

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

GIÁO TRÌNH

Kỹ thuật vi xử lý

TẬP 1



NHÀ XUẤT BẢN BƯU ĐIỆN

GIÁO TRÌNH

Kỹ thuật vi xử lý

(TẬP 1)

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

GIÁO TRÌNH

Kỹ thuật vi xử lý

(TẬP 1)

Biên soạn: TS. Hồ Khánh Lâm

NHÀ XUẤT BẢN BƯU ĐIỆN

Hà Nội - 2007

Mã số: GT 01 HM 07

LỜI GIỚI THIỆU

Chiến lược phát triển công nghệ thông tin và truyền thông (ICT) của đất nước ta coi đào tạo nguồn nhân lực, đặc biệt là đội ngũ kỹ sư hệ thống công nghệ điện tử - viễn thông - tin học là một yêu cầu cấp bách đối với công tác nghiên cứu và giáo dục đào tạo ở hệ cao đẳng và đại học. Kỹ thuật vi xử lý là kiến trúc lõi cơ bản nhất trong hầu hết các hệ thống thiết bị điện tử, viễn thông và tự động hóa, cũng như mạng lưới và các hệ thống xử lý thông tin hiện đại khác. Do vậy, hiểu biết "Kỹ thuật vi xử lý" trở thành nhu cầu không thể thiếu của các kỹ thuật viên, cán bộ, chuyên gia trong lĩnh vực Điện tử - Viễn thông - Tin học.

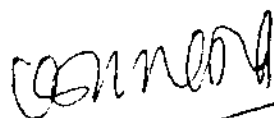
Cùng với nhiều tài liệu, sách tham khảo "Kỹ thuật vi xử lý" đang lưu hành, Học viện Công nghệ Bưu chính Viễn thông và tác giả cho ra đời quyển giáo trình "Kỹ thuật vi xử lý" này trên cơ sở tuân thủ nội dung của một giáo trình, được Hội đồng nghiệm thu phê duyệt, với nhiều nội dung tham khảo phong phú nhằm mở rộng kiến thức cho nhiều loại đối tượng sử dụng trong giảng dạy, học tập và nghiên cứu: giảng viên, sinh viên các trường Đại học, Cao đẳng, các trường Kỹ thuật dạy nghề, các học viên cao học và nói chung những ai khi học tập nghiên cứu về kỹ thuật vi xử lý, máy vi tính nói riêng và kỹ thuật tin học nói chung.

Nội dung của giáo trình đề cập những khái niệm cơ bản về vi xử lý, về công nghệ của một số vi xử lý thông dụng từ 8-bit đến 64-bit, từ kiến trúc tập lệnh phức tạp (CISC) đến kiến trúc tập lệnh giảm thiểu (RISC), được các nhà sản xuất hàng đầu thế giới như Intel, Zilog, Motorola, HP, IBM, Sun,... tạo ra; diễn giải các tập lệnh của các vi xử lý thông qua lập trình hợp ngữ, các nguyên tắc phối ghép và trao đổi thông tin giữa vi xử lý với

thế giới vật lý bên ngoài (bộ nhớ, thiết bị ngoại vi,...); trình bày các loại thiết bị nhớ bằng vật liệu bán dẫn, vật liệu từ, quang học. Giáo trình tập trung trình bày kỹ công nghệ vi xử lý Intel từ 808x đến Pentium M công nghệ Centrino Mobile do sự ứng dụng phổ biến của chúng trong các thế hệ máy vi tính từ trước đến nay và cũng để đảm bảo thỏa mãn đa số nội dung giảng dạy và thực tập môn kỹ thuật vi xử lý trong các cơ sở đào tạo. Các loại vi điều khiển và xử lý tín hiệu cũng được trình bày một cách cơ bản với những ví dụ ứng dụng đơn giản của chúng. Giáo trình có phần bài tập giúp cho người đọc ôn luyện kiến thức dễ dàng.

Nội dung của giáo trình thể hiện sự đầu tư sức lao động nghiêm túc, công phu cùng với kinh nghiệm thực tế và đào tạo của tác giả trong lĩnh vực "Kỹ thuật vi xử lý" và máy tính điện tử nhiều năm với mong muốn đóng góp cho sự nghiệp đào tạo của ngành Bưu chính Viễn thông và công nghệ thông tin nói riêng và giáo dục đào tạo nói chung trong sự nghiệp phát triển nguồn nhân lực trí tuệ cho đất nước.

Xin trân trọng giới thiệu.



GS. TSKH. ĐỖ TRUNG TÁ

LỜI NÓI ĐẦU

Ngày nay, bộ vi xử lý được sử dụng rộng rãi không chỉ trong lĩnh vực máy tính điện tử mà còn trong rất nhiều hệ thống điều khiển khác. Với sự phát triển của công nghệ mạch tích hợp một bộ vi xử lý có thể được chế tạo từ một hoặc vài vi mạch tích hợp cỡ lớn, chứa hàng ngàn hoặc hàng triệu transistor. Dưới sự điều khiển của các chương trình chứa trong bộ nhớ, bộ vi xử lý thực hiện các phép tính số học, logic, các phép toán khác và kết nối, trao đổi dữ liệu với các thiết bị bên ngoài thông qua các cổng vào/ra. Do vậy, vấn đề đặt ra là phải hiểu về kỹ thuật vi xử lý và lập trình cho vi xử lý như thế nào để tự động điều khiển mang lại hiệu quả cao.

Để đáp ứng yêu cầu trên, Học viện Công nghệ Bưu chính Viễn thông phối hợp với Nhà xuất bản Bưu điện xuất bản cuốn sách **“Giáo trình Kỹ thuật Vi xử lý”** do TS. Hồ Khánh Lâm chủ biên. Giáo trình gồm 8 chương, chia làm 02 tập. Tập 1 từ chương 1 đến chương 4 giới thiệu những khái niệm cơ bản về hệ thống vi xử lý như kiến trúc Von Neumann, kiến trúc Harvard; đặc điểm, cấu trúc và đặc tính nâng cao tốc độ của bộ vi xử lý; cấu trúc và hoạt động của họ vi xử lý từ 8085, 8086/8088, 8087, 80x86, Z80, đến Pentium III, IV, Motorola68000,... Bên cạnh đó cuốn sách không chỉ giới thiệu về lệnh, các dạng lệnh, cách mã hóa lệnh, các mã lệnh và tập lệnh của vi xử lý mà còn giới thiệu cụ thể về các khái niệm, cách thức lập trình hợp ngữ, cơ chế làm việc, cấu trúc của chương trình con, macro và các vấn đề liên giúp cho người lập trình tự tạo được những lệnh điều khiển

riêng theo ý muốn. Tập 2 từ chương 5 đến chương 8 trình bày về cách ghép nối CPU với bộ nhớ và các thiết bị ngoại vi; các phương pháp điều khiển vào/ra dữ liệu; giới thiệu các bộ vi xử lý công nghệ tiên tiến và bộ vi điều khiển. Ngoài ra cuốn sách còn giới thiệu một số phụ lục về các tập lệnh của Intel 8086, Intel x86, Z80, Motorola M680X0, Intel 8051, Intel 8085A,... để bạn đọc tiện tham khảo. Đặc biệt cuốn sách còn có phần bài tập ôn luyện giúp cho bạn đọc tự củng cố và nâng cao kiến thức của bản thân.

Giáo trình này không chỉ đề cập đến những khái niệm cơ bản mà còn đưa ra một số kỹ năng, kinh nghiệm cần thiết để phát triển (hay lập trình) các chương trình điều khiển cụ thể. Hy vọng cuốn sách sẽ hữu ích cho học sinh, sinh viên, các kỹ thuật viên, cán bộ, chuyên gia ngành điện tử viễn thông và công nghệ thông tin, các ngành kỹ thuật có liên quan khác và những người yêu thích tìm hiểu về kỹ thuật vi xử lý.

Xin trân trọng giới thiệu cùng bạn đọc.

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Chương 1

GIỚI THIỆU CHUNG

1.1. TRAO ĐỔI THÔNG TIN TRONG VI XỬ LÝ

1.1.1. Khái niệm vi xử lý

Vi xử lý là một vi mạch có mức tích hợp rất lớn (VLSI- Very Large Scale Integrated circuit) có thể lập trình được, và dưới sự điều khiển của chương trình chứa trong bộ nhớ, nó thực hiện các phép tính số học, logic, các phép toán khác, và kết nối trao đổi dữ liệu với các thiết bị bên ngoài thông qua các cổng vào/ra (Input/Output ports).

Vi xử lý có hai loại phổ biến: vi xử lý đa năng (general-purpose microprocessors) và vi điều khiển (microcontrollers).

Vi xử lý đa năng chứa bên trong chip tất cả các thành phần chủ yếu cho tính toán ngoại trừ bộ nhớ và các cổng vào/ra.

Vi điều khiển chứa trong chip bộ nhớ và các cổng vào/ra.

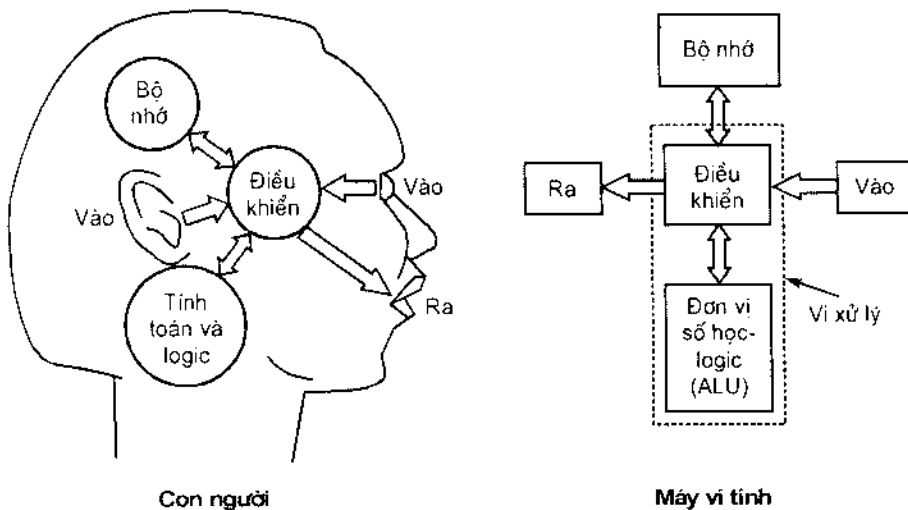
Xử lý dữ liệu là chức năng chính của vi xử lý. Xử lý dữ liệu bao gồm cả tính toán và vận chuyển dữ liệu.

1.1.2. Trao đổi thông tin trong vi xử lý

Máy vi tính (microcomputer) là một hệ thống gồm có 5 khối phân cứng: vào, ra, bộ nhớ, đơn vị số học - logic (ALU) và điều khiển. Vi xử lý là đơn vị xử lý trung tâm của máy vi tính gồm hai khối phân cứng là đơn vị số học - logic và điều khiển (hình 1.1). Hệ thống xử lý thông tin trong bộ não người cũng có 5 khối chức năng tương tự: vào (mắt, tai), ra (miệng), điều khiển, bộ nhớ, tính toán và logic trong bộ não.

Khối các mạch logic bên trong vi xử lý thực hiện phép tính số học và logic như: cộng, trừ, nhân, chia, và, hoặc, so sánh, tăng, giảm,... với các dữ liệu thường được gọi là đơn vị số học-logic ALU

(Arithmetic Logic Unit). Dữ liệu để ALU thực hiện phép tính phải được đặt vào đúng chỗ qui định. ALU không thể tự chuyển dữ liệu từ chỗ này đến chỗ khác, mà thay vào đó nó chỉ đơn thuần thực hiện các phép tính với dữ liệu ở chỗ nào đó và nó để lại kết quả ở lại chính chỗ đó. Như vậy để có dữ liệu để tính toán, ALU phải chờ đợi dữ liệu hiện diện ở một chỗ nào đó ấn định cho nó.



Hình 1.1: Năm khối phân chức của máy vi tính giống như năm khối xử lý thông tin ở con người

Vi xử lý có một số mạch logic, bên ngoài ALU, thực hiện vận chuyển dữ liệu từ chỗ này đến chỗ khác cần thiết cho xử lý dữ liệu. Chúng cùng với ALU làm thành logic xử lý dữ liệu: tính toán và vận chuyển.

Nhưng làm thế nào để ALU biết được rằng nó phải thực hiện xử lý dữ liệu? Cái gì mách bảo ALU thực hiện xử lý dữ liệu và bằng phép tính nào? Như vậy phải có những chỉ dẫn cho ALU. Các phép tính khác nhau đòi hỏi ALU phải có những thao tác khác nhau, do đó phải có những chỉ dẫn khác nhau. Và bởi vì vi xử lý là tập hợp các mạch logic nên mỗi một chỉ dẫn (instruction), thường gọi là lệnh máy, phải là một tập hợp các giá trị logic “0” và “1”, sắp xếp theo một trật tự nào đó có ý nghĩa khác nhau đối với các chỉ dẫn khác nhau. Mỗi vi xử lý

có một tập hợp các lệnh máy riêng. Nếu muốn vi xử lý thực hiện một loạt các hành vi nào đó thì ta phải có một tập hợp các lệnh sắp xếp theo một trình tự. Sự thực hiện của vi xử lý theo một trình tự các lệnh gọi là sự thực hiện theo chương trình. Một chương trình (program) gồm các lệnh sắp xếp theo thứ tự tiến hành xử lý dữ liệu của vi xử lý. Do vậy, vi xử lý là một vi mạch lập trình, nghĩa là bằng chương trình có thể điều khiển vi xử lý thực hiện các thao tác xử lý dữ liệu theo mong muốn của người lập trình.

Chương trình được lưu trong bộ nhớ bên ngoài vi xử lý và người lập trình có thể thay đổi được nó, tức là có thể thay đổi hành vi của vi xử lý. Từng lệnh của chương trình phải được đọc từ bộ nhớ (instruction fetch) vào vi xử lý để tác động vi xử lý. Trong trường hợp này vi xử lý cần phải có một số mạch logic tiếp nhận lệnh từ bộ nhớ và diễn giải lệnh sao cho vi xử lý hiểu phải làm gì. Khối mạch logic này gọi là logic điều khiển, và chức năng diễn lệnh dẫn gọi là giải mã lệnh (decode instruction). Kết quả giải mã lệnh là những tín hiệu đưa đến các mạch logic trong và ngoài ALU để thao tác xử lý dữ liệu theo một trình tự. Đó là bước thực hiện lệnh (execute instruction). Kết quả thực hiện lệnh bên trong vi xử lý có thể đòi hỏi phải được chuyển ra bộ nhớ, hoặc phải được tiếp tục xử lý thêm cùng với dữ liệu được đọc từ bộ nhớ bên ngoài vi xử lý. Như vậy, theo từng lệnh, bên trong vi xử lý diễn ra quá trình đọc lệnh từ bộ nhớ bên ngoài vi xử lý, giải mã lệnh, thực hiện lệnh, và chuyển kết quả thực hiện lệnh ra bộ nhớ bên ngoài.

Như vậy với chức năng chính là xử lý dữ liệu thì bên trong của vi xử lý có các khối logic xử lý dữ liệu và khối logic điều khiển dữ liệu. Logic xử lý dữ liệu thực hiện vận chuyển dữ liệu từ chỗ này đến chỗ khác, và thực hiện các phép tính với dữ liệu. Logic điều khiển xác định các mạch logic nào thực hiện xử lý dữ liệu và một nhiệm vụ quan trọng nữa là: điều khiển vi xử lý làm việc với tất cả mạch kết nối với vi xử lý như bộ nhớ, các cổng vào/ra với các thiết bị ngoại vi để trao đổi dữ liệu.

Sự vận chuyển dữ liệu từ chỗ này đến chỗ khác, từ khối logic chức năng này đến khối logic chức năng khác, bên trong vi xử lý được thực hiện thông qua những tuyến đường dữ liệu chung, gọi là đường trục dữ liệu trong (Internal data bus). Sự vận chuyển dữ liệu giữa vi xử lý và các mạch bên ngoài cũng được thực hiện qua bus dữ liệu trong. Đối với các vi điều khiển, vì bên trong chúng có bộ nhớ và các cổng vào/ra, nên sự trao đổi dữ liệu giữa vi điều khiển và các mạch bên ngoài được thực hiện thông qua các cổng vào/ra.

Mọi quá trình xử lý dữ liệu bên trong vi xử lý và sự trao đổi dữ liệu giữa vi xử lý phải được đồng bộ theo một tiến trình thời gian. Điều này có nghĩa là vi xử lý phải có một đồng hồ chuẩn (clock). Phải có một bộ tạo nhịp đồng hồ cho vi xử lý. Thời gian đồng hồ nhanh hay chậm đảm bảo vi xử lý thực hiện xử lý dữ liệu nhanh hay chậm. Công nghệ chế tạo chip vi xử lý quyết định nhịp đồng hồ phải nhanh hay chậm.

1.2. SỰ PHÁT TRIỂN VÀ ỨNG DỤNG CỦA CÁC BỘ VI XỬ LÝ

Kỹ thuật vi xử lý gắn liền với sự phát triển của công nghệ mạch tích hợp. Mức độ tích hợp càng cao thì các chip vi xử lý càng có công suất xử lý lớn hơn.

Năm 1970, Intel đã cho ra đời chip vi xử lý đầu tiên là i4004 với 4-bit bus dữ liệu (data bus), 12-bit bus địa chỉ (address bus) (kết hợp bus dữ liệu và bus địa chỉ), công nghệ mạch tích hợp PMOS, đóng vỏ gốm kích thước 24 mm² 16 chân DIP (Dual In-line Package) và chứa 2250 transistor. Bộ vi xử lý này chủ yếu dùng để chế tạo các máy tính để bàn và bỏ túi (calculators), nó dễ dàng thực hiện các tính toán ở mã nhị phân BCD (Binary Coded Decimal).

Năm 1972, ra đời chip i8008 với 8-bit bus dữ liệu, 14-bit bus địa chỉ (kết hợp bus dữ liệu và bus địa chỉ), tần số làm việc 300 kHz, công nghệ tích hợp PMOS, đóng vỏ gốm 18 chân và chứa 3.300 transistor, nhưng chưa tìm được ứng dụng nhiều để chế tạo máy tính.

Phải đến năm 1974, chip Intel i8080/2 MHz, công nghệ PMOS, và i8080A/2,67 MHz ÷ 3,125 MHz, công nghệ NMOS với 8-bit bus dữ

liệu, 16-bit bus địa chỉ, đóng vỏ gồm 40 chân DIP trở thành cơ sở để chế tạo các máy tính cá nhân sử dụng hệ điều hành CP/M. Cũng thời gian này Motorola đã sản xuất loạt chip vi xử lý MC6800 với 8-bit bus dữ liệu, 16-bit bus địa chỉ, tần số làm việc $1 \div 2$ MHz, đóng vỏ DIP có 68E3 transistor, và là CPU trong các họ máy tính cá nhân Apple II.

Năm 1976, nhà sản xuất Zilog dựa trên cơ sở của 8080 để cho ra đời họ vi xử lý Z80/8-bit với các tần số làm việc: 2,5 MHz, 4 MHz, 6 MHz, 8 MHz, 10 MHz, và đóng vỏ 40 chân DIP. Nó bao gồm các lệnh của 8080, nhưng có thêm một số lệnh mới như: lệnh SIMK (Set Interrupt Mask), RIM (Read Interrupt Mask) và nhiều chức năng khác hay hơn. Điều này có nghĩa là chương trình viết cho 8080 có thể chạy được bình thường trên Z80. Cũng trong thời gian này, xuất hiện trên thị trường chip 8-bit bus dữ liệu và 16-bit bus địa chỉ (kết hợp bus dữ liệu và bus địa chỉ) là i8085A, 3 MHz, công nghệ NMOS và i8085AH, 5 MHz $\div$ 6 MHz, công nghệ HMOS, đóng vỏ gồm DIP, mật độ 6.200 transistor. Những chip này đã tìm được nhiều ứng dụng thực tế trong các hệ thống thiết bị ứng dụng vi xử lý và là chip mẫu dùng trong đào tạo kiến thức về vi xử lý thời kỳ này.

Năm 1978, Intel đã sản xuất các họ vi xử lý i8086A/i80C86A 16-bit bus dữ liệu trong (internal data bus) và 16-bit bus dữ liệu ngoài (external data bus), 20-bit bus địa chỉ (kết hợp bus dữ liệu và bus địa chỉ), tần số 4 MHz (công nghệ NMOS), 5 $\div$ 10 MHz (công nghệ CMOS) đóng vỏ gồm 40 chân DIP, được dùng làm CPU trong IBM PC/XT. Do chi phí đắt, và các thiết bị ngoại vi vẫn chỉ thỏa mãn trao đổi dữ liệu 8-bit nên dòng sản phẩm này không thông dụng. Trong khi đó, các chip i8088A/i80C88A, với 16-bit đường trục dữ liệu trong, 8-bit đường trục dữ liệu ngoài, tần số 5 $\div$ 10 MHz, công nghệ CMOS, đóng vỏ 40 chân DIP, ra đời vào năm 1979 tìm được ứng dụng thực tế trong các loạt IBM PC/XT bán chạy trên thị trường.

Năm 1982, i80286 với 16-bit bus dữ liệu trong và bus dữ liệu ngoài, 24-bit bus địa chỉ, tần số 6 $\div$ 20 MHz, công nghệ HMOS, đóng vỏ gồm 68 chân DIP, đã trở thành chip vi xử lý cho các loại IBM

PC/AT. Nhưng trước đó, năm 1979 Motorola đã cho ra đời bộ vi xử lý MC68000 16-bit bus dữ liệu trong và bus dữ liệu ngoài, 24-bit bus địa chỉ, là CPU của loạt các máy tính cá nhân như Atari ST, Commodore Amiga, Apple Lisa, Macintosh. Nó là cơ sở cho họ vi xử lý MC680X0 nổi tiếng.

Năm 1982, Motorola đã đi trước Intel cho ra đời loạt vi xử lý MC68020 với 32-bit bus dữ liệu trong và bus dữ liệu ngoài, 32-bit bus địa chỉ, đặc biệt có bộ nhớ truy cập nhanh (Cache memory) bên trong 256 byte, đơn vị quản lý bộ nhớ MMU (Memory Management Unit) bên ngoài, tần số $16,67 \div 33,33$ MHz, được sử dụng trong các hệ thống máy tính đa nhiệm có quản lý địa chỉ ảo, như Amiga, Apple Macintosh, Sun3, Atari TT030, Atari Falcon 030, và các hệ thống tổng đài điện thoại, chuyển mạch gói,... do chúng có tập thanh ghi lớn tạo khả năng lập trình linh hoạt.

Trong khi đó, năm 1985 Intel mới có các chip i80386/i80386DX (DX: Double-word eXternal) với 32-bit bus dữ liệu trong và bus dữ liệu ngoài, 32-bit bus địa chỉ, tần số $12 \div 33$ MHz, công nghệ 0,8 micron CMOS, đóng vỏ 132 chân PGA (Pin Grid Array) với mật độ 275E3 transistor, và là CPU trong các loạt IBM PC386DX thông dụng.

Mãi tới năm 1988, loạt chip i80386SX với 32-bit bus dữ liệu trong, 16-bit bus dữ liệu ngoài (SX: Single-word eXternal), tần số $16 \div 33$ MHz, công nghệ 0,8 micron CMOS, đóng vỏ 100 pin QFP (Quad Flat Package), công nghệ 0,8 micron CMOS, đóng vỏ 100 chân QFP (Quad Flat Package), 256 byte dữ liệu và 16 byte cache lệnh bên trong, MMU bên ngoài, và i80386SL, một phiên bản của i80386SX tiêu thụ nguồn mới xuất hiện để làm CPU trong các IBM PC386SX. Tiếp theo i80386, năm 1989, loạt 80486DX P4 có FPU (Floating Point Unit) và MMU bên trong, tần số $20 \div 33$ MHz (công nghệ CHMOS IV), 50 MHz (công nghệ CHMOS V), đóng vỏ 168 chân PGA kích thước 165 mm^2 với mật độ 1,2E6 transistor. Đến năm 1993, đã có các phiên

bản của 80486DX P4 là loạt i80486DX2/DX4, tần số $25 \div 100$ MHz. Tương tự như i80386, năm 1991 cũng có loạt i80486SX, i80486SL và i80486SXL.

Năm 1993 ra đời loạt vi xử lý Intel Pentium, i586 hay P5, với 32-bit bus dữ liệu trong, 64-bit bus dữ liệu ngoài, 32-bit bus địa chỉ, đặc biệt có kiến trúc siêu hướng 5 giai đoạn 2 cửa ra (2-issue 5-stage superscalar) bằng đường ống FPU 8 giai đoạn, 16 kbyte bộ nhớ cache bên trong chip (8 kbyte cho lệnh và 8 kbyte cho dữ liệu), tần số $60 \div 133$ MHz, công nghệ 0,8 micron biCMOS, đóng gói kích thước 18×16 mm, 273 chân PGA mật độ $3,1E6$ transistor. Tiếp theo là các loạt Pentium với công nghệ MMX (Matrix Math eXtensions, Multi-Media eXtensions), tần số $66 \div 266$ MHz và Pentium II/MMX, tần số $66 \div 300$ MHz, công nghệ 0,35 micron CMOS, đóng gói kích thước 203 mm^2 mật độ $7,5E6$ transistor. Các loạt Pentium Celeron, tần số $66 \div 333$ MHz, là các phiên bản dùng cho máy tính cá nhân không hỗ trợ cho cấu hình đa xử lý cũng được tung ra thị trường với giá rẻ. Tiếp theo là loạt Pentium II Xeon (P6 core) có hỗ trợ đa xử lý, và tần số đạt tới 450 MHz (1998).

Năm 2000 Intel cho ra thị trường loạt Pentium III/Xeon, tần số đến $500 \text{ MHz} \div 1 \text{ GHz}$, và năm 2001, xuất hiện Pentium IV $1,7 \div 2 \text{ GHz}$.

Đáng kể trong cạnh tranh với Intel là các nhà sản xuất chip Cyrix, AMD và Motorola. Năm 2000 AMD đã cho ra thị trường loạt vi xử lý tương đương với Pentium III là Athlon 1,1 GHz. Motorola cũng đã lần lượt cho ra đời các loạt MC68030 với 256×2 byte bộ nhớ cache trong chip (1987) và mật độ $270E3$ transistor, MC68040 (1990-1991) với 4×2 kbyte bộ nhớ cache trong chip, mật độ $1,2E6$ transistor, MC68060 (1994) với 8×2 kbyte bộ nhớ cache trong chip, kiến trúc siêu hướng, mật độ $2,3E6$ transistor và làm việc ở tần số 66 MHz. Các hãng khác như Zilog cũng có các chip vi xử lý Z80000 32-bit, National Semiconductor với NS16000 16-bit và NS32xxx 32-bit, DEC

với loạt DECchip-210 Alpha với kiến trúc siêu hướng 64-bit, máy tính có tập lệnh tối thiểu RISC (Reduced Instruction Set Computer), tần số đến 600 MHz, và loạt DECchip-211 Alpha, tần số đến 1 GHz (1999), công nghệ 0,18 micron CMOS. Hewlett-Packard với các chip PA-RISC, AMD với các chip Am29000, IBM và Motorola với các chip PowerPC 604-750 (1998), và SUN Micro systems với các chip SPARC, UltraSPARC cũng chiếm nhiều thị phần về các chip vi xử lý kiến trúc siêu hướng đường ống và RISC trên thị trường máy tính thế giới. Các họ máy tính có kiến trúc RISC đã được Intel quan tâm và cho ra họ i80860, Motorola cho ra họ MM88000.

Các bộ vi xử lý hiện nay là các hệ thống 32-bit và 64-bit đầy đủ và chúng hoạt động ở các tần số 300 MHz ÷ 2 GHz và cao hơn, được đóng gói trong các chip vi mạch có mức tích hợp rất lớn VLSI (Very Large-Scale Integration) với hàng chục triệu transistor.

Phần lớn các bộ vi xử lý mới thực hiện các lệnh trong một chu kỳ, và tất cả chúng đều có đơn vị xử lý dấu phẩy động FPU (Floating-point Unit), MMU và cache nhớ bên trong. Chúng có số lượng lớn các thanh ghi chung 16 - 32-bit. Do vậy, tập hợp các thanh ghi bên trong chip được gọi chung là tệp thanh ghi (register file). Có tệp thanh ghi cho đơn vị nguyên (IU: Interger Unit) và tệp thanh ghi cho FPU. Bộ nhớ cache bên trong (là bộ nhớ trung gian tốc độ nhanh) có dung lượng lên đến 32 kbyte ÷ 128 kbyte. Sự phân chia bộ nhớ cache bên trong thành: cache lệnh (Icache: Instruction cache) dùng cho lệnh và cache dữ liệu (Dcache: Data cache) với dung lượng tối ưu đảm bảo nâng cao hiệu suất trao đổi dữ liệu giữa CPU và hệ thống nhớ của máy tính.

Sự phát triển của công nghệ mạch tích hợp, công nghệ vi xử lý và đánh dấu một mốc lịch sử phát triển của công nghệ máy tính. Thế hệ máy tính điện tử vạn năng đầu tiên đáng kể hơn cả là UNIVAC (Universal Automatic Computer) (1951) và ENIAC (Electronic Numerical Intergrator and Calculator) (1960) của Hãng Eckert-Mauchly và trường Đại học Tổng hợp Pennsylvania (Mỹ) dựa trên các

bóng đèn chân không ba cực (triode). Thế hệ máy tính điện tử vạn năng thứ hai dựa trên công nghệ bóng bán dẫn ra đời trong khoảng những năm 1955 ÷ 1964 (IBM 7090). Cuối những năm 1960, thế hệ máy tính điện tử vạn năng thứ ba đã dựa trên công nghệ vi mạch tích hợp cỡ nhỏ SSI (Small-Scale Intergration) và trung bình MSI (Medium-Scale Intergration) (các loạt IBM 360-370). Thế hệ máy tính điện tử thứ tư dựa trên công nghệ tích hợp lớn LSI và rất lớn VLSI, mà cơ sở của chúng là các chip vi xử lý. Sự phát triển rõ rệt và đem lại cuộc cách mạng trong công nghệ máy tính bắt đầu từ cuối những năm 1970 đầu năm 1980, khi những thế hệ đầu tiên của máy tính cá nhân sử dụng công nghệ vi xử lý 8-bit xuất hiện trên thị trường. Các bộ vi xử lý công nghệ cao hiện nay (advanced microprocessors) đã thỏa mãn được yêu cầu chế tạo các loại máy tính khác nhau: máy vi tính, máy tính lớn (mainframes) và các siêu máy tính (superecomputers).

1.3. HỆ THỐNG VI XỬ LÝ

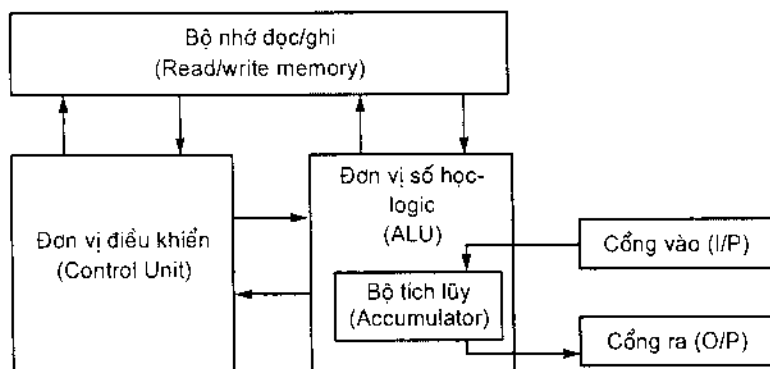
1.3.1. Khái niệm máy tính kiến trúc Von Neumann và kiến trúc Havard

Các máy tính điện tử số thường có hai loại kiến trúc: kiến trúc Von Neumann và kiến trúc Havard.

Các máy tính có kiến trúc Von Neumann dựa trên ba khái niệm cơ bản, đó là:

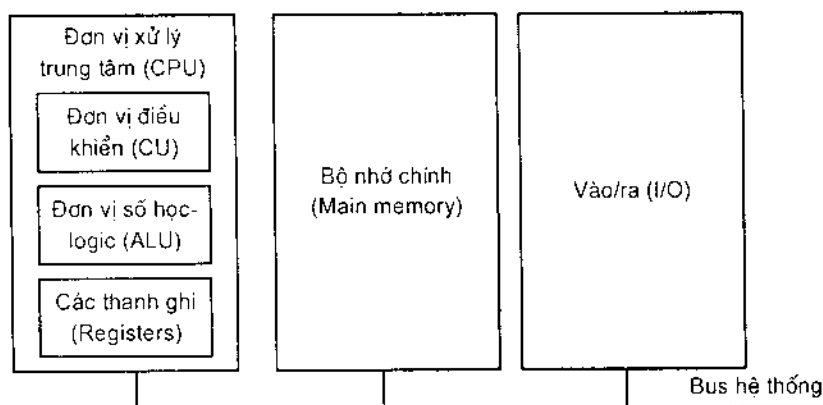
- + Dữ liệu (data) và các lệnh (instructions) lưu giữ trong bộ nhớ đọc/ghi riêng.
- + Các nội dung của bộ nhớ đọc/ghi được đánh địa chỉ theo vùng, không phụ thuộc vào loại dữ liệu mà bộ nhớ lưu giữ.
- + Quá trình thực hiện các lệnh xảy ra theo tuần tự - lệnh tiếp theo lệnh.

Sơ đồ kiến trúc của máy tính Von Neumann nguyên thủy mô tả trong hình 1.2.



Hình 1.2: Kiến trúc máy tính Von Neumann nguyên thủy

Sơ đồ kiến trúc của máy tính Von Neumann hiện đại mô tả trong hình 1.3.



Hình 1.3: Kiến trúc máy tính Von Neumann hiện đại

Đơn vị xử lý trung tâm (CPU: Central Processing Unit) thực hiện các chức năng chính quyết định tên gọi của máy tính: đó là thực hiện các phép xử lý dữ liệu như các phép tính số học, logic,... các trao đổi dữ liệu bộ nhớ, với các ngoại vi... CPU bao gồm tập hợp các khối mạch logic điều khiển, gọi là đơn vị điều khiển (CU), các mạch logic thực hiện các phép tính số học và logic, gọi là đơn vị số học-logic (ALU) và tập hợp các thanh ghi (Registers) có khả năng lưu trữ tạm thời, trao đổi dữ liệu, xử lý dữ liệu với mức độ tích hợp khác nhau. Các lệnh và dữ liệu được lưu giữ trong bộ nhớ chính. Các thiết bị ngoại vi

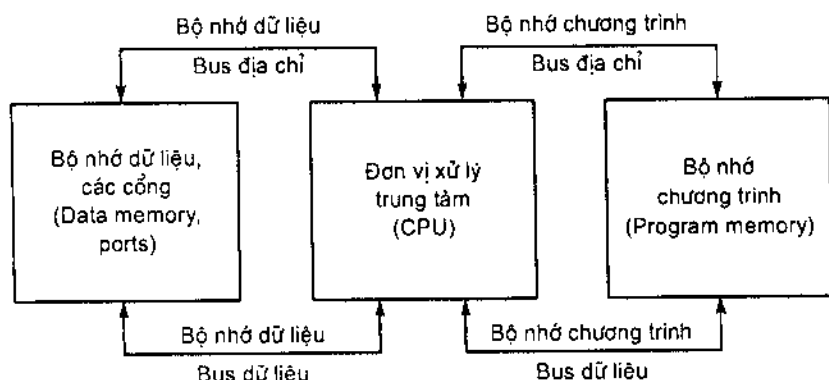
kết nối với CPU và bộ nhớ chính thông qua các cổng và điều khiển vào/ra (I/O). Sự trao đổi dữ liệu giữa các thành phần trong máy tính kiến trúc Von Neumann hiện đại được thực hiện thông qua bus hệ thống (system bus). Theo khái niệm cơ bản, sự thực hiện lệnh được đơn vị xử lý trung tâm thực hiện theo từng bước như sau:

1. Đọc lệnh tiếp theo từ bộ nhớ chính vào trong thanh ghi lệnh của CPU;
2. Thay đổi bộ đếm chương trình (PC: Program Counter) để trở tới lệnh tiếp theo;
3. Xác định kiểu lệnh vừa đọc vào từ bộ nhớ;
4. Nếu lệnh sử dụng dữ liệu trong bộ nhớ thì xác định dữ liệu đó ở đâu;
5. Đọc dữ liệu (nếu có) vào một thanh ghi bên trong của CPU;
6. Thực hiện lệnh;
7. cất giữ kết quả thực hiện lệnh;
8. Trở lại bước 1.

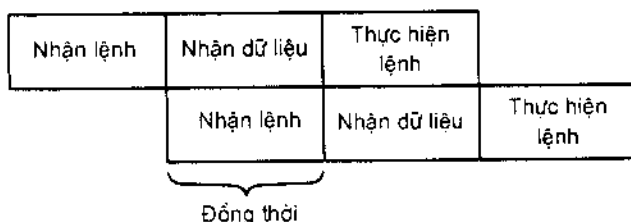
Trình tự này thường gọi là: chu trình đọc-giải mã-thực hiện lệnh.

Bus hệ thống của máy tính kiến trúc Von Neumann thường được phân ra: bus dữ liệu, bus địa chỉ và bus điều khiển (control bus). Máy tính kiến trúc Von Neumann cho phép dễ dàng thực hiện thiết kế, mở rộng cấu hình. Nhưng vì sử dụng chung bus hệ thống cho mọi trao đổi dữ liệu giữa các thiết bị kết nối trên bus nên tốc độ xử lý chậm, hiệu suất thấp.

Máy tính có kiến trúc Havard mô tả trong hình 1.4. lại có sự phân chia bộ nhớ chính thành bộ nhớ chỉ chứa chương trình (program memory) và bộ nhớ chứa dữ liệu và các cổng vào/ra (data memory and ports), như vậy kết nối giữa đơn vị xử lý trung tâm với hai bộ nhớ được thực hiện qua các bus riêng biệt. Nhờ đó, tốc độ xử lý dữ liệu nhanh hơn, hiệu suất cao hơn (hình 1.5).



Hình 1.4: Kiến trúc máy tính Havard

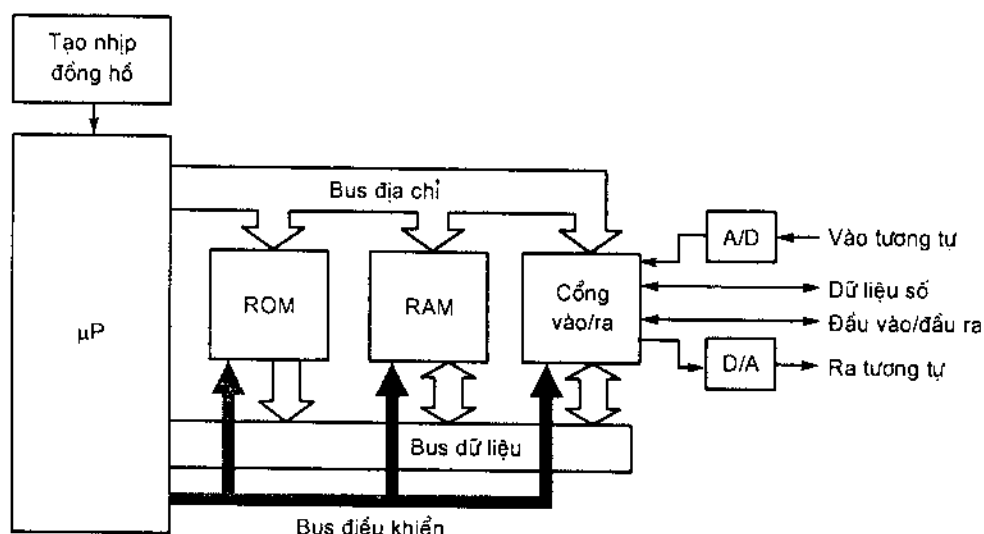


Hình 1.5: Nhận dữ liệu và lệnh đồng thời trong máy tính kiến trúc Havard

1.3.2. Hệ thống vi xử lý kiến trúc Von Neumann và kiến trúc Havard

Bộ vi xử lý (μP), cũng giống như CPU của máy tính kiến trúc Von Neumann hay Havard, nó thực hiện xử lý và tính toán dữ liệu dưới sự điều khiển của chương trình. Nhưng điều khác cơ bản với CPU của máy tính bình thường ở chỗ là toàn bộ các khối mạch logic (CU, ALU, các thanh ghi bên trong) được tích hợp trong một chip vi mạch tích hợp mật độ cao (LSI: Large-Scale Integration) hay rất cao (VLSI: Very Large-Scale Integration), điều này làm tăng tốc độ xử lý lên rất nhiều vì sự trễ tín hiệu do liên kết giữa các mạch số không đáng kể nữa. Logic điều khiển của vi xử lý (CU) điều khiển vi xử lý làm việc với tất cả mạch kết nối với vi xử lý như bộ nhớ, các cổng vào/ra với các thiết bị ngoại vi để trao đổi dữ liệu. Như vậy bộ vi xử lý có những chức năng của một đơn vị xử lý trung tâm (CPU) của một máy tính, đồng thời cũng như CPU, vi xử lý cũng có thể kết nối với bộ nhớ, các thiết bị vào/ra theo kiến trúc Von Neumann hay kiến trúc Havard.

Một hệ thống vi xử lý kết nối theo kiến trúc Von Neumann tối thiểu gồm bộ vi xử lý ghép nối với khối bộ nhớ (ROM, RAM), các khối cổng vào/ra (I/O Ports) với các thiết bị trao đổi tín hiệu số, tương tự/số (A/D) hoặc tín hiệu số/tương tự (D/A), khối tạo nhịp đồng hồ (Clock generator), thông qua các đường tín hiệu của bus hệ thống gồm bus dữ liệu, bus điều khiển và bus địa chỉ thì ta có thể có được một hệ thống vi xử lý (hình 1.6).

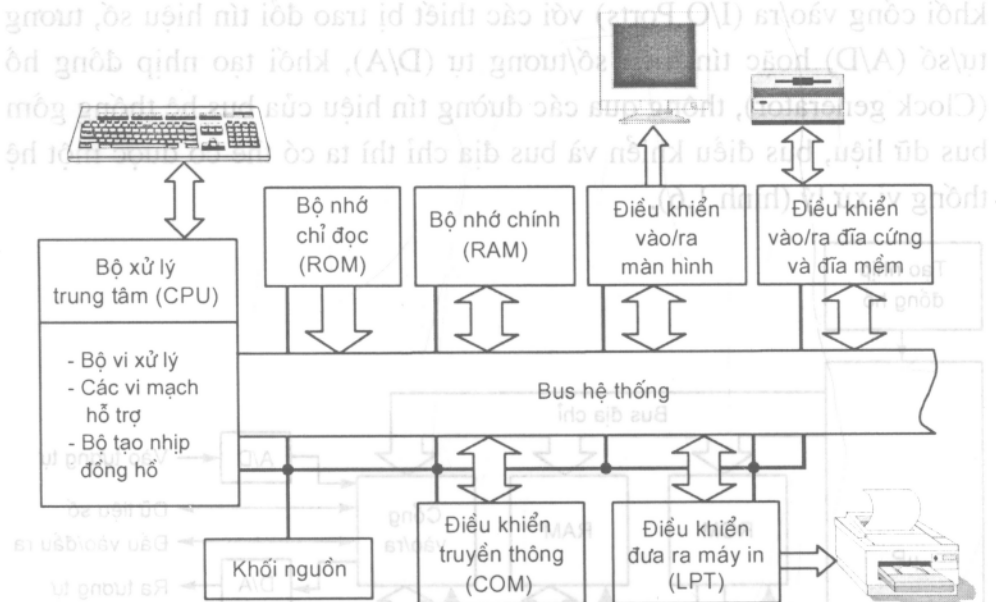


Hình 1.6: Hệ thống vi xử lý kiến trúc Von Neumann cơ bản với ghép nối bộ nhớ (ROM, RAM), và các cổng vào/ra dữ liệu số, tương tự

Do tính dễ thiết kế, mở rộng phát triển và tốc độ không cao nên kiến trúc Von Neumann được áp dụng rộng rãi cho thiết kế các máy vi tính dùng cho cá nhân (máy tính cá nhân) sử dụng đơn vị xử lý trung tâm là bộ vi xử lý. Người sử dụng máy vi tính thường không có yêu cầu khắt khe về tốc độ và hiệu suất xử lý thông tin. Các máy vi tính là ví dụ của hệ thống vi xử lý tối thiểu (hình 1.7).

Để có thể thiết kế được một hệ thống vi xử lý theo kiến trúc Von Neumann, các bộ vi xử lý phải có đường tín hiệu cho bus hệ thống gồm bus dữ liệu, bus địa chỉ và bus điều khiển. Đa số những chip vi xử

lý đa năng đều có bus hệ thống như vậy, ví dụ: Intel 808x, 80x86, Pentiums, Motorola 680x, 680xx,...



Hình 1.7: Máy vi tính cấu hình tối thiểu là một hệ vi xử lý kiến trúc Von Neumann đặc trưng

Bộ vi xử lý kết nối với bộ nhớ chính, với các thiết bị nhớ ngoài như: các thiết bị nhớ trên đĩa từ, đĩa quang, băng từ, và các thiết bị ngoại vi khác như: máy in, màn hình,... thông qua bus hệ thống. Bus hệ thống của máy vi tính cũng bao gồm bus dữ liệu, bus địa chỉ và bus điều khiển (control bus). Bộ nhớ chính của máy vi tính là bộ nhớ bán dẫn. Nó gồm có bộ nhớ truy cập ngẫu nhiên RAM (Random Access Memory) và bộ nhớ chỉ đọc ROM (Read Only Memory) để lưu chương trình điều khiển vào/ra tối thiểu của máy vi tính. Bộ nhớ RAM có dung lượng từ vài trăm kbyte tới vài GB. Bộ nhớ ROM có dung lượng từ vài chục kbyte đến vài MB. Dung lượng nhớ của các thiết bị nhớ ngoài có thể lên tới vài chục GB. Các thiết bị nhớ ngoài và ngoại vi kết nối với CPU để trao đổi thông tin thông qua bus hệ thống bằng những bộ điều khiển vào/ra riêng. Các máy vi tính hiện nay có những

bộ điều khiển vào/ra ngoại vi đa năng cho phép kết nối các loại thiết bị ngoại vi khác nhau. Bộ nhớ chính (Main memory) chứa các dữ liệu và chương trình hoạt động thường trú của hệ thống điều hành, của các ứng dụng, lưu giữ dữ liệu trung gian trong quá trình xử lý. Trên CPU có các vi mạch lập trình hỗ trợ cho vi xử lý để điều khiển ghép nối với các thiết bị ngoại, ví dụ: mạch điều khiển và tạo ra các tín hiệu cho bus hệ thống, mạch điều khiển ngắt, điều khiển truy nhập DMA, thu/phát không đồng bộ đa năng UART (Universal Asynchronous Receiver Transmitter) cho các vào/ra truyền thông (COMM), đếm thời gian (Time counter), giao tiếp vào/ra song song (PPI),... Bàn phím được kết nối truyền thông ngay với bảng mạch chính của máy vi tính. Thiết bị nhớ trên đĩa cứng (HDD), đĩa mềm (FDD), đĩa quang (CD-ROM) là những thiết bị nhớ ngoài cơ bản để lưu trữ thông tin với dung lượng lớn (hệ điều hành máy vi tính, các chương trình ứng dụng, các loại dữ liệu dạng văn bản, hình ảnh, âm thanh số hóa,...).

Có một số loại vi xử lý chuyên dụng, ví dụ, các chip vi điều khiển (Micro controllers) như Intel 8051 lại có phân chia không gian nhớ thành bộ nhớ dữ liệu và chương trình riêng, các cổng vào/ra riêng cho phép thiết kế hệ thống vi xử lý theo kiến trúc Harvard có khả năng xử lý thông tin nhanh và hiệu suất cao. Những loại vi xử lý này được ứng dụng cho thiết kế các hệ thống vi xử lý xử lý tín hiệu trong truyền thông, tự động hóa, điện tử công nghiệp và y tế,...

1.4. NHỮNG ĐẶC ĐIỂM CẤU TRÚC CỦA BỘ VI XỬ LÝ

1.4.1. Công suất của bộ vi xử lý

Công suất (power) của bộ vi xử lý là khả năng xử lý dữ liệu, nó gồm có những đặc điểm sau đây: độ dài từ dữ liệu (data word length), tính bằng số byte; dung lượng nhớ vật lý có thể đánh địa chỉ (addressing capacity); tốc độ xử lý lệnh của bộ vi xử lý (instruction execute speed). Công suất của máy vi tính, hay nói cách khác, tốc độ xử lý thông tin, khả năng lưu trữ thông tin, khả năng kết nối nhiều loại thiết bị ngoại vi... phụ thuộc vào công suất của bộ vi xử lý trong CPU.

- *Độ dài từ:*

Phụ thuộc vào từng thế hệ vi xử lý và mức độ phát triển của công nghệ vi xử lý, độ dài từ có thể là 4-bit, 8-bit (1-byte), 16-bit (2-byte), 32-bit (4-byte), 64-bit (8-byte). Tập lệnh của bộ vi xử lý có các lệnh thực hiện theo từ và theo byte. Nếu một từ là 2-byte, thì cũng phân biệt byte cao (upper byte) và byte thấp (lower byte). Byte thấp chiếm các bit từ 0 đến 7, byte cao chiếm các bit từ 8 đến 15.

Độ rộng từ có độ dài bao nhiêu bit thì cũng có bấy nhiêu bit đối với các thanh ghi, ALU và bus dữ liệu bên trong của bộ vi xử lý. Bus dữ liệu ngoài cũng thường là có chừng đó độ dài, nhưng cũng có thể chỉ một Byte trong khi độ dài từ xử lý bên trong của bộ vi xử lý là 16-bit. Độ dài từ càng lớn càng tạo ra nhiều khả năng tính toán của bộ vi xử lý: khoảng biểu diễn số rộng hơn, tốc độ tính toán nhanh hơn.

- *Khả năng đánh địa chỉ:*

Các từ dữ liệu và lệnh máy lưu trong bộ nhớ tại các ngăn nhớ khác nhau. Mỗi ngăn nhớ phải có địa chỉ nhận biết. Dải đánh địa chỉ càng rộng thì dung lượng bộ nhớ càng lớn. Để đánh địa chỉ, bộ vi xử lý thường có thanh ghi địa chỉ (address register). Độ rộng của thanh ghi địa chỉ quyết định dải địa chỉ của vùng nhớ vật lý mà bộ vi xử lý thỏa mãn. Độ rộng của thanh ghi là 16-bit có thể đánh được địa chỉ khoảng nhớ vật lý là $2^{16} = 2^6 \times 2^{10} = 64 \text{ kbyte} = 65.536 \text{ từ } 8\text{-bit}$. Khái niệm kilô (k) trong kỹ thuật được coi là giá trị 1000, còn chính xác trong máy tính là $2^{10} = 1024 = 1 \text{ k}$. Dung lượng nhớ lớn hơn thì dùng đến các đơn vị như M (Mega): $2^{20} = 1 \text{ M}$, nghĩa là triệu (Million); G (Giga): $2^{30} = 1 \text{ G}$, nghĩa là tỷ (Billion), và hơn nữa, 1 T (Tera) = 2^{40} . Tương ứng với độ dài từ và độ dài thanh ghi địa chỉ, khả năng đánh địa chỉ ở một số bộ vi xử lý cho trong bảng 1.1. Với số mũ ở hệ cơ số hai (nhị phân) ta có thể đánh giá ngay được độ rộng của thanh ghi địa chỉ hay bus địa chỉ, ví dụ để đánh được địa chỉ đến 32 GB, cần phải có 35 đường dây địa chỉ (A0-A34). Khả năng đánh địa chỉ càng lớn thì càng cho phép tạo ra

một hệ thống máy tính có cấu hình mạnh với nhiều loại thiết bị ngoại vi, bộ nhớ chính có dung lượng lớn và khả năng xử lý nhanh.

Bảng 1.1: Độ dài từ và khả năng đánh địa chỉ của một số bộ vi xử lý

Độ dài từ dữ liệu	Khoảng đánh địa chỉ (phụ thuộc độ dài thanh ghi địa chỉ)
4-bit	4096 (4 k) = 2^{12} , 8.196 (8 k) = 2^{14}
8-bit (1-byte)	65.536 (64 k) = 2^{16}
16-bit (2-byte)	32.768 (32 k), 65.536 (64 k), 1.048.576 (1 M) = 2^{20} , 2.097.152 (2 M), 4.194.304 (4 M)
32-bit (4-byte)	4.294.967.296 (4 G) = $2^2 \times 2^{30}$, 34.359.738.367 (32 G) = $2^5 \times 2^{30}$
64-bit (8-byte)	2 G, 32 G, 64 G, 1 T (Tera) = 2^{40}

• *Tốc độ thực hiện lệnh:*

Tốc độ thực hiện lệnh của bộ vi xử lý thường đo bằng tốc độ thực hiện các lệnh dấu phẩy động MFLOPS (Millions of Floating Point Operations Per Second) hoặc tính bằng triệu lệnh/giây (MIPS - Millions of Instructions Per Second). Công thức tính MIPS theo kiến trúc Von NeuMann là:

$$\text{MIPS} = \frac{f \times N}{M + T} \quad (1.1)$$

Trong đó:

f - tần số làm việc của bộ vi xử lý;

N - số lượng các đơn vị xử lý số học và logic (ALU) không phụ thuộc vào nhau bên trong bộ vi xử lý;

M - số lượng vi lệnh (microinstruction) trung bình của một lệnh trong bộ vi xử lý (thường để thực hiện một lệnh vi xử lý cần từ 4 ÷ 7 vi lệnh);

T - hệ số thời gian truy cập bộ nhớ (chu trình chờ đợi trong khi truy cập bộ nhớ).

Theo công thức (1.1), tốc độ thực hiện lệnh của bộ vi xử lý có thể thay đổi nhờ 4 yếu tố. Để nâng cao tốc độ vi xử lý kiến trúc song song, đường ống, đồng xử lý, nhớ dự trữ (cache memory) và bus rộng (wide bus) đã được áp dụng cho các chip vi xử lý công nghệ cao hiện nay.

Metal-Oxide Semiconductor) cho phép đạt được tần số gần 20 MHz nhưng tiêu thụ năng lượng nhiều hơn vì thế các chip vi xử lý loại này rất nhanh nóng. Công nghệ CMOS (Complementary Metal-Oxide Semiconductor) tiêu thụ năng lượng thấp hơn và hoạt động trong dải nhiệt độ rộng, dải tần số nhịp cao hơn NMOS. Vì vậy, CMOS cho phép sản xuất các loại vi xử lý có mật độ tích hợp cao và chúng cũng đắt hơn loại NMOS. Tần số nhịp của các bộ vi xử lý theo CMOS hiện nay đạt đến tới 2 GHz.

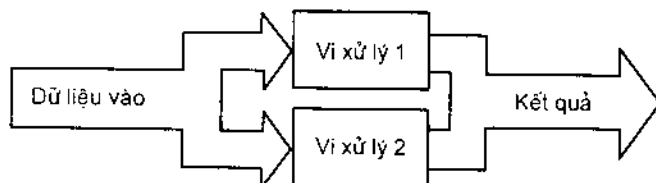
1.4.2. Những đặc tính nâng cao tốc độ của bộ vi xử lý

Một số đặc tính kiến trúc được sử dụng trong chế tạo các bộ vi xử lý nhằm nâng cao tốc độ xử lý là:

- Xử lý song song (parallel processing) và kiến trúc siêu hướng (superscalar);
- Đồng xử lý (coprocessing);
- Kỹ thuật nhớ trung gian Cache (cache memory technique);
- Kỹ thuật đường ống (pipelining technique);
- Bus rộng (wider bus).

• Xử lý song song

Xử lý song song có nghĩa là hai quá trình tính toán cùng xảy ra đồng thời. Trong kiến trúc máy tính, sự kết hợp hai bộ vi xử lý trong khối xử lý trung tâm (CPU) tạo ra khả năng xử lý song song trong cùng một thời gian (hình 1.8).

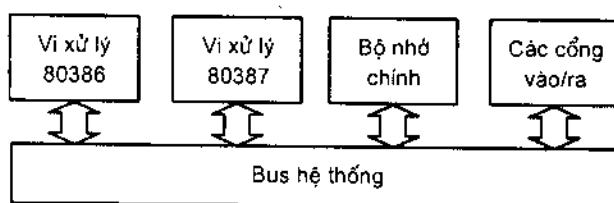


Hình 1.8: Xử lý song song bằng 2 bộ vi xử lý

Kiểu kiến trúc này có thể tạo ra tốc độ xử lý dữ liệu tăng gấp đôi so với kiến trúc chỉ dùng một bộ vi xử lý. Cũng có thể thực hiện xử lý song song ngay bên trong cấu trúc của bộ vi xử lý, bằng cách thiết kế sao cho quá trình xử lý dữ liệu bên trong chip vi xử lý chia thành các phiên khác nhau và thực hiện song song nhờ sự phân chia khối logic điều khiển (CU) bên trong thành hai phần riêng. Kiến trúc của bộ vi xử lý có các khối chức năng xử lý song song bên trong gọi là kiến trúc siêu hướng (superscalar architecture), nghĩa là cùng một lúc có nhiều hướng xử lý khác nhau bên trong vi xử lý. Kiến trúc siêu hướng là sự phát triển tiếp theo của kiến trúc tính toán với tập lệnh giảm thiểu RISC (Reduced Instruction Set Computing). Nó không những nâng cao tốc độ xử lý mà còn nâng cao độ tin cậy của CPU, bởi vì khi có sự cố xảy ra ở một chip vi xử lý (hay ở một đơn vị xử lý bên trong một chip) thì chip vi xử lý còn lại (hay đơn vị xử lý còn lại bên trong một chip) vẫn đảm nhiệm chức năng được bình thường. Hiện nay, các máy tính sử dụng các bộ vi xử lý có kiến trúc RISC và các máy vi tính sử dụng các chip Intel Pentium II, III, IV đều có khả năng xử lý song song bằng nhờ kết nối song song $2 \div 4$ chip vi xử lý cùng loại.

- *Đồng xử lý (coprocessing)*

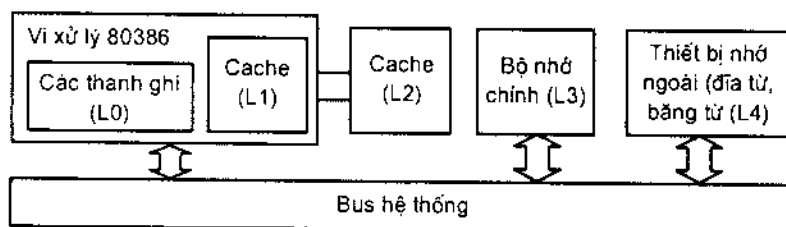
Tương tự như xử lý song song, bộ đồng xử lý là một bộ vi xử lý riêng biệt kết nối với bộ vi xử lý chính thông qua bus hệ thống. Ví dụ như trong họ vi xử lý Intel có các bộ đồng vi xử lý toán học (math coprocessor) 8087, 80287, và 80387 làm việc với 8086/8088, 80286 và 80386 tương ứng (hình 1.9). Bộ đồng xử lý chỉ thực hiện một số chức năng đặc biệt, ví dụ như các phép toán đòi hỏi sự chính xác sử dụng dấu phẩy động. Tốc độ xử lý của bộ đồng xử lý những phép tính này sẽ nhanh hơn rất nhiều so với bộ xử lý chính. Các bộ vi xử lý công nghệ cao hiện nay đã cấy vào bên trong đơn vị xử lý dấu phẩy động (FPU) càng làm tăng tốc độ tính toán các phép tính nhanh và chính xác hơn nhiều.



Hình 1.9: Kết nối đồng xử lý trong máy vi tính

• Bộ nhớ trung gian cache (cache memory)

Các bộ nhớ cache được sử dụng như là bộ nhớ đệm (buffer memory) cho bộ nhớ chính. Các bộ nhớ cache được sử dụng ở các dạng khác nhau để giảm thời gian cần thiết cho bộ vi xử lý truy cập các địa chỉ, các lệnh, hoặc dữ liệu lưu trữ trong bộ nhớ chính. Tức là giảm thời gian T . Biện pháp này được áp dụng trong các vi xử lý kiến trúc RISC và các vi xử lý công nghệ cao. Các vi xử lý kiến trúc RISC chỉ làm việc với các thanh ghi và chỉ có 2 lệnh truy cập bộ nhớ là LOAD và STORE.



Hình 1.10: Hệ thống nhớ trong máy vi tính

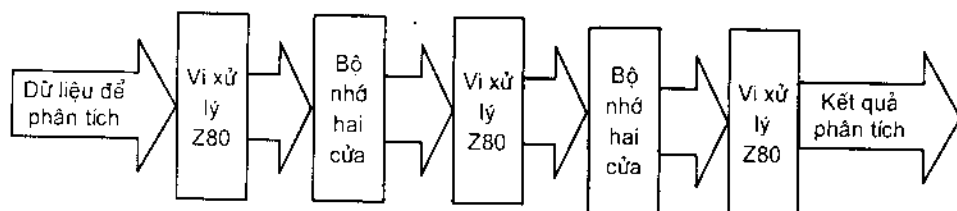
Xây dựng một tổ chức bộ nhớ cho hệ thống vi xử lý sao cho sự trao đổi dữ liệu giữa bộ vi xử lý và bộ nhớ chính, các thiết bị nhớ ngoài đạt hiệu suất cao nhất, tối ưu nhất để giảm được thời gian T . Tổ chức bộ nhớ này có 5 mức (level): mức L0: tập hợp (tập) các thanh ghi bên trong chip vi xử lý, mức L1: bộ nhớ trung gian cache bên trong chip vi xử lý (internal cache), mức L2: bộ nhớ trung gian cache bên ngoài chip vi xử lý (external cache), mức L3: bộ nhớ chính (RAM, ROM), và mức L4: các thiết bị nhớ ngoài. Tổ chức của hệ thống nhớ vi xử lý có thể được mô tả trong hình 1.10.

Cache là bộ nhớ có tốc độ rất cao (thường dùng công nghệ nhớ tĩnh), nó có thể nằm ngay bên trong bộ vi xử lý (L1) với dung lượng hạn chế, hoặc kết nối bên ngoài và trực tiếp với chip vi xử lý (L2) không thông qua bus hệ thống với dung lượng đủ lớn. Trong khi đó bộ nhớ chính kết nối với bộ vi xử lý thông qua bus hệ thống. Sự trao đổi dữ liệu giữa bộ nhớ chính và bộ vi xử lý bị hạn chế về tốc độ, vì vậy để tăng tốc độ xử lý, phải tổ chức làm sao khi thực hiện chương trình, bộ vi xử lý trước hết tìm kiếm lệnh ở cache trước, nếu không có lệnh chứa trong cache thì mới phải tìm tới bộ nhớ chính. Điều này có nghĩa là nếu đa số lệnh không có trong các cache thì tốc độ xử lý chậm hơn gần gấp đôi so với truy nhập thẳng vào bộ nhớ chính. Phải tổ chức sao cho các lệnh có tần suất xuất hiện trong chương trình cao thì có thể cất trong các cache. Tổ chức và dung lượng của các cache phải tối ưu để đảm bảo tỷ lệ được sử dụng (hệ số trúng đích H) cache cao. Các cache được phân ra hai loại: cache chứa các lệnh Icache (Instruction cache) và cache chứa dữ liệu Dcache (Data cache). Hiện tại dung lượng cache bên trong các chip vi xử lý chưa cao (32 kbyte: 16 kbyte Dcache, 16 kbyte Icache). Dung lượng của cache bên ngoài (L2) có thể đạt 256 kbyte, tối đa có thể lên tới $2 \div 4$ MB.

- *Kỹ thuật đường ống (pipelining technique)*

Trong dây chuyền lắp ráp trong công nghiệp chế tạo các sản phẩm như ô tô, điện tử dân dụng, đồ hộp,... tại mọi thời điểm, ở đầu vào dây chuyền các bộ phận của sản phẩm cần phải lắp ráp ban đầu được đưa vào, ở đầu ra của dây chuyền có các sản phẩm hoàn chỉnh xuất hiện và trên từng công đoạn của dây chuyền tiến trình lắp ráp sản phẩm đều xảy ra đồng thời. Hay trong hệ thống đường ống dẫn dầu hoặc dẫn nước, ở mọi thời điểm và mọi đoạn ống luôn có dòng chảy (dầu, nước). Mô phỏng theo dây chuyền lắp ráp máy móc, hệ thống đường ống dẫn, các bộ vi xử lý hiện nay có khả năng thực hiện các lệnh máy liên tục theo một dây chuyền với 5 công đoạn: nhập dữ liệu của lệnh từ bộ nhớ, giải mã lệnh, thực hiện các lệnh, ghi kết quả thực hiện lệnh vào bộ nhớ. Khi lệnh thứ nhất bắt đầu bước vào thực hiện ở

giai đoạn hai thì mã lệnh của lệnh tiếp theo được đọc từ bộ nhớ ra để thực hiện bước một (giải mã lệnh). Cứ như vậy các lệnh được thực hiện theo một dây chuyền liên tục như là dòng nước đi trong đường ống. Tốc độ xử lý lệnh vì thế được tăng lên rất cao. Hình 1.11 là một ví dụ hệ thống tính toán phân tích dùng cấu trúc đường ống với 3 chip vi xử lý Z80, kết nối theo dây chuyền tính toán liên tục.



Hình 1.11: Cấu trúc đường ống của hệ thống phân tích

- **Bus rộng**

Kỹ thuật bus rộng áp dụng cho cả bên trong lẫn bên ngoài bộ vi xử lý. Thực chất của giải pháp này là bên trong bộ vi xử lý, thanh tích lũy có độ dài gấp đôi bus, như vậy tốc độ tính toán sẽ nhanh hơn, bởi vì nó hạn chế sự cần thiết phải thực hiện các phép truy nhập với bộ nhớ để lưu giữ các kết quả trung gian của các phép tính.

BÀI TẬP ÔN LUYỆN

1. Nêu tóm tắt trao đổi thông tin bên trong vi xử lý?
2. Phân biệt máy tính kiến trúc Von Neumann và Harvard?
3. Thế nào là vi xử lý? Phân biệt vi xử lý và CPU, vi xử lý và máy vi tính?
4. Đặc điểm công nghệ vi xử lý NMOS và CMOS?
5. Đặc điểm công suất của vi xử lý?
6. Những đặc tính nâng cao tốc độ vi xử lý?
7. Chức năng của các thanh ghi bên trong vi xử lý?
8. Các kiểu đánh địa chỉ của vi xử lý?
9. Bộ vi xử lý không bao gồm các mạch nào sau đây:
 - a: Logic;
 - b: Tính toán;
 - c: Nhớ;
 - d: Tất cả.
10. Chu kỳ đọc lệnh/thực hiện lệnh của vi xử lý được sử dụng để thực hiện:
 - a: Thao tác logic;
 - b: Các lệnh;
 - c: Thao tác số học;
 - d: ALU.
11. ALU được sử dụng để thực hiện:
 - a: Phép cộng;
 - b: Phép chuyển dữ liệu;
 - c: Đọc lệnh;
 - d: Tất cả a, b, c.

12. Bộ vi xử lý tạo ra các tín hiệu để điều khiển các mạch:

- a: Nhớ;
- b: Vào (Input);
- c: Ra (Output);
- d: Tất cả a, b, c.

13. Hệ thống vi xử lý có ít nhất các mạch:

- a: Bộ nhớ (ROM hoặc RAM);
- b: Input/Output;
- c: CPU;
- d: Tất cả a, b, c.

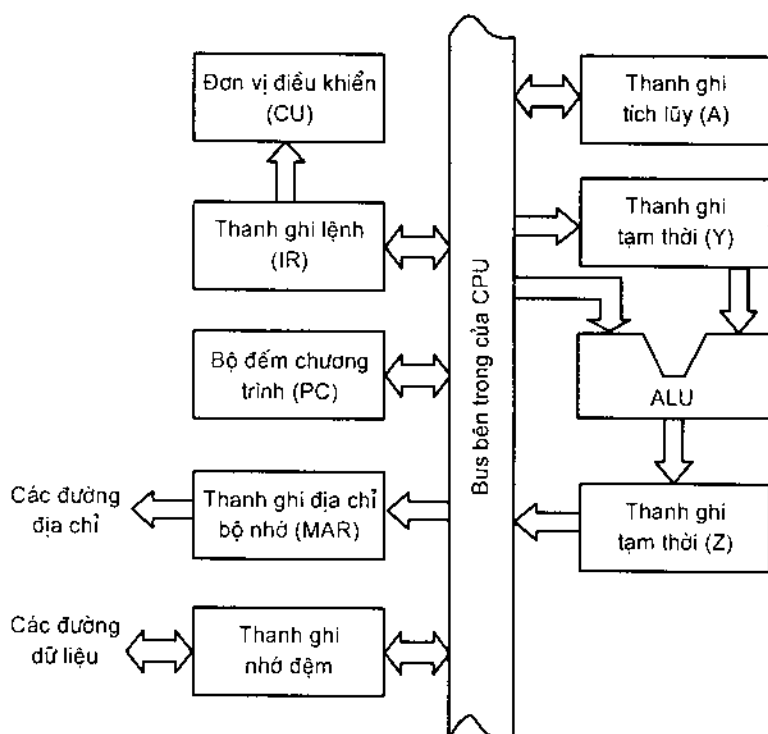
Chương 2

CẤU TRÚC VÀ HOẠT ĐỘNG CỦA VI XỬ LÝ

2.1. SƠ ĐỒ CẤU TRÚC TỔNG QUÁT CỦA VI XỬ LÝ

2.1.1. Cấu trúc bên trong của đơn vị xử lý trung tâm

Hình 2.1 là một cấu trúc tổng quát bên trong một đơn vị xử lý trung tâm của một máy tính có kiến trúc Von Neumann.



Hình 2.1: Cấu trúc bên trong của một đơn vị xử lý trung tâm

Đơn vị xử lý trung tâm (CPU) của một máy tính thường gồm có các thanh ghi, đơn vị số học-logic (ALU) và đơn vị điều khiển (CU). Các thành phần bên trong CPU kết nối thông tin với nhau thông qua bus trong.

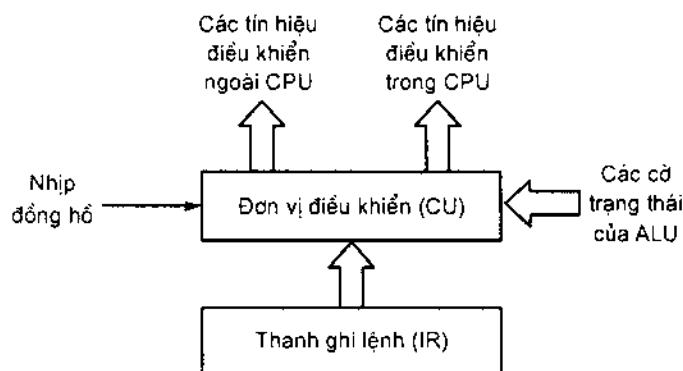
Đơn vị điều khiển (CU) thực hiện:

1. Đưa ra ngoài CPU các tín hiệu điều khiển để tác động đến các trao đổi dữ liệu với bộ nhớ và vào/ra;
2. Đưa ra các tín hiệu điều khiển bên trong CPU để vận chuyển dữ liệu giữa các thanh ghi và tác động ALU thực hiện các nhiệm vụ cụ thể.

Đơn vị điều khiển thực hiện những tác động này nhờ:

1. Diễn giải được (giải mã) các mã lệnh (Op-codes);
2. Định tuần tự các vi thao tác;
3. Thực hiện các vi thao tác;
4. Ra các quyết định dựa trên các cờ trạng thái của ALU.

Như vậy, đơn vị điều khiển có các đầu vào và đầu ra như sau (hình 2.2):



Hình 2.2: Các đầu vào và ra của đơn vị điều khiển

Nếu có một lệnh cộng nội dung thanh ghi tích lũy A với nội dung một ngăn nhớ có địa chỉ là 107Ah: ADD A, 107Ah thì quá trình diễn ra trong CPU như sau:

Cho rằng lệnh ADD A, 107Ah đã được đọc từ bộ về CPU. Khi đó các thao tác tiếp theo sẽ là:

1. Chuyển địa chỉ (107Ah) từ thanh ghi lệnh (IR) đến thanh ghi địa chỉ bộ nhớ (MAR) ra các đường địa chỉ;
2. Đọc nội dung của ngăn nhớ có địa chỉ 107Ah về thanh ghi nhớ đệm (MBR) qua các đường dữ liệu;
3. Chuyển nội dung của MBR đến thanh ghi tạm thời Y qua bus bên trong của CPU;
4. Cộng các nội dung của thanh ghi tích lũy A và của thanh ghi tạm thời Y nhờ ALU và cất kết quả phép cộng vào thanh ghi tạm thời Z;
5. Chuyển kết quả từ thanh ghi tạm thời Z tới thanh ghi tích lũy A qua bus trong của CPU.

Quá trình này được điều khiển và định thời nhờ đơn vị điều khiển của CPU.

Bộ vi xử cũng là một CPU của một máy vi tính, nó cũng có cấu trúc bên trong với đầy đủ các khối chức năng của một CPU.

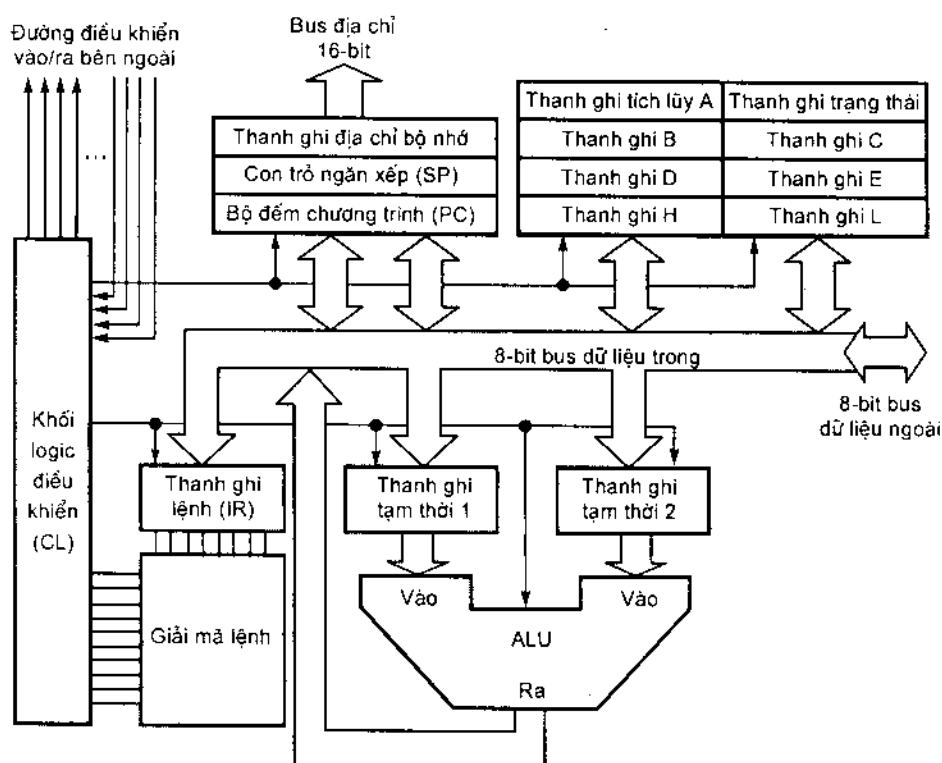
2.1.2. Sơ đồ cấu trúc tổng quát của bộ vi xử lý

Cấu trúc chung của bộ vi xử lý có những thành phần cơ bản, đó là: các thanh ghi, ALU, đơn vị giao tiếp bus (BIU), CU và tập lệnh. BIU chịu trách nhiệm điều khiển các bus địa chỉ và dữ liệu khi truy cập bộ nhớ chính và dữ liệu trong bộ nhớ cache. Các thế hệ vi xử lý 8-bit của Intel, Motorola, Zilog đã một thời làm mẫu để tìm hiểu cấu trúc và hoạt động chung của các loại vi xử lý. Các thế hệ vi xử lý 16-bit và 32-bit sau này vẫn kế thừa những chức năng cơ bản nhất trong cấu trúc của vi xử lý 8-bit. Các chương trình viết cho thế hệ vi xử lý 8-bit vẫn có thể chạy trên các hệ thống vi xử lý 16-bit và 32-bit của cùng một hãng sản xuất.

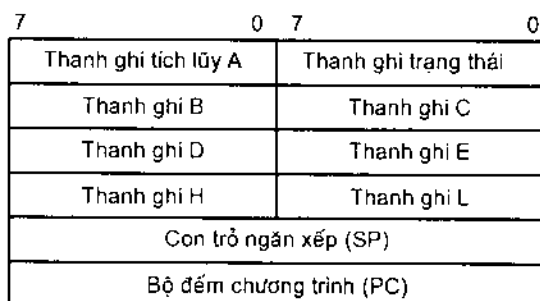
Sơ đồ khối cấu trúc tổng quát của bộ vi xử lý 8-bit trình bày ở hình 2.3.

Sơ đồ khối không mô tả chi tiết phần cứng mà chỉ mô tả các chức năng logic để xử lý và điều khiển dữ liệu. Nó cũng chỉ ra từng chức

năng logic kết nối với các chức năng khác như thế nào. Ngoài sơ đồ khối, cần có thêm mô hình lập trình (programming model) của vi xử lý (hình 2.4), trong đó chỉ ra những thành phần là những thanh ghi của vi xử lý có thể được người lập trình can thiệp vào để ghi/đọc nội dung của chúng.



Hình 2.3: Sơ đồ khối cấu trúc tổng quát của vi xử lý 8-bit



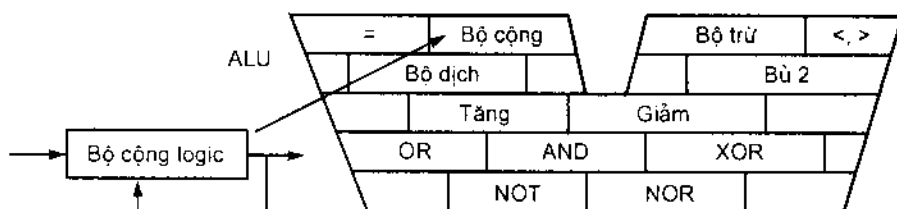
Hình 2.4: Mô hình lập trình của vi xử lý 8-bit

Sơ đồ khối giúp ta dễ dàng hiểu kiến trúc của vi xử lý. Mô hình lập trình giúp ta dễ dàng hiểu vi xử lý trong môi trường lập trình.

2.2. CHỨC NĂNG VÀ HOẠT ĐỘNG CỦA CÁC THÀNH PHẦN BÊN TRONG VI XỬ LÝ

2.2.1. ALU

Đơn vị ALU là một trong những thành phần chính trong vi xử lý. ALU chứa các khối logic thực hiện các phép tính số học và logic. Các khối logic này có tên gọi tương ứng như: bộ cộng (Adder), bộ trừ (Subtractor), bộ dịch (Shifter), các so sánh ($=$, $<$, $>$), bù 2 (2s Complementer), tăng (Incrementer), giảm (Decrementer), hoặc (OR), và (AND), cộng module 2 (XOR), đảo (NOT), hoặc đảo (NOR),... (hình 2.5).



Hình 2.5: Các khối logic bên trong ALU

ALU của đa số các loại vi xử lý có thể thực hiện các chức năng sau:

ADD (cộng)	2s Complement (bù nhị phân)
SUB (trừ)	Shift Right (dịch phải)
AND (Và logic)	Shift Left (dịch trái)
OR (Hoặc logic)	Increment (tăng)
XOR (Cộng module 2)	Decrement (giảm)

Phép nhân và chia số học được thực hiện nhờ kết hợp các phép cộng, trừ, dịch trái và dịch phải.

ALU có hai cổng vào (IN) và một cổng ra (OUT). Các cổng IN để nhận từ dữ liệu vào ALU. Cổng OUT để lấy kết quả xử lý dữ liệu

của ALU ra ngoài. Các cổng IN được nhớ đệm bằng các thanh ghi tạm thời 1 và 2. Những thanh ghi này làm nhiệm vụ nhận dữ liệu từ các nơi khác nhau bên trong bộ vi xử lý thông qua bus dữ liệu bên trong và lưu giữ tạm thời dữ liệu cho hai cổng IN trong quá trình xử lý dữ liệu trong ALU. Tương tự, cổng OUT kết nối với bus dữ liệu bên trong, do đó kết quả phép toán có thể được đưa tới các nơi khác nhau. Trong hầu hết các loại vi xử lý có thể có hơn một cổng IN và một cổng OUT của ALU.

Thường xuyên, ALU nhận dữ liệu từ thanh ghi tích lũy A (Accumulator A). Thanh ghi tích lũy A là một trong những thanh ghi dùng chung và có độ dài 8-bit (vi xử lý 8-bit). Trong các xử lý, thường xuyên dữ liệu được gửi đến thanh tích A thông qua bus dữ liệu bên trong. Ví dụ, khi ALU thực hiện phép tính cộng hai từ dữ liệu (2 toán hạng), một trong hai từ được đặt trong thanh ghi tích lũy, từ thứ hai nằm ở thanh ghi nào đó hay được chuyển từ một ngăn nhớ bên ngoài bộ vi xử lý. Sau khi cộng xong, từ kết quả được chuyển đến thanh ghi tích lũy và lưu giữ tại đó. ALU có thể thực hiện các phép tính với một từ hay hai từ, phụ thuộc vào loại phép tính. Với các phép tính với hai từ thì cả hai cổng IN của ALU đều được sử dụng. Có một số lệnh chỉ có một từ, ví dụ: lệnh lấy mã bù nhị phân, thì chỉ cần sử dụng một cổng vào của ALU.

2.2.2. Các thanh ghi của vi xử lý

Các thanh ghi bên trong vi xử lý có những chức năng khác nhau. Một số thanh ghi có thể dùng cho nhiều mục đích, đa năng. Chúng được gọi là những thanh ghi đa năng (general-purpose registers). Một số khác lại có những nhiệm vụ riêng. Trong hầu hết các loại vi xử lý đều có các thanh ghi cơ bản, đó là: thanh ghi tích lũy (A), con trỏ ngăn xếp (SP), thanh ghi trạng thái (SR), thanh ghi địa chỉ bộ nhớ (MAR: Memory Address Register), thanh ghi lệnh (IR), bộ đếm chương trình (PC), các thanh ghi đa năng, và các thanh ghi tạm thời. Không phải tất cả các thanh ghi bên trong vi xử lý được trình bày trong sơ đồ tổng

quát, bởi vì có một số thanh ghi người lập trình không thể can thiệp vào được. Tất cả những thanh ghi mà người lập trình có thể can thiệp vào được (ghi/đọc được) thường được trình bày trong mô hình lập trình của vi xử lý.

• *Thanh ghi tích lũy A (Accumulator)*

Thanh ghi tích lũy A tham gia vào phần lớn các phép tính. Nó cất giữ toán hạng (operand) hoặc các kết quả phép tính của ALU. Các phép tính số học và logic phải sử dụng ALU và thanh ghi tích lũy A. Vì vậy, thanh ghi tích lũy A là một thanh ghi chính của vi xử lý, và là một thành phần trong mô hình lập trình của vi xử lý. Độ dài từ xử lý của vi xử lý phụ thuộc vào độ dài của thanh ghi tích lũy. Cho nên, thanh ghi tích lũy của các loại vi xử lý 8-bit có độ dài 8-bit (1 byte). Tuy nhiên, ở một số loại vi xử lý có thể có những thanh ghi tích lũy có độ dài gấp đôi độ dài từ xử lý của vi xử lý. Chúng được gọi là thanh ghi tích lũy độ dài kép. Thậm chí một số loại vi xử lý còn có một số thanh ghi tích lũy, ví dụ một thanh ghi tích lũy là A, thanh ghi tích lũy khác khác gọi là B. Trong trường hợp này phải có lệnh chỉ cho bộ vi xử lý chuyển kết quả tính của ALU vào thanh ghi tích lũy A, và một lệnh chuyển dữ liệu vào thanh ghi tích lũy B. Tương tự, có lệnh xóa thanh ghi tích lũy A và lệnh xóa thanh ghi tích lũy B.

Ví dụ một phép cộng hai toán hạng: $C = A + B$ được thanh ghi tích lũy A thực hiện như sau:

1. Toán hạng A được đưa vào thanh ghi tích lũy A;
2. Toán hạng B từ một ngăn nhớ bên ngoài (hoặc từ một thanh ghi khác) được cộng với toán hạng chứa trong thanh ghi tích lũy A;
3. Kết quả phép cộng $C = A + B$ nằm lại thanh ghi tích lũy A (toán hạng A bị kết quả C thay thế trong thanh ghi tích lũy A).

Như vậy, với đa số các phép tính số học, thanh ghi tích lũy ban đầu chứa một toán hạng và sau đó chứa kết quả phép tính. Kết quả

trong thanh ghi tích lũy A có thể được chuyển đến thanh ghi khác hoặc đưa ra một ngõ nhớ ở bộ nhớ bên ngoài vi xử lý, hoặc ra một cổng vào/ra dữ liệu nào đó.

Những lệnh vào/ra (I/O) với ngoại vi thường là những lệnh trao đổi byte dữ liệu giữa thanh ghi tích lũy A với các thanh ghi của điều khiển ngoại vi thông qua bus hệ thống và bus dữ liệu bên trong của vi xử lý. Ví dụ, lệnh: IN port, là lệnh đọc dữ liệu từ cổng (port) của ngoại vi (địa chỉ port) vào thanh ghi tích lũy A, và OUT port - đọc nội dung của thanh ghi tích lũy A ra cổng ngoại vi.

- *Bộ đếm chương trình (PC: Program Counter)*

Bộ đếm chương trình là một trong những thanh ghi quan trọng nhất trong vi xử lý. Chương trình là tập hợp liên tục các chỉ dẫn (instructions) lưu giữ trong bộ nhớ bên ngoài vi xử lý. Mỗi một chỉ dẫn (lệnh máy) trong chương trình phải được vi xử lý thực hiện theo một thứ tự mong muốn của người lập trình. Bộ đếm chương trình có nhiệm vụ gìn giữ tiến trình thực hiện chương trình theo mong muốn đó. Trong tiến trình thực hiện chương trình, bộ đếm chương trình luôn chứa địa chỉ ngõ nhớ lưu giữ chỉ dẫn tiếp theo mà vi xử lý phải thực hiện.

Bộ đếm chương trình thường có độ dài lớn hơn độ dài từ mà vi xử lý thực hiện. Đối với vi xử lý 8-bit, bộ đếm chương trình có độ dài 16-bit. Và vì nó dùng để chứa địa chỉ ngõ nhớ nên không gian mà nó có thể địa chỉ được là 64 kbyte từ 8-bit, tức là một bộ nhớ với dung lượng $64 \text{ kbyte} = 2^{16}$ (hay 65.536). Chỉ có một số ít chỉ dẫn làm việc với bộ đếm chương trình. Trước khi vi xử lý có thể thực hiện một chương trình, bộ đếm chương trình phải được nạp một giá trị. Giá trị này là địa chỉ của ngõ nhớ chứa lệnh đầu tiên phải thực hiện trong chương trình. Thường thì giá trị này là toàn các bit 0, hay toàn các bit 1. Toàn các bit 0 nghĩa là địa chỉ ngõ nhớ đầu tiên của bộ nhớ (0000h). Toàn các bit 1 nghĩa là địa chỉ của ngõ nhớ cuối cùng của bộ nhớ (FFFFh). Trong ngõ nhớ này phải chứa nội dung của lệnh đầu tiên

của chương trình. Tất cả những ngăn nhớ còn lại của chương trình có thể chứa lệnh hay dữ liệu là tùy thuộc vào cấu trúc chương trình.

Địa chỉ của ngăn nhớ chứa lệnh đầu tiên của chương trình được chuyển từ bộ đếm chương trình đến thanh ghi địa chỉ bộ nhớ. Thanh ghi địa chỉ bộ nhớ cũng phải có độ dài 16-bit. Lúc này, nội dung của bộ đếm chương trình và thanh ghi địa chỉ bộ nhớ là giống nhau. Qua 16-bit bus địa chỉ thanh ghi địa chỉ bộ nhớ gửi địa chỉ ngăn nhớ chứa lệnh đầu tiên của chương trình ra các mạch nhớ kết nối trên bus địa chỉ. Sau đó bộ nhớ gửi lên bus dữ liệu 8-bit nội dung của ngăn nhớ được chọn, đó chính là nội dung của mã lệnh đầu tiên của chương trình. Thông qua bus dữ liệu 8-bit của hệ thống và bus dữ liệu 8-bit bên trong vi xử lý nội dung của mã lệnh đầu tiên được đưa vào thanh ghi lệnh (IR). Thanh ghi lệnh cũng phải có độ dài 8-bit để chứa đủ nội dung của mã lệnh. Ngay sau khi có nội dung của mã lệnh, khối logic giải mã lệnh thực hiện giải mã lệnh. Kết quả của giải mã lệnh được khối logic điều khiển tạo ra các tín hiệu điều khiển giai đoạn thực hiện lệnh.

Quá trình đọc lệnh từ bộ nhớ vào thanh ghi trong vi xử lý lệnh gọi là giai đoạn đọc lệnh (Instruction fetch). Tiếp sau là các giai đoạn giải mã lệnh, thực hiện lệnh. Một khi lệnh đầu tiên đã được đọc từ bộ nhớ vào thanh ghi lệnh và vi xử lý bắt đầu thực hiện chỉ dẫn đầu tiên, vi xử lý tự động tăng nội dung bộ đếm chương trình lên (Increment), như vậy bộ đếm chương trình có giá trị mới là địa chỉ của ngăn nhớ chứa lệnh tiếp theo phải thực hiện (địa chỉ của lệnh tiếp theo). Đây là điều quan trọng cần ghi nhớ, bởi vì sẽ có trường hợp, ví dụ, trong chương trình gặp phải một lệnh gọi (CALL) tới một chương trình con, khi đó lệnh tiếp theo phải thực hiện là lệnh đầu tiên trong chương trình con, tức là bộ đếm chương trình phải chứa địa chỉ của ngăn nhớ lưu nội dung của lệnh đầu tiên của chương trình con. Khi vi xử lý bắt đầu thực hiện lệnh cuối cùng của chương trình con thì bộ đếm chương trình lại phải có địa chỉ của ngăn nhớ chứa lệnh tiếp theo lệnh CALL trong

thứ tự sắp xếp của chương trình chính lưu trong bộ nhớ. Không những thế các lệnh nhảy vô điều kiện (JMP), có điều kiện, cũng làm thay đổi thứ tự thực hiện tuần tự các lệnh sắp xếp trong chương trình được lưu trong bộ nhớ.

Thanh ghi trạng thái của vi xử lý còn được gọi là thanh ghi cờ (flag register), và dùng để ghi kết quả của các lệnh kiểm tra, so sánh khi thực hiện chương trình. Một số phép tính thực hiện với ALU và với các thanh ghi có thể thiết lập hoặc xóa một số bit trong thanh ghi trạng thái. Các bit trong thanh ghi trạng thái còn gọi các bit cờ. Mỗi một bit của nó có một ý nghĩa và bị tác động tùy theo lệnh máy thực hiện. Sử dụng các bit của thanh ghi trạng thái có thể thực hiện rẽ nhánh chương trình bằng các lệnh nhảy có điều kiện (nhảy theo giá trị các bit cờ), và bộ đếm chương trình phải được nạp địa chỉ của đích nhảy tới. Sự rẽ nhánh được thực hiện nếu giá trị cờ thỏa mãn. Chức năng của các bit trong thanh ghi trạng thái được giải thích trong hình 2.6.

• *Thanh ghi trạng thái*

Z	N	C	I	IF	O	P	1
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Bit 0: "1" logic 1. Không dùng.

Bit 1: P: Parity; chẵn lẻ. P = 1 khi kết quả của phép toán để lại một thanh ghi chỉ ra có số các bit = 1 là lẻ.

Bit 2: O: Overflow; tràn. O = 1 khi cả nhớ số học và nhớ dấu xuất hiện trong các phép toán với số bù 2 (Khi phép tính các số nhị phân có tràn).

Bit 3: IF: Interrupt flag; cờ ngắt. IF = 1 khi chương trình quyết định cho phép chức năng ngắt. IF = 0 khi chương trình không cho phép chức năng ngắt.
Nếu vi xử lý có nhiều ngắt thì có nhiều bit IF.

Bit 4: I: Intermediate carry; nhớ trung gian. I = 1 khi cộng 2 số 4 bit tạo ra nhớ cho bit thứ 5. Bit này thường gọi là nhớ một nửa, và dùng khi chuyển đổi các số BCD thành số nhị phân.

Bit 5: C: Carry bit; bit nhớ: C = 1 khi 2 số nhị phân cộng với nhau tạo ra nhớ từ bit thứ 8 hoặc có vay mượn (borrow) khi một số nhỏ hơn trừ một số lớn hơn.

Bit 6: N: Negative; âm. N = 1 (bit cao nhất) của giá trị là số âm trong phép bù 2.

Bit 7: Z: Zero bit. Z = 1 khi phép tính gây ra tất cả các bit của thanh ghi kết quả = 0.

Hình 2.6: Thanh ghi trạng thái của vi xử lý

Có thể lấy ví dụ phép cộng hai số 8-bit ảnh hưởng đến giá trị các bit cờ như thế nào:

+ Trường hợp 1: $a = 1110\ 1110$, $b = 0111\ 0000$.

$$\begin{array}{r} 0110\ 1110 \\ + 0111\ 0000 \\ \hline 1 \leftarrow 0101\ 1110 \\ \hline \end{array}$$

Nhớ 8-bit kết quả > 0

Giá trị nhớ 1 của phép tính từ lần cộng các bit thứ 8 của hai số thiết lập bit $C = 1$ của thanh ghi trạng thái.

+ Trường hợp 2: $a = 0001\ 1111$, $b = 0100\ 0001$.

$$\begin{array}{r} 0001\ 1111 \\ + 0100\ 0001 \\ \hline 0 \leftarrow 0110\ 0000 \\ \hline \end{array}$$

Nhớ 8-bit kết quả > 0

Phép tính không có nhớ từ lần cộng các bit thứ 8 của hai số, do đó bit C của thanh ghi trạng thái vẫn giữ $= 0$.

+ Trường hợp 3: $a = 1110\ 1110$, $b = 1111\ 0000$.

$$\begin{array}{r} 1110\ 1110 \\ + 1111\ 0000 \\ \hline 1 \leftarrow 1101\ 1110 \\ \hline \end{array}$$

Nhớ 8-bit kết quả < 0

Giá trị nhớ 1 của phép tính từ lần cộng các bit thứ 8 của hai số thiết lập bit $C = 1$, đồng thời kết quả cộng là số âm thiết lập bit $N = 1$ của thanh ghi trạng thái.

+ Trường hợp 4: $a = 1110\ 1110$, $b = 0001\ 0001$.

$$\begin{array}{r} 1110\ 1110 \\ + 0001\ 0001 \\ \hline 1 \leftarrow 0000\ 0000 \\ \hline \end{array}$$

Nhớ 8-bit kết quả $= 0$

Giá trị nhớ 1 của phép tính từ lần cộng các bit thứ 8 của hai số thiết lập bit C (carry) = 1, đồng thời 8 bit kết quả = 0 thiết lập bit Z = 1 của thanh ghi trạng thái.

Trong tập lệnh của vi xử lý các lệnh đều được mô tả với tác động đến các giá trị của các bit của thanh ghi trạng thái, nhờ đó người lập trình có thể thực hiện kiểm tra các giá trị các bit trạng thái và rẽ nhánh chương trình. Nội dung của thanh ghi trạng thái có thể được đọc ra nhờ chương trình thông qua bus dữ liệu bên trong, nhưng nó không thể nhận dữ liệu từ bus dữ liệu bên trong, do đó, thanh ghi ghi trạng thái chỉ có thể đọc được mà không ghi được bởi lập trình.

- *Con trỏ ngăn xếp SP (Stack Pointer)*

Ngăn xếp, đó là bộ nhớ có cơ chế truy cập theo kiểu LIFO (Last-In-First-Out), nghĩa là byte dữ liệu nào ghi vào ngăn xếp cuối cùng thì sẽ được đọc ra đầu tiên. Ngăn xếp có thể là một vùng nào đó của bộ nhớ chính, hay là một mảng thanh ghi riêng biệt. Nó làm nhiệm vụ lưu giữ những thông tin phải dùng đi dùng lại nhiều lần trong quá trình thực hiện chương trình, đặc biệt là khi có nhiều chương trình con.

Giống như bộ đếm chương trình, con trỏ ngăn xếp phải có độ dài đủ để chứa giá trị địa chỉ của một ngăn nhớ trong bộ nhớ. Trong vi xử lý 16-bit nó có độ dài 16-bit. Nội dung của con trỏ ngăn xếp là địa chỉ trỏ đến vùng nhớ tiếp theo. Ngăn xếp luôn được truy cập từ đỉnh (TOP). Trong hầu hết các loại vi xử lý, con trỏ ngăn xếp giảm địa chỉ tiếp theo thấp hơn sau khi nó được sử dụng. Điều này cho phép người lập trình xây dựng ngăn xếp từ đỉnh xuống đáy trong bộ nhớ. Các lệnh thao tác với ngăn xếp là các lệnh hai byte. Có nghĩa là nội dung của con trỏ ngăn xếp giảm đi 2 sau mỗi lần được sử dụng: (SP-1), (SP-2). Nội dung của con trỏ ngăn xếp được người lập trình khởi tạo ban đầu. Quá trình gọi là khởi tạo con trỏ ngăn xếp. Nội dung ban đầu của con trỏ ngăn xếp luôn trỏ đến đỉnh ngăn xếp. Nếu không khởi tạo con trỏ ngăn xếp thì ngăn xếp có thể bắt đầu ở một vùng bất kỳ trong bộ nhớ chính. Nhưng ngăn xếp không được khởi tạo ban đầu có thể gây ra sự

dề lên những dữ liệu quan trọng, hoặc ghi lên chính chương trình, hoặc ghi vào vùng không thuộc RAM. Trong những trường hợp như vậy có thể làm hỏng nội dung dữ liệu của ngăn xếp, làm hỏng chương trình thực hiện.

Các lệnh cất giữ dữ liệu vào ngăn xếp như: CALL (gọi tới chương trình con), PUSH (đẩy dữ liệu vào ngăn xếp), RST p (yêu cầu ngắt) kéo theo sự giảm nội dung của con trỏ ngăn xếp. Các lệnh phục hồi dữ liệu cất giữ trong ngăn xếp như: RET (trở về từ chương trình con), POP (phục hồi từ ngăn xếp) kéo theo tăng nội dung con trỏ ngăn xếp (SP+1). Khi khởi động hệ thống máy tính, con trỏ ngăn xếp luôn được khởi tạo về địa chỉ đỉnh của ngăn xếp.

- *Các thanh ghi đa năng*

Vi xử 8-bit có 6 thanh đa năng (B, C, D, E, H, L). Cũng giống như thanh ghi tích lũy, đây là những thanh ghi 8-bit. Các loại vi xử khác nhau có thể có nhiều thanh ghi đa năng hơn, nhiều ứng dụng hơn. Các thanh ghi đa năng có thể được sử dụng cho nhiều mục đích, và nhiều lệnh sử dụng các thanh ghi này: chứa các toán hạng để thực hiện các phép tính số học, logic, dịch,... Có thể thực hiện các lệnh xử lý các bit trong từng thanh ghi đa năng. Các thanh ghi đa năng có thể ghép lại thành từng đôi. Ví dụ, các cặp thanh ghi BC, DE, và HL có chức năng thống nhất: chúng có thể được thao tác như là các thanh ghi 16-bit riêng biệt. Điều này cho phép thực hiện các lệnh với các từ dữ liệu 16-bit. Cặp thanh ghi HL có thể được dùng để chứa địa chỉ ngăn nhớ, và nội dung của HL có thể được tăng lên 1 (increment) hoặc giảm đi 1 (decrement) để trở tới ngăn nhớ tiếp theo.

- *Thanh ghi địa chỉ bộ nhớ và logic*

Thanh ghi địa chỉ bộ nhớ (MAR: Memory Address Register) có nội dung là địa chỉ ngăn nhớ mà trong ngăn nhớ đó chứa lệnh hay dữ liệu mà vi xử lý cần đến hoặc phải ghi dữ liệu vào đó. Vì là địa chỉ bộ nhớ nên thanh ghi địa chỉ bộ nhớ phải có độ dài 16-bit. Các đường ra của thanh ghi địa chỉ bộ nhớ được điều khiển nối với bus địa chỉ 16-bit

của hệ thống, qua đó giá trị nội dung của thanh ghi được dẫn đến các mạch logic giải mã chọn địa chỉ ngăn nhớ của bộ nhớ hay các cổng vào/ra kết nối với các thiết bị ngoại vi. Logic giải mã chọn địa chỉ bộ nhớ và các cổng vào/ra có thể khác nhau tùy thuộc vào loại vi xử lý, bộ nhớ, và các thiết bị ngoại vi kết nối.

Trong chu kỳ đọc lệnh (instruction fetch), một lệnh máy được đọc từ bộ nhớ. Nội dung của MAR và nội dung của PC lúc này là như nhau. Nghĩa là, MAR trỏ tới ngăn nhớ chứa lệnh đầu tiên của chương trình đang được đọc từ bộ nhớ. Một khi lệnh được đọc từ bộ nhớ ra và được giải mã thì bộ đếm chương trình tự động tăng lên. Nhưng lúc này nội dung của MAR không tăng. Sau khi giải mã lệnh xong, bắt đầu chu kỳ thực hiện lệnh. Trong chu kỳ thực hiện lệnh nội dung của thanh ghi địa chỉ bộ nhớ phụ thuộc lệnh đang thực hiện. Nếu lệnh đó yêu cầu MAR một lần nữa để trỏ đến ngăn nhớ thì lần thứ hai, MAR được dùng cho một lệnh. Một số lệnh không cần tới địa chỉ ngăn nhớ, ví dụ lệnh xóa thanh ghi tích lũy, thì thanh ghi địa chỉ bộ nhớ chỉ dùng một lần cho đọc lệnh. Trong hầu hết các họ vi xử lý 8-bit, MAR có độ dài 16-bit. Phụ thuộc vào các loại vi xử lý, nó có thể có độ dài: 16, 32 và 64-bit.

MAR 16-bit có thể chia thành hai byte: byte cao và byte thấp, kết nối độc lập với 8-bit bus dữ liệu bên trong để có thể nhận 8-bit dữ liệu độc lập. MAR có thể nhận dữ liệu từ một số nguồn. Phần lớn các loại vi xử lý có các lệnh nạp MAR từ PC, từ các thanh ghi đa năng, từ SP, hoặc từ bộ nhớ. Có một số lệnh cho phép thiết lập giá trị mới cho MAR: giá trị mới này được tính bởi phép cộng (hoặc trừ) nội dung của PC với một số nguyên bù thêm (offset). Đây khái niệm đánh địa chỉ bộ nhớ có bù thêm (offset addressing).

- *Thanh ghi lệnh IR (Instruction Register)*

Thanh IR chứa lệnh đang thực hiện. Trong chu kỳ đọc lệnh (instruction cycle) từ bộ nhớ, lệnh được nạp vào IR thông qua bus dữ liệu trong. IR như là bộ đệm duy trì nội dung lệnh và đầu ra của thanh

ghi lệnh đưa tới bộ giải mã lệnh (instruction decoder) để tạo ra chuỗi các tín hiệu điều khiển thực hiện lệnh. Độ dài của IR tùy thuộc vào từng bộ vi xử lý, trong một số loại vi xử lý, độ dài của nó bằng độ dài của từ xử lý. Cũng có thể dựa vào số lượng lệnh của tập lệnh đoán được độ dài của thanh ghi lệnh, vì $2^N = \text{số lượng tối đa các lệnh}$ (maximal number of instructions), trong đó $N = \text{số bit của thanh ghi lệnh}$. Đôi khi độ dài của IR chỉ ngắn bằng 3 đến 4-bit.

- *Các thanh ghi dữ liệu tạm thời (Temporary data registers)*

ALU không có bộ đệm, do đó khi tính toán, các dữ liệu phải được lấy từ hoặc đưa trở lại bus dữ liệu trong theo thời điểm nhịp nào đó. Những việc này, nếu thiếu thanh ghi dữ liệu tạm thời thì không thể thực hiện. Ngoài ra, trong quá trình xử lý cần sự ổn định dữ liệu ở đầu vào ALU, do đó cần có thanh ghi đệm để lưu giữ dữ liệu trong thời gian ngắn đủ để cho ALU thực hiện được xong phép tính. Có hai thanh ghi dữ liệu tạm thời ở hai cổng IN của ALU cho phép ALU thực hiện các phép tính với hai từ dữ liệu 8-bit.

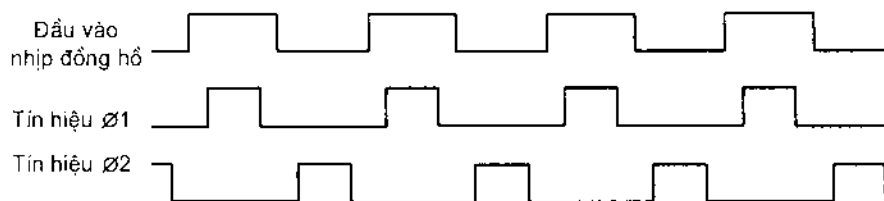
2.2.3. Khối logic điều khiển

Khối logic điều khiển (CL: Control Logic block) có liên hệ thông tin với tất cả các đơn vị trong bộ vi xử lý, bởi vì nó điều khiển toàn bộ hoạt động của những phần còn lại của vi xử lý cùng với nhau và trong một trình tự thời gian. CL nhận các lệnh từ thanh ghi lệnh, hình thành nên những gì phải thực hiện đối với dữ liệu, và sau đó đưa ra các chỉ thị cần thiết cho những phần logic còn lại của vi xử lý. Bình thường CL được vi chương trình hóa (microprogrammed logic unit) hoặc vi mã hóa (microcoded), tức là kiến trúc của CL giống như kiến trúc của của một vi xử lý chuyên dụng nằm bên trong một vi xử lý đa năng. Sự thông minh của CL chính ở bộ giải mã lệnh (instruction decoder). Bộ giải mã lệnh thực hiện giải mã lệnh chứa trong thanh ghi lệnh và đưa ra các tín hiệu điều khiển như là kết quả của giải mã cần thiết để thực hiện lệnh. Khối CL có các đường điều khiển đi đến từng khối chức năng của vi xử lý. Nhưng đường điều khiển này gồm có các đường tín

hiệu đọc/ghi bộ nhớ và vào/ra. Chúng thường không được chỉ ra trong sơ đồ khối bởi vì chúng làm phức tạp sơ đồ và người lập trình không thể can thiệp vào logic điều khiển được. Một tín hiệu vào quan trọng của CL là nhịp đồng hồ của vi xử lý. Nhịp đồng hồ là tín hiệu cơ bản của định thời bên trong vi xử lý. CL thường chuyển đổi nhịp đồng hồ thành tín hiệu nhiều pha.

Hình 2.7 mô tả nhịp đồng hồ 2 pha được tạo ra từ xung vuông như thế nào (đôi khi một vi xử lý sử dụng nhịp đồng hồ 4 pha). Như vậy có thể nhận thấy rằng có hai tác động có thể xảy ra trong một chu kỳ nhịp đồng hồ. Tác động thứ nhất xảy ra trong thời gian với tín hiệu nhịp đồng hồ Ø1, và tác động thứ hai có thể xảy ra trong thời gian với tín hiệu nhịp đồng hồ Ø2. Điều này là chung cho logic điều khiển để tạo ra tín hiệu hoặc pha 1 hoặc pha 2 và để đưa ra tín hiệu sử dụng các thiết bị khác như bộ nhớ và vào/ra.

Vi xử lý có thể lấy nhịp đồng hồ được tạo ra từ bộ tạo nhịp bên ngoài, hoặc nó có thể có nhịp đồng hồ được tạo ra từ một bộ tạo dao động bên trong. Thường thường nhịp đồng hồ của vi xử lý được điều khiển bởi thạch anh để đảm độ chính xác và ổn định.

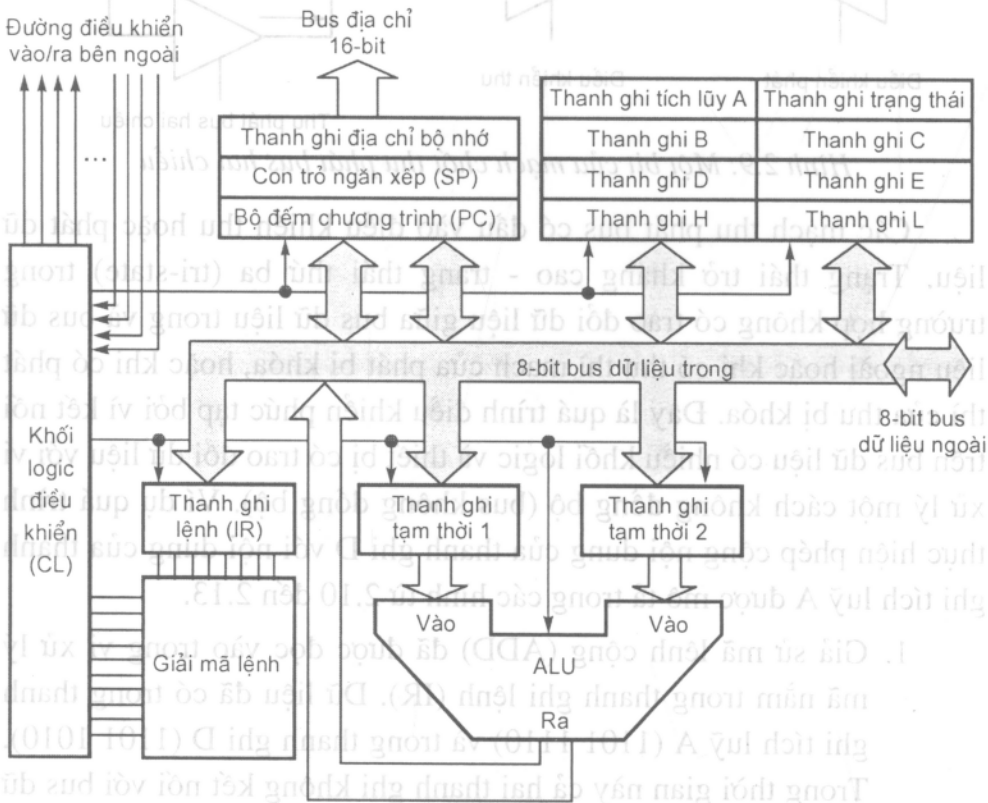


Hình 2.7: Nhịp đồng hồ hai pha. Các tín hiệu Ø1 và Ø2 được tạo ra từ dạng xung tần số cao

CL còn thực hiện một số ít các chức năng đặc biệt khác. Nó điều khiển trình tự bật nguồn, xử lý các ngắt, và quyết định khi nào và trong thứ tự nào thiết bị ngoài có yêu cầu ngắt tạm thời được phục vụ. Khi thực hiện những chức năng dạng như thế này CL giống như người trông nhà cho vi xử lý.

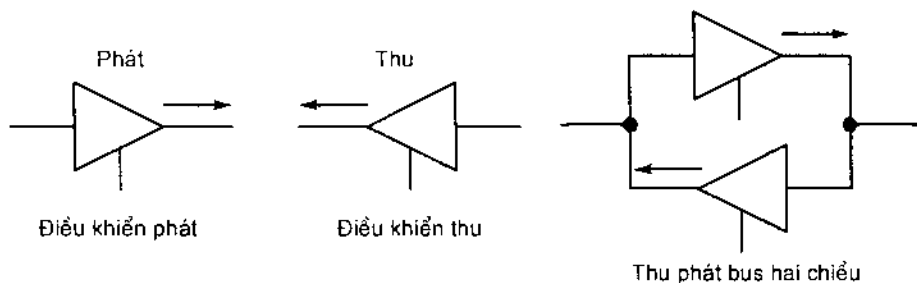
2.2.4. Bus dữ liệu trong

Trong các họ vi xử lý 8-bit, bus dữ liệu trong (internal data bus) đều có 8-bit. Trong hình 2.8 có thể thấy tất cả các thanh ghi và ALU kết nối với bus dữ liệu trong (đường tô đậm). Các khối chức năng bên trong vi xử lý trao đổi dữ liệu qua bus dữ liệu bên trong. Các tín hiệu điều khiển của CL có vai trò trong điều khiển sử dụng bus dữ liệu trong. Đầu ra của ALU là kết quả tính toán logic-số học được đưa vào bus dữ liệu. Các thanh ghi tạm thời 1 và 2 ở đầu vào của ALU nhận dữ liệu từ bus dữ liệu. Thanh ghi lệnh (IR) nhận mã lệnh từ bus dữ liệu. Các thanh ghi còn lại kết nối truyền thông hai chiều với bus dữ liệu bên trong vi xử lý. Tức là chúng có thể nhận dữ liệu từ bus và chuyển dữ liệu lên bus.



Hình 2.8: Bus dữ liệu trong của vi xử lý 8-bit

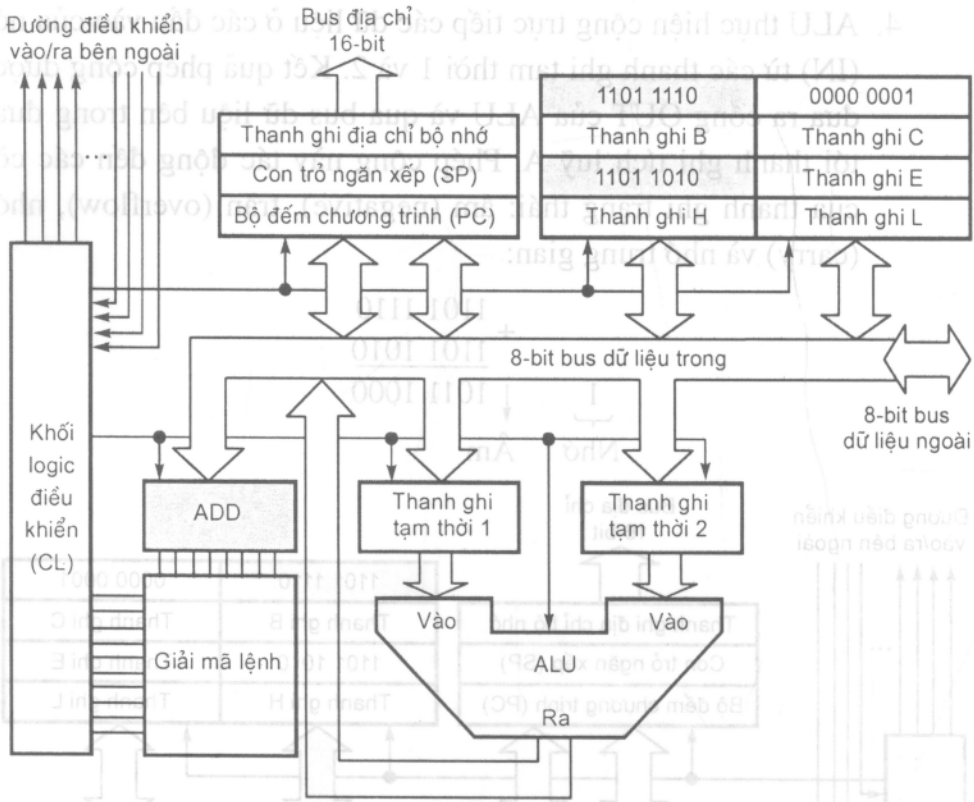
Bus dữ liệu trong kết nối với bus dữ liệu ngoài khi vi xử lý cần trao đổi dữ liệu với các thiết bị bên ngoài như: bộ nhớ, các thiết bị ngoại vi. Sự kết nối của bus dữ liệu trong và ngoài được thực hiện thông qua các mạch chốt có khả năng điều khiển hướng dữ liệu: từ bus dữ liệu ngoài vào bus dữ liệu trong (thu nhận) và ngược lại (phát). Những mạch chốt đó gọi là thu phát bus hai chiều (bidirectional bus transceiver). Mỗi một bit dữ liệu của một mạch thu phát bus hai chiều được điều khiển đi qua hai mạch phát (Transmitter) và thu (Receiver) như mô tả ở hình 2.9.



Hình 2.9: Một bit của mạch chốt thu phát bus hai chiều

Các mạch thu phát bus có đầu vào điều khiển thu hoặc phát dữ liệu. Trạng thái trở kháng cao - trạng thái thứ ba (tri-state) trong trường hợp không có trao đổi dữ liệu giữa bus dữ liệu trong và bus dữ liệu ngoài hoặc khi có thu thì mạch của phát bị khóa, hoặc khi có phát thì của thu bị khóa. Đây là quá trình điều khiển phức tạp bởi vì kết nối trên bus dữ liệu có nhiều khối logic và thiết bị có trao đổi dữ liệu với vi xử lý một cách không đồng bộ (bus không đồng bộ). Ví dụ quá trình thực hiện phép cộng nội dung của thanh ghi D với nội dung của thanh ghi tích lũy A được mô tả trong các hình từ 2.10 đến 2.13.

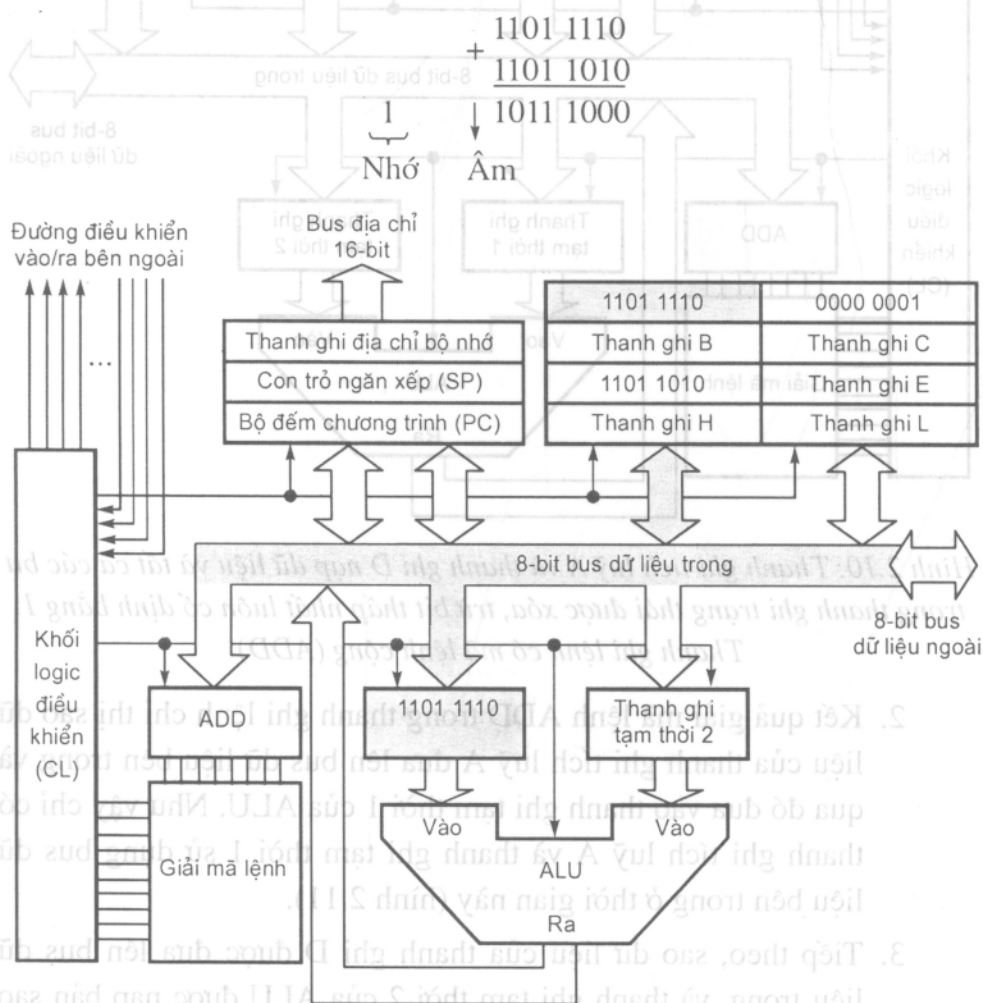
1. Giả sử mã lệnh cộng (ADD) đã được đọc vào trong vi xử lý mã nằm trong thanh ghi lệnh (IR). Dữ liệu đã có trong thanh ghi tích lũy A (1101 1110) và trong thanh ghi D (1101 1010). Trong thời gian này cả hai thanh ghi không kết nối với bus dữ liệu trong (hình 2.10).



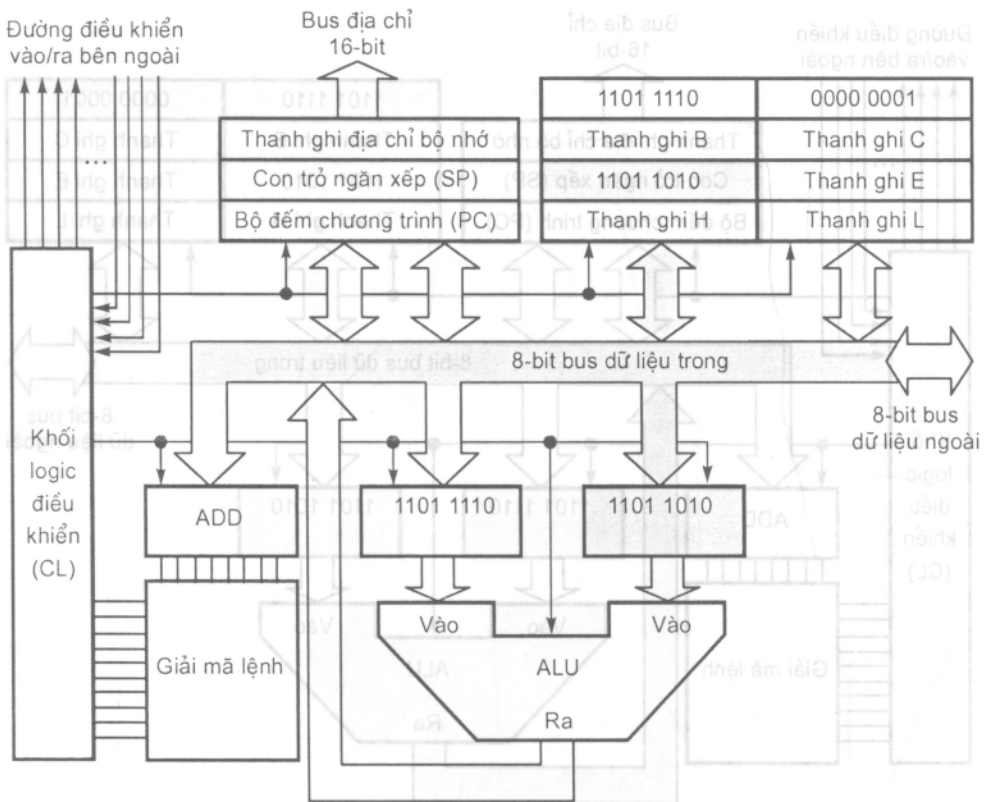
Hình 2.10: Thanh ghi tích lũy A và thanh ghi D nạp dữ liệu và tất cả các bit trong thanh ghi trạng thái được xóa, trừ bit thấp nhất luôn cố định bằng 1.
Thanh ghi lệnh có mã lệnh cộng (ADD)

- Kết quả giải mã lệnh ADD trong thanh ghi lệnh chỉ thị sao dữ liệu của thanh ghi tích lũy A đưa lên bus dữ liệu bên trong và qua đó đưa vào thanh ghi tạm thời 1 của ALU. Như vậy chỉ có thanh ghi tích lũy A và thanh ghi tạm thời 1 sử dụng bus dữ liệu bên trong ở thời gian này (hình 2.11).
- Tiếp theo, sao dữ liệu của thanh ghi D được đưa lên bus dữ liệu trong, và thanh ghi tạm thời 2 của ALU được nạp bản sao nội dung của thanh ghi D. Như vậy chỉ có thanh ghi D và thanh ghi tạm thời 2 sử dụng bus dữ liệu trong trong thời gian này (hình 2.12).

4. ALU thực hiện cộng trực tiếp các dữ liệu ở các đầu vào của nó (IN) từ các thanh ghi tạm thời 1 và 2. Kết quả phép cộng được đưa ra cổng OUT của ALU và qua bus dữ liệu bên trong đưa tới thanh ghi tích lũy A. Phép cộng này tác động đến các cờ của thanh ghi trạng thái: âm (negative), tràn (overflow), nhớ (carry) và nhớ trung gian:

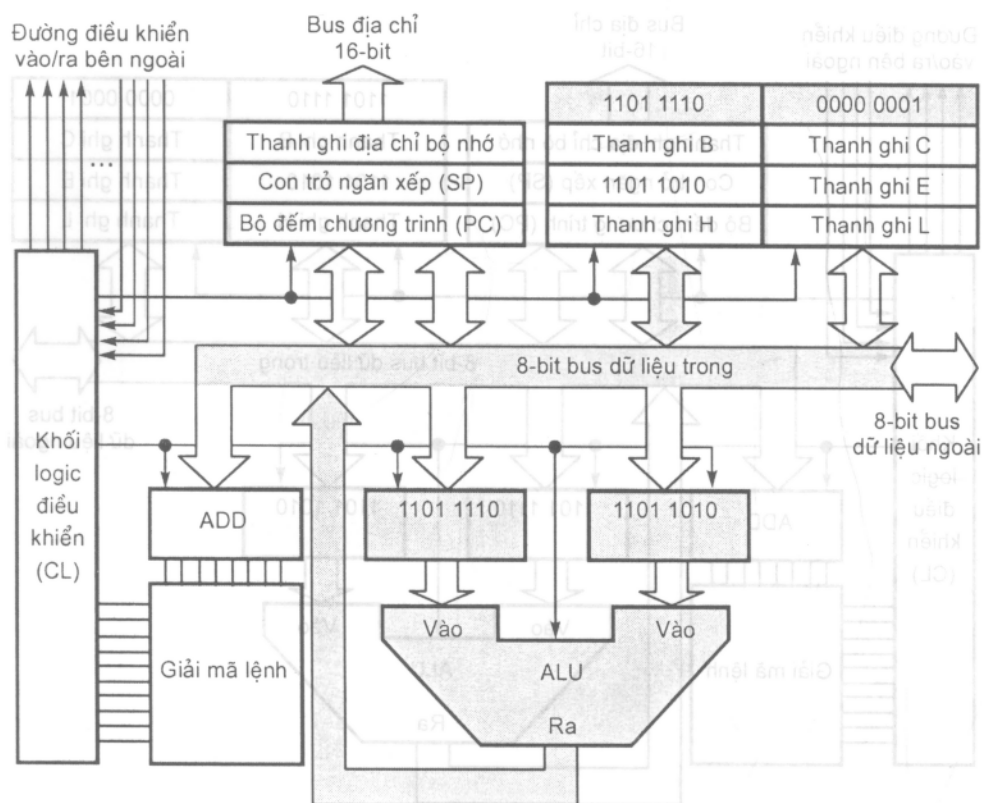


Hình 2.11: Chuyển nội dung thanh ghi tích lũy A đến thanh ghi tạm thời 1 của ALU thông qua bus dữ liệu trong



Hình 2.12: Chuyển nội dung thanh ghi D đến thanh ghi tạm thời 2 của ALU thông qua bus dữ liệu trong

Như vậy, trong thanh ghi tích lũy A có kết quả là 1011 1000 và thanh ghi trạng thái có nội dung mới là 0111 0101 (hình 2.13).



Hình 2.13: Cộng trong ALU, kết quả ghi vào thanh ghi tích lũy A và thay đổi các cờ trong thanh ghi trạng thái

2.3. CẤU TRÚC BỘ NHỚ

2.3.1. Phân loại các bộ nhớ

(1) Phân theo kiểu truy cập

- Bộ nhớ truy cập ngẫu nhiên RAM (Random Access Memory): tất cả các vùng chứa dữ liệu trong bộ nhớ loại này được truy cập để ghi hoặc đọc tùy tiện và trực tiếp trong quãng thời gian như nhau (access time).

- Bộ nhớ truy cập tuần tự SAM (Sequential Access Memory): dữ liệu trong bộ nhớ hoặc ghi vào bộ nhớ một cách tuần tự và trong quãng thời gian khác nhau (bằng từ).

- Bộ nhớ chỉ được đọc ROM (Read Only Memory): chỉ có thể đọc dữ liệu từ bộ nhớ loại này mà không thể ghi dữ liệu vào được.

(2) Phân theo khả năng duy trì dữ liệu

- Bộ nhớ không ổn định (Volatile memory): dữ liệu lưu trong bộ nhớ loại này bị mất đi khi mất nguồn nuôi (RAM).

- Bộ nhớ ổn định (Nonvolatile memory): dữ liệu lưu trong bộ nhớ loại này vẫn duy trì không thay đổi khi nguồn nuôi không có (ROM, PROM, EPROM, EEPROM, FLASH, đĩa từ, băng từ).

(3) Phân theo công nghệ chế tạo

- Bộ nhớ bán dẫn (RAM, ROM, PROM, EPROM, EEPROM, FLASH).

- Bộ nhớ trên vật liệu từ (đĩa từ, băng từ).

- Bộ nhớ công nghệ quang (CD-ROM, CD-RW).

2.3.2. Bộ nhớ truy cập ngẫu nhiên (RAM)

Trong tổ chức phân cấp bộ nhớ của hệ thống vi xử lý, tập các thanh ghi (cấp L0) và bộ nhớ cache (cấp L1) bên trong các chip vi xử lý đã được xét qua. Bộ nhớ cache bên trong (cấp L1) và bên ngoài (cấp L2) cũng như bộ nhớ chính (cấp L3) của một hệ thống máy tính là các bộ nhớ bán dẫn. Bộ nhớ chính (cấp L3), bao gồm các bộ nhớ cố định chỉ được phép đọc dữ liệu ra, gọi là các bộ nhớ chỉ đọc ROM (Read Only Memory), và bộ nhớ truy cập (ghi và đọc) ngẫu nhiên RAM (Random Access Memory). Thuật ngữ bộ nhớ truy cập ngẫu nhiên (RAM) có nghĩa là có thể ghi và đọc đến bất kỳ vùng nào dễ dàng như nhau trong bộ nhớ. Thực tế RAM rất thông dụng trong các hệ thống vi tính đến nỗi chúng ta thường nói RAM là bộ nhớ chính của vi xử lý. Điều quan trọng cần nhớ là thuật ngữ RAM không cho biết đó là bộ nhớ đọc ghi hay chỉ đọc. Ngày nay thuật ngữ RAM thường dùng để nói đến RAM đọc ghi. Chúng ta thường nói một cách đơn giản là RAM, khi chúng ta muốn nói đến RAM IC (vi mạch nhớ truy cập ngẫu nhiên).

Khi làm việc với bộ nhớ, thời gian truy cập bộ nhớ và chu kỳ của bộ nhớ đều đo thời gian thực hiện của bộ nhớ. Hai phép đo này có quan hệ với nhau, có nghĩa là thời gian truy cập càng nhanh thì chu kỳ bộ nhớ càng nhanh. Thời gian truy cập bộ nhớ cho biết trong bao lâu thì bộ nhớ có thể đưa thông tin từ vùng nhớ đã được chỉ ra bus dữ liệu, nó được gọi là thời gian truy cập đọc bộ nhớ hoặc gọi là thời gian đọc. Thời gian truy cập ghi được đo bằng thời gian cần thiết để bộ nhớ ghi dữ liệu vào vùng nhớ đã được chỉ định. Thời gian truy cập ghi bộ nhớ phụ thuộc vào cấu trúc bộ nhớ và tốc độ bộ nhớ. Ví dụ: một bộ nhớ mạch tổ hợp có thể có thời gian truy cập là 50 ns. Đối với bộ nhớ băng từ thì thời gian truy cập phụ thuộc vào vị trí mà địa chỉ của dữ liệu được tìm thấy trên băng. Nếu dữ liệu cần truy cập nằm ở đoạn đầu của băng thì thời gian truy cập tương đối ngắn, ngược lại nếu dữ liệu cần truy cập nằm ở cuối băng thì thời gian truy cập tương đối dài. Chu kỳ nhớ là thời gian giữa hai lần truy cập bộ nhớ liên nhau. Chu kỳ của bộ nhớ phụ thuộc vào hệ thống quản lý thời gian của máy tính.

Xét theo công nghệ chế tạo bộ nhớ bán dẫn thì ta chia bộ nhớ thành hai loại: bộ nhớ ổn định hay cố định (bộ nhớ tĩnh) và bộ nhớ không ổn định (bộ nhớ động). Bộ nhớ ổn định vẫn giữ được dữ liệu khi mất nguồn cung cấp, ngược lại bộ nhớ không ổn định thì mất dữ liệu khi mất nguồn.

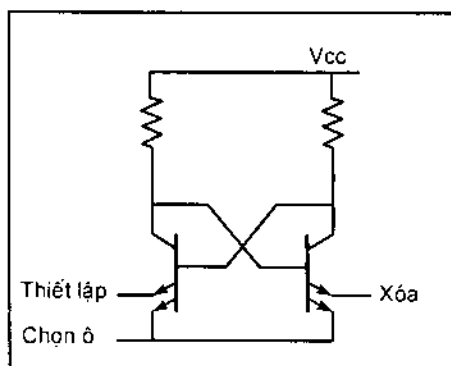
Máy tính không thể làm gì nếu không có lệnh. Như vậy cần có một vài bộ nhớ tĩnh, đó là nơi chứa những lệnh bắt đầu khi bật máy. Hầu hết những bộ nhớ tĩnh không cần lớn, chúng chỉ dùng để chứa những chương trình nhỏ còn phần lớn những chương trình lệnh khác có sẵn ở thiết bị nhớ ngoài. Những chương trình nhỏ giúp máy tính nạp những lệnh còn lại vào bộ nhớ chính từ thiết bị nhớ ngoài. Ví dụ: một chương trình nhỏ được gọi là bootstrap hoặc boot program (chương trình môi), chương trình này thường được chứa trong ROM (bộ nhớ chỉ được đọc). Như vậy tất cả những chương trình lệnh phải được lưu giữ trong bộ nhớ ổn định. Có hai công nghệ bán dẫn được sử dụng để

chế tạo thiết bị nhớ mạch tổ hợp để dùng cho máy tính ngày nay. Đó là công nghệ lưỡng cực và MOS (Metal Oxide Semiconductor).

Bộ nhớ lưỡng cực rất ít được sử dụng cho hệ thống vi xử lý. Ưu điểm của bộ nhớ lưỡng cực là thời gian truy cập nhanh. Chúng có nhiều nhược điểm so với bộ nhớ bán dẫn MOS. Nó tiêu hao năng lượng lớn và có ít bit nhớ trên cùng một chip, quá trình chế tạo bán dẫn lưỡng cực phức tạp hơn quá trình chế tạo MOS, do đó bộ nhớ lưỡng cực đắt hơn nhiều so với bộ nhớ MOS. Vì những lý do đó bộ nhớ lưỡng cực chỉ sử dụng cho những ứng dụng chịu được giá thành cao. Cho đến nay bộ nhớ công nghệ MOS là bộ nhớ thông dụng nhất dùng cho máy vi tính. Có hai cách để chế tạo các mạch tổ hợp (IC) cho bộ nhớ công nghệ MOS, bộ nhớ công nghệ MOS có cả tĩnh và động. Chúng ta thường thấy những từ viết tắt như SRAM (Static RAM), DRAM (Dynamic RAM) dùng để chỉ RAM tĩnh và RAM động. Bộ nhớ tĩnh chế tạo đơn giản hơn, đặc biệt đối với bộ nhớ cần dung lượng nhỏ và yêu cầu tốc độ truy cập nhanh. Bộ nhớ động sử dụng mạch tổ hợp (IC) có giá thành thấp nhưng đòi hỏi phải có mạch phụ, đồng thời bộ nhớ động phải thường xuyên được làm tươi, thời gian truy cập chậm, nên chúng thường sử dụng cho các bộ nhớ cần dung lượng lớn. Công nghệ bán dẫn tĩnh và động đều đang phát triển nhanh chóng, khi người ta có thể chế tạo các bộ nhớ có dung lượng lớn trên cùng một chip thì các bộ nhớ trở nên rẻ hơn. Khi bộ nhớ rẻ hơn thì máy tính trở nên mạnh hơn.

(1) Bộ nhớ RAM tĩnh (Statistic RAM)

Ngày nay hầu hết các bộ nhớ tĩnh đều được chế tạo bằng công nghệ MOS, cũng có một số được chế tạo bằng công nghệ lưỡng cực. Công nghệ MOS thường có hai loại: NMOS và CMOS. NMOS đơn giản hơn CMOS, CMOS tốn ít năng lượng hơn, chúng thường để chế tạo các bộ nhớ phải dùng nguồn dự trữ để ổn định trạng thái nhớ. Sơ đồ đơn giản của một ô nhớ lưỡng cực trình bày ở hình 2.14.

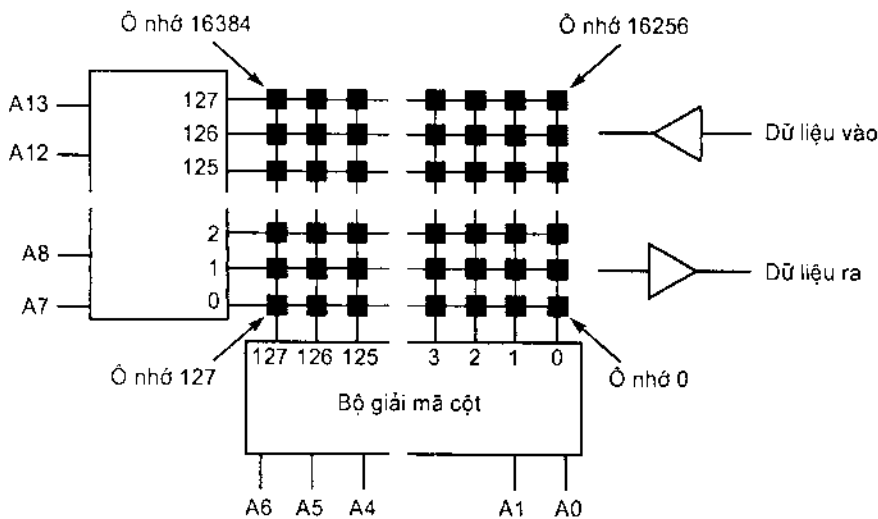


Hình 2.14: Ô nhớ (1 bit) lưỡng cực, sử dụng Transistor 2 Emitter đấu E chung, làm thành mạch lật đơn giản. Ô nhớ được thiết lập hoặc thiết lập lại bằng sự xác định Emitter phù hợp.

Cho phép chọn ô bằng cách đặt trở kháng ra ở mức cho phép

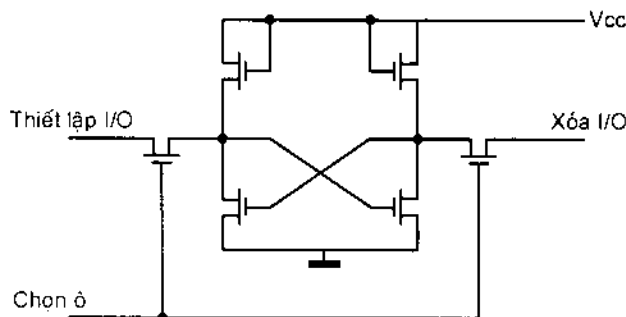
Một ô nhớ lưu giữ một bit thông tin. Mỗi ô nhớ được thực hiện bằng công nghệ nhiều cực phát Emitter TTL (Transistor - Transistor Logic). Ta thấy rằng ô nhớ này chỉ là mạch lật (Flip-Flop) đơn giản. Flip-flop này có thể được thiết lập trạng thái hoặc thiết lập lại trạng thái để lưu giữ dữ liệu. Nếu nó được thiết lập trạng thái thì nó sẽ được giữ nguyên trạng thái đó cho đến khi được thiết lập lại trạng thái hoặc đến khi nó bị mất nguồn nuôi. Nguồn nuôi Vcc có giá trị mức TTL. Để tạo một bộ nhớ có dung lượng tùy ý, hầu hết phải sắp xếp các ô nhớ (Flip-flop) theo một ma trận.

Hình 2.15 trình bày sơ đồ ma trận (128×128) của chip nhớ 16.384 bit. Sơ đồ khối đơn giản của một phân bộ nhớ này giống như bộ nhớ công nghệ lưỡng cực và bộ nhớ công nghệ MOS tĩnh và động. Mỗi một chip IC có 14 đường địa chỉ từ A0÷A13, các đường địa chỉ này được nối với ma trận giải mã địa chỉ. Bộ giải mã cột có thể giải mã được địa chỉ 7-bit để xác định được một trong 128 cột. Bảy đường địa chỉ đầu từ A0÷A6 để giải mã cột. Bảy đường địa chỉ sau từ A9÷A13 để giải mã hàng. Đầu ra bộ giải mã hàng - cột cho ta ô nhớ mong muốn (bit nhớ mong muốn). Mỗi ô nhớ có một địa chỉ xác định trong số 16.384 địa chỉ ô nhớ của chip nhớ. Khi một ô nhớ được chọn thì có thể ghi hoặc đọc dữ liệu vào ô nhớ.



Hình 2.15: Một chip nhớ tĩnh dung lượng 16.384 bit theo ma trận

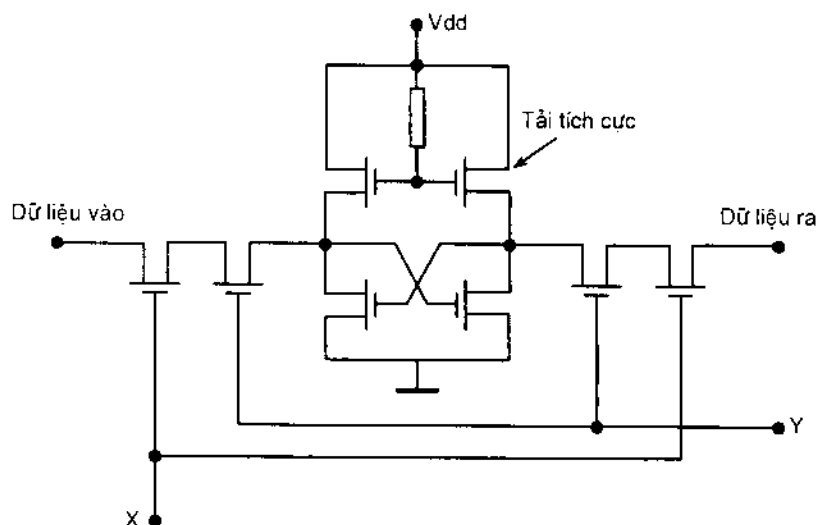
Sơ đồ đơn giản của một ô nhớ tĩnh NMOS trình bày như hình 2.16. Có thể thấy được ô này cũng là một mạch lật (Flip-flop). Cũng như hầu hết các loại bộ nhớ dùng công nghệ MOS, nó sử dụng MOS transistor có thiên áp cố định cho cực máng, thay cho điện trở. MOS transistor sử dụng kết hợp đưa dữ liệu vào/ra cùng với chọn ô. Giống như ô nhớ lưỡng cực, ô nhớ tĩnh công nghệ MOS có hai trạng thái: thiết lập và xoá.



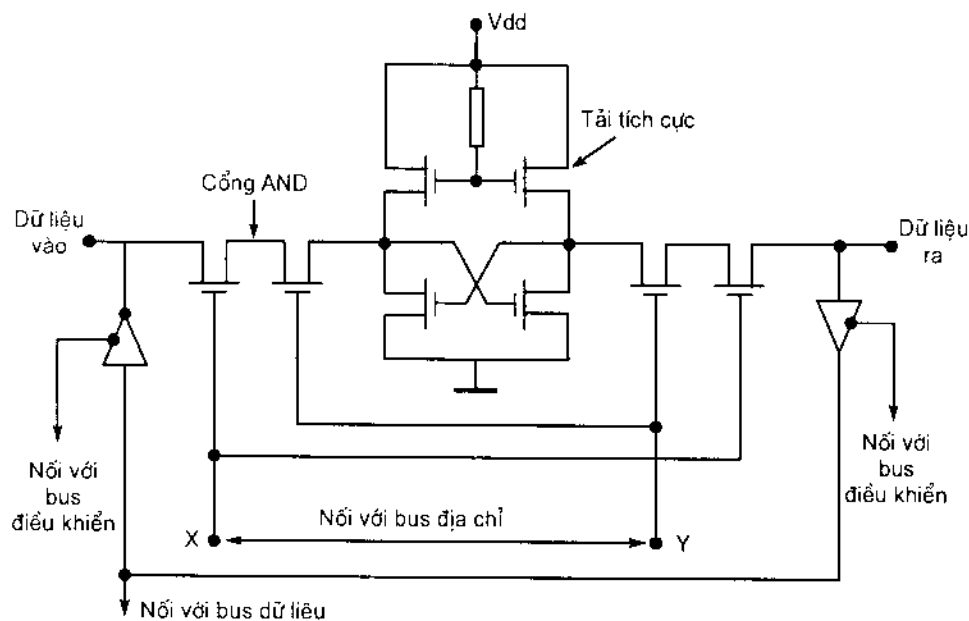
Hình 2.16: Một ô bit nhớ tĩnh bằng Flip-Flop với các bóng bán dẫn NMOS

Do các ô bit SRAM sắp xếp theo ma trận nên để chọn ô bit cần có đường dây chọn hàng (gọi là X) và chọn cột (gọi là Y). Như vậy,

một ô bit của SRAM có dạng cơ bản như trong hình 2.17 và kết nối với bus trong hình 2.18.

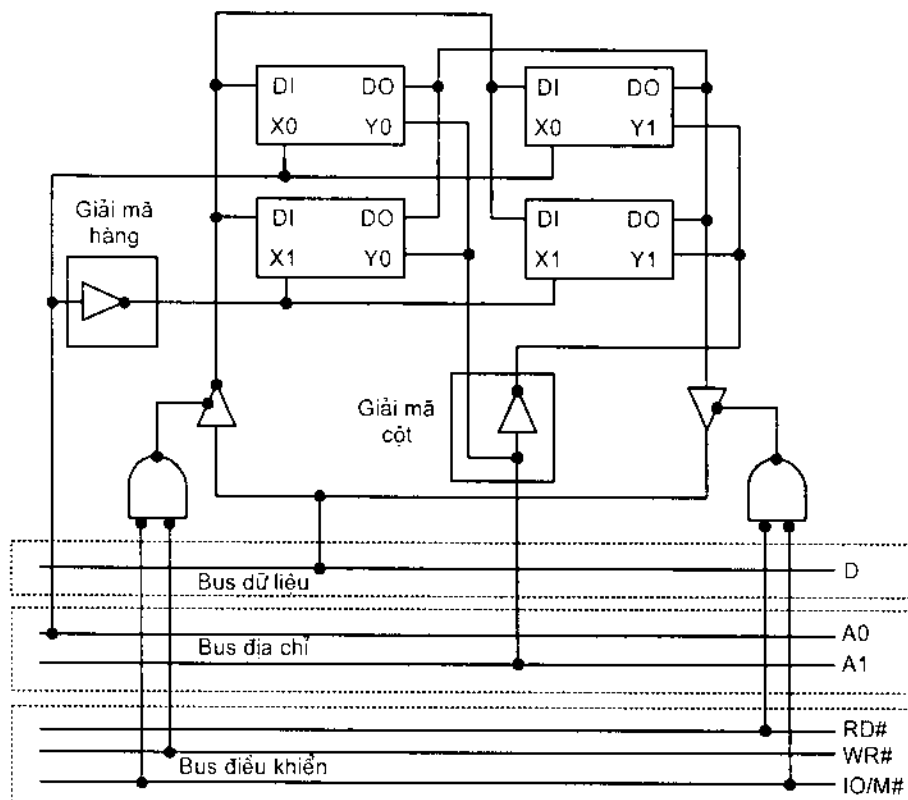


Hình 2.17: Một ô bit SRAM CMOS cơ bản

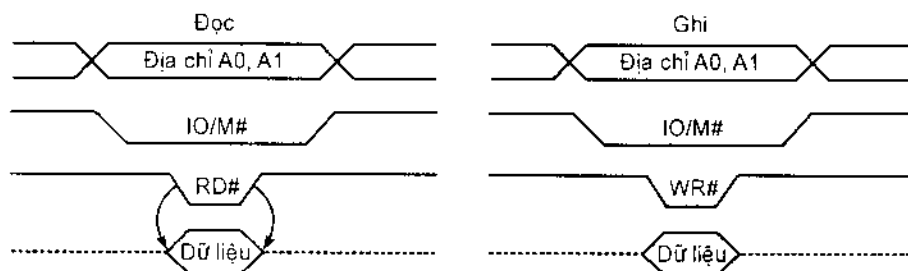


Hình 2.18: Một ô bit SRAM CMOS cơ bản nối với bus hệ thống máy tính

Hình 2.19 là kết nối của vi xử lý 8085 với một chip SRAM dung lượng 4×1 bit và đồ thị xung ghi/đọc của hệ thống này cho trong hình 2.20.

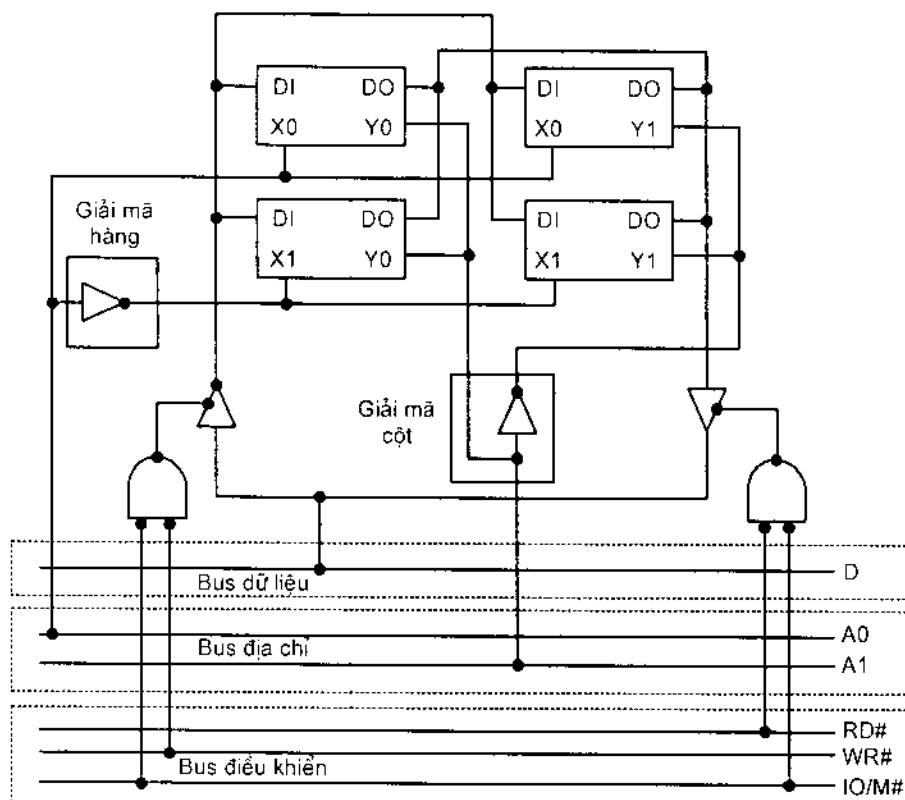


Hình 2.19: Kết nối 8085 với SRAM 4×1 với các ô bit

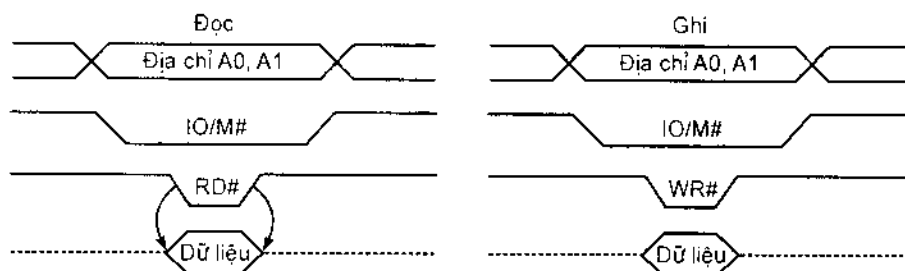


Hình 2.20: Đồ thị xung ghi/đọc SRAM 4×1 trong hệ 8085

Hình 2.19 là kết nối của vi xử lý 8085 với một chip SRAM dung lượng 4×1 bit và đồ thị xung ghi/đọc của hệ thống này cho trong hình 2.20.



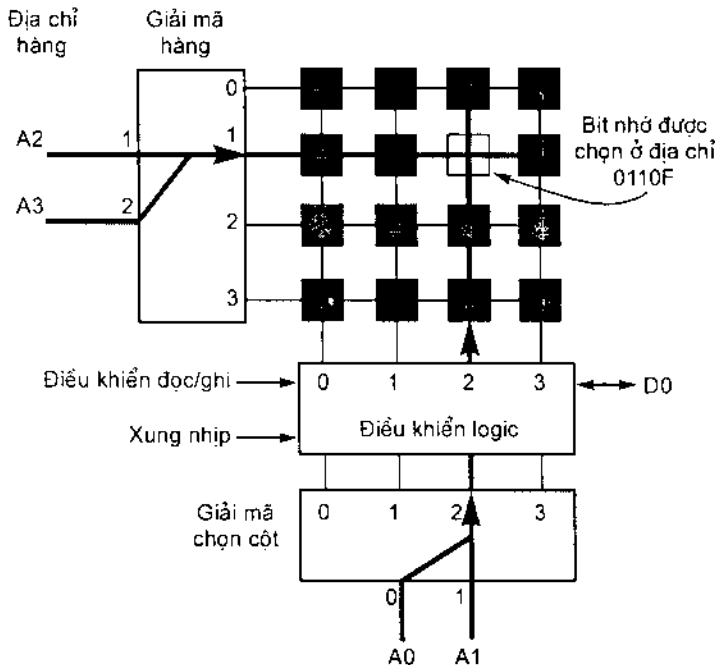
Hình 2.19: Kết nối 8085 với SRAM 4×1 với các ô bit



Hình 2.20: Đồ thị xung ghi/đọc SRAM 4×1 trong hệ 8085

Có RAM 4×4 chứa 16 ô nhớ (16-bit). Khi các ô nhớ được tổ chức thành 16×1 RAM thì ta có bộ nhớ tổ chức theo bit (bit-organized memory) (hình 2.21).

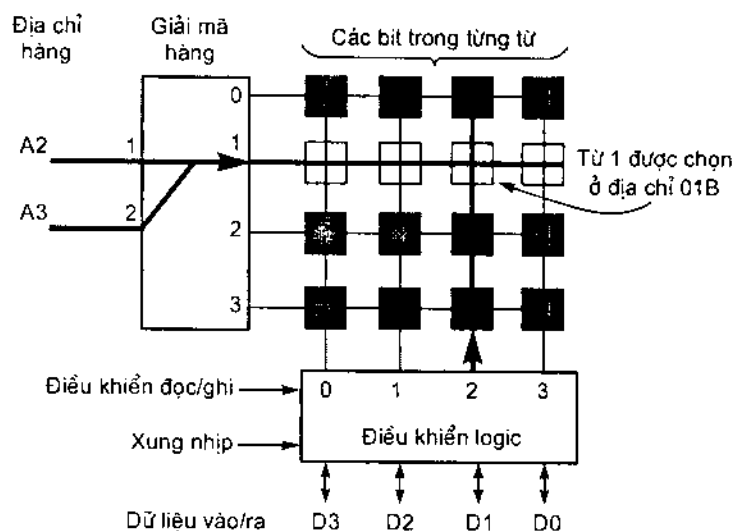
Khi tổ chức như là 4×4 RAM thì ta có bộ nhớ tổ chức theo từ (word-organized memory) (hình 2.22).



