HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

*****

PHẠM HOÀNG DUY

BÀI GIẢNG

KỸ THUẬT VI XỬ LÝ

HÀ NỘI 2011

LỜI NÓI ĐẦU

Các bộ vi xử lý được sử dụng phổ biến trong các hệ thống số như hệ thống thông tin liên lạc, hệ thống điều khiển... Tài liệu này tập trung giới thiệu bộ vi xử lý Intel 8086 và các ghép nối tiêu biểu để tạo nên hệ vi xử lý. Hệ vi xử lý dựa trên Intel 8086 tương đối đơn giản và bổ ích cho việc tìm hiểu cũng như phát triển các hệ vi xử lý phức tạp.

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 cuốn "Kỹ thuật Vi xử lý" của tác giả Văn Thế Minh, các tài liệu tham khảo khác, 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, 2011

Tác giả

MỤC LỤC

Chương 1. TỔNG QUAN VỀ VI XỬ LÝ VÀ HỆ VI XỬ LÝ

1. GIỚI THIỆU VỀ VI XỬ LÍ

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...

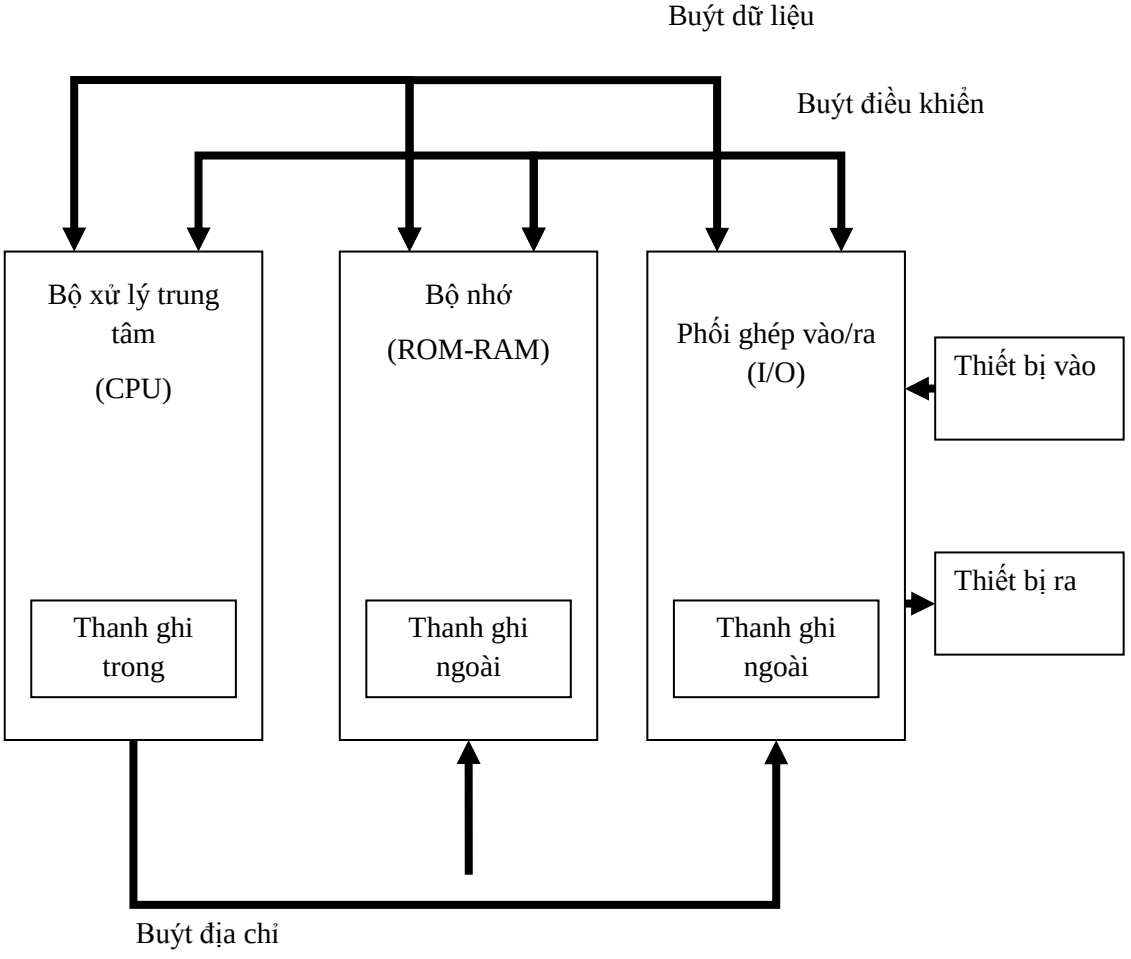
Máy tính con (*Minicomputer*) là một dạng thu nhỏ về kích thước cũng như về tính năng của máy tính lớn. Nó ra đời nhằm thoả mãn các nhu cầu sử dụng máy tính cho các ứng dụng vừa phải mà nếu dùng máy tính lớn vào đó thì sẽ gây lãng phí. Máy tính con thường được dùng cho các tính toán khoa học kỹ thuật, gia công dữ liệu quy mô nhỏ hay để điều khiển quy trình công nghệ.

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.

2. HỆ VI XỬ LÍ

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 1-1 giới thiệu sơ đồ khối tổng quát của một hệ vi xử lý.



Hình 1-1. 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 (CPU)
- Bộ nhớ bán dẫn (ROM-RAM)
- Khối phối ghép với các thiết bị ngoại vi (I/O)

– Các buýt truyền thông tin.

Ba khối chức năng đầu liên hệ với nhau thông qua qập các đường dây 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 ta có thể lưu 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.

3. CÁC ĐẶC ĐIỂM CẤU TRÚC CỦA VI XỬ LÍ

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 1-2.

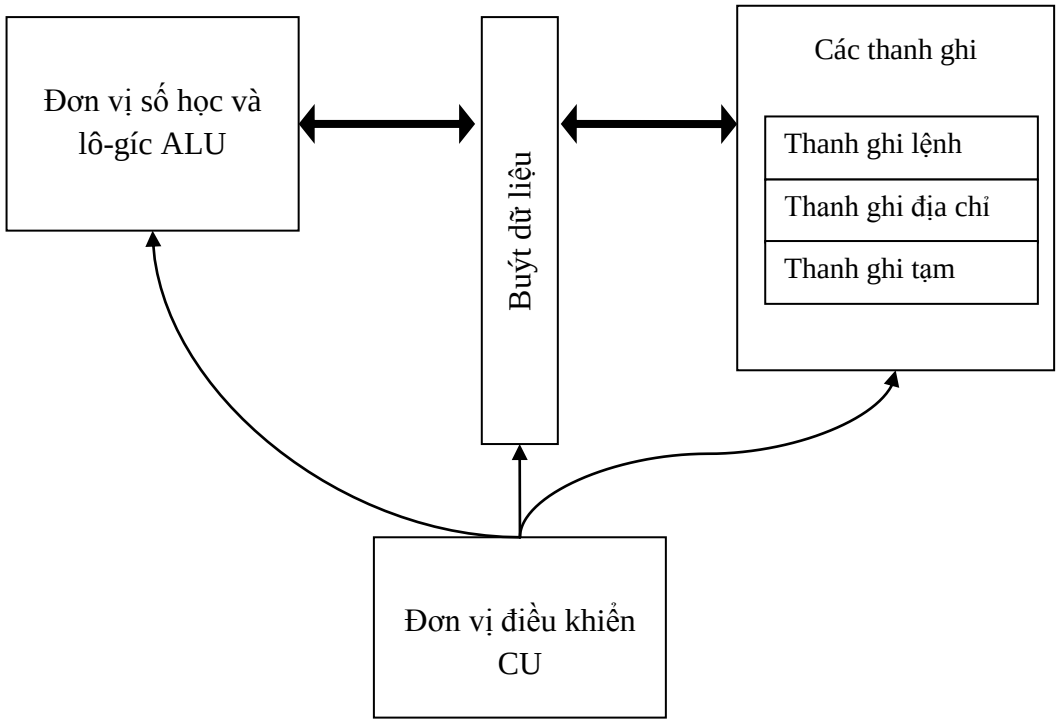
3.1.1 Các thanh ghi

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.



Hình 1-2. Sơ đồ khối chức năng vi xử lý

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 nhớ. 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 tích lũy (*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.

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 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.

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), …

3.1.3.a 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.

3.1.4 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.

4. LỊCH SỬ PHÁT TRIỂN VÀ PHÂN LOẠI CÁC BỘ VI XỬ LÍ

Phần này giới thiệu quá trình 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.

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.

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. 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.

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. Ưu điểm 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 64K 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.

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. 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 và các siêu máy tính. 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 (*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.

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 hịp 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.

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.

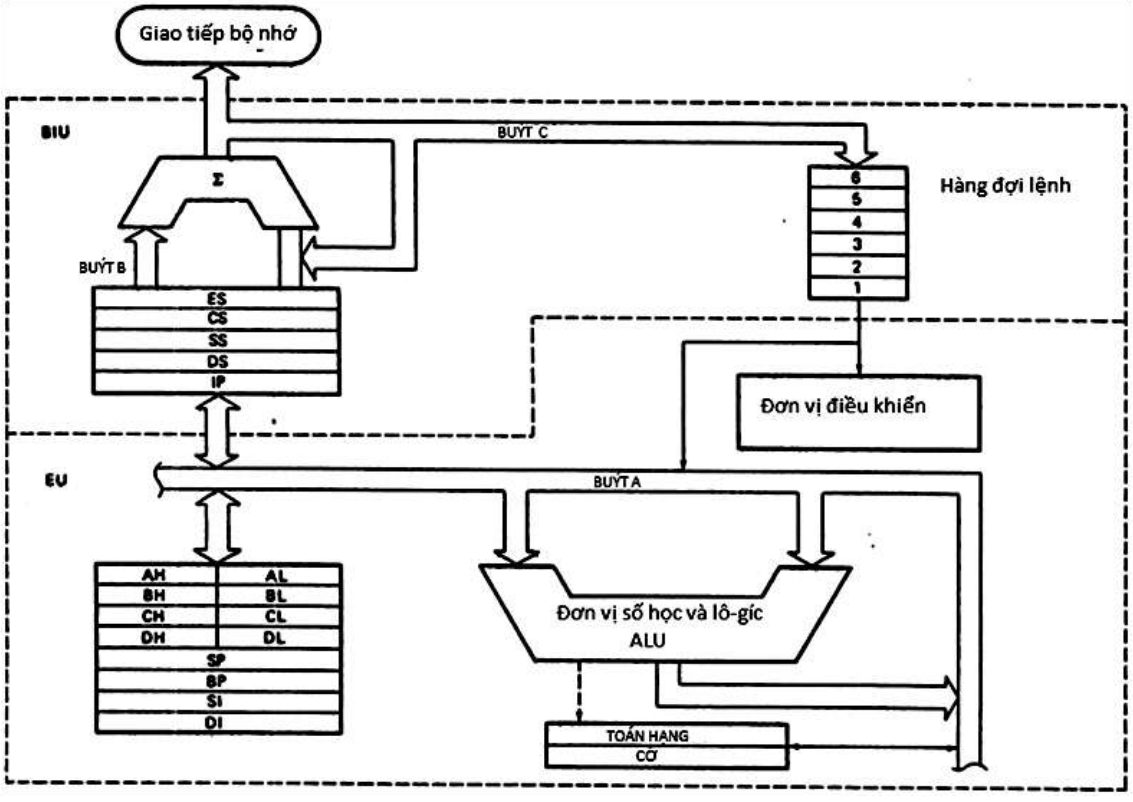
Bên cạnh các bộ vi xử lý cho máy PC và máy chủ, các hãng sản xuất vi xử lý cũng phát triển các dòng vi xử lý nhúng cho các thiết bị tính toán cá nhân. Ưu thế của các vi xử lý nhúng so với vi xử lý kể trên là mức tiêu thụ điện năng, năng lực xử lý và chi phí. Intel cung cấp các vi xử lý nhúng Atom có 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 2. BỘ XỬ LÝ INTEL 8086

1. CẤU TRÚC BÊN TRONG CỦA 8086

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. 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 vẫn có thể chạy trên các hệ thống tiên tiến sau này.

1.1 Sơ đồ khối



Hình 2-1. 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.

1.1.1 Đơn vị giao tiếp buýt và thực thi EU

Theo sơ đồ khối trên Hình 2-1 CPU 8086 có 2 khối chính: *khối phối ghép* 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).*

1.1.2 Các thanh ghi

1.1.2.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 Segment*)
- 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 1.2.

1.1.2.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 ghi tích lũy. Các kết quả 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 chứa.
- BX (*base*): thanh ghi cơ sở thường chứa địa chỉ cơ sở của một bảng.
- 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.

1.1.2.c Các thanh ghi con trỏ và chỉ số

Trong 8086 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 như trình bày trong phần 1.2.
- 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 như trình bày trong phần 1.2.

- 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 như trình bày trong phần 1.2.
- 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 như trình bày trong phần 1.2.
- 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 như trình bày trong phần 1.2.

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.

1.1.2.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.

Hình 2-2. 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 lập hoặc xoá tuỳ theo các điều kiện cụ thể sau các thao tác của ALU. Ngoài ra, bộ vi xử lý 8086 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)

1.2 Phân đoạn bộ nhớ của 8086

Khối BIU đưa ra trên buýt địa chỉ 20 bít địa chỉ, như vậy 8086 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. 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) 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 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 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_đọan×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 2-1).

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

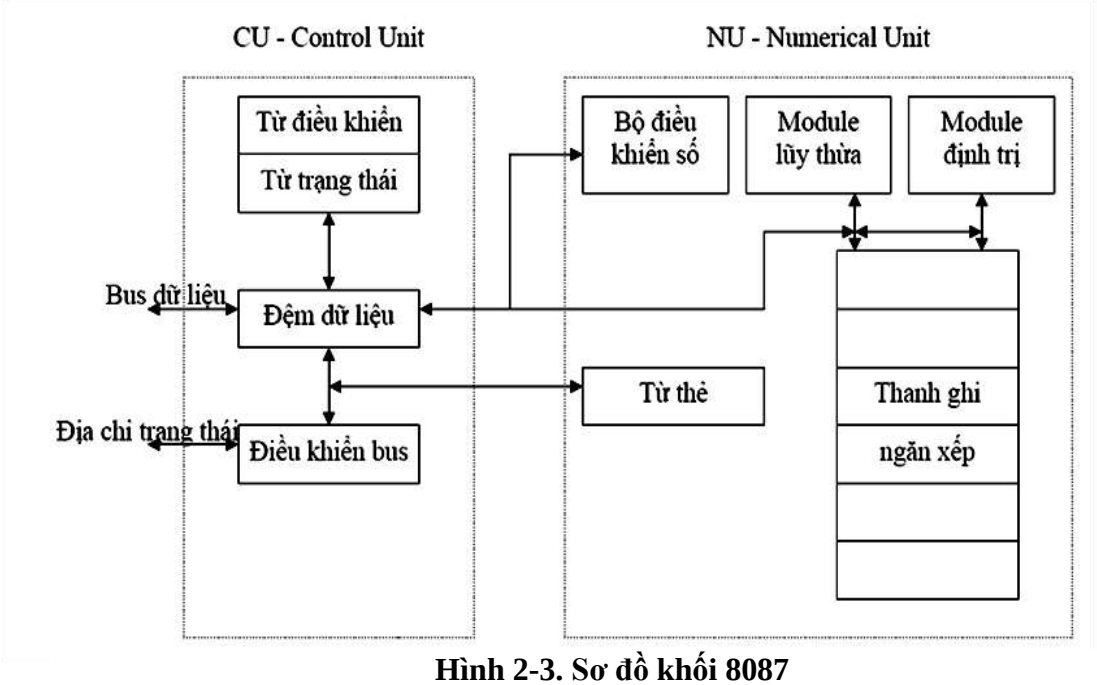
Ví dụ: Địa chỉ vật lý 12345H có thể được tạo ra từ các giá trị:

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

2. BỘ ĐỒNG XỬ LÍ TOÁN HỌC 8087

Như được trình bày trong phần trước, 8086 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.

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 2-3. 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 bằng các đường BHE /S7. Bộ đồng xử lý 8087 sẽ điều chỉnh độ dài của hàng lệnh cho phù hợp với CPU.

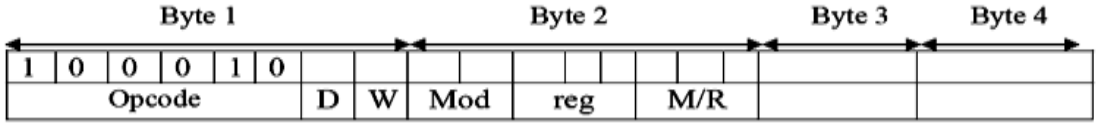
3. TẬP LỆNH CỦA 8086

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 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. Do lệnh 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 thiết kế 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 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 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.

Lệnh MOV *đích, gốc* dùng để chuyển dữ liệu giữa thanh ghi và ô nhớ. Chỉ nguyên với các thanh ghi của 8086, 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. Như vậy để mã hoá lệnh MOV 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 R/M (thanh ghi/bộ nhớ) 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)

3.2 Các chế độ địa chỉ của 8086

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 8086 và cả họ 80x86 nói chung đều có 7 chế độ địa chỉ sau:

1. Chế độ địa chỉ thanh ghi (*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*).

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 2-1

```
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
```

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 2-2

```
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.
```

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.

Ví dụ 2-3

```
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
```

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 không cho phép tham chiếu bộ nhớ 2 lần đối với một lệnh).

Ví dụ 2-4

```
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).
```

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 2-5

```
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.

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 2-6

```
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.
```

3.2.7 Phương pháp bỏ ngầm định thanh ghi đoạn

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 2-1 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 còn cho phép ta làm việc với các tổ hợp ngầm định đã kể, 8086 còn cho phép ta làm việc với các tổ hợp khác của các thanh ghi đoạn và thanh ghi lệch. Muốn loại bỏ các tổ hợp ngầm định nói trên, trong khi viết lệnh phải ghi rõ thanh ghi đoạn sẽ dùng để tính địa chỉ.

Bảng 2-1. 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.

3.3 Tập lệnh của 8086

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.

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 2-2. 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

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)]
```

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
```

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 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.

Địa chỉ cổng có thể được lưu trong thanh ghi DX. Cách này địa chỉ cổng hoá mềm dẻo hơn. Lúc này địa chỉ cổng nằm trong dải 0000H. . FFFFH và câu lệnh có 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.

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 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.

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.

Bảng 2-3. 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

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 nhó 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

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 để trong AL.

sau khi nhân: AX ← tích,

Nếu Gốc là số 16 bít: AX×Gốc,

số bị nhân phải là số 16 bít để 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 cho biết có thể bỏ đi bao nhiêu số 0 trong kết quả. Ví dụ: Nếu cần nhân một số 8 bít với một số 16 bít, số 16 bít đặt 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.

*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 để trong AX.
- Nếu Gốc là số 16 bít: DXAX/Gốc. Số bị chia phải là số không dấu 32 bít để 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 8086 thực hiện lệnh ngắt INT 0.

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

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.

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 là có các bít 0/1 ở các chỗ cần che đi/giữ nguyên tương ứng (toán hạng 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.
```

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.

Bảng 2-4. 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

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

Lệnh này khiến cho bộ vi xử lý 8086 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.

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

Viết lệnh: LOOP NHAN

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 số lần lặp mong muốn phải được nạp 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.

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 2-5. 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

4. NGẮT VÀ XỬ LÍ NGẮT TRONG 8086

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.

Trong các tín hiệu của CPU 8086 có tín hiệu cho các yêu cầu ngắt che được INTR và không che được NMI, chính các tín hiệu này sẽ được sử dụng vào việc đưa các yêu cầu ngắt từ bên ngoài đến CPU.

4.2 Các loại ngắt trong hệ 8086

Trong hệ vi xử lý 8086 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. **Error! Reference source not found.** 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.

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 2-6 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)

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 đic 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 địa chỉ 00000H và dài 1 KB (vi xử lý 8086 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 2-7. Bảng vectơ ngắt của 8086 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

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 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 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 chia 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 3. LẬP TRÌNH HỢP NGỮ VỚI 8086

1. GIỚI THIỆU KHUNG CỦA CHƯƠNG TRÌNH HỢP NGỮ

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.

