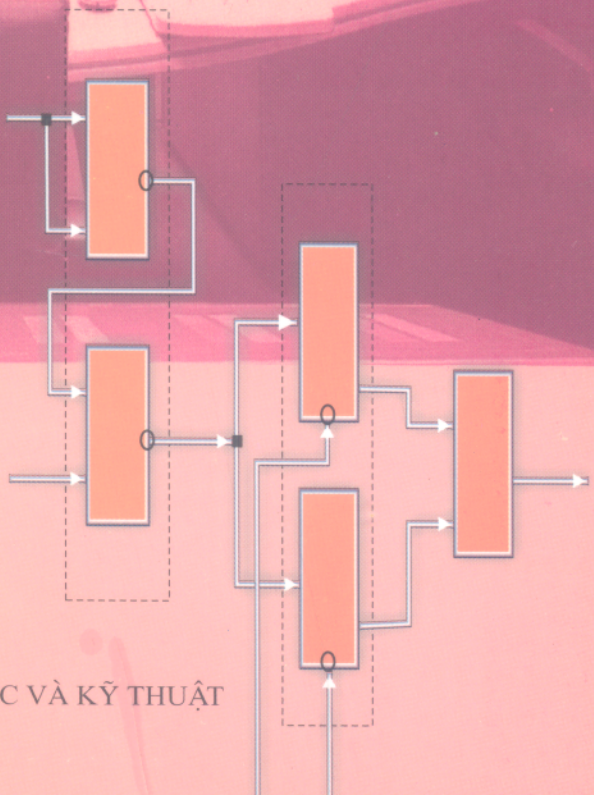
ĐỖ XUÂN TIẾN

Kỹ thuật lập trình điều khiển hệ thống



NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

Pgs, Ts. ĐỖ XUÂN TIẾN

KỸ THUẬT LẬP TRÌNH
ĐIỀU KHIỂN HỆ THỐNG

Hiệu đính: Pgs, Ts. ĐỖ ĐỨC GIÁO

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
HÀ NỘI - 2003

Chịu trách nhiệm xuất bản : **Pgs, Ts. TÔ ĐĂNG HẢI**
Biên tập : ĐỖ THỊ CẢNH
Sửa bản in : LÊ MINH
Trình bày bìa : HƯƠNG LAN

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
70 TRẦN HƯNG ĐẠO - HÀ NỘI

In 1.000 cuốn, khổ 16 x 24 cm. Tại Xí nghiệp in 19 - 8 số 3 đường Nguyễn Phong Sắc - Nghĩa Tân - Cầu Giấy - Hà Nội.
Giấy phép xuất bản số: 1270 - 23, Cục xuất bản cấp ngày 30/10/ 2002.
In xong và nộp lưu chiểu tháng 1 năm 2003.

MỤC LỤC

Trang

Chương 3

GIAO TIẾP GIỮA HỆ VI XỬ LÝ VÀ MÁY TÍNH PC

Chương 4

LẬP TRÌNH HỆ THỐNG TRÊN WINDOWS

Chương 5

LẬP TRÌNH ĐIỀU KHIỂN HỆ ĐA PHƯƠNG TIỆN

Chương 6

LẬP TRÌNH ĐIỀU KHIỂN HỆ TRUYỀN TIN MODEM

Chương 7

LẬP TRÌNH NETBIOS

PHỤ LỤC

MỞ ĐẦU

*Năm 1999 cuốn sách "**Kỹ thuật lập trình điều khiển hệ thống**" được xuất bản đã đáp ứng được phần nào các yêu cầu về khả năng thiết lập cơ chế đồng bộ trong hoạt động của các đối tượng có môi trường làm việc rất khác nhau (điện, điện tử, cơ , cơ - điện, quang điện tử ...) điều mà trước đây không thể thực hiện được bằng kỹ thuật cổ điển. Song nhiều lĩnh vực chủ chốt như kỹ thuật điện tử , tin học, viễn thông, công nghệ thông tin, kỹ thuật điều khiển tự động có những bước phát triển vượt bậc nên một số nội dung của cuốn tài liệu cũng cần được cập nhật kiến thức mới hoặc làm sâu sắc hơn nội dung cũ để có thể phục vụ hiệu quả hơn các ứng dụng trong điều khiển hệ thống. Trong lần tái bản này, tuy cấu trúc chương được giữ nguyên nhưng nội dung từng chương có những thay đổi cần thiết, đặc biệt từ chương 1 đến chương 6, cụ thể:*

Trong chương 1 và chương 2 đã phân tích sâu sắc hơn về tài nguyên hệ thống của hệ điều hành đơn nhiệm mà các chương trình hệ thống phải biết khai thác như bảng vector ngắt, tổ chức và tài nguyên giấu trong vùng ROMBIOS, phương thức truy nhập và quản lí các đối tượng rất phức tạp như RAM và các file chạy, kĩ thuật chặn, bắt và đánh tráo một ngắt của hệ thống, được trình bày dưới dạng một chương trình chặn ngắt 13H với đầy đủ các bước và phương pháp tạo giả lệnh INT để có thể gọi lại được ngắt 13H. Phần giao diện với ngoại vi được bổ sung thêm cấu trúc SLOT PCI là dạng SLOT chủ yếu trong các máy PC thế hệ mới.

Chương 3 là một giao diện điển hình của PC với ngoại vi là hệ vi xử lí chuyên dụng. Giao thức điều khiển được xây dựng chặt chẽ cả ở phần cứng (tạo card giao tiếp) cả ở phần mềm (chương trình điều khiển). Phần mềm được xây dựng trên ngôn ngữ ASSEMBLY nên đã khai thác được triệt để khả năng của phần cứng. Điều đặc biệt cần lưu ý ở đây là khả

năng mô phỏng chức năng của ROM của PC nên quá trình điều khiển ngoại vi này diễn ra rất triệt để.

Chương 4 và 5 trình bày một cách hệ thống tổ chức và phương thức sử dụng tài nguyên của hệ điều hành đa nhiệm WINDOWS . Công cụ lập trình hệ thống hướng đối tượng Delphi for win được sử dụng để thiết lập quan hệ từ nông đến sâu với hệ điều hành: từ bổ sung thuộc tính vào tài nguyên có sẵn tới việc tạo ra tài nguyên mới (component mới) và sâu hơn cả là điều khiển cơ chế ngắt trên Windows. Phần hệ thống được trình bày là hệ đa phương tiện hẹp (gồm CDPlayer và cấu trúc các file multimedia thông dụng) và phương pháp tổ chức hệ đa phương tiện mở cho phép ghép nối với các thiết bị chuẩn và phi chuẩn khác.

Chương 6 trình bày một hệ thống truyền thông điển hình với cơ chế chặn ngắt 14H của hệ thống. Để làm được điều đó, lần tái bản này có phân tích sâu cơ chế hàng đợi và các hàm hệ thống cho phép tạo DRIVER (dưới dạng một component mới) làm việc với mọi MODEM thông qua cổng COM của máy tính.

Tác giả xin chân thành cảm ơn Pgs. Ts. Đỗ Đức Giáo (Trường Đại học Khoa học Tự nhiên thuộc trường Đại học Quốc gia Hà Nội) đã bỏ nhiều công sức để hiệu đính cuốn tài liệu này, cảm ơn Nhà xuất bản Khoa học Kĩ thuật đã tạo mọi điều kiện thuận lợi để lần tái bản này cuốn sách được đến tay bạn đọc. *Tuy có nhiều cố gắng trong lần biên soạn này, song cuốn sách chắc chắn không tránh khỏi những thiếu sót. Chúng tôi chân thành mong nhận được ý kiến đóng góp của độc giả. Thư góp ý xin gửi về Nhà xuất bản Khoa học Kỹ thuật 70 Trần Hưng Đạo - Hà Nội.*

Tác giả

CHƯƠNG 1
TỔ CHỨC CHUNG CỦA MÁY TÍNH PC

1.1. CẤU TRÚC CỦA MÁY TÍNH PC

Trong kỹ thuật lập trình hệ thống thì máy tính PC hoặc mạng máy tính PC đóng vai trò như là phương tiện quan trọng trong tổ chức hệ thống cũng như trong xây dựng phần mềm điều khiển hệ thống đó. Nhờ đó mà các hệ thống này có tính mềm dẻo và thông minh trong các thao tác xử lý, gia công và tạo tín hiệu điều khiển cả theo chức năng và cả theo tham số thời gian thực. Máy tính PC mà bản chất là một hệ vi xử lý đa năng nên ngoài các chức năng được ấn định do cơ sở sản xuất ra còn có thể bổ xung vào cấu hình của nó rất nhiều các thành phần bổ trợ, cho phép biến chúng thành các hệ thống chuyên năng phục vụ cho bài toán thiết kế đặt ra. Để làm chủ được khả năng ứng dụng đó cần, trước hết phải làm chủ được kiến trúc của máy tính PC.

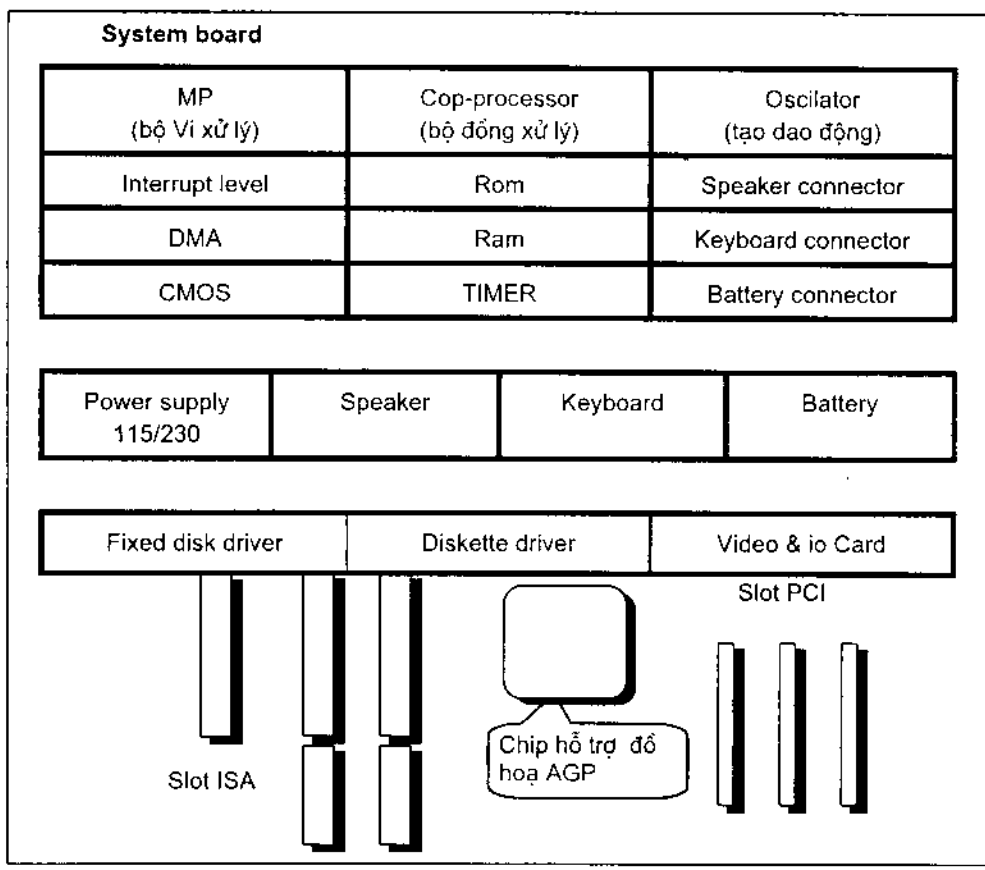
Cấu trúc bên trong của máy tính PC có dạng chung như hình 1.1. Các thành phần chủ yếu được liệt kê trong sơ đồ này là cấu hình cơ bản của máy tính PC. Các thế hệ máy tính khác nhau có kết cấu khác nhau, song chức năng của các thành phần hệ thống ít thay đổi.

1.1.1. Bộ xử lý trung tâm của máy tính PC

Phần tử cơ bản nhất để tạo thành bộ não của PC là bộ vi xử lý-MICROPROCESSOR (MP). Bộ vi xử lý MP có thể được hiểu như là hệ automat hữu hạn làm việc với một tập xác định các lệnh bằng ngôn ngữ máy. Mỗi lệnh ngôn ngữ có chức năng đơn giản, thực hiện một thao tác cụ thể nên nó có khả năng điều khiển hệ thống theo tham số thời gian

thực. Các ngôn ngữ bậc cao như BASIC, PACAL hay C có ưu điểm khi xây dựng các phần mềm chức năng lớn và tạo giao diện với người sử dụng, tuy nhiên những lệnh của ngôn ngữ bậc cao trước khi thực hiện được trên bộ vi xử lý phải được dịch thành một số tập hợp các lệnh ngôn ngữ máy. Thí dụ, phải cần hàng trăm lệnh ngôn ngữ máy mới tạo ra một lệnh cơ sở của PASCAL như Rectange. Các lệnh ngôn ngữ máy thay đổi tùy theo loại vi xử lý MP dùng trong máy tính.

Đối với máy PC, bộ xử lý trung tâm là các bộ vi xử lý 80286 (đối với máy AT), hoặc 80386, 80486, 80586... là những bộ vi xử lý của hãng INTEL- một trong các hãng sản xuất các chip vi xử lý lớn nhất thế giới.. Sự phát triển quan trọng nhất của các bộ vi xử lý bắt đầu từ MP 80286 trở đi là ở chỗ nó cung cấp khả năng làm việc đa nhiệm trên cơ sở của cơ chế quản lý bộ nhớ ảo.



Hình 1.1. **Sơ đồ khối của máy tính PC.**

Chế độ đa nhiệm là khả năng thực hiện đồng thời nhiều chương trình, nhiều thao tác. Máy tính chạy trên chế độ đa nhiệm bằng cách chuyển đổi phục vụ rất nhanh và liên tục giữa các chương trình tạo cảm giác là các nhiệm vụ này chạy song song. Khả năng này thực hiện được nhờ cấu trúc trong của bộ xử lý trung tâm khi nó được tổ chức và cài đặt cơ cấu quản lý đa nhiệm đó là tập hợp các thanh ghi ẩn, tổ chức bảng các bộ mô tả mảng nhớ, các tham số đặc tả trong các bộ mô tả, đặc quyền truy nhập, cơ chế kiểm soát và bảo vệ.

Cơ chế quản lý địa chỉ ảo là phương pháp cho phép chương trình sử dụng bộ nhớ lớn hơn nhiều so với bộ nhớ thực (RAM) của máy tính. Phần chương trình hoặc dữ liệu vượt quá dung lượng bộ nhớ RAM của máy tính được lưu ở bộ nhớ ngoài (đĩa mềm hoặc đĩa cứng) và được nạp vào RAM của máy tính khi cần thiết.

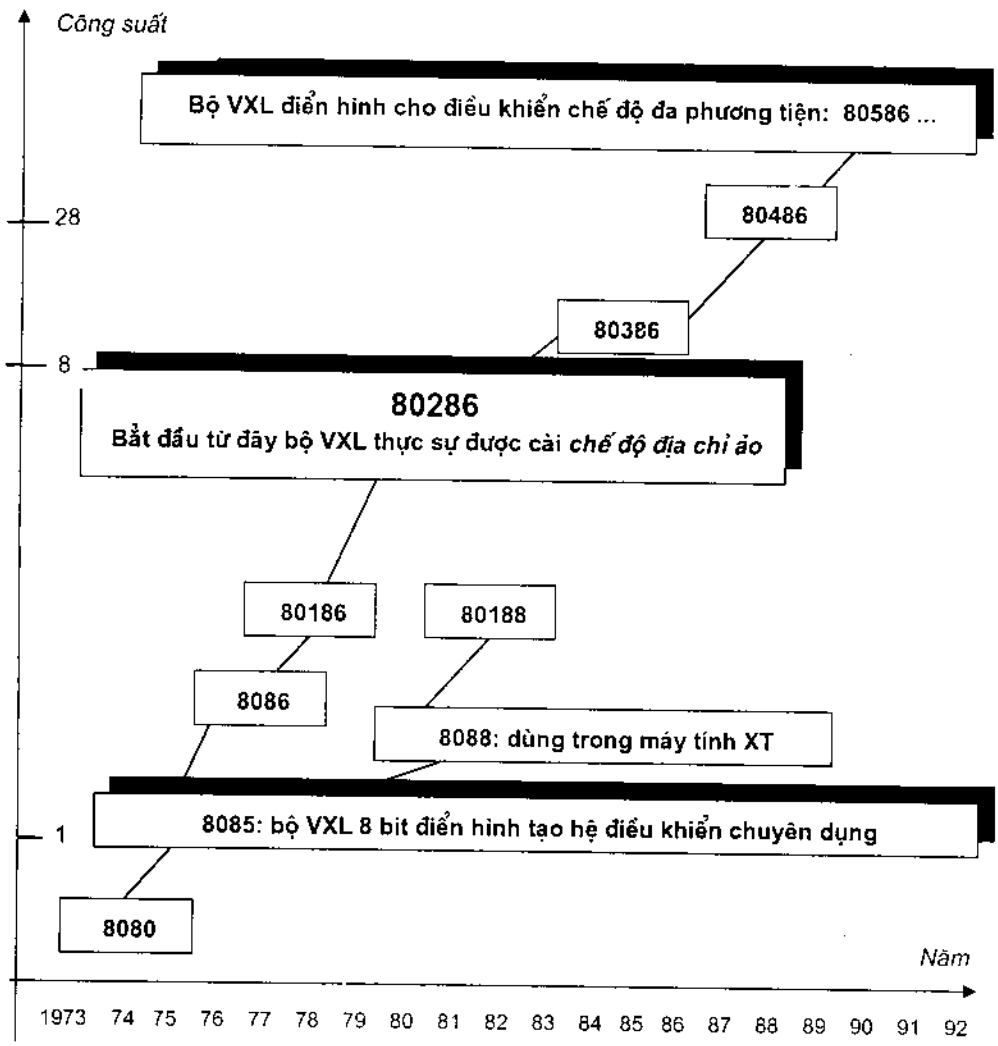
Công việc này được thực hiện một phần nhờ bộ xử lý trung tâm, một phần nhờ hệ điều hành. MP 80386, 80486, 80586... là hiện thân của trình độ kỹ thuật và công nghệ hiện đại. Chúng có một bộ lệnh lớn hơn so với 80286, chủ yếu là cho các chức năng ở chế độ địa chỉ ảo cho bộ nhớ.

Các bộ vi xử lý đều có khả năng tương thích theo hướng phát triển (hình 1.2). Các chương trình viết bằng ngôn ngữ máy cho 8086 có thể chạy trên các bộ vi xử lý cấp cao hơn. Nhưng ngược lại thì không hoàn toàn đúng. Một chương trình viết chuyên cho 80586 sẽ có khả năng rất lớn và nó không chạy được trên 8088 và cả trên 80286.

1.1.2. Trường thanh ghi đa năng của bộ vi xử lý

Trường thanh ghi đa năng của bộ vi xử lý là tập hợp các ngăn nhớ không nằm trong RAM của máy tính, mà nằm trực tiếp trong bộ vi xử. Bộ vi xử lý có thể truy nhập rất nhanh đến chúng so với thời gian truy nhập vào RAM. Bộ xử lý và bộ nhớ RAM về thực chất tạo thành hai thành phần riêng biệt của một hệ xử lý tin. Khi làm việc, bộ xử lý cần phải trao đổi số liệu với RAM, điều đó gây ra một thời gian trễ nhất định. Nhưng nếu chúng ta thực hiện những thao tác cần thiết với thanh

ghi của bộ vi xử lý, sau đó mới gửi kết quả ra RAM, thì số lần phải trao đổi với RAM sẽ giảm đáng kể, và do vậy, tốc độ xử lý sẽ nhanh hơn. Đó là một trong những lý do phải tổ chức các thanh ghi trong bộ vi xử lý và dung lượng của chúng càng lớn càng tốt.



Hình 1.2. **Sự phát triển của họ vi xử lý INTEL 80X86**

Các thanh ghi có độ dài tính theo bit là 8, 16, 32 bít. Các bộ vi xử lý từ 80286 trở về trước chỉ có các thanh ghi có độ dài tính theo bit là 8, 16. Giá trị nhỏ nhất mà một thanh ghi có thể nhận là giá trị 0, giá trị lớn

nhất là 65535. Từ MP 80386 trở đi có thêm các thanh ghi 32 bit. Dựa theo chức năng mà thanh ghi đảm nhận trong quá trình thực hiện chương trình, ta có thể chia thanh ghi thành 4 nhóm:

- Các thanh ghi đa năng (AX, BX, CX, DX, SI. DI, BP)
- Các thanh ghi quản lý mảng nhớ (CS. DS, SS, ES)
- Bộ đếm chương trình hay còn gọi là con trỏ chương trình (IP)
- Thanh ghi cờ trạng thái (F).

Vai trò các thanh ghi đa năng AX, BX, CX, và DX... có tầm quan trọng lớn khi kích hoạt các hàm của DOS hay kích hoạt các chương trình con trong ROMBIOS. Thông qua các thanh ghi này, ta sẽ lệnh cho hệ điều hành thực hiện những nhiệm vụ cần thiết hoặc để truyền tham số cho các chương trình con được gọi. Các thanh ghi đa năng được dùng chủ yếu trong các thao tác số học và các thao tác vào/ra để liên lạc với các thiết bị ngoại vi chuẩn cũng như phi chuẩn của PC.

Các thanh ghi đa năng 16 bit AX, BX, CX, DX còn có thể được chia ra thành các thanh ghi 8 bit (hình 1.3)

Hình 1.3. **Cấu trúc của thanh ghi 16 bit của bộ vi xử lý.**

Các thanh ghi 8 bit được gọi là H (High = 8 bit cao) và L (Low = 8 bit thấp). Thanh ghi 16 bit AX tạo thành từ hai thanh ghi 8 bit AH và AL. Thanh ghi L chứa các bit từ 0 đến 7, thanh ghi H chứa các bit từ 8 đến15 của thanh ghi X. Thí dụ, thanh ghi AH được tạo thành từ các bit 8 đến 15 và thanh ghi AL từ các bit 0 đến 7 của thanh ghi AX.

Tuy nhiên, ba thanh ghi AX, AH và AL không hoàn toàn độc lập với nhau. Thí dụ, khi bit 3 của thanh ghi AH bị thay đổi thì tương ứng bit 11 của AX sẽ bị thay đổi. chúng ta không thể thay đổi thanh ghi AH mà giữ nguyên được AX. Nhưng thanh ghi AL không bị thay đổi khi ta thay đổi AH, vì chúng không chứa các bit chung nhau. Mối quan hệ giữa các thanh ghi AX, AH và AL cũng đúng cho các cặp thanh ghi của BX, CX và DX.

Chúng ta có thể tính giá trị của thanh ghi X dựa trên giá trị của các thanh ghi H và L, và ngược lại. Để nhận được giá trị của X, ta nhân giá trị của H với 256 và cộng với giá trị của L.

Chúng ta minh họa điều đó thông qua thí dụ sau: giả sử giá trị của CH là 10, của CL là 118. Giá trị của CX được tính theo công thức: CH*256 + Cl, có nghĩa là 10*256 + 118 = 2678.

Nhờ có các thanh ghi 8 bit mà ta có thể đọc thông tin từ RAM, ghi thông tin vào RAM các số liệu 8 bit khi cần. Với các số liệu có độ dài lớn hơn 16 bit, ta phải truyền từng 16 bit một. Vì MP 80x86 có cả các thanh ghi 8 bit và 16 bit, nên ta phải xác định trong từng trường hợp cụ thể sẽ làm việc với số liệu 8 bít hay 16 bit.

1.1.3. Xác định địa chỉ cho không gian nhớ của PC

Để hiểu rõ ý nghĩa của một nhóm các thanh ghi khác - nhóm các thanh ghi mảng, chúng ta cần nhớ lại lịch sử phát triển của MP 80x86. Các bộ vi xử lý cao cấp (loại 16, 32 bit) không những có tập lệnh lớn hơn tập lệnh của các vi xử lý 8 bit (8085,6520, Z80...), mà còn có khả năng vượt qua giới hạn địa chỉ hóa của các vi xử lý 8 bít - 64 KB. Điều đó là cần thiết, bởi lẽ các bộ xử lý có tập lệnh lớn hơn thì có khả năng giải quyết các bài toán phức tạp hơn, và do vậy, nhất thiết phải có bộ nhớ lớn hơn. Những người thiết kế MP 8088 đặt ra một mục tiêu rất cao là, không dừng lại ở chỗ chỉ tăng gấp đôi khả năng địa chỉ hoá, mà phải đạt tới bộ nhớ 1 MB, có nghĩa là tăng bộ nhớ lên 16 lần.

Nhưng vấn đề là ở chỗ, số lượng các ngăn nhớ mà bộ xử lý có thể truy nhập được phụ thuộc trực tiếp vào kích thước của một thanh ghi đặc biệt - thanh ghi địa chỉ. Mỗi ngăn nhớ chỉ có thể được gọi thông qua một giá trị tương ứng được đặt trong thanh ghi địa chỉ, do vậy, thanh ghi này quyết định số lượng ngăn nhớ của bộ nhớ. Đối với các bộ vi xử lý trước 8088, thanh ghi địa chỉ là 16 bit. Nó chỉ có thể nhận được các giá trị trong khoảng từ 0 đến 65535, và do vậy, nó không thể địa chỉ hóa bộ nhớ lớn hơn 65536 ngăn (ngăn nhớ 0 cũng phải được tính). Nếu chúng ta muốn truy nhập hơn 1 triệu ngăn nhớ, thì phải có thanh ghi địa chỉ kích

thước lớn hơn 16 bit. Cần phải có 20 bit để đạt được một vùng địa chỉ là 1 MB (2^{20} bytes = 1024 KB = 1 MB). Cách giải quyết có vẻ không đến nỗi là phức tạp, về nguyên tắc, chỉ cần thêm một thanh ghi địa chỉ 20 bit.

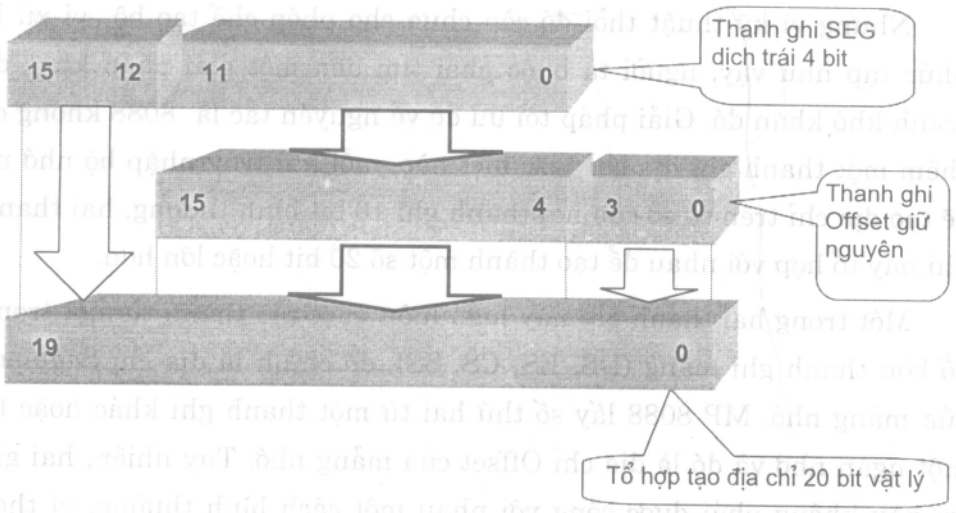
Nhưng vì kỹ thuật thời đó còn chưa cho phép chế tạo bộ vi xử lý phức tạp như vậy, người ta buộc phải tìm đến một giải pháp khác để tránh khó khăn đó. Giải pháp tối ưu đó về nguyên tắc là 8088 không có thêm một thanh ghi địa chỉ đặc biệt nào, mỗi khi truy nhập bộ nhớ nó sẽ tạo địa chỉ trên cơ sở của hai thanh ghi 16 bit bình thương, hai thanh ghi này tổ hợp với nhau để tạo thành một số 20 bit hoặc lớn hơn.

Một trong hai thanh ghi này luôn luôn được tạo thành từ một trong số bốn thanh ghi mảng (DS, ES, CS, SS), đó chính là địa chỉ Segment của mảng nhớ. MP 8088 lấy số thứ hai từ một thanh ghi khác hoặc từ một ngăn nhớ và đó là địa chỉ Offset của mảng nhớ. Tuy nhiên, hai giá trị này không phải được cộng với nhau một cách bình thường, vì theo cách bình thường thì ta không thể nhận được một số 20 bit, mà chỉ một số 17 bit là lớn nhất. Sự tổ hợp này phức tạp hơn nhiều, đầu tiên thanh ghi mảng được dịch trái 4 bit (tức là 1 số hexa), điều đó tương đương với việc nhân giá trị cũ của nó với 16. Sau đó, giá trị địa chỉ thứ hai được cộng với địa chỉ 20 bit vừa nhận dược bằng cách trên để tạo ra 20 bit địa chỉ. Địa chỉ offset cung cấp cho ta vị trí chính xác của ngăn nhớ trong một mảng. Bởi lẽ thanh ghi offset không thể có kích thước lớn hơn 16 bit, nên một đoạn có kích thước là 2^{16} bytes, có nghĩa là 64 KB.

Địa chỉ được tính từ tổ hợp của hai địa chỉ mảng và offset gọi là địa chỉ vật lý, nó chỉ ra địa chỉ tuyệt đối của ngăn nhớ còn bản thân địa chỉ mảng và địa chỉ offset được gọi là địa chỉ logic.

Tất nhiên, một mảng không thể bắt đầu từ một ngăn nhớ bất kỳ của bộ nhớ, bởi lẽ thanh ghi mảng được nhân với 16 để tạo thành địa chỉ mảng, hay địa chỉ bắt đầu của một mảng. Do vậy, địa chỉ bắt đầu của một mảng luôn là bội số của 16. Thí dụ, không thể bắt đầu một đoạn từ địa chỉ 22. Nguyên lý tổ hợp địa chỉ mảng nhớ và địa chỉ offset đã mở đầu một cách viết rất đặc biệt để chỉ ra địa chỉ của một ngăn nhớ nhớ.

Trước tiên, người ta viết địa chỉ mảng, rồi sau dấu hai chấm là địa chỉ offset của ngăn nhớ. Cả hai địa chỉ đều được viết dưới dạng số HEXA, mỗi địa chỉ là 4 số HEXA.



Hình 1.4. **Minh hoạ cách tính địa chỉ ngăn nhớ từ địa chỉ Segment và địa chỉ offset (minh hoạ cho bộ VXL 80286)**

Thí dụ, để chỉ ra địa chỉ của một biến trong bộ nhớ có địa chỉ mảng là 2000 h và địa chỉ offset là AF3h, người ta viết 2000:0AF3.

Phương pháp tính địa chỉ ngăn nhớ cho các bộ vi xử lý 80486, 80586... phức tạp hơn nhằm mục đích quản lý không gian nhớ lớn hơn nữa (đạt tới 512 MB).

Như chúng ta đã nói ở trên, 80X86 có 4 thanh ghi mảng đóng vai trò rất quan trọng trong quá trình thực hiện một chương trình. Thanh ghi mảng mã lệnh CS (Code Segment) cùng với thanh ghi IP (Instruction Pointer) đóng vai trò thanh ghi offset xác định địa chỉ của ngăn nhớ mà ở đó chứa lệnh tiếp theo của chương trình.

Mỗi khi bộ vi xử lý đọc xong một lệnh, thanh ghi IP tự động tăng lên sao cho nó chỉ đến một lệnh tiếp theo của chương trình. Lưu ý rằng một lệnh có thể là lệnh một byte hay lệnh nhiều byte. Nhờ vậy mà chương trình được thực hiện liên tục theo từng lệnh.

Khác với thanh ghi CS, thanh ghi mảng dữ liệu DS (Data Segment) được dùng để tạo thành địa chỉ mảng dữ liệu để cho bộ xử lý truy nhập đến dữ liệu của chương trình. Địa chỉ để cộng với DS có thể là nội dung của một thanh ghi khác (DX, BX, SI, DI...).

Vùng nhớ mà địa chỉ bắt đầu của nó được xác định bởi thanh ghi mảng ngăn xếp SS (Stack Segment) được gọi là ngăn xếp (Stack). Nó được dùng cho nhóm lệnh lưu trữ những thông tin trạng thái quan trọng đối với chương trình. Thí dụ như khi thực hiện lệnh CALL, là lệnh gọi chương trình con, thì địa chỉ mà chương trình cần phải trả về về sau khi thực hiện xong chương trình con, thực chất là địa chỉ của lệnh sau lệnh CALL, phải được cất giữ vào ngăn xếp. Khi truy nhập ngăn xếp, địa chỉ được tạo thành từ thanh ghi SS và thanh ghi SP hoặc BP.

Cuối cùng là thanh ghi mảng mở rộng ES (Extra Segment). Thanh ghi này rất quan trọng trong các thao tác chuyển dữ liệu. Nếu chúng ta chỉ dùng một thanh ghi mảng (thí dụ DS) để chỉ ra địa chỉ nguồn và đích, thì chúng ta không thể chuyển dữ liệu ra khỏi mảng đó để đến một vùng khác của không gian nhớ. Để giải quyết vấn đề này, người ta dùng một thanh ghi địa chỉ nguồn và một thanh ghi khác để chỉ ra địa chỉ đích, hai thanh ghi dùng cho công việc này thường là DS và ES. Khi thực hiện lệnh chuyển dữ liệu, nội dung của thanh ghi DI hoặc SI được cộng với nội dung của thanh ghi ES.

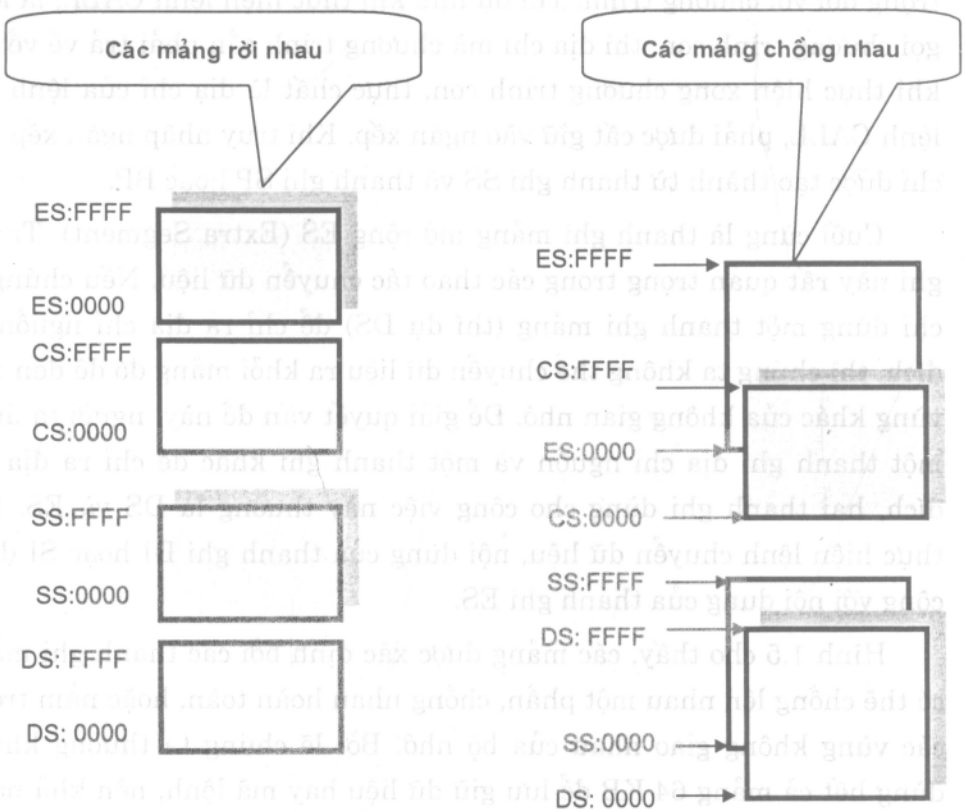
Hình 1.5 cho thấy, các mảng được xác định bởi các thanh ghi mảng có thể chồng lên nhau một phần, chồng nhau hoàn toàn, hoặc nằm trong các vùng không giao nhau của bộ nhớ. Bởi lẽ chúng ta thường không dùng hết cả mảng 64 KB để lưu giữ dữ liệu hay mã lệnh, nên khả năng thứ nhất trong ba khả năng trên là rất thực tế, bởi vì nó cho phép đặt dữ liệu (thanh ghi DS) ngay sau vùng lệnh (thanh ghi CS), và như vậy bộ nhớ được tiết kiệm rất nhiều.

Thanh ghi quan trọng cuối cùng đối với chúng ta là thanh ghi cờ. Các bit của thanh ghi cờ thông báo trạng thái của bộ xử lý sau mỗi lần thực hiện một lệnh assembler. Thí dụ, khi thực hiện một lệnh số học mà kết quả là âm thì cờ dấu S (Sign) sẽ thiết lập trạng thái 1. Cờ nhớ C

(Carry) sẽ thiết lập 1 nếu phép cộng hai số 8 bit cho kết quả là một số lớn hơn sức chứa của thanh ghi 8 bit.

Bộ xử lý không sử dụng hết 16 bit của thanh ghi cờ. Các bit không sử dụng thường nhận giá trị 0.

Các chương trình thí dụ minh họa việc sử dụng các hàm của BIOS, DOS sẽ giúp chúng ta sáng tỏ vấn đề được nêu ở trên.



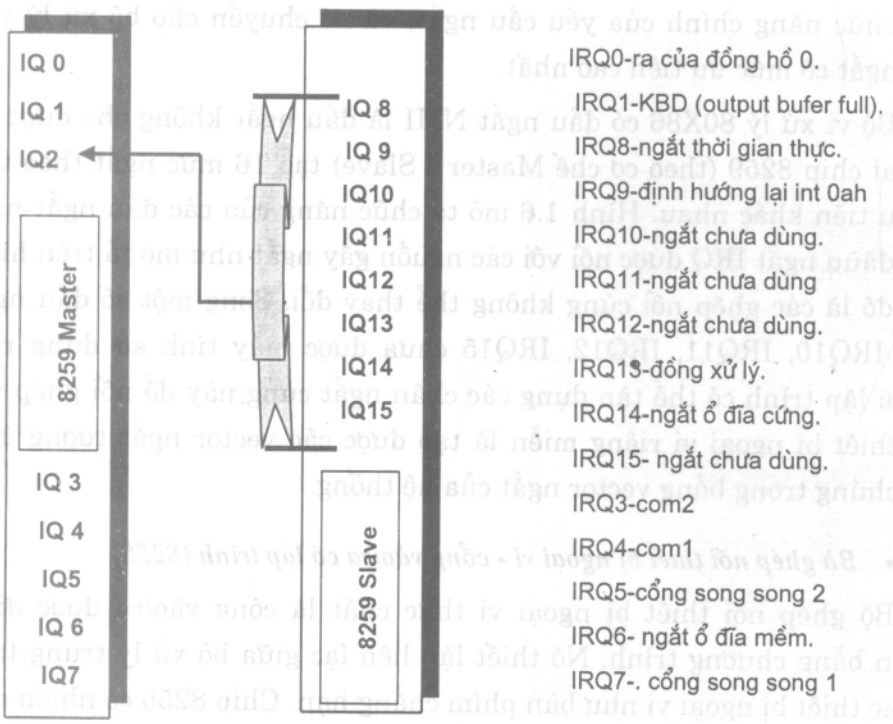
Hình 1.5. **Các mảng chồng và không chồng lên nhau**

1.1.4. Các chip IC bổ trợ cho bộ xử lý trung tâm

Mặc dù bộ vi xử lý là bộ não của máy tính PC và là chip IC thông minh nhất của một hệ thống, nó cũng không thể giám sát tất cả các hoạt động của một hệ phức tạp như máy tính PC. Bởi vậy mà người ta phải

bổ trợ thêm cho bộ vi xử lý một số chip IC chuyên dụng (Support Chips). Chúng cho phép giảm nhẹ một phần công việc của bộ vi xử lý và cho phép nó tập trung vào công việc chính là việc thực hiện chương trình. Các chip này được dùng trong việc liên lạc và điều khiển các thiết bị ngoại vi như ổ đĩa hay màn hình.

Một số trong số các chip đó có thể được gọi thông qua hai lệnh IN, OUT, và nhờ vậy, chúng có thể được lập trình hoặc đặt lại cấu hình. Bởi lẽ việc lập trình hệ thống không đơn giản, nên trước khi lxây dựng chương trình điều khiển hệ thống phải nghiên cứu, tìm hiểu kỹ cấu trúc và hoạt động của các đối tượng chấp hành. Sau đây là các chip bổ trợ chính cho bộ vi xử lý.



Hình 1.6. **Mô tả chức năng của các đầu ngắt cứng**

- ***Bộ điều khiển chế độ truy nhập trực tiếp DMA (8237 DMAC)***

Tên gọi của chip này (Direct Memory Access Controller) là dựa trên tính chất đọc và ghi số liệu trực tiếp vào RAM không qua bộ vi xử lý.

Khả năng này đặc biệt được dùng để vào/ra thông tin với đĩa trên ổ đĩa, là công việc tương đối chậm. Bằng cách đó, người ta có thể giảm nhẹ công việc cho bộ vi xử lý và tăng tốc độ thực hiện chương trình.

- *Bộ điều khiển cơ chế ngắt (8259)*

Các tín hiệu ngắt cho phép các đối tượng khác nhau của hệ thống yêu cầu bộ xử lý trung tâm tạm dừng công việc của mình để phục vụ chúng. Vì có nhiều ngắt (đến từ các thành phần khác nhau của hệ thống) có thể xuất hiện đồng thời cùng một lúc, nên trước tiên, chúng cần phải được chuyển đến bộ điều khiển ngắt, và sau đó, bộ điều khiển ngắt mới chuyển chúng đến bộ xử lý trung tâm. Để thực hiện công việc này, bộ điều khiển ngắt gán cho mỗi yêu cầu ngắt một mức ưu tiên dựa vào chức năng chính của yêu cầu ngắt, và nó chuyển cho bộ xử lý yêu cầu ngắt có mức ưu tiên cao nhất.

Bộ vi xử lý 80X86 có đầu ngắt NMI là đầu ngắt không che được và có hai chip 8259 (theo cơ chế Master - Slave) tạo 16 mức ngắt theo thứ tự ưu tiên khác nhau. Hình 1.6 mô tả chức năng của các đầu ngắt này. Các đầu ngắt IRQ được nối với các nguồn gây ngắt như mô tả trên hình 1.6, đó là các ghép nối cứng không thể thay đổi. Song một số đầu ngắt như IRQ10, IRQ11, IRQ12, IRQ15 chưa được máy tính sử dụng nên người lập trình có thể tận dụng các chân ngắt cứng này để nối ghép với các thiết bị ngoại vi riêng miễn là tạo được các vector ngắt tương ứng cho chúng trong bảng vector ngắt của hệ thống.

- *Bộ ghép nối thiết bị ngoại vi - cổng vào/ra có lập trình (8255)*

Bộ ghép nối thiết bị ngoại vi thực chất là cổng vào/ra được điều khiển bằng chương trình. Nó thiết lập liên lạc giữa bộ xử lý trung tâm và các thiết bị ngoại vi như bàn phím chẳng hạn. Chip 8255 có nhiều chế độ làm việc từ đơn giản đến phức tạp. Các chế độ này đều đặt được bằng chương trình.

- *Bộ tạo nhịp CLK*

Nếu người ta gọi bộ vi xử lý là bộ não của máy tính, thì có thể gọi bộ tạo nhịp là quả tim của hệ thống. Bộ tạo dao động nhịp này dao động với

tần số hàng trăm thậm chí hàng nghìn MHZ và nó tạo nhịp cho bộ vi xử lý và các thành phần khác của hệ thống hoạt động. Bởi vì không phải mọi thành phần của hệ thống đều có thể làm việc ở tần số cao như vậy, nên tần số đó được chia xuống thấp để nhận được các tần số phù hợp với mỗi chip IC cụ thể.

- ***Bộ tạo thời gian(8253)***

Bộ thời gian (timer) được dùng như bộ đếm hoặc bộ tạo thời gian. Nó cho phép đưa ra qua các đầu ra của nó các xung có tần số xác định và không đổi. Tần số của các xung này có thể điều khiển bằng chương trình sao cho mỗi dầu ra sẽ cung cấp một tần số nhất định nào đó. Mỗi đầu ra được nối với một thiết bị ngoại vi. Thí dụ, một đầu ra được nối với loa, đầu ra khác được nối với bộ điều khiển ngắt. Bộ điều khiển ngắt sẽ khởi động ngắt số 8 (ngắt timer) mỗi khi nhận được xung từ bộ thời gian.

- ***Bộ điều khiển màn hình***

Linh kiện chủ yếu của khối này là bộ điều khiển màn hình 6845. Nhiệm vụ của nó là hiển thị hình ảnh lên màn hình dựa vào các dữ liệu được đặt ở một vùng xác định trong RAM của PC. Để thực hiện công việc này, nó có cả một loạt các thanh ghi bên trong các thanh ghi này điều khiển việc hiển thị hình ảnh lên màn hình và chúng có thể lập trình được.

- ***Bộ điều khiển đĩa***

Chip này cũng không nằm trên bảng mẹ mà nằm trên card mở rộng. Nó được hệ điều hành gọi, và nó điều khiển trực tiếp hoạt động của ổ đĩa. Thí dụ, nó có thể di chuyển đầu từ đọc/ ghi đến rãnh bất kỳ, nó cũng có thể đọc hoặc ghi dữ liệu lên đĩa.

- ***SLOT - Kênh tổng hợp***

Kênh tổng hợp SLOT cung cấp:

- Địa chỉ cổng IO từ 100h đến 3FF h;
- Các bit của kênh địa chỉ;
- Lựa chọn kiểu dữ liệu để trao đổi dữ liệu thông tin;

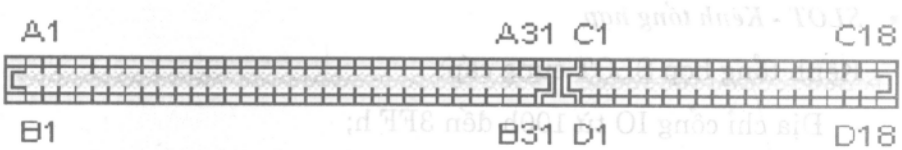
- Sử dụng các ngắt cứng;
- Sử dụng cơ chế DMA;
- Sử dụng cơ chế làm tươi bộ nhớ.
- Một số chức năng khác như điện áp nguồn nuôi các mức điện áp chuẩn.

Như vậy là các nhà sản xuất máy tính đã giành cho người sử dụng một số rãnh cắm đặc blệt kiểu SLOT ghép các thiết bị phần cứng vào máy tính. Tùy từng mục đích sử dụng mà có thể đưa thêm vào các CARD ghép nối để mở rộng khả năng của máy tính.

Ở máy tính PC thế hệ đầu có SLOT với độ rộng kênh dữ liệu cơ bản là 8 bít và tuân theo tiêu chuẩn ISA (Industry Standard Architecture). Từ máy tính AT trở đi việc bố trí chân tín hiệu trên SLOT trở nên phức tạp hơn, tùy theo tiêu chuẩn được lựa chọn khi chế tạo máy tính. Cấu trúc SLOT theo những tiêu chuẩn khác nhau có thể kể ra như sau:

- Rãnh cắm 16 bít theo chuẩn ISA (Industry Standard Architecture).
- Rãnh cắm PS/2 với 16 bít theo chuẩn MCA (Micro Channel Architecture).
- Rãnh cắm PS/2 với 32 bít theo chuẩn MCA (Micro Channel Architecture).
- Rãnh cắm 32 bít theo chuẩn EISA (Extended Industry Standard Architecture).
- Rãnh cắm 32 bít theo chuẩn VESA VLB (VESA Local Bus Standard).
- Rãnh cắm 32 bít theo chuẩn PCI (Peripheral Component Interconect - Standard).

Rãnh cắm theo chuẩn ISA:



Hình 1.7. SLOT ISA

Thông thường SLOT có 62 đường tín hiệu dùng cho mục đích thông tin với một Card cắm vào. Về cơ bản các đường tín hiệu này được chia thành các đường dẫn tín hiệu, đường dẫn địa chỉ và đường dẫn điều khiển. Bởi vì ngay từ máy tính PC/XT đã sẵn có các SLOT 62 chân này là SLOT có kênh dữ liệu là 8 bít nên chỉ những Card 8 bít mới cắm được vào SLOT này. Về sau máy tính PC/ AT (từ 80286 trở đi) ra đời có thêm một SLOT thứ hai nằm thẳng hàng với SLOT 8 bít kể trên và có 36 chân. Như vậy tổng cộng Slot của máy tính PC/AT có tất cả 98 chân.

Slot có kiểu 62 chân và loại bổ xung 36 chân. Sơ đồ chân nối của chúng được mô tả ở hình 1.7. Còn chức năng của các chân tín hiệu cho ở bảng 1.1.