Hình 2.21: SRAM 16×1 tổ chức theo ma trận

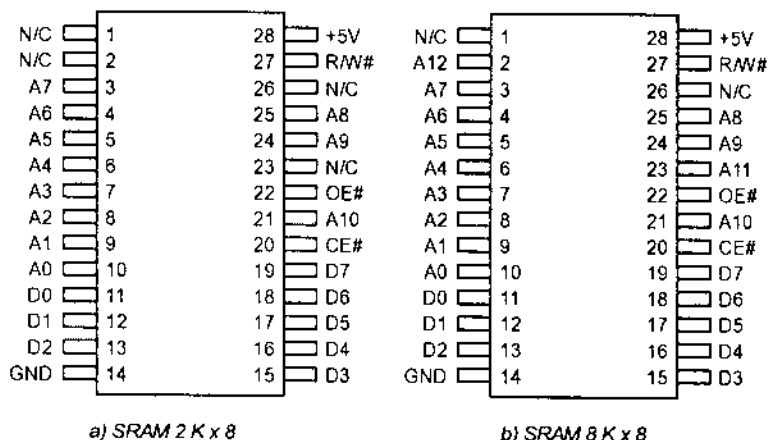
Hình 2.23 (a) là SRAM có thể lưu giữ được 16.384 bit dữ liệu, dữ liệu được tổ chức thành 2.048 từ 8-bit (8 bit = 1 byte), tức là 2 kbyte ($16.384 = 2.048 \times 8 = 2 \text{ kbyte} \times 8$). Vi mạch có 11 đường địa chỉ để chọn ra một byte trong 2.048 byte ($2^{11} = 2.048$). Một số bộ nhớ đặc biệt sử dụng 12 bit địa chỉ phục vụ thêm việc chọn chip IC.

Khi một chip IC được chọn thì chip IC khác không được chọn. Mỗi IC có 8 đường dữ liệu (8 bit), các đường dữ liệu này để nhận dữ liệu từ bus hoặc gửi dữ liệu đến bus. Ngoài ra còn có các chân điều khiển ghi đọc bộ nhớ.



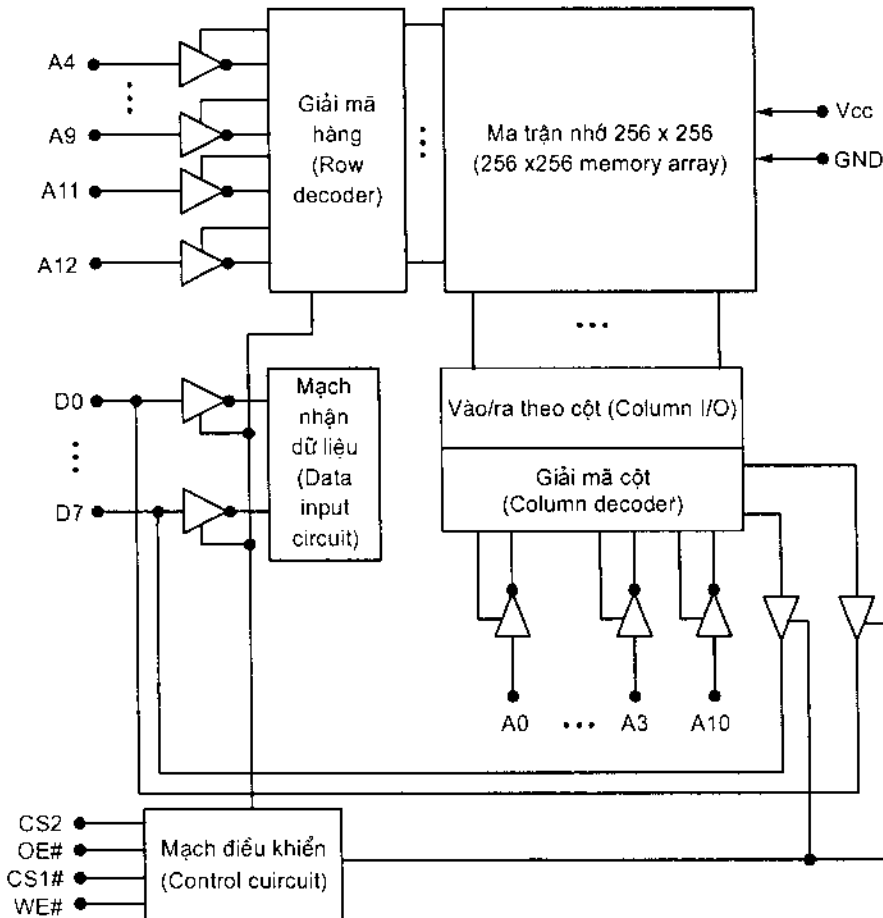
Hình 2.22: SRAM 4×4

Hình 2.23 (b) là một vi mạch nhớ SRAM có 65.536 bit ($8 \times 1.024 \times 8 \text{ bit} = 8 \text{ kbyte} \times 8 \text{ bit}$). IC có 13 đường địa chỉ (A0-A13) để địa chỉ tới 8 kbyte ($2^{13} = 8 \text{ kbyte} = 65.536 \text{ bit}$), 8 đường dữ liệu (8 bit), chọn chip, và các đường cho phép ghi hoặc đọc. IC được gọi 8K 58 SRAM (5164S/L8K×8 bit SRAM). Những IC SRAM 2K 58 và 8K 58 có sẵn với cả công nghệ NMOS và CMOS. Công nghệ CMOS rất đắt nhưng tiêu tốn ít năng lượng.



Hình 2.23: Hai loại SRAM phổ biến: 2K×8 và 8K×8 có thể dùng chung để cắm

Hình 2.24 là sơ đồ khối cấu trúc bên trong của SRAM $8K \times 8$ -bit được kết cấu theo ma trận 256×256 , tức là $2^8 \times 2^8$. Có 8 đường địa chỉ để giải mã chọn hàng của ma trận nhớ (256 hàng): A12, A11, A9, A8, A7, A6, A5, A4; và có 5 đường địa chỉ để giải mã chọn cột (32 cột): A10, A3, A2, A1, A0.



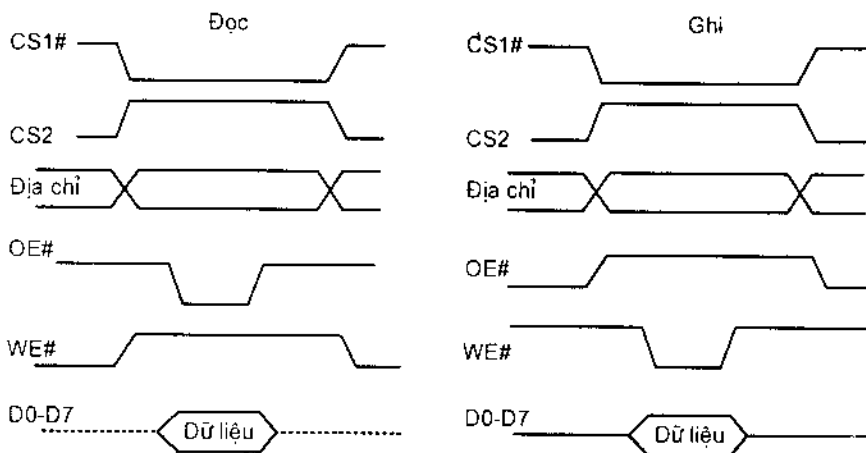
Hình 2.24: Sơ đồ khối 5164S/L $8K \times 8$ CMOS SRAM

Như vậy mỗi cột phải có 8 bit trên một hàng cùng đồng thời được chọn để đảm bảo: $2^5 \times 8 = 256$.

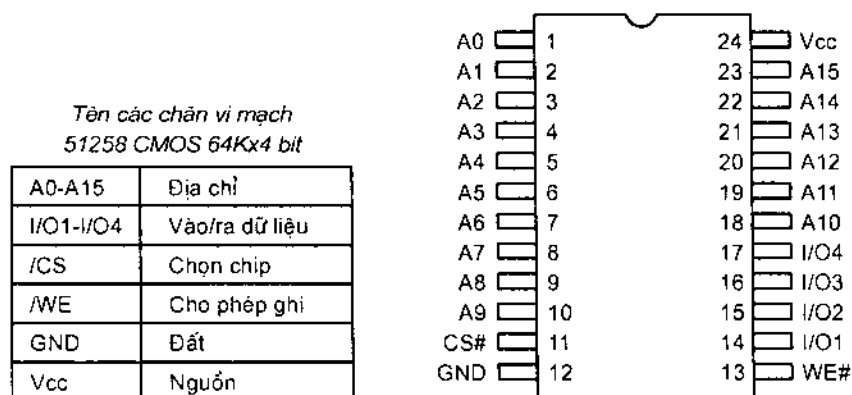
Byte dữ liệu D0-D7 hai chiều (cho ghi và đọc) do khối logic nhận dữ liệu và khối vào/ra theo cột đảm trách.

Mạch điều khiển nội bộ RAM tiếp nhận các tín hiệu điều khiển từ bus hệ thống của vi xử lý để chọn chip RAM (CS1#, CS2), đọc và ghi vào RAM (OE#, WE#).

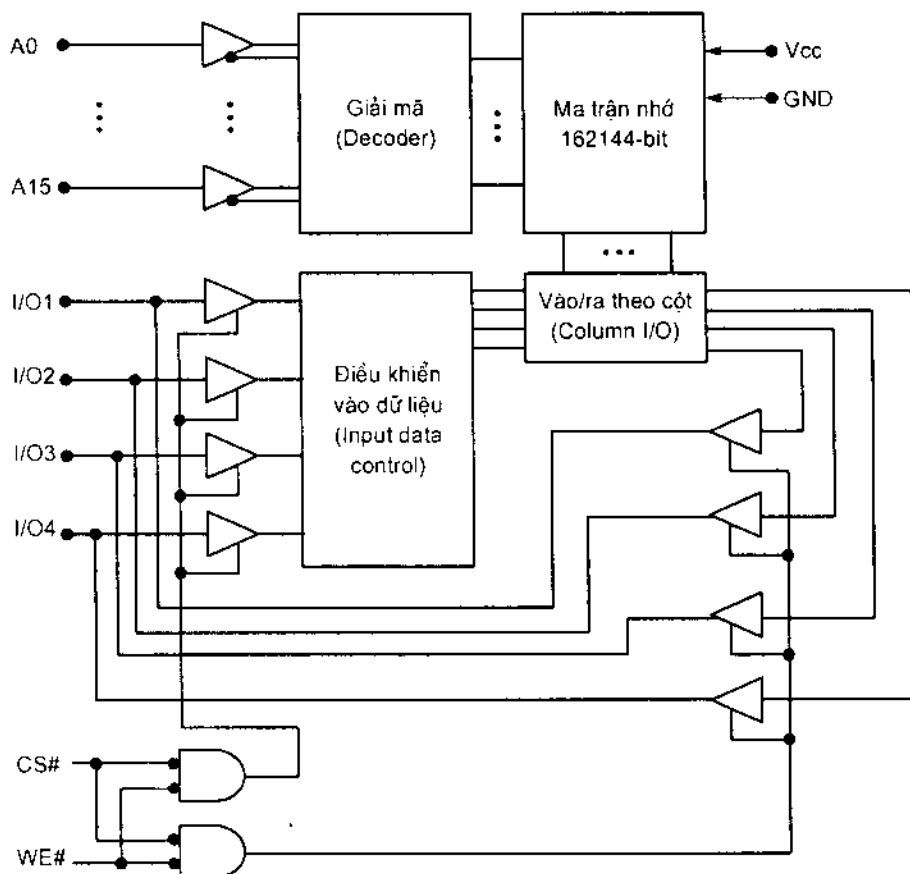
Quá trình điều khiển ghi/đọc của SRAM $8K \times 8$ bit được biểu diễn theo đồ thị xung ở hình 2.25. Để chọn chip SRAM, các tín hiệu giải mã chọn chip RAM phải đảm bảo CS1# xuống mức tích cực "0" và tín hiệu CS2 lên mức tích cực "1". Đồng thời, các đường dây địa chỉ (address) từ bus hệ thống (từ A12-A0) phân thành 2 nhóm: nhóm địa chỉ chọn hàng của ma trận nhớ gồm: A12-A11, A9-A4 và nhóm địa chỉ chọn cột: A10, A3-A0 đi vào các mạch giải mã hàng và cột bên trong RAM. Nếu là chu trình đọc dữ liệu từ RAM (Read) thì tín hiệu OE# xuống mức tích cực "0" và WE# phải ở mức "1" trong quãng thời gian các tín hiệu địa chỉ đã ổn định và đảm bảo đã có chọn hàng, cột ma trận nhớ xong. Dữ liệu đọc từ RAM (Data) được đưa ra bus hệ thống (D7-D0). Quá trình chọn chip RAM và giải mã chọn hàng, cột trong ma trận nhớ của chu trình ghi dữ liệu cũng tương tự, chỉ khác là để ghi dữ liệu OE# chuyển lên mức "1" và WE# chuyển xuống mức tích cực "0".



Hình 2.25: Ghi/đọc 5164S/L $8K \times 8$ bit SRAM



a) Bố trí chân vi mạch



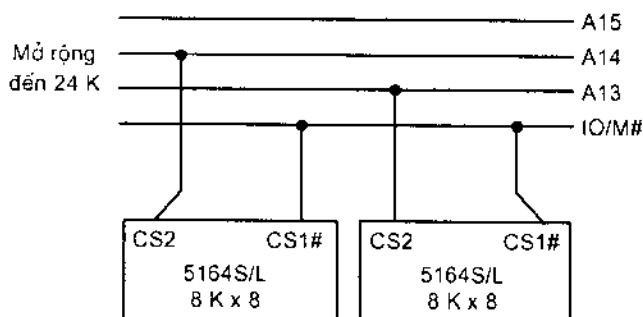
b) Sơ đồ khối

Hình 2.26: Bố trí chân vi mạch và sơ đồ khối
CMOS SRAM 51258 64Kx4 bit tốc độ cao

Bằng công nghệ tích hợp lớn, các chip SRAM có thể đạt đến dung lượng cao hơn nữa. Trong những năm đầu thập niên 1990, rất phổ dụng các chip SRAM 51258 (CMAOS 64K×4 bit) (Hình 2.26), tiếp sau đó là các chip 1M×4 bit, 16M×4 bit, bây giờ đã lên đến 32M×8 bit, 64M×8 bit và hơn nữa,... Do khả năng đánh địa chỉ của các bộ vi xử lý ngày càng mạnh, ví dụ như 80486 đánh địa chỉ tới 4 GB (4 tỷ byte), thì các chip SRAM có dung lượng lớn để dành cho kết nối với hệ thống vi xử lý công nghệ cao.

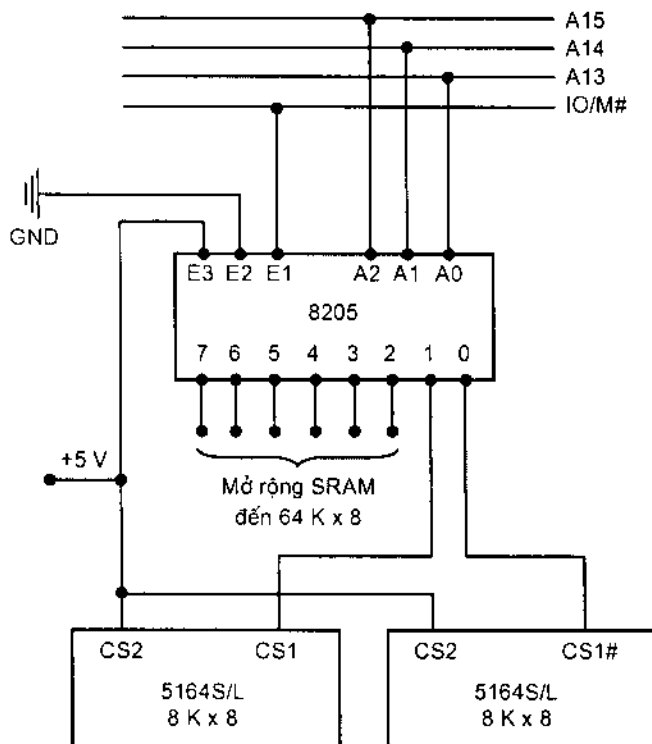
Để thiết kế các RAM có dung lượng lớn phải kết hợp các chip RAM dung lượng thấp hơn lại với nhau. Ví dụ nếu muốn có RAM 64K×8 bit trong một máy vi tính, ta có thể sử dụng 8 chip SRAM 8K×8 bit. Dĩ nhiên bộ vi xử lý phải có khả năng địa chỉ tới được 64 kbyte, tức là nó phải tạo ra 16 đường địa chỉ (A0-A15), vì $2^{16} = 2^6 \times 2^{10} = 64 \text{ kbyte}$. Câu hỏi đặt ra là: phải kết nối như thế nào các module 8K×8 bit?

Khi cần có dung lượng nhớ lớn, chúng ta phải dùng tới giải mã chọn các chip nhớ (hình 2.27). Trường hợp đơn giản nhất là nếu như có ít chip nhớ, và ta chỉ tạo ra RAM 24K×8 bit, thì có thể nối trực tiếp các đường địa chỉ A13, A14 vào các chân CS2 (Chip Select Two) của các chip SRAM 8K×8 (hình 2.28). Kỹ thuật này tiết kiệm kết nối, và nó thường được gọi là đánh địa chỉ tuyến tính (linear addressing).



Hình 2.27: Đánh địa chỉ trực tiếp bằng các đường địa chỉ để chọn ngay chip nhớ

Các RAM tĩnh và ROM cùng dung lượng được thiết kế các chân tín hiệu tương tự nhau, thực chất các thiết bị này giống nhau. Như vậy trên bảng mạch bộ vi xử lý trung tâm của máy tính có để cắm có thể cắm RAM tĩnh hoặc ROM.



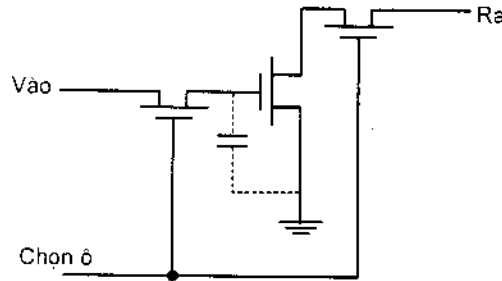
Hình 2.28: Giải mã chọn các chip SRAM, mở rộng đến 64Kx8

(2) Bộ nhớ RAM động (dynamic RAM)

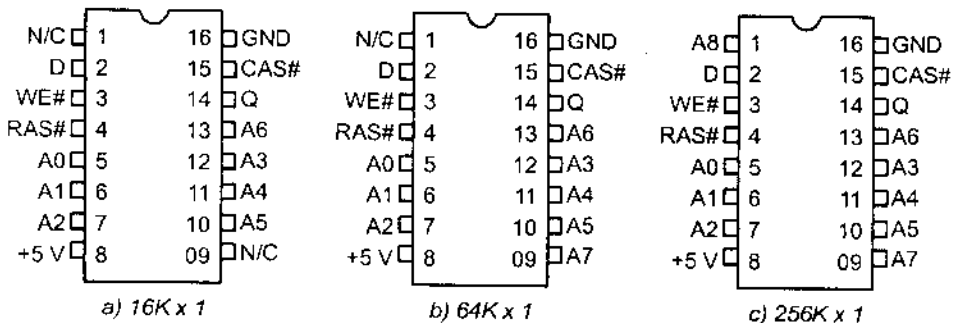
Ô bit nhớ của DRAM công nghệ MOS đơn giản hơn (hình 2.29). Có thể thấy rằng các ô nhớ này không phải là mạch flip-flop, chúng làm việc theo nguyên tắc tích điện của tụ điện.

Điện dung rất nhỏ được dùng như là một ô bit có giá trị logic 0 hoặc 1 của bộ nhớ. Điện dung ghi dữ liệu nhị phân, logic 1 hoặc 0, nhờ sự tích điện trong khoảng vài milli giây. Sau khoảng thời gian này dữ liệu cần phải được ghi lại ô bit. Sự ghi lại dữ liệu vào trong ô bit nhớ động MOS được gọi là làm tươi dữ liệu. Ô bit nhớ được làm tươi mỗi

khi nó được truy cập tới. Trong thời gian làm tươi mọi sự truy cập bộ nhớ đều bị cấm. Cứ vài chục milli giây logic làm tươi tự động truy cập đến từng cột của DRAM. DRAM được xây dựng sao cho sự truy cập đơn giản đến từng cột làm tươi được tất cả các bit nhớ trong cột. Quá trình làm tươi logic phải kết hợp đồng bộ với hoạt động của CPU để không cản trở hoạt động truy cập bộ nhớ của vi xử lý. Ví dụ nếu đơn vị xử lý cố gắng truy cập bộ nhớ trong khi bộ nhớ đang được làm tươi, thì logic làm tươi phải dành cho đơn vị xử lý quyền ưu tiên. Logic làm tươi phải làm tươi trước khi dữ liệu bị mất.



Hình 2.29: Ô nhớ động dùng transistor MOS. Tụ điện là điện dung vào của transistor. Tụ tích làm cho ô bit ở mức "1", không tích điện làm cho ô bit ở mức "0"



Hình 2.30: Các chip nhớ DRAM

Hình 2.30 đưa ra vị trí các chân của các DRAM: 16K, 64K, 256K. Có thể thấy những chip IC được đóng gói 16 chân loại DIP (Dual In-line Package). Kỹ thuật địa chỉ bộ nhớ đặc biệt được sử dụng

để tối thiểu hóa số chân. Mỗi một DRAM đưa ra ở hình 2.30 chỉ có một chân đưa dữ liệu ra và một chân đưa dữ liệu vào. Mỗi chip chỉ có một đầu vào dữ liệu ở chân số 2 (D) và một đầu ra lấy dữ liệu ở chân số 14 (Q). Những IC này chỉ lưu giữ theo bit đơn nghĩa là nếu muốn xây dựng một byte nhớ phải cần 8 IC loại này. Mỗi chip đều có chân cho phép ghi, khẳng định dữ liệu được ghi vào IC bằng tín hiệu đưa vào chân này (Enable write), nếu không được cho phép ghi thì chip đó là loại chỉ đọc (ROM IC). Ngoài ra còn có chân chọn hàng (RAS) và cột (CAS). DRAM 16K có 7 đường địa chỉ A0-A6. DRAM 64K có 8 đường địa chỉ A0-A7. DRAM 256K có 9 đường địa chỉ A0-A8. Để chọn bit nhớ các đường địa chỉ này cùng được đưa vào các mạch chốt chọn hàng và cột để giải mã chọn hàng và cột, như vậy mới đủ để địa chỉ hóa toàn bộ dung lượng của chip nhớ, nghĩa là:

$$2^7 \times 2^7 = 16.384 \text{ (16 K)}$$

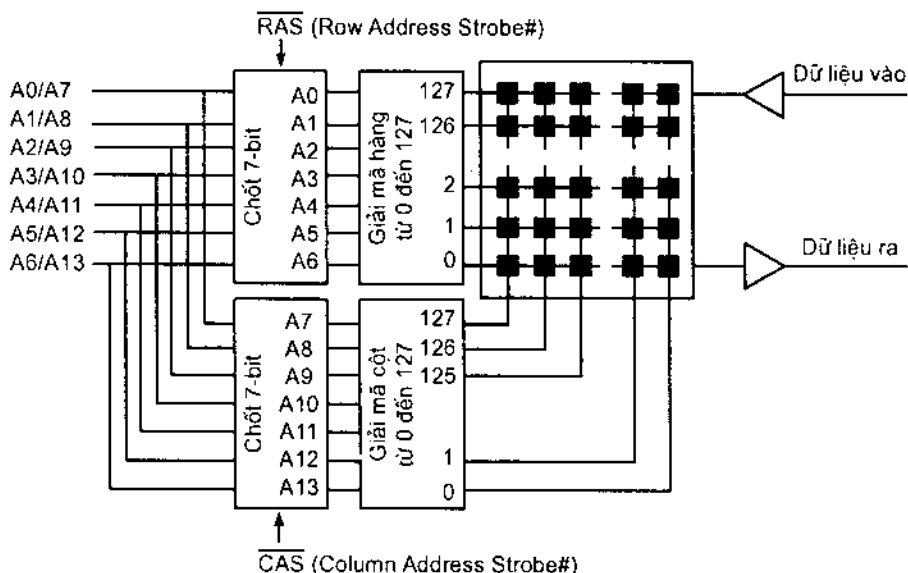
$$2^8 \times 2^8 = 65.536 \text{ (64 K)}$$

$$2^9 \times 2^9 = 262.144 \text{ (256 K)}$$

Cả ba IC nhớ này có những dòng địa chỉ A0-A6 là giống nhau (số chân). Một DRAM 16K không thể nối chân 1 với chân 9. Chân 9 trở thành A7 trên DRAM 64K. Chân 1 là A8 trên DRAM 256K. Những chip IC này không có chức năng chọn chip, do đó cần phải có logic chọn chip bên ngoài. Tất cả các chip IC này đều có 2 chân nguồn nuôi. Chân 16 nối đất, chân 16 là nguồn Vcc hoặc +5 V.

Hình 2.31 minh họa cấu trúc của một bộ nhớ có sử dụng logic để chọn ô nhớ. Bộ nhớ 16K sử dụng 7 bit để giải mã 128 cột, 7 bit để giải mã 128 hàng. Mỗi một bộ giải mã có 7 bit chốt để xác định địa chỉ bit nhớ trong bộ nhớ, sử dụng hai chu trình.

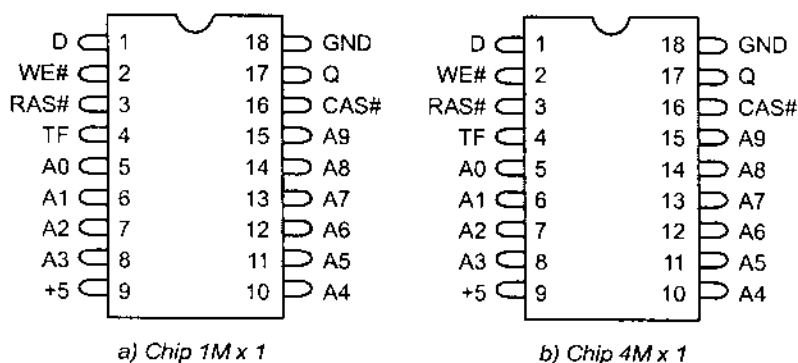
Chu trình 1: 7 đường địa chỉ thấp (A0-A6) của hệ thống nhớ được đưa tới 7 chân đầu vào của bộ chốt giải mã hàng. Tín hiệu xung RAS# (Row Address Strobe#) được đưa vào, tín hiệu này cho phép giữ 7 bit địa chỉ thấp trong chốt giải mã hàng.



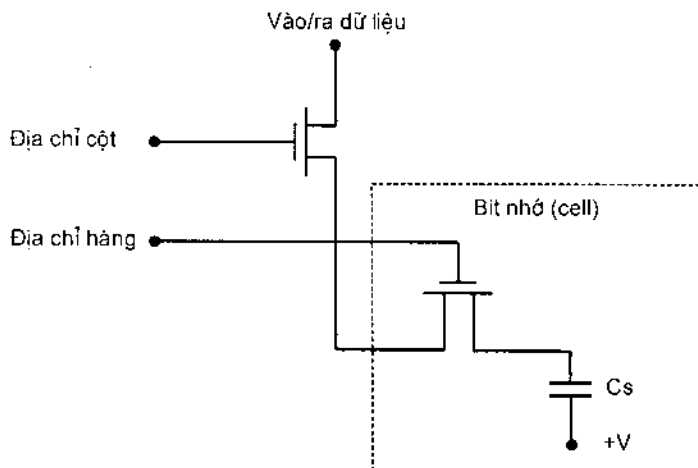
Hình 2.31: Chip DRAM $16K \times 1$ sử dụng địa chỉ hàng, cột ghép.
 Các bit A0-A6 đưa vào chốt hàng nhờ tín hiệu RAS#,
 các bit A7-A13 đưa vào chốt cột nhờ tín hiệu CAS#

Chu trình 2: 7 đường địa chỉ cao (A7-A13) được nối với 7 chân đầu vào của bộ chốt giải mã cột. Khi đưa vào tín hiệu xung CAS# (Column Address Strobe#), tín hiệu này cho phép giữ 7 bit địa chỉ cao trong chốt giải mã cột. Tất cả 14 bit địa chỉ được lưu giữ trong các chốt để thực hiện giải mã cột và hàng chọn một bit nhớ trong 16.386 bit. Kỹ thuật này được gọi là địa chỉ ghép (Multiplexed address). Các mạch nhớ 1 Mbit và 4 Mbit vào/ra nối tiếp cũng được sử dụng trong bộ vi xử lý (hình 2.32).

Hình 2.33 là một dạng ô nhớ động đơn giản được chế tạo phổ biến cho các vi mạch DRAM với một đường dây dữ liệu chung cho cả ghi và đọc (Data IN/OUT). Tụ điện Cs là ô bit nhớ. Giá trị điện dung khoảng 10^{-15} Fa-ra được hình thành giữa đế vi mạch và bản cực transistor CMOS.



Hình 2.32: Hai chip DRAM một đầu vào/một đầu ra có thể thay thế được

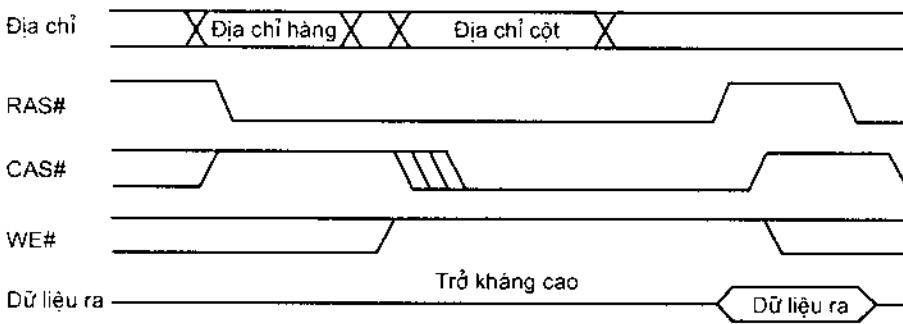


Hình 2.33: Bit nhớ động có kết nối chọn hàng, cột

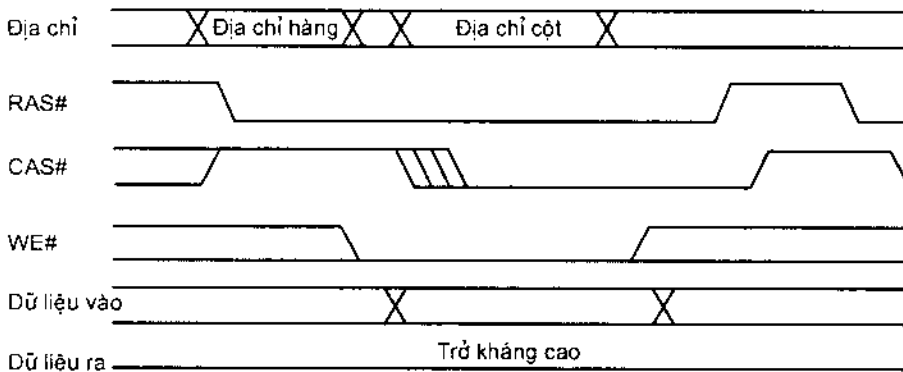
Để minh họa cho kết nối DRAM, sử dụng chip nhớ DRAM 4×1 (4 bit) với loại cấu trúc bit như trong hình 2.33, ta có thể xây dựng một sơ đồ kết nối DRAM 4×1 như trong hình 2.34. Địa chỉ A0 là địa chỉ hàng, địa chỉ A1 là địa chỉ cột. Các địa chỉ này được chốt nhờ các mạch chốt hàng và chốt cột và sườn xuống của tín hiệu nhịp địa chỉ hàng RAS# (Row Address Strobe#) và của tín hiệu CAS# (Column Address Strobe#). Trong mọi trường hợp: ghi/đọc và làm tươi, tín hiệu RAS# đều xuống ở mức tích cực thấp. Nhưng khi ghi/đọc dữ liệu thì có thêm CAS# xuống mức tích cực thấp. Khi làm tươi CAS# luôn ở mức cao và chỉ có RAS# thay đổi xuống mức thấp để làm các tụ điện

= 15.625 μ s. Nếu thời gian chu kỳ làm tươi là 200 ns, thì lượng thời gian cần thiết để làm tươi là 128×200 ns. Tỷ lệ phần trăm của thời gian tồn tại là: $(128 \times 200 \text{ ns} / 2 \text{ ms}) \times 100 = 1,28\%$. Cho một DRAM 256K $\times$ 1 với 256 hàng thì làm tươi cần phải xuất hiện từng $4 \text{ ms} / 256 = 15,6$ ns.

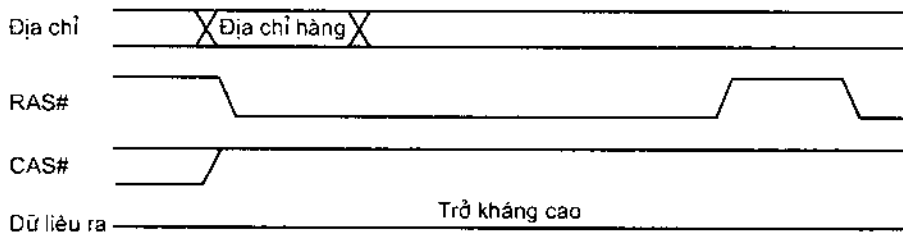
Các chu kỳ đọc, ghi và chu kỳ làm tươi DRAM được mô tả ở hình 2.35.



a) Chu kỳ đọc RAM



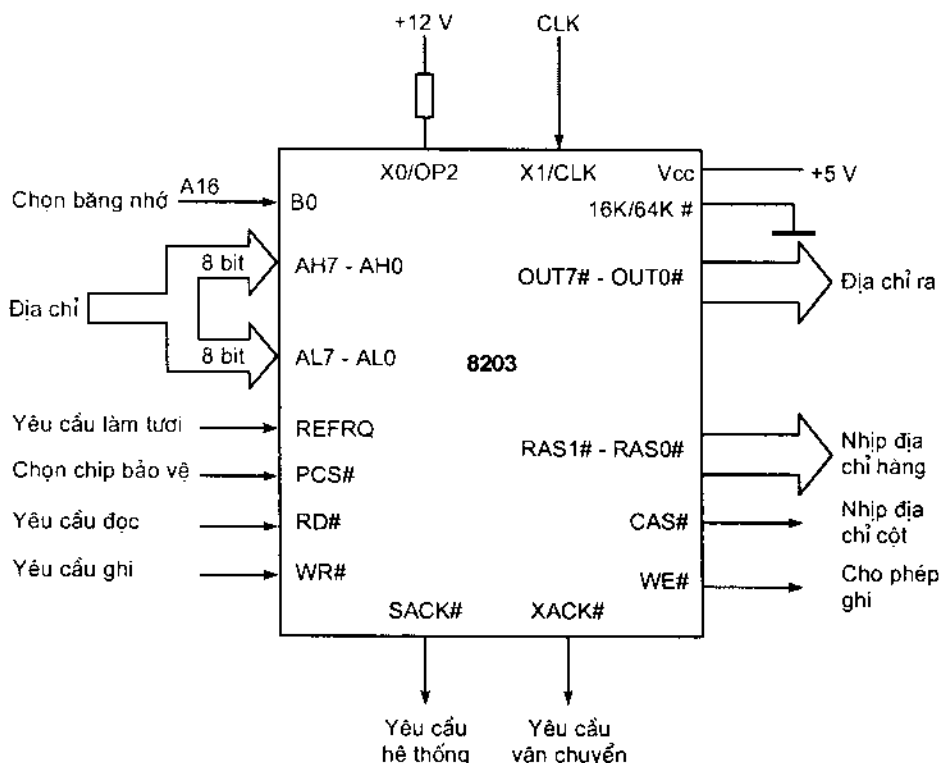
b) Chu kỳ ghi RAM



c) Chu kỳ làm tươi RAM

Hình 2.35: Các đồ thị chu kỳ ghi/đọc/làm tươi DRAM

DRAM tuy rẻ nhưng phức tạp hơn SRAM trong điều khiển và cần phải có logic điều khiển làm tươi. Do đó, kèm theo các chip DRAM còn có các chip điều khiển DRAM riêng. Ví dụ: chip điều khiển DRAM 8203 dùng cho các loại DRAM 16K và 64K có sơ đồ khối được mô tả trong hình 2.36.



Hình 2.36: Chip điều khiển 8203 cho các loại DRAM 16K và 64K

Để điều khiển, phân chia DRAM thành 2 băng nhớ, mỗi băng được xác định bằng bit địa chỉ A16 (trong trường hợp 64K). Chip điều khiển tạo ra các nhịp chọn địa chỉ cột và hàng, các địa chỉ hàng để làm tươi, phân định sử dụng DRAM ở chế độ ghi đọc bình thường và chế độ làm tươi nhờ các tín hiệu yêu cầu đọc (RD#), ghi (WR#) và làm tươi (REFRQ). Nó gửi các tín hiệu SCAK#, XACK# cho logic chờ trong hệ vi xử lý kết nối DRAM chậm.

(3) Các module nhớ

Ngày nay, công nghệ RAM đã cho chúng ta những chip DRAM dung lượng lớn hơn 1 MB, do đó bộ nhớ RAM của máy tính cá nhân được tổ chức theo các module nhớ, mỗi module nhớ là một thẻ gồm có các chip nhớ (DRAM, hay SRAM). Có nhiều loại module nhớ khác nhau về dung lượng và thời gian truy cập cũng như ứng dụng của chúng cho các thế hệ máy tính cá nhân.

- SIMM 30 chân:

SIMM (Single In-line Memory Module) là loại thẻ nhớ có 1 hàng để cắm 30 chân để cắm trên khe mở rộng của bo mạch chủ máy tính cá nhân. Chúng có dung lượng nhớ thấp, ví dụ:

4 MB SIMM với Parity: thời gian truy cập 60 ns, 3 chip SRAM

16 MB SIMM với Parity: thời gian truy cập 60 ns, 9 chip SRAM.

- EDO SIMM 72 chân:

Loại thẻ SIMM được sử dụng thông dụng cho các thế hệ máy tính cá nhân từ thế hệ 32-bit (từ 486 đến nay). Chúng có một số loại dung lượng khác nhau: 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, và sử dụng các chip DRAM dung lượng 1 MB, 2 MB, 4 MB, với thời gian truy cập 60 ns và nguồn nuôi là 3,3 V.

- EDO DIMM 72 chân và 144 chân:

Loại thẻ DIMM (Dual In-line Memory Module) với 2 hàng chân có dung lượng nhớ không lớn và được sử dụng cho các thế hệ máy tính cá nhân từ 386: 4 MB, 8 MB, 16 MB, 32 MB, thời gian truy cập 60 ns, và sử dụng các chip DRAM dung lượng 1 MB, 2 MB, 4 MB.

- SDRAM PC66 SIMM 144 chân:

Loại thẻ này có các dung lượng 16 MB, 32 MB, 64 MB, 128 MB, 256 MB, và dùng các chip SDRAM dung lượng 2 MB, 4 MB hỗ trợ PC bus 66 và nguồn nuôi 3,3 V.

- SDRAM PC100 và PC133 DIMM 144 chân:

Loại thẻ này có các dung lượng 32 MB, 64 MB, 128 MB, 256 MB, 512 MB và dùng các chip SDRAM dung lượng 4 MB, 8 MB, 16 MB, 32 MB hỗ trợ cho PC bus 100/133 và nguồn nuôi 3,3 V.

- EDO DIMM 168 chân:

Loại thẻ này có các dung lượng 16 MB, 32 MB, 64 MB và dùng các chip DRAM dung lượng 2 MB, 4 MB, 8 MB với nguồn nuôi 5 V.

- DIMM W/o Buffer 64 bit PC133 SDRAM 168 chân:

Loại thẻ này có các dung lượng 64 MB, 128 MB, 256 MB, 512 MB và sử dụng các chip SDRAM dung lượng 8M×64bit/8MB, 32M×64bit/32MB, nguồn nuôi 3,3 V và hỗ trợ PC bus 133.

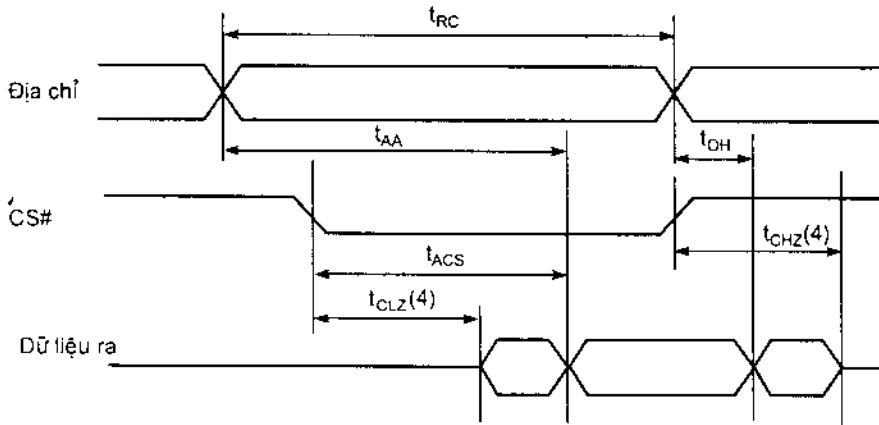
- DIMM W/o Buffer 72 bit PC133 SDRAM 168 chân:

Loại thẻ này có các dung lượng 64 MB, 128 MB, 256 MB, 512 MB và sử dụng các chip SDRAM dung lượng 16M×72bit/16MB, 32M×72bit/32MB, 64M×72bit/64M×4, nguồn nuôi 3,3 V và hỗ trợ PC bus 133.

Còn có nhiều loại module dung lượng và đầu cắm khác nhau nữa. Tuy vậy, chúng ta nhận thấy rằng các loại chip nhớ ngày nay đã có dung lượng lớn, đến 64M×64bit, 64M×72bit và 32 MB.

(4) Hiệu suất của RAM bán dẫn

Chi phí (cost), độ tin cậy (reliability), độ sẵn sàng (availability), kích thước và dung lượng (size and capacity), và tốc độ (speed) là 5 yếu tố quan trọng nhất để đánh giá hiệu suất các vi mạch RAM (SRAM, DRAM) trên thị trường tiêu thụ hàng loạt. Vì công suất của các bộ vi xử lý ngày càng mạnh nên công nghệ chế tạo RAM cũng cần phải theo kịp và đáp ứng yêu cầu ghép nối với các bộ vi xử lý. Đặc biệt xét theo yếu tố tốc độ, chúng ta thường xác định thời gian truy cập của vi mạch RAM (access time). Ví dụ đồ thị xung chu trình đọc dữ liệu của vi mạch SRAM 64K×4bit (hình 2.37).



Hình 2.37: Chu trình đọc SRAM 64K×4 bit

Thời gian truy cập SRAM 64K×4bit (51258) để đọc thông tin là quãng thời gian kéo dài từ khi bắt đầu chu trình đọc (khi địa chỉ bắt đầu được áp vào SRAM) cho tới thời điểm khi các dữ liệu đã ở trên các chân ra (output pins): I/O4 - I/O1. Đó chính là thời gian t_{AA} . Thời gian truy cập SRAM 51258 vào khoảng từ 20 ns đến 35 ns.

Bằng công nghệ MOS mới, từ N-MOS đến BiCMOS, có thể đạt được thời gian truy cập RAM thấp hơn 10 ns. Nếu sử dụng công nghệ Gallium Arsenide (GaAs) với sự tăng chuyển động của các điện tử lên gấp 5 lần, thì thời gian truy cập có thể đạt đến mức ngưỡng là 1 ns. Mục đích cuối cùng là đạt được tốc độ ánh sáng. Dựa vào dao động/hiệu ứng ngưỡng ta có thể đạt tới tốc độ khai thác 5 Gbit/s.

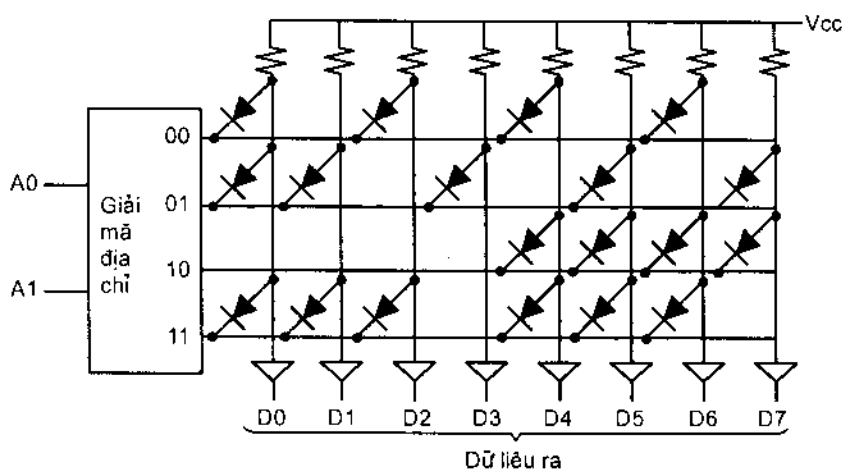
Một yếu tố liên quan đến hiệu suất của RAM là tiêu thụ điện và CMOS - thường được gọi là công nghệ lý tưởng xét về tiêu thụ điện, vì nó tiêu thụ rất ít công suất.

2.3.3. Các bộ nhớ chỉ đọc (ROM, EPROM, EAROM, EEPROM, FLASH Memory)

(1) ROM

ROM (Read Only Memory) là bộ nhớ chỉ đọc. Nó là một phần rất quan trọng đối với bất kỳ hệ vi xử lý nào. ROM là thiết bị nhớ đặc

biệt không thể thay đổi được khi đã nằm trong thiết bị. Nó được nhà sản xuất ghi sẵn nội dung bằng thiết bị đặc biệt. Nó có thể được chế tạo bằng công nghệ lưỡng cực hoặc MOS, ROM lưỡng cực có thời gian truy cập nhanh hơn, tính năng kích hoạt tốt hơn, ROM công nghệ MOS có thành phần chủ yếu là các transistor trường. Một hệ thống thường có nhiều ROM đủ để xây dựng chương trình để nạp vào RAM cùng những chương trình từ bộ nhớ ngoài như đĩa cứng, băng từ.



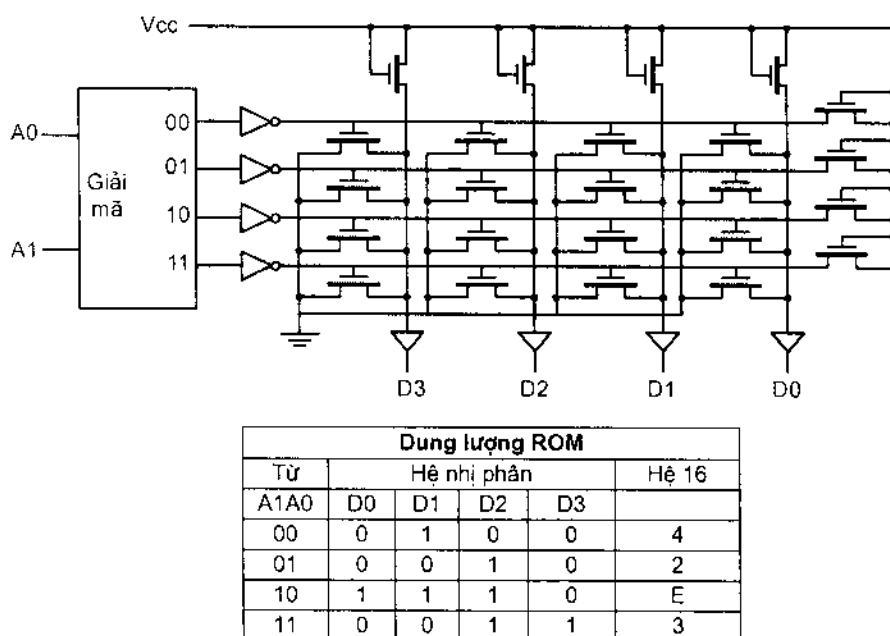
Dung lượng ROM									
Từ	Hệ nhị phân								Hệ 16
A1A0	D0	D1	D2	D3	D4	D5	D6	D7	
00	0	1	0	1	0	1	0	1	55
01	0	0	1	0	1	0	1	0	2A
10	1	1	1	1	0	0	0	0	F0
11	0	0	0	1	0	0	0	1	11

Hình 2.38: ROM loại di-ốt dung lượng 4 byte

Hình 2.38 là một chip ROM rất đơn giản cấu tạo bởi ma trận các di-ốt và một bộ giải mã TTL chọn 4 hàng. ROM này chứa 4 từ 8-bit hay có dung lượng 32 bit. Hai đường dây địa chỉ (A1, A0) thực hiện giải mã 4 hàng di-ốt. Khi một hàng được chọn thì hàng đó có mức logic "0", do đó các di-ốt của hàng được chọn thông, chúng có điện trở thuận nhỏ không đáng kể nên các bit tương ứng với cột có di-ốt thông của hàng được chọn sẽ có giá trị "0". Bảng dung lượng ROM

có giá trị byte dữ liệu đọc ra tương ứng với tổ hợp từ chọn địa chỉ hàng A1A0.

Ví dụ nếu tổ hợp vào A1A0 = 00 thì hàng 0 được chọn, byte dữ liệu đọc ra (D0-D7) có giá trị nhị phân tương ứng là: 01010101, và giá trị hệ 16 là 55h. Nếu tổ hợp vào A1A0 = 11 thì hàng thứ 4 được chọn, byte dữ liệu đọc ra (D0-D7) sẽ có giá trị nhị phân tương ứng là: 00010001, và giá trị hệ 16 là 11h.



Hình 2.39: NMOS ROM dung lượng 4×4 bit

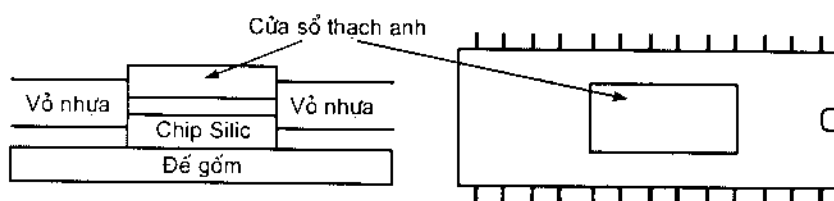
Hầu hết công nghệ ROM ngày nay là công nghệ MOS, cả NMOS và CMOS. Hình 2.39 là một loại ROM công nghệ NMOS, trong đó, thay vào chỗ đi-ốt là transistor. Dung lượng của ROM có 4 từ 4-bit, nghĩa là 16 bit.

(2) EPROM

EPROM (Erasable Programmable Read Only Memory) được xóa nhờ nguồn tia cực tím UV (Ultraviolet light) chiếu trên các transistor và có thể được lập trình ghi bằng xung điện thế cao. Các giá trị logic

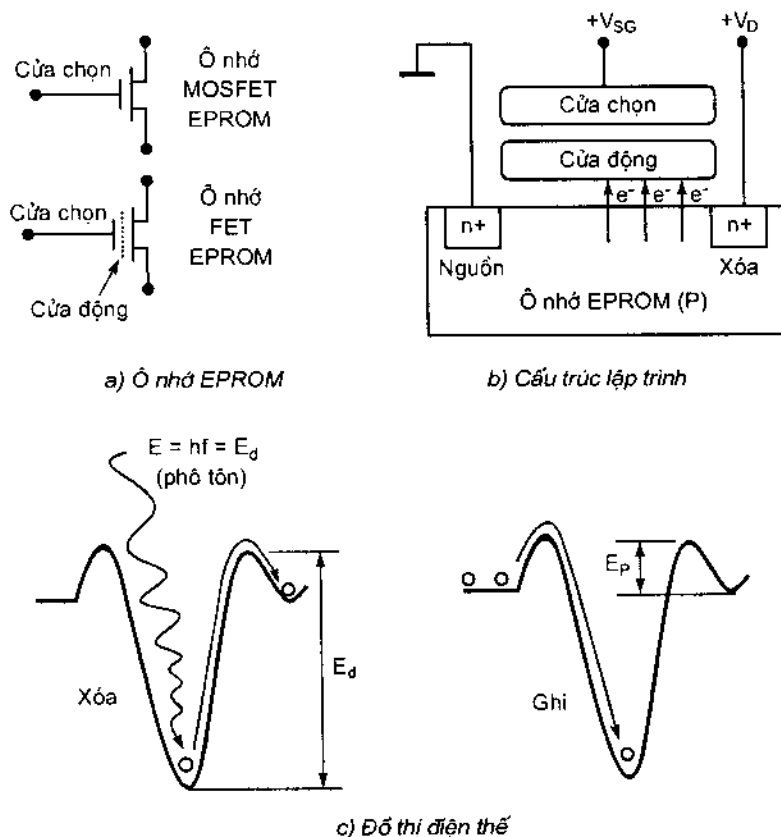
được ghi ngược lại với giá trị xoá. Khi sử dụng EPROM trong máy tính, nó chỉ có thể được đọc dữ liệu (erasable programable read only memory).

Hình 2.40 mô tả cấu trúc của EPROM sử dụng nguồn tia cực tím (UV: UltraViolet) để xoá. Tia cực tím chiếu qua cửa sổ và xoá dữ liệu trong EPROM.



Hình 2.40: Cấu trúc của một loại EPROM

Hình 2.41 giải thích sự thay đổi tín hiệu trong cấu trúc của EPROM khi lập trình. Để bật transistor của ô bit, cả hai cửa: cửa chọn (select gate), cửa động (floating gate) (hình 2.41 a, b) phải được nạp điện tích. Nếu cửa động không được nạp thì transistor của ô bit không thể bật được, ngay cả khi cửa chọn có điện thế tích cực dương. Trạng thái bật của transistor của ô bit có trạng thái logic "0". Transistor đóng (không bật) có trạng thái logic "1". Để nạp cửa động và lập trình ô bit đạt mức logic "0", một điện thế cao ($+V_{SG}$) được áp đặt vào cửa chọn đủ để đẩy các điện tử (e^-) (hình 2.41 a) đi xuyên qua rào chắn (barrier). Mức điện áp phải đủ để vượt qua rào chắn gọi là E_p (hình 2.41 c), và được lập trình EPROM tạo ra. Các điện tử vượt qua rào chắn được cửa động hút về và sâu vào bên trong, mà ở đó chúng bị rơi vào bẫy sâu, và chúng bị giữ lại trong bẫy sâu rất lâu (cửa động tích điện) có thể hàng trăm năm không thoát ra được. Hay nói cách khác, trạng thái logic "0" của ô bit có thể duy trì hàng trăm năm. Để xóa ô bit, cần phải đẩy các điện tử ra khỏi bẫy của cửa động. Để làm việc xoá này, sử dụng các photon của tia cực tím mới năng lượng là $E_d = E = hf$ sao cho điện tử nhận được đủ năng lượng vượt ra ngoài bẫy, ta gọi trường hợp này là cửa động không tích tụ điện tử mà phóng điện tử đi. Đó là nguyên tắc đơn giản của ghi và xóa EPROM.

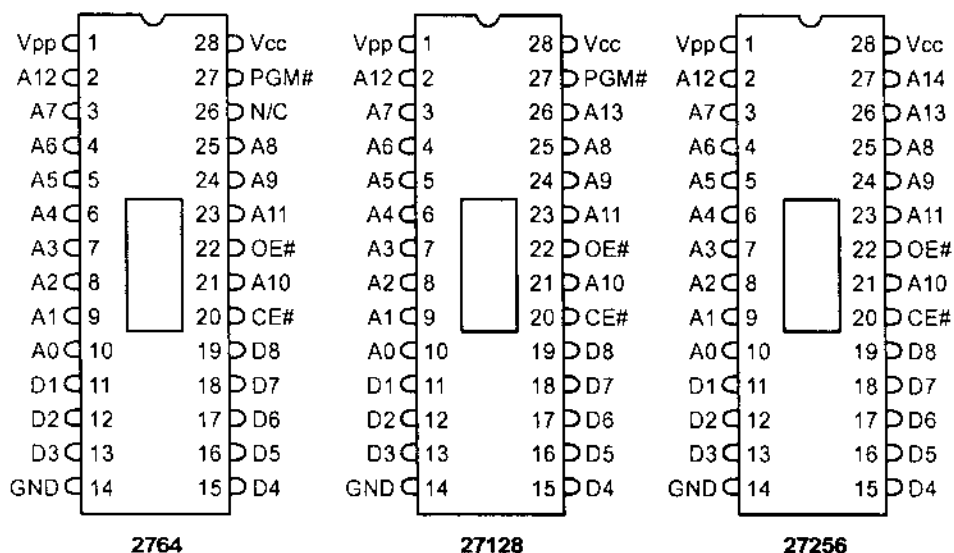


Hình 2.41: Lập trình EPROM

Hình 2.42 là 3 loại EPROM 8-bit và 16-bit thông dụng. Chân V_{pp} (programming output) không được sử dụng trong chế độ đọc. Nó được dành riêng cho lập trình ghi EPROM. Vi mạch 2764 có dung lượng 64 kbit ($8 \text{ kbyte} \times 8$), vi mạch 27128 có dung lượng 128 kbit ($16 \text{ kbyte} \times 8$), và vi mạch 27256 có dung lượng 256 kbit ($32 \text{ kbyte} \times 8$). Các vi mạch này có số chân giống nhau, điện thế nuôi và xung ghi giống nhau nên chúng tương thích với nhau, có thể cắm trên một loại đế cắm. Chúng cũng tương thích với các vi mạch SRAM. Cửa sổ bằng thạch anh cho phép thời gian xóa bằng tia UV trong khoảng 15 đến 20 phút.

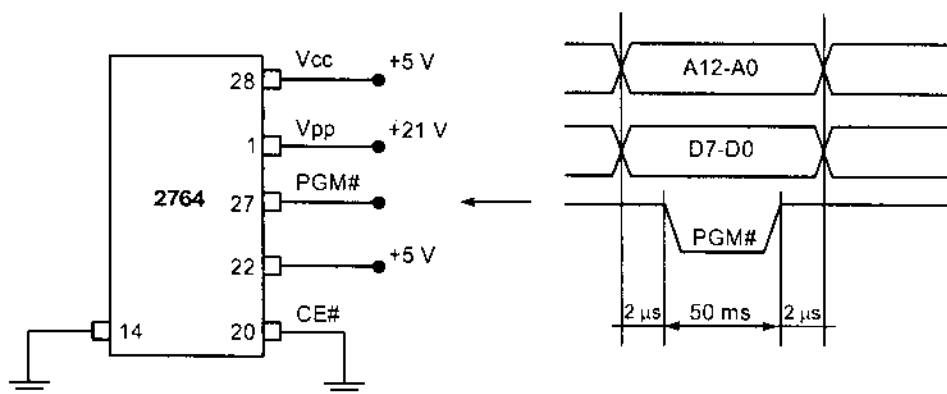
EPROM 2764 $8K \times 8$ là một trong những chip nhớ được sử dụng rộng rãi. Nó có thời gian truy cập (Access time) khoảng 180 ns, nguồn cung cấp tiêu chuẩn là $V_{cc} = +5 \text{ V}$. Để tăng tốc độ lập trình EPROM

2764 chúng ta thường phải xây dựng một chương trình với thuật toán để đảm bảo rằng đủ thời gian ghi mà không có thêm xung ghi thừa nào nữa với từng ô (bit) nhớ. Xung ghi (PGM#) đối với loại EPROM 2764 8K×8 kéo dài trong khoảng 50 ms. Các EPROM trước khi được ghi phải được xóa bằng nguồn tia cực tím. Đôi khi những loại EPROM chất lượng kém nếu để lâu ngoài ánh nắng mặt trời cũng có thể bị xóa.

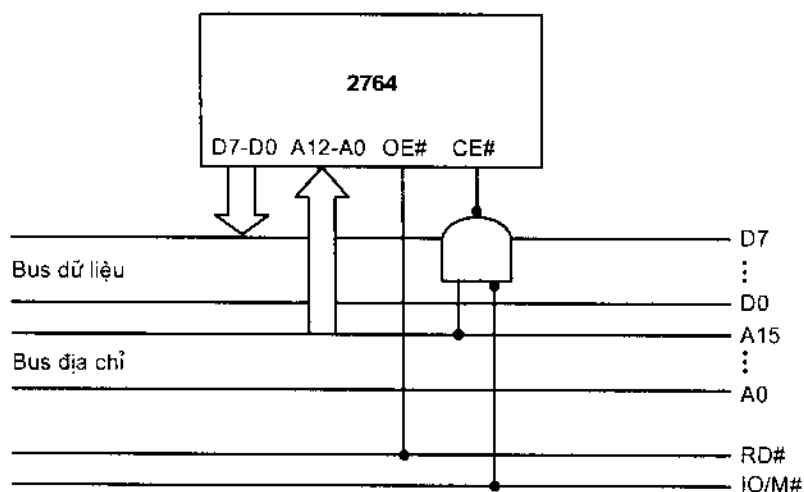


Hình 2.42: Các vi mạch EPROM 2764, 27128, 27256 công nghệ MOS

Hình 2.43 minh họa chế độ các tín hiệu, nguồn và đồ thị xung khi thực hiện lập trình ghi thông tin cho EPROM 2764 8K×8. Sau khi đã hoàn thành quá trình ghi thông tin, ta có thể thực hiện kết nối chip EPROM với bus hệ thống. Nếu ta đặt 8K×8 EPROM thuộc vùng nhớ có địa chỉ từ 80h đến 9Fh thì sơ đồ đấu nối giải mã địa chỉ phải dùng đến địa chỉ A15 (hình 2.44). Có các EPROM mật độ cao hơn như 128K×8, 256K×8, 512K×8. Các loại EPROM này được chế tạo theo công nghệ CHMOS có thời gian truy cập 150 ns. Hiện nay, dung lượng EPROM ngày càng lớn hơn tạo thuận lợi cho cài đặt các chương trình BIOS của các hệ điều hành lớn trong các hệ thống máy vi tính. Cấu trúc kết nối bên trong của EPROM 512K×8 (4M) 27C040 CHMOS được trình bày trong hình 2.45.



Hình 2.43: EPROM 2764 8K×8 trong chế độ lập trình (theo byte)

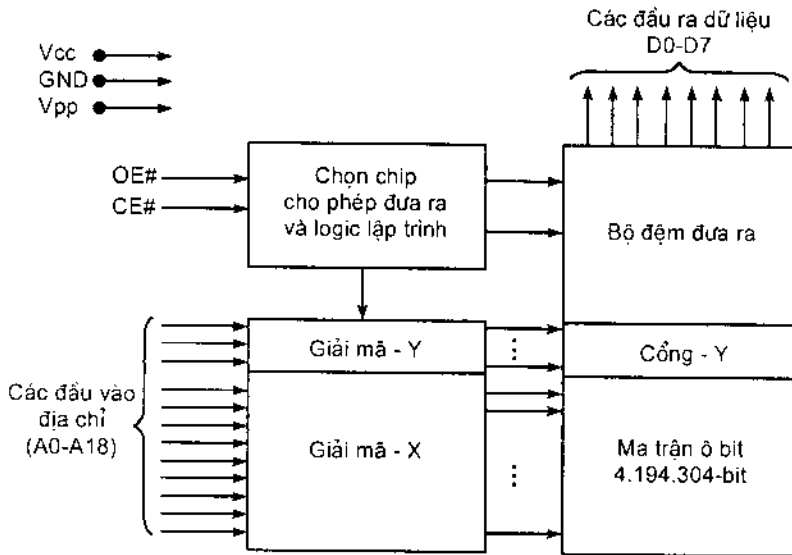


Hình 2.44: Kết nối EPROM 2764 với bus hệ thống

(3) EAROM

EAROM (Electrically Alterable Read Only Memory) được chế tạo tương tự như EPROM, có thể lập trình được và ưu việt hơn EPROM là xóa được bằng mạch điện tử và điện thế mức thấp. Nội dung của EAROM có thể được thay đổi nhờ sử dụng các tín hiệu điện trong một khoảng thời gian nào đó. Nó thường được đặt ngay trong máy tính, và thời gian lưu giữ tín hiệu ghi vào khoảng vài ms. EAROM được sử dụng trong các thiết bị điện tử dân dụng có áp dụng

kỹ thuật vi xử lý, ví dụ như trong bộ chọn kênh (selector) của ti vi. EAROM không bền bằng EPROM nhưng đắt hơn EPROM.



Hình 2.45: CHMOS EPROM 27040 4M (512Kx8)

(4) EEPROM

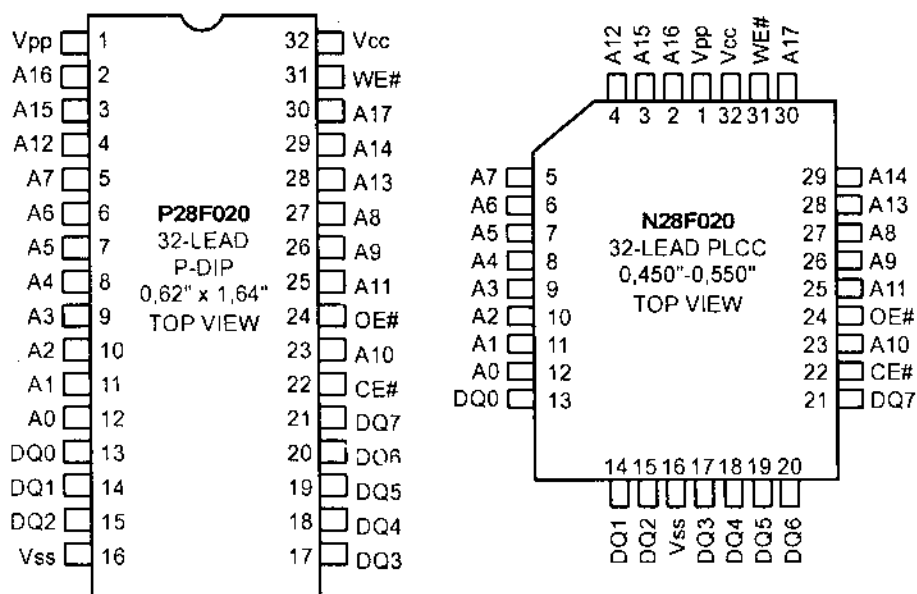
EEPROM (Electrically Erasable PROM) tương tự như PROM nhưng có thể ghi được bằng tín hiệu điện, đây chính là ưu điểm hơn hẳn so với EPROM, bởi EPROM muốn ghi được trước hết phải xóa bằng nguồn tia cực tím. Như vậy nội dung thông tin mới có thể ghi vào EEPROM thông qua các mạch điện tử cấy ngay trên bảng mạch của máy vi tính mà không cần phải tháo các vi mạch EEPROM ra khỏi bảng mạch CPU của máy vi tính. Do đặc điểm công nghệ này, nên các chip EEPROM có dung lượng nhớ thấp, giá rất đắt, và chúng thường được sử dụng trong các hệ thống thiết bị tính toán chuyên dụng.

(5) Flash Memory

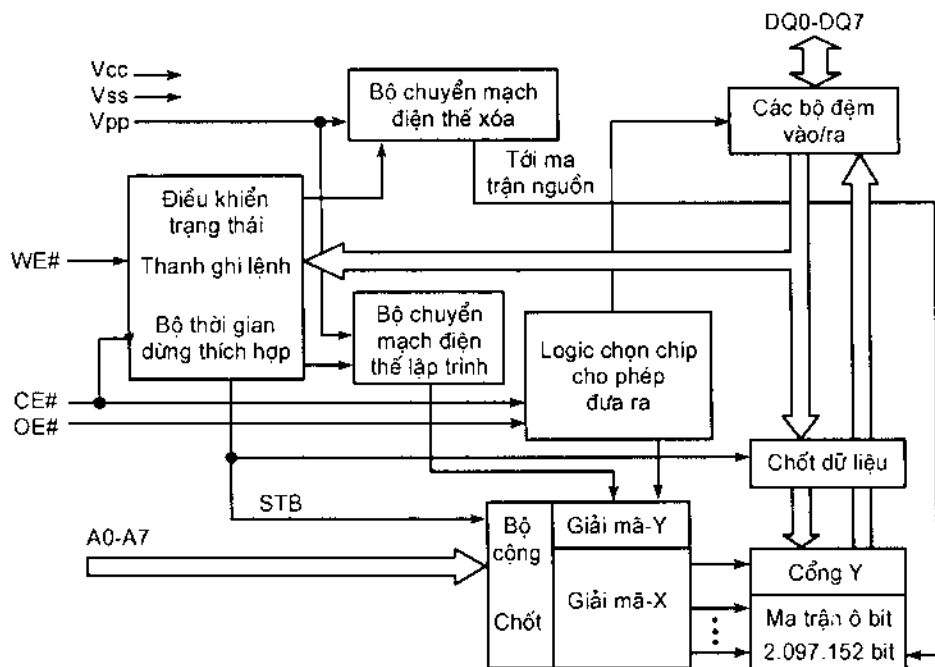
Nhược điểm của EPROM là khi muốn thay đổi nội dung phải bỏ EPROM ra khỏi mạch điện để thao tác xóa bên ngoài bằng nguồn tia cực tím rồi bằng mạch lập trình chuyên dụng ghi lại nội dung.

EEPROM tuy thực hiện xóa được bằng tín hiệu điện nhưng với thời gian lâu, quá trình cũng được thực hiện với tốc độ chậm.

Flash memory xuất hiện trong những năm 1990, và là loại EEPROM đặc biệt. Trước khi ghi nội dung vào Flash memory, cần phải thực hiện xóa trước. Cũng giống như EPROM, không thể xóa theo từng bit được mà phải xóa toàn bộ nội dung bộ nhớ cùng một lúc, nhưng không phải bằng nguồn cực tím bên ngoài, mà tín hiệu điện và nhanh hơn so với EEPROM bình thường (nhanh như ánh chớp, Flash). Một số loại Flash memory có thể cho phép xóa từng gian nhớ, hay đoạn nhớ, mỗi đoạn có thể có dung lượng 16 kbyte, 64 kbyte. Khi thực hiện xóa một đoạn nhớ, những đoạn nhớ khác không bị ảnh hưởng gì cả. Tuy nhiên, không phải loại Flash memory nào cũng có cơ chế phân đoạn cho lập trình xóa. Tất cả các bit của Flash memory khi xóa phải được thiết lập giá trị "0". Sau đó, cũng giống như EEPROM, những bit có giá trị "1" sẽ được ghi một vài lần để đảm bảo ổn định giá trị, nhưng tốc độ xóa nhanh hơn nhiều. Những bit giá trị "0" không cần phải tác động ghi gì cả. Flash memory có thời gian truy cập dao động trong khoảng 45ns - 70ns, nhanh hơn đa số các bộ nhớ DRAM cùng thời. Nó giống như EPROM, không giới hạn số các chu kỳ ghi. Do đó chúng được sử dụng nhiều trong các thiết bị có ứng dụng kỹ thuật vi xử lý, như máy tính, các thiết bị truyền dẫn, các bộ định tuyến, các thiết bị kết nối mạng,... là những thiết bị luôn phải thay đổi cấu hình phần mềm hệ thống. Flash memory còn được sử dụng nhiều trong các loại thẻ tín dụng (credit-card) với dung lượng 20 MB. Bước đầu Flash Memory đắt hơn đĩa cứng với cùng dung lượng, càng về sau này, dung lượng cao lên nhưng giá thì giảm xuống. Có thể nói sự ra đời và phát triển Flash Memory tạo ra sự phát triển lớn nhiều chủng loại thiết bị điện tử hiện đại khác nhau với các khả năng linh hoạt và dễ dàng về thay đổi chức năng, cấu hình,...



Hình 2.46: Đóng vỏ các chip CMOS Flash memory 28F020 256 kbyte



Hình 2.47: CMOS Flash Memory 28F020 256 kbyte

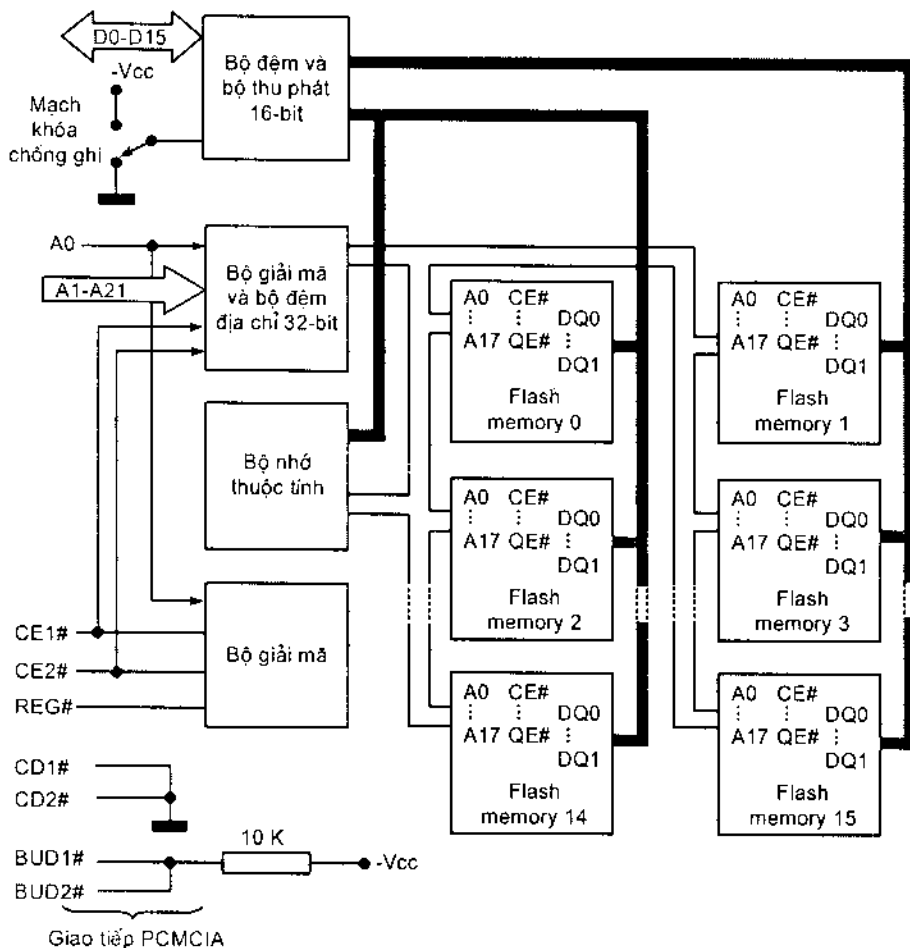
Các hình 2.46, 2.47 là sơ đồ về chip nhớ Flash công nghệ CMOS 28F020 256 kbyte (đóng vỏ 32 chân plastic DIP hoặc 32-lead PLCC). Nó có thời gian truy cập là 150 ns, thời gian xoá toàn bộ chip trong 2 giây, thời gian ghi chip theo chương trình là 4 giây, và tiêu thụ công suất thấp. Điện thế $V_{pp} = +12 V_{DC}$ dùng để xoá và lập trình các ô nhớ. Điện thế $V_{cc} = +5 V_{DC}$.

Thông thường, các chip Flash memory kết hợp với nhau để tạo ra hệ thống nhớ có dung lượng nhớ nhiều hơn. Có một loại đóng gói cho flash memory là PCMCIA (Personal Computer Memory Card Internal Association). Hình 2.48 là sơ đồ của thẻ PCMCIA với hệ thống nhớ dung lượng 4 MB sử dụng 16 chip Flash memory 256 kbyte. Nó có 1 chip EEPROM 512 byte để chứa thông tin về thẻ tương ứng với chuẩn của PCMCIA (CIS).

(6) PCMCIA

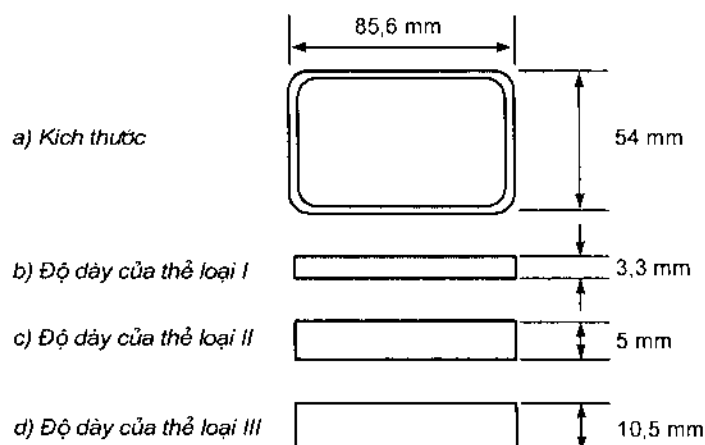
PCMCIA là một chuẩn cho các thiết bị ngoại vi kết nối với các máy tính cá nhân và các hệ thống có vi xử lý khác. Thẻ PCMCIA có kích thước tương tự như các thẻ tín dụng chuẩn; độ dày của thẻ PCMCIA có thể khác nhau tùy thuộc vào từng loại. Các loại thẻ PCMCIA ban đầu được coi như một cách để bổ sung dung lượng nhớ trong (RAM) cho các loại máy tính xách tay và những thiết bị nhỏ dùng kỹ thuật vi xử lý, bởi vì những hệ thống này có không gian cho mở rộng cấu hình rất nhỏ hẹp. Thẻ PCMCIA cũng là cách để bổ sung dung lượng ngoài thay vì phải dùng đến các thiết bị đĩa to và công kênh. Ngày nay, thẻ PCMCIA được coi là chuẩn cho mở rộng dung lượng nhớ cả bên trong và bên ngoài và được dùng trong các máy tính xách tay. Flash memory trong thẻ PCMCIA có thể từ 4 MB đến 256 MB.

Hiện nay có 3 loại thẻ PCMCIA phân biệt theo bề dày (hình 2.49). Loại gốc ban đầu, loại I có độ dày là 3,3 mm. Thẻ loại II dày 5 mm, là loại được dùng phổ biến nhất. Thẻ loại III dày 10,5 mm, và có thể dùng cho mở rộng bộ nhớ đĩa nhỏ bé.



Hình 2.48: Thẻ PCMCIA có dung lượng nhớ 4 MB với 16 Flash memory 256 kbyte. Có khóa chống ghi nhằm bảo vệ dữ liệu

Chuẩn PCMCIA xác định đầu kết nối thẻ (card connector) và giao thức truyền thông giữa hệ thống vi xử lý và thẻ PCMCIA, ngoài ra nó có thể kết nối được 1 hoặc 2 thẻ PCMCIA loại I hoặc loại II. Một phần của giao thức truyền thông là cấu trúc nhận dạng thẻ CIS (Card Identification Structure). CIS xác định thông tin của thẻ PCMCIA kết nối với hệ thống vi xử lý. CIS được lưu giữ bên trong một bộ nhớ riêng biệt trên thẻ PCMCIA, gọi là bộ nhớ thuộc tính (Attribute memory). Bộ nhớ thuộc tính có thể có dung lượng lên tới 8 kbyte, và nó cung cấp dữ liệu cho hệ thống vi xử lý theo khuôn dạng chuẩn.



Hình 2.49: Thẻ PCMCIA chuẩn

Thẻ PCMCIA thường được sử dụng với máy vi tính, cho nên có các chương trình con điều khiển nó được viết cho MS DOS và cho các hệ Windows. Sử dụng Flash memory trong thẻ PCMCIA làm cho nó rất thông dụng. Dung lượng Flash memory trong thẻ PCMCIA có thể là 4 MB, 8 MB, 16 MB hoặc 64 MB.

Một số loại thẻ PCMCIA được dùng làm thẻ kết nối mạng cục bộ: NIC (Network Communication Card) và truyền thông (dial-up) cho các máy tính xách tay. Trong trường hợp này các đầu kết nối không giống như loại thẻ dùng làm bộ nhớ mở rộng.

Một số loại thẻ PCMCIA lại có thể là các bộ thu phát sóng vô tuyến (radio transmitter-receiver), và chúng thường được gọi là modem vô tuyến (wireless modem).

Bảng 2.1 xác định các chân (pins) tín hiệu của thẻ PCMCIA. Thẻ PCMCIA có 68 chân, chia làm hai mặt thẻ, mỗi mặt có 34 chân.

Bảng 2.1: Các chân tín hiệu của thẻ PCMCIA

Chức năng	Tín hiệu	Chân	Chức năng	Tín hiệu	Chân
Cho phép card	CE1#	7	Địa chỉ 8	A8	12
Cho phép đầu ra	OE#	9	Địa chỉ 9	A9	11
Cho phép ghi	WE#/PGM	15	Khởi tạo lại card	RESET	58
Sẵn sàng/bàn	RDY/BSY#	16	Dữ liệu 0	D0	30
Phát hiện card	CD1#	36	Dữ liệu 1	D1	31
Cho phép card 2	CE2#	42	Dữ liệu 0	D10	66

Chức năng	Tín hiệu	Chân	Chức năng	Tín hiệu	Chân
La chọn thanh ghi	REG#	61	Dữ liệu 1	D11	37
Phát hiện điện thế pin 2	BVD2#	62	Dữ liệu 0	D12	38
Phát hiện điện thế pin 1	BVD1#	63	Dữ liệu 1	D13	39
Phát hiện card	CD2#	67	Dữ liệu 0	D14	40
Địa chỉ 0	A0	29	Dữ liệu 1	D15	41
Địa chỉ 1	A1	28	Dữ liệu 0	D2	32
Địa chỉ 10	A10	8	Dữ liệu 1	D3	2
Địa chỉ 11	A11	10	Dữ liệu 0	D4	3
Địa chỉ 12	A12	21	Dữ liệu 1	D5	4
Địa chỉ 13	A13	13	Dữ liệu 0	D6	5
Địa chỉ 14	A14	14	Dữ liệu 1	D7	6
Địa chỉ 15	A15	20	Dữ liệu 0	D8	64
Địa chỉ 16	A16	19	Dữ liệu 1	D9	65
Địa chỉ 17	A17	46	Chu kỳ bus ngoài	WAIT	59
Địa chỉ 18	A18	47	Nối đất	GND	1
Địa chỉ 19	A10	48	Nối đất	GND	34
Địa chỉ 2	A2	27	Nối đất	GND	35
Địa chỉ 20	A20	49	Nối đất	GND	68
Địa chỉ 21	A21	50	Điện thế lập trình 1	Vpp1	18
Địa chỉ 22	A22	53	Điện thế lập trình 1	Vpp1	52
Địa chỉ 23	A23	54	Làm tươi	RFSH	43
Địa chỉ 24	A24	55	Dự trữ	RFU	44
Địa chỉ 25	A25	56	Dự trữ	RFU	45
Địa chỉ 3	A3	26	Dự trữ	RFU	57
Địa chỉ 4	A4	25	Dự trữ	RFU	60
Địa chỉ 5	A5	24	Điện thế cung cấp	Vcc	17
Địa chỉ 6	A6	23	Điện thế cung cấp	Vcc	51
Địa chỉ 7	A7	22	Chống ghi	WP	53

2.3.4. Các vi mạch ứng dụng đặc biệt

Các vi mạch ứng dụng đặc biệt (ASIC: Application-Specific IC) thuộc vào thế hệ thứ ba của thiết kế logic. Các loại này có những đặc tính tương đối giống như bộ nhớ bán dẫn.

Giai đoạn đầu tiên của thiết kế logic là các mạch logic loại TTL/CMOS54/74/74C với các chip vi mạch thực hiện các hàm logic OR, AND và NOT, gọi là các mạch cổng. Giai đoạn thứ hai của thiết kế logic bắt đầu từ đầu những năm 1970 với sự ra đời các chip vi xử lý có công nghệ tích hợp lớn (LSI) và rất lớn (VLSI). Ngoài các chip vi xử lý còn có hàng loạt các chip vi mạch nhớ và các chip điều khiển ngoại vi. Giai đoạn thứ ba là công nghệ ASIC. Thời kỳ đầu của ASIC ít có sự hỗ trợ của CAD (Computer-Aided Design), do đó chúng có giá thành cao và hạn chế về các chức năng. Sau này, nhờ sự hỗ trợ của

CAD, ASIC đã được quan tâm trở lại và có sự phát triển mạnh, chiếm lĩnh vị trí cạnh tranh so với công nghệ vi xử lý.

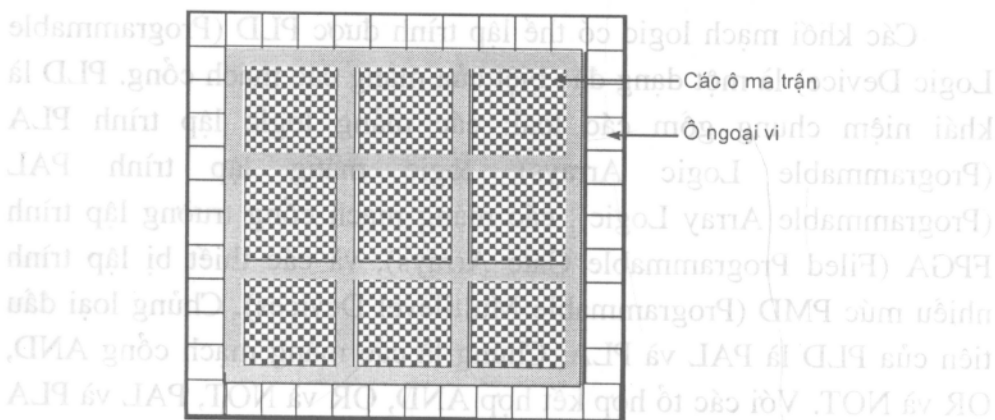
ASIC được phân ra 3 loại:

- Các mảng mạch cổng (Gate Arrays), gồm cả các khối logic có thể lập trình (PLD);
- Các ô chuẩn (Standard Cells);
- Các mạch hoàn toàn chuyên dụng (Full custom) theo yêu cầu của người dùng.

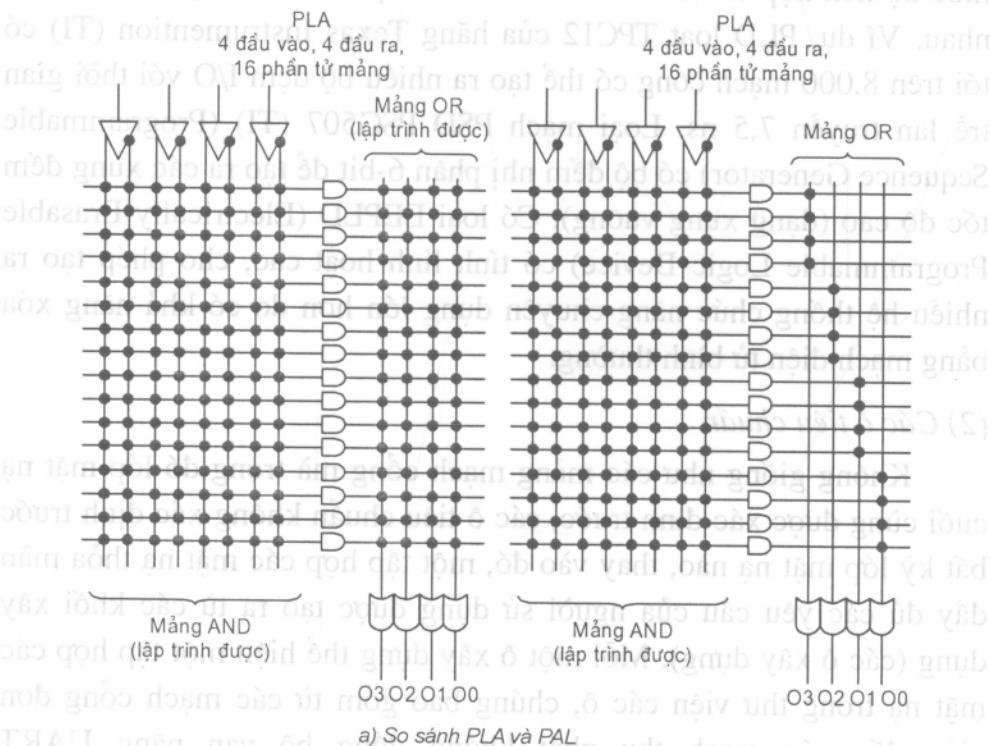
(1) Các mảng mạch cổng

Các mảng mạch cổng có nhiều ứng dụng khác nhau. Chúng được chế tạo theo công nghệ CMOS. Một mảng mạch cổng mức tích hợp CMOS trung bình chứa tới 1.000 ô ma trận (matrix cells), trong mỗi ô ma trận có một số linh kiện không liên kết với nhau như: bóng bán dẫn, điện trở, đi-ốt. Khi kết nối chúng lại theo kết cấu cơ bản, thì mỗi một ma trận có 2 mạch NOR 2 đầu vào. Do đó tổng thể mảng với 1.000 ô ma trận sẽ có tới 2.000 mạch cổng (hình 2.50). Xung quanh mảng là các ô ngoại vi để ghép nối vào/ra với các thiết bị khác ngoài. Khi một số ô ma trận kết hợp lại với nhau thì có thể tạo ra một số mạch chức năng cơ bản, như Flip-Flop, bộ đếm, các Schmitt trigger. Và khi các mạch chức năng cơ bản kết hợp lại với nhau sẽ tạo ra một mảng các mạch cổng thỏa mãn nhu cầu ứng dụng riêng biệt nào đó của người dùng, ví dụ, đồng hồ số đo điện thế-dòng (digital voltmeters), tổng hợp âm thanh (sound synthesizer), điều khiển tốc độ mô-tơ, nhiệt kế số,...

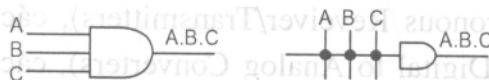
Nhiều loại mảng mạch cổng công nghệ CMOS có độ phức tạp hơn, chứa các mạch chức năng tương tự/số như các tụ điện CMOS, các đi-ốt zener, các bộ chuyển mạch tương tự, các mạch Flip-Flop, các điện trở. Sử dụng rất nhiều các tổ hợp kết nối khác nhau đối với các mạch cơ bản trong mảng, người thiết kế có thể tạo ra các hệ thống tương tự/số như bộ dao động RC, bộ lọc, bộ chuyển đổi tương tự/số (ADC) bằng các mảng mạch cổng, tất cả các bước, từ bước sắp đặt ban đầu đến khi làm mặt nạ để tạo mạch và kiểm tra đều được thực hiện nhờ sự hỗ trợ của CAD và phần mềm chuyên dụng.



Hình 2.50: Mảng mạch cổng gồm 2.000 mạch cổng



a) So sánh PLA và PAL



b) Mạch tương đương

Hình 2.51: Bên trong các mạch PLA và PAL

Các khối mạch logic có thể lập trình được PLD (Programmable Logic Device) là một dạng đặc biệt của mảng các mạch cổng. PLD là khái niệm chung gồm các loại: các mảng logic lập trình PLA (Programmable Logic Arrays), logic mảng lập trình PAL (Programmable Array Logic), các mảng mạch cổng trường lập trình FPGA (Filed Programmable Gate Arrays), và các thiết bị lập trình nhiều mức PMD (Programmable Multilevel Devices). Chúng loại đầu tiên của PLD là PAL và PLA. Chúng là các mảng mạch cổng AND, OR và NOT. Với các tổ hợp kết hợp AND, OR và NOT, PAL và PLA có thể cho những hệ thống chức năng lớn hơn. Những loại PLD có mức độ tích hợp cao có thể có rất nhiều mạch chức năng cơ bản khác nhau. Ví dụ, PLD loạt TPC12 của hãng Texas Instrumention (TI) có tới trên 8.000 mạch cổng có thể tạo ra nhiều bộ đếm I/O với thời gian trễ lan truyền 7,5 ns. Loại mạch PSD PSG507 (TI) (Programmable Sequence Generator) có bộ đếm nhị phân 6-bit để tạo ra các xung đếm tốc độ cao (dạng xung vuông). Có loại EEPLD (Electrically-Erasable Programmable Logic Device) có tính linh hoạt cao, cho phép tạo ra nhiều hệ thống chức năng chuyên dụng lớn hơn do có khả năng xóa bằng mạch điện tử bình thường.

(2) Các ô tiêu chuẩn

Không giống như các mảng mạch cổng mà trong đó lớp mặt nạ cuối cùng được xác định trước, các ô tiêu chuẩn không xác định trước bất kỳ lớp mặt nạ nào, thay vào đó, một tập hợp các mặt nạ thỏa mãn đầy đủ các yêu cầu của người sử dụng được tạo ra từ các khối xây dựng (các ô xây dựng). Mỗi một ô xây dựng thể hiện một tập hợp các mặt nạ trong thư viện các ô, chúng bao gồm từ các mạch cổng đơn giản đến các mạch thu phát không đồng bộ vạn năng UART (Universal Asynchronous Reveiver/Transmitters), các mạch biến đổi số-tương tự DAC (Digital to Analog Converters), các đơn vị số học-logic ALU,... Nhờ kỹ thuật CAD các tập hợp mặt nạ cho từng ô được kết hợp với nhau để tạo ra mặt nạ của một hệ thống chức năng theo

yêu cầu của người sử dụng (khách hàng). Một trong những loại của các ô tiêu chuẩn là loại có chứa bộ vi xử lý lõi (core microprocessor). Các ô tiêu chuẩn chứa các bộ vi xử lý lõi có kích thước và chi phí nhỏ trong khi chúng sử dụng các mạch chức năng cơ bản và đã chuẩn hóa.

(3) Các mạch hoàn toàn theo yêu cầu của khách hàng

Loại mạch này được chế tạo để thỏa mãn những ứng dụng riêng biệt, cụ thể, không có phần cứng nào được thiết kế trước như trong trường hợp chế tạo các mảng mạch cổng, và không có các thành phần mạch nào được thiết kế trước như đối với các ô tiêu chuẩn. Nói cách khác, chúng hoàn toàn được thiết kế theo yêu cầu, tức là chúng phụ thuộc vào yêu cầu chuyên dụng của khách hàng. Khách hàng đặt yêu cầu và cấu trúc cụ thể sẽ được định hình.

2.3.5. Thiết bị nhớ ngoài

(1) Thiết bị nhớ bằng vật liệu từ

Cho đến nay, các vật liệu từ tính như băng từ, đĩa từ được sử dụng rộng rãi để lưu trữ thông tin trong các hệ thống máy tính.

Thông tin ghi trên các vật liệu từ rất dễ bị phá hủy nếu như để gần với một vật có từ tính. Các từ trường ghi thông tin trên băng từ, đĩa từ tương đối yếu. Do đó khi có một tín hiệu từ nào được đưa đến gần chúng cũng có thể xóa hoặc làm sai dữ liệu. Hơn nữa, các tín hiệu được sản sinh ra ở đầu đọc cũng là những tín hiệu rất nhỏ. Điều đó cũng có nghĩa là chúng rất dễ bị hư hỏng do các yếu tố bên ngoài khác. Do đó, hầu hết các thiết bị nhớ bằng băng từ, đĩa từ đều được thiết kế kèm theo các hệ thống sửa lỗi. Trước đây chúng ta đã biết các mạch nhớ và trong kỹ thuật truyền thông đã sử dụng công nghệ kiểm tra lỗi, được gọi là kiểm tra chẵn lẻ. Tuy nhiên, hệ thống kiểm tra lỗi kiểu này không đủ cho các dữ liệu ghi trên thiết bị từ. Quá trình đọc và ghi trên các thiết bị từ đòi hỏi phải có công nghệ kiểm tra lỗi chính xác hơn rất nhiều. Một vài hệ thống đã sử dụng công nghệ này để phát hiện và sửa lỗi.

Băng từ là một dạng thiết bị lưu giữ thông tin từ tính mật độ cao. Trước khi được dùng để lưu giữ thông tin số, băng từ được dùng để ghi các thông tin tương tự dưới dạng các tín hiệu âm thanh. Phải dùng phương thức truy cập tuần tự để lấy thông tin đã được ghi trên băng từ. Ví dụ dữ liệu cần đọc ở giữa băng, để lấy được dữ liệu cần chuyển quay băng đến đoạn giữ, như vậy rất tốn thời gian. Hầu hết các hệ thống băng từ đều có bộ phận tua đi tua lại chạy tốc độ cao, nhưng chỉ nhanh hơn tốc độ khi đọc thông tin từ 2 đến 5 lần. Hơn nữa, ở tốc độ cao, không thể đọc và ghi thông tin được.

Mặc dù có những nhược điểm song hiện nay nó vẫn còn được sử dụng bởi vì kỹ thuật băng từ ngày càng cao đáp ứng được yêu cầu truy cập dữ liệu nhanh hơn, kích thước băng từ càng nhỏ và dung lượng càng lớn. Băng từ thường được dùng trong các hệ thống máy tính lớn hay máy chủ để lưu giữ dự phòng theo kiểu tuần tự hay ghi dữ liệu kiểu lũy tiến. Điểm bất lợi nữa là băng từ cần phải được bảo quản trong điều kiện môi trường không khí sạch, nhiệt độ, độ ẩm phù hợp. Nhiệt độ quá cao hay quá thấp cùng với độ ẩm cao dễ làm hư hỏng băng từ. Hiện nay công nghệ chế tạo có thể cho các cuộn băng từ cát-xét rất nhỏ nhưng dung lượng lưu giữ rất lớn, có thể đạt tới vài GB.

Thiết bị nhớ bằng đĩa từ có dung lượng nhớ cao và tốc độ truy cập nhanh không theo kiểu tuần tự như băng từ mà truy cập trực tiếp ngay tại vùng nhớ. Đĩa từ gồm những lớp vật lý khác nhau với những vòng tròn đồng tâm rất sát nhau, mỗi vòng tròn đó được gọi là rãnh (track) ghi. Những rãnh ghi phân chia thành một số sector. Khi làm việc, đĩa được quay tròn và đầu từ ghi/đọc dữ liệu chạy dọc trên bề mặt đĩa vuông góc với các rãnh ghi. Ví dụ cần lấy dữ liệu ở rãnh 15 mà vị trí hiện hành là rãnh 32 thì chỉ cần đưa đầu từ từ rãnh 32 tới rãnh 15 rồi bắt đầu quá trình đọc chứ không phải đi qua và duyệt tất cả dữ liệu trên các rãnh từ 32, 31,..., 16. Như vậy thời gian truy cập nhanh hơn rất nhiều vì đầu từ chuyển dịch theo đường thẳng (từ rãnh này sang rãnh khác) trên bề mặt của đĩa từ đang quay tròn.

Nếu sử dụng ổ đĩa quay với tốc độ 300 vòng/phút (300 rpm), thời gian đầu từ nhảy từ rãnh này sang rãnh khác là 6 ms, thời gian để xác định rãnh cần chọn (tìm kiếm rãnh) là 15 ms, có tất cả 40 rãnh, tính từ 00. Giả sử đầu từ đang ở rãnh 00, thì:

- Thời gian cần để đầu từ chuyển từ rãnh 00 đến được rãnh 19 là:

$$20 \text{ rãnh} \times 6 \text{ ms/rãnh} = 120 \text{ ms}$$

- Thời gian cần để tìm được rãnh 19 (thời gian xác định rãnh + thời gian chuyển đến rãnh):

$$120 \text{ ms} + 15 \text{ ms} = 135 \text{ ms}$$

- Tốc độ quay của đĩa tính trên 1 s là:

$$300 \text{ rpm} / (60 \text{ s/min}) = 5 \text{ rps, hay } 200 \text{ ms/rev và } 1/2 \text{ vòng quay, hết } 100 \text{ ms.}$$

- Thời gian cần để truy cập dữ liệu rãnh 19 tính từ lúc đầu từ đang ở rãnh 00 và kể cả thời gian trung bình đĩa quay được nửa vòng là:

$$135 \text{ ms} + 100 \text{ ms} = 235 \text{ ms.}$$

Tương tự, nếu đầu từ ở rãnh 00, ta tính cho thời gian truy cập dữ liệu ở rãnh cuối cùng, rãnh thứ 40 là:

- Thời gian cần để đầu từ chuyển từ rãnh 00 đến được rãnh 39 là:

$$40 \text{ rãnh} \times 6 \text{ ms/rãnh} = 240 \text{ ms}$$

- Thời gian cần để tìm được rãnh 40 (thời gian xác định rãnh + thời gian chuyển đến rãnh):

$$240 \text{ ms} + 15 \text{ ms} = 255 \text{ ms}$$

- Thời gian cần để truy cập dữ liệu rãnh 39 tính từ lúc đầu từ đang ở rãnh 00 và kể cả thời gian trung bình đĩa quay được nửa vòng là:

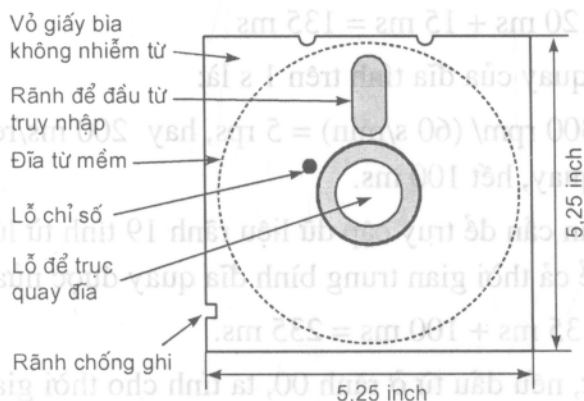
$$255 \text{ ms} + 100 \text{ ms} = 355 \text{ ms.}$$

Đĩa từ không phải là truy cập ngẫu nhiên hoàn toàn song nó vẫn nhanh hơn hệ thống truy cập tuần tự rất nhiều, vì vậy nó được sử dụng nhiều để lưu giữ các hệ điều hành, các phần mềm ứng dụng lớn cho máy vi tính. Nếu bề mặt của đĩa bị trầy xước thì một số bit thông tin

trên đĩa sẽ bị phá hủy, vì vậy cần cất giữ đĩa trong túi bảo vệ sạch sẽ, tránh để mốc ẩm. Đĩa từ phải để tránh xa nguồn từ trường mạnh vì nó có thể xóa sạch đĩa.

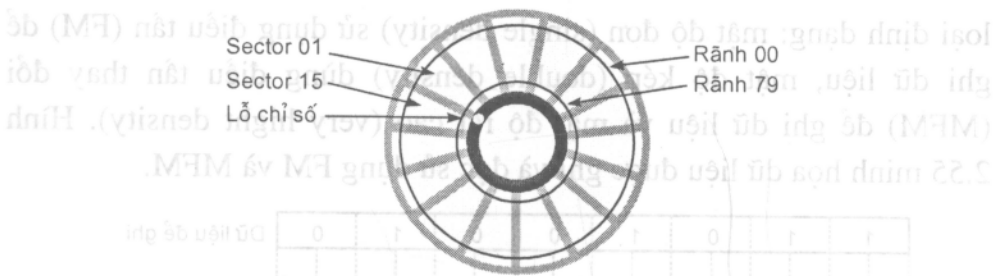
a) Đĩa mềm (floppy disk)

Thiết bị nhớ trên đĩa mềm rẻ tiền và tiện dụng cho người sử dụng. Đĩa mềm rất mềm và mỏng nên không gây nguy hiểm cho đầu từ như đĩa cứng. Đĩa mềm được chế tạo từ một tấm Mylar phủ ôxit từ và được đặt trong một vỏ bằng giấy, hoặc nhựa (hình 2.52).



Hình 2.52: Đĩa mềm dung lượng 1,2 MB, kích thước 5,25 inch

Trong những năm 1970 đĩa mềm thông dụng đóng trong vỏ giấy bìa 8 inch ra đời với các dung lượng: 197 kbyte, 256 kbyte, 512 kbyte, 612 kbyte, và 1024 kbyte. Đầu những năm 1980 đã xuất hiện loại đĩa mềm đóng trong vỏ giấy bìa 5,25 inch dung lượng 102 kbyte, 160 kbyte, 180 kbyte, 320 kbyte, 360 kbyte và 640 kbyte. Giữa những năm 1980 đã có loại đĩa 3,5 inch với dung lượng 720 kbyte và 1,44 MB. Hiện nay chỉ còn thông dụng loại đĩa mềm đóng trong vỏ nhựa 3,5 inch dung lượng 1,44 MB cho các máy vi tính. Đĩa mềm 5,25 inch có dung lượng 1,2 MB khi được định khuôn dạng cả hai mặt đĩa để sử dụng (formatting) thì dữ liệu được tổ chức thành 80 vòng rãnh (tracks), mỗi rãnh phân chia thành 15 sector, mỗi sector đặc trưng chứa 512 byte (hình 2.53): $80 \times 15 \times 512 \times 2 = 1.228.800 \text{ byte} = 1,2 \text{ MB}$.



Hình 2.53: Định khuôn dạng của đĩa mềm dung lượng 1,2 MB, kích thước 5,25 inch

Đĩa mềm 3,5 inch dung lượng 1,44 MB (hình 2.54) khi được định khuôn dạng cả hai mặt đĩa có tổ chức dữ liệu là: 80 rãnh, 18 sector, mỗi sector có 512 byte, nghĩa là: $80 \times 18 \times 512 \times 2 = 1.474.560$ byte.



Hình 2.54: Đĩa mềm dung lượng 1,44 MB, kích thước 3,5 inch

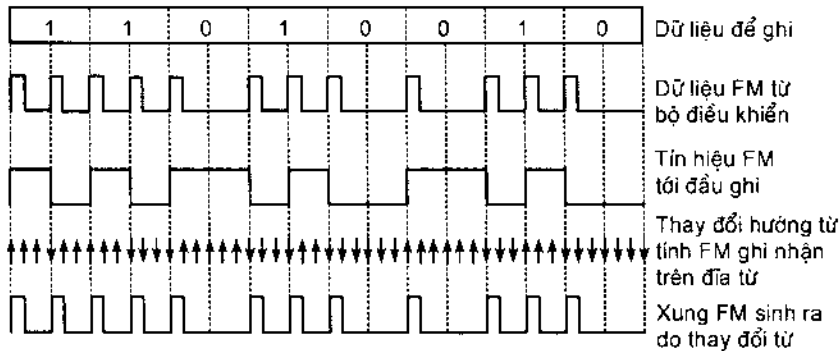
Như vậy số lượng rãnh, số lượng sector trên một rãnh quyết định dung lượng đĩa, như liệt kê trong bảng 2.2.

Bảng 2.2: Thông số các loại đĩa mềm

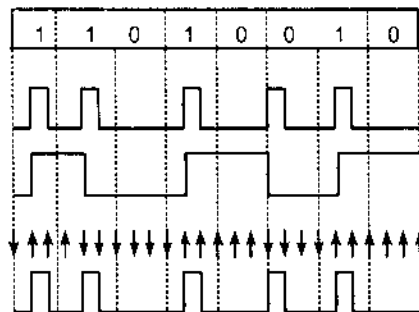
Loại đĩa	Dung lượng	Số rãnh/ mặt đĩa	Số sector/ rãnh	Số byte/ sector	Độ rộng của rãnh
5,25- inch	360 kbyte	40/mặt đĩa	9	512	0,330 mm
5,25- inch	1,2 MB	80/mặt đĩa	15	512	0,160 mm
3,5-inch	720 kbyte	80/mặt đĩa	9	512	0,115 mm
3,5-inch	1,44 MB	80/mặt đĩa	18	512	0,115 mm
3,5-inch	2,88 MB	80/mặt đĩa	36	512	0,115 mm

Tốc độ quay của đĩa mềm khoảng 300 hoặc 360 vòng/phút. Trước khi sử dụng đĩa mềm phải được định khuôn dạng (format). Có 3

loại định dạng: mật độ đơn (single density) sử dụng điều tần (FM) để ghi dữ liệu, mật độ kép (double density) dùng điều tần thay đổi (MFM) để ghi dữ liệu và mật độ rất cao (very high density). Hình 2.55 minh họa dữ liệu được ghi và đọc sử dụng FM và MFM.



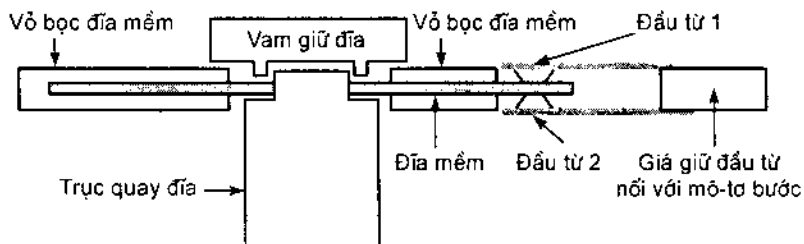
a) Ghi mật độ đơn bằng FM



b) MFM tạo ra dạng mật độ kép

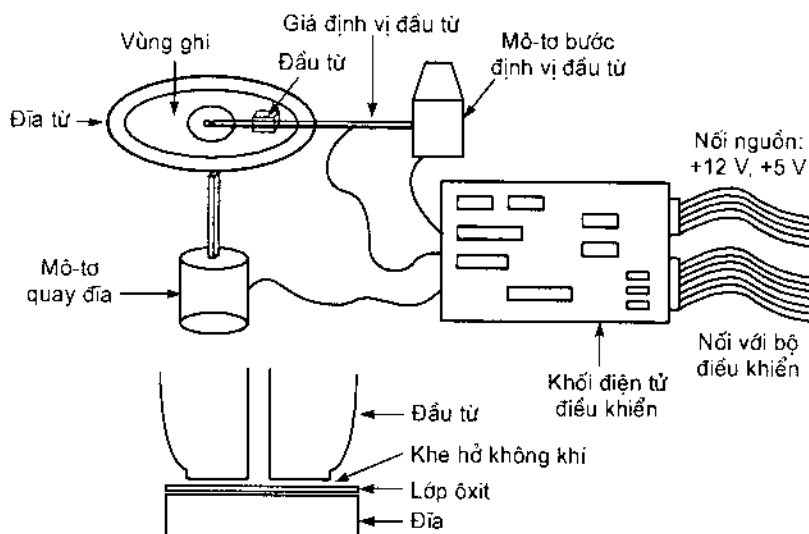
Hình 2.55: Ghi mật độ

Hình 2.56 minh họa đĩa mềm được đưa vào thiết bị đĩa như thế nào?



Hình 2.56: Đĩa mềm trong ổ đĩa

Phương pháp ghi MFM thường được dùng cho các loại đĩa dung lượng 360 kbyte, 720 kbyte, 1,2 MB và 1,44 MB. Hình 2.57 và hình 2.58 là sơ đồ khối của ổ đĩa mềm đặc trưng. Mô-tơ bước (step motor) dịch chuyển hai đầu từ (cho hai mặt đĩa) từ rãnh 00 đến rãnh trong cùng của đĩa. Mô-tơ bước được điều khiển tương tự như cơ chế loa tiếng.



Hình 2.57: Sơ đồ khối ổ đĩa mềm với phân điện tử và cơ khí

Ổ đĩa mềm (FDD) cần phải có hai tập hợp kết nối điện. Một tập hợp là các tín hiệu: điều khiển và tín hiệu mức TTL đi theo một cáp bện (ribbon cable) thường có 34 đường dây nối khối điện tử điều khiển của logic điều khiển ổ đĩa FDD (Drive control logic) và đơn vị điều khiển I/O với thiết bị đĩa (controller). Trong tập hợp này có: 4 tín hiệu chọn ổ đĩa (DS: 0, 1, 2, 3), 2 đường dây dữ liệu (dữ liệu tua lại, dữ liệu bản ghi), 6 đường dây lệnh điều khiển (mô-tơ hoạt động, bước, hướng, chọn mặt, bản ghi, mô-tơ hoạt động 2), 4 đường dây trạng thái (rãnh 00, sector ID, ổ đĩa sẵn sàng, bảo vệ chống ghi). Dữ liệu ghi vào và đọc từ FDD liên tục theo từng bit một, do đó chỉ có một đường dây dữ liệu đọc và một đường dây dữ liệu ghi. Khi lỗ chỉ số (index) đi qua

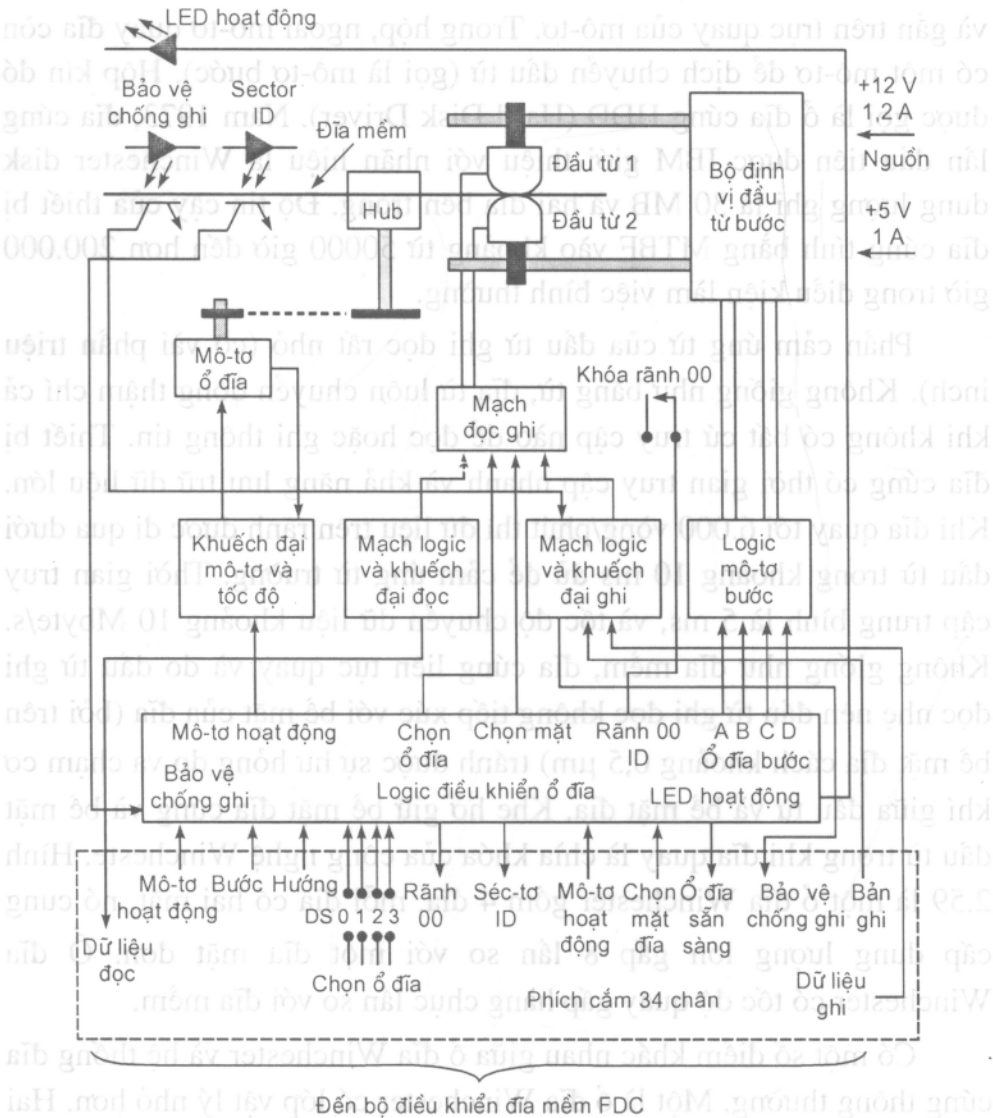
detector thì FDD sẵn sàng (ready). Đầu từ ở rãnh 00. Tập hợp còn lại là các đường dây nguồn cung cấp cho FDD, trong đó: +12V/1,2A cung cấp cho các mô-tơ và +5V/1A cho khối điện tử điều khiển. Mô-tơ bước dùng để chuyển động giá đầu từ tuyến tính để định vị đầu từ theo các rãnh. Một số loại FDD dùng cơ chế lỗi loa tiếng để định vị nhanh các đầu từ trên các rãnh đĩa. Trong những máy vi tính thế hệ vi xử lý i386 trở về trước (IBM-PC/AT), phần điện tử điều khiển I/O với bộ điều khiển đĩa (disk controller) là một bo mạch điện tử cắm trên khe mở rộng (slot) của máy vi tính. Bo mạch vừa để ghép nối thiết bị đĩa mềm vừa để ghép nối thiết bị đĩa cứng. Mỗi bo mạch cho phép ghép nối hai ổ đĩa mềm, hai ổ đĩa cứng. Có những bảng mạch điều khiển ghép nối vào/ra vạn năng bao gồm ghép nối FDD, HDD, máy in, truyền thông (COM). Thế hệ máy vi tính sử dụng i486 đến Pentium đã trang bị trên bo mạch chủ (main board) các khe phối ghép FDD, HDD, COM, máy in,...

Độ tin cậy của thiết bị đĩa mềm thường được xác định bằng thời gian làm việc trung bình giữa các sự cố MTBF (Mean Time Before Failure) và thường trong khoảng từ 30.000 giờ đến 50.000 giờ.

Thời gian truy cập (Access time) thiết bị đĩa mềm được tính bằng:

Thời gian truy cập = Thời gian tiềm tàng + Thời gian tìm kiếm

Thời gian truy cập là thời gian trung bình bắt đầu đọc đến khi xuất hiện byte dữ liệu đầu tiên. Thời gian tìm kiếm (seek time) là thời gian trung bình cần có để đầu từ chuyển đến lỗ tiếp xúc mặt đĩa, và thời gian tiềm tàng (latency time) là thời cần thiết để đĩa quay đủ tốc độ dưới đầu từ. Đặc trưng của thời gian tìm kiếm khoảng 15 ms, thời gian trễ khoảng 75 ms, thời gian truy cập trung bình là 90 ms. Như vậy, nếu như sector tìm thấy, tốc độ đọc dữ liệu trung bình trên 1 Mbit/s.



Hình 2.58: Sơ đồ khối một ổ đĩa mềm đặc trưng (FDD)

với các chức năng của khối điện tử điều khiển

b) Đĩa cứng (hard disk)

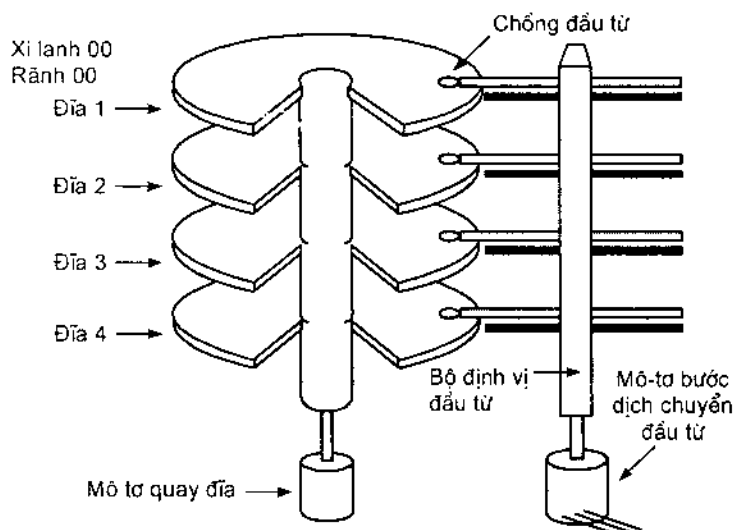
Đĩa cứng được chế tạo từ một đĩa bằng kim loại không từ tính kích cỡ khoảng 18÷20 inch, có tốc độ tối thiểu 33 vòng/phút khi không có rãnh. Bề mặt của đĩa được chế tạo kỹ lưỡng và được phủ một lớp mỏng ôxit sắt từ. Đĩa cứng để trong một hộp kín đã lọc trong sạch

và gắn trên trục quay của mô-tơ. Trong hộp, ngoài mô-tơ quay đĩa còn có một mô-tơ để dịch chuyển đầu từ (gọi là mô-tơ bước). Hộp kín đó được gọi là ổ đĩa cứng HDD (Hard Disk Driver). Năm 1973, đĩa cứng lần đầu tiên được IBM giới thiệu với nhãn hiệu là Winchester disk dung lượng ghi là 30 MB và hai đĩa bên trong. Độ tin cậy của thiết bị đĩa cứng tính bằng MTBF vào khoảng từ 50000 giờ đến hơn 200.000 giờ trong điều kiện làm việc bình thường.

Phản cảm ứng từ của đầu từ ghi đọc rất nhỏ (cỡ vài phần triệu inch). Không giống như băng từ, đĩa từ luôn chuyển động thậm chí cả khi không có bất cứ truy cập nào để đọc hoặc ghi thông tin. Thiết bị đĩa cứng có thời gian truy cập nhanh và khả năng lưu trữ dữ liệu lớn. Khi đĩa quay tới 6.000 vòng/phút thì dữ liệu trên rãnh được đi qua dưới đầu từ trong khoảng 10 ms đủ để cảm ứng từ trường. Thời gian truy cập trung bình là 5 ms, và tốc độ chuyển dữ liệu khoảng 10 Mbyte/s. Không giống như đĩa mềm, đĩa cứng liên tục quay và do đầu từ ghi đọc nhẹ nên đầu từ ghi đọc không tiếp xúc với bề mặt của đĩa (bởi trên bề mặt đĩa cách khoảng 0,5 μm) tránh được sự hư hỏng do va chạm cơ khí giữa đầu từ và bề mặt đĩa. Khe hở giữ bề mặt đĩa cứng và bề mặt đầu từ trong khi đĩa quay là chìa khóa của công nghệ Wincheste. Hình 2.59 là một ổ đĩa Winchester gồm 4 đĩa, mỗi đĩa có hai mặt, nó cung cấp dung lượng lớn gấp 8 lần so với một đĩa mặt đơn. Ổ đĩa Winchester có tốc độ quay gấp hàng chục lần so với đĩa mềm.

Có một số điểm khác nhau giữa ổ đĩa Winchester và hệ thống đĩa cứng thông thường. Một là ổ đĩa Winchester có lớp vật lý nhỏ hơn. Hai là quay với tốc độ thấp hơn so với ổ đĩa cứng thông thường. Ba là đĩa và đầu từ của ổ đĩa Wincheste gắn liền nhau nên khó di chuyển. Nếu muốn lưu giữ dữ liệu vào ổ đĩa Winchester thì phải sao chép dữ liệu vào hệ thống đĩa mềm hoặc băng từ. Hệ thống ổ đĩa Winchester yêu cầu một khối điều khiển đĩa (nằm trong ổ đĩa) đặc biệt. Khối điều khiển điều hành ổ đĩa mềm không thể phù hợp cho loại đĩa Wincheste. Giống như đĩa mềm, ổ đĩa Winchester cần mô-tơ quay đĩa (spindle

motor) và dịch chuyển đầu từ (step motor). Loại Winchester HDD dung lượng thấp (dùng cho các thế hệ IBM, PC/AT, XT) cần tới 2 bộ nối riêng để nối với các bảng mạch điều khiển thiết bị đĩa (Disk controller), trong đó: đầu nối 20 chân cho dữ liệu và 34 chân cho điều khiển. Độ dài cáp nối giữa HDD và bảng mạch điều khiển đĩa tối đa là 3 m. Winchester HDD sử dụng phương pháp ghi dữ liệu MFM, và kiểm tra dữ liệu theo CRC (Cyclic Redundancy Checking).



Hình 2.59: bên trong ổ đĩa cứng (HDD) loại Winchester (4 đĩa cứng)

Phương pháp CRC dựa trên lý thuyết toán cơ bản. Trước hết cần phải hiểu rằng một khối dữ liệu số có thể được viết như là đa thức (polynomial). Ví dụ một từ dữ liệu cần kiểm tra là 10011. Với từ dữ liệu này ta thực hiện tìm một đa thức:

$$1 \quad 0 \quad 0 \quad 1 \quad 1$$

Bước 1: $a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$

Bước 2: $1x^4 + 0x^3 + 0x^2 + 1x + 1$

Bước 3: $x^4 + x + 1$

Như vậy tương ứng với dữ liệu 10011 ta nhận được một đa thức:

$$D(x) = x^4 + x + 1.$$

Khi đa thức $D(x)$ được chia cho đa thức tạo ra $G(x)$ (generator polynomial), ta nhận được một đa thức dư $R(x)$ nào đó, gọi là CRCC:

$$D(x)/G(x) = Q(x) + R(x)/G(x), \text{ trong đó } R(x) = \text{CRCC}.$$

Đa thức $R(x)$ ngắn hơn đa thức gốc $D(x)$, và nó phải là một số duy nhất đối với từng khối dữ liệu. Đa thức dư này được gọi là CRCC (Cyclic Redundancy Check Character). Nếu thực hiện viết lại biểu thức trên như sau:

$$[D(x) - R(x)]/G(x) = Q(x) + 0$$

Thì sẽ không có một dư thừa nào cả. Đây chính là bản chất của phương pháp kiểm tra CRC. Nếu có dư thừa ở biểu thức này thì dữ liệu được kiểm tra bị lỗi. Nếu dư thừa bằng 0 thì dữ liệu được kiểm tra không bị lỗi.

Hiện nay, dung lượng của HDD đã lên đến vài chục GB và với kích thước đĩa nhỏ: 5,25; 3,25; 2,5; 1,8 và 1,3 inch và thậm chí nhỏ hơn nữa. Các HDD có giao tiếp 40 chân IDE và các giao tiếp SCSI. IBM vẫn là nhà sản xuất hàng đầu thế giới về HDD, hiện đã tung ra thị trường loại HDD model Ultrastar 36Z15 với các thông số kỹ thuật sau đây:

- Tốc độ quay đĩa: 15000 vòng/phút;
- Hai loại dung lượng: 18,4 GB và 36,7 GB;
- Thời gian đầu từ chuyển từ rãnh đến rãnh: 0,65 ms;
- Thời gian tiềm tàng trung bình (average latence time): 2,0 ms;
- Thời gian tìm kiếm: 3,4 ms loại 18,4 GB và 4,1 ms loại 36,7 GB;
- Thời gian truy cập: 5,4 ms = 2,0 ms + 3,4 ms của loại 18,4 GB và 6,1 ms của loại 36,7 GB;
- Tốc độ truyền dữ liệu tối đa: 453 ÷ 647 Mbit/s;
- Tốc độ truyền dữ liệu chịu đựng được: 36,6 ÷ 52,8 Mbit/s;
- Tốc độ giao tiếp: 160 MB/s (Ultra160 SCSI), 320 MB/s (Ultra320 SCSI) và 200/400 MB/s (FC-AL-2: giao tiếp cáp quang - Fibre channel attributed Loop);

- Mật độ ghi trên 1 inch² diện tích bề mặt đĩa: 10,7 tỷ bit;
- Số lượng đầu từ (GMR: Giant Magneto-Resistive): 8 cho dung lượng 18,4 GB và 12 cho dung lượng 36,7 GB;
- Dung lượng của sector: 512÷528 byte;
- Số lượng đĩa (tám đĩa làm từ vật liệu thủy tinh): 4 cho dung lượng 18,4 GB và 6 cho dung lượng 36,7 GB;
- Tốc độ lỗi: 10 trong 10E16 bit được đọc;
- Giao tiếp: Ultra160 SCSI và Ultra320 SCSI;
- Nguồn cung cấp: +5 V_{DC} (±5%), +12 V_{DC} (±5%);
- Môi trường: nhiệt độ làm việc 5^o đến 55^oC, nhiệt độ không làm việc: -40^o đến 70^oC, độ ẩm làm việc: 8% đến 90%, độ ẩm không làm việc: 5% đến 95%;
- Kích thước của HDD: chiều cao 25,7 mm; chiều rộng 101,6 mm; chiều sâu 146 mm;
- Trọng lượng của HDD: 0,75 kg.

Các loại ổ đĩa cứng của IBM đạt được sự tín nhiệm về chất lượng cao, tiếp sau là các loại HDD của Quantum, HDD của Seagate.

c) Thiết bị nhớ bằng băng từ (*magnetic tape storage*)

Nguyên tắc ghi đọc của thiết bị băng từ tương tự như các thiết bị đĩa từ, nhưng tốc độ truy cập chậm hơn rất nhiều và phải thực hiện truy cập tuần tự. Băng từ nhạy cảm với điều kiện môi trường như độ ẩm, nhiệt độ quá cao và quá thấp, bụi, trường điện từ, nên rất khó bảo quản và chất lượng ghi đọc thông tin không cao bằng đĩa từ. Tuy vậy, ưu điểm của thiết bị băng từ là người sử dụng có thể dùng không giới hạn số lượng băng từ để lưu giữ dữ liệu và có thể tháo rời bảo quản trong trạng thái không làm việc và băng từ có giá rẻ hơn nhiều so với đĩa từ. Thường sử dụng thiết bị băng từ để lưu giữ dự phòng (backup) dữ liệu của đĩa cứng. Sự lưu giữ dự phòng trên băng từ trong khai thác các hệ thống máy tính (hay tổng đài chuyển mạch) được thực hiện theo kiểu

lũy tiến (tăng dần). Băng từ dùng để lưu giữ dữ liệu số hóa DDS (Digital Data Storage) có các loại như sau:

- Băng từ 1/2 inch HIT (Haft-Inch Tape), băng từ cartridge 1/4 inch QIC (Quater-Inch Magnetic Cartridge) (4 mm), băng từ cartridge 1/3 inch (8 mm).

- Băng từ DDS dựa trên công nghệ của DAT (Digital Audio Tape), nhưng nó yêu cầu chất lượng cao hơn nhiều so với DAT và có sự khác nhau là: DAT được đọc theo chế độ dòng (streaming mode), nghĩa là băng từ phải được đọc tuần tự liên tục theo một hướng với một tốc độ cố định để có thể nghe đúng được âm thanh, và sự đọc ngược lại là không thể được. Khi ở một đoạn băng nào đó có vấn đề về âm thanh, chúng ta phải quay trở lại đầu đoạn băng đó để đọc lại. Sự quay đi quay lại của DAT là rất hạn chế. Trong khi đó, băng từ loại DDS lại cần phải quay đi quay lại trong quá trình ghi đọc dữ liệu với tần suất cao, điều này đòi hỏi nó phải có chất lượng rất cao so với DAT.

Loại HIT có lịch sử ra đời đã lâu và dùng cho các máy tính lớn, có độ tin cậy cao, nhưng dung lượng thấp và giá thành cao. Các thiết bị ghi đọc loại này to và cồng kềnh gây khó khăn cho việc cất giữ và vận chuyển. Loại này thường có mật độ 1.600 byte/inch (Bpi). Nó sử dụng đầu từ ghi đọc 9 đường (track). Như vậy cuộn băng 2.400 feet (1 feet = 12 inch) có thể lưu giữ dữ liệu với dung lượng hơn 46 triệu byte:

$$1.600 \text{ byte/inch} \times 12 \text{ inch} \times 2400 \text{ feet/cuộn} = 46.080.000 \text{ byte/cuộn.}$$

Một cuộn băng 9.600 feet có dung lượng:

$$1.600 \text{ byte/inch} \times 12 \text{ inch/feet} \times 9600 \text{ feet/cuộn} = 184.320.000 \text{ byte/cuộn.}$$

Cho rằng, cuộn băng này định vị ở vị trí đầu băng và dữ liệu cần đọc ở vị trí 1.200 feet (ở quãng giữa cuộn băng). Ta chuyển nhanh băng với tốc độ 180 inch/s đến vị trí 1185 feet và từ đây đọc dữ liệu ở 15 feet còn lại với tốc độ 60 inch/s. Như vậy, thời gian mất cho thao tác đọc băng này là:

$(1185 \text{ feet} \times 12 \text{ inch/feet})/180 \text{ inch/s} + (15 \text{ feet} \times 15 \text{ inch/feet})/60 \text{ inch/s} = 82 \text{ s.}$

Nếu dữ liệu cần đọc ở vị trí cuối băng thì thời gian đọc dữ liệu sẽ là:
 $(2385 \text{ feet} \times 12 \text{ inch/feet})/180 \text{ inch/s} + (15 \text{ feet} \times 12 \text{ inch/feet})/60 \text{ inch/s} = 162 \text{ s.}$

Ví dụ này cho thấy tốc độ truy cập dữ liệu trên băng từ theo kiểu tuần tự rất chậm. Cuộn băng càng lớn thì thời gian càng lâu.

Loại QIC chuyên dùng để lưu giữ dữ liệu cho các hệ thống máy tính nhỏ và cá nhân (mini, PC stations, PC servers). QIC nhỏ gọn, có dung lượng cao hơn 10 GB (dữ liệu chưa nén). Có các loại QIC cho DDS: DDS-1, DDS-2, DDS-3, DDS-4 và dung lượng của nó phụ thuộc vào độ dài của băng, ví dụ:

Loại: độ dài băng (m)	Dung lượng chứa nén xấp xỉ (GB)
DDS-2: 60 m	1,3 GB
DDS-2: 90 m	2,0 GB
DDS-2: 120 m	4,0 GB
DDS-3: 125 m	12 GB

QIC có độ tin cậy cao, môi trường làm việc: nhiệt độ $+5^{\circ}\text{C}$ đến 45°C , độ ẩm 20% đến 80%, bảo quản trong môi trường: nhiệt độ 5°C đến 32°C , độ ẩm 20% đến 60%. QIC có một số kích thước nhỏ, tốc độ truyền dữ liệu hơn 1 Mbit/s. Để tăng dung lượng lưu trữ, nhiều kỹ thuật nén dữ liệu được sử dụng ngay trong các thiết bị băng từ (firmware chip). Tỷ lệ nén đạt 2,5:1. Như vậy tốc độ truyền dữ liệu tăng lên 2,5 lần.

Loại cartridge 8 mm có dung lượng nguyên gốc chưa nén tới 60 GB (225 mét), 40 GB (150 mét), 20GB (75 mét) và dung lượng nén tới 150 GB (150 mét), 100 GB (150 mét), 50 GB (75 mét) với tỷ lệ nén là 2,5:1. Chúng cho phép đọc tốc độ tới 12 MB/s dữ liệu nguyên bản và 30 MB/s khi có nén dữ liệu theo tỷ lệ 2,5:1. Thời gian truy cập của cartridge nhỏ hơn 10 s. Môi trường khai thác của chúng là: nhiệt độ 5°C

đến 45°C, độ ẩm 20% đến 80%. Tổ chức dữ liệu trên băng từ được phân theo các khối (block), mỗi khối dữ liệu có độ dài: 128, 256, hoặc 512 byte, và có phân biệt đầu khối và kết thúc khối. Do đó, thiết bị băng từ có thể được điều khiển dừng ở bất kỳ khối nào. Các thiết bị ghi/đọc băng từ ngày nay có giao tiếp với hệ thống máy tính theo chuẩn SCSI 50 chân, và SCSI 68 chân Ultra2 LVD.

(2) Thiết bị nhớ bằng công nghệ quang

Năm 1983, một loại thiết bị sử dụng kỹ thuật quang bắt đầu xuất hiện do hai hãng Sony và Philips Electronic hợp tác sản xuất. Đó là đĩa quang CD-ROM (Compact Disk Read Only Memory), nó chỉ cho phép đọc dữ liệu (vì thế có chữ ROM). Người sử dụng không thể ghi lên đĩa dữ liệu giống như ghi lên đĩa từ, băng từ. Hệ thống lưu giữ quang có dung lượng tương đối lớn. Một đĩa 2 mặt kích thước 12 inch có thể lưu giữ tới 2 GB và đĩa CD kích thước 5 inch có thể lưu giữ được trên 680 MB, khả năng đó sẽ lớn hơn trong tương lai. Đĩa quang lưu giữ thông tin bằng những lỗ nhỏ trên bề mặt của đĩa. Ghi dữ liệu vào đĩa quang là một quá trình khó và cần phải có thiết bị ghi chuyên dụng. Có nhiều loại thiết bị ghi đọc đĩa quang với các tốc độ khác nhau. Đĩa quang có 3 loại hay được sử dụng đó là: OROM (Optical ROM: Bộ nhớ chỉ đọc quang), WORM (Write Once-Read Many: Ghi một lần - đọc nhiều lần), WMRA (Write Many-Read Always: Ghi, xóa nhiều lần). Kích thước đĩa quang thông dụng hiện nay là 5,25 inch. Các loại CD-ROM, CD-RW có các giao tiếp 50 chân IDE và các loại SCSI. Độ tin cậy của thiết bị đĩa quang thấp hơn so với đĩa cứng và giá thành còn cao.

a) CD-ROM loại OROM

OROM thường được gọi là CD-ROM, có giá thành thấp nhất được sử dụng cho thị trường âm thanh, video. Loại có dung lượng 680 MB, tốc độ truyền dữ liệu 150 kbyte/s, thời gian truy cập nhỏ hơn 300 ms và độ tin cậy theo MTBF khoảng 30.000 giờ trong môi trường nhiệt độ làm việc: 5°C đến 55°C.

b) CD-ROM loại WORM

WORM tiếp sau CD-ROM, công nghệ ghi một lần - đọc nhiều lần ra đời. Không giống như CD-ROM, người sử dụng có thể ghi thông tin lên đĩa một lần nhưng đọc nhiều lần. Công nghệ này khi ra đời đã có ngay nhiều ứng dụng, đặc biệt là trong các thiết bị nghe nhìn của điện tử dân dụng và các máy vi tính hiện nay. Chúng có các tốc độ truyền dữ liệu chưa nén: 2X, 4X, 8X, 14X, 24X, 32X, 40X, 50X. Tốc độ chuẩn (1X) = 153,6 kbyte/s. Thời gian tìm kiếm nhỏ hơn 300 ms.

c) CD-RW loại WMRA

WMRA, năm 1988 hãng Sony đã lần đầu tiên giới thiệu WMRA, nó sử dụng tốt cho những ứng dụng lưu giữ dữ liệu. Người sử dụng có thể ghi dữ liệu lên, xóa đi rồi lại ghi lên được. Sản phẩm thông dụng có dung lượng 650 MB, tốc độ truyền dữ liệu tương ứng với loại CD-ROM WORM. Với loại tốc độ 680 kbyte (4X), thời gian tìm kiếm trung bình là 70 ms và độ tin cậy theo MTBF là 40.000 giờ.

2.4. VI XỬ LÝ INTEL

2.4.1. Đặc điểm chính của các loại vi xử lý Intel

(1) Các đặc điểm của họ i808X

Đánh dấu sự khởi đầu phát triển của máy vi tính phải kể tới họ vi xử lý i808X. Đặc điểm của thế hệ này là xử lý 8-bit, tần số làm việc thấp (bảng 2.3). Chip i8086A với 16 bit bus dữ liệu trong/bus dữ liệu ngoài, và chip i8088A với 16 bit bus dữ liệu trong/8 bit bus dữ liệu ngoài đã được dùng làm CPU trong các loại IBM PC/XT. Khi đó, các thiết bị ngoại vi và bộ nhớ bán dẫn mới chỉ thỏa mãn trao đổi dữ liệu 8-bit, do đó, hiệu quả sử dụng hơn (về chi phí) là IBM PC/XT88 với CPU là chip i8088A. Họ i808X có mức tích hợp không lớn, nên chúng sử dụng phương pháp đóng vỏ gốm 40 chân DIP với sự kết hợp chung (multiplexed) các đường của bus địa chỉ với 8 đường dây địa chỉ thấp của bus địa chỉ. Kiến trúc và tập lệnh của i8080 là cơ sở cho những

phát triển các loại vi xử lý sau này của Intel và một số nhà sản xuất khác. Các chương trình viết theo tập lệnh i8080 có thể chạy trên các hệ thống vi xử lý i808X sau này. Kiến trúc và tập lệnh của i8085 cũng là một thí dụ để nghiên cứu và phổ biến kiến thức về kỹ thuật vi xử lý rộng rãi trong các cơ sở đào tạo. Sự phát triển của thế hệ i808X xác định trong khoảng 10 năm, và tiếp theo chúng là một thế hệ mới, thế hệ i80X86, từ 1980, bắt đầu bằng chip i80286.

Bảng 2.3: Các đặc điểm của họ Intel 808X

Đặc điểm	i8080/i8080A	i8085A/i8085AH	i8086A/i80C86A	i8088A/i80C88A
Bus dữ liệu	8-bit trong/ngoài	8-bit trong/ngoài	16-bit trong/ngoài	16-bit trong/ 8-bit ngoài
Bus địa chỉ	16-bit (multiplexed)	16-bit (multiplexed)	20-bit (multiplexed)	20-bit (multiplexed)
Nhịp đồng hồ	2-3,125 MHz	3-6 MHz	4-12 MHz	5-12 MHz
Bộ nhớ icache trong	-	-	-	-
Bộ nhớ Dcache trong	-	-	-	-
Đồng xử lý toán học (Coprocessor)	-	-	8087 ngoài	8087 ngoài
Đơn vị quản lý bộ nhớ (MMU)	-	-	MMU ngoài	MMU ngoài
Đánh địa chỉ bộ nhớ	64 kbyte	64 kbyte	1 MB	1 MB
Tập lệnh	8080	8080 mở rộng	8080 mở rộng	8080 mở rộng
Kiến trúc siêu hướng/đường ống	-	-	-	-
Công nghệ	PMOS, NMOS	NMOS, HMOS	NMOS, CMOS, 2 μ	CMOS
Mật độ transistor	4000 - 4500	6200,	29E3	
Đóng vỏ	Gồm 40 chân DIP	Gồm 40 chân DIP	Gồm 40 chân DIP	Gồm 40 chân DIP
Nguồn cung cấp	5,0 V	5,0 V	5,0 V	5,0 V
Máy tính, OS	PC, CP/M	PC	IBM PC/XT	IBM PC/XT
Năm sản xuất	1973	1976	1978	1979

(2) Các đặc điểm của họ i80X86

Các chip i80286 được coi là loạt vi xử lý khởi đầu cho các loạt vi xử lý tiên tiến của Intel, gọi chung là X86 (kể cả Pentium), và nó là CPU của chủng loại IBM PC/AT (Advanced Technology), mặc dù i80286 chưa xếp vào X86. Đặc điểm của họ i80X86 được liệt kê trong bảng 2.4. Mức độ tích hợp của các loại vi xử lý này khá cao, đặc biệt là từ i80386, do đó, phương pháp đóng vỏ không còn theo kiểu truyền thống DIP nữa (ngoại trừ chip i80286 vẫn còn giữ kiểu đóng vỏ gôm 68 chân DIP), mà theo mảng chân nối PGA (Pin Grid Array) và vuông phẳng QFP (Quad Flat Pack). Có thể có các loại đóng vỏ QFP như kiểu vỏ nhựa PQFP (Plastic QFP), kiểu kim loại MQFP (Metal QFP), và kiểu thu nhỏ SQFP (Shrink QFP). Công nghệ đặc trưng của chúng là CMOS với nguồn tiêu thụ thấp (đặc biệt là các phiên bản SL), giải nhiệt độ rộng, và tần số làm việc cao hơn so với công nghệ NMOS của họ i808X. Do mức độ tích hợp cao nên độ rộng xử lý tăng lên đến 32-bit bus dữ liệu (trừ i80286 xử lý 16-bit), và 32-bit bus địa chỉ cho phép đánh địa chỉ tới 4 GB nhớ. Bus dữ liệu và bus địa chỉ đã được tách riêng làm đơn giản hơn điều khiển bus. Khác với họ i808X, các chip i80486 đã áp dụng kỹ thuật làm tăng tốc độ truy cập bộ nhớ là kỹ thuật bộ nhớ trung gian tốc độ nhanh (Cache memory). Tổ chức bộ nhớ cache gồm 2 cấp: cấp L1-cache bên trong chip, cấp L2-cache bên ngoài chip. Sở dĩ xuất hiện kỹ thuật bộ nhớ cache là vì công nghệ bộ nhớ bán dẫn chưa đảm bảo thỏa mãn yêu cầu tốc độ truy cập nhanh kịp với tốc độ của CPU, trong khi đó phải có dung lượng lớn và rẻ. Bộ nhớ cache (theo công nghệ SRAM) là một đệm trung gian giữa CPU và bộ nhớ chính và không kết nối thông qua bus hệ thống. Nó có dung lượng thấp, nhưng tối ưu về tổ chức chứa các loại dữ liệu và lệnh có tần suất sử dụng nhiều nhất, sao cho tỷ lệ truy cập tới nó (hệ số trúng đích: H-hit) là cao nhất. Như thế sẽ giảm nhiều thời gian trao đổi dữ liệu giữa CPU và bộ nhớ chính (RAM, ROM). Cache L1 được tích hợp bên trong các chip từ i80386 trở đi.

Đơn vị xử lý dấu phẩy động FPU, đơn vị quản lý bộ nhớ MMU, và kiến trúc siêu hướng với đường ống lệnh đặc trưng 5 giai đoạn đã được tích hợp bên trong chip i80486DX. Những đặc tính này đã nâng cao tốc độ xử lý lên nhiều.

Bảng 2.4: Các đặc điểm của họ Intel 80x86

Đặc điểm	i80286	i80386DX	i80386SX	i80486DX	i80486SX
Bus dữ liệu	16-bit trong/ ngoài	32-bit trong/ ngoài	32-bit trong/ 16-bit ngoài	32-bit trong/ ngoài	32-bit trong/ ngoài
Bus địa chỉ	24-bit	32-bit	24-bit	32-bit	32-bit
Xung nhịp	6-20 MHz	12-33 MHz	16-33 MHz	20-100 MHz	16-100 MHz
Bộ nhớ Lcache trong	-	16 byte	16 byte	32 byte	32 byte
Bộ nhớ Dcache trong	-	256 byte	256 byte	8 kbyte	8 kbyte
Đồng xử lý toán học (Coprocessor)	80287 ngoài	80387 ngoài	80387 ngoài	FPU trong	80387 ngoài
Đơn vị quản lý bộ nhớ (MMU)	MMU ngoài	MMU ngoài	MMU ngoài	MMU trong	MMU ngoài
Đánh địa chỉ bộ nhớ	16 MB	4 GB	16 MB	4 GB	4 GB
Kiến trúc siêu hướng/ đường ống	-	-	-	Đường ống 5 giai đoạn	Đường ống 5 giai đoạn
Hỗ trợ đa xử lý	-	-	-	-	-
Công nghệ MMX	-	-	-	-	-
Công nghệ	HMOS	CMOS 0,8 μ	CMOS 0,8 μ	biCMOS/ CHMOS 0,6 μ	CMOS
Mật độ transistor	134E3	275E3		1,6E6	0,9E6
Đóng vỏ	Gồm 68 chân DIP	PGA 132 chân	QFP 100 chân	PGA 168 chân SQFP 208 chân	PGA 168 chân FQFP 208 chân
Nguồn cung cấp	5,0 V	3,3 V - 5,0 V	3,3 V - 5,0 V	3,3 V - 5,0 V	3,3 V - 5,0 V
Máy tính, OS	IBM PC/AT, MS-DOS	IBM PC 386DX, MS- DOS, Wins.	IBM PC486SX MS-DOS, Wins	IBM PC486DX MS-DOS, Wins	IBM PC486SX MS- DOS, Wins
Năm sản xuất	1982	1985	1988	1993	1991

Loạt i80386SX, i80486SX (SX: Single-word eXternal) có các phiên bản i80386SL, và i80486SL sử dụng nguồn cung cấp thấp (3,3 V). Phiên bản i80486DX4 P54LM dùng cho các máy tính xách tay (notebooks) chỉ dùng nguồn 2,5 ÷ 2,9 V. Đặc điểm tiêu thụ nguồn thấp này được thực hiện nhờ đưa vào chế độ quản lý hệ thống và nguồn SMM (System Management Mode). Chế độ này được khai thác ở 2 mức: mức 1 - nguồn được cung cấp cho toàn bộ hệ thống (kể cả các ngoại vi), và mức 2 - mức vi xử lý, còn gọi là chế độ "ngủ" (Sleep mode), hay mức tiết kiệm nguồn, khi đó, không có sử dụng các lệnh máy, nhịp đồng hồ được tự động dừng (clock auto stop), và vi xử lý duy trì trạng thái sử dụng tiết kiệm nguồn cung cấp. Các mạch bên trong chip vi xử lý duy trì trạng thái máy tính không được sử dụng. Sự tiêu thụ nguồn thấp tạo cơ hội đóng vỏ chip nhỏ hơn (QFP), tỏa nhiệt ít hơn (chip ít nóng hơn).

Kỹ thuật nguồn thấp SL bao gồm các đặc tính như: Tự động dừng nhịp đồng hồ (clock auto stop), tự động dừng nguồn (auto halt power down), tự động dừng nguồn rồi (auto idle power down), khởi động lại vào/ra (I/O Restart), và SMM. Loại i80486SX cũng là thế hệ đầu tiên được áp dụng để cho ra các phiên bản chip nhúng (embedded processor) sử dụng nguồn siêu thấp (Ultra Low-Power Intel 486SX embedded processor). Loại này có đóng vỏ 176 chân TQFP (176-lead) và khai thác với nguồn $V_{cc} = 2,4$ V. Nó cho phép đạt được sự giảm nguồn tiêu thụ tới 50% so với i80486SX nhờ sử dụng SMM. Nó có mức tiêu tán nhiệt và nhiễu hệ thống thấp. Ultra Low-Power embedded Intel486SX sử dụng các kỹ thuật RISC để đảm bảo thực hiện các lệnh trong một chu kỳ của nhịp đồng hồ.

Khoảng 10 năm phát triển của họ vi xử lý công nghệ cao X86 này được chuyển tiếp bằng thế hệ Pentium. Họ vi xử lý Pentium là giải pháp lý tưởng cho các ứng dụng hiệu suất cao, đặc biệt là các ứng dụng đa phương tiện (audio, video, data) tốc độ cao, và trong nhiều lĩnh khác. Bus dữ liệu được mở rộng tới 64-bit bus trong/bus ngoài cho

phép CPU Pentium kết nối với nhiều loại thiết ngoại vi. Gia đình Pentium gồm: Pentium, Pentium Pro, Pentium II, Celeron, Xeon, Pentium III, Pentium IV, và Pentium M (công nghệ Intel Centrino Mobile).

(3) Các đặc điểm của họ Pentium

Họ vi xử lý Pentium là giải pháp lý tưởng cho các ứng dụng hiệu suất cao, đặc biệt là các ứng dụng đa phương tiện (âm thanh, video, dữ liệu) tốc độ cao, và không chỉ trong lĩnh vực PC mà còn ở trong nhiều loại hệ thống điện tử khác. Độ rộng xử lý dữ liệu với ngoại vi được mở rộng tới bus 32-bit trong/64-bit ngoài cho phép CPU Pentium kết nối với nhiều loại thiết bị ngoại vi.

Các chip Pentium có kiến trúc siêu hướng (kiến trúc P5) với đường ống lệnh (hay là luồng lệnh) của FPU 8 giai đoạn, và hỗ trợ cho kết nối đa xử lý (2-4 chip Intel Pentium). Nguồn cung cấp cho Pentium thấp hơn so với họ i80X86. Tập lệnh của Pentium tương thích với tập lệnh của i80486, do đó các chương trình viết cho i80486 có thể chạy được trên các họ PC Pentium. Pentium có đóng vỏ 273 chân PGA 5V, hoặc 296 chân 3,3 V Socket 4,5,7 mật độ 3,1 triệu transistor. Họ Pentium với các loại làm việc ở các tần số nhịp 60/66/75/100/120/133/150/166/200 MHz. Intel (1975) đã đưa công nghệ MMX (Matrix Math eXtensions, Multi-Media eXtensions), vốn dĩ được AMD và Cyrix phát triển, bổ sung cho các chip Pentium II. Ngoài ra, các chip Celeron, Xeon và Mobile Pentium cũng được bổ sung MMX và được sử dụng rộng rãi vì tốc độ cao, nguồn tiêu thụ thấp, giá rẻ. Khối MMX có 57 SIMD lệnh (Single-Instruction, Multiple-Data) dành riêng cho xử lý đồ họa, âm thanh, video, hình ảnh, tiếng nói, trong các lĩnh vực truyền thông, tài chính, khoa học kỹ thuật. Pentium với công nghệ MMX có thể đảm bảo tăng từ 10 đến 20% hiệu suất so với Pentium bình thường (không có MMX), và tăng gấp đôi dung lượng cache. Năm 1999, công nghệ KNI (Katmai New Instructions), còn gọi là MMX2, đã bổ sung thêm Pentium 70 lệnh cho MMX cơ bản để xử lý đồ họa 3D (bảng 2.5).

Bảng 2.5: Các đặc điểm của họ Intel Pentium

Đặc điểm	Pentium 586 (P5)	Pentium Pro P6	Pentium II	Pentium III	Pentium IV
Bus dữ liệu	64-bit trong/ 64-bit ngoài	32-bit trong/ 64-bit ngoài	32-bit trong/ 64-bit ngoài	32-bit trong/ 64-bit ngoài	32-bit trong/ 32-bit ngoài
Bus địa chỉ	32-bit	32-bit	32-bit	32-bit	32-bit
Xung nhịp	60 - 133 MHz	60 - 200 MHz	66 - 450 MHz	350 MHz - 1 GHz	1,3 - 1,7 GHz
Bộ nhớ Lcache trong	8 kbyte	8 kbyte	16 kbyte	16 kbyte	32 kbyte
Bộ nhớ Dcache trong	8 kbyte	8 kbyte	16 kbyte	16 kbyte	8 kbyte
Đồng xử lý toán học (Coprocessor)	FPU trong	FPU trong	FPU trong	FPU trong	FPU trong
Đơn vị quản lý bộ nhớ (MMU)	MMU trong	MMU trong	MMU trong	MMU trong	MMU trong
Đánh địa chỉ bộ nhớ	4 GB	4 GB	4 GB	4 GB	4 GB
Kiến trúc siêu hướng/đường ống	Đường ống FPU 8 giai đoạn	Đường ống FPU 8 giai đoạn	Đường ống FPU 8 giai đoạn	Đường ống FPU 8 giai đoạn	Đường ống 5 giai đoạn
Hỗ trợ đa xử lý	2 Pentium	2-4 Pentium Pro	2 Pentium II	2 Pentium III	4 Pentium IV
Tốc độ bus ngoài	66 MHz	66 MHz	66 MHz	100 MHz - 133 MHz	100 MHz - 133 MHz
Công nghệ MMX	-	-	Có MMX 57 lệnh của SIMD	Có MMX mở rộng của SIMD cho caching và nâng hiệu suất TCP/IP	Có MMX 144 lệnh mới mở rộng của SIMD
Công nghệ	BiCMOS 0,8 μ	biCMOS 0,6 μ	CMOS 0,35 μ	CMOS 0,8 μ	CMOS, CHMOS
Mật độ transistor	3,1E6	5,5E6	7,5E6		1,2E6
Đóng vỏ	273 chân PGA	387 chân PGA	242 chân BGA	FC-PGA, PPGA	432 chân OLGA
Nguồn cung cấp	5 V	3,1 V - 3,5 V	2,8 V - 3,3 V	1,35 V - 2 V	1,7 V
Chipset được cung cấp	82430HX/ 82430TX PCIsset	82430HX/ 82430TX PCIsset	440BX AGPset	440BX, 810, 815, 840 Chipset	850 Chipset
Máy tính, OS	PC Pentium P5, MS-DOS, Wins.	PC Pentium P6, MS-DOS, Wins.	PC Pentium II, MS-DOS, Wins.	PC Pentium III, Wins.	PC Pentium IV, Wins.
Năm sản xuất	1993	1995	1997	2000	Cuối 2000

- *Những đặc điểm chính của vi xử lý Pentium với công nghệ MMX*

- Tốc độ: 166, 200, 233, 266 MHz
- Tương thích với mạch hỗ trợ: 430TX PCIset
- Đóng vỏ HL-PBGA, PPGA
- Công nghệ, mật độ transistor: CMOS, 4,5E6 transistor
- Bộ nhớ cache L1: 16 kbyte Icache, 16 kbyte Dcache
- Nguồn cung cấp: 2,8 V lõi (phiên bản nguồn thấp: 1,8-2 V), 3,3 V I/O
- Tốc độ bus ngoài (external bus speed): 66 MHz
- Có phiên bản cho Mobile Module trong Mini Cartridge
- Mở rộng nhiệt độ làm việc: đến +115°C.

- *Những đặc điểm chính của vi xử lý Celeron Pentium (từ Pentium II)*

- Tốc độ: 300, 366, 400, 433, 566, 733, 850 MHz
- Bộ nhớ cache L1: 16 kbyte Icache, 16 kbyte Dcache
- Có MMX hỗ trợ các ứng dụng đồ họa 2D, 3D
- Đóng vỏ: 370 chân FC-PGA (loại 566, 733 và 850 MHz), 370 chân PPGA (loại 300, 366 và 433 MHz), BGA2 (loại 400 MHz)
- Công nghệ và mật độ transistor: 0,25 micron CMOS, 7,5E6 transistor
- Tương thích với các mạch hỗ trợ: 810, 815, 815E, 815E2, 440MX Chipset và 440BX AGPset
- Tốc độ bus ngoài: 66 MHz
- Không hỗ trợ đa xử lý
- Có phiên bản cho Mobile Module trong Mini Cartridge
- Nguồn cung cấp: 1,75 - 2,0 V lõi (phiên bản nguồn thấp: 1,35 V), 3,3 V I/O.