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. . .

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 để trong bộ nhớ thì byte thấp của nó sẽ được để ở ô nhớ có địa chỉ thấp, byte cao sẽ được để ở 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.

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.

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 3-1.

Bảng 3-1. 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

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 ví dụ sau:

```
. Data
    MSG   DB        'helo!$'
    CR    DB        13
    LF    EQU       10
```

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.

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. Để chương trình có thể chạy đúng 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, 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 3-1. 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.

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

Trên máy tính IBM PC ngoài tệp chương trình với đuôi .EXE, có thể 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 dạng 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. Với việc tạo ra tệp này còn tiết kiệm được cả

không gian nhớ khi phải lưu trữ 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.

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

Ví dụ 3-2. 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).

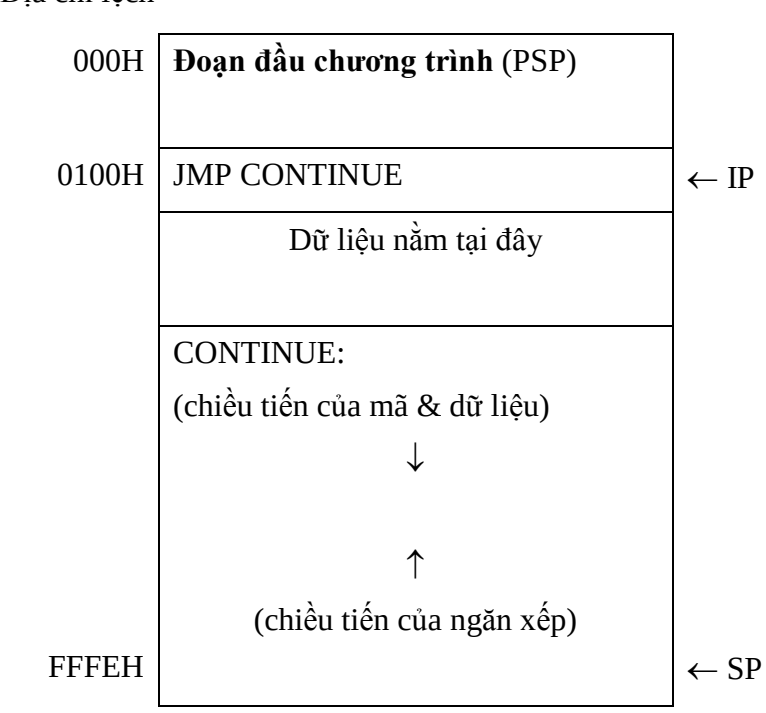
Lệnh JMP sau nhãn START dùng để nhảy qua phần bộ nhớ dành cho việc định nghĩa và khai báo dữ liệu (về nguyên tắc, dữ liệu có thể được đặt ở đầu hoặc ở cuối đoạn mã, nhưng ở đây ta đặt nó ở đầu đoạn mã để có thể áp dụng các định nghĩa đơn giản đã nói). Đích của lệnh nhảy là phần đầu của chương trình chính. Hình 3-1 biểu diễn việc một chương trình kiểu .COM được nạp vào và sắp xếp trong một đoạn mã của bộ nhớ ra sao.

Trong Hình 3-1 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 có thể làm cho đỉnh ngăn xếp trong khi hoạt động dâng lên nhiều về phía địa chỉ thấp của đoạn).



Hình 3-1. Tệp chương trình .COM trong bộ nhớ

Tóm lại phải chắc chắn 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 cần dùng ngắt INT 20H của DOS để làm cho chương trình gọn hơn. Tất nhiên 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, 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.

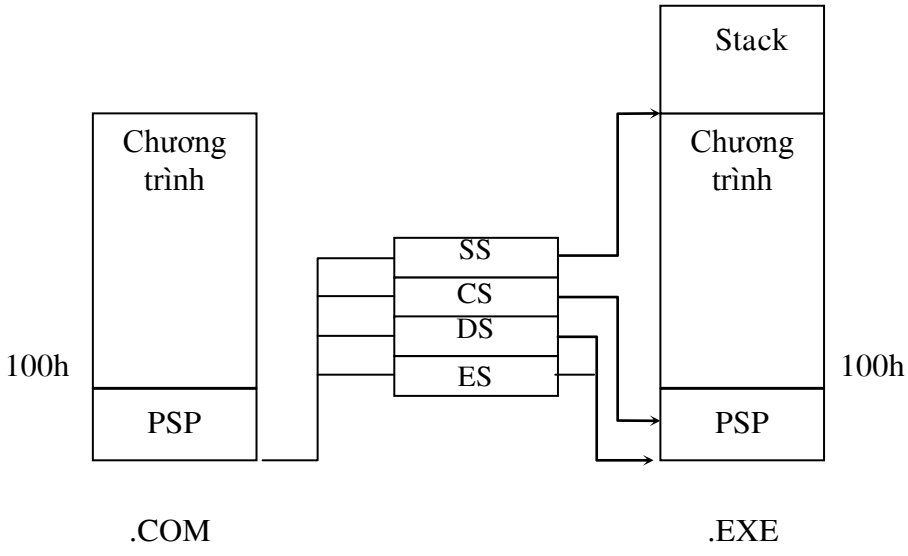
Sau đây là ví dụ của một chương trình hợp ngữ để dịch ra tệp chương trình chạy được với đuôi .COM.

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



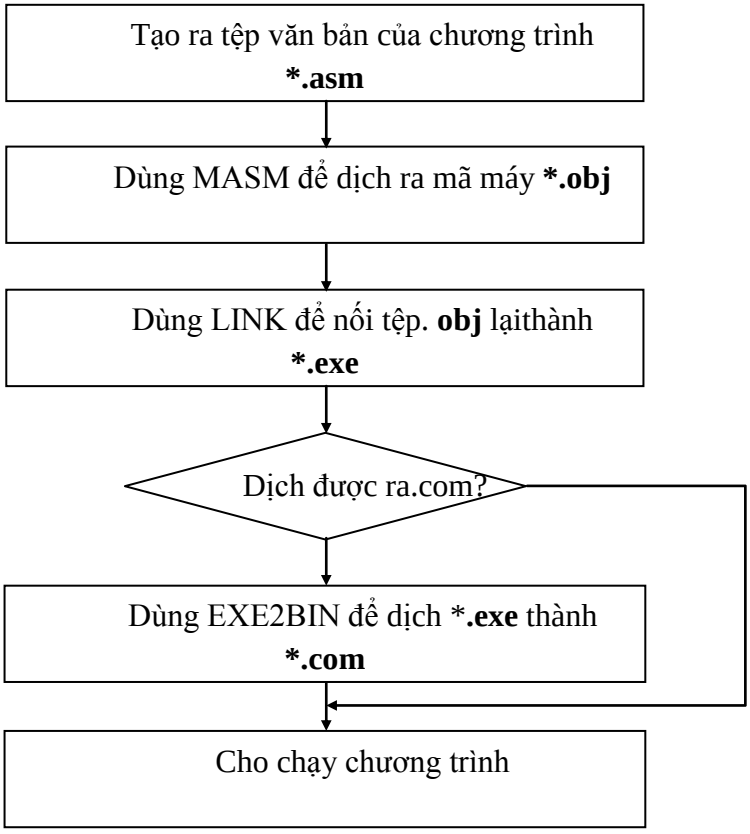
Hình 3-2. Môđun chương trình. COM và. EXE trong bộ nhớ.

Hình 3-2 biểu diễn cấu trúc của các chương trình .COM và .EXE khi chúng được tải vào trong bộ nhớ.

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. Các bước bao gồm:

1. Dùng các phần mềm soạn thảo văn bản 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 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 3-3. Các bước để tạo ra và chạy chương trình hợp ngữ

3. CÁC CẤU TRÚC LẬP TRÌNH CƠ BẢN

Trong khi thực hiện các khối chức năng thành phần của chương trình, 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 dễ phát triển, dễ hiệu chỉnh hoặc cải tiến và dễ lập tài liệu.

Các cấu trúc lập trình cơ bản bao gồm:

+ Cấu trúc tuần tự.

+ Cấu trúc lựa chọn (IF-THEN-ELSE) và

+ Cấu trúc lặp (WHILE. DO).

Thay đổi các cấu trúc này 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* chỉ có một lối vào cấu trúc và một lối ra để ra khỏi cấu trúc đó.

3.1 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 3-1

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

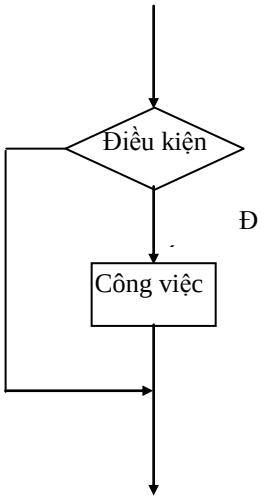
Ta có thể thực hiện công việc trên bằng mẫu chương trình sau:

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

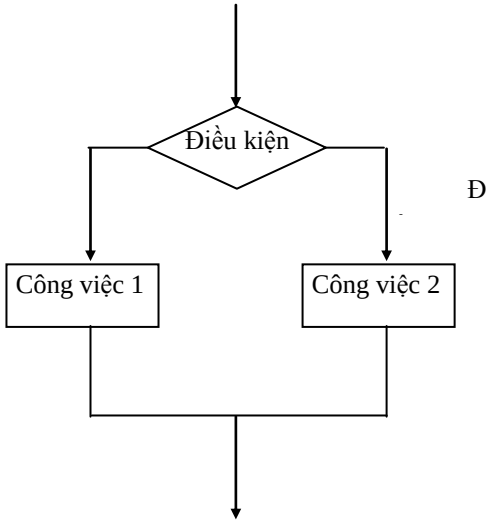
3.1.1 Cấu trúc IF - THEN

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 3-6.Cấu trúc IF-THEN



Hình 3-5 Cấu trúc IF-THEN-ELSE

Bài tập 3-2.

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.
```

3.1.2 Cấu trúc IF - THEN - ELSE

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

Từ ngữ pháp của cấu trúc IF-THEN-ELSE, 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 trình.

Bài tập 3-3.

Gán cho CL giá trị bít dấu của AX.

Giải

Ta có thể thực hiện các công việc trên bằng 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.
```

3.1.3 Cấu trúc CASE

CASE **Biểuthức**

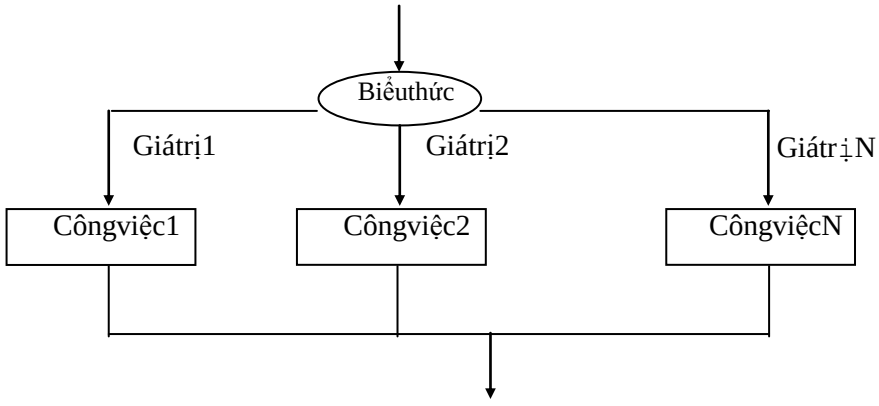
Giátrị1: Côngviệc1

Giátrị2: Côngviệc2

. . .

GiátrịN: CôngviệcN

END CASE



Hình 3-7. 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 3-4.

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

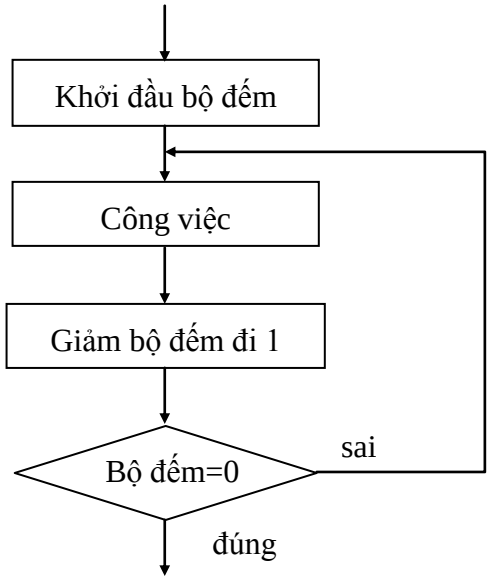
Ta có thể thực hiện các công việc trên bằng mẫu chương trình sau:

```
        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.
```

3.1.4 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 3-8. Cấu trúc lặp FOR - DO.

Bài tập 3-5

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.

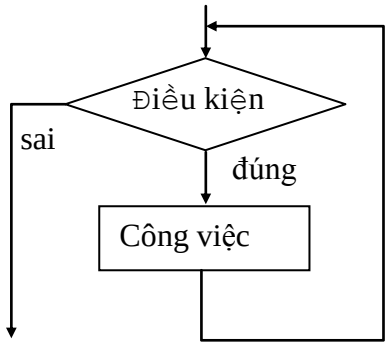
Sau đây là mẩu chương trình thực hiện các công việc trên:

```
        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.
```

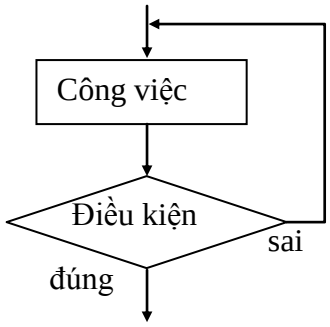
3.1.5 Cấu trúc lặp WHILE - DO

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 3-10. Cấu trúc *WHILE - DO*



Hình 3-10. Cấu trúc *REPEAT - UNTIL*

Bài tập 3-6

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 mẫu 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.
```

3.1.6 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 3-7

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.
```

4. MỘT SỐ VÍ DỤ

Trong phần này giới thiệu 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.

Dưới đây là một số hàm của các loại ngắt có trong máy IBM PC với hệ điều hành MS DOS.

Bảng 3-2. 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

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 bỏ bớt đi các dòng cách ở đầu và cuối để chương trình.

```
. 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 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, cần phải gắn giá trị khác cho thanh ghi CX.

4.2 Ví dụ 2

Trên cơ sở ví dụ trước, hãy 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, 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 chú ý rằng giữa giá trị số và mã ASCII của số đó chênh nhau 30H. Ví dụ số 9 có mã ASCII là 39H hay 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 có thể làm được bằng một trong các lệnh sau:

+ ADD DL, 30H

+ OR DL, 30H

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 là đọc được số hệ hai và cất nó tại BX, trong phần tiếp theo 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.

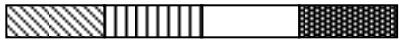
Một số Chú ý từ chương trình trên:

- Lệnh xóa thanh ghi BX là rất cần thiết để sau này khi gõ vào các bít của nó không nhất thiết phải gõ đủ 16 bít mà vẫn xác định được giá trị của thanh ghi này.
- Chương trình này dùng lệnh ROL để quay tròn thanh ghi BX, vì vậy sau khi quay và hiện thị tất cả 16 bít của BX, giá trị của thanh ghi BX vẫn được bảo toàn.
- Chương trình này 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.

4.4 Ví dụ 4

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

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
```

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. Các bước thực hiện:

- 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.