Bảng 1.1a: **ISA SLOT chính**

Chân Tín hiệu	*Chức năng*	*Hướng tín hiệu*	*Chân tín hiệu*	*Chức năng*	*Hướng tín hiệu*
(1)	(2)	(3)	(4)	(5)	(6)
A1	I/O CHCK	I	B1	GND	
A2	SD7	IO	B2	RESET DRV	O
A3	SD6	IO	B3	+5VDC	
A4	SD5	IO	B4	IRQ9	I
A5	SD4	IO	B5	-5VDC	
A6	SD3	IO	B6	DRQ2	I
A7	SD2	IO	B7	-12VDC	
A8	SD1	IO	B8	OWS	I
A9	SD0	IO	B9	+12VDC	
A10	I/O CH RDY	I	B10	GND	

(1)	(2)	(3)	(4)	(5)	(6)
A11	AEN	O	B11	-SMEMW	O
A12	SA19	IO	B12	-SMEMR	O
A13	SA18	IO	B13	-IOW	IO
A14	SA17	IO	B14	-IOR	IO
A15	SA16	IO	B15	-DACK3	O
A16	SA15	IO	B16	DRQ3	I
A17	SA14	IO	B17	-DACK1	O
A18	SA13	IO	B18	DRQ1	I
A19	SA12	IO	B19	-REFRESH	IO
A20	SA11	IO	B20	CLK	O
A21	SA10	IO	B21	IRQ7	I
A22	SA9	IO	B22	IRQ6	I
A23	SA8	IO	B23	IRQ5	I
A24	SA7	IO	B24	IRQ4	I
A25	SA6	IO	B25	IRQ3	I
A26	SA5	IO	B26	-DACK2	O
A27	SA4	IO	B27	T/C	O
A28	SA3	IO	B28	BALE	O
A 29	SA2	IO	B29	+5VDC	
A 30	SA1	IO	B30	óC	O
A 31	SA0	IO	B31	GND	

Bảng 1.1b: ISA SLOT phụ

Chân tín hiệu	*Chức năng*	*Hướng tín hiệu*	*Chân tín hiệu*	*Chức năng*	*Hướng tín hiệu*
C1	SDHE	IO	D1	-MEM CS16	I
C2	LA23	IO	D2	-IO CS16	I
C3	LA22	IO	D3	IRQ10	I
C4	LA21	IO	D4	IRQ11	I
C5	LA20	IO	D5	IRQ12	I
C6	LA19	IO	D6	IRQ15	I
C7	LA18	IO	D7	IRQ14	I
C8	LA17	IO	D8	-DACK0	O
C9	-MEMR	IO	D9	DRQ0	I
C10	-MEMW	IO	D10	-DACK5	O
C11	SD8	IO	D11	DRQ5	I
C12	SD9	IO	D12	-DACK6	O
C13	SD10	IO	D13	DRQ6	I
C14	SD11	IO	D14	-DACK7	O
C15	SD12	IO	D15	DRQ7	I
C16	SD13	IO	D16	+5VDC	
C17	SD14	IO	D17	-MASTER	I
C18	SD15	IO	D18	GND	

Từ cách sắp xếp chân ra, rõ ràng là 98 đường tín hiệu của SLOT nằm cả ở mặt hàn chân cũng như mặt linh kiện. Do đó các Card cắm thêm vào bao giờ cũng là những Card có cấu trúc là mạch in hai mặt.

Bảng dưới đây sẽ mô tả các chân tín hiệu trên SLOT của PC. Tất cả các đường tín hiệu đều tương thích với mức TTL.

Tín hiệu	*Hướng*	*Mô tả*
(1)	(2)	(3)
A0- A19	Ra	Các bít địa chỉ từ A0 đến A19 được sử dụng để đánh địa chỉ bộ nhớ và mạch I/O bên trong hệ thống. Có 20 đường địa chỉ thêm vào đó LA 17 đến LA 23 cho phép thâm nhập tới 16 Mb bộ nhớ. Từ A0 tới A19 cho qua cổng chuyển lên Bus hệ thống khi "ALE " ở mức cao và được chốt trên sườn xuống của "ALE ". Các tín hiệu này được tạo ra bởi bộ vi xử lý hoặc bộ điều khiển DMA. Chúng cũng có thể được điều khiển bởi các bộ vi xử lý khác hoặc bộ điều khiển DMA thuộc kênh I/O.
LA17-LA23	Ra	Các tín hiệu này (không chốt) được sử dụng để đánh địa chỉ bộ nhớ và mạch vào ra (I/O) bên trong hệ thống. Các tín hiệu này có hiệu lực khi "ALE" ở mức cao. Mục đích của chúng là tạo ra mã nhớ cho chu kì nhớ 1 trạng thái đợi. Các mã này sẽ bị chốt ở sườn xuống của "ALE" bởi bộ phối hợp I/O. Các tín hiệu này cũng có thể được điều khiển bởi bộ vi xử lý khác hoặc bộ điều khiển DMA thuộc kênh I/O.
CLK	Ra	Nhịp đồng hồ hệ thống của PC. Nó có chu kì xử lý đồng bộ với chu kì thời gian là 167 ns. Thời gian làm việc của chu kỳ đồng hồ là 50%. Được sử dụng để đồng bộ.
Reset DRV	Ra	"Reset drive " cấp tín hiệu Reset để thiết lập lại hoặc khởi động lại hệ thống lôgíc ở thời điểm bật máy. Sau khi bật máy tính, đường dẫn Reset sẽ kích hoạt trong thời gian ngắn, để đưa Card đã cắm thêm vào trạng thái ban đầu. Tín hiệu này tích cực ở mức cao.
D0 – D7 SD08-SD15	Vào/ra	Đường tín hiệu này cung cấp Bus dữ liệu từ bit 0 đến bit 15 cho bộ vi xử lý, bộ nhớ và các thiết bị vào ra. D0 là bit có giá trị thấp nhất và D15 là bit có giá trị cao nhất. Tất cả các thiết bị 8 bít trên kênh I/O sẽ sử dụng từ D0 đến D7 để truyền tới bộ vi xử lý. Các thiết bị 16 bit sẽ sử dụng thêm từ SD08 đến SD15. Đối với các thiết bị phụ trợ 8 bít thì dữ liệu từ SD08 đến SD15 sẽ được cho qua tới D0 đến D7 trong khi truyền 8 bít tới các thiết bị này. Còn các bộ vi xử lý 16 bít khi truyền tới các thiết bị 8 bít sẽ được chuyển đổi thành 2 lần truyền 8 bít.

(1)	(2)	(3)
ALE	Ra	"Address latch enable" được phát từ bộ điều khiển Bus 82288 và được sử dụng trên bản mạch hệ thống để chốt các địa chỉ hợp lệ và mã bộ nhớ từ bộ vi xử lý. Nó có hiệu lực đối với kênh I/O như là con trỏ của bộ vi xử lí hợp lệ hoặc địa chỉ DMA (Khi sử dụng với "AEN "). Địa chỉ bộ vi xử lý từ A0 đến A19 được chốt ở sườn xuống của "ALE ". "ALE " bị cưỡng bức ở mức cao trong chu kỳ DMA.
/ IO CH CK	Vào	"/IO channel check"- Kiểm tra kênh vào ra. Card ghép nối phát ra tín hiệu này (có sai số tới bản mạch để chỉ thị sai số (Thí dụ sai số chẵn lẻ trên bản mạch nhớ mở rộng). Khi tín hiệu này tích cực, nó chỉ ra lỗi mà hệ thống không sửa được.
IO CH RDY	Vào	"I/O channel ready"- Nhận tín hiệu READY từ các đơn vị địa chỉ trên Card ghép nối. Nếu chân này còn ở mức thấp CPU hoặc Chip DMA sẽ xen vào chu kỳ Bus các chu kỳ đợi. Chu kì này được mở rộng bằng bội số của chu kì đồng hồ.
IRQ3-IRQ7 IRQ9-IRQ12 và IRQ14, IRQ15	Vào	Yêu cầu ngắt 3-7, 9-12, 14-15 được sử dụng làm tín hiệu để báo vi xử lý báo cho thiết bị vào ra (I/O) cần lưu ý. Yêu cầu ngắt được ưu tiên, đối với IRQ9 đến IRQ12 và IRQ14, IRQ15 có ưu tiên cao hơn (IRQ9 là cao nhất) và IRQ3 đến IRQ7 có ưu tiên thấp hơn (IRQ7 là thấp nhất). Yêu cầu ngắt được tạo ra khi đường IRQ tăng từ thấp đến cao. Đường này phải duy trì mức cao cho đến khi bộ vi xử lý báo nhận được yêu cầu ngắt (Chương trình con phục vụ ngắt). Ngắt 13 được sử dụng trên bản mạch hệ thống và không có hiệu lực trên kênh I/O Ngắt 8 được sử dụng cho đồng hồ thời gian thực.
/IOR	Vào/ra	"IO Read". Mức thấp của đường dẫn địa chỉ này báo hiệu sự truy nhập đọc trên một Card mở rộng. Tín hiệu này lệnh cho thiết bị vào ra đưa dữ liệu của nó lên Bus dữ liệu. Trong thời gian này các dữ liệu có giá trị cần phải sắp xếp để rồi sau đó được đón nhận bằng sườn trước. Nó có thể bị điều khiển bởi bộ vi xử lý hoặc bộ điều khiển DMA, hoặc bởi bộ vi xử lý hoặc bộ điều khiển DMA thường trú trên kênh vào ra.

(1)	(2)	(3)
/IOW	Vào/ra	"I/O Write". Tín hiệu này sẽ kích hoạt khi truy nhập ghi lên Card mở rộng. Mức thấp chỉ ra rằng các dữ liệu có giá trị đang chờ để đưa ra ở các Bus dữ liệu. Các dữ liệu được đón nhận bằng sườn trước. Nó có thể được điều khiển bởi bộ vi xử lý hoặc bộ điều khiển DMA trong hệ thống.
/ MEMR /SMEMR	Vào/ra Ra	CPU hoặc DMA muốn đọc số liệu từ bộ nhớ chính. Khi bộ vi xử lý trên kênh vào ra muốn điều khiển "/MEMR ", thì nó phải có đường địa chỉ hợp lệ trên Bus trong 1 chu kì đồng hồ hệ thống trước khi điều khiển " /MEMR " tích cực. Cả hai tín hiệu đều tích cực ở mức thấp.
/MEMW /SMEMW	Vào/ra Ra	CPU hoặc DMA muốn ghi số liệu vào bộ nhớ chính. Khi bộ vi xử lý trên kênh I/O muốn điều khiển "/MEMW", nó phải có đường địa chỉ hợp lệ trên Bus trong một chu kì đồng hồ hệ thống trước khi điều khiển "/MEMW " tích cực. Cả hai tín hiệu đều tích cực ở mức thấp.
DRQ0-DRQ3 DRQ5-DRQ7	Vào	Yêu cầu DMA từ 0 đến 3 và từ 5 đến 7 là các yêu cầu không đồng bộ, được sử dụng bởi các thiết bị ngoại vi và kênh vào ra của bộ vi xử lý để tăng dịch vụ DMA (hoặc điều khiển của hệ thống). Thứ tự ưu tiên: với "DRQ0" có mức ưu tiên cao nhất và "DRQ7" có mức ưu tiên thấp nhất. Yêu cầu được tạo thành khi DRQ tích cực. Đường DRQ phải giữ ở mức cao cho đến khi tín hiệu DACK trở nên tích cực, nếu không yêu cầu DMA bị bỏ qua. Từ "DRQ0" tới "DRQ3 " sẽ thực hiện truyền DMA 8 bít, từ DRQ5 tới DRQ7 sẽ thực hiện truyền 16 bít. "DRQ4" được sử dụng trên bản mạch hệ thống và không có hiệu lực trên kênh vào ra.
/DACK0-DACK3 /DACK5-DACK7	Ra	Các tín hiệu này dùng để chấp nhận yêu cầu DMA từ DRQ1 đến DRQ3 và từ DRQ5 đến DRQ7 còn DACK0 dùng làm tươi bộ nhớ.
AEN	Ra	"Address Enable" Tín hiệu điều khiển AEN được dùng để phân biệt chu trình truy nhập DMA và chu trình truy nhập bộ xử lý. AEN ở mức cao (High) thì DMA sẽ giám sát qua Bus địa chỉ và Bus dữ liệu. Khi AEN ở mức tích cực, nó thông báo rằng bộ điều khiển DMA dùng Bus để truyền số liệu, lúc này CPU và các đơn vị xử lý khác bị tách khỏi Bus hệ thống.

(1)	(2)	(3)
REFRESH	Ra	Tín hiệu này được sử dụng để chỉ ra chu kì làm tươi và được điều khiển bởi bộ vi xử lý.
T/C	Ra	"Terminal count": xung tín hiệu này chỉ thị việc kết thúc chu trình DMA.
SBHE	Vào/ra	"Bus High Enable" lệnh truyền dữ liệu ở byte trên của Bus dữ liệu: SD08 đến SD15. Các thiết bị 16 bit sử dụng " SBHE " để quy định vùng đệm Bus dữ liệu được liên kết với SD08 - SD15.
/MASTER	Vào	Tín hiệu này được sử dụng cho đường DRQ để tăng mức điều khiển của hệ thống. Bộ vi xử lý hoặc bộ điều khiển DMA trên kênh I/O có thể sử dụng DRQ tới 1 kênh DMA ở chế độ phân cấp và nhận "/DACK". Dựa trên việc thu nhận "/DACK" thì I/O của bộ vi xử lý có thể kéo "/MASTER" xuống thấp, cho phép nó điều khiển địa chỉ hệ thống, đường dữ liệu và đường điều khiển (với điều kiện biết 3 trạng thái), sau khi "/MASTER" là mức thấp, I/O của bộ vi xử lý phải đợi 1 chu kì đồng hồ hệ thống trước khi điều khiển đường địa chỉ và đường dữ liệu và 2 chu kì đồng hồ trước khi phát ra lệnh đọc và ghi. Nếu tín hiệu này duy trì ở mức nhỏ hơn 15 µs, bộ nhớ hệ thống có thể bị mất vì sự tạm ngừng của bộ làm tươi.
/MEM CS16	Vào	"MEM 16 Chip Select"- là tín hiệu báo cho hệ thống biết rằng, hiện đang truyền dữ liệu một chu kỳ đợi, 16 bit, chu kỳ nhớ. Được phát ra từ bộ giải mã LA17 - LA23.
/IO CS16	Vào	"I/O 16 bit Chip Select" - là tín hiệu báo cho hệ thống biết rằng, hiện đang truyền dữ liệu: 16 bit. Được phát bởi bộ giải mã địa chỉ "/IO CS16 "; tích cực ở mức thấp.
OSC	Ra	"Oscillator". Tạo tín hiệu nhịp làm việc. Tín hiệu này không đồng bộ với đồng hồ hệ thống của PC. Chu kì làm việc là 50%
OWS	Vào	"Zero Wait State" tín hiệu thông báo cho bộ vi xử lí biết nó có thể hoàn thành chu kì Bus hiện tại mà không cần xen thêm chu kì đợi.

Địa chỉ vào/ ra của Card mở rộng không được phép trùm lên vùng địa chỉ vào/ ra của máy tính. Khi sử dụng một Card mở rộng vào sử dụng, thì việc đầu tiên là phải chú ý tới sự sắp xếp của vùng địa chỉ vào / ra của máy tính PC được chỉ ra trong bảng sau:

Địa chỉ (Hex) vào/ ra	*Chức năng*
000 - 01F	Bộ điều khiển DMA 1 (8232)
020 - 03F	Bộ điều khiển ngắt (8259)
040 - 04H	Bộ phát thời gian (8254)
060 - 06F	Bộ kiểm tra bàn phím (8242)
070 - 07F	Đồng hồ thời gian thực (MC 146818)
080 - 09F	Thanh ghi trang DMA (LS 670)
0A0 - 0AF	Bộ điều khiển ngắt 2 (8259)
0CH - 0CF	Bộ điều khiển DMA 2 (8237)
0E0 - 0EF	Dự trữ cho mảng mạch chính
0F8 - 0FF	Bộ đồng xử lý 80x87
1F0 - 1F8	Bộ điều khiển đĩa cứng
200 - 20F	Cổng dùng cho trò chơi (Game)
278 - 27F	Cổng song song 2 (LPT 2)
2B0 - 2DF	Card EGA 2
2E8 - 2EF	Cổng nối tiếp 4 (COM 4)
2F8 - 2FF	Cổng nối tiếp 2 (COM 2)
300 - 31F	Dùng cho Card mở rộng
320 - 32F	Bộ điều khiển đĩa cứng
360 - 36F	Cổng nối mạng (LAN)
378 - 37F	Cổng song song 1 (LPT 1)
380 - 38F	Cổng nối tiếp đồng bộ 2
3A0 - 3AF	Cổng nối tiếp đồng bộ 1
3B0 - 3B7	Màn hình đơn sắc
3C0 - 3CF	Card EGA
3D0 - 3DF	Card CGA
3E8 - 3EF	Cổng nối tiếp 3 (COM 3)
3F0 - 3F7	Bộ điều khiển đĩa mềm
3F8 - 3FF	Cổng nối tiếp 1 (COM 1)

Từ bảng này ta thấy rõ là các địa chỉ từ 300 đến 31F (Hex) đã được dự tính để dùng cho Card mở rộng. Có 10 đường địa chỉ từ A0 đến A9 để đánh địa chỉ cho vùng này. Thường thì trên một Card mở rộng có nhiều khối chức năng như: bộ biến đổi A/ D, bộ biến đổi D/ A, khối xuất và nhập dữ liệu số và các khối này được điều khiển từ máy tính bởi những địa chỉ khác nhau.

Rãnh cắm theo chuẩn PSI:

- Các chân tín hiệu hệ thống:

  CLK (vào): nhịp truyền dữ liệu qua PCI.

  RTS# (in): khởi đầu lại.

- Các chân tín hiệu địa chỉ và dữ liệu:

  AD[31::0] (3 tr.th): kênh chung cho cả địa chỉ và dữ liệu.

  C/BE[3::0] (3 tr.th): Bus command and byte Enable.

  PAR(3 tr.th): Parity chẵn cho AD[31::0] và C/BE[3::0] và C/BE[3::0].

- Các tín hiệu điều khiển giao tiếp:

  FRAME#: chu kỳ khung.

  IRDY# (Initiator ready): đánh dấu điểm đầu sự hoàn tất của pha dữ liệu hiện hành.

  TRDY# (Taget ready): đánh dấu điểm cuối sự hoàn tất của pha dữ liệu hiện hành.

  STOP# thông báo là đích hiện hành đang yêu cầu master phải dừng thao tác hiện hành.

  LOCK# thông báo cầu bị khoá.

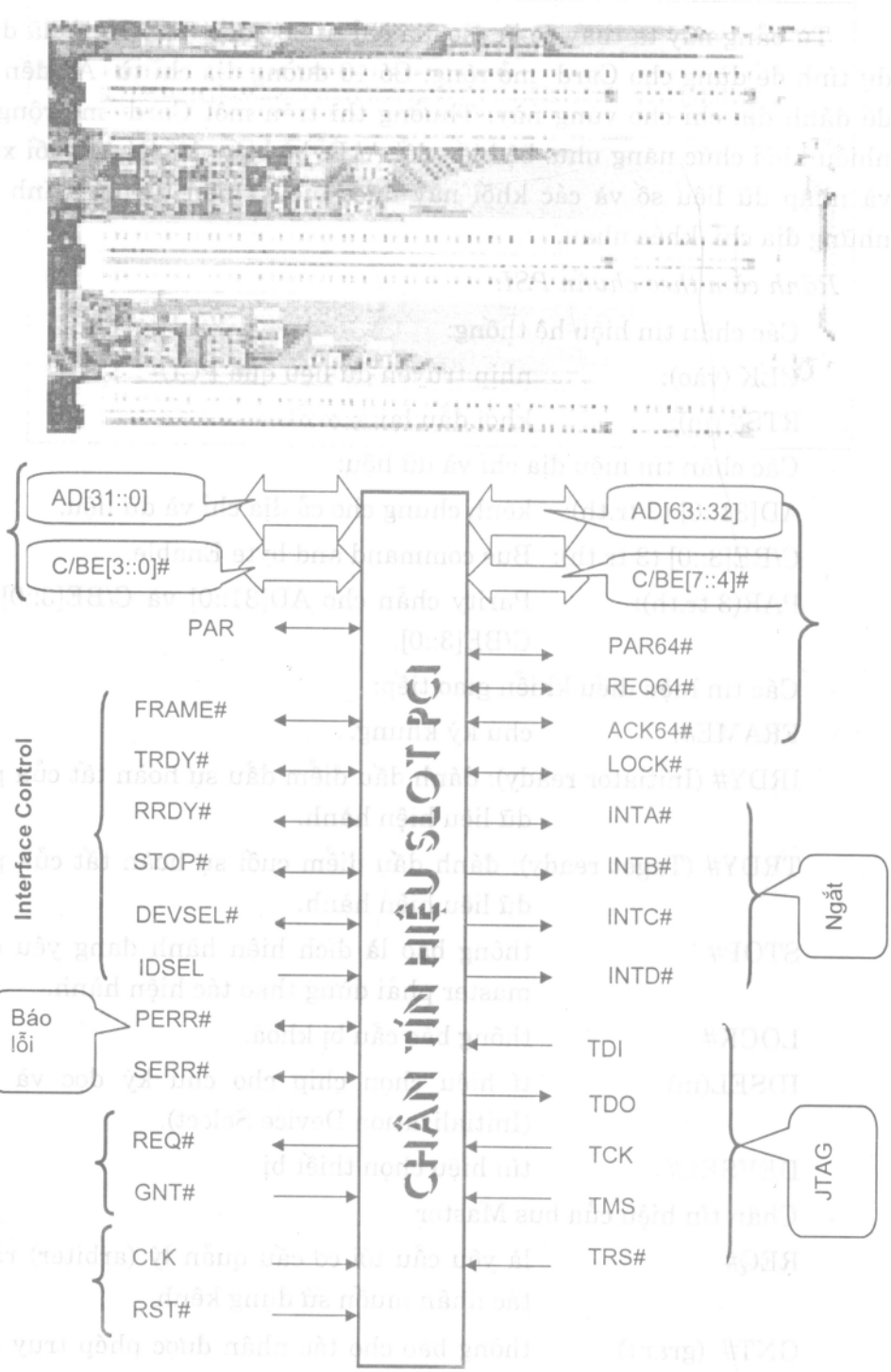
  IDSEL(in) tí hiệu chọn chip cho chu kỳ đọc và ghi (Initialization Device Select).

  DEVSEL# tín hiệu chọn thiết bị.

- Chân tín hiệu của bus Master

  REQ# là yêu cầu tới cơ cấu quản lý (arbiter) rằng tác nhân muốn sử dụng kênh.

  GNT# (grant) thông báo cho tác nhân được phép truy cập bus.



Hình 1. 8. **Chân tín hiệu SLOT PCI**

- Chân tín hiệu thông báo lỗi

  PERR# lỗi parity.

  SERR# lỗi hệ thống.

- Các chân ngắt

  INTA# yêu cầu ngắt.

  INTB#, INTC#, INTD# yêu cầu ngắt và chỉ có nghĩa đối với thiết bị đa chức năng.

- Các tín hiệu bổ xung

  PRSNT[1:2]# (in) Tín hiệu báo sự có mặt (Present).

  CLKRUN# (in) Tín hiệu vào cho thiết bị để xác định trạng thái CLK.

  M66EN (in) Tín hiệu thông báo cho thiết bị biết bus hoạt động ở tốc độ 66 hay 33MH (66MHZ ENABLE).

  PME# Tín hiệu (POWER Management Event) yêu cầu thay đổi trạng thái nguồn.

  3.3Vaux (in) Tín hiệu nguồn 3.3 Volt.

  Kênh mở rộng 64 bit

  AD[62::32] Các chân địa chỉ/ Dữ liệu.

  C/BE[7::4]# Bus Commmand/ Byte Enable.

  REQ64# Yêu cầu truyền kênh 64 bit.

  ACK 64# Chấp nhân truyền kênh 64 bit.

  PAR64 Parity chẵn.

- Tín hiệu quét JTAG/Boundary

  TCK (in) Kiểm tra dữ liệu vào và lệnh cho thiết bị (TEST Clock).

  TDI (in) Kiểm tra dữ liệu dịch nối tiếp vào thiết bị(TEST Data Input).

  TDO (out) Dùng cho kiểm tra dữ liệu dịch nối tiếp và lệnh ra cho thiết bị (TEST Output)

TMS (in)	Điều khiển trạng thái TAP khi chọn chế độ (TEST Mode Select).
TRST# (in)	Khởi đầu cho TAP (Test Reset).

Nhằm tăng tốc quá trình xử lý các thao tác đồ hoạ, các Main Board hiện đại được tổ chức thêm Chip IC AGP.

- ***Bộ đồng xử lý***

Vì MP80286, 80386... không có khả năng xử lý trực tiếp các số dấu phẩy động, các số thực, cho nên trên bảng mẹ của PC dành sẵn một đế cắm cho bộ đồng xử lý toán học. Đối với PC/XT, đó là 8087, đối với AT, đó là 80287, và đối với PC 80386 là 80387...

Việc xử lý số dấu phẩy động (có nghĩa là những số như 3,1445), có thể thực hiện được bằng phần mềm, tức là bằng chương trình, nhưng bộ đồng xử lý toán học có thể thực hiện công việc đó với độ chính xác cao hơn nhiều và với tốc độ hàng trăm lần nhanh hơn. Bộ xử lý 8087 và 80287... không những có thể thực hiện các phép toán cộng trừ, nhân, chia..., mà còn có thể thực hiện các phép tính phức tạp với hàm lượng giác sin, cosin,khai căn bậc hai...

1.2. BỘ NHỚ TRUNG TÂM CỦA MÁY TÍNH PC

Những chip IC mà chúng ta đã xét cho đến nay là các chip thông minh của hệ thống, còn bộ nhớ là cấu trúc thụ động, nó dùng để lưu trữ dữ liệu và trả lại khi cần thiết. Mỗi ngăn nhớ là một byte (8 bit). Chúng được đánh số bắt đầu từ 0.

Để trao đổi thông tin với bộ nhớ, các linh kiện của hệ thống (kể cả bộ vi xử lý) sử dụng kênh dữ liệu hệ thống DATA BUS. Đầu tiên, số thứ tự của ngăn nhớ được gọi, tức là giá trị địa chỉ của nó được đưa lên BUS địa chỉ. BUS này gồm một số xác định các đường dây liên lạc (các bit), các đường này có thể nhận giá trị 0 hoặc 1. Như vậy, các đường này cùng nhau tạo thành một số hệ cơ số 2, giá trị của nó là số thứ tự của ngăn nhớ cần truy nhập. Vì chỉ có thể truy nhập được ngăn nhớ mà số thứ tự của nó biểu diễn được qua các dây của BUS địa chỉ nên số các dây của

BUS địa chỉ quyết định số lượng các ngăn nhớ có thể địa chỉ hóa được. Trong khi PC/XT có BUS địa chỉ 20 bit (20dây), nó cho phép địa chỉ hóa 1 MB, thì AT có BUS địa chỉ 24 bit và nó cho phép địa chỉ hóa được 16 MB. Các bộ vi xử lý cao cấp hơn có thể địa chỉ hóa được dung lượng bộ nhớ lớn hơn nữa.

Sau khi bộ nhớ biết được địa chỉ của ngăn nhớ được gọi, các số liệu có thể được trao đổi giữa cơ cấu gọi và ngăn nhớ được gọi. Kênh dữ liệu DATA BUS được sử dụng trong việc truyền số liệu này. Số lượng dây kênh dữ liệu DATA của BUS quyết định số bit có thể được truyền đồng thời đến hoặc đi từ bộ nhớ.

Đối với máy tính thế hệ cũ PC/XT là 8 bit số liệu, và do vậy, nó chỉ có thể truyền từng byte. Nhưng 8088 lại là bộ xử lý 16 bít, nên các số liệu 16 bit thường được truyền tới bộ nhớ. Số đường dây không đủ để làm việc đó, nên người ta giải quyết vấn đề này bằng cách chia 16 bít thành hai gía trị 8 bít và truyền lần lươt từng byte. Tuy nhiên phương pháp này không tối ưu, không hiệu quả. Vấn đề đó không được đặt ra đối với AT (với máy xây dựng trên cơ sở 80X86) vì nó có BUS số liệu là 16 /32 bit cho phép truyền đồng thời 16/32 bit tới bộ nhớ. Đó chính một trong số các lý do để AT làm việc nhanh hơn PC/XT.

Tất cả các thành viên của họ vi xử lý INTEL 80x86 đều có phương pháp lưu trữ các WORD (16 bit) trong bộ nhớ. Các bit từ d0 đến d7, tức là byte thấp, được đặt ở ngăn nhớ có địa chỉ thấp, và các bit từ d8 đến d15, có nghĩa là byte cao, được đặt ở ngăn nhớ có địa chỉ cao, tiếp ngay sau byte thấp. Thí dụ, nếu một từ 3F87h được cất ở địa chỉ 0000:0400, thì byte thấp 87h được cất ở ngăn nhớ có địa chỉ 0000:0400, còn byte cao là 3Fh ở ngăn nhớ 0000:0401.

Con trỏ địa chỉ cung cấp giá trị địa chỉ ngăn nhớ dưới dạng địa chỉ mảng Seg và địa chỉ số gia Offset. Vì cả địa chỉ mảng và địa chỉ offset đều viết dưới dạng một WORD, nên loại dữ liệu con trỏ chiếm 4 byte trong bộ nhớ. Địa chỉ offset đặt trước địa chỉ mảng, byte thấp của mỗi WORD lại được đặt trước byte cao của WORD đó.

Các từ kép (DWORD) được dùng cho loại dữ liệu con trỏ. Nó thể hiện một số 32 bit được lưu trữ dưới dạng 2 WORD. WORD thấp - từ bit 0 đến 15, được đặt trước WORD cao-từ bit 16 đến 31.

Khi thao tác với bộ nhớ cần lưu ý hai điểm: thứ nhất, khi địa chỉ xác định một ngăn nhớ, cần chỉ rõ là nó nằm trong ROM hay trong RAM. Sự khác nhau giữa ROM và RAM là ở chỗ, nếu chương trình định ghi thông tin vào trong ROM, thì thao tác này sẽ kết thúc không thành công. Thứ hai, chỉ có những ngăn nhớ nào tồn tại một cách vật lý, thì chúng ta mới gọi được nó. Bộ vi xử lý có thể địa chỉ hóa được không gian nhớ cực đại ứng với khả năng của kênh địa chỉ, nhưng không có nghĩa rằng bộ nhớ RAM hay ROM luôn tồn tại trong toàn bộ không gian nhớ đó. Mặc dù bản thân bộ vi xử lý không áp đặt một sự phân chia bộ nhớ nào, nhưng đại đa số các máy tính IBM PC qui định một sự tổ chức bộ nhớ mà các máy tương thích phải tuân thủ. Sự tổ chức này chia vùng địa chỉ cơ sở 1M (1024 KB đầu tiên của không gian nhớ) của bộ vi xử lý thành 16 mảng 64KB.

Mười mảng đầu tiên của bộ nhớ dành cho RAM trung tâm, kích thước cực đại của nó là 640 KB. Kích thước thực của RAM phụ thuộc vào từng loại máy tính, nhưng ít nhất là 64 KB được đặt ở mảng đầu tiên - mảng 0. Nếu RAM mở rộng được thêm vào, thì nó sẽ đặt ngay sau RAM đã cài đặt để tránh các khoảng trống giữa các mảng nhớ RAM. Mảng RAM 0 có vai trò rất quan trọng, vì nó chứa dữ liệu hệ thống và các chương trình con của bảo trì hệ điều hành.

Vùng nhớ RAM kế tiếp theo, mảng A000, dành cho card màn hình đồ họa EGA. Mảng B000 dành cho card màn hình mono và màu. Mảng này được chia làm hai phần: phần RAMVIDEO mono chiếm 32 KB thấp, phần RAMVIDEO màu chiếm 32 KB cao. Tuy nhiên, không gian nhớ này không có nghĩa là đã cài đặt một lượng RAM đủ cho nó. Đối với card mono là 4 KB, card màu là 16 KB.

Các mảng tiếp theo sẽ là ROM, bắt đầu mảng C000. Mảng này được dành cho các chương trình ROM BIOS mở rộng. Vì vùng này còn chưa được sử dụng hết, nên chúng ta có thể giả thiết rằng các chương trình BIOS mở rộng trong tương lai sẽ được đặt ở đây.

[illegible] F000 được dành cho ROM cartridge (ROM có thể cài đặt hoặc rút ra như kiểu băng dạn). Điều này cho thấy các chức năng mở rộng trang bị thêm cho máy tính có thể được cài đặt bằng các chương trình ROM bổ sung. Khả năng này còn chưa được khai thác với PC, và do vậy, vùng này hầu như không được dùng. Đây chính là cấu trúc cho phép người lập trình hệ thống cài chương trình của mình vào một khi muốn chương trình này có thể vận hành máy tính mà không cần sự có mặt của hệ điều hành. Nói cách khác, nó là chương trình MONITOR một khi ta muốn biến máy tính PC thành hệ xử lý chuyên dụng.

Mảng nhớ cuối cùng, đoạn F000 chứa các chương trình thực sự của BIOS, chương trình khởi động hệ thống cũng như BASIC nạp trong ROM.

Bảng 1.2. **Tổ chức bộ nhớ của PC**

Khối	*Địa chỉ*	*Nội dung*
15	F000:0000 - F000:FFFF	ROM của BIOS
14	E000:0000 - E000:FFFF	Dành cho ROM cartridge
13	D000:0000 - D000:FFFF	Dành choROM cartridge
12	C000:0000 - C000:FFFF	ROM BIOS bổ sung
11	B000:0000 - B000:FFFF	RAM màn hình
10	A000:0000 - A000:FFFF	RAM màn hình bổ sung (EGA/VGA)
9	9000:0000 - 9000:FFFF	RAM
8	8000:0000 - 8000:FFFF	RAM
7	7000:0000 - 7000:FFFF	RAM
6	6000:0000 - 6000:FFFF	RAM
5	5000:0000 - 5000:FFFF	RAM
4	4000:0000 - 4000:FFFF	RAM
3	3000:0000 - 3000:FFFF	RAM
2	2000:0000 - 2000:FFFF	RAM
1	1000:0000 - 1000:FFFF	RAM
0	0000:0000 - 0000:FFFF	RAM (Bảng vector ngắt, các biến của DOS và BIOS, hai file ẩn của DOS)

1.3. TỔ CHỨC CƠ CHẾ NGẮT TRONG MÁY TÍNH PC

Nếu các nội dung trên mới cho chúng ta một cái nhìn khái quát về cấu trúc của MP 80xx86 thì ở mục này chúng ta sẽ nghiên cứu sâu hơn chức năng đặc biệt của MP 80x86. Chúng ta sẽ nghiên cứu đến cơ chế ngắt, một cơ chế đóng một vai trò rất quan trọng trong quan điểm xây dựng kiến trúc của MP 80x86. Nhờ chúng mà người lập trình có thể dễ dàng thâm nhập vào hệ điều hành và sử dụng các hàm chức năng của nó để phục vụ cho chương trình của mình. Các hàm chức năng chứa trong ROMBIOS hay trong hệ điều hành chính là tài nguyên phần mềm rất quan trọng của máy tính. Sử dụng thành thạo chúng sẽ làm giảm nhẹ công việc lập trình đi rất nhiều. Vì vậy mục này đề cập tới phương pháp khai thác triệt để khả năng của tài nguyên hệ thống thông qua cơ chế ngắt.

Ngắt là khả năng dừng chương trình chính đang chạy để thực hiện một chương trình khác gọi là chương trình con ngắt. Chương trình con này được kết thúc bằng lệnh IRET (Interrupt RETurn), nó ra lệnh cho bộ xử lý quay về trao trả quyền điều khiển cho chương trình chính đúng từ chỗ mà nó bị ngắt.

Bộ xử lý trung tâm giúp người lập trình không phải thực hiện công việc đó vì quá trình đó là quá trình tự động khi có lệnh ngắt, nhưng người lập trình phải chịu trách nhiệm về việc các thanh ghi (AX, BX, CX, ..) không được thay đổi khi quay về chương trình chính.

Giả sử ta có một chương trình ngôn ngữ máy đang chạy trên máy PC. Chương trình đang thực hiện việc nạp nội dung của một biến vào thanh ghi AX. Lệnh tiếp sau là nhân biến đó với 10 chẳng hạn. Bộ xử lý không đủ thời gian để thực hiện thao tác này, vì một ngắt xuất hiện và bắt nó thực hiện một chương trình xử lý ngắt. giả thiết rằng chương trình được gọi sử dụng thanh ghi AX để thao tác các biến và nó thay đổi giá trị ban đầu của thanh ghi của AX đã bị thay đổi bởi chương trình con, nên lúc này một giá trị hoàn toàn khác trước sẽ được nhân 10. Loại lỗi này sẽ dẫn đến những hậu quả không lường trước được, ít nghiêm

trọng nhất là chương trình cho kết quả sai, nhưng cũng có thể, trong trường hợp xấu nhất, làm treo máy tính. Như vậy, phải thực hiện này.

Chương trình chính được khôi phục, và bây giờ, nó thực hiện nhân giá trị trong AX với 10, công việc mà lúc trước nó chưa kịp làm. Vì giá trị thao tác cất giữ thông tin trạng thái cho chương trình chính trước khi kích hoạt chương trình con và phải có khả năng trao trả quyền điều khiển cho chương trình chính trước khi kết thúc chương trình con.

1.3.1. Cấu trúc bảng các vector ngắt

Chúng ta mới chỉ nói đến cơ chế ngắt nói chung, còn MP 80x86 được trang bị tới 256 ngắt khác nhau, chúng được đánh số từ 0 đến 255. Mỗi ngắt ứng với một chương trình con xử lý ngắt và sẽ được thực hiện khi được gọi. Bảng các vector ngắt là bảng chứa mối quan hệ giữa mỗi ngắt và chương trình con tương ứng. Bảng này chứa địa chỉ bắt đầu của các chương trình con xử lý ngắt, mỗi địa chỉ gọi là một vector.

Khi một ngắt được gọi, bộ xử lý sẽ tự động tìm địa chỉ bắt đầu của chương trình xử lý ngắt tương ứng trong bảng vector ngắt và thực hiện chương trình đó.

Địa chỉ bắt đầu của mỗi chương trình con được xác định trong bảng bởi địa chỉ mảng và địa chỉ offset, địa chỉ offset được đặt trước địa chỉ mảng. Hai địa chỉ này đều là 16 bit (2 byte), như vậy, mỗi địa chỉ ngắt chiếm 4 byte trong bộ nhớ. Độ dài của cả bảng do vậy sẽ là 256*4 = 1024(byte).

Bảng vector ngắt chiếm các ngăn nhớ từ địa chỉ 0 đến 3FFh. Số thứ tự của ngắt bằng số thứ tự của vector ngắt. Địa chỉ của chương trình xử lý ngắt số 0 nằm ở ngăn nhớ 0, có nghĩa là trong các ngăn nhớ từ 0 đến 3h. Sau đó, trong các ngăn nhớ từ 4h đến 7h, là địa chỉ của chương trình xử lý ngắt số 1. Ngắt cuối cùng, ngắt số 255, chiếm 4 ngăn nhớ từ 3FCh đến 3FFh và là kết thúc của bảng.

Để xác định ngăn nhớ chứa địa chỉ bắt đầu của chương trình con xử lý ngắt, ta chỉ cần nhân số thứ tự ngắt với 4. Chúng ta sẽ cần đến bảng vector này mỗi khi chúng ta muốn thay đổi ngắt, có nghĩa là thay đổi

vector ngắt làm cho nó trỏ đến một chương trình do chúng ta viết và chương trình này sẽ được thực hiện khi ngắt đó được gọi. Để làm điều đó, chúng ta chỉ cần tìm vị trí của vector ngắt tương ứng trong bảng và thay đổi giá trị của nó, điều này là thực hiện được, vì bảng vector ngắt đặt trong RAM (chúng được nạp vào mỗi khi khởi động hệ thống).

Chúng ta cũng sẽ dùng bảng vector ngắt này để tính địa chỉ bắt đầu của các chương trình con, sau đó vào chương trình DEBUG của DOS, rồi dùng lệnh U địa chỉ bắt đầu, để xem chương trình con xử lý ngắt viết bằng ngôn ngữ Assembly.

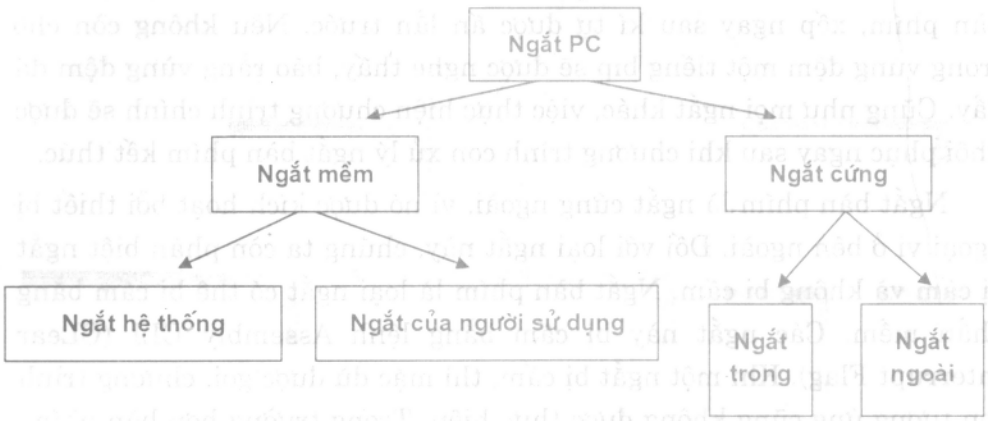
Cơ chế ngắt có ý nghĩa quan trọng ở chỗ, người lập trình có thể viết chương trình cơ sở (thí dụ khi xây dựng các chương trình điều khiển đối tượng chuẩn và phi chuẩn) như các chương trình con xử lý ngắt, và sau đó, một chương trình bất kỳ có thể gọi ngắt đó để sử dụng. Một chương trình có thể gọi chương trình con loại này mà không cần biết địa chỉ của nó, chỉ cần gọi ngắt với số lượng tương ứng, và bộ xử lý sẽ làm tất cả phần việc còn lại. Khả năng này được sử dụng trong hệ điều hành của máy tính PC, phần lớn các chức năng, các hàm của DOS được tổ chức như thư viện các chương trình con và chúng được gọi thông qua các ngắt.

Bảng 1.3. **Bảng vector ngắt của máy tính PC.**

Địa chỉ RAM (Hex)	*Thanh ghi CS:IP (giá trị vector ngắt)*	*Số thứ tự ngắt (số hiệu vector ngắt)*	*Chức năng của chương trình con phục vụ ngắt*
0000: 03FE	CS	255	Không dùng
0000:03FC	IP		
0000: 000E	CS	3	Thực hiện chương trình đến điểm dừng.
0000: 000C	IP		
0000: 000A	CS	2	Lỗi RAM (ngắt không che)
0000: 0008	IP		
0000: 0006	CS	1	Thực hiện chương trình theo từng lệnh
0000: 0004	IP		
0000: 0002	CS	0	Xử lý lỗi chia 0
0000: 0000	IP		

1.3.2. Phân loại ngắt sử dụng trong PC

Từ trước đến nay, chúng ta mới chỉ quan tâm đến các ngắt về mặt nguyên tắc, mà không phân biệt các loại ngắt. Sự khác nhau xuất phát từ chỗ, các ngắt không chỉ có ý nghĩa quan trọng đối với phần mềm (logic), mà cả với phần cứng (vật lý), do vậy, tồn tại những phương thức khác nhau để gọi ngắt. Hình 1.9 giới thiệu các loại ngắt khác nhau:



Hình 1.9. Các kiểu ngắt của PC

- *Ngắt mềm*

Ngắt mềm là ngắt được gọi bằng một lệnh ở trong chương trình ngôn ngữ ASEMBLY. Lệnh đó là INT, nó được sử dụng cùng với số hiệu ngắt. Thí dụ, lệnh gọi ngắt số 5 (in trang màn hình) được viết là INT 5. Các ngắt mềm cho phép gọi trực tiếp các chương trình con của hệ điều hành. Điều này không những thực hiện được ở mức ngôn ngữ Assembly, mà còn bằng cả ngôn ngữ bậc cao. Tuy nhiên phải hiểu rằng lệnh gọi ngắt từ ngôn ngữ bậc cao thì trình biên dịch sau đó sẽ được dịch ra thành lệnh Assembly INT, vì chỉ có lệnh này mới cho phép khởi động ngắt mềm. Do vậy, lệnh ngắt mềm bản chất là lệnh gọi CALL đặc biệt. Nó được gọi một cách chủ động bằng chương trình của người lập trình.

- *Ngắt cứng*

Khác với ngắt mềm, ngắt cứng không được khởi động bởi chương trình mà bởi các thành phần có trong phần cứng của PC (thí dụ như ổ

đĩa hay bàn phím...). Loại ngắt này là một cơ cấu đơn giản và hiệu quả để bộ xử lý phản ứng kịp thời với các sự kiện không đồng bộ xảy ra trong máy tính.

Điều này được minh họa thông qua thí dụ về cơ chế hoạt động của ngắt bàn phím. Mỗi khi ấn hay nhả một phím thì ngắt, gọi là ngắt bàn phím (ngắt số hiệu 9), sẽ được kích hoạt. Chương trình xử lý ngắt sẽ được khởi động bằng cách chuyển kí tự được ấn vào vùng bộ đệm của bàn phím, xếp ngay sau kí tự được ấn lần trước. Nếu không còn chỗ trong vùng đệm một tiếng bíp sẽ được nghe thấy, báo rằng vùng đệm đã đầy. Cũng như mọi ngắt khác, việc thực hiện chương trình chính sẽ được khôi phục ngay sau khi chương trình con xử lý ngắt bàn phím kết thúc.

Ngắt bàn phím là ngắt cứng ngoài, vì nó được kích hoạt bởi thiết bị ngoại vi ở bên ngoài. Đối với loại ngắt này, chúng ta còn phân biệt ngắt bị cấm và không bị cấm. Ngắt bàn phím là loại ngắt có thể bị cấm bằng phần mềm. Các ngắt này bị cấm bằng lệnh Assembly CLI (CLear Interrupt Flag). Khi một ngắt bị cấm, thì mặc dù được gọi, chương trình con tương ứng cũng không được thực hiện. Trong trường hợp bàn phím, điều đó có nghĩa là không ký tự nào được vào từ bàn phím nữa.

Để hủy bỏ chế độ cấm ngắt, người ta dùng lệnh STI (Set Interrupt Flag), nó cho phép các ngắt bị cấm trở lại hoạt động bình thường. Các ngắt không thể bị cấm sẽ luôn được thực hiện, kể cả khi ngắt này được gọi ngay sau lệnh CLI. Thí dụ, ngắt 2 là ngắt loại này. Nó báo hiệu có lỗi trong bộ nhớ và cho hiển thị một thông báo lên màn hình, cho biết một hay nhiều IC RAM bị hỏng.

Loại ngắt cuối cùng mà chúng ta cần có khái niệm là ngắt cứng nội bộ (trong). Những ngắt này không bị kích hoạt bởi thiết bị ngoài, mà bởi các chip IC bổ trợ nằm trên Main Board của PC như ngắt số hiệu 8 là ngắt thời gian, nó được gọi bởi bộ tạo thời gian (thường là TIMER LSI 8253) 18,2 lần/giây.

1.3.3. Danh sách các ngắt

Danh sách 256 ngắt (bảng 1.4) và chức năng của chúng được liệt kê trong bảng dưới đây. Bảng này cho phép tìm hiểu ý nghĩa các ngắt trong quá trình điều khiển và sử dụng PC.