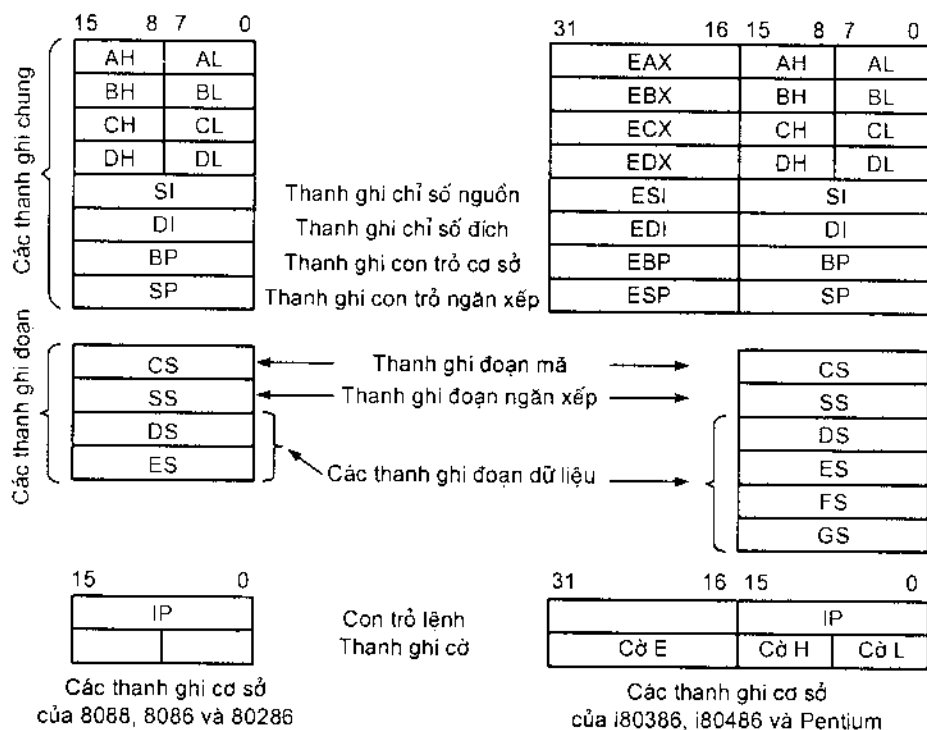
- *Những đặc điểm chính của vi xử lý Xeon Pentium (từ Pentium II)*
 - Tốc độ: 100 - 500 MHz
 - Bộ nhớ cache L1: 16 kbyte Icache, 16 kbyte Dcache
 - Có MMX hỗ trợ các ứng dụng đồ họa 2D, 3D
 - Tương thích các mạch hỗ trợ: 450NX Chipset
 - Hỗ trợ đa xử lý: 4 CPU với 450NX Chipset
 - Đóng vỏ: 330 chân SEC module (Single Edge Contact)
 - Công nghệ, mật độ transistor: 0,25 micron CMOS, 7,5E6 transistor
 - Nguồn cung cấp: 2,0 V lõi, 3,3 V I/O.

2.4.2. Mô hình lập trình chung của họ Intel

Thực tế, cấu trúc bên trong các chip vi xử lý rất phức tạp, và không thể mô tả chi tiết được tất cả các mối liên kết thông tin của các khối chức năng, các mạch logic với nhau được. Sự can thiệp của người lập trình hệ thống chỉ đến được các khối chức năng lớn như các tệp thanh ghi chung, thanh ghi địa chỉ, các thanh ghi đoạn,... Vì vậy, để có thể hiểu được vận hành của các chip vi xử lý, chúng ta có thể lập mô hình lập trình cho chúng.

Mô hình lập trình bao gồm tất cả các thành phần mà người lập trình có thể can thiệp để xử lý dữ liệu. Những khối chức năng không có trong mô hình lập trình thì có nghĩa là chúng không thể can thiệp vào được thông qua lập trình. Để có thể so sánh, chúng ta nêu ra đây mô hình lập trình chung cho các loại vi xử lý của Intel (hình 2.60).

Trong mô hình lập trình chung này có phân biệt: mô hình lập trình cho các họ i808X và i80286, và mô hình lập trình cho các họ i80X86 và Pentium. Các thanh ghi của họ i808X và i80286 là 16 bit, của i80386, i80486 và Pentium được mở rộng đến 32 bit (với ký hiệu mở rộng là E: EAX, EBX, EDI, ESI,...). Các chương trình viết cho họ i808X và i80286 có thể chạy được trên các hệ thống i80386, i80486 và Pentium.



Hình 2.60: Mô hình lập trình chung của Intel

Mô hình lập trình chung của Intel gồm 3 nhóm thanh ghi:

- **Nhóm thanh ghi chung (8 thanh ghi):** Các thao tác tính toán số học và logic, vận chuyển dữ liệu đều thực hiện trên nhóm thanh ghi này. Dữ liệu xử lý có thể thực hiện theo từng bit, nhóm bit, theo byte, theo từ 16 bit hay 32 bit. Các thanh ghi đều giống nhau ở tên gọi trong toàn bộ họ vi xử lý Intel. Phần mở rộng 32 bit có thêm E.

- i808X và i80286 (8 và 16 bit): AX (AH, AL), BX (BH, BL) và EBX, CX (CH, CL), DX (DH, DL), SI (chỉ số nguồn), DI (chỉ số đích), SP (con trỏ ngăn xếp), BP (con trỏ cơ sở). Chúng có dung lượng 16 bit.

- i80X86 và Pentium (16 và 32 bit): AX+EAX, BX+EBX, CX+ECX, DX+EDX, SI+ESI, DI+EDI, BP+EBP, SP+ESP.

- **Nhóm thanh ghi segment (thanh ghi đoạn):** Nhóm thanh ghi này đều có độ rộng 16 bit và giống nhau cho tất các họ Intel. Chúng

dùng để chứa địa chỉ cơ sở của đoạn nhớ. Bởi vì Intel có cơ chế tạo địa chỉ và định đoạn nhớ (segment) có dung lượng đặc trưng tối thiểu là 64 kbyte. Do đó, địa chỉ đoạn vùng nhớ 64 kbyte cần tới 16 bit của thanh ghi đoạn.

- i808X và i80286 có các thanh ghi: đoạn mã: CS, đoạn ngăn xếp: SS, đoạn dữ liệu: DS, đoạn bổ sung: ES.
- i80386, i80486 và Pentium: CS, SS, DS, ES và thêm FS, GS.
- Nhóm thanh ghi thứ ba: gồm con trỏ lệnh IP (Instruction Pointer) và thanh ghi cờ (Flags).

Con trỏ lệnh IP chứa địa chỉ của lệnh máy tiếp theo cần phải thực hiện trong một trình tự chương trình. Khi bộ vi xử lý đang thực hiện một lệnh máy nào đó, thì IP chứa địa chỉ của lệnh máy tiếp theo.

Thanh ghi cờ (Flags register) chứa trạng thái hiện thực của bộ vi xử lý. Kết quả thực hiện xong một lệnh máy có thể làm thay đổi các bit trong thanh ghi cờ. Dựa vào giá trị các bit của thanh ghi cờ, người lập trình có thể điều khiển rẽ nhánh trình tự thực hiện chương trình bằng các lệnh nhảy (rẽ nhánh) theo điều kiện giá trị các bit cờ.

- i808X và i80286: IP và thanh ghi cờ có 16 bit. 16 bit của thanh ghi cờ được phân bố ý nghĩa như sau (hình 2.61):

15	14	12	13	11	110	9	8	7	6	5	4	3	2	1	0
0	NT	IOPL		O	D	I	T	S	Z	X	A	X	P	X	C

Hình 2.61: Thanh ghi cờ của i808X và i80286

- i80386, i80486 và Pentium: IP có thể là 16 hay 32 bit. Thanh ghi cờ có 32 bit với ý nghĩa từng bit như hình 2.62.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	I	V	V	A	V	R	0	N	IOP	O	D	I	T	S	Z	X	A	X	P	X	C
											D	I	P	F	C	M	F														

Hình 2.62: Thanh ghi cờ của i80X86 và Pentium

Thanh ghi cờ của i80X86 và Pentium với 16 bit thấp chính là nội dung thanh ghi cờ của i808X và i80286. Những bit cờ còn lại là mở rộng riêng của chúng. Các bit cờ có ý nghĩa như sau:

- Bit 0, C: Carry Flag (cờ nhớ qua): $C = 1$ nếu có nhớ sau khi thực hiện phép toán đến bit cao MSB, nếu không có nhớ xuất hiện ở bit MSB thì $C = 0$. MSB là bit 7, 15, 31 trong các phép tính độ rộng từ tương ứng. Cờ C cũng có thể được thiết lập nhờ lệnh STC và được xóa nhờ lệnh CLC.

- Bit 2, P: Parity Flag (cờ chẵn lẻ): $P = 1$, nếu tổng số bit = 1 trong từ (8-bit, 16-bit, 32-bit tương ứng) là lẻ. Ngược lại, tổng số bit = 1 là chẵn chẵn thì $P = 0$.

- Bit 4, A: Auxiliary Flag (cờ nhớ phụ): $A = 1$ nếu có nhớ ở bit 3 trong các phép toán với mã BCD.

- Bit 6, Z: Zero Flag (cờ zero): $Z = 1$ khi kết quả phép toán là 0, ngược lại $Z = 0$.

- Bit 7, S: Sign Flag (cờ dấu): $S = 1$ khi bit MSB = 1, số âm, $S = 0$, khi MSB = 0, số dương.

- Bit 8, T: Trap Enable Flag (cờ cho phép bẫy): $T = 1$, bộ vi xử lý thực hiện các lệnh từng bước để gỡ rối (debugging). Trong chế độ này, sau mỗi lần một lệnh máy thực hiện xong thì có sự dừng, và ta có thể kiểm tra được lệnh đã thực hiện như thế nào thông qua nội dung các thanh ghi của mô hình lập trình của bộ vi xử lý.

- Bit 9, I: Interrupt Enable Flag (cờ cho phép ngắt): $I = 1$ cho phép các ngắt bên ngoài được thực hiện (tín hiệu ngắt xảy ra ở chân INTR của chip vi xử lý). IF thiết lập bằng lệnh STI và được xóa (để cấm ngắt) bằng lệnh CLI (đối với ngắt không che được NMI thì cờ I không có tác dụng).

- Bit 10, D: Direction Flag (cờ hướng): D xác định cho SI và DI tăng hoặc giảm nội dung của chúng trong quá trình thực hiện các lệnh chuỗi (string instructions). $D = 0$, thực hiện tăng nội dung của SI và

DI. D = 1 thực hiện giảm nội dung của SI và DI. D được thiết lập bằng lệnh STD và được xóa bằng lệnh CLD.

- Bit 11, O: Overflow Flag (cờ tràn): O = 1 nếu kết quả phép tính có tràn số.

- Bit 13, 12: IOPL (từ i80286), Input/Output Privilege Level Flag (cờ các mức thẩm quyền vào/ra): IOPL có giá trị bằng: 0, 1, 2, 3 (hay số mã nhị phân: 00, 01, 10, 11) chỉ ra giá trị mức thẩm quyền được phép để truy cập tới vùng địa chỉ vào ra.

- Bit 14, NT: Nested Task Flag (từ i80286) (cờ nhiệm vụ thường trú). NT dành cho chế độ đa nhiệm (multitasking). NT = 1 chỉ ra rằng, một chương trình đang thực hiện là thường trú trong một chương trình khác và nó có liên kết thực với một chương trình trước nó.

- Bit 16, RF: Resume Flag (i80386) (cờ quay trở lại): Khi RF = 1, thì nó tạm thời bỏ qua các lỗi gỡ rối (debug faults) sao cho lệnh có thể được khởi động lại sau một lỗi gỡ rối mà không gây ra một lỗi gỡ rối khác.

- Bit 17, VM: Virtual Mode Flag (i80386) (cờ chế độ ảo): Nếu VM = 1, bộ vi xử lý sẽ làm việc trong chế độ thực 8086, để mô phỏng môi trường lập trình của bộ vi xử lý 8086.

- Bit 18, AC: Alignment Check Flag (từ 80486 đến Pentium) (cờ kiểm tra chỉnh chuẩn): Cờ có trong i80486, được sử dụng để chỉ ra rằng i80486 đã truy cập một từ ở địa chỉ lẻ (một từ ở địa chỉ byte lẻ) hoặc một từ kép cất giữ ở vùng nhớ biên không cho từ kép. Sự thiết lập cờ AC và bit AM trong thanh ghi điều khiển 0 (CR0) cho phép thực hiện kiểm tra khi tham chiếu bộ nhớ.

- Bit 19, VIF: Virtual Interrupt Flag (cờ ngắt ảo). VIF là ảnh của cờ I sử dụng cùng với VIF.

- Bit 20, VIP: Virtual Interrupt Pending Flag (cờ chờ đợi ngắt ảo). VIP cùng với VIF cho phép chương trình ứng dụng chạy trong môi trường đa nhiệm có các giá trị ảo của cờ I.

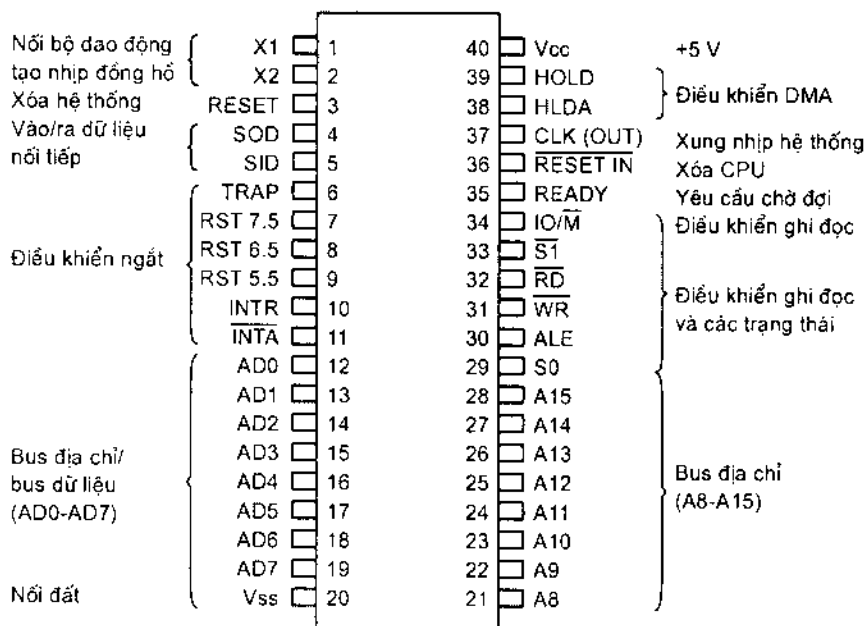
- Bit 21, ID: Identification Flag (cờ nhận dạng) (80486): khả năng của một chương trình thiết lập và xóa cờ ID chỉ ra rằng bộ xử lý hỗ trợ lệnh nhận dạng của CPU (CPUID).

Các bit còn lại của thanh ghi cờ của họ vi xử lý Intel không liệt kê ở đây là bỏ trống, không sử dụng tới. Nội dung của thanh ghi cờ của hệ thống i80386 và Pentium bao gồm cả nội dung thanh ghi cờ của họ i808X chứng tỏ rằng phần mềm viết cho họ i808X, trong đó có các xử lý cờ đều có thể chạy được trên i80X86 và Pentium.

2.4.3. Vi xử lý 8085

(1) Đóng vỏ và sơ đồ khối của 8085

Chip 8085 đóng vỏ gồm 40 chân DIP (hình 2.63) được dùng làm CPU trong các hệ thống máy tính cá nhân đầu tiên sử dụng hệ điều hành CP/M.

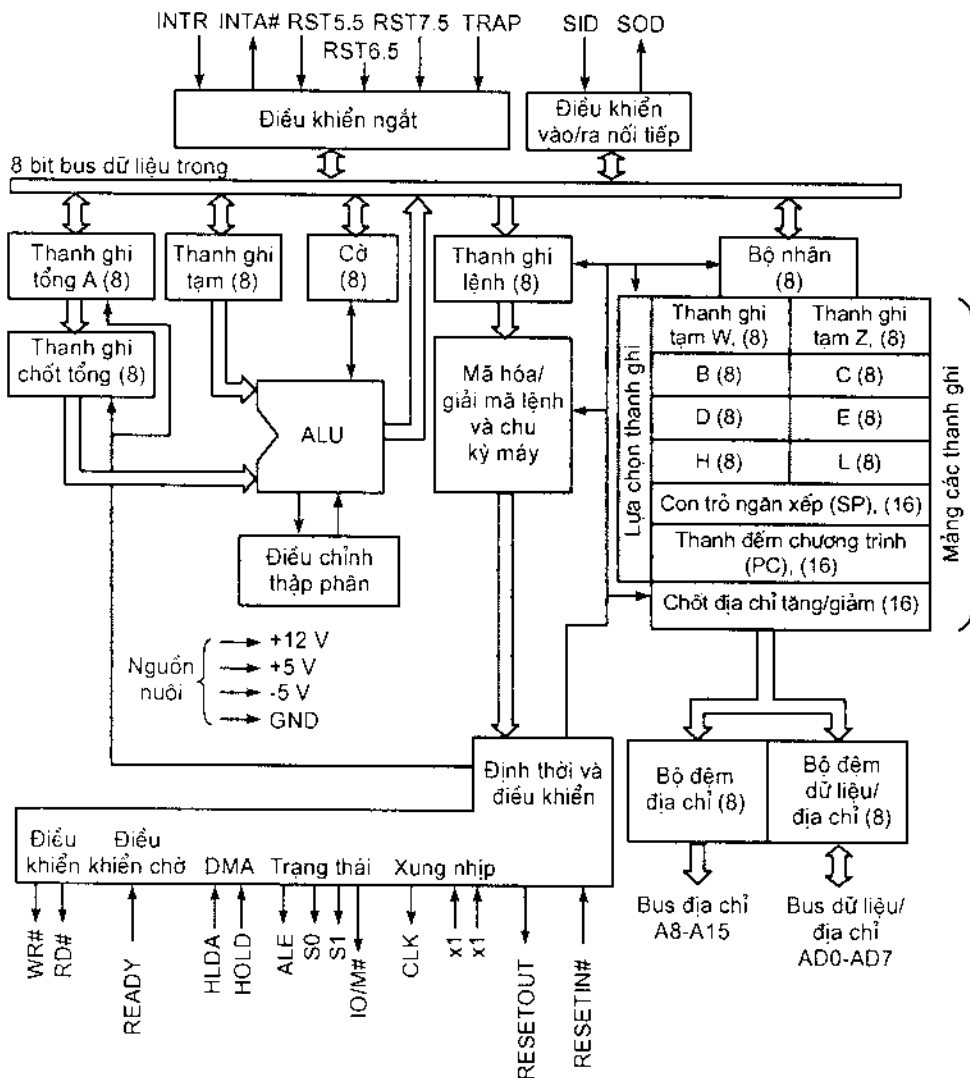


Hình 2.63: Các chân tín hiệu của 8085

8085 có các nhóm thanh ghi như sau:

- Nhóm thanh ghi chung: gồm các thanh ghi 8-bit: A, B, C, D, E, H, L, W, và Z.

Những thanh ghi này được sử dụng cho thao tác tính toán với thanh tổng A (Accumulator). Thao tác dữ liệu có thể với từng thanh ghi (8-bit) hay từng cặp thanh ghi (16-bit), như cặp BC, DE, HL. Để đánh địa chỉ ngăn nhớ có thể dùng các cặp thanh ghi DE và HL. Các thanh ghi W, Z dùng để lưu trữ tạm thời kết quả trung gian trong quá trình tính toán hoặc xử lý dữ liệu.



Hình 2.64: Sơ đồ khối chức năng của 8085

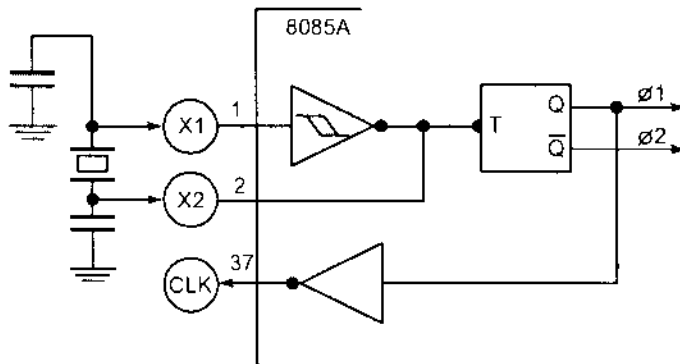
- Con trỏ ngăn xếp SP (Stack Pointer) 16-bit: chứa địa chỉ của đỉnh ngăn xếp.

- Thanh đếm chương trình PC (Program Counter) 16-bit: chứa địa chỉ lệnh máy tiếp theo sẽ thực hiện.

- Thanh ghi lệnh (Instruction Register, 8-bit): chứa mã lệnh nhận từ bộ nhớ trong chu kỳ đọc lệnh (instruction fetch) thông qua bus dữ liệu hệ thống (D0-D7) và 8-bit bus dữ liệu trong. Mã lệnh được dem chốt tại đây để giải mã tạo ra các tín hiệu điều khiển trong khối định thời và điều khiển (Timing & control).

Sơ đồ khối chức năng của 8085 và các tín hiệu được trình bày trong hình 2.64. Các khối chức năng và các thanh ghi trao đổi dữ liệu với nhau thông qua 8 bit bus dữ liệu trong. Sự trao đổi dữ liệu giữa các thanh ghi của 8085 với bộ nhớ và ngoại vi thông qua bus hệ thống. Bus hệ thống gồm bus dữ liệu, bus địa chỉ và bus điều khiển.

(2) Nhịp đồng hồ hệ thống của 8085



Hình 2.65: Logic tạo nhịp đồng hồ bên trong 8085A

Chip vi xử lý là CPU của hệ thống máy tính, và nó cần phải được hoạt động theo các nhịp thời gian chính xác. Từ nó sản sinh ra các tín hiệu điều khiển cho toàn bộ hệ thống máy tính. Nhịp đồng hồ hệ thống (system clock) chính xác cần phải được tạo ra cho chip vi xử lý để định thời các tín hiệu của hệ thống máy tính. Chip 8085 có bên trong mạch tạo nhịp đồng hồ cho hệ thống máy tính (hình 2.65). Mạch dao

động bên ngoài bằng thạch anh ghép nối với mạch tạo nhịp đồng hồ bên trong qua 2 đầu vào: X1 (chân 1), X2 (chân 2). 8085A có tần số nhịp là 3 MHz, và 8085AH tần số nhịp là 6 MHz.

(3) Định thời của 8085

a) Các chu kỳ thời gian của 8085

Máy tính vận hành theo chương trình. Chương trình gồm nhiều lệnh máy lưu trong bộ nhớ. Các lệnh máy lần lượt được đọc từ bộ nhớ vào vi xử lý để thực hiện. Kết quả thực hiện lệnh được chuyển trở lại bộ nhớ hoặc được gửi đến một thanh ghi bên trong nào đó. Thời gian để thực hiện xong một lệnh máy gọi là một chu kỳ lệnh (instruction cycle).

Chu kỳ lệnh gồm: thời gian đọc lệnh từ bộ nhớ (instruction fetch), và thời gian thực hiện lệnh (instruction execution), trong đó kể cả thời gian lưu kết quả trở lại bộ nhớ (nếu có xảy ra). Thời gian của chu kỳ lệnh có thể khác nhau phụ thuộc vào loại lệnh và tần số nhịp đồng hồ. Một chu kỳ lệnh đặc trưng của 8085 cần khoảng 3 μ s, nghĩa là, 8085 có thể thực hiện trung bình 330.000 lệnh trong 1 s. Một chu kỳ lệnh có thể gồm từ 1 đến 5 chu kỳ máy (machine cycle). Chu kỳ máy liên quan đến thao tác chuyển dữ liệu đi đâu và khi nào. Nó cần thiết phải có mỗi khi có một byte dữ liệu được chuyển giữa bộ nhớ hoặc ngoại vi và 8085A thông qua bus hệ thống. Ta xét ví dụ thực hiện lệnh OUT và IN của 8085 để khảo sát xem có bao nhiêu chu kỳ máy.

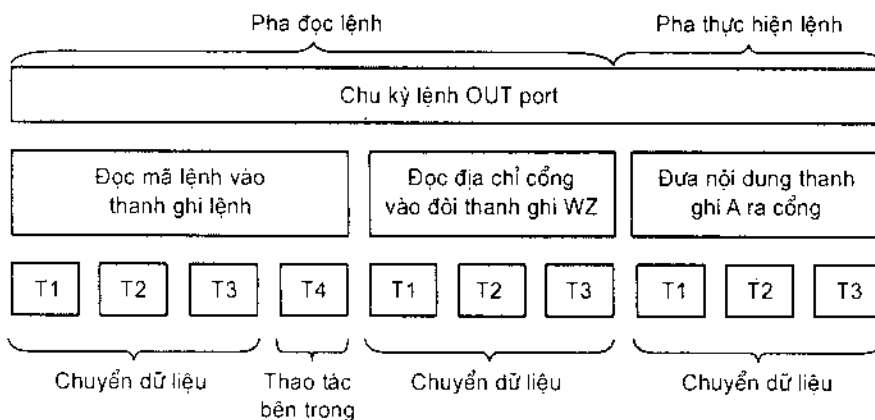
1. Trước hết cần có thời gian đọc lệnh từ bộ nhớ vào thanh ghi lệnh của 8085. Tất cả các lệnh ít nhất phải có mã lệnh (độ dài 8 bit), nghĩa là cần ít nhất 1 chu kỳ máy để đọc lệnh.

2. Lệnh OUT port (hoặc IN port) có phần địa chỉ port (địa chỉ toán hạng của lệnh OUT, IN). Mã lệnh chuyển vào thanh ghi lệnh, còn địa chỉ toán hạng chuyển vào cặp thanh ghi WZ (16 bit). Như vậy cần có 1 chu kỳ máy để chuyển dữ liệu (địa chỉ port) đến WZ, vì địa chỉ port chỉ có 8 bit. Địa chỉ Port được đưa ra bus dữ liệu (từ WZ) để giải

mã chọn cổng thiết bị ngoại vi. Đến đây giai đoạn đọc lệnh để giải mã lệnh kết thúc, và giai đoạn này cần 2 chu kỳ máy: chu kỳ máy 1 để đọc mã lệnh, chu kỳ máy 2 để đọc địa chỉ port.

3. Thực hiện lệnh OUT port (hoặc IN port) là quá trình vận chuyển 1 byte dữ liệu từ thanh ghi tổng A của CPU ra cổng ngoại vi (hay từ cổng ngoại vi vào thanh ghi tổng A của CPU). Quá trình này cần 1 chu kỳ máy.

Như vậy, để thực hiện lệnh OUT port (hoặc IN port) cần có 3 chu kỳ máy. Mỗi một chu kỳ máy được chia thành 3 cho tới 6 trạng thái (state). Một trạng thái được xác định là một chu kỳ nhịp đồng hồ (clock cycle). Chu kỳ nhịp đồng hồ được xác định bằng cách chia tần số thạch anh bên ngoài và là đơn vị nhỏ nhất của một vị trí thời gian. Trong hệ thống 8085AH, nhịp đồng hồ có tần số 6 MHz, thì chu kỳ nhịp đồng hồ kéo dài trong khoảng $1/3 \mu s$.



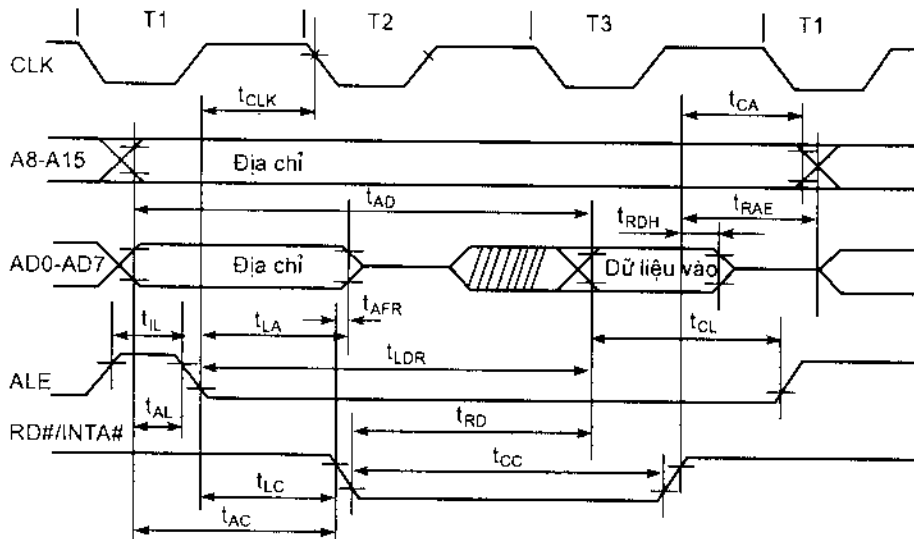
Hình 2.66: Chu kỳ thực hiện lệnh OUT port

Đồ thị thời gian thực hiện một lệnh OUT port với các chu kỳ nhịp được biểu diễn trong hình 2.66. Có 3 trạng thái cơ bản đầu tiên: T1, T2, T3 phải trải qua trong quá trình đọc lệnh vào thanh ghi lệnh. Thời gian T4 cần thiết để dành cho thời gian giải mã lệnh, ba trạng thái: T1, T2, T3 tiếp theo là chu kỳ máy 2 để đọc địa chỉ toán hạng vào

đôi thanh ghi WZ. Chu kỳ máy thứ 3 với T1, T2, T3 là thời gian chuyển byte dữ liệu từ thanh ghi A ra cổng ngoại vi. Hai chu kỳ máy cuối cùng của lệnh OUT port hoàn toàn dành để vận chuyển các byte dữ liệu mà không cần phải có thêm thời gian chờ đợi nào khác, nên chỉ cần có 3 chu kỳ nhịp đồng hồ.

b) Trạng thái chờ (Wait) và sẵn sàng (Ready) của 8085

Bộ nhớ chính (RAM, ROM) kết nối với 8085 phải có đảm bảo là tốc độ truy cập của nó phải đủ nhanh để phản xạ trong quãng thời gian yêu cầu của các thao tác lệnh ghi/đọc của 8085. Tuy nhiên, điều này không phải lúc nào cũng đạt được. Ví dụ, tương ứng với thao tác đọc dữ liệu từ bộ nhớ của 8085A, gần 250 ns sau khi một địa chỉ của ngăn nhớ của ROM được gửi lên các đường dây địa chỉ của bus địa chỉ, thì dữ liệu thực phải xuất hiện ở các đường dây vào dữ liệu của 8085A (hình 2.67).

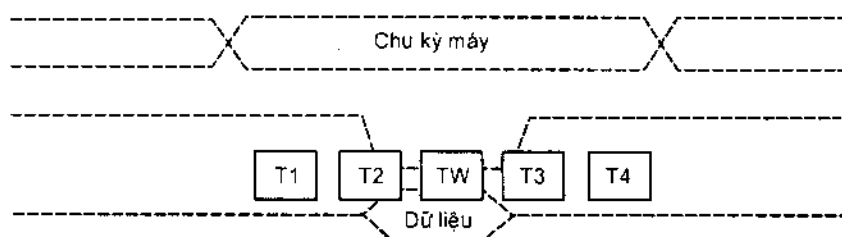


Hình 2.67: Đồ thị thời gian địa chỉ và đọc dữ liệu của 8085

Nhưng sẽ là vấn đề, nếu giả sử, ta phải chọn chip ROM có thể thời gian truy cập xấp xỉ 250 ns. Các tín hiệu địa chỉ từ 8085 phải đi qua các mạch giải mã chọn để tới được chip ROM, và nếu ta sử dụng

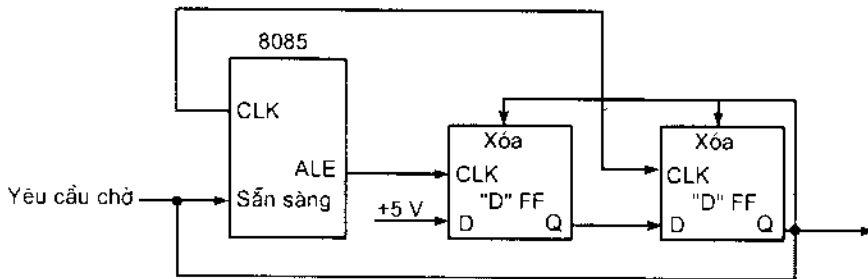
chip giải mã 8205 thì phải mất khoảng 20 ns trễ qua nó. Khi byte dữ liệu thực của ngăn nhớ của ROM tới được các đường vào dữ liệu của chip 8085A thì có thể đã muộn hơn khoảng 20 ns. Có thể chọn chip ROM nhanh hơn, nhưng chi phí sẽ cao.

Một giải pháp thực tế hơn, rẻ tiền hơn, khi kết nối với bộ nhớ tốc độ chậm, đó là đưa vào trạng thái chờ đợi cho chip vi xử lý (wait). Tín hiệu chờ được logic tạo tín hiệu chờ được tạo ra. Một trạng thái chờ đợi (T_w) có thời gian bằng một chu kỳ máy, và tùy thuộc vào tốc độ truy cập của ROM, RAM mà ta đưa vào một hay một số trạng thái chờ đợi. Trạng thái chờ đợi được chen vào giữa 2 chu kỳ máy T_2 và T_3 (hình 2.68). Trong khoảng T_2 , địa chỉ đã được giải mã và dữ liệu bắt đầu được đọc (hình 2.67). Trong thời gian T_3 dữ liệu được đọc xong và đưa lên bus dữ liệu. T_w đặt giữa T_2 và T_3 phải đảm bảo các đường địa chỉ, dữ liệu và điều khiển vẫn còn tích cực.



Hình 2.68: Chu kỳ máy được bổ sung thời gian chờ

Yêu cầu chờ thông qua tín hiệu sẵn sàng (READY) ở mức thấp một yêu cầu chờ được gửi tới 8085, như thế có nghĩa là bộ vi xử lý chưa sẵn sàng. Khi nào READY còn ở mức thấp thì trạng thái chờ còn tiếp tục duy trì và các trạng thái T_w còn tiếp tục bổ sung thêm giữa T_2 và T_3 . Ví dụ một mạch tạo ra tín hiệu yêu cầu chờ đối với từng chu kỳ máy cho trong hình 2.69.

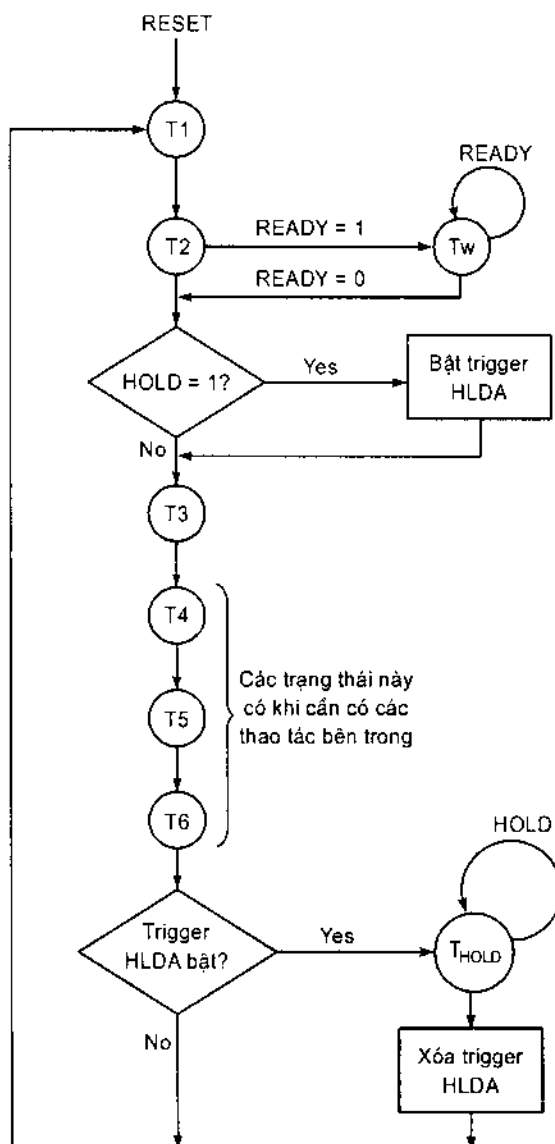


Hình 2.69: Mạch tạo tín hiệu chờ cho 8085

c) Quá trình chuyển đổi trạng thái của 8085

Khi 8085 nhận tín hiệu RESET IN# (xóa) tại chân 36, nó khởi đầu bằng chu kỳ máy thứ nhất, bắt đầu với trạng thái T1 (chu kỳ nhịp đầu tiên). Đối với mỗi chu kỳ máy, trong trạng thái T2, 8085 kiểm tra tín hiệu vào READY xem có yêu cầu chờ đợi hay không. Nếu READY ở mức thấp, thì nó đưa vào 1 hoặc nhiều hơn trạng thái Tw giữa 2 trạng thái T2 và T3. Khi READY chuyển lên mức cao, 8085 thoát khỏi trạng thái Tw và kiểm tra tín hiệu vào HOLD xem có yêu cầu DMA hay không. Nếu HOLD ở mức cao thì nó thiết lập trạng thái bật cho trigger HLDA bên trong nó rồi chuyển đến trạng thái T3. Nếu HOLD ở mức thấp (không có yêu cầu DMA), 8085 chuyển đến trạng thái T3 ngay. Tiếp sau T3, là các trạng thái T4, T5 và T6 của chu kỳ máy. Các trạng thái T4, T5, T6 có hay không phụ thuộc vào yêu cầu của thao tác bên trong 8085 (phụ thuộc vào lệnh máy thực hiện). Sau trạng thái cuối cùng của chu kỳ máy, 8085 kiểm tra trạng thái của trigger HLDA. Nếu trigger đang bật, thì 8085 chuyển vào chế độ HOLD, nó treo các tín hiệu của bus và dừng mọi tác động ra bên ngoài (từ bỏ quyền điều khiển bus). Khi chế độ HOLD được giải phóng, hoặc trạng thái T cuối cùng của chu kỳ máy kết thúc, 8085 trở về đầu của một chu kỳ máy tiếp theo (trở về đầu của biểu đồ trong hình 2.70). Khi thực hiện các thao tác nội bộ ở các trạng thái T4, T5 và T6, 8085 không cần sử dụng tới bus hệ thống. Điều này có nghĩa là, trong quá trình thực hiện DMA, có thể xảy ra các thao tác bên trong 8085. Mỗi một chu kỳ lệnh có ít nhất một trạng thái thao tác bên trong (trạng thái

T4 của thời gian đọc lệnh được giành để giải mã lệnh). Quá trình này gọi là lấy lén chu kỳ (cycle stealing).

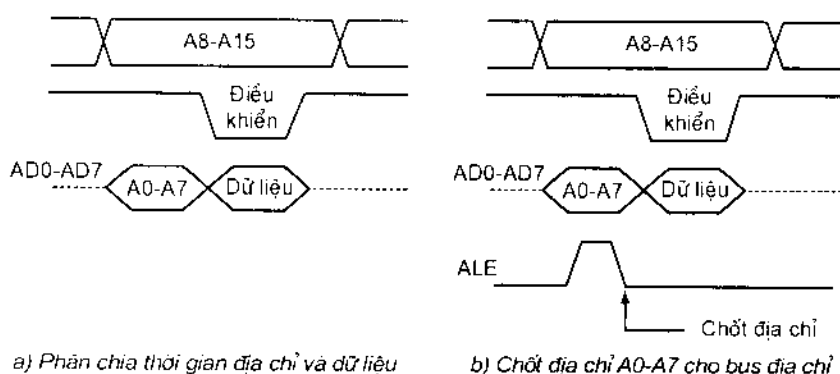


Hình 2.70: Chuyển đổi trạng thái trong 8085

d) Ghép và phân chia theo thời gian địa chỉ và dữ liệu của 8085

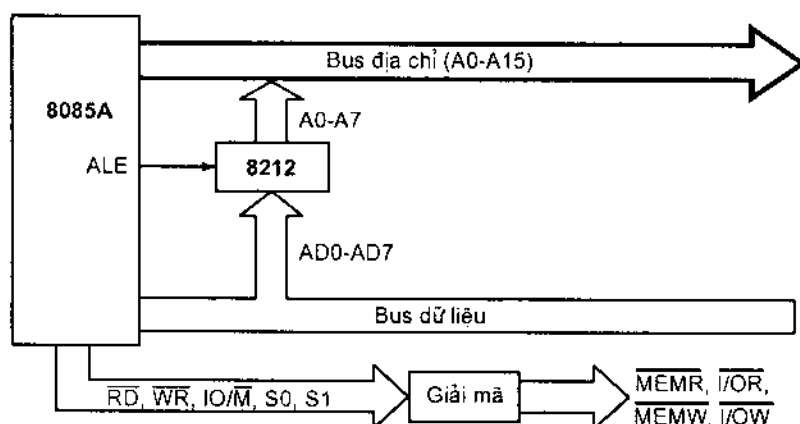
Quá trình thực hiện lệnh được tiến hành thứ tự: đọc lệnh, giải mã lệnh, đưa ra địa chỉ toán hạng rồi mới đến thao tác với các dữ liệu. Dữ

liệu không xuất hiện trên bus dữ liệu khi quá trình địa chỉ chưa xong. Lợi dụng tính chất này và để tiết kiệm 8085 sử dụng cách ghép chung 8 đường địa chỉ thấp (A0-A7) với 8 đường dữ liệu (D0-D7) thành 8 đường chung AD0-AD7, và phân chia thời gian sử dụng cho địa chỉ và dữ liệu. Sơ đồ xung ở hình 2.67 chỉ rõ: trong chu kỳ T1, các đường AD0-AD7 có giá trị địa chỉ, còn trong chu kỳ T2 và T3, các đường dây AD0-AD7 là giá trị dữ liệu. Đồng thời trong chu kỳ T1, các đường dây địa chỉ cao A8-A15 được đưa ra bus địa chỉ. Như thế kết hợp lại ta có 16 đường địa chỉ (A0-A15) cho ngăn nhớ (địa chỉ toán hạng) (hình 2.71).



Hình 2.71: Phân chia AD0-AD7 và chốt địa chỉ A0-A7

Để chốt các đường địa chỉ thấp (A0-A7) cho bus địa chỉ (A0-A15) và sau đó có thể dùng đường ghép chung AD0-AD7 cho dữ liệu, cần phải chốt A0-A7. Mạch chốt 8212 với xung điều khiển chốt ALE trong mỗi chu kỳ máy là một thí dụ đơn giản mạch chốt địa chỉ (hình 2.72). Tín hiệu ALE chuyển lên mức cao khi các đường địa chỉ đã có giá trị ổn định. Khi đã chốt xong A0-A7 (bằng sườn xung xuống của ALE), đường dây chung AD0-AD7 trở thành bus dữ liệu của hệ thống. Còn A0-A7 và A8-A15 tạo nên đầy đủ đường địa chỉ cho bus địa chỉ. Chỉ vào thời điểm này, bus địa chỉ đảm bảo giải mã chọn ngăn nhớ (toán hạng) đầy đủ. Còn bus dữ liệu sẵn sàng đón nhận dữ liệu đưa lên nó.



Hình 2.72: Ghép và phân chia theo thời gian các đường địa chỉ/dữ liệu của 8085

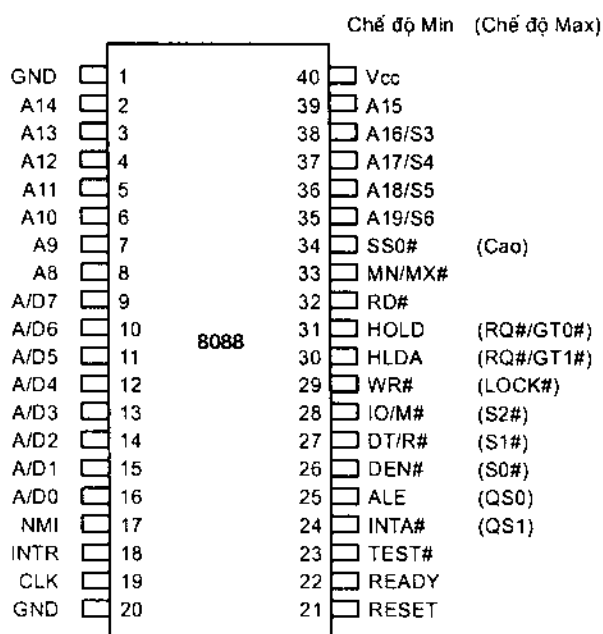
(4) Tập lệnh của 8085

Tập lệnh của 8085 gồm có 74 kiểu lệnh (xem phần phụ lục). 74 kiểu lệnh được phân thành 5 nhóm chính: nhóm lệnh vận chuyển dữ liệu, nhóm lệnh số học, nhóm lệnh logic, nhóm lệnh rẽ nhánh, nhóm lệnh ngăn xếp, vào/ra, và điều khiển hệ thống. Có các kiểu đánh địa chỉ: tuyệt đối, thanh ghi, ngay lập tức, trực tiếp, gián tiếp thanh ghi.

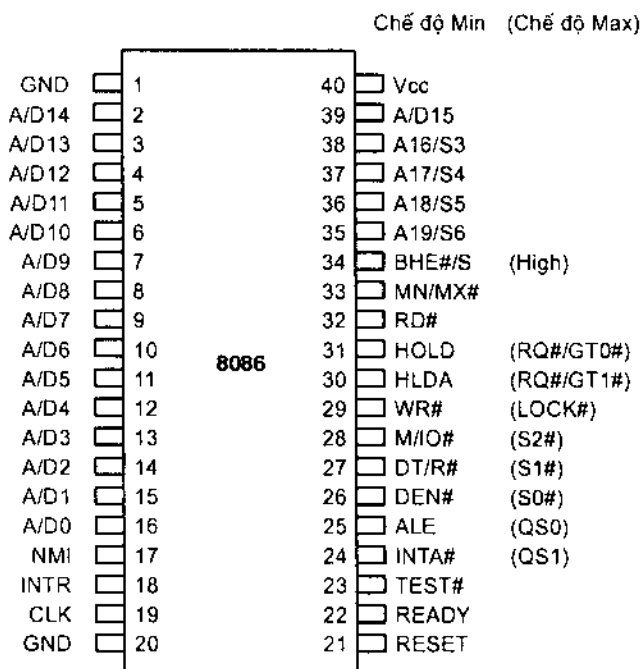
2.4.4. Vi xử lý 8086/8088

(1) Đóng vỏ và sơ đồ khối của 8086/8088

Các chip 8086/8088 là sự phát triển tiếp theo của 8080/8085. Bảng 2.3 đã có so sánh các đặc tính của các loại i808X. 8086/8088 giống nhau về cấu trúc bên trong, nhưng bên ngoài thì các chân tín hiệu có đôi chút khác nhau vì, riêng 8088 có bus dữ liệu bên ngoài 8-bit, trong khi đó 8086 có bus dữ liệu trong/ngoài đều 16-bit. 8086/8088 là những bộ vi xử lý 16-bit đầu tiên của Intel. Hình 2.73 là đóng vỏ DIP và các tín hiệu của 8088 và hình 2.74 là đóng vỏ và các tín hiệu của 8086. Chỉ có sự khác nhau đối với các tín hiệu địa chỉ cao (A8-A15), trong đó, đối với 8086 chúng được ghép chung với 8 đường dữ liệu cao (D8-D15) để tạo thành A/D8-A/D16. Chính vì sự ghép chung các đường địa chỉ và dữ liệu này (byte cao) nên cần phải có logic chốt byte địa chỉ cao bên ngoài trong hệ thống 8086.

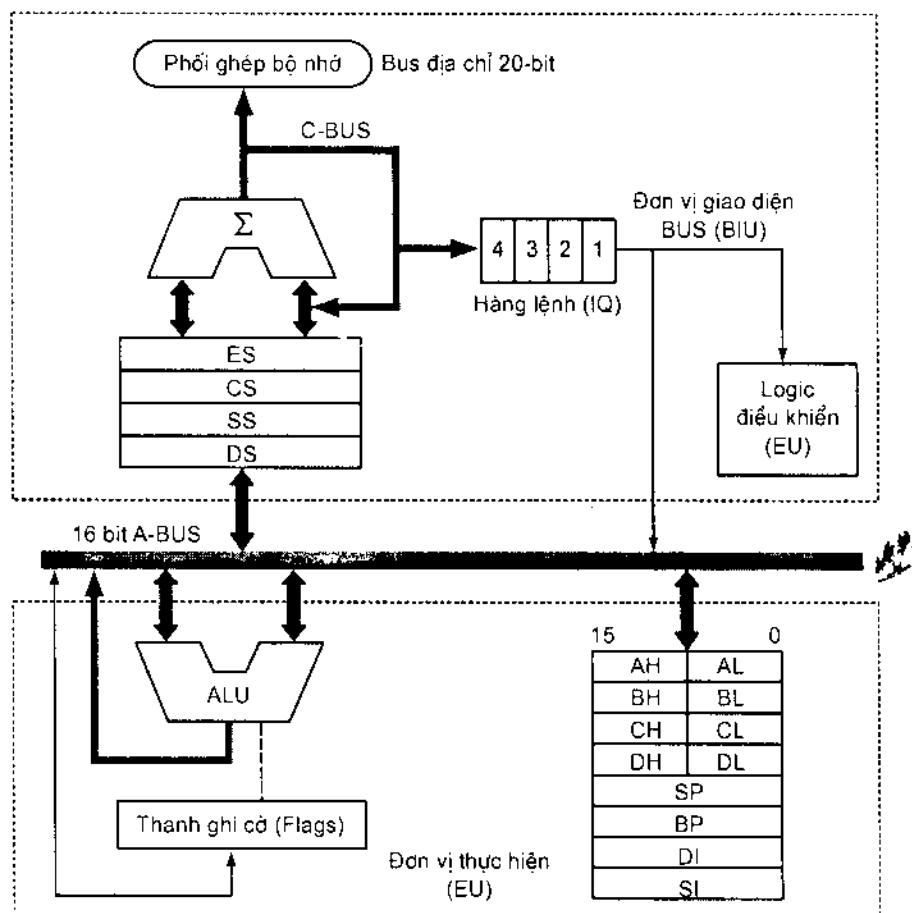


Hình 2.73: Đóng vỏ DIP 40 chân của 8088



Hình 2.74: Đóng vỏ DIP 40 chân và các tín hiệu của 8086

(trong 8086), logic điều khiển bus (trong 8086), và điều khiển EU (trong 8088).

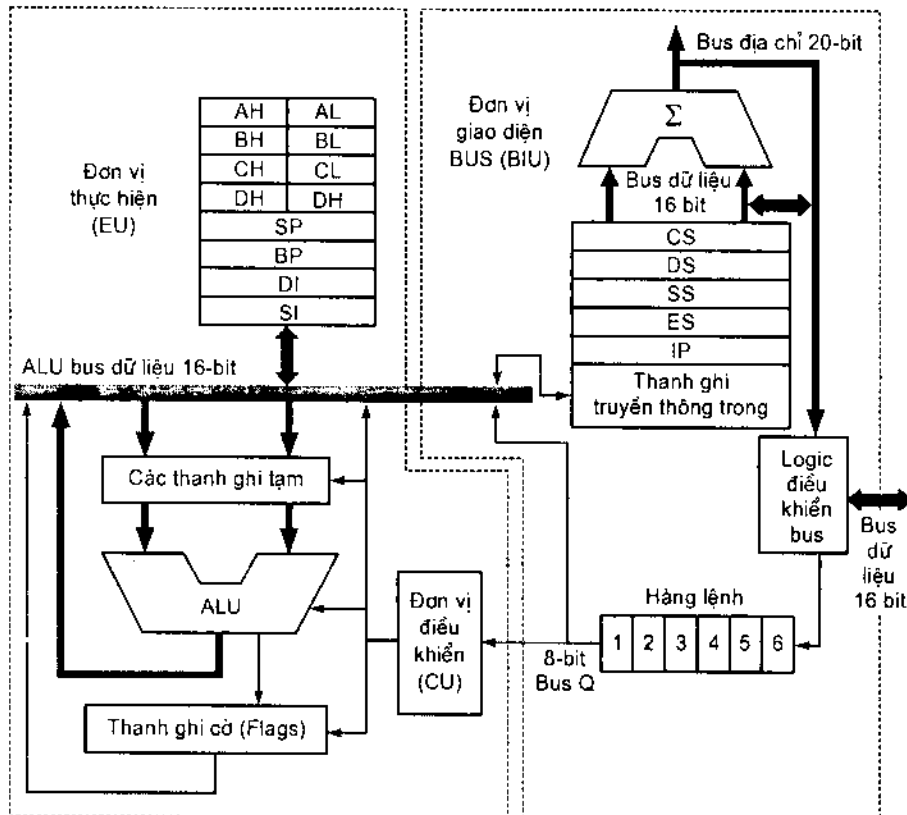


Hình 2.76: Sơ đồ khối của Intel 8088

Các đơn vị EU và BIU liên kết với nhau thông qua các bus bên trong: 16-bit A-BUS (trong 8088), 16-bit ALU bus dữ liệu (trong 8086). BIU có nhiệm vụ bảo đảm cho bus được sử dụng hết dung lượng để tăng tốc độ thao tác. Sự tăng tốc độ thao tác được thực hiện theo hai cách:

- + Cách 1: Tiễn đọc lệnh vào hàng lệnh (IQ) để EU xử lý liên tục.

+ Cách 2: BIU đảm nhận tất cả chức năng điều khiển bus để EU rảnh rỗi tập trung vào thực hiện các lệnh và xử lý dữ liệu.



Hình 2.77: Sơ đồ khối của Intel 8086

Bên trong 8086/8088 có các nhóm thanh ghi như sau:

- Các thanh ghi chung: gồm 4 thanh ghi chung 16 bit AX, BX, CX, DX. Những thanh ghi này có thể được thao tác theo từ 16-bit, theo byte thấp: AL, BL, CL, DL và theo byte cao: AH, BH, CH, và DH.

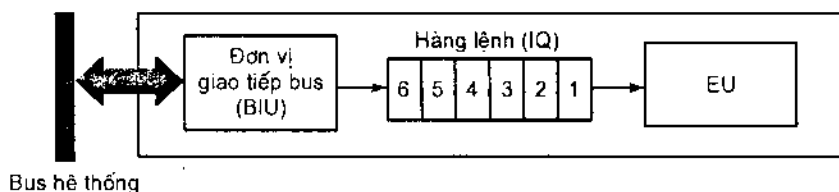
- Các thanh ghi con trỏ và chỉ số 16-bit: con trỏ ngăn xếp (SP), con trỏ cơ sở (BP), chỉ số nguồn (SI) và chỉ số đích (DI). SI và DI dùng cho các lệnh vận chuyển các khối dữ liệu giữa các vùng nhớ. SP và BP được dùng cho các thao tác với ngăn xếp.

- Các thanh ghi đoạn (segment registers) 16-bit: 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 phụ ES (Extra Segment). Những thanh ghi này chứa địa chỉ cơ sở (địa chỉ đầu) của đoạn nhớ có tên tương ứng, vì tổ chức bộ nhớ trong 8086/8088 là theo đoạn và trang, mỗi đoạn có dung lượng 64 kbyte.

- Con trỏ lệnh IP (Instruction Pointer) 16-bit: giống như PC của 8085, IP chứa địa chỉ của lệnh tiếp theo.

- Thanh ghi cờ 16-bit: nội dung và ý nghĩa các bit cờ đã được trình bày phần trên.

- Hàng lệnh IQ (Instruction Queue): thay vì thanh ghi lệnh của 8085, trong 8086/8088 có hàng lệnh có thể tiếp nhận 4 (trong 8088) hoặc 6 (trong 8086) lệnh để lần lượt đưa ra giải mã và thực hiện. Như vậy tốc độ thực hiện lệnh sẽ tăng lên đáng kể vì để thực hiện những lệnh tiếp theo không phải chờ đợi thời gian đọc lệnh. Những lệnh tiếp theo đã được đọc trước từ bộ nhớ vào hàng đợi IQ trong khi một lệnh máy đang được CPU thực hiện. Cơ cấu tiên đọc lệnh được mô tả trong hình 2.78.



Hình 2.78: Cơ cấu tiên đọc lệnh vào hàng lệnh trong 8086/8088

Các thanh ghi bên trong 8086/8088 mô tả ở hình 2.79 và 2.80. Số lượng thanh ghi trong 8086/8088 nhiều hơn so với 8085, và những thanh ghi này bổ sung thêm một số tính năng về quản lý bộ nhớ và một số lệnh máy, qua đó có thêm những khả năng lập trình hơn.

Tần số làm việc của 8086/8088 cao hơn gấp đôi so với 8085A. 8086 có công suất mạnh hơn so với 8088 về tốc độ xử lý và trao đổi dữ liệu (gấp đôi so với 8088), tuy nhiên, do thị trường các thiết bị ngoại vi và các ứng dụng lúc đó chỉ ở mức độ 8 bit nên 8086 không tìm được

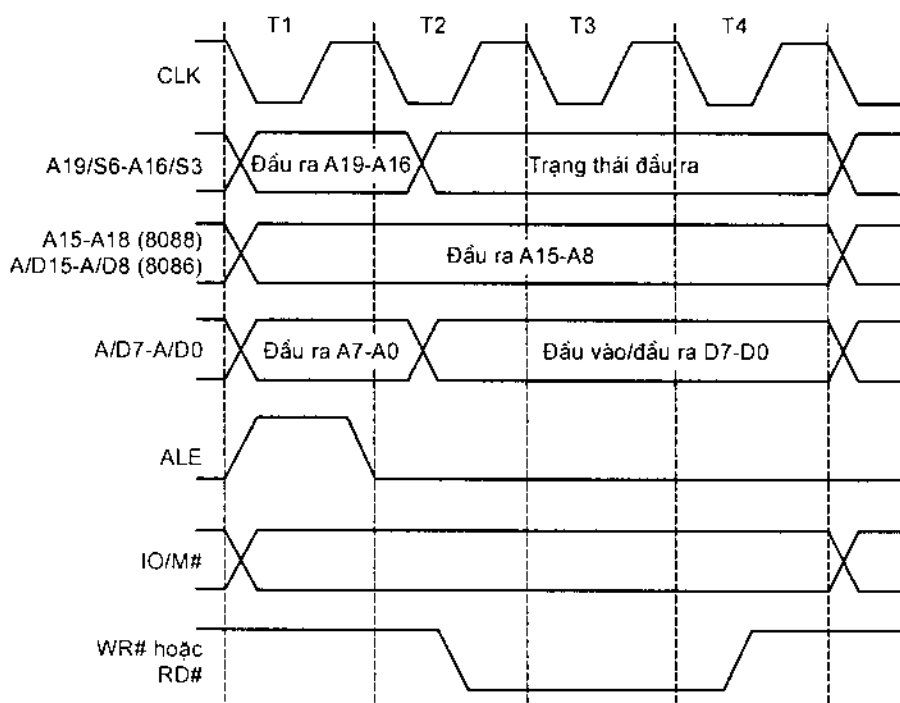
Tập lệnh của 8088 hoàn toàn giống tập lệnh của 8086. Khi sử dụng 8086, 16-bit bus dữ liệu ngoài của nó không phát huy hiệu quả trong khi vẫn cần phải có 2 chu kỳ máy cho thực hiện các lệnh I/O 16-bit. Các chu kỳ thời gian thực hiện lệnh của 8086/8088 tương tự như của 8085, nhưng có sự khác biệt trong các tín hiệu điều khiển. 8086/8088 không có các tín hiệu ngắt đặc biệt RST 7.5 - RST 5.5, mà chỉ có một đường tín hiệu vào yêu cầu ngắt INTR. Tín hiệu ngắt không che được TRAP của 8085 được thay bằng NMI. Về bản chất thì chúng tương tự nhau. Cũng vì vậy, chỉ có 1 tín hiệu RESET được lấy từ ngoài vào để xóa 8086/8088 về trạng thái ban đầu. Điều đặc biệt là 8086/8088 có nhiều tín hiệu trạng thái hơn và phân biệt ở hai chế độ làm việc: chế độ tối thiểu (minimum mode) và chế độ tối đa (maximum mode). Chế độ tối thiểu được sử dụng khi kết nối với ít thiết bị ngoài và chỉ duy nhất một chip vi xử lý 8086/8088 làm CPU. Chế độ tối đa dùng cho trường hợp kết nối đa xử lý (ví dụ kết nối 8086/8088 với đồng xử lý toán học 8087) và nhiều thiết bị ngoại vi. Đặt chế độ này thông qua đặt điện thế cho chân tín hiệu MN/MX#. Nếu MN/MX# = 0 thì đặt chế độ tối đa, MN/MX# = 1 thì đặt chế độ tối thiểu.

(2) Định thời (timing) trong 8086/8088

Các chu kỳ lệnh, chu kỳ máy và trạng thái thời gian của 8086/8088 tương tự như của 8085.

Chu kỳ lệnh là quãng thời gian thực hiện xong hoàn toàn một lệnh máy. Trong mỗi chu kỳ lệnh có một hoặc một số chu kỳ máy.

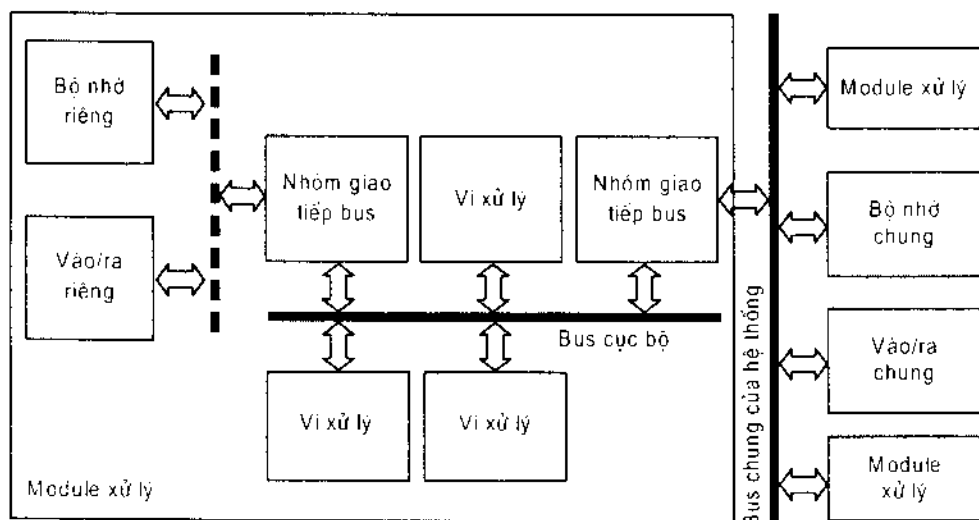
Mỗi chu kỳ máy thực hiện một cuộc chuyển đơn thuần dữ liệu. Trong mỗi chu kỳ máy có 3 hoặc hơn trạng thái. Mỗi trạng thái là một chu kỳ nhịp đồng hồ đơn thuần. Đồ thị của chu kỳ máy của 8086/8088 được mô tả trong hình 2.81.



Hình 2.81: Đồ thị xung chu kỳ máy của 8086/8088

(3) Kết nối đa xử lý (Multiprocessing) của 8086/8088

Đa xử lý là một đặc điểm để nâng cao tốc độ xử lý của máy tính mà 8088/8086 đã đưa vào. Trong đó, một số chip vi xử lý có thể kết nối với nhau trong một cấu trúc phân cấp theo nguyên tắc chủ/tớ (master/slave). Để kết nối một số chip 8086/8088 trong hệ thống, phải đặt chân tín hiệu MN/MX#="0" (chế độ tối đa). Kết nối giữa chip vi xử lý chủ (8086/8088) với các chip vi xử lý thợ (ví dụ, 8086/88 hoặc đồng xử lý 8087) được thực hiện thông qua bus cục bộ. Bus cục bộ (Local bus) là môi trường để các chip vi xử lý liên kết thông tin với nhau. Bộ nhớ và các tài nguyên vào/ra của bất kỳ một module xử lý nào (processing module) có thể kết nối trên bus riêng của hệ thống (private system bus). Bus công cộng của hệ thống (public system bus) là môi trường để các thiết bị vào/ra và xử lý công cộng kết nối. Hình 2.82 là ví dụ một hệ thống máy tính gồm nhiều vi xử lý và các thiết bị nhớ hoặc vào/ra riêng và công cộng.

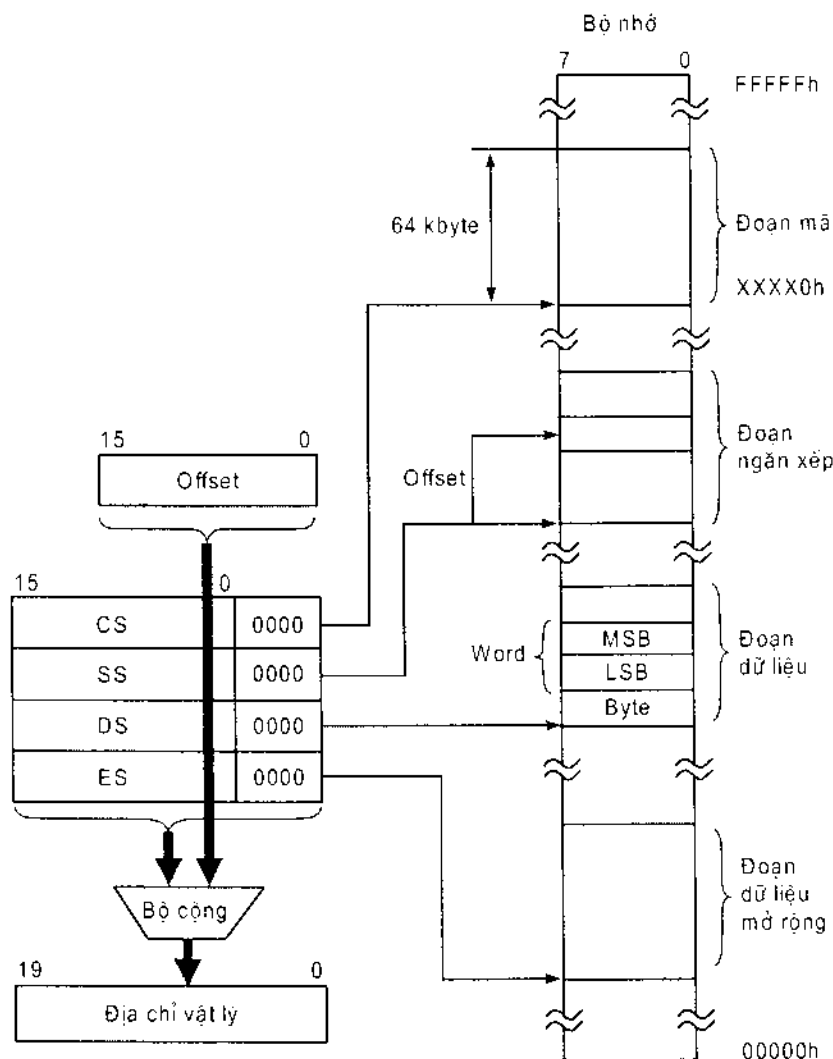


Hình 2.82: Hệ thống máy tính đa xử lý 8086/8088

(4) Phân đoạn bộ nhớ trong hệ thống 8086/8088

Để các chương trình chạy trên môi trường đa nhiệm, chúng phải được xây dựng thành các module nhỏ chiếm ít dung lượng nhớ và dễ được cất giữ hoặc phân bố lại ở bất kỳ vùng nhớ nào. Bài toán này được giải quyết cho 8086/8088 bằng cách tạo cơ chế phân đoạn bộ nhớ (segmentation) và đánh địa chỉ theo đoạn nhớ nhờ các thanh ghi đoạn 16-bit: CS, DS, SS, và ES. Phân đoạn nhớ của 8086/8088 được giải thích bằng sơ đồ trên hình 2.83. Các thanh ghi đoạn: CS, SS, DS, ES chứa các địa chỉ cơ sở của các đoạn tương ứng (segment base address). Địa chỉ cơ sở đoạn là địa chỉ 16-bit, do đó, dung lượng tối đa của đoạn sẽ là $64 \text{ kbyte} = 2^{16}$. Tổng dung lượng nhớ cơ sở của 8086/8088 là $256 \text{ kbyte} = 64 \text{ kbyte} \times 4$. Vị trí của ngăn nhớ so với địa chỉ đầu tiên của đoạn là một giá trị tương đối và được gọi là số bù thêm (Offset). Giá trị Offset lớn nhất là FFFFh . Để xác định các giá trị Offset cần phải dùng 16 bit.

Địa chỉ vật lý 20-bit thực của một ngăn nhớ trong hệ thống 8086/8088 được xác định bằng phép cộng trong bộ cộng (ADDER) của 8086/8088 (ADDER) nội dung được dịch trái 4 bit của thanh ghi đoạn (địa chỉ cơ sở đoạn) (tức là nhân với 16) với giá trị 16-bit Offset. Chúng ta sẽ xét vấn đề này trong chương 3 kỹ hơn.



Hình 2.83: Phân đoạn bộ nhớ của 8086/8088

(5) Tập lệnh và khuôn dạng lệnh của 8086/8088

Tập lệnh của 8088/8086 (xem phần phụ lục) tương thích với tập lệnh của 8085 (bộ vi xử lý ra đời trước chúng), nhưng có một số bổ sung mới trong các lệnh nhân và chia, lệnh chuỗi, các lệnh dùng cho chế độ đa xử lý, các lệnh xử lý bit, các lệnh tính toán với các số có dấu. Tập lệnh 8086/8088 có khoảng 300 kiểu lệnh và chúng đặc biệt hiệu quả cho dịch hợp ngữ và các ngôn ngữ lập trình bậc cao.

2.4.5. Đồng xử lý toán học 8087

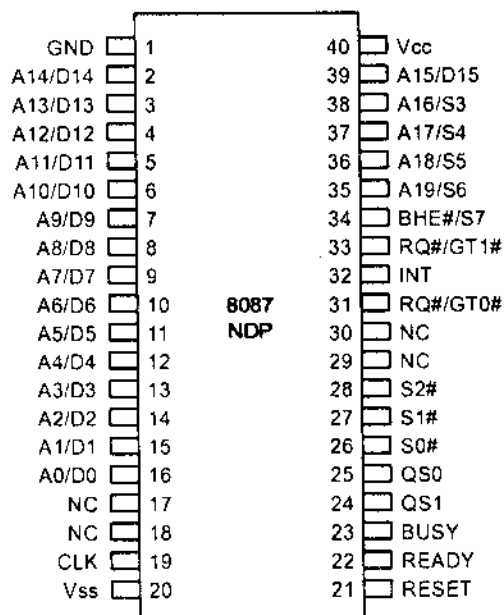
(1) Đóng vỏ và sơ đồ khối của 8087

Chip vi xử lý 8087 là đồng xử lý toán học dùng chung với 8086/8088 trong máy vi tính, trong đó 8087 là vi xử lý thợ (slave) còn 8086/8088 là vi xử lý chủ (master). 8087 chuyên thực hiện các phép tính các số với dấu phẩy động thuộc nhiều loại dữ liệu khác nhau. 8086/8088 thực hiện các phép tính này thông qua thực hiện các mô phỏng, do đó tốc độ thực hiện chúng rất chậm. Nếu dùng 8087 sẽ tăng tốc độ tính toán các số dấu phẩy động lên rất nhiều. Bảng 2.6 so sánh thời gian thực hiện các phép tính số dấu phẩy động với các loại dữ liệu khác nhau bằng 8087 và 8086/8088 (5 MHz)

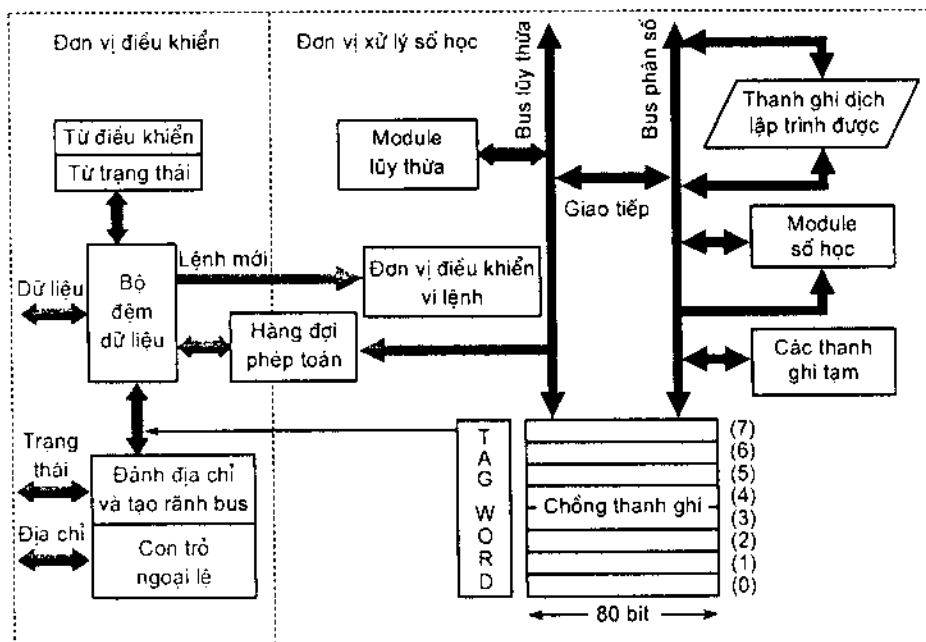
Bảng 2.6: Thời gian thực hiện phép tính số dấu phẩy động của 8087 và 8086/8088

Lệnh	Thời gian thực hiện (μ s) (5 MHz)	
	8087	8086/8088
Nhân các số chính xác đơn	19	1600
Nhân các số chính xác kép	27	2100
Cộng	17	1600
Chia các số chính xác đơn	39	3200
So sánh	9	1300
Nạp số chính xác đơn	9	1700
Cất giữ số chính xác đơn	18	1200
Khai căn bậc hai	36	19600
Hàm lượng giác tang	90	13000
Hàm mũ	100	17100

Công nghệ vào thời kỳ ra đời 8086/8088 không cho phép cấy 8087 vào bên trong 8086/8088 như trong i486 và Pentium, do đó phải chế tạo chip riêng để thực hiện phép tính số dấu phẩy động. Hình 2.84 là đóng vỏ DIP và 40 chân tín hiệu của 8087. Các đường tín hiệu địa chỉ và dữ liệu được ghép chung từ A0/D0 đến A19/D15. Như vậy, 8087 có thể trao đổi dữ liệu theo byte và theo từ 16 bit. Các tín hiệu trạng thái S0#, S1#, S2# và S3, S4, S5, S6, S7 có trong chế độ làm việc tối đa của 8086/8088. Nhịp đồng hồ CLK của 8087 có tần số như tần số của nhịp đồng hồ 8086/8088, bởi vì 8087 được thiết kế để làm việc cùng với 8086/8088 trong hệ thống máy vi tính.



Hình 2.84: Đóng vỏ DIP 40 chân và các tín hiệu của chip đồng xử lý toán học 8087



Hình 2.85: Sơ đồ khối chức năng của 8087

Hình 2.85 là sơ đồ khối chức năng của 8087. Trong 8087 có 2 đơn vị chức năng chính: đơn vị điều khiển (control unit) và đơn vị xử lý số học (numeric execution unit).

(2) Tập thanh ghi bên trong của 8087

Tập thanh ghi bên trong 8087 được tổ chức theo cơ chế ngăn xếp (register stack), và có khuôn dạng như trong hình 2.86. Trong cơ chế ngăn xếp dữ liệu được đẩy vào (push) đỉnh ngăn xếp, vị trí ST(0), và những dữ liệu cũ trong ngăn xếp được tự động đẩy xuống phía đáy ngăn xếp. Khi đọc ra, dữ liệu được lấy từ đỉnh ST(0), những dữ liệu còn lại trong ngăn xếp tự động di lên đỉnh.

	79	78	64	63	0
ST(0)					
ST(1)					
ST(2)					
ST(3)					
ST(4)					
ST(5)					
ST(6)					
ST(7)					
	Tín hiệu	Hàm mũ			Logarit

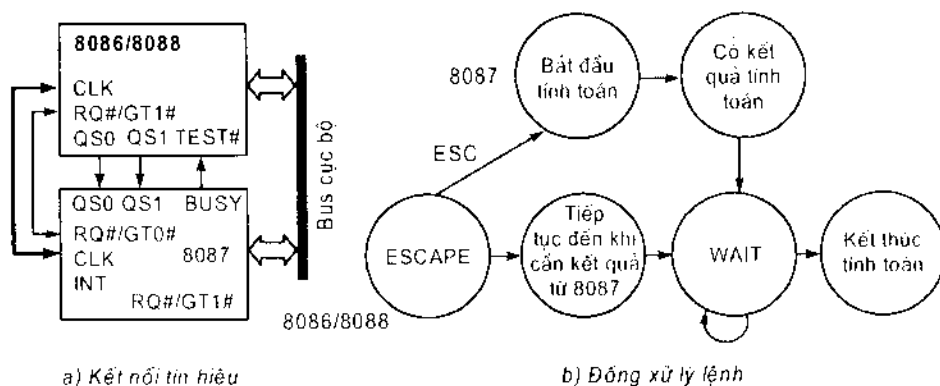
Hình 2.86: Tập thanh ghi bên trong của 8087

(3) Kết nối 8087 và 8086/8088

Sự liên kết giữa 8086/8088 với 8087 được thực hiện cả phần cứng và phần mềm và được mô tả trong hình 2.87.

Về phần cứng, tín hiệu RQ#/GT0# của 8087 thực hiện giám sát truy cập bus, về phần mềm, 8087 theo dõi các lệnh được 8086/8088 đọc ra có tiền tố là ký hiệu đặc biệt ESCAPE (ESC) hay không. Nếu tiền tố của lệnh máy có ký tự ESC thì 8087 tiếp nhận lệnh đưa từ 8086/8088 và sẽ bắt đầu quá trình thực hiện lệnh, còn 8088/8086 thì vẫn tiếp tục xử lý riêng cho đến lúc nó cần đến kết quả tính toán từ 8087. Lúc này 8086/8088 bước vào trạng thái chờ đợi WAIT (Lệnh WAIT chờ tín hiệu TEST). Khi 8087 đã tính xong và đưa kết quả vào

bộ nhớ, nó gửi tín hiệu $BUSY = 0$ đến chân $TEST\#$ để thông báo cho 8086/8088. Trạng thái chờ của 8086/8088 kết thúc khi $TEST\#$ khẳng định ở mức 0, và 8086/8088 tiếp tục quá trình xử lý của nó. Trong chế độ tối đa, $INTA\#$ và ALE trở thành hai tín hiệu trạng thái $QS1$ và $QS0$, đó là 2 bit trạng thái của bộ vi xử lý trong quá trình thực hiện lệnh. Chúng cần thiết cho bộ đồng xử lý và các thiết bị ngoại vi làm việc kết hợp chặt chẽ với bộ vi xử lý.



Hình 2.87: Kết nối 8086/8088 với đồng xử lý toán học 8087

Quá trình tính toán của 8087 thực hiện theo 3 bước như sau:

1. Dữ liệu được nạp từ bộ nhớ chính của hệ thống vào tập hợp các thanh ghi bên trong (register stack).
2. 8087 xử lý các dữ liệu vừa được nạp.

Kết quả được cất trữ lại vào bộ nhớ chính của hệ thống.

(4) Tập lệnh của 8087

Tập lệnh của 8087 (bảng 2.7) gồm nhiều lệnh tính toán với các số nguyên, BCD và dấu phẩy động. Chúng được xem là các lệnh mở rộng tập lệnh của 8086/8088. Khi viết chương trình, trong đó có sử dụng cả các lệnh của 8087, và 8088, thì thứ tự chương trình không cần bận tâm. Tất cả các tên lệnh của tập lệnh 8087 đều có chữ cái đầu là "F", nghĩa là Floating-point.

Bảng 2.7: Tập lệnh của 8087

Vận chuyển số thực (Real Transfers)		Hàm lượng giác (Transcendental)	
FLD	Load real	FPTAN	Partial tangent
FST	Store real	FPATAN	Partial arctangent
FSTP	Store real and pop	F2XM1	$2^x - 1$
FXCH	Exchange registers	FYL2X	$Y \cdot \log_2 X$
Vận chuyển số nguyên (Integer Transfers)		FYL2XP1	$Y \cdot \log_2 (X+1)$
FILD	Integer load	So sánh (Comparison)	
FIST	Integer store	FCOM	Compare real
FISTP	Integer store and pop	FCOMP	Compare real and pop
Vận chuyển các hệ mười đóng gói (Packed Decimal transfers)		FCOMPP	Compare real and pop twice
FBLD	Packed decimal (BCD) load	FICOM	Integer compare
FBSTP	Packed decimal (BCD) store and pop	FICOMP	Integer compare and pop
Tải dữ liệu (Data transfer)		Hằng số (Constant)	
Cộng (Addition)		FLDZ	Load + 0.0
FADD	Add real	FLD1	Load + 1.0
FADDP	Add real and pop	FLDPI	Load π
FIADD	Integer add	FLDL2T	Load $\log_2 10$
Trừ (Subtraction)		FLDL2E	Load $\log_2 e$
FSUB	Subtract real	FLDLG2	Load $\log_{10} 2$
FSUBP	Subtract real and pop	FLDLN2	Load $\log_e 2$
FISUB	Integer subtract	Điều khiển bộ xử lý (Processor control)	
FSUBR	Subtract real reversed	FINIT/FNINIT	Initialize processor
FISUBR	Integer subtract reversed	FDISI/FNDISI	Disable interrupts
Nhân (Multiplication)		FENI/FNENI	Enable interrupts
FMUL	Multiply real	FLDCW	Load control word
FMULP	Multiply real and pop	FSTCW/FNSTSW	Store status word
FIMUL	Integer multiply	FSTENV/FNSTENV	Store environment
Chia (Division)		FLDENV	Load environment
FDIV	Divide real	FSAVE/FNSAVE	Save state
FDIVP	Divide real and pop	FRSTOR	Restore state
FIDIV	Integer divide	FINCSTP	Increment stack pointer
FDIVR	Divide real reversed	FDECSTP	Decrement stack pointer
FDIVRP	Divide real reversed and pop	FFREE	Free register
FIDIVR	Integer divide reversed	FNOP	No operation
Các phép toán khác (Other Operations)		FWAIT	CPU wait
FSQRT	Square root		
FSCALE	Scale		
FPREM	Partial remainder		
FPREM1	Partial remainder		
FRNDINT	Round to integer		
FXTRACT	Extract exponent and significand		
FABS	Absolute value		
FCHS	Change sign		

Các lệnh của 8087 phân thành các nhóm như sau:

1. Các lệnh vận chuyển các số thực (Real transfers).
2. Các lệnh vận chuyển các số nguyên (Integer transfers).
3. Các lệnh vận chuyển các số hệ mười đóng gói (Packed decimal transfers).
4. Các lệnh cộng (Addition).
5. Các lệnh trừ (Subtraction).
6. Các lệnh nhân (Multiplication).
7. Các lệnh chia (Division).
8. Các lệnh khác (đổi dấu, khai căn bậc hai, làm tròn số, số mũ, lấy giá trị tuyệt đối,...)
9. Các lệnh hàm lượng giác (Transcendental), logarit, số mũ.
10. Các lệnh so sánh (Comparison).
11. Các lệnh với hằng số (Constant).
12. Các lệnh điều khiển bộ xử lý (processor control).

Nhóm lệnh điều khiển bộ xử lý là những lệnh như: khởi tạo ban đầu bộ xử lý, cấm ngắt, cho phép ngắt, nạp từ điều khiển, cất giữ từ điều khiển, xóa các ngoại lệ, nạp và cất giữ môi trường, cất giữ và phục hồi trạng thái, tăng con trỏ ngăn xếp, giảm con trỏ ngăn xếp, làm trống thanh ghi, chờ CPU, và không thực hiện gì cả.

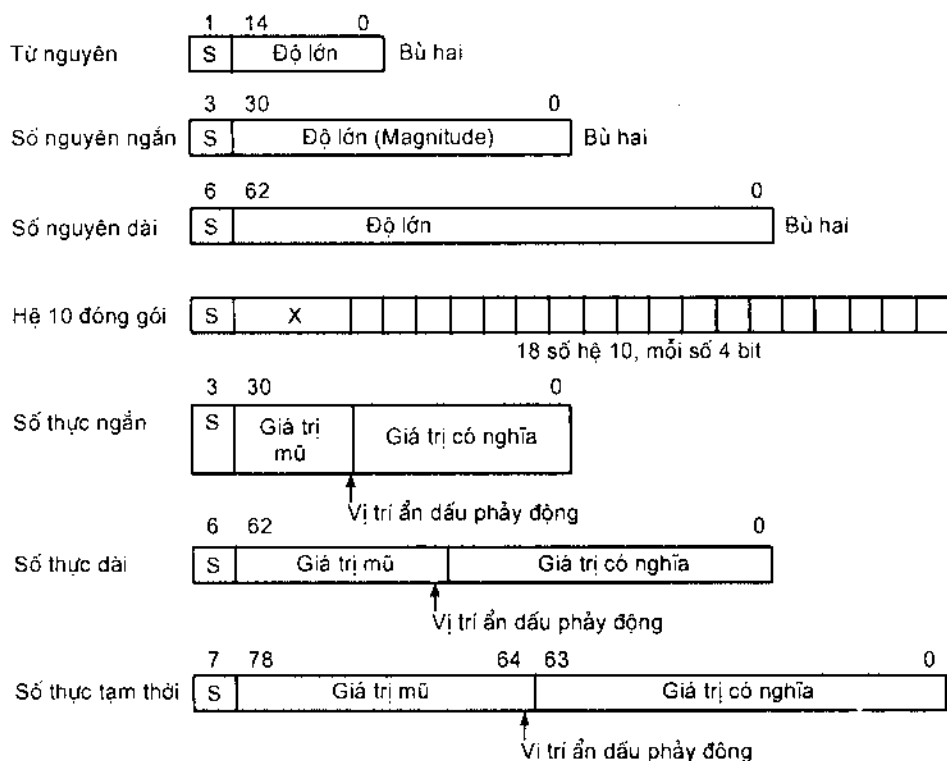
(5) Khuôn dạng dữ liệu của 8087

Khi cất giữ vào trong bộ nhớ chính của hệ thống, dữ liệu có thể ở bất kỳ khuôn dạng nào, nhưng trong 8087, dữ liệu chỉ có thể được xử lý và cất giữ dưới các dạng số qui định cho các tính toán số dấu phẩy động. Khi chuyển từ 8087 trở lại bộ nhớ chính của hệ thống, dữ liệu được tự động chuyển về dạng thích hợp cho xử lý bên trong 8086/8088. Bảng 2.8 phân loại các kiểu dữ liệu và hình 2.87 liệt kê các khuôn dạng dữ liệu được xử lý trong 8087.

Bảng 2.8: Phân loại các kiểu dữ liệu

Kiểu dữ liệu	Bit	Các chữ số hệ mười có nghĩa	Khoảng giá trị (hệ mười)
Từ nguyên	16	4	$-32768 \leq X \leq +32767$
Số nguyên ngắn	32	9	$-2 \times 10^9 \leq X \leq +2 \times 10^9$
Số nguyên dài	64	18	$-9 \times 10^{18} \leq X \leq +9 \times 10^{18}$
Hệ mười đóng gói	80	18	$-99...99 \leq X \leq +99...99$
Số thực ngắn	32	6-7	$8,43 \times 10^{-37} \leq X \leq 3,37 \times 10^{38}$
Số thực dài	64	15-16	$4,19 \times 10^{-307} \leq X \leq 1,67 \times 10^{308}$
Số thực tạm thời	80	19	$3,4 \times 10^{-4932} \leq X \leq 1,2 \times 10^{4932}$

Số thực ngắn có giá trị lớn nhất là 127 (7Fh), số thực dài có giá trị lớn nhất là 1023 (3FFh), và số thực tạm thời có giá trị lớn nhất là 16383 (3FFFh).



Hình 2.88: Các khuôn dạng dữ liệu của 8087

2.4.6. Vi xử lý 80286

(1) Đóng vỏ và sơ đồ khối của 80286

80286 được coi là loạt vi xử lý khởi đầu cho các loạt vi xử lý công nghệ tiến tiến của Intel, gọi chung là X86 (kể cả Pentium). 80286 làm việc ở tần số nhịp 8 MHz có khả năng thực hiện trung bình 1,5 triệu lệnh máy trong 1 s (1,5 MIPS). 80286 có các kiểu đóng vỏ: gồm 68 chân DIP, mảng chân nổi PGA (Pin Grid Array), 68 chân GAP hoặc 68 chân QFP.

80286 là sự nâng cấp hoàn chỉnh của 8086. Nó là vi xử lý 16-bit với địa chỉ 24-bit (A0÷A23) cho phép đánh địa chỉ tới 16 MB nhớ vật lý. Khác với 8086, 80286 có bus dữ liệu riêng (D0-D15) không ghép chung với địa chỉ. Tần số làm việc cũng cao hơn nhiều so với 8086. 80286 có đi kèm đồng xử lý 80287 và chúng là trung tâm của hệ máy vi tính thế hệ IBM PC/AT, IBM PS/2 model 50 rất thông dụng trong những năm 1980. Cấu trúc bên trong của 80286 tương tự như 8086, nhưng đặc biệt nó có đơn vị quản lý bộ nhớ (MMU) bên ngoài hỗ trợ cho cơ chế đa sử dụng, đa nhiệm và địa chỉ ảo trong chế độ bảo vệ. Các hệ điều hành đa nhiệm đã có thể cài đặt trên các máy tính dùng chip 80286. Mô hình lập trình của 80286 giống như của 8086/8088.

Các chế độ đánh địa chỉ của 80286 cũng tương tự như ở 8086. Tất cả các lệnh của 8088/8086 đều chạy được trên 80286.

Sơ đồ khối chức năng bên trong 80286 có sự khác biệt với 8086 (hình 2.89). 80286 có 4 đơn vị xử lý riêng biệt được tổ chức theo kiến trúc đường ống.

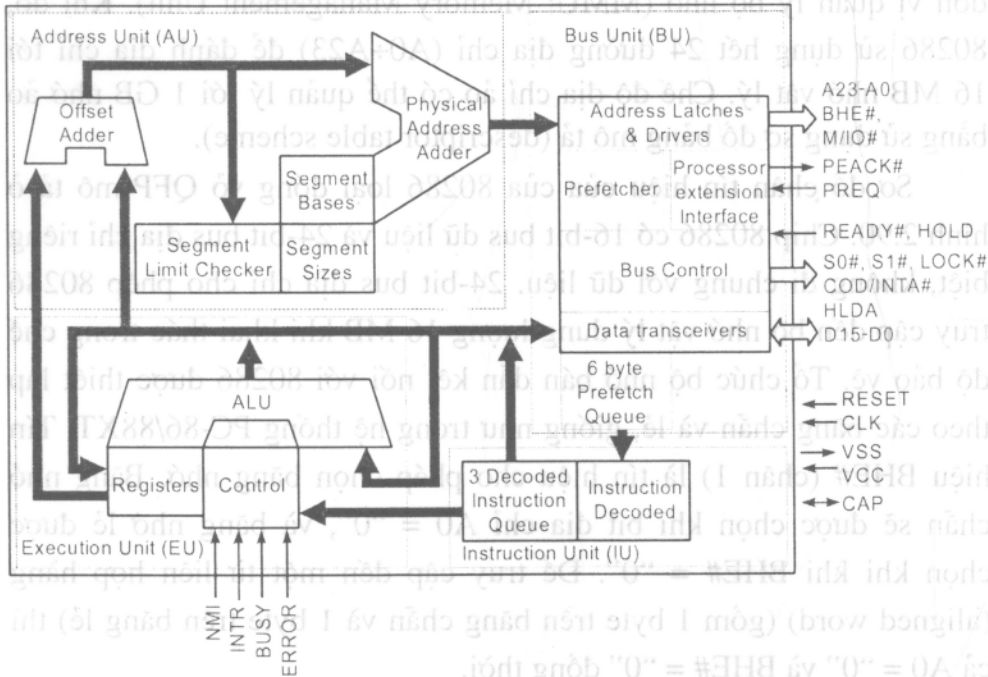
- *Đơn vị bus BU (Bus Unit):*

BU thực hiện tất cả chức năng ghi/đọc với bộ nhớ, tiền đọc (prefetch) các lệnh, và điều khiển vận chuyển dữ liệu từ và đến bộ đồng xử lý toán học 80287.

- *Đơn vị lệnh IU (Instruction unit):*

IU thực hiện giải mã đầy đủ 3 lệnh được đọc trước và giữ chúng trong hàng lệnh. Tại IU đơn vị thực hiện EU (Execution Unit) sẽ truy

cập tới để thực hiện lệnh. Đây là kiểu kiến trúc đường ống lệnh, trong đó, các lệnh được đọc trước vào mà không cần phải có thời gian chờ đợi lệnh đang xử lý thực hiện xong.



Hình 2.89: Sơ đồ khối chức năng bên trong 80286

• **Đơn vị thực hiện EU:**

EU sử dụng 16-bit ALU để thực hiện các lệnh mà nó nhận từ IU. Khi thao tác trong chế độ địa chỉ thực (real address mode), tập thanh ghi của 80286 giống như tập thanh ghi của 8086 ngoại trừ thanh ghi trạng thái (PSW) 16-bit (thanh ghi cờ).

• **Đơn vị địa chỉ AU (Address Unit):**

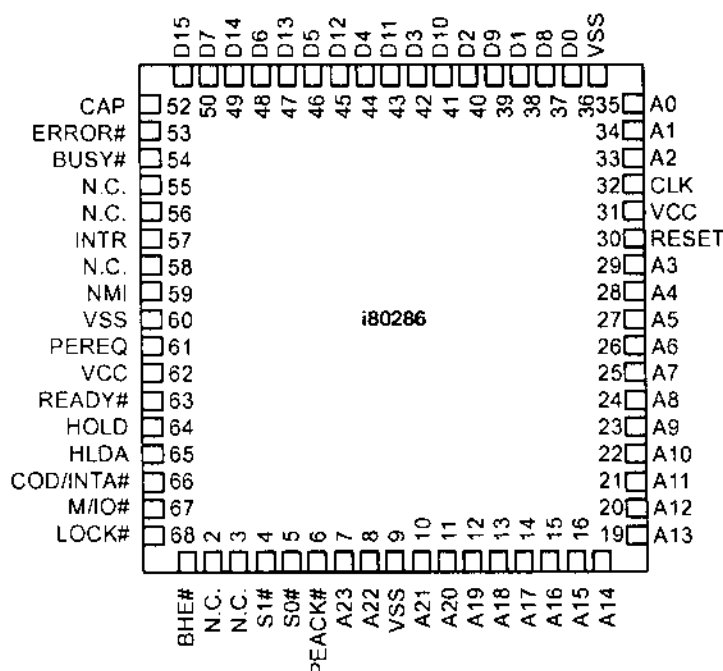
AU tính toán các địa chỉ vật lý có thể được gửi từ BU ra ngoài bộ nhớ hoặc thiết bị I/O. 80286 có thể thao tác ở một trong hai chế độ đánh địa chỉ bộ nhớ: chế độ địa chỉ thực và chế độ địa chỉ ảo bảo vệ (protected virtual address mode). Nếu 80286 ở trong chế độ thực, thì AU tính các địa chỉ bằng cách sử dụng 16 bit địa chỉ cơ sở đoạn chứa trong các thanh ghi đoạn CS, DS, ES, SS và Offset 8 bit và 16-bit giống

như trong 8086. Như vậy, trong chế độ thực 80286 chỉ có khả năng đánh địa chỉ tới 1MB nhớ vật lý bằng 20 bit địa chỉ ($A0 \div A19$) giống như 8086. Trong chế độ địa chỉ ảo bảo vệ, AU có chức năng như một đơn vị quản lý bộ nhớ (MMU: Memory Management Unit). Khi đó, 80286 sử dụng hết 24 đường địa chỉ ($A0 \div A23$) để đánh địa chỉ tới 16 MB nhớ vật lý. Chế độ địa chỉ ảo có thể quản lý tới 1 GB nhớ ảo bằng sử dụng sơ đồ bảng mô tả (descriptor table scheme).

Sơ đồ chân tín hiệu của của 80286 loại đóng vỏ QFP mô tả ở hình 2.90. Chip 80286 có 16-bit bus dữ liệu và 24-bit bus địa chỉ riêng biệt, không đi chung với dữ liệu. 24-bit bus địa chỉ cho phép 80286 truy cập đến bộ nhớ vật lý dung lượng 16 MB khi khai thác trong chế độ bảo vệ. Tổ chức bộ nhớ bán dẫn kết nối với 80286 được thiết lập theo các băng chân và lẻ, giống như trong hệ thống PC-86/88XT. Tín hiệu BHE# (chân 1) là tín hiệu cho phép chọn băng nhớ. Băng nhớ chẵn sẽ được chọn khi bit địa chỉ $A0 = "0"$, và băng nhớ lẻ được chọn khi $BHE# = "0"$. Để truy cập đến một từ liên hợp hàng (aligned word) (gồm 1 byte trên băng chẵn và 1 byte trên băng lẻ) thì cả $A0 = "0"$ và $BHE# = "0"$ đồng thời.

Nhiều tín hiệu của 80286 giống với 8086. Và theo cách điều khiển thì các chức năng của 80286 giống với 8086 ở chế độ tối đa. Các tín hiệu trạng thái $S0\#, S1\#$, và $M/IO\#$ được giải mã bằng mạch điều khiển bus bên ngoài 82288 (ở 8086 giải mã bằng mạch 8288) để tạo ra các tín hiệu điều khiển của bus, đọc, ghi, và các tín hiệu chấp nhận ngắt.

Các tín hiệu HOLD (chân 64), HLDA (pin 65), INTR (chân 57), INTA# (chân 66), NMI (chân 59), READY# (chân 63), LOCK# (chân 68) và RESET (chân 30) có chức năng tương tự như chúng có ở 8086/8088. Mạch tạo nhịp đồng hồ 82284 (cho 8086/8088 là 8284) bên ngoài tạo ra tín hiệu đồng hồ cho 80286 và để đồng bộ các tín hiệu RESET và READY#.



Hình 2.90: Sơ đồ chân tín hiệu của 80286

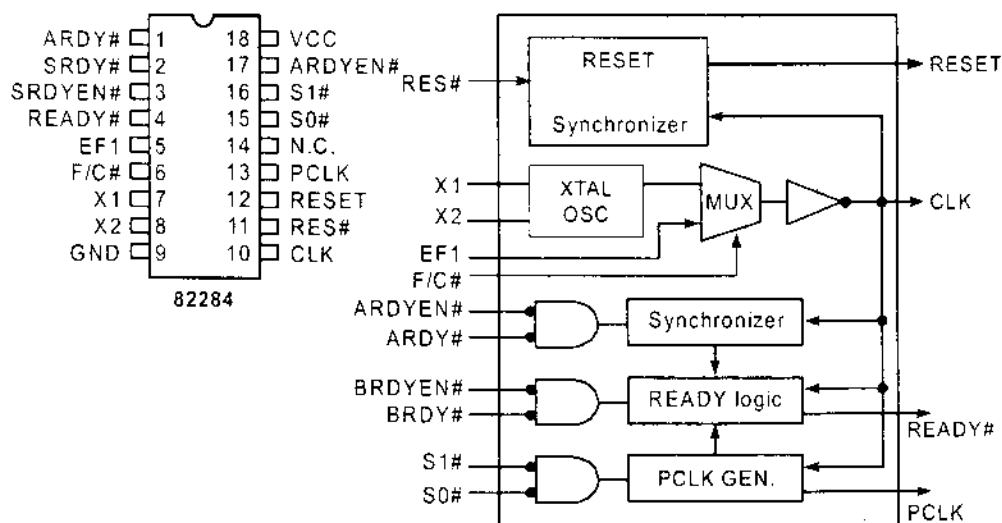
Tín hiệu vào “yêu cầu mở rộng xử lý” PEREQ = “1” (Processor Extension Acknowledge) (chân 61) được đưa từ bộ đồng xử lý toán học 80287 đến để yêu cầu 80286 thực hiện cuộc chuyển dữ liệu đến hoặc từ bộ nhớ cho 80287. Khi 80286 bắt đầu thực hiện yêu cầu này, nó gửi tín hiệu “chấp nhận mở rộng xử lý” PEACK# = “0” (Processor Extension Acknowledge) (chân 6) đến 80287 để thông báo rằng cuộc chuyển dữ liệu theo yêu cầu của 80287 đã bắt đầu. Các cuộc chuyển dữ liệu của 80286 được thực hiện sao cho 80287 sử dụng được khả năng quản lý bộ nhớ bảo vệ và ảo của MMU trong chip 80286.

Tín hiệu vào BUSY# (pin 54) của 80286 có chức năng giống như TEST# của 8086. Khi 80286 thực hiện một lệnh chờ (WAIT), nó duy trì vòng lặp WAIT cho đến khi nó nhận được tín hiệu BUSY# từ 80287 chuyển lên mức “cao”. Nếu 80287 phát hiện ra lỗi trong quá trình xử lý lệnh, nó sẽ gửi tín hiệu ERROR# = “0” tới 80286, và 80286 sẽ tự động thực hiện một cuộc gọi (call) xử lý ngắt số hiệu 16h.

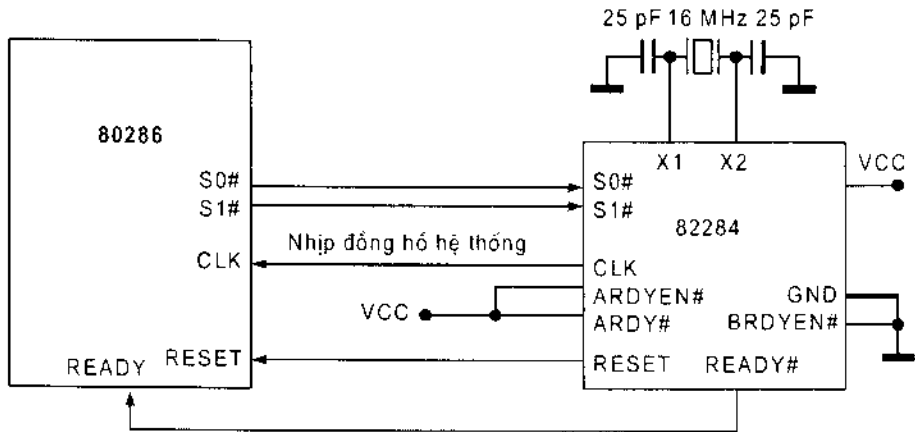
Mô hình lập trình chung của 80286 đã được mô tả trong mô hình lập trình chung của các loại vi xử lý của Intel. Trong đó, có sự giống nhau của 8086/8088 với 80286. Điều này có nghĩa là các chương trình viết cho 80286 ở chế độ thực (chế độ 8086) có thể chạy được trên các hệ thống i808X. Nhưng ở chế độ bảo vệ sự sử dụng các thanh ghi đoạn và các thanh ghi chung để đánh địa chỉ có phức tạp hơn và chúng sẽ được xét cụ thể sau.

(2) Nhịp đồng hồ hệ thống của 80286

Tương tự như 8086/8088, 80286 sử dụng chip tạo đồng hồ hệ thống ở bên ngoài. Nhưng do tần số nhịp của 80286 cao hơn, nên nó phải dùng tới chip 82284 (hình 2.91). Chip này tạo nhịp 16 MHz từ dao động thạch anh 16 MHz. Nhịp đồng hồ (CLK) này được 80286 chia thành 2 để các vi lệnh (microinstructions) thực hiện ở tốc độ 8 MHz. 80286 có các phiên bản cho 6 MHz, 8 MHz, 10 MHz, và 12.5 MHz của 80286. Nhưng tần số càng cao thì chi phí cho các chip bên ngoài 80286 càng đắt. Phương án 8 MHz được chấp nhận hơn cả. Việc đấu nối 80286 với 82284 được mô tả trong hình 2.92.



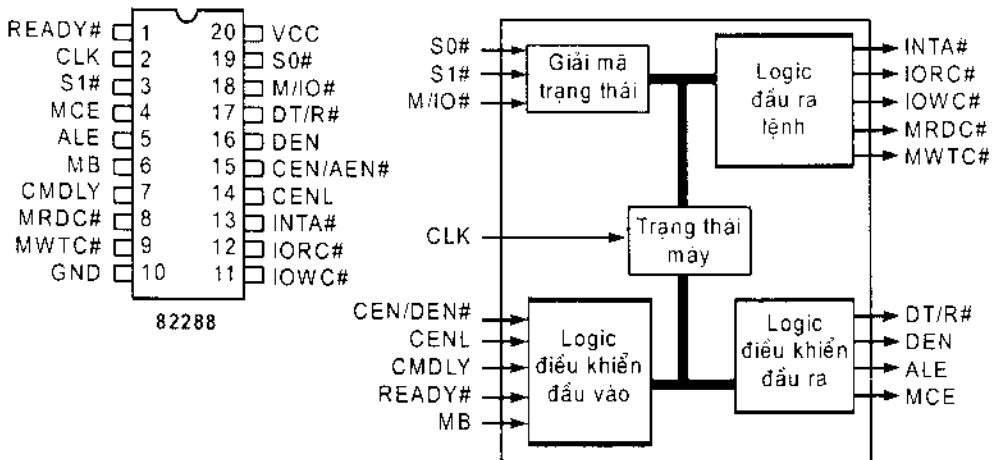
Hình 2.91: Chip tạo nhịp đồng hồ 82284



Hình 2.92: Kết nối 80286 và 82284

(3) Điều khiển bus của 80286

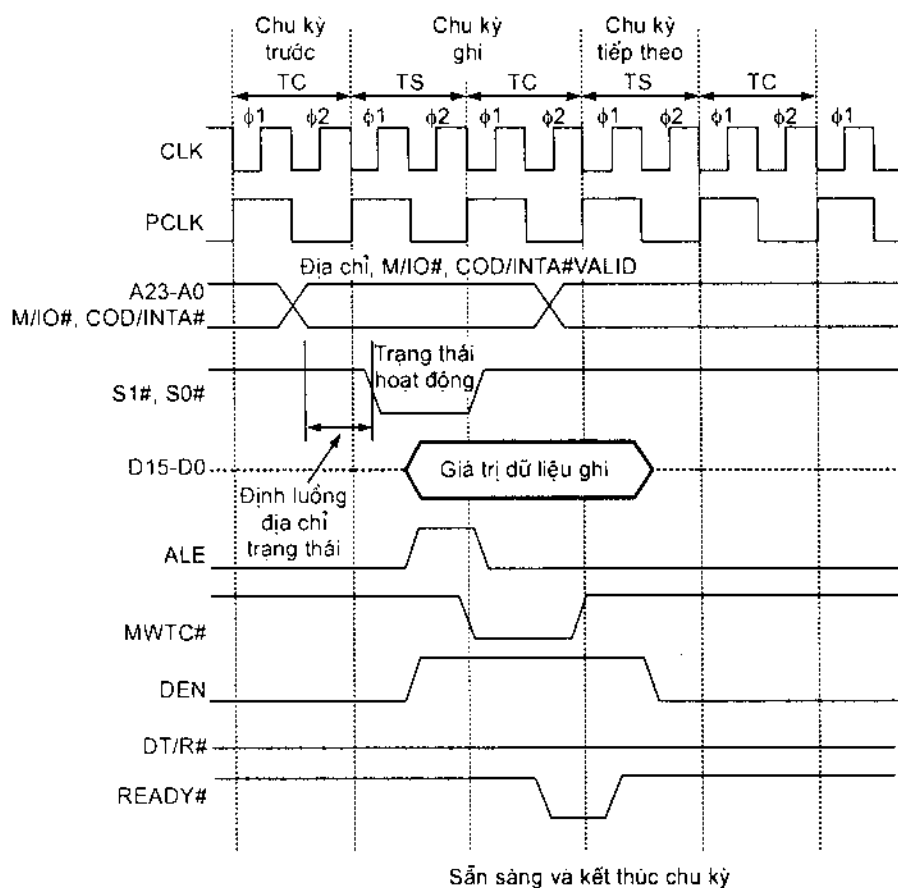
8086/8088 sử dụng chip điều khiển bus 8288 để tạo ra các tín hiệu điều khiển cho bus điều khiển từ các tín hiệu trạng thái S0#, S1#, S2# và IO/M#. Cũng tương tự, 80286 dùng chip điều khiển bus 82288 (hình 2.93) tạo ra các tín hiệu điều khiển (DT/R#, DEN, ALE, MCE) và các xung lệnh (INTA#, IORC#, IOWC#, MRDC#, MWTC#) từ các tín hiệu trạng thái S0#, S1#, M/IO# và xung nhịp CLK (16MHz) từ 82284.



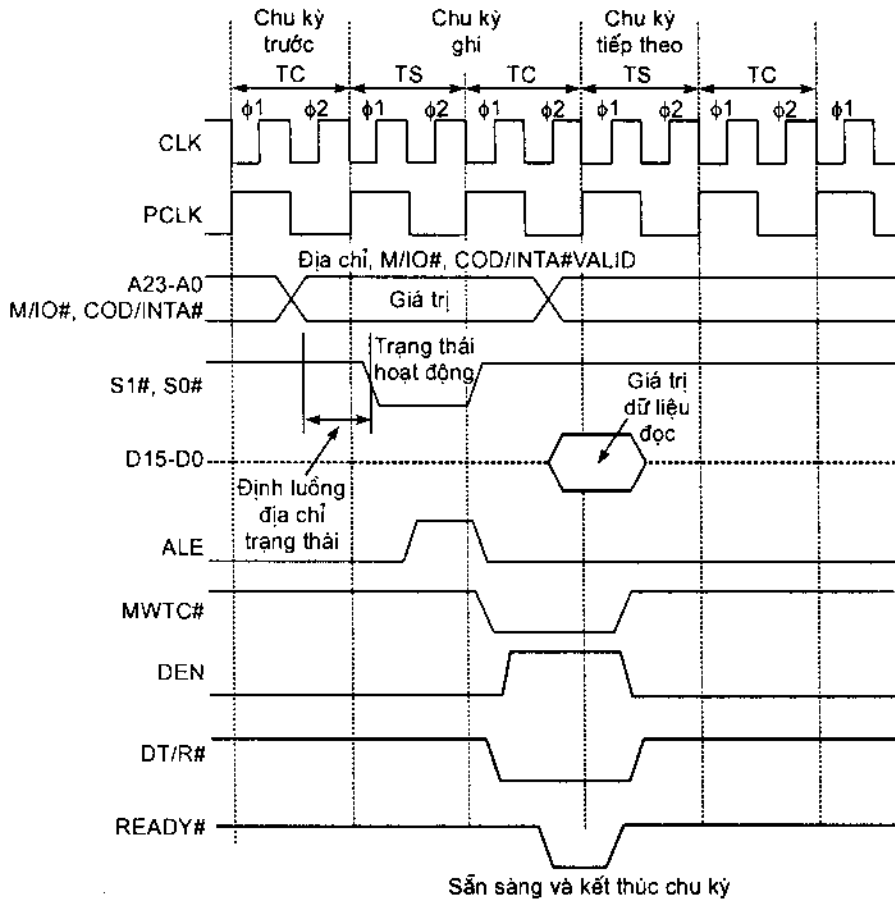
Hình 2.93: Chip điều khiển bus 82288

(4) Chu kỳ bus của 80286

80286 điều khiển các phần cứng của hệ thống máy tính với sự khởi đầu bằng các chu kỳ bus. Mỗi một chu kỳ bus thường gồm các thao tác ghi/đọc. Một chu kỳ bus gồm tối thiểu 2 chu kỳ nhịp. Khi sử dụng nhịp 8 MHz, 80286 thực hiện từ 3 đến 3,5 triệu chu kỳ bus trong một giây. Đồ thị xung chu kỳ bus của 80286 khi thực hiện ghi ra ngoài bus mô tả trong hình 2.94, và khi thực hiện đọc từ bus mô tả trong hình 2.95.



Hình 2.94: Đồ thị xung chu kỳ bus ghi của 80286



Hình 2.95: Đồ thị xung chu kỳ bus đọc của 80286

Bất kỳ một chu kỳ bus nào đều bắt đầu với việc đưa ra địa chỉ lên các đường dây địa chỉ. Do có tiền đọc lệnh vào bên trong hàng lệnh nên ta có địa chỉ được xử lý theo đường ống (hàng các lệnh) và địa chỉ xuất hiện tiếp ngay sau thời điểm kết thúc của chu kỳ lệnh trước. Nếu chu kỳ là chu kỳ lệnh ghi thì ngay trong quãng thời gian địa chỉ và các tín hiệu M/I/O#, COD/INTA#, S1#, S0# giá trị dữ liệu đã có hiệu lực được ghi lên bus. Tín hiệu chốt địa chỉ ALE luôn luôn có xung tích cực ở vào thời điểm của sườn sau của xung nhịp CLK (ngay sau khi 82288 xác nhận rằng trạng thái của 80286 đang tích cực). Tín hiệu ALE chốt địa chỉ đường ống và giữ địa chỉ tồn tại trong suốt chu kỳ.

Các tín hiệu DEN và DT/R# cho phép và điều khiển hướng đi của dòng dữ liệu. Tín hiệu vào READY của 80286 cho phép kéo dài bất kỳ chu kỳ bus nào có thể lâu bao nhiêu nếu cần thiết, ví dụ như trong trường hợp kết nối với các thiết bị I/O và bộ nhớ có tốc độ truy cập chậm. Chu kỳ bus tối thiểu không có thời gian kéo dài chỉ gồm 2 chu kỳ nhịp 8 MHz (PCLK).

Bảng 2.9: Trạng thái chu kỳ bus của 80286

COD/INTA#	M/IO#	S1#	S0#	Chu kỳ bus được khởi tạo
0 (LOW)	0	0	0	Chấp nhận ngắt
0	0	0	1	Sẽ không xuất hiện
0	0	1	0	Sẽ không xuất hiện
0	0	1	1	Không phải chu kỳ trạng thái
0	1	0	0	Nếu A1=1 thì dừng, nếu không thì tắt máy (shutdown)
0	1	0	1	Đọc dữ liệu từ bộ nhớ
0	1	1	0	Ghi dữ liệu ra bộ nhớ
0	1	1	1	Không phải chu kỳ trạng thái
1 (HIGH)	0	0	0	Sẽ không xuất hiện
1	0	0	1	Đọc từ thiết bị I/O
1	0	1	0	Ghi ra thiết bị I/O
1	0	1	1	Không phải chu kỳ trạng thái
1	1	0	0	Sẽ không xuất hiện
1	1	0	1	Đọc lệnh từ bộ nhớ
1	1	1	0	Sẽ không xuất hiện
1	1	1	1	Không phải chu kỳ trạng thái

Các tín hiệu trạng thái M/IO#, S1#, S0# và COD/INTA# xác định trạng thái của chu kỳ bus trong bảng 2.9. Chu kỳ bus của 80286 cũng như của các chip vi xử lý khác gồm có 5 trường hợp:

- + Chu kỳ đọc bộ nhớ
- + Chu kỳ ghi bộ nhớ
- + Đọc từ thiết bị I/O
- + Ghi ra thiết bị I/O.

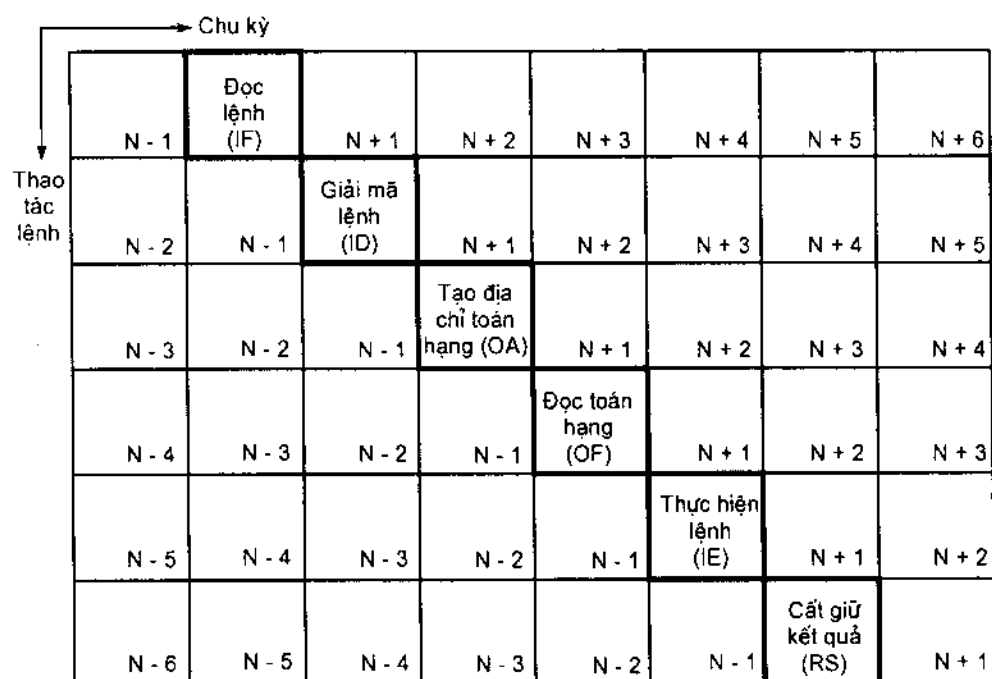
Bất kỳ chu kỳ bus nào cũng tác động đến 82288 tạo ra các tín hiệu lệnh, đó là:

- + Đọc bộ nhớ: MRDC# (chân 8)
- + Ghi bộ nhớ: MWTC# (chân 9)
- + Đọc từ thiết bị ngoại vi: IORC# (chân 12)
- + Ghi ra thiết bị ngoại vi: IOWC# (chân 11)
- + Chấp nhận ngắt: INTA (chân 13).

Đối với EPROM có thời gian truy cập đặc trưng 200 ns thì chu kỳ đọc từ các bộ nhớ này của 80286/8 MHz không cần đến trạng thái chờ đợi. Những chu kỳ đọc các bộ nhớ này gồm khoảng 75% của toàn bộ các chu kỳ bus thực hiện nên các trạng thái chờ đợi trung bình cho từng chu kỳ bus gần bằng 0. Đối với RAM có thời gian truy cập đặc trưng 150 ns cần phải có một trạng thái chờ đợi bổ sung cho chu kỳ bus. Các mạch điều khiển của các thiết bị I/O thường yêu cầu kéo dài thời gian các xung ở mức tích cực ở các chân tín hiệu CS (chọn chip), RD và WR: ghi đọc dữ liệu. Do đó chu kỳ bus để đọc/ghi dữ liệu với các thiết bị I/O đòi hỏi phải bổ sung ít nhất 3 trạng thái chờ đợi.

(5) Kỹ thuật đường ống (pipelining) của 80286

Trong sơ đồ khối chức năng 80286 có 4 đơn vị tổ chức theo kỹ thuật đường ống: IU, EU, AU và BU. Để nâng cao tốc độ thực hiện lệnh, sự liên kết thực hiện lệnh của những đơn vị này được tổ chức thành chuỗi liên tiếp (đường ống), gọi là đường ống lệnh (instruction pipeline) (hình 2.96). Sự thực hiện liên tục các lệnh trong đường ống lệnh nhờ đầu vào ống lệnh là hàng lệnh IQ đọc trước các lệnh và lần lượt đưa vào giải mã trong khi đường ống lệnh đang thực hiện lệnh trước đó đã đưa vào ống lệnh. Tại đầu vào của ống lệnh, lệnh từ IQ được đưa vào, và đồng thời ở phía đầu ra của ống lệnh kết quả được đưa đi cất giữ. Sự liên tục thực hiện này giống như dây chuyền lắp ráp trong công nghiệp, mà trên đó, ở mọi điểm đều có thao tác lắp ráp, và cùng lúc, phía đầu vào dây chuyền có bán thành phẩm nhập vào, còn phía đầu ra đã có sản phẩm hoàn chỉnh.

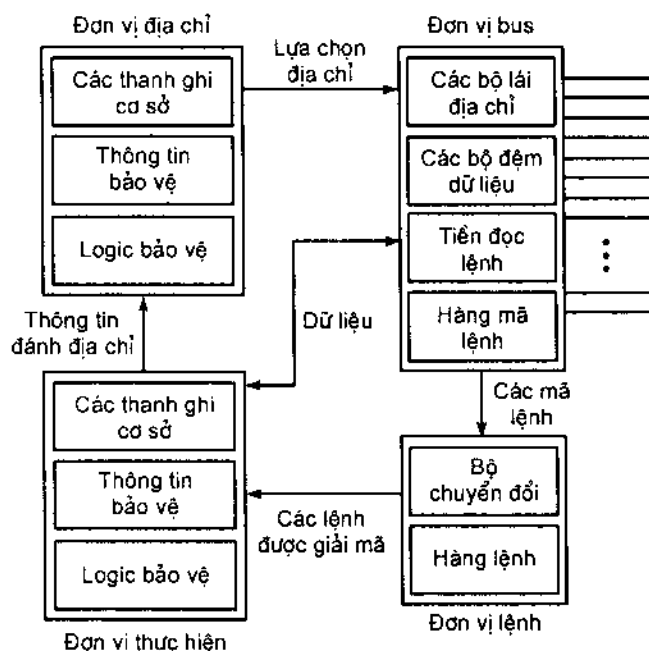


Hình 2.96: Đường ống lệnh bên trong 80286

Thao tác của bốn đơn vị xử lý bên trong 80286 theo đường ống lệnh được mô tả trong hình 2.97. Nhờ kỹ thuật đường ống lệnh được đưa vào cho 80286 mà tốc độ thực hiện các lệnh máy tăng lên rất nhiều, tăng gấp 3,5 lần so với 8086/8088. Bảng 2.10 là sự so sánh hiệu suất thực hiện một số lệnh máy của các chip 8086/8088 và 80286.

Bảng 2.10: So sánh hiệu suất thực hiện lệnh của các chip 8086/8088 và 80286

Lệnh máy	Số nhịp đồng hồ		
	8088	8086	80286
MOV AX, [BX + DI]	23	19	5
JMP [BX + DI]	26	26	7
ADD [BX + DI], CX	31	22	5
SUB [BX + DI], CX	31	23	5
MUL AX, [BX + DI]	144	140	24
DIV AX, [BX + DI]	171	167	25



Hình 2.97: Bốn đơn vị xử lý bên trong 80286
thao tác các lệnh theo đường ống

(6) Các thanh ghi của 80286

Vì 80286 có thể làm việc ở 2 chế độ: chế độ địa chỉ thực, hay còn gọi là chế độ 8086 và chế độ địa chỉ bảo vệ và ảo, nên sự tham gia của các thanh ghi của 80286 trong các chế độ làm việc có khác nhau. Vì vậy nên xét riêng trong mục lớn về các thanh ghi của 80286.

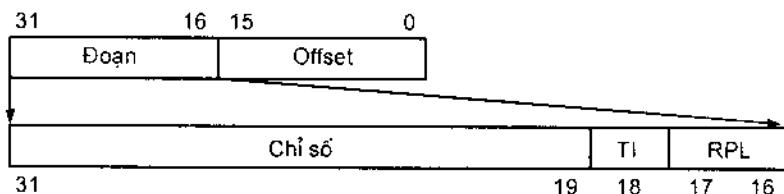
• Các thanh ghi chung, con trỏ lệnh, và thanh ghi cờ của 80286

Trong mô hình lập trình, tất cả 8 thanh ghi chung: AX, BX, CX, DX, SI, DI, BP, SP, con trỏ lệnh IP, và thanh ghi cờ (Flags) của 80286 giống 8086/8088, chúng đều có độ rộng 16 bit. Các thanh ghi chung có thể thao tác với các dữ liệu theo byte cao (H), byte thấp (L) và từ (W), hay để chứa địa chỉ. Các thanh ghi: BX, SI, DI, BP, và SP thường dùng để chứa Offset của ngăn nhớ. Thanh ghi DX chứa địa chỉ cổng I/O cho các lệnh IN N và OUT N, trong đó N là số hiệu cổng I/O.

- Các thanh ghi đoạn của 80286 và các bảng mô tả đoạn

Các thanh ghi đoạn dùng để đánh địa chỉ ngăn nhớ. Trong chế độ thực (chế độ 8086), giống như 8086/8088, các thanh ghi đoạn: CS, SS, DS, và ES được dùng để chứa địa chỉ cơ sở đoạn 16-bit. Khi đó, 20 bit địa chỉ vật lý của ngăn nhớ được hình thành bởi địa chỉ cơ sở đoạn dịch trái 4-bit cộng với 16-bit Offset (vị trí tương đối của ngăn nhớ trong đoạn). Không gian địa chỉ vật lý quản lý là 1 MB.

Trong chế độ bảo vệ, nhờ có MMU, không gian địa chỉ được quản lý lên tới 1 GB. Khi đó, 32-bit địa chỉ ảo của ngăn nhớ được xác định bằng cặp Selector: Offset (hình 2.98), gọi là con trỏ địa chỉ (address pointer). Giá trị chọn 16-bit (Selector) được nạp vào một trong các thanh ghi đoạn CS, SS, DS, và ES. Đó là địa chỉ trỏ tới một vùng nhớ gọi là bảng mô tả đoạn SDT (Segment Descriptor Table). Dữ liệu trong SDT được sử dụng thay cho địa chỉ cơ sở đoạn (giống như trong 8086/8088) để tạo địa chỉ bộ nhớ. Con trỏ địa chỉ 32-bit được biểu diễn dưới dạng hệ đếm thập lục phân (Hex), ví dụ: 04AE:0015, FF00:FE00, hoặc 0000:0000.

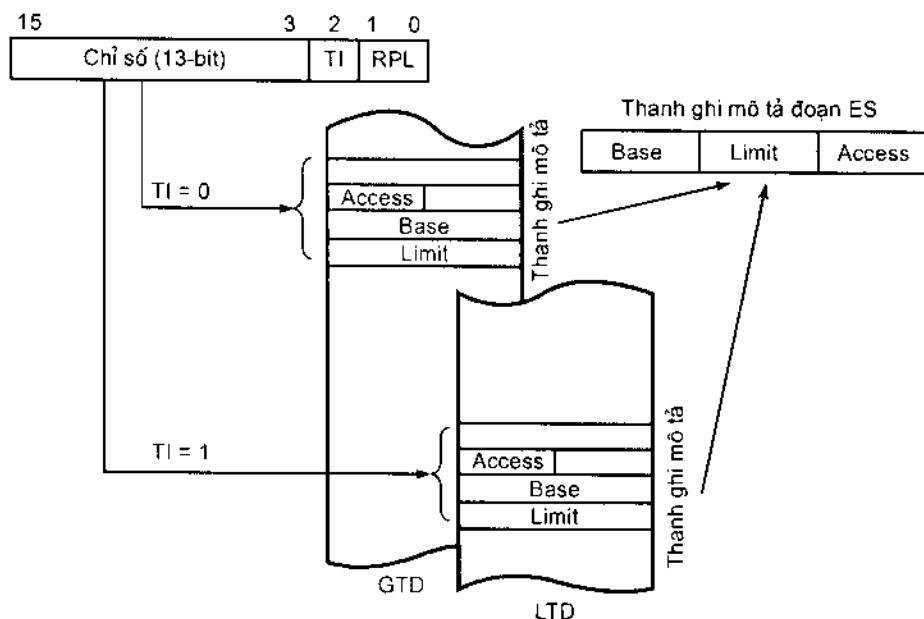


Hình 2.98: 32-bit địa chỉ ảo là con trỏ địa chỉ

Selector gồm có các giá trị: chỉ số (Index) (bit 31-bit 19), chỉ thị bảng TI (Table Indicator bit 18), và RPL (bit 17 - bit 16). Trường Index và TI xác định một mô tả (Descriptor) (hình 2.99).

Khi TI = 0, nó xác định một thanh ghi mô tả (Descriptor) trong bảng mô tả toàn cục GDT (Global Descriptor Table). Khi bit TI = 1, nó xác định một thanh ghi mô tả trong bảng mô tả cục bộ LDT (Local Descriptor Table). 13-bit INDEX xác định một mô tả đoạn trong một bảng mô tả (GDT hay LDT). Mỗi một đoạn có một mô tả đoạn liên

quan. Giá trị Index = 0 trở tới mô tả (0), Index = 1 trở tới mô tả (1),... Mỗi một mô tả đoạn có độ dài 8 byte (hình 2.100). Các bảng mô tả đoạn là những mảng nhớ có dung lượng thay đổi. Chúng có thể có dung lượng tối thiểu là 8 byte (nếu chỉ có một mô tả) cho đến 64 kbyte (giới hạn trên là: $2^{13} = 8192$ mô tả, mỗi mô tả 8 byte).



Hình 2.99: Index chọn thanh ghi mô tả trong GDT hay LDT

Địa chỉ cơ sở đoạn A24 - A31					Byte 7
G	16 hay 32 bit 32	0	bit sử dụng	Giới hạn đoạn D16-D19	Byte 6
P	Mức thẩm quyền của mô tả đoạn	Kiểu đoạn	Mô tả kiểu		Byte 5
Địa chỉ cơ sở đoạn A16-A23					Byte 4
Địa chỉ cơ sở đoạn A8-A15					Byte 3
Địa chỉ cơ sở đoạn A0-A7					Byte 2
Giới hạn đoạn D8-D15					Byte 1
Giới hạn đoạn D0-D7					Byte 0

Trong đó: G = Kích thước đoạn là 1 MB hay 4 GB

Nếu byte 6 và 7 bằng 0 thì mô tả đoạn dành cho 80286. Byte 5 chứa các thông tin về: Loại mô tả (Type of descriptor): toàn cục (Global), cục bộ (Local); Loại đoạn; mức thẩm quyền của mô tả (descriptor privilege level), và P: có đoạn trong bộ nhớ hay không.

Hình 2.100: Cấu trúc của một mô tả đoạn

Bảng mô tả toàn cục GDT trỏ đến thông tin dùng cho tất cả các chương trình (thường đó là phần mềm hệ điều hành). Bảng mô tả cục bộ LDT trỏ đến thông tin chỉ dùng cho một chương trình cụ thể nào đó đang đánh địa chỉ (thường đó là cho người sử dụng cụ thể hoặc một nhiệm vụ cụ thể). Các bảng mô tả GDT và LDT có thể nằm ở bất kỳ đâu trong vùng nhớ vật lý chính của hệ thống máy tính. Các byte này cung cấp địa chỉ cơ sở đoạn 24-bit (80286) hoặc 32-bit (từ 80386 đến Pentium) và các thông tin bổ sung: hai byte đầu tiên (byte 0 và byte 1) cộng với 4 bit ở byte 6 (D16-D19) là giá trị dung lượng giới hạn của đoạn (segment limit) (tức là một đoạn có bao nhiêu byte nhớ). Thông tin này mạch bảo vi xử lý biết được địa chỉ ranh giới của bộ nhớ. Nếu mỗi một mô tả lại mô tả một đoạn nhớ có dung lượng tối đa 64 kbyte (2^{16}) thì cả 2 bảng mô tả thiết lập một không gian nhớ ảo dung lượng tối đa là 1 tỷ byte = $(2 \times 2^{13}) \times 2^{16} = 2^{30} = 1 \text{ GB}$.

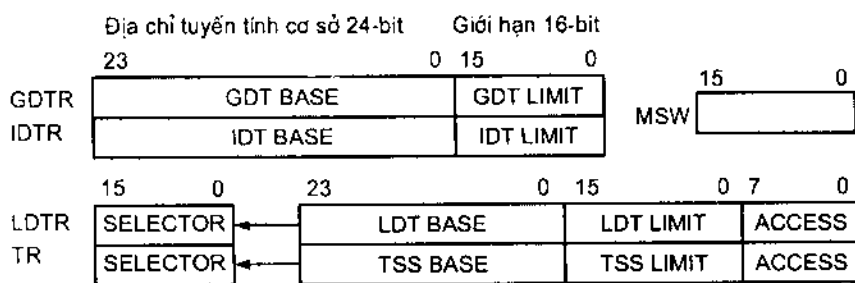
• *Các thanh ghi hệ thống của 80286*

Có 5 thanh ghi hệ thống, đó là:

- GDTR (Global Descriptor Table Register): thanh ghi bảng mô tả toàn cục.
- IDTR (Interrupt Descriptor Table Register): thanh ghi bảng mô tả ngắt.
- MSW (Machine Status Word): từ trạng thái của máy.
- TR (Task Register): thanh ghi của nhiệm vụ.
- LDTR (Local Descriptor Table Register): thanh ghi bảng mô tả cục bộ.

Khi viết các chương trình chạy ở mức hệ thống, trong đó có các truy cập tới các thanh ghi của hệ thống, để cho phép chế độ bảo vệ, trước tiên cần phải khởi tạo 2 thanh ghi hệ thống GDTR và IDTR. Sau đó chế độ bảo vệ được cho phép nhờ khởi tạo thanh ghi PSW. Khi đã ở trong chế độ bảo vệ, 80286 sẽ thực hiện một nhiệm vụ nào đó (task). Nếu muốn 80286 thực hiện đồng thời một số nhiệm vụ (chế độ đa

nhiệm) thì cần phải khởi tạo 2 thanh ghi cuối cùng: TR và LDTR tương ứng với các thanh ghi lựa chọn của đoạn nơi chứa nhiệm vụ. Đến đây quá trình khởi tạo các giá trị ban đầu cho 5 thanh ghi hệ thống mới kết thúc để đảm bảo 80286 làm việc trong chế độ bảo vệ địa chỉ ảo và đa nhiệm. Sự can thiệp ở mức hệ thống có thể cho phép người lập trình hệ thống thay đổi nội dung của các thanh ghi TR và LDTR. Bốn thanh ghi GDTR, IDTR, LDTR, và TR là những thanh ghi quản lý bộ nhớ (hình 2.101):



Hình 2.101: Các thanh ghi hệ thống lập trình trong chế độ bảo vệ

• GDTR:

Thanh ghi của bảng mô tả toàn cục có độ dài 40-bit, nó chứa thông tin về vị trí trong bộ nhớ vật lý của bảng mô tả toàn cục GDT. GDT có tác dụng như là một bảng sắp xếp bộ nhớ trong chế độ bảo vệ mô tả một khoảng nhớ hoạt động cố định hay toàn cục của toàn bộ không gian nhớ ảo. Bởi vì GDTR phải được nạp giá trị ngay trước khi chuyển vào chế độ bảo vệ nên GDTR phải được can thiệp trong chế độ thực và chế độ bảo vệ.

GDTR gồm có 2 trường: trường GDT BASE là 24-bit địa chỉ cơ sở của bảng GDT trong bộ nhớ vật lý. Trường GDT LIMIT là giá trị Offset lớn nhất (ranh giới) bảng GDT. Byte đầu tiên của GDT có giá trị Offset = 0, do đó, giá trị giới hạn tính bằng byte là GDT LIMIT + 1. Bởi vì trường GDT LIMIT có 16 bit nên giá trị Offset lớn nhất là 0FFFFh hay 65535 ($2^{16} = 64$ kbyte). Như vậy GDT có dung lượng là 64 kbyte. Vì mỗi thanh ghi mô tả có độ dài 8 byte (2^3), nên GDT có 8.182 thanh ghi mô tả (2^{13}). Vì mỗi một thanh ghi mô tả mô tả một

đoạn nhớ dung lượng 64 kbyte nên GDT mô tả một vùng nhớ hoạt động cố định hay toàn cục dung lượng bằng 0,5 tỷ byte. Khi đã khởi tạo GDTR, về nguyên tắc, không thể thay đổi nội dung của GDTR, vì GDT mà GDTR tham chiếu tới là bảng phân bố của một vùng nhớ hoạt động cố định hay toàn cục trong chế độ bảo vệ. Sự chuyển đổi nhiệm vụ (task switching) cũng không làm thay đổi GDTR.

• **IDTR:**

Thanh ghi của bảng mô tả ngắt chứa thông tin về vị trí của bảng mô tả ngắt IDT trong bộ nhớ vật lý. Bảng IDT hoạt động cố định trong chế độ bảo vệ để trở cho CPU tới thực hiện các chương trình xử lý ngắt hoặc xử lý các sự kiện xảy ra không mong muốn. Bởi vì IDTR được khởi tạo ngay trước khi chuyển vào chế độ bảo vệ nên nó phải được can thiệp trong chế độ thực và chế độ bảo vệ. IDTR cũng có 2 trường tương tự như GDTR, trường IDT BASE là giá trị 24-bit địa chỉ cơ sở của bảng IDT, trường IDT LIMIT là giá trị Offset lớn nhất (ranh giới) của bảng IDT. Khi đã khởi động IDTR, nó không được thay đổi nữa, vì IDT mà nó trỏ tới là một bảng thường trú trong bộ nhớ mà CPU sử dụng trong chế độ bảo vệ. Sự chuyển đổi nhiệm vụ cũng không làm thay đổi nội dung IDTR.

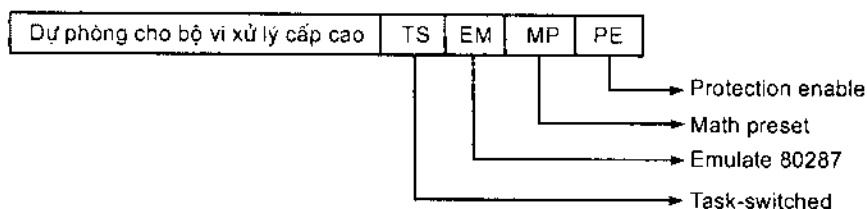
Vì 80286 hỗ trợ 256 véc-tơ ngắt nên phải có 256 cửa vào (gate) trong bảng IDT. Vì mỗi một cửa chiếm 8 byte trong IDT, nên IDT chỉ cần chiếm 2047 byte = $(256 \times 8) - 1$. Như thế với 16 bit của IDT LIMIT sẽ hoang phí lớn dung lượng nhớ. Bảng ngắt như dung lượng có thể được nạp bằng các lệnh với mảng dữ liệu 6 byte từ bộ nhớ.

24-bit địa chỉ cơ sở của GDT và IDT được nạp vào các trường GDT BASE và IDT BASE của các thanh ghi tương ứng GDTR và IDTR là những địa chỉ vật lý sử dụng trong chế độ bảo vệ. Những địa chỉ khác dùng trong chế độ bảo vệ là những địa chỉ ảo được biểu diễn bởi cặp thanh ghi lựa chọn: Offset. Đó là những địa chỉ độ dài 32-bit, hay còn gọi là con trỏ địa chỉ. Giá trị lựa chọn được nạp vào thanh ghi đoạn.

• **MSW:**

Từ trạng thái máy chứa thông tin về trạng thái và cấu hình của 80286 (hình 2.102). Nó được sử dụng để thiết lập sự cho phép chế độ bảo vệ hoạt động. Vì vậy nó phải được truy cập cả ở chế độ thực và chế độ bảo vệ. 80286 có lệnh nạp từ trạng thái máy LMSW.

MSW là thanh ghi 16-bit, nhưng chỉ 4 bit thấp nhất là có ý nghĩa, những bit còn lại không dùng tới và dự phòng cho 80286. Bốn bit thấp của MSW được sử dụng cho những mục đích quan trọng. Ba bit trong đó là các bit điều khiển (PE, MP, EM) và được thiết lập nhờ phần mềm ở mức hệ thống khi khởi tạo hệ thống ở chế độ bảo vệ. Bit thứ tư (TS) là bit trạng thái được CPU tự động thiết lập ở thời gian nào đó.



Hình 2.102: Từ trạng thái máy của 80286

Bit PE cho phép chế độ bảo vệ khi nó được thiết lập = "1" và ở trạng thái 1 cho đến khi nào xóa (RESET) chip 80286. Khi xóa chip 80286, MSW có giá trị = 0. Trước khi thiết lập PE = 1, phải khởi tạo GDTR và IDTR sao cho các thanh ghi này tham chiếu tới các bảng tương ứng GDT và IDT. Để có khả năng tương thích với các bộ vi xử lý trong tương lai (80386, 80486) và cho kỹ thuật lập trình được thuận tiện, ta phải thay đổi chỉ những bit nào của MSW cần đến.

Các bit MP và EM liên quan đến sự có mặt của bộ đồng xử lý toán học 80287. Khi có 80287 thì thiết lập MP = 1, khi không có 80287 thì thiết lập EM = 1. Đừng bao giờ thiết lập MP và EM cùng thời gian. Trường hợp cùng thiết lập MP và EM = 1 chỉ cho chế độ thử của nhà sản xuất.

Khi MP hoặc EM được thiết lập = 1, 80286 trong chế độ bảo vệ biết được các lệnh của đồng xử lý có thể được thực hiện nhờ 80287 hay nhờ phần mềm mô phỏng đồng xử lý. Chúng ta có thể sử dụng các bit này ở cả trong chế độ thực và bảo vệ. Nếu chạy trong chế độ thực (chế độ 8086) thì MSW không thay đổi và không có ý nghĩa gì cả vì 8086 không có MSW. Trong trường hợp này, MSW có giá trị ngầm định = 0 sau khi xóa chip 80286, và 80286 hiểu ngầm định là có 80287 như ở hệ thống 8086 thực hiện.

Bit TS, chỉ thị chuyển đổi nhiệm vụ, là bit trạng thái liên quan đến sự chuyển đổi nhiệm vụ trong chế độ bảo vệ và bộ đồng xử lý toán học. Bit TS được tự động thiết lập ngay khi 80286 thực hiện một sự chuyển đổi nhiệm vụ. Nếu có lệnh phải thực hiện nhờ bộ đồng xử lý 80287 ở trong nhiệm vụ mới trong khi TS = 1 thì một bẫy (TRAP) được tự động sinh ra và một chương trình con xử lý bẫy này sẽ được thực hiện. Nó phải cất giữ trạng thái các thanh ghi của 80287 của nhiệm vụ trước đó đang thực hiện, và nạp mới trạng thái của nhiệm vụ mới được chuyển (nhiệm vụ hiện hành), sau đó xóa bit TS nhờ lệnh CLTS. Khi TS = 0, bộ đồng xử lý 80287 sẽ không bẫy nữa (ngắt mềm).

• **TR:**

Một khi chế độ bảo vệ được thiết lập (PE = 1 của MSW) thì 80286 làm cho nội dung của TR chứa thông tin về đoạn trạng thái của nhiệm vụ TSS (Task State Segment) hiện hành. Đó là một đoạn trong bộ nhớ vật lý dùng để chứa các nội dung của các thanh ghi của nhiệm vụ hiện hành nếu 80286 được chỉ định phải chuyển đổi sang một nhiệm vụ mới. Hình 2.101 mô tả TR và phần nhớ cache chứa mô tả mà TR tham chiếu tới.

TR chỉ được can thiệp trong chế độ bảo vệ, bởi vì chế độ đa nhiệm tự động và sự chuyển đổi nhiệm vụ chỉ thực hiện trong chế độ bảo vệ. TR phải được khởi tạo bằng giá trị lựa chọn trong chế độ bảo vệ, sau đó TR được thay đổi tự động. TR được nạp lại giá trị một cách tự động mỗi khi CPU thực hiện một chuyển đổi nhiệm vụ trong chế độ

bảo vệ. Nó phải được nạp lại để chọn một đoạn trạng thái cho nhiệm vụ mới chuyển qua, vì mỗi một nhiệm vụ có trạng thái các thanh ghi và một đoạn trạng thái nhiệm vụ (TSS) duy nhất cho nó. Vì TR chứa giá trị lựa chọn tham chiếu đến vùng nhớ cache chứa TSS nên nó cũng tương tự như thanh ghi đoạn. Khi lựa chọn được nạp vào TR, bộ nhớ cache mô tả liên quan với TR được tự động nạp giá trị mô tả mà TR tham chiếu đến. Mô tả là thực thể chứa thông tin trở tới TSS.

- **LDTR:**

LDTR chứa nội dung là giá trị thanh ghi lựa chọn trở tới bảng mô tả cục bộ LDT cho một nhiệm vụ hiện hành. Tương tự như TR, LDTR chỉ được truy cập trong chế độ bảo vệ. LDTR được khởi tạo một giá trị lựa chọn ban đầu cho một nhiệm vụ, sau đó, nó được tự động nạp giá trị lựa chọn mới mỗi khi có sự chuyển đổi nhiệm vụ mới. Mỗi một nhiệm vụ có một LDT riêng. Khi LDTR được nạp một giá trị lựa chọn thì LDT được tham chiếu tới.

Tương tự như GDT, LDT có 8.192 thanh ghi mô tả, mỗi thanh ghi mô tả có độ dài 8 byte, và LDT có dung lượng 64 kbyte. Mỗi thanh ghi mô tả trong LDT lại trở tới một đoạn dung lượng 64 kbyte, do đó LDT quản lý một không gian nhớ ảo dung lượng 0,5 GB, giống như GDT. Gộp cả GDT và LDT, chúng quản lý toàn bộ không gian bộ nhớ ảo là 1 GB.

Có các lệnh LGDT, LLDT, và LIDT nạp (load) địa chỉ cơ sở 24 bit (Base) và giá trị giới hạn 16 bit (Limit) của các bảng GDT, LDT và IDT vào các thanh ghi tương ứng GDTR, LDTR, và IDTR. Có các lệnh SGDT, SLDT, và SIDT cất giữ (store) các nội dung của các thanh ghi tương ứng GDTR, LDTR, và IDTR vào vùng địa chỉ đích xác định.

(7) Tập lệnh

Là loại vi xử lý đầu tiên của thế hệ vi xử lý công nghệ cao của Intel 80X86 và Pentium với 2 chế độ làm việc: chế độ thực, hay còn gọi là chế độ 8086 và chế độ địa chỉ ảo, hay còn gọi là chế độ bảo vệ.

Chúng hỗ trợ cho kỹ thuật đa xử lý (multiprocessing) và đa nhiệm (multitasking). Chính bắt đầu trên các hệ thống máy tính 80286 các hệ điều hành đa nhiệm kiểu UNIX như SCO UNIX, XENIX, có thể cài đặt. Vì vậy, mặc dù về tổ chức các thanh ghi, 80286 tương tự như 8086, nhưng nó có thêm các thanh ghi hệ thống và đơn vị quản lý bộ nhớ MMU bên ngoài để đảm bảo quản lý hệ thống nhiều dung lượng hơn với cơ chế bảo vệ ở các mức thẩm quyền khác nhau và tập lệnh cũng có sự mở rộng thêm cho các chế độ làm việc khác nhau. Tập lệnh của 80286 là sự mở rộng của tập lệnh 8086, do đó hầu hết các lệnh của 80286 là các lệnh của 8086 và các chương trình viết trên 80286 chạy ở chế độ thực đều chạy được trên hệ thống 8086. So với 8086 tập lệnh của 80286 được bổ sung thêm một số lệnh như sau:

- *Các lệnh thực hiện ở cả 2 chế độ thực (8086) và bảo vệ*

INS - (Input String): nhận từ ngoại vi chuỗi dữ liệu.

OUTS - (Output String): đưa ra ngoại vi chuỗi dữ liệu.

PUSHA: cất (đẩy) vào ngăn xếp nội dung của tất cả 8 thanh ghi chung.

POPA: phục hồi nội dung của tất cả 8 thanh ghi chung từ ngăn xếp.

PUSH immediate: cất vào ngăn xếp một giá trị ngay lập tức.

SHIFT/ROTATE destination, immediate: dịch hoặc quay vòng nội dung thanh ghi đích hoặc ngăn nhớ đi một số bit xác định.

IMUL destination, immediate: nhân có dấu với một giá trị ngay lập tức.

IMUL destination, multiplicand, immediate multiplier: nhân có dấu, kết quả chuyển vào đích xác định.

ENTER: thiết lập một khung của ngăn xếp trong thủ tục, cất giữ BP, trở BP tới TOS, và phân bố không gian ngăn xếp cho các biến cục bộ.

LEAVE: lệnh này làm mất tác dụng của lệnh ENTER trước khi có lệnh RET trong thủ tục.

BOUND - (Array bounds Check): kiểm tra ranh giới mảng.

LMSW: nạp từ trạng thái máy, được sử dụng để chuyển 80286 từ chế độ thực sang chế độ bảo vệ.

• *Các lệnh thực hiện trong chế độ bảo vệ*

CTS: xóa bit TS trong MSW.

LGDT: nạp giá trị vào thanh ghi GDTR từ bộ nhớ.

SGDT: cất giữ các nội dung của thanh ghi GDTR vào bộ nhớ.

LIDT: nạp giá trị vào thanh ghi IDTR từ bộ nhớ.

LLDT: nạp giá trị thanh ghi lựa chọn và thanh ghi mô tả tương ứng vào thanh ghi LDTR.

SLDT: cất giữ thanh ghi lựa chọn từ LDTR vào thanh ghi hoặc bộ nhớ.

LTR: nạp thanh ghi TR giá trị thanh ghi lựa chọn và thanh ghi mô tả cho TSS.

STR: cất giữ thanh ghi lựa chọn từ TR vào thanh ghi hoặc bộ nhớ.

LMSW: nạp thanh ghi PSW từ thanh ghi hoặc bộ nhớ.

SMSW: cất giữ nội dung PSW vào thanh ghi hoặc bộ nhớ.

LAR: nạp byte quyền truy cập của thanh ghi mô tả vào thanh ghi hoặc bộ nhớ.

LSL: nạp giá trị giới hạn đoạn (segment limit) từ thanh ghi mô tả vào thanh ghi hoặc bộ nhớ.

ARPL: thay đổi mức thẩm quyền của thanh ghi lựa chọn về mức thấp hơn.

DERR: xác định xem có xảy ra sự kiện thanh ghi lựa chọn trở tới đoạn bị cấm.

DERW: xác định xem có xảy ra sự kiện thanh ghi lựa chọn trở tới đoạn có thể bị ghi lại.

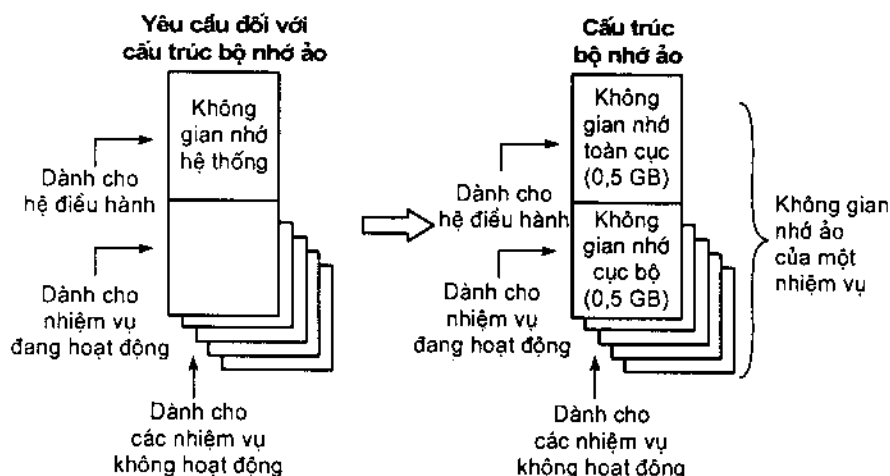
(8) Quản lý bộ nhớ của 80286

80286 là chip vi xử lý đầu tiên của thế hệ X86 được thương mại hoá có đơn vị quản lý bộ nhớ (MMU) bên trong chip. Nhờ đó, không gian nhớ ảo có thể lên đến 1 GB, hay 1 tỷ byte. Nếu sử dụng chip nhớ 64 kbit thì phải cần tới 131.072 chip nhớ này. Đó là điều không thể, CPU với MMU giải quyết vấn đề không gian nhớ lớn này bằng cách làm cho mỗi một nhiệm vụ (một chương trình) tin rằng nó có quyền truy cập tới không gian địa chỉ dành cho nó tối đa 1GB, bao gồm bộ nhớ chính là RAM và thiết bị nhớ ngoài (đĩa cứng). Như vậy, một phần chương trình và dữ liệu của nó có thể chứa trên đĩa cứng và sẽ được đọc vào RAM khi nào cần thực hiện.

Đơn vị quản lý bộ nhớ (MMU) nằm bên trong chip đảm bảo hiệu suất cao cho phần cứng bởi vì các hoạt động của quản lý bộ nhớ được thực hiện theo kiến trúc đường ống cùng với các thao tác khác của CPU, và các chip nhớ bên ngoài có thể sử dụng toàn bộ thời gian truy cập của chu kỳ bus mà không cần phải mất thời gian giành cho MMU bên ngoài (nếu sử dụng MMU bên ngoài). MMU thao tác với các bảng mô tả GDT, LDT, và IDT nằm ở trong bộ nhớ chính. Cấu trúc hệ thống nhớ của hệ 80286 hỗ trợ cho các hệ điều hành đa nhiệm (Multitasking). Bản chất của hệ thống nhớ này là toàn bộ không gian nhớ ảo được phân chia thành không gian nhớ toàn cục của hệ thống (global memory system space) và không gian nhớ cục bộ (hay không gian gian của từng nhiệm vụ). Không gian nhớ toàn cục cũng như không gian nhớ cục bộ chiếm tối đa 0,5 tỷ byte (0,5 GB) toàn bộ không gian nhớ ảo (xem GDTR, và LDTR). Có nhiều vùng nhớ cho từng nhiệm vụ (không gian nhớ cục bộ), nhưng trong từng thời điểm, chỉ có một nhiệm vụ tích cực (hình 2.103). 80286 quản lý bộ nhớ thông qua LGTR và GDTR.

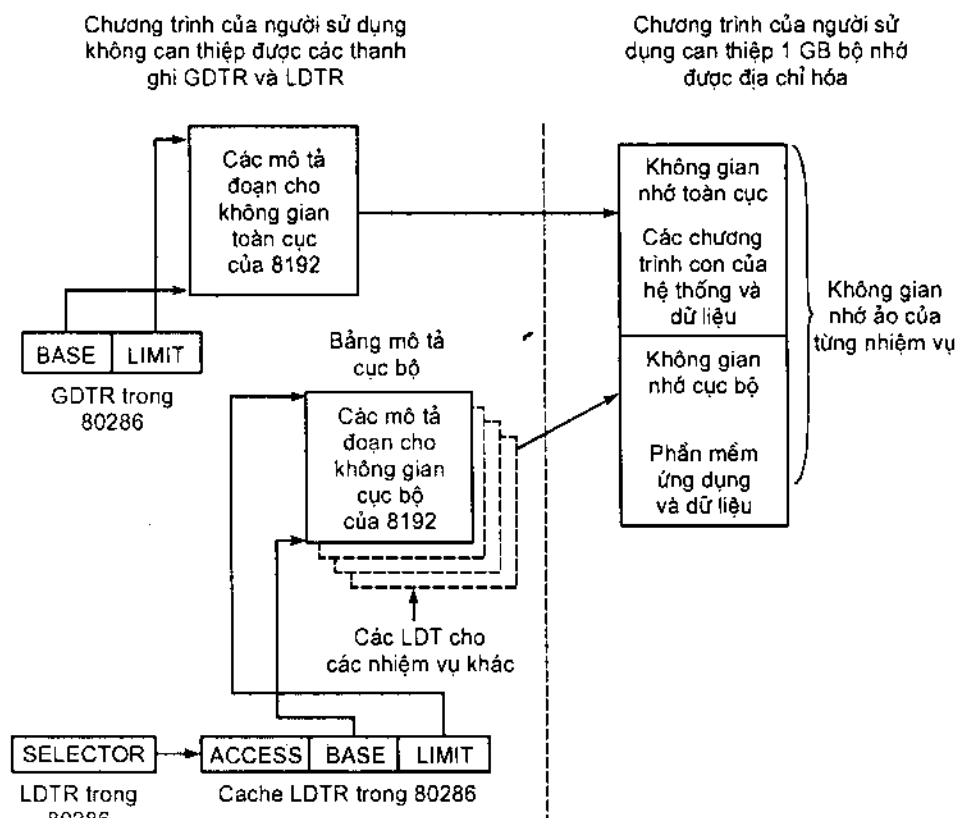
Để xác định các bảng mô tả tương ứng với các không gian nhớ toàn cục và cục bộ đối với từng nhiệm vụ trong chế độ làm việc đa nhiệm, 80286 có các thanh ghi mà ta đã nói tới, đó là: GDTR và

LDTR. Những thanh ghi này được hệ thống khởi tạo giá trị (người sử dụng không can thiệp được). Khi đã được khởi tạo, GDTR thỉnh thoảng được thay đổi vì hầu hết các mô tả liên quan đến hệ điều hành và các chương trình con xử lý ngắt không cần phải thay đổi. Nhưng LDTR thì được tự động nạp lại nhờ CPU khi thực hiện chuyển đổi nhiệm vụ thực hiện, vì mỗi nhiệm vụ lại có một LDT riêng, trong đó xác định các đoạn nhớ cục bộ riêng chứa chương trình và dữ liệu của nhiệm vụ. Hình 2.104 giải thích 80286 quản lý các vùng nhớ thông qua LGTR và GDTR như thế nào.



Hình 2.103: Cấu trúc không gian nhớ ảo cho đa nhiệm của 80286

Người sử dụng không thể can thiệp vào các thanh ghi GDTR và LDTR nhưng có thể nạp các giá trị thanh ghi lựa chọn vào các thanh ghi đoạn tương ứng, và mong đợi quyền truy cập vào các đoạn nhớ mong muốn nhờ sự hỗ trợ không gian ảo của 80286. Bởi vì giá trị thanh ghi lựa chọn là 16-bit cao của địa chỉ ảo 32-bit. 16-bit thấp của địa chỉ ảo là độ dịch (Offset). Thanh ghi lựa chọn là chỉ số (Index) trỏ tới bảng mô tả toàn cục (GDT) hay LDT riêng của từng nhiệm vụ, tùy thuộc vào giá trị của bit TI.



