương ứng việc sử dụng tín hiệu điều khiển RD hoặc WR. Trên cơ sở của mạch giải mã này ta cũng có thể phối ghép cả 2 tín hiệu đọc (RD) và ghi (WR) để tạo ra tín hiệu chọn cho việc đọc/ghi từng cổng vào/ra cụ thể. Ta cũng có thể sử dụng 74LS138 để giải mã cho 4 cổng vào và 4 cổng ra như sau:

Do tín hiệu điều khiển RD và WR là ngược pha nhau nên khi đọc cổng (RD = 0) => C = 0 => A1, A0 chọn ra 1 trong 4 cổng thấp (y0 – y3). Ngược lại khi ghi cổng (WR = 0) => RD = 1 => C = 1 => A1, A0 chọn ra 1 trong 4 cổng cao (y4 – y7).



Hình vẽ: Mạch giải mã địa chỉ cho 4 Input port và 4 output port

4.3.3. Các mạch cổng đơn giản

Trong thực tế có rất nhiều vi mạch tổ hợp cỡ vừa có thể dùng làm mạch cổng phối ghép với thiết bộ vi xử lý để vào/ra dữ liệu. Các mạch này thường được cấu tạo từ các mạch:

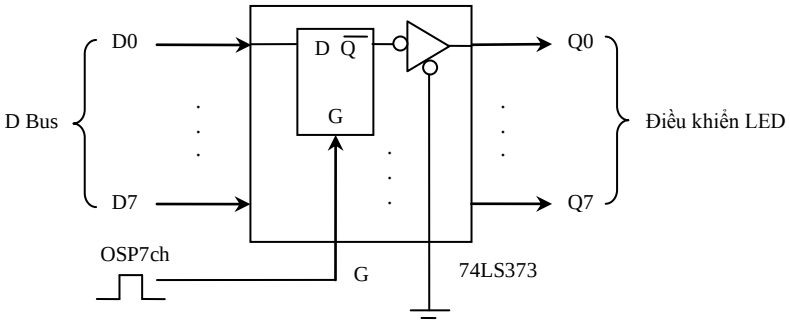
- Chốt 8 bit có đầu ra 3 trạng thái:
  - 74LS373: kích theo mức.
  - 74LS374: kích theo sườn.
- Các mạch khuếch đại đệm 8 bit 1 chiều có đầu ra 3 trạng thái (đầu ra đảo hoặc không đảo):
  - 74LS240: kích theo mức.
  - 74LS244: kích theo mức.

- Các mạch khuếch đại đệm 8 bit 2 chiều có đầu ra 3 trạng thái: 74LS245.

Tất thảy, chúng được dùng trong các phối ghép đơn giản để làm cho CPU và thiết bị ngoại vi hoạt động tương thích nhau, ví dụ như để đệm bus hoặc các mạch cổng để tạo các tín hiệu móc nối (handshaking signal).

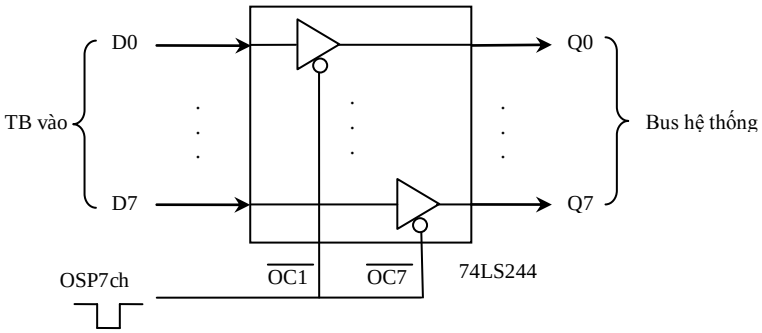
(a): Sơ đồ ví dụ về mạch phối ghép đơn giản dùng mạch 74LS373 để đưa tín hiệu ra điều khiển đèn LED bằng lệnh:

OUT 7ch, al.



(b): Sơ đồ ví dụ về một mạch phối ghép đơn giản dùng mạch 74LS 244 để đọc tín hiệu từ thiết bị ngoại vi và CPU bằng lệnh:

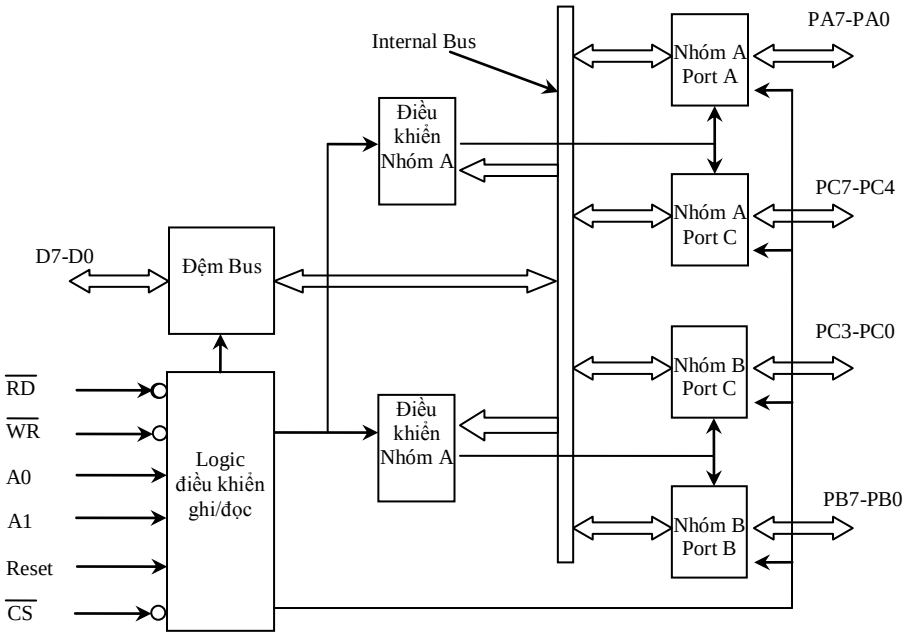
IN al, 7ch.



Hình vẽ: Sơ đồ một số mạch đệm chốt: ...244, ...245, ...373, ...374

4.3.4. Mạch phối ghép vào/ra song song lập trình được PPI 8255

Mạch PPI 8255 được gọi là vi mạch phối ghép vào/ra song song lập trình được (Programmable Peripheral Interface – PPI). Do khả năng mềm dẻo trong các ứng dụng nên nó là mạch phối ghép được dùng rất phổ biến cho các hệ vi xử lý: 8, 16, và cao hơn.



Hình vẽ: Sơ đồ khối của PPI 8255

Các chân tín hiệu của PPI 8255:

Reset [I]: Phải nối với tín hiệu Reset chung của toàn hệ thống (khi reset thì tất cả các cổng của vi mạch được định nghĩa là cổng vào để không gây sự cố cho các mạch điều khiển và các cơ cấu chấp hành).

CS [I]: Được nối với mạch giải mã, nhận tín hiệu xung chọn thiết bị để đặt 8255 vào một địa chỉ cơ sở nào đó.

A1, A0 [I]: Là các bits địa chỉ, thường được nối với các bits địa chỉ thấp nhất của Address bus để chọn ra 4 thanh ghi bên trong 8255 (3 thanh ghi 8 bits cho 3 cổng và một thanh ghi 8 bits để ghi từ điều khiển cho hoạt động của 8255, Control Word – CW)

Theo bảng bên ta thấy: PA có địa chỉ trùng với địa chỉ cơ sở của vi mạch 8255.

CS	A1	A0	Chọn ra
1	x	x	Không chọn
0	0	0	PA
0	0	1	PB
0	1	0	PC
0	1	1	CWR

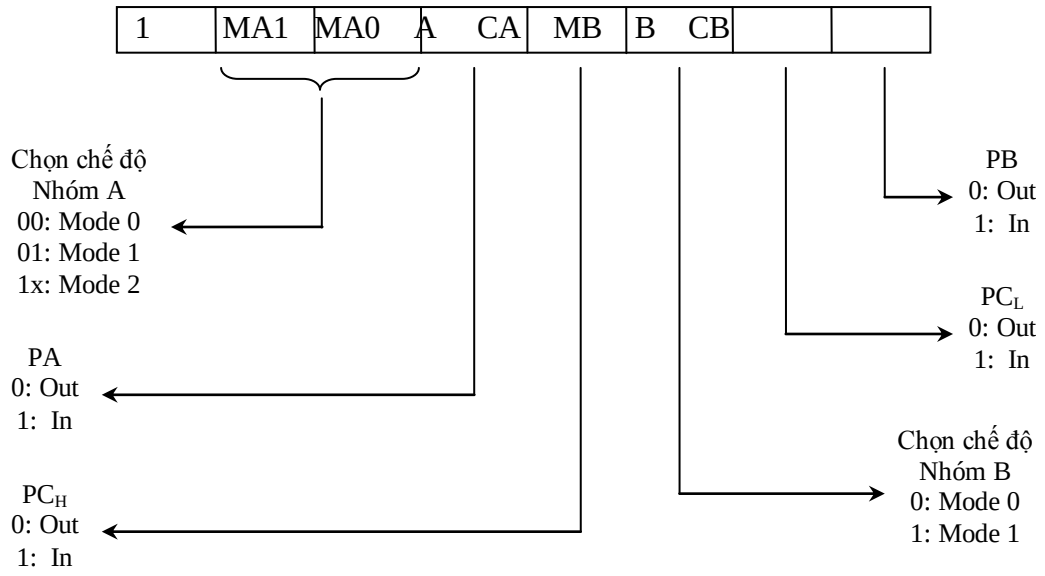
Bảng: Các thanh ghi được chọn theo CS và A1, A0

Có 2 loại từ điều khiển cho 8255 là:

- Từ điều khiển định nghĩa cấu hình cho các cổng: PA, PB, PC.
- Từ điều khiển lập/xoá các bits ở thanh ghi ra (cổng) PC (PCi).

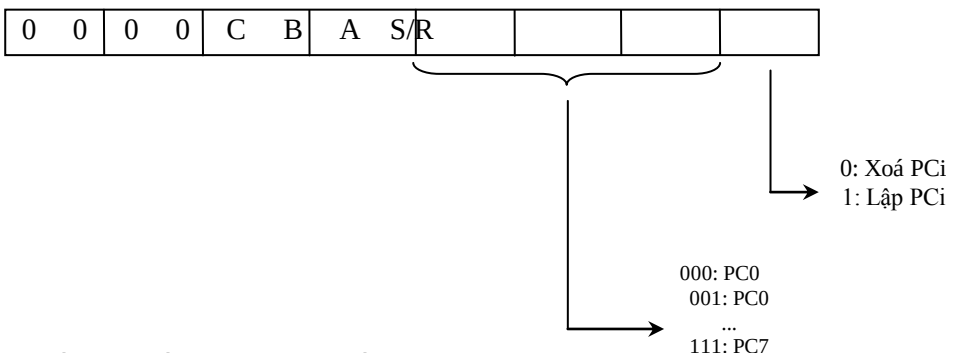
Từ điều khiển định nghĩa cấu hình

Từ điều khiển định nghĩa cấu hình (CWR) có dạng thức như sau:



Từ điều khiển lập xoá các bits đầu ta của PC (PCi):

Nhận xét: Ta thấy các chế độ (modes) làm việc của vi mạch PPI 8255 có thể định nghĩa bằng từ điều khiển CWR. Cụ thể 8255 có 4 chế độ làm việc như sau:



- Chế độ 0 (mode 0): Chế độ vào/ra cơ sở: Còn được gọi là chế độ vào/ra đơn giản. Trong chế độ này mỗi cổng PA, PB, PCH, PCL đều có thể được định nghĩa là cổng vào hoặc cổng ra nên có 16 tổ hợp khác nhau trong chế độ này.
- Chế độ 1 (mode 1): Vào/ra có xung cho phép: Trong chế độ này, mỗi cổng PA, PB có thể được định nghĩa là cổng vào hoặc ra (4 tổ hợp) với các tín hiệu móc nối (bắt tay – handshaking) do các bit tương ứng trên cổng PC đảm nhiệm.
- Chế độ 2 (mode 2): Vào/ra 2 chiều: Trong chế độ này chỉ riêng cổng PA có thể được định nghĩa là cổng vào/ra 2 chiều với các tín hiệu móc nối do các bit ở cổng PC đảm nhiệm. Còn cổng PB có thể làm việc ở chế độ 0 hoặc 1.
- Lập/xoá các bits PCi: Chế độ này được hoạt động theo sự thiết lập từ điều khiển lập/xoá bit ra PC. Một vài ứng dụng của chế độ này:
  - Tạo dãy xung nào đó trên PC.
  - Điều khiển cho việc đóng cắt cơ cấu nào đó.
  - Điều khiểm motor bước (điều chỉnh tốc độ chẳng hạn).

Cụ thể 8255 có 4 chế độ làm việc như sau: Cụ thể các chế độ làm việc 0,1 và 2 của 8255 như sau:

Chế độ 0 (Mode 0): Vào/ra cơ sở (vào/ra đơn giản)

Như đã biết trong chế độ này, bốn cổng PA, PB, PCH, PCL đều có thể được định nghĩa là cổng vào hoặc cổng ra. Như vậy, với tổ hợp tất cả các khả năng vào/ra cho bốn cổng trên ta có được 16 cấu hình tổ hợp khác nhau.

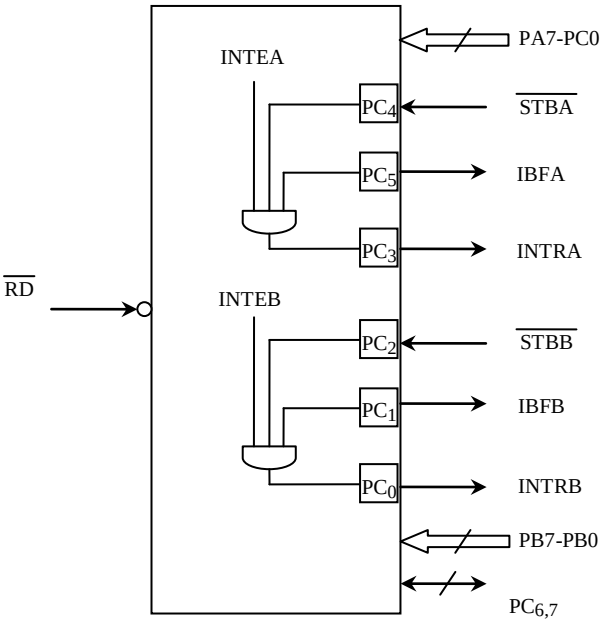
Cấu hình	PA	PB	PC_H	PC_L
1	Vào	Vào	Vào	Vào
2	Vào	Vào	Vào	Ra
...				
16	Ra	Ra	Ra	Ra

Khi cổng là ra: số liệu viết ra cổng sẽ được chốt. Byte số liệu sẽ tồn tại trên các đầu ra của cổng cho đến khi một byte mới được viết tới đó.

Khi cổng là vào: Đọc được byte giá trị của byte trên các đầu nối của cổng. Các số liệu này không được chốt (luôn luôn thay đổi theo giá trị đầu vào).

Chế độ 1 (Mode 1): Vào/ra có xung cho phép

Như đã biết, trong chế độ này mỗi cổng PA, PB có thể được định nghĩa là cổng vào hoặc cổng ra với các tín hiệu móc nối (handshaking) do các bit tương ứng của cổng PC trong cùng nhóm đảm nhiệm. Để đơn giản, ta coi PA và PB cùng được định nghĩa là cổng vào hoặc cổng ra (cùng vào hoặc cùng ra).



Hình vẽ: Vào dữ liệu trong chế độ 1 (PA, PB: cùng vào)

Vào dữ liệu trong chế độ 1

Trong đó:

STB: Strobe
IBF: Input Buffer Full
INTR: Interrupt
INTE: Interrup Enable

Ở đây các cổng PA và PB cùng được định nghĩa là cổng vào và có các tín hiệu móc nối tương đương nhau cho việc trao đổi dữ liệu. Để đơn giản ta chỉ cần nói các tín hiệu điều khiển cho PA, còn các tín hiệu cho PB hoàn toàn tương tự.

D7	D6	D5	D4	D3	D2	D1	D0
I/O	I/O	IBFA	/STBA	INTRA	/STBB	IBFB	INTRB

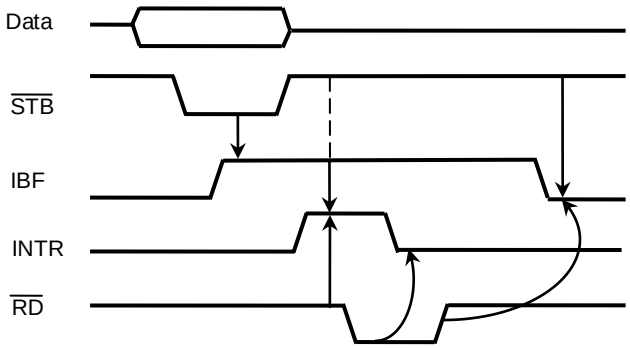
Hình vẽ: Các bits (control signals) trên PC (vào dữ liệu chế độ 1)

/STB (Strobe – Cho phép chốt dữ liệu): Khi dữ liệu đã săn sàng để được đọc vào cổng PA, thiết bị ngoại vi phải dùng tín hiệu /STB để báo cho PPI 8255 biết mà chốt dữ liệu.

IBFA (Input Buffer Full A port): Sau khi PPI 8255 chốt được dữ liệu do thiết bị ngoại vi đưa đến, nó đưa ra tín hiệu IBFA để báo cho thiết bị ngoại vi biết là nó đã chốt xong.

INTRA (Interrupt A Port – Yêu cầu ngắt từ cổng PA): Tín hiệu (của 8255) để báo cho CPU biết là đã có dữ liệu sẵn sàng để đọc từ PA. Đây là kết quả thu được từ quan hệ giữa các tín hiệu khác của 8255 trong quá trình hội thoại với thiết bị ngoại vi (cần chú ý xem kỹ biểu đồ thời gian về quan hệ giữa các tín hiệu).

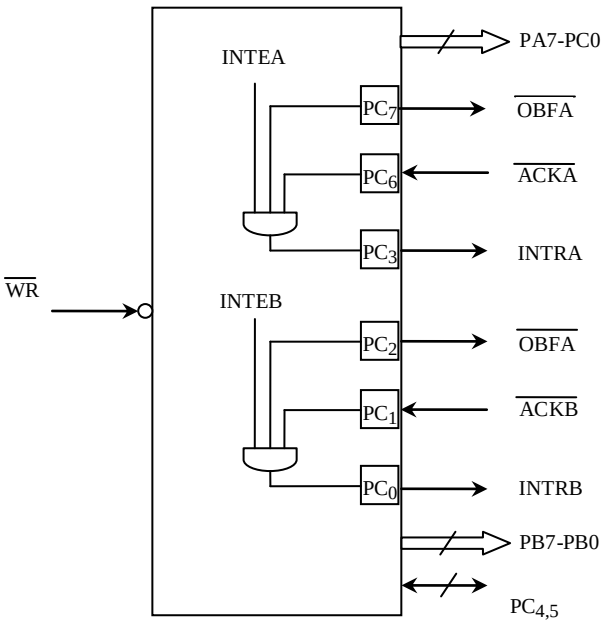
INTEA (Interrupt Enable A Port): Là tín hiệu của một mạch lật bên trong 8255 để cho phép hoặc cấm yêu cầu ngắt INTRA của PA. Tín hiệu INTEAđược lập xoá (cùng pha) với tín hiệu (thông qua) tại bit PC4 của cổng PC. Điều này có nghĩa là khi /STB là tích cực thì không có tín hiệu INTRA. Tại mức thấp của tín hiệu /STB (tích cực) thì xuất hiện tín hiệu IBF báo cho thiết bị ngoại vi là 8255 đã chốt xong. Khi STB = 1 thì xuất hiện tín hiệu INTRA yêu cầu ngắt CPU để phục vụ cho việc đọc dữ liệu từ cổng PA. (INTE ≅ STB).



Hình vẽ: Biểu đồ thời gian của các tín hiệu của 8255 ở chế độ 1 - vào

Một trong số các tín hiệu đối thoại – trạng thái nói trên (đều) có thể lấy được trực tiếp từ các chân tương ứng của vi mạch hoặc có thể được đọc vào thông qua việc đọc PC (Theo hình vẽ: Các bit trên PC – Vào dữ liệu trong chế độ 1).

Các tín hiệu đối thoại – trạng thái có thể được sử dụng làm tín hiệu móc nối cho các kiểm tra vào/ra dữ liệu bằng cách ngắt bộ vi xử lý hay bằng cách thăm dò trạng thái sẵn sàng của thiết bị ngoại vi (mà chúng ta sẽ nghiên cứu trong phần sau).



Hình vẽ: Vào dữ liệu trong chế độ 1 (PA, PB: cùng vào)

Ra dữ liệu trong chế độ 1

Trong đó:

ACK: Strobe
OBF: Input Buffer Full
INTR: Interrupt

INTE: Interrup Enable

D7	D6	D5	D4	D3	D2	D1	D0
/OBFA	/ACKA	I/O	I/O	INTRA	/OBFA	/ACKB	INTRB

Hình vẽ: Các bits (control signals) trên PC
(RA dữ liệu chế độ 1)

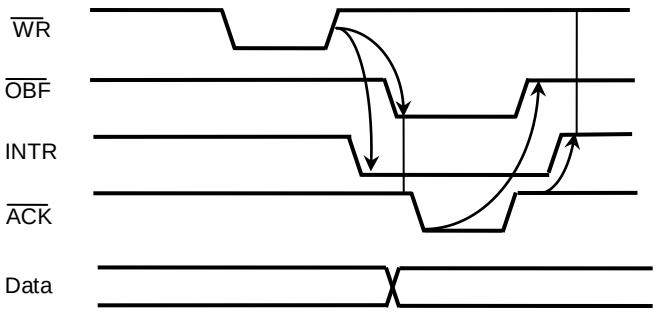
Ở đây các cổng PA, PB cùng được định nghĩa là cổng ra và có các tín hiệu móc nối tương đương nhau cho việc trao đổi dữ liệu. Để đơn giản ta chỉ xét tới các tín hiệu (cho) liên quan cổng PA, còn các tín hiệu cho PB hoàn toàn tương tự.

/OBFA (Output Buffer Full A port - Đệm ra của PA đầy): Tín hiệu báo cho thiết bị ngoại vi biết CPU đã ghi dữ liệu vào cổng (PA) để chuẩn bị đưa ra. Tín hiệu này thường được nối với /STB của thiết bị ngoại vi (thiết bị nhận).

ACKA (Acknowlegde A port – Trả lời (xác nhận) đã nhận được dữ liệu): Đây là tín hiệu của thiết bị ngoại vi trả lời cho biết là nó đã nhận được dữ liệu từ PA của 8255.

INTRA (Interrupt A Port – Yêu cầu ngắt từ cổng PA): Đây là kết quả thu được từ quan hệ giữa các tín hiệu của 8255 trong quá trình đối thoại với thiết bị ngoại vi, nó được dùng để phản ánh yêu cầu ngắt của PA tới CPU (cần chú ý và xem kỹ biểu đồ thời gian về các quan hệ giữa các tín hiệu – ra dữ liệu trong chế độ 1).

INTEA (Interrupt Enable A Port): Là tín hiệu của một mạch lật bên trong 8255 để cho phép hoặc cấm yêu cầu ngắt INTRA của PA. Tín hiệu INTEA được lập xoá (cùng pha) với tín hiệu (thông qua) tại bit PC6 của cổng PC.

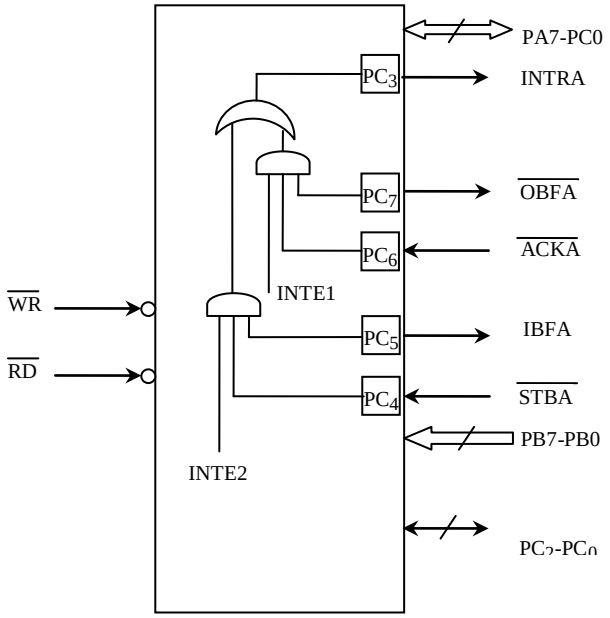


Hình vẽ: Biểu đồ thời gian của các tín hiệu của 8255 ở chế độ 1 - vào

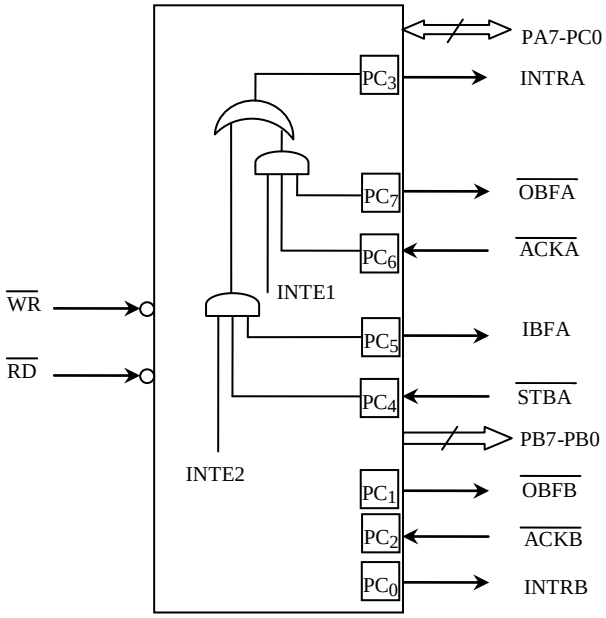
Các tín hiệu đối thoại – trạng thái nói trên có thể lấy trực tiếp từ các chân tương ứng của vi mạch hoặc đọc vào CPU thông qua việc đọc cổng PC (Theo hình vẽ: Các bits trên PC – ra trong chế độ 1).

- **Chế độ 2 (Mode 2): Vào/ra 2 chiều cho PA**

Trong chế độ này, chỉ riêng cổng PA được định nghĩa để làm việc như một cổng 2 chiều có các tín hiệu móc nối do 1 số bits của cổng PC đảm nhiệm, còn cổng PB thì có thể làm việc trong chế độ 0 hoặc 1 tuỳ theo giá trị các bit điều khiển trong CWR (do người sử dụng lựa chọn và đặt). Các chân tín hiệu còn lại của PC có thể được định nghĩa để làm việc như các tín chân vào hoặc ra, hoặc phục vụ cho cổng PC.



Hình vẽ: PA: Mode 2; PB Mode 0/vào
PC2-PC0: I/O tuỳ theo CWR



Hình vẽ: PA: Mode 2
PB: Mode 1/ra

D7	D6	D5	D4	D3	D2	D1	D0
/OBFA	/ACKA	IBFA	/STBA	INTRA	I/O	I/O	I/O

Hình vẽ: Các bits (control signals) trên PC
(RA dữ liệu chế độ 1)

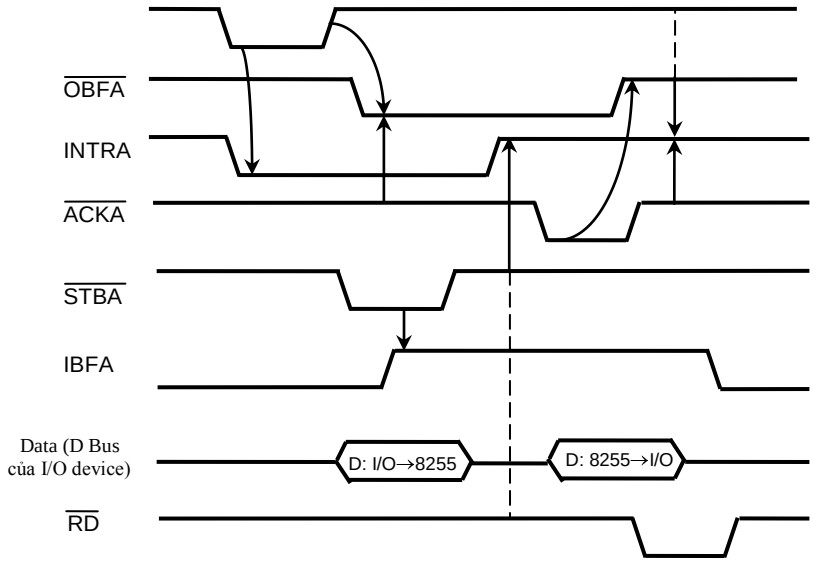
D7	D6	D5	D4	D3	D2	D1	D0
/OBFA	/ACKA	IBFA	/STBA	INTRA	/ACKB	/OBFB	INTRB

Hình vẽ: Các bits (control signals) trên PC
(RA dữ liệu chế độ 1)

Một số tín hiệu móc nối đặc biệt của PA gồm:

INTRA (Interrupt A Port – Yêu cầu ngắt từ cổng PA): Tín hiệu yêu cầu ngắt cho việc vào hoặc ra dữ liệu của PA (dữ liệu có thể vào/ra theo 2 chiều).

INTEA1, INTEA2: Là 2 tín hiệu của 2 mạch lật trong 8255 để: cho phép hoặc cấm các ngắt của cổng PA (thiết bị nối vào cổng PA). Các tín hiệu này được lập/xoá cùng pha bởi các bits tương ứng là PC6 và PC4 của cổng PC.



Hình vẽ: Biểu đồ thời gian của các tín hiệu của 8255 ở chế độ 2

Một số trong các tín hiệu đối thoại – trạng thái kể trên đều có thể lấy được trực tiếp từ các chân tương ứng của vi mạch hoặc có thể được đọc vào CPU từ cổng PC và cho phép điều khiển việc trao đổi dữ liệu bằng cách thăm dò các tín hiệu này.

Khi dùng PPI 8255 trong mode 2 (vào/ra 2 chiều) để trao đổi dữ liệu theo cách thăm dò ta phải kiểm tra xem bit OBFA có bằng 1 (đệm ra rỗng) hay không trước khi dùng lệnh OUT để đưa dữ liệu ra cổng PA, hoặc kiểm tra bit IBFA có bằng 0 (đệm vào rỗng) hay không trước khi dùng lệnh IN để đọc dữ liệu vào từ cổng PA.

Một số ví dụ về cách lập trình cho PPI 8255

Ví dụ 1: Xây dựng mạch giải mã cho PPI 8255 tại địa chỉ 314h

- Đọc PA, tăng giá trị đọc được lên 1 rồi đưa ra PB.
- Đọc PCH, giảm giá trị đọc được đi1 rồi đưa ra PCL.

Công việc trên cần thực hiện 100 lần.

Giải:

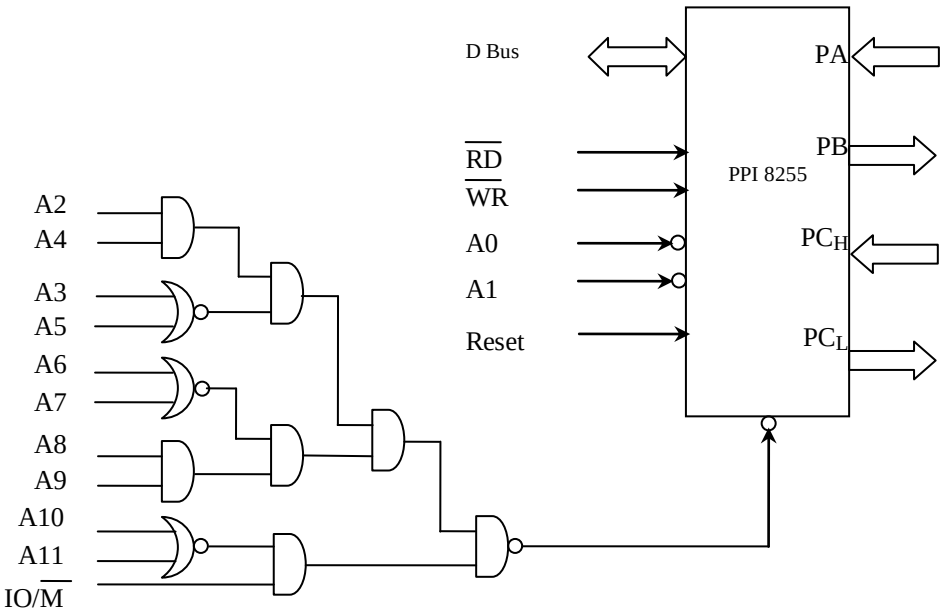
Theo yêu cầu đề bài ta có:

CONFIG	MA0	MA1	A	CA	MB	B	CB
1	0	0	1	1	0	0	0

PA: vào; PB: ra; PCH: vào; PCL: ra; 8255 làm việc ở mode 0.

Giá trị cần khởi tạo tại CWR là: 98h.

Có thể dùng các phần tử gates logic để thực hiện (xây dựng) mạch giải mã cho 8255 tại địa chỉ 314h = 0011 0001 0100 như hình vẽ:



Hình vẽ: Sơ đồ

Đoạn chương trình thực hiện:

```
; Định nghĩa các hằng địa chỉ cho các cổng để đơn giản trong việc viết lệnh
      PA      Equ    314h
```

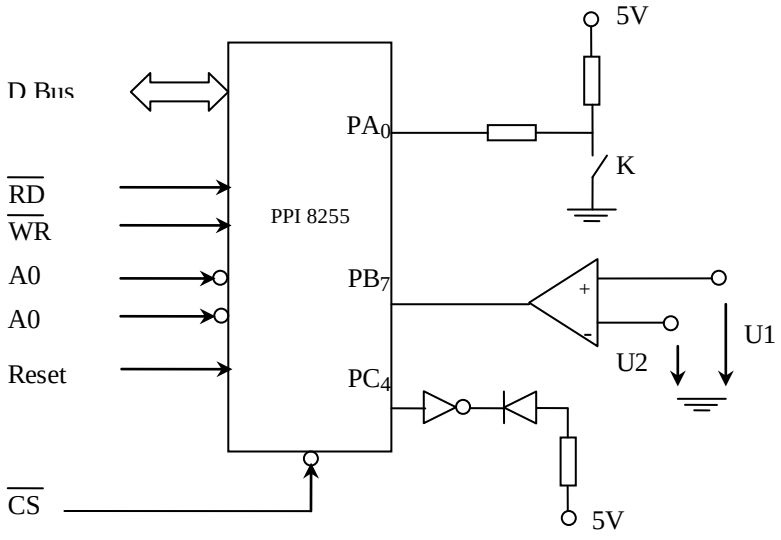
```
PB      Equ   315h
PC      Equ   316h
CWR     Equ   317h
  …
Mov     al, 98h       ; Giá trị cần khởi tạo cho 8255
Out     CWR, al             ; theo yêu cầu đề bài
Mov     cx, 100             ; Thực hiện 100 lần đọc và xử lý số liệu
Lặp:
In      al, PA              ; Đọc cổng PA
Inc     al                  ; Tăng giá trị đọc được lên 1 đơn vị
Out     PB, al              ; rồi đưa ra cổng PB
In      al, PC              ; Đọc cổng PCH
Push    cx                  ; Cất tạm biến đếm vòng lặp
Mov     cl, 4               ; Quay al 4 lần để
Rol     al, cl              ; để chuyển 4 bit cao thành 4 bit thấp
Dec     al                  ; rồi giảm đi 1 đơn vị và
Out     PC, al              ; đưa ra cổng PCL
Pop     cx                  ; Lấy lại biến đếm vòng lặp
Loop    Lặp
Ra: …
```

Ví dụ 2: (Chỉnh lại ví dụ 3 trong sách KT vi xử lý – Văn Thế Minh)

Có mạch 8255 với địa chỉ cơ sở là 300h nối với các phần tử ngoại vi đơn giản (hình vẽ). Lập trình để khi có U2>U1 thì đọc trạng thái công tắc K, nếu K đóng thì cho LED tắt, ngược lại cho LED sáng.

Giải:

Theo yêu cầu đề bài, để giải quyết ta (cần) định nghĩa các hằng cho các cổng và từ điều khiển để định nghĩa cấu hình cho 8255 sao cho:



PA, PB là cổng vào, PCH là cổng ra, 8255 làm việc ở mode 0.

Trong chương trình: đọc PB, kiểm tra bit PB7 = 1?, nếu không thì đọc lại, nếu có thì đọc PA, kiểm tra bit PA0, = 0?, nếu có (K đóng) thì ghi vào PC với PC4=0 (tắt đèn), ngược lại ghi vào PC với PC4=1 (sáng đèn).

CONFIG	MA0	MA1	A	CA	MB	B	CB
1	0	0	1	0	0	1	1

PA, PB: vào; PCL không dùng (vào hoặc ra); PCH: ra; 8255 làm việc ở mode 0.

Giá trị cần khởi tạo tại CWR là: 93h.

Đoạn chương trình thực hiện:

```
; Định nghĩa các hằng địa chỉ cho các cổng để đơn giản trong việc viết lệnh
    PA      Equ   300h
    PB      Equ   301h
    PC      Equ   302h
    CWR     Equ   303h
      …
    Mov     al, 98h       ; Giá trị cần khởi tạo cho 8255
    Out     CWR, al             ; theo yêu cầu đề bài
DocPB:
    In      al, PB              ; Đọc cổng PB
    And     al, 80h       ; PB7 = 1?
    Jz      DocPB               ; Không, đọc lại PB để kiểm tra PB7
    In      al, PA              ; Phải, đọc PA
    And     al, 1               ; PA0 = 0?
    Jz      Off                 ; Phải, đến tắt đèn
    Mov     al, 10h       ; Không, cho đèn sáng (lal = 0001 0000)
    Ou      PC, al              ; Ghi PC4 ra cổng PCH
    Jmp     Ra
Off:
    Out     PC, al              ; Ghi PC4 ra cổng PCL
Ra: …
```

Chú ý: Chương trình điều khiển việc vào/ra dữ liệu theo phương pháp thăm dò (polling) mà ta sẽ xét trong phần sau. 8255 sử dụng ở mode 0 (vào/ra đơn giản). Ta cũng có thể sử dụng 8255 ở mode 1 để giải quyết vấn đề của bài toán.

Ví dụ 3: (Theo tài liệu KT vi xử lý – Văn Thế Minh)

Lập trình để bit PC4 của 8255 tạo ra 256 xung với chu kỳ T = 50 ms, độ rỗng 50%. Giả thiết có sẵn chương trình con làm nhiệm vụ trễ 25 ms có tên là Delay25ms, địa chỉ của 8255 là: 304h.

Giải:

Với yêu cầu của bài toán như vậy, ta nghĩ đến việc cho 8255 làm việc ở chế độ lập/xoá các bit trên PC (PC4).

Vì địa chỉ của 8255 là 304h nên địa chỉ của CWR là 307h.

Giá trị cần đưa ra CWR khi có xung là: 0000 1001 = 09h

Giá trị cần đưa ra CWR khi không có xung là: 0000 1000 = 08h

Đoạn chương trình thực hiện:

```
; Định nghĩa các hằng địa chỉ cho các cổng để đơn giản trong việc viết lệnh
    CWR     Equ   307h
      …
    Mov     cx, 100h            ; Cần tạo ra 256 xung
    Cli                         ; Cấm các ngắt che được
    Mov     al, 09              ; Chuẩn bị để tạo xung
Lặp:
    Out     CWR, al             ; Lập bit (xung) tại PC4
    Call    Delay25ms           ; Kéo dài 25 ms
```

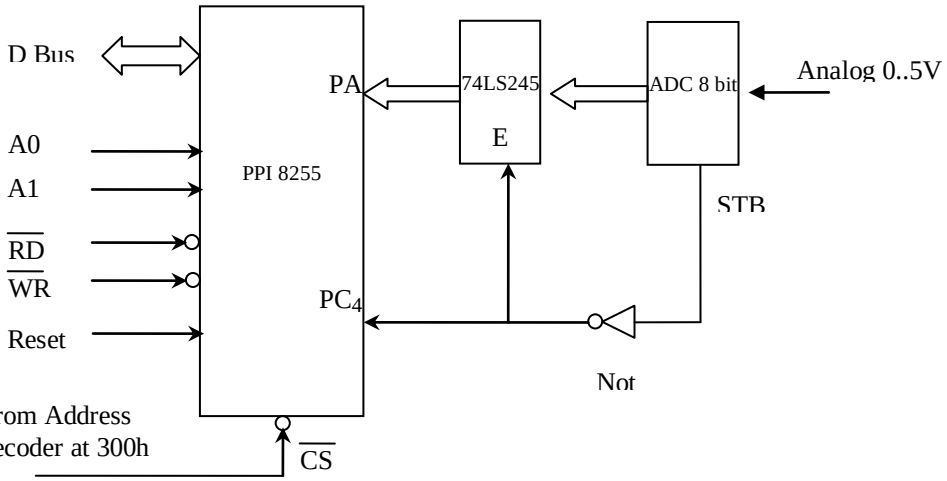
```
Dec     al              ; al = 08, xoá bit
Out     CWR, al         ; Xoá xung (xoá PC4)
Call    Delay25ms       ; Kéo dài 25 ms (độ rỗng 25%)
Inc     al              ; Chuẩn bị để lập PC4
Loop    Lặp
Sti                     ;Cho phép ngắt trở lại
…
```

Chú ý: Thực ra dãy xung này chưa hoàn toàn chính xác vì bản thanh trong chương trình con Delay25ms *cũng khó có thể xây dựng một cách chính xác tuyệt đối. Hơn nữa, còn một (số) lệnh xen thêm vào việc lập/xoá PC4 là Inc và Dec, các lệnh này cũng tiêu tốn một số chu kỳ đồng hồ. Ngay bản thân các lệnh Out, Call cũng tốn thêm một số chu kỳ đồng hồ nữa. Tuy nhiên, trong một số ứng dụng cụ thể nào đó mà không yêu cầu cao về độ chính xác thì ta có thể áp dụng cách trên (phục vụ điều khiển chẳng hạn).*

Ví dụ 4: (Sử dụng 8255 làm việc ở mode 1 – PA: vào.)

Xây dựng mạch mở rộng để vào dữ liệu qua cổng PA của 8255 – mode 1, địa chỉ cơ sở của 8255 là 300h.

Giải:



Giá trị cần đưa ra CWR là: 1011 1111 = 0BFh

(Các bit: CA, MA, B, CB là không quan tâm nên đặt bằng 1)

Đoạn chương trình thực hiện:

;Định nghĩa các hằng địa chỉ cho các cổng để đơn giản trong việc viết lệnh

CONFIG	MA0	MA1	A	CA	MB	B	CB
1	0	0	1	0	0	1	1

```
PA      Equ   300h
PC      Equ   302h
CWR     Equ   303h
  …
Mov     al, 0BFhh       ; Giá trị cần đưa ra CWR để
Out     CWR, al         ; khởi tạo cho 8255 làm việc theo yêu cầu đề bài
Lặp:
In      al, PC
Test    al, 08h         ; PC3 = 0? (And al, 08h)
```

```
Jz      Lặp             ; phải, kiểm tra lại
In      al, PA          ; Không (PC3 = 1), đọc PA
Call    hien_thi        ; Hiển thị kết quả đọc được trong thanh ghi al
Jmp     Lặp             ; Thể hiện công việc như vòng lặp vô hạn
…
```

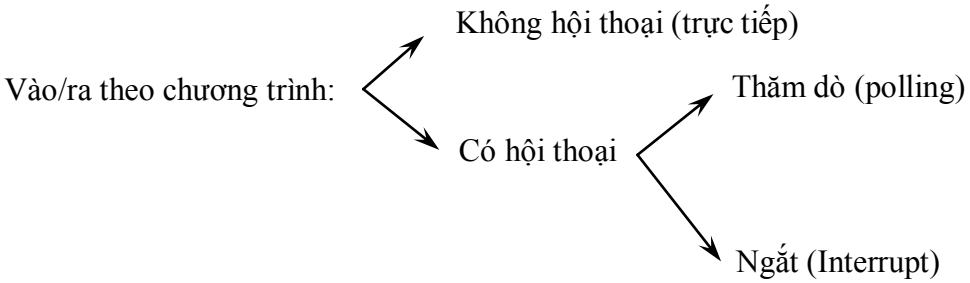
Chú ý: *Giả thiết thủ tục* hien_thi *đã có sẵn, nó thực hiện đổi giá trị trong thanh ghi al ra dãy các ký tự biểu diễn số đó và hiển thị dãy ký tự. Mạch và chương trình sử dụng phương pháp vào/ra dữ liệu bằng cách thăm dò (polling) mà ta sẽ xét trong phần sau.*

Chương 5:
VÀO RA DỮ LIỆU BẰNG CÁCH THĂM DÒ

5.1. Giới thiệu chung về các phương pháp điều khiển vào/ra dữ liệu

Qua chương 4, chúng ta đã tìm hiểu về một số mạch thường dùng cho việc phối ghép CPU với thiết bị ngoại vi. Ta đã tiến hành các phương pháp (phương thức) điều khiển việc trao đổi dữ liệu. Các mạch phối ghép vào/ra đã trình bày trước đây có thể được ứng dụng để phục vụ cho mục đích này.

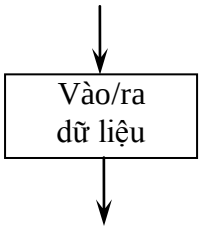
Máy tính trao đổi thông tin (vào/ra dữ liệu) với thieets bị ngoại vi theo một trong 2 chế độ sau:



Truy nhập trực tiếp bộ nhớ (DMA – Direct Memorry Access

Với cách vào/ra dữ liệu theo chương trình có thể phân thành các phương pháp cụ thể hơn là:

Đồng bộ (Không điều kiện - không hội thoại)

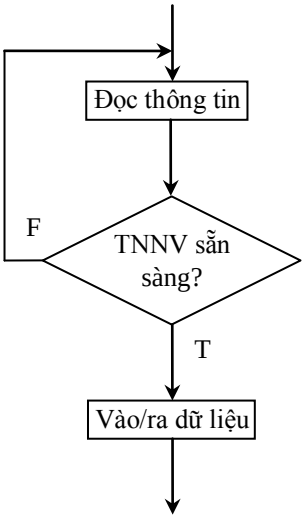


Sau khi đã khởi động (khởi tạo) thiết bị ngoại vi, CPU (và thiết bị ngoại vi) không "quan tâm" tới việc thiết bị ngoại vi có sẵn sàng cho việc trao đổi dữ liệu hay không. Nó luôn thực hiện các lệnh trao đổi dữ liệu (IN, OUT) một cách trực tiếp. Phương pháp này yêu cầu:

- Thiết bị ngoại vi luôn sẵn sàng trao đổi dữ liệu.
- Tốc độ trao đổi dữ liệu của CPU và thiết bị ngoại vi là phù hợp nhau (TBNV có tốc độ làm việc nhanh).

Ví dụ, trong một hệ đo nhiệt độ ghép nối với máy tính, điện áp trên cặp nhiệt điện được khuếch đại lên, tiếp theo là được biến đổi thành tín hiệu số rồi được đọc vào máy tính. Ta thấy rằng tín hiệu này lúc nào cũng sẵn sàng cho CPU đọc vì nhiệt độ là một đại lượng biến đổi rất chậm so với tốc độ của máy tính và cùng với nó, CPU cũng có thể bất kỳ lúc nào đưa số liệu hiển thị số đo đó. Đây gọi là quá trình vào/ra đồng bộ hay không có móc nối (không có hội thoại).

Không đồng bộ (Có điều kiện - có hội thoại)



- **Polling:** CPU và thiết bị ngoại vi chỉ trao đổi dữ liệu khi có tín hiệu móc nối báo sẵn sàng (Ready/Akc) của các phía. Sau khi máy tính khởi động thiết bị ngoại vi (khối ghép nối), máy tính luôn chờ và kiểm tra trạng thái sẵn sàng của thiết bị ngoại vi gồm các bước:
  (1) Đọc thông tin về trạng thái sẵn sàng của thiết bi ngoại vi
  (2) Kiểm tra: Nếu thiết bị ngoại vi sẵn sàng thì trao đổi dữ liệu, ngược lại thì về bước (1) để kiểm tra lại.

Phương pháp này được dùng khi tốc độ trao đổi dữ liệu của các bên (CPU và thiết bị ngoại vi) rất không bằng nhau và nó có một số đặc điểm sau:

Việc trao đổi thông tin là tin cậy vì chỉ trao đổi khi thiết bị ngoại vi sẵn sàng.

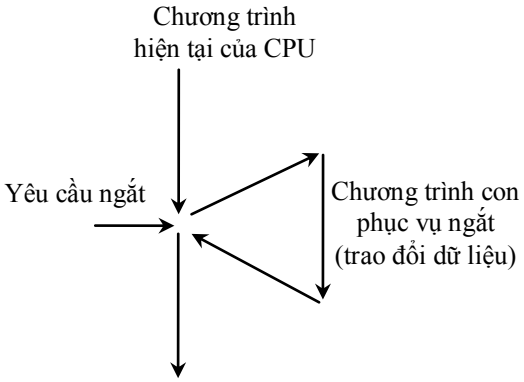
Tốn thời gian CPU vì phải kiểm tra trạng thái sẵn sàng của thiết bị ngoại vi (việc kiểm tra này co CPU đảm nhiệm) nên giảm hiệu suất của hệ thống.

Phù hợp với những hệ thống không đòi hỏi cao về tốc độ trao đổi dữ liệu, hệ thống có ít thiết bị ngoại vi.

Chúng ta sẽ xem xét cụ thể phương pháp vào/ra này trong phần sau.

- **Vào/ra dữ liệu điều khiển bằng ngắt chương trình (ngắt bộ vi xử lý - Interrupt):**

Bình thường máy tính thực hiện một chương trình (công việc) nào đó. Khi thiết bị ngoại vi có yêu cầu trao đổi dữ liệu, nó sẽ gửi tín hiệu yêu cầu ngắt CPU dừng công việc hiện tại, phục vụ cho trao đổi dữ liệu thông qua tín hiệu yêu cầu ngắt IRQ (Interrupt Request) tác động vào chân INTR (chân tiếp nhận yêu cầu ngắt) của CPU. CPU nhận được yêu cầu ngắt, nếu chấp nhận nó sẽ đưa ra xung INTA xác nhận tới thiết bị ngoại vi, sau đó CPU tìm chương trình con phục vụ ngắt tương ứng số hiệu ngắt và thực hiện nó. Đó chính là chương trình con thực hiện trao đổi (vào/ra) dữ liệu do thiết bị ngoại vi yêu cầu. Khi trao đổi xong (ISR – Interrupt Service Routine) kết thúc thì CPU tiếp tục công việc (chương trình) đã bị gián đoạn.



- **Vào/ra dữ liệu điều khiển bằng phần cứng phụ để thâm nhập trực tiếp bộ nhớ (DMA - Direct Memorry Access):**

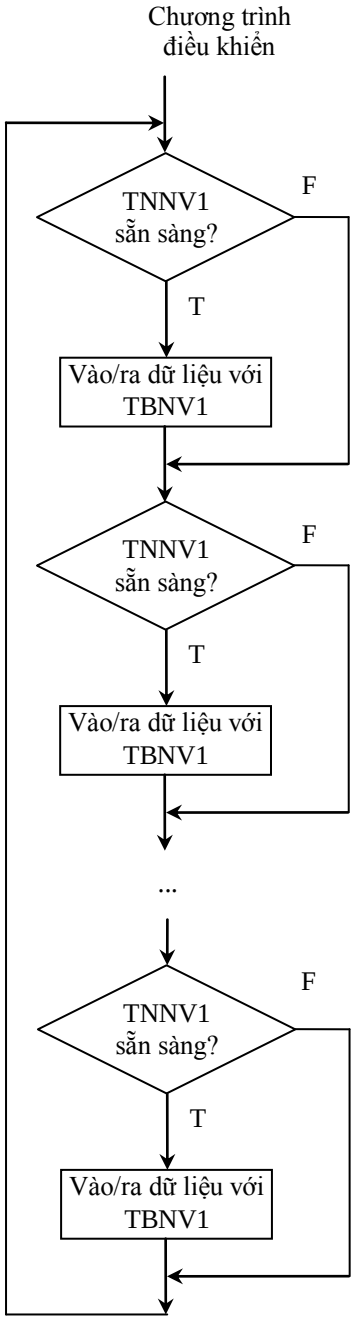
Trong các phương pháp vào/ra dữ liệu bằng chương trình kể trên, dữ liệu phải được chuyển qua lại từ bộ nhớ đến CPU rồi đến thiết bị ngoại vi hoặc ngược lại bằng việc thực hiện từng lệnh (MOV, IN hoặc OUT) của CPU với sự tham gia của các thanh ghi. Dữ liệu của mỗi lần vận chuyển là byte hoặc word (2 byte), tốc độ trao đổi dữ liệu phụ thuộc rất nhiều vào tốc độ thực hiện các lệnh trao đổi dữ liệu kể trên. Nói chung, tốc độ trao đổi dữ liệu là không thể nhanh được. Với các thiết bị làm việc với bộ nhớ khối như màn hình, ổ đĩa, ... yêu cầu trao đổi cả mảng dữ liệu thì phương pháp vào/ra dữ liệu bằng chương trình là không phù hợp. Khi đó người ta nghĩ đến việc điều khiển dữ liệu vào/ra trực tiếp từ bộ nhớ đến thiết bị ngoại vi hoặc ngược lại mà không thông qua CPU bằng những lệnh trao đổi dữ liệu như MOV, IN hoặc OUT. Đó là phương pháp vào/ra dữ liệu bằng cách truy nhập trực tiếp bộ nhớ (DMA – Direcr Memory Access). Trong trường hợp này CPU trao quyền điều khiển cho một mạch phần cứng phụ điều khiển việc vào/ra dữ liệu, đó là DMAC – DMA Controller. Chi tiết về phương pháp này sẽ được trình bày kỹ trong phần sau.

Qua việc trình bày sơ lược về các phương pháp điều khiển việc vào/ra dữ liệu trên ta thấy mỗi phương pháp có những đặc điểm khác nhau (ưu nhược điểm), chúng được ứng dụng phù hợp trong những hoàn cảch cụ thể khác nhau. Một trong những phương pháp điều khiển đơn giản nhất mà chúng ta xem xét trong chương này là phương pháp trao đổi không đồng bộ (có hội thoại) hay gọi là phương pháp thăm dò trạng thái sẵn sàng của thiết bị ngoại vi (polling) trước khi thực hiện vào/ra dữ liệu.

5.2. Vào/ra dữ liệu bằng cách thăm dò trạng thái sẵn sàng của thiết bị

Phương pháp vào/ra dữ liệu bằng cách thăm dò trạng thái sẵn sàng của thiết bị ngoại vi là phương pháp điều khiển vào/ra dữ liệu bằng chương trình – có hội thoại. Tín hiệu hội thoại (handshaking signal) ở đây là tín hiệu báo trạng thái sẵn sàng của thiết bị ngoại vi.

Việc điều khiển vào/ra dữ liệu sẽ trở nên rất đơn giản nếu thiết bị ngoại vi lúc nào cũng sẵn sàng chờ làm việc (trao đổi dữ liệu) với CPU – Như đã đề cập đến trong phần giới thiệu – đó là phương pháp vào/ra dữ liệu đồng bộ (trực tiếp – không hội thoại). Tuy nhiên trong thực tế không phải lúc nào CPU cũng làm việc với các thiết bị ngoại vi "liên tục sẵn sàng" như trên, vì tốc độ làm việc của thiết bị ngoại vi và CPU là rất chênh lệch. Thông thường khi CPU muốn trao đổi dữ liệu với một thiết bị ngoại vi nào đó, thông qua chương trình nó liên tục kiểm tra trạng thái sẵn sàng của thiết bị ngoại để xem có yêu cầu trao đổi dữ liệu (yêu cầu phục vụ) hay không? Đến khi có một thiết bị ngoại vi nào đó có yêu cầu trao đổi dữ liệu (tức là có tín hiệu báo trạng thái sẵn sàng – readly) thì chương trình sẽ chuyển sang thực hiện việc trao đổi dữ liệu (công việc trao đổi này có thể là các lệnh ngay trong chương trình chính hoặc là một chương trình con phục vụ – hoặc ISR) rồi lại tiếp tục kiểm tra thiết bị ngoại vi kế tiếp. Quá trình này được thực hiện tiếp tục đến thiết bị ngoại vi cuối cùng rồi lại được lặp lại từ đầu. Vậy, nếu làm việc theo phương pháp thăm dò thì thông thường CPU phải được dành riêng cho công việc trao đổi dữ liệu vì nó phải liên tục kiểm tra trạng thái sẵn sàng của thiết bị ngoại vi thông qua các tín hiệu móc nối. Các tín hiệu này được lấy từ mạch phối ghép, do người thiết kế mạch tạo ra để cho chương trình có thể kiểm tra nó. Việc này được tính toán đồng bộ khi xây dựng mạch và chương trình điều khiển.



Hình vẽ: Lưu đồ điều khiển của Phương pháp thăm dò trạng thái sẵn sàng

Nói chung, phương pháp này đơn giản trong tổ chức phần cứng và phần mềm nhưng gặp phải nhược điểm là khi số lượng thiết bị ngoại vi tăng lên thì sẽ làm lãng phí thời gian CPU cho việc kiểm tra trạng thái sẵn sàng và khả năng đáp ứng tức thời với phục vụ của CPU là rất thấp.

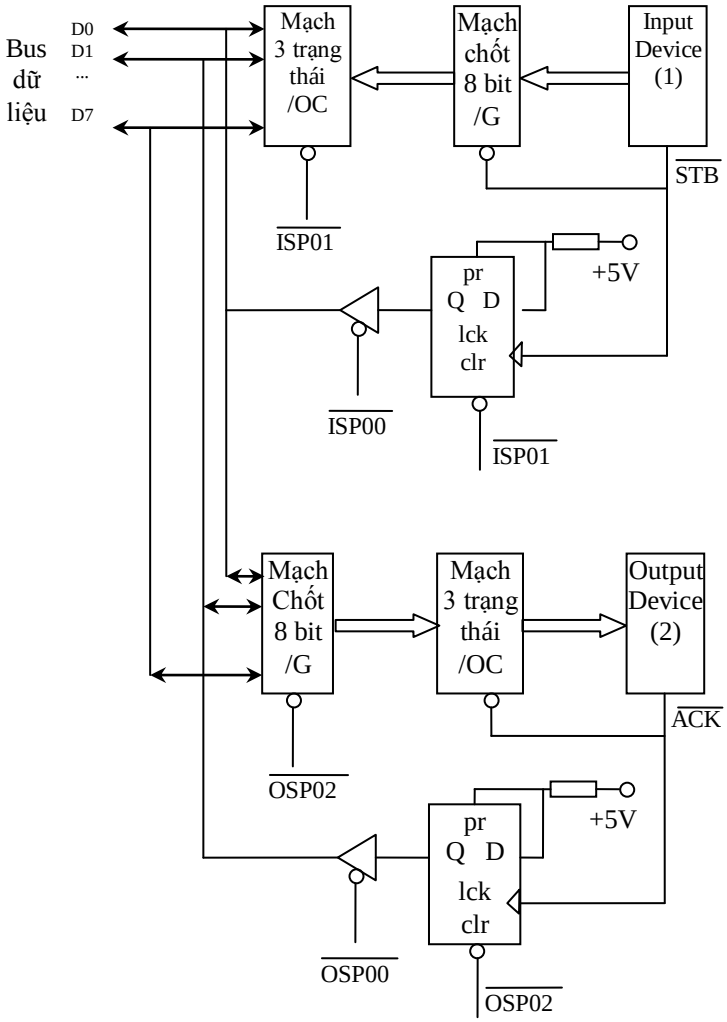
Hình vẽ cho thấy lưu đồ của chương trình điều khiển theo phương pháp thăm dò khi số lượng số thiết bị ngoại vi > 1.

Nếu thông tin được phát hoặc nhận từ một thiết bị ngoại vi nào đó sẽ thiết lập một cờ trạng thái thích hợp (thông thường là bằng một mạch lật trạng thái bởi xung STB của thiết bị ngoại vi). Bộ vi xử lý sẽ kiểm tra cờ trạng thái này và sẽ thực hiện modul chương trình trao đổi dữ liệu khi cờ trạng thái báo rằng thiết bị ngoại vi sẵn sàng cho việc trao đổi dữ liệu. Nó như một tín hiệu yêu cầu phục vụ (thụ động) cho việc vào/ra dữ liệu.

Sau đây là một ví dụ về cách tạo tín hiệu móc nối trong tổ chức phần cứng và lưu đồ thuật toán (chương trình điều khiển) dùng cho việc trao đổi dữ liệu giữa CPU và thiết bị ngoại vi.

Để đơn giản, trong ví dụ này ta giả thiết CPU chỉ làm việc với 1 thiết bị ngoại vi và và 1 thiết bị ngoại vi ra. Việc tổ chức phối ghép phần cứng được thực hiện trên các vi mạch (IC) cỡ vừa để dễ theo dõi các tín hiệu.

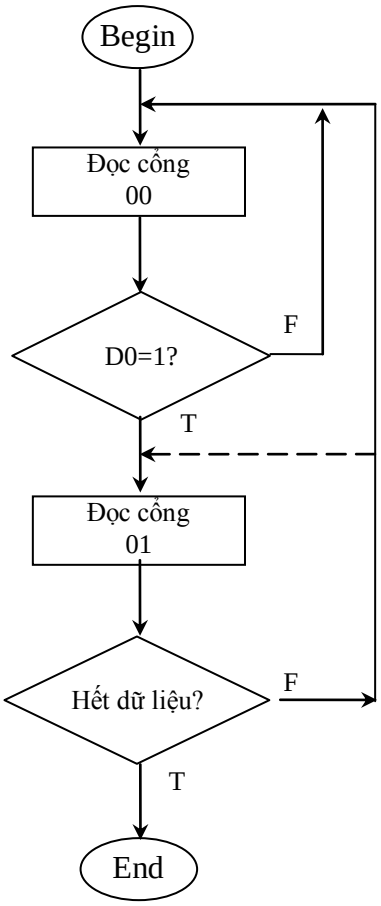
Một cổng vào có địa chỉ 00 được dùng để đọc trạng thái sẵn sàng của thiết bị ngoại vi số 1 và số 2. (TB1: vào; TB2: ra). Tín hiệu báo sẵn sàng của thiết bị ngoại vi số 1 (có địa chỉ 01) được đặt vào bit D0 và tín hiệu báo sẵn sàng của thiết bị ngoại vi số 2 (có địa chỉ 02) được đặt vào bit D1 của bus dữ liệu. Các bit tín hiệu này sẽ có giá trị bằng 1 khi thiết bị ngoại vi tương ứng ở trạng thái sẵn sàng trao đổi dữ liệu với CPU và chúng sẽ được đưa vào bus dữ liệu khi CPU đọc nó bằng lệnh đọc cổng có địa chỉ 00. Chương trình điều khiển trao đổi dữ liệu sẽ kiểm tra các bit báo sẵn sàng này và có các đáp ứng thích hợp.



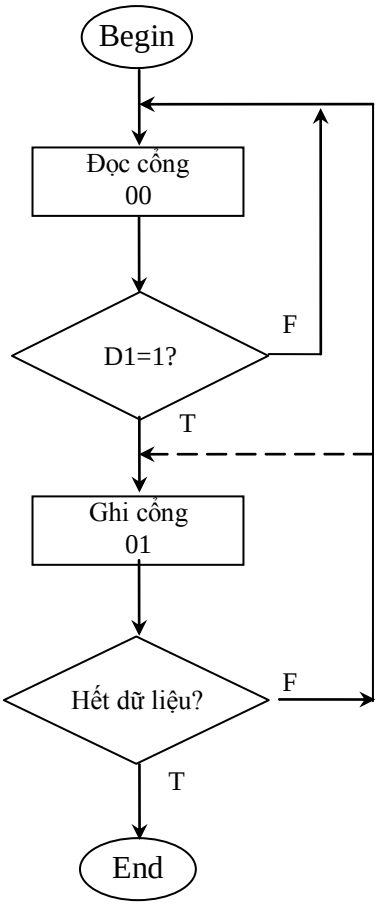
Hình vẽ: Sơ đồ mạch phần cứng tạo tín hiệu móc nối

Cụ thể hoạt động của phần mạch vào dữ liệu như sau: Khi thiết bị vào (TB1) có (1 byte) dữ liệu cần (trao đổi) đưa vào hệ thống, nó đưa ra xung STB để cho phép mạch chốt 8 bit chốt lấy dữ liệu đó đồng thời kích cho mạch lật D (mạch tạo tín hiệu sẵn sàng) làm việc => đầu ra Q = 1. Bộ vi xử lý (CPU) thông qua lệnh đọc cổng 00 để thăm dò trạng thái sẵn sàng của thiết bị ngoại vi số 1 thông qua bit D0. Khi thấy D0 = 1, nó đọc 1 byte dữ liệu vào đồng thời xoá luôn Q (Q = 0, thiết bị ngoại vi không sẵn sàng) để chuẩn bị lần đọc byte dữ liệu khác.

Tương tự như vậy ta có thể thấy được sự hoạt động của phần mạch ra dữ liệu – thiết bị ngoại vi số 2.



(a) Lưu đồ đọc dữ liệu từ cổng 01



(a) Lưu đồ ghi dữ liệu vào 02

Hình vẽ: Lưu đồ điều khiển vào/ra dữ liệu

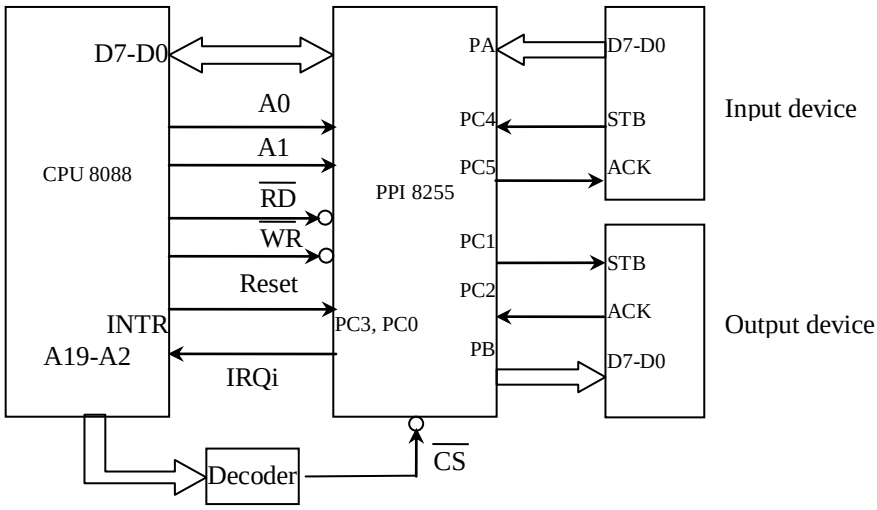
Yêu cầu: Lập trình theo lưu đồ (a) – đường nét liền để thực hiện việc đọc vào một byte dữ liệu mỗi khi cổng 01 báo sẵn sàng rồi hiển thị byte dữ liệu đó. Giả thiết có 100 byte dữ liệu phải đọc và có sẵn thủ tục thực hiện hiển thị giá trị trong thanh ghi al lên màn hình có tên là hien_thi.

```
    …
    Mov     cx, 100         ; Số byte dữ liệu cân đọc trong cx
Lặp:
    In      al, 0           ; Đọc cổng 00
    Test    al, 1           ; D0 = 1?
    Jz      Lặp             ; không, đọc lại cổng 00
    In      al, 1           ; phải, đọc cổng 01
    Call    Hien_thi        ; hiển thị kết quả
    Loop    Lặp             ; chưa hết dữ liệu, quay lại
Ra:
    …
```

Trong trường hợp thiết bị ngoại vi vào/ra hoạt động theo cách khác: mỗi khi cờ báo trạng thái sẵn sàng, cho phép CPU đọc/ghi nhiều byte dữ liệu cùng lúc thì ta có nhánh đi theo đường nét đứt trên lưu đồ. Khi này ta cần thiết sửa đổi cả tổ chức phần cứng và chương trình để hệ thống hoạt động chính xác.

Trở về ví dụ 4 trong chương trước ta thấy: tổ chức phần cứng và chương trình điều khiển vào/ra là theo phương pháp này (polling – thăm dò trạng thái sẵn sàng của thiết bị ngoại vi). ở đó ta sử dụng vi mạch cổng chuyên dụng PPI 8255 làm việc trong chế độ 1 (mode 1) và bản thân vi mạch này có khả năng tạo ra các tín hiệu móc nối được cả với CPU và thiết bị ngoại vi. Việc đọc (thăm dò) đơn giản chỉ là đọc bit PC3 của cổng PC (theo ví dụ đó). Tín hiệu của bit này chính là tín hiệu INTR. Nếu ta không kiểm tra bit này bằng cách đọc cổng PC mà nối thẳng nó tới chân tiếp nhận yêu cầu ngắt của CPU là chân INTR thì ta có một phương pháp phối ghép khác để vào/ra dữ liệu – đó là điều khiển vào/ra dữ liệu bằng cách ngắt CPU. Phương pháp này sẽ được trình bày chi tiết trong chương sau.

Nếu kiểm tra PC3, PC0 (là tín hiệu báo sẵn sàng tương ứng cho các cổng PA và PB), bằng lệnh đọc cổng PC trong chương trình thì ta có phương pháp vào/ra dữ liệu là thăm dò trạng thái sẵn sàng của thiết bị ngoại vi.



Sơ đồ ghép nối dùng PPI 8255 - Mode 1

Chương 6
NGẮT VÀ XỬ LÝ NGẮT TRONG VXL 8088

6.1. Sự cần thiết phải ngắt CPU

Như đã biết, với cách tổ chức trao đổi dữ liệu với thiết bị ngoại vi bằng cách thăm dò trạng thái sẵn sàng của thiết bị ngoại vi, trước khi thực hiện bất kỳ một lệnh (hay một nhóm lệnh) trao đổi dữ liệu thì CPU phải dành toàn bộ thời gian vào việc kiểm tra (thăm dò) để xác định trạng thái sẵn sàng làm việc của thiết bị ngoại vi. Trong một hệ vi xử lý với cách làm việc như vậy, thông thương bộ vi xử lý (CPU) chủ yếu là được dành cho việc vào/ra dữ liệu và thực hiện một vài xử lý liên quan.

Nhưng thực tế thì sức mạnh của CPU trong một hệ vi xử lý là rất lớn, nó làm việc với tốc độ rất cao và tập lệnh phong phú (khả năng xử lý dữ liệu lớn) mà công việc vào/ra dữ liệu thì không tốn nhiều thời gian (thời gian thực hiện các lệnh vận chuyển dữ liệu). Để tận dụng khả năng làm việc của CPU để làm thêm được nhiều công việc khác nữa, người ta mong muốn: CPU không tốn nhiều thời gian vào việc thăm dò trạng thái sẵn sàng của thiết bị ngoại vi, chỉ khi nào thiết bị ngoại vi sẵn sàng và cần trao đổi dữ liệu với hệ thống (CPU) thì thiết bị ngoại vi chủ động yêu cầu (thông báo) CPU tạm dừng công việc hiện tại để phục vụ cho việc trao đổi dữ liệu. Sau khi hoàn thanh công việc trao đổi dữ liệu thì CPU quay về thực hiện tiếp công việc đang thực hiện (công việc bị gián đoạn). Cách làm việc theo kiểu này gọi là ngắt CPU (gián đoạn sự hoạt động của CPU) để thực hiện trao đổi dữ liệu. Treen cơ sở như vậy, "ngắt" được mở rộng cho hầu hết các ứng dụng khác nhằm độc lập hoá với công việc của CPU và tăng thêm hiệu suất của hệ thống (như các phục vụ của DOS và của BIOS).

Để có thể thực hiện được phương pháp ngắt CPU cho vào/ra dữ liệu thì ta phải có cách tổ chức hệ thống phần cứng và phần mềm sao cho có thể tận dụng được khả năng thực hiện các chương trình con phục vụ ngắt (ISR – Interrupt Service Routine) tại các địa chỉ xác định của CPU. Khi nghiên cứu các chân tín hiệu của 8088, ta thấy vi mạch này có các chân tiếp nhận yêu cầu ngắt che được INTR và không che được NMI, chính các chân này sẽ được sử dụng vào việc đưa các yêu cầu ngắt từ bên ngoài tới CPU 8088.

6.2. Ngắt trong vi xử lý 8088

6.2.1. Các loại ngắt trong hệ 8088

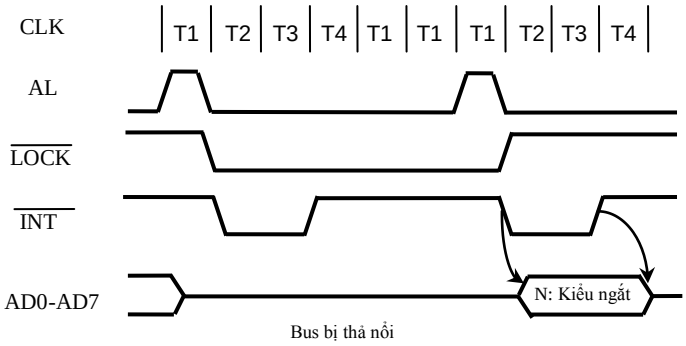
Trong hệ vi xử lý có bộ vi xử lý 8088 (và 80x86 family) có thể phân loại các ngắt theo nguyên nhân gây ngắt CPU thành 3 nhóm:

- Ngắt cứng: Đây là các yêu cầu ngắt CPU do mạch ngoài (thiết bị ngoại vi hoặc các vi mạch điều khiển khác) tác động đến chân INTR và NMI của 8088.

Ngắt cứng NMI (Non Maskable Interrupt) là yêu cầu ngắt không che được tương ứng với ngắt mềm INT 2. Trong hệ thống, các nguyên nhân gây lỗi như lỗi bộ nhớ, chẵn lẻ, sự cố hệ thống như sụt điện áp, cháy máy, … sẽ được chuyển thành tín hiệu báo về chân NMI của bộ vi xử lý. Các lệnh như CLI (xoá cờ IF) STI (lập cờ IF) không ảnh hưởng tới sự nhận biết của tín hiệu yêu cầu ngắt NMI.

Ngắt cứng INTR là yêu cầu ngắt che được. Các lệnh CLI, STI có ảnh hưởng trực tiếp tới trạng thái của cờ IF trong thanh ghi cờ của bộ vi xử lý, tức là ảnh hưởng tới việc CPU có thể nhận biết được các yêu cầu ngắt tại chân này hay không. Cụ thể, nếu IF=1 thì CPU nhận biết được các yêu cầu ngắt tác động chân INTR của nó; IF=0, ngược lại. Yêu cầu ngắt tại

chân INTR có thể có kiểu ngắt N nằm trong khoảng 00h – FFh, kiểu ngắt này phải được đưa vào bus dữ liệu của hệ thống để CPU có thể đọc được khi có xung INTA (Interrupt Acknowledge) trong chu kỳ trả lời chấp nhận ngắt.

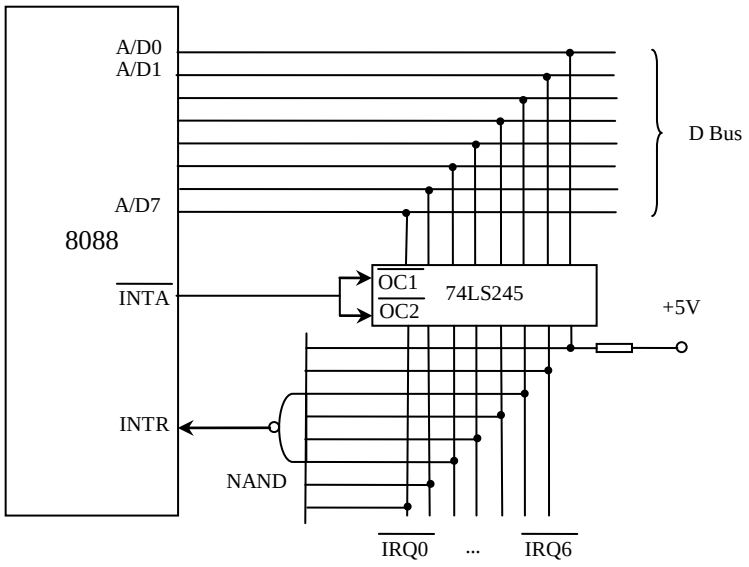


Hình vẽ: Chu kỳ trả lời ngắt của 8088

- Ngắt mềm: Khi CPU thực hiện các lệnh ngắt dạng INT N (lệnh gọi ngắt trong assembly) – trong đó N là số hiệu (kiểu) ngắt nằm trong khoảng 00h – FFh. Dựa vào số hiệu ngắt, theo cách thiết kế mà CPU sẽ tìm được chương trình con (phục vụ) ngắt để thực hiện thông qua bảng vector ngắt (sẽ nói trong phần sau).
- Ngắt ngoại lệ: Đây là các ngắt xuất hiện do các lỗi sinh ra trong quá trình hoạt động của CPU. Ví dụ như chia cho 0, tràn khi tính toán, … và một số lỗi khi tính toán với số thực dấu chấm động (trong các bộ vi xử lý tiên tiến).

Các yêu cầu ngắt, cuối cùng cũng dẫn đến: CPU thực hiện một đoạn chương trình (chương trình con) – nó được coi như những "điều kiện cho rẽ nhánh" sự hoạt động của CPU. Các yêu cầu này được CPU kiểm tra thường xuyên tại chu kỳ đồng hồ cuối cùng của mỗi lệnh.

Để đưa số hiệu ngắt vào bus dữ liệu khi có tín hiệu yêu cầu ngắt tác động vào CPU (INTR hoặc NMI) của một thiết bị ngoại vi nào đó (đay là ngắt cứng), giả thiết trong một thời điểm nhất định chỉ có một yêu cầu ngắt IRQi (Interrupt ReQuest i) được tác động và khi đó CPU sẽ tiếp nhận được yêu cầu ngắt và đọc số hiệu ngắt N. Ta có thể sử dụng sơ đồ đơn giản sau:



Hình vẽ: Một cách đơn giản để đưa số hiệu ngắt N vào bus dữ liệu

AD7 IRQ6 IRQ5 IRQ4 IRQ3 IRQ2 IRQ1 IRQ0 N

1	1	1	1	1	1	1	0	FEh (245)
1	1	1	1	1	1	0	1	FDh (253)
1	1	1	1	1	0	1	1	FBh (251)
1	1	1	1	0	1	1	1	F7h (247)
1	1	1	0	1	1	1	1	EFh (239)
1	1	0	1	1	1	1	1	DFh (223)
1	0	1	1	1	1	1	1	BFh (191)

Bảng quan hệ giữa IRQi và số hiệu ngắt N

Tại một thời điểm nếu có 1 tín hiệu yêu cầu ngắt IRQi nào đó tác động (mức thấp – low) thì đầu ra của mạch NAND sẽ có xung yêu cầu ngắt đến CPU. Tín hiệu IRQi được đồng thời đưa qua mạch khuếch đại đệm để tạo ra số hiệu ngắt tương ứng, số hiệu ngắt này sẽ được CPU đọc vào khi nó đưa tín hiệu trả lời /INTA (xung thứ hai). Trong trường hợp có 2 hay nhiều hơn các yêu cầu ngắt tác động thì với mạch ví dụ trên không thể đáp ứng được vì vỡi một xung yêu cầu ngắt nhận được từ chân INTR nhưng số hiệu ngắt thì không thuộc (không tương ứng) yêu cầu ngắt nào cả! Vì vậy cần có sự xử lý trước tình huống đó. Sẽ được nói rõ trong phần sau.

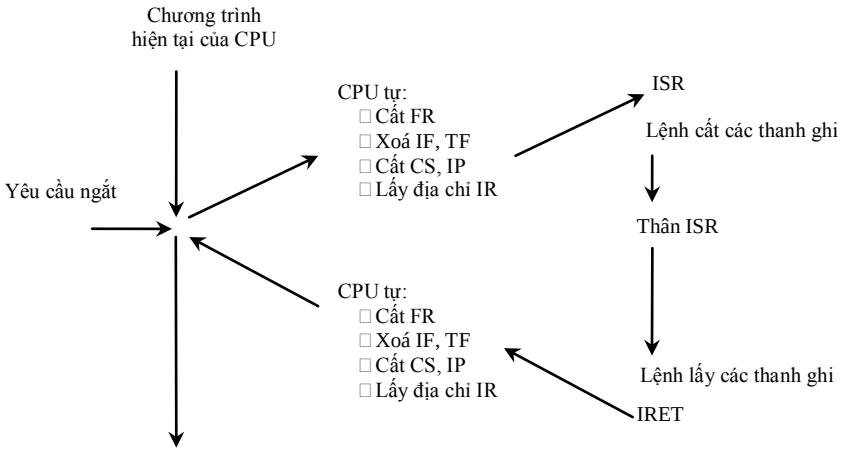
6.2.2. Đáp ứng của CPU khi có yêu cầu ngắt

Khi có yêu cầu ngắt với số hiệu ngắt N tác động đến chân tiếp nhận yêu cầu ngắt của CPU và nếu yêu cầu ngắt đó được chấp nhận (với những yêu cầu ngắt che được) thì CPU sẽ thực hiện dãy công việc sau:

- SP <= SP-2; {SP} <= FR (*chỉ ra đỉnh mới của Stack, cất thanh ghi cờ vào stack, trong đó {SP} là ô nhớ do SP lưu giữ địa chỉ (trỏ tới)*).
- IF <= 0; TF <= 0; (*cấm các ngắt che được, chạy suốt*).
- SP <= SP-2; {SP} <= CS; (*chỉ ra đỉnh mới của Stack, cất địa chỉ segment của địa chỉ trở về vào stack*).
- SP <= SP-2; {SP} <= IP; (*chỉ ra đỉnh mới của Stack, cất địa chỉ offset của địa chỉ trở về vào stack*).
- IP <= {N*4}; CS <= {N*4+2}; (*lấy lệnh tại địa chỉ mới của chương trình con phục vụ ngắt kiểu N (số hiệu ngắt N) tương ứng trong bảng vector ngắt*).
- Tại cuối chương trình con phục vụ ngắt, khi gặp lệnh IRET, CPU sẽ thực hiện dãy các công việc trên theo thứ tự ngược lại, cụ thể:
  - {SP} => IP; SP <= SP+2
  - {SP} => CS; SP <= SP+2
  - {SP} => FR; SP <= SP+2

(Bộ vi xử lý quay lại thực hiện chương trình chính tại địa chỉ với giá trị cũ và thanh ghi cờ được lấy ra từ stakc *)*

Trong thực tế, các ngắt mềm INT N đã bao trùm các loại ngắt khác nhau bởi vì Intel đã quy định một số kiểu ngắt đặc biệt được xếp vào đầu dãy ngắt mềm như sau:



Hình vẽ: Minh hoạ về mặt cấu trúc chương trình có ngắt xảy ra và sự liên hệ giữa chương trình chính (CTC) và chương trình con phục vụ ngắt (ISR)

- INT 0: Ngắt mềm do phép chia cho 0 gây ra.
- INT 1: Ngắt mềm để chạy từng lệnh với trường hợp cờ TF=1.
- INT 2: Ngắt cứng do tín hiệu tích cực tại chân NMI gây ra.
- INT 3: Ngắt mềm để đặt điểm dừng của chương trình tại một địa chỉ nào đó.
- INT 4: (Hoặc lệnh INTO) là ngắt mềm ứng với trường hợp tràn (OF=1).

Các kiểu ngắt khác còn lại được dành cho Intel và người sử dụng (IBM không hoàn toàn tuân thủ các quy định này khi chế tạo các máy tính PC/XT, PC/AT):

- INT 5 – INT 1Fh: Dành riêng cho Intel trong các bộ vi xử lý cao cấp.
- INT 20h – INT FFh: Dành cho người sử dụng (trong đó hệ điều hành và BIOS đã sử dụng một phần).

Các kiểu ngắt N (trong INT N) đều tương ứng với các địa chỉ xác định của ISR mà ta có thể tra (lấy – hoặc đọc trực tiếp) trong bảng vector ngắt. Intel quy định bảng này nằm trong RAM bắt đầu từ địa chỉ 00000h và kéo dài 1 KB (kích thước này là vì: 8088 và 80x86 có 256 kiểu ngắt, mỗi kiểu ngắt ứng với một vector ngắt, một vector ngắt cần 4 byte để chứa địa chỉ đầy đủ cho CS và IP của chương trình con phục vụ ngắt, 2 byte để lưu CS, 2 byte khác để lưu IP nên: 256 (byte) x 4 = 1024 (byte) = 1 KB).

6.2.3. Xử lý ưu tiên ngắt

Như đã đề cập đến, vấn đề trở nên phức tạp là tại cùng một thời điểm mà có nhiều tín hiệu yêu cầu ngắt tác động đến CPU (thuộc các kiểu ngắt khác nhau). Khi đó CPU sẽ xử lý như thế nào? Thực hiện (phục vụ) chương trình con phục vụ ngắt nào? Thực ra CPU xử lý các yêu cầu ngắt theo mức ưu tiên (theo thiết kế) với nguyên tắc ngắt nào có mức ưu tiên cao nhất sẽ được CPU nhận biết và phục vụ trước.

Ngay từ khi thiết kế và chế tạo (thường gọi là ngầm định) CPU 8088 có khả năng phân biệt các mức ưu tiên khác nhau cho các loại ngắt theo thứ tự từ cao xuống thấp như sau:

Các ngắt	Mức ưu tiên
Ngắt nội bộ: INT 0 (phép chia 0), INT N, INTO	0: Cao nhất
Ngắt NMI	1
Ngắt INTR	2
Ngắt để chạy từng lệnh (INT 1)	3: Thấp nhất

Để thấy rõ sự hoạt động của CPU trong cơ chế ngắt ưu tiên này, ta có thể lấy môtj ví dụ cụ thể như sau: Giả sử tại một thời điểm nào đó, khi CPU (ở trạng thái cho phép ngắt với cờ IF=1) đang thực hiện phép chia và có lỗi xảy ra do số chia bằng 0, tai thời điểm này CPU cũng nhận được yêu cầu ngắt từ đầu bào INTR. Theo thứ tự ưu tiên ngầm định trong việc xử lý ngắt của 8088 thì INT o có mức ưu tiên cao hơn ngắt che được INTR nên CPU sẽ thực hiện ISR tương ứng INT 0 để đáp ứng với lỗi đặc biệt do phép chia 0 gây ra và cờ IF được xoá về 0 => yêu cầu ngắt INTR sẽ (tự động) bị cấm cho tới khi ISR tương ứng INT 0 kết thúc và trở về nhờ lệnh IRET, cờ IF được khôi phục trở lại (toàn bộ thanh ghi cờ). Tiếp đó CPU sẽ đáp ứng yêu cầu ngắt INTR bằng cách thực hiện ISR tương ứng INTR đó.

6.2.4. Mạch điều khiển ngắt ưu tiên PPI 8259A

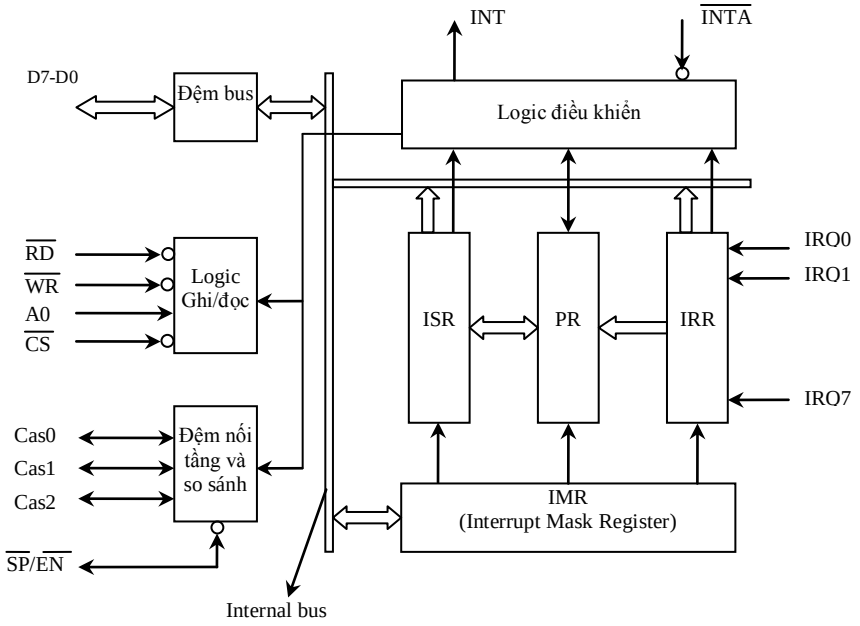
Khi vào/ra dữ liệu với thiết bị ngoại vi theo nguyên tắc ngắt (ngắt cứng), tín hiệu yêu cầu ngắt (yêu cầu phục vụ ngắt) hay tín hiệu sẵn sàng (ready) từ phía thiết bị ngoại vi nối với lối vào tiếp nhận yêu cầu ngắt INTR của CPU. Theo đó (mà) máy tính có thể thực hiện được các công việc khác cho tới khi nó bị ngắt bởi tín hiệu này từ một thiết bị ngoại vi nào đó. Chương trình con phục vụ ngắt tương ứng sẽ nhập hoặc xuất dữ liệu trong vòng một vài micro giây rồi kết thúc để CPU quay trở về chương trình (công việc) đã bị gián đoạn. Như vậy, các thao tác nhập/xuất dữ liệu chỉ chiếm một phần nhỏ thời gian của bộ vi xử lý. Có nhiều ứng dụng được sử dụng ngắt cứng như việc đọc vào mã ASCII các ký tự của các phím trên bàn phím.

- Đếm số ngắt từ một bộ định thời để tạo thành một đồng hồ thời gian thực theo giờ, phút, giây.
- Phát hiện các tình huống khẩn cấp hay kiểm tra công việc nào đó dựa trên cơ chế ngắt, …

Về nguyên tắc, mỗi ứng dụng này yêu cầu một lối vào ngắt riêng biệt (cụ thể) để CPU nhận biết đúng ứng dụng đó, nhưng với bộ vi xử lý 8088 và 80x86 nói chung chỉ có 2 chân (lối vào) tiếp nhận yêu cầu ngắt là INTR và NMI. Nếu (hầu hết) dành lối vào NMI cho ngắt khi mất nguồn nuôi (sụt áp) thì còn lại duy nhất một lối vào INTR cho tất cả các ứng dụng còn lại (Các ngắt INTR chỉ là một mức ưu tiên trong 4 mức ưu tiên ngầm định mà 8088 có thể xử lý được). Do đó, khi có nhiều yêu cầu ngắt che được của các thiết bị ngoại vi khác nhau cùng gửi đến CPU yêu cầu phục vụ (trao đổi dữ liệu) thì CPU không thể nhận biết được yêu cầu đó là của thiết bị nào. Trong trường hợp đó, phải sử dụng một mạch bổ trợ để "dồn" các tín hiệu ngắt cứng từ các thiết bị ngoại vi khác nhau vào một lối vào duy nhất INTR của bộ vi xử lý, đồng thời có thể xử lý ưu tiên cho các tín hiệu yêu cầu ngắt đó gọi là bộ điều khiển ngắt ưu tiên: PIC – Priority Interrupt Controller.

Trong các máy tính IBM PC, PC XT/AT, … (và các hệ vi xử lý khác nói chung) thường dùng vi mạch PIC có sẵn 8259. Đó là một vi mạch cỡ lớn lập trình được, nó có thể xử lý được 8 yêu cầu ngắt đồng thời gửi đến với 8 mức ưu tiên khác nhau để tạo ra một yêu cầu ngắt duy nhất (có mức ưu tiên cao nhất trong số đó) đưa tới chân INTR của CPU (là yêu cầu ngắt che được ứng với một thiết bị nào đó.

Khi hệ thống có số lượng thiết bị ngoại vi lơn hơn 8, cần mở rộng khả năng tiếp nhận và xử lý ngắt thì 8259 cho phép nối tầng. Nếu nối tầng một mạch 8259 chủ và 8 mạch 8259 thợ thì có thể xử lý được 64 ngắt đồng thời gửi đến với 64 mức ưu tiên khác nhau (Với các máy PC XT/AT: thường nối tầng một 8259 chủ và một 8259 thợ).



Hình vẽ: Sơ đồ khối và sơ đồ đóng vỏ DIP của PIC 8259A

Một số ký hiệu trên sơ đồ:

- IRQ0 – IRQ7 (IRQi) – Interrupt ReQuest: các yêu cầu ngắt.
- IRR – Interrupt Request Register: thanh ghi yêu cầu ngắt.
- PR – Priority Resolver: bộ xử lý ưu tiên.
- SP/EN – Slave Program/ENable buffer: lập trình thành mạch thợ/mở đệm bus dữ liệu.
- ISR – In Service Register: thanh ghi yêu cầu ngắt đang được phục vụ.
- Cas0 – Cas2: tín hiệu nối tầng giữa các PIC với nhau.

Các khối chức năng chính của 8259A bao gồm

(Các) yêu cầu ngắt từ các thiết bị ngoại vi gửi tới IRQi được xử lý bởi 3 thanh ghi 8 bit, mỗi bit tương ứng với một yêu cầu ngắt IRQi (i=0-7).

(1) IMR (Interrupt Mask Register – Thanh ghi che ngắt): Thanh ghi này dùng để cấm (che) hoặc cho phép (không che) từng lối vào yêu cầu ngắt riêng biệt. Mỗi bit của nó tương ứng với một lối vào có cùng chỉ số i. Để cho phép một lối vào ngắt, phải gửi đi (ghi) một từ lệnh có bit tương ứng lối vào ấy bằng 0.

(2) IRR (Interrupt Request Register – Thanh ghi yêu cầu ngắt): Thanh ghi này có nhiệm vụ ghi nhớ để theo dõi các yêu cầu ngắt đang yêu cầu phục vụ (có tại các IRQi). Tín hiệu yêu cầu ngắt có tại lối vào nào thì bit tương ứng với lối vào đó của IRR sẽ được lập. Tín hiệu yêu cầu ngắt phải tồn tại trên lối vào IRRQ cho tới khi xuất hiện sườn xuống của xung INTA thứ nhất.

(3) ISR (In Service Register – Thanh ghi ngắt đang được phục vụ): Thanh ghi này ghi nhớ yêu cầu ngắt nào đang được phục vụ trong số các yêu cầu ngắt IRQi. Với mỗi IRQi đang được phục vụ thì bit thứ i tương ứng với nó trong ISR sẽ được lập (=1).

(4) PR (Priority Resolver – Bộ xử lý ưu tiên): Đây là mạch xử lý ưu tiên ngắt dựa trên nội dung 3 thanh ghi: IRR, ISR và IMR để quyết định một yêu cầu ngắt IRQi nào đó có được gửi tới CPU 8088 thông qua chân INT của khối logic điều khiển tại (một) thời điểm hiện tại hay bắt nó phải chờ.

(5) Control Logic – Logic điều khiển: Khối này có nhiệm vụ gửi các yêu cầu ngắt IRQi tới chân INTR của CPU 8088 khi có tín hiệu tại các lối vào IRQi và nhận trả lời chấp nhận yêu cầu ngắt INTA từ CPU để rồi điều khiển việc đưa ra số hiệu ngắt N trên bus dữ liệu (tại xung INTA thứ hai).

(6) Đệm bus dữ liệu: Dùng để phối ghép 8259A với bus dữ liệu của CPU.

(7) Logic điều khiển ghi/đọc: Dùng cho việc ghi các từ điều khiển và đọc các từ trạng thái của 8259A.

(8) Khối đệm nối tầng và so sánh: Ghi nhớ và so sánh các số hiệu của các mạch 8259A có mặt trong hệ thống vi xử lý.

- **Các tín hiệu của PIC 8259:**

Một số tín hiệu trong mạch PIC 8259A có tên gọi giống như các tín hiệu tiêu chuẩn của hệ vi xử lý 8088 (hình vẽ – sơ đồ khối), ngoài ra còn có một số tín hiệu đặc biệt sau:

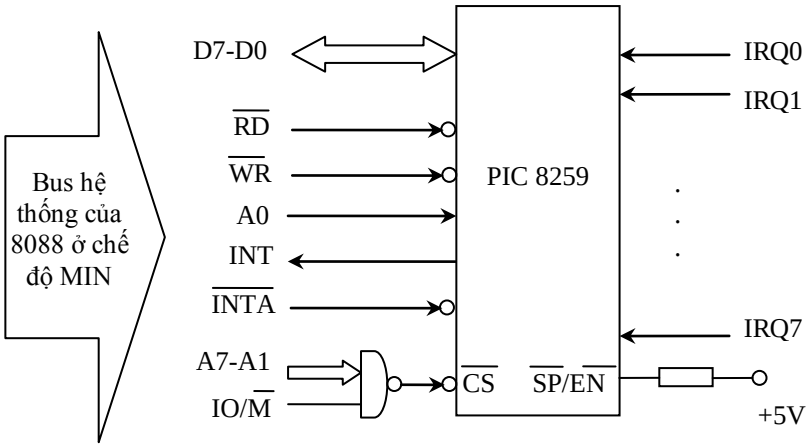
Cas0 – Cas2 [I/O]: Đây là đầu vào đối với mạch PIC thợ và là đầu ra đối với mạch PIC chủ. Chúng được sử dụng khi nối tầng để tăng thêm số lượng các yêu cầu ngắt cần xử lý.

SP/EN [I/O]: Khi 8259 làm việc ở chế độ không có đệm bus dữ liệu thì đây là tín hiệu vào để lập trình cho mạch 8259 thành mạch thợ (SP = 0) hoặc mạch chủ (SP =1). Khi 8259 làm việc trong hệ vi xử lý ở chế độ có đệm bus dữ liệu thì đây là tín hiệu EN dùng để mở đệm bus dữ liệu cho 8259 (và 8088) thông vào bus dữ liệu của hệ thống. Lúc này việc định nghĩa mạch 8259 là mạch chủ hay mạch thợ phải thực hiện thông qua từ điều khiển ICW4 (sẽ nói kỹ ở phần sau).

INT [O]: Là tín hiệu yêu cầu ngắt đến chân INTR của 8088.

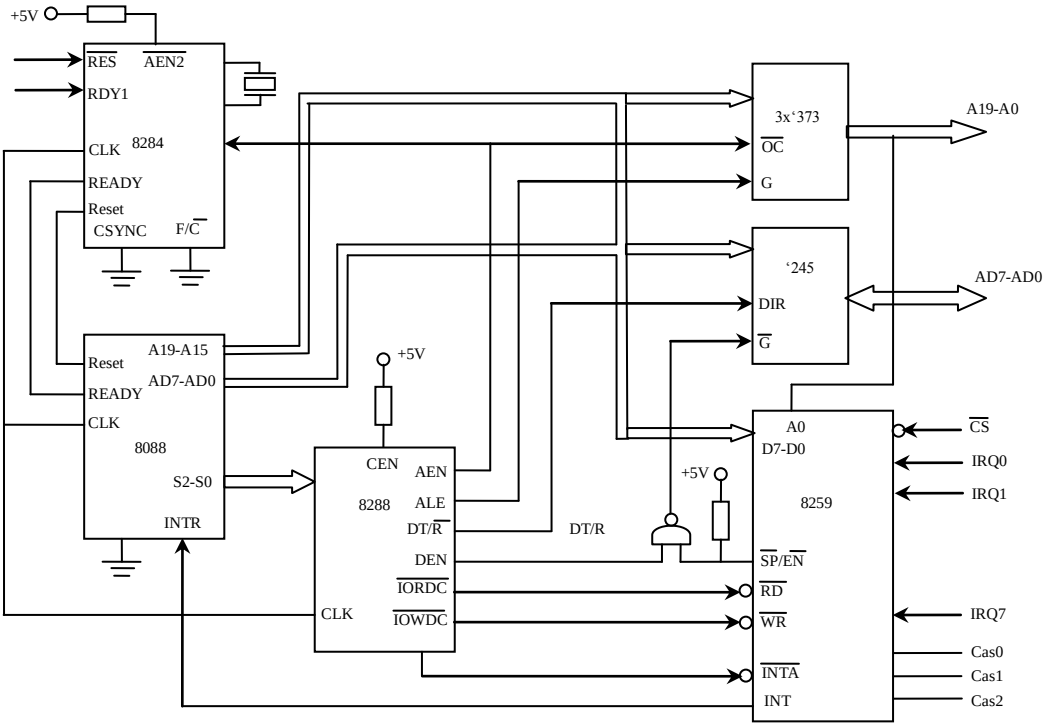
INTA [I]: Nối với chân báo chấp nhận ngắt của CPU. 8259 sử dụng xung INTA thứ nhất phát ra từ 8088 để thực hiện một số thao tác nội bộ (xử lý ưu tiên), tuỳ thuộc vào mode hoạt động mà nó được lập trình. Khi nhận được xung INTA thứ hai từ 8088, 8259 sẽ xuất số hiệu ngắt N tương ứng yêu cầu ngắt IRQi lên bus dữ liệu. Số hiệu ngắt này được xác định: thứ nhất là IRQi, thứ hai là số hiệu ngắt mà người lập trình viết cho 8259 khi khởi động nó.

RD/WR [I]: Các tín hiệu điều khiển đọc/ghi sẽ điều khiển các quá trình truyền, nhận từ điều khiển và từ trạng thái giữa 8088 và 8259 thông qua bus dữ liệu khi CS ở mức thấp.



Hình vẽ: PIC 8259 làm việc độc lập (mạch chủ ở chế độ không đệm bus) nối với 8088 làm việc ở chế độ MIN thông qua bus hệ thống, địa chỉ mạch PIC 8259A tại 0FEh – 0FFh.

ở sơ đồ này ta thấy chân SP/EN của 8259 được nối với +5V, CS được lấy từ mạch giải mã địa chỉ. Vì chỉ có một mạch PIC trong hệ thống nên có thể xử lý được 8 yêu cầu ngắt đồng thời: IRQ0 – IRQ7.



Hình vẽ: 8259 chủ (làm việc độc lập) nối với 8088 ở chế độ MAX

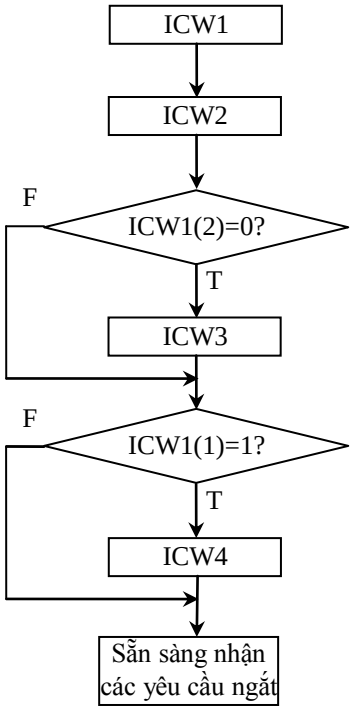
Nếu hệ vi xử lý 8088 làm việc ở chế độ MAX thường ta phải dùng mạch điều khiển bus 8288 và các đệm bus để cung cấp các tín hiệu thích hợp cho hệ thống. Mạch PIC 8259 phải làm việc ở chế độ có đệm bus để nối được với bus hệ thống này.

Theo hình vẽ ví dụ (8088 chế độ MAX nối với PIC 8259) ta thấy tín hiệu địa chỉ cho 8259 được lấy ra từ bus hệ thống (chứ không lấy trực tiếp từ 8088), trong khi đó tín hiệu dữ liệu của nó được nối với bus dữ liệu của bộ vi xử lý và từ đó được thông qua các đệm để nối vào bus hệ thống.

- **Lập trình cho PIC 8259**

Vi mạch điều khiển ngắt ưu tiên PIC 8259A là vi mạch lập trình cỡ lớn lập trình được. Để mạch PIC 8259 hoạt động được theo yêu cầu, sau khi bật nguồn cấp điện cho nó, PIC cần phải được lập trình (thiết lập chế độ hoạt động) bằng cách ghi vào các thanh ghi (tương ứng với các cổng) bên trong nó các từ điều khiển khởi đầu ICW (Initialization Control Word) và tiếp sau đó là các từ điều khiển hoạt động OCW (Operation Control Word).

Các từ điều khiển khởi đầu dùng để tạo nên các kiểu làm việc cơ bản ho PIC, còn các từ điều khiển hoạt động sẽ quyết định cách thức hoạt động cụ thể của PIC. Từ điều khiển khởi đầu cần được ghi trước khi PIC làm việc (tức là trước khi PIC nhận được các yêu cầu ngắt để xử lý) còn từ điều khiển hoạt động sẽ được ghi khi ta muốn thay đổi (cách thức) hoạt động của PIC 8259.



Hình vẽ: Lưu đồ ghi các từ điều khiển khởi đầu cho PIC 8259

Cụ thể các từ điều khiển nói trên như sau:

i, Các từ điều khiển khởi đầu:

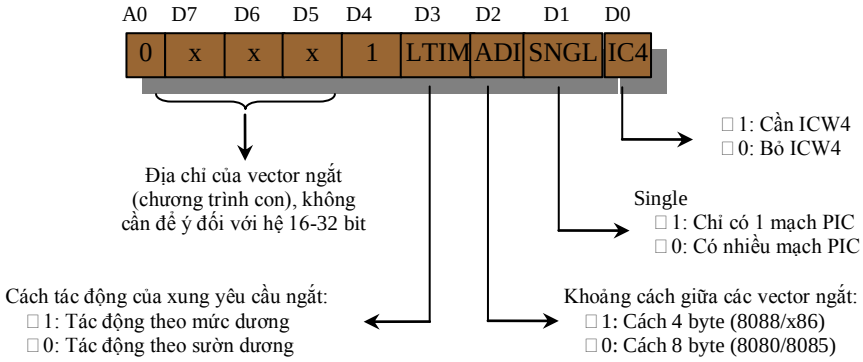
PIC 8259 có tất cả 4 từ điều khiển khởi đầu là ICW1 – ICW4. Trong khi lập trình cho 8259 không phải lúc nào ta cũng cần dùng cả 4 từ điều khiển khởi đầu đó (tức là ghi tất cả chúng). Tuỳ theo các trường hợp ứng dụng cụ thể mà có lúc ta cần ghi liên tiếp cả 4 từ điều khiển khởi đầu nhưng có lúc ta chỉ cần ghi 2 hoặc 3 từ điều khiển khởi đầu (trong số đó) là đủ. Thứ tự và điều kiện ghi các từ điều khiển khởi đầu ICW vào 8259 được thực hiện theo lưu đồ sau:

- **ICW1**

Như đã biết, PIC 8259 chỉ có một đường địa chỉ vào (A0), trong khi đó nó có rất nhiều thanh ghi điều khiển và trạng thái tương đương (tương ứng) với các cổng. Tuy nhiên để phân biệt các thanh ghi trong đó (trong 8259) thì đầu vào địa chỉ A0 và *thứ tự ghi* sẽ giúp ta phân biệt ra cá thanh ghi khác nhau bên trong 8259 để ghi dữ liệu cho các từ điều khiển. Ví dụ, A0 = 0 là dấu hiệu để nhận biết rằng ICW1 được (ghi) đưa vào thanh ghi có địa chỉ chẵn trong PIC, còn khi A0 = 1 thì các từ điều khiển ICW2, ICW3 và ICW4 sẽ được (ghi) đưa vào các thanh ghi có địa chỉ lẻ trong PIC.

Khi giới thiệu từng thanh ghi cụ thể với ý nghĩa các bit dữ liệu, ta còn ghi rõ thêm cả giá trị A0 tương ứng cho mỗi ICWi đó.

Bit D0 của ICW1 (IC4) quyết định 8259 sẽ được nối với hệ vi xử lý nào. Để làm việc với hệ 16, 32 bit (8088/x86) thì trong ICW1 nhất thiết phải có IC4 = 1 (tức là ta luôn cần đến từ điều khiển khởi đầu ICW4), còn đối với hệ vi xử lý 8 bit (như khi xử dụng 8080/8085) thì ta phải có IC4 = 0 (và như vậy các bit của ICW4 sẽ vị xoá về 0).



Hình vẽ: Thanh ghi khởi đầu ICW1

Các bit còn lại của ICW1 định nghĩa cách thức làm tác động của xung yêu cầu ngắt (tác động theo sườn hay theo mức) tại các chân yêu cầu ngắt IRQi của 8259 và việc bố trí các mạch PIC 8259 khác trong hệ làm việc đơn lẻ hay theo chế độ nối tầng.

Khi sử dụng 8259 trong hệ 16 – 32 bit (8088/x86), bit D2 (ADI) có thể nhận các giá trị tuỳ ý, nó chỉ có ý nghĩa khi làm việc với hệ 8 bit nên thường được chọn là 0.

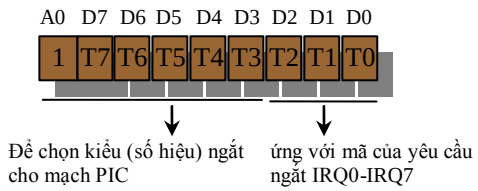
Nếu chỉ sử dụng 1 vi mạch PIC 8259 trong hệ, đặt bit D1 = 1 (SNGL (S) = 1). Còn nếu có nhiều mạch PIC nối tầng trong hệ thống thì bit D1 của ICW1 trong các mạch PIC phải được xoá về 0.

Bit D3 (LTIM) được dùng để xác lập trigger theo mức hay trigger theo sườn xung. Trong chế độ trigger theo mức, chương trình ngắt (yêu cầu ngắt) được yêu cầu mỗi khi xuất hiện mức điện áp cao trên lối vào yêu cầu ngắt IRQ. Trong chế độ trigger theo sườn xung, tín hiệu yêu cầu ngắt trên các lối vào IRQ phải được chuyển từ mức thấp lên mức cao và duy trì ở mức cao cho đến khi chương trình con phục vụ ngắt thực hiện. Vì vậy có thể nói bit này quy định cách thức tác động của xung yêu cầu ngắt là theo mức hay theo sườn (dương).

Khi hoạt động trong hệ 16 – 32 bit thì không cần quan tâm tới các bit D5, D6 và D7 do đó thường đặt chúng bằng 0 cho đơn giản.

Ngày nay, thường sử dụng bộ vi xử lý 8088 hoặc cao hơn khi xây dựng các hệ vi xử lý nên có thể sử dụng giá trị: 0000 1010 để ghi cho ICW1.

- **ICW2**



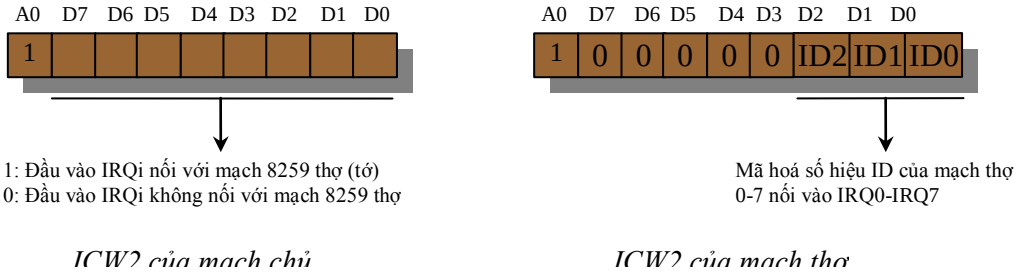
Hình vẽ: Thanh ghi khởi đầu ICW2

Trong một hệ vi xử lý 16 – 32 bit, từ điều khiển khởi đầu này được dùng để (cho phép) chọn kiểu ngắt (số hiệu ngắt N) tương ứng với các bit T7 – T3 cho các đầu vào yêu cầu ngắt. Các bit T2 – T0 được 8259 tự động gán giá trị tuỳ theo đầu vào yêu cầu ngắt IRQi cụ thể. Thực chất chúng có ý nghĩa như sau: Toàn bộ các bit của ICW2 sẽ báo cho 8259 biết số hiệu ngắt phải gửi (đi) ra bus dữ liệu để đáp ứng tín hiệu yêu cầu ngắt IRQ0. Để đáp ứng các yêu cầu ngắt khác (IRQ1 – IRQ7), 8259 sẽ tự động cộng chỉ số lối vào với giá trị cơ sở này ("không trừ") và gửi kết quả cho 8088 dùng làm số hiệu ngắt cho các tín hiệu yêu cầu ngắt đó. Do các ngắt có số hiệu ngắt từ 0 – 31 hoặc là các ngắt chuyên dụng hoặc là để dự trữ nên số 32 (20h) là số ngắt thấp nhất cho các ứng dụng mở rộng của người sử dụng. Giả sử, cho ICW2 = 0010 0000 = 32d, 8259 sẽ gửi số hiệu ngắt này lên bus dữ liệu cho 8088

làm số hiệu ngắt cho IRQ0. Với tín hiệu yêu cầu ngắt trên IRQ1, 8259 sẽ gán cho số hiệu ngắt 0010 0001 = 33d (tiếp tục), tương tự như vậy có: IRQ2 = 0010 0010 = 34d, IRQ3 = 0010 0011 = 34d, …

Trong bất kỳ ICW2 nào nạp vào 8259, 3 bit thấp nhất luôn luôn bằng 0 vì 8259 sẽ tự động cấp phát các bit này sao cho tương ứng với chỉ số lối vào yêu cầu ngắt IRQi.

- **ICW3**



Hình vẽ: Thanh ghi khởi đầu ICW2

Từ điều khiển khởi đầu này chỉ sử dụng đến khi bit D2 (SNGL – S) thuộc từ điều khiển khởi đầu ICW1 có giá trị = 0, có nghĩa là trong hệ thống có nhiều mạch PIC 8259 làm việc trong chế độ nối tầng (chủ – tớ). Chính vì vậy mà tồn tại (có) 2 loại ICW3:

ICW3 cho mạch 8259 chủ: từ điều khiển này để chỉ ra đầu vào yêu cầu ngắt IRQi nào của nó có (tín hiệu INT của) mạch tớ nối vào. Cần phải ghi ICW3 vào mạch chủ để nó biết với các lối vào IRQ thì phải xuất ID của các vi mạch 8259 tớ lên các đường Cas0 – Cas2.

ICW3 cho mạch 8259 tớ: dùng làm phương tiện để các mạch tớ này được nhận biết, vì vậy từ điều khiển này (“phải”) chứa mã số i ứng với đầu vào IRQi của mạch chủ mà mạch 8259 tớ đã nối vào. Mã số i (chính là ID) được sử dụng khi yêu cầu ngắt từ 8259 tớ. Khi vi mạch 8259 chủ nhận một yêu cầu ngắt từ mạch tớ nào đó – nó sẽ biết được yêu cầu ngắt đó là t mạch tớ nào thông qua IRQ tương ứng. Khi yêu cầu ngắt đó đã được chấp nhận, mạch chủ sẽ yêu cầu mạch tớ đó xuất số hiệu ngắt lên bus dữ liệu bằng cách nó đưa số ID ra các chân Cas0 – Cas 2, vi mạch 8259 tớ sẽ so sánh ID của nó với số ID trên Cas0 – Cas2, vi mạch tớ nào thấy trùng (có nghĩa là nó đã nhận ra ID của nó) thì nó sẽ đưa số hiệu ngắt lên bus dữ liệu (Sự xử lý ưu tiên ở đây có nghĩa là: Tại một thời điểm có thể có nhiều yêu cầu ngắt (từ các thiết bị ngoại vi) gửi tới mạch tớ. Trước hết chúng cùng gửi các yêu cầu ngắt tới mạch chủ và chờ xung INTA. Nhận được các yêu cầu ngắt từ các mạch tớ, mạch chủ cũng gửi xung yêu cầu ngắt qua chân INT tới 8088. Nếu yêu cầu ngắt này được chấp nhận thì 8088 sẽ gửi xung chấp nhận ngắt thứ nhất INTA tới tất cả các mạch 8259 (chân này tại các mạch PIC chủ và tớ được nối chung). Nhận được xung INTA này, tất cả các mạch PIC đều có những xử lý ưu tiên để chuẩn bị đưa số hiệu ngắt (của ngắt có chứa mức ưu tiên cao nhất) lên bus dữ liệu khi có xung INTA thứ hai. Nhưng khi có xung INTA thứ hai, các mạch tớ còn phải xem xét xem có được sự “cho phép” từ mạch chủ hay không thông qua việc so sánh ID của nó với ID của mạch chủ “chỉ định” qua các chân Cas0 – Cas2. Số hiệu ID này trùng với một trong những chỉ số i của IRQi trên mạch chủ mà trước đó có tín hiệu yêu cầu ngắt từ mạch tớ gửi đến. Như vậy khi xung INTA thứ hai xuất hiện thì số hiệu ngắt (của ngắt nào đó) từ mạch thợ được chấp nhận nào đó sẽ được đưa lên bus dữ liệu để 8088 đọc được.

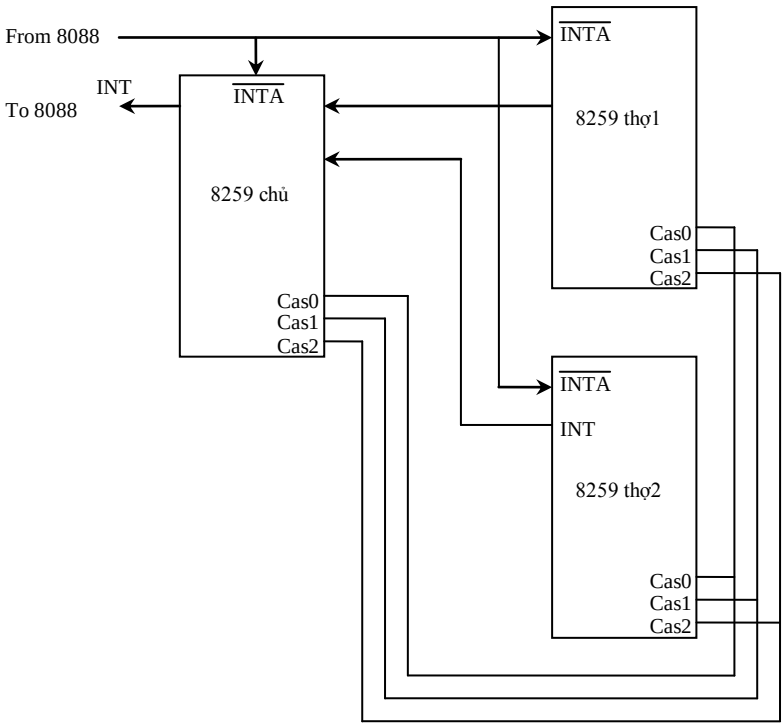
Ví dụ: Trong một hệ vi xử lý ta có một mạch PIC 8259A chủ và 2 mạch 8259A thợ nối vào chân IRQ0 và IRQ2 của mạch chủ. Tìm giá trị các từ điều khiển khởi đầu ICW3 để gán cho các mạch PIC.

Giải:

Mạch chủ: ICW3 = 0000 0101b = 05h

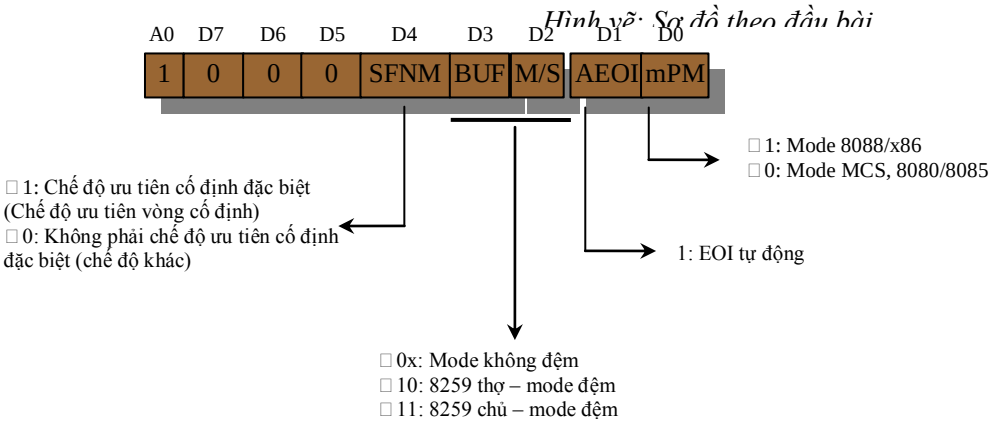
Mạch thợ 1 (nối vào IRQ0 của mạch chủ): ICW3 = 0000 0000b = 00h

Mạch thợ 2 (nối vào IRQ2 của mạch chủ): ICW3 = 0000 0010b = 02h



Hình vẽ: Sơ đồ theo đầu bài

- **ICW4**



Hình vẽ: Thanh ghi khởi đầu ICW4

Từ điều khiển khởi đầu này chỉ dùng đến khi trong từ điều khiển khởi đầu ICW1 có bit IC4 = 1 (cần thêm ICW4).

Bit PM (Microprocessor Mode): Cho phép các bộ vi xử lý 8088/86 hoặc cao hơn (80x86) làm việc với 8259. Nếu µPM = 0 thì cho phép các bộ vi xử lý 8080/85 làm việc với 8259.

Bit SFNM = 1, cho phép ta chọn *chế độ ưu tiên cố định đặc biệt*. Trong chế độ này yêu cầu ngắt với mức ưu tiên cao nhất hiện thời từ một mạch thợ làm việc theo kiểu nối tầng sẽ được mạch chủ nhận biết ngay cả khi mạch chủ còn đang phục vụ một yêu cầu ngắt ở mạch thợ khác nhưng với mức ưu tiên thấp hơn (như đã biết, khi 8088 nhận được yêu cầu ngắt, nếu yêu cầu ngắt đó được chấp nhận thì trong các công việc nó làm để chuẩn bị thực hiện ISR có công đoạn: Xoá cờ IF và TF, điều này có nghĩa là khi nó đang thực hiện một ISR thì nó cấm các ngắt che được khác tác động. Tuy nhiên, nếu ngay đầu ISR lại có các lệnh lập cờ IF thì nó vẫn có thể nhận biết các yêu cầu ngắt che được khác ngay khi đang thực hiện một ISR nào đó, vấn đề này còn được đề cập đến trong phần sau). Sau khi các yêu cầu ngắt

được phục vụ xong thì chương trình con phục vụ ngắt phải có lệnh kết thúc yêu cầu ngắt (EOF) đặt trước lệnh (IRET) trở về để đưa đến cho mạch PIC chủ.

Khi bit SFNM = 0 thì *chế độ ưu tiên cố định* được chọn (IRQ0: mức ưu tiên cao nhất, …, IRQ7: mức ưu tiên thấp nhất). Thực ra đối với mạch 8259 nếu không dùng ICW4 thì chế độ này được chọn như là ngầm định. Trong chế độ ưu tiên cố định, tại một thời điểm chỉ có một yêu cầu ngắt i được phục vụ (bit ISRi = 1), lúc này tất cả các yêu cầu ngắt khác với mức ưu tiên thấp hơn đều bị cấm và các yêu cầu ngắt khác với mức ưu tiên cao hơn có thể ngắt yêu cầu ngắt khác với mức ưu tiên thấp hơn.

Bit BUF (Buffer): Cho phép định nghĩa mạch 8259 để làm việc với CPU trong trường hợp có đệm hoặc không có đệm nối với bus hệ thống. Khi BUF = 1: 8259 làm việc ở chế độ có đệm bus, bit M/S = 1/0 cho phép ta chọn mạch 8259 để làm việc ở chế độ chủ/thợ (Master/Slave). SP/EN trở thành đầu ra cho phép mở đệm bus để PIC 8259 và CPU thông với bus hệ thống.

Bit AEOI (Automatic End Of Interrupt): Cho phép chọn cách kết thúc yêu cầu ngắt tự động.

- AEOF = 1 thì 8259 tự động xoá ISRi = 0 khi xung INTA cuối cùng chuyển lên mức cao mà không làm thay đổi thứ tự ưu tiên.
- Ngược lại, khi ta chọn cách kết thúc yêu cầu ngắt thường (AEOF = 0) thì chương trình con phục vụ ngắt phải có thêm lệnh EOI đặt trước lệnh IRET để kết thúc cho 8259.

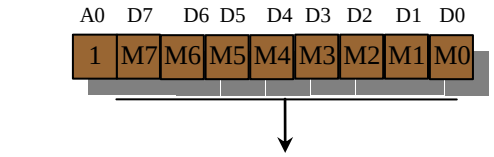
Những vấn đề liên quan đến chế độ ưu tiên và kết thúc yêu cầu ngắt còn được nói thêm trong phần giới thiệu về các thanh ghi từ điều khiển hoạt động.

ii, Các từ điều khiển hoạt động:

Các từ điều khiển hoạt động OCWi sẽ quyết định 8259 hoạt động như thế nào sau khi nó đã được khởi đầu bằng các từ điều khiển khởi đầu ICWi. Tất cả các từ điều khiển hoạt động sẽ được ghi vào các thanh ghi trong PIC khi A0 = 0, trừ OCW1 được hi khi A0 = 1 (chỉ số các thanh ghi và địa chỉ là ngược lại với các thành ghi điều khiển khởi đầu).

- **OCW1**

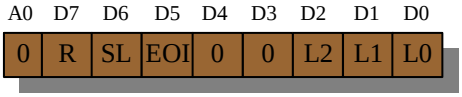
Thanh ghi này để ghi các bit mặt nạ vào thanh ghi mặt nạ ngắt IMR. Khi một bit mặt nạ nào đó của thanh ghi IMR được lập thì yêu cầu ngắt tương ứng với mặt nạ đó sẽ không được 8259 nhận biết nữa (bị che). Từ điều khiển này phải được (ghi) đưa đến 8259 ngay sau khi các từ điều khiển khởi đầu ICWi được ghi vào 8259 để cho các lối vào yêu cầu ngắt IRTQi muốn đáp ứng. Ta cũng có thể đọc lại IMR bất kỳ lúc nào để xác định tình trạng mặt nạ ngắt hiện tại (có nghĩa, kiểm tra xem trong thời điểm hiện tại yêu cầu ngắt nào được phép, yêu cầu ngắt nào bị che).



Mặt nạ ngắt tại các yêu cầu ngắt
1: Có mặt nạ (yêu cầu ngắt tương ứng bị che)
0: Không có mặt nạ (yêu cầu ngắt tương ứng được nhận biết)

Hình vẽ: Thanh ghi hoạt động OCW1

- **OCW2**



Mã hoá mức ưu tiên ngắt bị tác động
000: mức 0
001: mức 1
...
111: mức 7
Trong đó: (*): dùng tổ hợp L2L1L0

	R	SL	EOI	ý nghĩa
Kết thúc ngắt (EOI)	0	0	1	Lệnh EOI thường
	0	1	1	Lệnh EOI chỉ đích danh (*)
Đổi mức ưu tiên tự động (quay vòng)	1	0	1	Đổi mức ưu tiên khi có EOI thường
	1	0	0	Lập chế độ quay khi có EOI tự động
	0	0	0	Xoá chế độ quay khi có EOI tự động
Đổi mức ưu tiên đích danh (đặc biệt)	1	1	1	Đổi mức ưu tiên khi có EOI chỉ đích danh (*)
	1	1	0	Lệnh lập mức ưu tiên (*)
	0	1	0	Không làm gì

Hình vẽ: Thanh ghi hoạt động OCW2

Các bit: R, SL và EOI (theo bảng) phối hợp nhau cho phép chọn ra cách thức kết thúc yêu cầu ngắt khác nhau. Trong đó, một vài cách kết thúc yêu cầu ngắt còn tác động tới các yêu cầu ngắt được chỉ đích danh với mức ưu tiên được mã hoá bởi 3 bit L2, L1, L0 (L2L1L0).

Một số chế độ làm việc của 8259

(1) Chế độ ưu tiên cố định: Đây là chế độ làm việc ngầm định của 8259 sau khi nó đã được nạp các từ điều khiển khởi đầu. Trong chế độ này, các đầu vào yêu cầu ngắt IRQ7 – IRQ0 được gán cho các mức ưu tiên cố định. Cụ thể: IRQ0 được gán mức ưu tiên cao nhất, … IRQ7 được gán mức ưu tiên thấp nhất. Thứ tự mức ưu tiên này được giữ cố định (không thay đổi) cho đến khi mạch 8259 được lập trình khác đi thông qua việc ghi từ điều khiển hoạt động OCW2. Trong chế độ ưu tiên cố định, tại một thời điểm chỉ có một yêu cầu ngắt i được phục vụ (bit ISRi = 1), lúc này tất cả các yêu cầu ngắt khác có mức ưu tiên cao hơn nào đó có thể ngắt các yêu cầu ngắt có mức ưu tiên thấp hơn.

(2) Chế độ quay mức ưu tiên tự động (ưu tiên luân phiên): ở chế độ này, sau khi một yêu cầu ngắt nào đó được phục vụ xong, 8259 sẽ xoá bit tương ứng của nó trong thanh ghi ISR và gán cho đầu vào của nó (IRQ) mức ưu tiên thấp nhất để tạo điều kiện cho các yêu cầu ngắt khác (với mức ưu tiên thấp) có cơ hội (thời cơ) được phục vụ.

(3) Chế độ quay (đổi) mức ưu tiên chỉ đích danh: ở chế độ này ta cần chỉ rõ (chỉ đích danh) đầu vào yêu cầu ngắt IRQi nào với i = L2L1L0 được gán mức ưu tiên thấp nhất, đầu vào IRQi+1 sẽ được tự động gán mức ưu tiên cao nhất.

Đó là 3 chế độ làm việc tiêu biểu của 8259. Trên cơ sở đó, nhìn lại sự phối hợp giữa các bit R, SL, EOI như thế nào để tạo ra các lệnh quy định các cách thức kết thúc yêu cầu ngắt cho các chế độ làm việc khác nhau như đã nói qua.

(1) Kết thúc yêu cầu ngắt thường: Trong chương trình con phục vụ ngắt phải có lệnh EOI đặt trước lệnh trở về IRET cho 8259. Mạch 8259 sẽ xác định yêu cầu ngắt IRQi vừa được phục vụ và xoá bit ISRi tương ứng của nó để tạo điều kiện cho chính yêu cầu ngắt này hoặc các ngắt có mức ưu tiên thấp hơn có thể được tác động (phục vụ).

(2) Kết thúc yêu cầu ngắt chỉ đích danh: Trong chương trình con phục vụ ngắt phải có lệnh EOI chỉ đích danh đặt trước lệnh trở về IRET cho 8259. Mạch 8259 sẽ xoá đúng bit ISRi (i = L2L1L0) để tạo điều kiện cho chính yêu cầu ngắt này hoặc các yêu cầu ngắt có mức ưu tiên thấp hơn có thể được tác động.

(3) Quay (đổi) mức ưu tiên khi kết thúc yêu cầu ngắt thường: Trong chương trình con phục vụ ngắt phải có lệnh EOI đặt trước lệnh trở về IRET cho 8259. Mạch 8259 sẽ xác định yêu cầu ngắt thứ i vừa được phục vụ, xoá bit ISRi tương ứng và gán luôn mức ưu tiên thấp nhất cho đầu vào IRQi này, còn đầu vào IRQi+1 sẽ được gán mức ưu tiên cao nhất.

Ví dụ:

Thanh ghi ISR trước khi IRQ4 được chấp nhận

(0: mức ưu tiên cao nhất, 7: mức ưu tiên thấp nhất)

Các bit	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
Trạng thái của ISR	0	1	0	1	0	0	0	0
Mức ưu tiên	7	6	5	4	3	2	1	0

Thanh ghi ISR sau khi IRQ4 được chấp nhận và sau khi có lệnh quay đổi

Các bit	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
Trạng thái của ISR	0	1	0	0	0	0	0	0
Mức ưu tiên	2	1	0	7	6	5	4	3

Thanh ghi ISR trước khi IRQ5 được chấp nhận

(0: mức ưu tiên cao nhất, 7: mức ưu tiên thấp nhất)

Các bit	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
Trạng thái của ISR	0	0	1	0	0	1	0	0
Mức ưu tiên	7	6	5	4	3	2	1	0

Thanh ghi ISR sau khi IRQ4 được chấp nhận và sau khi có lệnh quay đổi

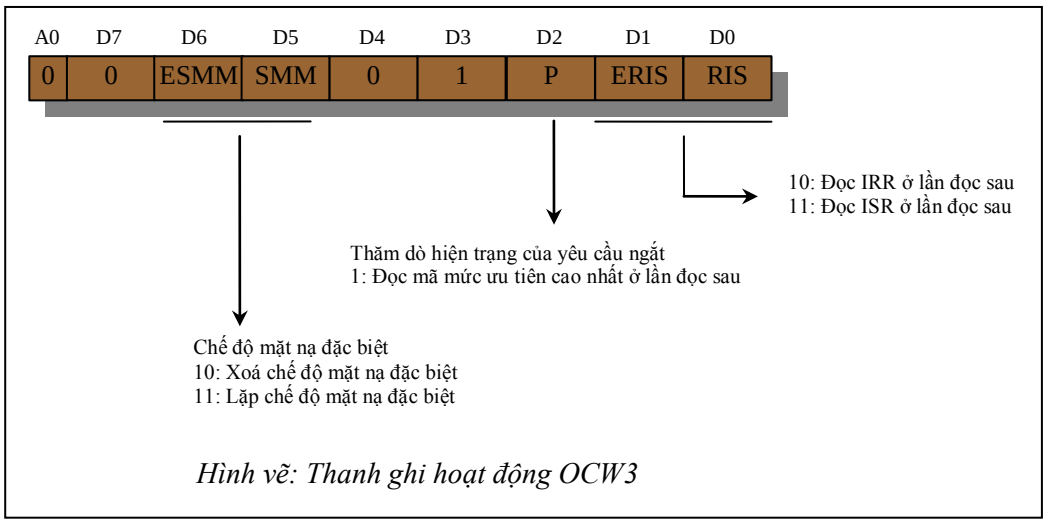
Các bit	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
Trạng thái của ISR	0	0	0	0	0	1	0	0
Mức ưu tiên	1	0	7	6	5	4	3	2

(1) Quay (đổi) mức ưu tiên trong chế độ kết thúc ngắt tự động: Trong chế độ này chỉ cần một lần đưa lệnh chọn chế độ mức ưu tiên khi kết thúc yêu cầu ngắt từ động. Có thể chọn lệnh này bằng lập "chế độ quay khi có EOI tự động". Từ đó trở đi 8259 sẽ đổi mức ưu tiên mỗi khi kết thúc yêu cầu ngắt tự động tương tự mục trước. Muốn loạ bỏ chế độ này ta có thể dùng lệnh xoá "chế độ quay khi có EOI tự động".

(2) Quay (đổi) mức ưu tiên khi kết thúc yêu cầu ngắt chỉ đích danh: Trong chương trình con phục vụ ngắt phải có lệnh EOI đích danh cho 8259 đặt trước lệnh trở về IRET. Mạch 8259 sẽ xoá đúng bit ISRi (i = L2L1L0).

(3) Lập mức ưu tiên: Chế độ này cho phép thay đổi mức ưu tiên có định hoặc mức ưu tiên gán trước đó bằng cách gán mức ưu tiên thấp nhất cho yêu cầu ngắt IRQi chỉ đích danh ứng với tổ hợp mã i = L2L1L0. Yêu cầu ngắt IRQi+1 sẽ được gán mức ưu tiên cao nhất.

- **OCW3**

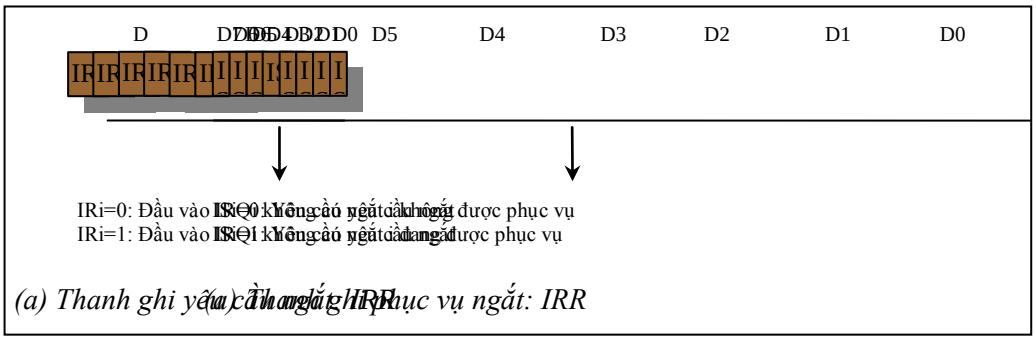


Hình vẽ: Thanh ghi hoạt động OCW3

Từ điều khiển hoạt động OCW3 sau khi được nạp vào 8259 cho phép:

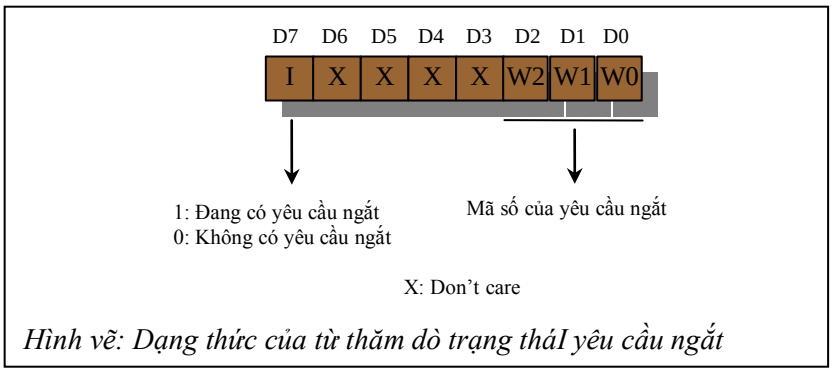
- Chọn ra các thanh ghi để đọc (trạng thái)
- Thăm dò trạng thái yêu cầu ngắt bằng cách đọc trạng thái của đâu vào yêu cầu ngắt IRQi với mức ưu tiên cao nhất cùng với mã của đầu vào đó.
- Thao tác với mặt nạ đặc biệt.

Các thanh ghi IRR và ISR có thể được nạp sau khi nạp từ điều khiển OCW3 vào 8259 với bit ERIS = 1 (D1). Bit (D0) RIS = 0, cho phép đọc IRR ở lần đọc sau; RIS = 1, cho phép đọc ISR ở lần đọc sau.



(a) Thanh ghi yêu cầu ngắt và thanh ghi phục vụ ngắt: IRR

Bằng việc đưa vào 8259 từ điều khiển hoạt động OCW3 với bit P = 1, ta có thể đọc được trên bus dữ liệu ở lần đọc tiếp ngay sau đấy từ thăm dò, trong đó các thông tin về yêu cầu ngắt với mức ưu tiên cao nhất đang hoạt động và mã tương ứng với yêu cầu ngắt đó theo dạng sau:



Hình vẽ: Dạng thức của từ thăm dò trạng thái yêu cầu ngắt

Ta có thể coi đây là chế độ thăm dò yêu cầu ngắt và chế độ này thường được ứng dụng trong trường hợp có nhiều chương trình con phục vụ ngắt giống nhau cho một yêu cầu ngắt

và việc chọn chương trình nào để sử dụng là (trách nhiệm) công việc của người sử dụng (người lập trình).

Vậy, muốn dùng chế độ thăm dò của 8259 để xác định yêu cầu ngắt hiện thời ta cầ thực hiện lần lượt dãy thao tác sau:

- Cấm các yêu cầu ngắt che được bằng lệnh CLI (xoá IF).
- Ghi từ lệnh OCW3 với bit P=1.
- Đọc từ thăm dò trạng thái yêu cầu ngắt trên bus dữ liệu.

Bit ESMM = 1: Cho phép 8259 thao tác với chế độ mặt nạ đặc biệt. Bit SMM = 1: cho phép chế độ mặt nạ đặc biệt. Chế độ mặt nạ đặc biệt được dùng để thay đổi thứ tự ưu tiên ngay bên trong chương trình con phục vụ ngắt. Ví dụ, trong trường hợp có một yêu cầu ngắt bị cấm (bị che bởi chương trình con phục vụ ngắt với từ điều khiển hoạt động OCW1) mà ta lại muốn cho phép các yêu cầu ngắt với mức ưu tiên thấp hơn so với mưc ưu tiên của yêu cầu ngắt bị cấm đó được tác động thì ta sẽ dùng chế độ mặt nạ đặc biệt. Nếu đã được thiết lập, chế độ mặt nạ đặc biệt sẽ tồn tại cho đến khi xoá đi bằng cách ghi vào 8259 một từ điều khiển OCW3 khác với bit SMM = 0. Mặt nạ đặc biệt không ảnh hưởng tới các yêu cầu ngắt có mức ưu tiên cao hơn.

Tóm lược hoạt động của 8259 trong hệ vi xử lý 8088:

(1) Khi có yêu cầu ngắt từ thiết bị ngoại vi tác động vào chân IRQi nào đó của 8259 thì nó sẽ gửi xung INT = 1 đến chân INTR của CPU 8088.
(2) Nếu chấp nhận, 8088 sẽ đưa xung INTA (low active) đầu tiên đến 8259.
(3) 8259 dùng xung INTA đầu này như là thông báo để nó hoàn tất các xử lý nội bộ cần thiết, kể cả việc xử lý ưu tiên nếu có nhiều yêu cầu ngắt cùng tác động.
(4) 8088 đưa xung INTA thứ hai khiến 8259 đưa ra bus dữ liệu 1 byte là số hiệu ngắt của yêu cầu ngắt vừa được chấp nhận (yêu cầu ngắt có mức ưu tiên cao nhất (nếu có nhiều yêu cầu ngắt cùng tác động) – tuỳ theo chế độ làm việc của 8259).
(5) 8088 tính toán địa chỉ của chương trình con phục vụ ngắt dựa trên số hiệu ngắt, cụ thể như sau:
   - Cất FR; xoá IF, TF; cất CS, IP vào Stack
   - Lấy CS, IP của chương trình con phục vụ ngắt từ bảng vector ngắt và thực hiện nó.

Tuỳ theo những yêu cầu khác nhau về sự hoạt động của 8259 mà trong chương trình con phục vụ ngắt có lệnh ghi OCW1, OCW2 hay OCW3 vào 8259.

Chương 7
VÀO RA DỮ LIỆU BẰNG DMA

7.1. Nguyên tắc của việc trao đổi dữ liệu với thiết bị ngoại vi bằng cách thâm nhập trực tiếp vào bộ nhớ (DMA)

Trong các kiểu điều khiển trao đổi dữ liệu giữa thiết bị ngoại vi và hệ VXL bằng cách thăm dò trạng thái sẵn sàng của thiết bị ngoại vi hay bằng cách ngắt bộ VXL đã được đưa ra ở các chương trước, thì chúng ta nhận thấy là dữ liệu được chuyển từ bộ nhớ tới thiết bị ngoại vi và ngược lại đều phải qua bộ VXL. Vi thế tốc độ truyền dữ liệu phục thuộc vào tốc độ của các lệnh di chuyển dữ liệu như là MOV, IN, OUT, .v.v.

Tuy nhiên trong thực tế có những lúc ta cần trao đổi dữ liệu tốc độ cao với thiết bị ngoại vi, ví dụ như cần đưa dữ liệu ra màn hình, hoặc trao đổi dữ liệu giữa hai thiết bị lưu trữ. Trong các trường hợp đó ta cần có được khả năng ghi/đọc dữ liệu trực tiếp với bộ nhớ (direct memory access-DMA). Phương pháp này sẽ tiến hành thâm nhập trực tiếp vào bộ nhớ để chuyển dữ liệu tới đích mà không thông qua CPU. Để làm được điều này các hệ VXL nói chung đều phải dùng thêm một mạch chuyên dụng để điều khiển việc thâm nhập trực tiếp vào bộ nhớ (direct memory access controller-DMAC).

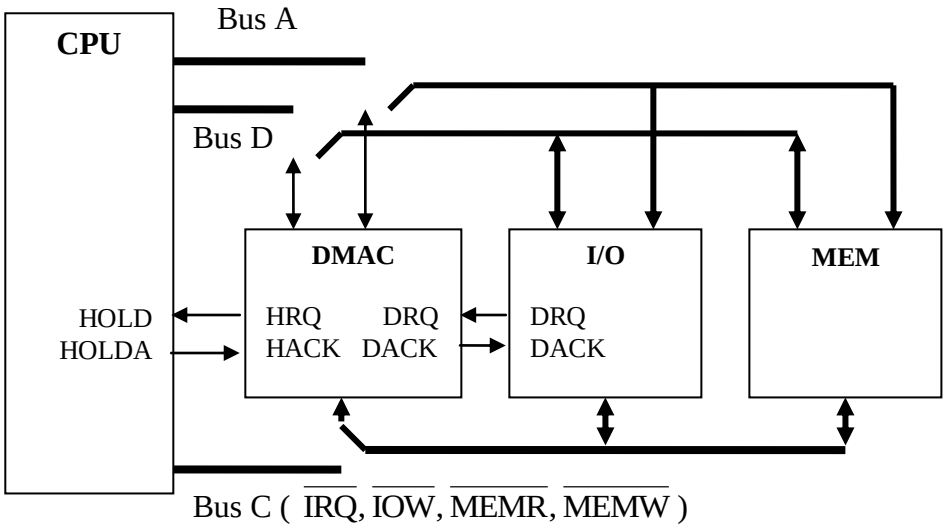
Có thể so sánh tốc độ truyền 1 byte nhớ từ bộ nhớ ra thiết bị ngoại vi sử dụng hai phương pháp DMA và CPU như sau:

CPU	LAP: Mov AL,[SI] Out Port,AL Inc SI Loop LAP	10 chu kỳ 10 chu kỳ 2 chu kỳ 17 chu kỳ	Tổng là 39 chu kỳ
DMA	Chuyển một Byte dữ liệu	4 chu kỳ	Tổng là 4 chu kỳ

Để hỗ trợ việc trao đổi dữ liệu với thiết ịi ngoại vi bằng cách thâm nhập trực tiếp vào bộ nhớ, tại mỗi vi mạch CPU thường thồn tại hai chân tín hiệu điều khiển sau

- HOLD: Chân này là tín hiệu yêu cầu treo của DMAC cảnh báo cho CPU biết rằng nó cần sử dụng BUS dữ liệu cho mục đích trao đổi.
- HOLDA: Chân này là tín hiệu chấp nhận treo của CPU, chấp nhận nhường BUS lại cho DMAC sử dụng. Khi CPU nhận được HOLD nếu chấp nhận nó sẽ tự treo chính chính mình tức là tách nó ra khỏi hệ thống bằng cách đưa BUS dữ liệu của nó về trạng thái trở kháng cao, sau đó sẽ HOLDA nên mức tích cực báo cho DMA được phép sử dụng BUS dữ liệu.

Sơ đồ khôi của một hệ VXL có khả năng trao đổi dữ liệu theo kiểu DMA như sau:



Hệ VXL với DMAC

Ta nhận thấy trong hệ thống này, khi CPU tự tách ra khỏi hệ thống bằng việc tự treo (ứng với vị trí của các công tắc chuyển mạch hiện thời) để trao quyền sử dụng BUS dữ liệu cho DMAC thì DMAC phải chịu trách nhiệm điều khiển toàn bộ hoạt động trao đổi dữ liệu của hệ thống. Để làm được điều đó DMAC phải có khả năng tạo ra được các tín hiệu điều khiển cần thiết giống như CPU và bản thân nó phải là một thiết bị lập trình được (để CPU có thể đưa các thông tin cần thiết về công việc mà nó phải làm trước khi CPU tách ra khỏi hệ thống),

Quá trình hoạt động của hệ thống trên có thể được tóm tắt như sau:

Khi thiết bị ngoại vi có yêu cầu trao đổi dữ liệu kiểu DMA với bộ nhớ, nó đưa yêu cầu DRQ=1 đến DMAC, DMAC nếu chấp nhận tín hiệu này sẽ gửi tín hiệu HRQ=1 đến chân HOLD của CPU. Nhận được yêu cầu treo, nếu chấp nhận CPU sẽ treo các BUS của mình và gửi tín hiệu trả lời HLDA=1 chấp nhận treo tới chân HACK của DMAC. Khi đó DMAC sẽ trả lời thiết bị ngoại vi bằng tín hiệu DACK=1, và cho phép thiết bị ngoại vi được phép trao đổi đổi dữ liệu theo kiểu DMA. Kết thúc quá trình trao đổi DMA sẽ đặt tín hiệu HRQ=0, trả quyền điều khiển BUS dữ liệu lại cho CPU.

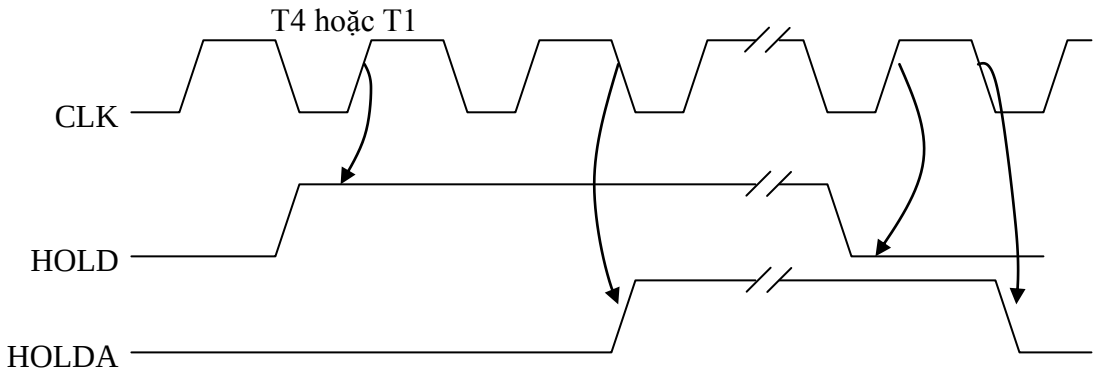
Trong thực tế có 3 kiểu trao đổi dữ liệu bằng cách thâm nhập trực tiếp vào bộ nhớ như sau:

- Treo CPU trong một khoảng thời gian để trao đổi cả mảng dữ liệu.
- Treo CPU để trao đổi từng Byte.
- Tận dụng thời gian không sử dụng BUS của CPU để trao đổi dữ liệu.

7.2. DMAC 8237-5 trong hệ vi xử lý 8088

7.2.1. Tín hiệu HOLD và HLDA trong CPU 8088

Hai tín hiệu yêu cầu treo và trả lời chấp nhận treo trong chế độ MIN của CPU 8088 là HOLD và HOLDA. Quan hệ giữa hai tín hiệu được thể hiện như sau:



7.2.2. Mạch DMAC 8237-5 của Intel

Vấn đề:

Trong thực tế tại một thời điểm có thể có nhiều yêu cầu ngắt từ các loại ngắt khác nhau, cùng đòi hỏi CPU phục vụ. CPU cùng một lúc không thể phục vụ tất cả các yêu cầu ngắt. Vì thế cần có một quy trình nhất định dựa trên một đặc tính tính ưu tiên nào đó mà lần lượt các yêu cầu ngắt sẽ được đáp ứng.

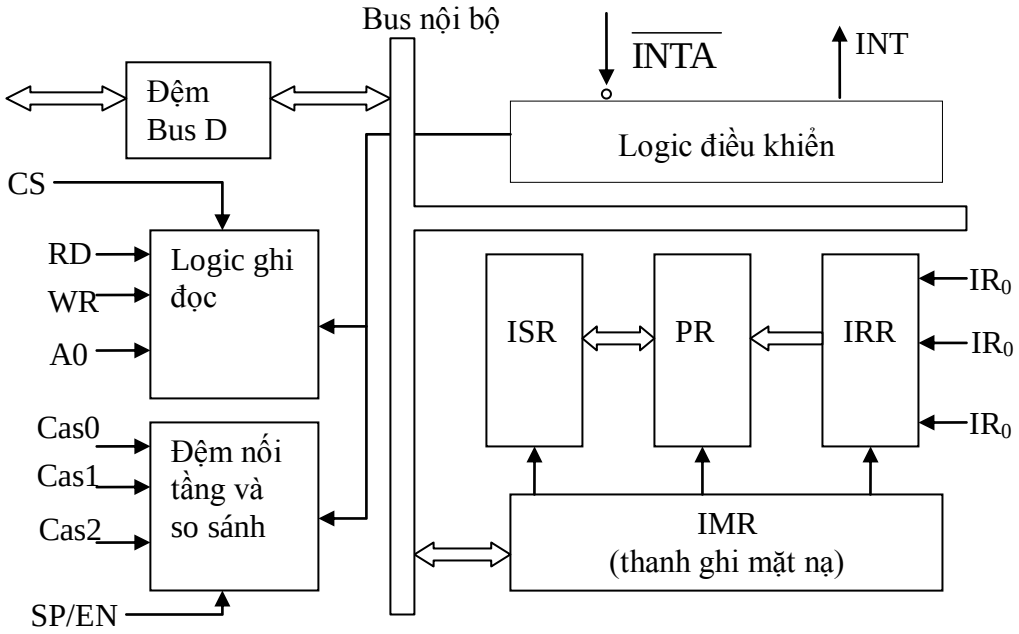
Bản thân 8088 cũng được thiết kế để có thể phân và nhận diện các mức ưu tiên ngắt theo tính chất quan trong của nó như sau:

Mức ưu tiên

- Ngắt nội bộ: INT 0 (phép chia cho 0). **…..Cao nhất**
- Ngắt không che được NMI
- Ngắt che được ỈNT
- Ngắt để chạy từng lệnh **….Thấp nhất**

Khi có nhiều yêu cầu ngắt cùng một lúc thì CPU lần lượt phục vụ theo mức ưu tiên giảm dần.

Trong trường hợp có nhiều ngắt che được từ bên ngoài phải phục vụ ta thường dùng vi mạch có sẵn 8259A để giải quyết vấn đề ưu tiên. Mạch 8259A được gọi là mạch điều khiển ngắt ưu tiên (priority interupt controller. PIC). Đó là 1 vi mạch cỡ lớn lập trình được, có thể xử lý trước được 8 yêu cầu ngắt với 8 mức ưu tiên khác nhau để tạo ra 1 yêu cầu ngắt đưa đến đầu vào INTR(yêu cầu ngắt che được) của CPU 8088. Nếu nối tầng 1 mạch 8259A chủ với 8 mạch 8259A ta có thể nâng tổng số các yêu cầu ngắt với các mức ưu tiên khác nhau lên thành 64.



Sơ đồ khối của PIC 8259A

Ghi chú:

- IR0-IR7: các yêu cầu ngắt.
- IRR: Thanh ghi yêu cầu ngắt.
- PR: Bộ xử lý ưu tiên.
- SP-EN: Slave program/Enable bufer(lập trình thành mạch thợ/mở đệm BUS dữ liệu).
- ISR: Thanh ghi các yêu cầu ngắt đang được phục vụ.
- Cas0-Cas2: Tín hiệu nối tầng giữa các PIC với nhau.

Các khối chức năng chính của 8259A

- Thanh ghi IRR: Ghi nhớ các yêu cầu ngắt có tại đầu vào IRi
- Thanh ghi ISR: Ghi nhớ các yêu cầu ngắt đang được phục vụ trong số các yêu cầu ngắt Iri.
- Thanh ghi IMR: ghi nhớ mặt nạ ngắt đối với các yêu cầu ngắt IRi.
- Logic điều khiển: Khối này có nhiệm vụ gửi yêu cầu ngắt tới INTR của 8088 khi có tín hiệu tại các chân IRi và nhận trả lời chấp nhận trả lời chấp nhận yêu cầu ngắt INTA từ CPU để rồi điều khiển việc đưa ra kiểu ngắt trên bus dữ liệu.
- Đệm bus dữ liệu: Dùng để phối ghép 8259A với bus dữ liệu của CPU.
- Logic điều khiển ghi/đọc: Dùng cho việc ghi các từ điều khiển và đọc các từ trạng thái của 8259A.
- Khối đệm nối tầng và so sánh: Ghi nhớ và so sánh số hiệu của các mạch 8259A có mặt trong hệ vi xử lý.

Các tín hiệu của 8259A:

Một số tín hiệu trong mạch 8259A có tên giống như các tín hiệu tiêu chuẩn của 8088 ta có thể thấy rõ và hiểu được ý nghĩa của chúng như ở hình trên. Ngoài các tín hiệu này ra còn có một số tín hiệu khác đặc biệt khác như sau:

Cas0-Cas2 [I.O]: Là các đầu vào đối với các mạch 8259A thợ hoặc các đầu ra của mạch 8259A chủ dùng khi cần nối tầng.

$\overline{SP}$ / $\overline{EN}$ [I.O]: Khi 8259A làm việc ở chế độ không có đệm bus dữ liệu thì đây là tín hiệu vào dùng lập trình để biến mạch 8259A thành mạch thợ ($\overline{SP}$ =0) hoặc chủ ($\overline{SP}$ =1). Khi 8259A làm việc trong hệ vi xử lý ở chế độ có đệm bus dữ liệu thì chân này là tín hiệu ra $\overline{EN}$ dùng mở đệm bus dữ liệu để 8088 và 8259 thông vào bus dữ liệu hệ thống. Việc định nghĩa 8259A là chủ hoặc thợ phải thực hiện thông qua từ điều khiển ICW4.

- INT [O]: Tín hiệu yêu cầu ngắt đến chân INTR của 8088.
- $\overline{INTA}$ [I]: Nối với tín hiệu báo chấp nhận ngắt INTA của CPU.

BỘ GIAO DỤC & ĐAO TẠO
TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHỆ
THÀNH PHỐ HỒ CHÍ MINH
KHOA ĐIỆN – ĐIỆN TỬ
--- oOo ---

GIÁO TRÌNH

VI XỬ LÝ

Tác giả: ThS. PHẠM HÙNG KIM KHÁNH

08/2006

LỜI NÓI ĐẦU

Giáo trình Vi xử lý được biên soạn nhằm cung cấp cho sinh viên kiến thức cơ bản về vi xử lý, cấu trúc của một hệ vi xử lý cũng như cách thức lập trình điều khiển thiết bị dựa cơ sở trên Vi xử lý 8086/8088.

Giáo trình được sử dụng cho khóa học 60 tiết dành cho sinh viên hệ đại học Khoa Điện Điện tử trường Đại học Dân lập Kỹ thuật Công nghệ TPHCM.

Bố cục giáo trình gồm 4 chương dựa theo đề cương môn học Kỹ thuật Vi xử lý dành cho sinh viên ngành Điện Tử Viễn Thông:

Chương 1. Tổ chức hệ thống Vi xử lý

Chương 2. Lập trình hợp ngữ

Chương 3. Tổ chức nhập / xuất

Chương 4. Giao tiếp với các thiết bị đơn giản

Phụ lục 1: 8255

Phụ lục 2: Tập lệnh của họ 8086

PHẠM HÙNG KIM KHÁNH

MỤC LỤC

CHƯƠNG 1: TỔ CHỨC HỆ THỐNG VI XỬ LÝ

1. Các hệ thống số dùng trong máy tính và các loại mã

1.1. Hệ thập phân (Decimal Number System)

Trong thực tế, ta thường dùng hệ thập phân để biểu diễn các giá trị số. Ở hệ thống này, ta dùng các tổ hợp của các chữ số 0..9 để biểu diễn các giá trị. Một số trong hệ thập phân được biểu diễn theo các số mũ của 10.

VD: Số 5346.72 biểu diễn như sau:

$5346.72 = 5x10^3 + 3x10^2 + 4x10 + 6 + 7x10^{-1} + 2x10^{-2}$

Tuy nhiên, trong các mạch điện tử, việc lưu trữ và phân biệt 10 mức điện áp khác nhau rất khó khăn nhưng việc phân biệt hai mức điện áp thì lại dễ dàng. Do đó, người ta sử dụng hệ nhị phân để biểu diễn các giá trị trong hệ thống số.

1.2. Hệ nhị phân (Binary Number System)

Hệ nhị phân chỉ dùng các chữ số 0 và 1 để biểu diễn các giá trị số. Một số nhị phân (binary digit) thường được gọi là ***bit***. Một chuỗi gồm 4 bit nhị phân gọi là ***nibble***, chuỗi 8 bit gọi là ***byte***, chuỗi 16 bit gọi là ***word*** và chuỗi 32 bit gọi là ***double word***. Chữ số nhị phân bên phải nhất của chuỗi bit gọi là ***bit có ý nghĩa nhỏ nhất*** (***least significant bit – LSB***) và chữ số nhị phân bên trái nhất của chuỗi bit gọi là ***bit có ý nghĩa lớn nhất*** (***most significant bit – MSB***). Một số trong hệ nhị phân được biểu diễn theo số mũ của 2. Ta thường dùng chữ ***b*** cuối chuỗi bit để xác định đó là số nhị phân.

VD: Số 101110.01b biểu diễn giá trị số:

$101110.01b \rightarrow 1x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 1x2^1 + 0 + 0x2^{-1} + 1x2^{-2}$

❖ **Chuyển số nhị phân thành số thập phân:**

Để chuyển một số nhị phân thành một số thập phân, ta chỉ cần nhân các chữ số của số nhị phân với giá trị thập phân của nó và cộng tất cả các giá trị lại.

VD: $1011.11B \rightarrow 1x2^3 + 0x2^2 + 1x2^1 + 1 + 1x2^{-1} + 1x2^{-2} = 11.75$

❖ **Chuyển số thập phân thành số nhị phân:**

Để chuyển một số thập phân thành số nhị phân, ta dùng 2 phương pháp sau:

- **Phương pháp 1**: Ta lấy số thập phân cần chuyển trừ đi 2^i trong đó 2^i là số lớn nhất nhỏ hơn hay bằng số thập phân cần chuyển. Sau đó, ta lại lấy kết quả này và thực hiện tương tự cho đến 2^0 thì dừng. Trong quá trình thực hiện, ta sẽ ghi lại các giá trị 0 hay 1 cho các bit tuỳ theo trường hợp số thập phân nhỏ hơn 2^i (0) hay lớn hơn 2^i (1).

VD: Xét số 21 thì số 2^i lớn nhất là 2^4

	2^4	2^3	2^2	2^1	2^0
	16	8	4	2	1
21 =	1	0	1	0	1
	5	5	1	1	0

(21 → 10101B)

- **Phương pháp 2**: Lấy số cần chuyển chia cho 2, ta nhớ lại số dư và lấy tiếp thương của kết quả trên chia cho 2 và thực hiện tương tự cho đến khi thương cuối cùng bằng 0. Kết quả chuyển đổi sẽ là chuỗi các bit là các số dư lấy theo thứ tự ngược lại.

VD: Chuyển 227 ra số nhị phân

Số bị chia	Thương	Số dư
227	113	1 (LSB)
113	56	1
56	28	0
28	14	0
14	7	0
7	3	1
3	1	1
1	0	1 (MSB)

(227 → 11100011b)

- Để thực hiện chuyển các số thập phân nhỏ hơn 1 sang các số nhị phân, ta làm như sau: lấy số cần chuyển nhân với 2, giữ lại phần nguyên và lại lấy phần lẻ nhân với 2. Quá trình tiếp tục cho đến khi phần lẻ bằng 0 thì dừng. Kết quả chuyển đổi là chuỗi các bit là giá trị các phần nguyên.

VD: Chuyển 0.625 thành số nhị phân:

0.625 × 2 = 1.25
0.25 × 2 = 0.5
0.5 × 2 = 1.0
(0.625 = 0.101b)

- Để thực hiện chuyển đổi số nhị phân bất kỳ, ta thực hiện chuyển đổi tương ứng với số nhị phân lớn hơn 1 và nhỏ hơn 1 như trên.

VD: Chuyển 227.625 thành số nhị phân:

227 → 11100011b
0.625 → 0.101b
227.625 → 11100011.101b

1.3. Hệ thập lục phân (Hexadecimal Number System)

Như đã biết ở trên, nếu dùng hệ nhị phân thì sẽ cần một số lượng lớn các bit để biểu diễn. Giả sử như số $1024 = 2^{10}$ sẽ cần 10 bit để biểu diễn. Để rút ngắn kết quả

biểu diễn, ta dùng hệ thập lục phân dựa cơ sở trên số mũ của 16. Khi đó, 4 bit trong hệ nhị phân (1 nibble) sẽ biểu diễn bằng 1 chữ số trong hệ thập lục phân (gọi là số hex).

Trong hệ thống này, ta dùng các số 0..9 và các kí tự A..F để biểu diễn cho một giá trị số. Thông thường, ta dùng chữ ***h*** ở cuối để xác định đó là số thập lục phân.

1.4. Mã BCD (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

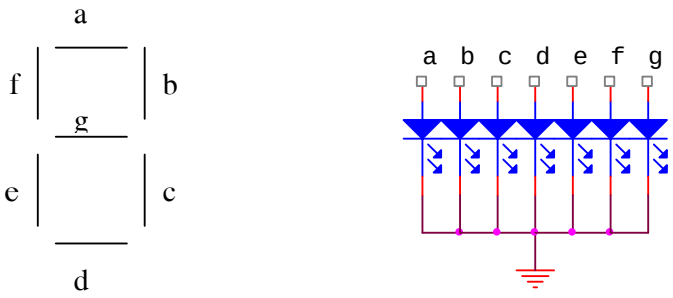
VD: Số thập phân 5 2 9

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

Số BCD nén 0101b 0010b 1001b

1.5. Mã hiển thị Led 7 đoạn (7-segment display)

Đối với các ứng dụng dùng hiển thị số liệu ra Led 7 đoạn, ta dùng mã hiển thị Led 7 đoạn. Ứng với mỗi loại Led 7 đoạn (anode hay cathode chung) và tuỳ theo sơ đồ kết nối sẽ có một bảng mã riêng. Một ví dụ của mã Led 7 đoạn cho trong bảng 1.1.



Hình 1.1 – Led 7 đoạn dạng cathode chung

Bảng 1.1:

Số thập phân	Số thập lục phân	Số nhị phân	Mã Led 7 đoạn	
			a b c d e f g	Hiển thị
0	0	0000	1 1 1 1 1 1 0	0
1	1	0001	0 1 1 0 0 0 0	1
2	2	0010	1 1 0 1 1 0 1	2
3	3	0011	1 1 1 1 0 1 1	3
4	4	0100	0 1 1 0 0 1 1	4
5	5	0101	1 0 1 1 0 1 1	5
6	6	0110	1 0 1 1 1 1 1	6
7	7	0111	1 1 1 0 0 0 0	7

8	8	1000	1 1 1 1 1 1 1	8
9	9	1001	1 1 1 0 0 1 1	9
10	A	1010	1 1 1 1 1 0 1	A
11	B	1011	0 0 1 1 1 1 1	B
12	C	1100	0 0 0 1 1 0 1	C
13	D	1101	0 1 1 1 1 0 1	D
14	E	1110	1 1 0 1 1 1 1	E
15	F	1111	1 0 0 0 1 1 1	F

2. Các phép toán số học

2.1. Hệ nhị phân

2.1.1. Phép cộng

Phép cộng trong hệ nhị phân cũng thực hiện giống như trong hệ thập phân. Bảng sự thật của phép cộng 2 bit với 1 bit nhớ (carry) như sau:

Bảng 1.2:

Vào			Ra	
A	**B**	**C_{IN}**	**S**	**C_{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

$S = A \oplus B \oplus C_{IN}$

$C_{OUT} = AB + C_{IN}(A \oplus B)$

VD: 1001 1010b

1 +1100 1100b

Nhớ 0111 0110b

2.1.2. Số bù 2 (2's component)

Trong hệ thống số thông thường, để biểu diễn số âm ta chỉ cần thêm dấu – vào các chữ số. Tuy nhiên, trong hệ thống máy tính, ta không thể biểu diễn được như trên. Phương pháp thông dụng là dùng bit có ý nghĩa lớn nhất (MSB) làm bit dấu (sign bit): nếu MSB = 1 sẽ là số âm còn MSB = 0 là số dương. Khi đó, các bit còn lại sẽ biểu diễn độ lớn (magnitude) của số. Như vậy, nếu ta dùng 8 bit để biểu diễn thì sẽ thu được 256 tổ hợp ứng với các giá trị 0..255 (số không dấu) hay –127.. –0 +0 … +127 (số có dấu).

Để thuận tiện hơn trong việc tính toán số có dấu, ta dùng một dạng biểu diễn đặc biệt là số bù 2. Số bù 2 của một số nhị phân xác định bằng cách lấy đảo các bit rồi cộng thêm 1.

VD: Số 7 biểu diễn là : 0000 0111b có MSB = 0 (biểu diễn số dương)

Số bù 2 là : 111 1000b + 1b = 111 1001b. Số đại diện cho số – 7 là: 1111 1001b có MSB = 1 (biểu diễn số âm)

Ta thấy, để thực hiện việc xác định số bù 2 của một số A, cần phải:

- Biểu diễn số A theo mã bù 2 của nó.
- Đảo các bit (tìm số bù 1 của A).
- Cộng thêm 1 vào để nhận được số bù 2.

Khi biểu diễn theo số bù 2, nếu sử dụng 8 bit ta sẽ có các giá trị số thay đổi từ -128..127.

2.1.3. Phép trừ

Phép trừ các số nhị phân cũng được thực hiện tương tự như trong hệ thập phân. Bảng sự thật của phép trừ 2 bit với 1 bit mượn (borrow) như sau:

Bảng 1.3:

Vào			Ra	
A	**B**	**BIN**	**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

$S = A \oplus B \oplus BIN$

$BOUT = \overline{A}B + \overline{(A \oplus B)}B_{IN}$

VD: 0110 1101b → 149

- 0011 0001b → 49

0011 1100b → 100

Ngoài cách trừ như trên, ta cũng có thể thực hiện phép trừ thông qua số bù 2 của số trừ.

VD: 0110 1101b

- 0011 0001b → + 1100 1111b

1 0011 1100b

Số bù 1 Nhớ

100 1110b + 1b = 100 1111b (Số bù 2)

Trong phép cộng với số bù 2, ta bỏ qua bit nhớ cuối cùng → kết quả phép cộng số bù 2 là 0011 1100. Đây cũng chính là kết quả phép trừ, bit MSB = 0 cho biết kết quả là số dương.

VD:

77	0100 1101b		0100 1101b
- 88	- 0101 1000b	→	+ 1010 1000b
- 11			1111 0101b

Số 88 → 0101 1000b → số bù 1 là 010 0111 → số bù 2: 010 1000 và bit dấu = 1 → -88 trở thành 1010 1000b

Kết quả phép cộng số bù 2 là 1111 0101b có MSB = 1 nên là số âm. Số bù 1 là 000 1010b → số bù 2: 000 1011b. Kết quả này chính là 11 nên phép trừ sẽ cho kết quả là –11.

Ta thấy, để thực hiện chuyển số bù 2 thành số có dấu thì cần thực hiện:

- Lấy bù các bit để tìm số bù 1.
- Cộng với 1.
- Thêm dấu trừ để xác định là số âm.

2.1.4. Phép nhân

Phép nhân các số nhị phân cũng tương tự như đối với các số thập phân. Chú ý rằng đối với phép nhân nếu nhân 2 số 4 bit sẽ có kết quả là số 8 bit, 2 số 8 bit sẽ có kết quả là số 16 bit, …

VD:

11	1011b
X 9	1001b
99	1011
	0000
	0000
	1011
	1100011b

Đối với máy tính, phép nhân được thực hiện bằng phương pháp cộng và dịch phải (add-and-right-shift):

- Thành phần dầu tiên của tổng sẽ chính là số bị nhân nếu như LSB của số nhân là 1. Ngược lại, nếu LSB của số nhân bằng 0 thì thành phần này bằng 0.
- Mỗi thành phần thứ i kế tiếp sẽ được tính tương tự với điều kiện là phải dịch trái số bị nhân i bit.
- Kết quả cần tìm chính là tổng các thành phần nói trên.

2.1.5. Phép chia

Phép chia các số nhị phân cũng tương tự như đối với các số thập phân.

VD: 30/5 = 6

```
11110 b  | 110b
110      | 101b
 011     |
 000     |
  110    |
  110    |
    0    |
```

Tương tự như đối với phép nhân, ta có thể dùng phép trừ và phép dịch trái cho đến khi không thể thực hiện phép trừ được nữa. Tuy nhiên, để thuận tiện cho tính toán, thay vì dùng phép trừ đối với số chia, ta sẽ thực hiện phép cộng đối với số bù 2 của số chia.

- Đổi số chia ra số bù 2 của nó.
- Lấy số bị chia cộng với số bù 2 của số chia.
  + Nếu kết quả này có bit dấu = 0 thì bit tương ứng của thương = 1.
  + Nếu kết quả này có bit dấu = 1 thì bit tương ứng của thương = 0 và ta phải khôi phục lại giá trị của số bị chia bằng cách cộng kết quả này với số chia.
- Dịch trái kết quả thu được và thực hiện tiếp tục như trên cho đến khi kết quả là 0 hay nhỏ hơn số chia.

2.2. Hệ thập lục phân

2.2.1. Phép cộng

Thực hiện chuyển các số hex cần cộng thành các số nhị phân, tính kết quả trên số nhị phân và sau đó chuyển lại thành số hex.

VD: 7Ah → 0111 1010b

3Fh → 0011 1111b

B9h ← 1011 1001b

Thực hiện cộng trực tiếp trên số hex, nếu kết quả cộng lớn hơn 15 thì sẽ nhớ và trừ cho 16.

VD: 7 Ah

3 Fh

10_{10} 25_{10} → B9h

Ah + Fh = $10_{10} + 15_{10} = 25_{10}$ → nhớ 1 và $25_{10} - 16_{10} = 9_{10} = 9h$

7h + 3h = $7_{10} + 3_{10} = 10_{10}$ → cộng số nhớ: $10_{10} + 1_{10} = 11_{10} = Bh$

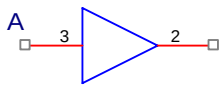
2.2.2. Phép trừ

Thực hiện tương tự như phép cộng.

3. Các thiết bị số cơ bản

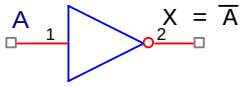
3.1. Cổng đệm (buffer) và các cổng logic (logic gate)

❖ Cổng đệm:



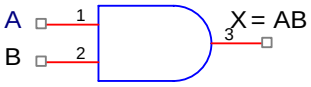
A	X
0	0
1	1

❖ Cổng NOT:



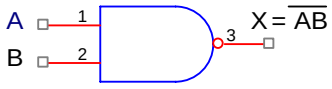
A	X
0	1
1	0

❖ Cổng AND:



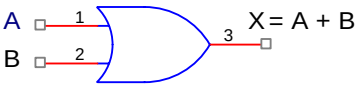
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

❖ Cổng NAND:



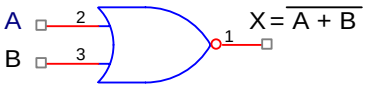
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

❖ Cổng OR:



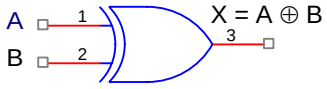
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

❖ Cổng NOR:



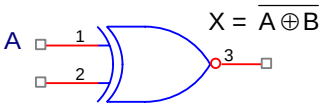
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

❖ **Cổng EX-OR:**



A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

❖ **Cổng EX-NOR:**



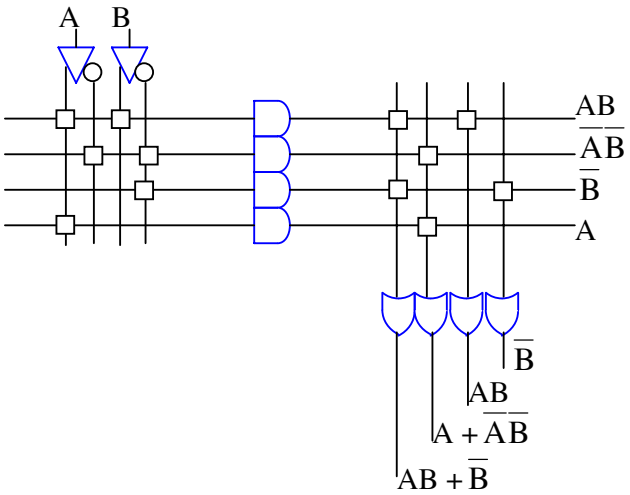
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

3.2. Thiết bị logic lập trình được

Thay vì sử dụng các cổng logic rời rạc, ta có thể dùng các thiết bị logic lập trình được (programmable logic device) như PLA (Programmable Logic Array), PAL (Programmable Array of Logic) để liên kết các thiết bị LSI (Large Scale Intergration).

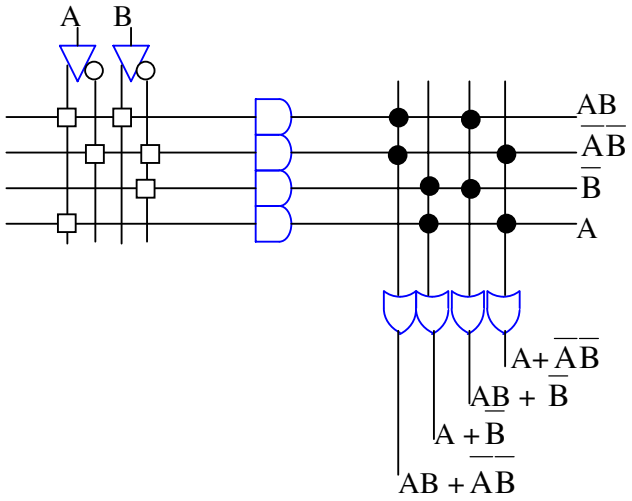
❖ **PLA (hay FPLA – Field PLA):**

Dùng ma trận cổng AND và OR để lập trình bằng cácc phá huỷ các cầu chì. FPLA rất linh động nhưng lại khó lập trình.



Hình 1.2 – Sơ đồ PLA

❖ **PAL**: ma trận OR đã cố định sẵn và ta chỉ lập trình trên ma trận AND.

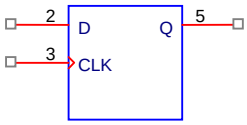


Hình 1.3 – Sơ đồ PAL

3.3. Chốt, flipflop và thanh ghi

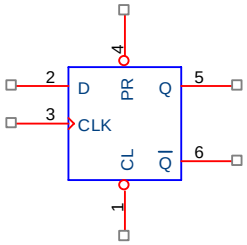
❖ **Chốt (latch):**

Chốt là thiết bị số lưu trữ lại giá trị số tại ngõ ra của nó.



D	CLK	Q
X	0	QN
0	1	0
1	1	1

❖ **Flipflop:**



PR	CL	D	CLK	Q	$\overline{Q}$
1	1	1	↑	1	0
1	1	0	↑	0	1
1	1	X	0	Q_N	$\overline{Q_N}$
1	1	X	1	Q_N	$\overline{Q_N}$
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	.	.

CL: clear PR: Preset CLK: Clock

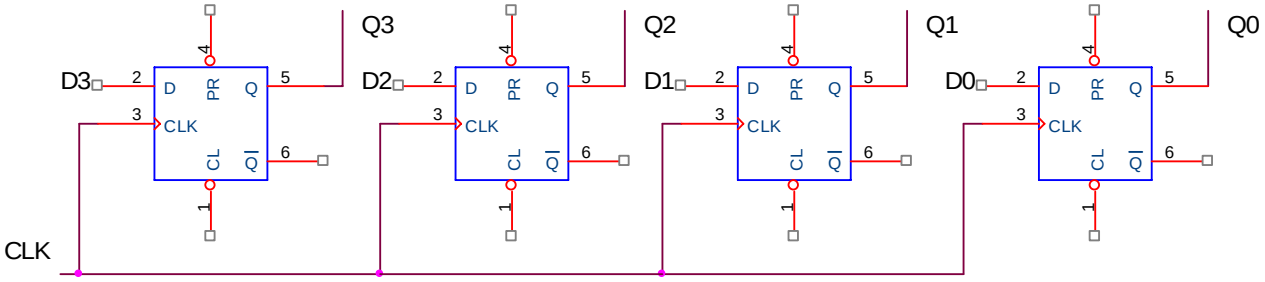
- Nếu xuất hiện cạnh lên của tín hiệu CLK thì ngõ ra Q sẽ có giá trị theo dữ liệu tại D.

- Nếu PR = 0 thì Q = 1. Nếu CL = 0 thì Q = 0.

- Trạng thái PR = CL = 0 là trạng thái cấm, ngõ ra sẽ không ổn định.

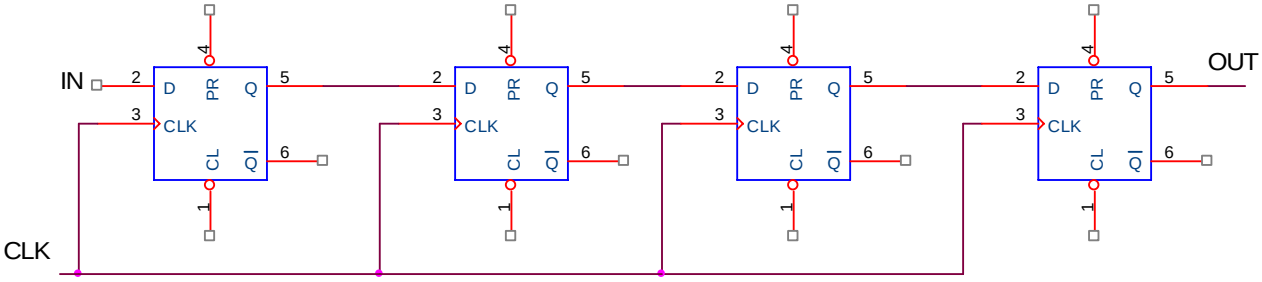
❖ **Thanh ghi (register):**

Thanh ghi là một nhóm các flipflop được kết nối song song để lưu trữ các số nhị phân. Giá trị nhị phân sẽ được đưa vào ngõ vào của các flipflop. Khi có tác động cạnh lên của tín hiệu CLK thì ngõ ra các flipflop sẽ lưu trữ giá trị nhị phân cho đến khi một số nhị phân mới được đưa vào và tác động một cạnh len cho tín hiệu CLK.



Hình 1.4 – Thanh ghi dạng đơn giản

Trong trường hợp các flipflop được kết nối nối tiếp với nhau, ta sẽ có thanh ghi dịch (shift register).



Hình 1.5 – Thanh ghi dịch

3.4. Bộ nhớ

3.4.1. Các kiểu bộ nhớ

❖ **ROM (Read Only Memory):**

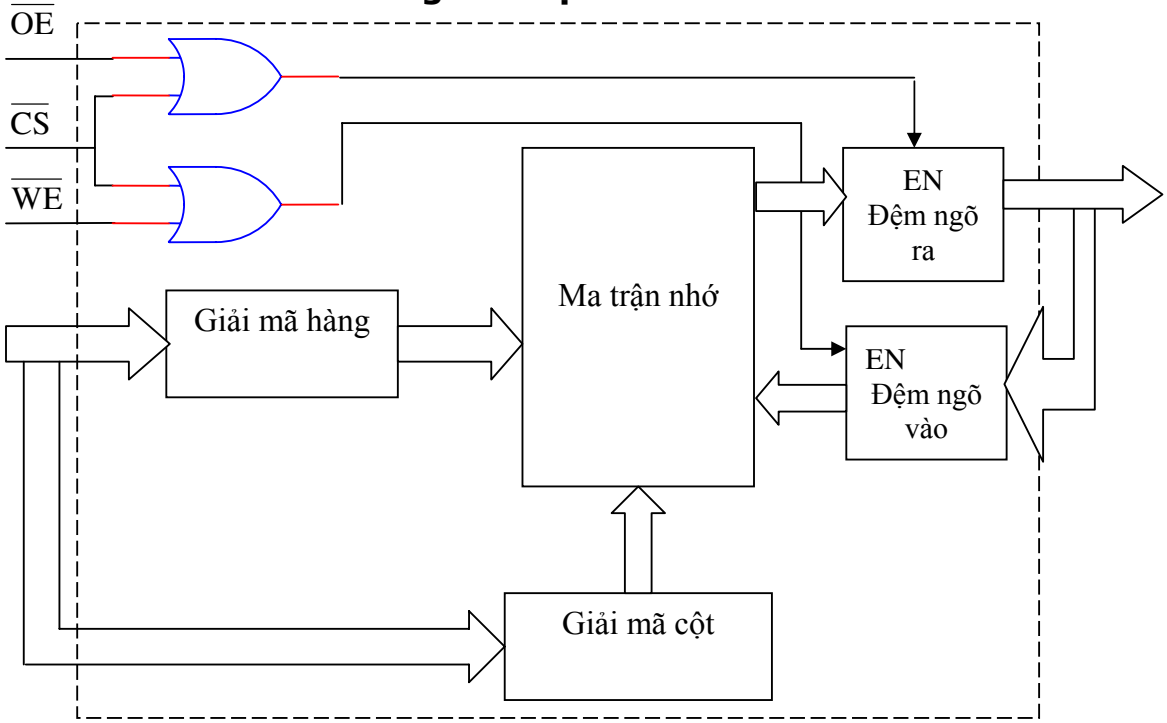
Đặc tính chung của ROM là dữ liệu lưu trữ sẽ không bị mất đi dù cho không còn nguồn cung cấp cho ROM (tính nonvolatile – ổn định). Ta chỉ có thể thực hiện tác vụ đọc đối với ROM. ROM có thể được chia thành: ROM che mặt nạ (Masked ROM), PROM (ROM lập trình được), EPROM (ROM có thể xoá bằng tia cực tím) và EEPROM (ROM có thể xoá bằng điện).

❖ **RAM (Random Access Memory):**

RAM có đặc tính là tất cả nội dung chứa trong RAM sẽ bị mất đi khi không còn nguồn cung cấp cho RAM (tính volatile – không ổn định). Có 2 loại RAM: tĩnh và động.

- **SRAM (Static RAM):** dùng các ma trận flipflop để lưu trữ dữ liệu nên ta có thể ghi các giá trị nhị phân vào RAM bằng cách đưa dữ liệu vào các ngõ vào các flipflop và cấp xung clock cho các flipflop này.
- **DRAM (Dynamic RAM):** tạo ra bằng các cổng transistor và lưu trữ bằng điện tích. Tuy nhiên, do hiện tượng rò rỉ điện tích theo thời gian, ta phải thực hiện nạp điện lại. Quá trình này gọi là làm tươi (refreshing) bộ nhớ. Thuận lợi của DRAM là một số lượng lớn transistor có thể được đặt trên một chip nhớ nên nó có dung lượng cao hơn và nhanh hơn SRAM.

3.4.2. Cấu trúc bên trong của bộ nhớ



Hình 1.6 – Cấu trúc nội một bộ nhớ tiêu biểu

$\overline{CS}$ (Chip Select):cho phép bộ nhớ hoạt động

$\overline{OE}$ (Output Enable): cho phép đọc dữ liệu từ bộ nhớ ra ngoài

$\overline{WE}$ (Write Enable): cho phép ghi dữ liệu vào trong bộ nhớ

4. Giới thiệu vi xử lý

4.1. Các thế hệ vi xử lý

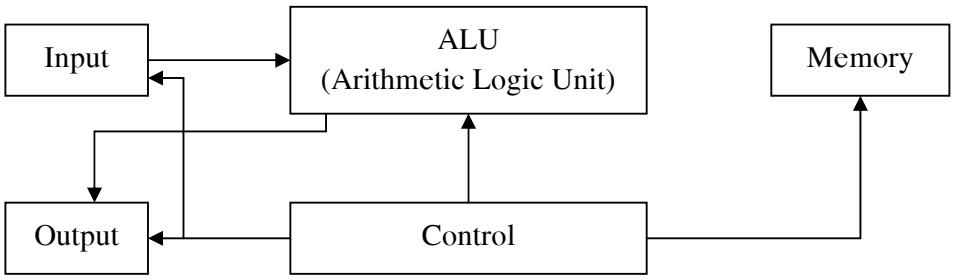
- ***Thế hệ 1 (1971 – 1973):*** vi xử lý 4 bit, đại diện là 4004, 4040, 8080 (Intel) hay IPM-16 (National Semiconductor).
  + Độ dài word thường là 4 bit (có thể lớn hơn).
  + Chế tạo bằng công nghệ PMOS với mật độ phần tử nhỏ, tốc độ thấp, dòng tải thấp nhưng giá thành rẻ.
  + Tốc độ 10 ÷ 60 μs / lệnh 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 (Zilog).
  + 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

4.2. Vi xử lý (μP – microproccessor)

4.2.1. Phân loại vi xử lý

- ***Multi chip***: dùng 2 hay nhiều chip LSI (Large Scale Intergration: tích hợp từ 1000 ÷ 10000 transistor) cho ALU và control.
- ***Microprocessor***: dùng 1 chip LSI/VLSI (Very Large Scale Intergration: tích hợp ÷ 10000 transistor) cho ALU và control.
- ***Single chip microprocessor*** (còn gọi là microcomputer / microcontroller): là 1 chip LSI/VLSI chứa toàn bộ các khối như hình 1.7.

4.2.2. Sơ đồ khối một máy tính cổ điển



Hình 1.7 – Sơ đồ khối một máy tính cổ điển

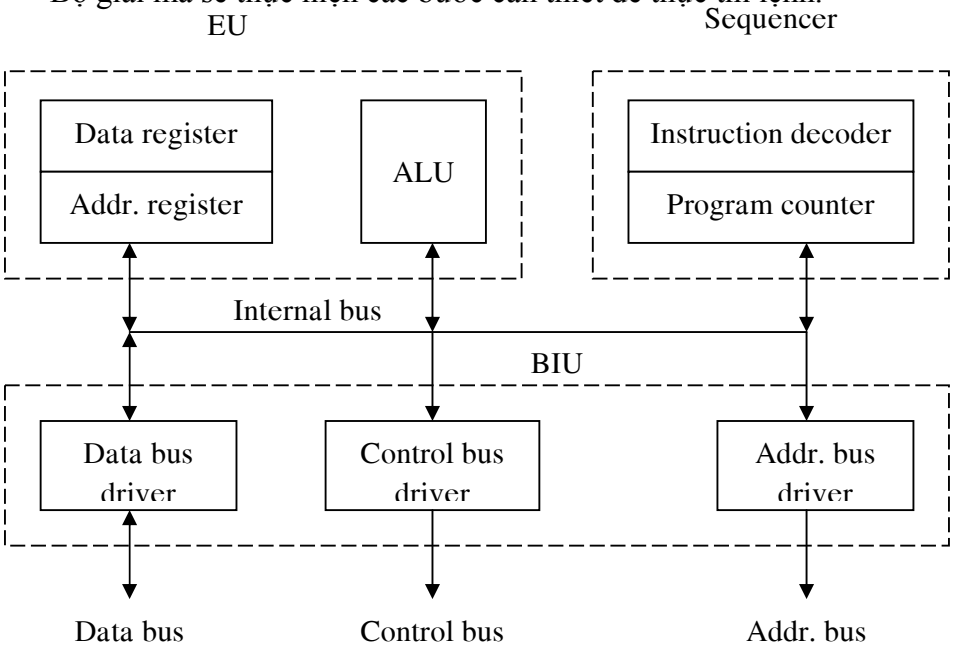
- *ALU (đơn vị logic số học)*: thực hiện các bài toán cho máy tính bao gồm: +, -, *, /, phép toán logic, …
- *Control (điều khiển)*: điều khiển, kiểm soát các đường dữ liệu giữa các thành phần của máy tính.
- *Memory (bộ nhớ)*: lưu trữ chương trình hay các kết quả trung gian.
- *Input (nhập), Output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

4.2.3. Sơ đồ khối của μP

Có 3 khối chức năng: đơn vị thực thi (EU - Execution unit), bộ tuần tự (Sequencer) và đơn vị giao tiếp bus (BIU – Bus interface unit).

- EU: thực hiện các lệnh số học và logic. Các toán hạng được chứa trong các thanh ghi dữ liệu (data register) hay thanh ghi địa chỉ (address register), hay từ bus nội (internal bus).
- Bộ tuần tự: gồm bộ giải mã lệnh (instruction decoder) và bộ đếm chương trình (program counter)

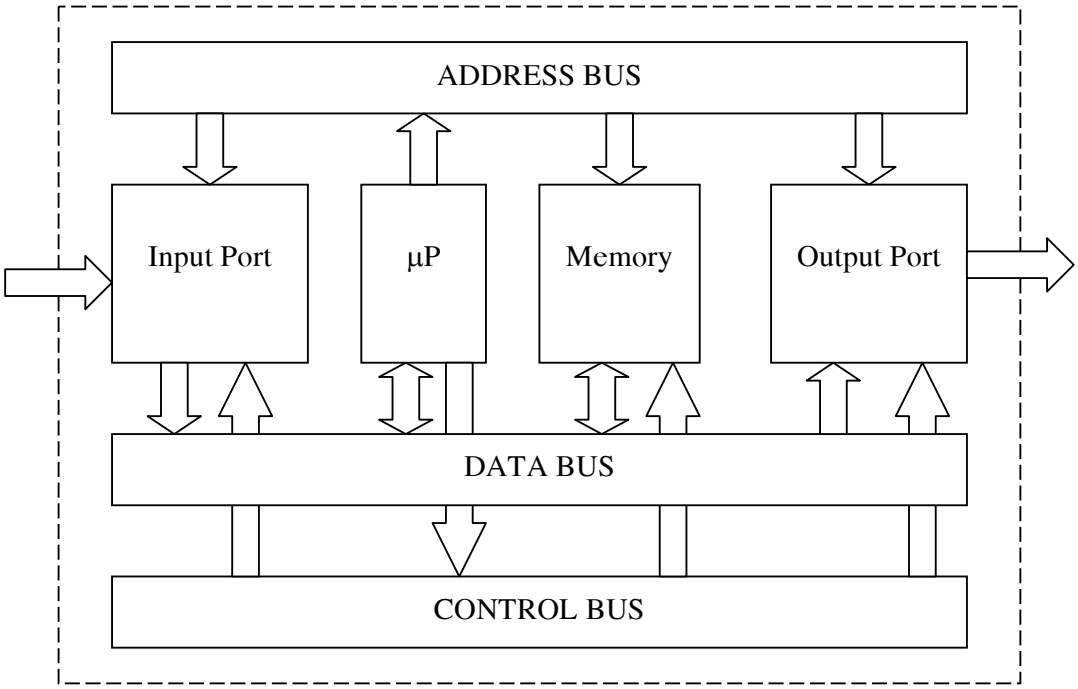
+ Bộ đếm chương trình chứa các lệnh kế tiếp sẽ thực hiện
+ Bộ giải mã sẽ thực hiện các bước cần thiết để thực thi lệnh.



Hình 1.8 – Sơ đồ khối của vi xử lý

Khi chương trình bắt đầu, bộ đếm chương trình (PC) sẽ ở địa chỉ bắt đầu. Địa chỉ này được chuyển qua bộ nhớ thông qua address bus. Khi tín hiệu Read đưa vào control bus, nội dung bộ nhớ liên quan sẽ đưa vào bộ giải mã lệnh. Bộ giải mã lệnh sẽ khởi động các phép toán cần thiết để thực thi lệnh. Quá trình này đòi hỏi một số chu kỳ máy (machine cycle) tuỳ theo lệnh. Sau khi lệnh đã thực thi, bộ giải mã lệnh sẽ đặt PC đến địa chỉ của lệnh kế.

4.2.4. Sơ đồ khối của hệ vi xử lý cơ bản



Hình 1.9 – Sơ đồ khối hệ vi xử lý

Mọi hoạt động cơ bản của một hệ vi xử lý đều giống nhau, không phụ thuộc loại vi xử lý hay quá trình thực hiện. µP sẽ đọc một lệnh từ bộ nhớ (memory), thực thi lệnh và sau đó đọc lệnh kế. Quá trình đọc lệnh gọi là instruction fetch còn quá trình thực hiện tuần tự như trên gọi là fetch – execute sequence. Tuy nhiên có một số µP sẽ nhận một số lệnh rồi mới bắt đầu thực thi.

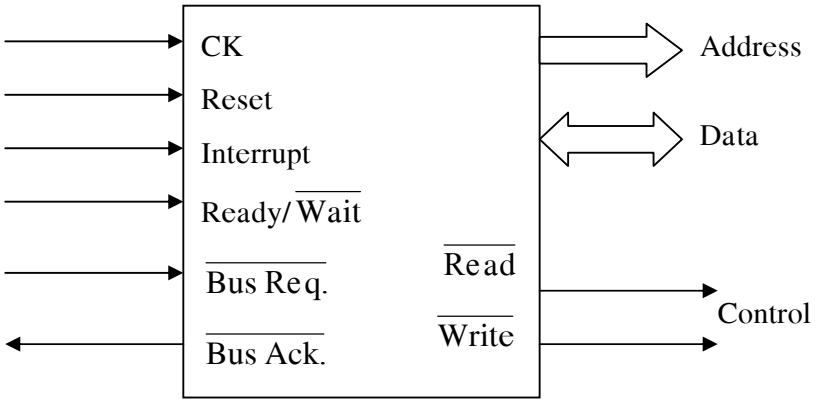
❖ **Các port I/O:**

Các port nhập (input) và xuất (output) dùng để giao tiếp giữa µP và thiết bị ngoại vi (không thể nối trực tiếp với các bus).

Port xuất là một thanh ghi. Khi µP ghi dữ liệu ra địa chỉ của Port thì Port sẽ chứa dữ liệu hiện tại trên data bus. Dữ liệu này sẽ được chốt tại Port cho đến khi µP ghi dữ liệu mới ra Port.

Port nhập là một driver 3 trạng thái. Khi µP đọc vào từ địa chỉ của Port, driver 3 trạng thái lái dữ liệu từ bên ngoài vào data bus. Sau đó, µP đọc dữ liệu từ bus.

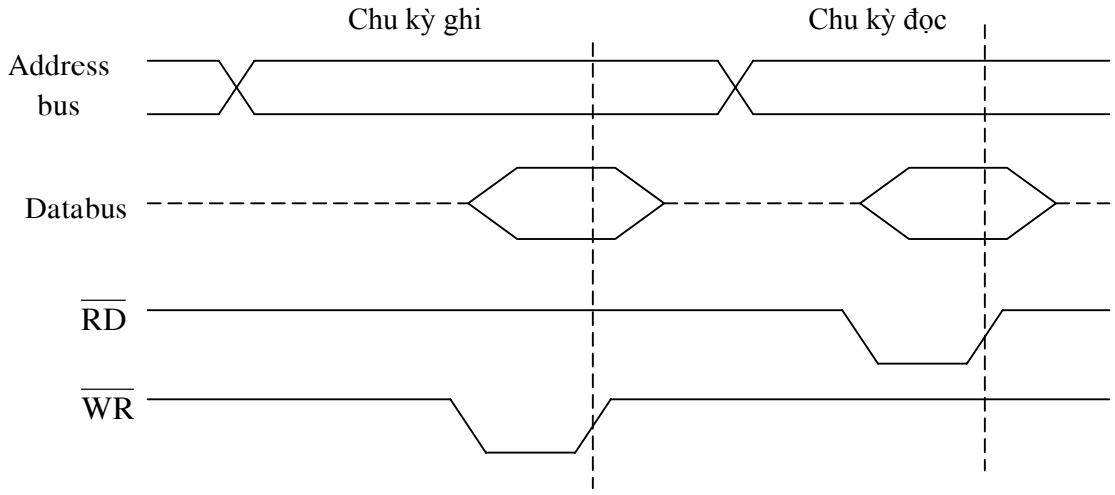
❖ **Các tín hiệu tiêu biểu của một µP:**



Hình 1.10 – Các tín hiệu cơ bản trong µP

Các bus dùng để liên kết các thành phần của hệ thống với µP. µP sẽ chọn một thiết bị cần sử dụng thông qua address bus và đọc hay ghi dữ liệu thông qua data bus. Data bus là bus 2 chiều, dùng chung cho tất cả các quá trình trao đổi dữ liệu. Mỗi chu kỳ bus (bus cycle) là việc thực hiện trao đổi một từ dữ liệu giữa µP và ô nhớ hay thiết bị I/O.

Mỗi chu kỳ bus bắt đầu khi µP xuất một địa chỉ nhằm chọn thiết bị I/O hay chọn một ô nhớ nào đó.



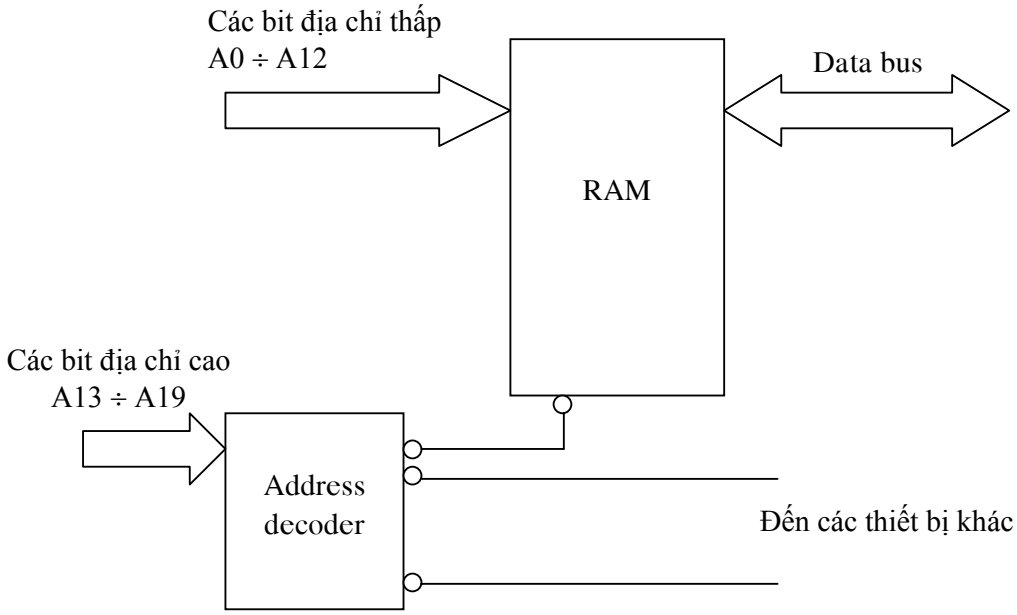
Hình 1.11 – Định thì bus cơ bản

4.3. Giao tiếp với bộ nhớ

4.3.1. Giao tiếp bus cơ bản

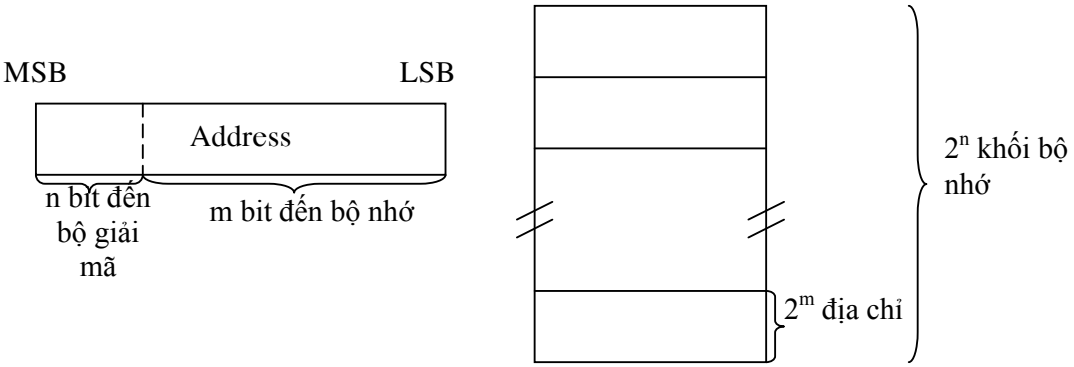
- Các bit địa chỉ thấp (giả sử 13 đường A0 ÷ A12) nối trực tiếp đến chip bộ nhớ (giả sử RAM có dung lượng 8K × 8)

- Các bit địa chỉ cao (giả sử A13 ÷ A19) nối với bộ giải mã địa chỉ (address decoder) tạo tín hiệu cho phép chip bộ nhớ. Do đó, khi thiết kế ta phải xác 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 1.12 – Giao tiếp bus cơ bản

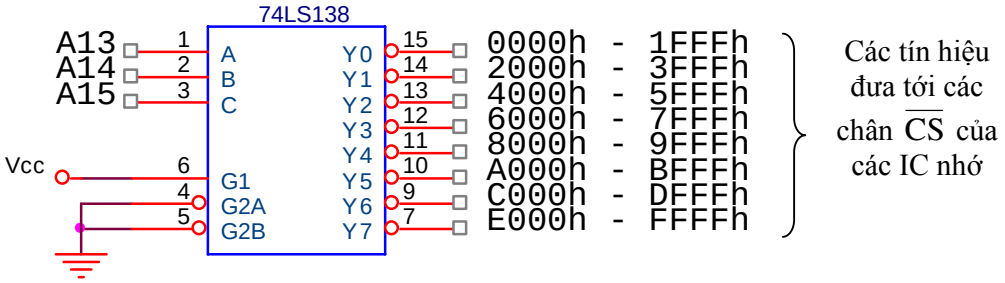
Quan hệ giữa giải mã địa chỉ và bảng bộ nhớ:



Hình 1.13 – Bảng bộ nhớ

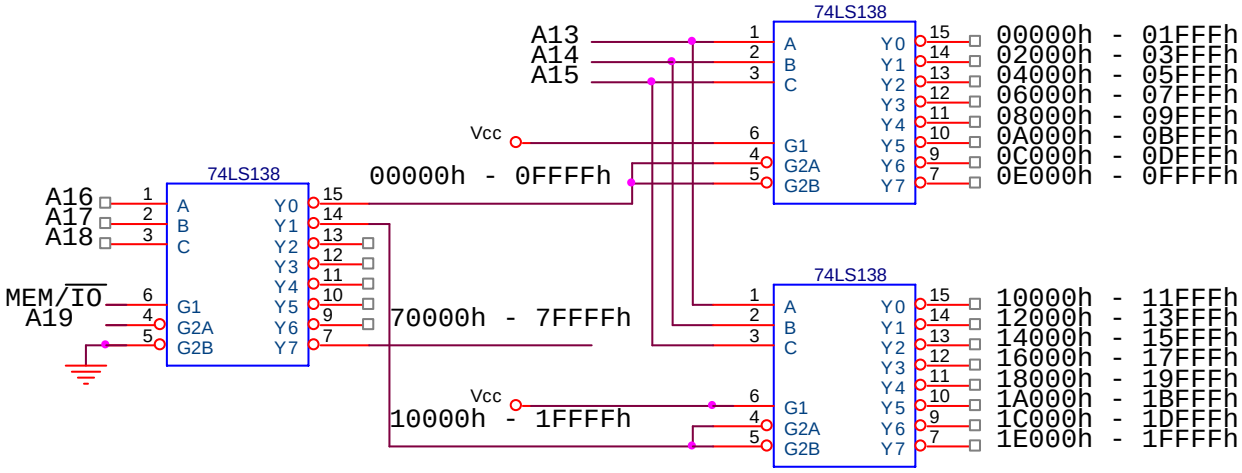
4.3.2. Giải mã địa chỉ

4.3.2.1. Dùng 74LS138



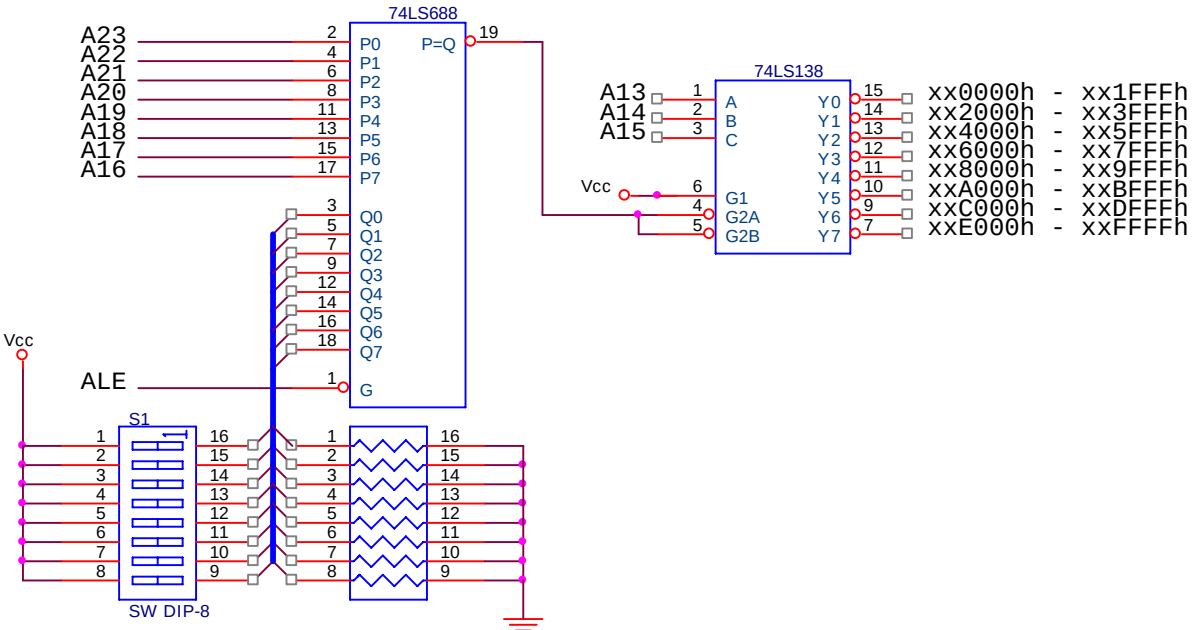
Hình 1.14 – Giải mã địa chỉ dùng 74LS138

4.3.2.2. Dùng nhiều 74LS138



Hình 1.15 – 74LS138 mắc cascaded (liên tầng)

4.3.2.3. Dùng bộ so sánh



Hình 1.16 – Giải mã dùng bộ so sánh

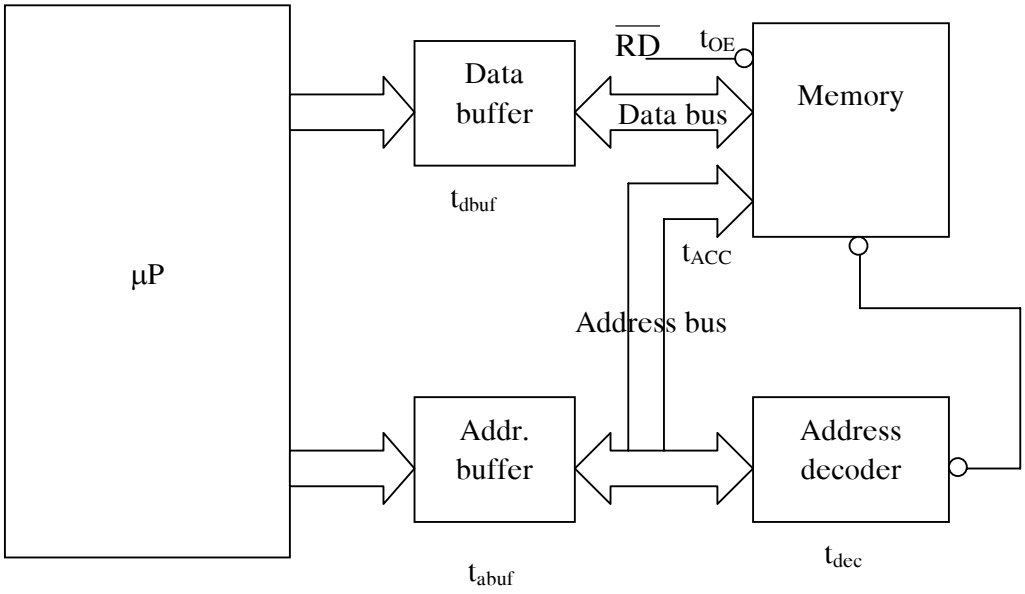
4.3.3. Định thì bộ nhớ

❖ **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.



Hình 1.17 – Các đường trì hoãn trong giao tiếp µP với bộ nhớ

t_{dbuf}: thời gian trì hoãn ở bộ đệm dữ liệu (data buffer)
t_{abuf}: thời gian trì hoãn 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 tACC = tcs
t_{dec}: thời gian trì hoãn ở bộ giải mã (decoder)

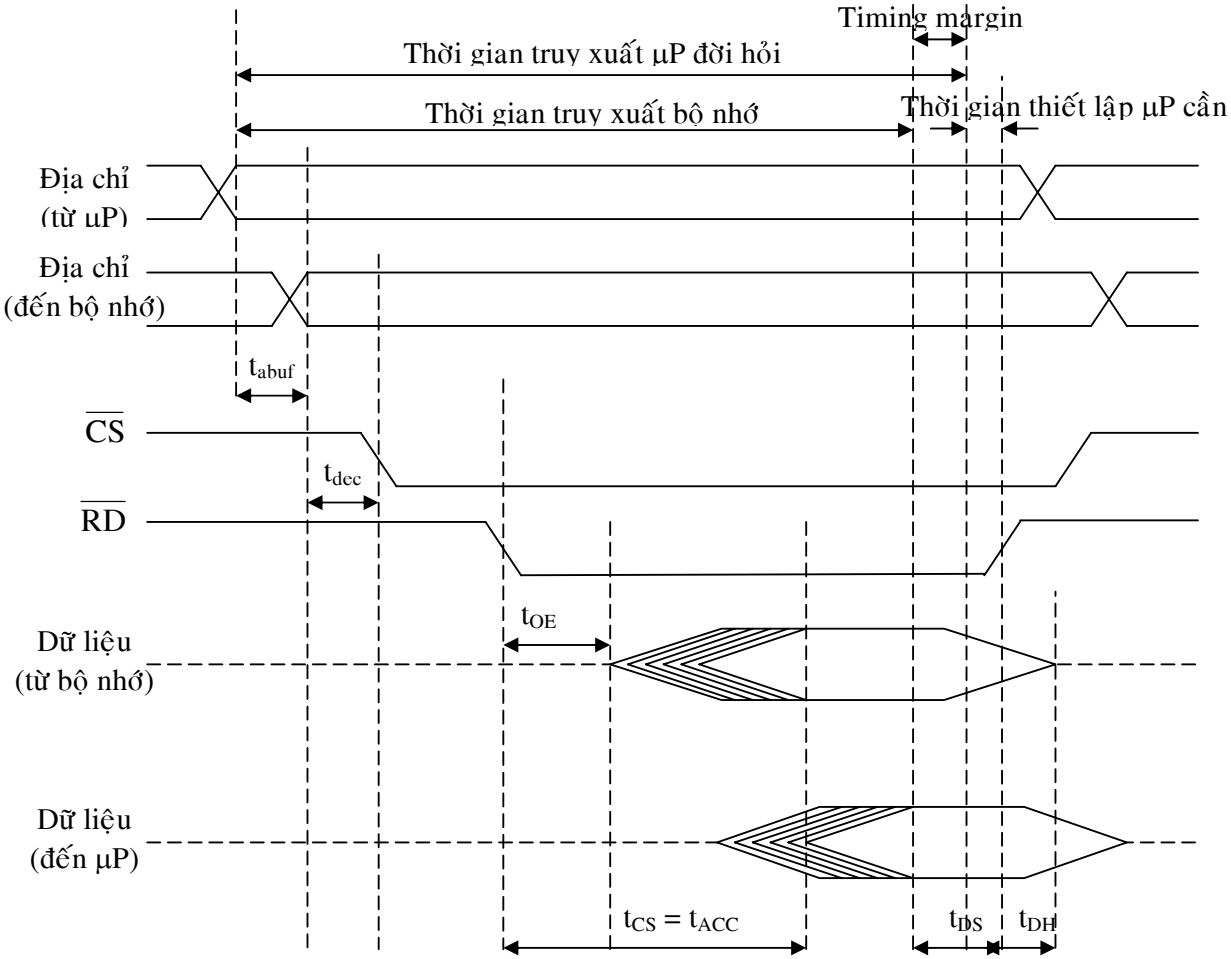
❖ **Định thì đọc bộ nhớ:**

Thời gian truy xuất tổng cộng của hệ thống bộ nhớ chính là tổng thời gian trì hoãn 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 thì (timing margin).

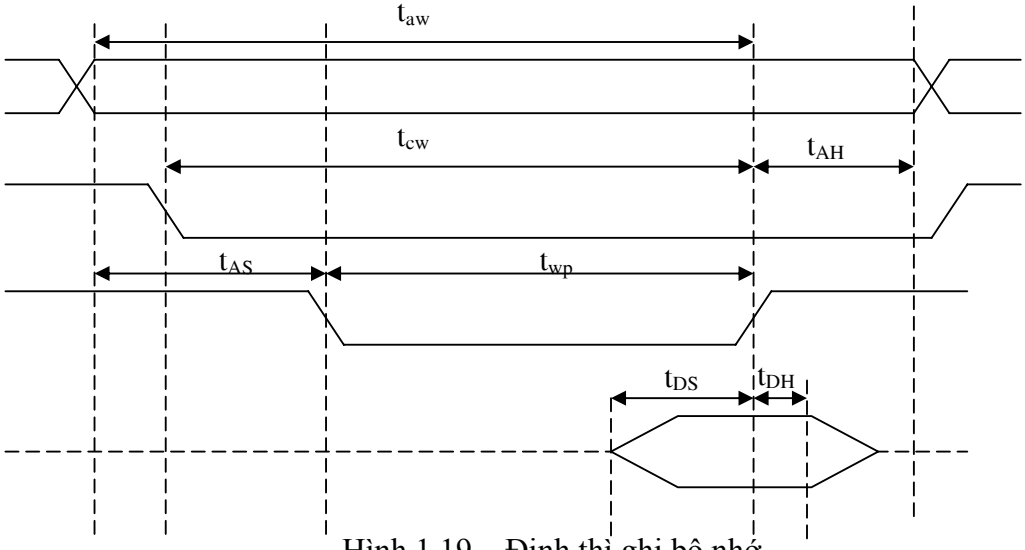
t_{DS} (Data Setup): thời gian thiết lập dữ liệu cung cấp bởi hệ thống bộ nhớ

t_{DH} (Data Hold): thời gian giữ dữ liệu cung cấp bởi hệ thống bộ nhớ



Hình 1.18 – Định thì đọc bộ nhớ

❖ **Định thì ghi bộ nhớ:**



Hình 1.19 – Định thì ghi 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 $\overline{WR} = 0$

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

5. µP 8086/8088

5.1. Giới thiệu

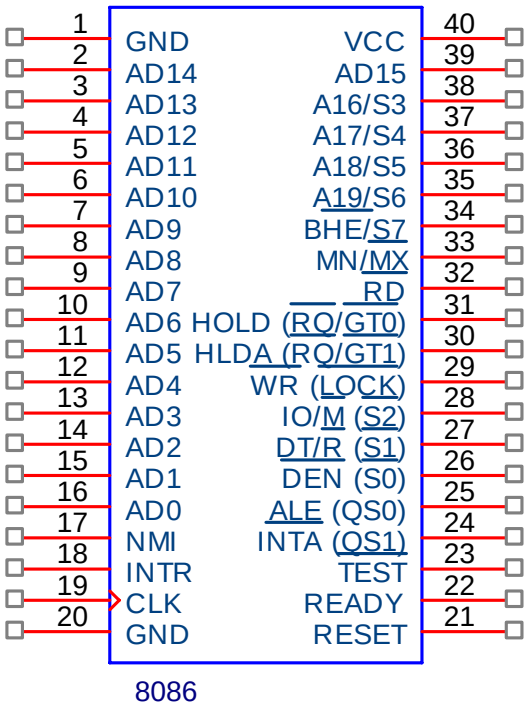
Tất cả các máy vi tính IBM họ PC hoặc các máy vi tính tương thích IBM đều sử dụng µP Intel họ iAPX. Bảng 2.1 liệt kê các đặc tính cơ bản của một số µP của Intel trong đó 80486 chứa một bộ điều khiển cache tích hợp và 8 KB RAM tĩnh, Pentium chứa cache 16 KB RAM tĩnh.

Bảng 1.4: Kiến trúc các µP của Intel 8 bit, 16 bit và 32 bit

	Tốc độ	Bus	Số transistor	Dung lượng bộ nhớ tối đa	Bộ nhớ ảo
4004	108 KHz	4 bits	2,300 (10 microns)	640 bytes	
8008	108 KHz	8 bits	3,500	16 KBytes	
8080	2 MHz	8 bits	6,000 (6 microns)	64 KBytes	
8086	5 MHz 8 MHz 10 MHz	16 bits	29,000 (3 microns)	1 Megabyte	
8088	5 MHz 8 MHz	8 bits	29,000 (3 microns)		
80286	8 MHz 10 MHz 12 MHz	16 bits	134,000 (1.5 microns)	16 Megabytes	1 gigabyte
Intel386(TM)DX Microprocessor	16 MHz 20 MHz 25 MHz 33 MHz	32 bits	275,000 (1 micron)	4 gigabytes	64 terabytes
Intel386(TM)SX Microprocessor	16 MHz 20 MHz	16 bits	275,000 (1 micron)	4 gigabytes	64 terabytes
Intel486(TM)DX Microprocessor	25 MHz 33 MHz 50 MHz	32 bits	1,200,000 (1 micron, .8 micron with 50 MHz)	4 gigabytes	64 terabytes
Intel486(TM)SX Microprocessor	16 MHz 20 MHz 25 MHz 33 MHz	32 bits	1,185,000 (.8 micron)	4 gigabytes	64 terabytes
Pentium® Processor	60MHz	32	3.1 million	4 gigabytes	64

	66MHz 75MHz 90MHz 100MHz 120MHz 133MHz 150MHz 166MHz	bits	(.8 micron)		terabytes
Pentium® Pro Processor	150MHz 180MHz 200MHz	32 bits	5.5 million (.32 micron)	4 gigabytes	64 terabytes

5.2. Mô tả chân



Hình 1.20 – Sơ đồ chân của 8086

8086 có bus địa chỉ 20 bit, bus dữ liệu 16 bit, 3 chân nguồn và 17 chân dùng cho các chức năng điều khiển. Tuy nhiên, ta có thể dùng kỹ thuật ghép kênh thời gian (time multiplexing) để cho phép một chân có nhiều chức năng nên các chân sẽ được phân ra:

- 16 chân dữ liệu và địa chỉ (AD0 ÷ AD15): các chân này sẽ là các đường địa chỉ trong trạng thái T1 và dữ liệu trong các trạng thái T2 – T4.
- 4 chân địa chỉ và trạng thái
- 3 chân nguồn
- 17 chân định thì và điều khiển

8086 có thể hoạt động ở chế độ tối thiểu (minimum mode) hay chế độ tối đa (maximum mode). Chế độ tối thiểu chỉ dùng cho các hệ thống μP đơn giản còn chế độ tối đa dùng cho các hệ thống phúc tạp hơn giao tiếp với các bộ nhớ và I/O riêng.

❖ **Các tín hiệu chung cho cả hai chế độ tối đa và tối thiểu:**

Bảng 1.5:

Chân	Chức năng	Loại
AD15 ÷ AD0	Bus dữ liệu / địa chỉ	2 chiều, 3 trạng thái
A19/S6 ÷ A16/S3	Địa chỉ / trạng thái	Ngõ ra 3 trạng thái
$\overline{MX}$	Điều khiển chế độ	Ngõ vào
$\overline{RD}$	Điều khiển đọc	Ngõ ra 3 trạng thái
$\overline{TEST}$	Chờ kiểm tra điều khiển	Ngõ vào
READY	Chờ trạng thái điều khiển	Ngõ vào
RESET	Reset hệ thống	Ngõ vào
NMI	Yêu cầu ngắt không thể che	Ngõ vào
INTR	Yêu cầu ngắt	Ngõ vào
CLK	Xung nhịp hệ thống	Ngõ vào
VCC	+5V	Ngõ vào
GND	GND	Ngõ vào

❖ **Các tín hiệu chỉ dùng trong chế độ tối thiểu:**

Bảng 1.6:

Chân	Chức năng	Loại
HOLD	Yêu cầu giữ	Ngõ vào
HLDA	Ghi nhận giữ	Ngõ vào
$\overline{WR}$	Điều khiển ghi	Ngõ ra 3 trạng thái
IO/$\overline{M}$	Điều khiển I/O và bộ nhớ	Ngõ ra 3 trạng thái
DT/$\overline{R}$	Truyền / nhận dữ liệu	Ngõ ra 3 trạng thái
$\overline{DEN}$	Cho phép dữ liệu	Ngõ ra 3 trạng thái
$\overline{BHE}$/S7	Đường trạng thái	Ngõ ra 3 trạng thái
ALE	Cho phép chốt địa chỉ	Ngõ ra
$\overline{INTA}$	Ghi nhận ngắt	Ngõ ra

❖ **Các tín hiệu chỉ dùng trong chế độ tối đa:**

Bảng 1.7:

Chân	Chức năng	Loại
$\overline{RQ}$ / $\overline{GT1,0}$	Yêu cầu / cấp bus	2 chiều
$\overline{LOCK}$	Điều khiển khóa ưu tiên bus	Ngõ ra 3 trạng thái
$\overline{S2} \div \overline{S0}$	Trạng thái chu kỳ bus	Ngõ ra 3 trạng thái
QS1, QS2	Trạng thái hàng lệnh	Ngõ ra

❖ **Trạng thái bus:**

Bảng 1.8:

Ngõ vào trạng thái			Chu kỳ CPU
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	
0	0	0	Ghi nhận ngắt
0	0	1	Đọc I/O port
0	1	0	Ghi I/O port
0	1	1	Ngừng
1	0	0	Nhận lệnh
1	0	1	Đọc bộ nhớ
1	1	0	Ghi bộ nhớ
1	1	1	Thụ động

❖ **Trạng thái hàng lệnh:**

Bảng 1.9:

QS1	QS0	Trạng thái hàng lệnh
0	0	Không hoạt động
0	1	Lấy byte đầu tiên của lệnh
1	0	Hàng rỗng
1	1	Lấy byte kế tiếp

❖ **Nguồn cung cấp và xung nhịp (VCC, GND và CLK):**

- 8086 sử dụng nguồn cấp điện +5V và có 2 chân đất.
- Dòng điện cực đại là 340 mA (10 mA cho loại CMOS).
- Xung nhịp dùng dạng xung chữ nhật có chu kỳ với thời gian cạnh lên và xuống nhỏ hơn 10 ns.
- Tiêu hao công suất và tần số xung nhịp cực đại:

❖ **Các chân trạng thái trong chế độ tối đa (S0, S1 và S2 - status):**

Các chân này sử dụng bởi bộ điều khiển bus 8288 để tạo các tín hiệu điều khiển như bảng 2.5.

❖ **Các chân điều khiển bus (HOLD, HLDA, $\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$, $\overline{LOCK}$):**

Chế độ tối thiểu:

- HOLD (giữ): ngõ vào tác động mức cao làm cho μP hở mạch tất cả các bus của nó, tách μP khỏi bộ nhớ của nó và I/O để cho phép thiết bị khác xử lý

bus hệ thống. Quá trình này gọi là truy xuất bộ nhớ trực tiếp (DMA – Direct Memory Access).

- HLDA (Hold acknowledge): ghi nhận yêu cầu DMA đối với bộ điều khiển DMA.

Chế độ tối đa:

- $\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$ (Request / Grant): các chân này dùng cả hai chức năng vào (nhận yêu cầu) và ra (chấp nhận yêu cầu). Khi một thiết bị muốn lấy điều khiển của bus cục bộ, nó sẽ phát yêu cầu bằng cách đưa tín hiệu mức thấp vào chân yêu cầu. Sau khi nhận yêu cầu, 8086 sẽ ở trạng thái HOLD và gởi tín hiệu chấp nhận ra chân này. Ở đây, chân $\overline{RQ}/\overline{GT0}$ có độ ưu tiên cao hơn chân $\overline{RQ}/\overline{GT1}$.
- $\overline{LOCK}$: báo cho các thiết bị khác biết không thể lấy điều khiển của bus cục bộ.

❖ **Các chân ngắt (NMI, INTR và $\overline{INTA}$):**

INTR và NMI là các yêu cầu ngắt khởi động bằng phần cứng, làm việc chính xác như các ngắt mềm. NMI (Non-Maskable Interrupt) là ngõ vào tác động cạnh lên. NMI là ngắt không thể che được và luôn được phục vụ, thường dùng cho các sự kiện như hư nguồn hay các lỗi bộ nhớ. INTR tác động mức cao và có thể bị che bằng cách xoá cờ IF trong thanh ghi cờ (xem 2.3.4) bằng lệnh CLI.

Khi NMI tích cực, điều khiển sẽ được chuyển đến địa chỉ chứa trong các vị trí 00008h ÷ 0000Bh. Khi INTR tích cực, chu kỳ ghi nhận ngắt (interrupt acknowledge cycle) được thực hiện. Quá trình này giống như chu kỳ đọc bộ nhớ ngoại trừ $\overline{INTA}$ tích cực thay vì $\overline{RD}$. Thiết bị tạo ngắt sẽ đặt một giá trị 8 bit vào data bus và chuyển điều khiển đến vị trí giá trị × 4 đến giá trị × 4 + 3.

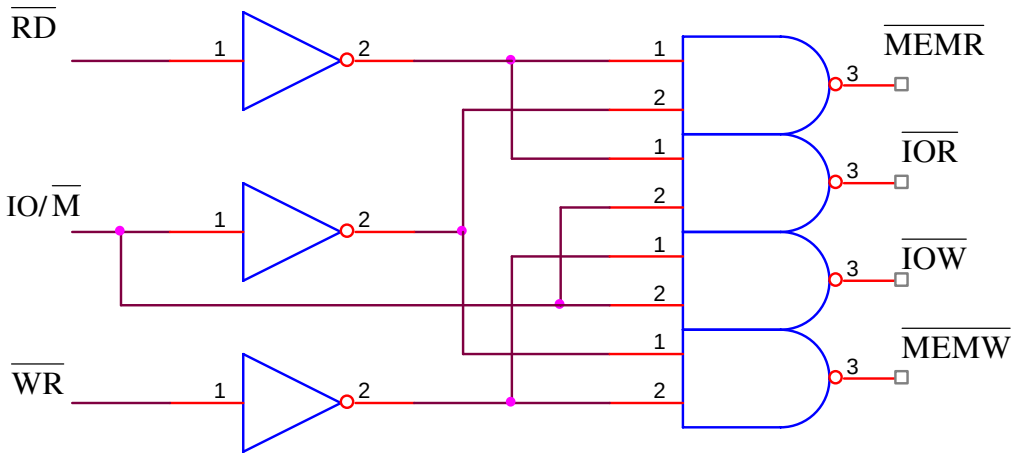
❖ **Chân RESET**: hoạt động khi có xung tác động mức cao, dùng để khởi động lại (P. Sau khi khởi động, (P sẽ đọc lệnh tại địa chỉ FFFF0h. RESET được sử dụng khi hệ thống có sự cố.

❖ **Các chân điều khiển bus (READY, $\overline{RD}$, ALE, $\overline{DEN}$, DT/$\overline{R}$, $\overline{WR}$ và IO/$\overline{M}$):**

Trong các chân điều khiển này, chỉ có hai chân READY và $\overline{RD}$ làm việc ở chế độ tối đa.

- Chân READY: ngõ vào READY được lấy mẫu ở cạnh lên của xung nhịp T2. Nếu chân này ở mức thấp (không sẵn sàng) thì sẽ thêm vào một chu kỳ T3 nữa. Chu trình này sẽ tiếp tục cho đến khi nào chân READY lên mức cao. Ngõ vào này thường được điều khiển bởi thiết bị bộ nhớ chậm, không thể cung cấp dữ liệu kịp thời cho μP.

- Chân IO/$\overline{M}$ (IO/Memory – Xuất nhập /Bộ nhớ): xác định chu kỳ bus hiện hành làm việc với bộ nhớ (mức thấp) hay I/O (mức cao).
- Chân $\overline{RD}$ (Read): tín hiệu tác động mức thấp chỉ chiều truyền dữ liệu từ bộ nhớ hay I/O đến µP. Ta có thể kết hợp với tín hiệu này với IO/$\overline{M}$ để tạo các tín hiệu $\overline{MEMR}$ và $\overline{IOR}$. Nó được xuất ra trong trạng thái T2 và lấy đi trong trạng thái T4. Thiết bị bộ nhớ hay I/O giả sử là đã đặt byte hay word vào các đường dữ liệu khi $\overline{RD}$ trở về mức cao.
- Chân $\overline{WR}$ (Write): tín hiệu này ngược với $\overline{RD}$, nó xác định chiều truyền dữ liệu từ µP đến I/O hay bộ nhớ.