Bảng 1.4. **Danh sách các ngắt của PC.**

STT	*Địa chỉ (Hex)*	*Chức năng của chương trình con ngắt*
00	000 - 003	CPU: Lỗi chia cho 0
01	004 - 007	CPU: Thực hiện từng lệnh
02	008 - 00B	CPU: Lỗi mạch RAM (ngắt NMI)
03	00C - 00F	CPU: Thực hiện đến điểm dừng
04	010 - 013	CPU: Tràn số
05	014 - 017	In trang màn hình (phím Print Screen)
06	018 - 01B	Lệnh lạ (chỉ với 80286)
07	01C - 01F	Để dành
08	020 - 023	IRQ0: Timer (được gọi 18,2 lần/giây)
09	024 - 027	IRQ1: Bàn phím
0a	028 - 02B	IRQ2: Bộ 8259 thứ hai (chỉ với AX)
0B	02C - 02F	IRQ3: Giao diện nối tiếp 2
0C	030 - 033	IRQ4: Giao diện nối tiếp 1
0D	034 - 037	IRQ5: Đĩa cứng
0E	038 - 03B	IRQ6: Đĩa mềm
0F	03C - 03F	IRQ7: Máy in
10	040 - 043	BOS: Màn hình
11	044 - 047	BOS: Xác định cấu hình
12	048 - 04B	BOS: Xác định kích thước bộ nhớ RAM
13	04C - 04F	BOS: Đĩa mềm, đĩa cứng
14	050 - 053	BOS: Truy nhập giao diện nối tiếp
15	054 - 057	BOS: Các phục vụ cassette hay mở rộng
16	058 - 05B	BOS: Kiểm tra bàn phím
17	05C - 05F	BIOS: Truy nhập máy in song song
18	060 - 063	Gọi BASIC trong ROM
19	064 - 067	BIOS:Khởi động hệ thống (ALT+CTRL+DEL)
1A	068 - 06B	BIOS: Thông báo thời gian
1B	06C - 06F	Phím Break (chứ không phải CTRL-C) bị ấn
1C	070 - 073	Được gọi sau INT 08.
1D	074 - 077	Địa chỉ của bảng tham số màn hình
1E	078 - 07B	Địa chỉ của bảng tham số đĩa mềm.
1F	07C - 07F	Địa chỉ của bảng phông các k/t mở rộng
20	080 - 083	DOS: Kết thúc chương trình

21	084 - 087	DOS: gọi các hàm của DOS
22	088 - 08B	Kết thúc chương trình
23	08C - 08F	Địa chỉ của thủ tục Ctrl-Break của DOS
24	090 - 093	Địa chỉ của thủ tục lỗi của DOS
25	094 - 097	DOS: đọc đĩa mềm, đĩa cứng
26	098 - 09B	DOS: ghi đĩa mềm, đĩa cứng
27	09C - 09F	DOS: Kết thúc chương trình và giữ nội trú
28	0A0 -	Dành cho các hàm không được DOS cung cấp
3F	- 0FF	tài liệu
40	100 - 103	BIOS: Các phục vụ đĩa mềm
41	104 - 107	Địa chỉ của bảng đĩa cứng 1
42	[illegible]	[illegible]
45	[illegible]	
[illegible]	118 - 11B	Địa chỉ của bảng đĩa cứng 2
47	11C	Người lập trình có thể [illegible]
49	- 127	ngắt này.
4A	128 - 12B	Hẹn giờ (chỉ cho AT)
4B	12C -	Người lập trình có thể tự xác định nghĩa ngắt này
67	- 19F	Không dùng
68	1A0 -	
6F	- 1BF	
70	1C0 - 1C3	IRQ08: Đồng hồ [illegible] (chỉ cho AT)
71	1C4 - 1C7	IRQ09: (Chỉ cho AT)
72	1C8 - 1CB	IRQ10: (Chỉ cho AT)
73	1CC - 1CF	IRQ11: (Chỉ cho AT)
74	1D0 - 1D3	IRQ12: (Chỉ cho AT)
75	1D4 - 1D7	IRQ13: 80287 NMI (Chỉ cho AT)
76	1D8 - 1DB	IRQ14: Đĩa cứng (Chỉ cho AT)
77	1DC - 1DF	IRQ15: (Chỉ cho AT)
78-	1CE -	
7F	- 1FE	Không dùng
80-	200 -	
F0	- 3CE	Được dùng trong bộ thông dịch BASIC
F1-	3C4 -	
FF	- 3CF	Không dùng

1.4. GỌI NGẮT TỪ CÁC NGÔN NGỮ LẬP TRÌNH

Đối với người lập trình bằng ngôn ngữ Assembly, việc gọi ngắt là không có vấn đề gì là khó khăn. Chỉ cần nạp giá trị cần thiết vào các thanh ghi tương ứng và sau đó thực hiện lệnh INT.

Đối với người lập trình ngôn ngữ bậc cao thì không có khả năng trực tiếp đó, nên phải dùng đến các hàm, các thủ tục hay các chương trình con để gọi ngắt.

- *Gọi ngắt trong Turbo Pascal*

Khi nhắc đến các ngôn ngữ lập trình hay các môi trường lập trình PC, ta không thể bỏ qua Turbo Pascal. Vì lý do này và cũng vì Turbo Pascal cho phép gọi ngắt tương đối dễ dàng, nên giáo trình chọn Turbo Pascal làm đại diện cho môi trường Pascal. Chúng ta sẽ sử dụng version 6.0, 7.0 là ngôn ngữ Turbo Pascal cho DOS hiện đang được dùng rộng rãi.

Để gọi một ngắt, Turbo Pascal cung cấp cho ta thủ tục INTR, nó được gọi với các tham số sau:

```
INTR (numberinterrupt: byte, regs: Registers);
```

Tham số numberinterrupt phải là một hằng nguyên cho biết số hiệu ngắt cần gọi. Ta có thể gọi bất kỳ ngắt nào trong khoảng từ 0 đến 225. Thủ tục MSDOS là một dạng đặc biệt của thủ tục INTR. Nó được gọi gần giống như INTR:

```
MSDOS (Regs: Registers);
```

Trong lệnh này số hiệu ngắt bị bỏ qua, vì lệnh được tổ chức theo cơ cấu tự động gọi ngắt số 21(hex). Ngắt 21H chính là một thư viện phong phú chứa các hàm chức năng của DOS Thông qua ngắt này, ta có thể gọi hầu hết các hàm chức năng của hệ điều hành.

Hai thủ tục trên nhận một tham số kiểu Registers. Tham số này chứa các giá trị dành cho các thanh ghi của bộ xử lý trước khi gọi các thủ tục INTR và MSDOS. Các giá trị này sẽ được nạp vào các thanh ghi tương ứng trước khi gọi ngắt.

Kiểu Registiers được định nghĩa như sau:

```
TYPE Registers = Record
case interger of
      0: (AX, BX, CX, DX, BP, SI, DI, DS, ES  Flags: word);
      1: (AL, AH, BL, BH, CL, CH, DL, DH: byte);
end;
```

Khi biến regs được định nghĩa bằng lệnh var Regs: Registers thì nó là một biến kiểu Registers gồm các thành phần sau: Regs.ax, Regs.bx, Regs.cx, Regs.ah,Regs.al,...

Để hiểu rõ cơ chế này ta minh họa qua thí dụ: giả sử giá trị 254 cần được nạp vào thanh ghi AX để gọi ngắt, ta làm như sau:

```
Regs.AX: = 254;
```

Phương pháp này có thể được áp dụng đối với tất cả các thanh ghi khác. Trước khi gọi ngắt, ta cần nạp các giá trị vào các thanh ghi có tham gia vào việc gọi ngắt. Nội dung của các thanh ghi khác ta không cần quan tâm. Sau khi gọi ngắt, nội dung các thanh ghi của bộ vi xử lý được copy vào biến Regs. Như vậy, người lập trình có thể kiểm tra các thông tin mà hàm được gọi trả lại thông qua biến này.

Trong chương trình không chỉ các thanh ghi đa năng mà cả các bit cờ trạng thái của thanh ghi cờ cũng được dùng để truyền thông tin cho chương trình gọi ngắt. Các hàm của DOS rất hay dùng cờ Carry cho việc này. Sau khi gọi một hàm của ngắt, cờ này sẽ bằng 1, nếu việc gọi hàm không thành công vì một lý do nào đó.

Để kiểm tra một cờ, ta có thể dùng các lệnh Pascal dưới đây. Chúng cho kết quả là TRUE (đúng) hay FALSE(sai), tùy theo cờ tương ứng là 1 hay 0:

```
Flag Carry: (Regs.flags and 1)
Flag Rero: (Regs.flags and 64)
Flag Sign: (Regs.flags and 128)
```

Nhiều khi cũng cần phải truyền cho ngắt địa chỉ của một biến (thường là địa chỉ một vùng đệm văn bản). Trong trường hợp này, ta sử

dụng các hàm ofs và Seg của Turbo Pascal, chúng cho địa chỉ offset và địa chỉ mảng của một biến.

```
ofs (Tên_Biến)
Seg (Tên_Biến)
```

Dưới đây là thí dụ sử dụng cách dùng lệnh INTR của Pascal để kích hoạt các chương trình con ngắt của DOS hay của BIOS. Chương trình này minh họa cách lấy thông tin về lịch hệ thống dạng ngày - tháng - năm và hiển thị lên màn hình kết quả. Nếu người lập trình phải tự tìm lấy các số liệu này thì rất khó khăn do phải biết chính xác nơi nào trong máy tính chứa các thông số đó. Trong khi đó một tài nguyên hệ thống dưới dạng một chương trình con phục vụ ngắt - hàm 2Ah của vector ngắt 21h lại cho kết quả về thông số ngày - tháng - năm vào các thanh ghi tương ứng DL - DH - CX của bộ vi xử lý. Ta chỉ cần tổ chức các biến của chương trình để nhận lại các kết quả đó từ các thanh ghi trên là được. Bằng cách đặt vấn đề như thế, chương trình minh họa có dạng:

```
program Lich_HeThong; {lấy lịch có trong đồng hồ PC}
Uses DOS;
var
      Regs: Registers;
      date, year, month, day:string;
                              {các biến lịch hệ thống}
begin
      Regs.AH:=$2a;           {hàm 2ah của ngắt 21h}
      With regs do
      begin
           INTR ($21,regs);   {khởi động ngắt}
           Str(CX,year);
           Str(DH,month);
           Str(DL,day);
      end;
      date:=day+month+year;
      Writeln(Today is,date);
end.
```

- *Gọi ngắt từ ASSEMBLY*

Khác với những người lập trình trên ngôn ngữ bậc cao, các lập trình viên của Assembly không phải cần đến các hàm, các thủ tục phức tạp để gọi ngắt. Đối với ngôn ngữ Assembly chỉ cần nạp các tham số của ngắt bằng lệnh MOV vào các thanh ghi dành cho việc này, rồi sau đó gọi ngắt bằng lệnh INT.

Một số ngắt hoặc một số hàm của các ngắt được gọi rất thường xuyên trong các chương trình điều khiển hệ thống. Hàm 09h của ngắt DOS 21h hiển thị một dãy ký tự ra màn hình là một thí dụ. Nó được gọi với số hiệu của hàm 09h đặt ở thanh ghi AH, với địa chỉ Offset của dãy ký tự đặt ở thanh ghi DX. Trong một chương trình Assembler, việc gọi hàm này được viết như sau:

```
MOV ah,09h;                 Nap thứ tự hàm 09h
MOV dx, offset text;        Nap địa chỉ offset của text
INT 21h;                    Goi ngắt 21h của DOS
```

1.5. KỸ THUẬT ĐÁNH TRÁO NGẮT VÀ CHẶN BẮT NGẮT

Khi nói tới bảng vector ngắt ta đã nói rằng đây là tài nguyên phần mềm của máy tính cần phải tận dụng triệt để, song trong nhiều trường hợp một tài nguyên nào đó của máy tính không thỏa mãn các yêu cầu chức năng của chương trình mà ta phải xây dựng, thì trong trường hợp đó ta có thể giải quyết theo hai giải pháp:

- Viết một chương trình con có chức năng cần thiết, đáp ứng mọi thao tác theo yêu cầu của phần mềm điều khiển rồi đánh tráo vào vector ngắt tương ứng. Điều này có nghĩa là khi ngắt tương ứng bị kích hoạt máy tính sẽ gọi chương trình của ta ra hoạt động thay vì chương trình con phục vụ ngắt cũ của máy tính và do đó các chức năng được ấn định trong chương trình của ta sẽ được thực hiện trọn vẹn.

- Viết một chương trình con có chức năng cần thiết, bổ xung các chức năng còn thiếu mà chương trình con phục vụ ngắt của máy tính không có rồi đánh tráo vào vector ngắt tương ứng. Khi cần tới chương trình con phục vụ ngắt cũ của máy tính, chương trình con đã đánh tráo sẽ gọi lại chính chương trình con phục vụ ngắt cũ này.

Để thực hiện kỹ thuật đánh tráo ngắt, về bản chất là thực hiện việc đánh tráo nội dung 4 byte của của vector ngắt tương ứng (trong bảng vector ngắt). Nội dung 4 byte này là địa chỉ vào của chương trình được ngắt gọi ra thực hiện. Vậy là chỉ cần biết địa chỉ vào của chương trình của người lập trình cần đánh tráo và gán vào nội dung 4 byte của vector ngắt hiện hành là được. Tuy nhiên để đề phòng trường hợp cần khôi phục lại ngắt cũ cần lưu trữ nội dung 4 byte của vector ngắt cũ trước khi gán giá trị mới cho nó.

Máy tính có một dịch vụ làm được điều này đó là hàm 35h của ngắt 21h dùng để lưu giá trị vector ngắt cũ (nội dung 4 byte của vector ngắt) v ới các yêu cầu đầu vào như sau:

```
Số hiệu hàm phải gán vào thanh ghi AH.
Số hiệu ngắt phải gán vào thanh ghi AL.
Giá trị ngắt được trả về cho cặp thanh ghi ES: BX.
```

và hàm25h của ngắt 21h dùng để tráo giá trị vector ngắt v ới các yêu cầu đầu vào như sau:

```
Số hiệu hàm phải gán vào thanh ghi AH.
Số hiệu ngắt phải gán vào thanh ghi AL.
Địa chỉ Offset của chương trình cần đánh tráo phải chứa
trong DX.
```

1.5.1. Chặn bắt ngắt bằng ngôn ngữ bậc thấp ASSEMBLY

Để minh họa kỹ thuật trên bằng ngôn ngữ ASSEMBLY ta sẽ thực hiện trên ngắt 13H là ngắt phục vụ các thao tác truy nhập ổ đía (đọc/ ghi đĩa). Chức năng truy nhập ổ đía là chức năng điều khiển rất phức tạp do phải điều khiển đồng bộ hoạt động của các cơ cấu khác nhau như

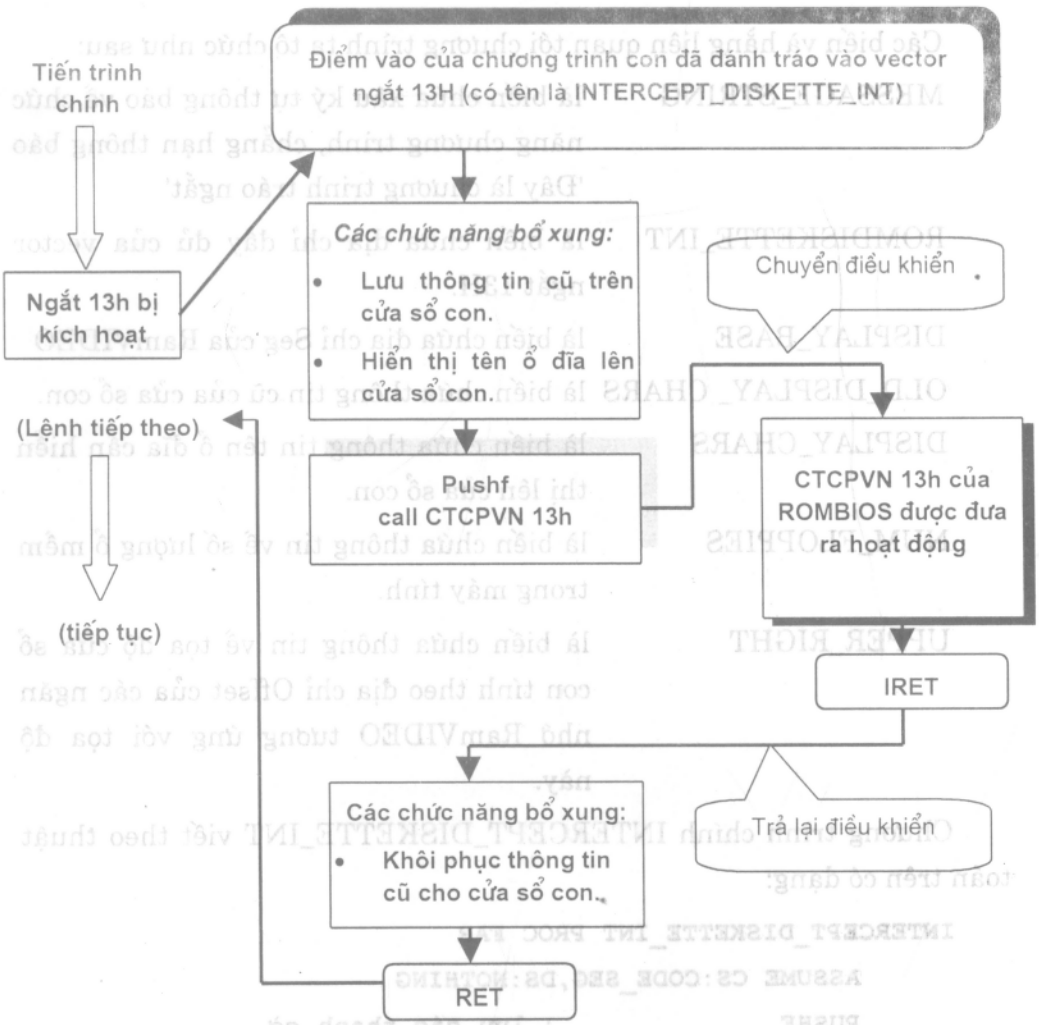
cơ cấu chuyển dữ liệu từ đĩa vào Ram và ngược lại, cơ cấu cơ-điện cho đầu từ đọc/ ghi... Do vậy nếu phải tráo ngắt này ta chỉ nên bổ xung các chức năng cần thiết, còn khi cần thao tác với ổ đĩa ta lại gọi lại chính chương trình con phục vụ ngắt 13h của hệ thống.

Chức năng bổ xung là chức năng hiển thị tên ổ đĩa lên một cửa sổ con của màn hình. Giả sử máy tính chạy ở chế độ TEXT độ phân giải 80 cột X 25 dòng thì cửa sổ con chỉ cần 2 ký tự là đủ để tạo thành dạng hiển thị tên ổ đĩa chuẩn (như A:). Ký tự thứ nhất là tên bằng chữ in hoa, ký tự thứ hai là dấu ':'. Cửa ssổ này ta để ở góc trên bên phải màn hình cho tiện theo dõi. Chương trình mà ta sẽ tráo vào ngắt 13h đặt tên là INTERCEPT_DISKETTE_INT.

INTERCEPT_DISKETTE_INT in ra tên ổ đĩa bằng cách theo dõi các dịch vụ đọc/ghi đĩa của ROMBIOS - tương ứng với dịch vụ 13h. Sau khi tráo được vectơ ngắt, mỗi khi có ngắt 13H kích hoạt nó sẽ trỏ đến thủ tục của chúng ta thì chương trình của ta sẽ nắm được quyền điều khiển ngắt INT 13h của BIOS. Để gọi lại ngắt 13h, không thể dùng lệnh INT 13h được vì chương trình sẽ bị quẩn. Cũng không dùng lệnh CALL được vì chương trình con được gọi là trong chương trình chứa trình con phục vụ ngắt (có lệnh IRET ở cuối). Vậy là phải tạo giả lệnh INT 13h bằng tổ hợp lệnh CALL bình thường và lệnh PUSHF vì lệnh INT không những lưu giữ con trỏ chương trình (CS:IP)mà còn cất giữ cả thanh ghi cờ F vào STACK và lệnh IRET ở cuối chương trình con của nó không những khôi phục lại giá trị cho con trỏ chương trình mà cả nội dung thanh ghi cờ. Do đó tổ hợp lệnh tạo giả lệnh INT có cấu trúc như sau:

```
pushF
call Dia chỉ của CTCPVN 13H
```

Việc đánh tráo ngắt sẽ thực hiện bằng thủ tục riêng có tên là INIT_VECTOR, còn chương trình chính INTERCEPT_DISKETTE_INT sẽ được xây dựng theo thuật toán mô tả tren hình 1.10.



Hình 1.10. Cơ chế chặn, bắt ngắt 13h của chương trình INTERCEPT_DISKETTE_INT

Trong sơ đồ 1.10, CTCPVN (chương trình con phục vụ ngắt) 13h là địa chỉ đầy đủ của ngắt 13h. Ta đặt Rom_diskette_int là biến bộ nhớ của chương trình có kích thước 4 byte để lưu trữ địa chỉ đầy đủ Seg:Off của ngắt 13h. Khi dịch vụ này kết thúc công việc của mình, quyền điều khiển lại thuộc về chương trình của ta. Bằng cách đó có thể bổ xung mọi điều cần thiết cho chương trình của ta kể cả trước khi gọi ngắt 13h cũng như sau khi gọi n

Các biến và hằng liên quan tới chương trình ta tổ chức như sau:

MESSAGE_STRING	là biến chứa xâu ký tự thông báo về chức năng chương trình, chẳng hạn thông báo 'Đây là chương trình tráo ngắt'
ROMDISKETTE_INT	là biến chứa địa chỉ đầy đủ của vector ngắt 13H.
DISPLAY_BASE	là biến chứa địa chỉ Seg của RamVIDEO
OLD_DISPLAY_ CHARS	là biến chứa thông tin cũ của cửa sổ con.
DISPLAY_CHARS	là biến chứa thông tin tên ổ đĩa cần hiển thị lên cửa sổ con.
NUM_FLOPPIES	là biến chứa thông tin về số lượng ổ mềm trong máy tính.
UPPER_RIGHT	là biến chứa thông tin về tọa độ cửa sổ con tính theo địa chỉ Offset của các ngăn nhớ RamVIDEO tương ứng với tọa độ này.

Chương trình chính INTERCEPT_DISKETTE_INT viết theo thuật toán trên có dạng:

```
INTERCEPT_DISKETTE_INT PROC FAR
      ASSUME CS:CODE_SEG,DS:NOTHING
      PUSHF                  ; lưu các thanh cờ
      PUSH     AX            ; lưu các thanh ghi khác
      PUSH     SI
      PUSH     DI
      PUSH     DS
      PUSH     ES
      CALL GET DISPLAY_BASE  ; tính địa chỉ mảng VIDEORAM
      CALL SAVE_SCREEN       ; lưu 2 ký tự góc phía trên bên
                             ;phải màn hình
      CALL  DISPLAY_DRIVER_LETTER ;in tên ổ đĩa
      POP ES
```

```
        POP DS
        POP DI
        POP SI
        POP AX
        POPF            ;khôi phục TG cờ
        PUSHF           ;tạo giả lệnh INT
        CALL ROM_DISKETTE_INT
                        ; gọi dịch vụ chương trình con
                        ;phục vụ  ngắt 13H của ROM_BIOS
        PUSHF
        PUSH     AX
        PUSH     SI
        PUSH     DI
        PUSH     DS
        PUSH     ES
        LEA SI,OLD_DISPLAY_CHARS     ; trỏ đến 2 ký tự cũ
        CALL WRITE_TO_SCREEN         ;in lại các ký tự này
        POP ES
        POP DS
        POP DI
        POP SI
        POP AX
        POPF
        RET 2               ; sau khi thực hiện xong SP
                            ;  ← SP+2 để bỏ qua POPF
INTERCEPT_DISKETTE_INT ENDP
```

Như vậy, chương trình chính là một loạt lệnh gọi tới các chương trình con, mỗi cái có một chức năng riêng:

Chương trình con GET DISPLAY_BASE lấy địa chỉ mảng của VIDEORAM nơi có các ngăn nhớ chứa thông tin cửa sổ con. Địa chỉ này lưu vào biến DISPLAY_BASE.

Chương trình con SAVE_SCREEN có chức năng lưu 2 ký tự cửa sổ con (góc phía trên bên phải màn hình) vào biến OLD_DISPLAY_ CHARS để sau này còn khôi phục lại.

Chương trình con DISPLAY_DRIVER_LETTER thực hiện in tên ổ đĩa lên cửa sổ con từ nội dung biến DISPLAY_CHARS.

Chương trình con WRITE_TO_SCREEN có chức năng lkhôi phục 2 ký tự cửa sổ con từ biến OLD_DISPLAY_ CHARS mà thủ tục SAVE_SCREEN đã lưu vào trước đó.

Riêng việc lấy thông tin về số lượng ổ đĩa mềm cần một thủ tục GET_NUM_FLOPPIES để gán kết quả vào biến NUM_FLOPPIES và thủ tục này phải được gọi trước thủ tục DISPLAY_DRIVER_LETTER để thủ tục DISPLAY_DRIVER_LETTER có thể in đúng tên ổ nếu nó là ổ cứng. Lưu ý rằng thủ tục này chỉ cần chạy một lần.

Chương trình con cuối cùng được xét tới lại là chương trình con phải chạy đầu tiên để thực hiện việc đánh tráo ngắt 13H. Ta đặt tên cho nó là thủ tục INIT_VECTORS. Thủ tục này sẽ:

- Thông báo chức năng lên màn hình.
- Đánh tráo ngắt 13H.
- Đánh dấu hiệu thường trú cho chương trình này.

Vì chương trình của chúng ta có chức năng giám sát mọi thao tác truy nhập ổ đĩa nên nó phải thường xuyên có mặt trong RAM và ở trạng thái hoạt động. Muốn như vậy vùng RAM mà hệ điều hành cấp phát cho nó phải được bảo vệ. Muốn được như thế chương trình phải được đánh dấu thường trú. Chương trình con phục vụ ngắt 27H cho phép thực hiện chức năng này với yêu cầu là địa chỉ điểm cuối của chương trình phải được gán vào cho DX. Nhờ đó mà ta loại bỏ được phần chương trình không cần thiết cho hoạt động sau này của chương trình. Cụ thể ta sẽ loại bỏ 2 thủ tục là INIT_VECTORS và GET_NUM_FLOPPIES. Vì vậy, 2 thủ tục này được viết ở cuối chương trình.

Sau toàn bộ các phân tích trên ta có chương trình đầy đủ như sau.

```
CODE_SEG   SEGMENT          ; mở mảng mã lệnh
      ASSUME CS:CODE_SEG,DS:CODE_SEG
      ORG 100H
      BEGIN: JMP INIT_VECTORS
                            ; nhảy ngay tới thủ tục khởi tạo
;-------------------------------------------------
;Phần này sẽ khai báo các hằng và các biến cần thiết
;cho chương trình.
;-------------------------------------------------
MESSAGE_STRING DB 'Đây là chương trình tráo ngắt'
                                    DB 0DH,0AH,'$'
      ROMDISKETTE_INT DD?
      DISPLAY_BASE            DW?
      OLD_DISPLAY_ CHARS      DB 4 DUP(?)
      DISPLAY_CHARS           DB 'A',70H.':',70H
      NUM_FLOPPIES            DB?
      UPPER_RIGHT             EQU (80-2)*2
;-------------------------------------------------
;Thủ tục này là thủ tục ngắt do người sử dụng viết:
;     1. in tên ổ đĩa ở phía trên bên phải màn hình;
;     2. gọi ngắt trong BIOS
;     3. phục hồi ký tự cũ của màn hình;
;-------------------------------------------------
INTERCEPT_DISKETTE_INT PROC FAR
      ASSUME CS:CODE_SEG, DS:NOTHING
      PUSHF               ; lưu thanh ghi cờ vào Stack
      PUSH     AX         ; lưu các thanh ghi khác
      PUSH     SI
      PUSH     DI
      PUSH     DS
      PUSH     ES
      CALL GET DISPLAY_BASE
                          ; tính địa chỉ mảng VIDEORAM
```

```
        CALL SAVE_SCREEN      ; lưu 2 ký tự góc phía trên bên
                              ;phải màn hình
        CALL  DISPLAY_DRIVER_LETTER ;in tên ổ đĩa
        POP ES
        POP DS
        POP DI
        POP SI
        POP AX
        POPF                  ;khôi phục thanh ghi cờ
        PUSHF                 ;tạo giả lệnh INT
        CALL ROM_DISKETTE_INT
                              ; gọi dịch vụ chương trình con
                              ;phục vụ  ngắt 13H của ROM_BIOS
        PUSHF
        PUSH      AX
        PUSH      SI
        PUSH      DI
        PUSH      DS
        PUSH      ES
        LEA SI,OLD_DISPLAY_CHARS      ; trỏ đến 2 ký tự cũ
        CALL WRITE_TO_SCREEN          ;in lại các ký tự này
        POP ES
        POP DS
        POP DI
        POP SI
        POP AX
        POPF
        RET 2           ; sau khi thực hiện xong SP
                        ; ← SP+2 để bỏ qua POPF
INTERCEPT_DISKETTE_INT ENDP
;-------------------------------------------------
;Thủ tục này tính địa chỉ SEG của VIDEORAM
;-------------------------------------------------
```

```
GET_DISPLAY_BASE  PROC NEAR
      ASSUME CS:CODE_SEG,DS:NOTHING
      INT 11H         ;hàm 11h lấy cấu hình hệ thống
      AND AX,30H      ;d5d4 cung cấp thông tin về màn hình
      CMP AX,30H      ;d5d4=11---> màn hình MONO
      MOV AX,0B800H ;là địa chỉ mảng của RAMVIDEO màu
      JNE DONE_GET_BASE
      MOV AX,0B000H  ;là địa chỉ mảng của RAMVIDEO MONO
      DONE_GET_BASE:
          MOV DISPLAY_BASE, AX    ;đưa vào biến
          RET
GET_DISPLAY_BASE  ENDP
;-----------------------------------------------------
; Thủ tục này lưu 2 ký tự góc trên bên phải của màn hình
;-----------------------------------------------------
SAVE_SCREEN PROC NEAR
      ASSUME CS:CODE_SEG,DS:NOTHING
      MOV SI,UPPER_RIGHT
                  ; SI trỏ tới ký tự trên cửa sổ con
      LEA DI,OLD_DISPLAY_CHARS ; DI trỏ tới biến lưu
      MOV AX, DISPLAY_BASE     ;gán địa chỉ mảngVIDEORAM
      MOV DS, AX      ; Tạo trỏ DS:S I--> Thông tin trên
                      ; cửa sổ con
      MOV AX, CS
      MOV ES, AX      ; Tạo trỏ ES:D I--> biến lưu
      CLD       ;cờ hướng DF=0 để làm việc với Ram thường
      MOVSW     ;chuyển 2 ký tự (mỗi k/t 2 byte: byte
                ;         k/t+thuộc tính)
      MOVSW
      RET
SAVE_SCREEN       ENDP
;-----------------------------------------------------
;Thủ tục này in tên ổ đĩa
;-----------------------------------------------------
```

```
DISPLAY_DRIVER_LETTER  PROC NEAR
      ASSUME CS:CODE_SEG,DS:NOTHING
      MOV AL,DL           ;lấy số hiệu ổ đĩa
      CMP AL, 80H         ;có phải là ổ đĩa cứng?
      JB DISPLAY_LETTER   ;nếu không, lấy tiếp
      SUB AL.80H          ;đổi thành số hiệu ổ đĩa cứng
      ADD AL,NUM_FLOPPIES;+ với số lượng ổ mềm
      DISPLAY_LETTER:
           ADD AL,'A'     ;đổi thành ổ tương ứng
           LEA SI,DISPLAY_CHARS
                          ;trỏ tới biến chứa tên ổ
           MOV CS:[SI],AL;lưu k/t này
           CALL WRITE_TO_SCREEN
           RET
DISPLAY_DRIVER_LETTER  ENDP
;-------------------------------------------------
;Thủ tục này in 2 k/t vào góc trên bên phải màn hình
;-------------------------------------------------
WRITE_TO_SCREEN PROC NEAR
      ASSUME CS:CODE_SEG,DS:NOTHING
      MOV DI,UPPER_RIGHT ;DI trỏ tới tọa độ cửa sổ con
      MOV AX,DISPLAY_BASE;gán d/c mảng VIDEORAM
      MOV ES,AX           ;ES: DI trỏ tới tới cửa sổ con
      MOV AX,CS
      MOV DS,AX           ;DS: SI trỏ tới biến chứa tên
      CLD
      MOVSW               ;chuyển 2 ký tự (mỗi k/t 2 byte:
                          ;byte k/t+thuộc tính)
      MOVSW
      RET
WRITE_TO_SCREEN   ENDP
;-------------------------------------------------
;Thủ tục này khởi tạo chức năng tráo ngắt và thường
trú chương trình
;-------------------------------------------------
```

```
INIT_VECTORS PROC NEAR
      ASSUME CS:CODE_SEG,DS: CODE_SEG
      LEA DX, MESSAGE_STRING  ;in thông báo
      MOV AH,9                ;bằng hàm N9 của ngắt 21H
      INT 21H
      CALL GET_NUM_FLOPPIES   ;có mấy ổ đĩa mềm?
      MOV AH,35H     ;nhận giá trị của vectơ 13h-->es:bx
      MOV AL,13H
      INT 21H
      MOV WORD PTR ROM_DISKETTE_INT,BX
                              ; chuyển vào biến
      MOV WORD PTR ROM_DISKETTE_INT[2],ES
      MOV AH,25H          ; tráo vectơ ngắt bằng cách
      MOV AL,13H          ; cho ngắt 13h trỏ tới ds:dx là
                          ; thủ tục của ta
      MOV DX,OFFSET INTERCEPT_DISKETTE_INT
      INT 21H;
MOV DX,OFFSET INIT_VECTOR
                          ; loại bỏ khỏi RAM từ phần này
INT 27H                   ; kết thúc và ở lại thường trú
                          ; phần còn lại
INIT_VECTORS   ENDP
;-------------------------------------------------
;Thủ tục kiểm tra xem có bao nhiêu ổ đĩa mềm
;-------------------------------------------------
GET_NUM_FLOPPIES PROC NEAR
      ASSUME CS:CODE_SEG,DS: CODE_SEG
      INT 11H         ;lấy cấu hình thiết bị
      MOV CL,6        ;d7d6 chứa thông tin ổ mềm
      SHR AX,CL
      AND AL,3        ;bỏ qua các cờ khác
      INC AL          ;trả lại giá trị 0 nếu có một ổ đĩa mềm
      CMP AL,1        ;có phải có có một ổ đĩa mềm?
```

```
        JA DONE_FLOPPIES          ; không,vậy số đúng
        MOV AL,2                  ; có, vậy có 2 ổ lôgic
DONE_FLOPPIES:
        MOV NUM_FLOPPIES,AL       ; lưu lại để đấy
        RET
GET_NUM_FLOPPIES ENDP
CODE_SEG ENDS
END BEGIN
```

Tóm lại, thủ tục INTERCEPT_DISKETTE_INT là thủ tục thay thế cho chương trình con phục vụ ngắt 13h và được khai báo như là thủ tục xa PROC FAR. Đầu tiên, nó gọi thủ tục con GET DISPLAY_BASE để lấy địa chỉ mảng của vùng VIDEORAM là nơi chứa thông tin cho màn hình hiển thị. Kế đến là gọi thủ tục con SAVE_SCREEN để lưu 2 ký tự góc phải màn hình vào biến OLD_DISPLAY_ CHARS rồi gọi thủ tục con DISPLAY_DRIVER_LETTER để in tên ổ đĩa chứa trong biến DISPLAY_CHARS. Như vậy tên ổ đĩa đang truy cập đã hiển thị ở phía góc trên phía bên phải màn hình đúng theo yêu cầu. Đến đây INTERCEPT_DISKETTE_INT gọi lại chương trình con phục vụ ngắt 13h bằng lệnh giả INT 13h. Sau khi chức năng của int 13h thực hiện xong, INTERCEPT_DISKETTE_INT khôi phục lại thông tin ở góc trên phía phải bằng cách gọi thủ tục con tương ứng WRITE_TO_SCREEN để in lại các ký tự cũ chứa trong biến OLD_DISPLAY_CHARS.

Thủ tục INIT_VECTORS và GET_NUM_FLOPPIES cho thấy cách tạo phương pháp chặn vectơ ngắt 13h và giữ lại một chương trình thường trú trong bộ nhớ, sau khi đã trả quyền điều khiển cho DOS.

Lưu ý rằng cả hai thủ tục khởi tạo đều đặt cuối chương trình vì chúng chỉ dùng một lần khi nó được nạp vào bộ nhớ, do đó phần này sẽ loại bỏ khỏi chương trình thường trú để giảm kích thước của mình. Lệnh INT 27h sẽ giải phóng vùng mã chương trình bắt đầu từ địa chỉ ofsset chứa trong DX. Do đó khi cho DX trỏ tới INIT_VECTORS là chúng ta muốn DOS giữ lai thường trú DISKLITE trừ hai thủ tục cuối cùng là tục INIT_VECTORS và GET_NUM_FLOPPIES.

Phần lớn các chương trình thường trú được viết bằng ngôn ngữ assembler để tạo khả năng tối đa truy cập ROM_BIOS và bộ nhớ trung tâm, vì kích thước của chúng rất nhỏ. Chúng ta nghiên cứu kỹ thuật này thông qua một chương trình thí dụ, chương trình kiểm soát sự hoạt động của ổ đĩa có trong máy tính PC - chương trình DISKLITE. Chương trình này sẽ in lên góc trên bên phải màn hình tên ổ đĩa đang thao tác, thí dụ A:, hay B: hay C:...Như vậy ta không cần nhìn đèn ổ đĩa mà chỉ cần nhìn lên góc màn hình là đã biết ổ đĩa nào đang hoạt động.

Các chương trình thường trú phải làm việc thật chặt chẽ với ROMBIOS để thừa kế các chức năng có sẵn trong nó cũng chính là để chương trình thường trú thật ngắn, gọn.

1.5.2. Chặn, bắt ngắt từ ngôn ngữ bậc cao

Bằng ngôn ngữ bậc cao, như TURBO PASCAL chẳng hạn, ta có thể chặn và bắt ngắt trỏ tới các thủ tục của người lập trình thay vì trỏ tới bảng vectơ ngắt của DOS. Các địa chỉ của ngắt trong bảng vectơ ngắt thay đổi bằng hai cách: thay trực tiếp bảng vectơ ngắt hoặc vẫn sử dụng các dịch vụ của DOS theo mục đích của người lập trình.

Để thay đổi bảng vectơ ngắt, ta tổ chức một mảng có cấu trúc ứng với vùng nhớ RAM của bảng vectơ ngắt:

```
var
AsyncVector:Pointer;
InterruptVector:    array [0..$ff] of pointer
                    Absolute $0000:$0000;
```

Pointer là kiểu dữ liệu 4 byte đủ để chứa cả địa chỉ segment và địa chỉ offset của chương trình con phục vụ ngắt. Biến InterruptVector là mảng 256 địa chỉ bắt đầu từ địa chỉ tuyệt đối 0000:0000h - địa chỉ đầu tiên của vùng nhớ RAM của PC. Để thay đổi địa chỉ nào đó của bảng vectơ ngắt ta sử dụng các câuu lệnh sau:

```
AsyncVector:=InterruptVector[$0c];
InterruptVector[$0c]:=@AsyncInt;
```

Câu lệnh đầu dùng để cất giữ địa chỉ của vectơ ngắt số hiệu 0ch vào biến AsyncVector. Câu lệnh thứ hai dùng để gán cho vectơ của ngắt số hiệu 0ch địa chỉ của thủ tục TURBO PASCAL có tên là AsyncInt.

Như vậy, nếu bây giờ ngắt 0ch được kích hoạt thì thủ tục asyncInt sẽ được gọi.

Khi cần khôi phục lại giá trị địa chỉ cũ của vectơ ngắt thì TURBO PASCAL sử dụng câu lệnh:

```
InterruptVector[$0c]:= AsyncVector;
```

Phương pháp thứ hai là sử dụng dịch vụ sẵn của DOS: dịch vụ 35h của DOS cung cấp địa chỉ của vectơ ngắt, còn dịch vụ 25h cho phép thay giá trị địa chỉ mới cho vectơ ngắt. Sử dụng 2 câu lệnh sau để thực hiện đánh tráo ngắt:

```
GetIntVec($0c,Asyncvector);
SetIntVec($0c,@AsyncInt);
```

Câu lệnh đầu dùng để cất giữ địa chỉ cũ của vectơ ngắt 0ch vào biến Asyncvector, còn câu lệnh sau dùng để gán địa chỉ của thủ tục TURBO PASCAL cho vectơ ngắt 0ch.

CHƯƠNG 2
CƠ CHẾ QUẢN LÝ CỦA HỆ ĐIỀU HÀNH ĐƠN NHIỆM MSDOS

Hệ điều hành đơn nhiệm DOS (Disk Operating System) được cài đặt trong IBM PC ngay từ những thế hệ máy tính PC đầu tiên. Các version sau được cải tiến, nâng cấp đáng kể, song nguyên lý chung vẫn không thay đổi. Vì vậy vấn đề đặt ra để nghiên cứu trong mục này vẫn mang tính tổng quát cao. Về bản chất, DOS là một giao diện đơn nhiệm mà từ đó ta kích hoạt các chương trình cần chạy hoặc khởi động các lệnh có trong DOS. Song chúng ta sẽ không dừng lại ở giao diện của DOS, mà sẽ nghiên cứu, tìm hiểu tổ chức bên trong của hệ điều hành, nhằm khai thác triệt để những tài nguyên có sẵn trong máy tính để phục vụ cho các chương trình hệ thống sau này. Trong đó có một lượng rất lớn các dịch vụ, các chương trình (các hàm) mà không những cần cho DOS để thực hiện công việc của nó, và người lập trình hệ thống cũng có thể sử dụng chúng.

2.1. TỔ CHỨC CỦA DOS

Hệ điều hành DOS là phần mềm điều khiển gồm nhiều thành phần, mỗi thành phần thực hiện một nhiệm vụ xác định trong một hệ thống. Ba thành phần chính của DOS là DOS BIOS, hạt nhân của DOS và bộ xử lý lệnh. Mỗi thành phần này được đặt trong một file riêng biệt.

Toàn bộ chức năng của DOS-BIOS đặt trong file IBMBIO.COM, nó có thuộc tính ẩn (Hidden) và hệ thống (System). Do vậy, không thể hiển

thị chúng bằng lệnh DIR. File IBMBIO.COM gồm các chương trình điều khiển thiết bị (driver) cho các thiết bị sau: CON (Bàn phím và màn hình), PRN (Máy in chuẩn), AUX (Giao diện nối tiếp), CLOCK (Đồng hồ), ổ đĩa A, B, C...

Khi DOS muốn truy nhập vào một thiết bị nào đó, nó gọi driver chứa trong file IBMBIOS.COM, driver này lại sử dụng các chương trình con của BIOS có trong ROMBIOS. Do vậy, DOS - BIOS là thành phần có quan hệ trực tiếp nhất với phần cứng và phải thay đổi tùy theo cấu trúc (model) của máy tính.

Hạt nhân của DOS nằm trong file IBMDOS.COM. File IBMDOS.COM chứa các chương trình con cho phép truy nhập tới cấu trúc file, sử dụng RAM và điều khiển việc xuất/nhập ký tự. Các chương trình con này được xây dựng không phụ thuộc vào phần cứng của máy tính và nó dựa vào các driver của DOS - BIOS để truy nhập các thiết bị như bàn phím, màn hình và ổ đĩa. IBMDOS.COM có thể được sử dụng cho các PC khác loại. Các chương trình của người sử dụng có thể truy nhập các hàm (chương trình con) này như là truy nhập các hàm của BIOS trong ROM BIOS. Mỗi hàm có thể được gọi bằng cách gọi một ngắt mềm, dùng các thanh ghi của bộ vi xử lý để truyền số hiệu của hàm và các tham số cần thiết khác.

Khác với hai file trên, bộ xử lý lệnh nằm trong một file nhìn thấy được có tên là COMMAND.COM. Chính file này hiển thị lên màn hình dấu nhắc A> hoặc C>, nó nhận các lệnh của người sử dụng và thực hiện chúng. Lưu ý là COMMAND.COM chỉ là một chương trình đặc biệt chạy dưới sự giám sát của DOS chứ không phải nó là hệ điều hành hay đại diện cho cả hệ điều hành.

Bộ xử lý lệnh COMMAND.COM còn gọi là "SHELL", về phần mình lại được cấu trúc từ 3 module: một phần thường trú, một phần tạm trú và một chương trình con cài đặt.

Phần thường trú COMMAND.COM, có nghĩa là phần nằm thường trực trong bộ nhớ của máy tính gồm các chương trình con khác nhau-

được kích hoạt bằng cơ chế ngắt, chúng cho phép phản ứng với các sự kiện khác nhau. Các sự kiện này có thể là phím Ctrl-C hay Ctrl-Break được người sử dụng ấn, hoặc một số lỗi sinh ra khi thực hiện thao tác với các thiết bị ngoại vi như đĩa cứng, các ổ đĩa mềm hay máy in.

Phần tạm trú của COMMAND.COM đưa ra dấu nhắc (A>). Do đây là phần tạm trú nên không gian nhớ RAM mà nó chiếm không được bảo vệ. Mỗi khi chương trình kết thúc, điều khiển được trả về cho phần thường trú của bộ xử lý lệnh, bằng cách tính tổng bộ nhớ bộ xử lý lệnh xác định xem phần tạm trú có bị xóa bởi chương trình ứng dụng hay không. Nếu phần này bị xoá, phần thường trú sẽ nạp lại phần tạm trú.

Module cài đặt của COMMAND.COM được nạp vào bộ nhớ khi hệ thống khởi động và chức năng của nó là khởi tạo hệ thống. Mỗi khi công việc của module này kết thúc, nó không được dùng nữa và phần RAM mà nó chiếm được giải phóng.

Các lệnh mà phần tạm trú của bộ xử lý lệnh COMMAND.COM có thể nhận được nằm ở một trong 3 nhóm: nhóm lệnh nội trú, nhóm lệnh ngoại trú và nhóm xử lý file batch (gói file).

Nhóm thứ nhất là các lệnh mà mã của nó nằm trong phần tạm trú của bộ xử lý lệnh. Nó gồm các lệnh COPY, REN, và DIR... Nhóm thứ hai là các lệnh ngoại trú được đặt trong đĩa mềm hay đĩa cứng và sẽ được nạp vào bộ nhớ trong để thực hiện mỗi khi chúng được gọi như các lệnh FORMAT và CHKDISK. Sau khi thực hiện xong các lệnh này, phần RAM mà các lệnh ngoại trú chiếm sẽ được giải phóng.

Nhóm thứ ba là nhóm xử lý gói file (File batch). Nó chính là một file văn bản chứa các lệnh của DOS. Sau khi được người sử dụng gọi, loại file này được xử lý bởi một bộ thông dịch đặc biệt nằm trong phần tạm trú của bộ vi xử lý lệnh. Bộ thông dịch này thực hiện các lệnh liệt kê trong file batch như là chúng được đưa vào từ bàn phím. Đó là một file đặc biệt mà DOS nếu tìm thấy, sẽ gọi và thực hiện tự động mỗi khi khởi động hệ thống. Đầu tiên, các lệnh của file batch sẽ được kiểm tra để xác định đó là các lệnh nội trú, hay ngoại trú, hay là gọi một file batch khác.

Trong trường hợp thứ nhất, lệnh có thể được thực hiện ngay vì mã của nó đã có trong bộ nhớ (ở phần tạm trú của bộ xử lý lệnh). Trong các trường hợp còn lại, đó là lệnh ngoại trú hay file batch, trước tiên, bộ thông dịch sẽ tìm file mang tên lệnh trong thư mục hiện thời. Nếu không tìm thấy file nào như vậy trong thư mục hiện thời, nó sẽ tìm trong tất cả các thư mục đã chỉ ra trong lệnh PATH. Cần nói thêm là chỉ có các file có phần mở rộng COM, EXE và BAT được tìm.

2.2. CHƯƠNG TRÌNH COM VÀ CHƯƠNG TRÌNH EXE

Trong cơ chế quản lý của hệ điều hành, chúng ta đã biết 2 đối tượng phức tạp nhất cần kiểm soát chặt chẽ là các file chạy và không gian rộng lớn của bộ nhớ RAM, nơi chứa các dữ liệu rất đa dạng vừa có cấu trúc tĩnh vừa có cấu trúc động. Chúng ta lần lượt xét các đối tượng này trước hết là chương trình chạy với kiểu COM và EXE.

2.2.1. Chương trình kiểu COM

Đối với chương trình COM, trong một mảng 64 KB, các thanh ghi quản lý mảng đều nhận cùng một giá trị khi khởi động và trong quá trình thực hiện chương trình, giá trị này chỉ ra địa chỉ bắt đầu của một mảng 64 KB chứa chương trình COM.

Chỉ có nội dung của thanh ghi ES là có thể có giá trị khác, vì nó không có tác động trực tiếp tới việc thực hiện chương trình.

Các file COM lưu trên đĩa mềm hay đĩa cứng là hình ảnh chính xác của RAM khi thực hiện chương trình. File chương trình COM được nạp vào RAM và được khởi động thông qua hàm EXEC của DOS. Trước khi EXEC nạp chương trình vào bộ nhớ, nó sử dụng các hàm khác của DOS để chuẩn bị vùng nhớ RAM mà chương trình được gọi sẽ chiếm giữ. Hàm EXEC đặt vào đầu vùng nhớ này một cấu trúc dữ liệu gọi là PSP (Progam Segment Prefix). Chương trình sẽ được nạp vào ngay sau PSP, các thanh ghi mảng và thanh ghi ngăn xếp được khởi tạo giá trị ban đầu. Khi chương trình thực hiện xong, vùng nhớ RAM mà chương trình và PSP chiếm sẽ được giải phóng.