Hình 2.104: 80286 hỗ trợ không gian nhớ ảo 1 GB

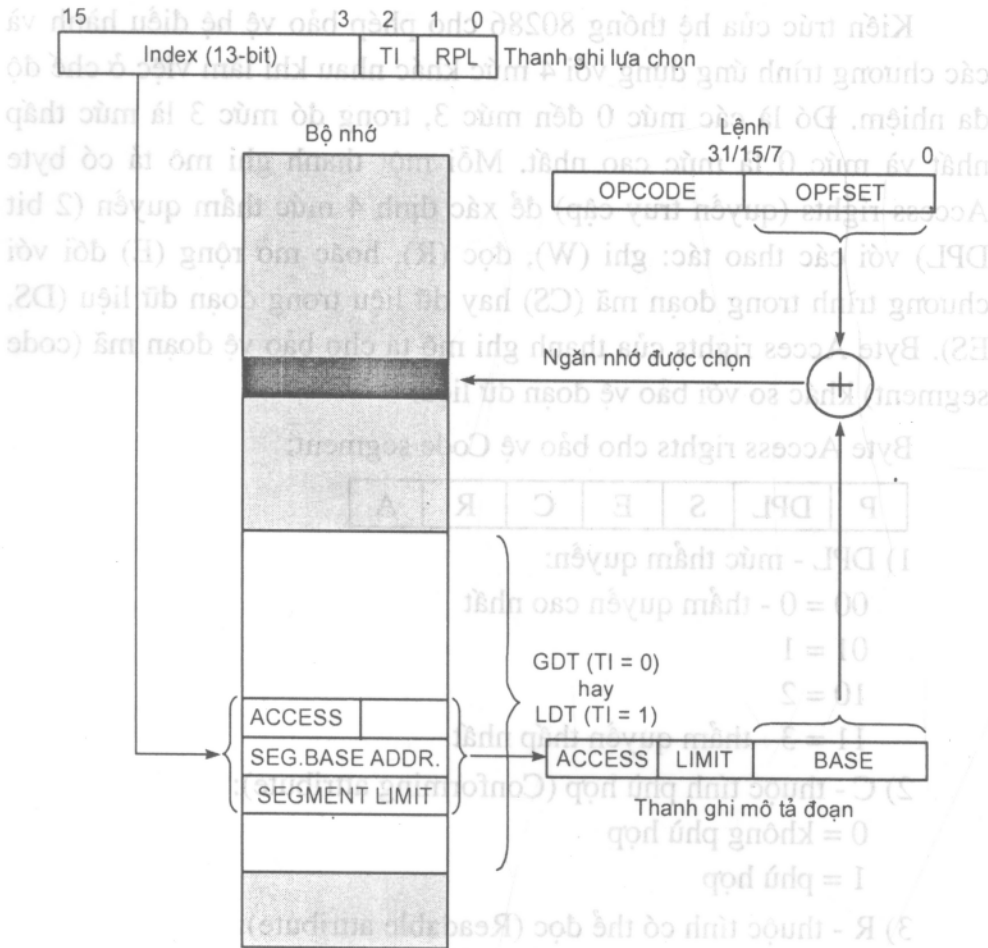
Khi một ngăn nhớ được đánh địa chỉ trong chế độ bảo vệ, thì các nội dung của thanh ghi đoạn trở đến một bảng mô tả tương ứng. Một thanh ghi đoạn ứng với một thanh ghi mô tả đoạn (hình 2.105). Phần dữ liệu của mô tả đoạn gồm: Access (truy cập), Base (cơ sở) và Limit (giới hạn) được nạp vào thanh ghi mô tả đoạn trong MMU. Nội dung của thanh ghi mô tả đoạn tương ứng cộng với phần địa chỉ logic (Offset) trong từ lệnh tạo ra địa chỉ vật lý của ngăn nhớ (hình 2.106).

Các thanh ghi đoạn

Các thanh ghi mô tả đoạn

Đoạn mã	←	Địa chỉ cơ sở đoạn	Giới hạn đoạn	Quyền truy cập
Đoạn ngăn xếp	←	Địa chỉ cơ sở đoạn	Giới hạn đoạn	Quyền truy cập
Đoạn dữ liệu	←	Địa chỉ cơ sở đoạn	Giới hạn đoạn	Quyền truy cập
Đoạn bổ sung	←	Địa chỉ cơ sở đoạn	Giới hạn đoạn	Quyền truy cập

Hình 2.105: Sự tương ứng của các thanh ghi đoạn với các thanh ghi mô tả đoạn



Hình 2.106: Sự tạo địa chỉ vật lý của một ngăn nhớ

Ví dụ: lệnh:

MOV ES, AX; nạp giá trị thanh ghi lựa chọn vào thanh ghi đoạn ES trong chế độ bảo vệ.

Với việc thực hiện lệnh này, CPU trở tới một bộ mô tả trong bảng mô tả (GDT hoặc LDT) tùy thuộc vào bit TI. Nội dung: BASE, LIMIT và Access của thanh ghi mô tả chọn được đọc ra thanh ghi mô tả đoạn ES (hình 2.106). Các giá trị này giúp cho CPU ghi/đọc với vùng nhớ trong đoạn ES.

Kiến trúc của hệ thống 80286 cho phép bảo vệ hệ điều hành và các chương trình ứng dụng với 4 mức khác nhau khi làm việc ở chế độ đa nhiệm. Đó là các mức 0 đến mức 3, trong đó mức 3 là mức thấp nhất và mức 0 là mức cao nhất. Mỗi một thanh ghi mô tả có byte Access rights (quyền truy cập) để xác định 4 mức thẩm quyền (2 bit DPL) với các thao tác: ghi (W), đọc (R), hoặc mở rộng (E) đối với chương trình trong đoạn mã (CS) hay dữ liệu trong đoạn dữ liệu (DS, ES). Byte Access rights của thanh ghi mô tả cho bảo vệ đoạn mã (code segment) khác so với bảo vệ đoạn dữ liệu:

Byte Access rights cho bảo vệ Code segment:

P	DPL	S	E	C	R	A
---	-----	---	---	---	---	---

1) DPL - mức thẩm quyền:

00 = 0 - thẩm quyền cao nhất

01 = 1

10 = 2

11 = 3 - thẩm quyền thấp nhất

2) C - thuộc tính phù hợp (Conforming attribute):

0 = không phù hợp

1 = phù hợp

3) R - thuộc tính có thể đọc (Readable attribute):

0 = không thể đọc được

1 = có thể đọc được

4) S - Segment bit:

0 = không phải đoạn mã hay dữ liệu

1 = là đoạn mã hay đoạn dữ liệu (S luôn = 1)

5) E - Executable bit:

0 = là đoạn không có thể thực hiện được (đối với đoạn dữ liệu E luôn = 0)

1 = là đoạn chương trình có thể thực hiện được (đối với đoạn mã E = 1).

Đối với đoạn dữ liệu:

Byte Access rights cho bảo vệ Data segment:

P	DPL	1	0	ED	W	A
---	-----	---	---	----	---	---

1) DPL - mức thẩm quyền:

00 = 0 - thẩm quyền cao nhất

01 = 1

10 = 2

11 = 3 - thẩm quyền thấp nhất

2) ED - thuộc tính hướng mở rộng (Expansion direction attribute):

0 = mở rộng lên phía trên

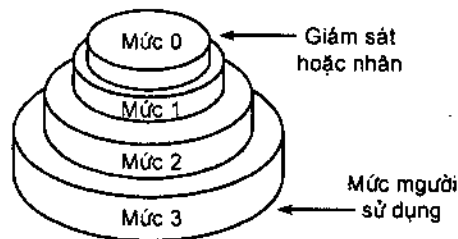
1 = mở rộng về phía dưới

3) W - thuộc tính có thể ghi (Writable attribute):

0 = không thể ghi được

1 = có thể ghi được.

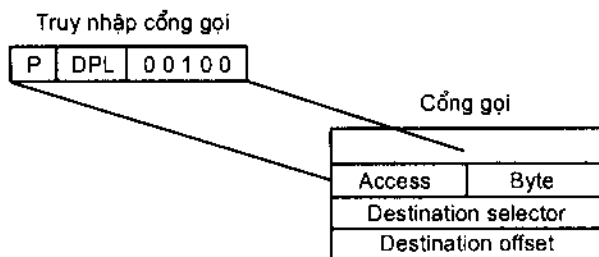
Quy định mức thẩm quyền cao nhất (mức 0) dành cho giám sát (supervisor) và nhân (kernel) của hệ điều hành, mức thẩm quyền thấp nhất (mức 3) dành cho người sử dụng (user level) (hình 2.107).



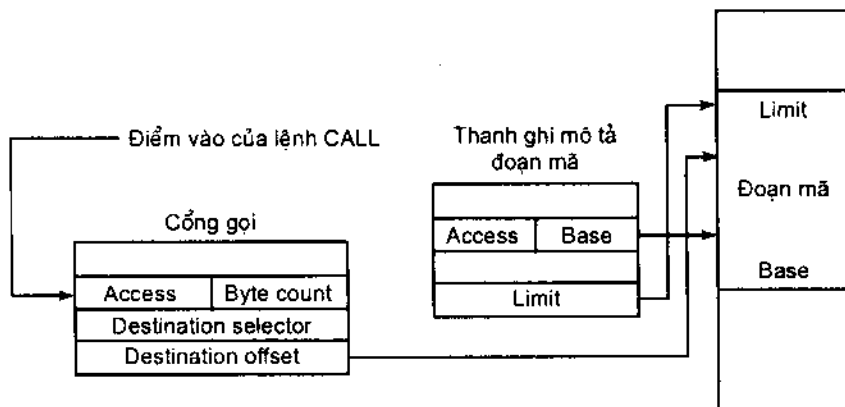
Hình 2.107: Các mức thẩm quyền trong hệ thống

Các truy cập chỉ có thể thực hiện ở cùng mức hoặc thấp hơn thẩm quyền. Một chương trình đang thực hiện ở mức 3 chỉ có thể truy cập tới đoạn dữ liệu ở mức 3. Một lời gọi (CALL) chỉ có thể tới được một đoạn chương trình con ở đồng mức thẩm quyền, nhưng cũng có thể gián tiếp mức cao hơn nếu gọi tới một cổng gọi (call gate). Cổng gọi

có độ dài 8 byte, giống như một thanh ghi mô tả, nó nằm trong bảng mô tả và có khuôn dạng như mô tả trong hình 2.108.



Hình 2.108: Khuôn dạng của cổng gọi



Hình 2.109: Cơ chế chuyển địa chỉ của cổng gọi

Có sự khác nhau cơ bản giữa thanh ghi mô tả và cổng gọi là: nội dung của thanh ghi mô tả trỏ tới một đoạn nhớ nào đó, trong khi đó cổng gọi lại trỏ tới một thanh ghi mô tả. Một thanh ghi mô tả có địa chỉ cơ sở (Base) 24 bit của đoạn nhớ, trong khi đó cổng gọi chứa một địa chỉ ảo 32-bit của đích tới (16-bit Destination Selector và 16-bit Offset). Như vậy, thanh ghi mô tả là một cơ chế sắp xếp bộ nhớ, còn cổng gọi lại là một cơ chế chuyển tiếp địa chỉ.

Khi một địa chỉ hiệu dụng trong câu lệnh gọi CALL trỏ tới một cổng gọi, thì cơ chế cổng gọi cho phép CPU tự động chuyển điều khiển tới địa chỉ đích xác định trong cổng gọi (hình 2.109).

Byte đếm (Count byte) của cổng gọi dùng để sao chép tự động tới 31 byte các thông số của chương trình con từ ngăn xếp hiện hành đến ngăn

xếp đích có mức thẩm quyền cao hơn khi có lệnh CALL. Sự sao chép tự động này của lệnh CALL được thực hiện nếu giá trị byte đếm $\neq 0$.

Xử lý đa nhiệm, chuyển đổi nhiệm vụ, các mức thẩm quyền vào ra IOPL (Input-Output Privilege Level), và các chế độ bảo vệ của 80286 sau này được các hệ thống 80386, 80486 và Pentium phát triển và hoàn thiện hơn. Do đó chúng ta sẽ xét thêm những vấn đề này khi xét tới các chip 80386, 80486 và Pentium.

80286 sử dụng các vi mạch lập trình hỗ trợ đã dùng cho 8085/8086/8088 như 8253/8254 Timer, 8258A PIC, 8237A DMA controller, 8255 PPI, 8250 UART và các vi mạch thanh ghi đệm cho bus dữ liệu và bus địa chỉ như 74LS373, 74LS393, 74LS244, 74LS245, 8282, 8283... Tuy nhiên, phải sử dụng vi mạch tạo nhịp đồng hồ là chip 82284, và mạch điều khiển bus 82282 chuyên dùng cho 80286. 80286 là CPU của PC-AT286 với các phiên bản nhịp đồng hồ 8MHz/10MHz/12MHz/16MHz/20MHz thông dụng trong những năm 1989 - 1990.

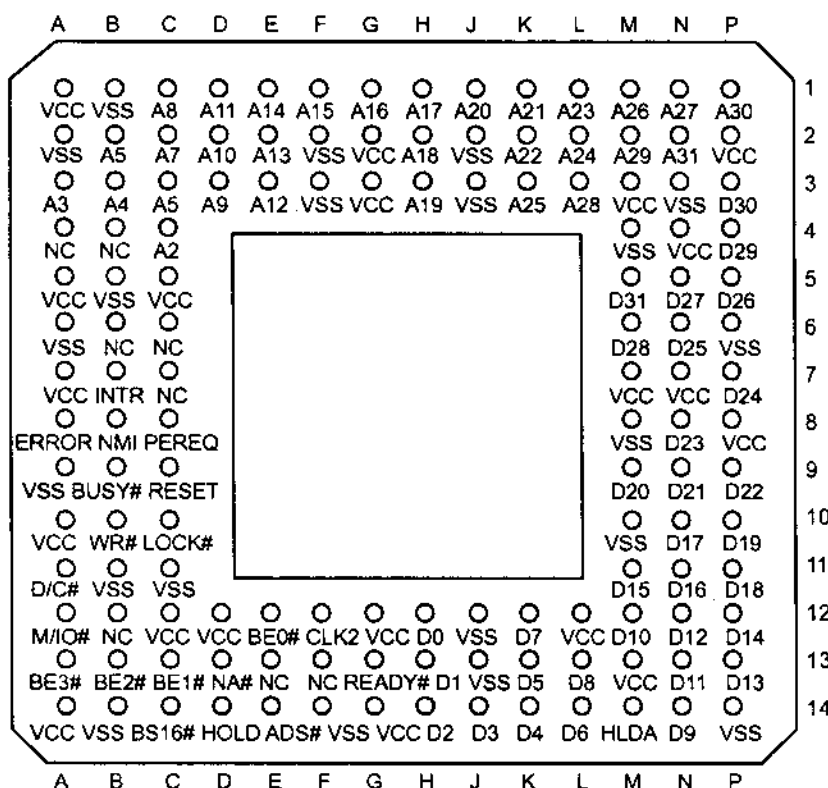
2.4.7. Vi xử lý 80386

(1) Đóng vỏ và các chân tín hiệu của 80386

Chip 80286 có một số hạn chế, đó là, nó chỉ xử lý dữ liệu 16-bit, chỉ có dung lượng đoạn nhớ tối đa là 64 kbyte, và không thể chuyển đổi dễ dàng giữa 2 chế độ làm việc: chế độ thực (chế độ 8086) và chế độ bảo vệ. Chip 80386 xử lý dữ liệu 32-bit (ALU xử lý dữ liệu 32-bit), có cơ chế chuyển đổi chế độ thực và bảo vệ dễ dàng, dung lượng đoạn nhớ có thể đạt tới 4 GB, và một chương trình có thể chiếm tới 16.384 đoạn nhớ. Không gian địa chỉ ảo có thể đạt tới 64 TB (Terabyte), hay bằng 16.384×4 GB. Bus địa chỉ 32-bit của 80386 có thể địa chỉ tới 4 GB bộ nhớ vật lý. 80386 có chế độ "8086 ảo" cho phép chuyển đổi dễ dàng giữa 2 chế độ thực 8086 và chế độ bảo vệ. Đây là điểm rất khác so với chip 80286.

80386 có 2 phiên bản: 386DX và 386SX. 386DX có bus địa chỉ 32-bit và bus dữ liệu 32-bit. Các chip 80386 đóng gói vỏ gồm PGA

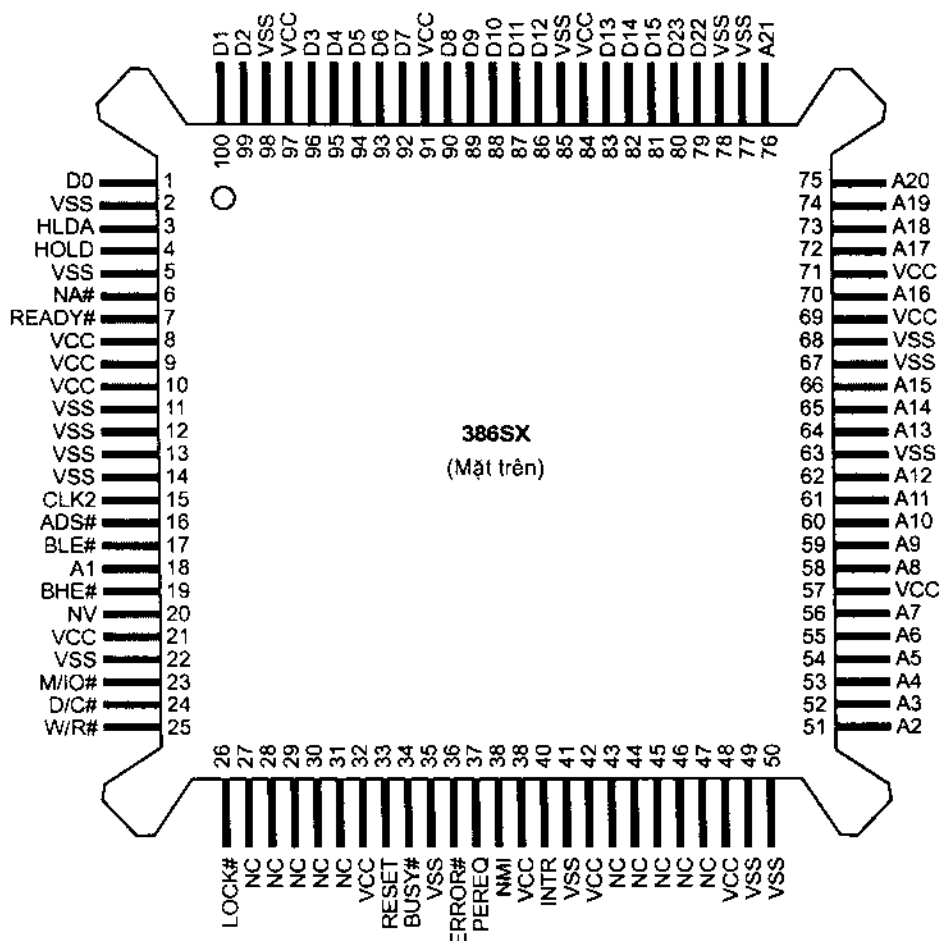
(Ceramic pin grid array package) 132-pin (hình 2.110) với công nghệ CHMOS III. 386SX có cấu trúc bên trong giống như 386DX nhưng chỉ có bus địa chỉ 24-bit và bus dữ liệu 16-bit. Nó đóng gói vỏ vuông phẳng 100 pin (flatpack) (hình 2.111).



Hình 2.110: Đóng vỏ gốm PGA và các tín hiệu của 386DX

386SX ra đời sau 386DX, và phù hợp với thiết bị nhớ và ngoại vi 8-bit và 16-bit thông dụng lúc bấy giờ và do đó máy tính dùng chip 386SX có giá thành hạ. Nó được dùng trong các hệ thống máy tính nhỏ với bo mạch chủ dùng cho cả 80286. Phiên bản mới của 80386 là 80386SL cũng được dùng trong nhiều loại IBM PC/AT.

Cũng như truyền thống, 80386 dùng điện thế V_{ss} , $V_{cc} = 5,0$ V và tiêu thụ dòng trung bình 550 mA đối với phiên bản tần số 25 MHz, 500 mA - phiên bản 20 MHz, 450 mA - phiên bản 16 MHz, 600 mA - phiên bản 33 MHz.



Hình 2.111: Đóng vỏ mặt phẳng vuông và các tín hiệu của 386SX

Mỗi một chân tín hiệu ra của 80386 có khả năng hạ xuống mức thấp (logic 0) với 4,0 mA (các đường địa chỉ và dữ liệu) hoặc 5,0 mA (các đường tín hiệu khác). Mỗi một chân tín hiệu vào chỉ yêu cầu một tải tối thiểu với dòng $\pm 10 \mu A$. Trong hầu hết các hệ thống máy tính mức dòng nhỏ như thế này yêu cầu phải có các bộ đệm. Để tiện cho đọc tín hiệu, vị trí các chân tín hiệu của 386DX và 386SX được liệt kê theo tọa độ chân trên vỏ như cho trong hình 2.112.

H12	D0	BE0#	E12	18	A1	D0	1
H13	D1	BE1#	C13	51	A2	D1	100
H14	D2	BE2#	B13	52	A3	D2	99
J14	D3	BE3#	A13	53	A4	D3	98
K14	D4	A2	C4	54	A5	D4	95
K13	D5	A3	A3	55	A6	D5	94
L14	D6	A4	B3	56	A7	D6	93
K12	D7	A5	B2	58	A8	D7	92
L13	D8	A6	C3	59	A9	D8	90
N14	D9	A7	C2	60	A10	D9	89
M12	D10	A8	C1	61	A11	D10	88
N13	D11	A9	D3	62	A12	D11	87
N12	D12	A10	D2	64	A13	D12	86
P13	D13	A11	D1	65	A14	D13	83
P12	D14	A12	E3	66	A15	D14	82
M11	D15	A13	E2	70	A16	D15	81
N11	D16	A14	E1	72	A17		
N10	D17	A15	F1	73	A18	ADS#	16
P11	D18	A16	G1	74	A19	BHE#	19
P10	D19	A17	H1	75	A20	BLE#	17
M9	D20	A18	H2	76	A21	D/C#	24
N9	D21	A19	H3	79	A22	M/IO#	23
P9	D22	A20	J1	80	A23	HOLD	4
N8	D23	A21	K1			HLDA	3
P7	D24	A22	K2	34	BUSY#		
N6	D25	A23	L1	15	CLK2	INTR	40
P5	D26	A24	L2	36	ERROR#	NMI	38
N5	D27	A25	K3	28	FLT#		
M6	D28	A26	M1	26	LOCK#	RESET	33
P4	D29	A27	N1	6	NA#		
P3	D30	A28	L3	37	PEREQ	W/R#	25
M5	D31	A29	M2	7	READY#		
		A30	P1				
E14	ADS#	A31	N2				
D13	NA#						
C14	BS16#	WR#	B10				
G13	READY#	D/C#	A11				
		M/IO#	A12				
D14	HOLD	LOCK#	C10				
M14	HOLDA						
		PEREQ	C8				
B7	INTR	BUSY#	B9				
B8	NMI	ERROR#	A8				
C9	RESET						
F12	CLK2						

386DX

Hình 2.112: Các tín hiệu của 386DX/SX
và sắp xếp theo tọa độ các chân tín hiệu

Chức năng của các chân tín hiệu của 80386 được phân chia thành các nhóm như sau:

1. A31-A2: bus địa chỉ: sử dụng để đánh địa chỉ 1 G×32 vùng nhớ. Các đường A0 và A1 được ký hiệu thành các tín hiệu cho phép bus BE3#-BE0# (bus enable) (80386SX chỉ có các đường địa chỉ A23-A1).

2. D31-D0: bus dữ liệu: sử dụng để vận chuyển dữ liệu giữa vi xử lý và bộ nhớ, thiết bị vào/ra (80386 chỉ có D15-D0).

3. BE3#-BE0#: Cho phép băng nhớ: sử dụng để truy cập một byte, một từ, từ kép của dữ liệu. Những tín hiệu này được vi xử lý tạo ra nhờ A1 và A0 (386SX có BHE# và BLE# cũng để dùng để chọn các băng nhớ).

4. M/IO# (Memory/IO): sử dụng để chọn bộ nhớ (khi M/IO# = 1) hoặc chọn thiết bị vào/ra (khi M/IO# = 0). Trong quá trình thao tác với thiết bị vào/ra chỉ dùng 16 đường địa chỉ để đánh địa chỉ các cổng vào/ra).

5. W/R# (Write/read): chỉ ra rằng chu trình bus là chu trình ghi (khi W/R# = 1) hay chu trình đọc (khi W/R# = 0).

6. ADS# (Address Data Strobe): trở nên tích cực khi 80386 đã đưa ra các giá trị địa chỉ có nghĩa của bộ nhớ hay thiết bị vào/ra. (Tín hiệu này kết hợp với W/R# tạo ra các tín hiệu ghi và đọc riêng biệt trong 8088/8086).

7. RESET: xóa: khởi tạo trạng thái ban đầu cho 80386 để bắt đầu thực hiện chương trình ở địa chỉ nhớ FFFFFFF0h. 80386 được xóa đưa về chế độ đọc và 12-bit cao của địa chỉ duy trì giá trị FFFh cho đến khi một lệnh nhảy xa hoặc gọi xa được thực hiện khi đó chúng được xóa về 000h.

8. CLK2: nhịp đồng hồ 2: được tạo ra nhờ nhân đôi tần số nhịp vận hành của 80386. Ví dụ, để vận hành 80386 ở tần số 16 MHz, chúng ta ghim vào chân CLK2 này một nhịp đồng hồ tần số 32 MHz.

9. **READY#**: Ready: điều khiển một số trạng thái chờ được đưa vào để truy cập bộ nhớ, vì tốc độ vi xử lý thường nhanh hơn so với bộ nhớ.

10. **LOCK#**: khóa: trở về trạng thái logic 0 khi có một lệnh bị khóa nhờ LOCK. Tín hiệu này dùng cho trường hợp có truy cập DMA.

11. **D/C#** (Data/Control): chỉ ra rằng bus dữ liệu chứa dữ liệu gửi tới (hoặc từ) bộ nhớ hay thiết bị vào/ra khi ở mức logic 1. Nếu $D/C\# = 0$ thì vi xử lý bị dừng hoặc đang thực hiện một xác nhận ngắt.

12. **BS16#**: kích thước bus là 16-bit: Khi $BS16\# = 1$ thì chọn bus dữ liệu 32-bit, ngược lại khi $BS16\# = 0$ thì chọn bus dữ liệu 16-bit.

13. **NA#** (Next Address): gây ra cho 80386 đưa ra địa chỉ của lệnh hoặc dữ liệu tiếp theo trong chu trình bus. Tín hiệu này thường dùng để cho địa chỉ đường ống.

14. **HOLD**: yêu cầu một tác động DMA (giống như trong 8088/8086).

15. **HLDA** (chấp nhận HOLD): chỉ ra rằng 80386 đang trong trạng thái HOLD.

16. **PEREQ#** (Coprocessor Request): yêu cầu 80386 kết nối với 80387.

17. **BUSY#** (Bận): một đầu vào được sử dụng nhờ lệnh WAIT hoặc FWAIT để chờ đợi bộ đồng xử lý rồi. Nó kết nối trực tiếp 80386 và 80387 (tương tự như TEST# trong 8088/8086).

18. **ERROR#** (Lỗi): chỉ cho vi xử lý 80386 rằng lỗi xuất hiện ở 80387.

19. **INTR** (Yêu cầu ngắt): mạch logic xử lý ngắt bên ngoài tạo ra tín hiệu này để yêu cầu vi xử lý phục vụ ngắt (ví dụ, yêu cầu vào/ra).

20. **NMI** (ngắt không che được): một yêu cầu phục vụ ngắt không che được (cờ IF không có tác dụng).

(2) Các thanh ghi trong mô hình lập trình của 80386

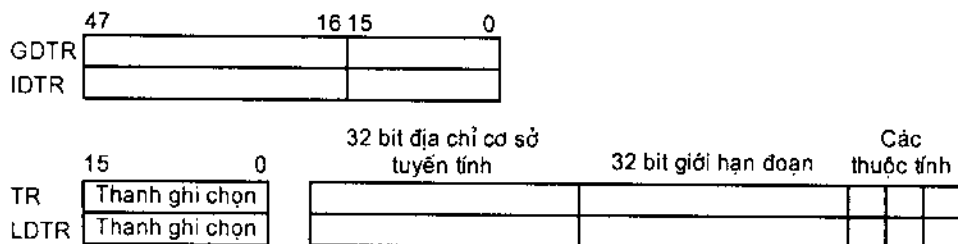
Khi xét đặc điểm của các loại vi xử lý X86, chúng ta đã nói đến mô hình lập trình của X86, kể từ 80386 đến Pentium. Tám thanh ghi chung 16-bit: AX, BX, CX, DX, SP, BP, DI, SI của mô hình lập trình 8086/8088 đã được mở rộng đến 32 bit và chúng được gọi là: EAX, EBX, ECX, EDX, ESP, EBP, EDI, ESI. Như vậy, khi làm việc ở chế độ thực (8086) thì 80386 sử dụng các thanh ghi chung 16-bit và các chương trình viết cho 8086 có thể thực hiện trên 80386 ở chế độ thực. Ngoài các thanh đoạn 16-bit: CS, DS, ES, SS giống như của 8086/8088, 80386 còn có thêm 2 thanh ghi đoạn FS và GS. Và tất cả 6 thanh ghi đoạn này còn được gọi là các thanh ghi chọn (Selectors). Trong chế độ thực các thanh ghi đoạn chứa địa chỉ cơ sở của các đoạn nhớ dung lượng 64 kbyte và trong chế độ bảo vệ, chúng chứa một giá trị thanh ghi chọn. Giá trị này trỏ tới một thanh ghi mô tả (Descriptor) trong bảng các mô tả chứa trong bộ nhớ. Thanh ghi mô tả đến lượt mình trỏ tới vùng nhớ cần truy cập. Các thanh ghi FS và GS được sử dụng như là một tiền tố (prefix) trong câu lệnh mã ngữ để địa chỉ dữ liệu trong các đoạn FS hay GS ở bất kỳ chế độ địa chỉ nào, ví dụ: MOV EAX, GS: DATA.

Con trỏ lệnh IP 16-bit của 8086 đã được mở rộng thành 32-bit, và gọi là EIP. Thanh ghi cờ Flags 16-bit của 8086 cũng được mở rộng thành 32-bit và gọi là EFlags. Thanh ghi EFlags bao gồm 16-bit thấp có ý nghĩa như Flags của 8086/8088. Phần các cờ mở rộng đã được trình bày khi xét đến đặc điểm của thế hệ X86. Trong chế độ thực, 16-bit thấp của EIP chứa Offset của lệnh tiếp theo trong chương trình. Trong chế độ bảo vệ, tất cả 32 bit của EIP được sử dụng để địa chỉ đoạn chứa chương trình hoặc dữ liệu có dung lượng 4 GB.

• Các thanh ghi quản lý bộ nhớ của 80386

Cũng như trong 80286, chip 80386 có các thanh ghi GDTR, LDTR, IDTR, và TR. Những thanh ghi này có chức năng giống như chúng đã có trong 80286, chỉ khác ở chỗ: địa chỉ cơ sở (Base address)

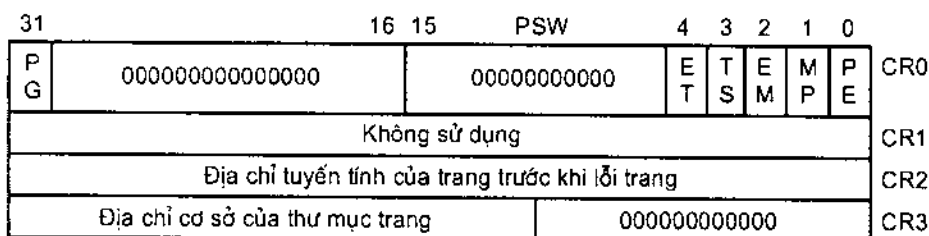
32-bit và giá trị giới hạn (Limit) 20-bit thay cho giá trị địa chỉ cơ sở 24-bit và LIMIT 16-bit trong 80286 (hình 2.113).



Hình 2.113: Các thanh ghi quản lý bộ nhớ

• Các thanh ghi điều khiển của 80386

Bổ sung cho EIP và EFlags, có thêm các thanh ghi điều khiển trong 80386. Đó là thanh ghi CR0 (tương tự như thanh ghi MSW trong 80286, chỉ khác là CR0 có 32 bit, còn MSW có 16 bit), CR1, CR2, và CR4 (hình 2.114).



Hình 2.114: Các thanh ghi điều khiển của 80386

Thanh ghi CR1 không dùng trong 80386, nhưng để dự phòng. Thanh ghi CR2 chứa địa chỉ tuyến tính của trang được truy cập cuối cùng trước khi xảy ra ngắt do lỗi trang. Địa chỉ tuyến tính (linear address) là địa chỉ 32-bit được tạo ra trong lệnh máy của vi xử lý mà nó có thể giống hoặc không giống như địa chỉ vật lý. Thanh ghi CR3 chứa địa chỉ cơ sở của thư mục trang. 12 bit thấp của CR3 bằng 0 và kết hợp với các bit còn lại của CR3 để xác định dải địa chỉ bắt đầu từ 4 kbyte.

Thanh ghi CR0 có một số bit điều khiển sau đây:

1. PG: Khi PG = 1, chọn chuyển đổi bảng trang của các địa chỉ tuyến tính sang địa chỉ vật lý. Sự chuyển đổi bảng trang cho phép bất kỳ địa chỉ tuyến tính nào sang địa chỉ vật lý.

2. ET: Khi ET = 0, chọn 80287 và khi ET = 1, chọn 80387. Sở dĩ có bit ET vì 80386 ra đời trước khi có 80387, và 80386 có kết nối với 80287 được.

3. TS: chỉ ra rằng 80386 có sự chuyển nhiệm vụ (task switching). Nếu TS = 1, thì lệnh đồng xử lý gây ra ngắt loại 7 (ngắt khi không có 80287 hoặc 80387).

4. EM: được thiết lập để gây ra ngắt loại 7 cho các lệnh ESC (ESCAPE instruction). Các lệnh ESC được sử dụng để mã hóa các lệnh cho 80387. Chúng ta thường sử dụng ngắt này để mô phỏng (bằng phần mềm) chức năng của đồng xử lý. Sự mô phỏng làm giảm thiểu chi phí hệ thống nhưng đòi hỏi phải mất hơn 100 lần thời gian để thực hiện các lệnh đồng xử lý.

5. MP: được thiết lập để chỉ ra rằng bộ đồng xử lý (80287 hoặc 80387) đã có trong hệ thống.

6. PE: được thiết lập để chuyển từ chế độ thực sang chế độ bảo vệ cho 80386. Có thể sử dụng lệnh MOV vào CR0 hoặc dùng lệnh LMSW để thiết lập PE. Bit PE có thể được xóa (cũng bằng cách như khi thiết lập) để chuyển từ chế độ bảo vệ đến chế độ thực. Trong 80286 bit này cũng được thiết lập tương tự nhưng không thể được xóa bằng phần mềm mà phải bằng phần cứng RESET.

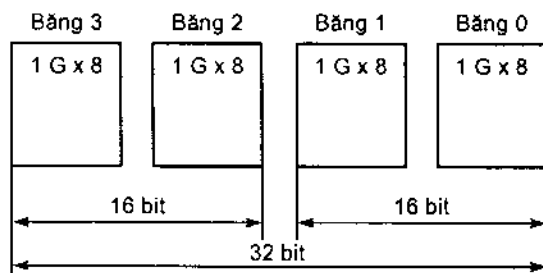
• Các thanh ghi gỡ rối (DR0 ÷ DR7) và kiểm tra (TR0 ÷ TR7) của 80386

Các thanh ghi 32-bit DR0÷DR7 và TR0÷TR7 không có trong 8086/8088/80286. DR0÷DR7 được sử dụng cho các mục đích gỡ rối (debugging) và TR0-TR7 dùng cho các mục đích kiểm tra. DR0÷DR3 chứa 32 bit địa chỉ tuyến tính của điểm ngắt (Breakpoint linear

addresses), DR4÷DR5 không sử dụng (để dự phòng). Các địa chỉ điểm ngắt có thể có trong lệnh hoặc trong dữ liệu được liên tục so sánh với các địa chỉ có trong chương trình. Nếu có sự trùng hợp thì 80386 gây ra một ngắt loại 1 (TRAP hoặc debug interrupt). Thanh ghi DR6 chỉ ra trạng thái hiện thời của các điểm ngắt gỡ rối, thanh ghi DR7 được sử dụng để thiết lập các điểm ngắt gỡ rối. Chỉ những chương trình chạy ở mức thẩm quyền cao mới có thể xâm nhập vào các thanh ghi này.

(3) Hệ thống nhớ của 80386

Không gian nhớ: Bộ nhớ vật lý của 80386DX tối đa là 4 GB. Không gian địa chỉ ảo có thể đến 64 TB (1 Tera = 2^{40}) và sắp xếp thành các không gian nhớ vật lý dung lượng 4 GB nhờ MMU. Hình 2.115 mô tả tổ chức hệ thống nhớ vật lý của 80386.



Hình 2.115: Hệ thống nhớ tổ chức thành 4 băng của 80386

Với độ rộng dữ liệu 32-bit (386DX), có thể tổ chức truy cập trực tiếp bộ nhớ theo byte, từ và từ kép trong một chu trình nhớ, trong khi đó, trong 8088 phải cần tới 4 chu trình nhớ và 80286 cần tới 2 chu trình nhớ. Các byte nhớ trong hệ thống máy tính 80386 được đánh địa chỉ bằng 8 chữ số hệ 16 (Hex) từ 00000000h đến FFFFFFFFh. Truy cập hai băng nhớ trong 8086 thông qua A0 và BHE#, trong khi đó trong 80386 để truy cập 4 băng nhớ cần tới BE0# đến BE3#. Như vậy, địa chỉ đầu của băng nhớ 1 là 00000000h, địa chỉ đầu của băng nhớ 2 là 00000001h, địa chỉ đầu của băng 3 là 00000002h, và địa chỉ đầu của băng 4 là 00000003h. 80386DX không có A0 và A1 vì chúng được biểu diễn thông qua BE0# và BE1#.

80386 16 MHz yêu cầu kết nối bộ nhớ DRAM với thời gian truy cập 50 ns hoặc nhỏ hơn để vận hành đủ tốc độ. Nhưng thực tế, không có loại DRAM nào vào thời điểm ra đời 80386 đạt được tốc độ truy cập nhanh như vậy. Các loại DRAM chỉ đảm bảo thời gian truy cập vào khoảng từ 55 ns đến 100 ns. Điều này bắt buộc phải tìm phương pháp kỹ thuật kết nối các bộ nhớ tốc độ thấp với 80386. Đó là các phương pháp: kỹ thuật nhớ xen kẽ (interleaved memory), caching (nhớ trung gian tốc độ nhanh) và lập đường ống (pipeline).

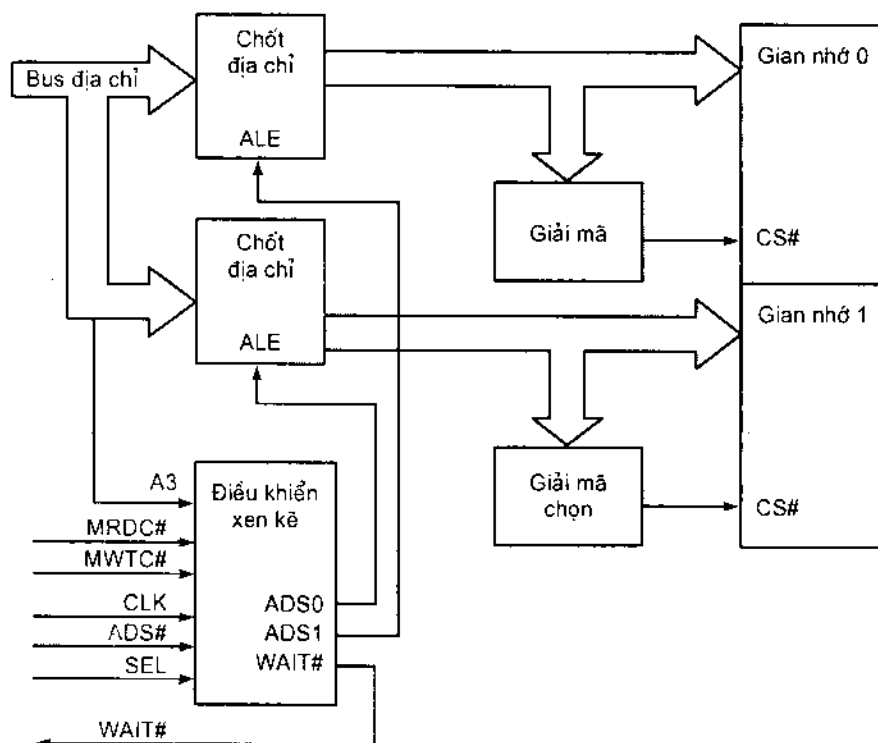
Kỹ thuật nhớ xen kẽ (interleaves): Kỹ thuật bộ nhớ xen kẽ được áp dụng đối với 80386 nhằm mục đích kéo dài thời gian truy cập bộ nhớ mà không cần phải đưa vào các trạng thái chờ đợi trong chu kỳ bus. Một hệ thống nhớ cần phải có hai tập hợp của bus và một logic điều khiển để cung cấp các địa chỉ cho từng bus. Bộ nhớ xen kẽ được chia thành hai phần (hình 2.116).

Đối với 80386, một phần có các địa chỉ 32-bit 000000h÷000003h, 000008h÷00000Bh,..., phần còn lại có các địa chỉ 000004h÷000007h, 00000Ch÷00000Fh,... Trong khi vi xử lý truy cập tới các vùng nhớ 000000h÷000003h, thì logic điều khiển xen kẽ (interleave logic) tạo ra tín hiệu định thời (address strobe) địa chỉ (ADS#) cho vùng nhớ 000004h÷000007h. Quá trình này liên tục do vi xử lý đưa ra các địa chỉ các vùng nhớ liên tiếp nhau.

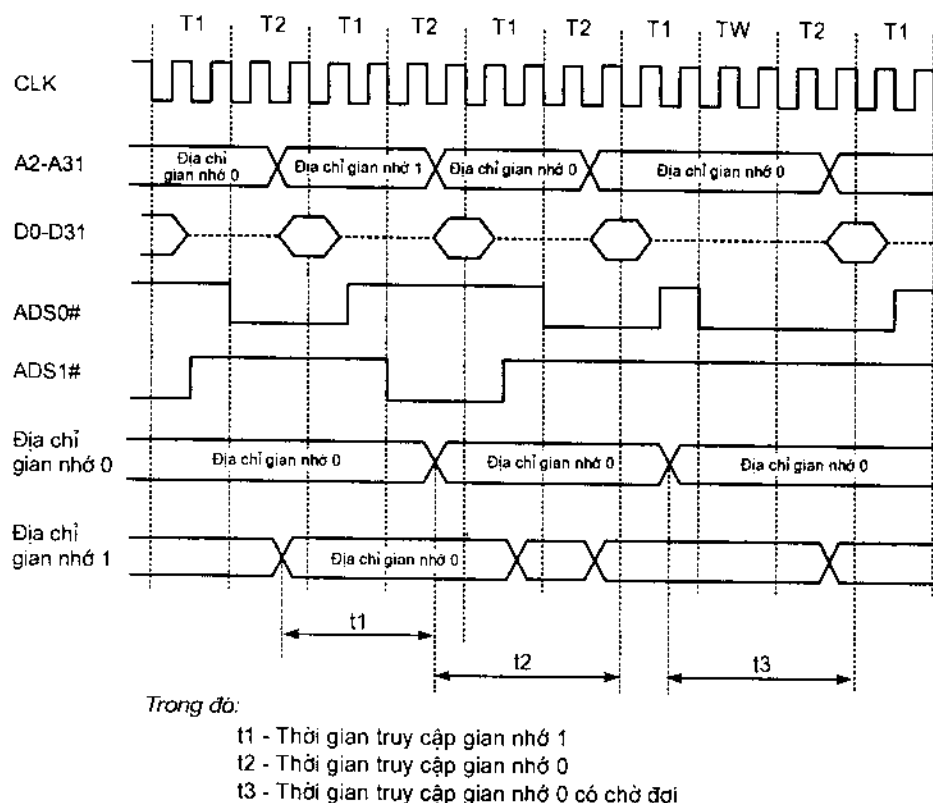
Việc đánh địa chỉ các băng nhớ luân phiên nhau làm kéo dài thời gian truy cập bộ nhớ bởi vì địa chỉ được tạo ra trước khi vi xử lý yêu cầu dữ liệu. Đây cũng bởi vì bộ vi xử lý định đường ống các địa chỉ bộ nhớ, gửi địa chỉ tiếp theo trước khi dữ liệu được đọc từ địa chỉ trước đó.

Thời gian truy cập bộ nhớ nhờ kỹ thuật xen kẽ này tăng lên từ 78 ns đến 145,5 ns với nhịp đồng hồ hệ thống là 16 MHz. Trong khi

đó, nếu không có kỹ thuật này thì phải đưa vào các trạng thái chờ khi truy cập bộ nhớ với tổng số thời gian là 140,5 ns nếu nhịp đồng hồ là 8 MHz. Như thế, kỹ thuật bộ nhớ xen kẽ cho một tốc độ truy cập tương tự như có sự đưa vào một trạng thái chờ. Nếu nhịp đồng hồ là 33 MHz, thì bộ nhớ xen kẽ yêu cầu 72,75 ns trong khi giao tiếp với bộ nhớ chuẩn cần 46 ns. Với nhịp đồng hồ cao như thế này thì một bộ nhớ DRAM 55 ns là thích hợp hơn khi sử dụng bộ nhớ kỹ thuật xen kẽ không cần có các trạng thái chờ đợi khi truy cập bộ nhớ trong chu trình bus. Sơ đồ khối một hệ thống nhớ dùng kỹ thuật xen kẽ với sự đánh địa chỉ luân phiên các gian nhớ (memory section). Trong ví dụ này có 2 gian nhớ.



Hình 2.116: Hệ thống nhớ xen kẽ với logic điều khiển xen kẽ và các chốt địa chỉ



Hình 2.117: Đồ thị thời gian truy cập bộ nhớ xen kẽ với 2 gian nhớ

Trước hết, nếu tín hiệu chọn gian nhớ $SEL = 0$ (không tích cực), thì tín hiệu $WAIT\#$ từ logic điều khiển xen kẽ ở mức logic 1 (không tích cực). Các tín hiệu ra $ADS0\#$ và $ADS1\#$ dùng để điểm nhịp (strobe) chốt địa chỉ cho các gian nhớ. Nếu cả 2 tín hiệu này ở mức logic 1 (tích cực) sẽ thực hiện chốt các địa chỉ. Khi $SEL = 1$, địa chỉ A3 được dùng để xác định chọn mạch chốt địa chỉ nào. Tín hiệu $ADS\#$ ở mức 0 được so sánh với các tín hiệu $ADS0\#$ và $ADS1\#$ với trạng thái trước đó. Nếu có một gian nhớ đang được truy cập lại được truy cập thêm lần thứ 2 nữa, thì tín hiệu $WAIT\#$ chuyển đến trạng thái 0 yêu cầu một trạng thái chờ đợi. Hình 2.117 là đồ thị thời gian truy cập bộ nhớ xen kẽ với các tín hiệu địa chỉ cho cả 2 gian nhớ.

(4) Kỹ thuật đường ống

Kỹ thuật đường ống trong 80386 cho phép bộ nhớ có thêm một chu kỳ nhịp phụ để truy cập dữ liệu. Nhịp phụ mở rộng thời gian truy cập từ 50 ns đến 81 ns của 80386 16 MHz. Đường ống, như thường gọi, được thiết lập bởi bộ vi xử lý. Khi một lệnh được đọc từ bộ nhớ, bộ vi xử lý có thêm thời gian trước khi đọc lệnh tiếp theo từ bộ nhớ. Trong quãng thời gian phụ này, địa chỉ của lệnh tiếp theo được gửi ra bus địa chỉ của hệ thống. Thời gian phụ này (một chu kỳ nhịp) được sử dụng để tạo ra thời gian bổ sung truy cập các vi mạch nhớ tốc độ chậm.

Tuy vậy, không phải tất cả các tham chiếu bộ nhớ có thể tận dụng được ưu điểm của kỹ thuật đường ống, bởi vì một số chu kỳ bộ nhớ không phải theo đường ống. Và những chu kỳ nhớ không theo đường ống yêu cầu một trạng thái chờ.

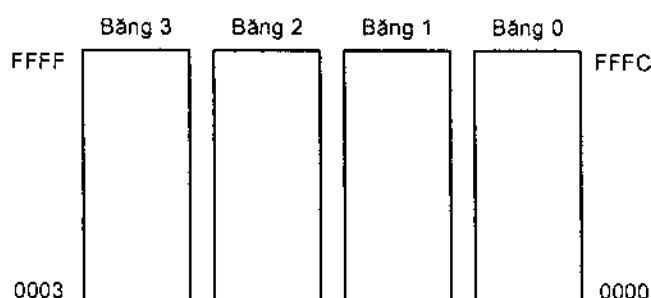
Kỹ thuật đường ống nói chung là một đặc điểm nhằm giảm thời gian truy cập bộ nhớ cần thiết áp dụng trong các hệ thống tốc độ chậm.

(5) Hệ thống vào/ra của 80386

Hệ thống vào/ra (I/O system) của 80386 dựa trên nguyên tắc chung đã dùng cho các loại 8088/8086. Có 64 kbyte khác nhau của không gian vào/ra. Địa chỉ cổng vào/ra xuất hiện trên bus địa chỉ A15-A2 với các tín hiệu BE3#-BE0# để chọn một byte, một từ, từ kép của dữ liệu vào/ra.

Không giống như 8088/8086, 80386 sử dụng hệ thống vào/ra rộng 32-bit và chia thành 4 băng. Hầu hết các trao đổi vào/ra là 8-bit vì chúng ta thường dùng mã ASCII 7-bit để chuyển dữ liệu ký tự. Nhưng hiện tại các thiết bị vào/ra đã cho phép truyền dữ liệu 16-bit và 32-bit (EISA và bus cục bộ) như thiết bị nhớ trên đĩa từ, và giao tiếp với màn hình. Với độ rộng dữ liệu này tốc độ vận chuyển dữ liệu giữa vi xử lý và thiết bị vào/ra cao hơn nhiều so với thế hệ 8-bit.

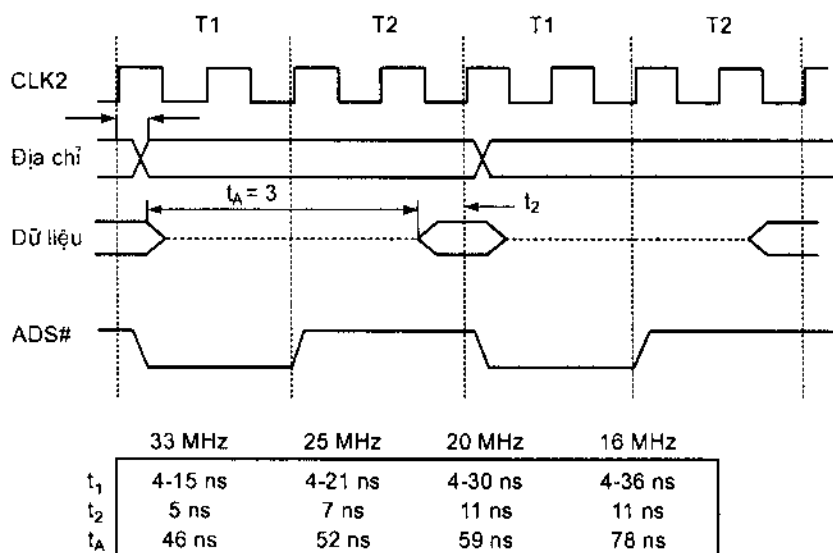
Các vùng vào/ra được đánh số từ 0000h đến FFFFh (hình 2.118). Bộ đồng xử lý toán học 80387 sử dụng vùng vào/ra 800000F8h-800000FFh để kết nối giữa 80387 và 80386.



Hình 2.118: Sắp xếp hệ thống vào/ra với 4 bảng, mỗi bảng 64 kbyte, từ 0000h đến FFFFh của 80386

(6) Định thời (timing) của 80386

Giống như 8088/8086, 80386 điều khiển bộ nhớ và vào/ra bằng các tín hiệu riêng biệt. Tín hiệu M/IO# chỉ ra khi nào có sự trao đổi dữ liệu với bộ nhớ hay thiết bị vào/ra. Cùng với M/IO# còn có xung ghi/đọc W/R#. Hình 2.119 là đồ thị thời gian của một chu kỳ đọc không sử dụng đường ống của 80386.

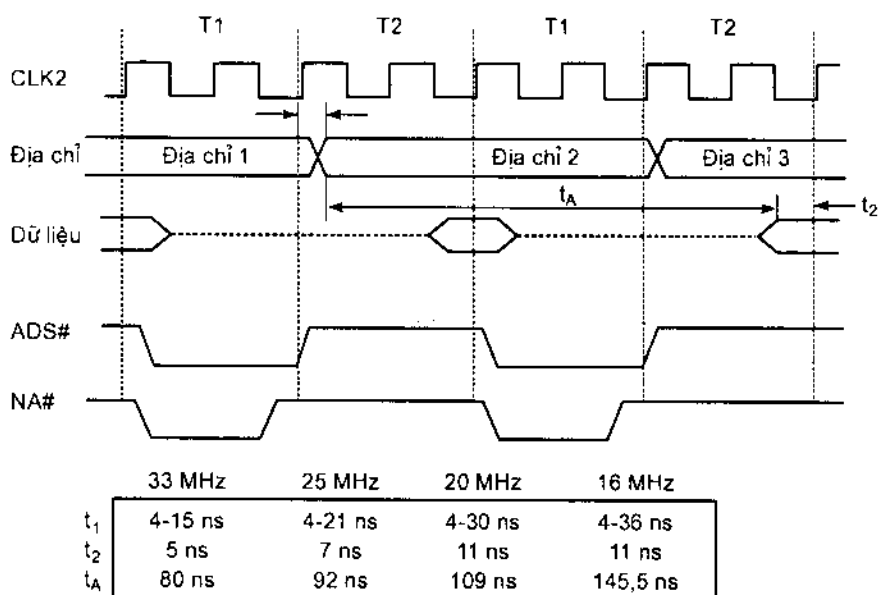


Hình 2.119: Đồ thị thời gian chu trình đọc không có đường ống của 80386

Chu trình thời gian được định thời bằng nhịp đồng hồ CLK2 (chân vào CLK2 của 80386). Một chu kỳ bus có 2 trạng thái nhịp T1

và T2. Mỗi trạng thái nhịp gồm 2 chu kỳ nhịp đồng hồ CLK2. Thời gian truy cập để đọc dữ liệu t_A được tính bằng quãng thời gian từ khi các đường địa chỉ chuyển mức tích cực và có nghĩa đến khi bắt đầu có dữ liệu lên bus. Phiên bản 80386 16 MHz cho $t_A = 78$ ns trước khi đưa các trạng thái chờ đợi vào trong chế độ vận hành không có đường ống. Để đặt chế độ không đường ống phải đặt hiệu NA# lên mức logic "1".

Chu kỳ đọc trong chế độ đường ống đạt được nếu đặt $NA\# = 0$ và sử dụng các mạch chốt địa chỉ để có được địa chỉ đường ống (hình 2.120). Xung nhịp đưa vào các mạch chốt địa chỉ là $ADS\#$. Các mạch chốt địa chỉ phải dùng trong hệ thống có kỹ thuật đường ống và hệ thống các băng nhớ xen kẽ. Số lượng tối thiểu của các băng nhớ xen kẽ là 2.



Hình 2.120: Đồ thị thời gian chu kỳ đọc có đường ống của 80386

Địa chỉ đường ống xuất hiện trong toàn bộ một chu kỳ trạng thái trước khi nó xuất hiện bình thường trong chế độ địa chỉ không đường ống. Trong phiên bản 80386 16 MHz sự xuất hiện trước này làm bổ sung thêm thời gian truy cập bộ nhớ là 67,5 ns. Vì trong chế độ địa chỉ

bình thường không có đường ống thời gian truy cập là 78 ns, do đó, trong chế độ đường ống tổng cộng ta có thời gian truy cập là 145,5 ns. Điều quan trọng và là ưu việt của chế độ đường ống là không có các trạng thái chờ đợi trong các chu trình bus và vi xử lý có thể kết nối với nhiều loại bộ nhớ bán dẫn có tốc độ chậm. Nhưng nhược điểm của kỹ thuật đường ống là cần phải có hệ thống nhớ gồm các băng xen kẽ nhau cho một đường ống, điều này làm phức tạp thêm mạch điều khiển, chi phí và các trạng thái trễ bên trong.

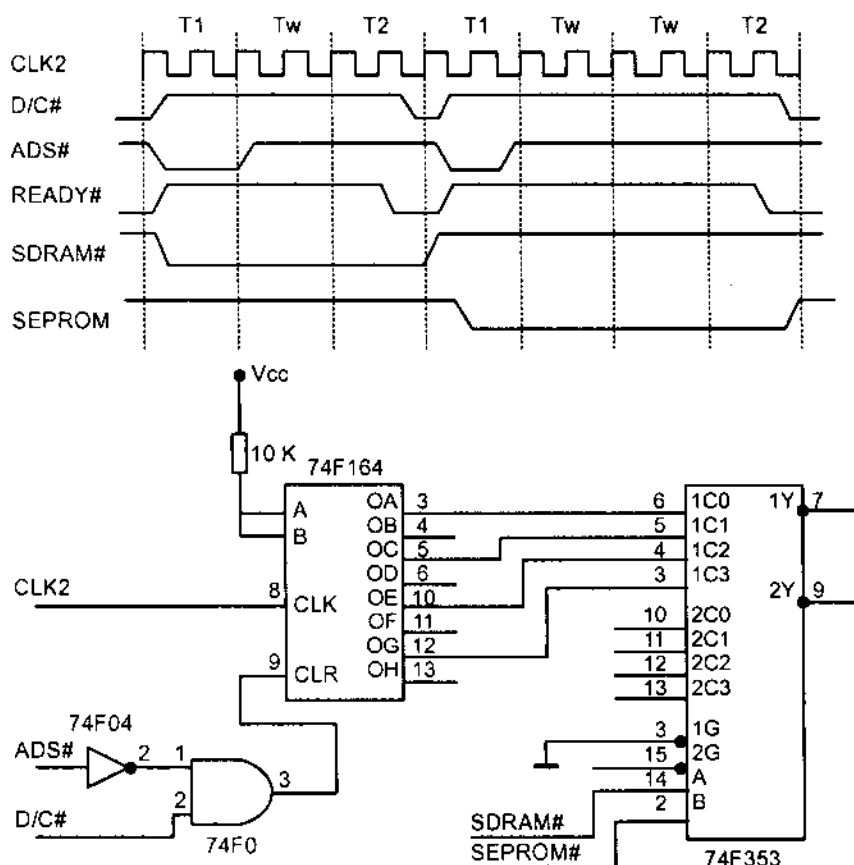
Các trạng thái chờ đợi cho 80386:

Cũng như 8088/8086, 80386 phải cần tới các trạng thái chờ nếu kết nối với các bộ nhớ bán dẫn có tốc độ truy cập chậm hơn so với tốc độ truy cập yêu cầu của chính vi xử lý. Trong chế độ không có đường ống của phiên bản 80386 33 MHz thời gian truy cập yêu cầu chỉ là 46 ns. Nhưng như vậy không có một loại bộ nhớ DRAM nào có tốc độ đạt nhỏ đến như vậy. Do đó, để truy cập bộ nhớ cần phải đưa vào các trạng thái chờ đợi. Thời gian truy cập tối thiểu của DRAM là 60 ns và của EPROM là 100 ns, nghĩa là cần phải có một thời gian chờ đối với DRAM là 60 ns và 2 thời gian chờ đối với EPROM.

Đầu vào READY# điều khiển đưa vào hoặc không đưa vào trạng thái chờ trong định thời chu trình bus. Chân vào READY# của 80386 là đầu vào linh hoạt, nó ở mức tích cực trong từng chu trình bus. Hình 2.121 cho ta thấy một số ít các chu trình bus không có trạng thái chờ và một chu trình bus có chu kỳ nhịp chờ (T_w). READY# = 0 ở cuối chu trình bus (một chu trình bus có hai trạng thái nhịp T1 và T2), hay ở cuối chu trình T2. Nếu READY# = 1 ở cuối chu kỳ nhịp đồng hồ thì nhịp đó là nhịp chờ đợi T_w và vi xử lý tiếp tục kiểm tra READY# cho đến khi READY# xuống mức logic "0" (kết thúc chu trình bus).

Trong chế độ không đường ống khi ADS# = 0 thì READY# = 1, sau đó ADS# trở về 1, còn READY# trở về 0 sau khi nhịp đồng hồ thứ nhất đưa vào 0 trạng thái chờ. Nếu 1 trạng thái chờ được đưa vào,

READY# duy trì mức logic 1 cho đến khi ít nhất có 2 nhịp đồng hồ. Tùy thuộc vào loại bộ nhớ bán dẫn nhanh hoặc chậm có thể có 0,1,2,3, hoặc 4 trạng thái chờ T_w , mỗi trạng thái chờ kéo dài 2 chu kỳ nhịp CLK2. Mạch tạo ra tín hiệu READY# và đồ thị xung tạo trạng thái chờ khi truy cập DRAM (1 trạng thái chờ) và EPROM (2 trạng thái chờ) mô tả trong hình 2.121, trong đó: SDRAM#: tín hiệu chọn DRAM, SEEPROM: tín hiệu chọn EPROM.



Hình 2.121: Mạch và đồ thị thời gian tạo trạng thái chờ cho DRAM và EPROM

(7) Tập lệnh của x86

Thế hệ vi xử lý công nghệ cao của Intel, được gọi tắt là x86 (phần Phụ lục 2), trong đó có 80386 có tập lệnh tương tự nhau nên

được gọi chung là tập lệnh x86. Tập lệnh x86 chứa tất cả các lệnh của 8086 nhưng bổ sung thêm một số lớn lệnh mới và các chế độ đánh địa chỉ mới. Nhiều lệnh mới bởi vì x86 đã xử lý với các thanh ghi 32-bit. Trong x86 (kể từ 80386 đến Pentium) không chỉ thực hiện các lệnh và dữ liệu theo byte, mà còn theo từ (16-bit) từ kép (32-bit).

(8) Quản lý bộ nhớ và các nhiệm vụ (tasks) của 80386

a) Chế độ thực (real mode), hay chế độ 8086 của 80386

Từ 80386 đến Pentium sự chuyển đổi chế độ thực và ảo đã linh hoạt và dễ dàng hơn. Chế độ thực có kiến trúc cơ sở giống như 8086 nhưng cho phép truy cập tới tệp thanh ghi 32-bit bên trong. Khi bộ vi xử lý bị xóa nhờ RESET hoặc khi bật nguồn, nó tự động chuyển về chế độ thực. Toán hạng của lệnh máy ở chế độ này có kích thước ngầm định 16-bit. Kích thước của đoạn cố định là 64 kbyte và các đoạn có thể xếp chồng lên nhau (overlap). Kích thước của bộ nhớ vật lý chỉ đạt 1 MB. Vì sự phân trang không có trong chế độ này nên địa chỉ logic (hay còn gọi là địa chỉ tuyến tính) giống địa chỉ vật lý. Địa chỉ vật lý được tạo ra trong chế độ này bằng phép cộng nội dung của thanh ghi đoạn 16-bit (có thể là CS, DS hay SS), tức là địa chỉ cơ sở của đoạn tương ứng, dịch về bên trái 4 bit với số gia EA (offset).

b) Quản lý bộ nhớ và chế độ bảo vệ (hay chế độ địa chỉ ảo) của x86

Chế độ bảo vệ của 80386 tương tự như ở 80286, cho phép thực hiện các ứng dụng của 8086, trong khi đó vẫn đảm bảo cơ chế bảo vệ bộ nhớ trong kiến trúc 32-bit. Chế độ thường xuyên của x86 là chế độ ảo PVAM (Protected Virtual Address Mode), hay thường gọi là chế độ bảo vệ. MMU quản lý bộ nhớ ảo.

MMU bên trong 80386 tương tự như MMU bên trong 80286, nhưng ngoài đơn vị phân đoạn (segmentation unit), MMU của 80386 có thêm đơn vị phân trang (paging unit). Đơn vị phân trang cho phép tạo ra các đoạn nhớ dung lượng đến 4 GB. Khi sử dụng phân trang, mỗi một đoạn có thể phân thành những đoạn nhỏ dung lượng 4 kbyte.

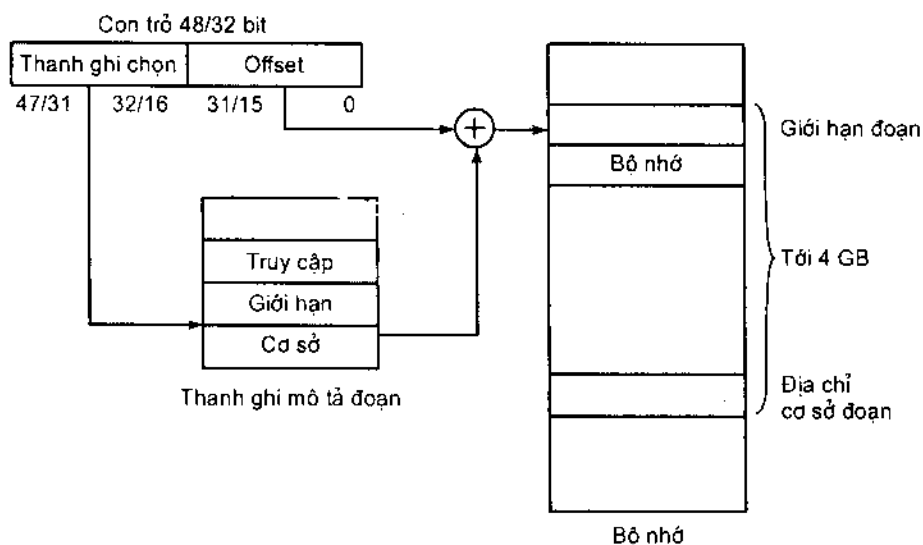
Đoạn và trang có thể được sử dụng độc lập, nhưng nếu kết hợp chung lại sẽ tạo ra một hệ thống nhớ ảo lớn. MMU thực hiện nhiệm vụ chuyển đổi các địa chỉ thành địa chỉ và sử dụng cơ chế phân trang để phân phối địa chỉ vật lý cho địa chỉ logic (địa chỉ ảo). Ví dụ, có một lệnh của chương trình truy cập tới vùng nhớ địa chỉ A0000h, thì địa chỉ vật lý thực của vùng nhớ này là 100000h, hoặc bất kỳ vùng nhớ nào khác nếu cơ chế phân trang được phép hoạt động. Sự phân trang cho phép viết chương trình với địa chỉ ảo cho bất kỳ vùng nhớ nào.

c) Các thanh ghi mô tả và các thanh ghi chọn của hệ thống 80386

80386 sử dụng các thanh ghi mô tả (Descriptors) tương tự như ở 80286. Trong cả 80286 và 80386, mỗi một thanh ghi mô tả có độ dài 8 byte chứa thông tin về vị trí của đoạn cần truy cập. Thanh ghi chọn (Selector) của 80386 được sử dụng như là một chỉ số (Index) để trở tới thanh ghi mô tả nằm trong bảng mô tả (Descriptor table) Sự khác nhau chính giữa 80286 và 80386 là 80386 có thêm các đoạn FS và GS, vì thế nó có thêm các thanh ghi mô tả cho các đoạn này. Ngoài ra, các thanh ghi mô tả của 80386 sử dụng địa chỉ cơ sở (Base) 32-bit và giá trị giới hạn đoạn (Limit) 20-bit thay cho địa chỉ cơ sở 24-bit và giá trị Limit 16-bit của các thanh ghi mô tả trong 80286. Như vậy 80386 đánh địa chỉ tới 4 GB nhớ và phân bộ nhớ thành các đoạn kích thước tới 1MB. Cũng như trong 80286, 80386 sử dụng giá trị thanh ghi chọn trở tới một thanh ghi mô tả của đoạn và giá trị OFFSET để trở tới ngăn nhớ trong đoạn cụ thể cần truy cập.

Thanh ghi mô tả của 80386 (hình 2.122) cũng bao gồm: địa chỉ cơ sở đoạn (Base address), giới hạn đoạn (Limit) và quyền truy cập tới đoạn (Access Rights). Giá trị thanh ghi chọn là mã 13-bit, nó xác định một trong 8192 thanh ghi mô tả trong bảng mô tả đoạn. Đó có thể là bảng mô tả cục bộ (LDT) hay bảng mô tả toàn cục (GDT). Nếu mỗi một thanh ghi mô tả mô tả một đoạn dung lượng tối đa là 4 GB (2^{32}) thì cả 2 bảng mô tả (LDT và GDT) thiết lập một không gian nhớ ảo dung lượng tối đa là $(2 \times 2^{13}) \times (2^{32}) = 2^6 2^{40} = 64 \text{ TB}$ (1 Terabyte = 1.024 GB

= 2^{40} byte). Thực chất hệ thống nhớ chính chỉ có tới 4 GB, nên nếu một chương trình cần tới dung lượng nhớ lớn hơn 4 GB thì phải thực hiện cơ chế đổi chỗ (Swapping) giữa hệ thống nhớ chính và thiết bị nhớ ngoài trên đĩa từ. Các hệ điều hành như Windows, Unix hay OS/2 đều có chức năng đổi chỗ trên thiết bị đĩa từ.



Hình 2.122: Đánh địa chỉ ở chế độ bảo vệ sử dụng thanh ghi đoạn chứa thanh ghi chọn

Thanh ghi mô tả của 80286

Dự trữ		Byte 6
Truy cập bên phải	Cơ sở (B23-B16)	Byte 4
Cơ sở (B15-B0)		Byte 2
Giới hạn (L15-L0)		Byte 0

Thanh ghi mô tả của 80386

Cơ sở (B24-B31)	G	D	0	A	Giới hạn (L16-L19)	Byte 6
Truy cập bên phải	Cơ sở (B23-B16)					Byte 4
Cơ sở (B15-B0)						Byte 2
Giới hạn (L15-L0)						Byte 0

Hình 2.123: So sánh các thanh ghi mô tả của 80286 và 80386

Giống như trong 80286, Giá trị thanh ghi chọn có bit TI và 2 bit RPL. Nếu TI = 0 thì chọn bảng GDT, nếu TI = 1 thì chọn bảng LDT. Ngoài GDT và LDT, còn có bảng mô tả ngắt (IDT) hay các cổng. Hình 2.123 so sánh sự khác nhau giữa các thanh ghi mô tả của 80286 và 80386. Cả hai loại đều sử dụng dữ liệu giống nhau của 6 byte đầu tiên (Base address, Limit, và Access Rights). Chỉ khác ở hai byte cuối cùng (byte 6, byte 7). Đặc tính này cho phép phần mềm của 80286 tương thích được với 80386.

Cũng giống như 80286, 80386 có 3 loại bảng mô tả: GDT, LDT, và IDT và để quản lý chúng có các thanh ghi GDTR, LDTR, và IDTR. Những thanh ghi này được nạp giá trị nhờ các lệnh tương ứng: LGDT, LLDT, và LIDT. Các trường của thanh ghi mô tả trong 80386 có những nội dung như sau:

1. Base (B0÷B31): là địa chỉ cơ sở 32 bit của đoạn nhớ trong không gian nhớ vật lý 4 GB.

2. Limit (L0÷L19): là giá trị giới hạn dung lượng của đoạn nhớ tính theo byte, từ 1 byte đến 1 MB, nếu bit G = 0. Hay là giá trị giới hạn của đoạn từ 4 kbyte đến 4 GB nếu bit G = 1.

3. Access Rights: là một byte xác định mức thẩm quyền, và những thông tin khác về đoạn nhớ. Nội dung của byte này khác nhau phụ thuộc vào loại thanh ghi mô tả.

4. Bit G: là bit nhân (granularity bit), nó chọn một số nhân là 1 hoặc 4 kbyte cho trường Limit. Nếu G = 0 thì số nhân là 1 và nếu G = 1 thì số nhân là 4 kbyte. Nếu Limit = $2^{20} = 1 \text{ MB}$ và G = 1 thì giới hạn đoạn tối đa là $2^{20} \times 2^{12} = 2^2 \cdot 2^{30} = 4 \text{ GB}$.

5. Bit D: chọn kích thước ngầm định cho thanh ghi. Nếu D = 0 thì các thanh ghi có độ dài 16 bit. Nếu D = 1 thì các thanh ghi có độ dài 32 bit. Lệnh mã ngữ USE16 và USE32 bổ sung cho lệnh giả SEGMENT trong hợp ngữ điều khiển thiết lập bit D. Trong chế độ thực, các thanh ghi luôn là 16 bit (D = 0).

6. Bit A(AVL): bit này thường chỉ ra rằng đoạn nhớ được mô tả nhờ thanh ghi mô tả có tồn tại (available).

Cơ sở (B24-B31)				G	0	0	0	Giới hạn (L16-L19)	6
P	DPL	0	Kiểu	Cơ sở (B23-B16)					4
Cơ sở (B15-B0)									2
Giới hạn (L15-L0)									0

Hình 2.124: Cấu trúc thanh ghi mô tả hệ thống của 80386

Bảng 2.11: Byte Access Rights của các thanh ghi mô tả đoạn

Vị trí bit	Tên bit	Chức năng
7	P: Present	P = 1: đoạn có tồn tại. P = 0: đoạn không có, và khi có sự truy cập qua thanh ghi mô tả thì một ngắt loại 11 sẽ xảy ra.
6-5	DPL: Descriptor Privilege Level	DPL = 00: thẩm quyền cao nhất, và DPL = 11: thẩm quyền thấp nhất. DPL sử dụng để bảo vệ truy cập vào đoạn. Nếu một đoạn được truy cập với mức thẩm quyền nhỏ hơn (số hiệu lớn hơn) DPL hiện hành thì sẽ có một ngắt sai phạm thẩm quyền xảy ra. DPL sử dụng trong các hệ thống để ngăn chặn truy cập tới một vùng nhớ.
4	S: Segment	S = 0: trường hợp thanh ghi mô tả hệ thống S = 1: trường hợp Segment (Code, Data, Stack) Segment
3	E: Executable	E = 0: chọn đoạn dữ liệu, E = 1: chọn đoạn code. E cũng xác định chức năng của 2 bit tiếp theo (X và RW)
2	X: Expansion Direction	X = 0, E = 0: đoạn dữ liệu mở rộng lên trên. OFFSET ≤ LIMIT X = 1, E = 0: đoạn ngăn xếp mở rộng xuống dưới, OFFSET > LIMIT X = 0, E = 1: mức thẩm quyền của đoạn mã bị loại bỏ X = 1, E = 1: mức thẩm quyền của đoạn mã được phục vụ
1	RW: Read/write	RW = 1, E = 0: đoạn dữ liệu có thể ghi được RW = 0, E = 0: đoạn dữ liệu không thể ghi được RW = 1, E = 1: đoạn mã có thể ghi được RW = 0, E = 1: đoạn mã không thể ghi được
0	A: Accessed	A được thiết lập mỗi khi CPU truy cập tới đoạn nhớ. Hệ thống dùng bit A để giữ trạng thái truy cập đoạn.

Các thanh ghi mô tả có trong 2 dạng: thanh ghi mô tả đoạn (Segment Descriptor) và thanh ghi mô tả hệ thống (System Descriptor). Thanh ghi mô tả đoạn xác định các đoạn dữ liệu, đoạn ngăn xếp và

đoạn mã. Còn thanh ghi mô tả hệ thống (hình 2.124) chứa thông tin về các bảng, các nhiệm vụ, và các cổng (gates) của hệ thống. Một thanh ghi mô tả đoạn của 80386 có cấu trúc như mô tả trong hình 2.127, nhưng các bit của byte Access Rights được xác định để chỉ ra các đoạn mã, đoạn dữ liệu và đoạn ngăn xếp được mô tả như thế nào (bảng 2.11).

Thanh ghi mô tả hệ thống của 80386 được sử dụng khi 80386 ở trong chế độ bảo vệ.

Giá trị 4 bit của Type trong byte Access Rights trong thanh ghi mô tả hệ thống cho ta 16 tổ hợp có ý nghĩa (bảng 2.12).

Bảng 2.12: Giá trị kiểu mô tả của thanh ghi của x86

Kiểu mô tả	Giải thích
0000	Sai (invalid)
0001	TSS trong 80286
0010	LDT
0011	TSS của 80286 bạn
0100	Cổng gọi của 80286
0101	Cổng nhiệm vụ (80286 hoặc 80386, hoặc 80486, hoặc Pentium)
0111	Cổng bẫy của 80286
1000	Sai (invalid)
1001	TSS trong 80386 (hoặc 80486, Pentium)
1010	Dự phòng cho Intel
1011	TSS của 80386 bạn
1100	Cổng gọi của 80386 (hoặc 80486, Pentium)
1101	Dự phòng cho Intel
1110	Cổng ngắt của 80386 (hoặc 80486, Pentium)
1111	Cổng bẫy của 80386 (hoặc 80486, Pentium)

Không phải tất cả các tổ hợp này đều dùng cho 80386. Một số tổ hợp dành cho 80286, một số dự phòng cho những chip vi xử lý thế hệ sau. Giá trị TYPE của 80386 giống như của 80486 và Pentium, vì vậy, ta có thể xét chung cho x86.

Có 4 loại cổng (Gates) dùng trong thế hệ x86, đó là:

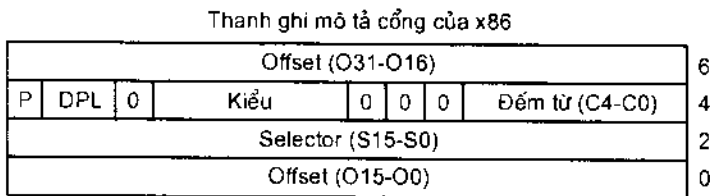
1. Cổng gọi: như đã trình bày trong 80286, cổng gọi phục vụ như là một trung gian giữa các đoạn mã (CS) có các mức thẩm quyền (PL) khác nhau. Cổng gọi dùng để thay đổi PL.

2. Cổng bẫy: được sử dụng để thực hiện chuyển đổi nhiệm vụ (task switching).

3. Cổng ngắt: được sử dụng để xác định các chương trình con phục vụ ngắt.

4. Cổng bẫy: được sử dụng để xác định các chương trình xử lý bẫy.

Cấu trúc của 4 loại thanh ghi mô tả cổng (Gate Descriptors) này trình bày trong hình 2.125.



Hình 2.125: Các thanh ghi mô tả cổng của 80386

Các cổng gọi (Call gates) được sử dụng trước tiên để chuyển điều khiển chương trình tới mức thẩm quyền cao hơn. Thanh ghi mô tả cổng gọi (Call gate Descriptor) gồm có 3 trường:

1. Byte các quyền truy cập (Byte Accesss Rights)

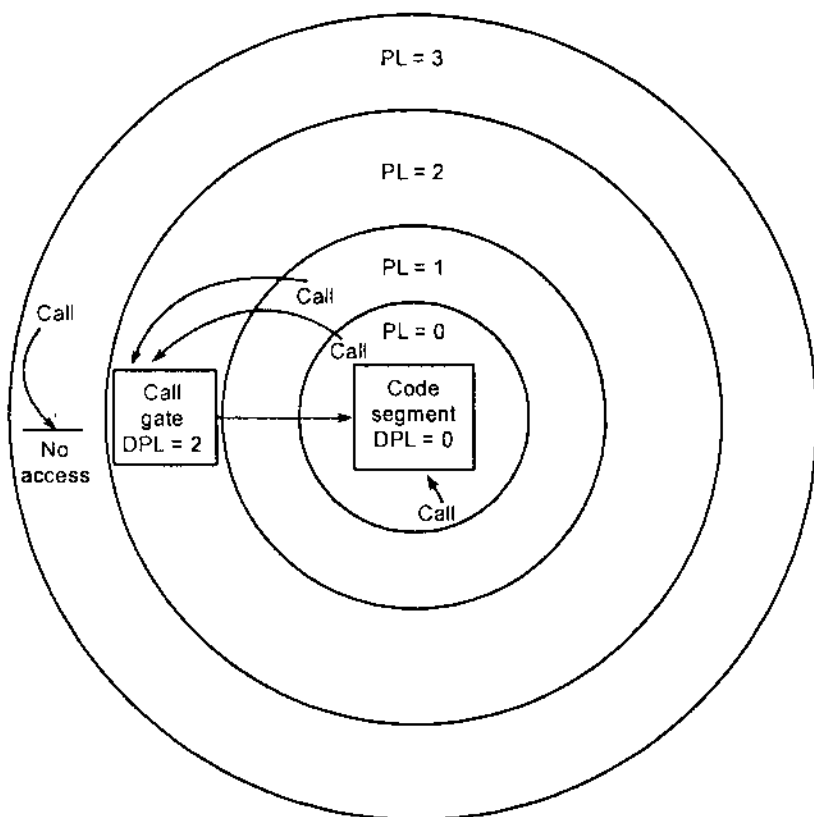
2. Con trỏ dài (long pointer): gồm 16-bit Selector và 32-bit Offset, trỏ tới địa chỉ đầu tiên của chương trình đích.

3. Số đếm từ (word count) gồm 5-bit: xác định bao nhiêu thông số được sao chép từ ngăn xếp của chương trình có lệnh gọi tới đến ngăn xếp của chương trình bị gọi tới. Trường này chỉ được dùng khi có sự thay đổi mức thẩm quyền. Những loại cổng khác không dùng tới trường này.

Có sự khác nhau giữa các cổng ngắt (Interrupt Gates) và cổng bẫy (Trap Gates) là cổng ngắt cấm ngắt (nó xóa bit IF trong EFLAGS), trong khi đó cổng bẫy không cấm ngắt. Các cổng gọi được truy cập

tới nhờ lệnh gọi (CALL) và về mặt cú pháp chúng tương tự như một lời gọi tới một thủ tục bình thường. Khi một lời gọi ở mức thẩm quyền nào đó được kích hoạt, thì xảy ra một quá trình như sau:

1. Nạp giá trị Selector và Offset từ cổng gọi vào CS:EIP (nghĩa là Selector nạp vào thanh ghi CS, còn Offset nạp vào con trỏ lệnh mở rộng (EIP), và kiểm tra tính đúng đắn.
2. Nội dung thanh ghi SS được đẩy vào ngăn xếp với mở rộng thêm các số 0 để đủ 32 bit.
3. Nội dung thanh ghi ESP được đẩy vào ngăn xếp.
4. Sao chép các thông số 32-bit (được đếm lượng số theo từ) từ ngăn xếp cũ đến ngăn xếp mới.
5. Đẩy giá trị địa chỉ trở về chương trình vào ngăn xếp.



Hình 2.126: Ví dụ về truy cập tới cổng gọi (Call gate)

Các cổng (gates) được truy cập từ một nhiệm vụ (task) nếu mức thẩm quyền EPL của nhiệm vụ đó bằng, hoặc lớn hơn mức thẩm quyền DPL trong thanh ghi mô tả cổng, tức là giá trị thẩm quyền của nhiệm vụ nhỏ hơn hoặc bằng giá trị DPL ghi trong thanh ghi mô tả cổng, và thỏa mãn bất đẳng thức sau:

$$(DPL \text{ đích}) \leq EPL = \max(RPL, CPL) \leq (\text{Gate DPL})$$

Trong đó, RPL: Requestor PL (mức thẩm quyền của yêu cầu), CPL: Current PL (mức thẩm quyền hiện hành) và $EPL = \max(RPL, CPL)$.

Ví dụ, một cổng gọi có DPL = 2 tham chiếu tới một đoạn mã có DPL = 0 (mức thẩm quyền cao nhất) (hình 2.126). Như vậy Call gate này có thể bị gọi bởi các lời gọi có các mức thẩm quyền 0, 1, và 2. Nó không thể bị gọi từ mức thẩm quyền = 3, bởi vì trong trường hợp này:

$$EPL = \max(RPL, CPL) > (\text{Gate DPL}).$$

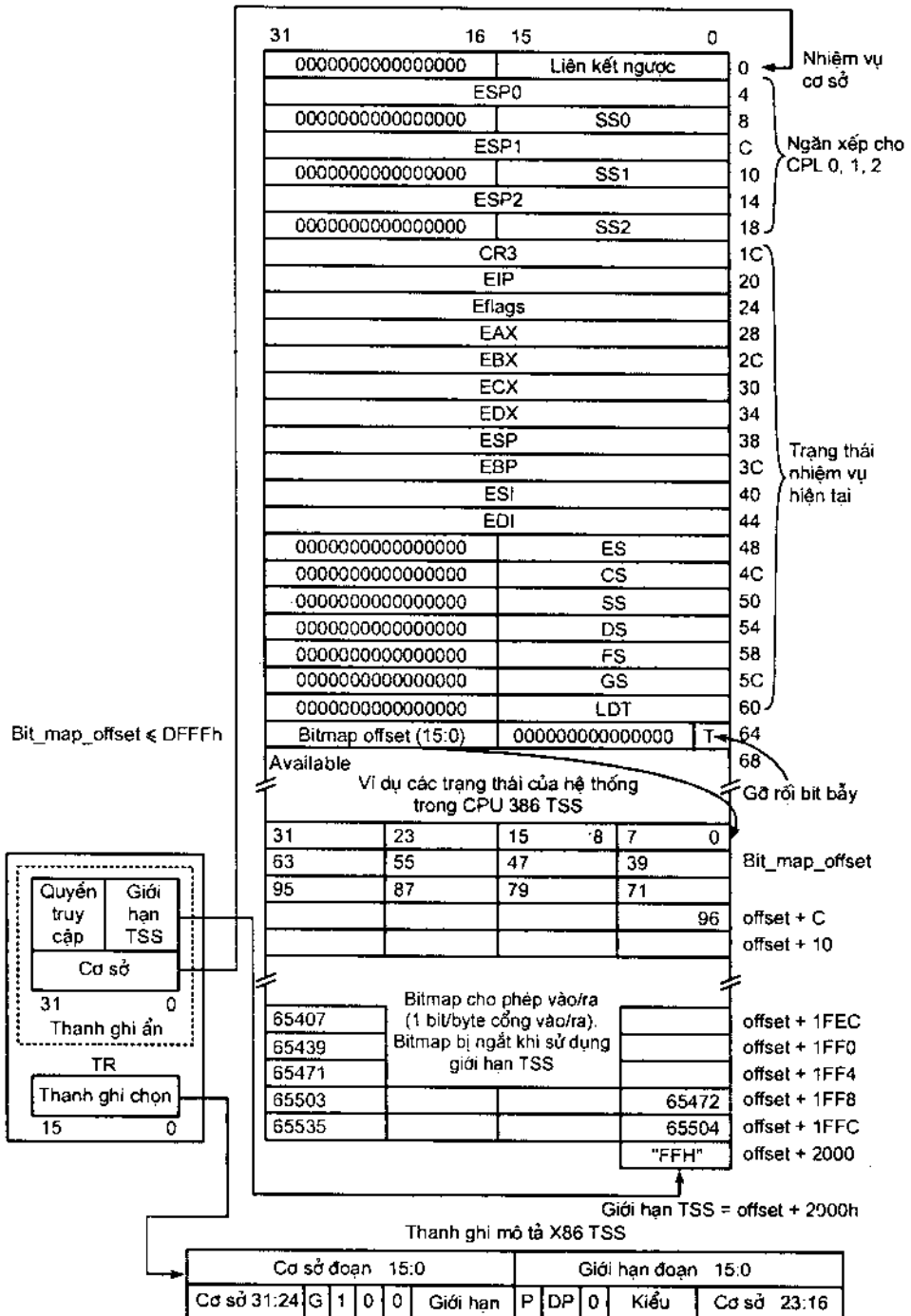
Người sử dụng có mức thẩm quyền PL = 3, người giám sát (supervisor) có các mức thẩm quyền PL = 0, 1, hoặc 2.

d) Quản lý nhiệm vụ (Task management)

Từ 80286 đến Pentium được thiết kế cho xử lý các nhiệm vụ (tasks) trong môi trường đa nhiệm. Một nhiệm vụ được xác định như là một quá trình cá biệt thực hiện một chương trình. Có một thuộc tính quan trọng của bất kỳ hệ điều hành đa nhiệm, đa sử dụng là khả năng chuyển đổi nhanh chóng giữa các nhiệm vụ. Các chip x86 hỗ trợ thao tác chuyển đổi nhiệm vụ bằng phần cứng. Thao tác chuyển đổi nhiệm vụ cất giữ toàn bộ trạng thái của CPU (nội dung của tất cả các thanh ghi, địa chỉ vùng nhớ, và một liên kết với nhiệm vụ trước đó), nạp trạng thái thực hiện mới, kiểm tra sự thực hiện bảo vệ, và thực hiện nhiệm vụ mới. Thao tác chuyển đổi nhiệm vụ kéo theo sự thực hiện lệnh nhảy vô điều kiện JMP hoặc lệnh gọi CALL giữa các đoạn khác nhau. Những lệnh này tham chiếu tới đoạn trạng thái nhiệm vụ TSS (Task State Segment) hoặc tham chiếu tới thanh ghi mô tả cổng nhiệm vụ TGD (Task Gate Descriptor) bên trong GDT hoặc LDT. Trong hầu

hết các trường hợp thường sử dụng CALL để khởi động nhiệm vụ mới. Lệnh ngắt INT n, bẫy (Trap), ngắt cứng từ bên ngoài, hay các ngoại lệ có thể cũng gây ra sự chuyển đổi nhiệm vụ nếu có một TGD trong thanh ghi mô tả IDT liên quan. Thanh ghi mô tả của TSS chứa thông tin về vị trí, kích thước và mức thẩm quyền của một TSS, giống như bất kỳ một thanh ghi mô tả nào. TSS không chứa dữ liệu hoặc chương trình, mà chứa toàn bộ trạng thái của nhiệm vụ và liên kết với các nhiệm vụ khác (hình 2.127). Một TGD chứa một thanh ghi chọn của TSS. Giá trị Limit của TSS phải lớn hơn 64h (100 decimal) byte và có thể bằng 4 GB. Mỗi một nhiệm vụ có một TSS liên quan. TSS hiện hành được xác định bằng một thanh ghi của CPU, đó là thanh ghi TR mà ta đã xét tới. 16-bit TR chứa thanh ghi chọn trỏ tới một thanh ghi mô tả của TSS (trong GDT), và thanh ghi mô tả trỏ tới TSS hiện hành. CPU cũng chứa một thanh ghi không xem (thanh ghi ẩn) được bằng chương trình, nhưng có quan hệ với TR, trong đó chứa các giá trị Base, TSS Limit, và Access Rights. Khi TR được nạp giá trị SELECTOR mới thì thanh ghi ẩn cũng được nạp giá trị tương ứng từ thanh ghi mô tả trong GDT của TSS.

Lệnh IRET thực hiện nhảy trở về từ nhiệm vụ. Khi lệnh IRET được thực hiện, điều khiển được trả về cho nhiệm vụ cũ đã bị ngắt. Trạng thái của nhiệm vụ đang thực hiện được cất giữ vào trong TSS và trạng thái của nhiệm vụ cũ bị ngắt được phục hồi trở lại từ TSS của nó. TSS chứa nhiều thông tin khác nhau. Từ đầu tiên của TSS là giá trị liên kết ngược (Back link). Đó là giá trị SELECTOR được sử dụng cho các lệnh nhảy trở về (RET hoặc IRET) nhiệm vụ cũ sau khi nạp giá trị Back link vào thanh ghi TR. Từ tiếp sau phải bằng 0. Các từ kép từ thứ 2 đến thứ 7 chứa các giá trị ESP và ESS cho các mức thẩm quyền 0-2. Từ thứ 8 (Offset 1Ch) là nội dung của CR3 chứa địa chỉ cơ sở cất giữ vào thanh ghi thư mục trang của nhiệm vụ cũ. Các nội dung của các từ kép tiếp theo được nạp vào các thanh ghi chỉ tên tương ứng. Khi một nhiệm vụ được truy cập tới, toàn bộ trạng thái của máy tính được cất giữ vào trong các vùng nhớ và sau đó được tải lại từ các vùng nhớ này vào TSS mới. Từ cuối cùng (Offset 66h) chứa địa chỉ cơ sở bit map cho phép vào/ra (I/O permission bit map base address).



Hình 2.127: Thanh ghi TTS và thanh ghi mô tả TSS

Một TGD tác động như là một điểm phối ghép giữa nhiệm vụ (hay chương trình của người sử dụng) và TSS. TGD chứa một thanh ghi chọn trỏ tới TSS và một byte Access Rights. Chương trình hiện hành với CPL và RPL của nó phải có đủ thẩm quyền để gọi tới cổng nhiệm vụ. Nguyên tắc thẩm quyền là: $\max(CPL, RPL) \leq (\text{task gate DPL})$

Có nghĩa là, chương trình gọi phải ở mức thẩm quyền ít nhất là như của cổng nhiệm vụ.

Bit T (debug trap bit, offset 64h) chỉ ra rằng CPU sẽ phải tạo ra một ngoại lệ để gỡ rối (debug) khi chuyển đổi nhiệm vụ. Nếu $T = 1$ thì khi vào nhiệm vụ mới một ngoại lệ gỡ rối sẽ được tạo ra.

Chuỗi sắp xếp các bit (Bitmap) cho phép vào/ra có độ dài 64 kbit (8 kbyte). Byte đầu tiên của bitmap cho phép vào/ra dành cho các cổng vào/ra 0000h-0007h. Chuỗi bitmap kéo dài cho đến cổng vào/ra địa chỉ FFFFh. Giá trị 0 của một bit trong bitmap cho phép một địa chỉ cổng vào/ra, giá trị 1 cấm địa chỉ cổng vào/ra. Quá trình chuyển đổi nhiệm vụ cần tới 17 μs để thực hiện và theo theo các bước như sau:

1. Toàn bộ trạng thái của nhiệm vụ hiện hành được cất giữ vào TSS dành riêng cho nó, mà TSS được trỏ bởi TR.

2. Các thanh ghi của CPU được nạp các giá trị từ TSS của nhiệm vụ mới (nhiệm vụ đích phải chuyển tới). Thanh ghi TR được nạp giá trị thanh ghi chọn cho TSS mới, và thanh ghi ẩn được nạp các nội dung từ thanh ghi mô tả của TSS mới trong GDT.

3. Thực hiện kiểm tra bảo vệ.

4. Bắt đầu thực hiện một nhiệm vụ mới.

Sự trở về nhiệm vụ cũ (nhiệm vụ gọi) sau khi thực hiện xong nhiệm vụ mới (nhiệm vụ bị gọi) được thực hiện theo từng bước như sau:

1. Trạng thái hiện hành của CPU được cất giữ vào trong TSS hiện hành.

2. Giá trị liên kết ngược (Selector) được nạp vào TR để truy cập đến TSS cũ để phục hồi trạng thái cũ của CPU. Sự trở về nhiệm vụ cũ được thực hiện nhờ lệnh IRET.

e) Chuyển đổi sang chế độ bảo vệ trong 80386

Để chuyển chế độ làm việc của 80386 từ chế độ thực sang chế độ bảo vệ, cần phải thực hiện một số bước. Chế độ thực được đưa vào ngay sau khi khởi động phần cứng của máy tính bằng RESET hoặc nhờ thiết lập bit PE = 0 trong thanh ghi CR0. Chế độ bảo vệ được thiết lập nhờ đặt bit PE = 1, nhưng trước đó, một số thứ phải được khởi tạo. Toàn bộ các bước nhằm chuyển từ chế độ thực sang chế độ bảo vệ trong 80386 như sau:

1. Khởi tạo IDT sao cho nó chứa các cổng ngắt cho ít nhất 32 số hiệu ngắt đầu tiên. IDT có thể chứa đến 256 số hiệu ngắt.

2. Khởi tạo GDT sao cho nó chứa một mô tả Null (Null Descriptor) ở thanh ghi mô tả 0, và các thanh ghi mô tả khác để cho ít nhất một đoạn mã, một đoạn ngăn xếp, một đoạn dữ liệu.

3. Chuyển đến chế độ bảo vệ bằng thiết lập bit PE = 1 trong CR0.

4. Thực hiện nhảy gần (JMP NEAR) bên trong đoạn để kích hoạt hàng xếp các lệnh bên trong và nạp giá trị thanh ghi mô tả TSS vào TR.

5. Nạp các giá trị lựa chọn (Selector) vào các thanh ghi đoạn tương ứng.

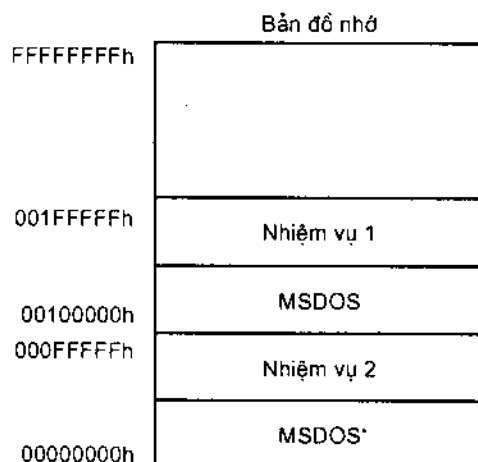
6. Bây giờ 80386 làm việc ở chế độ bảo vệ sử dụng các thanh ghi mô tả của các đoạn xác định trong GDT và IDT.

f) Chế độ 8086 ảo trong 80386

Một trong chế độ đặc biệt không có trong 80286 là chế độ 8086 ảo được bổ sung cho 80386. Chế độ này sử dụng để thực hiện đồng thời nhiều ứng dụng (tasks) trong chế độ thực 8086. Hình 2.128 là ví dụ 2 nhiệm vụ thực hiện trong chế độ 8086 ảo.

Nếu như hệ điều hành cho phép nhiều ứng dụng cùng thực hiện trong cùng thời gian thì thường đó là kỹ thuật phân mảnh thời gian (time-slicing). Tức là hệ điều hành phân phối một khoảng thời gian cho từng nhiệm vụ. Ví dụ, nếu có 3 nhiệm vụ được thực hiện thì hệ điều hành có thể phân phối khoản thời gian 1 ms cho từng nhiệm vụ.

Sau khi một nhiệm vụ thực hiện được 1 ms, phải xảy ra sự chuyển đổi nhiệm vụ, nghĩa là nhiệm vụ tiếp theo được thực hiện trong 1 ms. Sau 3 ms cả 3 nhiệm vụ được đưa vào thực hiện. Có thể điều chỉnh quãng thời gian gán cho từng nhiệm vụ. Kỹ thuật phân mảnh thời gian này có thể sử dụng cho hàng đợi in. Hàng đợi in có thể thực hiện trong vùng DOS và được dành khoảng 10% của thời gian hệ thống.



Hình 2.128: Ví dụ 2 nhiệm vụ thường trú trong bộ nhớ và thực hiện trong chế độ 8086 ảo của 80386

Sự khác nhau chính giữa chế độ bảo vệ và chế độ 8086 ảo của 80386 là: các thanh ghi đoạn trong chế độ 8086 ảo được sử dụng như trong chế độ thực 8086. Như vậy, khả năng đánh địa chỉ tối đa của chế độ 8086 cũng giống như chế độ 8086 thực, chỉ tối đa 1 MB (từ 00000h đến FFFFFh). Để chuyển vào chế độ 8086 ảo, cần phải thiết lập bit VM = 1 trong EFlags, và thông qua lệnh IRET nếu mức thẩm quyền là 0. Chế độ 8086 ảo có thể được sử dụng để chia sẻ một CPU với nhiều người sử dụng nhờ phân vùng nhớ, sao cho mỗi người sử dụng có một vùng nhớ của DOS riêng. Ví dụ, người sử dụng 1 được phân bổ vùng nhớ từ 00100000h đến 001FFFFFFh, người sử dụng 2: 00200000h - 002FFFFFFh,... Phần mềm của hệ thống chiếm vùng nhớ từ 00000000h đến 000FFFFFFh.

g) Cơ chế phân trang trong 80386

Cơ chế phân trang cho phép bất kỳ một địa chỉ tuyến tính nào (được tạo ra trong chương trình) được đặt trong một trang nhớ vật lý. Một trang nhớ tuyến tính là trang được địa chỉ bằng các giá trị Selectors và Offset trong chế độ thực và chế độ ảo. Một trang nhớ vật lý là trang nhớ tồn tại trong bộ nhớ vật lý. Ví dụ, vùng nhớ tuyến tính địa chỉ 20000h có thể được xếp vào vùng nhớ vật lý địa chỉ 30000h, hoặc bất kỳ địa chỉ vật lý nào khác. Nghĩa là, một lệnh truy cập tới địa chỉ tuyến tính 20000h thực chất là truy cập tới địa chỉ vật lý 30000h.

Mỗi một trang nhớ của 80386 có dung lượng 4 kbyte. Có 3 thành phần được sử dụng để chuyển đổi địa chỉ trang: thư mục trang (page directory), bảng trang (page table), và trang nhớ vật lý thực tế (actual physical memory page). Chúng ta xét từng thành phần này.

- *Thư mục trang (page directory)*: Thư mục trang chiếm một vùng nhớ gồm 1024 bảng chuyển đổi trang PTT (Page Translation Table). Mỗi một PTT chuyển đổi một địa chỉ logic thành địa chỉ vật lý. Thư mục trang được truy cập nhờ thanh ghi địa chỉ mô tả trang CR3 (page descriptor address register). CR3 chứa địa chỉ cơ sở của thư mục trang.

Một lệnh như: MOV CR3, reg được sử dụng để khởi tạo địa chỉ cơ sở của thư mục trang. Trong chế độ độ 8086 ảo, mỗi phần của DOS cho 8086 phải có một thư mục trang riêng.

Thư mục trang chứa tới 1024 điểm vào (entries), mỗi điểm vào có độ dài 4 byte. Bản thân một thư mục trang chiếm một trang nhớ dung lượng 4 kbyte. Mỗi một điểm vào trong thư mục trang chuyển đổi 10 bit cao nhất của địa chỉ bộ nhớ. 10 bit này của địa chỉ tuyến tính được sử dụng để xác định các bảng trang khác nhau cho các điểm vào bảng trang khác nhau. Địa chỉ bảng trang (page table address) A32-A2, cất giữ trong điểm vào thư mục trang, truy cập một bảng chuyển đổi trang (PTT) độ dài 4 kbyte. Để chuyển đổi hoàn toàn bất kỳ một địa chỉ tuyến tính nào thành một địa chỉ vật lý cần phải có 1024 bảng trang, độ dài mỗi bảng 4 kbyte, cộng với thư mục bảng

trang, độ dài mỗi thư mục 4 kbyte. Sơ đồ chuyển đổi địa chỉ này cần đến 4 MB và thêm 4 kbyte nhớ để chuyển đổi hoàn chỉnh bất kỳ địa chỉ tuyến tính nào thành địa chỉ vật lý. Chỉ có các hệ điều hành lớn mới hỗ trợ kích thước này. Nhiều hệ điều hành thông dụng thực hiện chuyển đổi chỉ 16 MB hoặc 32 MB của hệ thống nhớ nếu có định trang. Ví dụ, Windows 3.1.

Các bit điều khiển của điểm vào của thư mục bảng trang có những chức năng như sau (hình 2.129):

31	12	11	10	9	8	7	6	5	4	3	2	1	0
Địa chỉ bảng trang (Page table address)				Dự phòng	0	0	D	A	0	0	U/S	R/W	P

Hình 2.129: Điểm vào thư mục bảng trang (page table directory entry)

1. D-Dirty: được xác định cho các điểm vào thư mục bảng trang và dùng cho hệ điều hành.

2. A-Accessed: được thiết lập = 1 khi vi xử lý truy cập điểm vào thư mục trang.

3. R/W và U/S (Read/Write và User/Supervisor): được sử dụng trong sơ đồ bảo vệ liệt kê dưới đây:

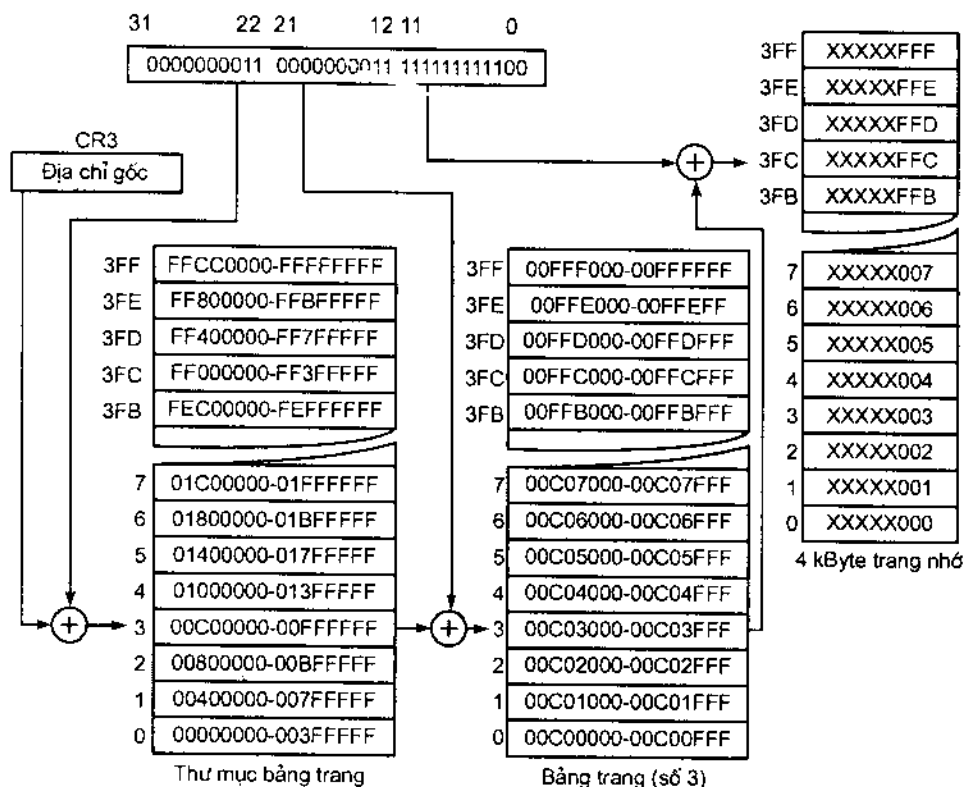
U/S	R/W	Truy cập mức 3
0	0	Không
0	1	Không
1	0	Chỉ đọc
1	1	Đọc/ghi

4. P-Present: nếu P = 1, thì nó chỉ rằng, điểm vào có thể được sử dụng trong chuyển đổi địa chỉ. Nếu P = 0, thì điểm vào không thể được sử dụng để chuyển đổi, và các bit còn lại của điểm vào có thể được sử dụng để chỉ ra vị trí của trang trên bộ nhớ đĩa.

• *Bảng trang (Page table)*: Bảng trang chứa 1024 địa chỉ vật lý được truy cập để chuyển đổi địa chỉ tuyến tính thành địa chỉ vật lý. Khuôn dạng của điểm vào bảng trang giống như điểm vào thư mục trang (hình 2.129). Sự khác nhau chính là điểm vào thư mục trang

chứa địa chỉ vật lý của một bảng trang, còn điểm vào bảng trang chứa địa chỉ vật lý của trang nhớ dung lượng 4 kbyte. Sự khác nhau nữa là bit D không có chức năng trong điểm vào thư mục trang, nhưng lại chỉ ra một trang được ghi vào điểm vào bảng trang.

Hình 2.130 mô tả cơ chế định trang trong 80386, trong đó, địa chỉ tuyến tính 00C03FFCh của chương trình được chuyển đổi thành địa chỉ vật lý XXXXX3FCh. (XXXXX là bất kỳ địa chỉ trang nhớ vật lý nào dung lượng 4 kbyte).



Hình 2.130: Chuyển đổi địa chỉ tuyến tính 00C03FFCh thành địa chỉ vật lý XXXXXFFCh, trong đó, giá trị XXXXX được xác định nhờ điểm vào bảng trang

Các chức năng của cơ chế định trang thực hiện như sau:

1. Thư mục trang dung lượng 4 kbyte được cất giữ như là địa chỉ vật lý trong CR3. Địa chỉ này thường được gọi là địa chỉ gốc (root)

address). Trong chế độ 8086 ảo, mỗi nhiệm vụ có một thư mục trang riêng cho phép các vùng khác nhau của bộ nhớ vật lý được gán cho các nhiệm vụ khác nhau.

2. 10 bit trên của địa chỉ tuyến tính (31-22) dùng để chọn một điểm vào trong thư mục trang.

3. Bảng trang được địa chỉ nhờ điểm vào cất trong thư mục trang. Có tới 4 kbyte bảng trang trong hệ thống chuyển đổi.

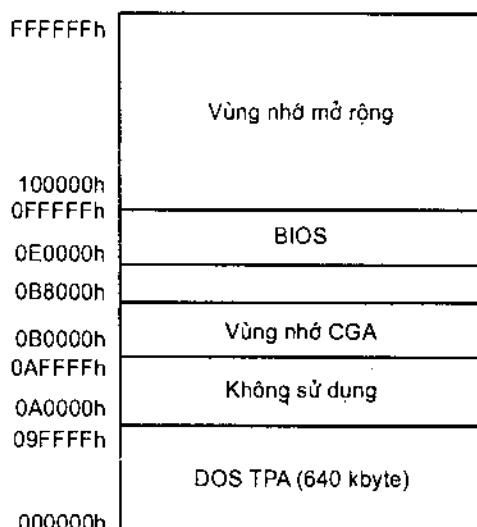
4. Một điểm vào trong bảng trang được địa chỉ nhờ 10 bit tiếp theo của địa chỉ tuyến tính (21-12).

5. Điểm vào bảng trang chứa địa chỉ vật lý thực tế của trang nhớ dung lượng 4 kbyte.

12 bit bên phải nhất (các bit thấp nhất) của địa chỉ tuyến tính (bit 11-0) chọn một vùng trong trang nhớ.

Cơ chế phân trang cho phép bộ nhớ vật lý được gán cho bất kỳ địa chỉ tuyến tính nào. Ví dụ, cho rằng, địa chỉ vật lý 20000000h được chọn nhờ một chương trình, nhưng ngăn nhớ này không tồn tại trong bộ nhớ vật lý. Khi đó, một trang nhớ 4 kbyte được chương trình tham chiếu đến như một vùng nhớ có dải địa chỉ 20000000h-20000FFFh. Bởi vì vùng nhớ này không tồn tại thực trong bộ nhớ vật lý, nên hệ điều hành có thể ấn định một trang nhớ vật lý tồn tại thực tế, ví dụ, vùng 12000000h-12000FFFh cho vùng địa chỉ tuyến tính trên.

Trong quá trình chuyển đổi địa chỉ, 10 bit cao nhất của địa chỉ tuyến tính cho điểm vào thư mục trang 200h nằm ở địa chỉ 800h trong thư mục trang. Điểm vào trang này chứa địa chỉ của bảng trang cho các địa chỉ tuyến tính 20000000h-203FFFFFFh. Các bit của địa chỉ tuyến tính (21-12) chọn điểm vào trong bảng trang này tương ứng với trang nhớ 4 kbyte. Đối với các địa chỉ tuyến tính 20000000h-20000FFFh, điểm vào đầu tiên (entry 0) trong bảng trang được chọn. Điểm vào đầu tiên này chứa địa chỉ vật lý của trang nhớ hiện thực hoặc 12000000h-12000FFFh trong ví dụ này.



Hình 2.131: Sắp xếp bộ nhớ chính trong hệ thống máy tính dùng các hệ điều hành dựa trên DOS

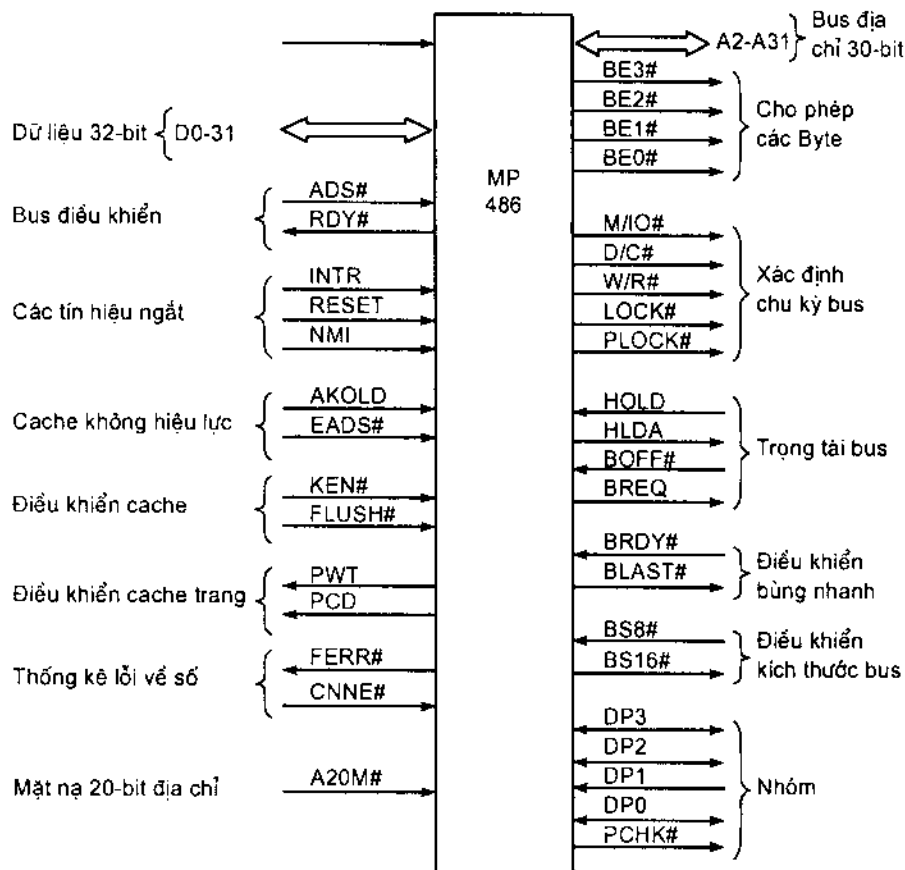
Lấy ví dụ, hệ điều hành DOS. Sắp xếp bộ nhớ của hệ thống như trong hình 2.131). Bình thường, bộ nhớ chính của máy tính dùng hệ điều hành dựa trên DOS bắt đầu từ địa chỉ 000000h và mở rộng đến 09FFFFh, đúng bằng 640 kbyte. Tiếp theo là các vùng nhớ dành cho các bảng điều khiển màn hình, điều khiển đĩa, và ROM BIOS. Trong ví dụ này, vùng nhớ ngay trên 09FFFFh không sử dụng (0A0000h-0AFFFFh). Vùng này có thể được DOS sử dụng. Như vậy, tổng dung lượng dùng cho DOS là 704 kbyte thay cho 640 kbyte. Vùng nhớ này cũng có thể được sắp xếp vào vùng nhớ mở rộng 102000h-11FFFFh. Để thực hiện chuyển đổi địa chỉ và khởi tạo thư mục trang và các bảng trang cần phải khởi tạo bộ nhớ với việc sử dụng thanh ghi CR3.

2.4.8. Vi xử lý 80486

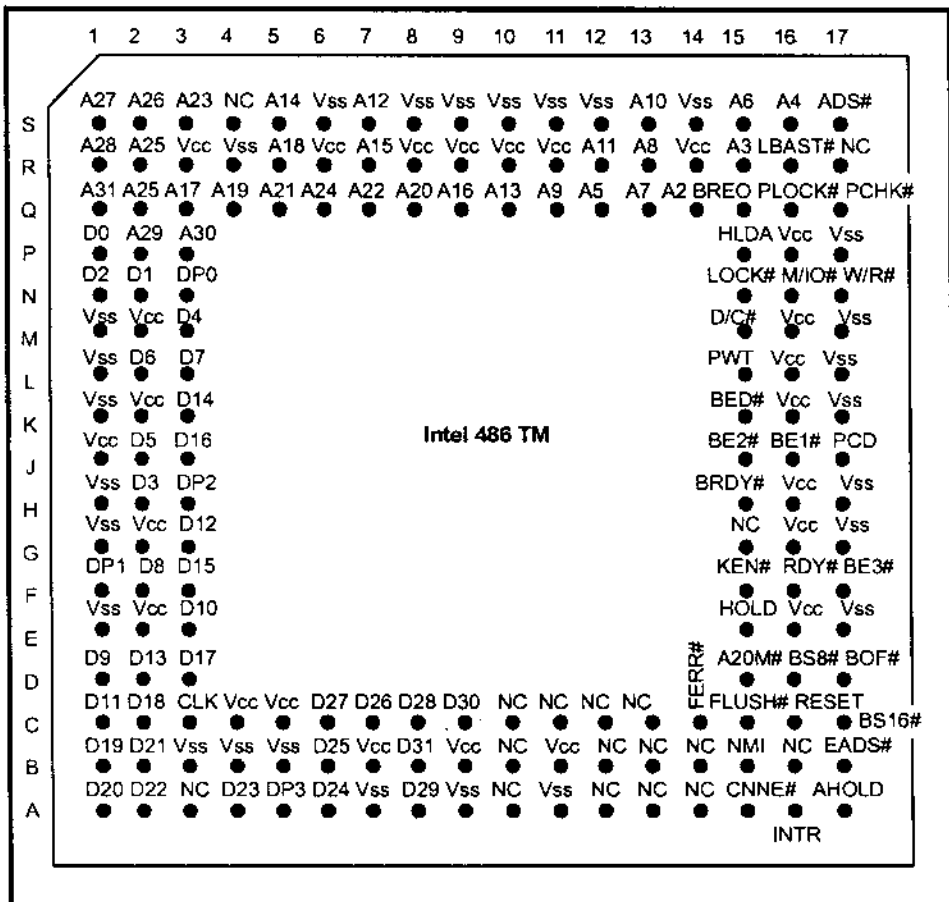
(1) Đóng vỏ và các chân tín hiệu của 80486

Cả 80486DX và 80486SX đều đóng gói PGA 168 chân. 80486 đóng vỏ GAP 168 chân công nghệ CHMOS IV, có tới 1,2 triệu transistor, là vi xử lý 32-bit, tốc độ cao hơn nhiều so với 80386, có tập lệnh tối thiểu RISC (Reduced Instruction Set) và tập lệnh mở rộng, có

sự bức xạ tối thiểu tần số vô tuyến (Reduced RF emissions) và tự kiểm tra khi bật nguồn, có các mức bảo vệ bộ nhớ ảo. Đặc biệt có 8 kbyte L1 cache bên trong (chung dữ liệu và lệnh). Cả 80386 và 80486 có thể đánh địa chỉ vật lý tới 4 GB nhớ ($2^{32} = 2^2 \cdot 2^{30}$) vật lý và tới 64 TB ($2^{46} = 2^{16} \cdot 2^{30}$) nhớ ảo. Tốc độ thực hiện lệnh của 80486 lên tới hơn 25 MIPS. Có 8 kbyte cache nhớ bên trong 80486. Có các đơn vị nguyên hiệu suất cao, có đơn vị xử lý dấu phẩy động bên trong (FPU), có cơ chế phân đoạn (segmentation) nhớ, phân trang, có kỹ thuật đường ống lệnh.



Hình 2.132: Các nhóm tín hiệu của 80486



Hình 2.133: Vị trí các chân tín hiệu của chip Intel 80486DX

Khi kết nối 80486, tất cả các chân Vcc và Vss đều phải được nối vào nguồn cung cấp $5,0\text{ V} \pm 10\%$ với dòng tới 1,2 A cho phiên bản 33 MHz. Dòng cung cấp trung bình là 650 mA cho phiên bản 33 MHz. Các tín hiệu ra ở mức logic 0 cho dòng 4,9 mA và ở mức logic 1 cho dòng 1,0 mA. Nếu muốn có kết nối nhiều tải hơn thì 80486 cần phải được kết nối với các bộ đệm. Thực tế, mang tải nhiều là các bus địa chỉ, dữ liệu, và các tín hiệu kiểm tra chẵn lẻ (parity signals), do đó chúng được kết nối với các bộ đệm. Đặc biệt, chip 80486 có 24 chân nối nguồn (Vcc, +5V), 28 chân nối đất (Vss) và 17 chân không dùng tới trong tổng số 168 chân. Chỉ những đường dây địa chỉ từ A31 đến A2 được đưa ra ngoài để tạo các địa chỉ từ kép (dword addresses).

Các chương trình ứng dụng viết trên 8088/8086 có thể chạy được trên hệ thống 80386, 80486 mà không cần phải thay đổi gì cả. 80386 là sự nâng cấp của 80386.

80486 có 2 phiên bản: phiên bản tốc độ cao DX với FPU bên trong, và phiên bản tốc độ thấp SX không có FPU bên trong mà phải dùng đồng xử lý bên ngoài. Có phiên bản tốc độ cao hơn hẳn, đó là phiên bản ECL (Emitter-Coupled-Logic) chạy với tốc độ 120 MIPS. Hình 2.132 là sơ đồ tín hiệu và hình 2.133 là vị trí chân tín hiệu trên chip của 80486DX. Có một số ít sự khác nhau về các chân tín hiệu giữa 80486DX và 80486SX:

- Tín hiệu NMI: chân B15 trên 80486DX, chân A15 trên 80486SX,
- Tín hiệu IGNNE#: chân A15 trên 80486DX, không có trên 80486SX,
- Tín hiệu FERR#: chân C14 trên 80486DX, chân B15 và C14 trên 80486SX không nối.

• *Các chức năng của các chân tín hiệu của 80486*

1. A31-A2 (các đường ra địa chỉ): cung cấp địa chỉ chọn bộ nhớ và các cổng vào/ra. A31-A4 dùng để điều khiển bộ vi xử lý.

2. A20M# (Address bit 20 Mask): mặt nạ bit địa chỉ 20, được dùng để ràng buộc 80486 địa chỉ ở vùng nhớ từ 000FFFFh đến 00000000h giống như 8086. Điều này để tạo ra bộ nhớ có dung lượng chỉ đến 1 MB. Hầu hết các hệ thống không sử dụng tín hiệu này vì chương trình HIMEM.SYS không thể truy cập tới vùng nhớ bổ sung ở địa chỉ 100000h-10FFEFh.

3. ADS (Address Data Strobe): trở thành mức logic 0 để chỉ ra rằng bus địa chỉ đã có một giá trị địa chỉ.

4. AHOLD (Address HOLD input): tín hiệu này làm 80486 đặt các đường dây địa chỉ của nó lên mức trở kháng cao, trong khi các bộ đệm vẫn ở trạng thái tích cực. Tín hiệu này thường được các thiết bị

làm chủ hệ thống bus sử dụng để có được truy cập cho chu trình làm mất hiệu lực cache.

5. BE3#-BE0# (Byte Enable outputs): các tín hiệu này dùng để chọn bằng nhớ, trong đó BE3# cho phép byte D31-D24, BE2# cho phép byte D23-D16, BE1# cho phép byte D15-D8, và BE0# cho phép byte D7-D0.

6. BLAST# (Burst last output): chỉ rằng chu trình bùng nhanh của bus được hoàn thành theo lần tích cực tiếp theo của tín hiệu BRDY#.

7. BOFF# (Backoff input): làm cho 80486 đặt các bus của nó lên trạng thái trở kháng cao trong chu kỳ nhịp đồng hồ tiếp theo. 80486 duy trì trong trạng thái giữ bus (bus hold state) cho đến khi BOFF# được đưa lên mức logic 1.

8. BRDY# (Burst ready input): được sử dụng để thông báo cho 80486 rằng chu trình bùng nhanh đã hoàn thành.

9. BREQ (Bus REQuest input): là tín hiệu cho 80486 biết rằng đã có phát sinh một yêu cầu của bus bên trong.

10. BS8# (Bus Size 8): để chỉ cho 80486 tự kiến trúc lại bus dữ liệu thành bus dữ liệu 8-bit phục vụ truy cập bộ nhớ và ngoại vi theo byte.

11. BS16# (Bus Size 16): để chỉ cho 80486 tự kiến trúc lại bus dữ liệu thành bus dữ liệu 16-bit phục vụ truy cập bộ nhớ và ngoại vi theo từ.

12. CLK (Clock input): cung cấp cho 80486 nhịp đồng hồ thời gian cơ bản. Tín hiệu này có mức TTL.

13. D31-D0 (Data bus): để là môi trường vận chuyển dữ liệu giữa CPU và bộ nhớ, thiết bị vào/ra. D7-D0 cũng dùng để tiếp nhận véctơ ngắt khi thực hiện xử lý ngắt.

14. D/C# (Data/Control#): chỉ ra rằng thao tác hiện hành là chu trình vận chuyển dữ liệu hoặc điều khiển. Bảng 2.13 liệt kê các tổ hợp D/C#, M/IO# và W/R# tạo ra các chu trình bus khác nhau như thế nào.

15. DP3-DP0 (Data Parity I/O): đảm bảo các dữ liệu thỏa mãn parity chẵn cho các thao tác ghi dữ liệu và kiểm tra chẵn lẻ cho các thao tác đọc dữ liệu. Nếu có lỗi chẵn lẻ trong chu trình đọc dữ liệu, thì đầu ra PCHK# trở thành mức logic 0. Nếu không sử dụng kiểm tra chẵn lẻ thì các đường dây tín hiệu này phải được nối điện thế +5,0 V.

Bảng 2.13: Các tổ hợp tạo các chu trình bus

M/I/O#	D/C#	W/R#	Chu trình bus (Bus cycle)
0	0	0	Chấp nhận ngắt (Interrupt acknowledge)
0	0	1	Dừng (Halt/special)
0	1	0	Đọc từ thiết bị vào/ra (I/O read)
0	1	1	Ghi ra thiết bị vào/ra (I/O write)
1	0	0	Đọc lệnh (Opcode fetch- chu trình lệnh)
1	0	1	Dự trữ
1	1	0	Đọc từ bộ nhớ (Memory read)
1	1	1	Ghi ra bộ nhớ (Memory write)

16. EADS# (External Address strobe input): được sử dụng cùng với AHOLD để chỉ rằng một địa chỉ ngoài được sử dụng để thực hiện chu trình làm mất hiệu lực cache.

17. FERR# (Floating-point ERRor output): chỉ ra rằng FPU đã có xác nhận lỗi. Tín hiệu này được sử dụng để bảo đảm sự tương thích với các phần mềm của DOS.

18. FLUSH# (Cache flush input): tín hiệu này làm 80486 xóa nhanh chóng toàn bộ nội dung của 8 kbyte cache bên trong chip.

19. HLDA (Hold Acknowledge output): chỉ rằng tín hiệu vào HOLD được tích cực và 80486 đã đặt các bus của nó lên trạng thái trở kháng cao.

20. HOLD (Hold input): được sử dụng để yêu cầu một tác động DMA. Nó gây ra việc các bus địa chỉ, dữ liệu và điều khiển được đặt vào trạng thái trở kháng cao và sau đó HLDA trở về mức logic 0.

21. IGNE# (IGNore Numeric Error input): tín hiệu này làm cho FPU bỏ qua lỗi số dấu phẩy động và tiếp tục xử lý dữ liệu. Tín hiệu này không ảnh hưởng đến trạng thái chân tín hiệu FERR#.

22. INTR (INTerrupt Request input): là tín hiệu yêu cầu 80486 phục vụ một yêu cầu ngắt từ một thiết bị ngoại vi.

23. KEN# (Cache ENable input): có tác dụng làm cho bus hiện hành kết nối với cache bên trong để ghi dữ liệu.

24. LOCK# (LOCK output): ở mức logic 0 cho bất kỳ một lệnh nào có tiền tố khóa.

25. M/IO# (Memory/IO#): xác định khi nào bus địa chỉ chứa địa chỉ của bộ nhớ hay số hiệu của cổng vào/ra. Tín hiệu này kết hợp với W/R# để tạo ra các tín hiệu điều khiển ghi/đọc bộ nhớ và thiết bị vào/ra.

26. NMI (Non-Maskable Interrupt input): là yêu cầu ngắt không che được (ngắt số 2). Bit cờ IF không có tác dụng (không che được).

27. PCD (Page Cache Disable output): phản ánh trạng thái của bit thuộc tính PCD trong bảng vào trang (PTE) hoặc bảng vào thư mục trang (PDE).

Bảng 2.14: Trạng thái của 80486 sau khi RESET

Thanh ghi	Giá trị ban đầu có Tự kiểm tra (Self-test)	Giá trị ban đầu không có Tự kiểm tra (Self-test)
EAX	00000000h	?
EDX	00000400h+ID	00000400h+ID
EFLAGS	00000002h	00000002h
EIP	0000FFF0h	0000FFF0h
ES	0000h	0000h
CS	F000h	F000h
DS	0000h	0000h
SS	0000h	0000h
FS	0000h	0000h
GS	0000h	0000h
IDTR	Base = 0, limit = 3FFh	Base = 0, limit = 3FFh
CR0	60000010h	60000010h
DR7	00000000h	00000000h

28. PCHK# (Parity CHecK output): chỉ ra rằng có một lỗi chẵn lẻ được xác nhận khi đọc dữ liệu thông qua các chân DP3-DP0.

29. PLOCK# (Pseudo-LOCK output): chỉ ra rằng thao tác hiện hành yêu cầu thêm một chu trình bus để thực hiện. Tín hiệu này bằng 0 khi thực hiện các phép toán của PFU, vì những phép toán này cần tới dữ liệu 64-bit hoặc 80-bit.

30. PWT (Page Write Through output): chỉ ra trạng thái của bit thuộc tính PWT trong bảng vào trang (PTE) hoặc trong bảng vào thư mục trang (PDE).

31. RDY# (ReaDY input): chỉ ra rằng một chu trình không bùng nhanh đã hoàn thành. Tín hiệu này phục vụ cho sự điều khiển trạng thái chờ đợi.

32. RESET (RESET input): khởi tạo 80486 về trạng thái ban đầu. Bảng 2.14 cho biết trạng thái ban đầu của 80486 sau khi RESET.

33. W/R# (Write/Read#): chu kỳ bus hiện hành là chu kỳ ghi hay đọc.

(2) Các khối chức năng chính trong 80486

1. Giao tiếp bus (bus interface), nó kết nối với bus hệ thống bên ngoài, cache bên trong chip và các đơn vị tiền đọc lệnh (prefetcher units).

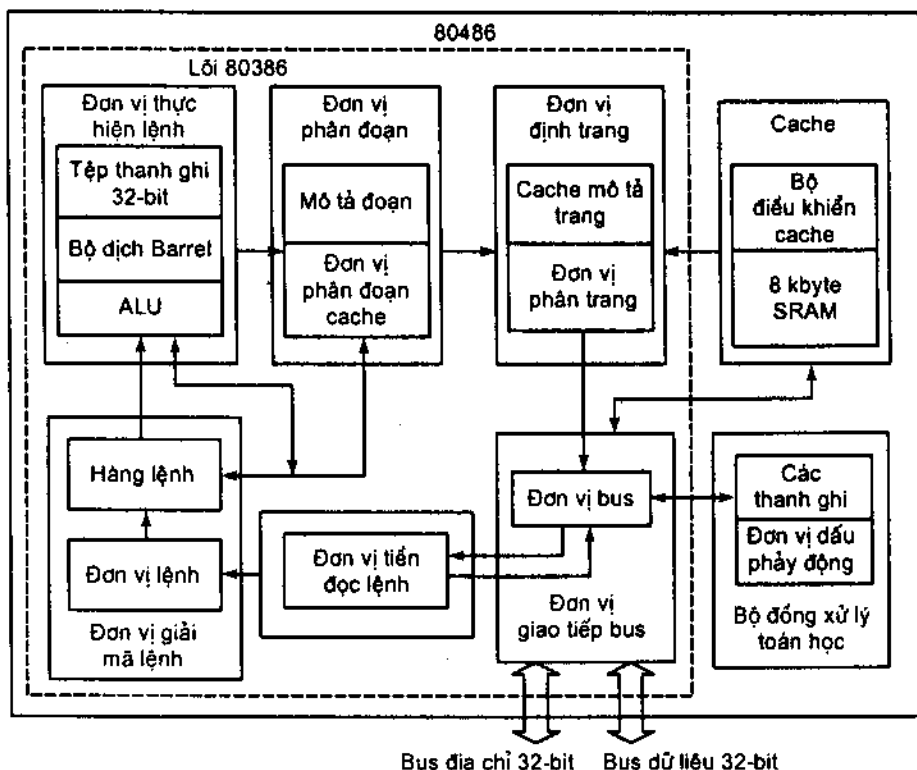
2. Đơn vị tiền đọc lệnh (Code prefetch unit) với hàng lệnh dài 32-byte được nối với giao tiếp bus, cache, giải mã lệnh và đơn vị phân đoạn (segmentation unit)

3. Đơn vị cache (cache unit), gồm có 8 kbyte cache (chứa dữ liệu và lệnh) và logic quản lý cache. Nó được kết nối với đơn vị phân đoạn, ALU và FPU thông qua bus 64-bit truyền dữ liệu giữa các khối (interunit data transfer bus). Đơn vị cache cũng kết nối trực tiếp với đơn vị định trang (paging unit), giao tiếp bus, bộ tiền đọc lệnh.

4. Đơn vị giải mã lệnh, nhận 3 byte của lệnh đọc từ hàng lệnh và chuyển lệnh được giải mã tới đơn vị điều khiển và kiểm tra bảo vệ CPTU (Control and Protection Test Unit). CPTU tạo ra các vi lệnh và gửi chúng tới các đơn vị khác để thực hiện kiểm tra bảo vệ.

9. Đơn vị định trang (paging unit), trong đó có logic quản lý phân trang và bộ đếm chuyển đổi 32 điểm vào bộ đếm chuyển đổi riêng TLB (Translation Lookup Buffer).

Trong sơ đồ khối của 80486 ở hình 2.134 có thể nhận ra được lõi (core) của 80486 là 80386. 80386 là công nghệ CHMOS III, đóng gói 132 chân, là vi xử lý 32-bit có cấu trúc tương tự như 80486. Nhưng 80386 không có đồng xử lý bên trong mà dùng đồng xử lý bên ngoài, chip 80387. Đơn vị FPU có tập các thanh ghi và tập các lệnh giống như của 80387. Dĩ nhiên tổ chức và công nghệ của FPU trong 80486 khác so với 80387. Hàng lệnh của 80486 có độ dài 32 byte, trong khi đó của 80386 có độ dài 16 byte.



Hình 2.135: Sơ đồ khối 80486 với lõi là 80386

80486 và 80386 có mô hình lập trình giống nhau và cũng có các thanh ghi mô tả bảng cục bộ LDTR, mô tả bảng toàn cục GDTR, và

mô tả bảng ngắt IDTR. Những thanh ghi này đã được trình bày trong 80386. Chức năng của MMU và đơn vị định trang của 80486 cũng giống như của 80386 đã được trình bày.

Tất cả các đơn vị bên trong 80486 được sắp xếp trong cấu trúc đường ống để có thể thực hiện song song các chức năng. Khi một lệnh được thực hiện thì một lệnh khác được giải mã, lệnh thứ ba được đọc từ bộ nhớ. Để giải thích rõ cơ chế xử lý song song theo cấu trúc đường ống ta hãy lấy ví dụ sau đây bằng thực hiện lệnh:

ADD BIEN, BX; BIEN là biến dữ liệu cộng với nội dung thanh ghi BX.

1. Cho rằng vùng nhớ có chứa lệnh ADD trên, và vùng nhớ chứa có chứa giá trị BIEN là phần được sử dụng lặp đi lặp lại nhiều lần, do đó, chúng được chuyển vào cache 8 kbyte bên trong vi xử lý 80486 nhờ tác động của phần mềm quản lý bộ nhớ.

2. Trong khi thực hiện lệnh lần đọc trước, đơn vị cache (cache unit) xem xét “trước” và chỉ thị đọc lệnh tiếp theo (ADD BIEN, BX). Nếu như lệnh tiếp theo này không có trong cache thì hệ thống quản lý mạng phải tự động khởi động quá trình đọc lệnh này từ bộ nhớ RAM (hay bộ nhớ ngoài) vào cache bên trong.

3. Đơn vị cache gửi địa chỉ logic của lệnh tiếp theo này tới đơn vị phân trang (paging unit) để tại đó, địa chỉ trang vật lý của lệnh được tạo ra.

4. Sử dụng địa chỉ trang, lệnh ADD BIEN, BX được đọc nhờ đơn vị giao tiếp bus (bus interface unit) và xếp vào hàng lệnh (prefetch queue) của đơn vị tiền đọc mã lệnh (Code prefetch unit).

5. Các lệnh máy xếp hàng trước đó được đẩy khỏi hàng lệnh tới đơn vị giải mã lệnh (instruction decode unit) để giải mã và tiếp theo tới đơn vị thực hiện lệnh (excution unit). Tiếp tới là lệnh ADD BIEN, BX được đẩy tới đơn vị giải mã lệnh để giải mã và sau đó được thực hiện.

6. Trong quá trình thực hiện, nội dung của thanh ghi BX và giá trị BIEN được đưa tới thực hiện.

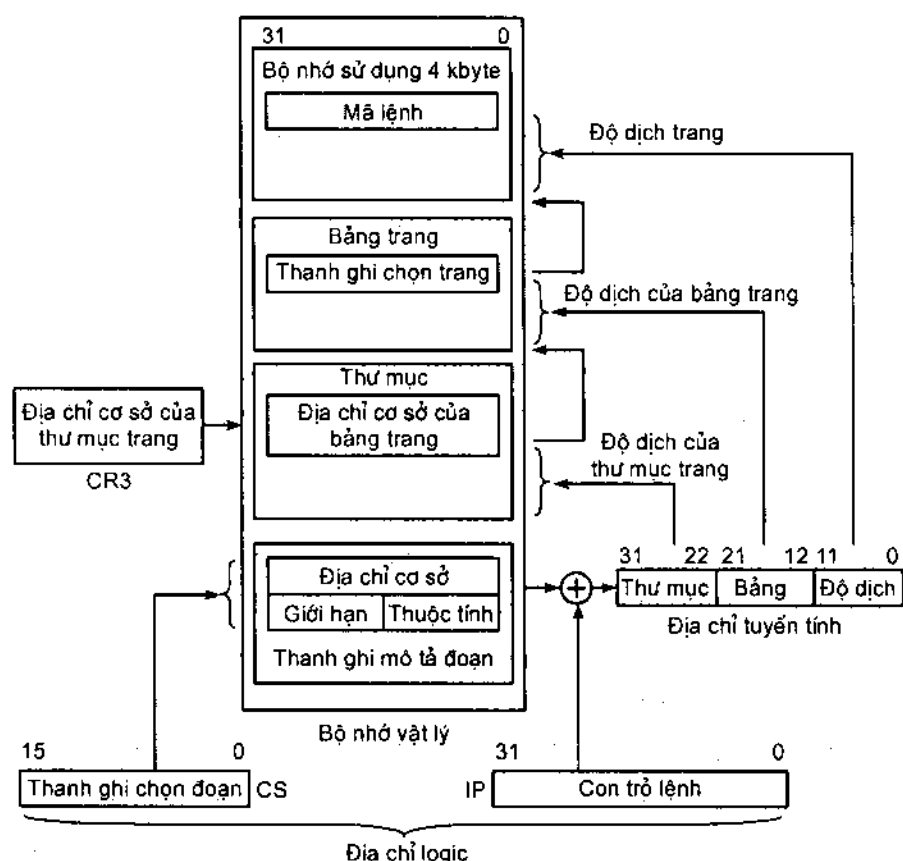
7. Kết quả của phép cộng được chuyển trở lại cache ở chỗ vị trí BIEN.

8. Nếu quá trình xử lý tiếp tục với chương trình chứa lệnh ADD BIEN, BX và giá trị BIEN thì lệnh và BIEN được chuyển từ cache trở lại RAM hay bộ nhớ ngoài.

(3) Quản lý bộ nhớ của 80486

a) Phân đoạn/trang nhớ

Đơn vị quản lý bộ nhớ MMU bên trong 80486 thực hiện chức năng quản lý bộ nhớ tương tự như 80386. Nó có các đơn vị chức năng định trang (paging unit) và định đoạn nhớ (segmentation unit) (hình 2.136).



Hình 2.136: Chuyển đổi địa chỉ theo định đoạn/trang của 80486

Đơn vị định đoạn đảm bảo các đoạn nhớ để đối chỗ chứa dữ liệu (swapping segments) có thể có dung lượng đến 4 GB.

- Khi sử dụng định trang, mỗi đoạn nhớ có thể chia thành các đoạn con (subsegment) dung lượng 4 kbyte. Như vậy quá trình quản lý bộ nhớ liên quan đến các khối nhớ dung lượng 4 kbyte. Định trang và định đoạn có thể thực hiện độc lập với nhau, nhưng khi kết hợp lại, chúng tạo ra một hệ thống nhớ ảo trong chế độ bảo vệ dung lượng lớn. Bộ nhớ ảo, như đã trình bày, có trong chế độ bảo vệ của quản lý bộ nhớ, là kỹ thuật cho phép các chương trình rất lớn có thể thực hiện trong vùng nhớ vật lý nhỏ (RAM hay cache).

Ta xét tuần tự quá trình xác định địa chỉ bắt đầu từ đọc lệnh từ bộ nhớ:

- Quá trình chuyển đổi địa chỉ bắt đầu bằng xác định thanh ghi chọn đoạn 16-bit (segment selector register) và giá trị offset 16-bit. Đây là hai thành phần của địa chỉ logic. Đối với đọc lệnh, chúng ta mặc định thanh ghi chọn đoạn mã (CS) để xác định phần dành cho đoạn của địa chỉ và con trỏ lệnh IP để chứa offset.

- Thanh ghi chọn đoạn mã (Code segment selector) trở tới thanh ghi mô tả đoạn mã (code segment descriptor) trong bộ nhớ.

- Địa chỉ cơ sở 32-bit bên trong mô tả đoạn được cộng với offset 32-bit (nội dung của IP) để tạo ra địa chỉ tuyến tính 32-bit (linear address). Địa chỉ tuyến tính gồm 3 offset thành phần: offset thư mục, offset bảng và offset vật lý.

- Thanh ghi CR3 trở tới cơ sở của thư mục trang. Phần offset thư mục của địa chỉ tuyến tính (10 bit cao nhất) trở tới điểm vào bảng trang tương ứng (page table base entry). Điểm vào bảng trang lại trở đến cơ sở của bảng trang (page table).

- Phần offset bảng (10 bit giữa) của địa chỉ tuyến tính xác định offset trong bảng trang, nơi có điểm vào mô tả trang (page descriptor entry).

- Mô tả trang lại trở đến cơ sở của khối nhớ 4 kbyte trong bộ nhớ vật lý.

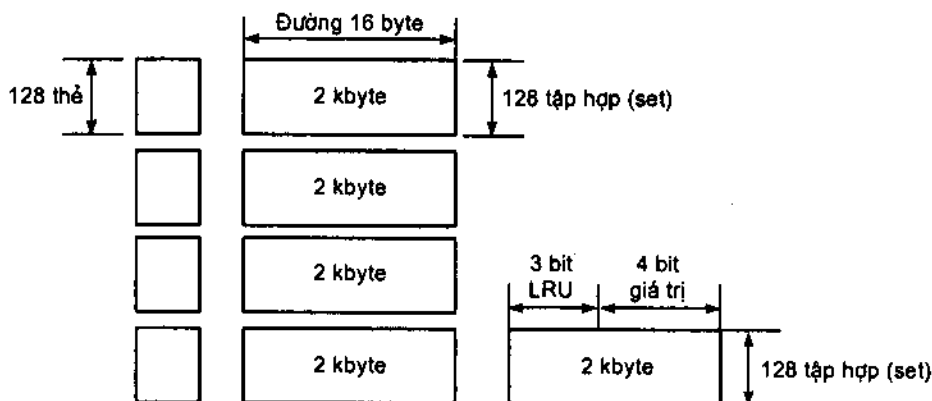
Phần offset vật lý (12 bit thấp) của địa chỉ tuyến tính xác định offset của địa chỉ vật lý nơi chứa mã lệnh (Op code) cần phải đọc ra.

Cơ chế chuyển đổi địa chỉ của 80486 cũng tương tự như 80386, do đó sơ đồ này cũng chỉ nhằm khái quát thêm quá trình này của 80386, 80486.

b) Bộ nhớ Cache của 80486

Hệ thống nhớ cho 80486 tương tự như cho 80386. 80486 địa chỉ vùng nhớ từ 00000000h đến FFFFFFFFh. 80486 có 8 kbyte L1 cache chung cho dữ liệu và lệnh.

Trong chip 80486 cache được tổ chức như trong hình 2.137. Điều khiển cache đơn giản và tỷ số trúng đích đối với cache chung cho cả dữ liệu và lệnh cao hơn so với cache được phân chia riêng Dcache và Icache. Nhưng nếu có phân chia riêng cache sẽ làm cho xử lý đường ống có hiệu quả hơn (vì vậy mà sau này trong các thế hệ Pentium, cache bên trong vẫn được phân chia làm 2 riêng biệt cho dữ liệu và cho lệnh).



Hình 2.137: Tổ chức cache bên trong 80486

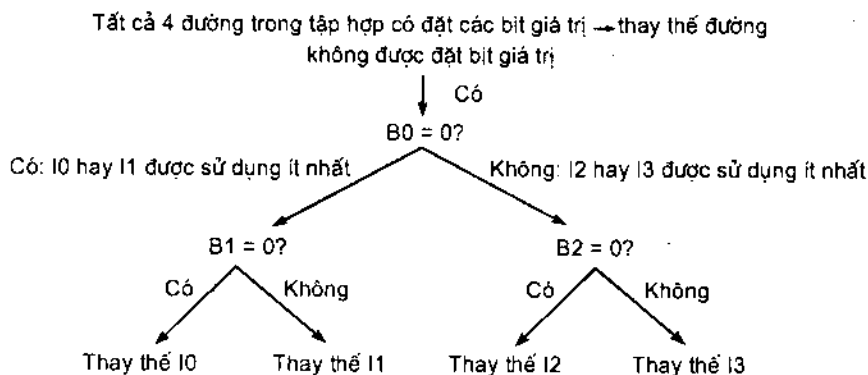
Cache của 80486 được tổ chức theo 4-đường liên kết tập hợp (4-way set-associative) để tăng hệ số trúng đích (hit) mà không cần

phải phức tạp thêm phần cứng. 8 kbyte cache của 80486 được chia thành 128 tập hợp. Cache được tổ chức thành 3 phần chính. Phần lớn nhất là phần chứa dữ liệu nằm bên phải trong hình 2.137. Đánh địa chỉ cho phần dữ liệu được phân thành 2 thành phần: chọn tập hợp và chọn đường (line). Phần địa chỉ chọn tập hợp có 7 bit (vì $2^7 = 128$ tập hợp). Phần chọn đường có 2 bit ($2^2 = 4$ đường). Mỗi một tập hợp có 4 đường (vì tên gọi của tổ chức cache là 4-đường liên hợp tập: tức là một tập hợp có 4 đường). Độ dài đường phụ thuộc vào dung lượng của cache. Nếu dung lượng cache là 8 kbyte thì độ dài đường sẽ là 16 byte = 8192:128:4. Phần thứ 2 là mảng thẻ (tag array), nằm bên trái trong hình 2.137. Một mảng thẻ gồm có 128 thẻ, mỗi thẻ có độ dài là 21-bit. Phần thứ 3 của cache là 128 tập hợp với mỗi tập hợp có 7 bit, trong đó 4 bit giá trị (valid bit) gán cho từng đường (4 đường) của một tập hợp trong phần chứa dữ liệu và 3 bit sử dụng cho thuật toán thay thế giả LRU (pseudo LRU), các bit này gọi là bit LRU. Như vậy, mỗi một tập hợp của phần thứ ba trở tới mỗi một tập hợp của phần chứa dữ liệu.

Mỗi một đường của một tập hợp trong phần chứa dữ liệu được gán nhãn bằng các bit giá trị I0, I1, I2, và I3 của tập hợp thuộc phần thứ ba. Các bit giá trị được xóa khi vi xử lý bị xóa bằng RESET hoặc khi cache được xóa. Sự thay thế trong cache được thực hiện nhờ thuật toán LRU giả (pseudo-LRU algorithm) khi cả 4 bit giá trị I0, I1, I2, và I3 được thiết lập cho cả 4 đường của tập hợp trong phần chứa dữ liệu. Hình 2.138 biểu diễn chiến lược thay thế trong cache của chip 80486.

Các bit LRU là B0, B1 và B2 dùng để xác định từng tập hợp trong 128 tập hợp. Các bit này được cập nhật cứ mỗi lần truy cập trúng đích (hit) hoặc có thay thế trong cache. Có một chiến lược thay thế trong cache như sau: nếu có nhiều truy cập hiện thời tới đường I0 hoặc I1 của một tập hợp chứa dữ liệu thì B0 được thiết lập bằng 1. Nếu hầu hết các truy cập hiện thời đều tới đường I2 hoặc I3 của tập hợp dữ liệu thì B0 được xóa về 0. Nếu có hầu hết các truy cập hiện thời tới các đường I0:I1 là tới I0 thì B1 được thiết lập bằng 1, ngược lại, B1 được

xóa về 0. Nếu có hầu hết các truy cập hiện thời tới các đường I2:I3 là tới I2 thì B2 được thiết lập bằng 1, ngược lại, B2 được xóa về 0.

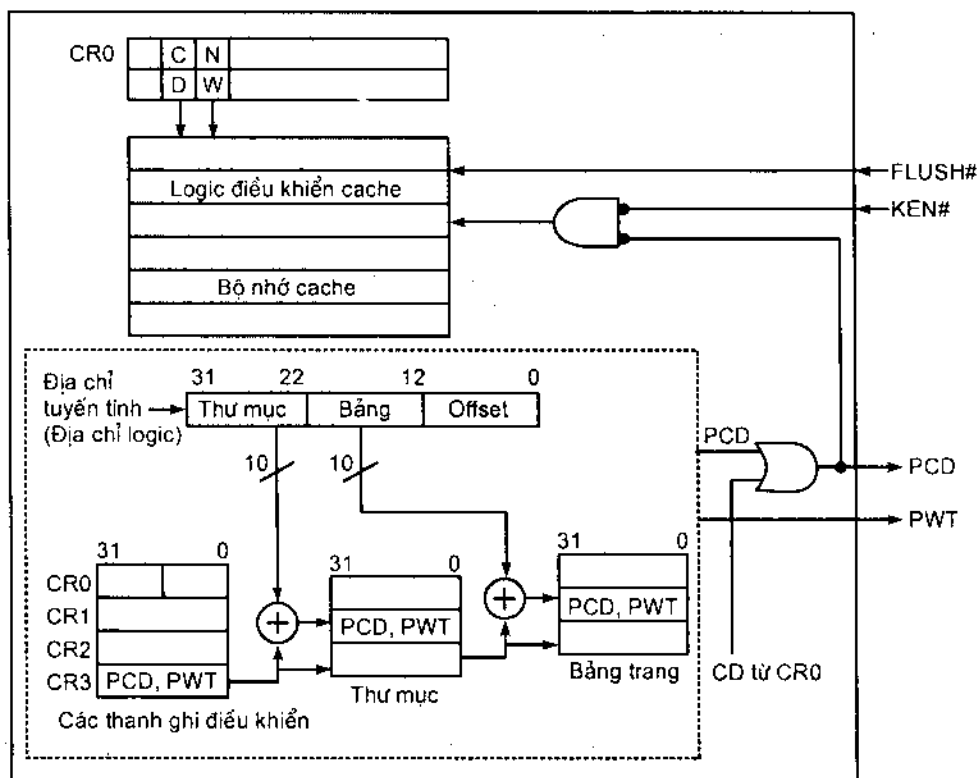


Hình 2.138: Chiến lược thay thế cache trong 80486

Ví dụ: trong lệnh vận chuyển, như MOV AX, BIEN, giá trị BIEN được lấy từ bộ nhớ. Nếu BIEN đã có sẵn trong cache thì tất yếu sẽ xảy ra “trúng đích” (Hit) và từ cache biến được lấy ra. Nếu BIEN không có trong cache thì một quá trình làm đầy đường cache (cache line full) sẽ phải sinh ra để lấy BIEN từ bộ nhớ RAM (hoặc bộ nhớ ngoài). Khi cache được làm đầy, phần mềm của đơn vị quản lý bộ nhớ MMU xác định đường nào đã ít được sử dụng nhất và thay thế đường đó bằng một đường dữ liệu mới.

Có 2 bit điều khiển trong thanh ghi CR3 trong tất cả các điểm vào trang (PTE) và thư mục trang (PDE). Các bit này ảnh hưởng đến các thao tác với cache. Giá trị của các bit này được điều khiển từ bên ngoài. Đó là các bit: cấm trang cache PCD (Page Cache Disable) và ghi thông suốt trang PWT (Page Write Through). Khi PCD = 0, thì cache bên trong chip 80486 được phép thao tác cho trang nhớ hiện hành. Thao tác với cache còn phụ thuộc vào tác động của tín hiệu KEN# và các bit CD (Cache Disable) và NW (Noncache Write through) trong thanh ghi CR0. Khi CD = 1, các thao tác với cache đều bị cấm. Bit NW để cấm thao tác ghi thông suốt vào cache. Bình thường CD = 0, và NW = 0. Khi thực hiện kiểm tra hệ thống có thể đặt CD = 1

và $NW = 1$. Để thao tác với cache bên trong chip cần phải $PCD = 0$, $CD = 0$ và $KEN\# = 0$. Khi $PWT = 1$, ta có chiến lược ghi thông suốt. Khi $PWT = 0$ ta có sự ghi lại (write-back) cache. Vì ghi vào cache bên trong chip 80486 luôn là ghi thông suốt, nên PWT chủ định cho chiến lược ghi cho cache bên ngoài chip 80486. Mỗi quan hệ giữa các bit PCD , PWT và tín hiệu $KEN\#$ được mô tả trong sơ đồ mạch cho phép thao tác trang cache bên trong chip 80486 (hình 2.139).

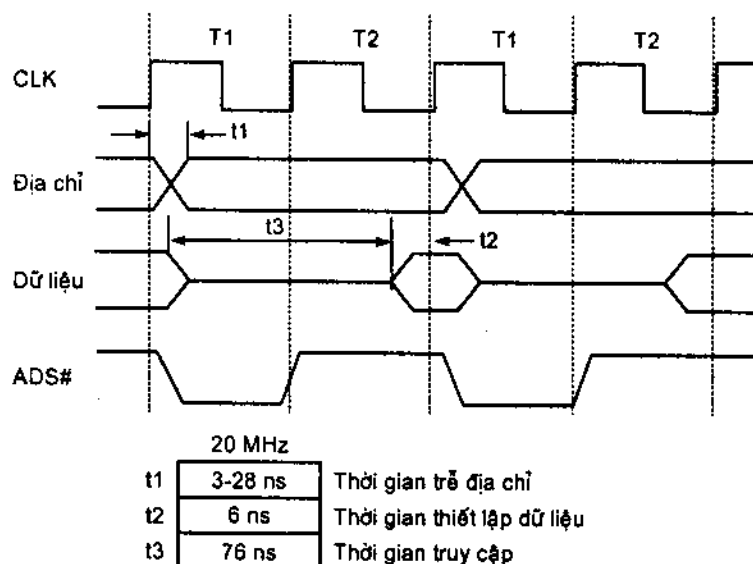


Hình 2.139: Cho phép trang cache bên trong 80486

Cache của 80486 được làm đầy bằng các chu trình bùng nhanh (burst cycles), mà các chu trình này không có trong 80386. Để làm đầy cache, 80486 có 4 số với độ dài 32-bit lưu trong bộ nhớ và chúng được đọc ra để làm đầy một đường của một tập hợp dữ liệu trong cache. Một chu trình bùng nhanh là một chu trình mà trong đó các số

(độ dài 32-bit) được đọc từ bộ nhớ trong 5 chu kỳ nhịp đồng hồ. Cho rằng tốc độ của bộ nhớ đủ nhanh và không cần thời gian chờ. Khi đó đối với phiên bản 80486 33 MHz chúng ta có thể làm đầy một đường của cache trong 167 ns. Tất cả các bit điều khiển liên quan đến quản lý bộ nhớ cache đều mới có trong 80486, mà không có trong 80386. Và như đã nhắc đến, hàng tiên đọc lệnh của 80486 có độ dài 32 byte, trong khi của 80386 chỉ có 16 byte.