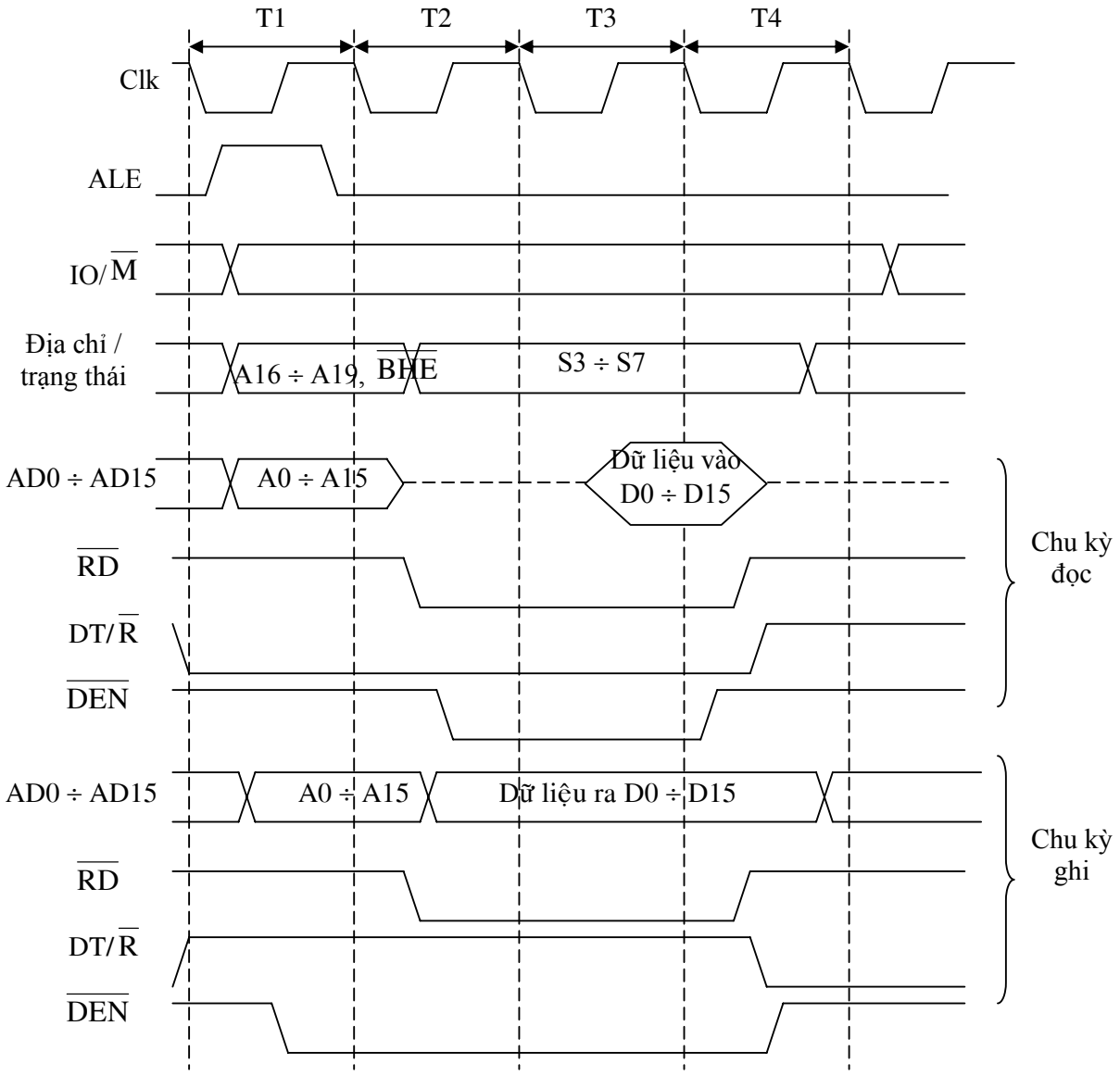


Hình 1.21 – Tạo tín hiệu điều khiển bộ nhớ và I/O

- Chân ALE (Address Latch Enable - cho phép chốt địa chỉ): tín hiệu ra trên chân này có thể dùng để phân kênh các đường địa chỉ, dữ liệu và trạng thái trên AD0 ÷ AD15, A16/S3 ÷ A19/S6 và $\overline{BHE}$/S7. Mọi chu kỳ bắt đầu với xung ALE trong trạng thái T1. Địa chỉ 20 bit được bảo đảm sẽ hợp lệ khi ALE chuyển từ mức cao xuống mức thấp.

- Chân $\overline{DEN}$ (Data Enable – cho phép dữ liệu): tín hiệu này được dùng với DT/$\overline{R}$ để cho phép nối các bộ đệm hai chiều vào data bus. Nó ngăn ngừa sự tranh chấp bus bằng cách cấm các bộ đệm dữ liệu cho đến trạng thái T2 khi các đường dữ liệu / địa chỉ không còn lưu trữ địa chỉ của bộ nhớ hay I/O.

- Chân DT/$\overline{R}$ (Data transmit/receive – truyền/nhận dữ liệu): dùng để điều khiển chiều của luồng dữ liệu qua các bộ đệm (nếu có) vào bus dữ liệu của hệ thống. Khi ở mức thấp, nó chỉ thực hiện tác vụ đọc và khi ở mức cao nó chỉ thực hiện tác vụ ghi.



Hình 1.22 – Các chu kỳ đọc và ghi của 8086

❖ **Các chân trạng thái (AD16/S3 ÷ AD19/S6 và $\overline{\text{BHE}}$/S7):**

5 tín hiệu trạng thái này được xuất ra trong các trạng thái T2 ÷ T4, dùng cho các mục đích kiểm tra. Bit S7 là bit trạng thái dư (không dùng), bit S6 luôn bằng 0, S5 mô tả trạng thái của cờ ngắt IF còn S3, S4 dùng để xác định đoạn đang sử dụng:

Bảng 1.10:

S4	S3	Đoạn
0	0	Thêm
0	1	Stack
1	0	Mã (hay không)
1	1	Dữ liệu

Tín hiệu $\overline{BHE}$/S7 (Bus High Enable) chỉ được xuất trong trạng thái T1. Khi chân này ở mức thấp, nó sẽ chỉ AD8 ÷ AD15 liên quan đến việc truyền dữ liệu. Quá trình này có thể xảy ra đối với các truy xuất bộ nhớ, I/O hay truy xuất 1 byte dữ liệu từ địa chỉ lẻ.

❖ **Bus dữ liệu (AD0 ÷ AD15):**

16 chân này tạo thành bus dữ liệu hai chiều. Các đường này chỉ hợp lệ trong các trạng thái T2 ÷ T4. Trong trạng thái T1, chúng giữ 16 bit thấp của địa chỉ bộ nhớ hoặc I/O.

❖ **Bus địa chỉ (AD0 ÷ AD15 và AD16/S3 ÷ AD19/S6):**

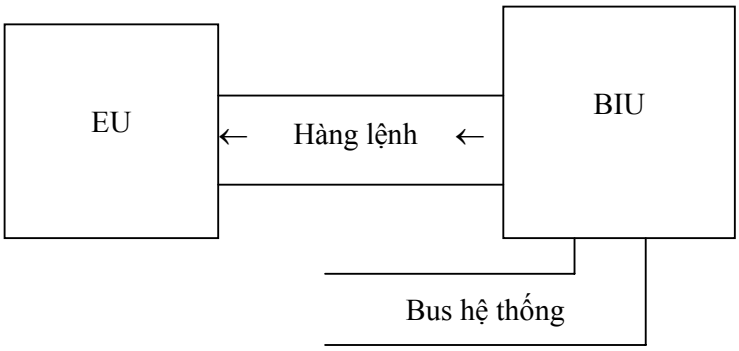
20 chân này tương ứng với bus địa chỉ 20 bit và cho phép μP truy xuất 1 MB vị trí bộ nhớ. Các đường ra này chỉ hợp lệ trong trạng thái T1, chuyển thành các đường dữ liệu và trạng thái trong trạng thái T2 ÷ T4.

❖ **Chọn chế độ $\overline{MX}$:**

Chân này dùng để chọn chế độ hoạt động cho 8086, nếu ở mức cao thì sẽ hoạt động ở chế độ tối thiểu còn ở mức thấp thì sẽ hoạt động ở chế độ tối đa.

5.3. Kiến trúc nội

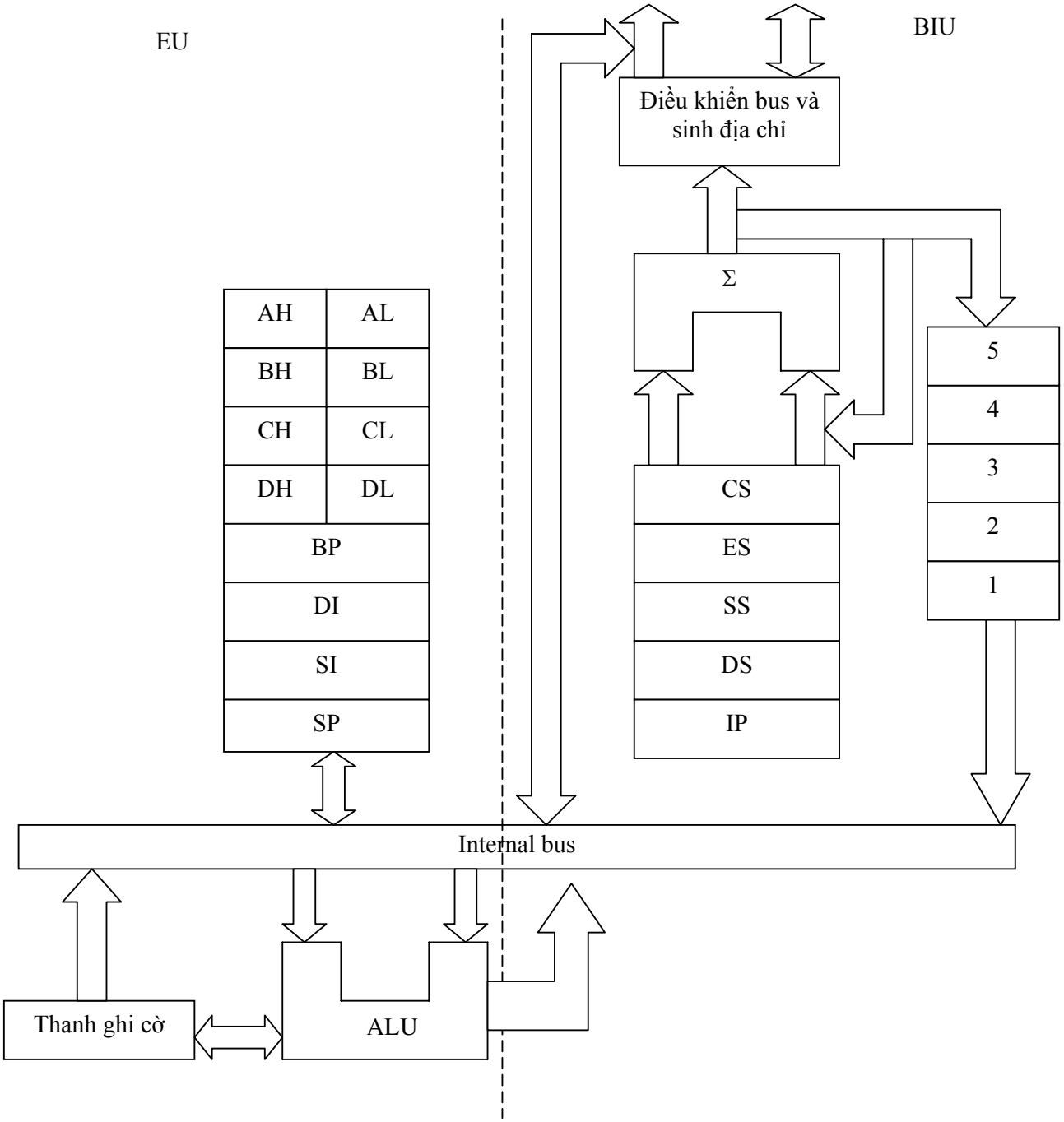
μP có khả năng thực hiện các tác vụ dữ liệu theo tập lệnh bên trong. Một lệnh được ghi nhận bằng mã đã được định nghĩa trước, gọi là mã lệnh (opcode). Trước khi thực thi một lệnh, μP phải nhận được mã lệnh từ bộ nhớ chương trình của nó. Quá trình xử lý này gọi là chu kỳ nhận lệnh (fetch cycle). Một khi các mã được nhận và được giải mã thì mạch bên trong μP có thể tiến hành thực thi (execute) mã lệnh.



Hình 1.23 – Kiến trúc nội của μP 8086

BIU (Bus Interface Unit – đơn vị giao tiếp bus) nhận các mã lệnh từ bộ nhớ và đặt chúng vào hàng chờ lệnh. EU (Execute Unit – đơn vị thực thi) sẽ giải mã và thực hiện các lệnh trong hàng. Chú ý rằng các đơn vị EU và BIU làm việc độc lập với nhau nên BIU có khả năng đang nhận một lệnh mới trong khi EU đang thực thi lệnh trước đó. Khi EU đã thực hiện xong lệnh, nó sẽ lấy mã lệnh kế tiếp trong hàng đợi lệnh (instruction queue).

Kiến trúc nội của μP 8086 ở hình 1.24. Nó có 2 bộ xử lý riêng: BIU và EU. BIU cung cấp các chức năng phần cứng, bao gồm tạo các địa chỉ bộ nhớ và I/O để chuyển dữ liệu giữa EU và bên ngoài μP.



Hình 1.24 – Kiến trúc nội của 8086

EU nhận các mã lệnh chương trình và dữ liệu từ BIU, thực thi các lệnh này và chứa các kết quả trong các thanh ghi. Ngoài ra, dữ liệu cũng có thể chứa trong một vị trí bộ nhớ hay được ghi vào thiết bị xuất. Chú ý rằng EU không có bus hệ thống nên phải thực hiện nhận và xuất tất cả các dữ liệu của nó thông qua BIU.

Sự khác biệt giữa µP 8086 và 8088 là BIU. Trong 8088, đường bus dữ liệu là 8 bit trong khi của 8086 là 16 bit. Ngoài ra hàng lệnh của 8088 dài 4 byte trong khi của 8086 là 6 byte.

Tuy nhiên do EU giữa hai loại µP này giống nhau nên ***các chương trình viết cho 8086 có thể chạy được trên 8088 mà không cần thay đổi gì cả***.

5.4. Các thanh ghi

µP 8086/8088 có tất cả 14 thanh ghi nội. Các thanh ghi này có thể phân loại 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 trạng thái và điều khiển (status & control register)

5.4.1. 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.

Bảng 1.11:

Thanh ghi	Sử dụng trong
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) 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)
DX	MUL, IMUL (toán hạng nguồn kích thước word) DIV, IDIV (toán hạng nguồn kích thước word)

AX (ACC – Accumulator): thanh ghi tích luỹ
BX (Base): thanh ghi cơ sở
CX (Count): đếm
DX (Data): thanh ghi dữ liệu

5.4.2. 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 doạn 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.

5.4.3. 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.

5.4.4. Các thanh ghi điều khiển và trạng thái

Thanh ghi con trỏ lệnh IP (Instruction Pointer) giống như bộ đếm chương trình (Program Counter). Thanh ghi điều khiển này do BIU quản lý nhằm lưu trữ offset từ bắt đầu đoạn mã đến lệnh thực thi kế tiếp. Ta không thể xử lý trực tiếp trên thanh ghi IP.

Thanh ghi cờ (Flag register) hay từ trạng thái 16 bit chứa 3 bit điều khiển (TF, IF và DF) và 6 bit trạng thái (OF, SF, ZF, AF, PF và CF) còn các bit còn lại mà 8086/8088 không sử dụng thì không thể truy xuất được.

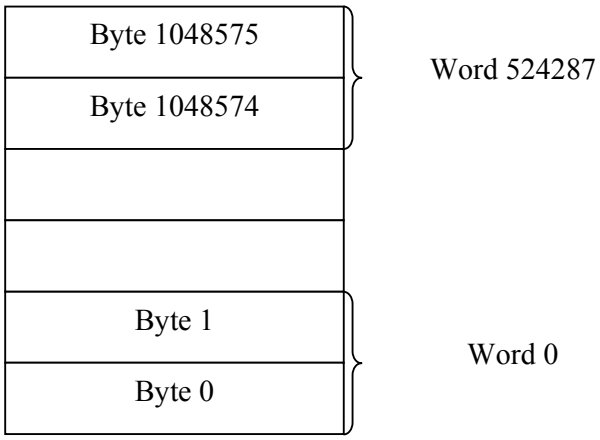
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

- OF (Overflow - tràn): OF = 1 xác định tràn số học, xảy ra khi kết quả vượt ra ngoài phạm vi biểu diễn
- DF (Direction- hướng): xác định hướng chuyển string, DF = 1 khi μP làm việc với string theo thứ tự từ phải sang trái.
- IF (Interrupt - ngắt): cho phép hay cấm các interrupt có mặt nạ
- TF (Trap - bẫy): đặt μP vào chế độ từng bước, dùng cho các chương trình gỡ rối (debugger).
- SF (Sign - dấu): dùng để chỉ các kết quả số học là số dương (SF = 0) hay âm (SF = 1).
- ZF (Zero): = 1 nếu kết quả của phép toán trước là 0.
- AF (Auxiliary – nhớ phụ): dùng trong các số thập phân để chỉ nhớ từ nửa byte thấp hay mượn từ nửa byte cao.
- PF (Parity): PF = 1 nếu kết quả của phép toán là có tổng số bit 1 là chẵn (dùng để kiểm tra lỗi truyền dữ liệu)
- CF (Carry): CF = 1 nếu có nhớ hay mượn từ bit cao nhất của kết quả. Cờ này cũng dùng cho các lệnh quay.

6. Phân đoạn bộ nhớ

Ta biết rằng dù 8086 là μP 16 bit (có bus dữ liệu 16 bit) nhưng vẫn dùng bộ nhớ theo các byte. Điều này cho phép μP làm việc với byte cũng như word, nó rất quan trọng trong giao tiếp với các thiết bị I/O như máy in, thiết bị đầu cuối và modem (chúng được thiết kế để chuyển dữ liệu mã hoá ASCII 7 hay 8 bit). Ngoài ra, nhiều mã lệnh của 8086/8088 có chiều dài 1 byte nên cần phải truy xuất được các byte riêng biệt để có thể xử lý các lệnh này.

8086/8088 có bus địa chỉ 20 bit nên có thể cho phép truy xuất 2^{20} = 1048576 địa chỉ bộ nhớ khác nhau.

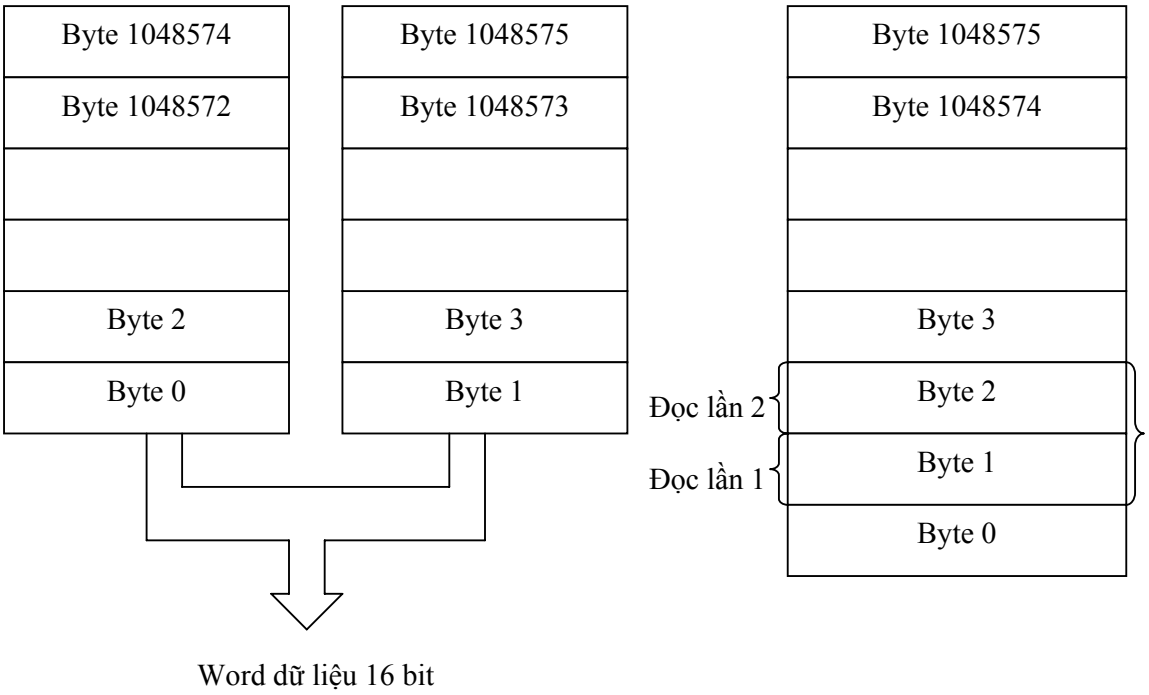


Hình 1.25 – Vùng nhớ của 8086/8088 có 1048576 byte hay 524288 word

Để thực hiện đọc 16 bit từ bộ nhớ, 8086 sẽ thực hiện đọc đồng thời byte có địa chỉ lẻ và byte có địa chỉ chẵn. Do đó, 8086 tổ chức bộ nhớ thành các bank chẵn và lẻ. Theo hình 1.25, ta có thể thấy rằng các word luôn bắt đầu tại địa chỉ chẵn nhưng ta vẫn

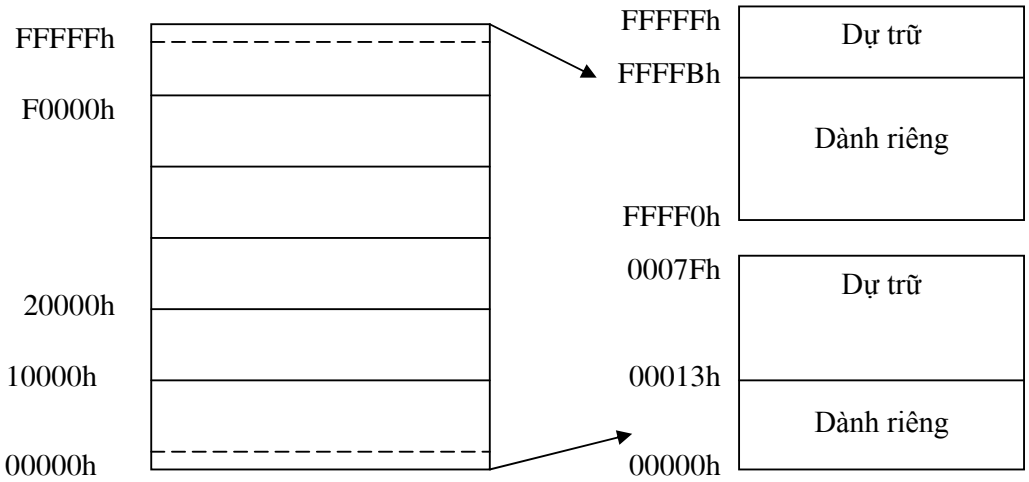
có thể đọc word có địa chỉ lẻ bằng cách thực hiện 2 chu kỳ đọc bộ nhớ: một chu kỳ đọc byte thấp và một chu kỳ đọc byte cao. Điều này sẽ làm chậm tốc độ xử lý.

Đối với 8088 thì do bus dữ liệu 8 bit nên dù word có địa chỉ chẵn hay lẻ, nó cũng cần phải thực hiện 2 chu kỳ đọc hay ghi bộ nhớ và giao tiếp với bộ nhớ như một bank.



Hình 1.26 – Đọc word địa chỉ chẵn và địa chỉ lẻ

Ngoài ra bộ nhớ cũng chia thành 16 khối, mỗi khối có kích thước 64 KB, bắt đầu ở địa chỉ 00000h và kết thúc ở FFFFFh. Địa chỉ bắt đầu mỗi khối sẽ tăng lên 1 ở số hex có ý nghĩa nhiều nhất khi thay đổi từ khối này sang khối kia. Ví dụ như khối 00000h → 10000h → 20000h …



Hình 1.27 – Bảng bộ nhớ cho 8086/8088

❖ Các thanh ghi phân đoạn:

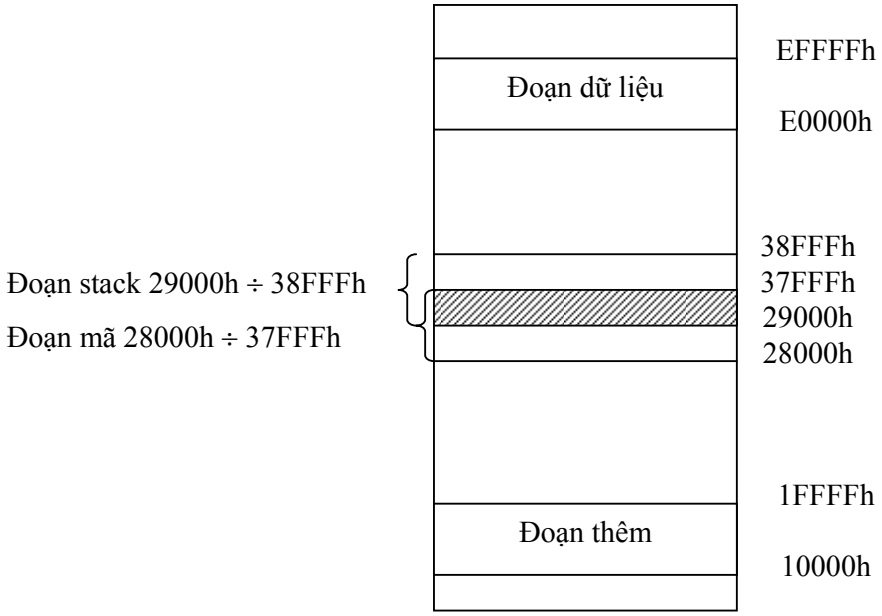
8086/8088 định nghĩa 4 khối bộ nhớ 64KB: đoạn mã (code segment) giữ các mã lệnh chương trình, đoạn ngăn xếp (stack segment) lưu các địa chỉ sẽ trả về từ các chương trình con (subroutine) hay trình phục vụ ngắt (interrupt subroutine), đoạn dữ liệu (data segment) lưu trữ dữ liệu cho chương trình và đoạn thêm (extra segment) thường dùng cho các dữ liệu dùng chung.

Các thanh ghi đoạn (CS, DS, SS và ES) dùng để chỉ vị trí nền của mỗi đoạn. Các thanh ghi này có 16 bit trong khi địa chỉ bộ nhớ là 20 bit nên để xác dịnh vị trí bộ nhớ, ta sẽ thêm 4 bit 0 vào các bit thấp của thanh ghi đoạn. Giả sử như thanh ghi CS chứa giá trị 1111h thì nó sẽ chỉ tới địa chỉ nền là 11110h. Chú ý rằng địa chỉ bắt đầu một đoạn không thể tuỳ ý mà phải bắt đầu tại một địa chỉ chia hết cho 16. Nghĩa là 4 bit thấp phải là 0. Ta cũng chú ý rằng 4 đoạn có thể không tách rời nhau mà chồng lấp lên nhau và ta cũng có thể cho 4 giá trị của các thanh ghi đoạn bằng nhau nghĩa là 4 đoạn này trùng nhau.

VD: Thanh ghi DS có giá trị là 1000h thì địa chỉ nền là 10000h. Địa chỉ kết thúc tìm được bằng cách cộng địa chỉ nền với giá trị FFFFh (64K) → địa chỉ kết thúc là 10000h + FFFFh = 1FFFFh. Như vậy đoạn dữ liệu có địa chỉ từ 10000h = 1FFFFh.

Các vị trí bộ nhớ không được định nghĩa trong các đoạn hiện hành không thể truy xuất được. Muốn truy xuất đến các vị trí đó, ta phải định nghĩa lại một trong các thanh ghi đoạn sau cho đoạn phải chứa vị trí đó. Như vậy, tại một thời điểm bất kỳ ta chỉ có thể truy xuất tối đa 4 × 64 KB = 256 KB bộ nhớ. Nội dung của các thanh ghi đoạn chỉ có thể xác định thông qua phần mềm.

VD: Giả sử các thanh ghi đoạn có các giá trị CS = 2800h, DS = E000h, SS = 2900h và ES = 1000h. Ta có vị trí các đoạn trong bảng bộ nhớ như sau:



Hình 1.28 – Vị trí các phân đoạn theo giá trị các thanh ghi đoạn

❖ **Địa chỉ logic và địa chỉ vật lý:**

Các địa chỉ trong một đoạn thay đổi từ 0000h ÷ FFFFh, tương ứng với chiều dài đoạn là 64 KB. Một địa chỉ trong một đoạn được gọi là ***địa chỉ logic hay offset***. Ví dụ như địa chỉ logic 0010h của đoạn mã trong hình 1.28 sẽ có địa chỉ thật sự là 28000h + 0010h = 28010h. Địa chỉ này gọi là ***địa chỉ vật lý***.

Như vậy, địa chỉ vật lý chính là địa chỉ thật sự xuất hiện ở bus địa chỉ, nó có chiều dài 20 bit còn địa chỉ logic là độ lệch (offset) từ vị trí 0 của một đoạn cho trước.

VD: Giả sử xét các đoạn như hình 1.28. Địa chỉ vật lý tương ứng với địa chỉ logic 1000h trong đoạn stack là:

29000h + 1000h = 2A000h

Địa chỉ vật lý tương ứng với địa chỉ logic 2000h trong đoạn mã là:

28000h + 2000h = 2A000h

Ta thấy rằng có thể địa chỉ vật lý trùng nhau khi địa chỉ logic khác nhau nghĩa là một địa chỉ vật lý có thể có nhiều địa chỉ logic khác nhau.

Để chỉ địa chỉ logic 1000h trong đoạn mã, ta dùng ký hiệu CS:1000h. Tương tự như vậy cho các đoạn khác, nghĩa là địa chỉ logic 1111h trong đoạn dữ liệu sẽ là DS:1111h.

Mọi lệnh tham chiếu bộ nhớ sẽ có một thanh ghi đoạn mặc nhiên. Thanh ghi IP cung cấp địa chỉ offset khi truy xuất đến đoạn mã và BP cho đoạn stack. Ví dụ như IP = 1000h và CS = 2000h thì BIU sẽ truy xuất đến địa chỉ 20000h + 1000h = 21000h và nhận byte tại vị trí này.

Bảng 1.12:

Tham chiếu bộ nhớ	Đoạn mặc nhiên	Đoạn khác	Offset
Nhận lệnh	CS	Không	IP
Tác vụ stack	SS	Không	SP
Dữ liệu tổng quát	DS	CS,ES,SS	Địa chỉ hiệu dụng
Nguồn của string	DS	CS,ES,SS	SI
Đích của string	ES	Không	DI
BX dùng làm con trỏ	DS	CS,ES,SS	Địa chỉ hiệu dụng
BP dùng làm con trỏ	SS	CS,ES,SS	Địa chỉ hiệu dụng

VD: Ta sử dụng lệnh MOV [BP],AL với BP = 2C00h. Ở đây BP dùng làm con trỏ nên dùng đoạn stack. Giả sử các phân đoạn như hình 2.11 thì địa chỉ vật lý sẽ là 29000h + 2C00h = 2BC00h

❖ **Định nghĩa các vị trí bộ nhớ:**

Thông thường ít khi nào ta cần biết đến địa chỉ vật lý của một vị trí bộ nhớ mà ta chỉ quan tâm đến địa chỉ logic của nó mà thôi. Lý do là vì địa chỉ vật lý còn phải phụ thuộc vào nội dung của các thanh ghi đoạn ngay cả khi địa chỉ logic giữ không đổi như đã xét ở trên.

7. Các cách định địa chỉ

Bảng 1.13:

Cách định địa chỉ	Mã đối tượng	Ví dụ			
		Từ gợi nhớ	Đoạn truy xuất	Hoạt động	Mô tả
Tức thời	B80010	MOV AX,1000h	Mã	AH ← 10h AL ← 00h	(1)
Thanh ghi	8BD1	MOV DX,CX	Trong µP	DX ← CX	(2)
Trực tiếp	8A260010	MOV AH,[1000h]	Döõ lieäu	AH ← [1000h]	(3)
Gián tiếp thanh ghi	8B04 FF25 FE4600 FF0F	MOV AX,[SI] JMP [DI] INC BYTE PTR [BP] DEC WORD PTR [BX]	Dữ liệu Dữ liệu Stack Dữ liệu	AL ← [SI]; AH ←[SI+1] IP←[DI+1:DI] [BP]←[BP]+1 [BX+1:BX]← [BX+1:BX]-1	(4)
Có chỉ số	8B4406 FF6506	MOV AX,[SI+6] JMP [DI+6]	Dữ liệu Dữ liệu	AL ← [SI+6]; AH ←[SI+7] IP←[DI+7:DI+6]	(5)
Có nền	8B4602 FF6702	MOV AX,[BP+2] JMP [BP+2]	Stack Dữ liệu	AL←[BP+2]; AH ←[BP+3] IP←[BX+3:BX+6]	(6)
Có nền và có chỉ số	8B00 FF21 FE02 FF0B	MOV AX,[BX+SI] JMP [BX+DI] INC BYTE PTR [BP+SI] DEC WORD PTR [BP+DI]	Dữ liệu Dữ liệu Stack Stack	AL←[BX+SI];AH←[BX+SI+1] IP←[BX+DI+1:BX+DI] [BP+SI]←[BP+SI]+1 [BP+DI+1:BP+DI]← [BP+DI+1:BP+DI]-1	(7)
Có nền và có chỉ số với độ dời	8B4005 FF6105 FE4205 FF4B05	MOV AX,[BX+SI+5] JMP [BX+DI+5] INC BYTE PTR [BP+SI+5] DEC WORD PTR [BP+DI+5]	Dữ liệu Dữ liệu Stack Stack	AL←[BX+SI+5] AH←[BX+SI+1] IP←[BX+DI+6:BX+DI+5] [BP+SI+5]←[BP+SI+5]+1 [BP+DI+6:BP+DI+5]← [BP+DI+6:BP+DI+5]-1	(8)
String	A4	MOVSB	Thêm, dữ liệu	[ES:DI] ← [DS:DI] Nếu DF = 0 thì SI ← SI + 1; DI ← DI + 1 Nếu DF = 1 thì SI ← SI - 1; DI ← DI - 1	(9)

- BYTE PTR và WORD PTR tránh lầm giữa truy xuất byte và word.
- Độ dời được cộng vào thanh ghi con trỏ hay nền là số nhị phân dạng bù 2.
- (1): nguồn dữ liệu trong lệnh
- (2): đích và nguồn là các thanh ghi của µP
- (3): địa chỉ bộ nhớ cung cấp trong lệnh
- (4): địa chỉ bộ nhớ cung cấp trong thanh ghi con trỏ hay chỉ số
- (5): địa chỉ bộ nhớ là tổng của thanh ghi chỉ số cộng với độ dời trong lệnh
- (6): địa chỉ bộ nhớ là tổng của thanh ghi BX hay BP cộng với độ dời trong lệnh
- (7): địa chỉ bộ nhớ là tổng của thanh ghi chỉ số và thanh ghi nền

- (8): địa chỉ bộ nhớ là tổng của thanh ghi chỉ số, thanh ghi nền và độ dời trong lệnh
- (9): địa chỉ nguồn bộ nhớ là thanh ghi SI trong đoạn dữ liệu và địa chỉ đích bộ nhớ là thanh ghi DI trong đoạn thêm

7.1. Định địa chỉ tức thời

Các lệnh dùng cách định địa chỉ tức thời lấy dữ liệu trong lệnh làm một phần của lệnh. Trong cách này, dữ liệu sẽ được chứa trong đoạn mã thay vì trong đoạn dữ liệu. Dữ liệu cho lệnh MOV AX,1000h được cung cấp tức thời sau mã lệnh B8. Chú ý rằng trong mã đối tượng byte dữ liệu cao đi sau byte dữ liệu thấp.

Cách định địa chỉ tức thời thường dùng để nạp một thanh ghi hay vị trí bộ nhớ với các dữ liệu ban đầu. Sau đó, các lệnh kế tiếp sẽ làm việc với các dữ liệu này. Tuy nhiên, cách định địa chỉ này không sử dụng được cho các thanh ghi đoạn.

7.2. Định địa chỉ thanh ghi

Một số lệnh chỉ làm công việc chuyển dữ liệu giữa các thanh ghi của μP. Ví dụ như MOV DX,CX sẽ chuyển dữ liệu từ thanh ghi CX vào thanh ghi DX. Ở đây ta không cần thực hiện tham chiếu bộ nhớ.

Ta có thể kết hợp cách định địa chỉ tức thời và định địa chỉ thanh ghi để nạp dữ liệu cho các thanh ghi đoạn.

VD:

```
MOV  AX, 1000h
MOV CS,AX
```

Sau khi thực hiện 2 lệnh này, giá trị của thanh ghi CS sẽ là 1000h.

7.3. Định địa chỉ trực tiếp

Ngoài 2 cách định địa chỉ trên, tất cả các cách định địa chỉ còn lại cho trong bảng 2.6 đều cần phải truy xuất đến bộ nhớ với ít nhất một toán hạng. Trong cách định địa chỉ trực tiếp, địa chỉ bộ nhớ được cung cấp trực tiếp như là một phần của lệnh. Ví dụ như lệnh MOV AH,[1000h] sẽ đưa nội dung chứa trong ô nhớ DS:1000h vào thanh ghi AH hay lệnh MOV [2000h],AX sẽ đưa nội dung chứa trong AX vào 2 ô nhớ liên tiếp DS:2000h và DS:2001h

7.4. Định địa chỉ truy xuất bộ nhớ gián tiếp

Các cách định địa chỉ trực tiếp sẽ thuận lợi cho các truy xuất bộ nhớ không thường xuyên. Tuy nhiên, nếu một ô nhớ cần phải truy xuất nhiều lần trong một chương trình thì quá trình nhận địa chỉ (2 byte) sẽ phải thực hiện nhiều lần. Điều này sẽ không hiệu quả.

Để giải quyết vấn đề này, ta thực hiện lưu trữ địa chỉ của ô nhớ cần truy xuất trong một thanh ghi con trỏ, chỉ số hay thanh ghi cơ sở (BX, BP, SI hay DI). Ngoài ra,

ta có thể sử dụng độ dời bù 2 bằng cách cộng vào các thanh ghi để dời đi so với vị trí được các thanh ghi chỉ đến.

Bảng 2.13:

Cách định địa chỉ	Địa chỉ hiệu dụng (EA – Effective Address)		
	Độ dời	**Thanh ghi nền**	**Thanh ghi chỉ số**
Gián tiếp thanh ghi	Không	BX hay BP	Không
	Không	Không	SI hay DI
Có chỉ số	-128 ÷ 127	Không	SI hay DI
Có nền	-128 ÷ 127	BX hay BP	Không
Có nền và chỉ số	Không	BX hay BP	SI hay DI
Có nền và chỉ số với độ dời	-128 ÷ 127	BX hay BP	SI hay DI

Như vậy, một độ dời có thể được cộng vào thanh ghi nền và kết quả này được cộng tiếp vào thanh ghi chỉ số. Địa chỉ thu được gọi là địa chỉ hiệu dụng EA.

Ngoài ra ta cũng có thể viết cách định địa chỉ gián tiếp như sau:

```
MOV AX,table[SI]
```

Trong đó ***table*** là nhãn gán cho một vị trí ô nhớ nào đó. Lệnh này sẽ truy xuất phần tử thứ SI trong dãy ***table*** (giả sử SI = 2 thì sẽ truy xuất phần tử thứ 2). Ta cũng có thể viết lệnh trên như sau:

```
MOV AX,[table + SI]
```

Chú ý rằng các đoạn mặc định cho các cách định địa chỉ gián tiếp là đoạn stack khi dùng BP, là đoạn dữ liệu khi dùng BX, SI hay DI.

VD: Lệnh:

```
MOV  AH,10h          thực hiện định địa chỉ tức thời
MOV  AX,[BP + 10]    thực hiện định địa chỉ có nền
MOV  AH,[BP + SI]    thực hiện định địa chỉ có nền và có chỉ số
```

7.5. Định địa chỉ chuỗi

Chuỗi là một dãy liên tục các byte hay word lưu trữ trong bộ nhớ dưới dạng các ký tự ASCII. 8086/8088 có các lệnh dùng để xử lý chuỗi, các lệnh này sử dụng cặp thanh ghi DS:SI để chỉ nguồn chuỗi ký tự và ES:DI để chỉ đích chuỗi. Lệnh MOVSB sẽ chuyển byte dữ liệu nguồn đến vị trí đích trong đó SI và DI sẽ tăng hay giảm tuỳ theo giá trị của DF (xem 2.3.4 và bảng 2.13)

7.6. Thay đổi thanh ghi đoạn mặc định

Như đã nói ở phần trên, khi sử dụng các lệnh định địa chỉ thanh ghi, ta chỉ cần dùng các thanh ghi để xác định độ lệch còn các thanh ghi đoạn thì được hiểu mặc định. Ví dụ như ta dùng lệnh MOV AH,[BP] thì sẽ đưa dữ liệu tại ô nhớ SS:BP vào thanh ghi AH. Trong trường hợp không muốn dùng thanh ghi đoạn mặc định, ta có thể thay đổi bằng cách thêm tên thanh ghi đoạn vào để loại bỏ thanh ghi đoạn mặc định. Ví dụ lệnh MOV AH,CS:[BP] sẽ đưa dữ liệu tại CS:[BP] vào AH.

BÀI TẬP CHƯƠNG 1

1. Cần bao nhiêu byte để tạo thành một word 32 bit?

2. Giả sử µP có tất cả 16 đường địa chỉ, hỏi nó có thể xử lý tất cả bao nhiêu địa chỉ?

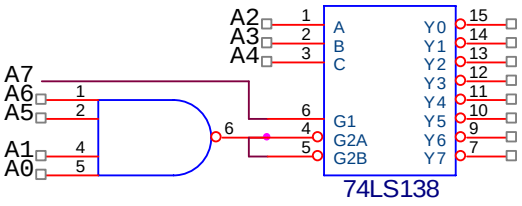
3. Nếu một IC nhớ có dung lượng 1024 × 4 bits thì để tạo 2KB bộ nhớ phải cần bao nhiêu IC?

   Nếu một IC nhớ có dung lượng 256 × 1 bits thì để tạo 1KB bộ nhớ phải cần bao nhiêu IC?

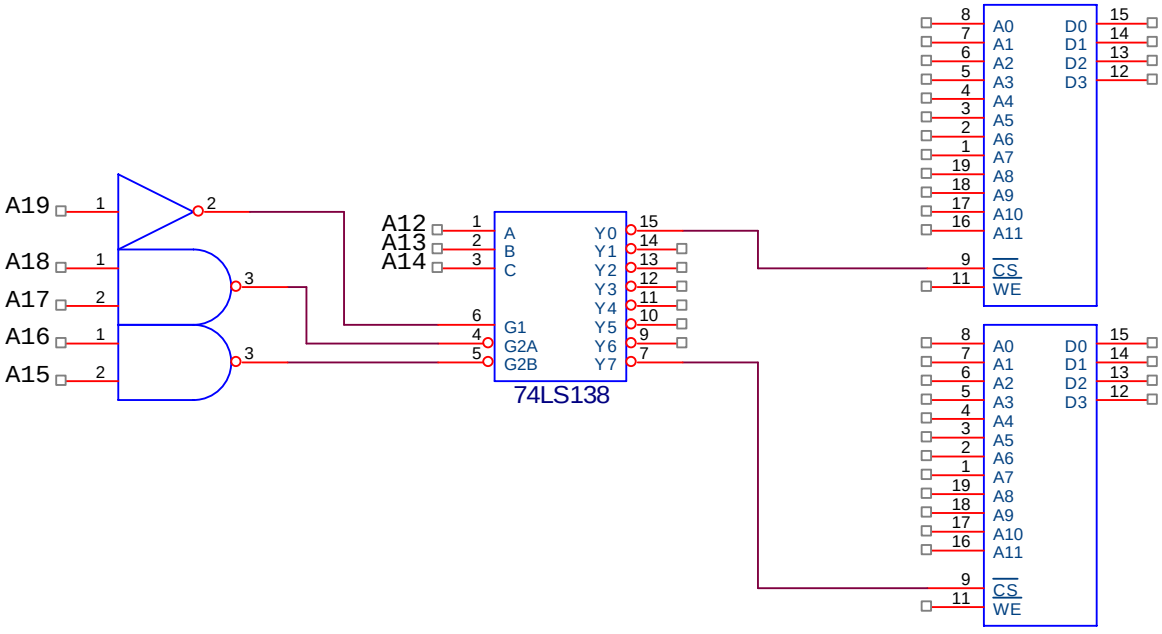
   Nếu một IC nhớ có dung lượng 8K × 8 bits thì cần phải có bao nhiêu đường địa chỉ?

4. Giả sử một IC nhớ có dung lượng 8 KB bắt đầu tại địa chỉ 1000h trong bảng bộ nhớ. Xác định vùng địa chỉ của IC.

5. Ngõ nào của bộ giải mã 74LS138 sẽ tích cực (mức thấp) nếu dữ liệu ngõ vào từ A7 ÷ A0 = 11110111? Nếu muốn ngõ ra Y4 tích cực thì dữ liệu ngõ vào phải là bao nhiêu?



6. Xác định bảng bộ nhớ:



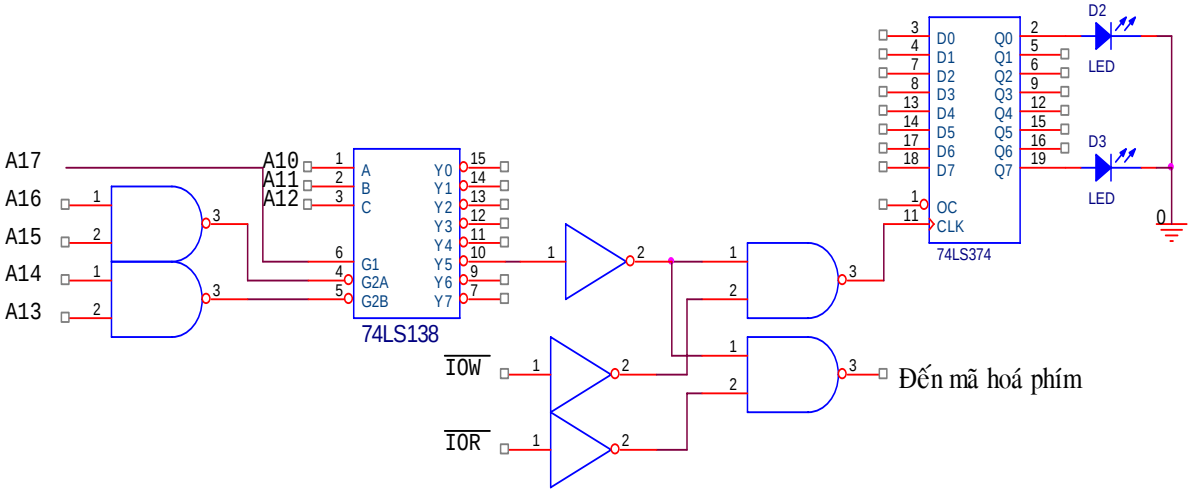
7. Thiết kế mạch giải mã địa chỉ dùng 74LS138 và các cổng logic cho 4 ROM kích thước 8K × 8 bits, 2 RAM 4K × 8 bits và 1 ROM 16K x 8 bits dùng bus địa chỉ 20 bits (A0 ÷ A19).

8. Thiết kế bộ nhớ có dung lượng 16KB từ các bộ nhớ có dung lượng 2K x 4 bits.

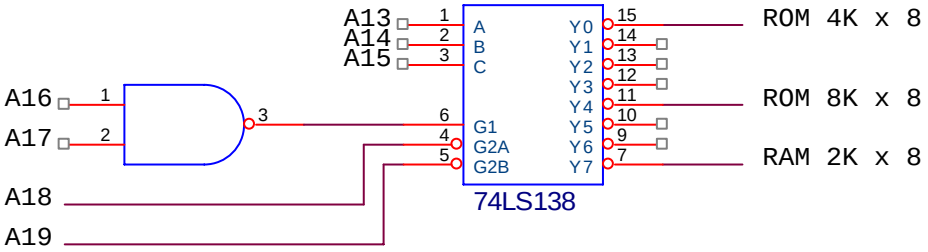
9. Thiết kế mạch giải mã địa chỉ chọn chip theo vùng địa chỉ sau:

   CS0: 8000h – 9FFFh CS2: C000h - DFFFh
   CS1: A000h – BFFFh CS3: E000h – FFFFh

10. Tìm địa chỉ của các I/O port và các giá trị của các chân $\overline{IOR}$, $\overline{IOW}$ khi thực hiện xuất dữ liệu ra Led và đọc bàn phím.



11. Xác định dãy địa chỉ của các thiết bị:



12. Xác định giá trị các cờ SF, ZF, AF và CF sau khi thực hiện các phép toán sau:

a. 1000h - 1234h
b. 1234h + 13FFh
c. ABCDh + 1234h
d. 2345h – 2345h

13. Xác định phương pháp định địa chỉ trong các lệnh sau:

a. MOV AH,DS:[SI]

b. MOV AL,AH

c. MOV DS:[BX+1],AX

d. MOV AL,DS:[BX+SI]

e. MOV CX,DS:[BX+SI+10]

f. MOV DX,20h

g. MOV DS:[10],CL

CHƯƠNG 2: LẬP TRÌNH HỢP NGỮ

1. 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.

1.1. Tập tin .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.

1.2. 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.

2. Khung của một chương trình hợp ngữ

Khung của một chương trình hợp ngữ có dạng như sau:

```
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:**

Bảng 2.1:

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
```

❖ **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.

3. Cú pháp của các lệnh trong chương trình hợp ngữ

Một dòng lệnh trong chương trình hợp ngữ gồm có các trường (field) sau (không nhất thiết phải đầy đủ tất cả các trường):

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

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 (A: hay Main:). 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 lệnh nằm phía sau dấu **;**.

3.1. 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
```

3.2. Khai báo biến

Khai báo biến nhằm để chương trình dịch cung cấp một địa chỉ xác định trong bộ nhớ. Ta dùng các lệnh giả sau để định nghĩa các biến ứng với các kiểu dữ liệu khác nhau: DB (define byte), DW (define word) và DD (define double word).

VD:

```
A1    DB    1     ; Định nghĩa biến A1 dài 1 byte (chương
                  ; trình dịch sẽ dùng 1 byte trong bộ nhớ để
                  ; lưu trữ A1), trị ban đầu A1 = 1
A2    DB    ?     ; Biến A2 kiểu byte, không có giá trị gán
```

```
                        ; ban đầu
A3      DB      'A'     ; Biến kiểu ký tự
A4      DW      1       ; Định nghĩa biến A4 dài 2 byte, giá trị ban
                        ; đầu A4 = 1, ta cũng có thể dùng dấu ? để
                        ; xác định biến không cần khởi tạo giá trị ban đầu
A5      DD      1       ; Biến A5 dài 4 byte
A6      DB      1,2,3   ; Định nghĩa biến mảng (array) gồm có 3
                        ; phần tử, mỗi phần tử dài 1 byte (nghĩa là
                        ; sẽ dùng 3 byte lưu trữ) với các giá trị ban
                        ; đầu của các phần tử lần lượt là 1,2,3
A7      DB      10      DUP(0)
                        ; Khai báo biến mảng gồm 10 phần tử, mỗi
                        ; phần tử có chiều dài 1 byte với giá trị gán
                        ; ban đầu là 0
A8      DB      10      DUP(?)
                        ; Khai báo biến mảng gồm 10 phần tử, mỗi
                        ; phần tử có chiều dài 1 byte, không cần
                        ; gán giá trị ban đầu
```

Ngoài ra ta có thể dùng các toán tử DUP lồng vào nhau khi khai báo biến mảng. Giả sử ta cần khai báo mảng A9 có các giá trị gán ban đầu 1,2,3,1,1,3,2,2,1,1,3,2,2. Ta có thể thực hiện như sau:

```
        A9      DB      1,2,3,1,1,3,2,2,1,1,3,2,2
Hay:    A9      DB      1,2,3,2 DUP(1,1,3,2,2)
Hay:    A9      DB      1,2,3,2 DUP(2 DUP(1),3,2 DUP(2))
```

Đối với các biến có nhiều hơn 1 byte, byte thấp sẽ chứa ở ô nhớ có địa chỉ thấp và byte cao sẽ chứa ở ô nhớ có địa chỉ cao.

VD:

```
        A10     DW      1234h
```

Biến A10 giả sử bắt đầu lưu tại địa chỉ 1000h thì ô nhớ 1000h chứa giá trị 34h còn ô nhớ 1001h chứa giá trị 12h.

Đối với biến kiểu chuỗi (string), thực chất là một mảng các ký tự, ta có thể khai báo như sau:

```
        A11     DB      'ABCD'
Hay     A11     DB      65h,66h,67h,68h
```

Sau lệnh khai báo này thì ô nhớ 1000h (giả sử biến A11 lưu trữ tại địa chỉ 1000h) chứa 'A', 1001h chứa 'B', 1002h chứa 'C' và 1003h chứa 'D'.

3.3. Khai báo hằng

Các hằng khai báo trong chương trình hợp ngữ bằng lệnh giả EQU để chương trình dễ hiểu hơn. Hằng có thể ở dạng số, ký tự hay chuỗi.

VD:

```
A12   EQU  10
A13   EQU  'AAA'
```

Sau khi sử dụng khai báo này, nếu ta dùng lệnh:

```
MOV  AH,A12
```

thì AH = 10h

```
A14   DB    'B',A13
```

thì khai báo chuỗi A14 với giá trị gán ban đầu là 'BAAA'.

4. Các toán tử trong hợp ngữ

❖ **Toán tử số học:**

Bảng 2.2:

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.

```
VD:  MOV  AH,(8+1)*7/3          ; AH ← 21
     MOV  AH, 00010001b shr 2   ; AH ← 0000 0100b
     MOV  AH,00010001b shl 2    ; AH ← 0100 0100b
     MOV  AH,100 mod 3          ; AH ← 1
```

❖ **Toán tử logic:**

Bao gồm các toán tử AND, OR, NOT, XOR

```
VD:  MOV  AH,10 OR 4 AND 2      ; AH = 10
     MOV  AH, 0F0h AND 7Fh      ; AH = 70h
```

❖ **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.

Bảng 2.3:

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

❖ **Các 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
```

5. Các cách định địa chỉ trong hợp ngữ

❖ **Toán hạng trực tiếp:**

Toán hạng trực tiếp là một biểu thức hằng xác định. Các hằng số có thể ở dạng thập phân (có dấu và không dấu), nhị phân, thập lục phân, các hằng số định nghĩa bằng lệnh EQU, …

```
VD:  MOV  AH,10
     MOV  AH,1010b
     MOV  AH,0Ah
     MOV  AH,A12
     MOV  AX,OFFSET msg
     MOV  AX,SEG msg
```

❖ **Toán hạng thanh ghi:**

Các thanh ghi có thể sử dụng trong phép định địa chỉ thanh ghi là AH, BH, CH, DH, AL, BL, CL, DL, AX, BX, CX, DX, SP, BP, SI, DI, CS, DS, ES, SS.

❖ **Toán hạng bộ nhớ:**

➢ **Trực tiếp:**

Toán hạng này xác định dữ liệu lưu trong bộ nhớ tại một địa chỉ xác định khi dịch, địa chỉ này là một biểu thức hằng (có thể kết hợp với toán tử chỉ số [ ] hay toán tử +, -, :). Thanh ghi đoạn mặc định là thanh ghi DS nhưng ta có thể dùng toán tử : để chỉ thanh ghi đoạn khác.

```
VD:  A      DW   1000h
     B      DB   100   DUP(0)
     MOV  AX,A              ; Chuyển nội dung của biến A vào
     MOV  AX,[A]            ; thanh ghi AX
     MOV  AH,B              ; Truy xuất phần tử đầu tiên của
     MOV  AH,B[0]           ; mảng B
     MOV  AH,B + 1          ; Truy xuất phần tử thứ hai của
     MOV  AH,B[1]           ; mảng B
     MOV  AH,B  + 5         ; Truy xuất phần tử thứ 6 của
     MOV  AH,B[5]           ; mảng B
```

Chú ý rằng lệnh MOV AX,[1000h] sẽ chuyển giá trị 1000h vào thanh ghi AX. Nếu muốn chuyển nội dung tại ô nhớ 1000h vào thanh ghi AX thì phải dùng lệnh MOV AX,DS:[1000h] hay MOV AX,DS:1000h

➢ **Gián tiếp:**

Toán hạng bộ nhớ gián tiếp cho phép dùng các thanh ghi BX, BP, SI, DI để chỉ các giá trị trong bộ nhớ.

VD:
```
MOV  BX,2
MOV  SI,3
MOV  AH,B[BX]          ; Chuyển phần tử thứ 3 của mảng B
                       ; vào thanh ghi AH
MOV  AH,B[BX+1]        ; Chuyển phần tử thứ 4 của mảng B
MOV  AH,B[BX]+1        ; vào thanh ghi AH (BX + 1 = 3)
MOV  AH,B[BX+SI]       ; Chuyển phần tử thứ 6 của mảng B
MOV  AH,B[BX][SI]      ; vào thanh ghi AH
MOV  AH,[B+BX+SI]      ; BX + SI = 5
MOV  AH,[B][BX][SI]
MOV  AH,B[BX+SI+5]     ; Chuyển phần tử thứ 11 của mảng B
MOV  AH,B[BX][SI]+5    ; vào thanh ghi AH
MOV  AH,[B+BX+SI+5]    ; BX + SI + 5 = 10
```

6. Tạo và thực thi chương trình hợp ngữ

Ta có thể tạo và thực thi một chương trình hợp ngữ trên một máy PC theo các bước sau:

- Dùng một chương trình soạn thảo văn bản ***không định dạng*** (như NC, Notepad, …) tạo một tập tin chứa chương trình hợp ngữ (gán phần mở rộng của tập tin này là .ASM, giả sử là TEMP.ASM).
- Dùng chương trình TASM.EXE (Turbo Assembler) để dịch ra mã máy dạng .OBJ: **TASM TEMP**
- Sau khi dịch xong, ta sẽ được file TEMP.OBJ chứa các mã máy của chương trình. Để chuyển thành file thực thi, ta dùng chương trình TLINK.EXE để chuyển thành tập tin .EXE: **TLINK TEMP**
- Nếu tập tin thực thi ở dạng .COM thì ta dùng thêm chương trình EXE2BIN.EXE: **EXE2BIN TEMP TEMP.COM**

7. Tập lệnh hợp ngữ

7.1. Nhóm lệnh chuyển dữ liệu

7.1.1. Nhóm lệnh chuyển dữ liệu đa dụng

❖ Lệnh **MOV dst,src**: chuyển nội dung toán hạng src vào toán hạng dst.

Toán hạng nguồn src có thể là thanh ghi (reg), bộ nhớ (mem) hay giá trị tức thời (immed); toán hạng đích dst có thể là reg hay mem.

Lệnh MOV có thể có các trường hợp sau:

```
Reg8 ← reg8          MOV  AL,AH
Reg16 ← reg16        MOV  AX,BX
Mem8 ← reg8          MOV  [BX],AL
Reg8 ← mem8          MOV  AL,[BX]
Mem16 ← reg16        MOV  [BX],AX
Reg16 ← mem16        MOV  AX,[BX]
Reg8 ← immed8        MOV  AL,04h
Mem8 ← immed8        MOV  mem[BX],01h
Reg16 ← immed16      MOV  AL,0F104h
```

Mem16 ← immed16	MOV mem[BX],0101h
SegReg ← reg16	MOV DS,AX
SegReg ← mem16	MOV DS,mem
Reg16 ← segreg	MOV AX,DS
Mem16 ← segreg	MOV [BX],DS

- Lệnh MOV không ảnh hưởng đến các cờ.
- Không thể chuyển trực tiếp dữ liệu giữa hai ô nhớ mà phải thông qua một thanh ghi

```
MOV  AX,mem1

MOV  mem2,AX
```

- Không thể chuyển giá trị trực tiếp vào thanh ghi đoạn

```
MOV  AX,1010h

MOV  DS,AX
```

- Không thể chuyển trực tiếp giữa 2 thanh ghi đoạn
- Không thể dùng thanh ghi CS làm toán hạng đích

❖ Lệnh **XCHG dst,src**: (Exchange) hoán chuyển nội dung 2 toán hạng.

Toán hạng chỉ có thể là reg hay mem.

- Lệnh XCHG không ảnh hưởng đến các cờ
- Không thể dùng cho các thanh ghi đoạn

❖ Lệnh **PUSH src**: cất nội dung một thanh ghi vào stack.

Toán hạng chỉ có thể là reg16

❖ Lệnh **POP dst**: lấy dữ liệu 16 bit từ stack đưa vào toán hạng dst.

Ta có thể dùng nhiều lệnh PUSH để cất dữ liệu vào stack nhưng khi dùng lệnh POP để lấy dữ liệu ra thì phải dùng theo thứ tự ngược lại.

```
PUSH    AX
PUSH    BX
PUSH    CX
…
POP     CX
POP     BX
POP     AX
```

❖ Lệnh **XLAT [src]**: chuyển nội dung của ô nhớ 8 bit vào thanh ghi AL.

Địa chỉ ô nhớ xác định bằng cặp thanh ghi DS:BX (nếu không chỉ ra src) hay src, địa chỉ offset chứa trong thanh ghi AL.

Lệnh XLAT tương đương với các lệnh:

```
MOV  AH,0
MOV  SI,AX
```

```
MOV  AL,[BX+SI]
```

7.1.2. Nhóm lệnh chuyển địa chỉ

- Lệnh **LEA reg16,mem16**: (Load Effective Address) chuyển địa chỉ offset của toán hạng bộ nhớ vào thanh ghi reg16.

Lệnh này sẽ tương đương với **MOV reg16, OFFSET mem16**

- Lệnh **LDS reg16,mem32**: (Load pointer using DS) chuyển nội dung bộ nhớ toán hạng mem32 vào cặp thanh ghi DS:reg16.

Lệnh LDS AX,mem tương đương với:

```
MOV  AX,mem
MOV  BX,mem+2
MOV  DS,BX
```

- Lệnh **LES reg16,mem32**: (Load pointer using ES) giống như lệnh LDS nhưng dùng cho thanh ghi ES

7.1.3. Nhóm lệnh chuyển cờ hiệu

- Lệnh **LAHF**: (Load AH from flag) chuyển các cờ SF, ZF, AF, PF và CF vào các bit 7,6,4,2 và 0 của thanh ghi AH (3 bit còn lại không đổi)

- Lệnh **SAHF**: (Store AH into flag) chuyển các bit 7,6,4,2 và 0 của thanh ghi AH vào các cờ SF, ZF, AF, PF và CF.

- Lệnh **PUSHF**: chuyển thanh ghi cờ vào stack

- Lệnh **POPF**: lấy dữ liệu từ stack chuyển vào thanh ghi cờ

7.1.4. Nhóm lệnh chuyển dữ liệu qua cổng

Mỗi I/O port giao tiếp với CPU sẽ có một địa chỉ 16 bit cho nó. CPU gởi hay nhận dữ liệu từ cổng bằng cách chỉ đến địa chỉ cổng đó. Tuỳ theo chức năng mà cổng có thể: chỉ đọc dữ liệu (input port), chỉ ghi dữ liệu (output port) hay có thể đọc và ghi dữ liệu (input/output port).

- Lệnh **IN**: đọc dữ liệu từ cổng và đưa vào thanh ghi AL

```
IN    AL,port8
IN    AL,DX
```

Nếu địa chỉ port chỉ có 8 bit thì có thể đưa giá trị trực tiếp vào, nếu là 16 bit thì phải thông qua thanh ghi AX.

- Lệnh **OUT**: ghi dữ liệu trong thanh ghi AL ra cổng

```
OUT  port8,AL
OUT  DX,AL
```

VD:

```
MOV  AL,3
OUT  61h,AL          ; Gởi giá trị 03h ra cổng 61h
MOV  AL,1
```

```
MOV  DX,03F8h          ; Xuất ra cổng máy in
OUT  DX,AL
MOV  DX,03F8h
IN   AL,DX             ; Đọc dữ liệu từ cổng máy in
```

7.2. Nhóm lệnh chuyển điều khiển

7.2.1. Lệnh nhảy không điều kiện JMP

JMP label

JMP reg/mem

Lệnh JMP dùng để chuyển điều khiển chương trình từ vị trí này sang vị trí khác (thay đổi nội dung cặp thanh ghi CS:IP).

7.2.2. Lệnh nhảy có điều kiện

Lệnh nhảy có điều kiện chỉ sử dụng cho các nhãn nằm trong khoảng từ –127 đến 128 byte so với vị trí của lệnh.

❖ Lệnh **JA label**: (Jump if Above)

Nếu CF = 0 và ZF = 0 thì JMP label

❖ Lệnh **JAE label**: (Jump if Above or Equal)

Nếu CF = 0 thì JMP label

❖ Lệnh **JB label**: (Jump if Below)

Nếu CF = 1 thì JMP label

❖ Lệnh **JBE label**: (Jump if Below or Equal)

Nếu CF = 1 hoặc ZF = 1 thì JMP label

❖ Lệnh **JNA label**: (Jump if Not Above)

Giống lệnh JBE

❖ Lệnh **JNAE label**: (Jump if Not Above or Equal)

Giống lệnh JB

❖ Lệnh **JNB label**: (Jump if Not Below)

Giống lệnh JAE

❖ Lệnh **JNBE label**: (Jump if Not Below or Equal)

Giống lệnh JA

❖ Lệnh **JG label**: (Jump if Greater)

Nếu SF = OF và ZF = 0 thì JMP label

❖ Lệnh **JGE label**: (Jump if Greater or Equal)

Nếu SF = OF thì JMP label

❖ Lệnh **JL label**: (Jump if Less)

Nếu SF <> OF thì JMP label

❖ Lệnh **JLE label**: (Jump if Less or Equal)

Nếu CF <> OF hoặc ZF = 1 thì JMP label

❖ Lệnh **JNG label**: (Jump if Not Greater)

Giống lệnh JLE

❖ Lệnh **JNGE label**: (Jump if Not Greater or Equal)

Giống lệnh JL

❖ Lệnh **JNL label**: (Jump if Not Less)

Giống lệnh JGE

❖ Lệnh **JNLE label**: (Jump if Not Less or Equal)

Giống lệnh JG

❖ Lệnh **JC label**: (Jump if Carry)

Giống lệnh JB

❖ Lệnh **JNC label**: (Jump if Not Carry)

Giống lệnh JNB

❖ Lệnh **JZ label**: (Jump if Zero)

Nếu ZF = 1 thì JMP label

❖ Lệnh **JE label**: (Jump if Equal)

Giống lệnh JZ

❖ Lệnh **JNZ label**: (Jump if Not Zero)

Nếu ZF = 0 thì JMP label

❖ Lệnh **JNE label**: (Jump if Equal)

Giống lệnh JNZ

❖ Lệnh **JS label**: (Jump on Sign)

Nếu SF = 1 thì JMP label

❖ Lệnh **JNS label**: (Jump if No Sign)

Nếu SF = 0 thì JMP label

❖ Lệnh **JO label**: (Jump on Overflow)

Nếu OF = 1 thì JMP label

❖ Lệnh **JNO label**: (Jump if No Overflow)

Nếu OF = 0 thì JMP label

❖ Lệnh **JP label**: (Jump on Parity)

Nếu PF = 1 thì JMP label

❖ Lệnh **JNP label**: (Jump if No Parity)

Nếu PF = 0 thì JMP label

❖ Lệnh **JCXZ label**: (Jump if CX Zero)

Nếu CX = 1 thì JMP label

7.2.3. Lệnh so sánh

CMP left(reg/mem), right(reg/mem/immed)

Lệnh CMP dùng để so sánh nội dung 2 toán hạng, kết quả chứa vào thanh ghi cờ và không làm thay đổi nội dung các toán hạng.

VD: Đoạn chương trình so sánh 2 số A và B: A >B thì nhảy đến label1, A = B thì nhảy đến label2, A < B thì nhảy đến label3.

```
MOV  AX,A
CMP  AX,B
JG   label1
JL   label2
JMP  label3
```

7.2.4. Các lệnh vòng lặp

❖ Lệnh **LOOP**:

LOOP label

Mô tả:

CX = CX – 1
Nếu CX <> 0 thì JMP label

❖ Lệnh **LOOPE**:

LOOPE label

Mô tả:
CX = CX – 1
Nếu (ZF = 1) và (CX <> 0) thì JMP label

❖ Lệnh **LOOPZ**:

Giống lệnh LOOPE

❖ Lệnh **LOOPNE**:

LOOPNE label

Mô tả:

CX = CX – 1
Nếu (ZF = 0) và (CX <> 0) thì JMP label

❖ Lệnh **LOOPNZ**:

Giống lệnh LOOPNE

7.2.5. Lệnh liên quan đến chương trình con

❖ Lệnh **CALL**:

Lệnh CALL dùng để gọi một chương trình con, có thể là near hay far.

```
CALL    label     ; Gọi chương trình con tại vị trí xác định
                  ; bởi nhãn label
CALL    reg/mem   ; Gọi chương trình con tại vị trí xác định
                  ; trong reg/mem
```

❖ Lệnh **RET**: (return)

RET [n]

RETN [n]

RETF [n]

Lệnh RET dùng để kết thúc chương trình con, điều khiển sẽ được đưa về địa chỉ trước khi gọi chương trình con. RETN để kết thúc chương trình con dạng near và RETF dùng để kết thúc chương trình con dạng far.

Trong trường hợp lệnh RET có hằng số n theo sau thì sẽ cộng với thanh ghi SP giá trị n (n phải là số chẵn). Lệnh này dùng để loại bỏ một số tham số chương trình con sử dụng ra khỏi stack.

7.3. Nhóm lệnh xử lý số học

7.3.1. Xử lý phép cộng

❖ Lệnh **ADD dst,src**:

dst ← dst + src

Toán hạng src có thể là reg, mem hay immed còn toán hạng dst là reg hay mem.

- Không thể cộng trực tiếp 2 thanh ghi đoạn
- Lệnh ADD ảnh hưởng đến các cờ sau:
  + Cờ CF: = 1 khi kết quả phép cộng có nhớ hay có mượn
  + Cờ AF: = 1 khi kết quả phép cộng có nhớ hay có mượn đối với 4 bit thấp
  + Cờ PF: = 1 khi kết quả phép cộng có tổng 8 bit thấp là một số chẵn.
  + Cờ ZF: = 1 khi kết quả phép cộng là 0.
  + Cờ SF: = 1 nếu kết quả phép cộng là một số âm
  + Cờ OF: = 1 nếu kết quả phép cộng bị sai dấu, nghĩa là vượt ra ngoài phạm vi lớn nhất hay nhỏ nhất mà số có dấu có thể chứa trong toán hạng dst.

❖ Lệnh **ADC dst, src**: (Add with Carry)

dst ← dst + src + CF

Lệnh ADC thường dùng để cộng các số lớn hơn 16 bit.

❖ Lệnh **INC dst**: (Increment)

dst ← dst + 1

Dst có thể là reg hay mem.

❖ Lệnh **AAA**: (ASCII Adjust for Addition)

Hiệu chỉnh kết quả phép cộng 2 số BCD dạng không nén (mỗi chữ số BCD lưu bằng 1 byte).

```
VD:  MOV  AX,9
     MOV  BX,3
     ADD  AL,BL        ; Kết quả là AX = 0Ch
     AAA               ; AX = 0102h (AH = 1, AL = 2)
```

Lệnh AAA chỉ ảnh hưởng đến các cờ AF và CF, không ảnh hưởng đến các cờ còn lại.

❖ Lệnh **DAA**: (Decimal Adjust for Addition)

Hiệu chỉnh kết quả phép cộng 2 số BCD dạng nén (mỗi chữ số BCD lưu bằng 4 bit, nghĩa là 1 byte biểu diễn được các số nguyên từ 0 đến 99).

```
VD:  MOV  AX,4338h
     ADD  AL,AH        ; AX ← 437Bh
     DAA               ; AX ← 4381h (43 + 38 = 81)
```

Lệnh DAA chỉ ảnh hưởng đến các cờ AF, CF, PF, SF, ZF và không ảnh hưởng đến thanh ghi AH.

7.3.2. Xử lý phép trừ

❖ Lệnh **SUB dst,src**:

dst ← dst - src

Toán hạng src có thể là reg, mem hay immed còn toán hạng dst chỉ có thể là reg hay mem.

- Không thể trừ trực tiếp thanh ghi đoạn
- Ảnh hưởng đến các cờ AF, CF, OF, PF, SF và ZF.

❖ Lệnh **SBB dst,src**:

dst ← dst – src – CF

Lệnh ADC thường dùng để trừ các số lớn hơn 16 bit.

❖ Lệnh **DEC dst**: (decrement)

dst ← dst – 1

dst là reg hay mem. Lệnh DEC ảnh hưởng đến các cờ AF, OF, PF, SF, ZF.

❖ Lệnh **NEG dst**:

dst ← - dst

dst là reg hay mem.

Lệnh NEG ảnh hưởng đến các cờ:

CF = 1 nếu nội dung kết quả là số khác 0.
SF = 1 nếu nội dung kết quả là số âm khác 0.
PF = 1 nếu tổng 8 bit thấp là một số chẵn.
ZF = 1 nếu nội dung kết quả là 0.
OF = 1 nếu nội dung toán hạng dst là 80h (dạng byte) hay 8000h (dạng word).

VD: Nếu muốn thực hiện phép toán 100 – AH, ta không thể cùng lệnh:

```
SUB   100,AH
```

mà phải dùng lệnh:

```
SUB   AH,100
NEG   AH
```

❖ Lệnh **AAS**: (Ascii Adjust for Substract)

Hiệu chỉnh kết quả phép trừ 2 số BCD dạng không nén (mỗi chữ số BCD lưu bằng 1 byte). Lệnh AAS chỉ ảnh hưởng cờ AF và CF.

❖ Lệnh **DAS**: (Decimal Adjust for Substract)

Hiệu chỉnh kết quả phép trừ 2 số BCD dạng nén (mỗi chữ số BCD lưu bằng 4 bit). Lệnh AAS chỉ ảnh hưởng cờ AF và CF.

7.3.3. Xử lý phép nhân

❖ Lệnh **MUL src**:

Nếu src là reg hay mem 8 bit: AX ← AL*src

Nếu src là reg hay mem 16 bit: DX:AX ← AX*src

Lệnh MUL chỉ ảnh hưởng đến cờ CF và OF.

❖ Lệnh **IMUL src**:

Giống như lệnh MUL nhưng kết quả là số có dấu.

❖ Lệnh **AAM**: (Ascii Adjust for Multiple)

Hiệu chỉnh kết quả phép nhân 2 số BCD dạng không nén, lệnh AAM thực hiện chia AL cho 10, lưu phần thương vào AL và phần dư vào AH. Lệnh AAM ảnh hưởng đến các cờ PF, SF và ZF.

7.3.4. Xử lý phép chia

❖ Lệnh **DIV src**:

Nếu src là reg/mem 8 bit: AL ← AX DIV src và AH ← AX MOD src

Nếu src là reg/mem 16 bit: AX ← DX:AX DIV src và DX ← DX:AX MOD src

Lệnh DIV không ảnh hưởng đến các cờ nhưng xảy ra tràn trong các trường hợp sau:

- Chia cho 0
- Thương lớn hơn 256 đối với dạng 8 bit.
- Thương lớn hơn 65536 đối với dạng 16 bit.

❖ Lệnh **IDIV src**:

Giống như lệnh DIV nhưng kết quả là số có dấu. Các trường hợp tràn:

- Chia cho 0
- Thương nằm ngoài khoảng (-128,127) đối với dạng 8 bit.
- Thương nằm ngoài khoảng (-32767,32768) đối với dạng 16 bit.

❖ Lệnh **AAD**: (Ascii Adjust for Division)

Hiệu chỉnh kết quả phép chia 2 số BCD dạng không nén. Lưu ý rằng lệnh AAD phải được thực hiện trước lệnh chia. Sau khi thực hiện chia thì phải hiệu chỉnh lại dạng BCD bằng cách dùng lệnh AAM.

❖ Lệnh **CBW**: (Convert Byte to Word)

Nếu AL < 80h thì AH = 0, ngược lại AH = 0FFh

Lệnh CBW dùng để chuyển số nhị phân có dấu 8 bit thành số nhị phân có dấu 16 bit.

❖ Lệnh **CWD**: (Convert Word to Double word)

Nếu AX < 8000h thì DX = 0, ngược lại DX = 0FFFFh

Lệnh CWD dùng để chuyển số nhị phân có dấu 16 bit thành số nhị phân có dấu 32 bit chứa trong DX:AX.

7.3.5. Dịch chuyển và quay

❖ Lệnh **SHL**: (Shift Logical Left)

SHL dst,1

SHL dst,CL

Dịch trái 1 bit hay CL bit.

CF ← dst7 ← dst6 … ← dst0 ← 0

❖ Lệnh **SHR**: (Shift Logical Right)

SHR dst,1

SHR dst,CL

Dịch phải 1 bit hay CL bit.

0 → dst7 → dst6 … → dst0 → CF

❖ Lệnh **SAL**: giống SHL

❖ Lệnh **SAR**:

Giống như lệnh SHR nhưng giá trị bit dst7 không thay đổi, nghĩa là

dst7 → dst7 → dst6 … → dst0 → CF

❖ Lệnh **ROL**: (Rotate Left)

ROL dst,1

ROL dst,CL

Quay trái 1 bit hay CL bit.

CF ← dst7 ← dst6 … ← dst0 ← dst7

❖ Lệnh **ROR**: (Rotate Right)

ROR dst,1

ROR dst,CL

Quay phải 1 bit hay CL bit.

dst0 → dst7 → dst6 … → dst0 → CF

❖ Lệnh **RCL**: (Rotate though Carry Left)

RCL dst,1

RCL dst,CL

Quay trái 1 bit hay CL bit.

CF ← dst7 ← dst6 … ← dst0 ← CF

❖ Lệnh **RCR**: (Rotate though Carry Right)

RCR dst,1

RCR dst,CL

Quay phải 1 bit hay CL bit.

CF → dst7 → dst6 … → dst0 → CF

7.3.6. Các lệnh logic

❖ Lệnh **AND**:

AND dst,src

dst ← dst AND src

CF ← 0, OF ← 0

Src là reg, mem hay immed còn dst là reg, mem.

❖ Lệnh **OR**:

OR dst,src

dst ← dst OR src

CF ← 0, OF ← 0

❖ Lệnh **XOR**:

XOR dst,src

dst ← dst XOR src

CF ← 0, OF ← 0

❖ Lệnh **NOT**:

NOT dst

dst ← NOT dst

Lệnh NOT không ảnh hưởng đến các cờ.

❖ Lệnh **TEST**:

TEST dst,src

Lệnh TEST thực hiện phép toán AND 2 toán hạng nhưng chỉ ảnh hưởng đến các cờ và không ảnh hưởng đến toán tử.

7.4. Nhóm lệnh xử lý chuỗi

Bao gồm các lệnh sau:

- Lệnh MOVS: chuyển dữ liệu từ vùng nhớ này sang vùng nhớ khác.

+ **MOVSB**: chuyển 1 byte từ vị trí chỉ đến bởi SI đến vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 1, DI ← DI + 1 còn nếu DF = 1 thì SI ← SI - 1, DI ← DI - 1.

+ **MOVSW**: chuyển 1 word từ vị trí chỉ đến bởi SI đến vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 2, DI ← DI + 2 còn nếu DF ← 1 thì SI ← SI - 2, DI ← DI - 2.

- Lệnh CMPS: so sánh nội dung 2 vùng nhớ
  + **CMPSB**: so sánh 1 byte tại vị trí chỉ đến bởi SI và tại vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 1, DI ← DI + 1 còn nếu DF ← 1 thì SI ← SI - 1, DI ← DI - 1.
  + **CMPSW**: so sánh 1 word tại vị trí chỉ đến bởi SI và tại vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 2, DI ← DI + 2 còn nếu DF = 1 thì SI ← SI - 2, DI ← DI - 2.

- Lệnh SCAS: tìm một phần tử trong vùng nhớ, địa chỉ vùng nhớ xác định bằng cặp thanh ghi ES:DI, giá trị cần tìm đặt trong thanh ghi AL, nếu tìm thấy thì ZF = 1. Giá trị của DI và SI thay đổi giống như trên.

- Lệnh LODS: đưa một byte hay word có địa chỉ xác định bởi cặp thanh ghi DS:SI vào thanh ghi AL hay AX. Giá trị của DI và SI thay đổi giống như trên.

- Lệnh STOS: chuyển nội dung của AL hay AX vào vùng nhớ xác định bởi cặp thanh ghi ES:DI. Giá trị của DI và SI thay đổi giống như trên.

8. Các cấu trúc cơ bản trong lập trình hợp ngữ

8.1. 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.

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
```

8.2. 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:
```

8.3. 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
            JG    DUONG
DUONG:      MOV   BH,0
            JMP   NEXT
AM:         MOV   BH,1
            JMP   NEXT
KHONG:      MOV   BH,2
NEXT:
```

8.4. 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
```

8.5. Cấu trúc lặp WHILE

WHILE Điều kiện DO 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:
```

8.6. Cấu trúc lặp REPEAT

REPEAT Công việc UNTIL Điều kiện

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:
```

9. Các ngắt của 8086

Bảng 2.4:

Vector ngắt	Công dụng
00h	CPU: tác động khi chia cho 0
01h	CPU: chương trình thực thi từng bước
02h	CPU: ngắt không che được
03h	CPU: tạo điểm dừng chochương trình
04h	CPU: tác động khi kết quả số học tràn
05h	Tác động khi nhấn Print Screen
06h - 07h	Dành riêng
08h	Tác động bởi nhịp đồng hồ (18.2 lần/s)
09h	Tác động khi có phím nhấn
0Ah	Dành riêng
0Bh - 0Ch	Tác động phần cứng liên lạc nối tiếp
0Dh	Đĩa cứng
0Eh	Đĩa mềm
0Fh	Máy in
10h	BIOS: màn hình
11h	BIOS: xác định cấu hình máy tính
12h	BIOS: thông báo kích thước RAM
13h	BIOS: gọi các phục vụ đĩa cứng/mềm

14h	BIOS: giao tiếp nối tiếp
15h	BIOS: truy xuất cassette hay mở rộng ngắt
16h	BIOS: xuất / nhập bàn phím
17h	BIOS: máy in
18h	Xâm nhập ROM basic
19h	BIOS: khởi động máy tính
1Ah	BIOS: ngày / giờ hệ thống
1Bh	Lấy điều khiển từ ngắt bàn phím
1Ch	Lấy điều khiển từ ngắt đồng hồ (sau int 08h)
1Dh	Địa chỉ bảng tham số màn hình
1Eh	Địa chỉ bảng tham số đĩa
1Fh	Địa chỉ bộ mã ký tự
20h	DOS: kết thúc chương trình
21h	DOS: các chức năng DOS
22h	Địa chỉ cần chuyển khi kết thúc chương trình
23h	Địa chỉ cần chuyển khi gặp Ctrl – Break
24h	Địa chỉ cần chuyển khi gặp lỗi
25h	DOS: đọc đĩa cứng / mềm
26h	DOS: ghi đĩa cứng / mềm
27h	DOS: chấm dứt chương trình và thường trú
28h – 3Fh	Dành riêng cho DOS
40h	BIOS: các chức năng đĩa mềm
41h	Bảng thông số đĩa cứng thứ nhất
42h – 45h	Dành riêng
46h	Bảng thông số đĩa cứng thứ hai
47h – 49h	Định nghĩa do người sử dụng
4Ah	Giờ báo hiệu (chỉ trong AT)
4Bh – 67h	Định nghĩa do người sử dụng
68h – 6Fh	Không sử dụng
70h	Đồng hồ thời gian thực (chỉ trong AT)
71h – 7Fh	Dành riêng
80h – 85h	Dành riêng
86h – F0h	Sử dụng bởi chương trình thông dịch BASIC
F1h – FFh	Không sử dụng

9.1. Ngắt 21h

❖ **Hàm 01h**: nhập một ký tự từ bàn phím và hiện ký tự nhập ra màn hình. Nếu không có ký tự nhập, hàm 01h sẽ đợi cho đến khi nhập.

- Gọi: AH = 01h
- Trả về: AL chứa mã ASCII của ký tự nhập

```
MOV  AH,01h
INT   21h          ; AL chứa mã ASCII của ký tự nhập
```

❖ **Hàm 02h**: xuất một ký tự trong thanh ghi DL ra màn hình tại vị trí con trỏ hiện hành

- Gọi AH = 02h, DL = mã ASCII của ký tự
- Trả về: không có

```
MOV  AH,02h
MOV  DL,'A'
INT  21h
```

❖ **Hàm 08h**: giống hàm 01h nhưng không hiển thị ký tự ra màn hình

❖ **Hàm 09h**: xuất một chuỗi ký tự ra màn hình tại vị trí con trỏ hiện hành, địa chỉ chuỗi được chứa trong DS:DX và phải được kết thúc bằng ký tự $

- Gọi AH = 09h, DS:DX = địa chỉ chuỗi
- Trả về: không có

```
.DATA
Msg  DB   'Hello$'
…
MOV  AH,09h
LEA  DX,Msg
INT  21h
```

❖ **Hàm 0Ah**: nhập một chuỗi ký tự từ bàn phím (tối đa 255 ký tự), dùng phím ENTER kết thúc chuỗi

- Gọi AH = 0Ah, DS:DX = địa chỉ lưu chuỗi
- Trả về: không có

Chuỗi phải có dạng sau:

- Byte 0: Số byte tối đa cần đọc (kể cả ký tự Enter)
- Byte 1: số byte đã đọc
- Byte 2: lưu các ký tự đọc

```
.DATA
Msg  DB   101          ; Đọc tối đa 100 ký tự
     DB   ?
     DB   101  DUP(?)
     …
     MOV  AH,0Ah
     LEA  DX,Msg
     INT  21h
```

❖ **Hàm 0Bh**: kiểm tra phím nhấn trên bàn phím

- Gọi: AH = 0Bh
- Trả về: AL = 0FFh nếu có nhấn phím, AL = 0 nếu không nhấn phím

❖ **Hàm 4Ch**: kết thúc chương trình

```
MOV   AH,4Ch
INT   21h
```

9.2. Ngắt 10h

❖ **Xoá màn hình:**

- Gọi AX = 02h

- Trả về: không có

```
MOV  AX,02h
INT  10h
```

❖ **Chuyển toạ độ con trỏ:**

- Gọi AH = 02h, DH = dòng, DL = cột

```
MOV  AH,02h
MOV  DX,0F15h
INT  10h
```

10. Truyền tham số giữa các chương trình

Trong lập trình, một vấn đề ta cần quan tâm là truyền tham số giữa chương trình chính và chương trình con. Để thực hiện truyền tham số, ta có thể dùng các cách sau đây:

- Truyền tham số qua thanh ghi
- Truyền tham số qua ô nhớ (biến)
- Truyền tham số qua ô nhớ do thanh ghi chỉ đến
- Truyền tham số qua stack

10.1. Truyền tham số qua thanh ghi

Ta thực hiện truyền tham số qua thanh ghi bằng cách: một chương trình con sẽ đưa giá trị vào thanh ghi và chương trình con khác sẽ xử lý giá trị trên thanh ghi đó.

VD: Cộng giá trị tại 2 ô nhớ 1000h và 1001h, kết quả chứa trong 1002h (bye cao) và 1003h (byte thấp).

```
.MODEL    SMALL
.STACK    100h
.CODE
main PROC
     MOV       AX,@DATA
     MOV       DS,AX
; ••a giá tr• vào các ô nh•
     MOV       BYTE PTR DS:[1000h],10h
     MOV       BYTE PTR DS:[1001h],0FFh
     CALL      Read
     CALL      Sum
     Mov       AH,4Ch
     INT       21h
main ENDP
Read PROC      ; ••c d• li•u vào thanh ghi AX
     MOV       AH,DS:[1000h]
     MOV       AL,DS:[1001h]
     RET
Read ENDP

Sum  PROC
     ADD       AH,AL
     JZ        next
     MOV       DS:[1003h],1
next:     MOV       DS:[1002h],AH
```

```
RET
Sum  ENDP
END  main
```

10.2. Truyền tham số qua ô nhớ (biến)

Quá trình truyền tham số cũng giống như trên nhưng thay vì thực hiện thông qua thanh ghi, ta sẽ thực hiện thông qua các ô nhớ.

VD: Cộng giá trị tại 2 ô nhớ m1 và m2, kết quả chứa trong m3 (byte cao) và m4 (byte thấp).

```
.MODEL    SMALL
.STACK    100h
.DATA
     m1   db   ?
     m2   db   ?
     m3   db   ?
     m4   db   ?
.CODE
main PROC
     MOV       AX,@data
     MOV       DS,AX
     MOV       m1,10h    ; ••a giá tr• vào
     MOV       m2,0FFh   ; các ô nh•
     CALL      Sum
     MOV       AH,4Ch
     INT       21h
main ENDP
Sum  PROC
     MOV       m4,0
     MOV       AH,m1
     ADD       AH,m2
     JNC       next
     MOV       m4,1
next:     MOV       m3,AH
RET
Sum  ENDP
END  main
```

10.3. Truyền tham số qua ô nhớ do thanh ghi chỉ đến

Trong cách truyền tham số này, ta dùng các thanh ghi SI, DI, BX để chỉ địa chỉ offset của các tham số còn thanh ghi đoạn mặc định là DS.

VD: Cộng giá trị tại 2 ô nhớ m1 và m2, kết quả chứa trong m3 (byte cao) và m4 (byte thấp).

```
.MODEL    SMALL
.STACK    100h
.DATA
     m1   db   ?
     m2   db   ?
     m3   db   ?
     m4   db   ?
.CODE
main PROC
```

```
        MOV     AX,@data
        MOV     DS,AX
        LEA     SI,m1
        LEA     DI,m2
        LEA     BX,m3
        MOV     [SI],10h ; ••a giá tr• vào
        MOV     [DI],0FFh; các ô nh•
        CALL    Sum
        MOV     AH,4Ch
        INT     21h
main ENDP
Sum  PROC
        MOV     AL,[SI]
        ADD     AL,[DI]
        JZ      next
        MOV     [BX+1],1
next:   MOV     [BX],AL
RET
Sum  ENDP
END  main
```

10.4. Truyền tham số qua stack

Trong phương pháp truyền tham số này, ta dùng stack làm nơi chứa các tham số cần truyền thông qua các tác vụ PUSH và POP.

VD: Cộng giá trị tại 2 ô nhớ m1 và m2, kết quả chứa trong m3 (byte cao) và m4 (byte thấp).

```
.MODEL  SMALL
.STACK  100h
.DATA
        m1  dw  ?
        m2  dw  ?
        m3  dw  ?
        m4  dw  ?
.CODE
main PROC
        MOV     AX,@data
        MOV     DS,AX
        LEA     SI,m1
        LEA     DI,m2
        MOV     [SI],1234h    ; ••a giá tr• vào
        MOV     [DI],0FEDCh   ; các ô nh•
        PUSH    m1            ; ••a vào stack
        PUSH    m2
        CALL    Sum
        POP     m3   ; L•y k•t qu• ••a vào stack
        POP     m4
        MOV     AH,4Ch
        INT     21h
main ENDP
Sum  PROC
        POP     DX   ; L•u ••a ch• tr• v• c•a l•nh CALL
        POP     AX   ; L•y d• li•u t• stack
        POP     BX
```

```
            ADD     AX,BX
            JNC     next
            PUSH    1
        next:
            PUSH    AX
            PUSH    DX   ; L•y ••a ch• tr• v• c•a l•nh CALL
        RET
        Sum ENDP
        END main
```

11. Các ví dụ minh hoạ

11.1. In chuỗi ký tự ra màn hình

```
    .MODEL   SMALL
    .STACK   100h
    .DATA
        msg DB   'Hello$'
    .CODE
    main PROC
        MOV AX,@DATA      ; Kh•i ••ng thanh ghi DS
        MOV DS,AX
        MOV AX,02h        ; Xoá màn hình
        INT 10h
        MOV AH,02h        ; Chuy•n to• •• con tr•
        MOV DX,0C15h      ; ••n dòng 12 (0Ch) và c•t 21
(15h)
        INT 10h
        LEA DX,msg        ; ••a ch• thông •i•p
        MOV AH,09h        ; In thông •i•p ra màn hình
        INT 21h
        MOV AH,4Ch        ; K•t thúc ch••ng trình
        INT 21h
    main ENDP
    END main
```

11.2. In chuỗi ký tự ra màn hình tại toạ độ nhập vào

```
    .MODEL   SMALL
    .STACK   100h
    .DATA
        msg  DB   'Hello$'
        msg1 DB   'Nhap vao toa do:$'
        Crlf DB   0Dh,0Ah,'$'
        Td   DB   3
             DB   ?
             DB   3    DUP(?)
    .CODE
    main PROC
        MOV AX,@DATA
        MOV DS,AX         ; Kh•i ••ng thanh ghi DS
        MOV AX,02h
        INT 10h           ; Xóa màn hình
        LEA DX,msg1
        MOV AH,09h        ; In thông •i•p
        INT 21h
        CALL Nhap         ; Nh•p dòng
```

```
          MOV  CL,AL
          LEA  DX,Crlf        ; Xu•ng dòng
          MOV  AH,09h
          INT  21h
          CALL Nhap           ; Nh•p c•t
          MOV  CH,AL
          MOV  AH,02h         ; Chuy•n t•a •• con tr•
          MOV  DX,CX
          INT  10h
          LEA  DX,msg
          MOV  AH,09h         ; In ra màn hình
          INT  21h
          MOV  AH,4Ch         ; K•t thúc ch••ng trình
          INT  21h
     main ENDP
     Nhap PROC
          MOV  AH,0Ah         ; Nh•p vào
          LEA  DX,Td
          INT  21h
          LEA  BX,Td              ; L•y ch• s• hàng ch•c
          MOV  AL,DS:[BX+2]
          SUB  AL,'0'             ; Chuy•n t• d•ng ký t•
sang d•ng s•
          MOV  BL,10
          MUL  BL             ; Nhân s• hàng ch•c v•i 10
          PUSH AX
          LEA  BX,Td              ; L•y ch• s• hàng d•n v•
          MOV  AL,DS:[BX+3]
          SUB  AL,'0'
          POP  BX
          ADD  AL,BL
          RET
     Nhap ENDP
     END  main
```

11.3. Cộng 2 số nhị phân dài 5 byte

```
     .MODEL    SMALL
     .STACK    100h
     .DATA
          m1   DB   00h,08h,10h,13h,24h,00h
          m2   DB   0FFh,0FCh,0FAh,0F0h,0F1h,00h;
          m3   DB   6    DUP(0)
     .CODE
     main PROC
          MOV  AX,@DATA
          MOV  DS,AX          ; Kh•i ••ng thanh ghi DS
          LEA  SI,m1
          LEA  DI,m2
          LEA  BX,m3
          MOV  CX,6
          XOR  AL,AL
     next:     MOV  AL,[SI]
          ADC  AL,[DI]
          MOV  [BX],AL
          INC  BX
```

```
        INC  SI
        INC  DI
        LOOP next
        MOV  AH,4Ch
        INT  21h
    main ENDP
    END  main
```

11.4. 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:$'
    .CODE
    main PROC
        MOV  AX,@DATA
        MOV  DS,AX          ; Kh•i ••ng thanh ghi DS
        MOV  ES,AX
        LEA  DX,m1
        MOV  AH,0Ah         ; Nh•p chu•i
        INT  21h
        LEA  SI,m1          ; L•y ••a ch• chu•i
        ADD  SI,2
        MOV  DI,SI          ; Chu•i ngu•n và •ích
                            ; trùng nhau
    Next:    LODSB          ; L•y ký t•
        CMP  AL,0Dh         ; N•u là ký t• Enter
                            ; thì k•t thúc
        JE   quit
        CMP  AL,'a'         ; N•u ký t• nh•p không ph•i
                            ; là ký t• th••ng t• 'a'
        JB   cont           ; ••n 'z' thì b• qua
        CMP  AL,'z'
        JA   cont
        SUB  AL,20h         ; Chuy•n ký t• th••ng
                            ; thành ký t• hoa
        STOSB               ; L•u ký t• v•a chuy•n
        DEC  DI             ; N•u là ký t• th••ng
                            ; thì dùng l•nh STOSB
                            ; nên DI t•ng lên 1,
                            ; ta ph•i gi•m DI
    cont:    INC  DI        ; T•ng lên ký t• k•
        JMP  next
    quit:    MOV  AL,'$'
        STOSB
        MOV  AX,02h         ; Xóa màn hình
        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
```

BÀI TẬP CHƯƠNG 2

1. Xác định địa chỉ offset của các ô nhớ có địa chỉ vật lý sau, biết rằng địa chỉ segment là 1ACDh:

a. 1FFFFh b. 20000h c. 2ABCDh d. 1ECDFh

2. Xác định địa chỉ vật lý của các địa chỉ sau:

a. 1234h:1234h b. 1111h:ABCDh c. AAAAh:BBBBh

3. Xác định nội dung của các thanh ghi AX, BX và các ô nhớ 1000h, 1001h, 1002h, 1003h sau khi thực thi các đoạn chương trình sau:

a.
```
MOV  AX,1000h
MOV  BL,7
MOV  BH,0F0h
AND  BH,AH
MOV  WORD PTR DS:[1000h],0F0h
MOV  BYTE PTR DS:[1002h],0Fh
MOV AX,DS:[1001h]
MOV AL,DS:[1002h]
```
b.
```
MOV  AX,1234h
MOV  BX,0AAAAh
MOV  CH,AL
MOV  CL,AH
MOV  DS:[1000h],AX
MOV  DS:[1002h],BX
```
c.
```
MOV  AX,12h
MOV  DS:[1001h],AX
MOV  BX,AX
ADD  BH,10h
MOV  DS:[1002h],BH
```

4. Viết chương trình xuất 10 chuỗi "Hello" ra màn hình tại dòng thứ 10, cột 10.

5. Viết chương trình nhập chuỗi từ bàn phím cho đến khi nhập ký tự 'T' thì xuất chuỗi nhập ra màn hình và kết thúc chương trình.

6. Viết chương trình thực hiện chuyển đổi một chuỗi ký tự trong bộ nhớ từ chữ thường thành chữ hoa và in chuỗi đã chuyển đổi lên màn hình.

7. Viết chương trình thực hiện in các ký tự chứa tại ô nhớ 1000h – 3000h theo thứ tự ngược lại.

8. Viết chương trình thực hiện chuyển đổi số nhị phân chứa trong thanh ghi DX thành số BCD chứa trong thanh ghi AX. Nếu kết quả chuyển đổi lớn hơn 16 bit thì giá trị trong thanh ghi AX là FFFFh.

9. Viết chương trình so sánh 2 arrray 8 bit A và B, mỗi array có 100 phần tử chứa từ địa chỉ 1000h (array A) và 2000h (array B). Nếu 2 array này giống

nhau thì lưu vào ô nhớ 3000h giá trị FFFFh. Ngược lại thì lưu vào ô nhớ 3000h địa chỉ đầu tiên của phần tử trong array A khác với phần tử trong array B.

10. Viết chương trình thực hiện đọc dữ liệu 16 bit liên tục từ thiết bị ngoại vi có địa chỉ 300h.

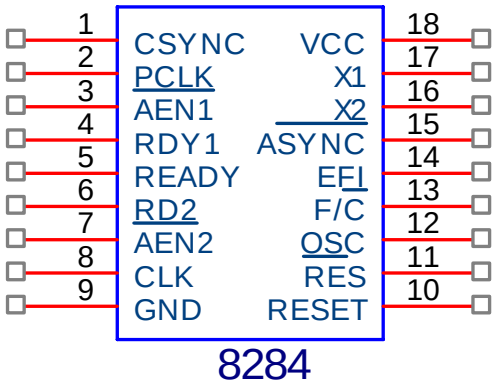
- Nếu dữ liệu đọc là 0 thì kết thúc chương trình
- Nếu dữ liệu đọc là FFFFh thì in lên màn hình chuỗi: “Error”
- Ngược lại thì xuất giá trị vừa nhập ra thiết bị ngoại vi có địa chỉ 301h

CHƯƠNG 3: TỔ CHỨC NHẬP / XUẤT

1. Các mạch phụ trợ 8284 và 8288

1.1. Mạch tạo xung nhịp 8284

Mạch tạo xung nhịp dùng để cung cấp xung nhịp cho µP.

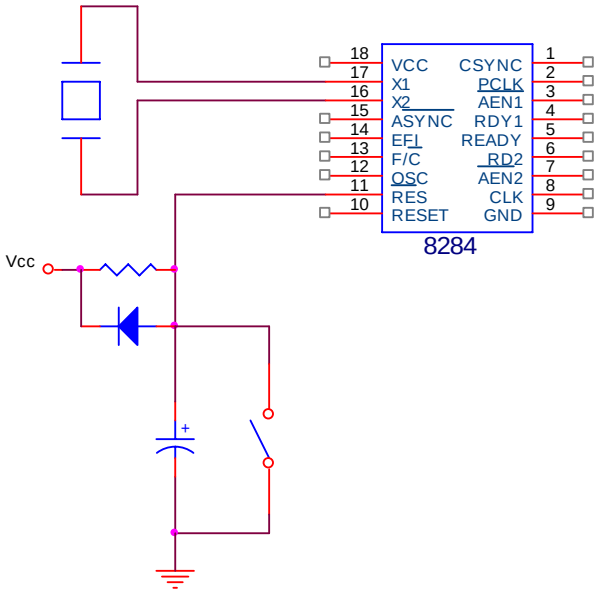


Hình 3.1 – Mạch tạo xung nhịp 8284

CSYNC (Clock Synchronisation): ngõ vào xung đồng bộ chung khi hệ thống có các 8284 dùng dao động ngoài tại chân EFI. Khi dùng mạch dao động trong thì phải nối đất.

PCLK (Peripheral Clock): xung nhịp f = f_X/6 (f_X là tần số thạch anh)

$\overline{AEN1}$, $\overline{AEN2}$ (Address Enable): cho phép chọn các chân RDY1, RDY2 báo hiệu trạng thái sẵn sàng của bộ nhớ hay thiết bị ngoại vi



Hình 3.2 – Mạch khởi động cho 8284

RDY1, RDY2 (Bus ready): tạo các chu kỳ đợi ở CPU

READY: nối đến chân READY của µP.

CLK (Clock): xung nhịp f = f_X/3, nối với chân CLK của µP.

RESET: nối với chân RESET của µP, là tín hiệu khởi động lại toàn hệ thống

$\overline{RES}$ (Reset Input): chân khởi động cho 8284

OSC: ngõ ra xung nhịp có tần số f_X

F/$\overline{C}$ (Frequency / Crystal): chọn nguồn tín hiệu chuẩn cho 8284, nếu ở mức cao thì chọn tần số xung nhịp bên ngoài, ngược lại thì dùng xung nhịp từ thạch anh

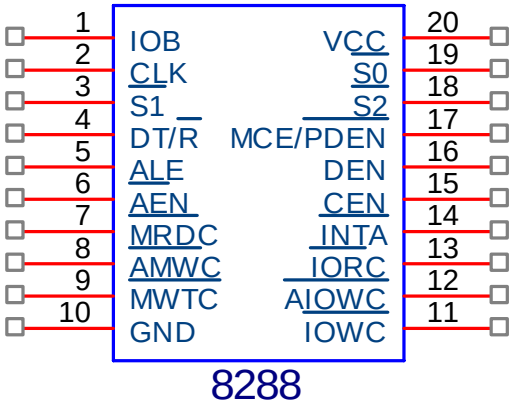
EFI (External Frequency Input): xung nhịp từ bộ dao động ngoài

$\overline{ASYNC}$: chọn chế độ làm việc cho tín hiệu RDY.

X1,X2: ngõ vào của thạch anh

1.2. Mạch điều khiển bus 8288

Mạch điều khiển bus 8288 lấy một số tín hiệu điều khiển của µP và cung cấp các tín hiệu điều khiển cần thiết cho hệ vi xử lý.



Hình 3.3 – Mạch điều khiển bus 8288

IOB (Input / Output Bus Mode): điều khiển để 8288 làm việc ở các chế độ bus khác nhau.

CLK (Clock): ngõ vào lấy từ xung nhịp hệ thống.

$\overline{S2}$, $\overline{S1}$, $\overline{S0}$: các tín hiệu trạng thái lấy trực tiếp từ µP. Tuỳ theo các giá trị nhận được mà 8288 sẽ đưa các tín hiệu theo bảng 3.1.

Bảng 3.1:

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	**Tạo tín hiệu**
0	0	0	$\overline{INTA}$
0	0	1	$\overline{IORC}$
0	1	0	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	Không
1	0	0	$\overline{MRDC}$
1	0	1	$\overline{MRDC}$
1	1	0	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	Không

DT/$\overline{R}$ (Data Transmit/Receive): μP truyền (1) hay nhận (0) dữ liệu.

ALE (Address Latch Enable): tín hiệu cho phép chốt địa chỉ

$\overline{AEN}$ (Address Enable): chờ thời gian trễ khoảng 150 ns sẽ tạo các tín hiệu điều khiển ở đầu ra của 8288 để đảm bảo rằng địa chỉ sử dụng đã hợp lệ.

$\overline{MRDC}$ (Memory Read Command): điều khiển đọc bộ nhớ

$\overline{MWTC}$ (Memory Write Command): điều khiển ghi bộ nhớ

$\overline{AMWC}$ (Advanced MWTC),: giống như $\overline{MWTC}$ nhưng hoạt động sớm hơn một chút dùng cho các bộ nhớ chậm đáp ứng kịp tốc độ μP.

$\overline{IOWC}$ (I/O Write Command): điều khiển ghi ngoại vi

$\overline{AIOWC}$ (Advanced IOWC),: giống như $\overline{IOWC}$ nhưng hoạt động sớm hơn một chút dùng cho các ngoại vi chậm đáp ứng kịp tốc độ μP.

$\overline{IORC}$ (I/O Read Command): điều khiển đọc ngoại vi

$\overline{INTA}$ (Interrupt Acknowledge): ngõ ra thông báo μP chấp nhận yêu cầu ngắt của thiết bị ngoại vi

CEN (Command Enable): cho phép đưa ra các tín hiệu của 8288.

DEN (Data Enable): tín hiệu điều khiển bus dữ liệu thành bus cục bộ hay bus hệ thống.

MCE / $\overline{PDEN}$ (Master Cascade Enable / Peripheral Data Enable): định chế độ làm việc cho mạch điều khiển ngắt PIC 8259.

2. Giao tiếp với thiết bị ngoại vi

2.1. Các kiểu giao tiếp nhập / xuất

2.1.1. 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/$\overline{M}$ và các lệnh trao đổi dữ liệu thích hợp.

Bộ nhớ: IO/$\overline{M}$ = 0, dùng lệnh MOV

Ngoại vi: IO/$\overline{M}$ = 1, dùng lệnh IN (nhập) hay OUT (xuất)

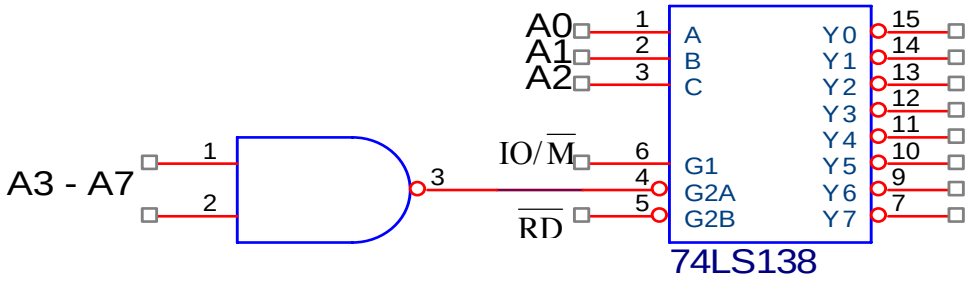
2.1.2. Thiết bị ngoại vi và bộ nhớ có chung không gian địa chỉ

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.

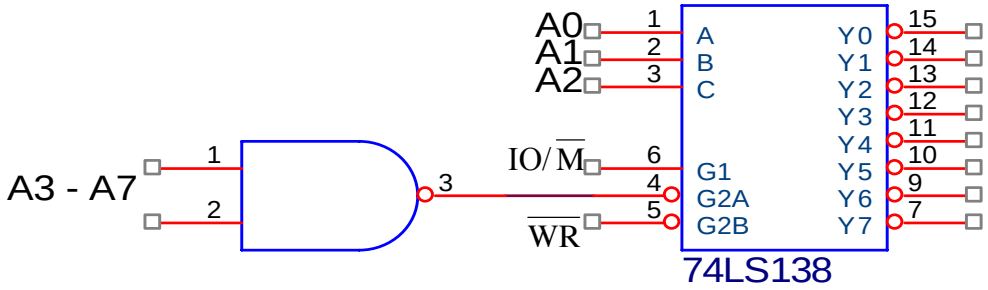
2.2. Giải mã địa chỉ cho thiết bị nhập / xuất

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 3.4 – Giải mã cho các cổng

2.3. 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.

2.4. Giao tiếp nhập / xuất song song lập trình được 8255A PPI (Programmable Peripheral Interface)

2.4.1. Giới thiệu

8255A là thiết bị xuất nhập song song lập trình được. Nó là một thiết bị I/O đa dụng có thể sử dụng với bất cứ µP nào, có thể lập trình để truyền dữ liệu, từ I/O thông thường đến I/O interrupt.

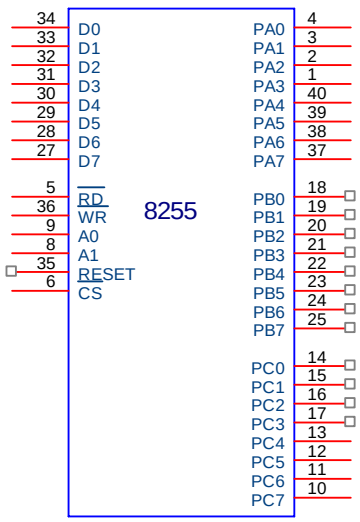
8255A có thể chia thành 3 Port: A, B và C; mỗi port 8 bit trong đó Port C có thể sử dụng như 8 bit riêng hay chia thành 2 nhóm, mỗi nhóm 4 bit: PCH (PC7 ÷ PC4) và PCL (PC3 ÷ PC0).

8255A có thể hoạt động ở 2 chế độ (mode): BSR (Bit Set/Reset) và I/O.

- **Chế độ BSR**: dùng để đặt hay xóa các bit của Port C.

- **Chế độ I/O**: gồm có 3 chế độ:

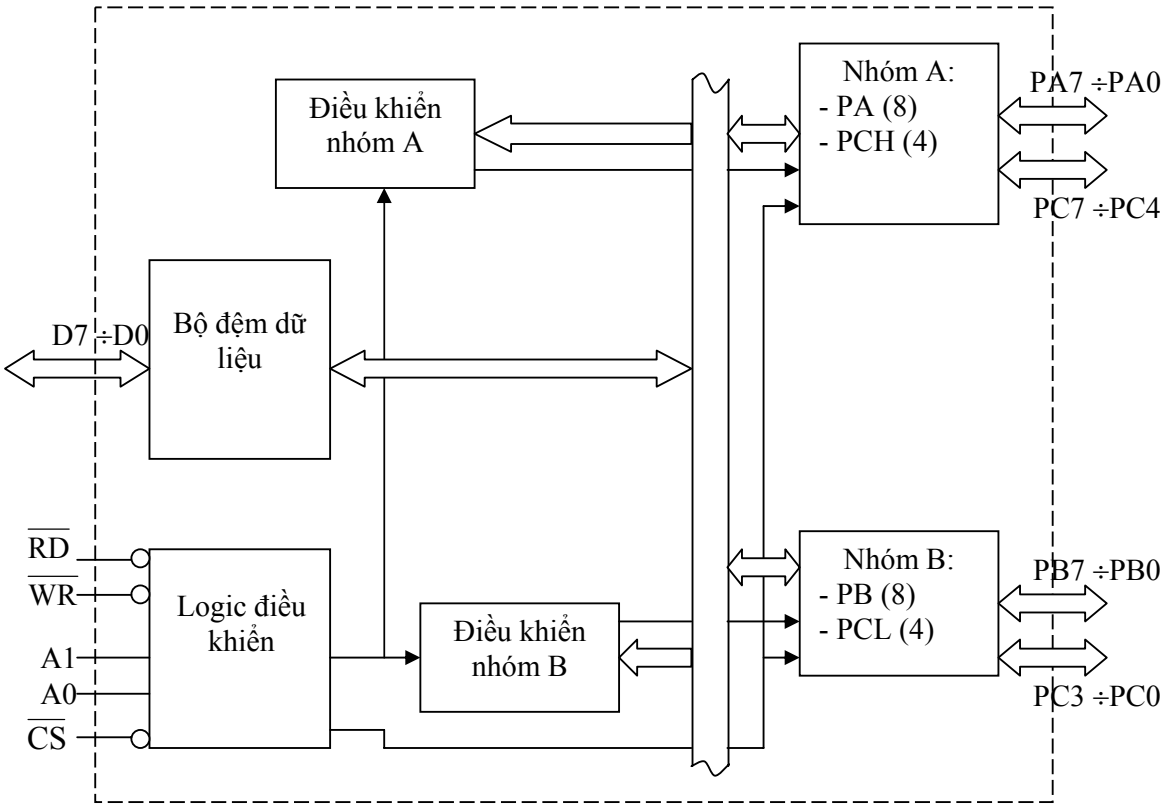
  - Chế độ 0: tất cả các Port làm việc như các Port I/O đơn giản.
  - Chế độ 1 (chế độ bắt tay: handshake): các Port A và B dùng các bit của Port C làm tín hiệu bắt tay. Trong chế độ này, các kiểu truyền dữ liệu I/O có thể được cài đặt, kiểm tra trạng thái và ngắt.
  - Chế độ 2: Port A có thể dùng để truyền dữ liệu song hướng dùng các tín hiệu bắt tay từ Port C còn Port B được thiết lập ở chế độ 0 hay 1.



D7 – D0: bus dữ liệu
PA7 – PA0: Port A
PB7 – PB0: Port B
PC7 – PC0: Port C
A1, A0: giải mã
RESET: ngõ vào Reset
$\overline{CS}$: Chip Select
$\overline{RD}$: Read
$\overline{WR}$: Write
VCC: +5V
GND: 0V

Hình 3.5 – Sơ đồ chân của 8255A

2.4.2. Sơ đồ khối



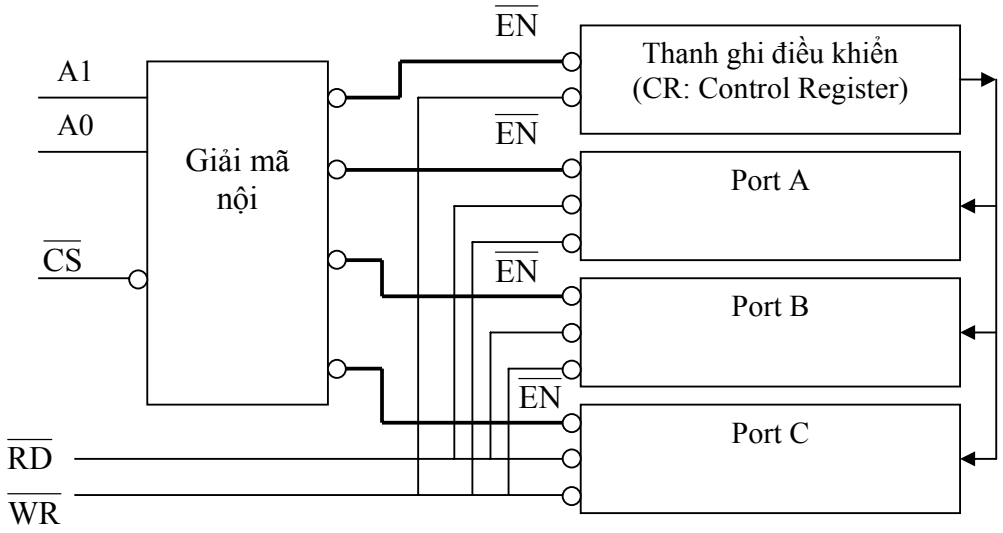
Hình 3.6 – Sơ đồ khối của 8255A

Logic điều khiển của 8255A gồm có 6 đường:

- $\overline{RD}$ (Read): cho phép ĐỌC. Khi chân này ở mức THẤP thì cho phép đọc dữ liệu từ Port I/O đã chọn.
- $\overline{WR}$ (Write): cho phép GHI. Khi chân này ở mức THẤP thì cho phép ghi dữ liệu ra Port I/O đã chọn.
- RESET: khi chân này ở mức cao thì sẽ xoá thanh ghi điều khiển và đặt các Port ở chế độ nhập.
- $\overline{CS}$ (Chip Select): chân chọn chip, thông thường $\overline{CS}$ được nối vào địa chỉ giải mã.
- A1, A0: giải mã xác định Port

Bảng 3.2:

$\overline{CS}$	A1	A0	Chọn
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Thanh ghi điều khiển
1	x	x	8255A không hoạt động

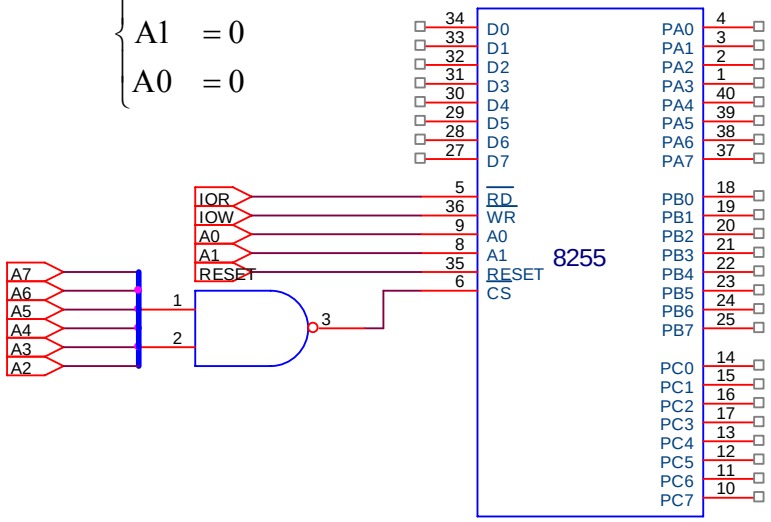


Hình 3.7 – Giải mã chọn các Port

Ví dụ: Xét sơ đồ kết nối 8255A như hình vẽ trang bên:

Theo bảng 3.2, để chọn Port A, ta phải có:

$$\begin{cases} \overline{CS} = 0 \\ A1 = 0 \\ A0 = 0 \end{cases}$$



Hình 3.8 – Logic chọn chip 8255A

Mà $\overline{CS}$ = 0 khi A7 = A6 = A5 = A4 = A3 = A2 = 1. Từ đó ta được địa chỉ Port I/O như sau:

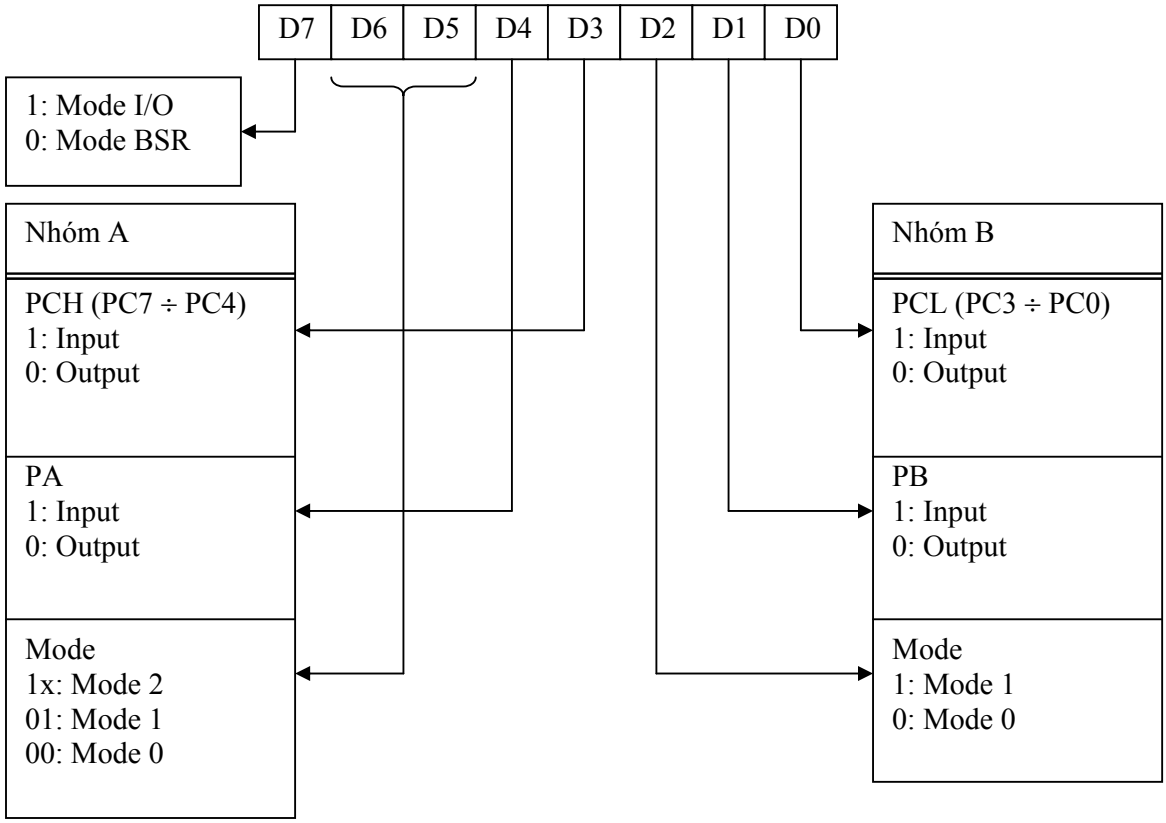
Bảng 3.3:

$\overline{CS}$						A1	A0	Port	Địa chỉ hex
A7	A6	A5	A4	A3	A2	A1	A0		
1	1	1	1	1	1	0	0	A	FCh
						0	1	B	FDh
						1	0	C	FEh
						1	1	CR	FFh

❖ **Thanh ghi điều khiển:**

Như đã biết, 8255A có 2 chế độ hoạt động và các Port của nó có thể có các chức năng I/O khác nhau. Để xác định chức năng của các Port, 8255A có một thanh ghi điều khiển (CR: Control Register). Nội dung của thanh ghi này gọi là từ điều khiển (CW: Control Word). Thanh ghi điều khiển sẽ được truy xuất khi A1 = A0 = 1. Chú ý rằng ta không thể thực hiện tác vụ Đọc đối với thanh ghi này.

Nếu bit D7 = 0, Port C làm việc ở chế độ BSR nhưng từ điều khiển BSR không ảnh hưởng đến chức năng các Port A, B.



Hình 3.9 – Dạng từ điều khiển cho 8255A ở chế độ I/O

2.4.3. Mode 0: Nhập / xuất đơn giản

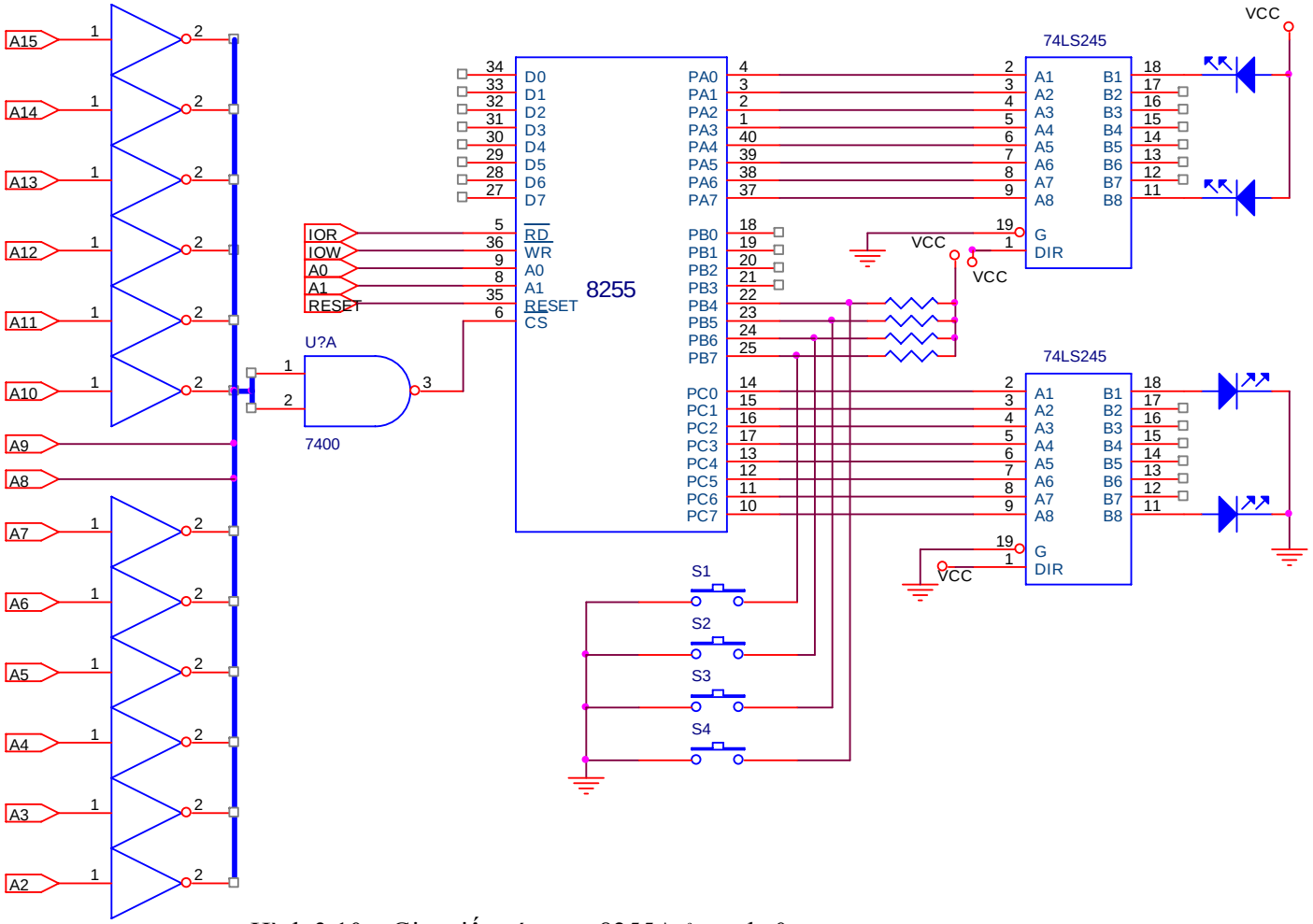
Trong chế độ này, mỗi port (hay nửa port của Port C) làm việc như các port nhập hay xuất với các tính chất sau:

- Các ngõ ra được chốt.
- Các ngõ vào không được chốt.
- Các port không có khả năng bắt tay và ngắt.

Để giao tiếp với ngoại vi thông qua 8255A cần phải:

- ***Xác định địa chỉ của các port A, B, C và CR thông qua các chân chọn chip $\overline{\text{CS}}$ và giải mã A1, A0.***
- ***Ghi từ điều khiển vào thanh ghi điều khiển.***
- ***Ghi các lệnh I/O để giao tiếp với ngoại vi qua các port A, B, C.***

Ví dụ: Xét sơ đồ kết nối 8255A như sau:



Hình 3.10 – Giao tiếp các port 8255A ở mode 0

- Xác định địa chỉ port:

Bảng 3.4:

$\overline{CS}$														A1	A0	Port	Địa chỉ hex
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0		
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	A	300h
														0	1	B	301h
														1	0	C	302h
														1	1	CR	303h

- Từ điều khiển:

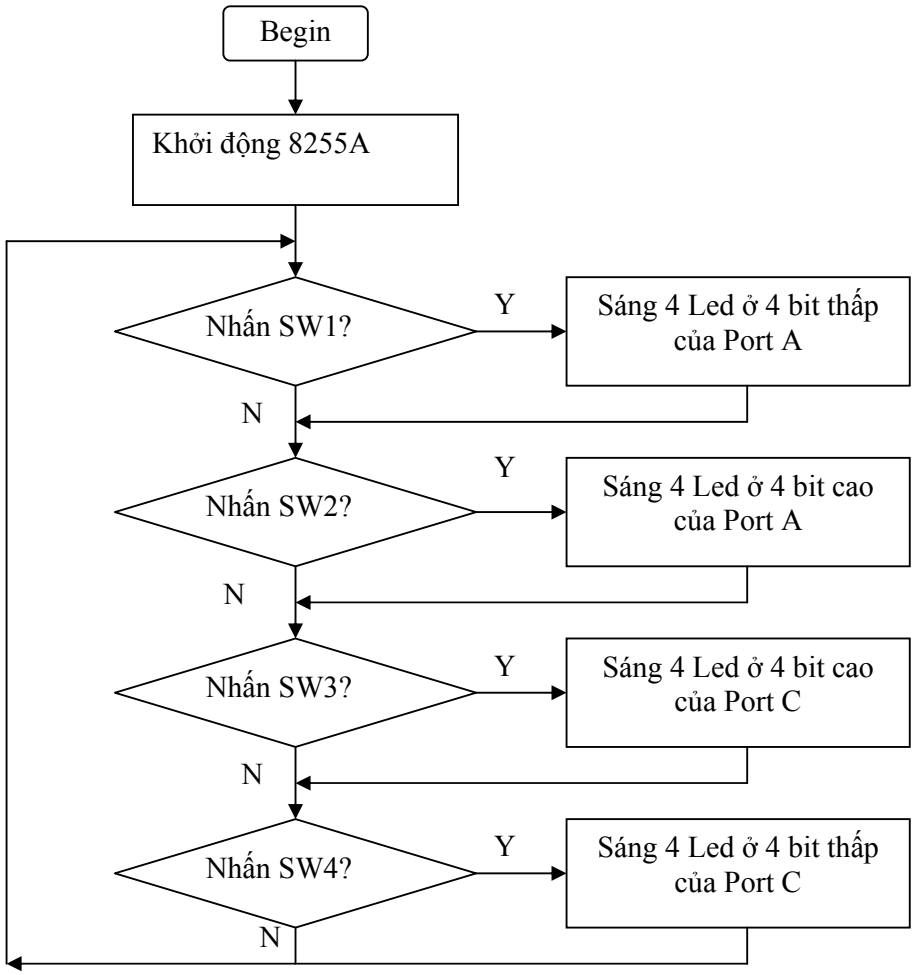
Bảng 3.5:

D7	D6	D5	D4	D3	D2	D1	D0	
1	0	0	0	0	0	1	0	= 82h
I/O mode	Nhóm A ở mode 0		PA: Output	PCH: Output	Nhóm B ở mode 0	PB: Input	PCL: Output	

Các Port của 8255A được khởi động bằng cách đặt từ điều khiển 82h vào thanh ghi điều khiển.

Trong sơ đồ kết nối này, 4 bit cao của Port B dùng làm Port nhập còn Port A và Port C làm Port xuất. Các tác vụ Đọc và Ghi được phân biệt bằng các tín hiệu điều khiển $\overline{IOR}$ và $\overline{IOW}$.

- Lưu đồ giải thuật:



- Chương trình:

```
        .MODEL    SMALL
        .STACK    100h
        .CODE
main    PROC
        ; ••nh c•u hình cho 8255
        MOV  AL,82h          ; T• •i•u khi•n (CW) là 82h
        MOV  DX,303h         ; ••a ch• thanh ghi
                             ; •i•u khi•n (CR)
        OUT  DX,AL           ; Ghi CW vào CR
cont:        MOV  DX,301h    ; ••a ch• Port B
        IN   AL,DX           ; ••c d• li•u t• Port B
                             ; (công t•c)
        AND  AL,0F0h         ; Che 4 bit th•p
        MOV  AH,AL
        CMP  AH,01110000b    ; Ki•m tra công t•c 1
        JNE  notSW1          ; N•u không nh•n
        MOV  AL,0Fh          ; N•u nh•n công t•c 1 thì
        MOV  DX,300h         ; xu•t ra Port A
        OUT  DX,AL           ; •• sáng 4 Led •
                             ; 4 bit th•p (Port A)

notSW1:  CMP  AH,10110000b   ; Ki•m tra công t•c 2
        JNE  notSW2          ; N•u không nh•n
```

```
          MOV  AL,0F0h          ; N•u nh•n công t•c 2 thì
          MOV  DX,300h          ; xu•t ra Port A •• sáng
          OUT  DX,AL            ; 4 Led • 4 bit cao (Port A)

notSW2:   CMP  AH,11010000b     ; Ki•m tra công t•c 3
          JNE  notSW3           ; N•u không nh•n
          MOV  AL,0Fh           ; N•u nh•n công t•c 3 thì
          MOV  DX,302h          ; xu•t ra Port C •• sáng 4
          OUT  DX,AL            ; Led • 4 bit cao (Port C)

notSW3:   CMP  AH,11100000b     ; Ki•m tra công t•c 4
          JNE  notSW4           ; N•u không nh•n
          MOV  AL,F0h           ; N•u nh•n công t•c 4 thì
          MOV  DX,302h          ; xu•t ra Port C •• sáng 4
          OUT  DX,AL            ; Led • 4 bit th•p (Port C)
notSW4:   JMP  cont
     main ENDP
     END  main
```

2.4.4. Mode BSR

Mode BSR chỉ liên quan đến 8 bit của Port C, có thể đặt hay xoá các bit bằng cách ghi một từ điều khiển thích hợp vào thanh ghi điều khiển. Một từ điều khiển với D7 = 0 gọi là từ điều khiển BSR, từ điều khiển này không làm thay đổi bất cứ từ điều khiển nào được truyền trước đó với D7 = 1, nghĩa là các hoạt động I/O của Port A và B không bị ảnh hưởng bởi từ điều khiển BSR.

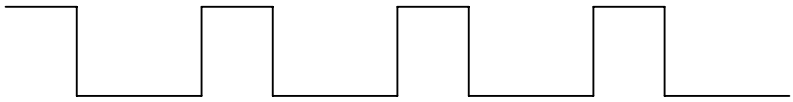
❖ **Từ điều khiển BSR:**

Từ điều khiển BSR khi được ghi vào thanh ghi điều khiển sẽ đặt hay xoá mỗi lần 1 bit.

D7	D6	D5	D4	D3	D2	D1	D0
0	x	x	X				S/R
Mode BSR	Không sử dụng			Chọn bit 000: PC0 001: PC1 010: PC2 011: PC3 100: PC4 101: PC5 110: PC6 111: PC7			0: Xoá (Reset) 1: Đặt (Set)

Ví dụ: Xét sơ đồ kết nối 8255A như hình 3.10. Giả sử ta cần tạo một sóng chữ nhật tại bit PC0.

Để tạo một sóng chữ nhật tại PC0, ta cần 2 mức logic là 0 và 1 tại PC0.



Bảng 3.6:

	D7	D6	D5	D4	D3	D2	D1	D0	
Đặt bit PC0 = 1	0	0	0	0	0	0	0	1	= 01h
Xoá bit PC0 = 0	0	0	0	0	0	0	0	0	= 00h

- Địa chỉ thanh ghi điều khiển (bảng 3.4): 303h

- Chương trình con:

```
bsr: MOV       AL,01h     ; T• •i•u khi•n BSR
     MOV       DX,303h    ; ••a ch• thanh ghi •i•u khi•n (CR)
     OUT       DX,AL      ; ••t PC0 = 1
     CALL      DELAY1     ; Ch•
     MOV       AL,00h     ; T• •i•u khi•n BSR
     OUT       DX,AL      ; Xóa PC0 = 0
     CALL      DELAY2     ; Ch•
     JMP       bsr
```

Khi sử dụng ở mode BSR, cần chú ý các điều sau:

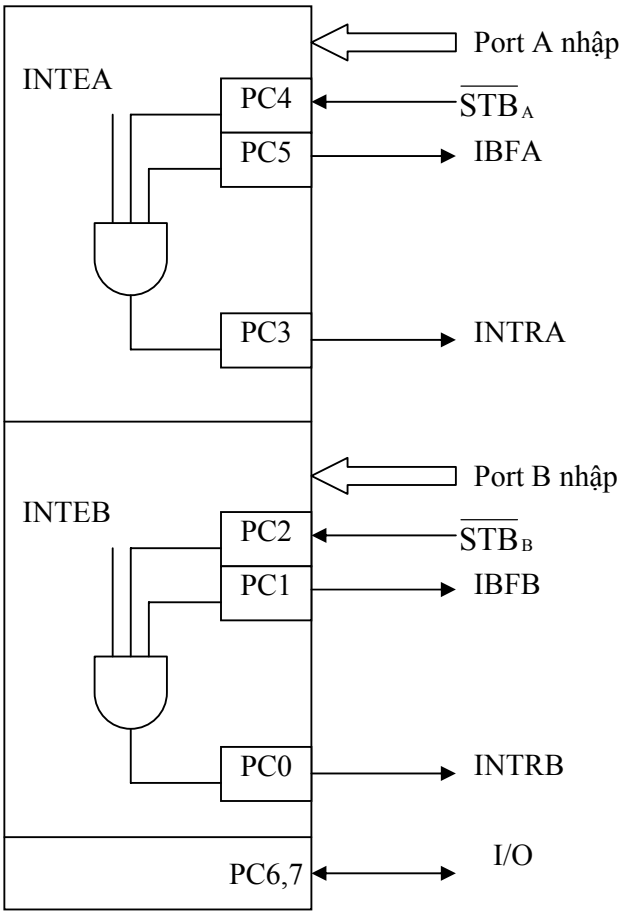
- ***Để đặt hay xoá các bit ở Port C, từ điều khiển được ghi vào thanh ghi điều khiển chứ không ghi vào Port C.***
- ***Một từ điều khiển BSR chỉ ảnh hưởng đến một bit của Port C.***
- ***Từ điều khiển BSR không ảnh hưởng đến I/O mode.***

2.4.5. Mode 1: Nhập / xuất với bắt tay (handshake)

Trong mode 1, các tín hiệu bắt tay được trao đổi giữa µP và thiết bị ngoại vi trước khi truyền dữ liệu. Các đặc tính ở chế độ này là:

- Hai Port A, B làm việc như các Port I/O 8 bit.
- Mỗi Port sử dụng 3 đường từ Port C làm các tín hiệu bắt tay. Hai đường còn lại có thể dùng cho các chức năng I/O đơn giản.
- Dữ liệu nhập / xuất được chốt.
- Hỗ trợ ngắt.

2.4.5.1. Các tín hiệu điều khiển nhập



Hình 3.11 – Cấu hình nhập của 8255A ở mode 1

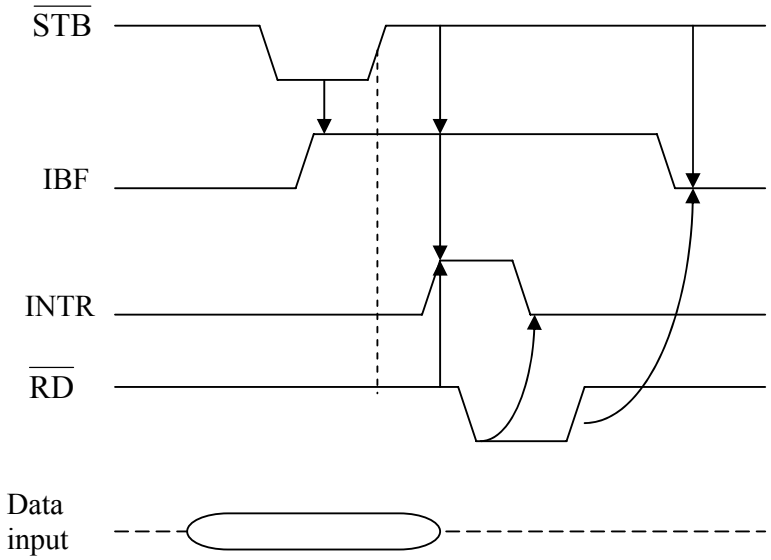
Theo hình vẽ, ta thấy Port A dùng 3 đường tín hiệu trên PC3, PC4 và PC5; Port B dùng 3 đường tín hiệu trên PC0, PC1 và PC2 làm các tín hiệu bắt tay. Các tín hiệu này có các chức năng sau khi các port A và B được đặt cấu hình là nhập:

- $\overline{STB}$ (Strobe Input): tích cực mức thấp, tín hiệu này được tạo bởi thiết bị ngoại vi để xác định rằng ngoại vi đã truyền 1 byte dữ liệu. Khi 8255A đáp ứng $\overline{STB}$, nó sẽ tạo ra IBF và INTR (hình 3.12).

- IBF (Input Buffer Full): tín hiệu này dùng để xác nhận 8255A đã nhận byte dữ liệu. Nó sẽ bị xoá khi μP đọc dữ liệu.

- INTR (Interrupt Request): Đây là tín hiệu xuất dùng để ngắt μP. Nó được tạo ra nếu $\overline{STB}$, IBF và INTE (flipflop bên trong) đều ở mức logic 1 và bị xoá bởi cạnh xuống của tín hiệu $\overline{RD}$ (Hình 3.12).

- INTE (Interrupt Enable): là một flipflop dùng để cho phép hay cấm quá trình tạo ra tín hiệu INTR. Hai flipflop INTEA và INTEB được đặt / xoá dùng BSR mode thông qua PC4 và PC2.



Hình 3.12 – Dạng sóng định thì cho ngõ vào có strobe

❖ **<u>Các từ điều khiển và trạng thái</u>:**

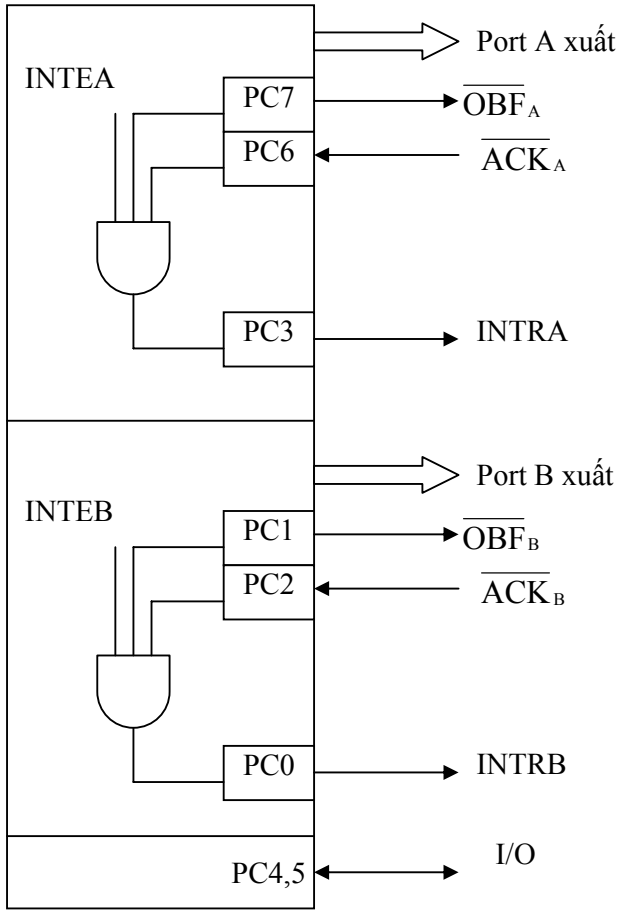
- Từ điều khiển: để xác định từ điều khiển, ta sử dụng hình 3.9

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	1/0	1	1	X
I/O mode	PA: Mode 1		PA: nhập	PC6,7 1: nhập 0: xuất	PB: Mode 1	PB: nhập	

- Từ trạng thái: sẽ được đặt trong thanh ghi tích luỹ nếu đọc Port C.

D7	D6	D5	D4	D3	D2	D1	D0
I/O	I/O	IBFA	INTEA	INTRA	INTEB	IBFB	INTRB

2.4.5.2. Các tín hiệu điều khiển xuất



Hình 3.13 – Cấu hình xuất của 8255A ở mode 1

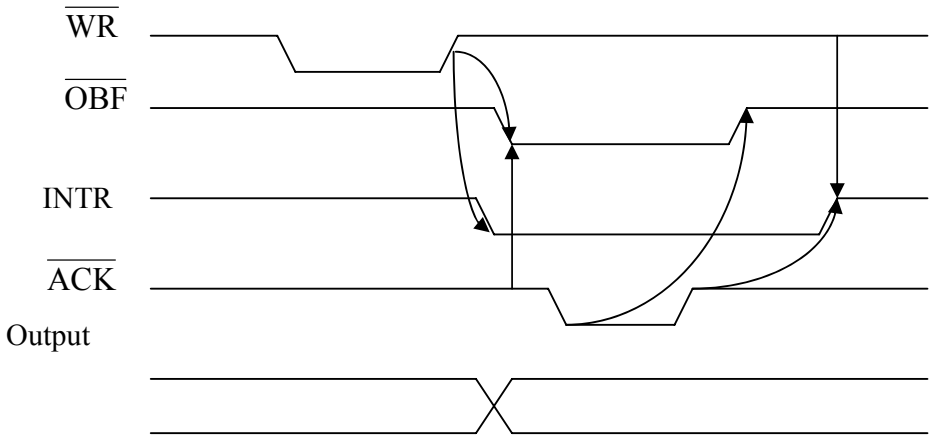
Chức năng các đường tín hiệu :

- $\overline{OBF}$ (Output Buffer Full): tín hiệu này sẽ xuống mức thấp khi μP ghi dữ liệu vào Port xuất của 8225A. Tín hiệu này đưa đến thiết bị ngoại vi để xác định dữ liệu sẵn sàng đưa vào ngoại vi (Hình 3.14). Nó sẽ lên mức cao khi 8255A nhận $\overline{ACK}$ từ ngoại vi.

- $\overline{ACK}$ (Acknowledge): đây là tín hiệu nhập từ ngoại vi (tích cực mức thấp) xác nhận dữ liệu đã nhập vào ngoại vi.

- INTR (Interrupt Request): đây là tín hiệu xuất, đặt bằng cạnh lên của tín hiệu $\overline{ACK}$. Tín hiệu này có thể dùng để ngắt μP yêu cầu byte dữ liệu kế tiếp để xuất. INTR được đặt khi $\overline{OBF}$, $\overline{ACK}$ và INTE ở mức logic 1 (Hình 4.14) và được xoá bởi cạnh xuống của tín hiệu $\overline{WR}$

- INTE (Interrupt Enable): đây là flipflop nội dùng để tạo tín hiệu INTR. Hai flipflop INTEA và INTEB điều khiển bằng các bit PC6 và PC2 thông qua BSR mode.



Hình 3.14 – Dạng sóng cho xuất strobe (có lấy mẫu) (với bắt tay)

❖ **Từ điều khiển và trạng thái:**

- Từ điều khiển:

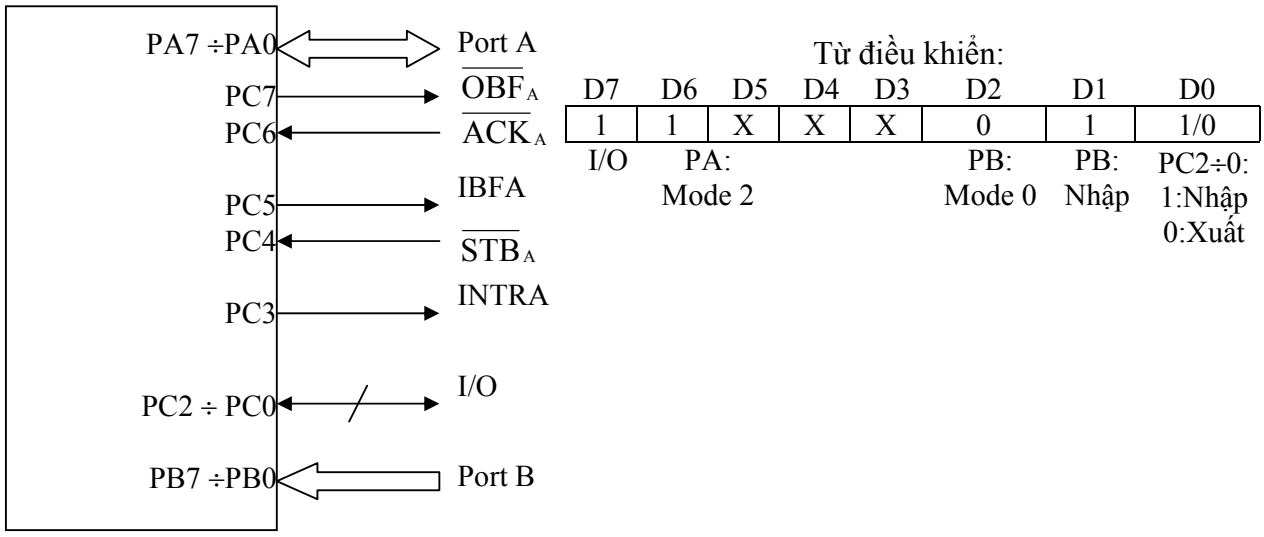
D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	0	1/0	1	0	X
I/O mode	PA: Mode 1		PA: xuất	PC4,5 1: nhập 0: xuất	PB: mode 1	PB: xuất	

- Từ trạng thái:

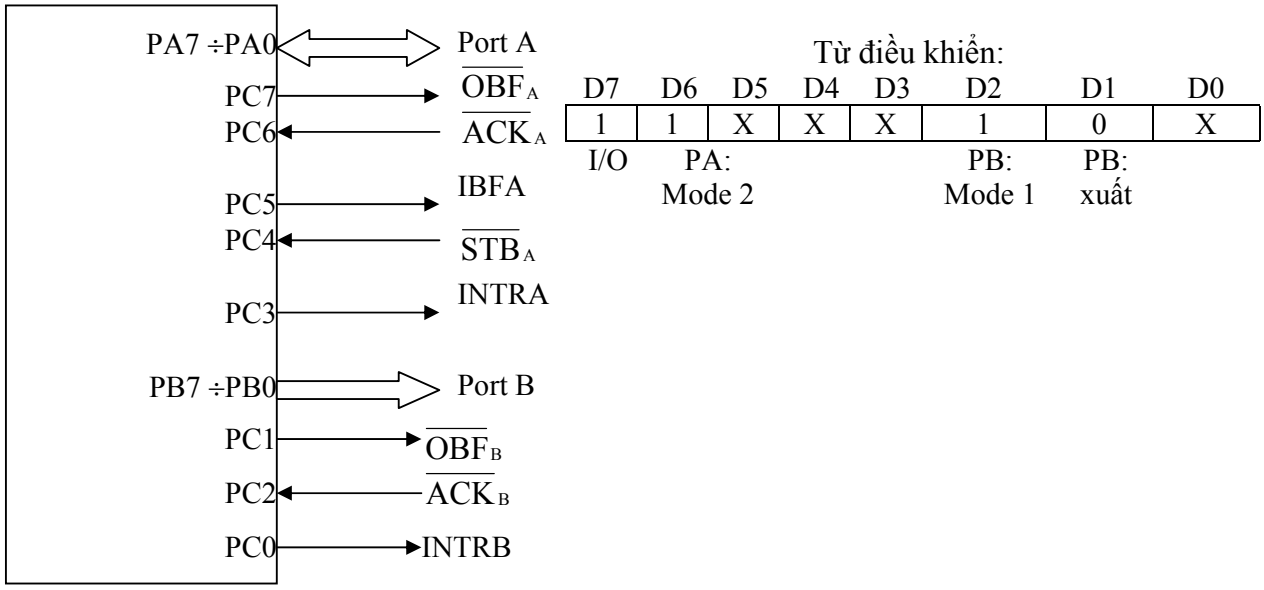
D7	D6	D5	D4	D3	D2	D1	D0
$\overline{OBF}_A$	INTEA	I/O	I/O	INTRA	INTEB	$\overline{OBF}_B$	INTRB

2.4.6. Mode 2: Truyền dữ liệu song hướng

Mode nay dùng chủ yếu trong các ứng dụng như truyền dữ liệu giữa hai máy tính hay giao tiếp bộ điều khiển đĩa mềm. Trong mode này, Port A dùng làm Port song hướng và Port B làm việc ở Mode 0 hay 1. Port A sử dụng 5 tín hiệu tại Port C làm các tín hiệu điều khiển để truyền dữ liệu. Ba tín hiệu còn lại của Port C được dùng làm I/O đơn giản hay bắt tay cho Port B.



(a) 8255A ở mode 2 và mode 0 (nhập)



(a) 8255A ở mode 2 và mode 1 (xuất)

Hình 3.15 – 8255A dùng ở Mode 2

2.4.7. Các ví dụ minh họa

2.4.7.1. Giao tiếp với bộ chuyển đổi A/D ADC0804 dùng 8255A ở Mode 0 và Mode BSR

Ta thiết lập 8255A hoạt động như sau:

- Dùng Port A để đọc dữ liệu.

- Dùng PC0, PC3 điều khiển các chân $\overline{RD}$, $\overline{WR}$ của ADC0804.

Xét sơ đồ mạch có logic chọn chip giống như hình 4.10. Tầm địa chỉ Port từ 300h ÷ 303h.

- **Từ điều khiển mode 0:**

Port A: nhập

Pot B: không sử dụng

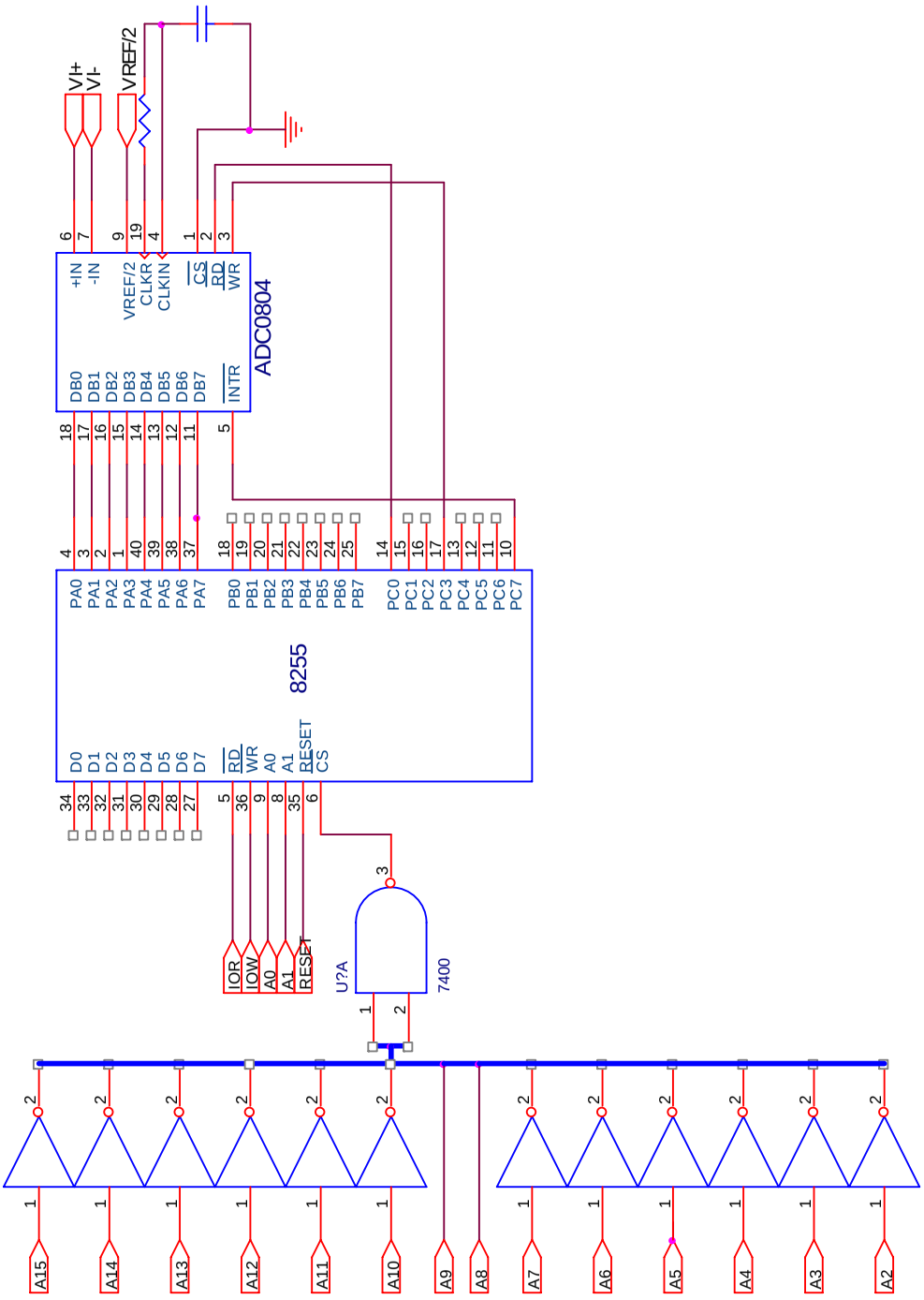
Port Clow: port xuất dùng để điều khiển 2 ngõ $\overline{RD}$, $\overline{WR}$ của ADC0804

Port Chigh: port nhập dùng để đọc trạng thái ở chân $\overline{INTR}$ của ADC0804

D7	D6	D5	D4	D3	D2	D1	D0	
1	0	0	1	0	0	0	0	= 90h
I/O	PA: mode 0		PA: nhập	PCH: xuất	PB: không sử dụng		PCL: xuất	

- **Từ điều khiển BSR:**

	D7	D6	D5	D4	D3	D2	D1	D0	
Đặt PC0	0	0	0	0	0	0	0	1	= 01h
Xoá PC0	0	0	0	0	0	0	0	0	= 00h
Đặt PC3	0	0	0	0	0	1	1	1	= 07h
Xoá PC3	0	0	0	0	0	1	1	0	= 06h



Hình 3.16 – Giao tiếp bộ chuyển đổi A/D ADC0804 dùng 8255A

- Mô tả chương trình:

  - Khởi động 8255A bằng cách đặt từ điều khiển mode 0 vào thanh ghi điều khiển.
  - Cấp một xung vào chân $\overline{RD}$ của 8255A.
  - Đọc trạng thái của ADC0804 từ chân $\overline{INTR}$.
  - Nếu $\overline{INTR}$ = 0 thì cấp một xung vào chân $\overline{WR}$ của ADC0804 để xuất dữ liệu.
  - Đọc dữ liệu từ ADC0804 vào thông qua Port A.

- Đoạn chương trình thực hiện:

```
adc: MOV       DX,303h   ; ••a ch• thanh ghi •i•u khi•n (CR)
     MOV       AL,90h    ; T• •i•u khi•n (CW)
     OUT       DX,AL     ;Ghi CW vào CR
     MOV       AL,01h    ; T• •i•u khi•n BSR •• PC0 = 1 (RD =
1)
     OUT       DX,AL     ; Xu•t ra CR
     MOV       AL,07h    ; T• •i•u khi•n BSR •• PC3 = 1
     OUT       DX,AL     ; Xu•t ra CR
     MOV       AL,06h    ; T• •i•u khi•n BSR •• PC3 = 0, t•o
                         ; xung WR
     OUT       DX,AL     ; Xu•t ra CR
     CALL      DELAY     ; Ch• quá trình chuy•n ••i th•c hi•n
xong
     MOV       AL,07h    ; T• •i•u khi•n BSR •• PC3 = 1
     OUT       DX,AL     ; Xu•t ra CR
     MOV       DX,300h   ; ••a ch• Port A
     IN        AL,DX     ; ••c d• li•u •ã chuy•n ••i t• ADC0804
     MOV       AL,01h    ; T• •i•u khi•n BSR •• PC0 = 1
                         ; (RD = 1)
     OUT       DX,AL     ; Xu•t ra CR
     RET                 ; vào t• Port A c•a 8255A
```

2.4.7.2. Giao tiếp với máy in trong chế độ bắt tay (Mode 1)

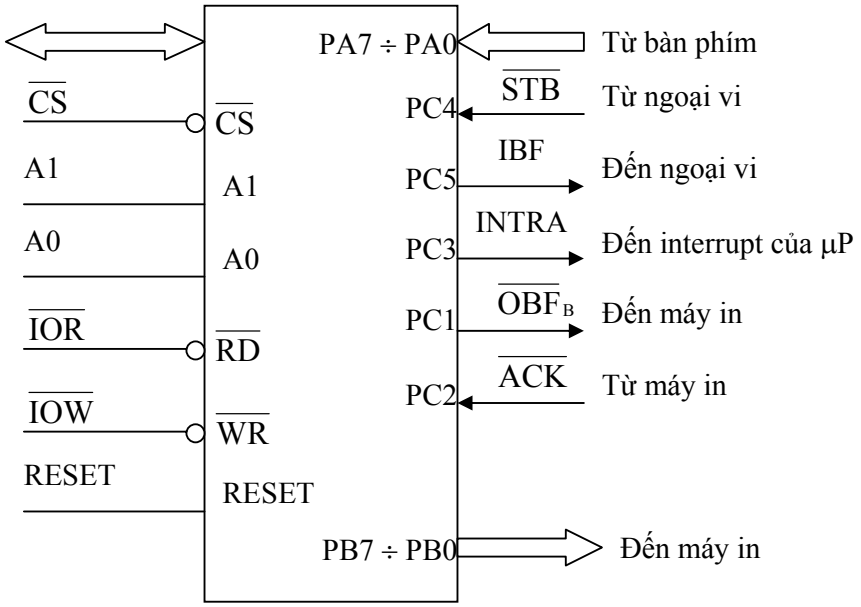
Xét mạch giao tiếp 8255A ở mode 1 với Port A được dùng làm Port nhập từ bàn phím với I/O interrupt và Port B được thiết kế làm Port xuất tới máy in với I/O kiểm tra trạng thái. Ta cần thực hiện các công việc sau:

- Xác định địa chỉ Port.
- Xác định từ điều khiển để Port A nhập và Port B xuất ở Mode 1.
- Xác định từ điều khiển BSR cho phép ngắt (INTEA).
- Xác định các byte mặt nạ để kiểm tra các đường $\overline{OBF_B}$ trong I/O kiểm tra trạng thái.
- Viết các lệnh khởi động và chương trình con in các ký tự chứa trong bộ nhớ.

Giả sử logic chọn chip như hình 3.10, địa chỉ Port cho trong bảng 3.4:

PA: FCh

PB: FDh
PC: FEh
CR: FFh



Hình 3.17 – Giao tiếp 8255A ở Mode 1

- **Từ điều khiển:** Port A nhập, Port B xuất ở Mode 1

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	0	1	0	0
I/O	PA: Mode 1		PA: nhập	Không sử dụng	PB: Mode 1	PB: xuất	Không sử dụng

= B4h

- **Từ điều khiển BSR**: dùng để đặt flipflop cho phép ngắt của Port A (INTEA), bit PC4 = 1

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	0	0	1
BSR mode	Không sử dụng			Bit PC4		Đặt bit (Set)	

= 09h

- **Từ trạng thái kiểm tra $\overline{OBF}_B$:**

D7	D6	D5	D4	D3	D2	D1	D0
X	x	x	x	x	x	$\overline{OBF}_B$	X

Byte mặt nạ: 0000 0010b

❖ **Khởi động:**

```
MOV     DX, 0FFh ; Kh•i ••ng 8255A
MOV     AL, 0B4h ; • Mode 1, Port A nh•p
OUT     DX, AL   ; Port B xu•t
```

```
     MOV       AL, 09h   ; ••t INTEA
     OUT       DX, AL    ; cho phép INTRA
     CALL      print
```

❖ **Chương trình con PRINT:**

```
print:
          LEA       DX,msg    ; Ch• ••n v• trí
                              ; ch•a các ký t•
          MOV       SI, DX
          ADD       SI,2
next:
          LODSB               ; L•y ký t• t• b• nh•
          CMP       AL,0      ; N•u không còn ký t• nào
          JNE       cont      ; thì k•t thúc
          JMP       exit
cont:
          MOV       AH,AL     ; L•u ký t• v•a ••c
          MOV       DX,0FEh
status:
          IN        AL,DX     ; ••c vào t• Port C
          AND       AL,02h    ; Ch• nh•n PC1
          JE        status    ; N•u máy in không
                              ; s•n sàng thì ch•
          MOV       AL,AH
          MOV       DX,0FDh   ; Xu•t ký t• •ã nh•n ra
          OUT       DX,AL     ; máy in (Port B)
          JMP       next      ; X• lý ký t• k• ti•p
exit:
          RET
```

❖ **Mô tả chương trình:**

- Ta sử dụng 8255A trong phần thiết kế này cho phép 2 hoạt động: xuất ra máy in và lấy dữ liệu vào từ bàn phím. Giao tiếp với máy in dùng kiểm tra trạng thái và giao tiếp bàn phím dùng ngắt.

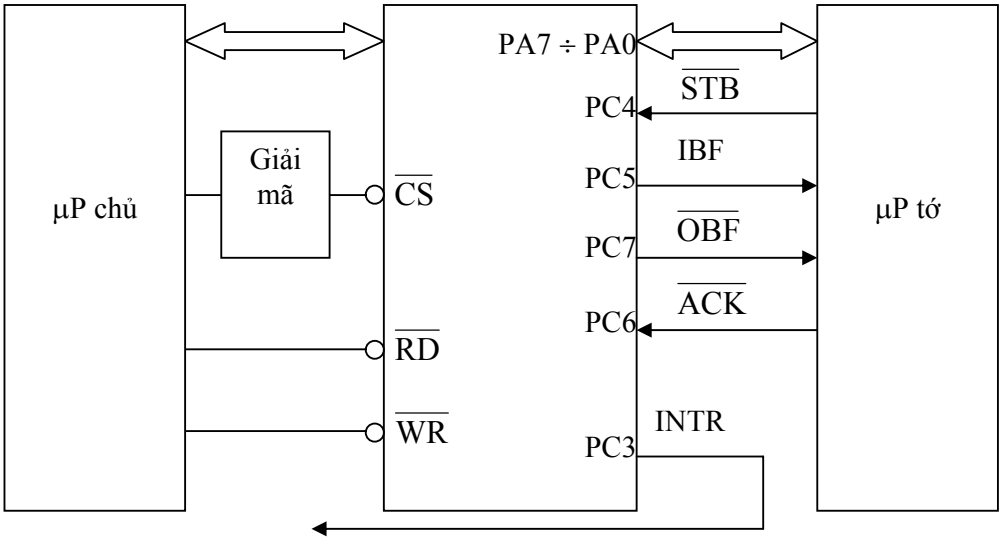
- Trong chương trình con PRINT, ký tự được đặt trong thanh ghi tích luỹ A và trạng thái đọc từ Port C. Ban đầu Port B trống, bit PC1 ($\overline{OBF_B}$) ở mức cao. Ta thực hiện lệnh OUT gởi dữ liệu ra Port B. Tín hiệu $\overline{OBF_B}$ sẽ xuống mức thấp do tác động cạnh lên của tín hiệu $\overline{WR}$, xác định rằng dữ liệu đã gởi ra máy in. Sau khi nhận byte dữ liệu, máy in gởi trở lại tín hiệu $\overline{ACK}$ xác định đã nhận. Tín hiệu $\overline{ACK}$ làm cho $\overline{OBF_B}$ ở mức cao xác định máy in sẵn sàng nhận ký tự kế tiếp và chương trình con PRINT tiếp tục thực hiện cho đến khi không còn ký tự nào trong vùng nhớ.

- Nếu một phím được nhấn khi chương trình con PRINT đang thực thi, byte dữ liệu truyền tới Port A và $\overline{STB_A}$ xuống mức thấp, đặt IBFA lên mức cao. Khi $\overline{STB_A}$ trở lại mức cao thì sẽ tạo ra INTRA. Tín hiệu này tạo ngắt đến μP và điều khiển được chuyển đến chương trình phục vụ

ngắt. Chương trình này sẽ đọc nội dung Port A, cho phép ngắt và quay về chương trình con PRINT.

2.4.7.3. Truyền dữ liệu giữa hai microprocessor trong xử lý phân bố dùng 8255A ở Mode 2

Ta thiết kế mạch giao tiếp để truyền dữ liệu hai chiều dạng chủ – tớ (master – slave) giữa hai µP.



Hình 3.18 – Thông tin 2 chiều giữa 2 µP dùng 8255A

Hình 3.18 chỉ sơ đồ khối thiết lập thông tin hay chiều giữa chủ và tớ . Sơ đồ khối chỉ hai data bas hai chiều – chủ và tớ – được nối với nhau thông qua 8225A, trong đó 8225A làm việc như thiết bị giao tiếp của µP chủ. Port A của 8225A được dùng để truyền dữ liệu hai chiều và 4 tín hiệu từ port C được dùng để bắt tay. Quá trình truyền dữ liệu tương tự như Mode 1 của 8225A. Khi µP chủ ghi 1 byte dữ liệu vào 8225A tín hiệu $\overline{OBF}$ xuống mức thấp để báo cho µP tớ biết là đã gởi dữ liệu vào, µP tớ sẽ báo nhận được khi nó đọc byte dữ liệu này. Tương tự, hai tín hiệu bắt tay khác được dùng khi µP tớ truyền 1 byte dữ liệu đến µP chủ.

µP chủ đòi hỏi các port I/O dùng để đọc và ghi dữ liệu và kiểm tra trạng thái của các tín hiệu bắt tay. Tương tự, µP tớ cần các port I/O để thực hiện Đọc và Ghi. Truyền dữ liệu có thể được thực hiện bằng cách kiểm tra trạng thái hay dùng ngắt. Tốc độ xử lý dữ liệu đối với µP chủ quan trọng hơn nên thường dùng µP chủ ở chế độ ngắt và µP tớ ở chế độ kiểm tra trạng thái. Ở ví dụ này, ta sẽ dùng cả 2 µP ở chế độ kiểm trạng thái.

Các hoạt động truyền dữ liệu giữa 2 I/O kiểm tra trạng thái có thể liệt kê như sau:

❖ **Truyền dữ liệu từ µP chủ đến µP tớ:**

1. µP chủ đọc trạng thái của $\overline{OBF}$ để kiểm tra xem µP tớ đã đọc dữ liệu chưa. Đây là chức năng nhập cho µP chủ.

2. µp chủ ghi dữ liệu vào Port A và 8225A báo cho µP tớ biết bằng cách đưa tín hiệu $\overline{OBF}$ xuống mức thấp. Đây là chức năng xuất của µP chủ.

3. µP tớ kiểm tra tín hiệu $\overline{OBF}$ (từ µP chủ) để xác định tính sẵn sàng của dữ liệu. Đây là chức năng nhập đối với µP tớ.

4. µP tớ đọc dữ liệu từ Port A và báo cho biết đã nhận được bằng cách đưa tín hiệu $\overline{ACK}$ xuống mức thấp . Đây là chức năng nhập đối với µP tớ.

❖ **Truyền dữ liệu từ µP tớ dến µp chủ:**

1. µP tớ kiểm tra tín hiệu bắt tay IBF để xem port A có sẵn sàng truyền dữ liệu hay không để truyền 1 byte . Đây là chức năng nhập đối với µP tớ .

2. µP đặt byte dữ liệu lên data bus và báo cho 8225A biết rằng sẵn sàng gởi dữ liệu bằng cách dùng tín hiệu $\overline{STB}$. Đây là chức năng xuất đối vói µP tớ.

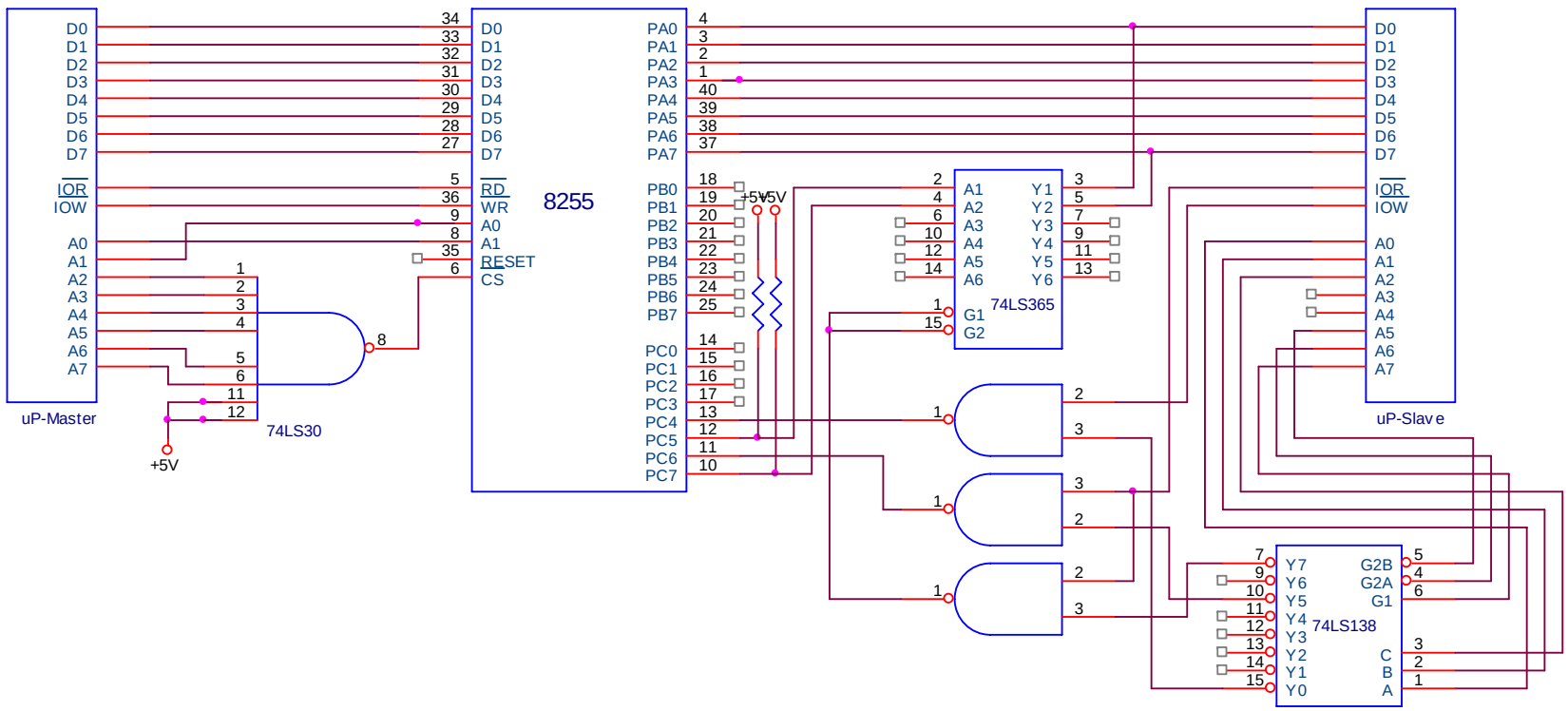
3. 8225A đưa IBF lên mức cao, µP chủ đọc tín hiệu này để xác định dữ liệu sẵn sàng chưa . Đây là chức năng nhập đối với µP chủ .

4. µP chủ đọc byte dữ liệu . Đây là chức năng nhập đối với µP chủ.

❖ **Kết nối phần cứng:**

Hình 3.19 cho thấy sơ đồ kết nối các port cần thiết và logic chọn chip cho 8255A. µP chủ thực hiện giải mã chọn 8255A dùng cổng NAND 8 ngõ vào nên 8255A được chọn khi tất cả các ngõ vào của cổng NAND đều ở mức 1. Từ đó, ta có các địa chỉ Port của 8255A đối với µP chủ là:

PA: FCh
PB: FDh
PC: FEh
CR: FFh



Hình 3.19 – Thông tin hai chiều giữa µP chủ và µP tớ

Port A được sử dụng ở Mode 2 dùng 4 tín hiệu từ Port C. µP chủ kiểm tra các tín hiệu $\overline{ACK}$ và $\overline{STB}$ bằng cách đọc các bit trạng thái $\overline{OBF}$ và IBF ở Port C.

Hai tín hiệu bắt tay khác - $\overline{OBF}$ và IBF – được nối tương ứng với các bit D7 và D0 của data bus của µP tớ thông qua bộ đệm 3 trạng thái 74LS365. Logic giải mã cho các đường tín hiệu tại Port C chính là bộ giải mã 3 sang 8 74LS138. Giả sử các đường logic không sử dụng (A3 và A4) ở mức 0, 8 đường ra của bộ giải mã sẽ cho phép vùng địa chỉ 80h ÷ 87h (Bảng 3.6). Hai đường ra của bộ giải mã được kết hợp với tín hiệu điều khiển $\overline{IOR}$ để tạo ra 2 xung chọn thiết bị nhận (85h và 87h). Xung chọn thiết bị nhập 87h được dùng để đọc trạng thái ở các đường dữ liệu D7 và D0. Đường giải mã có địa chỉ 80h được kết hợp với $\overline{IOW}$ để tạo tín hiệu $\overline{STB}$.

Bảng 3.7:

A7	A6	A5	A4	A3	A2	A1	A0	Chân giải mã	Địa chỉ hex
1	0	0	0	0	0	0	0	Y0	80h
					0	0	1	Y1	81h
					0	1	0	Y2	82h
					0	1	1	Y3	83h
					1	0	0	Y4	84h
					1	0	1	Y5	85h
					1	1	0	Y6	86h
					1	1	1	Y7	87h

❖ **Từ điều khiển mode 2:**

D7	D6	D5	D4	D3	D2	D1	D0	
1	1	0	0	0	0	0	0	= C0h
I/O	Mode 2			Không sử dụng				

❖ **Từ trạng thái mode 2:**

Trạng thái của hoạt động I/O ở Mode 2 có thể kiểm rta bằng cách đọc nội dung Port C.

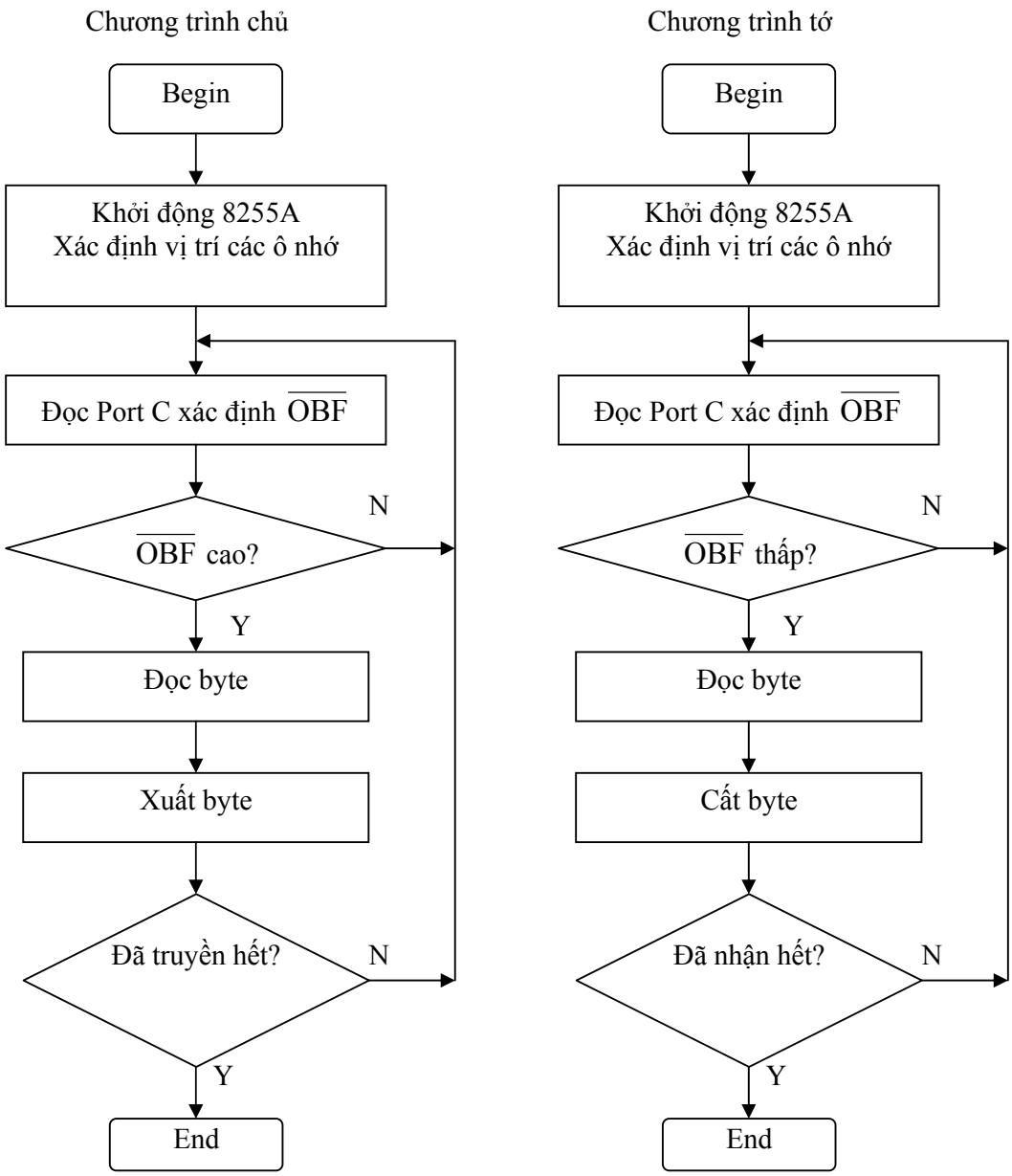
D7	D6	D5	D4	D3	D2	D1	D0
$\overline{OBF}_A$	$INTE_1$	IBFA	$INTE_2$	INTRA	X	X	X

Trạng thái của tín hiệu $\overline{OBF}$ được kiểm tra bằng cách đọc bit D7 và trang thái của IBF kiểm tra bằng bit D0.

❖ **Các tác vụ Đọc và Ghi của µP tớ:**

Một byte dữ liệu có thể được đọc bởi µP tớ từ Port A bằng cách gởi một xung chọn thiết bị tác động mức thấp đến tín hiệu $\overline{ACK}$, không cần xây dựng Port nhập. Tương tự, một byte dữ liệu có thể được ghi vào µP bằng cách đưa tín hiệu $\overline{STB}$ xuống thấp.

❖ **Lưu đồ giải thuật:**



❖ **Chương trình:**

➢ **Đoạn chương trình chủ: (Master program)**

```
        MOV     SP,stack1
        MOV     SI,master       ; ••a ch• các byte
                                ; c•n xu•t
        MOV     CX,byte_no      ; S• byte c•n xu•t
        MOV     AL,0C0h         ; T• •i•u khi•n
        MOV     DX,0FFh         ; ••a ch• thanh ghi
                                ; •i•u khi•n
        OUT     DX,AL
next:
        MOV     DX,0FEh         ; ••a ch• Port C
wait:
        IN      AL,DX           ; ••c vào t• Port C
        AND     AL,80h          ; Ki•m tra OBF
        JNE     wait            ; Ch• ••n khi OBF = 0
        LODSB                   ; ••c byte
        MOV     DX,0FCh         ; Xu•t byte v•a ••c
        OUT     DX,AL           ; ra Port A
        LOOP    next            ; N•u còn byte truy•n
                                ; thì ti•p t•c
```

➢ **Đoạn chương trình tớ: (Slave program)**

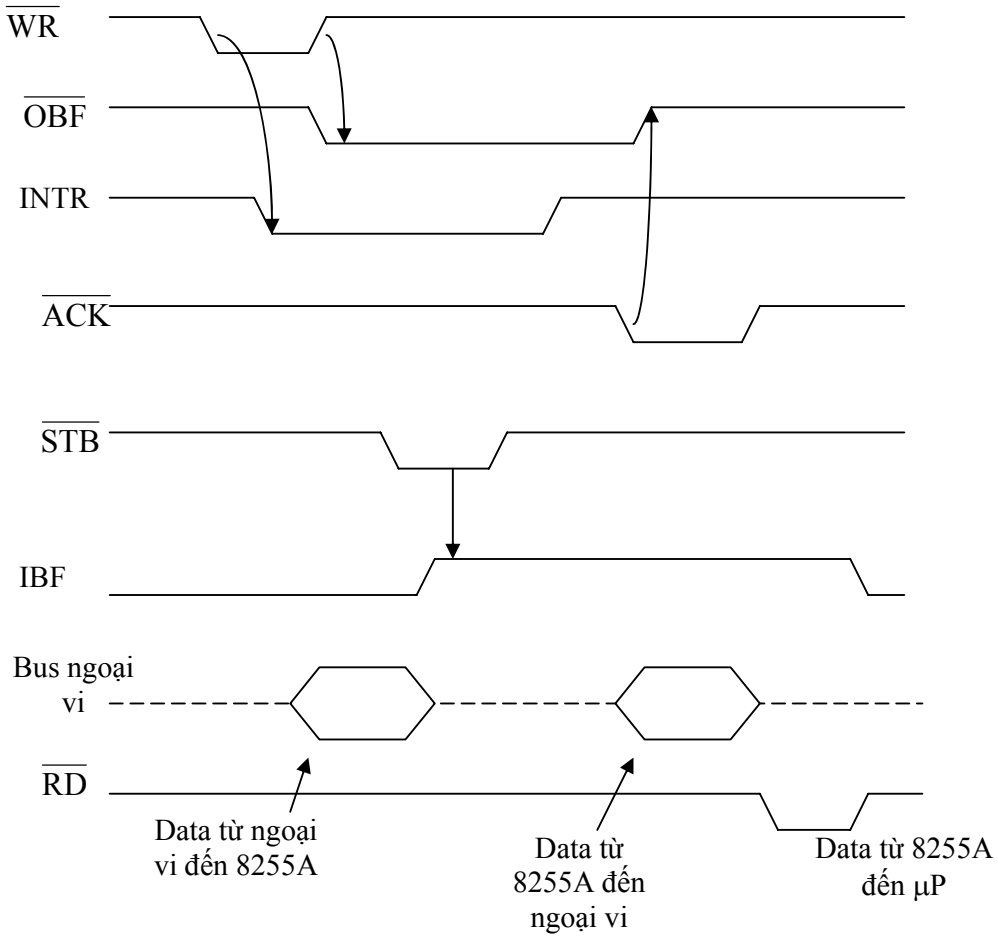
```
        MOV     ES,stack2
        MOV     DI,slave        ; ••a ch• các byte s• l•u
        MOV     CX,byte_no      ; S• byte c•n nh•n
next:
        MOV     DX,87h
wait:
        IN      AL,DX           ; ••c OBF
        AND     AL,80h          ; Ki•m tra OBF
        JE      wait            ; Ch• ••n khi OBF = 1
        MOV     DX,85h
        IN      AL,DX           ; ••c d• li•u
        STOSB                   ; C•t vào ô nh•
        LOOP    next            ; N•u còn byte truy•n
                                ; thì ti•p t•c
```

- Ta thấy rằng cả hai chương trình sẽ kiểm tra trạng thái $\overline{OBF}$. Chương trình chủ đợi cho đến khi $\overline{OBF}$ lên mức cao sẽ ghi một byte vào Port A. Ngược lại, chương trình tớ đợi cho đến khi $\overline{OBF}$ xuống mức thấp thì sẽ đọc dữ liệu.

- Khi μP chủ ghi một byte dữ liệu, nó sẽ chốt tại Port A và byte dữ liệu được đặt trên data bus của μP tớ khi $\overline{ACK}$ xuống mức thấp.

- Hai chương trình trên chỉ cho phép truyền một khối dữ liệu từ µP chủ đến µP tớ nhưng không thể truyền ngược lại. Để chuyển một khối dữ liệu từ µP tớ đến µP chủ, cần phải đọc tín hiệu IBF. µP chủ đợi cho đến khi IBF = 1 thì sẽ đọc một byte dữ liệu còn µP tớ đợi cho đến khi IBF = 0 thì ghi một byte dữ liệu.

- Giản đồ thời gian ở hình 3.20 cho thấy tín hiệu INTR dùng để truyền dữ liệu bằng ngắt. Trong ví dụ này, ta không sử dụng ngắt.



Hình 3.20 – Giản đồ thời gian ở Mode 2

BÀI TẬP CHƯƠNG 3

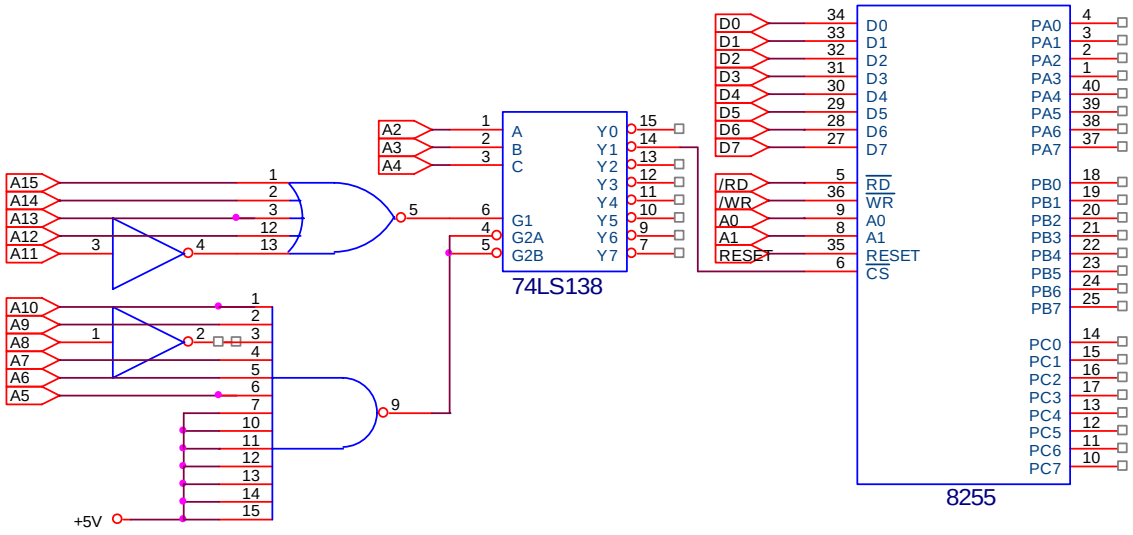
1. Xác định nội dung từ điều khiển của 8255 để:

a. Port A: nhập, Port B: xuất, PCH: nhập, PCL: xuất và hoạt động ở chế độ 0

b. Port A: xuất, Port B: nhập và hoạt động ở chế độ 1.

c. Nhóm A: chế độ 2, nhóm B: chế độ 1, Port B: nhập

2. Xác định địa chỉ của Port A, Port B, Port C và thanh ghi điều khiển.



3. Viết chương trình hợp ngữ thực hiện đọc nội dung tại Port B của 8255 và xuất nối tiếp giá trị vừa đọc ra PC2 theo thứ tự từ LSB → MSB.

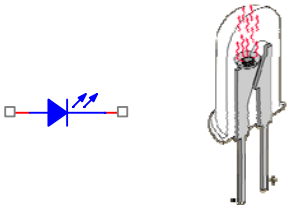
4. Viết chương trình hợp ngữ thực hiện đọc 10 giá trị từ Port A của 8255, sau đó xuất giá trị lớn nhất ra Port B.

CHƯƠNG 4: GIAO TIẾP VỚI CÁC THIẾT BỊ ĐƠN GIẢN

1. Giao tiếp LED (Light Emitting Diode)

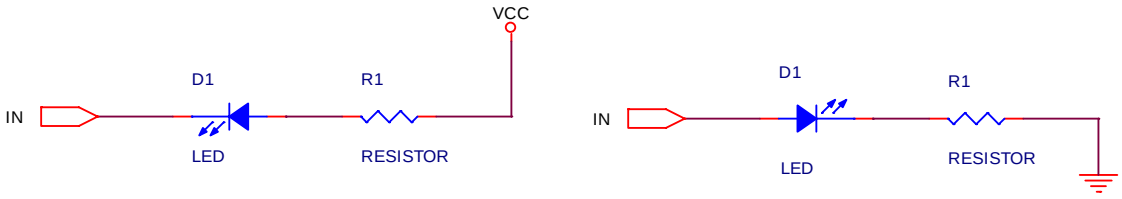
Để giúp cho người sử dụng có thể giao tiếp với máy móc thiết bị điều khiển, ta cần phải có một màn hình hiển thị số và chữ cái.Trong các hệ thống cần hiển thị một lượng lớn thông tin dữ liệu thường dùng CRT (màn hình) để hiển thị còn khi chỉ cần hiển thị một lượng nhỏ thông tin thì sẽ dùng các thiết bị hiển thị số đơn giản do giá rẻ và dễ điều khiển. Có nhiều loại màn hình hiển thị như CRT, LCD, LED, … Ta chỉ xét thiết bị hiển thị đơn giản là LED. Hiển thị số và chữ dùng LED có 3 loại chính.Với các ứng dụng hiển thị dùng để chỉ thị thì dùng LED đơn, để hiện số và chữ số thì dùng Led 7 đoạn hay Led 18 đoạn, để hiển thị ký tự bất kỳ thì dùng ma trận Led.

1.1. Giao tiếp LED đơn



Hình 4.1 - Mô tả LED và biểu diễn trong mạch

Khi LED sáng, dòng qua LED khoảng 10 – 40 mA và điện áp rơi trên LED vào khoảng 1.8V – 2V. Khi đó, ta có mạch điện điều khiển LED như sau:



Hình a Hình b

Hình 4.2 – Sơ đồ kết nối LED đơn

Giả sử mạch kết nối với 8255A có điện áp ứng với mức logic 0 từ 0 – 0.4V và mức logic 1 từ 4.6V – 5V. Chọn dòng qua LED là 20 mA và điện áp rơi trên LED là 2V.

Xét hình a: LED sáng khi mức logic tại chân IN là mức 0, ứng với điện áp 0.4V nên giá trị điện trở R1 là:

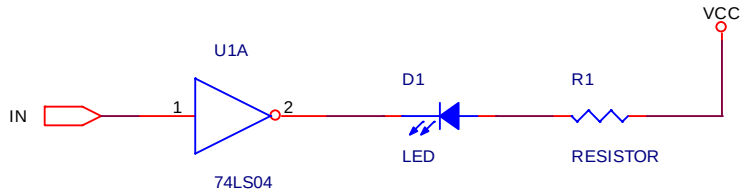
$$R1 = \frac{Vcc - V_{LED} - 0.4}{I_{LED}} = \frac{5 - 2 - 0.4}{20x10^{-3}} = 130\Omega \rightarrow \text{chọn } R1 = 150\ \Omega$$

Xét hình b: LED sáng khi mức logic tại chân IN là mức 1, ứng với điện áp 4.6V nên giá trị điện trở R1 là:

$$R1 = \frac{V_{IN} - V_{LED}}{I_{LED}} = \frac{4.6 - 2}{20x10^{-3}} = 130\Omega \rightarrow \text{chọn } R1 = 150\ \Omega$$

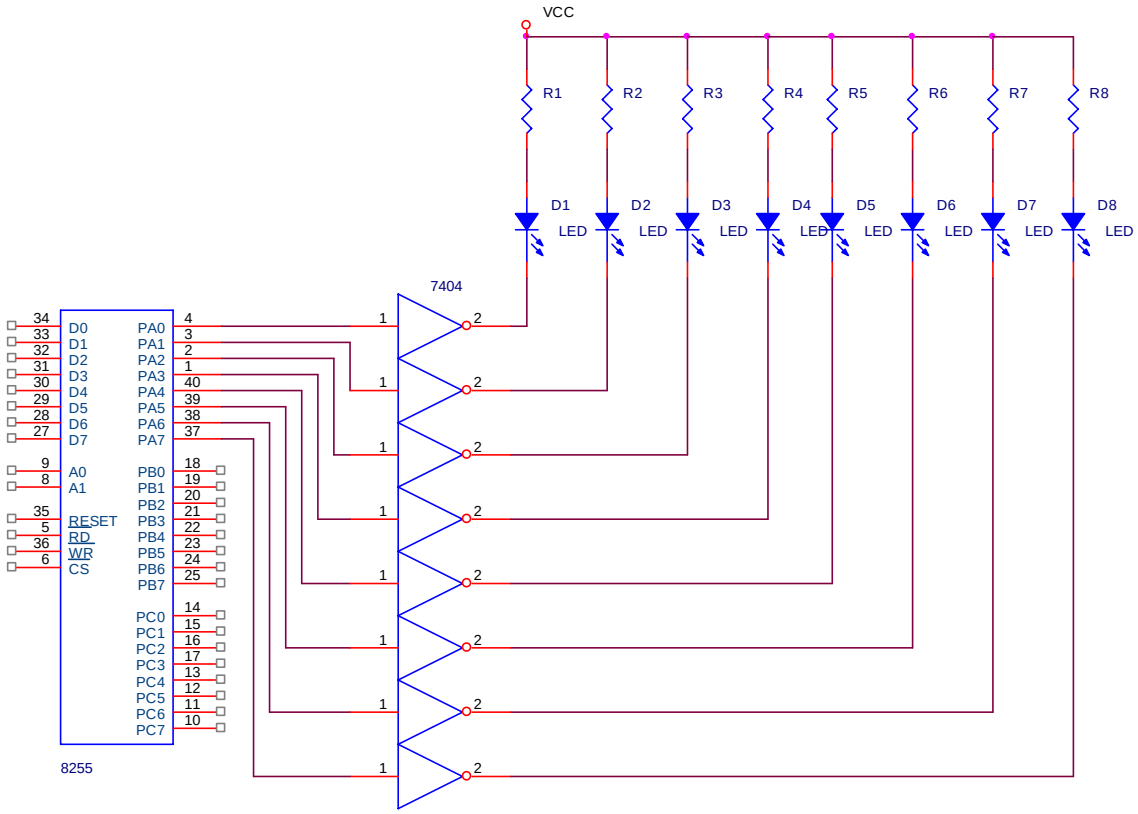
Tuy nhiên khi thiết kế mạch như hình b thì lưu ý rằng dòng tại chân IN phải đáp ứng được giá trị 20 mA. Đối với 8255A, dòng ngõ ra vào khoảng 2.5 mA nên không đáp ứng đủ dòng để sáng LED nên phải dùng thêm mạch khuếch đại.

Thông thường khi thiết kế giao tiếp với 8255A, ta sử dụng mạch hình a nhưng lưu ý là đối với cách thiết kế như trên thì dòng sẽ đi trực tiếp vào cổng vào/ra của 8255A nên ta có thể dùng thêm các cổng đệm hay đảo để tránh làm hư cổng.



Hình 4.3 – Sơ đồ kết nối LED đơn dùng cổng đảo

Xét sơ đồ kết nối giữa LED đơn và 8255 như sau:



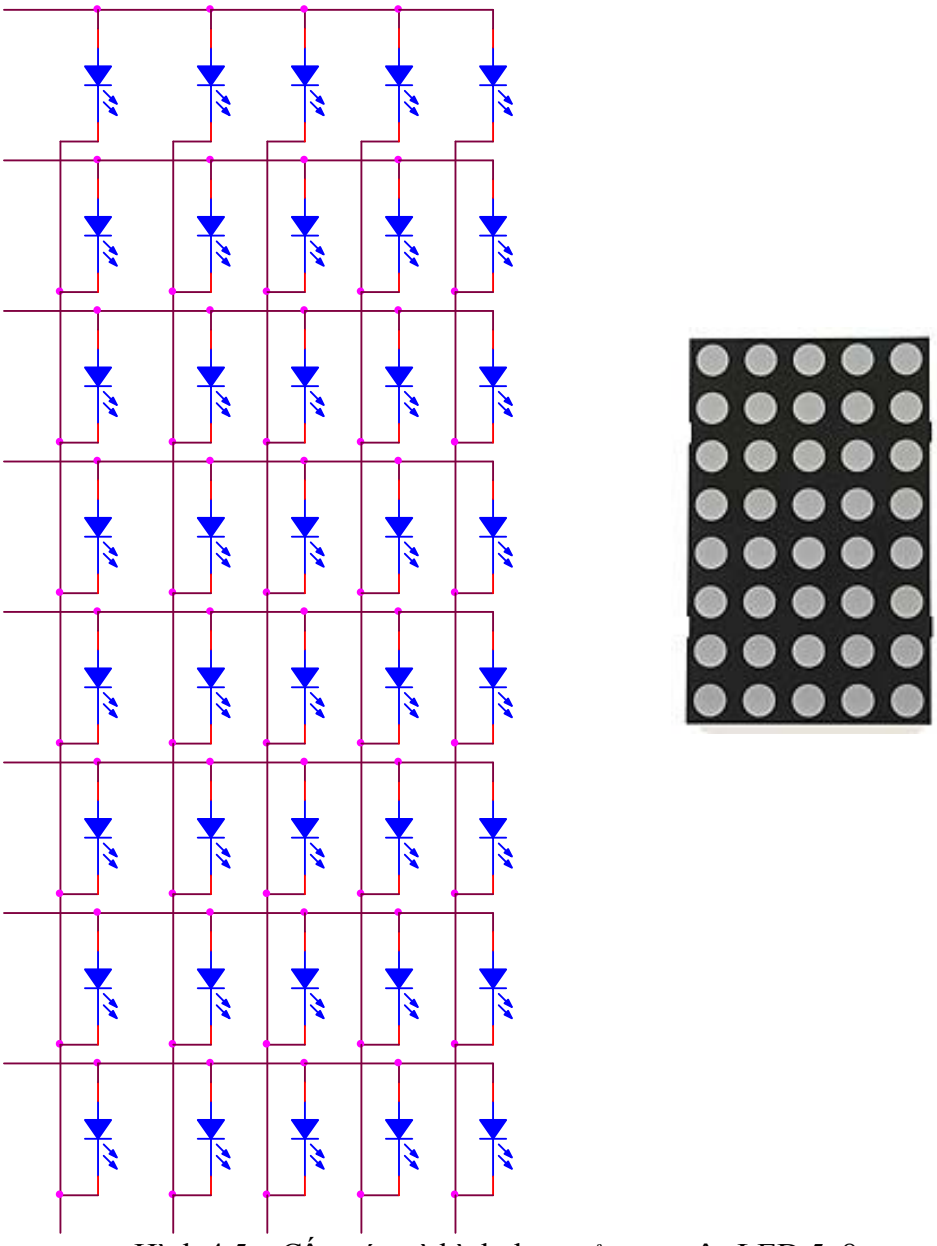
Hình 4.4 - Kết nối giữa LED đơn và 8255

Chương trình hợp ngữ quét LED từ trái sang phải như sau (giả sử địa chỉ Port A, Port B, Port C và CR của 8255 lần lượt là 300h, 301h, 302h và 303h):

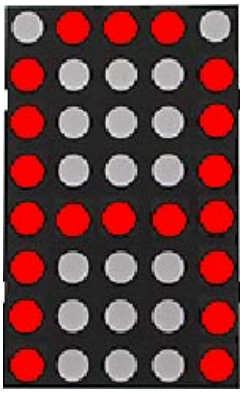
```
.MODEL SMALL
.STACK 100h
.DATA
     Led_data DB 01h,02h,04h,08h,10h,20h,40h,80h
.CODE
Main PROC
     MOV AX,@DATA
     MOV DS, AX          ; Gán địa chỉ cho Data segment
     MOV AL,80h          ; Định cấu hình cho 8255
     MOV DX,303h         ; Port A: xuất, Port B: xuất
     OUT DX,AL           ; Port C: xuất
     MOV BX,0
Lap:
     MOV AL,Led_data[BX]
     MOV DX,300h         ; Địa chỉ LED
     OUT DX,AL

     MOV CX,0FFh
Delay:                   ; Tạo thời gian trễ
     PUSH CX
     MOV CX,0FFFFh
     LOOP $
     POP CX
     LOOP delay

     INC BX
     CMP BX,8            ; LED có 8 trạng thái
     JNE lap

     MOV AH,4Ch          ; Kết thúc chương trình
     INT 21h
Main ENDP
END Main
```

1.2. Giao tiếp ma trận LED

Ma trận LED bao gồm nhiều LED cùng nằm trong một vỏ chia thành nhiều cột và hàng, mỗi giao điểm giữa hàng và cột có thể có 1 LED (ma trận LED một màu) hay nhiều LED (2 LED tại một vị trí tạo thành ma trận LED 3 màu). Để LED tại một vị trí nào đó sáng thì phải cấp điện áp dương tại Anode và điện áp âm tại Cathode (nghĩa là cấp mức logic 1 tại hàng và logic 0 tại cột ứng với ma trận LED có kết nối như hình 4.5). Trên cơ sở cấu trúc như hình vẽ 4.5, ta có thể mở rộng hàng và cột của ma trận LED để tạo thành các bảng quang báo.

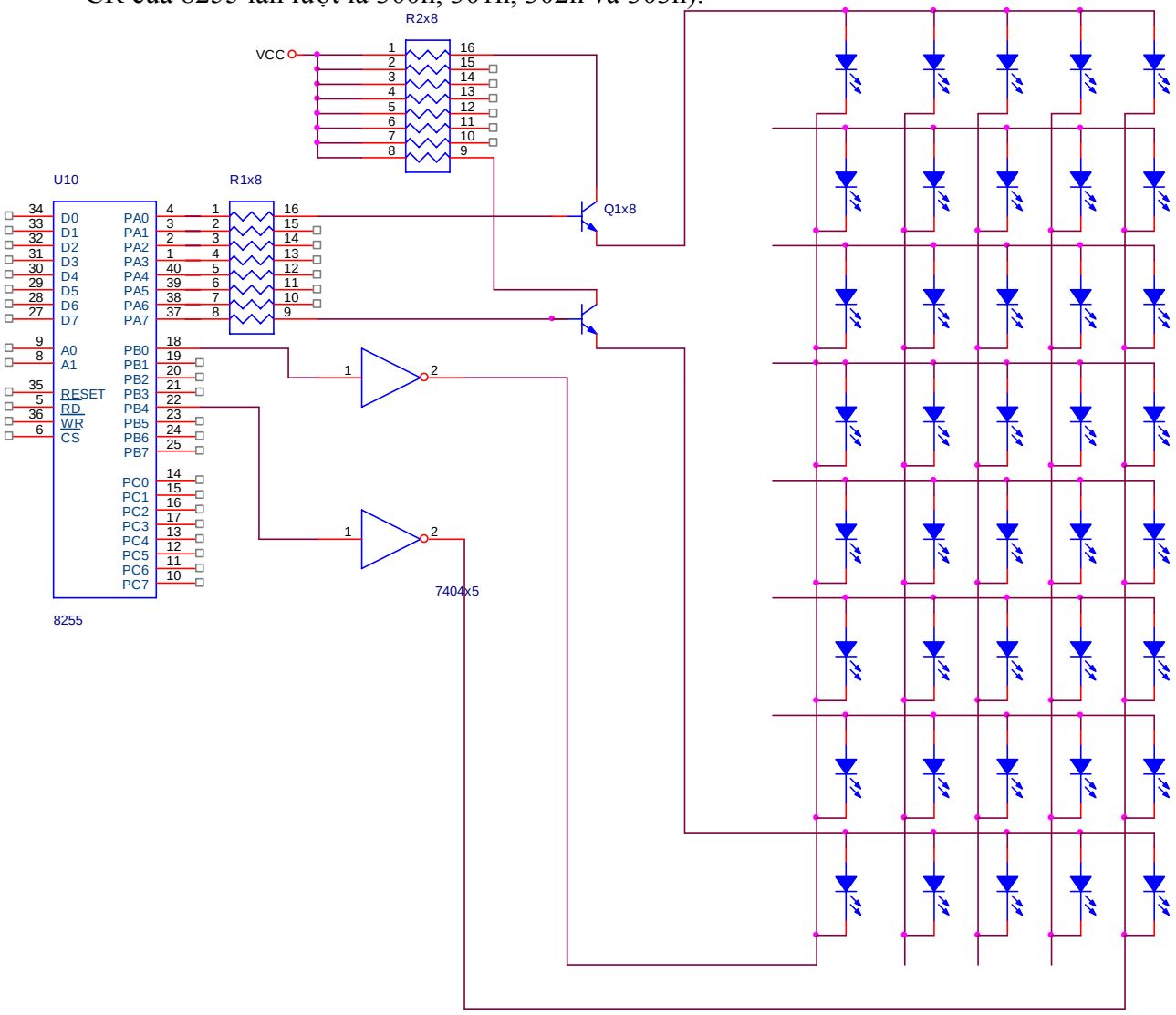


Hình 4.5 – Cấu trúc và hình dạng của ma trận LED 5x8



Hình 4.6 – Sáng chữ ‘A’ trên ma trận LED 5x8

Xét sơ đồ kết nối ma trận LED với 8255 như hình 4.7 trong đó Port A điều khiển hàng thông qua các transistor và Port B điều khiển cột (địa chỉ Port A, B, C và CR của 8255 lần lượt là 300h, 301h, 302h và 303h).



Hình 4.7 – Kết nối ma trận LED với 8255

Tính toán cho mạch:

Transistor Q1 hoạt động ở chế độ bão hoà với dòng I_c là dòng qua LED nên chọn Q1 có $\beta = 50$, $V_{CE} = 0{,}2V$, $V_{BE} = 0{,}7V$

$$R2 = \frac{Vcc - V_{LED} - V_{OL-7404} - V_{CE}}{I_{LED}} = \frac{5 - 2 - 0{,}4 - 0{,}2}{0{,}01} = 240\ \Omega, \text{ chọn } R2 = 220\ \Omega$$

$$R1 = \frac{V_{OH-8255} - V_{BE} - V_{LED} - V_{OL-7404}}{I_B} = \frac{4{,}6 - 0{,}7 - 2 - 0{,}4}{0{,}01/50} = 7.5\ K\Omega, \text{ chọn } R1 = 8.2\ K\Omega$$

Để hiển thị một ký tự trên ma trận LED, ta sẽ cho sáng các LED tương ứng. Ví dụ như để sáng chữ 'A' trên ma trận LED, ta cho tương ứng LED sáng như hình 4.6. Tuy nhiên, ta không thể cho sáng đồng thời 2 LED tại 2 vị trí hàng và cột khác nhau vì sẽ ảnh hưởng đến các LED còn lại. Ví dụ như khi sáng LED tại hàng 1, cột 2 (PA0 = 1, PB1 = 1) và hàng 2, cột 1 (PA1 = 1, PB0 = 1) thì do PA0 = 1, PB0 = 1; PA1 = 1, PB1 = 1 nên LED tại hàng 1, cột 1 và hàng 2, cột 2 cũng sáng.

Như vậy để sáng ký tự 'A' trên ma trận LED thì phải dùng phương pháp quét khi hiển thị dữ liệu trên ma trận LED. Quá trình quét có thể thực hiện quét dòng hay cột. Khi thực hiện quét cột, tại mỗi thời điểm chỉ có một cột sáng. Dữ liệu cho Port B và Port B của 8255A như sau:

- Cột 1: sáng 7 LED

```
PB0 = 1, PB1 – 4 = 0: PB = xxx0 0001b (01h)
PA0 = 0, PA1 – 7 = 1: PA = 1111 1110b (0FEh)
```

- Cột 2: sáng 2 LED

```
PB0 = 0, PB2 – 4 = 0, PB1 = 1: PB = xxx0 0010b (02h)
PA0 = 1, PA4 = 1, PA1 – 3 = 0, PA5 – 7 = 0:
     PA = 0001 0001b (11h)
```

- Cột 3: giống cột 2

```
PB0 - 1 = 0, PB3 – 4 = 0, PB2 = 1:
     PB = xxx0 0100b (04h)
PA0 = 1, PA4 = 1, PA1 – 3 = 0, PA5 – 7 = 0:
     PA = 0001 0001b (11h)
```

- Cột 4: giống cột 2

```
PB0 - 2 = 0, PB4 = 0, PB3 = 1: PB = xxx0 1000b (08h)
PA0 = 1, PA4 = 1, PA1 – 3 = 0, PA5 – 7 = 0:
     PA = 0001 0001b (11h)
```

- Cột 5: giống cột 1

```
PB0 - 3 = 0, PB4 = 1: PB = xxx1 0000b (10h)
PA0 = 0, PA1 – 7 = 1: PA = 1111 1110b (0FEh)
```

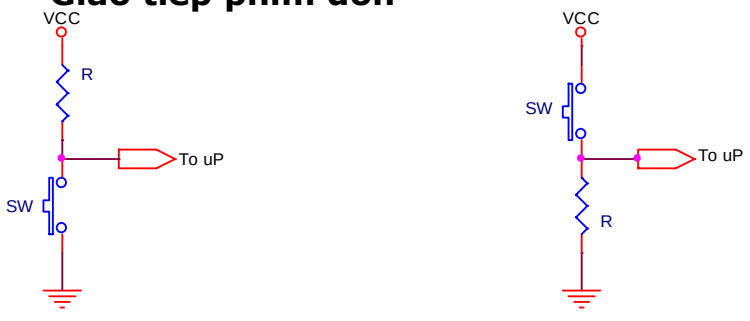
Chương trình hiển thị cho ma trận LED như sau:

```
.MODEL SMALL
.STACK 100h
.DATA
     pa DB 0FEh,11h,11h,11h,0FEh
     pb DB 01h,02h,04h,08h,10h
.CODE
Main PROC
     MOV AX,@DATA
     MOV DS, AX          ; Gán địa chỉ cho Data segment

     MOV AL,80h          ; Định cấu hình cho 8255
```

```
        MOV DX,303h         ; Port A: xuất, Port B: xuất
        OUT DX,AL           ; Port C: xuất
    Start:
        MOV AH,0Bh          ; Kiểm tra phím nhấn
        INT 21h
        CMP AL,0            ; Nếu có phím nhấn
        JNE Exit            ; thì kết thúc chương trình

        MOV BX,0
    Lap:
        MOV AL,pa[BX]
        MOV DX,300h         ; Xuất ra hàng
        OUT DX,AL

        MOV AL,pb[BX]
        MOV DX,301h         ; Xuất ra cột
        OUT DX,AL

        PUSH CX             ; Tạo thời gian trễ
        MOV CX,0FFFFh
        LOOP $
        POP CX

        INC BX
        CMP BX,5            ; Quét 5 cột
        JNE lap
        JMP Start

    Exit:
        MOV AH,4Ch          ; Kết thúc chương trình
        INT 21h
    Main ENDP

    END Main
```

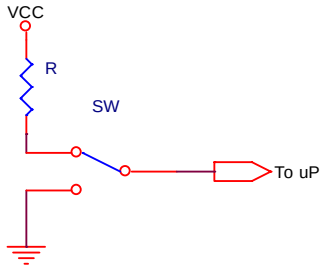
2. Giao tiếp bàn phím

2.1. Giao tiếp phím đơn



a/ b/

Hình 4.8 – Sơ đồ kết nối phím nhấn (2 chân) với vi xử lý



Hình 4.9 – Sơ đồ kết nối phím nhấn (3 chân) với vi xử lý

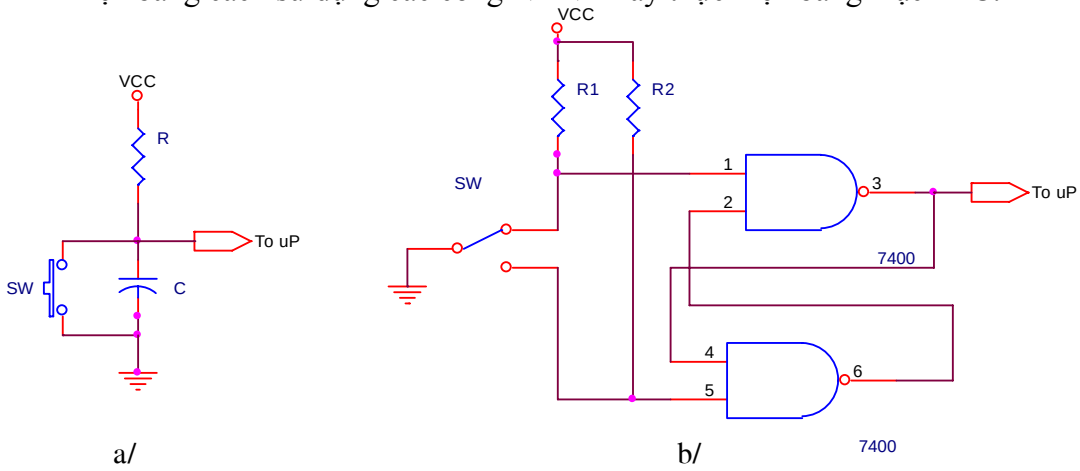
Các phím đơn dùng để điều khiển khi hệ thống vi xử lý không đòi hỏi nhiều giá trị nhập (chẳng như chỉ cần các điều khiển đóng mở thiết bị). Sơ đồ kết nối phím đơn mô tả như hình 4.8 hay 4.9.

Khi thực hiện kiểm tra phím nhấn, vấn đề cần thiết là phải thực hiện chống dội. Hiện tượng dội khi nhấn phím có thể mô tả như hình 4.10 (giả sử khi nhấn phím thì ngõ ra ở mức logic 1 và nhả phím thì ngõ ra ở mức logic 0 – hình 4.8b). Quá trình chống dội có thể thực hiện bằng phần mềm hay phần cứng.

Hình 4.10 – Hiện tượng dội khi nhấn phím

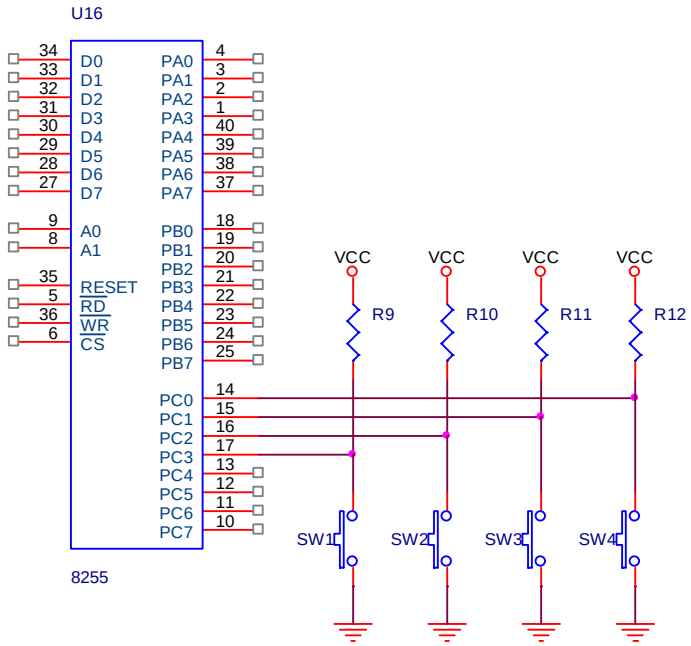
- **Phần mềm:** Do thời gian dội của phím vào khoảng 20ms nên quá trình chống dội bằng phần mềm đơn giản là tạo một thời gian trễ đủ lớn để chương trình bỏ qua ảnh hưởng khi dội.

- **Phần cứng:** Khi thực hiện chống dội bằng phần cứng, ta có thể thực hiện bằng cách sử dụng các cổng NAND hay thực hiện bằng mạch RC.



Hình 4.11 – Chống dội phím nhấn bằng phần cứng

Tuy nhiên khi thực hiện thiết kế phần cứng, thông thường ta sẽ chống dội bằng phần mềm để đơn giản mạch phần cứng. Xét sơ đồ phần cứng thiết kế như hình 4.12 (giả sử địa chỉ Port A, B, C và CR của 8255 lần lượt là 300h, 301h, 302h và 303h).



Hình 4.12 – Kết nối phím nhấn với 8255

Chương trình hợp ngữ kiểm tra các phím SW1, SW2, SW3, SW4 và hiển thị trạng thái phím nhấn lên màn hình:

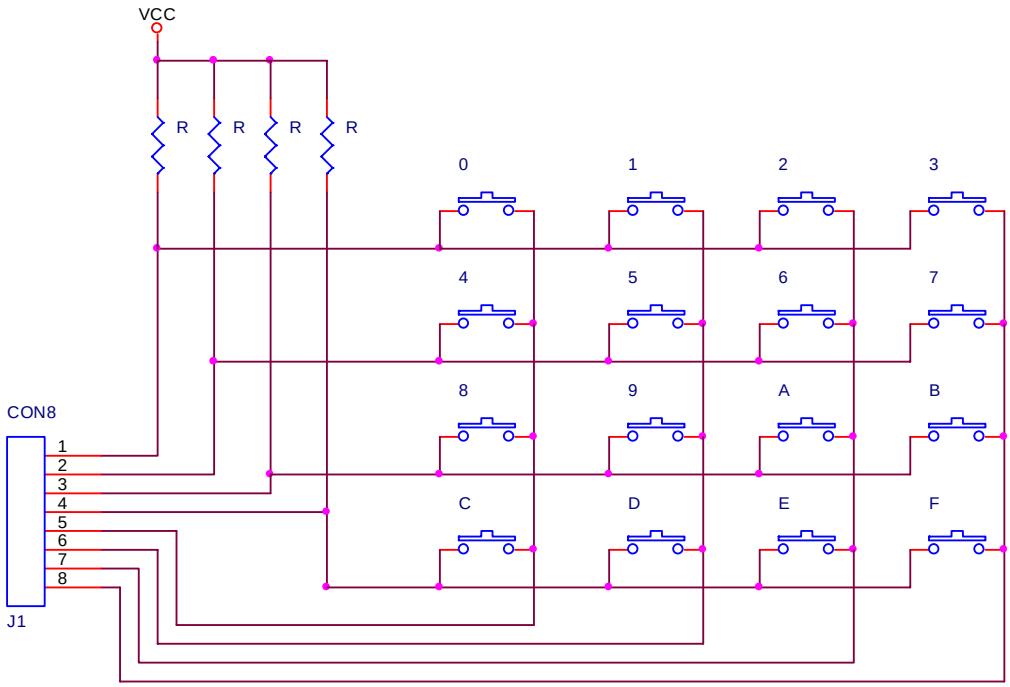
```
.MODEL SMALL
.STACK 100h
.DATA
     Msg DB 'Da nhan phim: SW'
     sw   DB   ?,13,10,'$'
.CODE
Main PROC
     MOV AX,@DATA
     MOV DS,AX

     MOV AL,81h          ; Định cấu hình cho 8255
     MOV DX,303h         ; PA, PB, PC (high): xuất
     OUT DX,AL           ; PC (low): nhập


Start:
     MOV AH,0Bh          ; Kiểm tra phím nhấn
     INT 21h
     CMP AL,0            ; Nếu có phím nhấn
     JNE Exit            ; thì kết thúc chương trình

     MOV DX,302h         ; Đọc từ Port C để kiểm tra
     IN AL,DX            ; phím nhấn
```

```
        AND AL,0Fh          ; Xoá 4 bit cao
        CMP  AL,00001110b   ; Nếu nhấn SW1 thì PC0 = 0
        JE Sw1
        CMP  AL,00001101b   ; Nếu nhấn SW2 thì PC1 = 0
        JE Sw2
        CMP  AL,00001011b   ; Nếu nhấn SW3 thì PC2 = 0
        JE Sw3
        CMP  AL,00000111b   ; Nếu nhấn SW4 thì PC3 = 0
        JE Sw4
        JMP Start
    Sw1:
        CALL Delay
        MOV sw,'1'
        MOV AH,09h
        LEA DX,Msg
        INT 21h
        JMP Start
    Sw2:
        CALL Delay
        MOV sw,'2'
        MOV AH,09h
        LEA DX,Msg
        INT 21h
        JMP Start
    Sw3:
        CALL Delay
        MOV sw,'3'
        MOV AH,09h
        LEA DX,Msg
        INT 21h
        JMP Start
    Sw4:
        CALL Delay
        MOV sw,'4'
        MOV AH,09h
        LEA DX,Msg
        INT 21h
        JMP Start

    Exit:
        MOV AH,4Ch          ; Kết thúc chương trình
        INT 21h

    Main ENDP
    ;-------------
    Delay PROC
        PUSH CX
        MOV CX,0FFFFh
        LOOP $
```

```
        POP CX
        RET
Delay ENDP
;-------------
END Main
```

2.2. Giao tiếp bàn phím Hex

Bàn phím Hex là bàn phím xây dựng theo cấu trúc ma trận gồm 16 phím chia thành 4 hàng và 4 cột (hình 4.13).



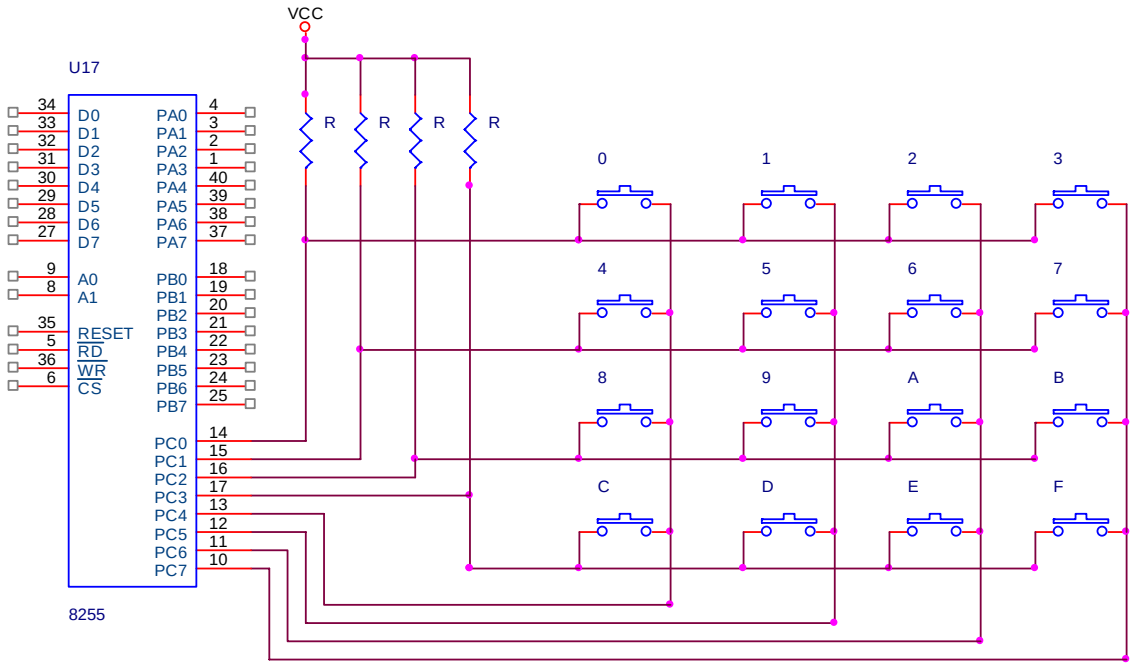
Hình 4.13 – Cấu trúc bàn phím Hex

Lưu ý rằng khi không nhấn phím thì hàng của bàn phím Hex nối với Vcc thông qua điện trở R nên có mức logic 1. Để phân biệt được trạng thái của phím nhấn thì mức logic khi nhấn phím phải là mức logic 0. Mà khi nhấn một phím nào đó thì tương ứng hàng và cột của bàn phím Hex sẽ kết nối với nhau. Do đó, để thực hiện kiểm tra một phím thì ta phải cho trước cột chứa phím tương ứng ở mức logic 0, sau đó kiểm tra hàng của phím, nếu hàng = 0 thì có nhấn phím còn hàng = 1 thì không nhấn phím.

Ví dụ như muốn kiểm tra phím 4 thì ta cho cột chứa phím 4 ở mức logic 0 (chân 5 của J1, các cột khác = 1, nghĩa là dữ liệu tại J1 là 1000xxxxb), sau đó thực hiện kiểm tra chân 2 của J1 (hàng của phím 4), nếu chân này = 0 thì phím 4 được nhấn.

Xét sơ đồ kết nối của bàn phím Hex với 8255 như hình 4.14 (giả sử địa chỉ Port A, B, C và CR của 8255 lần lượt là 300h, 301h, 302h và 303h). Ở đây ta thực hiện kết nối với Port C của 8255 do Port C có thể chia thành 2 phần: 4 bit cao và 4 bit thấp có thể xuất nhập độc lập trong khi đó Port A và Port B chỉ có thể đồng thời xuất hay nhập

còn để xác định có phím nhấn trên bàn phím Hex hay không thì phải xuất dữ liệu điều khiển cột và đọc dữ liệu từ hàng của bàn phím.



Hình 4.14 – Sơ đồ kết nối bàn phím Hex với 8255

Chương trình hợp ngữ kiểm tra bàn phím như sau:

```
.MODEL SMALL
.STACK 100h
.DATA
.CODE
Main PROC
     MOV AX,@DATA
     MOV DS,AX

     MOV AL,81h          ; Định cấu hình cho 8255
     MOV DX,303h         ; PA, PB, PC (high): xuất
     OUT DX,AL           ; PC (low): nhập


Start:
     MOV AH,0Bh          ; Kiểm tra phím nhấn
     INT 21h
     CMP AL,0            ; Nếu có phím nhấn
     JE Next
     JMP Exit            ; thì kết thúc chương trình

Next:
     MOV DX,302h         ; Địa chỉ Port C
     MOV AL,11100000b    ; Cho PC4 = 0 ứng với các
```

```
        OUT DX,AL           ; phím 0,4,8,C
        IN AL,DX            ; Đọc về kiểm tra hàng
        AND AL,0Fh          ; Xoá 4 bit cao
        CMP  AL,00001110b   ; Nếu nhấn phím 0 thì PC0 = 0
        JNE Not0
        CALL Sw0

    Not0:
        CMP  AL,00001101b   ; Nếu nhấn phím 4 thì PC1 = 0
        JNE Not4
        CALL Sw4

    Not4:
        CMP  AL,00001011b   ; Nếu nhấn phím 8 thì PC2 = 0
        JNE Not8
        CALL Sw8

    Not8:
        CMP  AL,00000111b   ; Nếu nhấn phím C thì PC3 = 0
        JNE NotC
        CALL SwC

    NotC:
        MOV DX,302h         ; Địa chỉ Port C
        MOV AL,11010000b    ; Cho PC5 = 0 ứng với các
        OUT DX,AL           ; phím 1,5,9,D
        IN AL,DX            ; Đọc về kiểm tra hàng
        AND AL,0Fh          ; Xoá 4 bit cao
        CMP  AL,00001110b   ; Nếu nhấn phím 1 thì PC0 = 0
        JNE Not1
        CALL Sw1

    Not1:
        CMP  AL,00001101b   ; Nếu nhấn phím 5 thì PC1 = 0
        JNE Not5
        CALL Sw5

    Not5:
        CMP  AL,00001011b   ; Nếu nhấn phím 9 thì PC2 = 0
        JNE Not9
        CALL Sw9

    Not9:
        CMP  AL,00000111b   ; Nếu nhấn phím D thì PC3 = 0
        JNE NotD
        CALL SwD

    NotD:
        MOV DX,302h         ; Địa chỉ Port C
```

```
        MOV AL,10110000b   ; Cho PC6 = 0 ứng với các
        OUT DX,AL          ; phím 2,6,A,E
        IN AL,DX           ; Đọc về kiểm tra hàng
        AND AL,0Fh         ; Xoá 4 bit cao
        CMP  AL,00001110b  ; Nếu nhấn phím 2 thì PC0 = 0
        JNE Not2
        CALL Sw2

    Not2:
        CMP  AL,00001101b  ; Nếu nhấn phím 6 thì PC1 = 0
        JNE Not6
        CALL Sw6

    Not6:
        CMP  AL,00001011b  ; Nếu nhấn phím A thì PC2 = 0
        JNE NotA
        CALL SwA

    NotA:
        CMP  AL,00000111b  ; Nếu nhấn phím E thì PC3 = 0
        JNE NotE
        CALL SwE

    NotE:
        MOV DX,302h        ; Địa chỉ Port C
        MOV AL,01110000b   ; Cho PC7 = 0 ứng với các
        OUT DX,AL          ; phím 3,7,B,F
        IN AL,DX           ; Đọc về kiểm tra hàng
        AND AL,0Fh         ; Xoá 4 bit cao
        CMP  AL,00001110b  ; Nếu nhấn phím 3 thì PC0 = 0
        JNE Not3
        CALL Sw3

    Not3:
        CMP  AL,00001101b  ; Nếu nhấn phím 7 thì PC1 = 0
        JNE Not7
        CALL Sw7

    Not7:
        CMP  AL,00001011b  ; Nếu nhấn phím B thì PC2 = 0
        JNE NotB
        CALL SwB

    NotB:
        CMP  AL,00000111b  ; Nếu nhấn phím F thì PC3 = 0
        JNE NotF
        CALL SwF

    NotF:
```

```
     JMP Start

Exit:
     MOV AH,4Ch         ; Kết thúc chương trình
     INT 21h
Main ENDP
;-----------

Sw0 PROC
     ; Chương trình cho phím 0
     RET
Sw0 ENDP
;-----------

Sw1 PROC
     ; Chương trình cho phím 1
     RET
Sw1 ENDP
;-----------

Sw2 PROC
     ; Chương trình cho phím 2
     RET
Sw2 ENDP
;-----------

Sw3 PROC
     ; Chương trình cho phím 3
     RET
Sw3 ENDP
;-----------

Sw4 PROC
     ; Chương trình cho phím 4
     RET
Sw4 ENDP
;-----------

Sw5 PROC
     ; Chương trình cho phím 5
     RET
Sw5 ENDP
;-----------

Sw6 PROC
     ; Chương trình cho phím 6
     RET
Sw6 ENDP
;-----------
```

```
Sw7 PROC
     ; Chương trình cho phím 7
     RET
Sw7 ENDP
;-----------

Sw8 PROC
     ; Chương trình cho phím 8
     RET
Sw8 ENDP
;-----------
Sw9 PROC
     ; Chương trình cho phím 9
     RET
Sw9 ENDP
;-----------