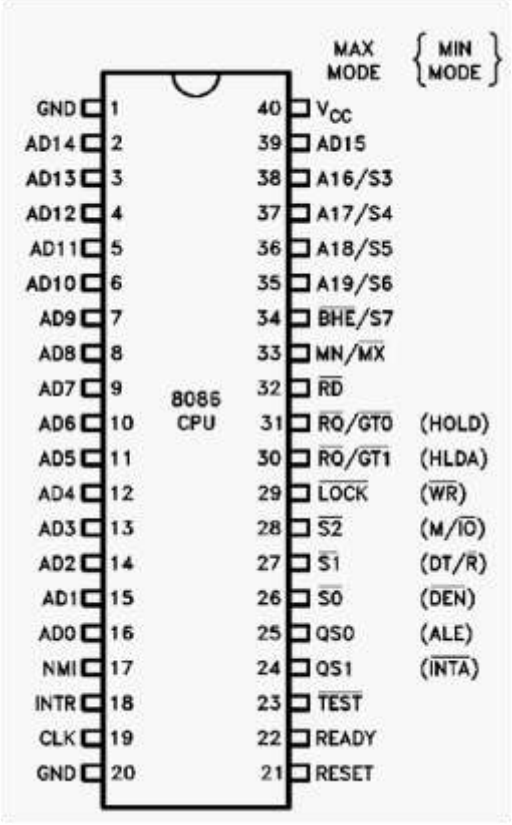
```
. Model Small
. Stack 100
. Data
        Str1    DB 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'
        Tbao    DB 'chuỗi đã được đổi: ', 10, 13
                DB '$'
. Code
MAIN Proc
        MOV  AX, @Data    ; khởi đầu đầu cho DS và ES
        MOV  DS, AX
        MOV  ES, AX
        LEA  SI, Str1+7   ; 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 4. PHỐI GHÉP VI XỬ LÍ VỚI BỘ NHỚ VÀ CÁC THIẾT BỊ VÀO/RA

1. CÁC TÍN HIỆU CỦA VI XỬ LÍ VÀ CÁC MẠCH PHỤ TRỢ

1.1 Các tín hiệu của 8086



Hình 4-1. Tín hiệu 8086

Hình vẽ trên cho chúng ta thấy 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.

Bảng 4-1. 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

Bít S6 = 0 liên tục, bít S5 phản ánh 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 như trong bảng.

- $\overline{RD}$ [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 tín hiệu này 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 chuyển sang thực hiện chương trình phục vụ ngắt kiểu INT2.
- RESET [I]: tín hiệu khởi động lại 8086. khi RESET = 1 kéo dài ít nhất trong thời gian 4 chu kỳ đồng hồ thì 8086 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 ở chế độ chạy từng lệnh).

- CLK [I]: Tín hiệu đồng hồ (xung nhịp).cung cấp xung nhịp làm việc cho CPU.
- Vcc [I]: Chân nguồn cung cấp 5V
- GND [O]: điểm 0V của nguồn nuôi.
- MN/MX [I]: Chân điều khiển hoạt động của CPU theo chế độ MIN/MAX.
  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.
     - 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 (các buýt ở trạng thái trở kháng cao) để bộ điều khiển DMA (*DMA Contrroller*) lấy quyền điều khiển hệ thống để thực hiện trao đổi dữ liệu.

Bảng 4-2. 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 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)

- HLDA [O]: Tín hiệu báo cho bên ngoài biết yêu cầu treo CPU để dùng các buýt đã được chấp nhận, và CPU 8086 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 mạch điều khiển buýt 8288. Chế độ MAX được sử dụng khi có 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 như trong bảng dưới đây.

Bảng 4-3. 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 (nghỉ)	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. $\overline{RQ}/\overline{GT0}$ 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) như trong Bảng 4-4. Khi có bộ đồng hồ xử lý toán học 8087, các tín hiệu này được mạch 8087 dùng để đồng bộ quá trình hoạt động với bộ vi xử lý 8086.

Bảng 4-4. Các trạng thái của lệnh đệm

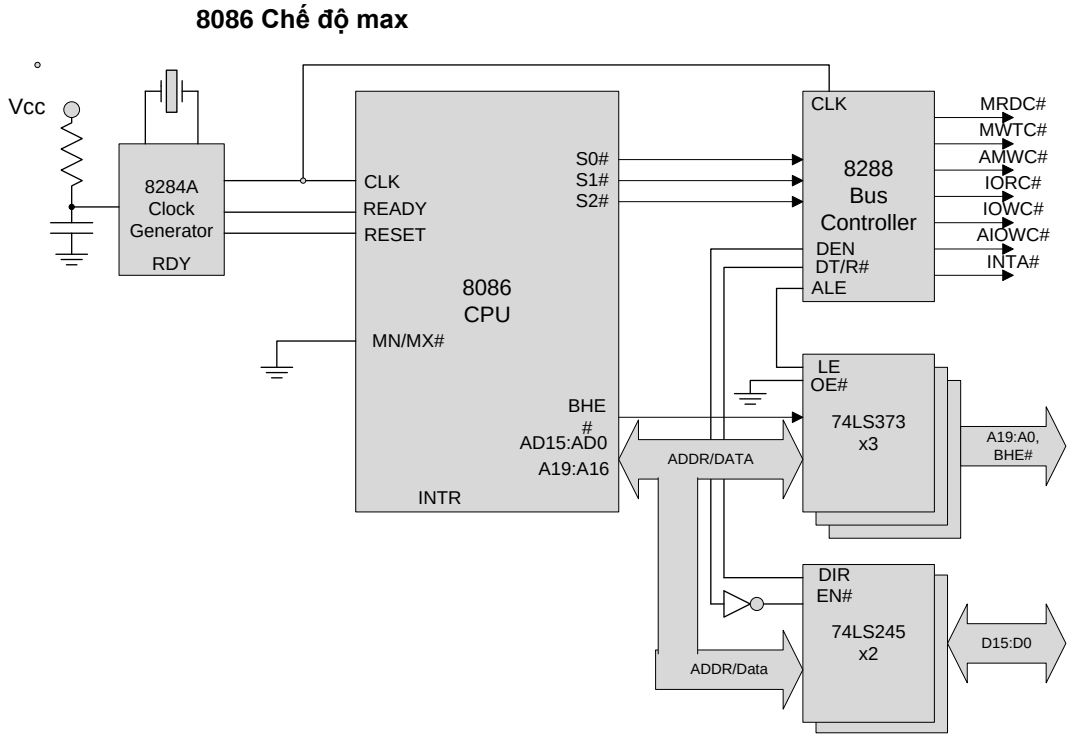
QS1	QS0	Trạng thái lệnh đệm
0	0	Không hoạt động
0	1	Đọc byte mã lệnh đầu tiên từ đệm lệnh
1	0	Đọc lệnh rỗng
1	1	Đọc byte tiếp theo từ đệm lệnh

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

Để hạn chế số lượng chân cho các tín hiệu của vi mạch CPU, người ta đã hạn chế số chân của vi mạch bằng cách dồn kênh nhiều tín hiệu trên cùng một chân. 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, cần 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). Đối với các chân dồn địa chỉ/trạng thái cũng phải làm tương tự. Để hỗ trợ cho việc tách thông tin, 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 4-3. Buýt hệ thống 8086 có đệm

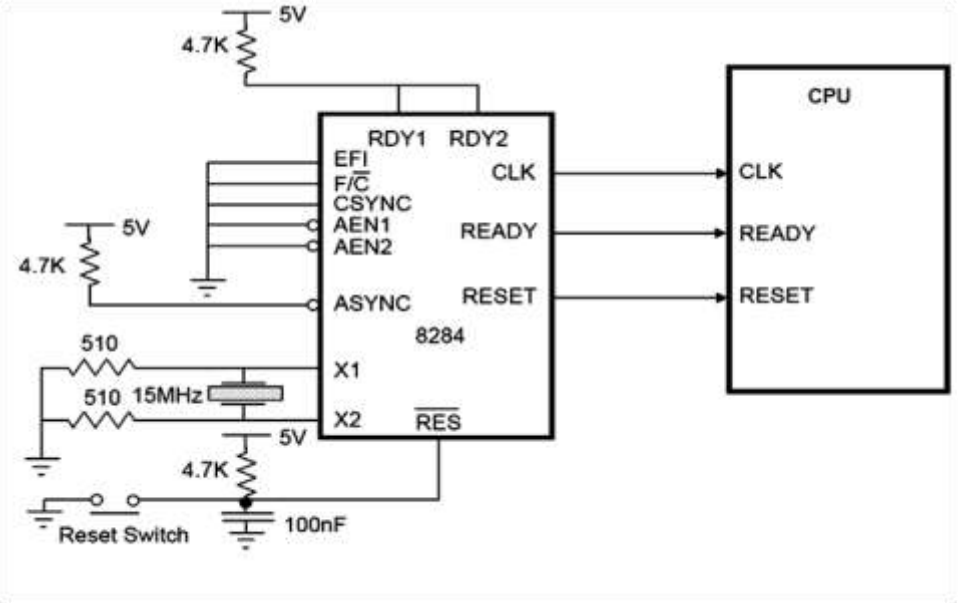
Hình 4-3 cho thấy một ví dụ về việc tách tín hiệu địa chỉ từ các tín hiệu dồn kênh chỉ/dữ liệu hoặc địa chỉ/điều khiển bằng các mạch chốt 74LS373 và việc sử dụng các bộ khuếch đại đệm 74LS244 và 74LS245 cho các tín hiệu của bộ vi xử lý 8086 làm việc ở chế độ MAX. Hình 4-3 còn thể hiện ghép nối với các mạch phụ trợ như: bộ điều khiển buýt 8288, bộ tạo ra xung đồng hồ 8284.

1.3 Mạch tạo xung nhịp 8284.

CPU 8086 luôn cần xung nhịp (xung đồng hồ) từ mạch tạo xung nhịp 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 4-4). Ý nghĩa của các tín hiệu như sau:

- $\overline{AEN1}$, $\overline{AEN2}$: 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.
- 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 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 4-4)

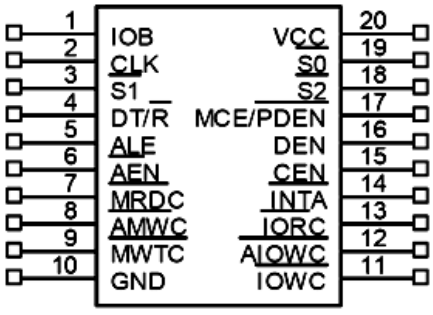
Hình 4-4 biểu diễn các đường nối tín hiệu chính của 8086 và 8284. Mạch 8284 nhận được xung khởi động từ bên ngoài thông qua mạch RC khi có nguồn hoặc xung khởi động lại khi bấm công tắc Reset. Từ xung này 8284 có nhiệm vụ đưa ra xung khởi động đồng bộ cho CPU cùng với tất cả các thành phần khác của hệ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



Hình 4-4. Nối 8284 với vi xử lý

1.4 Mạch điều khiển buýt 8288

Vi mạch 8288 là mạch điều khiển buýt sử dụng 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 4-5. 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.
- CLK [I]: 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]: 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]: 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]: 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).
- DEN [O]: đây là tín hiệu để điều khiển buýt dữ liệu trở thành buýt cục bộ hay buýt hệ thống.
- MCE/$\overline{PDEN}$ [O]: 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]: tín hiệu cho phép chốt địa chỉ tại các chân dồn kênh địa chỉ - dữ liệu AD0 - AD7.

1.5 Biểu đồ thời gian của các lệnh ghi/đọc

Hình 4-6 và Hình 4-7 là các biểu đồ thời gian đã được đơn giản hoá của các tín hiệu cơ bản trong CPU 8086 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.

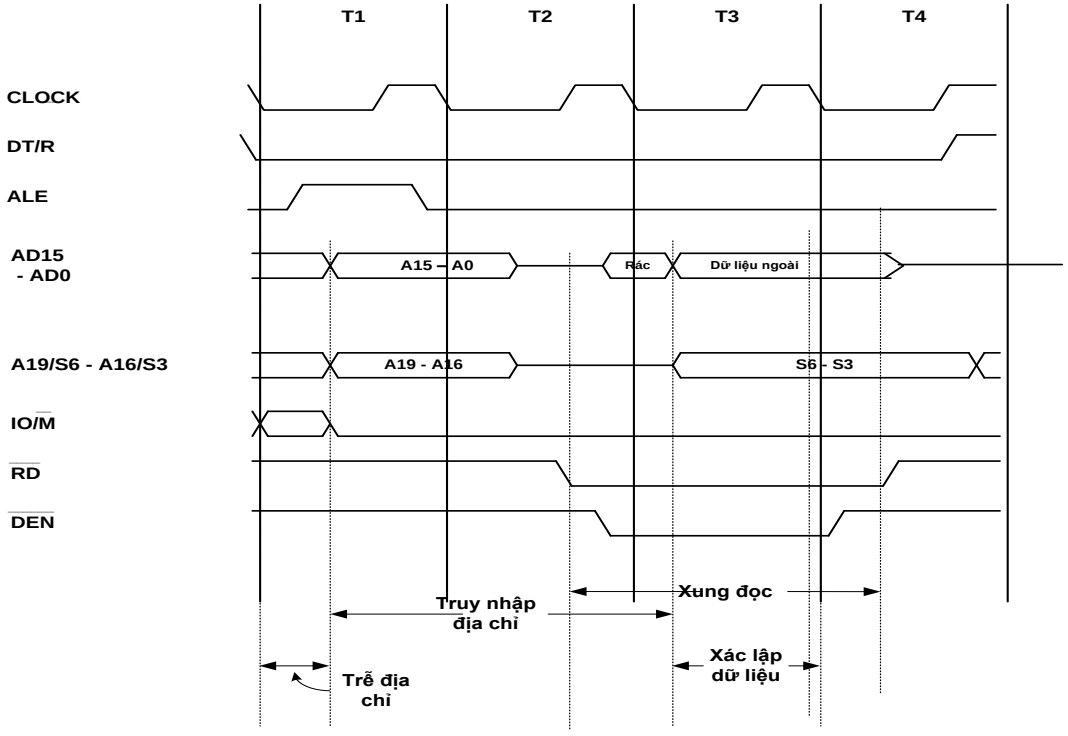
Chúng ta mô tả tóm tắt các hiện tượng xảy ra trong một chu kỳ T nói trên.

Chu kỳ T1:

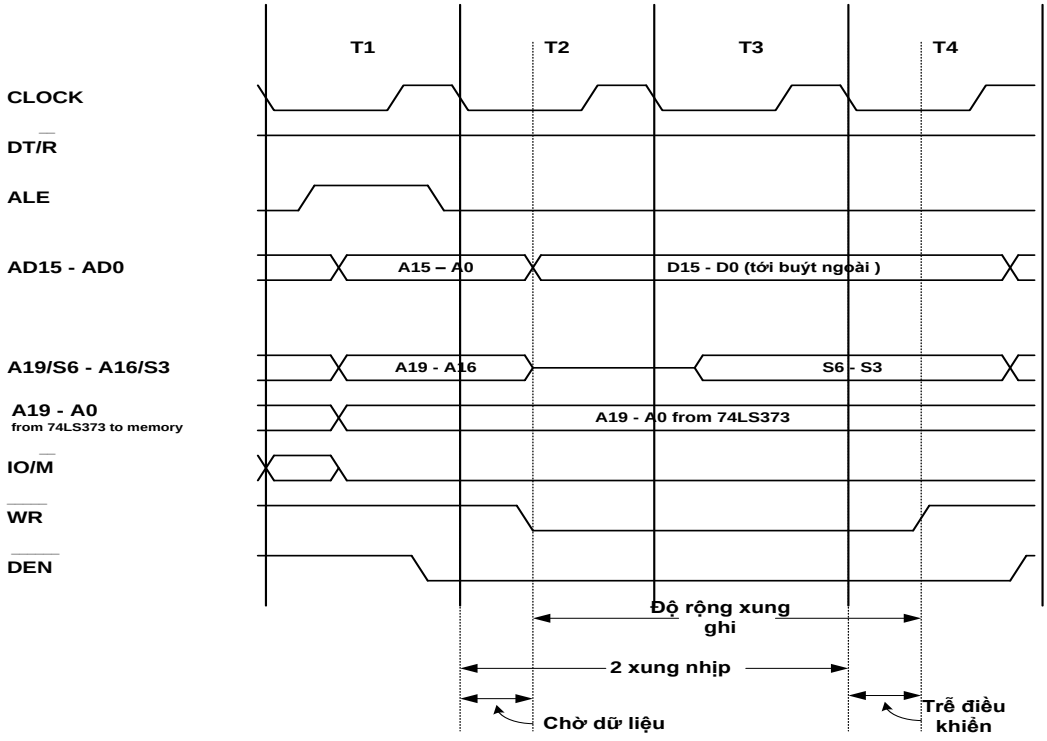
Trong chu kỳ này địa chỉ của bộ nhớ hay thiết bị ngoại vi được đưa ra trên các đường địa chỉ, hoặc địa chỉ/dữ liệu và địa chỉ/ trạng thái. Các tín hiệu điều khiển ALE, DT/ $\overline{R}$,IO/ $\overline{M}$ cũng được đưa ra để giúp việc hoàn tất việc giữ thông tin địa chỉ này.

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 4-6. Biểu đồ đọc đơn giản hóa



Hình 4-7. 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 4-6) 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 chuyền 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 - 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 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 4-7) 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 - 5MHz là t_{WR} = 340ns cũng là phù hợp với các bộ nhớ với thời gian truy nhập cỡ 450ns.

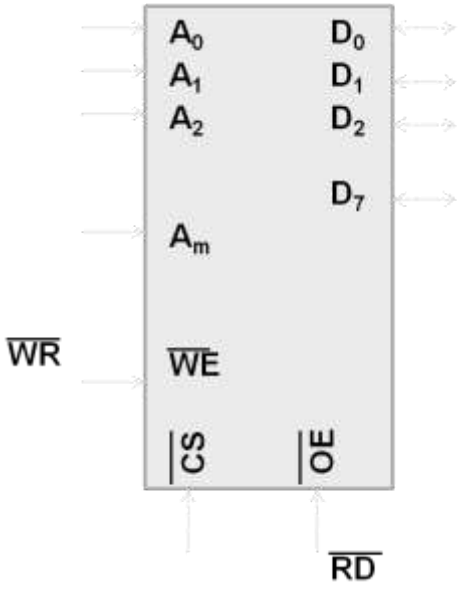
2. PHỐI GHÉP VI XỬ LÍ VỚI BỘ NHỚ

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*) hay bộ nhớ 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 (*Erasable Programmable ROM*) là bộ nhớ ROM có thể lập trình được bằng xung điện và xoá được bằng tia cực tím.
- Bộ nhớ truy nhập ngẫu nhiên RAM (*Random Access Memory*) thông tin ghi trong mạch bị mất khi mất nguồn điện nuôi cho mạch. Trong các bộ nhớ RAM còn phân biệt ra loại RAM tĩnh (*Static RAM*), trong đó mỗi phần tử nhỏ là một mạch lật hay trạng thái ổn định) và loại RAM động (*Dynamic RAM*) trong đó mỗi phần tử nhớ là một tụ điện rất nhỏ được chế tạo bằng công nghệ MOS.

Một bộ nhớ thường được xây dựng từ nhiều vi mạch nhớ. Một vi mạch nhớ thường có dạng cấu trúc tiêu biểu như hình sau đây.



Hình 4-8. 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 của vi mạch 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 dung lượng mạch nhớ là 1K ô nhớ (1 kilô = 2^{10} = 1024) và với m=20 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 loại 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) 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. Khi mạch nhớ không được mở bởi $\overline{OE}$ thì buýt dữ liệu được 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.

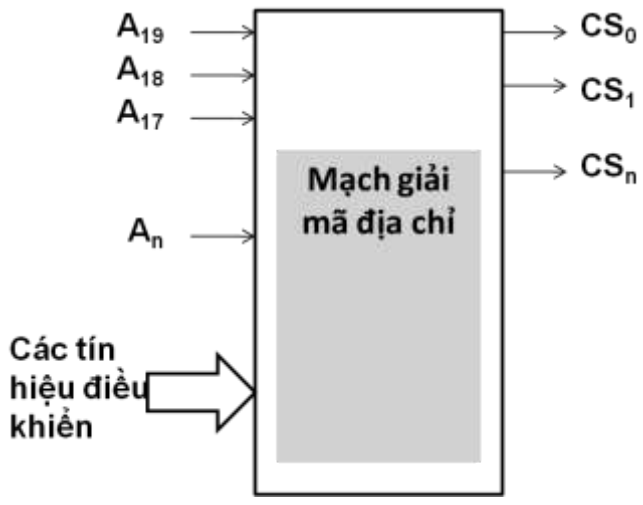
Một thông số đặc trưng khác của bộ nhớ là thời gian truy nhập t_{ac} được định nghĩa như là thời gian kể từ khi có xung địa chỉ trên buýt địa chỉ cho đến khi có dữ liệu ra ổn định trên buýt dữ liệu. Thời gian truy nhập bộ nhớ phụ thuộc rất nhiều vào công nghệ chế tạo.

2.2 Giải mã địa chỉ cho bộ nhớ

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. Việc phân vùng ô nhớ tùy thuộc vào thiết kế của hệ vi xử lý.

Về nguyên tắc một bộ giải mã địa chỉ khái quát thường có cấu tạo như trên Hình 4-9 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 4-9. 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. 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.

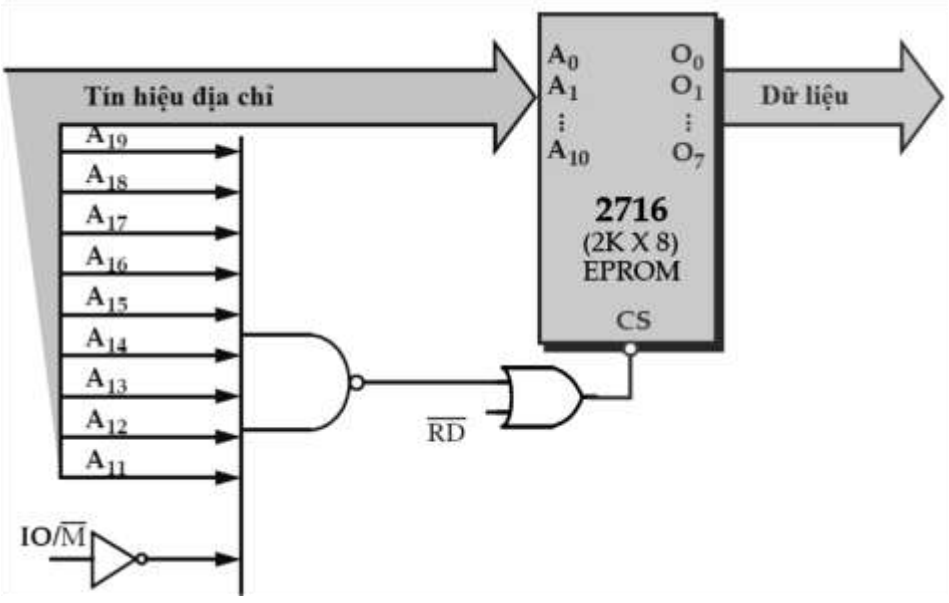
Giải mã địa chỉ thiếu hay giải mã rút gọn thì ta chỉ dùng một nhóm trong số các tín hiệu địa chỉ để sinh ra tín hiệu chọn vỏ cho mạch nhớ. Như vậy, từ một tổ hợp các tín hiệu địa chỉ có thể sinh ra nhiều tín hiệu chọn vỏ khác nhau. Vì sử dụng ít tín hiệu hơn nên mạch giải mã thiếu cần ít linh kiện hơn 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 dư ra một chút để 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.

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.



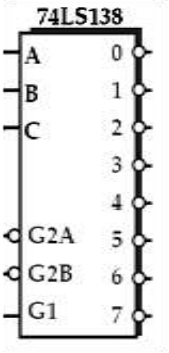
Hình 4-10. Mạch giải mã dùng mạch lô-gíc

Hình 4-10 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 tín 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.

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.



Inputs						Output							
Enable			Select										
G2A	G2B	G1	C	B	A	0	1	2	3	4	5	6	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 4-11. 74LS138 và bảng trạng thái

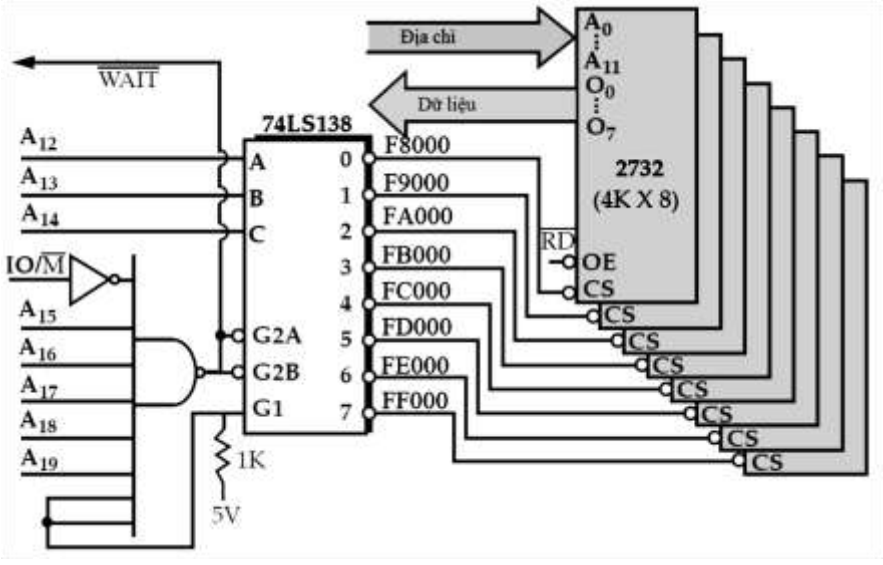
Giả sử chúng ta cần xây dựng mạch giải mã cho không gian nhớ 256KB tương ứng với dải địa chỉ F8000H-FFFFFH trong đó mỗi mạch nhớ 2732 có dung lượng 4K×8. Từ dải địa chỉ được gán và dung lượng của từng mạch nhớ, có thể thấy rằng từ các tín hiệu địa chỉ A_{13} tới A_{19} cần phải sinh ra 8 tín hiệu kích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

Bảng 4-5. 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:



Hình 4-12. Giải mã sử dụng 74LS138

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 ví 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.

2.2.4 Thực hiện bộ giải mã dùng PROM

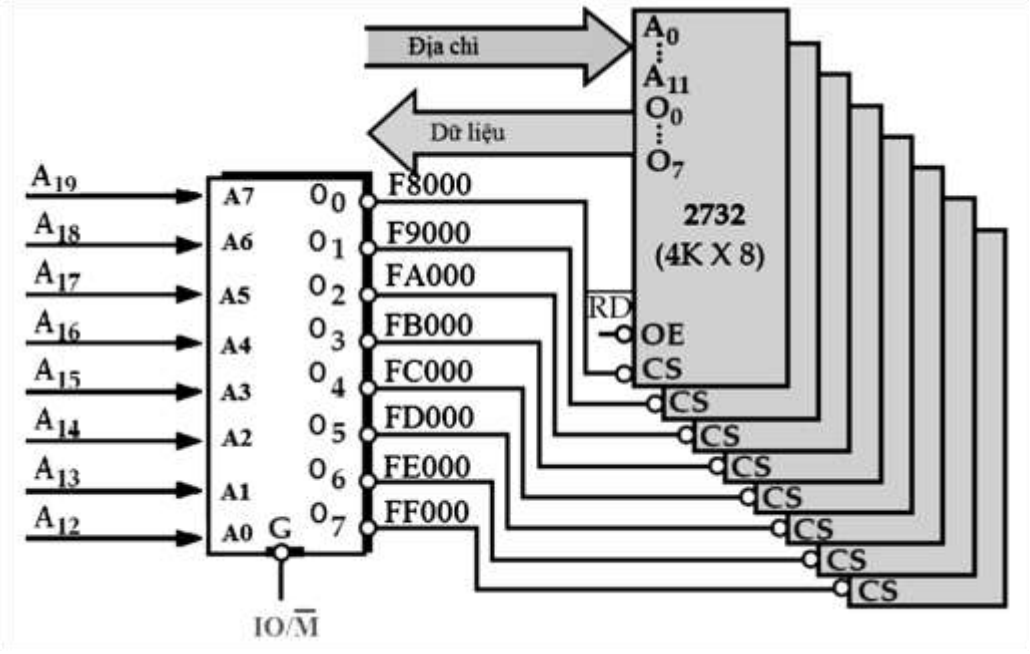
Việc sử dụng bộ nhớ ROM làm bộ giải mã lợi dụng số lượng lớn các tín hiệu địa chỉ đầu vào, điều khiển và dữ liệu ra của mạch nhớ ROM. Với mỗi tổ hợp tín hiệu địa chỉ và điều khiển đầu vào, mạch nhớ ROM sẽ sinh ra một nhóm tín hiệu trên kênh dữ liệu. Trạng thái của các tín hiệu dữ liệu này tùy thuộc vào 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.

Dưới đây sử dụng mạch PROM 256 byte để làm bộ giải mã cho ví dụ phân vùng bộ nhớ cho ROM trong phần trước. Trong bảng dưới đây là mẫu các bít để ghi vào PROM cho trường hợp cụ thể này.

Bảng 4-6. 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 4-13. 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.

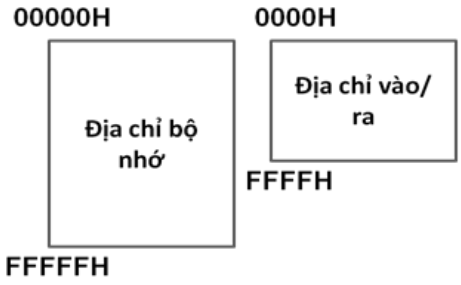
3. PHỐI GHÉP VI XỬ LÍ VỚI THIẾT BỊ VÀO RA

3.1 Giới thiệu về thiết bị vào/ra

Đối với 8086 (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

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.



Hình 4-14. Không gian nhớ của thiết bị vào/ra và bộ nhớ chính

b) Thiết bị vào/ra và bộ nhớ có chung không gian địa chỉ

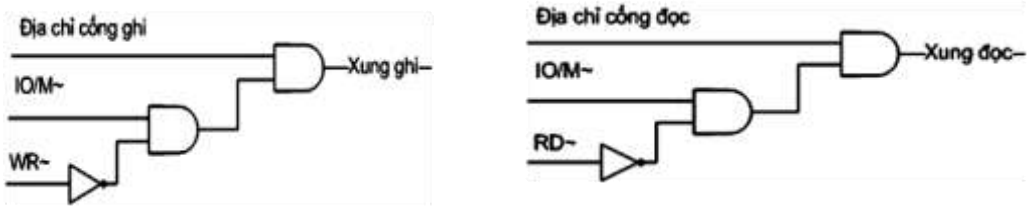
Trong cách phối ghép này, bộ nhớ và thiết bị ngoại vi cùng chia nhau không gian địa chỉ 1MB mà CPU 8086 có khả năng địa chỉ hóa. Các thiết bị ngoại vi sẽ chiếm một vùng nào đó trong không gian 1MB, phần còn lại là của bộ nhớ. Tất nhiên trong trường hợp này ta dùng chung tín hiệu IO/$\overline{M}$ =0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

3.2 Giải mã địa chỉ thiết bị vào ra

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. Thông thường các cổng có địa chỉ 8 bít tại A0-A7, trong một số hệ vi xử lý khác 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.

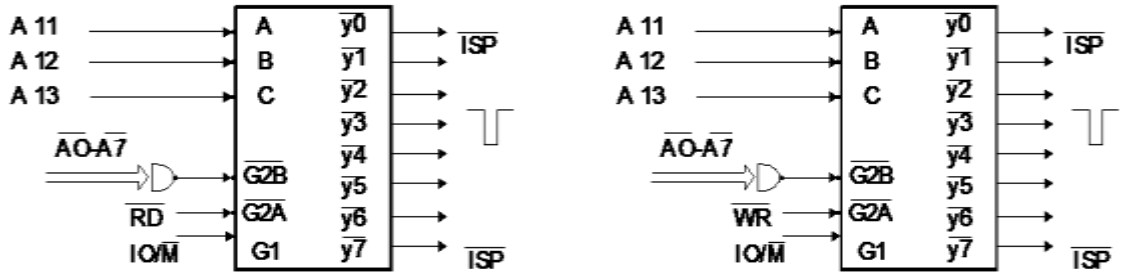
Các mạch giải mã đơn giản có thể tạo được từ mạch lô-gíc đơn giản như sau:



Hình 4-15. 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.

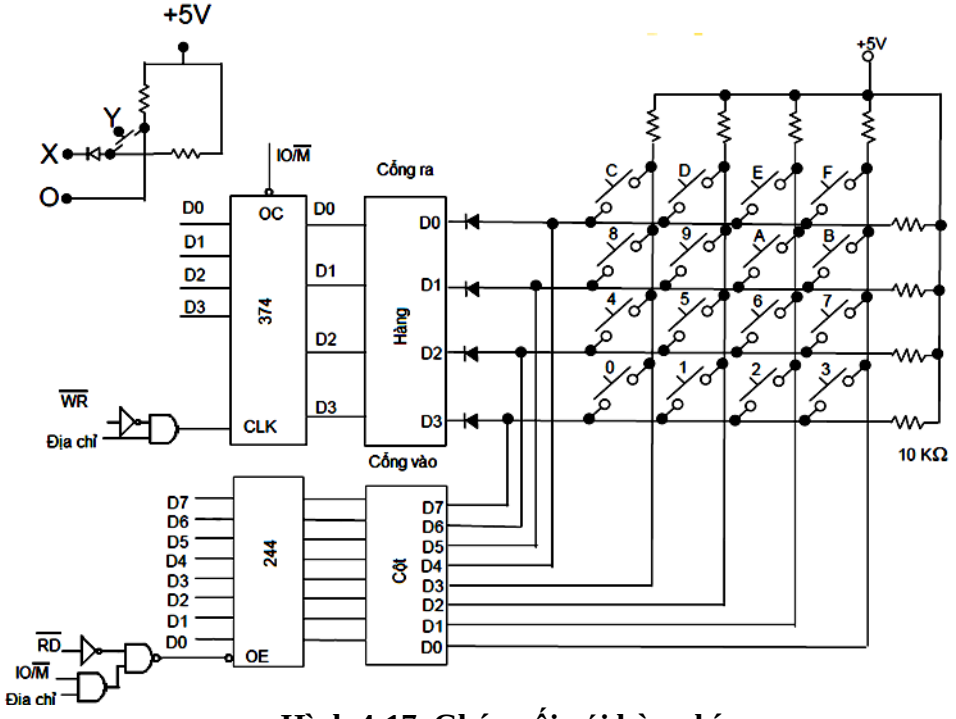
Trên cơ sở mạch này ta cũng có thể phối hợp với cả hai tín hiệu đọc và ghi để tạo ra tín hiệu chọn cho việc đọc/ghi từng cổng vào/ra ra cụ thể.



Hình 4-16. Giải mã địa chỉ cổng dùng 74LS138

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 4-17. Ghép nối với bàn phím

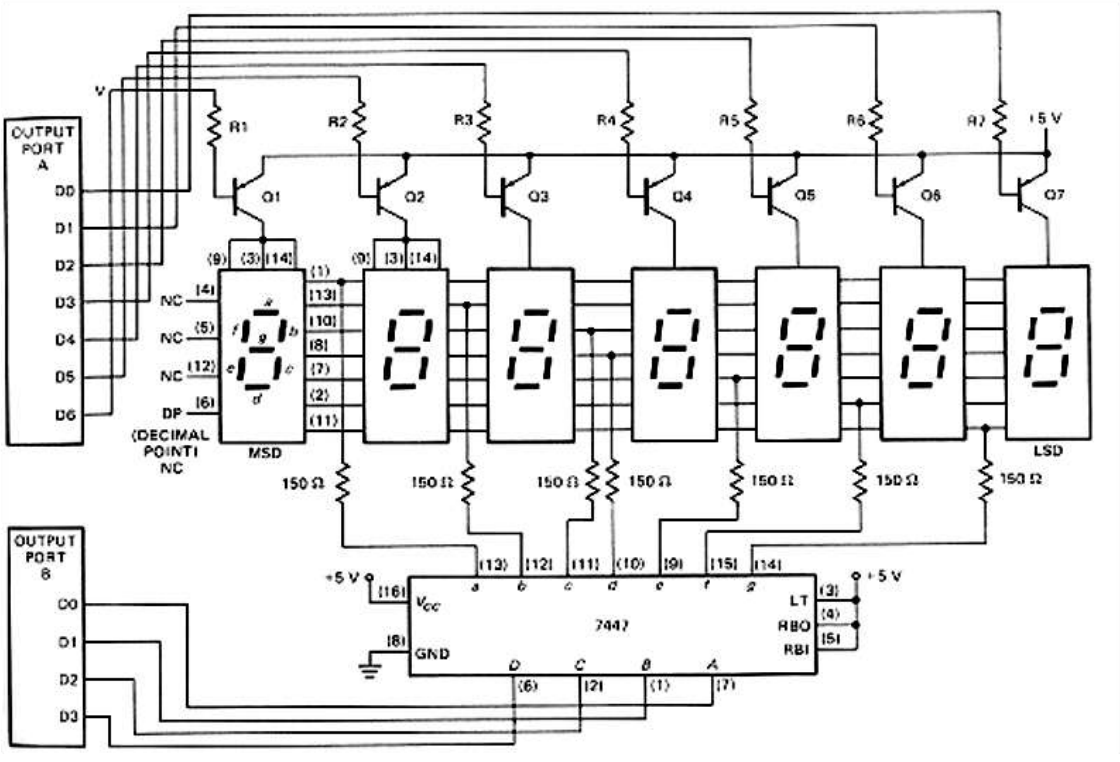
Hình 4-17 biểu diễn ghép nối giữa 8086 với bàn phím 16 số dạng tiếp điểm. Vi mạch 74LS374 được dùng để điều khiển các tín hiệu hàng và 74LS244 dùng để điều khiển các tín hiệu cột của bàn phím. Nguyên tắc hoạt động của một phím như sau. Nếu tín hiệu X ở mức cao (lô-gíc 1) thì đi-ốt sẽ khóa lại, vậy nên 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 4-18 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,64           ; Trễ bằng 64 lệnh NOP
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
```