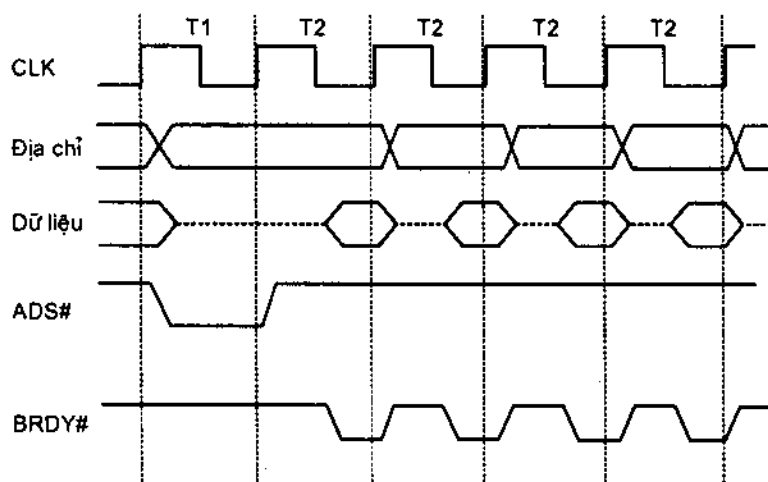
(4) Định thời đọc bộ nhớ (memory read timing)



Hình 2.140: Định thời chu kỳ đọc bùng phát của 80486

Hình 2.140 trình bày định thời gian đọc bộ nhớ không bùng phát của 80486. Hai chu kỳ nhịp đồng hồ được sử dụng để chuyển dữ liệu. Chu kỳ nhịp T1 đảm bảo có giá trị địa chỉ trên bus và các tín hiệu điều khiển. Chu kỳ nhịp T2 là quãng thời gian dữ liệu được chuyển giữa vi xử lý và bộ nhớ thông qua bus dữ liệu của hệ thống. Tín hiệu RDY# cần phải ở mức logic 0 để cho dữ liệu được vận chuyển và để kết thúc chu trình bus. Thời gian truy cập được xác định bằng quãng thời gian 2 chu kỳ nhịp T1, T2 trừ đi thời gian cần thiết để các giá trị địa chỉ có

trên bus địa chỉ trừ đi thời gian thiết lập kết nối bus dữ liệu. Đối với phiên bản 80486 20 MHz, tổng thời gian 2 nhịp T1 và T2 là 100 ns, thời gian thiết lập địa chỉ trên bus địa chỉ là 28 ns, và thời gian thiết lập kết nối bus dữ liệu là 3 ns, do đó thời gian truy cập là 69 ns. Nếu như tính cả thời gian giải mã, và trễ thì thời gian truy cập bộ nhớ sẽ nhỏ hơn.



Hình 2.141: Một chu kỳ đọc nhanh 4 từ kép trong 5 chu kỳ nhịp 80486

Trong hình 2.141, một chu kỳ làm đầy một đường 4 từ kép trong cache cực nhanh cần đến 5 chu kỳ nhịp đồng hồ (CLK): một chu kỳ T1 và 4 chu kỳ T2 (vì cache bên trong 80486 tổ chức thành 128 tập hợp, mỗi tập hợp có 4 đường độ dài 16 byte, hay 4 từ kép, mỗi từ kép độ dài 4 byte). Các đường dây địa chỉ A31-A4 xuất hiện trong chu kỳ thời gian T1 và duy trì không đổi trong suốt chu trình cực nhanh. A2 và A3 thay đổi trong từng chu kỳ T2 để địa chỉ 4 số 32-bit liên tiếp nhau trong bộ nhớ. Thời gian truy cập của phiên bản 80486 20 MHz đối với từ kép thứ hai và những từ kép tiếp theo là: $19 \text{ ns} = 50 \text{ ns} - 28 \text{ ns} - 3 \text{ ns}$, nếu cho rằng không có trễ trong hệ thống. Để sử dụng chế độ bùng nhanh (burst mode), cần phải có bộ nhớ tốc độ cao. Nếu sử dụng DRAM thì chỉ đạt thời gian truy cập tốt nhất là 55 ns, do vậy phải sử dụng SRAM mới sử dụng được chế độ bùng nhanh. Ngay cả với

SRAM nhanh chóng chỉ có thể sử dụng 80486 20 MHz là đảm bảo hơn cả. Chân tín hiệu BRDY# xác nhận sự chuyển dữ liệu của bộ nhớ trong chế độ cực nhanh hơn là tín hiệu RDY#, vì RDY# dùng để xác nhận vận chuyển dữ liệu của bộ nhớ trong chế độ bình thường.

(5) Đồng xử lý toán học trong chip 80486

Bên trong 80486, bộ đồng xử lý toán học, hay còn gọi là đơn vị xử lý số dấu phẩy động (FPU), thỏa mãn hoàn toàn các phép toán số dấu phẩy động theo tiêu chuẩn ANSI/IEEE 754-1985. Vì nằm bên trong chip 80486 nên có thể thực hiện được các phép cộng từ các số nguyên không dấu 8-bit đến các tính toán lượng giác với số thực 80-bit, trong khoảng từ 0-255 đến $\pm 10E \pm 4932$. FPU của 80486 có các thanh ghi và tập lệnh tương tự như trong 80387.

(6) Đa nhiệm (multitasking) của 80486

Đặc điểm đa nhiệm cho phép 80486 thực hiện một số chương trình đồng thời nhờ cơ chế chuyển đổi giữa chúng. Khi chuyển một chương trình (một nhiệm vụ) này sang một chương trình khác, có tới khoảng 100 từ 32-bit cần phải lưu giữ và phục hồi thông qua ngăn xếp. Để tăng tốc độ chuyển đổi này, Intel đã không theo phương pháp truyền thống dùng phần mềm mà thực hiện bằng phần cứng trong 80486. Như vậy hiệu suất chuyển đổi bằng phần cứng so với bằng phần mềm đạt tỷ lệ 10:1. Ngoài ra, 80486 còn cho phép chuyển đổi giữa các hệ điều hành khác nhau. Ví dụ, một máy tính trạm dùng 80486 có thể chạy một chương trình ứng dụng dưới hệ điều hành Unix trong chế độ ảo (chế độ bảo vệ), và một ứng dụng khác có thể chạy dưới MSDOS hay Windows trong chế độ thực cùng thời gian.

(7) Tập lệnh của 80486

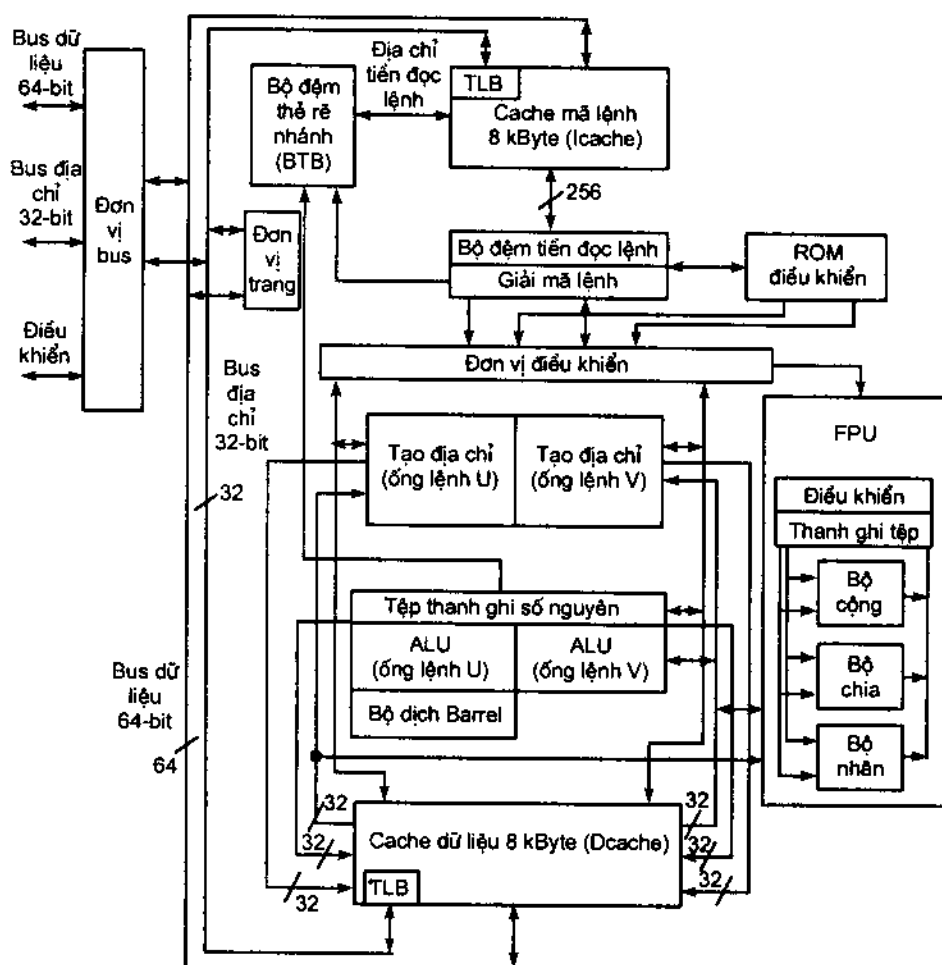
Tập lệnh của Intel X86 được trình bày trong phần Phụ lục C. Các chương trình viết bằng các lệnh của 8088/8086/80286/80386 đều có thể chạy trên được 80486. So với tập lệnh của 80386 thì 80486 có thêm một số lệnh mới: BSWAP, XADD, CMPXCHG, INVD, WBINVD, và

INVLPG. Sự khác nhau chính là nhiều lệnh của 80486 thực hiện trong một chu kỳ nhịp (tập lệnh giảm thiểu), trong khi đó 80386 cần 2 chu kỳ nhịp để thực hiện. Do đó, tốc độ thực hiện lệnh trong 80486 được tăng lên đến 50% so với 80386.

2.4.9. Vi xử lý Pentium

(1) Đóng vỏ và sơ đồ khối của Pentium

Pentium, là thế hệ sau 80486 của Intel, ra đời năm 1993 với công nghệ BiCMOS, 0,8 micron, 3 lớp metal, đóng vỏ 273 chân PGA 5 V, hay 296 chân 3,3 V Socket 4, 5, 6, với 3,1 triệu transistor. Nó có bus dữ liệu 64-bit cả trong và ngoài, có bus địa chỉ 32-bit đánh địa chỉ vật lý tới 4 GB. Họ Pentium gồm các loại làm việc ở các tần số nhịp 60/66/75/90/100/120/133/150/166/200 MHz. Đặc biệt, khác với 8 kbyte cache bên trong của 80486 chung cho cả dữ liệu và lệnh, Pentium có 8 kbyte cache bên trong riêng biệt để chứa dữ liệu (Dcache) và 8 kbyte cache riêng để chứa lệnh (Icache). Vì vậy nó có một bộ đệm chuyển đổi địa chỉ riêng TLB (Translation Lookup Buffer). Đây là một đặc tính mới trong Pentium. Sự tồn tại 2 loại cache bên trong và hai TLB tạo cho bộ vi xử lý thực hiện đồng thời lệnh và truy cập dữ liệu nhanh. Pentium có sự phân biệt với các thế hệ vi xử lý CISC có kỹ thuật xử lý song song mức lệnh ILP; nó có kiến trúc P5 siêu hướng 2 cửa ra (nghĩa là cho phép đồng thời đưa ra luồng lệnh để xử lý theo đường ống lệnh và kết quả đưa ra 2 cửa). Kiến trúc siêu hướng 2 cửa ra cho phép đọc và giải mã đồng thời 2 lệnh độc lập. Hiệu suất của Pentium có thể so sánh với một số loại vi xử lý RISC. Sơ đồ khối của Pentium cho ở hình 2.142. Với bus địa chỉ 32-bit cả bên trong và bên ngoài, giống như 80486. Có 256 đường giữa Icache và các bộ đệm tiền đọc lệnh (prefetch buffers), nên có thể thực hiện đọc trước các lệnh vào hàng lệnh 32 byte. Hiện nay, Pentium IV làm việc ở tốc độ trên 3,4 GHz.

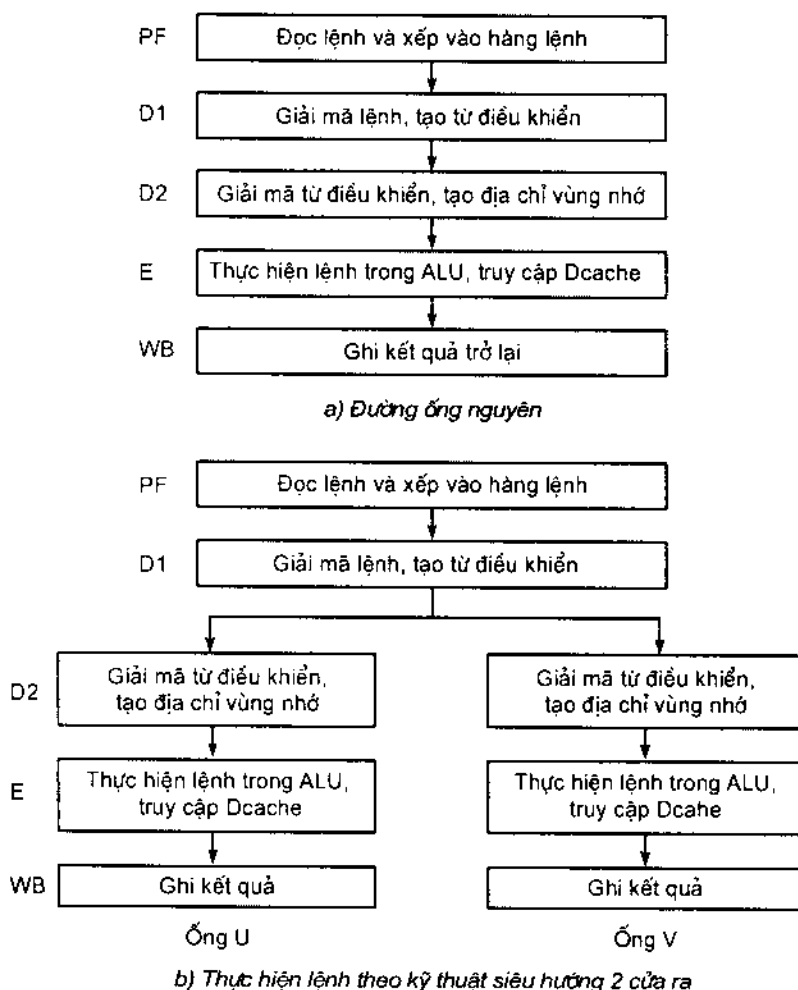


Hình 2.142: Sơ đồ khối Intel Pentium

(2) Kiến trúc đường ống của Pentium

Pentium có hai ống lệnh nguyên song song (parallel integer instruction pipelines): ống lệnh U (U-pipeline) và ống lệnh V (V-pipeline). Ống lệnh U có thể xử lý bất kỳ chỉ dẫn nào trong khi ống lệnh V chỉ có thể xử lý các chỉ dẫn đơn giản. Ống lệnh U có bộ dịch kiểu ống (barrel shifter) bổ sung cho ALU. Cũng có ống lệnh riêng của FPU với các đơn vị cộng, nhân, chia dấu phẩy động.

Dcache có hai cửa, nên cho phép các ống lệnh U và V đồng thời truy cập tới được. Bản thân cache là bộ nhớ xen kẽ liên tiếp nhau do đó nó cho phép thực hiện truy cập song song. Có cả bộ đệm thẻ rẽ nhánh BTB (Branch Target Buffer) để cung cấp các địa chỉ nhảy đọc trước cho cache lệnh.



Hình 2.143: Đường ống lệnh của Pentium

Pentium có ống lệnh thực hiện xử lý số nguyên 5 phân đoạn (5-stage integer pipeline), có rẽ nhánh ra ngoài đến 2 phân U và ống V ở 3 phân đoạn cuối cùng (hình 2.143). Các phân đoạn của ống lệnh như sau:

1. PF- Đọc lệnh (Prefetch): CPU đọc trước mã lệnh từ Icache và xếp mã lệnh vào hàng lệnh ở vị trí byte đầu tiên của lệnh tiếp theo để giải mã lệnh.

2. D1- Giải mã lần đầu (First decode): CPU giải mã lệnh để tạo ra từ điều khiển. Từ điều khiển gây ra sự thực hiện trực tiếp lệnh. Những lệnh phức tạp có thể tạo ra một chuỗi các vi lệnh điều khiển thực hiện lệnh.

3. D2- Giải mã lần thứ 2 (Second decode): CPU giải mã từ điều khiển (được tạo ra từ lần giải mã đầu tiên, D1) để tạo ra kết quả cho đoạn E tiếp sau. Tại đoạn này, các địa chỉ để tham chiếu bộ nhớ được tạo ra (địa chỉ toán hạng).

4. E- Thực hiện lệnh (Execute): Lệnh được thực hiện trong ALU. Nếu cần thiết, bộ dịch ống (barrel shifter) và các khối chức năng khác có thể tham gia thực hiện lệnh và Dache có thể được truy cập ở đoạn này.

5. WB- Ghi trở lại (Write Back): CPU cất giữ kết quả và tạo lập các cờ.

Các lệnh đơn giản của Pentium được thực hiện bằng phần cứng, và nói chung chỉ trong một chu kỳ nhịp đồng hồ như trong các hệ thống RISC. Ngoại trừ các lệnh có thao tác thanh ghi tới bộ nhớ (register-to-memory) hoặc bộ nhớ tới thanh ghi (memory-to-register) cần tới 2 hoặc 3 chu kỳ nhịp đồng hồ. Các lệnh được coi là đơn giản là: chuyển nội dung thanh ghi hoặc ngăn nhớ hoặc trực tiếp một giá trị vào thanh ghi, chuyển thanh ghi hay trực tiếp một giá trị vào ngăn nhớ, các lệnh số học nguyên, tăng, giảm nội dung thanh ghi, cất (Push) hoặc phục hồi (Pop) nội dung thanh ghi, nạp địa chỉ hiệu dụng, nhảy (Jump), gọi (Call), nhảy gần có điều kiện và không phép toán (no operation).

Để thực hiện đồng thời 2 lệnh theo đường ống siêu hướng 2 cửa ra cần có các điều kiện sau đây:

1. Các 2 lệnh phải đơn giản, như đã liệt kê ở trên.
2. Cần phải có sự phụ thuộc dữ liệu: đọc sau ghi RAW (Read-After-Write) hoặc ghi sau ghi WAW (Write-After-Write) giữa 2 lệnh.
3. Các lệnh không chứa giá trị độ dịch (displacement) hoặc giá trị trực tiếp.

4. Nếu là lệnh nhảy thì chỉ có thể xuất hiện ở đường ống U.

Ta có thể viết một giải thuật về những điều kiện trên như sau:

Giải mã 2 lệnh liên tiếp nhau I1 và I2.

Nếu những điều sau đây là thực:

I1 và I2 là những lệnh đơn giản.

I1 không phải là lệnh nhảy.

Đích của I1 không phải là nguồn của I2.

Đích của I1 không phải là đích của I2.

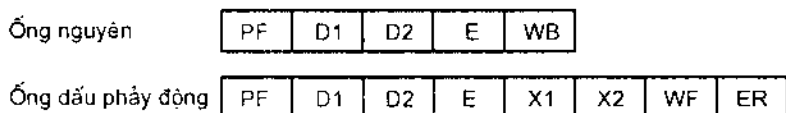
Thì đưa I1 vào đường ống U và I2 vào đường ống V.

Nếu không đưa I1 vào đường ống U.

Nếu lệnh thứ nhất được giải mã là lệnh nhảy, thì nó phải được đưa vào đường ống U để thực hiện, và không có lệnh nào được đẩy vào đường ống V. Trong trường hợp lệnh nhảy không thỏa mãn điều kiện thì đường ống được làm trễ bằng 3 hoặc 4 chu kỳ nhịp đồng hồ (lệnh nhảy được đọc trong phân đoạn E của đường ống). Khi lệnh nhảy được đọc đầu tiên, CPU phân phối một điểm vào trong 256 điểm của bộ đệm BTB để liên kết địa chỉ của lệnh nhảy với địa chỉ đích và khởi tạo lịch sử sử dụng trong thuật toán phỏng đoán. Khi các lệnh được giải mã, CPU tìm kiếm BTB để xác định điểm vào cho lệnh nhảy tương ứng. Khi có trúng đích (hit), CPU sử dụng lịch sử sử dụng để xác định việc lệnh nhảy cần được lấy. Nếu cần lấy, CPU sử dụng địa chỉ đích để bắt đầu đọc và giải mã các lệnh. Lệnh nhảy được thực hiện sớm trong giai đoạn WB, và nếu sự phỏng đoán sai, CPU xóa nhanh đường ống và trở lại đọc các lệnh theo nhánh đúng. CPU cập nhật lịch sử trong giai đoạn WB. Lệnh nhảy được đoán đúng thực hiện không có trễ.

Pentium có đường ống lệnh dấu phẩy động 8 phân đoạn (hình 2.144). Các phân đoạn này như sau:

1. PF- Đọc lệnh (prefetch): Các lệnh được đọc trước từ Icache.
2. D1- Giải mã lần đầu (First decode): Thực hiện giống như đoạn D1 của ống nguyên.
3. D2- Giải mã lần thứ 2 (Second decode): Thực hiện giống như D2 của ống nguyên.
4. E- đọc toán hạng (Operand fetch): Các toán hạng được đọc từ tệp các thanh ghi của FPU hay từ Dcache.
5. X1- Thực hiện lần thứ 1 (First execute): FPU thực hiện lệnh bước đầu tiên.
6. X2- Thực hiện lần thứ 2 (Second execute): FPU thực hiện lệnh bước thứ hai.
7. WF- Ghi kết quả (Write float): FPU hoàn thành thực hiện lệnh và ghi kết quả vào tệp thanh ghi.
8. ER- Thống kê lỗi (Error reporting): FPU thống kê các tình huống có thể yêu cầu xử lý bổ sung để hoàn thành quá trình thực hiện lệnh và tạo lập từ trạng thái của dấu phẩy động.



Hình 2.144: Đường ống dấu phẩy động

Ba đoạn đầu tiên của lệnh dấu phẩy động được tiến hành trong ống U (U-pipe), chúng tương tự như đối với các lệnh nguyên. Lệnh dấu phẩy động không thể tạo cặp với một lệnh khác (lệnh số nguyên hay số dấu phẩy động). Hai cửa của Dcache được sử dụng để truy cập các toán hạng số chính xác kép 64-bit.

(3) Cache bên trong của Pentium

Dcache và Icache bên trong Pentium đều có dung lượng 8 kbyte. Cache được tổ chức theo liên hợp tập 2 đường (2-way set-associative),

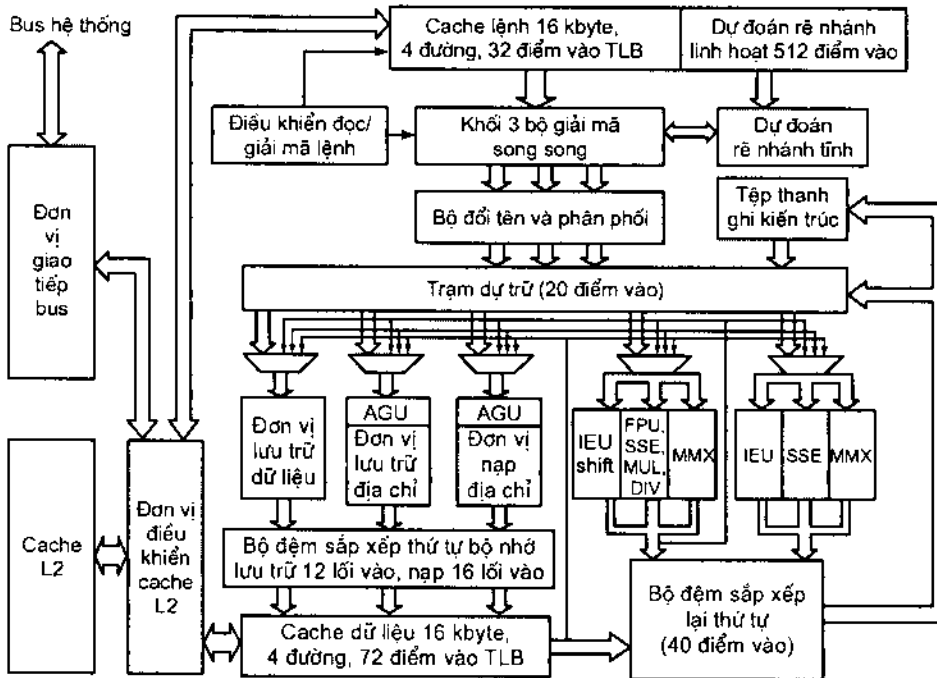
gồm 128 tập hợp, mỗi tập hợp có 2 đường. Như vậy, để chọn tập cần 7 bit, để chọn đường chỉ cần bit (một Flip-Flop), nghĩa là chỉ cần một byte để thực hiện thuật toán thay thế LRU. Trong trường hợp này, mạch logic quản lý cache sẽ đơn giản hơn. Dcache là cache được ghi lại (write-back) và sử dụng thủ tục kết dính cache: modified/exclusive/shared/invalid (MESI). Pentium sử dụng các trang dung lượng 4 kbyte thay đổi. Tuy vậy nó cũng có phương án chọn trang dung lượng 4 MB cho các gói phần mềm lớn. Mỗi một cache có TLB riêng (đây là đặc điểm mới trong Pentium, vì nó phân chia Dcache và Icache). TLB của Dcache có tổ chức liên hợp tập 4 đường, có 64 điểm vào cho các trang 4 kbyte. Có TLB của Dcache cho các trang 4 MB, có 8 điểm vào và các thông số tương tự như TLB cho các trang 4 kbyte. Chỉ có một TLB của Icache với 32 điểm vào cho các trang 4 kbyte. TLB này có tổ chức liên hợp tập 4 đường. Sự thay thế trong TLB được thực hiện nhờ thuật toán giả LRU tương tự như trong 80486, nhưng chỉ cần tới 3 bit cho một tập hợp.

2.4.10. Pentium III

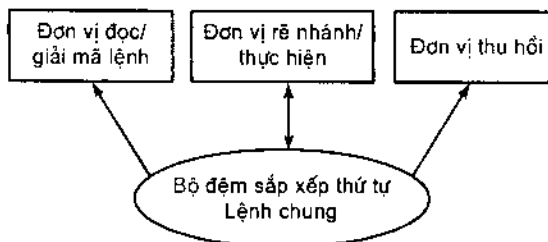
Hình 2.145 là sơ đồ khối của các loại Pentium III. Chúng tích hợp vi kiến trúc linh hoạt P6 (P6 Dynamic microarchitecture), kiến trúc 2 bus độc lập kép DIB (Dual Independent Bus), công nghệ bus hệ thống đa giao dịch (multi-transaction system bus) công nghệ đa phương tiện tiên tiến Intel MMX. Kiến trúc DIB làm tăng băng thông và hiệu suất cho từng bus riêng. Công nghệ vi kiến trúc linh hoạt P6 bao gồm khối đoán trước nhiều lệnh rẽ nhánh tĩnh và động (Dynamic Branch Predictor, Static Branch Predictor), phân tích luồng dữ liệu và thực hiện lấy lén.

Vi kiến trúc P6 bắt đầu được áp dụng cho Pentium Pro từ năm 1995 và vẫn được sử dụng cho Pentium III, nhưng linh hoạt hơn. Sự linh hoạt ở chỗ nó loại bỏ ràng buộc của trình tự các chỉ dẫn giữa các pha đọc và thực hiện truyền thống. Có một bộ đệm sắp xếp thứ tự ROB (Reorder buffer) (hình 2.146) lưu giữ các chỉ dẫn chưa thực hiện được ngay và xác định những phụ thuộc của các vi thao tác, phân bổ và sắp

xếp thứ tự các thanh ghi cho chúng, lập thời gian biểu sử dụng trạm dự phòng RS (Reservation Station). Một thời gian biểu tối ưu yêu cầu pha thực hiện được thay bằng cập pha rẽ nhánh/thực hiện dispatch/execute) và pha thu hồi (retire) sao cho các vi thao tác có thể bắt đầu thực hiện trong một thứ tự và phải hoàn thành và thu hồi kết quả trong thứ tự nguyên gốc. Điều này cho phép tăng được hiệu suất do sử dụng được nhiều tài nguyên của vi xử lý.



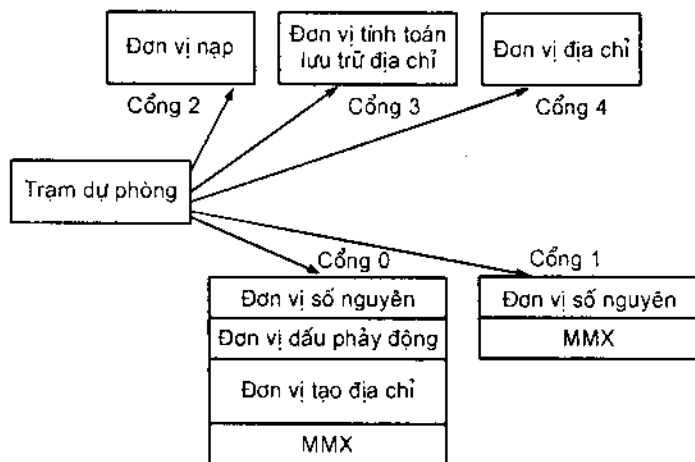
Hình 2.145: Sơ đồ khối Pentium III



Hình 2.146: Pha thực hiện thay bằng cập pha rẽ nhánh/thực hiện và thu hồi trong vi kiến trúc P6 lõi

Vi kiến trúc P6 lõi thực hiện các chỉ dẫn x86 bằng sự phân chia các chỉ dẫn thành các vi chỉ dẫn, gọi là vi thao tác (micro-ops). Nhiệm vụ này được khối 3 bộ giải mã song song ($3 \times$ parallel instruction Decoder) thực hiện trong phân đoạn D1 (giải mã lần đầu) của ống lệnh nguyên. Bộ giải mã thứ nhất chỉ có khả năng giải mã một chỉ dẫn x86 thành 4 hoặc một số ít vi thao tác trong từng chu kỳ nhịp, trong khi hai bộ giải mã khác có thể giải mã một chỉ dẫn x86 ra một vi thao tác trong từng chu kỳ nhịp. Một khi các vi thao tác được giải mã, chúng sẽ được đưa ra trạm dự phòng RS (Reservation Station) theo thứ tự trước sau. Trong RS các vi thao tác chờ đợi cho đến khi các toán hạng dữ liệu của chúng đã sẵn sàng. Khi một vi thao tác đã có tất cả tài nguyên dữ liệu cần thiết, nó sẽ rẽ nhánh từ RS đến các đơn vị thực hiện. Khi vi thao tác thực hiện xong, nó trở về bộ đệm sắp xếp lại thứ tự ROB và chờ để thu hồi (retirement). Trong giai đoạn này, các giá trị dữ liệu được ghi trở lại bộ nhớ và tất cả các vi thao tác được thu hồi theo thứ tự, 3 vi thao tác trong 1 lần. Có thể trong một nhịp đồng hồ lập thời gian biểu cho 5 vi thao tác, mỗi vi thao tác cho một cổng tài nguyên (hình 2.147). Trung bình, hầu hết các chỉ dẫn của x86 được giải mã thành 2 vi thao tác. Một số chỉ dẫn như "AND", "OR", "XOR" hoặc "ADD" lại thường giải mã ra chỉ một thao tác, trong khi các chỉ dẫn "DIV" hoặc "MUL", hoặc các toán hạng địa chỉ gián tiếp lại sản sinh nhiều thao tác hơn. Những chỉ dẫn phức tạp như các tính toán lượng giác thậm chí có thể sản sinh ra hàng trăm thao tác vượt ra khỏi độ dài sắp xếp của khối chuỗi vi lệnh. Khi làm việc với các chỉ dẫn của MMX, tất cả các mã lệnh chỉ cần có 1 vi thao tác ngoại trừ trường hợp tham chiếu toán hạng trong bộ nhớ và ghi vào bộ nhớ. Tập hợp các thanh ghi của MMX gồm 8 thanh ghi, do đó chúng có thể sử dụng để giảm bớt số tham chiếu đến bộ nhớ khi gặp các lệnh có nhiều tham chiếu bộ nhớ. Ngoài ra Pentium III cũng bổ sung các mở rộng SIMD (SSE) cho Internet với 70 lệnh xử lý hình ảnh, âm thanh và video 3D (MPEG2), nhận dạng tiếng nói. Pentium III làm việc ở các tần số từ 450 MHz đến 1,33 GHz. Khối giao tiếp bus BIU (Bus Interface Unit) kết nối với 100÷133 MHz bus hệ thống với 64-bit và tốc độ 1 GB/s.

BIU có các bộ đệm: 4 bộ đệm ghi lại (writeback buffer), 6 bộ đệm làm đầy (full buffer), và 8 cửa vào hàng bus (bus queue entries). Có 32 kbyte L1 cache không theo khối, gồm 16 kbyte Icache và 16 kbyte Dcache. Pentium III kết hợp trực tiếp ngay trên chip (on-die) 256 kbyte L2 ATC (Advanced Transfer Cache) có xử lý lỗi theo ECC. L2 cache là 8-đường liên hợp tập hợp (8-way set associativity), không theo khối, và kết nối qua bus 256-bit trực tiếp với chip vi xử lý. Đơn vị tạo địa chỉ AGU (Address Generation Unit) cũng quan trọng như ALU, bởi vì nó chịu trách nhiệm để dữ liệu từ hoặc tới đúng địa chỉ để nạp hay cất giữ.



Hình 2.147: Rẽ nhánh các vi thao tác từ trạm RS đến các cổng tài nguyên trong vi kiến trúc P6 lõi

1	2	3	4	5	6	7	8	9	10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sch	Dispatch	Exec

Hình 2.148: Ống lệnh 10 phân đoạn của vi kiến trúc P6

Pentium III kiến trúc siêu ống lệnh với 10 phân đoạn (hình 2.148). Các đơn vị thực hiện tính toán các số nguyên IEU (Integer Execution Unit) và FPU có kiến trúc đường ống xử lý các số dấu phẩy động 32-bit, 64-bit và 80-bit chính xác kép theo chuẩn 754. Pentium III có các loại đóng vỏ: S.E.C.C, Flip-Chip PGA (FC-PGA, FC-PGA2).

2.4.11. Pentium IV

Chúng ta đã xét các vi kiến trúc P5 trong các Pentium cổ điển, P6 trong các Pentium Pro, Pentium II và Pentium III với các công nghệ siêu đường ống lệnh 10 phân đoạn (Hyper Pipelined Technology). Nhưng bây giờ cho Pentium IV, Intel đã chuyển sang một công nghệ mới, đó là kiến trúc NetBurst. Pentium IV khởi đầu được sản xuất bằng quá trình 0,18 micron có mật độ chip lên đến 42 triệu transistor.

Kiến trúc NetBurst của Pentium IV bao gồm: Cache vận chuyển tiên tiến ATC (Advanced Transfer Cache), bus hệ thống 400 MHz, công nghệ siêu đường ống (Hyper Pipeline Technology), ETC, động cơ thực hiện nhanh REE (Rapid Execution Engine), và mở rộng SSE2 (Streaming SIMD extension 2).

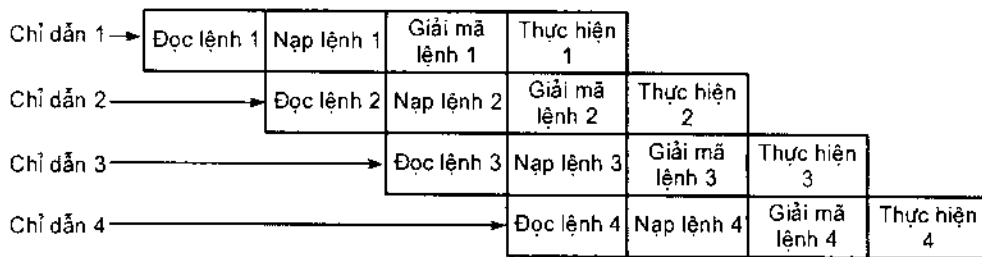
Tất cả các vi xử lý đều hỗ trợ bus mặt tiền FSB (Front Side Bus). FSB là kênh kết nối vi xử lý với bộ nhớ (RAM). Để có hiệu suất cao, kênh FSB cần phải hỗ trợ tốc độ truyền dữ liệu cao. Pentium III có bus 133 MHz, rộng 64-bit, tốc độ dữ liệu 1,06 GB/s. Trong Pentium IV, kênh FSB cũng có độ rộng 64-bit nhưng có tốc độ hiệu dụng tối đa 400 MHz. Đó là FSB 400 MHz, một ưu điểm của kiến trúc NetBurst. Chúng ta nói tốc độ hiệu dụng là bởi vì nhịp đồng hồ của bus là 100 MHz, và tốc độ này được nâng giá trị lên 400 MHz nhờ sử dụng chu kỳ nhịp đồng hồ hiệu quả hơn. Nghĩa là thay vì chuyển 1 bit dữ liệu trong 1 nhịp đồng hồ, ta chuyển được 4 bit. Như vậy bus được đẩy tốc độ lên gấp 4 lần. Tức là tốc độ dữ liệu đạt đến đỉnh là 3,2 GB/s. Với FSB có thể kết nối với các module nhớ tốc độ cao, như RDRAM và DDR SDRAM.

Siêu đường ống lệnh: một siêu đường ống lệnh của Pentium IV có tới 20 phân đoạn (hình 2.149), trong khi của kiến trúc P6 là 10 đoạn.

1 TC NXT	2 IP	3 TC	4 Fetch	5 Drive	6 Alloc	7 Rename	8	9 Que	10 Sch	11 Sch	12 Sch	13 Disp	14 Disp	15 RF	16 RF	17 Ex	18 Flgs	19 Br	20 Drive
----------------	---------	---------	------------	------------	------------	-------------	---	----------	-----------	-----------	-----------	------------	------------	----------	----------	----------	------------	----------	-------------

Hình 2.149: Ống lệnh 20 phân đoạn của Pentium IV

Chúng ta đã nói đến kỹ thuật đường ống lệnh trong Pentium. Đường ống lệnh đảm bảo cho vi xử lý thực hiện song song nhiều nhiệm vụ. Một đường ống bao gồm: ALU, các thanh ghi của CPU, các bộ giải mã lệnh,... Lý tưởng, càng có nhiều ống lệnh thì tốc độ vi xử lý càng nhanh hơn. Đó chính là kỹ thuật siêu đường ống lệnh. Các đường ống lệnh độc lập với nhau, và các chỉ dẫn độc lập với nhau được đẩy vào từng đường ống lệnh. Chẳng hạn như hình 2.150 cho ta ví dụ 4 ống lệnh độc lập với nhau, và có 4 chỉ dẫn được đưa vào 4 ống lệnh được xử lý song song theo các phân đoạn: đọc lệnh (Fetch), nạp lệnh (Load), giải mã lệnh (Decode), và thực hiện (Execute) trong một chu kỳ nhịp đồng hồ. Trong khi chỉ dẫn 1 được nạp vào thanh ghi, thì chỉ dẫn 2 được đọc từ bộ nhớ vào ống lệnh. Khi chỉ dẫn 1 được thực hiện, thì chỉ dẫn 2 được giải mã, chỉ dẫn 3 được nạp vào thanh ghi, chỉ dẫn 4 được đọc từ bộ nhớ vào ống lệnh.



Hình 2.150: 4 đường ống lệnh xử lý 4 lệnh song song của công nghệ siêu đường ống lệnh

Nếu cho rằng các lệnh này có sự phụ thuộc vào nhau, còn đường ống lệnh có nhiều phân đoạn hơn, và một lệnh nào đó bị dự đoán sai thì toàn bộ đường ống lệnh bị xóa, và phải thực hiện lại từ phân đoạn đầu tiên của ống lệnh. Bây giờ, giả sử đường ống lệnh dài 20 phân

đoạn, và ở phân đoạn 19 mới xảy ra dự đoán sai lệnh, thì sẽ là lãng phí lớn số chu kỳ nhịp phải bỏ đi. Giải pháp để khắc phục những trường hợp như thế này là đưa vào ETC (thay cho L1 Icache có trong các loại Pentium). Vì ống lệnh của Pentium IV có 20 phân đoạn, đơn vị dự đoán rẽ nhánh BPU (Branch Prediction Unit) của Pentium IV là một thành phần đặc biệt, bởi nó được gán động cơ thực hiện linh hoạt tiên tiến (Advanced Dynamic Execution engine). Với động cơ này Pentium IV tăng số chỉ dẫn được BPU sử dụng lên đến 126 so với 42 trong kiến trúc P6 (Pentium III), điều này làm tăng khả năng dự đoán rẽ nhánh đúng đến 95%, giảm tỷ lệ dự đoán sai xuống 33% so với kiến trúc P6 (Pentium III). Đó cũng còn nhờ có bộ đệm đích rẽ nhánh BTB (Branch Target Buffer) tăng dung lượng lên 4 kbyte (so với 512 byte trong kiến trúc P6) để chứa nhiều chi tiết lịch sử của các rẽ nhánh đã qua cũng như có thuật toán dự đoán rẽ nhánh mới.

ETC (Execution Trace Cache): Kiến trúc P5 và P6 có L1 cache và phân thành Dcache và Icache, nhưng ở Pentium IV chỉ có 8 kbyte L1 Dcache còn L1 Icache được thay bằng cache thực hiện theo vết ETC (Execution Trace Cache) và đó là một phần đặc điểm của vi kiến trúc NetBurst. Tập lệnh x86 là tập lệnh phức tạp (CISC), chúng có các chỉ dẫn độ dài khác nhau (tính theo số byte), do đó làm phức tạp việc tính vị trí cho chỉ dẫn tiếp theo. Sẽ hiệu quả hơn nếu vi xử lý làm việc với các chỉ dẫn có độ dài cố định, gọi là các vi thao tác. Đó là công việc của đơn vị giải mã trong đường ống lệnh để chuyển các chỉ dẫn x86 thành các vi thao tác. Trong một số chu kỳ nhịp nó phải chuyển đổi các mảng các chỉ dẫn thành các vi thao tác. Trong Pentium IV, ETC cất giữ các vi thao tác (ít nhất là các vi thao tác thường xuyên được sử dụng) trong luồng thực hiện chương trình mà các kết quả của các rẽ nhánh được tích hợp trong cùng một đường cache. Những vi thao tác này, qua một số chu kỳ nhịp thực hiện, lẽ ra phải được đơn vị giải mã tạo ra nếu dự đoán đúng. Trong trường hợp dự đoán rẽ nhánh

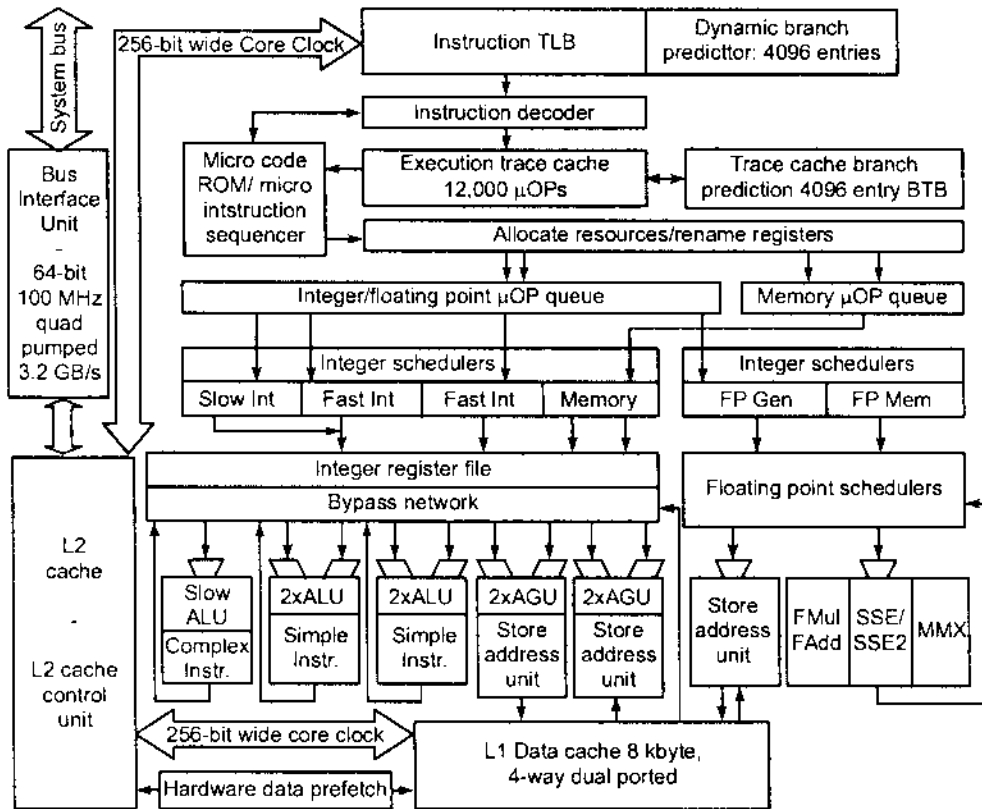
sai những vi thao tác này sẽ lấy từ ETC ra và đưa vào thực hiện ngay trong đường ống lệnh (phân đoạn thực hiện) mà không cần phải thực hiện lại từ phân đoạn đầu của ống lệnh như trong các Pentium trước đây.

ATC: Tương tự như Pentium III, L2 cache của Pentium IV cũng được thay đổi thành ATC. L2 ATC dung lượng 256 kbyte ÷ 1 MB làm việc với tốc độ của vi xử lý, đó là tốc độ đầy đủ. L2 ATC không tổ chức theo khối, có liên kết tập hợp 8 đường kết nối trực tiếp trên chip Pentium IV (on-die) bằng bus 256-bit và nhịp đồng hồ gốc (core clock) (hình 2.151). Nhưng có sự khác biệt là vận chuyển dữ liệu của ATC là trên từng nhịp đồng hồ, trong khi đó ở Pentium III dữ liệu vận chuyển trên từng chu kỳ nhịp khác. Do vậy tốc độ dữ liệu giữa L2 ATC của Pentium IV đạt đủ tốc độ nhịp đồng hồ (tốc độ đầy đủ), trong khi đó ở Pentium III thì đạt một nửa (nửa tốc độ). Tốc độ vận chuyển dữ liệu của ATC tới 48 GB/s ÷ 54,4 GB/s so với 16 GB/s của Pentium III. Ví dụ, so sánh Pentium IV với tần số nhịp 1,5 GHz và Pentium III với tần số nhịp 1 GHz:

$$\text{Intel Pentium IV } 1,5 \text{ GHz } 256\text{-bit } (32 \text{ byte}) \times 1 \times 1,5 \text{ GHz} = 48 \text{ GB/s}$$

$$\text{Intel Pentium III } 1 \text{ GHz } 256\text{-bit } (32 \text{ byte}) \times 0,5 \times 1 \text{ GHz} = 16 \text{ GB/s}$$

Động cơ thực hiện nhanh: Trong Pentium IV có 2 ALU và 2 AGU (hình 2.151). ALU chịu trách nhiệm thực hiện các tính toán số học và logic đơn giản. Ngoài ra còn có một ALU chậm chịu trách nhiệm thực hiện các tính toán phức tạp. Các chỉ dẫn cất giữ trong bộ nhớ (RAM) có địa chỉ theo 2 cách: trực tiếp và gián tiếp. Trong trường hợp đánh địa chỉ trực tiếp, địa chỉ là vùng nhớ chứa chỉ dẫn. Trong trường hợp đánh địa chỉ gián tiếp, địa chỉ lại chứa địa chỉ của vùng nhớ chứa chỉ dẫn. Các đơn vị AGU trước hết được sử dụng để giải quyết các địa chỉ gián tiếp. Với 2 ALU và 2 AGU Pentium IV xử lý gấp đôi số lệnh trong một chu kỳ nhịp.



Hình 2.151: Sơ đồ khối kiến trúc Pentium IV

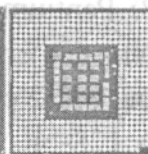
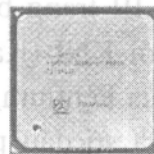
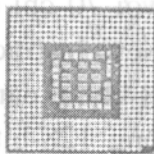
SSE2: Pentium III trang bị SSE với 70 chỉ dẫn mới và được bổ sung các chỉ dẫn mới KNI (Katmai New Instructions), trong khi đó các mở rộng SSE2 của Pentium IV có 144 chỉ dẫn mới (các phép toán số nguyên 128-bit và số dấu phẩy động 128-bit chính xác kép của SIMD). Hiệu quả của các ứng dụng như nhận dạng tiếng nói, hoạt hình và trò chơi 3D, chuỗi audio-video, nén và gỡ nén có thể được tăng lên nhờ những chỉ dẫn mới này. Pentium IV ban đầu sử dụng công nghệ lõi Willamette cho phép đạt tốc độ nhỏ hơn 2,0 GHz và được sản xuất bằng quá trình 180 nm (bảng 2.15). Tiếp sau là công nghệ lõi Northwood, ExtremeEdition (Gallatin), và hiện nay là Prescott. Để so sánh, transistor nhỏ nhất là ở công nghệ lõi Tualatin với độ dài mạch cổng là 70 nm. Trong khi của lõi Willamette trung bình là 100 nm. Hiện nay Pentium IV được giới thiệu dựa trên công nghệ lõi Prescott,

sản xuất theo Công nghệ lõi Prescott có những đặc điểm chính: tăng gấp đôi dung lượng của cả L1 và L2 cache, độ dài đường ống lệnh tăng lên đến 31 phân đoạn, đồng thời có một số nâng cấp các lệnh mở rộng SIMD (SSE3) và chức năng được bổ sung cho các đơn vị thực hiện, giảm chi phí sản xuất và định mức hiệu suất tốt hơn. So với công nghệ lõi Northwood, Prescott làm việc nóng hơn và đòi hỏi phải có hệ thống làm mát tốt.

Bảng 2.15: Phát triển của họ Pentium IV

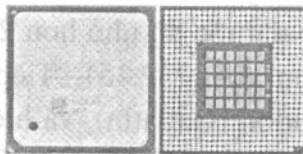
Công nghệ	Tần số lõi	L1 Data Cache	Unified L2 Cache	L3 Cache	Kích thước (Die Size)	Quá trình sản xuất (Process)	Tập lệnh mở rộng (Instruction Sets)
Willamette	< 2,0 GHz	8 kbyte 4-đường	256 kbyte	n/a	217 mm ²	180 nm	MMX, SSE, SSE2
Northwood	< 3,4 GHz	8 kbyte 4-đường	512 kbyte	n/a	146 mm ²	130 nm	MMX, SSE, SSE2
ExtremeEdition (Gallatin)	3,2 - 3,4 GHz	8 kbyte 4-đường	512 kbyte	2 MB	237 mm ²	130 nm	MMX, SSE, SSE2
Prescott	2,8 - 3,4 GHz	16 kbyte 8-đường	1024 kbyte	n/a	112 mm ²	90 nm	MMX, SSE, SSE2, SSE3

Hình 2.152 là đóng vỏ PGA 478 chân của 2 loại Pentium IV: lõi Prescott và lõi Northwood.



a) Pentium IV 2,8 A GHz Prescott.
Chỉ Prescott với 533 MHz FSB
và w/o Hyper-Threading

b) Pentium IV 3,2 E GHz Prescott
800 MHz FSB
Hyper-Threading



c) Pentium IV 3,4 GHz Northwood

Hình 2.152: Đóng vỏ 478 chân Pentium IV lõi Northwood và Prescott

2.4.12. Pentium M và công nghệ Centrino Mobile

Như đã trình bày, công nghệ Centrino Mobile gồm 3 thành phần chính: Pentium M, họ Intel 855 Chipset, và họ các kết nối vô tuyến Intel®PRO/Wireless 2100 hỗ trợ chuẩn 802.11b WLAN cho phép kết nối bất kỳ mạng WLAN nào (hot spots) với sự đảm bảo các giải pháp vô tuyến an ninh (cả cứng và mềm) gồm: WEP, 802.1X, WPA, AES, Cisco LEAP và chứng thực tương thích Cisco. Chuẩn 802.11b tần số mang sơ cấp 2,4 GHz có khoảng cách truyền tốt nhất và tốc độ vận chuyển dữ liệu 11 Mbit/s. Nó gồm cả sự chọn lựa ăng ten theo gói (per-packet antenna) để tối ưu hiệu suất của WLAN. Kết nối WLAN của Centrino có gói phần mềm Intel® PROSet cho phép dễ dàng cấu hình và quản lý các kết nối vô tuyến.

Vi kiến trúc P6+: Pentium M được gọi là Banias (thường gọi như vậy), thực sự là phiên bản sau cùng và lớn nhất của vi kiến trúc P6, mà từ Pentium Pro đến Pentium III đã sử dụng. Nhưng ngoài ra Banias còn áp dụng một số ít kỹ thuật của Pentium IV và có một số cải tiến hơn so với Pentium III và Pentium IV. Do đó gọi công nghệ lõi của Pentium M là P6+. Bảng 2.16 có sự so sánh ở một số thông số của Pentium M (Banias) và Pentium III-M, Pentium IV-M.

Bảng 2.16: So sánh Pentium M và Pentium III-M, Pentium IV-M

	Pentium III-M	Pentium IV-M	Pentium M
Lõi (core)	P6 (Tualatin 0,13 μ)	Netburst (Northwood 0,13 μ)	"P6 +" (Banias 0,13 μ)
L1 Cache (data + code)	16 kbyte + 16 kbyte	8 kbyte + 12 Kμops (TC)	32 kbyte + 32 kbyte
L2 Cache	512 kbyte	512 kbyte	1024 kbyte
Tập lệnh	MMX, SSE	MMX, SSE, SSE2	MMX, SSE, SSE2
Tần số tối đa (CPU/FSB)	1,2 GHz 133 MHz	2,4 GHz 400 MHz (QDR)	1,6 GHz 400 MHz (QDR)
Mật độ transistors	44 M	55 M	77 M
Mức tốc độ (SpeedStep)	Thế hệ 2	Thế hệ 2	Thế hệ 3

Pentium M có 2 loại đóng vỏ: Micro-FCPGA (Pin Grid Array) và Micro-FCBGA (Ball Grid Array), và 6 kiểu (bảng 2.17).

Bảng 2.17: Các loại Pentim M

Loại	Tần số (max/min)	Vcore (max/min)
Pentium M 1,6 GHz	1,6 GHz/600 MHz	1,484 V/0,956 V
Pentium M 1,5 GHz	1,5 GHz/600 MHz	1,484 V/ 0,956 V
Pentium M 1,4 GHz	1,4 GHz/600 MHz	1,484 V/ 0,956 V
Pentium M 1,3 GHz	1,3 GHz/600 MHz	1,388 V/ 0,956 V
Pentium M 1,1 GHz (nguồn thấp)	1,1 GHz/600 MHz	1,180 V/ 0,956 V
Pentium M 900 MHz (nguồn siêu thấp)	1,6 GHz/600 MHz	1,004 V/ 0,844 V

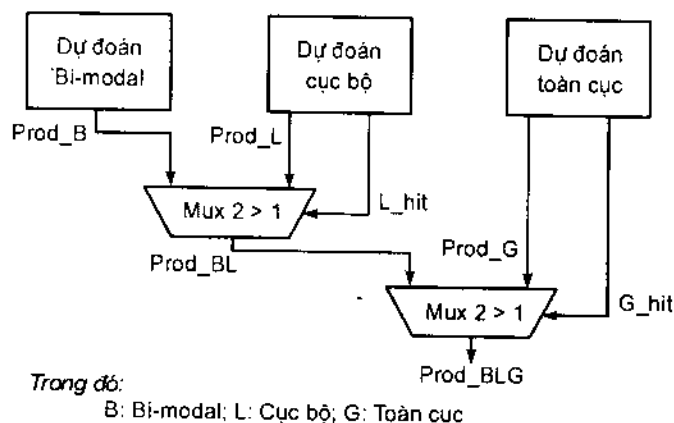
Pentium M 900 MHz có nguồn nuôi siêu thấp, chỉ xấp xỉ 1 V, và tiêu tán công suất chỉ 7 W. Trong khi đó, Pentium III-M 933 MHz cần nguồn nuôi 1,5 V và tiêu tán công suất hơn 20 W. Pentium M thực sự hiệu quả ứng dụng cho các thiết bị xách tay có nguồn nuôi thấp và tiết kiệm.

Advanced Branch Prediction: Vì kiến trúc P6 trong Pentium III (giống RISC) với 5 đơn vị và đường ống lệnh 10 phân đoạn. Nó đã tăng đáng kể hiệu suất và tốc độ xử lý. Pentium IV có đường ống lệnh 20 phân đoạn. Vì độ dài đường ống lệnh nhiều nên Pentium IV phải cải thiện các cơ chế dự đoán rẽ nhánh để tăng hiệu suất xử lý. Lỗi Pentium M (P6+), tần số 1,6 GHz chỉ có đường ống lệnh với số phân đoạn > 10 và < 20. Đó là giải pháp tối ưu đường ống lệnh, không ngăn và không quá sâu kết hợp với sự cải thiện các thuật toán dự đoán rẽ nhánh. Pentium III có 2 cơ chế rẽ nhánh. Pentium M cũng có 2 cơ chế dự đoán rẽ nhánh như của Pentium III nhưng tiên tiến hơn bởi có bổ sung thêm một cơ chế thứ 3 (hình 2.153):

+ Cơ chế tĩnh (static mechanism) sử dụng thuật toán cố định và liên quan đến các lệnh rẽ nhánh đơn giản. Thuật toán này cho phép quản lý các rẽ nhánh sao cho chúng phải được chắc chắn lấy hay không lấy. Nói cách khác, vi xử lý phải xác định hành vi của một số mẫu rẽ nhánh. Thuật toán này gọi là "bi-modal".

+ Cơ chế dự đoán linh hoạt (dynamic prediction mechanism), dùng cho các vòng lặp cục bộ (local loops) trong chương trình. Có một bộ đệm cục bộ được sử dụng để cất giữ lịch sử các rẽ nhánh. Khi gặp một rẽ nhánh, CPU tìm trong bộ đệm và tác động tương ứng với kết quả. Cơ chế này gọi là dự đoán cục bộ (Local Predictor).

+ Cơ chế dự đoán linh hoạt thứ 3, tương tự như cơ chế dự đoán linh hoạt trên, nhưng là toàn cục, và gọi là dự đoán toàn cục (Global Predictor).



Hình 2.153: Các cơ chế dự đoán rẽ nhánh của Pentium M

Với dự đoán rẽ nhánh tiên tiến (Advanced Branch Prediction) gồm 3 cơ chế dự đoán rẽ nhánh này Pentium M giảm tỷ lệ đoán sai đến 20% so với Pentium III.

Pentium M có sự quản lý các chỉ dẫn hiệu quả hơn. Pentium M có 2 mức cải tiến sự quản lý các chỉ dẫn. Ta biết rằng các chỉ dẫn được đọc từ bộ nhớ và được giải mã trong đường ống lệnh thành các vi thao tác (vi lệnh) cho các đơn vị thực hiện như ALU, FPU, Load Unit... Ví dụ, xét việc thực hiện lệnh sau đây:

ADD EAX,[memory] ; cộng nội dung một ô nhớ với nội dung thanh ghi EAX

Lệnh này có 2 thao tác nhỏ: đọc (hay nạp) nội dung ô nhớ vào một thanh ghi, được đơn vị nạp thực hiện (Load unit); thao tác cộng (addition), được ALU thực hiện. Như vậy, có một trình tự thực hiện lệnh trên với 2 bước như sau:

```
LOAD reg0,[memory]
```

```
ADD reg1,reg0
```

Sự cải tiến của Pentium M ở đây là liên kết 2 thao tác nhỏ thành một dòng thao tác (fuse) để lệnh chỉ cần thực hiện một bước thao tác mà thôi, tức là Pentium M nhìn nhận lệnh trên chỉ như là một thao tác:

```
ADD reg1,[memory]
```

Micro-Ops Fusion: cơ chế liên kết các thao tác thành một dòng gọi là Micro-Ops Fusion. Số lượng thao tác có thể giảm được đến 10% trong luồng chỉ dẫn chung nhờ cơ chế Micro-Ops Fusion trong Pentium M. Cơ chế này cho phép sử dụng ít tài nguyên của vi xử lý hơn.

Dedicated Stack Management: cải tiến thứ hai liên quan đến quản lý ngăn xếp (stack management). Sự quản lý ngăn xếp phải tiêu tốn thời gian nhiều nếu có nhiều chương trình con vì lúc đó phải thực hiện nhiều chỉ dẫn cất giữ và đọc dữ liệu với ngăn xếp. Trong Pentium M số lượng các thao tác con được giảm thiểu để tăng tốc xử lý với ngăn xếp nhờ quản lý ngăn xếp trực tiếp (Dedicated Stack Manager). Sự quản lý ngăn xếp trực tiếp được làm bằng phần cứng cho phép thực hiện các chỉ dẫn của chương trình không bị ngắt và sử dụng nguồn cung cấp nhỏ.

L1 cache: không giống Pentium IV, Pentium M không có TC (Trace Cache) nhưng lại có L1 Icache (code cache) như trong Pentium III. L1 cache của Pentium M giống như của Pentium III nhưng dung lượng lớn hơn. Bảng 2.18 có sự so sánh các Pentium theo thông số cache.

Bảng 2.18: So sánh Pentium theo thông số cache

L1 cache	Pentium III	Pentium IV	Pentium M
Dung lượng	Icache: 16 kbyte Dcache: 16 kbyte	TC: 12 Kops Dcache: 8 kbyte	Icache: 32 kbyte Dcache: 32 kbyte
Liên hợp	Icache: 4-đường Dcache: 4-đường	TC: 8-đường Dcache: 4-đường	Icache: 8-đường Dcache: 8-đường
Kích thước đường	Icache: 32 byte Dcache: 32 byte	TC: n.a Dcache: 64 byte	Icache: 64 byte Dcache: 64 byte
Chính sách ghi	Ghi trở lại	Ghi thông suốt	Ghi trở lại
Trễ (latency)	3 chu kỳ	2 chu kỳ	3 chu kỳ

Pentium IV có kích thước các đường cache là 64 byte. Kích thước này dùng cho tập lệnh của SSE sử dụng các thanh ghi và các biến 128-bit. Đường cache dài cho phép nạp đồng thời nhiều biến hơn từ bộ nhớ vào cache, làm tăng tốc độ xử lý. Nhưng trễ cache thường tăng lên cùng với kích thước của cache, nhưng với Pentium M, trễ là 3 chu kỳ giống như Pentium III, mà dung lượng cache và kích thước đường cache thì gấp đôi. Đó là ưu điểm của Pentium M.

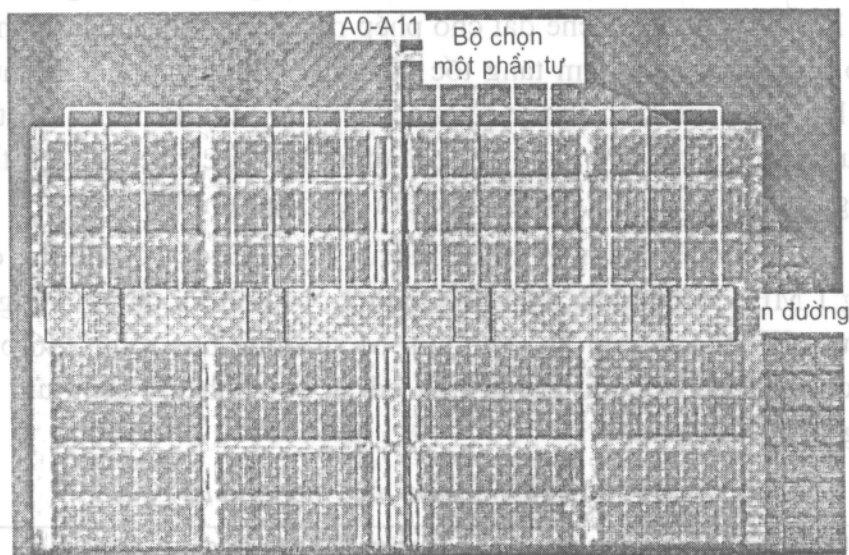
L2 cache: Pentium M có L2 cache: là ATC tốc độ đầy đủ, dung lượng 1 MB, chiếm phần diện tích lớn trên bề mặt lõi chip. Giống như Pentium III và Pentium IV, Pentium M sử dụng bus rộng 256-bit để trao đổi dữ liệu với lõi. Bảng 2.19 có sự so sánh các Pentium theo thông số L2 cache.

Bảng 2.19: So sánh Pentium theo thông số L2 cache

L2 Cache (ATC)	Pentium III-S	Pentium IV Northwood	Pentium M
Dung lượng	512 kbyte	512 kbyte	1024 kbyte
Liên hợp	8-đường	8-đường	8-đường
Kích thước đường	32 byte	64 byte	64 byte
Trễ (Latency)	4 chu kỳ	7 chu kỳ	5 chu kỳ

1 MB L2 cache của Pentium M được quản lý theo 8 khối 128 kbyte. Mỗi đường cache dài 64 byte như của Pentium IV, như vậy có 2048 đường trong một khối. Trong khi đó, ở Pentium III-S, có mỗi khối 64 kbyte, nhưng các đường kích thước 32 byte, mỗi khối cũng chứa 2048 đường.

Cơ chế tiên đọc lệnh: Pentium M cũng sử dụng một cơ chế tiên đọc lệnh (Prefetch mechanism) ứng dụng trong lõi P6 (Tualatin). Đặc tính này gồm các giao dịch với bộ nhớ để lọc các mẫu truy cập và dự đoán các địa chỉ nào của bộ nhớ sẽ cần thiết trong tương lai gần. Như vậy, những vùng nhớ liên quan sẽ được nạp trước vào cache trong các chu kỳ rỗi (idle) của bus. Do đó tăng được cơ hội tìm thấy dữ liệu tiếp theo trong cache (hệ số Hit cao). Cơ chế tiên đọc lệnh của Pentium M cho phép truy xét đồng thời 8 luồng lên (up-streaming) và 4 luồng xuống giao dịch bộ nhớ. Điều này làm cho cơ chế tiên đọc lệnh của Pentium M hiệu quả hơn Pentium III và Pentium IV.



Hình 2.154: Bộ chọn một phần tư khối của L2 cache (Quadrant Selector)

Bộ chọn một phần tư: với cache chung, sự đánh địa chỉ một đường (để nhận lại dữ liệu mà một đường cache có chứa) trước hết là làm tích cực toàn bộ khối chứa đường đó. Tức là đánh thức khối và báo rằng: “ta cần khối này”. Như vậy sự làm tích cực khối cần nguồn cung cấp. Khối càng lớn thì cần nhiều nguồn cung cấp hơn. Vậy kích thước khối của L2 cache của Pentium M là 128 kbyte lẽ ra phải cần công suất nguồn nuôi lớn. Như thế thì không phù hợp với công nghệ di

động. Pentium M đã giải quyết vấn đề này bằng cách: chia mỗi khối của L2 cache ra 4 phần (4 quadrant), mỗi phần kích thước 32 kbyte và sử dụng bộ chọn 1 phần tư (Quadrant Selector) (hình 2.154) để chỉ phải chọn một phần tư khối chứ không phải chọn cả khối. Tức là đã tiết kiệm nguồn nuôi đến 4 lần. Tiêu thụ nguồn cho 1 MB L2 cache của Pentium M chỉ bằng dùng cho 256 kbyte cache. Nhưng sự phân chia khối của L2 cache kéo theo trễ truy cập cache tăng lên, điều này cần nghĩ vì sao trong bảng 2.21 có ghi trễ truy cập L2 cache của Pentium M là 5 chu kỳ, chậm hơn so với L2 cache của Pentium III-S (4 chu kỳ).

Mức tốc độ thế hệ thứ ba (GeyserVille III): sự cải tiến nữa trong Pentium M là công nghệ mức tốc độ thế hệ thứ ba (SpeedStep III). Công nghệ mức tốc độ được áp dụng cho Mobile Pentium III (mobile). đây là cách giảm thiểu linh hoạt nguồn nuôi và tần số của vi xử lý để tiết kiệm tiêu thụ nguồn. Như thế vi xử lý sử dụng 2 sơ đồ làm việc: chế độ nguồn thấp, sử dụng khi vi xử lý rỗi (idle) hoặc khi mức pin thấp; và chế độ làm việc đầy đủ. Thế hệ thứ nhất của công nghệ mức tốc độ dựa trên 2 sơ đồ làm việc này. Pentium III-M (dựa trên lõi Tualatin) và Pentium IV-M đã cải tiến cơ chế mức tốc độ bằng việc bổ sung thêm một sơ đồ nguồn nữa ngoài 2 sơ đồ trên. Sơ đồ nguồn thứ 3 này gọi là chế độ thích ứng (adaptive mode), và cho phép chuyển mức tần số và nguồn tương ứng với hoạt động của CPU. Ngầm định, CPU sử dụng chế độ nguồn thấp, nhưng khi sự hoạt động của CPU tăng lên, nó tự chuyển cực nhanh vào chế độ làm việc đầy đủ. Sơ đồ nguồn mới này rất được thường xuyên sử dụng, bởi vì nó cho phép nhận được tốc độ đầy đủ của CPU chỉ khi nào nó cần thiết. Tất nhiên sự tiêu thụ nguồn phụ thuộc vào hoạt động của CPU, và CPU càng được sử dụng nhiều thì càng phải tiêu tốn nguồn hơn. Cơ chế mức tốc độ trong Pentium M được cải tiến hơn: điện thế và tần số của CPU có thể được chuyển nhiều mức hơn. Ví dụ, Pentium M 1,6 GHz cho phép linh hoạt chuyển 6 mức (bảng 2.20).

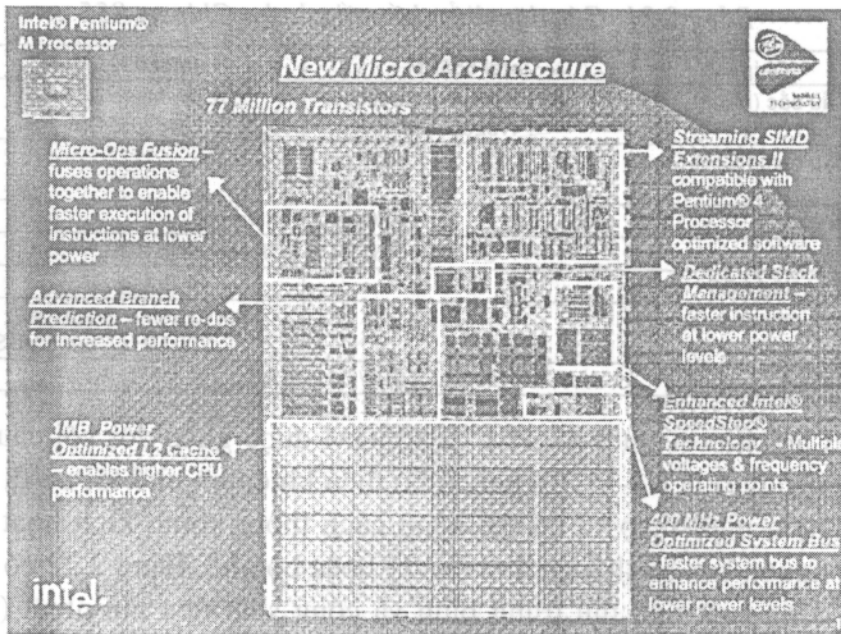
*Bảng 2.20: Lõi Pentium M 1,60 GHz Core VCC
với từng nấc tần số làm việc và Vcc tương ứng*

Tần số làm việc	Vcc
1,6 GHz	1,484 V
1,4 GHz	1,42 V
1,2 GHz	1,276 V
1 GHz	1,164 V
800 MHz	1,036 V
600 MHz	0,956 V

Sự điều chỉnh mức tần số/nguồn được thực hiện bằng phần mềm thông qua lập trình ghi vào thanh ghi chế độ MSRs (Model Specific Registers) của vi xử lý. Ngoài ra, có sự cải thiện chế độ giám sát nhiệt (Intel Thermal Monitor mode) của Pentium M. Khi bộ cảm biến nhiệt ở ngay chip (on-die thermal sensor) chỉ rằng nhiệt độ đã quá cao, Pentium M có thể tự động thực hiện một chuyển đổi xuống mức tần số/nguồn thấp được xác định trong thanh ghi MSR lập trình được. Pentium M chờ một quãng thời gian cố định. Nếu nhiệt độ hạ được xuống mức chấp nhận, thì vi xử lý thực hiện chuyển lên mức tần số/nguồn trước đó. Một ngắt được sinh ra để điều khiển các mức nhiệt độ lên hoặc xuống. Công nghệ chuyển mức tốc độ tiên tiến (Enhanced SpeedStep Technology) thế hệ III là một ưu điểm có hiệu năng rất cao nhờ tiết kiệm nguồn nuôi tối ưu. Pentium M có đặc tính: tự động dừng (Auto Halt), cho phép dừng (Stop Grant), ngủ sâu (Deep Sleep), và các trạng thái nguồn thấp ngủ sâu.

Hình 2.155 cho ta thấy sự sắp xếp các phân cứng các khối chức năng tạo nên những ưu điểm nêu trên của kiến trúc lõi P6+ của Pentium M.

Pentium M có mở rộng SSE2 của SIMD giống như Pentium IV, có bus hệ thống 400 MHz, sử dụng chuyển đồng bộ nguồn SST (Source-Synchronous Transfer) địa chỉ và dữ liệu để tăng gấp 4 lần trên một nhịp của bus hệ thống.



Hình 2.155: Sự sắp xếp trong lõi Pentium M

Banias là Pentium M được sản xuất bằng quá trình 130 nm (bảng 2.18). Hiện đã có Pentium M (gọi là Dothan) được sản xuất bằng quá trình 90 nm. Chúng bắt đầu ở tốc độ nhịp đồng hồ 1,8 GHz, nguồn nuôi 1,3 V và 2 MB L2 cache. Các chip Pentium M 735, 745, và 746 thuộc quá trình sản xuất 90 nm. Pentium M có đóng vỏ Micro-FCPGA (socketable Micro Flip-Chip Pin Grid Array) và Micro-FCBGA (Micro Flip-Chip Ball Grid Array) với đế cắm ZIP 478 lỗ (Zero Insertion Force socket), gọi là mPGA479M socket.

Họ Intel 855 chipset: gồm Intel 855PM (Odem) và 855GM (Montara-GM) hỗ trợ kết nối tới 2 GB RAM loại DDR333 với bổ sung DDR266/200. Các đặc điểm kết nối của họ chipset 855 mô tả trong bảng 2.21. Thế hệ chipset tiếp theo của Pentium M tung ra thị trường năm 2004 là PCI Express với PCI Express x16 interface AGP bus và hỗ trợ DDR-II SDRAM.

Bảng 2.21: Các đặc điểm kết nối của họ Chipset 855

Chipset	Intel 855PM	Intel 855GM
Cầu bắc (MCH)	593 pin Micro-FCBGA FW82855PM	732 pin Micro FCBGA FW82855GM
Cầu nam (ICH4-M)	421 pin Micro-BGA 82801DBM	
Vi xử lý	400 MHz Pentium M	
Bộ nhớ	200/266 MHz DDR SDRAM Up to 2GB	
Giao tiếp AGP	4X	4X (Integrated)
Tích hợp đồ họa	-	Intel Extreme Graphics 2 (2P1T, 200 MHz Core, LVDS)
Chức năng khác	2 x Ultra ATA-100/ 6 USB 2.0 ports, MAC, AC'97	

Các chipset Intel 855PM và 855GM được ghép đôi với cầu nam (SouthBridge) ICH4-M. Cả hai có bộ điều khiển bộ nhớ tối ưu di động với nguồn nuôi thấp và có thể bằng cách thông minh kích hoạt hoặc giảm nguồn bus hệ thống của vi xử lý hoặc bộ nhớ. Bảng 2.22 có sự so sánh công suất tiêu thụ trung bình và công suất tiêu tán nhiệt (TDP) của các loại Pentium III-M, Pentium IV-M với kết nối các chipset tương ứng. Dễ nhận thấy, tổng TDP của Pentium M nằm giữa Pentium IV-M và Pentium III-M nhưng công suất tiêu thụ trung bình (APC) của Pentium M là nhỏ nhất.

Bảng 2.22: So sánh Pentium III-M, Pentium IV-M với kết nối các chipset tương ứng

Thermal Dissipation Power (TDP)	PIV-M 2,4 GHz/ Intel 852GM	PIII-M 1,2 GHz/ Intel 830M	Pentium M 1,6 GHz/855GM
CPU	35 W	22 W	24,5 W
Bộ nhớ	12 W	8 W	12 W
Cầu bắc (NorthBridge) (Int. Graphics)	3 W	3,8 W	3,2 W
Cầu nam (SouthBridge)	2,2 W	2 W	2,2 W
Tổng TDP	52,2 W	35,8 W	41,9 W
Công suất tiêu thụ trung bình (APC)			
CPU	2 W	1,5 W	1 W
Bộ nhớ	0,3 W	0,2 W	0,3 W
Cầu bắc	2 W	2 W	1,7 W
Cầu nam	0,6 W	0,6 W	0,6 W
Tổng APC	4,9 W	4,3 W	3,6 W

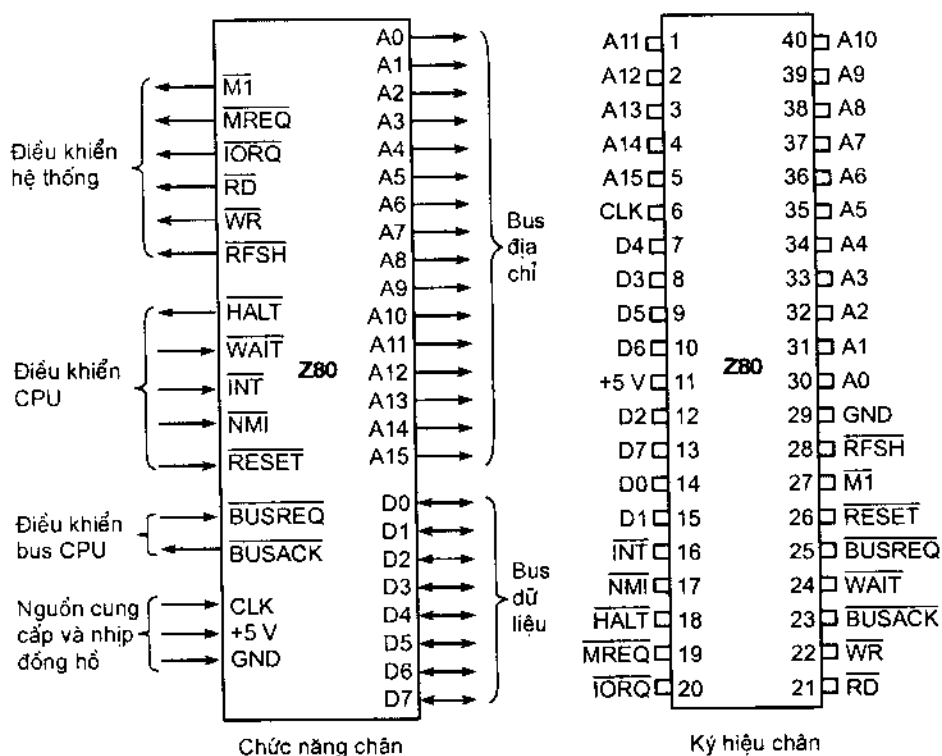
2.5. VI XỬ LÝ Z80

2.5.1. Đóng vỏ và sơ đồ khối của Z80

Z80 của Zilog là vi xử lý sau đã kế thừa tập lệnh và cấu trúc của 8080. Chip Z80 đóng vỏ DIP 40-pin (hình 2.156).

Các tín hiệu của Z80 phân thành 6 nhóm:

16 đường bus địa chỉ (A15-A0), 8 đường bus dữ liệu (D7-D0), 6 tín hiệu của bus điều khiển hệ thống (system control lines), 5 tín hiệu điều khiển của đơn vị xử lý trung tâm (CPU control), 2 tín hiệu của điều khiển bus của đơn vị xử lý trung tâm (CPU bus control), 3 tín hiệu cung nguồn và nhịp đồng hồ (supply, clock).



Hình 2.156: Đóng vỏ và các tín hiệu của vi xử lý Z80

*Các tín hiệu của Z80:**- Bus địa chỉ (A15-A0)*

16 đường bus địa chỉ cho phép đánh địa chỉ đến 64 kbyte nhớ (2^{16}). 8 đường địa chỉ thấp (A7-A0), được sử dụng để đánh địa chỉ 256 cổng vào/ra cho các thiết bị ngoại vi. Bus địa chỉ có thể chuyển về trạng thái trở kháng cao (trở bus) trong trường hợp thiết bị ngoại vi muốn chiếm đoạt bus (ví dụ, DMA).

- Bus dữ liệu (D7-D0)

8 đường bus dữ liệu 2 chiều kết nối với các đường dữ liệu của thiết bị ngoại vi và bộ nhớ (RAM, ROM). Trước hết phải có xuất hiện địa chỉ (A15-A0) trên bus để chọn ngăn nhớ của bộ nhớ hoặc thanh ghi nào đó của đơn vị điều khiển vào/ra của thiết bị ngoại vi, rồi sau đó mới xuất hiện các dữ liệu (D7-D0) theo hướng: từ bus vào CPU (đọc dữ liệu từ ngoại vi hay bộ nhớ), từ CPU ra bus (ghi dữ liệu từ CPU ra ngoại vi hay bộ nhớ). Bus dữ liệu có thể chuyển đến trạng thái trở kháng cao nếu có thiết bị ngoại vi chiếm đoạt bus hệ thống, ví dụ, để điều khiển DMA.

- Các đường tín hiệu điều khiển của hệ thống

Đó là 6 tín hiệu ra của Z80 để điều khiển hệ thống: M1#, MREQ#, IORD#, RD#, WR#, và RFSH#.

Tín hiệu M1# là tín hiệu định thời có nguồn gốc từ nhịp đồng hồ CLK. Nó được sử dụng để báo cho thiết bị ngoại vi biết rằng CPU đang thực hiện đọc lệnh (instruction fetch), đó là sự bắt đầu của một chu kỳ máy (machine cycle). MREQ# là tín hiệu yêu cầu truy cập bộ nhớ (memory request), IORD# là tín hiệu yêu cầu vào/ra với thiết bị ngoại vi. Khi MREQ# hoặc IORD# xuống mức tích cực "0" logic, chúng báo với bộ nhớ hoặc ngoại vi rằng Z80 đã đưa ra bus địa chỉ các tín hiệu địa chỉ (tức là đã có giá trị địa chỉ trên bus hệ thống). Tương tự, các tín hiệu RD# và WR# thực hiện để đọc và ghi dữ liệu trên bus.

Tín hiệu RFSH# là tín hiệu làm tươi bộ nhớ động (DRAM) nhằm đảm bảo không bị suy giảm mức dữ liệu (do tụ điện phóng điện khi đọc) trong từng bit của mạch DRAM.

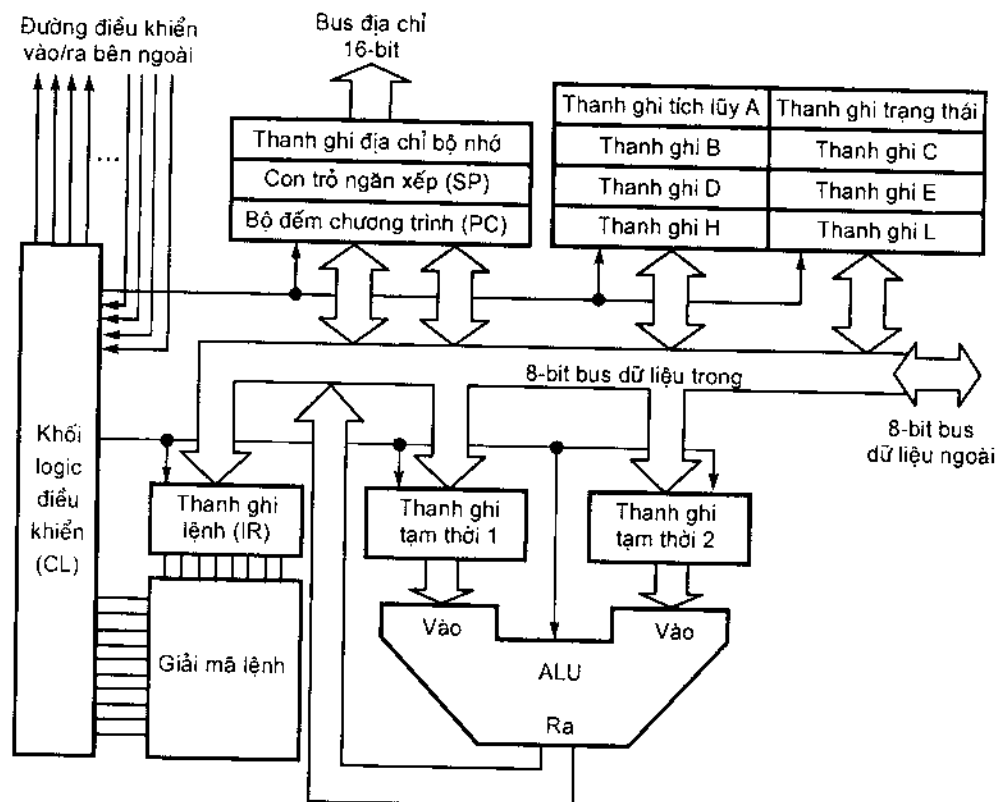
- Điều khiển đơn vị xử lý trung tâm

Khi tín hiệu ra HALT# xuống mức tích cực “0” thì đó là kết quả thực hiện lệnh HALT trong Z80, và sự thực hiện chương trình dừng lại. Z80 chỉ thực hiện các lệnh NOP (no operation) cho đến khi nào nó nhận được tín hiệu ngắt xuống mức tích cực “0”. Các tín hiệu vào ngắt là NMI# và INT#. Tiếp nhận các ngắt này, Z80 bắt đầu thực hiện chương trình con xử lý ngắt. Xung âm RESET# có độ dài tối thiểu bằng 3 chu kỳ nhịp đồng hồ để thực hiện khởi tạo hoàn toàn trạng thái ban đầu cho Z80: PC được thiết lập địa chỉ 0000h, flip-flop IFF1 = 0, véc-tơ ngắt đặt bằng 00h. Khi RESET# trở về trạng thái “1” thì Z80 bắt đầu thực hiện lệnh đầu tiên chứa ở ngăn nhớ có địa chỉ = 0000h. Tín hiệu vào WAIT# (chờ đợi) yêu cầu Z80 chờ đợi để cho phép bộ nhớ bán dẫn tốc độ truy cập chậm hoặc thiết bị ngoại vi có tốc độ vào/ra chậm có đủ thời gian để sẵn sàng dữ liệu chuyển cho Z80 hoặc tiếp nhận dữ liệu từ Z80.

- Điều khiển bus của đơn vị xử lý trung tâm

Một số thiết bị ngoại vi, như thiết bị nhớ bằng đĩa từ, có nhu cầu toàn quyền sử dụng bus hệ thống để truyền các khối dữ liệu trực tiếp với bộ nhớ RAM (truy cập trực tiếp bộ nhớ: DMA). Sự thực hiện DMA là cần thiết để đảm bảo tốc độ trao đổi nhanh dữ liệu mà không có sự can thiệp của CPU. Khi Z80 tiếp nhận từ thiết bị ngoại vi tín hiệu BUSREQ# (yêu cầu sử dụng bus) ở mức tích cực “0”, nó kết thúc chu kỳ máy đang thực hiện đang dở, sau đó đặt các đường dây địa chỉ và dữ liệu lên mức trở kháng cao và chấp nhận yêu cầu sử dụng bus của thiết bị ngoại vi bằng việc đưa ra tín hiệu BUSACK# = 0. Nhận được BUSACK# = 0, thiết bị ngoại vi bắt đầu sử dụng bus hệ thống để điều khiển trao đổi dữ liệu với RAM, trong khi vẫn duy trì BUSREQ# ở mức “0”. Khi kết thúc trao đổi dữ liệu với RAM, thiết bị ngoại vi

chuyển BUSREQ# về mức “1” để báo cho Z80 biết rằng nó trả lại quyền làm chủ bus hệ thống cho Z80. Tiếp nhận tín hiệu này Z80 đưa bus địa chỉ và dữ liệu thoát khỏi trạng thái trở kháng cao.



Hình 2.157: Sơ đồ khối cấu trúc tổng quát của vi xử lý Z80

- Nhịp đồng hồ (CLK)

Bên trong Z80 không có mạch tạo nhịp đồng hồ, vì vậy cần đến nhịp đồng hồ (CLK) được tạo ra từ mạch bên ngoài. Tùy theo các phiên bản của Z80, nhịp đồng hồ có các tần số khác nhau, như: 8 MHz cho Z80H, 6 MHz cho Z80B, 4 MHz cho Z80A, và 2,5 MHz cho Z80. Tần số nhịp càng lớn thì công nghệ chế tạo chip Z80 có ưu việt hơn và tốc độ thực hiện lệnh nhanh hơn.

- Nguồn cung cấp

Z80 cần nguồn Vcc = 5V (chân 11), và đất GND (chân 29) với dòng trong khoảng từ 150 mA đến 250 mA. Điều thường lệ ở các chip vi mạch đóng vỏ loại DIP là các chân tín hiệu nguồn và đất của Z80 nằm ở quãng giữa chip.

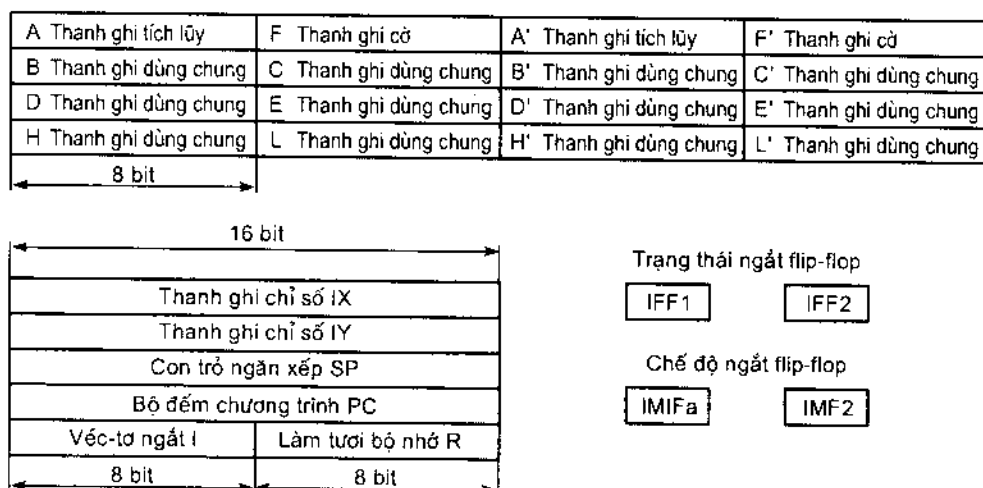
Hình 2.157 là sơ đồ khối của chip Z80. Z80 thực hiện được tất cả các lệnh của 8080 và bổ sung thêm nhiều lệnh riêng của nó nữa. Trên hệ thống Z80 có thể chạy được hoàn hảo và không có lỗi các chương trình viết cho 8080. Z80 có kiến trúc nhiều thanh ghi cho phép người lập trình có thể sử dụng các thanh ghi khác nhau để chứa dữ liệu trung gian trong khi vẫn thực hiện các thao tác khác. Những thanh ghi như vậy hợp thành một tập hợp các thanh ghi chính. Chúng ta đã trình bày về các khối chức năng và các thanh ghi trong sơ đồ khối này sơ đồ khối trong mục 2.1. Do đó ta đi sâu về mô hình lập trình của Z80.

2.5.2. Mô hình lập trình và các thanh ghi (register)

Z80 là có nhiều thanh ghi bên trong tạo cho người lập trình nhiều khả năng lưu giữ dữ liệu tạm thời trong khi vẫn thực hiện những thao tác tính toán khác. Trong mô hình lập trình của Z80 (hình 2.158) có 2 tập hợp thanh ghi tương tự nhau: tập hợp thanh ghi chính và tập hợp thanh ghi phụ. Ngoài 2 tập thanh ghi này, Z80 còn có các thanh ghi đặc biệt như: các thanh ghi chỉ số 16-bit IX và IY, bộ đếm chương trình 16-bit PC (Program Counter), con trỏ ngăn xếp 16-bit SP (Stack Pointer), thanh ghi véc-tơ ngắt 8-bit I (Interrupt vector register), và làm tươi bộ nhớ R (Refresh memory).

- Tập hợp thanh ghi chính

Tập hợp các thanh ghi chính bao gồm 8 thanh ghi 8-bit: A, F, B, C, D, E, H, L. Trong đó thanh ghi A là thanh ghi tích lũy (Accumulator), thanh ghi F là thanh ghi cờ, hay trạng thái. Các thanh ghi, B, C, D, E, H, L là thanh ghi dùng chung (general registers) cho: các lệnh số học và logic, chứa địa chỉ ngăn nhớ. Có thể dùng các cặp thanh ghi BC, DE, HL cho các dữ liệu 16-bit. Chúng ta đã xét về các thanh ghi này trong mục 2.1.



Hình 2.158: Mô hình lập trình của Z80

Thanh ghi trạng thái của Z80 có nội dung các bit được mô tả trong hình 2.159.

S	Z	X	H	X	P/V	N	C
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

S (sign): là bit dấu của số. Nếu bit MSB của kết quả (phép tính số học, logic) = 1 thì S = 1

Z (Zero): nếu kết quả = 0 thì Z = 1, nếu kết quả $\neq 0$ thì Z = 0

X = không sử dụng

H (Half carry): nếu phép cộng hoặc phép trừ gây ra một nhớ hoặc vay mượn từ bit thứ 4 thì H = 1

P/V (parity/overflow): là bit lẻ/tràn số. Khi kết quả kiểm tra là lẻ hoặc có tràn thì P/V = 1

N(negative): N = 1 khi phép toán trước đó là phép trừ

C (carry): C = 1 khi trong phép toán có nhớ từ bit cao nhất (MSB)

Hình 2.159: Thanh ghi cờ của Z80

- Tập hợp các thanh ghi phụ

Các thanh ghi này tương tự như những thanh ghi chính. Chúng cho phép người lập trình chuyển đổi nhanh chóng từ thực hiện một chương trình sang thực hiện một chương trình khác. Mỗi một chương trình có thể có tập hợp dữ liệu riêng cất giữ trong thanh ghi A, thanh ghi cờ, và các thanh ghi chung của tập hợp chính. Nếu như không có tập hợp thanh ghi phụ, thì sự chuyển đổi giữa 2 thanh ghi đòi hỏi chi phí nhiều thời gian, dòng lệnh máy và phức tạp, như, trước hết người

lập trình phải thực hiện cất giữ toàn bộ nội dung của các thanh ghi vào bộ nhớ. Sau đó, phải nạp dữ liệu của chương trình thứ hai vào các thanh ghi. Như thế đòi hỏi người lập trình phải sử dụng nhiều bước với nhiều câu lệnh máy. Nhưng nếu sử dụng tập hợp các thanh ghi phụ thì chỉ cần thực hiện lệnh đổi trao EX nội dung các thanh ghi phụ và chính (swap) chứa dữ liệu của hai chương trình cần chuyển đổi.

- Thanh ghi véc-tơ ngắt (I Interrupt vector) 8-bit

Thanh ghi này dùng để chứa byte cao của véc-tơ ngắt, mà người lập trình cần phải nạp vào trước. Điều này cần thiết khi Z80 thực hiện một loại xử lý ngắt nào đó. Byte thấp của véc-tơ ngắt được tạo ra từ thiết bị ngoại vi có yêu cầu ngắt.

- Thanh ghi làm tươi bộ nhớ động (R memory refresh) 8-bit

Nội dung của thanh ghi làm tươi được tự động tăng và đưa ra bus để làm tươi bộ nhớ động (DRAM) kết nối với vi xử lý trong từng thao tác đọc bộ nhớ. Người lập trình không điều khiển được thanh ghi này.

- Các thanh ghi chỉ số (IX và IY index registers) 16-bit

Z80 có nhiều lệnh máy sử dụng các thanh ghi chỉ số. Các thanh ghi chỉ số IX và IY được sử dụng để đánh địa chỉ theo chỉ số. Vì chúng tương tự nhau nên người lập trình có thể sử dụng một trong hai thanh ghi này tùy ý. Người lập trình có thể sử dụng hai giá trị offset 16-bit khác nhau mà không cần phải nạp lại thanh ghi chỉ số.

- Con trỏ ngăn xếp SP (Stack Pointer) 16-bit

Ngăn xếp, đó là bộ nhớ có cơ chế truy cập theo kiểu LIFO (Last-In-First-Out), nghĩa là byte dữ liệu nào ghi vào ngăn xếp cuối cùng thì sẽ được đọc ra đầu tiên. Ngăn xếp có thể là một vùng nào đó của bộ nhớ chính, hay là một mảng thanh ghi riêng biệt. Ngăn xếp luôn được truy cập ở đỉnh (TOP). Nó làm nhiệm vụ lưu trữ những thông tin phải dùng đi dùng lại nhiều lần trong quá trình thực hiện chương trình, đặc biệt là khi có nhiều chương trình con. Con trỏ ngăn xếp chứa địa chỉ của đỉnh ngăn xếp, và nó thay đổi tùy thuộc vào các lệnh tham chiếu tới ngăn xếp. Các lệnh máy cần cất giữ dữ liệu vào ngăn xếp như:

CALL (gọi tới chương trình con), PUSH (đẩy dữ liệu vào ngăn xếp), RST p (yêu cầu ngắt) kéo theo việc giảm nội dung của con trỏ ngăn xếp, (SP-1), (SP-2).

Các lệnh máy cần phục hồi dữ liệu cất giữ trong ngăn xếp như: RET (trở về từ chương trình con), POP (phục hồi từ ngăn xếp) kéo theo tăng nội dung con trỏ ngăn xếp (SP+1). Khi khởi động hệ thống máy tính, con trỏ ngăn xếp luôn được khởi tạo về địa chỉ đỉnh của ngăn xếp. Nếu nó không được khởi tạo, ngăn xếp có thể là bất kỳ vùng nào của bộ nhớ.

- Bộ đếm chương trình PC (Program Counter) 16-bit

Bộ đếm chương trình có độ dài nhiều hơn so với độ dài từ xử lý của bộ vi xử lý. Các bộ vi xử lý 8-bit thường có bộ đếm 16-bit. Một chương trình được bộ vi xử lý thực hiện phải chứa trong bộ nhớ chính (main memory). Bộ đếm chương trình chứa địa chỉ của lệnh trong bộ nhớ và chỉ ra cho bộ vi xử lý biết lệnh tiếp theo nằm ở ngăn nhớ nào để lấy ra thực hiện. Như vậy, độ dài của bộ đếm chính là khả năng đánh địa chỉ bộ nhớ chính có thể đạt được của bộ vi xử lý. Ví dụ, nếu bộ đếm có độ dài 16-bit thì ta có thể có dung lượng bộ nhớ lên tới $64 \text{ kbyte} = 2^{16}$. Nghĩa là có thể đánh địa chỉ từ 0 đến 65535 ngăn nhớ. Bộ đếm luôn được nạp địa chỉ lệnh (địa chỉ ngăn nhớ chứa lệnh máy) tiếp theo trong quá trình thực hiện bất kỳ một chương trình nào. Lệnh nào, nằm ở đâu, tùy thuộc vào trình tự thực hiện từng chương trình. Trong các bộ vi xử lý công nghệ cao có cơ chế quản lý bộ nhớ ảo (virtual memory).

Địa chỉ của ngăn nhớ chứa lệnh đầu tiên của chương trình là nội dung của bộ đếm chương trình được chuyển tới thanh ghi địa chỉ bộ nhớ (MAR) để ra bus địa chỉ, nghĩa là trên bus địa chỉ có địa chỉ của lệnh đầu tiên ở trạng thái tích cực. Bộ nhớ chính kết nối trên bus sẽ nhận được các đường địa chỉ tích cực này, giải mã chọn được ngăn nhớ chứa lệnh. Quá trình tiếp theo sẽ là đọc lệnh và thực hiện lệnh. Trong quá trình thực hiện chương trình, bộ đếm chương trình được tự động

tăng lên để trở địa chỉ của lệnh tiếp theo. Khi Z80 được xóa về trạng thái ban đầu, bộ đếm chương trình được khởi tạo giá trị = 0000h trở tới địa chỉ đầu tiên của bộ nhớ. Khi có các lệnh rẽ nhánh (nhảy, gọi tới chương trình con, ngắt) thì thứ tự tăng tự nhiên địa chỉ của các lệnh bị thay đổi và bộ đếm chương trình chứa địa chỉ của đoạn chương trình con, hay chương trình xử lý ngắt. Chỉ sau khi thực hiện xong chương trình con, hay xử lý ngắt thì địa chỉ của lệnh tiếp theo của chương trình được phục hồi trở lại từ ngăn xếp vào bộ đếm chương trình để tiếp tục thực hiện chương trình chính theo thứ tự bình thường.

- Các flip-flop phục vụ xử lý ngắt

Trong mô hình lập trình của Z80 còn có các flip-flop phục vụ xử lý ngắt, đó là flip-flop: IFF1: bit cho phép ngắt/cấm ngắt, IFF2: bit trạng thái ngắt, IMFa và IMFb: 2 bit chế độ ngắt.

Cũng như các loại vi xử lý khác, Z80 có loại ngắt: ngắt không che được NMI (Non-Maskable Interrupt) và ngắt che được INT (maskable interrupt).

Bit IFF1 không có tác dụng đối với NMI. Đầu vào ngắt NMI của Z80 khi được thiết lập ở mức tích cực "1", thì bộ đếm PC được thiết lập = 0066h và chương trình con xử lý ngắt NMI được thực hiện bắt đầu từ lệnh chứa trong ngăn nhớ 0066h. NMI thường dành cho những trường hợp khẩn cấp, ví dụ: sự cố nguồn. Bình thường NMI không được sử dụng.

Các lệnh EI, cho phép ngắt, thiết lập IFF1 = 1, cho phép các yêu cầu ngắt che được INT được thực hiện (RST p). Lệnh DI, cấm ngắt, xóa IFF1 = 0, không cho phép các yêu cầu ngắt INT được thực hiện. Nếu xuất hiện NMI thì nó được xử lý ngay. Trong khi NMI đang được thực hiện, ngắt che được đang thực hiện phải được cấm (nếu đang có). Giá trị hiện thời của IFF1 được cất trong IFF2 trong khi đang thực hiện NMI. Đến khi lệnh trở về từ chương trình xử lý ngắt NMI được thực hiện thì nội dung của IFF2 chuyển trở lại IFF1.

Ngắt che được có 3 chế độ khai thác: mode 0: $IMFa = IMFb = 0$; mode 1: $IMFa = 1, IMFb = 0$, và mode 3: $IFMa = IFMb = 1$. Tổ hợp $IFMa = 0, IFMb = 1$ không sử dụng.

Trong chế độ ngắt 0, Z80 chuyển vào một chu kỳ đọc lệnh đặc biệt, trong đó, nó chờ phản ứng điều khiển ngắt bên ngoài cung cấp lệnh RST (restart). Các lệnh RST là lệnh 1 byte (RST0-RST7). Giá trị nhị phân của số hiệu RST được sử dụng như là các bit địa chỉ A3, A4, A5. Có nghĩa là các lệnh RST bắt đầu các chương trình ngắt ở các vùng nhớ địa chỉ 0000h, 0008h, 0010h, 0018h, 0020h, 0028h, 0030h, và 0038h. Chế độ ngắt 0 cho phép Z80 khả năng thực hiện xử lý ngắt bắt đầu ở 1 trong 8 địa chỉ khác nhau trên đây. Nó tương tự như xử lý ngắt của 8080, 8085 của Intel.

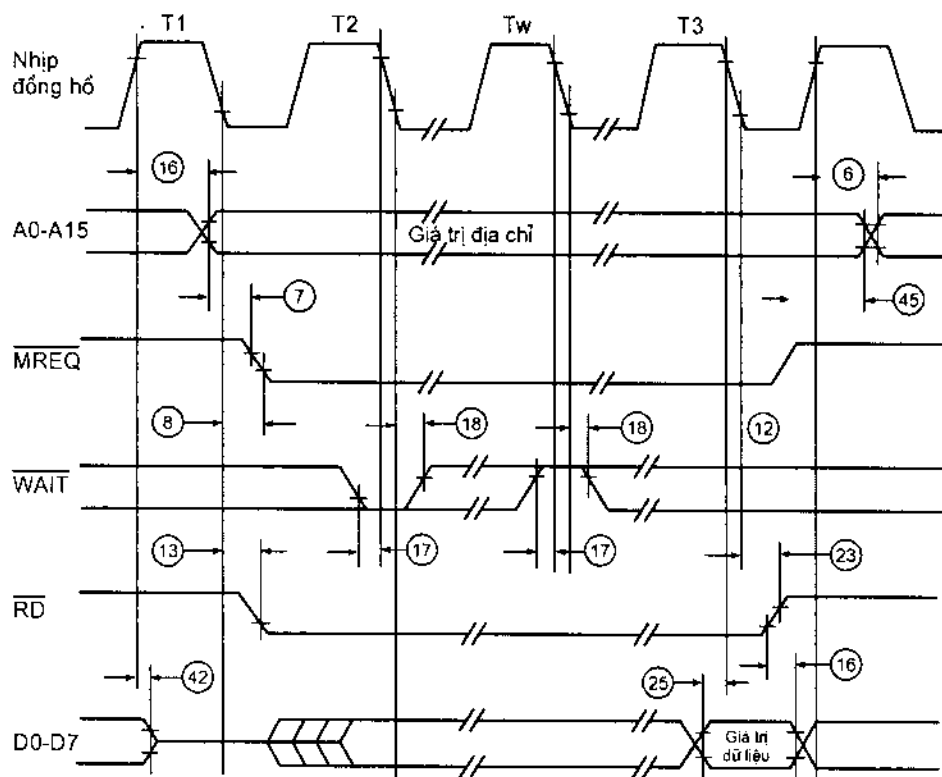
Trong chế độ ngắt 1, Z80 chỉ có 1 địa chỉ ngắt, 0038h, và không cần chờ đợi mạch điều khiển ngắt bên ngoài cung cấp lệnh RST. Ngắt này đơn giản và không có trong 8080. Ngắt chế độ 2 là ngắt thỏa mãn hơn cả, bởi vì nó cho phép Z80 xử lý ngắt bắt đầu ở bất kỳ địa chỉ nào. Ta biết rằng, 8 bit cao của véc-tơ ngắt được cung cấp bởi thanh ghi véc-tơ ngắt I, còn 8 bit thấp của véc-tơ ngắt được tạo ra từ thiết bị ngoại vi gây ngắt. Quá trình này cho phép thiết bị ngoại vi cung cấp bất kỳ giá trị véc-tơ ngắt nào. Mỗi véc-tơ ngắt gán với địa chỉ đầu của chương trình xử lý ngắt.

2.5.3. Tập lệnh và các kiểu đánh địa chỉ của Z80

Tập lệnh của Z80 gồm 158 lệnh, trong đó có 78 lệnh của Intel 8080. Tất cả các lệnh được phân thành 11 loại khác nhau, 11 loại lệnh này được liệt kê trong phần Phụ lục 3.

2.5.4. Định thời của Z80

Hình 2.160 là đồ thị thời gian đọc nội dung bộ nhớ kết nối trên bus hệ thống của Z80. Quãng thời gian chờ T_w đảm bảo cho bộ nhớ chậm đủ thời gian giải mã chọn ngân nhớ được địa chỉ (A0-A15). Khi dữ liệu đọc từ bộ nhớ đã có giá trị trên bus dữ liệu (valid data) thì $MREQ\#$ và $RD\#$ trở về mức "1" không tích cực.



Ghi chú: (X) : Giá trị thời gian

Hình 2.160: Đồ thị định thời gian đọc nội dung bộ nhớ của Z80

2.5.5. Các vi mạch lập trình hỗ trợ cho Z80

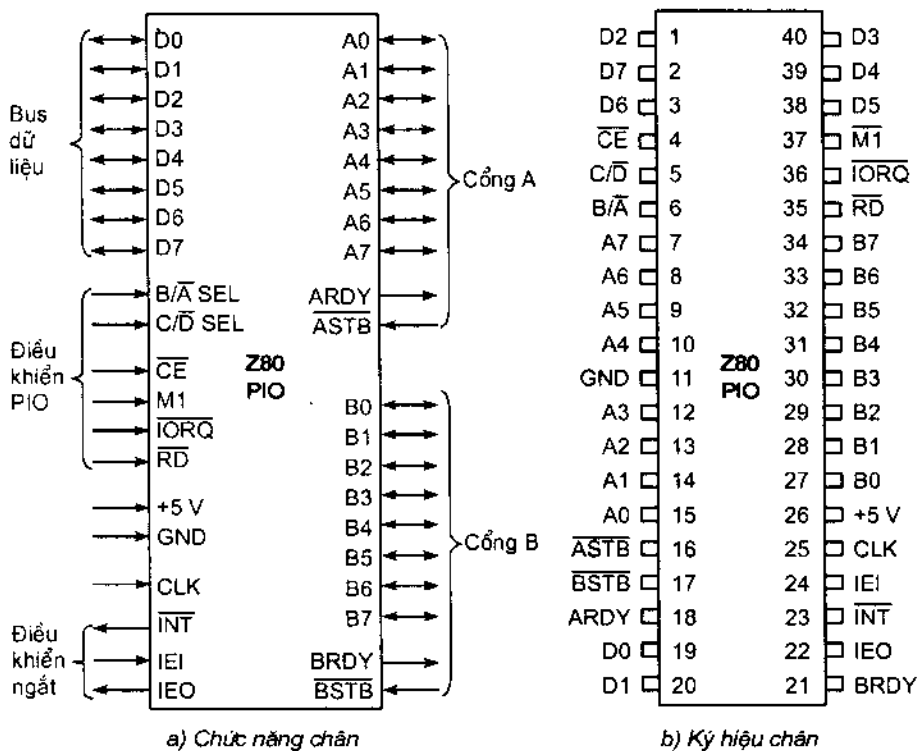
Để kết nối thành một hệ thống vi xử lý tối thiểu gồm CPU, RAM, ROM (hoặc EPROM), các cổng vào/ra song song và tuần tự, bộ đếm thời gian (timer/counter), Z80 cần có một số chip vi mạch lập trình hỗ trợ, đó là Z80PIO, Z80 SIO (điều khiển vào ra tuần tự), Z80CTC (mạch đếm thời gian).

(1) Điều khiển vào/ra song song Z80PIO

Z80PIO được thiết kế để kết nối trực tiếp với bus dữ liệu và các tín hiệu điều khiển (hình 2.161).

PIO có hai cổng vào/ra 8-bit song song 2 chiều (cho cả nhận và đưa ra dữ liệu) với các đường tín hiệu giao diện kiểu bắt tay (handshaking)

là: cổng A: $\overline{\text{ASTB\#}}$, ARDY , và cổng B: $\text{BSTB\#}$, BRDY . Người lập trình có thể thiết lập từng chân dữ liệu của cổng A (A0-A7), cổng B (B0-B7) như là một đầu nhận dữ liệu vào hay đưa ra dữ liệu. Nếu tín hiệu $\overline{\text{ASTB\#}}$ từ mạch ngoài đến PIO chuyển xuống mức tích cực “0”, thì nó báo cho PIO cần phải nhận dữ liệu vào cổng A. Tương tự, đầu vào $\text{BSTB\#}$ xuống mức “0” báo cho PIO nhận dữ liệu ở cổng B. Tín hiệu ARDY (hoặc BRDY) chuyển lên mức tích cực “1” là báo cho PIO biết rằng đã có giá trị dữ liệu ở cổng A (hoặc cổng B).



Hình 2.161: Đóng vỏ và các tín hiệu của PIOZ80

Trước khi có thể sử dụng PIO để đọc hoặc đưa ra dữ liệu, cần phải gửi ra PIO các tín hiệu điều khiển, bởi vì PIO có thể được lập trình làm việc ở nhiều chức năng khác nhau. Như vậy, PIO trước hết cần phải được thiết lập chế độ điều khiển, và sau đó là chế độ dữ liệu. Sự chọn lựa PIO ở trong chế độ điều khiển hay chế độ dữ liệu được

thực hiện bằng tín hiệu cào C/D# (Control/Data). Khi C/D# = "1" thì PIO ở vào chế độ được điều khiển, ngược lại, khi C/D# = "0", thì PIO ở chế độ đọc hoặc đưa ra dữ liệu.

Để chọn cổng A hay B, có chân tín hiệu B/A#. Như thế, logic giải mã chọn phải thực hiện tạo ra 4 địa chỉ chọn cho 2 đường vào chọn của PIO là C/D# và B/A#.

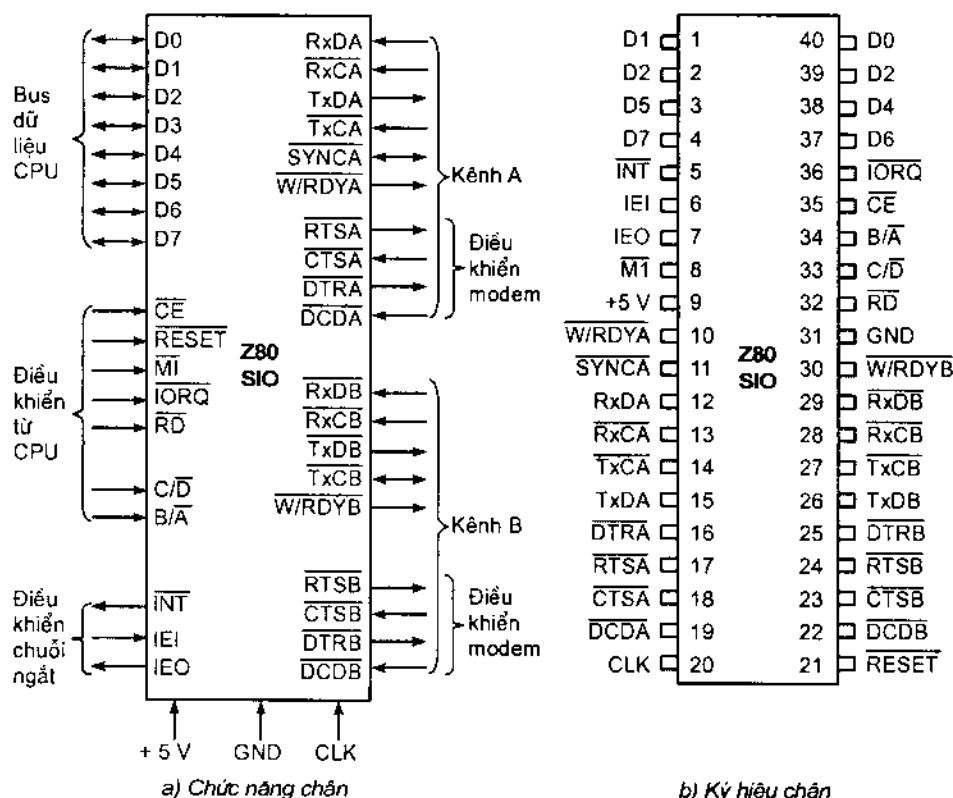
Trong chế độ điều khiển, có thể gửi cho PIO một loại byte qua bus dữ liệu (D0-D7) để điều khiển PIO vận hành và tạo ra tín hiệu ngắt như thế nào. Ví dụ, PIO có thể tạo ra tín hiệu ngắt INT# = "0" khi nó nhận được ở đầu vào ASTB# (hoặc BSTB#) = "0", hay có thể tạo ra ngắt khi một trong bit dữ liệu ở cổng A (hoặc cổng B) được dữ liệu vào. PIO được lập trình để gửi một byte, như là byte thấp của véc-tơ ngắt chế độ 2 khi yêu cầu ngắt của PIO được phục vụ.

PIO có 2 chân tín hiệu riêng dành cho thiết lập chuỗi ngắt (interrupt daisy-chain), đó là: IEI (Interrupt Enable Input) và IEO (Interrupt Enable Output). Một chuỗi ngắt gồm các chip PIO kết nối với nhau bằng các chân tín hiệu IEO → IEI. Mỗi PIO có thể kết nối với các mạch (thiết bị) vào/ra. Tất cả các chân tín hiệu INT# của các PIO được nối chung với nhau và đưa vào chân INT# của Z80. Các nối này gọi là nối dây logic OR (hoặc). Nếu PIOA hoặc PIOB hoặc PIOC tạo ra một tín hiệu ngắt, thì Z80 đều nhận được. Nếu PIOA tạo ra ngắt thì nó cũng đưa tín hiệu ra chân IEO, và qua chuỗi ngắt đến chân IEI của PIOB. Vì PIOB nhận được tín hiệu tại IEI, nên tín hiệu INT# của B bị cấm, và PIOB cũng đưa ra tín hiệu ở IEO. IEO của PIOB tiếp tục cấm PIOC gây ra ngắt. Như vậy, khi PIOA khởi tạo ngắt thì thông qua chuỗi ngắt nó cấm PIOB, PIOC gây ra ngắt. Nếu PIOA không gây ra ngắt thì khi PIOB tạo ra tín hiệu ngắt sẽ cấm PIOC gây ra ngắt. Nếu PIOA tạo ra tín hiệu ngắt trong khi ngắt của PIOB đang được xử lý, thì Z80 phải tiếp nhận xử lý ngắt cho PIOA, bởi vì trong chuỗi ngắt, PIOA có mức ưu tiên được phục vụ ngắt cao nhất (PIOA

đứng gần CPU nhất trong chuỗi). Trong chuỗi ngắt, các thiết bị vào/ra có tốc độ cao nhất thường được đặt ở đầu chuỗi, gần CPU nhất, nơi có mức ưu tiên cao nhất. Quá trình làm việc của PIO được định thời bởi nhịp đồng hồ hệ thống CLK.

(2) Điều khiển vào/ra tuần tự Z80SIO

SIO là một chip vi mạch lập trình để điều khiển thu-phát dữ liệu đồng bộ-không đồng bộ đa năng USART (Universal Synchronous-Asynchronous Receiver-Transmitter). Cũng giống như SIO có 2 cổng vào/ra tuần tự: cổng A, và B kết nối với truyền thông, trong đó có các tín hiệu điều khiển modem (modem control) (hình 2.162).



Hình 2.162: Đóng vỏ và các tín hiệu của SIOZ80

SIO kết nối dữ liệu với Z80 thông qua bus dữ liệu (D0-D7). Nó chuyển đổi dữ liệu nhận song song được từ bus dữ liệu Z80 thành

chuỗi tuần dữ liệu tuần tự theo bit cho cổng phát dữ liệu TxDA# (cổng A) hoặc TxDB# (cổng B). Ngược lại, SIO nhận dữ liệu tuần tự từ đường truyền thông ở chân RxDA (cổng A) hay ở chân RxDB (cổng B). Dữ liệu tuần tự được đồng bộ với các nhịp TxCA# (phát cổng A), RxCA# (nhận cổng A) và TxCB# (phát cổng B), RxCB# (nhận cổng B).

Cũng giống như PIO, SIO có các tín hiệu dành cho điều khiển từ CPU và kết nối chuỗi ngắt IEI, IEO, INT#. Tổ chức xử lý theo chuỗi ngắt cũng giống như PIO. Để chọn chế độ điều khiển hoặc dữ liệu cho SIO có chân tín hiệu C/D# và để chọn cổng A và B có chân tín hiệu A/B#. SIO được định thời bằng nhịp đồng hồ hệ thống CLK.

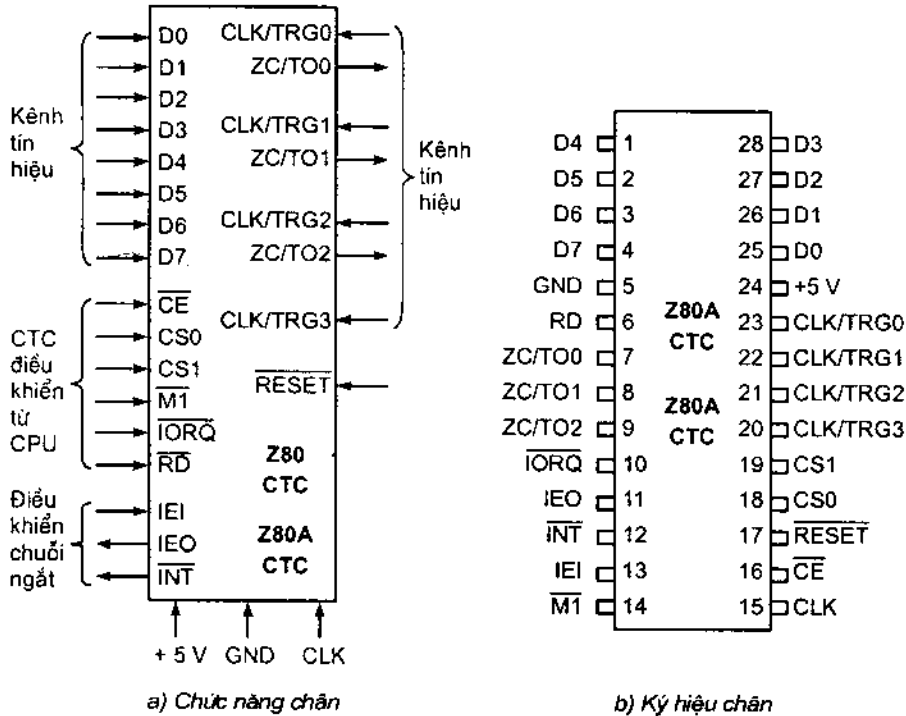
(3) Điều khiển đếm - định thời gian Z80CTC

CTC là chip lập trình đếm - định thời gian (hình 2.163). Nó có 4 kênh lập trình đếm - định thời gian. Các chip đếm và định thời gian thường được sử dụng trong các hệ thống vi xử lý phục vụ cho nhiều quá trình điều khiển bên trong hệ thống.

Mỗi một kênh đếm - định thời gian có thể được lập trình riêng biệt và chia nhỏ từ 1 đến 255. Chúng có thể làm việc ở một trong hai chế độ khác nhau: chế độ thời gian và chế độ đếm. Trong chế độ thời gian, các kênh gây ra một ngắt sau khi đã đếm xong một quãng thời đã được lập trình trước. Trong chế độ đếm các kênh chia tần số của tín hiệu vào cho giá trị n ($n = 1 \dots 255$). Có các số chia định trước: 1, 16 hoặc 64 để chia cho tín hiệu vào. Bộ đếm của CTC có thể được sử dụng để tạo ra các nhịp đồng hồ tần số thấp từ nhịp đồng hồ hệ thống của Z80.

Cũng tương tự như PIO và SIO, CTC giao tiếp dữ liệu với Z80 thông qua bus dữ liệu (D0-D7). Nó có các chân tín hiệu cho điều khiển từ CPU. Các chân tín hiệu IEI, IEO và INT# để kết nối chuỗi ngắt. Trong một hệ thống kết nối vi xử lý, các chip PIO, SIO, và CTC được giải mã chọn bằng các địa chỉ vào/ra của hệ thống. Khi tổ hợp chọn địa chỉ của một chip thỏa mãn, thì tín hiệu vào chọn chip tương ứng

CE# xuống mức tích cực “0”. Chỉ trong khoảng thời gian CE# = “0”, các tín hiệu chọn cổng A/# và chọn chế độ C/D# mới có hiệu lực.



Hình 2.163: Đóng vỏ và các tín hiệu của CTCZ80

2.6. HỌ VI XỬ LÝ MOTOROLA M68000

2.6.1. Đặc điểm của họ vi xử lý M68000

Họ vi xử lý Motorola M68000 được ứng dụng nhiều để chế tạo các thế hệ máy tính cá nhân có chức năng mạnh về đồ họa cho các ứng dụng CAD/CAM, các máy tính chuyên dụng dùng trong y tế, công nghiệp, tự động điều khiển, các tổng đài chuyển mạch trong truyền thông. M68000 được dùng làm CPU cho các họ máy vi tính Apple's Macintosh. Họ M68000 là một trong những sản phẩm cạnh tranh mạnh với Intel. Đặc tính cơ bản của chúng được trình bày ở bảng 2.23.

Bảng 2.23: Các đặc tính của họ vi xử lý M6800

Đặc tính	68000	68010	68020	68030	68040	68060
Bus dữ liệu	16-bit	16-bit	8,16,32-bit	8,16,32-bit	32-bit	32-bit
Bus địa chỉ	24-bit	24-bit	32-bit	32-bit	32-bit	32-bit
Tốc độ vận hành, MHz	8, 10, 12, 16	8, 10, 12	16, 20, 25, 33	16, 20, 25, 33, 40, 50	25, 40	50, 66
MIPS	2,4	-	6,5	12	39	> 100
MFLOPS	-	-	0,25	0,5	3,5	12
Cache lệnh	-	3 words	256 byte direct	256 byte direct	4 kbyte 4-đường	8 kbyte 4-đường
Cache dữ liệu	-	-	-	256 byte direct	4 kbyte 4-đường	8 kbyte 4-đường
FPU	68881 Bên ngoài	68881 Bên ngoài	68881 Bên ngoài	68882 Bên ngoài	Trong chip	Trong chip
MMU	Bên ngoài	Bên ngoài	Bên ngoài	Trong chip	Trong chip	Trong chip
Không gian nhớ vật lý	16 MB	16 MB	4 GB	4 GB	4 GB	4 GB
Đóng gói DIP	64 chân	64 chân	N/A	N/A	N/A	N/A
Đóng gói PGA	68 chân	68 chân	114 chân	128 chân	179 chân	223 chân
Đóng gói QFP	68 chân	68 chân	132 chân	132 chân	-	-

Vi xử lý 68000 là sản phẩm gốc, có tốc độ 2,4 MIPS, có kiến trúc 32-bit bên trong, 16-bit bus dữ liệu và với 24-bit bus địa chỉ có thể đánh địa chỉ tới 16 MB bộ nhớ vật lý. Chip 68000 dùng FPU và MMU bên ngoài. Chip 68010 có bổ sung chức năng đánh địa chỉ ảo. Chip 68020 có tốc độ 6,5 MIPS, có 32-bit bus dữ liệu trong khi có thể dùng các chế độ 8, 16-bit bus dữ liệu, đặc biệt cho phép kết nối đến 8 FPU bên ngoài và có 256 byte Icache bên trong chip. Chip 68030 là vi xử lý 32-bit, có tốc độ 12 MIPS, MMU, 256-byte Dcache và 256-byte Icache bên trong. Chip 68040 có tốc độ 39 MIPS, 4096-byte Dcache và 4096-byte Icache bên trong, bổ sung FPU bên trong. Chip 68060 có tốc độ hơn 100 MIPS, 8-kbyte Dcache, 8-kbyte Icache, FPU và MMU bên trong. Với 32-bit bus địa chỉ các loại vi xử lý 68020-68060 có thể địa chỉ đến 4 GB nhớ vật lý. Motorola là một trong những nhà sản xuất

các chip vi xử lý đầu tiên đưa vào kỹ thuật cache đôi (Icache, Dcache) trong các loại vi xử lý loại CISC.

Kiến trúc của tất cả các loại vi xử lý họ M68000 32-bit có đặc điểm riêng không giống với kiến trúc của các loại vi xử lý khác. Ngay chip đầu tiên 68000 là vi xử lý 16-bit đầy đủ khác với Intel 8088. Vi xử lý 68000 có 16-bit bus dữ liệu đầy đủ, 32-bit bus trong, và các thanh ghi bên trong 32-bit. Số lượng các thanh ghi 32-bit bên trong của M68000 không ít, do đó, chúng gộp lại chung thành tập hợp các thanh ghi.

Tập hợp các thanh ghi (register file) của kiến trúc 68000 được chia thành 2 phần chính: mô hình lập trình của người sử dụng (user programming model) và mô hình lập trình của người giám sát hệ thống (supervisor programming model).

Các loại vi xử lý của họ M68000 vận hành trong 2 chế độ thẩm quyền: người sử dụng (user) và người giám sát hệ thống (supervisor). Chế độ giám sát có mức thẩm quyền cao hơn chế độ sử dụng và được dùng cho những người lập trình hệ thống để can thiệp sâu đến các chức năng của hệ điều hành. Người lập trình hệ thống sử dụng thẩm quyền cao để giám sát và can thiệp vào cả các tài nguyên khác của từng người sử dụng. Chế độ sử dụng có mức thẩm quyền thấp và dành riêng cho từng người sử dụng. Một chương trình chạy ở chế độ sử dụng không thể truy cập tới bất kỳ thanh ghi nào trong mô hình lập trình của người giám sát hệ thống, ngược lại chương trình chạy trong chế độ giám sát có thể truy cập đến tất cả các thanh ghi của cả 2 mô hình lập trình.

2.6.2. Mô hình lập trình của người sử dụng

Mô hình lập trình của người dùng gồm có (hình 2.164):

- Phần nguyên (Integer part): cho tất cả họ M68000: 16 thanh ghi chung 32-bit (8 thanh ghi dữ liệu D0-D7, 8 thanh ghi địa chỉ A0-A7),

Thanh đếm chương trình 32-bit (PC), Thanh ghi mã điều kiện 8-bit CCR (Condition code register).

- Phần dấu phẩy động (Floating-point part): cho các bộ đồng xử lý 68881 hoặc 68882 và cho 68040-68060 với FPU bên trong: 8 thanh ghi dữ liệu dấu phẩy động 80-bit (FP0-FP7), thanh ghi điều khiển dấu phẩy động 8-bit (FPCR), thanh ghi trạng thái dấu phẩy động 32-bit (FPSR), thanh ghi địa chỉ lệnh dấu phẩy động 32-bit (FPIAR).

(1) Các thanh ghi dữ liệu D0-D7

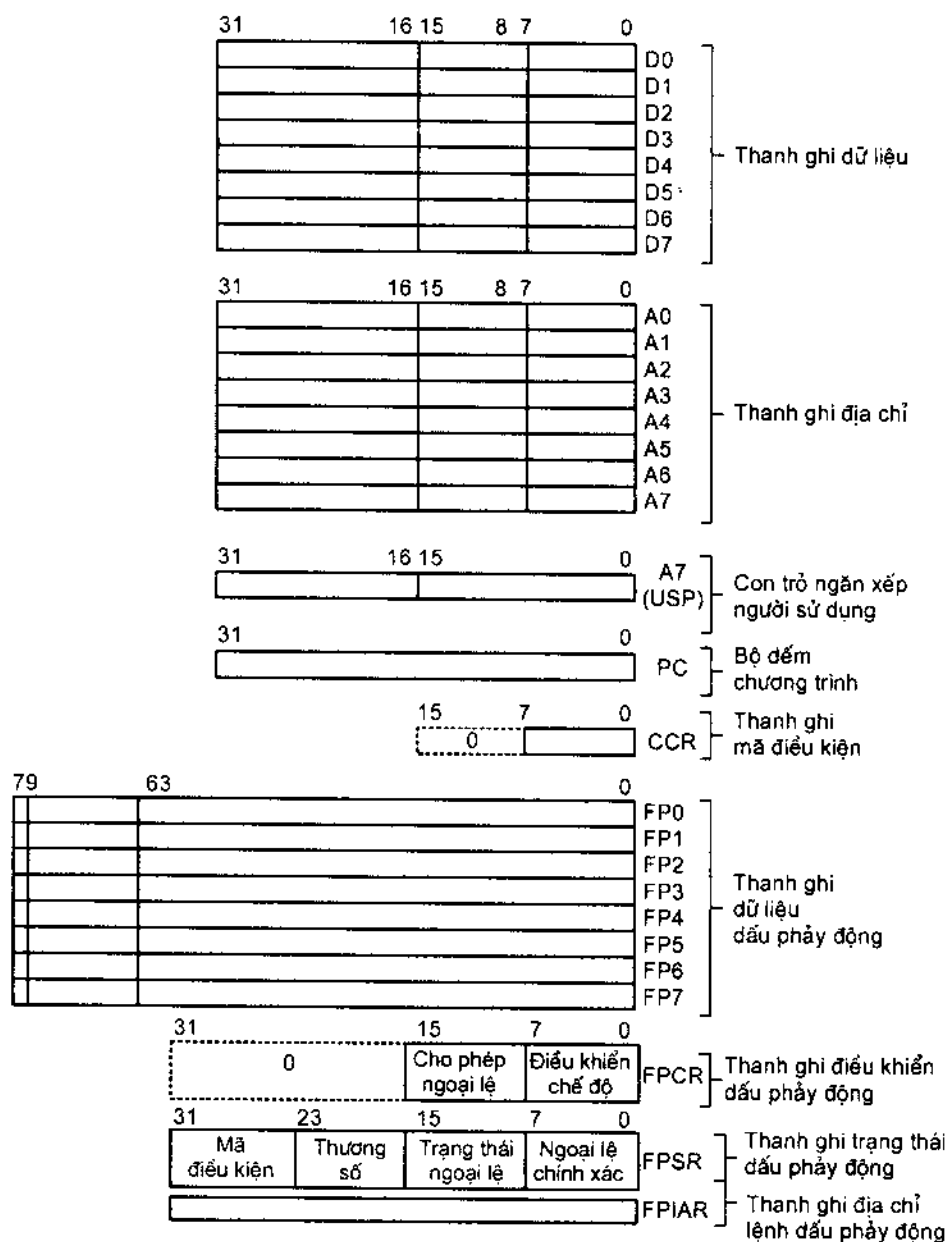
Chúng được dùng để xử lý dữ liệu theo từng bit và trường bit (1-32), BCD (4-bit), byte (8-bit), từ (16-bit), từ dài - longword (32-bit), hoặc 4 từ (64-bit). Chúng có thể được dùng như các thanh ghi chỉ số (index registers). Khi thực hiện phép xử lý theo byte, thì byte thấp nhất LSB (7-0) của thanh ghi được dùng, còn lại 24 bit cao không dùng. Lệnh xử lý theo từ thực hiện với từ thấp LSW (bit 15-0) và 16 bit cao còn lại không sử dụng.

(2) Các thanh ghi địa chỉ A0-A7

Chúng được dùng như những con trỏ ngăn xếp của chương trình, các thanh ghi chỉ số, các thanh ghi địa chỉ cơ sở hay gián tiếp để tính địa chỉ vùng nhớ. Chúng có thể được dùng cho các phép toán xử lý từ hay từ dài. Thanh ghi A7 dùng như con trỏ ngăn xếp cứng (SP) khi thực hiện các lệnh gọi đến chương trình con và xử lý ngoại lệ (exemption handling).

(3) Thanh đếm chương trình PC

PC chứa địa chỉ của lệnh tiếp theo sẽ phải được đọc ra từ bộ nhớ để thực hiện. Nó được tự động tăng lên địa chỉ lệnh tiếp theo sau khi lệnh hiện thời được đọc từ bộ nhớ vào vi xử lý (fetch). Nó cũng được dùng như con trỏ trong chế độ đánh địa chỉ tương đối với PC (PC-relative addressing).



Hình 2.164: Mô hình lập trình của người sử dụng của họ M68000

(4) Thanh ghi mã điều kiện CCR

CCR là byte thấp của thanh ghi trạng thái SR mà ta sẽ xét sau này.

(5) Các thanh ghi dữ liệu dấu phẩy động FP0-FP7

Chúng luôn chứa các số chính xác mở rộng 80-bit. Tất cả các toán hạng, tuân thủ tiêu chuẩn chính xác, đều được chuyển đổi thành giá trị chính xác mở rộng 80-bit trước khi được đưa vào tham gia tính toán hoặc cất trong thanh ghi dữ liệu dấu phẩy động.

(6) Thanh ghi điều khiển dấu phẩy động FPCR

FPCR chứa byte cho phép ngoại lệ (exception enable byte), nó cho phép hay loại bỏ các bẫy (traps) đối với từng kiểu ngoại lệ của dấu phẩy động, và byte điều khiển chế độ (mode control byte):

	15	14	13	12	11	10	9	8
FPCR	BSUN	SNAN	OPERR	OVFL	UNFL	DZ	INEX2	INEX1

Byte cho phép ngoại lệ của thanh ghi FPCR

BSUN: Branchset on unordered (Thiết lập rẽ nhánh không theo thứ tự)

SNAN: Signaling not a number (Bảo hiệu không phải số)

OPERR: Operand error (Lỗi toán hạng)

OVFL: Overflow (Tràn trên)

UNFL: Underflow (Tràn dưới)

DZ: Divide zero (Lỗi chia cho số 0)

INEX2: Inexact operation (Phép toán không chính xác)

INEX1: Inexact decimal input (Đưa vào số hệ 10 không chính xác).

	7	6	5	4	3	2	1	0
FPCR	PREC		RND		0	0	0	0

Byte điều khiển chế độ của thanh ghi FPCR

RND: Round mode (chế độ làm tròn): 00- về phía gần nhất, 01- về phía 0, 10- về phía trừ ban đầu, 11- về phía cộng ban đầu.

PREC: Rounding Precision (chính xác bằng cách làm tròn): 00- chính xác mở rộng, 01- chính xác đơn, 10- chính xác kép, 11- dự trữ.

(7) Thanh ghi trạng thái dấu phẩy động FPSR

FPSR chia ra 4 byte, trong đó:

Bit 31-24: Condition Code (Mã điều kiện)

Bit 23-16: Quotient (Thương số)

Bit 15-8 : Exception status (Trạng thái ngoại lệ)

Bit 7-0 : Accrued Exception (Ngoại lệ chính xác).

Byte mã điều kiện của FPSR:

Mã điều kiện của thanh ghi FPSR gồm có 4 bit điều kiện, mà chúng được thiết lập khi kết thúc thực hiện các lệnh số học xử lý dấu phẩy động:

	31	30	29	28	27	26	25	24
FPSR	0				N	Z	I	NAN

Byte mã điều kiện của thanh ghi FPSR

Bit 24: Not a number or unordered (không phải số hoặc không theo thứ tự)

Bit 25: Infinity (Vô cùng)

Bit 26: Zero (Số không)

Bit 27: Negative (Số âm).

Byte mã trạng thái ngoại lệ của thanh ghi FPSR:

	15	14	13	12	11	10	9	8
FPCR	BSUN	SNAN	OPERR	OVFL	UNFL	DZ	INEX2	INEX1

Byte cho phép loại trừ của thanh ghi FPCR

Các bit của byte này có thể xuất hiện trong hầu hết các phép tính số học dấu phẩy động và vận chuyển. Chúng có thể được sử dụng để cho từng xử lý ngoại lệ để xác định xem trong phép ngoại lệ dấu phẩy động, phép nào gây ra bẫy (trap).

Bit 8: Inexact decimal input (Vào không đúng số hệ mười)

Bit 9: Inexact operation (Phép toán sai)

Bit 10: Divide by Zero (Chia cho số 0)

Bit 11: Underflow (Tràn dưới)

Bit 12: Overflow (Tràn trên)

Bit 13: Operand error (Sai toán hạng)

Bit 14: Signalling not a number (Thông báo không phải là số)

Bit 15: Branch/set on unordered (Rẽ nhánh/đặt không theo thứ tự).

Byte ngoại lệ chính xác của thanh ghi FPSR:

	7	6	5	4	3	2	1	0
FPCR	IOP	OVFL	UNFL	DZ	INEX			

Byte ngoại lệ chính xác của thanh ghi FPCR

- Bit 3: Inexact (Không chính xác)
- Bit 4: Device by Zero (Chia cho số 0)
- Bit 5: Underflow (Tràn dưới)
- Bit 6: Overflow (Tràn trên)
- Bit 7: Invalid operation (Phép toán sai).

Tất cả các bit của thanh ghi FPSR có thể được đọc hoặc ghi nhờ người sử dụng. Chức năng RESET xóa nội dung FPSR.

(8) Thanh ghi địa chỉ lệnh dấu phẩy động FPIAR

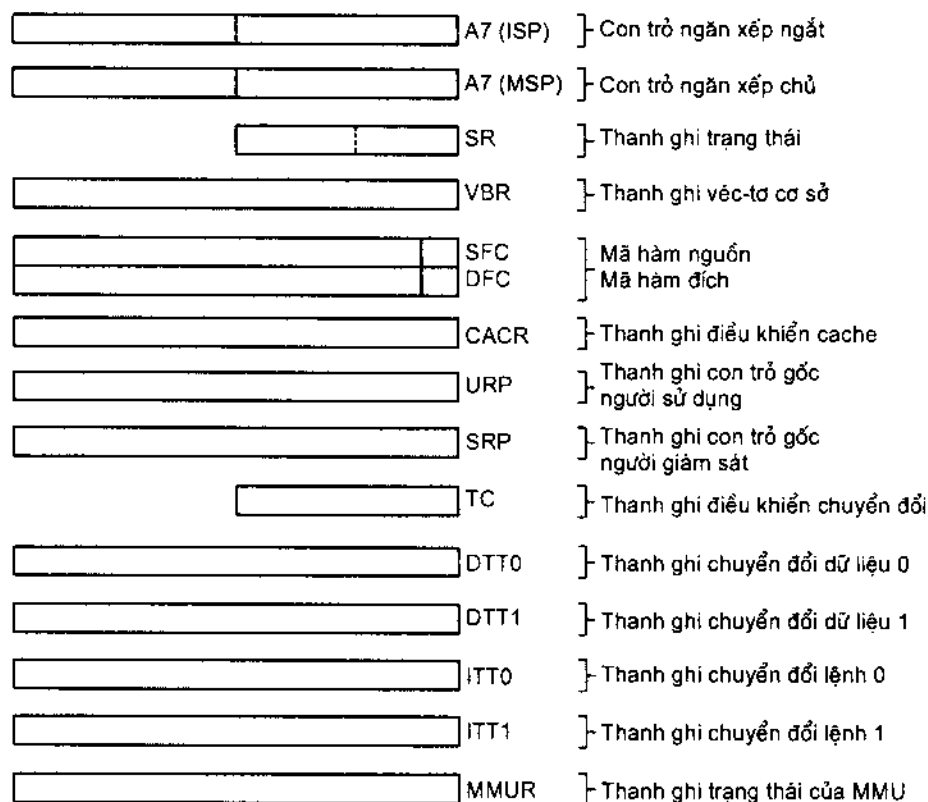
FPIAR được nạp địa chỉ logic của lệnh dấu phẩy động trước khi lệnh được thực hiện. Địa chỉ này có thể sau đó được sử dụng cho xử lý loại trừ dấu phẩy động để tìm lệnh dấu phẩy động nào đó phát sinh một loại trừ. FPIAR được xóa nhờ lệnh xóa (RESET).

2.6.3. Mô hình lập trình của người giám sát

Mô hình lập trình của người giám sát (hình 2.165) gồm các thanh ghi của mô hình lập trình của người sử dụng mà ta đã xét ở trên và các thanh ghi điều khiển sau đây:

- 2 con trỏ ngăn xếp 32-bit: ngắt và chủ (ISP và MSP),
- Thanh ghi trạng thái 16-bit (SR),
- Thanh ghi véc-tơ cơ sở 32-bit (VBR),
- 2 thanh ghi mã chức năng: mã chức năng nguồn (SFC) và mã chức năng đích (DFC),
- Thanh ghi điều khiển cache 32-bit (CACR),
- Con trỏ gốc của người sử dụng 32-bit (URP),
- Con trỏ gốc của người giám sát 32-bit (SRP),
- Thanh ghi điều khiển chuyển đổi 16-bit (TC),
- 2 thanh ghi chuyển đổi dữ liệu 32-bit (DTT0, DTT1),
- 2 thanh ghi chuyển đổi lệnh 32-bit (ITT0, ITT1),
- Thanh ghi trạng thái của MMU 32-bit (MMUSR),

Các thanh ghi URP, SRP, TC, DTT0, DTT1, ITT0, ITT1 và MMUSR sẽ được khảo sát trong phần MMU. Thanh ghi CACR sẽ xét ở phần Cache. Các thanh ghi ISP, MSP và VBR sẽ được xét ở phần xử lý ngoại lệ.



Hình 2.165: Mô hình lập trình của người giám sát hệ thống

Thanh ghi trạng thái SR:

SR cất giữ trạng thái của bộ xử lý, và có nội dung sau:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FPCR	T1	T0	S	M	0	I2	I1	I0	0	0	0	X	N	Z	V	C

Thanh ghi trạng thái SR

Bit 0 (C): Carry (Nhớ)

Bit 1 (V): Overflow (Tràn)

Bit 2 (Z): Zero (Giá trị 0)

Bit 3 (N): Negative (Giá trị âm)

Bit 4 (X): Extend (Mở rộng)

Bit 8, 9, 10 (I0, I1, I2): Interrupt priority mask (Mặt nạ ưu tiên ngắt)

Bit 12 (M): Master/interrupt state (Trạng thái chủ/ngắt).

$M = 0 \rightarrow \text{ISP}; M = 1 \rightarrow \text{MSP}$

Bit 13 (S): Supervisor/user state (trạng thái giám sát/sử dụng).

$S = 0 \rightarrow \text{user}; S = 1 \rightarrow \text{Super.}$

Bit 14, 15 (T0, T1): Trace enable (có thể thực hiện theo tiến trình)

T1	T2	Thao tác Trade
0	0	Không có Trade
0	1	Trace theo thay đổi của dòng lệnh
1	0	Trace theo thực hiện lệnh
1	1	Dự phòng

Toàn bộ nội dung của SR chỉ có thể truy cập được trong chế độ giám sát.

2.6.4. Dạng dữ liệu

Họ M68000 sử dụng kiểu dữ liệu với các qui ước sau đây cho các phép toán số học:

B - Byte interger, 8-bit (Byte nguyên)

W - Word interger, 16-bit (Từ nguyên)

L - Longword interger, 32-bit (Từ dài nguyên)

S - Single-precision real, 32-bit (Số thực chính xác đơn)

D - Double-precision real, 64-bit (Số thực chính xác kép)

X - eXtended-precision real, 80-bit (Số thực chính xác mở rộng).

Ba khuôn dạng dữ liệu nguyên là chung cho cả hai đơn vị tính IU và FPU (byte, word, và longword) và được biểu diễn diễn ở dạng mã bù 2 chuẩn trong kiến trúc của họ M68000. Khi một số nguyên được sử dụng trong phép toán dấu phẩy động, thì số nguyên được tự động

chuyển đổi thành số chính xác mở rộng của FPU trước khi được đưa vào sử dụng. Bảng 2.24 tổng hợp tất cả các kiểu dữ liệu mà họ M68000 sử dụng.

Bảng 2.24: Các kiểu dữ liệu hệ M68000

Loại dữ liệu của toán hạng	Kích thước, bit	Đơn vị xử lý	Ghi chú
Bit	1	IU	-
Trường bit	1-32	IU	Các bit liên tục nhau
BCD (hệ 2 10)	8	IU	Nhóm: 1 byte có 2 chữ số digit, mỗi digit 4 bit
			Không nhóm: 1 byte có 1 digit
Byte nguyên (integer)	8	IU, FPU	-
Từ nguyên	16	IU, FPU	-
Từ dài nguyên	32	IU, FPU	-
Từ gấp 4 nguyên	64	IU	Bất kỳ 2 thanh ghi dữ liệu nào
16-byte	128	IU	Chỉ là ô nhớ, chính theo ranh giới 16-byte
Chính xác đơn-thực	32	FPU	1-bit dấu, 8-bit số mũ, 23-bit phần định trị
Chính xác kép-thực	64	FPU	1-bit dấu, 11-bit số mũ, 52-bit phần định trị
Chính xác mở rộng-thực	80	FPU	15-bit số mũ, 64-bit phần định trị

2.6.5. Các chế độ đánh địa chỉ

Họ M68000 có 18 chế độ đánh địa chỉ và phân thành 4 loại đánh địa chỉ như sau:

- Dữ liệu (Data): tham chiếu cho toán hạng dữ liệu
- Bộ nhớ (Memory): tham chiếu cho toán hạng ô nhớ
- Thay đổi (Alterable): tham chiếu cho phép ghi
- Điều khiển (Control): qui chiếu tới các toán hạng ô nhớ mà không gán theo dung lượng (size)

Những phân loại này đôi khi kết hợp lại, ví dụ: thay đổi dữ liệu, thay đổi bộ nhớ.

2.6.6. Tập lệnh của họ M68000

Tập lệnh của họ vi xử lý M68000 được phân chia thành các nhóm đặc trưng sau đây:

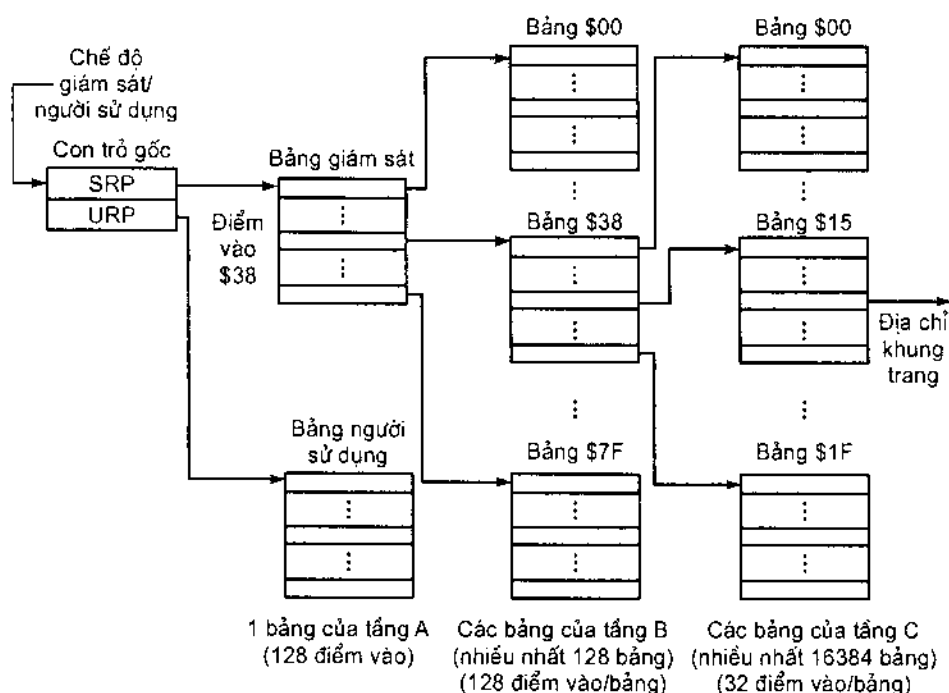
- Data movement (Chuyển dữ liệu)
- Integer arithmetic (Các phép toán số học với các số nguyên)
- Floating-point arithmetic (Các phép toán số học với các số dấu phẩy động)
- Logical (Các phép toán logic)
- Shift và rotate (Dịch và quay vòng)
- Bit manipulation (Xử lý bit)
- Bit-field manipulation (Xử lý theo trường các bit)
- BCD arithmetic (Các phép toán số học với số hệ nhị phân - thập phân)
- Program control (Điều khiển chương trình)
- System control (Điều khiển hệ thống)
- Memory management (Quản lý bộ nhớ)
- Cache maintenance (Bảo dưỡng bộ nhớ cache)
- Multiprocessor communications (Truyền thông đa xử lý).

2.6.7. Quản lý bộ nhớ

Một trong những đặc tính nổi bật của họ M68000 bắt đầu bằng 68040 là có đơn vị quản lý bộ nhớ kép (một cho nhớ lệnh và một cho nhớ dữ liệu) bên trong chip. Điều này tạo khả năng cho đơn vị xử lý trung tâm (CPU) truy cập song song vào bộ nhớ lệnh và bộ nhớ dữ liệu. Nó cũng đặc biệt có ý nghĩa cho 68060 với kiến trúc siêu hướng.

Mỗi một đơn vị quản lý bộ nhớ MMU có một cache chuyển đổi địa chỉ (ATC) (Address Translation Cache). ATC được gọi như TLB trong nhiều hệ thống khác (tương tự như trong họ X86 và Pentium mà ta đã xét). ATC dùng để chuyển địa chỉ ảo thành địa chỉ vật lý. Nếu không có cặp dịch địa chỉ trong ATC, thì CPU tìm các bảng chuyển

trường TIC phải có 6 bit (bit 12 của số gia trang phải thuộc về trường TIC trong trường hợp này). Mỗi một con trỏ gốc chứa địa chỉ cơ sở của bảng thuộc tầng đầu (tầng A). Địa chỉ cơ sở cho mỗi bảng được chỉ số nhờ một trường thuộc địa chỉ logic. Một địa chỉ logic được chia thành các trường chỉ số bảng. Trường chỉ số bảng A (TIA) có 7 bit, được sử dụng để chỉ số tới bảng con trỏ của tầng đầu tiên (tầng A) và chọn 1 trong số 128 (27) mô tả con trỏ (pointer descriptors). Trong tầng đó mỗi một mô tả tương ứng với một khối 32 MB nhớ và trỏ tới cơ sở của bảng tầng tiếp theo (tầng B). Chỉ số bảng B (TIB) chọn 1 trong số 128 mô tả con trỏ trong bảng tầng thứ 2 được chọn. Mỗi một mô tả lại trỏ bảng trang trong tầng tiếp theo (tầng C) và tương ứng với khối 256 kbyte nhớ. Chỉ số bảng C (TIC) cho 1 trong số 32 (đối với trang 8 kbyte) hoặc trong 64 (đối với trang 4 kbyte) mô tả trong bảng tầng thứ 3. Các mô tả trong các bảng chứa hoặc mô tả để chuyển địa chỉ.



Hình 2.166: Cây bảng chuyển đổi địa chỉ

Nếu so sánh với cơ chế chuyển đổi địa chỉ của các chip vi xử lý Intel X86, có thể thấy rằng, cơ chế chuyển đổi của họ M68000 đòi hỏi ít dung lượng nhớ hơn vì những lý do sau đây:

1. Bảng dung lượng nhỏ sẽ tiết kiệm dung lượng nhớ và cho phép truy cập nhanh hơn.

2. Bảng dung lượng nhỏ cho phép có nhiều bảng hơn để phân phối cho quá trình. Xu hướng chung của các hệ điều hành là sử dụng các quá trình nhỏ và ngắn để giảm các quá trình tạo và xóa, qua đó nâng cao hiệu suất nói chung. Các bảng nhỏ cho phép thực hiện các quá trình nhẹ nhàng hơn.

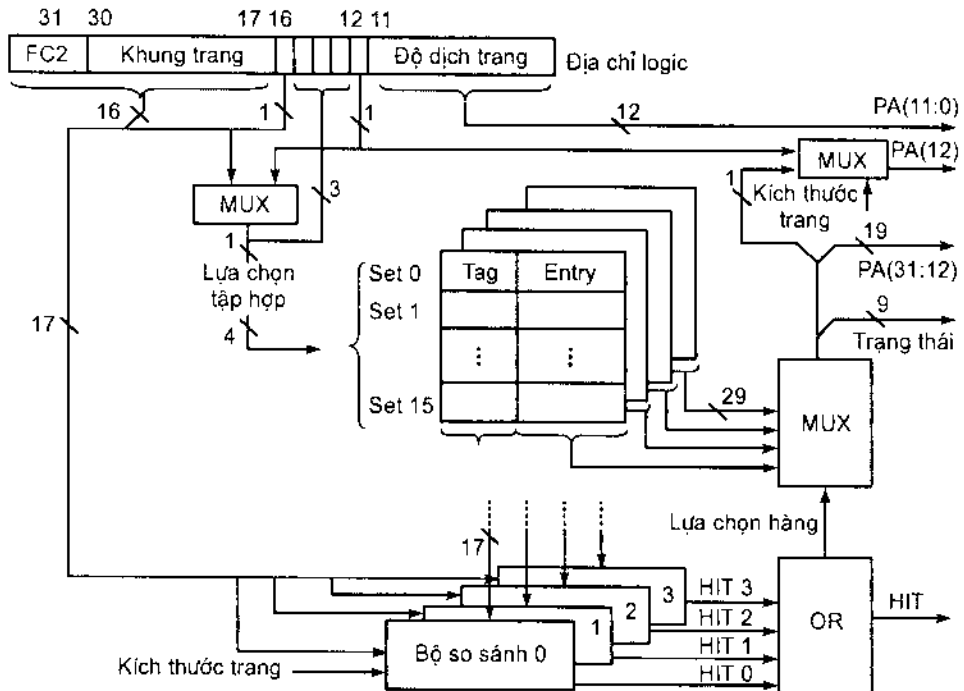
Đối với các trang dung lượng 4 kbyte, thì địa chỉ logic được phân thành 4 nhóm: 7, 7, 6, 12 bit. Tất cả các điểm vào của bảng có độ dài 4 byte. Như vậy, kích thước của các bảng ở cấp 1 (A) và cấp 2 (B) là: $2^7 \times 4 = 512$ byte, và kích thước của các bảng ở cấp 3 là: $2^6 \times 4 = 256$ byte. Trong khi đó, kích thước của các bảng của vi xử lý Intel x86 luôn cố định bằng 4 kbyte của bất kỳ trang nào.