Bảng 2.1. **Tổ chức của PSP (Progam Segment Prefix)**

Địa chỉ	*Nội dung*	*Kiểu*
+00h	Lệnh INT 20h	2 byte
+02h	Địa chỉ mảng của ngăn nhớ cuối cùng dành cho ch.tr	1 word
+04h	Để dành	1byte
+05h	FAR CALL tới chương trình điều phối hàm của DOS	5 byte
+0Ah	Bản sao vector ngắt 22h	1 con trỏ
+0Eh	Bản sao vector ngắt 23h	1 con trỏ
+12h	Bản sao vector ngắt 24h	1 con trỏ
+16h	Để dành	22 byte
+2Ch	Địa chỉ mảng của khối biến môi trường	1 word
+2Eh	Để dành	46 byte
+5Ch	FCB#1	16 byte
+6Ch	FCB#2	16 byte
+8Oh	Số lượng ký tự trong dòng lệnh (không tính ký tự Enter)	1 byte
+81h	Dòng lệnh	127byte

PSP bao giờ cũng chiếm 256 byte và nó chứa nhiều thông tin quan trọng cho cả DOS và chương trình ứng dụng. Ở ngăn nhớ 00h là lời gọi hàm của DOS để kết thúc chương trình, giải phóng bộ và trả điều khiển về cho bộ xử lý lệnh COMMAND.COM hoặc cho chương trình chính. Một lời gọi FAR CALL tương đương chức năng của ngắt 21h được đặt trong ngăn nhớ 05h của PSP. Cần lưu ý rằng chỉ có các hàm từ 0 đến 24h là được gọi theo cách này, và số thứ tự của hàm không phải được đặt trong thanh ghi AH như vẫn thường xảy ra với INT 21h, mà là trong thanh ghi CL. Ở ngăn nhớ 02h của PSP là địa chỉ 16 bit của mảng chứa ngăn nhớ cuối cùng của RAM dành cho chương trình. Nếu một chương trình

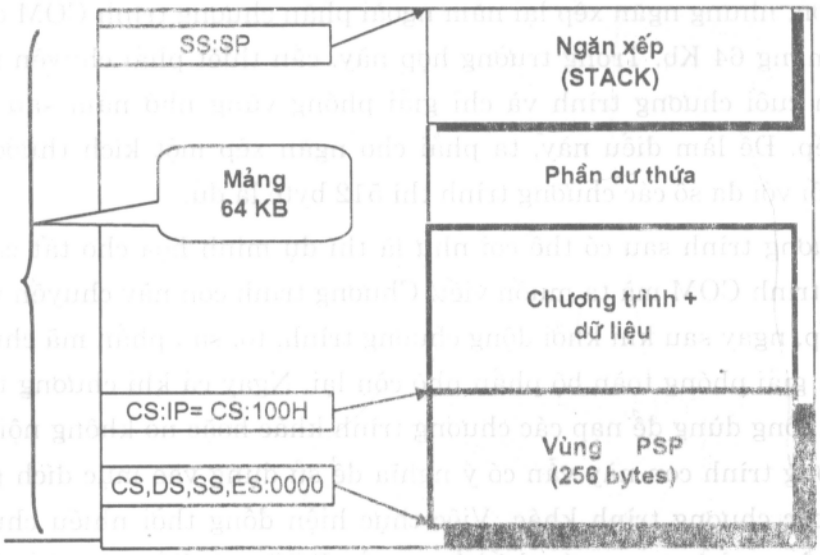
cần thêm RAM để thao tác, nó có thể xác định thông qua giá trị này. Nếu có thể sử dụng vùng nhớ đó cho các mục đích của mình. Nếu không, nó thông qua một số hàm của DOS để được cấp phát RAM bổ xung. Ngăn nhớ 0Ah, 0Eh và 12h là các vector ngắt để kết thúc chương trình, để phản ứng kịp thời khi các tổ hợp phím Ctrl-C hoặc Ctrl-Break được ấn và để phản ứng với một số lỗi xuất hiện. Nếu trong quá trình thực hiện mà nội dung các vector này bị thay đổi, thì sau khi chương trình kết thúc, DOS có thể ghi lại giá trị gốc của chúng vào bảng vector ngắt bằng cách copy lại các vector này từ PSP.

Địa chỉ mảng của khối biến môi trường được lưu dưới dạng một WORD ở ngăn nhớ 2Ch. Địa chỉ offset của khối này được bắt đầu từ 0000h. Khối biến môi trường là một loạt các chuỗi ký tự ASCII, mỗi chuỗi được kết thúc bằng một ký tự ASCII quy định. Các chuỗi ký tự này chứa các thông tin như đường dẫn tìm kiếm (lệnh PATH), thư mục chứa bộ xử lý lệnh.

Vùng bộ nhớ từ địa chỉ offset 80h còn được DOS dùng làm vùng chuyển dữ liệu đĩa DTA (Disk Transfer Area). Vì chương trình không cung cấp chỗ cho DTA ở một vùng nhớ khác, nên DOS dùng vùng này như là vùng đệm dữ liệu mỗi khi truy nhập file thông qua khối điều khiển file FCB.

Sau khi được nạp vào RAM, các file COM chạy ngay, do vậy các chương trình COM có thể được nạp và khởi động nhanh hơn so với các chương trình khác. Ngay sau khi chương trình COM được nạp vào bộ nhớ, nó được thực hiện bắt đầu từ ngăn nhớ ngay sau ngăn nhớ cuối cùng của PSP.

Khi điều khiển được trao cho chương trình COM, các thanh ghi mảng đều được đặt ở đầu của vùng PSP. Đầu của chương trình COM (so với đầu của PSP) luôn được đặt ở địa chỉ 100h. Con trỏ ngăn xếp nhận giá trị FFFEh ở cuối của đoạn 64 Kb mà chương trình COM chiếm. Cần lưu ý là DOS không nạp chương trình COM vào một địa chỉ xác định trước. Do vậy, các chương trình COM không nhất thiết phải chứa địa chỉ mảng một cách cụ thể.



Hình 2.1. Phân bố các mảng DS, CS, SS trong file COM.

Khi gọi một chương trình COM, DOS dành cho chương trình này toàn bộ RAM còn tự do. Sẽ không có vấn đề gì trong đa số các trường hợp, vì chương trình sẽ bị loại khỏi bộ nhớ sau khi kết thúc. Nhưng nếu một chương trình COM được chuyển thành nội trú, có nghĩa là nó còn tồn tại trong bộ nhớ sau khi thực hiện, thì một số vấn đề sẽ xuất hiện, vì dưới quan điểm của DOS, không còn bộ nhớ tự do, và do vậy, không thể nạp và chạy các chương trình khác.

Vấn đề cũng sẽ xuất hiện, khi một chương trình COM trong quá trình thực hiện yêu cầu nạp và thực hiện một chương trình khác thông qua hàm EXEC. Điều này cũng không thực hiện được, bởi vì DOS cho rằng không còn bộ nhớ tự do để nhận chương trình được gọi. Hai trường hợp trên có thể được giải quyết bằng cách giải phóng vùng nhớ mà chương trình không dùng đến.

Có hai khả năng để thực hiện điều này: hoặc là chỉ giải phóng phần bộ nhớ nằm ngoài mảng COM 64 KB, hoặc giải phóng tất cả các phần nhớ mà chương trình không dùng đến, tính cả vùng nhớ không sử dụng nằm trong mảng COM. Cách thứ hai cho phép giải phóng được nhiều

RAM 1 Mb, nhưng ngăn xếp lại nằm ngoài phần chương trình COM do nó ở cuối mảng 64 Kb. Trong trường hợp này, cần thiết phải chuyển ngăn xếp đến cuối chương trình và chỉ giải phóng vùng nhớ nằm sau đuôi ngăn xếp. Để làm điều này, ta phải cho ngăn xếp một kích thước xác định. Đối với đa số các chương trình thì 512 byte là đủ.

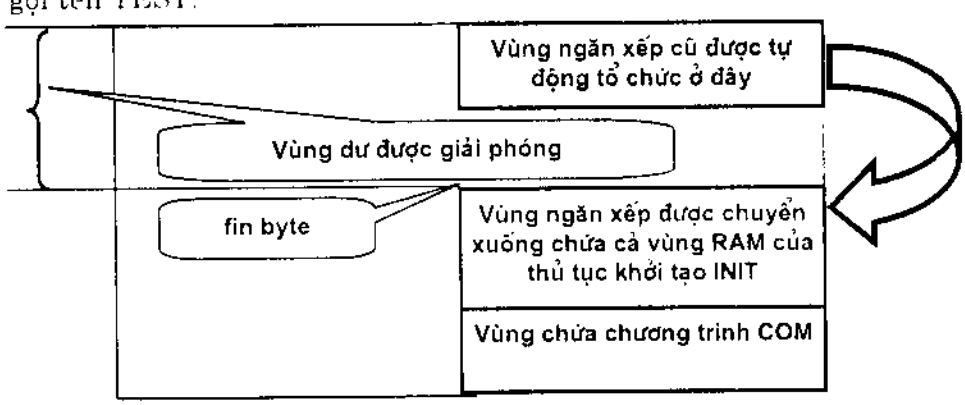
Chương trình sau có thể coi như là thí dụ minh họa cho tất cả các chương trình COM mà ta muốn viết. Chương trình con này chuyển vùng ngăn xếp, ngay sau khi khởi động chương trình, tới sau phần mã chương trình và giải phóng toàn bộ phần nhớ còn lại. Ngay cả khi chương trình của ta không dùng để nạp các chương trình khác hoặc nó không nội trú, thì chương trình con này vẫn có ý nghĩa để sử dụng vào mục đích ghép nó vào các chương trình khác. Việc thực hiện đồng thời nhiều chương trình là được phép trong các thế hệ sau của DOS, mà điều này chỉ có thể khả thi nếu mọi chương trình đều được sử dụng một bộ nhớ đủ cho nó. Muốn làm được như thế thì phần nhớ không sử dụng phải được giải phóng. Chương trình ASSEMBLY dưới đây minh họa cơ chế giải phóng vùng nhớ dư thừa mà hệ điều hành cấp phát cho chương trình.

Chương trình nguồn: COM_TYPE.ASM

```
;-------------------------------------------------------
code segment  para  'CODE'   ; Định nghĩa mảng mã lệnh code
  org 100h
  assume cs:code,ds:code,es: code,ss: code
  start:jmp init             ; gọi thủ tục khởi tạo
;-------------------------------------------------------
; vùng data: khai báo mọi kiểu dữ liệu ở đây
;-------------------------------------------------------
;------- vùng mã lệnh chứa chương trình ---------------
prog proc near   ; thủ tục này thực hiện ngay sau khi
                 ; khởi tạo
  mov ax,4c00h   ; kết thúc ch.tr bằng hàm 4ch của ngắt 21h
  int 21h
prog endp
```

```
  init:
    mov ah,4ah          ; Hàm giải phóng bộ nhớ
    mov bx,offset fin   ; Tính số lượng mảng(16 bytes)
                        ; mà ch.tr cần
    mov cl,4
    shr bx,cl
    inc bx
    int 21h; Gọi ngắt
    mov sp,offset fin   ; Đặt lại con trỏ ngăn xếp
    jmp prog
fin_init label near
;-------------------------------------------------------
; Vùng ngăn xếp
;-------------------------------------------------------
  dw (256-((fin_init-init)shr 1)) dup (?)
                            ; Vùng ngăn xếp kích thước
                            ; 256 bytes
  fin equ this byte         ; byte cuối cùng của ch.tr
;-------------------------------------------------------
code ends
end start
```

Mô tả chương trình trên bằng hình 2.2. Nếu các bước trên được tiến hành tốt có thể chạy chương trình TEST.COM từ mức DOS bằng cách gọi tên TEST.



Hình 2.2. Vùng RAM dư của file COM được giải phóng

2.2.2. Chương trình kiểu EXE

Đối với các chương trình EXE, sự tổ chức các mảng không ngặt nghèo như chương trình COM. Các khối mã lệnh, dữ liệu, ngăn xếp được đặt trong các mảng khác nhau, mỗi khối lại có thể tùy theo kích thước của mình mà được phân bố trong một hay nhiều mảng. Vì lẽ đó mà các thanh ghi mảng có giá trị khác nhau trong quá trình thực hiện một chương trình EXE.

So với chương trình COM, chương trình EXE có ưu điểm là không bị hạn chế bởi một mảng là 64 KB. Song cấu trúc các file EXE phức tạp hơn nhiều do một loạt thông tin cần hỗ trợ cho trình biên dịch biết về việc phân phối mảng nhớ cho từng vùng dữ liệu. Các chương trình EXE còn thể hiện một ưu điểm nữa là dễ thích ứng với các đổi mới của DOS, thí dụ như khả năng làm việc ở chế độ đa nhiệm.

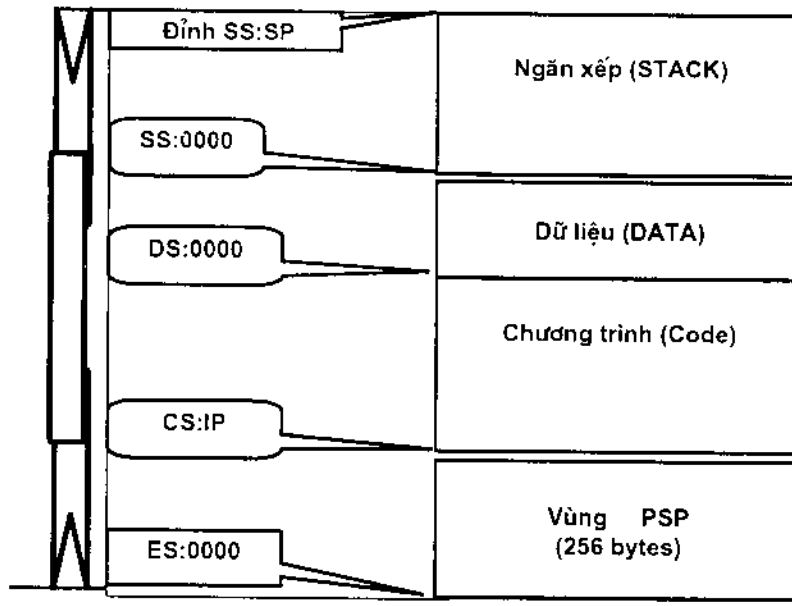
Chương trình EXE chứa các mảng riêng biệt cho mã lệnh, dữ liệu và ngăn xếp. Các mảng này có thể xuất hiện trong một trật tự bất kỳ. Khác với chương trình COM, các chương trình EXE không chạy ngay được, nó cần được chuẩn bị bởi một chương trình con của hàm EXEC.

Chương trình LINK đặt vào đầu các file EXE một cấu trúc dữ liệu chứa địa chỉ tương đối của các mảng. Địa chỉ mảng thực hiện trong bộ nhớ RAM nhận được bằng cách cộng địa chỉ tương đối này với địa chỉ của mảng mà chương trình được nạp vào, thường là địa chỉ mảng của PSP + 10h.

Khi hàm EXEC nạp một chương trình EXE, nó biết được địa chỉ các ngăn nhớ chứa các địa chỉ mảng cần phải sửa đổi lại cho phù hợp. Nó viết lại các giá trị này bằng cách cộng các giá trị đó với địa chỉ mảng bắt đầu (PSP + 10h). Thao tác này làm cho công việc gọi và khởi động một chương trình EXE lâu hơn một chương trình COM, và kích thước một file EXE bao giờ cũng dài hơn kích thước một file COM tương đương. Cấu trúc của khối tiêu đề (header) của một file EXE có dạng như bảng 2.2.

***Bảng 2.2.* Cấu trúc của khối tiêu đề file EXE**

Địa chỉ	*Nội dung*	*Kiểu*
+00h	Đánh dấu đây là một chương trình EXE (5A 4Dh)	Word
+02h	(Độ dài của file) MOD 512	Word
+04h	(Độ dài của file) DIV 512	Word
+06h	Số lượng địa chỉ mảng cần được sửa lại cho phù hợp	Word
+08h	Kích thước của header(đơn vị paragraph =16byte)	Word
+0Ah	Số lượng nhỏ nhất các paragraph cần bổ sung	Word
+0Ch	Số lượng lớn nhất các paragraph cần bổ sung	Word
+0Eh	Địa chỉ mảng tương đối của mảng ngăn xếp (để 1 Word nhận được địa chỉ mảng trong RAM, nó được cộng với PSP +10h)	Word
+10h	Nội dung thanh ghi SP lúc khởi động chương trình	Word
+12h	Kiểm tra chẵn lẽ phần tiêu đề của file.	Word
+14h	Nội dung thanh ghi IP vào lúc khởi động chương trình	Word
+16h	Địa chỉ bắt đầu của mảng mã trong file EXE	Word
+18h	Địa chỉ của mảng định lại giá trị (nó chứa địa chỉ của bằng các địa chỉ cần phải được định lại giá trị cho phù hợp khi nạp vào RAM)	Word
+1Ah	Số overlay	Word
+1Ch	Bộ nhớ đệm	Thay đổi
+??	Bảng chứa các địa chỉ cần xác định lại giá trị biến	Thay đổi
+??	Mã chương trình, các mảng số liệu và ngăn xếp.	Thay đổi



Hình 2.3. **Phân bố các mảng DS, CS, SS trong file EXE**

Sau khi các địa chỉ mảng được sửa lại thành các địa chỉ có hiệu lực, hàm EXEC cố định các thanh ghi mảng DS và ES theo đầu của PSP (DS = ES= PSP). Do vậy chương trình EXE có thể truy nhập dễ dàng đến các thông tin trong PSP, như địa chỉ khối biến môi trường và các tham số trong dòng lệnh. Địa chỉ của ngăn xếp và nội dung con trỏ ngăn xếp được lưu ở khối đầu file EXE, từ đó, nó được lấy lại để nạp vào bộ nhớ, chú ý là địa chỉ mảng của ngăn xếp SS phải được sửa lại. Điều đó cũng đúng đối với địa chỉ mảng mã lệnh và nội dung con trỏ lệnh. Sau khi chúng nhận được các giá trị xác định, chương trình bắt đầu thực hiện.

Cần lưu ý là phải dành bộ nhớ cho một chương trình EXE khi nó được nạp vào RAM. Điều này xảy ra hoàn toàn khác so với chương trình COM. Chương trình nạp EXE có thể nhận được từ file EXE kích thước các mảng của chương trình và do vậy cùng lúc nó nhận được cả kích thước toàn thể của chương trình trong bộ nhớ. Sau đó, nó có thể yêu cầu bộ nhớ cần thiết thông qua một hàm khác. Dung lượng bộ nhớ được

dành phải lớn hơn kích thước của chương trình, vì trong quá trình thực hiện chương trình cần thêm một vùng RAM cho chuyển đổi dữ liệu. Kích thước của vùng RAM bổ sung này được xác định theo nội dung của hai trường trong header của file EXE, chúng chỉ ra kích thước nhỏ nhất và lớn nhất của vùng nhớ bổ sung tính theo đơn vị paragraph (= 16 byte).

Chương trình nạp EXE đầu tiên thử dành cho chương trình số lượng lớn nhất các paragraph. Nếu không thể được, nó dành phải bằng lòng với bộ nhớ còn lại, với điều kiện là bộ nhớ này vẫn lớn hơn số lượng nhỏ nhất các paragraph. Hai trường trên được xác định không phải bởi chương trình LINK mà bởi chương trình Compiler hay Assembler.

Các chương trình EXE có cấu trúc không phù hợp khi muốn cài đặt chúng như là chương trình nội trú, nhưng cấu trúc này lại thuận tiện cho trường hợp khi một chương trình nào đó có nhu cầu gọi một chương trình khác trong khi nó đang thực hiện. Mà điều này chỉ thực hiện được, cũng như đối với chương trình COM, nếu bộ nhớ không dùng phải được giải phóng. Dưới đây là một minh họa việc giải phóng bộ nhớ không dùng đến của chương trình EXE.

Chương trình nguồn: EXE_TYPE.ASM

```
; ------------------------------
; Tổ chức mảng Ngăn xếp
; ------------------------------
stack segment para 'stack'
  dw 256 dup (?)
stack ends
; ------------------------------
; Tổ chức mảng mã lệnh
; ------------------------------
code segment para ' code '
  assume cs:code,ds:data,ss:stack
prog proc far
  mov ax,data
  mov ds,ax
```

```
   call setfree
   mov ax,4c00h
   int 21h
prog endp
setfree proc near
;-------------------------------------------------------
;es là địa chỉ của PSP
; mảng stack luôn là mảng cuối
;trong file  EXE
;es:0000 trỏ tới đầu còn ss:sp trỏ tới cuối chương
;trình trong bộ nhớ
;-------------------------------------------------------
   mov bx,ss
   mov ax,es
   sub bx,ax ;hiệu của hai mảng
   mov ax,sp ;nội dung của sp là kích thước mảng ngăn xếp
   mov cl,4
   shr ax,cl
   add bx,ax      ; cộng với số đã tính trước
   inc bx         ; tăng thêm mảng  đoạn cho chắc chắn
   mov ah,4ah     ; truyền kích thước mới cho DOS
   int 21h
   ret
setfree endp
code dends
end prog
;-----------------------------
```

Cấu trúc của chương trình gồm: một mảng mã lệnh, một mảng dữ liệu và một mảng ngăn xếp riêng biệt. Chừng nào các mảng mảng không vượt quá 64 KB, tức là không phải phân chia chúng thành nhiều mảng vật lý, thì có thể sử dụng chương trình trên đây làm cơ sở cho các chương trình của mình.

2.3. QUẢN LÝ BỘ NHỚ RAM CỦA HỆ ĐIỀU HÀNH

Như đã biết, bộ nhớ trung tâm của PC được chia làm hai vùng: vùng tham số hệ thống và vùng dành cho chương trình ứng dụng. DOS sử dụng vùng thứ nhất để chứa các tham số của hệ điều hành. Bắt đầu từ ngăn nhớ thấp nhất 0000:0000 để chứa bảng vector ngắt cũng như một số bảng, vùng đệm, các biến trong và phần mã nội trú của hệ điều hành. Ngoài ra còn có các chương trình DRIVER điều khiển thiết bị được ghép vào hệ thống, và chúng có thể được gọi như cách ta gọi một hàm nào đó của DOS.

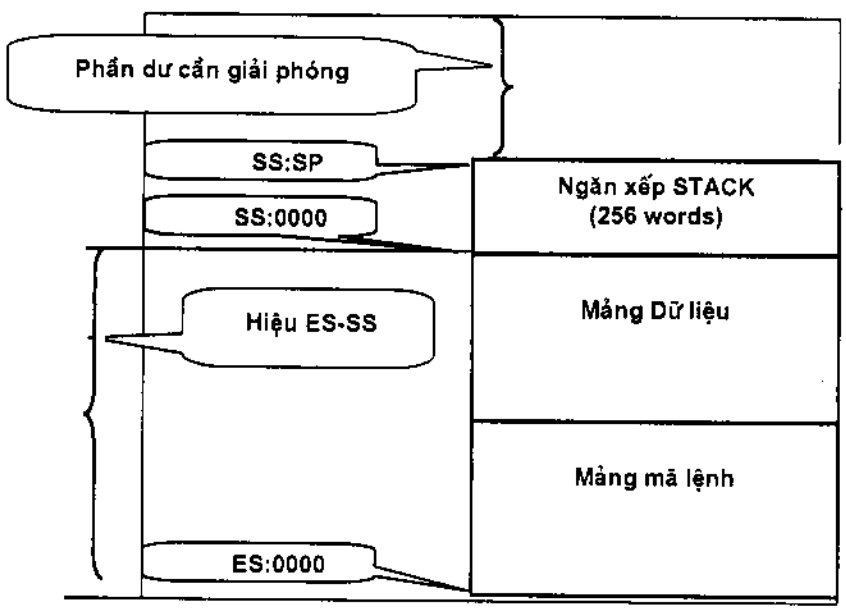
Vùng RAM thứ hai là vùng chứa các chương trình ứng dụng TPA (Transient Program Area). Vùng này chứa các chương trình cần chạy cũng như các khối môi trường tương ứng, và nó nằm ngay sau vùng hệ điều hành. Tùy theo yêu cầu của các chương trình, DOS cung cấp cho mỗi chương trình một vùng nhớ, vùng nhớ này được quản lý nhờ một khối dữ liệu đứng ngay trước mỗi vùng nhớ. Khối dữ liệu này là khối điều khiển bộ nhớ MCB(Memory Control Block). Nó luôn có kích thước là 16 byte (1 paagraphe) và nó được bắt đầu từ một địa chỉ là bội nguyên của 16 và đứng ngay trước vùng nhớ được cung cấp cho chương trình. Trong các hàm quản lý bộ nhớ, DOS luôn làm việc với địa chỉ mảng của vùng nhớ được cấp cho chương rình, nhưng địa chỉ mảng của MCB có thể dễ dàng tính được bằng cách lấy địa chỉ mảng của vùng nhớ trừ 1. Cấu trúc của một MCB được mô tả ở bảng 2.3.

***Bảng 2.3.* Cấu trúc của MCB**

Địa chỉ	*Nội dung*	*Kiểu*
+00h	("Z"= MCB cuối cùng,"M"= còn có MCB tiếp theo)	1 byte
+01h	Địa chỉ mảng của PSP của chương trình chạy tương ứng	1word
+03h	Số lượng Paragraphe trong vùng nhớ được cấp	1 word
+05h	Không dùng	11 byte
+10h	Vùng nhớ được cấp bắt đầu từ địa chỉ này	

Theo bảng cấu trúc ở bảng 2.3 MCB gồm 3 trường, ở trường thứ nhất là trường đặc tả ID chỉ chứa một trong hai ký tự: "M"- chỉ ra rằng sau MCB này còn có các MCB khác, còn " Z" chỉ ra rằng MCB này là MCB cuối cùng trong bộ nhớ. Trường thứ hai chứa địa chỉ mảng PSP của chương trình.

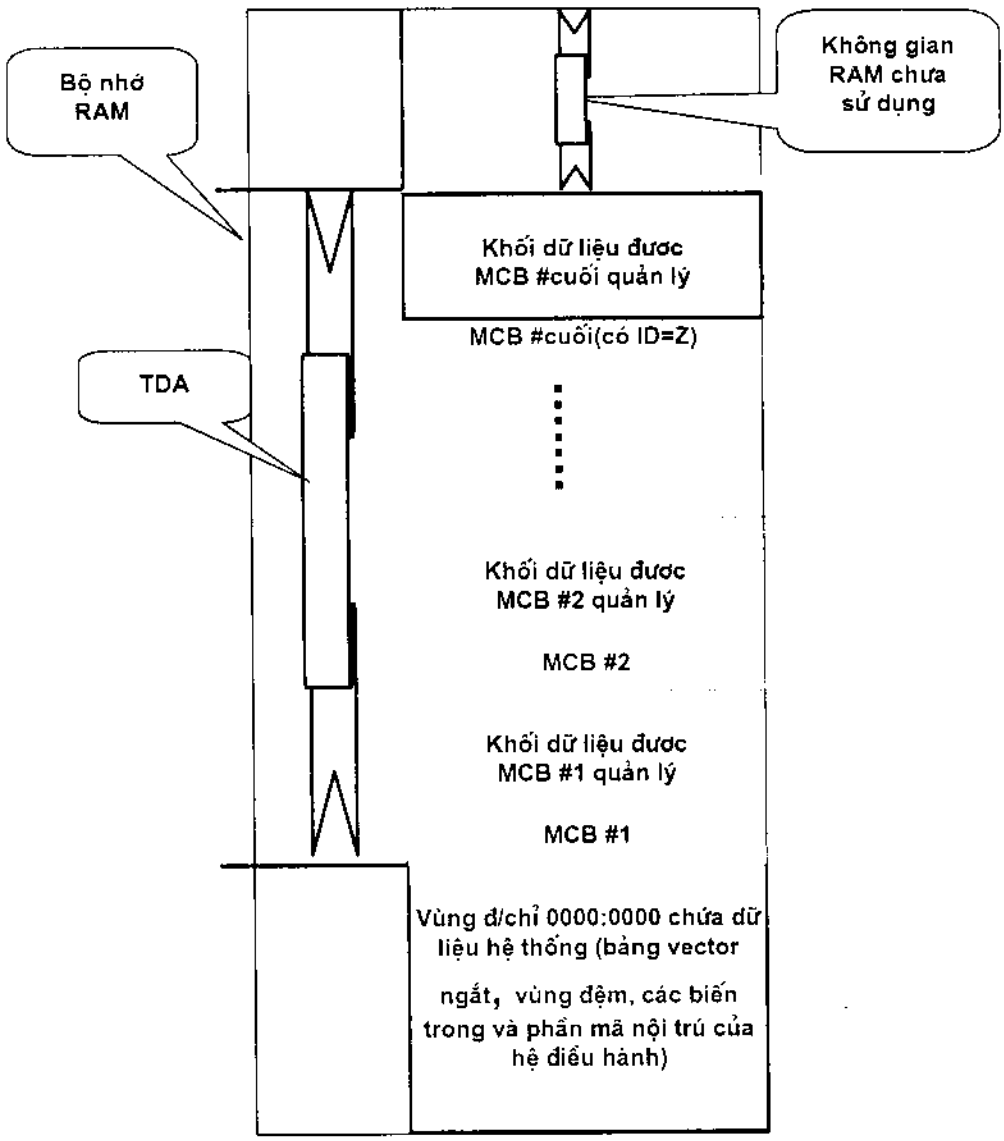
Trường thứ ba của MCB là kích thước của vùng nhớ được cấp theo đơn vị Paragraph. Thực vậy, vì MCB tiếp theo bắt đầu ngay sau vùng nhớ được cấp (nếu trường ID không chứa ký tự Z), nên trường này cũng chỉ ra khoảng cách đến MCB tiếp theo. Mỗi một MCB chỉ ra MCB tiếp theo một cách gián tiếp. Do vậy, ta nhận được một chuỗi cho phép tìm lại danh sách tất cả các MCB.



Hình 2.4. **Giải phóng phần RAM dư trong file EXE**

Chương trình minh họa dưới đây đủ thông minh để phân biệt các vùng nhớ chuẩn (chứa môi trường chạy của một chương trình- được gọi là vùng biến môi trường hoặc chứa vùng cấu trúc chuẩn PSP) hay vùng phi chuẩn (các thông tin khác). Nhiệm vụ của chương trình là chuyển từ khối MCB này đến MCB khác để kiểm tra các vùng nhớ được cấp phát. Để nhảy đến MCB tiếp theo, nó sử dụng trường thứ ba của MCB, thông

qua trường này, nó đặt con trỏ tới MCB tiếp theo. Bằng cách đó, nó tạo thành một chuỗi đi từ MCB đầu tới MCB cuối (trường ID của MCB cuối chứa ký tự "Z").



Hình 2.5. Cơ chế quản lý bằng MCB

Để có thể vào được bên trong chuỗi các MCB, cần phải tìm ra địa chỉ của mắt xích đầu tiên, có nghĩa là MCB đầu tiên DOS viết địa chỉ này

vào một cấu trúc bên trong gọi là DIB (DOS information Block- khối thông tin của DOS), các chương trình ứng dụng không truy nhập được tới nó, vì nó liên quan đến một hàm không được DOS cung cấp tài liệu. Tuy vậy, địa chỉ của cấu trúc này có thể nhận được thông qua hàm 52h, hàm này trả lại địa chỉ của DIB trong cặp thanh ghi ES: BX.

Địa chỉ nhận được trong cặp ES:BX không trỏ trực tiếp đến trường đầu tiên của DIB mà tới trường thứ hai. Nhưng vì trường đầu tiên mới là trường chứa địa chỉ MCB đầu tiên mà chúng ta quan tâm, nên thông tin cần tìm nằm ở phía trên của địa chỉ ES:BX. Con trỏ trỏ tới MCB đầu tiên gồm địa chỉ offset và địa chỉ đoạn, kích thước của nó là 4 byte, và do vậy nó nằm ở địa chỉa ES: (BX-4). Địa chỉ nhận được theo cách này cần được sử dụng một cách thận trọng. Thực vậy, ta có thể nghĩ rằng chỉ cần giảm BX đi 4 là nhận được địa chỉ thực nằm trong cặp thanh ghi ES: BX. Nhưng điều đó chỉ đúng với điều kiện là địa chỉ offset trong BX phải lớn hơn hoặc bằng 4. Nếu nó nhỏ hơn 4, cách này dẫn tới một kết quả sai, vì ta nhận được một giá trị âm, mà sự địa chỉ hóa bộ nhớ không cho phép nhận giá trị âm. Chúng ta lấy một thí dụ để hiểu vấn đề này hơn.

Nếu BX = 0, ta nhận được giá trị OFFFCh sau khi thực hiện BX-4. Trong các phép toán số học, giá trị này được hiểu là -4, nhưng khi truy nhập bộ nhớ, nó không chỉ địa chỉ -4 mà chỉ địa chỉ OFFFCh, và vậy là ta không về được đầu mảng đang xét, mà nhẩy đến cuối mảng này.

Để tránh sai sót này, trước tiên, chương trình giảm địa chỉ mảng đi 1 đơn vị. Khi đó địa chỉ thực tế trong ES:BX giảm đi 16 đơn vị. Sau đó ta cộng địa chỉ offset trong BX với 12, và bây giờ ta sẽ có địa chỉ mới nhỏ hơn 4 so với địa chỉ lúc ban đầu trong ES: BX.

Từ mắt xích đầu tiên này, ta sẽ lần ra tất cả các MCB tiếp theo. Chương trình sẽ kiểm tra, đánh gía các MCB, hiện thị ra màn hình một số thông tin về MCB và vùng nhớ mà nó quản lý theo các trường sau:

- Số thứ tự MCB
- Địa chỉ của nó trong bộ nhớ
- Địa chỉ của vùng nhớ mà MCB quản lý

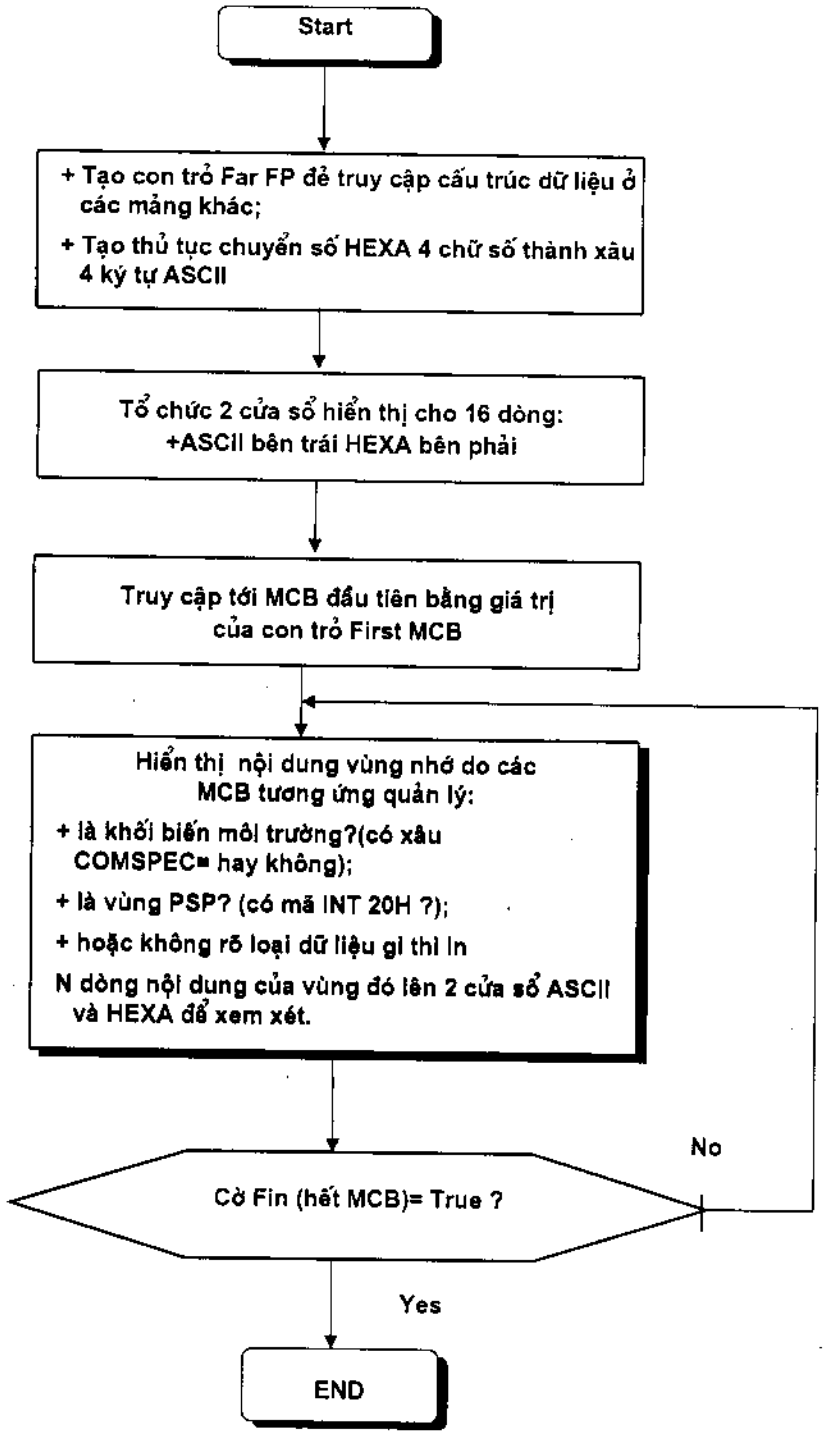
- Nội dung của trường ID ("M" hay "Z")
- Địa chỉ của PSP, nó có tồn tại thực sự hay không
- Kích thước vùng nhớ (tính bằng Paragraphe và byte)

Địa chỉ vùng nhớ nhận được bằng cách lấy địa chỉ mảng của MCB cộng 1 đơn vị. Đầu tiên ta kiểm tra xem vùng nhớ có phải là khối môi trường hay không. Ta có thể khẳng định nó là khối môi trường nếu xâu ký tự có dạng COMSPEC=, là chuỗi đầu tiên của mọi khối môi trường, có mặt ở đầu vùng nhớ. Nếu chuỗi này được nhận dạng, chương trình sẽ cho rằng đây là một khối môi trường và nó cho hiện tất cả các xâu ra màn hình. Ta cũng cần phải xử lý xâu ký tự của tên chương trình sở hữu khối môi trường nằm ở cuối khối này. Nếu vùng nhớ không phải là khối môi trường thì nó có thể là một PSP, theo sau đó là một chương trình tạm trú hay nội trú. Chương trình sẽ coi vùng nhớ là PSP nếu hai ô nhớ đầu của vùng nhớ chứa lệnh mã máy INT 20h (mã CDh và 20h), bởi mọi PSP đều bắt đầu bằng lệnh này.

Nếu hai trường hợp trên không được nhận dạng, thì chương trình không thể xác định vùng nhớ là mã chương trình, dữ liệu hay là một cái gì khác. Để có thể xem xét thêm, chương trình cho hiện thị lên màn hình 80 byte đầu tiên của vùng nhớ dưới dạng mã ASCII và hexa. Sau khi người sử dụng ấn một phím bất kỳ, chương trình sẽ xử lý MCB tiếp theo cho đến hết.

Chương trình minh họa viết bằng Pascal cho phép hiện thị chuỗi các MCB lên màn hình. Chương trình truy nhập đến bộ nhớ thông qua con trỏ FAR, vì các vùng nhớ nằm ngoài mảng dữ liệu của chương trình. Thuật toán chương trình được thể hiện ở hình 2.6.

Trong chương trình này giải quyết vấn đề chuyển hai địa chỉ mảng và offset thành một con trỏ FAR. Trong chương trình nguồn này, Turbo Pascal chuyển các địa chỉ thành con trỏ FAR được thực hiện bằng một thủ tục nhỏ Inline, nó viết hai địa chỉ trực tiếp vào các ngăn nhớ của con trỏ bằng các lệnh mã máy.



Hình 2.6. Lưu đồ thật toán của chương trình kiểm soát vùng RAM

```
program Manage_RAM;
  uses dos, crt;
  type byteptr=^byte;
    zone=array[0..1000] of byte;
    ptrzon=^zone;
    MCB=record
      idcode:char;
      psp:word;
      distance:word;
    end;
    mcbptr =^mcb;
   mcbptr2=^mcbptr;
    chahex=string[4];
  var cvhstr:chahex;
(*****************************************)
function  getdosver:byte;
  var regs:registers;
begin
  regs.ah:=$30;
  msdos(regs);
  getdosver:=regs.al+10+regs.ah;
end;
(*****************************************)
{$F+}     {Chỉ truy cập bằng con trỏ xa}
function  MK_FP(SEG,OFS:word):byteptr;
begin
   inline($8b/$46/$08/       {mov ax,[bp+8]}
          $89/$46/$fe/       {mov [bp-2],ax}
          $8b/$46/$06/       {mov ax,[bp+6]}
          $89/$46/$fc);      {mov [bp-4],ax}
end;
```

```
{$F-}     {lại dùng các thủ tục near}
(*************************************************)
(* số 2 bytes --> chuỗi hexa 4 chữ số*)
(*-----------------------------------------------*)
function  ChaineHexa(hexVal:word):chahex;
var
  compteur,
  nibble:byte;
begin
  cvhstr:='xxxx';
  for compteur:=4 downto 1 do
    begin
      nibble:=hexval and $000f;
      if(nibble>9) then
      cvhstr[compteur]:=chr(nibble-10+ord('A')) {đúng}
     else  cvhstr[compteur]:=chr(nibble+ord('0'));
     hexval:=hexval shr 4;
    end;
   chainehexa:=cvhstr;
end;
(**88888888888*****************************************)
(*------ cung cấp con trỏ tới MCB đầu tiên----------*)
(*-----------------------------------------------*)
function  FirstMCB:      MCBptr;
var
  regs:registers;
begin
  regs.ah:=$52;  {hàm lấy d/c khối tt của dos}
  msdos(regs);   {...es:(bx-4) trỏ mcb đầu tiên}
  firstMCB:=MCBptr2(MK_FP(regs.es-1,regs.bx+12))^;
end;
```

```
(*****************************************************)
(*--- Dump:hiển thị RAM dạng HEXA và ASCII---*)
(*-----------------------------------------------------*)
Procedure Dump(DPtr:ptrZon;Nmb:byte);
type
  HBStr=string[2];          {nhận số hex 2 số}
var
   offset,                  {offset vùng nhớ}
   z:integer;               {bộ đếm}
   chahex:hbstr;            {nhận số hex để hiển thị}
(*-----------------------------------------------------*)
procedure hexbyte(hbyte:byte);
  begin
       chahex[1]:=chr((hbyte shr 4)+ord('0'));
       if   chahex[1]>'9' then
            chahex[1]:=chr(ord(chahex[1])+7);
         chahex[2]:=chr((hbyte and 15)+ord('0'));
       if   chahex[2]>'9' then
            chahex[2]:=chr(ord(chahex[2])+7);
end;
(*-----------------------------------------------------*)
begin
   chahex:='xx';
  writeln;
   write('dump 0123456789ABCDEF 00 01 02 03 04 05 06 07 08');
   writeln('09 0A 0B 0C 0D 0E 0F');
  write  ('                                          ');
   writeln('                                          ');
   offset:=0;
  while Nmb>0 do
     begin
       write(chainehexa(offset),'   ');
       for z:=0 to 15 do
```

```
          if (dptr^[offset+z]>=32) then
          write(chr(dptr^[offset+z]))
          else
          write(' ');
        write('        ');
        for z:=0 to 15 do
            begin
            hexbyte(dptr^[offset+z]);
              write(chahex,' ');
            end;
        writeln;
        offset:=offset+16;
        dec(nmb);
     end;
    writeln;
  end;
(*******************************************************)
(*--------Lần theo danh sách MCB------------------  *)
(*-------------------------------------------------*)
Procedure TraceMCB;
const  comspec:array[0..7] of char='COMSPEC=';
  var mcbact: mcbptr;
      fin: boolean;
      touche:char;
      nrmcb,
      z:integer;
      memptr:ptrzon;
      dosver:byte;
   begin
      dosver:=getdosver;
      fin:=false;
      nrmcb:=1;
      mcbact:=firstmcb;
```

```
  repeat
    if mcbact^.idcode='2' then
      fin:=true;
    writeln('so mcb =',nrmcb);
   writeln('dia chi MCB =',chainehexa(seg(mcbact^)),':',
                          chainehexa(ofs(mcbact^)));
    writeln('dia chi vung nho = ',
                   chainehexa(succ(seg(mcbact^))),':',
                   chainehexa(ofs(mcbact^)));
    writeln('ID =',mcbact^.idcode);
    writeln('dia chi PSP     =',
                   chainehexa(mcbact^.PSP),':0000');
    writeln('kich thuoc      =',
         mcbact^.distance,'paragraphes',
        '(',longint(mcbact^.distance) shl 4,'byte)');
    write(' Noi dung       =');
(*---vùng nhớ là khối biến môi trường thì phải có xâu ký tự
nhận dạng là xâu 'COMSPEC=-' ở ngay đầu------*)
    z:=0;
    memptr:=ptrzon(mk_fp(succ(seg(mcbact^)),0));
    while((z<=7) and (ord(comspec[z])=memptr^[z])) do
      inc(z);
    if z>7 then
      begin
        writeln('Eviroment');
        memptr:=ptrzon(mk_fp(succ(seg(mcbact^)),0));
        if dosver>=30 then
          begin
          write('Tên chương trình =');
           z:=0;
          while not((memptr^[z]=0) and
                          (memptr^[z+1]=0)) do
                          inc(z);
               z:=z+4;
```

```
            if memptr^[z]<>0 then begin
              repeat
                write(chr(memptr^[z]));
                inc(z);
            until memptr^[z]=0;
          writeln;
        end
      else
        writeln('không nhận dạng được tên');
    end;

(*--------in ra các chuỗi biến môi trường----------*)
    writeln(#13,#10,'các chuỗi khối môi trường');
   z:=0;
   while memptr^[z]<>0 do
      begin
        write('            ');
        repeat
          write(chr(memptr^[z]));
          inc(z);
        until memptr^[z]=0;
        inc(z);
        writeln;
      end;
      end
   else
      begin
(* -- là  vùng chuẩn PSP? thì bắt đầu phải là lệnh INT 20H(mã
hex: $CD  $20)-------------------------------*)
              memptr:=ptrzon(mk_FP(succ(seg(mcbact^)),0));
  if ((memptr^[0]=$cd) and (memptr^[1]=$20)) then
  begin
      writeln('PSP (theo sau là chương trình)');
```

```
    end
   else
      begin
        writeln('không tìm được');
        dump(memptr, 5); {in 5 dòng}
      end;
    end;
   write  ('                                    ');
   writeln('  press any key                     ');
   if (not fin) then
     begin
      mcbact:=mcbptr(mk_fp(seg(mcbact^)+
              mcbact^.distance+1,0));
      inc(nrmcb);
      touche:=readkey;
    end;
   until (fin);
 end;
(*************************************************)
BEGIN
  CLRSCR;
  TRACEMCB;
END.
```

2.4. TỔ CHỨC CỦA ROM BIOS

BIOS (Basic Input Output System) là hệ thống vào/ra thông tin cơ sở nhất. Nhiệm vụ chính của BIOS là cung cấp các thủ tục vào/ra ở mức hệ thống để có thể thực hiện các trao đổi thông tin giữa bộ xử lý trung tâm và các thiết bị ngoại vi như bàn phím, màn hình, đĩa....

Nhờ có BIOS mà người lập trình không phải thực hiện công việc hết sức phức tạp, đó là sự thích ứng của chương trình với phần cứng của các PC khác nhau. BIOS có một ý nghĩa đặc biệt quan trọng là cho phép các chương trình viết cho IBM PC chạy được trên các máy PC tương thích,

mặc dù phần cứng và phần mã cụ thể của các thủ tục BIOS trên các máy này không hoàn toàn giống với IBM PC. Chính việc gọi các thủ tục của BIOS không phụ thuộc vào phần cứng của một máy cụ thể nào đã góp phần đáng kể làm cho PC trở nên phổ cập. Điều này cho phép các nhà sản xuất máy tính tạo ra những máy tương thích với IBM PC mà các phần mềm chuẩn đều có thể chạy trên chúng.

Chuẩn BIOS do IBM thiết lập bao gồm các hàm của BIOS được gọi thông qua các ngắt từ 10h đến 17h và ngắt. Việc sử dụng các thanh ghi của bộ vi xử lý cũng được tiêu chuẩn hoá. Các thanh ghi này dùng để trao đổi dữ liệu giữa chương trình gọi và ngắt.

10h: Gọi các hàm màn hình của BIOS

11h: Xác định cấu hình của máy tính

12h: Tính kích thước RAM

13h: Gọi các hàm điều khiển đĩa mềm/đĩa cứng của BIOS

14h: Gọi các hàm truyền tin không đồng bộ

15h: Gọi các hàm điều khiển cassette

16h: Kiểm tra bàn phím

17h: Gọi các hàm điều khiển máy in

1AH: Gọi các hàm điều khiển đồng hồ hệ thống

BIOS được nạp trong ROM của MAIN BOARD, ở vùng địa chỉ cao, bắt đầu từ F000:E000 (BIOS IBM chuẩn). BIOS trải dài tới địa chỉ F000: FFFF, là ngăn nhớ cuối cùng mà Intel 80x86 có thể gọi tới. Cũng như các thủ tục khác, các thủ tục của BIOS có thể tạo ra, cất giữ và thay đổi các nội dung các biến hệ thống. Các biến của BIOS được đặt trong RAM, ở vùng nhớ thấp, bắt đầu từ địa chỉ 0040:0000.

2.4.1. Tài nguyên phần mềm của BIOS

Tài nguyên BIOS dưới dạng các chương trình chứa trong Rom BIOS đặt ở mảng nhớ 64KB cuối cùng gồm 3 phần chính:

- Một tập hợp các chương trình con cho phép các chương trình ứng dụng và hệ điều hành trao đổi dữ liệu với tài nguyên phần cứng của PC.

- Những chương trình dịch vụ ngắt (Routine) giải quyết các dịch vụ ngắt cứng (hard interrup).
- Chương trình tự kiểm tra khi bật máy POST (Power - On Self Test).