Hình 4-18. Ghép nối hiển thị số

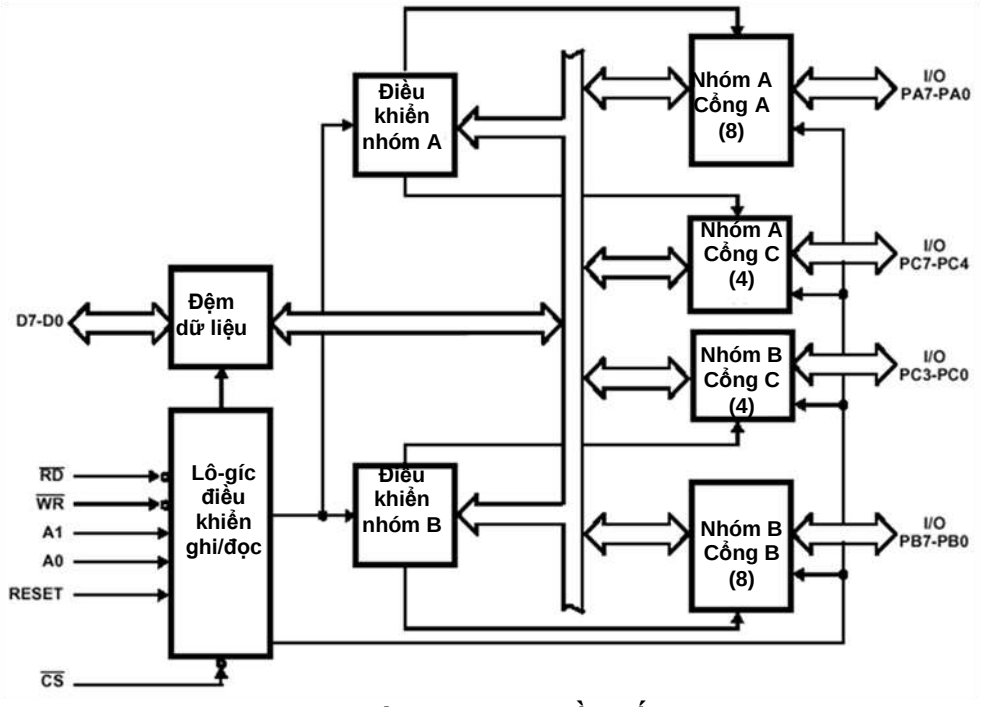
4. GIỚI THIỆU MỘT SỐ VI MẠCH HỖ TRỢ VÀO RA

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.

4.1 Ghép nối song song dùng 8255A

4.1.1 Giới thiệu

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.



Hình 4-19. Sơ đồ khối 8255A

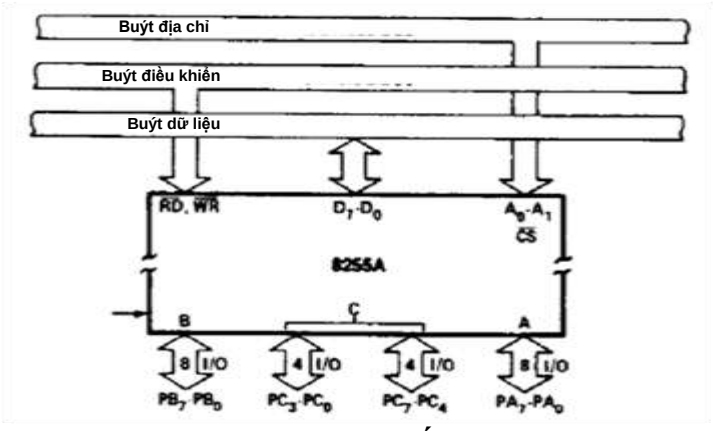
Các tín hiệu của 8255A có ý nghĩa như sau :

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.