SwA PROC
     ; Chương trình cho phím A
     RET
SwA ENDP
;-----------

SwB PROC
     ; Chương trình cho phím B
     RET
SwB ENDP
;-----------

SwC PROC
     ; Chương trình cho phím C
     RET
SwC ENDP
;-----------

SwD PROC
     ; Chương trình cho phím D
     RET
SwD ENDP
;-----------

SwE PROC
     ; Chương trình cho phím E
     RET
SwE ENDP
;-----------

SwF PROC
```

```
        ; Chương trình cho phím F
        RET
SwF ENDP
;-----------

Delay PROC
        PUSH CX
        MOV CX,0FFFFh
        LOOP $
        POP CX
        RET
Delay ENDP
END Main
```

BÀI TẬP CHƯƠNG 4

1. Giả sử Port A của 8255 kết nối như hình 4.4, Port C kết nối với 2 công tắc SW1, SW2 tương ứng tại PC7, PC6 tương tự như hình 4.12. Viết chương trình hợp ngữ điều khiển công tắc sao cho:

- Nhấn SW1: LED sáng tuần tự từ trong ra ngoài, mỗi lần sáng tương ứng 2 LED.

- Nhấn SW2: tắt các LED và kết thúc chương trình

2. Giả sử Port A và Port B của 8255 kết nối với ma trận LED 5x8 như hình 4.7 còn Port C kết nối với bàn phím Hex như hình 4.14. Viết chương trình hợp ngữ điều khiển bàn phím Hex sao cho:

- Nhấn phím 2: sáng số ‘2’ trên ma trận LED

- Nhấn phím D: sáng chữ ‘D’ trên ma trận LED

82C55A

CMOS Programmable Peripheral Interface

June 1998

Features

- **Pin Compatible with NMOS 8255A**
- **24 Programmable I/O Pins**
- **Fully TTL Compatible**
- **High Speed, No "Wait State" Operation with 5MHz and 8MHz 80C86 and 80C88**
- **Direct Bit Set/Reset Capability**
- **Enhanced Control Word Read Capability**
- **L7 Process**
- **2.5mA Drive Capability on All I/O Ports**
- **Low Standby Power (ICCSB) 10μA**

Ordering Information

PART NUMBERS		PACKAGE	TEMPERATURE RANGE	PKG. NO.
5MHz	8MHz			
CP82C55A-5	CP82C55A	40 Ld PDIP	0^oC to 70^oC	E40.6
IP82C55A-5	IP82C55A		-40^oC to 85^oC	E40.6
CS82C55A-5	CS82C55A	44 Ld PLCC	0^oC to 70^oC	N44.65
IS82C55A-5	IS82C55A		-40^oC to 85^oC	N44.65
CD82C55A-5	CD82C55A	40 Ld CERDIP	0^oC to 70^oC	F40.6
ID82C55A-5	ID82C55A		-40^oC to 85^oC	F40.6
MD82C55A-5/B	MD82C55A/B		-55^oC to 125^oC	F40.6
8406601QA	8406602QA	SMD#		F40.6
MR82C55A-5/B	MR82C55A/B	44 Pad CLCC	-55^oC to 125^oC	J44.A
8406601XA	8406602XA	SMD#		J44.A

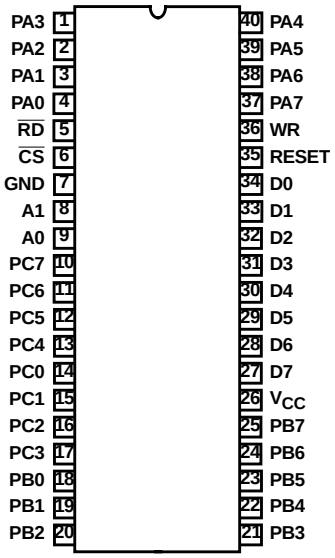
Description

The Intersil 82C55A is a high performance CMOS version of the industry standard 8255A and is manufactured using a self-aligned silicon gate CMOS process (Scaled SAJI IV). It is a general purpose programmable I/O device which may be used with many different microprocessors. There are 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. The high performance and industry standard configuration of the 82C55A make it compatible with the 80C86, 80C88 and other microprocessors.

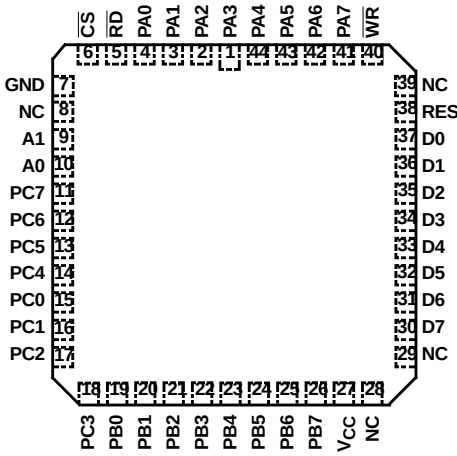
Static CMOS circuit design insures low operating power. TTL compatibility over the full military temperature range and bus hold circuitry eliminate the need for pull-up resistors. The Intersil advanced SAJI process results in performance equal to or greater than existing functionally equivalent products at a fraction of the power.

Pinouts

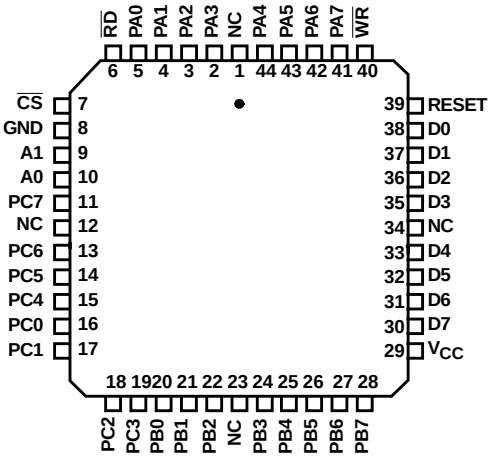
82C55A (DIP)
TOP VIEW



82C55A (CLCC)
TOP VIEW



82C55A (PLCC)
TOP VIEW

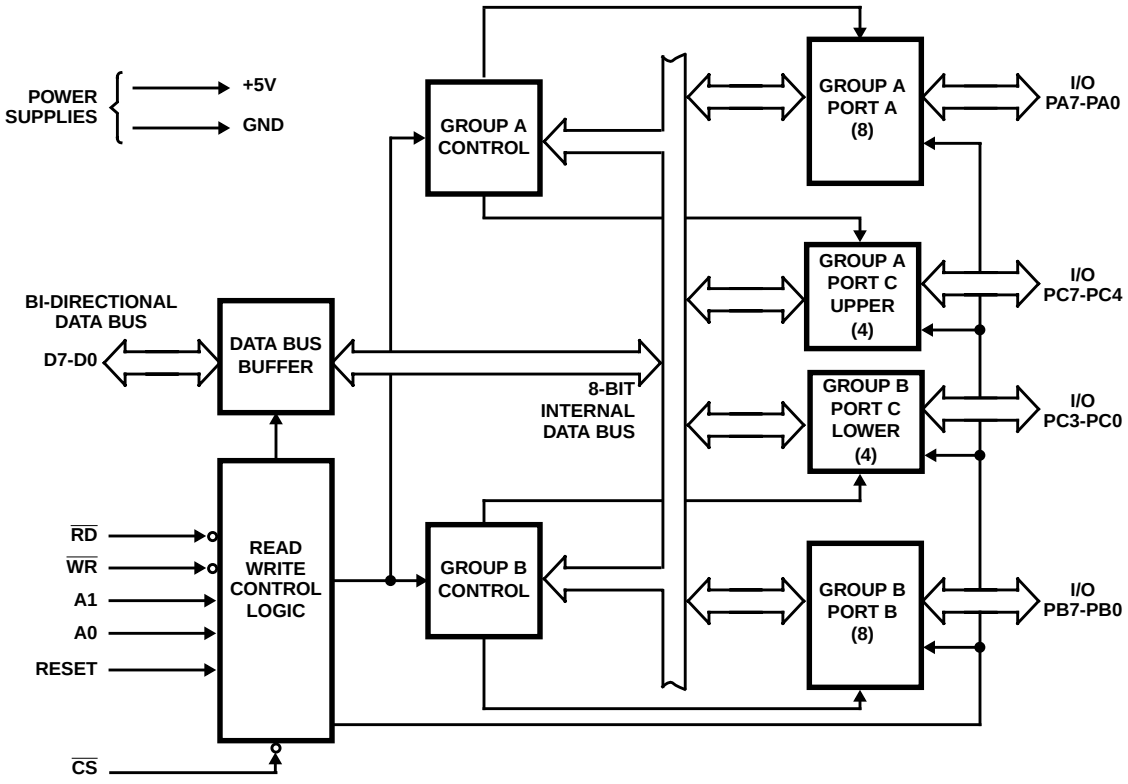


File Number **2969.2**

Pin Description

SYMBOL	PIN NUMBER	TYPE	DESCRIPTION
V_{CC}	26		V_{CC}: The +5V power supply pin. A 0.1µF capacitor between pins 26 and 7 is recommended for decoupling.
GND	7		GROUND
D0-D7	27-34	I/O	DATA BUS: The Data Bus lines are bidirectional three-state pins connected to the system data bus.
RESET	35	I	RESET: A high on this input clears the control register and all ports (A, B, C) are set to the input mode with the “Bus Hold” circuitry turned on.
$\overline{CS}$	6	I	CHIP SELECT: Chip select is an active low input used to enable the 82C55A onto the Data Bus for CPU communications.
$\overline{RD}$	5	I	READ: Read is an active low input control signal used by the CPU to read status information or data via the data bus.
$\overline{WR}$	36	I	WRITE: Write is an active low input control signal used by the CPU to load control words and data into the 82C55A.
A0-A1	8, 9	I	ADDRESS: These input signals, in conjunction with the $\overline{RD}$ and $\overline{WR}$ inputs, control the selection of one of the three ports or the control word register. A0 and A1 are normally connected to the least significant bits of the Address Bus A0, A1.
PA0-PA7	1-4, 37-40	I/O	PORT A: 8-bit input and output port. Both bus hold high and bus hold low circuitry are present on this port.
PB0-PB7	18-25	I/O	PORT B: 8-bit input and output port. Bus hold high circuitry is present on this port.
PC0-PC7	10-17	I/O	PORT C: 8-bit input and output port. Bus hold circuitry is present on this port.

Functional Diagram



Functional Description

Data Bus Buffer

This three-state bi-directional 8-bit buffer is used to interface the 82C55A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

(CS) Chip Select. A “low” on this input pin enables the communcation between the 82C55A and the CPU.

(RD) Read. A “low” on this input pin enables 82C55A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to “read from” the 82C55A.

(WR) Write. A “low” on this input pin enables the CPU to write data or control words into the 82C55A.

(A0 and A1) Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word register. They are normally connected to the least significant bits of the address bus (A0 and A1).

82C55A BASIC OPERATION

A1	A0	RD	WR	CS	INPUT OPERATION (READ)
0	0	0	1	0	Port A → Data Bus
0	1	0	1	0	Port B → Data Bus
1	0	0	1	0	Port C → Data Bus
1	1	0	1	0	Control Word → Data Bus
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	Data Bus → Port A
0	1	1	0	0	Data Bus → Port B
1	0	1	0	0	Data Bus → Port C
1	1	1	0	0	Data Bus → Control
					DISABLE FUNCTION
X	X	X	X	1	Data Bus → Three-State
X	X	1	1	0	Data Bus → Three-State

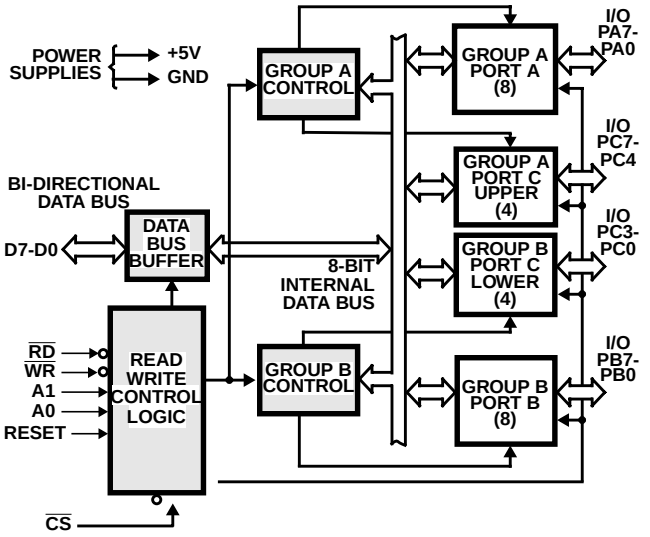


FIGURE 1. 82C55A BLOCK DIAGRAM. DATA BUS BUFFER, READ/WRITE, GROUP A & B CONTROL LOGIC FUNCTIONS

(RESET) Reset. A “high” on this input initializes the control register to 9Bh and all ports (A, B, C) are set to the input mode. “Bus hold” devices internal to the 82C55A will hold the I/O port inputs to a logic “1” state with a maximum hold current of 400µA.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU “outputs” a control word to the 82C55A. The control word contains information such as “mode”, “bit set”, “bit reset”, etc., that initializes the functional configuration of the 82C55A.

Each of the Control blocks (Group A and Group B) accepts “commands” from the Read/Write Control logic, receives “control words” from the internal data bus and issues the proper commands to its associated ports.

Control Group A - Port A and Port C upper (C7 - C4)

Control Group B - Port B and Port C lower (C3 - C0)

The control word register can be both written and read as shown in the “Basic Operation” table. Figure 4 shows the control word format for both Read and Write operations. When the control word is read, bit D7 will always be a logic “1”, as this implies control word mode information.

13Ports A, B, and C

The 82C55A contains three 8-bit ports (A, B, and C). All can be configured to a wide variety of functional characteristics by the system software but each has its own special features or “personality” to further enhance the power and flexibility of the 82C55A.

Port A One 8-bit data output latch/buffer and one 8-bit data input latch. Both “pull-up” and “pull-down” bus-hold devices are present on Port A. See Figure 2A.

Port B One 8-bit data input/output latch/buffer and one 8-bit data input buffer. See Figure 2B.

Port C One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal output and status signal inputs in conjunction with ports A and B. See Figure 2B.

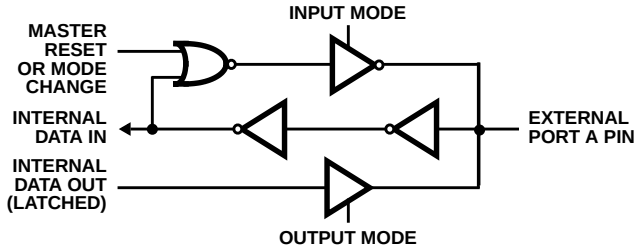


FIGURE 2A. PORT A BUS-HOLD CONFIGURATION

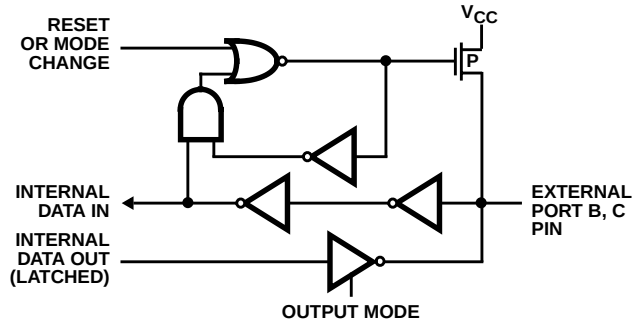


FIGURE 2B. PORT B AND C BUS-HOLD CONFIGURATION

FIGURE 2. BUS-HOLD CONFIGURATION

Operational Description

Mode Selection

There are three basic modes of operation than can be selected by the system software:

Mode 0 - Basic Input/Output
Mode 1 - Strobed Input/Output
Mode 2 - Bi-directional Bus

When the reset input goes “high”, all ports will be set to the input mode with all 24 port lines held at a logic “one” level by internal bus hold devices. After the reset is removed, the 82C55A can remain in the input mode with no additional initialization required. This eliminates the need to pullup or pull-down resistors in all-CMOS designs. The control word register will contain 9Bh. During the execution of the system program, any of the other modes may be selected using a single output instruction. This allows a single 82C55A to service a variety of peripheral devices with a simple software maintenance routine. Any port programmed as an output port is initialized to all zeros when the control word is written.

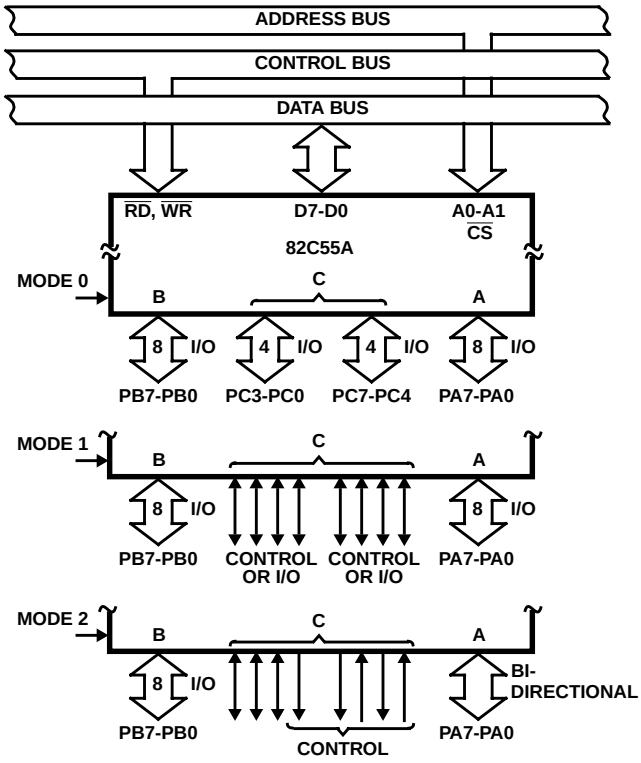


FIGURE 3. BASIC MODE DEFINITIONS AND BUS INTERFACE

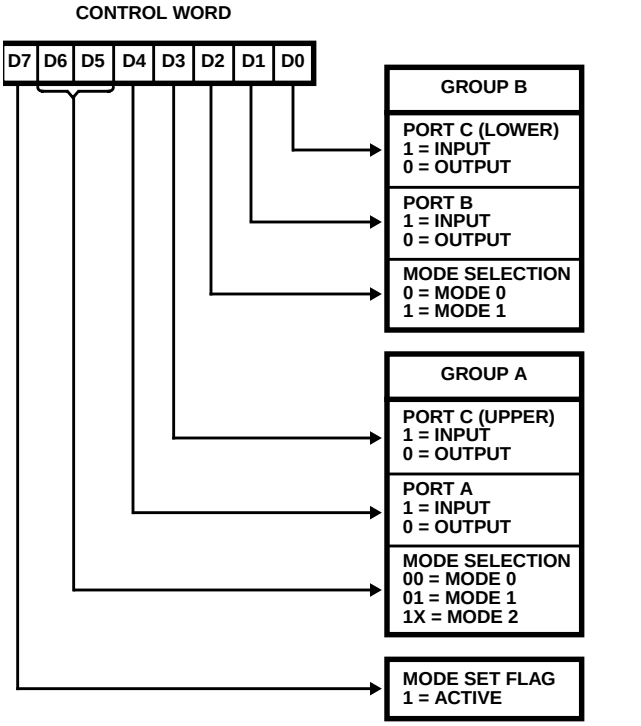


FIGURE 4. MODE DEFINITION FORMAT

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be “tailored” to almost any I/O structure. For instance: Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.

The mode definitions and possible mode combinations may seem confusing at first, but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 82C55A has taken into account things such as efficient PC board layout, control signal definition vs. PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature (Figure 5)

Any of the eight bits of Port C can be Set or Reset using a single Output instruction. This feature reduces software requirements in control-based applications.

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were output ports.

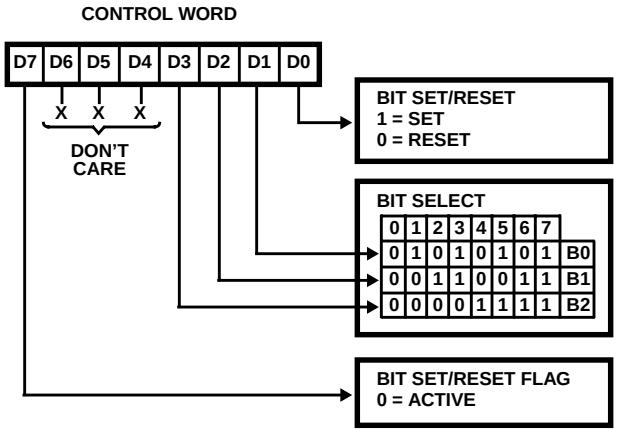


FIGURE 5. BIT SET/RESET FORMAT

Interrupt Control Functions

When the 82C55A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the bit set/reset function of port C.

This function allows the programmer to enable or disable a CPU interrupt by a specific I/O device without affecting any other device in the interrupt structure.

INTE Flip-Flop Definition

(BIT-SET)-INTE is SET - Interrupt Enable

(BIT-RESET)-INTE is Reset - Interrupt Disable

NOTE: All Mask flip-flops are automatically reset during mode selection and device Reset.

Operating Modes

Mode 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No handshaking is required, data is simply written to or read from a specific port.

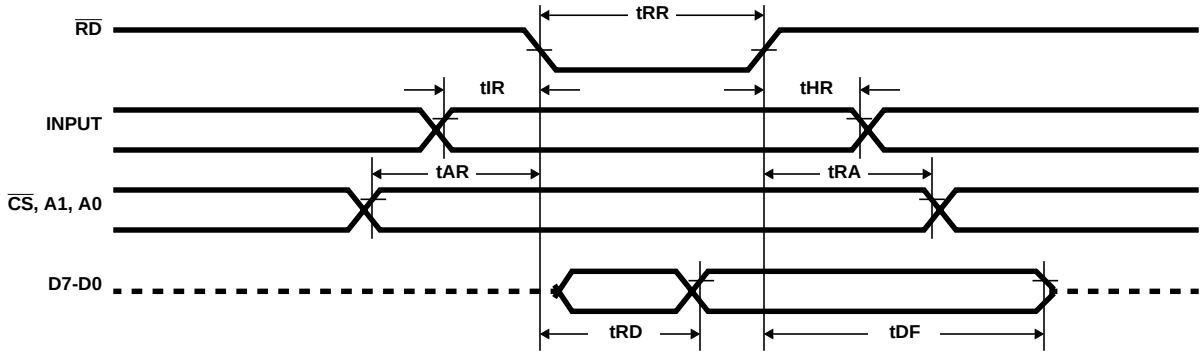
Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports
- Any Port can be input or output
- Outputs are latched
- Input are not latched
- 16 different Input/Output configurations possible

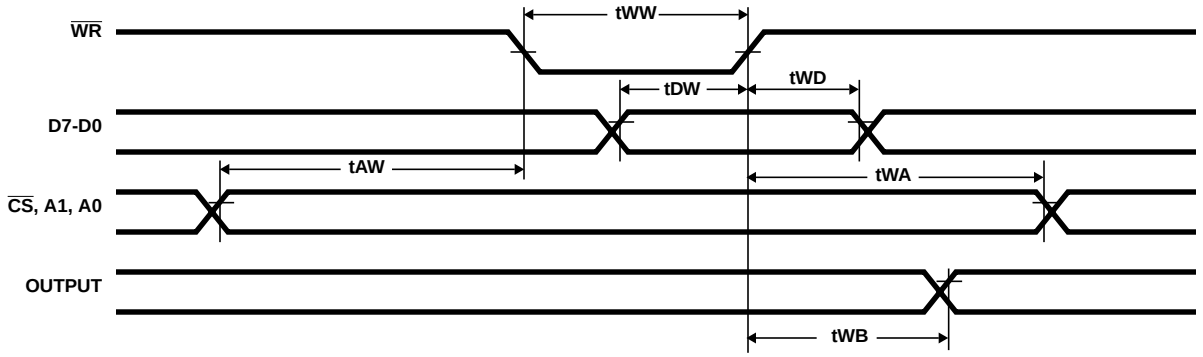
MODE 0 PORT DEFINITION

A		B		GROUP A			GROUP B	
D4	D3	D1	D0	PORT A	PORT C (Upper)	#	PORT B	PORT C (Lower)
0	0	0	0	Output	Output	0	Output	Output
0	0	0	1	Output	Output	1	Output	Input
0	0	1	0	Output	Output	2	Input	Output
0	0	1	1	Output	Output	3	Input	Input
0	1	0	0	Output	Input	4	Output	Output
0	1	0	1	Output	Input	5	Output	Input
0	1	1	0	Output	Input	6	Input	Output
0	1	1	1	Output	Input	7	Input	Input
1	0	0	0	Input	Output	8	Output	Output
1	0	0	1	Input	Output	9	Output	Input
1	0	1	0	Input	Output	10	Input	Output
1	0	1	1	Input	Output	11	Input	Input
1	1	0	0	Input	Input	12	Output	Output
1	1	0	1	Input	Input	13	Output	Input
1	1	1	0	Input	Input	14	Input	Output
1	1	1	1	Input	Input	15	Input	Input

Mode 0 (Basic Input)



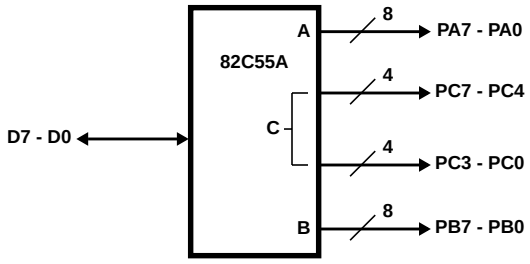
Mode 0 (Basic Output)



Mode 0 Configurations

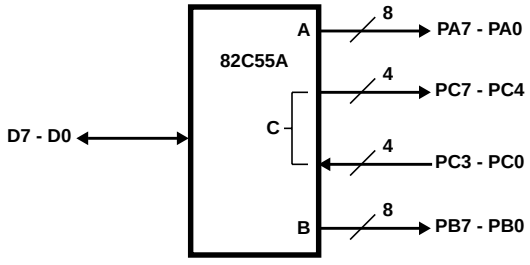
CONTROL WORD #0

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0



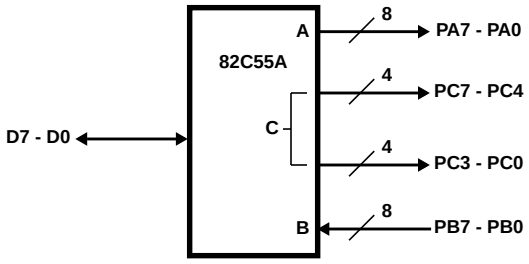
CONTROL WORD #1

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	1



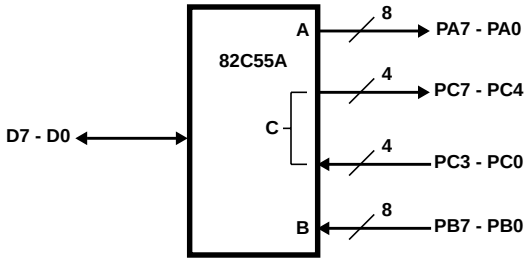
CONTROL WORD #2

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	1	0



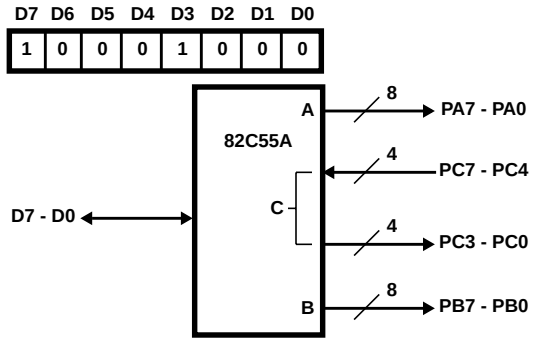
CONTROL WORD #3

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	1	1

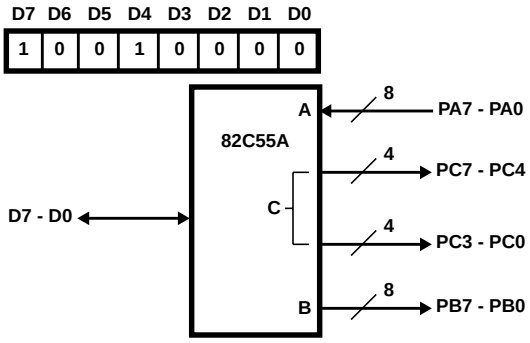


Mode 0 Configurations (Continued)

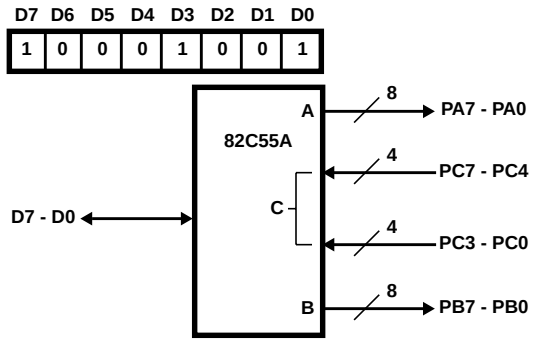
CONTROL WORD #4



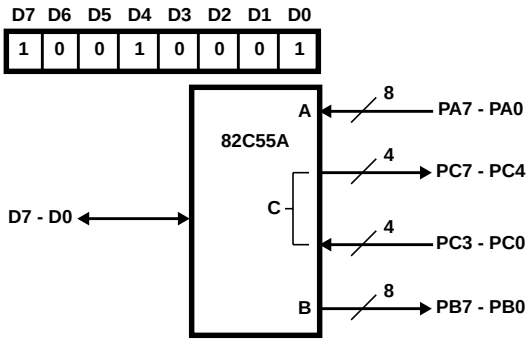
CONTROL WORD #8



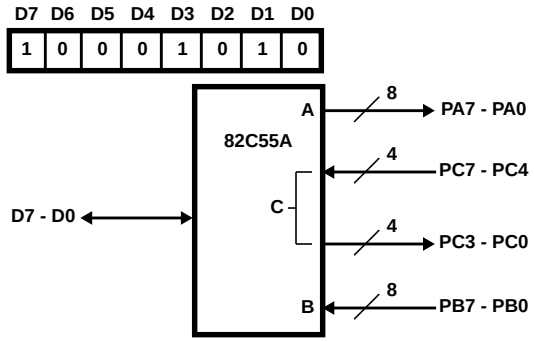
CONTROL WORD #5



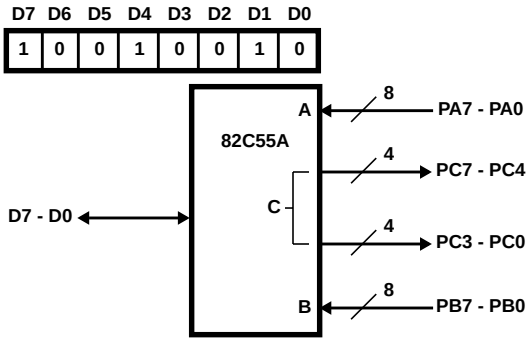
CONTROL WORD #9



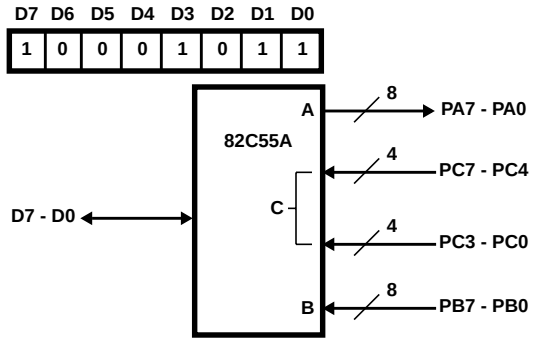
CONTROL WORD #6



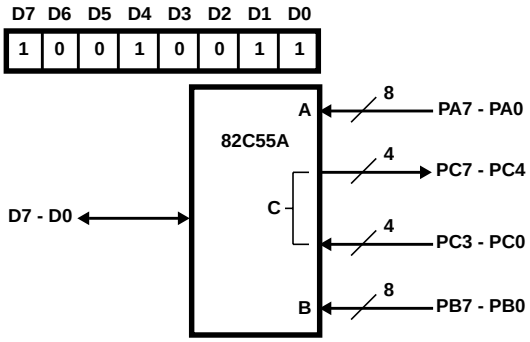
CONTROL WORD #10



CONTROL WORD #7

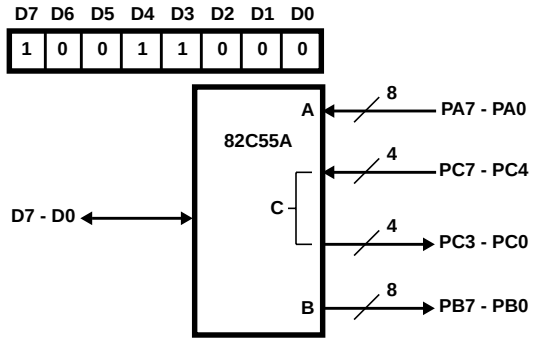


CONTROL WORD #11

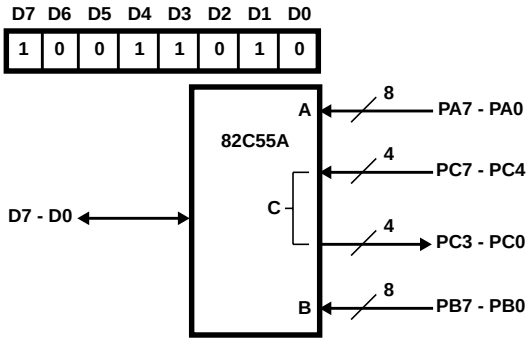


Mode 0 Configurations (Continued)

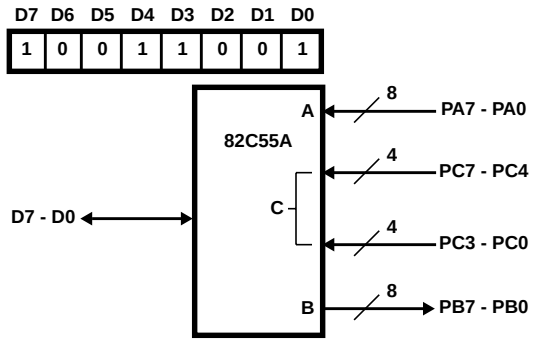
CONTROL WORD #12



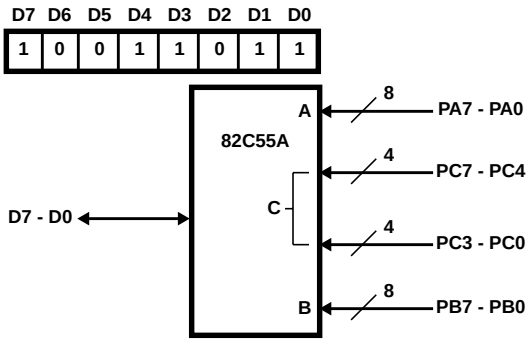
CONTROL WORD #14



CONTROL WORD #13



CONTROL WORD #15



Operating Modes

Mode 1 - (Strobed Input/Output). This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or “hand shaking” signals. In mode 1, port A and port B use the lines on port C to generate or accept these “hand shaking” signals.

Mode 1 Basic Function Definitions:

- Two Groups (Group A and Group B)
- Each group contains one 8-bit port and one 4-bit control/data port
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit port.

Input Control Signal Definition

(Figures 6 and 7)

STB (Strobe Input)

A “low” on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A “high” on this output indicates that the data has been loaded into the input latch: in essence, and acknowledgment. IBF is set by $\overline{STB}$ input being low and is reset by the rising edge of the $\overline{RD}$ input.

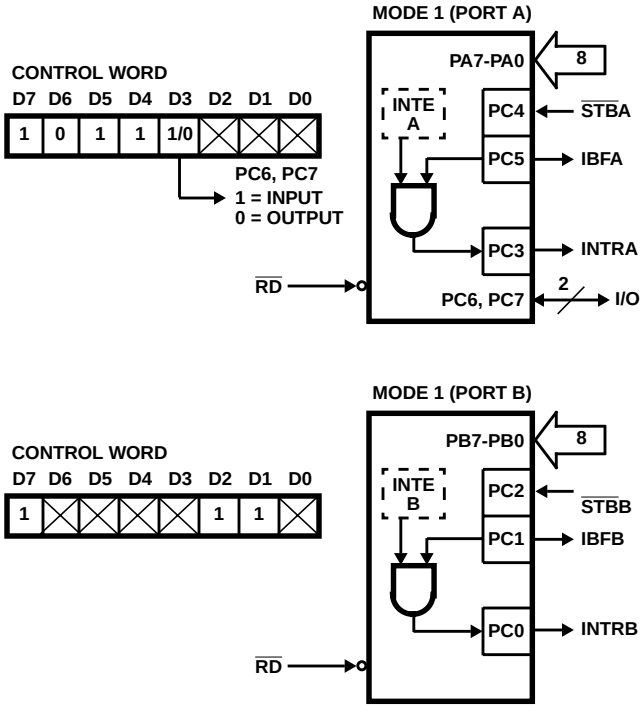


FIGURE 6. MODE 1 INPUT

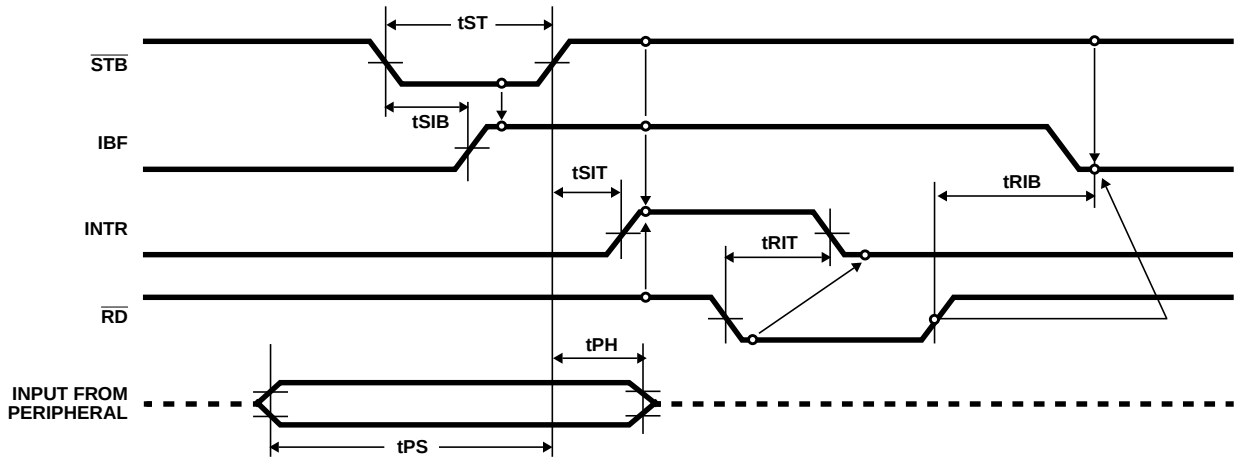


FIGURE 7. MODE 1 (STROBED INPUT)

INTR (Interrupt Request)

A “high” on this output can be used to interrupt the CPU when and input device is requesting service. INTR is set by the condition: $\overline{STB}$ is a “one”, IBF is a “one” and INTE is a “one”. It is reset by the falling edge of $\overline{RD}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A

Controlled by bit set/reset of PC4.

INTE B

Controlled by bit set/reset of PC2.

Output Control Signal Definition

(Figure 8 and 9)

$\overline{OBF}$ - Output Buffer Full F/F). The $\overline{OBF}$ output will go “low” to indicate that the CPU has written data out to be specified port. This does not mean valid data is sent out of the part at this time since $\overline{OBF}$ can go true before data is available. Data is guaranteed valid at the rising edge of $\overline{OBF}$, (See Note 1). The $\overline{OBF}$ F/F will be set by the rising edge of the $\overline{WR}$ input and reset by $\overline{ACK}$ input being low.

$\overline{ACK}$ - Acknowledge Input). A “low” on this input informs the 82C55A that the data from Port A or Port B is ready to be accepted. In essence, a response from the peripheral device indicating that it is ready to accept data, (See Note 1).

INTR - (Interrupt Request). A “high” on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when $\overline{ACK}$ is a “one”, OBF is a “one” and INTE is a “one”. It is reset by the falling edge of $\overline{WR}$.

INTE A

Controlled by Bit Set/Reset of PC6.

INTE B

Controlled by Bit Set/Reset of PC2.

NOTE:

1. To strobe data into the peripheral device, the user must operate the strobe line in a hand shaking mode. The user needs to send $\overline{OBF}$ to the peripheral device, generates an $\overline{ACK}$ from the peripheral device and then latch data into the peripheral device on the rising edge of $\overline{OBF}$.

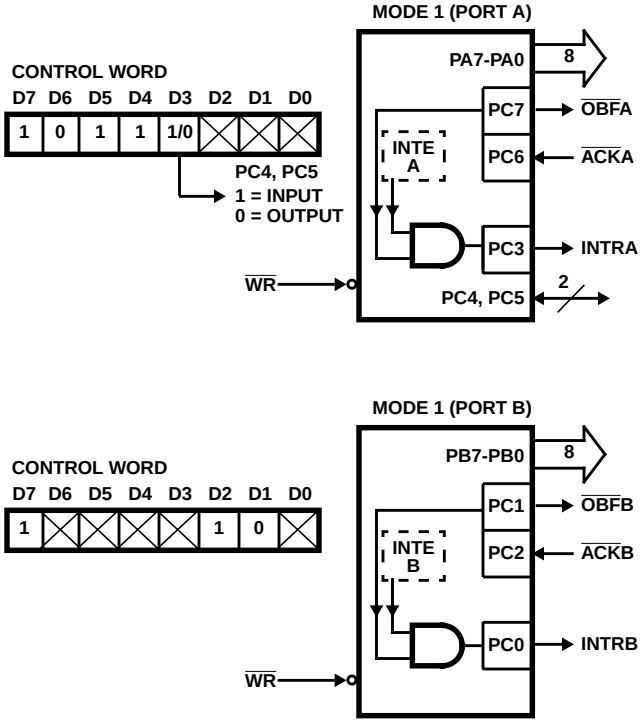


FIGURE 8. MODE 1 OUTPUT

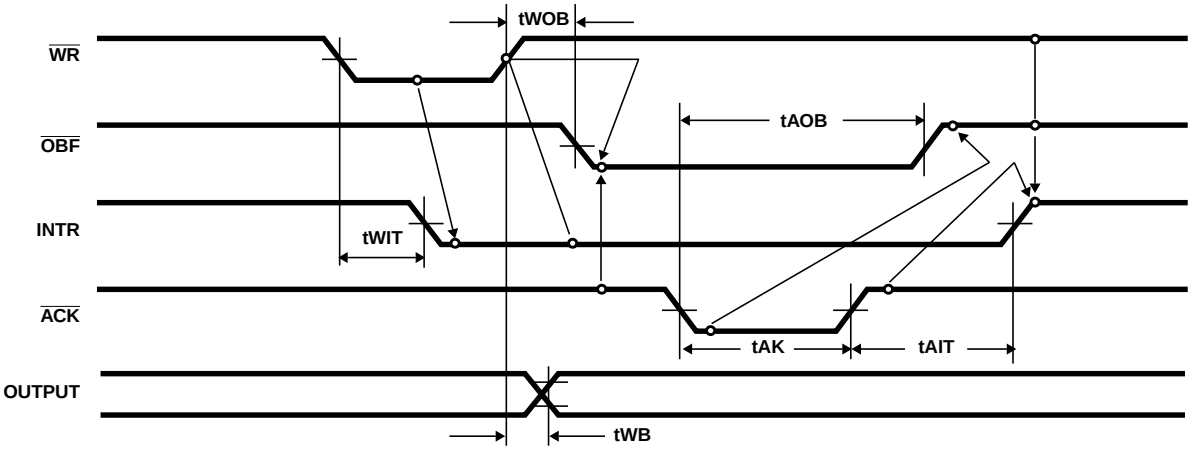
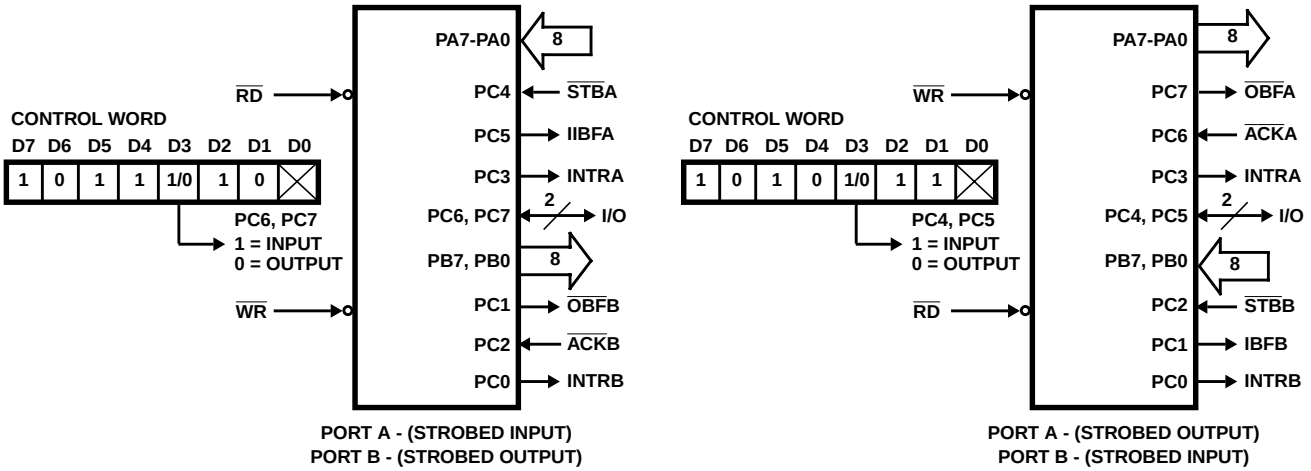


FIGURE 9. MODE 1 (STROBED OUTPUT)



Combinations of Mode 1: Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.

FIGURE 10. COMBINATIONS OF MODE 1

Operating Modes

Mode 2 (Strobed Bi-Directional Bus I/O)

The functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bi-directional bus I/O). “Hand shaking” signals are provided to maintain proper bus flow discipline similar to Mode 1. Interrupt generation and enable/disable functions are also available.

Mode 2 Basic Functional Definitions:

- Used in Group A only
- One 8-bit, bi-directional bus Port (Port A) and a 5-bit control Port (Port C)
- Both inputs and outputs are latched
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A)

Bi-Directional Bus I/O Control Signal Definition

(Figures 11, 12, 13, 14)

INTR - (Interrupt Request). A high on this output can be used to interrupt the CPU for both input or output operations.

Output Operations

$\overline{\text{OBF}}$ - (Output Buffer Full). The $\overline{\text{OBF}}$ output will go “low” to indicate that the CPU has written data out to port A.

$\overline{\text{ACK}}$ - (Acknowledge). A “low” on this input enables the three-state output buffer of port A to send out the data. Otherwise, the output buffer will be in the high impedance state.

INTE 1 - (The INTE flip-flop associated with $\overline{\text{OBF}}$). Controlled by bit set/reset of PC4.

Input Operations

$\overline{\text{STB}}$ - (Strobe Input). A “low” on this input loads data into the input latch.

IBF - (Input Buffer Full F/F). A “high” on this output indicates that data has been loaded into the input latch.

INTE 2 - (The INTE flip-flop associated with IBF). Controlled by bit set/reset of PC4.

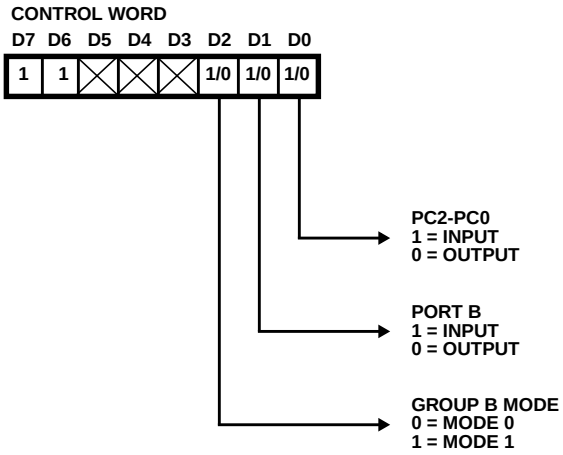


FIGURE 11. MODE CONTROL WORD

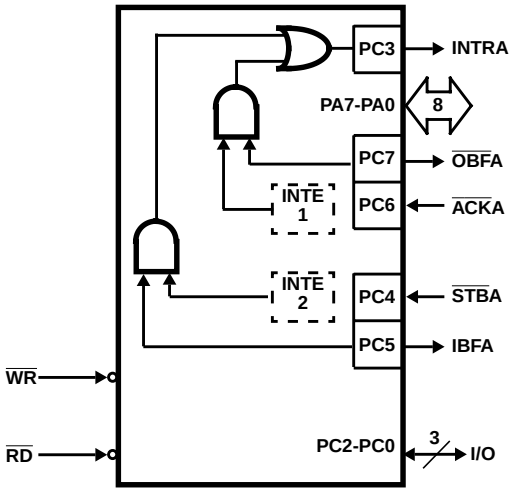
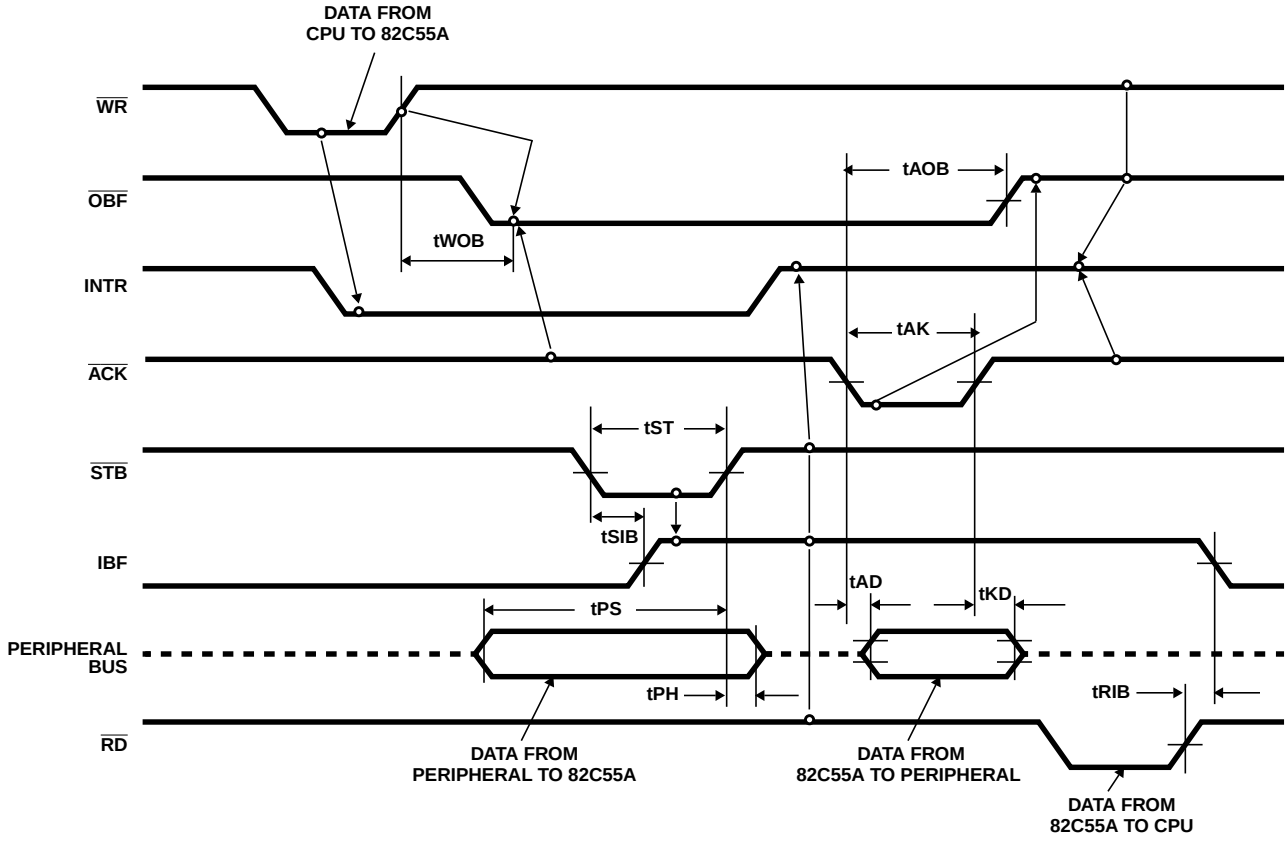


FIGURE 12. MODE 2



NOTE: Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before RD is permissible. (INTR = IBF • MASK • $\overline{STB}$ • $\overline{RD}$ ÷ $\overline{OBF}$ • MASK • $\overline{ACK}$ • $\overline{WR}$)

FIGURE 13. MODE 2 (BI-DIRECTIONAL)

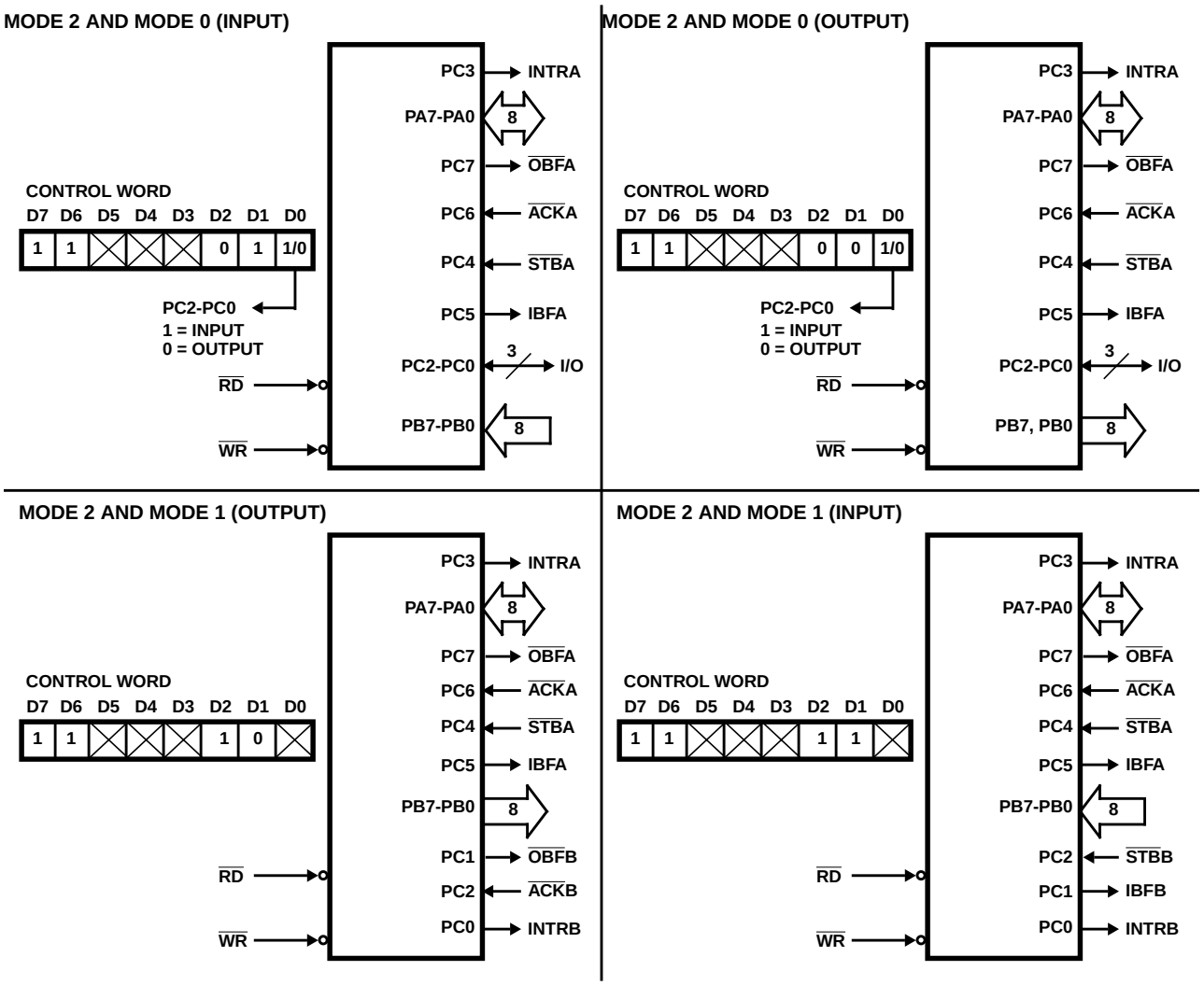


FIGURE 14. MODE 2 COMBINATIONS

MODE DEFINITION SUMMARY

	MODE 0		MODE 1		MODE 2	
	IN	OUT	IN	OUT	GROUP A ONLY	
PA0	In	Out	In	Out	← →	
PA1	In	Out	In	Out	← →	
PA2	In	Out	In	Out	← →	
PA3	In	Out	In	Out	← →	
PA4	In	Out	In	Out	← →	
PA5	In	Out	In	Out	← →	
PA6	In	Out	In	Out	← →	
PA7	In	Out	In	Out	← →	
PB0	In	Out	In	Out		Mode 0 or Mode 1 Only
PB1	In	Out	In	Out		
PB2	In	Out	In	Out		
PB3	In	Out	In	Out		
PB4	In	Out	In	Out		
PB5	In	Out	In	Out		
PB6	In	Out	In	Out		
PB7	In	Out	In	Out		
PC0	In	Out	INTRB	INTRB	I/O	
PC1	In	Out	IBFB	$\overline{OBF}$B	I/O	
PC2	In	Out	$\overline{STB}$B	$\overline{ACK}$B	I/O	
PC3	In	Out	INTRA	INTRA	INTRA	
PC4	In	Out	$\overline{STB}$A	I/O	$\overline{STB}$A	
PC5	In	Out	IBFA	I/O	IBFA	
PC6	In	Out	I/O	$\overline{ACK}$A	$\overline{ACK}$A	
PC7	In	Out	I/O	$\overline{OBF}$A	$\overline{OBF}$A	

Special Mode Combination Considerations

There are several combinations of modes possible. For any combination, some or all of Port C lines are used for control or status. The remaining bits are either inputs or outputs as defined by a “Set Mode” command.

During a read of Port C, the state of all the Port C lines, except the $\overline{ACK}$ and $\overline{STB}$ lines, will be placed on the data bus. In place of the $\overline{ACK}$ and $\overline{STB}$ line states, flag status will appear on the data bus in the PC2, PC4, and PC6 bit positions as illustrated by Figure 17.

Through a “Write Port C” command, only the Port C pins programmed as outputs in a Mode 0 group can be written. No other pins can be affected by a “Write Port C” command, nor can the interrupt enable flags be accessed. To write to any Port C output programmed as an output in Mode 1 group or to change an interrupt enable flag, the “Set/Reset Port C Bit” command must be used.

With a “Set/Reset Port Cea Bit” command, any Port C line programmed as an output (including IBF and $\overline{OBF}$) can be written, or an interrupt enable flag can be either set or reset. Port C lines programmed as inputs, including $\overline{ACK}$ and $\overline{STB}$ lines, associated with Port C fare not affected by a “Set/Reset Port C Bit” command. Writing to the corresponding Port C bit positions of the $\overline{ACK}$ and $\overline{STB}$ lines with the “Set Reset Port C Bit” command will affect the Group A and Group B interrupt enable flags, as illustrated in Figure 17.

INPUT CONFIGURATION

D7	D6	D5	D4	D3	D2	D1	D0
I/O	I/O	IBFA	INTEA	INTRA	INTEB	IBFB	INTRB
GROUP A					GROUP B		

OUTPUT CONFIGURATION

D7	D6	D5	D4	D3	D2	D1	D0
OBFA	INTEA	I/O	I/O	INTRA	INTEB	OBFB	INTRB
GROUP A					GROUP B		

FIGURE 15. MODE 1 STATUS WORD FORMAT

D7	D6	D5	D4	D3	D2	D1	D0
OBFA	INTE1	IBFA	INTE2	INTRA	X	X	X
GROUP A					GROUP B		

(Defined by Mode 0 or Mode 1 Selection)

FIGURE 16. MODE 2 STATUS WORD FORMAT

Current Drive Capability

Any output on Port A, B or C can sink or source 2.5mA. This feature allows the 82C55A to directly drive Darlington type drivers and high-voltage displays that require such sink or source current.

Reading Port C Status (Figures 15 and 16)

In Mode 0, Port C transfers data to or from the peripheral device. When the 82C55A is programmed to function in Modes 1 or 2, Port C generates or accepts “hand shaking” signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the “status” of each peripheral device and change the program flow accordingly.

There is not special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.

INTERRUPT ENABLE FLAG	POSITION	ALTERNATE PORT C PIN SIGNAL (MODE)
INTE B	PC2	$\overline{ACK}$B (Output Mode 1) or $\overline{STB}$B (Input Mode 1)
INTE A2	PC4	$\overline{STB}$A (Input Mode 1 or Mode 2)
INTE A1	PC6	$\overline{ACK}$A (Output Mode 1 or Mode 2)

FIGURE 17. INTERRUPT ENABLE FLAGS IN MODES 1 AND 2

Applications of the 82C55A

The 82C55A is a very powerful tool for interfacing peripheral equipment to the microcomputer system. It represents the optimum use of available pins and flexible enough to interface almost any I/O device without the need for additional external logic.

Each peripheral device in a microcomputer system usually has a “service routine” associated with it. The routine manages the software interface between the device and the CPU. The functional definition of the 82C55A is programmed by the I/O service routine and becomes an extension of the system software. By examining the I/O devices interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the detailed operational description, a control word can easily be developed to initialize the 82C55A to exactly “fit” the application. Figures 18 through 24 present a few examples of typical applications of the 82C55A.

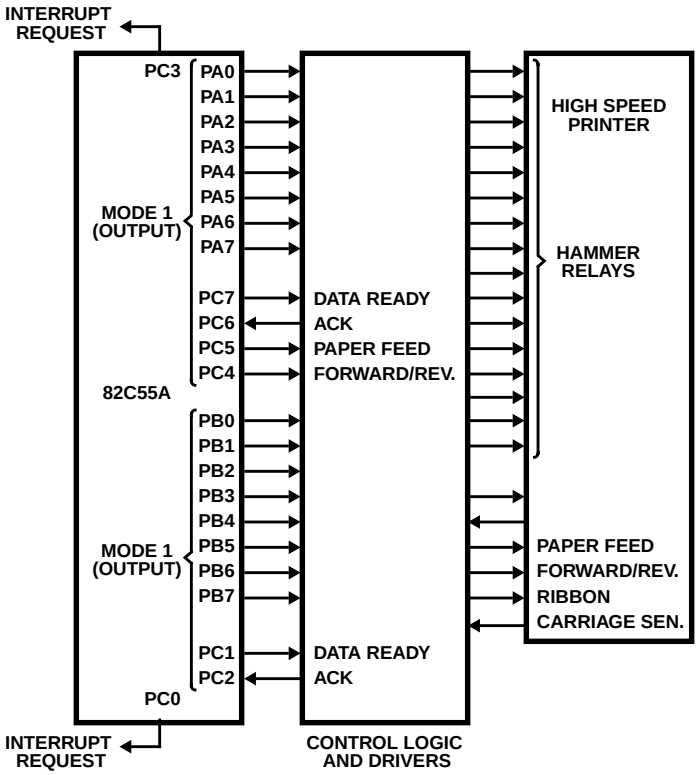


FIGURE 18. PRINTER INTERFACE

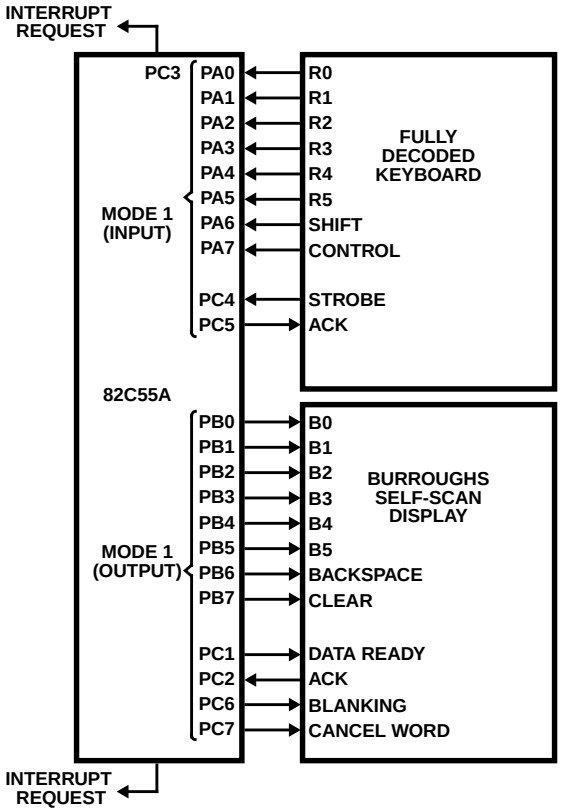


FIGURE 19. KEYBOARD AND DISPLAY INTERFACE

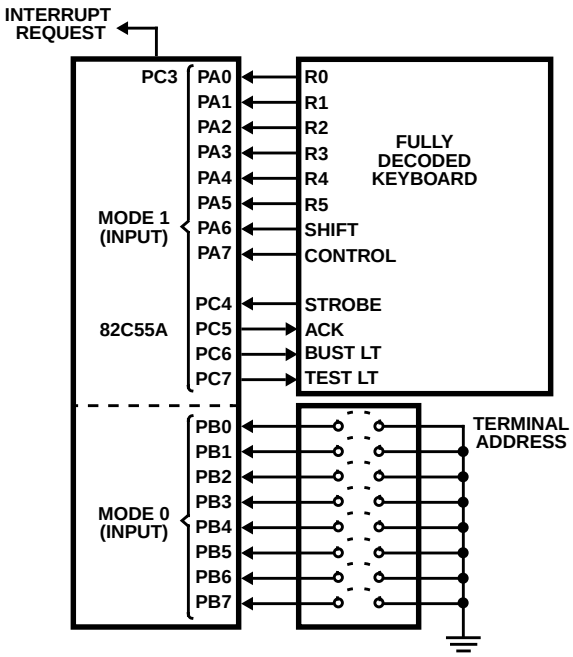


FIGURE 20. KEYBOARD AND TERMINAL ADDRESS INTERFACE

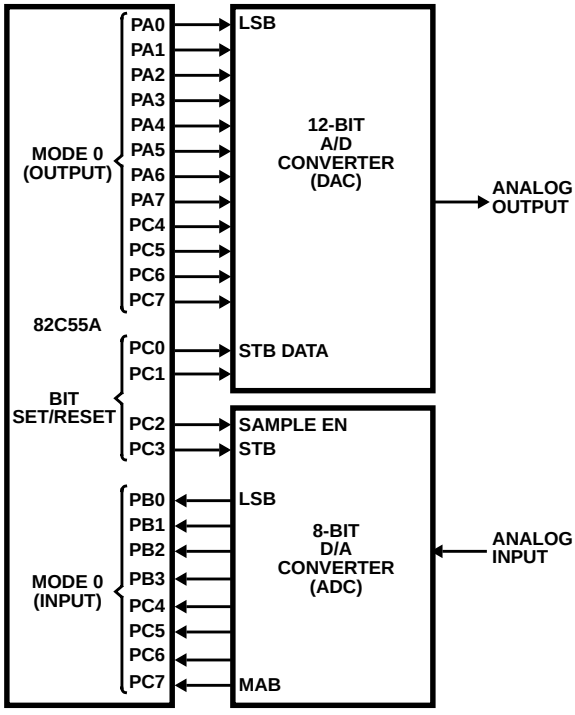


FIGURE 21. DIGITAL TO ANALOG, ANALOG TO DIGITAL

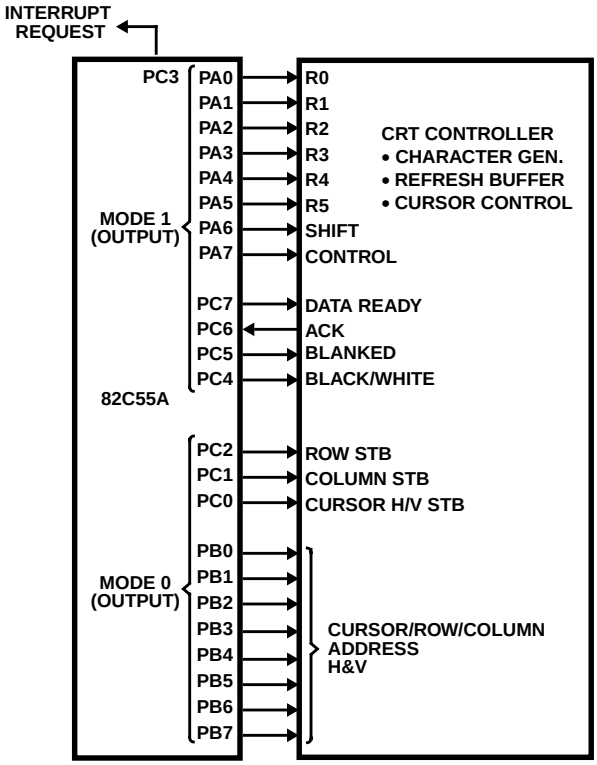


FIGURE 22. BASIC CRT CONTROLLER INTERFACE

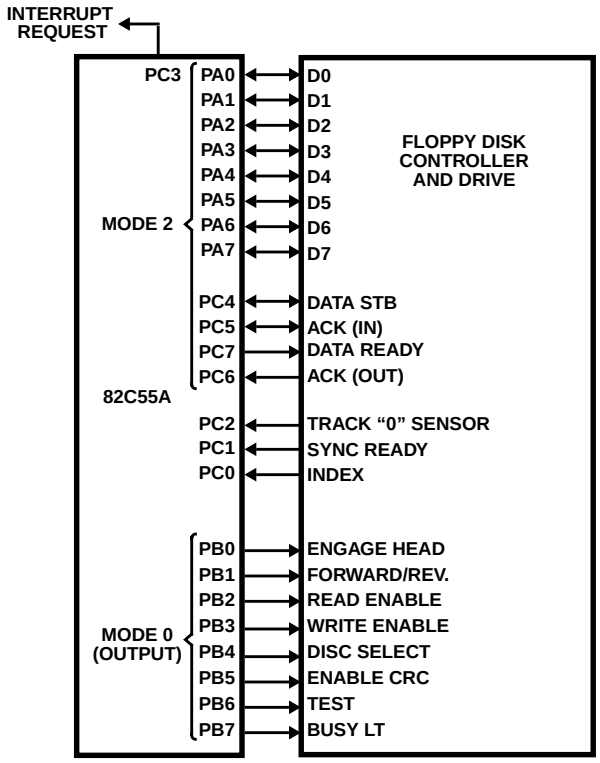


FIGURE 23. BASIC FLOPPY DISC INTERFACE

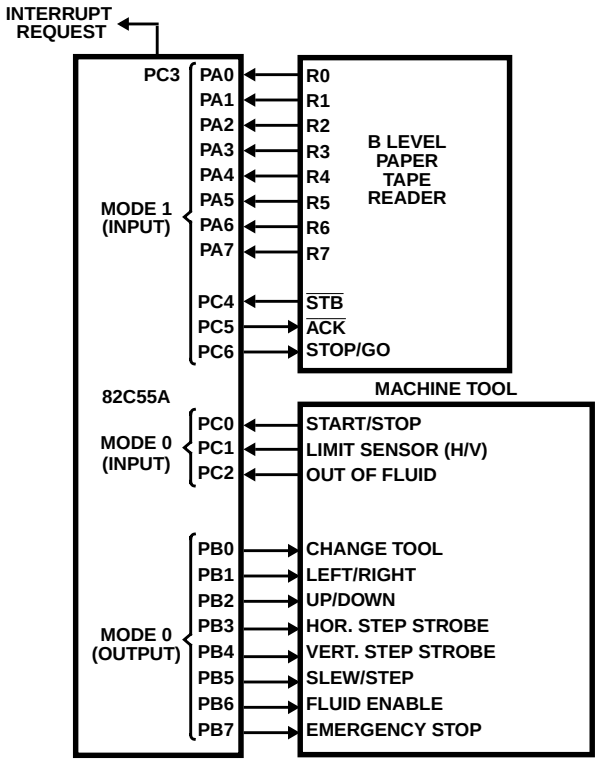


FIGURE 24. MACHINE TOOL CONTROLLER INTERFACE

Absolute Maximum Ratings $T_A = 25^oC$

Supply Voltage . . . . . +8.0V
Input, Output or I/O Voltage . . . . . GND-0.5V to V_{CC}+0.5V
ESD Classification . . . . . Class 1

Operating Conditions

Voltage Range . . . . . +4.5V to 5.5V
Operating Temperature Range
C82C55A . . . . . 0^oC to 70^oC
I82C55A . . . . . -40^oC to 85^oC
M82C55A . . . . . -55^oC to 125^oC

Thermal Information

Thermal Resistance (Typical, Note 1)	θ_{JA}	θ_{JC}
CERDIP Package	$50^oC/W$	$10^oC/W$
CLCC Package	$65^oC/W$	$14^oC/W$
PDIP Package	$50^oC/W$	N/A
PLCC Package	$46^oC/W$	N/A

Maximum Storage Temperature Range . . . . . -65^oC to 150^oC
Maximum Junction Temperature
CDIP Package . . . . . 175^oC
PDIP Package . . . . . 150^oC
Maximum Lead Temperature (Soldering 10s) . . . . . 300^oC
(PLCC Lead Tips Only)

Die Characteristics

Gate Count . . . . . 1000 Gates

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTE:

1. θ_{JA} is measured with the component mounted on an evaluation PC board in free air.

Electrical Specifications

V_{CC} = 5.0V ±10%; $T_A = 0^oC$ to $+70^oC$ (C82C55A);
$T_A = -40^oC$ to $+85^oC$ (I82C55A);
$T_A = -55^oC$ to $+125^oC$ (M82C55A)

SYMBOL	PARAMETER	LIMITS		UNITS	TEST CONDITIONS
		MIN	MAX		
V_{IH}	Logical One Input Voltage	2.0 2.2	-	V	I82C55A, C82C55A, M82C55A
V_{IL}	Logical Zero Input Voltage	-	0.8	V	
V_{OH}	Logical One Output Voltage	3.0 V_{CC} -0.4	-	V	I_{OH} = -2.5mA, I_{OH} = -100µA
V_{OL}	Logical Zero Output Voltage	-	0.4	V	I_{OL} +2.5mA
I_I	Input Leakage Current	-1.0	+1.0	µA	$V_{IN} = V_{CC}$ or GND, DIP Pins: 5, 6, 8, 9, 35, 36
IO	I/O Pin Leakage Current	-10	+10	µA	VO = V_{CC} or GND DIP Pins: 27 - 34
IBHH	Bus Hold High Current	-50	-400	µA	VO = 3.0V. Ports A, B, C
IBHL	Bus Hold Low Current	50	400	µA	VO = 1.0V. Port A ONLY
IDAR	Darlington Drive Current	-2.5	Note 2, 4	mA	Ports A, B, C. Test Condition 3
ICCSB	Standby Power Supply Current	-	10	µA	V_{CC} = 5.5V, $V_{IN} = V_{CC}$ or GND. Output Open
ICCOP	Operating Power Supply Current	-	1	mA/MHz	$T_A = +25^oC$, V_{CC} = 5.0V, Typical (See Note 3)

NOTES:

2. No internal current limiting exists on Port Outputs. A resistor must be added externally to limit the current.
3. ICCOP = 1mA/MHz of Peripheral Read/Write cycle time. (Example: 1.0µs I/O Read/Write cycle time = 1mA).
4. Tested as V_{OH} at -2.5mA.

Capacitance $T_A = 25^oC$

SYMBOL	PARAMETER	TYPICAL	UNITS	TEST CONDITIONS
CIN	Input Capacitance	10	pF	FREQ = 1MHz, All Measurements are referenced to device GND
CI/O	I/O Capacitance	20	pF	

AC Electrical Specifications V_{CC} = +5V± 10%, GND = 0V; T_A = -55°C to +125°C (M82C55A) (M82C55A-5);
T_A = -40°C to +85°C (I82C55A) (I82C55A-5);
T_A = 0°C to +70°C (C82C55A) (C82C55A-5)

SYMBOL	PARAMETER	82C55A-5		82C55A		UNITS	TEST CONDITIONS
		MIN	MAX	MIN	MAX		
READ TIMING							
(1) tAR	Address Stable Before $\overline{RD}$	0	-	0	-	ns	
(2) tRA	Address Stable After $\overline{RD}$	0	-	0	-	ns	
(3) tRR	$\overline{RD}$ Pulse Width	250	-	150	-	ns	
(4) tRD	Data Valid From $\overline{RD}$	-	200	-	120	ns	1
(5) tDF	Data Float After $\overline{RD}$	10	75	10	75	ns	2
(6) tRV	Time Between $\overline{RD}$s and/or $\overline{WR}$s	300	-	300	-	ns	
WRITE TIMING							
(7) tAW	Address Stable Before $\overline{WR}$	0	-	0	-	ns	
(8) tWA	Address Stable After $\overline{WR}$	20	-	20	-	ns	
(9) tWW	$\overline{WR}$ Pulse Width	100	-	100	-	ns	
(10) tDW	Data Valid to $\overline{WR}$ High	100	-	100	-	ns	
(11) tWD	Data Valid After $\overline{WR}$ High	30	-	30	-	ns	
OTHER TIMING							
(12) tWB	$\overline{WR}$ = 1 to Output	-	350	-	350	ns	1
(13) tIR	Peripheral Data Before $\overline{RD}$	0	-	0	-	ns	
(14) tHR	Peripheral Data After $\overline{RD}$	0	-	0	-	ns	
(15) tAK	ACK Pulse Width	200	-	200	-	ns	
(16) tST	STB Pulse Width	100	-	100	-	ns	
(17) tPS	Peripheral Data Before STB High	20	-	20	-	ns	
(18) tPH	Peripheral Data After STB High	50	-	50	-	ns	
(19) tAD	ACK = 0 to Output	-	175	-	175	ns	1
(20) tKD	ACK = 1 to Output Float	20	250	20	250	ns	2
(21) tWOB	$\overline{WR}$ = 1 to OBF = 0	-	150	-	150	ns	1
(22) tAOB	ACK = 0 to OBF = 1	-	150	-	150	ns	1
(23) tSIB	STB = 0 to IBF = 1	-	150	-	150	ns	1
(24) tRIB	$\overline{RD}$ = 1 to IBF = 0	-	150	-	150	ns	1
(25) tRIT	$\overline{RD}$ = 0 to INTR = 0	-	200	-	200	ns	1
(26) tSIT	STB = 1 to INTR = 1	-	150	-	150	ns	1
(27) tAIT	ACK = 1 to INTR = 1	-	150	-	150	ns	1
(28) tWIT	$\overline{WR}$ = 0 to INTR = 0	-	200	-	200	ns	1
(29) tRES	Reset Pulse Width	500	-	500	-	ns	1, (Note)

NOTE: Period of initial Reset pulse after power-on must be at least 50µsec. Subsequent Reset pulses may be 500ns minimum.

Timing Waveforms

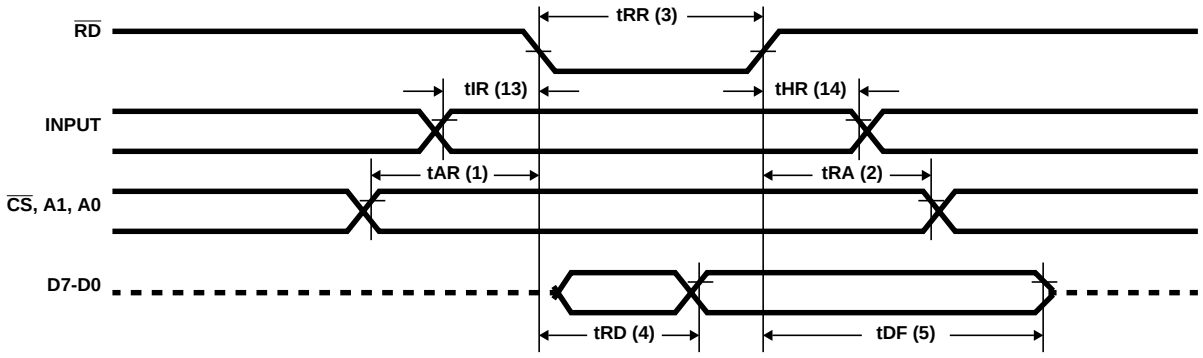


FIGURE 25. MODE 0 (BASIC INPUT)

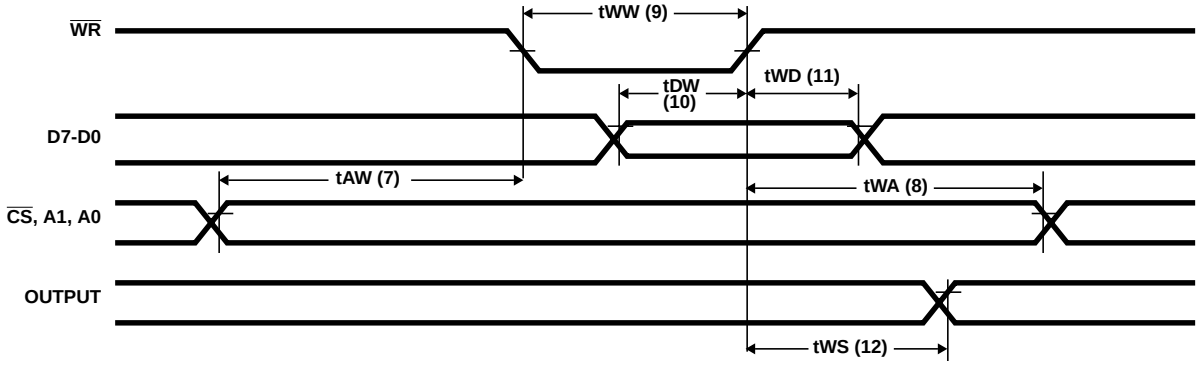


FIGURE 26. MODE 0 (BASIC OUTPUT)

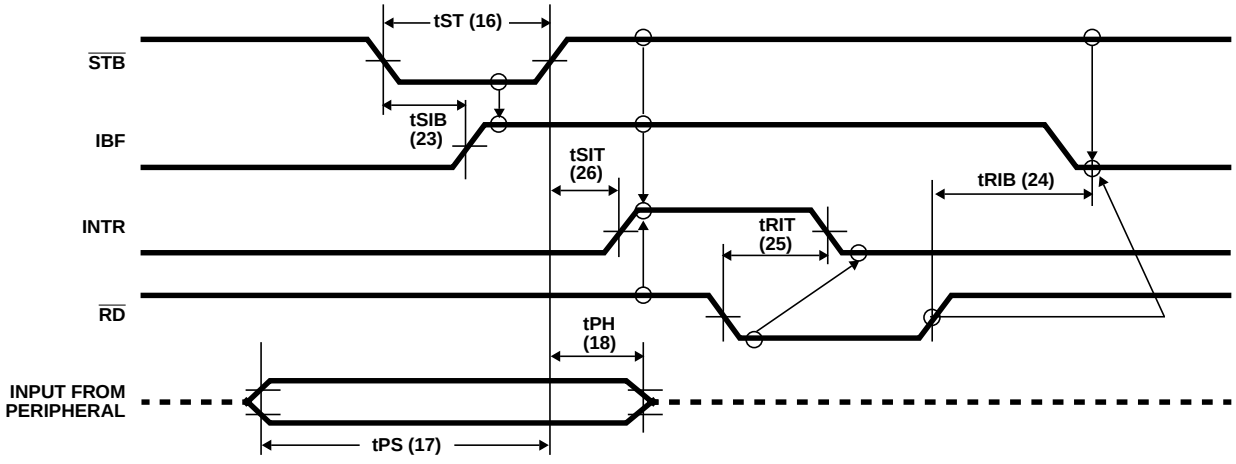


FIGURE 27. MODE 1 (STROBED INPUT)

Timing Waveforms (Continued)

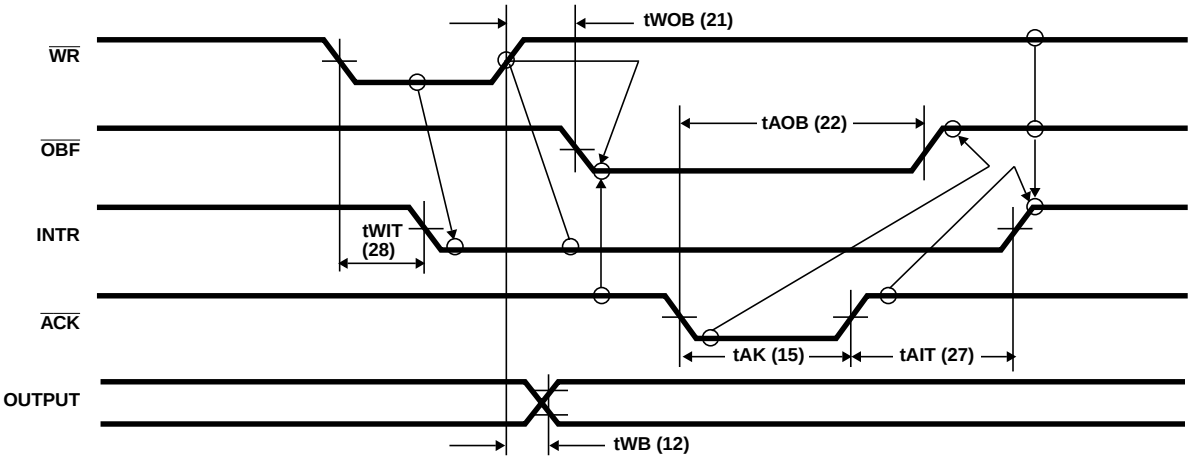


FIGURE 28. MODE 1 (STROBED OUTPUT)

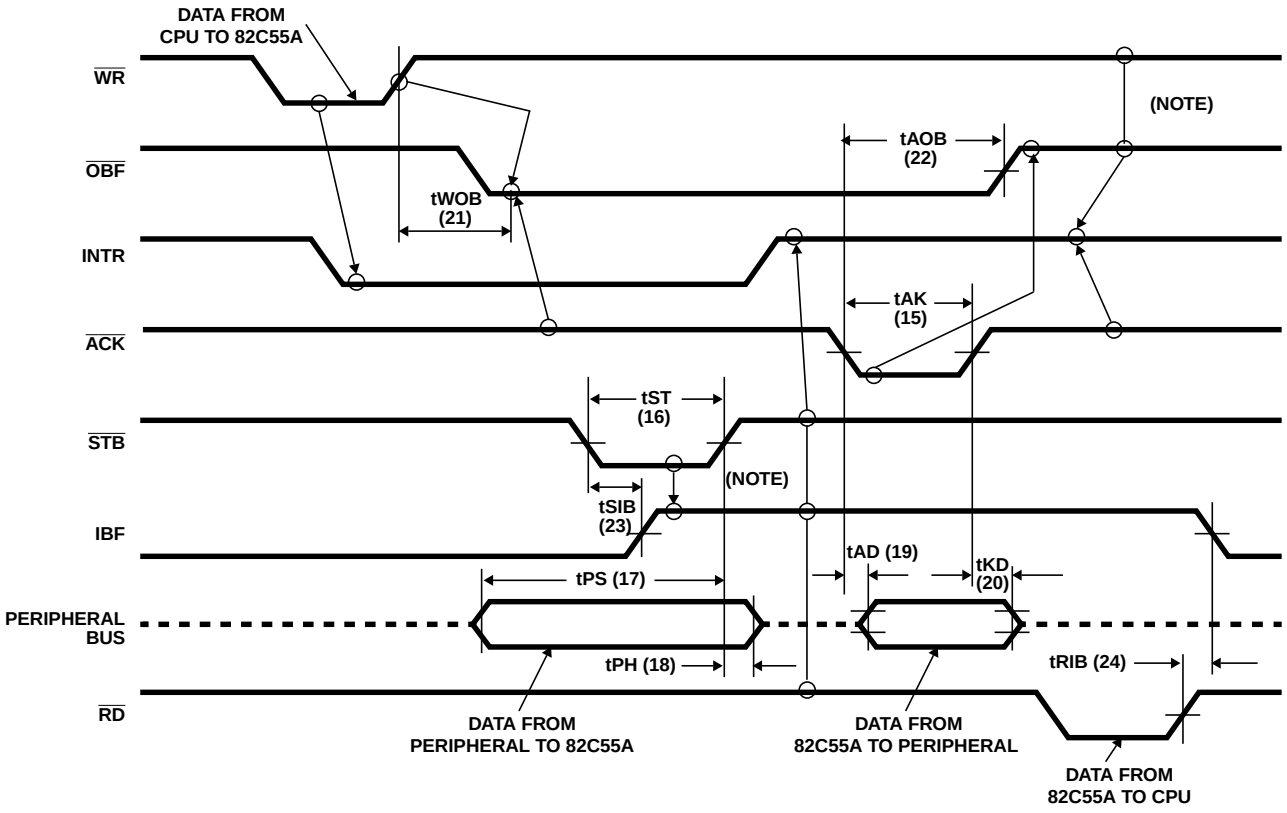


FIGURE 29. MODE 2 (BI-DIRECTIONAL)

NOTE: Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before $\overline{RD}$ is permissible. (INTR = IBF • $\overline{MASK}$ • $\overline{STB}$ • $\overline{RD}$ • $\overline{OBF}$ • $\overline{MASK}$ • $\overline{ACK}$ • $\overline{WR}$)

Timing Waveforms (Continued)

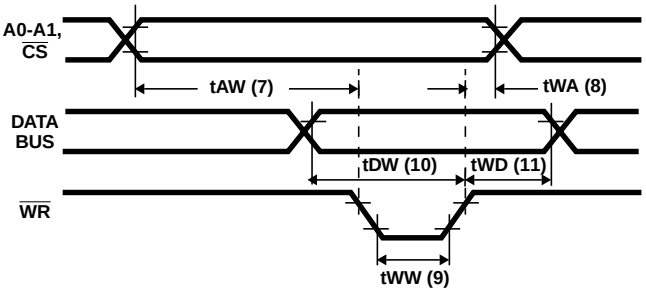


FIGURE 30. WRITE TIMING

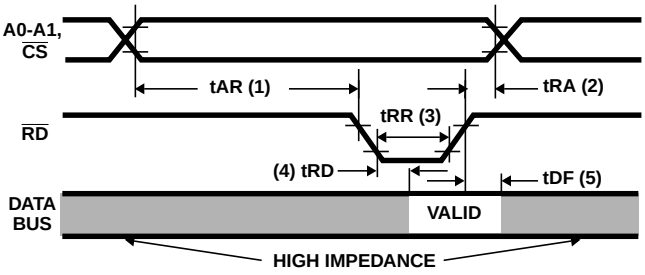
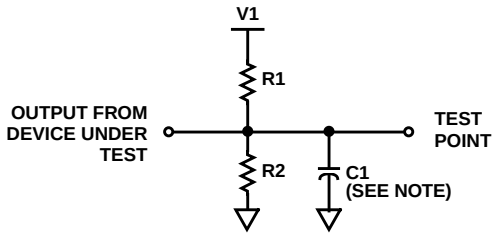


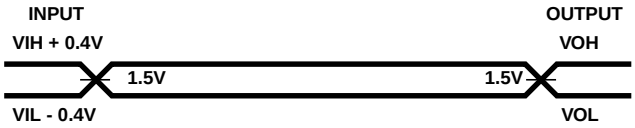
FIGURE 31. READ TIMING

AC Test Circuit



NOTE: Includes STRAY and JIG Capacitance

AC Testing Input, Output Waveforms



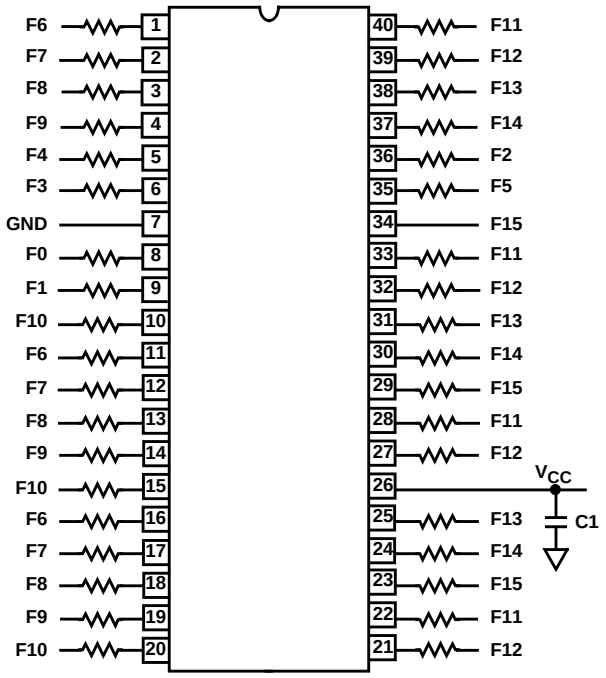
AC Testing: All AC Parameters tested as per test circuits. Input RISE and FALL times are driven at 1ns/V.

TEST CONDITION DEFINITION TABLE

TEST CONDITION	V1	R1	R2	C1
1	1.7V	523Ω	Open	150pF
2	V_{CC}	2kΩ	1.7kΩ	50pF
3	1.5V	750Ω	Open	50pF

Burn-In Circuits

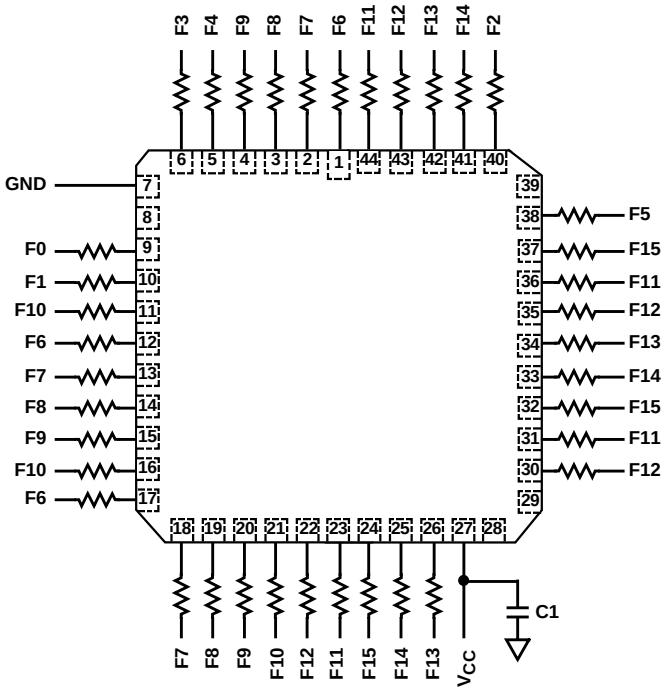
MD82C55A CERDIP



NOTES:

1. V_{CC} = 5.5V ± 0.5V
2. VIH = 4.5V ± 10%
3. VIL = -0.2V to 0.4V
4. GND = 0V

MR82C55A CLCC



NOTES:

1. C1 = 0.01µF minimum
2. All resistors are 47kΩ ± 5%
3. f0 = 100kHz ± 10%
4. f1 = f0 ÷ 2; f2 = f1 ÷ 2; . . . ; f15 = f14 ÷ 2

Die Characteristics

DIE DIMENSIONS:
95 x 100 x 19 ±1mils

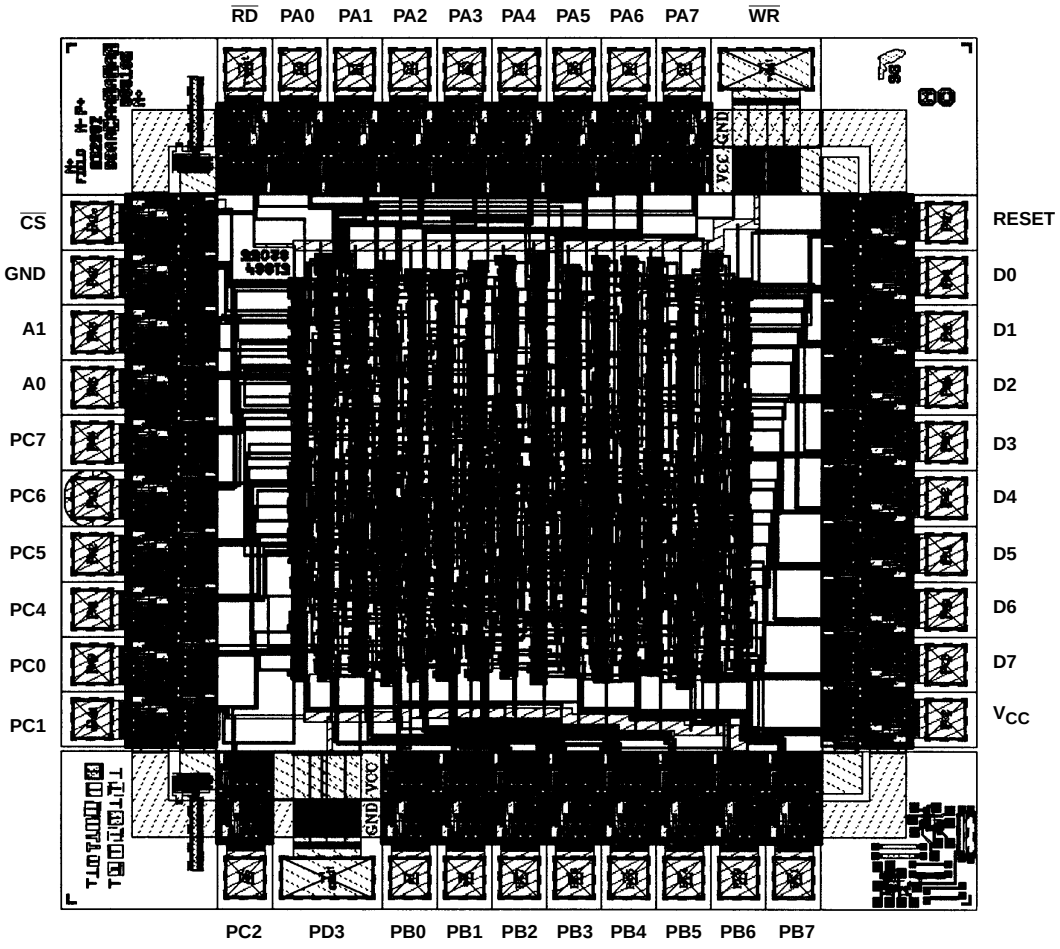
METALLIZATION:
Type: Silicon - Aluminum
Thickness: 11kÅ ±1kÅ

GLASSIVATION:
Type: SiO_2
Thickness: 8kÅ ±1kÅ

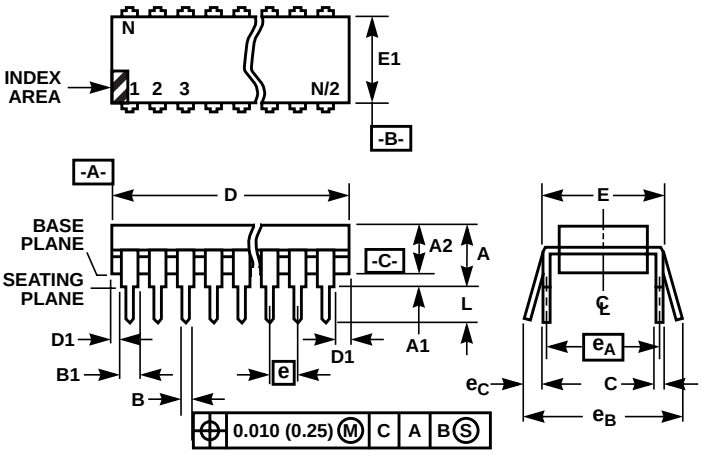
WORST CASE CURRENT DENSITY:
0.78×10^5 A/cm^2

Metallization Mask Layout

82C55A



Dual-In-Line Plastic Packages (PDIP)



NOTES:

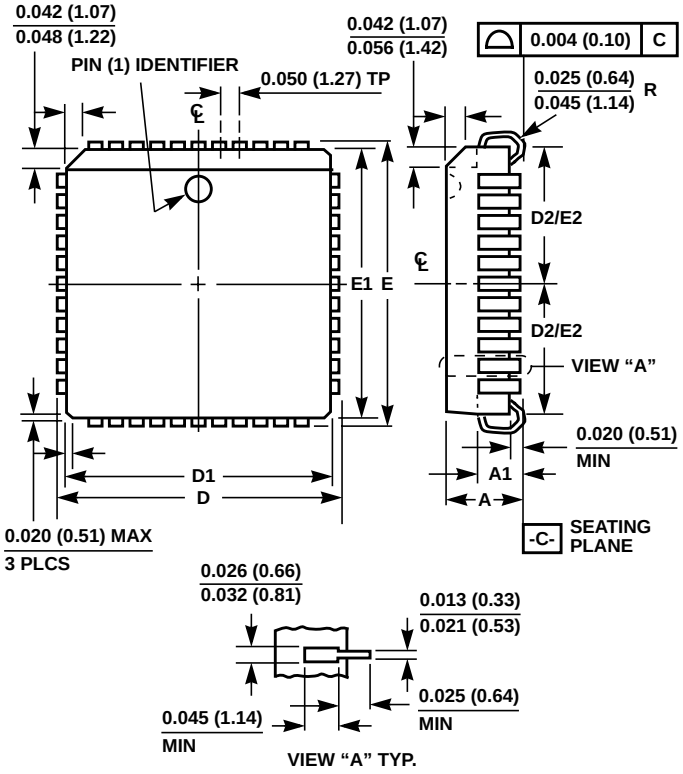
1. Controlling Dimensions: INCH. In case of conflict between English and Metric dimensions, the inch dimensions control.
2. Dimensioning and tolerancing per ANSI Y14.5M-1982.
3. Symbols are defined in the “MO Series Symbol List” in Section 2.2 of Publication No. 95.
4. Dimensions A, A1 and L are measured with the package seated in JEDEC seating plane gauge GS-3.
5. D, D1, and E1 dimensions do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.010 inch (0.25mm).
6. E and e_A are measured with the leads constrained to be perpendicular to datum -C-.
7. e_B and e_C are measured at the lead tips with the leads unconstrained. e_C must be zero or greater.
8. B1 maximum dimensions do not include dambar protrusions. Dambar protrusions shall not exceed 0.010 inch (0.25mm).
9. N is the maximum number of terminal positions.
10. Corner leads (1, N, N/2 and N/2 + 1) for E8.3, E16.3, E18.3, E28.3, E42.6 will have a B1 dimension of 0.030 - 0.045 inch (0.76 - 1.14mm).

E40.6 (JEDEC MS-011-AC ISSUE B)
40 LEAD DUAL-IN-LINE PLASTIC PACKAGE

SYMBOL	INCHES		MILLIMETERS		NOTES
	MIN	MAX	MIN	MAX	
A	-	0.250	-	6.35	4
A1	0.015	-	0.39	-	4
A2	0.125	0.195	3.18	4.95	-
B	0.014	0.022	0.356	0.558	-
B1	0.030	0.070	0.77	1.77	8
C	0.008	0.015	0.204	0.381	-
D	1.980	2.095	50.3	53.2	5
D1	0.005	-	0.13	-	5
E	0.600	0.625	15.24	15.87	6
E1	0.485	0.580	12.32	14.73	5
e	0.100 BSC		2.54 BSC		-
e_A	0.600 BSC		15.24 BSC		6
e_B	-	0.700	-	17.78	7
L	0.115	0.200	2.93	5.08	4
N	40		40		9

Rev. 0 12/93

Plastic Leaded Chip Carrier Packages (PLCC)



N44.65 **(JEDEC MS-018AC ISSUE A)**
44 LEAD PLASTIC LEADED CHIP CARRIER PACKAGE

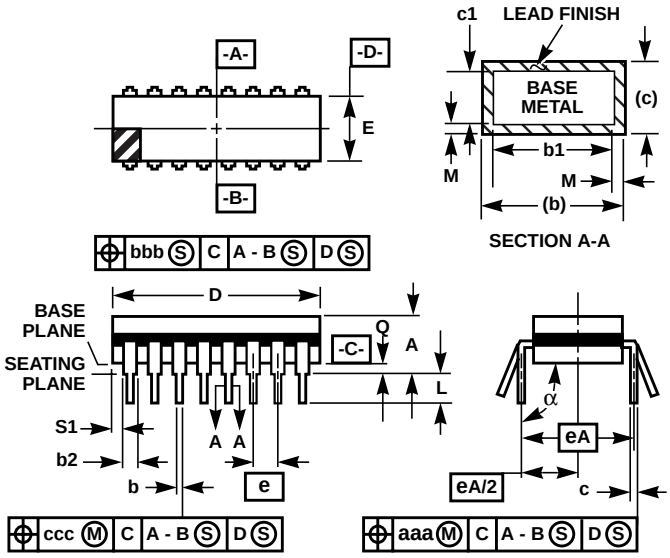
SYMBOL	INCHES		MILLIMETERS		NOTES
	MIN	MAX	MIN	MAX	
A	0.165	0.180	4.20	4.57	-
A1	0.090	0.120	2.29	3.04	-
D	0.685	0.695	17.40	17.65	-
D1	0.650	0.656	16.51	16.66	3
D2	0.291	0.319	7.40	8.10	4, 5
E	0.685	0.695	17.40	17.65	-
E1	0.650	0.656	16.51	16.66	3
E2	0.291	0.319	7.40	8.10	4, 5
N	44		44		6

Rev. 2 11/97

NOTES:

1. Controlling dimension: INCH. Converted millimeter dimensions are not necessarily exact.
2. Dimensions and tolerancing per ANSI Y14.5M-1982.
3. Dimensions D1 and E1 do not include mold protrusions. Allowable mold protrusion is 0.010 inch (0.25mm) per side. Dimensions D1 and E1 include mold mismatch and are measured at the extreme material condition at the body parting line.
4. To be measured at seating plane -C- contact point.
5. Centerline to be determined where center leads exit plastic body.
6. "N" is the number of terminal positions.

Ceramic Dual-In-Line Frit Seal Packages (CERDIP)



F40.6 MIL-STD-1835 GDIP1-T40 (D-5, CONFIGURATION A)
40 LEAD CERAMIC DUAL-IN-LINE FRIT SEAL PACKAGE

SYMBOL	INCHES		MILLIMETERS		NOTES
	MIN	MAX	MIN	MAX	
A	-	0.225	-	5.72	-
b	0.014	0.026	0.36	0.66	2
b1	0.014	0.023	0.36	0.58	3
b2	0.045	0.065	1.14	1.65	-
b3	0.023	0.045	0.58	1.14	4
c	0.008	0.018	0.20	0.46	2
c1	0.008	0.015	0.20	0.38	3
D	-	2.096	-	53.24	5
E	0.510	0.620	12.95	15.75	5
e	0.100 BSC		2.54 BSC		-
eA	0.600 BSC		15.24 BSC		-
eA/2	0.300 BSC		7.62 BSC		-
L	0.125	0.200	3.18	5.08	-
Q	0.015	0.070	0.38	1.78	6
S1	0.005	-	0.13	-	7
α	90^o	105^o	90^o	105^o	-
aaa	-	0.015	-	0.38	-
bbb	-	0.030	-	0.76	-
ccc	-	0.010	-	0.25	-
M	-	0.0015	-	0.038	2, 3
N	40		40		8

Rev. 0 4/94

NOTES:

1. Index area: A notch or a pin one identification mark shall be located adjacent to pin one and shall be located within the shaded area shown. The manufacturer's identification shall not be used as a pin one identification mark.
2. The maximum limits of lead dimensions b and c or M shall be measured at the centroid of the finished lead surfaces, when solder dip or tin plate lead finish is applied.
3. Dimensions b1 and c1 apply to lead base metal only. Dimension M applies to lead plating and finish thickness.
4. Corner leads (1, N, N/2, and N/2+1) may be configured with a partial lead paddle. For this configuration dimension b3 replaces dimension b2.
5. This dimension allows for off-center lid, meniscus, and glass overrun.
6. Dimension Q shall be measured from the seating plane to the base plane.
7. Measure dimension S1 at all four corners.
8. N is the maximum number of terminal positions.
9. Dimensioning and tolerancing per ANSI Y14.5M - 1982.
10. Controlling dimension: INCH.

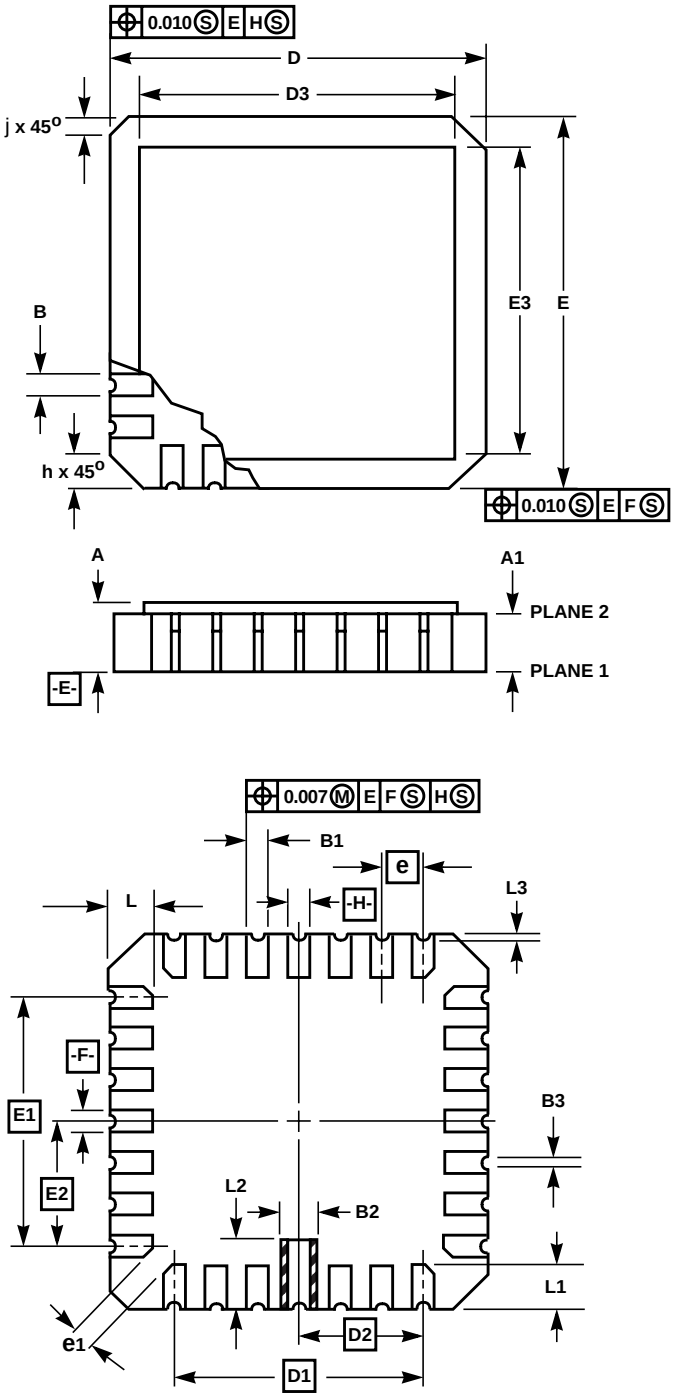
Sales Office Headquarters

NORTH AMERICA
Intersil Corporation
P. O. Box 883, Mail Stop 53-204
Melbourne, FL 32902
TEL: (407) 724-7000
FAX: (407) 724-7240

EUROPE
Intersil SA
Mercure Center
100, Rue de la Fusee
1130 Brussels, Belgium
TEL: (32) 2.724.2111
FAX: (32) 2.724.22.05

ASIA
Intersil (Taiwan) Ltd.
Taiwan Limited
7F-6, No. 101 Fu Hsing North Road
Taipei, Taiwan
Republic of China
TEL: (886) 2 2716 9310
FAX: (886) 2 2715 3029

Ceramic Leadless Chip Carrier Packages (CLCC)



J44.A **MIL-STD-1835 CQCC1-N44 (C-5)**
44 PAD CERAMIC LEADLESS CHIP CARRIER PACKAGE

SYMBOL	INCHES		MILLIMETERS		NOTES
	MIN	MAX	MIN	MAX	
A	0.064	0.120	1.63	3.05	6, 7
A1	0.054	0.088	1.37	2.24	-
B	0.033	0.039	0.84	0.99	4
B1	0.022	0.028	0.56	0.71	2, 4
B2	0.072 REF		1.83 REF		-
B3	0.006	0.022	0.15	0.56	-
D	0.640	0.662	16.26	16.81	-
D1	0.500 BSC		12.70 BSC		-
D2	0.250 BSC		6.35 BSC		-
D3	-	0.662	-	16.81	2
E	0.640	0.662	16.26	16.81	-
E1	0.500 BSC		12.70 BSC		-
E2	0.250 BSC		6.35 BSC		-
E3	-	0.662	-	16.81	2
e	0.050 BSC		1.27 BSC		-
e1	0.015	-	0.38	-	2
h	0.040 REF		1.02 REF		5
j	0.020 REF		0.51 REF		5
L	0.045	0.055	1.14	1.40	-
L1	0.045	0.055	1.14	1.40	-
L2	0.075	0.095	1.90	2.41	-
L3	0.003	0.015	0.08	0.38	-
ND	11		11		3
NE	11		11		3
N	44		44		3

Rev. 0 5/18/94

NOTES:

1. Metallized castellations shall be connected to plane 1 terminals and extend toward plane 2 across at least two layers of ceramic or completely across all of the ceramic layers to make electrical connection with the optional plane 2 terminals.
2. Unless otherwise specified, a minimum clearance of 0.015 inch (0.38mm) shall be maintained between all metallized features (e.g., lid, castellations, terminals, thermal pads, etc.)
3. Symbol “N” is the maximum number of terminals. Symbols “ND” and “NE” are the number of terminals along the sides of length “D” and “E”, respectively.
4. The required plane 1 terminals and optional plane 2 terminals (if used) shall be electrically connected.
5. The corner shape (square, notch, radius, etc.) may vary at the manufacturer’s option, from that shown on the drawing.
6. Chip carriers shall be constructed of a minimum of two ceramic layers.
7. Dimension “A” controls the overall package thickness. The maximum “A” dimension is package height before being solder dipped.
8. Dimensioning and tolerancing per ANSI Y14.5M-1982.
9. Controlling dimension: INCH.

Intel 8086 Family Architecture

General Purpose Registers

AH/AL AX (EAX) Accumulator
BH/BL BX (EBX) Base
CH/CL CX (ECX) Counter
DH/DL DX (EDX) Data

(Exx) indicates 386+ 32 bit register

Segment Registers

CS Code Segment
DS Data Segment
SS Stack Segment
ES Extra Segment
(FS) 386 and newer
(GS) 386 and newer

Pointer Registers

SI (ESI) Source Index
DI (EDI) Destination Index
IP Instruction Pointer

Stack Registers

SP (ESP) Stack Pointer
BP (EBP) Base Pointer

Status Registers

FLAGS Status Flags (see FLAGS)

Special Registers (386+ only)

CR0	Control Register 0	DR0	Debug Register 0
CR2	Control Register 2	DR1	Debug Register 1
CR3	Control Register 3	DR2	Debug Register 2
		DR3	Debug Register 3
TR4	Test Register 4	DR6	Debug Register 6
TR5	Test Register 5	DR7	Debug Register 7
TR6	Test Register 6		
TR7	Test Register 7		

Register	Default Segment	Valid Overrides
BP	SS	DS, ES, CS
SI or DI	DS	ES, SS, CS
DI strings	ES	None
SI strings	DS	ES, SS, CS

- see CPU DETECTING Instruction Timing

Instruction Clock Cycle Calculation

Some instructions require additional clock cycles due to a "Next Instruction Component" identified by a "+m" in the instruction clock cycle listings. This is due to the prefetch queue being purge on a control transfers. Below is the general rule for calculating "m":

88/86 not applicable
286 "m" is the number of bytes in the next instruction
386 "m" is the number of components in the next instruction
(the instruction coding (each byte), plus the data and the displacement are all considered components)

8088/8086 Effective Address (EA) Calculation

Description	Clock Cycles

Displacement	6
Base or Index (BX,BP,SI,DI)	5
Displacement+(Base or Index)	9
Base+Index (BP+DI,BX+SI)	7
Base+Index (BP+SI,BX+DI)	8
Base+Index+Displacement (BP+DI,BX+SI)	11
Base+Index+Displacement (BP+SI+disp,BX+DI+disp)	12

- add 4 cycles for word operands at odd addresses
- add 2 cycles for segment override
- 80188/80186 timings differ from those of the 8088/8086/80286

Task State Calculation

"TS" is defined as switching from VM/486 or 80286 TSS to one of the following:

	New Task				
Old Task	486 TSS (VM=0)	486 TSS (VM=1)	386 TSS (VM=0)	386 TSS (VM=1)	286 TSS
386 TSS (VM=0)			309	226	282
386 TSS (VM=1)			314	231	287
386 CPU/286 TSS			307	224	280
486 CPU/286 TSS	199	177			180

Miscellaneous

- all timings are for best case and do not take into account wait states, instruction alignment, the state of the prefetch queue, DMA refresh cycles, cache hits/misses or exception processing.
- to convert clocks to nanoseconds divide one microsecond by the processor speed in MegaHertz:

  (1000MHz/(n MHz)) = X nanoseconds

- see 8086 Architecture

FLAGS - Intel 8086 Family Flags Register

```
³11³10³F³E³D³C³B³A³9³8³7³6³5³4³3³2³1³0³
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  CF Carry Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  1
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  PF Parity Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  0
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  AF Auxiliary Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  0
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  ZF Zero Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  SF Sign Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  TF Trap Flag  (Single Step)
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  IF Interrupt Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  DF Direction Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  OF Overflow flag
```

```
³  ³ ³ ³ ÀÄÁÄÄÄ  IOPL I/O Privilege Level  (286+ only)
³  ³ ³ ÀÄÄÄÄÄ  NT Nested Task Flag  (286+ only)
³  ³ ÀÄÄÄÄÄ  0
³  ÀÄÄÄÄÄ  RF Resume Flag (386+ only)
ÀÄÄÄÄÄÄ  VM  Virtual Mode Flag (386+ only)
```

- see PUSHF POPF STI CLI STD CLD

MSW - Machine Status Word (286+ only)

```
³31³30-5³4³3³2³1³0³  Machine Status Word
  ³   ³  ³ ³ ³ ³ ÀÄÄÄÄ Protection Enable (PE)
  ³   ³  ³ ³ ³ ÀÄÄÄÄÄ Math Present (MP)
  ³   ³  ³ ³ ÀÄÄÄÄÄÄ Emulation (EM)
  ³   ³  ³ ÀÄÄÄÄÄÄÄ Task Switched (TS)
  ³   ³  ÀÄÄÄÄÄÄÄÄ Extension Type (ET)
  ³   ÀÄÄÄÄÄÄÄÄÄÄ Reserved
  ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄ Paging (PG)
```

Bit 0	PE	Protection Enable, switches processor between protected and real mode
Bit 1	MP	Math Present, controls function of the WAIT instruction
Bit 2	EM	Emulation, indicates whether coprocessor functions are to be emulated
Bit 3	TS	Task Switched, set and interrogated by coprocessor on task switches and when interpretting coprocessor instructions
Bit 4	ET	Extension Type, indicates type of coprocessor in system
Bits 5-30		Reserved
bit 31	PG	Paging, indicates whether the processor uses page tables to translate linear addresses to physical addresses

- see SMSW LMSW

8086/80186/80286/80386/80486 Instruction Set

AAA - Ascii Adjust for Addition

Usage: AAA
Modifies flags: AF CF (OF,PF,SF,ZF undefined)

Changes contents of AL to valid unpacked decimal. The high order nibble is zeroed.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	8	3	4	3	1

AAD - Ascii Adjust for Division

Usage: AAD
Modifies flags: SF ZF PF (AF,CF,OF undefined)

Used before dividing unpacked decimal numbers. Multiplies AH by 10 and the adds result into AL. Sets AH to zero. This instruction is also known to have an undocumented behavior.

```
        AL := 10*AH+AL
        AH := 0

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        none              60    14    19    14            2
```

AAM - Ascii Adjust for Multiplication

```
        Usage:  AAM
        Modifies flags: PF SF ZF (AF,CF,OF undefined)

        AH := AL / 10
        AL := AL mod 10

        Used after multiplication of two unpacked decimal numbers, this
        instruction adjusts an unpacked decimal number.  The high order
        nibble of each byte must be zeroed before using this instruction.
        This instruction is also known to have an undocumented behavior.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        none              83    16    17    15            2
```

AAS - Ascii Adjust for Subtraction

```
        Usage:  AAS
        Modifies flags: AF CF (OF,PF,SF,ZF undefined)

        Corrects result of a previous unpacked decimal subtraction in AL.
        High order nibble is zeroed.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        none               8     3     4     3            1
```

ADC - Add With Carry

```
        Usage:  ADC     dest,src
        Modifies flags: AF CF OF SF PF ZF

        Sums two binary operands placing the result in the destination.
        If CF is set, a 1 is added to the destination.

                               Clocks                   Size
        Operands         808x  286   386   486          Bytes

        reg,reg            3     2     2     1            2
        mem,reg          16+EA   7     7     3           2-4  (W88=24+EA)
        reg,mem           9+EA   7     6     2           2-4  (W88=13+EA)
        reg,immed          4     3     2     1           3-4
        mem,immed        17+EA   7     7     3           3-6  (W88=23+EA)
        accum,immed        4     3     2     1           2-3
```

ADD - Arithmetic Addition

Usage: ADD dest,src
Modifies flags: AF CF OF PF SF ZF

Adds "src" to "dest" and replacing the original contents of "dest". Both operands are binary.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	16+EA	7	7	3	2-4	(W88=24+EA)
reg,mem	9+EA	7	6	2	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem,immed	17+EA	7	7	3	3-6	(W88=23+EA)
accum,immed	4	3	2	1	2-3	

AND - Logical And

Usage: AND dest,src
Modifies flags: CF OF PF SF ZF (AF undefined)

Performs a logical AND of the two operands replacing the destination with the result.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	16+EA	7	7	3	2-4	(W88=24+EA)
reg,mem	9+EA	7	6	1	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem,immed	17+EA	7	7	3	3-6	(W88=23+EA)
accum,immed	4	3	2	1	2-3	

ARPL - Adjusted Requested Privilege Level of Selector (286+ PM)

Usage: ARPL dest,src
(286+ protected mode)
Modifies flags: ZF

Compares the RPL bits of "dest" against "src". If the RPL bits of "dest" are less than "src", the destination RPL bits are set equal to the source RPL bits and the Zero Flag is set. Otherwise the Zero Flag is cleared.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,reg	-	10	20	9	2
mem,reg	-	11	21	9	4

BOUND - Array Index Bound Check (80188+)

Usage: BOUND src,limit
Modifies flags: None

Array index in source register is checked against upper and lower bounds in memory source. The first word located at "limit" is

the lower boundary and the word at "limit+2" is the upper array bound.

Interrupt 5 occurs if the source value is less than or higher than the source.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,mem32	-	nj=13	nj=10	7	2
reg32,mem64	-	nj=13	nj=10	7	2

- nj = no jump taken

BSF - Bit Scan Forward (386+)

Usage: BSF dest,src
Modifies flags: ZF

Scans source operand for first bit set. Sets ZF if a bit is found set and loads the destination with an index to first set bit. Clears ZF is no bits are found set. BSF scans forward across bit pattern (0-n) while BSR scans in reverse (n-0).

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,reg	-	-	10+3n	6-42	3
reg,mem	-	-	10+3n	7-43	3-7
reg32,reg32	-	-	10+3n	6-42	3-7
reg32,mem32	-	-	10+3n	7-43	3-7

BSR - Bit Scan Reverse (386+)

Usage: BSR dest,src
Modifies flags: ZF

Scans source operand for first bit set. Sets ZF if a bit is found set and loads the destination with an index to first set bit. Clears ZF is no bits are found set. BSF scans forward across bit pattern (0-n) while BSR scans in reverse (n-0).

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,reg	-	-	10+3n	6-103	3
reg,mem	-	-	10+3n	7-104	3-7
reg32,reg32	-	-	10+3n	6-103	3-7
reg32,mem32	-	-	10+3n	7-104	3-7

BSWAP - Byte Swap (486+)

Usage: BSWAP reg32
Modifies flags: none

Changes the byte order of a 32 bit register from big endian to little endian or vice versa. Result left in destination register is undefined if the operand is a 16 bit register.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg32	-	-	-	1	2

BT - Bit Test (386+)

Usage: BT dest,src
Modifies flags: CF

The destination bit indexed by the source value is copied into the Carry Flag.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,immed8	-	-	3	3	4-8
mem16,immed8	-	-	6	6	4-8
reg16,reg16	-	-	3	3	3-7
mem16,reg16	-	-	12	12	3-7

BTC - Bit Test with Compliment (386+)

Usage: BTC dest,src
Modifies flags: CF

The destination bit indexed by the source value is copied into the Carry Flag after being complimented (inverted).

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,immed8	-	-	6	6	4-8
mem16,immed8	-	-	8	8	4-8
reg16,reg16	-	-	6	6	3-7
mem16,reg16	-	-	13	13	3-7

BTR - Bit Test with Reset (386+)

Usage: BTR dest,src
Modifies flags: CF

The destination bit indexed by the source value is copied into the Carry Flag and then cleared in the destination.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,immed8	-	-	6	6	4-8
mem16,immed8	-	-	8	8	4-8
reg16,reg16	-	-	6	6	3-7
mem16,reg16	-	-	13	13	3-7

BTS - Bit Test and Set (386+)

Usage: BTS dest,src
Modifies flags: CF

The destination bit indexed by the source value is copied into the Carry Flag and then set in the destination.

Operands	Clocks 808x	286	386	486	Size Bytes
reg16,immed8	-	-	6	6	4-8
mem16,immed8	-	-	8	8	4-8
reg16,reg16	-	-	6	6	3-7
mem16,reg16	-	-	13	13	3-7

CALL - Procedure Call

Usage: CALL destination
Modifies flags: None

Pushes Instruction Pointer (and Code Segment for far calls) onto stack and loads Instruction Pointer with the address of proc-name. Code continues with execution at CS:IP.

Operands	Clocks 808x	286	386	486
rel16 (near, IP relative)	19	7	7+m	3
rel32 (near, IP relative)	-	-	7+m	3
reg16 (near, register indirect)	16	7	7+m	5
reg32 (near, register indirect)	-	-	7+m	5
mem16 (near, memory indirect)	-	21+EA	11	10+m
5				
mem32 (near, memory indirect)	-	-	10+m	5
ptr16:16 (far, full ptr supplied)	28	13	17+m	18
ptr16:32 (far, full ptr supplied)	-	-	17+m	18
ptr16:16 (far, ptr supplied, prot. mode)	-	26	34+m	20
ptr16:32 (far, ptr supplied, prot. mode)	-	-	34+m	20
m16:16 (far, indirect)	37+EA	16	22+m	17
m16:32 (far, indirect)	-	-	22+m	17
m16:16 (far, indirect, prot. mode)	-	29	38+m	20
m16:32 (far, indirect, prot. mode)	-	-	38+m	20
ptr16:16 (task, via TSS or task gate)	-	177	TS	37+TS
m16:16 (task, via TSS or task gate)	-	180/185	5+TS	37+TS
m16:32 (task)	-	-	TS	37+TS
m16:32 (task)	-	-	5+TS	37+TS
ptr16:16 (gate, same privilege)	-	41	52+m	35
ptr16:32 (gate, same privilege)	-	-	52+m	35
m16:16 (gate, same privilege)	-	44	56+m	35
m16:32 (gate, same privilege)	-	-	56+m	35
ptr16:16 (gate, more priv, no parm)	-	82	86+m	69
ptr16:32 (gate, more priv, no parm)	-	-	86+m	69
m16:16 (gate, more priv, no parm)	-	83	90+m	69
m16:32 (gate, more priv, no parm)	-	-	90+m	69
ptr16:16 (gate, more priv, x parms)	-	86+4x	94+4x+m	77+4x
ptr16:32 (gate, more priv, x parms)	-	-	94+4x+m	77+4x
m16:16 (gate, more priv, x parms)	-	90+4x	98+4x+m	77+4x
m16:32 (gate, more priv, x parms)	-	-	98+4x+m	77+4x

CBW - Convert Byte to Word

Usage: CBW
Modifies flags: None

Converts byte in AL to word Value in AX by extending sign of AL throughout register AH.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	3	3	1

CDQ - Convert Double to Quad (386+)

Usage: CDQ
Modifies flags: None

Converts signed DWORD in EAX to a signed quad word in EDX:EAX by extending the high order bit of EAX throughout EDX

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	-	2	3	1

CLC - Clear Carry

Usage: CLC
Modifies flags: CF

Clears the Carry Flag.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2	2	1

CLD - Clear Direction Flag

Usage: CLD
Modifies flags: DF

Clears the Direction Flag causing string instructions to increment the SI and DI index registers.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2	2	1

CLI - Clear Interrupt Flag (disable)

Usage: CLI
Modifies flags: IF

Disables the maskable hardware interrupts by clearing the Interrupt flag. NMI's and software interrupts are not inhibited.

	Clocks				Size
Operands	808x	286	386	486	Bytes

none	2	2	3	5	1	

CLTS - Clear Task Switched Flag (286+ privileged)

Usage: CLTS
Modifies flags: None

Clears the Task Switched Flag in the Machine Status Register. This is a privileged operation and is generally used only by operating system code.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
none	-	2	5	7	2	

CMC - Complement Carry Flag

Usage: CMC
Modifies flags: CF

Toggles (inverts) the Carry Flag

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
none	2	2	2	2	1	

CMP - Compare

Usage: CMP dest,src
Modifies flags: AF CF OF PF SF ZF

Subtracts source from destination and updates the flags but does not save result. Flags can subsequently be checked for conditions.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	9+EA	7	5	2	2-4	(W88=13+EA)
reg,mem	9+EA	6	6	2	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem,immed	10+EA	6	5	2	3-6	(W88=14+EA)
accum,immed	4	3	2	1	2-3	

CMPS - Compare String (Byte, Word or Doubleword)

Usage: CMPS dest,src
CMPSB
CMPSW
CMPSD (386+)
Modifies flags: AF CF OF PF SF ZF

Subtracts destination value from source without saving results. Updates flags based on the subtraction and the index registers (E)SI and (E)DI are incremented or decremented depending on the state of the Direction Flag. CMPSB inc/decrements the index registers by 1, CMPSW inc/decrements by 2, while CMPSD increments or decrements by 4. The REP prefixes can be used to process

entire data items.

	Clocks				Size
Operands	808x	286	386	486	Bytes
dest,src	22	8	10	8	1 (W88=30)

CMPXCHG - Compare and Exchange

Usage: CMPXCHG dest,src (486+)
Modifies flags: AF CF OF PF SF ZF

Compares the accumulator (8-32 bits) with "dest". If equal the "dest" is loaded with "src", otherwise the accumulator is loaded with "dest".

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,reg	-	-	-	6	2
mem,reg	-	-	-	7	2

- add 3 clocks if the "mem,reg" comparison fails

CWD - Convert Word to Doubleword

Usage: CWD
Modifies flags: None

Extends sign of word in register AX throughout register DX forming a doubleword quantity in DX:AX.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	5	2	2	3	1

CWDE - Convert Word to Extended Doubleword (386+)

Usage: CWDE
Modifies flags: None

Converts a signed word in AX to a signed doubleword in EAX by extending the sign bit of AX throughout EAX.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	-	3	3	1

DAA - Decimal Adjust for Addition

Usage: DAA
Modifies flags: AF CF PF SF ZF (OF undefined)

Corrects result (in AL) of a previous BCD addition operation. Contents of AL are changed to a pair of packed decimal digits.

	Clocks				Size
Operands	808x	286	386	486	Bytes

```
        none              4     3     4     2            1


DAS - Decimal Adjust for Subtraction

        Usage:  DAS
        Modifies flags: AF CF PF SF ZF (OF undefined)

        Corrects result (in AL) of a previous BCD subtraction operation.
        Contents of AL are changed to a pair of packed decimal digits.

                                 Clocks                 Size
        Operands         808x  286   386   486          Bytes

        none              4     3     4     2            1


DEC - Decrement

        Usage:  DEC     dest
        Modifies flags: AF OF PF SF ZF

        Unsigned binary subtraction of one from the destination.

                                 Clocks                 Size
        Operands         808x  286   386   486          Bytes

        reg8              3     2     2     1            2
        mem             15+EA   7     6     3           2-4
        reg16/32          3     2     2     1            1


DIV - Divide

        Usage:  DIV     src
        Modifies flags: (AF,CF,OF,PF,SF,ZF undefined)

        Unsigned binary division of accumulator by source.  If the source
        divisor is a byte value then AX is divided by "src" and the
quotient
        is placed in AL and the remainder in AH.  If source operand is a
word
        value, then DX:AX is divided by "src" and the quotient is stored in
AX
        and the remainder in DX.

                                 Clocks                 Size
        Operands         808x  286   386   486          Bytes

        reg8             80-90  14    14    16           2
        reg16           144-162 22    22    24           2
        reg32              -     -     38    40           2
        mem8        (86-96)+EA  17    17    16          2-4
        mem16     (150-168)+EA  25    25    24          2-4  (W88=158-
176+EA)
        mem32              -     -     41    40          2-4

ENTER - Make Stack Frame  (80188+)

        Usage:  ENTER   locals,level
        Modifies flags: None
```

Modifies stack for entry to procedure for high level language. Operand "locals" specifies the amount of storage to be allocated on the stack. "Level" specifies the nesting level of the routine. Paired with the LEAVE instruction, this is an efficient method of entry and exit to procedures.

	Clocks				Size
Operands	808x	286	386	486	Bytes
immed16,0	-	11	10	14	4
immed16,1	-	15	12	17	4
immed16,immed8	-	12+4(n-1)	15+4(n-1)	17+3n	4

ESC - Escape

Usage: ESC immed,src
Modifies flags: None

Provides access to the data bus for other resident processors. The CPU treats it as a NOP but places memory operand on bus.

	Clocks				Size
Operands	808x	286	386	486	Bytes
immed,reg	2	9-20	?		2
immed,mem	2	9-20	?		2-4

HLT - Halt CPU

Usage: HLT
Modifies flags: None

Halts CPU until RESET line is activated, NMI or maskable interrupt received. The CPU becomes dormant but retains the current CS:IP for later restart.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	5	4	1

IDIV - Signed Integer Division

Usage: IDIV src
Modifies flags: (AF,CF,OF,PF,SF,ZF undefined)

Signed binary division of accumulator by source. If source is a byte value, AX is divided by "src" and the quotient is stored in AL and the remainder in AH. If source is a word value, DX:AX is divided by "src", and the quotient is stored in AL and the remainder in DX.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg8	101-112	17	19	19	2	
reg16	165-184	25	27	27	2	
reg32	-	-	43	43	2	
mem8	(107-118)+EA	20	22	20	2-4	
mem16	(171-190)+EA	38	30	28	2-4	(W88=175-194)
mem32	-	-	46	44	2-4	

IMUL - Signed Multiply

```
        Usage:  IMUL    src
                IMUL    src,immed        (286+)
                IMUL    dest,src,immed8  (286+)
                IMUL    dest,src         (386+)
        Modifies flags: CF OF (AF,PF,SF,ZF undefined)
```

Signed multiplication of accumulator by "src" with result placed in the accumulator. If the source operand is a byte value, it is multiplied by AL and the result stored in AX. If the source operand is a word value it is multiplied by AX and the result is stored in DX:AX. Other variations of this instruction allow specification of source and destination registers as well as a third immediate factor.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	80-98	13	9-14	13-18	2
reg16	128-154	21	9-22	13-26	2
reg32	-	-	9-38	12-42	2
mem8	86-104	16	12-17	13-18	2-4
mem16	134-160	24	12-25	13-26	2-4
mem32	-	-	12-41	13-42	2-4
reg16,reg16	-	-	9-22	13-26	3-5
reg32,reg32	-	-	9-38	13-42	3-5
reg16,mem16	-	-	12-25	13-26	3-5
reg32,mem32	-	-	12-41	13-42	3-5
reg16,immed	-	21	9-22	13-26	3
reg32,immed	-	21	9-38	13-42	3-6
reg16,reg16,immed	-	2	9-22	13-26	3-6
reg32,reg32,immed	-	21	9-38	13-42	3-6
reg16,mem16,immed	-	24	12-25	13-26	3-6
reg32,mem32,immed	-	24	12-41	13-42	3-6

IN - Input Byte or Word From Port

```
        Usage:  IN      accum,port
        Modifies flags: None
```

A byte, word or dword is read from "port" and placed in AL, AX or EAX respectively. If the port number is in the range of 0-255 it can be specified as an immediate, otherwise the port number must be specified in DX. Valid port ranges on the PC are 0-1024, though values through 65535 may be specified and recognized by third party vendors and PS/2's.

	Clocks				Size
Operands	808x	286	386	486	Bytes
accum,immed8	10/14	5	12	14	2
accum,immed8 (PM)			6/26	8/28/27	2
accum,DX	8/12	5	13	14	1
accum,DX (PM)			7/27	8/28/27	1

- 386+ protected mode timings depend on privilege levels.

  first number is the timing if: CPL ó IOPL
  second number is the timing if: CPL > IOPL or in VM 86 mode (386)
  CPL ò IOPL (486)

```
          third number is the timing when:    virtual mode on 486 processor
        - 486 virtual mode always requires 27 cycles

INC - Increment

        Usage:  INC     dest
        Modifies flags: AF OF PF SF ZF

        Adds one to destination unsigned binary operand.

                                  Clocks                  Size
        Operands         808x  286   386   486           Bytes

        reg8              3     2     2     1             2
        reg16             3     2     2     1             1
        reg32             3     2     2     1             1
        mem             15+EA   7     6     3             2-4  (W88=23+EA)


INS - Input String from Port  (80188+)

        Usage:  INS     dest,port
                INSB
                INSW
                INSD    (386+)
        Modifies flags: None

        Loads data from port to the destination ES:(E)DI  (even if a
        destination operand is supplied).  (E)DI is adjusted by the size
        of the operand and increased if the Direction Flag is cleared and
        decreased if the Direction Flag is set.  For INSB, INSW, INSD no
        operands are allowed and the size is determined by the mnemonic.

                                  Clocks                  Size
        Operands         808x  286   386    486          Bytes

        dest,port         -     5     15     17            1
        dest,port (PM)    -     5    9/29 10/32/30         1
        none              -     5     15     17            1
        none (PM)         -     5    9/29 10/32/30         1

        - 386+ protected mode timings depend on privilege levels.

          first number is the timing if:    CPL ó IOPL
          second number is the timing if:   CPL > IOPL
          third number is the timing if:    virtual mode on 486 processor


INT - Interrupt

        Usage:  INT     num
        Modifies flags: TF IF

        Initiates a software interrupt by pushing the flags, clearing the
        Trap and Interrupt Flags, pushing CS followed by IP and loading
        CS:IP with the value found in the interrupt vector table.
Execution
        then begins at the location addressed by the new CS:IP

                                              Clocks           Size
            Operands                    808x  286   386   486  Bytes

 3 (constant)                          52/72  23+m   33    26    2
```

3 (prot. mode, same priv.)	-	40+m	59	44	2
3 (prot. mode, more priv.)	-	78+m	99	71	2
3 (from VM86 to PL 0)	-	-	119	82	2
3 (prot. mode via task gate)	-	167+m	TS	37+TS	2
immed8	51/71	23+m	37	30	1
immed8 (prot. mode, same priv.)	-	40+m	59	44	1
immed8 (prot. mode, more priv.)	-	78+m	99	71	1
immed8 (from VM86 to PL 0)	-	-	119	86	1
immed8 (prot. mode, via task gate)	-	167+m	TS	37+TS	1

INTO - Interrupt on Overflow

Usage: INTO
Modifies flags: IF TF

If the Overflow Flag is set this instruction generates an INT 4 which causes the code addressed by 0000:0010 to be executed.

		Clocks			Size
Operands	808x	286	386	486	Bytes
none: jump	53/73	24+m	35	28	1
no jump	4	3	3	3	
(prot. mode, same priv.)	-	-	59	46	1
(prot. mode, more priv.)	-	-	99	73	1
(from VM86 to PL 0)	-	-	119	84	1
(prot. mode, via task gate)	-	-	TS	39+TS	1

INVD - Invalidate Cache (486+)

Usage: INVD
Modifies flags: none

Flushes CPU internal cache. Issues special function bus cycle which indicates to flush external caches. Data in write-back external caches is lost.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	-	-	4	2

INVLPG - Invalidate Translation Look-Aside Buffer Entry (486+)

Usage: INVLPG
Modifies flags: none

Invalidates a single page table entry in the Translation Look-Aside Buffer. Intel warns that this instruction may be implemented differently on future processors.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	-	-	12	2

- timing is for TLB entry hit only.

IRET/IRETD - Interrupt Return

```
        Usage:  IRET
                IRETD  (386+)
        Modifies flags: AF CF DF IF PF SF TF ZF

        Returns control to point of interruption by popping IP, CS
        and then the Flags from the stack and continues execution at
        this location.  CPU exception interrupts will return to the
        instruction that cause the exception because the CS:IP placed
        on the stack during the interrupt is the address of the offending
        instruction.
```

Operands	808x	286	386	486	Size Bytes
iret	32/44	17+m	22	15	1
iret (prot. mode)	-	31+m	38	15	1
iret (to less privilege)	-	55+m	82	36	1
iret (different task, NT=1)	-	169+m	TS	TS+32	1
iretd	-	-	22/38	15	1
iretd (to less privilege)	-	-	82	36	1
iretd (to VM86 mode)	-	-	60	15	1
iretd (different task, NT=1)	-	-	TS	TS+32	1

- 386 timings are listed as real-mode/protected-mode

Jxx - Jump Instructions Table

Mnemonic	Meaning	Jump Condition
JA	Jump if Above	CF=0 and ZF=0
JAE	Jump if Above or Equal	CF=0
JB	Jump if Below	CF=1
JBE	Jump if Below or Equal	CF=1 or ZF=1
JC	Jump if Carry	CF=1
JCXZ	Jump if CX Zero	CX=0
JE	Jump if Equal	ZF=1
JG	Jump if Greater (signed)	ZF=0 and SF=OF
JGE	Jump if Greater or Equal (signed)	SF=OF
JL	Jump if Less (signed)	SF != OF
JLE	Jump if Less or Equal (signed)	ZF=1 or SF != OF
JMP	Unconditional Jump	unconditional
JNA	Jump if Not Above	CF=1 or ZF=1
JNAE	Jump if Not Above or Equal	CF=1
JNB	Jump if Not Below	CF=0
JNBE	Jump if Not Below or Equal	CF=0 and ZF=0
JNC	Jump if Not Carry	CF=0
JNE	Jump if Not Equal	ZF=0
JNG	Jump if Not Greater (signed)	ZF=1 or SF != OF
JNGE	Jump if Not Greater or Equal (signed)	SF != OF
JNL	Jump if Not Less (signed)	SF=OF
JNLE	Jump if Not Less or Equal (signed)	ZF=0 and SF=OF
JNO	Jump if Not Overflow (signed)	OF=0
JNP	Jump if No Parity	PF=0
JNS	Jump if Not Signed (signed)	SF=0
JNZ	Jump if Not Zero	ZF=0
JO	Jump if Overflow (signed)	OF=1
JP	Jump if Parity	PF=1
JPE	Jump if Parity Even	PF=1
JPO	Jump if Parity Odd	PF=0
JS	Jump if Signed (signed)	SF=1
JZ	Jump if Zero	ZF=1

	Clocks	Size

Operands	808x	286	386	486	Bytes
Jx: jump	16	7+m	7+m	3	2
no jump	4	3	3	1	
Jx near-label	-	-	7+m	3	4
no jump	-	-	3	1	

- It's a good programming practice to organize code so the expected case is executed without a jump since the actual jump takes longer to execute than falling through the test.
- see JCXZ and JMP for their respective timings

JCXZ/JECXZ - Jump if Register (E)CX is Zero

Usage: JCXZ label
JECXZ label (386+)
Modifies flags: None

Causes execution to branch to "label" if register CX is zero. Uses unsigned comparision.

		Clocks			Size
Operands	808x	286	386	486	Bytes
label: jump	18	8+m	9+m	8	2
no jump	6	4	5	5	

JMP - Unconditional Jump

Usage: JMP target
Modifies flags: None

Unconditionally transfers control to "label". Jumps by default are within -32768 to 32767 bytes from the instruction following the jump. NEAR and SHORT jumps cause the IP to be updated while FAR jumps cause CS and IP to be updated.

	Clocks			
Operands	808x	286	386	486
rel8 (relative)	15	7+m	7+m	3
rel16 (relative)	15	7+m	7+m	3
rel32 (relative)	-	-	7+m	3
reg16 (near, register indirect)	11	7+m	7+m	5
reg32 (near, register indirect)	-	-	7+m	5
mem16 (near, mem indirect)	18+EA	11+m	10+m	5
mem32 (near, mem indirect)	24+EA	15+m	10+m	5
ptr16:16 (far, dword immed)	-	-	12+m	17
ptr16:16 (far, PM dword immed)	-	-	27+m	19
ptr16:16 (call gate, same priv.)	-	38+m	45+m	32
ptr16:16 (via TSS)	-	175+m	TS	42+TS
ptr16:16 (via task gate)	-	180+m	TS	43+TS
mem16:16 (far, indirect)	-	-	43+m	13
mem16:16 (far, PM indirect)	-	-	31+m	18
mem16:16 (call gate, same priv.)	-	41+m	49+m	31
mem16:16 (via TSS)	-	178+m	5+TS	41+TS
mem16:16 (via task gate)	-	183+m	5+TS	42+TS
ptr16:32 (far, 6 byte immed)	-	-	12+m	13
ptr16:32 (far, PM 6 byte immed)	-	-	27+m	18
ptr16:32 (call gate, same priv.)	-	-	45+m	31
ptr16:32 (via TSS)	-	-	TS	42+TS

ptr16:32 (via task state)				-	-	TS 43+TS
m16:32 (far, address at dword)				-	-	43+m 13
m16:32 (far, address at dword)				-	-	31+m 18
m16:32 (call gate, same priv.)				-	-	49+m 31
m16:32 (via TSS)				-	-	5+TS 41+TS
m16:32 (via task state)				-	-	5+TS 42+TS

LAHF - Load Register AH From Flags

Usage: LAHF
Modifies flags: None

Copies bits 0-7 of the flags register into AH. This includes flags AF, CF, PF, SF and ZF other bits are undefined.

AH := SF ZF xx AF xx PF xx CF

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	4	2	2	3	1

LAR - Load Access Rights (286+ protected)

Usage: LAR dest,src
Modifies flags: ZF

The high byte of the of the destination register is overwritten by the value of the access rights byte and the low order byte is zeroed
depending on the selection in the source operand. The Zero Flag is set if the load operation is successful.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,reg16	-	14	15	11	3
reg32,reg32	-	-	15	11	3
reg16,mem16	-	16	16	11	3-7
reg32,mem32	-	-	16	11	3-7

LDS - Load Pointer Using DS

Usage: LDS dest,src
Modifies flags: None

Loads 32-bit pointer from memory source to destination register and DS. The offset is placed in the destination register and the segment is placed in DS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading
of far pointers from the stack and the interrupt vector table.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,mem32	16+EA	7	7	6	2-4
reg,mem (PM)	-	-	22	12	5-7

LEA - Load Effective Address

Usage: LEA dest,src
Modifies flags: None

Transfers offset address of "src" to the destination register.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,mem	2+EA	3	2	1	2-4

- the MOV instruction can often save clock cycles when used in place of LEA on 8088 processors

LEAVE - Restore Stack for Procedure Exit (80188+)

Usage: LEAVE
Modifies flags: None

Releases the local variables created by the previous ENTER instruction by restoring SP and BP to their condition before the procedure stack frame was initialized.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	5	4	5	1

LES - Load Pointer Using ES

Usage: LES dest,src
Modifies flags: None

Loads 32-bit pointer from memory source to destination register and ES. The offset is placed in the destination register and the segment is placed in ES. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,mem	16+EA	7	7	6	2-4	(W88=24+EA)
reg,mem (PM)	-	-	22	12	5-7	

LFS - Load Pointer Using FS (386+)

Usage: LFS dest,src
Modifies flags: None

Loads 32-bit pointer from memory source to destination register and FS. The offset is placed in the destination register and the segment is placed in FS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,mem	-	-	7	6	5-7
reg,mem (PM)	-	-	22	12	5-7

LGDT - Load Global Descriptor Table (286+ privileged)

Usage: LGDT src
Modifies flags: None

Loads a value from an operand into the Global Descriptor Table (GDT) register.

	Clocks				Size
Operands	808x	286	386	486	Bytes
mem64	-	11	11	11	5

LIDT - Load Interrupt Descriptor Table (286+ privileged)

Usage: LIDT src
Modifies flags: None

Loads a value from an operand into the Interrupt Descriptor Table (IDT) register.

	Clocks				Size
Operands	808x	286	386	486	Bytes
mem64	-	12	11	11	5

LGS - Load Pointer Using GS (386+)

Usage: LGS dest,src
Modifies flags: None

Loads 32-bit pointer from memory source to destination register and GS. The offset is placed in the destination register and the segment is placed in GS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,mem	-	-	7	6	5-7
reg,mem (PM)	-	-	22	12	5-7

LLDT - Load Local Descriptor Table (286+ privileged)

Usage: LLDT src
Modifies flags: None

Loads a value from an operand into the Local Descriptor Table Register (LDTR).

	Clocks	Size

Operands	808x	286	386	486	Bytes
reg16	-	17	20	11	3
mem16	-	19	24	11	5

LMSW - Load Machine Status Word (286+ privileged)

Usage: LMSW src
Modifies flags: None

Loads the Machine Status Word (MSW) from data found at "src"

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16	-	3	10	13	3
mem16	-	6	13	13	5

LOCK - Lock Bus

Usage: LOCK
LOCK: (386+ prefix)
Modifies flags: None

This instruction is a prefix that causes the CPU assert bus lock signal during the execution of the next instruction. Used to avoid two processors from updating the same data location. The 286 always asserts lock during an XCHG with memory operands. This should only be used to lock the bus prior to XCHG, MOV, IN and OUT instructions.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	0	0	1	1

LODS - Load String (Byte, Word or Double)

Usage: LODS src
LODSB
LODSW
LODSD (386+)
Modifies flags: None

Transfers string element addressed by DS:SI (even if an operand is supplied) to the accumulator. SI is incremented based on the size of the operand or based on the instruction used. If the Direction Flag is set SI is decremented, if the Direction Flag is clear SI is incremented. Use with REP prefixes.

	Clocks				Size
Operands	808x	286	386	486	Bytes
src	12/16	5	5	5	1

LOOP - Decrement CX and Loop if CX Not Zero

Usage: LOOP label
Modifies flags: None

Decrements CX by 1 and transfers control to "label" if CX is not Zero. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction

		Clocks				Size
Operands		808x	286	386	486	Bytes
label:	jump	18	8+m	11+m	6	2
	no jump	5	4	?	2	

LOOPE/LOOPZ - Loop While Equal / Loop While Zero

Usage: LOOPE label
LOOPZ label
Modifies flags: None

Decrements CX by 1 (without modifying the flags) and transfers control to "label" if CX != 0 and the Zero Flag is set. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction.

		Clocks				Size
Operands		808x	286	386	486	Bytes
label:	jump	18	8+m	11+m	9	2
	no jump	5	4	?	6	

LOOPNZ/LOOPNE - Loop While Not Zero / Loop While Not Equal

Usage: LOOPNZ label
LOOPNE label
Modifies flags: None

Decrements CX by 1 (without modifying the flags) and transfers control to "label" if CX != 0 and the Zero Flag is clear. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction.

		Clocks				Size
Operands		808x	286	386	486	Bytes
label:	jump	19	8+m	11+m	9	2
	no jump	5	4	?	6	

LSL - Load Segment Limit (286+ protected)

Usage: LSL dest,src
Modifies flags: ZF

Loads the segment limit of a selector into the destination register if the selector is valid and visible at the current privilege level.
If loading is successful the Zero Flag is set, otherwise it is cleared.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,reg16	-	14	20/25	10	3
reg32,reg32	-	-	20/25	10	3

```
        reg16,mem16       -     16  21/26   10           5
        reg32,mem32       -     -   21/26   10           5

        - 386 times are listed "byte granular" / "page granular"

LSS - Load Pointer Using SS (386+)

        Usage:  LSS     dest,src
        Modifies flags: None

        Loads 32-bit pointer from memory source to destination register
        and SS.  The offset is placed in the destination register and the
        segment is placed in SS.  To use this instruction the word at the
        lower memory address must contain the offset and the word at the
        higher address must contain the segment.  This simplifies the
loading
        of far pointers from the stack and the interrupt vector table.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg,mem           -     -     7     6           5-7
        reg,mem (PM)      -     -     22    12          5-7


LTR - Load Task Register (286+ privileged)

        Usage:  LTR     src
        Modifies flags: None

        Loads the current task register with the value specified in "src".

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg16             -     17    23    20           3
        mem16             -     19    27    20           5


MOV - Move Byte or Word

        Usage:  MOV     dest,src
        Modifies flags: None

        Copies byte or word from the source operand to the destination
        operand.  If the destination is SS interrupts are disabled except
        on early buggy 808x CPUs.  Some CPUs disable interrupts if the
        destination is any of the segment registers

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg,reg           2     2     2     1           2
        mem,reg          9+EA   3     2     1           2-4  (W88=13+EA)
        reg,mem          8+EA   5     4     1           2-4  (W88=12+EA)
        mem,immed       10+EA   3     2     1           3-6  (W88=14+EA)
        reg,immed         4     2     2     1           2-3
        mem,accum         10    3     2     1           3    (W88=14)
        accum,mem         10    5     4     1           3    (W88=14)
        segreg,reg16      2     2     2     3           2
        segreg,mem16     8+EA   5     5     9           2-4  (W88=12+EA)
        reg16,segreg      2     2     2     3           2
        mem16,segreg     9+EA   3     2     3           2-4  (W88=13+EA)
```

reg32,CR0/CR2/CR3	-	-	6	4	
CR0,reg32	-	-	10	16	
CR2,reg32	-	-	4	4	3
CR3,reg32	-	-	5	4	3
reg32,DR0/DR1/DR2/DR3	-		22	10	3
reg32,DR6/DR7	-	-	22	10	3
DR0/DR1/DR2/DR3,reg32	-	-	22	11	3
DR6/DR7,reg32	-	-	16	11	3
reg32,TR6/TR7	-	-	12	4	3
TR6/TR7,reg32	-	-	12	4	3
reg32,TR3				3	
TR3,reg32				6	

- when the 386 special registers are used all operands are 32 bits

MOVS - Move String (Byte or Word)

Usage: MOVS dest,src
MOVSB
MOVSW
MOVSD (386+)
Modifies flags: None

Copies data from addressed by DS:SI (even if operands are given) to the location ES:DI destination and updates SI and DI based on the size of the operand or instruction used. SI and DI are incremented when the Direction Flag is cleared and decremented when the Direction
Flag is Set. Use with REP prefixes.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
dest,src	18	5	7	7	1	(W88=26)

MOVSX - Move with Sign Extend (386+)

Usage: MOVSX dest,src
Modifies flags: None

Copies the value of the source operand to the destination register with the sign extended.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,reg	-	-	3	3	3
reg,mem	-	-	6	3	3-7

MOVZX - Move with Zero Extend (386+)

Usage: MOVZX dest,src
Modifies flags: None

Copies the value of the source operand to the destination register with the zeroes extended.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,reg	-	-	3	3	3

reg,mem	-	-	6	3	3-7

MUL - Unsigned Multiply

Usage: MUL src
Modifies flags: CF OF (AF,PF,SF,ZF undefined)

Unsigned multiply of the accumulator by the source. If "src" is a byte value, then AL is used as the other multiplicand and the result is placed in AX. If "src" is a word value, then AX is multiplied by "src" and DX:AX receives the result. If "src" is a double word value, then EAX is multiplied by "src" and EDX:EAX receives the result. The 386+ uses an early out algorithm which makes multiplying any size value in EAX as fast as in the 8 or 16 bit registers.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	70-77	13	9-14	13-18	2
reg16	118-113	21	9-22	13-26	2
reg32	-	-	9-38	13-42	2-4
mem8	(76-83)+EA	16	12-17	13-18	2-4
mem16	(124-139)+EA	24	12-25	13-26	2-4
mem32	-	-	12-21	13-42	2-4

NEG - Two's Complement Negation

Usage: NEG dest
Modifies flags: AF CF OF PF SF ZF

Subtracts the destination from 0 and saves the 2s complement of "dest" back into "dest".

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg	3	2	2	1	2	
mem	16+EA	7	6	3	2-4	(W88=24+EA)

NOP - No Operation (90h)

Usage: NOP
Modifies flags: None

This is a do nothing instruction. It results in occupation of both space and time and is most useful for patching code segments. (This is the original XCHG AL,AL instruction)

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	3	3	3	1	1

NOT - One's Compliment Negation (Logical NOT)

Usage: NOT dest
Modifies flags: None

Inverts the bits of the "dest" operand forming the 1s complement.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg	3	2	2	1	2	
mem	16+EA	7	6	3	2-4	(W88=24+EA)

OR - Inclusive Logical OR

Usage: OR dest,src
Modifies flags: CF OF PF SF ZF (AF undefined)

Logical inclusive OR of the two operands returning the result in the destination. Any bit set in either operand will be set in the destination.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	16+EA	7	7	3	2-4	(W88=24+EA)
reg,mem	9+EA	7	6	2	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem8,immed8	17+EA	7	7	3	3-6	
mem16,immed16	25+EA	7	7	3	3-6	
accum,immed	4	3	2	1	2-3	

OUT - Output Data to Port

Usage: OUT port,accum
Modifies flags: None

Transfers byte in AL,word in AX or dword in EAX to the specified hardware port address. If the port number is in the range of 0-255 it can be specified as an immediate. If greater than 255 then the port number must be specified in DX. Since the PC only decodes 10 bits of the port address, values over 1023 can only be decoded by third party vendor equipment and also map to the port range 0-1023.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
immed8,accum	10/14	3	10	16	2	
immed8,accum (PM)	-	-	4/24	11/31/29	2	
DX,accum	8/12	3	11	16	1	
DX,accum (PM)	-	-	5/25	10/30/29	1	

- 386+ protected mode timings depend on privilege levels.

  first number is the timing when: CPL ó IOPL
  second number is the timing when: CPL > IOPL
  third number is the timing when: virtual mode on 486 processor

OUTS - Output String to Port (80188+)

Usage: OUTS port,src
OUTSB
OUTSW
OUTSD (386+)
Modifies flags: None

Transfers a byte, word or doubleword from "src" to the hardware port specified in DX. For instructions with no operands the "src" is located at DS:SI and SI is incremented or decremented by the size of the operand or the size dictated by the instruction format. When the Direction Flag is set SI is decremented, when clear, SI is incremented. If the port number is in the range of 0-255 it can be specified as an immediate. If greater than 255 then the port number must be specified in DX. Since the PC only decodes 10 bits of the port address, values over 1023 can only be decoded by third party vendor equipment and also map to the port range 0-1023.

	Clocks				Size
Operands	808x	286	386	486	Bytes
port,src	-	5	14	17	1
port,src (PM)	-	-	8/28	10/32/30	1

- 386+ protected mode timings depend on privilege levels.

  first number is the timing when: CPL ó IOPL
  second number is the timing when: CPL > IOPL
  third number is the timing when: virtual mode on 486 processor

POP - Pop Word off Stack

Usage: POP dest
Modifies flags: None

Transfers word at the current stack top (SS:SP) to the destination then increments SP by two to point to the new stack top. CS is not a valid destination.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16	8	5	4	4	1
reg32	4	-	-	4	1
segreg	8	5	7	3	1
mem16	17+EA	5	5	6	2-4
mem32	5	-	-	6	2-4

POPA/POPAD - Pop All Registers onto Stack (80188+)

Usage: POPA
POPAD (386+)
Modifies flags: None

Pops the top 8 words off the stack into the 8 general purpose 16/32 bit registers. Registers are popped in the following order: (E)DI, (E)SI, (E)BP, (E)SP, (E)DX, (E)CX and (E)AX. The (E)SP value popped from the stack is actually discarded.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	19	24	9	1

POPF/POPFD - Pop Flags off Stack

```
        Usage:  POPF
                POPFD  (386+)
        Modifies flags: all flags

        Pops word/doubleword from stack into the Flags Register and then
        increments SP by 2 (for POPF) or 4 (for POPFD).

                                  Clocks                  Size
        Operands         808x  286   386   486           Bytes

        none             8/12   5     5     9             1  (W88=12)
        none  (PM)        -     -     5     6             1


PUSH - Push Word onto Stack

        Usage:  PUSH    src
                PUSH    immed   (80188+ only)
        Modifies flags: None

        Decrements SP by the size of the operand (two or four, byte values
        are sign extended) and transfers one word from source to the stack
        top (SS:SP).

                                  Clocks                  Size
        Operands         808x  286   386   486           Bytes

        reg16           11/15   3     2     1             1
        reg32             -     -     2     1             1
        mem16           16+EA   5     5     4            2-4  (W88=24+EA)
        mem32             -     -     5     4            2-4
        segreg          10/14   3     2     3             1
        immed             -     3     2     1            2-3


PUSHA/PUSHAD - Push All Registers onto Stack  (80188+)

        Usage:  PUSHA
                PUSHAD  (386+)
        Modifies flags: None

        Pushes all general purpose registers onto the stack in the
following
        order: (E)AX, (E)CX, (E)DX, (E)BX, (E)SP, (E)BP, (E)SI, (E)DI.  The
        value of SP is the value before the actual push of SP.

                                  Clocks                  Size
        Operands         808x  286   386   486           Bytes

        none              -     19    24    11            1

PUSHF/PUSHFD - Push Flags onto Stack

        Usage:  PUSHF
                PUSHFD  (386+)
        Modifies flags: None

        Transfers the Flags Register onto the stack.  PUSHF saves a 16 bit
        value while PUSHFD saves a 32 bit value.

                                  Clocks                  Size
        Operands         808x  286   386   486           Bytes
```

```
        none            10/14   3     4     4            1
        none  (PM)        -     -     4     3            1
```

RCL - Rotate Through Carry Left

```
        Usage:  RCL     dest,count
        Modifies flags: CF OF
```

```
        Rotates the bits in the destination to the left "count" times with
        all data pushed out the left side re-entering on the right.  The
        Carry Flag holds the last bit rotated out.

                                 Clocks                 Size
        Operands         808x  286   386   486          Bytes

        reg,1              2     2     9     3           2
        mem,1            15+EA   7     10    4           2-4   (W88=23+EA)
        reg,CL            8+4n  5+n    9    8-30         2
        mem,CL         20+EA+4n 8+n    10   9-31         2-4   (W88=28+EA+4n)
        reg,immed8         -    5+n     9   8-30         3
        mem,immed8         -    8+n    10   9-31         3-5
```

RCR - Rotate Through Carry Right

```
        Usage:  RCR     dest,count
        Modifies flags: CF OF
```

```
        Rotates the bits in the destination to the right "count" times with
        all data pushed out the right side re-entering on the left.  The
        Carry Flag holds the last bit rotated out.

                                 Clocks                 Size
        Operands         808x  286   386   486          Bytes

        reg,1              2     2     9     3           2
        mem,1            15+EA   7     10    4           2-4   (W88=23+EA)
        reg,CL            8+4n  5+n    9    8-30         2
        mem,CL         20+EA+4n 8+n    10   9-31         2-4
(W88=28+EA+4n)
        reg,immed8         -    5+n    9    8-30         3
        mem,immed8         -    8+n    10   9-31         3-5
```

REP - Repeat String Operation

```
        Usage:  REP
        Modifies flags: None

        Repeats execution of string instructions while CX != 0.  After
        each string operation, CX is decremented and the Zero Flag is
        tested.  The combination of a repeat prefix and a segment override
        on CPU's before the 386 may result in errors if an interrupt occurs
```

before CX=0. The following code shows code that is susceptible to this and how to avoid it:

```
 again:  rep movs  byte ptr ES:[DI],ES:[SI]   ; vulnerable instr.
             jcxz  next              ; continue if REP successful
             loop  again             ; interrupt goofed count
 next:
```

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2		1

REPE/REPZ - Repeat Equal / Repeat Zero

Usage: REPE
REPZ
Modifies flags: None

Repeats execution of string instructions while CX != 0 and the Zero Flag is set. CX is decremented and the Zero Flag tested after each string operation. The combination of a repeat prefix and a segment override on processors other than the 386 may result in errors if an interrupt occurs before CX=0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2		1

REPNE/REPNZ - Repeat Not Equal / Repeat Not Zero

Usage: REPNE
REPNZ
Modifies flags: None

Repeats execution of string instructions while CX != 0 and the Zero Flag is clear. CX is decremented and the Zero Flag tested after each string operation. The combination of a repeat prefix and a segment override on processors other than the 386 may result in errors if an interrupt occurs before CX=0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2		1

RET/RETF - Return From Procedure

Usage: RET nBytes
RETF nBytes
RETN nBytes
Modifies flags: None

Transfers control from a procedure back to the instruction address saved on the stack. "n bytes" is an optional number of bytes to release. Far returns pop the IP followed by the CS, while near returns pop only the IP register.

	Clocks				Size
Operands	808x	286	386	486	Bytes

```
        retn            16/20  11+m  10+m    5          1
        retn immed      20/24  11+m  10+m    5          3
        retf            26/34  15+m  18+m    13         1
        retf (PM, same priv.)   -    32+m    18         1
        retf (PM, lesser priv.) -      68    33         1
        retf immed      25/33  15+m  18+m    14         3
        retf immed (PM, same priv.)  32+m    17         1
        retf immed (PM, lesser priv.)  68    33         1
```

ROL - Rotate Left

```
        Usage:  ROL     dest,count
        Modifies flags: CF OF

        ÚÄ¿     ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
        ³C³<þÂÄþ³7 <ÄÄÄÄÄÄÄÄÄÄ 0³<Ä¿
        ÀÄÙ  ³  ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ  ³
             ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Rotates the bits in the destination to the left "count" times with all data pushed out the left side re-entering on the right. The Carry Flag will contain the value of the last bit rotated out.

```
                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg,1              2     2     3     3            2
        mem,1            15+EA   7     7     4           2-4  (W88=23+EA)
        reg,CL             8+4n  5+n    3     3           2
        mem,CL         20+EA+4n 8+n    7     4           2-4
(W88=28+EA+4n)
        reg,immed8         -     5+n    3     2           3
        mem,immed8         -     8+n    7     4          3-5
```

ROR - Rotate Right

```
        Usage:  ROR     dest,count
        Modifies flags: CF OF

           ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿     ÚÄ¿
        ÚÄ>³7 þÄÄÄÄÄÄÄÄÄ> 0³þÄÂÄ>³C³
        ³  ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ  ³  ÀÄÙ
        ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Rotates the bits in the destination to the right "count" times with all data pushed out the right side re-entering on the left. The Carry Flag will contain the value of the last bit rotated out.

```
                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg,1              2     2     3     3            2
        mem,1            15+EA   7     7     4           2-4  (W88=23+EA)
        reg,CL             8+4n  5+n    3     3           2
        mem,CL         20+EA+4n 8+n    7     4           2-4
(W88=28+EA+4n)
        reg,immed8         -     5+n    3     2           3
        mem,immed8         -     8+n    7     4          3-5
```

SAHF - Store AH Register into FLAGS

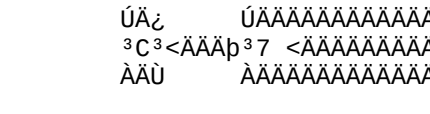
Usage: SAHF
Modifies flags: AF CF PF SF ZF

Transfers bits 0-7 of AH into the Flags Register. This includes AF, CF, PF, SF and ZF.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	4	2	3	2	1

SAL/SHL - Shift Arithmetic Left / Shift Logical Left

Usage: SAL dest,count
SHL dest,count
Modifies flags: CF OF PF SF ZF (AF undefined)

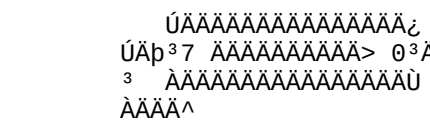


Shifts the destination left by "count" bits with zeroes shifted in on right. The Carry Flag contains the last bit shifted out.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,1	2	2	3	3	2	
mem,1	15+EA	7	7	4	2-4	(W88=23+EA)
reg,CL	8+4n	5+n	3	3	2	
mem,CL	20+EA+4n	8+n	7	4	2-4	(W88=28+EA+4n)
reg,immed8	-	5+n	3	2	3	
mem,immed8	-	8+n	7	4	3-5	

SAR - Shift Arithmetic Right

Usage: SAR dest,count
Modifies flags: CF OF PF SF ZF (AF undefined)



Shifts the destination right by "count" bits with the current sign bit replicated in the leftmost bit. The Carry Flag contains the last bit shifted out.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,1	2	2	3	3	2	
mem,1	15+EA	7	7	4	2-4	(W88=23+EA)
reg,CL	8+4n	5+n	3	3	2	
mem,CL	20+EA+4n	8+n	7	4	2-4	(W88=28+EA+4n)
reg,immed8	-	5+n	3	2	3	
mem,immed8	-	8+n	7	4	3-5	

SBB - Subtract with Borrow/Carry

Usage: SBB dest,src
Modifies flags: AF CF OF PF SF ZF

Subtracts the source from the destination, and subtracts 1 extra if the Carry Flag is set. Results are returned in "dest".

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	16+EA	7	6	3	2-4	(W88=24+EA)
reg,mem	9+EA	7	7	2	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem,immed	17+EA	7	7	3	3-6	(W88=25+EA)
accum,immed	4	3	2	1	2-3	

SCAS - Scan String (Byte, Word or Doubleword)

Usage: SCAS string
SCASB
SCASW
SCASD (386+)
Modifies flags: AF CF OF PF SF ZF

Compares value at ES:DI (even if operand is specified) from the accumulator and sets the flags similar to a subtraction. DI is incremented/decremented based on the instruction format (or operand size) and the state of the Direction Flag. Use with REP prefixes.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
string	15	7	7	6	1	(W88=19)

SETAE/SETNB - Set if Above or Equal / Set if Not Below (386+)

Usage: SETAE dest
SETNB dest
(unsigned, 386+)
Modifies flags: none

Sets the byte in the operand to 1 if the Carry Flag is clear otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETB/SETNAE - Set if Below / Set if Not Above or Equal (386+)

Usage: SETB dest
SETNAE dest
(unsigned, 386+)
Modifies flags: none

Sets the byte in the operand to 1 if the Carry Flag is set
otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETBE/SETNA - Set if Below or Equal / Set if Not Above (386+)

Usage: SETBE dest
SETNA dest
(unsigned, 386+)
Modifies flags: none

Sets the byte in the operand to 1 if the Carry Flag or the Zero
Flag is set, otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETE/SETZ - Set if Equal / Set if Zero (386+)

Usage: SETE dest
SETZ dest
Modifies flags: none

Sets the byte in the operand to 1 if the Zero Flag is set,
otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETNE/SETNZ - Set if Not Equal / Set if Not Zero (386+)

Usage: SETNE dest
SETNZ dest
Modifies flags: none

Sets the byte in the operand to 1 if the Zero Flag is clear,
otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETL/SETNGE - Set if Less / Set if Not Greater or Equal (386+)

Usage: SETL dest
SETNGE dest
(signed, 386+)

Modifies flags: none

Sets the byte in the operand to 1 if the Sign Flag is not equal to the Overflow Flag, otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETGE/SETNL - Set if Greater or Equal / Set if Not Less (386+)

Usage: SETGE dest
SETNL dest
(signed, 386+)
Modifies flags: none

Sets the byte in the operand to 1 if the Sign Flag equals the Overflow Flag, otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETLE/SETNG - Set if Less or Equal / Set if Not greater or Equal (386+)

Usage: SETLE dest
SETNG dest
(signed, 386+)
Modifies flags: none

Sets the byte in the operand to 1 if the Zero Flag is set or the Sign Flag is not equal to the Overflow Flag, otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETG/SETNLE - Set if Greater / Set if Not Less or Equal (386+)

Usage: SETG dest
SETNLE dest
(signed, 386+)
Modifies flags: none

Sets the byte in the operand to 1 if the Zero Flag is clear or the Sign Flag equals to the Overflow Flag, otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETS - Set if Signed (386+)

Usage: SETS dest
Modifies flags: none

Sets the byte in the operand to 1 if the Sign Flag is set, otherwise
sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETNS - Set if Not Signed (386+)

Usage: SETNS dest
Modifies flags: none

Sets the byte in the operand to 1 if the Sign Flag is clear,
otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETC - Set if Carry (386+)

Usage: SETC dest
Modifies flags: none

Sets the byte in the operand to 1 if the Carry Flag is set,
otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETNC - Set if Not Carry (386+)

Usage: SETNC dest
Modifies flags: none

Sets the byte in the operand to 1 if the Carry Flag is clear,
otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETO - Set if Overflow (386+)

```
        Usage:  SETO    dest
        Modifies flags: none

        Sets the byte in the operand to 1 if the Overflow Flag is set,
        otherwise sets the operand to 0.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg8              -     -     4     3            3
        mem8              -     -     5     4            3


SETNO - Set if Not Overflow (386+)

        Usage:  SETNO   dest
        Modifies flags: none

        Sets the byte in the operand to 1 if the Overflow Flag is clear,
        otherwise sets the operand to 0.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg8              -     -     4     3            3
        mem8              -     -     5     4            3

SETP/SETPE - Set if Parity / Set if Parity Even  (386+)

        Usage:  SETP    dest
                SETPE   dest
        Modifies flags: none

        Sets the byte in the operand to 1 if the Parity Flag is set,
        otherwise sets the operand to 0.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg8              -     -     4     3            3
        mem8              -     -     5     4            3


SETNP/SETPO - Set if No Parity / Set if Parity Odd (386+)

        Usage:  SETNP   dest
                SETPO   dest
        Modifies flags: none

        Sets the byte in the operand to 1 if the Parity Flag is clear,
        otherwise sets the operand to 0.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg8              -     -     4     3            3
        mem8              -     -     5     4            3


SGDT - Store Global Descriptor Table (286+ privileged)

        Usage:  SGDT    dest
        Modifies flags: none
```

Stores the Global Descriptor Table (GDT) Register into the specified operand.

	Clocks				Size
Operands	808x	286	386	486	Bytes
mem64	-	11	9	10	5

SIDT - Store Interrupt Descriptor Table (286+ privileged)

Usage: SIDT dest
Modifies flags: none

Stores the Interrupt Descriptor Table (IDT) Register into the specified operand.

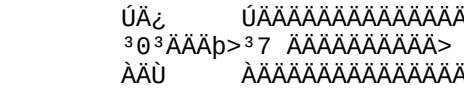
	Clocks				Size
Operands	808x	286	386	486	Bytes
mem64	-	12	9	10	5

SHL - Shift Logical Left

See: SAL

SHR - Shift Logical Right

Usage: SHR dest,count
Modifies flags: CF OF PF SF ZF (AF undefined)



Shifts the destination right by "count" bits with zeroes shifted in on the left. The Carry Flag contains the last bit shifted out.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,1	2	2	3		2	
mem,1	15+EA	7	7		2-4	(W88=23+EA)
reg,CL	8+4n	5+n	3		2	
mem,CL	20+EA+4n	8+n	7		2-4	(W88=28+EA+4n)
reg,immed8	-	5+n	3		3	
mem,immed8	-	8+n	7		3-5	

SHLD/SHRD - Double Precision Shift (386+)

Usage: SHLD dest,src,count
SHRD dest,src,count
Modifies flags: CF PF SF ZF (OF,AF undefined)

SHLD shifts "dest" to the left "count" times and the bit positions opened are filled with the most significant bits of "src". SHRD shifts "dest" to the right "count" times and the bit positions opened are filled with the least significant bits of the second operand. Only the 5 lower bits of "count" are used.

Operands	Clocks 808x	286	386	486	Size Bytes
reg16,reg16,immed8	-	-	3	2	4
reg32,reg32,immed8	-	-	3	2	4
mem16,reg16,immed8	-	-	7	3	6
mem32,reg32,immed8	-	-	7	3	6
reg16,reg16,CL	-	-	3	3	3
reg32,reg32,CL	-	-	3	3	3
mem16,reg16,CL	-	-	7	4	5
mem32,reg32,CL	-	-	7	4	5

SLDT - Store Local Descriptor Table (286+ privileged)

Usage: SLDT dest
Modifies flags: none

Stores the Local Descriptor Table (LDT) Register into the specified operand.

Operands	Clocks 808x	286	386	486	Size Bytes
reg16	-	2	2	2	3
mem16	-	2	2	3	5

SMSW - Store Machine Status Word (286+ privileged)

Usage: SMSW dest
Modifies flags: none

Store Machine Status Word (MSW) into "dest".

Operands	Clocks 808x	286	386	486	Size Bytes
reg16	-	2	10	2	3
mem16	-	3	3	3	5

STC - Set Carry

Usage: STC
Modifies flags: CF

Sets the Carry Flag to 1.

Operands	Clocks 808x	286	386	486	Size Bytes
none	2	2	2	2	1

STD - Set Direction Flag

Usage: STD
Modifies flags: DF

Sets the Direction Flag to 1 causing string instructions to auto-decrement SI and DI instead of auto-increment.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2	2	1

STI - Set Interrupt Flag (Enable Interrupts)

Usage: STI
Modifies flags: IF

Sets the Interrupt Flag to 1, which enables recognition of all hardware interrupts. If an interrupt is generated by a hardware device, an End of Interrupt (EOI) must also be issued to enable other hardware interrupts of the same or lower priority.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2	5	1

STOS - Store String (Byte, Word or Doubleword)

Usage: STOS dest
STOSB
STOSW
STOSD
Modifies flags: None

Stores value in accumulator to location at ES:(E)DI (even if operand is given). (E)DI is incremented/decremented based on the size of the operand (or instruction format) and the state of the Direction Flag. Use with REP prefixes.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
dest	11	3	4	5	1	(W88=15)

STR - Store Task Register (286+ privileged)

Usage: STR dest
Modifies flags: None

Stores the current Task Register to the specified operand.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16	-	2	2	2	3
mem16	-	3	2	3	5

SUB - Subtract

Usage: SUB dest,src
Modifies flags: AF CF OF PF SF ZF

The source is subtracted from the destination and the result is stored in the destination.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	16+EA	7	6	3	2-4	(W88=24+EA)
reg,mem	9+EA	7	7	2	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem,immed	17+EA	7	7	3	3-6	(W88=25+EA)
accum,immed	4	3	2	1	2-3	

TEST - Test For Bit Pattern

Usage: TEST dest,src
Modifies flags: CF OF PF SF ZF (AF undefined)

Performs a logical AND of the two operands updating the flags register without saving the result.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	1	1	2	
reg,mem	9+EA	6	5	1	2-4	(W88=13+EA)
mem,reg	9+EA	6	5	2	2-4	(W88=13+EA)
reg,immed	5	3	2	1	3-4	
mem,immed	11+EA	6	5	2	3-6	
accum,immed	4	3	2	1	2-3	

VERR - Verify Read (286+ protected)

Usage: VERR src
Modifies flags: ZF

Verifies the specified segment selector is valid and is readable at the current privilege level. If the segment is readable, the Zero Flag is set, otherwise it is cleared.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16	-	14	10	11	3
mem16	-	16	11	11	5

VERW - Verify Write (286+ protected)

Usage: VERW src
Modifies flags: ZF

Verifies the specified segment selector is valid and is ratable at the current privilege level. If the segment is writable, the Zero Flag is set, otherwise it is cleared.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16	-	14	15	11	3
mem16	-	16	16	11	5

WAIT/FWAIT - Event Wait

Usage: WAIT
FWAIT
Modifies flags: None

CPU enters wait state until the coprocessor signals it has finished its operation. This instruction is used to prevent the CPU from accessing memory that may be temporarily in use by the coprocessor. WAIT and FWAIT are identical.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	4	3	6+	1-3	1

WBINVD - Write-Back and Invalidate Cache (486+)

Usage: WBINVD
Modifies flags: None

Flushes internal cache, then signals the external cache to write back current data followed by a signal to flush the external cache.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	-	-	5	2

XCHG - Exchange

Usage: XCHG dest,src
Modifies flags: None

Exchanges contents of source and destination.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	4	3	3	3	2	
mem,reg	17+EA	5	5	5	2-4	(W88=25+EA)
reg,mem	17+EA	5	5	3	2-4	(W88=25+EA)
accum,reg	3	3	3	3	1	
reg,accum	3	3	3	3	1	

XLAT/XLATB - Translate

Usage: XLAT translation-table
XLATB (masm 5.x)
Modifies flags: None

Replaces the byte in AL with byte from a user table addressed by BX. The original value of AL is the index into the translate table.
The best way to discripe this is MOV AL,[BX+AL]

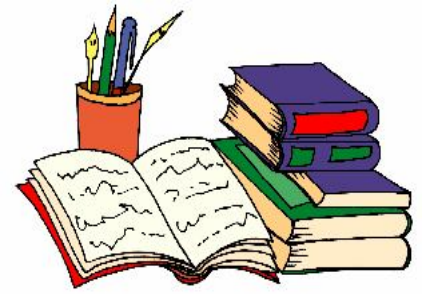
	Clocks				Size
Operands	808x	286	386	486	Bytes
table offset	11	5	5	4	1

XOR - Exclusive OR

Usage: XOR dest,src
Modifies flags: CF OF PF SF ZF (AF undefined)

Performs a bitwise exclusive OR of the operands and returns the result in the destination.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	16+EA	7	6	3	2-4	(W88=24+EA)
reg,mem	9+EA	7	7	2	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem,immed	17+EA	7	7	3	3-6	(W88=25+EA)
accum,immed	4	3	2	1	2-3	



Giáo trình
Vi xử lý - Vi điều khiển

MỤC LỤC

Danh mục hình vẽ

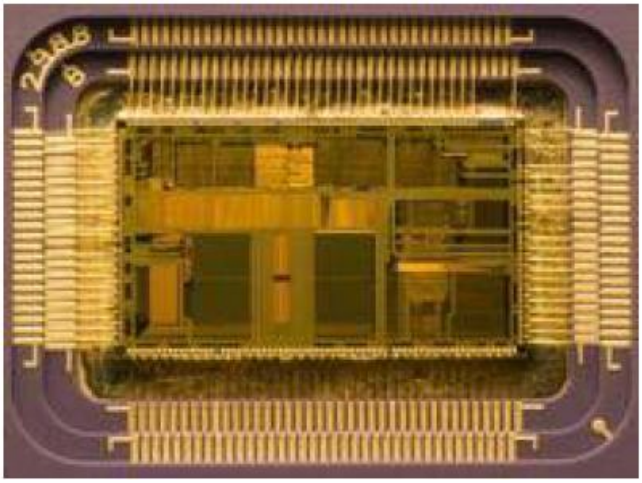
Danh mục bảng biểu

CHƯƠNG 1
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

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 hơn. 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. 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 lý đã 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.

1.1.3. 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

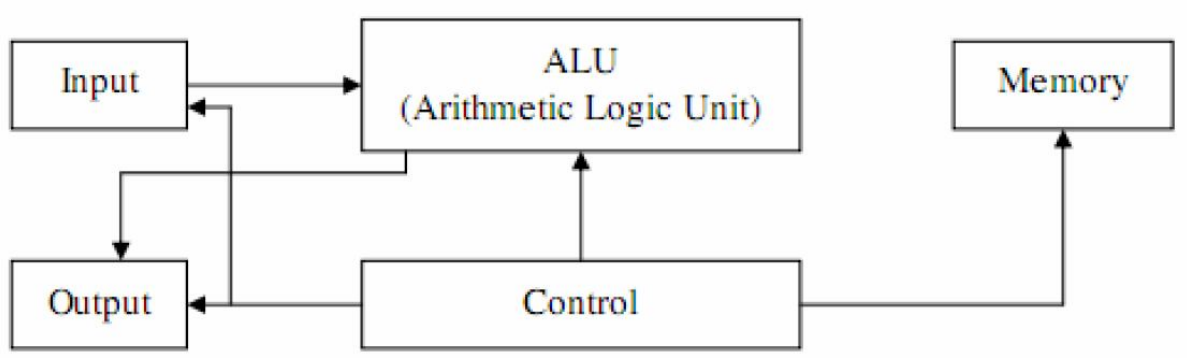
Ổn định tốc độ động cơ

Đếm sản phẩm của 1 nhà máy, xí nghiệp,…

Máy vận hành tự động (dạng CNC)

1.2. CẤU TRÚC CHUNG CỦA MỘT HỆ VI XỬ LÝ

Sơ đồ khối một máy tính cổ điển



Hình 1-2. 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).

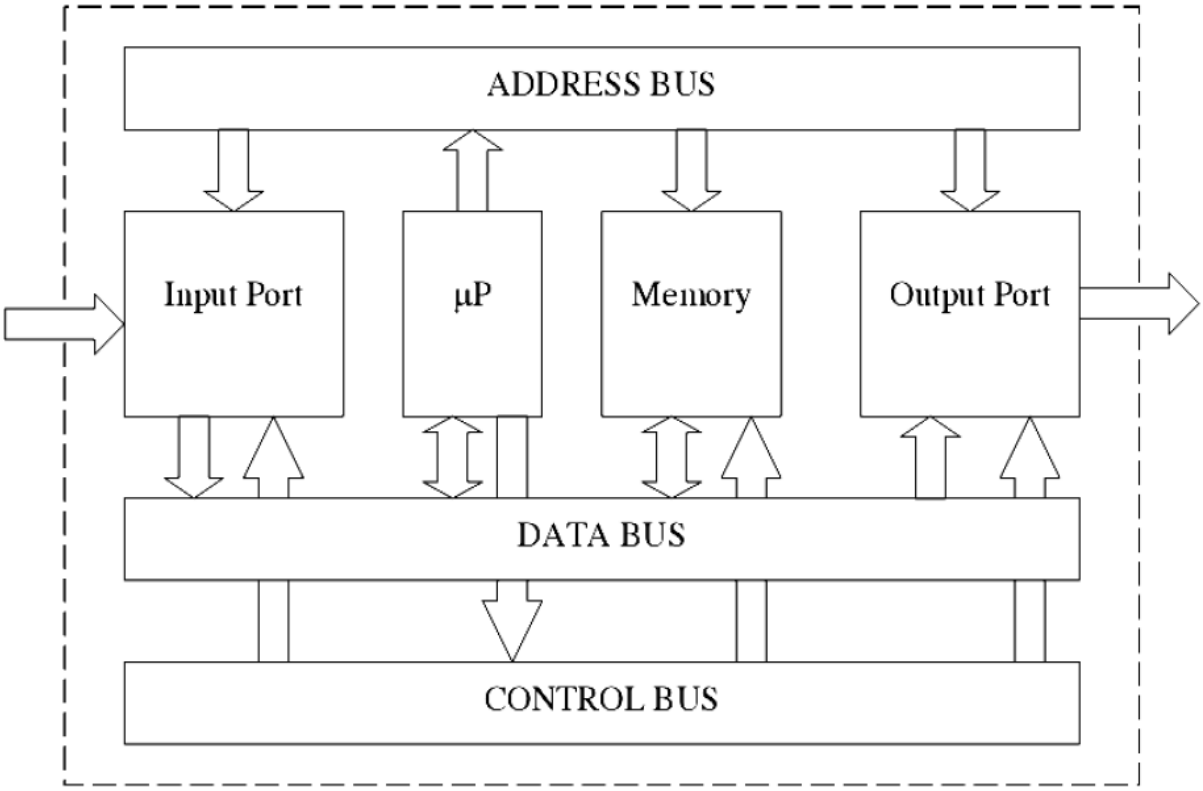
Các cổng vào/ra (song song (Parallel I/O Ports), nối tiếp (Serial I/O Ports))

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.

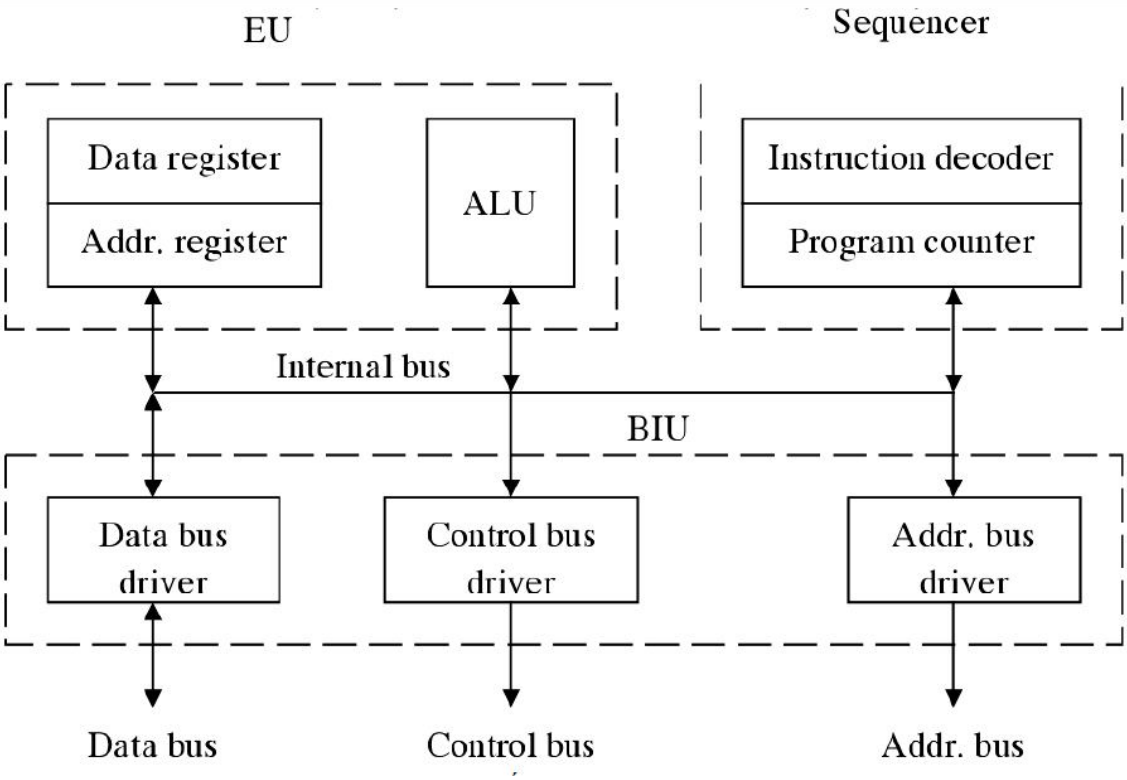


Hình 1-3. Sơ đồ khối hệ vi xử lý

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 1-4. Khối xử lý trung tâm

1.2.2. Hệ thống bus

Là các đường tín hiệu song song 1 chiều nối từ CPU đến bộ nhớ, bao gồm:

- Bus địa chỉ
- Address bus

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.

a. Bus dữ liệu - Data bus

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.

Độ rộng Bus: 4, 8, 16, 32 hay 64 bits

Bất kỳ thiết bị nào được 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.

b. 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:

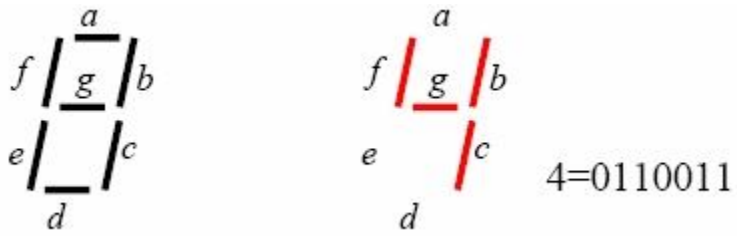
- Mã BCD không nén (unpacked): biểu diễn một số BCD bằng 8 bit nhị phân
- Mã BCD nén (packed): biểu diễn một số BCD bằng 4 bit nhị phân

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 1-5.LED 7 thanh và cách mã hóa

- **Các mã hệ đếm thông dụng**

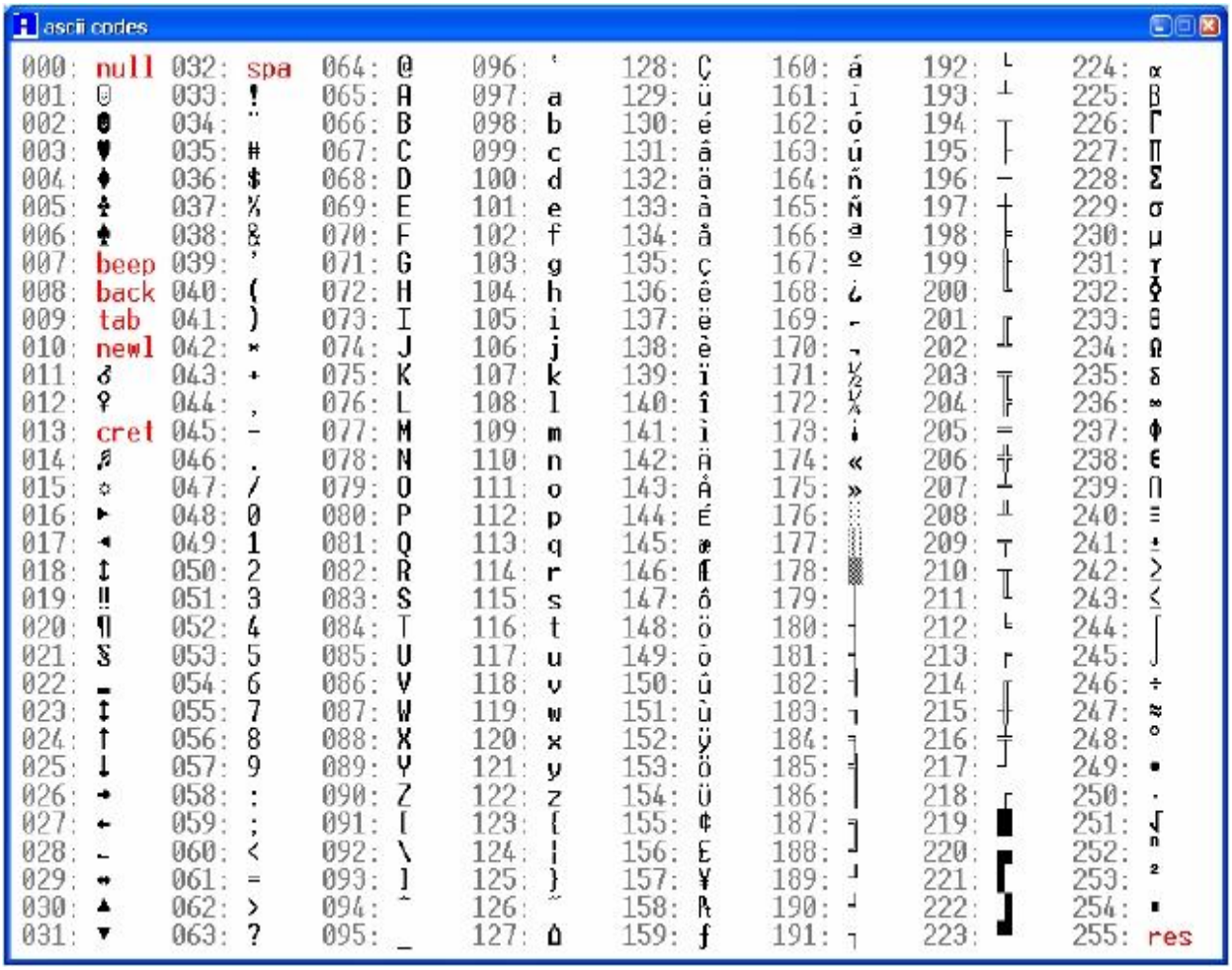
Hệ 10	Hệ 2	Hệ 8	Hệ 16	Binary-Coded Decimal		Gray Code	7-Segment	
				8421 BCD	EXCESS-3		abcdefg	Display
0	0000	0	0	0000	0011 0011	0000	111111	0
1	0001	1	1	0001	0011 0100	0001	011000	1
2	0010	2	2	0010	0011 0101	0011	110110	2
3	0011	3	3	0011	0011 0110	0010	111100	3
4	0100	4	4	0100	0011 0111	0110	011001	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

Bảng 1-2. Bảng mã ASCII



Bảng 1-3. Bảng mã ASCII có cả ký tự trong phần mở rộng

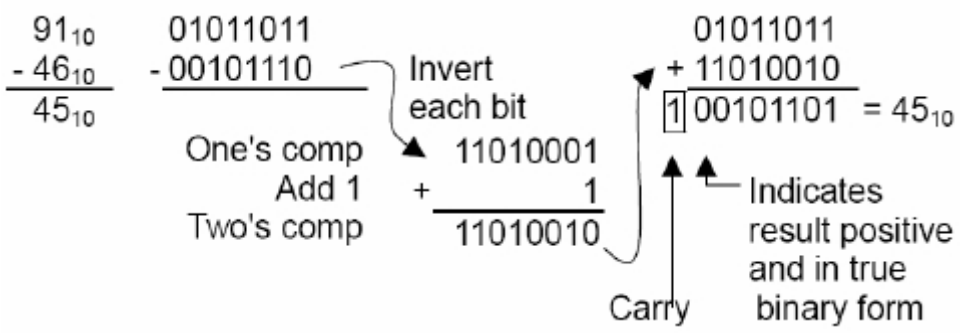
1.3.3. Các phép toán số học trên hệ đếm nhị phân

Vào			Ra	
A	B	BIN	D	BOUT
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

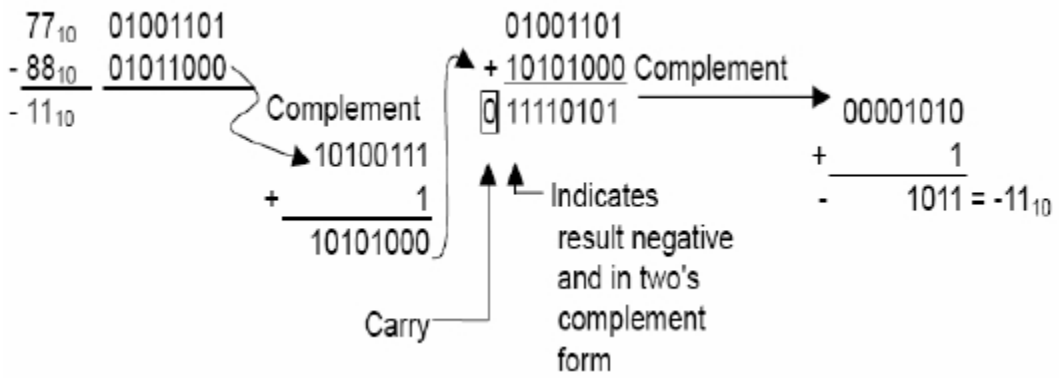
Vào			Ra	
A	B	BIN	D	BOUT
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Bảng 1-4 Phép cộng nhị phân và phép trừ nhị phân

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
VI ĐIỀU KHIỂN

2.1. ĐẶT VẤN ĐỀ

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

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

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

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

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

2.2. 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.

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,…

2.3. 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:

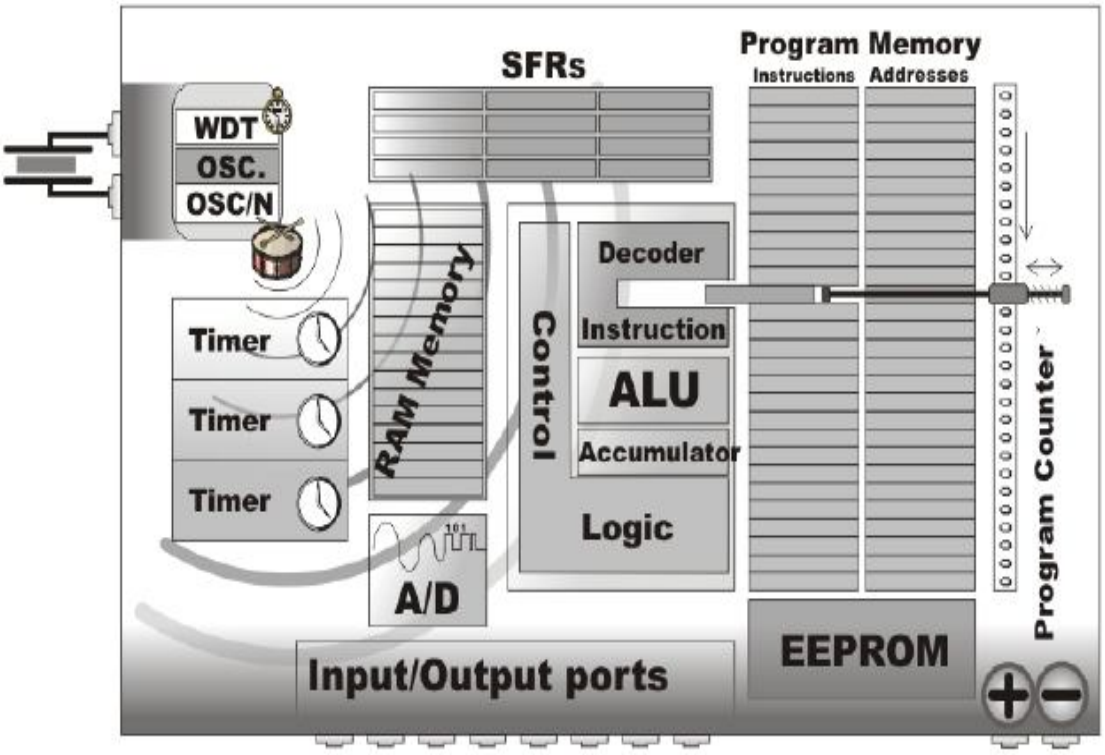
1. Khi không có nguồn điện cung cấp, vi điều khiển chỉ là một con chip có chương trình nạp sẵn vào trong đó và không có hoạt động gì xảy ra.

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 2-1. Cấu trúc chung họ VĐK

2.4. 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.

2.4.1.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

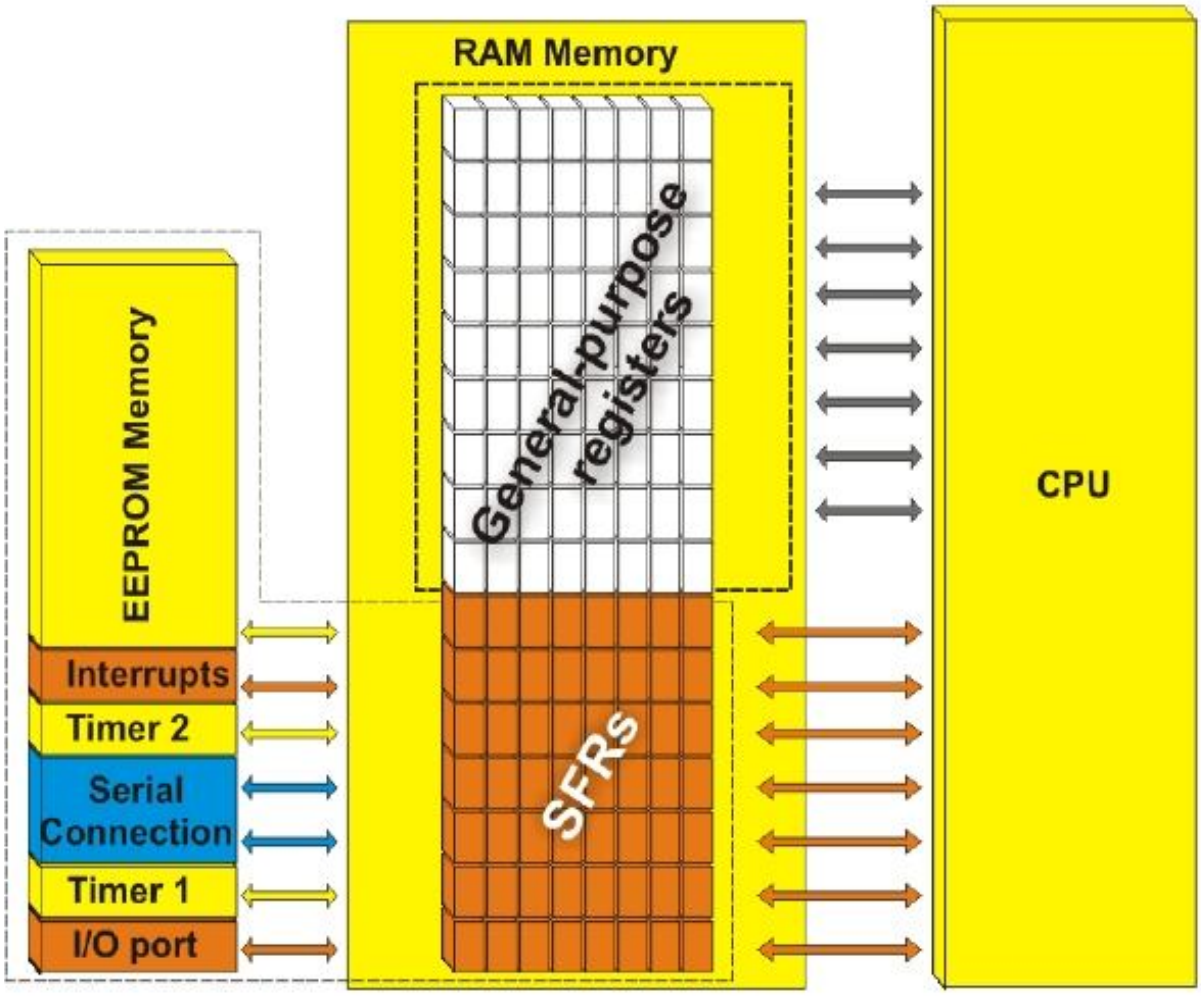
nối với môi trường ngoại vi. Kích thước của dãy ROM từ 512B đến 64KB

2.4.2. 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.

2.4.3. 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 2-2. Giao tiếp bộ nhớ

2.4.4. 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.

2.4.5. 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.

2.4.6. Central Processor Unit (CPU)

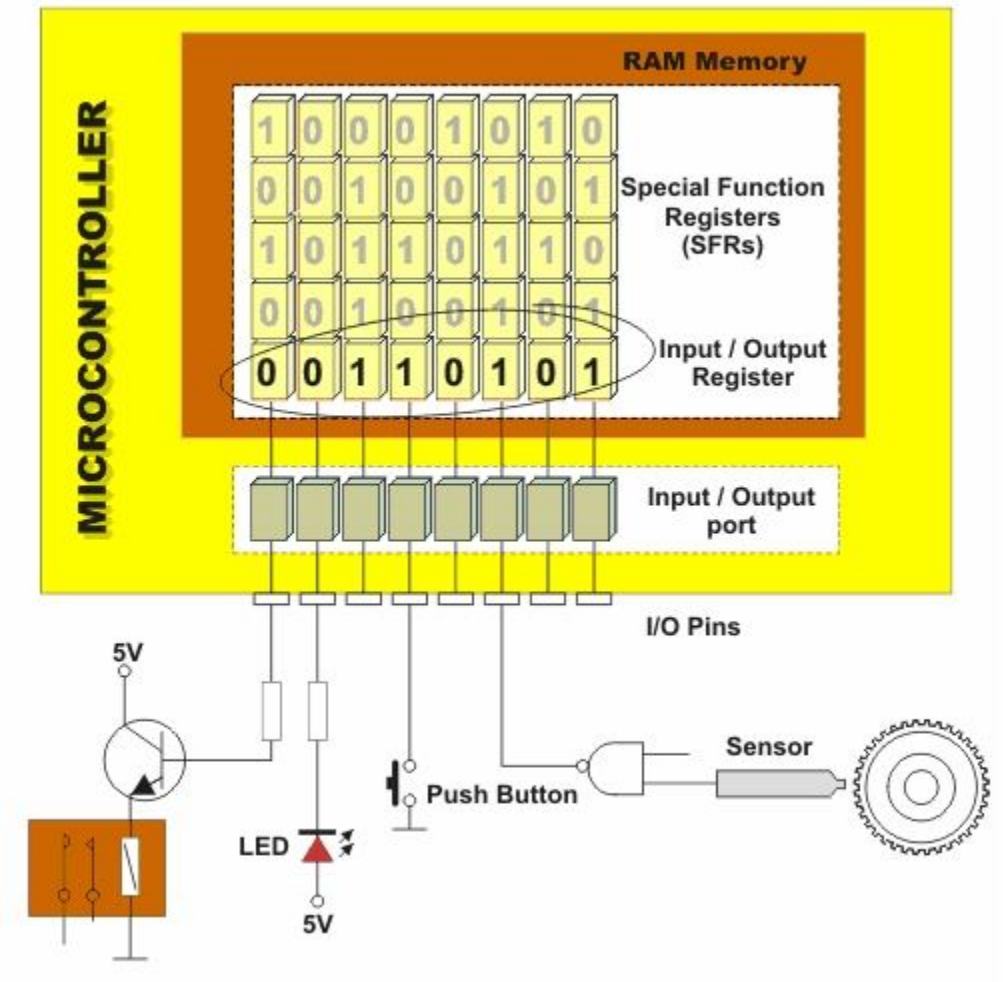
Đây là một đơn vị có nhiệm vụ điều khiển và giám sát tất cả các hoạt động bên trong vi điều khiển và người sử dụng không thể tác động vào hoạt động của nó. Nó bao gồm một số đơn vị con nhỏ hơn, trong đó quan trọng nhất là:

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 được 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.

2.4.7. Các cổng vào/ra (I/O Ports)

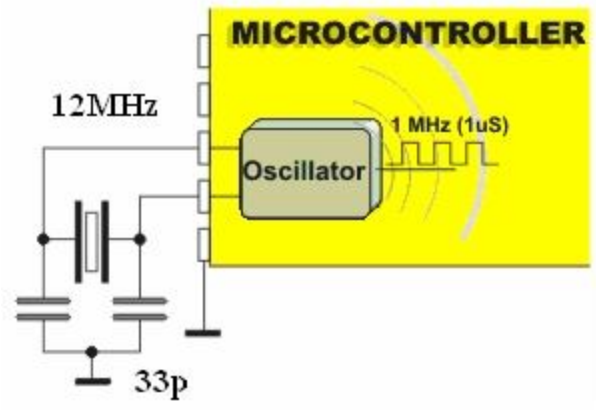
Để vi điều khiển có thể hoạt động hữu ích, nó cần có sự kết nối với các thiết bị ngoại vi. Mỗi vi điều khiển sẽ có một hoặc một số thanh ghi (được gọi là cổng) được kết nối với các chân của vi điều khiển.



Hình 2-3. 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.

2.4.8. Bộ dao động (Oscillator)

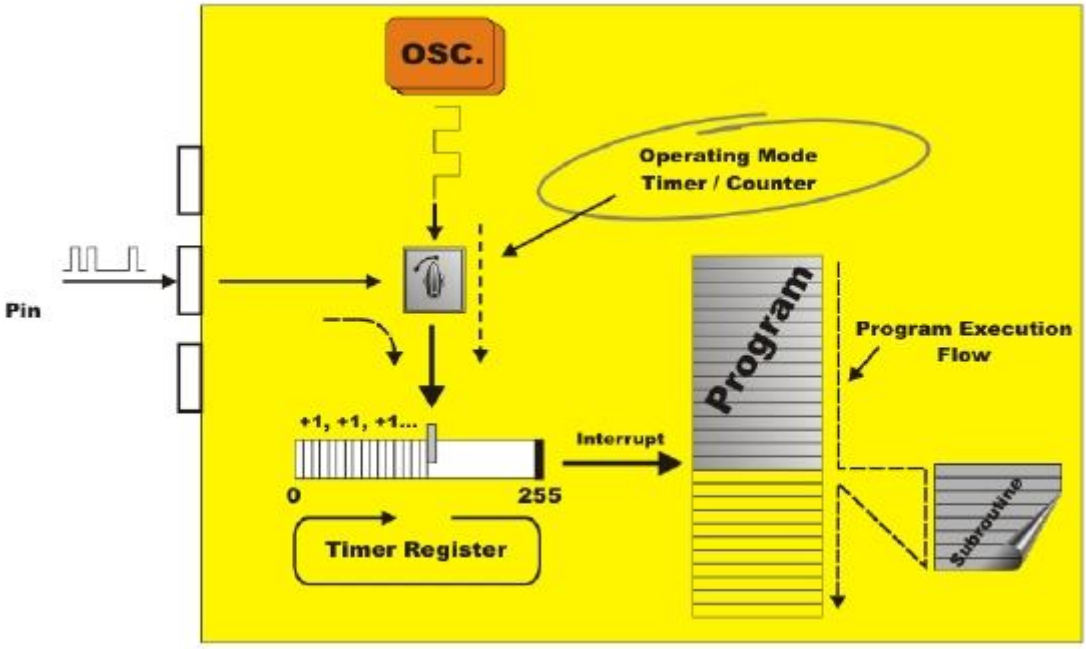


Hình 2-4. Ghép nối bộ dao động

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.

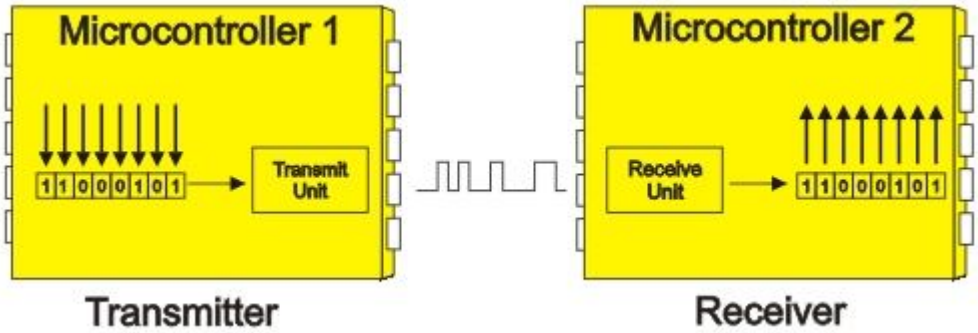
2.4.9. Bộ định thời/đếm (Timers/Counters)

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 2-5. Bộ định thời/đếm

2.4.10. Truyền thông nối tiếp



Hình 2-6. 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

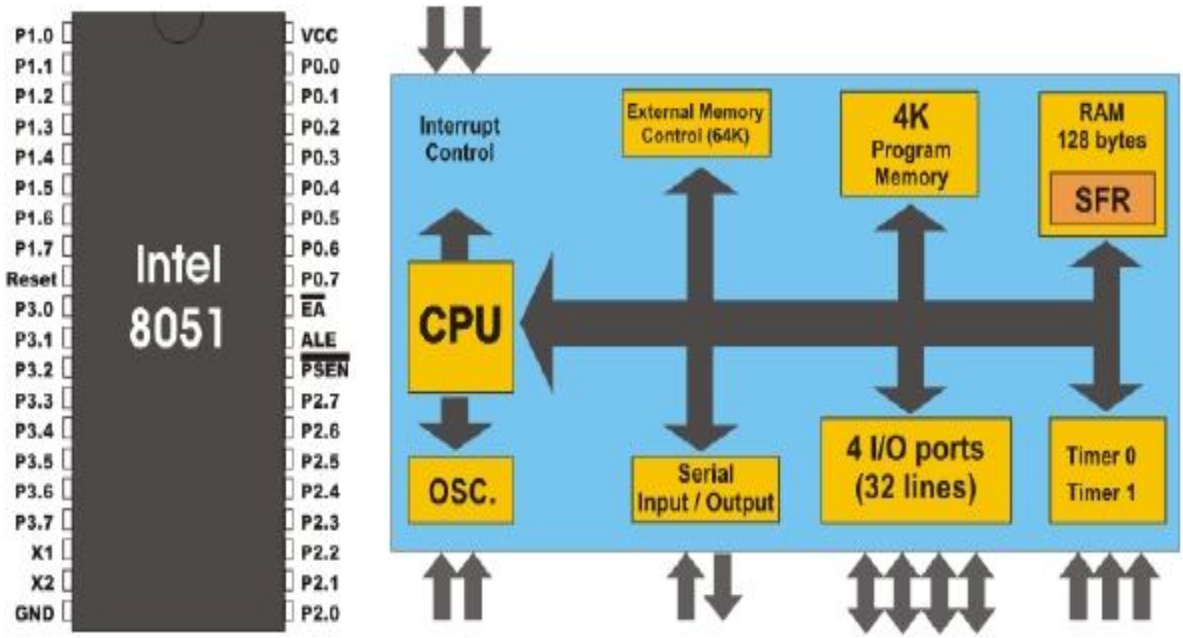
Không giống như các mạch tích hợp, chỉ cần kết nối các thành phần với nhau và bật nguồn, vi điều khiển cần phải lập trình trước. Để viết một chương trình cho vi điều khiển, có một vài ngôn ngữ lập trình bậc thấp có thể sử dụng như Assembly, C hay Basic. Viết một chương trình bao gồm việc viết các câu 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.

CHƯƠNG 3
KIẾN TRÚC VI ĐIỀU KHỂN 8051

3.1. 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 3-1.Kiến trúc vi điều khiển 8051

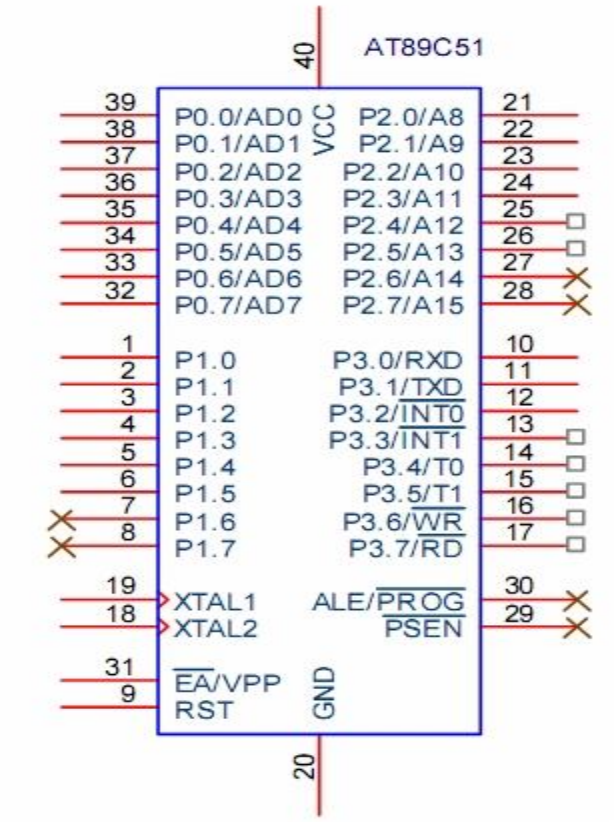
AT89C51 là vi điều khiển do Atmel sản xuất, chế tạo theo công nghệ CMOS có các đặc tính như sau:

+ 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. CHÂN VI ĐIỀU KHIỂN 8051



Hình 3-2. 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.

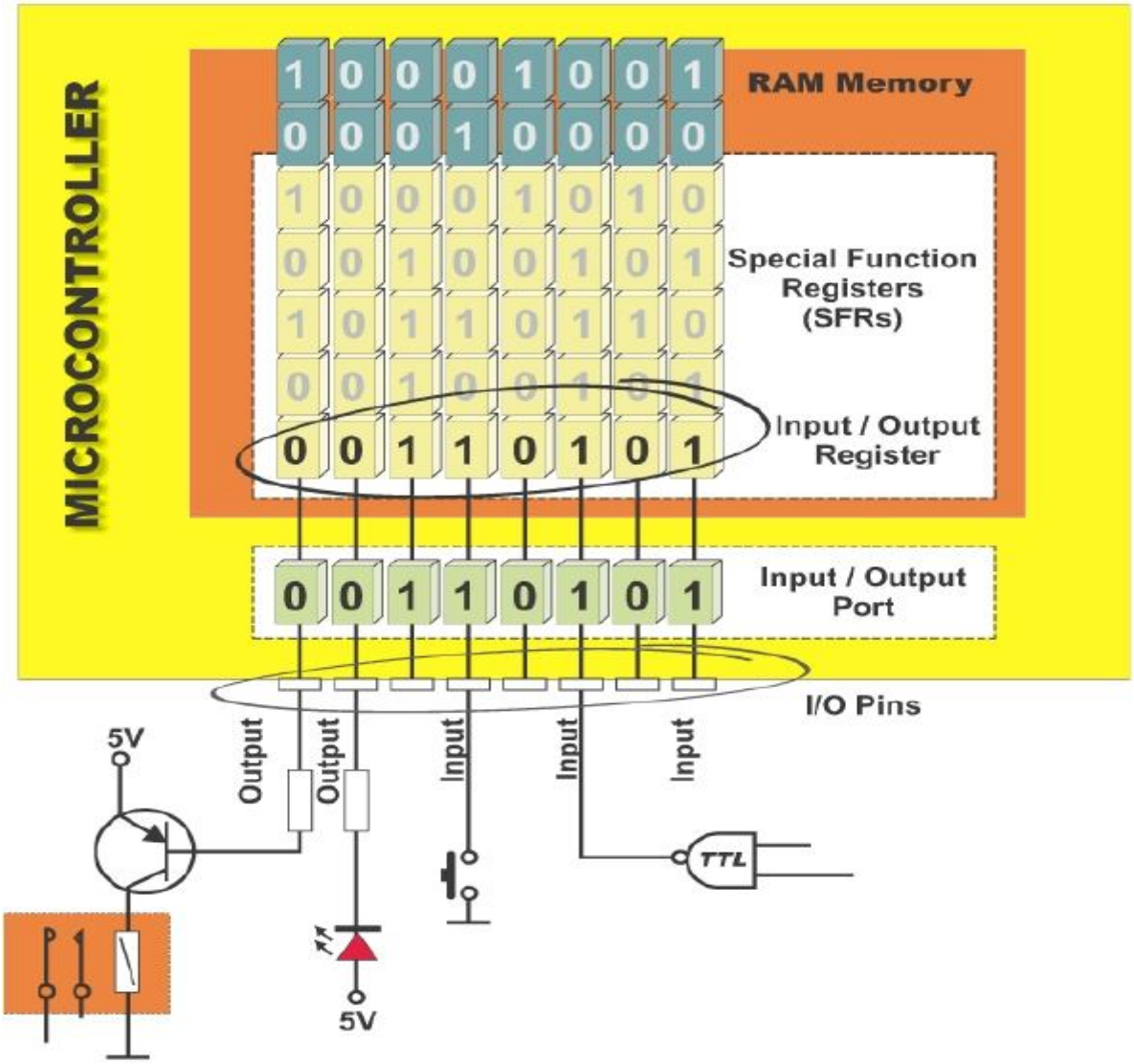
Hãy nhớ rằng : bình thường chân /PSEN sẽ được thả trống (No Connect).Chỉ khi nào cho /EA ở mức thấp thì lúc đó: /PSEN sẽ ở mức thấp trong thời gian lấy lệnh. Các mã nhị phân của chương trình được lấy từ EPROM qua bus dữ liệu và được chốt vào thanh ghi lệnh của 8951 để giải mã lệnh. /PSEN ở mức thụ động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

Các chân nguồn:

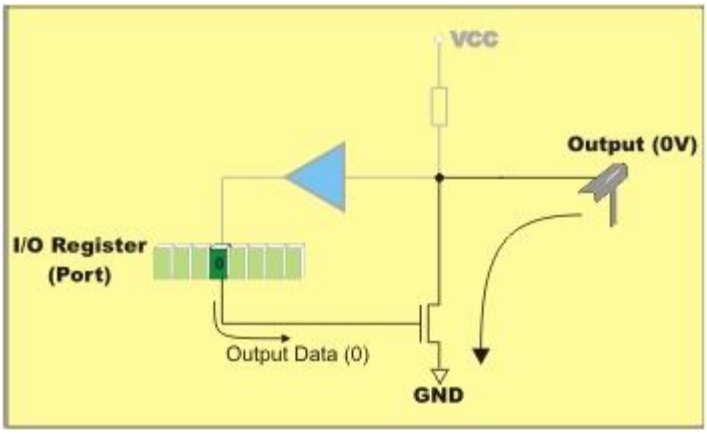
AT89C51 hoạt động ở nguồn đơn +5V. Vcc được nối vào chân 40, và Vss (GND) được nối vào chân 20.

3.3. CỔNG VÀO/ RA

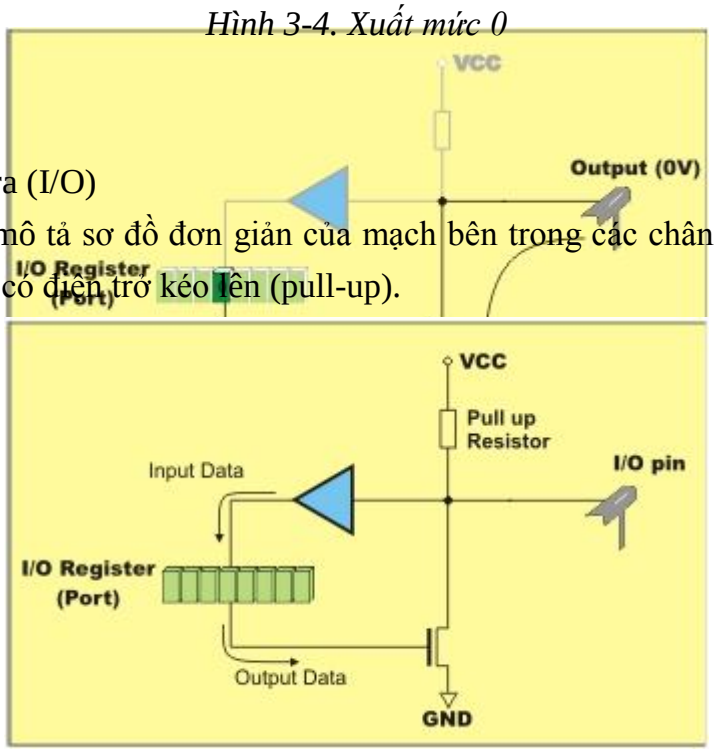
Tất cả các vi điều khiển 8051 đều có 4 cổng vào/ra 8 bit có thể thiết lập như cổng vào hoặc ra. Như vậy có tất cả 32 chân I/O cho phép vi điều khiển có thể kết nối với các thiết bị ngoại vi.



Hình 3-3. Cổng vào/ra



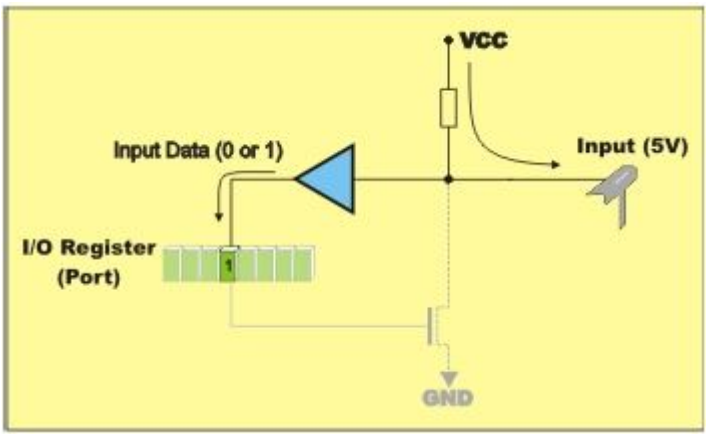
Hình 3-4. Xuất mức 0



Hình 3-5. 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 3-6. 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.

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).

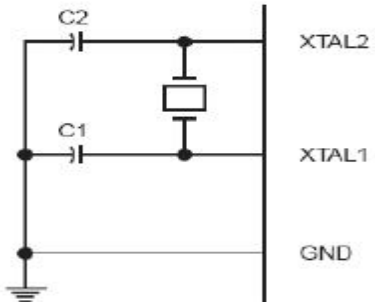
Ngoài ra, chân /EA được lấy làm chân cấp nguồn 12V khi lập trình cho ROM.

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.

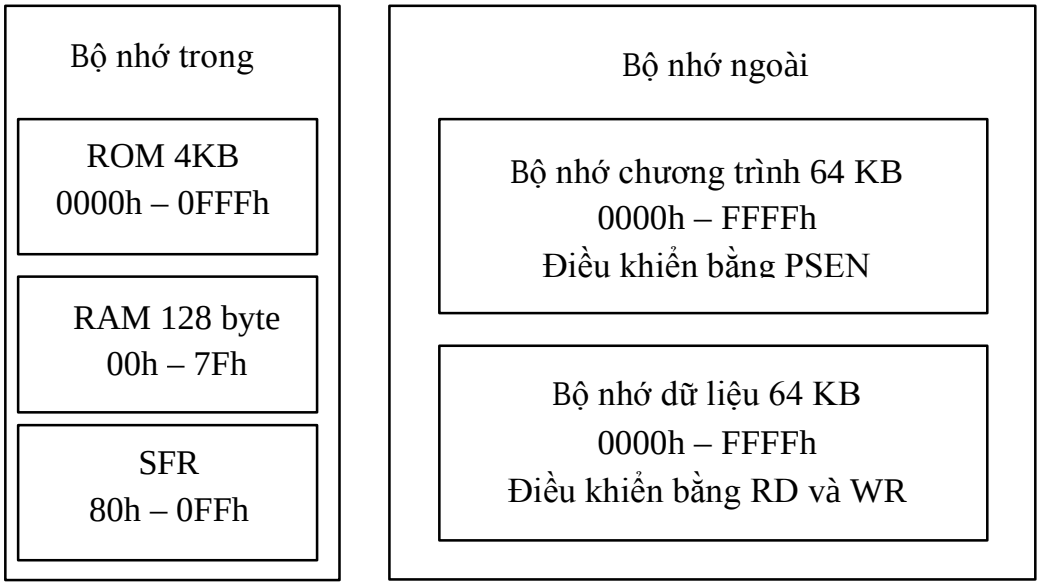
X1, X2:

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.



Hình 3-7 . Sơ đồ kết nối thạch anh

3.4 . TỔ CHỨC BỘ NHỚ



Hình 3-8. 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.

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

3.4.1 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):

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

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)								

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.

a. 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).

Bảng 3-3. Địa chỉ RAM nội 8051

b. 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 đích như trên ngoài các chức năng đề cập như phần sau.

c. 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.

d. 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. 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.

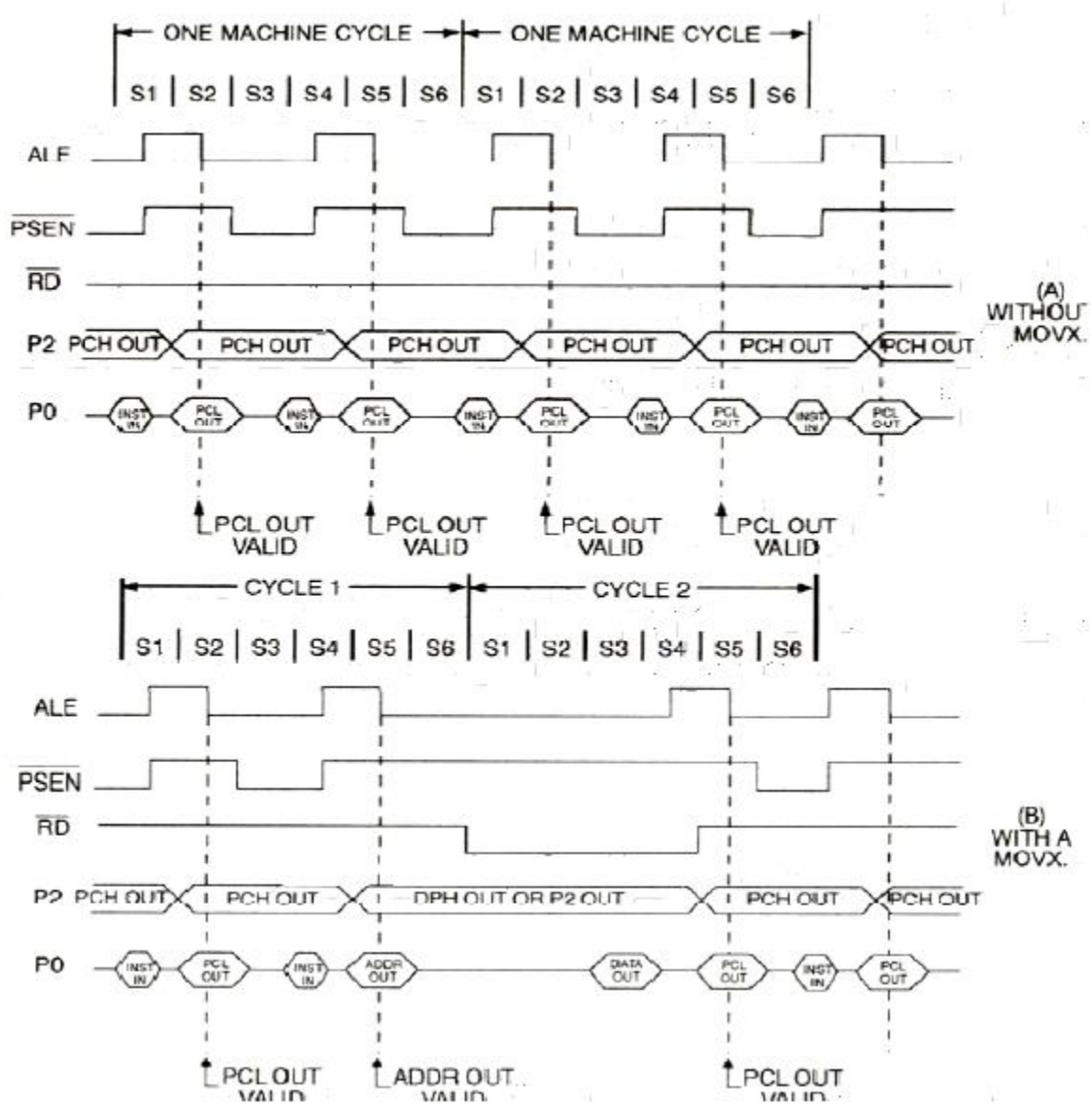
Port 0 được dùng làm địa chỉ thấp/ dữ liệu đa hợp. Tín hiệu /ALE để tách byte địa chỉ và đưa vào bộ chốt ngoài.

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 3-9. Thực thi bộ nhớ chương trình ngoài

a. Bộ nhớ chương trình ngoài:

Quá trình thực thi lệnh khi dùng bộ nhớ chương trình ngoài có thể mô tả như "Hình 3-9. 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 3-8. 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.

b. 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).

c. 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.

d. Giải mã địa chỉ

Trong các ứng dụng dựa trên AT89C51, ngoài giao tiếp bộ nhớ dỡ 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.

3.5. 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
```

- 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

Hình 3-10. Thanh ghi PSW

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

Hình 3-11. Chọn bank thanh ghi

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: 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

Hình 3-12. Thanh ghi PCON

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

3.6. BỘ ĐẾM / 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).

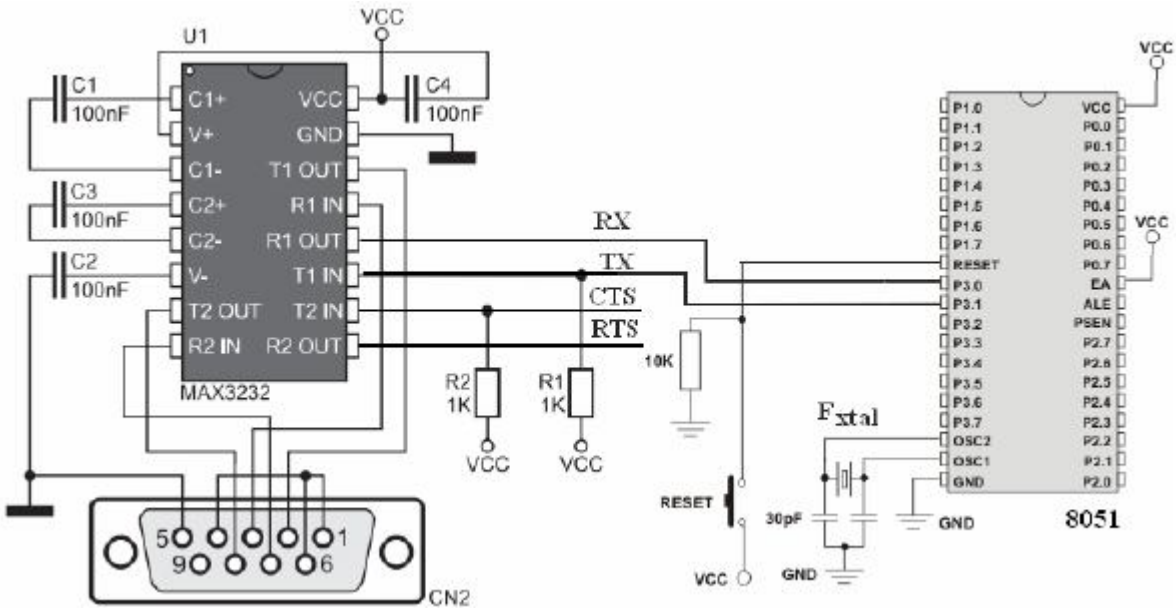
Hai timer có nguyên lý hoạt động hoàn toàn giống nhau và độc lập. Sau khi cho phép chạy, mỗi khi có thêm một xung tại đầu vào đếm, giá trị của timer sẽ tự động được tăng lên 1 đơn vị, cứ như vậy cho đến khi giá trị tăng lên vượt quá giá trị 65535 mà thanh ghi đếm có thể biểu diễn thì giá trị đếm lại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

3.7. 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 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 RS 232.