Khi M68000 dùng cho một hệ điều hành Unix, thì tối thiểu phải phân bổ 4 phân đoạn (segments) cho một quá trình, như vậy cấu trúc bảng của M68000 cần phải có thêm: $512 + 512 + 4 \times 256 = 2$ kbyte, trong khi đó, sự phân chia các nhóm bit của địa chỉ ở vi xử lý Intel cần phải có thêm: $4 \text{ kbyte} + 4 \times 4 \text{ kbyte} = 20 \text{ kbyte}$.

Họ M68000 với phân chia địa chỉ logic thành các nhóm bit: 7, 7, 6, 12 đảm bảo tối đa: $2^{14} = 16$ kbyte phân đoạn cho một quá trình. Còn ở Intel với phân địa chỉ thành các nhóm bit: 10, 10, 12 đảm bảo tối đa: $2^{10} = 1024 = 1$ kbyte phân đoạn cho một quá trình.

Phương án chấp nhận được đối với M68000 là chọn sự phân nhóm địa chỉ logic thành 7, 7, 6, 12 bit cho các trang dung lượng 4 kbyte và phân nhóm: 7, 7, 5, 13 cho các trang dung lượng 8 kbyte. Và kiến trúc chuyển đổi địa chỉ theo 3 tầng không yêu cầu nhiều dung lượng thêm và cho phép có nhiều quá trình ngắn phía dưới ghép thêm cho quá trình trên. Nhưng xu hướng chung chọn dung lượng trang 4 kbyte.

Bốn thanh ghi thông dịch trong mô hình lập trình giám sát: DTT0 và DTT1 trong dữ liệu MMU, ITT0 và ITT1 trong lệnh MMU, xác định bốn khối của không gian địa chỉ logic mà nó được chuyển đổi trực tiếp thành địa chỉ vật lý. Các khối địa chỉ được xác định bởi các thanh ghi này bao gồm ít nhất 16 MB không gian địa chỉ logic. Bốn khối này có thể xếp chồng lên nhau hay đứng tách biệt. Nếu một trong các thanh ghi mTTx ($m = D$ hoặc I , và $x = 0$ hoặc 1) gặp được trong quá trình truy cập đến đơn vị nhớ (chứa dữ liệu hay lệnh), thì truy cập được chuyển đổi một cách thông suốt (thông dịch). Nếu cả 2 thanh ghi gặp được, thì các bit trạng thái của thanh ghi mTT0 được sử dụng để truy cập. Sự chuyển đổi trong suốt có thể cũng được thực hiện nhờ các bảng chuyển đổi địa chỉ của các cây chuyển đổi nếu các địa chỉ vật lý của trang được thiết lập bằng các địa chỉ logic của chúng.



Hình 2.167: Tổ chức của ATC trong họ MM68000

31	12	11	10	9	8	7	6	5	4	3	2	1	0
Địa chỉ vật lý	B	G	U1	U0	S	CM	M	0	W	T	R		

Trong đó:

Bit 31 - 12 (Địa chỉ vật lý): chứa 20 bit cao của địa chỉ vật lý được chuyển đổi.

Bit 11: (Bus error (B)): lỗi bus, được thiết lập nếu có lỗi truyền trong quá trình tìm kiếm bảng (table)

Bit 10 (Global (G)): được thiết lập để chỉ ra rằng entry là toàn cục. Các entry toàn cục không có nghĩa khi thực hiện lệnh PFLUSH.

Bits 9, 8: Thuộc tính trạng của người sử dụng (U1, U0) và được người sử dụng xác định.

Bit 7 (Supervisor protection (bảo vệ giám sát) (S)): được thiết lập, thì chỉ các chương trình chạy ở chế độ giám sát được phép truy cập tới một vùng không gian địa chỉ logic. Nếu S bị xóa thì cả người giám sát và sử dụng được phép truy cập tới không gian địa chỉ logic.

Bits 6,5 (Cache Mode (M)):

00 cacheable, write through (ghi thông suốt)

01 cacheable, write back (ghi trở lại)

10 cache inhibited (cấm cache), serialized: một trình tự của truy cập đọc và ghi với trạng trùng với trình tự của thứ tự lệnh.

Bit 4 (Modified (M)): được thiết lập khi xuất hiện một truy cập ghi tới địa chỉ logic tương ứng với entry.

Bit 2 (Write protected (W)): khi W được thiết lập, một truy cập ghi hoặc RMW tới địa chỉ logic tương ứng với điểm vào gây ra một loại trừ lỗi bus.

Bit 1 (Transparent translation register hit (T)): nếu T thiết lập, đó có nghĩa là địa chỉ lệnh PTEST trùng với TTR dữ liệu hoặc lệnh. Trong trường hợp này bit R được thiết lập, và tất cả các bit khác = 0.

Bit 0: Resident (R).

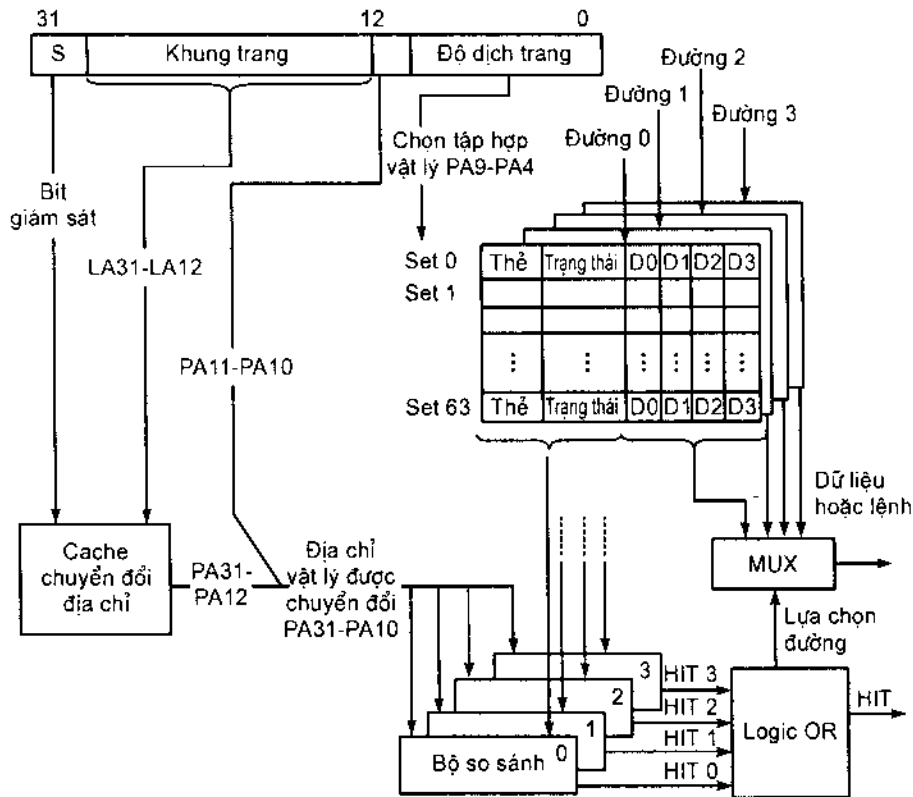
Hình 2.168: Thanh ghi trạng thái của MMU (MMUSR)

Mỗi một ATC có 64 điểm vào và liên hợp tập 4 đường (64-entry, 4-way associative). Hình 2.167 là tổ chức của ATC trong họ M68000. Bốn bit của địa chỉ logic nằm ở phần độ dịch trang (Page offset) (bit 16-13 cho các trang 8 kbyte, bit 15-12 cho các trang 4 kbyte), chỉ số tới 16 tập hợp (set 0 - set 15) của các điểm vào. Các thẻ (tags) được so sánh với những bit cao còn lại của địa chỉ logic và bit FC (khi FC2 = 1, thì vi xử lý ở trong chế độ giám sát, khi FC2 = 0, vi xử lý ở trong chế độ người sử dụng). Nếu một trong các thẻ gặp được (match), thì điểm vào tương ứng với thẻ đó được chọn nhờ đơn vị MUX để tạo ra một địa chỉ vật lý (PA) và trạng thái thông tin. Nếu không gặp bất kỳ thẻ nào, thì chúng ta có sự trượt qua (miss) và phải cần đến sự tìm kiếm các bảng. Sự chuyển đổi địa chỉ của ATC xảy ra đồng thời với sự chỉ tới các cache bên trong chip.

MMU có thanh ghi trạng thái, chứa thông tin trạng thái trả về sau khi thực hiện lệnh PTEST. Lệnh PTEST tìm kiếm các bảng dịch để xác định thông tin trạng thái về sự dịch một địa chỉ logic xác định nào đó. Nội dung của thanh ghi trạng thái của MMU (MMUSR) như hình 2.168.

2.6.8. Bộ nhớ Cache

Các cache lệnh (Icache) và dữ liệu (Dcache) là một phần của MMU tương ứng. Các cache được tổ chức theo một khuôn mẫu xác định. Chúng là tập của 4 khối 16 byte/đường liên kết. Kích thước của mỗi Cache ở 68060 là 8 kbyte (tổng số là 16 kbyte), và ở 68040 là 4 kbyte (tổng số là 8 kbyte). Hình 2.169 trình bày các cache bên trong của M68040.



Hình 2.169: Cache bên trong của M68000 (M68040, M68060)

Mỗi một đường cache (cache line) có một thẻ địa chỉ - address tag (TAG), thông tin trạng thái, và 4 từ dài của dữ liệu (D0 - D3). Mỗi thẻ địa chỉ TAG chứa 22 bit cao của địa chỉ vật lý.

Thông tin trạng thái đối với cache lệnh (Icache) có một bit giá trị đơn ứng với toàn bộ đường. Thông tin trạng thái đối với Dcache chứa một bit giá trị và 4 bit bổ sung để chỉ trạng thái bị biến đổi cho từng từ dài trong đường (longword in the line). Bởi vì sự định giá trị của điểm vào được đảm bảo ở cơ sở đường, một đường phải được nạp từ bộ nhớ hệ thống để cache cất giữ một điểm vào.

Trong khi các bit cao của địa chỉ logic đưa tới ATC để chuyển đổi thành địa chỉ vật lý, thì các bit thấp được sử dụng để truy cập trực tiếp tới cache. Bốn bit thấp (3 - 0) chọn một byte trong 16 byte của đường cache (line cache). Các bit từ 9 đến 4 (6 bit) chọn một trong 64 tập hợp (trong M68040). Trong cache 8 kbyte có 128 tập hợp cùng giá trị và thông số, và để chọn 128 tập hợp cần trường 7 bit (bit 4 - 10).

Các cache được địa chỉ vật lý cho phép khối logic bên ngoài truy cập đến chúng mà không cần phải chuyển đổi ngược lại từ địa chỉ vật lý thành địa chỉ logic. Đây là một đặc tính quan trọng cho các giao thức truy cập cache. Khi cache đã có địa chỉ vật lý thì có thể giải quyết được vấn đề gán địa chỉ. Các cache có thể sử dụng riêng lệnh MOVEC để truy cập đến thanh ghi 32-bit điều khiển cache CACR (hình 2.170). Thanh ghi CACR chứa 2 bit cho phép một cách độc lập các cache lệnh và dữ liệu (Icache, Dcache). Sự thiết lập 1 bit trong 2 bit này cho phép cache tương ứng ảnh hưởng đến bất kỳ đường nào bên trong cache. Có phần cứng dùng để xóa thanh ghi CACR, và cấm cả 2 loại cache. Nhưng, các thẻ, thông tin trạng thái, và dữ liệu trong các cache không bị ảnh hưởng bởi logic xóa này và phải được xóa bằng lệnh CINV trước khi cho phép các cache.

31 30																16 15 14																0															
DE 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																IE 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																															

DE: cho phép cache dữ liệu (enable data cache)
 IE: cho phép cache lệnh (enable instruction cache)

Bảng 2.25: Gán các véc-tơ ngoại lệ

Số hiệu véc-tơ ngắt	Địa chỉ offset của véc-tơ (Hex)	Gán
0	000	Xóa con trỏ ngăn xếp ngắt ban đầu
1	004	Xóa bộ đếm chương trình ban đầu
2	008	Lỗi truy cập
3	00C	Lỗi địa chỉ
4	010	Lệnh sai
5	014	Chia số nguyên cho 0
6	018	Các lệnh CHK, CHK2
7	01C	Các lệnh FTRAPcc, TRAPcc, TRAPV
8	020	Lỗi thẩm quyền
9	024	Thực hiện tuần tự (Trace)
10	028	Mã lệnh A-line không thể thực hiện được (Line 1010 emulator)
11	02C	Mã lệnh F-line không thể thực hiện được (Line 1111 emulator)
12	030	Dự phòng
13	034	Dùng cho MC68020 và MC68030, không dùng cho MC68040
14	038	Lỗi khuôn dạng
15	03C	Ngắt không che được (NMI)
16-23	040-05C	Dự phòng
24	060	Ngắt sai
25	064	Véc-tơ tự động của ngắt mức 1 (level 1 interrupt autovector)
26	068	Véc-tơ tự động của ngắt mức 2 (level 2 interrupt autovector)
27	06C	Véc-tơ tự động của ngắt mức 3 (level 3 interrupt autovector)
28	070	Véc-tơ tự động của ngắt mức 4 (level 4 interrupt autovector)
29	074	Véc-tơ tự động của ngắt mức 5 (level 5 interrupt autovector)
30	078	Véc-tơ tự động của ngắt mức 6 (level 6 interrupt autovector)
31	07C	Véc-tơ tự động của ngắt mức 7 (level 7 interrupt autovector)
32-47	080-0BC	Các vector lệnh bẫy TRAP #0-15
48	0C0	Rẽ nhánh dấu phẩy động hoặc thiết lập điều kiện không theo thứ tự

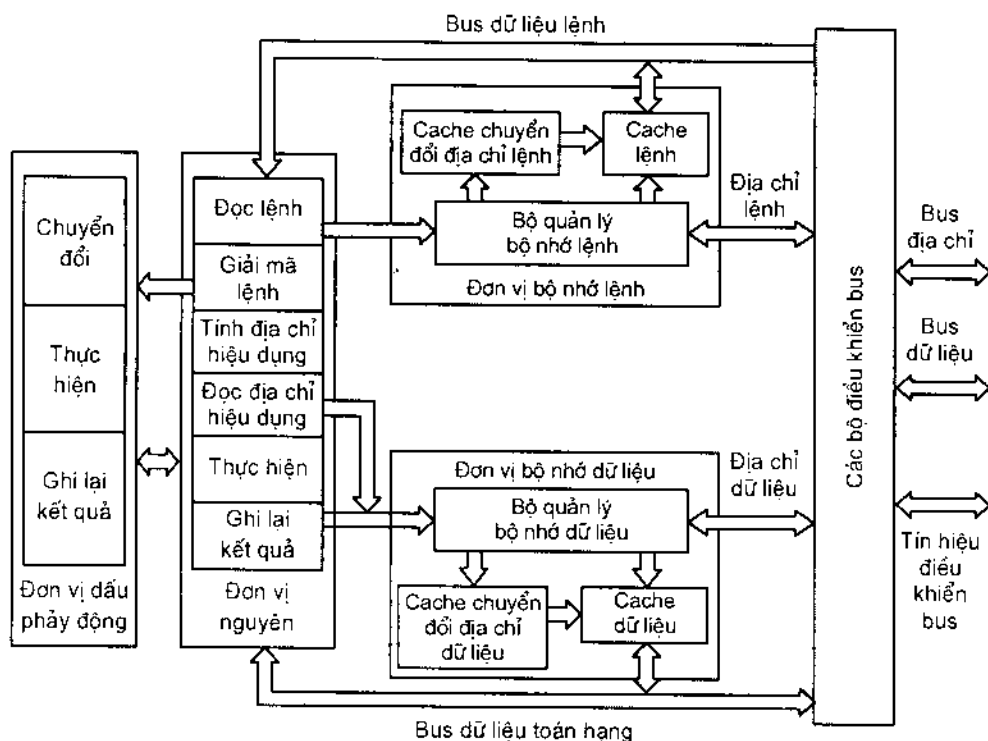
Số hiệu véc-tơ ngắt	Địa chỉ offset của véc-tơ (Hex)	Gán
49	0C4	Kết quả số dấu phẩy động không chính xác
50	0C8	Chia số dấu phẩy động cho 0
51	0CC	Tràn dưới số dấu phẩy động
52	0D0	Lỗi toán hạng số dấu phẩy động
53	0D4	Lỗi tràn trên số dấu phẩy động
54	0D8	Báo hiệu không phải số dấu phẩy động (FP signaling NAN)
55	0DC	Loại dữ liệu dấu phẩy động không thực hiện được
56	0E0	Xác định cho MC68030 và MC68851, không dùng cho MC68040
57	0E4	Xác định cho MC68851, không dùng cho MC68040
58	0E8	Xác định cho MC68851, không dùng cho MC68040
59-63	0EC-0FC	Dự phòng
64-255	100-3FC	Dự phòng

3. Cất giữ từ trạng thái hiện hành của đơn vị xử lý trung tâm nhờ việc tạo ra một khung ngăn xếp ngoại lệ (exception stack frame) trên ngăn xếp đang tích cực của chế độ giám sát. Nếu đây là một ngắt và bit M của SR đã được thiết lập, thì Bit M phải được xóa và một khung ngăn xếp thứ 2 trên ngăn xếp ngắt phải được tạo ra.

4. Khởi tạo sự thực hiện chương trình con xử lý ngoại lệ. Nhân số hiệu véc-tơ ngắt với 4 để xác định số gia của véc-tơ ngoại lệ. Cộng số gia với giá trị cất giữ trong VBR để có được địa chỉ của vùng nhớ của véc-tơ loại trừ và nạp vào PC (và ISP trong trường hợp là ngoại lệ là RESET).

2.6.10. Kiến trúc của M68040

Vi xử lý M68040 là chip vi xử lý công nghệ HCMOS 0,8 micron, 1,2 triệu transistor, 179 chân PGA. Các đặc tính chính của M68040 đã liệt kê trong bảng 2.18. Sơ đồ khối và phân nhóm các tín hiệu của M68040 mô tả trong hình 2.171 và 2.172.

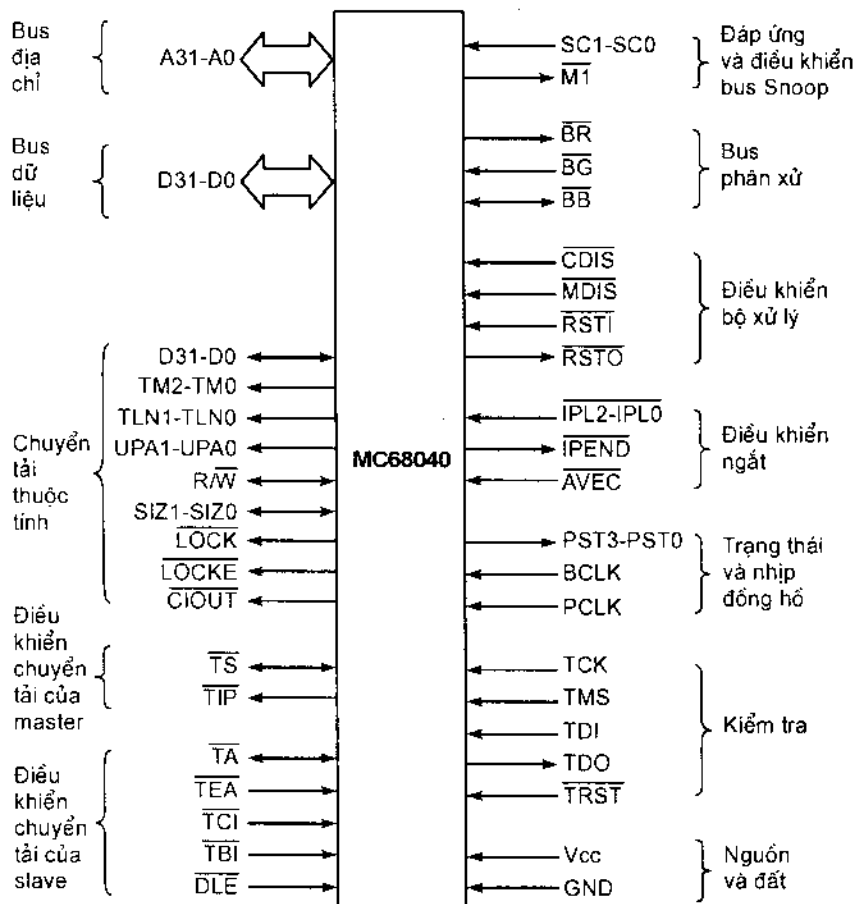


Hình 2.171: Sơ đồ khối của M68040

M68040 được thiết kế với mục đích thỏa mãn nhu cầu của đa số các nhà chế tạo máy tính là phải có hiệu suất bus cao (có thời gian trễ thấp) để dùng các loại bộ nhớ rẻ tiền như DRAM (Dynamic Random Access Memory). Như vậy sẽ hạn chế tối thiểu việc sử dụng các bộ nhớ tĩnh SRAM (Static Random Access Memory) có tốc độ cao và đắt tiền nhằm đạt hiệu suất cao cho hệ thống máy tính và không cần đến bộ nhớ cache để tăng tốc độ truy cập bộ nhớ. M68040 có đường ống nguyên (IU pipeline) 6 giai đoạn và đường ống dấu phẩy động (FPU pipeline) 3 giai đoạn kết nối với đường ống nguyên (hình 2.173).

Đường ống nguyên (IU pipeline) 6 giai đoạn gồm có: đọc lệnh (instruction fetch), giải mã lệnh (decode), tạo địa chỉ và đọc toán hạng thứ nhất (EA calculate), tạo địa chỉ và đọc toán hạng thứ hai (EA calculate), thực hiện lệnh (execute), cất giữ kết quả (writeback).

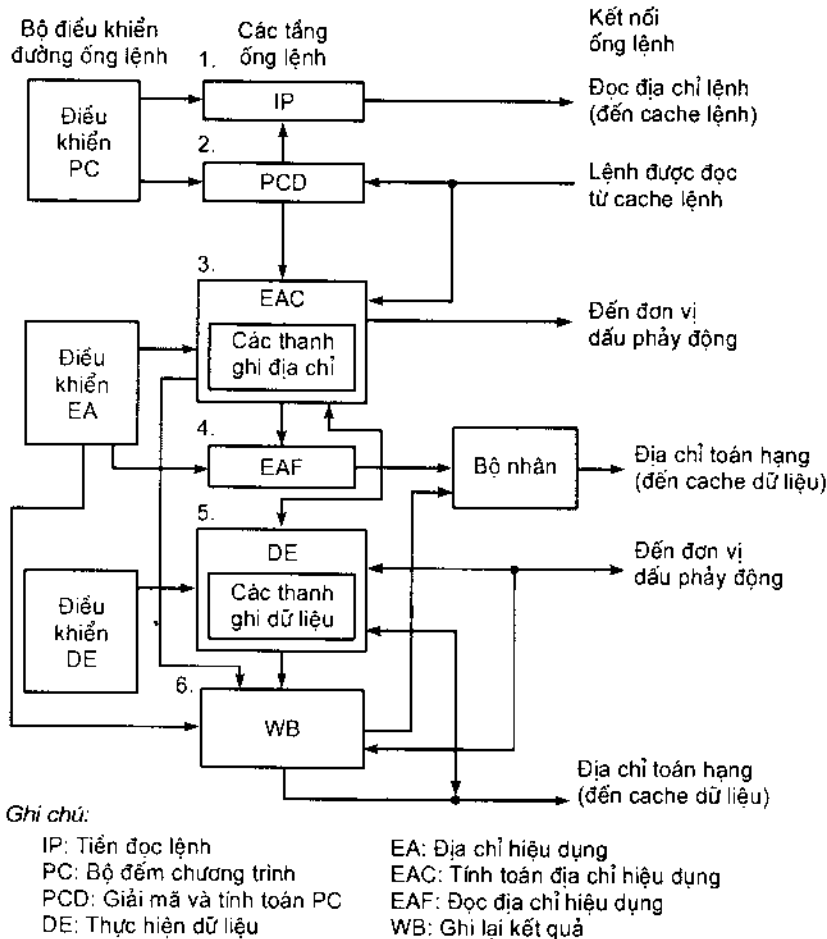
Đường ống dẫn phẩy động (FPU pipeline) 3 giai đoạn thực hiện xử lý các phép tính dấu phẩy động: chuyển đổi (convert), thực hiện (execute), cất giữ kết quả (writeback).



Hình 2.172: Phân nhóm các tín hiệu của M68040

Kiến trúc của Cache bên trong chip 68040 có các thông số: 4 cửa liên hợp tập với 16 byte/đường.

Vi xử lý M68040 được ứng dụng có hiệu quả trong các hệ thống máy tính như NEXT systems, Apple's Macintosh Quadra 950, Macintosh Centris 650 và nhiều hệ thống khác.

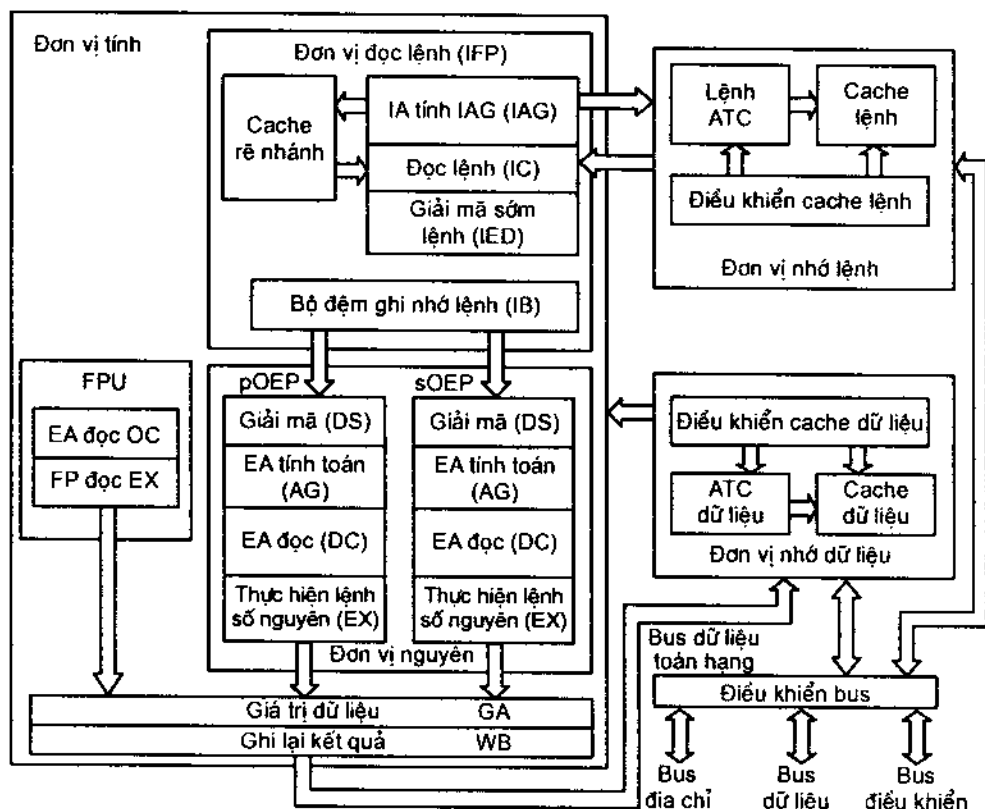


Hình 2.173: Đường ống nguyên của M68040

2.6.11. Kiến trúc của M68060

M68060 là chip vi xử lý công nghệ CMOS, 0,5 micron, 2,5 triệu transistor, 3 lớp kim loại (metal), đóng vỏ 223 chân PGA. Các đặc tính chính của M68060 đã liệt kê trong bảng 2.18. Phiên bản ban đầu của M68060 làm việc ở 50 MHz với tốc độ xử lý 75 MIPS, sau đó 66 MHz, đến 75 MHz và 113 MIPS cho các loạt 68LC060 và 68EC060. Vi xử lý M68060 là hệ thống có cấu trúc siêu hướng, có các FPU và MMU bên trong hiệu suất cao toàn bộ tính năng đầy đủ của họ M68000.

M68060 có kiến trúc siêu hướng 2 cửa ra, và hoàn toàn tương thích với các loại vi xử lý đời trước của họ M68000. Sơ đồ khối của M68060 được mô tả trong hình 2.174.



Hình 2.174: Sơ đồ khối của 68060

Vi xử lý M68060 gồm có các khối chức năng sau đây:

1. Đơn vị đọc lệnh (instruction fetch unit)
2. Các đơn vị cấu trúc ống hai toán hạng (dual operand pipeline unit)
3. Các đơn vị tính (execution units)
 - a) Đơn vị tính hai số nguyên (dual interger execution unit)
 - b) Đơn vị tính các số dấu phẩy động (floating-point execution unit)
4. Các đơn vị nhớ
 - a) Đơn vị nhớ lệnh (instruction memory unit) gồm:

- Cache chuyển đổi địa chỉ lệnh ATC (instruction address translation Cache)
- Cache chứa lệnh (Icache) 8 kbyte
- Điều khiển nhớ lệnh (instruction memory controller)

b) Đơn vị nhớ dữ liệu (data memory unit) gồm:

- Data ATC
- Cache chứa dữ liệu (Dcache) 8 kbyte
- Điều khiển nhớ dữ liệu (data memory controller)

5. Đơn vị giao tiếp bus BIU (Bus Interface Unit)

Các tín hiệu của M68060 được phân thành các nhóm như trong hình 2.175. Như vậy cache bên trong của M68060 có dung lượng tăng gấp đôi so với cache bên trong của M68040.

Vi xử lý 68060 có các ống như sau (hình 2.176):

1. Ống đọc lệnh 4 giai đoạn IFP:

- 1.1. Tạo địa chỉ IAG (Instruction Address Generation)
- 1.2. Truy cập cache lệnh IC (Instruction Cache access)
- 1.3. Giải mã sớm lệnh IED (Instruction Early Decode)
- 1.4. Bộ đệm ghi nhớ lệnh IB (Instruction Buffer)

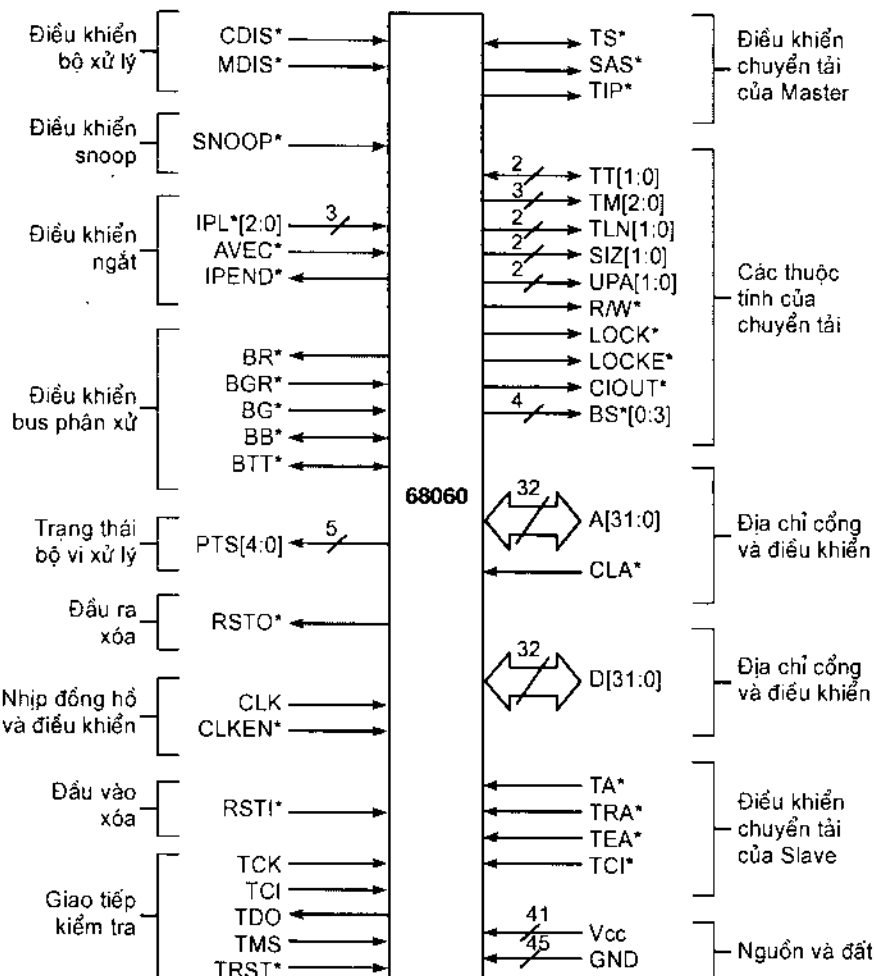
2. Ống tính toán hạng thứ nhất 4 giai đoạn pOEP (primary Operand Execution Pipeline) và ống tính toán hạng thứ hai 4 giai đoạn sOEP (second OEP). Cả 4 giai đoạn của hai ống này giống nhau và gồm:

- 2.1. Giải mã và chọn lệnh DS (Decode and Select instruction)
- 2.2. Tạo địa chỉ của toán hạng AG (Address Generation of the operand - EA calculate)
- 2.3. Truy cập cache toán hạng OC (Operand Cache access - EA fetch)
- 2.4. Tính EX (interger EXecute)

Hai ống pOEP và sOEP được tiếp nối bằng 2 giai đoạn chung là:

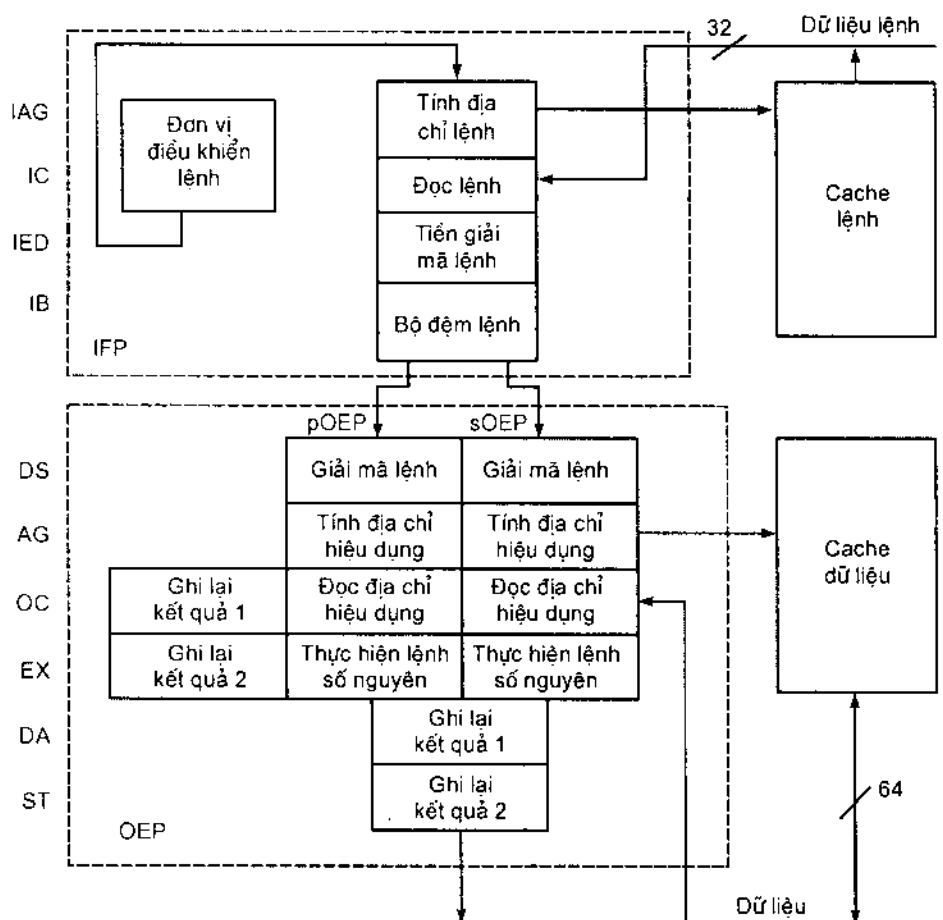
a) Có dữ liệu DA (Data Available)

b) Ghi lại kết quả WB (WriteBack)



Hình 2.175: Phân nhóm các tín hiệu của M68060

Như vậy, ống OEP có tất cả 6 giai đoạn để xử lý các số nguyên. Hai giai đoạn cuối (DA, WB) được sử dụng khi bộ xử lý ghi dữ liệu (kết quả tính) và bộ nhớ.



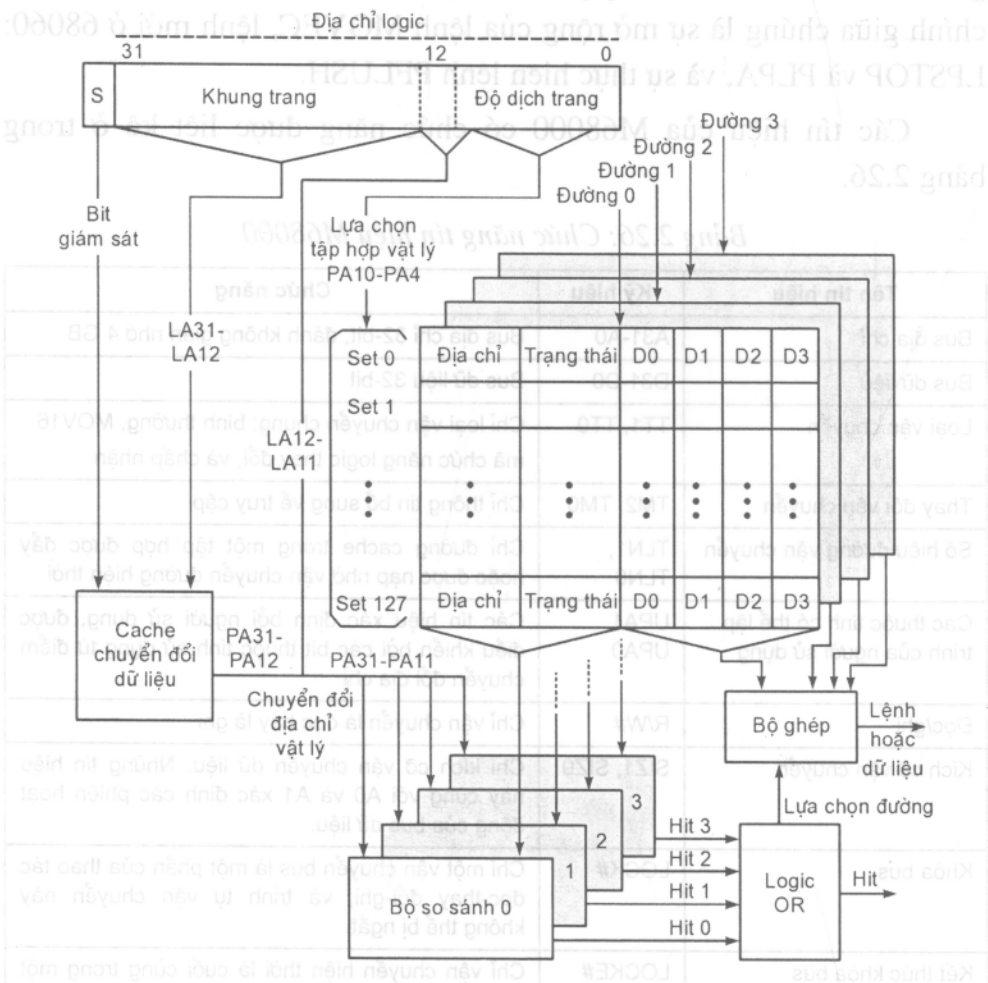
Hình 2.176: Đường ống của M68060

Kiến trúc của vi xử lý 68060 có những đặc điểm sau đây:

1. Vi xử lý 68060 chỉ có một con trỏ ngăn xếp ở chế độ giám sát giống như trong 68000.
2. Trong thanh ghi trạng thái chỉ có 1 bit T (trace bit) giống như trong 68000
3. Không có MMUSR
4. Có hai thanh ghi hệ thống mới:
 - a) Thanh ghi cấu hình bộ xử lý PCR 8-bit (processor configuration register), trong đó chỉ có 3 bit được dùng: bit 7 - EDEBUG, cho phép

gỡ rồi (enable debug) khi nó được lập; bit 1 - DFP, loại bỏ FPU khi được thiết lập; bit 0 - ESS, cho phép phân nhánh siêu hướng (enable superscalar dispatch khi) được thiết lập.

b) Thanh ghi điều khiển bus BUSCR (BUS Control Register) 32-bit, trong đó chỉ có 2 bit được dùng: bit 31- L, bit khóa (lock bit), khi L = 1, có tín hiệu khóa LOCK từ bên ngoài tác động vào, ngược lại, L = 0; bit 30- khóa sao chép bóng SL (Shadow copy Lock bit), khi SL = 1, có một chuỗi khóa ở thời gian loại trừ (time of exception), ngược lại SL = 0.



Hình 2.177: Cache bên trong của M68060

5. Kiến trúc của Cache bên trong M68060 mô tả ở hình 2.177. Các thông số của cache trong M68060 cũng tương tự như của cache trong M68040, đó là: 4 cửa liên hợp tập với 16 byte/đường, ngoại trừ kích thước của cache trong M68060 lớn gấp đôi (bằng 8 kbyte so với 4 kbyte trong M68040). Số lượng các tập hợp lên đến 128.

6. ATC của M68060 về cơ bản giống ATC của M68040.

7. Tập lệnh của M68060 được phân loại cho siêu hướng của đường ống pOEP và sOEP (xem phần Phụ lục 4). Có sự khác nhau giữa 68060 và 68040 trong phần cứng hỗ trợ các lệnh. Sự khác nhau chính giữa chúng là sự mở rộng của lệnh MOVEC, lệnh mới ở 68060: LPSTOP và PLPA, và sự thực hiện lệnh PFLUSH.

Các tín hiệu của M68000 có chức năng được liệt kê ở trong bảng 2.26.

Bảng 2.26: Chức năng tín hiệu M68000

Tên tín hiệu	Ký hiệu	Chức năng
Bus địa chỉ	A31-A0	Bus địa chỉ 32-bit, đánh không gian nhớ 4 GB
Bus dữ liệu	D31-D0	Bus dữ liệu 32-bit
Loại vận chuyển	TT1, TT0	Chỉ loại vận chuyển chung: bình thường, MOV16 mã chức năng logic thay đổi, và chấp nhận
Thay đổi vận chuyển	TM2, TM0	Chỉ thông tin bổ sung về truy cập
Số hiệu đường vận chuyển	TLN1, TLN0	Chỉ đường cache trong một tập hợp được đẩy hoặc được nạp nhờ vận chuyển đường hiện thời
Các thuộc tính có thể lập trình của người sử dụng	UPA1, UPA0	Các tín hiệu xác định bởi người sử dụng, được điều khiển bởi các bit thuộc tính sử dụng từ điểm chuyển đổi địa chỉ
Đọc/ghi	R/W#	Chỉ vận chuyển là đọc hay là ghi
Kích cỡ vận chuyển	SIZ1, SIZ0	Chỉ kích cỡ vận chuyển dữ liệu. Những tín hiệu này cùng với A0 và A1 xác định các phiên hoạt động của bus dữ liệu.
Khóa bus	LOCK#	Chỉ một vận chuyển bus là một phần của thao tác đọc-thay đổi-ghi, và trình tự vận chuyển này không thể bị ngắt.
Kết thúc khóa bus	LOCKE#	Chỉ vận chuyển hiện thời là cuối cùng trong một trình tự các vận chuyển bị khóa.

Tên tín hiệu	Ký hiệu	Chức năng
Ngoài cache	CIOUT#	Chỉ vi xử lý sẽ không cache vận chuyển bus hiện thời
Bắt đầu vận chuyển	TS#	Chỉ sự bắt đầu của một vận chuyển bus
Vận chuyển trong tiến trình	TIP#	Tích cực trong khi có vận chuyển bus
Chấp nhận vận chuyển	TA#	Tích cực để chấp nhận một vận chuyển
Chấp nhận lỗi vận chuyển	TEA#	Chỉ ra có lỗi vận chuyển
Cấm vận chuyển cache	TCI#	Chỉ vận chuyển bus không được lưu cache
Cấm bùng phát vận chuyển	TBI#	Chỉ thiết bị tớ (slave) không thể xử lý truy cập bùng phát
Cho phép chốt dữ liệu	DLE	Nhịp đồng hồ được sử dụng để chốt dữ liệu vào khi bộ xử lý đang khai thác trong chế độ DLE
Điều khiển dò tìm	SC1, SC0	Chỉ thao tác dò tìm được yêu cầu khi thiết bị chủ bus truy cập
Cấm bộ nhớ	MI#	Cấm các thiết bị nhớ trả lời cho các truy cập của thiết bị chủ trong các thao tác truy cập dò tìm
Yêu cầu bus	BR#	Được bộ xử lý khẳng định để yêu cầu làm chủ bus
Chấp nhận bus	GB#	Được trọng tài (arbiter) khẳng định để cho phép bộ xử lý làm chủ bus
Bus bận	BB#	Được một chủ bus hiện thời khẳng định để chỉ ra rằng nó đang làm chủ bus
Không cho phép cache	CDIS#	Không cho phép các cache bên trong hỗ trợ mô phỏng
Không cho phép MMU	NDIS#	Không cho phép cơ chế chuyển đổi của MMU
Xóa vào (Reset in)	RSTI#	Xóa bộ xử lý
Xóa ra	RSTO#	Khẳng định sự thực hiện lệnh xóa RESET để xóa các thiết bị ngoài
Mức ưu tiên ngắt	IPL2#-IPL0#	Đảm bảo một mức ngắt được mã hóa cho bộ xử lý
Chờ đợi ngắt	IPEND#	Chỉ một ngắt đang chờ đợi
Véc-tơ tự động (autovector)	AVEC#	Được sử dụng khi chuyển một chấp nhận ngắt để yêu cầu tạo ra bên trong bộ xử lý một số hiệu véc-tơ
Trạng thái bộ xử lý	PST3-PST0	Chỉ trạng thái bên trong của bộ xử lý
Nhịp đồng hồ của bus	BCLH	Nhịp đồng hồ sử dụng cho định thời các tín hiệu bus

Tên tín hiệu	Ký hiệu	Chức năng
Nhịp đồng hồ bộ xử lý	PCLK	Nhịp đồng hồ sử dụng cho định thời cho các logic bên trong bộ xử lý Tần số của PCLK đúng bằng gấp đôi tần số của BCLK
Nhịp đồng hồ kiểm tra	TCK	Tín hiệu đồng hồ cho IEEE P1149 cổng truy cập kiểm tra (TAP)
Chọn chế độ kiểm tra	TMS	Chọn các theo tác nguyên tắc của mạch hỗ trợ kiểm tra
Vào dữ liệu kiểm tra	TDI	Vào nối tiếp cho cổng TAP
Ra dữ liệu kiểm tra	TDO	Ra nối tiếp cho cổng TAP
Xóa kiểm tra	TRST#	Xóa không đồng bộ điều khiển cổng TAP
Nguồn cung cấp	Vcc	Nguồn nuôi bộ xử lý
Ground	GND	Nối đất.

BÀI TẬP ÔN LUYỆN

1. Sơ đồ khối hay mô hình lập trình mô tả chi tiết hơn về cấu trúc vi xử lý?
2. Chức năng nào sau đây không phải là chức năng của ALU?
a: Cộng; b: Dịch trái; c: Chuỗi bật nguồn; d: Bù.
3. ALU có 2 đầu vào, đối với vi xử lý 8-bit Z80 trình bày trong chương này thì 2 đầu vào này được nối với:
a: PC;
b: Bus dữ liệu trong;
c: Logic điều khiển;
d: Thanh ghi địa chỉ bộ nhớ.
4. Công việc chính của ALU là:
a: Thực hiện phép cộng;
b: Như là đầu vào của thanh ghi tích lũy (Accumulator);
c: Thay đổi logic hoặc số học các từ dữ liệu;
d. Tất cả a, b, c.
5. Hầu hết các phép toán số học và logic trong vi xử lý thực hiện thao tác giữa các nội dung của vùng nhớ hoặc nội dung của một thanh ghi với:
a: Thanh ghi tích lũy (Accumulator);
b: PC;
c: Thanh ghi địa chỉ bộ nhớ;
d: Thanh ghi lệnh.
6. Thanh tổng kết nối với các đơn vị phân cứng khác trong vi xử lý nhờ bus dữ liệu trong và nó có thể:
a: Chỉ nhận dữ liệu;
b: Chỉ đưa ra dữ liệu;
c: Cả đưa vào đưa ra dữ liệu;
d. Chỉ trao đổi với ALU.

7. Một dữ liệu được lập trình chuyển giữa cổng vào 40h và ngăn nhớ địa chỉ 0E9Ch sử dụng để cất giữ tạm thời dữ liệu:
- a: PC;
 - b: ALU;
 - c: Ngăn nhớ địa chỉ 0E9Ch;
 - d: Thanh ghi tổng.
8. Vi xử lý 16-bit có không gian đánh địa chỉ là 2^{20} . Vậy ta cần phải đảm bảo PC trong vi xử lý này có độ dài là bit.
- a: 4; b: 8; c: 16; d: 20; e: 22; f: 32.
9. Vi xử lý 16-bit có không gian đánh địa chỉ là 2^{20} . Vậy ta cần phải đảm bảo thanh ghi địa chỉ bộ nhớ trong vi xử lý này có độ dài là bit.
- a: 8; b: 16; c: 20; d: 220; e: 24; f: 32.
10. Các thanh ghi B, C, D, và E có thể được sử dụng như:
- a: PC;
 - b: Thanh ghi địa chỉ bộ nhớ;
 - c: Thanh ghi chung;
 - d: Cặp thanh ghi DC.
11. Cặp thanh ghi HL được sử dụng như là con trỏ vùng nhớ bởi vì:
- a: Nó gắn với PC;
 - b: Nó gắn với thanh ghi địa chỉ bộ nhớ;
 - c: Nó có thể được sử dụng như 2 thanh ghi 8-bit độc lập;
 - d: 16 bit để địa chỉ tất cả các vùng nhớ.
12. Nếu ta cần sử dụng một thanh ghi như là một bộ đếm 16-bit giảm dần, thì cần phải dùng:
- a: Thanh ghi D;
 - b: Thanh ghi C;
 - c: Thanh ghi B;
 - d: Đôi thanh ghi BC.

13. Thanh ghi địa chỉ bộ nhớ trữ tới:
 - a: Các nội dung của các ngăn nhớ;
 - b: Vùng nhớ;
 - c: Một thanh ghi nhớ;
 - d: Vùng của ALU.
14. Trong khi thực hiện một lệnh, thanh ghi lệnh (instruction register) lưu giữ lệnh:
 - a. Trước; b. Hiện thời; c. Sau đó; d. Tất cả a, b, c.
15. Độ dài của thanh ghi lệnh phụ thuộc vào:
 - a: Kiến trúc vi xử lý;
 - b: Thiết kế vi xử lý 8-bit hay 16-bit;
 - c: Kích thước của bộ nhớ được đánh địa chỉ;
 - d: Tốc độ vi xử lý.
16. Đối với vi xử lý 16-bit ta cần phải đảm bảo thanh ghi tạm thời bên trong có độ dài là bit
 - a: 8; b: 16; c: 20; d: 220; e: 24; f: 32.
17. Mục đích chính của thanh ghi tạm thời là:
 - a: Kết nối ALU với bus dữ liệu trong vi xử lý.
 - b: Kết nối ALU với thanh ghi tổng;
 - c: Cách biệt các đầu vào và ra của ALU;
 - d: Đảm bảo lưu dữ liệu của thanh tổng.
18. Đơn vị logic điều khiển (Control logic) không có trong mô hình lập trình của vi xử lý vì:
 - a: Một số vi xử lý không sử dụng logic điều khiển;
 - b: Sẽ chiếm nhiều chỗ trong sơ đồ mô hình lập trình;
 - c: Người lập trình không thể thay đổi dữ liệu trong logic điều khiển;
 - d: Không phải a, b, c, mà logic điều khiển có trong mô hình lập trình.

19. Mục đích của nhịp đồng hồ (CLOCK) của vi xử lý là:
 - a: Cung cấp thông tin về thời gian (Time-day);
 - b: Vận hành cơ chế cất giữ dữ liệu của ALU;
 - c: Cung cấp tín hiệu định thời cho trình tự của logic điều khiển;
 - d: Để chắc chắn lưu giữ dữ liệu trong các thanh ghi.
20. Một hệ thống vi xử lý Z80 phải bao gồm các vi mạch lập trình nào?
21. Cấu trúc xử lý ngắt theo chuỗi trong Z80? Phải thiết kế mức ưu tiên phục vụ ngắt cho các thiết bị ngoại vi có tốc độ trao đổi dữ liệu khác nhau như thế nào?
22. Thiết kế cấu trúc kết nối điều khiển vào/ra song song của Z80 với 2 vi mạch Z80PIO: có giải mã địa chỉ vào/ra chọn các PIO và các thanh ghi của PIO, có xử lý chuỗi ngắt. Thiết bị ngoại vi nối với 2 PIO là các bộ chỉ thị LED (mã hóa 2-10). Viết một đoạn chương trình điều khiển đưa ra chỉ thị LED.
23. Thiết kế cấu trúc kết nối điều khiển vào/ra nối tiếp của Z80 với 2 Z80SIO: có giải mã địa chỉ vào/ra chọn các chip SIO và các thanh ghi bên trong của chúng, có xử lý ngắt theo chuỗi. Thiết bị ngoại vi nối với 2 PIO là các bộ chỉ thị LED (mã hóa 2-10). Viết một đoạn chương trình điều khiển đưa ra chỉ thị LED.
24. Tương tự như bài 23 nhưng thiết bị ngoại vi là 1 cụm thiết bị đo số dùng ADC (chỉ thị đo đưa về CPU qua 1 cổng PIO sau đó từ CPU đưa ra kết quả hiển thị LED ở một cổng PIO thứ 2.
25. Tương tự như 24 nhưng cho SIO.
26. Nêu những đặc điểm chính của công nghệ vi xử lý Intel 808X?
27. Nêu những đặc điểm chính của công nghệ vi xử lý Intel 80X86?
28. Nêu những đặc điểm chính của công nghệ vi xử lý Intel Pentium?
29. Sự khác nhau cơ bản của các loại Pentium MMX, Celeron, và Xeon?
30. Mô hình lập trình tổng quát của công nghệ vi xử lý Intel?

31. Độ dài của con trỏ lệnh IP của 8088 là:
a: 4 bit; b: 8 bit; c: 16 bit; d: 32 bit.
32. Hàng tiên đọc lệnh hoặc cache lệnh xuất hiện đầu tiên ở vi xử lý:
a: 8088; b: 8086; c: 286; d: 386SX; e: 386DX;
f: 486SX; g: 486DX; h: tất cả a, b, c, d, e, f, g.
33. So sánh 8085 với 8080, 8085 với Z80 về cấu trúc phần cứng, bus dữ liệu, bus địa chỉ, các tín hiệu điều khiển, tần số làm việc, và tập lệnh (nên lập bảng so sánh)?
34. Tập lệnh của vi xử lý Intel?
35. Trong những điều kiện nào thì WAIT cần có?
36. Khi WAIT được yêu cầu, thì CPU phải thực hiện gì (trạng thái nào)?
37. Sự khác nhau cơ bản giữa WAIT và HOLD?
38. Phân biệt sự khác nhau cơ bản giữa 8086 và 8088?
39. Có những bổ sung gì trong mô hình lập trình của 8086/8088 so với 8085?
40. Chương trình ở 8086 có chạy được trên 8088 không?
41. Chế độ Min và Max của 8086/8088 được dùng trong những trường hợp nào. Những tín hiệu nào thay đổi khi thay đổi chế độ Min và Max?
42. Vì sao cần có trạng thái trở kháng cao, khi nào, và ở đâu trong cấu trúc vi xử lý?
43. Vì sao phải phân đoạn bộ nhớ trong hệ thống nhớ của 8086/8088?
44. Cơ chế tiên đọc lệnh trong 8086/8088 tăng tốc độ xử lý lệnh lên bao nhiêu, vì sao?
45. Chương trình viết cho 8080 có chạy được trên 8086/8088 hay không, vì sao?
46. Dung lượng ngăn xếp trong hệ thống 8086/8088 tối đa là bao nhiêu? Những tín hiệu nào hỗ trợ cho kết nối đồng xử lý 8087?

47. Phân nhóm các lệnh của 8085?
48. Trình bày khuôn dạng lệnh, dữ liệu, các kiểu đánh địa chỉ của 8085?
49. Hãy vẽ đồ thị xung chu kỳ ghi đọc bộ nhớ?
50. Hãy vẽ đồ thị xung chu kỳ vào/ra (I/O) với thiết bị ngoại vi?
51. Vì sao chu kỳ máy đọc lệnh cần ít nhất 4 trạng thái?
52. Trong những điều kiện nào thì DMA được yêu cầu?
53. Khi DMA được yêu cầu, thì CPU phải thực hiện gì (trạng thái nào)?
54. Vì sao các chân dữ liệu (bus dữ liệu) của 8085 phải trộn. Sự trộn bus dữ liệu này có ảnh hưởng đến tốc độ xử lý hay không?
55. Tín hiệu nào của 8085 thực hiện chốt các đường dây địa chỉ thấp?
56. Sự khác nhau giữa 2 cách vận chuyển dữ liệu: Load và Store?
57. Khi sử dụng kiểu đánh địa chỉ gián tiếp thanh ghi thì địa chỉ vùng nhớ lấy ở đâu?
58. Thế nào là địa chỉ nhị phân của thanh ghi D, của cặp thanh ghi DE?
59. Vẽ đồ thị xung thực hiện lệnh MVI A,03h?
60. Vẽ đồ thị xung thực hiện lệnh MOV M, B?
61. Byte dữ liệu 77h cần được chuyển vào ngăn nhớ, mà địa chỉ của ngăn nhớ này cất trong các ngăn nhớ địa chỉ 2020h và 2021h. Sử dụng lệnh LHLD addr và lệnh MVI M, data để thực hiện vận chuyển này?
62. Cần chuyển một byte dữ liệu từ một ngăn nhớ đến ngăn nhớ khác. Cặp thanh ghi HL chứa địa chỉ của ngăn nhớ nguồn, cặp thanh ghi DE chứa địa chỉ của ngăn nhớ đích. Hãy sử dụng các lệnh MOV r,M; MOV M,r và XCHG để thực hiện sự chuyển này?
63. Thanh ghi tích lũy (Accumulator) đóng vai trò gì khi xử lý các lệnh OR, AND và EXCLUSIVE?
64. Những nhóm lệnh nào tác động đến các cờ, cờ nào không bị tác động bởi lệnh?
65. Phân biệt lệnh JMP và CALL?

66. Cặp thanh ghi WZ đóng vai trò gì khi thực hiện lệnh nhảy?
67. Tín hiệu LOCK# có tác dụng:
- a: Khóa các ngắt;
 - b: Giữ bất kỳ thiết bị ngoại vi nào sử dụng bus;
 - c: Chỉ ra trạng thái bận của vi xử lý;
 - d: Chỉ ra chế độ của vi xử lý.
68. Cặp tín hiệu HOLD và HLDA:
- a: Thực hiện điều khiển thanh ghi đoạn đang được sử dụng;
 - b: Được thiết bị ngoại vi xử lý dụng để giành lấy điều khiển bus của 8086/8088;
 - c: Cả hai là các đầu ra;
 - d: Cả hai là tín hiệu vào.
69. Trong chế độ tối đa các tín hiệu S0#, S1#, và S2# được:
- a: Giải mã nhờ mạch điều khiển bus bên ngoài;
 - b: Thiết bị bên ngoài sử dụng để giành lấy điều khiển bus của vi xử lý;
 - c: Dùng để chỉ ra trạng thái hiện tại của hệ thống ngắt;
 - d: Luôn luôn ở mức thấp.
70. Mục đích của tín hiệu WAIT và tín hiệu đi cùng với nó, TEST# là:
- a: Đảm bảo một dạng khác của ngắt;
 - b: Cho phép những thiết bị ngoại vi có tốc độ (phản ứng) chậm đồng bộ với vi xử lý;
 - c: Đảm bảo hạ tốc độ phối ghép của vi xử lý với bộ nhớ tốc độ chậm;
 - d: Tất cả trên.
71. Tín hiệu WAIT gây ra sự kiểm tra đầu vào tín hiệu Nếu đầu vào này không thay đổi thì vi xử lý chờ một chu kỳ khác.
- a: TEST# hoặc BUSY#; b: INTR#; c: NMI; d: RESET.
72. Có thể kết nối 8086 ở chế độ tối thiểu với 8087 không, vì sao?
73. Bộ nhớ RAM tĩnh cất giữ dữ liệu trong:
- a: ROM; b: SRAM; c: Flip-Flop; d: Tụ điện.

74. Bộ nhớ động phải:
- a: Được làm tươi;
 - b: Được truy cập ngẫu nhiên;
 - c: Truy cập tuần tự;
 - d: Tất cả a, b, c.
75. Vì sao RAM của máy tính thường dùng bộ nhớ động?
76. Lập bảng so sánh ROM, EPROM, EAROM, EEPROM, FLASH MEMORY với các tiêu chí sau:
- a: Công nghệ bán dẫn;
 - b: Dữ liệu cất giữ;
 - c: Cần làm tươi;
 - d: Dung lượng;
 - e: Chi phí;
 - f: Phương pháp ghi;
 - g: Phương pháp xóa/thời gian xóa;
 - h: Thời gian truy cập.
77. Trong các công nghệ ROM (EPROM, EAROM, EEPROM, FLASH MEMORY) công nghệ nào ngày nay được áp dụng nhiều và tiện dụng nhất?
78. Công nghệ ASIC: sự khác nhau giữa mảng mạch cổng (gate array) và ô chuẩn (standard cell).
79. Hãy thiết kế một hệ thống nhớ cho hệ vi xử lý 8085 với dung lượng 640 kbyte RAM, 64 kbyte EPROM bằng các chip RAM tự chọn. Cho giải địa chỉ RAM, ROM?
80. Hãy thiết kế một hệ thống nhớ cho hệ vi xử lý 8086/8088 với dung lượng 24 kbyte RAM, 16 kbyte EPROM bằng các chip RAM tự chọn. Cho giải địa chỉ RAM, ROM?

81. FLASH MEMORY không cần điều khiển bên ngoài độ rộng xung lập trình bởi vì xung lập trình được điều khiển bởi:
- a: Logic bên trong FLASH MEMORY;
 - b: Phần mềm của vi xử lý;
 - c: Các chu kỳ thời gian đọc/ghi bộ nhớ của vi xử lý;
 - d: Không phải tất cả trên, vì xung lập trình không cần điều khiển.
82. Tìm hiểu một số loại EPROM, EEPROM và FLASH MEMORY (nguồn trên Internet) và thiết kế thử sơ đồ bộ ghi đọc chúng, sử dụng một trong các chip vi xử lý 8-bit tùy chọn: Z80, 8051, 8085, 8086/8088. Hoặc đây là thiết bị độc lập, hoặc được ghép nối với IBM PC (viết chương trình điều khiển)?
83. Có thể dùng chung mạch lập trình cho EEPROM và FLASH MEMORY hay không?
84. Trước khi lập trình FLASH MEMORY cần phải:
- a: Xóa toàn bộ tất cả các sector của FLASH MEMORY;
 - b: Chỉ xóa sector cần lập trình;
 - c: Thực hiện lưu giữ nội dung của FLASH MEMORY vào bộ nhớ chính phòng trừ trường hợp lập trình không thực hiện thành công;
 - d: Tất cả trên.
85. Mục đích ban đầu của thẻ PCMCIA là:
- a: Cung cấp một thẻ ROM cho máy tính xách tay;
 - b: Một cách chuẩn để kết nối mở rộng cho bộ nhớ của PC khi không có không gian cho các bảng nhớ chuẩn;
 - c: Một cách để bổ sung thiết bị nhớ ngoài loại nhỏ cho PC nhỏ;
 - d: Tất cả trên.

86. Khi thẻ PCMCIA được sử dụng để bổ sung bộ nhớ Flash ROM vào cho PC, mục đích của nó là:
- a: Cung cấp một thẻ ROM cho máy tích xách tay.
 - b: Một cách chuẩn để kết nối mở rộng bộ nhớ của PC khi không có không gian cho các băng nhớ chuẩn.
 - c: Một cách để bổ sung thiết bị nhớ ngoài loại nhỏ cho PC nhỏ.
 - d: Tất cả trên.
87. Một phần của chuẩn PCMCIA xác định một bộ nối 68 chân và các tín hiệu trên từng chân của bộ nối. Vậy các tín hiệu kết nối với các chân 18 (Vpp1) và chân 52 (Vpp2) là:
- a: Cung cấp điện thế cao (+12 Vdc) cần thiết cho Flash ROM để xóa và lập trình bộ nhớ này;
 - b: Không cần thiết khi Flash ROM chỉ dùng điện thế thấp (+5 Vdc);
 - c: Không cần thiết khi thẻ PCMCIA kết nối để đọc;
 - d: Tất cả trên.
88. Giải mã chọn các băng nhớ chính (RAM) được thực hiện như thế nào. Hãy chỉ ra trong sơ đồ bo mạch chủ và viết các biểu thức logic cho những tín hiệu chọn này. Vùng địa chỉ RAM là bao nhiêu?
89. Giải mã chọn các băng nhớ EPROM được thực hiện như thế nào. Hãy chỉ ra trong sơ đồ bo mạch chủ và viết các biểu thức logic cho những tín hiệu chọn này. Vùng địa chỉ EPROM là bao nhiêu?
90. Vẽ đồ thị xung quá trình làm tươi DRAM trong hệ thống IBM PC/XT. Giải thích?
91. Hãy tìm hiểu thêm về phương pháp lập trình các mạch ASIC (nguồn trên Internet)?
92. Cho mẫu dữ liệu 11010001, hãy vẽ đồ thị phương pháp ghi MFM và FM, và tạo ra đa thức kiểm tra dữ liệu D(x)?
93. Sự khác nhau chính giữa thiết bị đĩa từ Winchester và các hệ thống đĩa cứng nói chung?

94. Phân loại các thiết bị băng từ. Nguyên tắc ghi, đọc thông tin trên băng từ với đĩa từ có giống nhau không, vì sao?
95. Hiện nay công nghệ thiết bị nhớ trên đĩa từ như thế nào (hãy tìm hiểu thêm từ nguồn Internet)?
96. Có thể dùng thiết bị nhớ trên băng từ làm nơi lưu trữ hệ điều hành được không, vì sao?
97. Phân loại CD-ROM. Các loại CD-ROM hiện nay có tốc độ bao nhiêu, phương pháp ghi đọc như thế nào?
98. Phân biệt các kết nối thiết bị nhớ ngoài đang sử dụng trong các máy tính hiện nay: IDE, EIDE, SCSI, SCSI-2, SCSI-3, và những chủng loại máy tính nào sử dụng chúng (tìm hiểu thêm nguồn từ Internet)?
99. Vẽ đồ thị xung quá trình làm tươi DRAM trong hệ thống IBM PC/XT. Giải thích?

Chương 3

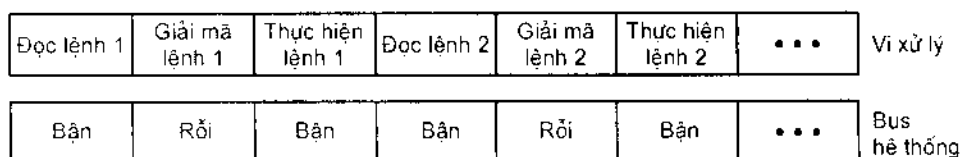
TẬP LỆNH CỦA VI XỬ LÝ

3.1. KHÁI NIỆM VỀ LỆNH, DẠNG LỆNH VÀ CÁCH MÃ HÓA LỆNH CỦA VI XỬ LÝ

3.1.1. Lệnh và thực hiện lệnh

Lệnh của vi xử lý là một từ nhị phân. Khi lệnh được đọc từ bộ nhớ vào vi xử lý, lệnh chỉ dẫn vi xử lý thực hiện một nhiệm vụ nào đó. Mỗi một lệnh chỉ dẫn vi xử lý thực hiện nhiệm vụ riêng không giống với lệnh khác. Hầu hết các lệnh của vi xử lý là những lệnh vận chuyển hoặc xử lý dữ liệu. Dữ liệu có thể có trong bộ nhớ bên ngoài vi xử lý hoặc là nội dung của các thanh ghi bên trong vi xử lý. Một số lệnh khác có chức năng điều khiển vi xử lý, rẽ nhánh xử lý,...

Tất cả các lệnh được thực hiện trong 3 pha: đọc lệnh (Fetch), giải mã lệnh (Decode) và thực hiện lệnh (Execute) (có thể truy cập bộ nhớ để đọc các toán hạng và cất giữ các kết quả thông qua bus hệ thống). Các pha đọc lệnh và thực hiện lệnh có sự trao đổi dữ liệu giữa vi xử lý và bộ nhớ thông qua bus hệ thống, do đó, những quãng thời gian này bus hệ thống bận (busy). Pha giải mã lệnh chỉ xảy ra bên trong vi xử lý, do đó bus rỗi (idle). Quá trình thực hiện tuần tự các lệnh ở các hệ thống vi xử lý 8-bit diễn ra như mô tả trong hình 3.1.



Hình 3.1: Quá trình thực hiện tuần tự các lệnh trong các hệ thống vi xử lý 8-bit

Các loại vi xử lý tiên tiến hiện nay có thể xử lý đồng thời một số lệnh ở các pha khác nhau của quá trình thực hiện lệnh nhờ có kiến trúc đường ống (pipeline). Hình 3.2 mô tả quá trình thực hiện đồng thời một số lệnh của vi xử lý Intel 80486 với kiến trúc đường ống lệnh.

Đọc lệnh 1	Đọc lệnh 2	Đọc lệnh 3	Đọc lệnh 4	Lưu trữ 1	Đọc lệnh 5	Đọc lệnh 6	Đọc bộ nhớ 2	Đọc lệnh 7	Đơn vị bus BU
	Giải mã lệnh 1	Giải mã lệnh 2	Giải mã lệnh 3	Giải mã lệnh 4	Rỗi	Giải mã lệnh 5	Giải mã lệnh 6	Rỗi	Đơn vị lệnh IU
		Thực hiện lệnh 1	Thực hiện lệnh 2	Thực hiện lệnh 3	Thực hiện lệnh 4	Rỗi	Thực hiện lệnh 5	Thực hiện lệnh 6	Đơn vị thực hiện
			Địa chỉ 1			Địa chỉ 2			Đơn vị địa chỉ

Hình 3.2: Quá trình thực hiện đồng thời một số lệnh của các vi xử lý tiên tiến có kiến trúc đường ống lệnh (ví dụ Intel 80486)

Thời gian thực hiện xong một lệnh gọi là một chu kỳ lệnh. Mỗi một pha gồm một số chu kỳ máy (M1, M2, M3,...). Mỗi chu kỳ máy gồm một số chu kỳ nhịp đồng hồ. Một chu kỳ nhịp đồng hồ là một trạng thái. Một chu kỳ máy ngắn nhất kéo dài 3 chu kỳ nhịp đồng hồ. Thời gian truy cập vào bộ nhớ cần tối thiểu 3 chu kỳ nhịp đồng hồ (T1, T2, T3).

Một chu kỳ lệnh có thể gồm có:

1. Chu kỳ nhận lệnh (instruction fetch)
2. Chu kỳ đọc bộ nhớ (memory read)
3. Chu kỳ ghi bộ nhớ (memory write)
4. Chu kỳ đọc vào/ra (I/O read)
5. Chu kỳ ghi vào/ra (I/O write)
6. Chu kỳ chấp nhận ngắt (interrupt acknowledge)
7. Bus rỗi (bus idle).

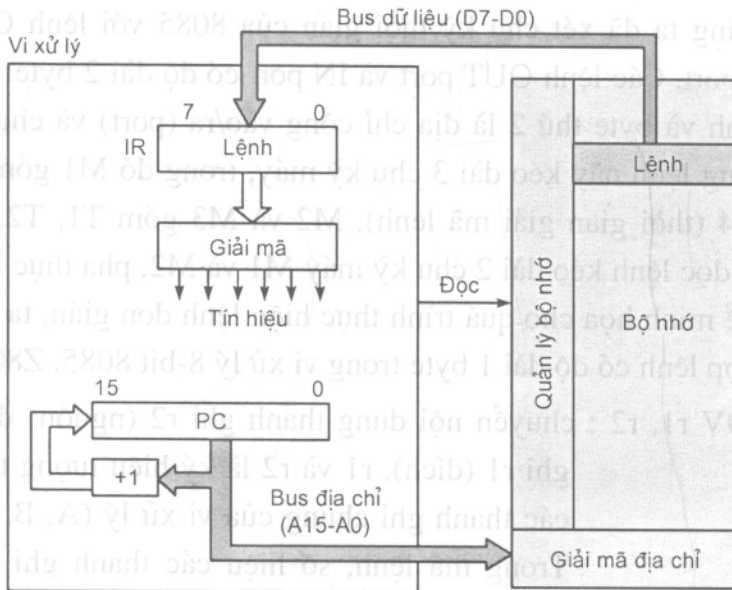
Tổng số chu kỳ máy cần thiết để thực hiện toàn bộ một lệnh phụ thuộc vào từng lệnh, và độ dài của từ lệnh (1 byte hay một số byte).

Chúng ta đã xét chu kỳ thời gian của 8085 với lệnh OUT port hoặc IN port. Các lệnh OUT port và IN port có độ dài 2 byte: byte đầu là mã lệnh và byte thứ 2 là địa chỉ cổng vào/ra (port) và chu kỳ thực hiện những lệnh này kéo dài 3 chu kỳ máy, trong đó M1 gồm T1, T2, T3, và T4 (thời gian giải mã lệnh), M2 và M3 gồm T1, T2, T3. Pha đọc lệnh kéo dài 2 chu kỳ máy M1 và M2, pha thực hiện lệnh ở M3. Để minh họa cho quá trình thực hiện lệnh đơn giản, ta lấy ví dụ trường hợp lệnh có độ dài 1 byte trong vi xử lý 8-bit 8085, Z80:

MOV r1, r2 ; chuyển nội dung thanh ghi r2 (nguồn) đến thanh ghi r1 (đích), r1 và r2 là ký hiệu tượng trưng cho các thanh ghi chung của vi xử lý (A, B, C, D,...). Trong mã lệnh, số hiệu các thanh ghi chung là SSS (nguồn) và DDD (đích). Tương tự, trong 8080 là LD r1, r2.

Các tác động của lệnh theo thời gian của chu kỳ máy và các chu kỳ nhịp đồng hồ được diễn giải trong hình 3.3. Trong đó, quá trình đọc 1 byte lệnh từ bộ nhớ và chu kỳ lệnh của lệnh MOV r1, r2 chỉ thực hiện ở chu kỳ máy M1 kéo dài thêm 1 chu kỳ nhịp đồng hồ T5 để thực hiện vận chuyển dữ liệu đến thanh ghi đích (có số hiệu là DDD).

MNEMONIC:	MOV r1, r2	
	11010011	
OPCODE:	01DDDSSS	
	T1	PC OUT: Địa chỉ ngăn nhớ chứa mã lệnh đưa ra bus địa chỉ (A15-A0)
		Tín hiệu đọc (read) được tạo ra trên bus điều khiển
M1	T2	PC = PC+1 Nội dung PC tăng lên 1
		Giải mã địa chỉ chọn ngăn nhớ, đọc ngăn nhớ
	T3	Mã lệnh được đưa lên bus dữ liệu (D7-D0) và vào IR
	T4	Giải mã lệnh
		Nội dung thanh ghi nguồn SSS chuyển đến thanh ghi tạm thời (temp)
	T5	Nội dung thanh ghi tạm thời chuyển đến thanh ghi đích DDD



Hình 3.3: Quá trình đọc mã lệnh từ bộ nhớ (Instruction fetch)

3.1.2. Các dạng lệnh, mã lệnh

Khi chọn các dạng lệnh, người thiết kế quan tâm đến các yếu tố sau đây:

- Số lượng các lệnh được biểu diễn.
- Khả năng đánh địa chỉ của các chế độ đánh địa chỉ.
- Dễ dàng mã hóa các lệnh.
- Kiểu của trường lệnh (cố định hay thay đổi).
- Chi phí về phần cứng cần thiết để giải mã và thực hiện lệnh.

Lệnh của vi xử lý là một từ nhị phân có dạng chung gồm có hai thành phần:

Mã lệnh (Opcode), và các địa chỉ của toán hạng (Addresses of operands).

Từ lệnh của vi xử lý có độ dài khác nhau phụ thuộc vào loại lệnh. Các loại vi xử lý từ 8-bit đến 64-bit có các từ lệnh độ dài tối thiểu 8-bit. Phụ thuộc vào độ dài từ xử lý và khả năng đánh địa chỉ độ dài từ

lệnh của chúng có thể dài 16 bit, 24 bit, 32 bit, 64 bit. Một số lệnh phức tạp có thể có độ dài đến 3 từ. Mã lệnh có độ dài khác nhau, phụ thuộc vào loại vi xử lý. Nếu mã lệnh có 1 byte, nghĩa là có thể có tới 256 lệnh khác nhau trong tập lệnh của bộ vi xử lý. Một số lệnh của các bộ vi xử lý 8-bit mà các toán hạng nằm trong các thanh ghi thì phần mã lệnh dài 5 bit, phần địa chỉ dài 3 bit. Các bộ vi xử lý 16-bit và 32-bit có mã lệnh dài 1 byte. Trong một số loại vi xử lý, có thể dùng 2 byte cho mã lệnh, nhưng byte thứ 2 không dùng hết các bit, nghĩa là có thể mã hóa tới gần 2^{16} lệnh khác nhau.

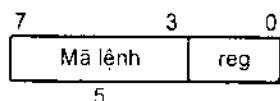
Toán hạng có thể là nội dung của thanh ghi bên trong bộ vi xử lý, có thể là nội dung ô nhớ, hay là một cổng vào/ra. Các thanh ghi viết trong lệnh được ký hiệu bằng các chữ cái, ví dụ, thanh ghi tích lũy (accumulator) là A.

Địa chỉ của toán hạng (ngăn nhớ, thanh ghi, cổng vào/ra) có độ dài từ 1 byte đến 4 byte tùy thuộc vào khả năng đánh địa chỉ của vi xử lý. Ví dụ, Z80 có 16-bit địa chỉ, do vậy phần địa chỉ của lệnh có 2 byte, nếu đó là địa chỉ của ngăn nhớ chứa toán hạng. Nếu là lệnh vào/ra như IN n, OUT n, thì n là địa chỉ 8-bit của cổng vào/ra. Trong vi xử lý 32-bit, phần địa chỉ có thể dài 24 bit (3 byte) hay 32-bit (4 byte), hoặc mở rộng đến 7 byte.

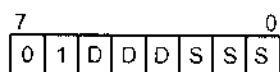
(1) Các dạng lệnh của vi xử lý 8-bit (8080/Z80/8085)

Các vi xử lý 8-bit có một số dạng lệnh cơ bản (instruction formats): lệnh có độ dài 1 từ, 2 từ, 3 từ, mỗi từ có độ dài 1 byte. Độ dài của lệnh phụ thuộc và các toán hạng là thanh ghi, bộ nhớ, hay cổng vào/ra. Có các loại thao tác dữ liệu thanh ghi-thanh ghi (register-to-register), thanh ghi-bộ nhớ (register-to-memory), bộ nhớ-bộ nhớ (memory-to-memory), gọi và nhánh (call, branch). 8080 sử dụng các lệnh dài 1, 2, hoặc 3 byte. Z80 được trang bị bổ sung một số lệnh chỉ số cần thêm 1 byte nữa. Một số lệnh của Z80 có phần mã lệnh có độ dài tới 2 byte.

+ Dạng lệnh dài 1 byte



Các lệnh thao tác giữa thanh ghi tích lũy và thanh ghi chung, trong đó, reg là số hiệu thanh ghi chung



Các lệnh thao tác giữa 2 thanh ghi chung, trong đó, SS, DDD là số hiệu thanh ghi nguồn và đích tương ứng (lệnh LD r,r')

Lệnh dài 1 byte thường là các lệnh thực hiện xử lý dữ liệu giữa các thanh ghi với nhau (thanh ghi-thanh ghi), hoặc giữa ngăn nhớ mà địa chỉ chứa trong cặp thanh ghi (HL) với thanh ghi (memory-to-register, register-to-memory), ví dụ:

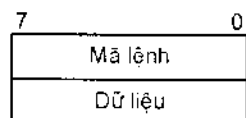
LD r, r'; MOV r1, r2; MOV r, M; ADD r; ADD M; CMP r,...

Chu kỳ thực hiện lệnh có thể kéo dài 1 chu kỳ máy (như lệnh RD r,r') hoặc 2 chu kỳ máy (như các lệnh thao tác giữa ngăn nhớ và thanh ghi).

+ Dạng lệnh dài 2 byte

Trong từ lệnh 2 byte, byte thứ nhất (đọc ra IR đầu tiên từ bộ nhớ) là mã lệnh, byte thứ 2 là có thể là dữ liệu (data) trực tiếp (immediate), ví dụ:

MVI M, data; ADD A, n; IN port; OUT port,...



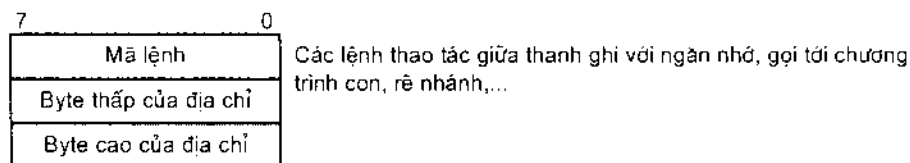
Các lệnh thao tác giữa thanh ghi với dữ liệu trực tiếp (immediate)

Đó là các lệnh thao tác giữa thanh ghi hoặc ngăn nhớ mà địa chỉ chứa trong cặp thanh ghi (HL) và dữ liệu là 1 byte ghi trong lệnh. Chu kỳ thực hiện các lệnh này kéo dài 2 hoặc 3 chu kỳ máy.

+ Dạng lệnh dài 3 byte

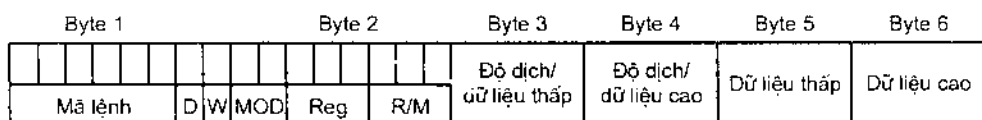
Các lệnh dạng này có thao tác giữa ngăn nhớ và thanh ghi, rẽ nhánh, gọi tới chương trình con, mà địa chỉ ngăn nhớ là 2 byte tiếp theo ghi trong lệnh, ví dụ:

LDA addr; STA addr; CALL addr; JMP addr, SHLD addr;... Chu kỳ thực hiện các lệnh này kéo dài từ 3 đến 5 chu kỳ máy.



(2) Các dạng lệnh của vi xử lý 16-bit 8086/8088/80286

Các vi xử lý 8086/8088/80286 xử lý từ 16-bit. Chúng có các dạng lệnh dài: từ 1 byte đến 6 byte. Cấu trúc chung của các dạng lệnh của 8086/8088 minh họa trong hình 3.4.



Hình 3.4: Khuôn dạng lệnh trong Intel 8086/8088

Trong đó, các trường có ý nghĩa như sau:

- R/M: trường chỉ toán hạng thanh ghi/các thanh ghi (register operand/registers) dùng để tính EA.
- REG: trường toán hạng thanh ghi/mở rộng (register operand/extension) cho mã lệnh (opcode).
- MOD: trường chế độ thanh ghi/bộ nhớ (register mode/memory mode) với độ dài của độ dịch (displacement).
- W: trường từ/byte (word/byte) để chỉ thao tác lệnh theo từ hay byte.
- D: trường hướng đến thanh ghi hoặc từ thanh ghi (direction to/from register).
- OPCODE: trường mã lệnh.

+ Dạng lệnh dài 1 byte

Có một số lệnh chỉ thao tác bên trong thanh ghi 8-bit AL cho các chuyển đổi số, ví dụ:

XLAT: chuyển đổi byte trong AL,

AAA: hiệu chỉnh số ASCII cho phép cộng,

DAA: hiệu chỉnh số hệ 10 cho phép cộng

DAS: hiệu chỉnh số hệ 10 cho phép trừ,...

Trong cấu trúc lệnh chỉ có mã lệnh 8-bit mà thôi, và chu kỳ thực hiện các lệnh chỉ có 1 chu kỳ máy với 4 chu kỳ nhịp đồng hồ.

+ Các dạng lệnh dài 2 byte, 3 byte, 4 byte, 5 byte, 6 byte

Phụ thuộc vào chế độ đánh địa chỉ, thao tác với bộ nhớ, xử lý theo từ hay theo byte ($W = 1/0$) độ dài của lệnh có thể dài 2, 3, 4, 5, hoặc 6 byte. Ví dụ, nếu thực hiện lệnh vận chuyển (MOV) giữa ngăn nhớ hoặc thanh ghi với một từ dữ liệu 2 byte ghi trong lệnh thì lệnh này có độ dài 4 byte, trong đó byte thứ 3, thứ 4 là giá trị dữ liệu:

Byte 1						Byte 2				Byte 3				Byte 4			
1	1	0	0	0	1	1			0	0	0					Dữ liệu thấp	Dữ liệu cao
Mã lệnh						D	W	MOD	Reg		R/M						

Nếu thực hiện gọi (CALL) hoặc nhảy (JMP) bên trong đoạn (nhảy gần) thì lệnh có độ dài 3 byte, trong đó byte 2 và byte 3 là giá trị độ dịch 16-bit (displacement) trong đoạn. Nếu gọi hay nhảy tới một đoạn (nhảy xa) khác thì độ dài lệnh có 5 byte, trong đó byte 2 là giá trị độ dịch 16-bit trong đoạn mới, và byte 4 byte 5 là giá trị địa chỉ cơ sở 16-bit của đoạn mới.

(3) Các dạng lệnh của vi xử lý 32-bit

Tập lệnh của các vi xử lý 32-bit có tập lệnh được bổ sung những khả năng mã hóa mới so với các vi xử lý 8-bit và 16-bit.

a) Các bộ vi xử lý Intel 32-bit (từ 80386 đến Pentium)

Khuôn dạng lệnh của các bộ vi xử lý 32-bit Intel (từ 80386 đến Pentium) được mô tả trong hình 3.5.

1 - 4 byte	1 - 2 byte	0 - 1 byte	0 - 1 byte	0 - 4 byte	0 - 4 byte
Các mào đầu của lệnh	Mã lệnh	Mod R/M	SIB	Độ dịch	Dữ liệu trực tiếp

Hình 3.5: Khuôn dạng lệnh trong các vi xử lý 32-bit (từ 80386 đến Pentium)

Trong đó các trường có ý nghĩa như sau:

- *Các mào đầu của lệnh (Instruction prefixes)*

Các mào đầu của lệnh là trường có thể có tới 4 mào đầu, mỗi mào đầu dài 1 byte. Như vậy, kích thước tối đa của trường các mào đầu của lệnh dài 4 byte. Mỗi mào đầu thuộc 1 trong 4 nhóm mào đầu khác nhau và có thể đặt không theo thứ tự với các giá trị sau:

- Nhóm 1: các mào đầu cho các lệnh khóa và lặp (Lock and Repeat prefixes):
 - + F0h: mào đầu lệnh LOCK
 - + F2h: mào đầu các lệnh lặp REPNE/REPZ theo điều kiện NE/NZ (chỉ dùng cho các lệnh chuỗi)
 - + F3h: mào đầu lệnh lặp REP (chỉ dùng cho các lệnh chuỗi)
 - + F3h: mào đầu các lệnh lặp REPE/REPZ theo các điều kiện E/Z (chỉ dùng cho các lệnh chuỗi)
- Nhóm 2:
 - + Các mào đầu cho các trường hợp bỏ qua đoạn (segment override):
 - 2Eh: mào đầu bỏ qua đoạn mã (CS)
 - 36h: mào đầu bỏ qua đoạn ngăn xếp (SS)
 - 3Eh: mào đầu bỏ qua đoạn dữ liệu (DS)
 - 26h: mào đầu bỏ qua đoạn mở rộng (ES)
 - 64h: mào đầu bỏ qua đoạn FS
 - 65h: mào đầu bỏ qua đoạn GS
 - + Hỗ trợ cho các lệnh rẽ nhánh:
 - 2Eh: rẽ nhánh không xảy ra (chỉ dùng với các lệnh nhảy có điều kiện)
 - 3Eh: rẽ nhánh xảy ra (chỉ dùng với các lệnh nhảy có điều kiện).

- Nhóm 3: mào đầu bỏ qua kích thước toán hạng (Operand-size override): 66h
- Nhóm 4: mào đầu bỏ qua kích thước địa chỉ (Address-size override): 67h

Mào đầu lệnh LOCK đảm bảo loại trừ sự sử dụng bộ nhớ chia sẻ trong môi trường đa xử lý với tín hiệu LOCK#.

Các mào đầu lệnh lặp REP tác động để lặp lại thực hiện các lệnh chuỗi như MOVS, CMPS, SCAS, STOS, INS, và OUTS.

Các mào đầu hỗ trợ các lệnh rẽ nhánh cho phép chương trình thực hiện các lệnh rẽ nhánh có điều kiện.

Tất cả các lệnh tham chiếu bộ nhớ của các vi xử lý Intel 32-bit đều sử dụng thanh ghi đoạn mặc định. Như vậy khi chọn một chế độ đánh địa chỉ cho lệnh, một thanh ghi đoạn được chọn tự động. Nếu không muốn sử dụng thanh ghi đoạn mặc định thì có thể sử dụng một thanh ghi đoạn khác. Để có thể chọn thanh ghi đoạn khác cần phải bổ sung vào từ lệnh mào đầu bỏ qua đoạn (segment override). Mào đầu bỏ qua kích thước toán hạng cho phép chương trình chuyển kích thước toán hạng giữa 16-bit và 32-bit. Tương tự, mào đầu bỏ qua kích thước địa chỉ cho phép chương trình chuyển đánh địa chỉ giữa 16-bit và 32-bit.

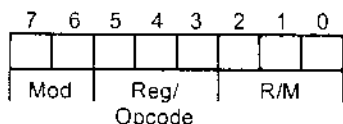
• Mã lệnh

	6	5	4	3	2	1	0
Mã lệnh						D	W

Mã lệnh (Opcode) là trường mã lệnh đầu tiên (Primary Opcode), có độ dài 1 hoặc 2 byte. Trường mã lệnh bổ xung gồm có 3-bit nằm trong trường Mod R/M. Tương tự như ở các vi xử lý Intel 16-bit, bên trong trường Opcode có các trường con. Những trường con này xác định hướng của tác động lệnh hay thao tác lệnh, kích thước của giá trị độ dịch, số hiệu thanh ghi, mã điều kiện, hoặc dấu mở rộng.

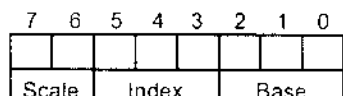
• *Mod R/M và SIB*

Mod R/M và SIB là các trường có độ dài 1 byte. ModR/M là byte định dạng đánh địa chỉ, có ý nghĩa tương tự như trong lệnh của các vi xử lý 16-bit 8086/8088.



Trường Mod kết nối với trường R/M hình thành 32 giá trị khác nhau: 8 thanh ghi và 24 chế độ đánh địa chỉ. Trường Reg/Opcode (thanh ghi/mã lệnh bổ sung) xác định hoặc là số hiệu thanh ghi hoặc là 3 bit của mã lệnh bổ sung (ghép với trường mã lệnh đầu tiên). Trường R/M có thể xác định số hiệu thanh ghi như là một toán hạng hoặc là kết hợp với trường Mod để mã hóa chế độ đánh địa chỉ.

Tổ hợp mã hóa của byte ModR/M cần thêm 1 byte đánh địa chỉ thứ hai, đó là SIB (Scale-Index-Base) để hình thành một khuôn dạng đánh địa chỉ 32-bit đầy đủ của vi xử lý Intel 32-bit. Byte SIB không có trong vi xử lý 16-bit 8086/8088.



SIB (8 bit) gồm có các trường:

+ Scale (2 bit): xác định hệ số cấp độ nhân (SCALE) với chỉ số.

Cụ thể:

Scale = 00, SCALE = 1. Scale = 01, SCALE = 2, Scale = 10, SCALE = 4.

Scale = 11, SCALE = 8.

Index $\times$ SCALE

+ Index (3 bit): xác định số hiệu của thanh ghi chỉ số

+ Base (3 bit): xác định số hiệu của thanh ghi cơ sở

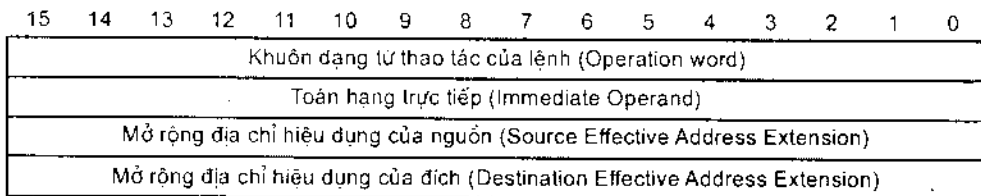
• *Giá trị độ dịch và toán hạng ngay lập tức*

Một số dạng đánh địa chỉ bao gồm giá trị độ dịch (displacement) kèm ngay theo byte ModR/M (hoặc byte SIB nếu có). Giá trị độ dịch

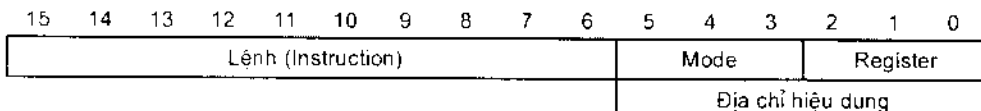
có thể dài bằng 1, 2 hoặc đến 4 byte. Nếu lệnh có xác định toán hạng ngay lập tức (immediate operand) thì toán hạng phải đi ngay sau giá trị độ dịch. Toán hạng ngay lập tức có độ dài bằng 1, 2 hoặc 4 byte.

b) Họ vi xử lý Motorola M68000

Các dạng lệnh của họ Motorola M68000 khác nhiều so với các vi xử lý của Intel. Nhờ sử dụng 8 thanh ghi dữ liệu (D0-D7) và 8 thanh ghi địa chỉ (A0-A7) thông qua các trường Mode và Register nên mã hóa lệnh của họ vi xử lý Motorola tạo ra nhiều lệnh với các chế độ đánh địa chỉ khác nhau tương tự như ở các vi xử lý tiên tiến của Intel. Hình 3.6 là khuôn dạng lệnh của họ Motorola M68000.



Khuôn dạng tứ thao tác của lệnh (Operation word)



Trong đó:

Operation word: là tứ thao tác, nó xác định những thao tác và các chế độ đánh địa chỉ của lệnh.

Immediate Operand: là toán hạng trực tiếp, nó là giá trị số ghi trong lệnh và có độ dài 1 byte, 2 byte hay 4 byte.

Source Effective Address Extension: là mở rộng địa chỉ hiệu dụng của nguồn, nó có thể có độ dài 1 hoặc 2 từ.

Destination Effective Address Extension: là mở rộng địa chỉ hiệu dụng của đích, nó có thể có độ dài 1 hoặc 2 từ.

Instruction: là trường mã lệnh 10-bit (từ bit 6 đến bit 15), nó chỉ ra thao tác lệnh.

Mode: là trường 3-bit chế độ đánh địa chỉ của lệnh (từ bit 3 đến bit 5).

Register: là trường 3-bit thanh ghi (từ bit 0 đến bit 2), nó xác định số hiệu thanh ghi trong vi xử lý.

Chú ý rằng M68000/M68010 có 23 bit địa chỉ bộ nhớ và 16 bit dữ liệu, còn M68020/M68030/M68040/M68060 có 32 bit địa chỉ bộ nhớ và 32 bit dữ liệu.

Hình 3.6: Khuôn dạng lệnh của họ vi xử lý Motorola M68000

3.2. CÁC PHƯƠNG PHÁP ĐÁNH ĐỊA CHỈ CỦA VI XỬ LÝ

Lệnh của vi xử lý gồm có hai thành phần: mã lệnh và địa chỉ. Nhưng một số lệnh không cần có địa chỉ thực, mà có địa chỉ ẩn, hoặc không có địa chỉ mà thay vào đó là giá trị trực tiếp của toán hạng. Những lệnh không cần có các địa chỉ riêng là những ngoại lệ. Dưới đây các phương pháp (hay chế độ) đánh địa chỉ của hầu hết các loại vi xử lý từ 8-bit đến tiên tiến hiện nay.

3.2.1. Đánh địa chỉ trực tiếp thanh ghi (implied, register direct)

Toán hạng là nội dung của thanh ghi nào đó trong vi xử lý và số hiệu của thanh ghi được ghi trong từ lệnh. Các loại lệnh với 2 toán hạng: register, register đặc trưng cho dạng đánh địa chỉ này.

a) Trong vi xử lý 8-bit (8080/8085/Z80)

0	1	D	D	D	S	S	S
---	---	---	---	---	---	---	---

DDD là số hiệu thanh ghi đích, SSS là số hiệu thanh ghi nguồn
MOV A,B; chuyển nội dung thanh ghi B vào thanh ghi tích lũy A

0	1	1	1	1	0	0	0
---	---	---	---	---	---	---	---

b) Trong vi xử lý Intel 16-bit (8086/8088/80286)

Phương pháp đánh địa chỉ trực tiếp thanh ghi trong các vi xử lý Intel từ 16-bit đến 32-bit gọi là đánh địa chỉ thanh ghi (register addressing). Tám thanh ghi chung của chúng có thể dùng 8-bit, 16-bit (8086/8088/ 80386) hay 32-bit (từ 80386 đến Pentium), tức là có thể thực hiện xử lý dữ liệu trong các thanh ghi chung theo byte hay theo từ. Trong lệnh sử dụng chế độ đánh địa chỉ này các trường (hình 3.4) có các giá trị: MOD = 11, R/M và REG là 3 bit số hiệu của 8 thanh ghi chung (chứa 2 toán hạng của lệnh), và W = 0 nếu xử lý theo byte, W = 1 nếu xử lý theo từ. Ví dụ:

MOV AX,CX; $AX \leftarrow (CX)$: trong từ lệnh trường MOD = 11, REG = 000 chỉ AX, R/M = 001 chỉ CX và W = 1 chỉ thực hiện từ 16-bit (xem bảng 3.2 ở phần sau).

c) Trong vi xử lý Intel 32-bit

Tương tự như ở Intel 16-bit.

MOV EAX, EBX; $EAX \leftarrow (EBX)$

SUB EAX, EAX; $EAX \leftarrow (EAX) - (EBX)$

INC EBX; $EBX \leftarrow (EBX) + 1$.

d) Trong họ vi xử lý Motorola M68000

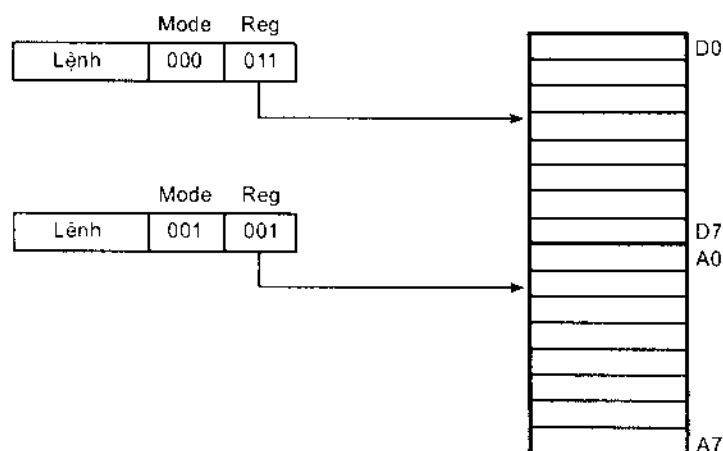
Các lệnh sử dụng chế độ đánh địa chỉ trực tiếp thanh ghi trong họ vi xử lý Motorola phải có giá trị trường Mode = 000 nếu dùng các thanh ghi dữ liệu Dn (D0 - D7), ví dụ lệnh:

EXG Dx, Dy; $Dx \leftrightarrow Dy$, trao đổi nội dung của các thanh ghi dữ liệu Dx và Dy (D0 - D7).

và Mode = 001 nếu dùng các thanh ghi địa chỉ An (A0 - A7), ví dụ lệnh:

EXG Ax, Ay; $Ax \leftrightarrow Ay$, trao đổi nội dung của các thanh ghi địa chỉ Ax và Ay (A0 - A7).

Trường thanh ghi (Reg) ghi số hiệu của thanh ghi được tham chiếu (hình 3.7).



Hình 3.7: Đánh địa chỉ trực tiếp thanh ghi trong họ Motorola M68000

3.2.2. Đánh địa chỉ ngay lập tức (immediate)

Toán hạng là một phần của lệnh, nó là một giá trị cụ thể (một hằng số). Các loại lệnh với 2 toán hạng: register, immediate và memory, immediate đặc trưng cho dạng đánh địa chỉ này.

a) Trong vi xử lý 8-bit (8080/8085/Z80)

MVI C, 4Dh; $C \leftarrow 4Dh$

0	0	0	0	1	1	1	0	0Eh
0	1	0	0	1	1	0	1	4D

b) Trong vi xử lý Intel 16-bit (8086/8088/80286)

ADD AX, 1010h; $AX \leftarrow (AX) + 1010h$

OR AL, 01101100B, $AL \leftarrow (AL) \vee 01101100B$

c) Trong vi xử lý Intel 32-bit

ADD EAX, 12342143h; $EAX \leftarrow (EAX) + 12342143h$;

d) Trong họ vi xử lý Motorola M68000

ADD #50, D1; $D1 \leftarrow (D1) + 50$. Đứng trước giá trị số là dấu #.

Giá trị số có thể viết ở mã 16 (h), nhị phân (b), bát phân (o), thập phân.

3.2.3. Đánh địa chỉ trực tiếp hay địa chỉ tuyệt đối (direct)

Địa chỉ của toán hạng trong bộ nhớ là một phần của từ lệnh. Nó thường là một tên tượng trưng nào đó mà trước đó đã có lệnh gán giá trị cụ thể, hay là một giá trị địa chỉ cụ thể. Tên cụ thể có thể là nhãn (label) của một đoạn chương trình.

a) Trong vi xử lý 8-bit (8080/8085/Z80)

Các lệnh sử dụng địa chỉ trực tiếp có thể có độ dài từ 2 đến 3 byte. Byte thứ nhất là mã lệnh, các byte thứ 2 và 3 là địa chỉ 16-bit của vùng nhớ toán hạng trong bộ nhớ. Địa chỉ 16-bit có thể đánh địa chỉ trực tiếp một không gian nhớ gồm 65536 (64 kbyte) byte, từ giá trị 0000h đến FFFFh.

LDA addr; nạp trực tiếp giá trị địa chỉ vào thanh ghi tích lũy A.

Cụ thể:

0	0	1	1	1	0	1	0
Byte thấp của địa chỉ 16-bit							
Byte cao của địa chỉ 16-bit							

JMP 1200h; nhảy tới đoạn chương trình có địa chỉ là 1200h.

b) Trong vi xử lý Intel 16-bit (8086/8088)

Trong 8086/8088, đánh địa chỉ trực tiếp sử dụng các thanh ghi đoạn (CS, DS, SS, ES) 16-bit chứa giá trị địa chỉ cơ sở của đoạn nhớ và giá trị độ dịch của ngăn nhớ (chứa toán hạng) so với địa chỉ cơ sở trong đoạn. Mỗi một đoạn nhớ có dung lượng tối đa là 64 kbyte. Như vậy, giá trị độ dịch và địa chỉ phải là các từ 16-bit. Trong các lệnh dùng đánh địa chỉ trực tiếp, phần địa chỉ gồm byte thứ 2 và thứ 3 là địa chỉ logic chính và là giá trị độ dịch của ngăn nhớ trong đoạn. Địa chỉ vật lý (địa chỉ thực) 20-bit của ngăn nhớ là kết quả của phép cộng giá trị độ dịch 8-bit hoặc 16-bit ghi ở byte thứ 2 (độ dịch 8-bit) hoặc các byte thứ 2 và 3 (độ dịch 16-bit) của lệnh với giá trị địa chỉ cơ sở 16-bit của đoạn được dịch trái 4 bit. Phép dịch trái địa chỉ cơ sở 16-bit ghi trong thanh ghi đoạn là phép nhân với 10h (hay 16 hệ 10). Vì là nhân với 16 nên địa chỉ cơ sở của đoạn bắt đầu ở ranh giới 16-byte và kết thúc ở vị trí 1048560 của bộ nhớ (16 byte nhỏ hơn 1 MB). Không gian nhớ vật lý được đánh địa chỉ trực tiếp trong 8086/8088 là 1 MB (2^{20}). Để tạo địa chỉ vật lý cho ngăn nhớ trong 8086/8088 cần phải kết hợp từng cặp: thanh ghi đoạn và một trong những thanh ghi IP, SP, SI, DI, BP, BX hoặc EA (địa chỉ hiệu dụng) theo nguyên tắc cho trong bảng (bảng 3.1). Bảng 3.2 giải thích mã hóa các trường của lệnh máy và cách tính EA. Địa chỉ hiệu dụng EA (Effective Address) của một ngăn nhớ được tính toán phụ thuộc vào các phương pháp đánh địa chỉ ghi trong cấu trúc lệnh máy. Các lệnh sử dụng cách đánh địa chỉ trực tiếp trong 8086/8088 có trường: MOD = 00, và R/M = 110.

Bảng 3.1: Nguyên tắc kết hợp thanh ghi theo cặp

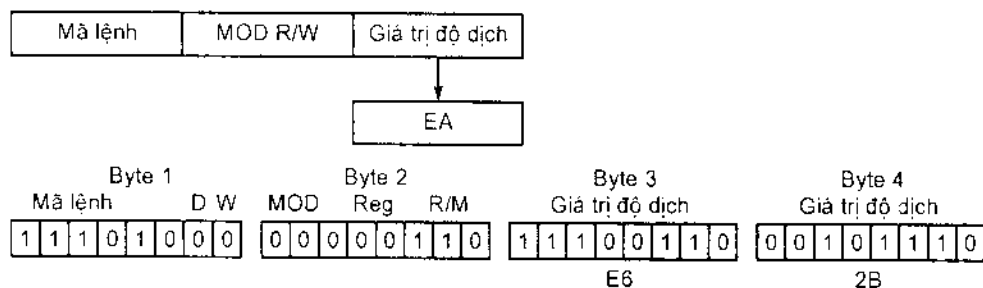
Kiểu tham chiếu bộ nhớ	Cơ sở đoạn mặc định	Cơ sở đoạn chọn lựa khác	Địa chỉ logic (Độ dịch)
Đọc lệnh	CS	NONE	IP
Ngăn xếp	SS	NONE	SP
Nguồn chuỗi dữ liệu	DS	CS, ES, SS	SI
Đích chuỗi dữ liệu	ES	NONE	DI
BP - Thanh ghi cơ sở	SS	CS, DS, ES	EA- địa chỉ hiệu dụng
Ghi/đọc dữ liệu	DS	CS, ES, SS	EA- địa chỉ hiệu dụng

Bảng 3.2: Mã hóa trường lệnh và cách tính EA trong vi xử lý Intel 16-bit

MOD=11			Tính địa chỉ EA			
REG	W=0	W=1	R/M	MOD=00	MOD=01	MOD=10
000	AL	AX	000	(BX)+(SI)	(BX)+(SI)+D8	(BX)+(SI)+D16
001	CL	CX	001	(BX)+(DI)	(BX)+(DI)+D8	(BX)+(DI)+D16
010	DL	DX	010	(BP)+(SI)	(BP)+(SI)+D8	(BP)+(SI)+D16
011	BL	BX	011	(BP)+(DI)	(BP)+(DI)+D8	(BP)+(DI)+D16
100	AH	SP	100	(SI)	(SI)+D8	(SI)+D16
101	CH	BP	101	(DI)	(DI)+D8	(DI)+D16
110	DH	SI	110	D8 (địa chỉ trực tiếp)	(BP)+D8	(BP)+D16
111	BH	DI	111	(BX)	(BX)+D8	(BX)+D16

Ví dụ đánh địa chỉ trực tiếp:

JMP 2BE6h; lệnh nhảy tới đoạn chương trình có địa chỉ logic 2BE6h. Nếu cho rằng thanh ghi đoạn lệnh CS chứa địa chỉ cơ sở đoạn 16-bit là: 3000h, thì địa chỉ tuyệt đối của lệnh phải nhảy tới là: $(CS) \times 10h + 2BE6h = 30000h + 2BE6h = 32BE6h$.



Hình 3.8: Cấu trúc lệnh JMP 2BE6h

Trong đó: Opcode của lệnh JMP có giá trị 111010. Các giá trị REG, D và W không có ý nghĩa (vì chỉ dùng cho MOD = 11), MOD = 00 chỉ ra rằng địa chỉ của ngăn nhớ được tính với giá trị độ dịch. Giá trị R/M = 110 chỉ rằng dùng địa chỉ trực tiếp. Trong đánh địa chỉ trực tiếp giá trị độ dịch là địa chỉ hiệu dụng (EA), hay là địa chỉ logic trong lệnh (hình 3.8).

c) Trong vi xử lý Intel 16-bit 80286 và 32-bit

Các vi xử lý Intel 32-bit có không gian địa chỉ vật lý là 4 GB (2^{32}). Từ 80286 đến Pentium đã có sự hỗ trợ đơn vị quản lý bộ nhớ MMU trong các chế độ hoạt động: chế độ thực (real mode) và chế độ bảo vệ (Protected Mode). Trong chế độ thực, tất cả các vi xử lý đánh địa chỉ trực tiếp bộ nhớ chỉ giới hạn là 1 MB, tương tự như trong 8086/8088. Chế độ này còn được gọi là chế độ 8086. Trong chế độ bảo vệ, các vi xử lý có thể đánh địa chỉ trực tiếp bộ nhớ dung lượng nhiều hơn: 16 MB (80286), và 4 GB (từ 80386 đến Pentium). Khi khởi động hệ thống vi xử lý 32-bit và 80286, đầu tiên các vi xử lý được đặt ở chế độ thực. Khi đó bit “cho phép bảo vệ” PE (Protection Enable bit) của từ trạng thái máy MSW bằng 0. Để đưa vào chế độ bảo vệ, phải đặt bit PE bằng 1. MSW có trong các vi xử lý từ 80286 đến Pentium. Vì các vi xử lý 32-bit có nhiều thanh ghi đoạn nên chúng cho phép đánh địa chỉ với nhiều đoạn hơn và dung lượng của mỗi đoạn nhớ không bị giới hạn bởi 64 kbyte như trong 8086/8088 mà có thể tới 1 MB hay 4 GB. Khác với đánh địa chỉ trong chế độ 8086, ở chế độ bảo vệ của các vi xử lý từ 80286 đến Pentium, địa chỉ trong thanh ghi đoạn, gọi là SELECTOR, trỏ tới một vùng nhớ được gọi là bảng mô tả đoạn SDT (Segment Descriptor Table) (GDT hay LDT). Nội dung của các trường ACCESS RIGHT, SEGMENT LIMIT, và SEGMENT BASE ADDRESS của mô tả đoạn được chọn trong bảng GDT hay SDT nạp vào thanh ghi mô tả đoạn tương ứng. Địa chỉ trực tiếp của ngăn nhớ là kết quả của phép cộng giá trị OFFSET có trong từ lệnh với địa chỉ cơ sở đoạn (SEGMENT BASE ADDRESS) trong thanh ghi mô tả đoạn (hình 2.106).

Bảng 3.3: Mã hóa các trường lệnh và cách tính EA trong vi xử lý Intel 32-bit

MOD = 11				Tính địa chỉ EA						
	W = 0		W = 1		MOD = 00		MOD = 01		MOD = 10	
REG	8-bit	16-bit	32-bit	R/M	16-bit	32-bit	16-bit	32-bit	16-bit	32-bit
000	AL	AX	EAX	000	(BX)+(SI) (EAX)		(BX)+(SI)+D8		(BX)+(SI)+D16	
001	CL	CX	ECX	001	(BX)+(DI) (ECX)		(BX)+(DI)+D8		(BX)+(DI)+D16	
010	DL	DX	EDX	010	(BP)+(SI) (EDX)		(BP)+(SI)+D8		(BP)+(SI)+D16	
011	BL	BX	EBX	011	(BP)+(DI) (EBX)		(BP)+(DI)+D8		(BP)+(DI)+D16	
100	AH	SP	ESP	100	(SI)	(SIB)	(SI)+D8 (SIB)+D8		(SI)+D16 (SIB)+D32	
101	CH	BP	EBP	101	(DI)	D32	(DI)+D8 (EBP)+D8		(DI)+D16 (EBP)+D32	
110	DH	SI	ESI	110	D16	ESI	(BP)+D8 (ESI)+D8		(BP)+D16 (ESI)+D32	
111	BH	DI	EDI	111	(BX)	EDI	(BX)+D8 (EDI)+D8		(BX)+D16 (EDI)+D32	

Bảng 3.3 giải thích mã hóa các trường của lệnh máy và cách tính EA trong các vi xử lý Intel 32-bit. Giá trị (SIB) có nghĩa là byte SIB đi theo sau byte ModR/M. Trong các vi xử lý Pentium có các chức năng MMX và XMM. Chúng có 8 thanh ghi cho lập trình, đó là: MMX0 - MMX7, và XMM0 - XMM7. Khi các thanh ghi này được dùng như các toán hạng thì chúng được chọn bằng trường REG (bit 5,4,3) và/hoặc trường R/M (bit 2, 1, 0) của byte ModR/M (bảng 3.4).

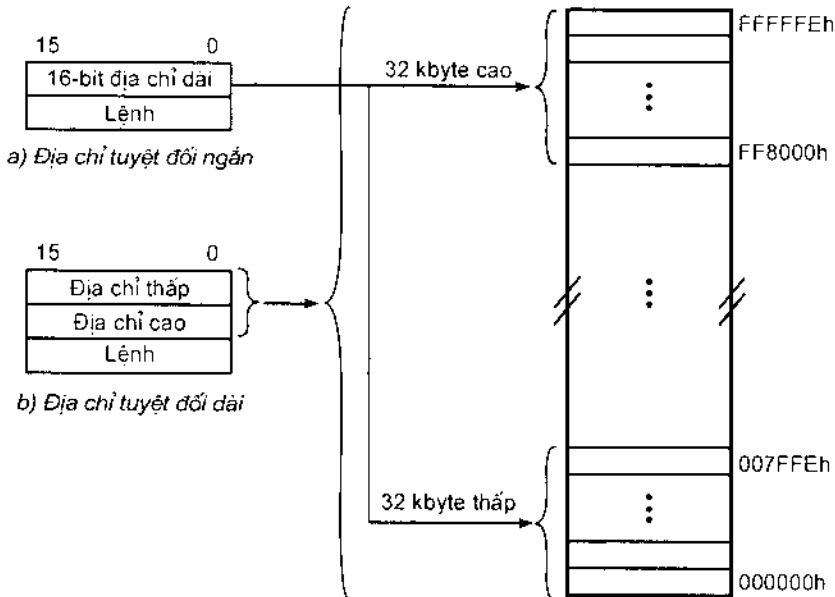
Bảng 3.4: Chọn các chức năng MMX và XMM bằng trường REG

REG	MMX	XMM
000	MMX0	XMM0
001	MMX1	XMM1
010	MMX2	XMM2
011	MMX3	XMM3
100	MMX4	XMM4
101	MMX5	XMM5
110	MMX6	XMM6
111	MMX7	XMM7

Nếu một lệnh MMX hay XMM có thao tác với các thanh ghi chung thì các thanh ghi chung được chọn nhờ trường R/M của byte ModR/M.

d) Trong họ vi xử lý Motorola M68000

Có 2 kiểu đánh địa chỉ tuyệt đối trong họ M68000: địa chỉ tuyệt đối ngắn (absolute short address) và địa chỉ tuyệt đối dài (absolute long address). Khi đó, trong từ lệnh giá trị trường Mode = 111. Đánh địa chỉ tuyệt đối ngắn sử dụng lệnh dài 2 từ 16-bit. Từ 16-bit thứ nhất là từ lệnh (instruction), từ thứ 2 là địa chỉ ngắn 16-bit, gọi là địa chỉ hiệu dụng (EA). Đánh địa chỉ tuyệt đối ngắn xác định các ngăn nhớ trong vùng từ 000000h đến 007FFEh và trong vùng từ FF8000h đến FFFFFFFeh. Đó là 2 vùng 32 kbyte thấp nhất và cao nhất của không gian nhớ của M68000. Địa chỉ tuyệt đối dài sử dụng các lệnh dài 3 từ 16-bit. Từ thứ 2 (16-bit cao) và thứ 3 (16-bit thấp) là địa chỉ hiệu dụng dài 32 bit (hình 3.9).



Hình 3.9: Đánh địa chỉ tuyệt đối ngắn và dài trong họ Motorola M68000

3.2.4. Đánh địa chỉ gián tiếp thanh ghi (register indirect)

Đánh địa chỉ gián tiếp thanh ghi hay thường gọi là đánh địa chỉ gián tiếp sử dụng các lệnh có độ dài 1 từ. Lệnh không chứa dữ liệu như trong đánh địa chỉ ngay lập tức, hoặc không chứa địa chỉ của dữ liệu

như trong đánh địa chỉ gián tiếp mà nó chỉ ra thanh ghi chứa địa chỉ của ngăn nhớ chứa dữ liệu. Đánh địa chỉ gián tiếp thanh ghi thuận tiện cho các lệnh vận chuyển dữ liệu từ hoặc tới các vùng nhớ liên tiếp nhau thường xuyên được truy cập tới, và các lệnh sử dụng kiểu đánh địa chỉ này ngắn hơn, dễ mã hóa và thời gian thực hiện nhanh.

a) Trong vi xử lý 8-bit (8080/8085/Z80)

Để chứa địa chỉ ngăn nhớ, các vi xử lý loại này phải dùng tới các cặp thanh ghi như: BC, DE, HL, và có thể con trỏ ngăn xếp SP. Ví dụ lệnh của 8085:

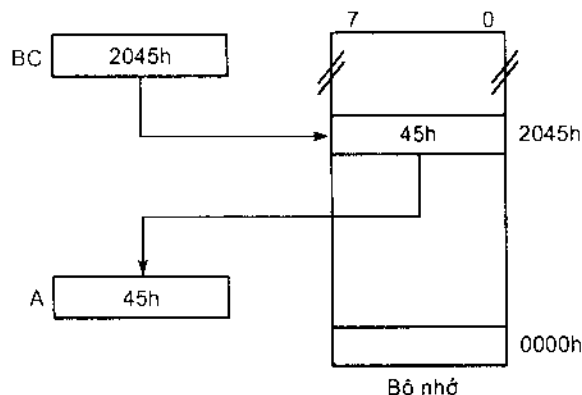
LDAX rp; (A) $\leftarrow$ ((rp)): nạp dữ liệu vào thanh ghi tích lũy A từ ngăn nhớ có địa chỉ ghi trong một cặp thanh ghi rp (BC, DE, HL).

LDAX B; (A) $\leftarrow$ ((B)(C)): nạp dữ liệu vào thanh ghi tích lũy A từ ngăn nhớ có địa chỉ ghi trong cặp thanh ghi BC.

0	0	0	0	1	0	1	0	0Ah
---	---	---	---	---	---	---	---	-----

Các cặp thanh ghi trong 8085 có các số hiệu tương ứng là: 00 - BC, 01 - DE, 10 - HL, và 11 - SP. Giả sử BC = 2045h, ngăn nhớ 2045h chứa giá trị 45h, khi đó sơ đồ diễn giải lệnh thực hiện như mô tả trong hình 3.10.

Tương tự như trong Z80: LD A, (HL)

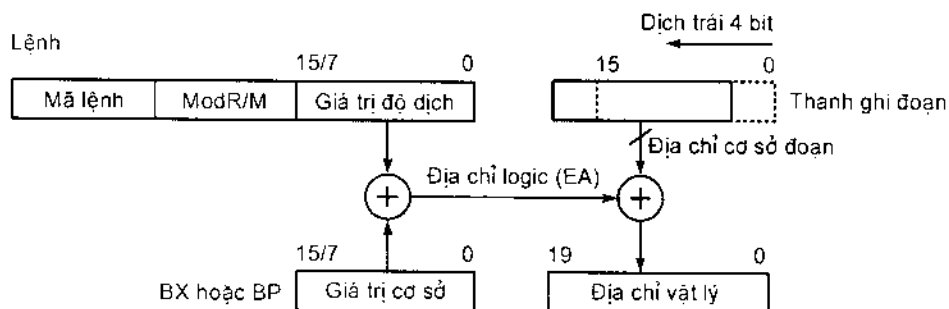


Hình 3.10: Diễn giải thực hiện lệnh LDAX B sử dụng đánh địa chỉ gián tiếp thanh ghi

b) Trong vi xử lý Intel 16-bit (8086/8088/80286)

Đánh địa chỉ gián tiếp thanh ghi trong các vi xử lý Intel 16-bit sử dụng các thanh ghi cơ sở (base register) BP, BX cùng với các thanh ghi đoạn (bảng 3.1 và 3.2). Giá trị độ dịch (displacement, hay offset) chứa ngay trong lệnh (từ thứ 2 của lệnh), còn giá trị cơ sở (base) ở trong các thanh ghi cơ sở (BP, BX). Trường MOD R/M của toán hạng (BYTE2 của lệnh) gián tiếp xác định thanh ghi cơ sở.

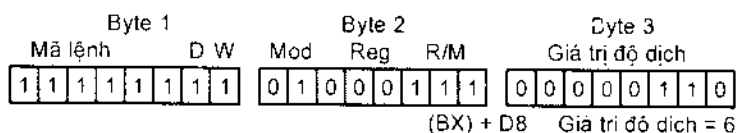
Căn cứ bảng 3.2, các lệnh sử dụng chế độ đánh địa chỉ gián tiếp thanh ghi có thể có các trường: MOD = 00, 01, 10 và R/M = 000, 001, 010, 011, 110, 111 với các giá trị độ dịch (displacement) 8-bit hoặc 16-bit (là từ thứ 2 trong lệnh). Việc tạo địa chỉ vật lý bằng cách đánh địa chỉ gián tiếp thanh ghi cơ sở được mô tả trong hình 3.11.



Hình 3.11: Đánh địa chỉ gián tiếp thanh ghi cơ sở trong Intel 16-bit

Ví dụ, lệnh: INC [BX+6]; tăng nội dung ngăn nhớ mà địa chỉ logic (hay EA) của nó là kết quả của (BX) + 6.

Cho rằng thanh ghi đoạn lệnh CS chứa địa chỉ cơ sở đoạn là: 3000h, thanh ghi BX chứa giá trị 2BE0h, khi đó lệnh tăng nội dung ngăn nhớ có địa chỉ là 32BE6h lên 1 có khuôn dạng là:

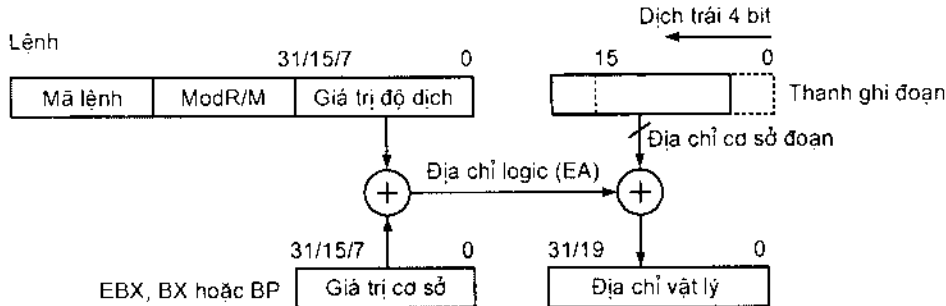


Trong đó: Opcode của lệnh INC có giá trị 111111. Lệnh tăng một từ ($W = 1$), Theo bảng 3.2, MOD = 01 chỉ ra rằng địa chỉ của ngăn nhớ được tính bằng (BX)+D8 nội dung của BX cộng với giá trị độ dịch 8-bit (6). Giá trị R/M = 111 chỉ rằng dùng (BX). Giá trị REG = 000 không có ý nghĩa ở đây. Theo dạng lệnh này ta có giá trị địa chỉ vật lý của ngăn nhớ như sau:

$$(CS) \times 4 + (BX) + D8 = 30000h + 2BE0h + 6 = 32BE6h.$$

c) Trong vi xử lý Intel 32-bit (từ 80386 đến Pentium)

Khi các vi xử lý 32-bit làm việc trong chế độ 8086 thì cách đánh địa chỉ gián tiếp thanh ghi cơ sở cũng tương tự như trong các vi xử lý Intel 16-bit, chỉ khác là giá trị độ dịch có thể có các độ dài 8/16/32-bit (hình 3.12).



Hình 3.12: Đánh địa chỉ gián tiếp thanh ghi cơ sở trong Intel 32-bit

Ví dụ các lệnh:

DISPL DW 0120h; định nghĩa giá trị độ dịch = 0120h;

MOV ECX, DISPL[EBX]; $ECX \leftarrow M[DS:EBX+0120h]$, đọc dữ liệu 32-bit của ngăn nhớ ở địa chỉ vật lý DS: (EBX) + 0120h vào ECX.

MOV AX, [BX]; $AX \leftarrow M[DS:BX]$, chuyển dữ liệu 16-bit vào AX từ ngăn nhớ có địa chỉ vật lý 20-bit trong DS:BX.

MOV [BP], DL; $M[SS:BP] \leftarrow (DL)$.

d) Trong họ vi xử lý Motorola M68000

Có 4 kiểu đánh địa chỉ gián tiếp thanh ghi trong họ Motorola M68000, đó là:

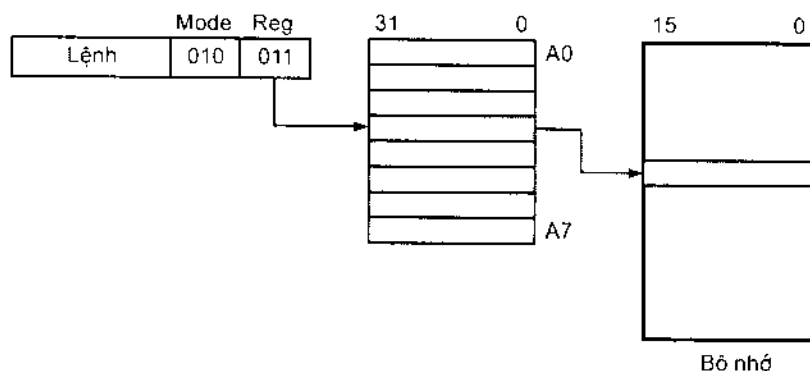
Gián tiếp thanh ghi địa chỉ: Mode = 010, (An)

Gián tiếp thanh ghi địa chỉ với sự tăng sau: Mode = 011, (An)+

Gián tiếp thanh ghi địa chỉ với sự giảm trước: Mode = 100, -(An)

Gián tiếp thanh ghi địa chỉ với displacement 16-bit: Mode = 101, (d16,An).

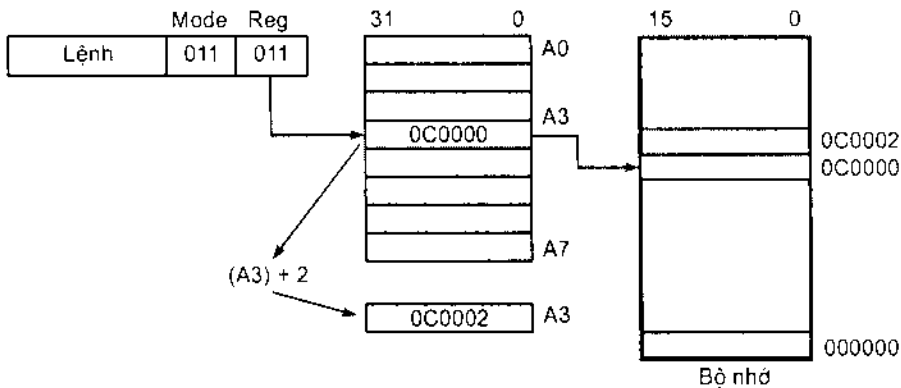
Phương pháp đánh địa chỉ gián tiếp thanh ghi được sử dụng khi muốn thanh ghi địa chỉ An trở tới vùng nhớ cần truy cập. Khi đó trong từ lệnh trường Register phải ghi số hiệu của một trong 8 thanh ghi địa chỉ (A0 - A7) chứa địa chỉ 32-bit của ngăn nhớ và trường Mode = 010 (chú ý rằng các thanh ghi địa chỉ và dữ liệu trong họ Motorola M68000 đều có độ dài 32-bit) (hình 3.14). Như vậy với chế độ đánh địa chỉ gián tiếp thanh ghi địa chỉ có thể đánh địa chỉ trực tiếp một không gian nhớ là 4 GB (2^{32}).



Hình 3.13: Đánh địa chỉ gián tiếp thanh ghi địa chỉ trong họ Motorola M68000

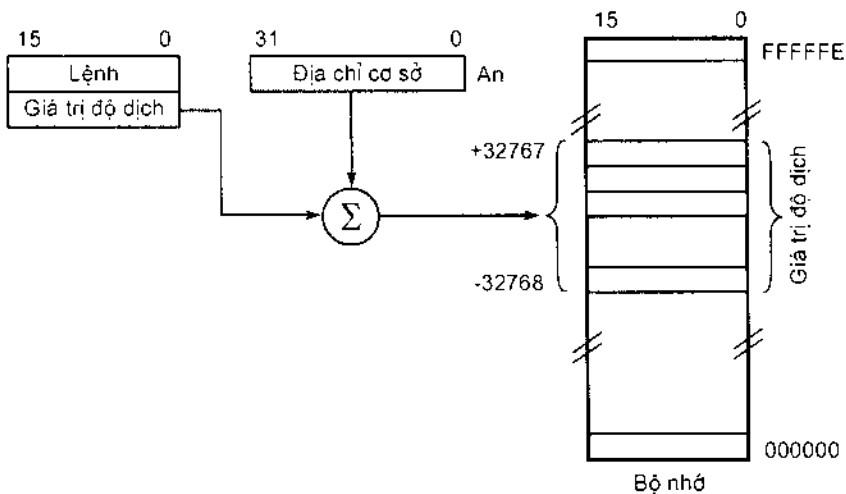
Phương pháp địa chỉ gián tiếp thanh ghi địa chỉ với tăng sau: Mode = 011, (An)+ cho phép sử dụng bất kỳ thanh ghi địa chỉ nào như một con trỏ ngăn xếp. Sau khi địa chỉ ngăn nhớ và thao tác dữ liệu của lệnh hoàn thành, thì giá trị ghi trong thanh ghi địa chỉ được tăng lên

sao cho nó trở đến vị trí tiếp sau của ngăn xếp. Giá trị trong thanh ghi địa chỉ được tăng lên 1, 2, hoặc 4 tùy thuộc vào kích thước dữ liệu xác định trong từ lệnh.



Hình 3.14: Đánh địa chỉ gián tiếp thanh ghi địa chỉ với tăng sau trong họ Motorola M68000

Hình 3.14 mô tả lệnh truy cập ngăn nhớ có địa chỉ vật lý là 0C0000h ghi trong thanh ghi địa chỉ A3, nhưng với Mode = 011 thì sau đó nội dung thanh ghi A3 được lên 2. Kết quả $(A3) + 2 = 0C0002h$.



Hình 3.15: Đánh địa chỉ gián tiếp thanh ghi địa chỉ với giảm trước trong họ Motorola M68000. Khoảng địa chỉ từ $An - 32768$ đến $An + 32767$

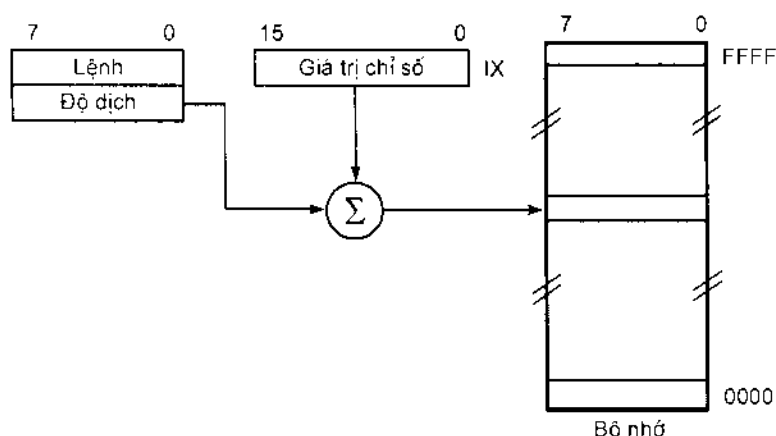
Trong phương pháp địa chỉ gián tiếp thanh ghi địa chỉ với sự giảm trước: Mode = 101, $-(A_n)$, giá trị trong thanh ghi địa chỉ được xem như là địa chỉ cơ sở để tính địa chỉ vật lý của ngăn nhớ. Từ thứ 2 trong lệnh là giá trị độ dịch 16-bit. Để thực hiện $-(A_n)$, độ dịch được cộng ở mã bù 2 vào giá trị địa chỉ cơ sở 32-bit. Như vậy khoảng đánh địa chỉ là từ -32768 đến +32767 byte tương đối với các giá trị cơ sở ghi trong thanh ghi địa chỉ, tức là từ $A_n - 32768$ đến $A_n + 32767$. Hình 3.15 mô tả địa chỉ vật lý của ngăn nhớ được tính như thế nào bằng cách đánh địa chỉ gián tiếp thanh ghi địa chỉ với sự giảm trước.

3.2.5. Đánh địa chỉ gián tiếp thanh ghi chỉ số

Đánh địa chỉ gián tiếp thanh ghi chỉ số (index register indirect), hay thường gọi là đánh địa chỉ chỉ số được thực hiện nhờ có các thanh ghi chỉ số có trong vi xử lý.

a) Trong vi xử lý 8-bit

Một số vi xử lý 8-bit có các thanh ghi chỉ số 16-bit để dùng cho đánh địa chỉ chỉ số, như IX, IY của Z80. Nội dung của thanh ghi chỉ số được cộng với giá trị byte thứ 2 trong từ lệnh sử dụng đánh địa chỉ chỉ số để tạo ra địa chỉ của ngăn nhớ. Byte thứ 2 của từ lệnh là giá trị lệch thêm (Offset), nó có giá trị từ 00h đến FFh, tức là lệch so với giá trị chỉ số từ 0 đến 255, nghĩa là phương pháp đánh địa chỉ chỉ số cho phép chỉ tới 256 ngăn nhớ mà không cần thay đổi nội dung thanh ghi chỉ số. Ví dụ: ADD A, (IX + 15), thực hiện cộng các nội dung của thanh tích lũy và một ngăn nhớ có địa chỉ là kết quả của phép cộng nội dung thanh ghi chỉ số IX với giá trị Offset là 15. Kết quả phép cộng chuyển lại vào thanh ghi tích lũy. Từ lệnh này dài 2 byte, trong đó byte thứ nhất là mã lệnh và byte thứ 2 là độ dịch (Offset). Lệnh thực hiện trong 5 chu kỳ máy. Nói chung các lệnh sử dụng đánh địa chỉ chỉ số có thể kéo dài thời gian thực hiện so với cách đánh địa chỉ gián tiếp thanh ghi (cơ sở). Hình 3.16 là tạo địa chỉ ngăn nhớ theo cách đánh địa chỉ chỉ số.



Hình 3.16: Đánh địa chỉ gián tiếp thanh ghi chỉ số trong vi xử lý 8-bit (Z80)

b) Trong vi xử lý Intel 16-bit (8086/8088/80286)

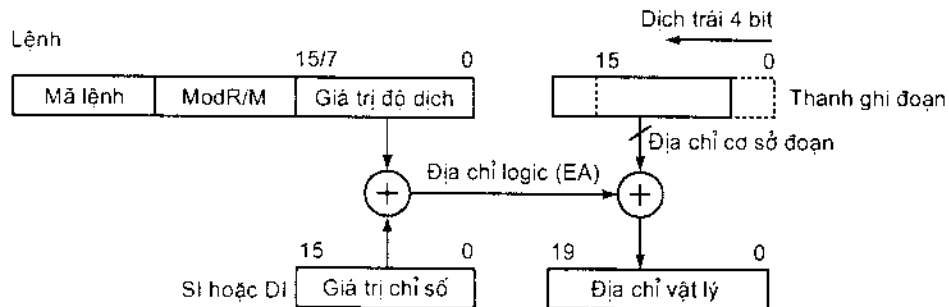
Để đánh địa chỉ gián tiếp chỉ số các vi xử lý Intel 16-bit sử dụng các thanh ghi chỉ số SI, DI, các thanh ghi cơ sở BP, BX và các độ dịch 8-bit (D8) hoặc 16-bit (D16) (bảng 3.2). Các trường R/M và MOD = 00, 01, 10 kết hợp chọn chế độ đánh địa chỉ và tính giá trị EA (bảng 3.2). Có các chế độ đánh địa chỉ gián tiếp chỉ số như sau: chỉ số, chỉ số + độ dịch, cơ sở + chỉ số, cơ sở + chỉ số + độ dịch (hình 3.17). Sở dĩ có nhiều kiểu đánh địa chỉ như vậy trong 8086/8088 vì mục đích của các loại vi xử lý này là làm cho việc dịch các chương trình viết ở ngôn ngữ bậc cao thành các lệnh máy đạt hiệu quả cao hơn. Ví dụ: nếu viết câu lệnh gán giá trị cho mảng ở ngôn ngữ bậc cao là $C(j) = X$ thì sẽ thuận tiện hơn nếu sử dụng kiểu đánh địa chỉ “thanh ghi chỉ số+độ dịch”. Khi đó, chỉ số trở tới địa chỉ đầu của mảng, và độ dịch trở tới từng thành phần trong mảng.

Ví dụ:

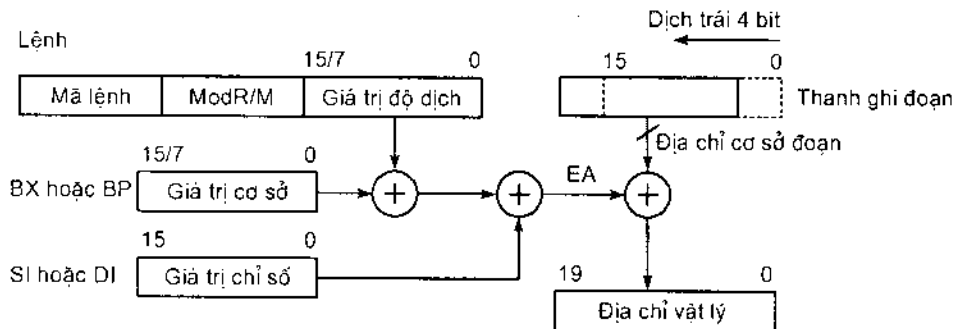
MOV SI, Offset databuf; nạp địa chỉ đầu của vùng nhớ chứa dữ liệu vào SI

MOV AL,[SI]; đọc nội dung ngăn nhớ ở địa chỉ EA = (SI) vào thanh ghi AL.

MOV [SI+BX], AL; nội dung AL vào ngăn nhớ ở địa chỉ EA = (SI) + (BX).



a) Gián tiếp thanh ghi chỉ số



b) Gián tiếp thanh ghi cơ sở/chỉ số

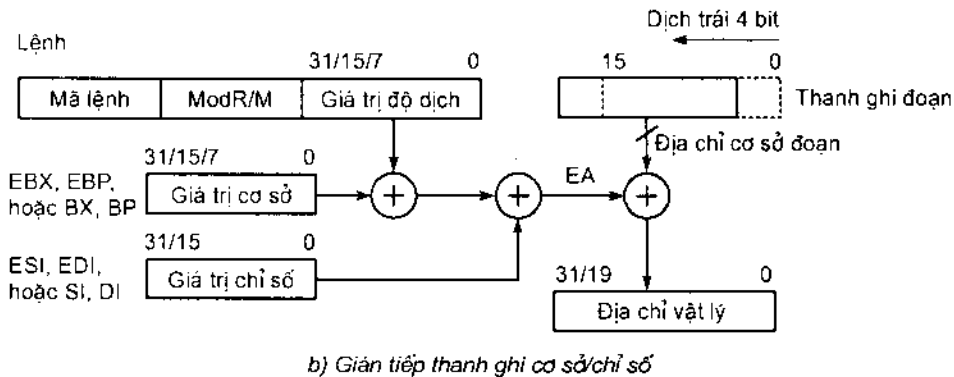
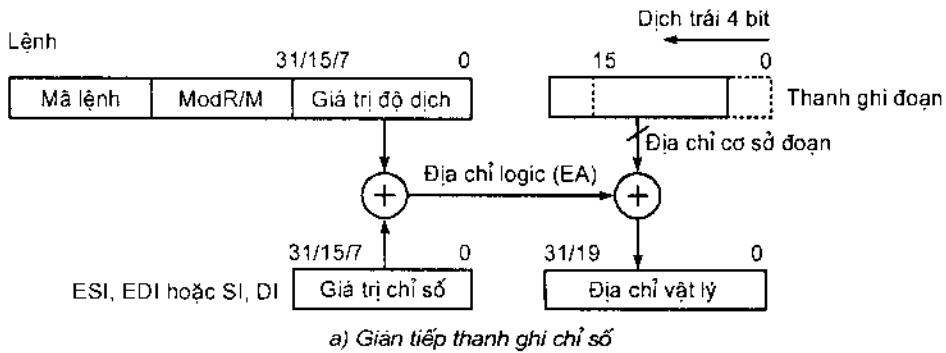
Hình 3.17: Đánh địa chỉ chỉ số trong Intel 16-bit

c) Trong vi xử lý Intel 32-bit (từ 80386 đến Pentium)

Khi các vi xử lý 32-bit làm việc trong chế độ 8086 thì cách đánh địa chỉ gián tiếp thanh ghi chỉ số cũng tương tự như trong các vi xử lý Intel 16-bit, chỉ khác là giá trị độ dịch có thể có các độ dài 8/16/32-bit. Các thanh ghi chung đều có thể được chọn làm thanh ghi chỉ số hoặc thanh ghi cơ sở. Bảng 3.6 giải thích sự chọn lựa bằng các trường REG và/hoặc R/M trong từ lệnh.

Bảng 3.6: Các trường REG và/hoặc R/M trong từ lệnh

REG	INDEX	BASE
000	EAX	EAX
001	ECX	ECX
010	EDX	EDX
011	EBX	EBX
100	Không dùng	ESP
101	EBP	Nếu MOD = 00, D32 Nếu MOD ≠ 00, EBP
110	ESI	ESI
111	EDI	EDI



Hình 3.18: Đánh địa chỉ chỉ số chỉ số trong Intel 32-bit

Hình 3.19 là ví dụ mô tả phương pháp đánh địa chỉ gián tiếp thanh ghi chỉ số, trong đó, các thanh ghi SI, DI, hoặc ESI, EDI được làm thành ghi chỉ số. Các thanh ghi BP, BX, hoặc EBP, EBX được

dùng làm thanh ghi cơ sở. Có sự nâng cấp so với các vi xử lý Intel 16-bit là các vi xử lý Intel 32-bit có bổ sung các chế độ đánh địa chỉ gián tiếp thanh ghi chỉ số với giá trị cấp độ (SCALE) nhờ trường Scale 2 bit trong byte SIB (Scale, Index, Base).

Bảng 3.4 tổng hợp cách tính địa chỉ EA ở các chế độ đánh địa chỉ nhờ mã hóa các byte ModR/M và SIB, trong đó có các chế độ đánh địa chỉ gián tiếp với chỉ số, và chỉ số có thể được nhân với cấp độ.

Ví dụ:

MOV EAX, [ESI + EBX]; nội dung của ngăn nhớ ở địa chỉ EA = (ESI) + (EBX) chuyển vào thanh ghi EAX.

MOV EAX, [EBP + EDI*4 + FFF0h]; nội dung của ngăn nhớ ở địa chỉ EA = (EBP) + (EDI)*4 + FFF0h chuyển vào thanh ghi EAX.

DISPL DW 0120h; định nghĩa giá trị độ dịch = 0120h;

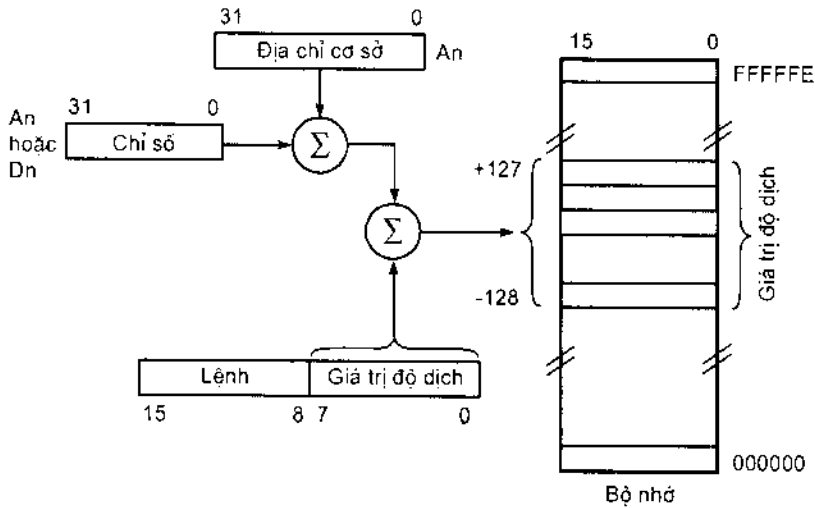
MOV EAX, DISPL[EBX].

d) Trong họ vi xử lý Motorola M68000

Có 2 chế độ đánh địa chỉ gián tiếp chỉ số, đó là: gián tiếp thanh ghi địa chỉ với chỉ số (displacement 8-bit): Mode = 110, (d8, An, Xn), và gián tiếp chỉ số: Mode = 110, (bd, An, Xn) thực hiện tương tự như trường hợp trên, chỉ khác là giá trị độ dịch cơ sở (base displacement) có độ dài 32-bit. Nó là từ thứ 2 của lệnh.

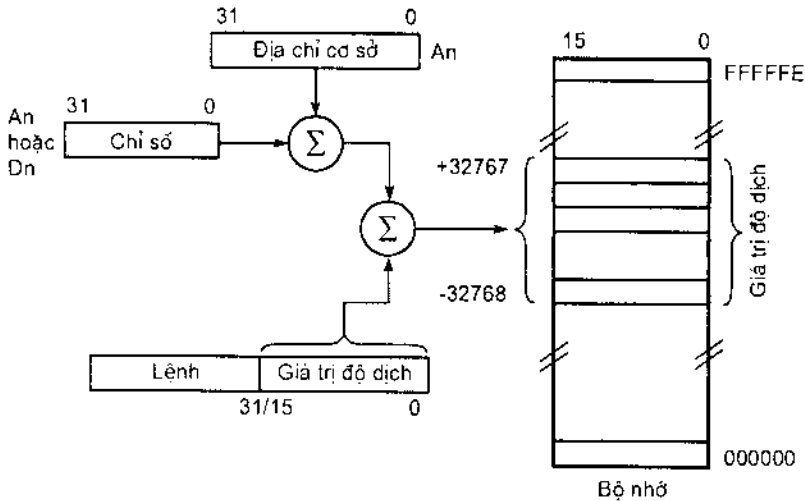
Đánh địa chỉ gián tiếp chỉ số: Mode = 110, (d8, An, Xn) thực hiện cộng 3 giá trị d8 (displacement 8-bit): nội dung 32-bit của thanh ghi địa chỉ An và nội dung của thanh ghi chỉ số Xn, tạo địa chỉ hiệu dụng, tức là $EA = (An) + (Xn) + d8$. Displacement 8-bit là byte thứ 2 (byte thấp) của từ lệnh (hình 3.19).

Chế độ đánh địa chỉ gián tiếp thanh ghi địa chỉ với chỉ số (base displacement): Mode = 110, (bd, An, Xn) tạo $EA = (An) + (Xn) + db$. Trong đó, bd là độ dịch cơ sở (base displacement) ở mã bù 2 độ dài 16-bit hoặc 32-bit (hình 3.20).



Hình 3.19: Đánh địa chỉ gián tiếp thành ghi chỉ số:
Mode = 110, (d8, An, Xn)

Bất kỳ thanh ghi nào trong vi xử lý M68000 đều có thể được sử dụng như là thanh ghi chỉ số (An hoặc Dn).



Hình 3.20: Đánh địa chỉ gián tiếp thành ghi chỉ số:
Mode = 110, (bd, An, Xn)

Những lệnh sử dụng phương pháp đánh địa chỉ gián tiếp thanh ghi chỉ số thuận tiện cho đánh địa chỉ các mảng dữ liệu 2 chiều vì có 2 độ dịch (offset, displacement) tương đối so với địa chỉ cơ sở ghi trong thanh ghi An. Khoảng địa chỉ theo chỉ số là từ -32768 đến +32767 và khoảng địa chỉ theo offset là từ -128 đến +127 tương đối với các giá trị cơ sở ghi trong thanh ghi địa chỉ An. Giá trị chỉ số có thể được nhân với một cấp độ (scale value) trước khi cộng với (An) và d8 hoặc bd. Khi đó giá trị chỉ số có thể là: $X_n \times \text{SCALE}$.

3.2.6. Đánh địa chỉ gián tiếp bộ nhớ (memory indirect)

Các bộ vi xử Motorola 68020, 68030, 68040 có phương pháp đánh địa chỉ gián tiếp bộ nhớ như sau:

Gián tiếp bộ nhớ với chỉ số sau: Mode = 110, ([bd, An], X_n , od)

Gián tiếp bộ nhớ với chỉ số trước: Mode = 110, ([bd, An, X_n], od).

Trong chế độ đánh địa chỉ gián tiếp bộ nhớ với chỉ số sau cả toán hạng và địa chỉ của nó đều nằm trong bộ nhớ. Khi đó địa chỉ hiệu dụng của ngăn nhớ chứa toán hạng được tính là:

$$EA = (An + bd) + X_n \times \text{SCALE} + od;$$

Trong đó: od là độ dịch ngoài xa (outer displacement) 32-bit, bd: độ dịch cơ sở 32-bit, X_n : chỉ số 32-bit. Giá trị $(An + bd)$ là địa chỉ trung gian trở đến ngăn nhớ chứa địa chỉ gián tiếp chỉ số của toán hạng, tức là nội dung này phải được cộng với giá trị chỉ số $X_n \times \text{SCALE}$ và sau đó cộng với giá trị od 32-bit mới tạo được địa chỉ EA của toán hạng.

Trong chế độ đánh địa chỉ gián tiếp bộ nhớ với chỉ số trước địa chỉ hiệu dụng của ngăn nhớ chứa toán hạng được tính là:

$$EA = (An + bd + X_n \times \text{SCALE}) + od;$$

Trong đó, giá trị $(An + bd + X_n \times \text{SCALE})$ là địa chỉ trung gian trở tới ngăn nhớ chứa địa chỉ gián tiếp chỉ số của toán hạng.

3.2.7. Địa chỉ tương đối với thanh đếm chương trình (PC relative)

Đánh địa chỉ tương đối rất giống với đánh địa chỉ chỉ số, nhưng có một số khác nhau. Trước hết, trong chế độ đánh địa chỉ chỉ số, độ dịch (displacement, offset) được cộng với nội dung của một thanh ghi, còn trong chế độ đánh địa chỉ tương đối, giá trị độ dịch là tương đối với giá trị nội dung hiện thời của thanh đếm chương trình PC. Thứ hai, đánh địa chỉ tương đối thường sử dụng phép số học ở mã bù 2 (tức là giá trị độ dịch ở mã bù 2) để có thể khả thi đối với các lệnh rẽ nhánh tiến khi độ dịch có giá trị bit cao nhất bằng 0 logic, hoặc rẽ nhánh lùi khi độ dịch có giá trị bit cao nhất bằng 1. Một số lệnh rẽ nhánh có điều kiện trong một số loại vi xử lý sử dụng chế độ đánh địa chỉ tương đối. Nó cho phép ghi mã độc lập với vị trí (position-independent code), hay còn gọi là mã tương đối.

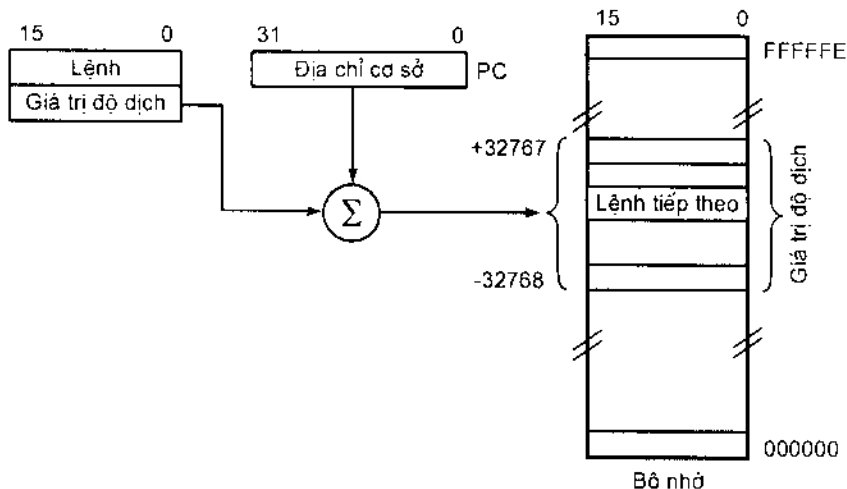
Thanh đếm chương trình (PC) chứa địa chỉ lệnh. Sau khi lệnh được đọc ra, và được thực hiện thì nội dung của PC được tăng lên để chỉ tới lệnh tiếp theo (nếu lệnh có độ dài 4 byte, thì nội dung của PC được tăng lên 4). Đây được gọi là PC được cập nhật, và nội dung của PC cập nhật được sử dụng trong chế độ gián tiếp thanh đếm chương trình (PC relative mode). Chế độ địa chỉ tương đối với thanh đếm chương trình được sử dụng với các lệnh điều khiển chương trình trong nhiều loại vi xử lý. Một chương trình được viết ở địa chỉ tương đối có thể được sắp xếp vào bất cứ vùng nhớ nào. Cách gọi đánh địa chỉ tương đối chủ yếu dùng trong các loại vi xử lý 8-bit như 8080/8085/Z80, mà ở chúng không có cơ chế phân đoạn bộ nhớ và khái niệm đánh địa chỉ tương đối là đúng nghĩa tương đối với thanh đếm chương trình. Trong các loại vi xử lý Intel từ 16-bit đến 32-bit, thay thế cho PC là con trỏ lệnh IP (Instruction Pointer), và IP cũng chứa địa chỉ tương đối của lệnh tiếp theo. Tuy nhiên thanh ghi đoạn mã CS lại chứa địa chỉ cơ sở và nội dung IP là tương đối với địa chỉ cơ sở đoạn CS. Địa chỉ vật lý của lệnh bằng giá trị địa chỉ cơ sở trong CS

(Code Segment register) dịch trái 4 bit rồi cộng với nội dung IP. Do vậy thay cho địa chỉ tương đối các loại vi xử lý này sử dụng các chế độ đánh địa chỉ gián tiếp thanh ghi như đã trình bày.

Các loại vi xử lý họ Motorola thì có các chế độ đánh địa chỉ gián tiếp thanh đếm chương trình với độ dịch hay với chỉ số.

3.2.8. Đánh địa chỉ gián tiếp thanh đếm chương trình với độ dịch

Phương pháp đánh địa chỉ gián tiếp thanh đếm chương trình với độ dịch (PC indirect with displacement) có trong họ M68000 với Mode = 111, (d16, PC). Địa chỉ của lệnh tiếp theo là tổng của địa chỉ 32-bit chứa trong thanh đếm lệnh (PC) và độ dịch 16-bit (d16). Các lệnh sử dụng kiểu đánh địa chỉ này có độ dài 2 từ. Từ thứ hai của lệnh là giá trị độ dịch 16-bit. Như vậy địa chỉ EA của các lệnh nằm trong khoảng từ -32768 đến +32767 byte tương đối so với giá trị cơ sở chứa trong PC (hình 3.21) và là: $EA = (PC) + d16$.