Nhờ các dịch vụ của BIOS mà các chương trình ứng dụng không phải làm các công việc đơn giản và cơ bản nữa. Hệ điều hành sẽ dựa chủ yếu vào BIOS để liên lạc với thiết bị ngoại vi được nối với hệ thống PC.

Các chương trình dịch vụ được phân bố trong Rom BIOS đều có chung địa chỉ segment là F000H, do đó các địa chỉ của các vùng nhớ mô tả dưới đây là các địa chỉ offset của nó.

Từ 0000h ÷ 0015h: Các vùng dữ liệu của BIOS.

Từ 0017h ÷ 003Eh: Các vùng dữ liệu bàn phím.

Từ 003Eh ÷ 0049h: Các vùng dữ liệu đĩa.

Từ 0049h ÷ 0066h: Vùng dữ liệu màn hình.

Từ 0067h ÷ 006Bh: Vùng dữ liệu Cassette.

Từ 006Ch ÷ 0070h: Vùng dữ liệu thời gian.

Từ 0071h ÷ 0072h: Vùng dữ liệu hệ thống.

Từ 0074h ÷ 0076h: Vùng dữ liệu đĩa cố định.

Từ 0078h ÷ 007Ch: Vùng dữ liệu máy in và RS232.

Từ 0080h ÷ 0082h: Vùng dữ liệu mở rộng của bàn phím.

- ***E018h ÷ E0AEh***

Chứa chương trình thực hiện việc đọc/ ghi bộ nhớ và kiểm tra từng khối 16KB của bộ nhớ.

Vào:

ES = địa chỉ bắt đầu của mảng bộ nhớ được kiểm tra.

DS = địa chỉ bắt đầu của mảng bộ nhớ được kiểm tra.

Ra:

Cờ ZF = 0 Nếu bộ nhớ lỗi.

AL = 0 biểu thị kiểm tra chẵn lẻ.

AX, BX, CX, DX, SI, DI bị phá hủy nội dung cũ.

- ***E0AEh ÷ E0D1h***

Chứa chương trình tổng kiểm tra ROS I (Read Only Test). Tổng kiểm tra được thực hiện đối với Module 8K gồm P00 và BIOS.

- ***E0D3h ÷ E149h***

Chứa chương trình thiết lập trạng thái ban đầu cổng thanh ghi kiểm tra đối với chíp DMA 8237. Làm mất khả năng của bộ điều khiển truy nhập trực tiếp DMA 8237. Kiểm tra thời gian hoàn thành một chức năng. Ghi/ đọc các luồng địa chỉ và các thanh ghi đếm từ cho tất cả các kênh. Gán giá trị ban đầu và khởi động DMA để làm tươi bộ nhớ.

- ***E148h ÷ E1B2h***

Chứa chương trình kiểm tra bộ nhớ đọc ghi cơ sở 16K. Ghi/ đọc/ kiểm tra các kiểu dữ liệu FF, 55, AA, 01 và 00 đến 16K thứ nhất của bộ nhớ. Địa chỉ kiểm tra được thiết lập ban đầu của bộ nhớ gán cho chíp điều khiển 8259 để kiểm tra sự hoạt động của chế độ kiểm tra thứ 2.

- ***E1B4h ÷ E1C2h***

Chứa chương trình thiết lập giá trị ban đầu cho chíp điều khiển 8259.

- ***E1C4h ÷ E211h***

Chứa chương trình kiểm tra sự hoạt động của chế độ kiểm tra 2 bằng cách nạp chương trình kiểm tra từ bàn phím.

- ***E21Ah ÷ E23Eh***

Chứa chương trình kiểm tra chíp điều khiển ngắt 8259. Đọc/ ghi mặt nạ ngắt thanh ghi (IMK) bằng các số 1, 0 với các ngắt hệ thống có thể. Mặt nạ là cách tắt các ngắt, kiểm tra các ngắt nóng (không mong muốn).

- ***E23Fh ÷ E2ABh***

Chứa chương trình của bộ đếm thời gian 8253. Kiểm tra thời gian của hệ thống (0). Không đếm quá nhanh hoặc quá chậm.

- ***E2ADh ÷ E31Ch***

Chứa chương trình thiết lập giá trị ban đầu và khởi động bộ điều khiển CRT 6845. Kiểm tra đọc/ ghi bộ nhớ màn hình. Thiết lập lại tín hiệu kích hoạt Video. Chọn lựa chế độ gồm có cả ký tự và số, 40x25, B & W. Đọc/ ghi các kiểu dữ liệu đến STG, kiểm tra khả năng định vị STG.

- ***E31Eh ÷ E32Dh***

Chứa chương trình đặt dữ liệu Video trên màn hình đối với việc kiểm tra dòng Video. Kích hoạt tín hiệu Video và thiết lập chế độ.

- ***E32Eh ÷ E380h***

Chứa chương trình kiểm tra các đường giao diện CRT.

- ***E382h ÷ E31Ch***

Chứa chương trình kiểm tra cổng I/ O mở rộng. Việc kiểm tra được thực hiện đến khi nào gặp cổng mở rộng nếu nó được cài đặt. Kiểm tra dữ liệu và các kênh địa chỉ đến cổng I/ O.

- ***E3C4h ÷ E439h***

Chứa chương trình kiểm tra bộ nhớ đọc/ghi mở rộng.

- ***E43Bh ÷ E481h***

Chứa chương trình kiểm tra bàn phím. Khởi động bàn phím và kiểm tra mã quét 'AA' được quay trở lại CPU. Kiểm tra các phím dính cứng.

- ***E483h ÷ E4D9h***

Chứa chương trình kiểm tra bộ lọc dữ liệu cassette chuyển sang cắt cassette monitor, ghi một bit ra BVS dữ liệu cassetee kiểm tra dữ liệu cassetee đọc trong một vùng cho phép.

- ***E4BCh ÷ E4D9h***

Chứa chương trình kiểm tra đối với ROM tùy chọn từ C8000C → F4000 trong 2KB gia số (Một Module cho phép có '55AA' trong hai trường đầu. Độ dài chỉ thị trong trường 3 và mã Test/Init bắt đầu trong trường thứ 4).

- ***E4DCh ÷ E4F0h***

Chứa chương trình tổng kiểm tra ROS II. Một tổng kiểm tra được thực hiện cho 4 Modules ROS chứa mã ở cơ sở.

- ***E4F1h ÷ E5CDh***

Chứa chương trình kiểm tra đĩa lắp thêm. Kiểm tra IPL (Information Processing Language) của đĩa được lắp với hệ thống. Nếu đã được lắp, kiểm tra trang thái của NEC FDC sau một lần seset, tiến hành gọi lại và phát lệnh tới đĩa mềm và kiểm tra trạng thái, hoàn thành sự khởi động ban đầu của hệ thống sau đó qua điều khiển để nạp chương trình mới.

- ***E5CFh ÷ E624h***

Chứa thường trình phụ-kiểm tra khả năng khởi tạo. Chường trình này sẽ phát một tần số dài (3s) và một hoặc nhiều hơn tần số ngắn (1s) để chỉ thị một lỗi trên Mainboand, một lỗi trên Module RAM hoặc một vấn đề với CRT.

- ***E625h ÷ E643h***

Chứa chương trình chuyển đổi và in mã ASCII.

AL phải chứa số cần chuyển đổi

AX, BX mất hiệu dụng

- ***E643h ÷ E66Ch***

Chứa chương trình giải một phần mềm reset tới bàn phím.

Mã quét 'AA' sẽ được quay trở lại CPU.

- ***E66Dh ÷ E689h***

Chứa chương trình chớp LED và các kiểm tra MFG Burn-in và Run-in. Nếu LED sáng nó trở nên tối, nếu tối trở nên sáng.

- ***E6BAh ÷ E6E3h***

Chứa chương trình sẽ in một dòng lệnh lên màn hình.

SI = offset của bộ đệm lệnh.

CX = số byte đếm lệnh.

Lệnh có độ dài tối đa là 36 ký tự.

2.4.2. Phân bố các vector ngắt trong RomBIOS

Các chương trình con phục vụ ngắt (CTCPV Ngắt) trong ROMBIOS được tổ chức tại các vùng bộ nhớ có địa chỉ xác định.

- ***E6E4h ÷ E726h: CTCPV Ngắt 19***

Nạp chương trình mồi (Bootstrap).

Véc tơ ngắt này được lấy sau khi POST thực hiện xong để nạp một đoạn mã đặc biệt là chương trình "bootstrap" trên đĩa mềm hay đĩa cứng. Nếu một đĩa mềm 5"1/4 là hợp lệ trên hệ thống, dữ liệu trong sector 1 của trach 0 được nạp vào vùng mới ở địa chỉ tuyệt đối 0: 07C00h và điều khiển được trao cho chương trình ở địa chỉ này. Nếu không có đĩa mềm hoặc nếu đĩa cứng bị lỗi thì điều khiển được trỏ đến điểm vào của chương trình thường trú cơ sở.

- ***E729h ÷ E82Ch - CTCPV Ngắt 14 (RS232-I0)***

Chương trình này cung cấp chuỗi byte đến cổng liên lạc theo các tham số.

AH = 0 khởi tạo cổng truyền thông.

AL chứa các tham số khởi tạo.

d7d6d5	Tốc độ
0 0 0 =	110
0 0 1 =	150
0 1 0 =	300
0 1 1 =	600
1 0 0 =	1200
1 0 1 =	2400
1 1 0 =	4800
1 1 1 =	9600

d4d3	parity
x 0 =	None
0 1 =	lẻ

d 2	bit STOP	d1d0	độ dài
0 = 1	1	0	= 7bit
1 = 2	1	1	= 8 bit

AH = 1 Gửi ký tự trong AL ra cổnh RS-232 đã chọn.

Khi thoát bit 7 của AH được thiết lập nếu không thể chuyển byte dữ liệu trên dường truyền.

Nếu bit 7 của AH không được thiết lập, nội dung của AH được thiết lập phản ánh trạng thái hiện tại của đường truyền.

AH = 2 Nhận một ký tự từ cổng RS-232 đã chọn. Khi thoát, AH là trạng thái đường dây hiện tại.

AH có bit 7 =1 sẽ ứng với Time out.

AH = 3 Lấy trạng thái cổng truyền thông (chứa trong AX).

AH = chứa dòng trạng thái (tình trạng đường điều khiển).

Bit 7 = Time out (hết thời gian).

Bit 6 = Thanh ghi dịch phát rỗng.

Bit 5 = Thanh ghi phát rỗng.

Bit 4 = Phát hiện ngắt quãng.

Bit 3 = lỗi khung.

Bit 2 = Lỗi chẵn lẻ.

Bit 1 = Lỗi chạy quá tốc độ.

Bit 0 = dữ liệu sẵn sàng.

AL chứa trạng thái MODEM.

Bit 7 = Đã phát hiện được tín hiệu thu.

Bit 6 = Chuông chỉ báo.

Bit 5 = DSR (Dữ liệu được thiết lập sẵn sàng).

Bit 4 = CTS.

Bit 3 = Phát hiện tín hiệu thu Delta.

Bit 2 = phát hiện chuông.

Bit 1 = Dữ liệu Delta đã sẵn sàng.

Bit 0 =CTS - kiểu Delta.

DX = Các tham số của card RS-232.Vùng dữ liệu RS_232 cơ sở chứa địa chỉ cơ sở của 8250 trên card. Vùng 400h chứa tới 4 địa chỉ của RS-232.

Vùng dữ liệu nhãn RS-232-TIM-OUT (byte) chứa bộ đếm giá trị cho TIME_OUT (Defaul = 1).

- ***E82Eh ÷ EC49h - CTCPV Ngắt 16H***

Vào/ ra bàn phím.

Đây là một giao diện ở mức ứng dụng đối với bàn phím.

AH = 0 Đọc ký tự ASCII tiếp theo gõ từ bàn phím.

AL = ký tự ASCII (nếu AL = 0, AH là một phím ASCII mở rộng).

AH = Mã quét hay phím ASCII mở rộng.

AH = 1 Đặt cờ ZF để chỉ thị nếu một ký tự ASCII chuẩn bị đọc là hợp lệ.

ZF = 1 Mã không hợp lệ.

ZF = 0 Mã hợp lệ.

AH = 2 Đọc trạng thái phím Shiff. Xác định các phím Shiff nào đang được ấn và bàn phím có ở trạng thái Numlock hay không...

AL = Phím shiff và tình trạng 'khoá' như trong các cờ hiệu bàn phím.

AH = 3 Đặt tốc độ gõ phím từ động và thời trễ.

Vào: AL = 05h

BL = Tốc độ gõ tự động 0 = 30 lần/sec, 1=26...18H=2.

BH = Thời trễ (0=250ms, 1=500ms, 2=750ms,3=1s)

Ra: (không).

AH = 4 (chưa dùng).

AH = 5 Đặt một phím vào bộ đệm bàn phím.

Vào: CL = Ký từ ASCII

CH = Byte mã quét (hoặc 0 nếu ta không quan tâm tới)

Ra: AL = Tình trạng: 0 = thành công; 1 = đầy bộ đệm

AH = 06H ÷ 0fH (chưa dùng)

AH = 10h, 11h, 12h, tương tự như trên nhưng dùng chuyen vào bàn phím 101 phím.

Ra: Chỉ thanh ghi AX và các cờ bị thay đổi.

Các thanh ghi khác được bảo toàn.

- ***EC4Ch ÷ EC58h***

Chứa chương trình phụ cho chức năng tổng kiểm tra ROS.

- ***EC59h ÷ EFD1h - CTCPV Ngắt 13H***

Vào/ ra đĩa.

Đây là giao diện bảo đảm truy nhập tới các bộ điều hợp đĩa mềm/ cứng

Vào:

AH = 0 Khởi động hệ thống đĩa. Khởi động lại đĩa cứng, chuẩn bị lệnh, gọi các yâu cầu mở tất cả các đĩa.

AH = 1 Đọc trạng thái của hệ thống trong AL.

Trạng thái đĩa từ phép toán (thao tác) cuối cùng được sử dụng các thanh ghi để đọc/ghi/kiểm tra và định dạng.

DL = Số hiệu ổ đĩa (0-3 được cho phép, giá trị được kiểm tra).

DH = Số hiệu đầu dọc (0-1được cho phép, giá trị không được kiểm tra).

CH = Số hiệu track (0-39, không giá trị nào được kiểm tra).

CL = Số hiệu sector (1-8, không giá trị náo được kiểm tra).

AL = Số hiệu của các sector (cực đại bằng 8).

ES: BX = Địa chỉ của bộ đệm (không yêu cầu kiểm tra).

AH = 2 Đọc các sector mong muốn trong bộ nhớ.

AH = 3 Ghi các sector mong muốn trong bộ nhớ.

AH = 4 Kiểm tra các sector mong muốn.

AH = 5 Định dạng các track mong muốn.

Để thực hiện việc định dạng, con trỏ (ES:BX) cần phải trỏ đến tập hợp địa trỉ các file mong muốn đối với các rãnh. Mỗi file bao gồm 4 byte (C, H, R, N), ở đây C = số track, H = số đầu đọc, R = số sector, N = số của các byte sector dự doán trước (00 = 128, 01 = 256, 02 = 512, 03 = 1024). Cần phải nhập một lần cho mọi sector trên một rãnh. Thông tin này được sử dụng để tìm sector yêu cầu trong suốt quá trình truy nhập đọc/ ghi.

Biến dữ liệu - con trỏ đĩa.

Con trỏ 2 word được thiết lập cho các tham số đĩa.

Ra:

AH = Trạng thái của thao tác.

CY = 0 thao tác thành công

CY = 1 thao tác bị lỗi (AH có lỗi cú pháp)

Đối với chức năng đọc/ ghi/ kiểm tra:

DS, BX, DX, CH, CL được bảo vệ.

AL = số các sector đọc thực tế. Lưu ý AL có thể không đúng nếu xảy ra lỗi TIME_OUT.

Chú ý: Nếu một lỗi được thông báo bởi mã đĩa, một tác động thích hợp sẽ khởi động lại đĩa.

- ***EFD2h ÷ F041h - CTCPV Ngắt 17H***

Chương trình này dùng để điều khiển máy in.

Vào:

AH = 0 In các ký tự trong AL.

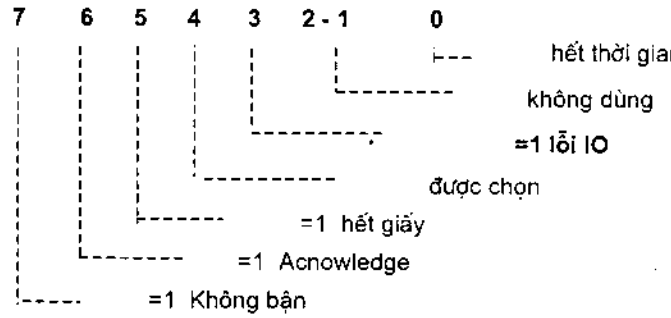
AH = 1 Nếu ký tự không in được.

Các bit khác đặt như trạng thái gọi bình thường.

AH = 1 Khởi tạo một cổng máy in.

AH chứa trạng thái máy in.

AH = 2 Đọc trạng thái máy in trong AH.



- ***F045h ÷ F839h CTCPV Ngắt 10h (Video_I/O)***

Chường trình này thực hiện việc vào ra màn hình và đặt chế độ màn hình.

Các chương trình phục vụ giao diện theo các chức năng phục vụ sau:

AH = 0 Đặt chế độ màn hình, (AL) chứa giá trị chế độ.

AL = 0	40x25	Đen trắng
AL = 1	40x25	Màu
AL = 2	80x25	Đen trắng
AL = 3	80x25	Màu

Các chế độ đồ hoạ

AL = 4	320x200	Màu
AL = 5	320x200	Đen trắng
AL = 6	640x200	Đen trắng

AH = 1 Định kích thước (dạng con trỏ).

CH = các bit 4 - 0 = dòng bắt đầu đối với con trỏ.

CL = các bit 4 - 0 = dòng cuối của con trỏ.

AH = 2 Đặt vị trí con trỏ.

(DH, DL) = hàng, cột trên cùng bên trái.

BH = số trang (tính từ 0 đối với các chế độ đồ hoạ).

AH = 3 Đọc vị trí con trỏ.

BH = số trang (tính từ 0 đối với các chế độ đồ hoạ).

Ra:

(DH, DL) = hàng, cột của con trỏ hiện thời.

(CH, CL) = đạt mã con trỏ hiện thời.

AH = 4 Đọc vị trí bút sáng.

Ra:

AH = 0 bút sáng chuyển mạch chưa bật sáng.

AH = 1 trả về các giá trị bút sáng hợp lệ trong thanh ghi

(DH, DL) = hàng, cột của ký tự.

CH = mành sáng (0 - 199).

BX = cột của điểm sáng (0 - 319, 693).

AH = 5 Chọn trang màn hình (chỉ dùng cho chế độ văn bản).

AL = trang (0 - 7 đối với các mode 0 và 1, 0 - 3 đối với các mode 2 và 3).

AH = 6 Cuộn trang màn hình lên.

AL = Số dòng cần cuộn lên. AL = 0 có nghĩa xóa toàn bộ cửa sổ.

CH = hàng của góc trên cùng bên trái của cửa sổ.

CL = cột của góc trên cùng bên trái của cửa sổ.

DH = hàng của góc trên cùng bên phải của cửa sổ.

DL = cột của góc trên cùng bên phải của cửa sổ.

BH = Thuộc tính màn hình.

AH = 7 Cuộn trang màn hình xuống.

AL = số dòng được cuộn. AL = 0 có nghĩa xóa toàn bộ cửa sổ.

CH = hàng của góc trên cùng bên trái của cửa sổ.

CL = cột của góc trên cùng bên trái của cửa sổ.

DH = hàng của góc trên cùng bên phải của cửa sổ.

DL = cột của góc trên cùng bên phải của cửa sổ.

BH = Thuộc tính màn hình.

AH = 8 Đọc thuộc tính ký/ tự tại vị trí hiện thời của con trỏ.

BH = trang màn hình (chỉ đúng với chế độ Text)

Ra:

AL = ký tự đọc.

AH = thuộc tính của ký tự đọc (chỉ đúng với chế độ Text).

AH = 9 In thuộc tính/ ký tự tại vị trí hiện thời của con trỏ.

BH = trang màn hình hiển thị (chỉ đúng với chế độ Text).

CX = số các ký tự để ghi.

AL = ký tự để ghi ra.

BL = thuộc tính của ký tự (trong chế độText) hay màu của ký tự (trong chế độ đồ hoạ)

AH = 10 ghi ký tự ở vị trí hiện thời của con trỏ.

BH = trang màn hình hiển thị (chỉ đúng với chế độ Text).

CX = đếm các ký tự được ghi.

AL = ký tự cần ghi.

Đối với việc đọc / ghi ký tự khi giao diện đang ở trong chế độ đồ hoạ. Các ký tự được thực hiện từ một ký tự ảnh, trong ROM hệ thống chỉ chứa 128 ký tự đầu tiên. Để đọc / ghi 128 ký tự tiếp theo người sử dụng cần phải thiết lập vị trí ban đầu của con trỏ tại ngắt 1FH (vùng 0000H) để chỉ đến bảng 1KB chứa mã của 128 ký tự thứ 2 (128 - 255).

Đối với việc ghi ký tự trong chế độ đồ hoạ. Số lần lặp lại được chứa trong thanh ghi CX, trên đầu vào sẽ chỉ đưa ra các kết quả hợp lệ đối với các ký tự được chứa trong cùng một ROM. Sử mở rộng đến các vùng kế tiếp sẽ không đúng.

Giao diện đồ họa

AH = 11 Đặt bảng màu.

BH = Bảng màu IO sẽ được đặt (0 - 127).

BL = giá trị màu sẽ được sử dụng cùng với màu IO.

Chú ý: Đối với card màu hiện thời, điểm vào này chỉ có nghĩa đối với đồ họa 320x200.

Màu BH = 1 chọn màu nền (0 - 15).

BH = 0 = Green(1) / Red(2) / Yellow(3).

= 1 = Cyan(1) / Magenta(2) / White(3).

Trong chế độ văn bản 40x25 hoặc 80x25, giá trị 0 của bảng màu chỉ được sử dụng cho màu viền (giá trị nằm trong khoảng từ 0 - 31, ở đây giá trị từ 16 - 31chọn đối với độ sáng nền cao).

AH = 12 Ghi chấm sáng đồ thị.

DX = số hiệu hàng.

CX = số hiệu cột.

AL = giá trị màu.

Nếu bit 7 của AL = 1 thì giá trị màu là cao nhất hoặc nội dung dòng hiện thời của chấm sáng.

AH = 13 Đọc chấm sáng.

DX = số hiệu hàng.

CX = số hiệu cột.

AL = quay trở về đọc chấm sáng.

AH = 14 Ghi ký tự vào trang màn hình hoạt động.

AL = ký tự được ghi.

BL = màu chữ trong chế độ đồ hoạ.

Chú ý: Độ rộng màn hình được điều chỉnh bởi chế độ đặt trước.

AH = 15 Trạng thái màn hình hiện thời.

AL = Mode hiện thời.

AH = số cột ký tự trên màn hình.

BH = trang màn hình hoạt động hiện thời.

- ***F840h ÷ F856h CTCPV Ngắt 11H***

Kiểm tra thiết bị hệ thống.

Ngắt 11h trả về các bit cờ trong AX chỉ ra các thiết bị nào được cài đặt hoặc đang hoạt động.

- ***F859h ÷ FE66h CTCPV Ngắt 15H***

Các phục vụ mở rộng.

Ngắt 15h chứa nhiêù chức năng chuyên dùng như:

AH = 0 motor Cassette chạy (ON).

AH = 1 motor Cassette dừng (OFF).

AH = 2 Đọc một hoặc nhiều khối 256 byte từ Cassette.

AH = 3 Ghi một hoặc nhiều khối 256 byte từ Cassette.

Ngoài ra nó còn có các chức năng: Mở thiết bị, đóng thiết bị, kết thúc chương trình, đợi biến cố, Joystick, phím SysReq, wait, di chuyển vùng nhớ mở rộng, lấy kích thước vùng nhớ mở rộng, chuyển sang Mode ảo, vòng lặp thiết bị bận, hoàn tất ngắt.

- ***FE6Eh ÷ FF53h CTCPV Ngắt 1Ah (Time I/ O)***

Phục vụ này cung cấp các truy xuất đến đồng hồ hệ thống BIOS của PC làm việc với một "nhịp đếm" cho mỗi khoảng thời gian 55ms tính từ lúc bắt đầu bật máy hoặc reset. BIOS cũng cung cấp truy xuất đến các giá trị của đồng hồ thời gian được cập nhật đều đặn và được cất trong vùng nhớ CMOS.

Vào:

AH = 0 Đọc đồng hồ (nhịp đếm).

CX, DX = nhịp đếm kể từ khi reset.

CX là cao, DX là thấp

AL = 0 nếu bộ định thời không bị tràn sau 24 giờ kể từ lần đọc cuối cùng và AL<> 0 nếu chuyển sang ngày khác.

AH = 1 Thiết lập tốc độ đồng hồ (nhịp đếm).

CX, DX = nhịp đếm, CX là cao, DX là thấp.

Chú ý: Đồng hồ cập nhập với tốc độ 1193180/ 65536 (nhịp/sec). Trong phục vụ nàychứa chương trình điều khiển bộ định thời ngắt từ kênh 0 của bộ định thời 8253, vào tần số là 119318 MHz và hệ số chia là 65536, kết quả xấp xỉ 18,2 ngắt mỗi giây, trong suốt quá trình mở máy có các vector ngắt chuyển vào trong vùng ngắt của 8086. Chỉ các

OFFSET được hiển thị ở đây còn mã SEGMENT sẽ được cộng vào tất cả chúng trừ vùng cấm.

- ***FF54h ÷ FFCAh CTCPV Ngắt 05h (In màn hình)***

Ngắt 05H được sử dụng trong PC để thực hiện một chương trình trong ROM BIOS để đưa nội dung màn hình ra máy in. Nó được gọi trực tiếp từ ngắt bàn phím INT 09H khi có một phím PtrScr (hoặc PrintScreen) được ấn. Nó cũng có thể được gọi bởi chương trình phần mềm và ta có thể can thiệp vào ngắt này nếu ta muốn viết một chương trình riêng để chuyển nội dung màn hình ra máy in.

Địa chỉ	50: 0		Chứa trạng thái của màn hình máy in.
	50: 0	= 0	Có nghĩa là hoặc máy in không được gọi hoặc là được gọi trở lại từ một cuộc gọi chỉ thị rằng nó đang bị chiếm bởi một đối tượng khác.
		= 1	In màn hình ở trạng thái hoạt động.
		= 255	Lỗi.

2.4.3. CTCPV Ngắt 14H và các phục vụ của nó

CTCPV Ngắt 14H của BIOS là chương trình phục vụ cho việc khởi tạo các cổng truyền thông, cho phép truy xuất đến các cổng truyền thông với chuẩn RS_232. Việc lập trình để truyền thông giữa các máy tính thông qua các MODEM sẽ đơn giản hơn rất nhiều nếu ta biết sử dụng ngắt 14H của BIOS. Để có thể ứng dụng được ngắt 14H vào việc lập trình điều khiển truyền thông, trước tiên chúng ta phải hiểu được chương trình nguồn ASM của ngắt 14H trong BIOS được tổ chức như thế nào, nó thực hiện những nhiệm vụ gì, việc khởi tạo cổng truyền thông cũng như việc ghi đọc dữ liệu qua cổng truyền thông sẽ được thực hiện như thế nào khi lệnh INT 14H được gọi.

Chương trình con phục vụ ngắt 14H được viết bằng Assembly có dạng sau:

```
ASSUME   CS:Code, DS:Data
ORG 0E729H
A1   Label  WORD          ; Bảng giá trị khởi tạo tốc độ baud
          DW       1047   ; 110
          DW       768    ; 150
          DW       348    ; 300
          DW       192    ; 600
          DW       96     ; 1200
          DW       48     ; 2400
          DW       24     ; 4800
          DW       12     ; 9600
RS232_IO    PROC    FAR
     STI; trả lại các ngắt
    PUSH    DS      ; cất các tham số hiện thời trong cac
    PUSH    DX      ; thanh ghi DS, DX, SI, DI, CX, BX
    PUSH    SI
    PUSH    DI
    PUSH    CX
     PUSH   BX
     MOV    SI, DX ; chuyển giá trị RS232 vào SI
     MOV    DI, DX
     SHL    SI, 1
     CALL   DDS
     MOV    DX, RS232_BASE[SI]    ; DX chứa địa chỉ cơ
                                  ; sở của RS232 (3F8h)
     OR     DX, DX ; cổng truyền thông có hay không?
     JZ     A3     ; Không có quay về chương trình chính
     OR    AH, AH  ; Có, kiểm tra AH = 0 ?
     JZ     A4     ; AH = 0 khởi tạo cổng truyền thông
     DEC    AH     ; Kiểm tra AH = 1 ?
    JZ      A5     ; AH = 1, gửi ký tự từ (AL) ra cổng
     DEC    AH     ; Kiểm tra AH = 2 ?
     JZ     A12    ; AH = 2, nhận ký tự từ cổng vào (AL)
```

```
  A2:
     DEC     AH       ; Kiểm tra AH = 3 ?
     JNZ     A3       ; không, thoát khỏi chương trình
     JMP     A18      ; AH = 3, ghi lại tình trạng cổng
  A3:                 ; Thoát trở về chương trình chính
     POP     BX       ; trả lại các tham số ban đầu cho
     POP     CX       ; các thanh ghi
     POP     DI
     POP     SI
     POP     DX
     POP     DS
     IRET
;------- khởi tạo cổng truyền thông ------------------;
  A4:
     MOV     AH, AL   ; chuyển các tham số khởi tạo vào AH
     ADD     DX, 3    ; DX chứa OFFSET của thanh ghi LCR
                      ; của UART 8250
     MOV     AL, 80H  ; Bit DLAB = 1 cho phép truy nhập bộ
     OUT     DX, AL   ; chia để tính tốc độ Baud
;------- xác định tốc độ truyền dữ liệu -------------;
     MOV     DL, AH   ; chuyển các tham số khởi tạo vào DL
     MOV     CL, 4
    ROL      DL, CL
    AND      DL, 0EH  ;tách lấy các bit xác định tốc độ Baud
     MOV     DI, OFFSET A1    ; DI chứa OFFSET bảng hệ số chia
     ADD     DI, DX
     MOV     DX, RS232_BASE[SI]
     INC   DX         ; DX chứa OFFSET của thanh
                      ; ghi chốt / bộ chia phần cao (MSB)
        MOV     AL, CS:[DI]+1 ; lấy phần cao của hệ số chia
        OUT     DX, AL        ;MSB chứa phần cao của hệ số chia
        DEC     DX            ; DX chứa OFFSET của thanh ghi
                              ; chôt/bộ chia phần thấp (LSB)
```

```
        MOV     AL, CS:[DI]     : lấy phần thấp của hệ số chia
        OUT     DX, AL          ;LSB chứa phần thấp của hệ số chia
        ADD     DX,  3          ;DX chứa OFFSET của thanh ghi LCR
        MOV     AL, AH     ;trả lại các tham số khởi tạo cho AL
        AND     AL, 01FH   ; loại bỏ các bit cao
                           ; (các bit tốc độ Baud)
        OUT     DX, AL     ; LCR chứa tham số khởi tạo về số bit
                           ; trong 1 từ, parity, số bit stop
        DEC     DX
        DEC     DX         ; OFFSET thanh ghi IER (vì DLAB = 1)
        MOV     AL, 0
        OUT     DX, 0      ;cấm tất cả các ngắt có thể (vì đang
                           ;thực hiện khởi tạo cổng truyền thông
        JMP     SHORT      A18
;-------------------------------------------------------;
;------ gửi ký tự trong (AL) ra cổng truyền thông ----;
;-------------------------------------------------------;
        PUSH    AX         ; cất ký tự cần truyền vào ngăn xếp
        ADD     DX,4       ; OFFSET thanh ghi đk MODEM (MCR)
        MOV     AL,3
        OUT     DX,AL      ; đặt bít DTR = 1 và RTS = 1
        INC     DX
        INC     DX         ; OFFSET thanh ghi trạng thái MODEM
        MOV     BH, 30H; DCE sẵn sàng (DTR) và sẵn sàng
                           ; nhận CTS
        CALL    WAIT_FOR_STATUS; DTR = 1 ? và CTS = 1 ?
        JE      A9         ; Đúng, nhảy đến A9
    A7:
      POP     CX           ; Không, trả ký tự cần gửi về CX
        MOV     AL,CL      ; trả ký tự về AL
    A8:
        OR      AH,80H
        JMP     A3         ; thoát khỏi chương trình
```

```
A9:
   DEC    DX          ;OFFSET thanh ghi trạng thái đường dây
A10:
   MOV  BH, 20H       ; đường truyền đã sẵn sàng
                      ; (bit THRE = 1)
   CALL   WAIT_FOR_STATUS     ; kiểm tra bit THRE = 1 ?
   JNZ   A7           ;Không, hủy bỏ việc truyền dữ liệu A7
A11:
   SUB    DX, 5       ; DX chứa địa chỉ cơ sở của cổng
                      ; truyền thông
   POP   CX
   MOV    AL, CL      ; lấy lại ký tự cần gửi
   OUT    DX, AL      ; thực hiện gửi
   JMP   A3           ; thoát
;-----------------------------------------------------;
;---------------nhận một ký tự từ đường truyền--------;
;-----------------------------------------------------;
A12:
   ADD    DX, 4       ; OFFSET thanh ghi đk MODEM (MCR)
   MOV    AL, 1
   OUT    DX, AL      ; thiết bị đầu cuối sẵn sàng
                      ; (bit DTR = 1)
   INC   DX
   INC   DX           ; OFFSET thanh ghi trạng thái MODEM
A13:
   MOV    BH, 20H     ; DCE sẵn sàng (bit DSR = 1)
   CALL  WAIT_FOR_STATUS      ; kiểm tra xem DSR = 1 ?
   JNZ   A8           ; không,  hủy bỏ việc nhận dữ liệu
A15:
   DEC    DX          ; DSR= 1, OFFSET thanh ghi trạng thái
                      ; đường dây
   A16:
     MOV    BH, 1          ; đã nhận được một ký tự
                           ; (bit R_DR = 1)
```

```
        CALL    WAIT_FOR_STATUS
                                ; kiểm tra xem bit RxDR = 1?
        JNZ     A8      ;RxDR=1, đã nhận dữ liệu và thoát ra
      A17:
        AND     AH, 00011110B           ; sai, kiểm tra lỗi
        MOV     DX, RS232_BASE[SI]
                                ; OFFSET thanh ghi đệm thu
        IN      AL, DX          ; chuyển ký tự thu vào AL
        JM      A3              ; thoát
;-------------------------------------------------;
;---------- Trạng thái cổng truyền thông ----------;
;-------------------------------------------------;
      A18:
        MOV     DX, RS232_BASE[SI]
        ADD     DX, 5
        IN      AL, DX
        MOV     AH, AL
        INC     DX
        IN      AL, DX
        JMP     A3
;-------------------------------------------------      ;
; Đợi kiểm tra trạng thái                               ;
; Vào:                                                  ;
;         BH = bit trạng thái cần kiểm tra              ;
;         DX = địa chỉ thanh ghi trạng thái             ;
; Ra:                                                   ;
;         Cờ  ZF = 1 tương ứng trạng thái tìm được      ;
;         ZF = 0   Time_out                             ;
;         AH = Trạng thái cuối cùng                     ;
;-------------------------------------------------;
      WAIT_FOR_STATUS     PROC    NEAR
        MOV     BL,RS232_TIM_OUT[DI]
      WFS0:
```

```
    SUB     CX, CX
WFS1:
    IN      AL, DX
    MOV     AH, AL
    AND     AL, BH
    CMP     AL, BH
    JE      WFS_END
    LOOP    WFS1
    DEC     BL
    JNZ     WFS0
    OR      BH, BH
WFS_END:
    RET
WAIT_FOR_STATUS       ENDP
RS232_IO        ENDP
```

Tùy thuộc vào giá trị do người lập trình đặt trong thanh ghi AH mà chương trình sẽ thực hiện các chức năng khác nhau:

AH = 00H Thực hiện việc khởi tạo cổng truyền thông.

AH = 01H Thực hiện việc gửi dữ liệu ra cổng truyền thông.

AH = 02H Thực hiện thu dữ liệu từ cổng truyền thông.

AH = 03H Thực hiện việc thu trạng thái của cổng truyền thông.

Hoạt động của máy tính khi lệnh INT14H được gọi thì CPU của máy tính sẽ dừng tất cả các chương trình ứng dụng đang chạy, cất nội dung các thanh ghi hiện thời vào STACK để quay ra phục vụ cho ngắt 14H. Con trỏ CS:IP sẽ có nội dung là F0000h: 0E729h đây chính là địa chỉ bắt đầu của vùng nhớ chứa chương trình phục vụ ngắt 14H.

Địa chỉ cơ sở của cổng truyền thông dùng chuẩn RS232 sẽ được đưa vào thanh ghi SI và DI. Máy tính sẽ kiểm tra xem có cổng truyền thông chuẩn RS232 nào được kết nối hay không, nếu không tìm thấy nó sẽ thoát ra khỏi chương trình phục vụ ngắt và trở về chương trình chính, nếu tìm thấy có cổng truyền thông chuẩn RS232 thì nó sẽ tiếp tục kiểm

tra xem phục vụ ngắt nào của INT14H được gọi đến (AH = 00; AH = 01; AH = 02; AH = 03). Sau đây ta sẽ xem xet một cách cụ thể các phục vụ ngắt được đáp ứng như thế nào.

- ***Khi AH = 00H***

Khi kiểm tra thấy AH = 00H thì máy tính sẽ thực hiện chức năng khởi tạo cổng truyền thông RS232. Tất cả các tham số khởi tạo như: tốc độ truyền số liệu, số bit trong 1 từ mã, số bit stop, có sử dụng mã kiểm tra chẵn lẻ (parity) hay không đều được đặt theo ý muốn của người lập trình và được chứa trong thanh ghi AL.

Đầu tiên các tham số khởi tạo được cất vào thanh ghi AH, bit DLAB (D_7) của thanh ghi điều khiển đường dây được đặt bằng 1 để truy cập bộ chia tần số. Tùy thuộc vào ta định truyền số liệu với tốc độ là bao nhiêu mà ta xác định hệ số chia cho phù hợp, phần cao của hệ số chia sẽ được chứa trong thanh ghi chốt / bộ chia phần cao MSB - có địa chỉ offset là 1, phần thấp của hệ số chia sẽ được chứa trong thanh ghi chôt / bộ chia phần thấp LSB - có địa chỉ offset là 0. Sau khi xác lập xong tốc độ Baud bít DLAB được trả về 0 và các tham số khởi tạo lại được trả về thanh ghi AL từ thanh ghi AH. Tiếp theo các tham số này sẽ được chuyển vào thanh ghi điều khiển đường dây LCR - có địa chỉ offset là 3. Trong quá trình khởi tạo tất cả các bit của thanh ghi cho phép ngắt IER – có địa chỉ offset là 1 (với bit DLAB lúc này = 0) - đều dặt về 0 thực hiện cấm tất cả các ngắt có thể có vì đang thực hiện khởi tạo. Cuối cùng trạng thái của việc truyền tín hiệu trên đường dây như: phát hiện Break (dòng tin bị ngắt), các lỗi máy thu (như tràn khung, chẵn lẻ, tràn số, số liệu không sẵn sàng cà trạng thái không có số liệu truyền TxE) trong thanh ghi traịng thái đường truyền LSR – có địa chỉ offset là 5 của UART 8250 được ghi vào thanh ghi AL, còn trạng thái hiện thời của các tín hiệu diều khiển MODEM từ đường dây trong thanh ghi trạng thái MODEM MSR- có địa chỉ offset là 6 sẽ được ghi vào trong thanh ghi AH và quá trình khởi tạo kết thúc.

Tóm lại sau khi khởi tạo xong thì giá trị các thanh ghi là:

- Thanh ghi chốt / bộ chia phần thấp LSB và phần cao MSB chứa hệ số chia xác định tốc độ Baud

- Thanh ghi điều khiển đường dây LCR chứa các tham số về độ dài từ mã, parity, số bit stop.

- Thanh ghi AL chứa trạng thái đường dây.

- Thanh ghi AH chứa trạng thái hiện thời của các tín hiệu điều khiển MODEM.

- ***Khi AH = 01H***

Khi kiểm tra thấy AH = 01H thì máy tính sẽ thực hiện chức năng gửi một ký tự ra cổng RS232. Ký tự được gửi chứa trong AL. Quá trình gửi được thực hiện như sau.

Thanh ghi điều khiển MODEM MCR - có địa chỉ offset là 4 được gọi tới để thực hiện điều khiển tín hiệu tại các chân DTR và RTS của UART 8250. Để điều khiển được tín hiệu DTR (thiết bị truyền thông sẵn sàng) và RTS (yêu cầu phát) thì các bit D_0D_1 phải có mức lôgic là 11. CPU thực hiện kiểm tra, nếu $D_0D_1 \neq 11$ có nghĩa là MODEM không sẵn sàng để truyền thông và việc truyền thông không thực hiện được, còn nếu $D_0D_1 = 11$ có nghĩa là MODEM đã sẵn sàng và yêu cầu CPU gửi dữ liệu ra. Khi này để xem trạng thái truyền thông xảy ra như thế nào ta xem xét thanh ghi trạng thái đường truyền LSR bằng việc kiểm tra bit D_5 của nó, nếu $D_5 = 0$ việc gửi không được thực hện, nếu bit $D_5 = 1$ có nghĩa ký tự đã được chuyển từ thanh ghi đệm phát THR - có địa chỉ offset là 0 - sang thanh ghi đệm thu RHR - cũng có offset là 0 và khi này ký tự cần gửi đi đang ở trong thanh ghi AX sẽ được gửi cổng truyền thông.

- ***Khi AH = 02H***

Khi kiểm tra thấy AH = 02H thì máy tính sẽ thực hiện chức năng thu một ký tự từ cổng RS232. Quá trình thu được thực hiện như sau.

Trước tiên phải đặt máy tính ở chế độ sẵn sàng nhận ký tự từ sổng RS232 bằng việc đặt bit D_0 của thanh ghi điều khiển MODEM MCR lên mức lôgic 1.

Tiếp theo CPU sẽ kiểm tra xem MODEM đã sẵn sàng chưa bằng việc kiểm tra bit D_5 của thanh ghi trạng thái MODEM MSR, nếu bit $D_5 = 0$ có nghĩa MODEM chưa sẵn sàng để gửi, còn nếu $D_5 = 1$ có nghĩa MODEM đã sẵn sàng để gửi ký tự đến máy tính.

Sau đó kiểm tra xem bộ đệm thu đã sẵn sàng nhận ký tự hay chưa bằng việc kểm tra bit D_0 của thanh ghi trạng thái đường dây LSR, nếu bít $D_0 = 0$ có nghĩa là CPU đang đọc thanh ghi đệm thu và nó không sẵn sàng để nhận dữ liệu, còn nếu bit $D_0 = 1$ có nghĩa làbộ đệm thu sẵn sàng nhận dữ liệu.

Cuối cùng thực hiện chuyển ký tự từ cổng truyền thông RS232 vào thanh ghi AL và thoát trở về chương trình chính.

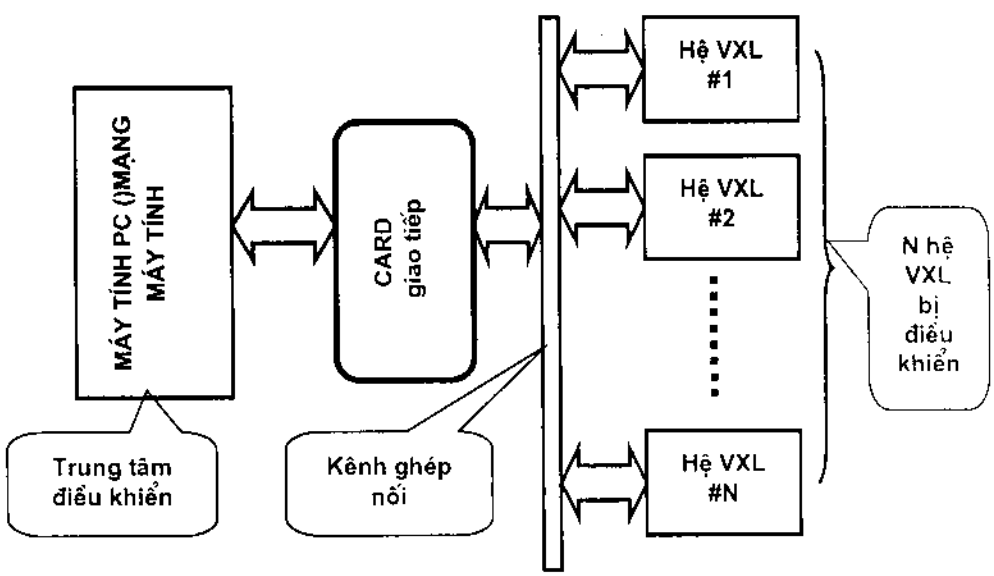
- ***Khi AH = 03H***

Khi kiểm tra thấy AH = 03H thì máy tính sẽ thực hiện chức năng thu trạng thái của cổng truyền thông. Trạng thái đường truyền được ghi vào thanh ghi AL. Trạng thái MODEM được ghi vào thanh ghi AH.

CHƯƠNG 3

GIAO TIẾP GIỮA HỆ VI XỬ LÝ VÀ MÁY TÍNH PC

Hệ vi xử lý chuyên dụng được sử dụng rất nhiều trong các hệ điều khiển lớn, Chúng có thể hoạt động độc lập và có thể hoạt động trong khuôn khổ của hệ thống khác. Trường hợp thứ hai chúng thường được nối ghép với máy tính (PC hay LAN hoặc lớn hơn nữa là Main Frame, máy tính Super) và nhận điều khiển trực tiếp từ các trung tâm này. Phương thức điều khiển này tạo ta các đối tượng chấp hành thông minh vì hệ vi xử lý chỉ cần nhận lệnh thì nó đã tự biết phương thức hoạt động tiếp theo. Đây cũng là nền tảng cho việc xây dựng các hệ điều khiển đa phương tiện hiện đại (hình 3.1).



Hình 3.1. Hệ thống điều khiển lớn

3.1. HỆ VI XỬ LÝ CHUYÊN DỤNG

Hệ vi xử lý chuyên dụng lúc này là đối tượng chấp hành của hệ thống lớn mà trung tâm điều khiển là máy tính PC hay mạng máy tính như đã trình bầy ở trên. Hệ vi xử lý chuyên dụng hoạt động được nhờ vào chương trình MONITOR nạp sẵn trong bộ nhớ EPROM, chương trình này sẽ quản lý điều hành toàn bộ hệ vi xử lý để hệ vi xử lý hoạt động theo đúng chức năng, yêu cầu đề ra.

Để PC có thể can thiệp sâu vào chức năng của hệ vi xử lý chuyên dụng, phương pháp tốt nhất là can thiệp thẳng vào nội dung của chương trình MONITOR của nó. Muốn như vậy EPROM của hệ phải là RAM để PC có thể tải file MONITOR vào bất kỳ lúc nào khi cần thiết, đó chính là kỹ thuật tạo ROM MONITOR mô phỏng gọi tắt là ROM mô phỏng. ROM mô phỏng được tổ chức sao cho có thể đảm nhiệm hai chức năng sau:

Hệ vi xử lý chuyên dụng coi nó như bộ nhớ chương trình của mình và có thể hoạt động được trên bộ nhớ giả ROM đó.

Có thể thay đổi nội dung trên bộ nhớ giả ROM, tức là có thể sửa đổi hoàn thiện chương trình cho hệ vi xử lý chuyên dụng tới khi hệ hoạt động một cách tối ưu

Có hai giải pháp thực hiện kỹ thuật mô phỏng trên là: sử dụng bộ nhớ RAM có dung lượng tương đương bộ nhớ EPROM của hệ vi xử lý cần thiết kế và sử dụng một vùng RAM của PC để tạo giả bộ nhớ EPROM của hệ vi xử lý.

- *Giải pháp dùng bộ nhớ RAM mô phỏng chức năng EPROM*

Trước đây, giải pháp thông thường là sử dụng một vi mạch RAM có dung lượng tương đương với EPROM để làm giả bộ nhớ EPROM. vi mạch RAM này được gắn vào một mạch thích nghi. Mạch thích nghi có vai trò nối ghép RAM với máy tính (dùng điều khiển điều quá trình mô phỏng) và với hệ vi xử lý chuyên dụng để hệ vi xử lý chạy thử chương trình đó.

Về cấu trúc của phần cứng, mạch thích nghi có hạt nhân là một chip RAM được tổ chức sao cho nó có thể được truy nhập từ hai phía (hình 3.2):

Máy PC có thể chuyển dữ liệu vào chip RAM này.

Hệ vi xử lý chuyên dụng có thể đọc dữ liệu từ nó.

Việc truy nhập không xảy ra đồng thời để tránh việc xung đột tranh chấp kênh dữ liệu.

Chức năng của các khối cơ bản của mạch thích nghi như sau:

Cổng ghép nối PC và cổng ghép nối với hệ vi xử lý: là các cổng ba trạng thái sẽ được mở không đồng thời bởi các tín hiệu CS do bộ logic điều khiển cung cấp.