Hình 3-13 - 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).

3.8. 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.

CHƯƠNG 4
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.

4.1 CÁC CHẾ ĐỘ ĐỊA CHỈ

Có thể truy cập dữ liệu theo nhiều cách khác nhau. Dữ liệu có thể ở trong một thanh ghi hoặc trong bộ nhớ hoặc được cho như một giá trị tức thời các cách truy cập dữ liệu khác nhau được gọi là các chế độ đánh địa chỉ. Chương này chúng ta bàn luận về các chế độ đánh địa chỉ của 8051 trong phạm vi một số ví dụ.

Các chế độ đánh địa chỉ khác nhau của bộ vi xử lý được xác định như nó được thiết kế và do vậy người lập trình không thể đánh địa chỉ khác nhau là:

1. Tức thời 2. Theo thanh ghi 3. Trực tiếp
4. Gián tiếp 5. Theo chỉ số

4.1.1. 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  DPTR, #MYDATA
MOV  A, # 25H              ; Nạp giá trị 25H vào thanh ghi A
MOV  R4, #62               ; Nạp giá trị  62 thập phân vào R4
MOV  B, #40H               ; Nạp giá trị 40 H vào thanh ghi B
MOV  DPTR, #4521H          ; Nạp 4512H vào con trỏ dữ liệu DPTR
```

Mặc dù thanh ghi DPTR là 16 bit nó cũng có thể được truy cập như 2 thanh ghi 8 bit DPH và DPL trong đó DPH là byte cao và DPL là byte thấp.

Cũng lưu ý rằng lệnh dưới đây có thể tạo ra lỗi vì giá trị nạp vào DPTR lớn hơn16 bit:

```
MOV   DPTR, # 68975      ; Giá tri không hợp lệ  > 65535 (FFFFH)
```

4.1.2. 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  R2, A        ; Sao chép nội  dung thanh ghi  A vào thanh ghi R2
ADD  A, R5        ; Cộng nội dung  thanh ghi R5 vào thanh ghi A
```

Ta có thể chuyển dữ liệu giữa thanh ghi tích luỹ A và thanh ghi Rn (n từ 0 đến 7) nhưng việc chuyển dữ liệu giữa các thanh ghi Rn thì không được phép.

Ví dụ: MOV R4, R7 là không hợp lệ.

Lưu ý rằng các thanh ghi nguồn và đích phải phù hợp về kích thước. Hay nói cách khác, nếu viết " MOV DPTR, A" sẽ cho một lỗi vì nguồn là thanh ghi 8 bit và đích lại là thanh ghi 16 bit.

4.1.3. Địa chỉ trực tiếp

8051 có 128 byte bộ nhớ RAM. Bộ nhớ RAM được gán các địa chỉ từ 00 đến 7FH 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.

Mặc dù toàn bộ byte của bộ nhớ RAM có thể được truy cập bằng chế độ đánh địa chỉ trực tiếp, nhưng chế độ này thường được sử dụng nhất để truy cập các ngăn nhớ RAM từ 30H đến 7FH. Đây là do một thực tế là các ngăn nhớ dành cho băng ghi được truy cập bằng thanh ghi theo các tên gọi của chúng là R0 - R7 còn các ngăn nhớ khác của RAM thì không có tên như vậy. Trong chế độ đánh địa chỉ trực tiếp thì dữ liệu ở trong một ngăn nhớ RAM mà địa chỉ của nó được biết và địa chỉ này được cho như là một phần của lệnh. Khác với chế độ đánh địa chỉ tức thì mà toán hạng tự nó được cấp với lệnh. Dấu (# 0 là sự phân biệt giữa hai chế độ đánh địa chỉ. Xét các ví dụ dưới đây và lưu ý rằng các lệnh không có dấu (#):

```
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
MOV  R4, 7FH      ; Chuyển nội dung ngăn nhớ 7FH của RAM  vào R4
```

Các thanh ghi R0 - R7 có thể được truy cập theo 2 cách như sau:

```
MOV  A, 4    ; Hai lệnh này giống nhau đều sao nội dung thanh ghi  R4 vào A
MOV  A, R4  ;
```

4.1.4. Địa chỉ gián tiếp.

Trong chế độ này, một thanh ghi được sử dụng như một con trỏ đến dữ liệu. Nếu dữ liệu ở bên trong CPU thì chỉ các thanh ghi R0 và R1 được sử dụng cho mục đích này. Hay nói cách khác các thanh ghi R2 - R7 không thể dùng được để giữ địa chỉ của toán hạng nằm trong RAM khi sử dụng chế độ đanh địa chỉ này khi R0 và R1 được dùng như cac con trỏ, nghĩa là khi chúng giữ các địa chỉ của các ngăn nhớ RAM thì trước chúng phải đặt dấu (@) như chỉ ra dưới đây.

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

MOV @ R1, B ; Chuyển nội dung của B vào ngăn nhớ RAM có địa chỉ ở R1

Lưu ý rằng R0 cũng như R1 luôn có dấu "@" đứng trước. Khi không có dấu này thì đó là lệnh chuyển nội dung các thanh ghi R0 và R1 chứ không phải dữ liệu ngăn nhớ mà địa chỉ có trong R0 và R1.

* Ưu điểm của chế độ đánh địa chỉ gián tiếp thanh ghi.

Một trong những ưu điểm của chế độ đánh địa chỉ gián tiếp thanh ghi là nó làm cho việc truy cập dữ liệu năng động hơn so với chế độ đánh địa chỉ trực tiếp.

Ví dụ:

Viết chương trình để sao chép giá trị 55H vào ngăn nhớ RAM tại địa chỉ 40H đến 44H sử dụng:

- Chế độ định địa chỉ trực tiếp
- Chế độ đánh địa chỉ gián tiếp thanh ghi không dùng vòng lặp

Lời giải:

- Chế độ địa chỉ trực tiếp.

```
MOV  A, #55H          ; Nạp A giá trị  55H
MOV  40H, A           ; Sao chép A vào ngăn nhớ RAM  40H
MOV  41H, A           ; Sao chép A vào ngăn nhớ RAM  41H
MOV  42H, A           ; Sao chép A vào ngăn nhớ RAM  42H
MOV  43H, A           ; Sao chép A vào ngăn nhớ RAM  43H
MOV  44H, A           ; Sao chép A vào ngăn nhớ RAM  44H
```

- Chế độ đánh địa chỉ gián tiếp thanh ghi không dùng vòng lặp

```
MOV  A, # 55H      ; Nạp vào A giỏ trị 55H
MOV  R0, #40H      ; Nạp con trỏ  R0 = 40 H
MOV  @R0, A      ; Sao chép A vào vị trí ngăn nhớ RAM do R0 chỉ đến
INC    R0          ; Tăng con trỏ. Bây giờ R0 = 41H
MOV  @R0, A        ; Sao chép A vào vị trí ngăn nhớ RAM do R0 chỉ
```

```
INC   R0          ; Tăng con trỏ. Bây giờ R0 = 42H
MOV  @R0,A        ; Sao chép A vào vị trí ngăn nhớ RAM do R0 chỉ
INC   R0          ; Tăng con trỏ. Bây giờ R0 = 43H
MOV  @R0, A       ; Sao chép A vào vị trí ngăn nhớ RAM do R0 chỉ
INC   R0          ;Tăng con trỏ. Bây giờ R0 = 44H
MOV  @R0, A
```

4.1.5. Địa chỉ theo 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

Thanh ghi 16 bit DPTR là thanh ghi A được dùng để tạo ra địa chỉ của phần tử dữ liệu được lưu cất trong ROM trên chíp. Do các phần tử dữ liệu được cất trong không gian mã (chương trình) của ROM trên chip của 8051, nó phải dùng lệnh MOVC thay cho lệnh MOV (chủ C ở cuối lệnh là chỉ mà lệnh Code). 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.

4.2. TẬP LỆNH TRONG 8051

4.2.1. 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

4.2.2. 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.

Ví dụ:

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

4.2.3. Các lệnh toán học

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.

a. Phép cộng

ADD A, nguồn ; A = A + nguồn

Lệnh ADD được dùng để cộng hai toán hạng. Toán hạng đích luôn là thanh ghi A trong khi đó toán hạng nguồn có thể là một thanh ghi dữ liệu trực tiếp hoặc là ở trong bộ nhớ. Hãy nhớ rằng các phép toán số học từ bộ nhớ đến bộ nhớ không bao giờ được phép trong hợp ngữ. Lệnh này có thể thay đổi một trong các bit AF, CF hoặc PF của thanh ghi cờ phụ thuộc vào các toán hạng liên quan.

Ví dụ:

Hãy biểu diễn xem các lệnh dưới đây tác động đến thanh ghi cờ như thế nào?

MOV A, # 0F5H ; A = F5H

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

Lời giải:

F5H		1111 0101
+0BH	+	0000 1011
100H		0000 0000

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

b. Phép trừ các số không dấu.

Cú pháp: SUBB A, nguồn ; A = A - nguồn - CY.

Trong rất nhiều các bộ xử lý có hai lệnh khác nhau cho phép trừ đó là SUB và SUBB (trừ có mượn). Trong 8051 ta chỉ có một lệnh SUBB duy nhất. Để thực hiện SUB từ SUBB, do vậy có hai trường hợp cho lệnh SUBB là: với CY = 0 và với CY = 1. Lưu ý ở đây ta dùng cờ CY để mượn.

- Lệnh SUBB với CY = 0.

Trong phép trừ thì các bộ vi xử lý 8051 (thực tế là tất cả mọi CPU hiện đại) đều sử dụng phương pháp bù 2. Mặc dù mỗi CPU đều có mạch cộng, nó có thể quá cồng kềnh (và cần nhiều bóng bán dẫn) để thiết kế mạch trừ riêng biệt. Vì ly do đó mà 8051 sử dụng mạch cộng để thực hiện lệnh trừ. Giả sử 8051 sử dụng mạch cộng để thực hiện lệnh trừ và rằng CY - 0 trước khi thực hiện lệnh thì ta có thể tóm tắt các bước mà phần cứng CPU thực hiện lệnh SUBB đối với các số không dấu như sau:

1. Thực hiện lấy bù 2 của số trừ (toán hạng nguồn)
2. Cộng nó vào số bị trừ (A)
3. Đảo nhớ

Đây là 3 bước thực hiện bởi phần cứng bên trong của CPU 8051 đối với mỗi lệnh trừ SUBB bất kể đến nguồn của các toán hạng được cấp có được hỗ trợ chế độ đánh địa chỉ hay không? Sau ba bước này thì kết quả có được và các cờ được bật.

Ví dụ :

Phân tích chương trình sau:

```
        CLR  C
        MOV  A, #4CH        ; Nạp A giá trị 4CH (A = 4CH)
        SUBB A, #6EH        ; Trừ A cho 6EH
        JNC  NEXT           ; Nếu CY = 0 nhảy đến đích NEXT
        CPL  A              ; Nếu CY = 1 thực hiện bù 1
        INC  A              ; Tăng 1 để có bù 2
NEXT:   MOV  R1, A          ; Lưu A vào R1
```

- Lệnh SUBB khi CY = 1.

Lệnh này được dùng đối với các số nhiều byte và sẽ theo dõi việc mượn của toán hạng thấp. Nếu CY = 1 trước khi xem thực hiện SUBB thì nó cũng trừ 1 từ kết quả.

Ví dụ:

Phân tích chương trình sau:

```
CLR  C          ; CY = 0
MOV  A, #62     ; A = 62H
SUBB A, #96H    ; 62H - 96H = CCH with CY = 1
MOV  R7, A      ; Save the result
MOV  A, #27H    ; A = 27H
SUBB A, #12H    ; 27H - 12H - 1 = 14H
MOV  R6, A      ;
```

Sau khi SUBB thì A = 62H - 96H = CCH và cờ nhớ được lập báo rằng có mượn. Vì CY = 1 nên khi SUBB được thực hiện lần thứ 2 thì a = 27H - 12H - 1 = 14H. Do vậy, ta có 2762H - 1296H = 14CCH.

c. Nhân hai số không dấu.

MUL AB ; Là phép nhân A × B và kết quả 16 bit được đặt trong A và B.

Khi nhân byte với byte thì một trong các toán hạng phải trong thanh ghi A và toán hạng thứ hai phải ở trong thanh ghi B. Sau khi nhân kết quả ở trong các thanh ghi A và B. Phần tiếp thấp ở trong A, còn phần cao ở trong B. Ví dụ dưới đây trình bày phép nhân 25H với 65H. Kết quả là dữ liệu 16 bit được đặt trong A và B.

```
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
```

Nhân	Toán hạng 1	Toán hạng 2	Kết quả
Byte*Byte	A	B	A = byte thấp, B = byte cao

Bảng 4-1: Tóm tắt phép nhân hai số không dấu (MUL AB)

d. Chia hai số không dấu.

8051 cùng chỉ hỗ trợ phép chia hai số không dấu byte cho byte với cú pháp:

DIV AB ; Chia A cho B

Khi chia một byte cho một byte thì tử số (số bị chia) phải ở trong thanh ghi A và mẫu số (số chia) phải ở trong thanh ghi B. Sau khi lệnh chia DIV được thực hiện thì thương số được đặt trong A, còn số dư được đặt trong B. Xét ví dụ dưới đây:

```
MOV     A, #95      ; Nạp số bị chia vào A = 95
MOV     B, #10      ; Nạp số chia vào B = 10

DIV     AB          ; A = 09 (thương số); B = 05 (số dư)
```

Lưu ý các điểm sau khi thực hiện "DIV AB"

Lệnh này luôn bắt CY = 0 và OV = 0 nếu tử số không phải là số 0

Nếu tử số là số 0 (B = 0) thì OV =1 báo lỗi và CY = 0. Thực tế chuẩn trong tất cả mọi bộ vi xử lý khi chia một số cho 0 là bằng cách nào đó báo có kết quả không xác định. Trong 8051 thì cờ OV được thiết lập lên 1.

Phép chia	Tử số	Mẫu số	Thương số	Số dư
Byte cho Byte	A	B	A	B

Bảng 4-2. Tóm tắt phép chia không dấu (DIV AB).

4.2.4. Các lệnh logic

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

ANL: phép toán "Và" logic
ORL: phép toán "Hoặc " logic
XRL: phép toán "XOR " 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

a. Lệnh Và (ANL).

Cú pháp: ANL đích, nguồn ; đích = đích Và nguồn

Lệnh này sẽ thực hiện một phép Và lô-gíc trên hai toán hạng đích và nguồn và đặt kết quả vào đích. Đích thường là thanh ghi tổng (tích luỹ). Toán hạng nguồn có thể là thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. Lệnh AND đối với toán hạng theo byte không có tác động lên các cờ. Nó thường được dùng để che (đặt về 0) những bit nhất định của một toán hạng.

Ví dụ:

Trình bày kết quả của các lệnh sau:

```
MOV  A, #35H          ; Gán A = 35H
ANL  A, #0FH          ; Thực hiện Và logic A và 0FH (Bây giờ A = 05)
```

Lời giải:

```
35H   0  0  1  1  0  1  0  1
0FH   0  0  0  0  1  1  1  1
05H   0  0  0  0  0  1  0  1          35H và 0FH = 05H
```

b. Lệnh Hoặc (ORL).

Cú pháp ORL đích, nguồn ; đích = đích hoặc nguồn.

Các toán hạng đích và nguồn được Hoặc với nhau và kết quả được đặt vào đích. Phép Hoặc có thể được dùng để thiết lập những bit nhất định của một toán hạng 1. Đích thường là thanh ghi tổng, toán hạng nguồn có thể là một thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. Lệnh ORL đối với các toán hạng đánh địa chỉ theo byte sẽ không có tác động đến bất kỳ cờ nào.

Ví dụ: Trình bày kết quả của đoạn mã sau:

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

Lời giải:

```
04H   0000  0100
68H   0110  1000
6CH   0110  1100
 04  OR  68 = 6CH
```

c. Lệnh XOR (hoặc loại trừ).

Cú pháp: XRL đích, nguồn ; đích = đích Hoặc loại trừ nguồn.

Lệnh này sẽ thực hiện phép XRL trên hai toán hạng và đặt kết quả vào đích. Đích thường là thanh ghi tổng. Toán hạng nguồn có thể là một thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. Lệnh XRL đối với các toán hạng đánh địa chỉ theo byte sẽ không có tác động đến bất kỳ cờ nào.

Ví dụ: Trình bày kết quả của đoạn mã sau:

```
MOV  A, #54H
XRL   A, #78H
```

Lời giải:

```
54H   0  1  0  1  0  1  0  0
78H   0  1  1  1  1  0  0  0
2CH   0  0  1  0  1  1  0  1
54H  XRL  78H = 2CH
```

Lệnh XRL có thể được dùng để xoá nội dung của một thanh ghi bằng cách XOR nó với chính nó.

d. Lệnh bù thanh ghi tổng CPL A.

Lệnh này bù nội dung của thanh ghi tổng A. Phép bù là phép biến đổi các số 0 thành các số 1 và đổi các số 1 sang số 0. Đây cũng còn được gọi là phép bù 1.

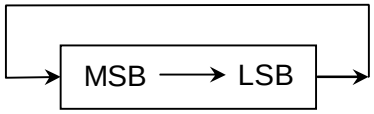
```
MOV          A, #55H
CPL          A            ; Bây giờ nội dung của thanh ghi A là AAH
```

e. Quay phải: RR

Cú pháp: **RR A** ; Quay các bit thanh ghi A sang phải.

Trong phép quay phải, 8 bit của thanh ghi tổng được quay sang phải một bit và bit D0 rời từ vị trí bit thấp nhất và chuyển sang bit cao nhất D7. Xem đoạn mã dưới đây.

```
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
```

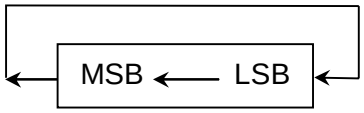


f. Quay trái: RL

Cú pháp: RL A ; Quay trái các bit của thanh ghi A.

Trong phép quay trái thì 8 bit của thanh ghi A được quay sang trái 1 bit và bit D7 rời khỏi vị trí bit cao nhất chuyển sang vị trí bit thấp nhất D0. Xem biểu đồ mã dưới đây.

```
MOV  A, #72H      ; A = 0111 0010
RL   A            ; A = 1110 0100
RL   A            ; A = 1100 1001
```



Lưu ý rằng trong các lệnh RR và RL thì không có cờ nào bị tác động.

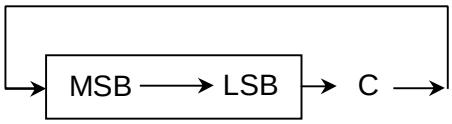
g. Quay có nhớ.

Cú pháp: RRC A và RLC A

* Quay phải có nhớ: RRC A

Trong quay phải có nhớ thì các bit của thanh ghi A được quay từ trái sang phải 1 bit và bit thấp nhất được đưa vào cờ nhớ CY và sau đó cờ CY được đưa vào vị trí bit cao nhất. Hay nói cách khác, trong phép RRC A thì LSB được chuyển vào CY và CY được chuyển vào MSB. Trong thực tế thì cờ nhớ CY tác động như là một bit bộ phận của thanh ghi A làm nó trở thành thanh ghi 9 bit.

```
CLR  C            ; make CY = 0
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
```



* Quay trái có nhớ: RLC A.

Trong RLC A thì các bit được dịch phải một bit và đẩy bit MSB vào cờ nhớ CY, sau đó CY được chuyển vào bit LSB. Hay nói cách khác, trong RLC thì bit MSB được chuyển vào CY và CY được chuyển vào LSB. Hãy xem đoạn mã sau.

```
SETB C          ; Make CY = 1
MOV  A, #15H    ; A = 0001 0101
RRC  A          ; A = 0101 1011  CY = 0
RRC  A          ; A = 0101 0110  CY = 0
RCC  A          ; A = 1010 1100  CY = 0
RCC  A          ; A = 1000 1000  CY = 1
```

4.2.5. 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)

4.2.6. Các lệnh thao tác bit

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
JNC đích	Nhảy tới đích nếu CY = 0
CLR C	Xoá bit nhớ CY = 0
JC đích	Nhảy tới đích nếu CY = 1

Ví dụ:

Hãy viết chương trình thực hiện các công việc sau:

a) Duy trì hiển thị bit P1.2 cho đến khi nó lên cao
b) Khi P1.2 lên cao, hãy ghi giá trị 45H vào cổng P0
c) Gửi một xung cao xuống thấp (H-to-L) tới P2.3

Lời giải:

```
        SETB P1.2          ; Tạo bit P1.2 là đầu vào
        MOV  A, #45H       ; Gán A = 45H
AGAIN:  JNB  P1.2, AGAIN   ; Thoát khi P1.2 = 1
```

```
MOV  P0, A          ; Xuất A tới cổng P0
SETB P2.3           ; Đưa P2.3 lên cao
CLR  P2.3           ; Tạo P2.3 xuống thấp để xung H-T0-L
```

Trong chương trình này lệnh "JNB P1.2, AGCN" (JNB có nghĩa là nhảy nếu không bit) ở lại vòng lặp cho đến khi P1.2 chưa lên cao. Khi P1.2 lên cao nó thoát ra khỏi vòng lặp ghi giá trị 45H tới cổng P0 và tạo ra xung H-to-L bằng chuỗi các lệnh SETB và CLR.

4.2.7. 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 4-3. 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 4-4. Lệnh đọc cổng ra

4.2.8. 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:

a. Các lệnh nhảy có điều kiện.

Các lệnh nhảy có điều kiện đối với 8051 được tổng hợp trong bảng 2.2. Các chi tiết về mỗi lệnh được cho trong phụ lục AppendixA. Trong bảng 2.2 lưu ý rằng một số lệnh như JZ (nhảy nếu A = 0) và JC (nhảy nếu có nhớ) chỉ nhảy nếu một điều kiện nhất định được thoả mãn. Kế tiếp ta xét một số lệnh nhảy có điều kiện với các Ví dụ minh hoạ sau.

Lệnh JZ: (nhảy nếu A = 0). Trong lệnh này nội dung của thanh ghi A được kiểm tra. Nếu nó bằng không thì nó nhảy đến địa chỉ đích. Ví dụ xét đoạn mã sau:

```
MOV     A, R0     ; Nạp giá trị của R0 vào A
JZ      OVER      ; Nhảy đến OVER nếu A = 0
MOV     A, R1     ; Nạp giá trị của R1 vào A
JZ      OVER      ; Nhảy đến OVER nếu A = 0
OVER ...
```

Trong chương trình này nếu R0 hoặc R1 có giá trị bằng 0 thì nó nhảy đến địa chỉ có nhãn OVER. Lưu ý rằng lệnh JZ chỉ có thể được sử dụng đối với thanh ghi A. Nó chỉ có thể kiểm tra xem thanh ghi A có bằng không không và nó không áp dụng cho bất kỳ thanh ghi nào khác. Quan trọng hơn là ta không phải thực hiện một lệnh số học nào như đếm giảm để sử dụng lệnh JNZ như ở ví dụ dưới đây.

Ví dụ :

Viết một chương trình để xác định xem R5 có chứa giá trị 0 không? Nếu nạp thì nó cho giá trị 55H.

Lời giải:

```
        MOV     A, R5     ; Sao nội dung R5 vào A
        JNZ     NEXT      ; Nhảy đến NEXT nếu A không bằng 0
        MOV     R5, #55H  ;
NEXT:           ...
```

Lệnh JNC: (nhảy nếu không có nhớ, cờ CY = 0).

Trong lệnh này thì bit cờ nhớ trong thanh ghi cờ PSW được dùng để thực hiện quyết định nhảy. Khi thực hiện lệnh "JNC nhãn" thì bộ xử lý kiểm tra cờ nhớ xem nó có được bật không (CY = 1). Nếu nó không bật thì CPU bắt đầu nạp và thực hiện các lệnh từ địa chỉ của nhãn. Nếu cờ CY = 1 thì nó sẽ không nhảy và thực hiện lệnh kế tiếp dưới JNC.

Cần phải lưu ý rằng cũng có lệnh "JC nhãn". Trong lệnh JC thì nếu CY = 1 nó nhảy đến địa chỉ đích là nhãn. Ta sẽ xét các ví dụ về các lệnh này trong các ứng dụng ở các chương sau.

Ngoài ra lệnh JB (nhảy nếu bit có mức cao), JNB (nhảy nếu bit có mức thấp).

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

Bảng 4-5. Các lệnh nhảy có 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).

Lời giải:

```
        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 = GE + 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
```

Tất cả các lệnh nhảy có điều kiện đều là những phép nhảy ngắn.

Cần phải lưu ý rằng tất cả các lệnh nhảy có điều kiện đều là các phép nhảy ngắn, có nghĩa là địa chỉ của đích đều phải nằm trong khoảng -127 đến +127 byte của nội dung bộ đếm chương trình PC.

b. Các lệnh nhảy không điều kiện.

Lệnh nhảy không điều kiện là một phép nhảy trong đó điều khiển được truyền không điều kiện đến địa chỉ đích. 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.

Hãy nhớ rằng, mặc dù bộ đếm chương trình trong 8051 là 16 bit, do vậy cho không gian địa chỉ là 64k byte, nhưng bộ nhớ chương trình ROM trên chíp lớn như vậy. 8051 đầu tiên chỉ có 4k byte ROM trên chíp cho không gian chương trình, do vậy mỗi byte đều rất quý giá. Vì lý do đó mà có cả lệnh nhảy gần SJMP chỉ có 2 byte so với lệnh nhảy xa LJMP dài 3 byte. Điều này có thể tiết kiệm được một số byte bộ nhớ trong rất nhiều ứng dụng mà không gian bộ nhớ có hạn hẹp.

Lệnh 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.

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. Do vậy LCALL có thể được dùng để gọi các chương trình con ở bất kỳ vị trí nào trong phạm vi 64k byte, không gian địa chỉ của 8051. Để đảm bảo rằng sau khi thực hiện một chương trình được gọi để 8051 biết

được chỗ quay trở về thì nó tự động cất vào ngăn xếp địa chỉ của lệnh đứng ngay sau lệnh gọi LCALL. Khi một chương trình con được gọi, điều khiển được chuyển đến chương trình con đó và bộ xử lý cất bộ đếm chương trình PC vào ngăn xếp và bắt đầu nạp lệnh vào vị trí mới. Sau khi kết thúc thực hiện chương trình con thì lệnh trở về RET chuyển điều khiển về cho nguồn gọi. Mỗi chương trình con cần lệnh RET như là lệnh cuối cùng.

Cần phải lưu ý các điểm sau đây:

1. Chương trình con DELAY khi thực hiện lệnh "LCALL DELAY" đầu tiên thì địa chỉ của lệnh ngay kế nó là "MOV A, #0AAH" được đẩy vào ngăn xếp và 8051 bắt đầu thực hiện các lệnh ở địa chỉ 300H.
2. Trong chương trình con DELAY, lúc đầu bộ đếm R5 được đặt về giá trị 255 (R5 = FFH). Do vậy, vòng lặp được lặp lại 256 lần. Khi R5 trở về 0 điều khiển rơi xuống lệnh quay trở về RET mà nó kéo địa chỉ từ ngăn xếp vào bộ đếm chương trình và tiếp tục thực hiện lệnh sau lệnh gọi CALL.

Ví dụ :

Hãy viết một chương trình để chốt tất cả các bit của cổng P1 bằng cách gửi đến nó giá trị 55H và AAH liên tục. Hãy đặt một độ trễ thời gian giữa mỗi lần xuất dữ liệu tới cổng P1. Chương trình này sẽ được sử dụng để kiểm tra các cổng của 8051 trong chương tiếp theo.

Lời giải:

```
            ORG   0000
BACK:       MOV   A, #55H          ; Nạp A với giá trị 55H
            MOV   P1, A            ; Gửi 55H đến cổng P1
            LCALL       DELAY      ; Tạo trễ thời gian
            MOV   A, #0AAH         ; Nạp A với giá trị  AAH
            MOV   P1, A            ; Gửi AAH đến cổng P1
            LCALL       DELAY      ; Giữ chậm
            SJMP  BACK             ; Lặp lại vô tận
; ------------------ - Đây là chương trình con tạo độ trễ thời gian
            ORG   300H             ; Đặt trễ thời gian ở địa chỉ 300H
DELAY:      MOV   R5, #00H         ; Nạp bộ đếm R5 = 255H (hay FFH)
AGAIN:      DJNZ  R5, AGAIN        ; Tiếp tục cho đến khi R5 = 0
            RET                    ; Trả điều khiển về nguồn gọi (khi R5 = 0)
            END                    ; Kết thúc tệp tin của hợp ngữ
```

4.3 CẤU TRÚC CHUNG CHƯƠNG TRÌNH HỢP NGỮ CHO 8051

4.3.1. Các thành phần cơ bản của ngôn ngữ Assembly

Cấu trúc của một lệnh hợp ngữ có 4 trường như sau:

[nhãn:] [từ gợi nhớ] [các toán hạng] [; chú giải]

Các trường trong dấu ngoặc vuông là tuỳ chọn và không phải dòng lệnh nào cũng có chúng. Các dấu ngoặc vuông không được viết vào. Với dạng thức trên đây cần lưu ý các điểm sau:

a. Trường nhãn cho phép chương trình tham chiếu đến một dòng lệnh bằng tên. Nó không được viết quá một số ký tự nhất định. Hãy kiểm tra quy định này của hợp ngữ mà ta sử dụng.

b. Từ gợi nhớ (lệnh) và các toán hạng là các trường kết hợp với nhau thực thi công việc thực tế của chương trình và hoàn thiện các nhiệm vụ mà chương trình được viết cho chúng. Trong hợp ngữ các câu lệnh như:

“ ADD A, B”

“MOV A, #67H”

thì ADD và MOV là những từ gợi nhớ tạo ra mã lệnh, còn “A, B” và “A, #67H” là những toán hạng thì hai trường có thể chứa các lệnh giả hoặc chỉ lệnh của hợp ngữ. Hãy nhớ rằng các chỉ lệnh không tạo ra mã lệnh nào (mã máy) và chúng chỉ dùng bởi hợp ngữ, ngược lại đối với các lệnh là chúng được dịch ra mã máy (mã lênh) cho CPU thực hiện.

c. Trường chú giải luôn phải bắt đầu bằng dấu chấm phẩy (;). Các chú giải có thể bắt đầu ở đầu dòng hoặc cuối dòng. Hợp ngữ bỏ qua các chú giải nhưng chúng lại rất cần thiết đối với lập trình viên. Mặc dù các chú giải là tuỳ chọn, không bắt buộc nhưng ta nên dùng chúng để mô tả chương trình để giúp cho người khác đọc và hiểu chương trình dễ dàng hơn.

d. Lưu ý đến nhãn HERE trong trường nhãn của ví dụ mẫu, một nhãn bất kỳ tham chiếu đến một lệnh phải có dấu hai chấm (:) đứng ở sau. Trong câu lệnh nhảy ngắn SJMP thì 8051 được ra lệnh ở lại trong vòng lặp này vô hạn. Nếu hệ thống của chúng ta có một chương trình giám sát thì takhông cần dòng lệnh này và nó có thể được xoá đi ra khỏi chương trình.

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.

4.3.2. 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 2 byte 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 (chú ý đến dấu #). Vậy ưu điểm của việc sử dụng EQU là gì? Giả sử có một hằng số (một giá trị cố định) được dùng trong nhiều chỗ khác nhau trong chương trình và lập trình viên muốn thay đổi giá trị của nó trong cả chương trình. Bằng việc sử dụng chỉ lệnh EQU ta có thể thay đổi một lần và hợp ngữ sẽ thay đổi tất cả mọi lần xuất hiện của nó.

- **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 4-6. 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				

Bảng 4-7. Một số từ khóa của Assembly

4.3.3. 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       ; Nhảy qua vùng ngắt
MAIN:
MOV R1,#10       ;(nạp cho R1 giá trị là 10).
LAP1:
DJNZ  R1, LAP1   ; Nếu R1 khác 0 thì giảm R1 và nhảy tới nhãn 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
```

CHƯƠNG 5
BỘ ĐỊNH THỜI, BỘ ĐẾM

5.1. 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.

5.1.1. 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 5-1. Các thanh ghi của bộ Timer 0

5.1.2. 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 5-2. Các thanh ghi của bộ Timer 1

5.1.3. 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.

MSB							LSB
GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer1				Timer0			

Hình 5-3. Timer TMOD

a. 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 5-1. Chế độ hoạt động của Timer/Counter

b. C/ T (Bộ đếm / 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.

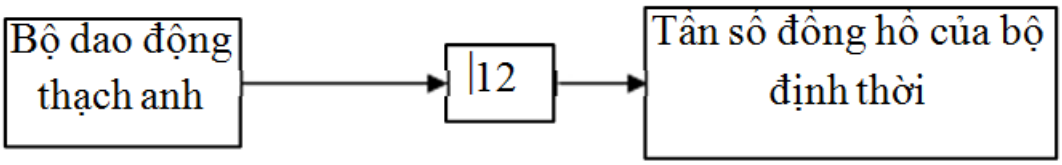
c. Nguồn xung đồng hồ cho bộ định thời

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:

Đoạn mã dưới đây trình bày tần số thạch anh cho 3 hệ thống dựa trên 8051 khác nhau. Hãy tìm chu kỳ máy của mỗi trường hợp: a) 11.0592MHz b) 16MHz và c) 20MHz.

Lời giải:



a) 11.0592/12 = 921.6kHz; Chu kỳ máy là 1/921.6kHz = 1.085μs (micro giây)

b) 16MHz/12 = 1.333MHz; Chu kỳ máy MC = 1/1.333MHz = 0.75μs

c) 20MHz/12 = 1.66MHz ⇒ MC = 1/1.66MHz = 0.60μs

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.

d. 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:

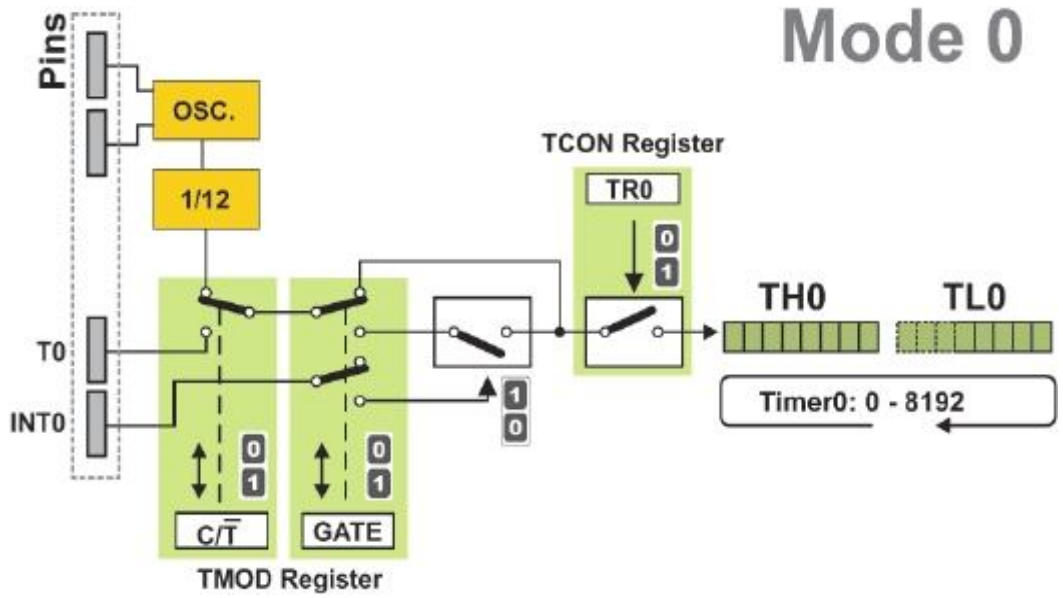
Tìm giá trị cho TMOD nếu ta muốn lập trình bộ Timer0 ở chế độ 2 sử dụng thạch anh XTAL 8051 làm nguồn đồng hồ và sử dụng các lệnh để khởi động và dừng bộ định thời.

Lời giải:

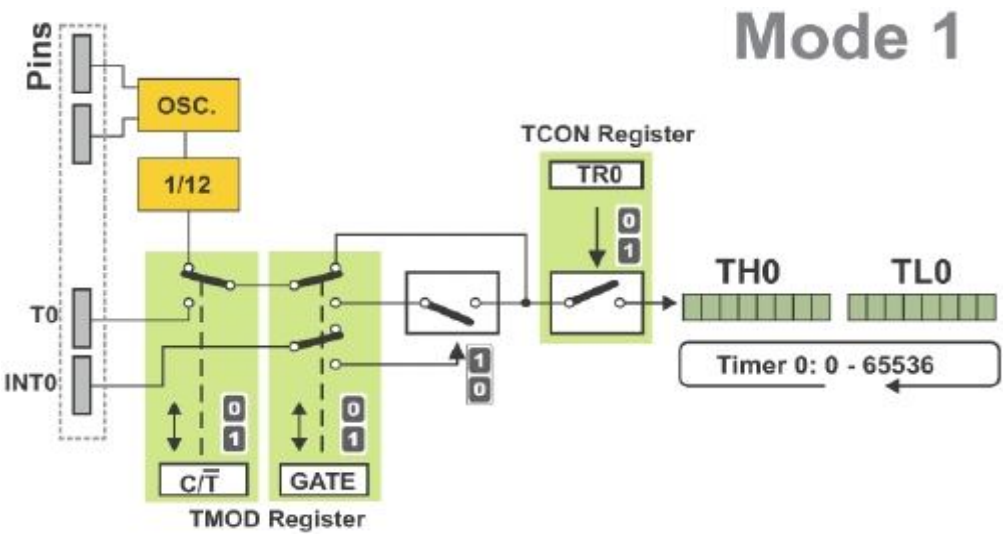
TMOD = 0000 0010: Bộ định thời Timer0, chế độ 2 C/T = 0 dùng nguồn XTAL GATE = 0 để dùng phần mềm trong để khởi động và dừng bộ định thời.

5.2. CÁC CHẾ ĐỘ CỦA BỘ ĐẾM / ĐỊNH THỜI (Timer Mode)

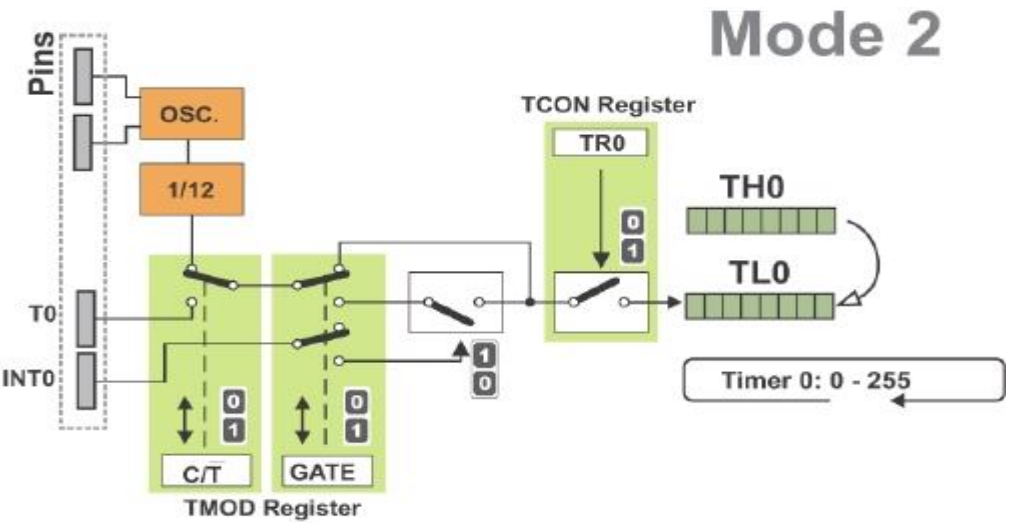
Như vậy, bây giờ chúng ta đã có hiểu biết cơ bản về vai trò của thanh ghi TMOD, chúng ta sẽ xét chế độ của bộ định thời và cách chúng được lập trình như thế nào để tạo ra một độ trễ thời gian. Do chế độ 1 và chế độ 2 được sử dụng rộng rãi nên ta đi xét chi tiết từng chế độ một.



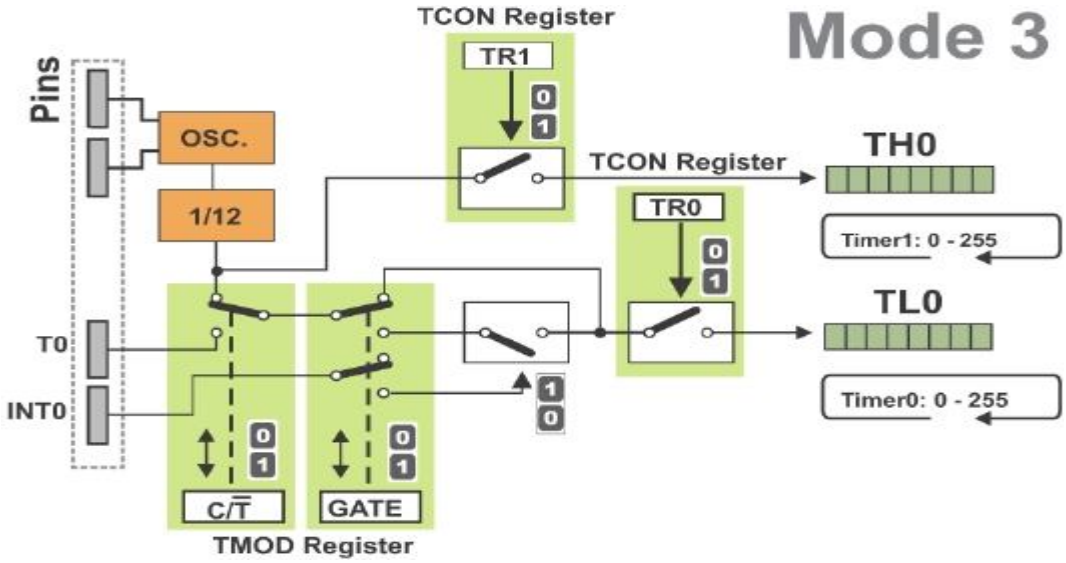
Hình 5-4. Timer 0 – Mode 0



Hình 5-5. Timer 0 – Mode 1



Hình 5-6. Timer 0 – Mode 2



Hình 5-7. Timer 0 – Mode 3

5.3. NGẮT TIMER

Các ngắt timer có địa chỉ Vector ngắt là 000BH (timer 0) và 001BH (timer 1). Ngắt timer xảy 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:

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 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

Cũng để ý rằng phần cao và phần thấp của xung sóng vuông là bằng nhau.

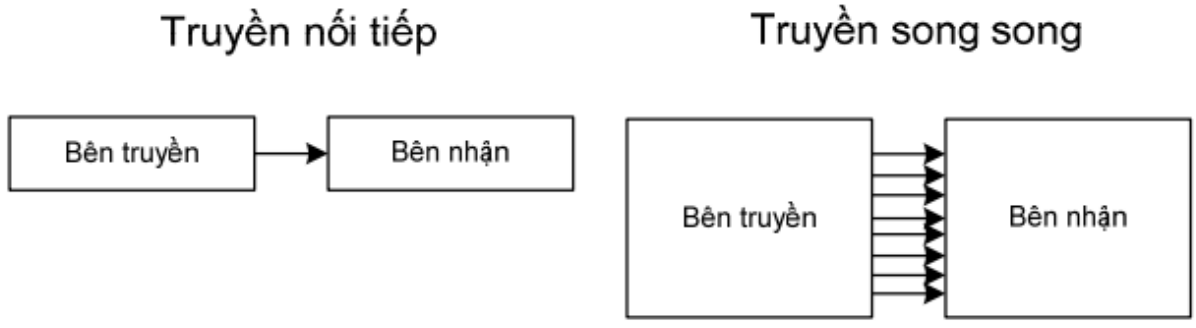
Trong tính toán trên đây là chưa kể đến tổng phí các lệnh vòng lặp.

CHƯƠNG 6
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 dudx 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.

6.1. 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 6-1. 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.

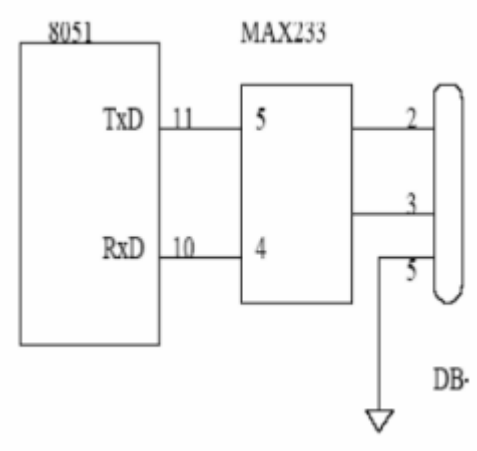
8051 có hai chân được dùng chuyên cho truyền và nhận dữ liệu nối tiếp. Hai chân này được gọi là TxD và RxD và là một phần của cổng P3 (đó là P3.0 và P3.1). chân 11 của 8051 là P3.1 được gán cho TxD và chân 10 (P3.0) được dùng cho RxD. Các chân này tương thích với mức lô-gích TTL. Do vậy chúng đòi hỏi 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.

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

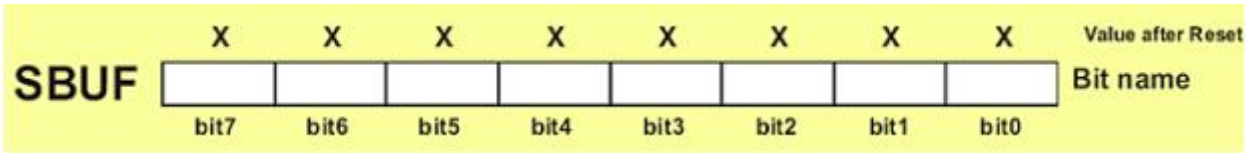
cần phải cấu hình cho cổng này làm việc thông qua các thanh ghi điều khiển và ghép nối tương thích với chuẩn rs232.



Hình 6-2. Ghép nối RS232 với 8051

6.2. CÁC THANH GHI ĐIỀU KHIỂN TRUYỀN THÔN

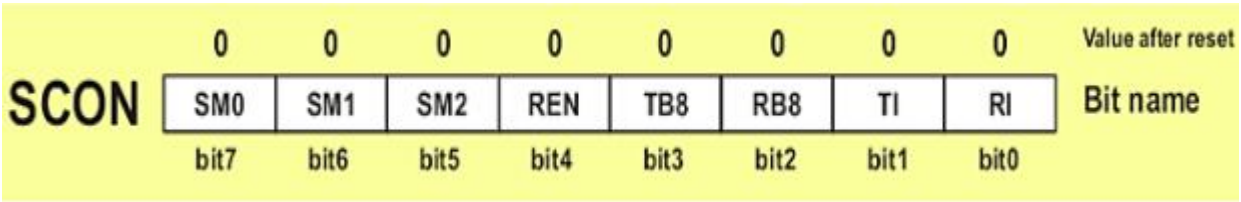
6.2.1. SBUF: Vùng đệm truyền thông dữ liệu ra/vào cổng nối tiếp.



Hình 6-3. Thanh ghi SBUF

- Việc truyền dữ liệu tương ứng với việc nạp cho SBUF một giá trị
- Dữ liệu nhận từ RxD cũng được lưu vào SBUF

6.2.2. SCON: Thanh ghi điều khiển hoạt động cổng nối tiếp



Hình 6-4. Thanh ghi SCON

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

Bảng 6-1. Các bít của thanh SCON

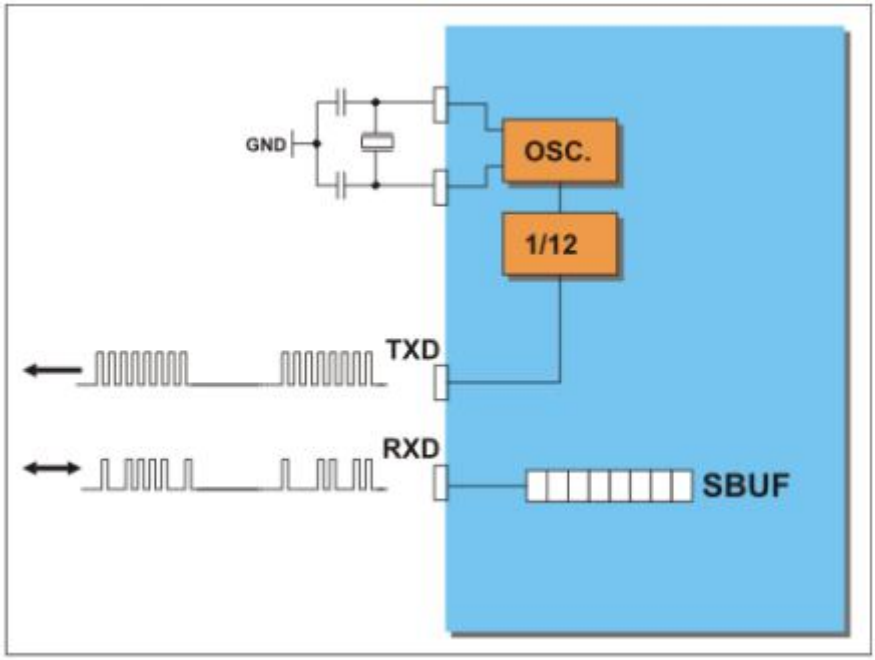
6.3. LỰA CHỌN CHẾ ĐỘ TRUYỀN THÔNG

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

Bảng 6-2. Lựa chọn chế độ làm việc

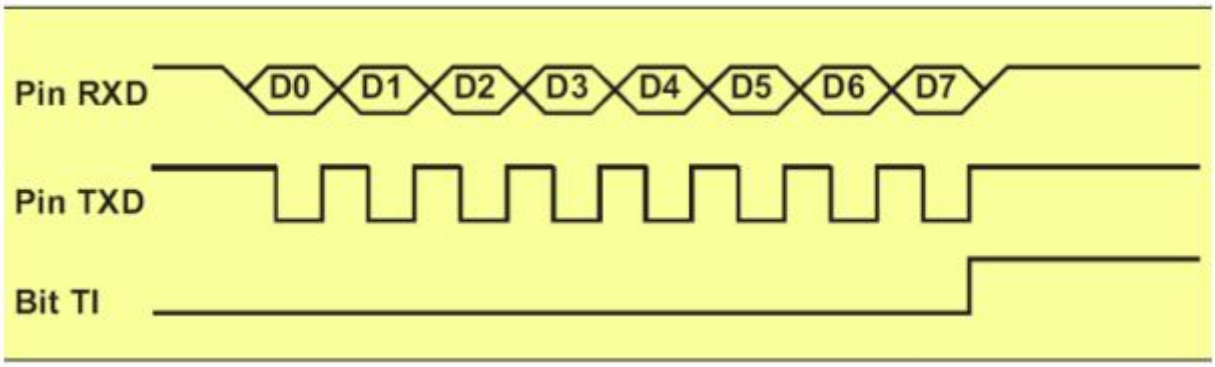
6.3.1. Mode 0

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.



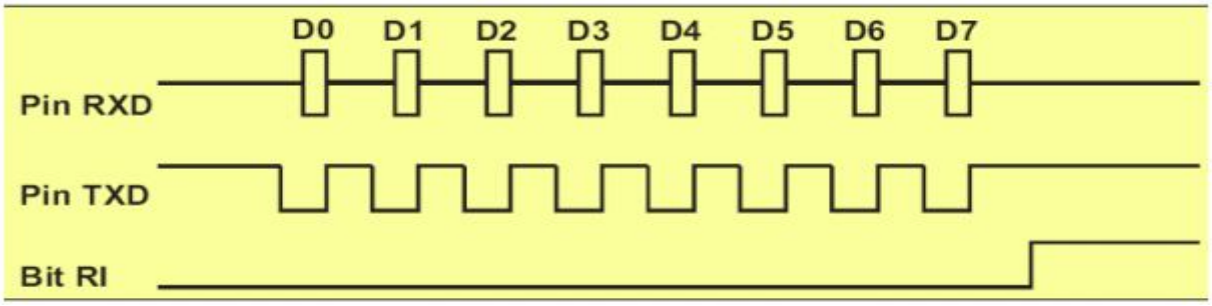
Hình 6-5. Truyền thông nối tiếp – Mode 0

- Quá trình truyền bắt đầu khi ghi giá trị vào SBUF, kết thúc được báo qua TI



Hình 6-6. Giản đồ thời gian truyền nối tiếp – Mode 0

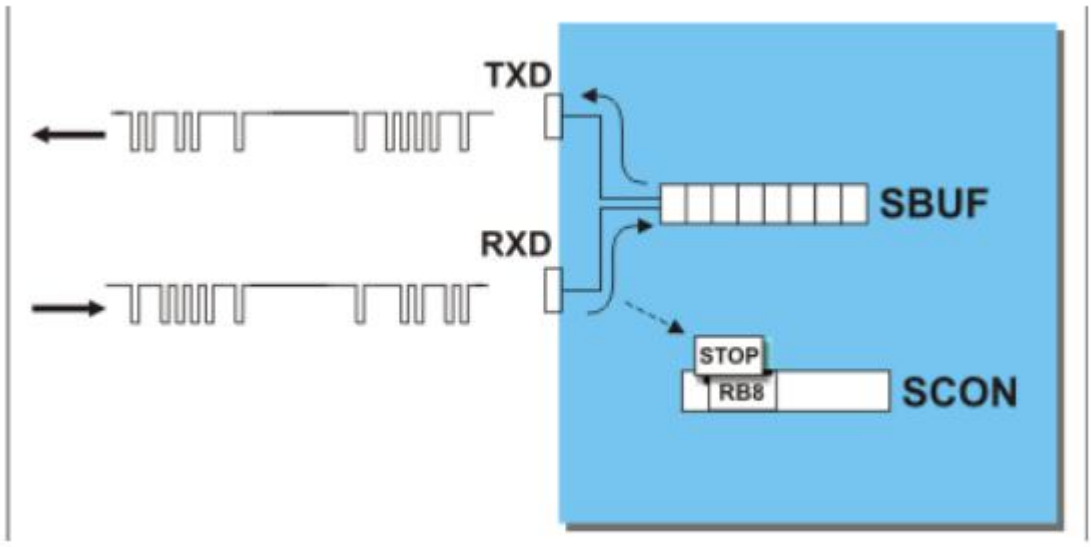
- Quá trình nhận tự động bởi hệ thống và kết thúc khi RI=1



Hình 6-7. Giản đồ thời gian nhận nối tiếp – Mode 0

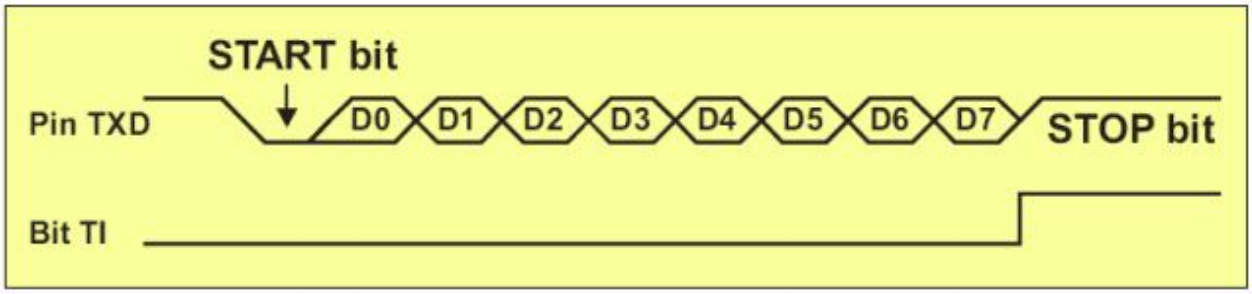
6.3.2. Mode 1

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



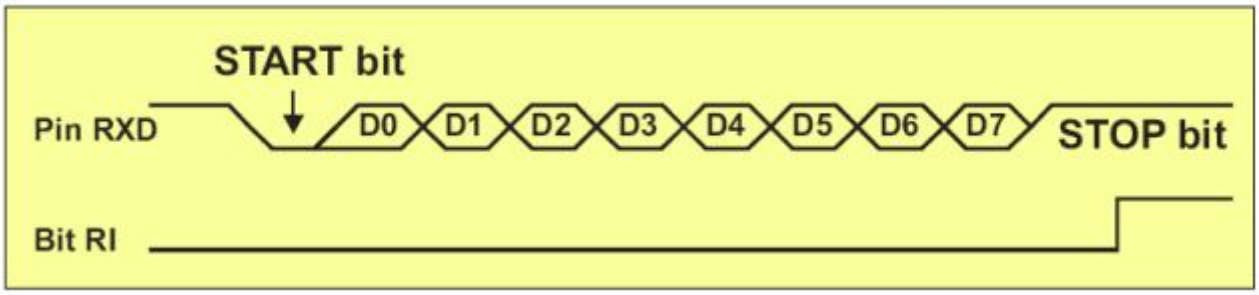
Hình 6-8. Truyền nhận nối tiếp – Mode 1

- Quá trình truyền:



Hình 6-9. Giản đồ thời gian truyền nối tiếp – Mode 1

- Quá trình nhận



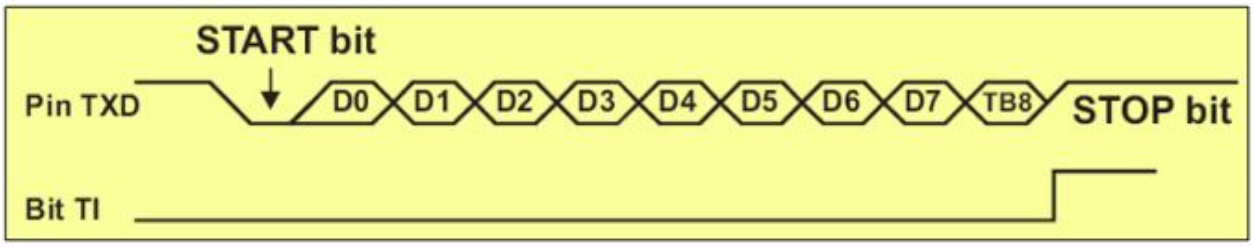
Hình 6-10. Giản đồ thời gian nhận nối tiếp – Mode 1

6.3.3. 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

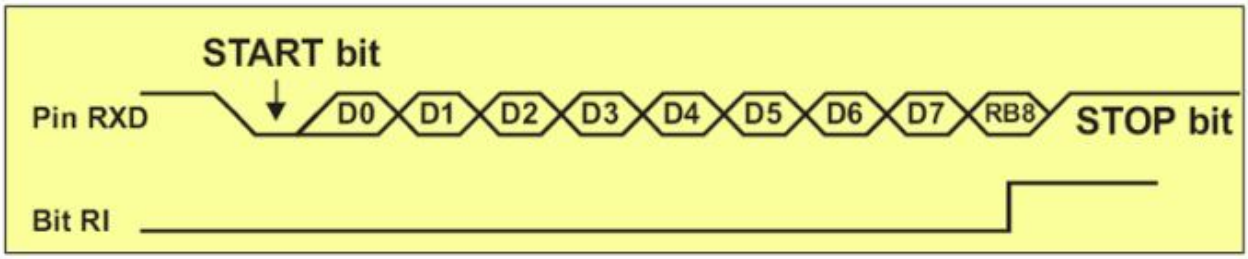
truyền, RxD nhận dữ liệu, tốc độ truyền cài đặt qua Timer 1. Bit thứ 9 thường được dùng là bit phát hiện lỗi party.

- Quá trình truyền



Hình 6-11. Giản đồ thời gian truyền nối tiếp – Mode 2

- Quá trình nhận:



Hình 6-12. Giản đồ thời gian nhận nối tiếp – Mode 2

6.3.4. Mode 3

Mode 3 tương tự mode 2 về mọi mặt ngoại trừ tốc độ baud

Tốc độ Baud

Trong một số hoạt mode 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 hoặt động và cho phép ngắt tràn timer 1
- Cấu hình cho timer 1 làm việc ở chế độ tự nạp lại

Công thức tính:

$$Baud\_Rate = \frac{2^{SMOD}.F_{xtal}}{6^{(1-SPD)}.12.32.\left[256-(BRL)\right]}$$

$$(BRL) = 256 - \frac{2^{SMOD}.F_{xtal}}{6^{(1-SPD)}.12.32.Baud\_Rate}$$

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 6-3. Một số giá trị thường dùng trong truyền thông nối tiếp

6.4. 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.

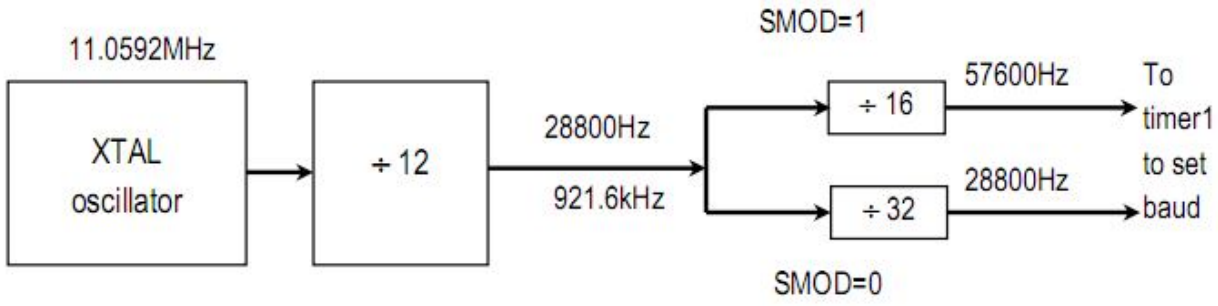
a) 9600 b) 4800 nếu SMOD = 1 và tần số XTAL = 11.0592MHz

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?

CHƯƠNG 7
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.

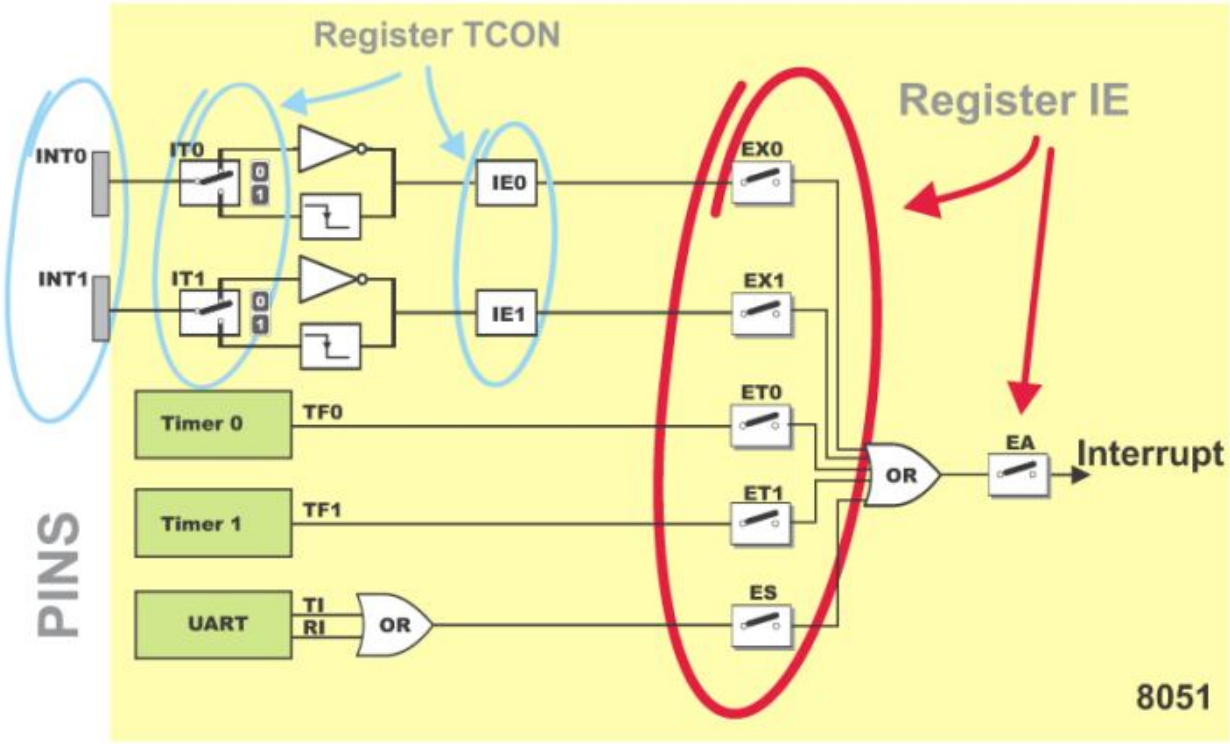
Ví dụ trong các bộ định thời, ta đã dùng lệnh "JNB TF, đích" và đợi cho đến khi bộ định thời quay trở về 0. Trong ví dụ đó, trong khi chờ đợi thì ta có thể làm việc được gì khác có ích hơn, chẳng hạn như khi sử dụng phương pháp ngắt thì bộ vi điều khiển có thể đi làm các việc khác và khi cờ TF bật lên nó sẽ ngắt bộ vi điều khiển cho dù nó đang làm bất kỳ điều gì.

7.1. 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.

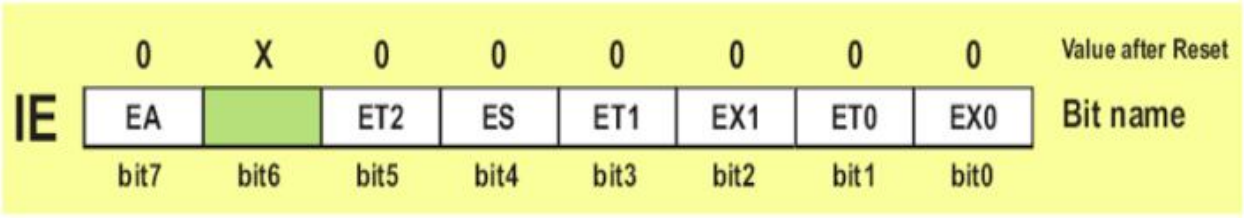
Các tín hiệu điều khiển ngắt có thể được mô tả như hình dưới



Hình 7-1. 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



Hình 7-2. Thanh ghi điều khiển ngắt

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

Bảng 7-1. Các bít của thanh ghi diều khiển ngắt

7.2. 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	

Bảng 7-2. Bảng vector ngắt và ví dụ

7.3. 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.

7.4. 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 dộng 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:

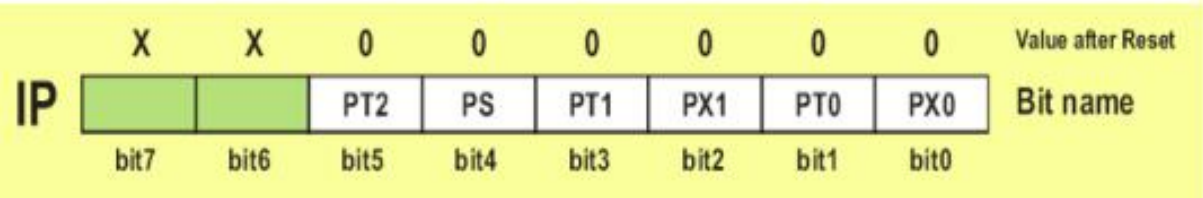
+ Nếu 1 có độ ưu tiên cao hơn một ngắt đang được xử lý xuất hiện thì, ngắt có ưu tiên thấp ngay lập tức bị dừng để ngắt kia được thực hiện

+ 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



Hình 7-3. 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

CHƯƠNG 8
PHỐI GHÉP 8051 VỚI THẾ GIỚI THỰC

Chương này khám phá một số ứng dụng của 8051 với thế giới thực. Chúng ta giải thích làm cách nào phối ghép 8051 với các thiết bị như là LCD, ADC và các cảm biến.

8.1. PHỐI GHÉP VỚI LCD

Ở phần này ta sẽ mô tả các chế độ hoạt động của các LCD và sau đó mô tả cách lập trình và phối ghép một LCD tới 8051.

8.1.1. Hoạt động của LCD.

Trong những năm gần đây LCD đang ngày càng được sử dụng rộng rãi thay thế dần cho các đèn LED (các đèn LED 7 đoạn hay nhiều đoạn). Đó là vì các nguyên nhân sau:

1. Các LCD có giá thành hạ.
2. Khả năng hiển thị các số, các ký tự và đồ hoạ tốt hơn nhiều so với các đèn LED (vì các đèn LED chỉ hiển thị được các số và một số ký tự).
3. Nhờ kết hợp một bộ điều khiển làm tươi vào LCD làm giải phóng cho CPU công việc làm tươi LCD. Trong khi đèn LED phải được làm tươi bằng CPU (hoặc bằng cách nào đó) để duy trì việc hiển thị dữ liệu.
4. Dễ dàng lập trình cho các ký tự và đồ hoạ.

8.1.2. Mô tả các chân của LCD.

LCD được nói trong mục này có 14 chân, chức năng của các chân được cho trong bảng 12.1. Vị trí của các chân được mô tả trên bảng 6.1 cho nhiều LCD khác nhau.

1. Chân V_{CC}, V_{SS} và V_{EE}: Các chân V_{CC}, V_{SS} và V_{EE}: Cấp dương nguồn - 5v và đất tương ứng thì V_{EE} được dùng để điều khiển độ tương phản của LCD.
2. Chân chọn thanh ghi RS (Register Select).

Có hai thanh ghi rất quan trọng bên trong LCD, chân RS được dùng để chọn các thanh ghi này như sau: Nếu RS = 0 thì thanh ghi mà lệnh được chọn để cho phép người dùng gửi một lệnh chẳng hạn như xoá màn hình, đưa con trỏ về đầu dòng v.v… Nếu RS = 1 thì thanh ghi dữ liệu được chọn cho phép người dùng gửi dữ liệu cần hiển thị trên LCD.

3. Chân đọc/ ghi (R/W).

Đầu vào đọc/ ghi cho phép người dùng ghi thông tin lên LCD khi R/W = 0 hoặc đọc thông tin từ nó khi R/W = 1.

4. Chân cho phép E (Enable).

Chân cho phép E được sử dụng bởi LCD để chốt thông tin hiện hữu trên chân dữ liệu của nó. Khi dữ liệu được cấp đến chân dữ liệu thì một xung mức cao xuống thấp phải được áp đến chân này để LCD chốt dữ liệu trên các chân dữ liêu. Xung này phải rộng tối thiểu là 450ns.

5. Chân D0 - D7.

Đây là 8 chân dữ liệu 8 bít, được dùng để gửi thông tin lên LCD hoặc đọc nội dung của các thanh ghi trong LCD.

Để hiển thị các chữ cái và các con số, chúng ta gửi các mã ASCII của các chữ cái từ A đến Z, a đến f và các con số từ 0 - 9 đến các chân này khi bật RS = 1.

Cũng có các mã lệnh mà có thể được gửi đến LCD để xoá màn hình hoặc đưa con trỏ về đầu dòng hoặc nhấp nháy con trỏ. Bảng 6.2 liệt kê các mã lênh.

Chúng ta cũng sử dụng RS = 0 để kiểm tra bít cờ bận để xem LCD có sẵn sàng nhân thông tin. Cờ bận là D7 và có thể đượcđọc khi R/W = 1 và RS = 0 như sau:

Nếu R/W = 1, RS = 0 khi D7 = 1 (cờ bận 1) thì LCD bận bởi các công việc bên trong và sẽ không nhận bất kỳ thông tin mới nào. Khi D7 = 0 thì LCD sẵn sàng nhận thông tin mới. Lưu ý chúng ta nên kiểm tra cờ bận trước khi ghi bất kỳ dữ liệu nào lên LCD.

Chân	Ký hiệu	I/O	Mô tả
1	V_{SS}	-	Đất
2	V_{CC}	-	Dương nguồn 5v
3	V_{EE}	-	Cấp nguồn điều khiển phản
4	RS	I	RS = 0 chọn thanh ghi lệnh. RS = 1 chọn thanh dữ liệu
5	R/W	I	R/W = 1 đọc dữ liệu. R/W = 0 ghi
6	E	I/O	Cho phép
7	DB0	I/O	Các bít dữ liệu
8	DB1	I/O	Các bít dữ liệu
9	DB2	I/O	Các bít dữ liệu

10	DB3	I/O	Các bít dữ liệu
11	DB4	I/O	Các bít dữ liệu
12	DB5	I/O	Các bít dữ liệu
13	DB6	I/O	Các bít dữ liệu
14	DB7	I/O	Các bít dữ liệu

Bảng 8-1. Mô tả các chân của LCD.

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

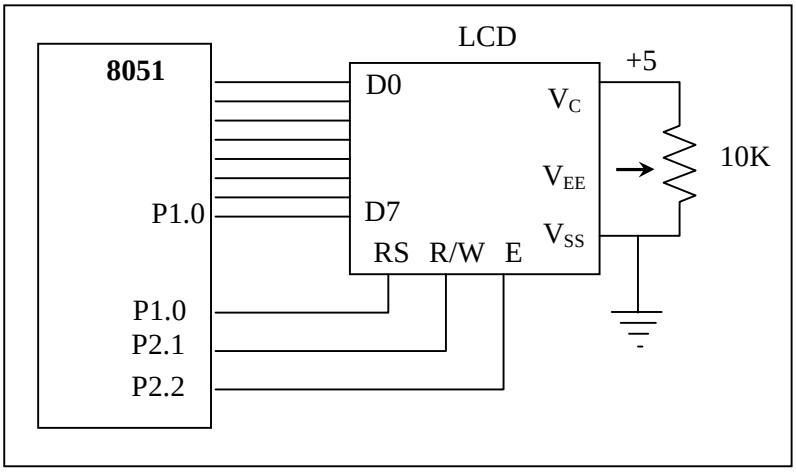
Bảng 8-2. Các mã lệnh LCD.

8.1.3 Gửi các lệnh và dữ liệu đến LCD với một độ trễ.

Để gửi một lệnh bất kỳ từ bảng 6.2 đến LCD ta phải đưa chân RS về 0. Đối với dữ liệu thì bật RS = 1 sau đó gửi một sườn xung cao xuống thấp đến chân E để cho phép chốt dữ liệu trong LCD. Điều này được chỉ ra trong đoạn mã chương trình dưới đây.

```
; gọi độ thời gian trễ trước khi gửi dữ liệu/ lệnh kế tiếp.
; chân P1.0 đến P1.7 được nối tới chân dữ dữ liệu D0 - D7 của  LCD.
; Chân P2.0 được nối tới chân RS của LCD.
; Chân P2.1 được nối tới chân R/W của LCD.
; Chân P2.2 được nối đến chân E của LCD.
        ORG
        MOV  A, # 38H          ; Khởi tạo LCD hai dòng với ma trận 5 × 7
        ACALL COMNWRT          ; Gọi chương trình con lệnh
        ACALL DELAY            ; Cho LCD một độ trễ
        MOV  A, # 0EH          ; Hiển thị màn hình và con trỏ
        ACALL COMNWRT          ; Gọi chương trình con lênh
        ACALL  DELAY           ; Cấp một độ trễ cho LCD
        MOV  AM # 01           ; Xoá LCD
        ACALL COMNWRT          ; Gọi chương trình con lệnh
        ACALL DELAY            ; Tạo độ trễ cho LCD
        MOV  A, # 06H          ; Dịch con trỏ sang phải
        ACALL COMNWRT          ; Gọi chương trình con lệnh
        ACALL DELAY            ; Tạo độ trễ cho LCD
        MOV  AM # 48H          ; Đưa con trỏ về dòng 1 cột 4
        ACALL COMNWRT          ; Gọi chương trình con lệnh
        ACALL DELAY            ; Tạo độ trễ cho LCD
        MOV  A, # “N”          ; Hiển thị chữ N
        ACALL DATAWRT          ; Gọi chương trình con hiển thị DISPLAY
        ACALL DELAY            ; Tạo độ trễ cho LCD
        MOV  AM # “0”          ; Hiển thị chữ 0
        ACALL DATAWRT          ; Gọi DISPLAY
AGAIN:  SJMP  AGAIN            ; Chờ ở đây
COMNWRT:                       ; Gửi lệnh đến LCD
        MOV     P1, A          ; Sao chép thanh ghi A đến cổng P1
        CLR     P2.0           ; Đặt RS = 0 để gửi lệnh
```

```
                CLR     P2.1        ; Đặt R/W = 0 để ghi dữ liệu
                SETB    P2.2        ; Đặt E = 1 cho xung cao
                CLR     P2.2        ; Đặt E = 0 cho xung cao xuống thấp
                RET
DATAWRT:                            ; Ghi dữ liệu ra LCD
                MOV     P1, A       ; Sao chép thanh ghi A đến cổng P1
                SETB    P2.0        ; Đặt RS = 1 để gửi dữ liệu
                CLR     P2.1        ; Đặt R/W = 0 để ghi
                SETB    P2.2        ; Đặt E = 1 cho xung cao
                CLR     P2.2        ; Đặt E = 0 cho xung cao xuống thấp
                RET
DELAY:  MOV  R3, # 50               ; Đặt độ trễ 50µs hoặc cao hơn cho CPU
HERE2:  MOV  R4, # 255              ; Đặt R4 = 255
HERE:   DJNZ R4, HERE               ; Đợi ở đây cho đến khi R4 = 0
        DJNZ R3, HERE2
        RET
        END
```



Hình 8-1. Ghép Nối LCD.

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

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

```
        ; Kiểm tra cờ bận trước khi gửi dữ liệu, lệnh ra LCD
        ; Đặt P1 là cổng dữ liệu
        ; Đặt P2.0 nối tới cổng RS
        ; Đặt P2.1 nối tới chân R/W
        ; Đặt P2.2 nối tới chân E
              ORG
              MOV  A, # 38H          ; Khởi tạo LCD hai dòng với ma trận 5 × 7
              ACALL COMMAND          ; Xuất lệnh
              MOV  A, # 0EH          ; Dịch con trỏ sang phải
              ACALL COMMAND          ; Xuất lệnh
              MOV  A, # 01H          ; Xoá lệnh LCD
              ACALL COMMAND          ; Xuất lệnh
              MOV  A, # 86H          ; Dịch con trỏ sang phải
              ACALL COMMAND          ; Đưa con trỏ về dòng 1 lệnh 6
              MOV  A, # “N”          ; Hiển thị chữ N
              ACALL DATA DISPLAY
              MOV  A, # “0”          ; Hiển thị chữ 0
              ACALL DATA DISPLAY
HERE:         SJMP  HERE             ; Chờ ở đây
COMMAND:ACALL READY                  ; LCD đã sẵn sàng chưa?
              MOV  P1, A             ; Xuất mã lệnh
              CLR   P2.0             ; Đặt RS = 0 cho xuất lệnh
              CLR   P2.1             ; Đặt R/W = 0 để ghi dữ liệu tới LCD
              SETB P2.2              ; Đặt E = 1 đối với xung cao xuống thấp
              CLR   P2.2             ; Đặt E = 0 chốt dữ liệu
              RET
DATA-DISPLAY:
              ACALL READY            ; LCD đã sẵn sàng chưa?
              MOV  P1, A             ; Xuất dữ liệu
              SETB P2.0              ; Đặt RS = 1 cho xuất dữ liệu
              CLR   P2.1             ; Đặt R/W = 0 để ghi dữ liệu ra LCD
              SETB P2.2              ; Đặt E = 1 đối với xung cao xuống thấp
              CLR   P2.2             ; Đặt E = 0 chốt dữ liệu
              RET
DELAY:
```

```
        SETB P1.7           ; Lấy P1.7 làm cổng vào
        CLR  P2.0           ; Đặt RS = 0 để truy cập thanh ghi lệnh
        SETB P2.1           ; Đặt R/W = 1 đọc thanh ghi lệnh
; Đọc thanh ghi lệnh và kiểm tra cờ lệnh
BACK:   CLR  P2.2           ; E = 1 đối với xung cao xuống thấp
        SETB P2.2           ; E = 0 cho xung cao xuống thấp?
        JB P1.7, BACK       ; Đợi ở đây cho đến khi cờ bận = 0
        RET
        END
```

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

8.2. PHỐI GHÉP VỚI ADC.

Phần này sẽ khám phá ghép các chíp ADC (bộ chuyển đổi tương tự số) và các cảm biến nhiệt với 8051.

8.2.1. Các thiết bị ADC.

Các bộ chuyển đổi ADC thuộc trong những thiết bị được sử dụng rộng rãi nhất để thu dữ liệu. Các máy tính số sử dụng các giá trị nhị phân, nhưng trong thế giới vật lý thì mọi đại lượng ở dạng tương tự (liên tục). Nhiệt độ, áp suất (khí hoặc chất lỏng), độ ẩm và vận tốc và một số ít trọng những đại lượng vật lý của thế giới thực mà ta gặp hàng ngày. Một đại lượng vật lý được chuyển về dòng điện hoặc điện áp qua một thiết bị được gọi là các bộ biến đổi. Các bộ biến đổi cũng có thể được coi như các bộ cảm biến. Mặc dù chỉ có các bộ cảm biến nhiệt, tốc độ, áp suất, ánh sáng và nhiều đại lượng tự nhiên khác nhưng chúng đều cho ra các tín hiệu dạng dòng điện hoặc điện áp ở dạng liên tục. Do vậy, ta cần một bộ chuyển đổi tương tự số sao cho bộ vi điều khiển có thể đọc được chúng. Một chíp ADC được sử dụng rộng rãi là ADC 804.

8.2.2. Chíp ADC 0804.

Chíp ADC 804 là bộ chuyển đổi tương tự số trong họ các loạt ADC 800 từ hãng National Semiconductor. Nó cũng được nhiều hãng khác sản xuất, nó làm việc với +5v và có độ phân giải là 8 bít. Ngoài độ phân giải thì thời gian chuyển đổi cũng là một yếu tố quan trọng khác khi đánh giá một bộ ADC. Thời gian chuyển đổi được định

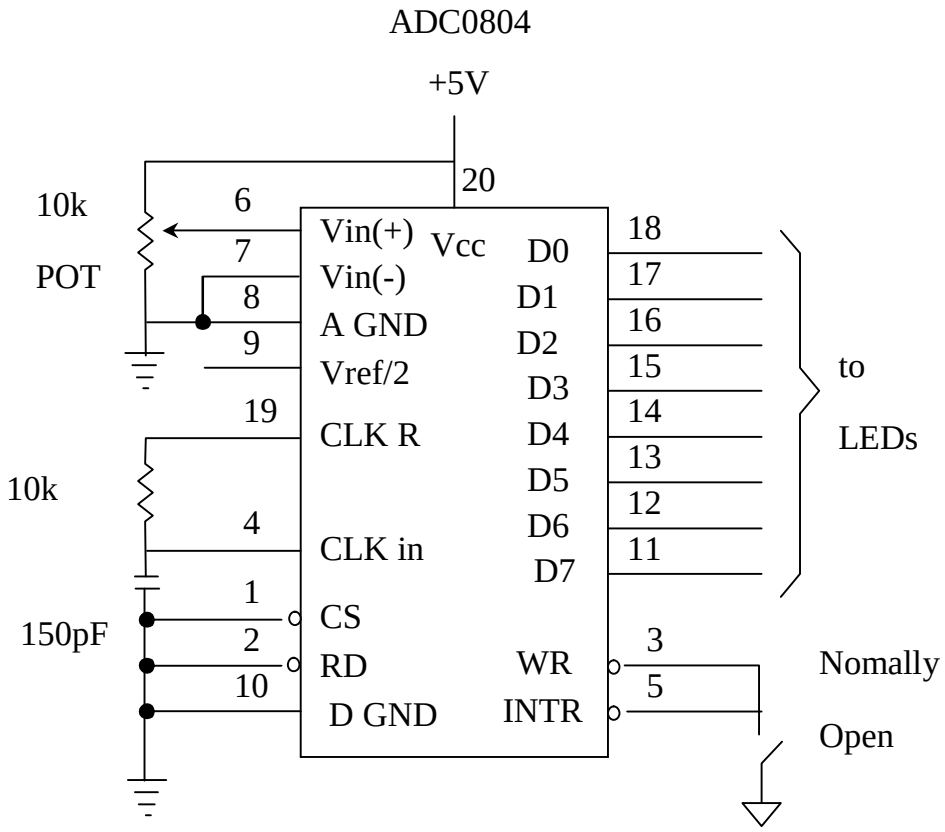
nghĩa như là thời gian mà bộ ADC cần để chuyển một đầu vào tương tự thành một số nhị phân. Trong ADC 804 thời gian chuyển đổi thay đổi phụ thuộc vào tần số đồng hồ được cấp tới chân CLK và CLK IN nhưng không thể nhanh hơn 110µs. Các chân của ADC 804 được mô tả như sau:

1. Chân $\overline{CS}$ - chọn chíp: Là một đầu vào tích cực mức thấp được sử dụng để kích hoạt chíp ADC 804. Để truy cập ADC 804 thì chân này phải ở mức thấp.
2. Chân $\overline{RD}$ (đọc): Đây là một tín hiệu đầu vào được tích cực mức thấp. Các bộ ADC chuyển đổi đầu vào tương tự thành số nhị phân tương đương với nó và giữ nó trong một thanh ghi trong. $\overline{RD}$ được sử dụng để nhận dữ liệu được chuyển đổi ở đầu ra của ADC 804. Khi CS = 0 nếu một xung cao - xuống - thấp được áp đến chân $\overline{RD}$ thì đầu ra số 8 bít được hiển diện ở các chân dữ liệu D0 - D7. Chân $\overline{RD}$ cũng được coi như cho phép đầu ra.
3. Chân ghi $\overline{WR}$ (thực ra tên chính xác là "Bắt đầu chuyển đổi"). Đây là chân đầu vào tích cực mức thấp được dùng để báo cho ADC 804 bắt đầu quá trình chuyển đổi. Nếu CS = 0 khi $\overline{WR}$ tạo ra xung cao - xuống - thấp thì bộ ADC 804 bắt đầu chuyển đổi giá trị đầu vào tương tự V_{in} về số nhị phân 8 bít. Lượng thời gian cần thiết để chuyển đổi thay đổi phụ thuộc vào tần số đưa đến chân CLK IN và CLK R. Khi việc chuyển đổi dữ liệu được hoàn tất thì chân INTR được ép xuống thấp bởi ADC 804.
4. Chân CLK IN và CLK R.

Chân CLK IN là một chân đầu vào được nối tới một nguồn đồng hồ ngoài khi đồng hồ ngoài được sử dụng để tạo ra thời gian. Tuy nhiên 804 cũng có một máy tạo xung đồng hồ. Để sử dụng máy tạo xung đồng hồ trong (cũng còn được gọi là máy tạo đồng hồ riêng) của 804 thì các chân CLK IN và CLK R được nối tới một tụ điện và một điện trở như chỉ ra trên hình 6.4. Trong trường hợp này tần số đồng hồ được xác định bằng biểu thức:

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

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



Hình 8-2. Kiểm tra ADC 0804 ở chế độ chạy tự do.

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

Đây là chân đầu ra tích cực mức thấp. Bình thường nó ở trạng thái cao và khi việc chuyển đổi hoàn tất thì nó xuống thấp để báo cho CPU biết là dữ liệu được chuyển đổi sẵn sàng để lấy đi. Sau khi $\overline{\text{INTR}}$ xuống thấp, ta đặt CS = 0 và gửi một xung cao 0 xuống - thấp tới chân $\overline{\text{RD}}$ lấy dữ liệu ra của 804.

6. Chân V_{in} (+) và V_{in} (-).

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

7. Chân V_{CC}.

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

8. Chân $V_{ref/2}$.

Chân 9 là một điện áp đầu vào được dùng cho điện áp tham chiếu. Nếu chân này hở (không được nối) thì điện áp đầu vào tương tự cho ADC 804 nằm trong dải 0 đến +5v (giống như chân V_{CC}). Tuy nhiên, có nhiều ứng dụng mà đầu vào tương tự áp

đến V_{in} cần phải khác ngoài dải 0 đến 5v. Chân $V_{ref/2}$ đượcdùng để thực thi các điện áp đầu vào khác ngoài dải 0 - 5v. Ví dụ, nếu dải đầu vào tương tự cần phải là 0 đến 4v thì $V_{ref/2}$ được nối với +2v.

V_{ref}/ 2(V)	V_{in}(V)	Step Size (mV)
Hở *	0 đến 5	5/256 = 19.53
2.0	0 đến 4	4/255 = 15.62
1.5	0 đến 3	3/256 = 11.71
1.28	0 đến 2.56	2.56/256 = 10
1.0	0 đến 2	2/256 = 7.81
0.5	0 đến 1	1/256 = 3.90

Bảng 8-3. Điện áp $Vref_{/2}$ liên hệ với dải V_{in}.

Ghi chú: - V_{CC} = 5V

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

9. Các chân dữ liệu D0 - D7.

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

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

10. Chân đất tương tự và chân đất số.

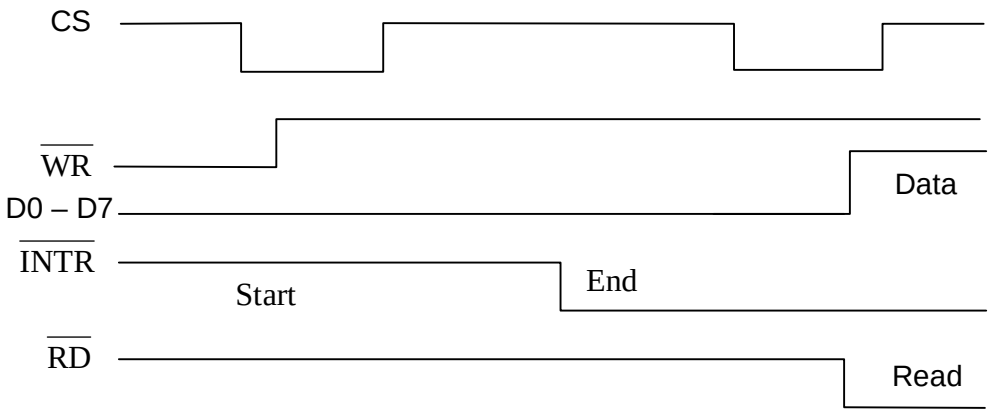
Đây là những chân đầu vào cấp đất chung cho cả tín hiệu số và tương tự. Đất tương tự được nối tới đất của chân V_{in} tương tự, còn đất số được nối tới đất của chân V_{cc}. Lý do mà ta phải có hai đất là để cách ly tín hiệu tương tự V_{in} từ các điện áp ký sinh tạo ra việc chuyển mạch số được chính xác. Trong phần trình bày của chúng ta thì các chân này được nối chung với một đất. Tuy nhiên, trong thực tế thu đo dữ liệu các chân đất này được nối tách biệt.

Từ những điều trên ta kết luận rằng các bước cần phải thực hiện khi chuyển đổi dữ liệu bởi ADC 804 là:

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

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

c) Sau khi chân $\overline{INTR}$ xuống thấp, ta bật CS = 0 và gửi một xung cao - xuống - thấp đến chân $\overline{RD}$ để lấy dữ liệu ra khỏi chíp ADC 804. Phân chia thời gian cho quá trình này được trình bày trên hình 6.6

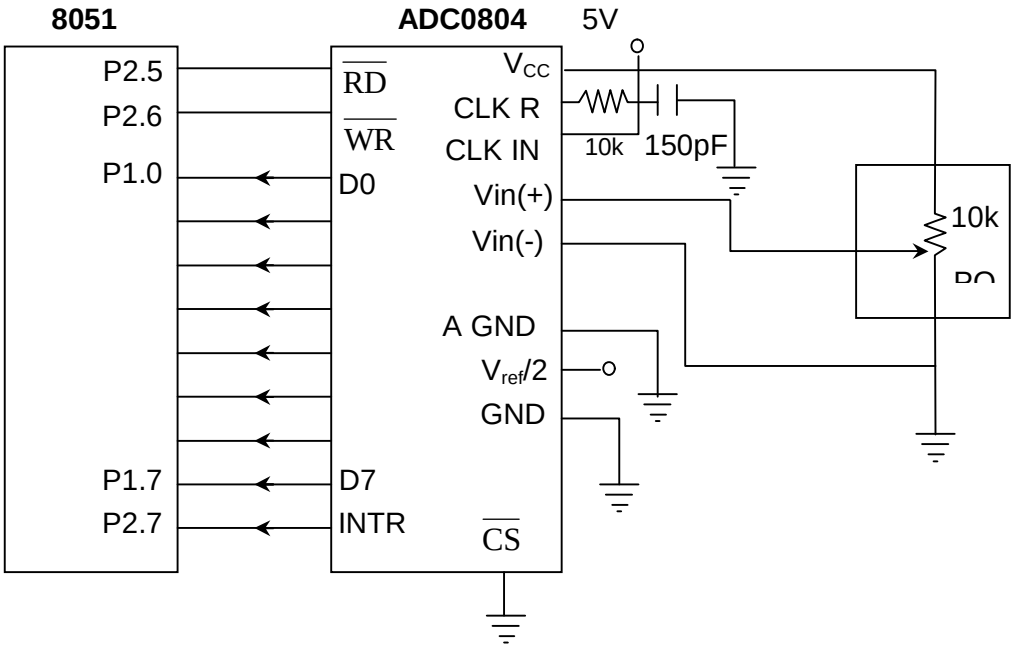


Hình 8-3. Phân chia thời gian đọc và ghi của ADC 804.

8.2.3. Ghép nối 8051 với ADC 0804.

Chúng ta có thể kiểm tra ADC 804 bằng cách sử dụng sơ đồ mạch trên hình 7-6. thiết lập này được gọi là chế độ kiểm tra chạy tự do và được nhà sản xuất khuyến cao nên sử dụng. Hình 6.4 trình bày một biến trở được dùng để cấp một điện áp tương tự từ 0 đến 5V tới chân đầu vào.

$V_{in}(+)$ của ADC 804 các đầu ra nhị phân được hiển thị trên các đèn LED của bảng huấn luyện số. Cần phải lưu ý rằng trong chế độ kiểm tra chạy tự do thì đầu vào CS được nối tới đất và đầu vào $\overline{WR}$ được nối tới đầu ra $\overline{INTR}$. Tuy nhiên, theo tài liệu của hãng National Semiconductor "nút WR và INTR phải được tạm thời đưa xuống thấp kế sau chu trình cấp nguồn để bảo đảm hoạt động".



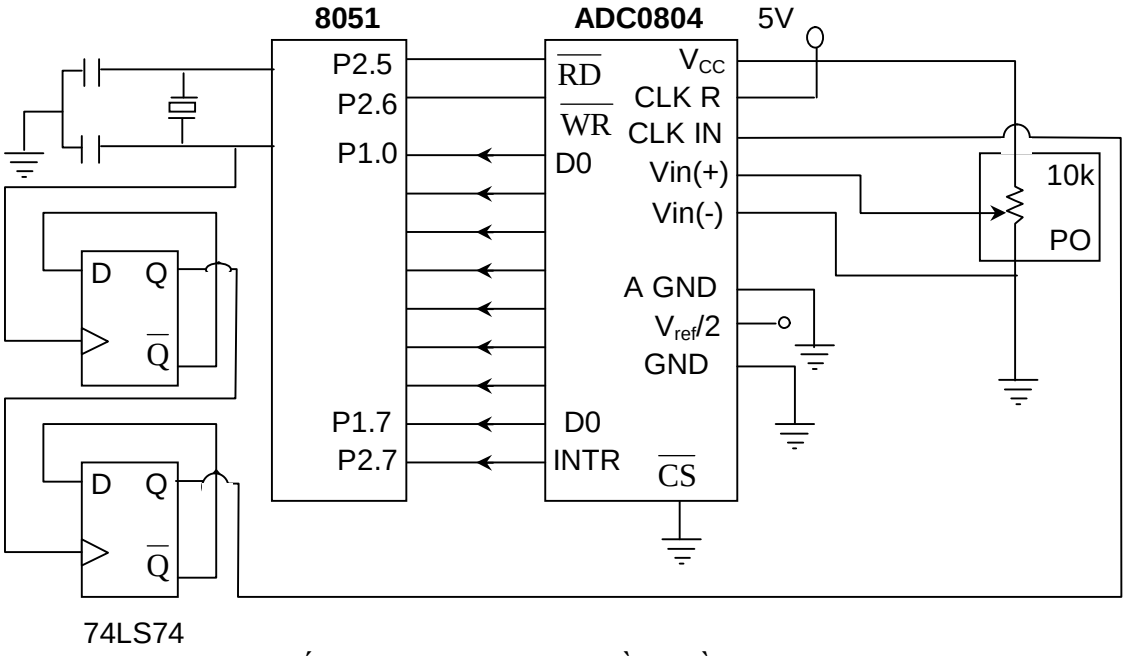
Hình 8-4. Nối ghép ADC 0804

Ví dụ:

Hãy thử nối ghép ADC 804 với 8051 theo sơ đồ 6.7. Viết một chương trình để hiển thị chân INTR và lấy đầu vào tương tự vào thanh ghi A. Sau đó gọi một chương trình chuyển đổi mã Hex ra ASCII và một chương trình hiển thị dữ liệu. Thực hiện điều này liên tục.

Lời giải:

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



Hình 8-5. Nối ghép ADC 804 với đồng hồ từ XTAL2 của 8051.

Trên hình 7-7 ta có thể thấy rằng tín hiệu đồng hồ đi vào ADC 804 là từ tần số thạch anh của 8051. Vì tần số này quá cao nên ta sử dụng hai mạch lật Rlip - Flop kiểu D (74LS74) để chia tần số này cho 4. Một mạch lật chia tần số cho 2 nếu ta nối đầu $\overline{Q}$ tới đầu vào D. Đối với tần số cao hơn thì ta cần sử dụng nhiều mạch Flip - Plop hơn.

PHỤ LỤC

Phụ lục A: Các ký hiệu sử dụng mô tả lệnh

Ký hiệu	Mô
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.
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).

Phụ lục B: Tập lệnh 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
2	ADD	A,direct	A = A + direct	2	1
3	ADD	A,@Ri	A = A + @Ri	1	1
4	ADD	A,#data	A = A + #data	2	1
5	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					
25	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
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 bít

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
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

Lệnh dich chuyển dữ liệu

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	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
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

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 nhãn nếu Rn#data	3	2
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

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

Vi Xử Lý 8051

Created by mercury

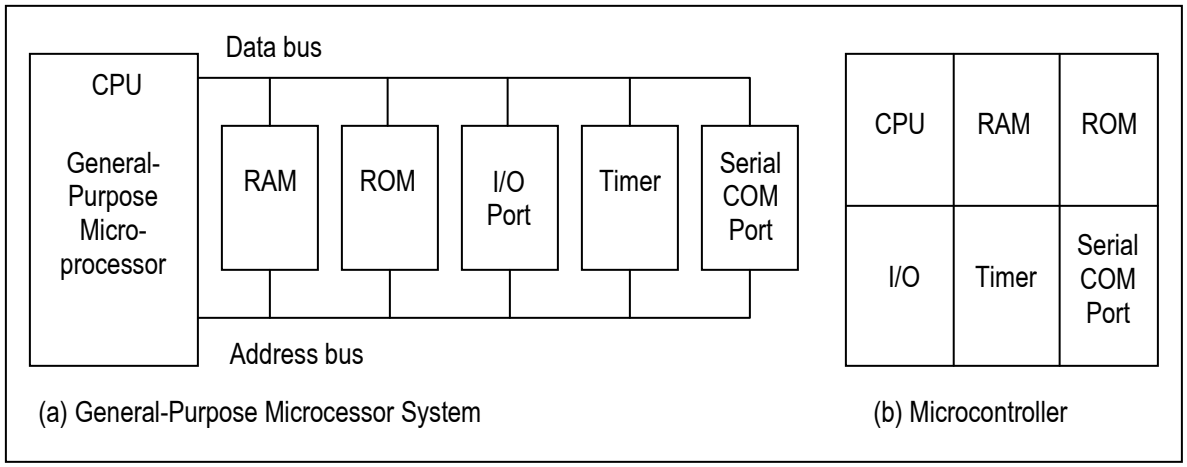
CHƯƠNG I
Các bộ vi điều khiển 8051

1.1 các bộ vi điều khiển và các bộ xử lý nhúng.

Trong mục này chúng ta bàn về nhu cầu đối với các bộ vi điều khiển (VĐK) và so sánh chúng với các bộ vi xử lý cùng dạng chung như Pentium và các bộ vi xử lý × 86 khác. Chúng ta cùng xem xét vai trò của các bộ vi điều khiển trong thị trường các sản phẩm nhúng. Ngoài ra, chúng ta cung cấp một số tiêu chuẩn về cách lựa chọn một bộ vi điều khiển như thế nào.

1.1.1 Bộ vi điều khiển so với bộ vi xử lý cùng dùng chung

Sự khác nhau giữa một bộ vi điều khiển và một bộ vi xử lý là gì? Bộ vi xử lý ở đây là các bộ vi xử lý công dung chung như họ Intell × 86 (8086, 80286, 80386, 80486 và Pentium) hoặc họ Motorola 680 × 0(68000, 68010, 68020, 68030, 68040 v.v...). Những bộ VXL này không có RAM, ROM và không có các cổng vào ra trên chíp. Với lý do đó mà chúng được gọi chung là các bộ vi xử lý công dụng chung.



Hình 1.1: Hệ thống vi xử lý được so sánh với hệ thống vi điều khiển.

a) Hệ thống vi xử lý công dụng chung

b) Hệ thống vi điều khiển

Một nhà thiết kế hệ thống sử dụng một bộ vi xử lý công dụng chung chẳng hạn như Pentium hay 68040 phải bổ xung thêm RAM , ROM, các cổng vào ra và các bộ định thời ngoài để làm cho chúng hoạt động được. Mặc dù việc bổ xung RAM, ROM và các cổng vào ra bên ngoài làm cho hệ thống cồng cềnh và đắt hơn, nhưng chúng có ưu điểm là linh hoạt chẳng hạn như người thiết kế có thể quyết định về số lượng RAM, ROM và các cổng vào ra cần thiết phù hợp với bài toán trong tầm tay của mình.

Điều này không thể có được đối với các bộ vi điều khiển. Một bộ vi điều khiển có một CPU (một bộ vi xử lý) cùng với một lượng cố định RAM, ROM, các cổng vào ra và một bộ định thời tất cả trên cùng một chíp. Hay nói cách khác là bộ xử lý, RAM, ROM các cổng vào ra và bộ định thời đều được nhúng với nhau trên một chíp; do vậy người thiết kế không thể bổ xung thêm bộ nhớ ngoài, cổng vào ra hoặc bộ định thời cho nó. Số lượng cố định của RAM, ROM trên chíp và số các cổng vào - ra trong các bộ vi điều khiển làm cho chúng trở nên lý tưởng đối với nhiều ứng dụng mà trong đó giá thành và không gian lại hạn chế. Trong nhiều ứng dụng, ví dụ một điều khiển TV từ xa thì không cần công suất tính toán của bộ vi sử lý 486 hoặc thậm chí như 8086. Trong rất nhiều ứng dụng thì không gian nó chiếm, công suất nó tiêu tốn và giá thành trên một đơn vị là những cân nhắc nghiêm ngặt hơn nhiều so với công suất tính toán. Những ứng dụng thường yêu cầu một số thao tác vào - ra để đọc các tín hiệu và tắt - mở những bit nhất định. Vì lý do này mà một số người gọi các bộ xử lý này là IBP ("Itty-Bitty-Processor"), (tham khảo cuốn "Good things in small packages are Generating Big product opportunities" do Rick Grehan viết trên tạp BYTE tháng 9.1994; WWW. Byte. Com để biết về những trao đổi tuyệt vời về các bộ vi điều khiển).

Điều thú vị là một số nhà sản xuất các bộ vi điều khiển đã đi xa hơn là tích hợp cả một bộ chuyển đổi ADC và các ngoại vi khác vào trong bộ vi điều khiển.

Bảng 1.1: Một số sản phẩm được nhúng sử dụng các bộ vi điều khiển

Thiết bị nội thất gia đình	**Văn phòng**	**ô tô**
Đồ điện trong nhà	Điện thoại	Máy tính hành trình

Máy đàm thoại Máy điện thoại Các hệ thống an toàn Các bộ mở cửa ga-ra xe Máy trả lời Máy Fax Máy tính gia đình Tivi Truyền hình cáp VCR Máy quy camera Điều khiển từ xa Trò chơi điện tử Điện thoại tổ ong Các nhạc cụ điện tử Máy khâu Điều khiển ánh sáng Máy nhắn tin Máy chơi Pootball Đồ chơi Các dụng cụ tập thể hình	Máy tính Các hệ thống an toàn Máy Fax Lò vi sóng Máy sao chụp Máy in lazer Máy in màu Máy nhắn tin	Điều khiển động cơ Túi đệm khí Thiết bị ABS Đo lường Hệ thống bảo mật Đíũu khiển truyền tin Giải trí Điều hoà nhiệt độ Điện thoại tổ ong Mở cửa không cần chìa khoá

1.1.2 Các bộ VĐK cho các hệ thống nhúng.

Trong tài liệu về các bộ vi xử lý ta thường thấy khái niệm hệ thống nhúng (Embeded system). Các bộ vi xử lý và các bộ vi điều khiển được sử dụng rộng rãi trong các sản phẩm hệ thống nhúng. Một sản phẩm nhúng sử dụng một bộ vi xử lý (hoặc một bộ vi điều khiển để thực hiện một nhiệm vụ và chỉ một mà thôi. Một máy in là một ví dụ về một việc nhúng vì bộ xử lý bên trong nó chỉ làm một việc đó là nhận dữ liệu và in nó ra. Điều này khác với một máy tình PC dựa trên bộ xử lý Pentium (hoặc một PC tương thích với IBM × 86 bất kỳ). Một PC có thể được sử dụng cho một số bất kỳ các trạm dịch vụ in, bộ đầu cuối kiểm kê nhà băng, máy chơi trò chơi điện tử, trạm dịch vụ mạng hoặc trạm đầu cuối mạng Internet. Phần mềm cho các ứng dụng khác nhau có thể được nạp và chạy. Tất nhiên là lý do hiển nhiên để một PC thực hiện hàng loạt các công việc là nó có bộ

nhớ RAM và một hệ điều hành nạp phần mềm ứng dụng thường được đốt vào trong ROM. Một máy tính PC × 86 chứa hoặc được nối tới các sản phẩm nhúng khác nhau chẳng hạn như bàn phím, máyin, Modem, bộ điều khiển đĩa, Card âm thanh, bộ điều khiển CD = ROM. Chuột v.v... Một nội ngoại vi này có một bộ vi điều khiển bên trong nó để thực hiện chỉ một công việc, ví dụ bên trong mỗi con chuột có một bộ vi điều khiển để thực thi công việc tìm vị trí chuột và gửi nó đến PC Bảng 1.1 liệt kê một số sản phẩm nhúng.

4.1.3 Các ứng dụng nhúng của PC × 86.

Mặc dù các bộ vi điều khiển là sự lựa chọn ưa chuộng đối với nhiều hệ thống nhúng nhưng có nhiều khi một bộ vi điều khiển không đủ cho công việc. Vì lý do đó mà những năm gần đây nhiều nhà sản xuất các bộ vi sử lý công dụng chung chẳng hạn như Intel, Motorla, AMD (Advanced Micro Devices, Inc...). Và Cyric (mà bây giờ là một bộ phận của National Senicon ductir, Inc) đã hướng tới bộ vi xử lý cho hiệu suất cao của thị trường nhúng. Trong khi Intel, AMD và Cyrix đẩy các bộ xử lý × 86 của họ vào cho cả thị trường nhúng và thị trường máy tính PC để bán thì Motorola vẫn kiên định giữ họ vi xử lý 68000 lại chủ yếu hướng nó cho các hệ thống nhúng hiệu suất cao và bây giờ Apple không còn dùng 680 × trong các máy tính Macintosh nữa. Trong những năm đầu thập kỷ 90 của thế kỷ 20 máy tính Apple bắt đầu sử dụng các bộ vi xử lý Power PC (như 603, 604, 620 v.v...) thay cho 680 ×0 đối với Macintosh. Bộvi xử lý Power PC là kết quả liên doanh đầu tư của IBM và Motorola và nó được hướng cho thị trướng nhúng hiệu suất cao cũng như cho cả thị trường máy tính PC. Cần phải lưu ý rằng khi một công ty hướng một bộ vi xử lý công dụng chung cho thị trường nhúng nó tối ưu hoá bộ xử lý được sử dụng cho các hệ thống nhúng. Vì lý do đó mà các bộ vi xử lý này thường được gọi là các bộ xử lý nhúng hiệu suất cao. Do vậy các khái niệm các bộ vi điều khiển và bộ xử lý nhúng thường được sử dụng thay đổi nhau.

Một trong những nhu cầu khắt khe nhất của hệ thống nhúng là giảm công suất tiêu thụ và không gian.

Điều này có thể đạt được bằng cách tích hợp nhiều chức năng vào trong chíp CPU. Tất cả mọi bộ xử lý nhúng dựa trên × 86 và 680 × 0 đều có công suất tiêu thu thấp ngoài ra được bổ xung một số dạng cổng vào - ra, cổng COM và bộ nhớ ROM trên một chíp.

Trong các bộ xử lý nhúng hiệu suất cao có xu hướng tích hợp nhiều và nhiều chức năng hơn nữa trên chíp CPU và cho phép người thiết kế quyết định những đặc tính nào họ muốn sử dụng. Xu hướng này cũng đang chiếm lĩnh thiết kế hệ thống PC. Bình thường khi thiết kế bo mạch chủ của PC (Motherboard) ta cần một CPU cộng một chíp - set có chứa các cổng vào - ra, một bộ điều khiển cache, một bộ nhớ Flash ROM có chứa BIOS và cuối cùng là bộ nhớ cache thứ cấp. Những thiết kế mới đang khẩn trương đi vào công nghiệp sản xuất hàng loạt. Ví dụ Cyrix đã tuyên bố rằng họ đang làm việc trên một chíp có chứa toàn bộ một máy tính PC ngoại trừ DRAM. Hay nói cách khác là chúng ta sắp nhìn thấy một máy tính PC trên một chíp.

Hiện nay do chuẩn hoá MS - DOS và Windows nên các hệ thống nhúng đang sử dụng các máy tình PC × 86 . Trong nhiều trường hợp việc sử dụng các máy tính PC × 86 cho các ứng dụng nhúng hiệu suất cao là không tiết kiệm tiền bạc, nhưng nó làm rút ngắn thời gian phát triển vì có một thư viện phần mềm bao la đã được viết cho nền DOS và Windows. Thực tế là Windows là một nền được sử dụng rộng rãi và dễ hiểu có nghĩa là việc phát triển một sản phẩm nhúng dựa trên Windows làm giảm giá thành và rút ngắn thời gian phát triển đáng kể.

1.1.4 Lựa chọn một bộ vi điều khiển.

Có 4 bộ vi điều khiển 8 bit chính. Đó là 6811 của Motorola, 8051 của Intel z8 của Xilog và Pic 16 × của Microchip Technology. Mỗi một kiểu loại trên đây đều có một tập lệnh và thanh ghi riêng duy nhất, nếu chúng đều không tương thích lẫn nhau. Cũng có những bộ vi điều khiển 16 bit và 32 bit được sản xuất bởi các hãng sản xuất chíp khác nhau. Với tất cả những bộ vi điều khiển khác nhau như thế này thì lấy gì làm tiêu chuẩn lựa chọn mà các nhà thiết kế phải cân nhắc? Có ba tiêu chuẩn để lựa chọn các bộ vi điều khiển là:

1) Đáp ứng nhu cầu tính toán của bài toán một cách hiệu quả về mặt giá thành và đầy đủ chức năng có thể nhìn thấy được (khả dĩ).

2) Có sẵn các công cụ phát triển phần mềm chẳng hạn như các trình biên dịch, trình hợp ngữ và gỡ rối.

3) Nguồn các bộ vi điều khiển có sẵn nhiều và tin cậy.

1.1.5 Các tiêu chuẩn lựa chọn một bộ vi điều khiển.

1. Tiêu chuẩn đầu tiênvà trước hết tronglựa chọn một bộ vi điều khiển là nó phải đáp ứng nhu cầu bài toán về một mặt công suất tính toán và giá thành hiệu quả. Trong khi phân tích các nhu cầu của một dự án dựa trên bộ vi điều khiển chúng ta trước hết phải biết là bộ vi điều khiển nào 8 bit, 16 bit hay 32 bit có thể đáp ứng tốt nhất nhu cầu tính toán của bài toán mộ tcách hiệu quả nhất? Những tiêu chuẩn được đưa ra để cân nhắc là:
a) Tốc độ: Tốc độ lớn nhất mà bộ vi điều khiển hỗ trợ là bao nhiêu.
b) Kiểu đóng vỏ: Đó là kiểu 40 chân DIP hay QFP hay là kiểu đóng vỏ khác (DIP -đóng vỏ theo 2 hàng chân. QFP là đóng vỏ vuông dẹt)? Đây là điều quan trọng đối với yêu cầu về không gian, kiểu lắp ráp và tạo mẫu thử cho sản phẩm cuối cùng.
c) Công suất tiêu thụ: Điều này đặc biệt khắt khe đối với những sản phẩm dùng pin, ắc quy.
d) Dung lượng bộ nhớ RAM và ROM trên chíp.
e) Số chân vào - ra và bộ định thời trên chíp
f) Khả năng dễ dàng nâng cấp cho hiệu suất cao hoặc giảm công suất tiêu thụ.
g) Giá thành cho một đơn vị: Điều này quan trọng quyết định giá thành cuối cùng của sản phẩm mà một bộ vi điều khiển được sử dụng. Ví dụ có các bộ vi điều khiển giá 50 cent trên đơn vị khi được mua 100.000 bộ một lúc.
2) Tiêu chuẩn thứ hai trong lựa chọn một bộ vi điều khiển là khả năng phát triển các sản phẩm xung quanh nó dễ dàng như thế nào? Các cân nhắc chủ yếu bao gồm khả năng có sẵn trình lượng ngữ, gỡ rối, trình biên dịch ngôn ngữ C hiệu quả về mã nguồn, trình mô phỏng hỗ trợ kỹ thuật và khả năng sử dụng trong nhà và ngoài môi trường. Trong nhiều trường hợp sự hỗ trợ nhà cung cấp thứ ba (nghĩa là nhà cung cấp khác không phải là hãng sản xuất chíp) cho chíp cũng tốt như, nếu không được tốt hơn, sự hỗ trợ từ nhà sản xuất chíp.
3) Tiêu chuẩn thứ ba trong lựa chọn một bộ vi điều khiển là khả năng sẵn sàng đáp ứng về số lượng trong hiện tại và tương lai. Đối với một số nhà thiết kế điều này thậm chí còn quan trong hơn cả hai tiêu chuẩn đầu tiên. Hiện nay, các bộ vi điều khiển 8 bit dấu đầu, họ 8051 là có số lương lớn nhất các nhà cung cấp đa dạng (nhiều nguồn). Nhà cung cấp có nghĩa là nhà sản xuất bên cạnh nhà sáng chế của bộ vi điều khiển. Trong trường hợp 8051 thì nhà sáng chế

của nó là Intel, nhưng hiện nay có rất nhiều hãng sản xuất nó (cũng như trước kia đã sản xuất).

Các hãng này bao gồm: Intel, Atmel, Philips/signe-tics, AMD, Siemens, Matra và Dallas, Semicndictior.

Bảng 1.2: Địa chỉ của một số hãng sản xuất các thành viên của họ 8051.

Hãng	**Địa chỉ Website**
Intel Antel Plips/ Signetis Siemens Dallas Semiconductor	www.intel.com/design/mcs51 www.atmel.com www.semiconductors.philips.com www.sci.siemens.com www.dalsemi.com

Cũng nên lưu ý rằng Motorola, Zilog và Mierochip Technology đã dành một lượng tài nguyên lớn để đảm bảo khả năng sẵn sàng về một thời gian và phạm vi rộng cho các sản phẩm của họ từ khi các sản phẩm của họ đi vào sản xuất ổn định, hoàn thiện và trở thành nguồn chính. Trong những năm gần đây họ cũng đã bắt đầu bán tế bào thư viện Asic của bộ vi điều khiển.

1.2 Tổng quan về họ 8051.

Tr ong mục này chúng ta xem xét một số thành viên khác nhau của họ bộ vi điều khiển 8051 và các đặc điểm bên trong của chúng. Đồng thời ta điểm qua một số nhà sản xuất khác nhau và các sản phẩm của họ có trên thị trường.

1.2.1 Tóm tắt về lịch sử của 8051.

Vào năm 1981. Hãng Intel giới thiệu một số bộ vi điều khiển được gọi là 8051. Bộ vi điều khiển này có 128 byte RAM, 4K byte ROM trên chíp, hai bộ định thời, một cổng nối tiếp và 4 cổng (đều rộng 8 bit) vào ra tất cả được đặt trên một chíp. Lúc ấy nó được coi là một "hệ thống trên chíp". 8051 là một bộ xử lý 8 bit có nghĩa là CPU chỉ có thể làm việc với 8 bit dữ liệu tại một thời điểm. Dữ liệu lớn hơn 8 bit được chia ra thành các dữ liệu 8 bit để cho xử lý. 8051 có tất cả 4 cổng vào - ra I/O mỗi cổng rộng 8 bit (xem hình 1.2). Mặc dù 8051 có thể có một ROM trên chíp cực đại là 64 K byte, nhưng các nhà sản xuất lúc đó đã cho xuất xưởng chỉ với 4K byte ROM trên chíp. Điều này sẽ được bàn chi tiết hơn sau này.

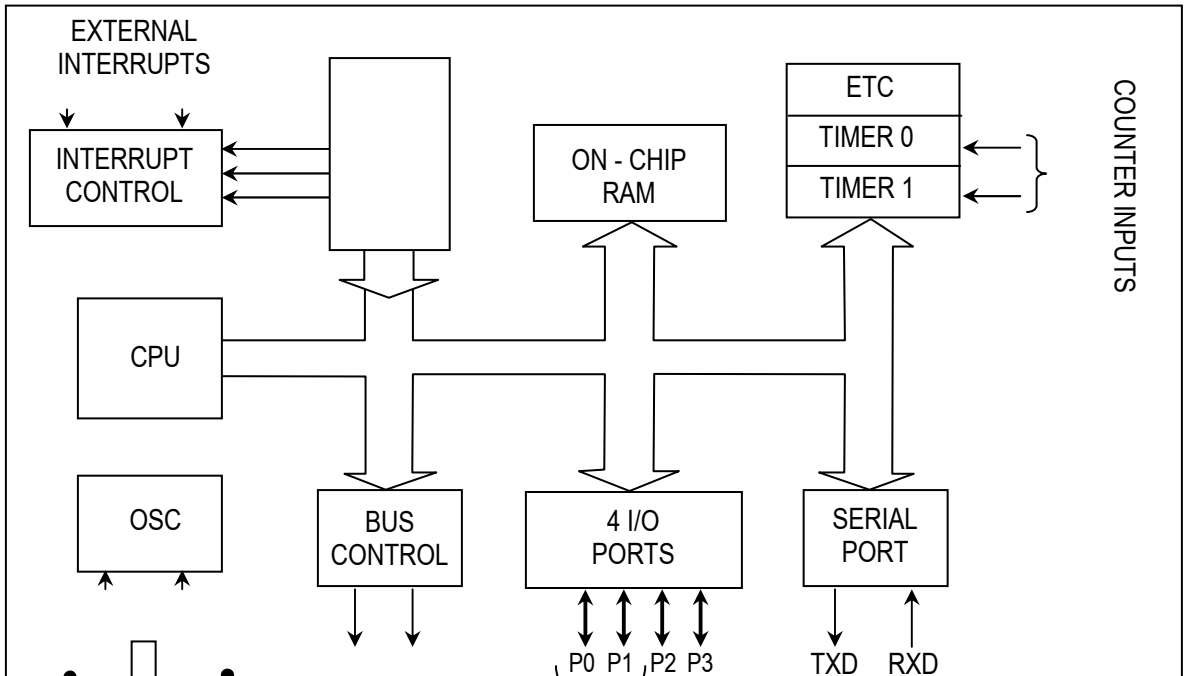
8051 đã trở nên phổ biến sau khi Intel cho phép các nhà sản xuất khác sản xuất và bán bất kỳ dạng biến thế nào của 8051 mà họ thích với điều kiện họ phải để mã lại tương thích với 8051. Điều này dẫn đến sự ra đời nhiều phiên bản của 8051 với các tốc độ khác nhau và dung lượng ROM trên chíp khác nhau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

Bảng 1.3: Các đặc tính của 8051 đầu tiên.

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

1.2.2 Bộ ví điều khiển 8051

Bộ vi điều khiển 8051 là thành viên đầu tiên của họ 8051. Hãng Intel ký hiệu nó như là MCS51. Bảng 3.1 trình bày các đặc tính của 8051.



Hình 1.2: Bố trí bên trong của sơ đồ khối 8051.

1.2.3 các thành viên khác của họ 8051

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

a- Bộ vi điều khiển 8052:

Bộ vi điều khiển 8052 là một thành viên khác của họ 8051, 8052 có tất cả các đặc tính chuẩn của 8051 ngoài ra nó có thêm 128 byte RAM và một bộ định thời nữa. Hay nói cách khác là 8052 có 256 byte RAM và 3 bộ định thời. Nó cũng có 8K byte ROM. Trên chíp thay vì 4K byte như 8051. Xem bảng 1.4.

Bảng1.4: so sánh các đặc tính của các thành viên họ 8051.

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

Như nhìn thấy từ bảng 1.4 thì 8051 là tập con của 8052. Do vậy tất cả mọi chương trình viết cho 8051 đều chạy trên 8052 nhưng điều ngược lại là không đúng.

b- Bộ vi điều khiển 8031:

Một thành viên khác nữa của 8051 là chíp 8031. Chíp này thường được coi như là 8051 không có ROM trên chíp vì nó có OK

byte ROM trên chíp. Để sử dụng chíp này ta phải bổ xung ROM ngoài cho nó. ROM ngoài phải chứa chương trình mà 8031 sẽ nạp và thực hiện. So với 8051 mà chương trình được chứa trong ROM trên chíp bị giới hạn bởi 4K byte, còn ROM ngoài chứa chương trinh được gắn vào 8031 thì có thể lớn đến 64K byte. Khi bổ xung cổng, như vậy chỉ còn lại 2 cổng để thao tác. Để giải quyết vấn đề này ta có thể bổ xung cổng vào - ra cho 8031. Phối phép 8031 với bộ nhớ và cổng vào - ra chẳng hạn với chíp 8255 được trình bày ở chương 14. Ngoài ra còn có các phiên bản khác nhau về tốc độ của 8031 từ các hãng sản xuất khác nhau.

1.2.4. Các bộ vi điều khiển 8051 từ các hãng khác nhau.

Mặc dù 8051 là thành viên phổi biến nhất của họ 8051 nhưng chúng ta sẽ thấy nó trong kho linh kiện. Đó là do 8051 có dưới nhiều dạng kiểu bộ nhớ khác nhau như UV - PROM, Flash và NV - RAM mà chúng đều có số đăng ký linh kiện khác nhau. Việc bàn luận về các kiểu dạng bộ nhớ ROM khác nhau sẽ được trình bày ở chương 14. Phiên bản UV-PROM của 8051 là 8751. Phiên bản Flash ROM được bán bởi nhiều hãng khác nhau chẳng hạn của Atmel corp với tên gọi là AT89C51 còn phiên bản NV-RAM của 8051 do Dalas Semi Conductor cung cấp thì được gọi là DS5000. Ngoài ra còn có phiên bản OTP (khả trình một lần) của 8051 được sản xuất bởi rất nhiều hãng.

a- Bộ vi điều khiển 8751:

Chíp 8751 chỉ có 4K byte bộ nhớ UV-EPROM trên chíp. Để sử dụng chíp này để phát triển yêu cầu truy cập đến một bộ đốt PROM cũng như bộ xoá UV- EPROM để xoá nội dung của bộ nhớ UV-EPROM bên trong 8751 trước khi ta có thể lập trình lại nó. Do một thực tế là ROM trên chíp đối với 8751 là UV-EPROM nên cần phải mất 20 phút để xoá 8751 trước khi nó có thể được lập trình trở lại. Điều này đã dẫn đến nhiều nhà sản xuất giới thiệu các phiên bản Flash Rom và UV-RAM của 8051. Ngoài ra còn có nhiều phiên bản với các tốc độ khác nhau của 8751 từ nhiều hãng khác nhau.

b- Bộ vi điều khiển AT8951 từ Atmel Corporation.

Chíp 8051 phổ biến này có ROM trên chíp ở dạng bộ nhớ Flash. Điều này là lý tưởng đối với những phát triển nhanh vì bộ nhớ Flash có thể được xoá trong vài giây trong tương quan so với 20 phút hoặc hơn mà 8751 yêu cầu. Vì lý do này mà AT89C51 để phát triển một hệ thống dựa trên bộ vi điều khiển yêu cầu một bộ đốt ROM mà

có hỗ trợ bộ nhớ Flash. Tuy nhiên lại không yêu cầu bộ xoá ROM. Lưu ý rằng trong bộ nhớ Flash ta phải xoá toàn bộ nội dung của ROM nhằm để lập trình lại cho nó. Việc xoá bộ nhớ Flash được thực hiện bởi chính bộ đốt PROM và đây chính là lý do tại sao lại không cần đến bộ xoá. Để loại trừ nhu cầu đối với một bộ đốt PROM hãng Atmel đang nghiên cứu một phiên bản của AT 89C51 có thể được lập trình qua cổng truyền thông COM của máy tính IBM PC .

Bảng 1.5: Các phiên bản của 8051 từ Atmel (Flash ROM).

Số linh kiện	ROM	RAM	Chân I/O	Timer	Ngắt	Vcc	Đóng v
AT89C51	4K	128	32	2	6	5V	40
AT89LV51	4K	128	32	2	6	3V	40
AT89C1051	1K	64	15	1	3	3V	20
AT89C2051	2K	128	15	2	6	3V	20
AT89C52	8K	128	32	3	8	5V	40
AT89LV52	8K	128	32	3	8	3V	40

Chữ C trong ký hiệu AT89C51 là CMOS.

Cũng có những phiên bản đóng vỏ và tốc độ khác nhau của những sản phẩm trên đây. Xem bảng 1.6. Ví dụ để ý rằng chữ "C" đứng trước số 51 trong AT 89C51 -12PC là ký hiệu cho CMOS "12" ký hiệu cho 12 MHZ và "P" là kiểu đóng vỏ DIP và chữ "C" cuối cùng là ký hiệu cho thương mại (ngược với chữ "M" là quân sự). Thông thường AT89C51 - 12PC rát lý tưởng cho các dự án của học sinh, sinh viên.

Bảng 1.6: Các phiên bản 8051 với tốc độ khác nhau của Atmel.

Mã linh kiện	Tốc độ	Số chân	Đóng vỏ	Mục đích
AT89C51-12PC	42MHZ	40	DTP	Thương mại

c- Bộ vi điều khiển DS5000 từ hãng Dallas Semiconductor.

Một phiên bản phổ biến khác nữa của 8051 là DS5000 của hãng Dallas Semiconductor. Bộ nhớ ROM trên chíp của DS5000 ở dưới dạng NV-RAM. Khả năng đọc/ ghi của nó cho phép chương trình được nạp vào ROM trên chíp trong khi nó vẫn ở trong hệ thống (không cần phải lấy ra). Điều này còn có thể được thực hiện thông qua cổng nối tiếp của máy tính IBM PC. Việc nạp chương trình trong hệ thống (in-system) của DS5000 thông qua cổng nối tiếp của PC làm cho nó trở thành một hệ thống phát triển tại chỗ lý tưởng. Một ưu việt của NV-RAM là khả năng thay đổi nội dung của ROM theo từng byte tại một thời điểm. Điều này tương phản với bộ nhớ Flash và EPROM mà bộ nhớ của chúng phải được xoá sạch trước khi lập trình lại cho chúng.

Bảng 1.7: Các phiên bản 8051 từ hãng Dallas Semiconductor.

Mã linh kiện	ROM	RAM	Chân I/O	Timer	Ngắt	Vcc	Đóng vỏ
DS5000-8	8K	128	32	2	6	5V	40
DS5000-32	32K	128	32	2	6	5V	40
DS5000T-8	8K	128	32	2	6	5V	40
DS5000T-8	32K	128	32	2	6	5V	40

Chữ "T" đứng sau 5000 là có đồng hồ thời gian thực.

Lưu ý rằng đồng hồ thời gian thực RTC là khác với bộ định thời Timer. RTC tạo và giữ thời gian l phút giờ, ngày, tháng - năm kể cả khi tắt nguồn.

Còn có nhiều phiên bản DS5000 với những tốc độ và kiểu đóng gói khác nhau.(Xem bảng 1.8). Ví dụ DS5000-8-8 có 8K NV-RAM và tốc đọ 8MHZ. Thông thường DS5000-8-12 hoặc DS5000T-8-12 là lý tưởng đối với các dự án của sinh viên.

Bảng 1.8:Các phiên bản của DS5000 với các tốc độ khác nhau

Mã linh kiện	NV- RAM	Tốc độ
DS5000-8-8	8K	8MHz
DS5000-8-12	8K	12MHz
DS5000-32-8	32K	8MHz

DS5000T-32-12 DS5000-32-12 DS5000-8-12	32K 32K 8K	8MHz (with RTC) 12MHz 12MHz (with RTC)

d- Phiên bản OTP của 8051.

Các phiên bản OTP của 8051 là các chíp 8051 có thể lập trình được một lần và được cung cấp từ nhiều hãng sản xuất khác nhau. Các phiên bản Flash và NV-RAM thường được dùng để phát triển sản phẩm mẫu. Khi một sản pohẩm được thiết kế và được hoàn thiện tuyệt đối thì phiên bản OTP của 8051 được dùng để sản hàng loạt vì nó sẽ hơn rất nhiều theo giá thành một đơn vị sản phẩm

e- Họ 8051 từ Hãng Philips

Một nhà sản xuất chính của họ 8051 khác nữa là Philips Corporation. Thật vậy, hãng này có một dải lựa chọn rộng lớn cho các bộ vi điều khiển họ 8051. Nhiều sản phẩm của hãng đã có kèm theo các đặc tính như các bộ chuyển đổi ADC, DAC, cổng I/0 mở rộng và cả các phiên bản OTP và Flash.

CHƯƠNG 2
Lập trình hợp ngữ 8051

2.1 Bên trong 8051.

Trong phần này chúng ta nghiên cứu các thanh ghi chính của 8051 và trình bày cách sử dụng với các lệnh đơn giản MOV và ADD.

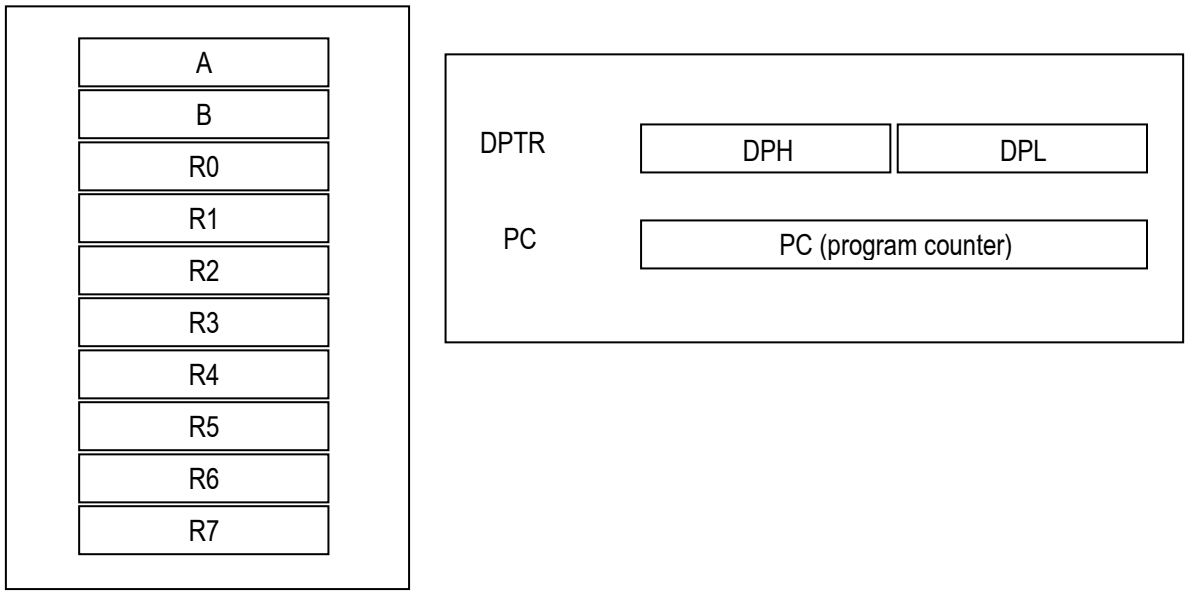
2.1.1 Các thanh ghi.

Trong CPU các thanh ghi được dùng để lưu cất thông tin tạm thời, những thông tin này có thể là một byte dữ liệu cần được sử lý hoặc là một địa chỉ đến dữ liệu cần được nạp. Phần lớn các thanh ghi của 8051 là các thanh ghi 8 bit. Trong 8051 chỉ có một kiểu dữ liệu: Loại 8 bit, 8 bit của một thanh ghi được trình bày như sau:

D7	D6	D5	D4	D3	D2	D1	D0

với MSB là bit có giá trị cao nhất D7 cho đến LSB là bit có giá trị thấp nhất D0. (MSB - Most Sigfican bit và LSB - Leart Significant Bit). Với một kiểu dữ liệu 8 bit thì bất kỳ dữ liệu nào lớn hơn 8 bit đều phải được chia thành các khúc 8 bit trước khi được xử lý. Vì có một số lượng lớn các thanh ghi trong 8051 ta sẽ tập trung vào một số thanh ghi công dụng chung đặc biệt trong các chương kế tiếp. Hãy tham khảo phụ lục Appendix A.3 để biết đầy đủ về các thanh ghi của 8051.

Hình 2.1: a) Một số thanh ghi 8 bit của 8051
b) Một số thanh ghi 16 bit của 8051



Các thanh ghi được sử dụng rộng rãi nhất của 8051 là A (thanh ghi tích luỹ), B, R0 - R7, DPTR (con trỏ dữ liệu) và PC (bộ đếm chương trình). Tất cả các dữ liệu trên đều là thanh g hi 8 bit trừ DPTR và PC là 16 bit. Thanh ghi tích luỹ A được sử dụng cho tất cả mọi phép toán số học và lô-gíc. Để hiểu sử dụng các thanh ghi này ta sẽ giới thiệu chúng trong các ví dụ với các lệnh đơn giản là ADD và MOV.

2.1.2 Lệnh chuyển MOV.

Nói một cách đơn giản, lệnh MOV sao chép dữ liệu từ một vị trí này đến một ví trí khác. Nó có cú pháp như sau:

```
MOV             ; Đích, nguồn; sao chép nguồn vào đích
```

Lệnh này nói CPU chuyển (trong thực tế là sao chép) toán hạng nguồn vào toán hạng đích. Ví dụ lệnh "MOV A, R0" sao chép nội dung thanh ghi R0 vào thanh ghi A. Sau khi lênh này được thực hiện thì thanh ghi A sẽ có giá trị giống như thanh ghi R0. Lệnh MOV không tác động toán hạng nguồn. Đoạn chương trình dưới đây đầu tiên là nạp thanh ghi A tới giá trị 55H 9là giá trị 55 ở dạng số Hex) và sau đó chuyển giá trị này qua các thanh ghi khác nhau bên trong CPU. Lưu ý rằng dấu "#" trong lệnh báo rằng đó là một giá trị. Tầm quan trọng của nó sẽ được trình bày ngay sau ví dụ này.

```
MOV   A, #55H;      ; Nạp trí trị 55H vào thanh ghi A (A = 55H)
MOV   R0, A         ; Sao chép nội dung A vào R0 (bây giờ R0=A)
MOV   R1, A         ; Sao chép nội dung A và R1 (bây giờ R1=R0=A)
MOV   R2, A         ; Sao chép nội dung A và R2 (bây giờ R2=R1=R0=A)
MOV   R3, #95H      ; Nạp giá trị 95H vào thanh ghi R3 (R3 = 95H)
MOV   A, R3         ; Sáo chép nội dung R3 vào A (bây giờ A = 95H)
```

Khi lập trình bộ vi điều khiển 8051 cần lưu ý các điểm sau:

1. Các giá trị có thể được nạp vào trực tiếp bất kỳ thanh ghi nào A, B, R0 - R7. Tuy nhiên, để thông báo đó là giá trị tức thời thì phải đặt trước nó một ký hiệu "#" như chỉ ra dưới đây.

```
MOV   A, #23H       ; Nạp giá trị 23H vào A (A = 23H)
MOV   R0, #12H      ; Nạp giá trị 12H vào R0 (R0 = 2BH)
MOV   R1, #1FH      ; Nạp giá trị 1FH vào R1 (R1 = 1FH)
MOV   R2, #2BH      ; Nạp giá trị 2BH vào R2 (R2 = 2BH)
MOV   B, # 3CH      ; Nạp giá trị 3CH vào B (B = 3CH)
MOV   R7, #9DH      ; Nạp giá trị 9DH vào R7 (R7 = 9DH)
MOV   R5, #0F9H     ; Nạp giá trị F9H vào R5 (R5 = F9H)
MOV   R6, #12       ;Nạp giá trị thập phân 12 = 0CH vào R6
                    (trong R6 có giá trị 0CH).
```

Để ý trong lệnh "MOV R5, #0F9H" thì phải có số 0 đứng trước F và sau dấu # báo rằng F là một số Hex chứ không phải là một ký tự. Hay nói cách khác "MOV R5, #F9H" sẽ gây ra lỗi.

2. Nếu các giá trị 0 đến F được chuyển vào một thanh ghi 8 bit thì các bit còn lại được coi là tất cả các số 0. Ví dụ, trong lệnh "MOV A,#5" kết quả là A=0.5, đó là A = 0000 0101 ở dạng nhị phân.
3. Việc chuyển một giá trị lớn hơn khả năng chứa của thanh ghi sẽ gây ra lỗi ví dụ:

```
MOV   A, #7F2H      ; Không hợp lệ vì 7F2H > FFH
MOV   R2, 456       ; Không hợp lệ vì 456 > 255 (FFH)
```

4. Để nạp một giá trị vào một thanh ghi thì phải gán dấu "#" trước giá trị đó. Nếu không có dấu thì nó hiểu rằng nạp từ một vị trí nhớ. Ví dụ "MOV A, 17H" có nghĩa là nạp giá trị trong ngăn nhớ có giá trị 17H vào thanh ghi A và tại địa chỉ đó dữ liệu có thể có bất kỳ giá trị nào từ 0 đến FFH. Còn để nạp giá trị là 17H vào thanh ghi A thì cần phải có dấu "#" trước 17H như thế này. "MOV A, #17H". Cần lưu ý rằng nếu thiếu dấu "#" trước một thì sẽ không gây lỗi vì hợp ngữ cho đó là một lệnh hợp

lệ. Tuy nhiên, kết quả sẽ không đúng như ý muốn của người lập trình. Đ ây sẽ là một lỗi thường hay gặp đối với lập trình viên mới.

2.1.3 Lệnh cộng ADD.

Lệnh cộng ADD có các phép như sau:

```
ADD a, nguồn          ; Cộng toán hạng nguồn vào thanh ghi A.
```

Lệnh cộng ADD nói CPU cộng byte nguồn vào thanh ghi A và đặt kết quả thanh ghi A. Để cộng hai số như 25H và 34H thì mỗi số có thể chuyển đến một thanh ghi và sau đó cộng lại với nhau như:

```
MOV   A, #25H       ; Nạp giá trị 25H vào A
MOV   R2, #34H      ; Nạp giá trị 34H vào R2
ADD   A, R2         ; Cộng R2 vào A và kết quả A = A + R2
```

Thực hiện chương trình trên ta được A = 59H (vì 25H + 34H = 59H) và R2 = 34H, chú ý là nội dụng R2 không thay đổi. Chương trình trên có thể viết theo nhiều cách phụ thuộc vào thanh ghi được sử dụng. Một trong cách viết khác có thể là:

```
MOV   R5, #25H      ; Nạp giá trị 25H vào thanh ghi R5
MOV   R7, #34H      ; Nạp giá trị 34H vào thanh ghi R7
MOV   A, #0         ; Xoá thanh ghi A (A = 0)
ADD   A, R5         ; Cộng nội dung R5 vào A (A = A + R5)
ADD   A, R7         ; Cộng nội dung R7 vào A (A = A + R7 = 25H + 34H)
```

Chương trình trên có kết quả trong A Là 59H, có rất nhiều cách để viết chương trình giống như vậy. Một câu hỏi có thể đặt ra sau khi xem đoạn chương trình trên là liệu có cần chuyển cả hai dữ liệu vào các thanh ghi trước khi cộng chúng với nhau không? Câu trả lời là không cần. Hãy xem đoạn chương trình dưới đây:

```
MOV   A, #25H       ; Nạp giá trị thứ nhất vào thanh ghi A (A = 25H)
ADD   A, #34H       ; Cộng giá trị thứ hai là 34H vào A (A = 59H)
```

Trong trường hợp trên đây, khi thanh ghi A đã chứa số thứ nhất thì giá trị thứ hai đi theo một toán hạng. Đây được gọi là toán hạng tức thời (trực tiếp).

Các ví dụ trước cho đến giờ thì lệnh ADD báo rằng toán hạng nguồn có thể hoặc là một thanh ghi hoặc là một dữ liệu trực tiếp (tức thời) nhưng thanh ghi đích luôn là thanh ghi A, thanh ghi tích luỹ. Hay nói cách khác là một lệnh như "ADD R2, #12H" là lệnh không hợp lệ vì mọi phép toán số học phải cần đến thanh ghi A và lệnh "ADD R4, A" cũng không hợp lệ vì A luôn là thanh ghi đích cho mọi phép số học. Nói một cách đơn giản là trong 8051 thì mọi phép toán số học đều cần đến thanh A với vai trò là toán hạng đích. Phần trình bày trên đây giải thích lý do vì sao thanh ghi A như là thanh thi tích luỹ. Cú pháp các lệnh hợp ngữ mô tả cách sử dụng chúng và liệt kê các kiểu toán hạng hợp lệ được cho trong phụ lục Appendix A.1.

Có hai thanh ghi 16 bit trong 8051 là bộ đếm chương trình PC và con trỏ dữ liệu APTR. Tầm quan trọng và cách sử dụng chúng được trình bày ở mục 2.3. Thanh ghi DPTR được sử dụng để truy cập dữ liệu và được làm kỹ ở chương 5 khi nói về các chế độ đánh địa chỉ.

2.2 Giới thiệu về lập trình hợp ngữ 8051.

Trong phần này chúng ta bàn về dạng thức của hợp ngữ và định nghĩa một số thuật ngữ sử dụng rộng rãi gắn liền với lập trình hợp ngữ.

CPU chỉ có thể làm việc với các số nhị phân và có thể chạy với tốc độ rất cao. Tuy nhiên, thật là ngán ngậm và chậm chạp đối với con người phải làm việc với các số 0 và 1 để lập trình cho máy tính. Một chương trình chứa các số 0 và 1 được gọi là ngôn ngữ máy.

Trong những ngày đầu của máy tính, các lập trình viên phải viết mã chương trình dưới dạng ngôn ngữ máy. Mặc dụ hệ thống thập lục phân (số Hex) đã được sử dụng như một cách hiệu quả hơn để biểu diễn các số nhị phân thì quá trình làm việc với mã máy vẫn còn là công việc cồng kềnh đối với con người. Cuối cùng, các nguồn ngữ hợp ngữ đã được phát, đã cung cấp các từ gợi nhớ cho các lệnh mã máy cộng với những đặc tính khác giúp cho việc lập trình nhanh hơn và ít mắc lỗi hơn. Thuật ngữ từ gợi nhớ (mnemonic) thường xuyên sử dụng trong tài liệu khoa học và kỹ thuật máy tính để tham chiếu cho các mã và từ rút gọn tương đối dễ nhớ, các chương trình hợp ngữ phải được dịch ra thanh mã máy bằng một chương trình được là trình hợp ngữ (hợp dịch). Hợp ngữ được coi như là một ngông ngữ bậc thấp vì nó giao tiếp trực tiếp với cấu trúc bên trong của CPU. Để lập trình trong hợp ngữ, lập trình viên phải biết tất cả các thanh ghi của CPU và kích thước của chúng cũng như các chi tiết khác.

Ngày nay, ta có thể sử dụng nhiều ngôn ngữ lập trình khác nhau, chẳng hạn như Basic, Pascal, C, C^{++}, Java và vô số ngôn ngữ khác. Các ngôn ngữ này được coi là nhưng ngôn ngữ bậc cao vì lập trình viên không cần phải tương tác với các chi tiết bên trong của CPU. Một trình hợp dịch được dùng để dịch chương trình hợp ngữ ra mã máy còn (còn đôi khi cũng còn được gọi mà đối tượng (Object Code) hay mã lệnh Opcode), còn các ngôn ngữ bậc cao được dịch thành các ngôn ngữ mã máy bằng một chương trình gọi là trình biên dịch. Ví dụ, để viết một chương trình trong C ta phải sử dụng một trình biên dịch C để dịch chương trình về dạng mã máy. Bây giờ ta xét dạng thức hợp ngữ của 8051 và sử dụng trình hợp dịch để tạo ra một chương trình sẵn sàng chạy ngay được.

2.2.1 Cấu trúc của hợp ngữ.

Một chương trình hợp ngữ bao gồm một chuỗi các dòng lệnh hợp ngữ. Một lệnh hợp ngữ có chứa một từ gợi nhớ (mnemonic) và tuy theo từng lệnh và sau nó có một hoặc hai toán hạng. Các toán hạng là các dữ liệu cần được thao tác và các từ gợi nhớ là các lệnh đối với CPU nói nó làm gì với các dữ liệu.

```
        ORG   0H          ; Bắt đầu (origin) tại ngăn nhớ 0
        MOV   R5, #25H    ; Nạp 25H vào R5
        MOV   R7, #34H    ; Nạp 34H vào R7
        MOV   A, #0       ; Nạp 0 vào thanh ghi A
        ADD   A, R5       ; Cộng nôi dụng R5 vào A (A = A + R5)
        ADD   A, R7       ; Cộng nội dung R7 vào A (A = A + R7)
        ADD   A, #121H    ; Cộng giá trị 12H vào A (A = A + 12H)
HERE: SJMP HERE           ; ở lại trong vòng lặp này
        END               ; Kết thúc tệp nguồn hợp ngữ
```

Chương trình 2.1: Ví dụ mẫu về một chương trình hợp ngữ.

Chương trình 2.1 cho trên đây là một chuỗi các câu lệnh hoặc các dòng lệnh được viết hoặc bằng các lệnh hợp ngữ như ADD và MOV hoặc bằng các câu lệnh được gọi là các chỉ dẫn. Trong khi các lệnh hợp ngữ thì nói CPU phải làm gì thì các chỉ lệnh

(hay còn gọi là giả lệnh) thì đưa ra các chỉ lệnh cho hợp ngữ. Ví dụ, trong chương trình 2.1 thì các lệnh ADD và MOV là các lệnh đến CPU, còn ORG và END là các chỉ lệnh đối với hợp ngữ. ORG nói hợp ngữ đặt mã lệnh tại ngăn nhớ 0 và END thì báo cho hợp ngữ biết kết thúc mã nguồn. Hay nói cách khác một chỉ lệnh để bắt đầu và chỉ lệnh thứ hai để kết thúc chương trình.

Cấu trúc của một lệnh hợp ngữ có 4 trường như sau:

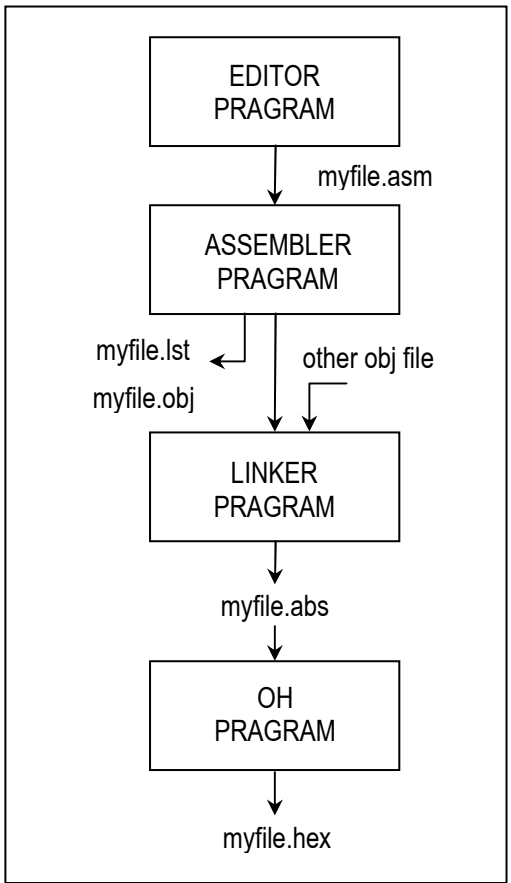
[nhãn:] [từ gợi nhớ] [các toán hạng] [; chú giải]

Các trường trong dấu ngoặc vuông là tuỳ chọn và không phải dòng lệnh nào cũng có chúng. Các dấu ngoặc vuông không được viết vào. Với dạng thức trên đây cần lưu ý các điểm sau:

1. Trường nhãn cho phép chương trình tham chiếu đến một dòng lệnh bằng tên. Nó không được viết quá một số ký tự nhất định. Hãy kiểm tra quy định này của hợp ngữ mà ta sử dụng.
2. Từ gợi nhớ (lệnh) và các toán hạng là các trường kết hợp với nhau thực thi công việc thực tế của chương trình và hoàn thiện các nhiệm vụ mà chương trình được viết cho chúng. Trong hợp ngữ các câu lệnh như:

   " ADD A, B"
   "MOV A, #67H"

   thì ADD và MOV là những từ gợi nhớ tạo ra mã lệnh, còn "A, B" và "A, #67H" là những toán hạng thì hai trường có thể chứa các lệnh giả hoặc chỉ lệnh của hợp ngữ. Hãy nhớ rằng các chỉ lệnh không tạo ra mã lệnh nào (mã máy) và chúng chỉ dùng bởi hợp ngữ, ngược lại đối với các lệnh là chúng được dịch ra mã máy (mã lênh) cho CPU thực hiện. Trong chương trình 2.1 các lệnh ORG và END là các chỉ lệnh (một số hợp ngữ của 8051 sử dụng dạng .ORG và .END). Hãy đọc quy định cụ thể của hợp ngữ ta sử dụng.
3. Chương chú giải luôn phải bắt đầu bằng dấu chấm phẩy (;). Các chú giải có thể bắt đầu ở đầu dòng hoặc giữa dòng. Hợp ngữ bỏ qua (làm ngơ) các chú giải nhưng chúng lại rất cần thiết đối với lập trình viên. Mặc dù các chú giải là tuỳ chọn, không bắt buộc nhưng ta nên dùng chúng để mô tả chương trình để giúp cho người khác đọc và hiểu chương trình dễ dàng hơn.
4. Lưu ý đến nhãn HERE trong trường nhãn của chương trình 2.1. Một nhãn bất kỳ tham chiếu đến một lệnh phải có dấu hai chấm (:) đứng ở sau. Trong câu lệnh nhảy ngắn SJMP thì 8051 được ra lệnh ở lại trong vòng lặp này vô hạn. Nếu hệ thống của chúng ta có một chương trình giám sát thì takhông cần dòng lệnh này và nó có thể được xoá đi ra khỏi chương trình.



2.3 Hợp dịch và chạy một chương trình 8051.

Như vậy cấu trúc của một chương trình hợp ngữ ta đã được biết, câu hỏi đặt ra là chương

trình sẽ được tạo ra và hợp dịch như thế nào và làm thế nào để có thể chạy được? Các bước để tạo ra một chương trình hợp ngữ có thể chạy được là:

1. Trước hết ta sử dụng mộ trình soạn thảo để gỡ vào một chương trình giống như chương trình 2.1. Có nhiều trình soạn thảo tuyệt vời hoặc các bộ sử lý từ được sử dụng để tạo ra và/ hoặc để soạn thảo chương trình. Một trình soạn thảo được sử dụng rộng rãi là trình soạn thảo EDIT của MS-DOS (hoặc Noterad của Windows) đều chạy trên hệ điều hành Microsoft. Lưu ý rằng, trình soạn thảo phải có khả năng tạo ra tệp mã ASCII. Đối với nhiều trình hợp ngữ thì các tên tệp tuân theo các quy ước thường lệ củ DOS, nhưng phần mở rộng của các tệp nguồn phải là "asm" hay "src" tuỳ theo trình hợp ngữ mà ta sử dụng.
2. Tệp nguồn có phần mở rộng "asm" chứa mã chương trình được tạo ra ở bước 1 được nạp vào trình hợp dịch của 8051. Trình hợp dịch chuyển các lệnh ra mã máy. Trình hợp dịch sẽ tạo ra một tệp đối tượng và một tệp liệt kê với các thành phần mở rộng "obj" và "lst" tương ứng.
3. Các trình hợp dịch yêu cầu một bước thứ ba gọi là liên kết. Chương trình liên kết lấy một hoặc nhiều tệp đối tượng và tạo ra một tệp đối tượng tuyệt đối với thành phần mở rộng "abs". Tệp "abs" này được sử dụng bởi thùng chứa của 8051 có một chương trình giám sát.
4. Kế sau đó tệp "abs" được nạp vào một chương trình được gọi là "0H" (chuyển đối tượng object về dạng số Hex) để tạo ra một tệp với đuôi mở rộng "Hex" có thể nạp tốt vào trong ROM. Chương trình này có trong tất cả mọi trình hợp ngữ của 8051 các trình hợp ngữ dựa trên Windows hiện nay kết hợp các bước 2 đến 4 vào thành một bước.

Hình 2.2: Các bước để tạo ra một chương trình.

2.3.1 Nói thêm về các tệp ".asm" và ".object".

Tệp ".asm" cũng được gọi là tệp nguồn và chính vì lý do này mà một số trình hợp ngữ đòi hỏi tệp này phải có một phần mở rộng "src" từ chữ "source" là nguồn. Hãy kiểm tra hợp ngữ 8051 mà ta sử dụng xem nó có đòi hỏi như vậy không? Như ta nói trước đây tệp này được tạo ra nhờ một trình biên tập chẳng hạn như Edit của DOS hoặc Notepad của Windows. Hợp ngữ của 8051 chuyển đổi các tệp hợp ngữ trong tệp .asm thành ngôn ngữ mã máy và cung cấp tệp đối tượng .object. Ngoài việc tạo ra tệp đối tượng trình hợp ngữ cũng cho ra tệp liệt kê "lst" (List file).

2.3.2 Tệp liệt kê ".lst".

Tệp liệt kê là một tuỳ chọn, nó rất hữu ích cho lập trình viên vì nó liệt kê tất cả mọi mã lệnh và địa chỉ cũng như tất cả các lỗi mà trình hợp ngữ phát hiện ra. Nhiều trình hợp ngữ giả thiết rằng, tệp liệt kê là không cần thiết trừ khi ta báo rằng ta muốn tạo ra nó. Tệp này có thể được truy cập bằng một trình biên dịch như Edit của DOS hoặc Notepad của Window và được hiển thị trên màn hình hoặc được gửi ra máy in. Lập trình viên sử dụng tệp liệt kê để tìm các lỗi cú pháp. Chỉ sau khi đã sửa hết các lỗi được đánh dấu trong tệp liệt kê thì tệp đối tượng mới sẵn sàng làm đầu vào cho chương trình liên kết.

```
1 0000              ORG   0H          ; Bắt đầu ở địa chỉ 0
2 0000  7D25        MOV   R5, #25H    ; Nạp giá trị 25H vào R5
3 0002 7F34         MOV   R7, #34H    ; Nạp giá trị 34H vào R7
4 0004  7400        MOV   A, #0       ; Nạp 0 vào A (xoá A)
5 0006 2D           ADD   A, R5       ; Cộng nội dung R5 vào A (A = A + R5)
6 0007 2F           ADD   A, R7       ; Cộng nội dung R7 vào A (A = A + R7)
7 0008 2412         ADD   A, #12H     ; Cộng giá trị 12H vào A (A = A + 12H)
```

```
8 00A  BCEF  HERE: SJMP HERE          ; ở lại vòng lặp này
9 000C             END                ; Kết thúc tệp .asm
```

Chương trình 2.2: Tệp liệt kê.

2.4 Bộ đếm chương trình và không gian ROM trong 8051.

2.4.1 Bộ đếm chương trình trong 8051.

Một thanh ghi quan trọng khác trong 8051 là bộ đếm chương trình . Bộ đếm chương trình chỉ đếm địa chỉ của lệnh kế tiếp cần được thực hiện. Khi CPU nạp mã lệnh từ bộ nhớ ROM chương trình thì bộ đếm chương trình tăng lên chỉ đếm lệnh kết tiếp. Bộ đếm chương trình trong 8051 có thể truy cập các địa chỉ chương trình trong 8051 rộng 16 bit. Điều này có nghĩa là 8051 có thể truy cập các địa chỉa chương trình từ 0000 đến FFFFH tổng cộng là 64k byte mã lệnh. Tuy nhiên, không phải tất cả mọi thành viên của 8051 đều có tất cả 64k byte ROM trên chíp được cài đặt. Vậy khi 8051 được bật nguồn thì nó đánh thức ở địa chỉa nào?

2.4.2 Địa chỉ bắt đầu khi 8051 được cấp nguồn.

Một câu hỏi mà ta phải hỏi về bộ vi điều khiển bất kỳ là thì nó được cấp nguồn thì nó bắt đầu từ địa chỉ nào? Mỗi bộ vi điều khiển đều khác nhau. Trong trường hợp họ 8051 thì mọi thành viên kể từ nhà sản xuất nào hay phiên bản nào thì bộ vi điều khiển đều bắt đầu từ địa chỉ 0000 khi nó được bật nguồn. Bật nguồn ở đây có nghĩa là ta cấp điện áp V_{cc} đến chân RESET như sẽ trình bày ở chương 4. Hay nói cách khác, khi 8051 được cấp nguồn thì bộ đếm chương trình có giá trị 0000. Điều này có nghĩa là nó chờ mã lệnh đầu tiên được lưu ở địa chỉa ROM 0000H. Vì lý do này mà trong vị trí nhớ 0000H của bộ nhở ROM chương trình vì đây là nơi mà nó tìm lệnh đầu tiên khi bật nguồn. Chúng ta đạt được điều này bằng câu lệnh ORG trong chương trình nguồn như đã trình bày trước đây. Dưới đây là hoạt động từng bước của bộ đếm chương trình trong qúa trình nạp và thực thi một chương trình mẫu.

2.4.3 Đặt mã vào ROM chương trình.

Để hiểu tốt hơn vai trò của bộ đếm chương trình trong quá trình nạp và thực thi một chương trình, ta khảo sát một hoạt động của bộ đếm chương trình khi mỗi lệnh được nạp và thực thi. Trước hết ta khảo sát một lần nữa tệp liệt kê của chương trình mẫu và cách đặt mã vào ROM chương trình 8051 như thế nào? Như ta có thể thấy, mã lệnh và toán hạng đối với mỗi lệnh được liệt kê ở bên trái của lệnh liệt kê.

Chương trình 2.1: Ví dụ mẫu về một chương trình hợp ngữ.

Chương trình 2.1 cho trên đây là một chuỗi các câu lệnh hoặc các dòng lệnh được viết hoặc bằng các lệnh hợp ngữ như ADD và MOV hoặc bằng các câu lệnh được gọi là các chỉ dẫn. Trong khi các lệnh hợp ngữ thì nói CPU phải làm gì thì các chỉ lệnh (hay còn gọi là giả lệnh) thì đưa ra các chỉ lệnh cho hợp ngữ. Ví dụ, trong chương trình 2.1 thì các lệnh ADD và MOV là các lệnh đến CPU, còn ORG và END là các chỉ lệnh đối với hợp ngữ. ORG nói hợp ngữ đặt mã lệnh tại ngăn nhớ 0 và END thì báo cho hợp ngữ biết kết thúc mã nguồn. Hay nói cách khác một chỉ lệnh để bắt đầu và chỉ lệnh thứ hai để kết thúc chương trình.

Cấu trúc của một lệnh hợp ngữ có 4 trường như sau:

[nhãn:] [từ gợi nhớ] [các toán hạng] [; chú giải]

Các trường trong dấu ngoặc vuông là tuỳ chọn và không phải dòng lệnh nào cũng có chúng. Các dấu ngoặc vuông không được viết vào. Với dạng thức trên đây cần lưu ý các điểm sau:

Trường nhãn cho phép chương trình tham chiếu đến một dòng lệnh bằng tên. Nó không được viết quá một số ký tự nhất định. Hãy kiểm tra quy định này của hợp ngữ mà ta sử dụng.

Từ gợi nhớ (lệnh) và các toán hạng là các trường kết hợp với nhau thực thi công việc thực tế của chương trình và hoàn thiện các nhiệm vụ mà chương trình được viết cho chúng. Trong hợp ngữ các câu lệnh như:

" ADD A, B"
"MOV A, #67H"

Thì ADD và MOV là những từ gợi nhớ tạo ra mã lệnh, còn "A, B" và "A, #67H" là những toán hạng thì hai trường có thể chứa các lệnh giả hoặc chỉ lệnh của hợp ngữ. Hãy nhớ rằng các chỉ lệnh không tạo ra mã lệnh nào (mã máy) và chúng chỉ dùng bởi hợp ngữ, ngược lại đối với các lệnh là chúng được dịch ra mã máy (mã lênh) cho CPU thực hiện. Trong chương trình 2.1 các lệnh ORG và END là các chỉ lệnh (một số hợp ngữ của 8051 sử dụng dạng .ORG và .END). Hãy đọc quy định cụ thể của hợp ngữ ta sử dụng.

Trương chú giải luôn phải bắt đầu bằng dấu chấm phẩy (;). Các chú giải có thể bắt đầu ở đầu dòng hoặc giữa dòng. Hợp ngữ bỏ qua (làm ngơ) các chú giải nhưng chúng lại rất cần thiết đối với lập trình viên. Mặc dù các chú giải là tuỳ chọn, không bắt buộc nhưng ta nên dùng chúng để mô tả chương trình để giúp cho người khác đọc và hiểu chương trình dễ dàng hơn.

Lưu ý đến nhãn HERE trong trường nhãn của chương trình 2.1. Một nhãn bất kỳ tham chiếu đến một lệnh phải có dấu hai chấm (:) đứng ở sau. Trong câu lệnh nhảy ngắn SJMP thì 8051 được ra lệnh ở lại trong vòng lặp này vô hạn. Nếu hệ thống của chúng ta có một chương trình giám sát thì takhông cần dòng lệnh này và nó có thể được xoá đi ra khỏi chương trình.

Địa chỉ	Mã lệnh
0000	7D
0001	25
0002	F7
0003	34
0004	74
0005	00
0006	2D
0007	2F
0008	24
0009	12
000A	80
000B	FE

Chương trình 2.1: Tệp liệt kê

Sau khi chương trình được đốt vào trong ROM của thành viên họ 8051 như 8751 hoặc AT 8951 hoặc DS 5000 thì mã lệnh và toán hạng được đưa vào các vị trí nhớ ROM bắt đầu từ địa chỉ 0000 như bảng liệt kê dưới đây.

Địa chỉ ROM	Ngôn ngữ máy	Hợp ngữ
0000	7D25	MOV R5, #25H
0002	7F34	MOV R7, #34H
0004	7400	MOV A, #0
0006	2D	ADD A, R5
0007	2F	ADD A, R7
0008	2412	ADD A, #12H
000A	80EF	HERE: SJMP HERE

Bảng nội dung ROM của chương trình 2.1.

Bảng liệt kê chỉ ra địa chỉ 0000 chứa mã 7D là mã lệnh để chuyển một giá trị vào thanh ghi R5 và địa chỉ 0001 chứa toán hạng (ở đây là giá trị 254) cần được chuyển vào R5. Do vậy, lệnh "MOV R5, #25H" có mã là "7D25" trong đó 7D là mã lệnh,

còng 25 là toán hạng. Tương tự như vậy, mã máy "7F34" được đặt trong các ngăn nhớ 0002 và 0003 và biểu diễn mã lệnh và toán hạng đối với lệnh "MOV R7, #34H". Theo cách như vậy, mã máy "7400" được đặt tại địa chỉ 0004 và 0005 và biểu diễn mã lệnh và toán hạng đối với lệnh "MOV A, #0". Ngăn nhớ 0006 có mã 2D là mã đối với lệnh "ADD A, R5" và ngăn nhớ 0007 có nội dung 2F là mã lệnh cho "ADD A, R7". Mã lệnh đối với lệnh "ADD A, #12H" được đặt ở ngăn nhớ 0008 và toán hạng 12H được đặt ở ngăn nhớ 0009. Ngăn nhớ 000A có mã lệnh của lệnh SJMP và địa chỉ đích của nó được đặt ở ngăn nhớ 000B. Lý do vì sao địa chỉ đích là FE được giải thích ở chương 3.

2.4.4 Thực hiện một chương trình theo từng byte.

Giả sử rằng chương trình trên được đốt vào ROM của chíp 8051 hoặc(8751, AT 8951 hoặc DS 5000) thì dưới đây là mô tả hoạt động theo từng bước của 8051 khi nó được cấp nguồn.

1. Khi 8051 được bật nguồn, bộ đếm chương trình PC có nội dung 0000 và bắt đầu nạp mã lệnh đầu tiên từ vị trí nhớ 0000 của ROM chương trình. Trong trường hợp của chương trình này là mã 7D để chuyển một toán hạng vào R5. Khi thực hiện mã lệnh CPU nạp giá trị 25 vào bộ đếm chương trình được tăng lên để chỉ đến 0002 (PC = 0002) có chứa mã lệnh 7F là mã của lệnh chuyển một toán hạng vào R7 "MOV R7, ...".
2. Khi thực hiện mã lệnh 7F thì giá trị 34H được chuyển vào R7 sau đó PC được tăng lên 0004.
3. Ngăn nhớ 0004 chứa mã lệnh của lệnh "MOV A, #0". Lệnh này được thực hiện và bây giờ PC = 0006. Lưu ý rằng tất cả các lệnh trên đều là những lệnh 2 byte, nghĩa là mỗi lệnh chiếm hai ngăn nhớ.
4. Bây giờ PC = 0006 chỉ đến lệnh kế tiếp là "ADD A, R5". Đây là lệnh một byte, sau khi thực hiện lệnh này PC = 0007.
5. Ngăn nhớ 0007 chứa mã 2F là mã lệnh của "ADD A, R7". Đây cũng là lệnh một byte, khi thực hiện lệnh này PC được tăng lên 0008. Qúa trình này cứ tiếp tục cho đến khi tất cả moi lệnh đều được nạp và thực hiện. Thực tế mà bộ đếm chương trình chỉ đến lệnh kế tiếp cần được thực hiện giải thích tại sao một số bộ vi xử lý (đáng nói là × 86) gọi bộ đếm là con trỏ lệnh (Instruction Pointer).

2.4.5 Bản đồ nhớ ROM trong họ 8051.

Như ta đã thấy ở chương trước, một số thành viên họ 8051 chỉ có 4k byte bộ nhớ ROM trên chíp (ví dụ 8751, AT 8951) và một số khác như AT 8951 có 8k byte ROM, DS 5000-32 của Dallas Semiconductor có 32k byte ROM trên chíp. Dallas Semiconductor cũng có motọ 8051 với ROM trên chíp là 64k byte. Điểm cần nhớ là không có thành viên nào của họ 8051 có thể truy cập được hơn 64k byte mã lệnh vì bộ đếm chương trình của 8051 là 16 bit (dải địa chỉ từ 0000 đến FFFFH). Cần phải ghi nhớ là lệnh đầu tiên của ROM chương trình đều đặt ở 0000, còn lệnh cuối cùng phụ thuộc vào dung lượng ROM trên chíp của mỗi thành viên họ 8051. Trong số các thành viên họ 8051 thì 8751 và AT 8951 có 4k byte ROM trên chíp. Bộ nhớ ROM trên chíp này có các địa chỉ từ 0000 đến 0FFFH. Do vậy, ngăn nhớ đầu tiên có địa chỉ 0000 và ngăn nhớ cuối cùng có địa chỉa 0FFFH. Hãy xét ví dụ 2.1.

VÝ dô 2.1:

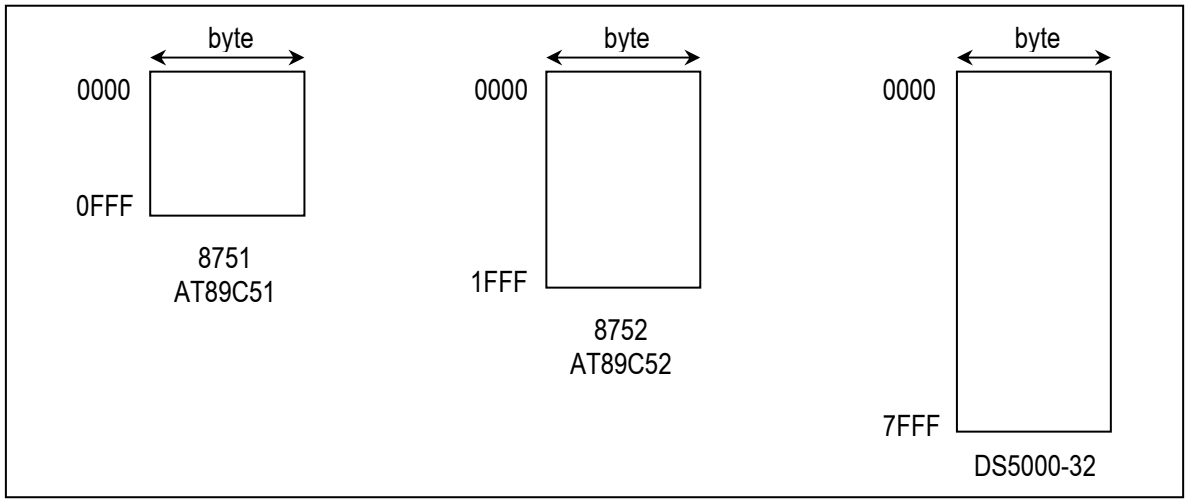
Tìm địa chỉ bộ nhớ ROM của mỗi thành viên họ 8051 sau đây.

a) AT 8951 (hoặc 8751) với 4k byte
b) DS 5000-32 với 32k byte

Lêi gi¶i:

a) Với 4k byte của không gian nhớ ROM trên chíp ta có 4096 byte bằng 1000H ở dạng Hex (4 × 1024 = 4096 hay 1000 ở dạng Hex). Bộ nhớ này được xắp xếp trong các ngăn nhớ từ 0000 đến 0FFFH. Lưu ý 0 luôn là ngăn nhớ đầu tiên.

b) Với 32k byte nhớ ta có 32.768 byte (32 × 1024). Chuyển đổi 32.768 về số Hex ta nhận được giá trị 8000H. Do vậy, không gian nhớ là dải từ 0000 đến 7FFFH.



Hình 2.3: Dải địa chỉ của ROM trên chíp một số thành viên họ 8051.

2.5 Các kiểu dữ liệu và các chỉ lệnh.

2.5.1 Kiểu dữ liệu và các chỉ lệnh của 8051.

Bộ vi điều khiển chỉ có một kiểu dữ liệu, nó là 8 bit và độ dài mỗi thanh ghi cũng là 8 bit. Công việc của lập trình viên là phân chia dữ liệu lớn hơn 8 bit ra thành từng khúc 8 bit (từ 00 đến FFH hay từ 0 đến 255) để CPU xử lý. Ví dụ về xử lý dữ liệu lớn hơn 8 bit được trình bày ở chương 6. Các dữ liệu được sử dụng bởi 8051 có thể là số âm hoặc số dương và về xử lý các số có dấu được bàn ở chương 6.

2.5.2 Chỉ lệnh DB (định nghĩa byte).

Chỉ lệnh 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 ASII. 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". Bất kể ta sử dụng số ở dạng thức nào thì hợp ngữ đều chuyển đổi chúng về thành dạng Hex. Để báo dạng thức ở dạng mã ASCII thì chỉ cần đơn giản đặt nó vào dấu nháy đơn 'như thế này'. Hợp ngữ sẽ gán mã ASCII cho các số hoặc các ký tự một cách tự động. Chỉ lệnh DB chỉ là chỉ lệnh mà có thể được sử dụng để định nghĩa các chuỗi ASCII lớn hơn 2 ký tự. Do vậy, nó có thể được sử dụng cho tất cả mọi định nghĩa dữ liệu ASCII. Dưới đây là một số ví dụ về DB:

```
        ORG    500H
DATA1:  DB     2B                ; Số thập phân (1C ở dạng Hex)
DATA2:  DB     00110101B         ; Số nhị phân (35 ở dạng Hex)
DATA3:  DB     39H               ; Số dạng Hex
               ORG    510H
DATA4:  DB     "2591"            ; Các số ASCII
               ORG    518H
DATA5   DB     "My name is Joe"  ; Các ký tự ASCII
```

Các chuỗi ASCII có thể sử dụng dấu nháy đơn 'như thế này' hoặc nháy kép "như thế này". Dùng dấu phẩy kép sẽ hữu ích hơn đối với trường hợp dấu nháy đơn được dùng sở hữu cách như thế này "Nhà O' Leary". Chỉ lệnh DB cũng được dùng để cấp phát bộ nhớ theo từng đoạn kích thước một byte.

2.5.3 Các chỉ lệnh của hợp ngữ.

1. Chỉ lệnh ORG: Chỉ lệnh ORG được dùng để báo bắt đầu của địa chỉ. Số đi sau ORG có kể ở dạng Hex hoặc thập phân. Nếu số này có kèm chữ H đằng sau thì là ở dạng Hex và nếu không có chữ H ở sau là số thập phân và hợp ngữ sẽ chuyển nó thành số Hex. Một số hợp ngữ sử dụng dấu chấm đứng trước "ORG" thay cho "ORG". Hãy đọc kỹ về trình hợp ngữ ta sử dụng.
2. Chỉ lệnh EQU: 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. Dưới đây sử dụng EQU cho hằng số bộ đếm và sau đó hằng số được dùng để nạp thanh ghi RS.

```
COUNT EQU 25
MOV   R3, #count
```

   Khi thực hiện lện "MOV R3, #COUNT" thì thanh ghi R3 sẽ được nạp giá trị 25 (chú ý đến dấu #). Vậy ưu điểm của việc sử dụng EQU là gì? Giả sử có một hằng số (một giá trị cố định) được dùng trong nhiều chỗ khác nhau trong chương trình và lập trình viên muốn thay đổi giá trị của nó trong cả chương trình. Bằng việc sử dụng chỉ lệnh EQU ta có thể thay đổi một số lần và hợp ngữ sẽ thay đổi tất cả mọi lần xuất hiện của nó là tìm toàn bộ chương trình và gắng tìm mọi lần xuất hiện.
3. Chỉ lệnh END: Một lệnh quan trọng khác là chỉ lệnh END. Nó báo cho trình hợp ngữ kết thúc của tệp nguồn "asm" chỉ lệnh END là dòng cuối cùng của chương trình 8051 có nghĩa là trong mã nguồn thì mọi thứ sau chỉ lệnh END để bị trình hợp ngữ bỏ qua. Một số trình hợp ngữ sử dụng .END có dấu chấm đứng trước thay cho END.

2.5.4 Các quy định đối với nhãn trong hợp ngữ.

Bằng cách chọn các tên nhãn có nghĩa là một lập trình viên có thể làm cho chương trình dễ đọc và dễ bảo trì hơn, có một số quy định mà các tên nhãn phải tuân theo. Thứ nhất là mỗi tên nhãn phải thống nhất, các tên được sử dụng làm nhãn trong hợp ngữ gồm các chữ cái viết hoa và viết thường, các số từ 0 đến 9 và các dấu đặc biệt như: dấu hỏi (?), dấu (≅), dấu gạch dưới (_), dấu đô là ($) và dấu chu kỳ (.). Ký tự đầu tiên của nhãn phải là một chữ cái. Hay nói cách khác là nó không thể là số Hex. Mỗi trình hợp ngữ có một số từ dự trữ là các từ gợi nhớ cho các lệnh mà không được dùng để làm nhãn trong chương trình. Ví dụ như "MOV" và "ADD". Bên cạnh các từ gợi nhớ còn có một số tự dự trữ khác, hãy kiểm tra bản liệt kê các từ dự phòng của hợp ngữ ta đang sử dụng.

2.6 Các bit cờ và thanh ghi đặc bệt PSW của 8051.

Cũng như các bộ vi xử lý khác, 8051 có một thanh ghi cờ để báo các điều kiện số học như bit nhớ. Thanh ghi cờ trong 8051 được gọi là thanh ghi từ trạng thái chương trình PSW. Trong phần này và đưa ra một số ví dụ về cách thay đổi chúng.

2.6.1 Thanh ghi từ trạng thái chương trình PSW.

Thanh ghi PSW là thanh ghi 8 bit. Nó cũng còn được coi như là thanh ghi cờ. Mặc dù thanh ghi PSW rộng 8 bit nhưng chỉ có 6 bit được 8051 sử dụng. Hai bit chưa dùng là các cờ ch người dùng định nghĩa. Bốn trong số các cờ được gọi là các cờ có điều kiện, có nghĩa là chúng báo một số điều kiện do kết quả của một lệnh vừa được thực hiện. Bốn cờ này là cờ nhớ CY (carry), cờ AC (auxiliary cary), cờ chẵn lẻ P (parity) và cờ tràn OV (overflow).

Như nhìn thấy từ hình 2.4 thì các bit PSW.3 và PSW.4 được gán như RS0 và RS1 và chúng được sử dụng để thay đổi các thanh ghi băng. Chúng sẽ được giải thích ở phần kế sau. Các bit PSW.5 và PSW.1 là các bit cờ trạng thái công dụng chung và lập trình viên có thể sử dụng cho bất kỳ mục đích nào.

CY	AC	F0	RS1	RS0	OV	–	P

CY PSW.7 ; Cờ nhớ
AC PSW.6 ; Cờ

- PSW.5 ; Dành cho người dùng sử dụng mục đích chung

RS1 PSW.4 ; Bit = 1 chọn băng thanh ghi
RS0 PSW.3 ; Bit = 0 chọn băng thanh ghi
OV PSW.2 ; Cờ bận

- PSW.1 ; Bit dành cho người dùng định nghĩa

P PSW.0 ; Cờ chẵn, lẻ. Thiết lập/ xoá bằng phần cứng mỗi chu kỳ lệnh báo tổng các số bit 1 trong thanh ghi A là chẵn/ lẻ.

RS1	RS0	Băng thanh ghi	Địa chỉ
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

H×nh 2.4: **C¸c bit cña thanh ghi PSW**

Dưới đây là giải thích ngắn gọn về 4 bit cờ của thanh ghi PSW.

1. Cờ nhớ CY: Cờ này được thiết lập mỗi khi có nhớ từ bit D7. Cờ này được tác động sau lệnh cộng hoặc trừ 8 bit. Nó cũng được thiết lập lên 1 hoặc xoá về 0 trực tiếp bằng lệnh "SETB C" và "CLR C" nghĩa là "thiết lập cờ nhớ" và "xoá cờ nhớ" tương ứng. Về các lệnh đánh địa chỉ theo bit được bàn kỹ ở chương 8.
2. Cờ AC: Cờ này báo có nhớ từ bit D3 sang D4 trong phép cộng ADD hoặc trừ SUB. Cờ này được dùng bởi các lệnh thực thi phép số học mã BCD (xem ở chương 6).
3. Cờ chẵn lẻ P: Cờ chẵn lẻ chỉ phản ánh số bit một trong thanh ghi A là chẵn hay lẻ. Nếu thanh ghi A chứa một số chẵn các bit một thì P = 0. Do vậy, P = 1 nếu A có một số lẻ các bit một.
4. Cờ chàn OV: Cờ này được thiết lập mỗi khi kết quả của một phép tính số có dấu quá lớn tạo ra bit bậc cao làm tràn bit dấu. Nhìn chung cờ nhớ được dùng để phát hiện lỗi trong các phép số học không dấu. Còn cờ tràn được dùng chỉ để phát hiện lỗi trong các phép số học có dấu và được bàn kỹ ở chương 6.

2.6.2 Lệnh ADD và PSW.

Bây giờ ta xét tác động của lệnh ADD lên các bit CY, AC và P của thanh ghi PSW. Một số ví dụ sẽ làm rõ trạng thái của chúng, mặc dù các bit cờ bị tác động bởi lệnh ADD là CY, P, AC và OV nhưng ta chỉ tập trung vào các cờ CY, AC và P, còn cờ OV sẽ được nói đến ở chương 6 vì nó liên quan đến phép tính số học số có dấu.

Các ví dụ 2.2 đến 2.4 sẽ phản ánh tác động của lệnh ADD lên các bit nói trên.

Bảng 2.1: Các lệnh tác động lên các bit cờ.

Instruction	CY	OV	AC
ADD	X	X	X
ADDC	X	X	X
SUBB	X	X	X
MUL	0	X	
DIV	0	X	
DA	X		
RRC	X		
RLC	X		
SETB C	1		
CLR C	0		
CPL C	X		
ANL C, bit	X		
ANL C,/ bit	X		
ORL C, bit	X		
ORL C,/bit	X		
MOV C, bit	X		
CJNE	X		

Ví dụ 2.2: Hãy trình bày trạng thái các bit cờ CY, AC và P sau lệnh cộng 38H với 2FH dưới đây:

```
MOV     A, #38H
ADD  A, #2FH              ; Sau khi cộng A = 67H, CY = 0
```

Lêi gi¶i:

```
    38        00111000
+   2F        00101111
-------------------------
    67        01100111
```

Cờ CY = 0 vì không có nhớ từ D7
Cờ AC = 1 vì có nhớ từ D3 sang D4
Cờ P = 1 vì thanh ghi A có 5 bit 1 (lẻ)

VÝ dô 2.3:

Hãy trình bày trạng thái các cờ CY, AC và P sau phép cộng 9CH với 64H.

Lêi gi¶i:

```
    9C        10011100
+   64        01100100
-------------------------
   100        00000000
```

Cờ CY = 1 vì có nhớ qua bit D7
Cờ AC = 1 vì có nhớ từ D3 sang D4
Cờ P = 0 vì thanh ghi A không có bit 1 nào (chẵn)

VÝ dô 2.4:

Hãy trình bày trạng thái các cờ CY, AC và P sau phép cộng 88H với 93H.

Lêi gi¶i:

```
    88        10001000
+   93        10010011
-------------------------
   11B        00011011
```

Cờ CY = 1 vì có nhớ từ bit D7
Cờ AC = 0 vì không có nhớ từ D3 sang D4
Cờ P = 0 vì số bit 1 trong A là 4 (chẵn)

2.7 Các băng thanh ghi và ngăn xếp của 8051.

Bộ vi điều khiển 8051 có tất cả 128 byte RAM. Trong mục này ta bàn vệ phân bố của 128 byte RAM này và khảo sát công dụng của chúng như các thanh ghi và ngăn xếp.

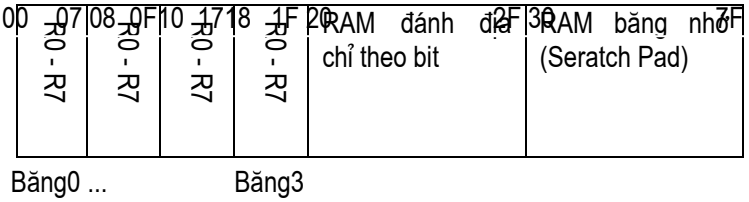
2.7.1 Phân bố không gian bộ nhớ RAM trong 8051.

Có 128 byte RAM trong 8051 (một số thành viên đang chú ý là 8052 có 256 byte RAM). 128 byte RAM bên trong 8051 được gán địa chỉ từ 00 đến 7FH. Như ta sẽ thấy ở chương 5, chúng có thể được truy cập trực tiếp như các ngăn nhớ 128 byte RAM này được phân chia thành từng nhóm như sau:

1. Tổng cộng 32 byte từ ngăn nhớ 00 đến 1FH được dành cho các thanh ghi và ngăn xếp.
2. Tổng cộng 16 byte từ ngăn nhớ 20H đến 2FH được dành cho bộ nhớ đọc/ ghi đánh địa chỉ được theo bit. Chương 8 sẽ bàn chi tiết về bộ nhớ và các lệnh đánh địa chỉ được theo bit.
3. Tổng cộng 80 byte từ ngăn nhớ 30H đến 7FH được dùng cho lưu đọc và ghi hay như vẫn thường gọi là bảng nháp (Serach pad). Những ngăn nhớ này (80 byte) của RAM được sử dụng rộng rãi cho mục đích lưu dữ liệu và tham số bởi các lập trình viên 8051. Chúng ta sẽ sử dụng chúng ở các chương sau để lưu dữ liệu nhận vào CPU qua các cổng vào-ra.

2.7.2 Các băng thanh ghi trong 8051.

Như đã nói ở trước, tổng cộng 32 byte RAM được dành riêng cho các băng thanh ghi và ngăn xếp. 32 byte này được chia ra thành 4 băng các thanh ghi trong đó mỗi băng có 8 thanh ghi từ R0 đến R7. Các ngăn nhớ RAM số 0, R1 là ngăn nhớ RAM số 1, R2 là ngăn nhớ RAM số 2 v.v... Băng thứ hai của các thanh ghi R0 đến R7 bắt đầu từ thanh nhớ RAM số 2 cho đến ngăn nhớ RAM số 0FH. Băng thứ ba bắt đầu từ ngăn nhớ 10H đến 17H và cuối cùng từ ngăn nhớ 18H đến 1FH là dùng cho băng các thanh ghi R0 đến R7 thứ tư.



Hình 2.5: Ngăn xếp các thanh nhớ RAM trong 8051.

Bank 0		Bank 1		Bank 2		Bank 3	
7	R7	F	R7	17	R7	1F	R7
6	R6	E	R6	16	R6	1E	R6
5	R5	D	R5	15	R5	1D	R5
4	R4	C	R4	14	R4	1C	R4
3	R3	B	R3	13	R3	1B	R3
2	R2	A	R2	12	R2	1A	R2
1	R1	9	R1	11	R1	19	R1
0	R0	8	R0	10	R0	18	R0

Hình 2.6: Các băng thanh ghi của 8051 và địa chỉ của chúng.

Như ta có thể nhìn thấy từ hình 2.5 băng 1 sử dụng cùng không gian RAM như ngăn xếp. Đây là một vấn đề chính trong lập trình 8051. Chúng ta phải hoặc là không sử dụng băng 1 hoặc là phải đánh một không gian khác của RAM cho ngăn xếp.

VÝ dô 2.5:

Hãy phát biểu các nội dung của các ngăn nhớ RAM sau đoạn chương trình sau:

```
MOV   R0, #99H          ; Nạp R0 giá trị 99H
MOV   R1, #85H          ; Nạp R1 giá trị 85H
MOV   R2, #3FH          ; Nạp R2 giá trị 3FH
MOV   R7, #63H          ; Nạp R7giá trị 63H
MOV   R5, #12H          ; Nạp R5 giá trị12H
```

Lêi gi¶i:

Sau khi thực hiện chương trình trên ta có:

Ngăn nhớ 0 của RAM có giá trị 99H

Ngăn nhớ 1 của RAM có giá trị 85H

Ngăn nhớ 2 của RAM có giá trị 3FH

Ngăn nhớ 7 của RAM có giá trị 63H

Ngăn nhớ 5 của RAM có giá trị 12H

2.6.3 Băng thanh ghi mặc định.

Nếu các ngăn nhớ 00 đến 1F được dành riêng cho bốn băng thanh ghi, vậy băng thanh ghi R0 đến R7 nào ta phải truy cập tới khi 8051 được cấp nguồn? Câu trả lời là các băng thanh ghi 0. Đó là các ngăn nhớ RAM số 0, 1, 2, 3, 4, 5, 6 và 7 được truy cập với tên R0, R1, R2, R3, R4, R5, R6 và R7 khi lập trình 8051. Nó dễ dàng hơn nhiều khi tham chiếu các ngăn nhớ RAM này ơví các tên R0, R1 v.v... hơn là số vị trí của các ngăn nhớ. Ví dụ 2.6 làm rõ khái niệm này.

VÝ dô 2.6:

Hãy viết lại chương trình ở ví dụ 2.5 sử dụng các địa chỉ RAM thay tên các thanh ghi.

Lêi gi¶i:

Đây được gọi là chế độ đánh địa chỉ trực tiếp và sử dụng địa chỉ các vị trí ngăn nhớ RAM đối với địa chỉ đích. Xem chi tiết ở chương 5 về chế độ đánh địa chỉ.

```
MOV   00, #99H          ; Nạp thanh ghi R0 giá trị 99H
MOV   01, #85H          ; Nạp thanh ghi R1 giá trị 85H
MOV   02, #3FH          ; Nạp thanh ghi R2 giá trị 3FH
MOV   07, #63H          ; Nạp thanh ghi R7giá trị 63H
MOV   05, #12H          ; Nạp thanh ghi R5 giá trị12H
```

2.6.4 Chuyển mạch các băng thanh ghi như thế nào?

Như đã nói ở trên, băng thanh ghi 0 là mặc định khi 8051 được cấp nguồn. Chúng ta có thể chuyển mạch sang các băng thanh ghi khác bằng cách sử dụng bit D3 và D4 của thanh ghi PSW như chỉ ra theo bảng 2.2.

Bảng 2.2: Bit lựa chọn các băng thanh ghi RS0 và RS1.

	RS1 (PSW.4)	RS0 (PSW.3)
Băng 0	0	0
Băng 1	0	1
Băng 2	1	0
Băng 3	1	1

Bit D3 và D4 của thanh ghi PSW thường được tham chiếu như là PSW.3 và PSW.4 vì chúng có thể được truy cập bằng các lệnh đánh địa chỉ theo bit như SETB và CLR. Ví dụ "SETB PSW.3" sẽ thiết lập PSW.3 và chọn băng thanh ghi 1. Xem ví dụ 2.7 dưới đây.

VÝ dô 2.7:

Hãy phát biểu nội dung các ngăn nhớ RAM sau đoạn chương trình dưới đây:

```
SETB    PSW.4           ; Chọn băng thanh ghi 4
MOV     R0, #99H        ; Nạp thanh ghi R0 giá trị 99H
MOV     R1, #85H        ; Nạp thanh ghi R1 giá trị 85H
MOV     R2, #3FH        ; Nạp thanh ghi R2 giá trị 3FH
MOV     R7, #63H        ; Nạp thanh ghi R7giá trị 63H
MOV     R5, #12H        ; Nạp thanh ghi R5 giá trị12H
```

Lêi gi¶i:

Theo mặc định PSW.3 = 0 và PSW.4 = 0. Do vậy, lệnh "SETB PSW.4" sẽ bật bit RS1 = 1 và RS0 = 0, bằng lệnh như vậy băng thanh ghi R0 đến R7 số 2 được chọn. Băng 2 sử dụng các ngăn nhớ từ 10H đến 17H. Nên sau khi thực hiện đoạn chương trình trên ta có nội dung các ngăn nhớ như sau:

Ngăn nhớ vị trí 10H có giá trị 99H
Ngăn nhớ vị trí 11H có giá trị 85H
Ngăn nhớ vị trí 12H có giá trị 3FH
Ngăn nhớ vị trí 17H có giá trị 63H
Ngăn nhớ vị trí 15H có giá trị 12H

2.6.5 Ngăn xếp trong 8051.

Ngăn xếp là một vùng bộ nhớ RAM được CPU sử dụng để lưu thông tin tạm thời. Thông tin này có thể là dự liệu, có thể là địa cỉ CPU cần không gian lưu trữ này vì số các thanh ghi bị hạn chế.

2.6.6 Cách truy cập các ngăn xếp trong 8051.

Nếu ngăn xếp là một vùng của bộ nhớ RAM thì phải có các thanh ghi trong CPU chỉ đến nó. Thanh được dùng để chỉ đến ngăn xếp được gọi là thanh ghi con trỏ ngăn xếp SP (Stack Pointer). Con trỏ ngăn xếp trong 8051 chỉ rộng 8 bit có nghĩa là nó chỉ có thể có thể được các địa chỉ từ 00 đến FFH.

Khi 8051 được cấp nguồn thì SP chứa giá trị 07 có nghĩa là ngăn nhớ 08 của RAM là ngăn nhớ đầu tiên được dùng cho ngăn xếp trong 8051. Việc lưu lại một thanh ghi

PCU trong ngăn xếp được gọi là một lần cất vào PUSH và việc nạp nội dung của ngăn xếp trở lại thanh ghi CPU được gọi là lấy ra POP. Hay nói cách khác là một thanh ghi được cất vào ngăn xếp để lưu cất và được lấy ra từ ngăn xếp để dùng tiếp công việc của SP là rất nghiêm ngặt mỗi khi thao tác cất vào (PUSH) và lấy ra (POP) được thực thi. Để biết ngăn xếp làm việc như thế nào hãy xét các lệnh PUSH và POP dưới đây.

2.6.7 Cất thanh ghi vào ngăn xếp.

Trong 8051 thì con trỏ ngăn xếp chỉ đến ngăn nhớ sử dụng cuối cùng của ngăn xếp. Khi ta cất dữ liệu vào ngăn xếp thì con trỏ ngăn xếp SP được tăng lên 1. Lưu ý rằng điều này đối với các bộ vi xử lý khác nhau là khác nhau, đáng chú ý là các bộ vi xử lý × 86 là SP giảm xuống khi cất dữ liệu vào ngăn xếp. Xét ví dụ 2.8 dưới đây, ta thấy rằng mỗi khi lệnh PUSH được thực hiện thì nội dung của thanh ghi được cất vào ngăn xếp và SP được tăng lên 1. Lưu ý là đối với mỗi byte của dữ liệu được cất vào ngăn xếp thì SP được tăng lên 1 lần. Cũng lưu ý rằng để cất các thanh ghi vào ngăn xếp ta phải sử dụng địa chỉ RAM của chúng. Ví dụ lệnh "PUSH 1" là cất thanh ghi R1 vào ngăn xếp.

VÝ dô 2.8:

Hãy biểu diễn ngăn xếp và con trỏ ngăn xếp đối với đoạn chương trình sau đây. Giả thiết vùng ngăn xếp là mặc định.

```
MOV     R6, #25H
MOV     R1, #12H
MOV     R4, #0F3H
PUSH    6
PUSH    1
PUSH    4
```

Lêi gi¶i:

	Sau PUSH 6	Sau PUSP 1	Sau PUSH 4
0B	0B	0B	0B
0A	0A	0A	0A F3
09	09	09 12	09 12
08	08 25	08 25	08 25
Bắt đầu SP = 07	SP = 08	SP = 09	SP = 0A

2.6.8 Lấy nôi dung thanh ghi ra từ ngăn xếp.

Việc lấy nội dung ra từ ngăn xếp trở lai thanh ghi đã cho là qúa trình ngược với các nội dung thanh ghi vào ngăn xếp. Với mỗi lần lấy ra thì byte trên đỉnh ngăn xếp được sao chép vào thanh ghi được xác định bởi lệnh và con trỏ ngăn xếp được giảm xuống 1. Ví dụ 2.9 minh hoạ lệnh lấy nội dung ra khỏi ngăn xếp.

VÝ dô 2.9:

Khảo sát ngăn xếp và hãy trình bày nội dung của các thanh ghi và SP sau khi thực hiện đoạn chương trình sau đây:

```
POP     3       ; Lấy ngăn xếp trở lại R3
POP     5       ; Lấy ngăn xếp trở lại R5
```

```
POP     2       ; Lấy ngăn xếp trở lại R2
```

Lêi gi¶i:

		Sau POP3		Sau POP 5		Sau POP 2	
0B	54	0B		0B		0B	
0A	F9	0A	F9	0A		0A	
09	76	09	76	09	76	09	
08	6C	08	6C	08	6C	08	6C
Bắt đầu SP = 0B		SP = 0A		SP = 09		SP = 08	

2.6.9 Giới hạn trên của ngăn xếp.

Như đã nói ở trên, các ngăn nhớ 08 đến 1FH của RAM trong 8051 có thể được dùng làm ngăn nhớ 20H đến 2FH của RAM được dự phòng cho bộ nhớ đánh địa chỉ được theo bit và không thể dùng trước cho ngăn xếp. Nếu trong một chương trình đã cho ta cần ngăn xếp nhiều hơn 24 byte (08 đến 1FH = 24 byte) thì ta có thể đổi SP chỉ đến các ngăn nhớ 30 đến 7FH. Điều này được thực hiển bởi lệnh "MOV SP, #XX".

2.6.10 Lệnh gọi CALL và ngăn xếp.

Ngoài việc sử dụng ngăn xếp để lưu cất các thanh ghi thì CPU cũng sử dụng ngăn xếp để lưu cất tam thời địa chỉ của lệnh đứng ngay dưới lệnh CALL. Điều này chính là để PCU biết chỗ nào để quay trở về thực hiện tiếp các lệnh sau khi chọn chương trình con. Chi tiết về lệnh gọi CALL được trình bỳ ở chương 3.

2.6.11 Xung đột ngăn xếp và băng thanh ghi số 1.

Như ta đa nói ở trên thì thanh ghi con trỏ ngăn xếp có thể chỉ đến vị trí RAM hiện thời dành cho ngăn xếp. Khi dữ liệu được lưu cất cào ngăn xếp thì SP được tăng lên và ngược lại khi dữ liệu được lấy ra từ ngăn xếp thì SP giảm xuống. Lý do là PS được tăng lên sau khi PUSH là phải biết lấy chắc chắn rằng ngăn xếp đang tăng lên đến vị trí ngăn nhớ 7FH của RAM từ địa chỉ thấp nhất đến địa chỉ cao nhất. Nếu con trỏ ngăn xếp đã được giảm sau các lệnh PUSH thì ta nên sử dụng các ngăn nhớ 7, 6, 5 v.v... của RAM thuộc các thanh ghi R7 đến R0 của băng 0, băng thanh ghi mặc định. Việc tăng này của con trỏ ngăn xếp đối với các lệnh PUSH cũng đảm bảo rằng ngăn xếp sẽ không với tới ngăn nhớ 0 của RAM (đáy của RAM) và do vậy sẽ nhảy ra khỏi không gian dành cho ngăn xếp. Tuy nhiên có vấn đề nảy sinh với thiết lập mặc định của ngăn xếp. Ví dụ SP = 07 khi 8051 được bật nguồn nên RAM và cũng thuộc về thanh ghi R0 củ băng thanh ghi số 1. Hay nói cách khác băng thanh ghi số 1 và ngăn xếp đang dùng chung một không gian của bộ nhớ RAM. Nếu chương trình đã cho cần sử dụng các băng thanh ghi số 1 và số 2 ta có thể đặt lại vùng nhớ RAM cho ngăn xếp. Ví dụ, ta có thể cấp vị trí ngăn nhớ 60H của RAM và cao hơn cho ngăn xếp trong ví dụ 2.10.

VÝ dô 2.10:

Biểu diễn ngăn xếp và con trỏ ngăn xếp đối với các lệnh sau:

```
MOV     SP, #5FH            ; Đặt ngăn nhớ từ 60H của RAM cho ngăn xếp
MOV     R2, #25H
MOV     R1, #12H
MOV     R4, #0F3H
PUSH    2
```

PUSH 1
PUSH 4

Lêi gi¶i:

	Sau PUSH 2	Sau PUSP 3	Sau PUSH 4
63	63	63	63
62	62	62	62 F3
61	61	61 12	61 12
60	60 25	60 25	60 25
Bắt đầuSP=5F	SP = 60	SP = 61	SP = 62

CHƯƠNG 3
Các lệnh nhảy, vòng lặp và lệnh gọi

Trong một chuỗi lệnh cần thực hiện thường có nhu cần cần chuyển điều khiển chương trình đến một vị trí khác. Có nhiều lệnh để thực hiện điều này trong 8051, ở chương này ta sẽ tìm hiểu các lệnh chuyển điều khiển có trong hợp ngữ của 8051 như các lệnh sử dụng cho vòng lặp, các lệnh nhảy có và không có điều khiển, lệnh gọi và cuối cùng là mô tả về một chương trình con giữ chậm thời gian.

3.1 Vòng lặp và các lệnh nhảy.

3.1.1 Tạo vòng lặp trong 8051.

Qúa trình lặp lại một chuỗi các lệnh với một số lần nhất định được gọi là vòng lặp. Vòng lặp là một trong những hoạt động được sử dụng rộng rãi nhất mà bất kỳ bộ vi sử lý nào đều thực hiện. Trong 8051 thì hoạt động vòng lặp được thực hiện bởi lệnh "DJNZ thanh ghi, nhãn". Trong lệnh này thanh ghi được giảm xuống, nếu nó không bằng không thì nó nhảy đến địa chỉ đích được tham chiếu bởi nhãn. Trước khi bắt đầu vòng lặp thì thanh ghi được nạp với bộ đếm cho số lần lặp lại. Lưu ý rằng, trong lệnh này việc giảm thanh ghi và quyết định để nhảy được kết hợp vào trong một lệnh đơn.

Ví dụ 3.1:

Viết một chương trình để: a) xoá ACC và sau đó b) cộng 3 vào ACC 10 lần.

Lời giải:

```
        MOV     A, #0       ; Xoá ACC,  A = 0
        MOV     R2, #10     ; Nạp bộ đếm R2 = 10
BACK:   ADD     A, #10      ; Cộng 03 vào ACC
        DJNZ    R2, AGAIN   ; Lặp lại cho đến khi R2 = 0 (10 lần)
        MOV     R5, A       ; Cắt A vào thanh ghi R5
```

Trong chương trình trên đây thanh ghi R2 được sử dụng như là bộ đếm. Bộ đếm lúc đầu được đặt bằng 10. Mỗi lần lặp lại lệnh DJNZ giảm R2 không bằng 0 thì nó nhảy đến địa chỉ đích gắn với nhãn "AGAIN". Hoạt động lặp lại này tiếp tục cho đến khi R2 trở về không. Sau khi R2 = 0 nó thoát khỏi vòng lặp và thực hiện đứng ngay dưới nó trong trường hợp này là lệnh "MOV R5, A".

Lưu ý rằng trong lệnh DJNZ thì các thanh ghi có thể là bất kỳ thanh ghi nào trong các thanh ghi R0 - R7. Bộ đếm cũng có thể là một ngăn nhớ trong RAM như ta sẽ thấy ở chương 5.

Ví dụ 3.2:

Số lần cực đại mà vòng lặp ở ví dụ 3.1 có thể lặp lại là bao nhiêu?

Lời giải:

Vì thanh ghi R2 chứa số đếm và nó là thanh ghi 8 bit nên nó có thể chứa được giá trị cực đại là FFH hay 155. Do vậy số lần lặp lại cực đại mà vòng lặp ở ví dụ 3.1 có thể thực hiện là 256.

3.2.1 Vòng lặp bền trong một vòng lặp.

Như trình bày ở ví dụ 3.2 số đếm cực đại là 256. Vậy điều gì xảy ra nếu ta muốn lặp một hành động nhiều hơn 256 lần? Để làm điều đó thì ta sử dụng một vòng

lặp bên trong một vòng lặp được gọi là vòng lặp lồng (Nested Loop). Trong một vòng lặp lồng ta sử dụng 2 thanh ghi để giữ số đếm. Xét ví dụ 3.3 dưới đây.

Ví dụ 3.3:

Hãy viết một chương trình a) nạp thanh ghi ACC với giá trị 55H và b) bù ACC 700 lần.

Lời giải:

Vì 700 lớn hơn 256 (là số cực đại mà một thanh ghi vó thể chứa được) nên ta phải dùng hai thanh ghi để chứa số đếm. Đoạn mã dưới đây trình bày cách sử dụng hai thanh ghi R2 và R3 để chứa số đếm.

```
        MOV     A, #55H     ; Nạp A = 55H
        MOV     R3, #10     ; Nạp R3 = 10 số đếm vòng lặp ngoài
NEXT:   MOV     R2, #70     ; Nạp R2 = 70 số đếm vòng lặp trong
AGAIN: `CPL     A           ; Bù thanh ghi A
        DJNZ    R2, AGAIN   ; Lặp lại 70 lần (vòng lặp trong)
        DJNZ    R3, NEXT
```

Trong chương trình này thanh ghi R2 được dùng để chứa số đếm vòng lặp trong. Trong lệnh "DJNZ R2, AGAIN" thì mỗi khi R2 = 0 nó đi thẳng xuống và lệnh "JNZ R3, NEXT" được thực hiện. Lệnh này ép CPU nạp R2 với số đếm 70 và vòng lặp trong khi bắt đầu lại quá trình này tiếp tục cho đến khi R3 trở về không và vòng lặp ngoài kết thúc.

3.1.3 Các lệnh nhảy có điều kiện.

Các lệnh nhảy có điều kiện đối với 8051 được tổng hợp trong bảng 3.1. Các chi tiết về mỗi lệnh được cho trong phụ lục AppendixA. Trong bảng 3.1 lưu ý rằng một số lệnh như JZ (nhảy nếu A = 0) và JC (nhảy nếu có nhớ) chỉ nhảy nếu một điều kiện nhất định được thoả mãn. Kế tiếp ta xét một số lệnh nhảy có điều kiện với các **Ví dụ minh hoạ sau.**

a- Lệnh JZ (nhảy nếu A = 0). Trong lệnh này nội dung của thanh ghi A được kiểm tra. Nếu nó bằng không thì nó nhảy đến địa chỉ đích. Ví dụ xét đoạn mã sau:

```
MOV     A, R0       ; Nạp giá trị của R0 vào A
JZ      OVER        ; Nhảy đến OVER nếu A = 0
MOV     A, R1       ; Nạp giá trị của R1 vào A
JZ      OVER        ; Nhảy đến OVER nếu A = 0
OVER ...
```

Trong chương trình này nếu R0 hoặc R1 có giá trị bằng 0 thì nó nhảy đến địa chỉ có nhãn OVER. Lưu ý rằng lệnh JZ chỉ có thể được sử dụng đối với thanh ghi A. Nó chỉ có thể kiểm tra xem thanh ghi A có bằng không không và nó không áp dụng cho bất kỳ thanh ghi nào khác. Quan trọng hơn là ta không phải thực hiện một lệnh số học nào như đếm giảm để sử dụng lệnh JNZ như ở ví dụ 3.4 dưới đây.

Ví dụ 3.4:

Viết một chương trình để xác định xem R5 có chứa giá trị 0 không? Nếu nạp thì nó cho giá trị 55H.

Lời giải:

```
        MOV     A, R5       ; Sao nội dung R5 vào A
        JNZ     NEXT        ; Nhảy đến NEXT nếu A không bằng 0
        MOV     R5, #55H    ;
NEXT:   ...
```

b- Lệnh JNC (nhảy nếu không có nhớ, cờ CY = 0).

Trong lệnh này thì bit cờ nhớ trong thanh ghi cờ PSW được dùng để thực hiện quyết định nhảy. Khi thực hiện lệnh "JNC nhãn" thì bộ xử lý kiểm tra cờ nhớ xem nó có được bật không (CY = 1). Nếu nó không bật thì CPU bắt đầu nạp và thực hiện các lệnh từ địa chỉ của nhãn. Nếu cờ CY = 1 thì nó sẽ không nhảy và thực hiện lệnh kế tiếp dưới JNC.

Cần phải lưu ý rằng cũng có lệnh "JC nhãn". Trong lệnh JC thì nếu CY = 1 nó nhảy đến địa chỉ đích là nhãn. Ta sẽ xét các ví dụ về các lệnh này trong các ứng dụng ở các chương sau.

Ngoài ra còn có lệnh JB (nhảy nếu bit có mức cao) và JNB (nhảy nếu bit có mức thấp). Các lệnh này được trình bày ở chương 4 và 8 khi nói về thao tác bit.

Bảng 3.1: Các lệnh nhảy có điều kiện.

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

Ví dụ 3.5:

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

Lời giải:

```
        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 = GE + 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
```

c- Tất cả các lệnh nhảy có điều kiện đều là những phép nhảy ngắn.

Cần phải lưu ý rằng tất cả các lệnh nhảy có điều kiện đều là các phép nhảy ngắn, có nghĩa là địa chỉ của đích đều phải nằm trong khoảng -127 đến +127 byte của nội dung bộ đếm chương trình PC.

3.1.4 Các lệnh nhảy không điều kiện.

Lệnh nhảy không điều kiện là một phép nhảy trong đó điều khiển được truyền không điều kiện đến địa chỉ đích. Trong 8051 có hai lệnh nhảy không điều kiện đó là: LJMP - nhảy xa và SJMP - nhảy gần.

a- 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.

Hãy nhớ rằng, mặc dù bộ đếm chương trình trong 8051 là 16 bit, do vậy cho không gian địa chỉ là 64k byte, nhưng bộ nhớ chương trình ROM trên chíp lớn như vậy. 8051 đầu tiên chỉ có 4k byte ROM trên chíp cho không gian chương trình, do vậy mỗi byte đều rất quý giá. Vì lý do đó mà có cả lệnh nhảy gần SJMP chỉ có 2 byte so với lệnh nhảy xa LZ0MP dài 3 byte. Điều này có thể tiết kiệm được một số byte bộ nhớ trong rất nhiều ứng dụng mà không gian bộ nhớ có hạn hẹp.

b- Lệnh nhảy gồm 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.

3.1.5 Tính toán địa chỉ lệnh nhảy gần.

Ngoài lệnh nhảy gần SJMP thì tất cả mọi lệnh nhảy có điều kiện như JNC, JZ và DJNZ đều là các lệnh nhảy gần bởi một thực tế là chúng đều lệnh 2 byte. Trong những lệnh này thì byte thứ nhất đều là mã lệnh, còn byte thứ hai là địa chỉ tương đối. Địa chỉ đích là tương đối so với giá trị của bộ đếm chương trình. Để tính toán địa chỉ đích byte thứ hai được cộng vào thanh ghi PC của lệnh đứng ngay sau lệnh nhảy. Để hiểu điều này hãy xét ví dụ 3.6 dưới đây.

Ví dụ 3.6:

Sử dụng tệp tin liệt kê dưới đây hãy kiểm tra việc tín toán địa chỉ nhảy về trước.

```
01      0000                            ORG         0000
02      0000    7800                    MOV         R0, #0
03      0002    7455                    MOV         A, #55H
04      0004    6003                    JZ          NEXT
05      0006    08                      NIC         R0
06      0007    04          AGAIN: INC          A
07      0008    04                      INC         A
08      0009    2477        NEXT:       ADD         A, #77h
09      000B    5005                    JNC         OVER
10      000D    E4                      CLR         A
```

```
11      000E        F8                      MOV     R0, A
12      000F        F9                      MOV     R1, A
13      0010        FA                      MOV     R2, A
14      0011        FB                      MOV     R3, A
15      0012        2B          OVER:       ADD     A, R3
16      0013        50F2                    JNC     AGAIN
17      0015        80FE        HERE:       SJMP    SHERE
18      0017                                END
```

Lời giải:

Trước hết lưu ý rằng các lệnh JZ và JNC đều là lệnh nhảy về trước. Địa chỉ đích đối với lệnh nhảy về trước được tính toán bằng cách cộng giá trị PC của lệnh đi ngay sau đó vào byte thứ hai của lệnh nhảy gần được gọi là địa chỉ tương đối. Ở dòng 04 lệnh "JZ NEXT" có mã lệnh 60 và toán hạng 03 tại địa chỉ 0004 và 0005. Ở đây 03 là địa chỉ tương đối, tương đối so với địa chỉ của lệnh kế tiếp là: "INC R0" và đó là 0006. Bằng việc cộng 0006 vào 3 thì địa chỉ đích của nhãn NEXT là 0009 được tạo ra. Bằng cách tương tự như vậy đối với dòng 9 thì lệnh "JNC OVER" có mã lệnh và toán hạng là 50 và 05 trong đó 50 là mã lệnh và 05 là địa chỉ tương đối. Do vậy, 05 được cộng vào OD là địa chỉ của lệnh "CLA A" đứng ngay sau lệnh "JNC OVER" và cho giá trị 12H chính là địa chỉ của nhãn OVER.

Ví dụ 3.7:

Hãy kiểm tra tính toán địa chỉ của các lệnh nhảy lùi trong ví dụ 3.6.

Lời giải:

Trong danh sách liệt kê chương trình đó thì lệnh "JNC AGAIN" có mã lệnh là 50 và địa chỉ tương đối là F2H. Khi địa chỉ tương đối của F2H được cộng vào 15H là địa chỉ của lệnh đứng dưới lệnh nhảy ta có 15H + F2H = 07 (và phần nhớ được bỏ đi). Để ý rằng 07 là địa chỉ nhãn AGAIN. Và hãy cũng xét lệnh "SJMP HERE" có mã lệnh 80 và địa chỉ tương đối FE giá trị PC của lệnh kế tiếp là 0017H được cộng vào địa chỉ tương đối FEH ta nhận được 0015H chính là địa chỉ nhãn HERE (17H + FEH = 15H) phần nhớ được bỏ đi). Lưu ý rằng FEH là -2 và 17h + (-2) = 15H. Về phép cộng số âm sẽ được bàn ở chương 6.

3.1.6 Tính toán địa chỉ đích nhảy lùi.

Trong khi ở trường hợp nhảy tới thì giá trị thay thế là một số dương trong khoảng từ 0 đến 127 (00 đến 7F ở dạng Hex) thì đối với lệnh nhảy lùi giá trị thay thế là một số âm nằm trong khoảng từ 0 đến -128 như được giải thích ở ví dụ 3.7.

Cần phải nhấn mạnh rằng, bất luận SJMP nhảy tới hay nhảy lùi thì đối với một lệnh nhảy bất kỳ địa chỉ của địa chỉ đích không bao giờ có thể lớn hơn 0 -128 đến +127 byte so với địa chỉ gắn liền với lệnh đứng ngay sau lệnh SJMP. Nếu có một sự nỗ lực nào vi phạm luật này thì hợp ngữ sẽ tạo ra một lỗi báo rằng lệnh nhảy ngoài phạm vi.

3.2 Các lệnh gọi CALL.

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 mà quyết định sử dụng lệnh nào đó phụ thuộc vào địa chỉ đích.

3.2.1 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. Do vậy LCALL có thể được dùng để gọi các chương trình con ở bất kỳ vị trí nào trong phạm vi 64k byte, không gian địa chỉ của 8051. Để đảm bảo rằng sau khi thực hiện một chương trình được gọi để 8051 biết được chỗ quay trở về thì nó tự động cất vào ngăn xếp địa chỉ của lệnh đứng ngay sau lệnh gọi LCALL. Khi một chương trình con được gọi, điều khiển được chuyển đến chương trình con đó và bộ xử lý cất bộ đếm chương trình PC vào ngăn xếp và bắt đầu nạp lệnh vào vị trí mới. Sau khi kết thúc thực hiện chương trình con thì lệnh trở về RET chuyển điều khiển về cho nguồn gọi. Mỗi chương trình con cần lệnh RET như là lệnh cuối cùng (xem ví dụ 3.8).

Các điểm sau đây cần phải được lưu ý từ ví dụ 3.8.

1. Lưu ý đến chương trình con DELAY khi thực hiện lệnh "LCALL DELAY" đầu tiên thì địa chỉ của lệnh ngay kế nó là "MOV A, #0AAH" được đẩy vào ngăn xếp và 8051 bắt đầu thực hiện các lệnh ở địa chỉ 300H.
2. Trong chương trình con DELAY, lúc đầu bộ đếm R5 được đặt về giá trị 255 (R5 = FFH). Do vậy, vòng lặp được lặp lại 256 lần. Khi R5 trở về 0 điều khiển rơi xuống lệnh quay trở về RET mà nó kéo địa chỉ từ ngăn xếp vào bộ đếm chương trình và tiếp tục thực hiện lệnh sau lệnh gọi CALL.

Ví dụ 3.8:

Hãy viết một chương trình để chốt tất cả các bit của cổng P1 bằng cách gửi đến nó giá trị 55H và AAH liên tục. Hãy đặt một độ trễ thời gian giữa mỗi lần xuất dữ liệu tới cổng P1. Chương trình này sẽ được sử dụng để kiểm tra các cổng của 8051 trong chương tiếp theo.

Lời giải:

```
                ORG     0000
BACK:           MOV     A, #55H         ; Nạp A với giá trị 55H
                MOV     P1, A           ; Gửi 55H đến cổng P1
                LCALL   DELAY           ; Tạo trễ thời gian
                MOV     A, #0AAH        ; Nạp A với giá trị  AAH
                MOV     P1, A           ; Gửi AAH đến cổng P1
                LCALL   DELAY           ; Giữ chậm
                SJMP    BACK            ; Lặp lại vô tận
; ------------------ - Đây là chương trình con tạo độ trễ thời gian
                ORG     300H            ; Đặt chương trình con trễ thời gian ở địa chỉ 300H
DELAY:          MOV     R5, #00H        ; Nạp bộ đếm R5 = 255H (hay FFH)
AGAIN:          DJNZ    R5, AGAIN       ; Tiếp tục cho đến khi R5 về không
                RET                     ; Trả điều khiển về nguồn gọi (khi R5 = 0)
                END                     ; Kết thúc tệp tin của hợp ngữ
```

Lượng thời gian trễ trong ví dụ 8.3 phục thuộc vào tần số của 8051. Cách tính chính xác thời gian sẽ được giải thích ở chương 4. Tuy nhiên ta có thể tăng thời gian độ trễ bằng cách sử dụng vòng lặp lồng như chỉ ra dưới đây.

```
DELAY:                                  ; Vòng lặp lồng  giữ chậm
                MOV     R4, #255        ; Nạp R4 = 255 (FFH dạng hex)
```

```
NEXT:   MOV   R5, #255     ; Nạp R5  = 255 (FFH dạng hex)
AGAIN:  DJNZ  R5, AGAIN    ; Lặp lại cho đến khi RT = 0
        DJNZ  R4, NEXT     ; Giảm R4
                           ;Tiếp tục nạp R5 cho đến khi R4 = 0
        RET                ; Trở về (khi R4 =  0)
```

3.2.2 Lệnh gọi CALL và vai trò của ngăn xếp.

Ngăn xếp và con trỏ ngăn xếp ta sẽ nghiên cứu ở chương cuối. Để hiểu được tầm quan trọng của ngăn xếp trong các bộ vi điều khiển bây giờ khảo sát nội dung của ngăn xếp và con trỏ ngăn xếp đối với ví dụ 8.3. Điều này được trình bày ở ví dụ 3.9 dưới đây.

Ví dụ 3.9:

Hãy phân tích nội dung của ngăn xếp sau khi thực hiện lệnh LCALL đầu tiên dưới đây.

```
001   0000                    OR6
002   0000  7455      BACK:   MOV   A, #55H      ; Nạp A với giá trị 55H
003   0002  F590              MOV   P1, A        ; Gửi 55H tới cổng P1
004   0004  120300            LCALL DELAY        ; Tạo trễ thời gian
005   0007  74AA              MOV   A, #0AAH     ; Nạp A với giá trị AAH
006   0009  F590              MOV   P1, A        ; Gửi AAH tới cổng      P1
007   000B  120300            LCALL DELAY        ; Tạo trễ thời gian
008   000E  80F0              SJMP  BACK         ; Tiếp tục thực hiện
009   0010
010   0010                    ; ..... Đây là chương trình con giữ chậm
011   0300                    MOV   300H
012   0300            DELAY:
013   0300  7DFF              MOV   R5, #FFH     ; Nạp R5 = 255
014   0302  DDFE      AGAIN:  DJNZ  R5, AGAIN    ; Dừng ở đây
015   0304  22                RET                ; Trở về nguồn gọi
016   0305                    END                ; Kết thúc nạp tin hợp ngữ
```

Lời giải:

Khi lệnh LCALL đầu tiên được thực hiện thì địa chỉ của lệnh "MOV A, #0AAH" được cất vào ngăn xếp. Lưu ý rằng byte thấp vào trước và byte cao vào sau. Lệnh cuối cùng của chương trình con được gọi phải là lệnh trở về RET để chuyển CPU kéo (POP) các byte trên đỉnh của ngăn xếp vào bộ đếm chương trình PC và tiếp tục thực hiện lệnh tại địa chỉ 07. Sơ đồ bên chỉ ra khung của ngăn xếp sau lần gọi LCALL đầu tiên.

0A		
09		00
08		07
SP	=	09

3.2.3 Sử dụng lệnh PUSH và POP trong các chương trình con.

Khi gọi một chương trình con thì ngăn xếp phải bám được vị trí mà CPU cần trở về. Sau khi kết thúc chương trình con vì lý do này chúng ta phải cẩn thận mỗi khi thao tác với các nội dung của ngăn xếp. Nguyên tắc là số lần đẩy vào (PUSH) và kéo

ra (POP) luôn phải phù hợp trong bất kỳ chương trình con được gọi vào. Hay nói cách khác đối với mỗi lệnh PUSH thì phải có một lệnh POP. Xem ví dụ 3.10.

3.2.4 Gọi các chương trình con.

Trong lập trình hợp ngữ thường có một chương trình chính và rất nhiều chương trình con mà chúng được gọi từ chương trình chính. Điều này cho phép ta tạo mới chương trình con trong một mô-đun riêng biệt. Mỗi mô-đun có thể được kiểm tra tách biệt và sau đó được kết hợp với nhau cùng với chương trình chính. Quan trọng hơn là trong một chương trình lớn thì các mô-đun có thể được phân cho các lập trình viên khác nhau nhằm rút ngắn thời gian phát triển.

Ví dụ 3.10:

Phân tích ngăn xếp đối với lệnh LCALL đầu tiên trong đoạn mã.

```
01  0000                    ORG     0
02  0000  7455      BACK:   MOV     A, #55H     ; Nạp A với giá trị 55H
03  0002  F590              MOV     P1, A       ; Gửi 55H ra cổng P1
04  0004  7C99              MOV     R4, #99H
05  0006  7D67              MOV     R5, #67H
06  0008  120300            LCALL   DELAY       ; Tạo giữ chậm thời gian
07  000B  74AA              MOV     A, #0AAH    ; Nạp A với AAH
08  000D  F590              MOV     P1, A       ; Gửi AAH ra cổng P1
09  000F  120300            LCALL   DELAY
10  0012  80EC              SJMP    BACK        ; Tiếp tục thực hiện
11  0014                    ; ..... Đây là chương trình con DELAY
12  0300                    ORG     300H
13  0300  C004      DELAY   PUSH    4           ; Đẩy R4 vào ngăn xếp
14  0302  C005              PUSH    5           ; Đẩy R5 vào ngăn xếp
15  0304  7CFF              MOV     R4, 00FH    ; Gán R4 = FFH
16  0306  7DFF      NEXT:   MOV     R5,  #00FH  ; Gán R5 = 255
17  0308  DDFE      AGAIN:  DJNZ    R5, AGAIN
18  030A  DCFA              DJNZ    R4, NEXT
19  030C  D005              POP     5           ; Kéo đỉnh ngăn xếp vào R5
20  030E  D004              POP     4           ; Kéo đỉnh ngăn xếp vào R4
21  0310  22                RET                 ; Trở về nguồn gọi
22        0311              END                 ; Kết thúc tệp tin hợp ngữ
```

Lời giải:

Trước hết lưu ý rằng đối với các lệnh PUSH và POP ta phải xác định địa chỉ trực tiếp của thanh ghi được đẩy vào, kéo ra từ ngăn xếp. Dưới đây là sơ đồ khung của ngăn xếp.

Sau lệnh LCALL thứ nhất			Sau lệnh PUSH 4			Sau lệnh POSH 5		
0B			0B			0B	67	R5
0A			0A	99	R4	0A	09	R4
09	00	PCH	09	00	PCH	09	00	PCL
08	0B	PCL	0B	0B	PCL	08	0B	PCL

Cần phải nhấn mạnh rằng trong việc sử dụng LCALL thì địa chỉ đích của các chương trình con có thể ở đâu đó trong phạm vi 64k byte không gian bộ nhớ của

8051. Điều này không áp dụng cho tất cả mọi lệnh gọi CALL chẳng hạn như đối với ACALL dưới đây:

```
; MAIN program calling subroutines
                ORG             0
MAIN:           LCALL           SUBR-1
                LCALL           SUBR-2
                LCALL           SUBR-3

HERE:           SJMP            MAIN
;---------------- end of MAIN
;
SUBR-1l         ...
                ...
                RET
; ---------------- end of subroutinel  1
; SUBR-1l       ...
                ...
                RET
; ---------------- end of subroutinel  2
 ; SUBR-1l      ...
                ...
                RET
; ---------------- end of subroutinel  3
                END                     ; end of the asm    file
```

Hình 3.1: Chương trình chính hợp ngữ của 8051 có gọi các chương trình con.

3.2.5 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. Không có sự khác biệt nào giữa ACALL và LCALL trong khái niệm cất bộ đếm chương trình vào ngăn xếp hay trong chức năng của lệnh trở về RET. Sự khác nhau duy nhất là địa chỉ đích của lệnh LCALL có thể nằm bất cứ đâu trong phạm vi 64k byte không gian địa chỉ của 8051, còn trong khi đó địa chỉ của lệnh ACALL phải nằm trong khoảng 2 byte. Trong nhiều biến thế của 8051 do các hãng cung cấp thì ROM trên chíp chỉ có 1k byte.. Trong những trường hợp như vậy thì việc sử dụng ACALL thay cho LCALL có thể tiết kiệm được một số byte bộ nhớ của không gian ROM chương trình.

Ví dụ 3.11:

Một nhà phát triển sử dụng chíp vi điều khiển Atmel AT89C1051 cho một sản phẩm. Chíp này chỉ có 1k byte ROM Flash trên chíp. Hỏi trong khi lệnh LCALL và ACALL thì lệnh nào hữu ích nhất trong lập trình cho chíp này.

Lời giải:

Lệnh ACALL là hữu ích hơn vì nó là lệnh 2 byte. Nó tiết kiệm một byte mỗi lần gọi được sử dụng.

Tất nhiên, việc sử dụng các lệnh gọn nhẹ, chúng ta có thể lập trình hiệu quả bằng cách có một hiểu biết chi tiết về tất cả các lệnh được hỗ trợ bởi bộ vi xử lý đã cho và sử dụng chúng một cách khôn ngoan. Xét ví dụ 3.12 dưới đây.

Ví dụ 3.12:

Hãy viết lại chương trình ở ví dụ 3.8 một cách hiệu quả mà bạn có thể:

Lời giải:

```
        ORG     0
        MOV     A, #55H         ; Nạp Avới giá trị 55H
BACK:   MOV     P1, A           ; Xuất giá trị trong A ra cổng P1
        ACALL   DELAY           ; Giữ chậm
        CPL     A               ; Bù thành ghi A
        SJMP    BACK            ; Tiếp tục thực hiện vô hạn
; -------- Đây là chương trình con giữ chậm DELAY
DELAY:
        MOV     R5,  #0FFH      ; Nạp R5 = 255 (hay FFH) làm cho bộ đếm
AGAIN:  DJNZ    R5, AGAIN       ; Dừng ở đây cho đến khi R5  = 0
        RET                     ; Trở về
        END                     ; Kết thúc
```

3.3 Tạo và tính toán thời gian giữ chậm.

3.3.1 Chu kỳ máy:

Đối với CPU để thực hiện một lệnh thì mất một chu kỳ đồng hồ này được coi như các chu kỳ máy. Phụ lục AppendixA.2 cung cấp danh sách liệt kê các lệnh 8051 và các chu kỳ máy của chúng. Để tính toán một độ trễ thời gian, ta sử dụng danh sách liệt kê này. Trong họ 8051 thì độ dài của chu kỳ máy phụ thuộc vào tần số của bộ dao động thạch anh được nối vào hệ thống 8051. Bộ dao động thạch anh cùng với mạch điện trên chip cung cấp xung đồng hồ cho CPU của 8051 (xem chương 4). Tần số của tinh thể thạch anh được nối tới họ 8051 dao động trong khoảng 4MHz đến 30 MHz phụ thuộc vào tốc độ chíp và nhà sản xuất. Thường xuyên nhất là bộ dao động thạch anh tần số 10.0592MHz được sử dụng để làm cho hệ 8051 tương thích với cổng nối tiếp của PC IBM (xem chương 10). Trong 8051, một chu kỳ máy kéo dài 12 chu kỳ dao động. Do vậy, để tính toán chu kỳ máy ta lấy 1/12 của tần số tinh thể thạch anh, sau đó lấy giá trị nghịch đảo như chỉ ra trong ví dụ 3.13.

Ví dụ 3.13:

Đoạn mã dưới đây trình bày tần số thạch anh cho 3 hệ thống dựa trên 8051 khác nhau. Hãy tìm chu kỳ máy của mỗi trường hợp: a) 11.0592MHz b) 16MHz và c) 20MHz.

Lời giải:

a) 11.0592/12 = 921.6kHz; Chu kỳ máy là 1/921.6kHz = 1.085μs (micro giây)
b) 16MHz/12 = 1.333MHz; Chu kỳ máy MC = 1/1.333MHz = 0.75μs
c) 20MHz/12 = 1.66MHz ⇒ MC = 1/1.66MHz = 0.60μs

Ví dụ 3.14:

Đối với một hệ thống 8051 có 11.0592MHz hãy tìm thời gian cần thiết để thực hiện các lệnh sau đây.

a) MOV R3, #55 b) DEC R3 c) DJNZ R2 đích

d) LJMP e) SJMP f) NOP g) MUL AB

Lời giải:

Chu kỳ máy cho hệ thống 8051 có tần số đồng hồ là 11.0592MHz Là 1.085μs như đã tính ở ví dụ 3.13. Bảng A-1 trong phụ lục Appendix A trình bày số chu kỳ máy đối với các lệnh trên. Vậy ta có:

Lệnh	Chu kỳ máy	Thời gian thực hiện
(a) MOV R3, #55	1	$1 \times 1.085\ \mu s = 1.085\ \mu s$
(b) DEC R3	1	$1 \times 1.085\ \mu s = 1.085\ \mu s$
(c) DJNZ R2, target	2	$2 \times 1.085\ \mu s = 2.17\ \mu s$
(d) LJMP	2	$2 \times 1.085\ \mu s = 2.17\ \mu s$
(e) SJMP	2	$2 \times 1.085\ \mu s = 2.17\ \mu s$
(f) NOP	1	$1 \times 1.085\ \mu s = 1.085\ \mu s$
(g) MUL AB	4	$4 \times 1.085\ \mu s = 4.34\ \mu s$

3.3.2 Tính toán độ trễ.

Như đã trình bày ở trên đây, một chương trình con giữ chậm gồm có hai phần: (1) thiết lập bộ đếm và (2) một vòng lặp. Hầu hết thời gian giữ chậm được thực hiện bởi thân vòng lặp như trình bày ở ví dụ 3.15.

Ví dụ 3.15:

Hãy tìm kích thước của thời gian giữ chậm trong chương trình sau, nếu tần số giao động thach anh là 11.0592MHz.

```
            MOV   A, #55H
AGAIN: MOV        P1, A
            ACALL DELAY
            CPL   A
            SJMP  AGAIN
; -------- Time delay
DELAY: MOV        R3, #200
HERE  :     DJNZ  R3, HERE
            RET
```

Lời giải:

Từ bảng A-1 của phụ lục Appendix A ta có các chu kỳ máy sao cho các lệnh của chương trình con giữ chậm là:

```
DELAY:    MOV     R3, #200     1
HERE  :   DJNZ    R3, HERE     2
          RET                  1
```

Do vậy tổng thời gian giữ chậm là $[(200 \times 2) + 1 + 1] \times 1.085 = 436.17\mu s$.

Thông thường ta tính thời gian giữ chậm dựa trên các lệnh bên trong vòng lặp và bỏ qua các chu kỳ đồng hồ liên quan với các lệnh ở ngoài vòng lặp.

Trong ví dụ 3.15 giá trị lớn nhất mà R3 có thể chứa là 255, do vậy một cách tăng độ trễ là sử dụng lệnh UOP (không làm gì) trong vòng lặp để tiêu tốn thời gian một cách đơn giản. Điều này được chỉ ra trong ví dụ 3.16 dưới đây.

Ví dụ 3.16:

Hãy tìm độ trễ thời gian cho chương trình con sau. Giả thiết tần số dao động thạch anh là 11.0592MHz.