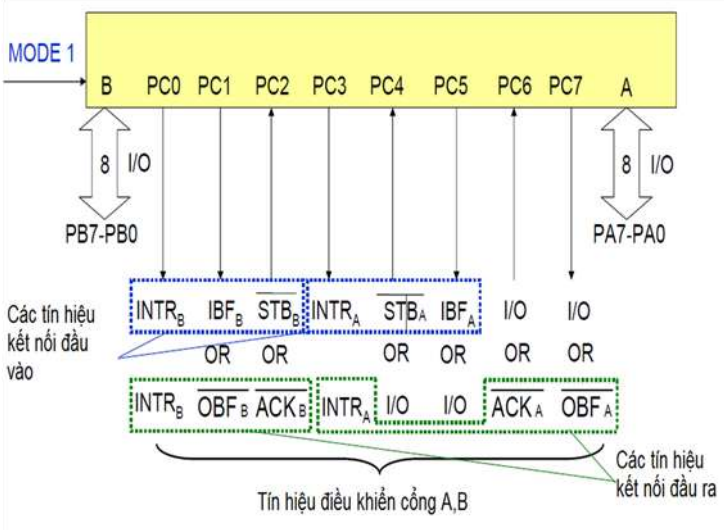
Hình 4-20. Các chế độ 0

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:

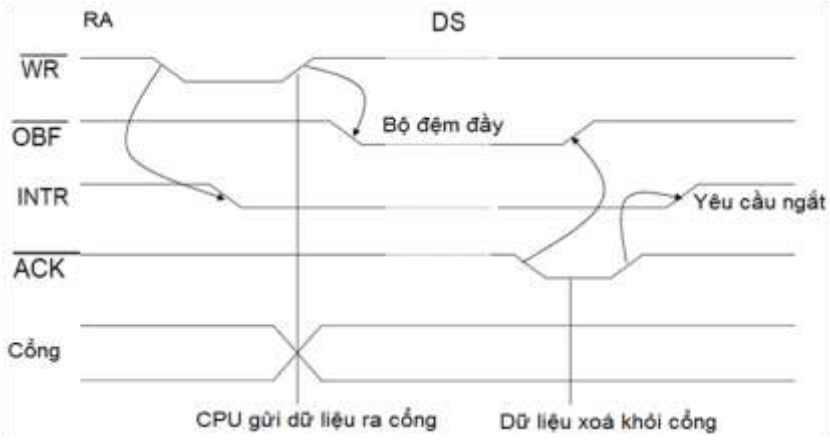
Đầu vào	**Đầu ra**
STB: Kiểm tra đầu vào (mức thấp)	OBF: Dữ liệu ra sẵn sàng (mức thấp)
IBF: Dữ liệu sẵn sàng (mức cao)	ACK: Nhận xong dữ liệu (mức thấp)
INTR: Báo ngắt CPU (mức cao)	INTR: Báo ngắt CPU (mức cao)

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



Hình 4-21. 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

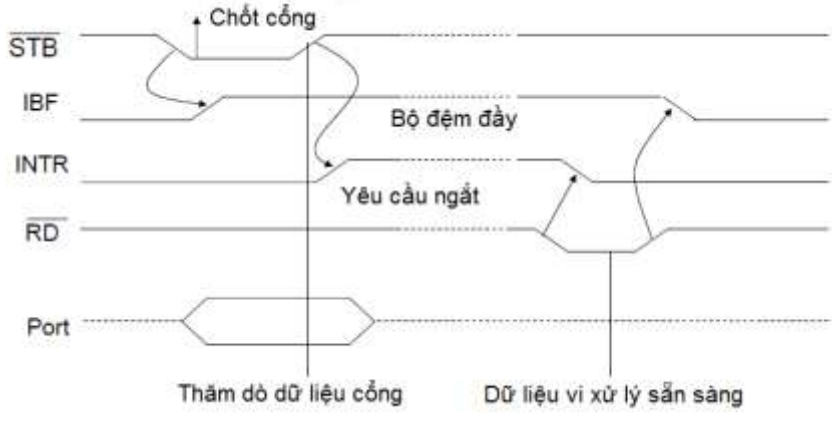


Hình 4-22. Biểu đồ thời gian tín hiệu ra

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 ảnh yêu cầu ngắt của PA tới CPU.
- 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

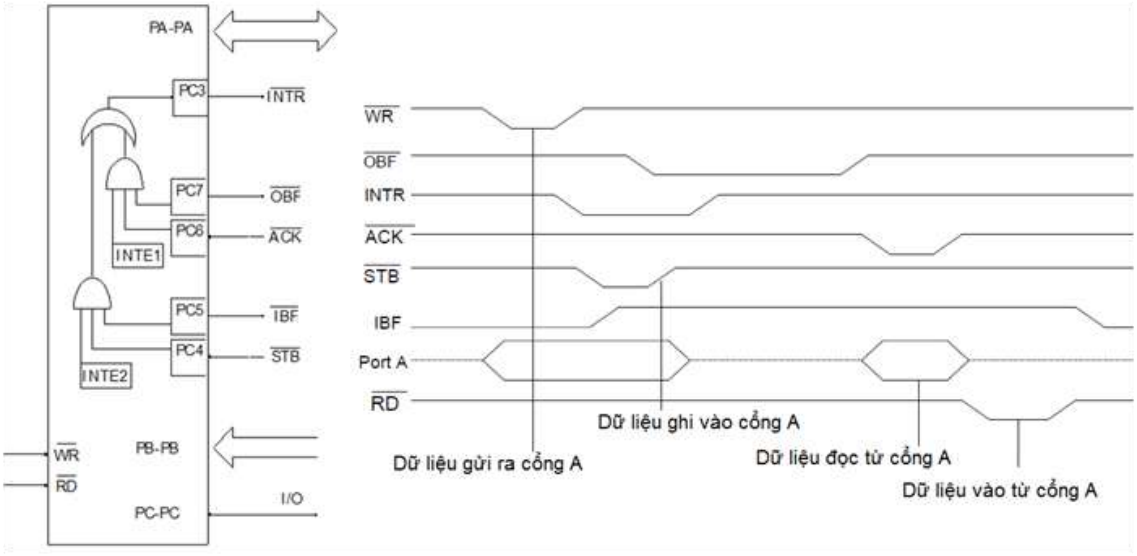


Hình 4-23. Biểu đồ thời gian tín hiệu vào

- $\overline{STB}$ (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 4-24. Các tín hiệu kết nối biến đổi tuỳ thuộc theo dữ liệu được gửi ra hay đọc về từ cổng A.



Hình 4-24. Các tín hiệu kết nối hai chiều và biểu đồ thời gian

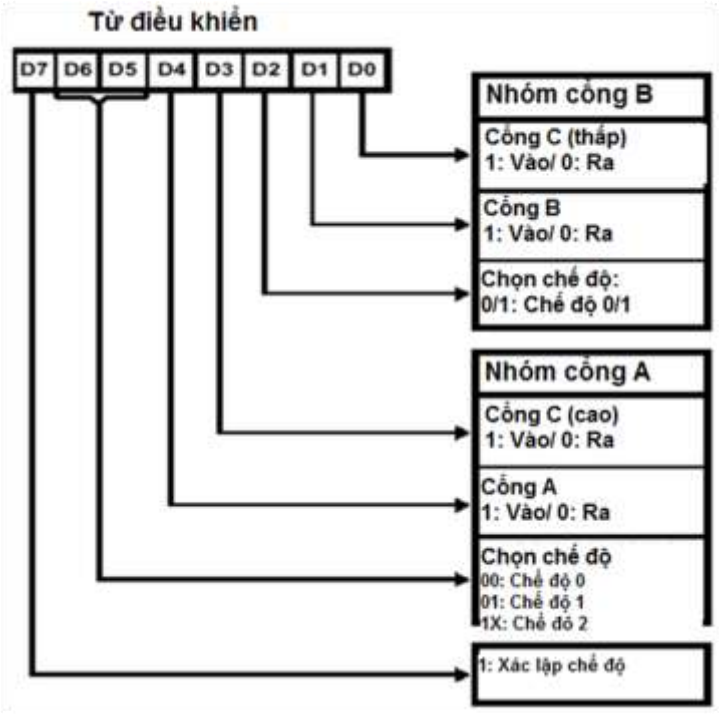
4.1.2 Lập trình 8255A

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

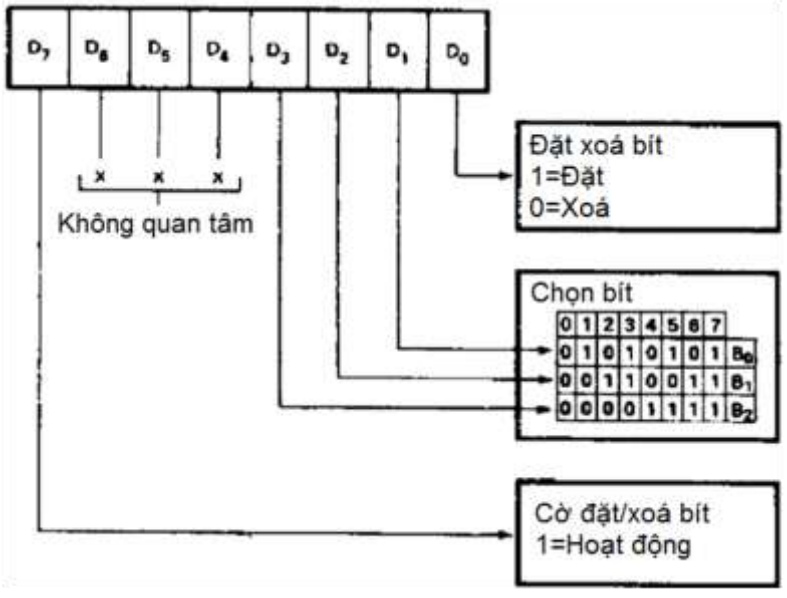
A1	A0	Thanh ghi
x	x	Không sử dụng
0	0	Cổng A (PA)
0	1	Cổng B (PB)
1	0	Cổng C (PC)
1	1	Điều khiển

Ý nghĩa các bít của thanh ghi điều khiển chế độ hoạt động như trong Hình 4-25. 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 4-25. Thanh ghi điều khiển chế độ

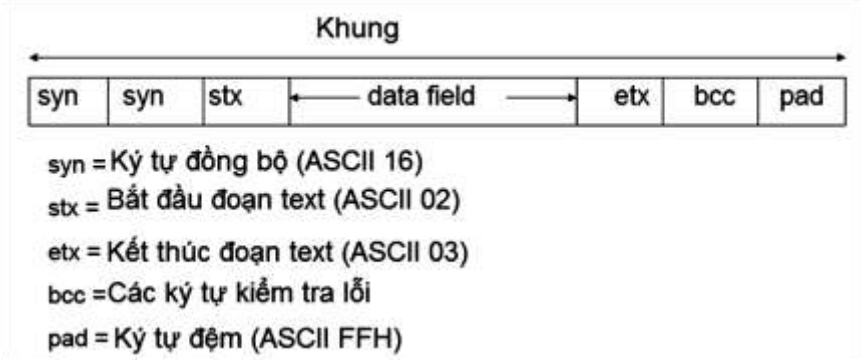


Hình 4-26. Đặt xoá các tín hiệu điều khiển trên cổng C

4.2 Truyền thông nối tiếp

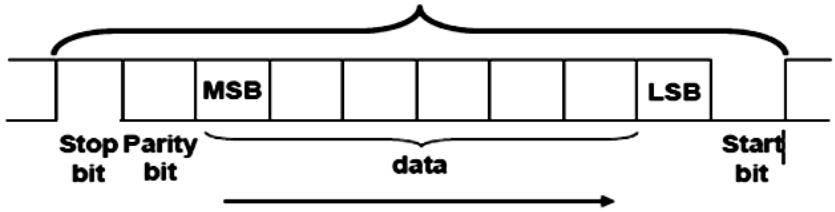
Việc truyền thông tin giữa các bộ phận nằm gần nhau trong hệ thống vi xử lý có thể thực hiện thông qua buýt song song mở rộng hoặc qua các mạch phối ghép song song trong đó các byte hoặc được truyền đi trên một tập các đường dẫn bằng mạch in hoặc dây cáp trong trường hợp cần phải truyền thông tin giữa 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 4-27 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 4-27. 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 4-28. 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, 9600. . .

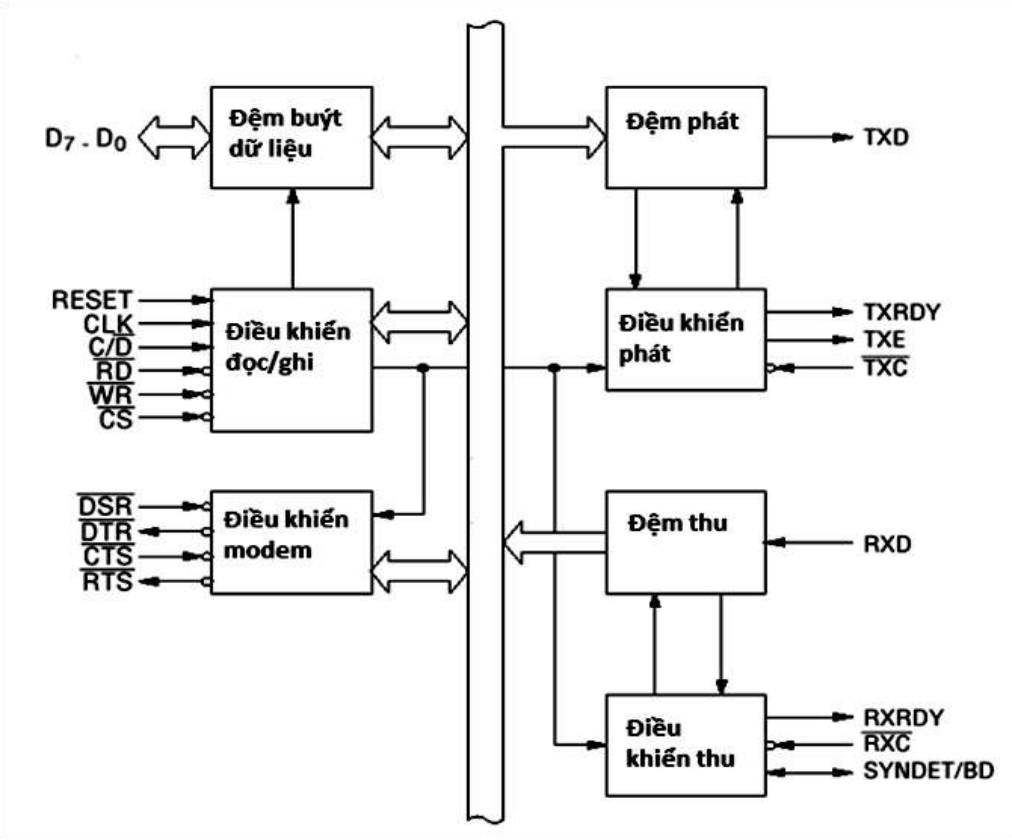
Để tạo điều kiện dễ dàng cho việc phối ghép đường truyền nối tiếp với hệ vi xử lý và để giảm tối đa các mạch phụ thêm ở ngoài. Người ta đã chế tạo ra các vi mạch tổ hợp cỡ lớn lập trình có khả năng hoàn thành các công việc cần thiết trong khi phối ghép đó là các mạch thu phát dị bộ vạn năng (*Universal Asynchronous Receiver - Transmitter UART*) và mạch thu phát đồng bộ - dị bộ vạn năng (*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 một cặp USART ở đầu phát và ở đầu thu.

4.2.1 Mạch USART 8251A

4.2.1.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 4-29. Sơ đồ khối 8251A

Các tín hiệu của mạch 8251A hầu hết là giống tín hiệu của 8086. 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.

+ 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.

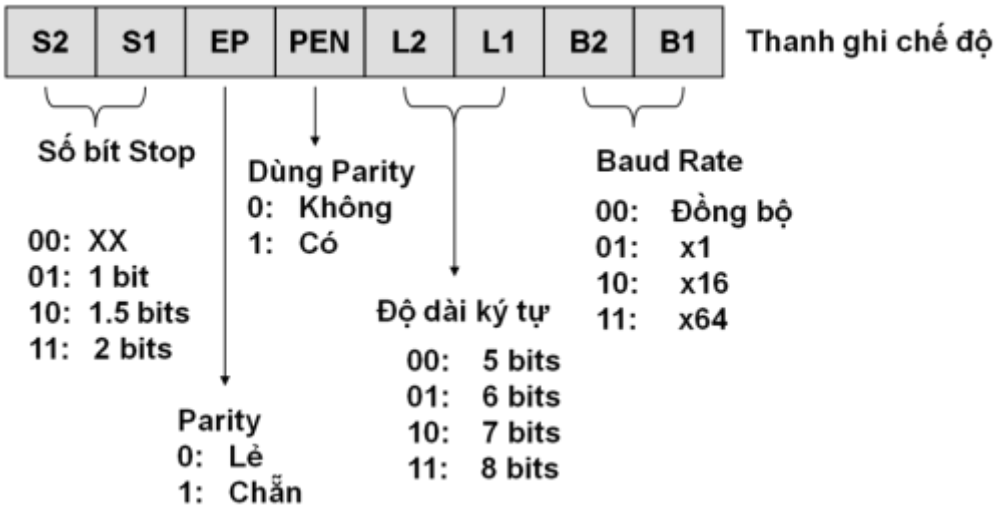
4.2.1.b Các thanh ghi bên trong của 8251A

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 4-7).

Bảng 4-7. 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:



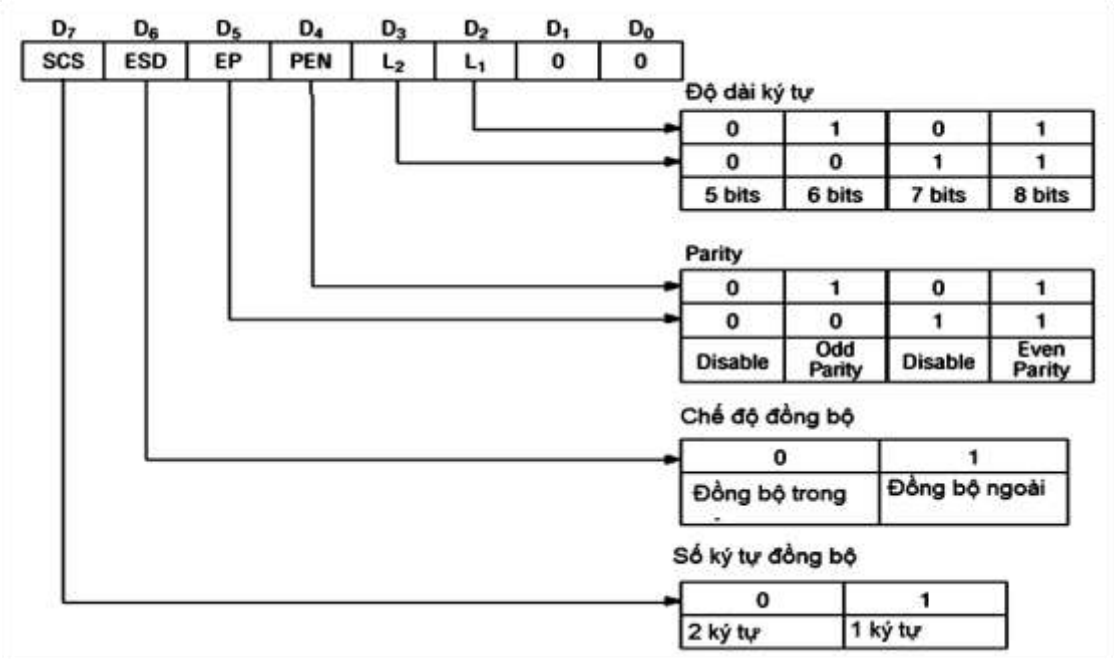
Hình 4-30. Thanh ghi chế độ (dị bộ)

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 4-31. Thanh ghi chế độ (đồng bộ)

Thanh ghi lệnh:

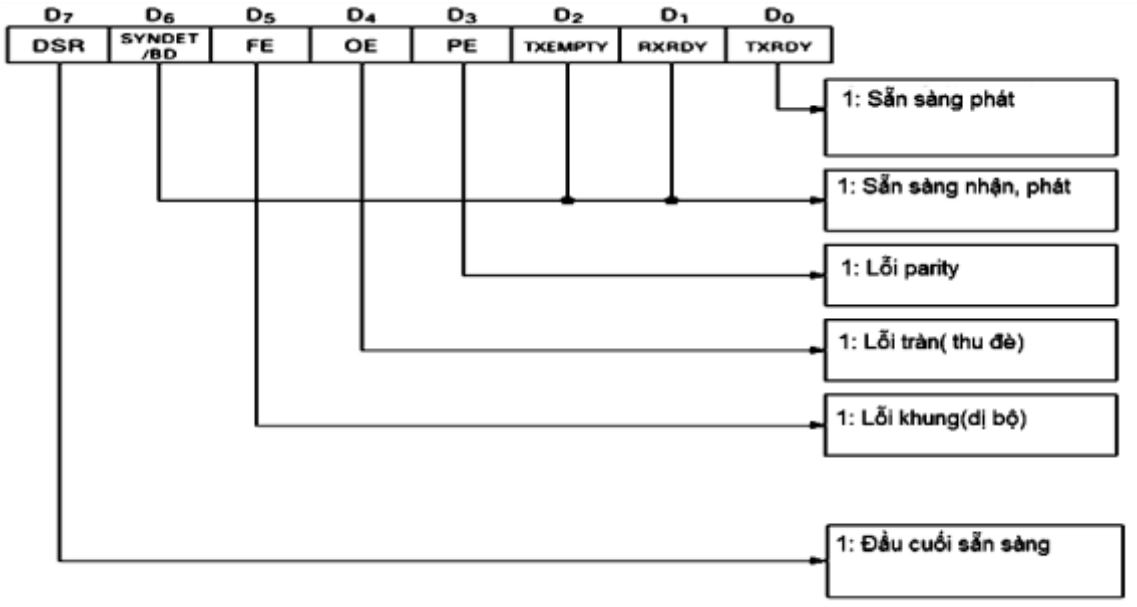
Cấu trúc thanh ghi lệnh như sau:

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 4-32. Cấu trúc thanh ghi lệnh

Thanh ghi trạng thái

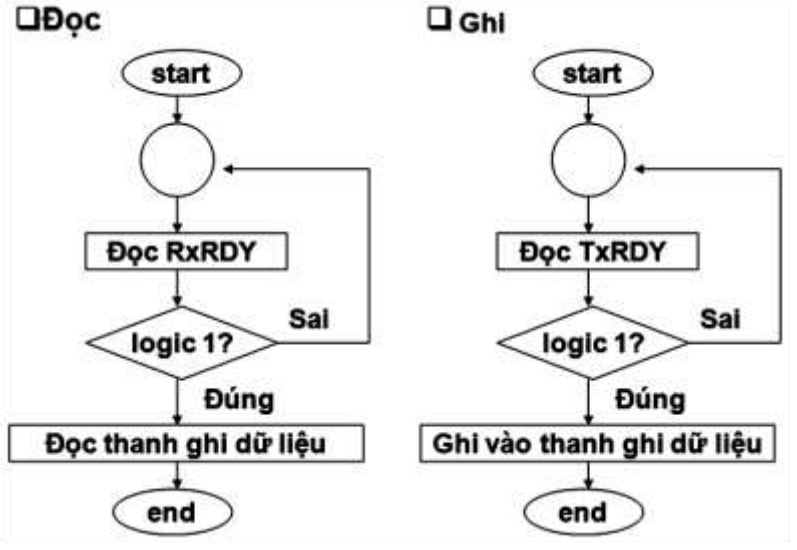
Giá trị trên các bít thanh ghi này cho ta biết tình trạng hoạt động của 8251A



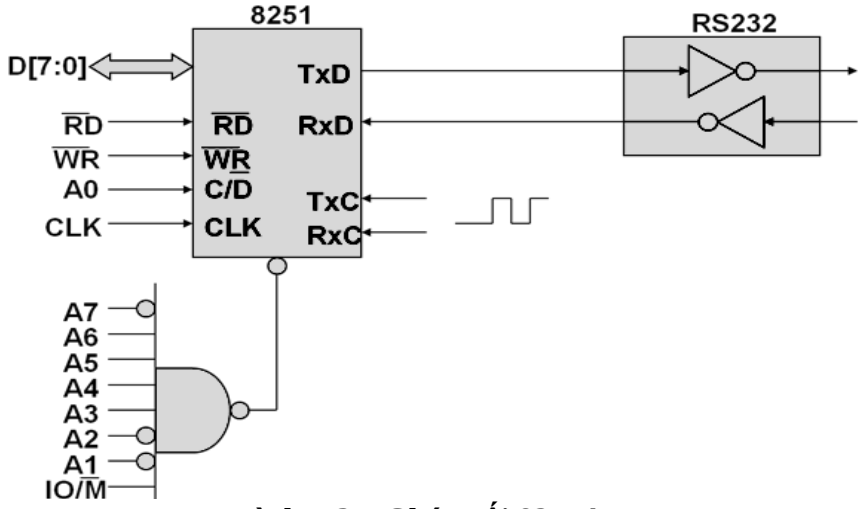
Hình 4-33. Cấu trúc thanh ghi trạng thái

4.2.1.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 4-34. Lưu đồ đọc/ghi đơn giản



Hình 4-35. Ghép nối 8251A

Hình 4-35 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 5. TỔNG QUAN VỀ CÁC PHƯƠNG PHÁP VÀO RA DỮ LIỆU

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 các 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 trong 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.