Bộ nhớ RAM để PC viết chương trình lên đó và sau đó sẽ được đọc bởi hệ vi xử lý.

Khối logic điều khiển được điều khiển từ PC bằng cách PC ghi lên thanh ghi điều khiển của nó các từ điều khiển quyết định việc RAM của mạch sẽ được truy nhập từ cổng ghép nối nào.

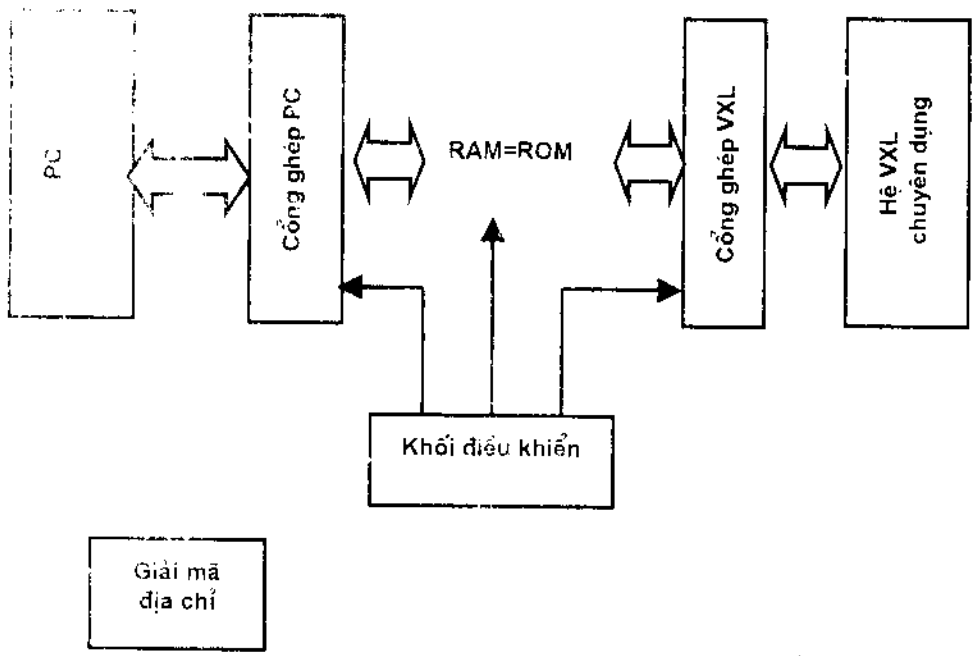
Khối giải mã địa chỉ sẽ giải mã địa chỉ cho một miền nhớ RAM không được sử dụng trong PC (ví dụ miền A0000h khi PC đặt màn hình ở chế độ text) và RAM của mạch thích nghi sẽ được đặt vào miền nhớ này của PC.

Như vậy, ta có thể tóm tắt việc xây dựng một hệ vi xử lý chuyên dụng khi sử dụng phương án này như sau:

- Xây dựng phần cứng cho hệ vi xử lý chuyên dụng.

- Thiết kế mạch thích nghi cho hệ vi xử lý đó.

- Viết chương trình cho hệ vi xử lý trên máy PC, dùng PC dịch chương trình ra mã máy của hệ vi xử lý, sau đó gửi chương trình ở dạng mã máy ra cho mạch thích nghi để hệ vi xử lý có thể chạy chương trình trên mạch này, với khả năng ta có thể tự do thay đổi chương trình một cách linh hoạt cho đến khi hoàn thiện.



Hình 3.2. Tạo ROM mô phỏng

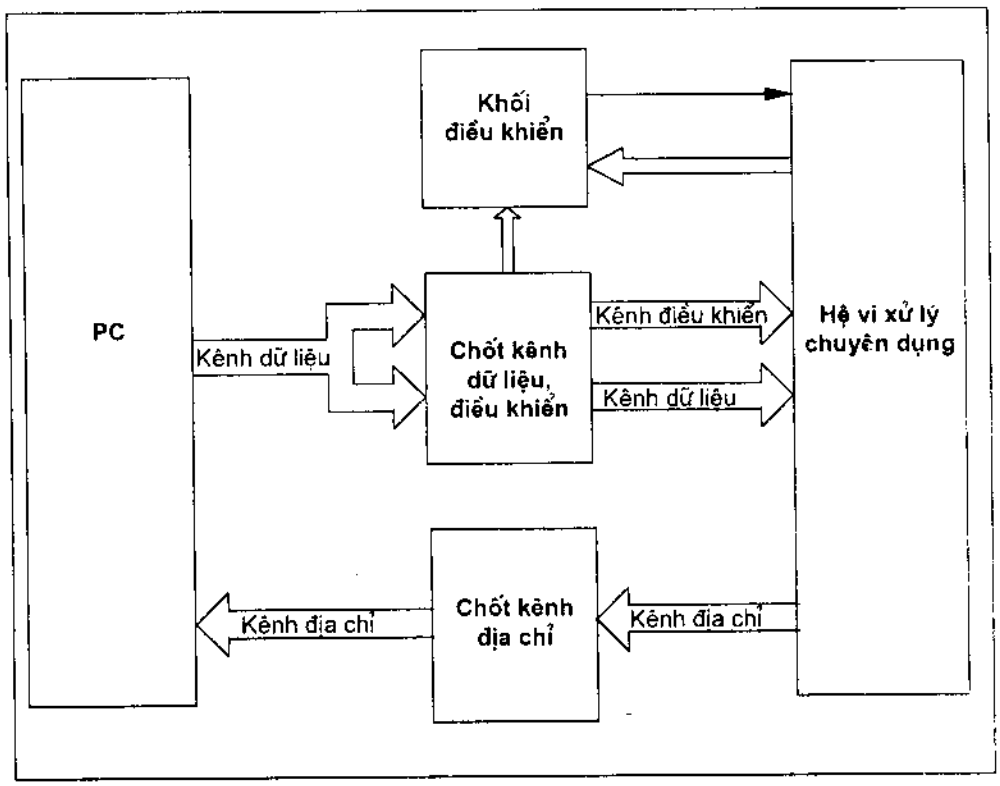
Các máy tính cá nhân PC được áp dụng rộng rãi trong lĩnh vực công nghệ thông tin với tư cách là trung tâm điều khiển có nhiều ngoại vi thông minh hoạt động. Mỗi ngoại vi thông minh thường là một hệ vi xử lý chuyên dụng. Ngày nay tốc độ xử lý của PC nhanh hơn rất nhiều so với tốc độ của các hệ vi xử lý chuyên dụng (trong khi hệ vi xử lý thực hiện một lệnh thì PC có thể đã thực hiện được hàng trăm lệnh thậm chí hàng ngàn lệnh). Do vậy có thể đặt vấn đề sử dụng PC kèm theo một chương trình phần mềm cần thiết, một card giao tiếp giữa PC và hệ vi xử lý chuyên dụng, sao cho có thể sử dụng một vùng RAM ngay trong PC để mô phỏng cho EPROM để nạp chương trình hệ thống của hệ chuyên dụng (hình 3.3).

Giải pháp này được thực hiện theo cơ chế như sau:

- Viết chương trình cho hệ vi xử lý trên máy PC, sau đó dịch sang dạng mã máy dạng file.OBJ.

- Một chương trình quản lý hệ thống chạy trên PC sẽ tải file.OBJ nêu trên vào một vùng RAM sao cho có thể quản lý địa chỉ và xuất dữ

liệu từ vùng RAM đó ra hệ vi xử lý chuyên dụng thông qua CARD giao tiếp.



***Hình 3.3.* Sơ đồ card giao tiếp với hệ Vi Xử Lý**

Đồng thời cũng qua CARD giao tiếp, chương trình hệ thống trên PC phải luôn luôn kiểm soát và nắm bắt các tín hiệu địa chỉ và các tín hiệu điều khiển từ hệ vi xử lý đưa sang để có thể khống chế và xuất dữ liệu thích hợp cho hệ vi xử lý trong suốt quá trình.

- Mỗi khi CPU của hệ vi xử lý cần đọc dữ liệu từ EPROM, nó sẽ phát ra các tín hiệu địa chỉ và điều khiển tương ứng, chương trình quản lý hệ thống sẽ đáp ứng và qui chiếu đến vùng RAM trong máy tính PC đã nói ở trên, sau đó xuất dữ liệu tương ứng ra BUS dữ liệu của hệ vi xử lý cho CPU đọc thay vì đọc từ EPROM.

Chức năng của các khối cơ bản như sau:

Khối điều khiển do PC điều khiển qua chốt điều khiển để phát hiện chu kỳ máy của CPU trên hệ vi xử lý cần truy nhập EPROM và tạo tín hiệu READY bắt CPU đợi PC xuất dữ liệu ra kênh dữ liệu hệ vi xử lý.

Bộ chốt dữ liệu, điều khiển đưa dữ liệu từ RAM mô phỏng trong PC ra kênh dữ liệu của hệ vi xử lý để hệ vi xử lý sử dụng và đưa tín hiệu điều khiển từ PC ra để điều khiển chế độ làm việc của hệ vi xử lý.

Bộ chốt địa chỉ để PC đọc địa chỉ trên kênh địa chỉ của hệ vi xử lý trong chu kỳ máy CPU truy nhập EPROM.

Như vậy, ta có thể tóm tắt việc xây dựng một hệ vi xử lý chuyên dụng khi sử dụng phương án này như sau:

- Xây dựng phần cứng cho hệ vi xử lý chuyên dụng.

- Viết chương trình cho hệ vi xử lý bằng các trình biên dịch tương ứng CPU, dịch chương trình ra mã máy của hệ vi xử lý.

- Điều khiển hệ vi xử lý chạy bằng chương trình này.

Giải pháp dùng một vùng RAM của PC để tạo giả bộ nhớ EPROM của hệ vi xử lý là khả thi vì CARD mô phỏng rất đơn giản, độ tin cậy cao và hệ thống này rất mềm dẻo vì mỗi một hệ mô phỏng được sử dụng cho các hệ vi xử lý khác nhau nếu chúng dùng cùng một loại CPU. Còn trong trường hợp hệ vi xử lý dùng CPU loại khác thì ta chỉ phải thay đổi một vài đường tín hiệu điều khiển tương ứng với loại CPU mới trên phần cứng của CARD giao tiếp. Do đó ta chọn giải pháp dùng một vùng RAM của PC để tạo giả bộ nhớ EPROM.

3.2. HỆ VI XỬ LÝ 8 BIT 8085 INTEL

Các hệ vi xử lý ngoại vi thường là các hệ 8 bit có cấu hình tiêu chuẩn bao gồm các thành phần chính như sau:

Đơn vị xử lý trung tâm được chọn là CPU Intel 8085A.

Bộ nhớ trong của hệ gồm hai thành phần: bộ nhớ RAM được chọn là loại 6264 và bộ nhớ EPROM được chọn là loại 2764. Cả hai loại trên đều là bộ nhớ có kích thước là (8k*8)bit nên hoàn toàn tương thích với bộ vi xử lý 8085A.

Cổng vào/ra để CPU trao đổi dữ liệu với thiết bị ngoại vi chọn là vi mạch 82C55.

Hình 3.4 là sơ đồ nối ghép CPU 8085A, RAM 67264, EPROM 2764, bộ giải mã 74LS138, chốt 74LS373, PI/O 82C55 của hệ vi xử lý Intel 8085A.

Do đặc điểm của CPU 8085A, kênh dữ liệu (D0-D7) và kênh địa chỉ phần thấp (A0- A7) sử dụng chung các chân AD0 -AD7 nên cần sử dụng bộ chốt địa chỉ IC 74LS 373 được điều khiển bằng tín hiệu ALE từ CPU 8085. Tám tín hiệu ra được chốt của IC 373 chính là tín hiệu địa chỉ phần thấp (A0 - A7) sử dụng trên BUS địa chỉ của hệ vi xử lý.

Việc quản lý các vùng địa chỉ cho ROM, RAM, PI/O (thường gọi là đánh địa chỉ) được thực hiện như sau:

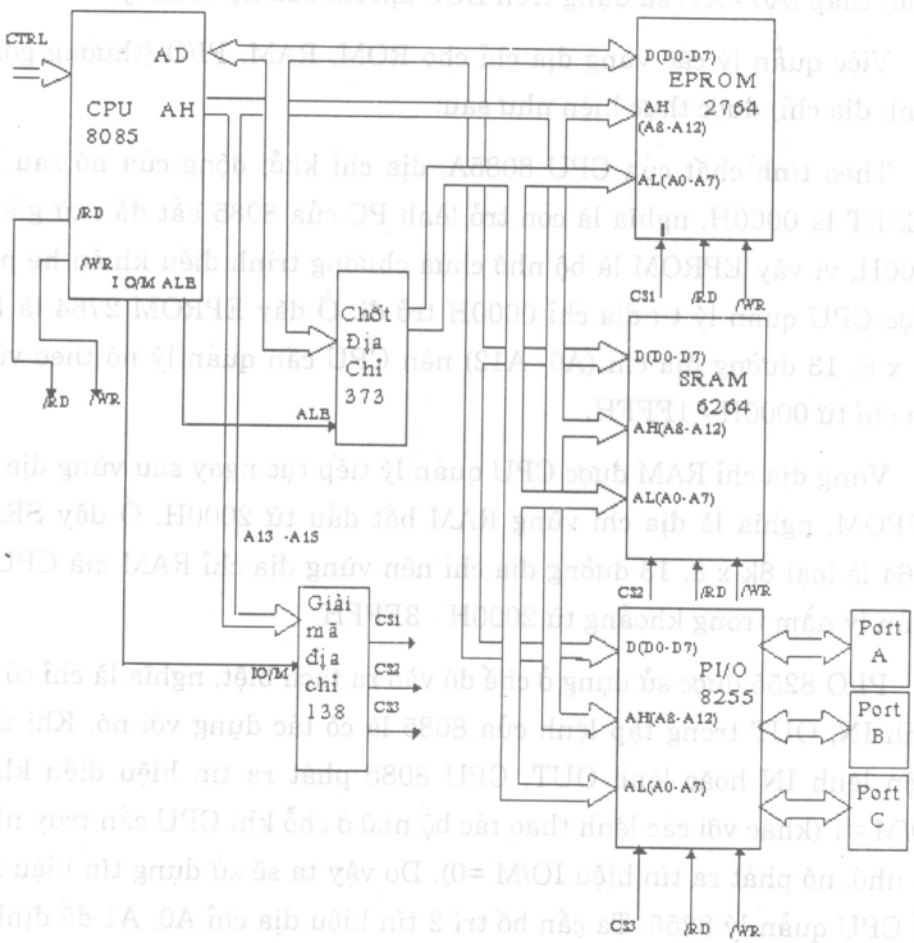
Theo tính chất của CPU 8085A, địa chỉ khởi động của nó sau khi RESET là 0000H, nghĩa là con trỏ lệnh PC của 8085 bắt đầu từ giá trị 0000H, vì vậy EPROM là bộ nhớ chứa chương trình điều khiển hệ phải được CPU quản lý từ địa chỉ 0000H trở đi. Ở đây EPROM 2764 là loại 8k x 8, 13 đường địa chỉ (A0- A12) nên CPU cần quản lý nó theo vùng địa chỉ từ 0000H - 1FFFH.

Vùng địa chỉ RAM được CPU quản lý tiếp tục ngay sau vùng địa chỉ EPPOM, nghĩa là địa chỉ vùng RAM bắt đầu từ 2000H. Ở đây SRAM 6264 là loại 8k x 8, 13 đường địa chỉ nên vùng địa chỉ RAM mà CPU sẽ quản lý nằm trong khoảng từ 2000H - 3FFFH.

PI/O 8255 được sử dụng ở chế độ vào ra tách biệt, nghĩa là chỉ có các lệnh IN, OUT trong tập lệnh của 8085 là có tác dụng với nó. Khi thực hiện lệnh IN hoặc lệnh OUT, CPU 8085 phát ra tín hiệu điều khiển IO/M =1 (khác với các lệnh thao tác bộ nhớ ở chỗ khi CPU cần truy nhập bộ nhớ, nó phát ra tín hiệu IO/M =0). Do vậy ta sẽ sử dụng tín hiệu này để CPU quản lý 8255. Ta cần bố trí 2 tín hiệu địa chỉ A0, A1 để định vị các cổng vào ra (PortA, PortB, PortC) cho 8255.

Bảng sự thật của IC 138

E3	E2	E1	C	IO/M	A13	A12-A0	CS1	CS2	CS3
1	0	0	0	0	0	x	0	1	1
1	0	0	0	0	1	x	1	0	1
1	0	0	0	1	0	x	1	1	0
1	0	0	0	x	x	x	1	1	1
1	0	0	0	x	x	x	1	1	1



Hình 3.4. Hệ vi xử lý Intel 8085

Với những lý do nêu trên, IC 138 được sử dụng làm bộ giải mã địa chỉ với chân tín hiệu A (chân 1) được nối với đường địa chỉ A13, chân B (chân 2) nối với tín hiệu IO/M, chân C (pin 3), các chân E1(chân 4), E2 (chân 5) được nối với đất, chân E3 nối với mức 1 (VCC) qua điện trở hạn chế R3. Các chân tín hiệu ra được nối như sau:

- Y0 (chân 15) nối với CS1 để chọn EPROM.

- Y1 (chân 5) nối với CS2 để chọn RAM.

- Y3 (chân 5) nối với CS3 để chọn PI/O.

Nhìn vào bảng sự thật của 138 ta thấy tất cả các địa chỉ 4000H không có tác dụng gì đối với hệ. Đây là điều cần lưu ý khi lập trình cho hệ.

Đường tín hiệu IO/M có vai trò trong việc hoặc là truy nhập bộ nhớ (cả ROM và RAM) hoặc là truy nhập cổng (thông qua PI/O 8255), cụ thể:

- Khi IO/M =0, CPU truy nhập bộ nhớ.

-Khi IO/M =1, CPU truy nhập cổng.

Khi CPU truy nhập cổng (IO/M =1) thì A13 luôn bằng 0 vì ta chỉ sử dụng địa chỉ cổng 2 bit (A0,A1), do vậy đảm bảo được sự đúng đắn của tín hiệu chọn P/IO(CS3=0).

Đường tín hiệu A13 làm nhiệm vụ chọn ROM hoặc chọn RAM.

Khi A13=0, CPU truy nhập ROM.

Khi A13=1, CPU truy nhập RAM.

Ta có bản đồ phân vùng bộ nhớ của hệ vi xử lý như hình 3.4.

Trừ một số đường tín hiệu điều khiển đặc biệt, các thao tác đọc/ghi là sự trao đổi thông tin duy nhất giữa CPU và các thành phần khác, và đó chính là tất cả sự cần thiết để thực hiện bất kỳ một lệnh hoặc một chương trình nào.

Để giúp cho việc thiết kế CARD làm nhiệm vụ giao tiếp giữa hệ vi xử lý và máy tính và thiết kế phần mềm hệ thống quản lý các thao tác mô phỏng cho EPROM, ta cần nắm rõ hơn về cơ chế truy nhập bộ nhớ và cổng của CPU 8085 mà cần thiết nhất ở đây là cơ chế nạp mã lệnh

(OPCODE FETCH) và đọc bộ nhớ (READ) vì thực chất cơ chế nạp mã lệnh cũng chính là cơ chế đọc lệnh từ ROM.

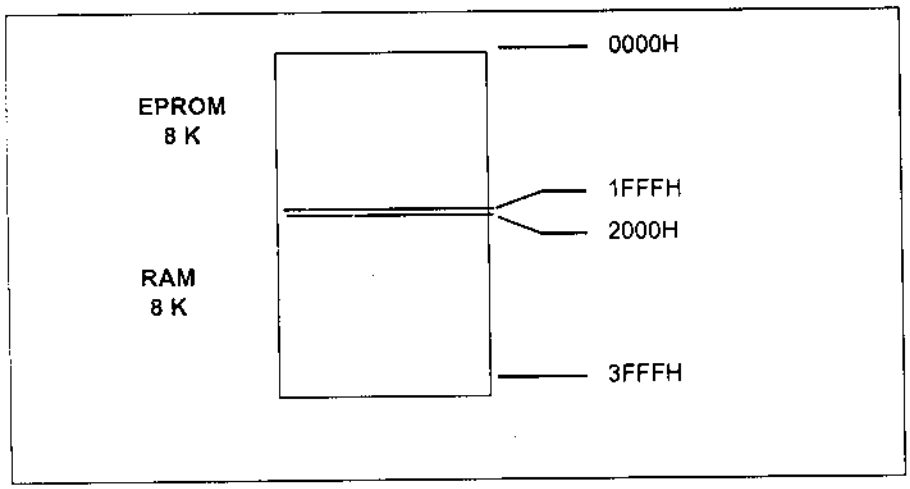
3.3. TỔ CHỨC CARD GIAO TIẾP GIỮA PC VÀ HỆ VI XỬ LÝ

Từ ý tưởng thực hiện mô phỏng EPROM của CPU 8085A trong hệ thống điều khiển cho ta thấy rằng có thể tận dụng trạng thái chờ (READY) của CPU 8085A, sao cho mỗi khi nó cần truy nhập EPROM để đọc mã lệnh hoặc dữ liệu, ta gán tín hiệu READY=0 để buộc CPU phải chờ, khi đó trên BUS địa chỉ của hệ vi xử lý vẫn duy trì giá trị địa chỉ cần truy nhập (phần cao A8 (A15 do chính CPU duy trì, phần thấp A0 (A7 đã được bộ chốt địa chỉ IC 373 chốt lại) và chương trình mô phỏng trong máy tính sẽ tiến hành đọc giá trị địa chỉ đó, qui chiếu với vùng RAM mô phỏng để xuất dữ liệu từ địa chỉ tương ứng ra BUS dữ liệu của hệ vi xử lý. Sau đó nếu ta cho CPU 8085A trở lại làm việc thì nó sẽ tiến hành đọc dữ liệu trên BUS dữ liệu mà ta đã đưa ra cho nó và tiến hành tiếp tục chu kỳ lệnh theo mã lệnh hoặc dữ liệu nó vừa nhận được. Còn khi CPU 8085A thực hiện chu kỳ máy không có truy nhập EPROM thì ta cứ để nó làm việc bình thường. Muốn thực hiện được điều đó, cần phải kết hợp giữa việc tổ chức phần cứng và việc tổ chức phần mềm.

Nhìn vào biểu đồ thời gian thực hiện một chu kỳ máy, ta thấy tối ưu nhất là gán tín hiệu READY =0 ngay trong giai đoạn T1 vì đến T2 CPU mới kiểm tra sự có mặt của tín hiệu READY. Ta sẽ sử dụng các tín hiệu ALE và CSROM (trên hệ vi xử lý CSROM được ký hiệu là CS1) sao cho khi có tín hiệu ALE =1 báo bắt đầu một chu kỳ máy và CSROM=0 báo CPU cần truy nhập EPROM thì mạch phần cứng tự động tạo tín hiệu READY=0. Sau khi chương trình hệ thống thực hiện xong các thao tác đọc địa chỉ và xuất ra dữ liệu, chương trình này sẽ tạo ra tín hiệu READY=1 để cho phép CPU làm việc trở lại.

CARD giao tiếp giữa hệ vi xử lý và máy tính phải làm được các nhiệm vụ như sau:

- Truyền các tín hiệu điều khiển (RESETIN, HOLD) từ chương trình hệ thống sang hệ vi xử lý để khống chế thao tác khởi động,cho phép hoặc cấm CPU làm việc.



Hình 3.5. **Bản đồ phân vùng bộ nhớ**

- Phát hiện sự xuất hiện tín hiệu ALE và CSROM để tạo tín hiệu READY=0 bắt CPU chờ.

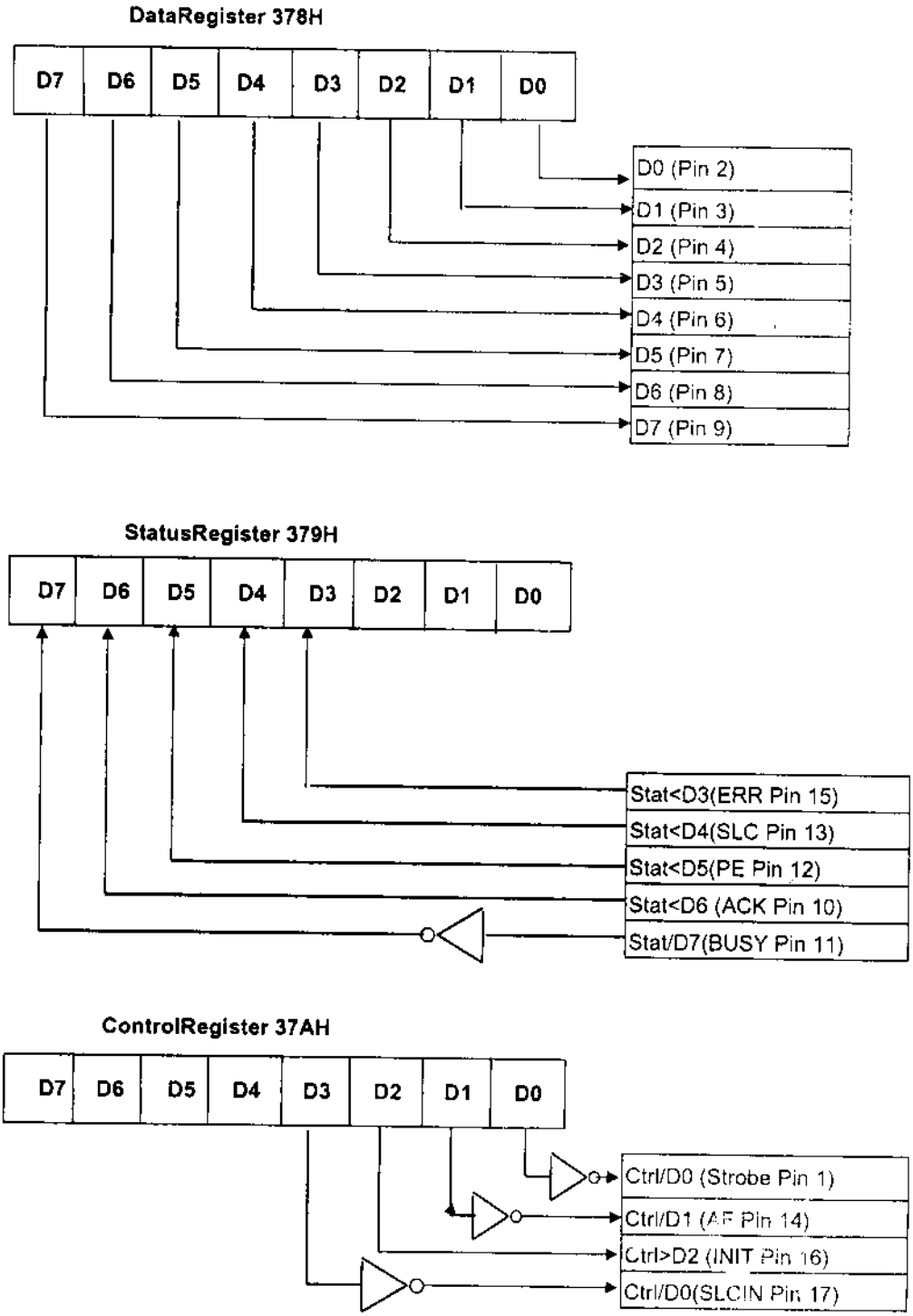
- Nhập các tín hiệu địa chỉ (A0-->A15) từ hệ vi xử lý vào cổng 379h (LPT1) của máy tính để chương trình hệ thống đọc và ánh xạ đến vùng RAM mô phỏng.

- Đưa dữ liệu mà chương trình hệ thống xuất ra từ vùng vùng RAM mô phỏng sang BUS dữ liệu của hệ vi xử lý.

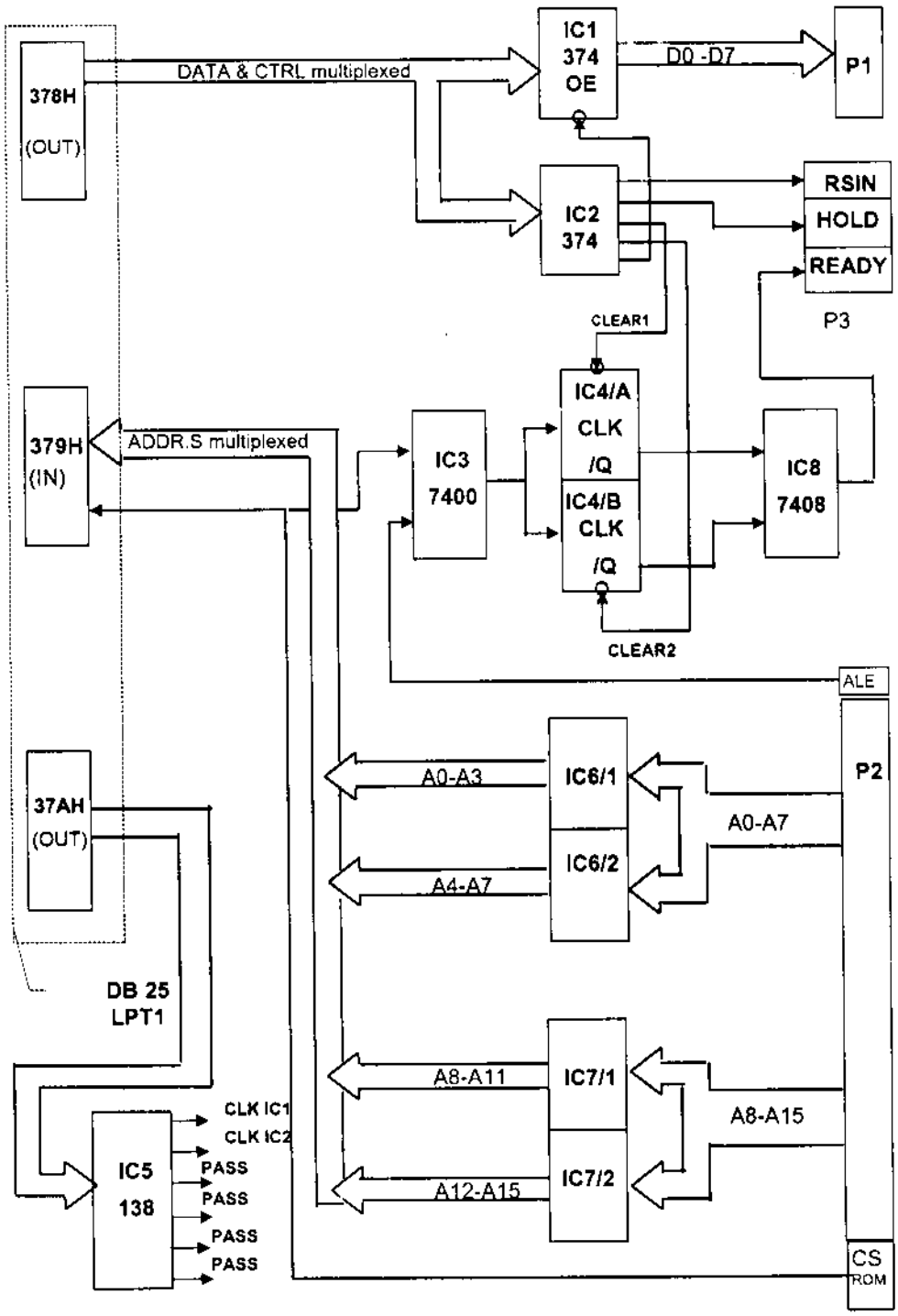
- Nhận tín hiệu cho phép CPU trở lại hoạt động của chương trình hệ thống để tạo tín hiệu READY=1cho CPU 8085.

Sơ dồ khối của CARD giao tiếp, sơ đồ nguyên lý của CARD được thể hiện trên hình 3.7.

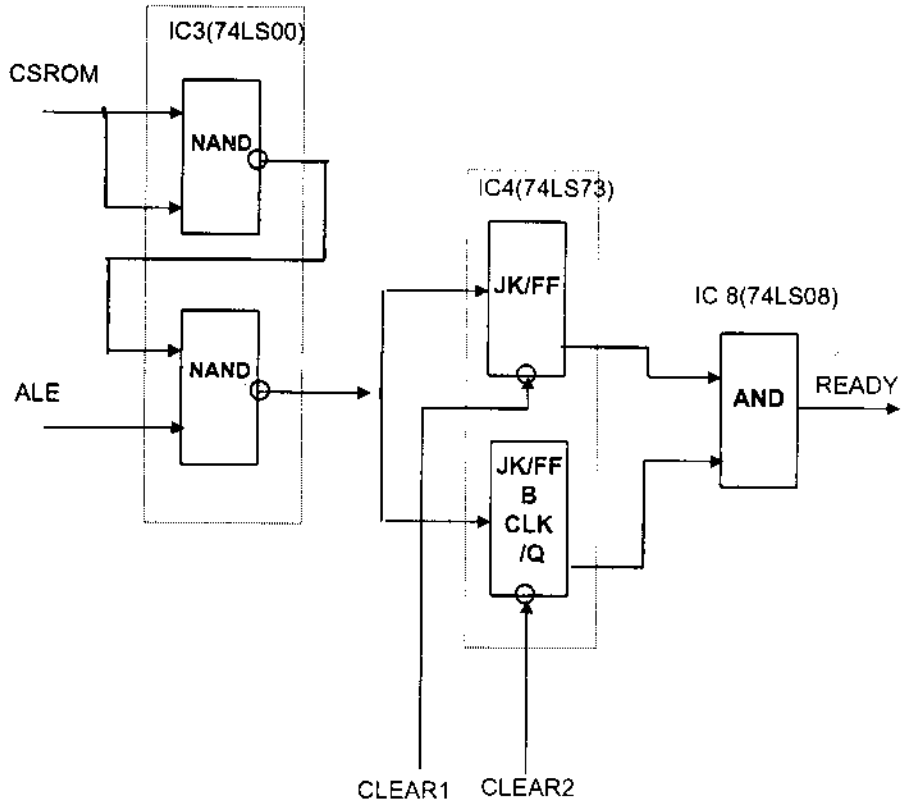
Khối các IC3 (74LS 00), IC 4 (74LS73), IC8 (4LS08) làm nhiệm vụ sẵn sàng bắt các tín hiệu ALE=1, CSROM=0 để tự động tạo tín hiệu READY=0 làm dừng CPU và nhận các tín hiệu Clear từ chương trình hệ thống để tạo READY=1 cho IC4 (74LS73). IC này gồm hai triger flipflop JK (IC 4/A và IC 4/B) với các chân J được nối với +5V qua điện trở hạn dòng, các chân K nối đất. Các cửa ra /Q được reset ở mức 1(dương) khi chân Clear =0. Khi chân Clear ở mức 1, triger sẽ đợi khi có một sườn xung xuống (chuyển từ mức 1 về mức 0) đặt vào chân CLK thì triger lật trạng thái và cửa ra /Q lật sang mức 0. Khi chân Clear ở mức 0 thì tín hiệu CLK không có tác dụng gì cho việc lật trạng thái triger.



Hình 3.6. **Phân bố các Thanh ghi cổng song song LPT1**



Hình 3.7. Sơ đồ nguyên lý của card giao tiếp



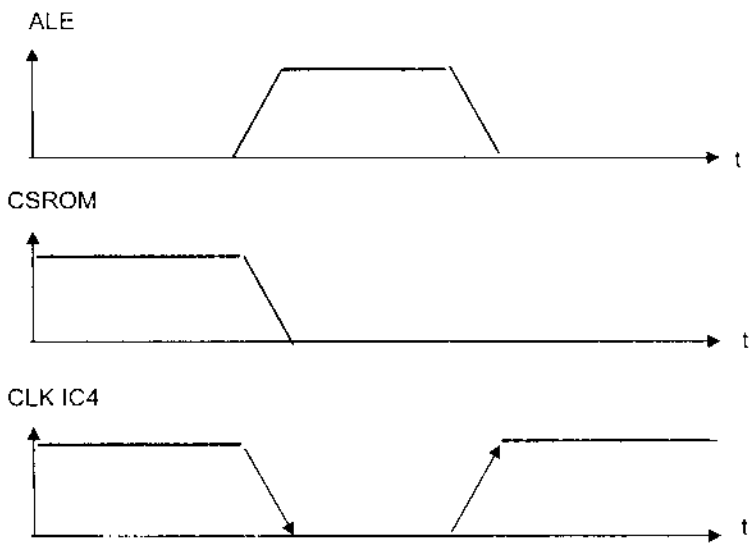
Hình 3.8. **Sơ đồ mạch tạo tín hiệu READY**

IC3 (74LS00) sử dụng hai mạch cổng NAND để bắt đồng thời 2 tín hiệu ALE=1và CSROM=0 để tạo sườn xung xuống kích cho các chân CLK của IC 4. Biểu đồ thời gian của tín hiệu đầu ra IC3 phụ thuộc vào các tín hiệu ALE và CSROM như sau: IC8(74LS08) sử dụng một cổng AND với đầu ra của nó là tín hiệu READY =0 khi một trong hai đầu ra /Q của IC4 ở mức 0. READY =1 chỉ khi cả hai đầu ra /Q của IC 4 đều ở mức 1.

Việc sử dụng đồng thời hai tín hiệu ALE và CSROM nhằm đảm bảo tạo tín hiệu READY =0 ngay khi bắt đầu một chu kì máy (giai đoạn T1) và chu kỳ máy đó chỉ là chu kỳ CPU cần truy suất EPROM.

Hoạt động của các IC3, IC4 và IC8 như sau:Trước khi thực hiện qúa trình mô phỏng, chương trình hệ thống cần tạo ra các tín hiệu Clear1, Clear2 mức 0 để reset IC 4 sao cho các cửa ra /Q đồng thời ở mức 1, do

đó tín hiệu READY từ sau mạch AND (IC8: 74LS08) có mức 1 là trạng thái sẵn sàng để CPU làm việc. Sau đó một trong hai tín hiệu CLEAR1, CLEAR2 được nâng lên mức 1, chẳng hạn CLEAR1=1, CLEAR2=0. IC 4/A ở trạng thái sẵn sàng đón đợi tác động của ALE và CSROM từ IC3 đưa sang, còn IC 4/B sẽ không bị tác động. Cả hai triger giữ trạng thái /Q=1 và READY=1(qua cổng AND IC8).



Hình 3.9. **Biểu đồ thời gian mạch tạo sườn xuống kích cho triger IC4 từ hai tín hiệu ALE và /CSROM.**

Khi bắt đầu một chu kỳ máy truy xuất EPROM, chẳng hạn bắt đầu reset CPU 8085, các tín hiệu ALE=1 và CSROM =0 tác động vào IC3 tạo ra sườn xung xuống kích vào IC 4 làm cho IC 4/Alật trạng thái /Q=0, IC 4/B không bị ảnh hưởng. Do đó READY =0. Vẫn trong thời đoạn T1 của chu kỳ máy đó khi ALE chuyển xuống mức 0 (còn CSROM =0 sẽ kéo dài suốt chu kỳ máy) thì đầu ra IC3 lật lên mức 1 sẵn sàng đón đợi cho chu kỳ máy tiếp theo, sườn lên của xung này không có ảnh hưởng gì đến các triger (IC 4/A, IC 4/B).

Chuyển sang thời đoạn T2 của chu kỳ máy đó, CPU nhận thấy có tín hiệu READY=0 và nó tạm dừng để chương trình hệ thống đọc địa chỉ và xuất mã lệnh hoặc dữ liệu tương ứng cho nó. Sau khi chương trình hệ

thống làm xong việc trên, chương trình hệ thống sẽ đưa đồng thời hai tín hiệu CLEAR1=0 và CLEAR2=1với mục đích reset cho IC 4/A nhằm đưa ra tín hiệu READY=1để CPU bắt đầu trở lại làm việc và đặt cho IC 4/B vào trạng thái chờ tác động của ALE và CSROM (nếu có) của chu kỳ máy tiếp sau.

Cứ như vậy trong suốt quá trình thực hiện mô phỏng, IC 4/A và IC 4/B luân phiên ở trạng thái dợi (Clear=1) và trạng thái reset (Clear=0) bằng cách chương trình hệ thống dưa đồng thời các tín hiệu CLEAR1 và CLEAR2 đảo mức với nhau. Kỹ thuật đảo vai trò của IC 4/A và IC 4/B này là biện pháp hữu hiệu đảm bảo cho IC4 luôn sẵn sàng đáp ứng với tác động của ALE =1 và CSROM=0.

Vì sao phải sử dụng cả hai tín hiệu ALE và CSROM để tạo ra tín hiệu READY mà không chỉ sử dụng một mình tín hiệu CSROM. Ta đã thấy rằng tín hiệu ALE=1 chỉ xuất hiện ở thời đoạn T1 đầu chu kỳ máy, còn CSROM=0 duy trì trong suốt chu kỳ máy CPU cần truy xuất EPROM. Nếu chỉ sử dụng CRSOM (sườn xuống) để kích hoạt IC4 tạo ra READY=0, thì có thể đến chu kỳ máy tiếp sau CPU lại cần truy xuất EPROM, như vậy tín hiệu CSROM khó có thể tạo sườn xuống để tiếp tục kích hoạt cho IC4. Khi sử dụng kết hợp cả hai tín hiệu ALE và CSROM thông qua IC3 thì ngay sau khi ALE chuyển về mức 0 thì ở đầu vào CLK của IC4 được nâng lên mức 1và IC4 luôn sẵn sàng để tạo READY=0 khi cần thiết.

IC1 có nhiệm vụ đưa dữ liệu (dữ liệu này được lấy từ thanh ghi 378H/LPT1) được chốt ra kênh dữ liệu (qua các đầu ra RD0-->RD7) để hệ vi xử lý sử dụng (mà lẽ ra CPU cần đọc nó từ EPROM). IC này chỉ được phép hoạt động (truyền 8 bit dữ liệu từ vùng DATBUFF sang hệ vi xử lý) khi chương trình hệ thống đòi hỏi. Còn thì bình thường các chân dữ liệu ra của nó phải ở mức trở kháng cao để không làm ảnh hưởng đến sự làm việc của hệ vi xử lý. Điều đó được thực hiện bằng cách đưa mức 0 (cho phép) hoặc mức 1 (cấm) vào chân /OE (pin 1) của nó.

IC2 được điều khiển từ chương trình hệ thống để làm các nhiệm vụ:

- Đưa các tín hiệu RESETIN (bit d0)và HOLD (bit d1) được chốt ra BUS điều khiển hệ vi xử lý.

- Tạo các tín hiệu CLEAR1(bit d2) CLEAR2 (bit d3) luân phiên đặt mức 1 và 0 trong quá trình mô phỏng để IC4 thường trực nắm bắt các tín hiệu ALE và CSROM và khi đó IC4 tự động phát ra tín hiệu READY=0. Cũng nhờ đó chương trình hệ thống có thể chủ động kiểm soát việc phát ra tín hiệu READY=1 cho phép CPU trở lại quá trình tiếp tục làm việc.

- Tạo tín hiệu cấm hoặc cho phép IC1 hoạt động (bit d7).

Các tín hiệu trên cũng được lấy từ thanh ghi 378H/LPT1 nối vào các cổng vào của IC2. Việc sử dụng thanh ghi 378H multiflexer vừa đảm nhiệm xuất dữ liệu cho hệ vi xử lý (qua IC1) vừa đảm nhiệm xuất các tín hiệu điều khiển cho hệ vi xử lý và cho bản thân CARD giao tiếp như trên cho thấy rõ hơn ý nghĩa của việc sử dụng các IC1 và IC2 là các vi mạch chốt (bằng sườn dương) 74LS374 đối với yêu cầu bắt buộc các tín hiệu đầu ra của chúng không bị ảnh hưởng trực tiếp đến nhau.

IC6 (74 LS 244) được tách thành hai phần: phần IC6/1 có nhiệm vụ nhập 4 bit thấp (A0-->A3) của byte thấp từ BUS địa chỉ hệ vi xử lý vào máy tính, phần IC6/2 có nhiệm vụ nhập 4 bit cao (A4--> A7) của byte thấp từ BUS địa chỉ vào máy tính.

IC7 (74LS 244) cũng được tách thành 2 phần: phần IC7/1 có nhiệm vụ nhập 4 bit thấp của byte cao (A8-->A11), còn phần IC7/2 làm nhiệm vụ nhập 4 bit cao của byte cao (A12-->A15).

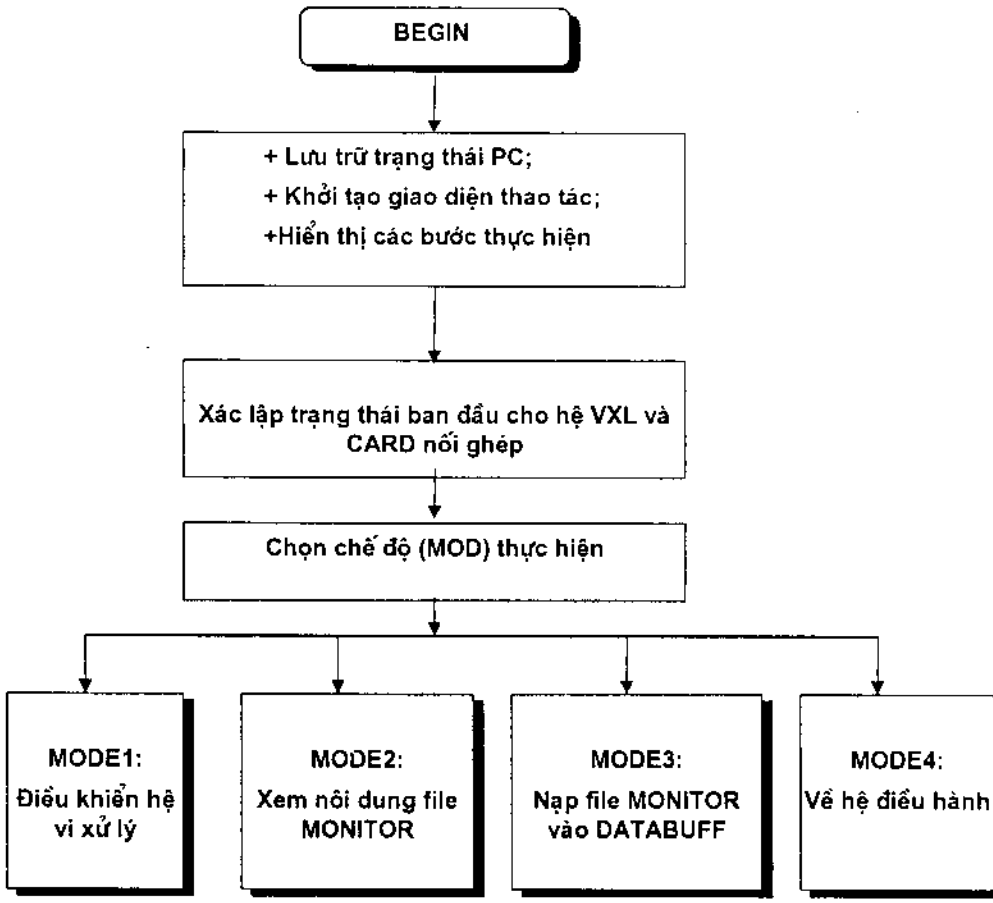
Việc tạo các tín hiệu xung chốt cho các IC1, IC2 (74LS 374) cũng do phần mềm hệ thống tạo ra qua IC5(74LS 138).

IC5 thực hiện các mệnh lệnh từ phần mềm hệ thống qua thanh ghi điều khiển 37AH cổng LPT1 của máy tính để tiến hành các nhiệm vụ như:

- Tạo các xung chốt (bằng sườn dương) vào các chân clock (pin 11) cho các IC1, IC2 bằng cách đưa các chân đó về mức 0 sau đó nâng lên mức 1. Chân CS1 của IC5 được nối với chân clock của IC1, còn CS2 của IC5 được nối với chân clock của IC2.

- Tạo các tín hiệu mở thông các IC6, IC7 để nhập các tín hiệu địa chỉ từ BUS địa chỉ hệ vi xử lý vào máy tính bằng cách lần lượt đưa mức 0

vào các chân 1G (pin 1), 2G (pin 19) ứng với mỗi IC cần mở để chương trình hệ thống đọc giá trị địa chỉ ở nible tương ứng. Chân CS5 của IC5 nối với 1G của IC6/1, chân CS6 nối với 2G của IC6/2, chân CS7 nối với 1G của IC7/1, chân CS8 nối với 2G của IC7/2.