```
                                        Số chu kỳ máy
DELAY:    MOV       R3, #250            1

HERE :    NOP                           1
          NOP                           1
          NOP                           1
          NOP                           1
          DJNZ      R3, HERE            2

          RET                           1
```

Lời giải:

Thời gian trễ bên trong vòng lặp HERE là [250 (1 + 1 + 1 + 1 + 1 + 2)] × 1.0851μs = 1627.5μs. Cộng thêm hai lệnh ngoài vòng lặp ta có 1627.5μs × 1.085μs = 1629.67μs.

3.3.3 Độ trễ thời gian của vòng lặp trong vòng lặp.

Một cách khác để nhận được giá trị từ độ trễ lớn là sử dụng một vòng lặp bên trong vòng lặp và cũng được gọi là vòng lặp lồng nhau. Xem ví dụ 3.17 dưới đây.

Ví dụ 3.17:

Đối với một chu kỳ máy 1.085μs hãy tính thời gian giữ chậm trong chương trình con sau:

```
DELAY:                                  chu kỳ máy
          MOV       R2, #200            1
AGAIN:    MOV       R3, #250            1
HERE:     NOP                           1
          NOP                           1
          DJNZ      R3, HERE            2
          DJNZ      R2, AGAIN           2
          RET                           1
```

Lời giải:

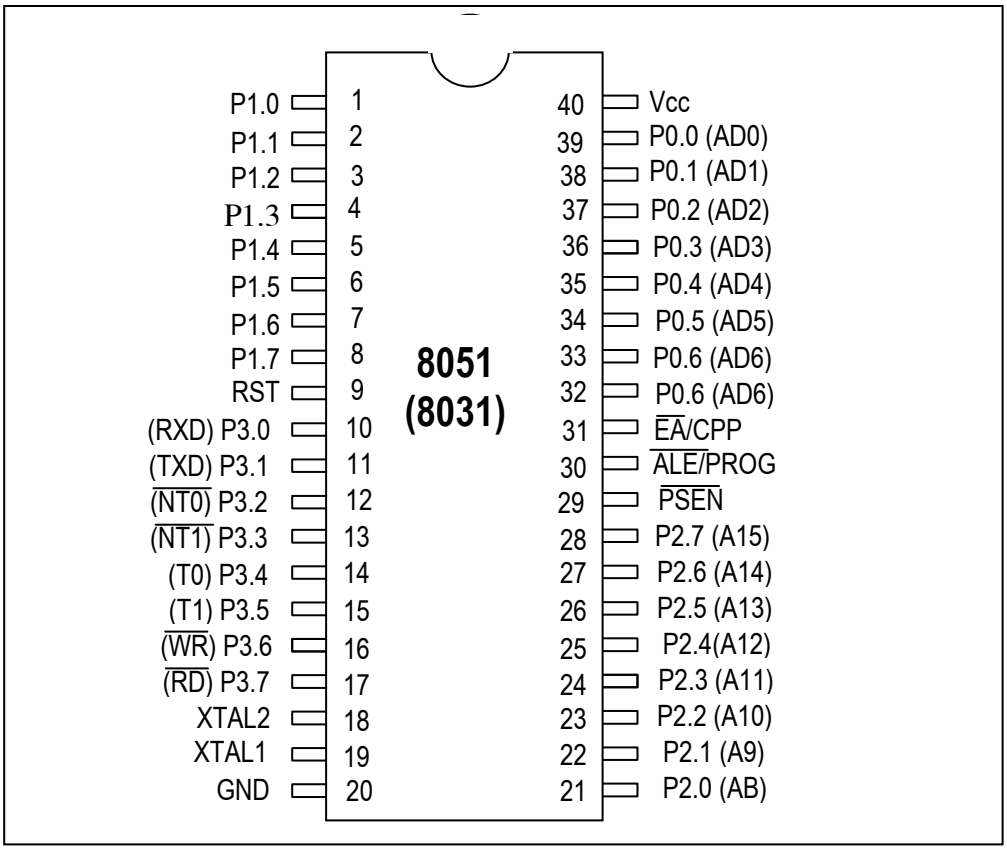
Đối với vòng lặp HERE ta có (4 × 250) × 1.085μs = 1085μs. Vòng lặp AGAIN lặp vòng lặp HERE 200 lần, do vậy thời gian trễ là 200 × 1085μs 217000μs, nên ta không tính tổng phí. Tuy nhiên, các lệnh "MOV R3, #250" và "DJNZ R2, AGAIN" ở đầu và cuối vòng lặp AGAIN cộng (3 × 200 × 1.085μs) = 651μs vào thời gian trễ và kết quả ta có 217000 + 651 = 217651μs = 217.651 miligiây cho tổng thời gian trễ liên quan đến chương trình con giữ chậm DELAY nói trên. Lưu ý rằng,

trong trường hợp vòng lặp lồng nhau cũng như trong mọi vòng lặp giữ chậm khác thời gian xấp xỉ gần dúng vì ta bỏ qua các lệnh đầu và cuối trong chương trình con.

CHƯƠNG 4
Lập trình cho cổng vào - ra I/0

4.1 Mô tả chân của 8051.

Mặc dù các thành viên của họ 8051 (ví dụ 8751, 89C51, DS5000) đều có các kiểu đóng vỏ khác nhau, chẳng hạn như hai hàng chân DIP (Dual In-Line Pakage) dạng vỏ dẹt vuông QFP (Quad Flat Pakage) và dạng chíp không có chân đỡ LLC (Leadless Chip Carrier) thì chúng đều có 40 chân cho các chức năng khác nhau như vào ra I/0, đọc $\overline{RD}$, ghi $\overline{WR}$, địa chỉ, dữ liệu và ngắt. Cần phải lưu ý rằng một số hãng cung cấp một phiên bản 8051 có 20 chân với số cổng vào-ra ít hơn cho các ứng dụng yêu cầu thấp hơn. Tuy nhiên, vì hầu hết các nhà phát triển chính sử dụng chíp đóng vỏ 40 chân với hai hàng chân DIP nên ta chỉ tập chung mô tả phiên bản này.



Hình 4.1: Sơ đồ bố trí chân của 8051.

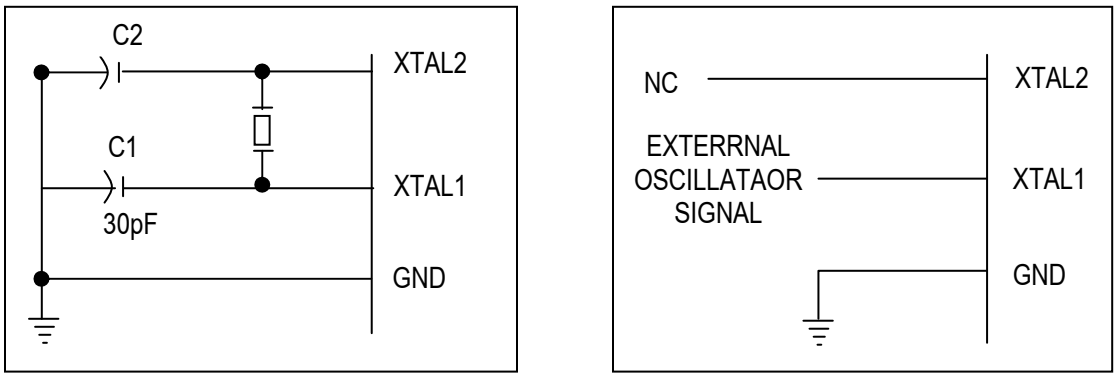
Trên hình 4.1 là sơ đồ bố trí chân của 8051. Ta thấy rằng trong 40 chân thì có 32 chân dành cho các cổng P0, P1, P2 và P3 với mỗi cổng có 8 chân. Các chân còn lại được dành cho nguồn V_{CC}, đất GND, các chângiao động XTAL1 và XTAL2 tái lập RST cho phép chốt địa chỉ ALE truy cập được địa chỉ ngoài $\overline{EA}$, cho phép cất chương trình $\overline{PSEN}$. Trong 8 chân này thì 6 chân V_{CC}, GND, XTAL1, XTAL2, RST và $\overline{EA}$ được các họ 8031 và 8051 sử dụng. Hay nói cách khác là chúng phải được

nối để cho hệ thống làm việc mà không cần biết bộ vi điều khiển thuộc họ 8051 hay 8031. Còn hai chân khác là $\overline{PSEN}$ và ALE được sử dụng chủ yếu trong các hệ thống dựa trên 8031.

1. Chân V_{CC}: Chân số 40 là V_{CC} cấp điện áp nguồn cho chíp. Nguồn điện áp là +5V.
2. Chân GND: Chân GND: Chân số 20 là GND.
3. Chân XTAL1 và XTAL2:

8051 có một bộ giao động trên chíp nhưng nó yêu cầu có một xung đồng hồ ngoài để chạy nó. Bộ giao động thạch anh thường xuyên nhất được nối tới các chân đầu vào XTAL1 (chân 19) và XTAL2 (chân 18). Bộ giao động thạch anh được nối tới XTAL1 và XTAL2 cũng cần hai tụ điện giá trị 30pF. Một phía của tụ điện được nối xuống đất như được trình bày trên hình 4.2a.

Cần phải lưu ý rằng có nhiều tốc độ khác nhau của họ 8051. Tốc độ được coi như là tần số cực đại của bộ giao động được nối tới chân XTAL. Ví dụ, một chíp 12MHz hoặc thấp hơn. Tương tự như vậy thì một bộ vi điều khiển cũng yêu cầu một tinh thể có tần số không lớn hơn 20MHz. Khi 8051 được nối tới một bộ giao động tinh thể thạch anh và cấp nguồn thì ta có thể quan sát tần số trên chân XTAL2 bằng máy hiện sóng. Nếu ta quyết định sử dụng một nguồn tần số khác bộ giao động thạch anh chẳng hạn như là bộ giao động TTL thì nó sẽ được nối tới chân XTAL1, còn chân XTAL2 thì để hở không nối như hình 4.2b.



Hình 4.2: a) Nối XTAL tới 8051 b) Nối XTAL tới nguồn đồng bộ ngoài.

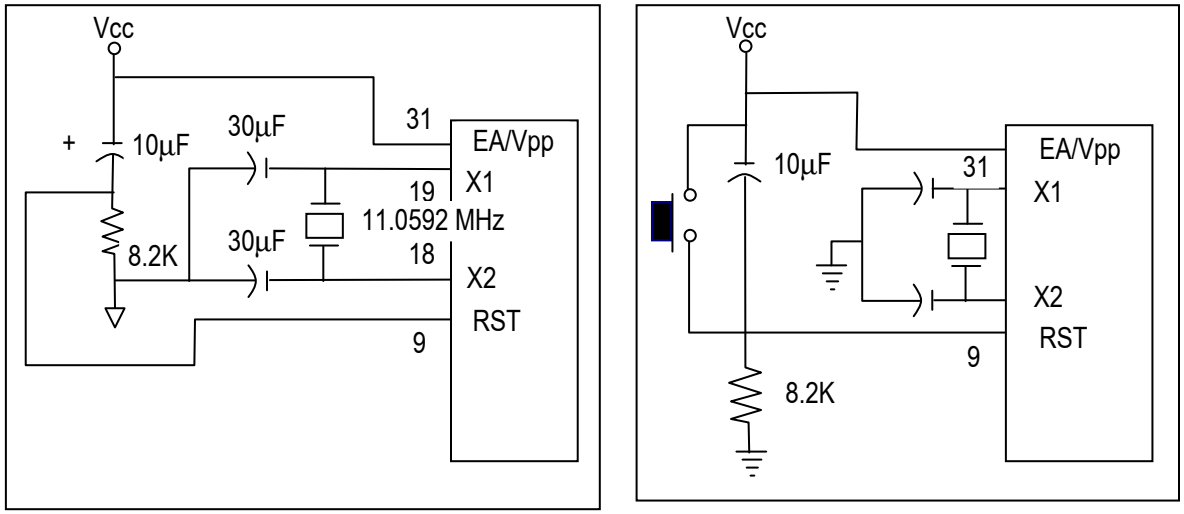
4. Chân RST.

Chân số 9 là chân tái lập RESET. Nó là một đầu vào và có mức tích cực cao (bình thường ở mức thấp). Khi cấp xung cao tới chân này thì bộ vi điều khiển sẽ tái lập và kết thúc mọi hoạt động. Điều này thường được coi như là sự tái bật nguồn. Khi kích hoạt tái bật nguồn sẽ làm mất mọi giá trị trên các thanh ghi. Bảng 4.1 cung cấp một cách liệt kê các thanh ghi của 8051 và các giá trị của chúng sau khi tái bật nguồn.

Bảng 4.1: Giá trị một số thanh ghi sau RESET.

Register	Reset Value
PC	0000
ACC	0000
B	0000
PSW	0000
SP	0000
DPTR	0007
	0000

Lưu ý rằng giá trị của bộ đếm chương trình PC là 0 khi tái lập để ép CPU nạp mã lệnh đầu tiên từ bộ nhớ ROM tại vị trí ngăn nhớ 0000. Điều này có nghĩa là ta phải đặt dòng đầu tiên của mã nguồn tại vị trí ngăn nhớ 0 của ROM vì đây là mã CPU tĩnh thức và tìm lệnh đầu tiên. Hình 4.3 trình bày hai cách nối chân RST với mạch bật nguồn.



Hình 4.3: a) Mạch tái bật nguồn RESET.
b) Mạch tái bật nguồn với Debounce.

Nhằm làm cho đầu vào RESET có hiệu quả thì nó phải có tối thiểu 2 chu kỳ máy. Hay nói cách khác, xung cao phải kéo dài tối thiểu 2 chu kỳ máy trước khi nó xuống thấp.

Trong 8051 một chu kỳ máy được định nghĩa bằng 12 chu kỳ dao động như đã nói ở chương 3 và được trình bày tại vị trí 4.1.

5. Chân $\overline{EA}$:

Các thành viên họ 8051 như 8751, 98C51 hoặc DS5000 đều có ROM trên chíp lưu cất chương trình. Trong các trường hợp như vậy thì chân $\overline{EA}$ được nối tới V_{CC}. Đối với các thành viên củ họ như 8031 và 8032 mà không có ROM trên chíp thì mã chương trình được lưu cất ở trên bộ nhớ ROM ngoài và chúng được nạp cho 8031/32. Do vậy, đối với 8031 thì chân $\overline{EA}$ phải được nối đất để báo rằng mã chương trình được cất ở ngoài. $\overline{EA}$ có nghĩa là truy cập ngoài (External Access) là chân số 31 trên vỏ kiểu DIP. Nó là một chân đầu vào và phải được nối hoặc với V_{CC} hoặc GND. Hay nói cách khác là nó không được để hở.

Ở chương 14 chúng ta sẽ trình bày cách 8031 sử dụng chân này kết hợp với $\overline{\text{PSEN}}$ để truy cập các chương trình được cất trên bộ nhớ ROM ở ngoài 8031. Trong các chíp 8051 với bộ nhớ ROM trên chíp như 8751, 89C51 hoặc DS5000 thì $\overline{\text{EA}}$ được nối với V_{CC}.

Ví dụ 4:

Hãy tìm chu kỳ máy đối với a) XTAL = 11.0592MHz b) XTAL = 16MHz.

Lời giải:

a) 11.0592MHz/12 = 921.6kHz.

Chu kỳ máy = 1/921.6kHz = 1.085μs.

b) 16MHz/12 = 1.333MHz

Chu kỳ máy = 1/1.333MHz = 0.75μs.

Các chân mô tả trên đây phải được nối mà không cần thành viên nào được sử dụng. Còn hai chân dưới đây được sử dụng chủ yếu trong hệ thống dựa trên 8031 và sẽ được trình bày chi tiết ở chương 11.

6. Chân $\overline{\text{PSEN}}$:

Đây là chân đầu ra cho phép cất chương trình (Program Store Enable) trong hệ thống dựa trên 8031 thì chương trình được cất ở bộ nhớ ROM ngoài thì chân này được nối tới chân OE của ROM. Chi tiết được bàn ở chương 14.

7. Chân ALE:

Chân cho phép chốt địa chỉ ALE là chân đầu ra và được tích cực cao. Khi nối 8031 tới bộ nhớ ngoài thì cổng 0 cũng được cấp địa chỉ và dữ liệu. Hay nói cách khác 8031 dồn địa chỉ và dữ liệu qua cổng 0 để tiết kiệm số chân. Chân ALE được sử dụng để phân kênh địa chỉ và dữ liệu bằng cách nối tới chân G của chíp 74LS373. Điều này được nói chi tiết ở chương 14.

8. Các chân cổng vào ra và các chức năng của chúng.

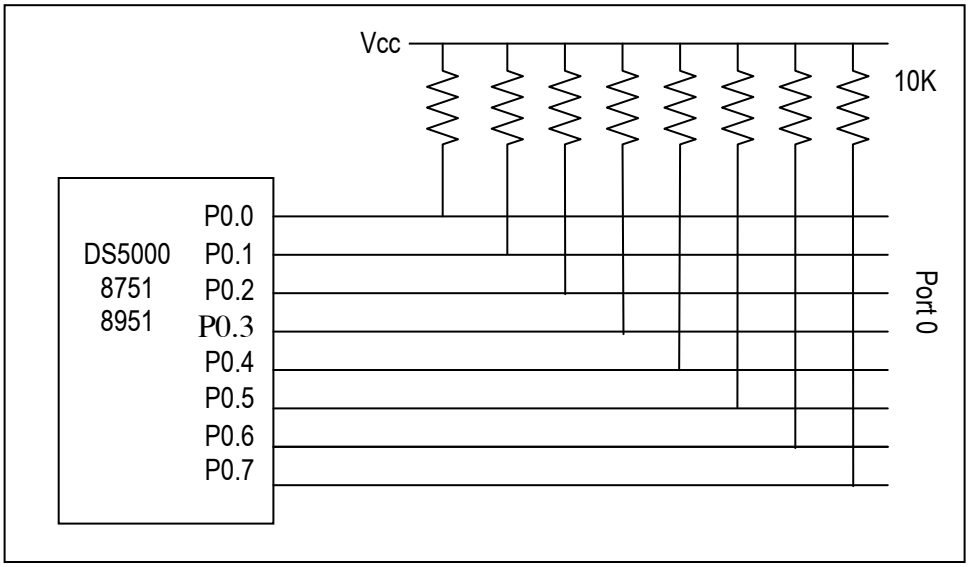
Bốn cổng P0, P1, P2 và P3 đều sử dụng 8 chân và tạo thành cổng 8 bít. Tất cả các cổng khi RESET đều được cấu hình như các đầu ra, sẵn sàng để được sử dụng như các cổng đầu ra. Muốn sử dụng cổng nào trong số các cổng này làm đầu vào thì nó phải được lập trình.

9. Cổng P0.

Cổng 0 chiếm tất cả 8 chân (từ chân 32 đến 39). Nó có thể được dùng như cổng đầu ra, để sử dụng các chân của cổng 0 vừa làm đầu ra, vừa làm đầu vào thì mỗi chân phải được nối tới một điện trở kéo bên ngoài 10kΩ. Điều này là do một thực tế là cổng P0 là một màng mở khác với các cổng P1, P2 và P3. Khái niệm máng mở được sử dụng trong các chíp MOS về chừng mực nào đó nó giống như Cô-lec-tơ hở đối với các chíp TTL. Trong bất kỳ hệ thống nào sử dụng 8751, 89C51 hoặc DS5000 ta thường nối cổng P0 tới các điện trở kéo, Xem hình 4.4 bằng cách này ta có được các ưu điểm của cổng P0 cho cả đầu ra và đầu vào. Với những điện trở kéo ngoài được nối khi tái lập cổng P0 được cấu hình như một cổng đầu ra. Ví dụ, đoạn mã sau đây sẽ liên tục gửi ra cổng P0 các giá trị 554 và AAH.

```
        MOV     A, #554
BACK:   MOV     P0, A
        ACALL   DELAY
```

```
        CPL     A
        SJMP    BACK
```



Hình 4.4: Cổng P0 với các điện trở kéo.

a) Cổng P0 đầu vào: Với các điện trở được nối tới cổng P0 nhằm để tạo nó thành cổng đầu vào thì nó phải được lập trình bằng cách ghi 1 tới tất cả các bit. Đoạn mã dưới đây sẽ cấu hình P0 lúc đầu là đầu vào bằng cách ghi 1 đến nó và sau đó dữ liệu nhận được từ nó được gửi đến P1.

b)

```
        MOV     A,#FFH      ; Gán A = FF dạng Hex
        MOV     P0, A       ; Tạo cổng P0 làm cổng đầu vào bằng cách
                            ; Ghi tất cả các bit của nó.
BACK:   MOV     A, P0       ; Nhận dữ liệu từ P0
        MOV     P1, A       ; Gửi nó đến cổng 1
        SJMP    BACK        ; Lặp lại
```

b) Vai trò kép của cổng P0: Như trình bày trên hình 4.1, cổng P0 được gán AD0 - AD7 cho phép nó được sử dụng vừa cho địa chỉ, vừa cho dữ liệu. Khi nối 8051/31 tới bộ nhớ ngoài thì cổng 0 cung cấp cả địa chỉ và dữ liệu 8051 dồn dữ liệu và địa chỉ qua cổng P0 để tiết kiệm số chân. ALE báo nếu P0 có địa chỉ hay dữ liệu khi ALE - 0 nó cấp dữ liệu D0 - D7. Do vậy, ALE được sử dụng để tách địa chỉ và dữ liệu với sự trợ giúp của chốt 74LS373 mà ta sẽ biết cụ thể ở chương 14.

10. Cổng P1.

Cổng P1 cũng chiếm tất cả 8 chân (từ chân 1 đến chân 8) nó có thể được sử dụng như đầu vào hoặc đầu ra. So với cổng P0 thì cổng này không cần đến điện trở kéo vì nó đã có các điện trở kéo bên trong. Trong quá trình tái lạp thì cổng P1 được cấu hình như một cổng đầu ra. Ví dụ, đoạn mã sau sẽ gửi liên tục các giá trị 55 và AAH ra cổng P1.

```
        MOV     A, #55H
BACK:   MOV     P1, A
```

```
        ACALL   DELAY
        SJMP    BACK
```

Cổng P1 như đầu vào: Để biến cổng P1 thành đầu vào thì nó phải được lập trình bằng cách ghi một đến tất cả các bit của nó. Lý do về điều này được bàn ở mục lục Appendix C.2. Trong đoạn mã sau, cổng P1 lúc đầu đươc cấu hình như cổng đầu vào bằng cách ghi 1 vào các bit của nó và sau đó dữ liệu nhận được từ cổng này được cất vào R7, R6 và R5.

```
        MOV     A, #0FFH    ; Nạp A = FF ở dạng hex
        MOV     P1, A       ; Tạo cổng P1 thành cổng đầu vào bằng
                            ; cách ghi 1 vào các bit của nó.
        MOV     A, P1       ; Nhận dữ liệu từ P1
        MOV     R7, A       ; Cất nó vào thanh ghi R7
        ACALL   DELAY       ; Chờ
        MOV     A,  P1      ; Nhận dữ liệu khác từ P1
        MOV     R6, A       ; Cất nó vào thanh ghi R6
        ACALL   DELAY       ; Chờ
        MOV     A, P1       ; Nhận dữ liệu khác từ cổng P1
        MOV     R5, A       ; Cất nó vào thanh ghi R5
```

11. Cổng P2:

Cổng P2 cũng chiếm 8 chân (các chân từ 21 đến 28). Nó có thể được sử dụng như đầu vào hoặc đầu ra giống như cổng P1, cổng P2 cũng không cần điện trở kéo vì nó đã có các điện trở kéo bên trong. Khi tái lập, thì cổng P2 được cấu hình như một cổng đầu ra. Ví dụ, đoạn mã sau sẽ gửi liên tục ra cổng P2 các giá trị 55H và AAH. Đó là tất cả các bit của P2 lên xuống liên tục.

```
        MOV     A, #55H
BACK:   MOV     P2,A
        ACALL   DELAY
        CPL     A
        SJMP    BACK
```

a) Cổng P2 như đầu vào.

Để tạo cổng P2 như đầu vào thì nó phải được lập trình bằng cách ghi các số 1 tới tất cả các chân của nó. Đoạn mã sau đây đầu tiên cấu hinh P2 là cổng vào bằng cách ghi một đến tất cả các chân của nó và sau đó dữ liệu nhận được từ P2 được gửi liên tục đến P1.

```
        MOV     A, 0FFH     ; Gán A giá trị FF dạng Hex
        MOV     P2, A       ; Tạo P2 là cổng đầu vào bằng cách
                            ; ghi một đến các chân của nó
BACK:   MOV     A, 2        ; Nhận dữ liệu từ P2
        MOV     P1, A       ; Gửi nó đến P1
        SJMP    BACK        ; Lặp lại
```

b) Vai trò kép của P2.

Trong các hệ thống dựa trên 8751, 89C51 và DS5000 thì P2 được dùng như đầu ra đơn giản. Tuy nhiên trong hệ thống dựa trên 80312 thì cổng P2 phải được dùng cùng với P0 để tạo ra địa chỉ 16 brt đối với bộ nhớ ngoài. Như chỉ ra trên hình 4.1 cổng P2 cũng được chỉ định như là A8 - A15 báo chức năng kép của nó. Vì một bộ 8031 có khả năng trung cập 64k byte bộ nhớ ngoài, nó cần một đường địa chỉ 16 bít. Trong khi P.0 cung cấp 8 bit thấp qua A0 - A7. Công việc của P2 là cung cấp các bít địa chỉ A8 - A15. Hay nói cách khác khi 8031được nối tới bộ nhớ ngoài thì P2 được dùng cho 8 bít của địa chỉ 16 bit và nó không thể dùng cho vào ra. Điều này sẽ được trình bày chi tiết ở chương 14.

Từ những trình bày trên đây ta có thể kết luận rằng trong các hệ thống dựa trên các bộ vi điều khiển 8751, 89C51 hoặc DS5000 thì ta có 3 cống P0, P1và P2 cho các thao tác vào ra và như thế là có thể đủ cho các ứng dụng với hầu hết các bộ vi điều khiển. Còn cấp P3 làđể dành cho ngắt và ta sẽ cùng bàn dưới đây.

11 - Cổng P3:

Cổng P3 chiếm tổng cộng là 8 chân từ chân 10 đến chân 17. Nó có thể được sử dụng như đầu vào hoặc đầu ra. Cống P3 không cần các điện trở kéo cũng như P1 và P2. Mặc dù cống P3 được cấu hình như một cống đầu ra khi tái lập, nhưng đây không phải là cách nó được ứng dụng phổ biến nhất. Cống P3 có chức năng bổ xung là cung cấp một số tín hiệu quan trọng đặc biệt chẳng hạn như các ngắt. Bảng 4.2 cung cấp các chức năng khác của cống P3. Thông tin này áp dụng cho cả 8051 và 8031.

Bảng 4.2: Các chức năng khác của cống P3

Bít của cống P3	Chức năng	chân số
P3.0	Nhận dữ liệu (RXD)	10
P3.1	Phát dữ liệu (TXD)	11
P3.2	Ngắt 0(INT0)	12
P3.3	Ngắt 1(INT1)	13
P3.4	Bộ định thời 0 (TO)	14
P3.5	1 Bộ định thời 1(T1)	15
P3.6	Ghi (WR)	16
P3.7	Đọc (RD)	17

Các bit P3.0 và P3.1 được dùng cho các tín hiệu nhận và phát dữ liệu trong truyền thông dữ liệu nối tiếp. Xem chương 10 đết biết các chúng được nối ghép như thế nào. Các bit P3.2 và P3.3 được dành cho các ngắt ngoài và chúng được trình bày chi tiết ở chương 11. Bit P3.4 và P3.5 được dùng cho các bộ định thêm 0 và 1và chi tiết được trình bày ở chương 9. Cuối cùng các bit P3.6 và P3.7 được cấp cho các tín hiệu ghi và đọc các bộ nhớ ngoài được nối tới các hệ thống dựa trên 8031. Chương 14 sẽ trình bày cách chúng được sử dụng như thế nào trong các hệ thống dựa trên 8031. Trong các hệ thống dựa trên 8751, 89C51 hoặc D35000 thì các chân P3.6 và P3.7 được dùng cho vào - ra còn các chân khác của P3 được sử dụng bình thường trong vai trò chức năng thay đổi.

4.2 Lập trình vào - ra: thao tác bit.

4.2.1 các cách khác nhau để truy cập toàn bộ 8 bit.

Trong đoạn mà dưới đây cũng như trong nhiều ví dụ vào ra trước đây toàn bộ 8 bit của cổng P1 được cập.

```
BACK:   MOV     A, # 55H
        MOV     P1,A
        ACALL   DELAY
        MOV     A, #0AAH
        MOV     P1, A
        ACALL   DELAY
        SJMP    BACK
```

Đoạn mà trên chốt mỗi bit của P1 một cách liên tục. Chúng ta đã thắng một biến thế của chương trình trên trước đó. Bây giờ ta có thể viết lại đoạn mã trên theo cách hiệu quả hơn bằng cách truy cập trực tiếp cổng mà không qua thanh ghi tổng như sau:

```
BACK:   MOV     P1, # 55H
        ACALL   DELAY
        MOV     P1, #00H
        CALL    DELAY
        SJMP    BACK
```

Ta có thể viết một dạng khác của đoạn mà trên bằng kỹ thuật đọc - sửa đổi ghi như ở mục 4.2.2 dưới đây.

4.2.2 Đặc điểm Đọc- sửa đổi - ghi (Read - Modify – Write).

Các cổng trong 8051 có thể được truy cập bằngkỹ thuật được gọi là Đọc-sửa đổi-ghi. Đặc điểm này tiết kiệm rất nhiều dòng lệnh bằng cách kết hợp tất cả 3 thao tác: 1đọc cổng, 2 sửa đổi nóvà 3 ghi nó ra cổng vào một lệnh đơn. Đoạn mà dưới đây trước hết đặt 01010101(nhị phân) vào cổng 1. Sau đólệnh "XLR P1, #0FFH" thực hiện phép lô-gích OR loại trừ là XOR trên cổng p1 với 1111 1111 (nhị phân) và sau đó ghi kết quả trở lại cổng P1.

```
        MOV     P1, #55H        ; P1 = 01010101
AGAIN:  XLR     P1,# 0FFH       ; EX - 0R P1 với 1111 1111
        ACALL   DELAY
        SJMP    AGAIN
```

Lưu ý rằng lệnh X0R của 55H và FFH sẽ cho kết quả là AAH. Tương tự như vậy lệnh X0R của AAH với FFH lại cho giá trị kết quả là 55H. Các lệnh lô-gích được trình bày ở chương 7.

4.2.3. Khả năng đánh địa chỉ theo bít của các cổng

Có nhiều lúc chúng ta cần truy cập chỉ 1 hoặc2 bít của cổng thay vì truy cập cả 8 bit của cổng. Một điểm mạnh của các cổng 8051 là chúng có khả năng truy cập từng bít riêng rẽ mà không làm thay đổi các bít còn lại trong cổng đó ví dụ, đoạn mà dưới đây chốt bit P1.2 liên tục:

```
BACK:   CPL     P1.2        ; Lấy bù 2 chỉ riêng bit P1.2
```

```
        ACALL   DELAY
        SJMP    BACK
```

Một biến thể khác của đoạn mã trên là:

```
AGACN:  SETB    P1.2    ; Chỉ thay đổi bít P1.2 lên cao
        ACALL   DELAY
        CLR     P1.2    ; Xoá bít P1.2 xuống thấp
        ACALL   DELAY
        SJMP    AGAIN
```

Lưu ý rằng bít P1.2 là bít thứ 3 của cổng P1, vì bít thứ nhất là P1.0 và bit thứ hai là P1.1 v.v...

Bảng 4.3 trình bày các bít của các cổng vào ra của 8051. Xem ví dụ 4.2 về thao tác bít của các bít vào - ra. Lưu ý rằng trong ví dụ 4.2 các bit không dùng đến là không bị ảnh hưởng. Đây là khả năng đánh địa chỉ theo bit của các cổng vào - ra và là một trong những điểm mạnh nhất của bộ vi điều khiển 8051.

Ví dụ 4.2: hãy viết chương trình thực hiện các công việc sau:

a) Duy trì hiển thị bit P1.2 cho đến khi nó lên cáp
b) Khi P1.2 lên cao, hãy ghi giá trị 45H vào cổng P0
c) Gửi một xung cao xuống thấp (H-to-L) tới P2.3

Lời giải:

```
        SET     P1.2        ; Tạo bit P1.2 là đầu vào
        MOV     A, #45H     ; Gán A = 45H
AGAIN:  JNB     P1.2, AGAIN ; Thoát khi P1.2 = 1
        MOV     P0, A       ; Xuất A tới cổng P0
        SETB    P2.3        ; Đưa P2.3 lên cao
        CLR     P2.3        ; Tạo P2.3 xuống thấp để có xung H-T0-L
```

Trong chương trình này lệnh "JNB P1.2, AGCN" (JNB có nghĩa là nhảy nếu không bit) ở lại vòng lặp cho đến khi P1.2 chưa lên cao. Khi P1.2 lên cao nó thoát ra khỏi vòng lặp ghi giá trị 45H tới cổng P0 và tạo ra xung H-to-L bằng chuỗi các lệnh SETB và CLR.

CHƯƠNG 5
Các chế độ đánh địa chỉ của 8051

CPC có thể truy cập dữ liệu theo nhiều cách khác nhau. Dữ liệu có thể ở trong một thanh ghi hoặc trong bộ nhớ hoặc được cho như một giá trị tức thời các cách truy cập dữ liệu khác nhau được gọi là các chế độ đánh địa chỉ. Chương này chúng ta bàn luận về các chế độ đánh địa chỉ của 8051 trong phạm vi một số ví dụ.

Các chế độ đánh địa chỉ khác nhau của bộ vi xử lý được xác định như nó được thiết kế và do vậy người lập trình không thể đánh địa chỉ khác nhau là:

1. tức thời 2. Theo thanh ghi 3. Trực tiếp
4. gián tiếp qua thanh ghi 5. Theo chỉ số

5.1 Các chế độ đánh địa chỉ tức thời và theo thanh ghi

5.1.1 Chế độ đánh 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   B, #40H           ; Nạp giá trị 40 H vào thanh ghi B
MOV   DPTR, #4521H      ; Nạp 4512H vào con trỏ dữ liệu DPTR
```

Mặc dù thanh ghi DPTR là 16 bit nó cũng có thể được truy cập như 2 thanh ghi 8 bit DPH và DPL trong đó DPH là byte cao và DPL là byte thấp. Xét đoạn mã dưới đây:

```
MOV         DPTR, #2550H
MOV         A, #50H
MOV         DPH, #25H
```

Cũng lưu ý rằng lệnh dưới đây có thể tạo ra lỗi vì giá trị nạp vào DPTR lớn hơn16 bit:

```
MOV         DPTR, # 68975       ; Giá tri không hợp lệ  > 65535 (FFFFH)
```

Ta có thể dùng chỉ lệnh Eqw để truy cập dữ liệu tức thời như sau

```
            COUNT       EDU     30
            ...         ...
            MOV         R4, #COUNT              ; R4 = 1E (30 = 1EH)
            MOV         DPTR, #MYDATA           ; DPTR = 200H

            ORG     200H
MYDATA:     DB      “America”
```

Lưu ý rằng ta cũng có thể sử dụng chế độ đánh được chỉ tức thời để gửi dữ liệu đến các cổng của 8051.
Ví dụ "MOV P1, #55H" là một lệnh hợp lệ.

5.1.2 chế độ đánh địa chỉ theo thanh ghi:

Chế độ đánh địa chỉ theo thanh ghi liên quan đến việc sử dụng các thanh ghi để dữ liệu cần được thao tác các ví dụ về đánh địa chỉ theo thanh ghi như sau:

```
MOV    A,  RO          ; Sao nội dung thanh ghi RO vào thanh ghi A
MOV    R2, A           ; Sao nội  dung thanh ghi  A vào thanh ghi R2
ADD    A; R5           ; Cộng nội dung  thanh ghi R5 vào thanh ghi A
ADD    A, R7           ; Cộng nội dung thanh ghi  R7 vào thanh ghi A
MOV    R6, A           ; Lưu nội dung thanh ghi A vào thanh ghi R6
```

Cũng nên lưu ý rằng các thanh ghi nguồn và đích phải phù hợp về kích thước. Hay nói cách khác, nếu viết " MOV DPTR, A" sẽ cho một lỗi vì nguồn là thanh ghi 8 bit và đích lại là thanh ghi 16 bit. Xét đoạn mã sau:

```
MOV    DPTR, #25F5H
MOV    R7,  DPL
MOV    R6,  DPH
```

Để ý rằng ta có thể chuyển dữ liệu giữa thanh ghi tích luỹ A và thanh ghi Rn (n từ 0 đến 7) nhưng việc chuyển dữ liệu giữa các thanh ghi Rn thì không được phép. Ví dụ, lệnh "MOV R4, R7" là không hợp lệ.

Trong hai chế độ đánh địa chỉ đầu tiên, các toán hạng có thể hoặc ở bên trong một trong các thanh ghi hoặc được gắn liền với lệnh. Trong hầu hết các chương trình dữ liệu cần được xử lý thường ở trong một số ngăn của bộ nhớ RAM hoặc trong không gian mà của ROM. Có rất nhiều cách để truy cập dữ liệu này mà phần tiếp theo sẽ xét đến.

5.2 Truy cập bộ nhớ sử dụng các chế độ đánh địa chỉ khác nhau.

5.2.1 Chế độ đánh địa chỉ trực tiếp.

Như đã nói ở chương 2 trong 8051 có 128 byte bộ nhớ RAM. 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.

Mặc dù toàn bộ byte của bộ nhớ RAM có thể được truy cập bằng chế độ đánh địa chỉ trực tiếp, nhưng chế độ này thường được sử dụng nhất để truy cập các ngăn nhớ RAM từ 30H đến 7FH. Đây là do một thực tế là các ngăn nhớ dành cho băng ghi được truy cập bằng thanh ghi theo các tên gọi của chúng là R0 - R7 còn các ngăn nhớ khác của RAM thì không có tên như vậy. Trong chế độ đánh địa chỉ trực tiếp thì dữ liệu ở trong một ngăn nhớ RAM mà địa chỉ của nó được biết và địa chỉ này được cho như là một phần của lệnh. Khác với chế độ đánh địa chỉ tức thì mà toán hạng tự nó được cấp với lệnh. Dấu (# 0 là sự phân biệt giữa hai chế độ đánh địa chỉ. Xét các ví dụ dưới đây và lưu ý rằng các lệnh không có dấu (#):

```
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
MOV    R4, 7FH         ; Chuyển nôi dung ngănnhớ 7FH của RAM  vào R4
```

Như đã nói ở trước thì các ngăn nhớ trừ 0 đến 7 của RAM được cấp cho bằng 0 của các thanh ghi R0 - R7. Các thanh ghi này có thể được truy cập theo 2 cách như sau:

```
MOV    A, 4            ; Hai lệnh này giống nhau đều sao nội dung thanh ghi  R4 vào A
MOV    A, R4

MOV    A, 7            ; Hai lệnh này đều như nhau là sao nội dung R7 vào thanh ghi A
MOV    A,R7
```

Để nhấn mạnh sự quan trọng của dấu (#) trong các lệnh của 8051. Xét các mã cho sau đây:

```
MOV    R2, #05         ; Gán R2=05
MOV    A, 2            ; Sao nội dung thanh ghi R2 vào A
MOV    B, 2            ; Sao nội dung thanh ghi R2 vào  B
MOC    7,2             ; Sao nội dung thanh ghi R7 vì lệnh "MOV R7, R2" là không hợp  lệ.
```

Mặc dù sử dụng các tên R0 - R7 dễ hơn các địa chỉ bộ nhớ của chúng nhưng các ngăn nhớ 30H đến 7FH của RAM không thể được truy cập theo bất kỳ cách nào khác là theo địa chỉ của chúng vì chúng không có tên.

5.2.2 các thanh ghi SFSR và các địa chỉ của chúng.

Trong các thanh ghi được nói đến từ trước đến giờ ta thấy rằng các thanh ghi R0 - R7 là một phần trong 128 byte của bộ nhớ RAM. Vậy còn các thanh ghi A, B, PSW và DPTR là một bộ phận của nhóm các thanh ghi nhìn chung được gọi là các thanh ghi đặc biệt SFR (Special Funtion Register). Có rất nhiều thanh ghi với chức năng đặc biệt và chúng được sử dụng rất rộng rãi mà ta sẽ trình bày ở các chương sáu. Các thanh ghi FR có thể được truy cập theo tên của chúng (mà dễ hơn rất nhiều) hoặc theo các địa chỉ của chúng. Ví dụ địa chỉ của thanh ghi A là EOH và thanh ghi B là FOH như cho ở trong bảng 5.1. Hãy để ý đến những cặp lệnh có cùng ý nghĩa dưới đây:

```
MOV    0E0H, #55H      ; Nạp 55H vào thanh ghi A(A=55H)
MOV    A,   #55H       ;

MOV    0F0H, #25H      ; Nạp 2SH vào thanh ghi  B ( B = 25)
MOV    3, #25H         ;

MOV    0E0H            ; Sao nội dung thanh ghi R2 vào A
MOV    A, R2           ;

MOV    0F0             ; Sao nội dung thanh ghi R0 vào B
MOV    B, R0           ;
```

Bảng 5.1 dưới đây liệt kê các thanh ghi chức năng đặc biệt SFR của 8051 và các địa chỉ của chúng. Cần phải lưu ý đến hai điểm sau về các địa chỉ của SFR:

1. Các thanh ghi SFR có địa chỉ nàm giữa 80H và FFH các địa chỉ này ở trên 80H, vì các địa chỉ từ 00 đến 7FH là địa chỉ của bộ nhớ RAM bên trong 8051.

2. không phải tất cả mọi địa chỉ từ 80H đến FFH đều do SFH sử dụng, nhưng vị trí ngăn nhớ từ 80H đến FFH chưa dùnglà để dữ trữ và lập trình viên 8051 cũng không được sử dụng.

Bảng 5.1: Các địa chỉ của thanh ghi chức năng đặc biệt SFR

Lệnh	Tên	Địa chỉ
ACC*	Thanh ghi tích luỹ (thanh ghi tổng) A	0E0H
B*	Thanh ghi B	0F0H
PSW*	Từ trạng thái chương trình	0D0H
SP	Con trỏ ngăn xếp	81H
DPTR	Con trỏ dữ liệu hai byte	
DPL	Byte thấp của DPTR	82H
DPH	Byte cao của DPTR	83H
P0*	Cổng 0	80H
P1*	Cổng 1	90H
P2*	Cổng 2	0A0H
P3*	Cổng 3	0B0H
IP*	Điều khiển ưu tiên ngắt	0B8H
IE*	Điều khiển cho phép ngắt	A08H
TMOD	Điều khiển chế độ bộ đếm/ Bộ định thời	89H
TCON*	Điều khiển bộ đếm/ Bộ định thời	88H
T2CON*	Điều khiển bộ đếm/ Bộ định thời 2	0C8H
T2MOD	Điều khiển chế độ bộ đếm/ Bộ định thời 2	0C9H
TH0	Byte cao của bộ đếm/ Bộ định thời 0	8CH
TL0	Byte thấp của bộ đếm/ Bộ định thời 0	8AH
TH1	Byte cao của bộ đếm/ Bộ định thời 1	8DH
TL1	Byte thấp của bộ đếm/ Bộ định thời 1	8BH
TH2	Byte cao của bộ đếm/ Bộ định thời 2	0CDH
TL2	Byte thấp của bộ đếm/ Bộ định thời 2	0CCH
RCAP2H	Byte cao của thanh ghi bộ đếm/ Bộ định thời 2	0CBH
RCAP2L	Byte thấp của thanh ghi bộ đếm/ Bộ định thời 2	0CAH
SCON*	Điều khiển nối tiếp	98H
SBUF	Bộ đệm dữ liệu nối tiếp	99H
PCON	Điều khiển công suất	87H

*Các thanh ghi có thể đánh địa chỉ theo bit.

Xét theo chế độ đánh địa chỉ trực tiếp thì cần phải lưu ý rằng giá trị địa chỉ được giới hạn đến 1byte, 00 - FFH. Điều này có nghĩa là việc sử dụng của chế độ đánh địa chỉ này bị giới hạn bởi việc truy cập các vị trí ngăn nhớ của RAM và các thanh ghi với địa chỉ được cho bên trong 8051.

Ví dụ 5.1:

Viết chương trình để gửi 55H đến cổng P1 và P2 sử dụng hoặc

a) Tên các cổng

b) Hoặc địa chỉ các cổng

Lời giải:

```
a)      MOV     A, #55H ; A = 55H
```

```
MOV    P1, A          ; P1 = 55H
MOV    P2, A          ; P2 = 55H
```

b) Từ bảng 5.1 ta lấy đại chỉ cổng P1 là 80H và P2 là A0H

```
MOV    A, #55H        ; A = 55H
MOV    80H, A         ; P1 = 55H
MOV    0A0H, A        ; P2 = 55H
```

5.2.3 Ngăn xếp và chế độ đánh địa chỉ trực tiếp.

Một công dụng chính khác của chế độ đánh địa chỉ trực tiếp là ngăn xếp. Trong họ 8051 chỉ có chế độ đánh địa chỉ trực tiếp là được phép đẩy vào ngăn xếp. Do vậy, một lệnh như "PVSH A" là không hợp lệ. Việc đẩy thanh ghi A vào ngăn xếp phải được viết dưới dạng "PVAH 0E0H" với 0E0H là địa chỉ của thanh ghi A. Tương tự như vậy để đẩy thanh ghi R3 rãnh 0 vào ngăn xếp ta phải viết là "PVSH 03". Chế độ đánh địa chỉ trực tiếp phải được sử dụng cho cả lệnh POP. Vì dụ "POP 04" sẽ kéo đỉnh của ngăn xếp vào thanh ghi R4 rãnh 0.

Ví dụ 5.2:

Trình bày mã để đẩy thanh ghi R5, R6 và A vào ngăn xếp và sau đó kéo chùng ngược trở lại R2, R3 và B tương ứng.

Lời giải:

```
PUSH    05      ; Đẩy R5 vào ngăn xếp
PUSH    06      ; Đẩy R6 vào ngăn xếp
PUSH    0E0H    ; Đẩy thanhghi A vào ngăn xếp
POP     0F0H    ; Kéo đỉnh ngăn xếp cho vào thanh ghi B
                ; Bây giờ B = A
POP     02      ; Kéo đỉnh ngăn xếp cho vào thanh ghi R2
                ; Bây giờ R2= R6
POP     03      ; Kéo đỉnh ngăn xếp cho vào thanh ghi
                ; Bây giờ R3 = R5
```

5.2.4 chế độ đánh địa chỉ gián tiếp thanh ghi.

Trong chế độ này, một thanh ghi được sử dụng như một con trỏ đến dữ liệu. Nếu dữ liệu ở bên trong CPU thì chỉ các thanh ghi R0 và R1 được sử dụng cho mục đích này. Hay nói cách khác các thanh ghi R2 - R7 không có thể dùng được để giữ địa chỉ của toán hạng nằm trong RAM khi sử dụng chế độ đánh địa chỉ này khi Ro và R1 được dùng như các con trỏ, nghĩa là khi chúng giữ các địa chỉ của các ngăn nhớ RAM thì trước chúng phải đặt dấu (@) như chỉ ra dưới đây.

```
MOV    A, @ R0        ; Chuyển nội dung của ngăn nhớ RAM có địa chỉ trong RO và A
MOV    @ R1, B        ; Chuyển nội dung của B vào ngăn nhớ RAM có địa chỉ ở R1
```

Lưu ý rằng R0 cũng như R1 luôn có dấu "@" đứng trước. Khi không có dấu này thì đó là lệnh chuyển nội dung các thanh ghi Ro và R1 chứ không phải dữ liệu ngăn nhớ mà địa chỉ có trong R0 và R1.

Ví dụ 5.3:

Viết chương trình để sao chép giá trị 55H vào ngăn nhớ RAM tại địa chỉ 40H đến 44H sử dụng:

a) Chế độ đánh địa chỉ trực tiếp

b) Chế độ đánh địa chỉ gián tiếp thanh ghi không dùng vòng lặp

c) Chế độ b có dùng vòng lặp

Lời giải:

```
        MOV   A, #55H       ; Nạp A giá trị  55H
        MOV   40H, A        ; Sao chép A vào ngăn nhớ RAM  40H
        MOV   41H, A        ; Sao chép A vào ngăn nhớ RAM  41H
        MOV   42H, A        ; Sao chép A vào ngăn nhớ RAM  42H
        MOV   43H, A        ; Sao chép A vào ngăn nhớ RAM  43H
        MOV   44H, A        ; Sao chép A vào ngăn nhớ RAM  44H
b)
        MOV   A, # 55H      ; Nạp vào A giá trị 55H
        MOV   R0, #40H      ; Nạp con trỏ  R0 = 40 H
        MOV   @R0, A        ; Sao chép A vào vị trí ngăn nhớ RAM do R0 chỉ đến
        INC   R0            ; Tăng con trỏ. Bây gì R0 = 41H
        MOV   @R0, A        ; Sao chép A vào vị trí ngăn nhớ RAM do R0 chỉ
        INC   R0            ; Tăng con trỏ. Bây giờ R0 = 42H
        MOV   @R0,A         ; Sao chép Avào vị trí ngăn nhớ RAM do R0 chỉ
        INC   R0            ; Tăng con trỏ. Bây giờ R0 = 43H
        MOV   @R0, A        ; Sao chép A vào vị trí ngăn nhớ RAM do R0 chỉ
        MOV   @R0, A        ;Tăng con trỏ. Bây gờ R0 = 44H
        MOV   @R0, A
c)
        MOV   A, # 55H      ; Nạp vào A giá trị 55H
        MOV   R0, #40H      ; Nạp con trỏ địa chỉ ngăn nhớ RAM R0 = 40H
        MOV   R2, #05       ; Nạp bộ đếm R2 = 5
AGAIN:  MOV   @R0, A        ; Sao chép A vào vị trí ngăn nhớ RAM do Ro chi đến
        INC                 ; Tăng con trỏ Ro
        DJNZ  R2, AGAIN     ; Lặp lại cho đến khi bộ đếm = 0.
```

5.2.5 Ưu điểm của chế độ đánh địa chỉ gián tiếp thanh ghi.

Một trong những ưu điểm của chế độ đánh địa chỉ gián tiếp thanh ghi là nó làm cho việc truy cập dữ liệu năng động hơn so với chế độ đánh địa chỉ trực tiếp.

Ví dụ 5.3 trình bày trường hợp sao chép giá trị 55H vào các vị trí ngăn nhớ của RAM từ 40H đến 44H .

Lưu ý rằng lời giải b) có hai lệnh được lặp lại với một số lần. Ta có thể tạo ra vòng lặp với hai lệnh này như ở lời giải c). Lời giải c) là hiệu quả nhất và chỉ có thể khi sử dụng chế độ đánh địa chỉ gián tiếp qua thanh ghi. Vòng lặp là không thể trong chế độ đánh địa chỉ trực tiếp. Đây là sự khác nhau chủ yếu giữa đánh địa chỉ trực tiếp và gián tiếp.

Ví dụ 5.4:

Hãy viết chương trình để xoá 16 vị trí ngăn nhớ RAM bắt đầu tại địa chỉ 60H.

Lời giải:

```
        CLR    A            ; Xoá A=0
        MOV    R1, #60H     ; Nạp con trỏ. R1= 60H
        MOV    R7, #16H     ;Nạp bộ đếm, R7 = 1 6  (10 H dạng hex)
```

```
AGAIN:    MOV   @R1, A      ; Xoá vị trí ngăn nhớ RAM do R1 chỉ đến
          INC   R1          ; Tăng R1
          DJNZ  R7, AGAiN   ; Lặp lại cho đến khi bộ đếm = 0
```

Một ví dụ về cách sử dụng cả R0 và R1 trong chế độ đánh địa chỉ gián tiếp thanh ghi khi truyền khối được cho trong ví dụ 5.5.

Ví dụ 5.5:

Hãy viết chương trình để sao chép một khối 10 byte dữ liệu từ vị trí ngăn nhớ RAM bắt đầu từ 35H vào các vị trí ngăn nhớ RAM bắt đầu từ 60H

Lời giải:

```
          MOV   R0, # 35H      ; Con trỏ nguồn
          MOV   R1, #60H       ; Con trỏ đích
          MOV   R3, #10        ; Bộ đếm
BACK:     MOV   A, @R0         ; Lấy 1byte từ nguồn
          MOV   @R1, A         ; Sao chép nó đến đích
          INC   R0             ; Tăng con trỏ nguồn
          INC   R1             ; Tăng con trỏ đích
          DJNZ  R3, BACK       ; Lặp lại cho đến khi sao chép hết 10 byte
```

5.2.6 Hạn chế của chế độ đánh địa chỉ gián tiếp thanh ghi trong 8051.

Như đã nói ở phần trước rằng R0 và R1 là các thanh ghi duy nhất có thể được dùng để làm các con trỏ trong chế độ đánh địa chỉ gián tiếp thanh ghi. Vì R0 và R1 là các thanh ghi 8 bit, nên việc sử dụng của chúng bị hạn chế ở việc truy cập mọi thông tin trong các ngăn nhớ RAM bên trong (các ngăn nhớ từ 30H đến 7FH và các thanh ghi SFR). Tuy nhiên, nhiều khi ta cần truy cập dữ liệu được cất trong RAM ngoài hoặc trong không gian mã lệnh của ROM trên chip. Hoặc là truy cập bộ nhớ RAM ngoài hoặc ROM trên chíp thì ta cần sử dụng thanh ghi 16 bit đó là DPTR.

5.2.7 Chế độ đánh địa chỉ theo chỉ số và truy cập bộ nhớ ROM trên chíp.

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 chương trình của 8051. Lệnh được dùng cho mục đích này là "Move A, @ A + DPTR". Thanh ghi 16 bit DPTR là thanh ghi A được dùng để tạo ra địa chỉ của phân tử dữ liệu được lưu cất trong ROM trên chíp. Do các phân tử dữ liệu được cất trong không gian mã (chương trình) của ROM trên chip của 8051, nó phải dùng lệnh Move thay cho lệnh Mov (chủ C ở cuối lệnh là chỉ mà lệnh Code). 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. Xét ví dụ 5.6.

Ví dụ 5.6:

Giả sử từ "VSA" được lưu trong ROM có đĩa chỉ bắt đầu từ 200H và chương trình được ghi vào ROM bắt đầu từ địa chỉ 0. Hãy phân tích cách chương trình hoạt động và hãy phát biểu xem từ "VSA" sau chương trình này được cất vào đâu?

Lời giải:

```
          ORG   0000H          ; Bắt đầu đốt ROM tại địa chỉ 00H
          MOV   DPTR, #200H    ; Địa chỉ bảng trình bày DPTR = 200H
          CLA   A              ; Xoá thanh ghi A (A = 0)
          MOVC  A, @A + DPTR   ; Lấy ký tự từ không gian nhớ chương trình
          MOV   R0, A          ; Cất nó vào trong R0
```

```
                INC         DPTR                ; DPTR = 201, chỉ đến ký tự kế tiếp
                CLR         A                   ; Xoá thanh ghi A
                MOVC        A, @A + DPTR        ; Lấy ký tự kế tiếp
                MOV         R1, A               ; Cất nó vào trong R1
                INC         DPTR                ; DPTR = 202  con trỏ chỉ đến ký tự sau đó
                CLA         A                   ; Xoá thanh ghi A
                MOVC        A, @A + DPTR        ; Nhận ký tự kế tiếp
                MOV         R2, A               ; Cất nó vào R2
HERE:           SJMP        HERE                ; Dừng lại ở đây.
; Dữ liệu được đốt trong không gian mã lệnh  tại địa chỉ  200H
                            ORG  200H
MYDATA:                     DB "VSA"
                            END                 ; Kết thúc chương trình
```

Ở trong chương trình nói trên thì các vị trí ngăn nhớ ROM chương trình 200H - 2002H có các nội dung sau:
200 = ('U'); 201= ('S') và 202 = ('A').

Chúng ta bắt đầu với DPTR = 200H và A = 0.Lệnh "MOVC A, @ A + DPTR chuyển nội dung của vị trí nhớ 200H trong ROM (200H + 0 = 200H) vào A.

Thanh ghi A chứa giá trị 55H là giá trị mà ASC của ký tự "U". ký tự này được cất vào R0. Kế đó, DPTR được tăng lên tạo thành DPTR = 201H. A lại được xoá về 0 để lấy nội dung của vị trí nhớ kế tiếp trong ROM là 201H chưa ký tự "S". Sau khi chương trình này chạy ta có R0 = 55H, R1 = 53H và R2 = 41H là các mã ASCII của các ký tự "U", "S" và "A".

Ví dụ 5.7:

Giả sử không gian ROM bắt đầu từ địa chỉ 250H có chứa "America", hãy viết chương trình để truyền các byte vào các vị trí ngăn nhớ RAM bắt đầu từ địa chỉ 40H.

Lời giải

```
; (a)  Phương pháp này sử dụng một bộ đếm
                ORG         000
                MOV         DPTR, # MYDATA          ; Nạp con trỏ ROM
                MOV         R0, #40H                ; Nạp con trỏ RAM
                MOV         R2, #7                  ; Nạp bộ đếm
BACK:  CLR              A                           ; Xoá thanh ghi A
                MOVC        A, @A + DPTR            ;Chuyển dữ liệu từ khong gian mã
                MOV         R0, A                   ;Cất nó vào ngăn nhớ RAM
                INC         DPTR                    ; Tăng con trỏ ROM
                INC         R0                      ; Tăng con trỏ  RAM
                DJNZ        R2, BACK                ; Lặp lại cho đếnkhi bộ đếm = 0
HERE:  SJMP             HERE
;--------------- -- không gian mã của ROM trên chíp  dùng để cất dữ liệu ORG  250H

MYDATA:         DB              "AMER1CA"
                END

;(b) phương pháp này sử dụng ký tự null để kết thúc chuỗi

                ORG         000
                MOV         DPTR, #MYDATA           ; Nạp  con trỏ ROM
                MOV         R0, #40                 ; Nạp con trỏ   RAM
BACK:           CLR         A       S               ; Xoá thanh ghi A(A=0)
```

```
            MOVC    A, @A + DPTR            ; Chuyển dữ liệu từ không gian mã
            JZ      HERE                    ; Thoát ra nếu có ký tự Null
            MOV     DPTR, #MYDATA           ; Cất nó vào ngăndn nhớ của RAM
            INC     @R0, A                  ; Tăng con trỏ ROM
            INC     R0                      ; Tăng con trỏ RAM
            SJM     BACK                    ; Lặp lại
HERE:       SJMP    HERE
;------------------ không gian mã của ROM trên chíp dùng để cất dữ liệu ORG 250H
MYADTA:     DB “AMER1CA”, 0                 ; Ký tự Null để kết thúc chuỗi  END
```

Lưu ý đến cách ta sử dụng lệnh JZ để phát hiện ký tự NOLL khi kết thúc chuỗi

5.2.8 Bảng xắp xếp và sử dụng chế độ đánh địa chỉ theo chỉ số.

Bảng xắp xế là khái niệm được sử dụng rất rộng rãi trong lập trình các bộ vi xử lý. Nó cho phép truy cập các phần từ của một bảng thường xuyên được sử dụng với thao tác cực tiểu. Như một ví dụ, hãy giả thiết rằng đối với một ứng dụng nhất định ta cần x^2 giá trị trong phạm vi 0 đến 9. Ta có thể sử dụng một bảng xắp xếp thay cho việc tính toán nó. Điều này được chỉ ra trong ví dụ 5.8.

Ví dụ 5.8

Hãy viết một chương trình để lấy x giá trị cống P1 và gửi giá trị x^2 tới cổng P2 liên tục.

Lời giải:

```
            ORG     000
            MOV     DPTR, #300 H        ; Nạp địa chỉ bảng xắp xêlps
            MOV     A, #0FFH            ; Nạp A giá trị FFH
            MOV     P1, A               ; Đặt cổng P1 là đầu vào
BACK:       MOV     A, P1               ; Lấy giá trị X từ P1
            MOVC    A, @A + DPTR        ; Lấy giá trị  X từ bảng  XSDQ-TABLE
            MOV     P2, A               ; Xuất nó ra cổng P2
            SJMP    BACK                ; Lặp lại

            ORG     300H
XSQR - TABLE:
            DB      0, 1,  4, 9, 16, 25, 36, 49, 64, 81
            END
```

Lưu ý bảng lệnh đầu tiên có thể thay bằng “MOV DPTR, #XSQR - TABLE”.

Ví dụ 5.9:

Trả lời các câu hỏi sau cho ví dụ 5.8.

a) Hãy chỉ ra nội dung các vị trí 300 - 309H của ROM

b) Tại vị trí nào của ROM có giá trị 6 và giá trị bào nhiêu

c) Giả sử P1 có giá trị là 9 thì giá trị P2 là bao nhiêu (ở dạng nhị phân)?

Lời giải:

a) Các giá trị trong các ngăn nhớ 300H - 309H của ROM là:

300 = (00) 301 = (01) 302 = (04) 303 = (09)

304 = (10) $4 \times 4 = 16 = 10$ in hex

305 = (19) $5 \times 5 = 25 = 19$ in hex

306 = (24) $6 \times 6 = 36 = 24$H

307 = (31) 308 = (40) 309 = (51)

b) vị trí chứa giá trị 306H và giá trị là 24H
c) 01010001B là giá trị nhị phân của 51H và 81 ($9^2 = 81$)

Ngoài việc sử dụng DPTR để truy cập không gian bộ nhớ ROM chương trình thì nó còn có thể được sử dụng để truy cập bộ nhớ ngoài nối với 8051 (chương 14).

Một thanh ghi khác nữa được dùng trong chế độ đánh địa chỉ theo chỉ số là bộ đếm chương trình (AppendixA).

Trong nhiều ví dụ trên đây thì lệnh MOV đã được sử dụng để đảm bảo đính rõ ràng, mặc dù ta có thể sử dụng bất kỳ lệnh nào khác chừng nào nó hỗ trợ cho chế độ đánh địa chỉ. Ví dụ lệnh "ADD A, @R0" sẽ cộng nội dung ngăn nhớ cho RO chỉ đến vào nội dung của thanh ghi A.

CHƯƠNG 6
Các lệnh số học và các chương trình

6.1 Phép cộng và trừ không dấu.

Các số không dấu được định nghĩa như những dữ liệu mà tất cả mọi bit của chúng đều được dùng để biểu diễn dữ liệu và khó có bit dành cho dấu âm hoặc dương. Điều này có nghĩa là toán hạng có thể nằm giữa 00 và FFH (0 đến 255 hệ thập phân) đối với dữ liệu 8 bit.

6.1.1 Phép cộng các số không dấu.

Trong 8051 để cộng các số với nhau thì thanh ghi tổng (A) phải được dùng đến. Dạng lệnh ADD là:

ADD A, nguồn; A = A + nguồn

Lệnh ADD được dùng để cộng hai toán hạng. Toán hạng đích luôn là thanh ghi A trong khi đó toán hạng nguồn có thể là một thanh ghi dữ liệu trực tiếp hoặc là ở trong bộ nhớ. Hãy nhớ rằng các phép toán số học từ bộ nhớ đến bộ nhớ không bao giờ được phép trong hợp ngữ. Lệnh này có thể thay đổi một trong các bit AF, CF hoặc PF của thanh ghi cờ phụ thuộc vào các toán hạng liên quan. Tác động của lệnh ADD lên cờ tràn sẽ được trình bày ở mục 6.3 vì nó chủ yếu được sử dụng trong các phép toán với số có dấu. Xét ví dụ 6.1 dưới đây:

Ví dụ 6.1:

Hãy biểu diễn xem cá lệnh dưới đây tác động đến thanh ghi cờ như thế nào?

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

Lời giải:

```
  F5H                    1111   0101
+ 0BH                  +  0000   1011
  100H                    0000   0000
```

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

6.1.1.1 Phép cộng các byte riêng rẽ.

ở chương 2 đã trình bày một phép cộng 5 byte dữ liệu. Tổng số đã được cất theo chú ý nhỏ hơn FFH là giá trị cực đại một thanh ghi 8 bit có thể được giữ. Để tính tổng số của một số bất kỳ các toán hạng thì cờ nhớ phải được kiểm tra sau mỗi lần cộng một toán hạng. Ví dụ 6.2 dùng R7 để tích luỹ số lần nhớ mỗi khi các toán hạng được cộng vào A.

Ví dụ 6.2:

Giả sử các ngăn nhớ 40 - 44 của RAM có giá trị sau: 40 = (7D); 41 = (EB); 42 = (C5); 43 = (5B) và 44 = (30). Hãy viết một chương trình tính tổng của các giá trị trên. Cuối chương trình giá trị thanh ghi A chứa byte thấp và R7 chứa byte cao (các giá trị trên được cho ở dạng Hex).

Lời giải:

```
        MOV   R0, #40H        ; Nạp con trỏ
        MOV   R2, #5          ; Nạp bộ đệm
        CLR   A               ; Xoá thanh ghi A
        MOV   R7, A           ; Xoá thanh ghi R7
AGAIN:  ADD   A, @R0          ; Cộng byte con trỏ chỉ đến theo R0
        JNC   NEXT            ; Nếu CY = 0 không tích luỹ cờ nhớ
        INC   R7              ; Bám theo số lần nhớ
NEXT:   INC   R0              ; Tăng con trỏ
        DJNZ  R2, AGAIN       ; Lặp lại cho đến khi R0 = 0
```

Phân tích ví dụ 6.2:

Ba lần lặp lại của vòng lặp được chỉ ra dưới đây. Phần dò theo chương trình dành cho người đọc tự thực hiện.

Trong lần lặp lại đầu tiên của vòng lặp thì 7DH được cộng vào A với CY = 0 và R7 = 00 và bộ đếm R2 = 04.

Trong lần lặp lại thứ hai của vòng lặp thì EBH được cộng vào A và kết quả trong A là 68H với CY = 1. Vì cờ nhớ xuất hiện, R7 được tăng lên. Lúc này bộ đếm R2 = 03.

Trong lần lặp lại thứ ba thì C5H được cộng vào A nên A = 2DH và cờ nhớ lại bận. Do vậy R7 lại được tăng lên và bộ đệm R2 = 02.

Ở phần cuối khi vòng lặp kết thúc, tổng số được giữ bởi thanh ghi A và R7, trong đó A giữ byte thấp và R7 chứa byte cao.

6.1.1.2 Phép cộng vó nhớ và phép cộng các số 16 bit.

Khi cộng hai toán hạng dữ liệu 16 bit thì ta cần phải quan tâm đến phép truyền của cờ nhớ từ byte thấp đến byte cao. Lệnh ADDC (cộng có nhớ) được sử dụng trong những trường hợp như vậy. Ví dụ, xét phép cộng hai số sau: 3CE7H + 3B8DH.

```
    3C  E7
+   3B  8D
    78  74
    79
```

Khi byte thứ nhất được cộng (E7 + 8D = 74, CY = 1). Cờ nhớ được truyền lên byte cao tạo ra kết quả 3C + 3B + 1 = 78. Dưới đây là chương trình thực hiện các bước trên trong 8051.

Ví dụ 6.3:

Hãy viết chương trình cộng hai số 16 bit. Các số đó là 3CE7H và 3B8DH. Cất tổng số vào R7và R6 trong đó R6 chứa byte thấp.

Lời giải:

```
CLR                   ; Xoá cờ CY = 0
MOV     A, #0E7H      ; Nạp byte thấp vào A → A = E7H
ADD     A, #8DH       ; Cộng byte thấp vào A → a = 74H và CY = 1
MOV     R6, A         ; Lưu byte thấp của tổng vào R6
MOV     A, #3CH       ; Nạp byte cao vào A → A = 3CH
ADDC    A, #3BG       ; Cộng byte cao có nhớ vào A → A = 78H
                      ;
```

```
MOV     R7, A          ; Lưu byte cao của tổng vào R7
```

6.1.1.3 Hệ thống số BCD (số thập phân mã hoá theo nhị phân).

Số BCD là số thập phân được mã hoá theo nhị phân 9 mà không dùng số thập phân hay số thập lục (Hex). Biểu diễn nhị phân của các số từ 0 đến 9 được gọi là BCD (xem hình 6.1). Trong tài liệu máy tính ta thường gặp hai khái niệm đối với các số BCD là: BCD được đóng gói và BCD không đóng gói.

Digit	BCD	Digit	BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Hình 6.1: Mã BCD.

a- BCD không đóng gói.

Trong số BCD không đóng gói thì 4 bít thấp của số biểu diễn số BCD còn 4 bit còn lại là số 9. Ví dụ "00001001" và "0000 0101" là những số BCD không đóng gói của số 9 và số 5. Số BCD không đóng gói đòi hỏi một byte bộ nhớ hay một thanh ghi 8 bit để chứa nó.

b- BCD đóng gói.

Trong số BCD đóng gói thì một byte có 2 số BCD trong nó một trong 4 bit thấp và một trong 4 bit cao. Ví dụ "0101 1001" là số BCD đóng gói cho 59H. Chỉ mất 1 byte bộ nhớ để lưu các toán hạng BCD. Đây là lý do để dùng số BCD đóng gói vì nó hiệu quả gấp đôi trong lưu giữ liệu.

Có một vấn để khi cộng các số BCD mà cần phải được khắc phục. Vấn đề đó là sau khi cộng các số BCD đóng gói thì kết quả không còn là số BCD. Ví dụ:

```
MOV     A, #17H
ADD     A, #28H
```

Cộng hai số này cho kết quả là 0011 1111B (3FH) không còn là số BCD! Một số BCD chỉ nằm trong giải 0000 đến 1001 (từ số 0 đến số 9). Hay nói cách khác phép cộng hai số BCD phải cho kết quả là số BCD. Kết quả trên đáng lẽ phải là 17 + 28 = 45 (0100 0101). Để giải quyết vấn đề này lập trình viên phải cộng 6 (0110) vào số thấp 3F + 06 = 45H. Vấn đề tương tự cũng có thể xảy ra trong số cao (ví dụ khi cộng hai số 52H + 87H = D94). Để giải quyết vấn đề này ta lại phải cộng 6 vào số cao (D9H + 60H = 139). Vấn đề này phổ biến đến mức mọi bộ xử lý như 8051 đều có một lệnh để sử lý vấn đề này. Trong 8051 đó là lệnh "DA A" để giải quyết vấn đề cộng các số BCD.

6.1.1.4 Lệnh DA.

Lệnh DA (Decimal Adjust for addition điều chỉnh thập phân đối với phép cộng) trong 8051 để dùng hiệu chỉnh sự sai lệch đã nói trên đây liên quan đến phép cộng các số BCD. Lệnh giả "DA". Lệnh DA sẽ cộng 6 vào 4 bit thấp hoặc 4 bit cao nếu cần. Còn bình thường nó đê nguyên kết quả tìm được. Ví dụ sau sẽ làm rõ các điểm này.

```
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). Lệnh "DA" chỉ làm việc với thanh ghi A. Hay nói cách khác trong thanh ghi nguồn có thể là một toán hạng của chế độ đánh địa chỉ bất kỳ thì đích phải là thanh ghi A để DA có thể làm việc được. Cũng cần phải nhấn mạnh rằng lệnh DA phải được sử dụng sau phép cộng các toán hạng BCD và các toán hạng BCD không bao giờ có thể có số lớn hơn 9. Nói cách khác là không cho phép có các số A - F. Điều quan trọng cũng phải lưu ý là DA chỉ làm việc sau phép cộng ADD, nó sẽ không bao giờ làm việc theo lệnh tăng INC.

Tóm tắt về hoạt động của lệnh DA.

Hoạt động sau lệnh ADD hoặc ADDC.

1. Nếu 4 bit thấp lớn hơn 9 hoặc nếu AC = 1 thì nó cộng 0110 vào 4 bít thấp.
2. Nếu 4 bit cao lớn hơn 9 hoặc cờ CY = 1 thì nó cộng 0110 vào 4 bit cao.

Trong thực tế thì cờ AC chỉ để dùng phục vụ cho phép cộng các số BCD và hiệu chỉnh nó. Ví dụ, cộng 29H và 18H sẽ có kết quả là 41H sai với thực tế khi đó các số BCD và để sửa lại thì lệnh DA sẽ cộng 6 vào 4 bit thấp để có kết quả là đúng (vì AC = 1) ở dạng BCD.

```
    29H                  0010  1001
+   18H              +   0001  1000
    41H                  0100  0001        AC = 1
+   6                +         0110
    47H                  0100  0111
```

Ví dụ 6.4:

Giả sử 5 dữ liệu BCD được lưu trong RAM tại địa chỉ bắt đầu từ 40H như sau: 40 = (71), 41 = (11), 42 = (65), 43 = (59) và 44 = (37). Hãy viết chương trình tính tổng của tất cả 5 số trên và kết quả phải là dạng BCD.

Lời giải:

```
        MOV     R0, #40H        ; Nạp con trỏ
        MOV     R2, #5          ; Nạp bộ đếm
        CLR     A               ; Xoá thanh ghi A
        MOV     R7, A           ; Xoá thanhg ghi R7
AGAIN:  ADD     A, @R0          ; Cộng byte con trỏ chỉ bởi R0
        DA      A               ; Điều chỉnh về dạng BCD đúng
        JNC     NEXT            ; Nếu CY = 0 không tích luỹ cờ nhớ
        JNC     R7              ; Tăng R7 bám theo số lần nhớ
NEXT:   INC     R0              ; Tăng R0 dịch con trỏ lên ô nhớ kế tiếp
        DJNZ    R2, AGAIN       ; Lặp lại cho đến khi R2 = 0
```

6.1.2 Phép trừ các số không dấu.

Cú pháp: SUBB A, nguồn; A = A - nguồn - CY.

Trong rất nhiều các bộ xử lý có hai lệnh khác nhau cho phép trừ đó là SUB và SUBB (trừ có mượn - Sub, tract with Borrow). Trong 8051 ta chỉ có một lệnh SUBB

duy nhất. Để thực hiện SUB từ SUBB, do vậy có hai trường hợp cho lệnh SUBB là: với CY = 0 và với CY = 1. Lưu ý rằng ở đây ta dùng cờ CY để mượn.

6.1.2.1 Lệnh SUBB với CY = 0.

Trong phép trừ thì các bộ vi xử lý 8051 (thực tế là tất cả mọi CPU hiện đại) đều sử dụng phương pháp bù 2. Mặc dù mỗi CPU đều có mạch cộng, nó có thể quá cồng kềnh (và cần nhiều bóng bán dẫn) để thiết kế mạch trừ riêng biệt. Vì ly do đó mà 8051 sử dụng mạch cộng để thực hiện lệnh trừ. Giả sử 8051 sử dụng mạch cộng để thực hiện lệnh trừ và rằng CY - 0 trước khi thực hiện lệnh thì ta có thể tóm tắt các bước mà phần cứng CPU thực hiện lệnh SUBB đối với các số không dấu như sau:

1. Thực hiện lấy bù 2 của số trừ (toán hạng nguồn)
2. Cộng nó vào số bị trừ (A)
3. Đảo nhớ

Đây là 3 bước thực hiện bởi phần cứng bên trong của CPU 8051 đối với mỗi lệnh trừ SUBB bất kể đến nguồn của các toán hạng được cấp có được hỗ trợ chế độ đánh địa chỉ hay không? Sau ba bước này thì kết quả có được và các cờ được bật. Ví dụ 6.5 minh hoạ 3 bước trên đây:

Ví dụ 6.5:

Trình bày các bước liên quan dưới đây:

```
CLR     C           ; Tạo CY = 0
MOV     A, #3FH     ; Nạp 3FH vào A (A = 3FH)
MOV     R3, #23H    ; Nạp 23H vào R3 (R3 = 23H)
SUBB    A, R3       ; Trừ A cho R3 đặt kết quả vào A
```

Lời giải:

```
    A =  3F     0011  1111        0011  1111
-   R3 = 23     0010  0011    +   1101  1101   bù 2 của R3 (bước 1)
      1C                        1  0001  1100 - 1C (bước 2)
                                0  CF = 0 (bước 3)
```

Các cờ sẽ được thiết lập như sau: CY = 0, AC = 0 và lập trình viên phải được nhìn đến cờ nhớ để xác định xem kết quả là âm hay dương.

Nếu sau khi thực hiện SUBB mà CY = 0 thì kết quả là dương. Nếu CY = 1 thì kết quả âm và đích có giá trị bù 2 của kết quả. Thông thường kết quả được để ở dạng bù 2 nhưng các lệnh bù CPL và tăng INC có thể được sử dụng để thay đổi nó. Lệnh CPL thực hiện bù 1 của toán hạng sau đó toán hạng được tăng lên 1 (INC) để trở thành dạng bù 2. Xem ví dụ 6.6.

Ví dụ 6.6:

Phân tích chương trình sau:

```
        CLR   C
        MOV   A, #4CH       ; Nạp A giá trị 4CH (A = 4CH)
        SUBB  A, #6EH       ; Trừ A cho 6EH
        JNC   NEXT          ; Nếu CY = 0 nhảy đến đích NEXT
        CPL   A             ; Nếu CY = 1 thực hiện bù 1
        INC   A             ; Tăng 1 để có bù 2
NEXT:   MOV   R1, A         ; Lưu A vào R1
```

Lời giải:

Các bước thực hiện lệnh "SUBB A, 6EH" như sau:

```
    4C      0100    1100                    0100    1100
-   6E      0110    1110    → lấy bù 2      1001    0010    (bước 1)
  - 22                              0       1101    1110 = (bước 2)
                                            đảo CY = 1(bước 3)
```

Cờ CY = 1, kết quả âm ở dạng bù 2.

6.1.2.2 Lệnh SUBB khi CY = 1.

Lệnh này được dùng đối với các số nhiều byte và sẽ theo dõi việc mượn của toán hạng thấp. Nếu CY = 1 trước khi xem thực hiện SUBB thì nó cũng trừ 1 từ kết quả. Xem ví dụ 6.7.

Ví dụ 6.7:

Phân tích chương trình sau:

```
CLR   C          ; CY = 0
MOV   A, #62     ; A = 62H
SUBB  A, #96H    ; 62H - 96H = CCH with CY = 1
MOV   R7, A      ; Save the result
MOV   A, #27H    ; A = 27H
SUBB  A, #12H    ; 27H - 12H - 1 = 14H
MOV   R6, A      ; Save the result
```

Lời giải:

Sau khi SUBB thì A = 62H - 96H = CCH và cờ nhớ được lập báo rằng có mượn. Vì CY = 1 nên khi SUBB được thực hiện lần thứ 2 thì a = 27H - 12H - 1 = 14H. Do vậy, ta có 2762H - 1296H = 14CCH.

6.2 Nhân và chia các số không dấu.

Khi nhân và chia hai số trong 8051 cần phải sử dụng hai thanh ghi A và B vì các lệnh nhân và chia chỉ hoạt động với những thanh ghi này.

6.2.1 Nhân hai số không dấu.

Bộ vi điều khiển chỉ hỗ trợ phép nhân byte với byte. Các byte được giả thiết là dữ liệu không dấu. Cấu trúc lệnh như sau:

MOV AB ; Là phép nhân A × B và kết quả 16 bit được đặt trong A và B.

Khi nhận byte với byte thì một trong các toán hạng phải trong thanh ghi A và toán hạng thứ hai phải ở trong thanh ghi B. Sau khi nhân kết quả ở trong các thanh ghi A và B. Phần tiếp thấp ở trong A, còn phần cao ở trong B. Ví dụ dưới đây trình bày phép nhân 25H với 65H. Kết quả là dữ liệu 16 bit được đặt trong A và B.

```
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
```

Bảng 6.1: Tóm tắt phép nhân hai số không dấu (MULAB)

Nhân	Toán hạng 1	Toán hạng 2	Kết quả
Byte*Byte	A	B	A = byte thấp, B = byte cao

6.2.2 Chia hai số không dấu.

8051 cùng chỉ hỗ trợ phép chia hai số không dấu byte cho byte với cú pháp:

DIV AB ; Chia A cho B

Khi chia một byte cho một byte thì tử số (số bị chia) phải ở trong thanh ghi A và mẫu số (số chia) phải ở trong thanh ghi B. Sau khi lệnh chia DIV được thực hiện thì thương số được đặt trong A, còn số dư được đặt trong B. Xét ví dụ dưới đây:

```
MOV     A, #95      ; Nạp số bị chia vào A = 95
MOV     B, #10      ; Nạp số chia vào B = 10
DIV     AB          ; A = 09 (thương số); B = 05 (số dư)
```

Lưu ý các điểm sau khi thực hiện "DIV AB"

Lệnh này luôn bắt CY = 0 và OV = 0 nếu tử số không phải là số 0

Nếu tử số là số 0 (B = 0) thì OV =1 báo lỗi và CY = 0. Thực tế chuẩn trong tất cả mọi bộ vi xử lý khi chia một số cho 0 là bằng cách nào đó báo có kết quả không xác định. Trong 8051 thì cờ OV được thiết lập lên 1.

Bảng 6.2: Tóm tắt phép chia không dấu (DIV AB).

Phép chia	Tử số	Mẫu số	Thương số	Số dư
Byte cho Byte	A	B	A	B

6.2.3 Một ứng dụng cho các lệnh chia.

Có những thời điểm khi một bộ ADC được nối tới một cổng và ADC biểu diễn một số dư nhiệt độ hay áp suất. Bộ ADC cấp dữ liệu 8 bit ở dạng Hex trong dải 00 - FFH. Dữ liệu Hex này phải được chuyển đổi về dạng thập phân. Chúng ta thực hiện chia lặp nhiều lần cho 10 và lưu số dư vào như ở ví dụ 6.8.

Ví dụ 6.8:

a- Viết một chương trình để nhận dữ liệu dạng Hex trong phạm vi 00 - FFH từ cổng 1 và chuyển đổi nó về dạng thập phân. Lưu các số vào trong các thanh ghi R7, R6 và R5 trong đó số có nghĩa nhỏ nhất được cất trong R7.

b- Phân tích chương trình với giả thiết P1 có giá trị FDH cho dữ liệu.

Lời giải:

a)

```
MOV     A, #0FFH
MOV     P1, A       ; Tạo P1 là cổng đầu vào
MOV     A, P1       ; Đọc dữ liệu từ P1
MOV     B, #10      ; B = 0A Hex (10 thập phân)
DIV     AB          ; Chia cho 10
MOV     R7, B       ; Cất số thấp
MOV     B, #10      ;
DIV     AB          ; Chia 10 lần nữa
MOV     R6, B       ; Cất số tiếp theo
MOV     R5, A       ; Cất số cuối cùng
```

b) Để chuyển đổi số nhị phân hay Hex về số thập phân ta thực hiện chia lặp cho 10 liên tục cho đến khi thương số nhỏ hơn 10. Sau mỗi lần chia số dư được lưu cất.

Trong trường hợp một số nhị phân 8 bit như FDH chẳng hạn ta có 253 số thập phân như sau (tất cả trong dạng Hex)

	Thương số	**Số dư**	
FD/0A	19	3	(Số thấp - cuối)
19/0A	2	5	(Số giữa)
		2	(Số đầu)

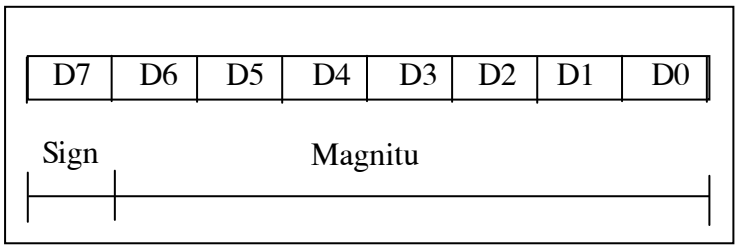
Do vậy, ta có FDH = 253. Để hiển thị dữ liệu này thì nó phải được chuyển đổi về ASCII mà sẽ được mô tả ở chương sau.

6.3 Các khái niệm về số có dấu và các phép tính số học.

Tất cả mọi dữ liệu từ trước đến giờ đều là các số không dấu, có nghĩa là toàn bộ toán hạng 8 bit đều được dùng cho bộ lớn. Có nhiều ứng dụng yêu cầu dữ liệu có dấu, phần này sẽ bàn về những lệnh liên quan đến các số có dấu.

6.3.1 Khái niệm về các số có dấu trong máy tính.

Trong cuộc sống hàng ngày các số được dùng có thể là số âm hoặc dương. Ví dụ 5 độ dưới 0^0C được biểu diễn là -5^0C và 20 độ trên 0^0C được biểu diễn là $+20^0$C. Các máy tính cũng phải có khả năng đáp ứng phù hợp với các số ấy. Để làm được điều ấy các nhà khoa học máy tính đã phát minh ra sự xắp xếp biểu diễn các số âm có dấu và số dương có dấu như sau: Bit cao nhất MSB được để dành cho bit dấu (+) hoặc (-), còn các bit còn lại được dùng biểu diễn độ lớn. Dấu được biểu diễn bởi 0 đối với các số dương và một số đối với các số âm (-). Biểu diễn của một byte có dấu được trình bày trên hình 6.2.



Hình 6.2: Các toán hạng 8 bit có dấu.

a- Các toán hạng 8 bit có dấu: Trong các toán hạng A byte có dấu thì bit cao nhất MSB là D7 được dùng để biểu diễn dấu, còn 7 bit còn lại từ D6 - D0 dùng để biểu diễn độ lớn của số đó. Nếu D7 = 0 thì đó là toán hạng dương và nếu D7 = 1 thì nó là toán hạng âm.

b- Các số dương: Dải của các số dương có thể được biểu diễn theo dạng cho trên hình 6.2 là từ 0 đến +127 thì phải sử dụng toán hạng 16 bit. Vì 8051 không hỗ trợ dữ liệu 16 bit nên ta không bàn luận đến.

c- Các số âm: Đối với các số âm thì D7 = 1, tuy nhiên độ lớn được biểu diễn ở dạng số bù 2 của nó. Mặc dù hợp ngữ thực hiện việc chuyển đổi song điều quan trọng là hiểu việc chuyển đổi diễn ra như thế nào. Để chuyển đổi về dạng biểu diễn số âm (bù 2) thì tiến hành theo các bước sau:

1. Viết độ lớn của số ở dạng nhị phân 8 bit (không dấu).
2. Đảo ngược tất cả các bit
3. Cộng 1 vào nó.

Ví dụ 6.9: Hãy trình bày cách 8051 biểu diễn số - 5.

Lời giải:

Hãy quan sát các bước sau:

0000	0101	Biểu diễn số 5 ở dạng 8 bit nhị phân
1111	1010	Đảo các bit
1111	1011	Cộng (thành số FB ở dạng Hex)

Do vậy, số FBH là biểu diễn số có dấu dạng bù 2 của số - 5.

Ví dụ 6.10: Trình bày cách 8051 biểu diễn - 34H.

Lời giải:

Hãy quan sát các bước sau:

0011	0200	Số 34 được cho ở dạng nhị phân
1100	1011	Đảo các bit
1100	1100	Cộng 1 (thành số CC ở dạng Hex)

Vậy số CCH là biểu diễn dạng bù 2 có dấu của - 34H.

Ví dụ 6.11: Trình bày cách 8051 biểu diễn - 128:

Lời giải:

Quan sát các bước sau:

1000	0000	Số 128 ở dạng nhị phân 28 bit
0111	1111	Đảo các bit
1000	0000	Cộng 1 (trở thành số 80 dạng Hex)

Vậy - 128 = 80H là biểu diễn số có dấu dạng bù 2 của - 128.

Từ các ví dụ trên đây ta thấy rõ ràng rằng dải của các số âm có dấu 8 bit là - 1 đến - 128. Dưới đây là liệt kê các số có dấu 8 bit:

Số thập phân	Số nhị phân	Số Hex
-128	1000 0000	80
-127	1000 0001	81
-126	1000 0010	82
...	.....	...
-2	1111 1110	FE
-1	1111 1111	FF
0	0000 0000	00
+1	0000 0001	01
+2	0000 0010	02
...	.....	...
-127	0111 1111	FE

6.3.2 Vấn đề tràn trong các phép toán với số có dấu.

Khi sử dụng các số có dấu xuất hiện một vấn đề rất nghiêm trọng mà phải được sử lý. Đó là vấn đề tràn, 8051 báo có lỗi bằng cách thiết lập cờ tràn OV nhưng trách nhiệm của lập trình viên là phải cẩn thận với kết quả sai. CPU chỉ hiểu 0 và 1 và nó làm ngơ với việc chuyển đổi số âm, số dương của con người. Vậy tràn số là gì? Nếu kết quả của một phép toán trên các số có dấu mà quá lớn đối với thanh ghi thì xuất hiện sự tràn số và lập trình viên phải được cảnh báo. Xét ví dụ 6.12 dưới đây.

Ví dụ 6.12:

Khảo sát đoạn mã sau và phân tích kết quả.

```
MOV     A, # + 96       ; A = 0110      0000 (A = 60H)
MOV     R1, # + 70      ; R1 = 0100     0110 (R1 = 46H)
ADD     A, R1           ; A = 1010      0110 = A6H = - 90
                          Sai !!!
```

Lời giải:

```
    + 96                0110    0000
+   + 70                0100    0110
  - 166                 1010    0110        và OV = 1
```

Theo CPU kết quả là -90 và đó là kết quả sai nên CPU bật cờ OV = 1 để báo tràn số.

Trong ví dụ 6.12 thì + 96 được cộng với + 70 và kết quả theo CPU là - 90. Tại sao vậy? Lý do là kết quả của + 96 + 70 = 172 lớn hơn số mà thanh ghi A có thể chứa được. Cũng như tất cả mọi thanh ghi 8 bit khác, thanh ghi A chỉ chứa được đến số + 127. Các nhà thiết kế của PCU tạo ra cờ tràn OV phục vụ riêng cho mục đích báo cho lập trình viên rằng kết quả của phép toán số có dấu là sai.

6.3.3 Khi nào thì cờ tràn OV được thiết lập?

Trong các phép toán với số có dấu 8 bit thì cờ OV được bật lên 1 khi xuất hiện một trong hai điều kiện sau:

1. Cờ nhớ từ D6 sang D7 nhưng không có nhớ ra từ D7 (cờ CY = 0)
2. Có nhớ ra từ D7 (cờ CY = 1) nhưng không có nhớ từ D6 sang D7

Hay nói cách khác là cờ tràn OV được bật lên 1 nếu có nhớ từ D6 sang D7 hoặc từ D7 nhưng không đồng thời xảy ra cả hai. Điều này có nghĩa là nếu có nhớ cả từ D6 sang D7 và từ D7 ra thì cờ OV = 0. Trong ví dụ 6.12 vì chỉ có nhớ từ D7 ra nên cờ OV = 1. Trong ví dụ 6.13, ví dụ 6.14 và 6.15 có minh hoạ thêm về sử dụng cờ tràn trong các phép số học với số có dấu.

Ví dụ 6.13:

Hãy quan sát đoạn mã sau để ý đến vai trò của cờ OV.

```
MOV     A, # -128       ; A = 1000      0000 (A= 80H)
MOV     R4, # -2        ; R4 = 1111     (R4 = FEH)
ADD     A, R4           ; A = 0111      1110 (A = 7EH = +126, invalid)
```

Lời giải:

```
- 128           1000    0000
+ - 2           1111    1110
-130            0111    1110    và OV = 1
```

Theo CPU thì kết quả + 126 là kết quả sai, nên cờ OV = 1.

Ví dụ 6.14:

Hãy quan sát đoạn mã sau và lưu ý cờ OV.

```
MOV     A, # -2      ; A = 1111    1110 (A = FEH)
MOV     R1, # -5     ; R1 = 1111   1011   (R1 = FBH)
ADD     A, R1        ; A = 1111    1001   (A = F9H = -7, correct, OV = 0)
```

Lời giải:

```
  - 2        1111   1110
+ - 5        1111   1011
-   7        1111   1001   và OV = 0
```

Theo CPU thì kết quả - 7 là đúng nên cờ OV = 0.

Ví dụ 6.15:

Theo dõi đoạn mã sau, chú ý vai trò của cờ OV.

```
MOV   A, # +7      ; A =  0000    0111 (A = 07H)
MOV   R1, # +18    ; R1 = 0001    0010 (R1 = 12H)
ADD   A, R1        ; A = 1111     1001   (A = 19H = -25, correct, OV = 0)
```

Lời giải:

```
    7      0000   0111
- 18       0001   0010
  25       0001   1001   và OV = 0
```

Theo CPU thì kết quả - 25 là đúng nên cờ OV = 0.

Từ các ví dụ trên đây ta có thể kết luận rằng trọng bất kỳ phép cộng số có dấu nào, cờ OV đều báo kết quả là đúng hay sai. Nếu cờ OV = 1 thì kết quả là sai, còn nếu OV = 0 thì kết quả là đúng. Chúng ta có thể nhấn mạnh rằng, trong phép cộng các số không dấu ta phải hiển thị trạng thái của cờ CY (cờ nhớ) và trong phép cộng các số có dấu thì cờ tràn OV phải được theo dõi bởi lập trình viên. Trong 8051 thì các lệnh như JNC và JC cho phép chương trình rẽ nhánh ngay sau phép cộng các số không dấu như ở phần 6.1. Đối với cờ tràn OV thì không có như vậy. Tuy nhiên, điều này có thể đạt được bằng lệnh "JB PSW.2" hoặc "JNB PSW.2" vì PSW thanh ghi cờ có thể đánh địa chỉ theo bit.

CHƯƠNG 7
Các lệnh lôgíc và các chương trình

7.1 Các lệnh lô-gíc và so sánh.

7.1.1 Lệnh VÀ (AND).

Cú pháp: ANL đích, nguồn; đích = đích Và nguồn (kẻ bảng).

Lệnh này sẽ thực hiện một phép Và lô-gíc trên hai toán hạng đích và nguồn và đặt kết quả vào đích. Đích thường là thanh ghi tổng (tích luỹ). Toán hạngnguồn có thể là thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. Hãy xem phụ lục Appendix A1 để biết thêm về các chế độ đánh địa chỉ dành cho lệnh này. Lệnh ANL đối với toán hạng theo byte không có tác động lên các cờ. Nó thường được dùng để che (đặt về 0) những bit nhất định của một toán hạng. Xem ví dụ 7.1.

Ví dụ:

Trình bày kết quả của các lệnh sau:

```
MOV    A, #35H          ; Gán A = 35H
ANL    A, #0FH          ; Thực hiện Và lô-gíc A và 0FH (Bây giờ A = 05)
```

Lời giải:

```
35H    0  0  1  1  0  1  0  1
0FH    0  0  0  0  1  1  1  1
05H    0  0  0  0  0  1  0  1          35H  và 0FH = 05H
```

7.1.2: Lệnh HOẶC (OR).

Cú pháp ORL đích = đích Hoặc nguồn (kẻ bảng)

Các toán hạng đích và nguồn được Hoặc với nhau và kết quả được đặt vào đích. Phép Hoặc có thể được dùng để thiết lập những bit nhất định của một toán hạng 1. Đích thường là thanh ghi tổng, toán hạng nguồn có thể là một thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. Hãy tham khảo phụ lục Appendix A để biết thêm về các chế độ đánh địa chỉ được hỗ trợ bởi lệnh này. Lệnh ORL đối với các toán hạng đánh địa chỉ theo byte sẽ không có tác động đến bất kỳ cờ nào. Xem ví dụ 7.2.

Ví dụ 7.2: Trình bày kết quả của đoạn mã sau:

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

Lời giải:

```
04H    0000  0100
68H    0110  1000
6CH    0110  1100              04  OR  68 = 6CH
```

7.1.3 Lệnh XOR (OR loại trừ?).

Cú pháp: XRL đích, nguồn; đích = đích Hoặc loại trừ nguồn (kẻ bảng).

Lệnh này sẽ thực hiện phép XOR trên hai toán hạng và đặt kết quả vào đích. Đích thường là thanh ghi tổng. Toán hạng nguồn có thể là một thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. Xem phụ lục Appendix A.1 để biết thêm về chế độ đánh địa chỉ của lệnh này. Lệnh XRL đối với các toán hạng đánh địa chỉ theo byte sẽ không có tác động đến bất kỳ cờ nào. Xét ví dụ 7.3 và 7.4.

Ví dụ 7.3: Trình bày kết quả của đoạn mã sau:

```
MOV    A, #54H
XRL    A, #78H
```

Lời giải:

```
54H    0  1  0  1  0  1  0  0
78H    0  1  1  1  1  0  0  0
2CH    0  0  1  0  1  1  0  1          54H  XOR  78H = 2CH
```

Ví dụ 7.4:

Lệnh XRL có thể được dùng để xoá nội dung của một thanh ghi bằng cách XOR nó với chính nó. Trình bày lệnh "XRL A, A" xoá nội dung của A như thế nào? giả thiết AH = 45H.

Lời giải:

```
45H            01000101
45H            01000101
00             00000000          54H  XOR  78H = 2CH
```

Lệnh XRL cũng có thể được dùng để xem nếu hai thanh ghi có giá trị giống nhau không? Lệnh "XRL A, R1" sẽ hoặc loại trừ với thanh ghi R1 và đặt kết quả vào A. Nếu cả hai thanh ghi có cùng giá trị thì trong A sẽ là 00. Sau đó có thể dùng lệnh nhả JZ để thực hiện theo kết quả. Xét ví dụ 7.5.

Ví dụ 7.5:

Đọc và kiểm tra cổng P1 xem nó có chứa giá trị 45H không? Nếu có gửi 99H đến cổng P2, nếu không xoá nó.

Lời giải:

```
        MOV    P2, #00          ; Xóa P2
        MOV    P1, #0FFH        ; Lấy P1 là cổng đầu vào
        MOV    R3, #45H         ; R3 = 45H
        MOV    A, P1            ; Đọc P1
        XRL    A, R3
        JNZ    EXIT             ; Nhảy nếu A có giá trị khác 0
        MOV    P2, #99H
EXIT: ...
```

Trong chương trình của ví dụ 7.5 lưu ý việc sử dụng lệnh nhảy JNZ. Lệnh JNZ và JZ kiểm tra các nội dung chỉ của thanh ghi tổng. Hay nói cách khác là trong 8051không có cờ 0.

Một ứng dụng rộng rãi khác của bộ xử lý là chọn các bit của một toán hạng. Ví dụ để chọn 2 bit của thnh ghi A ta có thể sử dụng mã sau. Mã này ép bit D2 của thanh ghi A chuyển sang giá trị nghịch đảo, còn các bit khác không thay đổi.

```
XRL     A, #04H         ; Nghĩa hoặc loại trừ thanh ghi A với
                        ; Giá trị 0000 0100
```

7.1.4 Lệnh bù thanh ghi tổng CPL A.

Lệnh này bù nội dung của thanh ghi tổng A. Phép bù là phép biến đổi các số 0 thành các số 1 và đổi các số 1 sang số 0. Đây cũng còn được gọi là phép bù 1.

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

Để nhận được kết quả bù 2 thì tất cả mọi việc ta cần phải làm là cộng 1 vào kết quả bù 1. Trong 8051 thì không có lệnh bù 2 nào cả. Lưu ý rằng trong khi bù một byte thì dữ liệu phải ở trong thanh ghi A. Lệnh CPL không hỗ trợ một chế độ đánh địa chỉ nào cả. Xem ví dụ 7.6 dưới đây.

Ví dụ 7.6: Tìm giá trị bù 2 của 85H.

Lời giải:

```
MOV         A, #85H     ; Nạp 85H vào A (85H = 1000 0101)
MOV         A           ; Lấy bù 1 của A (kết quả = 0111 1010)
ADD         A, #1       ; Cổng 1 vào A thành bù 2 A = 0111 1011 (7BH)
```

Ví dụ 7.1.5 Lệnh so sánh.

8051 có một lệnh cho phép so sánh. Nó có cú pháp như sau:

CJNE đích, nguồn, địa chỉ tương đối.

Trong 8051 thì phép so sánh và nhảy được kết hợp thành môt lệnh có tên là CJNE (so sánh và nhảy nếu kết quả không bằng nhau). Lệnh CJNE so sánh hai toán hạng nguồn và đích và nhảy đến địa chỉ tương đối nếu hai toán hạng không bằng nhau. Ngoài ra nó thay đổi cờ nhớ CY để báo nếu toán hạng đích lớn hơn hay nhỏ hơn. Điều quan trọng cần để là các toán hạng vẫn không giữ nguyên không thay thay đổi. Ví dụ, sau khi thực hiện lệnh "CJNE A, #67H, NEXT" thì thanh ghi A vẫn có giá trị ban đầu của nó (giá trị trước lệnh CJNE). Lệnh này so sánh nội dung thanh ghi A với giá trị 67H và nhảy đến giá trị đích NEXT chỉ khi thanh ghi A có giá trị khác 67H.

Ví dụ 7.7:

Xét đoạn mã dưới đây sau đó trả lời câu hỏi:

a) Nó sẽ nhảy đến NEXT không?

b) Trong A có giá trị bao nhiêu sau lệnh CJNE?

```
        MOV         A, #55H
        CJNE        A, #99H, NEXT
        ...
NEXT: ...
```

Lời giải:

a) Có vì 55H và 99H không bằng nhau

b) A = 55H đây là giá trị trước khi thực hiện CJNE.

Trong lệnh CJNE thì toán hạng đích có thể trong thanh ghi tổng hoặc trong một các thanh ghi Rn. Toán hạng nguồn có thể trong một thanh ghi, trong bộ nhớ hoặc giá trị cho sẵn. Hãy xem phụ lục Appendix A để biết thêm chi tiết về các chế độ đánh địa chỉ cho lệnh này. Lệnh này chỉ tác động cờ nhớ CY. Cờ này được thay đổi như chỉ ra trên bảng 7.1. Dưới đây trình bày phép so sánh hoạt động như thế nào đối với tất cả các điều kiện có thể:

```
            CJNE   R5, #80, NOT-EQUAL      ; Kiểm tra R5 có giá trị 80?
            ...                            ; R5 = 80
NOT-EQUAL:  JNC    NEXT                    ; Nhảy đến R5 > 80
            ...
NEXT:        ...
```

Bảng 7.1: Thiết kế cờ CY cho lệnh CJNE.

Compare	Carry Flag
Destinatio > Source	CY = 0
Destination < Source	CY = 1

Để ý rằng trong lệnh CJNE thì không có thanh ghi Rn nào có thể được so sánh với giá trị cho sẵn. Do vậy không cần phải nói đến thanh ghi A. Cũng cần lưu ý rằng cờ nhớ CY luôn được kiểm tra để xem lớn hơn hay nhỏ hơn, nhưng chỉ khi đã xác định là nó không bằng nhau. Xét ví dụ 7.8 và 7.9 dưới đây.

Ví dụ 7.8:

Hãy viết mã xác định xem thanh ghi A có chứa giá trị 99H không? Nếu có thì hãy tạo R1 = FFH còn nếu không tạo R1 = 0.

Lời giải:

```
            MOV    R1, #0          ; Xoá R1
            CJNE   A, #99H ; Nếu A không bằng 99H thì nhảy đến NEXT
            MOV    R1, #0FFH       ; Nếu chúng bằng nhau, gán R1 = 0FFH
NEXT: ...                          ; Nếu không bằng nhau, gán R1 = 0
OVER: ...
```

Ví dụ 7.9:

Giả sử P1 là một cổng đầu vào được nối tới một cảm biến nhiệt. Hãy viết chương trình đọc nhiệt độ và kiểm tra nó đối với giá trị 75. Theo kết quả kiểm tra hãy đặt giá trị nhiệt độ vào các thanh ghi được chỉ định như sau:

```
Nếu T = 75      thì A = 75
Nếu T < 75      thì R1 = T
Nếu T > 75      thì R2 = T
```

Lời giải:

```
        MOV   P1, 0FFH        ; Tạo P1 làm cổng đầu vào
        MOV   A, P1           ; Đọc cổng P1, nhiệt độ
        CJNE  A, #75, OVER    ; Nhảy đến OVER nếu A ≠ 75
        SJMP  EXIT            ; A = 75 thoát
OVER:   JNC   NEXT            ; Nếu CY = 0 thì A > 75 nhảy đến NEXT
        MOV   R1, A           ; Nếu CY = 1 thì A < 75 lưu vào R1
        SJMP  EXIT            ; Và thoát
NEXT:   MOV   R2, A           ; A > 75 lưu nó vào R2
EXIT:   ...
```

Lệnh so sánh thực sự là một phép trừ, ngoại trừ một điều là giá trị của các toán hạng không thay đổi. Các cờ được thay đổi tuỳ theo việc thực hiện lệnh trừ SUBB. Cần phải được nhấn mạnh lại rằng, trong lệnh CJNE các toán hạng không bị tác động bất kể kết quả so sánh là như thế nào. Chỉ có cờ CY là bị tác động, điều này bị chi phối bởi thực tế là lệnh CJNE sử dụng phép trừ để bật và xoá cờ CY.

Ví dụ 7.10:

Viết một chương trình để hiển thị liên tục cổng P1 đối với giá trị 63H. Nó chỉ mất hiển thị khi P1 = 63H.

Lời giải:

```
        MOV   P1, #0FFH       ; Chọn P1 làm cổng đầu vào
HERE:   MOV   A, P1           ; Lấy nội dung của P1
        CJNE  A, #63, HERE    ; Duy trì hiển thị trừ khi P1 = 63H
```

Ví dụ 7.11:

Giả sử các ngăn nhớ của RAM trong 40H - 44H chứa nhiệt độ hàng ngày của 5 ngày như được chỉ ra dưới đây. Hãy tìm để xem có giá trị nào bằng 65 không? Nếu giá trị 65 có trong bảng hãy đặt ngăn nhớ của nó vào R4 nếu không thì đặt R4 = 0.

40H = (76); 41H = (79); 42H = (69); 43H = (65); 44H = (64)

Lời giải:

```
        MOV   R4, #0          ; Xoá R4 = 0
        MOV   R0, #40H        ; Nạp con trỏ
        MOV   R2, #05         ; Nạp bộ đếm
        MOV   A, #65          ; Gán giá trị cần tìm vào A
BACK:   CJNE  A, @R0, NEXT    ; So sánh dữ liệu RAM với 65
        MOV   R4, R0          ; Nếu là 65, lưu địa chỉ vào R4
        SJMP  EXIT            ; Thoát
NEXT:   INC   R0              ; Nếu không tăng bộ đếm
        DJNZ  R2, BACK        ; Tiếp tục kiểm tra cho đến khi bộ đếm bằng 0.
EXIT:   ...
```

7.2 Các lệnh quay vào trao đổi.

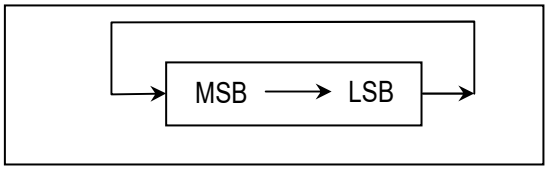
Trong rất nhiều ứng dụng cần phải thực hiện phép quay bit của một toán hạng. Các lệnh quay 8051 là R1, RR, RLC và RRC được thiết kế đặc biệt cho mục đích này. Chúng cho phép một chương trình quay thanh ghi tổng sang trái hoặc phải. Trong 8051 để quay một byte thì toán hạng phải ở trong thanh ghi tổng A. Có hai kiểu quay là: Quay đơn giản các bit của thanh ghi A và quay qua cờ nhớ (hay quay có nhớ).

7.2.1 Quay các bit của thanh ghi A sang trái hoặc phải.

a) Quay phải: RR A ; Quay các bit thanh ghi A sang phải.

Trong phép quay phải, 8 bit của thanh ghi tổng được quay sang phải một bit và bit D0 rời từ vị trí bit thấp nhất và chuyển sang bit cao nhất D7. Xem đoạn mã dưới đây.

```
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
```



b) Quay trái:

Cú pháp: RL A ; Quay trái các bit của thanh ghi A (hình vẽ)

Trong phép quay trái thì 8 bit của thanh ghi A được quay sang trái 1 bit và bit D7 rời khỏi vị trí bit cao nhất chuyển sang vị trí bit thấp nhất D0. Xem biểu đồ mã dưới đây.

```
MOV   A, #72H      ; A = 0111 0010
RL    A            ; A = 1110 0100
RL    A            ; A = 1100 1001
```

Lưu ý rằng trong các lệnh RR và RL thì không có cờ nào bị tác động.

7.2.2 Quay có nhớ.

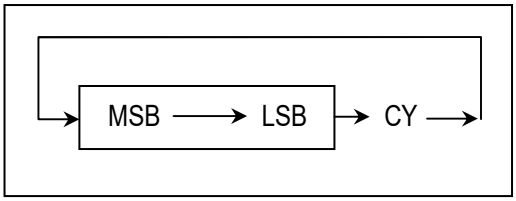
Trong 8051 còn có 2 kênh quay nữa là quay phải có nhớ và quay trái có nhớ.

Cú pháp: RRC A và RLC A

a) Quay phải có nhớ: RRC A

Trong quay phải có nhớ thì các bit của thanh ghi A được quay từ trái sang phải 1 bit và bit thấp nhất được đưa vào cờ nhớ CY và sau đó cờ CY được đưa vào vị trí bit cao nhất. Hay nói cách khác, trong phép RRC A thì LSB được chuyển vào CY và CY được chuyển vào MSB. Trong thực tế thì cờ nhớ CY tác động như là một bit bộ phận của thanh ghi A làm nó trở thành thanh ghi 9 bit.

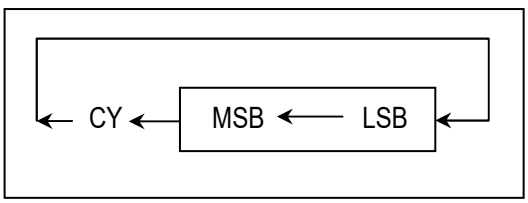
```
CLR   C            ; make CY = 0
MOV   A #26H       ; A = 0010 0110
RRC   A            ; A = 0001 0011  CY = 0
RRC   A            ; A = 0000 1001  CY = 1
RCC   A            ; A = 1000 0100  CY = 1
```



b) Quay trái có nhớ (hình vẽ): RLC A.

Trong RLC A thì các bit được dịch phải một bit và đẩy bit MSB vào cờ nhớ CY, sau đó CY được chuyển vào bit LSB. Hay nói cách khác, trong RLC thì bit MSB được chuyển vào CY và CY được chuyển vào LSB. Hãy xem đoạn mã sau.

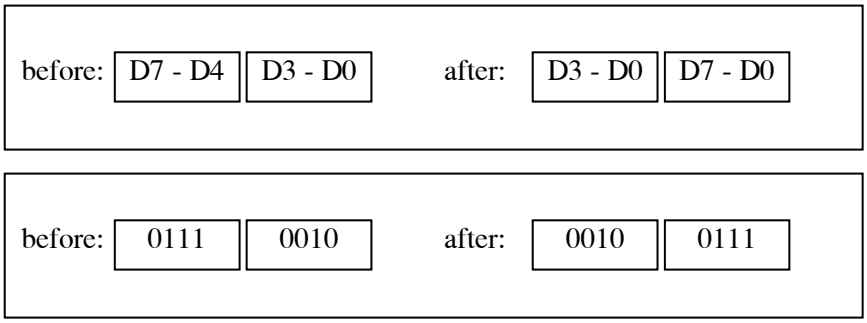
```
SETB  C            ; Make CY = 1
MOV   A #15H       ; A = 0001 0101
RRC   A            ; A = 0101 1011  CY = 0
RRC   A            ; A = 0101 0110  CY = 0
```



```
RCC     A               ; A = 1010 1100  CY = 0
RCC     A               ; A = 1000 1000  CY = 1
```

7.2.3 Lệnh trao đổi thanh ghi A: SWAP A

Một lệnh hữu ích khác nữa là lệnh trao đổi SWAP. Nó chỉ hoạt động trên thanh ghi A, nó trao đổi nửa phần cao của byte và nửa phần thấp của byte với nhau. Hay nói cách khác 4 bit cao được chuyển thành 4 bit thấp và 4 bit thấp thành 4 bit cao.



Ví dụ 7.12:

a) Hãy tìm nội dung của thanh ghi A ở đoạn mã sau.
b) Trong trường hợp không có lệnh SWAP thì cần phải làm như thế nào để trao đổi những bit này? Hãy viết một mã chương trình đơn giản về quá trình đó.

Lời giải:

a)

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

b)

```
        MOV     A, #72H ; A = 0111 0010
        RL      A               ; A = 1110 0100
        RL      A               ; A = 1100 1001
        RL      A               ; A = 0010 0111
```

Ví dụ 7.13:

Viết một chương trình để tìm số các số 1 trong một byte đã cho.

Lời giải:

```
        MOV     R1, #0          ; Chọn R1 giữ số các số 1
        MOV     R7, #8          ; Đặt bộ đếm  = 8 để quay 8 lần
        MOV     A, #97H         ; Tim các số 1 trong byte 97H
AGAIN:  RLC     A               ; Quay trái có nhớ một lần
        JNC     NEXT            ; Kiểm tra cờ CY
        INC     R1              ; Nếu CY = 1 thì cộng 1 vào bộ đếm
NEXT:   DJNZ    R7, AGAIN       ; Lặp lại quá trình 8 lần
```

Để truyền 1 byte dữ liệu nói tiếp thì dữ liệu có thể được chuyển đổi từ song song sang nối tiếp bằng các lệnh quay như sau:

```
RRC     A               ; Bít thứ nhất đưa vào cờ CY
MOV     P1.3, C         ; Xuất CY như một bit dữ liệu
RRC     A               ; Bit thứ hai đưa vào CY
```

```
MOV     P1.3, C         ; Xuất CY ra như một bit dữ liệu
RRC     A               ;
MOV     P1.3, C         ;
...
```

Đoạn mã trên đây là một phương pháp được sử dụng rộng rãi trong truyền dữ liệu tới các bộ nhớ nối tiếp như các EEPROM nối tiếp.

7.3 Các chương trình ứng dụng của mã BCD và ASCII.

Các số mã BCD đã được trình ở chương 6. Như đã nói ở đó rằng trong rất nhiều bộ vi điều khiển mới đều có một đồng hồ thời gian thực RTC (Real Time Clock) để giữ cho thời gian và cả lịch cho cả khi bị tắt nguồn. Các bộ vi điều khiển này cung cấp thời gian và lịch dưới dạng BCD. Tuy nhiên, để hiển thị chúng thì chúng phải được chuyển về mã ASCII. Trong phần này ta trình bày ứng dụng của các lệnh quay và các lệnh lô-gíc trong việc chuyển đổi mã BCD và ASCII.

Bảng 7.2: Mã ASCII cho các chữ số từ 0- 9.

Phím	Mã ASCII (Hex)	Mã ASCII nhị phân	Mã BCD (không đóng gói)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

7.3.1 Các số mã ASCII.

Trên các bàn phím ASCII khi phím "0" được kích hoạt thì "011 0001" (30H) được cấp tới máy tính. Tương tự như vậy 31H (011 0001) được cấp cho phím "1" v.v... như cyhỉ ra trong bảng 7.2.

Cần phải ghi nhớ rằng mặc dù mã ASCII là chuẩn ở mỹ (và nhiều quốc gia khác) nhưng các số mã BCD là tổng quát. Vì bàn phìm, máy in và màn hình đều sử dụng mã ASCII nên cần phải thực hiện đổi chuyển giữa các số mã ASCII về số mã BCD và ngược lại.

7.3.2 Chuyển đổi mã BCD đóng gói về ASCII.

Các bộ vi điều khiển DS5000T đều có đồng bộ thời gian thực RTC. Nó cung cấp hiển thị liên tục thời gian trong ngày (giờ, phút và giây) và lịch (năm, tháng, ngày) mà không quan tâm đến nguồn tắt hay bật. Tuy nhiên dữ liệu này được cấp ở dạng mã BCD đóng gói. Để hiển thị dữ liệu này trên một LCD hoặc in ra trên máy in thì nó phải được chuyển về dạng mã ASCII.

Để chuyển đổi mã BCD đóng gói về mã ASCII thì trước hết nó phải được chuyển đổi thành mã BCD không đóng gói. Sau đó mã BCD chưa đóng gói được móc với 011 0000 (30H). Dưới đây minh hoạ việc chuyển đổi từ mã BCD đóng gói về mã ASCII. Xem ví dụ 7.14.

Mã BCD đóng gói	Mã BCD không đóng gói	Mã ASCII
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 & 0000 1001	0011 0010 & 0011 1001

7.3.3 Chuyển đổi mã ASCII về mã BCD đóng gói.

Để chuyển đổi mã ASCII về BCD đóng gói trước thì trước hết nó phải được chuyển về mã BCD không đóng gói (để có thêm 3 số) và sau đó được kết hợp để tạo ra mã SCD đóng gói. Ví dụ số 4 và số 7 thì bàn phím nhận được 34 và 37. Mục tiêu là tạo ra số 47H hay "0100 0111" là mã BCD đóng gói. Qúa trình này như sau:

Phím	Mã ASCII	Mã BCD không đóng gói	Mã BCD đóng gói
4	34	0000 0100	
7	37	0000 0111	0100 0111 hay 47H

```
MOV     A, # '4'        ; Gán A = 34H mã ASCII của số 4
MOV     R1, # '7'       ; Gán R1 = 37H mã ASCII của số 4
ANL     A, #0FH         ; Che nửa byte cao A (A = 04)
ANL     R1, #0FH        ; Che nửa byte cao của R1 (R1 = 07)
SWAP    A               ; A = 40H
ORL     A, R1           ; A = 47H, mã BCD đóng gói
```

Sau phép chuyển đổi này các số BCD đóng gói được xử lý và kết quả sẽ là dạng BCD đóng gói. Như ta đã biết ở chương 6 có một lệnh đặt biệt là "DA A" đòi hỏi dữ liệu phải ở dạng BCD đóng gói.

Ví dụ 7.14:

Giả sử thanh ghi A có số mã BCD đóng gói hãy viết một chương trình để chuyển đổi mã BCD về hai số ASCII và đặt chúng vào R2 và R6.

Lời giải:

```
MOV     A, #29H         ; Gán A = 29, mã BCD đóng gói
MOV     R2, A           ; Giữ một bản sao của BCD trong R2
ANL     A, #0FH         ; Che phần nửa cao của A (A = 09)
ORL     A, #30H         ; Tạo nó thành mã ASCII A = 39H (số 9)
MOV     R6, A           ; Lưu nó vào R6 (R6 = 39H ký tự của ASCII)
MOV     A, R2           ; Lấy lại giá trị ban đầu của A (A = 29H)
ANL     A, #0F0H        ; Che nửa byte phần thấp của A (A = 20)
RR      A               ; Quay phải
RR      A               ; Quay phải
RR      A               ; Quay phải
RR      A               ; Quay phải (A = 02)
ORL     A, #30H         ; Tạo nó thành mã ASCII (A = 32H, số 2)
MOV     R2, A           ; Lưu ký tự ASCII vào R2
```

Trong ví dụ trên tất nhiên là ta có thể thay 4 lệnh RR quay phải bằng một lệnh trao đổi WAPA.

CHƯƠNG 8
Các lệnh một bít và lập trình

8.1 Lập trình với các lệnh một bít.

Trong hầu hết các bộ vi xử lý (BVXL) thì dữ liệu được truy cập theo từng byte. Trong các bộ vi xử lýnh địa chỉ theo byte này thì các nội dung của một thanh ghi, bộ nhớ RAM hay cổng đều phải được truy cập từng byte một. Hay nói cách khác, lượng dữ liệu tối thiểu có thể được truy cập là một byte. Ví dụ, trong bộ vi xử lý Pentium cổng vào/ ra (I/O) được định hướng theo byte, có nghĩa là để thay đổi một bít thì ta phải truy cập toàn bộ 8 bít. Trong khi đó có rất nhiều ứng dụng thì ta phải chỉ cần thay đổi giá trị của một bít chẳng hạn như là bật hoặc tắt một thiết bị. Do vậy khả năng đánh địa chỉ đến từng bít của 8051 rất thích hợp cho ứng dụng này. Khả năng truy cập đến từng bít một thay vì phải truy cập cả byte làm cho 805 trở thành trong những bộ vi điều khiển (BVĐK) 8 bít mạnh nhất trên thị trường. Vậy những bộ phận nào của CPU, RAM, các thanh ghi, cổng I/O hoặc ROM là có thể đánh địa chỉ theo bít được. Vì ROM chỉ đơn giản dữ mã chương trình thực thi nên nó không cần khả năng đánh địa chỉ theo bít. Tất cả mọi mã lệnh đều định hướng theo byte chỉ có các thanh ghi, RAM và các cổng I/O là cần được đánh địa chỉ theo bít. Trong 8051 thì rất nhiều vị trí của RAM trong một số thanh ghi và tất cả các cổng I/O là có thể đánh địa chỉ theo từng bít. Dưới đây ta chỉ đi sâu vào từng phần một.

8.1.1 Các lệnh một bít.

Các lệnh dùng các phép tính một bít được cho ở bảng 8.1. Trong phần này chúng ta làm về các lệnh này và đưa ra nhiều ví dụ về cách sử dụng chúng, các lệnh một bít khác mà chỉ liên quan đến cờ nhớ CY (Cary Flag) sẽ làm ở mục khác.

Bảng 8.1: Các lệnh một bít của 8051

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

8.1.2 Các cổng I/O và khả năng đánh địa chỉ theo bít.

Bộ vi điều khiển 8051 có bốn cổng I/O 8 bít là P0, P1, P2 và P3. Chúng ta có thể truy cập toàn bộ 8 bít hoặc theo một bít bất kỳ mà không làm thay đổi các bít khác còn lại. Khi truy cập một cổng theo từng bít, chúng ta sử dụng các cú pháp "SETB Y, Y" với X là số của cổng 0, 1, 2 hoặc 3, còn Y là vị trí bít từ 0 đến 7 đối với các bít dữ liệu đo đến 7. Ví dụ "SETB P1.5" là thiết lập bít cao số 5 của cổng 1. Hãy nhớ rằng do là bít có nghĩa thấp nhất LSB và D7 là bít có nghĩa là cao nhất MSB. Xem ví dụ 8.1.

Ví dụ 8.1: Viết các chương trình sau:

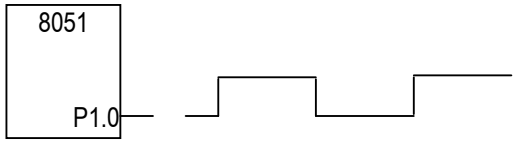
a) Tạo một sóng vuông (hàm xung vuông) với độ đầy xung 50% trên bít 0 của cổng 1.

b) Tạo một hàm xung vuông với 66% độ đầy xung trên bít 3 của cổng 1.

Lời giải:

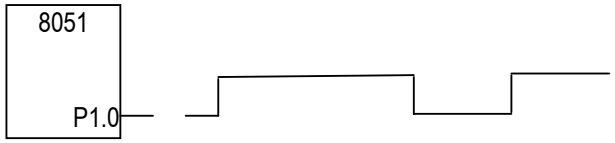
a) Hàm xung vuông với độ đầy xung 50% có nghĩa là trạng thái "bật" và "tắt" (hoặc phần cao và thấp của xung) có cùng độ dài. Do vậy ta chốt P1.0 với thời gian giữ chậm giữa các trạng thái.

```
HERE: SETB   P1.0          ;Thiết lập bít 0 cổng 1 lên 1.
        LCALL DELAY        ;Gọi chương trình con giữ chậm DELAY
      CLR P1.0             ;P1.0 = 0
      SJMP HERE            ;Tiếp tục thực hiện nó.
                           Có thể viết chương trình này theo cách khác:
HERE: CPL    P1.0          ;Bù bít 0 của cổng 1.
      LCALL DELAY          ;Gọi chương trình con giữ chậm DELAY
      SJMP   HERE          ;Tiếp tục thực hiện nó.
```



b) Hàm xung vuông với độ đầy xung 66% có nghĩa là trạng thái "bật" có độ dài gấp đôi trạng thái "tắt".

```
BACK: SETB  P1.3           ;Thiết lập bít 3 cổng 1 lên 1.
      LCALL DELAY          ;Gọi chương trình con DELAY
      LCALL DELAY          ;Gọi chương trình con DELAY lần nữa.
      CLR    P1.3          ;Xoá bít 3 của cổng 1 và 0.
      LCALL DELAY          ;Gọi chương trình con DELAY
      SJMP BACK            ;Tiếp tục thực hiện nó.
```



Lưu ý rằng, khi mã "P1.0" được hợp dịch nó trở thành "SETB 90H" vì P1.0 có địa chỉ trong RAM là 90h. Từ hình vẽ 8.1 ta thấy rằng các địa chỉ bít cho P0 là 80H đến 87H và cho P là 90H đến 97H v.v... Hình 8.1 cũng chỉ ra tất cả các thanh ghi có khả năng đánh địa chỉ theo bít.

Bảng 8.2: Khả năng đánh địa chỉ theo bít của các cổng.

P0	P1	P2	P3	Port's Bit
P0.0	P1.0	P2.0	P3.0	D0
P0.1	P1.1	P2.1	P3.1	D1
P0.2	P1.2	P2.2	P3.2	D2
P0.3	P1.3	P2.3	P3.3	D3
P0.4	P1.4	P2.4	P3.4	D4
P0.5	P1.5	P2.5	P3.5	D5
P0.6	P1.6	P2.6	P3.6	D6
P0.7	P1.7	P2.7	P3.7	D7

Ví dụ 8.2:

Đối với các lệnh dưới đây thì trạng thái của bít nào của SFR sẽ bị tác động (hãy sử dụng hình 8.1).

a) SETB 86H, b) CLR 87H, c) SETB 92H

b) SETB DA7H, e) CLR 0F2H, f) SETB OE7H

Lời giải

a) SETB	86H là dành cho SETB P0.6
b) CLR	87H là dành cho CLR P0.7
c) SETB	92H là dành cho SETB P1.2
d) SETB	0A7H là dành cho SETB P2.7
e) CLR	0F2H là dành cho CLR D2 của thanh ghi B
f) SETB	0E7H là dành cho SETB ACC.7 (bít D7 của thanh ghi A)

8.1.3 Kiểm tra một bít đầu vào.

Lệnh JNB (nhảy nếu bít = 0) và JB (nhảy nếu bít bằng 1) cũng là các phép thao tác đơn bít được sử dụng rộng rãi. Chúng cho phép ta hiển thị một bít và thực hiện quyết định phụ thuộc vào việc liệu nó là 0 hay là 1.

Ví dụ 8.3: Giả sử bít P2.3 là một đầu vào và biểu diễn điều kiện của một lô. Nếu nó bật lên 1 thì có nghĩa là lô nóng. Hãy hiển thị liên tục, mỗi khi nó lên cao thì hãy gửi một xung cao-xuống-thấp (Aigh-to-low) đến cổng P1.5 để bật còi báo.

Lời giải:

```
HERE:   JNB   P2.3, HERE      ; Duy trì hiển thị cao.
        SETB P1.5             ; Thiết lập P1.5 = 1
        CLR   P1.5            ; Thực hienẹ chuyển xung từ cao-xuống-thấp
```

Các lệnh JNB và JB có thể được dùng đối với các bít bất kỳ của các cổng I/O 0, 1, 2 và 3 vì tất cả các cổng này đều có khả năng đánh địa chỉ theo bít. Tuy nhiên, cổng 3 hầu như để dùng cho các tín hiệu ngắt và truyền thông nối tiếp và thông thường không dùng cho bất cứ vào/ ra theo bít hoặc theo byte nào. Điều này sẽ được bàn ở chương 10 và 11.

8.1.4 Các thanh ghi và khả năng đánh địa chỉ theo bít.

Trong tất cả các cổng I/O đều có khả năng đánh địa chỉ theo bít thì các thanh ghi lại không được như vậy. Ta có thể nhìn thấy điều đó từ hình 8.1: Chỉ thanh gh B, PSW, IP, IE, ACC, SCON và TCON là có thể đánh địa chỉ theo bít, ở đây ta sẽ tập trung vào các thanh ghi A, B và PSW còn các thanh ghi khác sẽ đề cập ở các chương sau. Từ hình 8.1 hãy để ý rằng cổng PO được gán địa chỉ bít 80H-87H. Còn đại chỉ bít 88-8FH được gán cho thanh ghi TCON.

Cuối cùng địa chỉ bít F0-F7H được gán cho thanh ghi B. Xét ví dụ 8.4 và 8.5 về việc sử dụng các thanh ghi này với khả năng đánh địa chỉ theo bít.

Byte address	Bit address								
FF									
F0	E7	F6	F5	F4	F3	F2	F1	F0	B
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
D0	D7	D6	D5	D4	D3	D2	D1	D0	PSW
B8	- -	- -	- -	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	F3
A8	AF	- -	- -	AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	not bit addressable								SBUF
98	9F	9E	9D	9C	9B	9A	99	99	SCON
90	97	96	95	94	93	92	91	90	P1
8D	not bit addressable								TH1
8C	not bit addressable								TH0
8B	not bit addressable								TL1
8A	not bit addressable								TL0
89	not bit addressable								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	not bit addressable								PCON
83	not bit addressable								DPH
82	not bit addressable								DPL
81	not bit addressable								SP
80	87	86	85	84	83	82	81	80	P0

Special Function Registers

Hình 8.1: Địa chỉa theo Byte và bít của bộ nhớ RAM các thanh ghi chức năng đặc biệt.

Ví dụ 8.4: Hãy viết chương trình để kiểm tra xem thanh ghi tích luỹ có chứa một số chẵn không? Nếu có thì chia nó cho 2, nếu không thì hãy làm chẵn nó và sau đó chia nó cho 2.

Lời giải:

```
MOV  B, # 2            ; Gán B = 2
JNB  ACC 0, YES        ; DO của thanh ghi A có bằng 0?
JNC  A                 ; Nếu có thì nhảy về YES
```

```
YES:    DIX   AB          ; Nếu là sô lẻ thì tăng lên 1 để thành chẵn
                          ; Chia A/B
```

Ví dụ 8.5: Hãy viết đoạn chương trình để kiểm tra xem các bít 0 và 5 của thanh ghi B có giá trị cao không? Nếu không phải thì đặt chúng lên 1 và lưu vào thanh ghi bộ.
Lời giải:

```
        JNB   OFOH, NEXT-1      ; Nhảy về NEXT-1 nếu B.0 = 0
        SETB OFOH               ; Đặt B.0 = 1
NEXT-1: JNB     OF5H, NEXT-2    ; Nhảy về NEXT-2 nếu B.5 = 0
        SETB OF5H               ; Đặt B.5 = 1
NEXT-2: MOV     RO, B           ; Cắt thanh ghi B
```

CY	AC	- -	RS1	RS0	OV	- -	P

RS1	**RS0**	**Register Bank**	**Address**
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

Hình 8.2: Các bít của thanh ghi PSW.

Như đã nói ở chương 2, trong than ghi PSW có hai bít dành riêng để chọn các bảng thanh ghi. Khi RESET thì bằng 0 được chọn, chúng ta có thể chọn các băng bất kỳ khác bằng cách sử dụng khả năng đánh địa chỉ theo bít của PSW.
Ví dụ 8.6: Hãy viết chương trình để lưu thanh ghi tích luỹ vào R7 của băng 2.
Lời giải:

```
CLR     PSW.3
SETB    PSW.4
MOV     R7.A
```

Ví dụ 8.7: Trong khi có hai lệnh JNC và JC để kiểm tra bít cờ nhớ CY thì lại không có các lệnh cho bít cờ tràn (0V) làm thế nào để ta có thể viết mã kiểm tra 0V.
Lời giải: Cờ 0V là bít PSW.2 của thanh ghi PSW. PSW là thanh ghi có thể đánh địa chỉ theo bít, do vậy ta có thể sử dụng lệnh sau để kiểm tra cờ 0V:

```
JB      PSW.2, TARGET       ; Nhảy về TARGET nếu 0V = 1
```

8.15 Vùng nhớ RAM có thể đánh địa chỉ theo bít.

Trong 128 byte RAM trong của 8051 thì chỉ có 16 byte của nó là có thể đánh địa chỉ theo bít được. Phần còn lại được định dạng byte. Các cùng RAM có thể đánh địa chỉ theo bít là 20H đến 2FH. Với 16 byte này của RAM có thể cung cấp khả năng đánh địa chỉ theo bít là 128 bít, vì $16 \times 8 = 128$. Chúng được đánh địa chỉ từ 0 đến 127. Do vậy, những địa chỉ bít từ 0 đến 7 dành cho byte đầu tiên, vị trí RAM trong 20H và các bít từ 8 đến OFH là địa chỉ bít của byte thứ hai của vị trí RAM trong 21H v.v... Byte cuối cùng của 2FH có địa chỉ bít từ 78H đến 7FH (xem hình 8.3). Lưu ý rằng các vị trí RAM trong 20H đến 2FH vừa có thể đánh địa chỉ theo byte vừa có thể đánh địa chỉ theo bít.

Để ý từ hình 8.3 và 8.1 ta thấy rằng các địa chỉ bít 00 - 7FH thuộc về các địa chỉ byte của RAM từ 20 - 2FH và các địa chỉ bít từ 80 đến F7H thuộc các thanh ghi đặc biệt SFR, các cổng P0, P1, v.v...
Ví dụ 8.8: Hãy kiểm tra xem các bít sau đây thuộc byte nào? Hãy cho địa chỉ của byte RAM ở dạng Hex.

```
a) SETB    42H          ; Set bit 42H to 1
b) CLR     67H          ; Clear bit 67
c) CLR     0FH          ; Clear bit OFH
d) SETB    28H          ; Set bit 28H to 1
e) CLR     12           ; Clear bit 12 (decimal)
f) SETB    05
```

Lời giải:

a) Địa chỉ bít 42H của RAM thuộc bít D2 của vị trí RAM 28H.
b) Địa chỉ bít 67H của RAM thuộc bít D7 của vị trí RAM 20H.
c) Địa chỉ bít 0FH của RAM thuộc bít D7 của vị trí RAM 21H.
d) Địa chỉ bít 28H của RAM thuộc bít D0 của vị trí RAM 25H.
e) Địa chỉ bít 12H của RAM thuộc bít D4 của vị trí RAM 21H.
f) Địa chỉ bít 05H của RAM thuộc bít D5 của vị trí RAM 20H.

Ví dụ 8.9: Trạng thái của các bít P1.2 và P1.3 của cổng vào/ra P1 phải được lưu cất trước khi chúng được thay đổi. Hãy viết chương trình để lưu trạng thái của P1.2 vào vị trí bít 06 và trạng thái P1.3 vào vị trí bít 07.

Lời giải:

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

Các câu hỏi ôn luyện:

1. Tất cả các cổng I/O của 8051 đều có khả năng đánh địa chỉ theo bít? (đúng sai)
2. Tất cả mọi thanh ghi của 8051 đều có khả năng đánh địa chỉ theo bít? (đúng sai)
3. Tất cả các vị trí RAM của 8051 đều có khả năng đánh địa chỉ theo bít? (đúng sai)
4. Hãy chỉ ra những thanh ghi nào sau đây có khả năng đánh địa chỉ theo bít:
   a) A, b) B, (c) R4 (d) PSW (e) R7
5. Trong 128 byte RAM của 8051 những byte nào có khả năng đánh địa chỉ theo bít. Hãy liệt kê chúng.
6. Làm thế nào để có thể kiểm tra xem bít D0 của R3 là giá trị cao hay thấp.
7. Hãy tìm xem các bít dau thuộc những byte nào? Hãy cho địa chỉ của các byte RAM theo số Hex:
   a) SETB 20 b) CLR 32 c) SETB 12H
   d) SETB 95 e) SETB 0ETB 12H
8. Các địa chỉ bít 00 - 7FH và 80 - F7H thuộc các vị trí nhớ nào?
9. Các cổng P0, P1, P2 và P3 là một bộ phận của SFR? (đúng sai)
10. Thanh ghi TCON có thể đánh địa chỉ theo bít (đúng sai)

8.2 Các phép toán một bít với cờ nhớ CY.

Ngoài một thực tế là cờ nhớ CY được thay đổi bởi các lệnh lô-gíc và số học thì trong 8051 còn có một số lệnh mà có thể thao tác trực tiếp cờ nhớ CY. Các lệnh này được cho trong bảng 8.3.

Trong các lệnh được chỉ ra sau trong bảng 8.3 thì chúng ta đã trình bày công dụng của lệnh JNC, CLR và SETB trong nhiều ví dụ trong một số chương trước đây. Dưới đây ta tiếp tục làm quen với một số ví dụ về cách sư dụng một số lệnh khác từ bảng 8.3.

Một số lệnh cho trong bảng 8.3 làm việc với các phép toán lô-gíc AND và OR. Các ví dụ ở mục này sẽ chỉ ra cách sử dụng chúng như thế nào?

Ở chương tiếp theo chúng ta sẽ chỉ ra nhiều ví dụ hơn về việc sử dụng của các lệnh đơn trong phạm vi các ứng dụng thực tế.

Bảng 8.3: Các lệnh liên quan đến cờ nhớ CY

Lệnh	chức năng
SETB C	Thực hiện (tao) CY = 1
CLR C	Xoá bít nhớ CY = 0
CPL C	Bù bít nhớ
MOV b, C	Sao chép trạng thái bít nhớ vào vị trí bít b = CY
MOV C, b	Sao chép bít b vào trạng thái bít nhớ CY = b
JNC đích	Nhảy tới đích nếu CY = 0
JC đích	Nhảy tới đích nếu CY = 1
ANL C. bít	Thực hiện phép AND với bít b và lưu vào CY
ANL C./ bít	Thực hiện phép AND với bít đảo và lưu vào CY
ORL C. bít	Thực hiện phép OR với bít và lưu vào CY
ORL C./ bít	Thực hiện phép OR với bít đảo và lưu vào CY

Ví dụ 8.10: Hãy viết một chương trình để lưu cất trạng thái của các bít P1.2 và P1.3 vào vị trí nhớ tương ứng trong RAM 6 và 7.

Lời giải:

```
MOV  C,  P1.2    ; Lưu trạng thái P1.2 vào CY.
MOV  06, C       ; Lưu trạng thái CY vào bít 6 của RAM
MOV  C, P1.3     ; Lưu trạng thái P1.2 vào CY
MOV  07, C       ; Lưu trạng thái CY vào vị trí RAM 07
```

Ví dụ 8.11:

Giả sử vị trí nhớ 12H trong RAM giữ trạng thái của việc có điện thoại hay không. Nếu nó ở trạng thái cao có nghĩa là đã có một cuộc gọi mới vì nó được kiểm tra lần cuối. Hãy viết một chương trình để hiển thị "có lời nhắn mới" ("New Message") trên màn hình LCD nếu bít 12H của RAM có giá trị cao. Nếu nó có giá trị thấp thì LCD hiển thị "không có lời nhắn mới" ("No New Message").

Lời giải:

```
        MOV  C, 12H          ; Sao trạng thái bít 12H của RAM vào CY
        JNC   NO             ; Kiểm tra xem cờ CY có giá trị cao không.
        MOV  DPTR, # 400H    ; Nếu nó nạp địa chỉ của lời nhắn.
        LCAL DISPLAY         ; Hiển thị lời nhắn.
        SJMP  NEXT           ; Thoát
NO:     MOV  DSTR, #420H     ; Nạp địa chỉ không có lời nhắn.
        LCAL DISPLAY         ; Hiển thị nó.
EXIT:                          Thoát
;  ____________________ data to be displayed on LCD
                ORG  400H
YES-MG:         DB   "NEW Message"
                ORG  420H
NO-MG:          DB   "No New Message"
```

Ví dụ 8.12:

Giả sử rằng bít P2.2 được dùng để kiểm tra đèn ngoài và bít P2.5 dùng để kiểm tra đèn trong của một toà nhà. Hãy trình bày làm thế nào để bật đèn ngoài và tắt đèn trong nhà.

Lời giải:

```
SETB    C           ; Đặt CY = 1
ORL     C, P2.2, C  ; Thực hiện phép OR với CY
MOV     P2.2, C     ; Bật đèn nếu nó chưa bật.
CLR     C           ; Xoá CY = 0
ANL     C, P2.5     ; CY = (P2.5 AND CY)
MOV     P2.5, C     ; Tắt nó nếu nó chưa tắt.
```

Câu hỏi ôn luyện:

1. Tìm trạng thái của cờ CY sau đoạn mã dưới đây:

```
a) CLR      A           b) CLR   C          c) CLR   C
   ADD      A, #OFFH     JNC     OVER        JC      OVER
   JWC      OVER        SETB  C              CPL     C
   CPL      C             OVER: ...           OVER: ...
OVER: ...
```

2. Hãy trình bày cách làm thế nào để lưu trạng thái bít P2.7 vào vị trí bít 31 của RAM.
3. Hãy trình bày các chuyển trạng thái bít 09 của RAM đến bít P1.4.

8.3 Đọc các chân đầu vào thông qua chốt 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 chân vào.
2. Đọc chốt trong của cổng ra.

Chúng ta phải phân biệt giữa hai dạng lệnh này vì sự lẫn lộn giữa chúng là nguyên nhân chính của các lỗi trong lập trình cho 8051, đặc biệt khi đã kết nối với phần cứng bên ngoài. Trong phần này ta bàn về sơ qua các lệnh này. Tuy nhiên, độc giả phải nghiên cứu và hiểu về các nội dung của chủ đề này và về hoạt động bên trong của các cổng được cho trong phụ lục Appendix C2.

8.3.1 Các lệnh đọc cổng vào.

Như đã nói ở chương 4 thì để biến một bít bất kỳ của cổng 8051 nào đó thành một cổng đầu vào, chúng ta phải ghi (lô-gíc cao) vào bít đó. Ssu khi cấu hình các bít của cổng là đầu vào, ta có thể sử dụng những lệnh nhất định để nhận dữ liệu ngoài trên các chân vào trong CPU. Bảng 8.4 là những lệnh nói trên.

Bảng 8.4: Các lệnh đọc một cổng vào.

Giả 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

8.3.2 Đọc chốt cho cổng đầu ra.

Một số lệnh nội dung của một chốt cổng trong thay cho việc đọc trạng thái của một chân ngoài. Bảng 8.5 cung cấp danh sách những lệnh này. Ví dụ, xét lệnh "ANL P1, A". Trình tự thao tác được thực hiện bởi lệnh này như sau:

1. Nó đã chốt trong của một cổng và chuyển dữ liệu đó vào trong CPU.
2. Dữ liệu này được AND với nội dung của thanh ghi A.
3. Kết quả được ghi ngược lại ra chốt cổng.
4. Dữ liệu tại chân cổng được thay đổi và có cùng giá trị như chốt cổng.

Từ những bàn luận trên ta kết luận rằng, các lệnh đọc chốt cổng thường đọc một giá trị, thực hiện một phép tính (và có thể thay đổi nó) sau đó ghi ngược lại ra chốt cổng. Điều này thường được gọi "Đọc-sửa-ghi", ("Read-Modify-Write"). Bảng 8.5 liệt kê các lệnh đọc-sửa-ghi sử dụng cổng như là toán hạng đích hay nói cách khác, chúng ta chỉ được dùng cho các cổng được cấu hình như các cổng ra.

Bảng 8.5: Các lệnh đọc một chốt (Đọc-sửa-ghi).

giả 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

Lưu ý: Chúng ta nên nghiên cứu phần C2 của phụ lục Appendix C nếu ta nối phần cứng ngoài vào hệ 8051 của mình. Thực hiện sai các chỉ dẫn hoặc nối sai các chân có thể làm hỏng các cổng của hệ 8051.

8.4 Tóm lược.

Chương này đã mô tả một trong các đặc tính mạnh nhất của 8051 là phép toán một bít. Các phép toán một bít này cho phép lập trình viên thiết lập, xoá, di chuyển và bù các bít riêng rẽ của các cổng, bộ nhớ hoặc các thanh ghi.

Ngoài ra có một số lệnh cho phép thao tác trực tiếp với cờ nhớ CY. Chúng ta cũng đã bàn về các lệnh đọc các chân cổng thông qua việc đọc chốt cổng.

8.5 Các câu hỏi kiểm tra.

1. Các lệnh "SETB A", "CLR A", "CPL A" đúng hay sai?
2. Các cổng vào/ ra nào và các thanh ghi nào có thể đánh địa chỉ theo bít.
3. Các lệnh dưới đây đúng hay sai? Đánh dấu lệnh đúng.

a) SETB P1 e) SETB B4
b) SETB P2.3 f) CLR 80H
c) CLR ACC.5 g) CLR PSW.3
d) CRL 90H h) CLR 87H

4. Hãy giết chương trình tạo xung vuông với độ đầy xung 75%, 80% trên các chân P1.5 và P2.7 tương ứng.
5. Viết chương trình hiển thị P1.4 nếu nó có giá trị cao thì chương trình tạo ra một âm thanh (sóng dung vuông 50% độ đầy xung) trên chân P2.7.
6. Nhưng địa chỉ bít nào được gán cho các cổng P0, P1, P2 và P3 cho các thanh ghi PCON, A, B và PSW.

7. Những địa chỉ bít dưới đây thuộc về cổng hay thanh ghi nào?

a) 85H b)87H c) 88H d)8DH e)93H

f) A5H g)A7H h) B3H i) D4H j) D8H

8. Hãy viết chương trình lưu các thanh ghi A, B vào R3 và R5 băng nhớ 2 tương ứng.

9. Cho một lệnh khác cho "CLR C", so sánh chúng.

10. Làm thế nào để kiểm tra trạng thái các bít cờ OV, CY, P và AC. Hãy tìm địa chỉ bít của các cờ này.

11. Các cùng nhớ 128 byte của RAM thì những vùng nào là đánh địa chỉ theo bít được? Hãy đánh dấu chúng.

12. Các địa chỉ sau thuộc vùng RAM nào?

a) 05H b) 47 c) 18H d) 2DH e) 53H

g) 15H h) 67H h) 55H i) 14H k) 37FH

13. Các địa chỉ nhỏ hơn 80H được gán cho địa chỉ 20-2FH của RAM phải không? (Đúng/ sai).

14. Viết các lệnh để lưu cờ CY, AC, D vào vị trí bít 4, 16H và 12H tương ứng.

15. Viết chương trình kiểm tra D7 của thanh ghi A. Nếu D7 = 1 thì gửi thông báo sang LCD báo rằng ACC có một số âm.

CHƯƠNG 9
Lập trình cho bộ đếm/ bộ định thời trong 8051

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 BVĐK. Trong chương này chúng ta sẽ tìm hiểu về cách lập trình cho chúng và sử dụng chúng như thế nào?

9.1 Lập trình các bộ định thời gian của 8051.

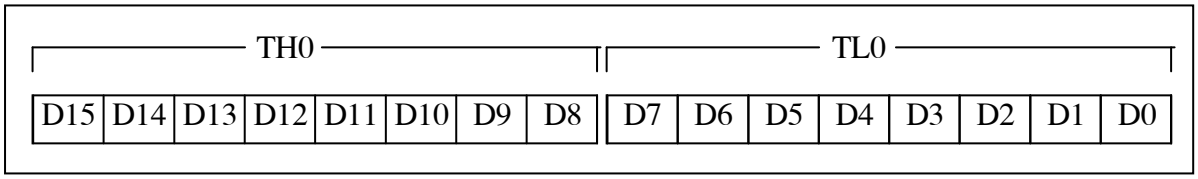
8051 có hai bộ định thời là Timer 0 và Timer1, ở phần này chúng ta bàn về các thanh ghi của chúng và sau đó trình bày cách lập trình chúng như thế nào để tạo ra các độ trễ thời gian.

9.1.1 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.

9.1.1.1 Các thanh ghi của bộ Timer 0.

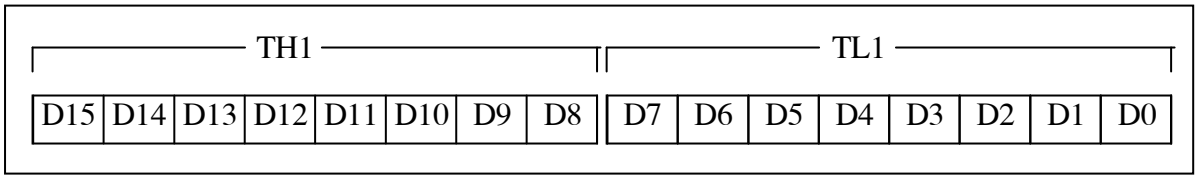
Thanh ghi 16 bít của bộ Timer 0 được truy cập như byte thấp và byte cao. Thanh ghi byte thấp được gọi là TL0 (Timer 0 bow byte) và thanh ghi byte cao là TH0 (Timer 0 High byte). Các thanh ghi này có thể được truy cập như mọi thanh ghi khác chẳng hạn như A, B, R0, R1, R2 v.v... Ví dụ, lệnh "MOV TL0, #4FH" 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.



Hình 9.1: Các thanh ghi của bộ Timer 0.

9.1.1.2 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.



Hình 9.2: Các thanh ghi của bộ Timer 1.

9.1.2 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.

(MSB) (MSB)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer1				Timer0			

Hình 9.3: Thanh ghi IMOD.

9.1.2.1 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

9.1.2.2 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 9.1: Hãy hiển thị xem chế độ nào và bộ định thời nào đối với các trường hợp sau:

a) MOV TMOD, #01H b) MOV TMOD, #20H c) MOV TMDO, #12H

Lời giải: Chúng ta chuyển đổi giá trị từ số Hex sang nhị phân và đối chiếu hình 93 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.

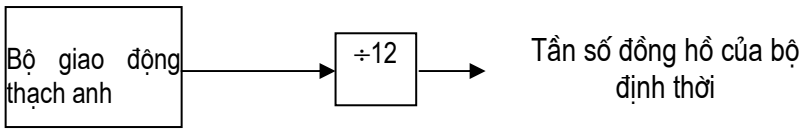
9.1.2.3 Nguồn xung đồng hồ cho bộ định thời:

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. Xem ví dụ 9.2.

Ví dụ 9.2:

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) $\frac{1}{12}\times 12\text{MHz} = 1\text{MHz}$ và $T = \frac{1}{1/1\text{MHz}} = 1\mu\text{s}$

b) $\frac{1}{12}\times 16\text{MHz} = 1{,}111\text{Mz}$ và $T = \frac{1}{1{,}333\text{MHz}} = 0{,}75\mu\text{s}$

c) $\frac{1}{12}\times 11{,}0592\text{MHz} = 921{,}6\text{kHz}$ và $T = \frac{1}{0{,}9216\text{MHz}} = 1{,}085\mu\text{s}$

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 hải làm việc với tần suất bouid 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, điều mà ta sẽ biết ở chương 10.

9.1.3 Bít cổng GATE.

Một bít khác của thanh ghi TMOD là bít cổng GATE. Để ý trên hình 9.3 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 tren 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 TRO" 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". Việc sử dụng phần cứng ngoài để khởi động và dừng bộ định thời ta sẽ bàn ở chương 11 khi bàn về các ngắt.

Ví dụ 9.3:

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:

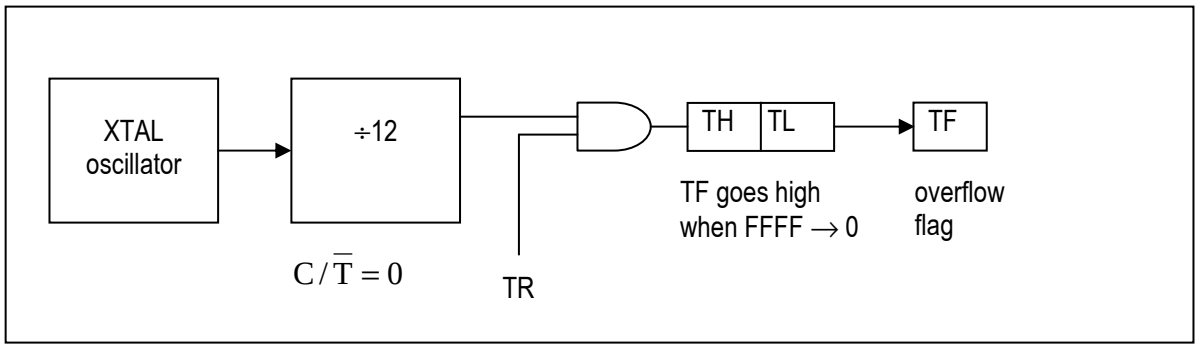
TMOD = 0000 0010: Bộ định thời Timer0, chế độ 2 C/T = 0 dùng nguồn XTAL GATE = 0 để dùng phần mềm trong để khởi động và dừng bộ định thời.

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.

9.1.4 Lập trình cho mỗi chế độ Mode1.

Dưới đây là những đặc tính và những phép toán của chế độ Mode1:

1. Nó là bộ định thời 16 bít, do vậy nó cho phép các giá trị 0000 đến FFFFH được nạp vào các thanh ghi TL và TH của bộ định thời.
2. Sau khi TL và TH được nạp một giá trị khởi tạo 16 bít thì bộ định thời phải được khởi động. Điều này được thực hiện bởi "SETB TR0" đối với Timer 0 và "SETB TR1" đối với Timer1.
3. Sau khi bộ định thời được khởi động, nó bắt đầu đếm lên. Nó đếm lên cho đến khi đạt được giới hạn FFFFH của nó. Khi nó quay qua từ FFFFH về 0000 thì nó bật lên bít cờ TF được gọi là cờ bộ định thời. Cờ bộ định thời này có thể được hiển thị. Khi cờ bộ định thời này được thiết lập từ một trong các phương án để dừng bộ định thời bằng các lệnh "CLR TR0" đối với Timer0 hoặc "CLR TR1" đối với Timer1. ở đây cũng cần phải nhắc lại là đối với bộ định thời đều có cờ TF riêng của mình: TF6 đối với Timer0 và TF1 đối với Timer1.



4. Sau khi bộ định thời đạt được giới hạn của nó và quay quan giá trị FFFFH, muốn lặp lại quá trình thì các thanh ghi TH và TL phải được nạp lại với giá trị ban đầu và TF phải được duy trì về 0.

9.1.4.1 Các bước lập trình ở chế độ Mode 1.

Để tạo ra một độ trễ thời gian dùng chế độ 1 của bộ định thời thì cần phải thực hiện các bước dưới đây.

1. Nạp giá trị TMOD cho thanh ghi báo độ định thời nào (Timer0 hay Timer1) được sử dụng và chế độ nào được chọn.
2. Nạp các thanh ghi TL và TH với các giáa trị đếm ban đầu.
3. Khởi động bộ định thời.
4. Duy trì hiển thị cờ bộ định thời TF bằng lệnh "JNB TFx, đích" để xem nó được bật không. Thoát vòng lặp khi TF được lên cao.
5. Dừng bộ định thời.
6. Xoá cờ TF cho vòng kế tiếp.
7. Quay trở lại bước 2 để nạp lại TL và TH.

Để tính toàn thời gian trễ chính xác và tần số sóng vuông được tạo ra trên chân P1.5 thì ta cần biết tần số XTAL (xem ví dụ 9.5).

Từ ví dụ 9.6 ta có thể phát triển một công thức tính toán độ trễ sử dụng chế độ Mode1 (16 bít) của bộ định thời đối với tần số thạch anh XTAL = 11, 0592MHz (xem hình 9.4). Máy tính trong thư mục Accessrry của Microsoft Windows có thể giúp ta tìm các giá trị TH và TL. Máy tính này hỗ trợ các phép tính theo số thập phân, nhị phân và thập lục.

a) Tính theo số Hex	b) Tính theo số thập phân
(FFFF - YYXX + 1). 1,085µs trong đó YYXX là các giá trị khởi tạo của TH, TL tương ứng. Lưu ý rằng các giá trị YYXX là theo số Hex.	Chuyển đổi các giá trị YYXX của TH, TL về số thập phân để nhận một số thập phân NNNNN sau đó lấy (65536 - NNNNN).1,085µs.

Hình 9.4: Công thức tính toán độ trễ thời gian đối với tần số XTAL = 11, 0592MHz.

Ví dụ 9.4:

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.

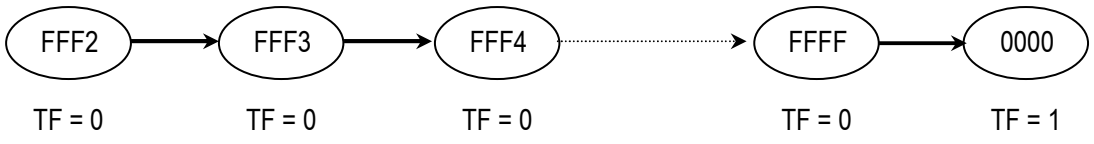
```
        MOV   TMOD, #01          ; Sử dụng Timer0 và chế độ 1(16 bít)
HERE:   MOV   TL0, #0F2H         ; TL0 = F2H, byte thấp
        MOV   TH0, #0FFH         ; TH0 = FFH, byte cao
        CPL   P1.5               ; Sử dụng chân P1.5
        ACALL DELAY
        SJMP  HERE               ; Nạp lại 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 9.5:

Trong ví dụ 9.4 hãy tính toán lượng thời gian trễ trong chương trình con DELAY được tạo ra bởi bộ định thời với giá thiết tần số XTAL = 11,0592MHz.

Lời giải:

Bộ định thời làm việc với tần số đồng hồ bằng 1/12 tần số XTAL, do vậy ta có $\frac{11,0592}{12} = 0,9216\text{MHz}$ là tần số của bộ định thời. Kết quả là mỗi nhịp xung đồng hồ có chu kỳ $T = \frac{1}{0,9216\text{MHz}} = 1,085\mu s$. Hay nói cách khác, bộ Timer0 đếm tăng sau 1,085μs để tạo ra bộ trễ bằng số đếm ×1,085μs.

Số đếm bằng FFFFH - FFF2H = ODH (13 theo số thập phân). Tuy nhiên, ta phải cộng 1 vào 13 vì cần thêm một nhịp đồng hồ để nó quay từ FFFFH về 0 và bật cờ TF. Do vậy, ta có 14 × 1,085μs = 15,19μs cho nửa chu kỳ và cả chu kỳ là T = 2 × 15,19μs = 30, 38μs là thời gian trễ được tạo ra bởi bộ định thời.

Ví dụ 9.6:

Trong ví dụ 9.5 hãy tính toán tần số của xung vuông được tạo ra trên chân P1.5.

Lời giải:

Trong tính toán độ thời gian trễ của ví dụ 9.5 ta không tính đến tổng phí của các lệnh trong vòng lặp. Để tính toán chính xác hơn ta cần bổ xung thêm các chu kỳ thời gian của các lệnh trong vòng lặp. Để làm điều đó ta sử dụng các chu kỳ máy từ bảng A-1 trong phụ lục Appendix A được chỉ dưới đây.

```
HERE:   MOV   TL0, #0F2H          2
        MOV   TH0, #0FFH          2
        CPL   P1-5                1
        ACALL DELAY               2
        SJMP HERE                 2
; _____ delay using timer0
DELAY:
        SETB  TR0                 1
AGAIN:  JNB   TF0, AGAIN          1
        CLR   TR0                 1
        CLR   TF0                 1
        RET                       1
                          Total   27
```

T = (2 × 27 × 1.085μs and F = 17067.75Hz).

Tổng số chu kỳ đã bổ xung là x7 nên chu kỳ thời gian trễ là T = 2 × 27 × 1.085μs = 58,59μs và tần số là F = 17067,75Hz.

Ví dụ 9.7:

Hãy tìm ra độ trễ được tạo ra bởi Timer0 trong đoạn mã sau sử dụng cả hai phương pháp của hình 9.4. Không tính các tổng phí của các lệnh.

```
        CLR   P2.3          ; Xoá P2.3
        MOV   TMOD, #01     ; Chọn Timer0, chế độ 1 (16 bít)
HERE:   MOV   TL0, #3EH     ; TL0 = 3EH, byte thấp
        MOV   TH0, #0B8G    ; TH0 = B8H, byte cao
        SETB  P2.3          ; Bật P2.3 lên cao
        SETB  TR0           ; Khởi động Timer0
AGAIN:  JNB   TF0, AGAIN    ; Hiển thị cờ bộ định thời TF0
```

```
        CLR     TR0             ; Dừng bộ định thời.
        CLR     TF0             ; Xoá cờ bộ định thời cho vòng sau
        CLR     P2.3
```

Lời giải:

a) Độ trễ được tạo ra trong mã trên là:

(FFFF - B83E + 1) = 47C2H = 18370 hệ thập phân 18370 × 1,085μs = 19, 93145μs.

b) Vì TH - TL = B83EH = 47166 (số thập phân) ta có 65536 - 47166 = 18370.

Điều này có nghĩa là bộ định thời gian đếm từ B83EH đến FFFF. Nó được cộng với một số đếm để về 0 thành một bộ tổng là 18370μs. Do vậy ta có 18370 × 1,085μs = 19,93145ms là độ rộng xung.

Ví dụ 9.8:

Sửa giá trị của TH và TL trong ví dụ 9.7 để nhận được độ trễ thời gian lớn nhất có thể. Hãy tính độ trễ theo miligiây. Trong tính toán cần đưa vào cả tổng phí của các lệnh.

Để nhận độ trễ thời gian lớn nhất có thể ta đặt TH và TL bằng 0. Điều này làm cho bộ định thời đếm từ 0000 đến FFFFH và sau đó quay qua về 0.

```
        CLR     P2.3            ; Xoá P2.3
        MOV     TMOD, #01       ; Chọn Timer0, chế độ 1 (16 bít)
HERE:   MOV     TL0, #0         ; Đặt TL0 = 0, byte thấp
        MOV     TH0, #0         ; Đặt TH0 = 0, byte cao
        SETB    P2.3            ; Bật P2.3 lên cao
        SETB    TR0             ; Khởi động bộ Timer0
AGAIN:  JNB     TF0, AGAIN      ; Hiển thị cờ bộ định thời TF0
        CLR     TR0             ; Dừng bộ định thời.
        CLR     TF0             ; Xoá cờ TF0
        CLR     P2.3
```

Thực hiện biến TH và TL bằng 0 nghĩa là bộ định thời đếm tăng từ 0000 đến FFFFH và sau đó quay qua về 0 để bật cờ bộ định thời TF. Kết quả là nó đi qua 65536 trạng thái. Do vậy, ta có độ trễ = (65536 - 0) × 1.085μs = 71.1065μs.

Trong ví dụ 9.7 và 9.8 chúng ta đã không nạp lại TH và TL vì nó là một xung đơn. Xét ví dụ 9.9 dưới đây để xem việc nạp lại làm việc như thế nào ở chế độ 1.

Ví dụ 9.9:

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 $35276 \times 1.085\mu s = 38274ms$ và tần số là $\frac{1}{38274}(Hz) = 26127Hz$.

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.

9.1.4.2 Tìm các giá trị cần được nạp vào bộ định thời.

giả sử rằng chúng ta biết lượng thời gian trễ mà ta cần thì câu hỏi đặt ra là làm thế nào để tìm ra được các giá trị cần thiết cho các thanh thi TH và TL. Để tính toán các giá trị cần được nạp vào các thanh ghi TH và TL chúng ta hãy nhìn vào ví dụ sau với việc sử dụng tần số dao động XTAL = 11. 0592MHz đối với hệ 8051.

Từ ví dụ 9.10 ta có thể sử dụng những bước sau để tìm các giá trị của các thanh ghi TH và TL.

1. Chia thời gian trễ cần thiết cho 1.0592μs
2. Thực hiện 65536 - n với n là giá trị thập phân nhận được từ bước 1.
3. Chuyển đổi kết quả ở bước 2 sang số Hex với yyxx là giá trị .hex ban đầu cần phải nạp vào các thanh ghi bộ định thời.
4. Đặt TL = xx và TH = yy.

Ví dụ 9.10:

Giả sử tần số XTAL = 11.0592MHz. Hãy tìm các giá trị cần được nạp vào các thanh ghi vào các thanh ghi TH và TL nếu ta muốn độ thời gian trễ là 5μs. Hãy trình bày chương trình cho bộ Timer0 để tạo ra bộ xung với độ rộng 5μs trên chân P2.3.

Lời giải:

Vì tần số XTAL = 11.0592MHz nên bộ đếm tăng sau mỗi chu kỳ 1.085μs. Điều đó có nghĩa là phải mất rất nhiều khoảng thời gian 1,085μs để có được một xung 5μs. Để có được ta chia 5ms cho 1.085μs và nhận được số n = 4608 nhịp. Để nhận được giá trị cần được nạp vào TL và TH thì ta tiến hành lấy 65536 trừ đi 4608 bằng 60928. Ta đổi số này ra số hex thành EE00H. Do vậy, giá trị nạp vào TH là EE Và TL là 00.

```
        CLR   P2.3          ; Xoá bít P2.3
        MOV   TMOD, #01     ; Chọn Timer0, chế độ 1 (16 bít)
HERE:   MOV   TL0, #0       ; Nạp TL = 00
        MOV   TH0, #EEH     ; Nạp TH = EEH
        SETB  P2.3          ; Bật P2.3 lên cao
        SETB  TR0           ; Khởi động bộ định thời Timer0
AGAIN:  JNB   TF0, AGAIN    ; Hiển thị cờ TF0 cho đến khi bộ đếm quay về 0
        CLR   TR0           ; Dừng bộ định thời.
        CLR   TF0           ; Xoá cờ TF0 cho vòng sau.
```

Ví dụ 9.11:

Giả sử ta có tần số XTAL là 11,0592MHz hãy viết chương trình tạo ra một sóng vuông tần số 2kHz trên chân P2.5.

Đây là trường hợp giống với ví dụ 9.10 ngoài trừ một việc là ta phải chọn bít để tạo ra sóng vuông. Xét các bước sau:

a) $T = \frac{1}{f} = \frac{1}{2kHz} = 500\mu s$ là chu kỳ của sóng vuông.

b) Khoảng thời gian cao và phần thấp là $\frac{1}{2}T$ bằng 250µs.

c) Số nhịp cần trong thời gian đó là $\frac{250\mu s}{1{,}085\mu s} = 230$ và giá trị cần nạp vào các thanh ghi cần tìm là 65536 - 230 = 65306 và ở dạng hex là FF1AH.

d) giá trị nạp vào TL là 1AH và TH là FFH.
Chương trình cần viết là:

```
        MOV   TMOD, #10H      ; Chọn bộ định thời Timer0, chế độ 1 (16 bít)
AGAIN:  MOV   TL1, #1AH       ; Gán giá trị byte thấp TL1 = 1AH
        MOV   TH1, #0FFH      ; Gán giá trị byte cao TH1 = FFH
        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.
        CPL   P1.5            ; Bù bít P1.5 để nhận giá trị cao, thấp.
        CLR   TF1             ; Xoá cờ TF1
        SUMP AGAIN            ; Nạp lại bộ định thời vì chế độ 1 không tự nạp
                                lại.
```

Ví dụ 9.12:

Trước hết ta thực hiện các bước sau:

a) Tính chu kỳ sóng vuông: $T = \frac{1}{50Hz} = 20\mu s$

b) Tính thời gian nửa chu kỳ cho phần cao: $\frac{1}{2}T = 10\mu s$

c) Tính số nhịp đồng hồ: $n = \frac{10\mu s}{1{,}085\mu s} = 9216$

d) Tính giá trị cần nạp vào TH và TL: 65536 - 9216 = 56320 chuyển về dạng Hex là DC00H và TH = DCH và TL = 00H.

```
        MOV   TMOD, #10H      ; Chọn bộ định thời Timer0, chế độ 1 (16 bít)
AGAIN:  MOV   TL1, #00        ; Gán giá trị byte thấp TL1 = 00
        MOV   TH1, #0DHCH     ; Gán giá trị byte cao TH1 = DC
        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.
        CPL   P2.3            ; Bù bít P1.5 để nhận giá trị cao, thấp.
        CLR   TF1             ; Xoá cờ TF1
        SUMP AGAIN            ; Nạp lại bộ định thời vì chế độ 1 không tự nạp
                                lại.
```

9.1.4.3 Tạo một độ trễ thời gian lớn.

Như ta đã biết từ các ví dụ trên là lượng thời gian trễ cần tạo ra phụ thuộc vào hai yếu tố:

a) Tần số thạch anh XTAL

b) Thanh ghi 16 bít của bộ định thời ở chế độ 1

Cả hai yếu tố này nằm ngoài khả năng điều chỉnh của lập trình viên 8051. Ví như ta đã biết giá trị lớn nhất của độ trễ thời gian có thể đạt được bằng cách đặt cả TH và TL

bằng 0. Nhưng điều này xảy ra khi như vậy đều không đủ? Ví dụ 9.13 dưới đây cách làm thế nào để có giá trị độ trễ thời gian lớn.

9.1.4.4 Sử dụng bàn tính của Windows để tìm TH và TL.

Bàn tính Calculator của Windows có ngay trong máy tính PC của chúng ta và rất dễ sử dụng để tìm ra các giá trị cho TH và TL. Giả sử tìm giá trị cho TH và TL với độ trễ thời gian lớn là 35.000 nhịp đồng hồ với chu kỳ 1,085μs. Ta thực hiện các bước như sau:

1. Chọn máy tính Calculator từ Windows và đặt chế độ tính về số thập phân Decimal.
2. Nhập số 35.000 vào từ bàn phím.
3. Chuyển về chế độ Hex trên Calculator nó cho ta giá trị 88B8H.
4. Chọn +/- để nhận số đổi dấu - 35.000 dạng thập phân và chuyển về dạng Hex là 7748H.
5. Hai số hex cuối là cho TL = 48 và hai số Hex tiếp theo là cho TH = 77. Ta bỏ quan các số F ở phía bên phải trên Calculator vì số của ta là 16 bít.

Ví dụ 9.13:

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.
```

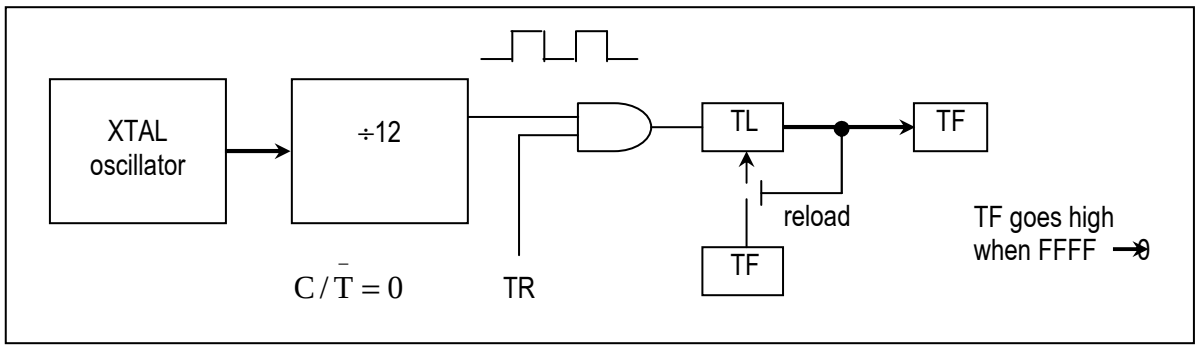
9.1.5 Chế độ O.

Chế độ 0 hoàn toàn giống chế độ 1 chỉ khác là bộ định thời 16 bít được thay bằng 13 bít. Bộ đếm 13 bít có thể giữ các giá trị giữa 0000 đến 1FFFF trong TH - TL. Do vậy khi bộ định thời đạt được giá trị cực đại của nó là 1FFFH thì nó sẽ quay trở về 0000 và cờ TF được bật lên.

9.1.6 Lập trình chế độ 2.

Các đặc trưng và các phép tính của chế độ 2:

1. Nó là một bộ định thời 8 bít, do vậy nó chỉ cho phép các giá trị từ 00 đến FFH được nạp vào thanh ghi TH của bộ định thời.
2. Sau khi TH được nạp với giá trị 8 bít thì 8051 lấy một bản sao của nó đưa vào TL. Sau đó bộ định thời phải được khởi động. Điều này được thực hiện bởi lệnh "SETB TR0" đối với Timer0 và "SETB TR1" đối với Timer1 giống như ở chế độ 1.
3. Sau khi bộ định thời được khởi động, nó bắt đầu đếm tăng lên bằng cách tăng thanh ghi TL. Nó đếm cho đến khi đại giá trị giới hạn FFH của nó. Khi nó quay trở về 00 từ FFH, nó thiết lập cờ bộ định thời TF. Nếu ta sử dụng bộ định thời Timer0 thì đó là cờ TF0, còn Timer1 thì đó là cờ TF1.



4. Khi thanh ghi TL quay trở về 00 từ FFH thì TF được bật lên 1 thì thanh ghi TL được tự động nạp lại với giá trị ban đầu được giữ bởi thanh ghi TH. Để lặp lại quá trình chúng ta đơn giản chỉ việc xoá cờ TF và để cho nó chạy mà không cần sự can thiệp của lập trình viên để nạp lại giả trị ban đầu. Điều này làm cho chế độ 2 được gọi là chế độ từ nạp lại so với chế độ 1 thì ta phải nạp lại các thanh ghi TH và TL.

Cần phải nhấn mạnh rằng, chế độ 2 là bộ định thời 8 bít. Tuy nhiên, nó lại có khả năng tự nạp khi tự nạp lại thì TH thực chất là không thay đổi với giá trị ban đầu được giữ nguyên, còn TL được nạp lại giá trị được sao từ TH. Chế độ này có nhiều ứng dụng bao gồm việc thiết lập tần số baud trong truyền thông nối tiếp như ta sẽ biết ở chương 10.

9.1.5.1 Các bước lập trình cho chế độ 2.

Để tạo ra một thời gian trễ sử dụng chế độ 2 của bộ định thời cần thực hiện các bước sau:

1. Nạp thanh ghi giá trị TMOD để báo bộ định thời gian nào (Timer0 hay Timer1) được sử dụng và chế độ làm việc nào của chúng được chon.
2. Nạp lại các thanh ghi TH với giá trị đếm ban đầu.
3. Khởi động bộ định thời.
4. Duy trì hiển thị cờ bộ định thời TF sử dụng lệnh "JNB TFx, đích" để xem nó sẽ được bật chưa. Thoát vòng lặp khi TF lên cao.
5. Xoá cờ TF.
6. Quay trở lại bước 4 vì chế độ 2 là chế độ tự nạp lại.

Ví dụ 9.14 minh hoạ những điều này. Để có được độ chễ lớn chúng ta có thể dùng nhiều thanh ghi như được chỉ ra trong ví dụ 9.15.

Ví dụ 9.14:

Giả sử tần số XTAL = 11.0592MHz. Hãy tìm a) tần số của sóng vuông được tạo ra trên chân P1.0 trong chương trình sau và b) tần số nhỏ nhất có thể có được bằng chương trình này và giá trị TH để đạt được điều đó.

```
        MOV   TMOD, #20H      ; Chọn Timer1/ chế độ 2/ 8 bít/ tự nạp lại.
        MOV   TH1, #5         ; TH1 = 5
        SETB  TR1             ; Khởi động Timer1
BACK:   JNB   TF1, BACK       ; giữ nguyên cho đến khi bộ định thời quay về 0
        CPL   P1.0            ; Dừng bộ định thời.
        CLR   TF1             ; Xoá cờ bộ định thời TF1
        SJMP  BACK            ; Chế độ 2 tự động nạp lại.
```

Lời giải:

a) Trước hết để ý đến đích của lệnh SJMP. Trong chế độ 2 ta không cần phải nạp lại TH vì nó là chế độ tự nạp. Bây giờ ta lấy (256 - 05).1.085μs = 251×1.085μs = 272.33μs là phần cao của xung. Cả chu kỳ của xung là T = 544.66μs và tần số là $\frac{1}{T} = 1{,}83597\text{kHz}$.

b) Để nhận tần số nhỏ nhất có thể ta cần tạo T chu kỳ lớn nhất có thể có nghĩa là TH = 00. Trong trường hợp này ta có $T = 2 \times 256 \times 1.085\mu s = 555.52\mu s$ và tần số nhỏ nhất sẽ là $\frac{1}{T} = 1{,}8\text{kHz}$.

Ví dụ 9.15:

Hãy tìm tần số của xung vuông được tạo ra trên P1.0.

Lời giải:

```
        MOV   TMOD, #2H      ; Chọn Timer0, chế độ 1 (8 bít tự nạp lại)
AGAIN:  MOV   TH0, #0        ; Nạp TH0 = 00
        MOV   R5, #250       ; Đếm cho độ trễ lớn
        ACALL DELAY
        CPL   P1.0
        SJMP  AGAIN
DELAY:  SETB  TR0            ; Khởi động Timer0
BACK:   JNB   TF1, BACK      ; giữ nguyên cho đến khi bộ định thời quay về 0
        CLR   TR0            ; Dừng Timer0.
        CLR   TF0            ; Xoá cờ TF0 cho vòng sau.
        DJNZ  R5, DELAY
        RET
```

$T = 2 \times (250 \times 256 \times 1.085\mu s) = 1.38.88\text{ms}$ và f = 72Hz.

Ví dụ 9.16:

Giả sử ta đang lập trình chế độ 2 hãy tìm các giá trị (dạng Hex) cần nạp vào TH cho các trường hợp sau:

a) MOV TH1, #200 b) MOV TH0, #-60
c) MOV TH1, #-3 d) MOV TH1, #-12
e) MOV TH0, #-48

Lời giải:

Chúng ta có thể sử dụng bàn tính Calculator của Windows để kiểm tra kết quả được cho bởi trình hợp ngữ. Hãy chọn Calculator ở chế độ Decimal và nhập vào số 200. Sau đó chọn Hex, rồi ấn +/ - để nhận giá trị của TH. Hãy nhớ rằng chúng ta chỉ sử dụng đúng hai chữ số và bỏ qua phần bên trái vì dữ liệu chúng ta là 8 bít. Kết quả ta nhận được như sau:

Dạng thập phân	*Số bù hai (giá trị TH)*
- 200	38H
- 60	C4H
- 3	FDH
- 12	F4H
- 48	DOH

9.1.5.2 Các trình hợp ngữ và các giá trị âm.

Vì bộ định thời là 8 bít trong chế độ 2 nên ta có thể để cho trình hợp ngữ tính giá trị cho TH. Ví dụ, trong lệnh "MOV TH0, # - 100" thì trình hợp ngữ sẽ tính toán – 100 = 9C và gán TH = 9CH. Điều này làm cho công việc của chúng ta dễ dàng hơn.

Ví dụ 9.17:

Hãy tìm a) tần số sóng vuông được tạo ra trong đoạn mã dưới đây và độ đầy xung của sóng này.

```
        MOV   TMOD, #2H          ; Chọn bộ Timer0/ chế độ 2/ (8 bít, tự nạp lại).
        MOV   TH0, # - 150       ; Nạp TH0 = 6AH là số bù hai của  - 150
        SETB  TR1                ; Khởi động Timer1
AGAIN:  SETB P1.3                ; P1.3 = 1
        ACALL DELAY
        ACALL P1.3               ; P1.3 = 0
        ACALL DELAY
        SJMP AGAIN

        SETB  TR0                ; Khởi động Timer0
BACK:   JNB   TF0, BACK          ;giữ nguyên cho đến khi bộ định thời quay về 0
        CLR   TR0                ; Dừng Timer0
        CLR   TF0                ; Xoá cờ TF cho vòng sau.
        RET
```

Lời giải:

Để tìm giá trị cho TH ở chế độ 2 thì trình hợp ngữ cần thực hiện chuyển đổi số âm khi ta nhập vào. Điều này cũng làm cho việc tính toán trở nê dễ dàng. Vì ta đang sử dụng 150 xung đồng hồ, nên ta có thời gian trễ cho chương trình con DELAY là $150 \times 1.085\mu s$ và tần số là $f = \frac{1}{T} = 2{,}048\text{kHz}$.

Để ý rằng trong nhiều tính toán thời gian trễ ta đã bỏ các xung đồng hồ liên quan đến tổng phí các lệnh trong vòng lặp. Để tính toán chính xác hơn thời gian trễ và cả tần số ta đang cần phải đưa chúng vào. Nếu ta dùng một máy hiện sóng số và ta không nhận được tần số đúng như ta tính toán thì đó là do tổng phí liên quan đến các lệnh gọi trong vòng lặp.

Trong phần này ta đã dùng bộ định thời 8051 để tạo thời gian trễ. Tuy nhiên, công dụng mạnh hơn và sáng tạo hơn của các bộ định thời này là sử dụng chúng như các bộ đếm sự kiện. Chúng ta sẽ bàn về công dụng của bộ đếm này ở phần kế tiếp.

9.2 Lập trình cho bộ đếm.

Ở phần trên đây ta đã sử dụng các bộ định thời của 8051 để tạo ra các độ trễ thời gian. Các bộ định thời này cũng có thể được dùng như các bộ đếm các sự kiện xảy ra bên ngoài 8051. Công dụng của bộ đếm/ bộ định thời như bộ đếm sự kiện sẽ được tình bày ở phần này. Chừng nào còn liên quan đến công dụng củ bộ định thời như bộ đếm sự kiện thì mọi vấn đề mà ta nói về lập trình bộ định thời ở phần trước cũng được áp dụng cho việc lập trình như là một bộ đếm ngoại trừ nguồn tần số. Đối với bộ định thời/ bộ đếm khi dùng nó như bộ định thời thì nguồn tần số là tần số thạch anh của 8051. Tuy nhiên, khi nó được dùng như một bộ đếm thì nguồn xung để tăng nội dung các thanh ghi TH và TL là từ bên ngoài 8051. Ở chế độ bộ đếm, hãy lưu ý rằng các thanh ghi TMOD và TH, TL cũng giống như đối với bộ định thời được bàn ở phần trước, thậm chí chúng vẫn có cùng tên gọi. Các chế độ của các bộ định thời cũng giống nhau.

9.2.1 Bít C/T trong thanh ghi TMOD.

Xem lại phần trên đây về bít C/T trong thanh ghi TMOD ta thấy rằng nó quyết định nguồn xung đồng hồ cho bộ định thời. Nếu bít C/T = 0 thì bộ định thời nhận các xung đồng hồ từ bộ giao động thạch anh của 8051. Ngược lại, khi C/T = 1 thì bộ định thời được sử dụng như bộ đếm và nhận các xung đồng hồ từ nguồn bên ngoài của 8051. Do vậy, khi bít C/T = 1 thì bộ đếm lên, khi các xung được đưa đến chân 14 và 15. Các chân này có tên là T0 (đầu vào của bộ định thời Timer0) và T1 (đầu vào của bộ Timer1). Lưu ý rằng hai chân này thuộc về cổng P3. Trong trường hợp của bộ Timer0 khi C/T = 1 thì chân P3.4 cấp xung đồng hồ và bộ đếm tăng lên đối với mỗi xung đồng hồ đi đến từ chân này. Tương tự như vậy đói với bộ Timer1 thì khi C/T = 1 với mỗi xung đồng hồ đi đến từ P3.5 bộ đếm sẽ đếm tăng lên 1.

Bảng 9.1: Các chân cổng P3 được dùng cho Timer0 và Timer1.

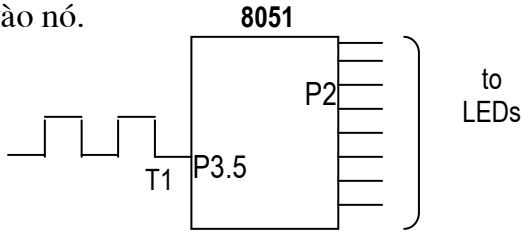
Chân	Chân cổng	Chức năng	Mô tả
14	P3.4	T0	Đầu vào ngoài của bộ đếm 0
15	P3.5	T1	Đầu vào ngoài của bộ đếm 1

Ví dụ 9.18:

Giả sử rằng xung đồng hồ được cấp tới chân T1, hãy viết chương trình cho bộ đếm 1 ở chế độ 2 để đếm các xung và hiển thị trạng thái của số đếm TL1 trên cổng P2.
Lời giải:

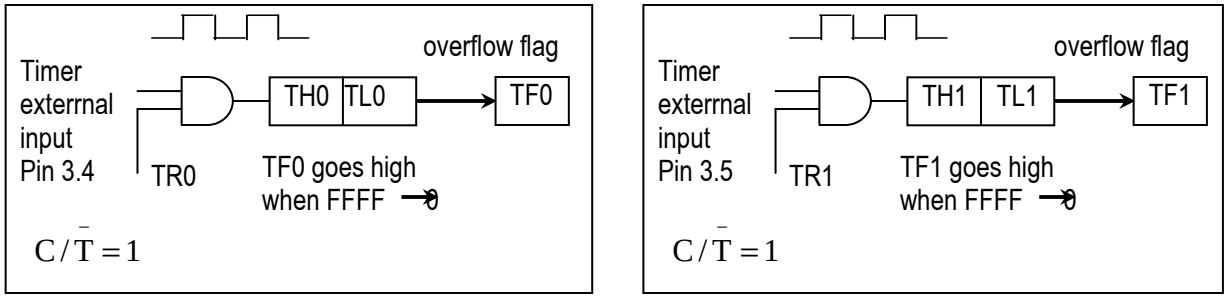
```
        MOV   TMOD, #01100000B   ; Chọn bộ đếm 1, chế độ 2, bít C/T = 1
                                   xung ngoài.
        MOV   TH1, #0            ; Xoá TH1
        SETB  P3.5               ; Lấy đầu vào T1
AGAIN:  SETB  TR1                ; Khởi động bộ đếm
BACK:   MOV   A, TL1             ; Lấy bản sao số đếm TL1
        MOV   P2, A              ; Đưa TL1 hiển thị ra cổng P2.
        JNB   TF1, Back          ; Duy trì nó nếu TF = 0
        CLR   TR1                ; Dừng bộ đếm
        CLR   TF1                ; Xoá cờ TF
        SJMP  AGAIN              ; Tiếp tục thực hiện
```

Để ý trong chương trình trên về vai trò của lệnh "SETB P3.5" vì các cổng được thiết lập dành cho đầu ra khi 8051 được cấp nguồn nên ta muốn P3.5 trở thành đầu vào thì phải bật nó lên cao. Hay nói cách khác là ta phải cấu hình (đưa lên cao) chân T1 (P3.5) để cho phép các xung được cấp vào nó.



Trong ví dụ 9.18 chúng ta sử dụng bộ Timer1 như bộ đếm sự kiện để nó đếm lên mỗi khi các xung đồng hồ được cấp đến chân P3.5. Các xung đồng hồ này có thể biểu diễn số người đi qua cổng hoặc số vòng quay hoặc bất kỳ sự kiện nào khác mà có thể chuyển đổi thành các xung.

Trong ví dụ 9.19 các thanh ghi TL được chuyển đổi về mã ASCII để hiển thị trên một LCD.



Hình 9.5: a) Bộ Timer0 với đầu vào ngoài (chế độ 1)
b) Bộ Timer1 với đầu vào ngoài (chế độ 1)

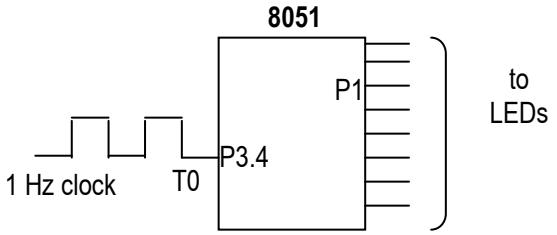
Ví dụ 9.19:

giả sử rằng một xung tần số 1Hz được nối tới chân đầu vào P3.4. Hãy viết chương trình hiển thị bộ đếm 0 trên một LCD. Hãy đặt số ban đầu của TH0 là - 60.

Lời giải:

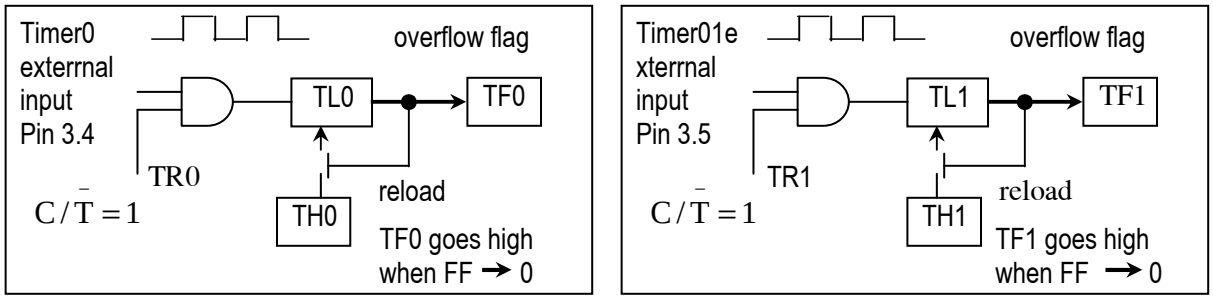
Để hiển thị số đếm TL trên một LCD ta phải thực hiện chuyển đổi giữ liệu 8 bít nhị phân về ASCII.

```
        ACALL   LCD-SET UP          ; Gọi chương trình con khởi tạo CLD
        MOV     TMOD, #000110B      ; Chọn bộ đếm 0, chế độ 2, bít C/T = 1
        MOV     TH0, # - 60         ; Đếm 60 xung
        SETB    P3.4                ; Lấy đầu vào T0
AGAIN:  SETB    TR0                 ; Sao chép số đếm TL0
BACK:   MOV     A, TL0              ; Gọi chương trình con để chuyển đổi
                                      trong các thanh ghi R2, R3, R4.
        ACALL   CONV                ; Gọi chương trình con hiển thị trên LCD
        ACALL   DISLAY              ; Thực hiện vòng lặp nếu TF = 0
        JNB     TF0, BACK           ; Dừng bộ đếm 0
        CLR     TR0                 ; Xoá cờ TF0 = 0
        CLR     TF0                 ; Tiếp tục thực hiện
        SJMP    AGAIN               ; Việc chuyển đổi nhị phân về mã ASCII
khi trả dữ liệu ASCII có trong các thanh ghi R4, R3, R2 (R2 có LSD) - chữ số nhỏ nhất.
CONV:  MOV      B, #10              ; Chia cho 10
        DIV     AB
        MOV     R2, B               ; Lưu giữ số thấp
        MOV     B, #10              ; Chia cho 10 một lần nữa
        DIV     AB
        ORL     A, #30H             ; Đổi nó về ASCII
        MOV     R4, A               ; Lưu chữ số có nghĩa lớn nhất MSD
        MOV     A, B                ;
        ORL     A, #30H             ; Đổi số thứ hai về ASCII
        MOV     R3, A               ; Lưu nó
        MOV     A, R2
        ORL     A, #30H             ; Đổi số thứ ba về ASCII
        MOV     R2, A               ; Lưu số ASCII vào R2.
        RET
```



Sử dụng tần số 60Hz ta có thể tạo ra các giây, phút, giờ.

Lưu ý rằng trong vòng đầu tiên, nó bắt đầu từ 0 vì khi RESET thì TL0 = 0; Để giải quyết vấn đề này hãy nạp TL0 với giá trị - 60 ở đầu chương trình.



Hình 9.6: Bộ Timer0 với đầu vào ngoài (chế độ 2)

Hình 9.7: Bộ Timer0 với đầu vào ngoài (chế độ 2)

Như một ví dụ ứng dụng khác của bộ định thời gian với bít C/T = 1, ta có thể nạp một sóng vuông ngoài với tần số 60Hz vào bộ định thời. Chương trình sẽ tạo ra các đơn vị thời gian chuẩn theo giây, phút, giờ. Từ đầu vào này ta hiển thị lên một LCD. Đây sẽ là một đồng hồ số tuyệt vời nhưng nó không thật chính xác. Ví dụ này có thể tìm thấy ở phụ lục Appendix E.

Trước khi kết thúc chương này ta cần nhắc lại hai vấn đề quan trọng.

1. Chúng ta có thể nghĩ rằng công dụng của lệnh "JNB TFx, đích" để hiển thị mức cao của cờ TF là một sự lãng phí thời gian của BVĐK. Điều đó đúng có một giải pháp cho vấn đề này là sử dụng các ngắt. Khi sử dụng các ngắt ta có thể đi thực hiện các công việc khác với BVĐK. Khi cờ TF được bật thì nó báo cho ta biết đây là điểm quan trọng về thế mạnh của 8051 (mà ta sẽ bàn ở chương 11).
2. Chúng ta muốn biết các thanh ghi TR0 và TR1 thuộc về đâu. Chúng thuộc về một thanh ghi gọi là TCON mã sẽ được ban sau ở đây (TCON - là thanh ghi điều khiển bộ đếm (bộ định thời)).

Bảng 9.2: Các lệnh tương đương đối với thanh ghi điều khiển bộ định thời.

Đối với Timer0	
	SETB TR0 = SETB TCON.4
	CLR TR0 = CLR TCON.4
	SETB TF = SETB TCON.5
	CLR TF0 = CLR TCON.5
Đối với Timer1	
	SETB TR1 = SETB TCON.6

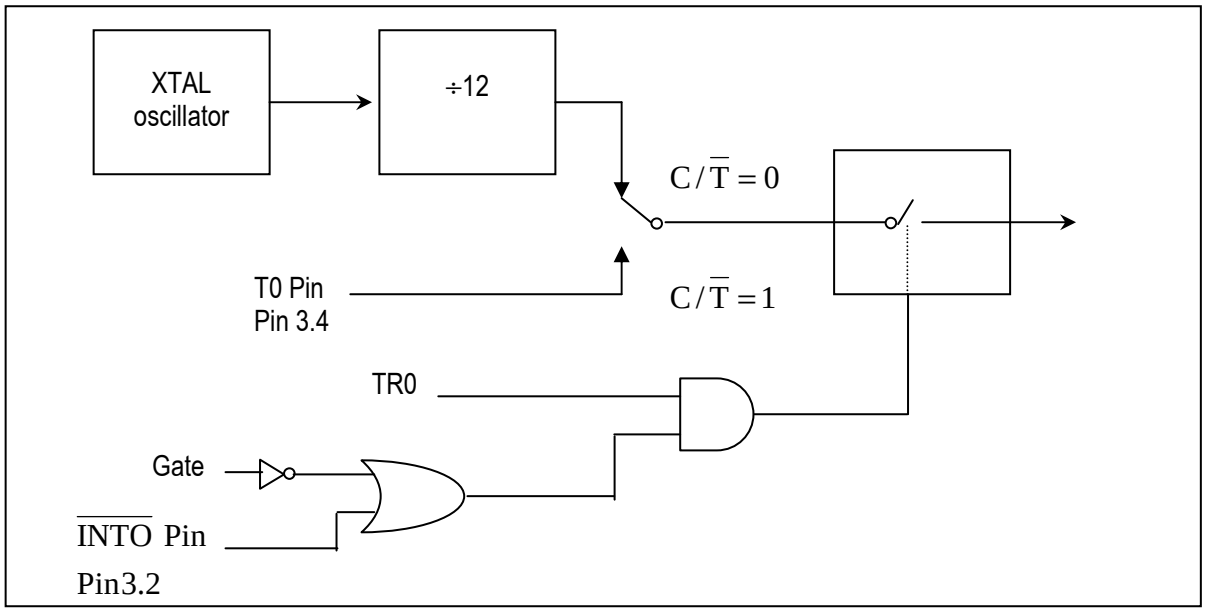
CLR TR1 = CLR TCON.6
SETB TF1 = SETB TCON.7
CLR TF1 = CLR TCON.7

9.2.2 Thanh ghi TCON.

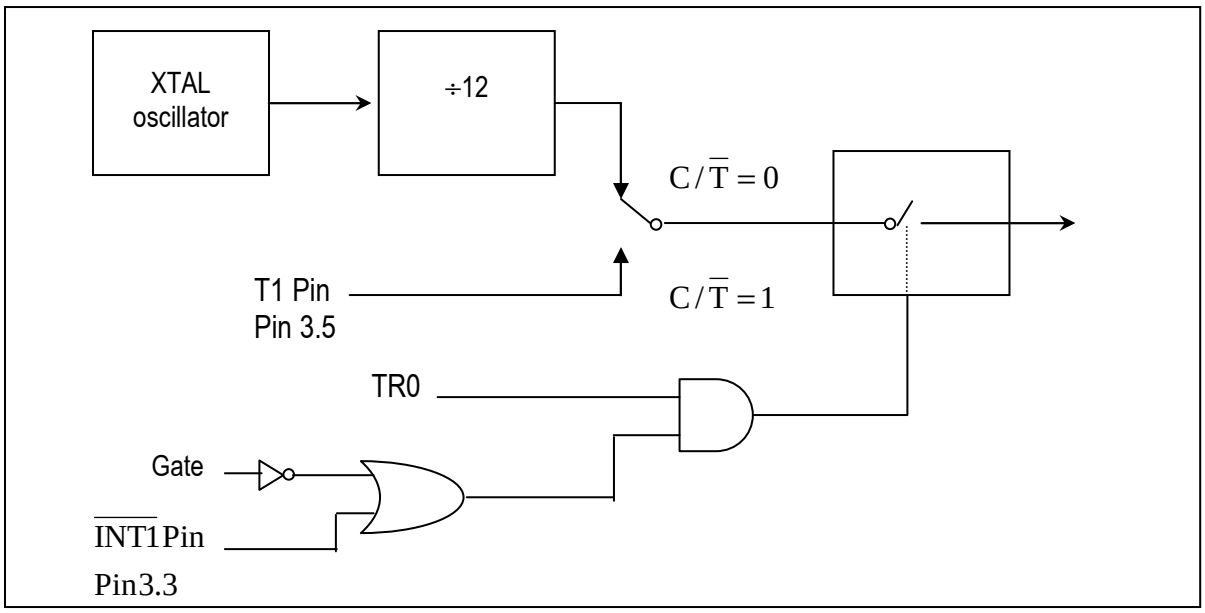
Trong các ví dụ trên đây ta đã thấy công dụng của các cờ TR0 và TR1 để bật/ tắt các bộ định thời. Các bít này là một bộ phận của thanh ghi TCON (điều khiển bộ định thời). Đây là thanh ghi 8 bít, như được chỉ ra trong bảng 9.2 thì bốn bít trên được dùng để lưu cất các bít TF và TR cho cả Timer0 và Timer1. Còn bốn bít thấp được thiết lập dành cho điều khiển các bít ngắt mà ta sẽ bàn ở chương 11. Chúng ta phải lưu ý rằng thanh ghi TCON là thanh ghi có thể đánh địa chỉ theo bít được. Nên ta có thể thay các lệnh như "SETB TR1" là "CLR TR1" bằng các lệnh tương ứng như "SET TCON.6" và "CLR TCON.6" (Bảng 9.2).

9.3 Trường hợp khi bít GATE = 1 trong TMOD.

Trước khi kết thúc chương ta cần bàn thêm về trường hợpợ khi bít GATE = 1trong thanh ghi TMOD. Tất cả những gì chúng ta vừa nói trong chương này đều giả thiết GATE = 0. Khi GATE = 0 thì bộ định thời được khởi động bằng các lệnh "SETB TR0" và "SETB TR1" đối với Timer0 và Timer1 tương ứng. Vậy điều gì xảy ra khi bít GATE = 1? Như ta có thể nhìn thây trên hình 9.8 và 9.9 thì nếu GATE = 1 thì việc khởi động và dừng bộ định thời được thực hiện từ bên ngoài qua chân P2.3 và P3.3 đối với Timer0 và Timer1 tương ứng. Mặc dù rằng TRx được bật lên bằng lệnh "SETB TRx" thì cũng cho phép ta khởi động và dừng bộ định thời từ bên ngoài tại bất kỳ thời điểm nào thông qua công tắc chuyển mạch đơn giản. Phương pháp điều khiển phần cứng để dừng và khởi động bộ định thời nay có thể có rất nhiều ứng dụng. Ví dụ, chẳng hạn 8051 được dùng trong một sản phẩm phát báo động mỗi giây dùng bộ Timer0 theo nhiều việc khác. Bộ Timer0 được bật lên bằng phần mềm qua lệnh "SETB TR0" và nằm ngoài sự kiểm soát của người dùng sản phẩm đó. Tuy nhiên, khi nối một công tắc chuyển mạch tới chân P2.3 ta có thể dừng và khởi động bộ định thời gian bằng cách đó để tắt báo động.



Hình 9.8: Bộ định thời/ bộ đếm 0.



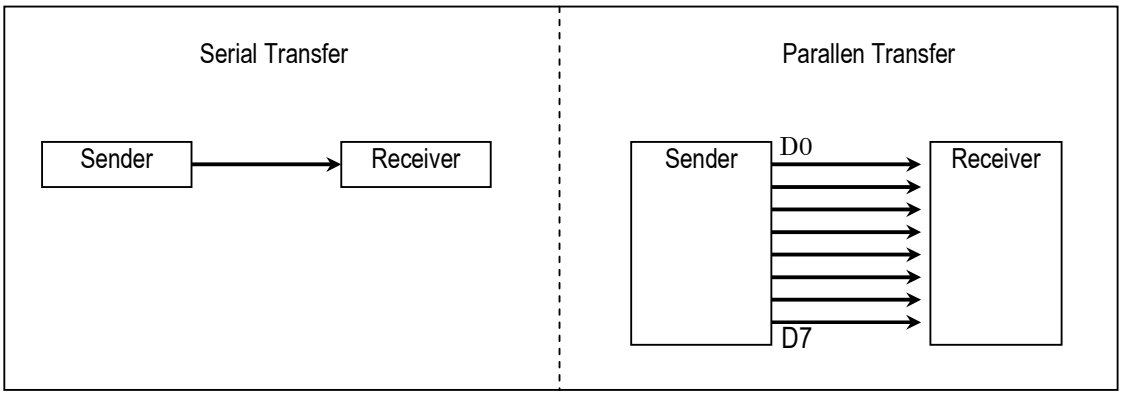
Hình 9.9: Bộ định thời/ bộ đếm 1.

CHƯƠNG 10
Truyền thông nối tiếp của 8051

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 dudx 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.

10.1 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 10.1 là sơ đồ truyền nối tiếp so với sơ đồ truyền song song.



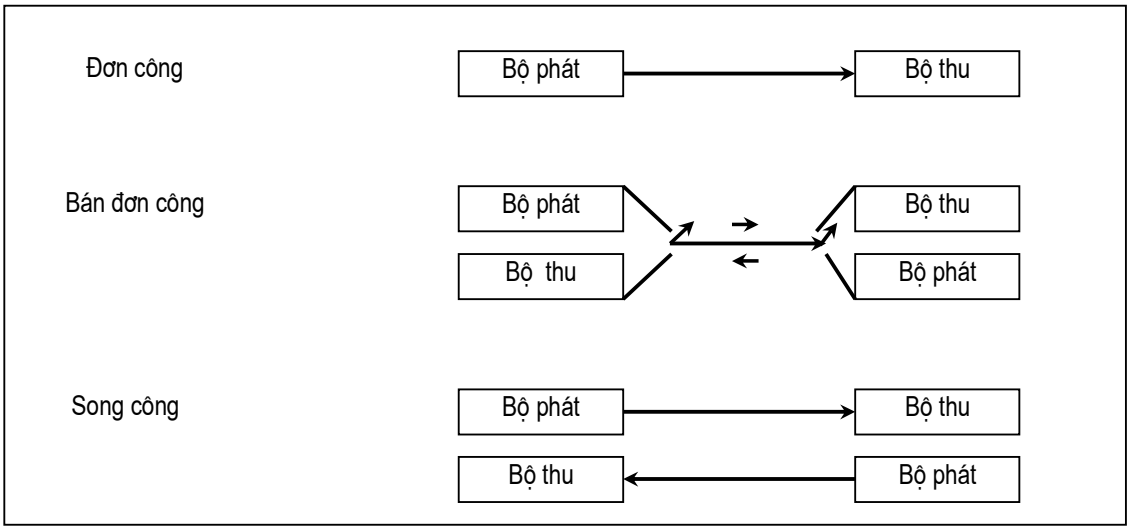
Hình 10.1: Sơ đồ truyền dữ liệu nối tiếp so với sơ đồ truyền song song.

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 (chuyển tín hiệu âm thanh về các số 0 và 1).

Truyền thông dữ liệu nối tiếp sử dụng hai phương pháp đồng bộ và dị bộ. Phương pháp đồng bộ truyền một khối dữ liệu (các ký tự) tại cùng thời điểm trong khi đó truyền dị bộ chỉ truyền từng byte một. Có thể viết phần mềm để sử dụng một trong hai phương pháp này, những chương trình có thể rất dài và buồn tẻ. Vì lý do này mà nhiều nhà sản xuất đã cho ra thị trường nhiều loại IC chuyên dụng phục vụ cho truyền thông dữ liệu nối tiếp. Những IC này phục vụ như các bộ thu - phát dị bộ tổng hợp VART (Universal Asynchronous Receiver Transmitter) và các bộ thu - phát đồng - dị bộ tổng hợp UBART (Universal Asynchronous Receiver Transmitter). Bộ vi điều khiển 8051 có một cài sẵn một UART mà nó sẽ được bàn kỳ ở mục 10.3.



Hình 10.2: Truyền dữ liệu đơn công, bán công và song công.

10.1.1 Truyền dữ liệu bán công và song công.

Trong truyền dữ liệu nếu dữ liệu có thể được vừa phát và vừa được thu thì gọi là truyền song công. Điều này tương phản với truyền đơn công chẳng hạn như các máy in chỉ nhận dữ liệu từ máy tính. Truyền song công có thể có hai loại là bán song công và song công hoàn toàn phụ thuộc vào truyền dữ liệu có thể xảy ra đồng thời không? Nếu dữ liệu được truyền theo một đường tại một thời điểm thì được gọi là truyền bán song công. Nếu dữ liệu có thể đi theo cả hai đường cùng một lúc thì gọi là song công toàn phần. Tất nhiên, truyền song công đòi hỏi hai đường dữ liệu (ngoài đường âm của tín hiệu), một để phát và một để thu dữ liệu cùng một lúc.

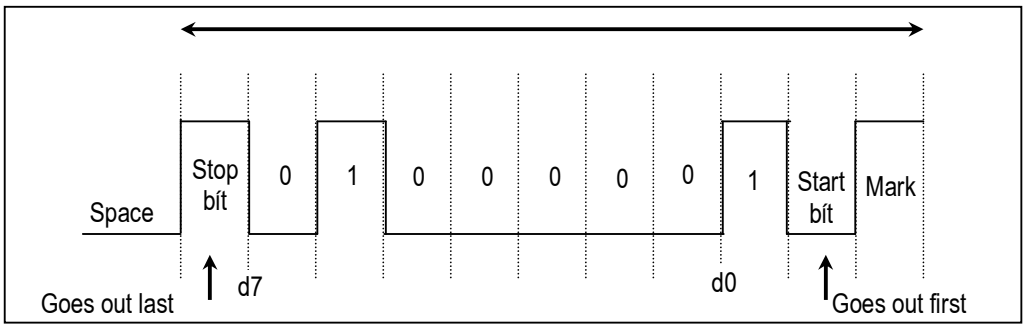
10.1.2 Truyền thông nối tiếp dị bộ và đóng khung dữ liệu.

Dữ liệu đi vào ở đầu thu của đường dữ liệu trong truyền dữ liệu nói tiếp toàn là các số 0 và 1, nó thật là khó làm cho dữ liệu ấy có nghĩa là nếu bên phát và bên thu không cùng thống nhất về một tệp các luật, một thủ tục, về cách dữ liệu được đóng gói, bao nhiêu bít tạo nên một ký tự và khi nào dữ liệu bắt đầu và kết thúc.

10.1.3 Các bít bắt đầu và dừng.

Truyền thông dữ liệu nối tiếp dị bộ được sử dụng rộng rãi cho các phép truyền hướng kỹ tự, còn các bộ truyền dữ liệu theo khối thì sử dụng phương phát đồng bộ.

Trong phương pháp dị bộ, mỗi ký tự được bố trí giữa các bít bắt đầu (start) và bít dừng (stop). Công việc này gọi là đóng gói dữ liệu. Trong đóng gói dữ liệu đối với truyền thông dị bộ thì dữ liệu chẳng hạn là các ký tự mã ASCII được đóng gói giữa một bít bắt đầu và một bít dừng. Bít bắt đầu luôn luôn chỉ là một bít, còn bít dừng có thể là một hoặc hai bít. Bít bắt đầu luôn là bít thấp (0) và các bít dừng luôn là các bít cao (bít 1). Ví dụ, hãy xét ví dụ trên hình 10.3 trong đó ký tự "A" của mã ASCII (8 bít nhị phân là 0100 0001) đóng gói khung giữa một bít bắt đầu và một bít dừng. Lưu ý rằng bít thấp nhất LSB được gửi ra đầu tiên.



Hình 10.3: Đóng khung một ký tự "A" của mã ASCII (41H) có tín hiệu là 1 (cao) được coi như là một dấu (mark), còn không có tín hiệu tức là 0 (thấp) thì được coi là khoảng trống (space). Lưu ý rằng phép truyền bắt đầu với start sau đó bít D0, bít thấp nhất LSB, sau các bít còn lại cho đến bít D7, bít cao nhất MSB và cuối cùng là bít dừng stop để báo kết thúc ký tự "A".

Trong truyền thông nối tiếp dị bộ thì các chíp IC ngoại vi và các modem có thể được lập trình cho dữ liệu với kích thước theo 7 bít hoặc 8 bít. Đây là chưa kể các bít dừng stop có thể là 1 hoặc 2 bít. Trong khi các hệ ASCII cũ hơn (trước đây) thì các ký tự là 7 bít thì ngay nay do việc mở rộng các ký tự ASCII nên dữ liệu nhìn chung là 8 bít. Trong các hệ cũ hơn do tốc độ chậm của các thiết bị thu thì phải sử dụng hai bít dừng để đảm bảo thời gian tổ chức truyền byte kế tiếp. Tuy nhiên, trong các máy tính PC hiện tại chỉ sử dụng 1 bít stop như là chuẩn.

Giả sử rằng chúng ta đang truyền một tệp văn bản các ký tự ASCII sử dụng 1 bít stop thì ta có tổng cộng là 10 bít cho mỗi ký tự gồm: 8 bít cho ký tự ASCII chuẩn và 1 bít start cùng 1 bít stop. Do vậy, đối với mỗi ký tự 8 bít thì cần thêm 2 bí vị chi là mất 25% tổng phí.

Trong một số hệ thống để nhằm duy trì tính toàn vẹn của dữ liệu thì người ta còn thêm một bít lẻ (parity bít). Điều này có nghĩa là đối với mỗi ký tự (7 hoặc 8 bít tuỳ từng hệ) ta có thêm một bít ngoài các bít start và stop. Bít chẵn lẻ là bít chẵn hoặc bít lẻ. Nếu là bít lẻ là số bít của dữ liệu bao gồm cả bít chẵn lẻ sẽ là một số lẻ các sô 1. Tương tự như vậy đối với trường hợp bít chẵn thì số bít của dữ liệu bao gồm cả bít chẵn - lẻ sẽ là một số chẵn của các số 1. Ví dụ, ký tự "A" của mã ASCII ở dạng nhị phân là 0100 0001, có bít 0 là bít chẵn. Các chíp UART đều cho phép việc lập trình bít chẵn - lẻ về chẵn, lẻ hoặc không phân biệt chẵn lẻ.

10.1.4 Tốc độ truyền dữ liệu.

Tốc độ truyền dữ liệu trong truyền thông dữ liệu nối tiếp được gọi là bít trong giây bps (bít per second). Ngoài ra, còn được sử dụng một thuật ngữ rộng rãi nữa là tốc độ baud. Tuy nhiên, các tốc baud và bps là hoàn toàn không bằng nhau. Điều này là do tốc baud là thuật ngữ của modem và được định nghĩa như là sô lần thay đổi của tín hiệu trong một giây. Trong các modem có những trường hợp khi một sự thay đổi của tín hiệu thì nó truyền vài bít dữ liệu. Nhưng đối với một dây dẫn thì tốc độ baud

và bps là như nhau nên trong cuốn sách này chúng ta có thể dùng thay đổi các thuật ngữ này cho nhau.

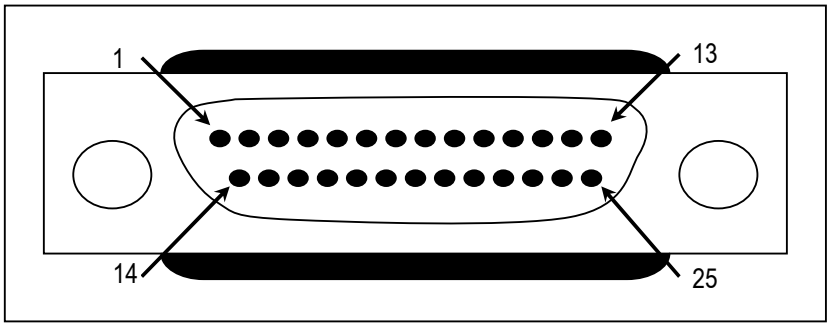
Tốc độ truyền dữ liệu của một hệ máy tính đã cho phụ thuộc vào các cổng truyền thông kết nối vào trong hệ thống đo. Ví dụ, các máy tính PC/XT trước đây của IBM có thể truyền dữ liệu với tốc độ 100 đến 9600 bps. Tuy nhiên, trong những năm gần đây thì các máy tính PC dựa trên Pentium truyền dữ liệu với tốc độ lên tới 56kbps. Cần phải nói thêm rằng trong truyền thông dữ liệu nối tiếp dị bộ thì tốc độ baud nhìn chung là bị giới hạn ở 100.000 bps.

10.1.5 Các chuẩn RS232.

Để cho phép tương thích giữa các thiết bị truyền thông dữ liệu được sản xuất bởi các hãng khác nhau thì một chuẩn giao diện được gọi là RS232 đã được thiết lập bởi hiệp hội công nghiệp điện tử EIA vào năm 19960. Năm 1963 nó được sửa chỉnh và được gọi là RS232A và vào các năm 1965 và 1969 thì được đổi thành RS232B và RS232C. Ở đây chúng ta đơn giản chỉ nói đến RS232. Ngày nay RS232 là chuẩn giao diện I/O vào - ra nối tiếp được sử dụng rộng rãi nhất. Chuẩn này được sử dụng trong máy tính PC và hàng loạt các thiết bị khác nhau. Tuy nhiên, vì nó được thiết lập trước họ lô-gíc TTL rất lâu do vậy điện áp đầu vào và đầu ra của nó không tương thích với mức TTL. 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. Các chíp IC MAX232 nhìn chung được coi như cá bộ điều khiển đường truyền. Kết nối RS232 đến MAX232 được thoả thuận ở phần 10.2.

10.1.6 Các chân của RS232.

Bảng 10.1 cung cấp sơ đồ chân của cáp RSE232 và các tên gọi của chúng thường được gọi là đầu nối DB - 25. Trong lý hiệu thì đầu nối cắm vào (đầu đực) gọi là DB - 25p và đầu nối cái được gọi là DB - 25s.



Hình 10.4: Đầu nối DB - 25 của RS232.

Vì không phải tất cả mọi chân đều được sử dụng trong cáp cảu máy tính PC, nên IBM đưa ra phiên bản của chuẩn vào/ra nối tiếp chỉ sử dụng có 9 chân gọi là DB - 9 như trình bày ở bảng 10:2 và hình 10.5.

Bảng 10.1: Các chân của RS232, 25 chân (DB - 25).

Số chân	Mô tả
1	Đất cách ly (Protective Cround)
2	Dữ liệu được truyền TxD (TráNsmitted data)
3	Dữ liệu được phân RxD (Received data)
4	Yêu cầu gửi RTS (Request To Send)
5	Xoá để gửi CIS (Clear To Send)
6	Dữ liệu sẵn sàng DSR (Data Set Ready)

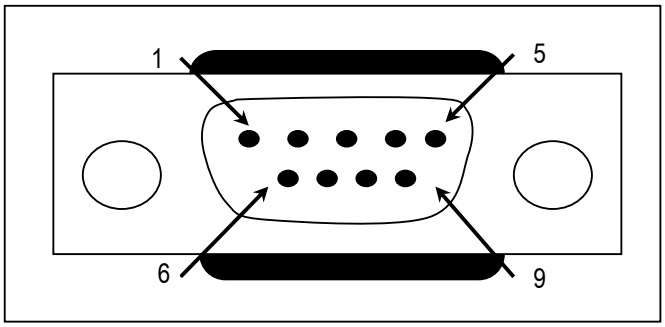
7	Đất của tín hiệu GND (Signal Cround)
8	Tách tín hiệu mạng dữ liệu DCD (Data Carrier Detect)
9/10	Nhận để kiểm tra dữ liệu (Received for data testing)
11	Chưa dùng
12	Tách tín hiệu mạng dữ liệu thứ cấp (Secondary data carrier detect)
13	Xoá đẻ nhận dữ liệu thứ cấp (Secondary Clear to Send)
14	Dữ liệu được truyền thứ cấp (Secondary Transmit Signal Element Timing)
15	Truyền phân chia thời gian phần tử tín hiệu (Transmit Signal Element
16	Timing)
17	Dữ liệu được nhận thứ cấp (Secondary Received data)
18	Nhận phân chia thời gian phần tử tín hiệu (Receiveo Signal Element
19	Timing)
20	Chưa dùng
21	Yêu cầu để nhận thứ cấp (Secondary Request to Send)
22	Đầu dữ liệu sẵn sàng (Data Terminal Ready)
23	Phát hiện chất lượng tín hiệu (Signal Qualyty Detector)
24	Báo chuông (Ring Indicator)
25	Chọn tốc độ tín hiệu dữ liệu (Data Signal Rate Select)
	Truyền phân chia thời gian tín hiệu (Transmit Signal Element Timing)
	Chưa dùng

10.1.7 Phân loại truyền thông dữ liệu.

Thuật ngữ hiện nay phân chia thiết bị truyền thông dữ liệu thành một thiết bị đầu cuối dữ liệu DTE (Data Terminal Equipment) hoặc thiết bị truyền thông dữ liệu DCE (Data Communication Equipment). DTE chủ yếu là các máy tính và các thiết bị đầu cuối gửi và nhận dữ liệu, còn DCE là thiết bị truyền thông chẳng hạn như các modem chịu trách nhiệm về truyền dữ liệu. Lưu ý rằng tất cả mọi định nghĩa về chức năng các chân RS232 trong các bảng 10.1 và 10.2 đều xuất phát từ gốc độ của DTE.

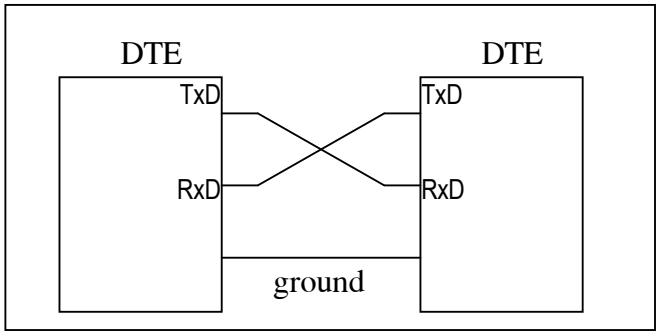
Kết nối đơn giản nhất giữa một PC và bộ vi điều khiển yêu cầu tối thiểu là những chân sau: TxD, RxD và đất như chỉ ra ở hình 10.6. Để ý rằng trên hình này thì các chân TxD và RxD được đổi cho nhau.

Hình 10.5: Sơ đồ đầu nối DB - 9 của RS232.



Bảng 10.2: Các tín hiệu của các chân đầu nối DB - 9 trên máy tính IBM PC.

Mô tả	Số chân	
1	Da ta carrier detect (DCD)	Tránh tín hiệu mạng dữ liệu
2	Received data (RxD)	Dữ liệu được nhận
3	Transmitted data (TxD)	Dữ liệu được gửi
4	Data terminal ready (DTR)	Đầu dữ liệu sẵn sàng
5	Signal ground (GND)	Đất của tín hiệu
6	Data set ready (DSR)	Dữ liệu sẵn sàng
7	Request to send (RTS)	Yêu cầu gửi
8	Clear to send (CTS)	Xoá để gửi
9	Ring indicator (RL)	Báo chuông



Hình 10.6: Nối kết không modem.

10.1.8 Kiểm tra các tín hiệu bắt tay của RS232.

Để bảo đảm truyền dữ liệu nhanh và tin cậy giữa hai thiết bị thì việc truyền dữ liệu phải được phối hợp tốt. Chẳng hạn như trong trường hợp của máy in, do một thực tế là trong truyền thông dữ liệu nối tiếp thiết bị thu có thể không có chỗ để chứa dữ liệu, do đó phải có cách để báo cho bên phát dừng gửi dữ liệu. Rất nhiều chân của RS232 được dùng cho các tín hiệu bắt tay. Dưới đây là mô tả về chúng như là một tham khảo và chúng có thể được bỏ qua vì chúng không được hỗ trợ bởi chíp UART của 8051.

1. Đầu dữ liệu sẵn sàng DTR: Khi thiết bị đầu cuối (hoặc một cổng COM của PC) được bật thì sau khi tự kiểm tra nó gửi một tín hiệu DTR báo rằng nó sẵn sàng cho truyền thông. Nếu có một cái gì đó trục trặc với cổng COM thì tín hiệu này không được kích hoạt. Đây là tín hiệu tích cực mức thấp và có thể được dùng để báo cho modem biết rằng máy tính đang hoạt động và đang sẵn sàng. Đây là chân đầu ra từ DTC (cổng COM của PC) và chân đầu ra của modem.
2. Dữ liệu sẵn sàng QSR: Khi DCE (chẳng hạn modem) được bật lên và đã chạy xong chương trình tự kiểm tra thì nó đòi hỏi DSR để báo rằng có đã sẵn sàng cho truyền thông. Do vậy, nó là đầu ra của modem (DCE) và đầu vào của PC (DTE). Đây là tín hiệu tích cực mức thấp. Nếu vì lý do nào đó mà modem không kích hoạt báo cho PC biết (hoặc thiết bị đầu cuối) rằng nó không thể nhận hoặc gửi dữ liệu.
3. Yêu cầu gửi RTS: Khi thiết bị DTE (chẳng hạn một PC) có một byte dữ liệu cần gửi thì nó yêu cầu RTS để báo cho modem biết rằng nó có một byte cần phải gửi đi. RTS là một đầu ra tích cực mức thấp từ DTE và một đầu vào tới modem.
4. Tín hiệu xáo để gửi CTS: Để đáp lại RTS thì khi modem có để chứa dữ liệu mà nó cần nhận thì nó gửi một tín hiệu CTS tới DTE (PC) để báo rằng bây giờ nó

có thể nhận dữ liệu. Tín hiệu đầu vào này tới DTE dùng để khởi động việc truyền dữ liệu.

5. Tách tín hiệu mang dữ liệu DCD: Modem yêu cầu tín hiệu DCD báo cho DTE biết rằng đã tách được một tín hiệu mang dữ liệu hợp lệ và rằng kết nối giữa nó và modem khác đã được thiết lập. Do vậy, DCD là một đầu ra của modem và đầu vào của PC (DTE).
6. Báo chuông RI: Một đầu ra từ modem (DCE) và một đầu vào tới máy tính PC (DTE) báo rằng điện thoại đang báo chuông. Nó tắt và bật đồng bộ với âm thanh đang đổ chuông. Trong 6 tín hiệu bắt tay thì tín hiệu này là ít được dùng nhất do một thực tế là các modem đã chịu trách nhiệm về trả lời điện thoại. Tuy nhiên, nếu trong một hệ thống đã cho mà PC phải chịu trách nhiệm trả lời điện thoại thì tín hiệu này có thể được dùng.

Từ mô tả trên thì việc truyền thông PC và modem có thể được tóm tắt như sau: Trong khi các tín hiệu DTR và DSR được dùng bởi PC và modem để báo rằng chúng đang hoạt động tốt thì các tín hiệu RTS và CTS thực tế đang kiểm tra luồng dữ liệu. Khi PC muốn gửi dữ liệu thì nó yêu cầu RTS và đáp lại, nếu modem sẵn sàng (có chỗ chứa dữ liệu) để nhận dữ liệu thì nó gửi lại tín hiệu CTS. Còn nếu không có chỗ cho dữ liệu thì modem không kích hoạt CTS và PC thôi không yêu cầu DTR và thử lại. Các tín hiệu RTS và CTS cũng được coi như tín hiệu luồng điều khiển phần cứng.

Đến đây kết thúc sự mô tả 9 chân quan trọng nhất của các tín hiệu bắt tay RS232 và các tín hiệu TxD, RxD và đất (Ground). Tín hiệu Ground này cũng được coi như là tín hiệu SG - đất của tín hiệu.

10.1.9 Các cổng COM của IBM PC và tương thích.

Các máy tính IBM PC và tương thích dựa trên các bộ vi xử lý × 86 (8086, 286, 384, 486 và Pentium) thường có hai cổng COM. Cả hai cổng COM đều có các đầu nối kiểu RS232. Nhiều máy tính PC sử dụng mỗi đầu nối một kiểu ổ cắm DB - 25 và DB - 9. Trong những năm gần đây, cổng COM1 được dùng cho chuột và COM2 được dùng cho các thiết bị chẳng hạn như Modem. Chúng ta có thể nối cổng nối tiếp của 8051 đến cổng COM2 của một máy tính PC cho các thí nghiệm về truyền thông nối tiếp.

Với nền kiến thức về truyền thông nối tiếp này chúng ta đã sẵn sàng làm việc với 8051.

10.2 Nối ghép 8051 tới RS232.

Như đã nói ở phần 10.1, chuẩn RS232 không tương thích với mức lô-gíc TTL, do vậy nó yêu cầu một bộ điều khiển đường truyền chẳng hạn như chíp MAX232 để chuyển đổi các mức điện áp RS232 về các mức TTL và ngược lại. Nội dung chính của phần này là bàn về nối ghép 8051 với các đầu nối RS232 thông qua chíp MAX232.

10.2.1 Các chân RxD và TxD trong 8051.

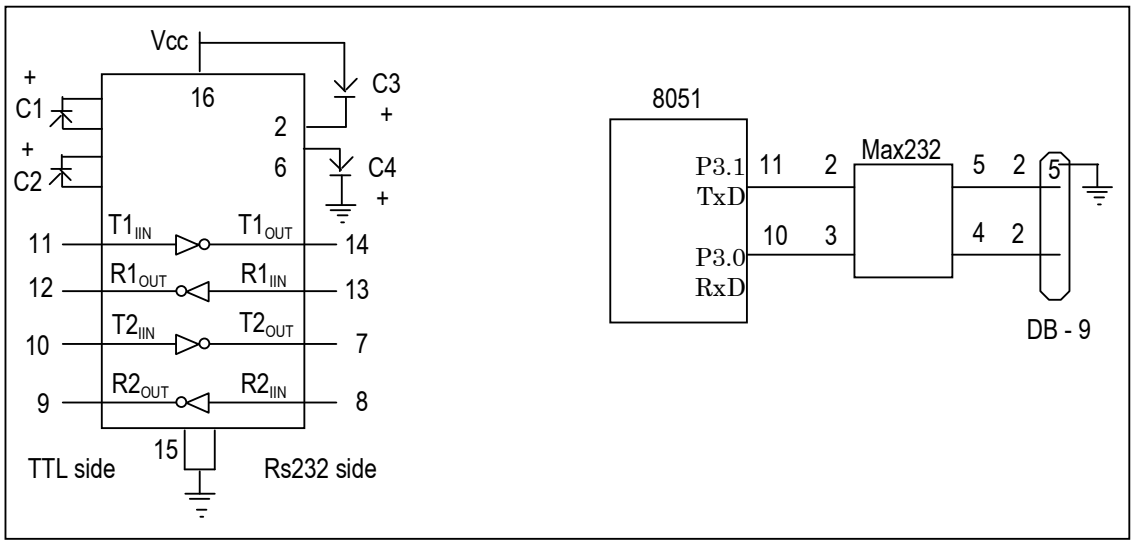
8051 có hai chân được dùng chuyên cho truyền và nhận dữ liệu nối tiếp. Hai chân này được gọi là TxD và RxD và là một phần của cổng P3 (đó là P3.0 và P3.1). chân 11 của 8051 là P3.1 được gán cho TxD và chân 10 (P3.0) được dùng cho RxD. Các chân này tương thích với mức lô-gích TTL. Do vậy chúng đòi hỏi một bộ điều khiển đường truyền để chúng tương thích với RS232. Một bộ điều khiển như vậy là chíp MAX232.

10.2.2 Bộ điều khiển đường truyền MAX232.

Vì RS232 không tương thích với các bộ vi xử lý và vi điều khiển hiện nay nên ta cần một bộ điều khiển đường truyền (bộ chuyển đổi điện áp) để chuyển đổi các tín hiệu RS232 về các mức điện áp TTL sẽ được chấp nhận bởi các chân TxD và RxD của 8051. Một ví dụ của một bộ chuyển đổi như vậy là chíp MAX232 từ hàng Maxim địa chỉ Website của hãng www.maxim-ic.com. Bộ MAX232 chuyển đổi từ các mức điện

áp RS232 sẽ về mức điện áp TTL và ngược lại. Một điểm mạnh của chíp MAX232 là nó dùng điện áp nguồng +5v cùng với điện áp nguồn của 8051. Hay nóic cách khác với nguồn điện áp nuối +5 chúng ta mà có thể nuôi 8051 và MAX232 mà không phải dùng hai nguồn nuôi khác nhau như phổ biến trong các hệ thống trước đây.

Bộ điều khiển MAX232 có hai bộ điều khiển thường để nhận và truyền dữ liệu như trình bày trên hình 10.7. Các bộ điều khiển đường được dùng cho TxD được gọi là T1 và T2. Trong nhiều ứng dụng thì chỉ có một cặp được dùng. Ví dụ T1 và R1 được dùng với nhau đối với TxD và RxD của 8051, còn cặp R2 và T2 thì chưa dùng đến. Để ý rằng trong MAX232 bộ điều khiển T1 có gán $T1_{in}$ và $T1_{out}$ trên các chân số 11 và 1 tương ứng. Chân $T1_{in}$ là ở phía TTL và được nối tới chân RxD của bộ vi điều khiển, còn $T1_{out}$ là ở phía RS232 được nối tới chân RxD của đầu nối DB của RS232. Bộ điều khiển đường R1 cũng có gán $R1_{in}$ và $R1_{out}$ trên các chân số 13 và 12 tương ứng. Chân $R1_{in}$ (chân số 13) là ở phía RS232 được nối tới chân TxD của đầu nối DB của RS232 và chân $R1_{out}$ (chân số 12) là ở phía TTL mà nó được nối tới chân RxD của bộ vi điều khiển, xem hình 10.7. Để ý rằng nối ghép modem không là nối ghép mà chân TxD bên phát được nối với RxD của bên thu và ngược lại.



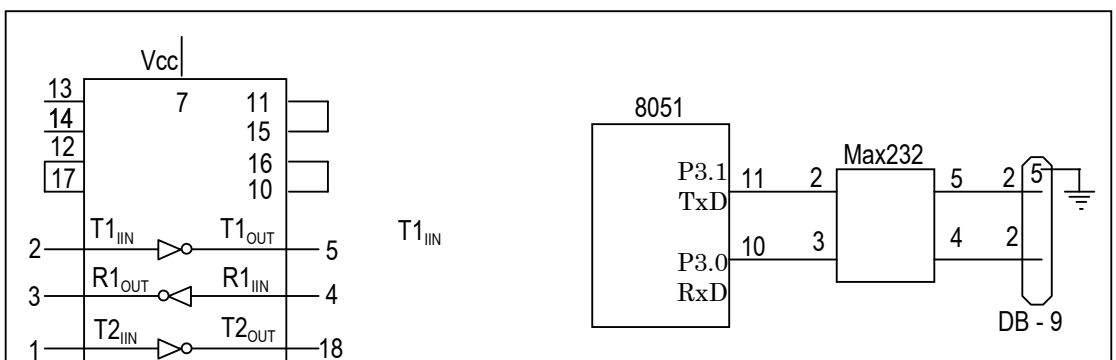
Hình 10.7: a) Sơ đồ bên trong của MAX232

b) Sơ đồ nối ghép của MAX232 với 8051 theo moden không.

Bộ MAX232 đòi hỏi 4 tụ điện giá trị từ 1 đến 22µF. Giá trị phổ biến nhất cho các tụ này là 22µF.

10.2.3 Bộ điều khiển MAX232.

Để tiết kiệm không gian trên bảng mạch, nhiều nhà thiết kế sử dụng chíp MAX232 từ hãng Maxim. Bộ điều khiển MAX232 thực hiện cùng những công việc như MAX232 lại không cần đến các tụ điện. Tuy nhiên, chíp MAX232 lại đắt hơn rất nhiều so với MAX233 không có sơ đồ chân giống nhau (không tương thích). Chúng ta không thể lấy một chíp MAX232 ra khỏi một bảng mạch và thay vào đó RS233. Hãy xem hình 10.8 để thấy MAX233 không cần đến tụ.



Hình 10.8: a) Sơ đồ bên trong của MAX233.
b) Sờ đồ nối ghép của MAX233 với 8051 theo modem không.

10.3 Lập trình truyền thông nối tiếp cho 8051.

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 10.3. Chúng ta có thể kiểm tra các tốc độ baud này bằng cách vào chương trình Windows Terminal và bấm chuột lên tuỳ chọn Communication Settings. Chương trình Terminal.exe của Window3.1 cũng làm việc tốt trên Windows95 và Window98. Trong Window95 và cao hơn ta có thể sử dụng chức năng Hyperterrminal. Hàm Hyperterminal hỗ trợ các tốc độ Baud cao hơn nhiều so với các tốc độ cho trong bảng 10.3.

Bảng 10.3: Các tốc độ Baud của máy tính PC486 và Pentium cho trong BIOS.

100	150	300	600	1200	2400	4800	9600	19200

Ví dụ 10.1:

Với tần số XTAL là 11.0592MHz. Hãy tìm giá trị TH1 cần thiết để có tốc độ baud sau:

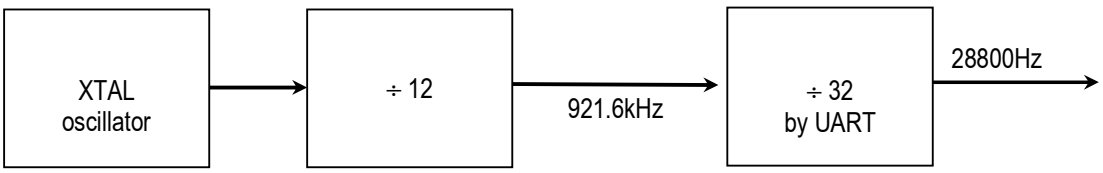
a) 9600 b) 2400 c) 1200

Lời giải:

Với tần số XTAL là 11.0592MHz thì ta có tần số chu trình máy của 8051 là 11.0592MHz : 12 = 921.6kHz và sau đó lấy 921.6kHz/32 = 28.800Hz là tần số được cấp bởi UART tới bộ định thời Timer1 để thiết lập tốc độ.

a) 28.800/3 = 9600 trong đó - 3 = FD được nạp vào TH1
b) 28.800/12 = 2400 trong đó - 12 = F4 được nạp vào TH1
c) 28.800/24 = 1200 trong đó - 24 = F8 được nạp vào TH1

Lưu ý rằng việc chia 1/12 của tần số thạch anh cho 32 là giá trị mặc định khi kích hoạt chân RESET của 8051. Chúng ta có thể thay đổi giá trị cài đặt mặc định này. Điều này sẽ được giải thích ở cuối chương.



10.3.1 Tốc độ baud trong 8051.

8051 truyền và nhận dữ liệu nối tiếp theo nhiều tốc độ khác nhau. Tốc độ truyền của nó có thể lập trình được. Điều ày thực hiện nhờ sự trợ giúp của bộ định thời Timer1. Trước khi ta đi vào bàn cách làm điều đó như thế nào thì ta sẽ xét quan hệ giữa tần số thạch anh và tốc độ baud trong 8051.

Như ta đã nói ở chương trước đây thì 8051 chia sóo thạch anh cho 12 để lấy tần số chu trình máy. Trong trường hợp XTAL = 11.0592MHz thì tần số chu trình là 921.6kHz (11.0592MHz : 12 = 921.6kHz). Mạch điện UART truyền thông nối tiếp của 8051 lại chia tần số chu trình máy cho 32 một lần nữa trước khi nó được dùng bởi bộ định thời gian Timer1 để tạo ra tốc độ baud. Do vậy, 921.6kHz : 32 = 28.800Hz. Đây là số ta sẽ dùng trong cả phần này để tìm giá trị của Timer1 để đặt tốc độ baud. Muốn Timer1 đặt tốc độ baud thì nó phải được lập trình về chế độ làm việc mode2, đó là chế độ thanh ghi 8 bít tự động nạp lại. Để có tốc độ baud tương thích với PC ta phải nạp TH1 theo các giá trị cho trong bảng 10.3. Ví dụ 10.1 trình bày cách kiểm tra giá trị dữ liệu cho trong bảng 10.3.

Bảng 10.3: Các giá trị của thanh ghi TH1 trong Timer1 cho các tốc độ baud khác nhau.

Tốc độ baud	TH1 (thập phân)	TH1 (số Hex)
9600	- 3	FD
4800	- 6	FA
2400	- 12	F4
1200	- 24	F8

10.3.2 Thanh ghi SBUF.

SBUF là thanh ghi 8 bít được dùng riêng cho truyền thông nối tiếp trong 8051. Đối với một byte dữ liệu cần phải được truyền qua đường TxD thì nó phải được đặt trong thanh ghi SBUF. Tương tự như vậy SBUF giữ một byte dữ liệu khi nó được nhận bởi đường RxD của 8051. SBUF có thể được truy cập bởi mọi thanh ghi bất kỳ trong 8051. Xét một ví dụ dưới đây để thấy SBUF được truy cập như thế nào?