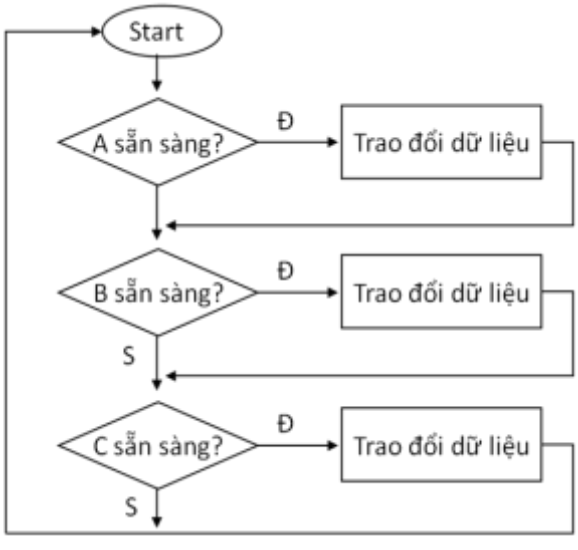
Trong máy tính sử dụng hệ điều hành, người dùng thường làm việc với thiết bị vào/ra ảo. Người dùng không phải quan tâm tới các đặc tính của thiết bị. Thay vào đó, người dùng thực hiện trao đổi dữ liệu thông qua các dịch vụ vảo/ra do hệ điều hành cung

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.

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 5-1. Vào/ra lập trình với nhiều thiết bị

Các mạch kết nối trong Hình 4-17 và Hình 4-18 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 5-1. 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.

3. VÀO/RA BẰNG NGẮT

3.1 Giới thiệu

Nhược điểm của vào/ra thăm dò là máy tính cần kiểm tra bít trạng thái bằng cách chờ. 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.

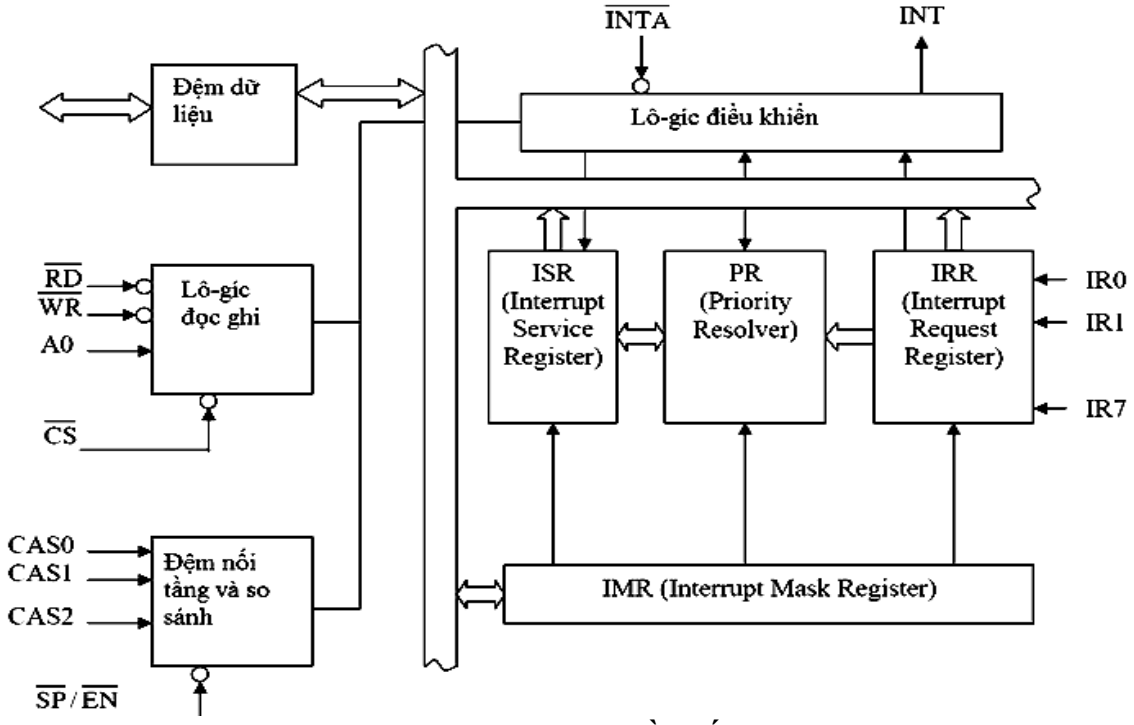
3.2 Bộ xử lý ngắt ưu tiên 8259

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. 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.

3.2.1 Các khối chức năng chính của 8259A

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



Hình 5-2. 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 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.

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. Ngoài ra, còn có một số tín hiệu đặc biệt khác của 8259A gồm:

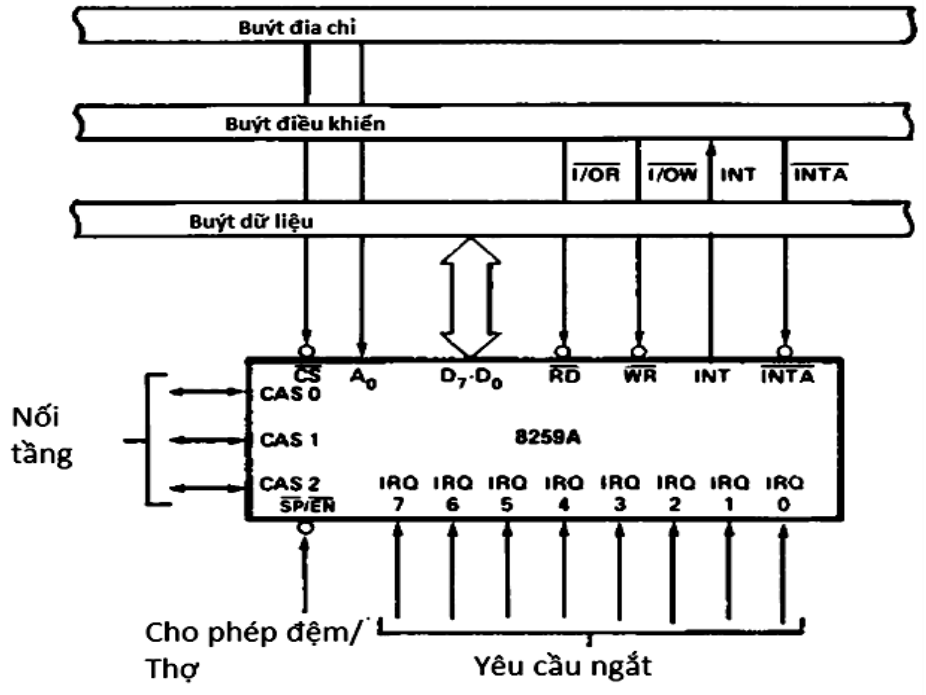
+ CAS_0-CAS_2 [I, O]: là các đầu vào đối với các mạch 8259A thợ hoặc các đầu ra của mạch 8259A chủ dùng khi cần nối tầng để tăng thêm các yêu cầu ngắt cần xử lý.

+ $\overline{SP}/\overline{EN}$ [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 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.

+ $\overline{INTA}$ [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. Nếu hệ vi xử lý 8086 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 5-3. Ghép nối 8259 với buýt 8086/8088

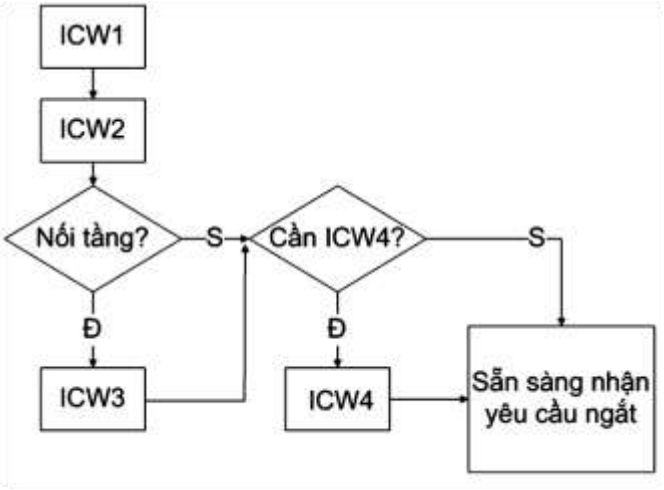
3.2.3 Lập trình cho PIC 8259A

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.

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. Hình dưới đây thể hiện thứ tự ghi và điều kiện để ghi các điều khiển ICW vào 8259A.

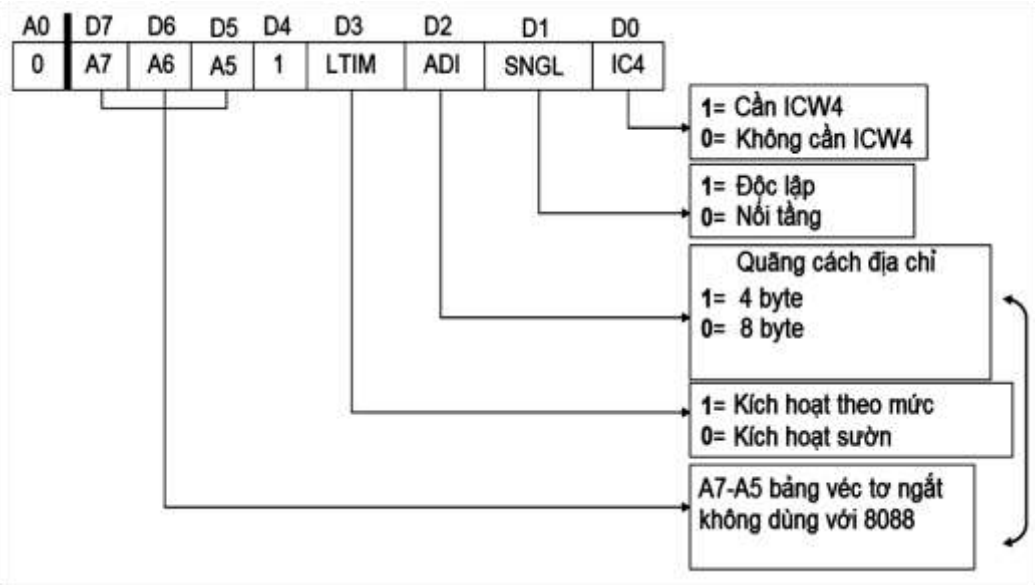


Hình 5-4. Trình tự sử dụng các thanh ghi khởi đầu

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**

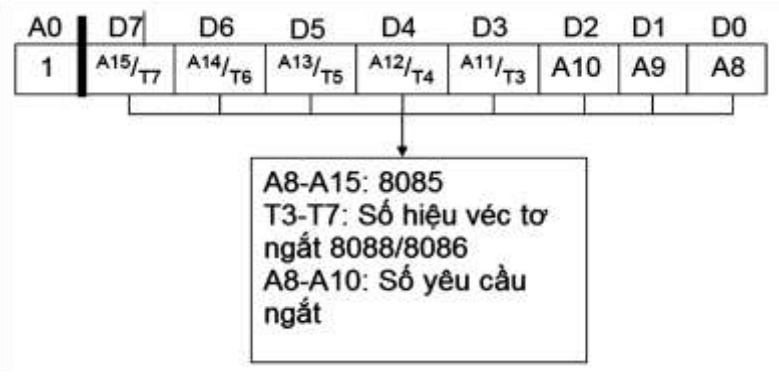
Bít D_0 của ICW1 quyết định 8259A sẽ được nối với họ vi xử lý nào. Để làm việc với hệ 16-32bít (họ x86) thì ICW nhất thiết phải có ICW4 = 0 (và như vậy các bít của ICW4 sẽ bị xóa về 0). Các bít còn lại của ICW1 định nghĩa cách thức tác động của xung yêu cầu ngắt (tác động theo sườn hay theo mức) tại 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 5-5. 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. . .



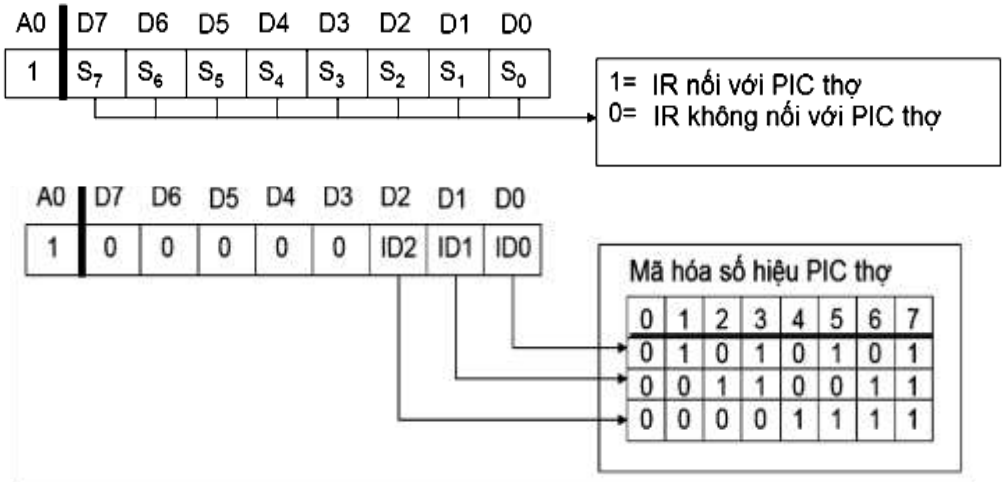
Hình 5-6. Dạng thức của ICW2

- **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.

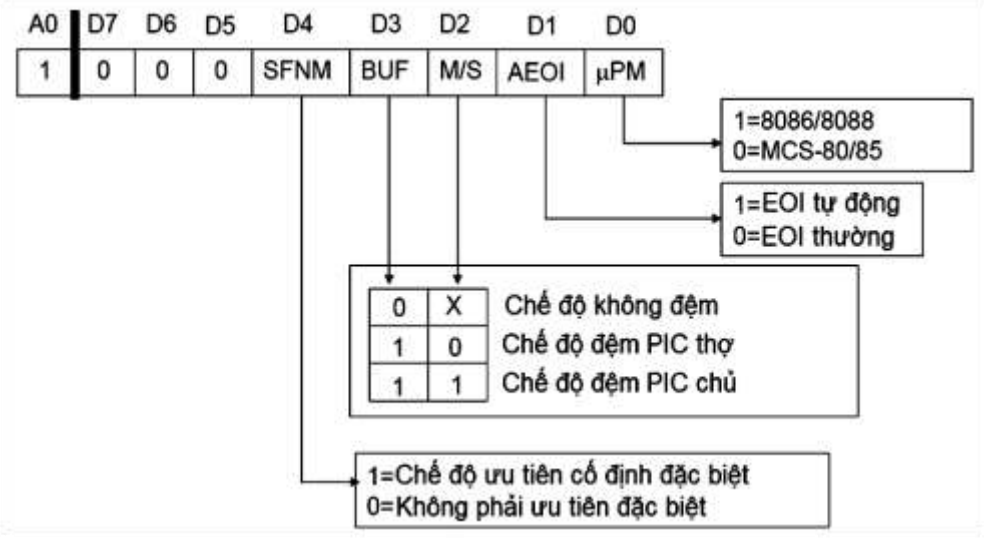
ICW3 cho mạch thợ: dùng làm phương tiện để các mạch này được nhận biết. Vì vậy từ điều khiển khởi đâùu này phải chứa mã số i ứng với đầu vào Iri của mạch chủ mà mạch thợ đã cho nối vào. Mạch thợ sẽ so sánh mã số này với mã số nhận được ở CAS_2-CAS_0. Nếu bằng nhau thì số hiệu ngắt sẽ được đưa ra buýt khi có INTA.



Hình 5-7. Dạng thức của ICW3 cho mạch chủ và thợ

- **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.



Hình 5-8. Dạng thức của ICW4

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 vuới 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 tất cả các yêu cầu khác với mức ưu tiên cao hơn có thể ngắt yêu cầu khác với mức ưu tiên thấp hơn.

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 với buýt hệ thống.

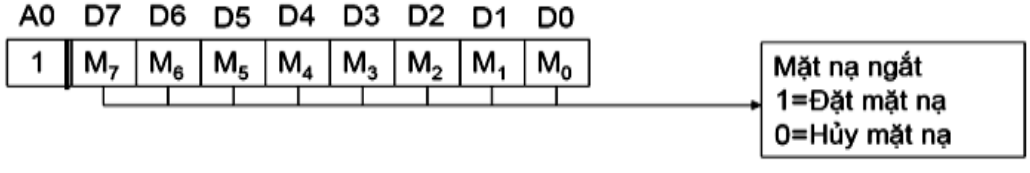
Bít AEOI = 1 cho phép chọn cách kết thúc yêu cầu ngắt tự động. Khi AEOI = 1 thì 8259A tự động xóa ISRi = 0 khi xung INTA cuối cùng chuyển lên mức cao mà không làm thay đổi thứ tự ưu tiên. Ngược lại. Khi ta chọn cách kết thúc yêu cầu ngắt thường (AEOI = 0) thì chương trình phục vụ ngắt phải có thêm lệnh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

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 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. Tình trạng mặt nạ ngắt hiện tại có thể được xác định bằng cách đọc IMR (xem trong thời điểm hiện tại yêu cầu ngắt nào bị che)



Hình 5-9. OCW1 Trạng thái yêu cầu ngắt

o OCW2

Các bít R. SL và EOI phối hợp với nhau cho phép chọn ra các cách thức kết thúc yêu cầu ngắt khác nhau. Một vài cách thức 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. Dưới đây là các chế độ làm việc của 8259A.