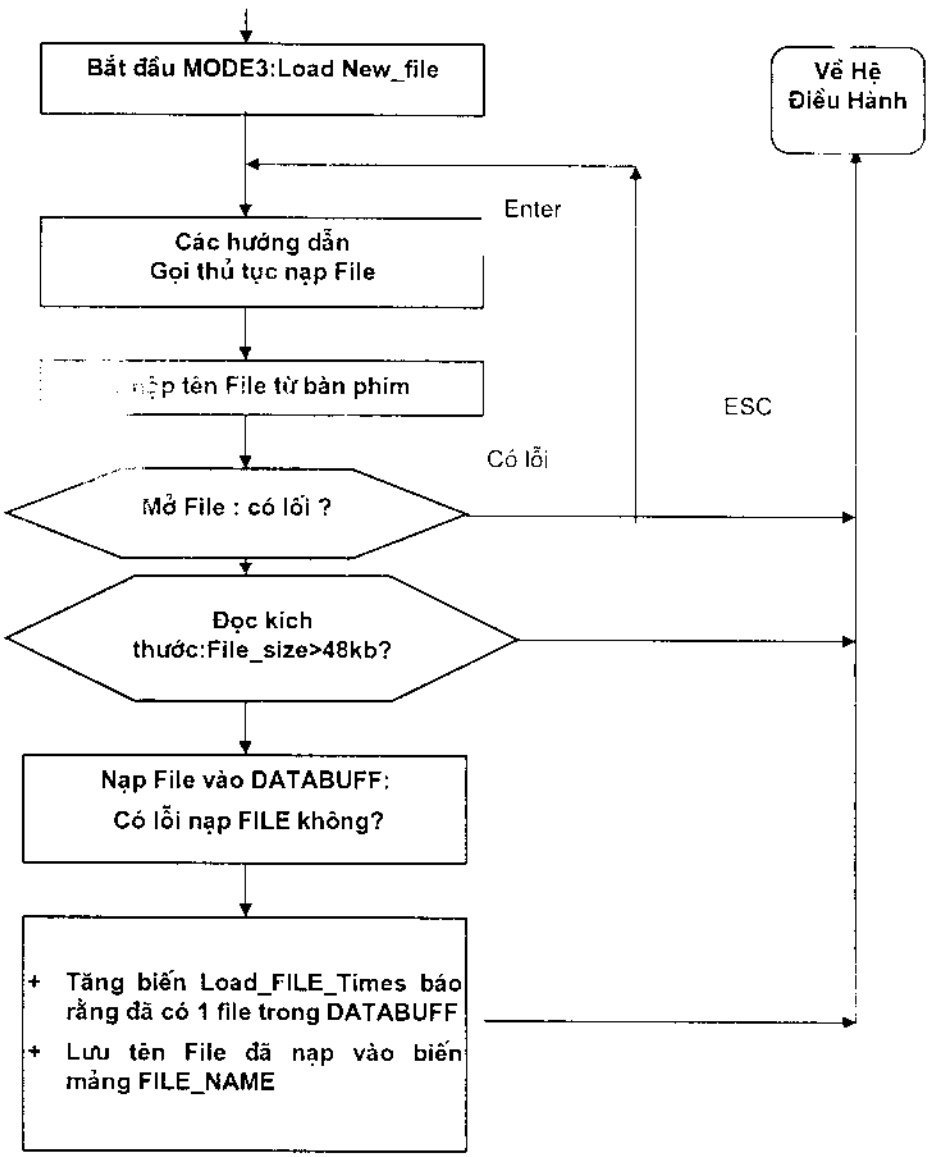
Hình 3.10. Lưu đồ thuật toán chương trình hệ thống

Các chân CS3, CS4 của IC5 không sử dụng.

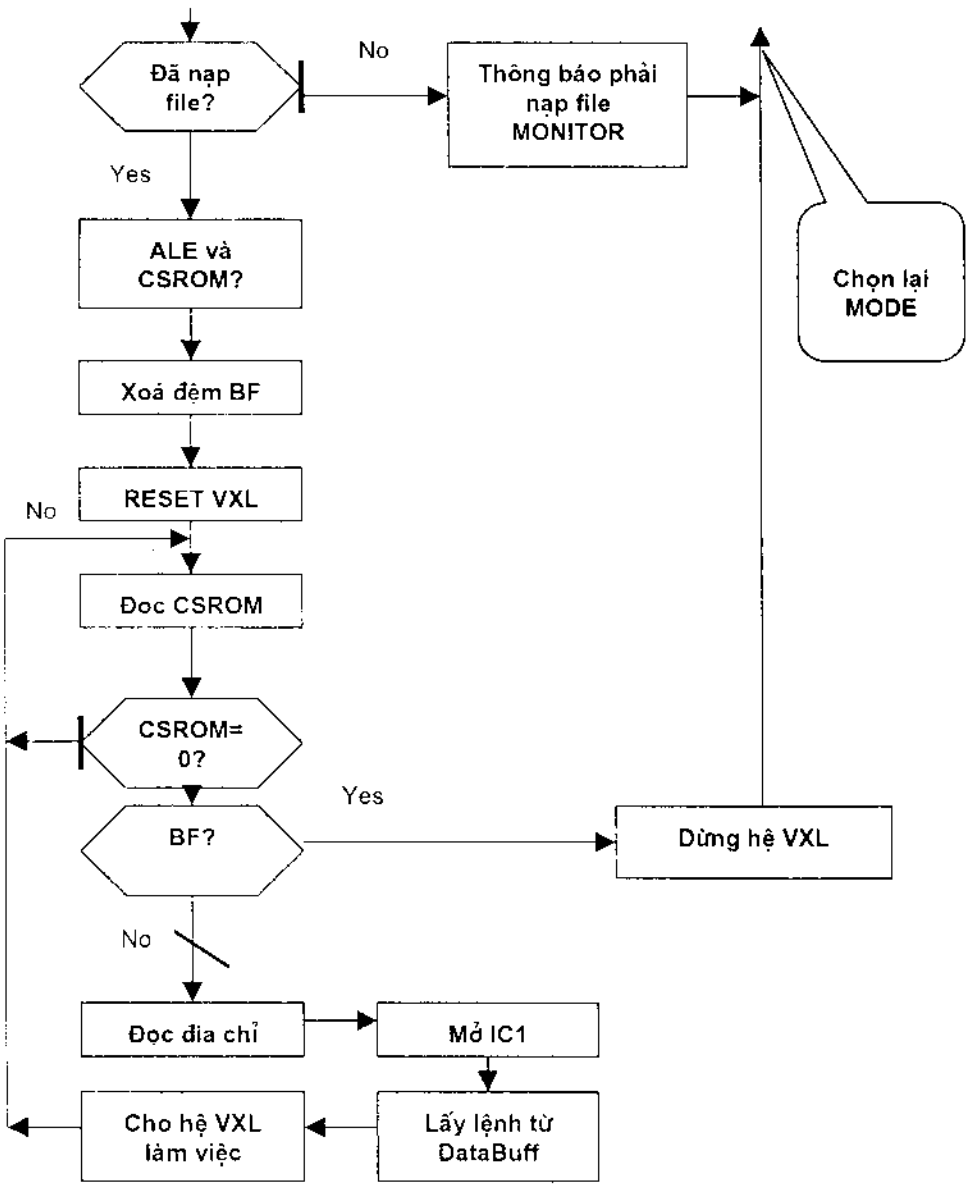
Các chân CS3, CS4 của IC5 không sử dụng.

IC5 làm việc theo nguyên tắc khi chương trình hệ thống cần kích hoạt IC nào trong số các IC1, IC2, IC6/1, IC6/2, IC7/1, IC7/2 thì đầu ra tương ứng của nó được hạ xuống mức 0, sau đó tất cả các đầu ra CS của nó đều ở mức 1. Đó là nhờ có các chân E1, E2 của nó nối vào chân

CTRL/D3 của thanh ghi 37AH(LPT1). Nếu CTRL/D3=0 thì cho phép kích hoạt hay mở thông các IC nói trên, còn khi CTRL/D3=1 thì tất cả các đầu ra CS của IC5 đều ở mức 1. Lưu ý rằng tín hiệu CTRL/D3 được đảo so với bit d3 của thanh ghi 37AH.

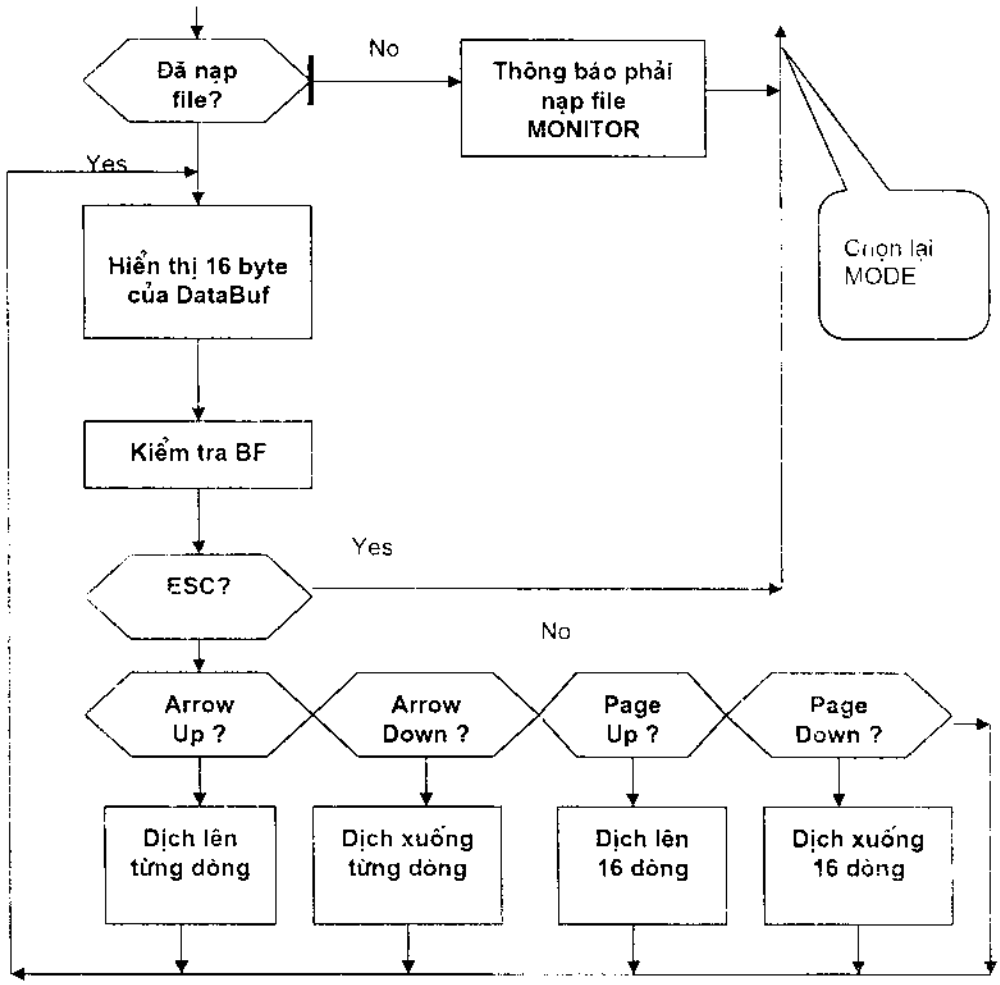


Hình 3.11. Lưu đồ thuật toán MODE3: Load_New_File



Hình 3.12. Lưu đồ thuật toán MODE1

Tín hiệu CSROM =0 ngoài vai trò giúp cho CARD giao tiếp tự động tạo tín hiệu READY=0 như đã nói ở trên, còn được chương trình hệ thống trực tiếp đọc vào qua đường tín hiệu STAT<D3 (bit d3 của thanh ghi 379H) làm cơ sở để chương trình hệ thống thực hiện các thao tác cần thiết trong quá trình mô phỏng.



Hình 3.13. Lưu đồ thuật toán MODE2

3.4. CHƯƠNG TRÌNH HỆ THỐNG

Với phần cứng đã được tổ chức như ở mục trên, ta đã có đầy đủ các tham số hệ thống để xây dựng phần mềm điều khiển cho hệ giao tiếp này.

3.4.1. lưu đồ thuật toán của chương trình hệ thống

Thuật toán điều khiển hệ vi xử lý nối ghép được bắt đầu từ việc khởi tạo giao diện thao tác. Một cửa sổ với các thực đơn được hiển thị ra cho

phép đưa tiếp các điều kiện điều khiển như MODE làm việc, file cần chạy, các cơ cấu thông báo lỗi...Trong PC sẽ chứa các chương trình điều khiển cho hệ vi xử lý chuyên dụng được nối với nó. Các chương trình con này có thể là độc lập cũng có thể là một tập hợp các unit khác nhau cùng nhiệm vụ điều khiển hệ vi xử lý theo chức năng phức tạp nào đó (hình 3.10).

Trong các MODE, thì MODE3 là MODE nạp file Monitor vào buffer dành cho ROM mô phỏng của hệ vi xử lý là rất quan trọng. Lưu đồ hình 3.11 sẽ trình bầy thuật toán điều khiển này. Tiếp theo là lưu đồ cho MODE2 (hình 3.13) là MODE cho phép kiểm tra chính chương trình có trong ROM mô phỏng trước khi đưa ra điều khiển hệ vi xử lý ngoại vi. Cuối cùng là lưu đồ cho MODE1 (hình 3.12) là MODE cho phép điều khiển hệ vi xử lý từ máy tính thông qua chương trình chứa trong ROM mô phỏng.