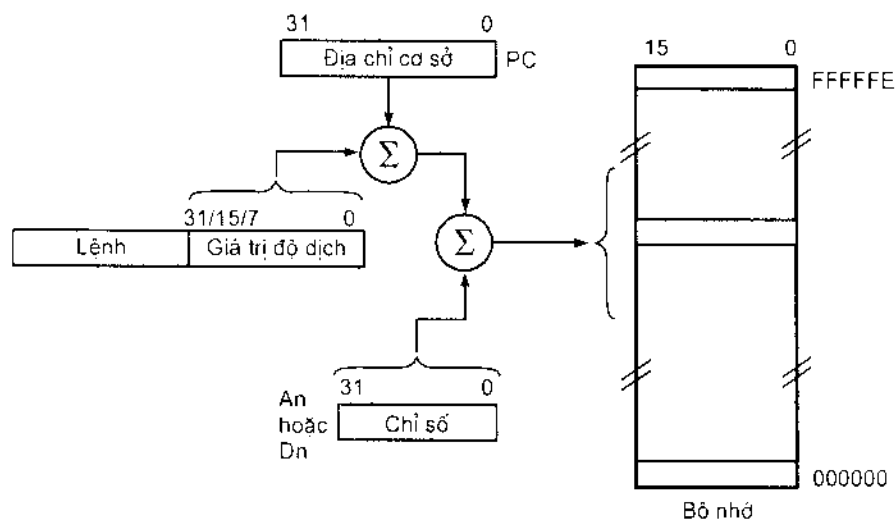
Hình 3.21: Đánh địa chỉ gián tiếp thanh đếm chương trình với độ dịch 16-bit trong họ Motorola M68000. Khoảng địa chỉ từ $An - 32768$ đến $An + 32767$

3.2.9. Đánh địa chỉ gián tiếp thanh đếm chương trình với chỉ số

Có 2 phương pháp đánh địa chỉ gián tiếp chương trình với chỉ số (PC indirect with index) trong họ M68000 (hình 3.22), đó là:

- Gián tiếp thanh đếm chương trình với chỉ số (8-bit displacement) có trong tất cả các vi xử lý M68000: Mode = 111, (d8, PC, Xn) và $EA = (PC) + (Xn) + d8$.

- Gián tiếp thanh đếm chương trình với chỉ số (base displacement) có trong tất cả các vi xử lý M68020, M68030, M68040: Mode = 111, (bd, PC, Xn) và $EA = (PC) + (Xn) + bd$. Giá trị chỉ số Xn có thể nhân với SCALE: $Xn \times SCALE$.



Hình 3.22: Đánh địa chỉ gián tiếp thanh đếm chương trình với chỉ số trong họ Motorola M68000

3.2.10. Đánh địa chỉ gián tiếp thanh đếm bộ nhớ thanh đếm chương trình

Có 2 dạng đánh địa chỉ gián tiếp thanh đếm bộ nhớ thanh đếm chương trình (PC memory indirect), đó là: đánh địa chỉ gián tiếp bộ nhớ thanh đếm chương trình với chỉ số sau với Mode = 111, ([bd, PC], Xn, od); và đánh địa chỉ gián tiếp bộ nhớ thanh đếm chương trình với chỉ số trước với Mode = 111, ([bd, PC, Xn], od). Cả toán hạng và địa chỉ toán hạng đều nằm trong bộ nhớ. Những phương pháp này tương tự như các phương pháp đánh địa chỉ gián tiếp bộ nhớ (mục 3.2.6) nhưng chỉ có trong các vi xử lý M68020, M68030, M68040. Với đánh địa chỉ

gián tiếp bộ nhớ thanh đếm chương trình với chỉ số sau địa chỉ EA được tính bằng:

$$EA = (bd + PC) + X_n \times SCALE + od$$

Trong đó, địa chỉ trung gian ($bd + PC$) trở tới ngăn nhớ chứa giá trị mà giá trị này phải được cộng với chỉ số $X_n \times SCALE$ và od để tạo ra địa chỉ toán hạng. Với đánh địa chỉ gián tiếp bộ nhớ thanh đếm chương trình với chỉ số trước thì địa chỉ EA được tính bằng:

$$EA = (bd + PC + X_n \times SCALE) + od$$

Trong đó, địa chỉ trung gian ($bd + PC + X_n \times SCALE$) trở tới ngăn nhớ chứa giá trị mà giá trị này phải được cộng với od để tạo ra địa chỉ toán hạng.

3.3. PHÂN LOẠI TẬP LỆNH CỦA VI XỬ LÝ

3.3.1. Tập lệnh phức tạp và tập lệnh giảm thiểu (CISC và RISC)

Các loại vi xử lý Intel x86 và họ Motorola M68000 đều có tập lệnh lớn và phức tạp. Chúng là các loại vi xử lý kiến trúc CISC. Tập lệnh lớn của các vi xử lý kiến trúc CISC làm cho các chương trình dịch ngôn ngữ bậc cao cũng có nhiều phương án hơn và khó chọn ra một phương án tối ưu. Kiến trúc đường ống trong các vi xử lý CISC cũng chỉ phát huy hiệu quả nếu các lệnh có kích thước và thời gian thực hiện tương đối giống nhau.

Bên cạnh đó, có các vi xử lý có tập lệnh tối thiểu (RISC). Tối thiểu về số lượng lệnh trong tập lệnh, về số lượng các chế độ đánh địa chỉ, về khuôn dạng lệnh, và về thời gian thực hiện lệnh. Đây là ưu điểm của RISC so với CISC. Kiến trúc đường ống là một kỹ thuật sử dụng rộng rãi trong các hệ thống RISC và sẽ phát huy hiệu quả nếu như đạt được sự đồng nhất về kích thước lệnh, thời gian thực hiện lệnh trong một chu kỳ. Trong một vi xử lý RISC lý tưởng tất cả các lệnh đều có độ dài giống nhau (thường là 32-bit) và được thực hiện chỉ trong một chu kỳ của CPU. Trong thực tế, chỉ có trên 80% các lệnh

trong các vi xử lý kiến trúc RISC (Reduced Instruction Set Computer) thực hiện được chung một chu kỳ CPU.

Ngoài những ưu điểm trên so với CISC, RISC có những nhược điểm. Trong RISC phải viết nhiều mã lệnh hơn để thực hiện một thao tác tương tự trong CISC. Trong khi đó CISC có tập lệnh phong phú, và một lệnh thực hiện trong CISC bằng 2, 3 hoặc nhiều hơn lệnh phải thực hiện trong RISC. Nhiều lệnh hơn đồng nghĩa với việc phải dành bộ nhớ nhiều hơn để chứa chương trình và dữ liệu trong RISC, điều này kéo theo sự tăng thời gian truy cập bộ nhớ của CPU. Để thực hiện một chức năng tương tự nhau, một chương trình chạy trên RISC dài hơn 30% so với chương trình chạy trên CISC. Tốc độ thực hiện lệnh bị giảm đi nếu có tham chiếu đến bộ nhớ bên ngoài vi xử lý. Vì vậy tăng số lệnh có thao tác thanh ghi-thanh ghi bên trong vi xử lý là một trong những biện pháp nâng cao hiệu suất xử lý lệnh. Các vi xử lý tiên tiến hiện nay được trang bị nhiều thanh ghi, và do vậy người ta gọi là tập các thanh ghi, đặc biệt là trong các vi xử lý RISC.

3.3.2. Phân loại tập lệnh của bộ vi xử lý

Đối với đa số các loại vi xử lý từ 8-bit đến tiên tiến hiện nay có thể phân tập lệnh thành 8, 9, hoặc 10 nhóm lệnh khác nhau. Một số nhà sản xuất các chip vi xử lý có thể có sự phân loại khác, nên sự phân loại tập lệnh chỉ mang tính tương đối.

Với các loại vi xử lý kiến trúc CISC tiên tiến hiện nay có thể phân tập lệnh của chúng thành 8 nhóm lệnh cơ bản như sau:

- + Chuyển dữ liệu (Data transfer)
- + Số học nguyên và logic (Arithmetic and logic)
- + Dịch và quay vòng (Logic, shift, rotate)
- + Chuyển điều khiển (Control transfer)
- + Xử lý bit (Bit manipulation)
- + Điều khiển hệ thống (System control)

- + Dấu phẩy động (Floating-point)
- + Các lệnh của các đơn vị chức năng đặc biệt (Special function unit).

Các loại vi xử lý thế hệ sau thường có tập lệnh gồm nhiều lệnh mở rộng mới so với các vi xử lý thế hệ cũ 8-bit, nên sự phân loại các lệnh như trên cũng có thể tương đối đúng là đã bao gồm các nhóm lệnh của các vi xử lý 8-bit cũ.

Với một số loại vi xử lý kiến trúc RISC như Alpha AXP 64-bit (DEC), họ PowerPC 64-bit (IBM, Motorola), SPARC và SuperSPARC (Sun), họ MIPS RX000 64-bit,... thì sự phân loại tập lệnh đơn giản hơn, tương đối ít nhóm lệnh hơn so với vi xử lý kiến trúc CISC. Ví dụ, Sun SPARC phân ra 6 nhóm lệnh, đó là:

- + Nạp/cất giữ (Load/store)
- + Số học/logic/dịch
- + Chuyển điều khiển
- + Ghi/đọc các thanh ghi điều khiển (Read/write control registers)
- + Dấu phẩy động
- + Vận hành bộ đồng xử lý (không cần thiết sau này đối với phiên bản vi xử lý có FPU bên trong).

Tập lệnh của Sun SPARC chỉ có 62 lệnh cơ bản và 5 chế độ đánh địa chỉ: trực tiếp thanh ghi, ngay lập tức, gián tiếp thanh ghi với độ dịch, gián tiếp thanh ghi chỉ số, và tương đối với PC.

3.4. MÔ TẢ TẬP LỆNH CỦA VI XỬ LÝ

3.4.1. Nhóm lệnh chuyển dữ liệu

Đó là các lệnh thực hiện vận chuyển dữ liệu:

- + Thanh ghi - đến - thanh ghi (register-to-register)
- + Bộ nhớ - đến - thanh ghi (memory-to-register)
- + Thanh ghi - đến - bộ nhớ
- + Bộ nhớ - đến - bộ nhớ

+ Trực tiếp - đến - thanh ghi (Immediate-to-register)

+ Trực tiếp - đến - bộ nhớ.

Trong một số loại vi xử lý, dữ liệu được vận chuyển hay xử lý có thể có kích thước khác nhau: byte (B), nửa từ H (16-bit), từ W (32-bit) và từ kép D (64-bit). Tương ứng với chúng là các lệnh vận chuyển theo byte, theo nửa từ, theo từ và theo từ kép. Những vi xử lý 8-bit chỉ xử lý byte dữ liệu, vì thế chúng không có phân biệt các lệnh như trên. Còn vi xử lý 16-bit chỉ xử lý từ dữ liệu 16-bit (2 byte), nên chúng có phân biệt lệnh vận chuyển theo byte và theo từ (nhưng không có nửa từ, hay từ kép).

a) Trong vi xử lý 8-bit (8080/8085/Z80)

LD A, B; $A \leftarrow B$, chuyển nội dung thanh ghi B vào thanh ghi tích lũy A.

EX DE, HL; $DE \leftrightarrow HL$, trao đổi nội dung các cặp thanh ghi DE và HL với nhau.

LD A, (HL); $A \leftarrow (HL)$, nạp nội dung ngăn nhớ có địa chỉ ghi đôi thanh ghi HL vào thanh ghi tích lũy A.

LD A, 41h; $A \leftarrow 41h$, nạp trực tiếp giá trị 41h vào thanh ghi tích lũy A.

MVI B, 0AAh; $B \leftarrow AAh$, nạp trực tiếp giá trị AAh vào thanh ghi B (8085).

MOV C, A; $C \leftarrow A$, nạp nội dung thanh ghi tích lũy A vào thanh ghi C (8085).

IN A, (n); $A \leftarrow (n)$, nhận byte dữ liệu vào thanh ghi tích lũy A từ một cổng vào/ra có số hiệu (địa chỉ cổng vào/ra) là n.

OUT (n), A; $(n) \leftarrow A$, chuyển byte dữ liệu từ thanh ghi tích lũy A ra cổng vào/ra địa chỉ n.

b) Trong vi xử lý Intel 16-bit (8086/8088/80286)

MOV AX, CX; $AX \leftarrow CX$, nạp từ dữ liệu 16-bit từ thanh ghi CX vào thanh ghi AX.

MOV ARRAY[SI], AL; nạp byte dữ liệu vào mảng nhớ ARRAY có địa chỉ ghi trong thanh ghi SI.

MOVS/MOVSB; vận chuyển chuỗi các byte dữ liệu/chuỗi các từ dữ liệu.

REP MOVSB; lặp lại lệnh vận chuyển chuỗi các từ dữ liệu.

IN AL, DX; $AL \leftarrow (DX)$, đọc 1 byte dữ liệu từ cổng vào/ra có địa chỉ ghi trong DX.

OUT DX, AL; $(DX) \leftarrow AL$, gửi ra cổng vào/ra 1 byte.

c) Trong vi xử lý Intel 32-bit (từ 80386 đến Pentium)

MOV CL, AH; $CL \leftarrow AH$, vận chuyển byte từ AH vào CL.

MOV AX, BX; $AX \leftarrow BX$, vận chuyển từ 16-bit.

MOV EDX, EAX; $EDX \leftarrow EAX$, vận chuyển một từ kép 32-bit.

MOV [2000h], EAX; $(2000h) \leftarrow EAX$, vận chuyển dữ liệu 32-bit từ EAX vào ngăn nhớ có địa chỉ là 2000h.

XCHG EBX, ESI; $EBX \leftrightarrow ESI$.

PUSHA; đẩy nội dung 16-bit (từ) của tất cả các thanh ghi AX, CX, DX, BX, SP, BP, SI, và DI vào ngăn xếp.

PUSHAD; đẩy nội dung 32-bit (từ kép) của tất cả các thanh ghi EAX, ECX, EDX, EBX, ESP, EBP, ESI, và EDI vào ngăn xếp.

POPA; đọc từ ngăn xếp các dữ liệu 16-bit về tất cả các thanh ghi AX, CX, DX, BX, SP, BP, SI, và DI.

POPAD; đọc từ ngăn xếp các dữ liệu 32-bit (từ kép) về tất cả các thanh ghi EAX, ECX, EDX, EBX, ESP, EBP, ESI, và EDI.

IN EAX, 6; EAX $\leftarrow$ (6), đọc 1 từ kép dữ liệu (32-bit) từ cổng vào/ra địa chỉ là 6 vào EAX.

OUT DX, EAX; (DX) $\leftarrow$ (EAX), chuyển 1 từ kép dữ liệu (32-bit) từ EAX ra cổng vào/ra địa chỉ trong DX.

c) Trong họ vi xử lý Motorola M68000

MOVE.W A1, A5; A5 $\leftarrow$ A1, chuyển 1 từ 16-bit, 16-bit cao không có tác dụng.

MOVE A1, A5; chuyển 1 từ (ngầm định)

MOVE.L (A0), D1; D1 $\leftarrow$ (A0), chuyển 1 từ dài từ ngăn nhớ đến D1

MOVE (A1), (A2); (A2) $\leftarrow$ (A1), chuyển 1 từ từ ngăn nhớ đến ngăn nhớ

MOVEQ #1234H, D0; D0 $\leftarrow$ 1234, chuyển trực tiếp giá trị 1234 vào thanh ghi dữ liệu D0.

MOVEP D0, (4, A1); (4, A1) $\leftarrow$ D0, chuyển dữ liệu 32-bit từ D0 đến ngăn nhớ có địa chỉ được tạo bởi địa chỉ cơ sở 32-bit cộng với độ dịch 4.

MOVEA (8, SP), A0; A0 $\leftarrow$ (8, SP), chuyển dữ liệu từ ngăn xếp có địa chỉ tính bằng nội dung SP và độ dịch 8.

EXG D0, A0; D0 $\leftrightarrow$ A1.

MOVE16 (A0)+, (A1)+; chuyển khối dữ liệu có địa chỉ trong A0 đến vùng nhớ có địa chỉ trong A1 với "sự tự động tăng địa chỉ sau mỗi lần chuyển từ dữ liệu.

3.4.2. Nhóm lệnh số học và logic

Đây là nhóm lệnh cơ bản mà đơn vị số học và logic (ALU) trong bất kỳ vi xử lý nào cũng phải thực hiện. Đó là các lệnh số học: Cộng

(ADD), trừ (SUB), nhân (MUL), chia (DIV) và các lệnh logic: AND (và), OR (hoặc), XOR... Nhóm lệnh số học này thực hiện với 2 toán hạng, 3 toán hạng, với các số nguyên. Chúng thực hiện các thao tác giữa các thanh ghi, giữa thanh ghi và ngăn nhớ, giữa các ngăn nhớ với nhau, giữa thanh ghi với giá trị trực tiếp, giữa ngăn nhớ với giá trị trực tiếp, với giá trị nhớ Carry (CY, CF, C),... Số lượng các lệnh của nhóm này nhiều. Dưới đây là các ví dụ.

a) Trong vi xử lý 8-bit (8080/8085/Z80)

ADC (HL); $A \leftarrow A + (HL) + CY$, cộng nội ngăn nhớ địa chỉ trong HL với nội dung thanh ghi tích lũy A và giá trị nhớ carry (CY), kết quả chuyển lại vào A.

ADD A, 45h; $A \leftarrow A + 45h$, cộng trực tiếp giá trị 45h với A, kết quả vào A.

AND A, (HL); $A \leftarrow A \cap (HL)$, logic “và” giữa A và ngăn nhớ, kết quả chuyển vào A.

XOR A, 0Fh; $A \leftarrow A \oplus 0Fh$, cộng module 2 giữa A và giá trị 0Fh, kết quả chuyển vào A.

b) Trong vi xử lý Intel 16-bit (8086/8088/80286)

ADD CX, DX; $CX \leftarrow CX + DX$

AND CX, 0F0h; $CX \leftarrow CX \cap 0F0h$

c) Trong vi xử lý Intel 32-bit (từ 80386 đến Pentium)

ADD AL, 8; $AL \leftarrow AL + 8$, cộng theo byte

ADD BX, CX; $BX \leftarrow BX + CX$, cộng theo từ 16-bit

SUB EAX, EBX; $EAX \leftarrow EAX - EBX$, trừ theo từ kép 32-bit

ADC EAX, [EBP]; $EAX \leftarrow EAX + (EBP) + CF$

CMP EAX, [EBP]; $EAX - (EBP)$, so sánh nội dung EAX và ngăn nhớ

IMUL DX, BX, 300; $DX \leftarrow BX \times 300$

IMUL EDX, EBP; $EDX \leftarrow EDX \times EBP$

DIV CL; AX/CL, thương số trong AL, số dư trong AH

IDIV BX; (DX, AX)/BX, thương số trong EAX, số dư trong EDX.

d) Trong họ vi xử lý *Motorola M68000*

MULS.L D0, D4:D5; $(D4, D5) \leftarrow D0 \times D5$, nhân từ dài

DIVS.L D0, D1:D2; $(D1, D2)/D0$, chia từ dài, thương số trong D2, số dư trong D1.

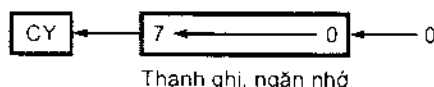
3.4.3. Nhóm lệnh dịch và quay vòng

Nhóm lệnh này có trong tất cả các loại vi xử lý từ 8-bit đến tiên tiến hiện nay, vì chúng là những lệnh không thể thiếu được cho xử lý của ALU. Đó là các lệnh dịch nội dung của thanh ghi trong vi xử lý về bên phải, bên trái, dịch quay vòng. Có các lệnh dịch logic và dịch số học. Xét các phép dịch của vi xử lý 8-bit.

Dịch logic chỉ thực hiện dịch nội dung thanh ghi về bên trái hay về bên phải đi một số bit, trong đó, cờ CARRY (CY) là một phần của nội dung cần dịch, và giá trị bit 0 được đưa vào chiếm chỗ của bit bị dịch đi:

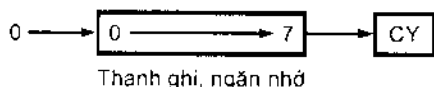
+ *Dịch trái logic:*

Dịch trái logic cho kết quả giống như dịch trái số học. Nội dung thanh ghi hay ngăn nhớ dịch về trái phía bit CY, bit 0 chiếm chỗ bit thấp nhất LSB:



+ *Dịch phải logic:*

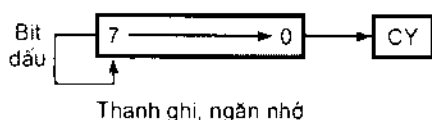
SRL m; Dịch phải logic nội dung thanh ghi hay ngăn nhớ về phía bit CY, bit 0 chiếm chỗ bit cao nhất (MSB):



Dịch số học được coi như là phép nhân 2 khi dịch trái 1 bit và chia 2 khi dịch phải 1 bit. Quy định bit thấp LSB nằm ở bên phải của số nhị phân, và bit cao nhất MSB nằm ở tận cùng bên trái. Nếu bit cao nhất là bit dấu thì lệnh dịch phải mở rộng bit dấu về bên phải theo số lượng bit được dịch.

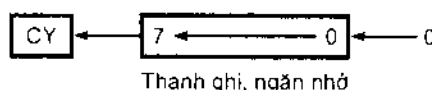
+ *Dịch phải số học:*

SRA m; Dịch trái số học nội dung thanh ghi hay ngăn nhớ về phía bit CY, bit 0 chiếm chỗ bit thấp nhất LSB:



+ *Dịch trái số học:*

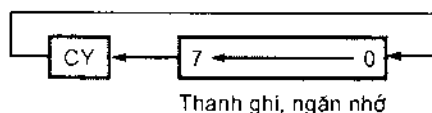
SLA m; Dịch trái số học nội dung thanh ghi hay ngăn nhớ về phía bit CY, bit 0 chiếm chỗ bit thấp nhất LSB:



Dịch quay vòng là bit thấp nhất và bit cao nhất thay chỗ cho nhau;

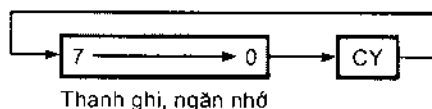
+ *Dịch trái quay vòng:*

RL m; Nội dung của thanh ghi hay ngăn nhớ dịch trái quay vòng tính cả CY:



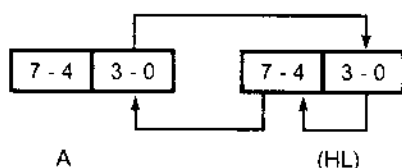
+ *Dịch phải quay vòng:*

RR m; Nội dung của thanh ghi hay ngăn nhớ dịch phải quay vòng tính cả CY:

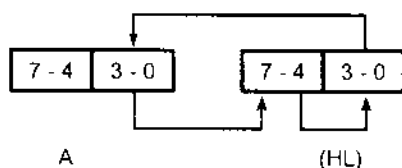


Ngoài ra có một số lệnh dịch các chữ số BCD (digit: 4 bit) sang phải hoặc trái để thực hiện cho các xử lý các số BCD. Ví dụ các lệnh của Z80:

RLD; Dịch trái và phải quay vòng chữ số BCD (4 bit) giữa thanh tích lũy A và ngăn nhớ có 16-bit địa chỉ trong cặp thanh ghi HL. Nội dung nửa cao thanh tích lũy không bị ảnh hưởng:



RRD; Dịch phải và phải quay vòng chữ số BCD (4 bit) giữa thanh tích lũy A và ngăn nhớ có 16-bit địa chỉ trong cặp thanh ghi HL. Nội dung nửa cao thanh tích lũy không bị ảnh hưởng:



3.4.4. Nhóm lệnh chuyển điều khiển

Nhóm lệnh này bao gồm: các lệnh nhảy hay rẽ nhánh (Jump, Branch), các lệnh gọi tới các chương trình con (Call), và nhảy trở lại từ chương trình con. Chúng đều có trong các vi xử lý từ 8-bit đến tiên tiến hiện nay và thực hiện các thao tác tương tự nhau. Phân biệt các lệnh chuyển điều khiển có điều kiện (phụ thuộc vào các giá trị các bit cờ trong thanh ghi cờ của bộ vi xử lý) và chuyển điều khiển vô điều kiện. Trong lập trình, các lệnh chuyển điều khiển được dùng để rẽ nhánh hay nhảy tới các chương trình con, thủ tục khi một điều kiện trạng thái của hệ thống được thỏa mãn. Trạng thái của hệ thống được thể hiện qua thanh ghi cờ (hoặc từ trạng thái PSW).

a) Trong vi xử lý 8-bit (8080/8085/Z80)

+ Nhảy vô điều kiện:

JMP addr; $PC \leftarrow \text{addr}$. Địa chỉ (addr) ghi trong từ lệnh là byte 2 và byte 3, byte 1 là mã lệnh (Opcode). Byte 2 là 8-bit thấp và byte 3 là 8-bit cao của địa chỉ 16-bit. JMP thực hiện nhảy tới thực hiện lệnh có địa chỉ addr.

+ Nhảy có điều kiện:

JNZ addr; $PC \leftarrow \text{addr}$, nếu điều kiện $Z = 0$ thỏa mãn (tức là kết quả của lệnh trước đó khác 0). Nếu điều kiện không thỏa mãn ($Z = 1$) thì lệnh JNZ không thực hiện, mà lệnh tiếp theo JNZ sẽ được thực hiện.

Trong các vi xử lý 8-bit này, có các lệnh rẽ nhánh có điều kiện theo các cờ: Z (Zero), CY (Carry), S (Sign), P (Parity), AC (Auxiliary).

+ Gọi tới chương trình con vô điều kiện:

CALL addr; $(SP-1) \leftarrow PC_H$, $(SP-2) \leftarrow PC_L$, $PC \leftarrow \text{addr}$.

Lệnh gọi tới chương trình con vô điều kiện khác với các lệnh nhảy vô điều kiện ở chỗ nó gửi địa chỉ của lệnh tiếp theo chứa trong PC của chương trình chính (sau lệnh CALL) vào cất trong ngăn xếp: byte cao $PC_H \rightarrow (SP-1)$, $PC_L \rightarrow (SP-2)$ và sau đó mới chuyển địa chỉ đầu của chương trình con addr vào PC để nhảy đến chương trình con thực hiện: $\text{addr} \rightarrow PC$. Cũng có dạng lệnh gọi tới chương trình con có điều kiện.

+ Gọi tới chương trình con có điều kiện:

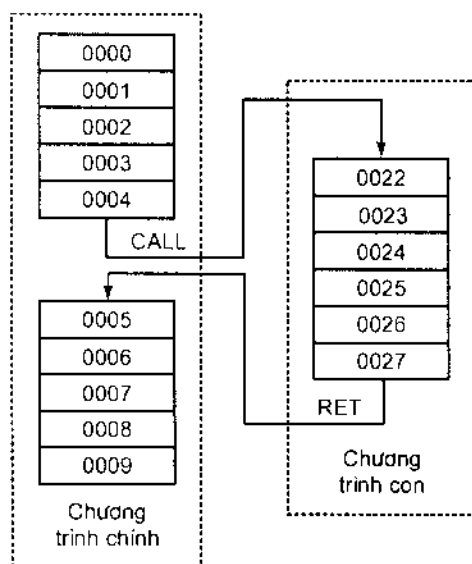
CALL NZ, addr;

Nếu điều kiện NZ thỏa mãn ($Z = 0$) thì CALL NZ, addr thực hiện như gọi vô điều kiện CALL addr. Nếu điều kiện NZ không thỏa mãn ($Z = 1$) thì lệnh gọi này không thực hiện được và lệnh tiếp theo nó trong chương trình chính sẽ thực hiện.

+ *Trở về từ chương trình con:*

Kết thúc chương trình con, lệnh cuối cùng phải là lệnh RET (nhảy trở lại chương trình chính từ chương trình con). RET phục hồi lại địa chỉ tiếp theo lệnh gọi từ ngăn xếp. Thao tác của RET là:

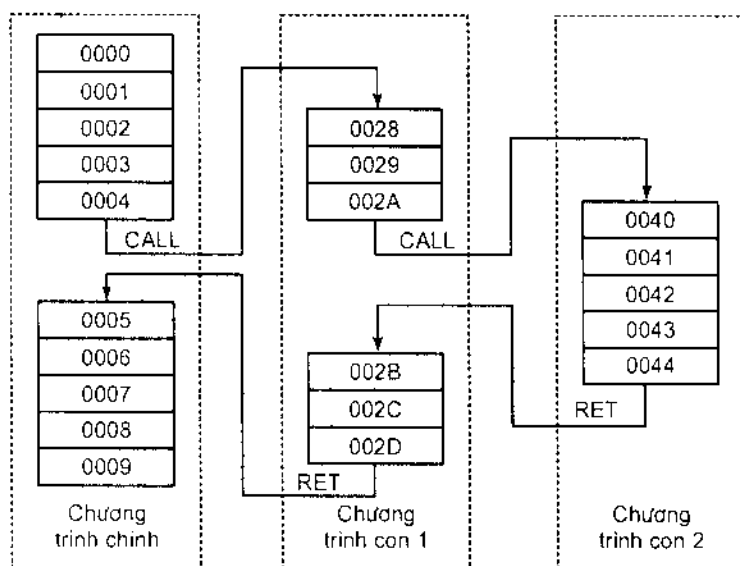
RET; $PC_L \leftarrow (SP)$, $PC_H \leftarrow (SP+1)$



Hình 3.23: Thực hiện gọi tới chương trình con và trở về từ chương trình con

Hình 3.23 minh họa tiến trình thực hiện một chương trình chính gồm 10 lệnh (từ 0000 đến 0009), và một chương trình con với lệnh đầu tiên là 0022 và lệnh cuối là 0027. Lệnh 0004 trong chương trình chính là lệnh gọi tới chương trình con. Lệnh gọi 0004 kéo theo sự cất giữ địa chỉ của lệnh tiếp theo của chương trình, lệnh 0005. Lệnh 0027 là lệnh RET của chương trình trở về chương trình chính. Lệnh 0027 thực hiện kéo theo sự phục hồi địa chỉ của lệnh 0005 về bộ đếm chương trình PC. Cũng có thể có những lệnh gọi tới chương trình con sâu hơn, tức là trong một chương trình con lại có lệnh gọi tới chương trình con khác. Trong tiến trình gọi tới các chương trình con địa chỉ đỉnh ngăn xếp trong con trở ngăn xếp giảm dần giá trị, và theo tiến trình nhảy trở về chương trình mẹ (chính) bằng lệnh RET đỉnh của ngăn xếp lại tăng

dẫn theo chiều ngược. Hình 3.24 mô tả tiến trình thực hiện các chương trình con phức tạp hơn.



Hình 3.24: Thực hiện gọi tới các chương trình con và trở về từ các chương trình con

+ Các lệnh ngắt:

Ngắt mềm (bằng lệnh ngắt) hay ngắt cứng (tín hiệu ngắt từ thiết bị ngoài) đều dẫn đến thao tác gọi tới chương trình con phục vụ ngắt (hay xử lý ngắt) tương tự như lệnh CALL. Chẳng hạn, Z80 có lệnh yêu cầu ngắt từ các cổng RST p, trong đó p là véc-tơ ngắt. Giá trị p là địa chỉ của chương trình con xử lý ngắt phải nạp mới vào bộ đếm chương trình khi thực hiện lệnh này để gọi tới chương trình con xử lý ngắt, thao tác của lệnh này như sau:

$$\text{RST } p; (\text{SP}-1) \leftarrow \text{PC}_H, (\text{SP}-2) \leftarrow \text{PC}_L, \text{PC}_H \leftarrow 0, \text{PC}_L \leftarrow p$$

8085 cũng có lệnh RST n, nhưng giá trị $n \times 8$ mới là địa chỉ của chương trình con xử lý ngắt phải chuyển vào bộ đếm chương trình khi thực hiện lệnh này. Ngắt có thể có ngắt che được và ngắt không che được (NMI), vì vậy trong chương trình con xử lý ngắt có thể dùng lệnh

trở về từ xử lý ngắt che được (RETI) hoặc trở về từ xử lý ngắt không che được RETN phụ thuộc vào nguyên nhân ngắt.

b) Trong vi xử lý Intel 16-bit và 32-bit

Trong các vi xử lý từ Intel 16-bit đến 32-bit do có quản lý bộ nhớ theo các đoạn, mà mỗi đoạn có dung lượng ngầm định là 64 kbyte, do vậy có sự phân biệt với vi xử lý 8-bit.

+ *Các lệnh nhảy và gọi:*

Có thể được loại bằng 2 cách:

Cách thứ nhất theo địa chỉ đích:

- Trục tiếp: địa chỉ đích của lệnh nhảy là một phần của từ lệnh.
- Gián tiếp: địa chỉ đích được cất giữ trong một thanh ghi hoặc một ngăn nhớ và lệnh chứa một con trỏ trỏ đến địa chỉ đích.

Ví dụ:

JMP addr; nhảy trực tiếp đến địa chỉ addr (tượng trưng)

JMP EBX; nhảy gián tiếp đến lệnh với địa chỉ chứa trong EBX

JMP [EBX]; nhảy gián tiếp đến lệnh với địa chỉ trong ngăn nhớ.

CALL sub1; gọi tới chương trình con địa chỉ sub1.

Cách thứ 2 theo vị trí địa chỉ lệnh có nằm trong cùng đoạn mã hay không:

- SHORT; nhảy ngắn đến lệnh nằm trong cùng đoạn mã với độ dịch 8-bit nhưng ở mã bù (bit 8 là bit dấu). Khoảng nhảy ngắn dịch tương đối từ -128 đến +127.

- NEAR; nhảy gần đến lệnh nằm trong cùng đoạn mã (CS) với độ dịch 16-bit và khoảng nhảy gần tương đối từ -32768 đến +32767.

- NEAR; nhảy gần đến lệnh nằm trong cùng đoạn mã (CS) với độ dịch 32-bit và khoảng nhảy gần tương đối từ -2^{31} đến $+2^{31}-1$.

- FAR; nhảy xa đến lệnh không cùng nằm trong một đoạn mã.

Các lệnh nhảy có điều kiện sử dụng nhảy ngắn. Ví dụ:

JMP SHORT; nhảy vô điều kiện đến địa chỉ ngắn cùng trong đoạn mã.

JMP NEAR PTR ADDR; nhảy vô điều kiện đến địa chỉ ADDR gần, trong cùng đoạn mã.

JMP FAR PTR ADDR; nhảy vô điều kiện đến địa chỉ ADDR xa không cùng đoạn mã.

JGE ADDR; nhảy ngắn nếu lớn hơn hoặc bằng.

JNS ADDR; nhảy ngắn nếu cờ dấu S = 0 (not sign).

CALL NEAR PTR PROC1; gọi tới chương trình con địa chỉ PROC1.

CALL AX; gọi gần đến chương trình con có địa chỉ chứa trong AX.

+ Các lệnh ngắt:

Các lệnh ngắt che được của Intel từ 16-bit và 32-bit có dạng:

INT n

Chúng gọi tới chương trình con xử lý ngắt, kèm theo cất giữ vào ngăn xếp địa chỉ của lệnh tiếp tiếp sau lệnh INT n trong chương trình chính, tức là cất giữ nội dung CS, IP và các giá trị cờ. Các lệnh IRET (hoặc IRETD) phải có trong chương trình con xử lý ngắt để thực hiện nhảy về chương trình chính, nhưng khác với RET, IRET đọc các cờ (trước đó đã được cất giữ vào ngăn xếp nhờ lệnh PUSHF) từ ngăn xếp ra thanh ghi cờ FLAGS (16-bit), và IRETD đọc các cờ (trước đó đã được cất giữ vào ngăn xếp nhờ lệnh PUSHFD) từ ngăn xếp ra thanh ghi cờ EFLAGS (32-bit).

+ Các lệnh lặp chu trình:

Các lệnh lặp chu trình vô điều kiện như LOOP, có điều kiện như LOOPE, LOOPZ, LOOPNZ, LOOPNE về bản chất cũng là những lệnh nhảy nhưng phụ thuộc vào giá trị đếm chứa trong CX (16-bit), hay ECX (32-bit). Lệnh LOOP trước hết giảm giá trị CX (hay ECX) đi 1 và kiểm tra ngay CX (ECX). Nếu CX (ECX) $\neq$ 0 thì chương trình

thực hiện nhảy. Nếu CX (ECX) = 0 thì thực hiện ngay lệnh tiếp theo lệnh LOOP.

Ví dụ đoạn chương trình:

MOV ECX, 10; nạp số vòng lặp chu trình

L: MOV EAX, [EBP]; đọc 1 từ kép (32-bit) từ ngăn nhớ vào EAX

OUT port, EAX; đưa ra cổng vào/ra

ADD EBP, 4; tạo địa chỉ ngăn nhớ tiếp theo

LOOP L; ECX $\leftarrow$ ECX - 1, nếu ECX khác 0 thì nhảy về lệnh địa chỉ L, nếu ECX = 0 thì thực hiện lệnh MOV ECX, 40.

MOV ECX, 40

d) Trong họ vi xử lý Motorola M68000

+ Các lệnh nhảy và gọi:

BRA L2; PC $\leftarrow$ PC + d, rẽ nhánh đến đoạn chương trình có nhãn L2 với độ dịch d

JSR addr; SP $\leftarrow$ SP - 4, (SP) $\leftarrow$ PC, PC $\leftarrow$ addr, nhảy đến chương trình con có địa chỉ addr (tương tự như CALL)

BSR L4; SP $\leftarrow$ SP - 4, (SP) $\leftarrow$ PC, PC $\leftarrow$ PC + d, rẽ nhánh đến chương trình con có địa chỉ với độ dịch d

BNE L5; PC $\leftarrow$ PC + d, rẽ nhánh đến nếu điều kiện NE (không bằng) thỏa mãn

CALLM (A0); gọi tới một module địa chỉ chứa trong A0 (chỉ có ở 68020).

Lệnh nhảy có điều kiện:

Bcc L2; PC $\leftarrow$ PC + d, rẽ nhánh đến địa chỉ L2 nếu thỏa mãn điều kiện

+ Trở về từ chương trình con:

RTS; PC $\leftarrow$ (SP), SP $\leftarrow$ SP + 4, trở về từ chương trình con

RTD #50h; $PC \leftarrow (SP)$, $SP \leftarrow SP + 4 + 50h$, trở về và phân bổ lại (deallocate)

RTR; $CCR \leftarrow (SP)$, $SP \leftarrow SP + 2$, $PC \leftarrow (SP)$, $SP \leftarrow SP + 4$, trở về với phục hồi các mã điều kiện và thanh ghi CCR.

RTM Rn; trở về từ một module chương trình (68020).

+ *Lệnh ngoại lệ (ngắt):*

ILLEGAL; là lệnh thực hiện một ngoại lệ (một chương trình con xử lý ngoại lệ, ví dụ, như xử lý ngắt): $SSP \leftarrow SSP - 2$, $(SSP) \leftarrow$ véc-tơ offset, $SSP \leftarrow SSP - 4$, $(SSP) \leftarrow PC$, $SSP \leftarrow SSP - 2$, $(SSP) \leftarrow SR$, và địa chỉ véc-tơ của lệnh ngoại lệ chuyển vào PC để thực hiện.

TRAP #vector; thực hiện bẫy, gọi tới chương trình con xử lý bẫy: $SSP \leftarrow SSP$,

$(SSP) \leftarrow$ offset, $SSP \leftarrow SSP - 4$, $(SSP) \leftarrow PC$,

$SSP \leftarrow SSP - 2$, $(SSP) \leftarrow SR$, $PC \leftarrow$ vector address.

Các lệnh bẫy theo mã có điều kiện (CC) trong thanh ghi mã điều kiện CCR:

TRAPcc; thực hiện nếu thỏa mãn cc

TRAPcc.L #<data>

Trong họ vi xử lý M68000 các ngoại lệ là những lệnh nhảy tới chương trình xử lý ngắt trong các trường hợp có lỗi: lỗi truy cập, lỗi địa chỉ, chia cho 0, cấu hình sai của MMU, sai thao tác của MMU, lỗi khuôn dạng, vi phạm thẩm quyền, sai lệnh, xóa ban đầu ngăn xếp, các ngắt cho vào/ra, bẫy,... và mỗi loại có gán véc-tơ ngoại lệ (Exception vector). Véc-tơ ngoại lệ có số hiệu (vector number) để tính độ dịch (vector offset) địa chỉ của chương trình con phục vụ một ngoại lệ. Có tới 256 số hiệu véc-tơ (từ 0 đến 255).

3.4.5. Nhóm lệnh xử lý bit

Nhóm này gồm các lệnh: kiểm tra bit (bằng 0 hay 1), tác động tới giá trị cờ như: xóa bit, thiết lập bit, trích bit và chuyển đến đích khác, trích bit và mở rộng dấu.

a) Trong vi xử lý 8-bit (8080/8085/Z80)

BIT b, r; $Z \leftarrow r_b$, kiểm tra giá trị bit thứ b của thanh ghi r (A, B, C, D, E, H, L), kết quả tác động đến cờ Z.

BIT b, (HL); $Z \leftarrow (HL)_b$, kiểm tra giá trị bit thứ b của ngăn nhớ (HL), kết quả tác động đến cờ Z.

SET b, r; $r_b \leftarrow 1$, thiết lập bit thứ b của thanh ghi r bằng 1.

SET b, (HL); $(HL)_b \leftarrow 1$, thiết lập bit thứ b của ngăn nhớ (HL) bằng 1.

RES b, m; $m_b \leftarrow 0$, xóa bit thứ b trong m, m là thanh ghi hay ngăn nhớ (HL), (IX + d), (IY + d).

SCF; $CY \leftarrow 1$, lập bit cờ carry = 1.

CCF; $CY \leftarrow \overline{CY}$, bù giá trị cờ carry.

EI; $IFF \leftarrow 1$, lập bit (flip-flop) cho phép ngắt = 1.

DI; $IFF \leftarrow 0$, xóa bit cho phép ngắt để cấm ngắt (ngắt che được).

b) Trong vi xử lý Intel 16-bit (8086/8088/80286)

STC; $C \leftarrow 1$, lập bit cờ carry = 1

STD; $D \leftarrow 1$, lập bit cờ hướng = 1

STI; $I \leftarrow 1$, lập bit cờ cho phép ngắt = 1

CLC; $C \leftarrow 0$, xóa bit cờ carry

CLD; $D \leftarrow 0$, xóa bit cờ D

CLI; $I \leftarrow 0$, xóa bit cờ I để không cho phép ngắt (ngắt che được).

c) Trong vi xử lý Intel 32-bit (từ 80386 đến Pentium)

Ngoài các lệnh STC, STD, STI, CLC, CLD, CLI giống như của vi xử lý Intel 16-bit, trong các vi xử lý Intel 32-bit có các lệnh kiểm tra bit trong thanh ghi hoặc trong ngăn nhớ như: BT - kiểm tra bit, BTC - kiểm tra bit và bù, BTR - kiểm tra bit và xóa, BTS - kiểm tra bit và thiết lập, BSF - quét kiểm tra tiến các bit (bit scan forward), BSR - quét

đảo các bit (bit scan reverse), CMC - bù cờ carry, CLTS - xóa cờ chuyển đổi nhiệm vụ trong thanh ghi CR0.

d) Trong họ vi xử lý Motorola M68000

Các lệnh xử lý bit trong họ M68000 tương đối phong phú, không chỉ có các lệnh thiết lập, kiểm tra (BSET, BTST) xóa từng bit (BCLR), kiểm tra và trao đổi bit (BCHG) trong thanh ghi hoặc ngăn nhớ mà xử lý cả một nhóm bit, như:

BFSET <EA>{offset:width}; lập một nhóm (trường) bit = 1 trong thanh ghi hoặc ngăn nhớ có địa chỉ EA, trường bit có độ rộng (số bit) width và có độ dịch (offset) tính từ bit thấp nhất (bit 0).

BFCLR <EA>{offset:width}; xóa một nhóm (trường) = 0 trong thanh ghi hoặc ngăn nhớ.

BFEXTS <EA>{offset:width}, Dn; chuyển một trường bit từ nguồn <EA> đến đích là thanh ghi Dn.

BFCHG <EA>{offset:width}; chuyển một trường bit từ nguồn đến một trường bit của đích.

3.4.6. Nhóm lệnh điều khiển hệ thống

Nhóm lệnh này tác động trực tiếp đến điều hành của bộ vi xử lý và các khối chức năng bên trong nó, như các khối quản lý bộ nhớ (MMU) và bộ nhớ dự trữ (cache memory), khối MMX (trong Intel 32-bit). Ví dụ:

+ Các lệnh dừng:

HALT; dừng CPU (Z80)

HLT; (các vi xử lý Intel)

Hầu như tất cả các loại vi xử lý đều có lệnh dừng (HALT, HLT), lệnh này bắt buộc bộ vi xử lý dừng hoạt động, Các thanh ghi và các cờ không có tác dụng.

STOP #data; $SR \leftarrow \text{data}$, chuyển dữ liệu vào thanh ghi trạng thái của vi xử lý và dừng vi xử lý trong chế độ giám sát (Motorola M68000).

+ *Lệnh khóa bus hệ thống:*

LOCK; tạo tín hiệu LOCK# = 0 (Intel từ 16-bit đến 32-bit) nhằm cấm bộ vi xử lý sử dụng BUS hệ thống trong các trường hợp có truy cập trực tiếp bộ nhớ của thiết bị ngoại vi (DMA). Lệnh LOCK chỉ được thực hiện cùng với các lệnh: BT, BTS, BTR, BTC, XCHG, ADD, OR, ADC, SBB, AND, SUB, XOR, NEG, INC, và DEC.

UNLOCK; bỏ khóa BUS hệ thống (Intel 80860).

+ *Một số lệnh khác:*

NOP - không thực hiện lệnh nào cả (nhiều vi xử lý khác nhau)

INVD; lỗi ở cache bên trong chip (Intel 32-bit từ 80486)

XLAT; chuyển đổi từ một hệ thống mã hóa sang hệ thống mã hóa khác, như: từ mã EBCDIC sang ASCII (intel 32-bit).

RESET; tạo tín hiệu xóa RST0 (M68000)

ANDI #data, SR; $SR \leftarrow data \cap SR$ (M68000)

ORI #data, SR; $SR \leftarrow data \cup SR$ (M68000)

EORI #data, SR; $SR \leftarrow data \oplus SR$ (M68000)

Nhìn chung sự phân loại nhóm lệnh này đối với từng loại vi xử lý của các nhà sản xuất khác nhau tương đối khác nhau.

3.4.7. Nhóm lệnh xử lý dấu phẩy động

Các bộ vi xử lý có khối xử lý dấu phẩy động bên trong (FPU) đòi hỏi phải có các lệnh này. Nhóm lệnh này cũng có các lệnh vận chuyển dữ liệu (FMOV), các lệnh số học (FADD, FSUB, FMUL, FDIV), so sánh (FCOMP), và đặc biệt là các lệnh thực hiện các phép tính hàm lượng giác (FCOS, FSIN), khai căn bậc hai (FSQRT), lấy giá trị tuyệt đối (FABS) và một số lệnh thực hiện các phép tính khác đòi hỏi độ chính xác cao. Các loại vi xử lý cũ thường phải có bộ đồng xử lý riêng chuyên để thực hiện nhóm lệnh này.

a) Trong vi xử lý Intel 32-bit (từ 80386 đến Pentium)

Các bit từ 11 đến 13 (trường TOP) của thanh ghi từ trạng thái của đơn vị FPU chỉ đến thanh ghi dữ liệu của FPU, mà thanh ghi đó hiện là đỉnh của ngăn xếp. Đỉnh của thanh ghi ngăn xếp được gán là ST(0) hay đơn giản là ST. Những thanh ghi khác được chỉ số hóa theo đỉnh của ngăn xếp. Ví dụ, cho rằng trường TOP của thanh ghi từ trạng thái của FPU chứa giá trị 011 (3). Như vậy, thanh ghi R3 là đỉnh của ngăn xếp, ST(0), và các thanh ghi còn lại được chỉ số hóa như sau:

R4 ST(1)

R5 ST(2)

R6 ST(3)

R7 ST(4)

R0 ST(5)

R6 ST(6)

R7 ST(7)

Đỉnh của ngăn xếp được sử dụng như là toán hạng đích trong nhiều lệnh xử lý các số dấu phẩy động của các vi xử lý Intel 32-bit có FPU bên trong chip. Trong ví dụ này đỉnh ngăn xếp là ST(0) = R3, hay đơn giản là ST, khi đó:

FADD ADDR; $ST \leftarrow (ADDR) + ST$, cộng các số dấu phẩy động trong ngăn nhớ (ADDR) và trong thanh ghi R3, kết quả nằm lại R3.

FADD ST(3); $R3 \leftarrow R6 + R3$

FABS; thay thế nội dung R3 bằng giá trị tuyệt đối của nó.

FLD ADDR; đẩy nội dung ngăn nhớ (ADDR) vào R2, sau đó R2 trở thành đỉnh của ngăn xếp, tức là ST(0) = R2.

Các vi xử lý Intel 16-bit và 32-bit 80386 dùng các chip đồng xử lý riêng (8087, 80287, 80387). Chỉ đến 80486 và Pentium mới các FPU bên trong.

+ *Lệnh khóa bus hệ thống:*

LOCK; tạo tín hiệu LOCK# = 0 (Intel từ 16-bit đến 32-bit) nhằm cấm bộ vi xử lý sử dụng BUS hệ thống trong các trường hợp có truy cập trực tiếp bộ nhớ của thiết bị ngoại vi (DMA). Lệnh LOCK chỉ được thực hiện cùng với các lệnh: BT, BTS, BTR, BTC, XCHG, ADD, OR, ADC, SBB, AND, SUB, XOR, NEG, INC, và DEC.

UNLOCK; bỏ khóa BUS hệ thống (Intel 80860).

+ *Một số lệnh khác:*

NOP - không thực hiện lệnh nào cả (nhiều vi xử lý khác nhau)

INVD; lỗi ở cache bên trong chip (Intel 32-bit từ 80486)

XLAT; chuyển đổi từ một hệ thống mã hóa sang hệ thống mã hóa khác, như: từ mã EBCDIC sang ASCII (intel 32-bit).

RESET; tạo tín hiệu xóa RST0 (M68000)

ANDI #data, SR; $SR \leftarrow data \cap SR$ (M68000)

ORI #data, SR; $SR \leftarrow data \cup SR$ (M68000)

EORI #data, SR; $SR \leftarrow data \oplus SR$ (M68000)

Nhìn chung sự phân loại nhóm lệnh này đối với từng loại vi xử lý của các nhà sản xuất khác nhau tương đối khác nhau.

3.4.7. Nhóm lệnh xử lý dấu phẩy động

Các bộ vi xử lý có khối xử lý dấu phẩy động bên trong (FPU) đòi hỏi phải có các lệnh này. Nhóm lệnh này cũng có các lệnh vận chuyển dữ liệu (FMOV), các lệnh số học (FADD, FSUB, FMUL, FDIV), so sánh (FCOMP), và đặc biệt là các lệnh thực hiện các phép tính hàm lượng giác (FCOS, FSIN), khai căn bậc hai (FSQRT), lấy giá trị tuyệt đối (FABS) và một số lệnh thực hiện các phép tính khác đòi hỏi độ chính xác cao. Các loại vi xử lý cũ thường phải có bộ đồng xử lý riêng chuyên để thực hiện nhóm lệnh này.

b) Trong họ vi xử lý Motorola M68000

Trong họ Motorola M68000, các lệnh dấu phẩy động là của các tập lệnh các bộ đồng xử lý M68881/M68882. Chúng được dùng kết hợp với chip vi xử lý từ M68000 đến M68030. Từ M68040 đã có FPU bên trong chip. Họ M68000 có 2 loại lệnh dấu phẩy động: Dyaic - 2 toán hạng, và Monadic - một toán hạng.

Trong các lệnh Dyaic (lệnh đôi), nguồn có thể là ngăn nhớ, một thanh ghi dữ liệu số nguyên Dn, hoặc thanh ghi dữ liệu dấu phẩy động Fpm. Đích luôn là thanh ghi dữ liệu dấu phẩy động Fpm, và dạng chung của lệnh là:

Fdop EA, Fpm; (EA) thao tác (Fpm) $\rightarrow$ Fpm, trong đó dop là các thao tác lệnh. Các lệnh đôi như:

Cộng: FADD, FSADD, FDAD

Số sánh: FCMP

Chia: FDIV, FSDIV, FDIV

Nhân: FMUL, FSMUL, FDMUL

Trừ: FSUB, FSSUB, FDSUB

Các chữ S, D là yêu cầu kết chính xác đơn (single) hay chính xác kép (double).

Các lệnh đơn như:

Lấy giá trị tuyệt đối: FABS, FSABS, FDABS

Đảo: FNEG, FSNEG, FDNEG

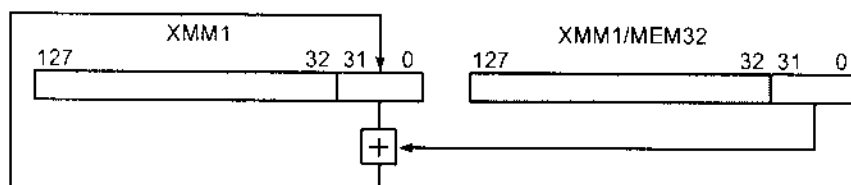
Căn bậc hai: FSQRT, FSSQRT, FDSQRT.

3.4.8. Nhóm lệnh cho các chức năng đặc biệt

Các khối chức năng đặc biệt được cấy trong các bộ vi xử lý công nghệ cao hiện nay có thể là khối đồ họa, khối xử lý tín hiệu, khối xử lý ảnh, khối xử lý các ma trận và véc-tơ. Chúng đòi hỏi phải có nhóm lệnh riêng, chuyên dụng để đảm bảo tốc độ xử lý nhanh, nâng cao hiệu suất xử lý của CPU. Chẳng hạn các khối MMX trong các vi xử lý

Pentium có tập lệnh MMX gồm: các lệnh số học với dữ liệu (byte, từ, từ dài) đóng gói (packed data), các lệnh vận chuyển, các lệnh logic theo bit, quản lý trạng thái của MMX (EMMS - làm trống trạng thái MMX), so sánh các dữ liệu đóng gói,... và thao tác với các thanh ghi MMX (MM0 - MM7), bộ nhớ. Ngoài ra từ Pentium III đã có bổ sung thêm các lệnh xử lý đa phương tiện với nhiều dữ liệu dấu phẩy động dạng SIMD (một lệnh xử lý với nhiều dữ liệu) với các thanh ghi dùng riêng XMM (XMM0 - XMM7), bởi Pentium III đã có kiến trúc giống như RISC. Những lệnh SIMD xử lý song song nhiều dữ liệu được đóng gói và vô hướng (Scalar). Các lệnh xử lý đa phương tiện này (multimedia) có phân biệt chữ S (Scalar) chỉ xử lý vô hướng và P (Parallel) để xử lý song song. Ví dụ các lệnh số học sau đây:

ADDSS XMM1, XMM2/MEM32; $XMM1 \leftarrow XMM1 + XMM2$ /MEM32, cộng vô hướng phần thập của dữ liệu dấu phẩy động trong thanh ghi XMM1 với nội dung của thanh ghi XMM2 hoặc ngăn nhớ dữ liệu 32-bit, kết quả số dấu phẩy động chính xác đơn 128-bit nằm lại XMM1 (hình 3.25).



Hình 3.25: Thực hiện phép cộng vô hướng ADDSS

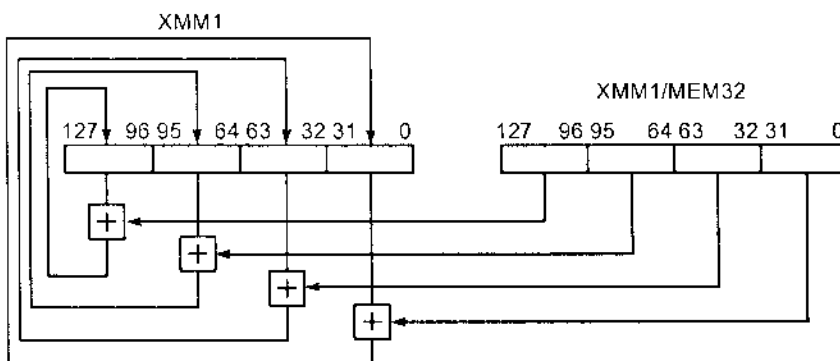
ADDPS XMM1, XMM2/MEM128; cộng kiểu SIMD (song song) 4 toán hạng 32-bit được đóng gói ở XMM1 và XMM2/hoặc ngăn nhớ MEM128, kết quả nằm lại XMM1 là dữ liệu gồm 4 đóng gói (hình 3.26).

ANDNPS XMM0, [EDI]; And-not song song nội dung của XMM0 và ngăn nhớ được địa chỉ bởi thanh ghi EDI.

DIVPS XMM0, XMM3; chia song song nội dung của XMM0 và XMM3.

XORPS XMM0, XMM1; cộng song song module 2 nội dung XMM0 và XMM1, kết quả chuyển vào XMM0.

Nhóm lệnh này có các lệnh nguyên tử (atomic instructions), chúng điều khiển các truy cập tới các vùng nguy hiểm trong các hệ thống máy tính có cấu trúc đa vi xử lý nhằm đảm bảo thực hiện đúng đắn các đánh tín hiệu của các cờ không bị ngắt và không sai sót. Nguyên tử, có nghĩa là không bị phân chia, không bị ngắt. Dạng chung của một lệnh nguyên tử trong một số loại vi xử lý hiện đại là thao tác TAF (Test And f), trong đó, $f(V, e)$ là hàm số học hoặc logic của 2 giá trị V và e. Chẳng hạn $f = \text{OR}(V, \text{TRUE})$, $e = \text{TRUE}$. Thao tác này gọi là TAS (Test And Set), kiểm tra và thiết lập. TAS là một lệnh không ngắt được và nó được thực hiện trong một chu kỳ bus đọc-thay đổi-ghi RMW (Read-Modify-Write). Nó đọc giá trị của tín hiệu cờ trong bộ nhớ, kiểm tra ở đâu 0 hoặc 1, và thiết lập các cờ điều kiện tương ứng (phần đọc của chu kỳ bus) và thiết lập giá trị tín hiệu cờ bằng 1 (phần thay đổi của chu kỳ bus) và cất giữ giá trị tín hiệu cờ trở lại chỗ cũ trong bộ nhớ (phần ghi của chu kỳ bus).



Hình 3.26: Thực hiện phép cộng vô hướng ADDPS

Một dạng khác của lệnh nguyên tử là CAS (Compare And Swap), so sánh và trao đổi, được sử dụng để cập nhật dữ liệu chia sẻ trong một hệ thống đa xử lý. Khoản dữ liệu chia sẻ bị khóa khi bắt đầu thực hiện

lệnh CAS. CAS thực hiện so sánh nội dung của một thanh ghi so sánh (Dc) với khoản dữ liệu chia sẻ, thiết lập các cờ mã điều kiện tương ứng với kết quả so sánh, và khoản dữ liệu được cập nhật nhờ nội dung của thanh ghi bổ sung (Du). Sau đó, sự truy cập vào khoản dữ liệu chia sẻ được bỏ khóa, và sự thực hiện CAS hoàn thành. Sự thực hiện các lệnh như TAS và CAS trong các hệ thống đa xử lý đảm bảo khả năng đồng bộ cho các thao tác đa xử lý. Các lệnh này có trong họ vi xử lý Motorola M68000 như:

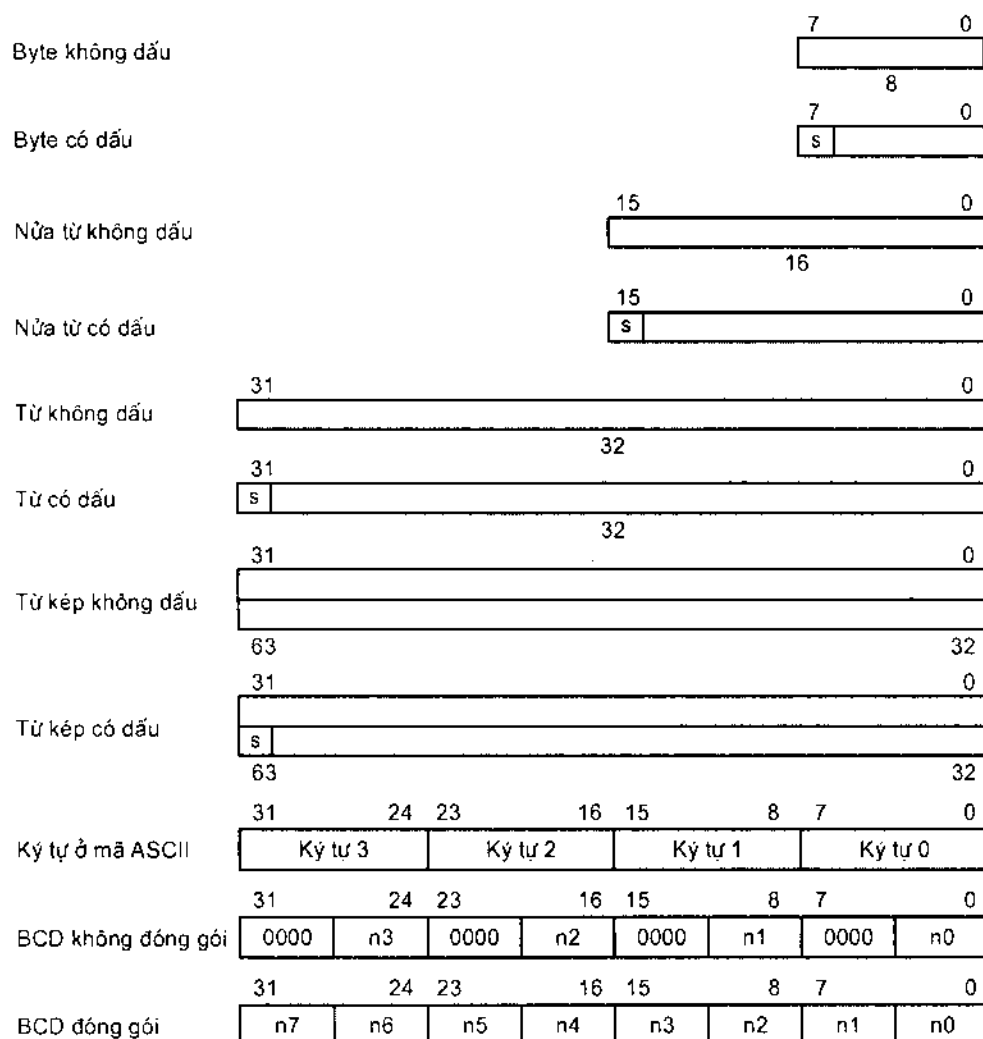
CAS Dc, Du, <EA>; cc $\leftarrow$ đích - Dc, nếu Z thì Du $\rightarrow$ đích, nếu không phải Z thì đích $\rightarrow$ Dc.

CAS2 Dc1:Dc2, Du1:Du2, (Rn1):(Rn2); như CAS nhưng thực hiện 2 lần. cc $\leftarrow$ đích1 - Dc1, nếu Z thì cc $\leftarrow$ đích2 - Dc2, nếu Z thì Du1 $\rightarrow$ đích1, Du2 $\rightarrow$ đích2, nếu không đích1 $\rightarrow$ Dc1, đích2 $\rightarrow$ Dc2.

Ngoài những nhóm lệnh kể trên có thể những lệnh khác, ví dụ như các lệnh của đơn vị quản lý bộ nhớ MMU, những lệnh hỗ trợ ngôn ngữ bậc cao (trong Intel 32-bit) như: BOUND, ENTER, LEAVE, SETcc byte.

3.5. CÁC DẠNG DỮ LIỆU

Các dạng dữ liệu (Data formats) là các số nguyên của các loại vi xử lý được phân biệt theo các số có dấu, không dấu, các ký tự mã ASCII, mã BCD (hệ 2-10) đóng gói hay không đóng gói. Phụ thuộc vào khả năng xử lý dữ liệu của các loại vi xử lý: 8-bit, 16-bit, 32-bit, 64-bit mà cách gọi về từ có khác nhau. Chẳng hạn, độ dài một từ trong vi xử lý 16-bit là 16-bit, nhưng trong vi xử lý 32-bit lại là 32-bit và những dữ liệu 16-bit gọi là nửa từ. Ngày nay chúng chủ yếu làm việc với các hệ thống vi xử lý 32-bit, và đa số chúng định dạng dữ liệu của 1 từ là 32-bit, song cũng có một số vi xử lý định dạng từ dữ liệu 16-bit, còn từ kép là 32-bit (hình 3.27).



Hình 3.27: Các dạng dữ liệu là các số nguyên

3.5.1. Không dấu và có dấu

B - Byte (8-bit), H - nửa từ (16-bit), W - từ (32-bit), D - từ kép (64-bit). Bit dấu là bit cao nhất (MSB).

Một từ dữ liệu 32-bit chứa 4 byte và lấy đến 4 địa chỉ các byte trong bộ nhớ. Có hai cách để dịch thứ tự các địa chỉ byte bên trong một từ:

+ Kết thúc nhỏ (little-endian): byte nhỏ nhất (LSB) có địa chỉ thấp nhất. Byte lớn nhất (MSB) có địa chỉ cao nhất. Trong một từ 32-bit các byte được địa chỉ theo thứ tự 3, 2, 1, 0, trong đó byte 0 là byte thấp nhất - là LSB, và byte 3 là byte cao nhất - là MSB.

+ Kết thúc lớn (Big-endian): byte nhỏ nhất (LSB) có địa chỉ cao nhất. Byte lớn nhất (MSB) có địa chỉ thấp nhất, nghĩa là các byte được địa chỉ theo thứ tự 0, 1, 2, 3. Byte 0 là MSB, byte 3 là LSB.

3.5.2. Các ký tự mã ASCII

Mỗi ký tự được mã hóa bằng 8-bit. Bốn ký tự ASCII có thể được đóng gói trong một từ 32-bit.

3.5.3. Dữ liệu ở mã BCD được đóng gói hay không đóng gói

Dữ liệu được biểu diễn bằng các số (digit) với 4-bit. Phân biệt dữ liệu ở mã BCD đóng gói và không đóng gói. Trong BCD đóng gói, hai số BCD trong một byte, 8 số có thể được đóng thành một từ 32-bit. Trong BCD không đóng gói, chỉ dùng 4-bit thấp trong từng byte để làm số BCD, 4-bit cao của từng byte không dùng tới và đều ghi giá trị 0.

Các số dấu phẩy động phân ra số chính xác đơn 32-bit (single-precision) và chính xác kép 64-bit (double precision) theo chuẩn IEEE 754-1985 (hình 3.28). Cũng có thể mở rộng độ chính xác đến 80-bit ở một số bộ vi xử lý. Số dấu phẩy động có bit lớn nhất (MSB) dùng làm dấu, giá trị dấu = 0, đó là số dương; giá trị dấu = 1, đó là số âm. Phân biệt các ký hiệu:

e - số thực nằm trong trường mũ; E - số mũ thực;

ne - số bit có trong trường mũ; m - phân định trị; S - dấu.

$$e = E + [2^{(ne-1)} - 1]; \quad E = e - [2^{(ne-1)} - 1];$$

Gọi $b = 2^{(ne-1)} - 1$ là Bias (hay là mũ chéo - biased exponent), như vậy b sẽ có giá trị như sau:

Số chính xác đơn ($ne = 8$): $b = 127$

Số chính xác kép ($ne = 11$): $b = 1023$

Số chính xác mở rộng (ne = 15): $b = 16383$

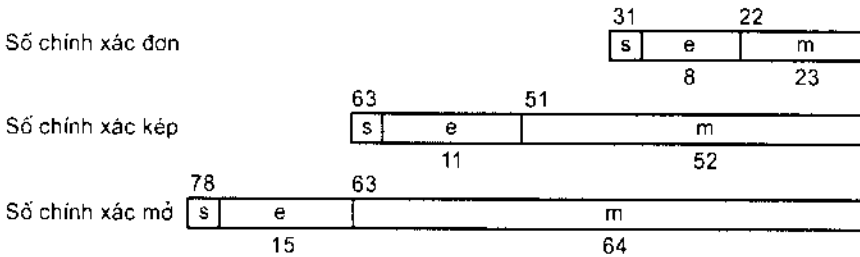
Số dấu phẩy động viết theo chuẩn IEEE là: $f = (-1)^s 1.m \times 2^{e-b}$.

Dải số dấu phẩy động xác định trong khoảng:

Số chính xác đơn: 10^{-38} đến 10^{38}

Số chính xác kép: 10^{-308} đến 10^{308}

Số chính xác mở rộng: 10^{-4932} đến 10^{4932}



Hình 3.28: Các dạng dữ liệu là các số dấu phẩy động

BÀI TẬP ÔN LUYỆN

1. Khi thực hiện CALL, thông tin gì được cất giữ vào ngăn xếp?
2. Vì sao lệnh CALL cần có 5 chu kỳ máy để thực hiện?
3. Chương trình con thường được kết thúc bằng lệnh gì?
4. Con trỏ ngăn xếp luôn trỏ vào đâu của ngăn xếp?
5. Sự khác nhau giữ lệnh JMP và RET?
6. Vẽ đồ thị xung thực hiện lệnh POP PSW?
7. Trong điều kiện nào thì cặp lệnh PUSH/POP được sử dụng trong chương trình con?
8. Trong hệ thống đa xử lý 8086, sự khác nhau như thế nào giữa các bus chung, bus cục bộ và bus riêng?
9. Tất cả các lệnh nhảy ngoại trừ JMP sử dụng:
 - a: Đánh địa chỉ gián tiếp;
 - b: Đánh địa chỉ trực tiếp;
 - c: Đánh địa chỉ gián tiếp thanh ghi;
 - d: Tất cả a, b, c.
10. Các lệnh IN, OUT và XLAT có một đặc điểm chung là:
 - a: Sử dụng thanh ghi AX như là một phần cho vận chuyển dữ liệu;
 - b: Chỉ giới hạn chuyển dữ liệu 8-bit;
 - c: Đều có thể làm việc hoặc với μ hoặc với bộ nhớ;
 - d: Không phải tất cả trên.
11. Lệnh REP cho phép:
 - a: Nhắc lại thực hiện một lệnh cho đến khi có ngắt hoặc chờ để nhận;
 - b: Thực hiện làm đầy bộ nhớ bằng một giá trị hằng số;
 - c: Thực hiện một phép chia nhị phân sử dụng lặp lại các phép trừ;
 - d: Thực hiện một chuỗi các lệnh cho đến khi sự giảm dần nội dung của thanh ghi CX đạt đến một điều kiện.

12. Khi nào thì các đường địa chỉ cao A19-A16 được dùng để đánh địa chỉ bộ nhớ, khi nào thì chúng được dùng làm các tín hiệu trạng thái S6, S5, S4, S3. Những tín hiệu này có tác dụng gì. Những tín hiệu này có phải chốt không. Nếu chốt thì bằng tín hiệu nào?
13. Khi được giải mã, các tín hiệu trạng thái S3#, S4# được sử dụng để báo cho thiết bị ngoại vi rằng:
 - a: Vi xử lý đang trong chế độ nào;
 - b: Trạng thái xếp hàng;
 - c: Trạng thái vi xử lý;
 - d: Thanh ghi đoạn nào đang được dùng.
14. 8086/8088 cần địa chỉ 20-bit vậy tại sao phải trộn lẫn các bit địa chỉ thấp nhất với dữ liệu (8-bit) và 4 bit địa chỉ cao với trạng thái.
15. Địa chỉ vật lý được xác định như thế nào khi sử dụng chế độ đánh địa chỉ chỉ số (indexed addressing)?
16. Trong hệ thống 8086/8088 địa chỉ vật lý được tính thế nào từ địa chỉ đoạn và giá trị offset (số gia, độ dịch)?
17. Vì sao IBM PC/XT 86 không phát triển bằng IBM PC/XT 88?
18. Vì sao cần phải có các bộ đệm bus dữ liệu và địa chỉ trong hệ thống 8086/8088?
19. 8087 thực hiện các lệnh gì. Vì sao cần có 8087?
20. Sự phối hợp thực hiện lệnh giữa 8086/8088 và 8087 như thế nào. Những tín hiệu nào tham gia vào đồng bộ quá trình phối hợp này. Vẽ sơ đồ và giải thích?
21. Nếu không có 8087 kết nối với 8086/8088 thì 8086/8088 thực hiện các lệnh dấu phẩy động như thế nào?
22. Bên trong tệp thanh ghi của 8087 dữ liệu được cất giữ và xử lý như thế nào?
23. Kết quả thực hiện lệnh trong 8087 được đưa về đâu, và như thế nào?
24. Giá trị dương lớn nhất là bao nhiêu có thể cất giữ trong 8087?

Chương 4

LẬP TRÌNH BẰNG HỢP NGỮ

4.1. KHÁI NIỆM VỀ CHƯƠNG TRÌNH HỢP NGỮ

4.1.1. Giới thiệu hợp ngữ (Assembly)

Các bộ vi xử lý chỉ làm việc với các chỉ thị có dạng nhị phân và đang ở trong bộ nhớ chính của hệ thống vi xử lý. Các chương trình nguồn ở các ngôn ngữ bậc cao như PASCAL, C, FORTRAN,... dễ hiểu và dễ viết cho người lập trình và muốn cho máy tính hiểu được và chạy được phải được dịch ra dạng nhị phân. Nhưng mỗi dòng lệnh trong chương trình nguồn có thể được dịch ra thành một số chỉ thị nhị phân, như vậy từ một chương trình nguồn ở ngôn ngữ bậc cao ngắn gọn có thể tạo ra một chương trình ở dạng nhị phân lớn chiếm nhiều không gian của bộ nhớ chính và chương trình được thực hiện chậm hơn. Do đó ngôn ngữ bậc cao không phù hợp để viết các chương trình hệ thống với yêu cầu chiếm ít dung lượng nhớ và tốc độ xử lý nhanh.

Một cách khác để viết chương trình nguồn, đó là dùng hợp ngữ (Assembly language). Hợp ngữ là ngôn ngữ lập trình cấp thấp. Bởi vì mỗi một dòng lệnh viết bằng hợp ngữ trong chương trình nguồn được biên dịch (gọi là hợp dịch - assembling) thành một chỉ thị dạng nhị phân của chính vi xử lý, và vi xử lý có thể thực hiện được. Hợp ngữ đảm bảo sự tương ứng một-một giữa một lệnh mã nguồn và một chỉ thị nhị phân, và cho phép can thiệp sâu trực tiếp vào các chức năng phần cứng của hệ thống vi xử lý như: các thanh ghi bên trong vi xử lý, các đơn vị điều khiển vào/ra ngoại vi, bộ nhớ,... Do đó, hợp ngữ là ngôn ngữ lập trình định hướng phần cứng. Chương trình được dịch từ mã nguồn hợp ngữ chiếm dung lượng nhớ nhỏ và thực hiện nhanh hơn gấp 2 và 3 lần so với các ngôn ngữ bậc cao như C, PASCAL. Do đó, hợp ngữ phù hợp để viết các chương trình hệ thống.

Tuy nhiên, để viết được chương trình bằng hợp ngữ cho một hệ thống vi xử lý, người lập trình cần phải hiểu biết về vi xử lý và tập lệnh của nó. Viết một chương trình hợp ngữ tối ưu không phải là đơn giản, bởi vì, để thực hiện một thao tác nào đó có thể dùng một số chỉ thị khác nhau, đặc biệt đối với các loại vi xử lý kiến trúc CISC thì với tập lệnh phức tạp, nhiều chế độ đánh địa chỉ. Những vi xử lý là vi điều khiển có tập lệnh gồm ít chỉ thị nên có thể dễ dàng cho viết chương trình. Ngoài ra, với hợp ngữ khó khăn hơn so với các ngôn ngữ bậc cao khác khi viết các giao diện thuận tiện cho người sử dụng và quản lý cơ sở dữ liệu.

Hiện nay đã có các công cụ và thư viện hỗ trợ viết các chương trình nguồn ở hợp ngữ trong môi trường Windows thuận tiện để tạo ra các cửa sổ giao diện, các thực đơn điều khiển cho người dùng.

4.1.2. Hợp ngữ cấp thấp LLA (Low Level Assembly) và cấp cao HLA (High Level Assembly)

Đối với chương trình nguồn viết ở hợp ngữ cấp thấp thì mỗi một lệnh của nó được dịch (hợp dịch) thành một lệnh máy (một chỉ dẫn) của vi xử lý. Nhưng một chương trình viết ở hợp ngữ cấp cao trước hết phải được dịch nhờ chương trình dịch (HLA compiler) để thành một chương trình ở hợp ngữ cấp thấp, sau đó nhờ hợp dịch mới trở thành chương trình đối tượng ở mã máy. Tiếp đến thực hiện liên kết chương trình đối tượng để thành chương trình máy chạy được.

HLA là một gói phần mềm gồm: phần mềm biên dịch (compiler) một chương trình hợp ngữ bậc cao thành chương trình ở hợp ngữ bậc thấp, thư viện chuẩn của HLA, và tập hợp các tệp INCLUDE cho thư viện chuẩn HLA. HLA là một ứng dụng cho các hệ điều hành LINUX và Windows.

Vì sao lại có HLA. Bởi vì, hợp ngữ cấp thấp không thuận tiện cho viết các giao diện thân thiện, quản lý cơ sở dữ liệu và những khả năng

ưu việt mà ngôn ngữ bậc cao có như: dễ viết, gắn với ngôn ngữ đời thường,... HLA đáp ứng được những tiêu chuẩn của ngôn ngữ bậc cao nhưng lại cho phép biên dịch chương trình viết ở HLA thành chương trình hợp ngữ bậc thấp. Đây là ưu điểm rất quan trọng giúp cho người sử dụng tiết kiệm thời gian lập trình mà vẫn tạo được những chương trình hợp ngữ cấp thấp hiệu quả. Ví dụ sau đây là một chương trình viết ở HLA có tên là HW.HLA (tên mở rộng .HLA chỉ chương trình nguồn viết ở HLA):

```
program HelloWorld;  
#include ("stdlib.hhf")  
begin HelloWorld;  
    stdout.put ("Hello, World of Assembly Language", nl);  
end HelloWorld;
```

Bằng chương trình biên dịch HLA thực hiện dịch chương trình HW.HLA thành chương trình hợp ngữ cấp thấp.

Ta nhận thấy một chương trình viết ở HLA thật đơn giản.

4.2. CÁCH TẠO VÀ CHẠY CHƯƠNG TRÌNH HỢP NGỮ

Để tạo một chương trình hợp ngữ và chạy nó trên máy tính PC thường phải thực hiện 4 bước lần lượt như sau:

1. Soạn thảo tệp chương trình nguồn ở hợp ngữ (source program file)
2. Hợp dịch tệp chương trình nguồn thành tệp đối tượng ở ngôn ngữ máy (object file)
3. Liên kết một hay nhiều tệp đối tượng tạo ra một tệp chương trình chạy được (run program file)
4. Chạy chương trình.

Chúng ta chỉ xét lập trình hợp ngữ cho các vi xử lý Intel hoặc tương thích với chúng.

4.2.1. Bước 1 - Soạn thảo tệp chương trình nguồn

Dùng một chương trình soạn thảo văn bản bất kỳ nào đó để soạn tệp chương trình nguồn theo qui định của một cấu trúc chương trình hợp ngữ. Đó là các trình soạn thảo trong Windows như NotePad, WordPad, và MS Word. Trong môi trường DOS có thể dùng ngay lệnh EDIT để soạn thảo. Có thể dùng ngay công cụ soạn thảo kèm theo chương trình Assembler, ví dụ, dùng QEDITOR (Quick Editor 2.8 hay 3.0) của Assembler MASM32 (Microsoft) làm việc trong môi trường Windows.

Tên của tệp chương trình nguồn hợp ngữ có phần mở rộng mặc định là .ASM. Trong hợp ngữ không phân biệt chữ hoa và chữ thường. Tệp chương trình nguồn hợp ngữ được lưu trên đĩa cứng hoặc đĩa mềm tùy ý nhưng nên trong thư mục và đường dẫn sao cho khi hợp dịch và liên kết có thể kết hợp được các thư viện và các macro mà trong chương trình nguồn phải liên kết tới.

4.2.2. Bước 2 - Hợp dịch tệp chương trình nguồn thành tệp đối tượng

Để hợp dịch tệp chương trình nguồn ra tệp đối tượng ngôn ngữ máy (object file) cần phải sử dụng chương trình hợp dịch Assembler. Thông dụng là TASM của Borlan hay Macro Assembler (MASM) của Microsoft.

Cần chú ý rằng, đối với các vi xử lý Intel 16-bit và 32-bit nên chọn các chương trình dịch MASM32 hoặc MASM 6.X. Cũng có thể chọn các phiên bản cho 16-bit và 32-bit của TASM.

Tệp đối tượng mặc định có phần mở rộng mặc định là .OBJ và tên trùng với tên tệp nguồn. Nếu muốn lấy tên khác thì viết tên cho tệp đối tượng nhưng phần mở rộng phải là .OBJ. Các tên mặc định được viết trong dấu ngoặc vuông [] trong cú pháp của dòng lệnh dịch.

Nếu dùng MASM32 (phiên bản 7 hay 8) thì lệnh ML trong thư mục masm32\bin là lệnh hợp dịch. Hệ chương trình MASM32 được nạp vào trong thư mục C:\masm32. Tạo thư mục masm32 là tùy chọn

khi thực hiện gỡ nén hệ chương trình MASM32.ZIP. Cần chú ý là phải thiết lập đường dẫn mặc định cho chương trình dịch bằng lệnh trong DOS:

```
PATH = %PATH%; C:\MASM32\BIN
```

Hoặc tạo dòng lệnh trong tệp C:\AUTOEXEC.BAT:

```
SET PATH = %PATH%; C:\MASM32\BIN.
```

Cho rằng chương trình nguồn có tên là LISTBOX.ASM có trong thư mục C:\MASM32\EXAMPLE\LISTBOX\ Khi đó ta viết lệnh:

```
C:\masm32\bin\ml /c /coff /Zi /Fl \masm32\example1\listbox\listbox.asm
```

Nếu dịch không có lỗi, màn hình hiển thị:

```
Microsoft (R) Macro Assembler Version 6.14.8444
```

```
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.
```

```
Assembling: \masm32\example1\listbox\listbox.asm
```

```
C:\
```

Ý nghĩa các khóa (switch):

/C tạo tệp đối tượng nhưng không thực hiện liên kết (linking)

/coff tạo ra tệp đối tượng ở dạng COFF - tương thích với tệp đối tượng khuôn dạng thông dụng và tạo được trong Windows NT

/Zi bổ sung thông tin gỡ rối (debug) tượng trưng cho tệp đối tượng

/Fl tạo tệp liệt kê danh sách (.LST).

Lệnh ML có tới 42 khóa có ý nghĩa khác nhau và là tùy chọn.

Trong những phiên bản chương trình dịch cũ, như MASM 5.1, ngoài tệp liệt kê danh sách (.LST) còn có tạo thêm tệp tham khảo chéo (cross reference file) có phần mở rộng là .XRF.

Tệp liệt kê danh sách (.LST) là một tệp văn bản với các dòng được đánh số hiển thị mã nguồn hợp ngữ bên cạnh với các thông tin

khác về chương trình nguồn. Nó có ích cho mục đích gỡ rối (debug) chương trình, bởi vì thông báo lỗi của assembler được đưa ra kèm theo số dòng bị lỗi.

Tập tham khảo chéo, CFR (trong TASM tương tự tạo ra tập tham khảo .XRF) là bảng liệt kê các tên xuất hiện trong chương trình và số thứ tự các dòng mà nó có mặt. Tập này cần thiết khi xác định các biến và các nhãn trong một chương trình lớn.

Tập đối tượng là một tập ở ngôn ngữ máy (gồm các chỉ dẫn) tuy nhiên nó chưa thể thực hiện được ngay, vì: nó chưa biết được sẽ được chứa ở vùng nào trong bộ nhớ chính để thực hiện, có thể có một số địa chỉ trong các lệnh máy chưa được điền vào, một số tên có thể chưa được định nghĩa trong chương trình lớn có thể phải tạo một vài tập, và có một thủ tục trong một tập này có thể có tham chiếu đến một tên được định nghĩa trong một tập khác.

4.2.3. Bước 3 - Liên kết một hay nhiều tập đối tượng tạo ra một tập chương trình chạy được

Dùng chương trình LINK trong thư mục \MASM32\BIN để thực hiện kết hợp một hay nhiều tập đối tượng thành một tập đối tượng chung, xác định các địa chỉ và các tên chưa được định nghĩa và tạo thành một tập chương trình chạy được với không gian nhớ (tương đối) được cấp phát, và tên có phần mở rộng là .EXE (nghĩa là executive).

```
C:\masm32\bin\link \masm32\example1\listbox\listbox.obj
```

Màn hình hiển thị trở lại nếu liên kết thành công:

Microsoft (R) Incremental Linker Version 5.12.8078

Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

```
C:\
```

Có thể tạo ra tập nguồn có cấu trúc để khi dịch và liên kết tạo ra tập chạy được có tên mở rộng, COM hoặc .EXE. Có sự khác biệt giữa tập .COM và .EXE (sẽ xét sau).

4.2.4. Bước 4 - Chạy chương trình

Để chạy chương trình, chỉ cần gõ tên chương trình mà không cần đưa vào phần mở rộng .EXE. Vì .EXE là mặc định.

C:\masm32\example1\listbox\listbox

Trên màn hình xuất hiện kết quả của chương trình chạy. LISTBOX là một chương trình ví dụ tạo hộp thực đơn danh mục. Nội dung của chương trình có trong phần phụ lục.

4.3. CÁC PHẦN CƠ BẢN CỦA HỢP NGỮ

Một chương trình nguồn viết ở hợp ngữ cấp thấp phải được viết theo qui định phù hợp cho hợp dịch. Các mã lệnh trong viết trong chương trình hợp ngữ không phân biệt chữ hoa và chữ thường. Qui định viết chương trình nguồn theo từng dòng lệnh, mỗi dòng lệnh gồm có 4 trường (cột), mỗi trường cách nhau ít nhất một ký tự trống hay TAB. Để cho dễ đọc nên có sự sắp xếp thẳng hàng theo các trường (không bắt buộc theo cú pháp). Các trường của dòng lệnh phải tuân thủ theo thứ tự như sau:

Tên Toán tử Toán hạng Lời bình

4.3.1. Trường tên

Trường tên được dùng để viết nhãn (Label) cho dòng lệnh, cho một biến, hay cho một thủ tục (procedure). Nhãn chính là con trỏ tượng trưng để tham chiếu đến một vị trí trong bộ nhớ (gọi là offset trong đoạn lệnh), nơi mà một chỉ thị, được mô tả bằng một từ viết tắt tiếng Anh gọi nhớ (Mnemonic), đang hiện diện. Nhãn có thể là nơi đến của một lệnh nhảy hay một lời gọi đến một chương trình con hay thủ tục. Nhãn có thể có độ dài từ 1 đến 31 ký tự. Các ký tự để viết nhãn có thể là chữ, chữ số, gạch dưới, ký tự dollar (\$), dấu hỏi (?), ký tự (@), Ký tự đầu tiên trong một nhãn không thể là chữ số (như thế là sai cú pháp). Đứng sau nhãn là dấu (:) như:

start:

```
PUSH    DS
XOR     AX, AX
```

Nhãn start: bắt đầu một thủ tục. Để chương trình dễ đọc hơn, chúng ta nên viết nhãn và dòng lệnh trên hai dòng khác nhau. Trình biên dịch không phân biệt nhãn chữ hoa hay chữ thường.

4.3.2. Trường toán tử

Trong trường toán tử có thể viết hai loại toán tử: các chỉ dẫn thuộc tập lệnh của vi xử lý và các chỉ dẫn giả (pseudo operation) phục vụ cho biên dịch.

- Các chỉ dẫn của vi xử lý phải được viết theo đúng cú pháp mà tập lệnh của vi xử lý qui định, ví dụ:

```
MOV     AH, 2
INT     21h
```

Trong đó MOV và INT là các toán tử còn AH, 2 và 21h là các toán hạng được viết trong trường toán hạng.

- Các toán tử giả (DIRECTIVE) là các chỉ dẫn giả dùng để điều khiển các hoạt động của hợp dịch Assembler. Mỗi một loại Assembler có tập toán tử giả kèm theo (xem phụ lục). Có thể gọi toán tử giả là chỉ dẫn của assembler. Nói chung các toán tử giả của các trình hợp dịch thường có các loại: toán tử định nghĩa các kiểu dữ liệu, các biến, định nghĩa các đoạn nhớ và kích thước đoạn nhớ, các thủ tục, các điều kiện,...

Ví dụ 4.1:

```
blank    EQU    20h ; blank là tên tương trưng của ký tự cách trống
                        giữa hai từ, nó có giá trị ASCII bằng 20h, do
                        đó toán tử giả EQU định nghĩa blank = 20h
MOV      AL, blank ; lệnh MOV chuyển giá trị 20h vào AL.
```

Định nghĩa đoạn dữ liệu chứa dòng hiển thị Hello, world:

```
data          SEGMENT
th_bao        DB    'Hello, world.$'
data          ENDS
```

Đoạn dữ liệu (data) được định nghĩa bằng cặp toán tử giả SEGMENT ... ENDS, trong đó chứa chuỗi ký tự để hiển thị Hello, world và kết thúc bằng \$. Toán tử giả định nghĩa các byte ký tự hiển thị

4.3.3. Trường toán hạng

Mỗi một chỉ dẫn của vi xử lý có thể có tác động lên các toán hạng. Một chỉ dẫn có thể có một, hai, hoặc không có toán hạng. Trong trường toán hạng, hai toán hạng được viết cách nhau bằng dấu phẩy. Đối với các chỉ dẫn trong tập lệnh của vi xử lý Intel thì toán hạng nguồn ở sau dấu phẩy, toán hạng đích đứng trước dấu phẩy. Các chỉ dẫn của assembler có thể có thêm thông tin trong trường toán hạng.

Ví dụ 4.2:

```
MOV    AX, data
MOV    DS, AX
MOV    AH, 9
MOV    DX, OFFSET th_bao
INT     21h
```

Trong đoạn chương trình này, 2 dòng đầu thực hiện tạo địa chỉ cơ sở đoạn dữ liệu (data), đã được định nghĩa trước đó, vào thanh ghi DS. Lệnh thứ 3 tạo mã chức năng 9 vào thanh ghi AH cho ngắt chức năng DOS (INT 21h). Lệnh thứ 4 có chỉ dẫn giả OFFSET định nghĩa độ dịch của địa chỉ th_bao (chứa chuỗi ký tự phải hiển thị) so với địa chỉ cơ sở đoạn. Cặp thanh ghi địa chỉ thực 20-bit của đoạn nhớ chứa chuỗi ký tự hiển thị là DS:DX. Lệnh thứ 5 là ngắt chức năng DOS đưa ra hiển thị màn hình một chuỗi ký tự kết thúc bằng dấu S.

4.3.4. Trường lời bình

Đứng trước trường lời bình bắt buộc phải là dấu (;). Trường này chỉ dành riêng cho người lập trình, nó không có ý nghĩa đối với assembler, không tác động gì đối với máy tính. Nó chỉ để người lập trình viết các giải thích về đoạn chương trình, dòng lệnh, hoặc là nội dung nào đó để dễ đọc, dễ hiểu, dễ kiểm tra chương trình. Viết lời bình dễ hiểu, ngắn gọn là một vấn đề mà người lập trình cần phải chú ý. Nếu lời bình dài phải xuống dòng thì mở đầu của dòng phải có dấu(;). Lời bình phải sao cho không thừa đối với câu lệnh đã quá rõ ràng rồi. Chẳng hạn:

Không nên viết lời bình:

MOV AH, message ; nạp vào byte cao bản tin

Nhưng có thể viết lời bình:

MOV AH, message ; nạp giá trị ban đầu của bản tin = 1101 1001.

4.3.5. Dữ liệu dùng trong chương trình hợp ngữ

Vi xử lý chỉ làm việc với các dữ liệu ở dạng nhị phân, do đó trình biên dịch assembler phải thực hiện chuyển đổi tất cả các dạng dữ liệu thành dạng nhị phân, mặc dù trong chương trình nguồn ta có thể biểu diễn dữ liệu dưới các dạng là các số hay các ký tự. Các số có thể là số nguyên hay số dấu phẩy động.

a) Các số

Các số trong hợp ngữ có thể là nguyên và số dấu phẩy động có dấu, và không có dấu. Chúng có thể được biểu diễn ở các hệ đếm nhị phân, bát phân, thập phân, và hex bằng cách viết thêm các cơ số đếm ở cuối số.

Để phân biệt, trong hợp ngữ các số ở hệ nhị phân được viết kèm theo cơ số B (b) (nếu cơ số mặc định là hệ thập phân) hoặc Y (y) ở cuối số:

MOV AH, 11011001b ; nạp vào AH giá trị nhị phân 11011001

MOV AH, 11011001y

Các số ở hệ thập phân có thể không kèm theo hoặc kèm theo cơ số D (d) (nếu cơ số mặc định là hệ thập phân), hoặc là T (t) ở cuối số:

MOV CX, 16; hoặc

MOV AX, 16d

MOV AX, 16t

Các số ở hệ bát phân được viết kèm theo cơ số O (o) (nếu cơ số mặc định là hệ thập phân) hoặc Q (q) ở cuối số:

MOV AL, 75o

MOV CX, 65q

Các số hex có kết thúc bằng cơ số H (h):

MOV AX, 8833h ; nạp 88 và 33 như là các số BCD đóng gói

MOV DX, 0FFFEh ; nếu là ký tự thì phải viết số 0 trước số hex.

Các giá trị số có thể là âm (có dấu - đứng trước) hay dương (có dấu + đứng trước):

mem8 SBYTE -5 ; định nghĩa một byte có dấu = -5

mem16 WORD +5 ; định nghĩa một từ có dấu = +5

mem32 SDWORD -5 ; định nghĩa một từ kép có dấu = -5

Khi đó các lệnh sau đây cho các kết quả:

MOV AL, mem8 ; nạp giá trị 8-bit là -5 (FBh)

CBW ; chuyển thành 16-bit -5 (FFFBh) trong AX

MOV AX, mem16 ; nạp 16-bit +5

CWD ; chuyển thành 32-bit +5 (0000:0005h)
; trong DX:AX

MOV AX, mem16 ; nạp 16-bit +5

CWDE ; chuyển thành 32-bit +5 (00000005h)
; trong EAX

```
MOV    EAX, mem32 ; nạp 32-bit -5 (FFFFFFFFBh)
CDQ                      ; chuyển thành 64-bit -5
                      ; (FFFFFFFF:FFFFFFFFBh) trong EDX:EAX
```

b) Các ký tự

Các ký tự và chuỗi ký tự phải được viết trong các dấu nháy đơn hay dấu nháy kép. Ví dụ: "Hello, world", 'Hello, world', "C", hoặc 'C'. Assembler sẽ dịch các ký tự đứng trong các dấu nháy đơn hay kép ra mã ASCII. Như vậy, người lập trình không cần nhớ mã số ASCII của các ký tự, khi cần sử dụng chúng trong các trường hợp ví dụ như đưa ra hiển thị, kiểm tra so sánh khi nhận từ bàn phím, hay từ đường truyền thông.

4.3.6. Các biến nguyên

Cũng như trong ngôn ngữ bậc cao, hợp ngữ cấp thấp cũng có các biến. Các biến là một dạng dữ liệu. Có các biến nguyên (kiểu byte, từ, từ kép, 4 từ, byte có dấu, từ có dấu), các biến kiểu mảng, và để định nghĩa các biến trong assembler có các toán tử giả.

a) Các biến nguyên kiểu byte, từ

Có các toán tử giả chỉ định kích thước của biến nguyên và giải giá trị như sau:

```
BYTE, DB    ; định nghĩa byte (8-bit) không có dấu giá trị từ 0
              ; đến 255
SBYTE       ; định nghĩa byte có dấu giá trị từ -128 đến +127
WORD, DW    ; định nghĩa từ (16-bit) không có dấu giá trị từ 0
              ; đến 65535 (64 kbyte)
SWORD      ; định nghĩa từ có dấu giá trị từ -32768 đến +32767
DWORD, DD   ; định nghĩa từ kép (32-bit) không có dấu giá trị từ 0
              ; đến 4294967295 (4 MB)
SDWORD     ; định nghĩa từ kép có dấu giá trị từ -2147483648
              ; đến + 2147483647
```

- FWORD, DF** ; định nghĩa từ xa (farword) 48-bit (6 byte) và sử dụng như là các biến trữ trong các vi xử lý 32-bit.
- QWORD, DQ** ; định nghĩa các biến nguyên 8-byte (64-bit) sử dụng trong các lệnh của đồng xử lý (từ 8087)
- TBYTE, DT** ; định nghĩa các biến nguyên 10 byte (80-bit).

Ví dụ 4.3:

long	DWORD	4294967295	
longnum	SDWORD	-2147433648	
empty	QWORD	?	; khởi tạo 8 byte trống liên tục
expression	WORD	4*3	; khởi tạo từ giá trị 12
tb	TBYTE	2345t	; khởi tạo số nhị phân dài ; 10 byte

Định nghĩa các biến cần phải được viết ở đầu chương trình nguồn. Một khi đã định nghĩa các biến, ta có thể sử dụng chúng để sao chép, vận chuyển, và thực hiện các phép tính. Và để thao tác với chính dữ liệu với kích thước đã được định trước cần phải dùng tới toán tử giả PTR. Cú pháp của PTR là:

type PTR expression

trong đó: type là kiểu dữ liệu, expression là biểu thức.

Có thể dùng PTR để truy cập phần cao của biến kích thước là từ kép:

.DATA

num DWORD 0

.CODE

MOV AX, WORD PTR num[0] ; nạp giá trị kích thước là
; một từ từ một biến kích
; thước từ kép

MOV DX, WORD PTR num[2] ;

b) Các biến nguyên kiểu mảng và chuỗi

Mảng (Array) là một tập hợp các biến có kích thước và kiểu giống nhau. Mỗi một biến là một phần tử (element).

Chuỗi (String) là một mảng gồm toàn các ký tự. Ví dụ "ABC" là một chuỗi các ký tự, mỗi ký tự là một phần tử. Có thể truy cập tới các phần tử trong mảng hoặc trong chuỗi theo địa chỉ tương đối với phần tử đầu tiên. Để làm việc với mảng và chuỗi trước hết chúng cần phải được định nghĩa ở đầu chương trình nguồn.

Ví dụ 4.4:

warray	WORD	1,2,3,4 ; khởi tạo một mảng gồm 4 từ với ; các giá trị của từng từ tương ứng ; là 1, 2, 3, 4
xarray	DWORD	0FFFFFFFh, 789ABCDEh ; khởi tạo mảng gồm 2 từ kép với các ; giá trị tương ứng
big	BYTE	21, 22, 23, 24, 25, 26, 27, 28 ; khởi tạo mảng gồm 8 byte với các ; giá trị tương ứng từng byte.
wstr	WORD	"OK" ; khởi tạo chuỗi ký tự OK
dstr	DWORD	"DATA"; khởi tạo chuỗi ký tự DATA

Một mảng có thể được phân đoạn thành một số dòng logic khi nó quá lớn về số lượng phần tử để viết trên một dòng:

var1	BYTE	10, 20, 30
	BYTE	40, 50, 60
	BYTE	70, 80, 90

Trong một mảng có thể gồm cả chuỗi ký tự, và cả giá trị số:

message1	DB	'Hello my love', 0Ah, 0Dh, 'S'
----------	----	--------------------------------

Có thể định nghĩa mảng bằng toán tử giả DUP. Toán tử giả DUP làm việc với bất kỳ toán tử giả nào chuyên về định nghĩa dữ liệu. Cú pháp của toán tử DUP là:

count DUP (initialvalue[, initialvalue]...)

Trong đó: count là giá trị số lần lặp lại tất cả các giá trị bên trong cặp dấu ngoặc đơn (). Initialvalue là giá trị nguyên, hằng số ký tự, hoặc toán tử DUP khác, và luôn phải xuất hiện bên trong dấu ngoặc đơn ().

Ví dụ 4.5:

barray	BYTE	5 DUP (1) ; khởi tạo giá trị 1 cho mảng 5 byte.
array	DWORD	10 DUP (1) ; khởi tạo 10 từ kép có giá trị ban ; đầu = 1.
buffer	BYTE	256 DUP (?) ; tạo bộ đệm gồm 256 byte
masks	BYTE	20 DUP (040h, 020h, 04h, 02h) ; khởi tạo bộ đệm 80 byte với các ; đánh dấu bit
three_d	DWORD	5 DUP (5 DUP (5 DUP (0))) ; 125 từ kép được khởi tạo giá trị 0

Mỗi một phần tử của mảng được tham chiếu nhờ chỉ số, bắt đầu bằng 0. Chỉ số của mảng được ghi trong dấu ngoặc vuông [] đứng sau tên mảng. Điều chú ý rằng trong hợp ngữ cấp thấp các chỉ số khác so với các chỉ số trong các ngôn ngữ lập trình cấp cao, mà trong đó chỉ số luôn tương ứng với vị trí của phần tử trong mảng. Ví dụ, trong ngôn ngữ lập trình C, array[9], tham chiếu đến phần tử thứ 10 của mảng tên là array, bất chấp phần tử có kích thước là một byte hay nhiều byte. Trong hợp ngữ cấp thấp chỉ số của một phần tử tham chiếu đến số lượng các byte giữa phần tử đó và bắt đầu của mảng. Sự khác biệt này với ngôn ngữ bậc cao có thể bị loại bỏ đối với các mảng gồm các phần tử có kích thước là 1 byte, vì khi đó vị trí mỗi phần tử trùng với chỉ số của nó. Ví dụ, trong hợp ngữ:

prime	BYTE	1, 3, 5, 7, 11, 13, 17
-------	------	------------------------

cho giá trị 1 của phần tử prime[0], giá trị 3 của phần tử prime[1],... Trong các mảng gồm các phần tử kích thước lớn hơn 1 byte, thì chỉ số (ngoại trừ chỉ số 0) không tương ứng với vị trí của phần tử. Trong trường hợp đó, cần phải nhân giá trị vị trí của phần tử với kích thước của phần tử để xác định giá trị chỉ số. Ví dụ:

wprime WORD 1, 3, 5, 7, 11, 13, 17

thì wprime[4] thể hiện phần tử thứ ba (5) là 4 byte tính từ bắt đầu mảng. Tương tự, wprime[6] thể hiện phần tử thứ tư (7) và wprime[10] thể hiện phần tử thứ sáu (13). Độ dịch (offset) cần thiết để tham chiếu đến một phần tử của mảng có thể được tính bằng công thức như sau cho phần tử thứ n:

array[(n-1)*kích thước của phần tử]

Khi áp dụng cho các mảng, các toán tử LENGTHOF, SIZEOF, và TYPE trả thông tin về độ dài và kích thước của mảng và về kiểu của các giá trị khởi tạo. Toán tử giả TYPE trả về kích thước của các phần tử của mảng.

Ví dụ 4.6:

array	WORD	40 DUP (5)	
larray	EQU	LENGTHOF	array ; 40 phần tử
sarray	EQU	SIZEOF	array ; 80 byte
tarray	EQU	TYPE	array ; một phần tử ; gồm 2 byte
num	DWORD	4, 5, 6, 7, 8, 9, 10, 11	
lnum	EQU	LENGTHOF	num ; 8 phần tử
snum	EQU	SIZEOF	num ; 32 byte
tnum	EQU	TYPE	num ; một phần tử ; 4 byte
warray	WORD	40 DUP (40 DUP (5))	
len	EQU	LENGTHOF	warray ; 1600 phần tử

siz	EQU	SIZEOF	warray ; 32 byte
typ	EQU	TYPE	warray ; một phần tử ; 2 byte.

Tương tự như mảng, chuỗi cũng có thể phân thành các dòng, nhưng mỗi dòng phải được cách bằng dấu phẩy:

```
str1    BYTE    "This is a long string that does not",
           "fit on one editor line"
```

Cũng có thể có các con trỏ của chuỗi:

```
PBYTE TYPEDEF  PTR  BYTE
        .DATA
msg1    BYTE    "Operation completed successfully."
msg2    BYTE    "Unknown command"
msg3    BYTE    "File not found"
prmsg   PBYTE    msg1 ; prmsg là một mảng, và
        PBYTE    msg2 ; PBYTE là con trỏ đến các
                        ; chuỗi msg1, msg2, msg3
        PBYTE    msg3 ;
```

Bởi vì các chuỗi cũng là các mảng gồm các phần tử có kích thước 1 byte, các toán tử giả LENGTHOF, SIZEOF, và TYPE cũng được dùng tương tự như trong mảng:

msg	BYTE	"This string extends", "over three lines"	
lmsg	EQU	LENGTHOF	msg ; 37 phần tử
smsg	EQU	SIZEOF	msg ; 37 byte
tmsg	EQU	TYPE	msg ; một phần tử kích ; thước 1 byte.

Trong tập lệnh của vi xử lý Intel có các xử lý chuỗi với mã gọi nhớ có đuôi là S, ví dụ: MOVS, STOS, CMPS, LODS, SCAS,... Và các

lệnh này được thực hiện lặp lại nhờ các lệnh REP, REPE (hoặc REPZ), và REPNE (hoặc REPNZ) khi tất cả các phần tử trong chuỗi chưa được xử lý xong.

Các phần tử của mảng và chuỗi có thể được lần lượt xử lý từ đầu mảng (chuỗi) đến cuối theo chỉ số tăng dần. Khi đó thanh ghi đếm số phần tử CX tham gia vào các lệnh xử lý mảng (chuỗi) phải có giá trị ban đầu = 0 và cờ D (hướng) được xoá ban đầu = 0 (nhờ lệnh CLD). Cặp các thanh ghi DS:SI và DS:DI chứa địa chỉ đầu của mảng (chuỗi). Trong quá trình xử lý chuỗi nội dung của SI và DI được tự động tăng lên 1 (nếu kích thước phần tử là 1 byte), lên 2 (nếu kích thước phần tử là từ). Ngược lại, có thể xử lý từ phần tử cuối của mảng (chuỗi) về đầu mảng. Khi đó cờ D phải thiết lập = 1 (nhờ lệnh STD), CX có giá trị ban đầu bằng số lượng phần tử trong mảng (chuỗi), các cặp DS:SI và DS:DI chứa địa chỉ của phần tử cuối của mảng (chuỗi) và trong quá trình xử lý mảng (chuỗi) chúng được tự động giảm để trở tới phần tử tiếp theo ngược lên về đầu mảng (chuỗi).

Ví dụ 4.7: dùng lệnh MOVS để sao chép dữ liệu từ một vùng nhớ đến vùng nhớ khác:

	.MODEL	SMALL	
	.DATA		
source	BYTE	10 DUP ('123456789')	
destin	BYTE	100 DUP (?)	
	.CODE		
MOV	AX, @data		; nạp cùng đoạn
MOV	DS, AX		; tạo địa chỉ cơ sở cho đoạn
			; dữ liệu
MOV	ES, AX		; tạo địa chỉ cơ sở cho đoạn
			; dữ liệu mở rộng (cùng đoạn
			; DS)

CLD		; cờ D = 0, số đếm theo chiều ; tăng
MOV	CX, LENGTHOF source	; thiết lập số đếm = 100
MOV	SI, OFFSET source	; nạp địa chỉ của nguồn
MOV	DI, OFFSET destin	; nạp địa chỉ của đích
REP	MOVSB	; chuyển 100 byte.

c) Các hằng số nguyên và các biểu thức hằng số

Một hằng số là một chuỗi liên tục các chữ số trong một hệ đếm.

Ví dụ:

```
MOV  AX, 25
MOV  BX, 0B3h
```

Các số 25 và 0B3h là các hằng số nguyên. Các hằng số được viết kèm theo các cơ số của hệ đếm.

Các hằng số nguyên được gán tên bằng toán tử giả EQU, và phải đặt ở đầu chương trình. Cú pháp là:

```
symbol EQU expression
```

Trường symbol là một tên duy nhất bất kỳ những không trùng với các tên mà trình assembler đã chọn.

Trường expression có thể một số nguyên, một biểu thức hằng số, một hằng số là chuỗi các ký tự, hoặc một biểu thức xác định địa chỉ. Các hằng được gán tên tượng trưng nhằm đơn giản khi sử dụng chúng trong chương trình, ví dụ như khi phải thay đổi giá trị hằng số, chỉ cần thay đổi giá trị gán cho tên tượng trưng ở dòng lệnh EQU. Hằng số có tên không được dành chỗ trong bộ nhớ:

column	EQU	80
row	EQU	25

screen	EQU	column*row	; hằng số là một biểu ; thức các hằng số.
line	EQU	row	
	.DATA		
	...		
	.CODE		
	...		
	MOV	CX, column	
	MOV	BX, line	

4.3.7. Các số thực (real)

Các chip đồng xử lý 8087, 80287, 80387, và các đơn vị FPU trong các chip 32-bit hiện nay thực hiện tính toán với các số dấu phẩy động (số thực).

Để định nghĩa các hằng số thực có thể sử dụng các toán tử giả REAL4, REAL8, và REAL10:

REAL4 xác định các số thực ngắn (32 bit)

REAL8 xác định các số thực dài (64 bit)

REAL10 xác định các số thực dài 10 byte (80 bit) và các số BCD

Khoảng xác định của các số thực như sau:

Số thực ngắn: 32 bit, 7-8 digit, giá trị từ $1,18 \times 10^{-38}$ đến $3,40 \times 10^{38}$

Số thực dài: 64 bit, 15-16 digit, giá trị từ $2,23 \times 10^{-308}$ đến $1,79 \times 10^{308}$

Số thực dài 10 byte: 80-bit, 19 digit, giá trị từ $3,37 \times 10^{-4932}$ đến $1,18 \times 10^{4932}$.

Các trình hợp dịch MASM phiên bản trước 6.0 chỉ có các toán tử giả DD, DQ và DT có thể định nghĩa các số thực. MASM 6.1 vẫn có các toán tử này, nhưng đúng hơn là để định nghĩa các biến nguyên chứ không phải là các biến thực.

Có thể định nghĩa các hằng số thực ở hệ thập phân hoặc hex trong khuôn dạng như sau:

$[[+|-]]integer[[fraction]][[E[+|-]]exponent]]$

Các số thực ở hệ hex phải có chữ R (r) ở cuối số.

Ví dụ 4.8:

; Các số thực ở hệ thập phân

short REAL4 25.53 ; theo định dạng của IEEE

double REAL8 2.523E1

tenbyte REAL10 2523.0E-2

; Các số thực ở hệ hex

ieeeshort REAL4 3F800000R

; 1.0 như chuẩn IEEE ngắn

ieeedouble REAL8 3F80000000000000R

; 1.0 theo chuẩn IEEE dài

temporary REAL10 3FFF8000000000000000R

; 1.0 theo số thực 10 byte.

4.3.8. Các số BCD

Các số BCD cho phép tính toán với các số lớn và làm tròn không có lỗi. Số nguyên BCD có thể được biểu diễn với bất kỳ độ chính xác nào. Ví dụ chip đồng xử lý 8087 có các số BCD giá trị trong khoảng $\pm 999.999.999.999.999.999$.

Phân biệt các số BCD không đóng gói và các số BCD đóng gói

Các số BCD không đóng gói là các byte chứa một chữ số (digit) mười trong 4 bit thấp của từng byte. Các số BCD đóng gói là các byte chứa 2 chữ số mười: một chữ số chiếm 4 bit cao và một chữ số chiếm 4 bit thấp của byte. Chữ số bên trái nhất chứa dấu (1 là số âm, và 0 là số dương). Các chữ số BCD đóng gói trong 8087 có độ dài tới 18 chữ số, mỗi chữ số 4 bit, 2 chữ số trong 1 byte.

Toán tử TBYTE có thể dùng để định nghĩa các hằng số nguyên và hằng số BCD đóng gói:

```
pos1    TBYTE    1234567890
                        ; định nghĩa như 00000000001234567890h
neg1    TBYTE    -1234567890
                        ; định nghĩa như 80000000001234567890h
```

Toán tử BYTE có thể dùng để định nghĩa các hằng số nguyên và hằng số BCD không đóng gói:

```
unpackedr  BYTE    1, 5, 8, 2, 5, 2, 9
                        ; khởi tạo giá trị 9 252 851
unpackedf  BYTE    9, 2, 5, 2, 8, 5, 1
                        ; khởi tạo giá trị 9 252 851
```

Hai dòng này sắp xếp các chữ số BCD xuôi và ngược phụ thuộc vào các chúng ta thực hiện tính toán các số này như thế nào.

Trong tập lệnh của vi xử lý Intel có các lệnh thao tác với các số BCD không đóng gói như AAA, AAS, AAM, AAD và các lệnh với các số BCD đóng gói như: DAA, DAS.

4.3.9. Các cấu trúc và các liên hợp

Một cấu trúc (structure) là một nhóm các kiểu dữ liệu và các biến khác nhau mà có thể được truy cập đến như là một liên hợp (union) hoặc đến bất kỳ một thành phần nào của nó. Cấu trúc có các trường, và các trường có thể có các kích thước và kiểu dữ liệu khác nhau.

Các liên hợp tương tự như cấu trúc, nhưng các trường của liên hợp lẫn lộn nhau trong bộ nhớ. Liên hợp cho phép xác định các khuôn dạng dữ liệu khác nhau cho cùng một không gian nhớ. Các liên hợp có thể cất giữ các loại dữ liệu khác nhau phụ thuộc vào hoàn cảnh cụ thể. Chúng cũng có thể cất giữ dữ liệu ở một loại và đọc ra ở một loại khác.

Mỗi một trường của cấu trúc có một độ dịch (offset) tương đối với byte đầu tiên của cấu trúc. Còn tất cả các trường của liên hợp lại

bắt đầu từ một độ dịch chung. Kích thước của cấu trúc là tổng của số các thành phần của nó, còn kích thước của liên hợp là độ dài của trường dài nhất.

Cấu trúc xác định trong hợp ngữ tương tự như struct trong ngôn ngữ C, như structure trong FORTRAN, và như một record (bản ghi) trong PASCAL. Các liên hợp của hợp ngữ tương tự như các liên hợp trong C, FORTRAN, và các bản ghi của PASCAL.

Để dùng được cấu trúc và liên hợp phải thực hiện từng bước như sau:

1. Khai báo kiểu cấu trúc (hoặc liên hợp)
2. Định nghĩa một hay một số biến cấu trúc (liên hợp) có các kiểu trên
3. Tham chiếu các trường trực tiếp hoặc gián tiếp bằng toán tử của trường.

a) Khai báo kiểu cấu trúc và liên hợp

Khi khai báo kiểu cấu trúc hay liên hợp là tạo ra một tạm thời cho dữ liệu. Tạm thời tuyên bố các kích thước, và có thể cả các giá trị ban đầu trong cấu trúc hoặc liên hợp, nhưng không phân phối bộ nhớ.

Từ khóa STRUCT đánh dấu sự bắt đầu của khai báo kiểu cho cấu trúc (chú ý là STRUCT và STRUC là tương đương nhau). Từ khóa UNION đánh dấu sự bắt đầu của khai báo kiểu liên hợp. Cú pháp như sau:

```
name{ STRUCT|UNION }[[alignment]][[,NOUNIQUE]]
fielddeclarations
name ENDS
```

Trường khai báo (fielddeclaration) là một chuỗi các biến khai báo. Có thể khai báo các giá trị ban đầu mặc định riêng biệt hoặc với toán tử giả DUP.

Khi khởi tạo các trường của một kiểu liên hợp, thì kiểu và giá trị của trường đầu tiên là kiểu và giá trị mặc định của liên hợp đó. Ví dụ, khai báo một liên hợp, kiểu mặc định của liên hợp là DWORD:

DWD	UNION	
d	DWORD	00FFh
w	WORD	?
b	BYTE	?
DWD	ENDS	

Trường tên (name) phải có các tên là duy nhất bên trong các mức tiếp theo bởi vì chúng thể hiện độ dịch từ bắt đầu của cấu trúc (hoặc liên hợp) đến trường tương ứng.

Truy cập dữ liệu trên các trường có sắp xếp (alignment) sẽ nhanh hơn so với các trường không sắp xếp của cấu trúc. Sự sắp xếp đảm bảo tiết kiệm chi phí bộ nhớ và tăng tốc độ truy cập dữ liệu. Đặc tính này có được trong các vi xử lý Intel 16-bit và 32-bit, nhưng không có trong vi xử lý Intel 8-bit 8088. Mỗi một trường trong cấu trúc có một độ dịch (offset) tương đối với 0. Nếu ta xác định một sự sắp xếp trong định nghĩa cấu trúc (hoặc với chọn lựa /Zpn), thì độ dịch đối với từng trường có thể được thay đổi nhờ chính sắp xếp đó (hoặc n). Giá trị sắp xếp (alignment value) là 1, 2, và 4. Giá trị mặc định là 1. Mỗi một trường được sắp xếp hoặc theo kích thước trường hoặc theo giá trị sắp xếp. Nếu kích thước trường tính theo byte lớn hơn giá trị sắp xếp thì trường đó được đệm thêm sao cho độ dịch của nó chia hết cho giá trị sắp xếp. Nếu không, trường được đệm thêm sao cho độ dịch của nó chia hết cho kích thước của chính trường đó. Sự đệm thêm vào trường bằng các số 0 phía trước trường. Kích thước của cấu trúc cũng phải chia hết cho giá trị sắp xếp của cấu trúc, như vậy các số 0 phải được bổ sung vào cuối của cấu trúc. Sự đệm thêm phải đạt được giá trị độ dịch cho trường trước khi trường đó được phân bổ.

Nếu không có sắp xếp hoặc sử dụng chọn lựa /Zp thì độ dịch được tăng theo kích thước của từng toán tử giả. Nó giống như là có giá trị sắp xếp bằng 1 (mặc định).

Ví dụ 4.9: xác định các độ dịch của trường:

```

STUDENT2  STRUCT    2 ; giá trị sắp xếp là 2
score      WORD      1 ; offset = 0
id         BYTE      2 ; offset = 2 (1 byte được đệm thêm)
year       DWORD     3 ; offset = 4
sname      BYTE      4 ; offset = 8 (1 byte được đệm thêm)
STUDENT2  ENDS

```

Trong ví dụ trên kích thước trường year (4 byte) lớn hơn giá trị sắp xếp nên cần 1 byte được đệm thêm cuối trường byte đầu tiên (id) để độ dịch của trường year chia hết cho giá trị sắp xếp. Do đó độ dịch của trường year bằng 4. Nếu không đệm thêm 1 byte thì trường year có độ dịch bằng 3, không chia hết cho 2 (giá trị sắp xếp). Các trường id và sname không cần bổ sung vì chúng có kích thước 1 byte. Tính đến hết trường sname thì cấu trúc có kích thước là 9 byte (score = 2, id = 1, year = 5, sname = 1), và không chia hết cho 2. Để đảm bảo kích thước của cấu trúc phải chia hết cho giá trị sắp xếp phải bổ sung thêm 1 byte trước kết thúc cấu trúc STUDENT2 (làm cho kích thước của cấu trúc bằng 10 byte).

Ví dụ 4.10:

```

STUDENT4  STRUCT    4 ; giá trị sắp xếp bằng 4
sname      BYTE      1 ; offset = 0 (1 byte được đệm thêm)
score      WORD     10 DUP (100) ; offset = 2
year       BYTE      2 ; offset = 22 (1 byte được
                        ; đệm thêm)
id         DWORD     3 ; offset = 24
STUDENT4  ENDS

```

Đặc điểm sắp xếp của cấu trúc trong hợp ngữ tương thích với ngôn ngữ C, và có thể dùng tiện ích H2INC trong MASM 6.1 để chuyển đổi các cấu trúc của C sang hợp ngữ.

Các toán tử ALIGN, EVEN, và ORG có thể thay đổi độ dịch của các trường trong khai báo cấu trúc.

b) Định nghĩa các biến cấu trúc và liên hợp

Cú pháp định nghĩa các biến cấu trúc hoặc liên hợp như sau:

```
[[name]] typename <[[initializer],[,initializer]]...]>
```

```
[[name]] typename {[[initializer]{[,initializer]]...}}
```

```
[[name]] typename constant DUP ([[initializer]{[,initializer]]...}))
```

Tên (name) là nhãn tượng trưng gán cho biến. Tên kiểu (typename) là tên của kiểu cấu trúc hay liên hợp được khai báo trước đó. Một khởi tạo (initializer) tương ứng với một trường. Khởi tạo phải tương ứng trong kiểu với trường xác định trong khai báo kiểu. Đối với các liên hợp, kiểu của khởi tạo phải giống kiểu của trường đầu tiên. Có thể sử dụng toán tử giả DUP để lập danh sách khởi tạo. Danh sách khởi tạo chỉ có thể được ngắt sau dấu phẩy trừ khi ta kết thúc dòng bằng ký tự (\). Các ký hiệu () hay (>) cuối cùng phải xuất hiện ở cùng dòng với khởi tạo cuối cùng.

Ví dụ 4.11:

ITEMS	STRUCT	
Iname	BYTE	'Item Name'
Inum	WORD	?
UNION	ITYPE	; từ khóa UNION xuất hiện trước
oldtype	BYTE	0;
newtype	WORD	?;
ENDS		
ITEMS	ENDS	

.	.DATA	
Item1	ITEMS	<> ; chấp nhận các khởi tạo mặc định
Item2	ITEMS	{}
Item3	ITEMS	<'Bolts', 126> ; bỏ qua giá trị mặc định của 2 trường đầu tiên, ; sử dụng mặc định của trường thứ 3
Item4	ITEMS	{ 'Bolts', ; Item name 126\ ; Part number }

Ví dụ trên xác định rằng: phân phối không gian nhớ cho 4 cấu trúc của kiểu ITEMS. Các cấu trúc được đặt tên là Item1, Item2, Item3 và Item4. Mỗi một định nghĩa yêu cầu đóng trong < > hay { }. Nếu khởi tạo nhiều hơn một trường thì phân biệt các giá trị bằng các dấu phẩy, như là trong Item3 và Item4. Không cần phải khởi tạo tất cả các trường trong một cấu trúc. Nếu một trường trống, thì trình biên dịch assembler sử dụng giá trị khởi tạo ban đầu của cấu trúc để gán cho trường trống đó trong định nghĩa. Nếu không có giá trị mặc định thì giá trị của trường bỏ qua không xác định.

Các mảng của cấu trúc và liên hợp:

Có thể xác định một mảng của các cấu trúc sử dụng toán tử giả DUP. Ví dụ, có thể xác định một mảng Item7 của các biến cấu trúc như:

```
Item7      ITEMS30 DUP ({, {10}})
```

Mảng Item7 được khai báo ở đây có 30 phần tử của kiểu ITEMS, với trường thứ 3 của từng phần tử (liên hợp) được khởi tạo bằng 10.

Cũng có thể lập danh sách các phần tử của mảng như sau:

```
Item8      ITEMS{'Bolt', 126, 10},  
           {'Pliers', 139, 10},  
           {'Saws', 414, 10}
```

Sử dụng các toán tử giả LENGTHOF, SIZEOF, và TYPE cho cấu trúc và liên hợp:

Toán tử giả SIZEOF trả về kích thước của một cấu trúc là giá trị độ dịch của trường cuối cùng cộng thêm kích thước của trường cuối cùng và bất kỳ số đệm thêm nào khi có sắp xếp. SIZEOF trả về kích thước của một liên hợp là kích thước dài nhất, cộng thêm cả các số đệm thêm khi có sắp xếp. Độ dài của liên hợp xác định bằng toán tử giả LENGTHOF là số lượng các khối tạo bên trong các ký hiệu < > hoặc { }. Toán tử giả TYPE trả về giá trị kiểu của trường dài nhất của liên hợp (bằng số byte).

Ví dụ 4.12: đối với cấu trúc:

INFO	STRUCT	
buffer	BYTE	100 DUP (?)
crlf	BYTE	13, 10
query	BYTE	'Filename:'
endmark	BYTE	36
drives	DISKDRIVES	<0, 1, 1>
INFO	ENDS	
info1	INFO	{, 'Dir'}
lotsof	INFO	{, 'file1',, {0, 0, 0}}, {, 'file2',, {0, 0, 1}}, {, 'file3',, {0, 0, 2}}
sinfo1	EQU	SIZEOF info1 ; 116 là số byte trong các khối tạo
linfo1	EQU	LENGTHOF info1 ; 1 là số các item
tinfo1	EQU	TYPE info1 ; 116 như giá trị kích thước
slotsof	EQU	SIZEOF lotsof ; 116*3 là số byte trong các khối tạo

```

tlotsof      EQU      LENGTHOF  lotsof
              ; 3 là số các item

tlotsof      EQU      TYPE       lotsof
              ; 116 như giá trị kích thước của cấu
              ; trúc kiểu INFO

```

Ví dụ 4.13: Đối với liên hợp:

```

DWD          UNION
d            DWORD    ?
w            WORD     ?
b            BYTE     ?
DWD          ENDS

num          DWD      {0FFFFh}
array        DWD      (100/SIZEOF DWD) DUP ({0})
              ; mảng 4*25

snum         EQU      SIZEOF      num
              ; = 4, vì lấy theo trường d của liên hợp

lnum         EQU      LENGTHOF    num
              ; = 1, chỉ có một giá trị khởi tạo trong các ký hiệu {}

tnum         EQU      TYPE        num
              ; = 4, vì lấy theo trường d

sarray       EQU      SIZEOF      array
              ; = 100 (vì 4*25)

larray       EQU      LENGTHOF    array ; = 25

tarray       EQU      TYPE        array ; = 4

```

c) Tham chiếu đến các cấu trúc, liên hợp, và các trường

Giống như các biến, các biến cấu trúc có thể được truy cập theo tên. Có thể truy cập các trường trong các biến cấu trúc bằng cú pháp sau:

Biến.trường (variable.field)

Ví dụ 4.14:

```

DATE      STRUCT
month     BYTE      ?
day       BYTE      ?
year      WORD      ?
DATE      ENDS
yesterday DATE      {1, 20, 2002}
          ; định nghĩa biến cấu trúc

...

MOV       AL, yesterday.day    ; sử dụng các biến cấu trúc
MOV       BX, offset yesterday ; nạp địa chỉ cấu trúc
MOV       AL, (DATE PTR [BX]).month
          ; sử dụng như là toán hạng gián tiếp
MOV       AL, [BX].date.month
          ; chỉ cần thiết khi month là trường trong một cấu
          ; trúc khác

```

Sử dụng các lệnh MOV có thể truy cập tới các phân tử của mảng các liên hợp. Ví dụ:

```

WB        UNION
w         WORD      ?
b         BYTE      ?
WB        ENDS
array     WB         (100/SIZEOF WB) DUP ({0})
...
MOV       array[12].w, 40h
MOV       array[32].b, 2

```

4.3.10. Các bản ghi

Các bản ghi (records) tương như cấu trúc, nhưng các trường trong các bản ghi là những chuỗi bit. Mỗi một trường bit trong biến bản ghi

có thể được dùng tách biệt trong các toán hạng hằng số hoặc các biểu thức hằng số. Bộ vi xử lý có các lệnh xử lý bit có thể truy cập tới các trường bit.

Các bản ghi là các byte, các từ, hoặc từ kép trong đó các bit riêng biệt hoặc các nhóm bit tập trung thành các trường. Để sử dụng các bản ghi, trong hợp ngữ cũng cần phải có 3 bước thực hiện:

1. Khai báo một kiểu bản ghi
2. Định nghĩa một hay một số biến có kiểu bản ghi
3. Tham chiếu các biến bản ghi.

a) Khai báo kiểu bản ghi

Khai báo kiểu bản ghi là tạo ra một tạm thời cho dữ liệu với các kích thước, và có thể cả giá trị ban đầu cho các trường bit trong bản ghi đó. Nó không thực hiện phân bổ không gian nhớ cho bản ghi.

Toán tử giả RECORD thực hiện khai báo một kiểu bản ghi (8-bit, 16-bit, hay 32-bit). Cú pháp khai báo kiểu bản ghi là:

```
recordname RECORD field [|,field|]...
```

Trường (field) khai báo tên, độ rộng của trường, và giá trị ban đầu cho trường, cú pháp của trường là:

```
fieldname: width[|=expression]]
```

Các nhãn toàn cục, các tên macro, và các tên trường của bản ghi phải là duy nhất, nhưng các tên trường của bản ghi có thể trùng tên như các tên của các trường của cấu trúc. Độ rộng (width) của trường là số bit trong trường, và biểu thức (expression) là một giá trị hằng số cho ban đầu (hoặc mặc định) của trường. Khai báo bản ghi có thể kéo dài hơn một dòng nếu dòng tiếp theo được bắt đầu bằng dấu phẩy. Ví dụ:

```
CW    RECORD    r1:3=0, ic:2=0, rc:2=0, pc:2=3, r2:2=1,  
                    mask:6=63
```


Bản ghi CW có 6 trường, mỗi bản ghi khai báo kiểu này có 16 bit chiếm trong bộ nhớ. Giá trị ban đầu được khởi tạo cho từng trường.

b) Định nghĩa các biến bản ghi

Một khi đã khai báo kiểu bản ghi, tiếp theo phải định nghĩa các biến bản ghi của kiểu bản ghi đó. Đối với mỗi biến, trình assembler phân bổ bộ nhớ ở dạng kiểu bản ghi được khai báo. Cú pháp là:

```
[[name]] recordname <[[initializer[[,initializer]]...]]>
```

```
[[name]] recordname {[[initializer[[,initializer]]...]]}
```

```
[[name]] recordname constant DUP ([[initializer[[,initializer]]...]])
```

Tên của bản ghi (recordname) là tên của bản ghi đã được khai báo ở kiểu bản ghi nhờ toán tử giả RECORD. Nếu sử dụng toán tử DUP để khởi tạo nhiều biến bản ghi thì phải dùng các ký hiệu < > bao bên ngoài giá trị khởi tạo. Ví dụ, định nghĩa một mảng các biến bản ghi sau đây:

```
xmas          COLOR      50 DUP (<1, 2, 0, 4>);
```

Ví dụ sau đây gồm cả khai báo kiểu bản ghi và định nghĩa biến bản ghi warning mà kiểu của biến này đã được khai báo trong kiểu bản ghi COLOR. Các giá trị ban đầu của các trường trong biến bản ghi là tập hợp các giá trị cho trong định nghĩa bản ghi:

```
COLOR RECORD      blink: 1, back: 3, intense: 1, fore: 3  
                  ; bản ghi COLOR
```

```
warning COLOR      <1, 0, 1, 4> ; định nghĩa biến bản ghi
```

Sử dụng các toán tử giả LENGTHOF, SIZEOF, và TYPE với các bản ghi:

Các toán tử SIZEOF và TYPE trả về số byte mà biến bản ghi chiếm. Không thể sử dụng LENGTHOF trong khai báo bản ghi, nhưng có thể sử dụng nó với định nghĩa các biến bản ghi. LENGTHOF trả về số bản ghi trong mảng các bản ghi, hoặc giá trị 1 đối với biến bản ghi đơn.

Ví dụ 4.15:

```

; Định nghĩa bản ghi
; 9 bit lưu trong 2 byte
RGBCOLOR RECORD    red: 3, green: 3, blue: 3
    MOV  AX, RGBCOLOR    ; chính là "MOV AX, 01FFh"
    MOV  AX, SIZEOF      RGBCOLOR
                        ; chính là "MOV AX, 2"
    MOV  AX, TYPE        RGBCOLOR
                        ; chính là "MOV AX, 2"

; Phần bản ghi
; 8 bit lưu trong 1 byte
RGBCOLOR2 RECORD    red: 3, green: 3, blue: 2
rgb                RGBCOLOR <1, 1, 1, 1>
                        ; khởi tạo giá trị 00100101y
    MOV  AX, RGBCOLOR2    ; chính là "MOV AX, 00FFh"
    MOV  AX, LENGTHOF     rgb ; chính là "MOV AX, 1"
    MOV  AX, SIZEOF       rgb ; chính là "MOV AX, 1"
    MOV  AX, TYPE         rgb ; chính là "MOV AX, 1"

```

c) Tham chiếu các biến bản ghi

Sử dụng các toán tử giả WIDTH và MASK với các bản ghi trong tham chiếu các biến bản ghi.

Toán tử WIDTH trả về độ rộng tính bằng số bit của một bản ghi hay của một trường bit của bản ghi. Toán tử MASK trả về một mặt nạ bit đối với các vị trí bit chiếm bởi một trường của bản ghi. Một bit trong mặt nạ có giá trị 1 nếu bit đó tương ứng với một trường bit.

Ví dụ 4.16:

```

.DATA
COLOR RECORD    blink: 1, back: 3, intense: 1, fore: 3
message        COLOR    <1, 5, 1, 1>
                ; blink = 1, back = 5, intense = 1, fore = 1

```

```

wblink EQU WIDTH blink ; "wblink" = 1, trường blink có 1 bit
wback EQU WIDTH back ; "wback" = 3, trường back có 3 bit
wintens EQU WIDTH intense
; "wintense" = 1, trường intense có 1 bit
wfore EQU WIDTH fore ; "wfore" = 3, trường fore có 3 bit
wcolor EQU WIDTH COLOR
; "wcolor" = 8, tổng cộng bản ghi có 8 bit
.CODE
...
MOV AH, message; nạp giá trị 1101 1001
AND AH, NOT MASK back
; NOT MASK back = 10001111, nên kết
; quả AND là 10001001 = (11011001) AND
; (10001111)
OR AH, MASK blink
; MASK blink = 10000000, nên kết quả OR
; là 10001001 = (10001001) OR (10000000)
XOR AH, MASK intense
; MASK intense = 00001000, nên kết XOR
; là 10000001 = (10001001) XOR (00001000)
IF (WIDTH COLOR) GT 8
; nếu color là 16 bit thì nạp vào AX
MOV AX, message
ELSE
MOV AL, message; nạp vào AL 8 bit
XOR AH, AH; xóa 8-bit cao của AX
ENDIF
...

```

Ví dụ này có thể được tiếp tục với các cách sử dụng khác nhau các trường của bản ghi như những toán hạng và các biểu thức tính toán.

4.3.11. Các ký hiệu tượng trưng của Assembler

Trình hợp dịch MASM có một số ký hiệu tượng trưng xác định trước cho phép người lập trình có thể sử dụng chúng bất kỳ chỗ nào trong chương trình để biểu diễn một giá trị của biểu thức. Ví dụ, ký hiệu tượng trưng @FileName thể hiện tên cơ sở của tệp tin hiện thời. Nếu một tệp tin nguồn có tên TASK.ASM, thì giá trị của @FileName là TASK. Các ký tự tượng trưng của MASM được liệt kê trong phụ lục. Chẳng hạn một số ký hiệu tượng trưng có tác động như sau:

@code	trả về tên của đoạn mã (code segment name)
@CodeSize	trả về giá trị nguyên là khoảng chiếm của đoạn mã mặc định
@Model	trả về chế độ của đoạn nhớ
@CurSeg	trả về tên của đoạn hiện thời.
@SizeStr	trả về độ dài của chuỗi đã cho
@Startup	bắt đầu chương trình với các lệnh khởi tạo địa chỉ cơ sở đoạn.

4.4. KHUNG CỦA CHƯƠNG TRÌNH HỢP NGỮ

Như đã trình bày, các vi xử lý Intel từ 8-bit đến 32-bit đều có cơ chế tổ chức bộ nhớ theo các đoạn (segments). Kiến trúc phân đoạn bộ nhớ phần nào là rào cản cho những người lập trình hợp ngữ. Đối với những chương trình nhỏ với cả dữ liệu và các lệnh chiếm không gian nhớ nhỏ hơn kích thước đoạn mặc định 64 kbyte (đối với 8086/8088 và 80286/80386/80486/Pentiums trong chế độ thực) thì việc lập trình hợp ngữ là đơn giản. Nhưng những chương trình lớn, ví dụ với dữ liệu chiếm không gian nhớ lớn hơn kích thước của một đoạn, và có thể cả phần chương trình cũng vượt quá giới hạn của đoạn thì phải giải quyết vấn đề phân đoạn cho chương trình. Vấn đề giới hạn của đoạn 64 kbyte không có ý nghĩa nữa khi các vi xử lý Intel 32-bit làm việc trong chế

độ bảo vệ, bởi vì lúc đó địa chỉ 32-bit có không gian nhớ đến 4 GB, đó là không gian địa chỉ phẳng (flat address).

Tổ chức phân đoạn nhớ của các vi xử lý Intel ảnh hưởng đến khung (hay cấu trúc) của chương trình hợp ngữ.

4.4.1. Các đoạn logic dùng trong hợp ngữ

Trình hợp dịch MASM sử dụng các đoạn logic. Các đoạn logic chứa 3 thành phần của một chương trình: code, dữ liệu và ngăn xếp. MASM tổ chức 3 thành phần này của chương trình hợp ngữ sao cho chúng chiếm các đoạn vật lý của bộ nhớ. Các thanh ghi đoạn CS, DS, và SS chứa các địa chỉ cơ sở của các đoạn vật lý tương ứng trong bộ nhớ mà các đoạn logic của chương trình trú ngụ.

Chúng ta có thể định nghĩa các đoạn bằng 2 cách: bằng các toán tử khai báo các đoạn đơn giản hóa và bằng các định nghĩa đầy đủ đoạn. Có thể sử dụng cả 2 loại trong cùng một chương trình.

Các toán tử giả khai báo các đoạn đơn giản hóa che dấu bớt đi nhiều chi tiết của định nghĩa đoạn, tạo ra mã thuộc tính đoạn, và sắp xếp trật tự của đoạn.

Các định nghĩa đầy đủ đoạn đòi hỏi cú pháp phức tạp nhưng đảm bảo kiểm tra đầy đủ hơn trình assembler tạo ra các đoạn như thế nào.

Cấu trúc chương trình hợp ngữ sử dụng các đoạn đơn giản hóa đòi hỏi phải sử dụng một số toán tử giả để gán các tên, sắp xếp, và đặt các thuộc tính cho các đoạn trong chương trình. Những toán tử này định nghĩa các đoạn theo cách dễ dàng liên kết với các ngôn ngữ bậc cao.

4.4.2. Khung của một module chính của chương trình sử dụng các đoạn đơn giản hóa

Một chương trình ở hợp ngữ có cấu trúc phân theo các module tạo bởi các đoạn, trong đó có một module chính mà ở đó chương trình bắt đầu thực hiện.

Module chính của chương trình có thể chứa các đoạn code (CS), đoạn dữ liệu (DS), hoặc đoạn ngăn xếp (SS). Những module bổ sung của chương trình chỉ chứa các đoạn mã và dữ liệu thôi.

Khung của module chính sử dụng các toán tử giả định nghĩa các đoạn đơn giản hóa có thể như sau:

Ví dụ 4.17:

```
.MODEL      memorymodel
            ; khai báo kiểu bộ nhớ

.STACK[size] ; khai báo ngăn xếp với kích thước tính bằng byte

.DATA      ; kết thúc đoạn ngăn xếp, khai báo dữ liệu
...        ; các khai báo dữ liệu: các biến, hằng số,...
...

.CODE      ; kết thúc đoạn dữ liệu, khai báo đoạn mã

.STARTUP   ; bắt đầu thực hiện chương trình ở đây
...        ; các lệnh thực hiện
...

.EXIT [code] ; trở về hệ điều hành với mã trở về

END        ; kết thúc đoạn mã, kết thúc module chính
```

Các toán tử giả `.DATA` và `.CODE` khai báo các đoạn đơn giản hóa tương ứng và không đòi hỏi phải có toán tử xác định điểm kết thúc đoạn. Chúng thực hiện đóng (kết thúc) đoạn được định nghĩa ở trên và mở một đoạn mới. Toán tử `.STACK` mở và đóng đoạn ngăn xếp nhưng không đóng đoạn hiện thời trên nó, vì vậy nó luôn được đặt ở tên cùng trong phần khai báo các đoạn. Toán tử `.END` đóng đoạn cuối cùng và đánh dấu kết thúc của chương trình nguồn hợp ngữ. Nó luôn ở cuối cùng trong từng module.

4.4.3. Xác định các thuộc tính của đoạn nhờ toán tử `.MODEL`

Mỗi một module chứa các đoạn phải được bắt đầu bằng toán tử `.MODEL`.

Toán tử **.MODEL** xác định các thuộc tính ảnh hưởng đến toàn bộ module chương trình đó là: kiểu bộ nhớ (memory model), gọi mặc định và các qui ước tên, hệ điều hành, và kiểu ngăn xếp. Nó cho phép sử dụng các đoạn đơn giản hóa và kiểm soát các tên của đoạn mã và khoảng cách mặc định đối với các thủ tục. Cần phải đặt **.MODEL** trước tất cả các toán tử giả khai báo các đoạn. Cú pháp là:

.MODEL memorymodel [[, modeloptions]]

Trường kiểu bộ nhớ (memorymodel) cần phải có ngay bên cạnh **.MODEL**, trường chọn lựa kiểu (modeloption) xác định thêm một số thuộc tính khác và có thể có hoặc không. Các chọn lựa kiểu phân cách bởi các dấu phẩy. MASM hỗ trợ các kiểu bộ nhớ chuẩn mà các ngôn ngữ bậc cao của Microsoft sử dụng: TINY, SMALL, COMPACT, MEDIUM, LARGE, HUGE, và FLAT. Bảng 4.1 xác định các thuộc tính của các kiểu bộ nhớ.

Bảng 4.1: Các thuộc tính của các kiểu bộ nhớ

Kiểu	Mã mặc định	Dữ liệu mặc định	Hệ điều hành	Kết hợp dữ liệu và mã
TINY	Near	Near	MS-DOS	Yes
SMALL	Near	Near	MS-DOS, Windows	No
MEDIUM	Far	Near	MS-DOS, Windows	No
COMPACT	Near	Far	MS-DOS, Windows	No
LARGE	Far	Far	MS-DOS, Windows	No
HUGE	Far	Far	MS-DOS, Windows	No
FLAT	Near	Near	Windows NT	Yes

Khi viết các module hợp ngữ cho ngôn ngữ bậc cao, ta cần phải sử dụng cùng một kiểu bộ nhớ như ngôn ngữ gọi tới. Ký hiệu tương ứng @Model của assembler trả về kiểu bộ nhớ. Bảy kiểu bộ nhớ này được phân thành 3 nhóm sau:

a) Các kiểu bộ nhớ SMALL, MEDIUM, COMPACT, LARGE và HUGE

- SMALL hỗ trợ cho một đoạn dữ liệu và một đoạn mã. Mặc định tất cả dữ liệu và mã là gần (Near).

- MEDIUM hỗ trợ cho nhiều đoạn mã và một đoạn dữ liệu.
- COMPACT hỗ trợ cho nhiều đoạn dữ liệu và một đoạn mã.
- LARGE hỗ trợ nhiều đoạn mã và nhiều đoạn dữ liệu. Mặc định tất cả dữ liệu và code là Far.
- HUGE tương tự như LARGE, hỗ trợ những khoản dữ liệu riêng biệt lớn hơn một đoạn, nhưng sự thực hiện chúng cần phải được người lập trình viết lệnh.

b) Kiểu bộ nhớ TINY

TINY nghĩa là rất nhỏ. Các chương trình sử dụng kiểu TINY chỉ chạy dưới MS-DOS. TINY đặt tất cả các dữ liệu và mã trong một đoạn và tổng kích thước chương trình nhỏ hơn 64 kbyte. Mặc định đối với mã và các khoản dữ liệu là gần (Near). Như vậy có thể sử dụng các dịch vụ phân phối bộ nhớ của MS-DOS để phân bố dữ liệu xa một cách động trong trong thời gian chạy chương trình. Kiểu TINY tạo ra các tệp dạng .COM của MS-DOS.

c) Kiểu bộ nhớ FLAT

Kiểu FLAT hỗ trợ cấu hình không phân đoạn trong các hệ điều hành 32-bit. Nó tương tự như kiểu TINY nhưng đặt tất cả các dữ liệu và mã trong một đoạn đơn 32-bit. Để viết một chương trình dùng kiểu FLAT, phải dùng các toán tử giả .386 hoặc .486 trước dòng lệnh .MODEL FLAT. Hệ điều hành tự động khởi tạo các thanh ghi đoạn khi nạp. Chúng ta chỉ cần phải thay đổi chúng khi trong chương trình dùng cả đoạn 16-bit và đoạn 32-bit. CS, DS, ES và SS tất cả chiếm cả siêu nhóm FLAT. Các địa chỉ và con trỏ luôn là 32-bit.

4.4.4. Đoạn ngăn xếp (stack segment)

Ngăn xếp thường dùng để chứa các biến tạm thời, các địa chỉ nhảy trở về, các giá trị cờ, nội dung các thanh ghi của vi xử lý nhờ các lệnh PUSH, POP. Nếu viết trong ngôn ngữ bậc cao thì trình biên dịch sẽ thực hiện các chi tiết tạo một ngăn xếp. Phải dùng .STACK để tạo một đoạn ngăn xếp với cú pháp:

.STACK [size]

Mặc định, assembler phân bổ 1 kbyte của bộ nhớ làm đoạn ngăn xếp. Nó đủ lớn cho hầu hết các chương trình kiểu nhỏ dùng kiểu SMALL. Để tạo khối tạo đoạn ngăn xếp có kích thước thì phải chỉ định giá trị kích thước cho .STACK. Ví dụ, dành 2 kbyte làm đoạn ngăn xếp:

.STACK 2048.

4.4.5. Đoạn dữ liệu (data segment)

Chương trình có thể chứa dữ liệu gần và xa. Thường cần phải đặt các dữ liệu quan trọng và thường xuyên phải dùng tới trong vùng gần để có thể truy cập nhanh hơn. Nhưng vùng đó có thể chật chội để chứa dữ liệu bởi vì trong các hệ điều hành 16-bit tổng dung lượng của tất cả các dữ liệu gần không thể vượt quá 64 kbyte. Do đó, chúng ta phải đặt các dữ liệu không được thường xuyên sử dụng tới hoặc các khoản dữ liệu lớn trong những đoạn nhớ xa. Các toán tử giả .DATA, .DATA?, .CONST, .FARDATA, .FARDATA? được dùng để tạo các đoạn dữ liệu.

a) Khởi tạo đoạn dữ liệu gần (near data segment)

Toán tử .DATA khởi tạo đoạn dữ liệu gần. Trong MS-DOS đoạn dữ liệu gần có dung lượng tối đa 64 kbyte hoặc 512 MB với kiểu FLAT trong Windows NT. Nó thuộc nhóm đặc biệt định nghĩa là DGROUP với dung lượng tối đa là 64 kbyte. Khi sử dụng .MODEL, assembler tự động xác định DGROUP cho đoạn dữ liệu gần. Các đoạn dữ liệu gần thuộc nhóm DGROUP được tham chiếu trực tiếp qua các thanh ghi DS và SS.

Các toán tử .DATA? và .CONST cũng khởi tạo các đoạn thuộc nhóm DGROUP nhưng nhằm tương thích với các ngôn ngữ bậc cao của Microsoft. .CONST dùng để định nghĩa dữ liệu hằng số như các số dấu phẩy động và các chuỗi để cất giữ trong bộ nhớ. .DATA? khởi tạo đoạn dữ liệu lưu trữ các biến không được khởi tạo ban đầu.

Ví dụ 4.18:

```
.DATA
msg    DB    'Hello, world of the macro assembler.$'
w      DW    7
b      DB    3
```

Có thể dùng @data để xác định nhóm của đoạn dữ liệu và @DataSize để xác định kích thước của kiểu bộ nhớ đã được khai báo nhờ .MODEL. Sử dụng @WordSize và @CurSeg sẽ nhận được kích thước và tên của đoạn hiện hành.

b) Khởi tạo đoạn dữ liệu xa (far data segment)

Mặc định, các kiểu bộ nhớ COMPACT, LARGE, và HUGE sử dụng các địa chỉ dữ liệu xa. Trong trường hợp này vẫn dùng được .DATA, .DATA?, và .CONST. Mặc định, chúng luôn tạo đoạn dữ liệu có kích thước tối đa 64 kbyte.

Khi sử dụng .FARDATA hoặc .FARDATA? trong các kiểu bộ nhớ SMALL và MEDIUM, assembler tạo ra các đoạn dữ liệu xa tương ứng FAR_DATA và FAR_BSS.

4.4.6. Đoạn mã

Có thể có cả các đoạn mã gần và xa trong một module.

a) Khởi tạo đoạn mã gần

Cú pháp: .CODE name khai báo bắt đầu đoạn mã của chương trình. Trong đó, tên (name) là tùy chọn nhưng phải là chuỗi ký đúng cú pháp của tên. Không cần thiết phải có tên trong chương trình dùng kiểu .SMALL, bởi vì như vậy assembler sẽ phát sinh lỗi.

Ví dụ 4.19:

```
.CODE
.STARTUP
...
```

... ; đặt trong này các lệnh thực hiện

...

.EXIT

END

Toán tử END kết thúc chương trình và đóng luôn đoạn mã.

b) Khởi tạo đoạn mã xa

Khi chương trình cần dung lượng nhớ lớn hơn 64 kbyte, ta sử dụng các kiểu bộ nhớ MEDIUM, LARGE, hoặc HUGE để tạo ra các đoạn mã xa. Mặc định, các kiểu MEDIUM, LARGE, và HUGE sử dụng các địa chỉ mã xa. Trình dịch Assembler tạo ra đoạn mã khác cho từng module. Nếu nhiều đoạn mã trong các chương trình kiểu SMALL, COMPACT, hay TINY thì phải dùng trình liên kết (Link) kết hợp các đoạn mã thành một đoạn mã.

Đối với các đoạn mã xa, một module riêng lẻ có thể chứa nhiều đoạn mã. Assembler đặt tên cho từng đoạn mã MODNAME_TEXT, trong đó MODNAME là tên của module. Còn đối với mã gần, assembler đặt tên từng đoạn mã là _TEXT để cho chương trình Link kết hợp được các đoạn thành một module. Có thể bỏ qua tên mặc định nếu sử dụng một đối số sau toán tử .CODE. Ví dụ sau đây khởi tạo hai đoạn mã riêng biệt trong một chương trình, FIRST_TEXT và SECOND_TEXT:

.CODE FIRST

... ; Các lệnh thực hiện

.CODE SECOND

... ; Các lệnh thực hiện

c) Bắt đầu (starting) và kết thúc đoạn mã nhờ .STARTUP và .EXIT

Cách đơn giản nhất để bắt đầu thực hiện và kết thúc một chương trình trong MS-DOS là sử dụng các toán tử giả .STARTUP và .EXIT trong module chính của chương trình. Module chính chứa điểm bắt đầu và thường cả điểm kết thúc. Không cần thiết phải có các toán tử

giả này trong module gọi tới module khác. Nhưng lưu ý là những toán tử này không sử dụng được trong các chương trình kiểu FLAT cho các hệ điều hành 32-bit như Windows NT. Toán tử .STARTUP thường bắt đầu ngay sau .CODE, mà ở đó sự thực hiện chương trình bắt đầu. Toán tử .EXIT tạo ra mã có thể thực hiện, và thường đứng ngay trước END. Trong khi đó END thì không tạo ra mã thực hiện. END thông báo cho assembler biết rằng đã đạt đến điểm kết thúc của một module.

Ví dụ 4.20:

```
.CODE
.STARTUP
...
...           ; các lệnh thực hiện
.EXIT
END
```

Tất cả các module phải được kết thúc bằng toán tử END. Nếu chúng ta không dùng .STARTUP thì cần phải đưa ra địa chỉ bắt đầu như là một đối số cho toán tử END. Khi đó toán tử .EXIT cũng không cần. Đó là địa chỉ tượng trưng kết thúc bởi dấu: và đặt ở trường tên ngay dòng lệnh sau .CODE, và chỉ ở toán tử END của chính module này địa chỉ đầu mới chính là đối số. Khi có .STARTUP thì assembler bỏ qua đối số của END. Trong ví dụ sau đây: start là địa chỉ bắt đầu của chương trình:

```
.CODE
start:
...           ; các lệnh thực hiện
...           ;
END          start
```

Đối với thuộc tính mặc định NEARSTACK, thì toán tử .STARTUP khi thực hiện tạo ra các lệnh khởi tạo địa chỉ đoạn dữ liệu DS trở tới DGROUP, và tập hợp SS:SP tương đối với DGROUP như sau:

@Startup:

```

MOV    DX, DGROUP
MOV    DS, DX
MOV    BX, SS
SUB    BX, DX
SHL    BX, 1 ; nếu .286 hoặc cao hơn
SHL    BX, 1
SHL    BX, 1
CLI ; cấm ngắt (nhưng không cần thiết trong .286 hoặc cao hơn)
MOV    SS, DX
ADD    SP, BX
STI ; cho phép ngắt (nhưng không cần thiết trong .286 hoặc cao hơn)
...
...
...
END    @Startup

```

Một chương trình thực hiện trong MS-DOS với thuộc tính FARSTACK không cần thiết phải điều chỉnh SS:SP, do đó .STARTUP chỉ thực hiện những lệnh như sau:

@Startup:

```

MOV    DX, DGROUP
MOV    DS, DX
...
...
END    @Startup

```

Người lập trình có thể tự viết đoạn đầu của module chính của chương trình với nhãn địa chỉ ban đầu gồm những lệnh khởi tạo địa chỉ

cơ sở các đoạn, nhưng nếu sử dụng toán tử giả `.STARTUP` của assembler sẽ tiết kiệm hơn.

Toán tử giả `.EXIT` thực hiện trở về hệ điều hành (dấu nhắc của hệ điều hành), và cú pháp đầy đủ là:

`.EXIT [value]`

Giá trị `value` trả về bằng 0 nếu không có lỗi trong chương trình, và bằng 1 nếu có lỗi. Giá trị trả về nằm trong thanh ghi `AL`. Nếu không có giá trị đối số ở `.EXIT` thì `.EXIT` sẽ trả về bất kỳ giá trị nào trong `AL`:

```
MOV    AL, value
MOV    AH, 04Ch
INT     21h
```

Như vậy khung module chính của chương trình viết ở mục 4.4.2 khi sử dụng địa chỉ đầu tượng trình sẽ được viết lại như sau:

Ví dụ 4.21:

```
.MODEL      memorymodel ; khai báo kiểu bộ nhớ
.STACK[size] ; khai báo ngăn xếp với kích thước tính bằng byte
.DATA       ; kết thúc đoạn ngăn xếp, khai báo dữ liệu
...         ; các khai báo dữ liệu: các biến, hằng số,...
...         ;
.CODE       ; kết thúc đoạn dữ liệu, khai báo đoạn mã
start:     ; bắt đầu thực hiện chương trình ở đây
MOV    AX, @DATA
MOV    DS, AX;
...
...
MOV    AL, value
MOV    AH, 04Ch
```

INT 21h ; kết thúc chương trình, trở về hệ điều hành với mã
; (code) trở về
END start ; kết thúc đoạn mã, kết thúc module chính

4.4.7. Định nghĩa đầy đủ các đoạn

Nếu chúng ta muốn có kiểm tra đầy đủ đối với các đoạn thì có thể định nghĩa đầy đủ các đoạn trong phần khai báo chúng. Cú pháp định nghĩa đầy đủ các đoạn như sau:

```
name SEGMENT[[align]][[READONLY]][[combine]][[use]]
                [['class']] statements
name ENDS
```

Trường tên (name) xác định tên của đoạn. Có thể chọn tên bất kỳ, nhưng phải đúng cú pháp của một nhân tên. Các chọn lựa đi sau toán tử SEGMENT đóng trong các ngoặc vuông [[]] có ý nghĩa là các kiểu của đoạn và có chức năng như sau:

Sắp xếp (align) xác định ranh giới bộ nhớ mà ở đó bắt đầu một đoạn mới.

Chỉ đọc (Readonly) đặt đoạn chỉ được đọc, cấm ghi. Khi có một lệnh nào đó có tham chiếu ghi vào đoạn thì assembler lập báo cáo lỗi.

Kết hợp (combine) xác định cho trình Linker sẽ kết hợp như thế nào các đoạn từ các module khác nhau khi xây dựng các tệp thực hiện được.

Sử dụng (use) chỉ dùng trong vi xử lý 32-bit và xác định kích thước của đoạn. Toán tử USE16 chỉ ra giá trị độ dịch là 16-bit và USE32 chỉ ra giá trị độ dịch là 32-bit.

Lớp (class) là tên một lớp của đoạn. Linker tự động nhóm các đoạn có cùng lớp trong bộ nhớ.

Các kiểu chọn lựa có thể xếp đặt không theo thứ tự nhưng không được lặp lại 2 lần trong câu lệnh SEGMENT.

a) Sắp xếp (align) các đoạn

Kiểu sắp xếp (align) trong toán tử SEGMENT xác định một khoảng các địa chỉ của bộ nhớ mà ở đó địa chỉ bắt đầu của đoạn có thể được chọn. Có một số loại sắp xếp như sau:

BYTE địa chỉ bắt đầu là địa chỉ byte sau đó.