```
MOV    SBUF, # "D"     ; Nạp vào SBUF giá trị 44H mã ACSII của ký tự D.
MOV    SBUF, A         ; Sao thanh ghi A vào SBUF.
MOV    A, SBUF         ; Sao SBUF vào thanh ghi A.
```

Khi một byte được ghi vào thanh ghi SBUF nó được đóng khung với các bít Start và Stop và đường truyền nối tiếp quan chân TxD. Tương tự như vậy, khi các bít được nhận nối tiếp từ RxD thì 8051 mở khung nó để loại trừ các bít Start và Stop để lấy ra một byte từ dữ liệu nhận được và đặt nó vào thanh ghi SBUF.

10.3.3 Thanh ghi điều khiển nối tiếp SCON.

Thanh ghi SCON là thanh ghi 8 bít được dùng để lập trình việc đóng khung bít bắt đầu Start, bít dừng Stop và các bít dữ liệu cùng với việc khác.

Dưới đây là mô tả các bít khác nhau của SCON:

SM0	SM1	SM2	REN	TB8	RB8	T1	R1

SM0	SCON.7	Số xác định chế độ làm việc cổng nối tiếp
SM1	SCON.6	Số xác định chế độ làm việc cổng nối tiếp
SM2	SCON.5	Dùng cho truyền thông giữa các bộ vi xử lý (SM2 = 0)
REN	SCON.4	Bật/xoá bằng phần mềm để cho phép/ không cho thu
TB8	SCON.3	Không sử dụng rộng rãi
RB8	SCON.2	Không sử dụng rộng rãi
T1	SCON.1	Cờ ngắt truyền, đặt bằng phần cứng khi bắt đầu bít Stop ở chế

Hình 10.2: Thanh ghi điều khiển cổng nối tiếp SCON.

10.3.3.1 Các bít SM0, SM1.

Đây là các bít D7 và D6 của thanh ghi SCON. Chúng được dùng để xác định chế độ đóng khung dữ liệu bằng cách xác định số bít của một ký tự và các bít Start và Stop. Các tổ hợp của chúng là:

SM0	*SM1*	
0	0	Chế độ nối tiếp 0
0	1	Chế độ nối tiếp 1, 8 bít dữ liệu, Start, Stop
1	0	Chế độ nối tiếp 2
1	1	Chế độ nối tiếp 3

Trong bốn chế độ ta chỉ quan tâm đến chế độ 1, các chế độ khác được giải thích ở Appendisk A3. Trong thanh ghi SCON khi chế độ 1 được chọn thì dữ liệu được đóng khung gồm 8 bít dữ liệu, 1 bít Start, 1 bít Stop để tương thích với cổng COM của IBM PC và các PC tương thích khác. Quan trọng hơn là chế độ nối tiếp 1 cho phép tốc độ baud thay đổi và được thiết lập bởi Timer1 của 8051. Trong chế độ nối tiếp 1 thì mỗi ký tự gồm có 10 bít được truyền trong đó có bít đầu là bít Start, sau đó là 8 bít dữ liệu và cuối cùng là bít Stop.

10.3.3.2 Bít SM2.

Bít SM2 là bít D5 của thanh ghi SCON. Bít này cho phép khả năng đa xử lý của 8051 và nó nằm ngoài phạm vi trình bày của chương này. Đối với các ứng dụng của chúng ta đặt SM2 = 0 vì ta không sử dụng 8051 trong môi trường đa xử lý.

10.3.3.3 Bít REN.

Đây là bít cho phép thu (Receive Enable), bít D4 của thanh ghi SCON. Bít REN cũng được tham chiếu như là SCON.4 vì SCON là thanh ghi có thể đánh địa chỉ theo bít. Khi bít REN cao thì nó cho phép 8051 thu dữ liệu trên chân RxD của nó. Và kết quả là nếu ta muốn 8051 vừa truyền và nhận dữ liệu thì bít REN phải được đặt lên 1. Khi đặt REN thì bộ thu bị cấm. Việc đặt REN = 1 hay REN = 0 có thể đạt được bằng lệnh "SETB SCON.4" và "CLR SCON.4" tương ứng. Lưu ý rằng các lệnh này sử dụng đặc điểm đánh địa chỉ theo bít của thanh ghi SCON. Bít này có thể được dùng để khống chế mọi việc nhận dữ liệu nối tiếp và nó là bít cực kỳ quan trọng trong thanh ghi SCON.

10.3.3.4 Bít TB8 và RB8.

Bít TB8 là bít SCON.3 hay là bít D3 của thanh ghi SCON. Nó được dùng để cho chế độ nối tiếp 2 và 3. Ta đặt TB8 vì nó không được sử dụng trong các ứng dụng của mình.

Bít RB8 (bít thu 8) là bít D2 của thanh ghi SCON. Trong chế độ nối tiếp 1 thì bít này nhận một bản sao của bít Stop khi một dữ liệu 8 bít được nhận. Bít này cũng

như bít TB8 rất hiếm khi được sử dụng. Trong các ứng dụng của mình ta đặt RB8 = 0 vì nó được sử dụng cho chế độ nối tiếp 2 và 3.

10.3.3.5 Các bít TI và RI.

Các bít ngắt truyền TI và ngắt thu RI là các bít D1 và D0 của thanh ghi SCON. Các bít này là cực kỳ quan trọng của thanh ghi SCON. Khi 8051 kết thúc truyền một ký tự 8 bít thì nó bật TI để báo rằng nó sẵn sàng truyền một byte khác. Bít TI được bật lên trước bít Stop. Còn khi 8051 nhận được dữ liệu nối tiếp qua chân RxD và nó tách các bít Start và Stop để lấy ra 8 bít dữ liệu để đặt vào SBUF, sau khi hoàn tất nó bật cờ RI để báo rằng nó đã nhận xong một byte và cần phải lấy đi kẻo nó bị mất cờ RI được bật khi đang tách bít Stop. Trong các ví dụ dưới đây sẽ nói về vai trò của các bít TI và RI.

10.3.4 Lập trình 8051 để truyền dữ liệu nối tiếp.

Khi lập trình 8051 để truyền các byte ký tự nối tiếp thì cần phải thực hiện các bước sau đây:

1. Nạp thanh ghi TMOD giá trị 204 báo rằng sử dụng Timer1 ở chế độ 2 để thiết lập chế độ baud.
2. Nạp thanh ghi TH1 các giá trị cho trong bảng 10.4 để thiết lập chế độ baud truyền dữ liệu nối tiếp (với giả thiết tần số XTAL = 11.0592MHz).
3. Nạp thanh ghi SCON giá trị 50H báo chế độ nối tiếp 1 để đóng khung 8 bít dữ liệu, 1 bít Start và 1 bít Stop.
4. Bật TR1 = 1để khởi động Timer1.
5. Xoá bít TI bằng lệnh "CLR TI"
6. Byte ký tự cần phải truyền được ghi vào SBUF.
7. Bít cờ TI được hiển thị bằng lệnh "JNB TI, xx" để báo ký tự đã được truyền hoàn tất chưa.
8. Để truyền ký tự tiếp theo quay trở về bước 5.

Ví dụ 10.2 trình bày chương trình để truyền nối tiếp với tốc độ 4800 baud. Ví dụ 10.3 trình bày cách truyền liên tục chữ "YES".

Ví dụ 10.2:

Hãy viết chương trình cho 8051 để truyền nối tiếp một ký tự "A" với tốc độ 4800 baud liên tục.

Lời giải:

```
        MOV   TMOD, #20H      ; Chọn Timer1, chế độ 2 (tự động nạp lại)
        MOV   TH1, # - 6      ; Chọn tốc độ 4800 baud
        MOV   SCON, #A"       ; Truyền 8 bít dữ liệu, 1 bít Stop cho phép thu
        SETB  TR1             ; Khởi động Timer1
AGAIN:  MOV   SBUF, #"A"      ; Cần truyền ký tự "A"
HERE:   JNB   TI, HERE        ; Chờ đến bít cuối cùng
        CLR   TI              ; Xoá bít TI cho ký tự kế tiếp
        SJMP  AGAIN           ; Tiếp tục gửi lại chữ A
```

Ví dụ 10.3:

Hãy viết chương trình để truyền chữ "YES" nối tiếp liên tục với tốc độ 9600 baud (8 bít dữ liệu, 1 bít Stop).

Lời giải:

```
        MOV   TMOD, #20H      ; Chọn bộ Timer1, chế độ 2
        MOV   TH1, # - 3      ; Chọn tốc độ 9600 baud
        MOV   SCON, #50H      ; Truyền 8 bít dữ liệu, 1 bít Stop cho phép thu
        SETB  TR1             ; Khởi động Timer1
AGAIN:  MOV   A, # "Y"        ; Truyền ký tự "Y"
        ACALL TRANS
```

```
        MOV     A, # "E"            ; Truyền ký tự "E"
        ACALL   TRANS
        MOV     A, # "S"            ; Truyền ký tự "S"
        ACALL   TRANS
        SJMP    AGAIN               ; Tiếp tục
; Chương trình con truyền dữ liệu nối tiếp.
TRANS:  MOV     SBUF, A             ; Nạp SBUF
HERE:   JNB     TI, HERE            ; Chờ cho đến khi truyền bít cuối cùng
        CLR     TI                  ; Chờ sẵn cho một byte kế tiếp
        RET
```

10.3.4.1 Tầm quan trọng của cờ TI.

Để hiểu tầm quan trọng của cờ ngắt TI ta hãy xét trình tự các bước dưới đây mà 8051 phải thực hiện khi truyền một ký tự quan đường TxD:

1. Byte ký tự cần phải truyền được ghi vào SBUF.
2. Truyền bít Start
3. Truyền ký tự 8 bít lần lượt từng bít một.
4. Bít Stop được truyền xong, trong quá trình truyền bít Stop thì cờ TI được bật (TI = 1) bởi 8051 để báo sẵn sàng để truyền ký tự kế tiếp.
5. Bằng việ hiển thị cờ TI ta biết chắc rằng ta không nạp quá vào thanh ghi SBUF. Nếu ta nạp một byte vào SBUF trước ghi TI được bật thì phần dữ liệu của byte trước chưa truyền hết sẽ bị mất. Hay nói cách khác là 8051 bật cờ TI khi đã truyền xong một byte và nó sẵn sàng để truyền byte kế tiếp.
6. Sau khi SBOF được nạp một byte mới tì cờ nhằm để có thể truyền byte mới này.

Từ phần trình bày trên đây ta kết luận rằng bằng việc kiểm tra bít cờ ngắt TI ta biết được 8051 có sẵn sàng để truyền một byte khác không. Quan trọng hơn cần phải nói ở đây là bít cờ TI được bật bởi từ 8051 khi nó hoàn tất việc truyền một byte dữ liệu, còn việc xoá nó thì phải được lập trình viên thực hiện bằng lệnh "CLR TI". Cũng cần lưu ý rằng, nếu ta ghi một byte vào thanh ghi SBUF trước khi cờ TI được bật thì sẽ có nguy cơ mất phần dữ liệu đang truyền. Bít cờ TI có thể kiểm tra bằng lệnh "JNB TI …" hoặc có thể sử dụng ngắt như ta sẽ thấy trong chương 11.

10.3.5 Lập trình 8051 để nhận dữ liệu.

Trong lập trình của 8051 để nhận các byte ký tự nối tiếp thì phải thực hiện các bước sau đây.

1. Nạp giá trị 20H vào thanh ghi TMOD để báo sử dụng bộ Timer1, chế độ 2 (8 bítm, tự động nạp lại) để thiết lập tốc độ baud.
2. Nạp TH1 các giá trị cho trong bảng 10.4 để tạo ra tốc độ baud với giả thiết XTAL = 10.0592MHz.
3. Nạp giá trị 50H vào thanh ghi SCON để báo sử dụng chế độ truyền nối tiếp 1 là dữ liệu được đóng gói bởi 8 bít dữ liệu, 1 bít Start và 1 bít Stop.
4. Bật TR1 = 1 để khởi động Timer1.
5. Xoá cờ ngắt RI bằng lệnh "CLR RI"
6. Bít cờ RI được hiển thị bằng lệnh "JNB RI, xx" để xem toàn bộ ký tự đã được nhận chưa.
7. Khi RI được thiết lập thì trong SBUF đã có 1 byte. Các nội dung của nó được cất lưu vào một nơi an toàn.
8. Để nhận một ký tự tiếp theo quay trở về bước 5.

Ví dụ 10.4:

Hãy lập trình cho 8051 để nhận các byte dữ liệu nối tiếp và đặt chúng vào cổng P1. Đặt tốc độ baud là 4800, 8 bít dữ liệu và 1 bít Stopl.

Lời giải:

```
        MOV     TMOD, #20H          ; Chọn bộ Timer1, chế độ 2 (tự động nạp lại)
        MOV     TH1, # - 6          ; Chọn tốc độ 4800 baud
        MOV     SCON, #50H          ; Chọn khung dữ liệu 8 bít Stop, bít.
        SETB    TR1                 ; Khởi động bộ Timer1
HERE:   JNB     R1, HERE            ; Đợi nhận toàn bộ lý tự vào hết
        MOV     A, SBUF             ; Lưu cất ký tự vào thanh A
        MOV     P1, A               ; Gửi ra cổng P.1
        CLR     RI                  ; Sẵn sàng nhận byte kế tiếp
        SJMP    HERE                ; Tiếp tục nhạn dữ liệu
```

Ví dụ 10.5:

Giả sử cổng nối tiếp của 8051 được nối vào cổng COM của máy tính IBM CP và mà đang sử chương trình Termina. Exe để gửi và nhận dữ liệu nối tiếp. Cổng P1 và P2 của 8051 được nố tới các đầ LED và các công tắc chuyển mạch tương ứng. Hãy viết một chương trrình cho 8051.

a) Gửi thông báo "We Are Ready" (chúng tôi đã sẵn sàng) tới máy tính PC.

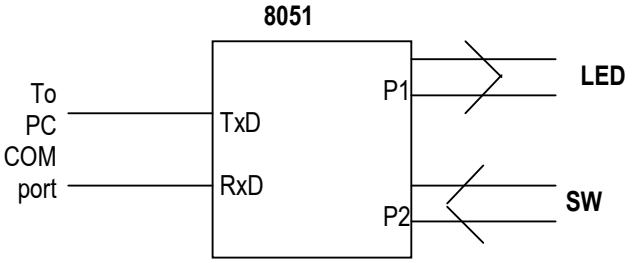
b) Nhận bất kỳ dữ liệu gì được PC gửi đến và chuyển đến các đèn LED đang nối đến các chân của cổng P1.

c) Nhận dữ liệu trên các chuyển mạch được nối tới P2 và gửi nó tới máy tính PC nối tiếp. Chương trình phải thực hiện một lần a), nhưng b) và c) chạy liên tục với tốc độ 4800 baud.

Lời giải:

```
                ORG     0
                MOV     P2, #0FFH           ; Lấy cổng P2 làm cổng vào
                MOV     TMOD, #20H          ; Chọn bộ Timer1, chế độ 2 (tự động nạp lại)
                MOV     TH1, # 0FAH         ; Chọn tốc độ 4800 baud
                MOV     SCON, #50H          ; Tạo khung dữ liệu 8 bít, 1bít Stop cho phép
                                              REN.
                SETB    TR1                 ; Khởi động bộ Timer1
                MOV     DPTR, #MYDATA       ; Nạp con trỏ đến thông báo
H - 1:          CLR     A
                MOVC    A, 'A + DPTR        ; Lấy ký tự
                JZ      DPTR                ; Nếu ký tự cuối cùng muốn gửi ra
                ACALL   SEND                ; Nếu chưa thì gọi chương trình con SEND
                INC     DPTR                ; Chạy tiếp
                SJMP    H - 1               ; Quay lại vòng lặp
B - 1:          MOV     A, P2               ; Đọc dữ liệu trên cổng P2
                ACALL   RECV                ; Truyền nó nối tiếp
                ACALL   RECV                ; Nhận dữ liệu nối tiếp
                MOV     F1, A               ; Hiển thị nó ra các đền LED
                SJMP    B - 1               ; ở lại vòng lặp vô hạn
; ------------------ Truyền dữ liệu nối tiếp ACC có dữ liệu
SEND:           MOV     SBUF, A             ; Nạp dữ liệu
H- 2:           JNB     TI, H - 2           ; ở lại vòng lặp vô hạn
                CLR     TI                  ; Truyền dữ liệu nối tiếp
                RET                         ; Nhận dữ liệu
; ------------------ Truyền dữ liệu nối tiếp ACC có dữ liệu
RECV:           JNB     RI, RECV            ; Nạp dữ liệu
                MOV     A, SBUF             ; ở lại đây cho đến khi gửi bít cuối cùng
                CLR     RI                  ; Sẵn sàng cho ký tự mới
                RET                         ; Trở về mời gọi
; ------------------ Nhận dữ liệu nối tiếp trong ACC
RECV:           JNB     RI, RECV            ; Đợi ở đây nhận ký tự
                MOV     A, SBUF             ; Lưu nó vào trong ACC
```

```
                CLR     RI                      ; Sẵn sàng nhận ký tự mã tiếp theo
                RET                             ; Trở về nơi gọi
; ------------------- Ngăn xếp chưa thông báo
MYDATA:         DB      "Chúng tôi đã sẵn sàng" 0
                END
```



10.3.5.1 Tầm quan trọng của cờ RT.

Khi nhận các bít quan chân RxD của nó thì 8051 phải đi quan các bước sau:

1. Nó nhận bít Start báo rằng bít sau nó là bít dữ liệu đầu tiên cần phải nhận.
2. Ký tự 8 bít được nhận lần lượt từng bít một. Khi bít cuối cùng được nhận thì một byte được hình thành và đặt vào trong SBUF.
3. Khi bít Stop được nhận thì 8051 bật RT = 1 để báo rằng toàn bộ ký tự được nhận và phải lấy đi trước khi nó bị byte mới nhận về ghi đè lên.
4. Bằng việc kiểm tra bít cờ RI khi nó được bật lên chúng ta biết rằng một ký tự đã được nhận và đang nằm trong SBUF. Tại sao nội dung SBUF vào nơi an toàn trong một thanh ghi hay bộ nhớ khác trước khi nó bị mất.
5. Sau khi SBUF được ghi vào nơi an toàn thì cờ RI được xoá về 0 bằng lệnh "CLR RI" nhằm cho các ký tự kế tiếp nhận được đưa vào SBUF. Nếu không làm được điều này thì gây ra mất ký tự vừa nhận được.

Từ mô tả trên đây ta rút ra kết luận rằng bằng việc kiểm tra cờ RI ta biết 8051 đã nhận được một byte ký tự chưa hay rồi. Nếu ta không sao được nội dung của thanh ghi SBUF vào nơi an toàn thì có nguy cơ ta bị mất ký tự vừa nhận được. Quan trọng hơn là phải nhớ rằng cờ RI được 8051 bật lên như lập trình viên phải xoá nó bằng lệnh "CLR RI". Cũng nên nhờ rằng, nếu ta sao nội dung SBUF vào nơi an toàn trước khi RI được bật ta mạo hiểm đã sao dữ liệu chưa đầy đủ. Bít cờ RI có thể được kiểm tra bởi lệnh "JNB RI, xx" hoặc bằng ngắt sẽ được bàn ở chương 11.

10.3.6 Nhân đôi tốc độ baud trong 8051.

Có hai cách để tăng tốc độ baud truyền dữ liệu trong 8051.

1. Sử dụng tần số thạch anh cao hơn.
2. Thay đổi một bít trong thanh ghi điều khiển công suất PCON (Power Control) như chỉ ra dưới đây.

D7							D0
SM0D	–	–	–	GF0	GF0	PD	IDL

Phương án một là không thực thi trong nhiều trường hợp vì tần số thạch anh của hệ thống là cố định. Quan trọng hơn là nó không khả thi vì tần số thạch anh mới không tương thích với tốc độ baud của các cổng COM nối tiếp của IBM PC. Do vậy, ta sẽ tập trung thăm dò phương án hai, có một cách nhân đôi tần số baud bằng phần mềm trong 8051 với tần số thạch anh không đổi. Điều này được thực hiện nhờ thanh ghi PCON, đây là thanh ghi 8 bít. Trong 8 bít này thì có một số bít không được dùng để điều khiển công suất của 8051. Bít dành cho truyền thông là D7, bít SMOD (chế độ nối tiếp - serial mode). Khi 8051 được bật nguồn thì bít SMOD của thanh ghi PCON ở

mức thấp 0. Chúng ta có thể đặt nó lên 1 bằng phần mềm và do vậy nhân đôi được tốc độ baud. Thứ tự các lệnh được sử dụng để thiết lập bít D7 của PCON lên cao như sau (thanh ghi PCON là thể đánh địa chỉ theo bít).

```
MOV   A, PCON          ; Đặt bản sao của PCON vào ACC
SETB  ACC.7            ; Đặt D7 của ACC lên 1.
MOV   PCON, A          ; Bây giờ SMOD = 1 mà không thay đổi bất kỳ bít nào khác.
```

Để biết tốc độ baud được tăng lên gấp đôi như thế nào bằng phương pháp này ta xét vai trò của bít SMOD trong PCON khi nó là 0 và 1.

a) Khi SMOD = 0.

Khi SMOD = 0 thì 8051 chia 1/12 tần số thạch anh cho 32 và sử dụng nó cho bộ Timer1 để thiết lập tốc độ baud. Trong trường hợp XTAL = 11.0592MHz thì ta có: Tần số chu trình máy $= \frac{11.0592\text{MHz}}{12} = 921.6\text{kHz}$ và $\frac{921.6\text{kHz}}{32} = 28.800\text{Hz}$ vì SMOD = 0.

Đây là tần số được Timer1 sử dụng để đặt tốc độ baud. Đây là cơ sở cho tất cả ví dụ từ trước đến giờ vì nó là giá trị mặc định của 8051 khi bật nguồn. Các tốc độ baud đối với SMOD = 0 được cho trong bảng 10.4.

b) Khi SMOD = 1.

Với tần số cố định thạch anh ta có thể nhân đôi tốc độ baud bằng cách đặt bít SMOD = 1. Khi bít D7 của PCON (bít SMOD) được đưa lên 1 thì 1/12 tần số XTAL được chia cho 16 (thay vì chia cho 32 như khi SMOD = 0) và đây là tần số được Timer dùng để thiết lập tốc độ baud. Trong trường hợp XTAL = 11.0592MHz ta có:

Tần số chu trình máy $= \frac{11.0592\text{MHz}}{12} = 921.6\text{kHz}$ và $\frac{921.6\text{kHz}}{16} = 57.600\text{kHz}$ vì SMOD = 1.

Đây là tần số mà Timer1 dùng để đặt tốc độ baud. Bảng 10.5 là các giá trị cần được nạp vào TH1 cùng với các tốc độ baud của 8051 khi SMOD = 0 và 1.

Bảng 10.5: So sánh tốc độ baud khi SMOD thay đổi.

TH1 (thập phân)	TH1 (Hex)	Tốc độ baud	
		SMOD = 0	SMOD = 1
-3	FD	9600	19200
-6	DA	4800	9600
-12	F4	2400	4800
-24	E8	1200	2400

Ví dụ 10.6:

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 10.7:

Tìm giá trị TH1 (ở dạng thập phân và hex) để đạt tốc độ baud cho các trường hợp sau.

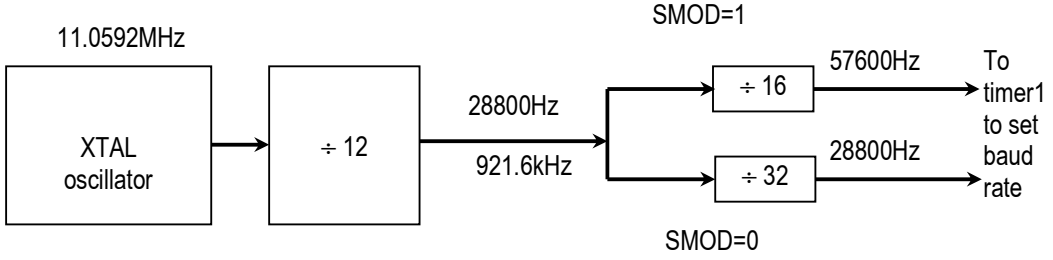
a) 9600 b) 4800 nếu SMOD = 1 và tần số XTAL = 11.0592MHz

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



Ví dụ 10.8:

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?

Lời giải:

Với tần số XTAL = 11.0592MHz và SMOD = 1 ta có tần số cấp cho Timer1 là 57.6kHz. Tốc độ baud là 57.600kHz/2 = 28.800. Tốc độ này không được hỗ trợ bởi các máy tính IBM PC và tương thích. Tuy nhiên, PC có thể được lập trình để truyền dữ liệu với tốc độ như vậy. Phần mềm của nhiều modem có thể làm cho điều này và Hyperterminal của Windows 95 cũng có thể hỗ trợ tốc độ này và các tốc độ khác nữa.

10.3.7 Truyền dữ liệudựa trên các ngắt.

Ta phải thấy rằng thật lãng phí để các bộ vi điều khiển phải bật lên xuống các cờ TI và RI. Do vậy, để tăng hiệu suất của 8051 ta có thể lập trình các cổng truyền thông nối tiếp của nó bằng các ngắt. Đây chính là nội dung chính sẽ bàn luận ở chương 11 dưới đây.

CHƯƠNG 11
Lập trình các 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.

11.1 Các ngắt của 8051.

11.1.1 Các ngắt ngược với thăm dò.

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. Nhằm để tránh ….. thì người ta sử dụng phương pháp ngắt. Ví dụ trong các bộ định thời được bàn đến ở chương 9 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.

11.1.2 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 11.1).

11.1.3 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.

11.1.4 Sáu ngắt trong 8051.

Thực tế chỉ có 5 ngắt dành cho người dùng trong 8051 nhưng nhiều nhà sản xuất đưa ra các bảng dữ liệu nói rằng có sáu ngắt vì họ tính cả lệnh tái thiết lập lại RESET. Sáu ngắt của 8051 được phân bố như sau:

1. RESET: Khi chân RESET được kích hoạt từ 8051 nhảy về địa chỉ 0000. Đây là địa chỉ bật lại nguồn được bàn ở chương 4.
2. Gồm hai ngắt dành cho các bộ định thời: 1 cho Timer0 và 1 cho Timer1. Địa chỉ của các ngắt này là 000B4 và 001B4 trong bảng véc tơ ngắt dành cho Timer0 và Timer1 tương ứng.
3. Hai ngắt dành cho các ngắt phần cứng bên ngoài chân 12 (P3.2) và 13 (P3.3) của cổng P3 là các ngắt phần cứng bên ngoài INT0 và INT1 tương ứng. Các ngắt ngoài cũng còn được coi như EX1 và EX2 vị trí nhớ trong bảng véc tơ ngắt của các ngắt ngoài này là 0003H và 0013H gán cho INT0 và INT1 tương ứng.
4. Truyền thông nối tiếp có một ngắt thuộc về cả thu và phát. Địa chỉ của ngắt này trong bảng véc tơ ngắt là 0023H.

Chú ý rằng trong bảng 11.1 có một số giới hạn các byte dành riêng cho mỗi ngắt. Ví dụ, đối với ngắt INT0 ngắt phần cứng bên ngoài 0 thì có tổng cộng là 8 byte từ địa chỉ 0003H đến 000AH dành cho nó. Tương tự như vậy, 8 byte từ địa chỉ 000BH đến 0012H là dành cho ngắt bộ định thời 0 là TI0. Nếu trình phục vụ ngắt đối mặt với một ngắt đã cho mà ngắn đủ đặt vừa không gian nhớ được. Nếu không vừa thì một lệnh LJMP được đặt vào trong bảng véc tơ ngắt để chỉ đến địa chỉ của ISR, ở trường hợp này thì các byte còn lại được cấp cho ngắt này không dùng đến. Dưới đây là các ví dụ về lập trình ngắt minh hoạ cho các điều trình bày trên đây.

Từ bảng 11.1 cùng để ý thấy một thực tế rằng chí có 3 byte của không gian bộ nhớ ROM được gán cho chân RESET. Đó là những vị trí địa chỉ 0, 1 và 2 của ROM. Vị trí địa chỉ 3 thuộc về ngắt phần cứng bên ngoài 0 với lý do này trong chương trình chúng ta phaỉ đặt lệnh LJMP như là lệnh đầu tiên và hướng bộ xử lý lệnh khỏi bảng véc tơ ngắt như chỉ ra trên hình 11.1.

Bảng 11.1: Bảng véc tơ ngắt của 8051.

Ngắt	Địa chỉ ROM	Chân
Bật lại nguồn (RESET)	0000	9
Ngắt phần cứng ngoài (INT0)	0003	12 (P3.2)
Ngắt bộ Timer0 (TF0)	000B	
Ngắt phần cứng ngoài 1 (INT1)	0013	13 (P3.3)
Ngắt bộ Timer1 (TF1)	001B	
Ngắt COM nối tiếp (RI và TI)	0023	

11.1.5 Cho phép và cấm ngắt.

Khi bật lại nguồn thì tất cả mọi ngắt đều bị cấm (bị che) có nghĩa là không có ngắt nào sẽ được bộ vi điều khiển đáp ứng nếu chúng được kích hoạt. Các ngắt phải được kích hoạt bằng phần mềm để bộ vi điều khiển đáp ứng chúng. Có một thanh ghi được gọi là cho phép ngắt IE (Interrupt Enable) chịu trách nhiệm về việc cho phép (không che) và cấm (che) các ngắt. Hình 11.2 trình bày thanh ghi IE, lưu ý rằng IE là thanh ghi có thể đánh địa chỉ theo bít.

Từ hình 11.2 ta thấy rằng D7 của thanh ghi IE được gọi là bít cho phép tất cả các ngắt EA (Euable All). Bít này phải được thiết lập lên 1 để phần còn lại của thanh ghi hoạt động được. Bít D6 chưa được sử dụng. Bít D54 được dành cho 8051, còn bít D4 dùng cho ngắt nối tiếp v.v…

11.1.6 Các bước khi cho phép ngắt.

Để cho phép một ngắt ta phải thực hiện các bước sau:

1. Bít D7 của thanh ghi IE là EA phải được bật lên cao để cho phép các bít còn lại của thanh ghi nhận được hiệu ứng.
2. Nếu EA = 1 thì tất cả mọi ngắt đều được phép và sẽ được đáp ứng nếu các bít tương ứng của chúng trong IE có mức cao. Nếu EA = 0 thì không có ngắt nào sẽ được đáp ứng cho dù bít tương ứng của nó trong IE có giá trị cao.

Để hiểu điểm quan trong này hãy xét ví dụ 11.1.

Hình 11.2: Thanh ghi cho phép ngắt IE.

D7 D0

EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA IE.7 Nếu EA = 0 thì mọi ngắt bị cấm
Nếu EA = 1 thì mỗi nguồn ngắt được cho phép hoặc bị cấm bằng các bật hoặc xoá bít cho phép của nó.

- - IE.6 Dự phòng cho tương lai
ET2 IE.5 Cho phép hoặc cấm ngắt tràn hoặc thu của Timer2 (8051)
ES IE.4 Cho phép hoặc cấm ngắt cổng nối tiếp
ET1 IE.3 Cho phép hoặc cấm ngắt tràn của Timer1
EX1 IE.2 Cho phép hoặc cấm ngắt ngoài 1
ET0 IE.1 Cho phép hoặc cấm ngắt tràn của Timer0
EX0 IE.0 Cho phép hoặc cấm ngắt ngoài 0

* Người dùng không phải ghi 1 vào bít dự phòng này. Bít này có thể dùng cho các bộ vi điều khiển nhanh với đặc tính mới

Ví dụ 11.1:

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).

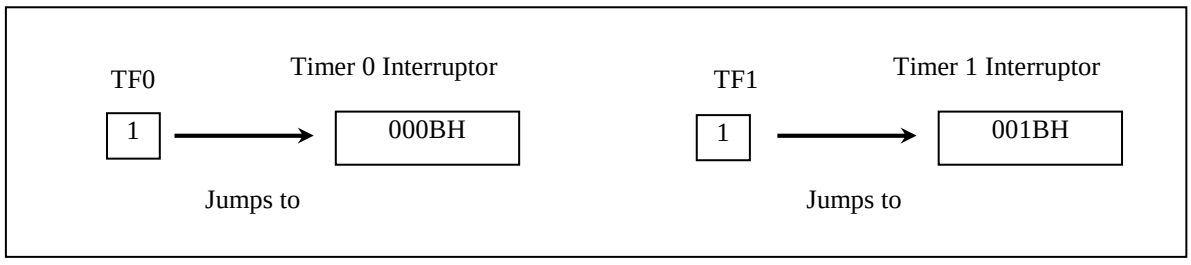
```
b) CLR  IE.1        ; Xoá (che) ngắt Timer0
c) CLR  IE.7        ; Cấm tất cả mọi ngắt.
```

11.2 Lập trình các ngắt bộ định thời.

Trong chương 9 ta đã nói cách sử dụng các bộ định thời Timer0 và Timer1 bằng phương pháp thăm dò. Trong phần này ta sẽ sử dụng các ngắt để lập trình cho các bộ định thời của 8051.

11.2.1 Cờ quay về 0 của bộ định thời và ngắt.

Trong chương 9 chúng ta đã nói rằng cờ bộ định thời TF được đặt lên cao khi bộ định thời đạt giá trị cực đại và quay về 0 (Roll - over). Trong chương trình này chúng ta cũng chỉ ra cách hiển thị cờ TF bằng lệnh "JNB TF, đích". Khi thăm dò cờ TF thì ta phải đợi cho đến khi cờ TF được bật lên. Vấn đề với phương pháp này là bộ vi điều khiển bị trói buộc khi cờ TF được bật lên và không thể làm được bất kỳ việc gì khác. Sử dụng các ngắt giải quyết được vấn đề này và tránh được sự trói buộc của bộ vi điều khiển. Nếu bộ ngắt định thời trong thanh ghi IE được phép thì mỗi khi nó quay trở về 0 cờ TF được bật lên và bộ vi điều khiển bị ngắt tại bất kỳ việc gì nó đang thực hiện và nhảy tới bảng véc tơ ngắt để phục vụ ISR. Bằng cách này thì bộ vi điều khiển có thể làm những công việc khác cho đến khi nào nó được thông báo rằng bộ định thời đã quay về 0. Xem hình 11.3 và ví dụ 11.2.



Hình 11.3: Ngắt bộ định thời TF0 và TF1.

Hãy để những điểm chương trình dưới đây của chương trình trong ví dụ 11.2.

1. Chúng ta phải tránh sử dụng không gian bộ nhớ dành cho bảng véc tơ ngắt. Do vậy, ta đặt tất cả mã khởi tạo tại địa chỉ 30H của bộ nhớ. Lệnh LJMP là lệnh đầu

tiên mà 8051 thực hiện khi nó được cấp nguồn. Lệnh LJMP lái bộ điều khiển tránh khỏi bảng véc tơ ngắt.

2. Trình phục vụ ISR của bộ Timer0 được đặt ở trong bộ nhớ bắt đầu tự địa chỉ 000BH và vì nó quá nhỏ đủ cho vào không gian nhớ dành cho ngắt này.
3. Chúng ta cho phép ngắt bộ Timer0 với lệnh "MOV IE, #1000 010H" trong chương trình chính MAIN.
4. Trong khi dữ liệu ở cổng P0 được nhận vào và chuyển liên tục sang công việc P1 thì mỗi khi bộ Timer0 trở về 0, cờ TF0 được bật lên và bộ vi điều khiển thoát ra khỏi vòng lặp BACK và đi đến địa chỉ 000BH để thực hiện ISR gắn liền với bộ Timer0.
5. Trong trình phục vụ ngắt ISR của Timer0 ta thấy rằng không cần đến lệnh "CLR TF0" trước khi lệnh RETI. Lý do này là vì 8051 xoá cờ TF bên trong khi nhảy đến bảng véc tơ ngắt.

Ví dụ 11.2:

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 200μs 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 100/1.085μs = 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 11.2 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 11.3 dưới đây.

Ví dụ 11.3:

Hãy viết lại chương trình ở ví dụ 11.2 để tạo sóng vuông với mức cao kéo dài 1085μs và mức thấp dài 15μs với giả thiết tần số XTAL = 11.0592MHz. Hãy sử dụng bộ định thời Timer1.

Lời giải:

Vì 1085μs là 1000 × 1085μs 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 = 1.085μs và 14 × 1.085μs = 15.19μs.

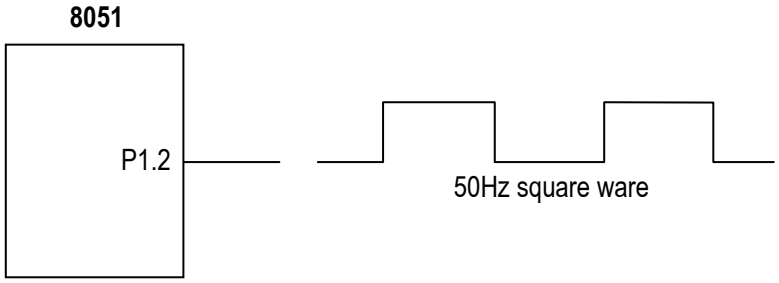
Ví dụ 11.4:

Viết một chương trình để tạo ra một sóng vuông tần số 50Hz trên chân P1.2. Ví dụ này tương tự ví dụ 9.12 ngoại trừ ngắt Timer0, giả sử XTAL = 11.0592MHz.

Lời giải:

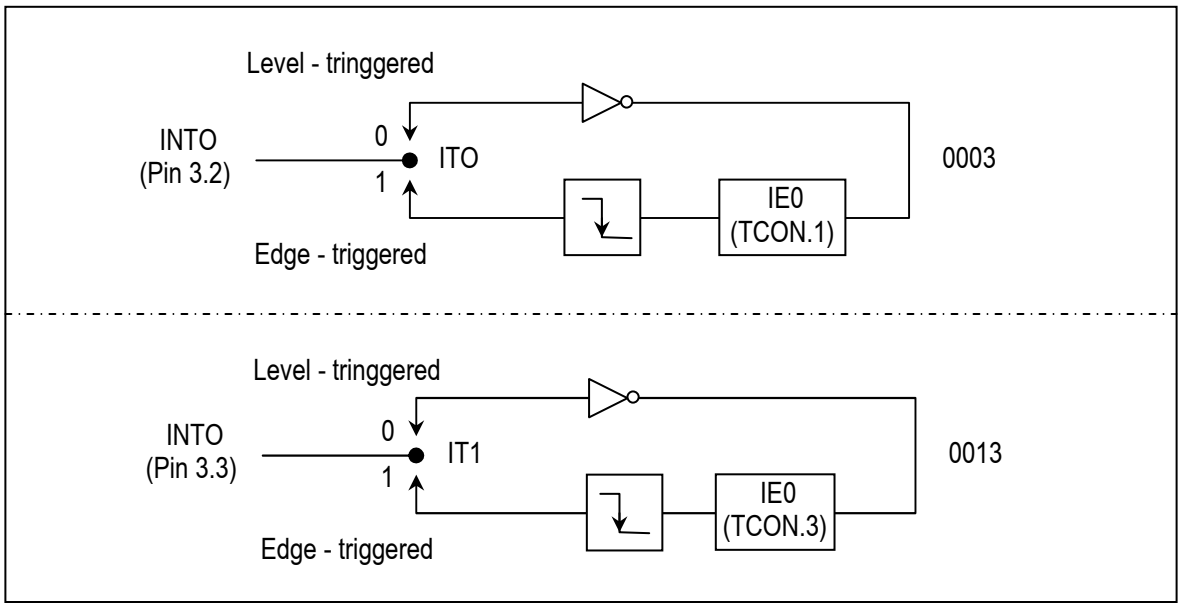
```
                ORG     0
                LJMP    MAIN
                ORG     000BH                   ; Chương trình con phục vụ ngắt cho Timer0
                CPL     P1.2
                MOV     TL0, # 00
                MOV     TH0, # 0DCH
                RETI
                ORG     30H
; ------------ main  program for initialization
MAIN:   MOV     TMOD, # 00000001B               ; Chọn Timer0 chế độ 1
```

```
        MOV   TL0, # 0DCH
        MOV   IE, # 82H          ; Cho phép ngắt Timer0
        SETB  TR0
HERE:   SJMP  HERE
        END
```



11.3 Lập trình các ngắt phần cứng bên ngoài.

Bộ vi điều khiển 8051 có hai ngắt phần cứng bên ngoài là chân 12 (P3.2) và chân 13 (P3.3) dùng cho ngắt INT0 và INT1. Khi kích hoạt những chân này thì 8051 bị ngắt tại bất kỳ công việc nào mà nó đang thực hiện và nó nhảy đến bảng véc tơ ngắt để thực hiện trình phục vụ ngắt.



11.3.1 Các ngắt ngoài INT0 và INT1.

Chỉ có hai ngắt phần cứng ngoài trong 8051 là INT0 và INT1. Chúng được bố trí trên chân P3.2 và P3.3 và địa chỉ của chúng trong bảng véc tơ ngắt là 0003H và 0013H. Như đã nói ở mục 11.1 thì chúng được ghép và bị cấm bằng việc sử dụng thanh ghi IE. Vậy chúng được kích hoạt như thế nào? Có hai mức kích hoạt cho các ngắt phần cứng ngoài: Ngắt theo mức và ngắt theo sườn. Dưới đây là mô tả hoạt động của mỗi loại.

11.3.2 Ngắt theo mức.

Ở chế độ ngắt theo mức thì các chân INT0 và INT1 bình thường ở mức cao (giống như tất cả các chân của cổng I/O) và nếu một tín hiệu ở mức thấp được cấp tới chúng thì nó ghi nhãn ngắt. Sau đó bộ vi điều khiển dừng tất cả mọi công việc nó

đang thực hiện và nhảy đến bảng véc tơ ngắt để phục vụ ngắt. Điều này được gọi là ngắt được kích hoạt theo mức hay ngắt theo mức và là chế độ ngắt mặc định khi cấp nguồn lại cho 8051. Tín hiệu mức thấp tại chân INT phải được lâýu đi trước khi thực hiện lệnh cuối cùng của trình phục vụ ngắt RETI, nếu không một ngắt khác sẽ lại được tạo ra. Hay nói cách khác, nếu tín hiệu ngắt mức thấp không được lấy đi khi ISR kết thúc thì nó không thể hiện như một ngắt khác và 8051 nhảy đến bảng véc tơ ngắt để thực hiện ISR. Xem ví dụ 11.5.

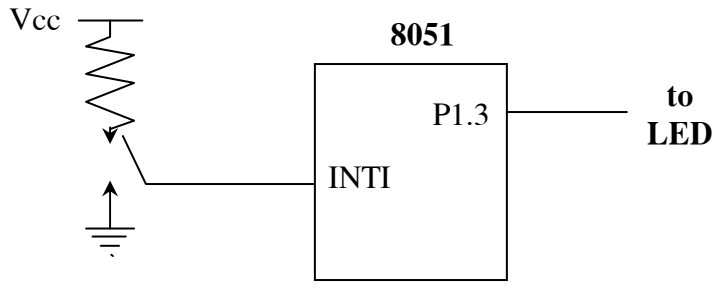
Ví dụ 11.5.

Giả sử chân INT1 được nối đến công tắc bình thường ở mức cao. Mỗi khi nó xuống thấp phải bật một đèn LED. Đèn LED được nối đến chân P1.3 và bình thường ở chế độ tắt. Khi nó được bật lên nó phải sáng vài phần trăm giây. Chừng nào công tắc được ấn xuống thấp đèn LED phải sáng liên tục.

Lời giải:

```
                ORG     0000H
                LJMP    MAIN                    ; Nhảy đến bảng véc tơ ngắt
; - - Chương trình con ISR cho ngắt cứng INT1 để bật đèn LED.
                ORG     0013H                   ; Trình phục vụ ngắt ISR cho INT1
                SETB    P1.3                    ; Bật đèn LED
                MOV     R3, # 255               ;
BACK:           DJNZ    R3, BACK                ; Giữ đèn LED sáng một lúc
                CLR     P1.3                    ; Tắt đèn LED
                RETI                            ; Trở về từ ISR
; - - Bắt đầu chương trình chính Main.
                ORG     30H
MAIN:           MOV     IE, #10000100B          ; Cho phép ngắt dài
                SJMP    HERE                    ; Chờ ở đây cho đến khi được ngắt
        END
```

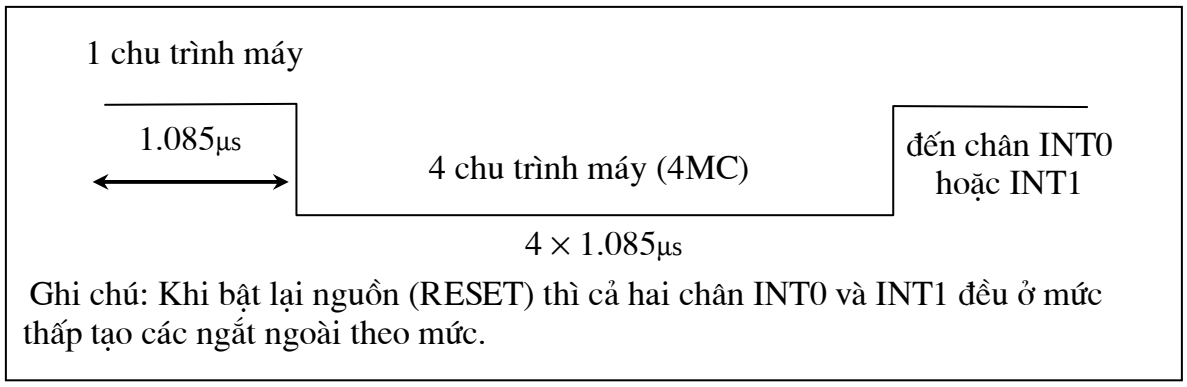
Ấn công tắc xuống sẽ làm cho đèn LED sáng. Nếu nó được giữ ở trạng thái được kích hoạt thì đèn LED sáng liên tục.



Trong chương trình này bộ vi điều khiển quay vòng liên tục trong vòng lặp HERE. Mỗi khi công tắc trên chân P3.3 (INT1) được kích hoạt thì bộ vi điều khiển thoát khỏi vòng lặp và nhảy đến bảng véc tơ ngắt tại địa chỉ 0013H. Trình ISR cho INT1 bật đèn LED lên giữ nó một lúc và tắt nó trước khi trở về. Nếu trong lúc nó thực hiện lệnh quay trở về RET1 mà chân INT1 vẫn còn ở mức thấp thì bộ vi điều khiển khởi tạo lại ngắt. Do vậy, để giải quyết vấn đề này thì chân INT1 phải được đưa lên cao tại thời điểm lệnh RET1 được thực hiện.

11.3.3 Trích mẫu ngắt theo mức.

Các chân P3.2 và P3.3 bình thường được dùng cho vào - ra nếu các bít INT0 và INT1 trong thanh ghi IE không được kích hoạt. Sau khi các ngắt phần cứng trong thanh gi IE được kích hoạt thì bộ vi điều khiển duy trì trích mẫu trên chân INTn đối với tín hiệu mức thấp một lần trong một chu trình máy. Theo bảng dữ liệu của nhà sản xuất của bộ vi điều khiển thì "chân ngắt phải được giữ ở mức thấp cho đến khi bắt đầu thực hiện trình phục vụ ngắt ISR. Nếu chân INTn được đưa trở lại mức cao trước khi bắt đầu thực hiện ISR thì sẽ chẳng có ngắt nào xảy ra". Tuy nhiên trong quá trình kích hoạt ngắt theo mức thấp nên nó lại phải đưa lên mức cao trước khi thực hiện lệnh RET1 và lại theo bảng dữ liệu của nhà sản xuất thì "nếu chân INTn vẫn ở mức thấp sau lệnh RETI của trình phục vụ ngắt thì một ngắt khác lại sẽ được kích hoạt sau khi lệnh RET1 được thực hiện". Do vậy, để bảo đảm việc kích hoạt ngắt phần cứng tại các chân INTn phải khẳng định rằng thời gian tồn tại tín hiệu mức thấp là khoảng 4 chu trình máy và không được hơn. Điều này là do một thực tế là ngắt theo mức không được chốt. Do vậy chân ngắt phải được giưa ở mức thấp cho đến khi bắt đầu thực hiện ISR.



Ghi chú: Khi bật lại nguồn (RESET) thì cả hai chân INT0 và INT1 đều ở mức thấp tạo các ngắt ngoài theo mức.

Hình 11.5: Thời gian tối thiểu của ngắt theo mức thấp (XTAL = 11.0592MHz)

11.3.4 Các ngắt theo sườn.

Như đã nói ở trước đây trong quá trình bật lại nguồn thì 8051 làm các chân INT0 và INT1 là các ngắt theo mức thấp. Để biến các chân này trở thành các ngắt theo sườn thì chúng ta phải viết chương trình cho các bít của thanh ghi TCON. Thanh thi TCON giữ các bít cờ IT0 và IT1 xác định chế độ ngắt theo sườn hay ngắt theo mức của các ngắt phần cứng IT0 và IT1 là các bít D0 và D2 của thanh ghi TCON tương ứng. Chúng có thể được biểu diễn như TCON.0 và TCON.2 vì thanh ghi TCON có thể đánh địa chỉ theo bít. Khi bật lại nguồn thì TCON.0 (IT0) và TCON.2 (IT1) đều ở mức thấp (0) nghĩa là các ngắt phần cứng ngoài của các chân INT0 và INT1 là ngắt theo mức thấp. Bằng việc chuyển các bít TCON.0 và TCON.2 lên cao qua các lệnh "SETB TCON.0" và "SETB TCON.2" thì các ngắt phần cứng ngoài INT0 và INT1 trở thành các ngắt theo sườn. Ví dụ, lệnh "SETB TCON.2" làm cho INT1 mà được gọi là ngắt theo sườn trong đó khi một tín hiệu chuyển từ cao xuống thấp được cấp đến chân P3.3 thì ở trường hợp này bộ vi điều khiển sẽ bị ngắt và bị cưỡng bức nhảy đến bảng véc tơ ngắt tại địa chỉ 0013H để thực hiện trình phục vụ ngắt. Tuy nhiên là với giải thiết rằng bít ngắt đã được cho phép trong thanh ghi IE.

D7							D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Hình 11.6: Thanh ghi TCON.

- Bít TF1 hay TCON.7 là cờ tràn của bộ Timer1. Nó được lập bởi phần cứng khi bộ đếm/ bộ định thời 1 tràn, nó được xoá bởi phần cứng khi bộ xử lý chỉ đến trình phục vụ ngắt.
- Bít TR1 hay TCON.6 là bít điều khiển hoạt động của Timer1. Nó được thiết lập và xoá bởi phần mềm để bật/ tắt Timer1.
- Bít TF0 hay TCON.5 tương tự như TF1 dành cho Timer0.
- Bít TR0 hay TCON.4 tương tự như TR1 dành cho Timer0.
- Bít IE1 hay TCON.3 cờ ngắt ngoài 1 theo sườn. Nó được thiết lập bởi CPU khi sườn ngắt ngoài (chuyển từ cao xuống thấp) được phát hiện. Nó được xóa bởi CPU khi ngắt được xử lý. Lưu ý: Cờ này không chốt những ngắt theo mức thấp.
- Bít IT1 hay TCON.2 là bít điều khiển kiểu ngắt. Nó được thiết lập và xoá bởi phần mềm để xác định kiểu ngắt ngoài theo sườn xuống hay mức thấp.
- Bít IE0 hay TCON.1 tương tự như IE1 dành cho ngắt ngoài 0.
- Bít IT0 hay TCON.0 tương tự như bít IT1 dành cho ngắt ngoài 0.

Xét ví dụ 11.6, chú ý rằng sự khác nhau duy nhất giữa vì dụ này và ví dụ 11.5 là ở trong hàng đầu tiên của MAIN khi lệnh "SETB TCON.2" chuyển ngắt INT1 về kiểu ngắt theo sườn. Khi sườn xuống của tín hiệu được cấp đến chân INT1 thì đèn LED sẽ bật lên một lúc. Đèn LED có thời gian sáng phụ thuộc vào độ trễ bên trong ISR của INT1. Để bật lại đèn LED thì phải có một sườn xung xuống khác được cấp đến chân P3.3. Điều này ngược với ví dụ 11.5. Trong ví dụ 11.5 do bản chất ngắt theo mức của ngắt thì đèn LED còn sáng chừng nào tín hiệu ở chân INT1 vẫn còn ở mức thấp. Nhưng trong ví dụ này để bật lại đèn LED thì xung ở chân INT1 phải được đưa lên cao rồi sau đó bị hạ xuống thấp để tạo ra một sườn xuống làm kích hoạt ngắt.

Ví dụ 11.6:

Giả thiết chân P3.3 (INT1) được nối với một máy tạo xung, hãy viết một chương trình trong đó sườn xuống của xung sẽ gửi một tín hiệu cao đến chân P1.3 đang được nối tới đèn LED (hoặc một còi báo). Hay nói cách khác, đèn LED được bật và tắt cùng tần số với các xung được cấp tới chân INT1. Đây là phiên bản ngắt theo sườn xung của ví dụ 11.5 đã trình bày ở trên.

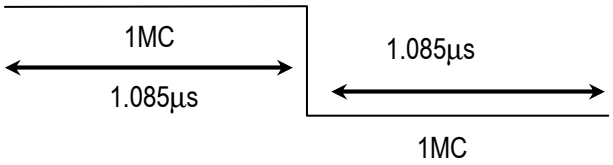
Lời giải:

```
                ORG     0000H
                LJMP    MAIN
; - - Trình  phục vụ ngắt ISR dành cho ngắt INT1 để bật đèn LED
                ORG     0013H           ; Nhảy đến địa chỉ của trình phục vụ ngắt INT1
                SETB    P1.3            ; Bật đèn LED (hoặc còi)
                MOV     R3, #225
BACK:           DJNZ    R3, HERE        ; giữ đèn LED (hoặc còi) một lúc
                CLR     P1.3            ; Tắt đèn LED (hoặc còi)
                RETI                    ; Quay trở về từ ngắt
; - - Bắt đầu chương trình chính
                ORG     30H
                SETB    TCON.2          ; Chuyển ngắt INT1 về kiểu ngắt theo sườn xung
```

```
        MOV   IE, #10001B     ; Cho phép ngắt ngoài INT1
HERE:   SJMP  HERE            ; Dừng ở đây cho đến khi bị ngắt
        END
```

11.3.5 Trình mẫu ngắt theo sườn.

Trước khi kết thúc phần này ta cần trả lời câu hỏi vậy thì ngắt theo sườn được trích mẫu thường xuyên như thế nào? Trong các ngắt theo sườn, nguồn ngoài phải giữ ở mức cao tối thiểu là một chu trình máy nữa đê đảm bảo bộ vi điều khiển nhìn thấy được sự chuyển dịch từ cao xuống thấp của sườn xung.



Thời hạn xung tối thiểu để phát hiện ra các ngắt theo sườn xung với tần số XTAL = 11.0592MHz

Sườn xuống của xung được chốt bởi 8051 và được giữa bởi thanh ghi TCON. Các bít TCON.1 và TCON.3 giữ các sườn được chốt của chân INT0 và INT1 tương ứng. TCON.1 và TCON.3 cũng còn được gọi là các bít IE0 và IE1 như chỉ ra trên hình 11.6. Chúng hoạt động như các cờ "ngắt đang được phục vụ" (Interrupt-in-server). Khi một cờ "ngắt đang được phục vụ" bật lên thì nó báo cho thế giới thực bên ngoài rằng ngắt hiện nay đang được xử lý và trên chân INTn này sẽ không có ngắt nào được đáp ứng chừng nào ngắt này chưa được phục vụ xong. Đây giống như tín hiệu báo bận ở máy điện thoại. Cần phải nhấn mạnh hạt điểm dưới đây khi quan tâm đến các bít IT0 và IT1 của thanh ghi TCON.

1. Khi các trình phục vụ ngắt ISR kết thúc (nghĩa là trong thanh ghi thực hiện lệnh RETI). Các bít này (TCON.1 và TCON.3) được xoá để báo rằng ngắt được hoàn tất và 8051 sẵn sàng đáp ứng ngắt khác trên chân đó. Để ngắt khác được nhận và thì tín hiệu trên chân đó phải trở lại mức cao và sau đó nhảy xuống thấp để được phát hiện như một ngắt theo sườn.
2. Trong thời gian trình phục vụ ngắt đang được thực hiện thì chân INTn bị làm ngơ không quan tâm đến nó có bao nhiêu lần chuyển dịch từ cao xuống thấp. Trong thực tế nó là một trong các chức năng của lệnh RETI để xoá bít tương ứng trong thanh ghi TCON (bít TCON.1 và TCON.3). Nó báo cho ta rằng trình phục vụ ngắt xắp kết thúc. Vì lý do này mà các bít TCON.1 và TCON.3 được gọi là các cơ báo "ngắt đang được phục vụ" cờ này sẽ lên cao khi một sườn xuống được phát hiện trên chân INT và dừng ở mức cao trong toàn bộ quá trình thực hiện ISR. Nó chỉ bị xoá bởi lệnh RETI là lệnh cuối cùng của ISR. Do vậy, sẽ không báo giờ cần đến các lệnh xoá bít này như "CLR TCON.1" hay "CLR TCON.3" trước lệnh RETI trong trình phục vụ ngắt đối với các ngắt cứng INT0 và INT1. Điều này không đúng với trường hợp của ngắt nối tiếp.

Ví dụ 11.7:

Sự khác nhau giữa các lệnh RET và RETI là gì? Giải thích tại sao ta không thể dùng lệnh RET thay cho lệnh RETI trong trình phục vụ ngắt.

Lời giải:

Các hai lệnh RET và RETI đều thực thi các hành vi giống nhau là lấy hai byte trên đỉnh ngăn xếp vào bộ đếm chương trình và đưa 8051 trở về nơi đó đã bỏ đi. Tuy nhiên, lệnh RETI còn thực thi một nhiệm vụ khác nữa là xoá cờ "ngắt đang được phục vụ" để báo rằng ngắt đã kết thúc và 8051 có thể nhập một ngắt mới trên chân này. Nếu ta dùng lệnh RET thay cho RETI như là lệnh cuối cùng của trình phục vụ ngắt như vậy là ta đã vô tình khoá mọi ngắt mới trên chân này sau ngắt đầu tiên vì trạng thái của chân báo rằng ngắt vẫn đang được phục vụ. Đây là trường hợp mà các cờ TF0, TF1, TCON.1 và TCON.3 được xoá bởi lệnh RETI.

11.3.6 Vài điều bổ xung về thanh ghi TCON.

Bây giờ ta xét kỹ về các bít của thanh ghi TCON để hiểu vai trò của nó trong việc duy trì các ngắt.

11.3.6.1 Các bít IT0 và IT1.

Các bít TCON.0 và TCON.2 được coi như là các bít IT0 và IT1 tương ứng. Đây là các bít xác định kiểu ngắt theo sườn xung hay theo mức xung của các ngắt phần cứng trên chân INT.0 và INT.1 tương ứng. Khi bật lại nguồn cả hai bít này đều có mức 0 để biến chúng thành ngắt theo tín hiệu mức thấp. Lập trình viên có thể điều khiển một trong số chúng lên cao để chuyển ngắt phần cứng bên ngoài thành ngắt theo ngưỡng. Trong một hệ thống dựa trên 8051 đã cho thì một khi ta đã đặt về 0 hoặc 1 thì các bít này sẽ không thay đổi vì người thiết kế đã cố định kiểu ngắt là ngắt theo sườn hay theo mức rồi.

11.3.6.2 Các bít IE0 và IE1.

Các bít TCON.1 và TCON.3 còn được gọi là IE0 và IE1 tương ứng. Các bít này được 8051 dùng để bám kiểu ngắt theo sườn xung. Nói các khác là nếu IT0 và IT1 bằng 0 thì có nghĩa là các ngắt phần cứng là ngắt theo mức thấp, các bít IE0 và IE1 không dùng đến làm gì. Các bít IE0 và IE1 được 8051 chỉ dùng để chốt sườn xung từ cao xuống thấp trên các chân INT0 và INT1. Khi có chuyển dịch sườn xung trên chân INT0 (hay INT1) thì 8051 đánh dấu (bật lên cao) các bít IEx trên thanh ghi TCON nhảy đến bảng véc tơ ngắt và bắt đầu thực hiện trình phục vụ ngắt ISR. Trong khi 8051 thực hiện ISR thì không có một sườn xung nào được ghi nhận trên chân INT0 (hay INT1) để ngăn mọi ngắt trong ngắt. Chỉ trong khi thực hiện lệnh RETI ở cuối trình phục vụ ngắt ISR thì các bít IEx mới bị báo rằng một sườn xung cao xuống thấp mới trên chân INT0 (hay INT1) sẽ kích hoạt ngắt trở lại. Từ phần trình bày trên ta thấy rằng các bít IE0 và IE1 được 8051 sử dụng bên trong để báo có một ngắt đang được xử lý hay không. Hay nói cách khác là lập trình viên không phải quan tâm đến cá bít này.

11.3.6.3 Các bít TR0 và TR1.

Đây là những bít D4 và D6 (hay TCON.4 và TCON.6) của thanh ghi TCON. Các bít này đã được giới thiệu ở chương 9 chúng được dùng để khởi động và dừng các bộ định thời Timer0 và Timer1 tương ứng. Vì thanh ghi TCON có thể đánh địa chỉ theo bít nên có thể sử dụng các lệnh "SETB TRx" và "CLR TRx" cũng như các lệnh "SETB TCON.4" và "CLR TCON.4".

11.3.6.4 Các bít TF0 và TF1.

Các bít này là D5 (TCON.5) và D7 (TCON.7) của thanh ghi TCON mà đã được giới thiệu ở chương 9. Chúng ta được sử dụng bởi các bộ Timer0 và Timer1 tương ứng để báo rằng các bộ định thời bị tràn hay quay về không. Mặc dù ta đã dùng các lệnh "JNB TFx, đích" và "CLR TFx" nhưng chúng ta cũng không thể sử

dụng các lệnh như “SETB TCON.5, đích” và “CLR TCON.5” vì TCON là thanh ghi có thể đánh địa chỉ theo bít.

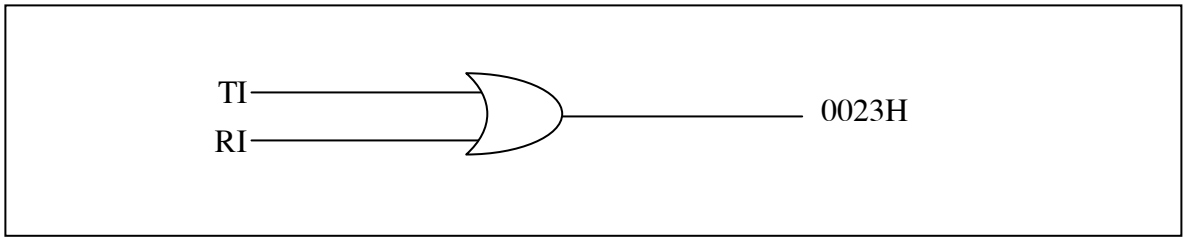
11.4 Lập trình ngắt truyền thông nối tiếp.

Trong chương 10 chúng ta đã nghiên cứu về truyền thông nối tiếp của 8051. Tất cả các ví dụ trong chương ấy đều sử dụng phương pháp thăm dò (polling). ậ chương này ta khám phá truyền thông dựa trên ngắt mà nó cho phép 8051 làm việc rất nhiều việc ngoài việc truyền và nhận dữ liệu từ cổng truyền thông nối tiếp.

11.4.1 Các cờ RI và TI và các ngắt.

Như đã nói ở chương 10 thì cờ ngắt truyền TI (Transfer interrupt) được bật lên khi bít cuối cùng của khung dữ liệu, bít stop được truyền đi báo rằng thanh ghi SBUF sẵn sàng truyền byte kế tiếp. Trong trường hợp cờ RI (Receive Interrupt) thì nó được bật lên khi toàn bộ khung dữ liệu kể cả bít stop đã đươc nhận. Hay nói cách khác khi thanh ghi SBUF đã có một byte thì cờ RI bật lên báo rằng byte dữ liệu nhận được cần lấy đi cất vào nơi an toàn trước khi nó bị mất (bị ghi đè) bởi dữ liệu mới nhận được. Chừng nào còn nói về truyền thông nối tiếp thì tất cả mọi khái niệm trên đây đều áp dụng giống như nhau cho dù sử dụng phương pháp thăm dò hay sử dụng phương pháp ngắt. Sự khác nhau duy nhất giữa hai phương pháp này là ở cách phục vụ quá trình truyền thông nối tiếp như thế nào. Trong phương pháp thăm dò thì chúng ta phải đợi cho cờ (TI hay RI) bật lên và trong lúc chờ đợi thì ta không thể làm gì được cả. Còn trong phương pháp ngắt thì ta được báo khi 8051 đã nhận được một byte hoặc nó sẵn sàng chuyển (truyền) byte kế tiếp và ta có thể làm các công việc khác trong khi truyền thông nối tiếp đang được phục vụ.

Trong 8051 chỉ có một ngắt dành riêng cho truyền thông nối tiếp. Ngắt này được dùng cho cả truyền và nhận dữ liệu. Nếu bít ngắt trong thanh gi IE (là bít IE.4) được phép khi RI và TI bật lên thì 8051 nhận được ngắt và nhảy đến địa chỉ trình phục vụ ngắt dành cho truyền thông nối tiếp 0023H trong bảng véc tơ ngắt để thực hiện nó. Trong trình ISR này chúng ta phải kiểm tra các cờ TI và RI để xem cờ nào gây ra ngắt để đáp ứng một cách phù hợp (xem ví dụ 11.8).



Hình 11.7: Ngắt truyền thông có thể do hai cờ TI và RI gọi.

11.4.2 Sử dụng cổng COM nối tiếp trong 8051.

Trong phần lớn các ứng dụng, ngắt nối tiếp chủ yếu được sử dụng để nhận dữ liệu và không bao giờ được sử dụng để truyền dữ liệu nối tiếp. Điều này giống như việc báo chuông để nhận điện thoại, còn nếu ta muốn gọi điện thoại thì có nhiều cách khác ngắt ta chứ không cần đến đổ chuông. Tuy nhiên, trong khi nhận điện thoại ta phải trả lời ngay không biết ta đang làm gì nếu không thuộc gọi sẽ (mất) đi qua. Tương tự như vậy, ta sử dụng các ngắt nối tiếp khi nhận dữ liệu đi đến để sao chép cho nó không bị mất: Hãy xét ví dụ 11.9 dưới đây.

Ví dụ 11.8:

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 9600.

Lời giải:

```
                ORG     0
                LJMP    MAIN
                ORG     23H
                LJMP    SERIAL                  ; Nhảy đến trình phục vụ ngắt truyền thông nối tiếp
MAIN:           MOVQP1, # 0FFH                  ; Lấy cổng P1 làm cổng đầu vào
                MOV     TMOD, # 20h             ; Chọn Timer1, chế độ 2 tự nạp lại
                MOV     TH1, # 0FDH             ; Chọn tốc độ baud = 9600
                MOV     SCON, # 50H             ; Khung dữ liệu: 8 bít dữ liệu, 1 stop à cho phép REN
                MOV     IE, # 10010000B         ; Cho phép ngắt nối tiếp
                SETB    TR1                     ; Khởi động Timer1
BACK:           MOV     A, P1                   ; Đọc dữ liệu từ cổng P1
                MOV     SBUF, A                 ; Lấy một bản sao tới SBUF
                MOV     P2, A                   ; Gửi nó đến cổng P2
                SJMP    BACK                    ; ở lại trong vòng lặp
;
; -------------------------Trình phục vụ ngắt cổng nối tiếp
                ORG     100H
SERIAL:         JB      TI,TRANS                ; Nhảy đến cờ TI cao
                MOV     A, SBUF                 ; Nếu không tiếp tục nhận dữ liệu
                CLR     RI                      ; Xoá cờ RI vì CPU không làm điều này
                RETI                            ; Trở về từ trình phục vụ ngắt
TRANS:          CLR     TI                      ; Xoá cờ TI vì CPU không làm điều này
                RETI                            ; Trở về từ ISR
                END
```

Trong vấn đề trên thấy chú ý đến vai trò của cờ TI và RI. Thời điểm một byte được ghi vào SBUF thì nó được đóng khung và truyền đi nối tiếp. Kết quả là khi bít cuối cùng (bít stop) được truyền đi thì cờ TI bật lên cao và nó gây ra ngắt nối tiếp được gọi khi bít tương ứng của nó trong thanh ghi IE được đưa lên cao. Trong trình phục vụ ngắt nối tiếp, ta phải kiểm tra cả cờ TI và cờ RI vì cả hai đều có thể gọi ngắt hay nói cách khác là chỉ có một ngắt cho cả truyền và nhận.

Ví dụ 11.9:

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. Giả thiết tần số XTAL là 11.0592MHz và tốc độ baud 9600.

Lời giải:

```
                ORG     0
                LJMP    MAIN
                ORG     23H
                LJMP            SERIAL          ; Lấy cổng P1 là cổng đầu vào
                ORG     03H
MAIN:           MOV     P1, # FFH
                MOV     TMOD, # 20H             ; Chọn Timer và chế độ hai tự nạp lại
                MOV     TH1, # 0FDH             ;Khung dữ liệu: 8 bít dữ liệu, 1 stop, cho phép REN
```

```
                MOV     SCON, # 50H             ; Cho phép ngắt nối tiếp
                MOV     IE, # 10010000B         ; Khởi động Timer1
                SETB    TR1                     ; Đọc dữ liệu từ cổng P1
BACK:           MOV     A, P1                   ; Gửi dữ liệu đến cổng P2
                MOV     P2, A                   ; ở lại trong vòng lặp
                SJMP    BACK
; -------------------------Trình phục vụ ngắt cổng nối tiếp.
                ORG     100H
SERIAL:         JB      TI, TRANS               ; Nhảy nếu Ti cao
                MOV     A, SBUF                 ; Nếu không tiếp tục nhận dữ liệu
                MOV     P0, A                   ; Gửi dữ lệu đầu vào đến cổng P0
                CLR     RI                      ; Xoá  vờ RI vì CPU không xoá cờ này
                RETI                            ; Trở về từ ISR
TRANS:          CLS     TI                      ; Xoá cờ TI và CUP không xoá cờ này.
                RETI                            ; ; trở về từ ISR
                END
```

11.4.3 Xoá cờ RI và TI trước lệnh RETI.

Để ý rằng lệnh cuối cùng trước khi trở về từ ISR là RETI là lệnh xoá các cờ RI và TI. Đây là điều cần thiết bởi vì đó là ngắt duy nhất dành cho nhận và truyền 8051 không biết được nguồn gây ra ngắt là nguồn nào, do vậy trình phục vụ ngắt phải được xoá các cờ này để cho phép các ngắt sau đó được đáp ứng sau khi kết thúc ngắt. Điều này tương phản với ngắt ngoài và ngắt bộ định thời đều được 8051 xoá các cờ. Các lệnh xoá các cờ ngắt bằng phần mềm qua các lệnh "CLR TI" và "CLR RI". Hãy xét ví dụ 11.10 dưới đây và để ý đến các lệnh xoá cờ ngắt trước lệnh RETI.

Ví dụ 11.10:

Hãy viết một chương trình sử dụng các ngắt để thực hiện các công việc sau:

a) Nhận dữ liệu nối tiếp và gửi nó đến cổng P0.
b) Lấy cổng P1 đọc và truyền nối tiếp và sao 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.

Lời giải:

```
                ORG     0
                LJMP    MAIN
                ORG     000BH                   ; Trình phục vụ ngắt dành cho Timer0
                CPL     P0.1                    : Tạo xung ở chân P0.1
                RETI                            ; Trở về từ ISR
                ORG     23H                     ; Nhảy đến địa chỉ ngắt truyền nối tiếp
                LJMP    SERIAL                  ; Lấy cổng P1 làm cổng đầu vào
                ORG     30H
MAIN :          MOV     P1, # 0FFH              ; Chọn Timer0 và Timer1 chế độ 2 tự nạp lại
                MOV     TMOD, # 22H             ; Chọn Timer0 và Timer1 chế độ 2 tự nạp lại
                MOV     TH1, # 0F6H             ; Chọn tốc độ baud 4800
                MOV     SCON, # 50H             ; Khung dữ liệu: 8 bít dữ liệu, 1 stop, cho phép REN
                MOV     TH0, # - 92             ; Tạo tần số 5kHz
                MOV     IE, # 10010010B         ; Cho phép ngắt nối tiếp
                SETB    TR1                     ; Khởi động Timer1
                SETB    TR0                     ; Khởi động Timer0
BACK:           MOV     A, P1                   ; Đọc dữ liệu từ cổng P1
                MOV     SBUF, A                 ; Lấy một lần bản sao dữ liệu
                MOV     P2, A                   ; Ghi nó vào cổng P2
```

```
                SJMP    BACK                ; ở lại trong vòng lặp
; ---------------------- Trình phục vụ ngắt cổng nối tiếp.
                ORG     100H
SERIAL:         JB      TI, TRANS           ; Nhảy nếu TI vào
                MOV     A, SBUF             ; Nếu không tiếp tục nhận dữ liệu
                MOV     P0, A               ; Gửi dữ liệu nối tiếp đến P0
                CLR     RI                  ; Xoá cờ RI vì 8051 không làm điều này
                RETI                        ; Trở về từ ISR
TRANS:          CLR     TI                  ; Xoá cờ TI vì 8051 không xoá
                RETI                        ; Trở về từ ISR.
                END
```

Trước khi kết thúc phần này hãy để ý đến danh sách tất cả mọi cờ ngắt được cho trong bảng 11.2. Trong khi thanh thi TCON giữ 4 cờ ngắt còn hai cờ TI và RI ở trong thanh ghi SCON của 8051.

Bảng 11.2: Các bít cờ ngắt.

Ngắt	**Cờ**	**Bít của thanh ghi SFR**
Ngắt ngoài 0	IE0	TCON.1
Ngắt ngoài 1	IE1	TCON.3
Ngắt Timer0	TF0	TCON.5
Ngắt Timer1	TF1	TCON.7
Ngắt cổng nối tiếp	T1	SCON.1
Ngắt Timer2	TF2	T2CON.7 (TA89C52)
Ngắt Timer2	EXF2	T2CON.6 (TA89C52)

11.5 Các mức ưu tiên ngắt trong 8051.

11.5.1 Các mức ưu tiên trong quá trình bật lại nguồn.

Khi 8051 được cấp nguồn thì các mức ưu tiên ngắt được gán theo bảng 11.3. Từ bảng này ta thấy ví dụ nếu các ngắt phần cứng ngoài 0 và 1 được kích hoạt cùng một lúc thì ngắt ngoài 0 sẽ được đáp ứng trước. Chỉ sau khi ngắt INT0 đã được phục vụ xong thì INT1 mới được phục vụ vì INT1 có mức ưu tiên thấp hơn. Trong thực tế sơ đồ mức ưu tiên ngắt trong bảng không có ý nghĩa gì cả mà một quy trình thăm dò trong đó 8051 thăm dò các ngắt theo trình tự cho trong bảng 11.3 và đáp ứng chúng một cách phù hợp.

Bảng 11.3: Mức ưu tiên các ngắt trong khi cấp lại nguồn.

Mức ưu tiên cao xuống thấp	
Ngắt ngoài 0	INT0
Ngắt bộ định thời 0	TF0
Ngắt ngoài 1	INT1
Ngắt bộ định thời 1	TF1
Ngắt truyền thông nối tiếp	(RI + TI)

Ví dụ 11.1:

Hãy bình luận xem điều gì xảy ra nếu các ngắt INT0, TF0 và INT1 được kích hoạt cùng một lúc. Giả thiết rằng các mức ưu tiên được thiết lập như khi bật lại nguồn và các ngắt ngoài là ngắt theo sườn xung.

Lời giải:

Nếu ba ngắt này được kích hoạt cùng một thời điểm thì chúng được chốt và được giữ ở bên trong. Sau đó kiểm tra tất cả năm ngắt theo trình tự cho trong bảng 11.3. Nếu một ngắt bất kỳ được kích hoạt thì nó được phục vụ theo trình tự. Do vậy, khi cả ba ngắt trên đây cùng được kích hoạt một lúc thì ngắt ngoài 0 (IE0) được phục vụ trước hết sau đó đến ngắt Timer0 (TF0) và cuối cùng là ngắt ngoài 1 (IE1).

D7 D0

--	--	PT2	PS	PT1	PX1	PT0	PX0

Hình 11.8: Thanh ghi mức ưu tiên ngắt IP, bít ưu tiên = 1 là mức ưu tiên cao, bít ưu tiên = 0 là mức ưu tiên thấp.

- Bít D7 và D6 hay IP.7 và IP.6 - chưa dùng.
- Bít D5 hay IP.5 là bít ưu tiên ngắt Timer2 (dùng cho 8052)
- Bít D4 hay IP.4 là bít ưu tiên ngắt cổng nối tiếp
- Bít D3 hay IP.3 là bít ưu tiên ngắt Timer1
- Bít D2 hay IP.2 là mức ưu tiên ngắt ngoài 1
- Bít D1 hay IP.1 là mức ưu tiên ngắt Timer 0
- Bít D0 hay IP.0 là mức ưu tiên ngắt ngoài 0

Người dùng không được viết phần mềm ghi các số 1 vào các bít chưa dùng vì chúng dành cho các ứng dụng tương lai.

11.5.2 Thiết lập mức ưu tiên ngắt với thanh ghi IP.

Chúng ta có thể thay đổi trình tự trong bảng 11.3 bằng cách gán mức ưu tiên cao hơn cho bất kỳ ngắt nào. Điều này được thực hiện bằng cách lập trình một thanh ghi gọi là thanh ghi mức ưu tiên ngắt IP (Interrupt Priority). Trên hình 11.8 là các bít của thanh ghi này, khi bật lại nguồn thanh thi 1P chứa hoàn toàn các số 0 để tạo ra trình tự ưu tiên ngắt theo bảng 11.3. Để một ngắt nào đó mức ưu tiên cao hơn ta thực hiện đưa bít tương ứng lên cao. Hãy xem ví dụ 11.12.

Một điểm khác nữa cần được làm sáng tỏ là mức ưu tiên ngắt khi hai hoặc nhiều bít ngắt trong thanh ghi IP được đặt lên cao. Trong trường hợp này thì trong khi các ngắt này có mức ưu tiên cao hơn các ngắt khác chúng sẽ được phục vụ theo trình tự cho trong bảng 11.3. Xem ví dụ 11.13.

Ví dụ 11.12:

a) Hãy lập trình thanh ghi IP để gán mức ưu tiên cao nhất cho ngắt INT1 (ngắt ngoài 1) sau đó.
b) Hãy phân tích điều gì xảy ra khi INT0, INT1 và TF0 được kích hoạt cùng lúc. Giả thiết tất cả các ngắt đều là các ngắt theo sườn.

Lời giải:

a) MOV IP, #0000 0100B ; Đặt bít IP.2 = 1 để gán INT1 mức ưu tiên cao nhất. Lệnh "SETB IP.2" cũng tác động tương tự bởi vì IP là thanh ghi có thể đánh địa chỉ theo bít.
b) Lệnh trong bước a) gán mức ưu tiên cao hơn INT1 so với các ngắt khác, do vậy khi INT0, INT1 và TF0 được kích hoạt cùng lúc thì trước hết INT1 được phục vụ

trước rồi sau đó đến INT0 và cuối cùng là TF0. Điều này là do INT1 có mức ưu tiên cao hơn hai ngắt kia ở bước a). Sau khi thực hiện song ngắt INT1 thì 8051 trở về phục vụ ngắt còn lại theo trình tự ưu tiên trong bảng 11.3.

Ví dụ 11.13:

Giả thiết rằng sau khi bật lại nguồn thì mức ưu tiên ngắt được thiết lập bởi lệnh "MOV IP, #0000 1100B". Hãy bình luận về quá trình các ngắt được phục vụ như thế nào?

Lời giải:

Lệnh "MOV IP, #0000 1100B" (chữ B là giá trị thập phân) thiết lập ngắt ngoài (INT1) và ngắt bộ Timer1 (TF1) có mức ưu tiên cao hơn các ngắt khác. Tuy nhiên, vì chúng được thăm dò theo bảng 11.3 nên chúng sẽ được phục vụ theo trình tự sau:

Mức ưu tiên cao nhất: Ngắt ngoài 1 (INT1)
Ngắt bộ Timer 1 (TF1)
Ngắt ngoài 0 (INT0)
Ngắt bộ Timer0 (TF0)
Mức ưu tiên thấp nhất: Ngắt cổng truyền thông nối tiếp (RI + RT).

11.5.3 Ngắt trong ngắt.

Điều gì xảy ra nếu 8051 đang thực hiện một trình phục vụ ngắt thuộc một ngắt nào đó thì lại có một ngắt khác được kích hoạt? Trong những trường hợp như vậy thì một ngắt có mức ưu tiên cao hơn có thể ngắt một ngắt có mức ưu tiên thấp hơn. Đây gọi là ngắt trong ngắt. Trong 8051 một ngắt ưu tiên thấp có thể bị ngắt bởi một ngắt có mức ưu tiên cao hơn chứ không bị ngắt bởi một ngắt có mức ưu tiên thấp hơn. Mặc dù tất cả mọi ngắt đều được chốt và gửi bên trong nhưng không có ngắt mức thấp nào được CPU quan tâm ngay tức khắc nếu 8051 chưa kết thúc phục vụ các ngắt mức cao.

11.5.4 Thu chộp ngắt bằng phần mềm (Triggering).

Có nhiều lúc ta cần kiểm tra một trình phục vụ ngắt bằng con đường mô phỏng. Điều này có thể được thực hiện bằng các lệnh đơn giản để thiết lập các ngắt lên cao và bằng cách đó buộc 8051 nhảy đến bảng véc tơ ngắt. Ví dụ, nếu bít IE dành cho bộ Timer1 được bật lên 1 thì một lệnh như "SETB TF1" sẽ ngắt 8051 ngừng thực hiện công việc đang làm bất kỳ và buộc nó nhảy đến bảng véc tơ ngắt. Hay nói cách khác, ta không cần đợi cho Timer1 quay trở về 0 mới tạo ra ngắt. Chúng ta có thể gây ra một ngắt bằng các lệnh đưa các bít của ngắt tương ứng lên cao.

Như vậy ở chương này chúng ta đã biết ngắt là một sự kiện bên trong hoặc bên ngoài gây ra ngắt bộ vi điều khiển để báo cho nó biết rằng thiết bị cần được phục vụ. Mỗi một ngắt có một chương trình đi kèm với nó được gọi là trình phục vụ ngắt ISR. Bộ vi điều khiển 8051 có sáu ngắt, trong đó năm ngắt người dùng có thể truy cập được. Đó là hai ngắt cho các thiết bị phần cứng bên ngoài INT0 và INT1, hai ngắt cho các bộ định thời là TF0 và TF1 và ngắt lành cho truyền thông nối tiếp.

8051 có thể được lập trình cho phép hoặc cấm một ngắt bất kỳ cũng như thiết lập mức ưu tiên cho nó theo yêu cầu của thuật toán ứng dụng.

CHƯƠNG 12
Phối ghép với thế giới thực: LCD, ADC và các cảm biến

Chương này khám phá một số ứng dụng của 8051 với thế giới thực. Chúng ta giải thích làm cách nào phối ghép 8051 với các thiết bị như là LCD, ADC và các cảm biến.

12.1 Phối ghép một LCD với 8051.

Ở phần này ta sẽ mô tả các chế độ hoạt động của các LCD và sau đó mô tả cách lập trình và phối ghép một LCD tới 8051.

12.1.1 Hoạt động của LCD.

Trong những năm gần đây LCD đang ngày càng được sử dụng rộng rãi thay thế dần cho các đèn LED (các đèn LED 7 đoạn hay nhiều đoạn). Đó là vì các nguyên nhân sau:

1. Các LCD có giá thành hạ.
2. Khả năng hiển thị các số, các ký tự và đồ hoạ tốt hơn nhiều so với các đèn LED (vì các đèn LED chỉ hiển thị được các số và một số ký tự).
3. Nhờ kết hợp một bộ điều khiển làm tươi vào LCD làm giải phóng cho CPU công việc làm tươi LCD. Trong khi đèn LED phải được làm tươi bằng CPU (hoặc bằng cách nào đó) để duy trì việc hiển thị dữ liệu.
4. Dễ dàng lập trình cho các ký tự và đồ hoạ.

12.1.2 Mô tả các chân của LCD.

LCD được nói trong mục này có 14 chân, chức năng của các chân được cho trong bảng 12.1. Vị trí của các chân được mô tả trên hình 12.1 cho nhiều LCD khác nhau.

1. Chân V_{CC}, V_{SS} và V_{EE}: Các chân V_{CC}, V_{SS} và V_{EE}: Cấp dương nguồn - 5v và đất tương ứng thì V_{EE} được dùng để điều khiển độ tương phản của LCD.
2. Chân chọn thanh ghi RS (Register Select).

Có hai thanh ghi rất quan trọng bên trong LCD, chân RS được dùng để chọn các thanh ghi này như sau: Nếu RS = 0 thì thanh ghi mà lệnh được chọn để cho phép người dùng gửi một lệnh chẳng hạn như xoá màn hình, đưa con trỏ về đầu dòng v.v… Nếu RS = 1 thì thanh ghi dữ liệu được chọn cho phép người dùng gửi dữ liệu cần hiển thị trên LCD.

3. Chân đọc/ ghi (R/W).

Đầu vào đọc/ ghi cho phép người dùng ghi thông tin lên LCD khi R/W = 0 hoặc đọc thông tin từ nó khi R/W = 1.

4. Chân cho phép E (Enable).

Chân cho phép E được sử dụng bởi LCD để chốt thông tin hiện hữu trên chân dữ liệu của nó. Khi dữ liệu được cấp đến chân dữ liệu thì một xung mức cao xuống thấp phải được áp đến chân này để LCD chốt dữ liệu trên các chân dữ liêu. Xung này phải rộng tối thiểu là 450ns.

5. Chân D0 - D7.

Đây là 8 chân dữ liệu 8 bít, được dùng để gửi thông tin lên LCD hoặc đọc nội dung của các thanh ghi trong LCD.

Để hiển thị các chữ cái và các con số, chúng ta gửi các mã ASCII của các chữ cái từ A đến Z, a đến f và các con số từ 0 - 9 đến các chân này khi bật RS = 1.

Cũng có các mã lệnh mà có thể được gửi đến LCD để xoá màn hình hoặc đưa con trỏ về đầu dòng hoặc nhấp nháy con trỏ. Bảng 12.2 liệt kê các mã lênh.

Chúng ta cũng sử dụng RS = 0 để kiểm tra bít cờ bận để xem LCD có sẵn sàng nhân thông tin. Cờ bận là D7 và có thể đượcđọc khi R/W = 1 và RS = 0 như sau:

Nếu R/W = 1, RS = 0 khi D7 = 1 (cờ bận 1) thì LCD bận bởi các công việc bên trong và sẽ không nhận bất kỳ thông tin mới nào. Khi D7 = 0 thì LCD sẵn sàng nhận thông tin mới. Lưu ý chúng ta nên kiểm tra cờ bận trước khi ghi bất kỳ dữ liệu nào lên LCD.

Bảng 12.1: Mô tả các chân của LCD.

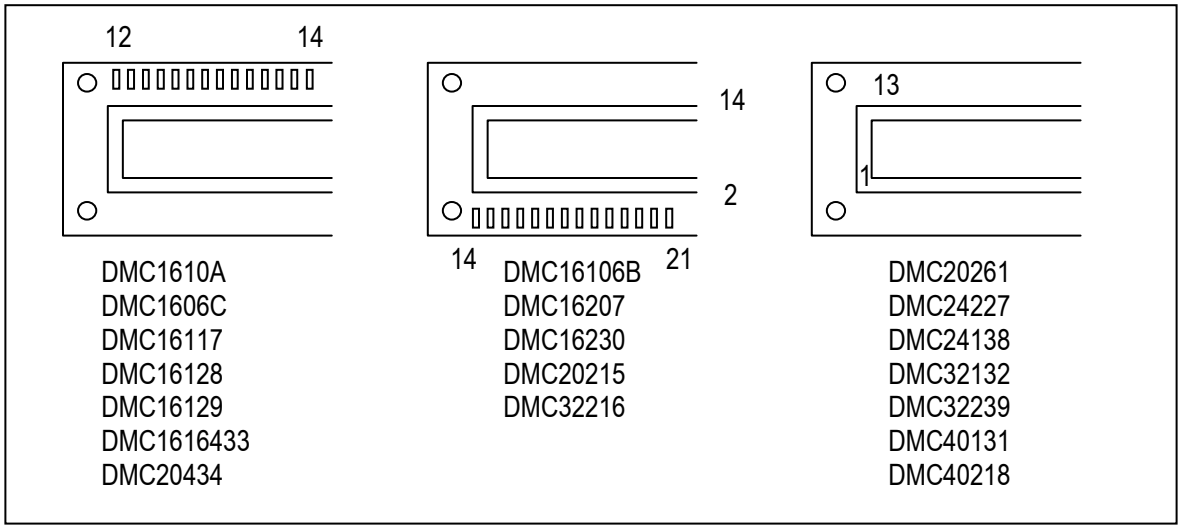
Chân	Ký hiệu	I/O	Mô tả
1	V_{SS}	-	Đất
2	V_{CC}	-	Dương nguồn 5v
3	V_{EE}	-	Cấp nguồn điều khiển phản
4	RS	I	RS = 0 chọn thanh ghi lệnh. RS = 1 chọn thanh dữ liệu
5	R/W	I	R/W = 1 đọc dữ liệu. R/W = 0 ghi
6	E	I/O	Cho phép
7	DB0	I/O	Các bít dữ liệu
8	DB1	I/O	Các bít dữ liệu
9	DB2	I/O	Các bít dữ liệu
10	DB3	I/O	Các bít dữ liệu
11	DB4	I/O	Các bít dữ liệu
12	DB5	I/O	Các bít dữ liệu
13	DB6	I/O	Các bít dữ liệu
14	DB7	I/O	Các bít dữ liệu

Bảng 12.2: Các mã lệnh LCD.

Mã (Hex)	Lệnh đến thanh ghi của LCD
1	Xoá màn hình hiển thị
2	Trở về đầu dòng
4	Giả con trỏ (dịch con trỏ sang trái)
6	Tăng con trỏ (dịch con trỏ sang phải)
5	Dịch hiển thị sang phải

7	Dịch hiển thị sang trái
8	Tắt con trỏ, tắt hiển thị
A	Tắt hiển thị, bật con trỏ
C	Bật hiển thị, tắt con trỏ
E	Bật hiển thị, nhấp nháy con trỏ
F	Tắt con trỏ, nhấp nháy con trỏ
10	Dịch vị trí con trỏ sang trái
14	Dịch vị trí con trỏ sang phải
18	Dịch toàn bộ hiển thị sang trái
1C	Dịch toàn bộ hiển thị sang phải
80	Ép con trỏ Vũ đầu dòng thứ nhất
C0	Ép con trỏ Vũ đầu dòng thứ hai
38	Hai dòng và ma trận 5×7

Ghi chú: Bảng này được mở rộng từ bảng 12.4.



Hình 12.1: Các vị trí chân của các LCD khác nhau của Optrex.

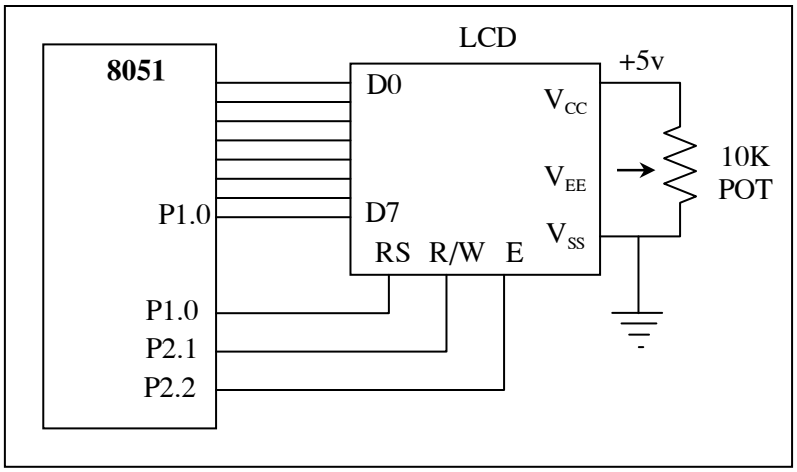
12.1.3 Gửi các lệnh và dữ liệu đến LCD với một độ trễ.

Để gửi một lệnh bất kỳ từ bảng 12.2 đến LCD ta phải đưa chân RS về 0. Đối với dữ liệu thì bật RS = 1 sau đó gửi một sườn xung cao xuống thấp đến chân E để cho phép chốt dữ liệu trong LCD. Điều này được chỉ ra trong đoạn mã chương trình dưới đây (xem hình 12.2).

```
; gọi độ thời gian trễ trước khi gửi dữ liệu/ lệnh kế tiếp.
; chân P1.0 đến P1.7 được nối tới chân dữ dữ liệu D0 - D7 của  LCD.
; Chân P2.0 được nối tới chân RS của LCD.
; Chân P2.1 được nối tới chân R/W của LCD.
; Chân P2.2 được nối đến chân E của LCD.
```

```
        ORG
        MOV     A, # 38H            ; Khởi tạo LCD hai dòng với ma trận
5 × 7
        ACALL       COMNWRT  ; Gọi chương trình con lệnh
        ACALL       DELAY           ; Cho LCD một độ trễ
        MOV     A, # 0EH    ; Hiển thị màn hình và con trỏ
        ACALL       COMNWRT  ; Gọi chương trình con lênh
        ACALL       DELAY           ; Cấp một độ trễ cho LCD
        MOV     AM # 01             ; Xoá LCD
        ACALL       COMNWRT  ; Gọi chương trình con lệnh
        ACALL       DELAY           ; Tạo độ trễ cho LCD
        MOV     A, # 06H            ; Dịch con trỏ sang phải
        ACALL       COMNWRT  ; Gọi chương trình con lệnh
        ACALL       DELAY           ; Tạo độ trễ cho LCD
        MOV     AM # 48H    ; Đưa con trỏ về dòng 1 cột 4
        ACALL       COMNWRT  ; Gọi chương trình con lệnh
        ACALL       DELAY           ; Tạo độ trễ cho LCD
        MOV     A, # “N”            ; Hiển thị chữ N
        ACALL       DATAWRT  ; Gọi chương trình con hiển thij
DISPLAY
        ACALL       DELAY           ; Tạo độ trễ cho LCD
        MOV     AM # “0”            ; Hiển thị chữ 0
        ACALL       DATAWRT  ; Gọi DISPLAY
AGAIN:      SJMP        AGAIN       ; Chờ ở đây
COMNWRT:                            ; Gửi lệnh đến LCD
        MOV     P1, A       ; Sao chép thanh ghi A đến cổng P1
        CLR     P2.0        ; Đặt RS = 0 để gửi lệnh
        CLR     P2.1        ; Đặt R/W = 0 để ghi dữ liệu
        SETB    P2.2        ; Đặt E = 1 cho xung cao
        CLR     P2.2        ; Đặt E = 0 cho xung cao xuống thấp
        RET
DATAWRT:                            ; Ghi dữ liệu ra LCD
        MOV     P1, A       ; Sao chép thanh ghi A đến cổng P1
        SETB    P2.0        ; Đặt RS = 1 để gửi dữ liệu
        CLR     P2.1        ; Đặt R/W = 0 để ghi
        SETB    P2.2        ; Đặt E = 1 cho xung cao
        CLR     P2.2        ; Đặt E = 0 cho xung cao xuống thấp
        RET
DELAY:  MOV     R3, # 50            ; Đặt độ trễ 50µs hoặc cao hơn cho
CPU nhanh
HERE2:      MOV     R4, # 255   ; Đặt R4 = 255
```

```
HERE:       DJNZ    R4, HERE    ; Đợi ở đây cho đến khi R4 = 0
            DJNZ    R3, HERE2
            RET
            END
```



Hình 12.2: Nối ghép LCD.

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

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

```
; Kiểm tra cờ bận trước khi gửi dữ liệu, lệnh ra LCD
; Đặt P1 là cổng dữ liệu
; Đặt P2.0 nối tới cổng RS
; Đặt P2.1 nối tới chân R/W
; Đặt P2.2 nối tới chân E
            ORG
            MOV     A, # 38H            ; Khởi tạo LCD hai dòng với ma trận 5 × 7
            ACALL       COMMAND     ; Xuất lệnh
            MOV     A, # 0EH        ; Dịch con trỏ sang phải
            ACALL       COMMAND     ; Xuất lệnh
            MOV     A, # 01H            ; Xoá lệnh LCD
            ACALL       COMMAND     ; Xuất lệnh
            MOV     A, # 86H            ; Dịch con trỏ sang phải
            ACALL       COMMAND     ; Đưa con trỏ về dòng 1 lệnh 6
            MOV     A, # "N"            ; Hiển thị chữ N
            ACALL       DATA DISPLAY
```

```
        MOV     A, # "0"            ; Hiển thị chữ 0
        ACALL           DATA DISPLAY
HERE:       SJMP        HERE        ; Chờ ở đây
COMMAND: ACALL          READY           ; LCD đã sẵn sàng chưa?
        MOV     P1, A       ; Xuất mã lệnh
        CLR     P2.0        ; Đặt RS = 0 cho xuất lệnh
        CLR     P2.1        ; Đặt R/W = 0 để ghi dữ liệu tới LCD
        SETB    P2.2        ; Đặt E = 1 đối với xung cao xuống thấp
        CLR     P2.2        ; Đặt E = 0 chốt dữ liệu
        RET
DATA-DISPLAY::
        ACALL           READY           ; LCD đã sẵn sàng chưa?
        MOV     P1, A       ; Xuất dữ liệu
        SETB    P2.0        ; Đặt RS = 1 cho xuất dữ liệu
        CLR     P2.1        ; Đặt R/W = 0 để ghi dữ liệu ra LCD
        SETB    P2.2        ; Đặt E = 1 đối với xung cao xuống thấp
        CLR     P2.2        ; Đặt E = 0 chốt dữ liệu
        RET
DELAY:
        SETB    P1.7        ; Lấy P1.7 làm cổng vào
        CLR     P2.0        ; Đặt RS = 0 để truy cập thanh ghi lệnh
        SETB    P2.1        ; Đặt R/W = 1 đọc thanh ghi lệnh
; Đọc thanh ghi lệnh và kiểm tra cờ lệnh
BACK:       CLR     P2.2        ; E = 1 đối với xung cao xuống thấp
        SETB    P2.2        ; E = 0 cho xung cao xuống thấp?
        JB      P1.7, BACK  ; Đợi ở đây cho đến khi cờ bận = 0
        RET
        END
```

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

12.1.5 Bảng dữ liệu của LCD.

Trong LCD ta có thể đặt dữ liệu vào bất cứ chỗ nào. dưới đây là các vị trí địa chỉ và cách chúng được truy cập.

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

Khi AAAAAAA = 0000000 đến 0100111 cho dòng lệnh 1 và AAAAAAA = 1100111 cho dòng lệnh2. Xem bảng 12.3.

Bảng 12.3: Đánh địa chỉ cho LCD.

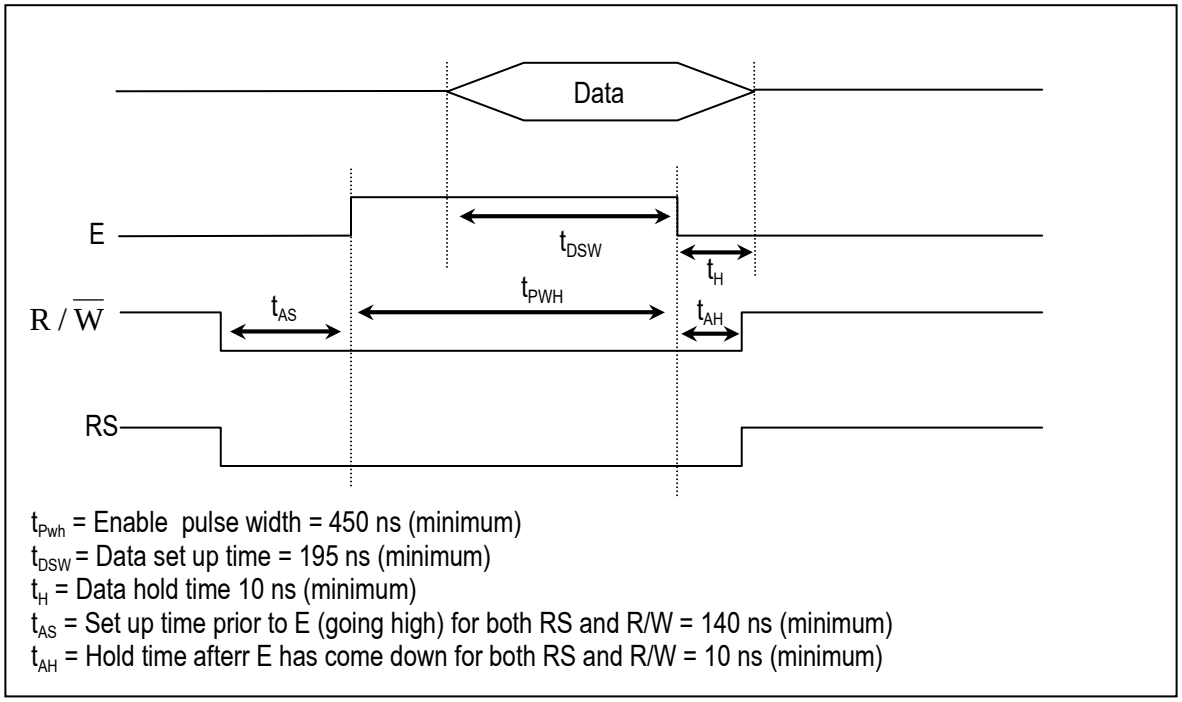
	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Dòng 1 (min)	1	0	0	0	0	0	0	0
Dòng 1 (max)	1	0	1	0	0	1	1	1
Dòng 2 (min)	1	1	0	0	0	0	0	0
Dòng 2 (max)	1	1	1	0	0	1	1	1

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

Từ những điều nói ở trên đây ta có thể nhận được các địa chỉ của vị trí con trỏ có các kích thước LCD khác nhau. Xem hình 12.3 chú ý rằng tất cả mọi địa chỉ đều ở dạng số Hex. Hình 12.4 cho một biểu đồ của việc phân thời gian của LCD. Bảng 12.4 là danh sách liệt kê chi tiết các lệnh và chỉ lệnh của LCD. Bảng 12.2 được mở rộng từ bảng này.

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

Hình 12.3: Các địa chỉ con trỏ đối với một số LCD.



t_{Pwh} = Enable pulse width = 450 ns (minimum)
t_{DSW} = Data set up time = 195 ns (minimum)
t_H = Data hold time 10 ns (minimum)
t_{AS} = Set up time prior to E (going high) for both RS and R/W = 140 ns (minimum)
t_{AH} = Hold time afterr E has come down for both RS and R/W = 10 ns (minimum)

Hình 12.4: Phân khe thời gian của LCD.

Bảng 12.4: Danh sách liệt kê các lệnh và địa chỉ lệnh của LCD.

Lệnh	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Mô tả	Thời gian thực hiện
Xoá màn hình	0	0	0	0	0	0	0	0	0	1	Xoá toàn bộ màn hình và đặt địa chỉ 0 của DD RAM vào bộ đếm địa chỉ	1.64 µs
Trở về đầu dòng	0	0	0	0	0	0	0	0	1	-	Đặt địa chỉ 0 của DD RAM như bộ đếm địa chỉ. Trả hiển thị dịch về vị trí gốc DD RAM không thay đổi	1.64 µs
Đặt chế độ truy nhập	0	0	0	0	0	0	0	1	I/D	S	Đặt hướng chuyển dịch con trỏ và xác định dịch hiển thị các thao tác này được thực hiện khi đọc và ghi dữ liệu	40 µs

Điều khiển Bật/tắt hiển thị	0	0	0	0	0	0	1	D	C	B	Đặt Bật/ tắt màn hình (D) Bật/ tắt con trỏ (C) và nhấp nháy ký tự ở vị trí con trỏ (B)	40 µs
Dịch hiển thị và con trỏ	0	0	0	0	0	1	S/C	R/L	-	-	Dịch con trỏ và dịch hiển thị mà không thay đổi DD RAM	40 µs
Đặt chức năng	0	0	0	0	1	DL	N	F	-	-	Thiết lập độ dài dữ liệu (DL) số dòng hiển thị (L) và phòng ký tự (F)	40 µs
Đặt địa chỉ CGRAM	0	0	0	1	AGC						Thiết lập địa chỉ C6 RAM dữ liệu CG RAM được gửi đi và nhận sau thiết lập này	40 µs
Thiết lập địa chỉ DD RAM	0	0	1	ADD							Thiết lập địa chỉ DD RAM dữ liệu DD RAM được gửi và nhận sau thiết lập này	40 µs
Cờ bận đọc và địa chỉ	0	1	BF	ADD							Cờ bận đọc (BF) báo hoạt động bên trong đang được thực hiện và đọc nội dung bộ đếm địa chỉ	40 µs
Ghi dữ liệu CG hoặc DD RAM	1	0	Ghi dữ liệu								Ghi dữ liệu vào DD RAM hoặc CG RAM	40 µs

Đọc dữ liệu CG hoặc DD RAM	1	1	Đọc dữ liệu	Đọc dữ liệu từ DD RAM hoặc CG RAM	40 μs

Ghi chú:

1. Thời gian thực là thời gian cực đại khi tần số f_{CP} hoặc f_{osc} là 250KHz
2. Thời gian thực thay đổi khi tần số thay đổi. Khi tần số f_{EP} hay f_{osc} Là 270kHz thì thời gian thực hiện được tính $250/270 \times 40 = 35\mu s$ v.v…
3. Các ký hiệu viết tắt trong bảng là:
4.

DD RAM RAM dữ liệu hiển thị (Display Data RAM)
CG RAM RAM máy phát ký tự (character Generator)
ACC Địa chỉa của RAM máy phát ký tự
ADD Địa chỉ của RAM dữ liệu hiển thị phù hợp với địa chỉ con trỏ.
AC Bộ đếm địa chỉ (Address Counter) được dùng cho các địa chỉ DD RAM và CG RAM.
1/D = 1 Tăng 1/D = 0 Giảm
S = 1 Kèm dịch hiển thị
S/C = 1 Dịch hiển thị S/C = 0 Dịch con trỏ
R/L = 1 Dịch sang phải R/L = 0 Dịch trái
DL = 1 8 bít DL = 0 4 bít
N = 1 2 dòng N = 1 1 dòng
F = 1 Ma trận điểm 5×10 F = 0 Ma trận điểm 5×7
BF = 1 Bận BF = 0 Có thể nhận lênh

12.2 Phối ghép 8051 với ADC và các cảm biến.

Phần này sẽ khám phá ghép các chíp ADC (bộ chuyển đổi tương tự số) và các cảm biến nhiệt với 8051.

12.1.1 Các thiết bị ADC.

Các bộ chuyển đổi ADC thuộc trong những thiết bị được sử dụng rộng rãi nhất để thu dữ liệu. Các máy tính số sử dụng các giá trị nhị phân, nhưng trong thế giới vật lý thì mọi đại lượng ở dạng tương tự (liên tục). Nhiệt độ, áp suất (khí hoặc chất lỏng), độ ẩm và vận tốc và một số ít trọng những đại lượng vật lý của thế giới thực mà ta gặp hàng ngày. Một đại lượng vật lý được chuyển về dòng điện hoặc điện áp qua một thiết bị được gọi là các bộ biến đổi.

Các bộ biến đổi cũng có thể được coi như các bộ cảm biến. Mặc dù chỉ có các bộ cảm biến nhiệt, tốc độ, áp suất, ánh sáng và nhiều đại lượng tự nhiên khác nhưng chúng đều cho ra các tín hiệu dạng dòng điện hoặc điện áp ở dạng liên tục. Do vậy, ta cần một bộ chuyển đổi tương tự số sao cho bộ vi điều khiển có thể đọc được chúng. Một chíp ADC được sử dụng rộng rãi là ADC 804.

12.2.2 Chíp ADC 804.

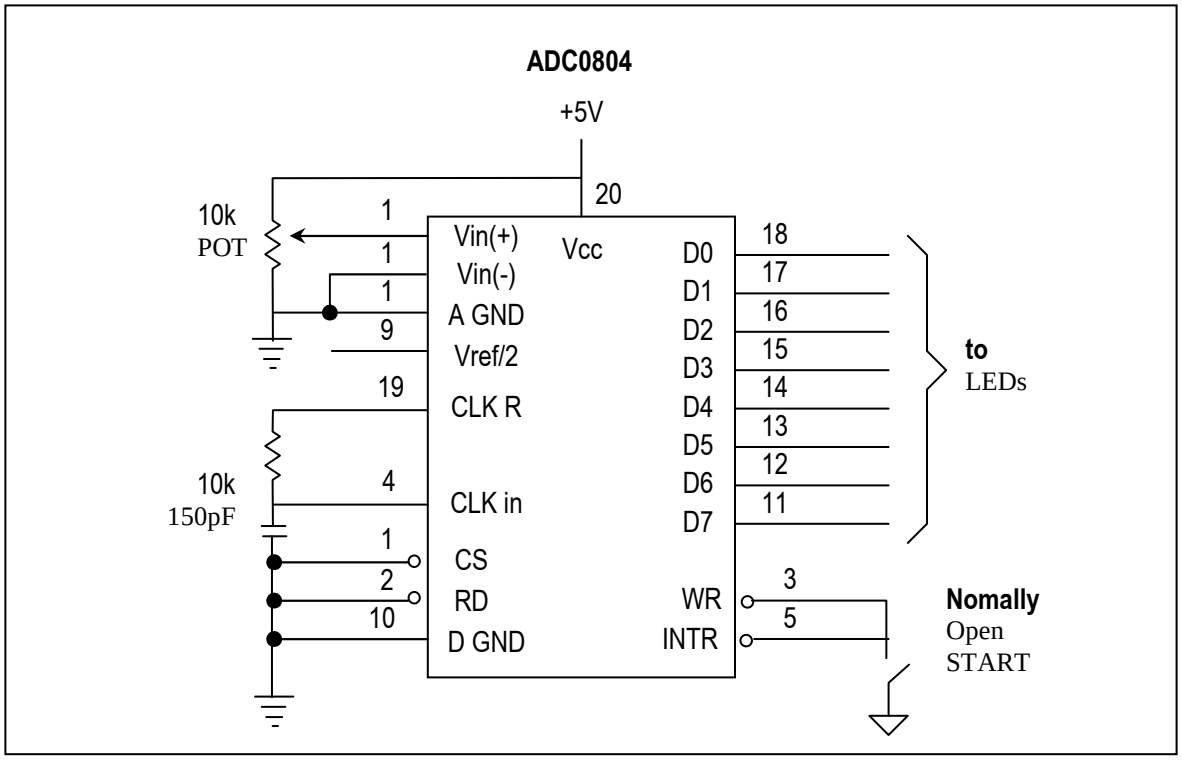
Chíp ADC 804 là bộ chuyển đổi tương tự số trong họ các loạt ADC 800 từ hãng National Semiconductor. Nó cũng được nhiều hãng khác sản xuất, nó làm việc với +5v và có độ phân giải là 8 bít. Ngoài độ phân giải thì thời gian chuyển đổi cũng là một yếu tố quan trọng khác khi đánh giá một bộ ADC. Thời gian chuyển đổi được định nghĩa như là thời gian mà bộ ADC cần để chuyển một đầu vào tương tự thành một số nhị phân. Trong ADC 804 thời gian chuyển đổi thay đổi phụ thuộc vào tần số đồng hồ được cấp tới chân CLK và CLK IN nhưng không thể nhanh hơn 110μs. Các chân của ADC 804 được mô tả như sau:

1. Chân $\overline{CS}$ - chọn chíp: Là một đầu vào tích cực mức thấp được sử dụng để kích hoạt chíp ADC 804. Để truy cập ADC 804 thì chân này phải ở mức thấp.
2. Chân $\overline{RD}$ (đọc): Đây là một tín hiệu đầu vào được tích cực mức thấp. Các bộ ADC chuyển đổi đầu vào tương tự thành số nhị phân tương đương với nó và giữ nó trong một thanh ghi trong. $\overline{RD}$ được sử dụng để nhận dữ liệu được chuyển đổi ở đầu ra của ADC 804. Khi CS = 0 nếu một xung cao - xuống - thấp được áp đến chân $\overline{RD}$ thì đầu ra số 8 bít được hiển diện ở các chân dữ liệu D0 - D7. Chân $\overline{RD}$ cũng được coi như cho phép đầu ra.
3. Chân ghi $\overline{WR}$ (thực ra tên chính xác là "Bắt đầu chuyển đổi"). Đây là chân đầu vào tích cực mức thấp được dùng để báo cho ADC 804 bắt đầu quá trình chuyển đổi. Nếu CS = 0 khi $\overline{WR}$ tạo ra xung cao - xuống - thấp thì bộ ADC 804 bắt đầu chuyển đổi giá trị đầu vào tương tự V_{in} về số nhị phân 8 bít. Lượng thời gian cần thiết để chuyển đổi thay đổi phụ thuộc vào tần số đưa đến chân CLK IN và CLK R. Khi việc chuyển đổi dữ liệu được hoàn tất thì chân INTR được ép xuống thấp bởi ADC 804.
4. Chân CLK IN và CLK R.

Chân CLK IN là một chân đầu vào được nối tới một nguồn đồng hồ ngoài khi đồng hồ ngoài được sử dụng để tạo ra thời gian. Tuy nhiên 804 cũng có một máy tạo xung đồng hồ. Để sử dụng máy tạo xung đồng hồ trong (cũng còn được gọi là máy tạo đồng hồ riêng) của 804 thì các chân CLK IN và CLK R được nối tới một tụ điện và một điện trở như chỉ ra trên hình 12.5. Trong trường hợp này tần số đồng hồ được xác định bằng biểu thức:

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

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



Hình 12.5: Kiểm tra ADC 804 ở chế độ chạy tự do.

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

Đây là chân đầu ra tích cực mức thấp. Bình thường nó ở trạng thái cao và khi việc chuyển đổi hoàn tất thì nó xuống thấp để báo cho CPU biết là dữ liệu được chuyển đổi sẵn sàng để lấy đi. Sau khi $\overline{\text{INTR}}$ xuống thấp, ta đặt CS = 0 và gửi một xung cao 0 xuống - thấp tới chân $\overline{\text{RD}}$ lấy dữ liệu ra của 804.

6. Chân V_{in} (+) và V_{in} (-).

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

7. Chân V_{CC}.

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

8. Chân $V_{ref/2}$.

Chân 9 là một điện áp đầu vào được dùng cho điện áp tham chiếu. Nếu chân này hở (không được nối) thì điện áp đầu vào tương tự cho ADC 804 nằm trong dải 0 đến +5v (giống như chân V_{CC}). Tuy nhiên, có nhiều ứng dụng mà đầu vào tương tự áp đến V_{in} cần phải khác ngoài dải 0 đến 5v. Chân $V_{ref/2}$ đượcdùng để thực thi các điện áp đầu vào khác ngoài dải 0 - 5v. Ví dụ, nếu dải đầu vào tương tự cần phải là 0 đến 4v thì $V_{ref/2}$ được nối với +2v.

Bảng 12.5 biểu diễn dải điện áp V_{in} đối với các đầu vào $V_{ref/2}$ khác nhau.

Bảng 12.5: Điện áp $V_{ref/2}$ liên hệ với dải V_{in}.

V_{ref}/ 2(V)	V_{in}(V)	Step Size (mV)
Hở *	0 đến 5	5/256 = 19.53
2.0	0 đến 4	4/255 = 15.62
1.5	0 đến 3	3/256 = 11.71
1.28	0 đến 2.56	2.56/256 = 10
1.0	0 đến 2	2/256 = 7.81
0.5	0 đến 1	1/256 = 3.90

Ghi chú: - V_{CC} = 5V

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

9. Các chân dữ liệu D0 - D7.

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

$$D_{out} = \frac{V_{in}}{\text{kich thuoc buoc}}$$

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

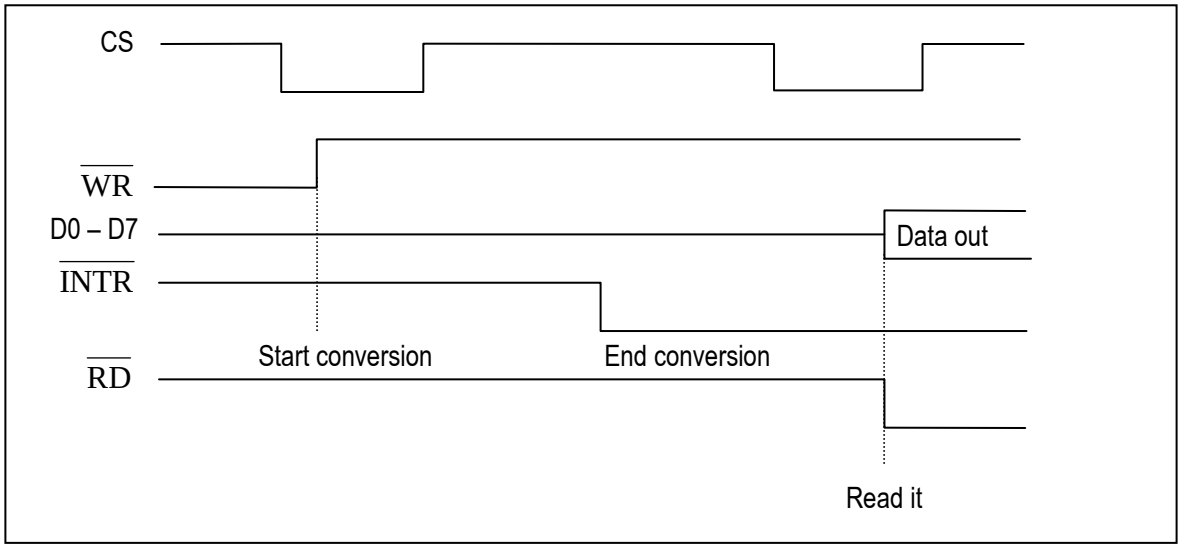
10. Chân đất tương tự và chân đất số.

Đây là những chân đầu vào cấp đất chung cho cả tín hiệu số và tương tự. Đất tương tự được nối tới đất của chân V_{in} tương tự, còn đất số được nối tới đất của chân V_{cc}. Lý do mà ta phải có hai đất là để cách ly tín hiệu tương tự V_{in} từ các điện áp ký sinh tạo ra việc chuyển mạch số được chính xác. Trong phần trình bày của chúng ta thì các chân này được nối chung với một đất. Tuy nhiên, trong thực tế thu đo dữ liệu các chân đất này được nối tách biệt.

Từ những điều trên ta kết luận rằng các bước cần phải thực hiện khi chuyển đổi dữ liệu bởi ADC 804 là:

a) Bật CS = 0 và gửi một xung thấp lên cao tới chân $\overline{WR}$ để bắt đầu chuyển đổi.
b) Duy trì hiển thị chân $\overline{INTR}$. Nếu $\overline{INTR}$ xuống thấp thì việc chuyển đổi được hoàn tất và ta có thể sang bước kế tiếp. Nếu $\overline{INTR}$ cao tiếp tục thăm dò cho đến khi nó xuống thấp.

c) Sau khi chân $\overline{INTR}$ xuống thấp, ta bật CS = 0 và gửi một xung cao - xuống - thấp đến chân $\overline{RD}$ để lấy dữ liệu ra khỏi chíp ADC 804. Phân chia thời gian cho quá trình này được trình bày trên hình 12.6.

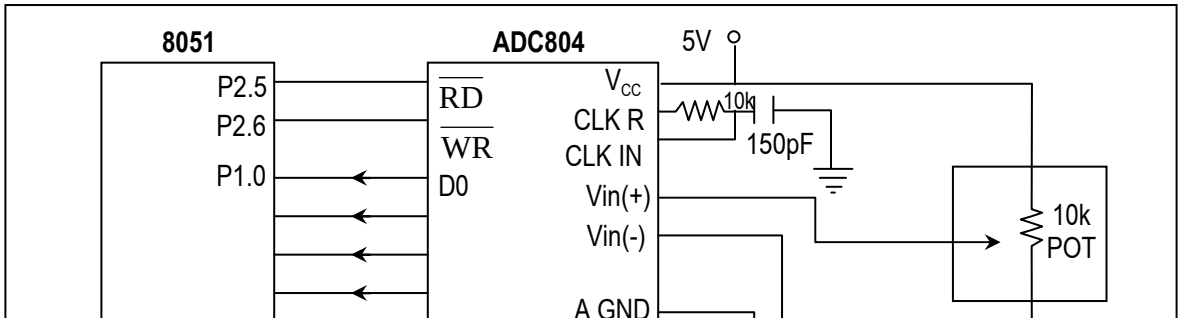


Hình 12.6: Phân chia thời gian đọc và ghi của ADC 804.

12.2.3 Kiểm tra ADC 804.

Chúng ta có thể kiểm tra ADC 804 bằng cách sử dụng sơ đồ mạch trên hình 12.7. thiết lập này được gọi là chế độ kiểm tra chạy tự do và được nhà sản xuất khuyến cao nên sử dụng. Hình 12.5 trình bày một biến trở được dùng để cấp một điện áp tương tự từ 0 đến 5V tới chân đầu vào.

$V_{in}(+)$ của ADC 804 các đầu ra nhị phân được hiển thị trên các đèn LED của bảng huấn luyện số. Cần phải lưu ý rằng trong chế độ kiểm tra chạy tự do thì đầu vào CS được nối tới đất và đầu vào $\overline{WR}$ được nối tới đầu ra $\overline{INTR}$. Tuy nhiên, theo tài liệu của hãng National Semiconductor "nút WR và INTR phải được tạm thời đưa xuống thấp kế sau chu trình cấp nguồn để bảo đảm hoạt động".



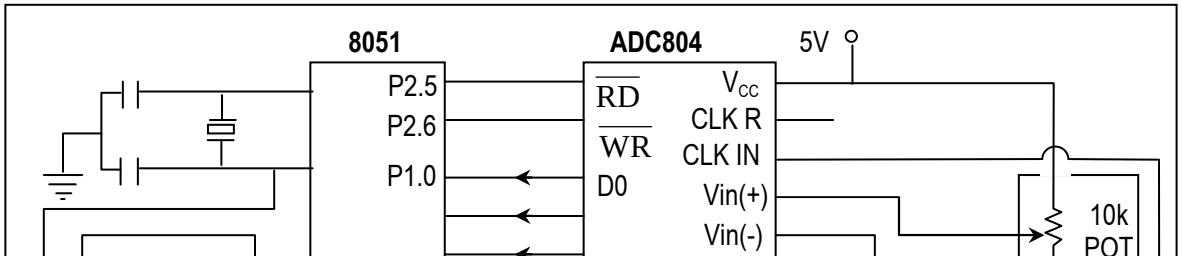
Hình 12.7: Nối ghép ADC 804 với nguồn đồng hồ riêng.

Ví dụ 12.7:

Hãy thử nối ghép ADC 804 với 8051 theo sơ đồ 12.7. Viết một chương trình để hiển thị chân INTR và lấy đầu vào tương tự vào thanh ghi A. Sau đó gọi một chương trình chuyển đổi mã Hex ra ASCII và một chương trình hiển thị dữ liệu. Thực hiện điều này liên tục.

Lời giải:

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



Hình 12.8: Nối ghép ADC 804 với đồng hồ từ XTAL2 của 8051.

Trên hình 12.8 ta có thể thấy rằng tín hiệu đồng hồ đi vào ADC 804 là từ tần số thạch anh của 8051. Vì tần số này quá cao nên ta sử dụng hai mạch lật Rlip - Flop kiểu D (74LS74) để chia tần số này cho 4. Một mạch lật chia tần số cho 2 nếu ta nối đầu $\overline{Q}$ tới đầu vào D. Đối với tần số cao hơn thì ta cần sử dụng nhiều mạch Flip - Plop hơn.

12.2.4 Phối ghép với một cảm biến nhiệt của 8051.

Các bộ biến đổi (Transducer) chuyển đổi các đại lượng vật lý ví dụ như nhiệt độ, cường độ ánh sáng, lưu tốc và tốc độ thành các tín hiệu điện phụ thuộc vào bộ biến đổi mà đầu ra có thể là tín hiệu dạng điện áp, dòng, trở kháng hay dung kháng. Ví dụ, nhiệt độ được biến đổi thành về các tín hiệu điện sử dụng một bộ biến đổi gọi là Rhermistor (bộ cảm biến nhiệt), một bộ cảm biến nhiệt đáp ứng sự thay đổi nhiệt độ bằng cách thay đổi trở kháng nhưng đáp ứng của nó không tuyến tính (xem bảng 12.6).

Bảng 12.6: Trở kháng của bộ cảm biến nhiệt theo nhiệt độ.

Nhiệt độ (^{0}C)	**Trở kháng của cảm biến (kΩ)**
0	29.490
25	10.000
50	3.893
75	1.700
100	0.817

Bảng 12.7: Hướng dẫn chọn loạt các cảm biến họ LM34.

Mã ký hiệu	Dải nhiệt độ	Độ chính xác	Đầu ra
LM34A	-55 F to + 300 C	+ 2.0 F	10mV/F
LM34	-55 F to + 300 C	+ 3.0 F	10mV/F
LM34CA	-40 F to + 230 C	+ 2.0 F	10mV/F
LM34C	-40 F to + 230 C	+ 3.0 F	10mV/F
LM34D	-32 F to + 212 C	+ 4.0 F	10mV/F

Bảng 12.8: Hướng dẫn chọn loạt các cảm biến nhiệt họ LM35.

Mã sản phẩm	Dải nhiệt độ	Độ chính xác	Đầu ra
LM35A	-55 C to + 150 C	+ 1.0 C	10 mV/F
LM35	-55 C to + 150 C	+ 1.5 C	10 mV/F
LM35CA	-40 C to + 110 C	+ 1.0 C	10 mV/F
LM35C	-40 C to + 110 C	+ 1.5 C	10 mV/F
LM35D	0 C to + 100 C	+ 2.0 C	10 mV/F

Tính chất gắn liện với việc viết phần mềm cho các thiết bị phi tuyến như vậy đã đưa nhiều nhà sản xuất tung ra thị trường các loạt bộ cảm biến nhiệt tuyến tính. Các bộ cảm biến nhiệt đơn giản và được sử dụng rộng rãi bao gồm các loạt họ LM34 và LM35 của hãng National Semiconductor Corp.

12.2.5 Các bộ cảm biến nhiệt họ LM34 và LM35.

Loạt các bộ cảm biến LM34 là các bộ cảm biến nhiệt mạch tích hợp chính xác cao mà điện áp đầu ra của nó tỷ lệ tuyến tính vơi nhiệt độ Fahrenheit (xem hình 12.7). loạt LM34 không yêu cầu cân chỉnh bên ngoài vì vốn nó đã được cân chỉnh rồi. Nó đưa ra điện áp 10mV cho sự thay đổi nhiệt độ 1^0F. bảng 12.7 hướng dẫn ta chọn các cảm biến loạt LM34.

Loạt các bộ cảm biến LM35 cũng là các bộ cảm biến nhiệt mách tích hợp chính xác cao mà điện áp đầu ra của nó tỷ lệ tuyến tính với nhiệt độ theo thang độ Celsius. Chúng cũng không yêu cầu cân chỉnh ngoài vì vốn chúng đã được cân chỉnh. Chúng đưa ra điện áp 10Mv cho mỗi sự thay đổi 1^0C. Bảng 12.8 hướng dẫn ta chọn các cảm biến họ LM35.

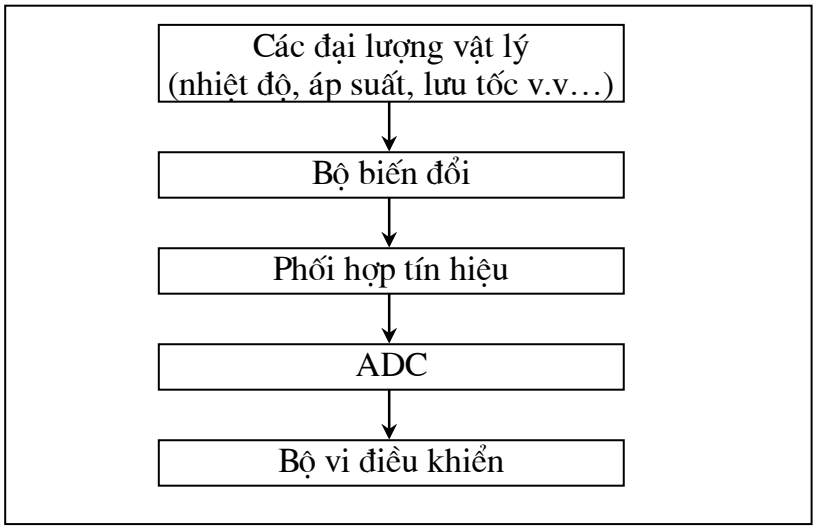
12.2.6 Phối hợp tín hiệu và phối ghép LM35 với 8051.

Phối hợp tín hiệu là một thuật ngữ được sử dụng rộng rãi trong lĩnh vực thu đo dữ liệu. Hầu hết các bộ biến đổi đều đưa ra các tín hiệu điện dạng điện áp, dòng điện, dung kháng hoặc trở kháng. Tuy nhiên, chúng ta cần chuyển đổi các tín hiệu này về điện áp nhằm gửi đầu vào đến bộ chuyển đổi ADC. Sự chuyển đổi (biến đổi) này được gọi chung là phối hợp tín hiệu. Phối hợp tín hiệu có thể là việc chuyển đổi dòng điện thành điện áp hoặc sự khuyếch đại tín hiệu. Ví dụ, bộ cảm biến nhiệt thay đổi trở kháng với nhiệt độ. Sự thay đổi trở kháng phải được chuyển thành điện áp để có thể được sử dụng cho các ADC. Xét trường hợp nối một LM35 tới một ADC 804 vì ADC 804 có độ phân dải 8 bít với tối đa 256 bước (2^8) và LM35 (hoặc ML34) tạo điện áp 10mV cho mỗi

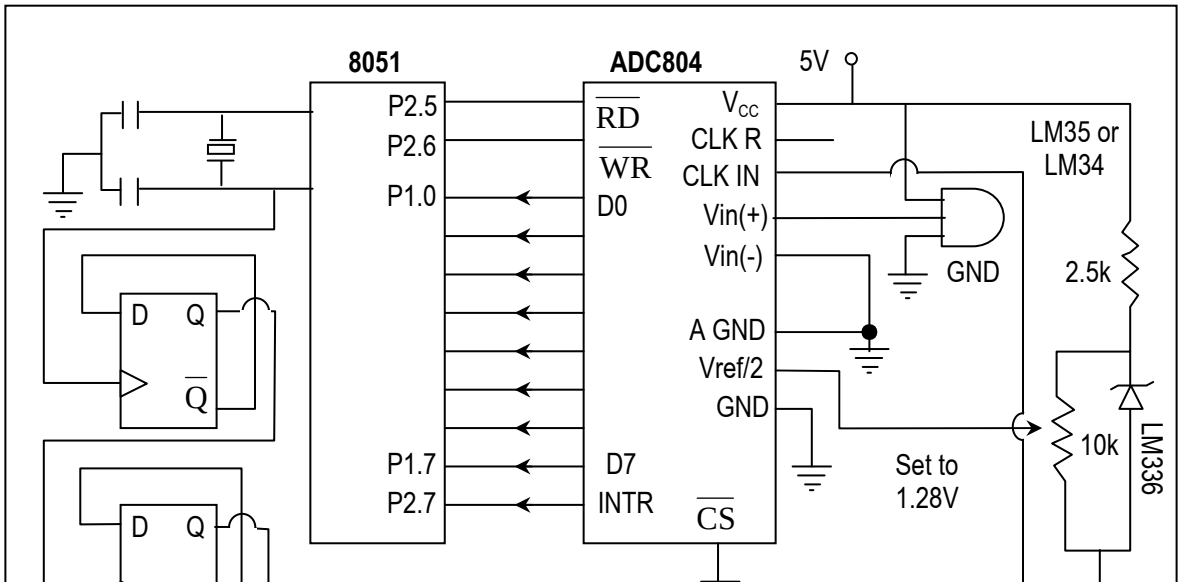
sự thay đổi nhiệt độ 1^0C nên ta có thể tạo điều kiện V_{in} của ADC 804 tạo ra một V_{out} = 2560mV (2,56V) cho đầu ra đầu thang đo. do vậy, nhằm tạo ra V_{out} đầy thang 2,56V cho ADC 804 ta cần đặt điện áp $V_{ref}/2$ = 1,28V. Điều này làm cho V_{out} của ADC 804 đáp ứng trực tiếp với nhiệt độ được hiển thị trên LM35 (xem bảng 12.9). Các giá trị của $V_{ref}/2$ được cho ở bảng 12.5.

Bảng 12.9: Nhiệt độ.

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



Hình 12.9: Thu đo các đại lượng vật lý.



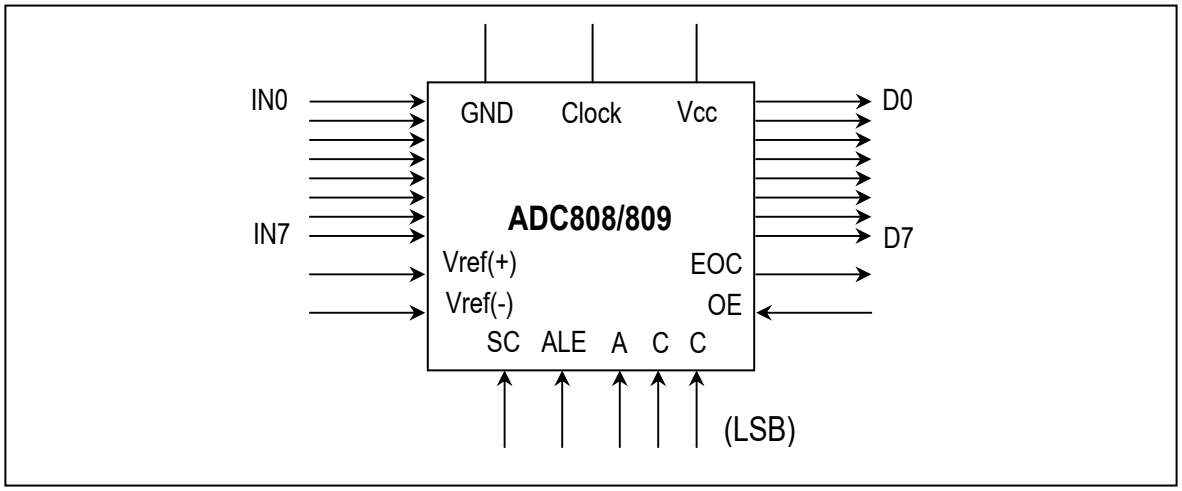
Hình 12.10

Hình 12.10: Nối ghép 8051 với DAC 804 và cảm biến nhiệt độ.

Hình 12.10 biểu diễn nối ghép của bộ cảm biến nhiệt đến ADC 804. Lưu ý rằng ta sử dụng đi ốt zener LM336 - 2.5 để cố định điện áp qua biến trở 10kΩ tại 2,5V. Việc sử dụng LM336 - 2.5 có thể vượt qua được mọi dao động lên xuống của nguồn nuôi.

12.2.7 Chíp ADC 808/809 với 8 kênh tương tự.

Một chíp hữu ích khác của National Semiconductor là ADC 808/809 (xem hình 12.11). Trong khi ADC 804 chỉ có một đầu vào tương tự thì chíp này có 8 kênh đầu vào. Như vậy nó cho phép ta hiển thị lên 8 bộ biến đổi khác nhau chỉ qua một chíp duy nhất. Lưu ý rằng, ADC 808/809 có đầu ra dữ liệu 8 bít như ADC 804. 8 kênh đầu vào tương tự được dồn kênh và được chọn theo bảng 12.10 sử dụng ba chân địa chỉ A, B và C.



Hình 12.11: Bộ biến đổi ADC 808/809.

Bảng 12.10: Chọn kênh tương tự của ADC 808.

Chọn kênh tương tự	C	B	A
IN0	0	0	0
IN1	0	0	1
IN2	0	1	0

IN3	0	1	1
IN4	1	0	0
IN5	1	0	1
IN6	1	1	0
IN7	1	1	1

Trong ADC 808/809 thì $V_{rer}(+)$ và $V_{ref}(-)$ thiết lập điện áp tham chiếu. Nếu $V_{ref}(-1)$ = Gnd và $V_{ref}(+)$ = 5V thì độ phân dải là 5V/256 = 19,53mV. Do vậy, để có độ phân dải 10mV ta cần đặt $V_{ref}(+)$ = 2,56V và $V_{ref}(-)$ = Gnd. Từ hình 12.11 ta thấy có chân ALE. Ta sử dụng các địa chỉ A, B và C để chọn kênh đầu vào IN0 – IN7 và kích hoạt chân ALE để chốt địa chỉ. Chân SetComplete để bắt đầu chuyển đổi (Start Conversion). Chân EOC được dùng để kết thúc chuyển đổi (End - Of - Conversion) và chân OE là cho phép đọc đầu ra (Out put Enable).

12.2.7 Các bước lập trình cho ADC 808/809.

Các bước chuyển dữ liệu từ đầu vào của ADC 808/809 vào bộ vi điều khiển như sau:

1. Chọn một kênh tương tự bằng cách tạo địa chỉ A, B và C theo bảng 12.10.
2. Kích hoạt chân ALE (cho phép chốt địa chỉ Address Latch Enable). Nó cần xung thấp lên cao để chốt địa chỉ.
3. Kích hoạt chân SCbằng xung cao xuống thấp để bắt đầu chuyển đổi.
4. Hiển thị OEC để báo kết thúc chuyển đổi. Đầu ra cao - xuống - thấp báo rằng dữ liệu đã được chuyển đổi và cần phải được lấy đi.
5. Kích hoạt OE cho phép đọc dữ liệu ra của ADC. Một xung cao xuống thấp tới chân OE sẽ đem dữ liệu số ra khỏi chíp ADC.

Lưu ý rằng trong ADC 808/809 không có đồng hồ riêng và do vậy phải cấp xung đồng bộ ngoài đến chân CLK. Mặc dù tốc độ chuyển đổi phụ thuộc vào tần số đồng hồ được nối đến CLK nhưng nó không nhanh hơn 100ms.

CHƯƠNG 13

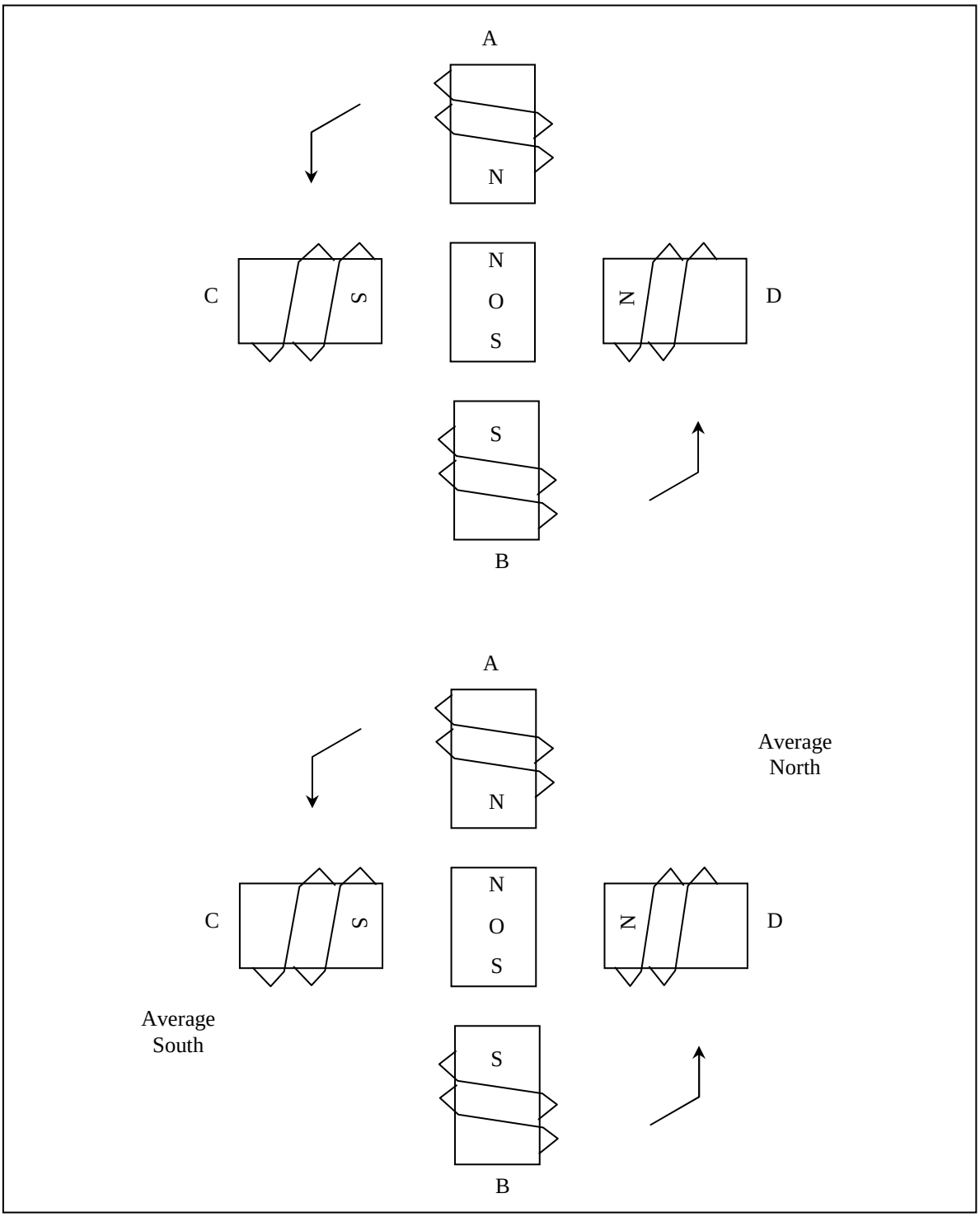
Phối ghép với thế giới kiểu II động cơ bước, bàn phím và các bộ DAC

13.1 Phối ghép với một động cơ bước.

Phần này bắt đầu với việc giới thiệu tổng quan về hoạt động của các động cơ bước. Sau đó chúng ta mô tả cách phối ghép một động cơ bước với bộ vi điều khiển 8051. Cuối cùng ta sử dụng các chương trình hợp ngữ để trình diễn điều khiển góc và hướng quay của động cơ bước.

13.1.1 Các động cơ bước.

Động cơ bước là một thiết bị sử dụng rộng rãi để chuyển các xung điện thành chuyển động cơ học. Trong các ứng dụng chẳng hạnnhư các bộ điều khiển đĩa, các máy in kim ma trận và các máy rô-bốt thì động cơ bước được dùng để điều khiển chuyển động. Mỗi động cơ bước đều có phần quay rôto là nam châm vĩnh cửu (cũng còn được gọi là trục dẫn - shaft) được bao bọc xung quanh là một **đứng** yên gọi stato (xem hình 131.1). Hầu hết các động cơ bước đều có chung có 4 stato mà các cuộn dây của chúng được bố trí theo cặp đối xứng với điểm giữa chung (xem hình 13.2), Kiểu động cơ bước này nhìn chung còn được coi như động cơ bước 4 pha. Điểm giữa cho phép một sự thay đổi của hướng dòng của một trong hai lõi khi một cuộn dây được nối đất tạo ra sự thay đổi cực của stato. Lưu ý rằng, trục của một động cơ truyền thống thì quay tự do, còn trục của động cơ bước thì chuyển động theo một độ tăng cố định lặp lại để cho phép ta chuyển dịch nó đến một vị trí chính xác. Chuyển động cố định lặp lại này có được là nhờ kết quả của lý thuyến từ trường cơ sở là các cực cùng dấu thì đẩy nhau và các cực khác dấu thì hút nhau. Hướng quay được xác định bởi từ trường của stato. Từ trường của stato được xác định bởi dòng chạyt quan lõi cuộn dây. khi hướng của dòng thay đổi thì cực từ trường cũng thay đổi gây ra chuyển động ngược lại của động cơ (đảo chiều). Động cơ bước được nối ở đây có 6 đầu dây: 4 đầu của cuộn dây stato và hai đầu dây chung điểm giữa của các cặp dây. Khi chuỗi xung nguồn được cấp đến mỗi cuộn dây stato thì động cơ sẽ quay. Có một số chuỗi xung được sử dụng rộng rãi với cấp độ chính xác khác nhau. Bảng 13.1 trình bày chuỗi 4 bước thông thường.



Hình 13.1: Căn chỉnh rôto.
Bảng 13.1: Chuỗi nguồn nuôi 4 bước thông thường.

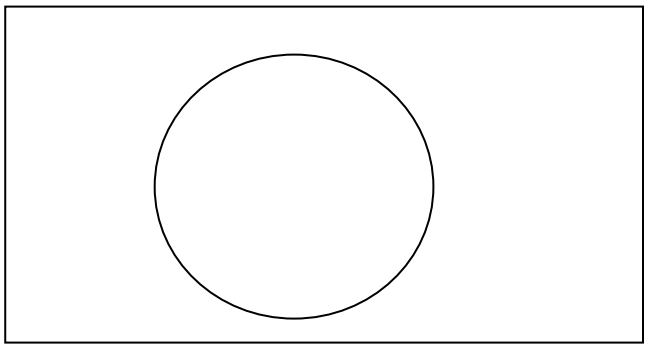
Chiều kim đồng hồ

Bước	Cuộn dây A	Cuộn dây B	Cuộn dây C	Cuộn dây D
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

Chiều quay bộ đếm

Bảng13.2: Các góc bước của động cơ bước.

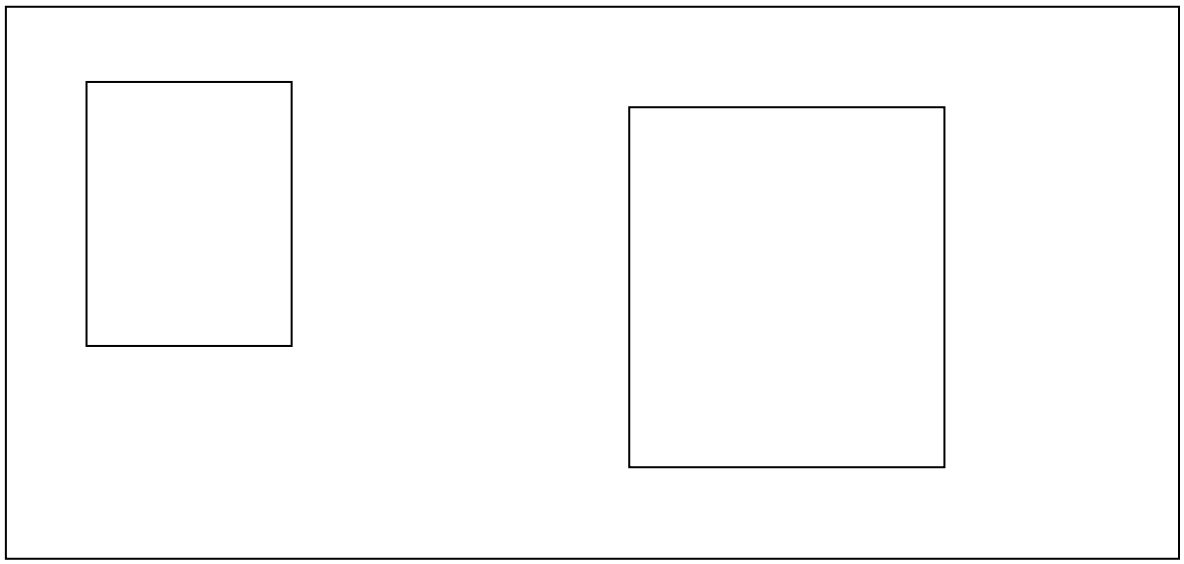
Góc bước	Số bước/ vòng
0.72	500
1.8	200
2.0	180
2.5	144
5.0	72
7.5	48
15	24



Hình 13.2:

Bảng13.2: Các góc bước của động cơ bước.

Hình 13.2: Bố trí các cuộn dây của stato.



Hình 13.3: Phối ghép 8051 với một động cơ bước.

Cần phải nhớ rằng mặc dù ta có thể bắt đầu với các chuỗi bất kỳ trong bảng 13.1. Nhưng khi đã bắt đầu thì ta phải tiếp tục với các chuỗi theo đúng thứ tự. Ví dụ ta bắt đầu bước thứ ba là chuỗi (0110) thì ta phải tiếp tục với chuỗi của bước 4 rồi sau đó 1, 2, 3 v.v...

Ví dụ 13.1:

Hãy mô tả kết nối 8051 với động cơ bước của hình 13.3 và viết một chương trình quay nó liên tục.

Lời giải:

Các bước dưới dây trình bày việc kết nối 8051 với động cơ bước và lập trình của nó.

1. Sử dụng một ôm kế để đo trở kháng của các đầu dây. Điều này xác định đầu chung (COM) nào được nối tới cuộn dây nào?
2. Các dây chung được nối tới đầu dương của nguồn cấp cho động cơ. Trong nhiều động cơ thì + 5V là đủ.
3. Bốn đầu củ cuộn dây stato được điểu khiển bởi 4 bít của cổng P1 trong 8051 (P1.0 - P1.3). Tuy nhiên, vì 8051 không đủ dòng để điều khiển các cuộn dây động cơ bước nên ta phải sử dụng một bộ điều khiển chẳng hạn như ULN2003 để cấp năng lượng cho stato. Thay cho ULN2003 ta có thể sử dụng các bóng bán dẫn làm các bộ điều khiển như chỉ ra trên hình 13.4. Tuy nhiên ta để ý rằng, nếu các bóng bán dẫn được sử dụng như các bộ điều khiển chúng ta cũng phải sử dụng các đi ôt để ngăn dòng cảm ứng được tạo ra khi tắt cuộn dây. Một lý do mà ULN2003 được ưu chuộng hơn các bóng bán dẫn như các bộ điều khiển là nó có đi ốt bên trong để ngăn cảm ứng điện từ ngược.

```
        MOV     A, # 66H        ; Nạp chuỗi xung bước
BACK:   MOV     P1, A           ; Xuất chuỗi xung đến động cơ
        RR      A               ; Quay theo chiều kim đồng hồ
        ACALL   DELAY           ; Chờ
        SJMP    BACK            ; Tiếp tục chạy
        --------
DELAY
        MOV     R2, # 100
H1:     MOV     R3, # 255
H2:     DJNZ    R3, H2
        DJNZ    R2, H1
        RET
```

Hãy thay đổi giá trị của DELAY để đặt tốc độ quay. Ta có thể sử dụng lệnh đơn bít SETB và CLR thay cho lệnh RRA để tạo ra chuỗi xung.

13.1.2 Góc bước (Step Angle).

Vậy mỗi bước có độ dịch chuyển là bao nhiêu? Điều này phụ thuộc vào cấu trúc bên trong của động cơ, đặc biệt là số răng của stato và rô to. Góc bước là độ quay nhỏ nhất của một bước. Các động cơ khác nhau có các góc bước khác nhau. Bảng 13.2 trình bày một số góc bước đối với các động cơ khác nhau. Bảng 13.2 có sử dụng thuật ngữ số bước trong một vòng (Steps per revolution). Đây là tổng số bước cần để quay hết một vòng 360^0 (chẳng hạn 180 bước $\times$ 2^0 = 360^0).

Cần phải nói rằng dường như trái ngược với ấn tượng ban đầu. Một động cơ bước không cần nhiều đầu dây cho stato hơn để có các bước nhỏ hơn. Tất cả mọi động cơ bước được nói ở đây chỉ có 4 đầu dây cho cuộn dây stato và 2 đầu dây chung cho nút giữa. Mặc dù nhiều hãng sản xuất chỉ dành một đầu chung thay cho hai thì họ cũng vẫn phải có 4 đầu cuộn dây stato.

13.1.3 Quan hệ số bước trong giây và số vòng quay trong phút RPM.

Quan hệ giữa số vòng quay trong phút RPM (revolutions per minute), số bước trong vòng quay và số bước trong vòng giây là quan hệ thuộc về trực giác và nó được biểu diễn như sau:

$$\text{So buoc trong giay} = \frac{\text{RPM} \times \text{So buoc trong vong quay}}{60}$$

13.1.4 Chuỗi xung bốn bước và số răng trên rô to.

Chuỗi xung chuyển mạch được trình bày trong bảng 13.1 được gọi là chuỗi chuyển mạch 4 bước bởi vì sau 4 bước thì hai cuộn dây giống nhau sẽ được bật "ON". Vậy độ dịch chuyển của 4 bước này sẽ là bao nhiêu? Sau mỗi khi thực hiện 4 bước này thì rô to chỉ dịch được một bước răng. Do vậy, trong động cơ bước với 200 bước/ vòng thì rô to của nó có 50 răng vì $50 \times 4 = 200$ bước cần để quay hết một vòng. Điều này dẫn đến một kết luận là góc bước tối thiểu luôn là hàm của số răng trên rô to. Hay nói cách khác góc bước càng nhỏ thì rô to quay được càng nhiều răng. Hãy xét ví dụ 13.2.

Ví dụ 13.2:

Hãy tính số lần của chuỗi 4 bước trong bảng 13.1 phải cấp cho một động cơ bước để tạo ra một dịch chuyển 80^0 nếu động cơ góc bước là 2^0.

Lời giải:

Một động cơ có góc bước là 2^0 thì phải có những đặc tính sau: góc bước 2^0, số bước/ vòng là 180^0, số răng của rô to là 45, độ dịch chuyển sau mỗi chuỗi 4 bước là 8^0. Vậy để dịch chuyển 80^0 thì cần 40 chuỗi 4 bước vì $10 \times 4 \times 2 = 80$.

Nhìn vào ví dụ 13.2 thì có người sẽ hỏi vậy muốn dịch chuyển đi 45^0 thì làm thế nào khi góc bước là 2^0. Muốn có độ phân giải nhỏ hơn thì tất cả mọi động cơ bước đều cho phép chuỗi chuyển mạch 8 bước, chuỗi 8 bước cũng còn được gọi chuỗi nửa bước (half - stepping), vì trong chuỗi 8 bước dưới đây thì mỗi bước là một nửa của góc bước bình thường. Ví dụ, một động cơ có góc bước là 2^0 có thể sử dụng góc bước 1^0 nếu áp dụng chuỗi ở bảng 13.3.

Bảng 13.3: Chuỗi xung 8 bước.

Chiều kim đồng hồ ↓

Bước	Cuộn A	Cuộn B	Cuộn C	Cuộn D
1	1	0	0	1
2	1	0	0	0
3	1	1	0	0
4	0	1	0	0
5	0	1	1	0
6	0	0	1	0
7	0	0	1	1
8	0	0	0	1

↑ Chiều quay bộ đếm

13.1.5 Tốc độ động cơ.

Tốc độ động cơ được đo bằng số bước trong một giây (bước/giây) là một hàm của tốc độ chuyển mạch. Để ý trong ví dụ 13.1 ta thấy rằng bằng việc thay đổi độ thời gian trễ ta có thể đạt được các tốc độ quay khác nhau.

13.1.6 Mô mem giữ.

Dưới đây là một định nghĩa về mô men giữ:

Mô men giữ là lượng mô men ngoài cần thiết để làm quay trục động cơ từ vị trí giữ của nó với điều kiện trục động cơ đang đứng yên hoặc đang quay với tốc độ RPM = 0. Đại lượng này được đo bằng tỷ lệ điện áp và dòng cấp đến động cơ. Đơn vị của mô men giữ là kilôgam - centimet (hay ounce - inch).

13.1.7 Chuỗi 4 bước điều khiển dạng sóng.

Ngoài các chuỗi 4 bước và 8 bước đã nó trên đây còn có một chuỗi khác được gọi là chuỗi 4 bước dạng sóng. Nó được trình bày trong bảng 13.4. Để ý 8 bước trong bảng 13.3 là một sự kết hợp đơn giản của các chuỗi 4 bước thường và chuỗi 4 bước điều khiển dạng sóng được cho ở bảng 13.1 và 13.4.

Chiều kim đồng hồ ↓

Bước	Cuộn dây A	Cuộn dây B	Cuộn dây C	Cuộn dây D
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

↑ Chiều quay bộ đếm

Hình 13.4: Sử dụng các bóng bán dẫn để điều khiển động cơ bước.

13.2 Phối ghép 8051 với bàn phím.

Các bàn phím và LCD là những thiết bị vào/ ra được sử dụng rộng rãi nhất của 8051 và cần phải thấu hiểu một cách cơ bản về chúng. Ở phần này trước hết ta giới thiệu các kiến thức cơ bản về bàn phím với cơ cấu ấn phím và tách phím, sau đó giới thiệu về giao tiếp 8051 với bàn phìm.

13.2.1 Phối ghép bàn phím với 8051.

Ở mức thấp nhất các bàn phím được tổ chức dưới dạng một ma trận các hàng và các cột. CPU truy cập cả hàng lẫn cột thông qua các cổng. Do vậy, với hai cổng 8 bít thì có thể nối tới một bàn phím 8×8 tới bộ vi xử lý. Khi một phím được ấn thì một hàng và một cột được tiếp xúc, ngoài ra không có sự tiếp xúc nào giữa các hàng và các cột. Trong các bàn phím máy tính IBM PC có một bộ vi điều khiển (bao gồm một bộ vi xử lý, bộ nhớ RAM và EPROM và một số cổng tất cả được bố trí trên một chíp) chịu trách nhiệm phối ghép phần cứng và phần mềm của bàn phím. Trong những hệ thống như vậy, nó là chức năng của các chương trình được lưu trong EPROM của bộ vi điều khiển để quét liên tục các phím, xác định xem phím nào đã được kích hoạt và gửi nó đến bo mạch chính. Trong phần này nghiên cứu về cơ cấu 8051 quét và xác định phím.

13.2.2 Quét và xác định phím.

Hình 13.5 trình bày một ma trận 4×4 được nối tới hai cổng. Các hàng được nối tới một đầu ra và các cột được nối tới một cổng vào. Nếu không có phím nào được ấn thì việc đóng cổng vào sẽ hoàn toàn là 1 cho tất cả các cột vì tất cả được nối tới dương nguồn V_{CC}. Nếu tất cả các hàng được nối đất và một phím được ấn thì một trong các cột sẽ có giá trị 0 vì phím được ấn tạo đường xuống đất. Chức năng của bộ vi điều khiển là quét liên tục để phát hiện và xác định phím được ấn.

Hình 13.5

Hình 13.5: Nối ghép bàn phím ma trận tới các cổng.

13.2.3 Nối đất các hàng và đọc các cột.

Để phát hiện một phím được ấn thì bộ vi điều khiển nối đất tất cả các hàng bằng cách cấp 0 tơío chốt đầu ra, sau đó nó đọc các hàng. Nếu dữ được đọc từ các cột là D3 - D0 = 1101 thì không có phím nào được ấn và quá trình tiếp tục cho đến khi phát hiện một phím được ấn. Tuy nhiên, nếu một trong các bít cột có số 0 thì điều đó có nghĩa là việc ấn phím đã xảy ra. Ví dụ, nếu D3 - D0 = 1101 có nghĩa là một phím ở cột 1 được ấn. Sau khi một ấn phím được phát hiện, bộ vi điều khiển sẽ chạy quá trình xác định phím. Bắt đầu với hàng trên cùng, bộ vi điều khiển nối đất nó bằng cách chỉ cấp mức thấp tới chân D0, sau đó nó đọc các cột. Nếu dữ liệu đọc được là toàn số 1 thì không có phím nào của hàng này được ấn và quá trình này chuyển sang hàng kế tiếp. Nó nối đất hàng kế tiếp, đọc các cột và kiểm tra xem có số 0 nào không? Qúa trình này tiếp tục cho đến khi xác định được hàng nào có phím ấn. Sau khi xác định được hàng có phím được ấn thì công việc tiếp theo là tìm ra phím ấn thuộc cột nào. Điều này thật là dễ dàng vì bộ vi điều khiển biết tại thời điểm bất kỳ hàng nào và cột nào được truy cập. Hãy xét ví dụ 13.3.

Ví dụ 13.3:

Từ hình 13.5 hãy xác định hàng và cột của phím được ấn cho các trường hợp sau đây:

a) D3 - D0 = 1110 cho hàng và D3 - D0 = 1011 cho cột.
b) D3 - D0 = 1101 cho hàng và D3 - D0 = 0111 cho cột.

Lời giải:

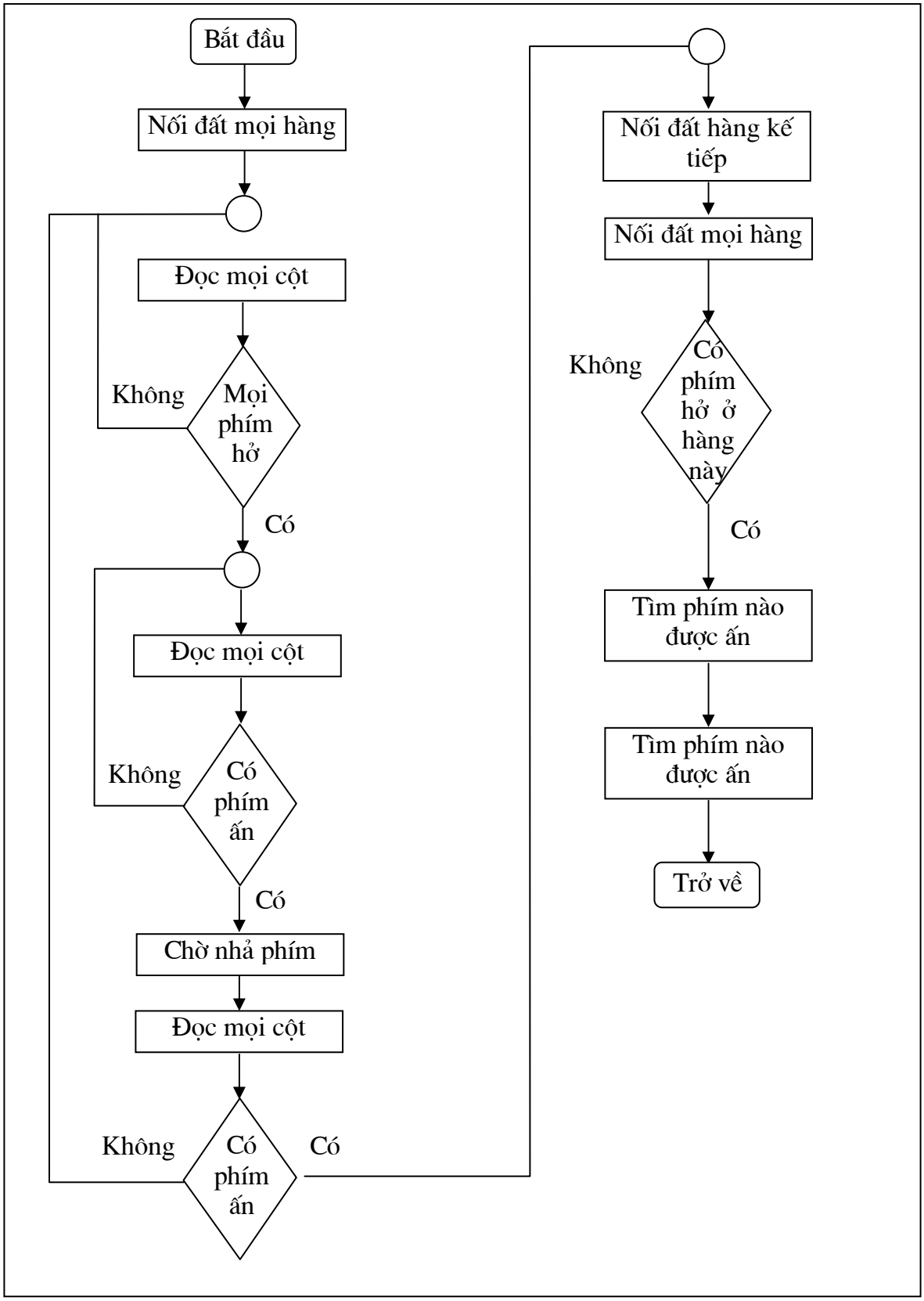
Từ hnhf 13.5 cột và hàng có thể được sử dụng xác định phím.

a) Hàng thuộc D0 và cột thuộc D2, do vậy phím số 2 đã được ấn.
b) Hàng thuộc D1 và cột thuộc D3, do vậy phím số 7 đã được ấn.

Chương trình 13.1 là chương trình hợp ngữ của 8051 để phát hiện và xác định sự kích hoạt phím. Trong chương trình này P1 và P2 được giả thiết là cổng ra và cổng vào tương ứng. Chương trình 13.1 đi qua 4 giai đoạn chính sau đây.

1. Khẳng định phím trước đó đã được nhả, các số không là đầu ra tới tất cả các hàng cùng một lúc và các cột được đọc và được kiểm tra chừng nào tất cả mọi cột đều cao. Khi tất cả các cột được phát hiện là đều cao thì chương trình chờ một thời gian ngắn trước khi nó chuyển sang giai đoạn kế tiếp để chờ một phím được ấn.
2. Để biết có một phím nào được ấn các cột được quét đi quét lại trong vòng vô tận cho đến khi có một cột có số 0. Hãy nhớ rằng các chốt đầu ra được nối tới các hàng vẫn có các số 0 ban đầu (được cấp ở giai đoạn 1) làm cho chúng được nối đất. Sau khi phát hiện ấn phím, nó đợi 20ms chờ cho phím nhả ra và sau đó quét lại các cột. Điều này phục vụ hai chức năng: a) nó đảm bảo rằng việc phát hiện ấn phím đầu tiên không bị sai do nhiễu và b) thời gian giữ chậm là 20ms ngăn ngừa việc ấn cùng một phím như là nhiều lần ấn. Nếu sau 20ms giữ chậm mà phím vẫn được ấn nó chuyển sang giai đoạn kế tiếp để phát hiện phím ấn thuộc hàng nào, nếu không nó quay trờ vòng lặp để phát hiện có một phím ấn thật.
3. Để phát hiện ấn phím thuộc hàng, nó nối đất mỗi hàng tại một thời điểm, đọc các cột mỗi lần. Nếu nó phát hiện tất cả mọi cột đều cao, điều này có nghĩa là ấn phím không thuộc hàng đó, do vậy nó nối đất hàng kế tiếp và tiếp tục cho đến khi phát hiện ra hàng có phím ấn. Khi tìm hàng có phím ấn, nó thiết lập địa chỉ bắt đầu cho bảng trình bày giữ các mã quét (hoặc giá trị ASCII) cho hàng đó và chuyển sang giai đoạn kế tiếp để xác định phím.

4. Để xác định phím ấn, nó quay các bít cột, mỗi lần một bít vào cờ nhớ và kiểm tra xem nó có giá trị thấp không? Khi tìm ra số 0, nó kéo mã ASCII dành cho phím đó ra từ bảng trình bày. Nếu không tìm được số 0 thì nó tăng con trỏ để chỉ đến phần tử kế tiếp của bảng trình bày. Hình 13.6 trình bày lưu đồ quá trình tìm phím ấn này.



Hình 13.6: Lưu đồ tìm phím ấn của chương trình 13.1.

Trong khi việc phát hiện ấn là chuẩn cho tất cả mọi bàn phím thì quá trình xác định phím nào được ấn lại không giống nhau. Phương phát sử dụng bảng trình bày được đưa ra trong chương trình 13.1 có thể được sửa đổi để làm việc với bất kỳ ma trận kích thước 8×8 nào. Hình 13.6 là lưu đồ thuật toán của chương trình 13.1 để quét và xác định phím ấn.

Có những chíp IC chẳng hạn như MM74C924 của hãng National Semiconductor kết hợp việc quét và giải mã bàn phím tất cả vào một chíp. Các chíp như vậy sử sự kết hợp các bộ đếm và các cổng lô gíc (không phải bộ vi điều khiển) để thực thi các khái niệm được trình bày trong chương trình 13.1 dưới đây.

Chương trình 13.1:

```
; Chương trình con bàn phím Keyboard này gửi mã ASCII
; Cha phím được ấn đến chân P0.1
; Các chân P1.0 – P1.3 được nối tới các hàng còn P2.0 – P2.3 tới các cột.
```

13.3 Phối ghép một DAC với 8051.

Phần này sẽ trình bày cách phối ghép một bộ biến đổi số tương tự DAC với 8051. Sau đó minh hoạ tạo một sóng hình sin trên máy hiện sóng sử dụng bọ DAC.

13.3.1 Bộ biến đổi số - tương tự DAC.

Bộ biến đổi - tương tự DAC là mmột thiết bị được sử dụng rộng rãi để chuyển đổi các xung số hoá về các tín hiệu tương tự. Trong phần này ta giới thiệu cơ sở phối ghép một bộ DAC với 8051.

Xem lại các kiến thức điện tử số ta thấy có hai cách tạo ra bộ DAC: Phương phát trọng số nhị phân và phương trình thang R/2R. Nhiều bộ DAC dựa trên các mạch tổ hợp, bao gồm MC1408 (DAC808) được sử dụng trong phần này đều sử dụng phương pháp hình thang R/2R vì nó có thể đạt độ chính xác cao hơn. Tiêu chuẩn đánh giá một bộ DAC đầu tiên là độ phân giải hàm của số đầu vào nhị phân. Các độ phân giải chúng là 8, 10 và 12 bít. Số các đầu vào bít dữ liệu quyết định độ phân giải của bộ DAC, vì số mức đầu ra tương tự bằng 2^n với n là đầu vào bít dữ liệu. Do vậy, một bộ DAC 8 bít như DAC808 chẳng hạn có 256 mức đầu ra điện áp (dòng điện) rời rạc. Tương tự như vậy, một bộ DAC 12 bít cho 4096 mức điện áp rời rạc. cũng có các bộ DAC 16 bít nhưng chúng rất đắt.

13.3.2 Bộ biến đổi DAC MC1408 (hay DAC808).

Trong bộ ADC808 các đầu vào số được chuyển đổi thành dòng (I_{out}) và việc nối một điện trở tới chân I_{out} ta chuyển kết quả thành điện áp. dòng tổng được cấp bởi chân I_{out} là một hàm số nhị phân ở các đầu vào D0 – D7 của DAC808 và tham chiếu I_{ref} như sau:

$$I_{out} = I_{ref}\left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256}\right)$$

Trong đó D0 là bít thấp nhất LSB và D7 là bít cao nhất MSB đối với các đầu vào

Chương 14
Phối phép 8031/51 với bộ nhớ ngoài

14.1 Bộ nhớ bán dẫn.

Trong phần này ta nhớ về các kiểu loại bộ nhớ bán dẫn khác nhau và các đặc tính của chúng như dụng lượng, tổ chức và thời gian truy cập. Trong thiết kế của tất cả các hệ thống dựa trên bộ vi sử lý thì các bộ nhớ bán dẫn được dùng như hơi lưu giữ chương trình và dữ liệu chính. Các bộ nhớ bán dẫn được nối trực tiếp với CPU và chúng là bộ nhớ mà CPU đầu tiên hỏi về thông tin chương trình và dữ liệu. Vì lý do này mà các bộ nhớ nhiều khi được coi như là nó phải đáp ứng nhanh cho CPU mà các điều này chỉ có các bộ nhớ bán dẫn mới có thể làm được. Các bộ nhớ bán dẫn được sử dụng rộng rãi nhất là ROM và RAM. Trước khi đi vào bàn các kiểu bộ nhớ ROM và RAM chúng ta làm quen với một số thuật ngữ quan trọng chung cho tất cả mọi bộ nhớ bán dẫn như là dung lượng, tổ chức và tốc độ.

14.1.1 Dung lượng nhớ.

Số lượng các bít mà một chíp nhớ bán dẫn có thể lưu được gọi là dung lượng của chíp, nó có đơn vị có thể là ki-lô-bít (Kbít), mê-ga-bit (Mbít) v.v… Điều này phải phân biết với dung lượng lưu trữ của hệ thống máy tính. Trong khi dung lượng nhớ của một IC nhớ luôn được tính theo bít, còn dung lượng nhớ của một hệ thống máy tính luôn được cho tính byte. Chẳng hạn, trên tạp chí kỹ thuật có một bài báo nói rằng chíp 16M đã trở nên phổ dụng thì mặc dù nó không nói rằng 16M nghĩa là 16 mê-ga-bit thì nó vẫn được hiểu là bài báo nói về chíp IC nhớ. Tuy nhiên, nếu một quảng cáo nói rằng một máy tính với bộ nhớ 16M vì nó đang nói về hệ thống máy tính nên nó được hiểu 16M có nghĩa là 16 mê-ga-byte.

14.1.2 Tổ chức bộ nhớ.

Các chíp được tổ chức vào một số ngăn nhớ bên trong mạch tích hợp IC. Mỗi ngăn nhớ có chứa bộ bít, 4 bít, 8 bít thậm chí đến 16 bít phụ thuộc vào cách nó được thiết kế bên trong như thế nào? Số bít mà mỗi ngăn nhớ bên trong chíp nhớ có thể chứa được luôn bằng số chân dữ liệu trên chíp. Vậy có bao nhiêu ngăn nhớ bên trong một chíp nhớ? Nó phụ thuộc vào số chân địa chỉ, số ngăn nhớ bên trong một IC nhớ luôn bằng 2 luỹ thừa với số chân địa chỉ. Do vậy, tổng số bít mà IC nhớ có thể lưu trữ là bằng số ngăn nhớ nhân với bít dữ liệu trên mỗi ngăn nhớ. Có thể tóm tắt lại như sau:

1. Một chíp nhớ có thể chứa 2^x ngăn nhớ, với x là số chân địa chỉ.
2. Mỗi ngăn nhớ chứa y bít với y là số chân dữ liệu trên chíp.
3. Toàn bộ chíp chứa ($2^x \times y$)bít với x là số chân địa chỉ và y là số chân dữ liệu trên chíp.

14.1.3 Tốc độ.

Một trong những đặc tính quan trọng nhất của chíp nhớ là tốc độ truy cập dữ liệu của nó. Để truy cập dữ liệu thì địa chỉ phải có ở các chân địa chỉa, chân đọc READ được tích cực và sau một khoảng thời gian thì dữ liệu sẽ xuất hiện ở các chân dữ liệu. Khoảng thời gian này càng ngắn càng tốt và tất nhiên là chíp nhớ càng đắt. Tốc độ của chíp nhớ thường được coi như là thời gian truy cập của nó. Thời gian truy cập của các chíp nhớ thay đổi từ vài na-nô-giây đến hàng trăm na-nô-giây phụ thuộc vào công nghệ sử dụng trong quá trình thiết kế và sản xuất IC.

Cả ba đặc tính quan trọng của bộ nhớ là dung lượng nhớ, tổ chức bộ nhớ và thời gian truy cập sẽ được sử dụng nhiều trong chương trình. Bảng 14.1 như một

tham chiếu để tính toán các đặc tính của bộ nhớ. Các ví dụ 14.1 và 14.2 sẽ minh hoạ những khái niệm vừa trình bày.

Bảng 14.1: Dung lượng bộ nhớ với số chân địac chỉ của IC.

x	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
$\mathbf{2^x}$	1K	2K	4K	8K	16 K	32 K	64 K	12 8K	25 6K	51 2K	1 M	2 M	4 M	8 M	16 M

Ví dụ 14.1:

Một chíp nhớ có 12 chân địa chỉ và 4 chân dữ liệu. Hãy tìm tổ chức bộ nhớ và dung lượng nhớ của nó.

Lời giải:

a) Chíp nhớ này có 4096 ngăn nhớ ($2^{12} = 4096$) và mỗi ngăn nhớ chứa 4 bít dữ liệu nên tổ chức nhớ của nó là 4096×4 và thường được biểu diễn là $4K \times 4$.

b) Dung lượng nhớ của chíp nhớ là 16K vì có 4 K ngăn nhớ và mỗi ngăn nhớ có 4 bít dữ liệu.

Ví dụ 14.2:

Một chíp nhớ 512k có 8 chân dữ liệu. Hãy tìm a) tổ chức của nó và b) số chân địa chỉ của chíp này.

Lời giải:

a) Một chíp có 8 chân dữ liệu có nghĩa là mỗi ngăn nhớ có 8 bít dữ liệu. Số ngăn nhớ trên chíp này bằng dung lượng nhớ chia cho số chân dữ liệu = 512k/8 = 64. Do vậy tổ chức nhớ của chíp là $64k \times 8$.

b) Số đường địa chỉ của chíp sẽ là 16 vì $1^{16} = 64k$.

14.1.4 Bộ nhớ ROM.

Bộ nhớ ROM là bộ nhớ chỉ đọc (Read - only Memory). Đây là một kiểu bộ nhớ mà không mất các nội dung của nó khi tắt nguồn. Với lý do này mà bộ nhớ ROM còn được gọi là bộ nhớ không bay hơi, có nhiều loại bộ nhớ ROM khác nhau như: PROM, EPROM, EEPROM, EPROM nhanh (flash) và ROM che.

14.1.4.1 Bộ nhớ PROM.

Bộ nhớ PROM là bộ nhớ ROM có thể lập trình được. Đây là loại bộ nhớ mà người dùng có thể đốt ghi thông tin vào. hay nói cách khác, PROM là bộ nhớ người dùng có thể lập trình được. Đối với mỗi bít của PROM có một cầu chì. Bộ nhớ PROM được lập trình bằng cách làm đứt những cầu trì. Nếu thông tin được đốt vào trong PROM mà sau thì phải bỏ vì các cầu trì của nó bên trong bị đứt vĩnh viễn với lý do này mà PROM mà được gọi là bộ nhớ ROM lập trình một lần. Việc lập trình ROM cũng được gọi là đốt ROM và nó đòi hỏi phải có một thiết bị đặc biệt gọi là bộ đốt ROM hay còn gọi là thiết bị lập trình ROM.

14.1.4.2 Bộ nhớ EPROM và UV - EPROM.

Bộ nhớ EPROM được phát minh ra để cho phép thực hiện thay đổi nội dung của PROM sau khi nó đã được đốt. Trong bộ nhớ EPROM ta có thể lập trình chíp nhớ và xoá nó hàng nghìn lần. Điều này là đặc biệt cần thiết trong quá trình phát triển mẫu thử của một dự án dựa trên bộ vi xử lý. Một EPR sử dụng rộng rãi được gọi là UV - EPROM (UV là chữ viết tắt của tia cực tím Ultra - Violet). Vấn đề tồn tại duy nhất của UV - EPROM là thời gian xoá của nó quá lâu (20 phút).

Tất cả các chíp nhớ UV - EPROM có một cửa sổ được dùng để chiếu tia bức xạ cực tím xoá các nội dung của nó. Với lý do này mà EPROM cũng còn được coi như là bộ nhớ EPROM được xoá bằng tia cực tím hay đơn giản được gọi là UV – EPROM. Hình 14.1 trình bày các chân của một chíp UV - EPROM.

Để lập trình cho một UV - EPROM cần thực hiện các bước.

1. Xoá các nội dung của nó, để xoá một chíp thì phải tháo nó ra khỏi để cắm trên bảng mạch hệ thống và đặt nó vào thiết bị xoá EPROM để chiếu xạ tia cực tím khoảng 15 - 20 phút.
2. Lập trình cho chíp. Để lập trình cho một chíp UV - EPROM thì đặt nó vào thiết bị đốt (thiết bị lập trình). Để đốt chương trình và dữ liệu vào EPROM thì thiết bị đốt ROM sử dụng điện áp 12.5V hoặc cao hơn phụ thuộc vào loại EPROM. Điện áp này được gọi là Vpp trong bảng dữ liệu của UV - EPROM.
3. Lắp chíp nhớ trở lại để cắm trên hệ thống.

Từ các bước trên đây ta thấy cũng như các thiết bị đốt EPROM thì cũng có các thiết bị xoá EPROM khác nhau. Và tất cả các kiểu bộ nhớ UV – EPROM đều có một nhược điểm chính là không thể được lập trình trực tiếp trên bảng mạch của hệ thống. Do vậy, bộ nhớ EPROM đã được ra đời để giải quyết vấn đề này.

Hãy để ý đến các ký hiệu mã số của các IC trong bảng 14.2. ví dụ, mã số bộ phận 27128 - 25 dành cho UV - EPROM có dung lượn và thời gian truy cập là 250ns. Dung lượng của chíp nhỏ được ký hiệu trên mã số bộ phận (part number) và thời gian truy cập được bỏ đi một số 0. Trong các mã số bộ phận thì chữ C là cộng nghệ CMOS, còn 27xx luôn chỉ các chíp nhớ EPROM. Để biết thêm chi tiết ta vào mục sổ tay các chíp nhớ của các hãng chẳng hạn JAMECO (jameco.com) hay JDR (jdr.com) theo đường mạng internet.

Bảng 14.2: Một số chíp nhớ UV - EPROM.

HìNH 14.1

Hình 14.1: Bố trí các chân của họ các chíp nhớ ROM 27xx.

Ví dụ 14.3:

Đối với ROM có mã bộ phận là 27128 có dung lượng nhớ là 128k bít. Tra bảng ta thấy tổ chức của nó là 16k × 8 (tất cả mọi ROM đều có 8 chân dữ liệu) điều này nói lên rằng nó có 8 chân dữ liệu và số chân địa chỉ được tìm thấy là 14 vì 2^{14} = 16k.

14.1.4.3 Bộ nhớ PROM có thể xoá được bằng điện EEPROM.

Bộ nhớ EEPROM có một số ưu điểm so với EPROM là do vì được xoá bằng điện nên quá trình xoá rất nhanh, nhanh rất nhiều so với thời gian xoá 20 phút của UEPROM. Ngoài ra trong EEPROM ta phải xoá toàn bộ nội dung của ROM. Tuy nhiên, ưu điểm chính của EEPROM là ta có thể lập trình và xoá khi chíp nhớ vẫn ở trên giá cắm của bảng mạch hệ thống mà không phải lấy ra như đối với UV – EPROM. Hay nói cách khác là EPROM không cần thiết bị lập trình và thiết bị xoá ngoài như đối với UV - EPROM. Để hoàn toàn tạo thuận lợi cho EPROM nhà thiết kế phải kết hợp vào bảng mạch của hệ thống cả mạch điện để lập trình EEPROM sử dụng điện áp Vpp = 5V nhưng chúng rất đắt. nhìn chung, giá thành cho 1 bít của EEPROM đắt hơn rất nhiều so với UV - EPROM.

Bảng 14.3: Một số chíp EEPROM và chíp nhớ Flash (nhanh).

CHƯƠNG 15
Phép ghép 8031/51 với 8255

Như đã nói ở chương 14 trong quá trình nối ghép 8031/51 với bộ nhớ ngoài thì hai cổng P0 và P2 bị mất. Trong chương này chúng ta sẽ trình bày làm thế nào để mở rộng các cổng vào/ ra I/O của 8031/51 bằng việc nối nó tới chíp 8255.

15.1 Lập trình 8255.

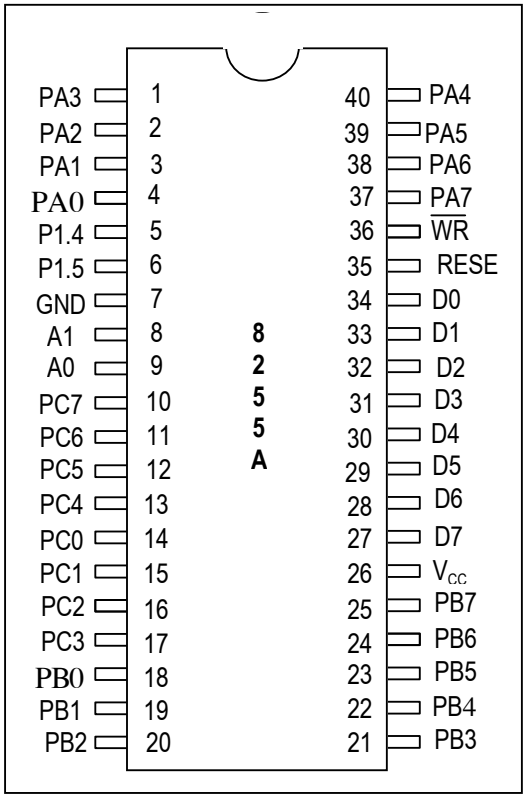
Trong mục này ta nghiên cứu 8255 như là một trong những chíp vào/ ra được sử dụng rộng rãi nhất. Trước hết ta mô tả những đặc tính của nó và sau đsó chỉ ra cách nối 8031/51 với 8255 như thế nào?

15.1 Lập trình 8255.

Trong mục này ta nghiên cứu 8255 như là một trong những chíp vào/ ra được sử dụng rộng rãi nhất. Trước hết ta mô tả những đặc tính của nó và sau đó chỉ ra cách nối 8031/51 với 8255 như thế nào?

15.1.1 Các đặc tính của 8255.

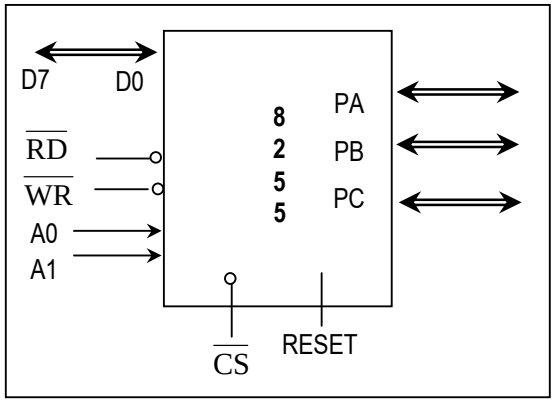
8255 là một chíp DIP 4 chân (xem hình 15.1). Nó có 3 cổng truy cập được riêng biệt. Các cổng đó có tên A, B và C đều là các cổng 8 bit. Các cổng này đều có thể lập trình như cổng đầu vào hoặc đầu ra riêng rẽ và có thể thay đổi một cách năng động. Ngoài ra, các cổng 8255 có khả năng bắt tay. Do vậy cho phép giao diện với các thiết bị khác cũng có giá trị tín hiệu bắt tay như các máy in chẳng hạn. Khả năng bắt tay của 8255 sẽ được bàn tới ở mục 15.3.



Hình 15.1: Chíp 8255.

15.1.1.1 Các chân PA0 - PA7 (cổng A).

Cả 8 bít của cổng A PA0 - PA7 có thể được lập trình như 8 bit đầu vào hoặc 8 bit đầu ra hoặc cả 8 bít hai chiều vào/ ra.S



Hình 15.2: Sơ đồ khối của 8255.

15.1.1.2 Các chân PB0 - PB7 (cổng B).

Cả 8 bit của cổng B có thể được lập trình hoặc như 8 bit đầu vào hoặc 8 bit đầu ra hoặc cả 8 bit hai chiều vào/ ra.

15.1.1.3 Các chân PC0 - PC7 (cổng C).

Tất cả 8 bit của cổng C (PC0 - PC7) đều có thể được lập trình như các bit đầu vào hoặc các bit đầu ra. 8 bit này cũng có thể được chia làm hai phần: Các bit cao (PC4 - PC7) là CU và các bit thấp (PC0 - PC3) là CL. Mỗi phần có thể được dùng hoặc làm đầu vào hoặc làm đầu ra. Ngoài ra từng bit của cổng C từ PC0 - PC7 cũng có thể được lập trình riêng rẽ.

15.1.1.4 Các chân $\overline{RD}$ và $\overline{WR}$.

Đây là hai tín hiệu điều khiển tích cực mức thấp tới 8255 được nối tới các chân dữ liệu $\overline{RD}$ và $\overline{WR}$ từ 8031/51 được nối tới các chân đầu vào này.

15.1.1.5 Các chân dữ liệu D0 - D7.

Các chân dữ liệu D0 - D7 của 8255 được nối tới các chân dữ liệu của bộ vi điều khiển để cho phép nó gửi dữ liệu qua lại giữa bộ vi điều khiển và chíp 8255.

15.1.1.6 Chân RESET.

Đây là đầu vào tín hiệu tích cực mức cao tới 8255 được dùng để xoá thanh ghi điều khiển. Khi chân RESET được kích hoạt thì tất cả các cổng được khởi tạo lại như các cổng vào. Trong nhiều thiết kế thì chân này được nối tới đầu ra RESET của bus hệ thống hoặc được nối tới đất để không kích hoạt nó. Cũng như tất cả các chân đầu vào của IC thì nó cũng có thể để hở.

15.1.1.7 Các chân A0, A1 và $\overline{CS}$.

Trong khi $\overline{CS}$ chọn toàn bộ chíp thì A0 và A1 lại chọn các cổng riêng biệt. Các chân này được dùng để truy cập các cổng A, B, C hoặc thanh ghi điều khiển theo bảng 15.1. Lưu ý $\overline{CS}$ là tích cực mức thấp.

15.1.2 Chọn chế độ của 8255.

Trong khi các cổng A, B và C được dùng để nhập và xuất dữ liệu thì thanh ghi điều khiển phải được lập trình để chọn chế độ làm việc của các cổng này. Các cổng của 8255 có thể được lập trình theo một chế độ bất kỳ dưới đây.

1. Chế độ 0 (Mode0): Đây là chế độ vào/ ra đơn giản. Ở chế độ này các cổng A, B CL và CU có thể được lập trình như đầu vào hoặc đầu ra. Trong chế độ này thì tất cả các bit hoặc là đầu vào hoặc là đầu ra. Hay nói cách khác là không có điều khiển theo từng bit riêng rẽ như ta đã thấy ở các cổng P0 - P3 của 8051. Vì đa phần các ứng dụng liên quan đến 8255 đều sử dụng chế độ vào/ ra đơn giản này nên ta sẽ tập chung đi sâu vào chế độ này.

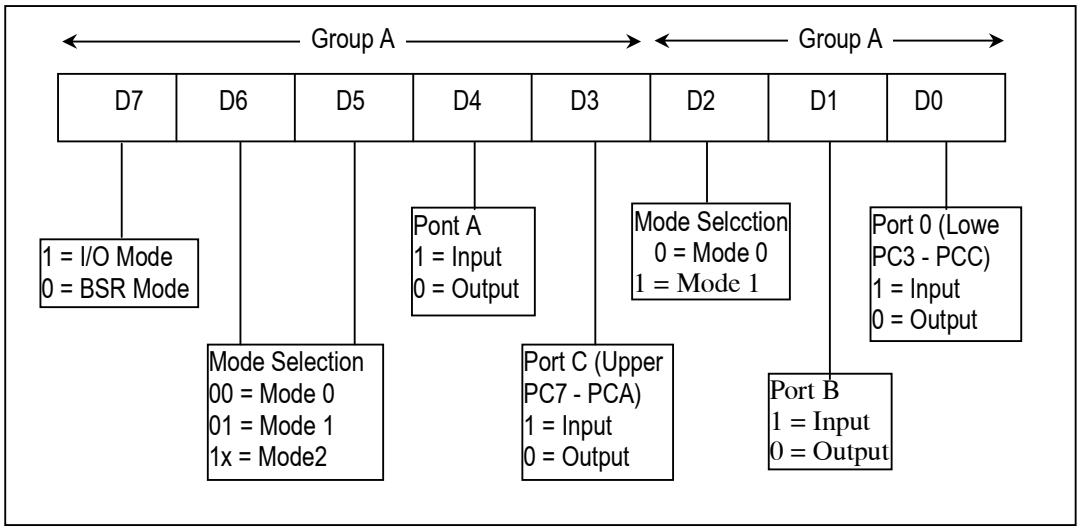
2. Chế độ 1 (Mode1): Trong chế độ này các cổng A và B có thể được dùng như các cổng đầu vào hoặc đầu ra với các khả năng bắt tay. Tín hiệu bắt tay được cấp bởi các bit của cổng C (sẽ được trình bày ở mục 15.3).

3. Chế độ 2 (Mode2): Trong chế độ này cổng A có thể được dùng như cổng vào/ ra hai chiều với khả năng bắt tay và các tín hiệu bắt tay được cấp bởi các bit cổng C. Cổng B có thể được dùng như ở chế độ vào/ ra đơn giản hoặc ở chế độ có bắt tay Mode1. Chế độ này sẽ không được trình bày trong tài liệu này.

Chế độ BSR: Đây là chế độ thiết lập/ xoá bit (Bit Set/ Reset). ở chế độ này chỉ có những bit riêng rẽ của cổng C có thể được lập trình (sẽ được trình bày ở mục 15.3).

Bảng 15.1: Chọn cổng của 8255.

$\overline{CS}$	A1	A0	Chọn cổng
0	0	0	Cổng A
0	0	1	Cổng B
0	1	0	Cổng C
0	1	1	Thanh ghi điều khiển
1	x	X	8255 không được chọn



Hình 15.3: Định dạng từ điều khiển của 8255 (chế độ vào/ ra).

15.1.3 Lập trình chế độ vào/ ra đơn giản.

Hãng Intel gọi chế độ 0 là chế độ vào/ ra cơ sở. Một thuật ngữ được dùng chung hơn là vào/ ra đơn giản. Trong chế độ này thì một cổng bất kỳ trong A, B, C được lập trình như là cổng đầu vào hoặc cổng đầu ra. Cần lưu ý rằng trong chế độ này một cổng đã cho không thể vừa làm đầu vào lại vừa làm đầu ra cùng một lúc.

Ví dụ 15.1:

Hãy tìm từ điều khiển của 8255 cho các cấu hình sau:

Tất cả các cổng A, B và C đều là các cổng đầu ra (chế độ 0).

PA là đầu vào, PB là đầu ra, PCL bằng đầu vào và PCH bằng đầu ra.

Lời giải:

Từ hình 15.3 ta tìm được:

a) 1000 0000 = 80H; b) 1001 000 = 90H

15.1.4 Nối ghép 8031/51 với 8255.

Chíp 8255 được lập trình một trong bốn chế độ vừa trình bày ở trên bằng cách gửi một byte (hãng Intel gọi là một từ điều khiển) tới thanh ghi điều khiển của 8255. Trước hết chúng ta phải tìm ra các địa chỉ cổng được gán cho mỗi cổng A, B, C và thanh ghi điều khiển. Đây được gọi là ánh xạ cổng vào/ ra (mapping).

Như có thể nhìn thấy từ hình 15.4 thì 8255 được nối tới một 8031/51 như thế nó là bộ nhớ RAM. Để việc sử dụng các tín hiệu $\overline{RD}$ và $\overline{WR}$. Phương pháp nối một chíp vào/ ra bộ nhớ vì nó được ánh xạ vào không gian bộ nhớ. Hay nói cách khác, ta sử dụng không gian bộ nhớ để truy cập các thiết bị vào/ ra. Vì ly??? do này mà ta dùng lệnh MOVX để truy cập RAM và ROM. Đối với một 8255

được nối tới 8031/51 thì ta cũng phải dùng lệnh MOVX để truyền thông với nó. Điều này được thể hiện trên ví dụ 15.2.

Ví dụ 15.2:

Đối với hình 15.4:

a) Hãy tìm các địa chỉ vào/ ra được gán cho cổng A, B, C và thanh ghi điều khiển.
b) Hãy lập trình 8255 cho các cổng A, B và C thành các cổng đầu ra.
c) Viết một chương trình để gửi 55H và AAH đến cổng liên tục.

Lời giải:

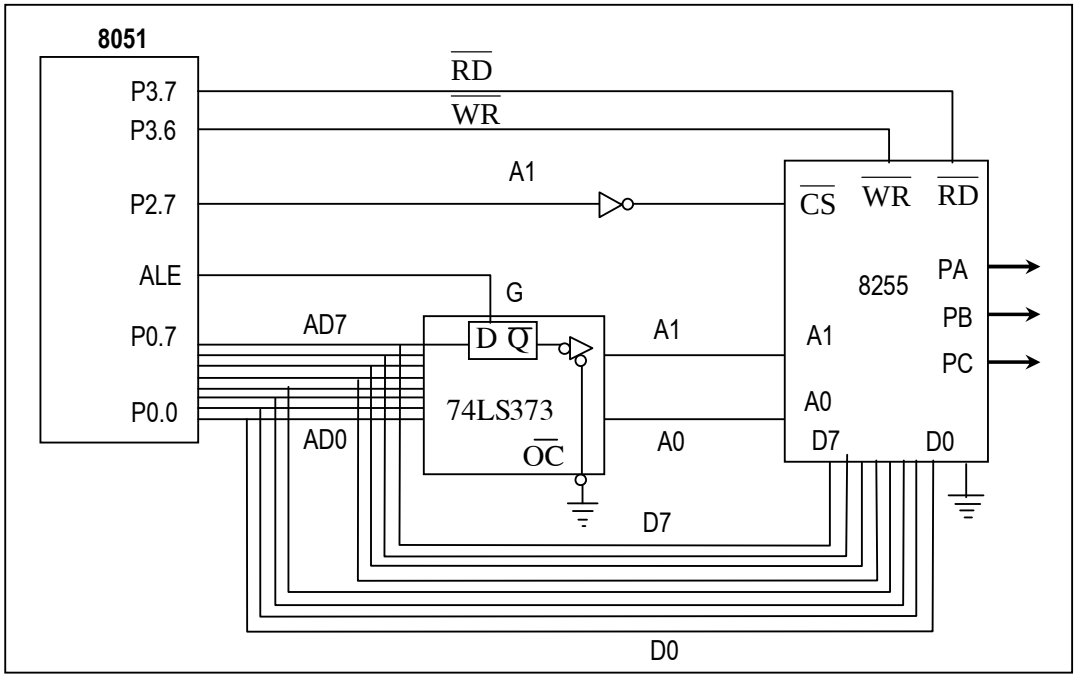
a) Địa chỉ cơ sở dành cho 8255 như sau:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
x	1	x	x	x	x	x	x	x	x	x	x	x	X	0	0	=4000HPA
x	1	x	x	x	x	x	x	x	x	x	x	x	X	0	1	=4000HPB
x	1	x	x	x	x	x	x	x	x	x	x	x	X	1	0	=4000HPC
x	1	x	x	x	x	x	x	x	x	x	x	x	X	1	1	=4000HCR

b) Byte (từ) điều khiển cho tất cả các cổng như đầu ra là 80H như được tính ở ví dụ 15.1.

c)

```
        MOV     A, #80H         ; Từ điển khiển
        MOV     DPTR, # 4003H   ; Nạp địa chỉ cổng của thanh ghi điều khiển
        MOVX    @DPTR, A        ; Xuất từ điển khiển
        MOV     A, # 55H        ; Gán A = 55
AGAIN:  MOV     DPTR, # 4000H   ; Địa chỉ cổng PA
        MOVX    @DPTR, A        ; Lấy các bit cổng PA
        INC     DPTR            ; Địa chỉ cổng PB
        MOVX    @DPTR, A        ; Lấy các bít cổng PB
        INC     DPTR            ; Địa chỉ cổng PC
        MOVX    @DPTR, A        ; Lấy các bít cổng PC
        CPL     A               ; Lấy các bít thanh ghi A
        ACALL   DELAY           ; Chờ
        SJMP    AGAIN           ; Tiếp tục
```



Hình 15.4: Nối ghép 8051 với 8255 cho ví dụ 15.2.

Ví dụ 15.3:

Đối với hình 15.5:

a) Tìm các địa chỉ cổng vào ra được gán cho các cổng A, B, C và thanhg ghi điều khiển.

b) Tìm byte điều khiển đối với PA bằng đầu vào, PB bằng đầu ra, PC bằng đầu ra

c) Viết một chương trình để nhận dữ liệu từ PA gửi nó đến cả cổng B và cổng C.

Lời giải:

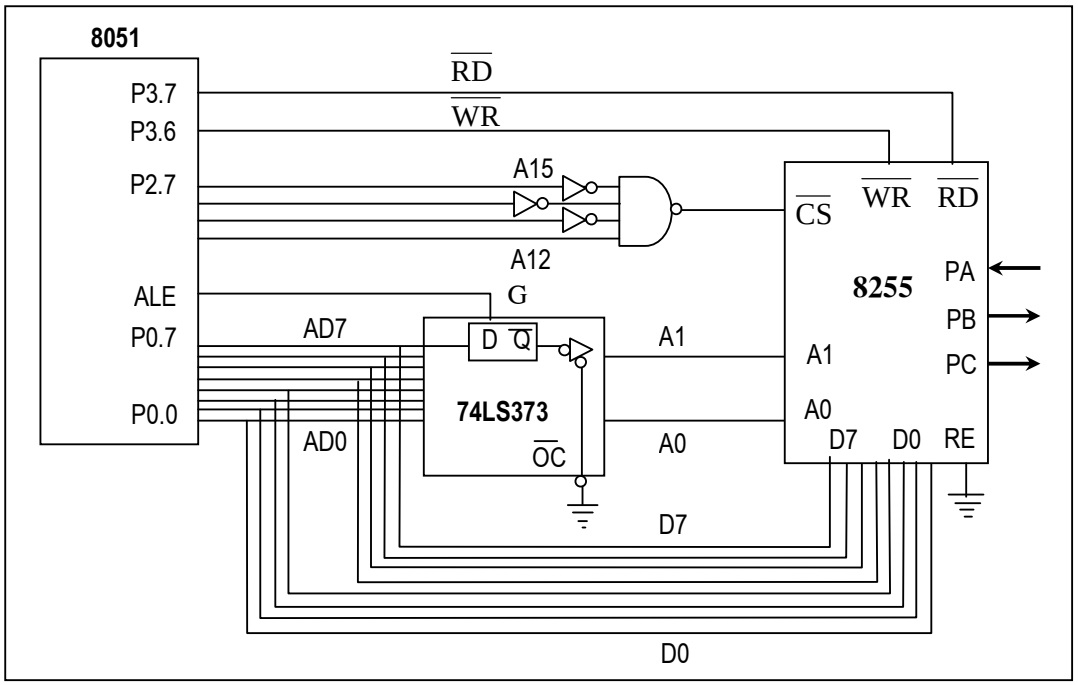
a) Giả sử tất các các bit không dùng đến là 0 thì địa chỉ cổng cơ sở cho 8255 là 1000H. Do vậy ta có:

1000H là PA; 1001H là PB; 1002H là PC và 1003H là thanh ghi điều khiển.

b) Từ điều khiển cho trường hợp này là 10010000 hay 90H.

c)

```
MOV     A, #90H ; PA là đầu vào, PB là đầu ra, PC là đầu ra
MOV     DPTR, #1003H   ; Nạp địa chỉ cổng của thanh ghi điều khiển
MOVX    @DPTR, A       ; Xuất từ điều khiển
MOV     DPTR, #1000H   ; Địa chỉ PA
MOVX    A, @DPTR       ; Nhận dữ liệu từ PA
INC     DPTR           ; Địa chỉ PB
MOVX    @DPTR, A       ; Gửi dữ liệu ra PB
INC     DPTR           ; Địa chỉ PC
MOVX    @DPTR, A       ; Gửi dữ liệu ra PC
```



Hình 15.5: Nối ghép 8051 tới 8255 cho ví dụ 15.3.

Đối với ví dụ 15.3 ta nên dùng chỉ lệnh EQU cho địa chỉ các cổng A, B, C và thanh ghi điều khiển CNTPORT như sau:

```
APORT  EQU    1000H
BPORT  EQU    1001H
CPORT  EQU    1002H
```

```
        CNTPORT     EQU    1003H

        MOV         A, #90H                 ; PA là đầu vào, PB là đầu ra, PC là đầu ra
        MOV         DPTR, #CNTPORT          ; Nạp địa chỉ của cổng thanh ghi điều khiển
        MOVX        @DPTR, A                ; Xuất từ điều khiển
        MOV         DPTR, #CNTPORT          ; Địa chỉ PA
        MOVX        DPTR, APORT             ; Nhận dữ liệu PA
        INC         A, @DPTR                ; Địa chỉ PB
        MOVX        DPTR                    ; Gửi dữ liệu ra PB
        INC         DPTR                    ; Địa chỉ PC
        MOVX        DPTR, A         ; Gửi dữ liệu ra PC
hoặc có thể viết lại như sau:
CONTRBYT    EQU     90H                     Xác định PA đầu vào, PB và PC đầu ra
BAS8255P    EQU     1000H                   ; Địa chỉ cơ sở của 8255

            MOV     A, #CONTRBYT
            MOV     DPTR, #BAS8255P+3       ; Nạp địa chỉ cổng C
            MOVX    @DPTR, A                ; Xuất từ điều khiển
            MOV     DPTR, #BASS8255P        ; Địa chỉ cổng A
            ...
```

Để ý trong ví dụ 15.2 và 15.3 ta đã sử dụng thanh ghi DPTR vì địa chỉ cơ sở gán cho 8255 là 16 bit. Nếu địa chỉ cơ sở dành cho 8255 là 8 bit, ta có thể sử dụng các lệnh "MOVX A, @R0" và "MOVX @R0, A" trong đó R0 (hoặc R1) giữ địa chỉ cổng 8 bit của cổng. Xem ví dụ 15.4, chú ý rằng trong ví dụ 15.4 ta sử dụng một cổng lôgíc đơn giản để giải mã địa chỉa cho 8255. Đối với hệ thống có nhiều 8255 ta có thể sử dụng 74LS138 để giải mã như sẽ trình bày ở ví dụ 15.5.

15.1.5 Các bí danh của địa chỉ (Addren Alias).

Trong các ví dụ 15.4 và 15.4 ta giải mã các bít địa chỉ A0 - A7, tuy nhiên trong ví dụ 15.3 và 15.2 ta đã giải mã một phần các địa chỉ cao của A8 - A15. Việc giải mã từng phần này dẫn đến cái gọi là các bí danh của địa chỉ (Address Aliases). Hay nói cách khác, cùng cổng vật lý giống nhau có các địa chỉ khác nhau, do vậy cùng một cổng mà được biết với các tên khác nhau. Trong ví dụ 15.2 và 15.3 ta có thể thay đổi tốt x thành các tổ hợp các số 1 và 0 khác nhau thành các địa chỉ khác nhau, song về thực chất chúng tham chiếu đến cùng một cổng vật lý. Trong tài liệu thuyết minh phần cứng của mình chúng ta cần phải bảo đảm ghi chú đầy đủ các bí danh địa chỉa nếu có sao cho mọi người dùng biết được các địa chỉ có sẵn để họ có thể mở rộng hệ thống.

Ví dụ 15.4:

Cho hình 15.6:

a) Hãy tìm các địa chỉ cổng vào/ ra được gán cho các cổng A, B, C và thanh ghi điều khiển.

b) Tìm từ điều khiển cho trường hợp PA là đầu ra, PB là đầu vào, PC - PC3 là đầu vào và CP4 - CP7 là đầu ra.

c) Viết một chương trình để nhận dữ liệu từ PB và gửi nó ra PA. Ngoài ra, dữ liệu từ PC1 được gửi đến CPU.

Lời giải:

a) Các địa chỉ cổng được tìm thấy như sau:

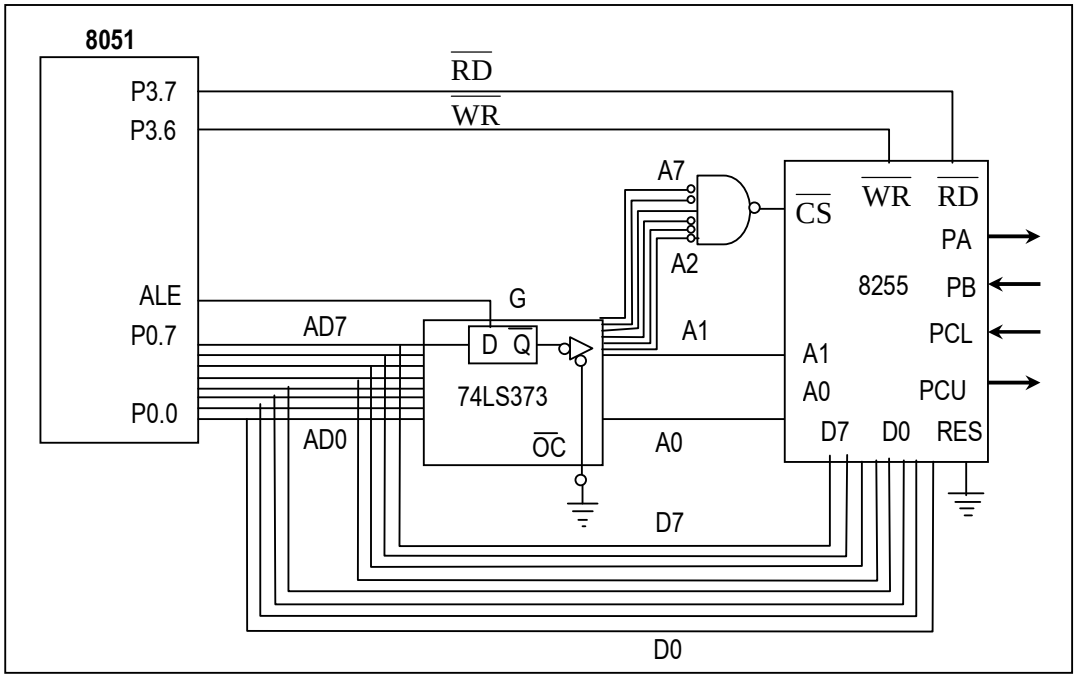
BB	$\overline{CS}$	A1	A0	Địa chỉ	Cổng
0010	00	0	0	20H	Cổng A
0010	00	0	1	21H	Cổng B

0010	00	1	0	22H	Cổng C
0010	00	1	1	23H	Thanh ghi điều khiển

b) Từ điều khiển là 10000011 hay 83H.

c)

```
CONTRBYT    EQU     83H         ; PA là đầu ra, PB,PCL là đầu vào
APORT       EQU     20H
BPORT       EQU     21H
CPORT       EQU     22H
CNTPORT     EQU     23H
            ...
            MOV     A, #CONTRBYT
            MOV     A, #CONTRBYT ; PA, PCU là đầu ra, PB và PCL là đầu vào
            MOV     R0, #CNTPORT ; Nạp địa chỉ của cổng thanh ghi điều khiển
            MOVX    @R0, A       ; Xuất từ điều khiển
            MOV     R0, #BPORT   ; Nạp địa chỉ PB
            MOVX    A, @R0       ; Đọc PB
            DEC     R0           ; Chỉ đến PA (20H)
            MOVX    @R0, A       ; Gửi nó đến PA
            MOV     R0, #CPORT   ; Nạp địa chỉ PC
            MOVX    A, @R0       ; Đọc PCL
            ANL     A, #0FH      ; Che phần cao
            SWAP    A            ; Trao đổi phần cao và thấp
            MOVX    @R0, A       ; Gửi đến PCU
```



Hình 15.6: Nối ghép 8051 với 8255 cho ví dụ 15.4.

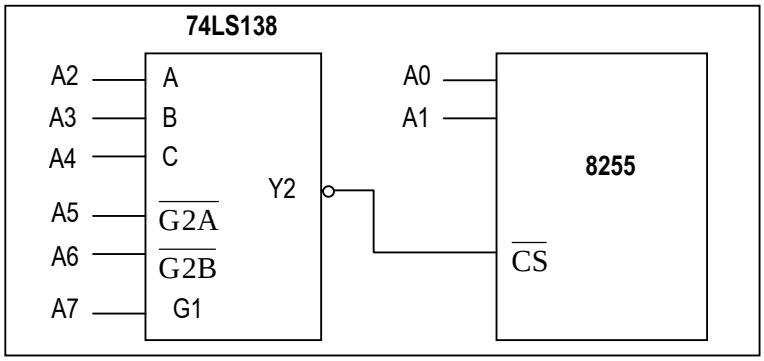
Ví dụ 15.5:

Hãy tìm địa chỉ cơ sở cho 8255 trên hình 15.7.

Lời giải:

GA	$\overline{G2B}$	$\overline{G2A}$	C	B	A			Địa chỉ
A7	A6	A5	A4	A3	A2	A1	A0	

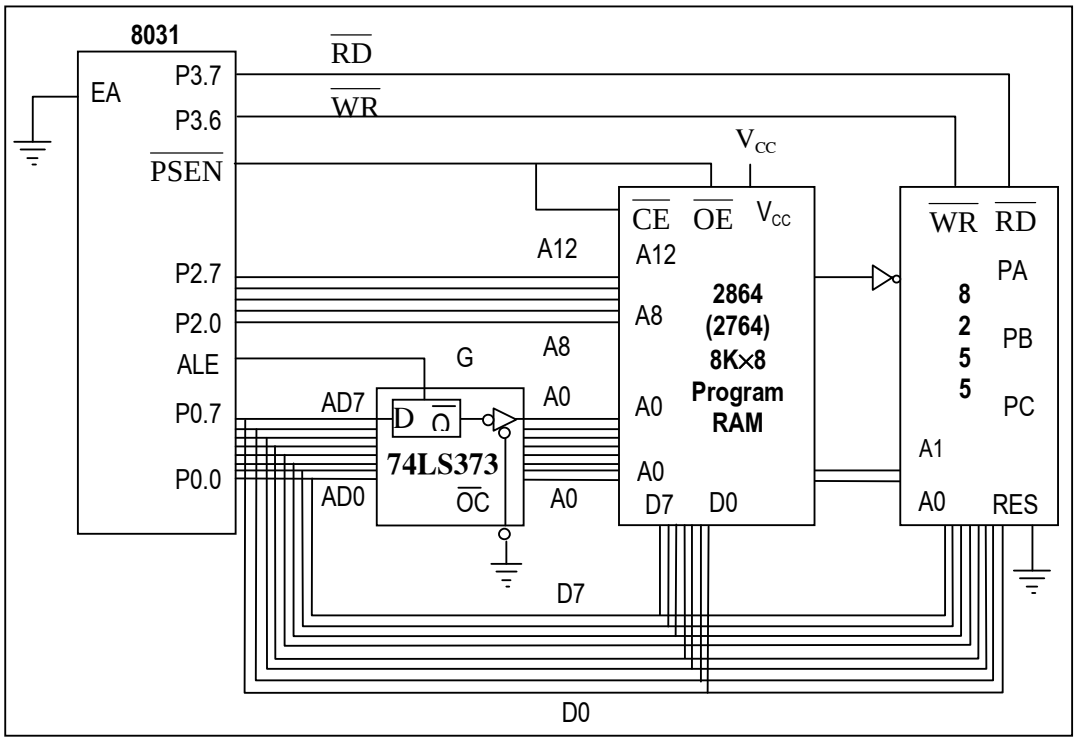
1	0	0	0	1	0	0	0	88H



Hình 15.7: Giải mã địa chỉ của 8255 sử dụng 74LS138.

15.1.6 Hệ 8031 với 8255.

Trong một hệ thống dựa trên 8031 mà bộ nhớ chương trình ROM ngoài là một sự bắt buộc tuyệt đối thì sử dụng một 8255 là rất được trào đón. Điều này là do một thực tế là trong giải trình phối ghép 8031 với bộ nhớ chương trình ROM ngoài ta bị mất hai cổng P0 và P2 và chỉ còn lại duy nhất cổng P1. Do vậy, việc nối với một 8255 là cách tốt nhất để có thêm một số cổng. Điều này được chỉ ra trên hình 15.8.

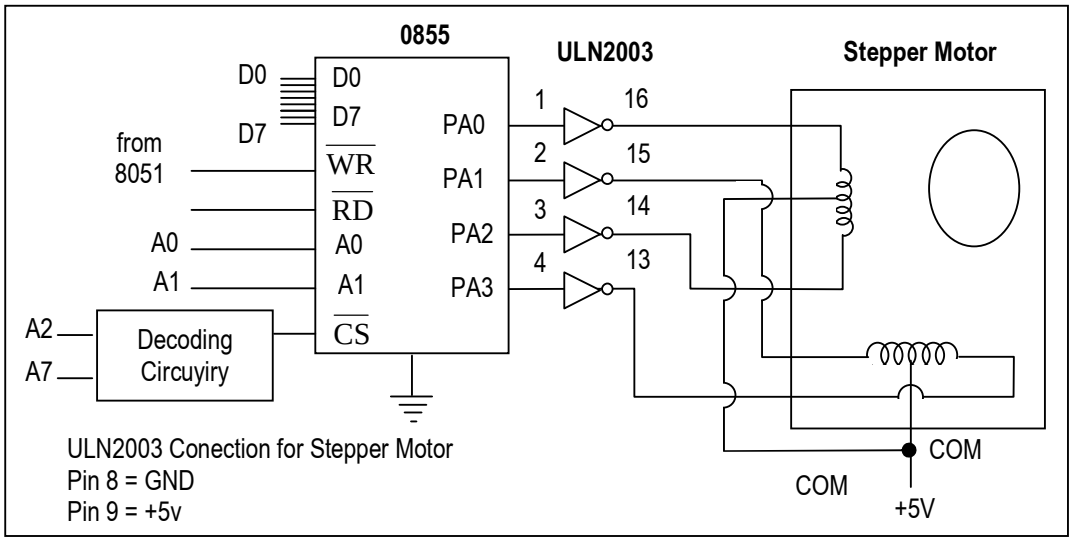


Hình 15.8: Nối 8031 tới một ROM chương trình ngoài và 8255.

15.2 Nối ghép với thế giới thực.

15.2.1 Phối ghép 8255 với động cơ bước.

Chương 13 đã nói chi tiết về phối ghép động cơ bước với 8051, ở đây ta trình bày nối ghép động cơ bước tới 8255 và lập trình (xem hình 15.9).



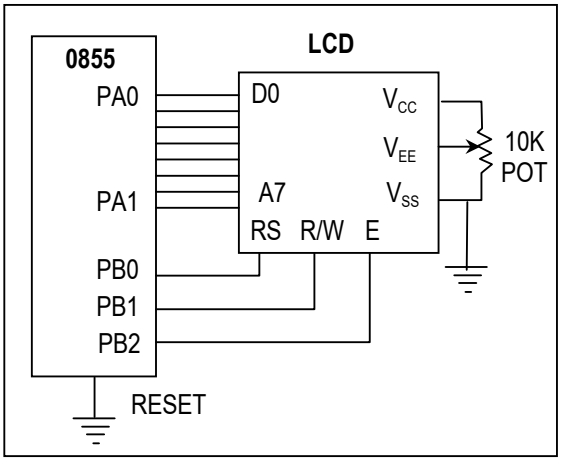
Hình 15.9: Nối ghép 8255 với một động cơ bước.

Chương trình cho sơ đồ nối ghép này như sau:

```
        MOV     A, #80H         ; Chọn từ điều khiển để PA là đầu ra
        MOV     R1, #CRPORT     ; Địa chỉ cổng thanh ghi điều khiển
        MOVX    @R1, A          ; Cấu hình cho PA đầu ra
        MOV     R1, #APORT      ; Nạp địa chỉ cổng PA
        MOV     A, #66H         ; Gán A = 66H, chuyển xung của động cơ bước
AGAIN:  MOVX    @R1, A          ; Xuất chuỗi động cơ đến PA
        RR      A               ; Quay chuỗi theo chiều kim đồng hồ
        ACALL   DELAY           ; Chờ
        SJMP    AGAIN
```

15.2.2 Phối ghép 8255 với LCD.

Chương trình 15.1 trình bày cách xuất các lệnh và dữ liệu tới một LCD được nối tới 8255 theo sơ đồ hình 15.10. Trong chương trình 15.1 ta phải đặt một độ trễ trước mỗi lần xuất thông tin bất kỳ (lệnh hoặc dữ liệu) tới LCD. Một cách tốt hơn là kiểm tra cờ bận trước khi xuất bất kỳ thứ gì tới LCD như đã nói ở chương 12. Chương trình 15.2 lặp lại chương trình 15.1 có sử dụng kiểm tra cờ bận. Để ý rằng lúc này không cần thời gian giữ chậm như ở vị trí 15.1.



Hình 5.10: Nối ghép 8255 với LCD.

Chương 15.1:

```
; Ghi các lệnh và dữ liệu tới LCD không có kiểm tra cờ bận.
; Giả sử PA của 8255 được nối tới D0 - D7 của LCD và
; IB - RS, PB1 = R/W, PB2 = E để nối các chân điều khiển LCD
                MOV     A, #80H         ; Đặt tất cả các cổng 8255 là đầu ra
                MOV     R0, #CNTPORT    ; Nẹp địa chỉ thanh ghi điều khiển
                MOVX    @R0, A          ; Xuất từ điều khiển
                MOV     A, #38H         ; Cấu hình LCD có hai dòng và ma trận 5×7
                ACALL   CMDWRT          ; Ghi lệnh ra LCD
                ACALL   DELAY           ; Chờ đến lần xuất kế tiếp (2ms)
                MOV     A, # 0EH        ; Bật con trỏ cho LCD
                ACALL   CMDWRT          ; Ghi lệnh này ra LCD
                ACALL   DELAY           ; Chờ lần xuất kế tiếp
                MOV     A, # 01H        ; Xoá LCD
                ACALL   CMDWRT          ; Ghi lệnh này ra LCD
                ACALL   DELAY           ; Dịch con trỏ sang phải
                MOV     A, # 06         ; Ghi lệnh này ra LCD
                ACALL   CMDWRT          ; Chờ lần xuất sau
                ACALL   DELAY           ; Ghi lệnh này ra LCD
                ...                     ; v.v... cho tất cả mọi lệnh LCD
                MOV     A, # 'N'        ; Hiển thị dữ liệu ra (chữ N)
                ACALL   DATAWRT         ; Gửi dữ liệu ra LCD để hiển thị
                ACALL   DELAY           ; Chờ lần xuất sau
                MOV     A, # '0'        ; Hiển thị chữ "0"
                ACALL   DATAWRT          ; Gửi ra LCD để hiển thị
                ACALL   DELAY           ; Chờ lần xuất sau
                ...                     ; v.v... cho các dữ liệu khác
; Chương trình con ghi lệnh CMDWRT ra LCD
CMDWRT:         MOV     R0, # APORT     ; Nạp địa chỉ cổng A
                MOVX    @R0, A          ; Xuất thông tin tới chân dữ liệu của LCD
                MOV     R0, # BPORT     ; Nạp địa chỉ cổng B
                MOV     A, # 00000100B  ; RS=0, R/W=1, E=1 cho xung cao xuống thấp

                MOVX    @R0, A          ; Kích hoạt các chânRS, R/W, E của LCD
                NOP                     ; Tạo độ xung cho chân E
                NOP
                MOV     A, # 00000000B  ; RS=0, R/W=1, E=1 cho xung cao xuống thấp

                MOVX    @R0, A          ; Chốt thông tin trên chân dữ liệu của LCD
                RET
; Chương trình con ghi lệnh DATAWRT ghi dữ liệu ra LCD.
CMDWRT:         MOV     R0, # APORT     ; Nạp địa chỉ cổng A
                MOVX    @R0, A          ; Xuất thông tin tới chân dữ liệu của LCD
                MOV     R0, # BPORT     ; Đặt RS=1, R/W=0, E=0 cho xung cao xuống thấp
                MOV     A, # 00000101B  ; Kích hoạt các chân RS, R/W, E
                MOVX    @R0, A          ; Tạo độ xung cho chân E
                NOP
                NOP
                MOV     A, # 00000001B  ; Đặt RS=1, R/W=0, E=0 cho xung cao xuống thấp
                MOVX    @RC, A          ; Chốt thông tin trên chân dữ liệu của LCD
                RET
```

Chương trình 15.2:

```
; Ghi các lệnh và dữ liệu tới LCD có sử dụng kiểm tra cờ bận.
; Giả sử PA của 8255 được nối tới D0 - D7 của LCD và
; PB0 = RS, PB1 = R/W, PB2 = E đối với 8255 tới các chân điều khiển LCD
```

```
                MOV     A, #80H         ; Đặt tất cả các cổng 8255 là đầu ra
                MOV     R0, #CNTPORT    ; Nạp địa chỉ thanh ghi điều khiển
                MOVX    @R0, A          ; Xuất từ điều khiển
                MOV     A, #38H         ; Chọn LCD có hai dòng và ma trận 5×7
                ACALL   NMDWRT          ; Ghi lệnh ra LCD
                MOV     A, # 0EH        ; Lệnh của LCD cho con trỏ bật
                ACALL   NMDWRT          ; Ghi lệnh ra LCD
                MOV     A, # 01H        ; Xoá LCD
                ACALL   NMDWRT          ; Ghi lệnh ra LCD
                MOV     A, # 06         ; Lệnh dịch con trỏ sang phải
                ACALL   CMDWRT          ; Ghi lệnh ra LCD
                ...                     ; v.v... cho tất cả mọi lệnh LCD
                MOV     A, # 'N'        ; Hiển thị dữ liệu ra (chữ N)
                ACALL   NCMDWRT         ; Gửi dữ liệu ra LCD để hiển thị
                MOV     A, # '0'        ; Hiển thị chữ "0"
                ACALL   NDADWRT         ; Gửi ra LCD để hiển thị
                ...                     ; v.v... cho các dữ liệu khác
; Chương trình con ghi lệnh NCMDWRT có hiển thị cờ bận
NCMDWRT:        MOV     R2,  A          ; Lưu giá trị thanh ghi A
                MOV     A, #90H         ; Đặt PA là cổng đầu vào để đọc trạng thái LCD
                MOV     R0, # CNTPORT   ; Nạp địa chỉ thanh ghi điều khiển
                MOVX    @R0, A          ; Đặt PA đầu vào, PB đầu ra
                MOV     A, # 00000110B  ; RS=0, R/W=1, E=1 đọc lệnh
                MOV     @R0, BPORT      ; Nạp địa chỉ cổng B
                MOVX    R0, A           ; RS=0, R/W=1 cho các chân RD và RS

                MOV     R0, APORT       ; Nạp địa chỉ cổng A
READY:          MOVX    @R0             ; Đọc thanh ghi lệnh
                RLC     A               ; Chuyển D7 (cờ bận) vào bit nhớ carry
                JC      READY           ; Chờ cho đến khi LCD sẵn sàng
                MOV     A, #80H         ; Đặt lại PA, PB thành đầu ra
                MOV     R0, #CNTPORT    ; Nạp địa chỉ cổng điều khiển
                MOVX    @R0, A          ; Xuất từ điều khiển tới 8255
                MOV     A, R2           ; Nhận giá trị trả lại  tới LCD
                MOV     R0, #APORT      ; Nạp địa chỉ cổng A
                MOVX    @R0, A          ; Xuất thông tin tới các chân dữ liệu của LCD
                MOV     R0, #BPORT      ; Nạp địa chỉ cổng B
                MOV     A, #00000100B   ; Đặt RS=0, R/W=0, E=1 cho xung thấp lên cao
                MOVX    @R0, A          ; Kích hoạt RS, R/W, E của LCD
                NOP                     ; Tạo độ rộng xung của chân E
                NOP
                MOV     A, #00000000B   ; Đặt RS=0, R/W=0, E=0 cho xung cao xuống thấp
                MOVX    @R0, A          ; Chốt thông tin ở chân dữ liệu LCD
                RET
; Chương trình con ghi dữ liệu mới NDATAWRT sử dụng cờ bận
NCMDWRT:        MOV     R2,  A          ; Lưu giá trị thanh ghi A
                MOV     A, #90H         ; Đặt PA là cổng đầu vào để đọc trạng thái LCD
                MOV     R0, # CNTPORT   ; Nạp địa chỉ thanh ghi điều khiển
                MOVX    @R0, A          ; Đặt PA đầu vào, PB đầu ra
                MOV     A, # 00000110B  ; RS=0, R/W=1, E=1 đọc lệnh
                MOV     @R0, BPORT      ; Nạp địa chỉ cổng B
                MOVX    R0, A           ; RS=0, R/W=1 cho các chân RD và RS

                MOV     R0, APORT       ; Nạp địa chỉ cổng A
READY:          MOVX    @R0             ; Đọc thanh ghi lệnh
                RLC     A               ; Chuyển D7 (cờ bận) vào bit nhớ carry
```

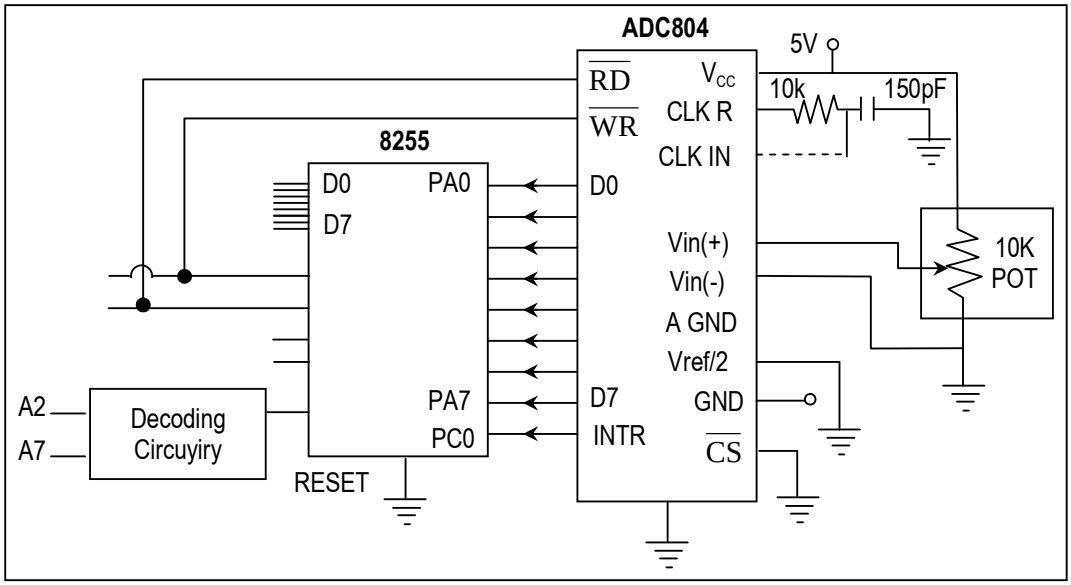
```
        JC      READY           ; Chờ cho đến khi LCD sẵn sàng
        MOV     A, #80H         ; Đặt lại PA, PB thành đầu ra
        MOV     R0, #CNTPORT    ; Nạp địa chỉ cổng điều khiển
        MOVX    @R0, A          ; Xuất từ điều khiển tới 8255
        MOV     A, R2           ; Nhận giá trị trả lại  tới LCD
        MOV     R0, #APORT      ; Nạp địa chỉ cổng A
        MOVX    @R0, A          ; Xuất thông tin tới các chân dữ liệu của LCD
        MOV     R0, #BPORT      ; Nạp địa chỉ cổng B
        MOV     A, #00000101B   ; Đặt RS=1, R/W=0, E=1 cho xung thấp lên cao
        MOVX    @R0, A          ; Kích hoạt RS, R/W, E của LCD
        NOP                     ; Tạo độ rộng xung của chân E
        NOP
        MOV     A, #00000001B   ; Đặt RS=1, R/W=0, E=0 cho xung cao xuống thấp
        MOVX    @R0, A          ; Chốt thông tin ở chân dữ liệu LCD
        RET
```

15.2.3 Nối ghép ADC tới 8255.

Các bộ ADC đã được trình bày ở chương 12. Dưới đây một chương trình chỉ một bộ ADC được nối tới 8255 theo sơ đồ cho trên hình 115.11.