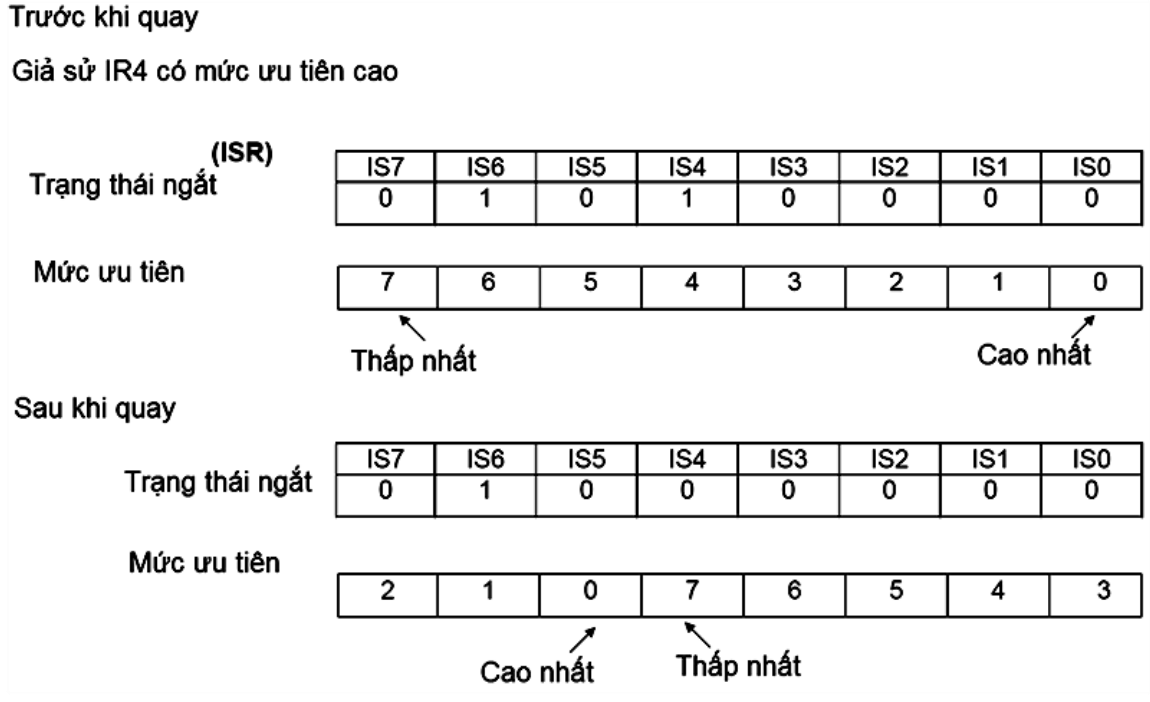
- Chế độ ưu tiên cố định:

  Đây là chế độ làm việc ngầm định của 8259A sau khi được nạp các từ điều khiển khởi đầu. Trong chế độ này, các đầu vào IR7-IR0 được gán cho các mức ưu tiên cố định: IR0 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 các yêu cầu khác với mức ưu tiên thấp hơn đều bị cấm.

- 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.



Hình 5-10. Trạng thái ngắt và chế độ quay mức ưu tiên

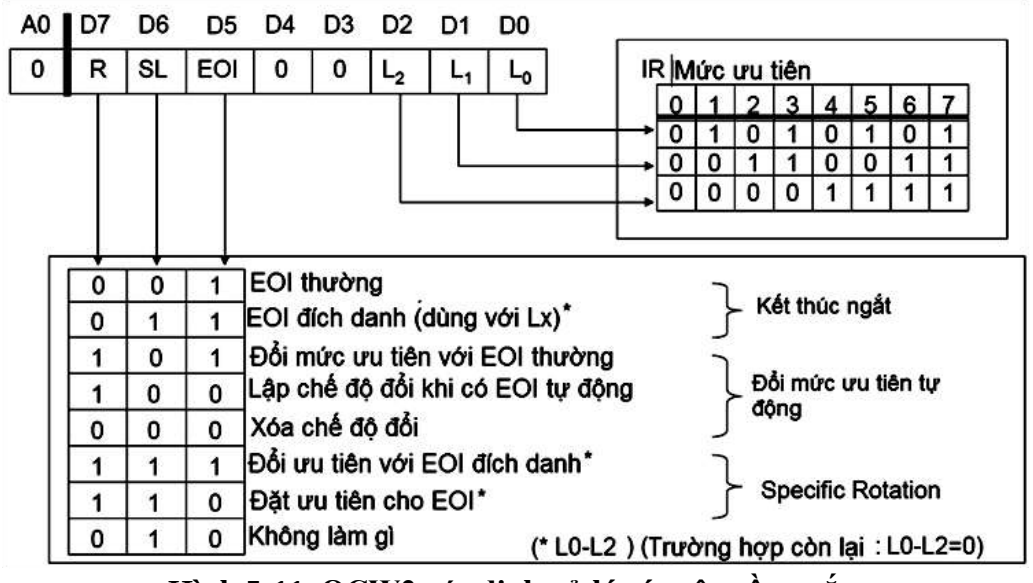
- Chế độ quay (đổi) mức ưu tiên chỉ đích danh:

Ở chế độ này 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.

Trở lại các vấn đề liên quan đến OCW, việc phối hợp các bít R, SL và EOI phối với nhau 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 như sau:

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 5-10.



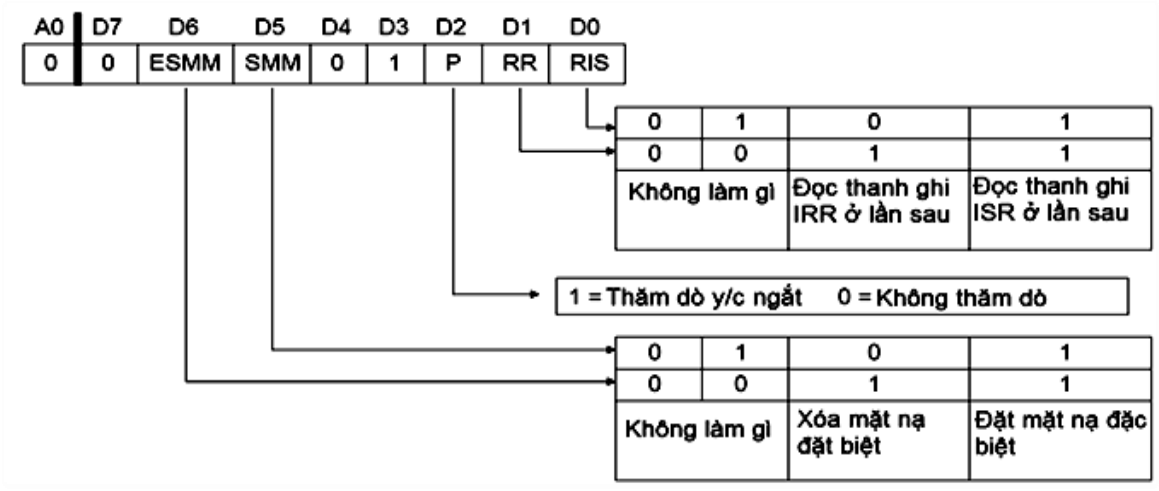
Hình 5-11. OCW2 xác định xử lý các yêu cầu ngắt

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 5-12. 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 5-13. 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 5-14. 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)

3.2.3.c Hoạt động của 8086 với 8259A

Cuối cùng để có cái nhìn một cách có hệ thống về hoạt động của hệ vi xử lý với CPU 8086 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.
2. 8086 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 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 dùng số hiệu ngắt để tính ra địa chỉ ngắt của vectơ ngắt tương ứng.
7. 8086 cất FR, xóa các cờ IF và TF và cất địa chỉ trở về CS:IP vào ngăn xếp.
8. 8086 lấy địa chỉ CS:IP của chương trình phục vụ ngắt từ bảng vectơ ngắt và thực hiện chương trình đó.

4. VÀO/RA BẰNG TRUY NHẬP TRỰC TIẾP BỘ NHỚ

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.

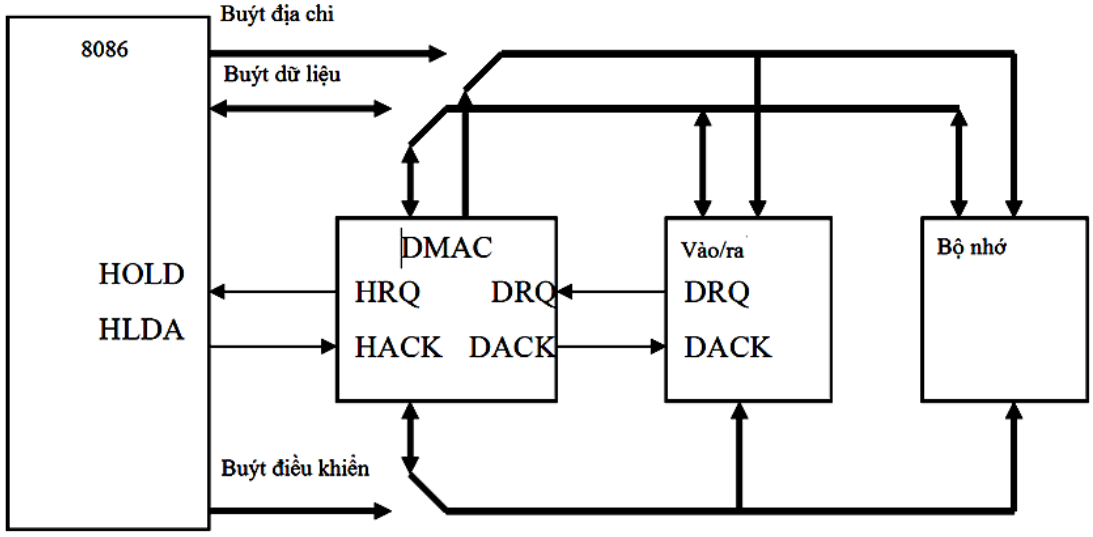
Trong thực tế có những khi ta cần trao đổi dữ liệu thật nhanh với thiết bị ngoại vi: như khi cần đưa dữ liệu hiện thị ra màn hình hoặc trao đổi dữ liệu với bộ điều khiển đĩa. Trong các trường hợp đó ta cần có khả năng ghi /đọc dữ liệu trực tiếp với bộ nhớ thì mới đáp ứng được yêu cầu về tốc độ trao đổi 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 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 5-15. 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.

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.

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 $\overline{\text{MEMR}}$=O để đọc một byte từ bộ nhớ ra buýt dữ liệu. tiếp đó DMAC đưa ra tín hiệu $\overline{\text{IOW}}$ =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.

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.

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.

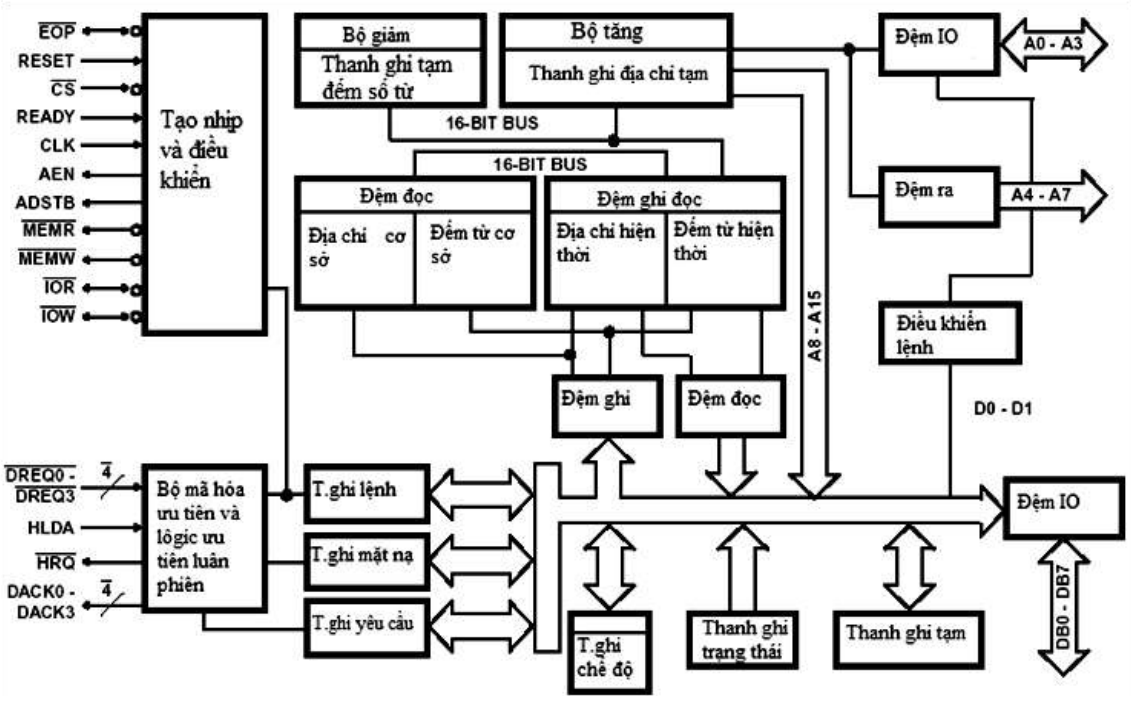
4.3 Bộ điều khiển truy nhập trực tiếp vào bộ nhớ Intel 8237A

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 5-16. Sơ đồ khối 8237A

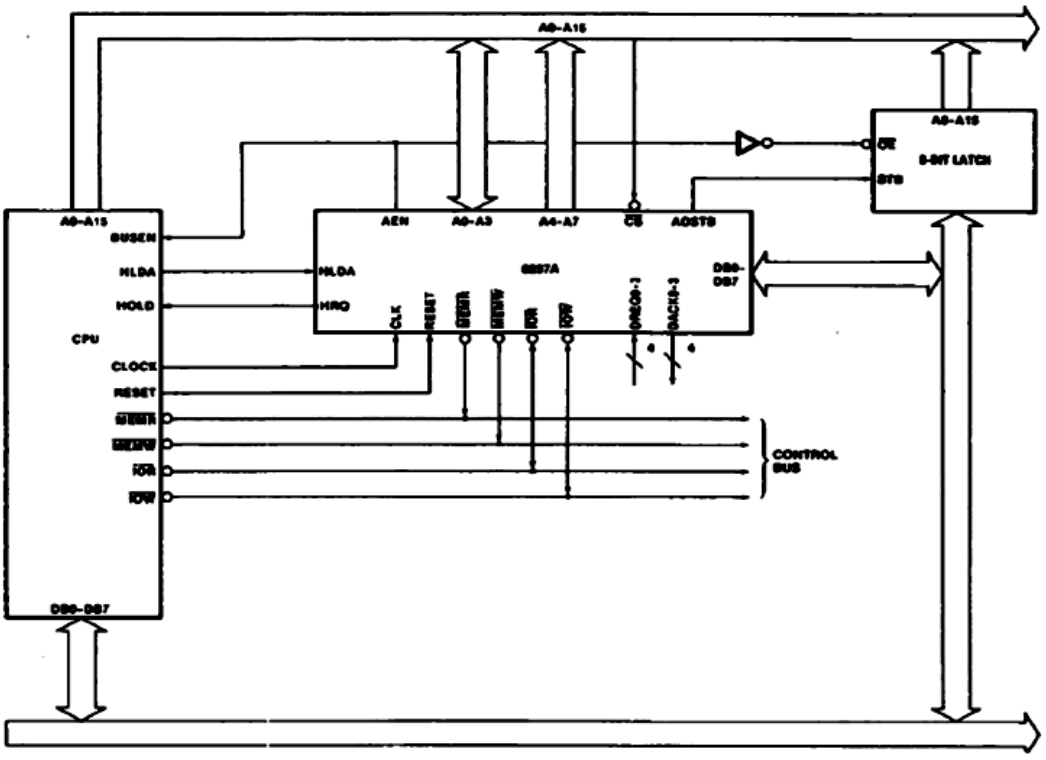
Mạch DMAC 8237A chứa 4 kênh trao đổi dữ liệu DMA với mức ưu tiên lập trình được. DMAC 8237A có tốc độ truyền 1 MB/s cho mỗi kênh, một kênh có thể truyền môt mảng có độ dài 64KB.

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 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.
- $DACK_0$-$DACK_3$[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 5-17. Ghép nối 8237 với buýt hệ vi xử lý

4.3.3 Các thanh ghi bên trong của DMAC 8237A

Các thanh ghi bên trong DMAC 8237A được CPU 8086 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 5-1. 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 5-2. 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 5-3. 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

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.

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.

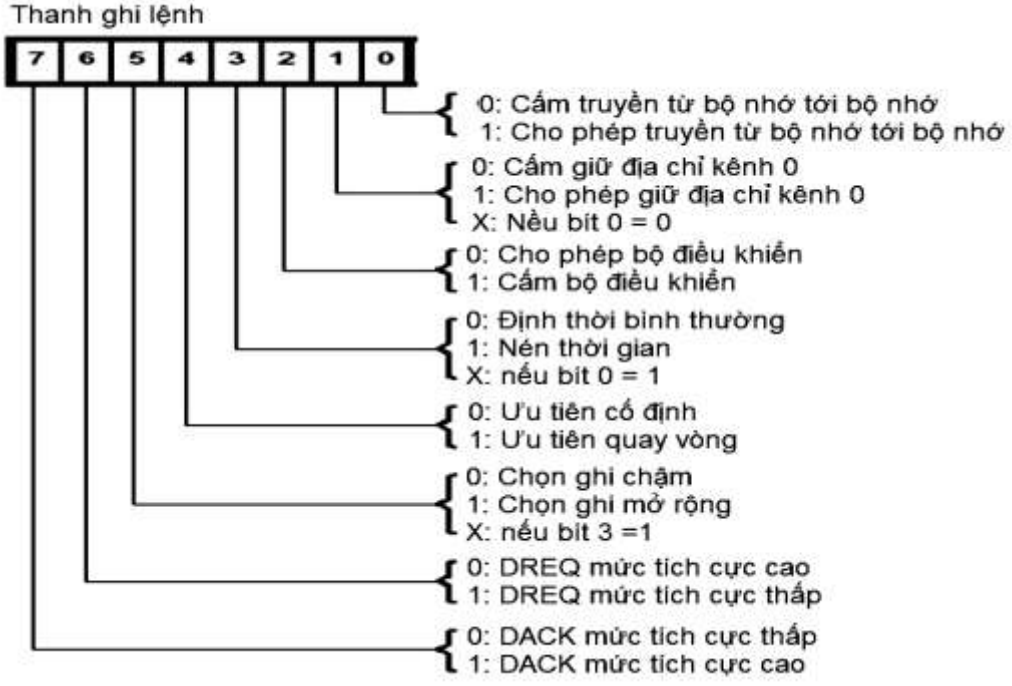
4.3.3.c Thanh ghi địa chỉ cơ sở và thanh ghi số đếm cơ sở:

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.

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 để 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.