WORD địa chỉ bắt đầu là địa chỉ từ sau đó.

DWORD địa chỉ bắt đầu là địa chỉ từ kép sau đó.

PARA địa chỉ bắt đầu là địa chỉ đoạn tin (paragraph) sau đó, mặc định một đoạn tin có 16 byte.

PAGE địa chỉ bắt đầu là địa chỉ trang sau đó. Mỗi một trang có 256 byte

b) Tạo đoạn chỉ được đọc

Kiểu chọn lựa Readonly tạo cho đoạn khởi tạo có thuộc tính được bảo vệ, chỉ được đọc, hoặc khi chúng ta viết chương trình để cài đặt lên ROM.

c) Kết hợp các đoạn

Chọn lựa combine kiểm soát hành vi của trình liên kết (Link), và nó gồm các loại như sau:

PRIVATE không kết hợp đoạn với đoạn từ các module khác nhau, ngay khi chúng có cùng tên. Đặt là mặc định.

PUBLIC kết hợp tất cả các đoạn trùng tên (ở các tệp nguồn khác nhau) để tạo ra một đoạn riêng và liên tục.

STACK kết hợp tất cả các đoạn trùng tên và gây ra cho hệ điều hành thiết lập SS:00 cho đáy (bottom) và SS:SP cho đỉnh (top) của đoạn ngăn xếp kết quả.

COMMON chồng lấn các đoạn. Độ dài của vùng kết quả là độ dài của đoạn dài nhất trong các đoạn kết hợp.

MEMORY sử dụng như là loại PUBLIC.

AT address coi địa chỉ (address) là vị trí của đoạn. Một đoạn AT không thể chứa bất kỳ mã hoặc dữ liệu nào nhưng nó tiện dụng cho định nghĩa các cấu trúc hoặc biến tương ứng với các vị trí nhớ xa, như bộ đệm của màn hình hoặc vùng nhớ thấp. Chúng ta không thể dùng AT combine trong các chương trình ở chế độ bảo vệ.

d) Thiết lập thứ tự các đoạn theo lớp

Hai đoạn không cùng lớp không thể kết hợp với nhau được. Để kết hợp nhanh, trình Linker sắp xếp các đoạn sao cho các đoạn có cùng lớp ở kề cận nhau trong một tệp thực hiện và theo một trật tự mà các toán tử giả .ALPHA, .SEQ, và DOSSEG xác định trong từng tệp đối tượng (.OBJ). Một trật tự các đoạn thường dùng là các đoạn mã phải xuất hiện trước trong tệp thực hiện. Đối với các chương trình kiểu TINY dạng .COM, các đoạn mã phải xuất hiện đầu tiên trong tệp thực hiện, bởi vì sự thực hiện phải bắt đầu ở địa chỉ 100h.

Toán tử .SEQ sắp xếp các đoạn theo thứ tự mà chúng được khai báo.

Toán tử .ALPHA sắp xếp các đoạn theo thứ tự bảng chữ cái.

Toán tử .DOSSEG sắp xếp các đoạn theo thứ tự chuẩn của các ngôn ngữ của Microsoft. Không thể dùng .DOSSEG trong module được gọi từ module khác. Thứ tự sắp xếp mà DOSSEG thực hiện là:

1. Các đoạn mã
2. Các đoạn dữ liệu và theo thứ tự là:
 - a. Các đoạn không thuộc lớp BSS hoặc ngăn xếp
 - b. Các đoạn BSS
 - c. Các đoạn ngăn xếp

e) Khởi tạo thanh ghi đoạn mặc định cho các đoạn

Toán tử giả .ASSUME được sử dụng để chỉ định thanh ghi đoạn cho một đoạn. Như vậy một đoạn mặc định hay các địa chỉ của nhóm được chỉ định cho các thanh ghi đoạn nhằm đảm bảo cho assembler

khi tham chiếu đến một địa chỉ nó biết được đoạn nào có chứa địa chỉ.

Cú pháp của `.ASSUME` như sau:

```
ASSUME segregister: seglocation [, segregister: seglocation]
ASSUME dataregister: qualifiedtype [, dataregister: qualifiedtype]
ASSUME register: ERROR [, register: ERROR]
ASSUME [register:] NOTHING [, register: NOTHING]
ASSUME register: FLAT [, register: FLAT]
```

Trường thanh ghi đoạn vị trí đoạn (seglocation) phải là tên của đoạn hay của nhóm được chỉ định cho một thanh ghi đoạn có tên trong trường thanh ghi đoạn (segregister). Ví dụ:

```
ASSUME CS: MY_CODE; chỉ định đoạn code tên là MY_CODE
                        ; cho CS
```

Từ khóa `NOTHING` xóa tất cả các chỉ định các thanh đoạn đã thực hiện bằng `.ASSUME` trước đó.

Bình thường, chỉ bằng một dòng lệnh `ASSUME` để chỉ định tất cả các thanh ghi đoạn: `CS`, `DS`, `ES`, và `SS` (và `FS` và `GS` trong 80386/80486/Pentiums) ở ngay đầu module chính của chương trình nguồn. Nhưng có thể sử dụng `ASSUME` ở bất kỳ điểm nào trong chương trình để thay đổi các chỉ định đoạn.

Có thể ngăn chặn sự sử dụng một thanh ghi nào đó bằng từ khóa `ERROR`:

```
ASSUME register: ERROR; register là tên thanh ghi bị cấm
                        ; sử dụng.
```

Chẳng hạn với lệnh `ASSUME CS: ERROR` assembler sẽ thông báo lỗi khi sử dụng các toán tử cho các đoạn đơn giản hóa để tạo các đoạn dữ liệu.

f) Khởi tạo các nhóm đoạn

Một nhóm đoạn là một tập hợp các đoạn mà tổng dung lượng của chúng không lớn hơn 64 kbyte trong chế độ 16-bit. Tổ chức nhóm các đoạn cho phép phát triển các đoạn logic riêng biệt đối với các loại dữ

liệu khác nhau và sau đó kết hợp chúng lại thành một đoạn, tức là một nhóm các đoạn, cho tất cả các dữ liệu. Sử dụng nhóm có thể tiết kiệm các thanh ghi đoạn để truy cập đến các đoạn khác nhau, và dĩ nhiên chương trình sẽ có ít lệnh hơn, thực hiện nhanh hơn.

Một số đoạn dùng trong Microsoft như `_DATA`, `_BSS`, `CONST`, và `STACK` được kết hợp thành một nhóm có tên là `DGROUP` cho các dữ liệu gần. Các ngôn ngữ bậc cao đặt tất cả các đoạn dữ liệu gần trong nhóm này. Mặc định ngăn xếp cũng được đặt trong nhóm này. Toán tử giả `.MODEL` tự động xác định `DGROUP`. Bình thường thanh ghi `DS` trỏ đến nơi bắt đầu của nhóm `DGROUP` qua đó có thể truy cập tương đối đến tất cả dữ liệu trong nhóm `DGROUP`. Cú pháp của khai báo nhóm là:

```
name GROUP segment [[, segment]]...
```

Trường tên (`name`) là tên của nhóm. Nó có thể tham chiếu đến nhóm đã được khai báo trước đó. Điều này cho phép bổ sung các đoạn vào nhóm một lần, ví dụ, nếu nhóm có tên là `MYGROUP` đã được khai báo trước để nhóm các đoạn có tên là `ASEG` và `BSEG` vào, thì dòng lệnh:

```
MYGROUP GROUP CSEG
```

là hợp lệ, nó bổ sung đoạn tên là `CSEG` thêm vào nhóm `MYGROUP` mà không loại bỏ các đoạn `ASEG` và `BSEG` khỏi nhóm.

Trong trường tên đoạn (`segment`) có thể viết bất kỳ một tên nào của đoạn đã được khai báo trước đó. Chỉ có một giới hạn là tên đoạn đó chỉ nằm trong một nhóm mà thôi. Nó không thể nằm trong hai nhóm được.

Toán tử `GROUP` không ảnh hưởng đến trật tự mà các đoạn trong nhóm sắp xếp. Chúng ta có thể đưa vào nhóm bao nhiêu đoạn trong chế độ 16-bit (chế độ 8086/8088) tùy ý miễn làm sao cho tổng dung lượng của tất cả các đoạn không vượt quá 64 kbyte (65536 byte).

Trường hợp phải thực hiện là tạo ra một tệp tin dạng .COM, trong đó kích thước tổng thể của tất cả các đoạn trong chương trình không vượt quá 64 kbyte, thì chúng ta có thể sử dụng toán tử GROUP nhóm các đoạn khác nhau: ngăn xếp, dữ liệu và mã vào chung một nhóm, khi đó câu lệnh nhóm là:

```
CGROUP    GROUP    code, data, stack
```

Trong đó, code, data, và stack là tên các đoạn mã, dữ liệu và ngăn xếp tương ứng, còn tên của nhóm là CGROUP. Như vậy các thanh ghi CS, DS, SS cùng trỏ đến địa chỉ đầu của nhóm.

g) Khung của module chính của chương trình với các định nghĩa đầy đủ các đoạn

Cũng như khi dùng các toán tử giả khai báo các đoạn đơn giản hóa, cấu trúc khung của module chính của chương trình nguồn để dịch ra tệp thực hiện ngay dạng .EXE với định nghĩa đầy đủ các đoạn như sau:

Ví dụ 4.22:

; Đoạn ngăn xếp:

```
stack    SEGMENT    STACK
          DB          16 DUP ('STACK  ')
```

```
stack    ENDS
```

; Đoạn dữ liệu:

```
data     SEGMENT
```

```
...                               ; các khai báo dữ liệu
```

```
data     ENDS
```

; Đoạn mã:

```
code     SEGMENT
```

```
          ASSUME     CS: code, DS: data
```

```
main     PROC        FAR    ; bắt đầu thủ tục chính xa
```

```

start:
        MOV     AX, data
        MOV     DS, AX ; đặt DS trỏ đến đoạn dữ liệu mang
                        ; tên data
        ...
        ...
main    ENDP      ; kết thúc thủ tục chính
code    ENDS      ; kết thúc đoạn mã
END     start

```

Đoạn ngăn xếp được khai báo tên là *stack* và kích thước 108 byte (16×8) với 8 byte đầu tiên chứa *STACK* (và 3 khoảng trống làm đầy 8 byte). Đoạn dữ liệu được khai báo tên là *data*. Đoạn mã được khai báo tên là *code*, và bắt đầu bằng toán tử giả *ASSUME* để chỉ định các thanh ghi CS và DS cho các đoạn và đoạn dữ liệu. Khi một chương trình dạng .EXE bắt đầu thực hiện thì thanh ghi CS trỏ đến đầu của đoạn lệnh, còn thanh ghi DS phải trỏ đến đầu của đoạn dữ liệu. Nhưng các thanh ghi DS và ES phải trỏ đến địa chỉ để trở về DOS sau khi chương trình kết thúc. Do vậy công việc đầu tiên chương module thực hiện chương trình là phải cất giữ địa chỉ trở về DOS trong DS vào ngăn xếp (*PUSH DS*) trước khi đặt DS trỏ đến đoạn dữ liệu mới mang tên *data*. Vì địa chỉ trở về DOS là DS: AX, trong đó $AX = 0$, do vậy cần xóa AX (bằng lệnh *XOR AX*) và cất giữ $AX = 0$ vào ngăn xếp (*PUSH AX*). Bước tiếp theo là đặt DS trỏ đến đoạn *data* bằng cặp lệnh: *MOV AX, data; MOV DS, AX*. Phải dùng AX làm trung gian vì các thanh ghi đoạn chỉ trao đổi dữ liệu với các thanh ghi.

Bên trong đoạn mã các lệnh thường tổ chức trong một hay nhiều thủ tục. Một thủ tục được bao bởi các toán tử giả:

```

label    PROC [[distance]][[...]]
...      ; các lệnh thực hiện trong thủ tục

```

```
...
RET
label ENDP
```

Trường label là tên thủ tục. PROC và ENDP là cặp toán tử giả để khai báo và kết thúc thủ tục. PROC có các đối có thể chọn lựa (chúng sẽ được xét ở mục 4.5) ví dụ các thuộc tính khoảng cách [[distance]]: FAR, NEAR,... Các chương trình dịch ra dạng .EXE thường là những thủ tục xa nên trong cú pháp chọn FAR. Kết thúc thủ tục bao giờ cũng có chỉ dẫn của vi xử lý RET (trở về từ một thủ tục). Như vậy sáu chỉ dẫn của vi xử lý sau đây là lõi của module chính của một chương trình nguồn để dịch và liên kết thành tệp dạng .EXE:

```
PUSH    DS
XOR      AX, AX
PUSH    AX
MOV      AX, data
MOV      DS, AX
```

```
...
RET
```

Phải luôn luôn có sáu chỉ dẫn này của vi xử lý nếu như chương trình .EXE được kết thúc bằng RET. Với kiểu khai báo đầy đủ thì đoạn code có thể viết:

Ví dụ 4.23:

; Đoạn mã:

```
code    SEGMENT
        ASSUME    CS: code, DS: data
main    PROC      FAR    ; bắt đầu thủ tục chính xa
start:
        MOV      AX, data
        MOV      DS, AX ; đặt DS trỏ đến đoạn dữ liệu mang
                        ; tên data
```

```

...                ; các lệnh thực hiện
...
                RET                ; trở về từ thủ tục chính
main  ENDP          ; kết thúc thủ tục chính
code  ENDS          ; kết thúc đoạn mã
END    start

```

Và với kiểu khai báo đoạn đơn giản, đoạn mã có thể viết:

; Đoạn mã:

```

                .CODE
main  PROC        FAR
start:
...                ; các lệnh của thủ tục chính
...
                RET
main  ENDP
                END    start

```

Tuy nhiên, những phiên bản sau này của MS-DOS không cần cất giữ vào ngăn xếp địa chỉ trở về DOS có trong DS, và cũng không cần có RET trong thủ tục chính, mà chỉ cần cho DS trở đến đoạn dữ liệu, sau đó kết thúc chương trình bằng lời gọi trở về DOS bằng ngắt chức năng DOS là INT 21h với mã chức năng AH = 4Ch. Như vậy chỉ cần có:

```

MOV     AX, data
MOV     DS, AX
...
MOV     AH, 4Ch
INT     21h

```

Ví dụ dưới đây là một module chương trình dùng để kết hợp tất cả các đoạn cùng tên ở các tệp nguồn khác nhau để tạo thành một đoạn

riêng và liên tục (SEGMENT PUBLIC), với chức năng đọc từ bảng mã bàn phím ASCII. Lệnh INT 21h là ngắt chức năng DOS với mã AH = 7 để đọc ký tự bàn phím và trả về ở AL = mã ASCII, AH = 1 nếu là mã ASCII, và AH = - 1, nếu là mã đặc biệt.

Ví dụ 4.24:

```

CGROUP      GROUP      CODE_SEG
              ASSUME     CS: CGROUP, DS: CGROUP
CODE_SEG     SEGMENT    PUBLIC
              PUBLIC     RD_BYTE
RD_BYTE      PROC       NEAR
              MOV        AH, 7
              ; đọc ký tự từ bảng phím không có echo
              INT        21h    ; đặt mã vào AL
              OR         AL, AL ; kiểm tra mã mở rộng?
              JZ         EXTENDED_CODE
              ; đúng là mã mở rộng

NOT_EXTENDED:
              MOV        AH, 1; báo là mã ASCII bình thường
DONE_READING:
              RET
EXTENDED_CODE:
              INT        21h    ; đọc mã mở rộng
              MOV        AH, 0FFh ; tín hiệu mã mở rộng
              JMP        DONE_READING
RD_BYTE      ENDP
CODE_SEG     ENDS
              END

```


Một ví dụ khác, chương trình in ra chuỗi ký tự kết thúc bằng ký hiệu \$ bằng chức năng DOS: INT 21h, AH = 9. Chương trình dịch ra dạng .EXE và không cần RET:

Ví dụ 4.25:

```

PAGE          , 132          ; định trang với 132 cột
TITLE HELLO          ; tên chương trình là HELLO
COMMENT  * In ra màn hình một chuỗi ký tự *
.MODEL        SMALL
.STACK        100h
.DATA
msg  DB          'Hello, world of the macro assembler.$'
.CODE
main PROC
    MOV        AX, @DATA
    MOV        DS, AX
; hiển thị ra màn hình thông báo
    MOV        DX, OFFSET msg
    MOV        AH, 9
    INT        21h
; trở về DOS
    MOV        AH, 4Ch
    INT        21h
main ENDP
        END    main

```

Hoặc có thể viết phân đoạn mã khác không có thủ tục:

```

.CODE
start:
    MOV        AX, @DATA
    MOV        DS, AX

```

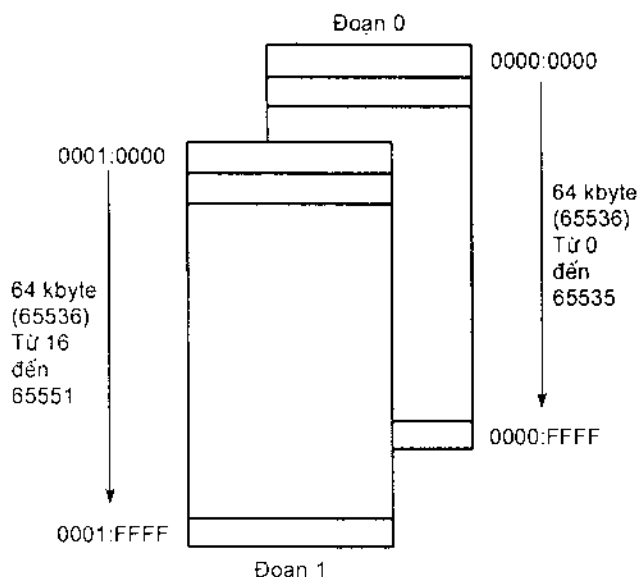
```

MOV     DX, OFFSET msg
MOV     AH, 9
INT     21h
MOV     AH, 4Ch
INT     21h
END     start

```

4.4.8. Các cấu trúc chương trình nguồn khi dịch ra dạng .exe và .com

MS-DOS hỗ trợ tổ chức bộ nhớ phân theo các đoạn là một đặc trưng của các vi xử lý Intel từ 16-bit đến 32-bit. Tuy nhiên, như chúng ta đã thấy, ở chế độ 16-bit (8086/8088) các đoạn nhớ chỉ có dung lượng tối đa là 64 kbyte (65536 byte). Nhưng bộ nhớ trong cơ chế phân đoạn 64 kbyte lại chia thành các đoạn gối lên nhau (overlapping segments), cứ 16 byte thì bắt đầu một đoạn mới (hình 4.1).



Hình 4.1: Các đoạn 64 kbyte gối lên nhau, cứ 16 byte thì bắt đầu một đoạn mới

Theo cơ chế sắp xếp này, đoạn đầu tiên (SEGMENT 0) được bắt đầu ở vị trí 0 của bộ nhớ, đoạn thứ hai (SEGMENT 1) bắt đầu ở vị trí 16 (hay 10h), đoạn thứ 3 (SEGMENT 2) bắt đầu ở 32 (20h),...

Các chương trình hợp ngữ viết để chạy trong MS-DOS khi được dịch và liên kết có thể có hai dạng thực hiện được ngay là .EXE và .COM. Để tạo .EXE chỉ cần dùng hợp dịch và sau đó là trình liên kết (Link). Nhưng để tạo tệp .COM thì phải thực hiện dùng chương trình EXE2BIN để chuyển tệp từ .EXE sang .COM. Nhưng điểm khác biệt quan trọng là chương trình nguồn để dịch sang .EXE có cấu trúc khác với chương trình nguồn để dịch ra .COM.

Một chương trình nguồn để dịch ra .EXE có cấu trúc đầy đủ các khai báo cho các đoạn: ngăn xếp, dữ liệu và mã thì cũng sẽ có một sự sắp xếp gối đầu. Chẳng hạn, nếu một chương trình có các đoạn SS, DS, ES, và CS thì các đoạn này có sự sắp xếp gối lên nhau.

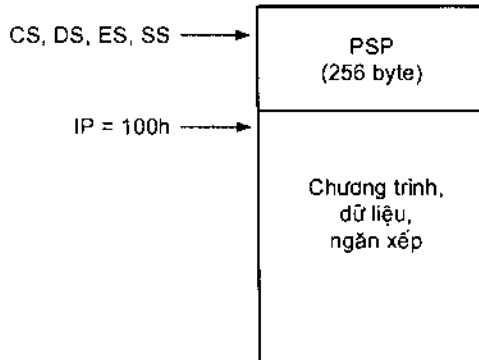
Khác với .EXE, tệp dạng .COM chỉ có một đoạn duy nhất. Khi nạp vào bộ nhớ và thực hiện chương trình .COM thì cả 4 thanh ghi đoạn: SS, DE, ES và CS đều trỏ đến đầu của vùng mào đầu đoạn chương trình PSP (Program Segment Prefix). PSP có dung lượng 256 byte (100h) và chứa một số thông tin cần cho MS-DOS sử dụng như: để thoát khỏi chương trình nhờ lệnh INT 20h hay INT 21h với AH = 4Ch (ngay đầu của PSP là chỉ dẫn INT 20h nếu dùng INT 20h để thoát khỏi chương trình trở DOS); tại độ dịch (offset) 5Ch và 6Ch của mào đầu PSP là hai khối điều khiển tệp FCB (File Control Block); tại độ dịch 5Dh có chứa biến thời gian (time argument); bắt đầu tại độ dịch 80H của mào đầu PSP chứa những ký tự gõ vào ngay sau tên của chương trình của dòng lệnh do người sử dụng đánh vào sau dấu nhắc của MS-DOS.

Như vậy trong tệp .COM phần mã thực hiện ngay luôn bắt đầu ở độ dịch 100h trong đoạn mã, ngay sau vùng PSP, tức là đặt IP = 100h. Trong khi đó ở tệp .EXE giá trị ban đầu của IP = 0. Vì vậy trong chương trình nguồn luôn phải có toán tử giả:

```
ORG 100h
```

ngay sau dòng lệnh .CODE đối với các khai báo các đoạn đơn giản hóa và sau dòng lệnh ASSUME đối với các khai báo đầy đủ các đoạn.

Toán tử giả ASSUME phải chỉ định các thanh ghi CS, DS, ES, và SS cùng trỏ đến một đoạn chung - đoạn mã (hình 4.2).



Hình 4.2: Chương trình .COM trong bộ nhớ

Cấu trúc của một chương trình nguồn dạng .COM sẽ như sau:

Ví dụ 4.26:

```

TITLE  progname
.MODEL  SMALL
.CODE
ORG      100h

start:
        JMP      main

msg      DB      "Hello, world of the macro assembler"

main     PROC
        MOV      DX, OFFSET msg
        MOV      AH, 9
        INT      21h

```

```

                MOV     AH, 4Ch
                INT     21h
main           ENDP
                END     start

```

Không cần thiết trong tệp dạng .COM phải khởi tạo thanh ghi DS trỏ tới đoạn dữ liệu:

```

                MOV     AX, @DATA
                MOV     DS, AX

```

Hoặc với khai báo đầy đủ các đoạn, tệp dạng .COM như sau:

Ví dụ 4.27:

```

                TITLE prname
code           SEGMENT NEAR
                ASSUME  CS: code, DS: code
                ORG     100h
start:
                JMP     main
msg           DB       "Hello, world of the macro assembler"
main          PROC
                MOV     DX, OFFSET msg
                MOV     AH, 9
                INT     21h
                MOV     AH, 4Ch
                INT     21h
main          ENDP
code          ENDS
                END     start

```

4.5. CƠ CHẾ LÀM VIỆC VÀ CẤU TRÚC CHƯƠNG TRÌNH CON

Một chương trình được viết ngắn gọn, dễ kiểm tra khi nó được viết thành các module, gọi là các thủ tục (procedure). Các thủ tục có thể nằm trong cùng một đoạn, khi đó chúng được gọi là các thủ tục gần (NEAR). Chương trình hay thủ tục khai báo bằng .MODEL với các kiểu bộ nhớ: TINY, SMALL, COMPACT và FLAT được coi là NEAR. Khi các thủ tục nằm ở khác đoạn thì chúng được gọi là các thủ tục xa (FAR). Chương trình hay thủ tục khai báo bằng .MODEL với các kiểu bộ nhớ: MEDIUM, LARGE và HUGE được coi là FAR. Một chương trình hay một thủ tục gọi đến một thủ tục khác bằng chỉ dẫn của vi xử lý CALL, cú pháp:

CALL procedure_name

Thủ tục là một chương trình con, và lệnh CALL gọi đến thủ tục kéo theo phải thực hiện cất giữ vào ngăn xếp (PUSH) địa chỉ trở về chương trình chính hay thủ tục gọi. Chúng ta đã khảo sát các lệnh CALL và RET với các thao tác ngăn xếp và chương trình khi nói về tập lệnh của vi xử lý rồi. Nhưng ở đây xét theo hợp ngữ thì cần phải có những toán tử giả để khai báo các thủ tục hay chương trình con.

4.5.1. Khai báo thủ tục

Như đã trình bày, cặp toán tử giả PROC và ENDP dùng để khai báo thủ tục. PROC xác định bắt đầu thủ tục và ENDP xác định kết thúc thủ tục. Cú pháp khai báo đầy đủ một thủ tục là:

```
label    PROC        [[NEAR|FAR]]
...                               ; các lệnh thực hiện trong thủ tục
RET                               ; trở về chương trình hay thủ tục gọi
label    ENDP        [[constant]]
```

Label là tên thủ tục, lệnh CALL label (trong chương trình hay thủ tục gọi) chính là để gọi tới thủ tục. Lệnh RET để trở về chương trình hay thủ tục gọi. Nó phục hồi (POP) từ ngăn xếp các nội dung

thanh ghi và địa chỉ (đã được cất giữ vào ngăn xếp khi thực hiện lệnh CALL label) của lệnh tiếp theo (sau lệnh CALL label) phải thực hiện trong chương trình hay thủ tục gọi. Khoảng cách gần hay xa (NEAR/FAR) đối với các loại vi xử lý 80386/80486/Pentiums lập trình khi lập với các đoạn 16-bit và 32-bit có thể sử dụng: NEAR16, NEAR32, FAR16, và FAR32. Giá trị chọn lựa hằng số (constant) sau RET là số byte được cộng thêm vào giá trị của thanh ghi SP sau khi trở về.

Chúng ta có thể khai báo các thủ tục không cần PROC và ENDP nhưng khi đó cần phải biết được chắc chắn kích thước của lệnh CALL trùng với kích thước của lệnh RET, tức là nếu gọi xa thì phải trở về xa, nếu gọi gần thì phải trở về gần. Nếu gọi gần mà trở về xa thì giá trị phục hồi từ ngăn xếp sẽ không đúng với giá trị cất giữ của gọi gần, bởi vì gọi gần cất giữ một từ địa chỉ của IP, còn trở về xa thì phục hồi hai từ địa chỉ: một từ địa chỉ của IP và một từ địa chỉ của CS. Có thể sử dụng RETN (trở về gần) hoặc RETF (trở về xa) thay cho RET để bỏ qua kích thước mặc định, chẳng hạn, với trường hợp như sau:

```
CALL      NEAR PTR  task ; CALL được khai báo là gọi gần
...       ; trở về từ thủ tục gần (task) sẽ thực hiện tiếp ở đây
task:     ; thủ tục bắt đầu với nhãn gần
...       ; Các lệnh của thủ tục ở đây
...
RETN      ; trở về từ thủ tục gần
```

Cú pháp của RETN:

```
label: | label LABEL NEAR
...
RETN [[constant]]
```

Cú pháp của RETF:

label LABELFAR

.....

RETF [[constant]]

4.5.2. Gọi các thủ tục xa và gần

Gọi đến một thủ tục gần gọi là gọi gần (NEAR CALL), gọi đến các thủ tục xa gọi là gọi xa (FAR CALL). Có thể dùng các toán tử giả RETN (trở về gần) hay RETF (trở về xa) thay cho RET.

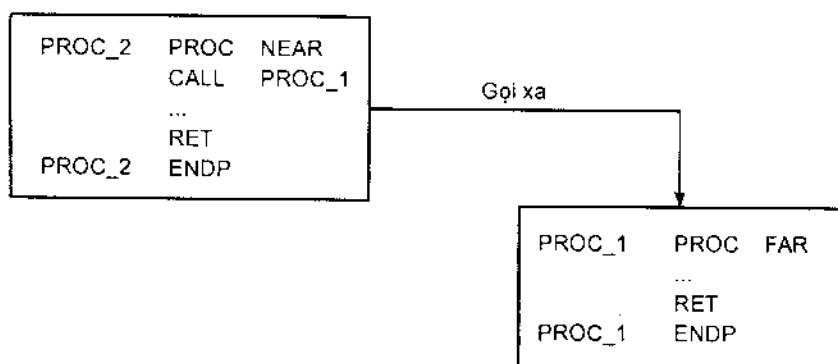
Các lệnh gọi gần bị giới hạn trong một đoạn, chúng chỉ thay đổi nội dung con trỏ lệnh (IP) mà không thay đổi nội dung của thanh ghi CS. Cũng do vậy mà các lệnh gọi gần còn được coi là gọi trong đoạn.

Các lệnh gọi xa thay đổi cả nội dung CS và IP, vì là gọi của một thủ tục trong một đoạn đến một thủ tục ở đoạn khác. Đó là gọi giữa các đoạn.

Gọi gần trong hệ 16-bit cất giữ vào ngăn xếp một từ (16-bit) địa chỉ của IP, đó là địa chỉ trở về thủ tục gọi. Lệnh trở về gần từ thủ tục bị gọi (RETN) phục hồi lại từ địa từ ngăn xếp vào IP.

Đối với gọi xa thì một từ địa chỉ là không đủ, mà cần phải cất giữ 2 từ địa chỉ: từ địa chỉ trong IP và từ địa chỉ trong CS. Lệnh trở về xa (RETF) từ thủ tục bị gọi phục hồi 2 từ địa chỉ từ ngăn xếp: một từ vào CS và từ thứ 2 vào IP. Hình 4.3 mô tả gọi xa, gần và trở về gần và xa.

Chương trình dịch assembler sử dụng khai báo thủ tục bị gọi để xác định gọi xa hay gần. Trong hình 4.3, Thủ tục gọi PROC_2 được khai báo là gần nhưng thủ tục bị gọi PROC_1 được khai báo là xa nên CALL PROC_1 trong thủ tục gần PROC_2 là gọi xa vì nó xác định theo thủ tục xa PROC_1. Ngược lại assembler lại sử dụng khai báo của thủ tục trong đó có RET để xác định trở về xa hay gần. Thủ tục PROC_1 được khai báo xa nên RET của nó là trở về xa. Còn RET trong thủ tục PROC_2 là trở về gần.



Hình 4.3: Gọi xa và trở về xa và gần

4.5.3. Ngắt là gọi đến thủ tục xa

Trong IBM PC các lệnh ngắt được thực hiện rất giống với lệnh CALL, chỉ khác là nó phát sinh từ sự kiện ngắt. Ngắt trong vi xử lý Intel từ 16-bit đến 32-bit có 2 loại. Ngắt cứng xuất hiện từ tín hiệu ngắt bên ngoài (thiết bị ngoại vi có yêu cầu được phục vụ chẳng hạn) gây ra vi xử lý thực hiện một thủ tục phục vụ yêu cầu ngắt và trở về nơi bị dừng trong chương trình đang thực hiện khi phục vụ ngắt xong. Ngắt mềm là trong chương trình hay thủ tục có lệnh ngắt INT gọi tới thủ tục phục vụ ngắt và sau đó trở về từ thủ tục phục vụ ngắt.

Giống như CALL, ngắt cũng gây ra cất giữ 2 từ địa chỉ trở về nơi đang thực hiện trước khi bị ngắt: 1 từ của CS và 1 từ của IP. Như vậy ngắt cũng là gọi xa. Thực chất là như vậy vì các ngắt chức năng DOS và BIOS đều gọi tới một vùng ROM riêng gọi là vùng BIOS có địa chỉ đoạn riêng ở vùng cao nhất của không gian nhớ của IBM PC, và đó là nơi chứa các chương trình hay thủ tục xa phục vụ các loại ngắt của hệ thống IBM PC. Nhưng khác với CALL là ngắt đòi hỏi phải cất giữ vào ngăn xếp các giá trị cờ của thanh ghi cờ trong vi xử lý trước khi cất giữ 2 từ địa chỉ. Ví dụ với ngắt chức năng DOS: sau khi thực hiện INT 21h, ngăn xếp được sắp xếp như sau:

Đỉnh (TOP) của ngăn xếp: Old IP (từ địa chỉ thứ nhất)
 Old CS (từ địa chỉ thứ hai)
 Old Flag register

Địa chỉ của thủ tục bị gọi được xác định theo số hiệu ngắt n (interrupt number) trong lệnh INT n . Như $n = 21h$ trong lệnh INT 21h. Qua số hiệu ngắt địa chỉ của thủ tục bị gọi (thủ tục phục vụ ngắt) trong vùng BIOS được xác định trong 2 từ của bảng véc-tơ ngắt (vùng 256 byte thấp nhất của bộ nhớ). Địa chỉ của 2 từ trong bảng véc-tơ ngắt được tính bằng: $n \times 4$.

Vậy INT 21h trở tới địa chỉ 84h ($4 \times 21h$). Trong 2 từ này của bảng véc-tơ ngắt chứa 2 từ địa chỉ của thủ tục phục vụ ngắt trong BIOS. Vấn đề này chúng ta xét tới khi xét về ngắt trong các vi xử Intel rồi.

Trương tự, không thể dùng RET trong các thủ tục phục vụ ngắt được mà thay vào đó là các lệnh IRET (trở về từ phục vụ ngắt). IRET phục hồi không chỉ 2 từ địa chỉ IP và CS cũ mà cả các cờ cũ.

4.5.4. Truyền dữ liệu giữa các thủ tục

Truyền dữ liệu giữa các thủ tục xảy ra khi có lời gọi từ một thủ tục đến một thủ tục khác. Có 3 cách truyền dữ liệu: dùng các biến toàn cục, truyền địa chỉ của dữ liệu nhờ các thanh ghi và sử dụng ngăn xếp.

a) Sử dụng các biến toàn cục

Toán tử giả PUBLIC và EXTERN được đặt trước tên thủ tục và là dòng lệnh khai báo thủ tục chỉ dẫn cho assembler rằng thủ tục có tên đứng sau PUBLIC hay EXTERN phải là chung hay có trong các tệp khác. PUBLIC nói rằng các tệp khác có thể gọi các thủ tục có tên đứng sau PUBLIC, còn EXTERN nói rằng thủ tục có tên đứng sau nó nằm ở trong tệp khác. Chú ý rằng PUBLIC đứng sau SEGMENT để trình liên kết (Link) kết nối các đoạn cùng tên với nhau nhưng nằm trong các tệp khác nhau thành một đoạn chung. Có thể dùng EXTERN để khai báo dữ liệu trong một thủ tục và tham chiếu đến dữ liệu đó từ một thủ tục khác.

Ví dụ 4.27:

Tạo một tệp Video_io.asm chứa 3 thủ tục gần (nằm cùng chung một đoạn mã) và được khai báo với PUBLIC. Chương trình dịch ra dạng .com.

```

        TITLE VIDEO_IO
        .MODEL      SMALL
        .CODE
        ORG         100h

TEST_WRITE_HEX PROC     NEAR ; module chính
        MOV         DL, 3Fh    ; kiểm tra với 3Fh
        CALL        WRITE_HEX
        INT         20h        ; trở về DOS
TEST_WRITE_HEX ENDP

        PUBLIC      WRITE_HEX ; thủ tục WRITE_HEX
                                ; là dùng chung

; Đây là thủ tục chuyển đổi một byte trong DL thành mã HEX
; và ghi 2 chữ số
; HEX của nó lên màn hình ở vị trí con trỏ. Thủ tục này gọi tới thủ
; tục hiển thị chữ số HEX WRITE_HEX_DIGIT
WRITE_HEX PROC     NEAR
        PUSH        CX
        PUSH        DX
        MOV         DH, DL
        MOV         CX, 4
        SHR         DL, CL
        CALL        WRITE_HEX_DIGIT ; hiển thị chữ số HEX
                                    ; đầu tiên

        MOV         DL, DH
        AND         DL, 0Fh

```

```

        CALL        WRITE_HEX_DIGIT ; hiển thị chữ số HEX
                                   ; thứ 2

        POP         DX
        POP         CX
        RET

WRITE_HEX  ENDP

        PUBLIC      WRITE_HEX_DIGIT
; WRITE_HEX_DIGIT là dùng chung thủ tục này chuyển đổi 4 bit
; thấp của DL thành một chữ số HEX và hiển thị ra màn hình. Thủ
; tục này gọi tới thủ tục hiển thị ra màn hình một ký tự
; WRITE_CHAR
WRITE_HEX_DIGIT PROC      NEAR
        PUSH        DX
        CMP         DL, 10
        JAE         HEX_LETTER
        ADD         DL, "0"
        JMP         SHORT      WRITE_DIGIT
HEX_LETTER:
        ADD         DL, "A" - 10 ; chuyển đổi thành chữ số HEX
WRITE_DIGIT:
        CALL        WRITE_CHAR
        POP         DX
        RET
WRITE_HEX_DIGIT ENDP

        PUBLIC      WRITE_CHAR ; thủ tục WRITE_CHAR là
                                   ; toàn cục
WRITE_CHAR PROC      NEAR
        PUSH        AX
        MOV         AH, 2        ; chức năng hiển thị một ký tự
        INT         21h

```

```

        POP        AX
        RET
WRITE_CHAR ENDP
        END        TEST_WRITE_HEX; kết thúc module chính

```

Trong chương trình này các thủ tục được khai báo gần nhưng có các lệnh INT 20h và INT 21h gọi tới các thủ tục của BIOS, đó là các lệnh gọi xa.

Ví dụ 4.28:

Tạo chương trình disk_io.asm đọc một séc-tơ từ đĩa mềm dùng khai báo đầy đủ các đoạn gộp thành nhóm, và toán tử EXTERN để khai báo thủ tục và biến toàn cục nằm ở tệp khác:

```

; tệp disk_io.asm
CGROUP    GROUP    CODE_SEG, DATA_SEG
        ASSUME    CS: CGROUP, DS: CGROUP
CODE_SEG  SEGMENT  PUBLIC
        EXTERN    DISP_HALF_SECTOR: NEAR
; thủ tục này đọc sector đầu tiên trên đĩa A và kết xuất một nửa
; sector lên màn hình
READ_SECTOR    PROC        NEAR
        MOV        AL, 0        ; số hiệu ổ đĩa (A)
        MOV        CX, 1        ; chỉ đọc 1 sector
        MOV        DX, 0        ; số hiệu sector
        LEA        BX, SECTOR ; nạp địa chỉ (EA) của bộ đệm
                                ; nơi lưu trữ nội dung sector
        INT        25h          ; đọc nội dung sector vào
                                ; bộ đệm
        POPF        ; phục hồi các cờ
        XOR        DX, DX      ; lập độ dịch 0 cho sector

```

```

        CALL    DISP_HALF_SECTOR
        INT     20h
READ_SECTOR    ENDP
CODE_SEG      ENDS
DATA_SEG      SEGMENT    PUBLIC
        EXTERN  SECTOR: BYTE ; biến SECTOR toàn cục
DATA_SEG      ENDS
        END     READ_SECTOR

```

Chương trình `disk_io.asm` sử dụng `EXTERN` để khai báo sử dụng một thủ tục nằm trong một tệp khác là `DISP_HALF_SECTOR` (hiển thị nội dung một nửa séc-tơ), và khai báo biến toàn cục `SECTOR` nằm trong tệp `Disp_sec.asm` để có thể tham chiếu tới nó từ các thủ tục khác. Cách dùng `EXTERN` để khai báo biến toàn cục là một trong những phương pháp truyền dữ liệu giữa các thủ tục mà ta sẽ đề cập ngay ở mục tiếp sau đây. Thủ tục `DISP_HALF_SECTOR` là thủ tục gắn nằm trong một tệp tin khác có tên là `disp_sec.asm`.

; tệp tin `disp_sec.asm`

```

CR      EQU      13          ; carriage return
LF      EQU      10          ; Line feed
CGROUP  GROUP    CODE_SEG, DATA_SEG
        ASSUME   CS: CGROUP, DS: CDROUP
CODE_SEG      SEGMENT    PUBLIC
        PUBLIC   DISP_HALF_SECTOR
DISP_HALF_SECTOR    PROC      NEAR
        XOR      DX, DX
        PUSH     CX
        PUSH     DX
        MOV      CX, 16

```

HALF_SECTOR:

```

CALL    DISP_LINE
CALL    SEND_CRLF
ADD     DX, 16
LOOP    HALF_SECTOR
POP     DX
POP     CX
RET
INT     20h

```

DISP_HALF_SECTOR ENDP

PUBLIC SEND_CRLF

SEND_CRLF PROC NEAR

```

PUSH    AX
PUSH    DX
MOV     AH, 2
MOV     DL, CR
INT     21h
MOV     DL, LF
INT     21h
POP     DX
POP     AX
RET

```

SEND_CRLF ENDP

PUBLIC DISP_LINE

DISP_LINE PROC NEAR

```

XOR     BX, BX
PUSH    BX
PUSH    CX
PUSH    DX

```

```
        MOV     BX, BX
        MOV     CX, 16
        PUSH    BX      ; cất giữ offset cho ASCII_LOOP
HEX_LOOP:
        MOV     DL, SECTOR[BX]
        CALL    WRITE_HEX
        MOV     DL, ' ' ; ghi ra dấu cách chữ giữa các số
        CALL    WITE_CHAR
        MOV     CX, 16
        POP     BX
        XOR     BX, BX
ASCII_LOOP:
        MOV     DL, SECTOR[BX]
        CALL    WRITE_CHAR
        INC     BX
        LOOP    ASCII_LOOP
        POP     DX
        POP     CX
        POP     BX
        RET
        INT     20h
DISP_LINE ENDP
        PUBLIC  WRITE_HEX
WRITE_HEX PROC    NEAR
        PUSH    CX
        PUSH    DX
        MOV     DH, DL
        MOV     CX, 4
        SHR     DL, CL
```



```

        CALL        WRITE_HEX_DIGIT ; hiển thị chữ số HEX
                                   ; đầu tiên

        MOV         DL, DH
        AND         DL, 0Fh
        CALL        WRITE_HEX_DIGIT ; hiển thị chữ số HEX
                                   ; thứ hai

        POP         DX
        POP         CX
        RET

WRITE_HEX  ENDP

        PUBLIC      WRITE_HEX_DIGIT ; WRITE_HEX_DIGIT
                                   ; là dùng chung

WRITE_HEX_DIGIT PROC      NEAR
        PUSH        DX
        CMP         DL, 10
        JAE         HEX_LETTER
        ADD         DL, "0"
        JMP         SHORT      WRITE_DIGIT
HEX_LETTER:
        ADD         DL, "A" - 10; chuyển đổi thành chữ số hex
WRITE_DIGIT:
        CALL        WRITE_CHAR
        POP         DX
        RET
WRITE_HEX_DIGIT ENDP

        PUBLIC      WRITE_CHAR ; thủ tục WRITE_CHAR là
                                   ; toàn cục

WRITE_CHAR PROC      NEAR
        PUSH        AX

```

```

        MOV     AH, 2          ; chức năng hiển thị một ký tự
        INT     21h
        POP     AX
        RET
WRITE_CHAR ENDP
CODE_SEG      ENDS
DATA_SEG      SEGMENT PUBLIC
               PUBLIC         SECTOR
SECTOR        DB    10h, 11h, 12h, 13h, 14h, 15h, 16h, 17h
               ; mẫu thử
               DB    18h, 19h, 1Ah, 1Bh, 1Ch, 1Dh, 1Eh, 1Fh
DATA_SEG      ENDS
END

```

b) Truyền dữ liệu nhờ ngăn xếp

Một phương pháp hay được dùng để truyền dữ liệu giữa các thủ tục gọi và bị gọi là sử dụng ngăn xếp và dữ liệu được thủ tục bị gọi được truy cập thông qua với cách đánh địa chỉ dữ liệu gián tiếp trong ngăn xếp. Cặp thanh ghi SS:BP dùng để đánh địa chỉ gián tiếp dữ liệu trong ngăn xếp. Chú ý rằng cặp thanh ghi SS:SP chỉ dùng để đánh địa chỉ trực tiếp dữ liệu cất giữ trong ngăn xếp.

Quá trình diễn ra như sau: trong thủ tục gọi, trước lệnh CALL, có các lệnh cất giữ (PUSH) các dữ liệu có thể cần cho thủ tục bị gọi và cho trở về từ thủ tục bị gọi. Tiếp theo, lệnh CALL trước hết cất giữ địa chỉ trở về vào đỉnh ngăn xếp và nạp địa chỉ thủ tục bị gọi để thực hiện thủ tục bị gọi. Trong thủ tục bị gọi phải có lệnh đầu tiên là cất giữ nội dung cũ của thanh ghi BP (PUSH BP), lệnh tiếp theo là phải chuyển nội dung SP vào BP (MOV BP, SP), nhờ đó mà BP trở đến đỉnh của ngăn xếp như SP. Tiếp sau là thực hiện lấy dữ liệu từ ngăn xếp (mà trước đó thủ tục gọi đã cất giữ vào ngăn xếp) ra để thực hiện nhưng

không phải bằng các lệnh POP và địa chỉ SS:SP mà bằng các lệnh bình thường với địa chỉ SS:BP. Ví dụ 4.29 dưới đây giải thích quá trình này.

Ví dụ 4.29:

; trong thủ tục gọi

```
MOV      AX, 10
PUSH     AX      ; cất vào ngăn xếp dữ liệu thứ nhất
PUSH     data2   ; cất vào ngăn xếp dữ liệu thứ hai
PUSH     CX      ; cất vào ngăn xếp dữ liệu thứ ba
CALL     addup   ; cất vào ngăn xếp địa chỉ trở về,
                  ; đỉnh con trỏ định vị tại đây, và gọi
                  ; đến thủ tục gần
```

...

; thủ tục bị gọi

```
addup     PROC      NEAR
    PUSH     BP      ; cất vào ngăn xếp giá trị của của BP
    MOV      BP, SP      ; nạp đỉnh con trỏ vào BP
    MOV      AX, [BP+4]   ; lấy từ ngăn xếp dữ liệu thứ
                          ; ba (nội dung CX)
    ADD      AX, [BP+6]   ; cộng với dữ liệu thứ hai
                          ; trong ngăn xếp, tức là
                          ; AX = CX + data2
    ADD      AX, [BP + 8] ; cộng với dữ liệu thứ nhất
                          ; trong ngăn xếp, tức là
                          ; AX = CX + data2 + 10
    POP      BP          ; phục hồi nội dung cũ của BP
    RET                          ; trở về thủ tục gọi
```

```
addup     ENDP
```

c) Truyền địa chỉ của dữ liệu nhờ các thanh ghi

Thủ tục gọi chuyển địa chỉ của các dữ liệu (mà thủ tục bị gọi sẽ dùng) vào trong các thanh ghi trước khi có lệnh CALL. Thủ tục bị gọi thực hiện lấy dữ liệu của thủ tục gọi với địa chỉ của chúng cất giữ trong các thanh ghi để thực hiện. Ví dụ 4.30 dưới đây mô tả sự thực hiện tính toán trong thủ tục bị gọi được trả về cho thủ tục gọi nhờ sử dụng sự đánh địa chỉ các dữ liệu bằng các thanh ghi SI, DI và BX.

Ví dụ 4.30:

```

; thủ tục gọi, chương trình tên là tinhtong1.asm
        TITLE tinh_tong1
        EXTERN    TINH_SUM: NEAR
        .MODEL    SMALL
        .STACK    100h
        .DATA
msg      DB      ' Enter two digits: $'
msg1     DB      0Dh, 0Ah, 'The sum of'
digit1   DB      ?
         DB      'AND'
digit2   DB      ?
         DB      'IS'
sum      DB      -30h, 'S'
        .CODE
main     PROC
        MOV     AX, @DATA
        MOV     DS, AX      ; chỉ định DS trỏ tới đoạn dữ liệu
        MOV     AH, 9
        LEA     DX, msg      ; lấy địa chỉ EA của chuỗi thông báo
                                ; msg (offset)

```

```

        INT     21h           ; hiển thị thông báo msg nhận từ bàn
                               ; phím 2 chữ số và AL để tính (digit1
                               ; và digit2)

        MOV     AH, 1

        INT     21h           ; nhận số thứ nhất
        MOV     digit1, AL

        INT     21h           ; nhận số thứ hai
        MOV     digit2, AL

        LEA     SI, digit1     ; cất địa chỉ (độ dịch) của digit1
        LEA     DI, digit2     ; cất địa chỉ của digit2
        LEA     BX, sum        ; cất địa chỉ của sum
        CALL    TINH_SUM

; hiển thị kết quả tính
        LEA     DX, msg1
        MOV     AH, 9
        INT     21h
        MOV     AH, 4Ch
        INT     21h           ; trở về DOS

main     ENDP
        END     main

; chương trình tính tên là tinhtong2.asm cùng đoạn với
; tinhtong1.asm
        TITLE   tinh_tong2
        PUBLIC   TINH_SUM
        .MODEL   SMALL
        .CODE

TINH_SUM  PROC      NEAR

        PUSH     AX
        MOV      AL, [SI]      ; lấy số thứ nhất

```

```

        ADD     AL, [DI]      ; cộng số thứ hai
        ADD     [BX], AL     ; cộng vào sum
        POP     AX
        RET
TINH_SUM  ENDP
        END

```

4.5.5. Thư viện các thủ tục

Khi có nhiều thủ tục phải sử dụng thì có thể tạo lập một thư viện các thủ tục để tiết kiệm dung lượng của chương trình nguồn. Thư viện là một tệp tin có phần mở rộng .LIB. Chương trình tham khảo tới thư viện bằng chỉ dẫn INCLUDE. Vì thư viện các thủ tục chứa các lệnh cần được thực hiện ngay nên phải chú ý đặt chỉ dẫn INCLUDE đúng nơi mà các thủ tục cần xuất hiện.

Ví dụ 4.31:

Tạo một thư viện có tên là get_one.lib chứa 3 thủ tục: get_path, get_name, scan_pa để xử lý đường dẫn tên tệp tin nhập vào từ bàn phím. Xử lý đường tên ở vùng đệm 80h do người sử dụng gõ vào hoặc nhắc người sử dụng gõ vào nếu chưa có. đặt thêm 0 ở cuối đường tên (thành chuỗi ASCII). Vùng đệm phụ được đặt sau thủ tục get_path. Kết quả ra của get_path: SI trỏ đến ký tự đầu tiên của tên tệp tin, DI trỏ đến 0 ở cuối đường tên tệp. Thủ tục chính get_path sử dụng 2 thủ tục gắn get_name và scan_pa.

```

get_path      PROC
; kiểm tra có tham số trên dòng lệnh
        MOV     DL, OFFSET buffer
        MOV     CL, [DI-1]      ; CL = số ký tự đã nhập
        MOV     CH, 0
        OR      CX, CX         ; CX = 0?
        JNZ     ck_par

```

```

no_parm:
    write_r <CR, LF, 'Hay dua vao ten tep:'>
                ; macro write_r
    MOV  DI, OFFSET buff_in
                ; trở đến vùng đệm phụ
    CALL get_name
ck_par:
    CALL scan_pa
    JC   no_parm
    RET
get_path      ENDP
handl  DW   ?                ; thẻ tệp tin
    mak_buf   buff_in, 45
                ; macro tạo vùng đệm phụ buff_in
; Thủ tục nhận đường dẫn tên tệp từ bàn phím và vùng đệm buff_in
; Input cho get_name: DI trở đến đầu của buff_in
; Output từ get_name: DI trở đến ký tự đầu tiên của tên, CX= số ký
; tự đã nhập
get_name      PROC
    read_bf    DI
; macro đọc vùng đệm bàn phím có địa chỉ tại DS:dia_chi?
    write_r <CR, LF>
    INC  DI                ; bỏ qua số ký tự tối đa
    MOV  CL, [DI]          ; CL= số ký tự thực sự nhập
    MOV  CH, 0
    INC  DI                ; DI trở đến ký tự nhập đầu tiên
get_name      ENDP
; Thủ tục xử lý đường tên: bỏ đi các khoảng trống thừa, thêm 0
; vào cuối đường tên (trở thành chuỗi ASCIIZ)

```

; Input cho scan_pa: DI trở đến ký tự đầu tiên của đường tên
 ; CX = số ký tự đã nhập
 ; Output từ scan_pa: SI trở đến ký tự đầu tiên của tên, DI trở đến 0
 ; ở cuối tên cờ CF = 1 khi có lỗi

scan_paPROC

```

      MOV  AL, blank
      REPE SCASB      ; tìm ký tự đầu tiên khác khoảng trống
      DEC  DI          ; DI trở đến ký tự này
      INC  CX          ; CX = số ký tự
      MOV  SI, DI
      CMP  AL, [SI]
      JZ   no_parms   ; khi toàn là khoảng trống
      REPNE SCASB
      JNZ  sdi         ; nếu là CR
      DEC  DI          ; blank

```

sdi:

; thêm 0 vào cuối đường tên

```

      MOV  [DI], CH
      INC  DI
      CLC
      RET

```

; đặt cờ CF = 1 khi lỗi đường tên

no_parms:

```

      STC
      RET

```

scan_paENDP

; Kết thúc thư viện ở đây

Chúng ta sẽ khảo sát cách chương trình chính tham khảo đến thư viện các thủ tục như thế nào khi nói tới MACRO.

4.6. MACRO VÀ CÁC VẤN ĐỀ LIÊN QUAN

Macro là tên tượng trưng gán cho một chuỗi các ký tự (macro văn bản) hoặc cho một hoặc một số các phát biểu (các thủ tục hoặc hàm macro). Chương trình dịch assembler khi duyệt các dòng của chương trình nguồn, nếu gặp tên macro, nó sẽ thay thế tên macro bằng nội dung văn bản của macro đó, tức là mở rộng macro, như vậy, người lập trình tránh phải viết một số nội dung trùng lặp một số lần trong chương trình nguồn. Tên của macro thường gắn liền với nội dung của macro.

Các macro văn bản mở rộng văn bản bên trong một phát biểu của chương trình nguồn.

Các thủ tục macro (thường gọi là các macro) mở rộng thêm thành một hoặc một số phát biểu và có thể nhận các thông số.

Các hàm macro giống các thủ tục macro và có thể được sử dụng giống như các macro văn bản nhưng có trả về giá trị.

Các macro của MASM có các toán tử như sau:

- < > mở và đóng chuỗi văn bản
- ! coi ký tự đứng sau như là ký tự văn bản
- % để mở rộng thêm một biểu thức hằng số hay một macro văn bản
- & để thay thế một thông số của macro hay macro văn bản bằng giá trị thực của chúng.

4.6.1. Các macro văn bản

Chúng ta có thể gán tên tượng trưng cho một chuỗi các ký tự và sau đó sử dụng tên đó đặt trong văn bản của chương trình nguồn. Toán tử giả TEXEQU được dùng để khai báo một macro văn bản, cú pháp như sau:

```
name    TEXEQU    <text>
name    TEXEQU    macrold|textmacro
name    TEXEQU    %constExpr
```

Trong đó, text là văn bản - một chuỗi các ký tự được đóng trong cặp dấu < >, macroId là hàm macro được khai báo trước, textmacro là macro văn bản đã được khai báo trước, và %constExpr là một biểu thức giá trị của văn bản. Dấu % mở rộng các macro văn bản hay chuyển đổi các biểu thức hằng số thành các biểu diễn văn bản. Dưới đây là một số ví dụ:

Ví dụ 4.32:

```
msg    TEXTEQU    <text> ; văn bản được gán cho tên msg
string TEXTEQU    msg  ; macro văn bản gán cho tên string
msg    TEXTEQU    <some other text>
                        ; văn bản mới gán cho tên msg
value  TEXTEQU    %(3 + num)
                        ; giá trị của biểu thức thể hiện văn
                        ; bản gán cho value
errstr TEXTEQU    <expresion! >255>
; errstr = "expresion > 255, dấu ! cho phép đưa < > như là một
; phần của giá trị văn bản
```

Các macro văn bản thuận tiện cho đặt tên các chuỗi văn bản không đánh giá trị nguyên. Ví dụ, có thể sử dụng một macro văn bản để đặt tên một hằng số dấu phẩy động hoặc một biểu thức trong ngoặc. Dưới đây là ví dụ.

Ví dụ 4.33:

```
pi      TEXTEQU    <3.1416>          ; pi = <3.1416>
WPT     TEXTEQU    <WORD PTR>        ; một chuỗi từ khóa
                        ; được đặt tên
arg1     TEXTEQU    <[bp + 4]>        ; biểu thức trong ngoặc vuông
                        ; được gán tên.
a        TEXTEQU    <3+4>            ; a = "3 + 4"
b TEXTEQU    %3 + 4                  ; b = "7"
c TEXTEQU    %(3 + 4)                ; c = "7"
```

Cần phân biệt các chỉ dẫn EQU để chỉ định giá trị hằng số và sử dụng macro văn bản, như trường hợp sau đây:

```
num    EQU    4                ; num = 4
numstr TEXT EQU    <4>        ; numstr = <4>
aTEXT EQU    %3 + num        ; a = <7>
bTEXT EQU    %3 + numstr     ; b = <7>
```

4.6.2. Các thủ tục macro

Nếu chương trình phải thực hiện một nhiệm vụ nhiều lần thì ta có thể tránh phải viết từng ấy lần những phát biểu để thực hiện trong chương trình bằng cách chỉ cần viết một thủ tục macro thay thế. Chẳng hạn, khi thực hiện với nhiều thủ tục, hay chương trình con, chúng thường phải thực hiện cất giữ một số nội dung các thanh ghi vào ngăn xếp nhiều lần, như:

```
PUSH AX
PUSH BX
PUSH CX
PUSH DX
```

Khi đó ta thực hiện tạo một thủ tục macro như sau:

```
push_abcd    MACRO
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
ENDM
```

Các thủ tục macro, hay thường gọi đơn giản là các macro, là cơ chế tự động phát sinh những văn bản lặp lại. Nói đến thủ tục macro, tức là phải có gọi đến thủ tục macro, và khai báo một thủ tục macro cũng tương tự như khai báo một thủ tục.

a) Khai báo các macro (thủ tục macro)

Cú pháp khai báo một thủ tục macro như sau:

```
name    MACRO  val1, val2,..., valn
...
;; các phát biểu
ENDM
```

Phần bình luận của thủ tục macro phải bắt đầu bằng dấu ;;

Trong đó, name là tên của macro, val1, val2,..., valn là danh sách các biến giả tùy chọn.

Ví dụ 4.34:

Đưa ra một chuỗi ký tự, chuỗi ký tự là một biến macro.

```
write_msg    MACRO    chuoi_kt
              LOCAL    msg, kthuc
              PUSH     AX
              PUSH     DX
              MOV      AH, 9
              MOV      DX, OFFSET msg
              INT      21h
              POP      DX
              POP      AX
              JMP      SHORT kthuc
msg           DB      chuoi_kt.'$'
kthuc:
              ENDM
```

Ví dụ 4.35:

Chương trình cần đưa ra tiếng beep mỗi khi xuất hiện lỗi nào đó.

Như vậy có thể viết một thủ tục macro xác định tiếng beep.

```

beep    MACRO
        MOV    AH, 2  ;; chức năng DOS đưa ra màn hình mã ký tự
                        ;; trong DL
        MOV    DL, 7  ;; chọn mã ASCII là 7 (bell)
        INT    21h    ;; gọi chức năng DOS
        ENDM

```

Ví dụ 4.36:

Trở về DOS bằng INT 20h

```

go_dos  MACRO
        INT    20h
        ENDM

```

Các khai báo các macro nên để ở đầu chương trình. Một khi đã khai báo thủ tục macro, thì chúng ta có thể thực hiện thông qua tên của chúng ở bất kỳ điểm nào trong chương trình bằng cú pháp gọi như sau:

```
name    r1, r2,..., rn
```

Trong đó, name là tên của macro đã được khai báo trước và r1, r2,..., rn là danh sách các biến thực. Chẳng hạn, ta có thể thực hiện gọi tới thực hiện 2 lần thủ tục macro beep cho trong ví dụ khi có lỗi:

```

.IF     error
beep
beep
.ENDF

```

Hoặc gọi tới macro cho trong ví dụ 4.34 để đưa ra một dòng thông báo trên màn hình, thì thực hiện dòng lệnh:

```
write_msg    'Hello, world of the Microsoft macro'
```

b) Định nghĩa các nhãn cục bộ trong các macro

Toán tử giả LOCAL được dùng để định nghĩa các nhãn cục bộ trong macro ngay sau dòng MACRO. Ví dụ 4.34 có định nghĩa các

nhân msg và kthuc là các nhãn cục bộ trong macro write_msg. Hoặc ví dụ 4.37 dưới đây:

Ví dụ 4.37:

```
power  MACRO      factor:REQ, exponent:REQ
      LOCAL      again, gotzero ;; các ký hiệu cục bộ
      SUB        DX, DX          ;; xóa đỉnh
      MOV        AX, 1           ;; nhân 1 ở vòng lặp đầu tiên
      MOV        CX, exponent    ;; nạp số đếm
      JCXZ       gotzero        ;; nhảy nếu số mũ = 0
      MOV        BX, factor      ;; nạp hệ số

again:
      MUL        BX              ;; nhân hệ số kết quả
                                   ;; trong AX
      LOOP       again

gotzero:
      ENDM
```

c) Macro tham chiếu đến macro khác

Một macro có tham chiếu đến một macro khác, ví dụ, nếu có một macro cất vào ngăn xếp các thanh ghi và một macro phục hồi các thanh ghi thì có thể có một macro thứ ba có 2 lệnh tham chiếu cả 2 macro trên bên trong nó để thực hiện:

Ví dụ 4.38:

```
; macro cất vào ngăn xếp
push_regs  MACRO      R1, R2, R3
      PUSH R1
      PUSH R2
      PUSH R3
      ENDM
```

; macro phục hồi từ ngăn xếp

```
pop_regs    MACRO    S1, S2, S3
            POP  S1
            POP  S2
            POP  S3
            ENDM
```

; macro có trong 2 macro trên

```
copy    MACRO    source, destination, length
        push_regs    CX, SI, DI
        LEA    SI, source
        LEA    DI, destination
        CLD
        MOV    CX, length
        REP    MOVSB
        pop_regs    DI, SI, CX
        ENDM
```

d) Định nghĩa các thông số yêu cầu và mặc định của thủ tục macro

Các thông số đứng sau MACRO có thể có các thuộc tính đặc biệt để làm cho chúng linh hoạt hơn và nâng cao hiệu quả xử lý lỗi cho assembler. Chúng ta có thể gán cho thông số (parameter) thuộc tính là yêu cầu, các giá trị mặc định, hoặc các số khác nhau (thuộc tính biến).

Cú pháp đặt thông số có thuộc tính yêu cầu là:

parameter:REQ

Nếu lệnh gọi đến macro không có sự trùng của tham số yêu cầu được định nghĩa trong macro thì assembler sẽ thông báo lỗi ở dòng chứa tham chiếu đến macro. REQ là giá trị mặc định yêu cầu, nó có thể cải thiện sự tổng hợp lỗi.

Ví dụ 4.39:

```

writechar    MACRO    char:REQ
              MOV  AH, 2
              MOV  DL, char
              INT   21h
              ENDM

```

Cú pháp gán cho thông số một giá trị mặc định là:

parameter:=textvalue

textvalue phải đặt trong cặp dấu < > và chỉ cho assembler xác nhận đó là giá trị mặc định. Cho rằng chúng ta thường đưa mã ASCII 7 (bell) - đưa ra tiếng beep. Khi đó khai báo macro sử dụng dấu = để bảo assembler phải chỉ định thông số char bằng 7, ví dụ dưới đây là đưa ra tiếng beep:

```

writechar    MACRO    char := <7>
              MOV  AH, 2
              MOV  DL, char
              INT   21h
              ENDM

```

Nếu trong lệnh tham chiếu đến macro này không chứa một biến char thì assembler điền vào chỗ trống (blank) ứng với vị trí của biến giá trị mặc định là 7 và macro phát sinh beep khi được gọi tới.

Cú pháp gán cho thông số một biến:

parameter:VARARG

Toán tử VARARG xác định thuộc tính biến (arg) của thông số trong khai báo macro:

```

work    MACRO    rarg:REQ, darg:=<5>, varg:VARARG
...
        ENDM

```


e) Sử dụng các toán tử lặp cho macro

MASM có một số toán tử giả thực hiện lặp. Các khối chỉ dẫn được thực hiện lặp đi lặp lại có thể được sử dụng bên ngoài macro, nhưng chúng cũng thường có trong các macro để thực hiện một số lặp thao tác bên trong macro. Như vậy, bản thân các khối lặp đó cũng được coi là các macro, và chúng cũng được định dạng có ENDM kết thúc.

Có bốn toán tử giả thực hiện lặp, đó là: REPEAT, WHILE, FOR, và FORC. Trong các MASM phiên bản từ 6.0 về trước, thì REPEAT được gọi là REPT, FOR được gọi là IRP, và FORC được gọi là IRPC. Trình biên dịch MASM 6.1 hay MASM32 công nhận những sử dụng cũ.

Lặp REPEAT (REPT):

Dùng để lặp lại thực hiện một số câu lệnh trong macro. Cú pháp như sau:

```
REPEAT      constexpr
           statements      ;; các câu lệnh
ENDM
```

Trong đó, constexpr có thể là một hằng số hoặc biểu thức hằng số.

Ví dụ 4.40:

Macro đưa ra beep 3 lần khi có lỗi.

```
beep  MACRO      iter:=<3>
      MOV  AH, 2      ;; chức năng đưa ra màn hình mã ký
                        ;; tự trong DL
      MOV  DL, 7      ;; ký tự Bell
      REPEAT  iter    ;; số lần lặp được khai báo ở macro
      INT  21h        ;; gọi chức năng DOS
      ENDM
      ENDM
```

Ví dụ 4.41:

Tạo mảng n từ với các giá trị $1!, 2!, \dots, n!$, chỉ ra cách tham chiếu đến mảng.

```
factorials      MACRO      N
    M = 1
    FAC = 1
    REPEAT      N
    DW  FAC
    M = M + 1
    FAC = M*FAC
    ENDM
    ENDM
```

Lặp WHILE:

Toán tử WHILE tương tự như REPEAT, nhưng vòng lặp của nó tiếp tục cho đến khi nào điều kiện thỏa mãn. Cú pháp là:

```
WHILE expression
    statements
ENDM
```

Trong đó, expression là biểu thức giá trị có thể assembler tính được trong thời gian biên dịch. Bình thường biểu thức giá trị sử dụng các toán tử quan hệ, nhưng có thể là bất kỳ biểu thức nào được đánh giá là 0 nếu không đúng (false) và không bằng 0 nếu đúng (true).

Ví dụ 4.42:

Tính căn bậc ba.

```
cubes LABELBYTE      ;; đặt tên dữ liệu
    root = 1          ;; khởi tạo căn
    cube = root * root * root  ;; lũy thừa ba lần thứ nhất
    WHILE cube LE 32767  ;; lặp lại cho đến khi kết quả
                        ;; quá lớn
```

```

WORD      cube
root = root + 1      ;; căn và lũy thừa tiếp theo
cube = root * root * root
ENDM

```

Lặp FOR (IRP):

Với FOR ta có thể thực hiện lặp thông qua một danh sách các biến (argument list). Cú pháp là:

```

FOR      parameter, <argumentlist>
statements
ENDM

```

Trong đó, parameter thể hiện tên của từng biến bên trong khối FOR. Danh sách các biến phải chứa dấu phẩy phân cách các biến và phải đóng bên trong cặp dấu < >.

Ví dụ 4.43:

Tạo mảng các byte giá trị từ 1 đến 10.

```

series LABELBYTE
FOR      arg, <1,2,3,4,5,6,7,8,9,10> ;; danh sách gồm 10 biến
BYTE      arg      DUP (arg)
ENDM

```

Trong lần lặp đầu tiên của macro này, biến arg đầu tiên của thông số thay bằng giá trị 1. Lần lặp thứ hai, arg được thay bằng 2. Kết quả là một mảng với các byte giá trị 1, 2,..., 10.

Toán tử FOR cũng có một cách cho thuận tiện xử lý macro nhờ một số thay đổi của các biến. Để thực hiện, bổ sung thêm toán tử VARARG (biến) cho thuộc tính biến nằm cuối cùng của khai báo thông số để chỉ rằng thông số có các giá trị biến thực. Ví dụ sau đây là khai báo macro với ba thuộc tính của thông số - yêu cầu, mặc định và biến. Thuộc tính biến luôn đặt ở cuối cùng trong khai báo thông số (xem mục d, và ví dụ 4.39):

```
work    MACRO      rarg:REQ, darg := <5>, darg:VARARG
...
        ENDM
```

Tham chiếu đến macro work:

```
work    4, , 6, 7, a, b
```

chỉ ra rằng, tham số rarg = 4, ở vị trí đầu tiên, tham số darg có giá trị mặc định là 5 nên trong tham chiếu vị trí này để trống (blank), còn lại bốn giá trị 6, 7, a, b được coi là một biến đơn đóng trong <6, 7, a, b>. Cách này cũng tương tự như là sử dụng FOR.

Ví dụ 4.44:

```
show    MACRO      chr:VARARG
        MOV        AH, 2
        FOR arg, <chr>
        MOV DL, arg
        INT 21h
        ENDM
        ENDM
```

Khi đó lời gọi macro này như sau:

```
show    'O','K',13,10
```

Thông số bên trong khai báo FOR có thể có thuộc tính yêu cầu (REQ), giá trị mặc định, và biến (VARARG). Ví dụ 4.44 có thể thay đổi để làm cho các biến trống (blank) phát sinh lỗi:

Ví dụ 4.45:

```
show    MACRO      chr:VARARG
        MOV AH, 2
        FOR arg:REG, <chr>
        MOV DL, arg
        INT 21h
        ENDM
        ENDM
```



```

BYTE '&arg' + 20h  ;; phân bổ chữ nhỏ
BYTE '&arg' - 40h   ;; phân thứ tự của chữ
ENDM

```

f) Sử dụng các toán tử điều kiện cho macro

Có thể dùng các toán tử điều kiện như IF, IFE, IFB, IFNB,... để bỏ qua các đối số định hay lệnh trong macro. Các toán tử điều kiện này có thể dùng ở mọi chỗ trong chương trình nguồn, nhưng chủ yếu dùng trong các macro. Dạng cơ bản chung của các toán tử điều kiện là:

```

conditional
    statements ;; các câu lệnh
ENDIF

```

và:

```

conditional
    statements1
ELSE
    statements2
ENDIF

```

Trong dạng thứ nhất nếu điều kiện (conditional) là đúng (True) thì các câu lệnh (statements) được thực hiện, nếu điều kiện sai (False) thì không có câu lệnh nào được thực hiện.

Trong dạng thứ hai nếu điều kiện đúng thì các câu lệnh 1 (statements1) được thực hiện, nếu điều kiện sai thì các câu lệnh 2 (statements2) được thực hiện.

Bảng 4.2 liệt kê các dạng điều kiện (conditional type).

Bảng 4.2: Các toán tử điều kiện cho macro

Dạng điều kiện (IF)	Điều kiện đúng (True)
IF expresion	Hằng biểu thức khác 0
IFE expresion	Hằng biểu thức bằng 0
IFB <arg>	Không có biến arg là blank (trống)

Dạng điều kiện (IF)	Điều kiện đúng (True)
IFNB <arg>	Có biến arg không phải là blank
IFDEF sym	Ký hiệu sym được định nghĩa trong chương trình (hay được khai báo bằng EXTERN)
IFNDEF sym	Sym không được định nghĩa hay khai báo bằng EXTERN
IFIDN <str1>, <str2>	Chuỗi str1 và str2 giống nhau
IFDIF <str1>, <str2>	Chuỗi str1 và str2 không giống nhau
IF1	Trình biên dịch assembler tiến hành lần duyệt thứ nhất
IF2	Trình biên dịch assembler tiến hành lần duyệt thứ hai

Điều kiện IF: biểu thức khác 0 thì thực hiện các lệnh sau.

Ví dụ 4.48:

Sử dụng toán tử IF để khai báo một macro định nghĩa một mảng n từ chứa k số nguyên đầu tiên theo sau là n - k số 0.

```

block    MACRO      n, k
        i = 1
        REPEAT      n
        IF          k + 1 - i
                DW    i
        ELSE
                DW    0
        ENDIF
        ENDM
        ENDM

```

Sử dụng macro này để tạo một mảng từ với 10 giá trị: 1, 2, 3, 5, 5, 0, 0, 0, 0, 0. Thực hiện lời gọi tới macro này như sau:

```

A        LABELWORD
block    10, 5

```

Mở rộng macro sẽ khởi tạo giá trị 10 cho n, 5 cho k, sau đó biên dịch tạo ra 10 câu lệnh DW, trong đó 5 câu lệnh đầu DW 1 ... DW5

còn thỏa mãn điều kiện $k + 1 - i = 5 + 1 - i > 0$. Với $i = 6$ điều kiện $5 + 1 - 6 = 0$ nên thực hiện năm lệnh DW 0 (cho đủ lặp $n = 10$). Kết quả cuối cùng là:

A DW 1, 2, 3, 4, 5, 0, 0, 0, 0, 0

Điều kiện IFNB: có biến không phải trống (blank).

Ví dụ 4.49:

Macro sau đây khai báo các tham số với các thuộc tính: yêu cầu (REQ), giá trị mặc định, và 4 biến để cuộn (scroll) cửa sổ trên màn hình.

```
Scroll    MACRO distance:REG, attrib := <7>, tcol, trow, bcol, brow
IFNB    <tcol>                            ;; bỏ qua các biến nếu có trống (blank)
         MOV    cl, tcol
ENDIF
IFNB    <trow>
         MOV    CH, trow
ENDIF
IFNB    <bcol>
         MOV    DL, bcol
ENDIF
IFNB    <brow>
         MOV    DH, brow
ENDIF
IFDIF <attrib>, <BH>                    ;; không nạp BH vào chính nó
         MOV    BH, attrib
ENDIF
IF    distance LE 0                    ;; âm cuộn lên, dương cuộn xuống
         MOV    AX, 0600h + (-(distance) AND 0FFh)
```


ELSE

MOV AX, 0700h + (distance AND 0FFh)

ENDIF

INT 10h

ENDM

Trong macro này, distance là thông số yêu cầu (REQ), thông số attrib có giá trị mặc định là 7 (trắng trên đen), nhưng macro cũng kiểm tra để đảm bảo rằng biến tương ứng không có trong BH, bởi vì sẽ không có tác dụng nếu BH tự nạp vào chính nó. Toán tử IFNB được sử dụng để kiểm tra các biến trống. Nó cho phép xử lý các giá trị hàng và cột trực tiếp trong các thanh ghi CX và DX khi thực hiện chương trình. Lời gọi đến macro này có thể như sau:

; cho rằng DL và CL đã được nạp các giá trị

DEC DH ; giảm hàng phía đỉnh màn hình

INC CH ; tăng hàng phía đáy màn hình

Scroll -3 ; cuộn cửa sổ động trắng trên đen của màn
; hình lên trên 3 dòng

Scroll 5, 17h, 2, 2, 14, 12
; cuộn cửa sổ tĩnh trắng trên xanh da trời
; xuống dưới 5 dòng

Trong lời gọi Scroll -3, giá trị khoảng cách (distance) yêu cầu (REQ) phải cuộn cửa sổ là -3, giá trị thuộc tính (attrib) mặc định là 7 được nạp vào thanh ghi BH bằng lệnh: MOV BH, attrib, còn giá trị thực cho các biến tccl, trow, bcol, brow không có trong lời gọi, tức là chúng được để trống. Do đó các lệnh IFNB không thỏa mãn điều kiện, và cửa sổ động trắng trên đen thực đơn cuộn nhờ các lệnh trong macro:

IF distance LE 0 ;; âm cuộn lên, dương cuộn xuống
MOV AX, 0600h + (-(distance) AND 0FFh)

Trong lời gọi Scroll 5, 17h, 2, 2, 14, 12, tất cả các tham số đều được cho giá trị, không có bỏ trống, và với distance = 5 thì sự cuộn cửa sổ tính (kích thước cửa sổ được định trong lời gọi) thực hiện với các lệnh:

```

IFNB    <tccl>
        MOV    cl, tccl        ;; nạp giá trị 2
ENDIF
IFNB    <trcw>
        MOV    CH, trcw        ;; nạp giá trị 2
ENDIF
IFNB    <bcol>
        MOV    DL, bcol        ;; nạp giá trị 14
ENDIF
IFNB    <brow>
        MOV    DH, brow        ;; nạp giá trị 12
ENDIF
IFDIF    <attrib>, <BH>        ;; giá trị 14h khác với giá trị mặc định 7
        MOV    BH, attrib        ;; thì nạp thuộc tính 14h vào BH,
                                ;; trắng trên xanh da trời
ENDIF
IF    distance LE 0        ;; không thỏa mãn vì 5 > 0
ELSE        ;; thực hiện lệnh này
        MOV    AX, 0700h + (distance AND 0FFh)
ENDIF
        INT    10h
ENDM

```

4.6.3. Các hàm macro

Một hàm macro là một nhóm có tên gồm các câu lệnh (statements) có trả về giá trị. Khi gọi tới hàm macro, ta cần phải đặt danh sách các biến của hàm vào trong cặp dấu ngoặc đơn (), ngay cả khi danh sách rỗng.

MASM 6.1 cung cấp một số các hàm macro xác định sẵn của nó và lời gọi đến các hàm macro xác định sẵn phải có ký hiệu @ đứng trước tên hàm và các biến. Một số phiên bản hợp ngữ cũ sử dụng @ đặt trước tên macro để gọi tới macro. Khai báo các hàm macro cũng giống như khai báo các thủ tục macro (hay macro), nhưng khác là trong hàm để trả về giá trị luôn phải có toán tử EXITM.

Ví dụ 4.50:

```

DEFINED      MACRO      symbol:REQ
IFDEF  symbol      ;;
              EXITM <-1>  ;; True
ELSE
              EXITM <0>   ;; False
ENDIF
ENDM

```

Ví dụ 4.51:

Sử dụng WHILE trong hàm macro để tính giai thừa:

```

factorial MACRO      num:REQ
              LOCAL   i, FAC
              FAC = num
              i = 1
WHILE  FAC      GT    1
              i = i * FAC
              FAC = FAC - 1
ENDM
EXITM  %i
ENDM

```

Kết quả nguyên của phép tính được đổi thành một chuỗi văn bản nhờ toán tử %. Macro giai thừa có thể xác định dữ liệu như sau:

```
var      WORD      factorial (4)
```

Như vậy câu lệnh này khởi tạo biến var bằng số 24 (giá trị trả về của tính giai thừa của 4)

Ví dụ 4.52:

Có thể sử dụng FOR để xử lý các thông số với thuộc tính VARARG trong hàm macro xác định trước của MASM 6.1:

```
@ArgCount    MACRO      arglist:VARARG
                ;; đây là cú pháp khai báo hàm

LOCAL        count
count = 0
FOR   arg, <arglist>
        count = count + 1    ;; tính số biến
ENDM
EXITM %count
ENDM
```

Hoặc trong hàm xác định trước @ArgI:

Ví dụ 4.53:

```
@ArgI MACRO      index:REG, arglist:VARARG
LOCAL        count, retstr
retstr TEXT EQU    <>    ;; khởi tạo count
count = 0          ;; khởi tạo chuỗi trả về
FOR   arg, <arglist>
        count = count + 1
        IF    count EQ    index
            retstr TEXT EQU    <arg>
        EXITM
    ENDIF
ENDM
EXITM retstr        ;; thoát khỏi hàm
ENDM
```

4.6.4. Thư viện macro

Trong thực tế, người lập trình phải gọi đến nhiều macro khác nhau, và nếu như các macro nằm rải rác ở nhiều nơi khác nhau thì sẽ không hiệu quả cho viết chương trình nguồn cũng như khi tham chiếu đến các macro. Các macro có liên quan chức năng với nhau có thể đặt chung trong một tệp riêng, như thế để tham chiếu đến tất cả các macro chỉ cần có lệnh ghép (INCLUDE) tệp thư viện macro đó vào chương trình nguồn đặt ở đầu chương trình, và câu lệnh tham chiếu đến các macro có thể đặt ở bất kỳ chỗ nào trong chương trình.

Thư viện macro là một tệp tin có tên phần mở rộng là .MAC. Trong tệp thư viện các dòng bình luận đầu tiên là danh sách các macro - gồm tên, các tham số (nếu có) và công dụng. Các khai báo của các macro có trong danh sách được đặt trong cấu trúc điều kiện:

IF1

... ; các khai báo các macro trong thư viện

ENDIF

Một khi thư viện macro được ghép vào chương trình chính (bằng INCLUDE), tất cả các khai báo macro của thư viện sẽ xuất hiện trong tệp .LST do assembler tạo ra, thậm chí ngay cả khi thư viện được tham chiếu đến trong chương trình chính. Để tránh điều này, các khai báo macro phải đặt trong khung điều kiện IF1 ... ENDIF. Như vậy IF1 chỉ dẫn assembler chỉ truy cập thư viện macro trong lần biên dịch thứ nhất khi các macro được mở rộng mà không truy cập thêm nữa trong biên dịch lần thứ hai khi tệp .LST được tạo ra. Ví dụ sau đây tạo một thư viện macro tên là mylib.mac gồm một số macro.

Ví dụ 4.54:

```
; Thư viện Macro cho MASM, tên tệp là mylib.mac
; Các macro trong thư viện
; head  MACRO tiêu_de?: cài tiêu đề chương trình trong bộ nhớ
; go_dos      MACRO: trở về DOS bằng INT 20h
```

```

; write_msg    MACRO   chuoi_kt?: đưa ra màn hình chuỗi ký tự
; read_bf     MACRO   dia_chi?: đọc vùng đệm bàn phím có địa chỉ
; tại DS:dia_chi?
; mak_buf     MACRO   ten?,do_dai?: tạo 1 vùng đệm nhập từ bàn phím
; d_char      MACRO   par?: đưa 1 ký tự ra màn hình
; write_r     MACRO   dia_chi?: đưa ra màn hình chuỗi ký tự ở dia_chi?
; d_write     MACRO   the_file?,con_tro?: ghi 1 tệp tin có thể là
; the_file?
; d_read      MACRO   the_file?,con_tro? : đọc 1 tệp tin có thể là
; the_file?
; create      MACRO   con_tro? : tạo 1 tệp tin trên đĩa
; open        MACRO   con_tro?,den_dau?: mở 1 tệp tin trên đĩa
; close       MACRO   the_file?: đóng tệp tin có tên là the_file?
; Thân của thư viện macro
        IF1
; Các hằng số của thư viện
blank    EQU    32          ; dấu cách hay khoảng trống
comma    EQU    44          ; dấu phẩy
CR       EQU    13          ; về đầu dòng
LF       EQU    10          ; chuyển dòng
EOF      EQU    1Ah         ; kết thúc tệp tin
esc      EQU    1Bh         ; escape
period   EQU    46          ; dấu chấm
tab      EQU    9           ; nhảy cách 8 khoảng trống
; Khai báo macro head
head     MACRO             tieu_de?
        LOCAL             qua_data
        JMP               SHORT qua_data
        DB                tieu_de?

```

```

qua_data:
    ENDM
; Khai báo macro go_dos
go_dos MACRO
    INT    20h
    ENDM
; Khai báo macro write_msg
write_msg    MACRO    chuoi_kt?
    LOCAL    msg, kthuc
    PUSH     AX
    PUSH     DX
    MOV      AH, 9
    MOV      DX, OFFSET msg
    INT      21h
    POP      DX
    POP      AX
    JMP      SHORT kthuc
msg    DB    chuoi_kt.'$'
kthuc:
    ENDM
; Khai báo macro read_buf
read_buf MACRO    dia_chi?
    PUSH     DX
    MOV      DX, dia_chi?
    MOV      AH, 0Ah
    INT      21h
    POP      DX
    ENDM
; Khai báo macro mak_buf

```

```

mak_buf MACRO    ten?, do_dai?
ten?    DB        do_dai?        ; số ký tự tối đa
        DB        ?            ; các số ký tự thực nhập
        DB        do_dai?DUP(?); tạo vùng đệm
        DB        CR            ; ký tự về đầu dòng
        ENDM

; Khai báo macro d_char
; Cú pháp:
        d_char    AL            ; in ra nội dung thanh ghi AL
        d_char    '*'          ; in ra hằng số
        d_char    ; in ra nội dung DL
d_char  MACRO    par?
        IFNB      <par?>
        PUSH      DX
        MOV        DL,par?
        ENDIF
        MOV        AH,2
        INT        21h
        IFNB      <par?>
        POP        DX
        ENDIF
        ENDM

; Khai báo macro write_r
write_r  MACRO    dia_chi?
        PUSH      DX
        MOV        AH, 9
        MOV        DX,dia_chi?
        INT        21h

```



```

        POP        DX
        ENDM

; Khai báo macro d_write
; ghi tệp tin có tên the_file?, con_tro? trở đến vùng đệm chứa
; thông tin cần ghi
        PUSH       DX
        MOV        DX, con_tro?
        IFNB       <the_file?>
        MOV        BX, the_file?
        ENDIF
        MOV        AH, 40h
        INT        21h
        POP        DX
        ENDM

; Khai báo macro d_read
; đọc từ đĩa với thẻ tệp tin đã được gán trước
; con_tro? trở đến vùng đệm, CX chứa số byte cần đọc từ đĩa,
; cờ CF = 1 khi có lỗi đọc
        PUSH       DX
        MOV        DX, con_tro?
        IFNB       <the_file?>
        MOV        BX, the_file?
        ENDIF
        MOV        AH, 3Fh
        INT        21h
        POP        DX
        ENDM

; Khai báo macro create
; con_tro? là địa chỉ của đường dẫn tên tệp.

```

; đường dẫn tên phải kết thúc bằng 0

; thẻ tệp hay mã lỗi nằm trong BX

; cú pháp: create [DI]

create <OFFSET vùng_dem>

PUSH DX

PUSH CX

MOV DX, con_tro?

XOR CX, CX

MOV AH, 3Ch

INT 21h

POP CX

POP DX

MOV BX, AX

ENDM

; Khai báo macro open

; mở tệp tin trên đĩa để chỉ đọc

; BX chứa thẻ tệp the_file? hoặc mã lỗi

; con_tro? trở đến vùng đệm chứa thông tin cần ghi

PUSH DX

MOV DX, con_tro?

MOV AL, 0 ; chỉ đọc, không ghi

MOV AH, 3Dh

INT 21h

MOV BX, AX ; thẻ tệp

POP DX

IFNB <den_dau?>

JC den_dau?

ENDIF

ENDM

ENDIF

Bây giờ tạo một tệp chương trình chính đưa ra thông báo mong muốn. Tệp có tên là `hellos.asm`. Trong tệp này để sử dụng được các macro `go_dos` và `write_msg` của thư viện macro cần phải có dòng toán tử giả:

```
INCLUDE      mylib.mac
```

Ví dụ 4.55:

```
TITLE loi_chao
.MODEL      SMALL
INCLUDE     mylib.mac
.CODE
ORG 100h

start:

write_msg   'Hello, world of the macro assembler'
go_dos
END start
```

Cũng có thể tạo một thư viện macro gồm các khai báo các macro nhưng không đóng khuôn trong cấu trúc điều kiện `IF1 ... ENDIF`. Khi đó trong chương trình phải có đưa vào dòng lệnh `INCLUDE mylib.mac` trong cấu trúc điều kiện:

```
IF1
INCLUDE     mylib.mac
ENDIF
```

Ví dụ 4.56:

Kết hợp sử dụng thư viện macro và thư viện các thủ tục, lập chương trình tên là `typew.asm` để xem một tập tin trên màn hình. Chương trình sử dụng định nghĩa đoạn đầy đủ.

```

PAGE          ,132
TITLE TYPEW
; chương trình sử dụng các macro: d_char, go_dos, head,
; mak_buf, open, read_bf,
; write_r của thư viện macro mylib.mac và các thủ tục của thư viện
; get_one.lib
INCLUDE      mylib.mac
blk_siz EQU   512*20      ; kích thước khối dữ liệu đọc vào
code SEGMENT
ASSUME      CS:code, DS:code
ORG 80h
buf_cnt DB   ?           ; số ký tự trong đuôi lệnh
buffer LABELBYTE ; vùng đệm tại 81h
ORG 100h      ; bắt đầu chương trình dạng .COM
start:
head 'TYPEW'      ; đưa ra tiêu đề chương trình
; lấy đường tên của tệp
CALL get_path
; gọi đến thủ tục của thư viện
; get_one.lib
open SI, error    ; thủ tục mở tệp tin của thư viện
; get_one.lib
; đọc tệp tin, BX chứa thẻ tệp tin
newbuff:
MOV CX, blk_siz   ; lấy số byte cần đọc
d_read,<OFFSET datbuff>
; đọc 1 tệp tin có thể là the_file?
JC error          ; cờ CF = 1 khi có lỗi đọc
; AX chứa số byte vừa đọc từ đĩa

```

```

        OR    AX, AX
        JZ    exit
; hiển thị vùng đệm tệp tin vừa đọc ra màn hình
        MOV  CX, CX      ; số ký tự sẽ đưa ra
        MOV  SI, OFFSET datbuff
        CALL do_char; do_char là thủ tục gắn của
                        ; chương trình
        JMP  newbuff

exit:
        go_dos           ; trở về DOS

error:
        write_r    'Error' ; hiển thị báo lỗi đọc tệp tin
        CMP  AX, 2
        JNZ  e2
        Write_r    "Không tìm thấy tệp tin"

e2:
        go_dos
; tham chiếu thư viện các thủ tục
        INCLUDE  get_one.lib
; thủ tục gắn do_char xử lý từng byte của tệp tin
; sẽ kết thúc khi hết tệp tin
; input cho do_char: SI trỏ đến vùng đệm datbuff
; CX = số ký tự cần xử lý
do_char PROC
newchar:
        MOV  DL, [SI]    ; lấy một ký tự
; đổi thành ASCII bằng cách sửa bit thứ 8
        AND  DL, 7Fh
        CMP  DL, EOF     ; đã kết thúc tệp chưa?

```

```

        JE    done           ; kết thúc tệp
        d_char              ; đưa ký tự này ra màn hình
        INC    SI           ; ký tự tiếp theo
        LOOP   newchar
        RET
done:
        POP    AX
        go_dos
do_char ENDP
datbuff LABELBYTE          ; vùng đệm cho dữ liệu đọc từ đĩa
code    ENDS
        END    start       ; kết thúc chương trình typew.asm

```

4.6.5. Các toán tử giả PTR và LABEL

a) Toán tử PTR

Chúng ta đã có dịp sử dụng PTR trong khai báo các loại dữ liệu. PTR được sử dụng chủ yếu để định lại kiểu đã khai báo của một biểu thức địa chỉ. PTR có cú pháp sử dụng như sau:

```
type    PTR    address_expression
```

Trong đó, type là kiểu của dữ liệu: BYTE, WORD, DWORD. Và biểu thức địa chỉ (address_expression) có kiểu đi theo là DB, DW, và DD. Ví dụ khai báo:

```
data1   DB    15h
data2   DB    25h
```

và muốn thực hiện chuyển giá trị data1 vào AL, giá trị data2 vào AH chỉ bằng một lệnh:

```
MOV     AX, data1
```

thì lệnh này không hợp lệ, bởi toán hạng đích (AX) là một từ, còn toán hạng nguồn data1 lại được khai báo là một byte. Để hợp lệ, ta phải dùng lệnh có PTR:

```
MOV    AX, WORD PTR data1
```

Như vậy AH nhận giá trị data2 = 25h và AL nhận giá trị data1 = 15h.

Trường hợp sau đây cũng không hợp lệ:

```
MOV    [BX], 15h
```

Vì không thể biết ngăn nhớ mà địa chỉ [BX] trỏ tới là byte hay là từ. Để hợp lệ, cần phải chỉ định ngăn nhớ là một byte bằng PTR, và câu lệnh sau đây là hợp lệ:

```
MOV    BYTE PTR [BX], 15h
```

b) Toán tử LABEL

Toán tử LABEL thường dùng để định dạng kiểu dữ liệu nhưng cho phép gán các dữ liệu có cùng địa chỉ nhằm khắc phục mâu thuẫn về kiểu trong các lệnh thao tác với dữ liệu. Trong ví dụ trên, nếu khai báo:

```
data1    DB    15h
```

```
data2    DB    25h
```

không thể dùng được cho lệnh:

```
MOV    AX, data1
```

để khắc phục trường hợp này ta sử dụng LABEL như sau:

```
dt12     LABELWORD
```

```
data1    DB    15h
```

```
data2    DB    25h
```

LABEL chỉ định dt12 là một biến từ, data1 và data2 là các biến byte, nhưng data1 và data2 được gán cho cùng một địa chỉ một từ. Như vậy dùng lệnh sau đây là hợp lệ:

```
MOV    AX, dt12      ; AH = data2, AL = data1
```

Nếu ta khai báo khoản dữ liệu như sau:

```
.DATA
```

```
dt1     DW           1525h
```

```
dt2     LABELBYTE
```

```
        DW           3545h
```

```
dt3     LABELWORD
```

```
dt4     DB           45h
```

```
dt5     DB           54h
```

thì trong đoạn mã những lệnh sau đây:

```
MOV    AX, dt2        ; không hợp lệ, vì dt2 khai báo là một byte
```

```
MOV    AL, dt2        ; hợp lệ, vì dt2 và AL là byte
```

```
MOV    DX, WORD PTR dt2
```

; hợp lệ, vì dt2 khai báo lại là từ, DX = 3545h

```
MOV    CX, dt3        ; hợp lệ, vì CX và dt3 đều là từ
```

```
MOV    AH, BYTE PTR dt2
```

; hợp lệ, vì AH và dt2 là byte

```
MOV    BX, WORD PTR dt3
```

; hợp lệ, vì dt3 là từ, BX = 5445h

```
MOV    DL, BYTE PTR dt3
```

; hợp lệ vì dt3 khai báo lại là byte, DL = 45h

LABEL cũng có thể dùng để khai báo dữ liệu kiểu FAR trong những trường hợp như cần tham chiếu đến dữ liệu trong một đoạn khác (FAR):

```
databuff LABELFAR
```

4.6.6. Tham chiếu đến dữ liệu trong một đoạn khác

Trong chương trình dạng .COM chúng ta có thể định nghĩa một đoạn dữ liệu hoặc một đoạn ngăn xếp riêng biệt để tham chiếu tới. Tuy nhiên, cũng có thể tham chiếu đến dữ liệu chứa trong một đoạn khác

nếu đưa ra địa chỉ đoạn bằng chỉ dẫn AT (xem mục 4.4.7 (e)) trong cú pháp khai báo đoạn đầy đủ:

```
seg_name    SEGMENT    AT address
```

```
...
```

```
seg_name    ENDS
```

Giả sử viết chương trình dạng .COM tham chiếu đến ROM BIOS để lấy thông tin về hệ thống máy tính, như: mã số máy, ngày sản xuất ra ROM BIOS. Để đơn giản, ta lấy hệ thống IBM PC/AT 16-bit. Chương trình có tên là ROMDATE.ASM.

Ví dụ 4.57:

```
; chương trình đọc mã số máy và ngày tháng của ROM BIOS
; cho các loại PC
; Cấu trúc đưa ra là:
; Ngày sản xuất      Mã số      Loại PC
; 04/24/81           FF         PC
; 10/19/81           FF         PC
; 10/27/82           FF         PC
; 11/08/82           FE         PC XT, xách tay
; 01/10/84           FC         PC AT, 20 MB HDD
; 06/10/85           FC         PC AT, 30 MB HDD
; 01/10/86           FB         PC XT, bàn phím 101 phím
;
;
; INCLUDE mylib.mac
; Vùng thủ tục ROM BIOS
romdate    SEGMENT    AT    0FFFFh
; địa chỉ đoạn FFFFh của ROM BIOS
; chứa mã số máy, ngày sản xuất
; ROM BIOS

ORG 5
```

```

date          LABELFAR
romdate       ENDS
; đoạn mã chính của chương trình ROMDATE.ASM
code          SEGMENT
              ASSUME    CS: code, DS: code, ES: code
              ORG      100h

start:

              MOV       AX, romdate
              ; nạp địa chỉ đoạn của ROM BIOS

              ASSUME    DS: romdate
              ; chỉ định lại DS trở tới đoạn ROM
              ; BIOS

              MOV       DS, AX
              MOV       SI, OFFSET date
              ; địa chỉ nguồn DS:SI trong đoạn
              ; ROM BIOS

              MOV       DI, OFFSET r_date
              ; địa chỉ đích ES:DI trong đoạn code
              ; của chương trình ROMDATE

              MOV       CX, OFFSET st_end - OFFSET r_date
              ; số byte sao chép dữ liệu ngày
              ; tháng của ROM BIOS đến đoạn
              ; lệnh mã

              REP       MOVSB
              ASSUME    DS: code
              ; chỉ định lại DS trở cùng CS: code

              MOV       AX, CS
              MOV       DS, AX
              ; trở về lại đoạn của chương trình
              ; ROMDATE.ASM sau khi đã lấy

```

```

; được dữ liệu từ đoạn của
; ROM BIOS
...
; có thể là các lệnh xử lý để in ra
; thông báo các thông tin về ROM
; BIOS và mã số máy
...
go_dos
; macro trở về DOS bằng INT 20h
; vùng dữ liệu ở đây
r_date DB 'xx/xx/87'
...
st_end LABEL BYTE
code ENDS
END start

```

Trong chương trình này khai báo đoạn của ROM BIOS chứa các thông tin cần lấy:

```

romdate SEGMENT AT 0FFFFh;
ORG 5
date LABEL FAR
romdate ENDS

```

Vì ngày sản xuất ROM BIOS ở tại độ dịch 5 so với địa chỉ cơ sở của đoạn ROM BIOS (0FFFFh) nên cần có lệnh:

```
ORG 5
```

vùng dữ liệu nguồn date được khai báo kiểu FAR nhờ LABEL vì nó lấy theo địa chỉ cơ sở của đoạn ROM BIOS (DS:SI), chương trình ROMDATE của ta phải tham chiếu tới vùng dữ liệu nguồn trong ROM BIOS về vùng dữ liệu r_date, địa chỉ trong ES:DI.

4.6.7. Phương pháp xếp chồng đoạn

Trong chế độ địa chỉ gián tiếp thanh ghi, nội dung của các thanh ghi BX, SI, DI là con trỏ chỉ địa chỉ EA (offset), hay độ dịch so với địa

chỉ cơ sở đoạn trong DS. Nhưng có thể định nghĩa địa chỉ của ngăn nhớ bằng giá trị offset trong các thanh ghi BX, SI, DI so với cơ sở của đoạn khác (không phải DS như bình thường). Cú pháp của lệnh này như sau:

segment_register: [point_register]

Ví dụ:

MOV AX, ES:SI

Toán hạng nguồn của lệnh này là ngăn nhớ có địa chỉ trong cặp thanh ghi ES:SI, trong đó ES chứa địa chỉ cơ sở đoạn dữ liệu, còn SI là giá trị offset (lẽ ra, bình thường phải là cặp DS:SI).

Phương pháp xếp chồng này có thể thực hiện được cho cả với các chế độ địa chỉ trực tiếp (giá trị độ dịch là một hằng số) và địa chỉ cơ sở.

4.6.8. Tạo một chương trình thường trú trong bộ nhớ

Thông thường một chương trình ứng dụng chỉ được nạp vào bộ nhớ RAM khi thực hiện và khi kết thúc, nó trả về hệ điều hành DOS hay Windows. Hệ điều hành sẽ giải phóng vùng nhớ dành cho trình ứng dụng đó để nạp một trình ứng dụng khác có nhu cầu thực hiện. Nhiều khi, đặc biệt là những chương trình nhỏ thường xuyên được truy cập để thực hiện, ví dụ, như các chương trình điều khiển vào/ra các ngoại vi (gọi là drives), hay của hệ điều hành phải được đặt thường trú trong bộ nhớ chính để tăng tốc độ xử lý.

Thường trú trong bộ nhớ tức là khi kết thúc thực hiện chương trình vẫn chiếm giữ vùng nhớ dành cho nó. Các chương trình thường trú gọi tắt là TSR (Terminate And Stay Resident). Để không chiếm nhiều không gian nhớ, các chương trình thường trú phải nhỏ, không lớn hơn 64 kbyte. Chúng có dạng .COM, và sử dụng lệnh ngắt INT 27h hay INT 21h với mã chức năng AH = 31h để kết thúc và thường trú. Khi đã thường trú trong bộ nhớ, chỉ cần gõ tên chương trình thường trú hoặc sử dụng một tổ hợp phím nào đó kích hoạt là chạy được nó.

Để thực hiện được lệnh INT 27h, đặt DX = độ lớn của phần thường trú.

Ví dụ 4.58:

Chương trình trong ví dụ 4.55 có thể được viết để đặt thường trú trong bộ nhớ.

```

        TITLE loi_chao
        .MODEL      SMALL
        INCLUDE     mylib.mac
        .CODE
        ORG 100h

start:

        write_msg   'Hello, world of the macro assembler'
        MOV  DX, OFFSET start - OFFSET pg_len
        INT  27h    ; kết thúc và ở lại thường trú
pg_len LABELBYTE
        END  start

```

Trong chương trình này, có hai dòng lệnh để đặt thường trú là:

```

MOV  DX, OFFSET start - OFFSET pg_len
INT  27h    ; kết thúc và ở lại thường trú

```

Dòng đầu đặt DX bằng độ lớn của phần chương trình được thường trú. Dòng thứ hai kết thúc chương trình và ở lại thường trú. Độ lớn phần thường trú được assembler tính toán thông qua các nhãn địa chỉ đầu (start) và địa chỉ cuối (pg_len).

Ví dụ 4.59:

Là một cách tạo nội dung DX để thường trú

```

        TITLE loi_chao
        INCLUDE     mylib.mac
code SEGMENT NEAR

```

```

{
    ASSUME    CS: code, DS: code
    ORG      100h
wr_kt  PROC   NEAR
    write_msg 'Hello, world of the macro assembler'
    LEA      DX, wr_kt      ; lấy giá trị độ dịch (địa chỉ
                           ; hiệu dụng)
    INT      27h            ; kết thúc và ở lại thường trú
wr_kt  ENDP
code   ENDS
      END      wr_kt

```

Để thực hiện được lệnh INT 21h với AH = 31h, và AL = mã lỗi khi trả về, và phải đặt DX = kích thước đoạn tính theo số đoạn, mỗi đoạn có kích thước bằng 16 byte:

Ví dụ 4.60:

```

; đoạn lệnh đặt thường trú bằng AH = 31h, INT 21h
MOV  AH, 31h
MOV  AL, err_code ; mã lỗi
MOV  DX, 100h     ; kích thước bằng 100h đoạn
                     ; (1000h byte)
INT  21h

```

a) Cấu trúc của chương trình thường trú

Cấu trúc của một chương trình thường trú (TSR) gồm có hai phần phân biệt thực hiện trong những thời gian khác nhau. Phần thứ nhất là phần cài đặt, và chỉ thực hiện một lần khi MS-DOS nạp chương trình. Phần cài đặt thực hiện bất kỳ những nhiệm vụ khởi tạo nào mà TSR yêu cầu và sau đó thoát về DOS nhờ chức năng kết thúc và ở lại thường trú.

Phần thứ hai của TSR gọi là phần thường trú. Nó gồm có mã và dữ liệu nằm lại bộ nhớ sau khi kết thúc chương trình. Phân loại TSR thành thụ động và tích cực.

b) TSR thụ động

Một cách đơn giản thực hiện một TSR là chuyển điều khiển cho TSR một cách rõ ràng từ một chương trình khác. Trong trường hợp này TSR là thụ động. Nếu chương trình gọi có thể xác định được địa chỉ vùng nhớ của TSR thì nó có thể cung cấp điều khiển thông qua một lệnh gọi hay lệnh nhảy xa. Thường thì một chương trình kích hoạt TSR thụ động thông qua ngắt mềm. Khi đó, trong phần cài đặt của TSR phải viết địa chỉ của phần thường trú ở vị trí phù hợp trong bảng véc-tơ ngắt. Các TSR thụ động thường thay thế các ngắt mềm. Ví dụ, một TSR thụ động có thể thay thế INT 10h phục vụ màn hình nhờ có các lời gọi ghi đọc màn hình với sự truy cập trực tiếp vào bộ đệm của màn hình. Các TSR thụ động cho phép truy cập giới hạn bởi chúng thực hiện được chỉ nhờ một chương trình khác. Chúng có ưu điểm khi thực hiện trong phạm vi của chương trình gọi và gây trở ngại quá trình khác.

c) TSR tích cực

Một phương pháp khác để thực hiện TSR là chuyển điều khiển cho TSR thông qua một sự kiện phần cứng, chẳng hạn như định nghĩa trước một chuỗi các phím ấn. Trong trường hợp này TSR gọi là tích cực bởi vì nó phải luôn luôn tìm kiếm tín hiệu khởi động. Các TSR tích cực có ưu điểm chính là ở khả năng truy cập của chúng. Chúng nhận điều khiển từ bất kỳ một ứng dụng nào đang hoạt động để thực hiện và trả về điều khiển theo yêu cầu. Nhưng TSR phải có thêm mã xác định thời điểm thích hợp để thực hiện. Mã thêm đó gồm một hoặc một số các chương trình con - gọi là "xử lý ngắt" (interrupt handler).

d) Các chương trình con xử lý ngắt trong các TSR tích cực

Khoản thường trú của TSR tích cực gồm có hai phần: phần thứ nhất chứa thân (body) của TSR - code và dữ liệu thực hiện những nhiệm vụ chính của chương trình. Phần còn lại chứa các chương trình con xử lý ngắt (interrupt handlers). Một chương trình con xử lý ngắt (đôi khi gọi là phục vụ ngắt) nhận điều khiển khi xuất hiện một ngắt đặc biệt. Nhưng thường thì chương trình con của TSR không phục vụ cho ngắt, mà thay vào đó nó chuyển điều khiển cho chương trình con xử lý ngắt cơ sở để phục vụ ngắt. Mỗi một chương trình con xử lý ngắt trong TSR thực hiện một hoặc một số chức năng sau đây: kiểm soát các sự kiện phần cứng có thể phát tín hiệu yêu cầu tới TSR, theo dõi trạng thái của hệ thống, và xác định yêu cầu nào phải được TSR phục vụ dựa vào trạng thái hiện thời của hệ thống.

Kiểm soát các sự kiện phần cứng yêu cầu TSR phục vụ:

TSR tích cực thường sử dụng một chuỗi các ấn phím hoặc sự kiện của bộ thời gian (timer) như là tín hiệu yêu cầu phục vụ. Như vậy phải có chương trình con xử lý các chuỗi ấn phím. Chương trình xử lý ấn phím nhận điều khiển mỗi khi có một chuỗi các ấn phím. Nó kiểm tra từng phím, tìm kiếm tín hiệu phù hợp hoặc "phím nóng". Khi phát hiện có tín hiệu của "phím nóng", nó chỉ đơn thuần thiết lập cờ yêu cầu và thoát (exit) chuyển đến xử lý chuỗi các ấn phím tiếp theo.

Ví dụ 4.61:

Một TSR tích cực kiểm soát các phím ấn thông qua chương trình con xử lý ngắt INT 9, ngắt của bàn phím:

keybrd	PROC	FAR	
	STI		; lập cờ cho phép ngắt
	PUSH	AX	; cất AX vào ngăn xếp
	IN	AL, 60h	; đọc mã quét của phím ấn
			; vào AL qua cổng đọc bàn
			; phím 60h


```

CALL      CheckHotKey ; gọi thủ tục kiểm tra
                        ; phím nóng
.IF      carry?      ; nếu phím nóng được ấn
...
keybrdENDP

```

Theo dõi trạng thái của hệ thống:

Một TSR mà sử dụng các thiết bị phần cứng như đĩa hoặc màn hình thì không thể ngắt khi thiết bị đang tích cực. TSR theo dõi thiết bị nhờ xử lý ngắt thiết bị đó. Mỗi chương trình con xử lý ngắt thiết lập cờ báo thiết bị đang được sử dụng và sau đó xóa cờ khi xử lý ngắt thực hiện xong. Chỉ những thiết bị được TSR sử dụng mới cần có theo dõi. Ví dụ, một TSR thực hiện vào/ra với thiết bị đĩa phải theo dõi sự sử dụng đĩa thông qua ngắt INT 13h. Chương trình con xử lý ngắt đĩa thiết lập cờ tích cực để báo đĩa đang bận nhằm ngăn chặn sự thực hiện của TSR trong khi đang có thao tác vào/ra đĩa. Nếu không, thao tác vào/ra đĩa của chính TSR sẽ làm thao tác vào/ra đĩa đang xảy ra và chuyển dịch đầu từ, điều này dẫn đến việc thao tác đĩa bị ngừng sẽ tiếp tục với đầu từ bị định vị không đúng khi TSR trả điều khiển về cho chương trình bị ngắt.

Xác định yêu cầu để TSR phục vụ:

Một khi chương trình con khi nhận được tín hiệu yêu cầu tới TSR, nó kiểm tra các cờ trạng thái đang tích cực, nếu có bất kỳ cờ nào được thiết lập thì chương trình con loại bỏ yêu cầu và thoát về. Nhưng nếu các cờ bị xóa, chương trình con gọi TSR thông qua lệnh gọi (xa hoặc gần).

Ví dụ 4.62:

Chương trình ALARM.ASM đưa ra loa tiếng beep vào thời gian báo thức định ra bằng cách đưa vào dòng lệnh:

ALARM time

trong đó time là biến thời gian đặt báo thức, ví dụ nếu đặt 7:45 AM thì viết:

ALARM 0745

biến time được chương trình tiếp nhận khi ta gõ dòng lệnh này. Ngắt thời gian INT 8h được xử lý ở đây. Chương trình kết thúc và ở lại thường trú bằng chức năng 31h của INT 21h. Một ngắt thời gian xuất hiện 18,2 lần trong 1 s hoặc 91 lần theo từng 5 s.

```
.MODEL      TINY          ; tạo ALARM.COM
.STACK                      ; mặc định ngăn xếp có 1 kbyte
.CODE
ORG 5Dh                  ; tại độ dịch 5Dh của PSP
                        ; (mào đầu của chương trình)
                        ; có biến thời gian
                        ; (xem mục 4.4.8)

countDown LABELWORD      ; định kiểu từ cho giá trị
                        ; khoảng cách 5 giây để quay
                        ; vòng lặp

.STARTUP                  ; bắt đầu chương trình thực
                        ; hiện ở đây (địa chỉ đầu)

JMP Install              ; nhảy tới phần cài đặt
; dữ liệu phải nằm trong đoạn mã để được Install tham chiếu
OldTimer      DWORD    ?    ; địa chỉ của chương trình
                        ; xử lý thời gian gốc

tick_91 BYTE    91          ; đếm 91 nhịp cho 5 giây
TimerActiveFlag BYTE 0      ; cờ tích cực cho chương trình
                        ; con xử lý thời gian

; NewTimer - chương trình con xử lý thời gian theo INT 8h
; Decrements CountDown từng 5 giây. Không có tác động gì
; trong khi Count Down chưa đạt được giá trị 0, mà ở đó phát
; loa kêu. Đây là đoạn xử lý ngắt thời gian mới thay cho thủ tục
; xử lý thời gian (đếm thời gian) gốc của ROM BIOS (bằng INT 8):
```

```

NewTimer    PROC    FAR
    .IF      CS: TimerActiveFlag !=0
                                ; nếu bộ thời gian bận
                                ; (biểu thức khác 0)
    JMP      CS: OldTimer      ; thì nhảy đến chương trình
                                ; thời gian gốc chỉ định CS là
                                ; địa chỉ đoạn cơ sở (CODE)
    .ENDIF
    INC      CS: TimerActiveFlag ; lập cờ active
    PUSHF                                ; cất các cờ vào ngăn xếp
    CALL     CS: OldTimer      ; gọi xa tới chương trình thời
                                ; gian gốc
    STI                                ; lập cờ cho phép ngắt
    PUSH DS
    PUSH CS
    POP  DS
    MOV      tick_91, 91        ; xóa bộ đếm thứ hai và
    DEC      CountDown          ; trừ khoảng cách 5 giây
    .IF      zero?              ; nếu đã lặp 91 lần
    CALL     Sound              ; gọi đến đưa ra loa
    INC      TimerActiveFlag    ; báo đã đưa ra loa, tăng cờ
    .ENDIF
    .ENDIF
    DEC      TimerActiveFlag    ;
    POP      DS                ;
    IRET                                ; trở về từ xử lý ngắt thời gian
NewTimer    ENDP
; Đưa ra loa tiếng kêu beep với khoảng cách
BEEP_TONE   EQU    440         ; tần số 440 Hz của tiếng beep

```

```

BEEP_DURATION    EQU    6        ; Số nhịp của beep, với 18
                                   ; nhịp bằng xấp xỉ 1 giây

Sound            PROC            USES AX BX CX DX ES
                                   ; toán tử giả USES dùng
                                   ; trong khai báo PROC chỉ
                                   ; định tên các thanh ghi được
                                   ; sử dụng trong thủ tục và
                                   ; phải được cất giữ trước tiên

    MOV    AL, 0B6h                ; khởi tạo channel 2 của chip
                                   ; timer

    OUT    43h, AL                ; cổng của channel 2 của chip
                                   ; timer

    MOV    DX, 12h                ; chia 1193180 Hz
    MOV    AX, 34DCh              ; (tần số nhịp đồng hồ) cho
    MOV    BX, BEEP_TONE          ; tần số yêu cầu
    DIV    BX                    ; kết quả là số đếm nhịp
                                   ; thời gian

    OUT    42h, AL                ; byte thấp của số đếm đưa ra
                                   ; chip thời gian

    MOV    AL, AH                ; byte cao của số đếm
    OUT    42h, AL                ; đưa ra chip thời gian

    IN     AL, 61h                ; đọc giá trị từ cổng 61h
    OR     AL, 3                  ; thiết lập 2 bit đầu tiên để
    OUT    61h, AL                ; bật loa

; Định khoảng thời gian cho beep loa
    MOV    AX, BEEP_DURATION
    SUB    CX, CX                ; CX: DX = số tích cho
                                   ; khoảng nghỉ

    MOV    ES, CX                ; trở ES đến vùng dữ liệu
    ADD    DX, ES:[46Ch]         ; cộng số tích vào CX:DX

```

```

        ADC    CX, ES:[46EH]      ; kết quả là số đếm trong
                                   ; CX:DX

        .REPEAT
        MOV    BX, ES:[46CH]      ; lặp lại
        MOV    AX, ES:[46EH]
        SUB    BX, DX              ; đạt được thời gian
        SBB    AX, CX
        .UNTIL    !carry?
        IN     AL, 61h             ; khi đã hết lặp
        XOR    AL, 3
        OUT    61h, AL            ; tắt loa
        RET

Sound      ENDP

; phần nạp
Install    PROC

DEFAULT_TIME    EQU    3600      ; đặt mặc định thời gian báo
                                   ; thức là 1 giờ

        MOV    AX, DEFAULT_TIME
        CWD                                ; DX:AX = thời gian mặc định
                                   ; tính theo giây

        .IF    BYTE    PTR Countdown != "
                                   ; nếu không phải biến trống
        XOR    Countdown[0], '00' ; đổi 4 byte của biến ASCII
        XOR    Countdown[2], '00' ; thành số nhị phân
        MOV    AL, 10              ; nhân chữ số (digit) đầu tiên
                                   ; của giờ với 10

        MUL    BYTE    PTR Countdown[0]
                                   ; và cộng thêm chữ số thứ 2
        ADD    BYTE    PTR Countdown[1] ;

```

```

MOV  BH, AL      ; BH = giờ để báo tắt
MOV  AL, 10      ; lặp lại thủ tục cho phút
MUL  BYTE PTR CountDown[2]
                    ; nhân chữ số đầu tiên của
                    ; phút với 10
ADD  BYTE PTR CountDown[3]
                    ; và cộng thêm chữ số thứ 2
                    ; của phút
MOV  BH, AL      ; BH = phút để báo tắt
MOV  AH, 2CH     ; yêu cầu chức năng 2Ch cho
                    ; INT 21h
INT  21h         ; nhận thời gian (CX = thời
                    ; gian giờ/phút)

MOV  DL, DH
SUB  DH, DH
PUSH DX          ; cất DX = số giây hiện hành
MOV  AL, 60      ; nhân số giờ hiện hành với 60
MUL  CH          ; để chuyển thành số phút
SUB  CH, CH
ADD  CX, AX      ; cộng số phút hiện hành với kết quả
                    ; CX = số phút từ nửa đêm

MOV  AL, 60      ; nhân giờ báo thức với 60
MUL  BH          ; để chuyển thành số phút
SUB  BH, BH
ADD  AX, BX      ; AX = số phút từ nửa đêm để đặt
                    ; báo thức

SUB  AX, CX      ; AX = thời gian tính theo phút để lặp
.IF  carry?      ; nếu thời gian là ngày hôm sau
ADD  AX, 24 * 60 ; cộng số phút của một ngày
.ENDIF

```

```

MOV  BX, 60
MUL  BX                ; DX:AX = số phút để lập
POP  BX                ; phục hồi số giây hiện hành
SUB  AX, BX            ; DX:AX = số giây để lập
                        ; trước báo thức
SBB  DX, 0             ; báo thức tích cực
.IF  carry?            ; nếu giá trị âm
MOV  AX, 5             ; đặt 5 giây
CWD
.ENDIF
.ENDIF
MOV  BX, 5             ; chia kết quả cho 5 giây
DIV  BX                ; AX = số khoảng cách 5 giây
MOV  CountDown, AX     ; để lập trước khi đưa ra loa
MOV  AX, 3508h         ; đặt chức năng 35h
                        ; (lấy véc-tơ ngắt)
INT  21h               ; nhận véc-tơ ngắt của timer
                        ; (INT 8)
MOV  WORD PTR OldTimer[0], BX
                        ; cất địa chỉ của thủ tục xử lý
                        ; ngắt thời gian gốc của
                        ; ROM BIOS
MOV  WORD PTR OldTimer[2], ES
MOV  AX, 2508h         ; chức năng 25h
                        ; (đổi véc-tơ ngắt)
MOV  DX, OFFSET NewTimer
                        ; DS:DX trở đến thủ tục xử lý
                        ; thời gian mới
INT  21h               ; đặt véc-tơ ngắt mới cho
                        ; NewTimer

```

```
MOV  DX, OFFSET Install ; DX = số byte của phần
                                ; thường trú
MOV  CL, 4                 ; chia cho 16 byte (dịch phải
                                ; 4 bit) để chuyển
SHR  DX, CL                ; thành số paragraphs
                                ; (1 paragraph = 16 byte)
INC  DX                   ; cộng thêm 1 paragraph
MOV  AX, 3100h            ; chức năng 31h, mã lỗi AL = 0
INT  21h                  ; kết thúc và thường trú
Install      ENDP
END
```

Trong chương trình này, chú ý rằng: hằng số BEEP_TONE để đặt âm cho báo thức. Các giá trị âm thường trong dải từ 100 đến 4000 Hz.

Thủ tục nạp (Install) đánh dấu điểm bắt đầu của phần nạp của chương trình ALARM.ASM. Sự thực hiện bắt đầu ở đây khi chương trình ALARM.COM được nạp vào bộ nhớ. Một TSR thường đặt phần mã nạp vào bộ nhớ của nó sau phần thường trú. Bởi vì phần nạp chỉ thực hiện một lần.

Để thực hiện thủ tục xử lý ngắt thời gian mới phải kết hợp sử dụng hai chức năng của DOS (INT 21h) đó là lấy véc-tơ ngắt (AH = 35h) và đổi véc-tơ ngắt (AH = 25h).

Ta có thể thay đổi hoạt động của một thủ tục của ROM BIOS (các thủ tục xử lý ngắt) hay của DOS bằng cách gán một giá trị mới cho véc-tơ ngắt tương ứng (đổi véc-tơ ngắt), tức là cho véc-tơ ngắt của thủ tục ROM BIOS hay DOS trở đến địa chỉ của thủ tục xử lý ngắt do chúng ta lập trình đặt ở một vùng nhớ khác. Cách làm an toàn là dùng hàm 25h của INT 21h. Để thực hiện đổi véc-tơ ngắt phải thực hiện thứ tự như sau:

1. Đặt DS:DX = giá trị mới cho véc-tơ ngắt - là địa chỉ của thủ tục do lập trình tạo mới

2. Đặt AL = số hiệu ngắt gốc của ROM BIOS, AH = 25h

3. Thực hiện INT 21h.

Nhưng để có thể trả lại xử lý ngắt bình thường cho ROM BIOS hay DOS trước khi thực hiện đổi véc-tơ ngắt phải thực hiện lấy véc-tơ ngắt cũ (AH = 35h) để cất giữ, nhằm sau phục hồi ngắt gốc của ROM BIOS. Để thực hiện lấy véc-tơ ngắt phải thực hiện thứ tự như sau:

1. Đặt AL = số hiệu ngắt gốc của ROM BIOS, và AH = 35h

2. Thực hiện INT 21h

Kết quả trả về của các lệnh này là cặp thanh ghi ES:BX = địa chỉ thủ tục xử lý ngắt gốc của ROM BIOS tương ứng với số hiệu ngắt trong AL.

Trong trường hợp ví dụ 4.60. Thủ tục đếm thời gian của ROM BIOS có số hiệu ngắt là 8 (INT 8), do đó ta có các lệnh lấy véc-tơ ngắt:

```
MOV    AX, 3508h    ; đặt chức năng 35h (lấy véc-tơ ngắt)
INT     21h          ; nhận véc-tơ ngắt của timer (INT 8)
MOV     WORD PTR OldTimer[0], BX
                                ; cất địa chỉ của chương trình xử lý ngắt
                                ; thời gian gốc
MOV     WORD PTR OldTimer[2], ES
```

Tiếp sau lấy véc-tơ ngắt là ta phải đổi véc-tơ ngắt để trở tới thủ tục xử lý thời gian mới:

```
MOV     AX, 2508h    ; chức năng 25h (đổi véc-tơ ngắt)
MOV     DX, OFFSET NewTimer
                                ; DS:DX trở đến chương trình xử lý thời
                                ; gian mới
INT     21h          ; đặt véc-tơ ngắt mới cho NewTimer
```

Như vậy sau khi đổi véc-tơ ngắt, với ngắt INT 8h, máy sẽ nhảy tới thủ tục xử lý thời gian mới NewTimer để thực hiện. Module xử lý ngắt thời gian của chương trình ALARM.ASM là thủ tục NewTimer,

nhưng phần nạp thường trú lại là thủ tục Install, vì vậy phải lấy kích thước của đoạn thường trú tính theo số paragraphs (mỗi paragraph bằng 16 byte), bằng các lệnh:

```
MOV    DX, OFFSET Install      ; DX = số byte của phần thường trú
MOV    CL, 4                   ; chia cho 16 byte ( tương đương dịch phải
                                ; 4 bit) để
SHR     DX, CL                 ; chuyển thành số paragraph
                                ; (1 paragraph = 16 byte)
INC     DX                     ; cộng thêm 1 paragraph
MOV     AX, 3100h              ; chức năng 31h, mã lỗi AL = 0
INT     21h                    ; kết thúc và thường trú
```

e) Tách TSR khỏi bộ nhớ

Nhiều khi cần phải tách TSR khỏi bộ nhớ (deinstall), trả lại bộ nhớ cho hệ thống mà TSR chiếm giữ. Chương trình tách (deinstall program) trước hết phải tìm TSR trong bộ nhớ thông qua địa chỉ yêu cầu. Khi tìm thấy TSR, chương trình tách so sánh các địa chỉ trong bảng véc-tơ ngắt với các địa chỉ của các chương trình con của TSR. Trong trường hợp không trùng địa chỉ thì chương trình tách ngăn cấm yêu cầu tách TSR. Nếu trùng địa chỉ thì quá trình tách TSR được tiếp tục thực hiện. Quá trình tách TSR khỏi bộ nhớ thường gồm có 3 bước:

1. Phục hồi cho bảng véc-tơ ngắt các véc-tơ ngắt của chương trình xử lý ngắt gốc
2. Đọc địa chỉ đoạn cất ở offset 2Ch của mào đầu PSP của chương trình TSR. Địa chỉ này trỏ đến khối thông tin (danh sách các biến) mà MS-DOS sao chép vào bộ nhớ khi nạp một chương trình. Đặt địa chỉ khối vào ES và vào AH mã chức năng DOS 49h (giải phóng khối nhớ) để trả khối nhớ (mà TSR chiếm đoạt) cho hệ điều hành.
3. Thực hiện chức năng DOS với AH = 49h, ES = địa chỉ của khối nhớ.

4.6.9. Viết các ứng dụng 32-bit

a) Định kiểu bộ nhớ và khai báo các đoạn

Để viết các chương trình ứng dụng 32-bit bằng hợp ngữ có thể sử dụng các toán tử giả cho các khai báo các đoạn đơn giản hóa. Chẳng hạn sử dụng toán tử giả .386 cho phép lập trình cho các hệ thống vi xử lý từ 80386 đến Pentiums. Tiếp sau phải dùng toán tử .MODEL với kiểu bộ nhớ FLAT. Chẳng hạn một chương trình 32-bit cho kiểu bộ nhớ FLAT và sử dụng các gọi chuẩn đến các thủ tục qui ước trong hệ điều hành Windows NT (stdcall) ta dùng các khai báo ban ở đầu chương trình như sau:

```
.386
.MODEL FLAT, stdcall
.STACK size_in_byte
.DATA
.CODE
... ; các chỉ dẫn có thể với các địa chỉ và các thanh ghi 32-bit
```

Ngoài các thanh đoạn 16-bit: CS, DS, ES, SS còn có hai thanh ghi đoạn dữ liệu bổ sung trong các vi xử lý 32-bit là GS và FS. Trong chế độ thực (8086), đánh địa chỉ của vi xử lý 32-bit được thực hiện như các vi xử lý 16-bit, tức là địa chỉ gồm 2 phần, phần cơ sở đoạn 16-bit trong các thanh ghi đoạn và phần offset 16-bit nằm trong các thanh ghi chung hay giá trị hằng số. Vấn đề lập trình trong chế độ thực không khác gì và hoàn toàn như cho vi xử lý 16-bit như 8086/8088. Trong độ bảo vệ, các thanh ghi đoạn không dùng trực tiếp như là một giá trị địa chỉ cơ sở đoạn mà là chỉ số trỏ tới bảng "chọn" (table of selectors), trong đó, mỗi một thanh ghi chọn mô tả một khối nhớ gồm cả các thuộc tính như kích thước, vị trí của khối, và các quyền truy cập (đọc, ghi, thực hiện). Địa chỉ EA của vùng nhớ được bằng cộng giá trị offset so với địa chỉ cơ sở của khối nhớ mô tả bởi lựa chọn. Chúng ta xét vấn

để đánh địa chỉ với cơ chế dùng các lựa chọn và mô tả ở chế độ bảo vệ của các vi xử lý Intel 32-bit ở chương 2, mục 2.4.

Trong khai báo đoạn đầy đủ (xem mục 4.4.7), kiểu use trong toán tử giả SEGMENT nhằm xác định kích thước của đoạn chỉ dùng cho 80386/80486. Trong đó, use có thể là:

USR16: chỉ định độ dịch là 16-bit (dùng trong chế độ thực 8086)

USE32: chỉ định độ dịch là 32-bit (dùng trong chế độ bảo vệ)

FLAT: đánh địa chỉ 32-bit, không phân đoạn, dung lượng nhớ tối đa 4 GB

Các chỉ định kích thước này có thể dùng trong khai báo các đoạn đơn giản hóa. Cụ thể:

Nếu dùng:

.MODEL

.386 ; hoặc .486

Thì USE16 là mặc định. Nếu dùng:

.386 ; hoặc .486

.MODEL

Thì USE32 là mặc định.

b) Cát giữ nội dung các thanh ghi vào ngăn xếp

Bắt đầu từ 80186 đến Pentiums có các lệnh PUSH và POP thực hiện cất giữ vào ngăn xếp và phục hồi từ ngăn xếp nội dung của tất cả các thanh ghi chung 16-bit của vi xử lý. Thứ tự cất giữ các thanh ghi (PUSH) như sau:

AX, CX, DX, BX, SP, BP, SI, và DI

riêng nội dung của SP được đẩy vào ngăn xếp là giá trị có trước khi thanh ghi đầu tiên được cất giữ (AX). Thứ tự phục hồi (POP) ngược lại:

DI, SI, BP, SP, BX, DX, CX và AX

Để cất giữ các thanh chung nhưng 32-bit thì sử dụng PUSHAD và POPAD

Thanh ghi cờ cũng thường được cất giữ vào ngăn xếp trong các thao tác gọi thủ tục. Có các lệnh cất giữ và phục hồi thanh ghi cờ 16-bit: PUSHF và POPF, và các lệnh cất giữ và phục hồi thanh ghi cờ 32-bit trong các vi xử lý Intel 32-bit: PUSHFD, POPFD. Nếu như không cần phải cất giữ toàn bộ nội dung thanh ghi cờ ta có thể sử dụng lệnh LAHF để nạp và cất giữ byte thấp của thanh ghi cờ vào thanh ghi AH. Lệnh SAHF phục hồi lại byte thấp của thanh ghi cờ từ AH.

4.6.10. Truy cập dữ liệu bằng con trỏ với toán tử giả TYPEDEF

Con trỏ truy cập dữ liệu là một biến chứa địa chỉ của một biến khác nào đó. Địa chỉ trong con trỏ trỏ đến một đối tượng khác. Sử dụng các con trỏ thuận tiện khi thực hiện chuyển một đối tượng dữ liệu lớn (như mảng dữ liệu) đến một thủ tục. Bởi vì khi đó thủ tục gọi chỉ cần đặt con trỏ ở trên ngăn xếp, mà thủ tục bị gọi sử dụng để xác định mảng dữ liệu.

Có sự phân biệt giữa một địa chỉ xa và một con trỏ xa. địa chỉ xa là địa chỉ của biến nằm ở một đoạn dữ liệu khác. Còn con trỏ xa là biến chứa địa chỉ đoạn và độ dịch của dữ liệu tham chiếu. Giống như bất kỳ một biến nào, con trỏ có thể nằm ở đoạn dữ liệu gần (mặc định) hoặc đoạn xa.

Khai báo con trỏ:

Toán tử giả TYPEDEF có thể xác định các kiểu của các biến trỏ. Cú pháp như sau:

```
typename      TYPEDEF    [[distance]] PTR qualifiedtype
```

Trong đó, typename là tên của kiểu của TYPEDEF (PBYTE, PWORD, NPBYTE, NPWORD, FPBYTE, FPWORD, PPBYTE,...). Distance có thể là NEAR hoặc FAR. Qualifiedtype có thể là bất kỳ kiểu dữ liệu nào của MASM (DATA, WORD, DWORD,...). hay của TYPEDEF (PBYTE,...)

Ví dụ 4.63:

```

PBYTE TYPEDEF PTR BYTE ; trỏ đến các byte
NPBYTE      TYPEDEF NEAR PTR BYTE
                        ; con trỏ gần trỏ đến các byte
FPBYTE      TYPEDEF FAR  PTR BYTE
                        ; con trỏ xa trỏ đến các byte
PWORD       TYPEDEF PTR  WORD
                        ; trỏ đến các từ
NPWORD      TYPEDEF NEAR PTR WORD
                        ; con trỏ gần trỏ đến các từ
FPWORD      TYPEDEF FAR  PTR WORD
                        ; con trỏ xa trỏ đến các từ
PPBYTE      TYPEDEF PTR  PBYTE
                        ; trỏ đến con trỏ trỏ các byte
                        ; (trong C, một mảng
                        ; các chuỗi)
PVOID TYPEDEF PTR      ; trỏ đến bất kiểu dữ liệu nào
PERSONAL STRUC          ; kiểu cấu trúc
    name BYTE 20 DUP(?)
    num  WORD   ?
PERSONAL ENDS
PPPERSON TYPEDEF PTR PERSON
                        ; trỏ đến kiểu cấu trúc

```

Khoảng cách (distance) của con trỏ có thể được đặt hoặc tự động xác định tự động trước nhờ kiểu bộ nhớ (bằng `.MODEL`) và kích thước đoạn (16 hoặc 32-bit). Nếu không dùng `.MODEL` để khai báo kiểu bộ nhớ thì các con trỏ gần là mặc định.

Trong chế độ 16-bit, con trỏ gần gồm 2 byte chứa độ dịch của đối tượng được trỏ. Con trỏ xa cần phải có độ dài 4 byte: cả độ dịch và cơ sở đoạn. Trong chế độ 32-bit, con trỏ gần là 4 byte và con trỏ xa là 6 byte.

Có thể khai báo biến trỏ bằng một kiểu trỏ tạo bởi TYPEDEF.

Ví dụ 4.64:

```

Array  WORD      25 DUP (0)   ; mảng từ
Msg     BYTE      "This is a sstring", 0
pMsg    PBYTE     Msg          ; trỏ đến chuỗi
pArray  PWORD     Array        ; trỏ đến mảng từ
npMsg   NPBYTE    Msg          ; trỏ gần đến chuỗi
npArray NPWORD    Array        ; trỏ gần đến mảng từ
fpArray FPWORD    Array        ; trỏ xa đến mảng từ
fpMsg   FPBYTE    Msg          ; trỏ xa đến chuỗi
S1      BYTE      "first", 0   ; chuỗi ký tự
S2      BYTE      "Second", 0
S3      BYTE      "third", 0
pS123   PBYTE     S1, S2, S3, 0 ; mảng các con trỏ trỏ đến
                                   ; các chuỗi
ppS123  PPBYTE    pS123       ; con trỏ trỏ đến các chuỗi
Andy    PERSON    <>          ; biến cấu trúc
pAndy   PPERSON   Andy        ; trỏ đến biến cấu trúc

```

Con trỏ tĩnh và động:

Những khai báo con trỏ trên đây đều là con trỏ tĩnh, tức là trỏ đến địa chỉ tĩnh của đối tượng dữ liệu. Nhưng thường con trỏ phải trỏ đến địa chỉ động, tức là địa chỉ phụ thuộc vào điều kiện của thời gian chạy. Ví dụ, MS-DOS phân phối bộ nhớ theo hàm 48h của INT 21h, và các địa chỉ trong các chỉ dẫn thực hiện chuỗi như SCAS và CMPS. Những trường hợp này cần khởi tạo con trỏ động.

Ví dụ 4.65:

```

fpBuf  FPBYTE    0
...
...
MOV  AH, 48h      ; chức năng phân phối bộ nhớ
MOV  BX, 10h      ; cần 16 paragraphs
INT  21h          ; Call DOS
JC   error        ; Return segment trong AX
MOV  WORD PTR fpBuf[2], AX
                        ; nạp cơ sở đoạn (offset đã đặt bằng 0)
...
error:  ...        ; xử lý lỗi

```

Sao chép các con trỏ:

Đôi khi một biến trỏ cần phải được khởi tạo nhờ sự sao chép từ con trỏ khác. Có hai cách sao chép một con trỏ xa:

Ví dụ 4.66:

```

fpBuf1      FPBYTE    ?
fpBuf2      FPBYTE    ?
...
; sao chép thông qua các thanh ghi
MOV  AX, WORD PTR fpBuf1[0]
MOV  WORD PTR fpBuf2[0], AX
MOV  AX, WORD PTR fpBuf1[2]
MOV  WORD PTR fpBuf2[2], AX
; sao chép thông qua ngăn xếp, không cần dùng thanh ghi
PUSH WORD PTR fpBuf1[0]
PUSH WORD PTR fpBuf1[2]

```



```
POP WORD PTR fpBuf2[2]
```

```
POP WORD PTR fpBuf2[0]
```

Các con trỏ là các biến (arguments):

Như là một biến dữ liệu, con trỏ có thể được chuyển cho thủ tục khác, ví dụ bằng ngăn xếp.

Ví dụ 4.67:

```
; đẩy vào ngăn xếp con trỏ xa
```

```
PUSH WORD PTR fpMsg[2]
```

```
; cất giữ cơ sở của ngăn xếp
```

```
PUSH WORD PTR fpMsg[0]
```

```
; cất giữ offset
```

```
; đẩy vào ngăn xếp địa chỉ xa như là con trỏ xa
```

```
MOV AX, SEG fVar
```

```
PUSH AX ; cất cơ sở đoạn
```

```
MOV AX, OFFSET fVar;
```

```
PUSH AX ; cất offset
```

Nạp con trỏ xa vào cặp thanh ghi:

Có thể dùng các chỉ dẫn LES và LDS để nạp con trỏ xa vào cặp thanh ghi, trong đó, từ thấp của biến trỏ chuyển vào ES hay DS, từ cao của biến trỏ nạp vào thanh ghi chỉ định trong chỉ dẫn LES hay LDS.

Ví dụ 4.68:

```
OutBuf BYTE 20 DUP (0)
```

```
fpOut FPBYTE OutBuf
```

```
...
```

```
LES DI, fpOut ; nạp con trỏ xa fpOut vào cặp thanh ghi  
; ES:DI
```

4.6.11. Nạp các địa chỉ vào các cặp thanh ghi

Thường xuyên trong chương trình hợp ngữ chúng phải thực hiện những thao tác nạp địa chỉ vào các thanh ghi. Đối với các địa chỉ gần (cùng trong một đoạn) thì chỉ cần nạp giá trị độ dịch 16-bit (offset) vào một thanh ghi chung nào đó đứng cùng cặp với thanh ghi đoạn nào để tạo địa chỉ thực 20-bit của vùng nhớ. Nhưng đối với những địa chỉ xa (đòi hỏi 2 từ), chúng phải nạp từ địa chỉ cơ sở đoạn vào thanh ghi đoạn và từ giá trị độ dịch vào một thanh ghi chung đứng cặp với thanh ghi đoạn. Có qui định thường dùng những cặp thanh ghi segment: offset để đánh địa chỉ bộ nhớ như sau:

DS:SI	địa chỉ nguồn cho các lệnh chuỗi chế độ 16-bit
DS:ESI	địa chỉ nguồn cho các lệnh chuỗi chế độ 32-bit
ES:DI	địa chỉ đích cho các lệnh chuỗi chế độ 16-bit
ES:EDI	địa chỉ đích cho các lệnh chuỗi chế độ 32-bit
DS:DX	địa chỉ vào của một số chức năng DOS (ví dụ đổi véc-tơ ngắt)
ES:BX	địa chỉ ra của một số chức năng DOS (ví dụ lấy véc-tơ ngắt)
CS:IP	địa chỉ của lệnh trong đoạn mã chế độ 16-bit
CS:EIP	địa chỉ của lệnh trong đoạn mã chế độ 32-bit
SS:SP hoặc SS:BP	địa chỉ ngăn nhớ trong ngăn xếp chế độ 16-bit
SS:ESP hoặc SS:EBP	địa chỉ ngăn nhớ trong ngăn xếp chế độ 32-bit

Các lệnh tham chiếu đến bộ nhớ đôi khi cần phải chỉ rõ cặp thanh ghi địa chỉ chẳng hạn:

```
MOV  AX, ES:[BX] ; AX ← [ES:BX]
ES: MOV  AX, [BX] ; AX ← [ES:BX]
MOV  AX, CS:[BP] ; AX ← [ES:BP]
```

MOVSF ; [ES:DI] ← [DS:SI]

ES: MOVSB ; [ES:DI] ← [DS:SI]

FS: MOVSB ; [ES:DI] ← [FS:SI]

Chúng ta đã xét vấn đề này trong các kiểu đánh địa của các vi xử lý Intel từ 16-bit.

Ví dụ 4.69:

Nạp địa chỉ vào cặp thanh ghi DS:BX

; nạp địa chỉ gần

.DATA

Msg BYTE "String"

...

MOV BX, OFFSET Msg ; nạp địa chỉ offset của Msg
; (DS đã được nạp)

...

; nạp địa chỉ xa

.FARDATA

Msg BYTE "String"

...

MOV AX, SEG Msg ; nạp địa chỉ đoạn dữ liệu
; chứa Msg

MOV ES, AX ; địa chỉ đoạn nạp vào ES

MOV BX, OFFSET Msg ; nạp offset của Msg, tạo địa
; chỉ ES:BX

Sao chép nội dung các cặp thanh ghi:

Có thể sao chép theo hai cách:

; sao chép chậm DS:SI vào ES:DI

PUSH DS ; lệnh 1 byte, 14 nhịp đồng hồ

POP ES ; lệnh 1 byte, 12 nhịp đồng hồ

MOV DI, SI ; lệnh 2 byte, 2 nhịp đồng hồ

; sao chép nhanh

MOV DI, DS ; 2 byte, 2 nhịp đồng hồ

MOV ES, DI ; 2 byte, 2 nhịp đồng hồ

MOV DI, SI ; 2 byte, 2 nhịp đồng hồ.

TÀI LIỆU THAM KHẢO

1. Bill Rieken, Lyle Weiman: *Advantures in Unix Network Applications Programming*, John Wwiley & Sons, Inc, 1992.
2. Charles M.Gilmore: *Micrprocessors Principles and Applications*, McGRAW-HILL International Editions, 1995.
3. Daniel Tabak: *Advanced Microprocessors*, McGRAW-HILL International Editions, 1995.
4. John P.Hayes: *Computer Architecture and Organization*, McGRAW-HILL International Editions, 1998.
5. Richard F. Ferraro: *Programmer's Guide to the EGA and VGA Cards*, Addison-Wesley Publishing Company, 1988.
6. R. P. Paul: *SPARC Architecture, Assembly Language Programming, and C*, Prentice-Hall, Englewood Cliffs, NJ, 1994.
7. Compaq Computer Corp.: *AlphaServer 1200, Compaq Tru64 UNIX*, 1999.
8. H. S. Stone: *High-Performance Computer Architecture*, Addition-Wesley, Reading, MA, 1993.
9. E. Strauss: *Inside the 80286*, Brady/Prentice-Hall, New York, 1986.
10. Roy W. Goody: *Intel Microprocessors*, McGRAW-HILL International Editions, 1993.
11. Hewlett Packard Corp.: *HP9000 Computer*, HP-UX Reference Release 10.0, 1995.
12. Microsoft Corp.: *MS-DOS and Windows file systems*, 2002.
13. Analog Devices Cor.: *DSP and ADSP-21XX*, 2002.
14. IBM Corp.: *RISC system/6000 Technology*, 1990.
15. Cisco System, Inc.: *First-Year Companion Guide*, 2001.
16. Clements: *68000 Family Assembly Language*, PWS Publ. Co., Bosston, MA, 1994.
17. www.Intel.com: *Intel Microprocessors technology*, 2004.

MỤC LỤC

Lời giới thiệu	5
Lời nói đầu.....	7
Chương 1: Giới thiệu chung	9
1.1. Trao đổi thông tin trong vi xử lý.....	9
1.1.1. Khái niệm vi xử lý	9
1.1.2. Trao đổi thông tin trong vi xử lý	9
1.2. Sự phát triển và ứng dụng của các bộ vi xử lý.....	12
1.3. Hệ thống vi xử lý.....	17
1.3.1. Khái niệm máy tính kiến trúc Von Neumann và kiến trúc Harvard	17
1.3.2. Hệ thống vi xử lý kiến trúc Von Neumann và kiến trúc Harvard	20
1.4. Những đặc điểm cấu trúc của bộ vi xử lý.....	23
1.4.1. Công suất của bộ vi xử lý	23
1.4.2. Những đặc tính nâng cao tốc độ của bộ vi xử lý	27
Bài tập ôn luyện.....	32
Chương 2: Cấu trúc và hoạt động của vi xử lý	35
2.1. Sơ đồ cấu trúc tổng quát của vi xử lý	35
2.1.1. Cấu trúc bên trong của đơn vị xử lý trung tâm.....	35
2.1.2. Sơ đồ cấu trúc tổng quát của bộ vi xử lý	37
2.2. Chức năng và hoạt động của các thành phần bên trong vi xử lý	39
2.2.1. ALU.....	39
2.2.2. Các thanh ghi của vi xử lý	40
2.2.3. Khối logic điều khiển	49
2.2.4. Bus dữ liệu trong	51

2.3. Cấu trúc bộ nhớ.....	56
2.3.1. Phân loại các bộ nhớ.....	56
2.3.2. Bộ nhớ truy cập ngẫu nhiên (RAM).....	57
2.3.3. Các bộ nhớ chỉ đọc (ROM, EDROM, EAROM, EEPROM, FLASH Memory)	80
2.3.4. Các vi mạch ứng dụng đặc biệt	93
2.3.5. Thiết bị nhớ ngoài	97
2.4. Vi xử lý Intel	113
2.4.1. Đặc điểm chính của các loại vi xử lý Intel	113
2.4.2. Mô hình lập trình chung của họ Intel	121
2.4.3. Vi xử lý 8085.....	126
2.4.4. Vi xử lý 8086/8088.....	136
2.4.5. Đồng xử lý toán học 8087	147
2.4.6. Vi xử lý 80286.....	154
2.4.7. Vi xử lý 80386.....	183
2.4.8. Vi xử lý 80486.....	219
2.4.9. Vi xử lý Pentium	239
2.4.10. Pentium III.....	245
2.4.11. Pentium IV	249
2.4.12. Pentium M và công nghệ Centrino Mobile	255
2.5. Vi xử lý Z80	265
2.5.1. Đóng vỏ và sơ đồ khối của Z80.....	265
2.5.2. Mô hình lập trình và các thanh ghi (register)	269
2.5.3. Tập lệnh và kiểu đánh địa chỉ của Z80.....	274
2.5.4. Định thời của Z80.....	274
2.5.5. Các vi mạch lập trình hỗ trợ cho Z80	275
2.6. Họ vi xử lý Motorola M68000	280
2.6.1. Đặc điểm của họ vi xử lý M68000	280
2.6.2. Mô hình lập trình của người sử dụng	282
2.6.3. Mô hình lập trình của người giám sát	287

2.6.4. Dạng dữ liệu	289
2.6.5. Các chế độ đánh địa chỉ	290
2.6.6. Tập lệnh của họ M68000.....	291
2.6.7. Quản lý bộ nhớ.....	291
2.6.8. Bộ nhớ Cache.....	297
2.6.9. Xử lý ngoại lệ	299
2.6.10. Kiến trúc của M68040.....	301
2.6.11. Kiến trúc của M68060.....	304
<i>Bài tập ôn luyện</i>	313
Chương 3: Tập lệnh của vi xử lý	325
3.1. <i>Khái niệm về lệnh, dạng lệnh và cách mã hóa lệnh của vi xử lý</i>	325
3.1.1. Lệnh và thực hiện lệnh	325
3.1.2. Các dạng lệnh, mã lệnh	328
3.2. <i>Các phương pháp đánh địa chỉ của vi xử lý</i>	337
3.2.1. Đánh địa chỉ trực tiếp thanh ghi (implied, register direct).....	337
3.2.2. Đánh địa chỉ ngay lập tức (immediate)	339
3.2.3. Đánh địa chỉ trực tiếp hay địa chỉ tuyệt đối (direct) ..	339
3.2.4. Đánh địa chỉ gián tiếp thanh ghi (register indirect) .	344
3.2.5. Đánh địa chỉ gián tiếp thanh ghi chỉ số.....	350
3.2.6. Đánh địa chỉ gián tiếp bộ nhớ (memory indirect)	356
3.2.7. Đánh địa chỉ tương đối với thanh đếm chương trình (PC relative).....	357
3.2.8. Đánh địa chỉ gián tiếp thanh đếm chương trình với độ dịch.....	358
3.2.9. Đánh địa chỉ gián tiếp thanh đếm chương trình với chỉ số	358
3.2.10. Đánh địa chỉ gián tiếp thanh đếm bộ nhớ thanh đếm chương trình.....	359

3.3. Phân loại tập lệnh của vi xử lý	360
3.3.1. Tập lệnh phức tạp và tập lệnh giảm thiểu (CISC và RISC).....	360
3.3.2. Phân loại tập lệnh của bộ vi xử lý	361
3.4. Mô tả tập lệnh của vi xử lý	362
3.4.1. Nhóm lệnh chuyển dữ liệu	362
3.4.2. Nhóm lệnh số học và logic	365
3.4.3. Nhóm lệnh dịch và quay vòng.....	367
3.4.4. Nhóm lệnh chuyển điều khiển.....	369
3.4.5. Nhóm lệnh xử lý bit.....	376
3.4.6. Nhóm lệnh điều khiển hệ thống*	378
3.4.7. Nhóm lệnh xử lý dấu phẩy động	379
3.4.8. Nhóm lệnh cho các chức năng đặc biệt.....	381
3.5. Các dạng dữ liệu.....	384
3.5.1. Không dấu và có dấu	385
3.5.2. Các ký tự mã ASCII.....	386
3.5.4. Dữ liệu ở mã BCD được đóng gói hay không đóng gói.....	386
Bài tập ôn luyện.....	388
Chương 4: Lập trình bằng hợp ngữ.....	391
4.1. Khái niệm về chương trình hợp ngữ	391
4.1.1. Giới thiệu hợp ngữ (Assembly)	391
4.1.2. Hợp ngữ cấp thấp LLA (Low Level Assembly) và cấp cao HLA (High Level Assembly)	392
4.2. Cách tạo và chạy chương trình hợp ngữ.....	393
4.2.1. Bước 1- Soạn thảo chương trình nguồn	394
4.2.2. Bước 2- Hợp dịch chương trình nguồn thành tệp đối tượng.....	394
4.2.3. Bước 3- Liên kết một hay nhiều tệp đối tượng tạo ra một tệp chương trình chạy được	396
4.2.4. Bước 4- Chạy chương trình.....	397

4.3. Các phần cơ bản của hợp ngữ	397
4.3.1. Trường tên	397
4.3.2. Trường toán tử	398
4.3.3. Trường toán hạng	399
4.3.4. Trường lời bình	400
4.3.5. Dữ liệu dùng trong chương trình hợp ngữ	400
4.3.6. Các biến nguyên	402
4.3.7. Các số thực (real)	410
4.3.8. Các số BCD	411
4.3.9. Các cấu trúc và các liên hợp	412
4.3.10. Các bản ghi	420
4.3.11. Các ký hiệu tượng trưng của Assembler	425
4.4. Khung của một chương trình hợp ngữ	425
4.4.1. Các đoạn logic dùng trong hợp ngữ	426
4.4.2. Khung của một module chính của chương trình sử dụng các đoạn đơn giản hóa	426
4.4.3. Xác định các thuộc tính của đoạn nhờ toán tử .MODEL	427
4.4.4. Đoạn ngăn xếp (stack segment)	429
4.4.5. Đoạn dữ liệu (data segment)	430
4.4.6. Đoạn mã	431
4.4.7. Định nghĩa đầy đủ các đoạn	436
4.4.8. Các cấu trúc chương trình nguồn khi dịch ra .exe và .com	447
4.5. Cơ chế làm việc và cấu trúc chương trình con	451
4.5.1. Khai báo thủ tục	451
4.5.2. Gọi các thủ tục xa và gần	453
4.5.3. Ngắt là gọi đến thủ tục xa	454
4.5.4. Truyền dữ liệu giữa các thủ tục	455
4.5.5. Thư viện các thủ tục	467

<i>4.6. Macro và các vấn đề liên quan</i>	<i>470</i>
4.6.1. Các macro văn bản	470
4.6.2. Các thủ tục macro.....	472
4.6.3. Các hàm macro.....	487
4.6.4. Thư viện macro.....	490
4.6.5. Các toán tử giả PTR và LABEL	499
4.6.6. Tham chiếu đến dữ liệu trong một đoạn khác	501
4.6.7. Phương pháp xếp chồng đoạn.....	504
4.6.8. Tạo một chương trình thường trú trong bộ nhớ	505
4.6.9. Viết các ứng dụng 32-bit.....	520
4.6.10. Truy cập dữ liệu bằng con trỏ với toán tử giả TYPEDEF.....	522
4.6.11. Nạp các địa chỉ vào các cặp thanh ghi.....	527
<i>Tài liệu tham khảo.....</i>	<i>531</i>

GIÁO TRÌNH
Kỹ thuật vi xử lý
(TẬP 1)

Chịu trách nhiệm xuất bản
LƯU ĐỨC VĂN

Biên tập: TRẦN VŨ THƯỜNG
BÙI NGỌC KHOA
Chế bản: NGUYỄN VĂN LUÂN
Sửa bản in: BÙI NGỌC KHOA
Trình bày bìa: TRỊNH THẾ VINH

(Giáo trình này được ban hành kèm theo Quyết định số 192/QĐ-QLNCKH ngày 15/4/2005 của Giám đốc Học viện Công nghệ Bưu chính Viễn thông)

NHÀ XUẤT BẢN BƯU ĐIỆN

Trụ sở: 18 - Nguyễn Du, Hà Nội

Điện thoại: 04-9432438, 9431284 Fax: 04-9431285

E-mail: bientap@hn.vnn.vn

Website: www.nxbbuudien.com.vn

Chi nhánh TP. HCM: 27 - Nguyễn Bình Khiêm, Quận 1, TP. Hồ Chí Minh

Điện thoại: 08-9100925 Fax: 08-9100924

E-mail: chinhanh-nxbbd@hcm.vnn.vn

Chi nhánh TP. Đà Nẵng: 42 - Trần Quốc Toản, Quận Hải Châu, TP. Đà Nẵng

Điện thoại: 0511-897467 Fax: 0511-897467

E-mail: pnbich@mpt.gov.vn

In 300 bản, khổ 16 x 24 cm, tại Công ty Cổ phần In Bưu điện
Số đăng ký kế hoạch xuất bản 62-2007/CXB/48₂ - 03/BuĐ
Số quyết định xuất bản: 115/QĐ-NXB BĐ ngày 14/6/2007
In xong và nộp lưu chiểu tháng 6 năm 2007.

NHÀ XUẤT BẢN BƯU ĐIỆN

Trụ sở chính: 18 Nguyễn Du - Hà Nội

ĐT Biên tập: 04.9431284

- ĐT Phát hành: 04.9432438

Fax: 04.9431285

E-mail: bientap@hn.vnn.vn

Website: www.nxbbuudien.com.vn

Chi nhánh TP.HCM: 27 Nguyễn Bỉnh Khiêm - Quận I - TP. Hồ Chí Minh

Điện thoại: 08.9100925

Fax: 08.9100924

E-mail: chinhanh-nxbbd@hcm.vnn.vn

Chi nhánh TP. Đà Nẵng: 42 - Trần Quốc Toản, TP. Đà Nẵng

Điện thoại: 0511.897467

Fax: 0511.897467

E-mail: pnbich@mpt.gov.vn

MỜI CÁC BẠN TÌM ĐỌC

1. TỪ ĐIỂN THUẬT NGỮ VIỄN THÔNG ANH - VIỆT
2. CÁC DỊCH VỤ VIỄN THÔNG CỦA TỔNG CÔNG TY BƯU CHÍNH VIỄN THÔNG VIỆT NAM
3. TRUYỀN HÌNH SỐ MẶT ĐẤT
4. CÔNG NGHỆ VoIP VÀ CÁC GIẢI PHÁP NÂNG CAO CHẤT LƯỢNG DỊCH VỤ
5. TIẾNG ANH CHUYÊN NGÀNH ĐIỆN TỬ - VIỄN THÔNG
6. CÁC HỆ THỐNG NGUỒN ĐIỆN, CHỐNG SÉT VÀ TIẾP ĐẤT TRẠM VIỄN THÔNG: NGUYÊN LÝ HOẠT ĐỘNG, QUẢN LÝ, VẬN HÀNH VÀ BẢO DƯỠNG
7. ẲNG TEN THÔNG MINH - GIẢI PHÁP CHO THÔNG TIN DI ĐỘNG TRONG TƯƠNG LAI
8. KỸ THUẬT ĐO KIỂM MẠNG VIỄN THÔNG
9. MẠNG NGOẠI VI - THIẾT KẾ, THI CÔNG VÀ QUẢN LÝ

SÁCH CỦA NHÀ XUẤT BẢN BƯU ĐIỆN CÓ BÁN TẠI:

1. Trung tâm Dịch vụ văn hóa Tiên phong VDC
292 Tây Sơn, Hà Nội
2. Nhà sách Bách Khoa
Số 1, Đường Giải Phóng, Hà Nội
86 - 107 Tô Hiến Thành, Quận 10, TP.HCM
3. Nhà sách Thăng Long
2 Bis Nguyễn Thị Minh Khai, Quận 1, TP.HCM

Giá: 68.000 đ