```
        MOV     A, #80H         ; Từ điều khiển với PA = đầu ra và PC  = đầu vào
        MOV     R1, #CRPORT     ; Nạp địa chỉ cổng điều khiển
        MOVX    @R1, A          ; Đặt PA = đầu ra và PC = đầu vào
BACK:   MOV     R1, #CPORT      ; Nạp địa chỉ cổng C
        MOVX    A, @R1          ; Đọc địa chỉ cổng C để xem ADC đã sẵn sàng chưa
        ANL     A,, #00000001B  ; Che tất cả các bít cổng C để xem ADC đã sẵn
sàng chưa
        JNZ     BACK            ; Giữ hiển thị PC0 che EOC
                                ; Kết thúc hội thoại và bây giờ nhận dữ liệu của ADC
        MOV     R1, #APORT      ; Nạp địa chỉ PA
        MOVX    A, @R1          ; A = đầu vào dữ liệu tương tự
```

Cho đến đây ta đã được trao đổi chế độ vào/ ra đơn giản của 8255 và trình bày nhiều ví dụ về nó. Ta xét tiếp các chế độ khác.

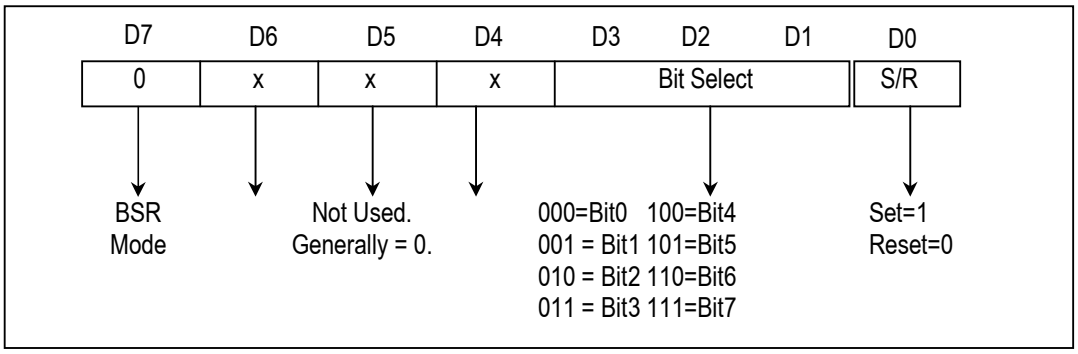


Hình 15.11: Nối ghép ADC 804 với 8255.

15.3 Các chế độ khác của 8255.

15.3.1 Chế độ thiết lập/ xoá bit BSR.

Một đặc tính duy nhất của cổng C là các bit có thể được điều khiển riêng rẽ. Chế độ BSR cho phép ta thiết lập các bit PC0 - PC7 lên cao xuống thấp như được chỉ ra trên hình 15.12. Ví dụ 15.6 và 15.7 trình bày cách sử dụng chế độ này như thế nào?

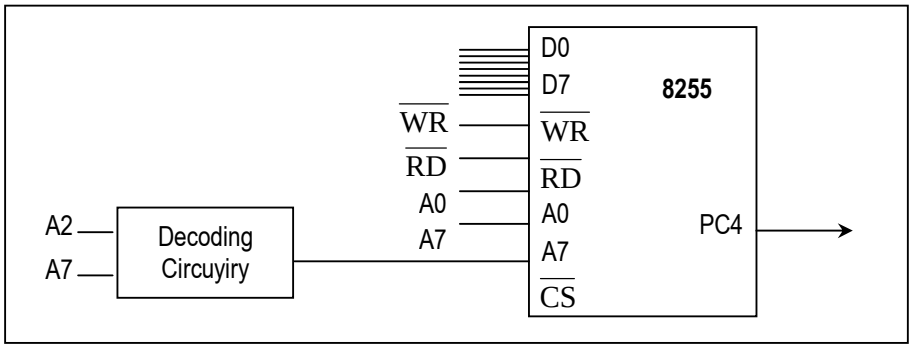


Hình 15.12: Từ điều khiển của chế độ BSR.

Ví dụ 15.6:

Hãy lập trình PCA của 8255 ở chế độ BSR thì bit D7 của từ điều khiển phải ở mức thấp. Để PC4 ở mức cao, ta cần một từ điều khiển là "0xxx1001" và ở mức thấp ta cần "0xxx1000". Các bít được đánh dấu x là ta không cần quan tâm và thường chúng được đặt về 0.

```
MOV     A, 00001001B    ; Đặt byte điều khiển cho PC4 =1
MOV     R1, #CNTPORT    ; Nạp cổng thanh ghi điều khiển
MOVX    @R1, A          ; Tạo PC4 = 1
ACALL   DELAY           ; Thời gian giữ chậm cho xung cao
MOV     A, #00001000B   ; Đặt byte điều khiển cho PC4 = 0
MOVX    @R1, A          ; Tạo PC4 = 0
ACALL   DELAY
```



Hình 15.13: Cấu hình cho ví dụ 15.6 và 15.7.

Ví dụ 15.7:

Hãy lập trình 8255 theo sơ đồ 15.13 để:

a) Đặt PC2 lên cao

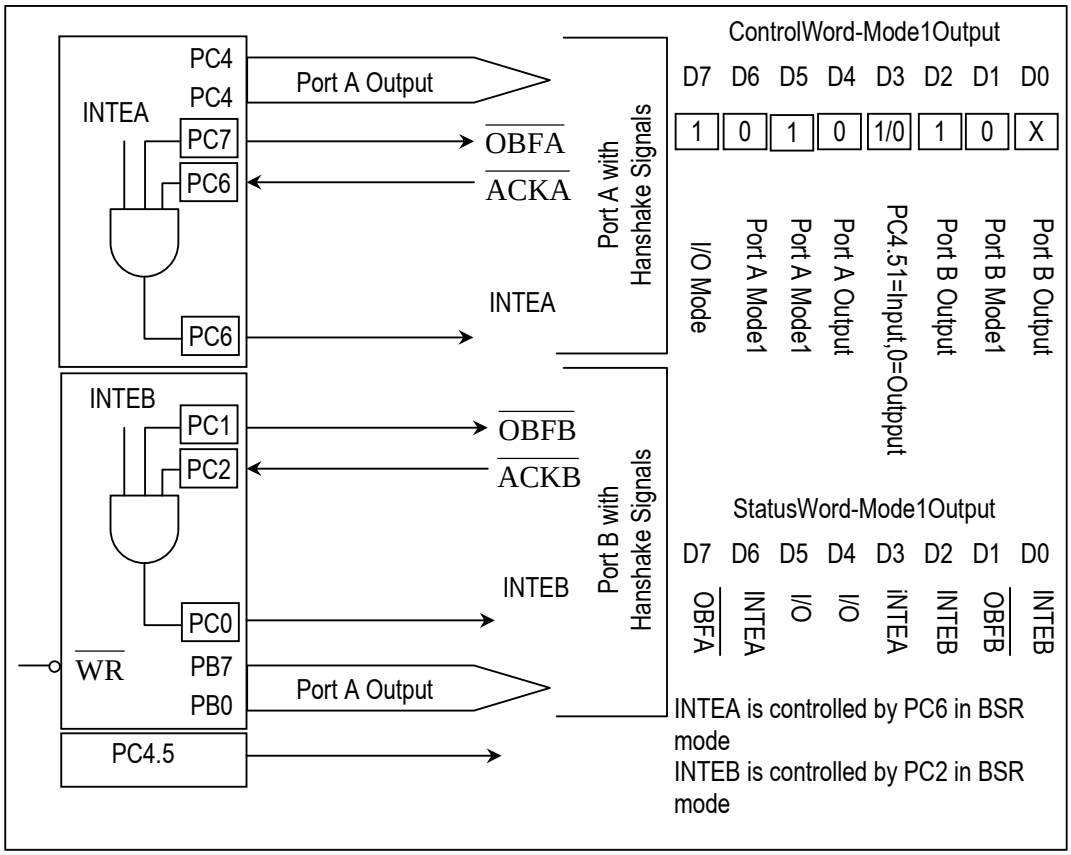
b) Sử dụng PC6 để tạo xung vuông liên tục với 66% độ đầy xung.

Lời giải:

a) MOV R0, # CNTPORT
MOV A, # 0XXX0101 ; Byte điều khiển
MOV @R0, A

b)

```
AGAIN:  MOV     A, #00001101B   ; Chọn PC6 = 1
        MOV     R0, #CNTPORT    ; Nạp địa chỉ thanh ghi điều khiển
        MOVX    @R0, A          ; Tạo PC6 = 1
        ACALL   DELAY
        ACALL   DELAY
        MOV     A, #00001100B   ; PC6 = 0
        ACALL   DELAY   ; Thời gian giữ chậm
        SJMP    AGAIN
```



Hình 15.15: Biểu đồ định thời của 8255 ở chế độ 1.

15.3.2 8255 ở chế độ 1: Vào/ ra với khả năng này bắt tay.

Một trong những đặc điểm mạnh nhất của 8255 là khả năng bắt tay với các thiết bị khác. Khả năng bắt tay là một quá trình truyền thông qua lại của hai thiết bị thông minh. Ví dụ về một thiết bị có các tín hiệu bắt tay là máy in. Dưới đây ta trình bày các tín hiệu bắt tay của 8255 với máy in.

Chế độ 1: Xuất dữ liệu ra với các tín hiệu bắt tay.

Như trình bày trên hình 15.14 thì cổng A và B có thể được sử dụng như các cổng đầu ra để gửi dữ liệu tới một thiết bị với các tín hiệu bắt tay. Các tín hiệu bắt tay cho cả hai cổng A và B được cấp bởi các bit của cổng C. Hình 15.15 biểu đồ định thời của 8255.

Dưới đây là các phần giải thích về các tín hiệu bắt tay và tính hợp lý của chúng đối với cổng A, còn cổng B thì hoàn toàn tương tự.

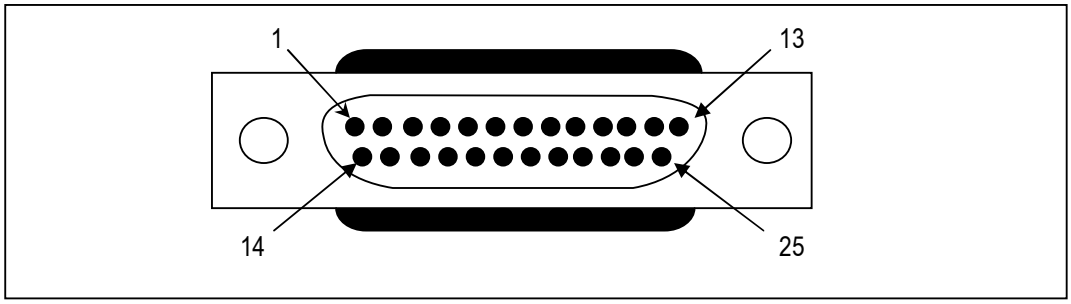
Tín hiệu $\overline{\text{OBFa}}$***:*** Đây là tín hiệu bộ đệm đầu ra đầy của cổng A được tích cực mức thấp đi ra từ chân PC7 để báo rằng CPU đã ghi 1 byte dữ liệu tới cổng A. Tín hiệu này phải được nối tới chân STROBE của thiết bị thu nhận dữ liệu (chẳng hạn như máy in) để báo rằng nó bây giờ đã có thể đọc một byte dữ liệu từ chốt cổng.

Tín hiệu $\overline{\text{ACKa}}$***:*** Đây là tín hiệu chấp nhận do cổng A có mức tích cực mức thấp được nhân tại chân PC6 của 8255. Qua tín hiệu $\overline{\text{ACKa}}$ thì 8255 biết rằng tín hiệu tại cổng A đã được thiết bị thu nhận lấy đi. Khi thiết bị nhận lấy dữ liệu đi từ cổng A nó báo 8255 qua tín hiệu $\overline{\text{ACKa}}$. Lúc này 8255 bật $\overline{\text{OBFa}}$ lên cao để báo rằng dữ liệu tại cổng A bây giờ là dữ liệu cũ và khi CPU đã ghi một byte dữ liệu mới tới cổng A thì $\overline{\text{OBFa}}$ lại xuống thấp v.v...

Tín hiệu INTRa: Đây là tín hiệu yêu cầu ngắt của cổng A có mức tích cực cao đi ra từ chân PC3 của 8255. Tín hiệu $\overline{\text{ACK}}$ là tín hiệu có độ dài hạn chế. Khi nó xuống thấp (tích cực) thì nó làm cho $\overline{\text{OBFa}}$ không tích cực, nó ở mức thấp một thời gian ngắn và sau đó trở nên cao (không tích cực). Sườn lên của $\overline{\text{ACK}}$ kích hoạt INTRa lên cao. Tín hiệu cao này trên chân INTRa có thể được dùng để gây chú ý của CPU. CPU được thông báo qua tín hiệu INTRa rằng máy in đã nhận byte cuối cùng và nó sẵn sàng để nhận byte dữ liệu khác. INTRa ngắt CPU ngừng mọi thứ đang làm và ép nó gửi byte kế tiếp tới cổng A để in. Điều quan trọng là chú ý rằng INTRa được bật lên 1 chỉ khi nếu INTRa, $\overline{\text{OBFa}}$ và $\overline{\text{ACKa}}$ đều ở mức cao. Nó được xoá về không khi CPU ghi một byte tới cổng A.

Tín hiệu INTEa: Đây là tín hiệu cho phép ngắt cổng A 8255 có thể cấm INTRa để ngăn nó không được ngắt CPU. Đây là chức năng của tín hiệu INTEa. Nó là một mạch lật Flip - Flop bên trong thiết kế để che (cấm) INTRa. Tín hiệu INTRa có thể được bật lên hoặc bị xoá qua cổng C trong chế độ BSR vì INTEa là Flip - Plop được điều khiển bởi PC6.

Từ trạng thái: 8255 cho phép hiển thị trạng thái của các tín hiệu INTR, OBF và INTE cho cả hai cổng A và B. Điều này được thực hiện bằng cách đọc cổng C vào thanh ghi tổng và kiểm tra các bit. Đặc điểm này cho phép thực thi thăm dò thay cho ngắt phần cứng.



Hình 15.16: Đầu cắm DB-25.

(hình 15.17 mờ quá không vẽ được)

Hình 15.17: Đầu cáp của máy in Centronics.

Bảng 15.2: Các chân tín hiệu của máy in Centronics.

Chân số	Mô tả	Chân số	Mô tả
1	STROBE	11	Bận (busy)
2	Dữ liệu D0	12	Hết giấy (out of paper)
3	Dữ liệu D1	13	Chọn (select)
4	Dữ liệu D2	14	Tự nạp ($\overline{\text{Autofeed}}$)
5	Dữ liệu D3	15	Lỗi ($\overline{\text{Error}}$)
6	Dữ liệu D4	16	Khởi tạo máy in
7	Dữ liệu D5	17	Chọn đầu vào ($\overline{\text{Select input}}$)
8	Dữ liệu D6	18-25	Đất (ground)
9	Dữ liệu D7		
10	$\overline{\text{ACK}}$ (chấp nhận)		

Các bước truyền thông có bắt tay giữa máy in và 8255.

Một byte dữ liệu được gửi đến bus dữ liệu máy in.

Máy in được báo có 1 byte dữ liệu cần được in bằng cách kích hoạt tín hiệu đầu vào STROBE của nó.

Khi máy nhận được dữ liệu nó báo bên gửi bằng cách kích hoạt tín hiệu đầu ra được gọi là $\overline{\text{ACK}}$ (chấp nhận).

Tín hiệu $\overline{\text{ACK}}$ khởi tạo quá trình cấp một byte khác đến máy in.

Như ta đã thấy từ các bước trên thì chỉ khi một byte dữ liệu tới máy in là không đủ. Máy in phải được thông báo về sự hiện diện của dữ liệu. Khi dữ liệu được gửi thì máy in có thể bận hoặc hết giấy, do vậy máy in phải được báo cho bên gửi khi nào nó nhận và lấy được dữ liệu của nó. Hình 15.16 và 15.17 trình các ổ cắm DB25 và đầu cáp của máy in Centronics tương ứng.

Kỹ thuật vi xử lý
Microprocessors

Giảng viên: **TS. Phạm Ngọc Nam**

Khoa
ĐIỆN TỬ -
VIỄN THÔNG

Your instructor

- Bộ môn kỹ thuật điện tử tin học
  - Office: C9-401
  - Email: pnnam-fet@mail.hut.edu.vn
- Member of Intel Higher Education Program
- Research:
  - FPGA, PSoC, hệ nhúng
  - Trí tuệ nhân tạo
- Education:
  - K37 điện tử-ĐHBK Hà nội (1997)
  - Master về trí tuệ nhân tạo 1999, Đại học K.U. Leuven, vương quốc Bỉ
    - ⇨ Đề tài: Nhận dạng chữ viết tay
  - Tiến sỹ kỹ thuật chuyên ngành điện tử-tin học, 9/ 2004, Đại học K.U. Leuven, Vương Quốc Bỉ
    - ⇨ Đề tài: quản lý chất lượng dịch vụ trong các ứng dụng đa phương tiện tiên tiến

Khoa ĐIỆN TỬ - VIỄN THÔNG

Nội dung môn học

1. Giới thiệu chung về hệ vi xử lý
2. Bộ vi xử lý Intel 8088/8086
3. Lập trình hợp ngữ cho 8086
4. Tổ chức vào ra dữ liệu
5. Ngắt và xử lý ngắt
6. Truy cập bộ nhớ trực tiếp DMA
7. Các bộ vi xử lý trên thực tế
8. Thiết kế bộ vi xử lý

Tài liệu tham khảo

- Slides
- Barry B. Brey, The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium and Pentium Pro Processor: Architecture, Programming, and Interfacing, Fourth Edition, Prentice Hall, 1997.
- Văn Thế Minh, Kỹ thuật vi xử lý, Nhà xuất bản giáo dục, 1997.
- Quách Tuấn Ngọc và cộng sự, Ngôn ngữ lập trình Assembly và máy vi tính IBM-PC, 2 tập, Nhà xuất bản giáo dục, 1995.
- Cảm ơn giáo sư Rudy Lauwereins đã cho phép sử dụng slides của ông

Mục đích của môn học

- Nắm được cấu trúc, nguyên lý hoạt động của bộ vi xử lý và hệ vi xử lý
- Có khả năng lập trình bằng hợp ngữ cho vi xử lý
- Có khả năng lựa chọn vi xử lý thích hợp cho các ứng dụng cụ thể
- Nắm được các bộ vi xử lý trên thực tế
- Nắm được nguyên lý thiết kế vi xử lý

Bài tập lớn và thi

- Bài tập lớn (20% điểm)
  - Thiết kế một hệ thống sử dụng vi xử lý (vi điều khiển, DSP…) hoặc
  - Thiết kế hệ thống card ngoại vi cho máy tính
  - Không được thi lần 1, 2 nếu không làm bài tập lớn
- Điểm chuyên cần (10% điểm)
  - Dự đủ 3 bài kiểm tra và đạt 2/3 bài kiểm tra
  - Không được thi lần 1 nếu vắng 2 bài kiểm tra hoặc không đạt cả 3 bài kiểm tra
- Thi cuối kỳ (70%)
  1. Lý thuyết: Xem mục đích của môn học
  2. Lập trình hợp ngữ
  3. Thiết kế bộ nhớ và thiết bị ngoại vi cho hệ vi xử lý

Khoa ĐIỆN TỬ - VIỄN THÔNG

Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

- 1.1.1 Thế hệ -1: Thời xa xưa (…-1642)
- 1.1.2 Thế hệ 0: Máy tính cơ khí (1642-1945)
- 1.1.3 Thế hệ 1: Đèn điện tử-Vacuum tubes (1945-1955)
- 1.1.4 Thế hệ 2: Transistor rời rạc-Discrete transistors (1955-1965)
- 1.1.5 Thế hệ 3: Mạch tích hợp-Integrated circuits (1965-1980)
- 1.1.6 Thế hệ 4: Mạch tích hợp cỡ lớn-VLSI (1980-?)

1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

1.1.1 Thế hệ -1: Thời xa xưa (…-1642)

1.1.2 Thế hệ 0: Máy tính cơ khí (1642-1945)

1.1.3 Thế hệ 1: Đèn điện tử-Vacuum tubes (1945-1955)

1.1.4 Thế hệ 2: Transistor rời rạc-Discrete transistors (1955-1965)

1.1.5 Thế hệ 3: Mạch tích hợp-Integrated circuits (1965-1980)

1.1.6 Thế hệ 4: Mạch tích hợp cỡ lớn-VLSI (1980-?)

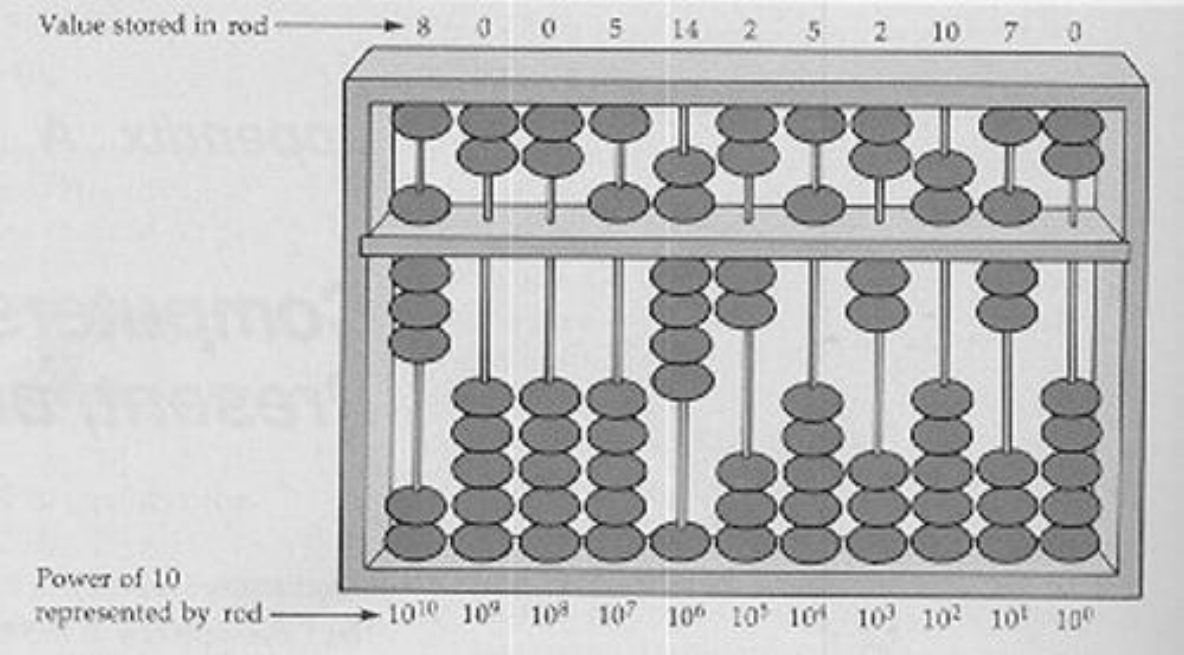
1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

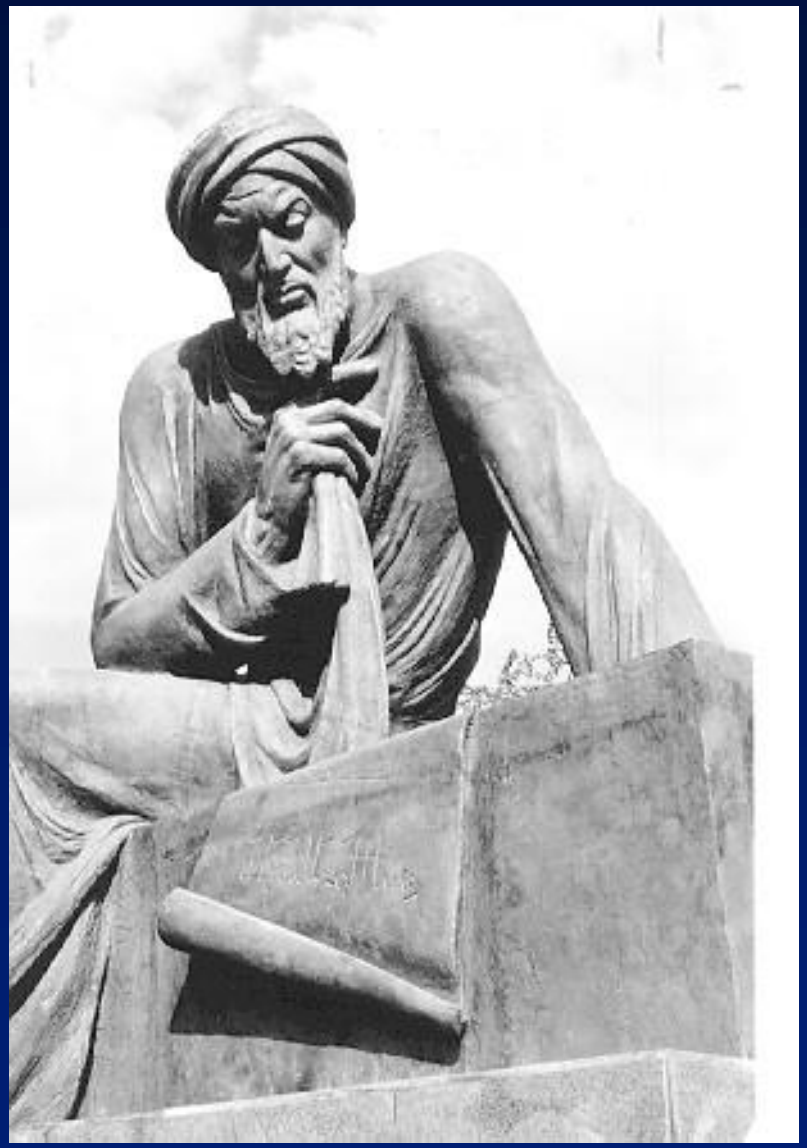
1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

Thế hệ -1: The early days (…-1642)

- Bàn tính, *abaci*, đã được sử dụng để tính toán. Khái niệm về giá trị theo vị trí đã được sử dụng



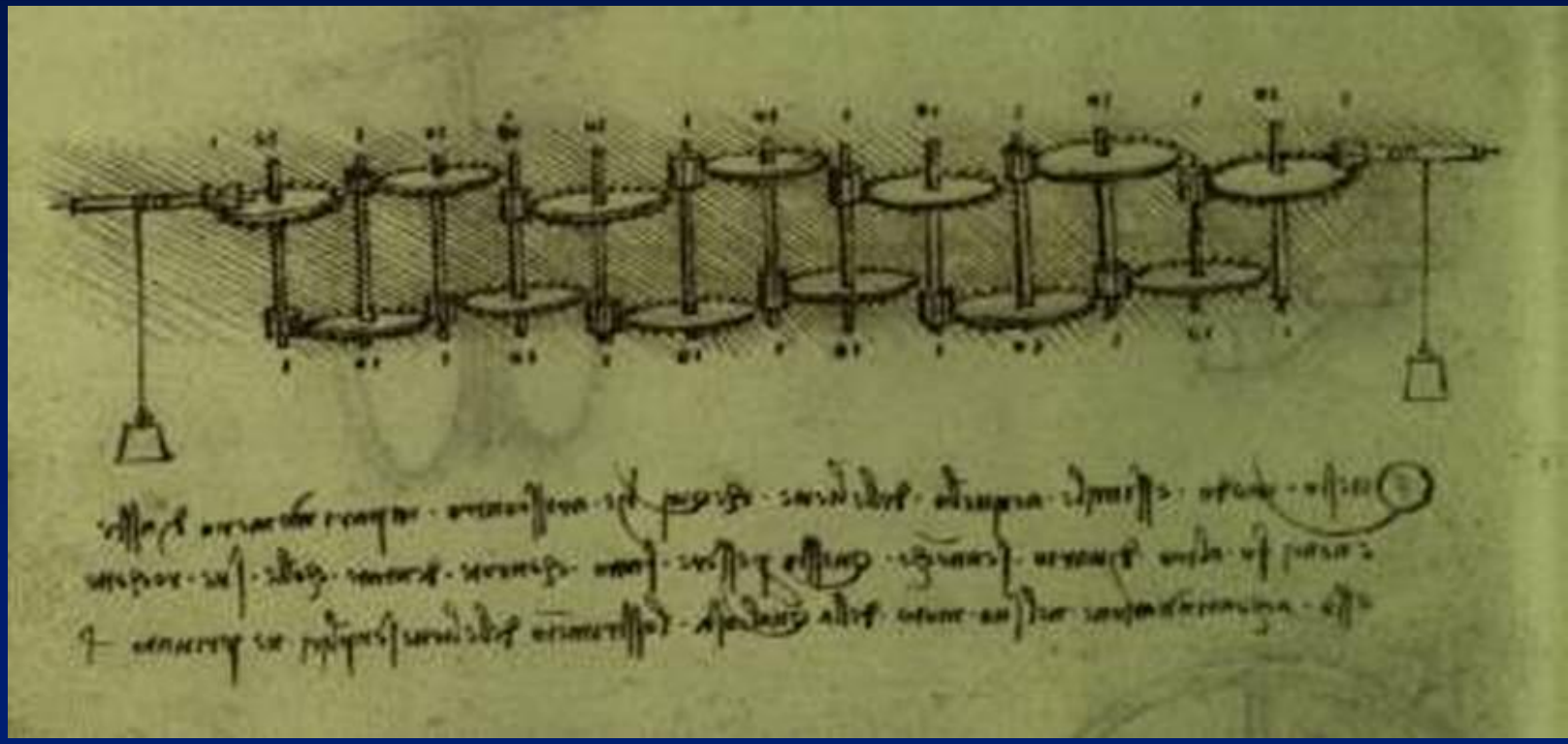
Thế hệ -1: The early days (…-1642)



- Thế kỷ 12: *Muhammad ibn Musa Al'Khowarizmi* đưa ra khái niệm về giải thuật *algorithm*
- Một danh sách các chỉ dẫn mô tả một cách chính xác các bước của một quá trình mà đảm bảo là quá trình này sẽ phải kết thúc sau một số bước nhất định với câu trả lời đúng cho từng trường hợp cụ thể của một vấn đề cần giải quyết

Thế hệ -1: The early days (…-1642)

- Codex Madrid - Leonardo Da Vinci (1500)
  - Vẽ một cái máy tính cơ khí



Khoa ĐIỆN TỬ - VIỄN THÔNG

Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

- 1.1.1 Thế hệ -1: Thời xa xưa (…-1642)
- **1.1.2 Thế hệ 0: Máy tính cơ khí (1642-1945)**
- 1.1.3 Thế hệ 1: Đèn điện tử-Vacuum tubes (1945-1955)
- 1.1.4 Thế hệ 2: Transistor rời rạc-Discrete transistors (1955-1965)
- 1.1.5 Thế hệ 3: Mạch tích hợp-Integrated circuits (1965-1980)
- 1.1.6 Thế hệ 4: Mạch tích hợp cỡ lớn-VLSI (1980-?)

1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

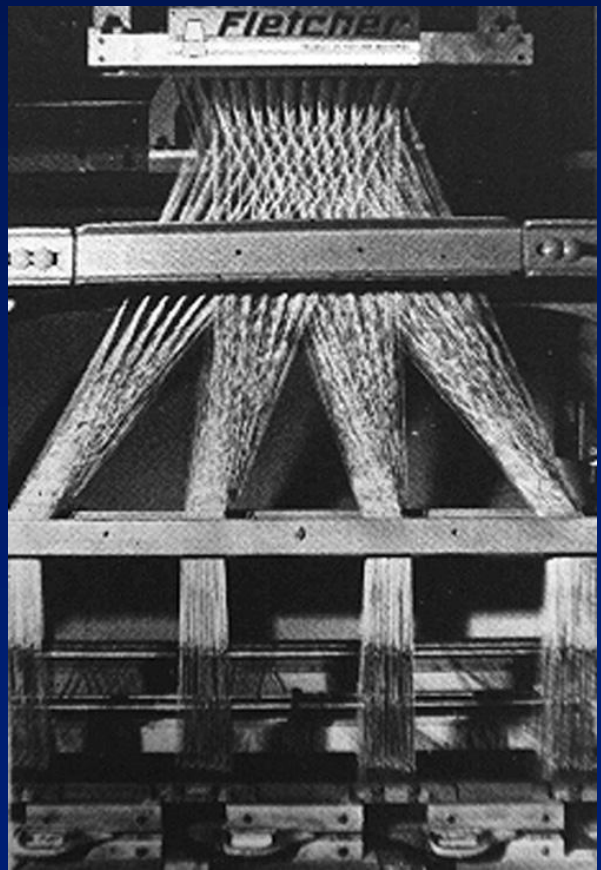
Thế hệ 0: Mechanical (1642-1945)

- Blaise Pascal, con trai của một người thu thuế, đã chế tạo một máy cộng có nhớ vào năm 1642

Thế hệ 0: Mechanical (1642-1945)

- Năm 1801, Joseph-Marie Jacquard đã phát minh ra máy dệt tự động sử dụng bìa đục lỗ để điều khiển hoạ tiết dệt trên vải
- Bìa đục lỗ lưu trữ chương trình: máy đa năng đầu tiên


[Jacquard-card Making.]

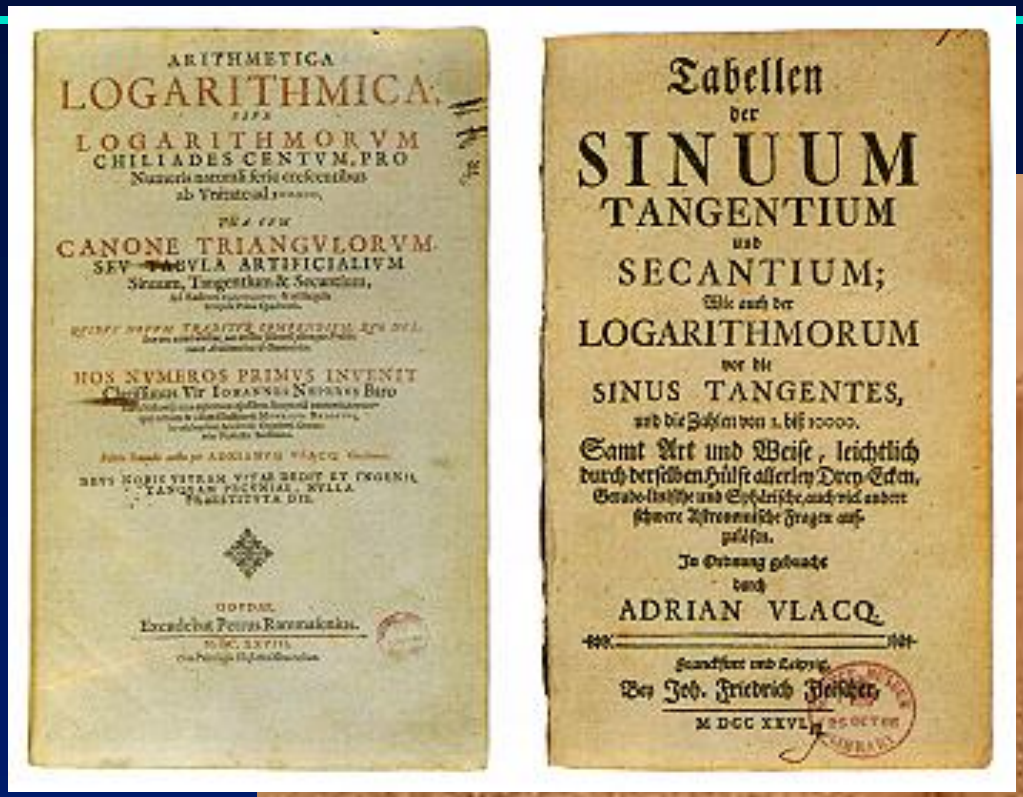


Thế hệ 0: Mechanical (1642-1945)

- 1822, Charles Babbage nhận ra rằng các bảng tính dùng trong hàng hải có quá nhiều lỗi dẫn tới việc rất nhiêu tàu bị mất tích
- Ông đã xin chính phủ Anh hỗ trợ để nghiên cứu về máy tính

Khoa ĐIỆN TỬ - VIỄN THÔNG

Thế hệ 0: Mechanical (1642-1945)



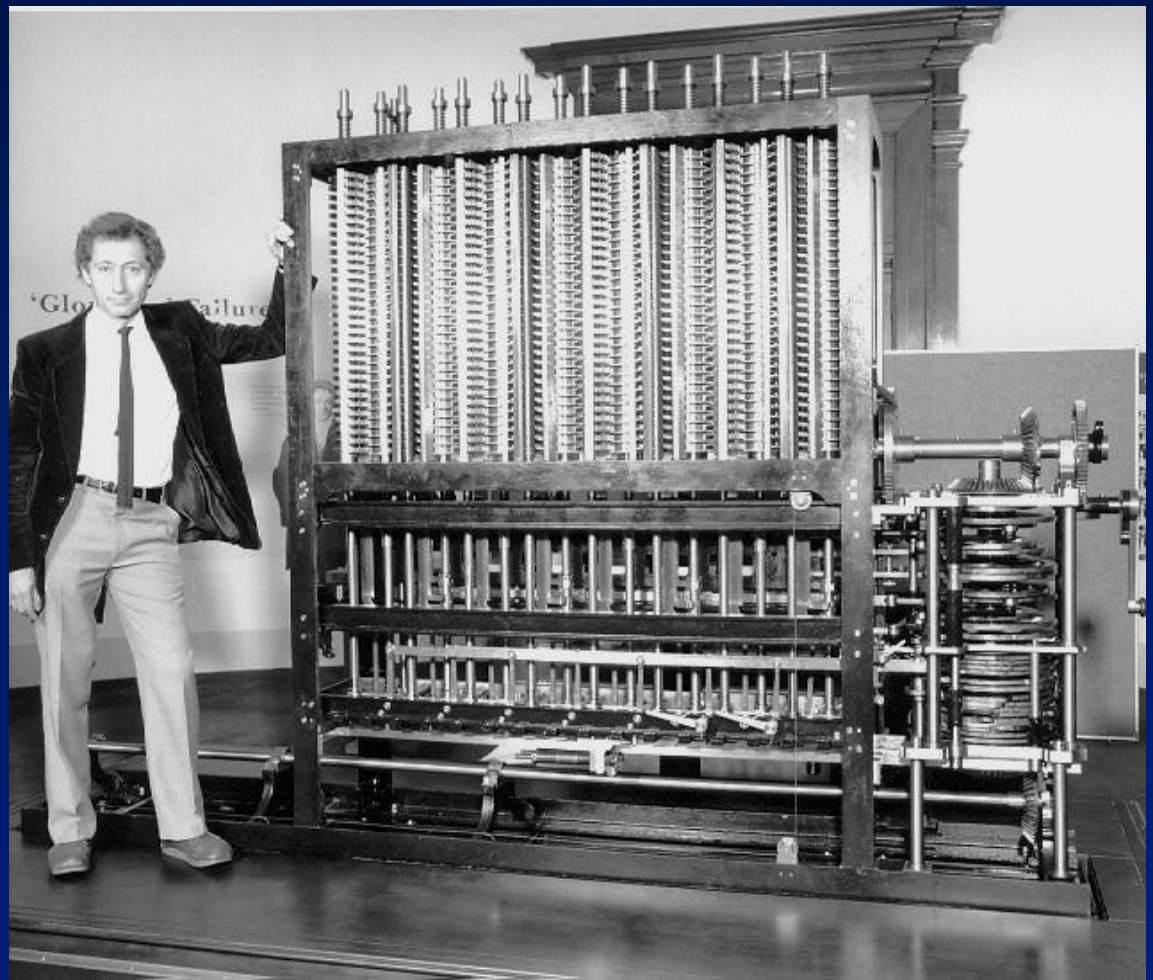
ARITHMETICA
LOGARITHMICA;
SIVE
LOGARITHMORVM
CHILIADES CENTVM, PRO
Numeris naturali serie crescentibus
ab Vnitate ...
ITEM ET
CANONE TRIANGVLORVM
SEV TABVLA ARTIFICIALIVM
Sinuum, Tangentium & Secantium,
HOS NVMEROS PRIMVS INVENIT
Clarissimus Vir IOANNES NEPERVS Baro
LONDINI,
Excudebat Petrus Rutakerius.

Tabellen
der
SINUUM
TANGENTIUM
und
SECANTIUM;
Wie auch der
LOGARITHMORUM
vor die
SINUS TANGENTES,
und die Zahlen von 1. biß 10000.
Samt Art und Weise, leichtlich
durch derselben Hülfe allerley Drey-Ecken,
Gerade-linichte und Sphärische, auch viel andere
schwere Astronomische Fragen auf-
gelöset.
In Ordnung gebracht
durch
ADRIAN VLACQ.
Frankfurt und Leipzig,
Bey Joh. Friedrich Fleischer,
M DCC XXVI.

3,700 errors

Thế hệ 0: Mechanical (1642-1945)

- Babbage đã thiết kế một cái máy vi phân Difference Engine để thay thế toàn bộ bảng tính: máy thực hiện một ứng dụng cụ thể đầu tiên (application specific hard-coded machine)



Thế hệ 0: Mechanical (1642-1945)

- Ada Augusta King, trở thành lập trình viên đầu tiên vào năm 1842 khi cô viết chương trình cho Analytical Engine, thiết bị thứ 2 của Babbage

Thế hệ 0: Mechanical (1642-1945)

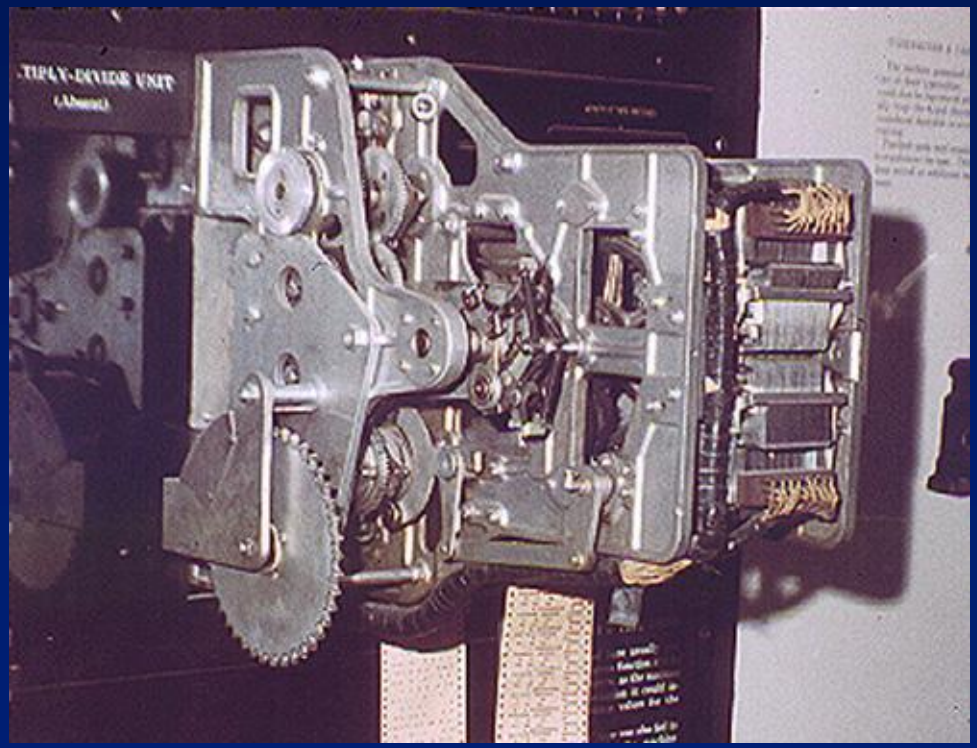
- Herman Hollerith, người Mỹ, thiết kế một máy tính để xử lý dữ liệu về dân số Mỹ 1890
- Ông thành lập công ty, Hollerith Tabulating Company, sau đấy là Calculating-Tabulating-Recording (C-T-R) company vào năm 1914 và sau này được đổi tên là IBM vào năm 1924.

Thế hệ 0: Mechanical (1642-1945)

- Konrad Zuse, Berlin, Đức, phát triển vào năm 1935 máy tính Z-1 sử dụng rơ le và số nhị phân
- Chu kỳ lệnh: 6 giây (0.17 Hz)

Thế hệ 0: Mechanical (1642-1945)

- Máy tính cơ điện tự động lớn đa năng đầu tiên là máy Harvard Mark I (IBM Automatic Sequence Control Calculator), phát minh bởi Howard Aiken vào cuối 1930
- ASCC không phải là máy tính có chương trình lưu trữ sẵn mà các lệnh được ghi vào các băng giấy.

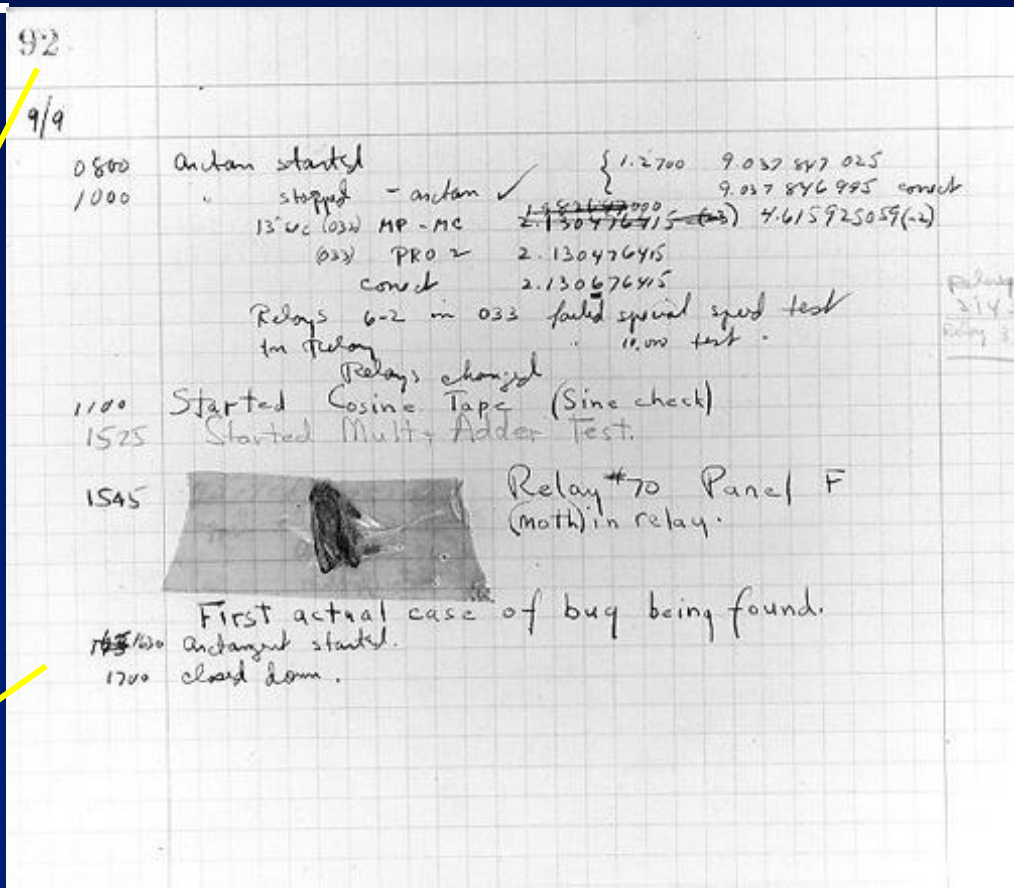


Thế hệ 0: Mechanical (1642-1945)

- Grace Murray Hopper found the first computer bug beaten to death in the jaws of a relay. She glued it into the logbook of the computer and thereafter when the machine stops (frequently) she told Howard Aiken that they are "debugging" the computer.

Numbered pages for USA patents

Lab book!!



Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

- 1.1.1 Thế hệ -1: Thời xa xưa (…-1642)
- 1.1.2 Thế hệ 0: Máy tính cơ khí (1642-1945)
- **1.1.3 Thế hệ 1: Đèn điện tử-Vacuum tubes (1945-1955)**
- 1.1.4 Thế hệ 2: Transistor rời rạc-Discrete transistors (1955-1965)
- 1.1.5 Thế hệ 3: Mạch tích hợp-Integrated circuits (1965-1980)
- 1.1.6 Thế hệ 4: Mạch tích hợp cỡ lớn-VLSI (1980-?)

1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

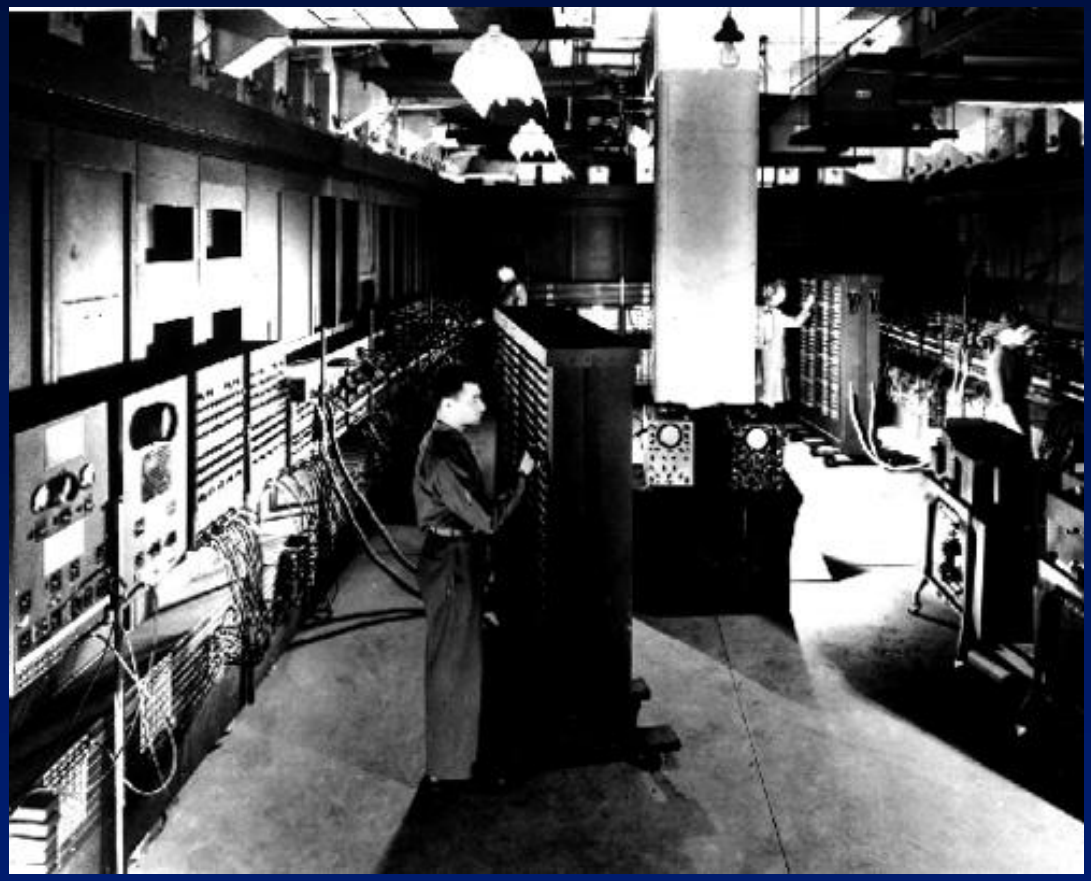
1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

Thế hệ 1: Đèn điện tử (1945-1955)

- Năm 1943, John Mauchly và J. Presper Eckert bắt đầu nghiên cứu về ENIAC

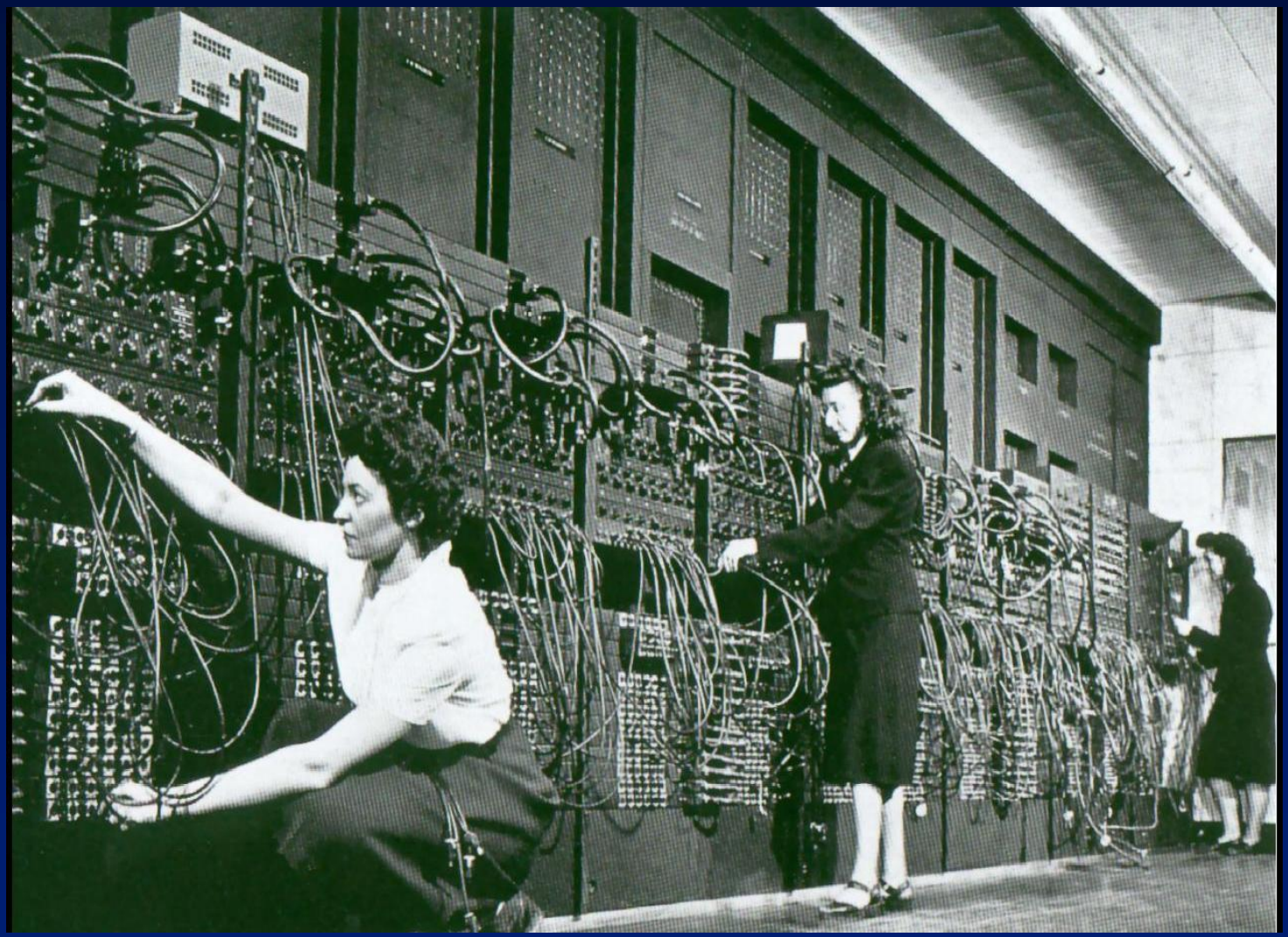
Khoa ĐIỆN TỬ - VIỄN THÔNG

Thế hệ 1: Đèn điện tử (1945-1955)



- 18000 đèn điện tử, 1500 rơ le, 30 tấn, 140 kW, 20 thanh ghi 10 chữ số thập phân, 100 nghìn phép tính/ giây
- “Trong tương lai máy tính sẽ nặng tối đa là 1.5 tấn” (Popular Mechanics, 1949)

Thế hệ 1: Đèn điện tử(1945-1955)



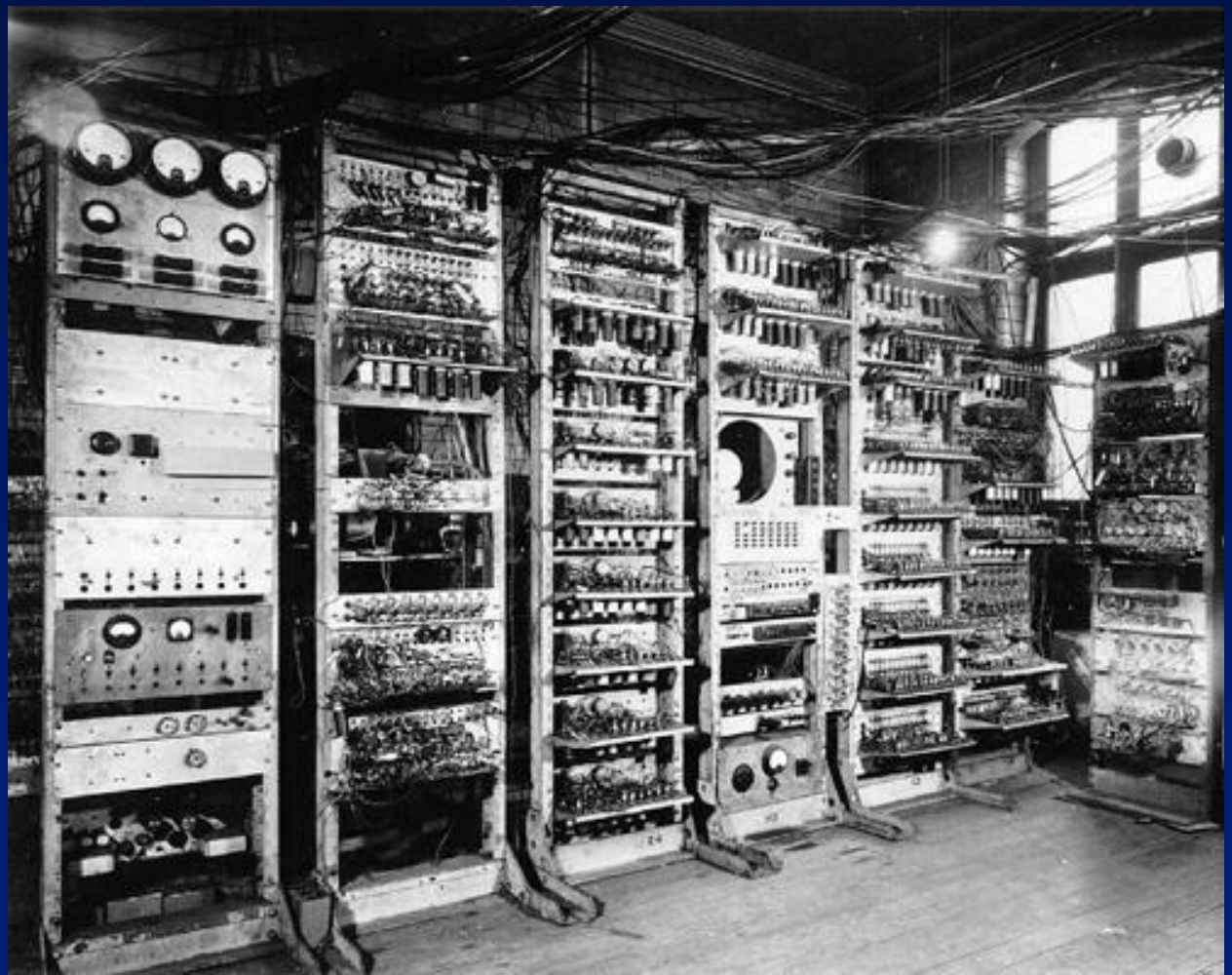
- Lập trình thông qua 6000 công tắc nhiều nấc và nặng hàng tấn

Thế hệ 1: Đèn điện tử (1945-1955)

- Năm 1946, John von Neumann phát minh ra máy tính có chương trình lưu trong bộ nhớ
- Máy tính của ông gồm có một đơn vị điều khiển, một đơn vị xử lý số học và logic (ALU), một bộ nhớ chương trình và dữ liệu và sử dụng số nhị phân thay vì số thập phân.
- Máy tính ngày nay đều có cấu trúc von Neumann
- ông đặt nền móng cho hiện tượng “von Neumann bottleneck”, sự không tương thích giữa tốc độ của bộ nhớ với đơn vị xử lý

Thế hệ 1: Đèn điện tử (1945-1955)

- Năm 1948, máy tính có chương trình lưu trữ trong bộ nhớ đầu tiên được vận hành tại trường đại học Manchester: Manchester Mark I

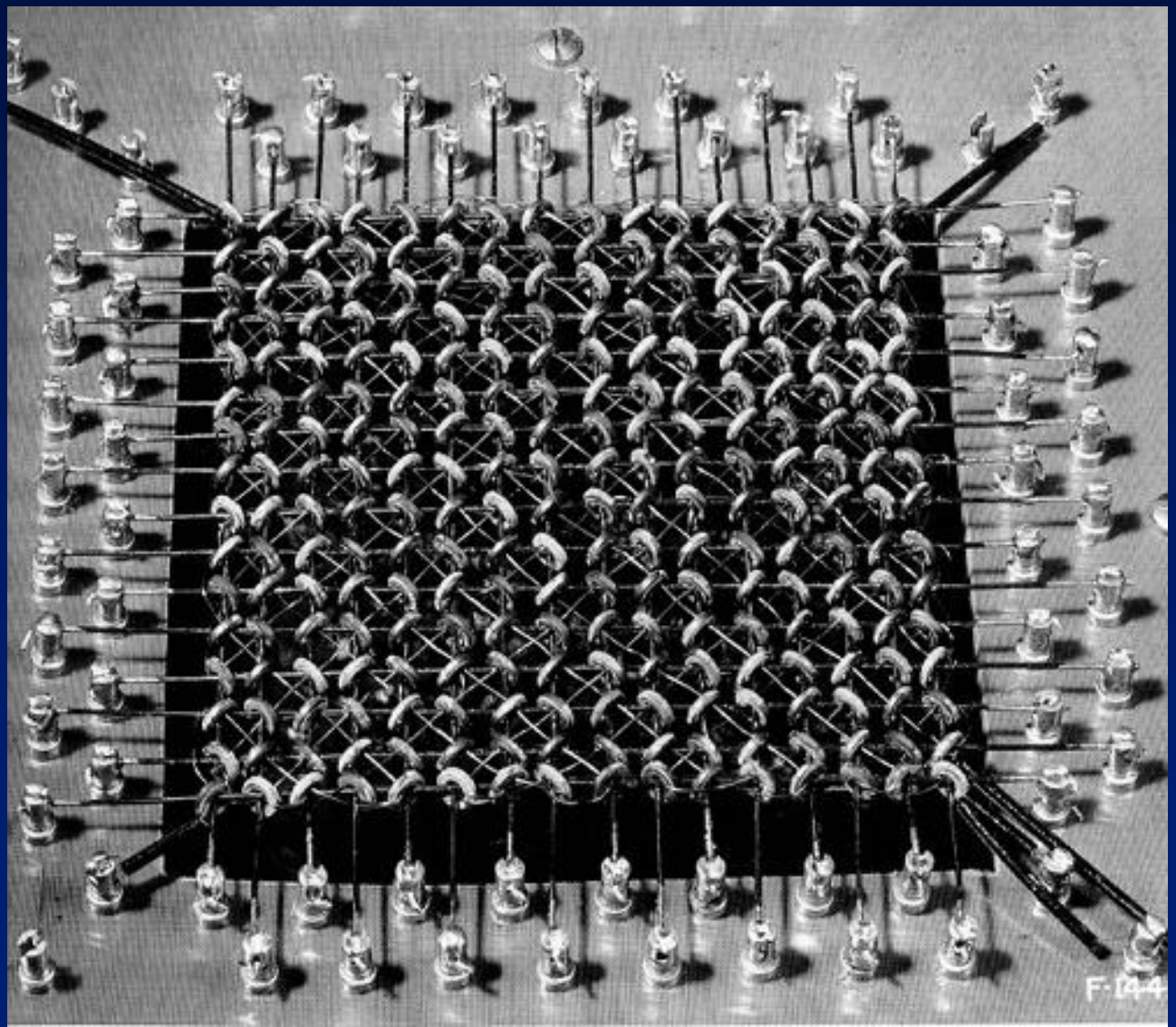


Thế hệ 1: Đèn điện tử (1945-1955)

- Năm 1951, máy tính Whirlwind lần đầu tiên sử dụng bộ nhớ lõi từ (magnetic core memories). Gần đây nguyên lý này đã được sử dụng lại để chế tạo MRAM ở dạng tích hợp.

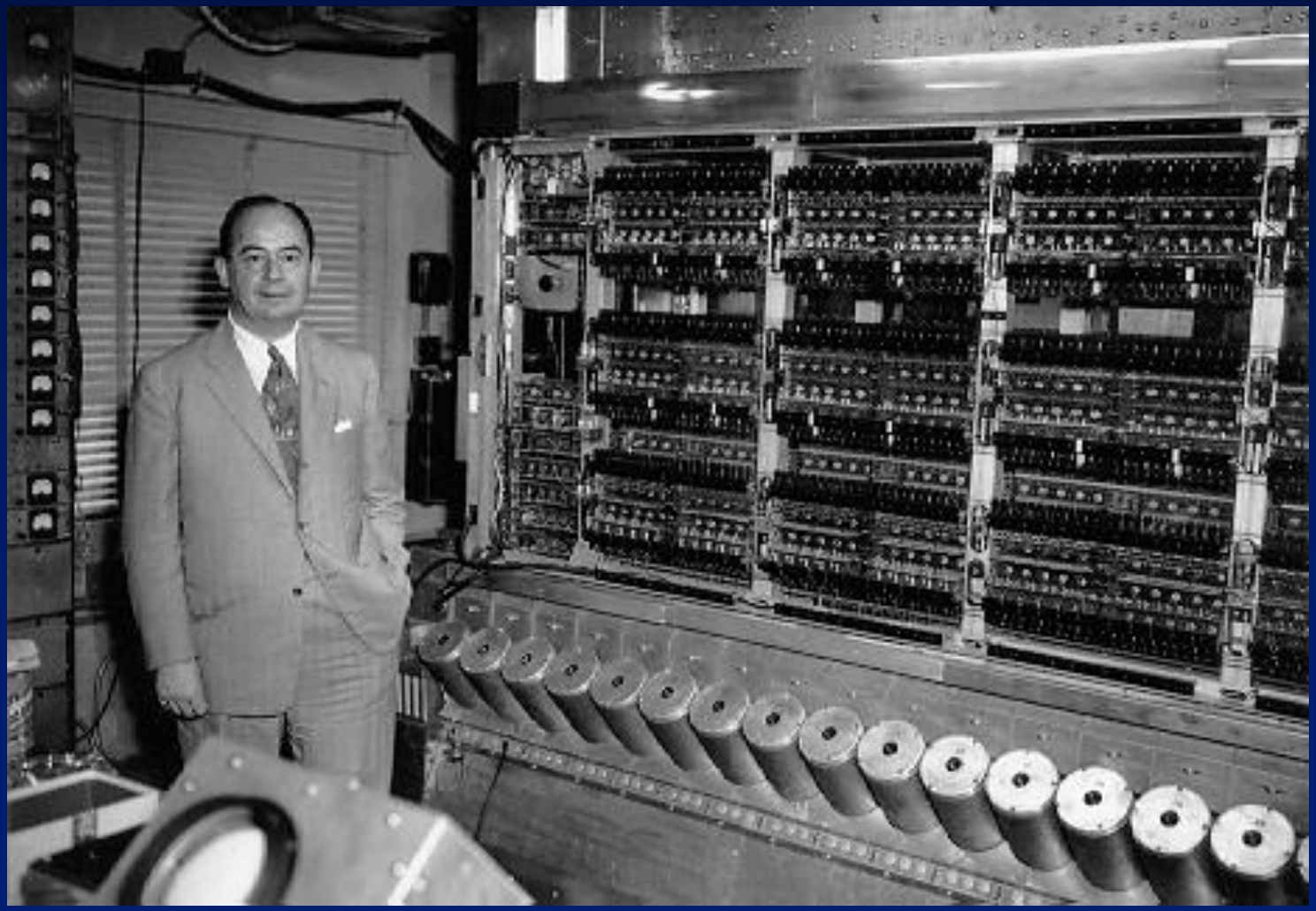
Thế hệ 1: Đèn điện tử (1945-1955)

- Một magnetic core lưu trữ 256 bits



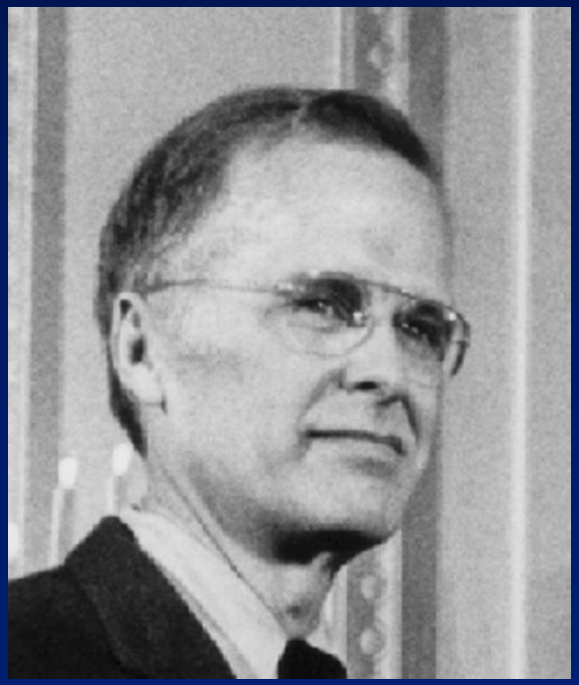
Thế hệ 1: Đèn điện tử (1945-1955)

- John von Neumann năm 1952 với chiếc máy tính mới của ông



Thế hệ 1: Đèn điện tử (1945-1955)

- Năm 1954, John Backus, IBM phát minh ra FORTRAN



Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

- 1.1.1 Thế hệ -1: Thời xa xưa (…-1642)
- 1.1.2 Thế hệ 0: Máy tính cơ khí (1642-1945)
- 1.1.3 Thế hệ 1: Đèn điện tử-Vacuum tubes (1945-1955)
- **1.1.4 Thế hệ 2: Transistor rời rạc-Discrete transistors (1955-1965)**
- 1.1.5 Thế hệ 3: Mạch tích hợp-Integrated circuits (1965-1980)
- 1.1.6 Thế hệ 4: Mạch tích hợp cỡ lớn-VLSI (1980-?)

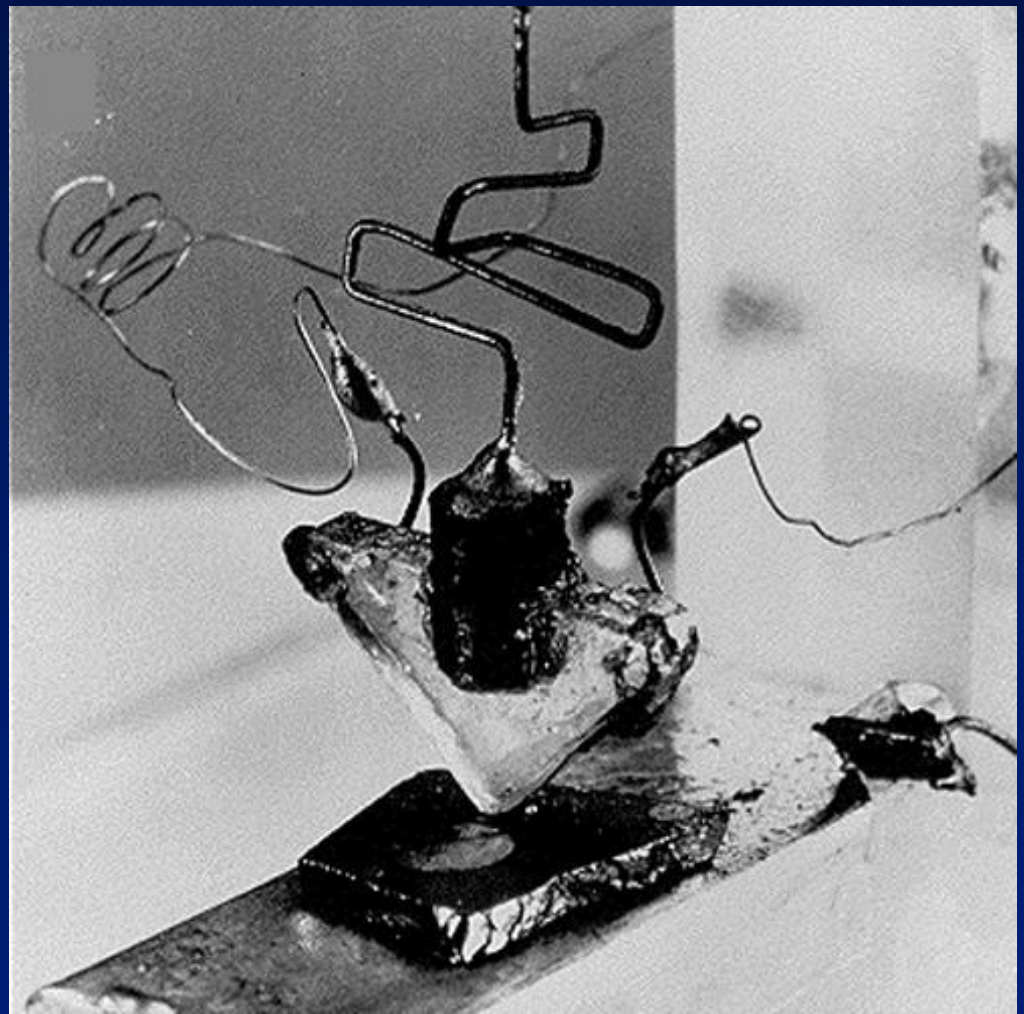
1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

Thế hệ 2: Discrete transistors (1955-1965)

- Năm 1947, William Shockley, John Bardeen, and Walter Brattain phát minh ra transistor



Thế hệ 2: Discrete transistors (1955-1965)

- Năm 1955, IBM công bố IBM704, máy tính mainframe sử dụng tranzistor
- Đây là máy tính với phép toán dấu phẩy động đầu tiên (5 kFlops, clock: 300 kHz)

Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

- 1.1.1 Thế hệ -1: Thời xa xưa (…-1642)
- 1.1.2 Thế hệ 0: Máy tính cơ khí (1642-1945)
- 1.1.3 Thế hệ 1: Đèn điện tử-Vacuum tubes (1945-1955)
- 1.1.4 Thế hệ 2: Transistor rời rạc-Discrete transistors (1955-1965)
- **1.1.5 Thế hệ 3: Mạch tích hợp-Integrated circuits (1965-1980)**
- 1.1.6 Thế hệ 4: Mạch tích hợp cỡ lớn-VLSI (1980-?)

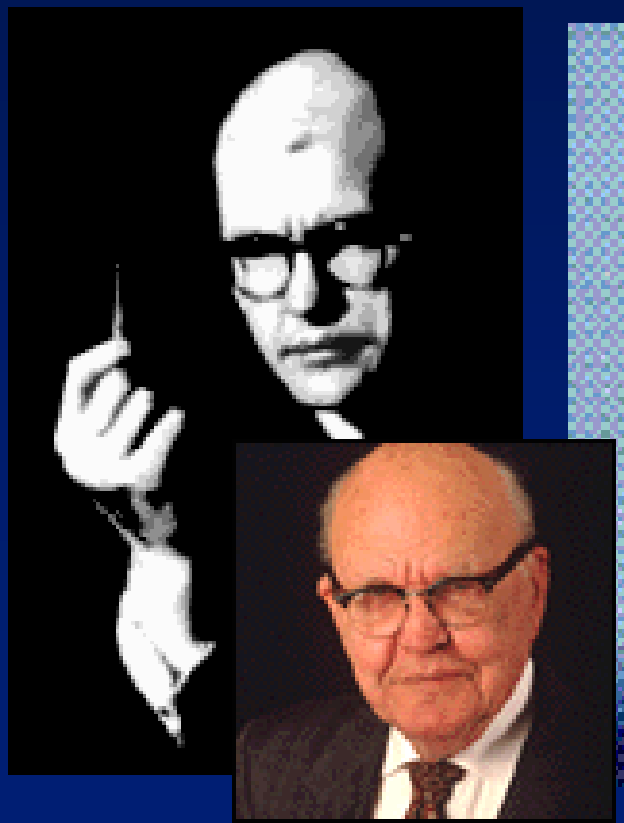
1.2 Phân loại vi xử lý

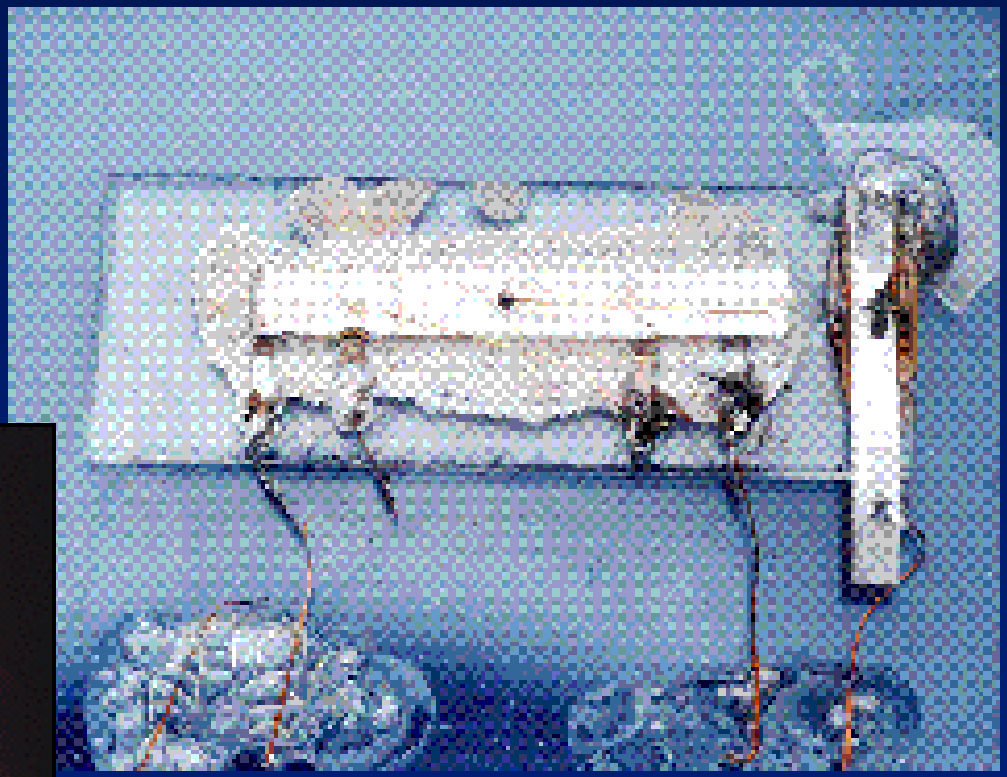
1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

Thế hệ 3: Integrated circuits (1965-1980)

- Năm 1958, Jack St. Clair Kilby of Texas Instruments (Nobel prize physics, 2000) đưa ra và chứng minh ý tưởng tích hợp 1 transistor với các điện trở và tụ điện trên một chip bán dẫn với kích thước 1 nửa cái kẹp giấy. Đây chính là IC.

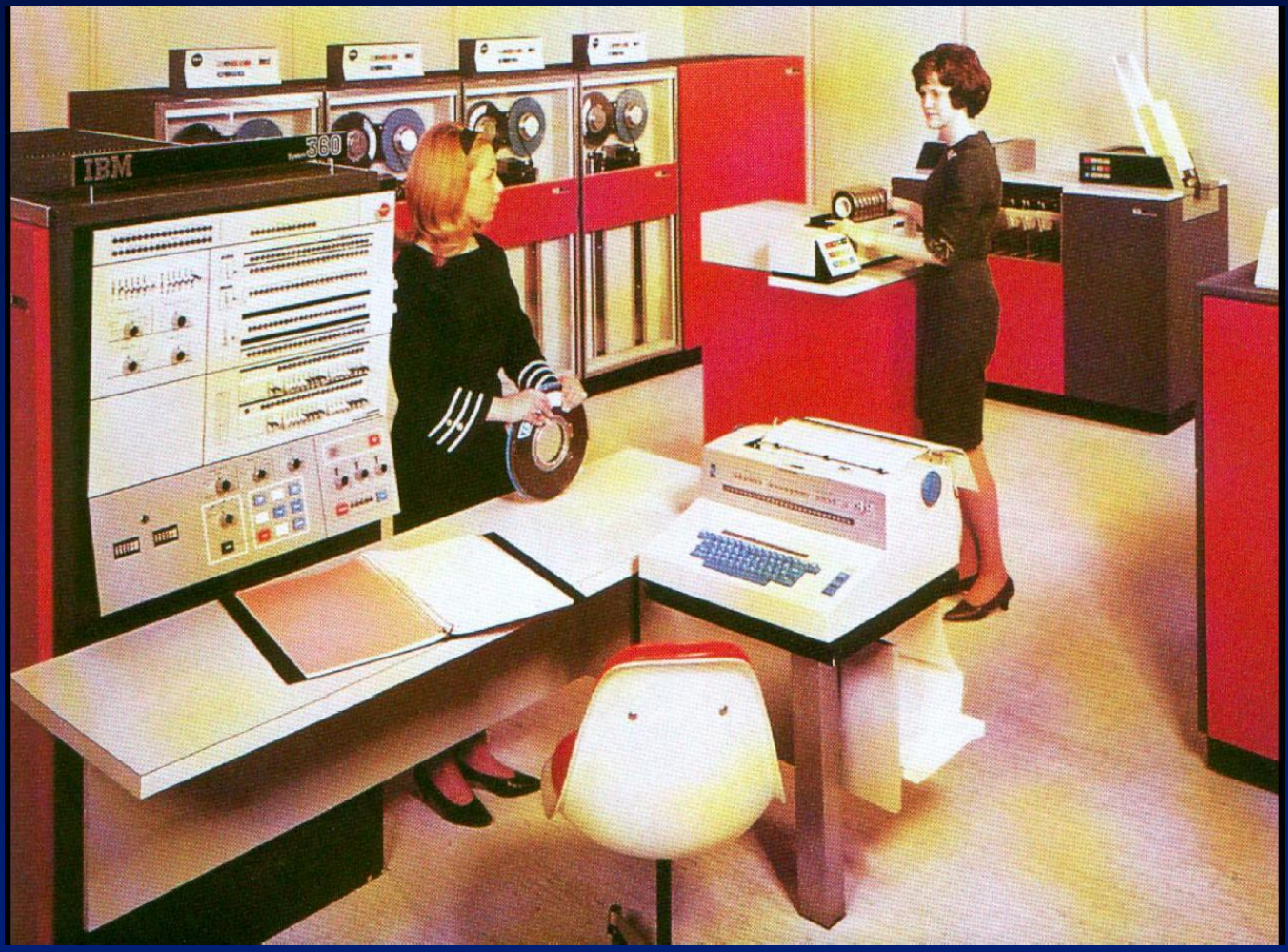




Khoa ĐIỆN TỬ - VIỄN THÔNG

Thế hệ 3: Mạch tích hợp (1965-1980)

- 7/4/1964 IBM đưa ra System/360, họ máy tính tương thích đầu tiên của IBM

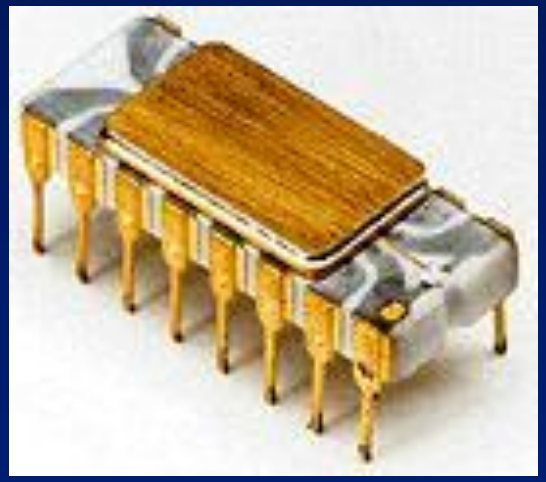


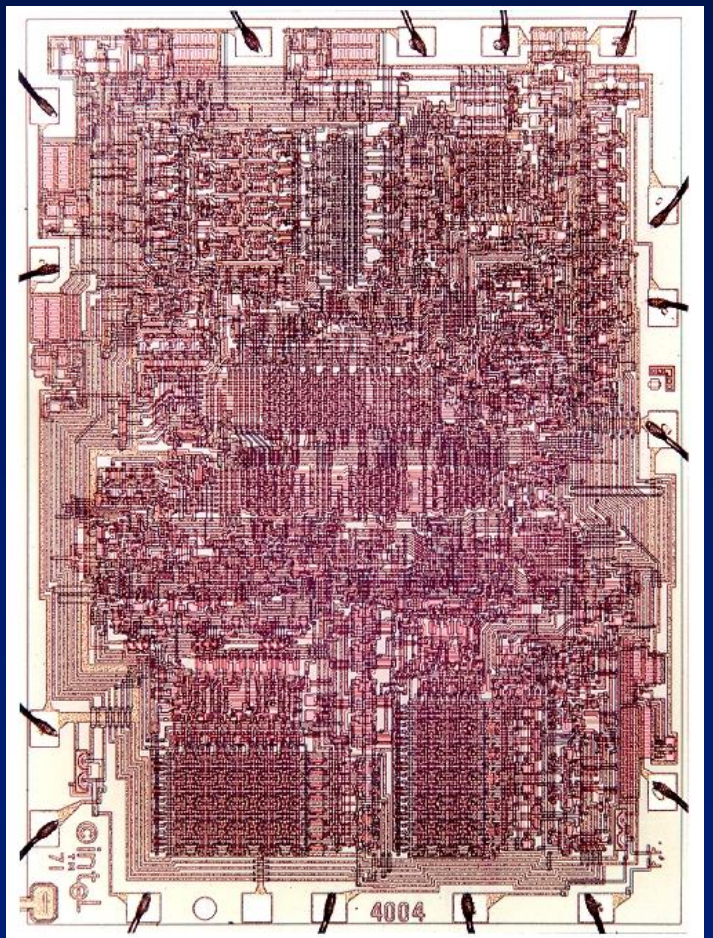
Thế hệ 3: Mạch tích hợp (1965-1980)

- Năm 1965, Digital Equipment Corporation, đưa ra chiếc máy tính mini đầu tiên DP-8

Thế hệ 3: Mạch tích hợp (1965-1980)

- Năm 1971, Ted Hoff chế tạo Intel 4004 theo đơn đặt hàng của một công ty Nhật bản để tạo chip sản xuất calculator. Đây là vi xử lý đầu tiên với 2400 transistor (microprocessor, processor-on-a-chip).
- 4 bít dữ liệu, 12 bit địa chỉ





Thế hệ 3: Mạch tích hợp (1965-1980)

- 1973-1974, Edward Roberts, William Yates and Jim Bybee chế tạo MITS Altair 8800, máy tính cá nhân đầu tiên
- Giá $375, 256 bytes of memory, không keyboard, không màn hình và không bộ nhớ ngoài
- Sau đó, Bill Gate và Paul Allen viết chương trình dịch BASIC cho Altair

Khoa ĐIỆN TỬ - VIỄN THÔNG

Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

- 1.1.1 Thế hệ -1: Thời xa xưa (…-1642)
- 1.1.2 Thế hệ 0: Máy tính cơ khí (1642-1945)
- 1.1.3 Thế hệ 1: Đèn điện tử-Vacuum tubes (1945-1955)
- 1.1.4 Thế hệ 2: Transistor rời rạc-Discrete transistors (1955-1965)
- 1.1.5 Thế hệ 3: Mạch tích hợp-Integrated circuits (1965-1980)
- **1.1.6 Thế hệ 4: Mạch tích hợp cỡ lớn-VLSI (1980-?)**

1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

Thế hệ 4: VLSI (1980-?)

- Năm 1981, IBM bắt đầu với IBM "PC" sử dụng hệ điều hành DOS.

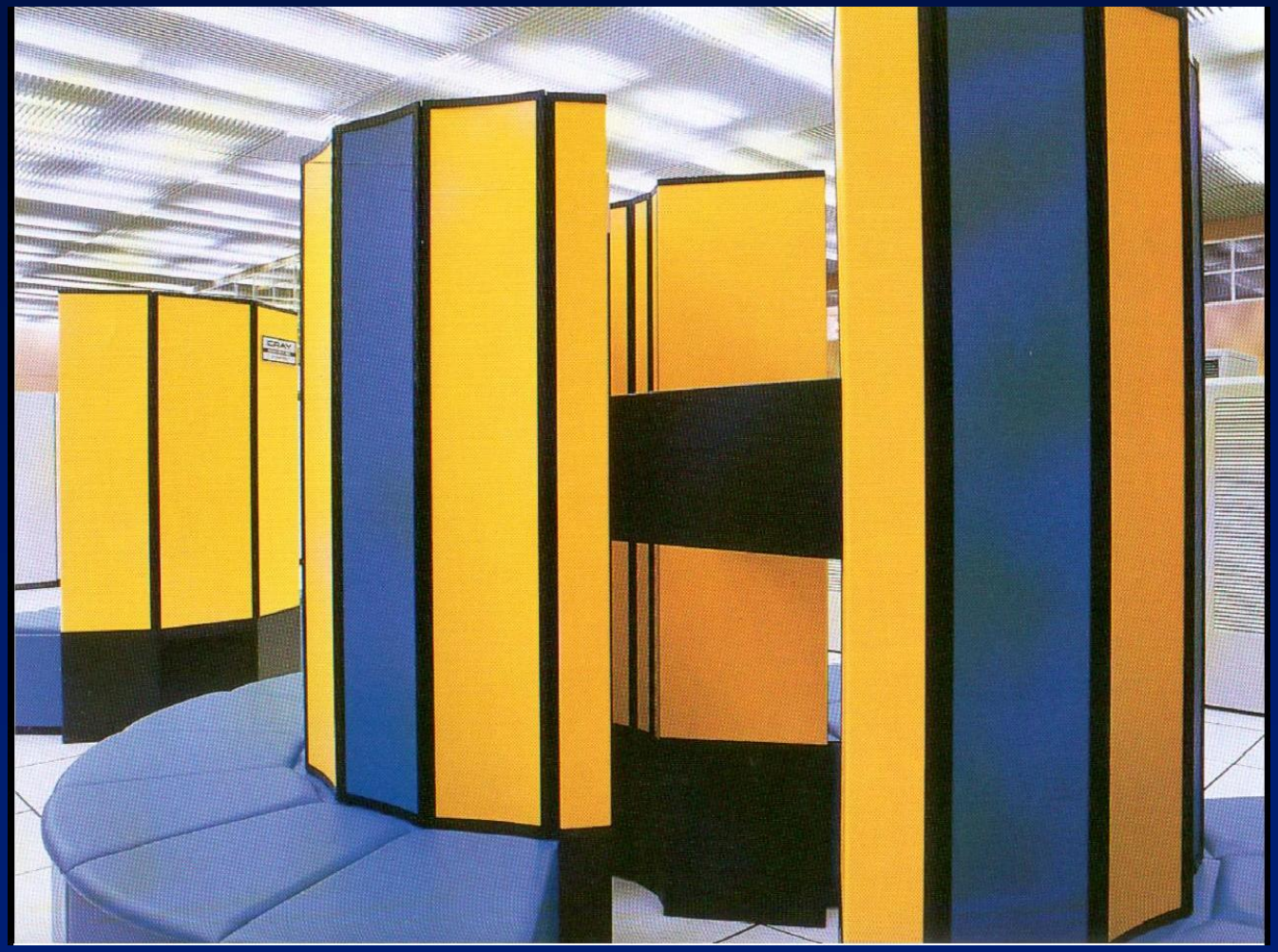
Thế hệ 4: VLSI (1980-?)

- Năm 1984, Xerox PARC (Palo Alto Research Center) đưa ra máy tính để bàn Alto với giao diện người và máy hoàn toàn mới: windows, biểu tượng, mouse

Con chuột đầu tiên

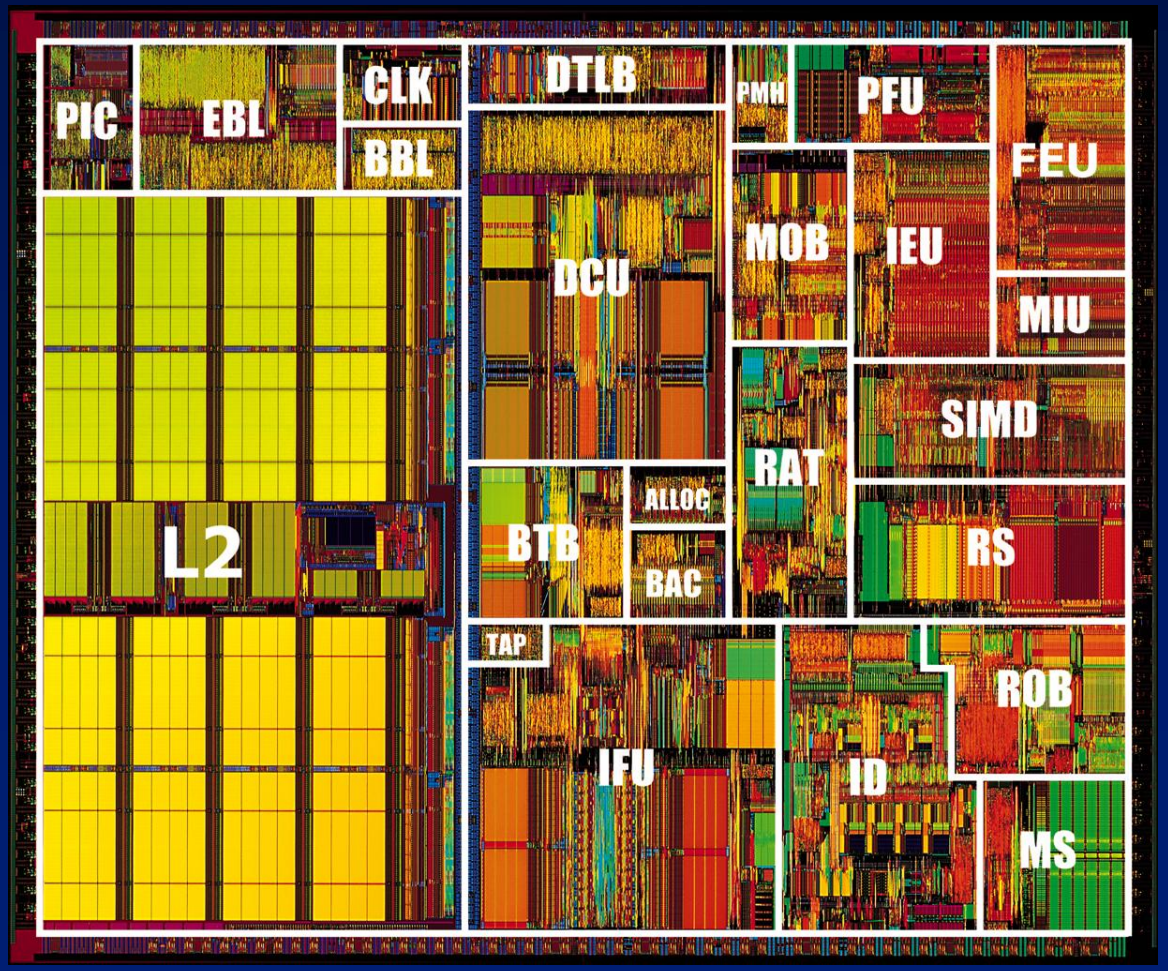
Thế hệ 4: VLSI (1980-?)

- Năm 1986, siêu máy tính Cray-XMP với 4 bộ xử lý đã đạt tốc độ tính toán là 840 MFlops. Nó được làm mát bằng nước

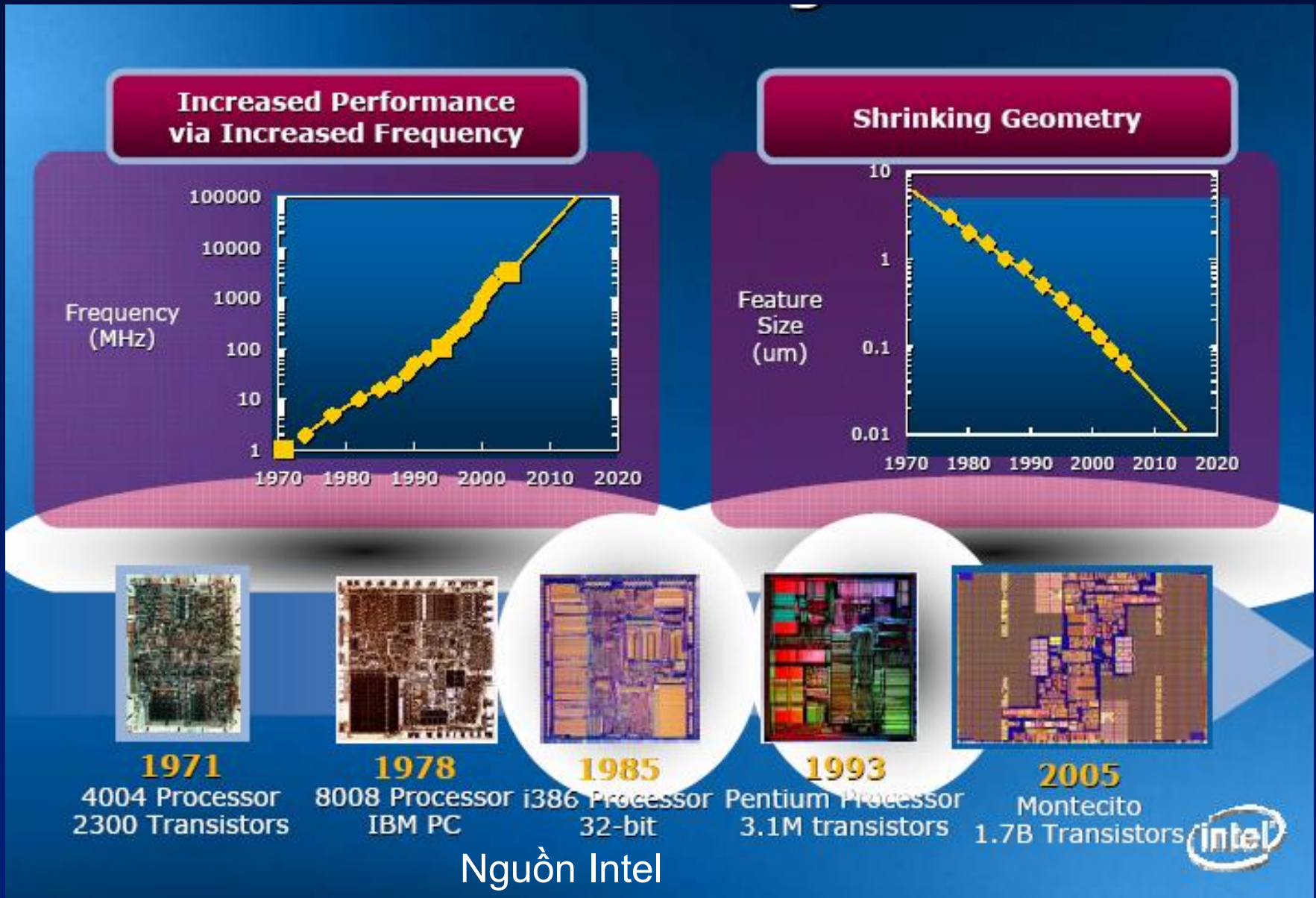


Thế hệ 4: VLSI (1980-?)

- Tốc độ tính toán này đã đạt được với máy tính cá nhân 1 vi xử lý, Pentium III, vào quý 1 năm 2000

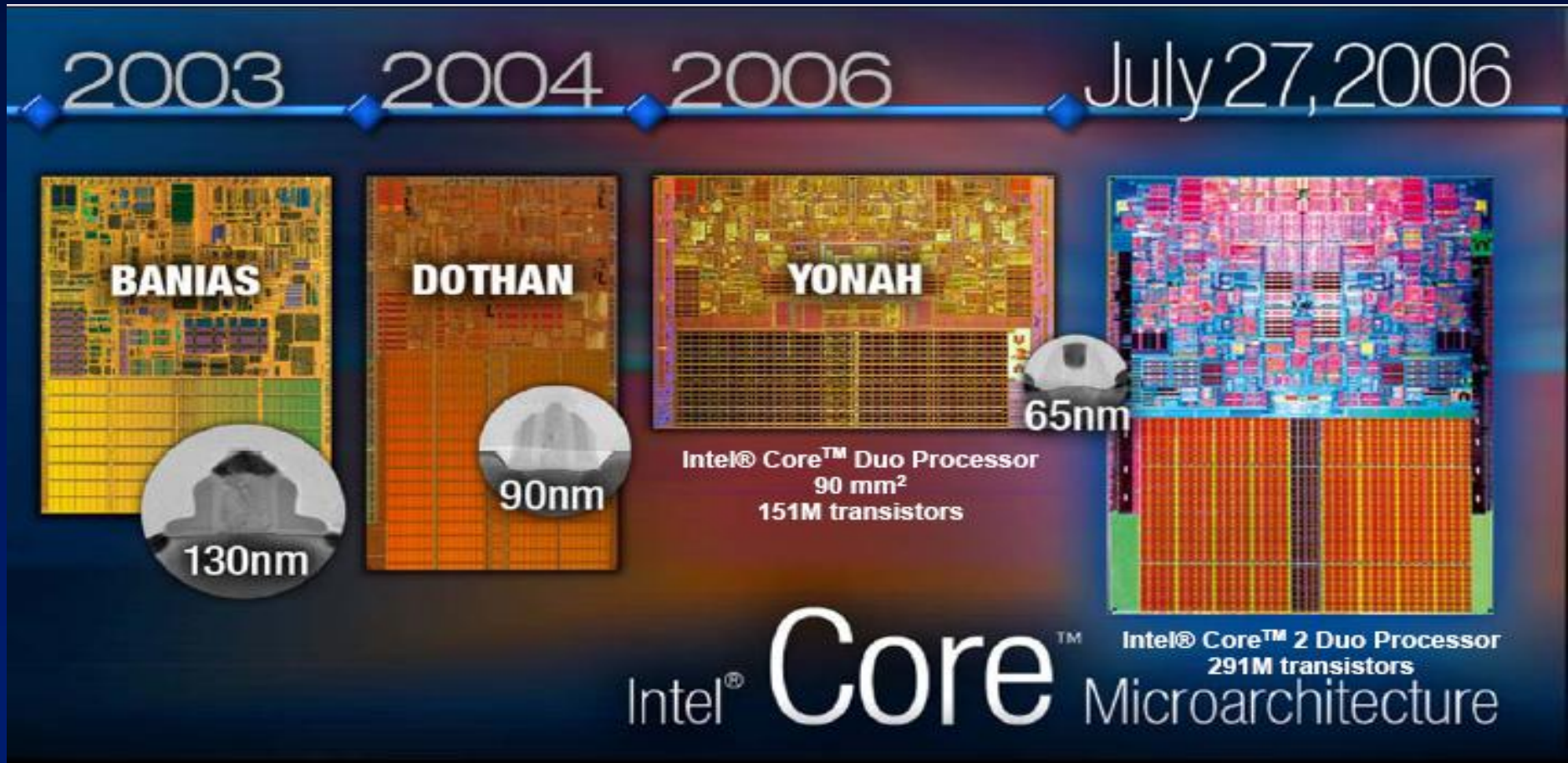


Thế hệ 4: VLSI (1980-?)



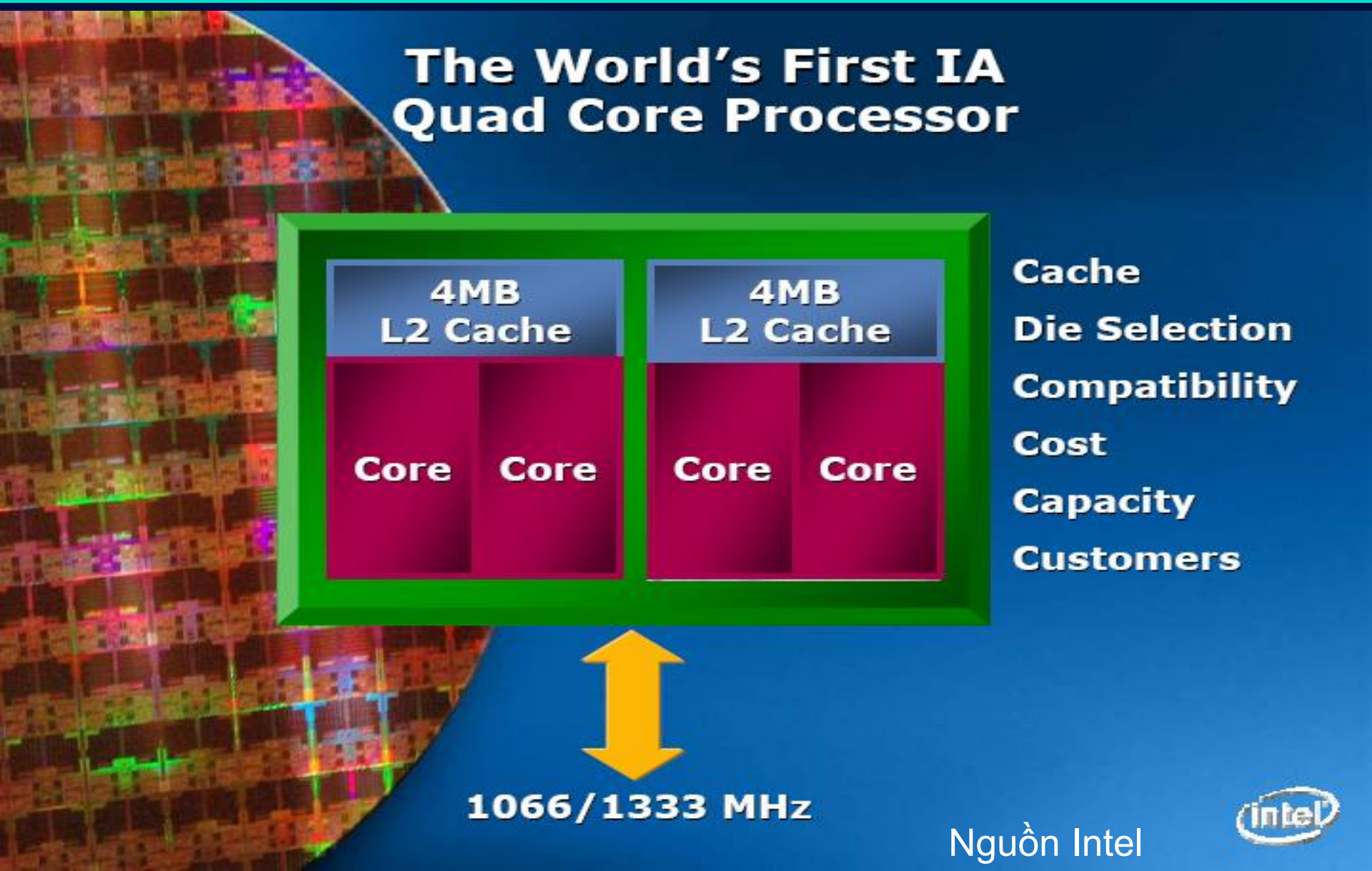
Nguồn Intel

Thế hệ 4: VLSI (1980-?)



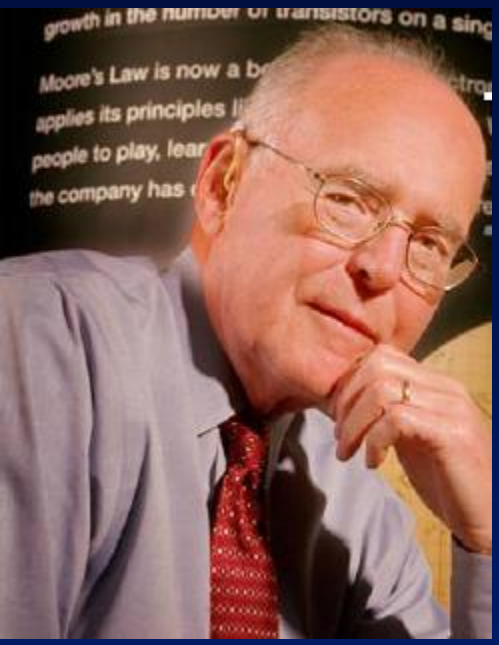
Nguồn Intel

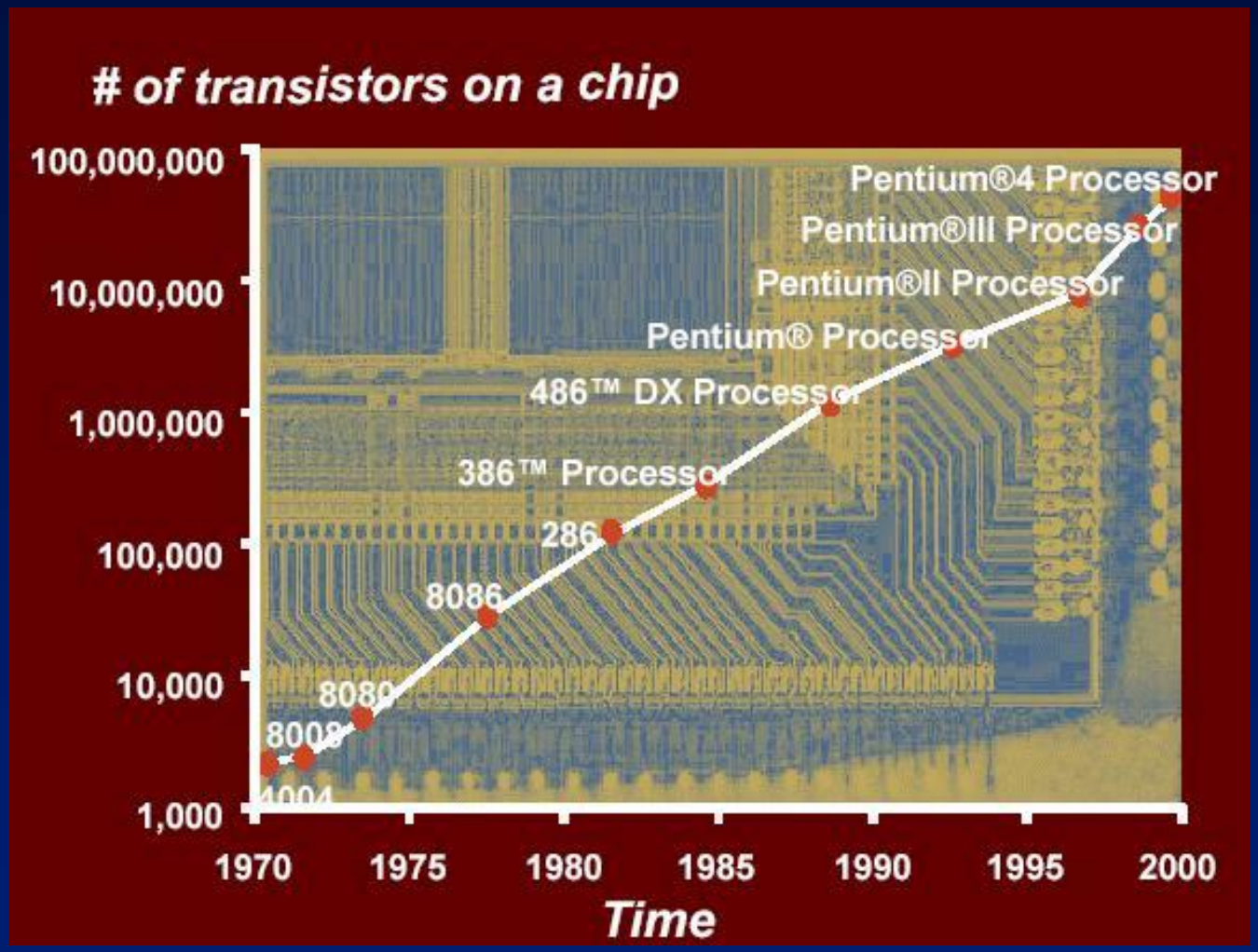
Thế hệ 4: VLSI (1980-?)



Nguồn Intel

Thế hệ 4: VLSI (1980-?)





Moore's law: number of transistors doubles every 18 months
(Gordon Moore, founder Intel Corp.)

Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

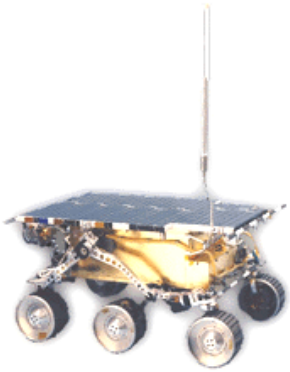
1.2 Phân loại vi xử lý





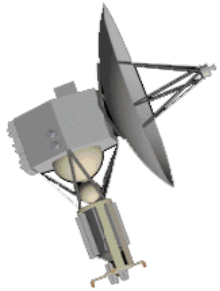




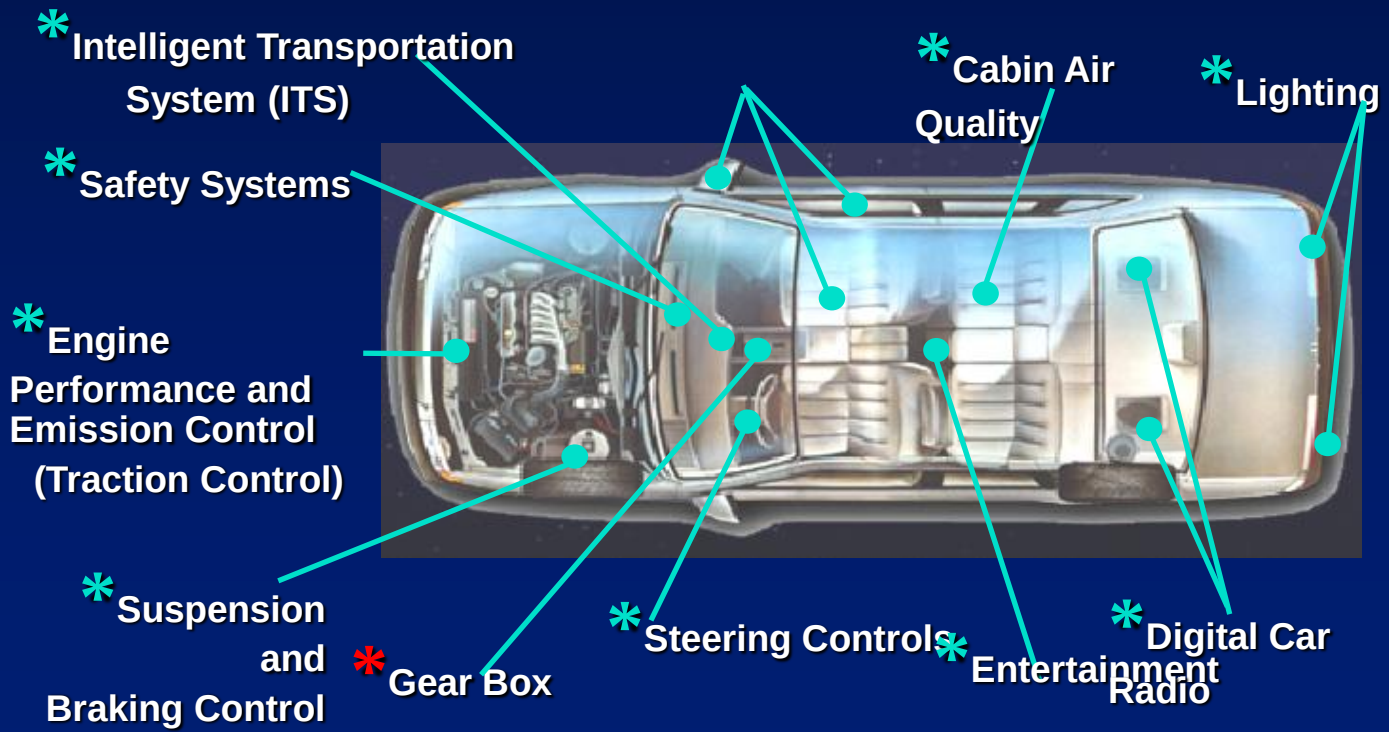






1.2 Phân loại vi xử lý

- **BMW > 100 processors**
- **Trung bình 1 công dân Mỹ ~ 75 processors**



1.2 Phân loại vi xử lý

Phân loại theo giá thành:

Type	Giá (USD)	Example application
Disposable system	1	Greeting cards
Embedded system	10	Watches, cars, appliances
Game computer	100	Home video games
Personal computer	1K	Desktop computer
Server	10K	Network server
Collection of workstations	100K	Departmental supercomputer
Mainframe	1M	Batch processing in bank
Supercomputer	10M	Weather forecasting

1.2 Phân loại vi xử lý

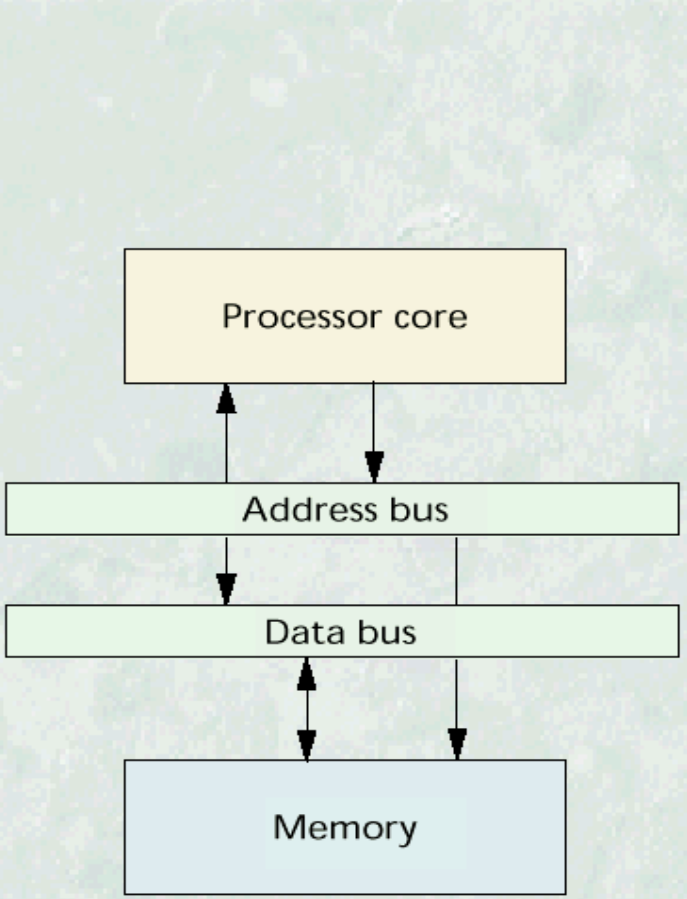
- **Phân loại theo chức năng:**
  - Vi xử lý đa năng (General Purpose Microprocessor)
  - DSP (Digital Signal Processor)
  - Vi điều khiển (Microcontroller)
  - Vi xử lý chuyên dụng ASIP (Application Specific Integrated Processor)

1.2 Phân loại vi xử lý

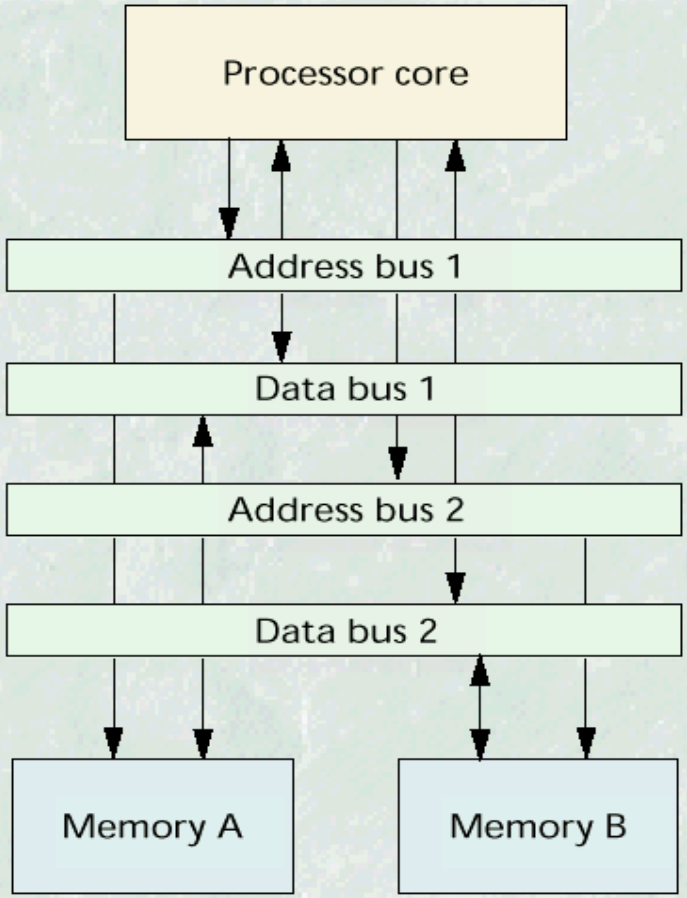
- **Phân loại theo tập lệnh:**
  - ❑ CISC (complex Instruction Set computer): máy tính có tập lệnh phức tạp
    - ⇨ nhiều lệnh
    - ⇨ cấu trúc phức tạp
    - ⇨ mỗi lệnh: có độ dài khác nhau và thực hiện trong 1 đến chục chu kỳ xung nhịp
    - ⇨ Ví dụ: Intel x86, AMD
  - ❑ RISC (reduced instruction Set computer): máy tính có tập lệnh rút gọn
    - ⇨ ít lệnh
    - ⇨ mỗi lệnh có độ dài cố định và thực hiện trong 1 đến 2 chu ký xung nhịp
    - ⇨ cấu trúc vi xử lý đơn giản, có nhiều thanh ghi
    - ⇨ tốc độ xung nhịp lớn và tiêu thụ năng lượng thấp
    - ⇨ Ví dụ: ARM, PowerPC

1.2 Phân loại vi xử lý

Von Neumann vs. Harvard



(a)



(b)

Khoa ĐIỆN TỬ - VIỄN THÔNG

Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

- 1.3.1 Thập phân, Nhị phân, Hệ 8, Hệ 16
- 1.3.2 Cộng, trừ, nhân, chia
- 1.3.3 Các số âm
- 1.3.4 Số nguyên, số thực, BCD, ASCII

1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

Khoa ĐIỆN TỬ - VIỄN THÔNG

Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

- 1.3.1 Thập phân, Nhị phân, Hệ 8, Hệ 16
- 1.3.2 Cộng, trừ, nhân, chia
- 1.3.3 Các số âm
- 1.3.4 Số nguyên, số thực, BCD, ASCII

1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

1.3.1 Các hệ đếm
Hệ thập phân

- $1234{,}567_{10}$=
  - 1•1000+2•100+3•10+4•1+5•0.1+6•0.01+7•0.001
  - $1 \bullet 10^3+2 \bullet 10^2+3 \bullet 10^1+4 \bullet 10^0+5 \bullet 10^{-1}+6 \bullet 10^{-2}+7 \bullet 10^{-3}$
  - r = cơ số (r = 10), d=digit ($0 \le d \le 9$), m = số chữ số trước dấu phẩy, n = số chữ số sau dấu phẩy

$$D = \sum_{i=-n}^{m-1} d_i \bullet r^i$$

1.3.1 Các hệ đếm
Hệ nhị phân

- $1011,011_2$=
  - 1•8+0•4+1•2+1•1+0•0.5+1•0.25+1•0.125
  - $1 \cdot 2^3+0 \cdot 2^2+1 \cdot 2^1+1 \cdot 2^0+0 \cdot 2^{-1}+1 \cdot 2^{-2}+1 \cdot 2^{-3}$
  - r = cơ số (r = 2), d=digit ($0 \le d \le 1$), m = số chữ số trước dấu phẩy, n = số chữ số sau dấu phẩy

$$B = \sum_{i=-n}^{m-1} d_i \bullet 2^i$$

1.3.1 Các hệ đếm
Hệ 8 (Octal)

- $7654,32_8$=
  - 7•512+6•64+5•8+4•1+3•0.125+2•0.015625
  - $7 \bullet 8^3+6 \bullet 8^2+5 \bullet 8^1+4 \bullet 8^0+3 \bullet 8^{-1}+2 \bullet 8^{-2}$
  - r = cơ số (r = 8), d=digit ($0 \le d \le 7$), m = số chữ số trước dấu phẩy, n = số chữ số sau dấu phẩy

$$O = \sum_{i=-n}^{m-1} d_i \bullet 8^i$$

1.3.1 Các hệ đếm
Hệ 16 (Hexadecimal)

- $FEDC,76_{16}$=
  - 15•4096+14•256+13•16+12•1+7•1/16+6•1/256
  - $15 \bullet 16^3+14 \bullet 16^2+13 \bullet 16^1+12 \bullet 16^0+7 \bullet 16^{-1}+6 \bullet 16^{-2}$
  - r = cơ số (r = 16), d=digit ($0 \le d \le F$), m = số chữ số trước dấu phẩy, n = số chữ số sau dấu phẩy

$$H = \sum_{i=-n}^{m-1} d_i \bullet 16^i$$

1.3.1 Các hệ đếm
Chuyển đổi giữa các hệ đếm

- Chuyển từ hệ thập phân sang nhị phân
  - Quy tắc: lấy số cần đổi chia cho 2 và ghi nhớ phần dư, lấy thương chia tiếp cho 2 và ghi nhớ phần dư. Lặp lại khi thương bằng 0. Đảo ngược thứ tự dãy các số dư sẽ được chứ số của hệ nhị phân cần tìm
  - Ví dụ: Đổi 34 sang hệ nhị phân: 100010
- Chyển từ hệ nhị phân sang hệ 16 và ngược lại
  - 1011 0111B = B7H

Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

- 1.3.1 Thập phân, Nhị phân, Hệ 8, Hệ 16
- 1.3.2 Cộng, trừ, nhân, chia
- 1.3.3 Các số âm
- 1.3.4 Số nguyên, số thực, BCD, ASCII

1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

1.3.2 Các phép toán
Cộng nhị phân

- **Cộng thập phân**

Nhớ	0 1 0
x	8 2 7 3
y	5 6 2
Tổng	8 8 3 5

- **Cộng nhị phân**

Nhớ	0 0 1 1 1 1 1
x	1 0 0 1 1 0 1 1
y	1 0 1 0 1 1 1
Tổng	1 1 1 1 0 0 1 0

1.3.2 Các phép toán
Trừ nhị phân

x	1 1 1 0 1
y	1 1 1 1
Mượn	1 1 1 0
Hiệu	0 1 1 1 0

1.3.2 Các phép toán
Nhân nhị phân

- Nguyên tắc: cộng và dịch

```
      1 1 1 0
      1 1 0 1
    -------------
      1 1 1 0
    0 0 0 0
  1 1 1 0
1 1 1 0
-----------------
1 0 1 1 0 1 1 0
```

1.3.2 Các phép toán
Chia nhị phân