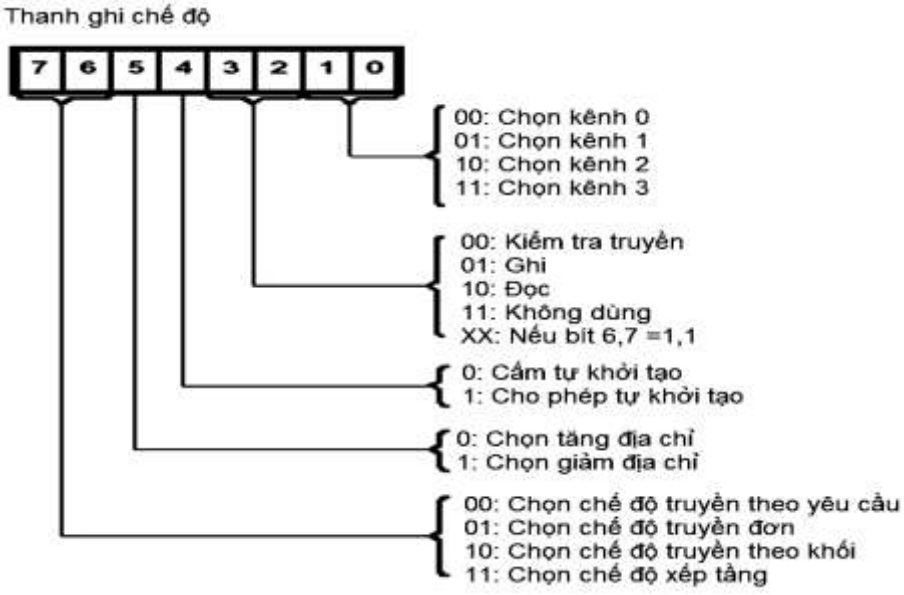
Bít D4 cho phép chọn chế độ ưu tiên cố định (kênh 0 có mức ưu tiên cao nhất. Kênh 3 có mức ưu tiên thấp nhất) hoặc chế độ ưu tiên luân phiên (kênh 0 lúc đầu có mức ưu tiên cao nhất. Sau khi kênh này được chọn để chuyển dữ liệu thì nó được nhận mức ưu tiên thấp nhất. Kênh 1 lại trở thành kênh có mức ưu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

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:



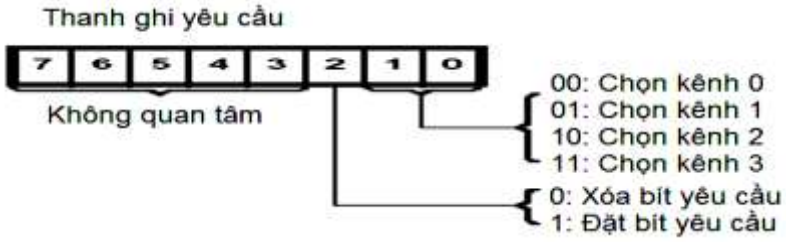
Trong chế độ DMA theo yêu cầu. DMAC tiến hành chuyển dữ liệu cho đến khi có tín hiệu EOP từ bên ngoài hoặc cho đến khi không còn yêu cầu DMA nữa (DREQ trở nên không tích cực)

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.

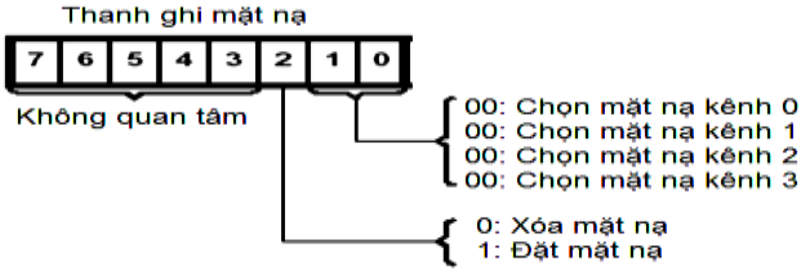
4.3.3.f Thanh ghi yêu cầu:

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:



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.



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.

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 — D0=1: Kênh 0 đạt số đếm

D6=1: Kênh 1 có yêu cầu — D1=1: Kênh 1 đạt số đếm

D5=1: Kênh 2 có yêu cầu — D2=1: Kênh 2 đạt số đếm

D4=1: Kênh 3 có yêu cầu — D3=1: Kênh 3 đạt số đếm

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.

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

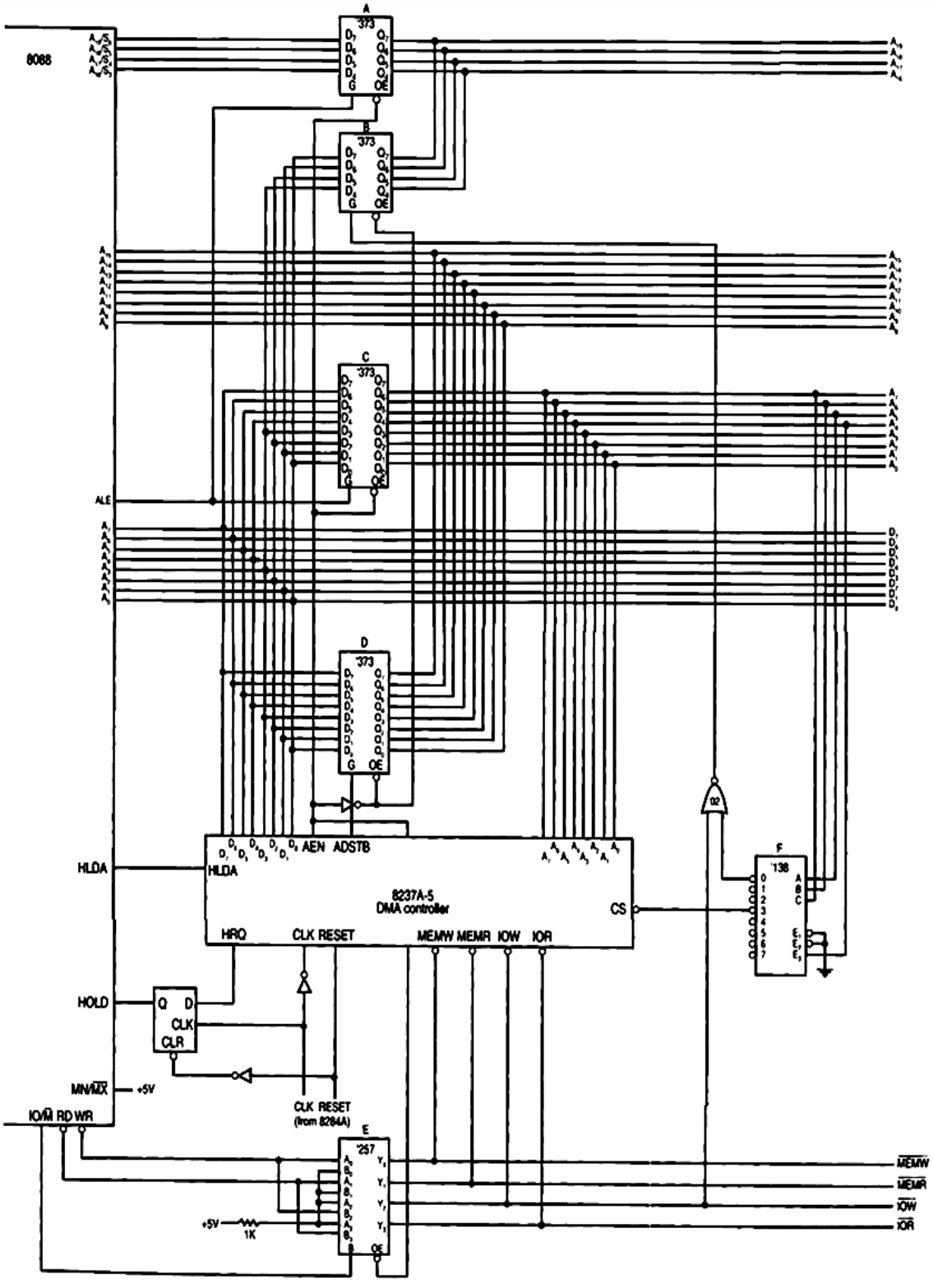
+ ghi LSB rồi MSB của thanh ghi địa chỉ

+ 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 như trong Hình 5-18.

```
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 5-18. Ghép nối 8237A với vi xử lý ở chế độ MIN

Chương 6. CÁC BỘ VI ĐIỀU KHIỂN

1. GIỚI THIỆU VỀ VI ĐIỀU KHIỂN VÀ CÁC HỆ NHÚNG

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.

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... 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 6-1. 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

2. HỌ VI ĐIỀU KHIỂN Intel 8051

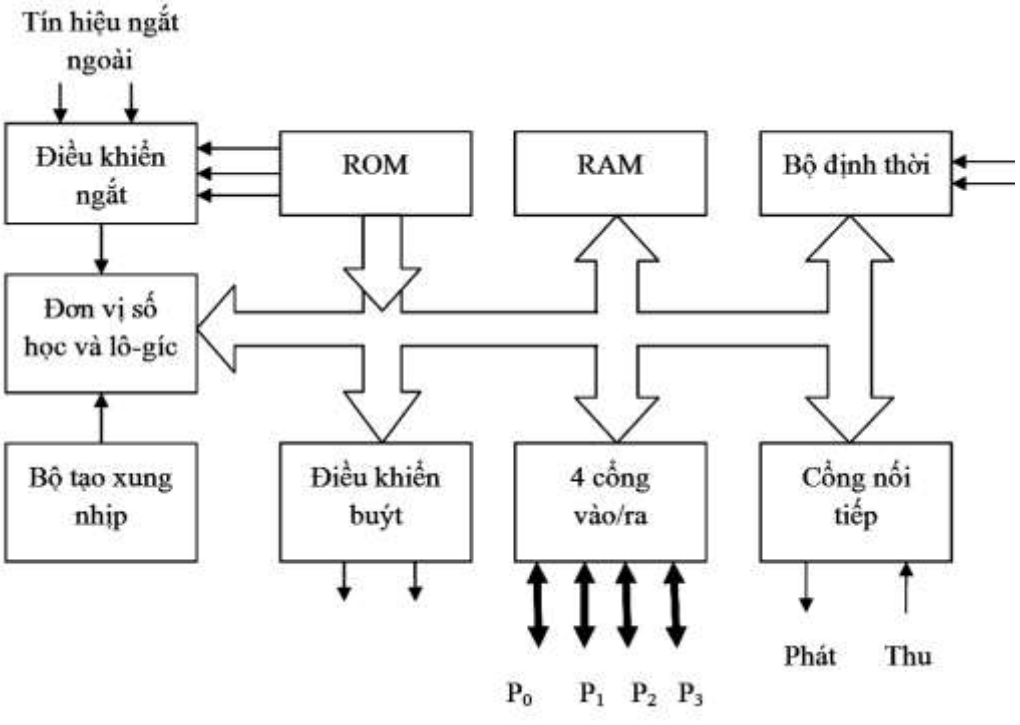
Vi điều khiển 8051 lần đầu tiên được Intel giới thiệu vào năm 1981. Đây là bộ vi điều khiển 8 bít với 128 byte RAM và 4KB ROM, một cổng nối tiếp và 4 cổng 8 bít trên một vi mạch đơn lẻ. Dòng vi điều khiển này trở nên phổ biến sau khi Intel cho phép các nhà sản xuất khác được chế tạo vi điều khiển tương 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 đáp ứng các yêu cầu ứng dụng khác nhau của các nhà phát triển.

Bảng 6-2. Thông số của một số vi điều khiển họ 8051

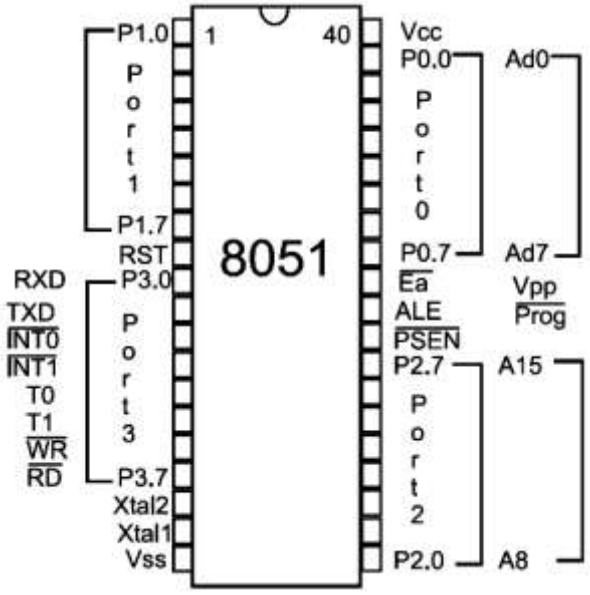
Tính năng	8051	8052	8031
ROM	4K	8K	-
RAM (Byte)	128	256	128
Bộ định thời	2	3	2
Tín hiệu vào/ra	32	32	32
Cổng nối tiếp	1	1	1
Nguồn ngắt	6	8	6

2.1 Sơ đồ khối



Hình 6-1. Sơ đồ khối 8051

Hình 6-1 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 6-2 cho biết sơ đồ tín hiệu của 8051, ý nghĩa các tín hiệu được giải thích trong Bảng 6-3.



Hình 6-2. 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 6-3. 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.

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 tích lũy (*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.

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 6-1

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
```

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.

3.1 Chuyển đổi số tương tự (D/A)

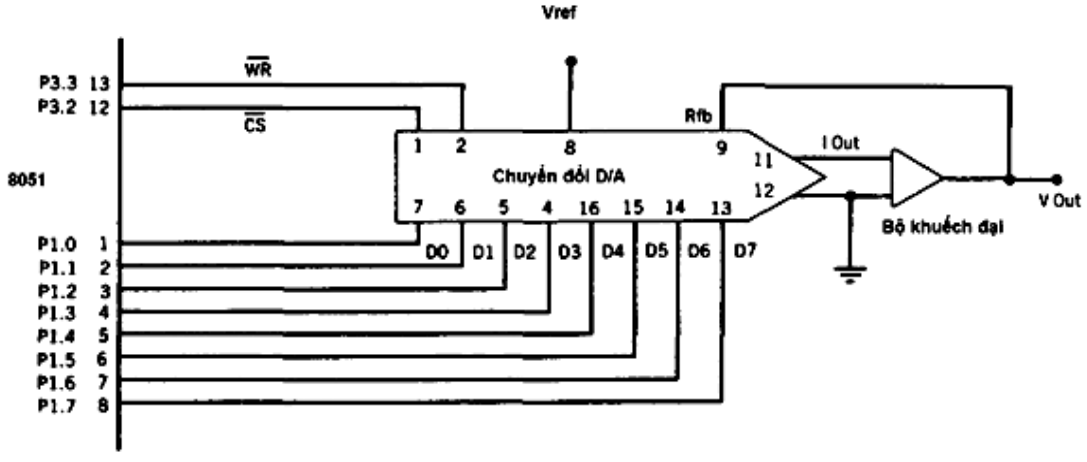
Hình 6-3 giới thiệu kết nối giữa vi điều khiển 8051 và bộ chuyển đổi D/A khái quát. Bộ chuyển đổi D/A có đặc điểm sau:

$V_{out} = -V_{ref} \times$(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 6-3. 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 6-2. 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)
```

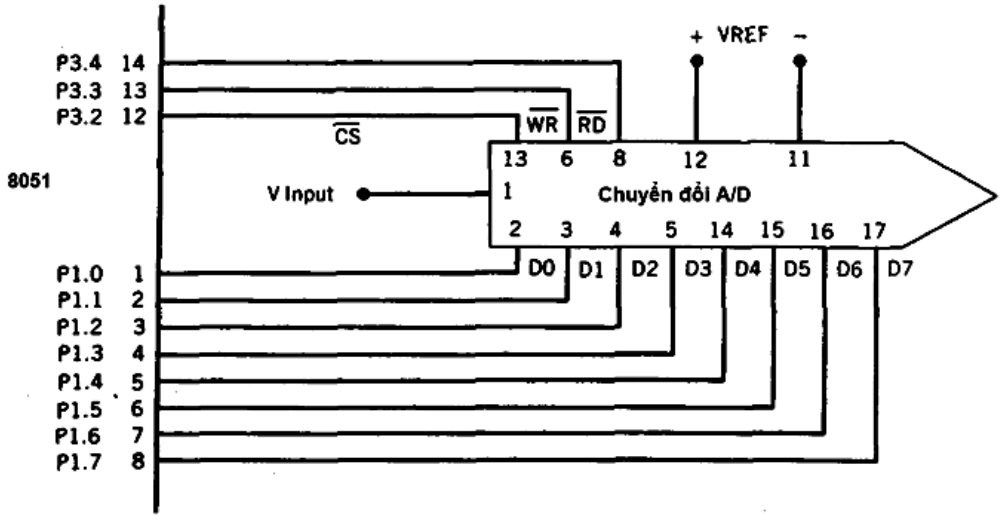
3.2 Chuyển đổi tương tự số (A/D)

Hình 6-4 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 6-4. Ghép nối 8051 và chuyển đổi A/D

Ví dụ 6-3. 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 7. GIỚI THIỆU MỘT SỐ VI XỬ LÍ TIÊN TIẾN

1. CÁC VI XỬ LÍ TIÊN TIẾN DỰA TRÊN KIẾN TRÚC INTEL IA-32

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.

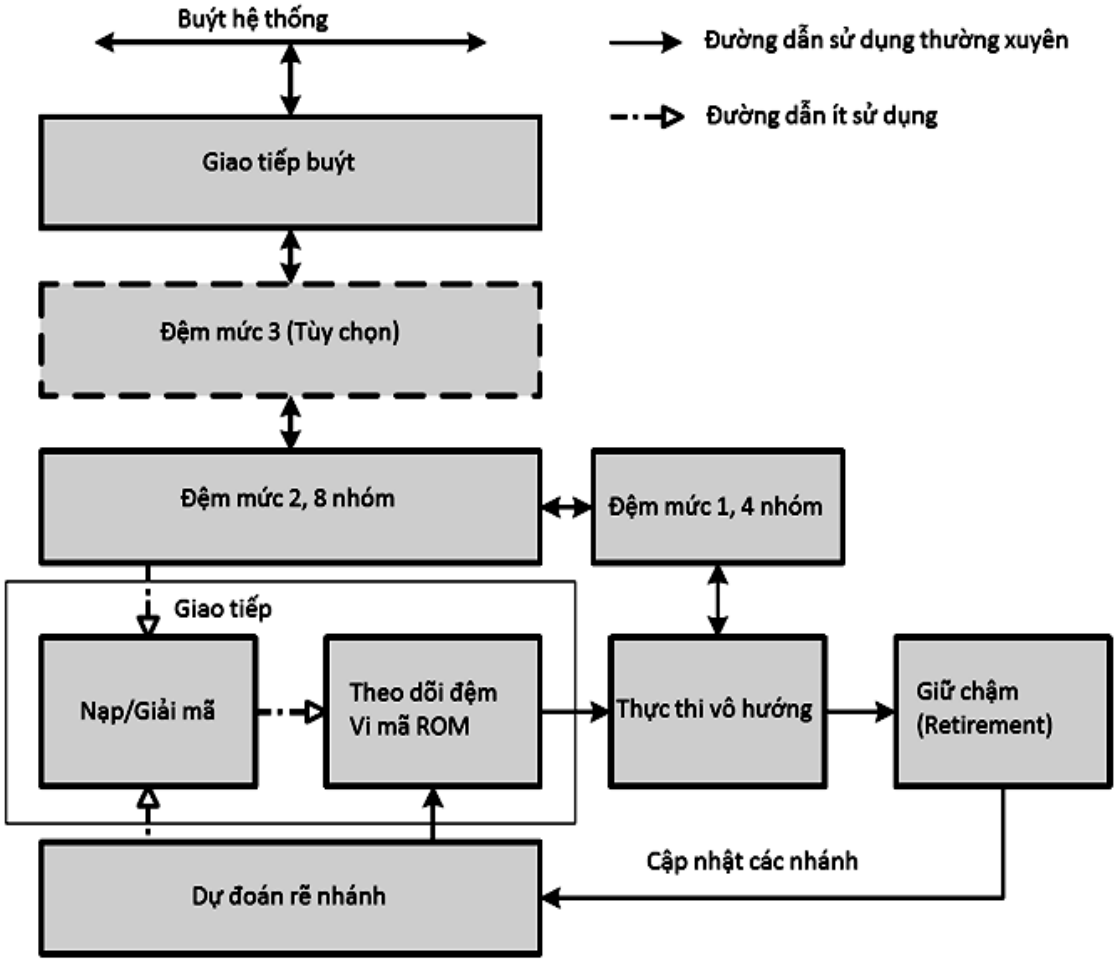
  Trong chế độ này bộ xử lý chuyển qua lại các không gian địa chỉ riêng biệt trong khi lưu lại ngữ cảnh căn bản của các chương trình đang chạy. Các đoạn mã SMM có thể được thực hiện hoàn toàn trong suốt. Ngay khi quay trở lại từ chế độ SMM, bộ xử lý được khôi phục lại trạng thái giống như trước khi ngắt 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 232 byte hay 4GB và không gian địa chỉ vật lý có thể lên tới 236 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 7-1. 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 7-1. 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 16383 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

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 7-1. 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 7-2. 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

2. CÁC VI XỬ LÍ 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 7-3. 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	2005	Netburst,	3, 73	32 GP	8, 5GB/s	64GB	16KB

4 Extreme		Siêu phân luồng, IA-64		80 FPU 64 MMX 128 XMM			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

3. CÁC VI XỬ LÍ 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 làm 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 7-4. 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 – 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.