```
1 0 1 1 1 0 1 0 | 1 1 1 0
  1 1 1 0       -------
  ---------       1 1 0 1
  1 0 0 1 0 1 0
    1 1 1 0
    ---------
      1 0 0 1 0
      0 0 0 0
      ---------
      1 0 0 1 0
        1 1 1 0
        -------
          1 0 0
```

- Nguyên tắc: trừ và dịch

Khoa ĐIỆN TỬ - VIỄN THÔNG

Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

- 1.3.1 Thập phân, Nhị phân, Hệ 8, Hệ 16
- 1.3.2 Cộng, trừ, nhân, chia
- 1.3.3 Các số âm
- 1.3.4 Số nguyên, số thực, BCD, ASCII

1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

Biểu diễn bằng dấu và độ lớn (Sign-Magnitude)

- Một số có dấu bao gồm 2 phần: dấu và độ lớn
- Ví dụ hệ 10: $+123_{10}$ (thông thường '123') và -123_{10}
- Hệ nhị phân: bít dấu là bít MSB; '0' = dương, '1' = âm
- Ví dụ: $01100_2 = +12_{10}$ và $11100_2 = -12_{10}$
- Các số có dấu 8 bít sẽ có giá trị từ -127 đến +127 với 2 số 0: 1000 0000 (-0) và 0000 0000 (+0)

Số bù 2

- Số bù 1 (bù lô gic): đảo bit
  - 1001 => 0110
  - 0100 => 1011
- Số bù 2 (bù số học): số bù 1 +1

- Ví dụ: Tìm số bù 2 của 13

13 = 0000 1101
Số bù 1 của 13 =1111 0010
Cộng thêm 1: 1
Số bù 2 của 13= 1111 0011 (tức là -13)

Số bù 2

- Ví dụ: Tìm số bù 2 của 0

0 = 0000 0000
Số bù 1 của 0 =1111 1111
Cộng thêm 1: 1
Số bù 2 của 0= 0000 0000 (tức là -0)

- Như vậy với số bù 2, số 0 được biểu diễn 1 cách duy nhất
- Số có dấu 8 bít sẽ có giá trị từ -128 đến 127

Khoa ĐIỆN TỬ - VIỄN THÔNG

Số bù 2

Decimal	Số bù 2	Sign-magnitude
-8	1000	-
-7	1001	1111
-6	1010	1110
-5	1011	1101
-4	1100	1100
-3	1101	1011
-2	1110	1010
-1	1111	1001
0	0000	1000 & 0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111

Khoa ĐIỆN TỬ - VIỄN THÔNG

Chương 1
Giới thiệu chung về hệ vi xử lý

1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

- 1.3.1 Thập phân, Nhị phân, Hệ 8, Hệ 16
- 1.3.2 Cộng, trừ, nhân, chia
- 1.3.3 Các số âm
- 1.3.4 Số nguyên, số thực, BCD, ASCII

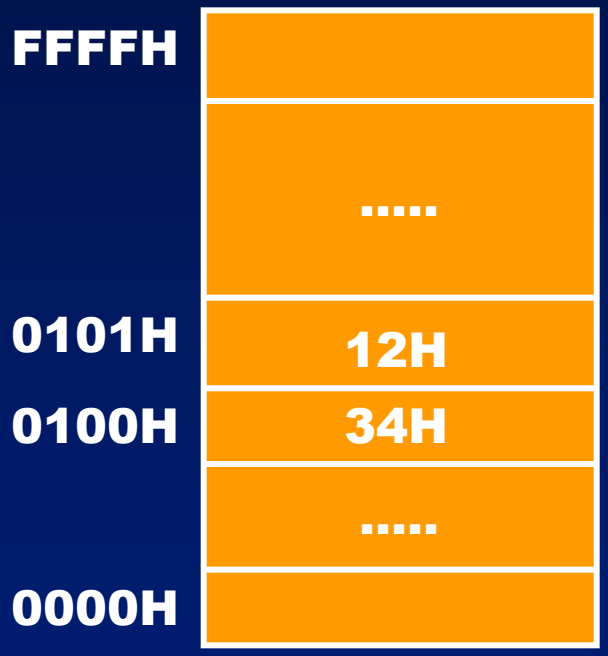
1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

Số nguyên (integer)

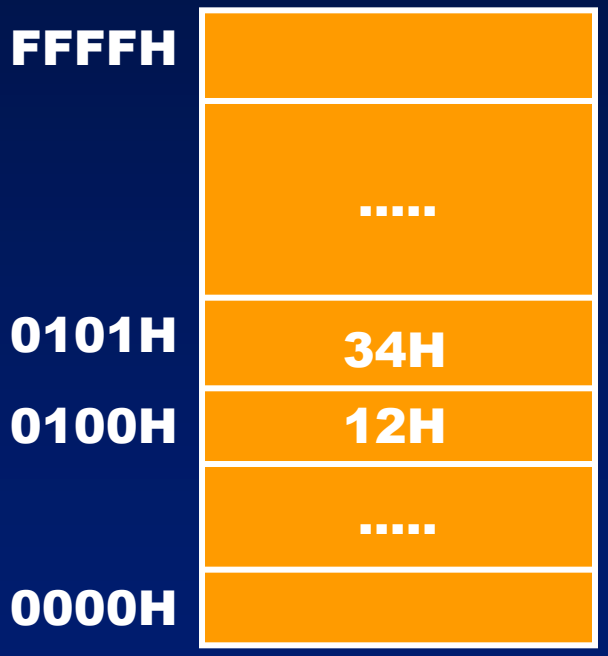
- 8 bit
  - unsigned: 0 đến 255
  - signed : -128 đến 127 (bù hai)
- 16 bit
  - unsigned: 0 đến 65535 (2^{16}-1)
  - signed : -32768 (2^{15}) đến 32767 (2^{15}-1)
- 32 bit
  - unsigned: 0 đến 2^{32}-1
  - signed : -2^{31} đến 2^{31}-1

Little endian và big endian

- Số 1234 H được lưu trữ thế nào trong bộ nhớ 8 bit?



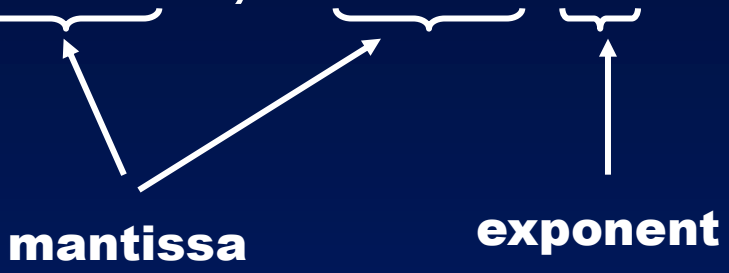
little edian
Intel microprocessors



big edian
Motorola microprocessors

Số thực (real number, floating point number)

- Ví dụ: $1,234=1,234*10^0=0,1234*10^1=...$
- $11,01\ B= 1,101*2^1=0,1101*2^2=...$



- Real number: (m, e) , e.g. (0.1101, 2)
  - ❑ Single precision: 32 bit
  - ❑ Double precision: 64 bit

Số thực (real number, floating point number)

- IEEE-754 format cho single-precision

1 sign bit: 0 dương, 1 âm
8 bit biased exponent= exponent + 127
24 bit mantissa chuẩn hoá = 1 bit ẩn + 23 bit fraction

Mantissa chuẩn hoá: có giá trị giữa 1 và 2 : 1.f

Ví dụ: biểu diễn 0.1011 dưới dạng IEEE-754

Sign bit s=0
chuẩn hoá mantissa: $0.1011=1.011*2^{-1}$
Biased exponent: -1 + 127=126=01111110
IEEE format: 0 01111110 01100000000000000000000

Số thực (real number, floating point number)

- IEEE-754 format cho double-precision

1 sign bit: 0 dương, 1 âm
11 bit biased exponent= exponent + 1023
53 bit mantissa chuẩn hoá = 1 bit ẩn + 52 bit fraction

single precision: $(-1)^s \times 2^{e-127} \times (1.f)_2$

double precision: $(-1)^s \times 2^{e-1023} \times (1.f)_2$

Số thực (real number, floating point number)

	Single Precision	Double Precision
Machine epsilon Độ chính xác	2^{-23} or 1.192×10^{-7}	2^{-52} or 2.220×10^{-16}
Smallest positive Số dương nhỏ nhất	2^{-126} or 1.175×10^{-38}	2^{-1022} or 2.225×10^{-308}
Largest positive Số dương lớn nhất	$(2- 2^{-23}) 2^{127}$ or 3.403×10^{38}	$(2- 2^{-52}) 2^{1023}$ or 1.798×10^{308}
Decimal Precision Độ chính xác thập phân	6 significant digits 6 chữ số sau dấu phảy	15 significant digits 15 chữ số sau dấu phảy

BCD

- **Binary Coded Decimal number**
  - ❑ BCD chuẩn (BCD gói, packed BCD):
    - ⇨ 1 byte biểu diễn 2 số BCD
    - ⇨ Ví dụ: 25: 0010 0101
  - ❑ BCD không gói (unpacked BCD) :
    - ⇨ 1 byte biểu diễn 1 số BCD
    - ⇨ ví dụ: 25: 00000010 00000101

Decimal digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

ASCII

- American Standard Code for Information Interchange (7-bit code)

b3b2b1b0	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	V
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Khoa ĐIỆN TỬ - VIỄN THÔNG

Chương 1
Giới thiệu chung về hệ vi xử lý

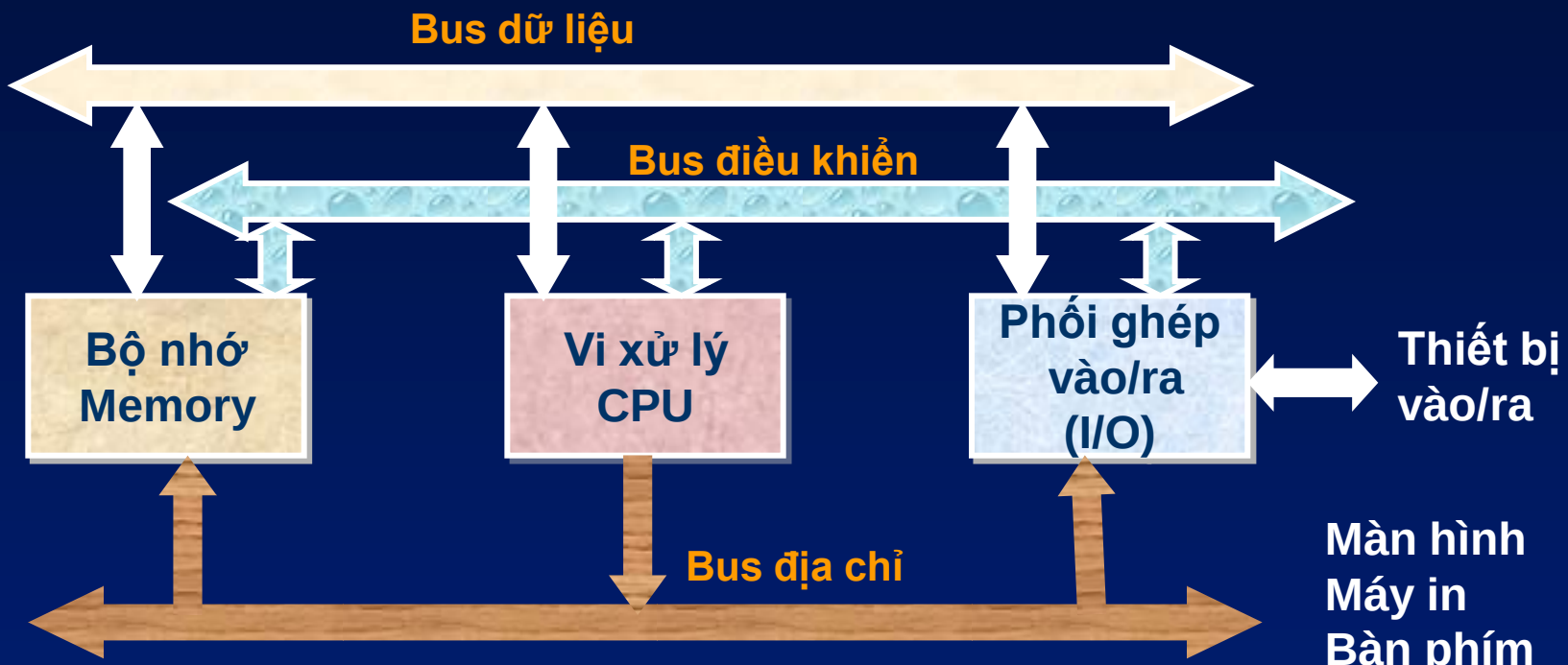
1.1 Lịch sử phát triển của các bộ vi xử lý và máy tính

1.2 Phân loại vi xử lý

1.3 Các hệ đếm dùng trong máy tính (nhắc lại)

1.4 Sơ lược về cấu trúc và hoạt động của hệ vi xử lý

1.4.1 Hệ vi xử lý



DRAM
SRAM
ROM
EEPROM
Flash

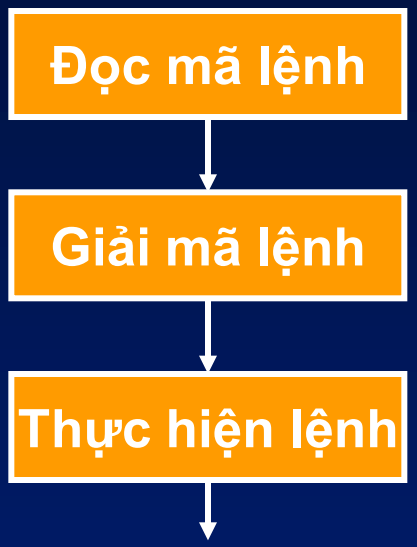
Intel 80X86
Motorola 680X
PowerPC
...

ISA
EISA
PCI
VESA
SCSI
USB
...

Màn hình
Máy in
Bàn phím
Con chuột
Ổ cứng
Ổ mềm
CD-ROM
DVD
...

1.4.1 Hệ vi xử lý

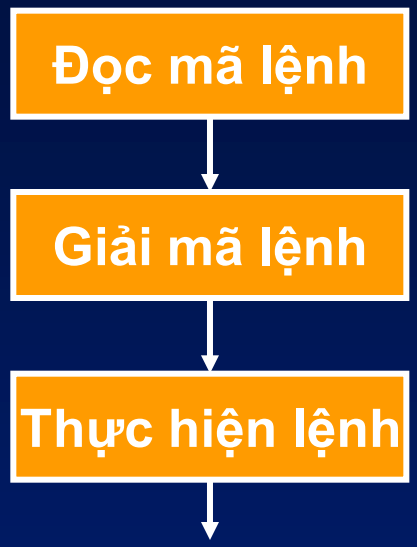
- CPU (Central Processing Unit)
  - ❑ Đơn vị số học và logic (Arithmetic Logical Unit)
    - ⇨Thực hiện các phép toán số học
      - ✓Cộng, trừ, nhân chia
    - ⇨Thực hiện các phép toán logic
      - ✓And, or, compare..
  - ❑ Đơn vị điều khiển (Control Unit)



  - ❑ Các thanh ghi (Registers)
    - ⇨Lưu trữ dữ liệu và trạng thái của quá trình thực hiện lệnh

1.4.1 Hệ vi xử lý

- Đọc thông tin từ bộ nhớ vào CPU
- Xác định xem lệnh đó là lệnh gì
- Thực hiện lệnh
  - ❑ Nếu cần thì đọc thêm thông tin từ bộ nhớ/cổng
  - ❑ Tính toán và ghi thông tin ra bộ nhớ/cổng



1 chu kỳ lệnh của CPU

1.4.2 Bộ nhớ, BUS

- Memory
  - ROM: không bị mất dữ liệu, chứa dữ liệu điều khiển hệ thống lúc khởi động
  - RAM: mất dữ liệu khi mất nguồn, chứa chương trình và dữ liệu trong quá trình hoạt động của hệ thống
- Bus dữ liệu
  - 8, 16, 32, 64 bit tùy thuộc vào vi xử lý
- Bus địa chỉ:
  - 16, 20, 24, 32, 36 bit
  - số ô nhớ có thể đánh địa chỉ: 2^N
  - Ví dụ: 8088/8086 có 20 đường địa chỉ => quản lý được 2^{20} bytes=1Mbytes

Khoa ĐIỆN TỬ - VIỄN THÔNG

1.4.2 Bộ nhớ, BUS

Nhà sản xuất	Tên vi xử lý	Bus dữ liệu	Bus địa chỉ	Khả năng địa chỉ
Intel	8088	8	20	1 M
	8086	16	20	1 M
	80186	16	20	1 M
	80286	16	24	16 M
	80386SX	16	24	16 M
	80386DX	32	32	4 G
	80486DX	32	32	4 G
	Pentium	64	32	4 G
	Pentium Pro	64	36	64 G
	Pentium I, II, III, IV	64	36	64 G
Motorola	68000	16	24	16 M
	68010	16	24	16 M
	68020	32	32	4 G
	68030	32	32	4 G
	68040	32	32	4 G
	68060	64	32	4 G
	PowerPC	64	32	4 G

Kỹ thuật Vi xử lý

Điện tử-Viễn thông

Đại học Bách khoa Đà Nẵng

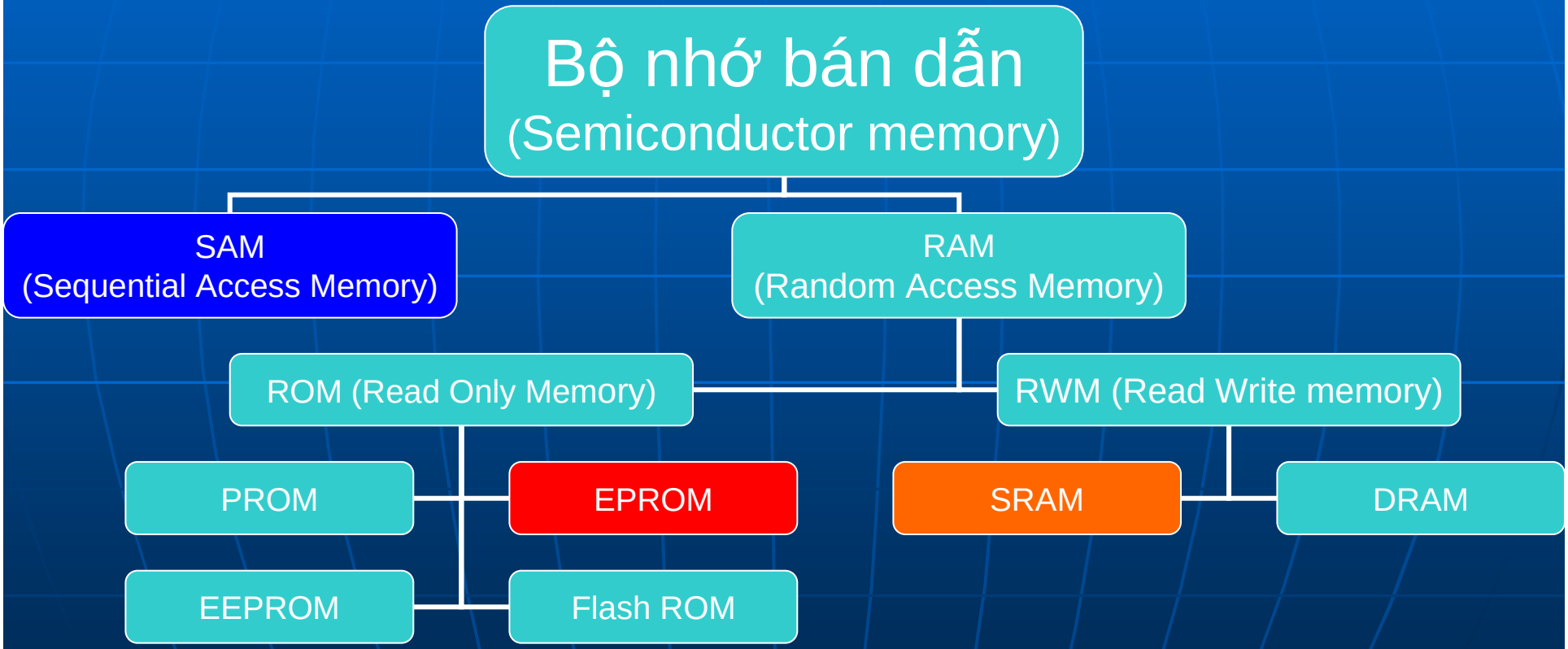
Chương 4

4.1 Phân loại bộ nhớ bán dẫn
4.2 Hoạt động của các chip EPROM
4.3 Hoạt động của các chip SRAM
4.4 Bus hệ thống của hệ vi xử lý 8088
4.5 Bài toán thiết kế bộ nhớ

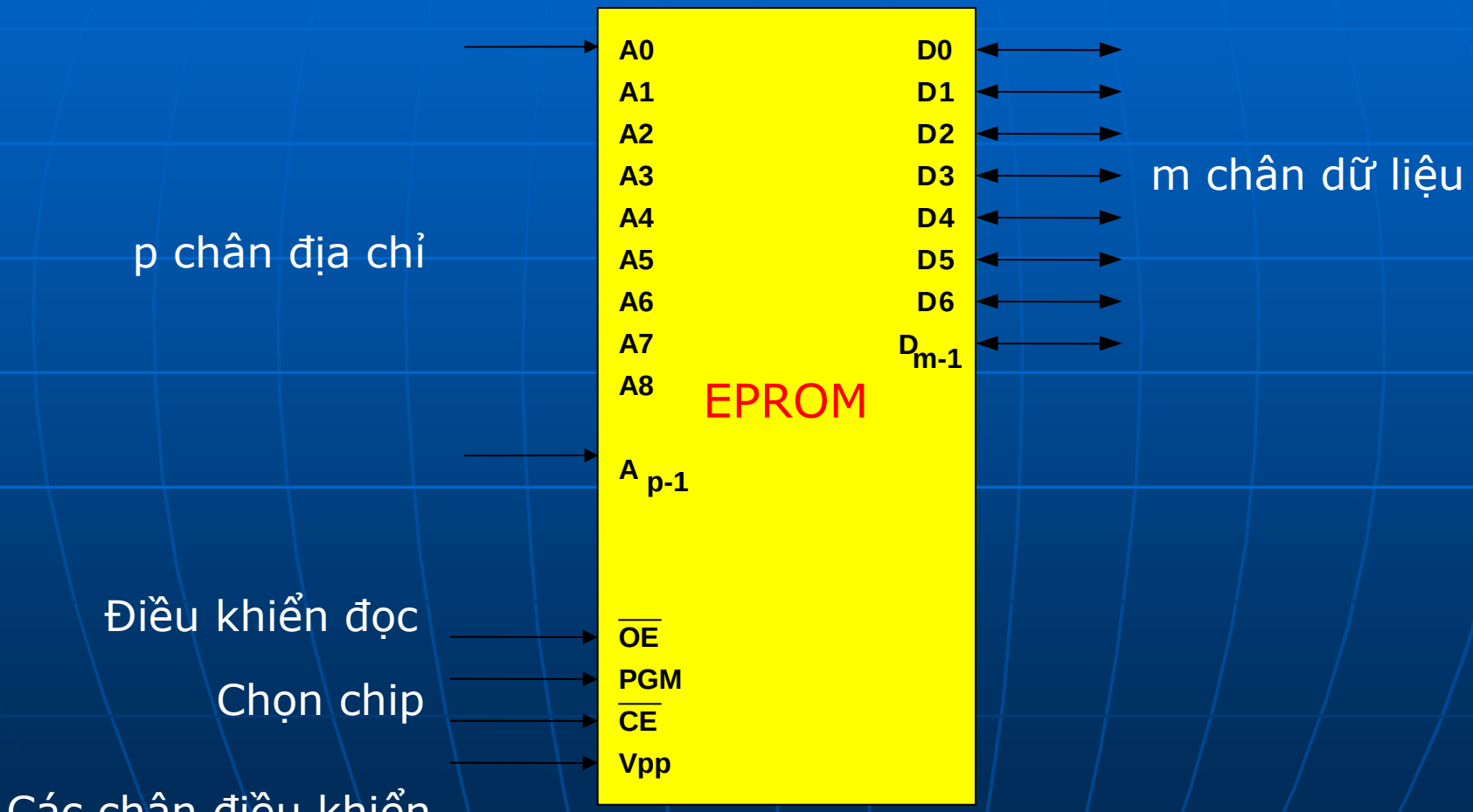
Mục tiêu và biện pháp thiết kế

- Ghép nối các chip nhớ EPROM và SRAM với Bus hệ thống sao cho không xảy ra xung đột:
  Các chip nhớ bị cấm khi vi xử lý truy cập các cổng I/O
  Chỉ có một chip nhớ hoạt động khi vi xử lý truy cập bộ nhớ
- Thực hiện một mạch giải mã địa chỉ bộ nhớ dùng các chip giải mã hoặc các cổng logic hoặc kết hợp cả hai

4.1 Phân loại bộ nhớ bán dẫn



4.2 Các chip EPROM



Dung lượng của 1 chip nhớ

- Một chip nhớ được xem như một mảng gồm n ô nhớ. Mỗi ô nhớ lưu trữ được m-bit dữ liệu
- Dung lượng của chip thường được biểu diễn: nxm
  Ví dụ: Một chip có dung lượng 2Kx8 nghĩa là chip đó có 2048 ô nhớ và mỗi ô nhớ có thể lưu trữ được 1 byte dữ liệu
- m chính là số chân dữ liệu của chip
- $\log_2(n) = p$ là số chân địa chỉ của chip

Hoạt động ghi dữ liệu vào EPROM

- Việc ghi dữ liệu vào EPROM được gọi là lập trình cho EPROM
- Được thực hiện bằng thiết bị chuyên dụng gọi là Bộ nạp EPROM
- Chân Vpp được cấp điện áp tương ứng với từng loại chip gọi là điện áp lập trình
- Dữ liệu tại các chân dữ liệu sẽ được ghi vào một ô nhớ xác định nhờ các tín hiệu đưa vào ở các chân địa chỉ và một xung (thường gọi là xung lập trình) đưa vào chân PGM

Hoạt động đọc dữ liệu từ một chip EPROM

Để đọc dữ liệu từ 1 ô nhớ nào đó của 1 chip EPROM nào đó, Bộ vi xử lý cần phải:

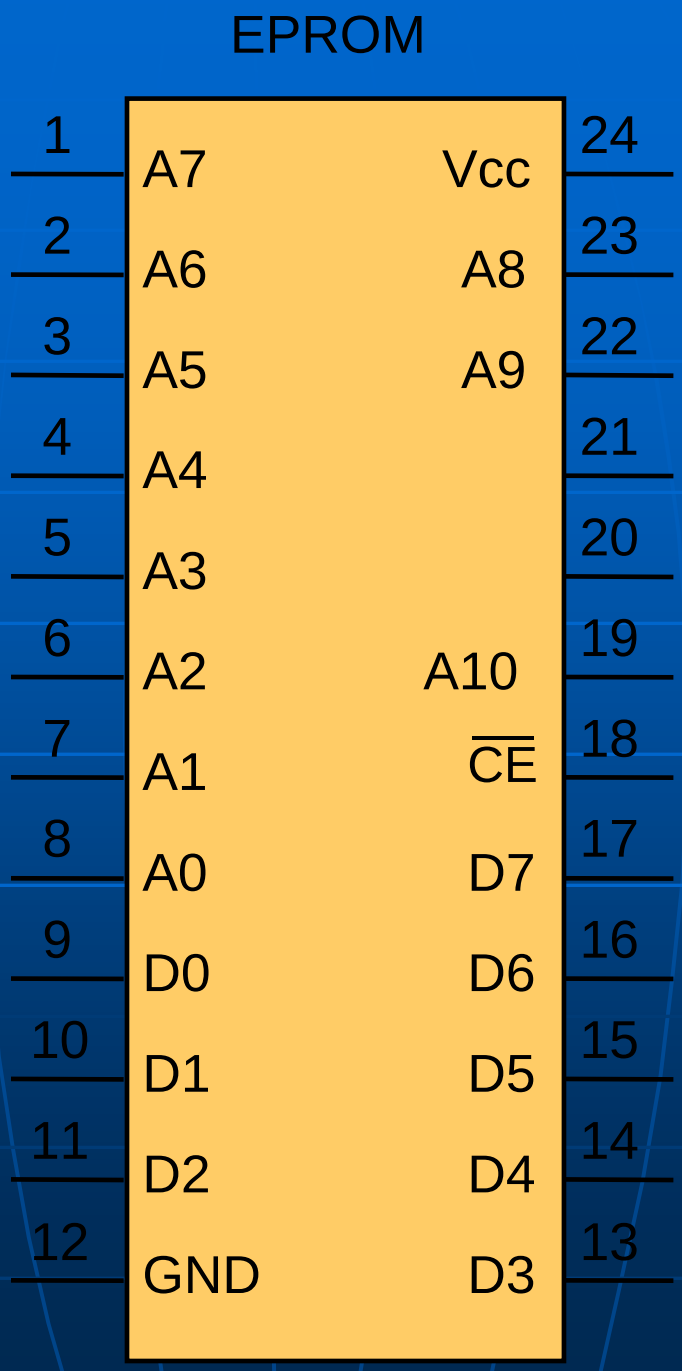
- Chọn chip đó: 0 -----> CE
- Áp các tín hiệu địa chỉ của ô nhớ cần đọc vào các chân địa chỉ $A_{p-1} – A_0$
- Đọc: 0 ------ > OE
- Kết quả là m bit dữ liệu cần đọc xuất hiện ở các chân dữ liệu $D_{m-1} – D_0$

Họ EPROM thông dụng 27x

Số hiệu của chip	Dung lượng
2716	2Kx8
2732	4Kx8
2764	8Kx8
27128	16Kx8
27256	32Kx8
27512	64Kx8

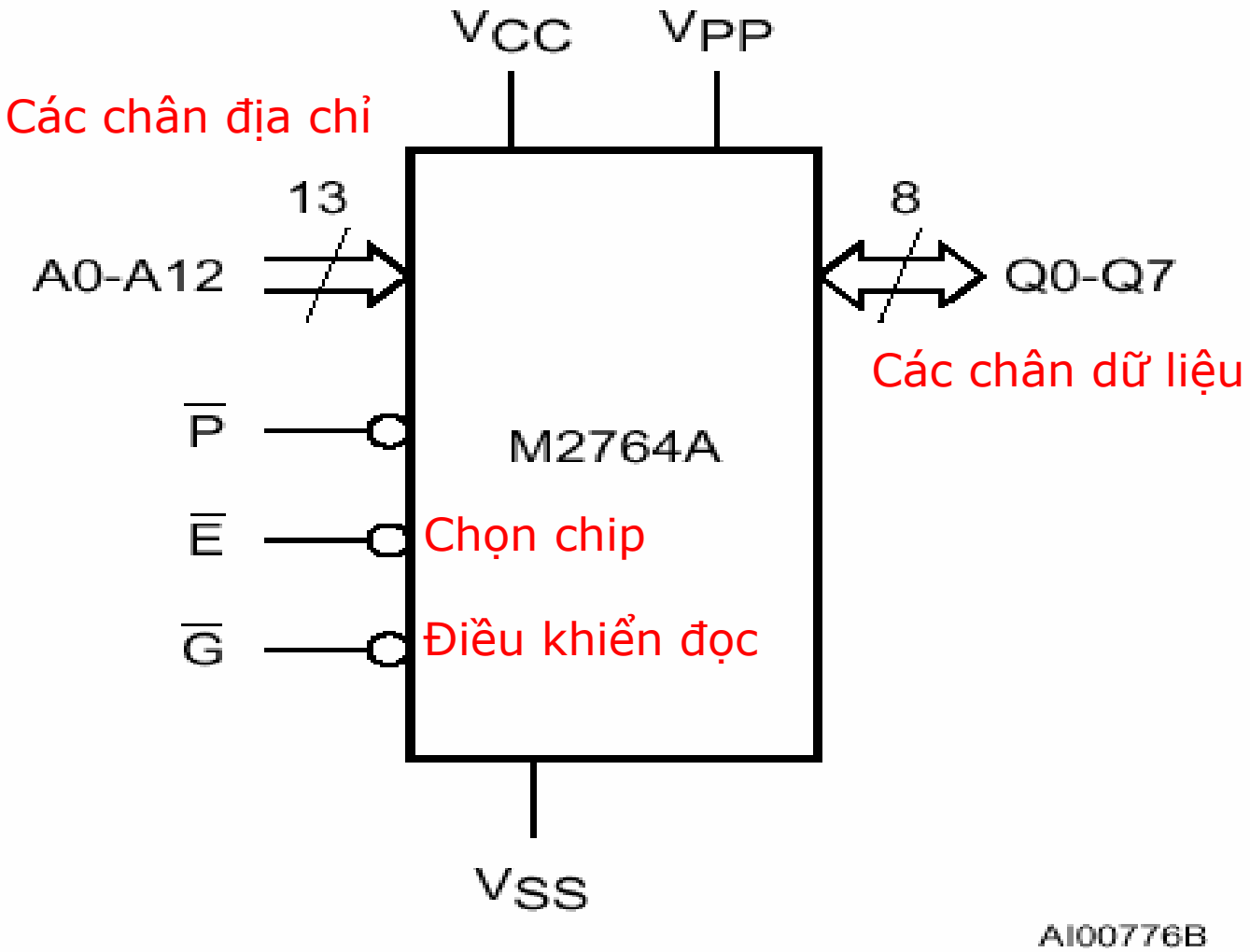
Bảng 4.1 Họ EPROM 27x

- Sơ đồ chân của 2716 và 2732



	2716	**2732**
21	Vpp	A11
20	$\overline{OE}$	$\overline{OE}$ / Vpp

EPROM 2764


VCC
VPP
Các chân địa chỉ
13
A0-A12
8
Q0-Q7
Các chân dữ liệu
P
M2764A
E
Chọn chip
G
Điều khiển đọc
VSS
AI00776B

EPROM 2764

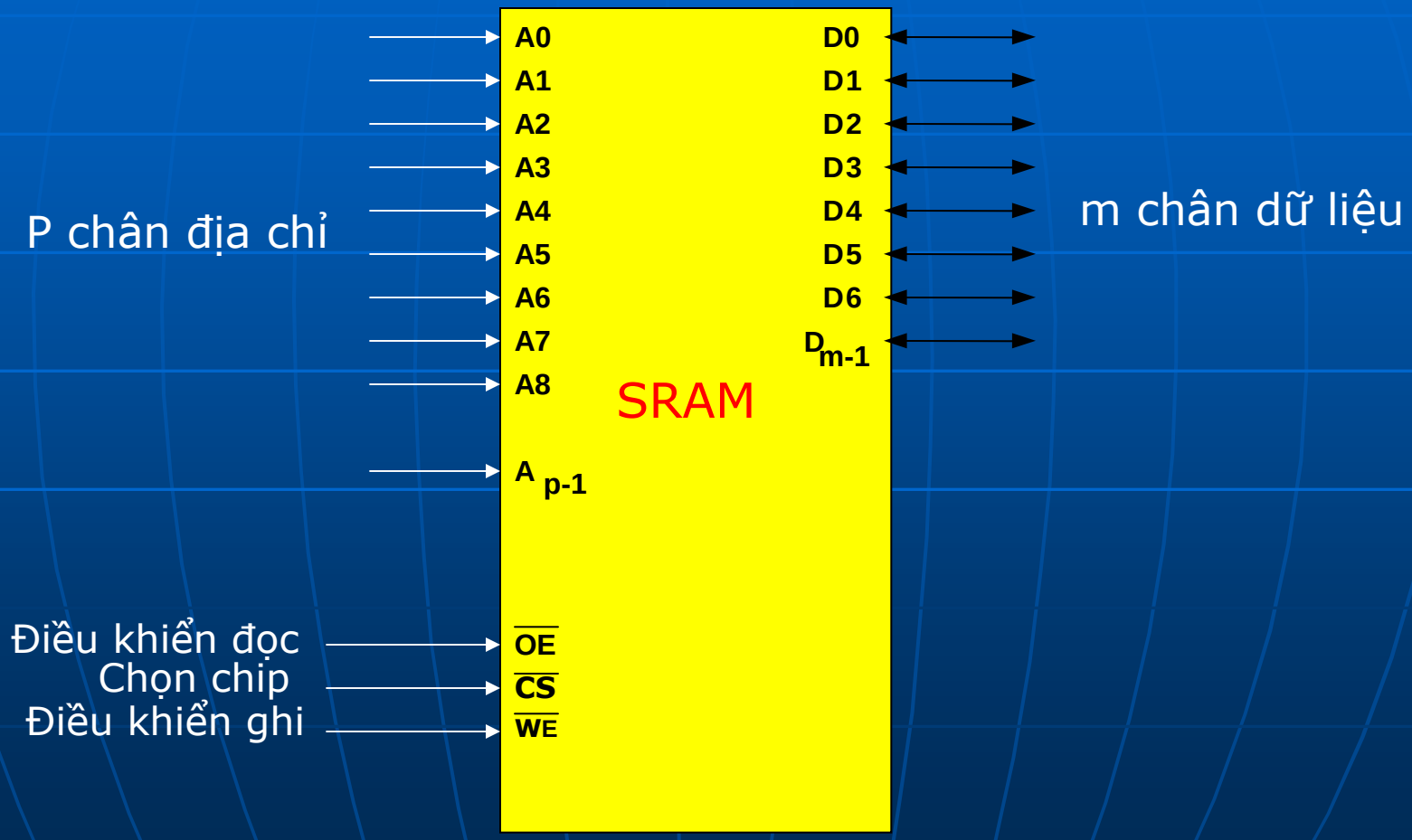
Mode	$\overline{E}$	$\overline{G}$	$\overline{P}$	A9	V_{PP}	Q0 - Q7
Read	V_{IL}	V_{IL}	V_{IH}	X	V_{CC}	Data Out
Output Disable	V_{IL}	V_{IH}	V_{IH}	X	V_{CC}	Hi-Z
Program	V_{IL}	V_{IH}	V_{IL} Pulse	X	V_{PP}	Data In
Verify	V_{IL}	V_{IL}	V_{IH}	X	V_{PP}	Data Out
Program Inhibit	V_{IH}	X	X	X	V_{PP}	Hi-Z
Standby	V_{IH}	X	X	X	V_{CC}	Hi-Z
Electronic Signature	V_{IL}	V_{IL}	V_{IH}	V_{ID}	V_{CC}	Codes Out

Note: X = V_{IH} or V_{IL}, V_{ID} = 12V ± 0.5%.

Lập trình cho 2764

- Trước hết cần phải xoá
  - Xoá một chip tức là làm cho tất cả các bit = 1
- Xoá một chip EPROM bằng tia cực tím
- Lập trình bằng cách:
  - VPP mắc ở mức 12.5V
  - E và P đều ở mức thấp TTL
- Các bit dữ liệu đưa vào các chân dữ liệu
- Các bit địa chỉ đưa vào các chân địa chỉ

4.3 Các chip SRAM



Đọc dữ liệu từ một chip SRAM

Để đọc dữ liệu từ 1 ô nhớ nào đó của 1 chip SRAM nào đó, vi xử lý cần phải:

- Chọn chip đó: 0 ----------> CS
- Áp các tín hiệu địa chỉ vào $A_{p-1} - A_0$
- Đọc: 0 ---------- > OE

Kết quả là m bit dữ liệu cần đọc xuất hiện ở các chân dữ liệu $D_{m-1} - D_0$

Ghi dữ liệu vào một chip SRAM

Để ghi m bit dữ liệu vào 1 ô nhớ nào đó của 1 chip SRAM nào đó, vi xử lý cần phải:

- Chọn chip đó: 0 --------> CS
- Áp các tín hiệu địa chỉ vào $A_{p-1} - A_0$
- Áp m bit dữ liệu cần ghi vào các chân dữ liệu $D_{m-1} - D_0$
- Ghi: 0 --------- > WE

Kết quả là các bit dữ liệu ở các chân dữ liệu sẽ được ghi vào ô nhớ đã chọn

4.4 Bus hệ thống của 8088

- Bus địa chỉ 20-bit: gồm các đường địa chỉ được ký hiệu từ A_{19} đến A_0
- Bus dữ liệu 8-bit: gồm các đường dữ liệu được ký hiệu từ D_7 đến D_0
- Bus điều khiển gồm các đường điều khiển riêng lẽ phục vụ cho hoạt động truy cập bộ nhớ và các cổng I/O, mỗi đường thường được ký hiệu bằng tên của tín hiệu điều khiển
- Bus hệ thống không nối trực tiếp với các chân của 8088: thông qua các mạch đệm, chốt.

80x86 Microprocessors

Product	***8008***	***8080***	***8085***	***8086***	***8088***	***80286***	***80386***	***80486***	***Pent.***	***Pent. Pro***
Year Introduced	1972	1974	1976	1978	1979	1982	1985	1989	1992	1995
Technology	PMOS	NMOS	NMOS	NMOS	NMOS	NMOS	CMOS	CMOS	BICMOS	BICMOS
Clock Rate	0.5-0.8	2-3	3-8	5-10	5-8	10-16?	16-40	66	60-66+	150
Number of Pins	18	40	40	40	40		132	168	273	387
Number of transistors	3000	4500	6500	29K	29K	130K	275K	1.2M	3M	5.5M
Number of instructions	66	111	113	133	133					
Physical Memory	16K	64K	64K	1M	1M	16M	16M4GB	4GB	4GB	64G
Virtual Memory	none	none	none	none	none	1G	64T	64T	64T	64T
Internal Data Bus	8	8	8	16	16	16	32	32	64	32
External Data Bus	8	8	8	16	8	16	16,32	32	64	64
Address Bus	8	16	16	20	20	24	24,32	32	32	36
Data Types	8	8	8	8,16	8,16	8,16	8,16,32	8,16,32	8,16,32	8,16,32

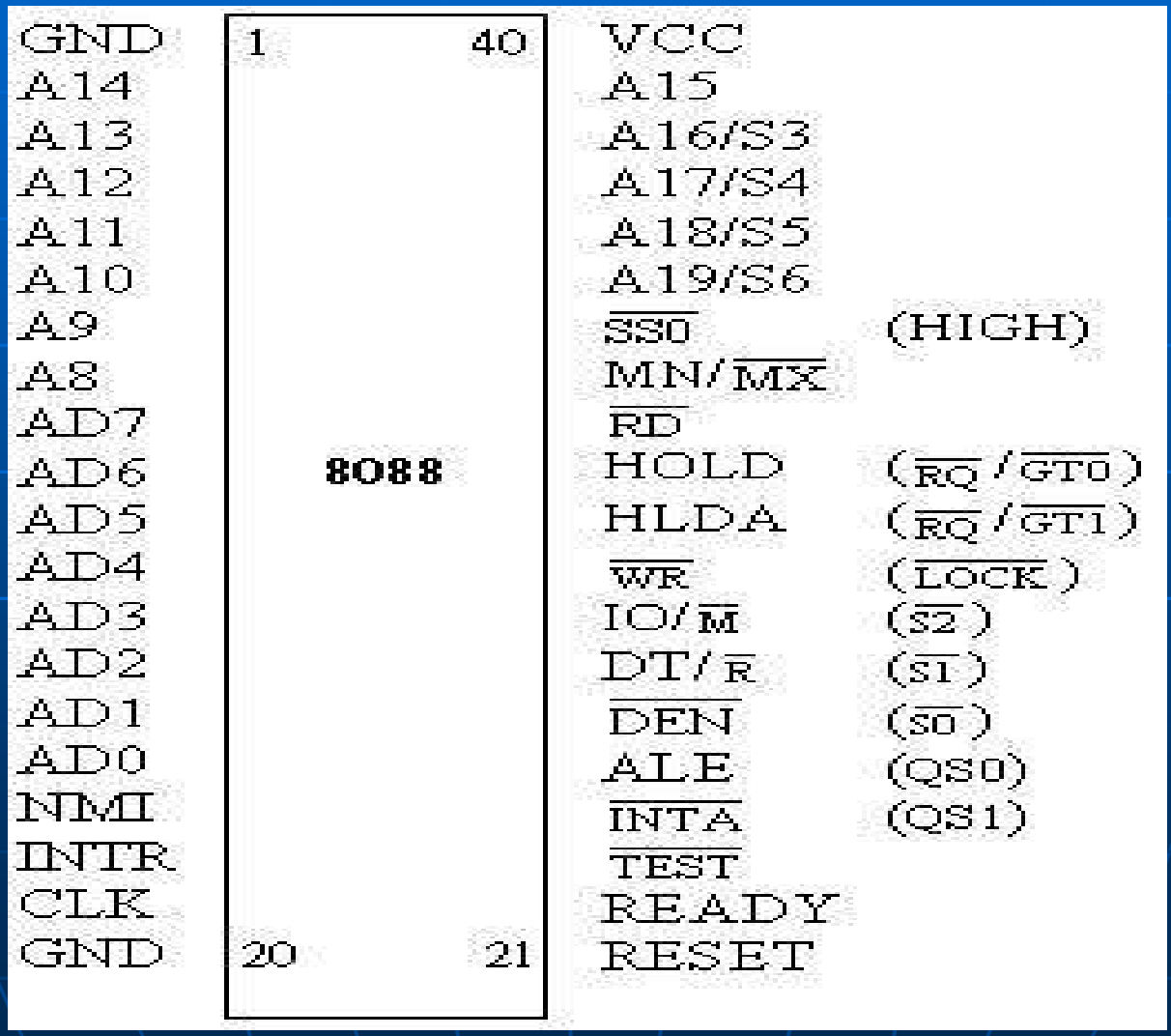
8088/8086 Microprocessor

- DIP 40 pin
- Data bus
  - Bus dữ liệu trong :16 bit
  - Bus dữ liệu ngoài của 8088: 8 bit dùng AD0-AD7
  - Bus dữ liệu ngoài của 8086:16 bit dùng AD0-AD15
  - ALE (Address Latch Enable)

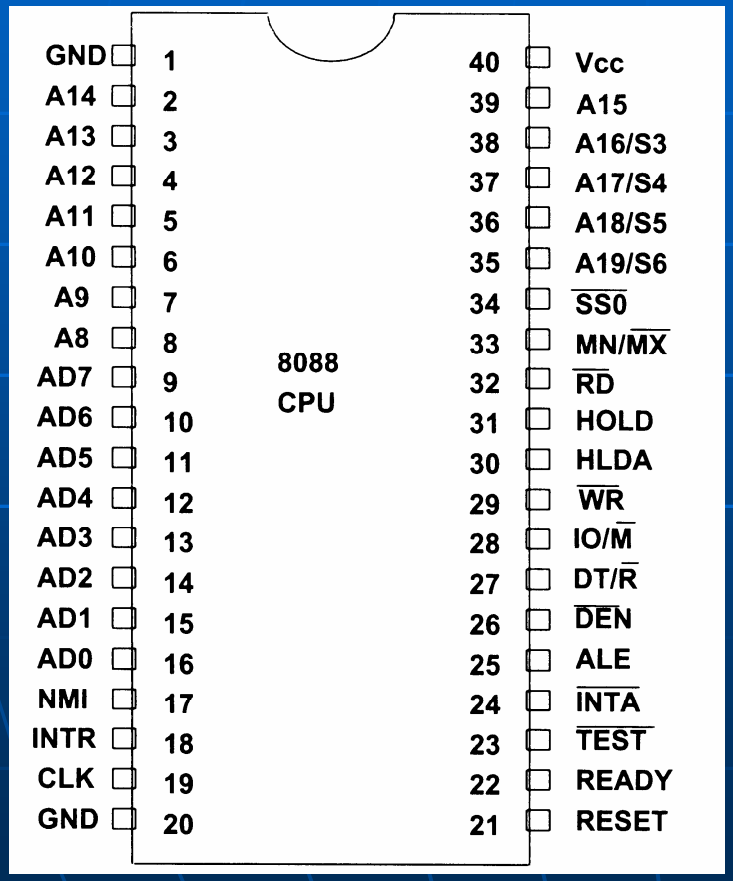
8088/8086 Microprocessor

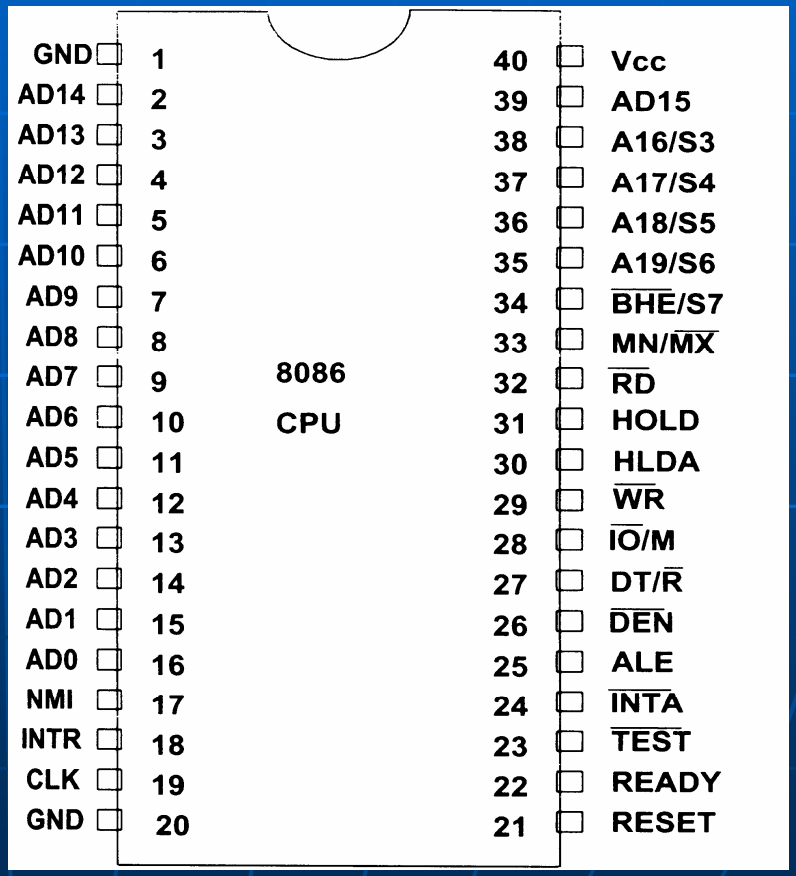
- Bus địa chỉ
  - ALE = 1
  - Sử dụng 74LS373 để tách và chốt địa chỉ
    - Đầu vào: AD0-AD7 (8088) hoặc AD0-AD15 (8086) và ALE
    - Đầu ra: A0-A7 (8088) hoặc A0-A15 (8086)

Sơ đồ chân của 8088



Sơ đồ chân 8088/8086 (Min Mode)

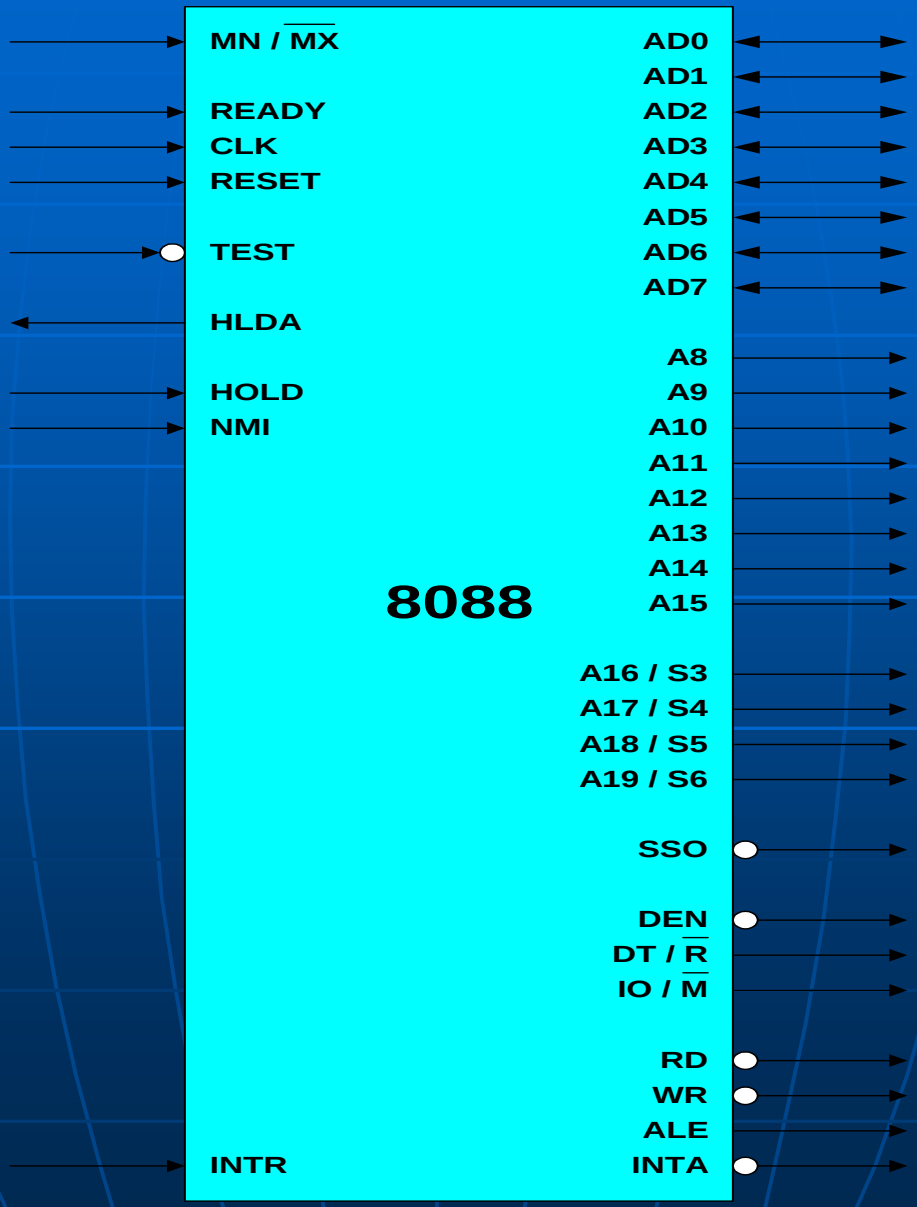




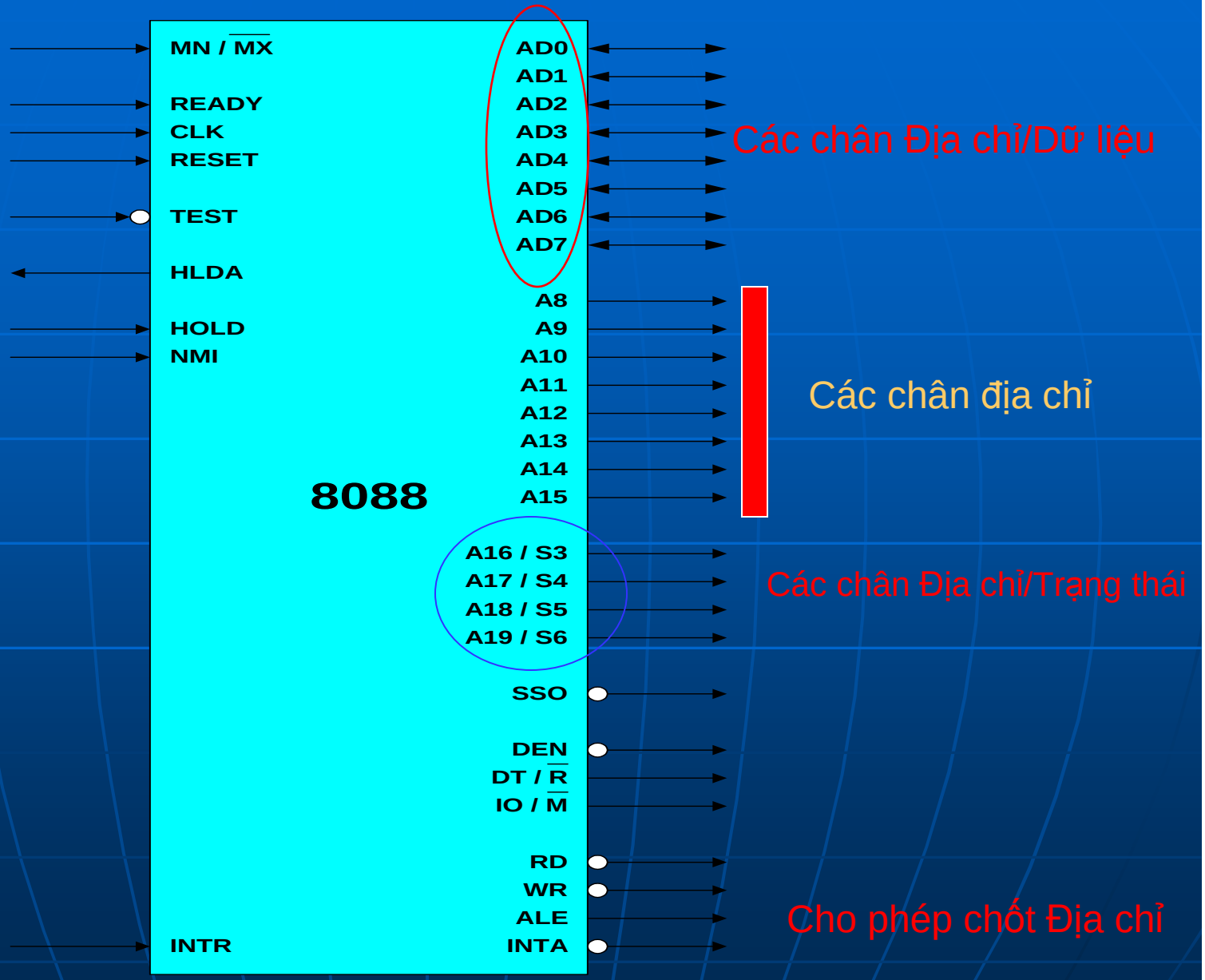
Minimum/Maximum Mode

- Ảnh hưởng đến các chân 24-31
- Minimum Mode
  - Các chân 24-31 là các tín hiệu điều khiển I/O và bộ nhớ
  - Các tín hiệu điều khiển đều từ 8088/8086
  - Tương tự với 8085A
- Maximum Mode
  - Một số tín hiệu điều khiển được tạo ra từ ngoài
  - Một số chân có thêm chức năng mới
  - Khi có dùng bộ đồng xử lý toán 8087

Sơ đồ chân của 8088



Tín hiệu ở các chân của 8088



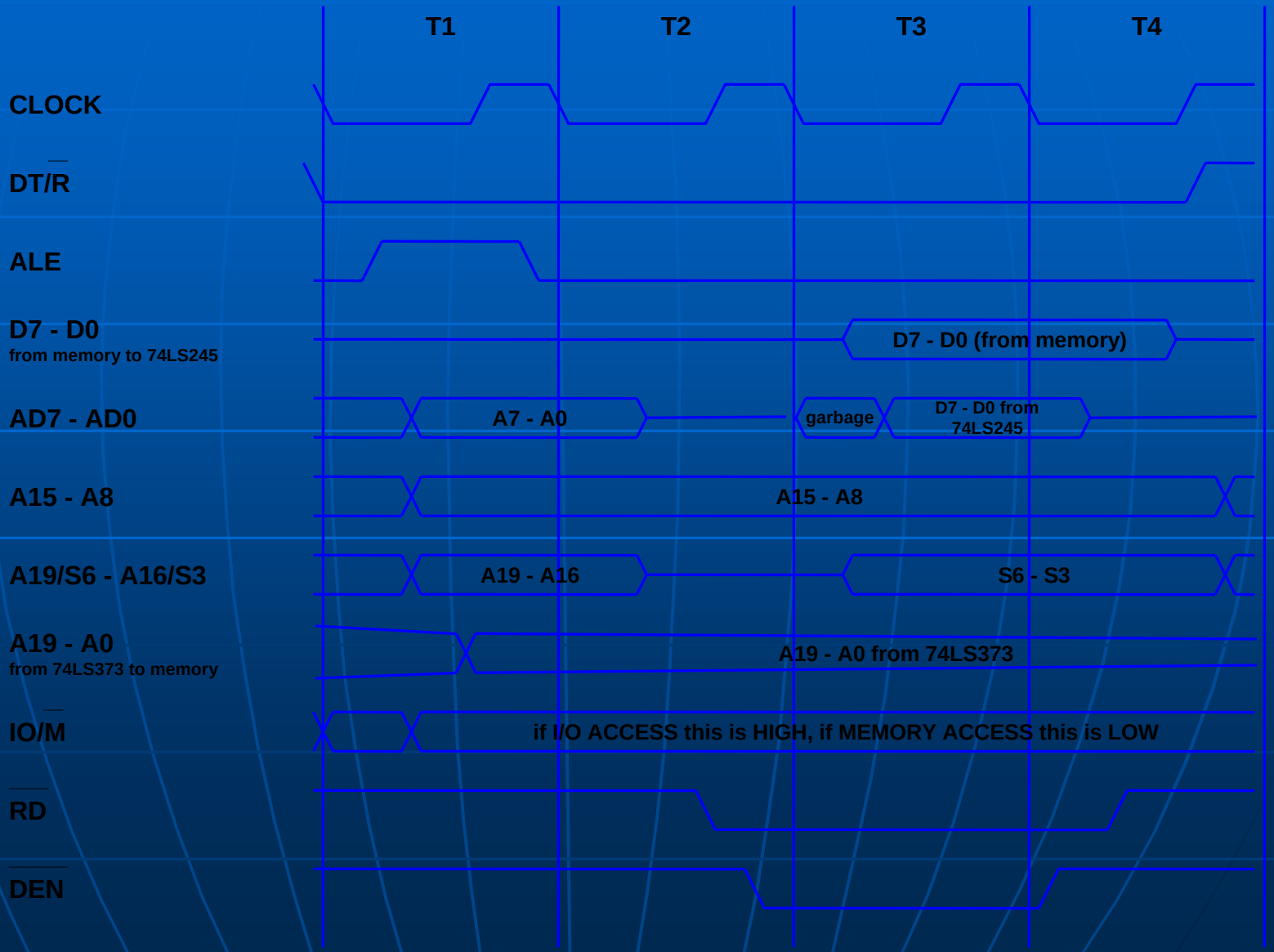
Các chân Địa chỉ/Dữ liệu

- Các chân AD_7 đến AD_0
- Kỹ thuật Multiplexing: Tín hiệu ở các chân này lúc này là tín hiệu địa chỉ, lúc khác là tín hiệu dữ liệu phụ thuộc vào tín hiệu điều khiển ALE (Address Latch Enable):
- ALE = 1: AD_7 đến AD_0 = A_7 đến A_0
- ALE = 0: AD_7 đến AD_0 = D_7 đến D_0

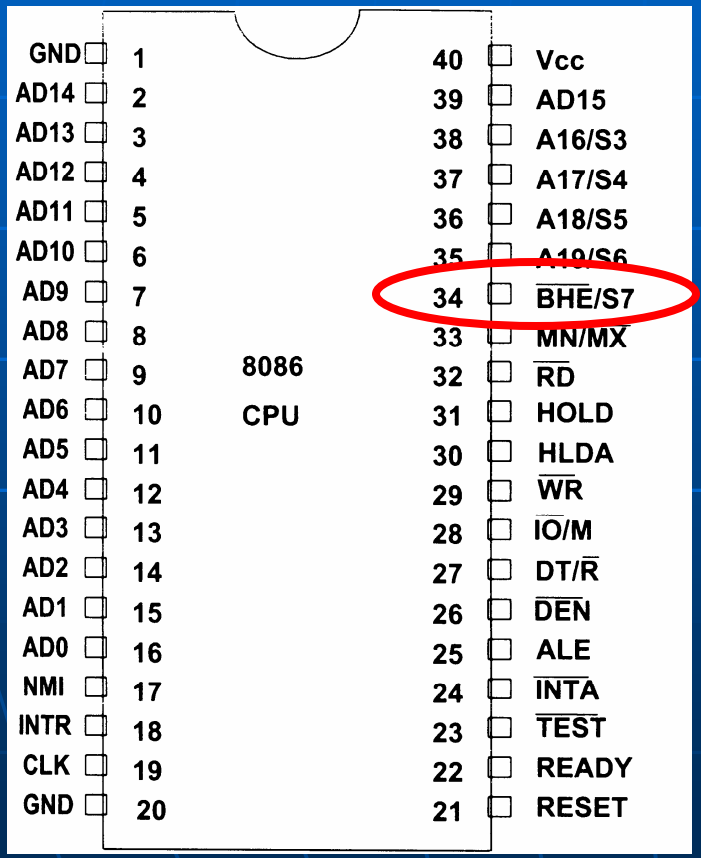
Các chân Địa chỉ và Các chân Địa chỉ/Trạng thái

- Các chân địa chỉ: A_{15} đến A_8
- Tín hiệu ở các chân này luôn là tín hiệu địa chỉ
- Các chân địa chỉ/trạng thái: A_{19}/S_6 đến A_{16}/S_3:
- ALE = 1: A_{19} đến A_{16}
- ALE = 0: S_6 đến S_3

Processor Timing Diagram of 8088 (Minimum Mode) for Memory or I/O Read (with 74245)

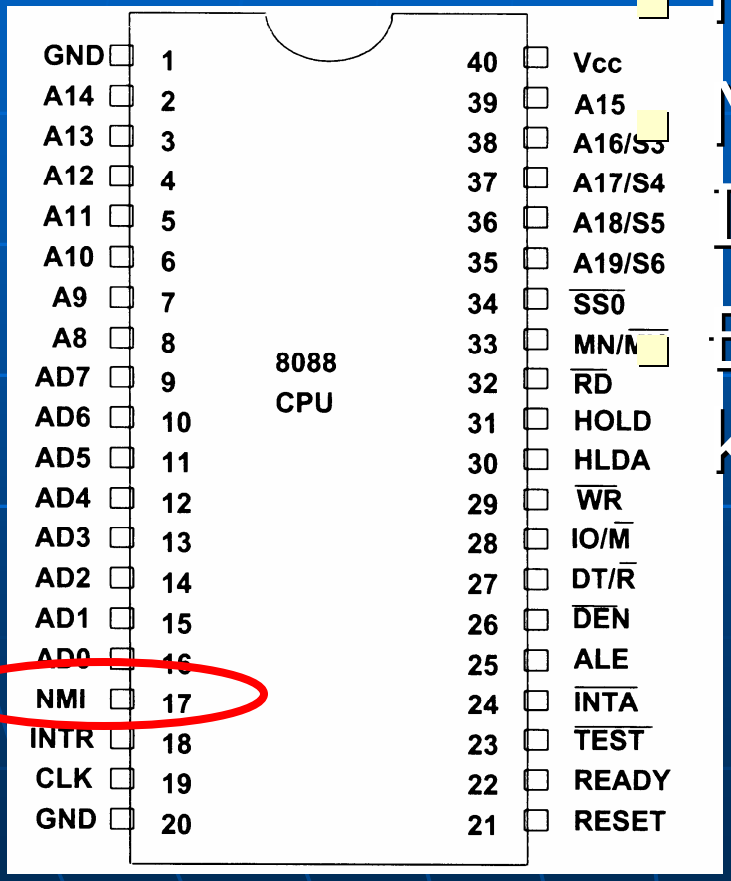


Mô tả chân



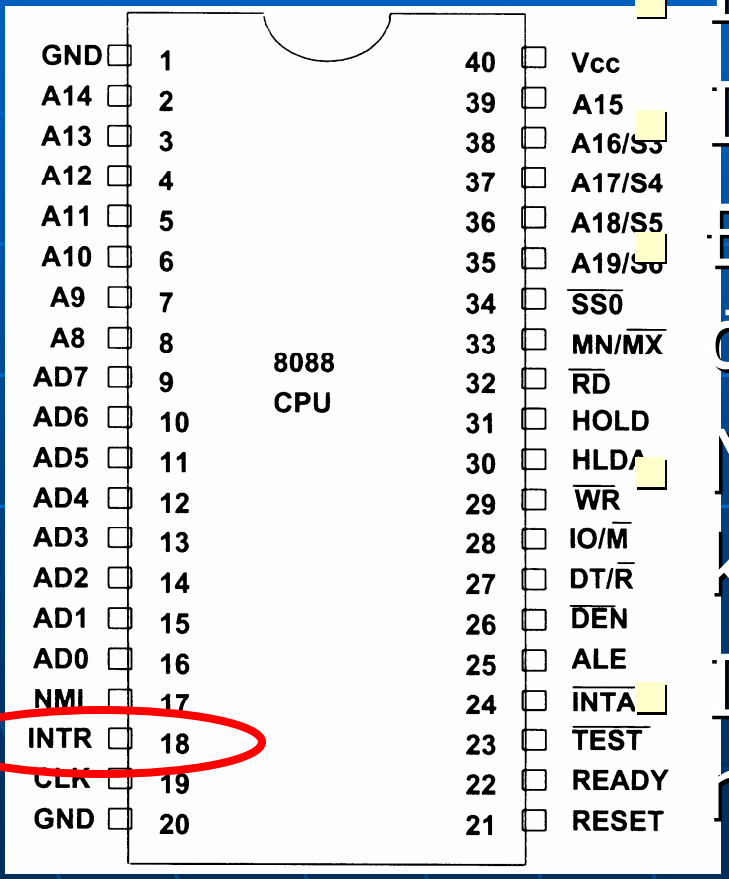
- BHE
  Bus High Enable
  Phân biệt byte thấp và byte cao của một từ (chỉ với 8086)

Mô tả chân



- NMI
- Non Maskable Interrupt
- Đầu vào ngắt không che được

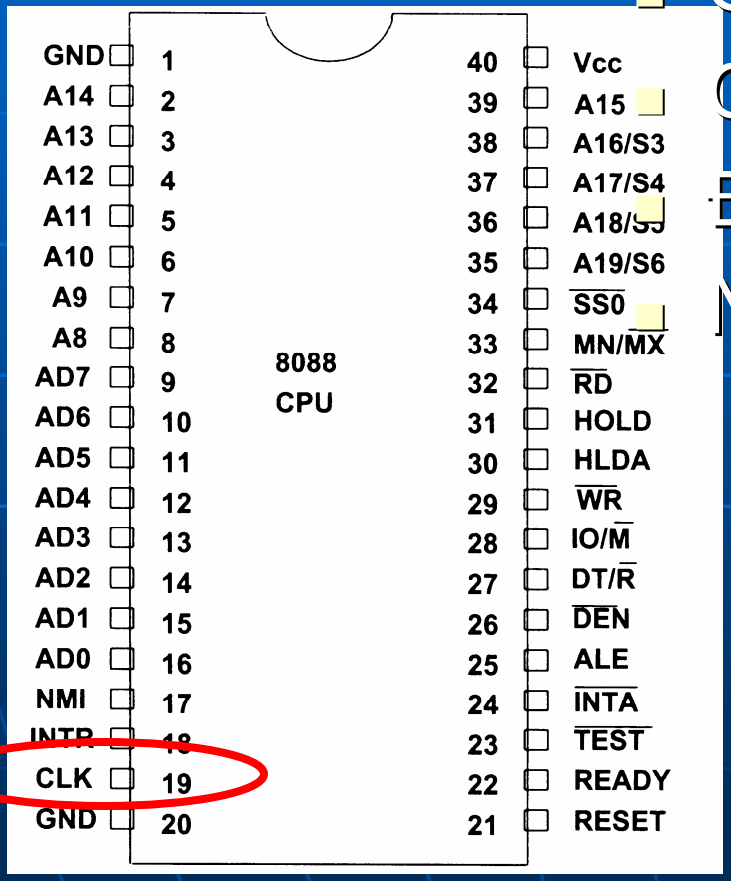
Mô tả chân



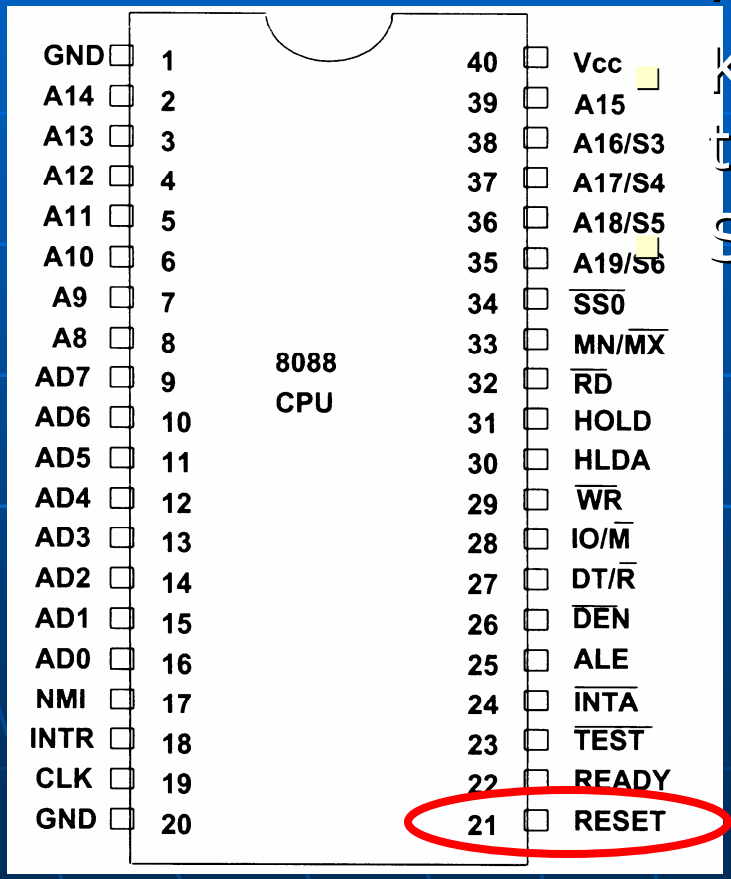
- INTR
  - Interrupt Request
  - Đầu vào ngắt che được
  - Nối với chip điều khiển ngắt 8259
- INTA: chấp nhận ngắt

Mô tả chân

- CLK
- Clock
- Đầu vào đồng hồ
- Nối với chip 8284



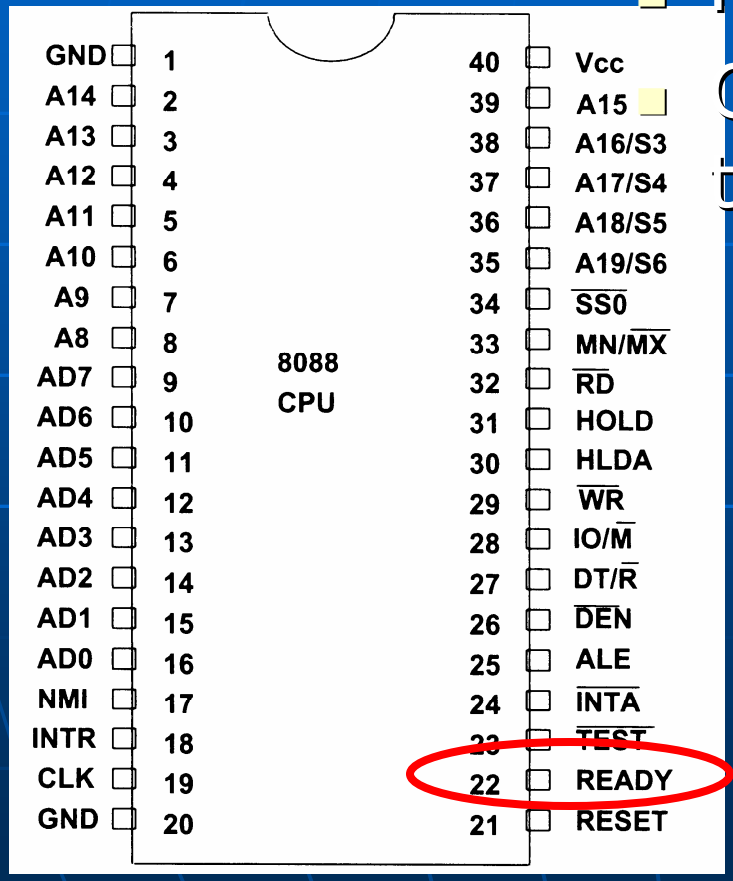
Mô tả chân



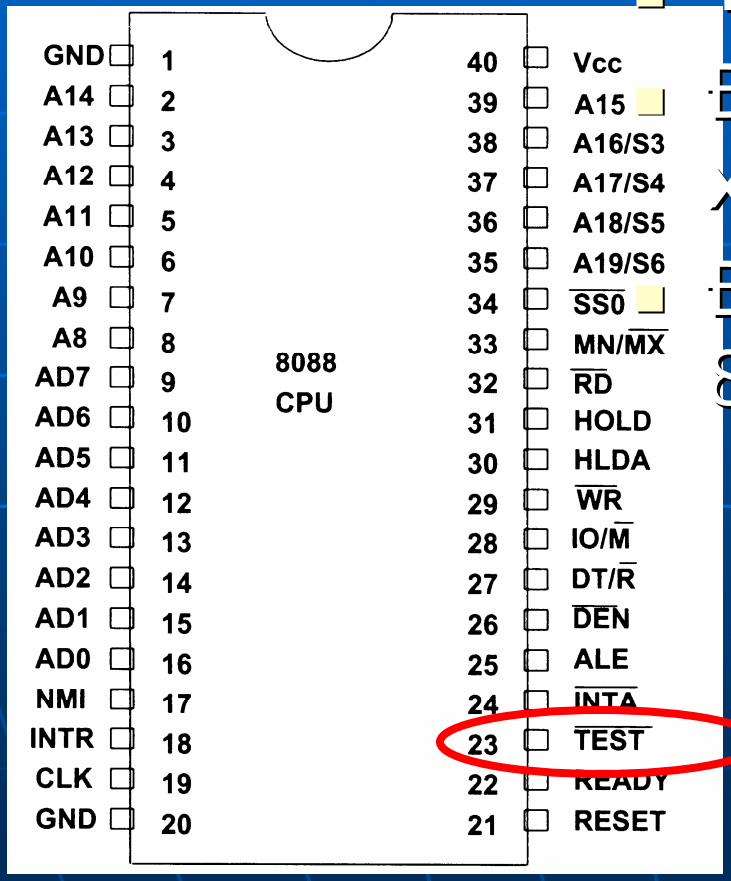
- RESET
- Kết thúc hoạt động hiện thời và huy bỏ mọi thứ
- Sau khi reset
  - CS=FFFFH
  - DS=0000H
  - SS=0000H
  - ES=0000H
  - IP=0000H
  - Các cờ bị xoá
  - Hàng đợI lệnh rỗng

Mô tả chân

- READY
  - Chèn thêm một trạng thái đợi (wait state)

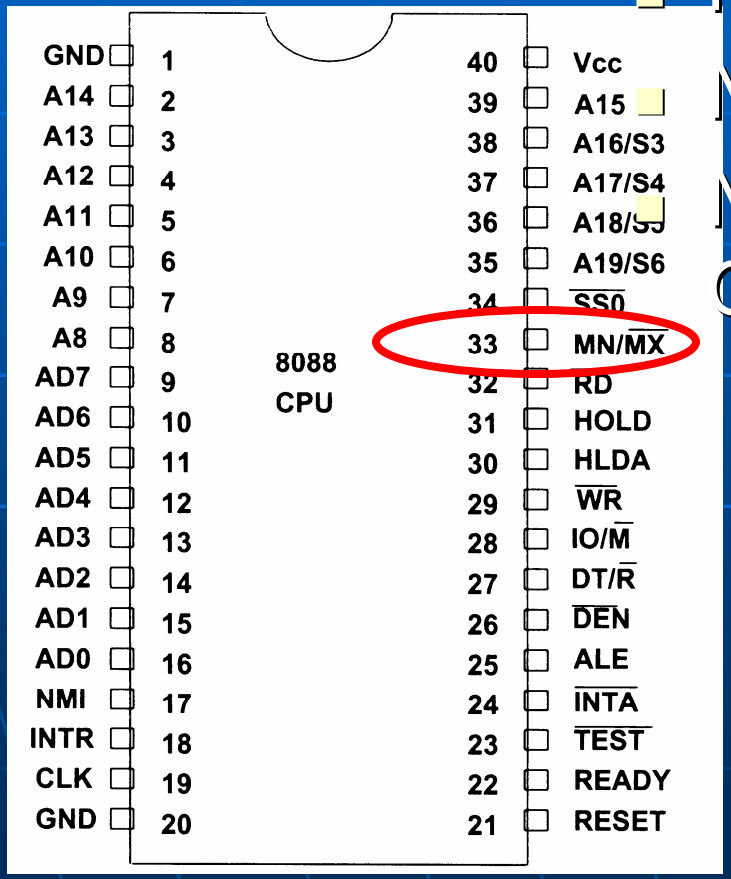


Mô tả chân



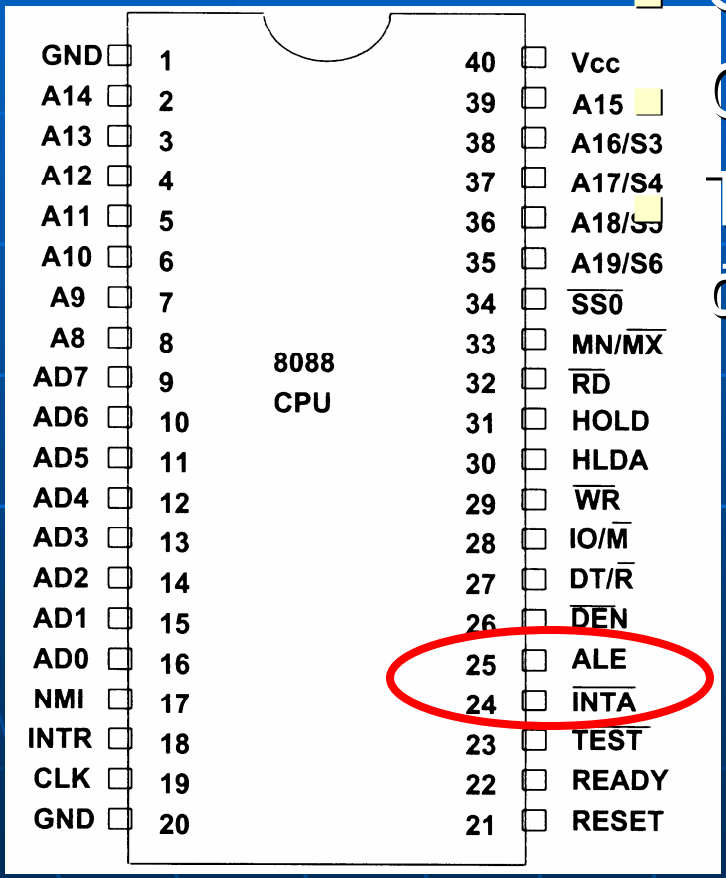
- TEST
  - Đến từ 8087 (Bộ đồng xử lý)
  - Đồng bộ 8088 và 8087

Mô tả chân



- MN/MX
  - Minimum mode = +5V
  - Maximum mode = Gnd

Mô tả chân – Max



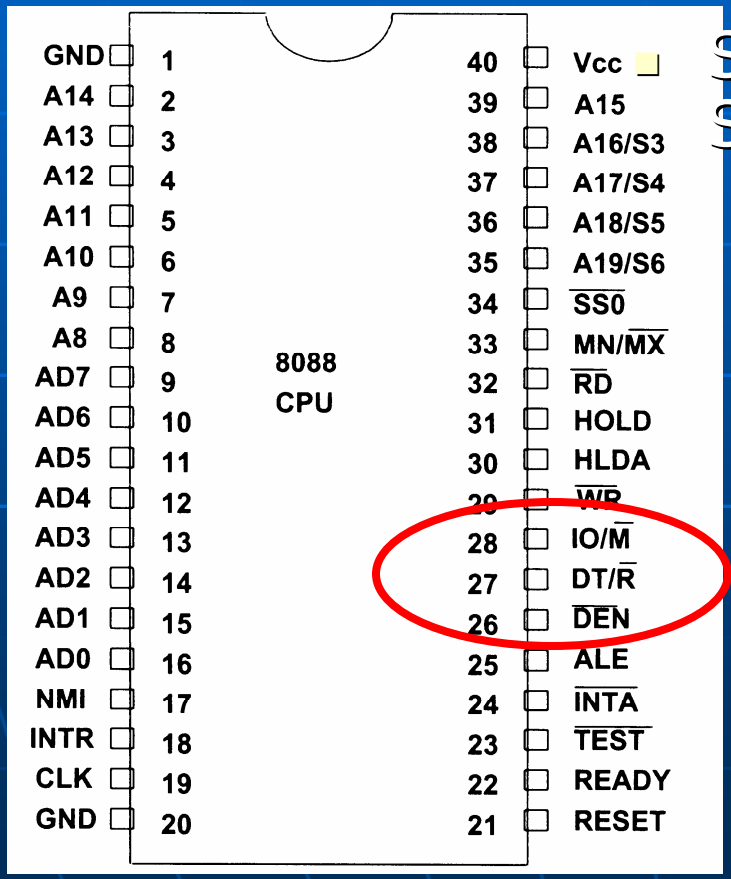
- QS0, QS1

Queue status

Trạng thái của hàng đợi lệnh:

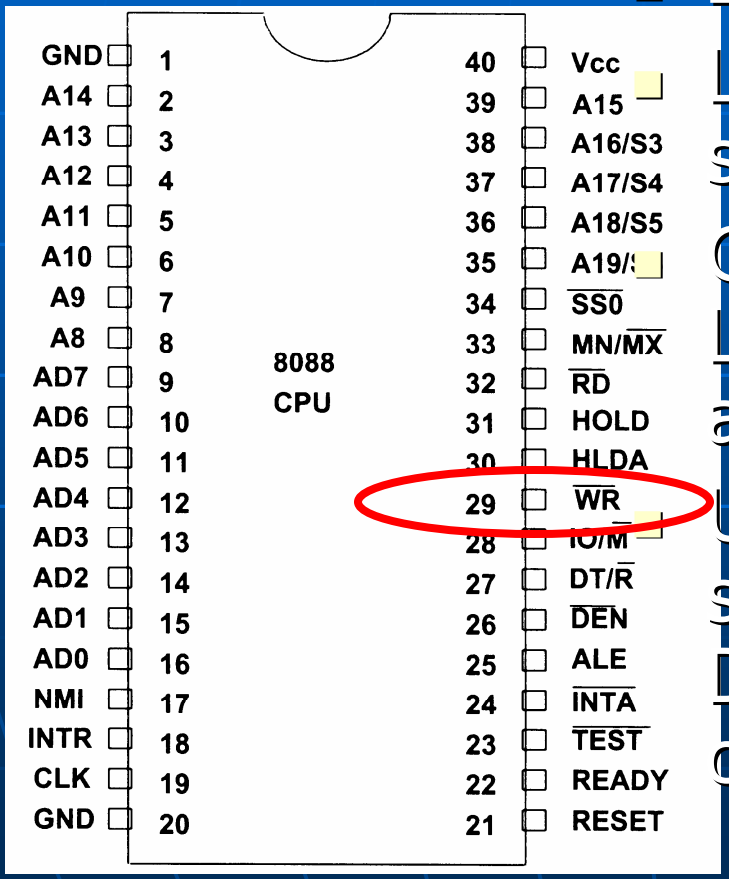
- 00 – No operation
- 01 – first byte of opcode from queue
- 10 – empty the queue
- 11 – subsequent byte from queue

Mô tả chân – Max



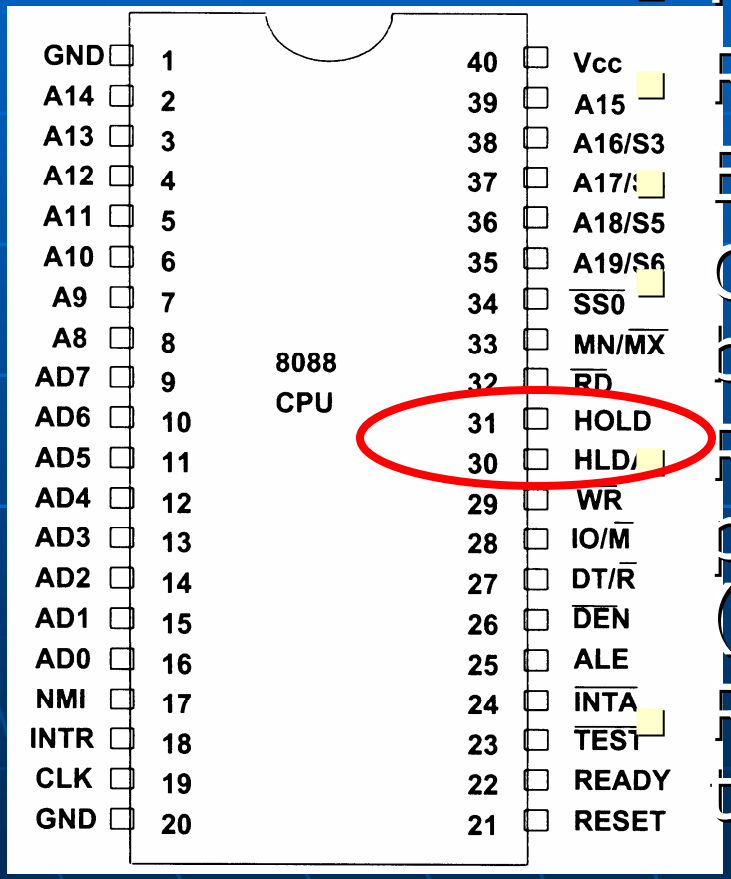
- S0, S1, S2 Status Signal Pins (S2-S0)
  - 000 – INTA – interrupt acknowledge
  - 001 – IORC – read I/O port
  - 010 – IOWC – write I/O port
  - 011 – none - halt
  - 100 – MRDC – code access
  - 101 – MRDC – read memory
  - 110 – MWTC – write memory
  - 111 – none - passive

Mô tả chân – Max



- LOCK
  - Locks processor to system bus
  - Gain the lock by using LOCK prefix on an assembly instruction
  - Used with status signals to prevent DMA from gaining control of the buses

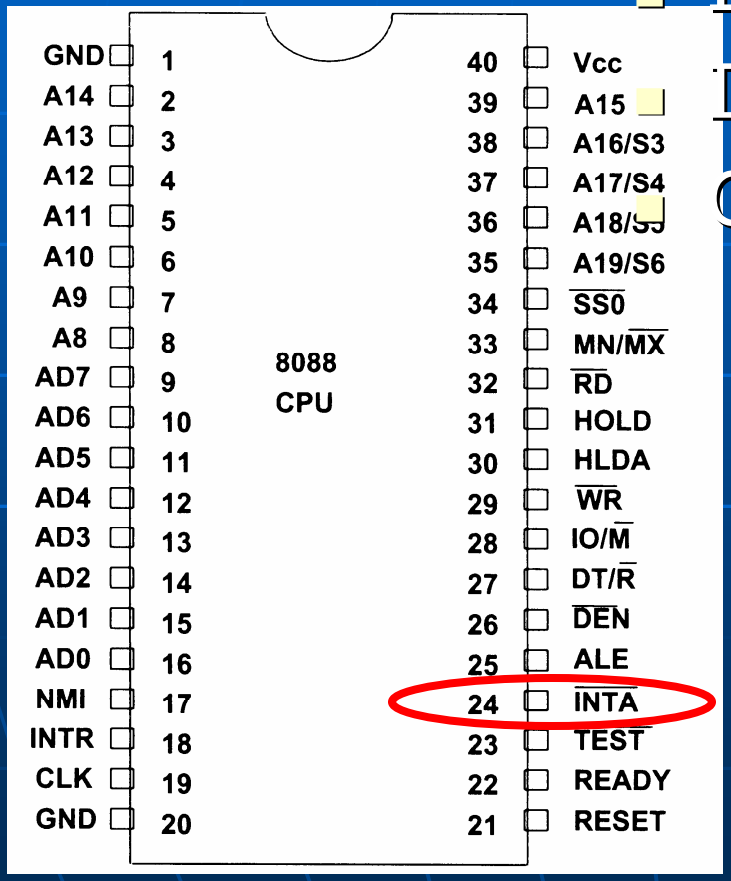
Mô tả chân – Max



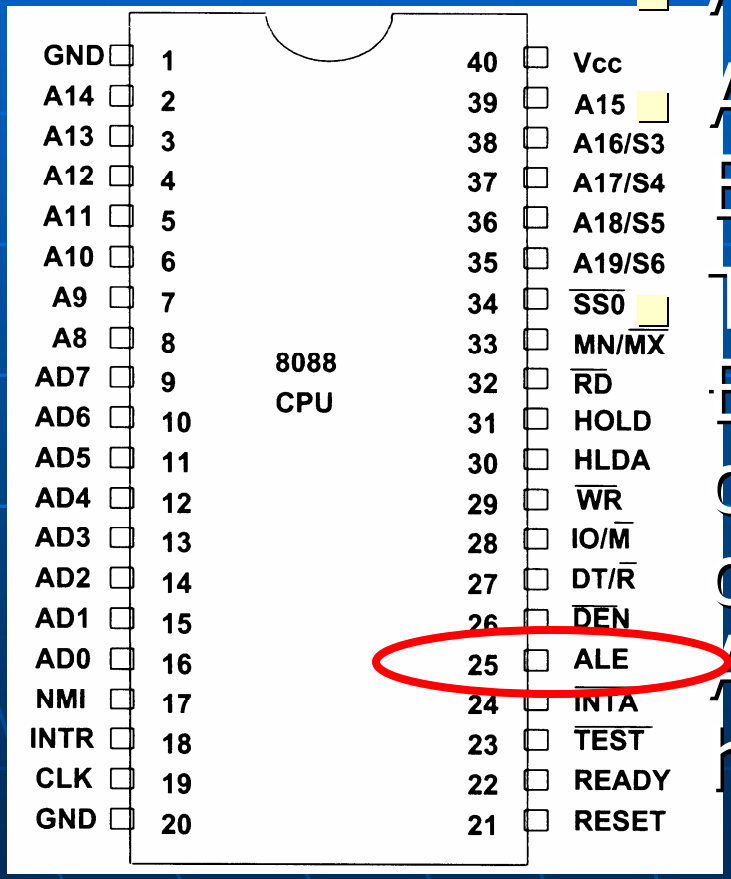
- RQ/GT0, RQ/GT1
  - Request/Grant
  - Bi-directional
  - Gain control of local bus
  - RQ/GT0 normally permanently high (disabled)
  - RQ/GT1 is connected to the 8087

Mô tả chân – Min

- $\overline{INTA}$
  - Interrupt acknowledge
  - Chấp nhận ngắt

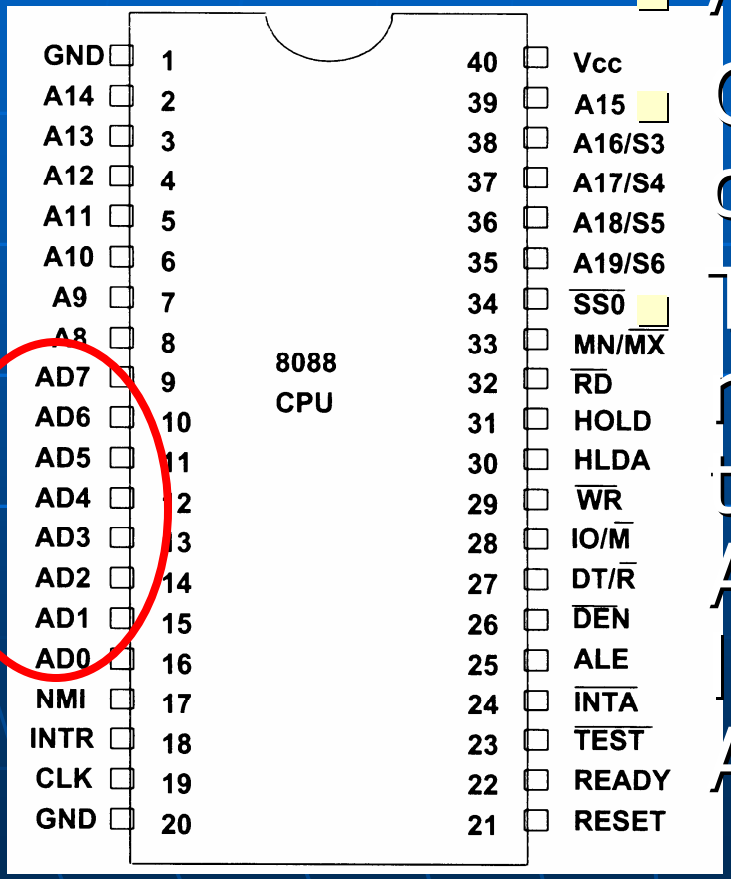


Mô tả chân – Min

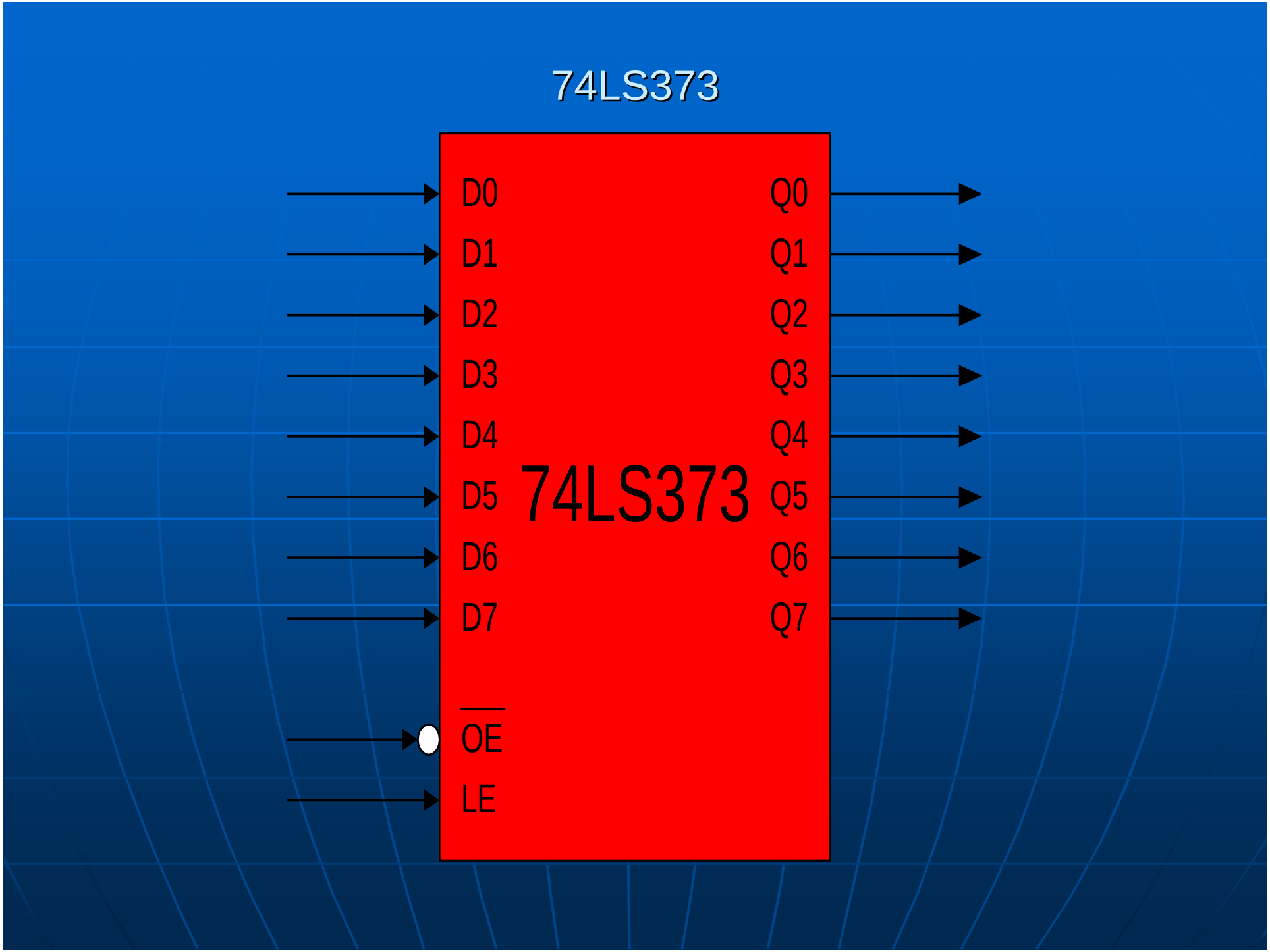


- ALE
- Address Latch Enable
- Tín hiệu ở các chân Địa chỉ/Dữ liệu và các chân Địa chỉ/Trạng thái lúc ALE = 1 là các tín hiệu địa chỉ

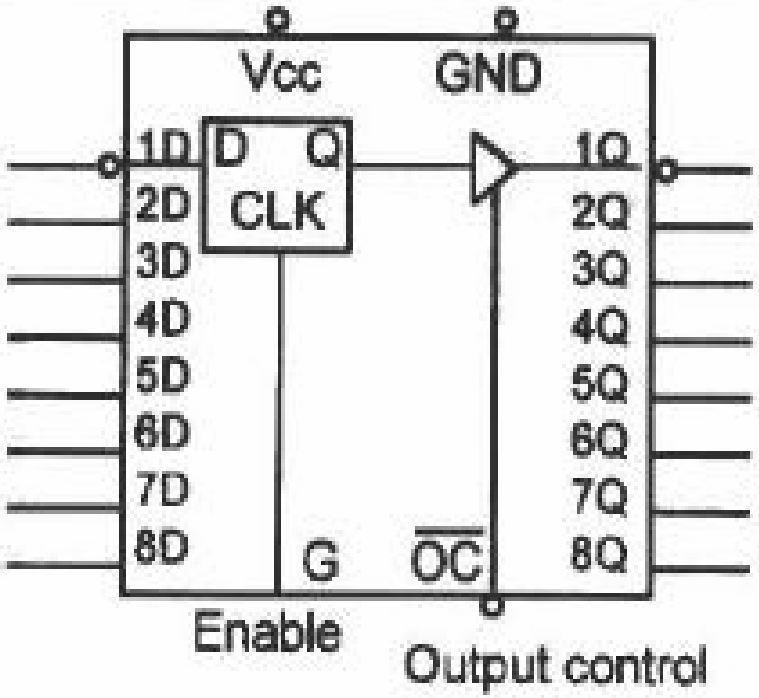
Mô tả chân



- AD0-AD7
  - Các chân Địa chỉ/Dữ liệu
  - Tín hiệu ở các chân này là 8 bit địa chỉ thấp A0 đến A7 khi ALE =1, là 8 bit dữ liệu D0 đến D7 khi ALE = 0


74LS373
D0
D1
D2
D3
D4
D5
D6
D7
74LS373
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
OE
LE

74LS373



Function Table

Output Control	Enable G	Enable D	Output
L	H	H	H
L	H	L	L
L	L	X	Q0
H	X	X	Z

Figure 11-1. 74LS373 D Latch

Dùng 74LS373 để tách và chốt địa chỉ

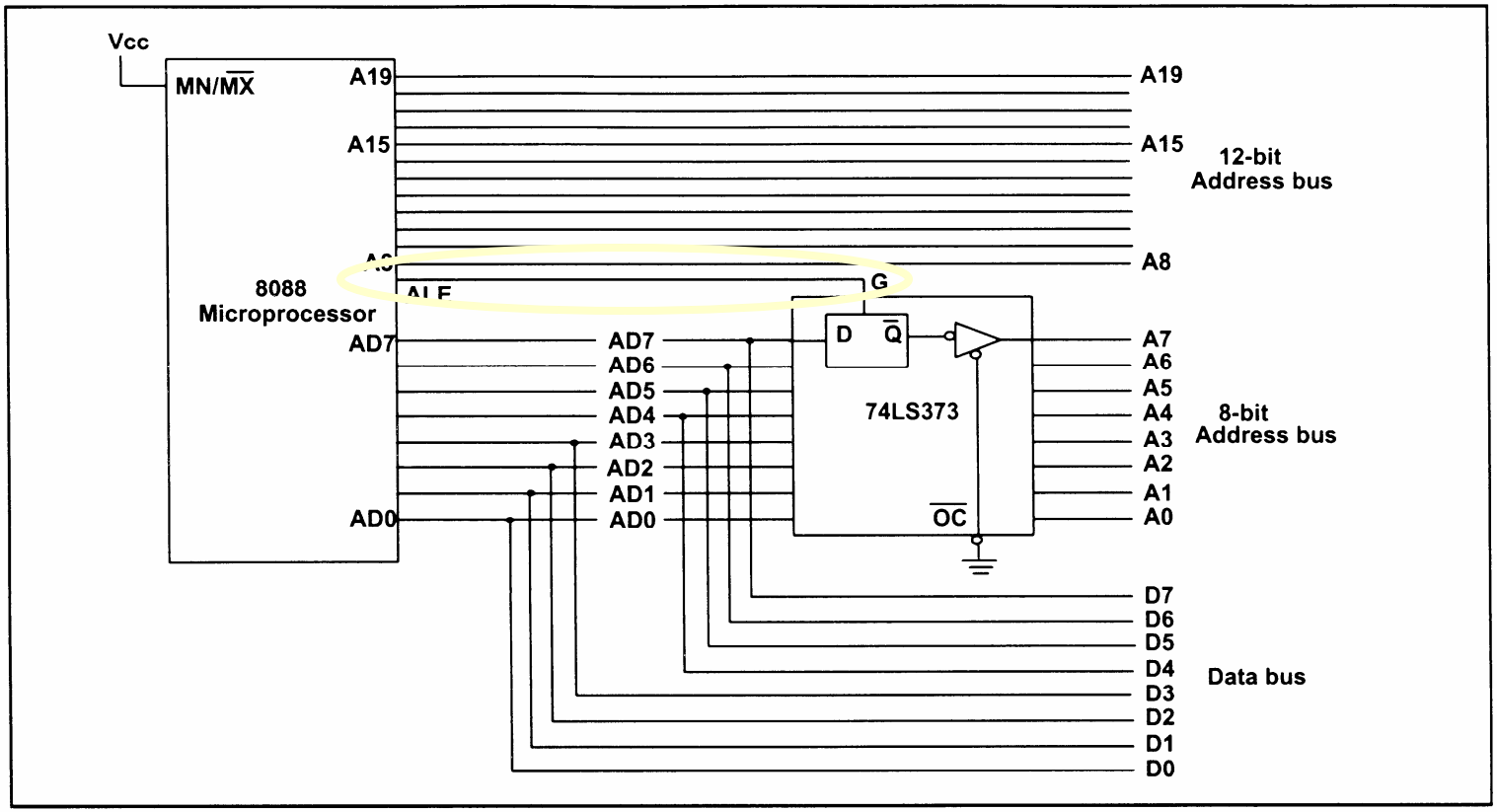
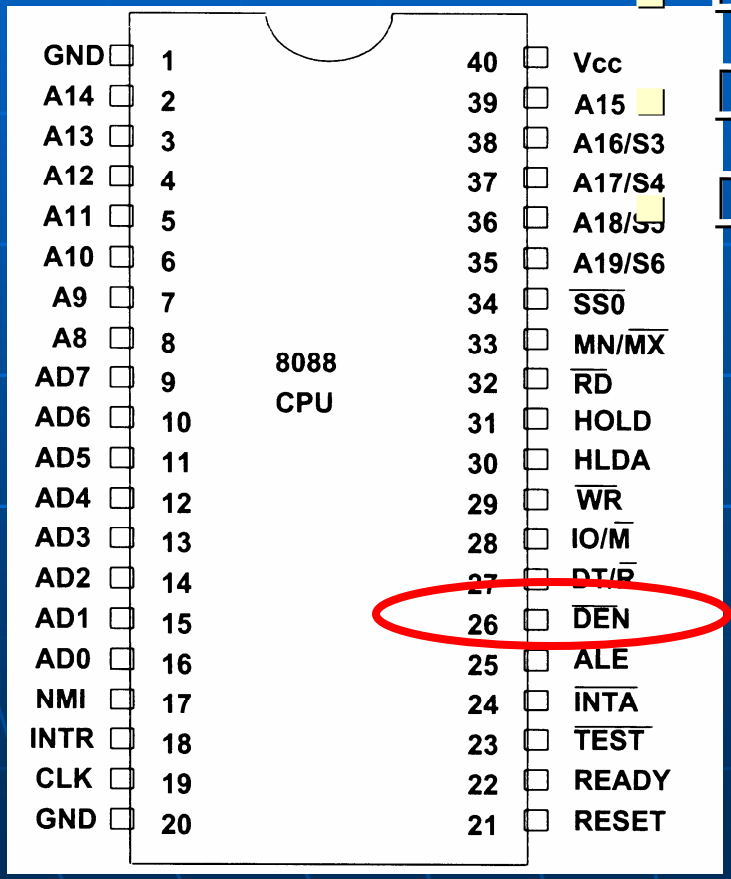


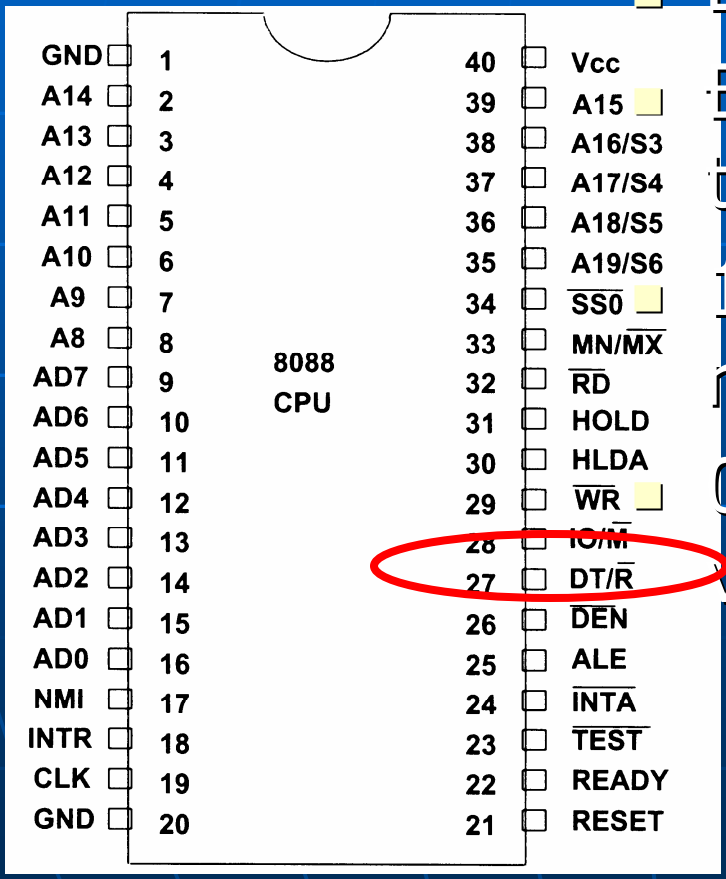
Figure 9-3. Role of ALE in Address/Data Demultiplexing

Mô tả chân – Min



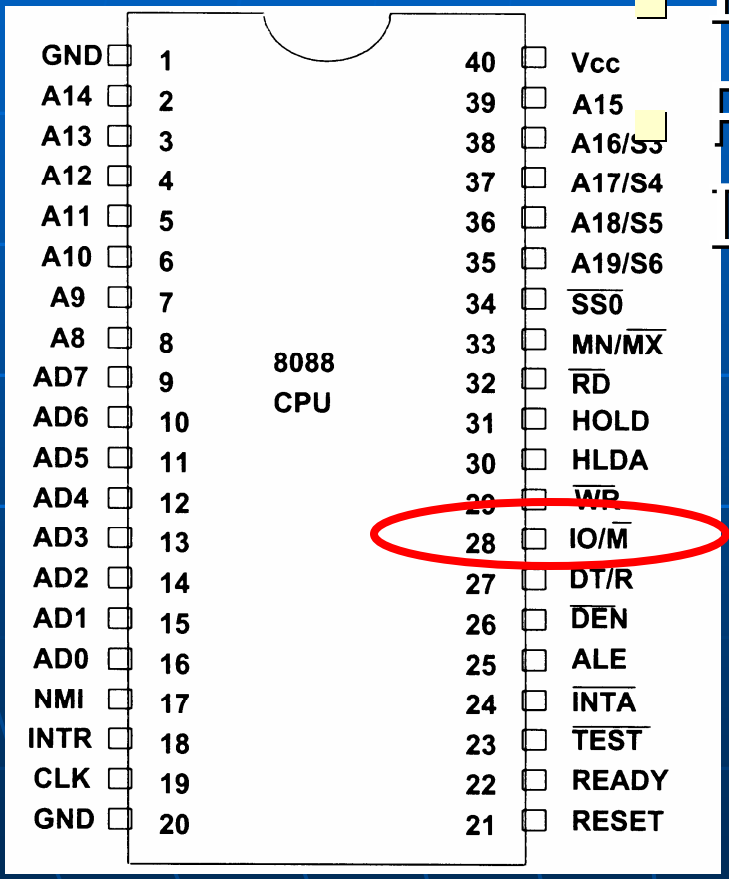
- DEN
  - Data Enable
  - Dữ liệu có nghĩa

Mô tả chân – Min



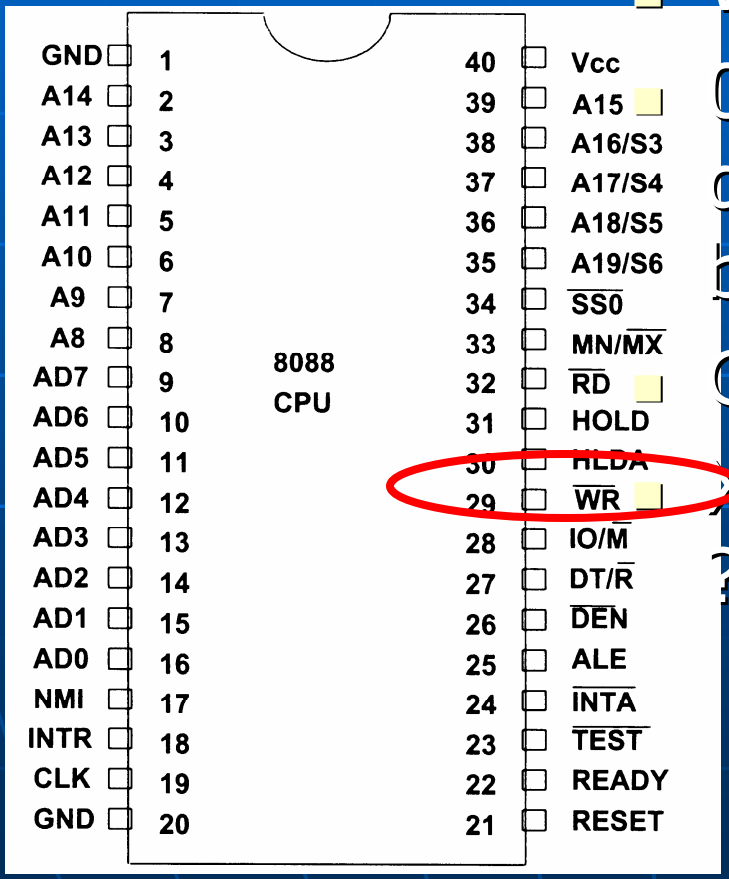
- DT/R _
  - Điều khiển hướng của tín hiệu dữ liệu:
  - 1: Tín hiệu dữ liệu đi ra từ 8088
  - 0: Tín hiệu dữ liệu đi vào 8088

Mô tả chân – Min



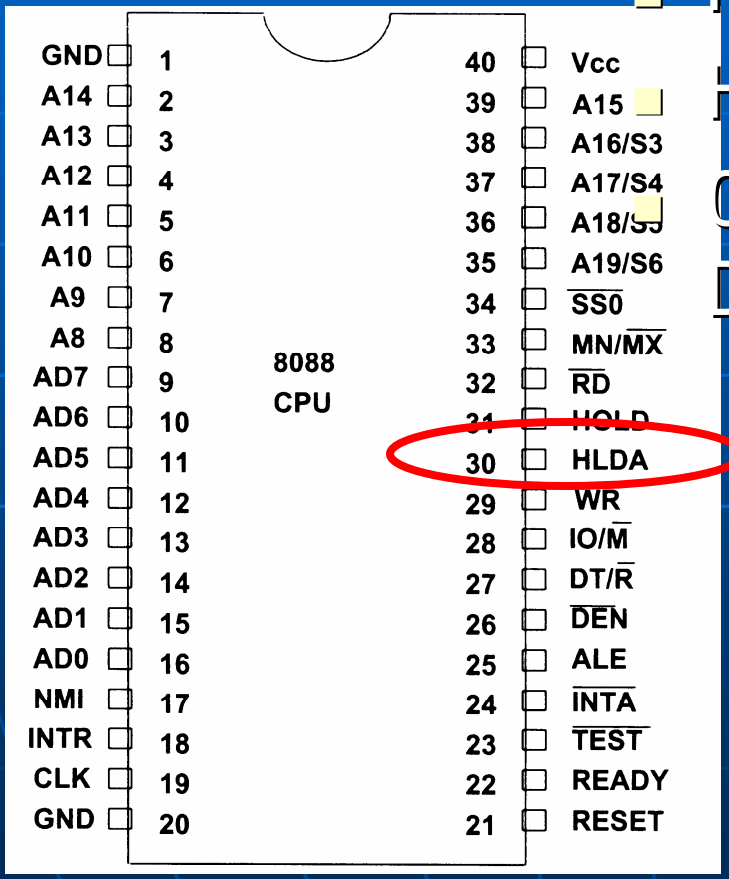
- IO/M _
- Phân biệt: truy cập I/O hay Bộ nhớ
  - 1: 8088 truy cập I/O
  - 0: 8088 truy cập bộ nhớ

Mô tả chân – Min



- WR__
  - 0: Tín hiệu trên bus dữ liệu được ghi vào bộ nhớ hoặc I/O
  - Ghi bộ nhớ: ?
  - Xuất dữ liệu ra cổng: ?

Mô tả chân – Min



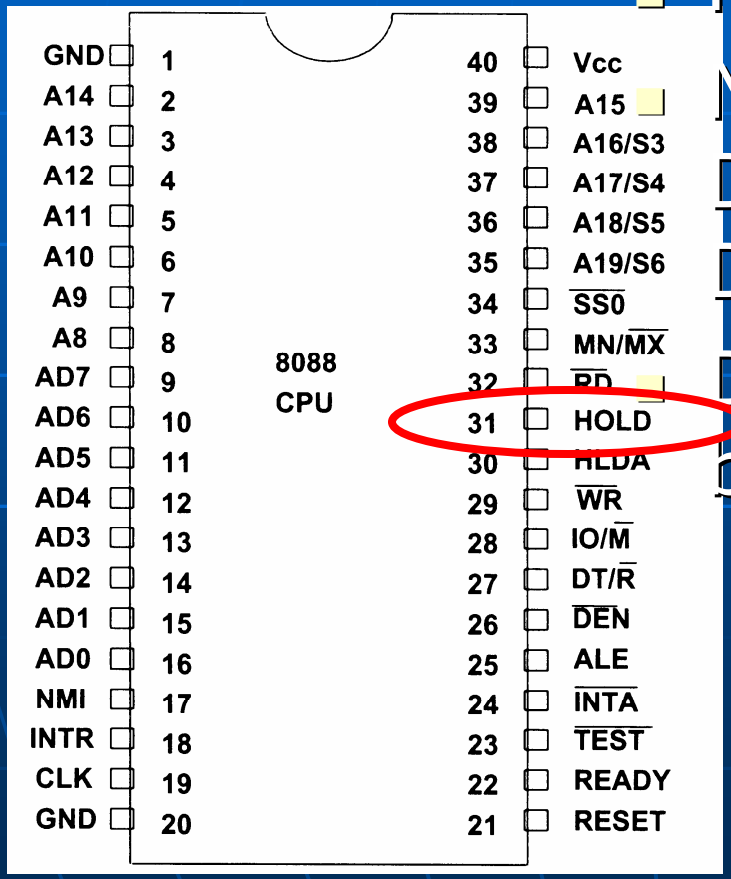
- HLDA

Hold Acknowledge

0: Chấp nhận yêu cầu DMA ở HOLD

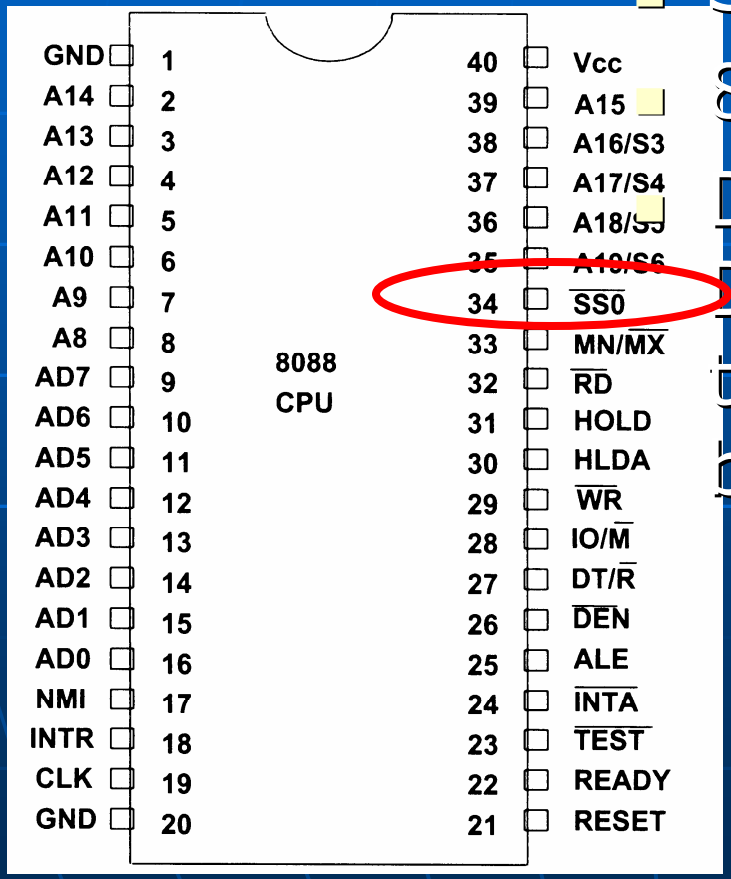
- Báo cho Bộ điều khiển DMA được phép sử dụng bus hệ thống

Mô tả chân – Min



- HOLD
  - Nhận tín hiệu yêu cầu DMA từ Bộ điều khiển DMA (DMAC)
  - DMAC muốn sử dụng bus hệ thống

Mô tả chân – Min



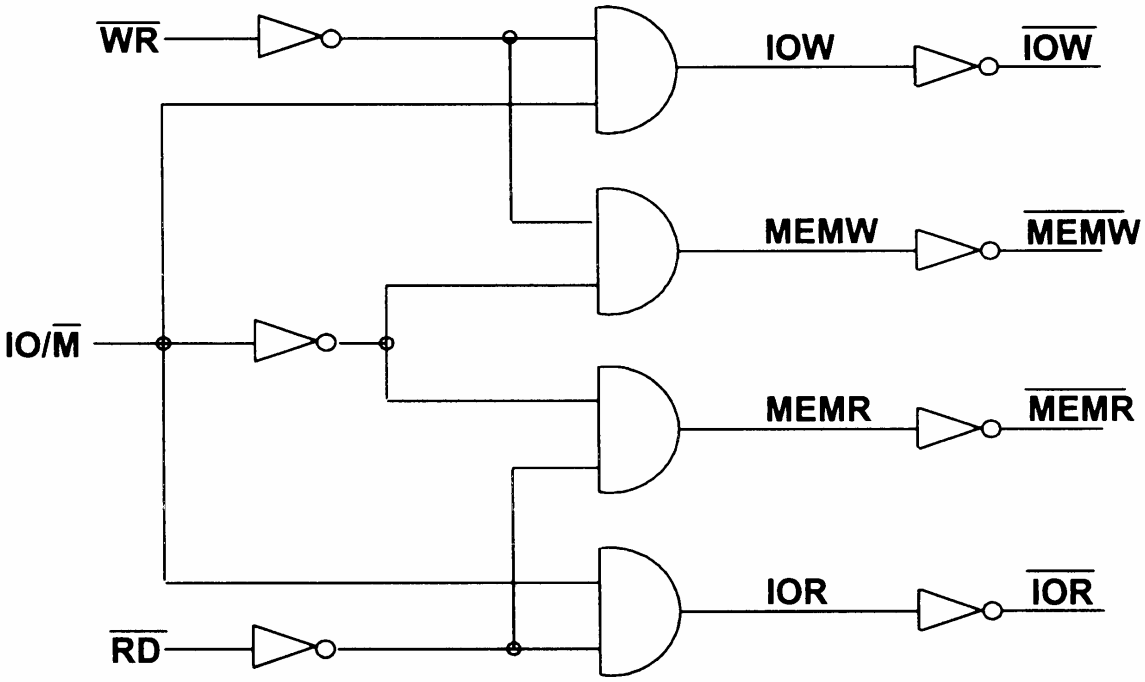
- SSO
- 8088
- Dùng với IO/M và DT/R để xác định trạng thái của chu kỳ bus hiện thời

Các tín hiệu điều khiển

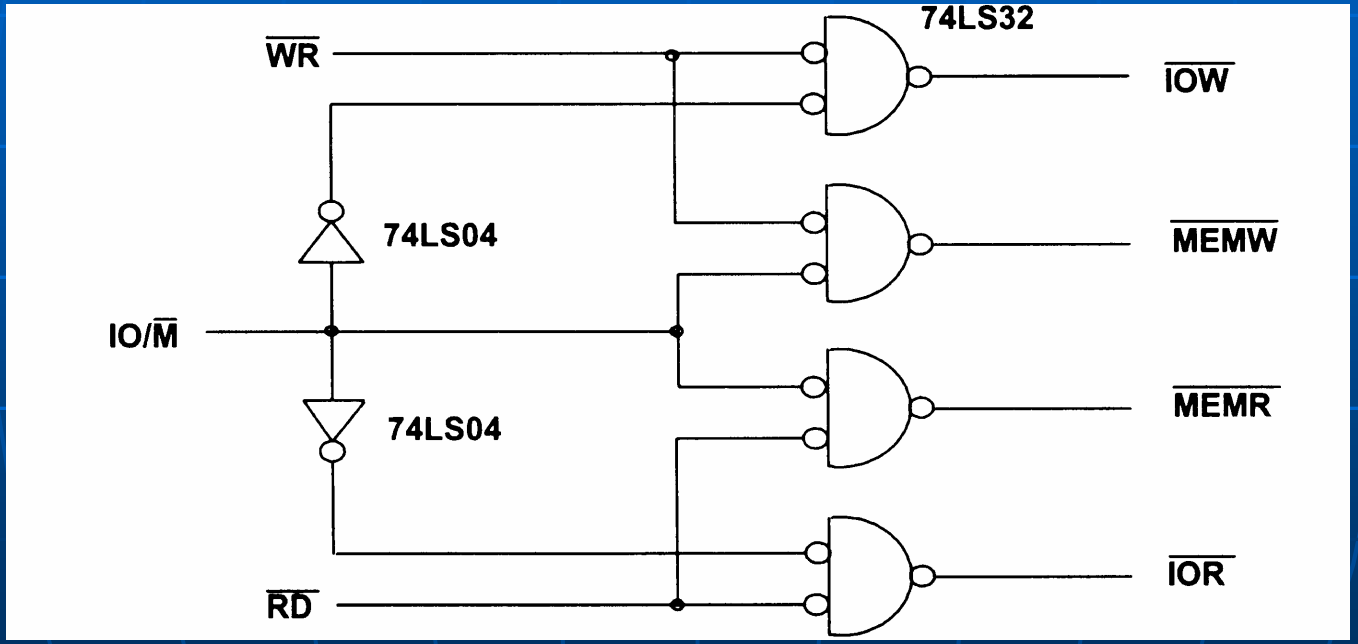
- Có thể sử dụng các cổng logic để tạo ra các tín hiệu điều khiển khác từ các tín hiệu điều khiển sẵn có
  - 3 Tín hiệu:
    - RD, WR and IO/M

$\overline{RD}$	$\overline{WR}$	IO/$\overline{M}$	Signal
0	1	0	$\overline{MEMR}$
1	0	0	$\overline{MEMW}$
0	1	1	$\overline{IOR}$
1	0	1	$\overline{IOW}$
0	0	X	Never happens

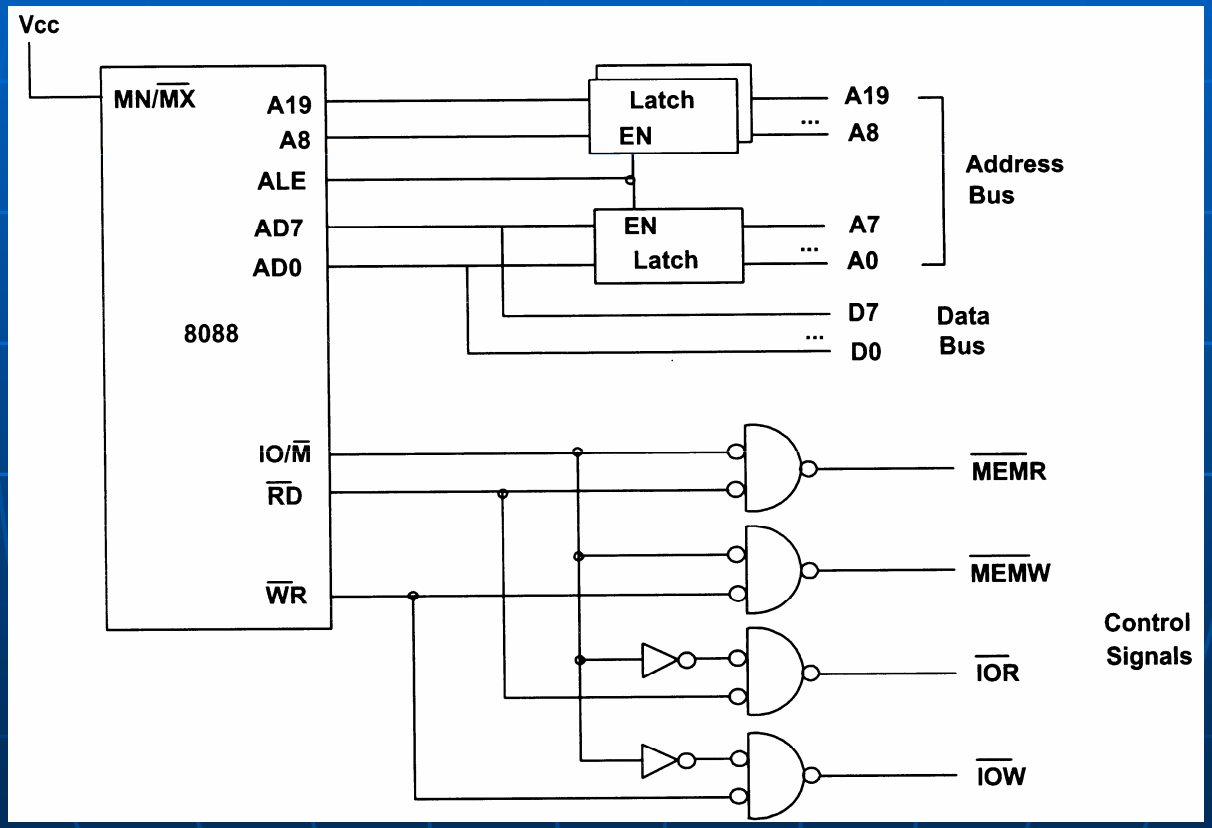
Tạo ra các tín hiệu điều khiển (Min Mode)



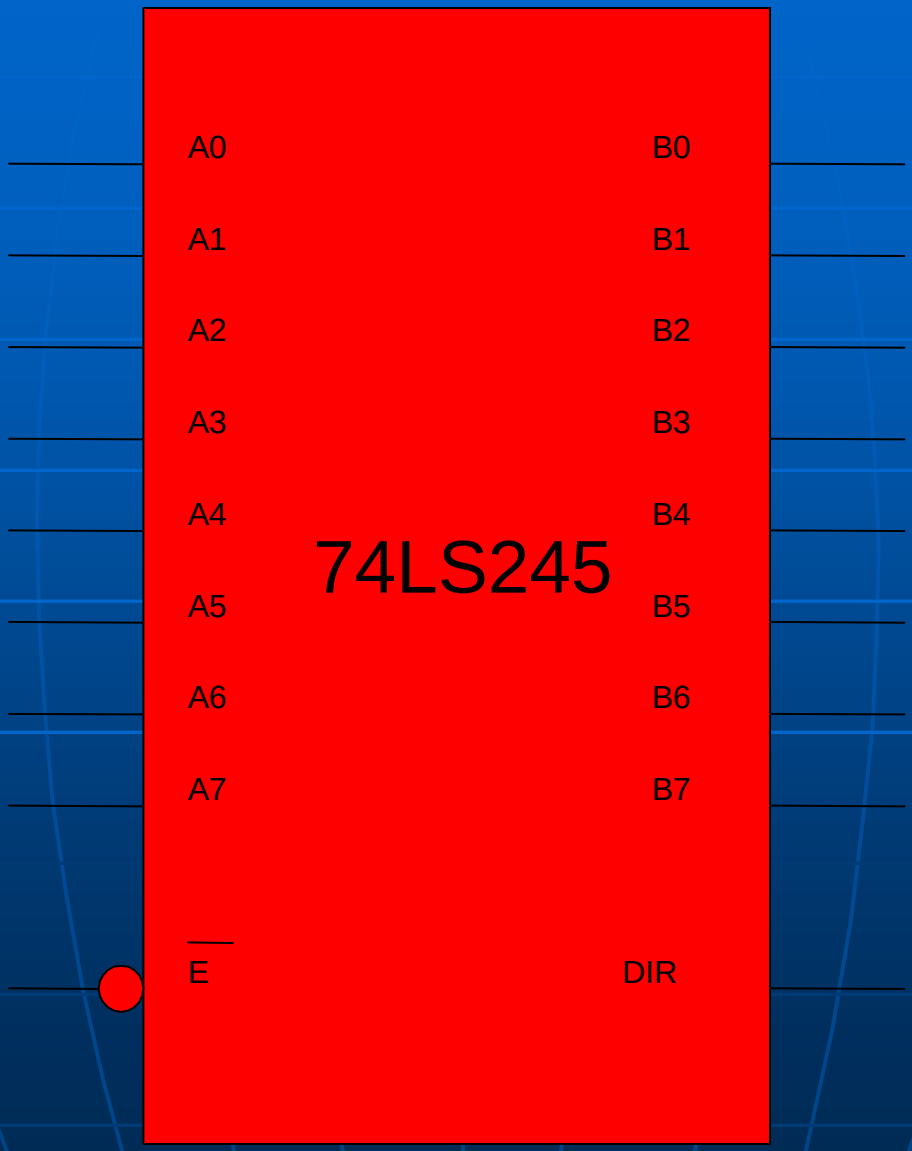
Tạo ra các tín hiệu điều khiển (Min Mode)



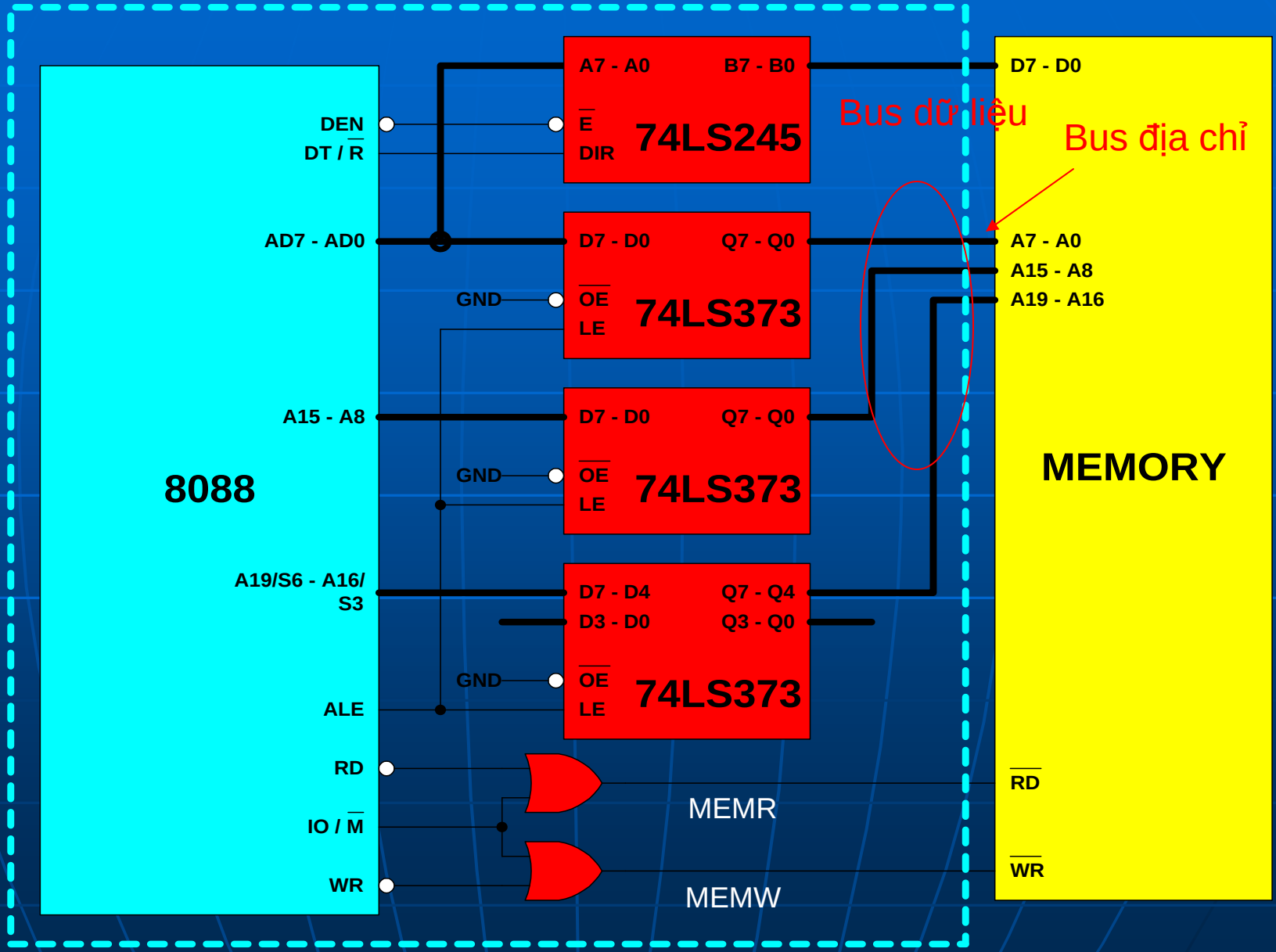
8088 Bus – Min Mode



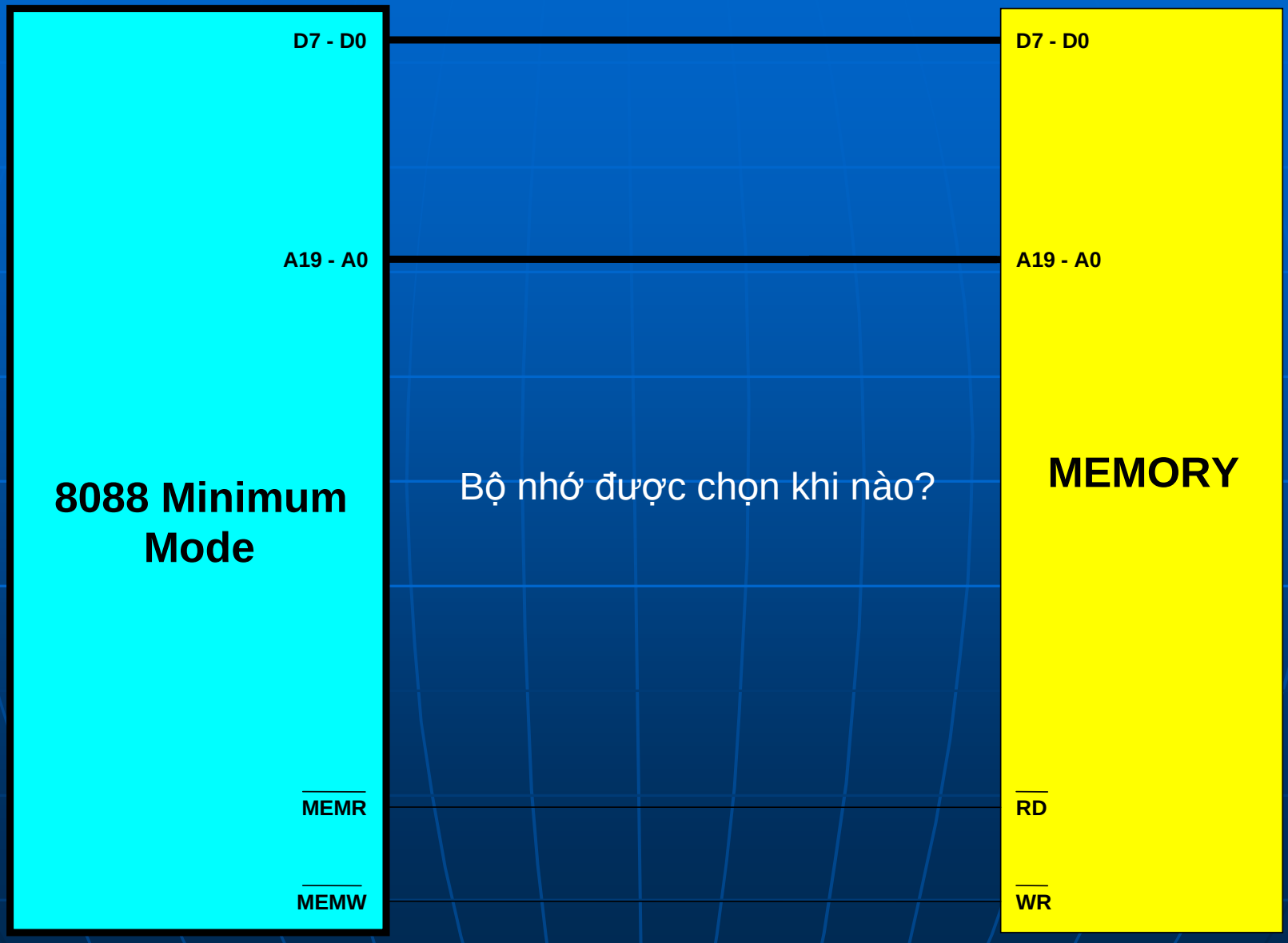
74LS245



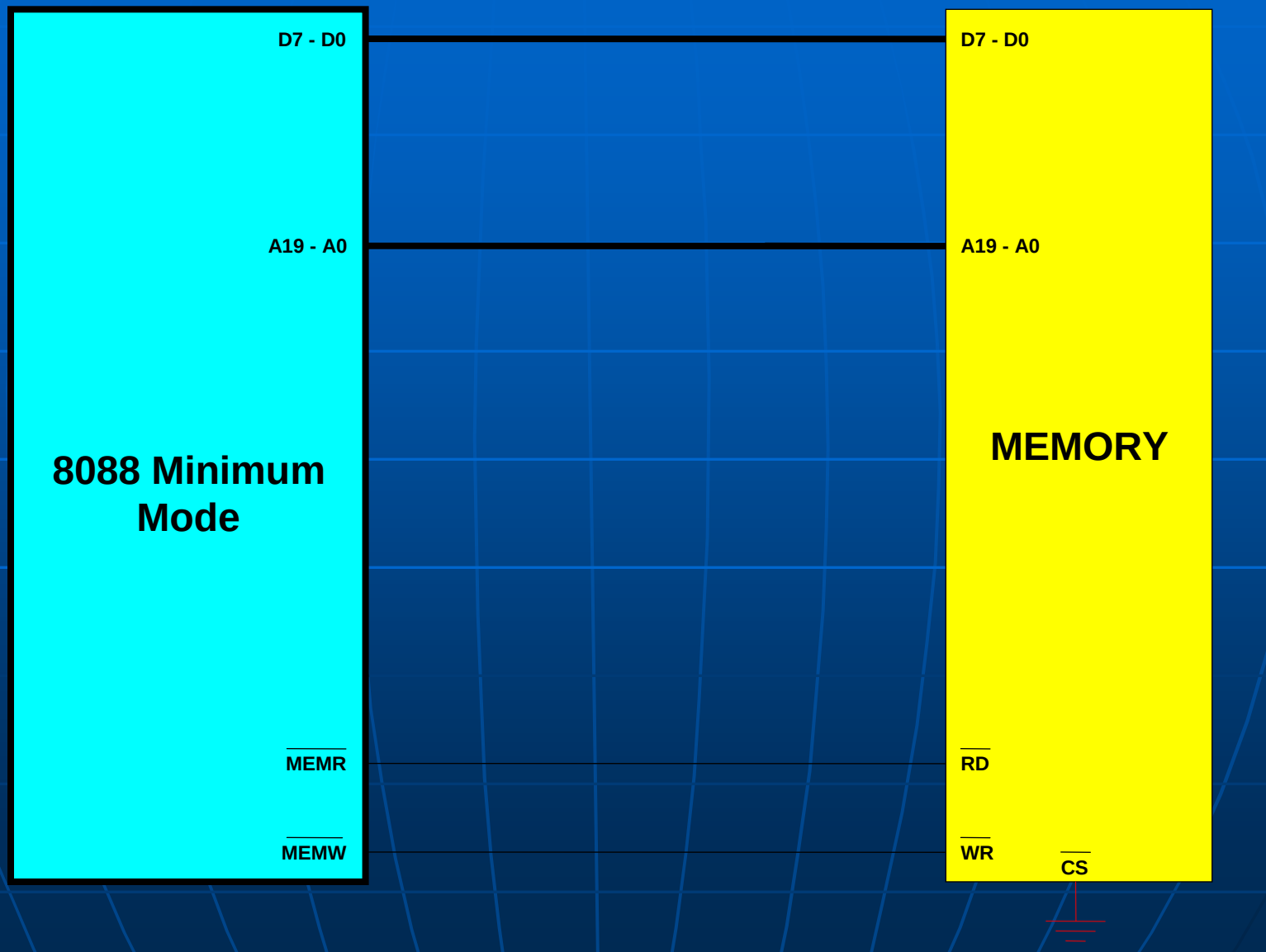
Bus hệ thống của hệ 8088 ở Mode Minimum



Minimum Mode 8088



Minimum Mode 8088



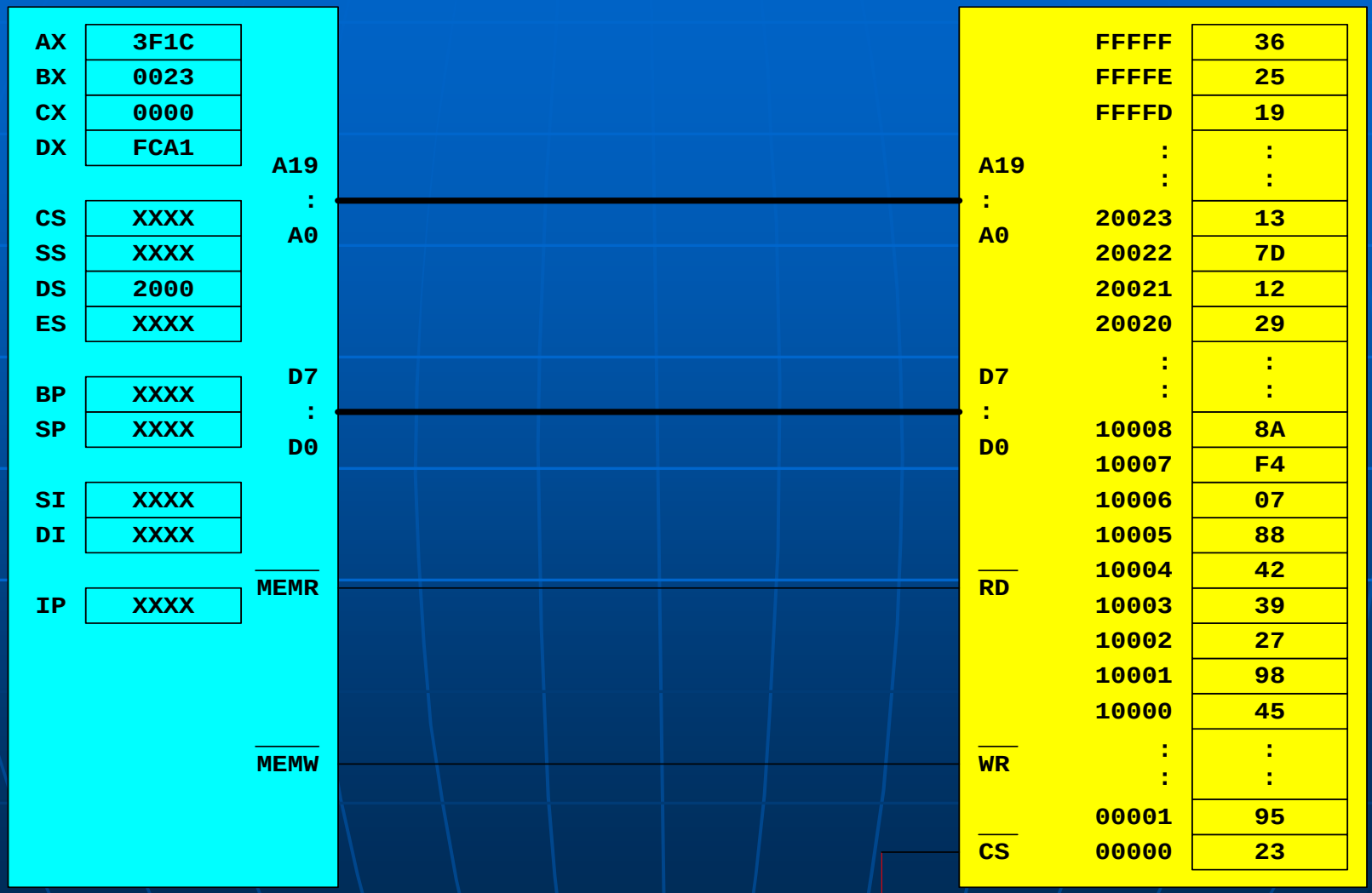
Không gian địa chỉ bộ nhớ 1M

A19 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

Ví dụ: Một địa chỉ bất kỳ 34FD0h

0011 0100 11111 1101 0000

Bộ nhớ đầy đủ 1MB



Nếu chỉ cần bộ nhớ có dung lượng nhỏ hơn 1MB thì giải quyết như thế nào?

- *Phụ thuộc vào các chip nhớ sẵn có*
- *Phụ thuộc yêu cầu phân bố địa chỉ cho các loại bộ nhớ vật lý khác nhau*
- *…*

512K đầu tiên của không gian địa chỉ bộ nhớ (Các địa chỉ có bit cao nhất A19 = 0)

A18 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111

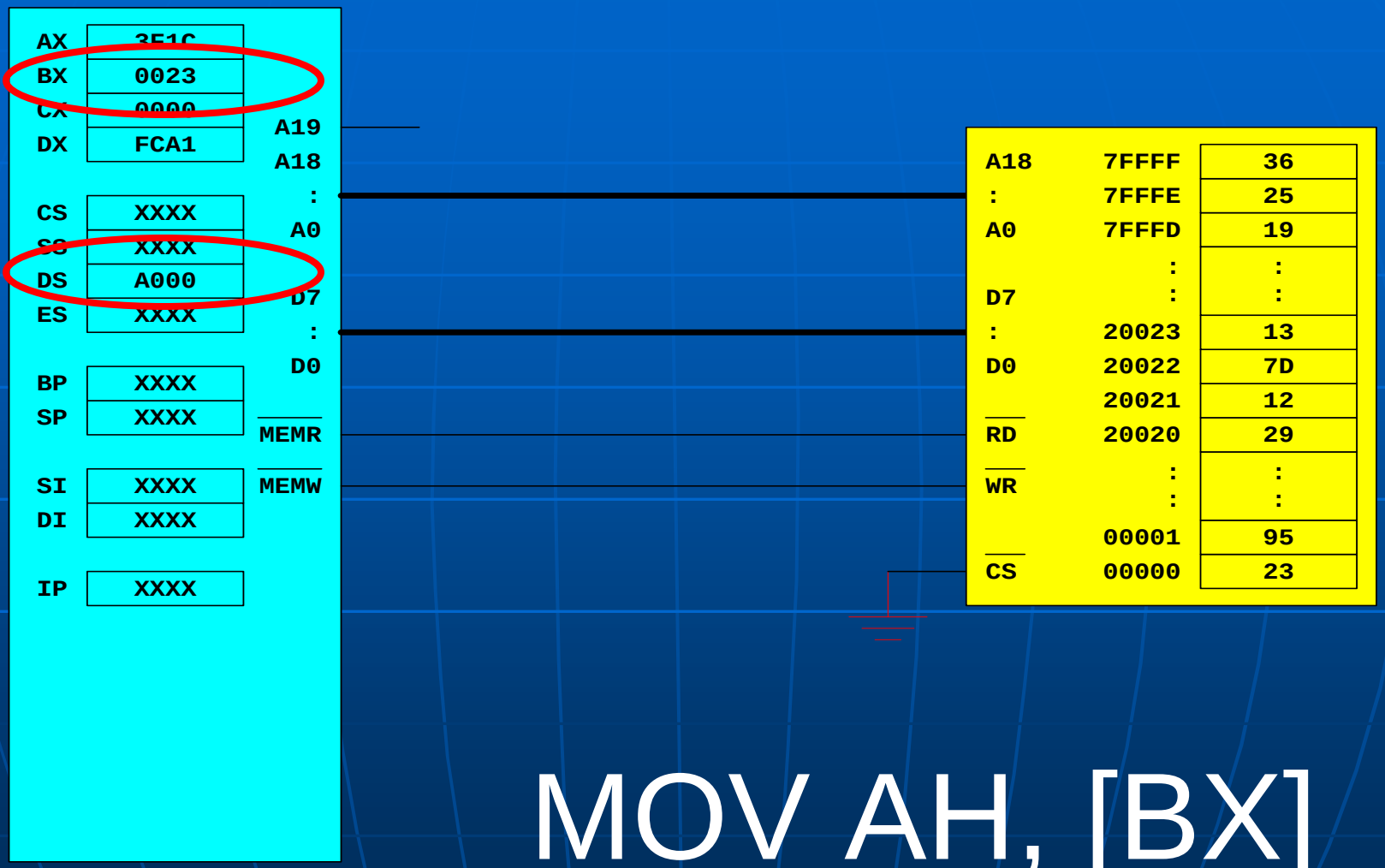
512K tiếp theo của không gian địa chỉ bộ nhớ (Các địa chỉ có bit cao nhất A19 = 1)

A18 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

Bộ nhớ 512KB


AX 3F1C
BX 0023
CX 0000
DX FCA1
CS XXXX
SS XXXX
DS 2000
ES XXXX
BP XXXX
SP XXXX
SI XXXX
DI XXXX
IP XXXX
A19
A18
:
A0
D7
:
D0
MEMR
MEMW
Làm gì với A19?
A18
:
A0
D7
:
RD
WR
CS
7FFFF 36
7FFFE 25
7FFFD 19
20023 13
20022 7D
20021 12
20020 29
00001 95
00000 23

Điều gì xảy ra nếu 8088 đọc ô nhớ A0023h?



MOV AH, [BX]

Điều gì xảy ra nếu 8088 đọc ô nhớ A0023h?

A19 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
A0023	1010	0000	0000	0010	0011

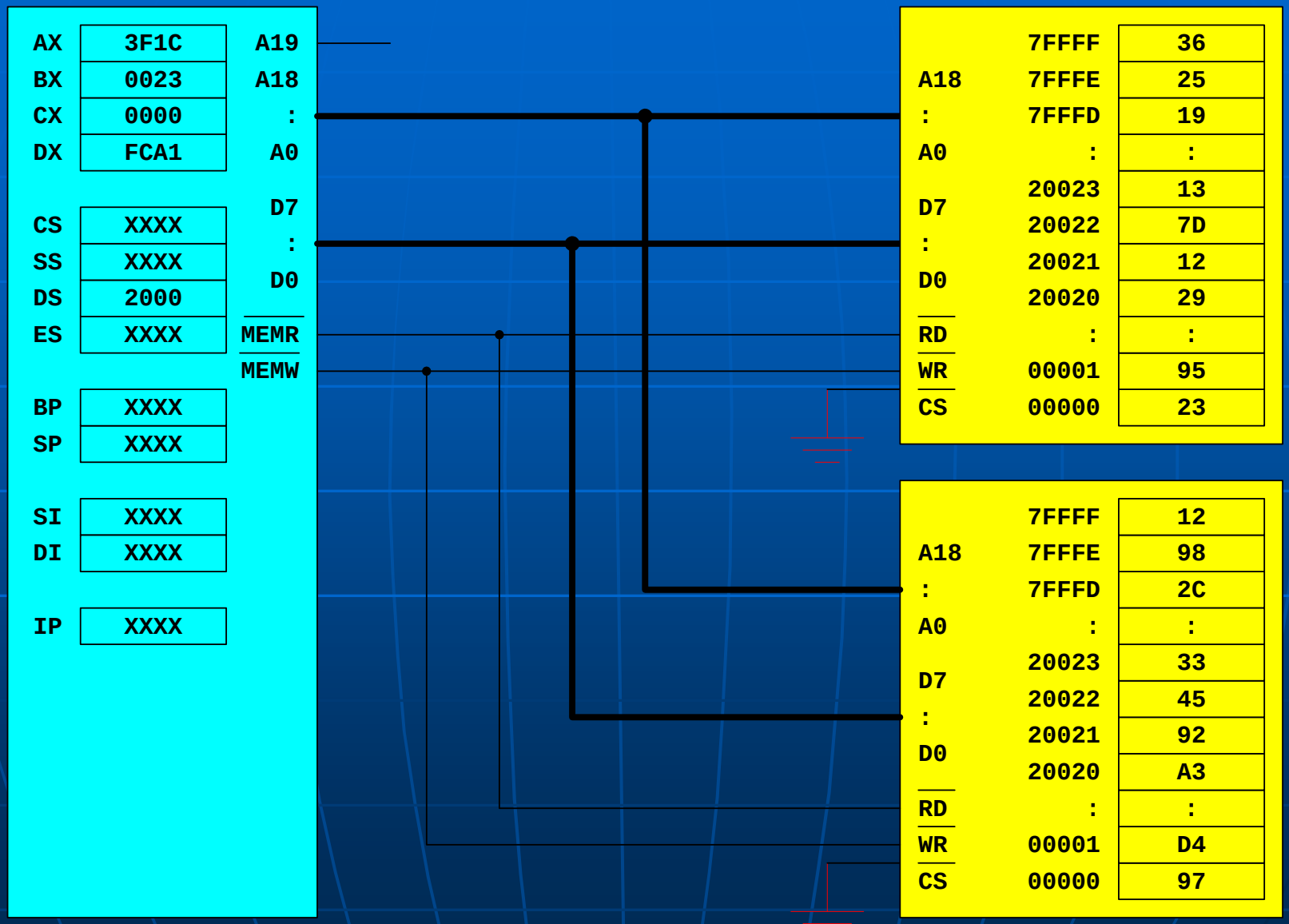
A19 không được nối đến bộ nhớ nên nếu 8088 phát logic "1" trên A19 thì bộ nhớ cũng không nhận biết được.

Điều gì xảy ra nếu 8088 đọc ô nhớ 20023h?

A18 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
20023	0010	0000	0000	0010	0011

Với bộ nhớ tình hình không có gì khác!

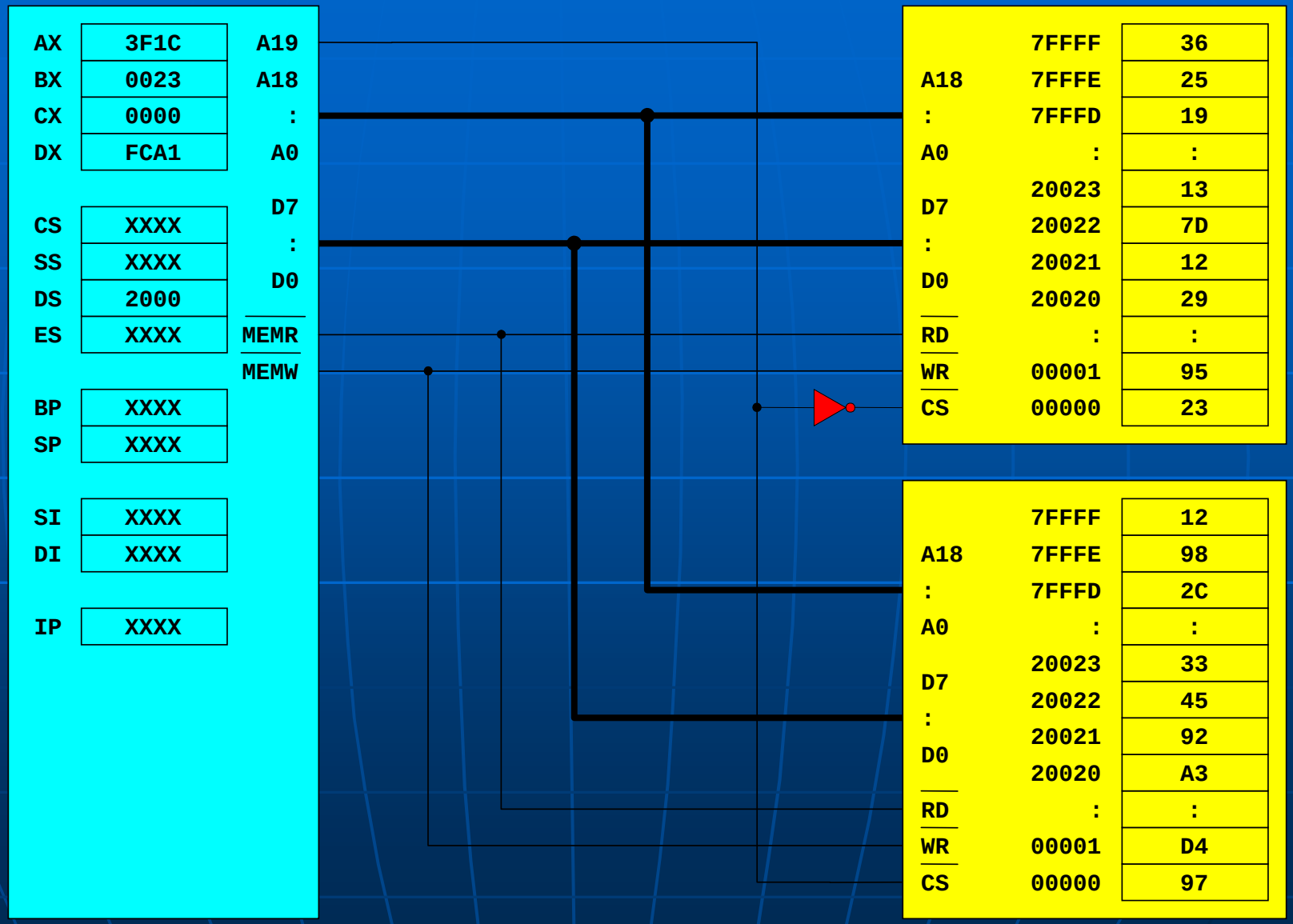
Nếu Bộ nhớ gồm 2 khối 512KB như thế này?



Có vấn đề !!!

- Vấn đề là: Xung đột Bus. Hai khối nhớ sẽ cung cấp dữ liệu cùng một lúc khi 8088 đọc bộ nhớ
- Giải pháp: Dùng A19 làm "người phân xử" để giải quyết xung đột trên bus. Nếu A19 ở mức logic "1" thì khối nhớ trên hoạt động (khối nhớ dưới bị cấm) và ngược lại

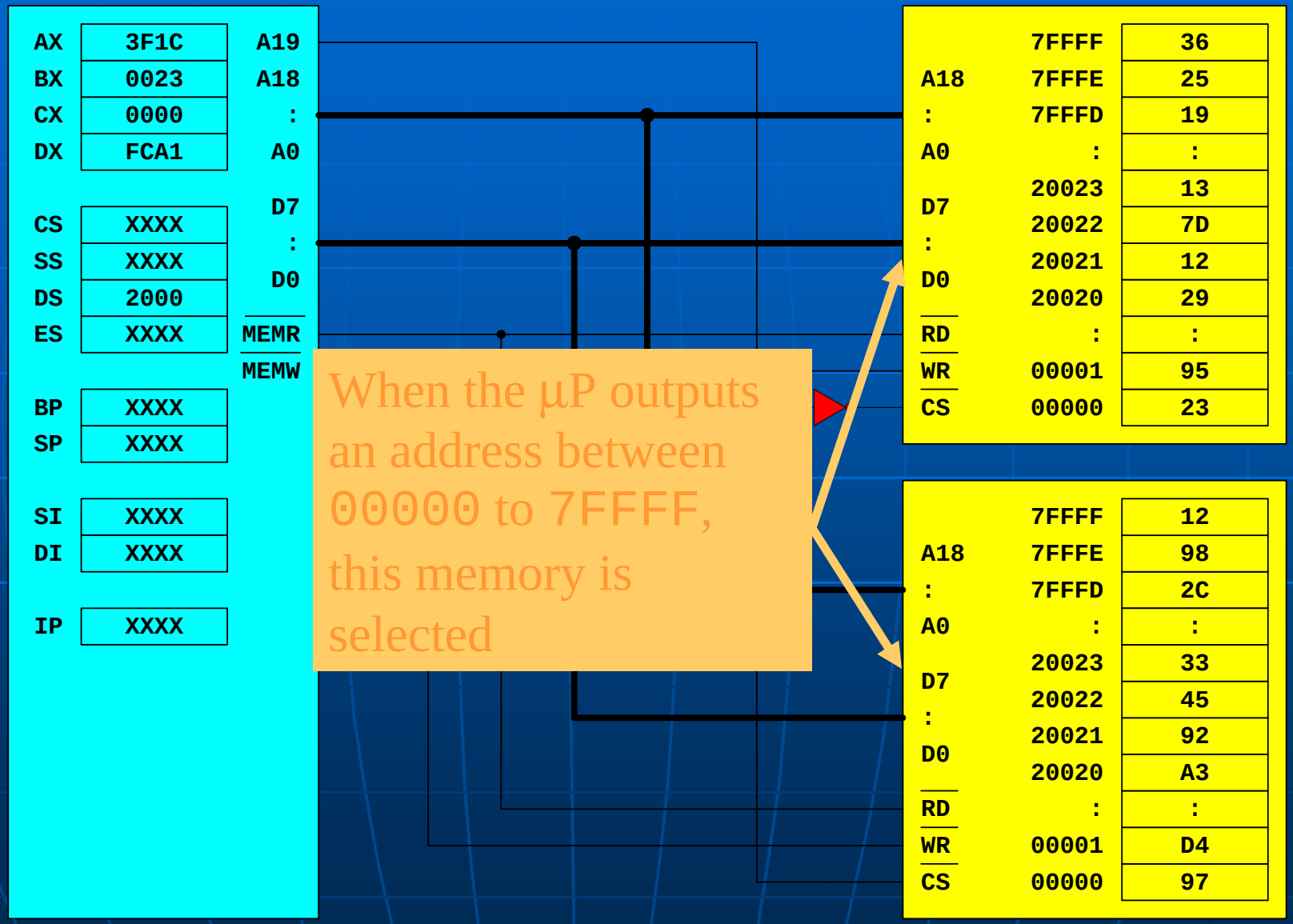
Bộ nhớ gồm hai khối nhớ 512KB



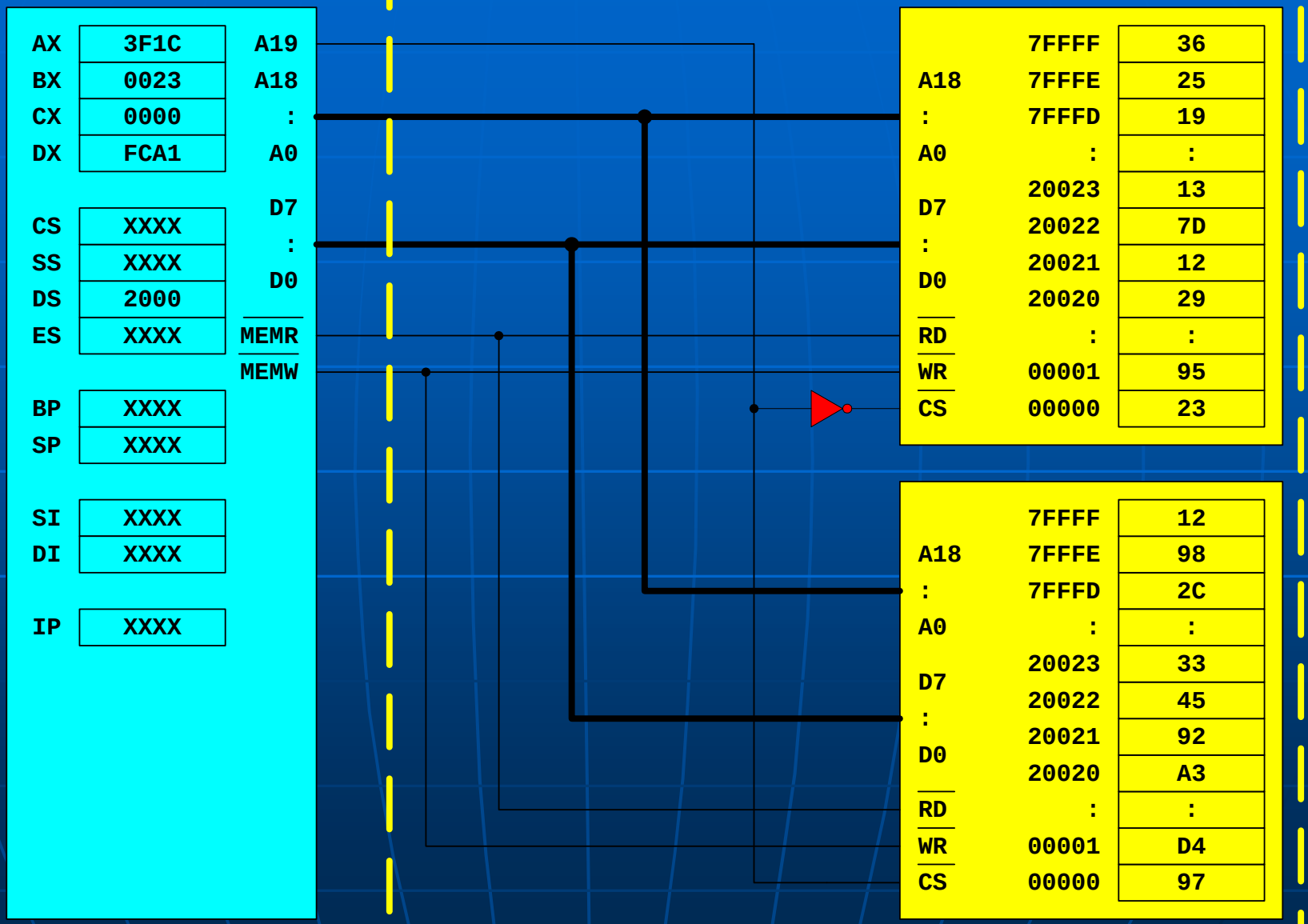
Không gian địa chỉ bộ nhớ 1M

A19 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

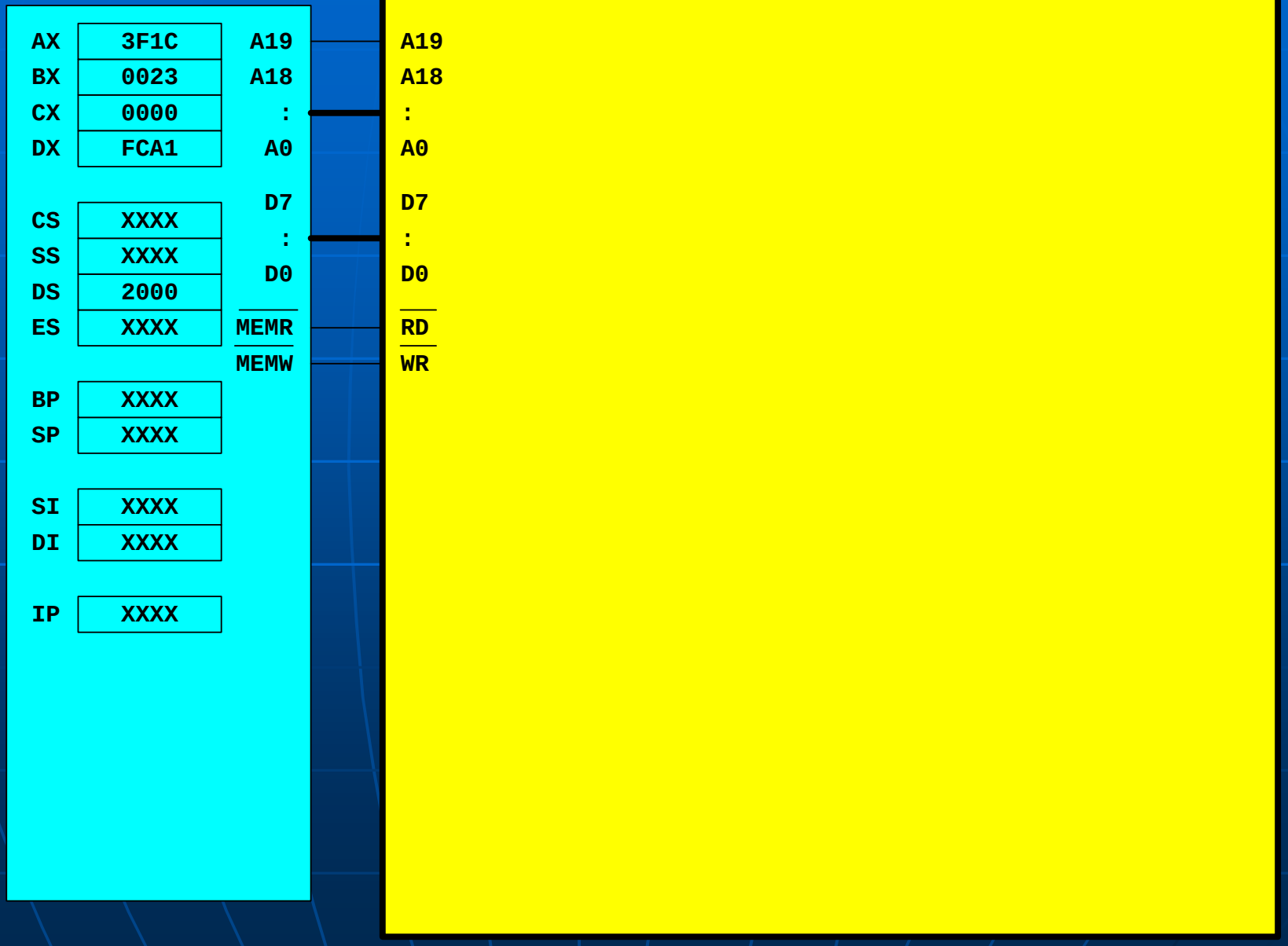
Interfacing **two** 512KB Memory to the 8088 Microprocessor



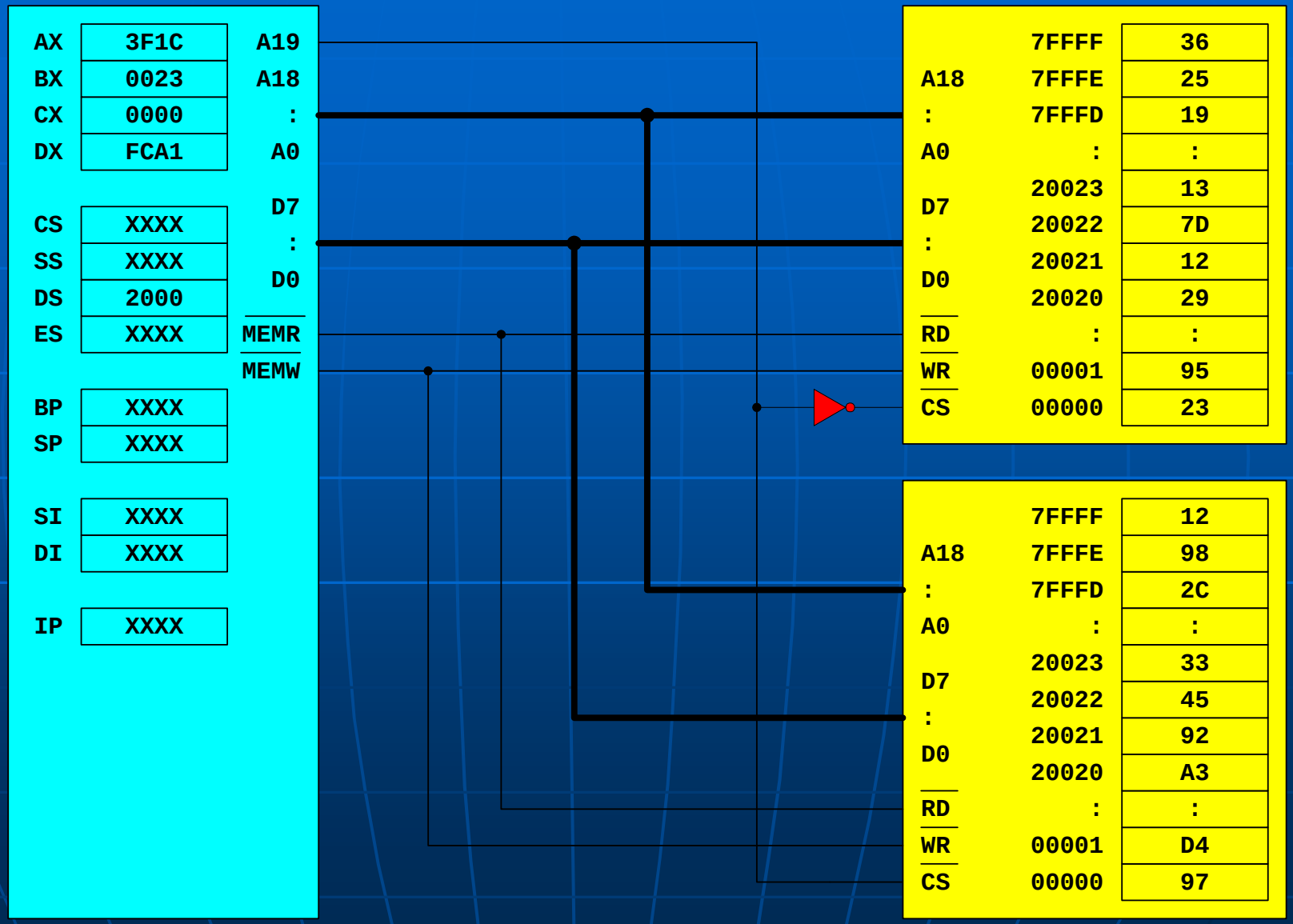
Interfacing **two** 512KB Memory to the 8088 Microprocessor



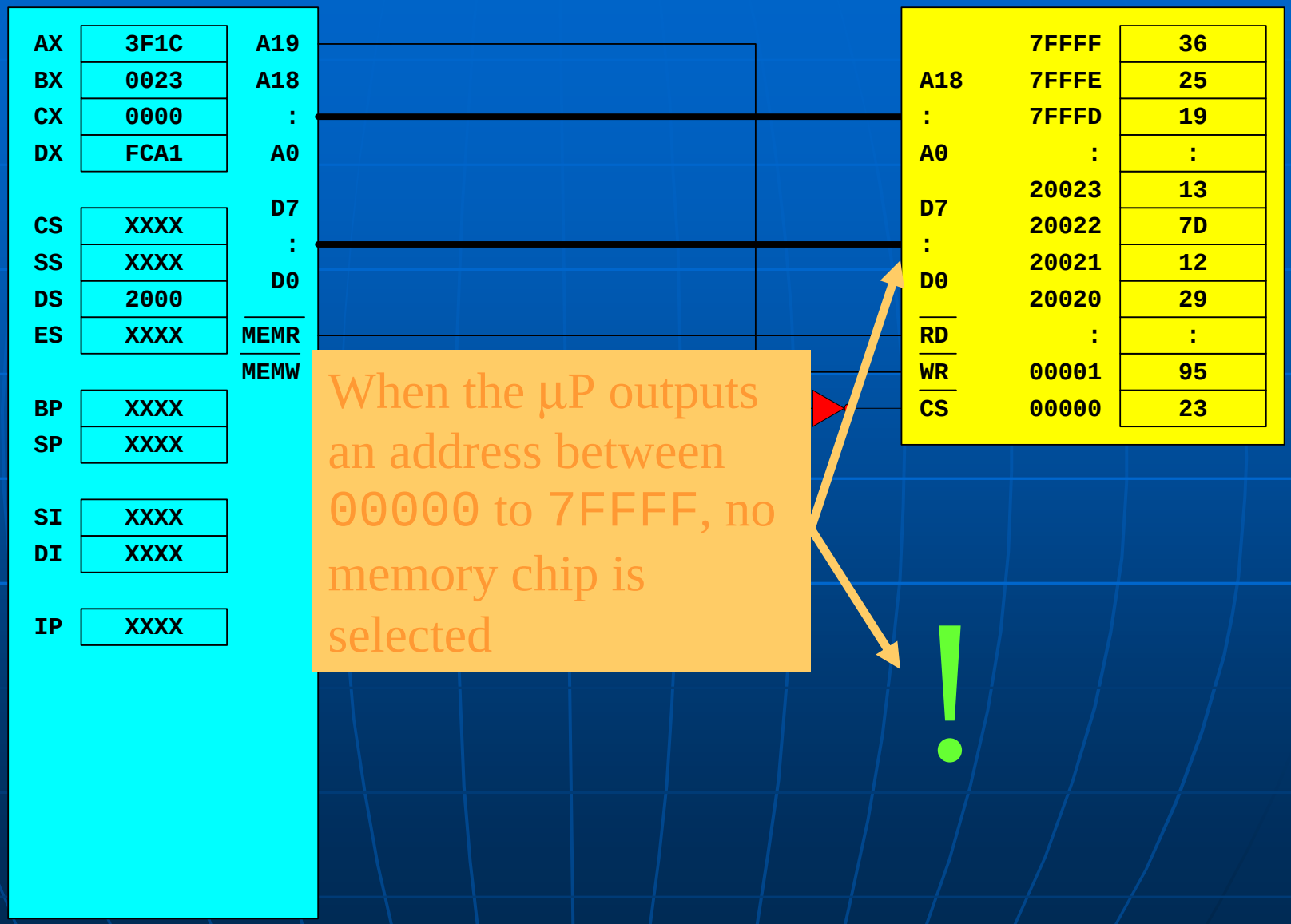
Interfacing **two** 512KB Memory to the 8088 Microprocessor



What if we remove the lower memory?



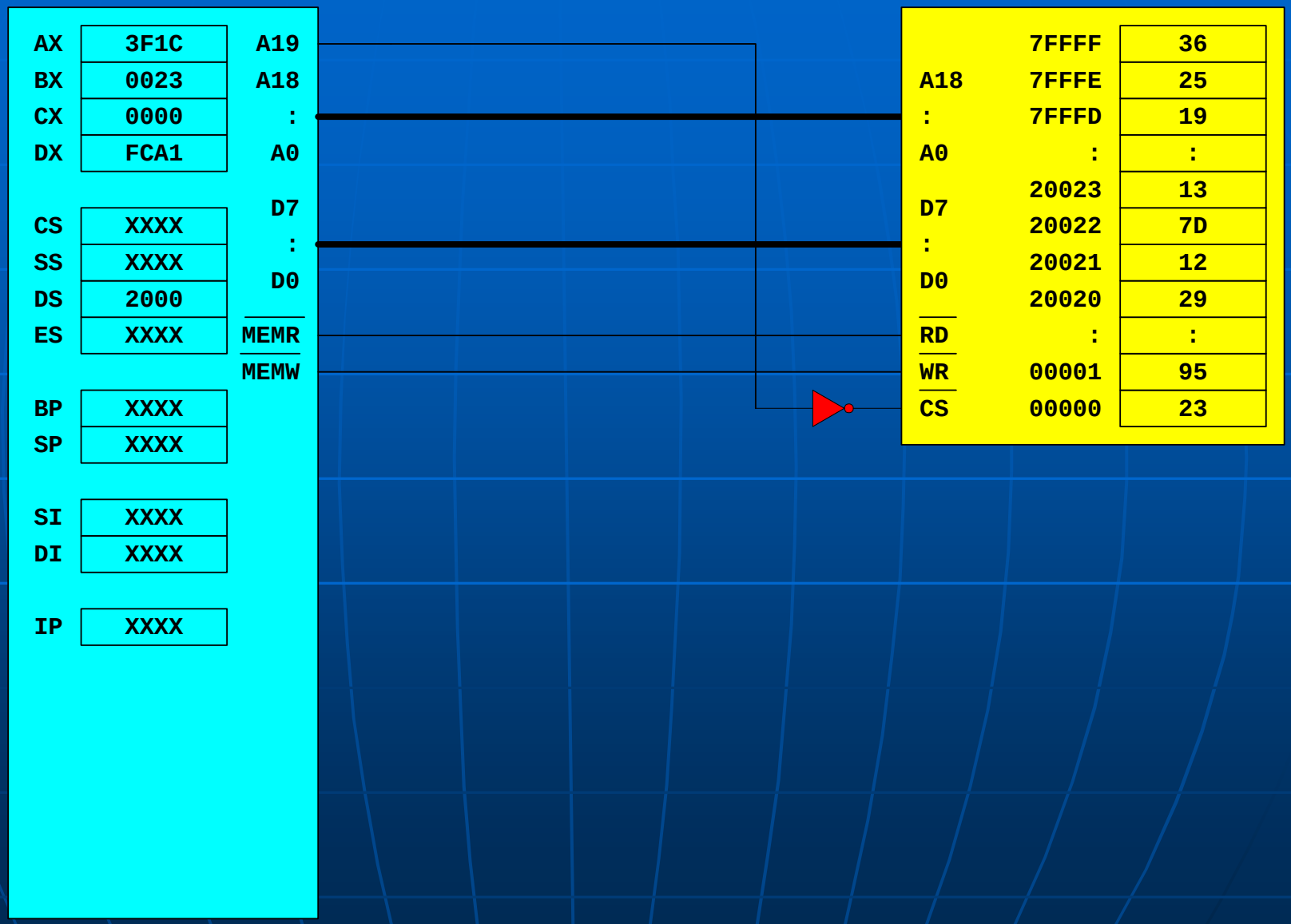
What if we remove the lower memory?



Full and Partial Decoding

- Full Decoding
  - When all of the "useful" address lines are connected the memory/device to perform selection
- Partial Decoding
  - When some of the "useful" address lines are connected the memory/device to perform selection
  - Using this type of decoding results into roll-over addresses

Full Decoding



Full Decoding

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

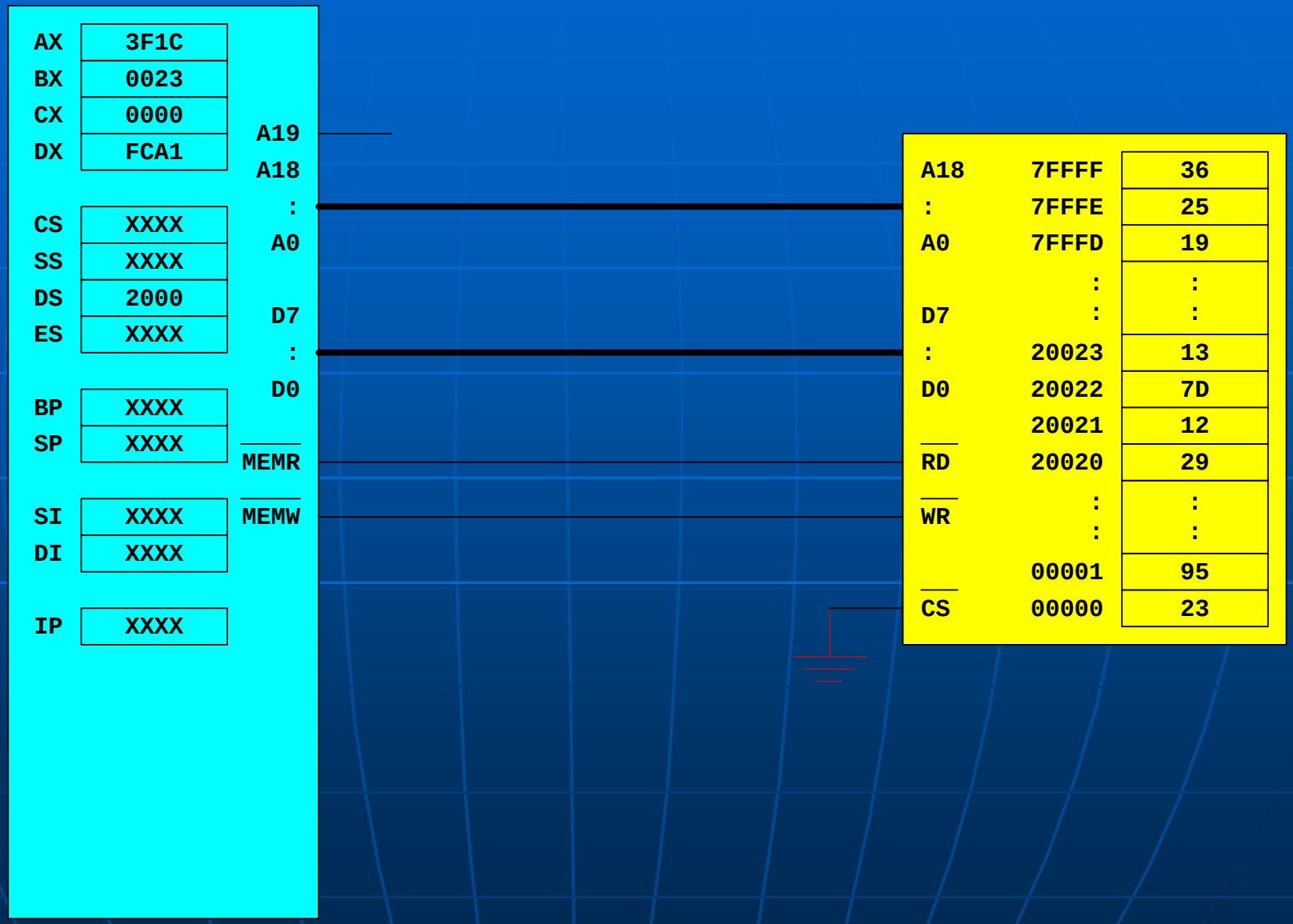
A19 should be a logic “1” for the memory chip to be enabled

Full Decoding

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111

Therefore if the microprocessor outputs an address between 00000 to 7FFFF, whose A19 is a logic “0”, the memory chip will not be selected

Partial Decoding



Partial Decoding

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

The value of A19 is INSIGNIFICANT to the memory chip, therefore A19 has no bearing whether the memory chip will be enabled or not

Partial Decoding

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

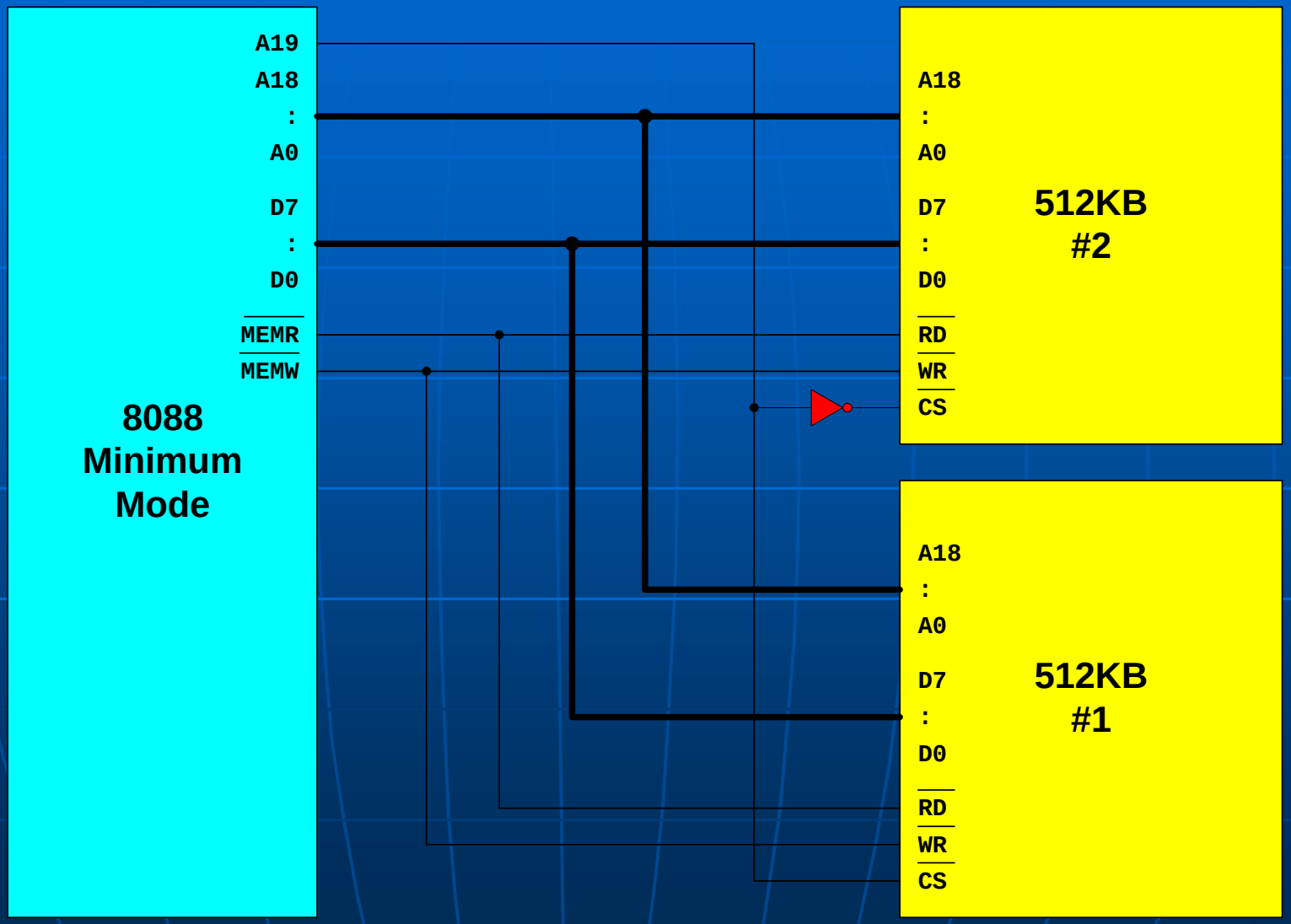
ACTUAL ADDRESS

Partial Decoding

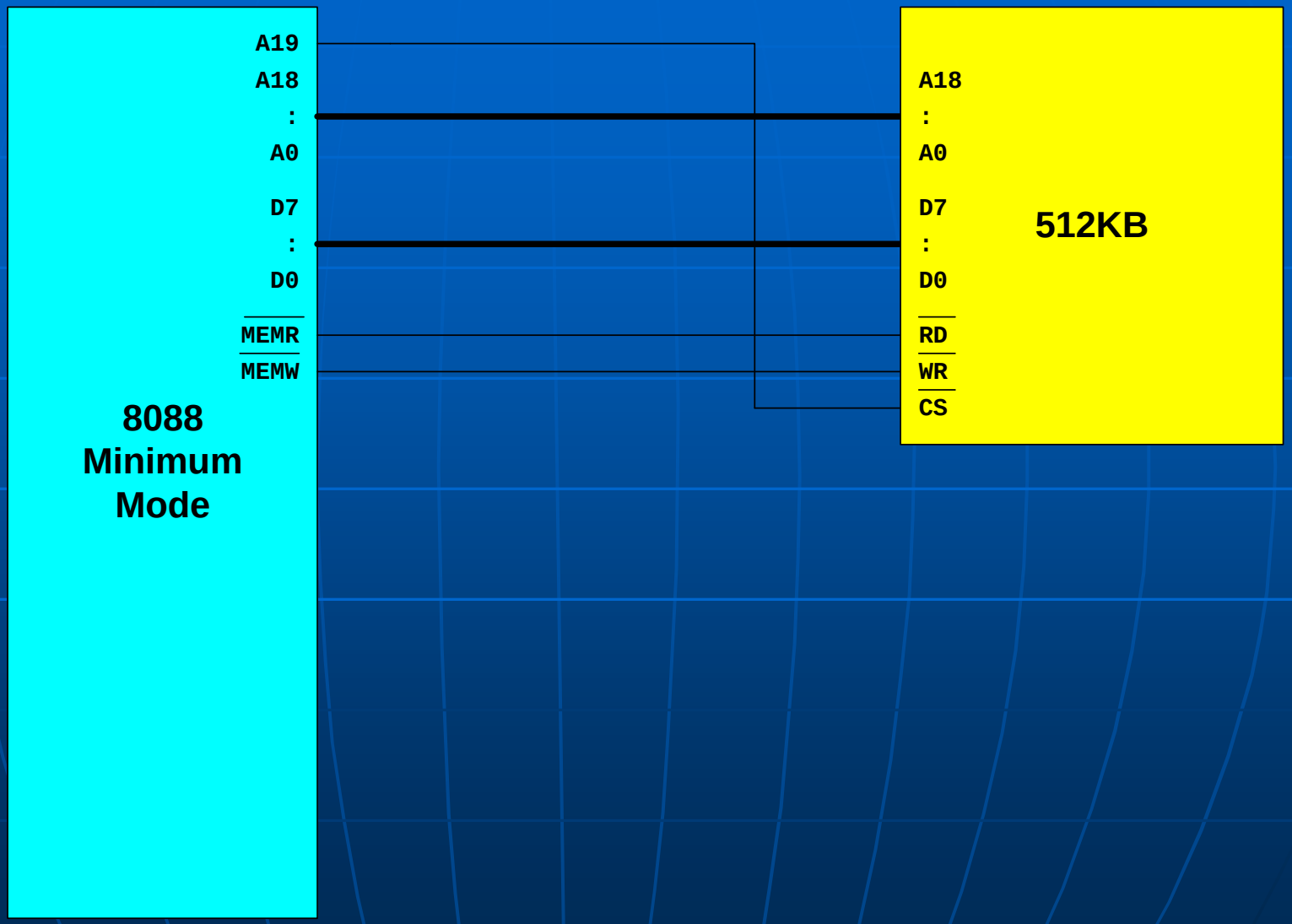
A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

ACTUAL ADDRESS

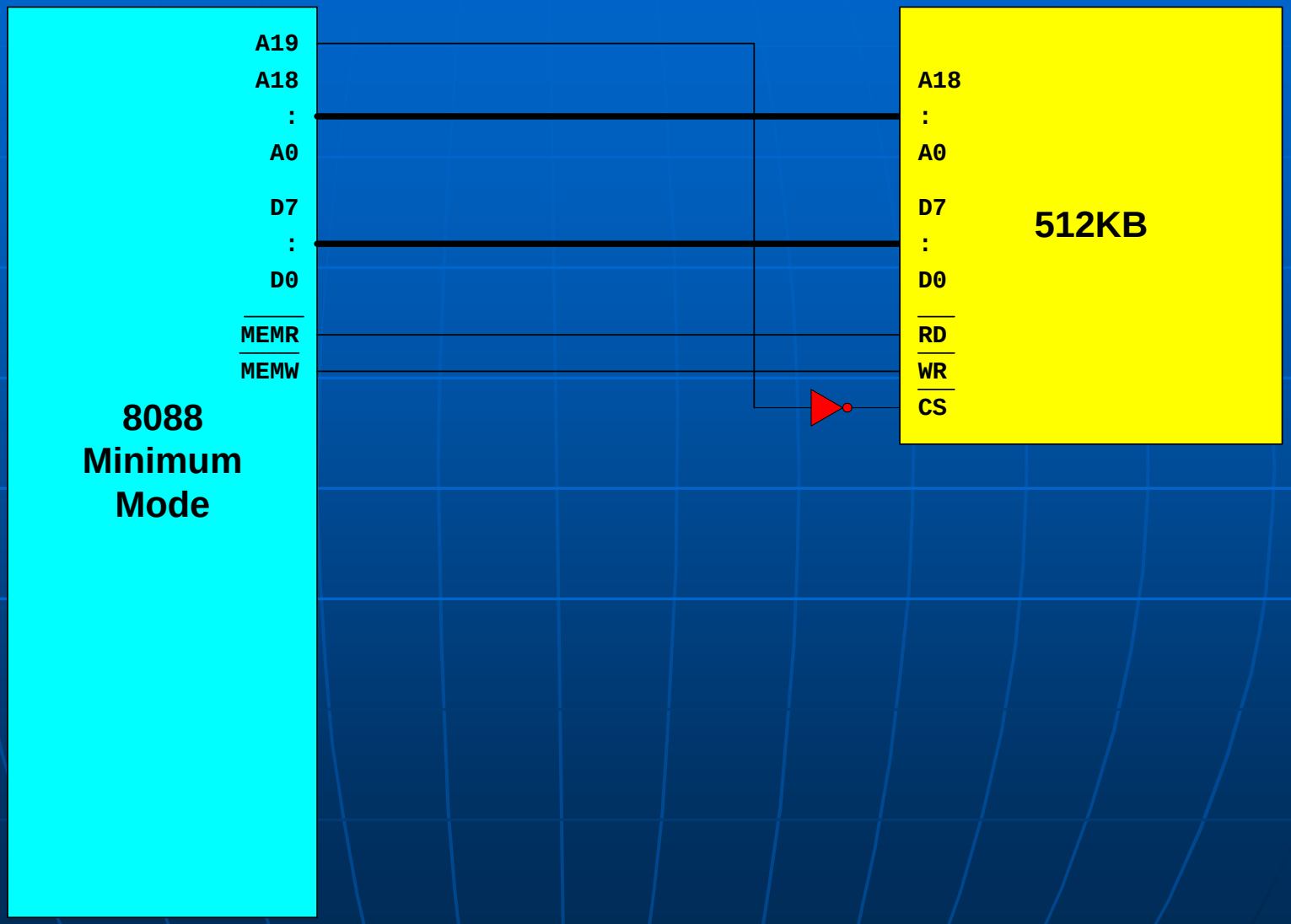
Interfacing two 512K Memory Chips to the 8088 Microprocessor



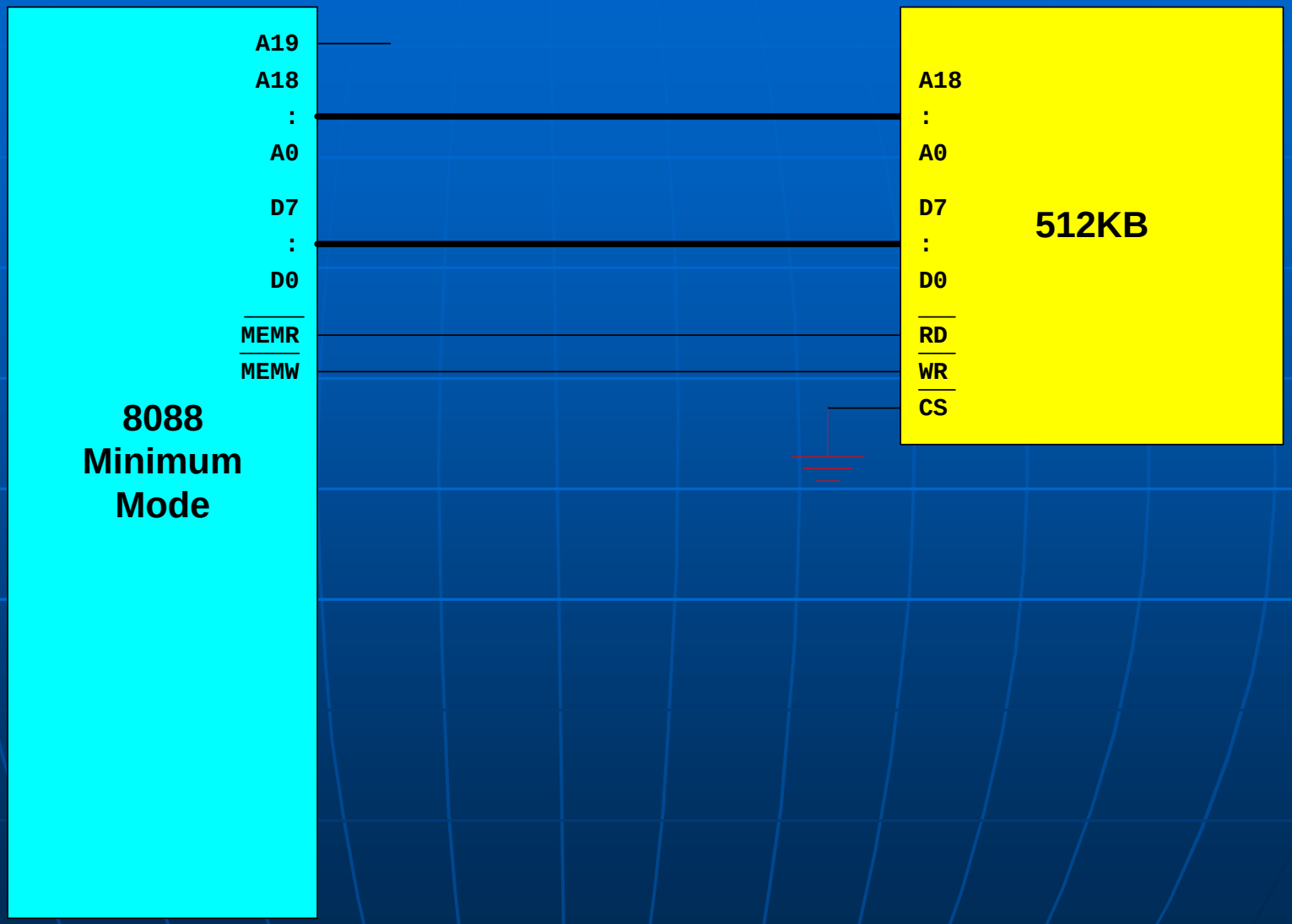
Interfacing one 512K Memory Chips to the 8088 Microprocessor



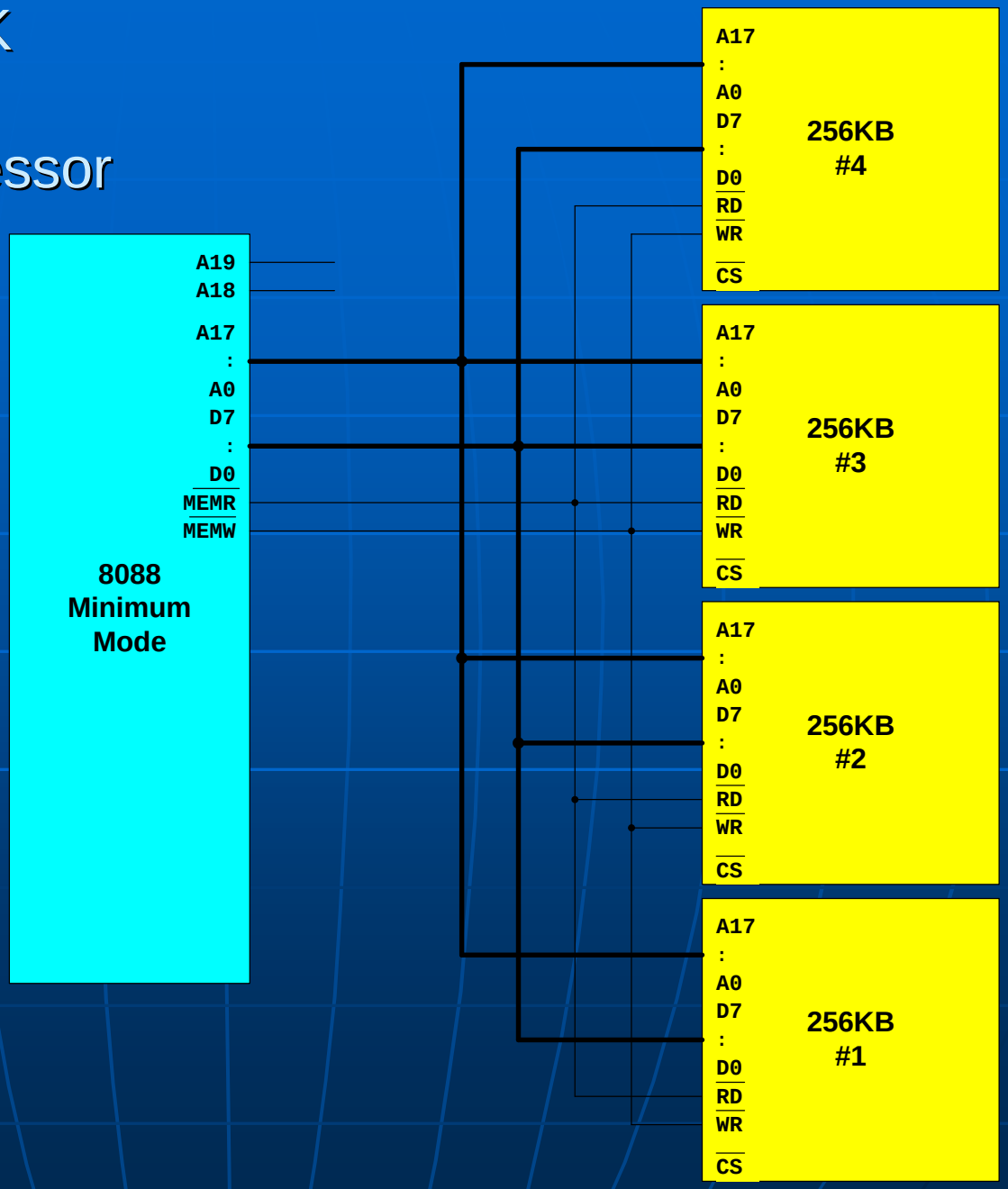
Interfacing one 512K Memory Chips to the 8088 Microprocessor (version 2)



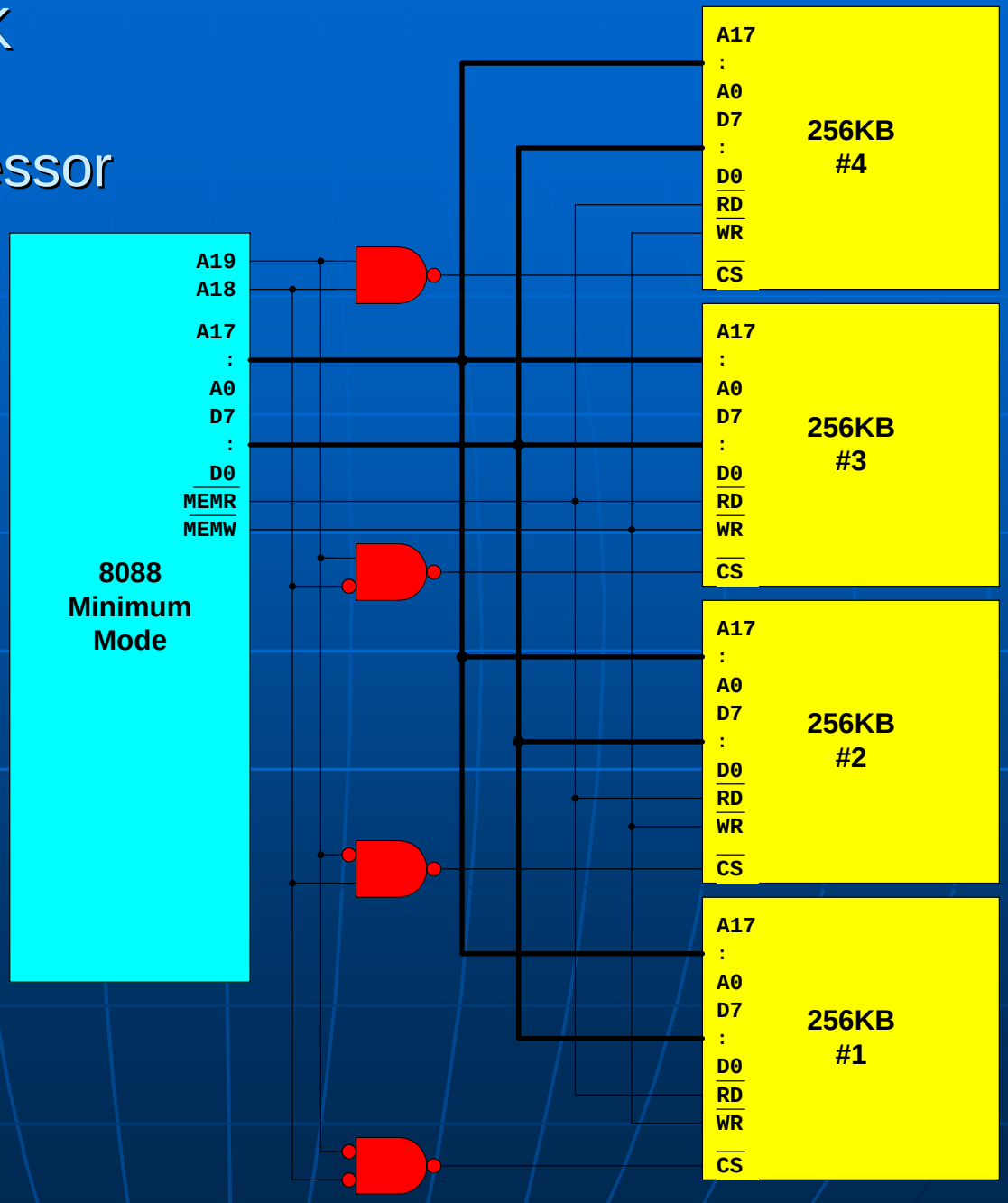
Interfacing one 512K Memory Chips to the 8088 Microprocessor (version 3)



Interfacing four 256K Memory Chips to the 8088 Microprocessor



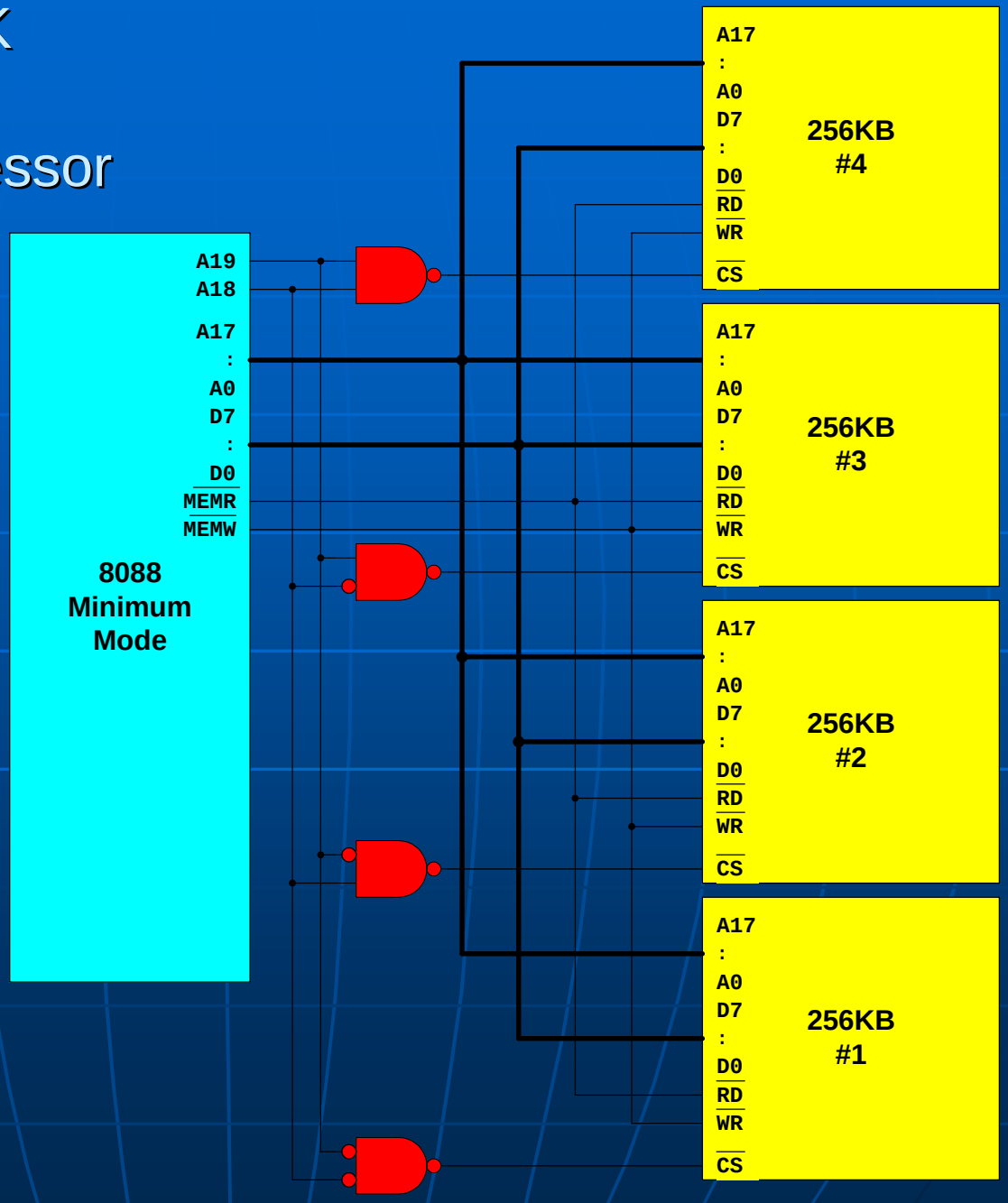
Interfacing four 256K Memory Chips to the 8088 Microprocessor



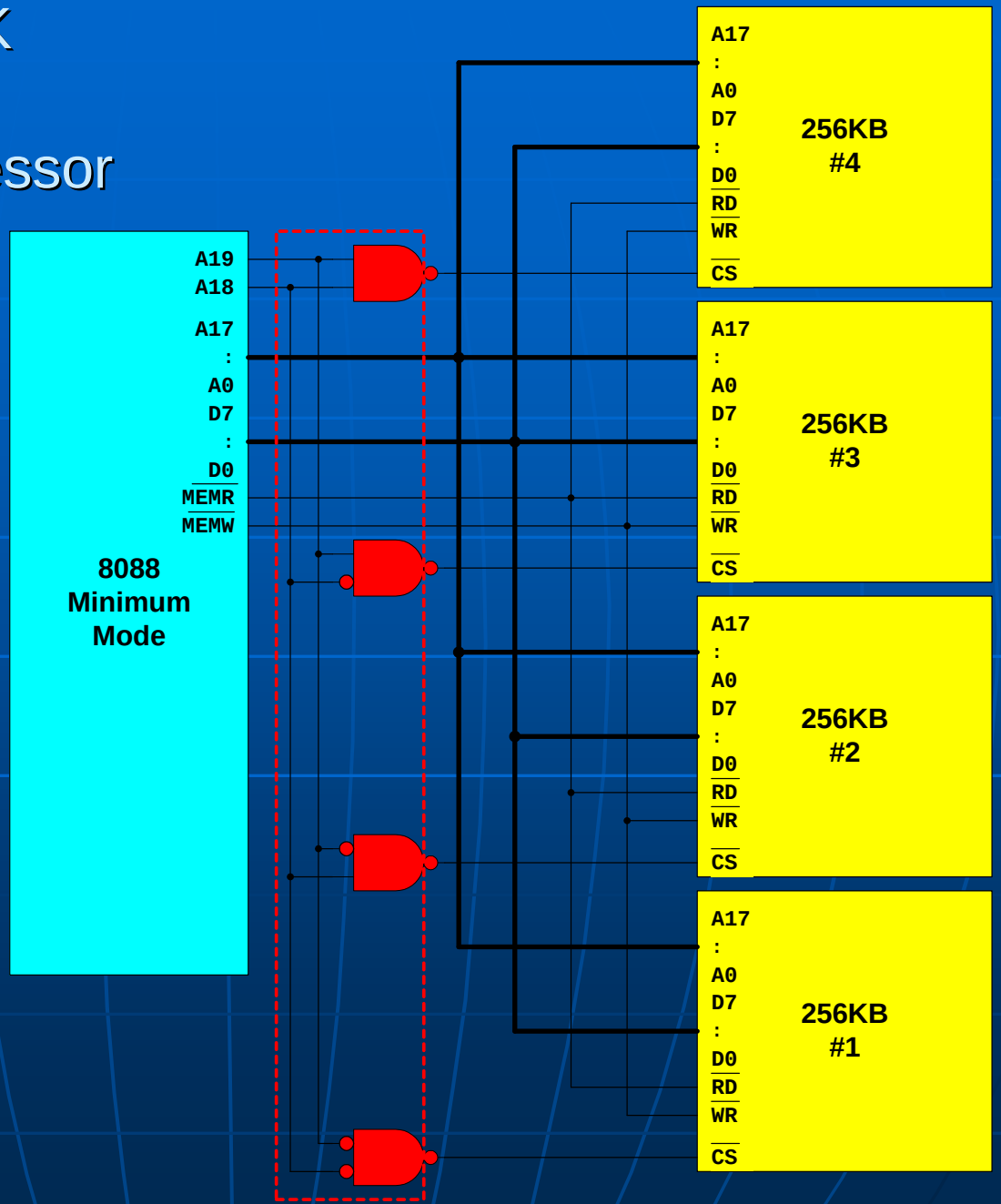
Memory chip#__ is mapped to:

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
-----	----	----	----	----	----
-----	----	----	----	----	----

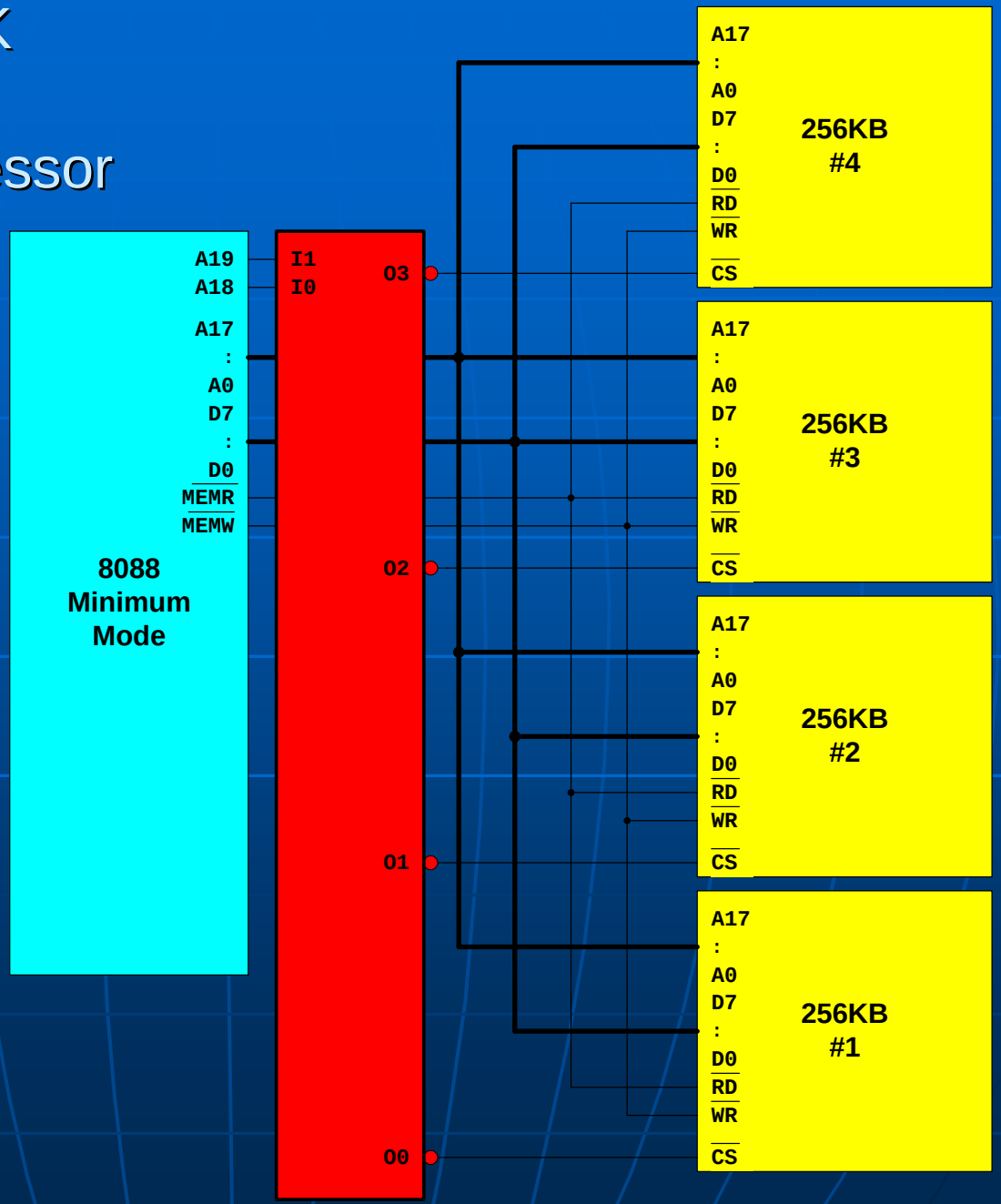
Interfacing four 256K Memory Chips to the 8088 Microprocessor



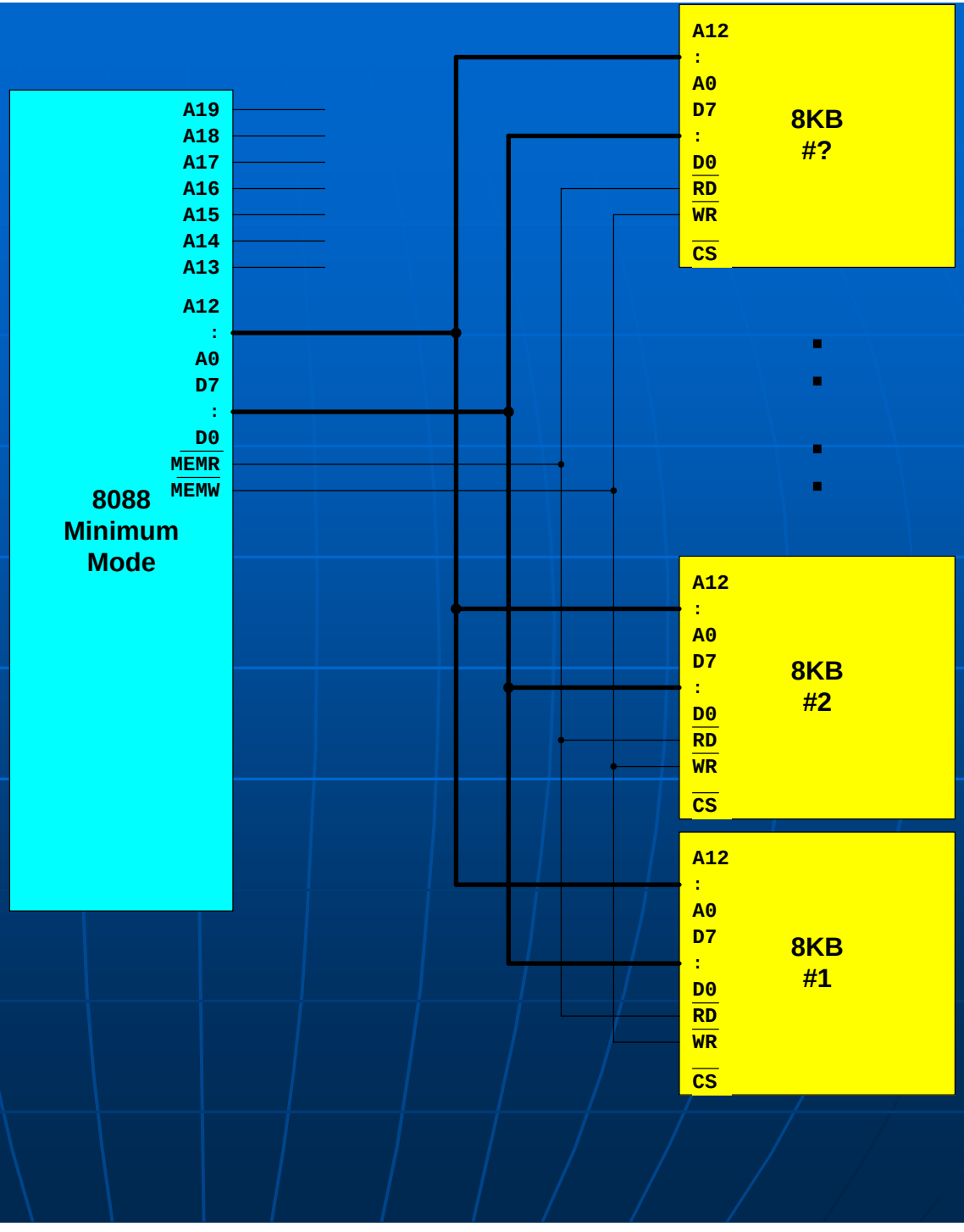
Interfacing four 256K Memory Chips to the 8088 Microprocessor



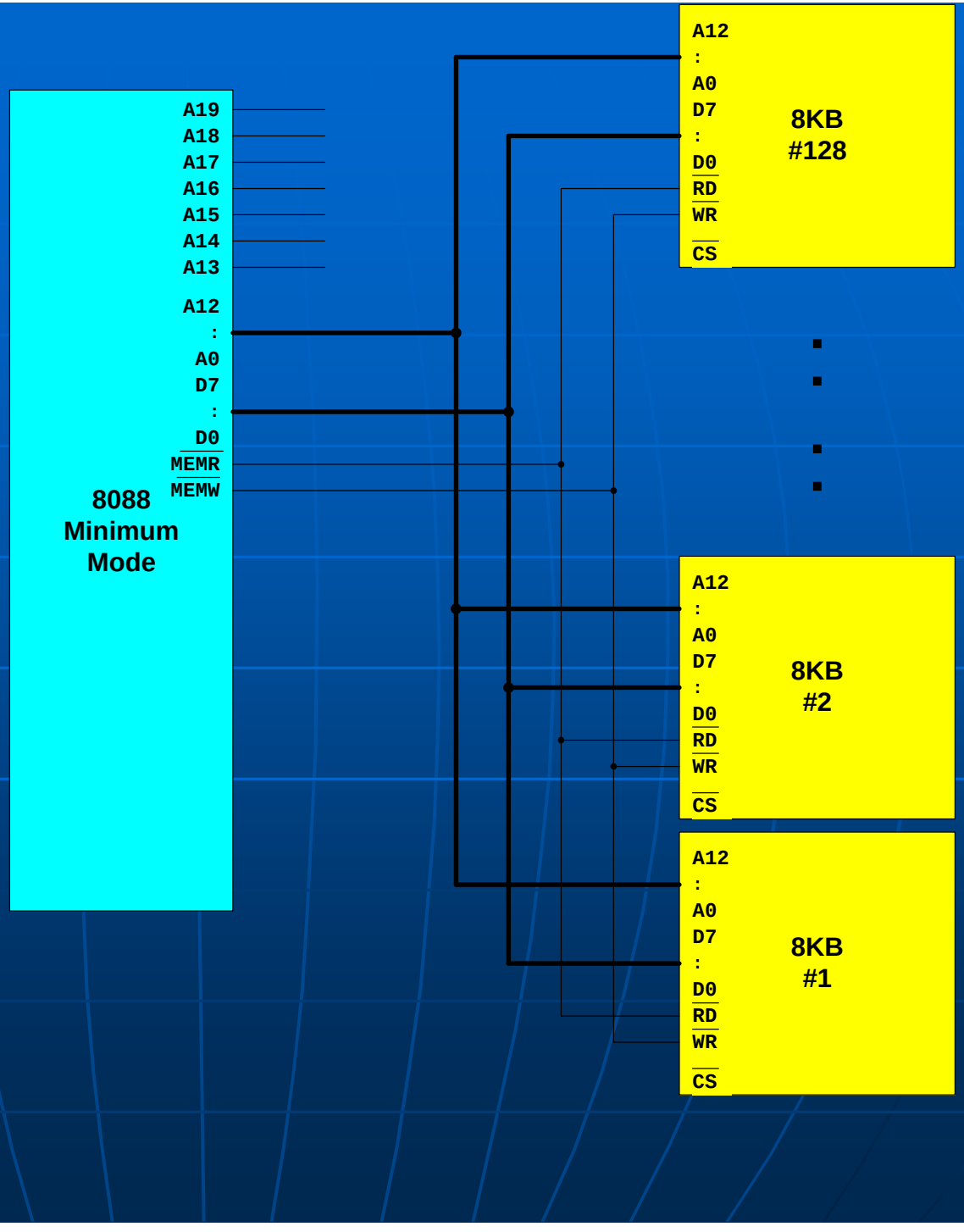
Interfacing four 256K Memory Chips to the 8088 Microprocessor



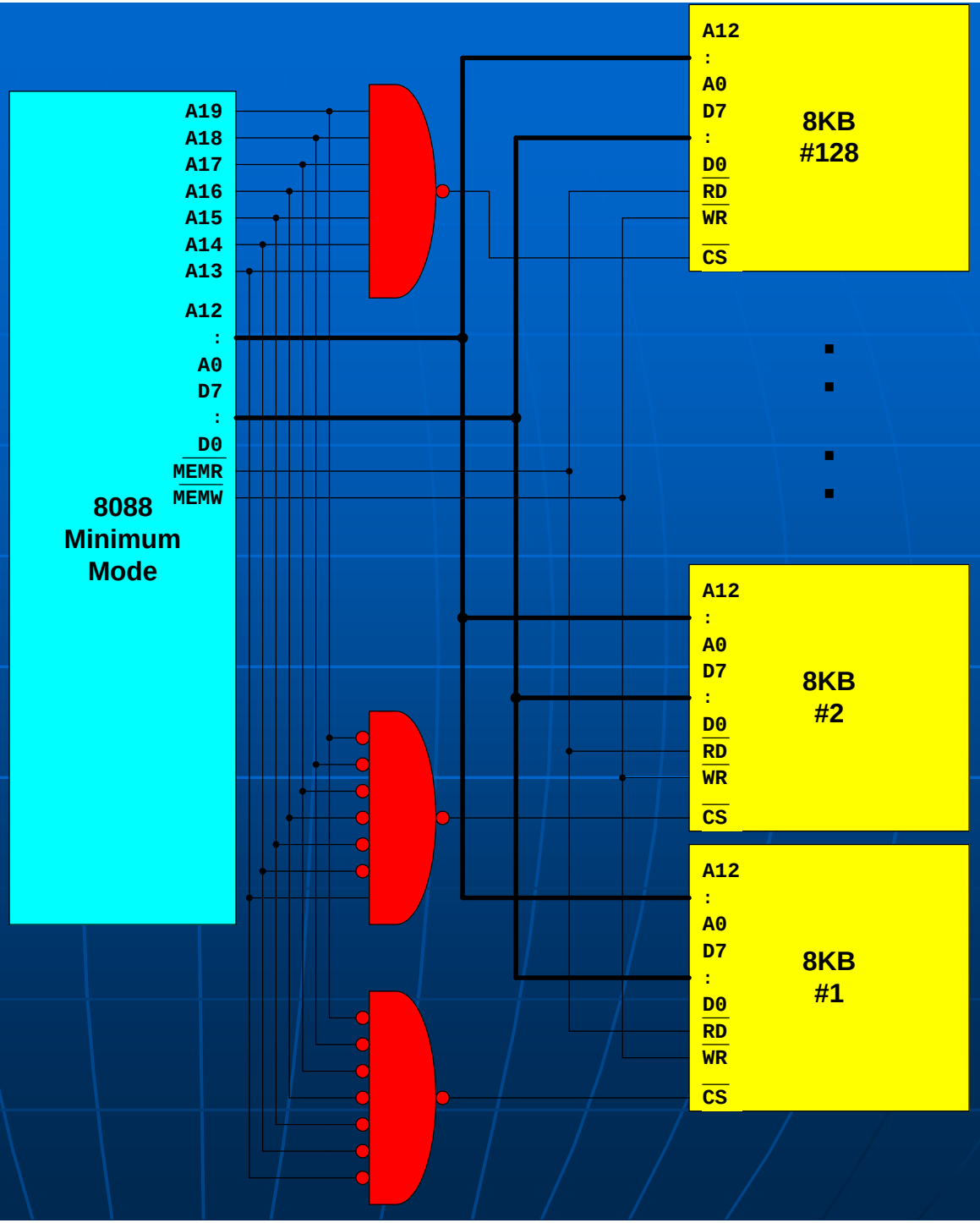
Interfacing several 8K Memory Chips to the 8088 µP



Interfacing 128 8K Memory Chips to the 8088 µP



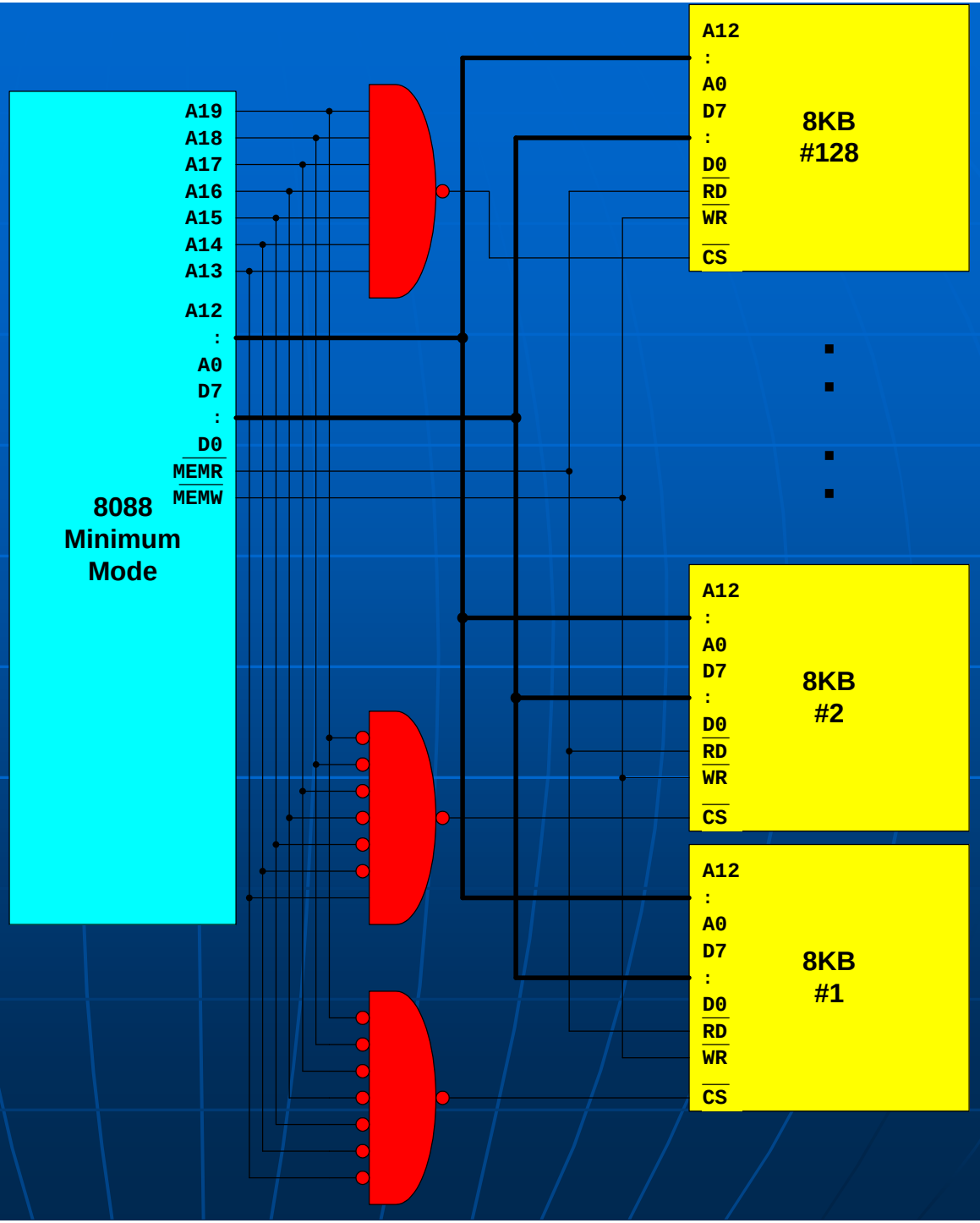
Interfacing 128 8K Memory Chips to the 8088 µP



Memory chip#__ is mapped to:

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
-----	----	----	----	----	----
-----	----	----	----	----	----

Interfacing 128 8K Memory Chips to the 8088 µP



VI XỬ LÝ

Khoa: Điện – Điện Tử

Bộ môn: Kỹ Thuật Máy Tính

Giảng viên: Trần Thiên Thanh

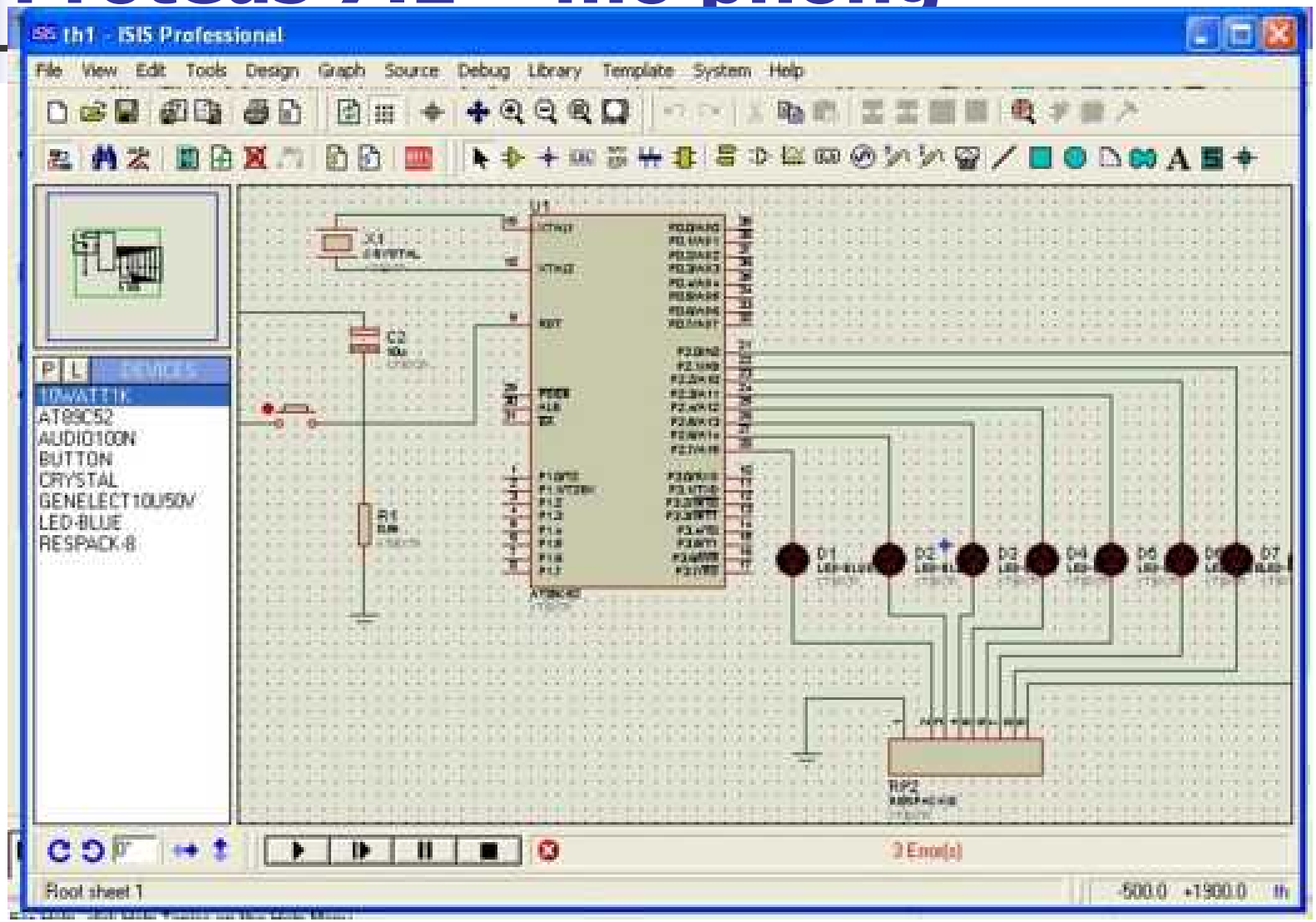
THÔNG TIN CHUNG MÔN HỌC

- Thời gian: 15 tuần – 60 tiết
  - Lý Thuyết: 45 tiết – 11 tuần
  - Bài tập-thực hành: 15 tiết – 03 tuần
- Điểm thi
  - Chuyên cần: 10%
  - Giữa kỳ: hết chương 3 – 10%
  - Bài tập lớn/Thực hành: 10% - hết chương 4
  - Thảo luận/bài tập: 10%
  - Cuối kỳ: 60% - vấn đáp

CHƯƠNG1: GIỚI THIỆU VI XỬ LÝ

- Thảo luận, báo cáo (5%)
  - Phân nhóm
  - Đề tài thảo luận
  - Báo cáo cuối buổi
- Bài tập tại lớp (5%)
- Bài tập lớn (10%)

Proteus 7.1 – mô phỏng



NỘI DUNG

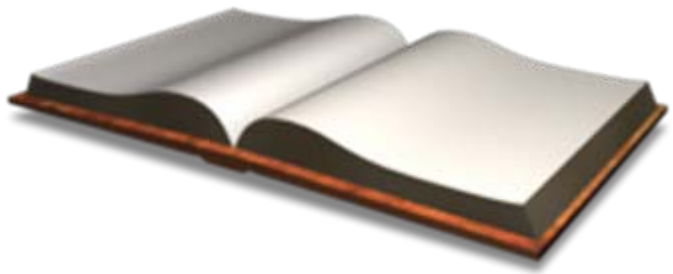
- Yêu cầu:
  - Hiểu về môn học và vai trò
  - Nắm các yêu cầu để học tốt
- Cách học tốt môn này
  - Nắm vững lý thuyết
  - Học thuộc tập lệnh 8051
  - Đọc tham khảo các chương trình ví dụ
  - Viết chương trình nhiều với các ứng dụng thực tiễn, dùng phần mềm mô phỏng

NỘI DUNG

- Giáo trình chính:
  - http://www.box.net/shared/ljtd2lzn65
- Sách tham khảo:
  - "The 8051 – Microcontroller" – I.Scott Mackenzie
  - "Họ vi điều khiển 8051" – Tống Văn On

MỤC LỤC

- CHƯƠNG 1: Giới thiệu vi xử lý
- CHƯƠNG 2: Phần cứng họ MCS-51
- CHƯƠNG 3: Lập trình hợp ngữ họ MCS-51
- CHƯƠNG 4: Các chức năng của họ vi điều khiển MCS-51
- CHƯƠNG 5: Giao tiếp



CHƯƠNG1: GIỚI THIỆU VI XỬ LÝ

- Mục tiêu (Tuần 1)
  - Hiểu và giải thích được cấu trúc chung và hoạt động của một hệ thống VXL. Vai trò các Bus
  - Hiểu chức năng các khối của VXL
  - Phân loại bộ nhớ
  - SV biết các thảo luận, báo cáo

CHƯƠNG1: GIỚI THIỆU VI XỬ LÝ

- I – Tổng quan hệ thống vi xử lý
- II – Các loại bus
- III – Vi xử lý
- IV – Bộ nhớ
- V – Nhập xuất (I/O)
- VI – Vi xử lý – Vi điều khiển
- (Tập lệnh 8051)

CHƯƠNG1: GIỚI THIỆU VI XỬ LÝ

- I – Tổng quan hệ thống vi xử lý
- II – Các loại bus
- III – Vi xử lý
- IV – Bộ nhớ
- V – Nhập xuất (I/O)
- VI – Vi xử lý – Vi điều khiển
- (Tập lệnh 8051)

Ch1: I - Tổng quan hệ thống VXL

- 1. Quá trình phát triển của máy vi tính
- 2. Ứng dụng của vi xử lý
- 3. Sơ đồ khối của hệ vi xử lý

Ch1: I Tổng quan hệ thống VXL

- 1. Quá trình phát triển của máy vi tính
  - 1971 - Intel giới thiệu 8080, là bộ vi xử lý đầu tiên, SDK-85
  - Các hãng khác: Motorola, RCA, MOS Technology, Zilog… giới thiệu 6800, 1801, 6502, Z80, D2, KIM-1, …
  - 1976 – Intel giới thiệu 8748, vi điều khiển thuộc họ MCS-48 → chuẩn công nghiệp
  - 1980 – Intel công bố chip 8051 (Simen: SAB80515

Ch1: I Tổng quan hệ thống VXL

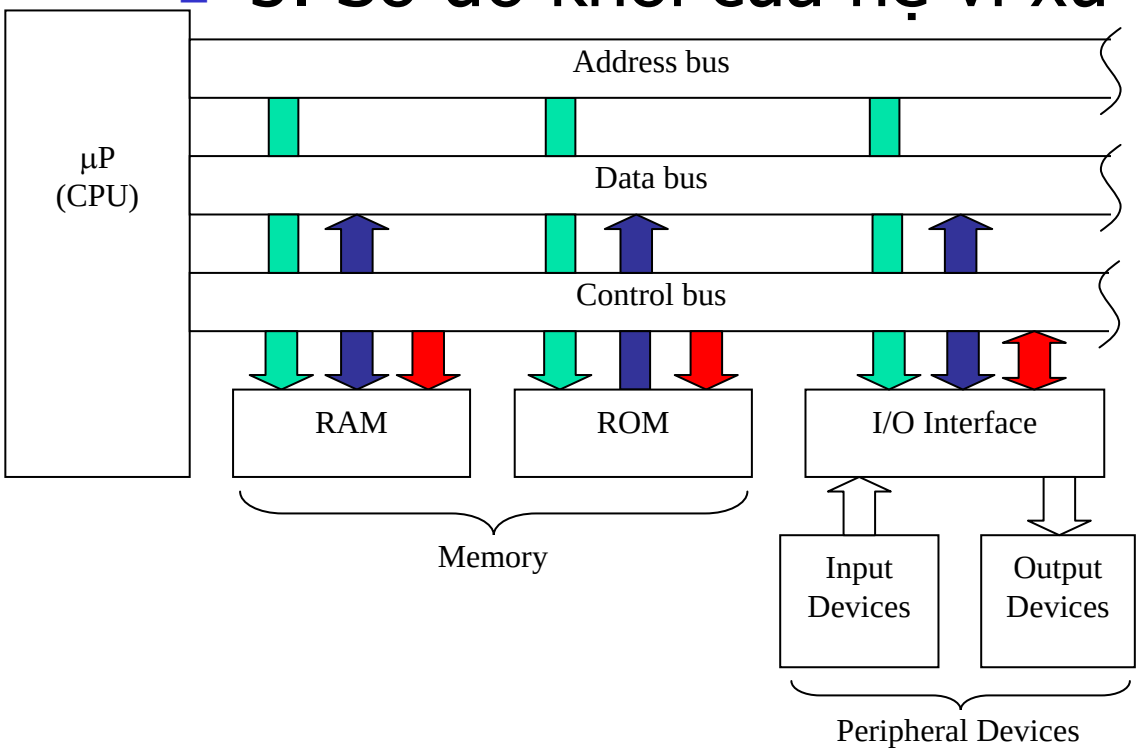
- 2. Ứng dụng của vi xử lý
  - Thay thế các thành phần cơ điện trong các sản phẩm
    - Máy giặt, bộ đèn điều khiển giao thông
  - Xe ô tô, thiết bị công nghiệp, các sản phẩm tiêu dùng
  - Các thiết bị ngoại vi của máy vi tính (thảo luận)

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý

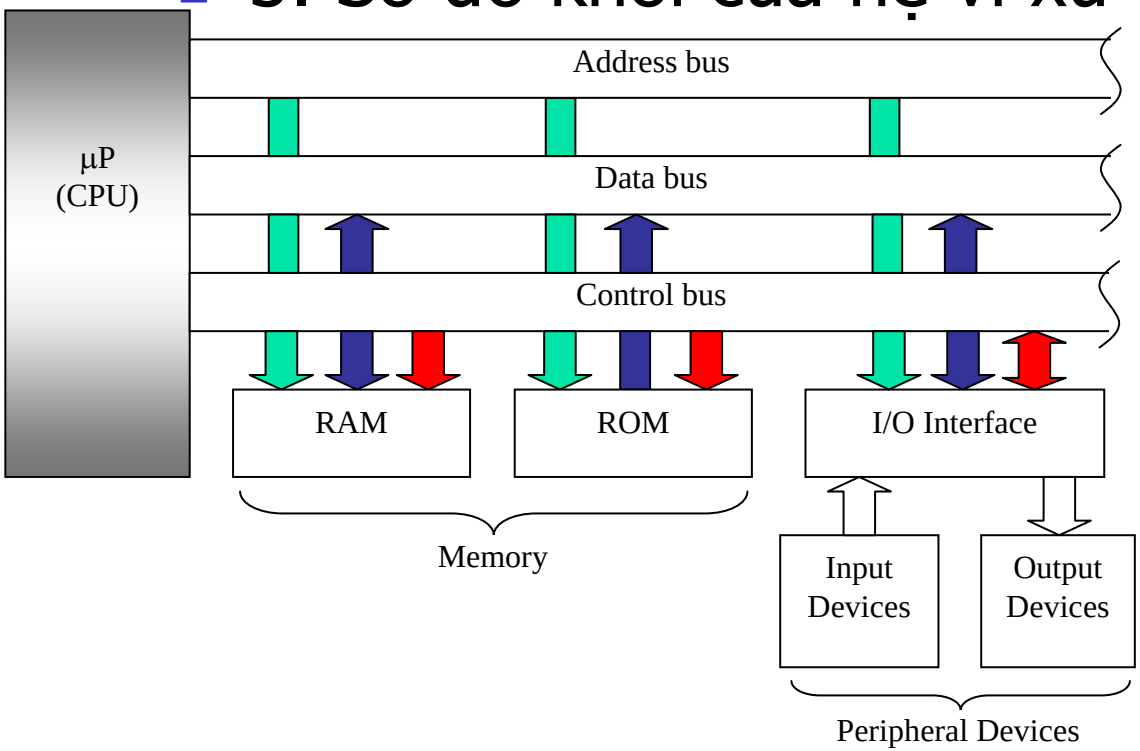


Hình 1.1

μP (Microprocessor): Vi xử lý
CPU (Central Processing Unit): Đơn vị xử lý trung tâm
Address bus: Bus địa chỉ
Data bus: Bus dữ liệu
Control bus: Bus điều khiển
RAM (Random Access Memory): Bộ nhớ truy xuất ngẫu nhiên
ROM (Read-Only Memory): Bộ nhớ chỉ đọc
I/O Interface: Khối giao tiếp nhập/xuất
Peripheral Devices: Thiết bị ngoại vi

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý



Hình 1.1

CPU

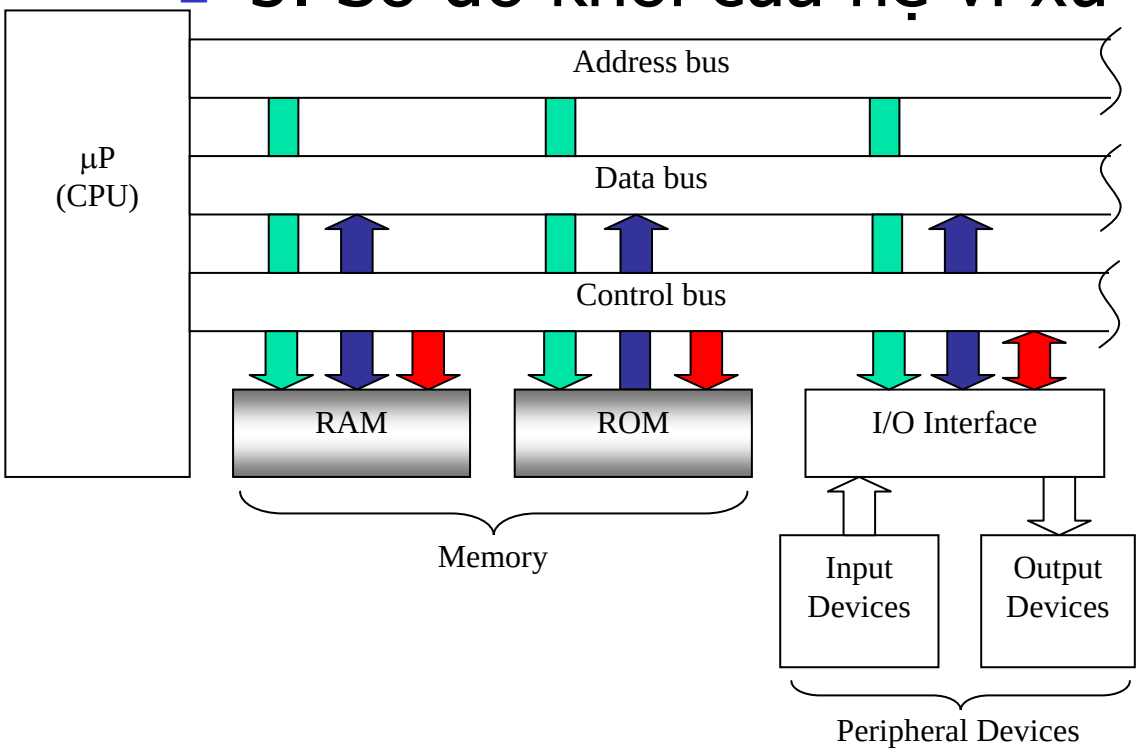
Nguyên tắc làm việc: thực hiện các lệnh liên tục và tuần tự

Mỗi lệnh được biểu diễn bằng mã máy (binary = opcode)

Kết nối với hệ thống bên ngoài thông qua hệ thống bus

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý



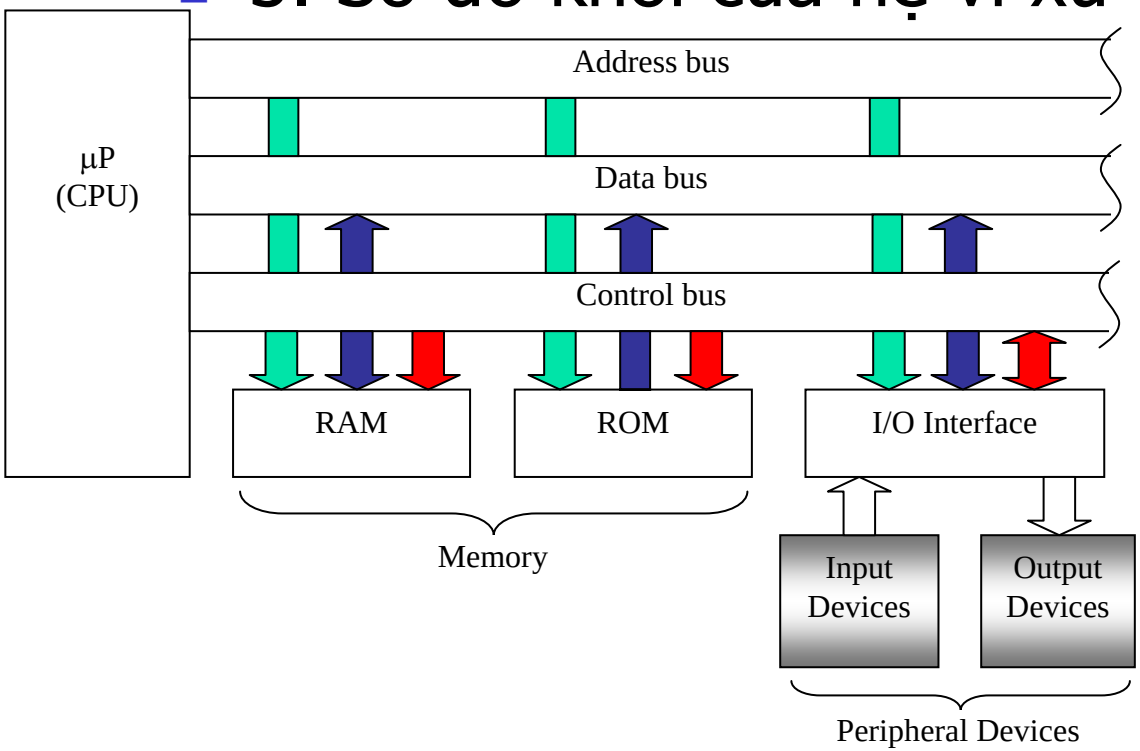
Hình 1.1

Bộ nhớ

Được phân chia theo chức năng: bộ nhớ chương trình: chứa mã lệnh (mã máy) và bộ nhớ dữ liệu: chứa dữ liệu để xử lý khi CPU thực hiện lệnh

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý



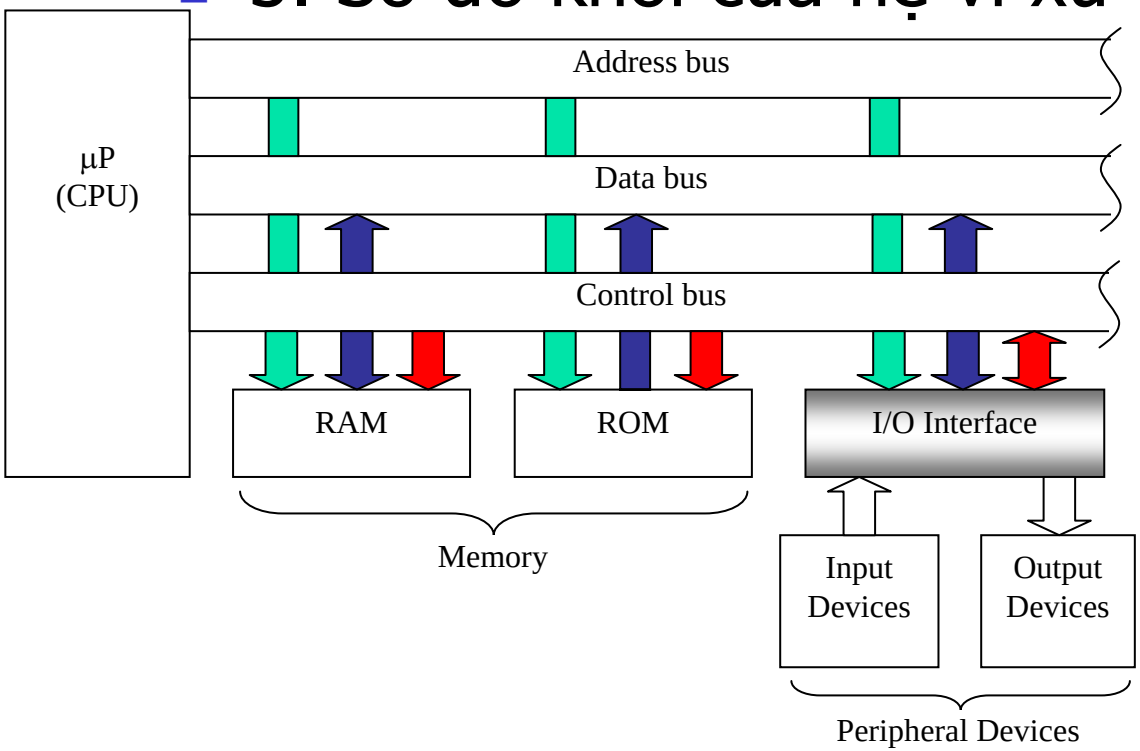
Hình 1.1

Ngoại vi

Thực chất là những cổng vào/ra để CPU giao tiếp với các thiết bị bên ngoài

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý



Hình 1.1

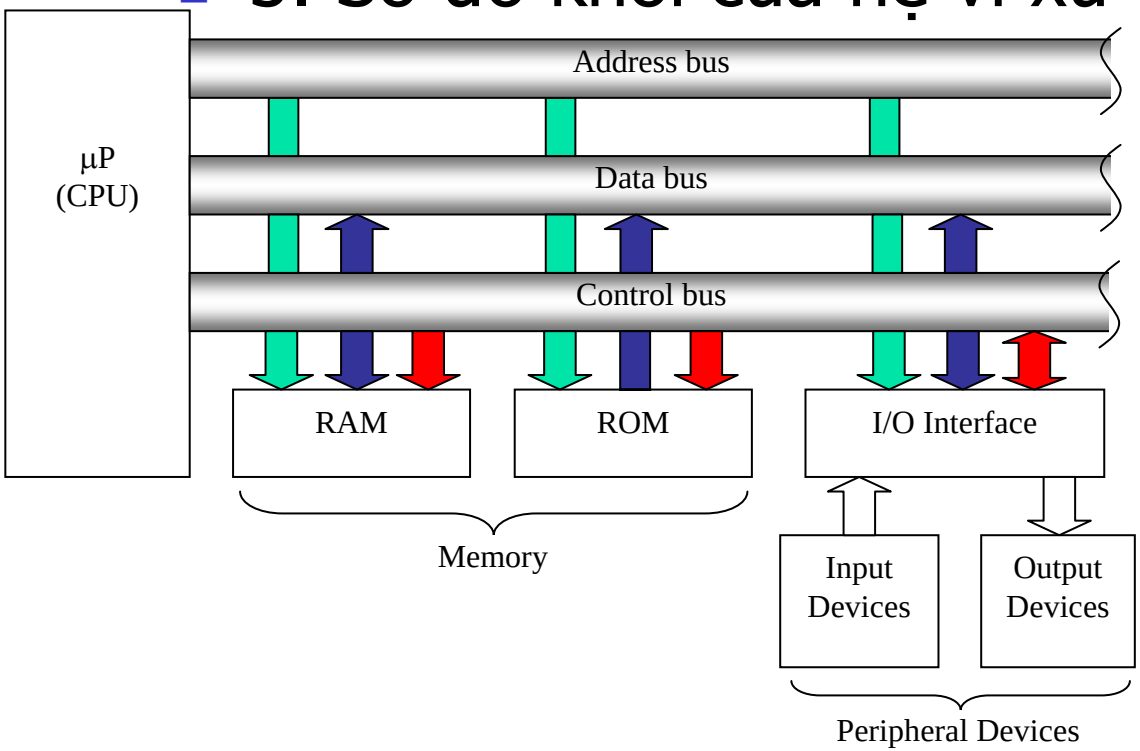
Giải mã địa chỉ

Bộ nhớ, ngoại vi kết nối chung bus, để tiết kiệm dây dẫn. Để trách hiện tượng xung đột logic thì bộ nhớ và I/O hoạt động ở 3 trạng thái (1,0,hi-Z)

Khi bộ nhớ hay I/O được kết nối vào bus data thì phần còn lại ở trạng thái hi-Z

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý



Hình 1.1

Hệ thống bus

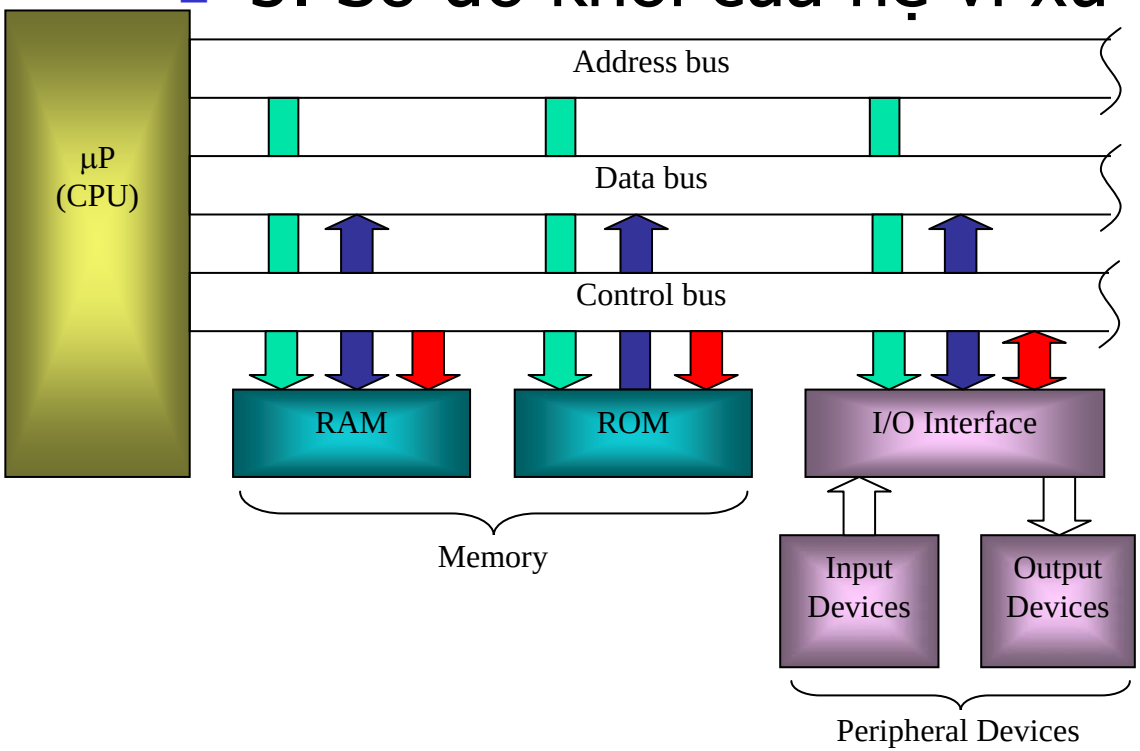
Bus địa chỉ: chứa định vị địa chỉ (được CPU xuất ra)

Bus data: tại 1 thời điểm, CPU chỉ giao tiếp được với 1 đơn vị bộ nhớ hoặc I/O (2chiều)

Bus điều khiển: gồm các tín hiệu đồng bộ hoạt động

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý



Hình 1.1

Ba khối chính

1. Bộ nhớ
2. CPU:
 - Đọc/ghi vào bộ nhớ
 - Đọc từ đầu vào
 - Ghi ra đầu ra
 - Thực hiện lệnh nội bộ : số học và logic
3. Phối ghép (giao tiếp) vào ra I/O

Không có đường trực tiếp từ 1 sang 3.

Ch1: I Tổng quan hệ thống VXL

- 1.1 Hãy nêu các thành phần cơ bản trong một hệ vi xử lý? Chức năng của từng phần

Ch1: II Các loại bus

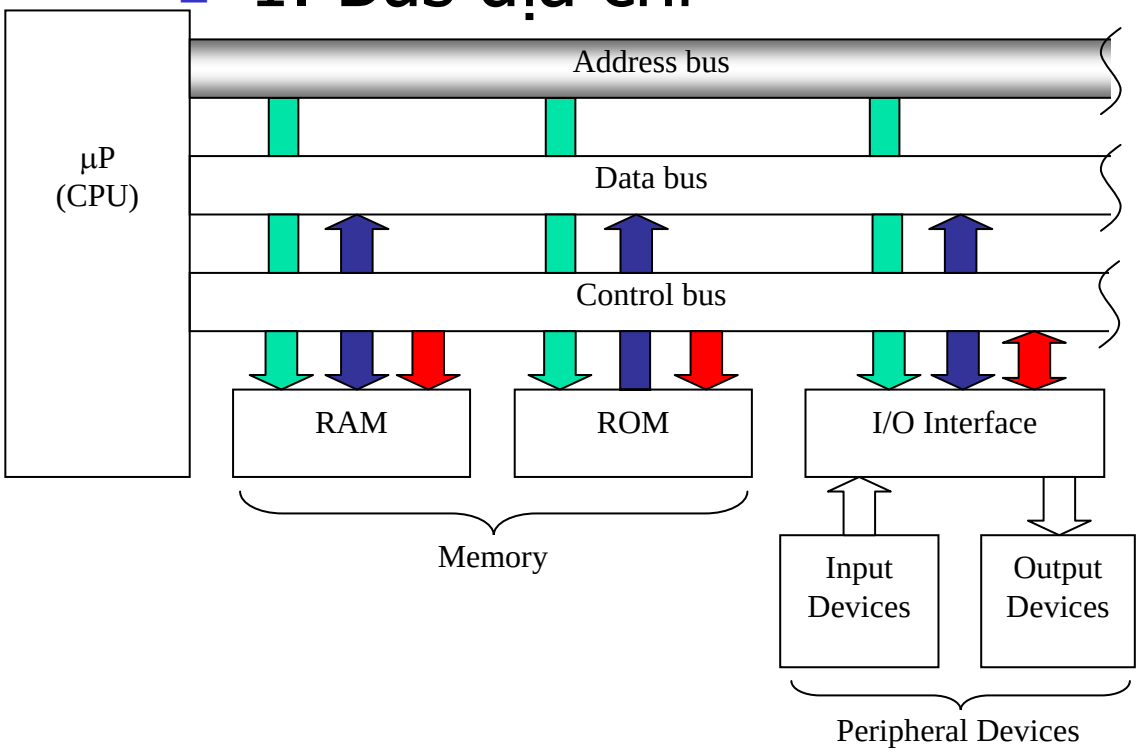
- 1. Bus địa chỉ
  - Đệm bus địa chỉ
- 2. Bus dữ liệu
  - Đệm bus dữ liệu
- Bus điều khiển

Ch1: II Các loại bus

- 1. Bus địa chỉ
  - Đệm bus địa chỉ
- 2. Bus dữ liệu
  - Đệm bus dữ liệu
- Bus điều khiển

Ch1: II Các loại bus

- 1. Bus địa chỉ



Hình 1.1

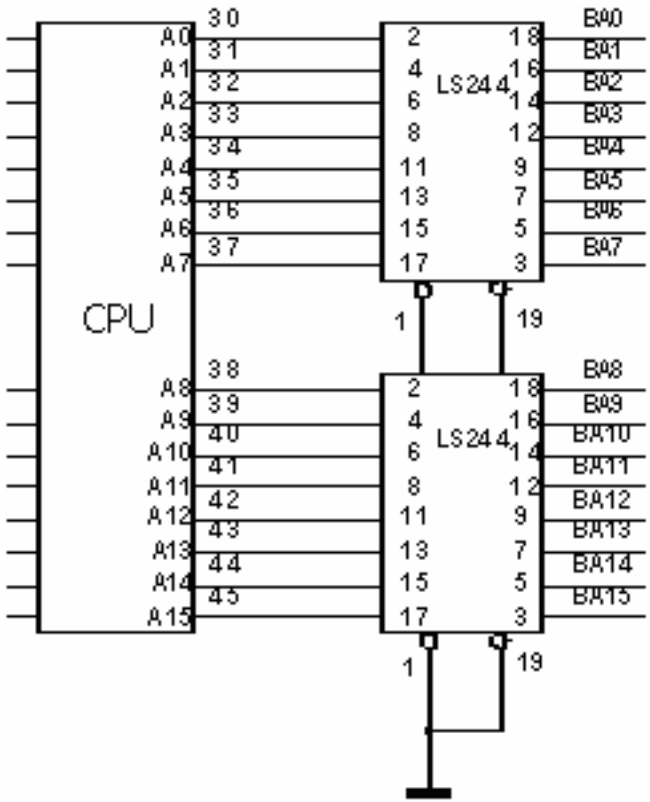
Nội dung: thông tin địa chỉ cần truy xuất (ngăn nhớ hoặc thiết bị)

Số lượng địa chỉ µP quản lý phụ thuộc số đường dây (16,20,24,32)

Bus một chiều đi từ **µP**
N đường dây → 2^N địa chỉ
8051 → N =16

Ch1: II Các loại bus

- 1. (Đệm bus địa chỉ)



Kết nối vật lý dẫn đến quá dòng:

Không hoạt động

Hoạt động không ổn định

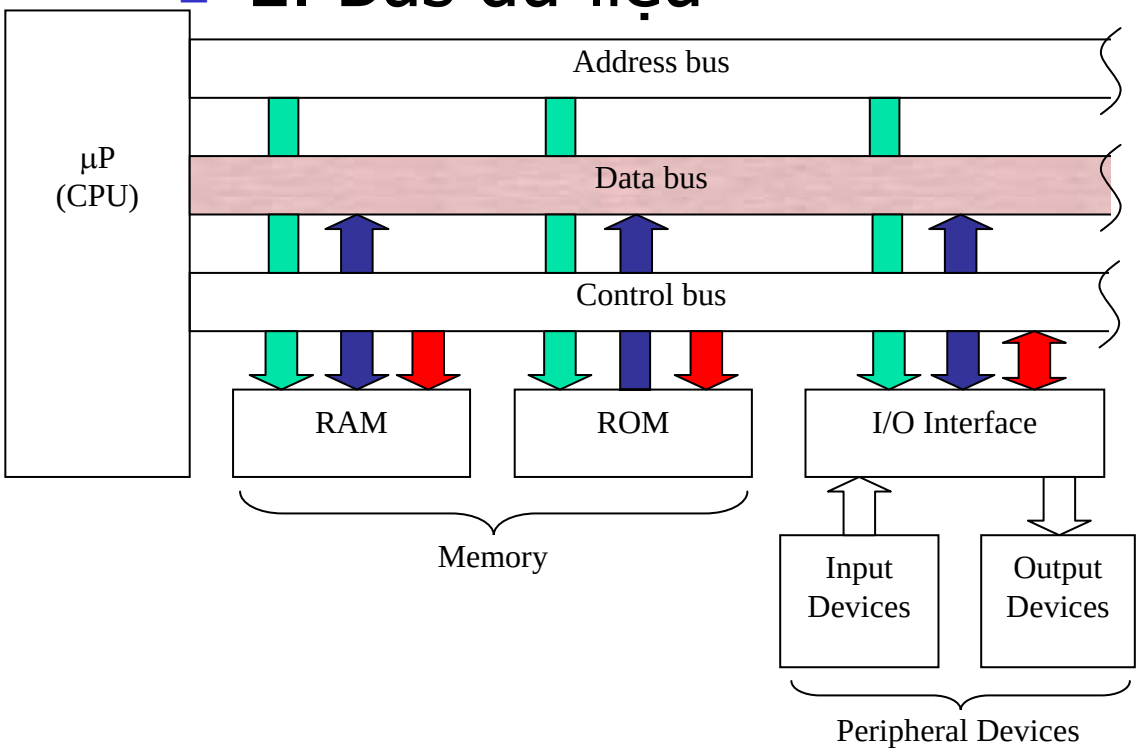
→ Dùng bộ đệm địa chỉ

Ch1: II Các loại bus

- 1. Bus địa chỉ
  - 1.9 Có bao nhiêu vị trí bộ nhớ có thể được định địa chỉ bởi một µP có 20 đường địa chỉ?

Ch1: II Các loại bus

- 2. Bus dữ liệu



Hình 1.1

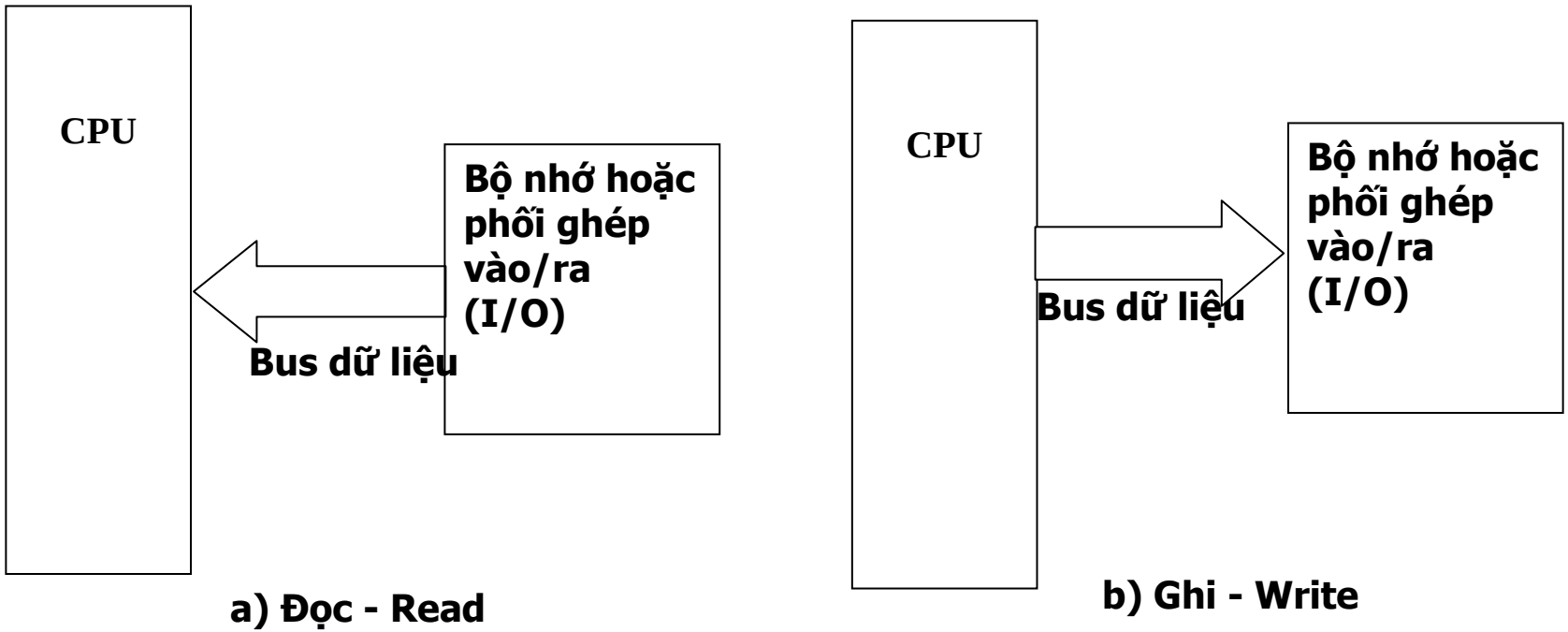
Nội dung: thông tin dữ liệu đến/từ **µP**

Số lượng đường dây quyết định số bit dữ liệu mà µP có khả năng quản lý cùng một lúc (8,16,32, 64 … bit)

Bus hai chiều

Ch1: II Các loại bus

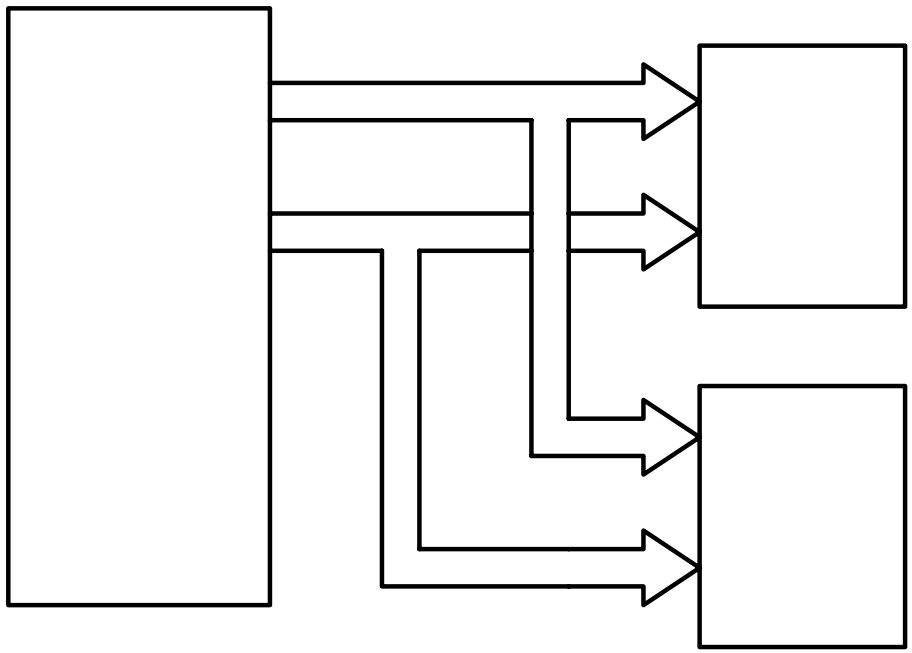
- 2. Bus dữ liệu



Tại 1 thời điểm, dữ liệu chỉ truyền theo 1 hướng

Ch1: II Các loại bus

- 2. Bus dữ liệu

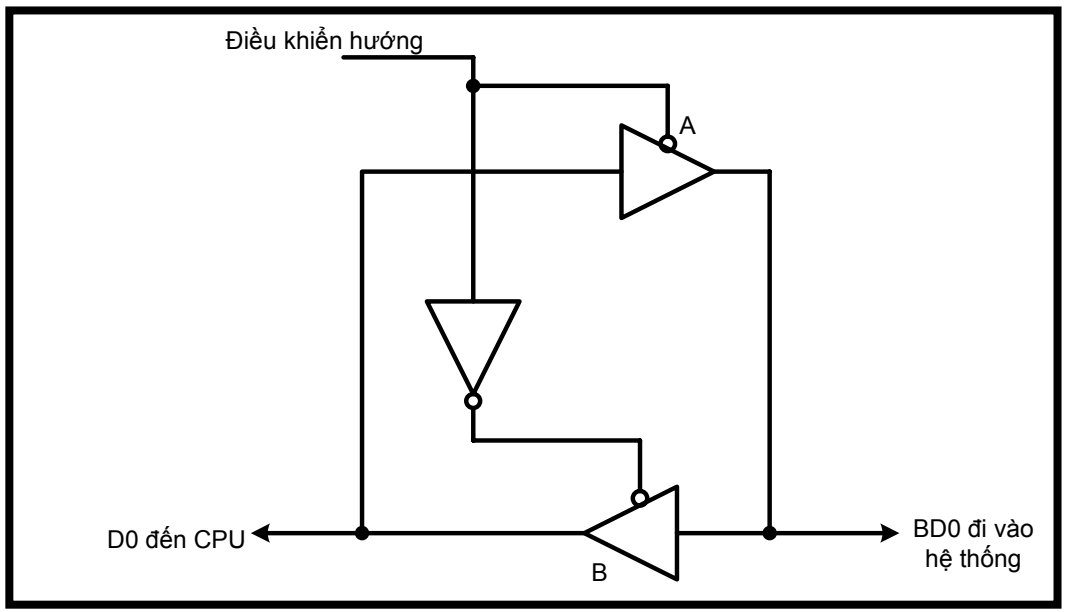


Bus địa chỉ hoạt động độc lập với bus dữ liệu

Address

Ch1: II Các loại bus

- 2. Bus dữ liệu



Kỹ thuật đệm 2 chiều sử dụng thêm một tín hiệu điều khiển, tín hiệu này sẽ quy định chiều dữ liệu sẽ được đệm.

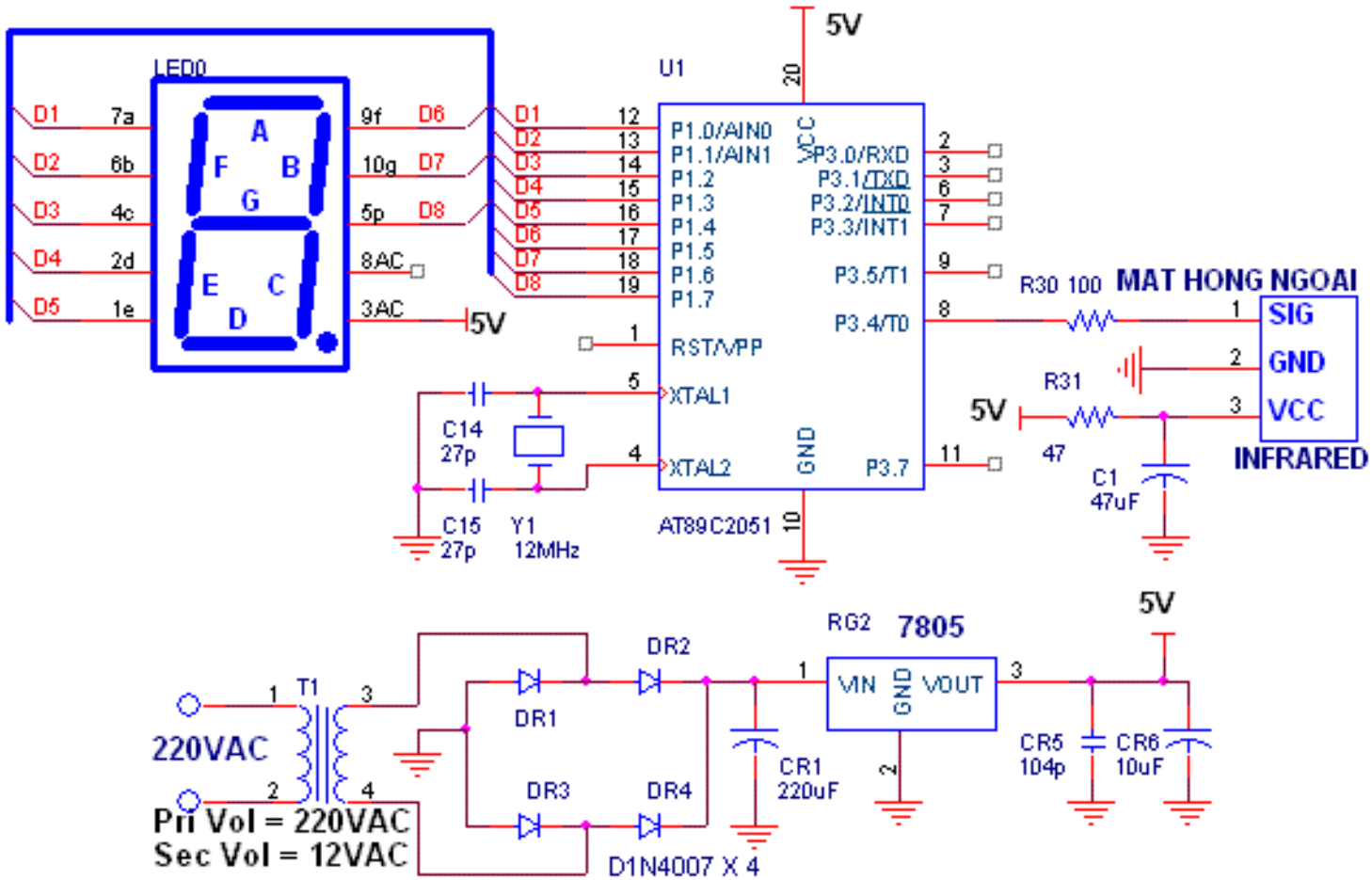
Ch1: II Các loại bus

- 2. Bus dữ liệu
  - 1.10 Nếu một chip bộ nhớ có kích thước là 1024x4bit thì phải cần bao nhiêu chip như vậy để tạo ra 2k(2048) bye bộ nhớ?
  - 1.11 Nếu một chip bộ nhớ có kích thước là 256x1bit thì phải cần bao nhiêu chip như vậy để tạo ra 1KB bộ nhớ?

Ch1: II Các loại bus

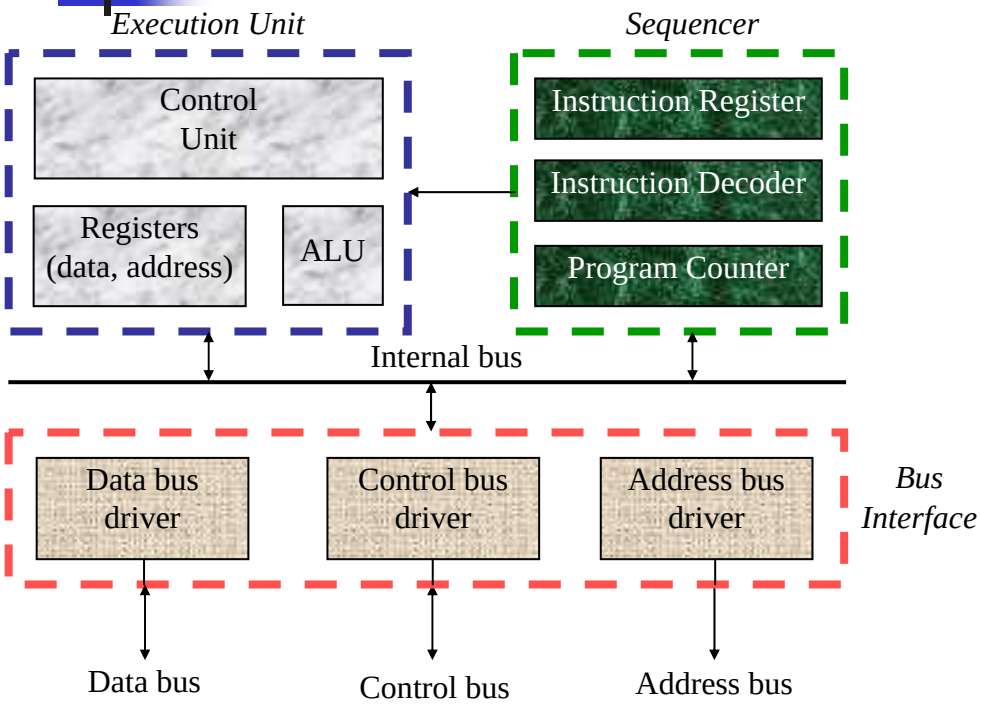
- 3. Bus điều khiển
  - Gồm các đường tín hiệu khác nhau ($\overline{}$, $\overline{RD}$
  - Hướng truyền tuỳ vào loại tín hiệu
  - 06 loại truyền thông tiêu biểu mà bus điều khiển phải xác định bằng tín hiệu điện
    - Đọc/ghi từ/vào bộ nhớ
    - Đọc/ghi từ/vào I/O
    - Nhận biết yêu cầu ngắt (interrupt acknowledge)
    - Nhận biết yêu cầu treo (phục vụ DMA, hold acknowledge)

Ch1: II Các loại bus



Ch1: II Các loại bus

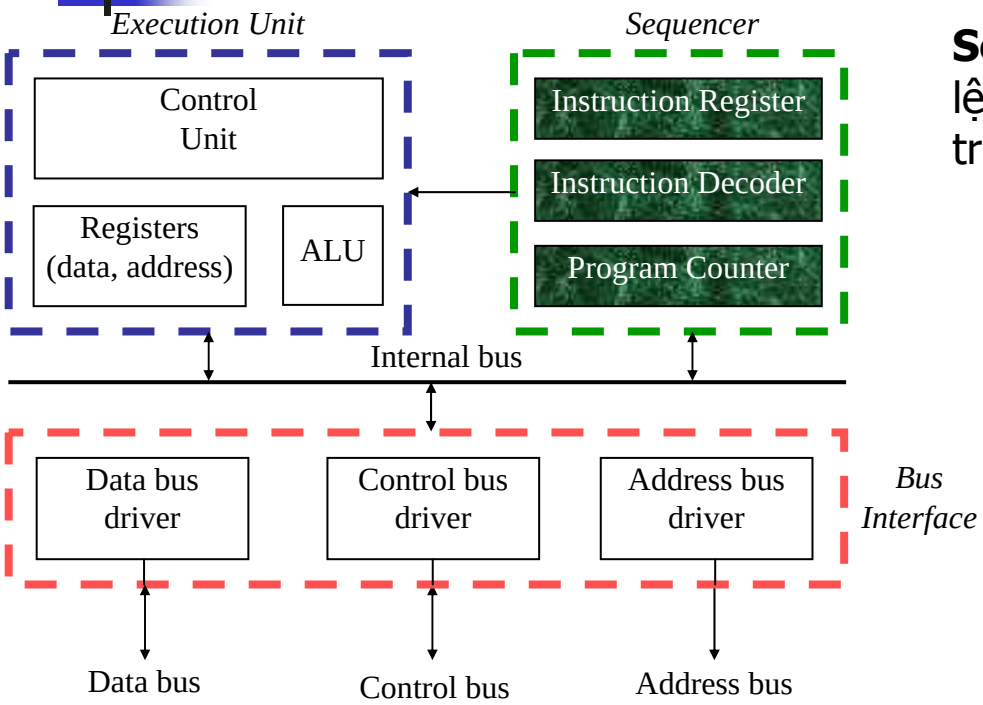
Ch1 III Chip Vi xử lý µP



Hình 1.6

Execution Unit: Khối thực thi
Control Unit: Khối điều khiển
Registers: Các thanh ghi
ALU (Arithmetic & Logic Unit): Khối logic - số học
Sequencer: Bộ điều khiển tuần tự
Instruction Register: Thanh ghi lệnh
Instruction Decoder: Bộ giải mã lệnh
Program Counter: Bộ đếm chương trình
Internal bus: Bus nội
Bus interface: Giao tiếp bus
Data bus driver: Bộ điều khiển bus dữ liệu
Control bus driver: Bộ điều khiển bus điều khiển
Address bus driver: Bộ điều khiển bus địa chỉ

Ch1 III Chip Vi xử lý µP

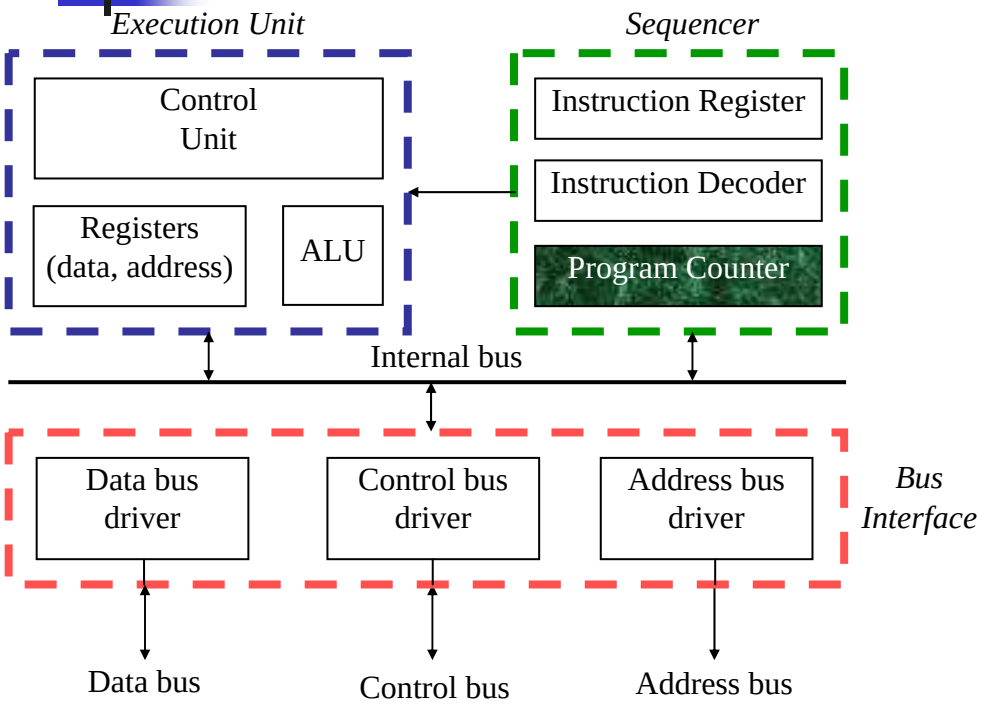


Hình 1.6

Sequencer: Bộ điều khiển tuần tự: nhận lệnh từ bộ nhớ, sau đó giải mã lệnh và truyền lệnh đã giải mã đến khối thực thi

Instruction Register: Thanh ghi lệnh
Instruction Decoder: Bộ giải mã lệnh
Program Counter: Bộ đếm chương trình

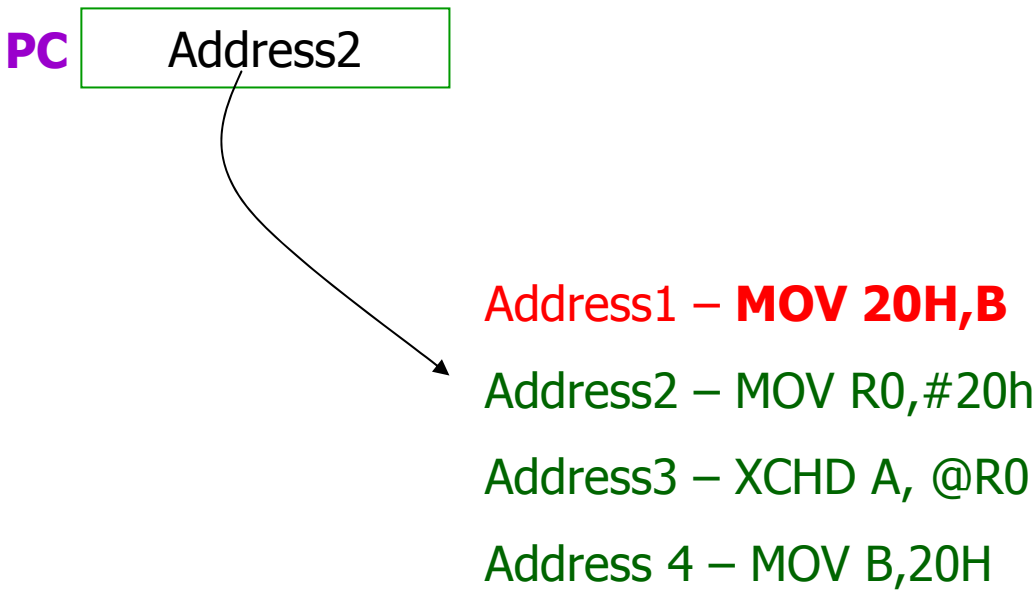
Ch1 III Chip Vi xử lý µP



Hình 1.6

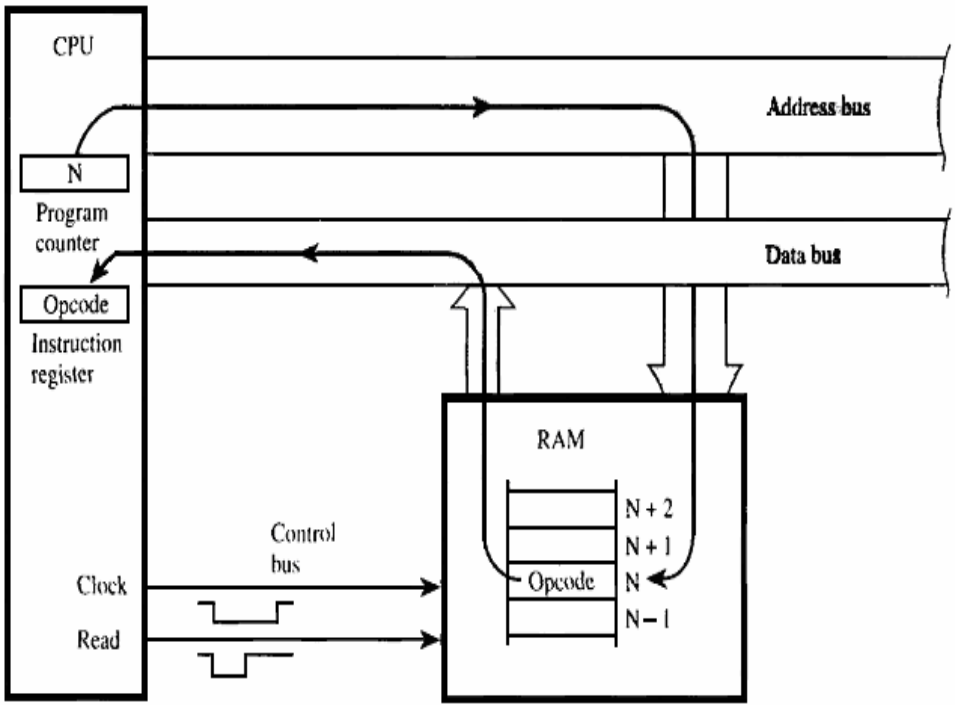
- Thanh ghi PC (bộ đếm chương trình):
  - Nội dung là địa chỉ ô nhớ chứa mã lệnh cần truy xuất (lệnh kế tiếp lệnh đang thực thi)

Ch1 III Chip Vi xử lý µP



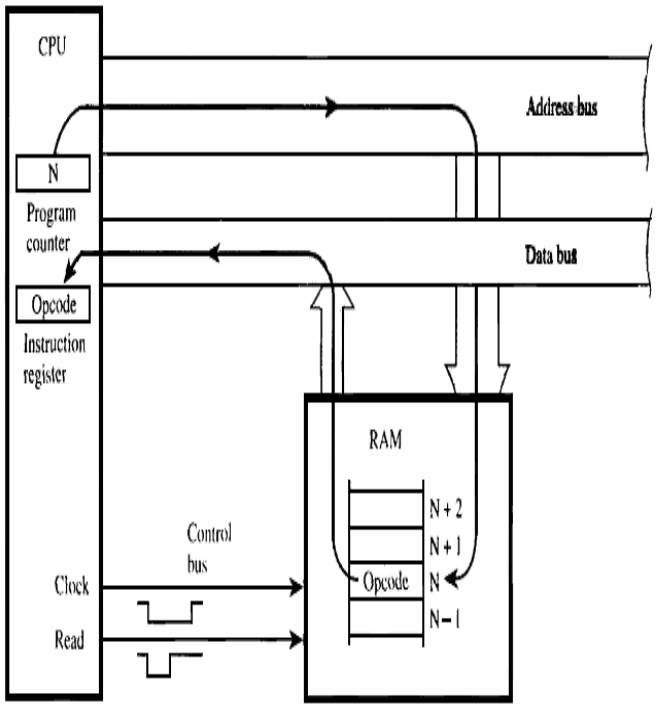
- Thanh ghi PC (bộ đếm chương trình):
  - Nội dung là địa chỉ ô nhớ chứa mã lệnh cần truy xuất (lệnh kế tiếp lệnh đang thực thi)
  - Gặp lệnh chuyển điều khiển (nhảy, gọi chương trình con…) thì nội dung PC bị thay đổi
  - Còn có tên là con trỏ lệnh IP (Instruction Pointer)

Ch1 III Chip Vi xử lý µP



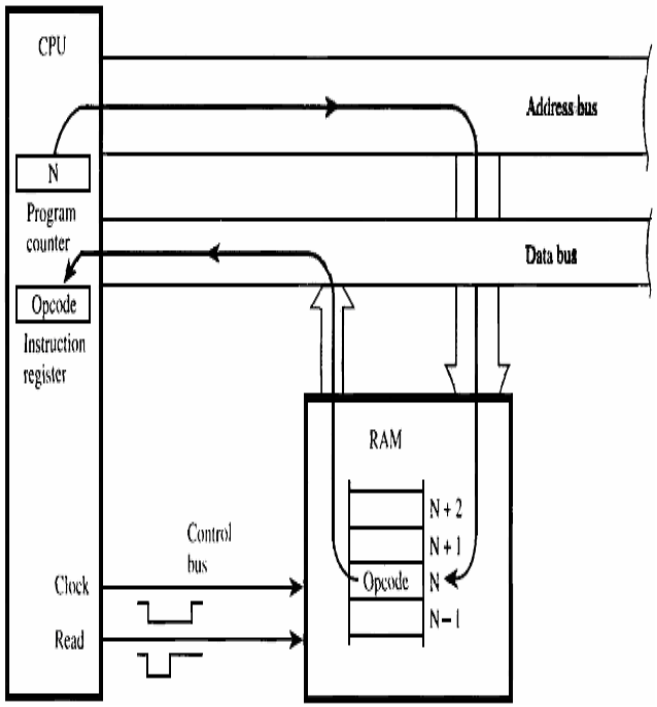
- *Việc tìm nạp lệnh từ bộ nhớ là một trong các thao tác cơ bản nhất mà µP thực hiện, gồm các bước như sau:*
  - *- Nội dung của PC được đặt lên bus địa chỉ.*
  - *- Tín hiệu điều khiển READ được xác lập (chuyển sang trạng thái tích cực).*
  - *- Mã lệnh được đọc từ bộ nhớ và đưa lên bus dữ liệu.*
  - *- Mã lệnh được chốt vào thanh ghi lệnh IR bên trong.*
  - *- PC được tăng lên để chuẩn bị tìm nạp lệnh kế từ bộ nhớ.*

Ch1 III Chip Vi xử lý µP



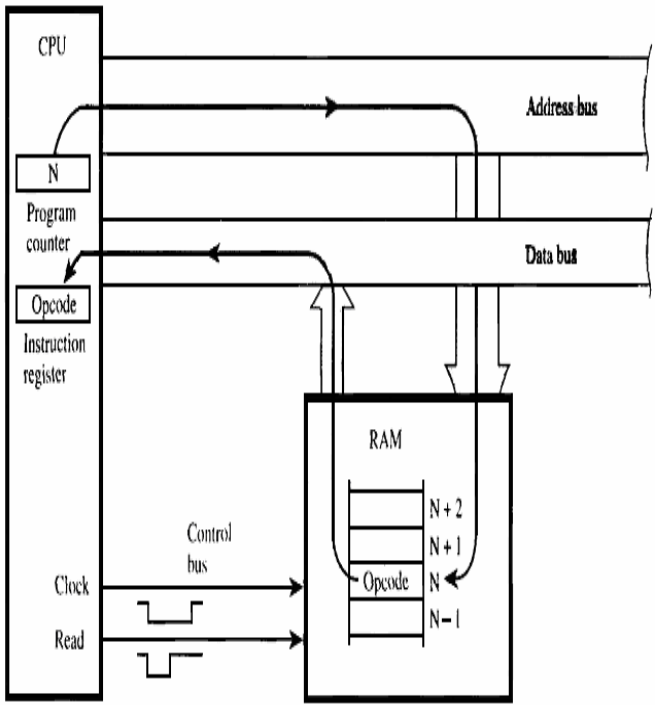
- *Một chu kỳ lệnh có thể chia thành 2 bước:*
  - *Chu kỳ nhận lệnh: CPU sẽ xuất nội dung thanh ghi PC ra bus địa chỉ, đồng thời xuất tín hiệu đọc lệnh trên bus dữ liệu➔ giải mã địa chỉ nhận lệnh(địa chỉ, tín hiệu điều khiển) và cho phép xuất ô nhớ có địa chỉ tương ứng, đặt dữ liệu (là mã lệnh) lên bus data. CPU đọc data này và cất trong IR. Đồng thời, nội dung PC tăng, trỏ vào địa chỉ mã lệnh kế tiếp. Thuật ngữ PC hiện hành là PC đã trỏ vào mã lệnh kế tiếp*

Ch1 III Chip Vi xử lý µP



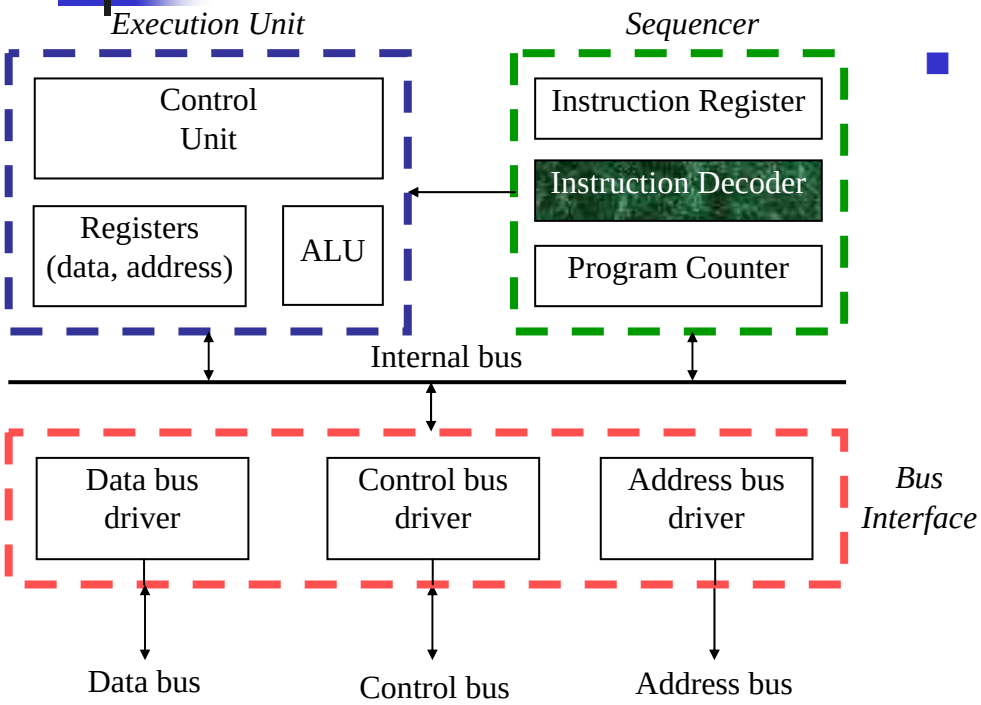
- *Một chu kỳ lệnh có thể chia thành 2 bước:*
  - *Chu kỳ thực thi lệnh: giải mã lệnh nhận lệnh từ IR, giải mã lệnh và phát tín hiệu điều khiển đến các khối liên quan để thực hiện lệnh. Tuỳ lệnh mà việc thực thi chỉ thực hiện bên trong CPU hay cần giao tiếp ra bên ngoài.*

Ch1 III Chip Vi xử lý µP



- *Một chu kỳ lệnh có thể chia thành 2 bước:*
  - *Hiện nay, người ta dùng kỹ thuật đường ống có nghĩa là 2 chu kỳ trên hoạt động cùng 1 thời điểm để tiết kiệm chu kỳ bus.*

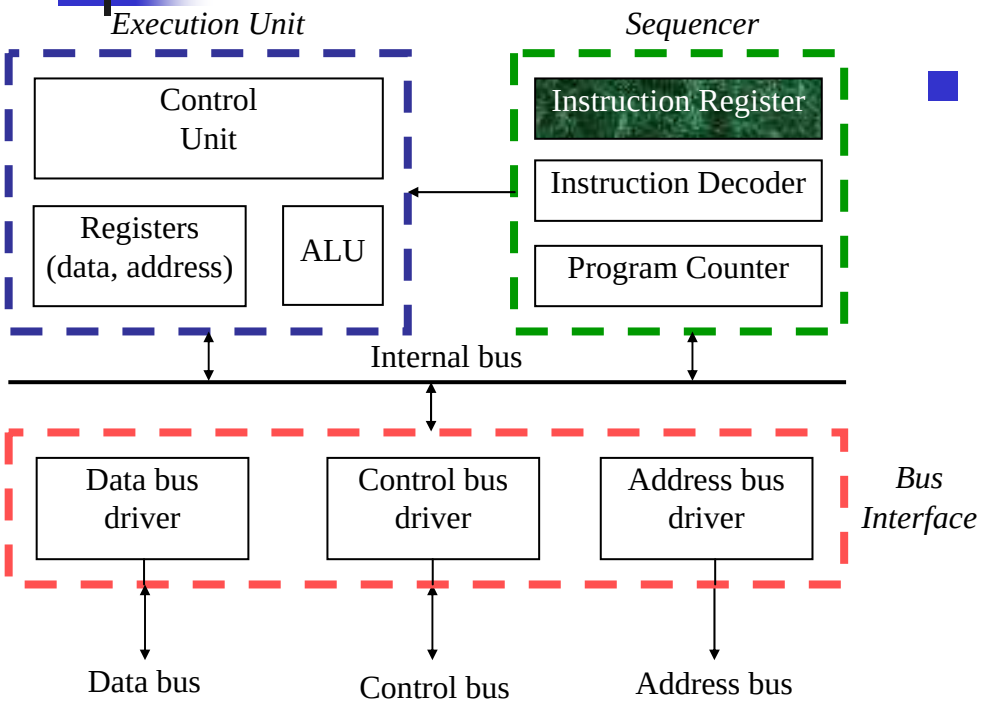
Ch1 III Chip Vi xử lý µP



Hình 1.6

- Bộ giải mã lệnh:
  - Nhận lệnh từ IR
  - Thông dịch (diễn dịch) các lệnh được nhận vào µP
  - Tác động đến những phần khác (ALU, các thanh ghi đa dụng…) để lệnh đó được thực hiện
  - Từ điển lưu nghĩa của mỗi lệnh
  - ID càng lớn thì PC càng hiểu nhiều lệnh

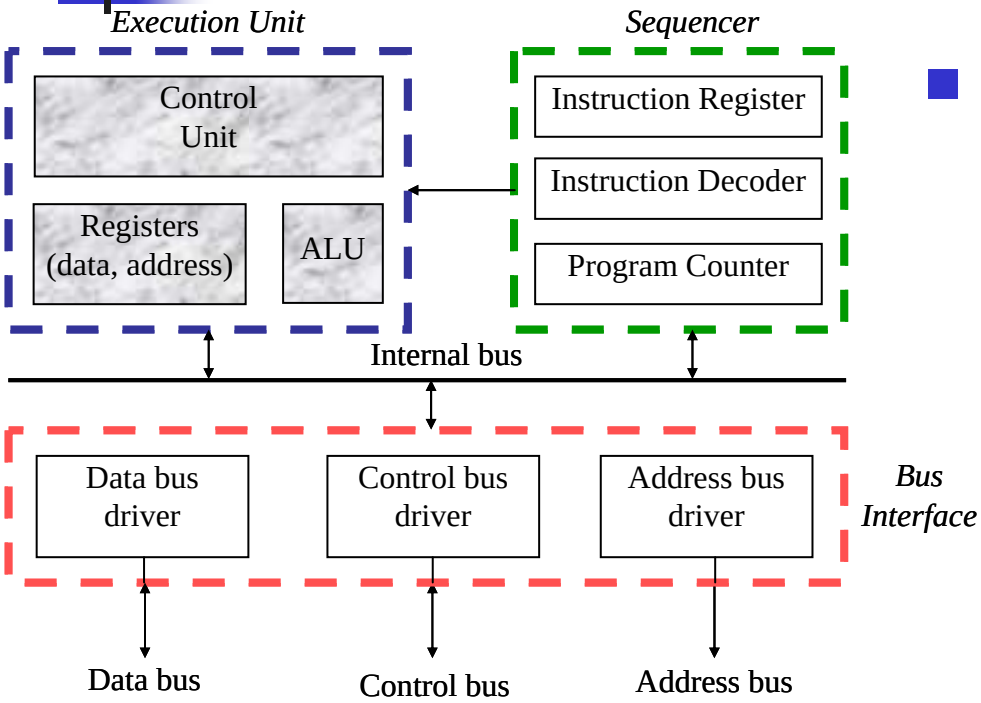
Ch1 III Chip Vi xử lý µP



Hình 1.6

- Thanh ghi lệnh:
  - Lưu trữ mã nhị phân của lệnh đang được thực thi

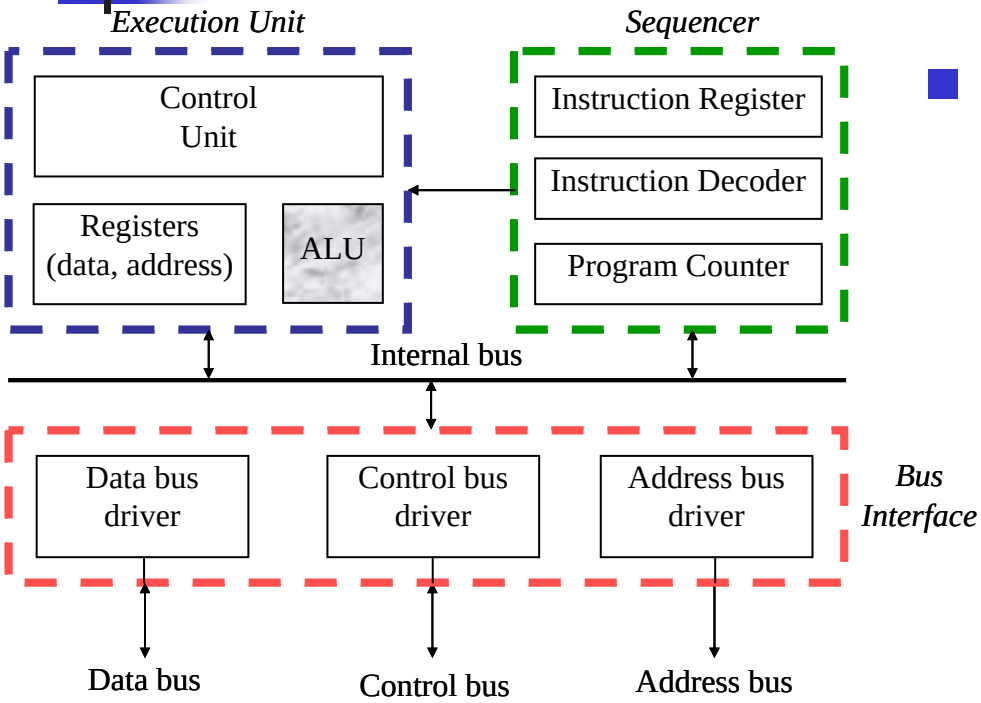
Ch1 III Chip Vi xử lý µP



Hình 1.6

- Khối thực thi:
  - Thực thi và ghi kết quả câu lệnh
  - Các toán hạng nằm ở thanh ghi hoặc ở bus nội

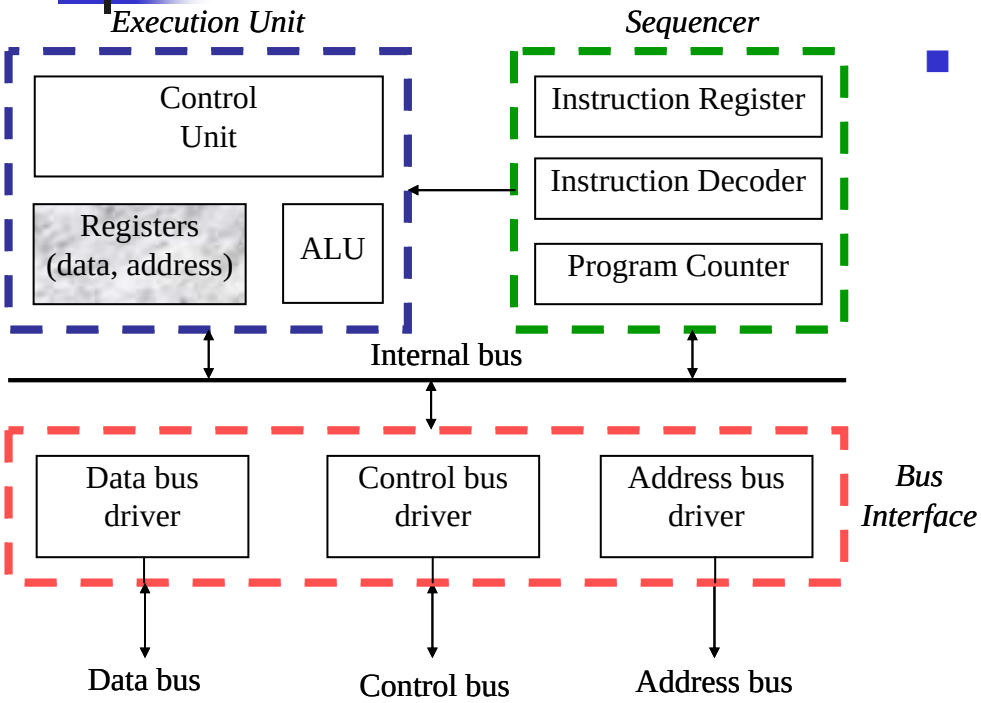
Ch1 III Chip Vi xử lý µP



Hình 1.6

- ALU:
  - Vi mạch điện tử
  - Thực hiện các phép toán số học (+,-,*,/) và logic(and, or, not, xor)

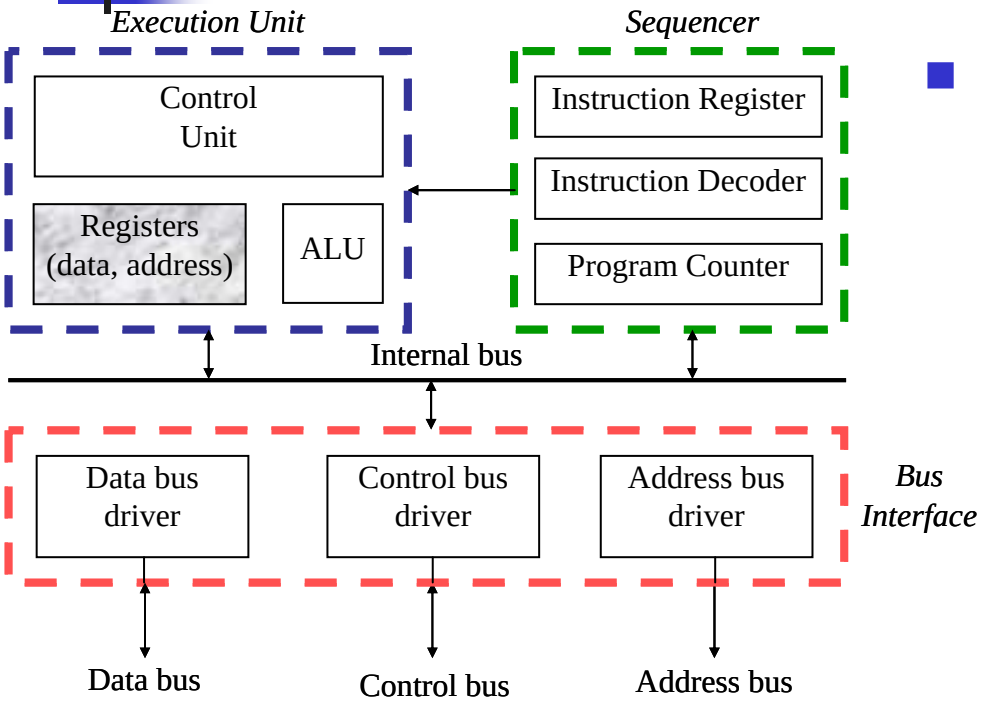
Ch1 III Chip Vi xử lý µP



Hình 1.6

- Thanh ghi đa dụng:
  - Chức năng chính: lưu trữ tạm thời dữ liệu
  - Nội dung: dữ liệu cần xử lý hoặc địa chỉ chứa giá trị cần xử lý nhận từ bộ nhớ hoặc I/O
  - Thanh ghi và độ rộng thanh ghi càng lớn → càng tốt. (không thực hiện nhiều phép truyền thông tin giữa µP và bộ nhớ do truy xuất trực tiếp từ thanh ghi

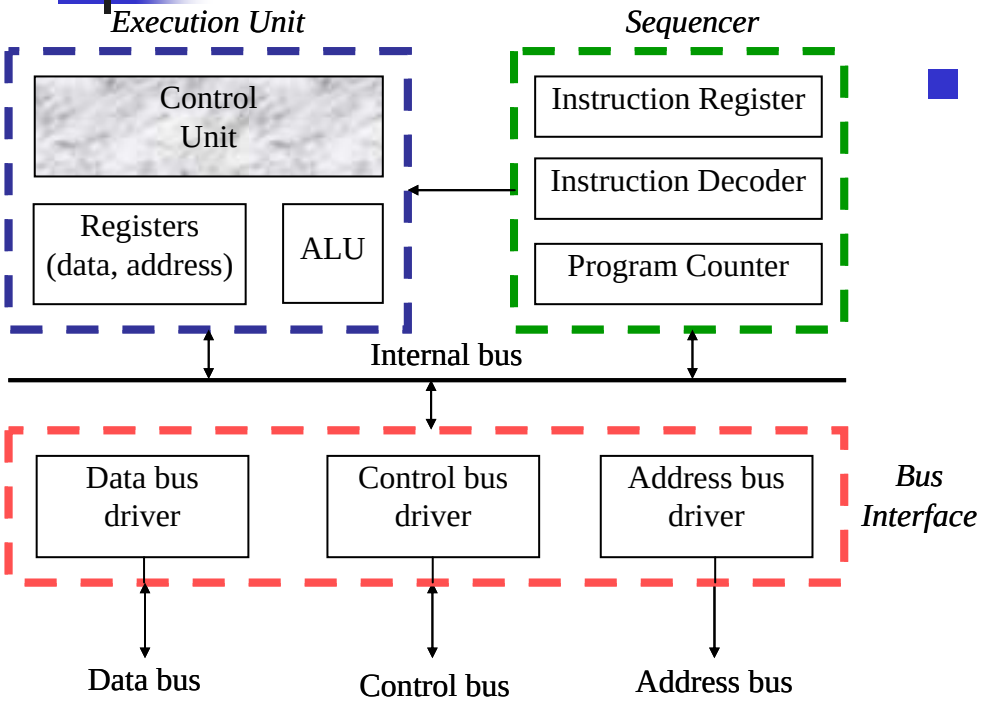
Ch1 III Chip Vi xử lý µP



Hình 1.6

- Thanh ghi đa dụng:
  - Mỗi thanh ghi có địa chỉ truy xuất đến (byte/bit tuỳ vị trí)
  - Nối với nhau hoặc đến các phần tử khác thuộc µP bằng bus nội
  - Độ rộng: 8 bit, 16bit, 32bit, 64bit.

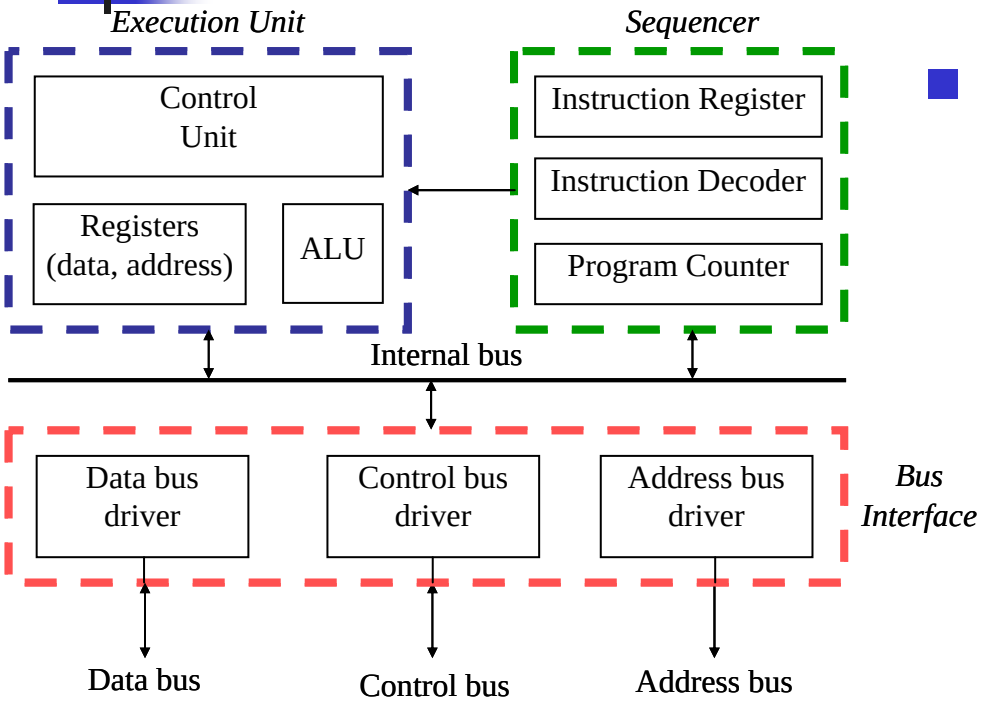
Ch1 III Chip Vi xử lý µP



Hình 1.6

- Khối điều khiển:
  - Tạo tín hiệu điều khiển cho các hoạt động bên trong + bên ngoài của µP .

Ch1 III Chip Vi xử lý µP



Hình 1.6

- Giao tiếp bus:
  - Gồm 3 bộ điều khiển để giao tiếp với bus bên ngoài tương ứng

Ch1 IV Bộ nhớ (memory)

- Nhắc lại các đơn vị bit, nibble, byte, word
  - Bit: 0,1
  - Nibble: 4 bit
  - Byte: 8 bit
  - Word
    - 2 byte
    - Dài:4 byte (theo thế hệ vi xử lý 16bit, 32bit)

Ch1 IV Bộ nhớ (memory)

- 1. Phân loại
- 2. Cấu trúc bên trong tiêu biểu của bộ nhớ
- 3. Truy xuất bộ nhớ
- 4. Giải mã địa chỉ cho bộ nhớ

Ch1 IV Bộ nhớ (memory)

- 1. Phân loại
  - Bộ nhớ thường được chia làm hai loại: bộ nhớ cơ bản (hay bộ nhớ chính – main memory) và bộ nhớ lưu trữ (storage memory).
  - - Bộ nhớ chính: ROM và RAM.
  - - Bộ nhớ lưu trữ: băng từ, đĩa mềm, đĩa cứng…
    - Thông thường bộ nhớ lưu trữ được xem như là thiết bị I/O

Ch1 IV Bộ nhớ (memory)

- ***a. Bộ nhớ chỉ đọc – ROM (Read-Only Memory)***
  - Là bộ nhớ chỉ đọc, không thể sửa đổi thông tin trong các hoạt động thông thường.
  - Thông tin ghi trong ROM sẽ không bị mất đi khi mất nguồn cung cấp.
  - ROM được ghi bằng thiết bị chuyên dụng.
  - ROM thường được dùng để chứa các chương trình và dữ liệu cố định (*chương trình khởi động, dữ liệu tra bảng …*)

Ch1 IV Bộ nhớ (memory)

- ***b. Bộ nhớ truy xuất ngẫu nhiên – RAM (Random Access Memory)***
  - Cho phép đọc/ghi thông tin bất kỳ lúc nào trong quá trình làm việc mà không cần thiết bị đặc biệt.
  - Thông tin trong RAM sẽ bị mất khi mất nguồn cung cấp.

Ch1 IV Bộ nhớ (memory)

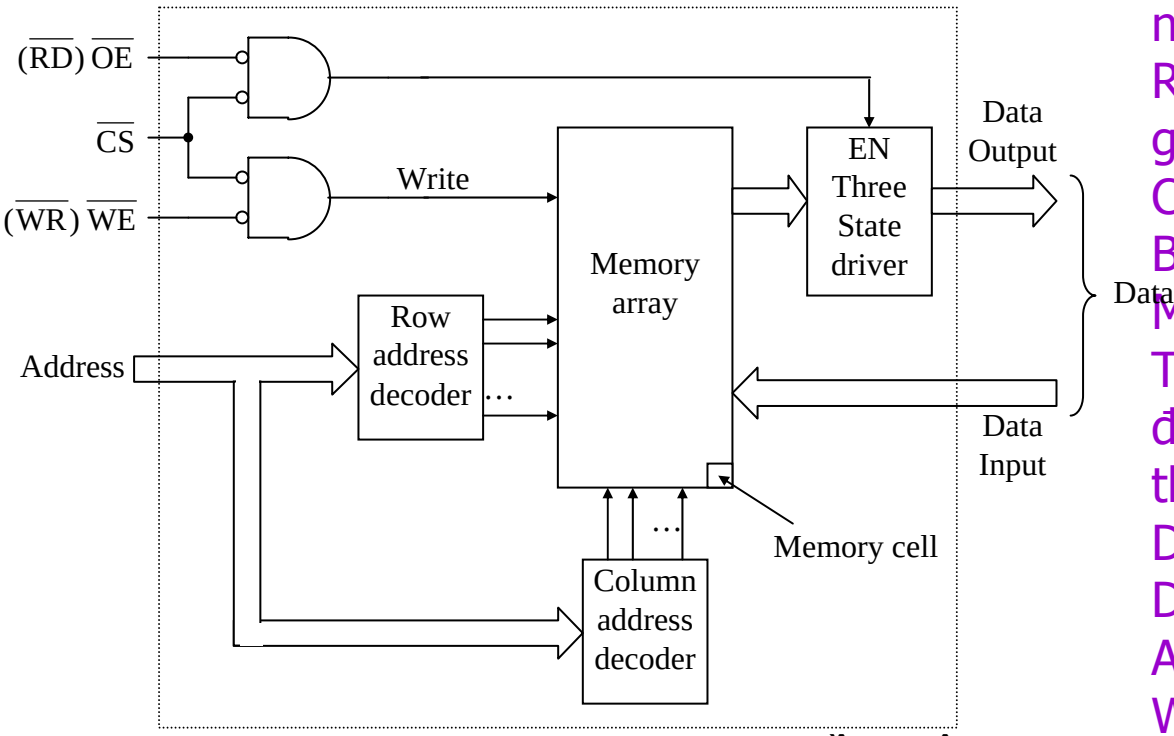
- Có hai loại RAM chính:
  - *RAM động – DRAM* (Dynamic RAM): có cấu tạo từ các transistor MOSFET và tụ điện (*1 phần tử nhớ*), lưu trữ thông tin bằng điện tích trong tụ nên thông tin có thể mất đi (rò rỉ hết) nếu không có biện pháp duy trì thích hợp. Do đó cần có quá trình *làm tươi* (refresh) định kì để phục hồi nội dung của các ô nhớ trước khi nó mất đi (rò rỉ hết). DRAM có thể tích hợp với dung lượng lớn.
  - *RAM tĩnh – SRAM* (Static RAM): cấu tạo từ những Flipflop (FF) (*1 phần tử nhớ*), mỗi FF lưu trữ một bit thông tin nên SRAM không cần quá trình làm tươi để duy trì nội dung. Tuy nhiên, nó khó tích hợp với dung lượng lớn

Ch1 IV Bộ nhớ (memory)

- **2. Cấu trúc bên trong tiêu biểu của bộ nhớ**
  - Bộ nhớ gồm các *phần tử nhớ* hay *ô nhớ* (memory cell) được tổ chức dưới dạng ma trận. Mỗi ô nhớ chứa một bit thông tin.
  - Mảng nhớ được phân chia thành một chuỗi các *ngăn nhớ* hay *từ nhớ* (word).
  - Mỗi ngăn nhớ đều có một địa chỉ duy nhất.
  - Một ngăn nhớ có thể có 4-bit, 8-bit, 16-bit …
  - Ký hiệu: ***số ngăn nhớ x độ rộng mỗi ngăn nhớ***
- Ví dụ: bộ nhớ 1024 x 8 bao gồm 210 ngăn nhớ, mỗi ngăn nhớ có 8-bit.

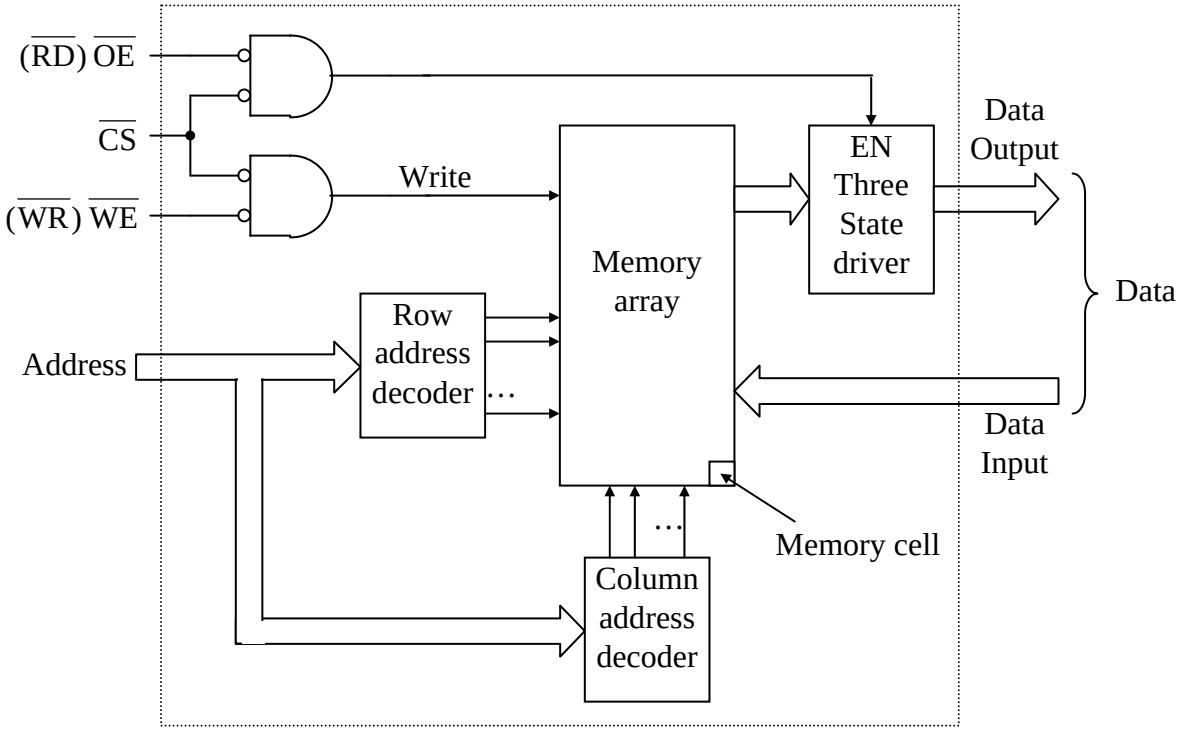
Ch1 IV Bộ nhớ (memory)

- Cấu trúc bên trong tiêu biểu của bộ nhớ:



Memory array: Mảng ô nhớ
Row address decoder: Bộ giải mã địa chỉ hàng
Column address decoder: Bộ giải mã địa chỉ cột
Memory cell: Ô nhớ
Three state driver: Bộ điều khiển ngõ ra 3 trạng thái
Data Output: Dữ liệu ra
Data Input: Dữ liệu vào
Address: Địa chỉ
Write: Ghi

Ch1 IV Bộ nhớ (memory)



Hình 1.8

Các tín hiệu tiêu biểu trên một chip nhớ:

+ (Chip Select): tín hiệu chọn chip (cho phép chip).
+ (Output Enable): tín hiệu cho phép xuất dữ liệu (*nhận xung kích từ µP*).
+ (Write Enable): tín hiệu cho phép ghi dữ liệu (*nhận xung kích từ µP*).
+ Address: các tín hiệu địa chỉ (*từ bus địa chỉ*) để chọn ngăn nhớ cần thao tác.

Ch1: II Các loại bus

- Các đặc trưng của tập lệnh
  - Tuỳ ứng dụng → có tập lệnh khác nhau