3.4.2. chương trình hệ thống

Chương trình hệ thống được xây dựng trên ngôn ngữ ASSEMBLY để đảm bảo tốc độ làm việc tối đa. Dưới đây là chương trình nguồn của phần mềm điều khiển này.

```
.model small
.code
;Cac MACRO cần thiết được xây dựng ở phần đầu chương trình:
IF1       ; chỉ duyệt một lần
;-------------------------------------------------------------
@write_str  MACRO row?,col?,attrib?,string?
  LOCAL lap,msg,overmsg
  push AX
  push BX
  push DX
  push SI
  push ES
  mov AX,0B800h
```

```
  mov ES,AX
  lea SI,msg
  mov BX,(160*row?+2*col?)
  mov DH,attrib?
  Loop1:
  mov DL,byte ptr [SI]
  mov word ptr ES:[BX],DX
  add BX,2
  inc SI
  cmp byte ptr [SI],'$'
  jne Loop1
  pop ES
  pop SI
  pop DX
  pop BX
  pop AX
  jmp overmsg
  msg db string?,'$'
  overmsg:
  ENDM
;*********************************
@close   MACRO the_file?
;-----------------------------------------------
;đóng một tập tin có thẻ file là the    a?
;cờ nhớ thiết lập nếu có lỗi
;-----------------------------------------------
  IFNB <the_file?>
  mov BX,the_file?
  ENDIF
  mov AH,3Eh
  int 21h
  ENDM
;************************************************
```

```
@mak_buf    MACRO ten?,do_dai?
;-----------------------------------------
;tạo một vùng đệm bàn phim
;-----------------------------------------
  ten? db do_dai?     ;số tối đa các kí tự
  db ?                ;số ki tự thực nhập
  db do_dai? dup(?)   ;vùng đệm
  db cr               ;kí tự về đầu dòng
  ENDM
;*************************************************
@open MACRO con_tro?,den_dau?
;-------------------------------------------------
;mở một file trên đĩa chỉ để đọc (read only)
;con_tro?:trỏ đến đầu đường tên (cs 0 ở cuối)
;nếu có mã lỗi thì nhảy đến địa chỉ đến đầu?(nếu có)
;BX chứa thẻ file hoặc mã lỗi
;cờ nhớ thiết lập nếu có lỗi
;-------------------------------------------------
  push DX
  mov DX,con_tro?  ;DS:DX trỏ tới dầu đường tên File để
                   ;sử dụng hàm mở file 3Dh
  mov AL,0         ;Al=0 chỉ đọc,không được ghi, Al=1:chỉ ghi
                   ;al=2 vừa đọc vừa ghi
  mov AH,3Dh       ;hàm 3Dh mở file, kết quả:AX chứa thẻ file
  int 21h
  mov BX,AX        ;phải chuyển thẻ file vào BX
  pop DX
  IFNB <den_dau?>  ;nếu có tham số thứ hai, nếu không thì bỏ
  jc den_dau?      ;nếu có lỗi thì cờ CY =1
  ENDIF
  ENDM
;*****************************************************
@read_bf    MACRO dia_chi?
```

```
;-------------------------------------------------------
;đọc từ bàn phím vào một vùng đệm có địa chỉ DS:dia_chi
;cú pháp: @read_bf <offset vung_dem>
;-------------------------------------------------------
  push DX
  mov DX,dia_chi?
  mov AH,0Ah
  int 21h
  pop DX
  ENDM
;*******************************************************
@D_read     MACRO the_file?,con_tro?
;-------------------------------------------------------
;đọc từ đĩa với the_file đã được gán trước (bằng @open)
;con_tro? trỏ đến vùng đệm.
;CX chứa số byte cần đọc vào,
;trước khi gọi cần đặt giá trị cho CX.
;cờ nhớ thiết lập nếu có lỗi.
;-------------------------------------------------------
  push DX
  mov DX,con_tro?     ; DS:DX trỏ tới vùng đệm RAM, nơi
                      ; chứa file
  IFNB <the_file?>    ; duyệt tham số thẻ file
  mov BX,the_file?    ; thẻ file phải đặt trong BX
  ENDIF               ; kết thúc duyệt tham số thẻ file
  mov AH,3Fh          ; khởi động hàm đọc file từ đĩa vào RAM
  int 21h
  pop DX
  ENDM
;*******************************************
  @restore_reg     MACRO regs?
  IRP D,<regs?>       ; Macro lặp IRP Indefinite repeat
                      ; (lặp không xác định)
```

```
   POP D
   ENDM      ; lệnh kết thúc của macro lặp IRP
   ENDM      ; lệnh kết thúc của macro @restore_reg
;***********************************************
@save_reg   MACRO regs?
   IRP D,<regs?>      ; Macro lặp IRP Indefinite repeat
                      ; (lặp không xác định)
   PUSH D
   ENDM               ; lệnh kết thúc của macro lặp IRP
   ENDM
;***********************************************
@go_dos     MACRO
   mov AH,4Ch         ; hàm kết thúc, quay lại trao quyền điều
                      ; khiển cho DOS
   int 21h
   ENDM
;***********************************************
ENDIF                 ; kết thúc khai báo các Macro
;###############################################
;Định nghĩa các hằng kí tự GRAPHIC để vẽ khung
;-----------------------------------------------
   VER_BAR        EQU  0BAh    ;(186d)= thanh đứng
   HOR_BAR        EQU  0CDh    ;(205d)= thanh ngang
   UPPER_LEFT     EQU  0C9h    ;(201d)= khép góc
   UPPER_RIGHT    EQU  0BBh    ;(187d)= khép góc
   LOWER_LEFT     EQU  0C8h    ;(200d)= khép góc
   LOWER_RIGHT    EQU  0BCh    ;(188d)= khép góc
   TOP_T_BAR      EQU  0CBh    ;(203d)= thanh chữ T xuôi
   BOT_T_BAR      EQU  0CAh    ;(202d)= thanh chữ T ngược
   LEFT_T_BAR     EQU  0CCh    ;(204d)=
   RIGHT_T_BAR    EQU  0B9h    ;(185d)=
;+++++++++++++++++++++++++++++++++++++++++++++++
   cr      EQU 13              ; về đầu dòng
```

```
  lf      EQU 10        ; xuống dòng
  escp    EQU 27        ; mã phím ESCAPE
  blank   EQU 32        ; mã khoảng trống
  HOLD    EQU 12h       ; 0001 0010; mã treo CPU 85
;################################################
CODE    SEGMENT         ; mở mảng mã lệnh
  assume CS:CODE,DS:CODE
  org 100h
  START:
  jmp OVER
;################################################
; --------Tổ chức các biến bộ nhớ---------------
;------------------------------------------------
  handle      dw ?    ;Thẻ file
  file_size   dw ?    ;kích thước file
  first_left  dw ?    ;lưu vị tri bắt đầu một dòng cửa sổ
  current_page db ?   ;Trang hiện thời của màn hình
  cursor_x    db ?    ;cột hiện thời của con trỏ
  cursor_y    db ?    ;dòng hiện thời của con trỏ
  file_name db 45 dup(?)
  load_file_times db 0 ;luu số lần file vào datbuff
ADDROM dw ?             ;địa chỉ ngăn nhớ ROM của hệ VXL
;#################################
  OVER:
  call HIDE_CURSOR
  ;
  call Clear_SCR
  call clrmainwd
  call VE_KHUNG
  call get_current_page
  @write_str 6,10,1Eh,'EPROM SIMULATOR PROGRAM'
@write_str 8,10,3Eh,'Availble for 8_bit microprocessor system'
@write_str 9,10,3Eh,'With up to 48 KB EPROM capacity built_in'
```

```
@write_str 19,10,97h,'Press any key to continue !'
call pause
   ;###################################
   ;thực hiện các thao tác cần thiết ban đầu
   ;###################################
   call clrmainwd
   call LOAD_VNFONT
   call VE_KHUNG
   @write_str 8,10,4Eh,'Khởi động CARD giao tiếp'
   @write_str 19,10,9Eh,'ấn phím bất kỳ để tiếp tục'
   call pause
   ;Treo hệ VXL bằng tín hiệu Hold
   mov DX,378h
   mov AL,HOLD
   out DX,AL
   call clock_IC2
   call clrmainwd
   call VE_KHUNG
   @write_str 8,10,4Eh,'Khởi động hệ VXL'
   @write_str 19,10,9Eh,','ấn phím bất kỳ để tiếp tục'
   call pause
   ;################################
   ;Lựa chọn và thực hiện theo MENU
   ;################################
   SELECT_MODE:
   call clrmainwd
   call VE_KHUNG
   call MENU
   cmp cl,'0';Thoat?
   je FOR_EXIT
   cmp cl,'1'
   je FOR_MODE1
   cmp cl,'2'
```

```
    je FOR_MODE2
    cmp cl,'3'
    je FOR_MODE3
    jmp SELECT_MODE
    ;
    FOR_MODE1:
    jmp MODE1
    FOR_MODE2:
    jmp MODE2
    FOR_MODE3:
    jmp MODE3
    FOR_EXIT:
    jmp THOAT
;########################################
;Bắt dầu MODE1: Điều khiển Hệ VXL
;########################################
MODE1: ; D0=/resetCPU85, D1=Hold, D2=/CLK1, D3=/CLK2,
       ; D4=/OE cho IC1-cổng chuyển mã monitor cho VXL
    call clrmainwd; xóa màn hình hiển thị
    call VE_KHUNG
    cmp load_file_times,00h    ;kiểm tra xem file MONITOR đã
                               ; có chưa
    je WARNING1; nếu =0 thì thoát để nạp file vào
    jmp ContinueMODE1;
    WARNING1:
    @write_str 8,10,1Eh,'Phải tải File vào DATBUFF'
    @write_str 9,10,1Eh,'trước khi mô phỏng ROM '
    @write_str 19,10,1Eh,'ấn phím bất kỳ để về MENU chính'
    @write_str 20,10,1Eh,'và tải File cần thiết'
    call pause
    jmp EXIT_MODE1
ContinueMODE1:;
    mov DX,378h
```

```
  mov AL,HOLD       ; hold=0001 0010
  xor AL,04h        ; đảo bit D2=clear1, AL=> 0001 0110
  out DX,AL
  call clock_IC2    ; xuất ra thanh ghi lệnh IC2
  @write_str 8,10,1Eh,' MODE 1đã được chọn'
  @write_str 8,40,9Eh,'đang mô phỏng EPROM'
  @write_str 13,15,1Eh, 'ADDR DATA'
  @write_str 19,10,1Eh,'ấn phím bất kỳ để trở về MENU'
  call clear_kbd;xóa vùng đệm bàn phím
  mov ADDROM, 0000h      ; gán địa chỉ khởi đầu 0 vào biến
                         ; ADDROM
;-----------------------------------------------------
;---RESET hệ vi xử lý cho hold=0=d1, reset=1=d0-------
;-----------------------------------------------------
  xor AL,02h; đảo bit d1=hold, AL= 0001 0100 -hold=>0
  out DX,AL
  call clock_IC2      ; hệ VXL chuẩn bị hoạt động
  call DELAY
  xor AL,01h          ;đảo bit d0=RIN=1-thụ động, AL=0001 0101
  out DX,AL
  call clock_IC2      ; bây giờ ReSetIn=1,
              ;sau khi reset, IC8 tự động bắt ALE tạo READY=0
PERFORM:
  Push AX       ; cất AL= 0001 0101 vào STACK
  mov DX, 379h
  in AL,DX      ; kiểm tra có tín hiệu CSROM =0 (chân STAT<D3)
  and AL, 08h ; 0000 1000;bit D3=CSROM được nhận biết,
              ; nếu bằng 0,->AL=0 -> có CSROM và ZF=1
  jz CONTINUE_PERFORM  ;có, tiếp tục
       ;đảo bit D3 để IC82 bắt ALE kế tiếp.
       ;đảo bit D2 clear IC81 phát ra READY=1,
       ;8085 tiếp tục làm việc.
  mov DX,378h
```

```
  xor AL,0Ch;0000 1100 + AL= 0001 1001
  out DX,AL
  call clock_IC2
  jmp PERFORM
CONTINUE_PERFORM:
    ;có tin hiệu CSROM, vậy dọc địa chỉ
    ;kiểm tra trong quá trình, bàn phím có phím nào bị ấn?
  mov AH,01h
  int 16h          ; bàn phím có bị tác động?
  jnz EXIT_MODE1   ; có, thóat!
             ; bàn phím không bị tác động,vậy thi tiếp tục'
  @write_str 15,15,1Eh,' IN '
  call readADRR ;dọc địa chỉ ROM ra BX
  pop AX         ;lấy lại giá trị AL= 0001 0101
                 ;ENABLE IC1 để dưa mã monitor cho VXL
  mov DX,378h
  xor AL,10h     ;AL=0000 0101 bit d4=/OE của IC1=0 là tích cực
  out DX,AL
  call clock_IC2
  push AX        ; cất AL=0000 0101 vào stack
                 ; xuất dữ liệu ra Data BUS
  mov AL,[offset datbuff+BX]     ; ánh xạ vào datbuff
                 ; để lấy byte lệnh monitor
  out DX, AL     ; dầu ra cổng 378h có dữ liệu
  call clock_IC1; chốt Data BUS -đầu ra IC1 có dữ liệu
  call pause     ; tạm dừng để kiểm tra BUS dữ liệu;
                 ;cho hệ tiếp tục làm việc
  pop AX         ; lấy lại AL=0000 0101 từ stack
  xor AL,1Ch     ; bây giờ AL= 0001 1001
  out DX,AL
  call clock_IC2
  jmp PERFORM    ;lặp lại quá trình
  EXIT_MODE1:
```

```
                ;hold
  mov DX,378h
  mov AL,HOLD
  out DX,AL
  call clock_IC2
  jmp SELECT_MODE
;#####################################
;Bắt đầu MODE2:Xem nội dung chương trình
;#####################################
MODE2:
  call clrmainwd
  call VE_KHUNG_MODE2
  cmp load_file_times,00h
  je WARNING2
  jmp TIEP_MODE2
  WARNING2:
  @write_str 8,10,1Eh,'Cần tải file vào DATBUFF'
  @write_str 9,10,1Eh,'trước khi xem nd của nó'
  @write_str 19,10,1Eh,'ấn fím bất kỳ để về MENU'
  @write_str 20,10,1Eh,'và tải File cần thiết'
  call pause
  jmp EXIT_MODE2
  TIEP_MODE2:
  @write_str 4,10,1Eh,' MODE 2 đã chọn '
  @write_str 6,10,1Eh,'Xem nội dung file trong DATBUFF'
  ;in dong thong bao ten file
  @write_str 8,10,1Eh,'Tên file đang nạp trong DATBUFF:'
  push BX
  push DX
  push SI
  lea SI,file_name
  mov BX,(160*9+2*10)
  mov DH,1Eh
```

```
call DSP_STR
pop SI
pop DX
pop BX
@write_str 10,10,1Eh,'Kích thước:'
mov DX,0A19h;dòng 10 cot 25
call GOTO_XY
mov AX,[file_size]
call out_dec;in ra kich thước file
@write_str 10,35,1Eh,'Bytes'
@write_str 12,6,1Eh,'Tương ứng địa chỉ ROM (hex) từ 0000 đến'
  ;chuyển con trỏ màn hình về cuối dòng thông báo trên
mov DH,12
mov DL,47
call GOTO_XY
mov BX,[file_size]
dec BX
mov AL,BH
call WRITE_HEX
mov AL,BL
call WRITE_HEX
@write_str 19,5,1Eh,'D—ng
     [Up_Arrow],[Down_Arrow](chuyển      từng dòng)'
@write_str 20,5,1Eh,'hoớc [Pg_Up],[Pg_Down](chuyển 16
     dòng) để xem'
@write_str 21,5,1Eh,'ấn [ESC] trở về MENU'
;###################################
;in ra cửa sổ con nội dung của datbuff;
;trỏ bởi BX va SI
;dịch chuyển vùng xem bằng các phím mũi tên,PGUp,PgDOWN.
call CLR_WD
mov DH,4
mov DL,61
```

```
mov BX,0h
lea SI,datbuff
call SHOW_CONTENT
LAP_TEST:
mov AH,0h; hàm số 0 của ngắt 16h
int 16h
cmp AL,27; có phải phím thoát ESC không?
je EXIT_TEST
cmp AH,72
je SCROLL_UP
cmp AH,80
je SCROLL_DOWN
cmp AH,73
je PAGE_UP
cmp AH,81
je PAGE_DOWN
jmp LAP_TEST
EXIT_TEST:
jmp EXIT_MODE2
SCROLL_UP:
cmp BX,1
jae TIEP_UP
jmp LAP_TEST
SCROLL_DOWN:
add BX,16
cmp BX,[file_size]
jb TIEP_DOWN
sub BX,16
jmp LAP_TEST
PAGE_UP:
cmp BX,16
ja TIEPPGUP
jmp LAP_TEST
```

```
   PAGE_DOWN:
   add BX,16
   cmp BX,[file_size]
   jb TIEPPGDOWN
   sub BX,16
   jmp LAP_TEST
   TIEP_UP:
   dec BX
   dec SI
   jmp TIEP_TUC
   TIEP_DOWN:
   sub BX,16
   inc BX
   inc SI
   jmp TIEP_TUC
   TIEPPGUP:
   sub BX,16
   sub SI,16
   jmp TIEP_TUC
   TIEPPGDOWN:
   add SI,16
   TIEP_TUC:
   call CLR_WD
   call SHOW_CONTENT
   jmp LAP_TEST
   EXIT_MODE2:
   jmp SELECT_MODE
;################################################
;Bắt đầu MODE3:Nạp file Monitor vào vùng đệmDatBuff
;################################################
MODE3:
     call clrmainwd
     call VE_KHUNG
```

```
    @write_str 4,10,1Eh,' MODE 3 đã chọn '
    call LOAD_NEW_FILE
    jmp SELECT_MODE
  THOAT:
    call SHOW_CURSOR
    call LOAD_ROM_FONT
    call Clear_SCR
    @go_dos
    ;**********************************
    ;Phần chứa các thủ tục
    ;**********************************
MENU PROC
  @save_reg <AX,BX,DX,ES,SI>
  @write_str 4,28,1Eh,'Thực hiện chương trình'
    ;
    HT0:
  @write_str 8,10,1Eh,'1-Mô phỏng EPROM...'
    ;
  @write_str 10,10,1Eh,'2-Xem nội dung file nạp trong DATBUFF'
    ;
  @write_str 12,10,1Eh,'3-Nạp file vào DATBUFF'
    ;
  @write_str 14,10,7Ch,'4-về DOS'
    ;
    mov cl,'0';trạng thái trở về DOS
    ;
  @write_str 19,10,1Eh,'Dùng các phím số- [1],[2],[3],[4]'
  @write_str 20,10,1Eh,'Hoặc [Up-Arrow],[Down-Arrow] để chọn'
  @write_str 21,10,1Eh,'ấn [ENTER] để thực hiện'
    jmp LAP_MENU
    ;
    HT1:
  @write_str 8,10,7Ch,'1-Mô phỏng EPROM...'
    ;
```

```
@write_str 10,10,1Eh,'2-Xem nội dung file trong DATBUFF'
   ;
@write_str 12,10,1Eh,'3-Nạp file vào DATBUFF'
   ;
@write_str 14,10,1Eh,'4-Về DOS'
   ;
   mov cl,'1'; đổi thực hiện menu1
   jmp LAP_MENU
   ;
   HT2:
@write_str 8,10,1Eh,'1- Mô phỏng EPROM...'
   ;
@write_str 10,10,7Ch,'2- Xem nội dung file trong DATBUFF'
   ;
@write_str 12,10,1Eh,'3- Nạp file vào DATBUFF'
   ;
@write_str 14,10,1Eh,'4- Về DOS'
   ;
   mov cl,'2'
   jmp LAP_MENU
   ;
   HT3:
@write_str 8,10,1Eh,'1- Mô phỏng EPROM...'
   ;
@write_str 10,10,1Eh,'2-Xem nội dung file trong DATBUFF'
   ;
@write_str 12,10,7Ch,'3-Nạp file vào DATBUFF'
   ;
@write_str 14,10,1Eh,'4- Về DOS'
   ;
   mov cl,'3'
   jmp LAP_MENU
   ;
```

```
LAP_MENU:
mov ah,0
int 16h; ngắt bàn phím cho nhập kí tự chọn menu
;
cmp al,13;ENTER?
je SELECT
cmp AL,'1';An phim <1>?
je HL1
cmp AL,'2';Phim <2> ?
je HL2
cmp AL,'3';phim <3> ?
je HL3
cmp AL,'4';phim <4> ?
je HL0
or al,al; Phim chức năng?
jne LAP_MENU;không phải
cmp ah,72
je UP
cmp ah,80
je DOWN
jmp LAP_MENU
;
UP:
cmp cl,'0'
je HL3
cmp cl,'3'
je HL2
cmp cl,'2'
je HL1
cmp cl,'1'
je HL0
;
```

```
    DOWN:
    cmp cl,'1'
    je HL2
    cmp cl,'2'
    je HL3
    cmp cl,'3'
    je HL0
    cmp cl,'0'
    je HL1
    ;
    HL0:
      jmp HT0
    HL1:
      jmp HT1
    HL2:
      jmp HT2
    HL3:
      jmp HT3
    SELECT:
    @restore_reg < SI,ES,DX,BX,AX>
    ret
    MENU ENDP;
;#############################
LOAD_NEW_FILE PROC; nạp file Monitor
  @save_reg <AX,BX,CX,DX,SI,DI>
  LOAD_FILE:
  @write_str 8,10,1Eh,'Tải FILE vào DATBUFF'
  @write_str 18,6,1Eh,'Tên file ([driver:\][path\]
        <file name>):'
  call SHOW_CURSOR ;đặt lại con trỏ dưới dòng
                   ;thông báo để nhập tên file
  mov DX,1306h     ;tại tọa độ dòng 19 cột 6
  call GOTO_XY
```

```
                ;Bây giờ lấy đường tên file Monitor
                ;SI sẽ trỏ đến đầu đường tên nếu không có lỗi
   call get_path1   ;ra: SI trỏ đến đầu đường tên
                    ;DI trỏ đến cuối đường tên
   call HIDE_CURSOR
   @open SI         ; mở file này bằng Macro @OPEN
   jc ERR_OPEN_FILE ;nếu có lỗi mở file thì chuyển tới
                    ;nhãn xử lý
   jmp NO_ERR       ;nếu không lỗi thì tiếp
   ERR_ OPEN _FILE:
   call clrmainwd   ; xóa màn hình
   call VE_KHUNG    ; vẽ lại khung
@write_str 19,10,97h,'Không tìm thấy hoặc không mở được file'
@write_str 20,10,1Eh,'ấn [ENTER] để tiếp tục hay [ESC] về MENU'
   READ_KBD:        ; hỏi bàn phím bằng
   mov AH,08h       ; hàm 8 (đọc không Tab)
   int 21h
   cmp AL,27        ;là phím ESC?
   je NO_LOAD_AGAIN ; đúng, không nạp nữa mà phải thoát ra
   cmp AL,cr        ; là phím ENTER?
   je LOAD_AGAIN1   ; đúng, nạp file
   jmp READ_KBD
   NO_ERR:
   jmp NO_ERR1
   NO_LOAD_AGAIN:
     call clrmainwd
     call VE_KHUNG
     jmp EXIT_LOAD2
   LOAD_AGAIN1:
     call clrmainwd
     call VE_KHUNG
     jmp LOAD_FILE ;nạp lại tên file cho đúng
   NO_ERR1:         ;bây giờ BX đang chứa thẻ file
```

```
;-------------------------------------------------;
Hàm 42h dịch chuyển con trỏ file tới vị trí mới (CX:DX) byte
tính từ vị trí chỉ;định trong AL: (AL)=0(đầu file, (AL)=1(vị
trí hiện hành, (AL)=2(cuối file.;(DX:AX) chứa số byte tính từ
đầu file. Vì vậy nếu(AL)=2 thì (DX:AX) chứa;kích thước file.
;-------------------------------------------------
mov [handle],BX    ;lưu thẻ file vào biến handle
   xor dx,dx    ;(CX:DX) chứa số bước cần chuyển
   xor cx,cx    ; hiện tại số bước=(CX:DX) =0
   mov ah,42h
   mov al,02h  ;trỏ tới cuối file
   int 21h      ; bây giờ (DX:AX) đang chứa kích thước file
   mov [file_size],AX ;lưu kích thước file vào biến file_size
   call clrmainwd
   call VE_KHUNG
   @write_str 10,10,1Eh,'Kích thước file vừa mở là:'
              ;dịch chuyển con trỏ màn hình về
              ;cuối dòng thông báo trên (27 kí tự)
   mov DX,0A28h; dòng 10 cột 40
   call GOTO_XY
   mov AX,[file_size]
   call out_dec     ;in ra kích thước file
   @write_str 10,50,1Eh,'Bytes'
   cmp AX,48*1024   ;kích thước file_size  =48 KB?
   ja WARNING3
   jmp READ_FILE
   WARNING3:
   @write_str 14,10,9Eh,'File quá lớn không tải vào DATBUFF !'
   @write_str 19,10,1Eh,' tải file kích thước <= 48 KB'
   @write_str 21,10,1Eh,'ấn phím bất kỳ trở về MENU'
   call pause
   @close [handle]
   jmp EXIT_LOAD2
```

```
  READ_FILE:
  xor dx,dx
  xor cx,cx
  mov BX,[handle]
  mov ah,42h
  mov al,00h
  int 21h;chuyển con trỏ file về đầu file chuẩn bị đọc vào
  mov CX,[file_size] ;số lượng byte cần đọc
  @d_read [handle],<offset datbuff>
                    ;dùng MACRO này để đọc File
  jc FOR_ER
  @write_str 19,10,9Eh,'ấn phím bấtt kỳ để tiếp tục'
  call pause
  jmp CONTINUE_LOAD
FOR_ER:
  jmp ER
CONTINUE _LOAD:
  call clrmainwd
  call VE_KHUNG
  @write_str 10,10,1Eh,'FILE đã được tải vào DATBUFF'
  @close [handle]
  jmp EXIT_LOAD
ER:
  call clrmainwd
  call VE_KHUNG
  @write_str 19,10,9Eh,'Error,không tải dược file vào datbuff'
  @write_str 21,10,1Eh,'ấn phím bất kỳ về MENU'
  @close [handle]
  jmp EXIT_LOAD2
  EXIT_LOAD:
  call get_filename_str
  inc load_file_times
  @write_str 20,10,9Eh,''ấn phím bất kỳ về MENU'
```

```
   call pause
   EXIT_LOAD2:
   @restore_reg <DI,SI,DX,CX,BX,AX>
   ret
   LOAD_NEW_FILE ENDP
   ;####################
DSP_STR PROC
        ;Thủ tục hiển thị một chuỗi ký tự
        ;bằng truy nhập trực tiếp bộ nhớ màn hình
        ;Vào:
        ;Chuỗi hiện thị:si=offset chuỗi
        ;chuỗi cần phải kết thúc bằng '$'
        ;Thuộc tính của chuỗi:dh
        ;(dl=[si])
        ;Vị trí hiện thị: bx
   @save_reg < AX,ES>
   mov ax,0b800h
   mov es,ax
   LAP5:
   mov dl,byte ptr [si]
   mov word ptr ES:[bx],dx
   add bx,2
   inc si
   cmp byte ptr [si],'$'
   jne LAP5
   @restore_reg <ES,AX>
   RET
   DSP_STR ENDP
;###################################
DISP_CHAR PROC
;Thủ tục in một ki tự ra màn hình bằng cách truy nhập trực
;tiếp bộ nhớ màn hình.
;vào:BX:vị trí trên màn hình:
```

```
;BX=(160*dong)+(cot*2)(dong=0-24,cot=0-79)
;DL:Ma ASCII cua ki tu
;DH:thuộc tính ki tự.
 @save_reg < AX,ES>
 mov AX,0B800h
 mov ES,AX
 mov word ptr ES:[BX],DX
 @restore_reg <ES,AX>
 ret
 DISP_CHAR ENDP
;###################
 VE_KHUNG PROC
   ;thủ tục vẽ một khung chữ nhật trên màn hình
   ;dùng các ki tự graphic
   ;sử dụng DISP_CHAR.
   @save_reg < AX,BX,CX,DX >
   mov DH,(16*1+14)    ; thuộc tính chữ vàng nền xanh
   mov DL,UPPER_LEFT
   mov BX,(160*1+3*2) ;dòng 1,cột 3
   call DISP_CHAR
   add BX,2            ;vị trí sau
   mov CX,70
   VE_HOR_BAR1:
   mov DL,HOR_BAR
   call DISP_CHAR
   add BX,2            ; vị trí tiếp sau
   loop VE_HOR_BAR1
   mov DL,UPPER_RIGHT
   call DISP_CHAR
   dong =2
   mov DL,VER_BAR
   mov CX,20
   mov BX,(160*dong+3*2)
```

```
VE_VER_BAR1:
call DISP_CHAR
add BX,160
loop VE_VER_BAR1
mov CX,20
mov BX,(160*dong+74*2)
VE_VER_BAR2:
call DISP_CHAR
add BX,160
loop VE_VER_BAR2
mov DL,LOWER_LEFT
mov BX,(160*22+3*2)   ;dòng 22, cột 3
call DISP_CHAR
add BX,2; vị trí sau
mov CX,70
VE_HOR_BAR2:
mov DL,HOR_BAR
call DISP_CHAR
add BX,2              ; vị trí tiếp sau
loop VE_HOR_BAR2
mov DL,LOWER_RIGHT
call DISP_CHAR
                      ;vẽ ký tự LEFT_T_BAR (dòng 17 cột 3)
mov DL,LEFT_T_BAR
mov BX,(160*17+3*2)
call DISP_CHAR
mov DL,HOR_BAR
mov CX,70
mov BX,(160*17+4*2)
VE_HOR_BAR3:
call DISP_CHAR
add BX,2
loop VE_HOR_BAR3
```

```
   ; vẽ ký tự RIGHT_T_BAR (dòng 17 cột 74)
   mov DL,RIGHT_T_BAR
   mov BX,(160*17+2*74)
   call DISP_CHAR
   @restore_reg <DX,CX,BX,AX>
   ret
   VE_KHUNG ENDP
   ;###################
VE_KHUNG_MODE2 PROC
   ; thủ tục vẽ một khung chữ nhật trên màn hình cho MODE 2
   ;dùng các ki tự graphic
   ;sử dụng DISP_CHAR
   @save_reg < AX,BX,CX,DX,SI >
   mov DH,(16*1+14)    ; thuộc tính chữ vàng nền xanh
                       ;vẽ góc trái:
   mov DL,UPPER_LEFT
   mov BX,(160*1+3*2) ;dòng 1, cột 3
   call DISP_CHAR
;vẽ đường ngang trên cùng:(dòng 1,cột 4->73)
   add BX,2            ;vị trí sau UPPER_LEFT
   mov CX,70           ;dịch chuyển 70 ki tự
   VE_HOR_BAR21:
   mov DL,HOR_BAR
   call DISP_CHAR
   add BX,2            ; vị trí tiếp sau
   loop VE_HOR_BAR21
   ;vẽ góc phải:(tọa độ 1,74)
   mov DL,UPPER_RIGHT
   call DISP_CHAR
   dong =2
   ;vẽ đường đứng bên trái:(dòng 2->22,cột 3)
   mov DL,VER_BAR
   mov CX,20           ;dịch chuyển 20 dòng
```

```
mov BX,(160*dòng+3*2)
VE_VER_BAR21:
call DISP_CHAR
add BX,160
loop VE_VER_BAR21
; vẽ thanh đứng bên trái:(dòng 2->22, cột 74)
mov CX,20       ; dịch chuyển 20 dòng
mov BX,(160*dong+74*2)
VE_VER_BAR22:
call DISP_CHAR
add BX,160
loop VE_VER_BAR22
; vẽ góc dưới bên trái
mov DL,LOWER_LEFT
mov BX,(160*22+3*2);dòng 22,cột 3
call DISP_CHAR
; vẽ đường ngang bên dưới
add BX,2;vi trí sau LOWER_LEFT
mov CX,70
VE_HOR_BAR22:
mov DL,HOR_BAR
call DISP_CHAR
add BX,2;vi trí tiếp sau
loop VE_HOR_BAR22
; vẽ góc dưới bên phải:(22,74)
mov DL,LOWER_RIGHT
call DISP_CHAR
;vẽ thanh LEFT_T_BAR (dòng 17 cột 3)
mov DL,LEFT_T_BAR
mov BX,(160*17+3*2)
call DISP_CHAR
; vẽ đường ngang ở giữa (dòng 17)
mov DL,HOR_BAR
```

```
mov CX,56
mov BX,(160*17+4*2)
VE_HOR_BAR23:
call DISP_CHAR
add BX,2
loop VE_HOR_BAR23
;vẽ thanh TOP_T_BAR: (dòng 1, cột 59)
mov BX,(160*1+2*59)
mov DL,TOP_T_BAR
call DISP_CHAR
;vẽ thanh đứng giữa (cột 59)
mov DL,VER_BAR
mov CX,20;dịch chuyển 20 dòng
mov BX,(160*dong+2*59)
VE_VER_BAR23:
call DISP_CHAR
add BX,160
loop VE_VER_BAR23
;vẽ thanh BOT_T_BAR: (dòng 21, cột 59)
mov DL,BOT_T_BAR
call DISP_CHAR
;
;vẽ thanh RIGHT_T_BAR (dòng 17, cột 59)
mov DL,RIGHT_T_BAR
mov BX,(160*17+2*59)
call DISP_CHAR
;hiển thị các tiêu đề trong cửa sổ nội dung file
@write_str 2,61,0Ch,'ADDR'
@write_str 2,68,0Ch,'DATA'
;Vẽ đường ngang dưới tiêu đề cửa sổ nội dung file
; vẽ thanh LEFT_T_BAR(dòng 3 cột 59)
mov DL,LEFT_T_BAR
mov BX,(160*3+2*59)
```

```
   call DISP_CHAR
   add BX,2;ví trí tiếp sau
   mov CX,14
   mov DL,HOR_BAR
   VE_HOR_BAR24:
   call DISP_CHAR
   add BX,2
   loop VE_HOR_BAR24
   ;ve ki tu RIGHT_T_BAR
   mov DL,RIGHT_T_BAR
   mov BX,(160*3+2*74)
   call DISP_CHAR
   @restore_reg <SI,DX,CX,BX,AX>
   ret

   VE_KHUNG_MODE2 ENDP
   ;*********************+
CLR_WD PROC
   ;thủ tục xóa cửa sổ địa chỉ-dữ liệu ROM
   @save_reg <AX,BX,CX,DX>
   mov AX,0600h
   mov BH,3Eh;nền xanh chữ vàng
   mov CX,043Dh;góc trên bên trái
   mov DX,1347h; góc dưới bên phải
   int 10h
   @restore_reg <DX,CX,BX,AX>
   ret
   CLR_WD ENDP
   ;++++++++++++++++++++++++
   clrmainwd PROC
   @save_reg <AX,BX,CX,DX>
   mov AX,0600h
   mov BH,17h
```

```
   mov CX,0102h
   mov DX,164Bh
   int 10h
   @restore_reg <DX,CX,BX,AX>
   ret
   clrmainwd ENDP
;*********************
SHOW_CONTENT PROC
;thủ tục hiển thị địa chỉ ROM (BX) và nội dung ([SI])
   @save_reg <AX,BX,CX,DX,SI>
   mov CX,16
   newbyte:
   cmp BX,[file_size]
   je EXIT_SHOW
   call GOTO_XY
   mov AL,BH
   call WRITE_HEX
   mov AL,BL
   call WRITE_HEX
   add DL,8
   call GOTO_XY
   mov AL,[SI]
   call WRITE_HEX
   inc SI
   inc BX
   inc DH
   mov DL,61
   loop newbyte
   EXIT_SHOW:
   @restore_reg <SI,DX,CX,BX,AX>
   ret
   SHOW_CONTENT ENDP
;##########################################
```

```
Clear_SCR PROC;Thủ tục xóa màn hình
   @save_reg <AX,BX,CX,DX>
   mov AH,6;hàm cuộn lên
   xor AL,AL;xóa cả màn hình
   xor CX,CX;góc trên trái(0,0)
   mov DX,184Fh; góc dưới bên phải (4Fh,18h)
   mov BH,7;thuộc tính video thường
   int 10h
   @restore_reg <DX,CX,BX,AX>
   RET
   Clear_SCR ENDP
;*******************************
Get_current_page PROC
   ;nhận biết trang màn hình hiện thời.
   @save_reg <AX,BX>
   mov AH,0Fh;Hàm lấy trạng thái màn hình
   int 10h; Trả về trang màn hình hiện thời trong BH
   mov current_page,BH;Ghi vào biến tổng thể
   @restor_reg < BX, AX>
   ret
   Get_current_page ENDP
;#####################################
clear_kbd PROC
;thủ tục xóa vùng đệm bàn phím
   push AX
   lap_kt:
   mov AH,01h
   int 16h
   jz khong_co
   mov AH,00h
   int 16h
   jmp lap_kt
   khong_co:
```

```
   pop AX
   ret
   clear_kbd ENDP
;*************************
handl DW ?;thẻ file
   @mak_buf buff_in,45;tạo vùng đệm phụ có tên buff_in
   ;*************************
   ;Thủ tục get_name lấy đường tên từ bàn phím
   ;Vùng đệm phụ là buff_in vừa định nghĩa
   ;Vào: DI trỏ đến đầu buff_in
   ;Ra:DI trỏ đến kí tự đầu tiên của tên
   ;CX:số kí tự đã nhập
   ;*************************
get_name PROC
   @read_bf DI
   inc DI; bỏ qua số kí tự tối đa
   mov CL,[DI]; số kí tự thực nhập
   mov CH,0
   inc DI;DI trỏ đến kí tự nhập đầu tiên
   RET
   get_name ENDP
;*****************************
;Thủ tục scan_pa:
;Xử lý đường tên:bỏ đi khoảng trắng thừa
;thêm 0 vào cuối thành xâu ASCIIZ
   ;Vào:DI trỏ đến kí tự đầu tiên của đường tên
   ; CX:số kí tự đã nhập.
   ;RA:SI trỏ đến kí tự đầu tiên của đường tên.
   ; DI trỏ đến 0 ở cuối tên.
   ;Thiết lập cờ nhớ nếu có lỗi.
;-------------------------------------------------------
scan_pa PROC
   mov AL,blank
```

```
        ;chuyển con trỏ qua các khoảng trắng
REPE SCASB;tìm kí tự đầu tiên khác ' ' cho tới khi ZF hạ-(0,
        ; DI tăng để chỉ vào ký tự tiếp theo,CX giảm
  dec DI;DI trỏ đến kí tự này
  inc CX;CX=số kí tự
        ;Cho SI trỏ đến kí tự đầu tiên khác khoảng trắng
  mov SI,DI
  cmp AL,[SI]
  jz no_parms; chỉ toàn khoảng trắng.
     ; tìm kí tự cuối cùng của đường tên là khoảng trắng
     ;hoặc kí tự cr.
     ;Al vẫn đang chứa kí tự blank
     ;nhưng CX sẽ bằng 0 khi đến kí tự cr.
  rep scasb
     jcxz sdi;cr; nếu CX=0 rồi mà vẫn không gặp 0 thì
;thêm 0 vào cuối đường tên
  dec DI;blank
 sdi:;thêm 0 vào cuối đường tên, để ý thấy CH đang =0
  mov [DI],CH
  inc DI; bây giờ DI trỏ đến 0 ở cuối tên.
  clc;xóa cờ nhớ,không lỗi
  RET
  no_parms:;đặt cờ nhớ vì đường tệp không phù hợp
  STC
  RET
  scan_pa ENDP
;*****************+
get_path1 PROC
  ;
  mov DI,offset buff_in;trỏ đến vùng đệm phụ
  call get_name
  call scan_pa
  ret
```

```
   get_path1 ENDP
   ;*************************
get_filename_str PROC
      ;thủ tục truy nhập buff_in
      ;lấy tên file lưu vào mảng file_name
      ;thêm '$' vào cuối tên để sử dụng khi gọi DSP_STR
   @save_reg <CX,SI,DI>
   pushF
   CLD;xóa cờ định hướng để xử lý chuỗi từ trái qua phải
   mov DI,offset file_name;chuỗi đích
   mov SI,offset buff_in;chuỗi nguồn ở vùng đệm phụ
   inc SI;bỏ qua byte lưu số ki tự tối đa
   mov CL,[SI];CL=số ki tự thực nhập
   mov CH,0
   inc SI;SI trỏ đến ki tự nhập đầu tiên
   REP MOVSB
   mov byte ptr [DI],'$';thêm ki tu '$' vào cuối chuỗi
file_name
   popf
   @restore_reg <DI,SI,CX>
   ret
   get_filename_str ENDP
   ;*************************
GOTO_XY PROC
   ;thủ tục dịch chuyển con trỏ về vi trí (x,y)
   ;vào: DH=Y (dòng)
   ; DL=X (cột)
   ; current_page:trang hiện thời
   @save_reg <AX,BX>
   mov BH,current_page
   mov AH,2
   int 10h
   @restore_reg < BX, AX>
```

```
   ret
   GOTO_XY ENDP
   ;*********************+
GET_CURSOR_POSITION PROC
      ;thủ tục xác định vi trí con trỏ hiện thời
      ;Vao: current_page
      ;RA: cursor_x
      ; cursor_Y
@save_reg <AX,BX,CX,DX>
   mov AH,3
   mov BH,current_page
   int 10h
   mov cursor_y,DH;dòng
   mov cursor_x,DL;cột
   @restorre_reg <DX,CX,BX, AX >
   ret
   GET_CURSOR_POSITION ENDP
   ;***********************
out_dec PROC
      ;in giá trị AX ra dạng thập phân
      ;tại vị trí con trỏ màn hình.
   @save_reg <AX,BX,CX,DX>
 xor CX,CX; CX đếm các chữ số thập phân
   mov BX,10d; BX chứa số chia
   repeat1:
   xor DX,DX; chuẩn bị word cao cho số bị chia
   div BX; Ax=thương số,DX=số dư
   push DX; cất số dư vào ngăn xếp
   inc CX;Dem=Dem+1
   or AX,AX; thương số=0?
   jnz repeat1; chưa,tiếp tục chia
   ;lấy vị trí hiện thời của con trỏ
   call GET_CURSOR_POSITION
```

```
   mov DH,cursor_y
   mov DL,cursor_x
   ;Đổi các chữ số thành ký tự khi đã chia xong
   ;FOR Dem DO
   print_loop:
   pop AX;chuyển các số dư trong ngăn xếp vào AL
   or AL,30h; Đổi nó thành ki tự
   push CX
   mov AH,0Ah;hàm in ki tự thuộc tính bình thường tại con trỏ
   mov CX,1;một lần in ki tự
   mov BH,current_page
   int 10h; in ra
   pop CX;lấy lại CX cho vòng loop
   mov AH,02h; hàm đặt lại con trỏ
   inc DL
   int 10h; dịch con trỏ sang phải
   loop print_loop;lặp cho đến khi xong
   ;END_FOR
     ;phục hồi các thanh ghi
   @restorre_reg <DX,CX,BX,AX >
   RET
   out_dec ENDP
   ;**************************
pause PROC
   ;thủ tục cho ấn phím bất kì để tiếp tục
   push AX
   mov AH,08h
   int 21h
   pop AX
   ret
   pause ENDP
   ;*********************************
write_hex_digit PROC
```

```
    ;thủ tục đổi 4 bit thấp của AL thành 1 chữ số hex
    ;in ra màn hình tại vị trí con trỏ
    ;đã định vị bằng goto_XY proc.
    ;Con trỏ màn hình tự động dịch
    ;sang phải một ki tự cho in tiếp.
  @save_reg <AX,BX,CX,DX>
  cmp AL,10;AL<10?
  jae HEX_LETTER;không (A_F)
  add AL,'0';có, đổi nó ra chữ số (0_9)
  jmp WRITE_DIGIT;in ki tự này
  HEX_LETTER:
  add AL,'A'-10; đổi nó ra chữ số hex
  WRITE_DIGIT:
  mov AH,0Ah
  mov BH,current_page
  mov CX,1;một lần in
  int 10h
  ;dịch con trỏ sang phải một ki tự
  call GET_CURSOR_POSITION
  mov DH,cursor_y
  mov DL,cursor_x
  inc DL
  mov AH,2
  mov BH,current_page
  int 10h
  @restorre_reg <DX,CX,BX,AX >
  ret
  write_hex_digit ENDP
  ;####################
write_hex PROC
    ;thủ tục đổi byte có trong AL
    ;ra dạng hex, in 2 ki tự hex của byte do
    ;sử dụng write_hex_digit PROC
```

```
@save_reg <AX,CX>
mov AH,AL; sao byte vào AH
   mov CX,04
   shr AL,CL; dịch phải 4 lần để lấy 4 bit cao
   call write_hex_digit; in ra chữ số hex thứ nhất
   mov AL,AH; lấy lại byte cần in
   and AL,0Fh; cho 4 bit cao bằng 0
   call write_hex_digit; in chữ số hex thứ hai
   @restorre_reg <CX,AX>
   ret
   write_hex ENDP
   ;*******************************
 HIDE_CURSOR PROC
   ;thủ tục làm không hiện con trỏ màn hình
    @save_reg < AX, CX>
    mov AH,1
    mov CL,0Dh;dòng quét đầu
    mov CH,20h;đặt bit 5 =1
    int 10h
    @restorre_reg <CX,AX>
    ret
    HIDE_CURSOR ENDP
   ;*****************************************
SHOW_CURSOR PROC
     ;thủ tục hiện lại con trỏ màn hình
@save_reg < AX, CX>
   mov AH,1
   mov CL,0Dh; dòng quét đầu
   mov CH,0Dh; dòng quét cuối
   int 10h
   @restorre_reg <CX,AX>
 ret
   SHOW_CURSOR ENDP
```

```
   ;##################################
   ; các thủ tục giao tiếp với
; hệ vi xử lý qua card giao tiếp
   ;++++++++++++++++++++++++++++++
clock_IC1 PROC
     ; thủ tục để chốt IC1(74LS374), dl
   @save_reg < AX, DX>
   mov DX,37Ah
   mov AL,03h;
   out DX,AL;
   mov AL,03h
 out DX,AL
 or AL,08h
 out DX,AL
 AND AL,00h
 out DX,AL
 @restorre_reg <DX,AX>
    ret
   clock_IC1 ENDP
   ;++++++++++++++++++++++++++++++
clock_IC2 PROC
       ; thủ tục chốt IC2(74LS374), dk
   @save_reg < AX, DX>
   mov DX,37Ah
   mov AL,02h;
   out DX,AL;
 mov AL,02h
 out DX,AL
 or AL,08h
 out DX,AL
 AND AL,00h
 out DX,AL
    @restorre_reg <DX,AX>
```

```
 ret
   clock_IC2 ENDP
   ;++++++++++++++++++++++++++++++
   ;++++++++++++++++++++++++++++++
pass_IC61 PROC
        ; thủ tục mở IC6/1(74LS244) để nhập
        ; 4 bit thấp (byte địa chỉ thấp)
 @save_reg < AX, DX >
   mov DX,37Ah
   mov AL,07h;
   out DX,AL;
 mov AL,07h
 out DX,AL
 or AL,08h
 out DX,AL
@restorre_reg <DX,AX>
ret
pass_IC61 ENDP
;++++++++++++++++++++++++++++++
pass_IC62 PROC
        ; thủ tục mở IC6/2(74LS244) để nhập
        ;4 bit cao (byte địa chỉ thấp)
   @save_reg < AX, DX >
   mov DX,37Ah
   mov AL,06h;
   out DX,AL;
 mov AL,06h
 out DX,AL
 or AL,08h
 out DX,AL
@restorre_reg <DX,AX>
 ret
 pass_IC62 ENDP
```

```
   ;+++++++++++++++++++++++++++++
  pass_IC71 PROC
        ; thủ tục mở IC7/1(74LS244) để nhập
        ;4 bit thấp (byte địa chỉ cao)
   @save_reg < AX, DX>
   mov DX,37Ah
   mov AL,05h;
   out DX,AL;
 mov AL,05h
 out DX,AL
 or AL,08h
 out DX,AL
@restorre_reg <DX,AX>
ret
pass_IC71 ENDP
   ;+++++++++++++++++++++++++++++
pass_IC72 PROC
        ; thủ tục mở IC7/2(74LS244) để nhập
        ;4 bit cao (byte địa chỉ cao)
@save_reg < AX, DX >
   mov DX,37Ah
   mov AL,04h;
   out DX,AL;
 mov AL,04h
 out DX,AL
 or AL,08h
 out DX,AL
   @restorre_reg <DX,AX>
   ret
   pass_IC72 ENDP
   ;+++++++++++++++++++++++++++++
readADRR PROC
     ; thủ tục dọc vào địa chỉ ROM của hệ vi xử lý
```

```
     ;ra:BX chứa địa chỉ ROM
   @save_reg <AX, CX,DX >
 xor AX,AX
 xor BX BX
   mov CL,4h; để dịch trái 4 lần
   mov DX,379h
   call pass_IC62
   in AL,DX; đọc vào 4 bit cao của byte thấp
   xor AL,80h;1000 0000? đảo bit 7 vì d7 của cổng 379h là mức
đảo
   shl AX,CL;chuyển 4 bit cao của AL sang AH
   call pass_IC61
   xor AL,AL
   in AL,DX; đọc vào 4 bit thấp của byte thấp
   xor AL,80h;dao bit 7
   shl AX,CL;bây giờ byte thấp nằm trong AH
   mov BL,AH; cất byte thấp vào BL
   call pass_IC72
   xor AL,AL
   in AL,DX; đọc vào 4 bit cao của byte cao
   xor AL,80h; 10000000b?dao bit 7
   shl AX,CL; chuyển 4 bit cao của AL sang AH
   call pass_IC71
   xor AL,AL
   in AL,DX; đọc vào 4 bit thấp của byte cao
   xor AL,80h;10000000b?dao bit 7
   shl AX,CL;bây giờ byte cao nằm trong AH
   mov BH,AH;cất byte cao vào BH,
AND BH,1Fh
mov ADDROM,BX;bây giờ BX chứa địa chỉ ngăn nhớ cần qui
;chiếu trong ROM VXL
@restorre_reg <DX,CX,AX>
 ret
```

```
readADRR ENDP
   ;++++++++++++++++++++++++++++++
 DELAY PROC
 @save_reg <AX,CX,DX >
 mov AH,86H
 mov CX,0
 mov DX,1000d; trễ 700 microsec
 int 15h
 @restorre_reg <DX,CX,AX>
 ret
 DELAY ENDP
;++++++++++++++++++++++++++++++++++
LOAD_VNFONT PROC
   @save_reg <AX,BX,CX,DX,BP>
   mov AH,11h
   mov AL,10h
   mov BH,16
   mov BL,0
   mov CX,185;để dành các ký tự từ 185->188 để vẽ khung
mov BP,offset VNFONT; trỏ tới font Việt nam của chương trình
   mov DX,0
   int 10h
   mov BP,offset (VNFONT+16*189)
   mov CX,11
   mov DX,189
   int 10h
   mov BP,offset (VNFONT+16*206)
   mov CX,49
   mov DX,206
   int 10h
   @restorre_reg <BP.DX,CX,BX,AX>
   ret
   LOAD_VNFONT ENDP
```

```
   ;####################
LOAD_ROM_FONT PROC
   @save_reg <AX,BX>push AX
   push BX
   mov AH,11h
   mov AL,4; tải font chữ 8*16 của ROM vào VGA
   mov BL,0
   int 10h
   @restorre_reg <BX,AX>
   ret
   LOAD_ROM_FONT ENDP
   ;######################
   ;######################
VNFONT label byte
   DB 0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h
   db 18h,18h,18h,18h,18h,18h,18h,0ffh,18h,18h,18h,18h,18h,18h,0h,0h
   db 18h,18h,18h,18h,18h,18h,18h,1fh,18h,18h,18h,18h,18h,18h,0h,0h
   db 0h,0h,0h,0h,0h,6ch,0feh,0feh,0feh,0feh,7ch,38h,10h,0h,0h,0h
   db 0h,0h,0h,0h,0h,10h,38h,7ch,0feh,7ch,38h,10h,0h,0h,0h,0h
   db 0h,0h,0h,0h,18h,3ch,3ch,0e7h,0e7h,0e7h,18h,18h,3ch,0h,0h,0h
   db 0h,0h,0h,0h,18h,3ch,7eh,0ffh,0ffh,7eh,18h,18h,3ch,0h,0h,0h
   db 0h,0h,0h,0h,0h,0h,0h,18h,3ch,3ch,18h,0h,0h,0h,0h,0h
   db 0ffh,0ffh,0ffh,0ffh,0ffh,0e7h,0c3h,0c3h,0e7h,0ffh,0ffh, 0ffh,
      0ffh,0ffh,0h,0h
   db 0h,0h,0h,0h,0h,0h,3ch,66h,42h,42h,66h,3ch,0h,0h,0h,0
   db 0ffh,0ffh,0ffh,0ffh,0c3h,99h,0bdh,0bdh,99h,0c3h,0ffh,0ffh,
      0ffh, 0ffh,0,0
   db 0f0h,0f0h,0f0h,0f0h,0f0h,0f0h,0f0h,0f0h,0f0h,0f0h,0f0h,0f0h,
      0f0h,0f0h,0,0
   db 0h,0h,0h,0h,3ch,66h,66h,66h,3ch,18h,7eh,18h,18h,0h,0,0
   db 0h,0h,0h,0h,3fh,33h,3fh,30h,30h,30h,70h,0f0h,0e0h,0h,0,0
   db 0h,0h,0h,0h,7fh,63h,7fh,63h,63h,63h,67h,0e7h,0e6h,0c0h,0,0
   db 18h,18h,18h,18h,18h,18h,18h,0f8h,18h,18h,18h,18h,18h,18h,0,0
```

```
db 0h,0h,0h,0h,80h,0c0h,0e0h,0f8h,0feh,0f8h,0e0h,0c0h,80h,0h,0,0
db 0h,0h,0h,0h,2h,6h,0eh,3eh,0feh,3eh,0eh,6h,2h,0h,0,0
db 0h,0h,0h,0h,0h,0h,0h,1fh,18h,18h,18h,18h,18h,18h,0,0
db 18h,18h,18h,18h,18h,18h,18h,1fh,0h,0h,0h,0h,0h,0h,0,0
db 18h,18h,18h,18h,18h,18h,18h,0f8h,0h,0h,0h,0h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,0h,0f8h,18h,18h,18h,18h,18h,18h,0,0
db 0h,0h,0h,0h,0h,0h,0h,0ffh,0h,0h,0h,0h,0h,0h,0,0
db 18h,18h,18h,18h,18h,18h,18h,18h,18h,18h,18h,18h,18h,18h,0,0
db 0h,0h,18h,3ch,7eh,18h,18h,18h,18h,18h,18h,0h,0h,0h,0,0
db 0h,0h,18h,18h,18h,18h,18h,18h,7eh,3ch,18h,0h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,4h,6h,0ffh,6h,4h,0h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,20h,60h,0ffh,60h,20h,0h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,0h,0ffh,18h,18h,18h,18h,18h,18h,0,0
db 18h,18h,18h,18h,18h,18h,18h,0ffh,0h,0h,0h,0h,0h,0h,0,0
db 0h,0h,0h,0h,0h,10h,38h,38h,7ch,7ch,0feh,0feh,0h,0h,0,0
db 0h,0h,0h,0h,0h,0feh,0feh,7ch,7ch,38h,38h,10h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0,0
db 0h,0h,0h,0h,18h,3ch,3ch,18h,18h,18h,0h,18h,0h,0h,0,0
db 0h,0h,0h,36h,36h,36h,0h,0h,0h,0h,0h,0h,0h,0h,0,0
db 0h,0h,0h,0h,0h,36h,36h,7fh,36h,7fh,36h,36h,0h,0h,0,0
db 0h,0h,0h,18h,18h,3eh,60h,60h,3ch,6h,6h,7ch,18h,18h,0,0
db 0h,0h,0h,0h,0h,63h,66h,0ch,18h,30h,63h,43h,0h,0h,0,0
db 0h,0h,0h,0h,0h,1ch,36h,1ch,3bh,6eh,66h,3bh,0h,0h,0,0
db 0h,0h,0h,30h,30h,60h,0h,0h,0h,0h,0h,0h,0h,0h,0,0
db 0h,0h,0h,0h,0ch,18h,30h,30h,30h,30h,18h,0ch,0h,0h,0,0
db 0h,0h,0h,0h,30h,18h,0ch,0ch,0ch,0ch,18h,30h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,66h,3ch,0ffh,3ch,66h,0h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,18h,18h,7eh,18h,18h,0h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,18h,18h,30h,0h,0,0
db 0h,0h,0h,0h,0h,0h,0h,0h,7eh,0h,0h,0h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,18h,18h,0h,0h,0,0
db 0h,0h,0h,0h,0h,3h,6h,0ch,18h,30h,60h,40h,0h,0h,0,0
db 0h,0h,0h,0h,3eh,63h,67h,6fh,7bh,73h,63h,3eh,0h,0h,0,0
```

```
db 0h,0h,0h,0h,18h,38h,78h,18h,18h,18h,18h,7eh,0h,0h,0,0
db 0h,0h,0h,0h,3ch,66h,6h,0ch,18h,30h,66h,7eh,0h,0h,0,0
db 0h,0h,0h,0h,3ch,66h,6h,1ch,6h,6h,66h,3ch,0h,0h,0,0
db 0h,0h,0h,0h,0eh,1eh,36h,66h,7fh,6h,6h,0fh,0h,0h,0,0
db 0h,0h,0h,0h,7eh,60h,60h,7ch,6h,6h,66h,3ch,0h,0h,0,0
db 0h,0h,0h,0h,1ch,30h,60h,7ch,66h,66h,66h,3ch,0h,0h,0,0
db 0h,0h,0h,0h,7eh,46h,0ch,0ch,18h,18h,18h,3ch,0h,0h,0,0
db 0h,0h,0h,0h,3ch,66h,66h,3ch,66h,66h,66h,3ch,0h,0h,0,0
db 0h,0h,0h,0h,3ch,66h,66h,3eh,6h,6h,0ch,38h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,18h,18h,0h,0h,18h,18h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,18h,18h,0h,0h,18h,18h,30h,0h,0,0
db 0h,0h,0h,0h,0h,0ch,18h,30h,60h,30h,18h,0ch,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,0h,7eh,0h,0h,7eh,0h,0h,0h,0,0
db 0h,0h,0h,0h,0h,30h,18h,0ch,6h,0ch,18h,30h,0h,0h,0,0
db 0h,0h,0h,0h,3ch,66h,46h,0ch,18h,18h,0h,18h,0h,0h,0,0
db 0h,0h,0h,0h,66h,66h,66h,3ch,18h,18h,18h,3ch,0h,18h,0,0
db 0h,0h,0h,0h,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 0h,0h,0h,0h,7eh,33h,33h,3eh,33h,33h,33h,7eh,0h,0h,0,0
db 0h,0h,0h,0h,1eh,33h,60h,60h,60h,60h,33h,1eh,0h,0h,0,0
db 0h,0h,0h,0h,7ch,36h,33h,33h,33h,33h,36h,7ch,0h,0h,0,0
db 0h,0h,0h,0h,7fh,31h,34h,3ch,34h,30h,31h,7fh,0h,0h,0,0
db 0h,0h,0h,0h,7fh,31h,34h,3ch,34h,30h,30h,78h,0h,0h,0,0
db 0h,0h,0h,0h,1eh,33h,63h,60h,67h,63h,33h,1eh,0h,0h,0,0
db 0h,0h,0h,0h,66h,66h,66h,7eh,66h,66h,66h,66h,0h,0h,0,0
db 0h,0h,0h,0h,3ch,18h,18h,18h,18h,18h,18h,3ch,0h,0h,0,0
db 0h,0h,0h,0h,0fh,6h,6h,6h,66h,66h,66h,3ch,0h,0h,0,0
db 0h,0h,0h,0h,73h,33h,36h,3ch,36h,33h,33h,73h,0h,0h,0,0
db 0h,0h,0h,0h,78h,30h,30h,30h,30h,31h,33h,7fh,0h,0h,0,0
db 0h,0h,0h,0h,63h,77h,7fh,7fh,6bh,63h,63h,63h,0h,0h,0,0
db 0h,0h,0h,0h,63h,73h,7bh,6fh,67h,63h,63h,63h,0h,0h,0,0
db 0h,0h,0h,0h,1ch,36h,63h,63h,63h,63h,36h,1ch,0h,0h,0,0
db 0h,0h,0h,0h,7eh,33h,33h,33h,3eh,30h,30h,78h,0h,0h,0,0
db 0h,0h,0h,0h,3ch,66h,66h,66h,66h,6eh,3ch,0eh,0h,0h,0,0
```

```
db 0h,0h,0h,0h,7eh,33h,33h,3eh,36h,33h,33h,73h,0h,0h,0,0
db 0h,0h,0h,0h,3ch,66h,70h,38h,1ch,0eh,66h,3ch,0h,0h,0,0
db 0h,0h,0h,0h,7eh,5ah,18h,18h,18h,18h,18h,3ch,0h,0h,0,0
db 0h,0h,0h,0h,66h,66h,66h,66h,66h,66h,66h,3ch,0h,0h,0,0
db 0h,0h,0h,0h,66h,66h,66h,66h,66h,66h,3ch,18h,0h,0h,0,0
db 0h,0h,0h,0h,63h,63h,63h,6bh,7fh,77h,63h,63h,0h,0h,0,0
db 0h,0h,0h,0h,63h,63h,36h,1ch,1ch,36h,63h,63h,0h,0h,0,0
db 0h,0h,0h,0h,66h,66h,66h,3ch,18h,18h,18h,3ch,0h,0h,0,0
db 0h,0h,0h,0h,7fh,63h,46h,0ch,18h,31h,63h,7fh,0h,0h,0,0
db 0h,0h,0h,0h,1eh,18h,18h,18h,18h,18h,18h,1eh,0h,0h,0,0
db 0h,0h,0h,0h,0h,0c0h,60h,30h,18h,0ch,6h,3h,0h,0h,0,0
db 0h,0h,0h,0h,3ch,0ch,0ch,0ch,0ch,0ch,0ch,3ch,0h,0h,0,0
db 0h,18h,30h,60h,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0ffh,0,0
db 0h,18h,0ch,6h,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 0h,0h,0h,0h,70h,30h,3ch,36h,33h,33h,33h,3eh,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,3eh,63h,60h,60h,63h,3eh,0h,0h,0,0
db 0h,0h,0h,0h,0eh,6h,1eh,36h,66h,66h,66h,3bh,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,3eh,63h,7fh,60h,63h,3eh,0h,0h,0,0
db 0h,0h,0h,0h,1ch,32h,30h,78h,30h,30h,30h,78h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,3bh,66h,66h,66h,3eh,6h,66h,3ch,0,0
db 0h,0h,0h,0h,70h,30h,36h,3bh,33h,33h,33h,73h,0h,0h,0,0
db 0h,0h,0h,0h,0ch,0h,1ch,0ch,0ch,0ch,0ch,1eh,0h,0h,0,0
db 0h,0h,0h,0h,6h,0h,0eh,6h,6h,6h,6h,66h,66h,3ch,0,0
db 0h,0h,0h,0h,70h,30h,33h,36h,3ch,36h,33h,73h,0h,0h,0,0
db 0h,0h,0h,0h,1ch,0ch,0ch,0ch,0ch,0ch,0ch,1eh,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,76h,7fh,6bh,6bh,6bh,63h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,6eh,33h,33h,33h,33h,33h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,6eh,33h,33h,33h,3eh,30h,30h,78h,0,0
db 0h,0h,0h,0h,0h,0h,3bh,66h,66h,66h,3eh,6h,6h, 0fh,0,0
db 0h,0h,0h,0h,0h,0h,6eh,3bh,33h,30h,30h,78h,0h,0h,0,0
```

```
db 0h,0h,0h,0h,0h,0h,3eh,63h,38h,0eh,63h,3eh,0h,0h,0,0
db 0h,0h,0h,0h,18h,18h,7eh,18h,18h,18h,1bh,0eh,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,66h,66h,66h,66h,66h,3bh,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,66h,66h,66h,66h,3ch,18h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,63h,63h,6bh,6bh,7fh,36h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,63h,36h,1ch,1ch,36h,63h,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,63h,63h,63h,63h,3fh,3h,6h,3ch,0,0
db 0h,0h,0h,0h,0h,0h,7fh,66h,0ch,18h,33h,7fh,0h,0h,0,0
db 0h,0h,42h,3ch,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,18h,0,0
db 0eh,2h,4h,0h,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 0h,32h,4ch,0h,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 0h,0h,0h,0h,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,18h,0,0
db 0h,0h,0h,0h,0h,0h,0h,8h,14h,22h,22h,3eh,0h,0h,0,0
db 30h,60h,0d8h,24h,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 0h,0h,1ch,4h,8h,0h,3eh,63h,7fh,60h,63h,3eh,0h,0h,0,0
db 0h,0h,6h,0ch,18h,0h,3eh,63h,7fh,60h,63h,3eh,0h,0h,0,0
db 0h,0h,8h,1ch,36h,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 0h,0h,0h,19h,26h,0h,3eh,63h,7fh,60h,63h,3eh,0h,0h,0,0
db 0h,0h,30h,18h,0ch,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,3eh,63h,7fh,60h,63h,3eh,0h,18h,0,0
db 0ch,6h,1bh,24h,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 0h,0h,8h,1ch,36h,0h,3eh,63h,7fh,60h,63h,3eh,0h,0h,0,0
db 30h,60h,0c8h,1ch,36h,0h,3eh,63h,7fh,60h,63h,3eh,0h,0h,0,0
db 0h,0h,30h,18h,0ch,0h,3eh,63h,7fh,60h,63h,3eh,0h,0h,0,0
db 0ch,6h,0bh,1ch,36h,0h,3eh,63h,7fh,60h,63h,3eh,0h,0h,0,0
db 7h,1h,1ah,24h,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 0h,0h,30h,18h,0ch,0h,1ch,0ch,0ch,0ch,0ch,1eh,0h,0h,0,0
db 7h,1h,0ah,1ch,36h,0h,3eh,63h,7fh,60h,63h,3eh,0h,0h,0,0
db 19h,26h,8h,1ch,36h,0h,3eh,63h,7fh,60h,63h,3eh,0h,0h,0,0
db 19h,26h,18h,24h,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 0h,0h,1ch,4h,8h,0h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 0h,0h,0h,19h,26h,0h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 0h,0h,8h,1ch,36h,0h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
```

```
db 0h,0h,0h,0h,0h,0h,3eh,63h,63h,63h,63h,3eh,0h,0ch,0,0
db 0h,0h,18h,0ch,6h,0h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 18h,30h,68h,1ch,36h,0h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 0h,0h,30h,18h,0ch,0h,66h,66h,66h,66h,66h,3bh,0h,0h,0,0
db 0ch,6h,0bh,1ch,36h,0h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 7h,1h,0ah,1ch,36h,0h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 19h,26h,8h,1ch,36h,0h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 0h,0h,8h,1ch,36h,0h,3eh,63h,63h,63h,63h,3eh,0h,0ch,0,0
db 0h,0h,1ch,4h,8h,0h,66h,66h,66h,66h,66h,3bh,0h,0h,0,0
db 0h,0h,0h,19h,26h,0h,66h,66h,66h,66h,66h,3bh,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,66h,66h,66h,66h,66h,3bh,0h,18h,0,0
db 0h,0h,0h,0h,7h,1h,66h,66h,66h,66h,66h,3bh,0h,0h,0,0
db 0h,0h,0ch,18h,30h,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 0h,0h,6h,0ch,18h,0h,1ch,0ch,0ch,0ch,0ch,1eh,0h,0h,0,0
db 0h,0h,0ch,18h,30h,0h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 0h,0h,0ch,18h,30h,0h,66h,66h,66h,66h,66h,3bh,0h,0h,0,0
db 0h,0ch,18h,30h,7h,1h,66h,66h,66h,66h,66h,3bh,0h,0h,0,0
db 0h,0c0h,60h,30h,7h,1h,66h,66h,66h,66h,66h,3bh,0h,0h,0,0
db 0h,38h,8h,10h,7h,1h,66h,66h,66h,66h,66h,3bh,0h,0h,0,0
db 0h,19h,26h,0h,7h,1h,66h,66h,66h,66h,66h,3bh,0h,0h,0,0
db 0h,0h,0h,0h,7h,1h,66h,66h,66h,66h,66h,3bh,0h,18h,0,0
db 0h,0h,0eh,2h,4h,0h,1ch,0ch,0ch,0ch,0ch,1eh,0h,0h,0,0
db 0h,0h,0h,19h,26h,0h,1ch,0ch,0ch,0ch,0ch,1eh,0h,0h,0,0
db 0h,0h,0h,0ch,0ch,0h,1ch,0ch,0ch,0ch,0ch,1eh,0h,0ch,0,0
db 0h,0h,8h,1ch,36h,0h,3eh,63h,7fh,60h,63h,3eh,0h,18h,0,0
db 0h,0h,6h,6h,1fh,6h,3eh,66h,66h,66h,66h,3fh,0h,0h,0,0
db 0h,0h,0h,7ch,36h,33h,33h,7bh,33h,33h,36h,7ch,0h,0h,0,0
db 0h,0h,18h,24h,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,18h,0,0
db 6h,0ch,5ah,3ch,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 60h,30h,5ah,3ch,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 1ch,4h,4ah,3ch,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 32h,4ch,42h,3ch,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 6h,0ch,18h,0h,7fh,31h,34h,3ch,34h,30h,31h,7fh,0h,0h,0,0
db 30h,18h,0ch,0h,7fh,31h,34h,3ch,34h,30h,31h,7fh,0h,0h,0,0
```

```
db 0eh,2h,4h,0h,7fh,31h,34h,3ch,34h,30h,31h,7fh,0h,0h,0,0
db 0h,19h,26h,0h,7fh,31h,34h,3ch,34h,30h,31h,7fh,0h,0h,0,0
db 0h,0h,0h,0h,7fh,31h,34h,3ch,34h,30h,31h,7fh,0h,18h,0,0
db 30h,60h,0d8h,24h,7fh,31h,34h,3ch,34h,30h,31h,7fh,0h,0h,0,0
db 0ch,6h,1bh,24h,7fh,31h,34h,3ch,34h,30h,31h,7fh,0h,0h,0,0
db 7h,1h,1ah,24h,7fh,31h,34h,3ch,34h,30h,31h,7fh,0h,0h,0,0
db 19h,26h,18h,24h,7fh,31h,34h,3ch,34h,30h,31h,7fh,0h,0h,0,0
db 0h,0h,18h,24h,7fh,31h,34h,3ch,34h,30h,31h,7fh,0h,18h,0,0
db 6h,0ch,18h,0h,3ch,18h,18h,18h,18h,18h,18h,3ch,0h,0h,0,0
db 18h, 0ch,6h,0h,3ch,18h,18h,18h,18h,18h,18h,3ch,0h,0h,0,0
db 1ch,4h,8h,0h,3ch,18h,18h,18h,18h,18h,18h,3ch,0h,0h,0,0
db 0h,19h,26h,0h,3ch,18h,18h,18h,18h,18h,18h,3ch,0h,0h,0,0
db 0h,0h,0h,0h,3ch,18h,18h,18h,18h,18h,18h,3ch,0h,18h,0,0
db 18h,30h,60h,0h,38h,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,0h,0,0
db 30h,18h,0ch,0h,38h,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,0h,0,0
db 1ch,4h,8h,0h,38h,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,0h,0,0
db 0h,32h,4ch,0h,38h,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,0h,0,0
db 0h,0h,0h,0h,38h,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,18h,0,0
db 60h,0c0h,38h,44h,38h,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,0h,0,0
db 0ch,6h,38h,44h,38h,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,0h,0,0
db 0eh,2h,3ah,44h,38h,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,0h,0,0
db 32h,4ch,38h,44h,38h,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,0h,0,0
db 0h,0h,38h,44h,38h,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,18h,0,0
db 30h,60h,0cch,2h,3ah,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,0h,0,0
db 0c0h,60h,0ch,2h,3ah,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,0h,0,0
db 0e0h,20h,4ch,2h,3ah,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,0h,0,0
db 32h,4ch,0eh,2h,3ah,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,0h,0,0
db 0h,0h,0ch,2h,3ah,6ch,0c6h,0c6h,0c6h,0c6h,6ch,38h,0h,18h,0,0
db 0ch,18h,30h,0h,66h,66h,66h,66h,66h,66h,66h,3ch,0h,0h,0,0
db 60h,30h,18h,0h,66h,66h,66h,66h,66h,66h,66h,3ch,0h,0h,0,0
db 1ch,4h,8h,0h,66h,66h,66h,66h,66h,66h,66h,3ch,0h,0h,0,0
db 0h,32h,4ch,0h,66h,66h,66h,66h,66h,66h,66h,3ch,0h,0h,0,0
db 0h,0h,0h,0h,66h,66h,66h,66h,66h,66h,66h,3ch,0h,18h,0,0
db 18h,30h,66h,2h,66h,66h,66h,66h,66h,66h,66h,3ch,0h,0h,0,0
db 0c0h,60h,36h,2h,66h,66h,66h,66h,66h,66h,66h,3ch,0h,0h,0,0
```

```
db 38h,8h,16h,2h,66h,66h,66h,66h,66h,66h,66h,3ch,0h,0h,0,0
db 64h,98h,6h,2h,66h,66h,66h,66h,66h,66h,66h,3ch,0h,0h,0,0
db 0h,0h,6h,2h,66h,66h,66h,66h,66h,66h,66h,3ch,0h,18h,0,0
db 0ch,18h,30h,0h,66h,66h,66h,3ch,18h,18h,18h,3ch,0h,0h,0,0
db 30h,18h,0ch,0h,66h,66h,66h,3ch,18h,18h,18h,3ch,0h,0h,0,0
db 38h,8h,10h,0h,66h,66h,66h,3ch,18h,18h,18h,3ch,0h,0h,0,0
db 0h,32h,4ch,0h,66h,66h,66h,3ch,18h,18h,18h,3ch,0h,0h,0,0
db 0h,0h,0eh,2h,4h,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 0h,0h,0h,19h,26h,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,3ch,6h,3eh,66h,66h,3bh,0h,18h,0,0
db 18h,30h,68h,1ch,36h,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 0ch,6h,0bh,1ch,36h,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 7h,1h,0ah,1ch,36h,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 19h,26h,8h,1ch,36h,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 0h,0h,8h,1ch,36h,0h,3ch,6h,3eh,66h,66h,3bh,0h,18h,0,0
db 0h,0h,0h,22h,1ch,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 3h,6h,0ch,22h,1ch,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 30h,18h,0ch,22h,1ch,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 0eh,2h,4h,22h,1ch,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 19h,26h,0h,22h,1ch,0h,3ch,6h,3eh,66h,66h,3bh,0h,0h,0,0
db 0h,0h,0h,22h,1ch,0h,3ch,6h,3eh,66h,66h,3bh,0h,18h,0,0
db 0h,0h,6h,0ch,18h,0h,63h,63h,63h,63h,3fh,3h,6h,3ch,0,0
db 0h,0h,18h,0ch,6h,0h,63h,63h,63h,63h,3fh,3h,6h,3ch,0,0
db 0h,0h,1ch,4h,8h,0h,63h,63h,63h,63h,3fh,3h,6h,3ch,0,0
db 0h,0h,0h,19h,26h,0h,63h,63h,63h,63h,3fh,3h,6h,3ch,0,0
db 0h,0h,0h,0h,0h,0h,63h,63h,63h,63h,3fh,3h,16h,3ch,0,0
db 0h,0h,0h,0h,7h,1h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 0h,0ch,18h,30h,7h,1h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 0h,0c0h,60h,30h,7h,1h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 0h,38h,8h,10h,7h,1h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 0h,19h,26h,0h,7h,1h,3eh,63h,63h,63h,63h,3eh,0h,0h,0,0
db 0h,0h,0h,0h,7h,1h,3eh,63h,63h,63h,63h,3eh,0h,18h,0,0
db 0h,0h,1ch,22h,1ch,36h,63h,63h,63h,63h,36h,1ch,0h,0h,0,0
db 0h,0h,6h,1h,1dh,36h,63h,63h,63h,63h,36h,1ch,0h,0h,0,0
db 0h,0h,7h,1h,66h,66h,66h,66h,66h,66h,66h,3ch,0h,0h,0,0
```

```
db 0h,0h,42h,3ch,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 0h,0h,18h,24h,18h,3ch,66h,66h,7eh,66h,66h,66h,0h,0h,0,0
db 0h,0h,1ch,22h,7fh,31h,34h,3ch,34h,30h,31h,7fh,0h,0h,0,0
db 0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0h,0,0

;########################################
DATBUFF LABEL BYTE;vùng đệm nạp file
;########################################
code ENDS
END START
*******************************
```

Bảng ký tự chữ hoa và chữ thường của font Việt nam VNFONT dùng trong chương trình được thể hiện trong bảng 3.1 và bảng 3.2.

Bảng 3.1

	À 96	Á 94	Ả 124	Ã 125	Ạ 126
Ă252	Ằ177	Ắ176	Ẳ178	Ẵ179	Ặ123
Â253	Ầ135	Ấ128	Ẩ140	Ẫ144	Ậ175
	È181	É180	Ẻ182	Ẽ183	Ẹ184
Ê254	(Ề186)	(Ế185)	(Ể187)	(Ễ188)	Ệ189
	Ì191	Í190	Ỉ192	Ĩ193	Ị194
	Ò196	Ó195	Ỏ197	Õ198	Ọ199
Ô249	(Ồ201)	(Ố200)	(Ổ202)	(Ỗ203)	(Ộ204)
Ơ250	Ờ206	(Ớ205)	Ở207	Ỡ208	Ợ209
	Ù211	Ú210	Ủ212	Ũ213	Ụ214
Ư251	Ừ216	Ứ215	Ử217	Ữ218	Ự219
	Ỳ221	Ý220	Ỷ222	Ỹ223	Ỵ64

Bảng 3.2

	à 133	á 160	ả 224	ã 225	ạ 226
ă232	ằ234	ắ233	ẳ235	ẵ236	ặ237
â131	ầ228	ấ227	ẩ229	ẫ230	ậ231
	è138	é130	ẻ129	ẽ132	ẹ134
ê136	ề139	ế137	ể142	ễ143	ệ172
	ì141	í161	ỉ169	ĩ170	ị171
	ò149	ó162	ỏ145	õ146	ọ148
ô147	ồ152	ố150	ổ153	ỗ154	ộ155
ơ243	ờ245	ớ244	ở246	ỡ247	ợ248
	ù151	ú163	ủ156	ũ157	ụ158
ư159	ừ165	ứ164	ử166	ữ167	ự168
	ỳ239	ý238	ỷ240	ỹ241	ỵ242

CHƯƠNG 4

LẬP TRÌNH HỆ THỐNG TRÊN WINDOWS

Khi máy tính PC được chế tạo với các tính năng như tốc độ ngày càng cao, độ rộng của kênh dữ liệu ngày càng lớn cùng các các tính năng khác ngày càng hoàn thiện thì hệ điều hành để điều khiển hoạt động của nó phải có khả năng đa nhiệm nhằm khai thác triệt để sức mạnh mới của máy tính. Các hãng phần mềm đã đưa ra nhiều hệ điều hành đa nhiệm khác nhau, trong đó nổi bật nhất là hệ điều hành của hãng MICROSOFT - hệ điều hành đa nhiệm dòng WINDOWS. Trong kỹ thuật lập trình trên WINDOWS cần phải chuyển tư duy sang một hướng khác - lập trình hướng đối tượng, cho phép tiếp cận và khai thác tài nguyên vốn vô cùng phong phú của MICROSOFT WINDOWS.

4.1. HỆ ĐIỀU HÀNH WINDOWS

Để hỗ trợ tốt hơn cho người sử dụng trên máy tính PC, hãng Microsoft liên tục phát triển các hệ điều hành mới cho máy tính PC và hiện nay hệ điều hành WINDOWS đang được dùng phổ biến với nhiều tính năng mạnh ưu việt hơn hẳn hệ điều hành DOS và hệ điều hành WINDOWS 3.x.

Hệ điều hành của Windows bao gồm các thành phần sau đây :

- ***Bộ quản lý máy ảo VMM***

Bộ quản lý máy ảo VMM (Virtual Machine Manager) là thành phần hạt nhân của hệ điều hành Windows. Nhiệm vụ chính của VMM là tạo, thực hiện, giám sát và kết thúc hoạt động một cách hợp thức của các

máy ảo. VMM cung cấp các dịch vụ như quản lý bộ nhớ, xử lý ngắt và ngăn chặn các lỗi hệ thống. VMM làm việc với các thiết bị ảo (virtual devices), các module 32-bit ở chế độ bảo vệ (protected-mode), cho phép các thiết bị ảo ngăn chặn các ngắt và các lỗi để điều khiển truy nhập của một ứng dụng tới các thiết bị phần cứng và cài đặt phần mềm cho các thiết bị phần cứng đó.

VMM cung cấp khả năng xử lý đa nhiệm. Nó thực hiện đồng thời nhiều ứng dụng cùng lúc bằng cách chia sẻ thời gian hoạt động của CPU.

- ***Các thiết bị ảo VxD***

Các thiết bị ảo *(Virtual devices - VxD)* là các chương trình 32 bit hỗ trợ các thiết bị độc lập với VMM bằng cách quản lý các thiết bị phần cứng (hardware devices) của máy tính và hỗ trợ phần mềm xử lý. VxD hỗ trợ tất cả các thiết bị phần cứng cho các máy tính PC bao gồm các bộ điều khiển ngắt lập trình được (Programmable Interrupt Controller - PIC), điều khiển thời gian (Timer), truy nhập trực tiếp bộ nhớ (Direct Memory Access - DMA), điều khiển các thiết bị (device), điều khiển đĩa (disk controller), điều khiển cổng nối tiếp (serial ports), điều khiển cổng song song (parallel ports), điều khiển bàn phím (keyboard), và điều khiển màn hình hiển thị (display adapter). Một VxD được yêu cầu cho bất cứ thiết bị phần cứng nào nếu thiết bị đó được thiết lập ở chế độ làm việc. Nói cách khác, nếu trạng thái của thiết bị phần cứng có thể bị phá vỡ hoặc thay đổi bởi việc chuyển đổi giữa nhiều máy ảo (virtual machines) hoặc giữa nhiều ứng dụng, thì thiết bị đó bắt buộc phải có một VxD phù hợp.

Nói chung, một VxD có thể cung cấp nhiều loại dịch vụ cho VMM và các thiết bị ảo khác. Windows cho phép người sử dụng cài đặt các chương trình điều khiển thiết bị ảo mới để hỗ trợ cho một thiết bị phần cứng mới được thêm vào máy tính hoặc cung cấp một vài dịch vụ mới cho hệ thống.

Một VxD cũng có thể cung cấp các hàm giao diện API (Application Programming Interface) cho các ứng dụng chạy trong chế độ mô phỏng

80X86 hoặc chế độ bảo vệ. Các hàm này có thể cho phép các ứng dụng truy nhập trực tiếp tới các tính năng của VxD.

Windows có một giao diện điều khiển các thiết bị vào/ra (Input and Output Controll - IOCTL) cho phép các ứng dụng dựa trên môi trường Microsoft Win32 có thể truyền thông trực tiếp với các VxD. Các ứng dụng sử dụng giao diện này thực hiện các hàm hệ thống của MS-DOS để thu lượm các thông tin về một thiết bị, hoặc thực hiện các thao tác vào/ra không có sẵn trong các hàm chuẩn của Wind32.

- *Các chương trình điều khiển thiết bị (Device drivers)*

Một chương trình điều khiển thiết bị trong Windows được đặt trong thư viện liên kết động được Windows sử dụng để tương tác với thiết bị phần cứng, chẳng hạn như bàn phím hoặc màn hình... Thay vì truy nhập trực tiếp tới một thiết bị phần cứng, Windows nạp trình điều khiển cho thiết bị đó và gọi các hàm trong trình điều khiển thiết bị để thực hiện các tác động trên thiết bị đó. Mỗi trình diều khiển thiết bị cung cấp một tập hợp các hàm chức năng. Windows gọi các hàm chức năng này để hoàn thành một thao tác đối với thiết bị bị điều khiển, chẳng hạn như truyền 1 byte ra cổng COM hoặc dịch mã quét của một phím được ấn. Các hàm chức năng của một trình điều khiển thiết bị cũng bao gồm các mã lệnh đặc trưng cần thiết của thiết bị để thực hiện các hành động trên thiết bị đó.

Windows đòi hỏi phải có các trình điều khiển thiết bị tương ứng cho màn hình hiển thị, bàn phím và các cổng truyền thông thông thường (như cổng nối tiếp và cổng song song). Các trình điều khiển thiết bị khác cũng có thể được yêu cầu nếu người dùng bổ sung thêm các thiết bị mới vào hệ thống.

- *Các thư viện liên kết động DLL*

Trong Microsoft Windows, thư viện liên kết động DLL (Windows dynamic-link libraries) là các module chứa các hàm và dữ liệu. Một thư viện liên kết động được nạp tại thời điểm thực hiện (runtime) bằng cách gọi các module của nó (.EXE or.DLL). Khi một thư viện liên kết động đã

được nạp, nó được sắp xếp vào vùng địa chỉ trống của tiến trình gọi thực hiện các hàm của thư viện liên kết động đó (Tiến trình cha).

Các thư viện liên kết động có thể định nghĩa hai kiểu hàm: Các hàm tổng thể và các hàm cục bộ. Các hàm tổng thể có thể được gọi bởi các module khác nhau. Các hàm cục bộ chỉ có thể bị gọi bên trong thư viện liên kết động nơi mà chúng được định nghĩa. Mặc dù một thư viện liên kết động có thể kết xuất dữ liệu, nhưng các dữ liệu của nó thường chỉ được sử dụng bởi chính các hàm của các thư viện động đó. Các DLL giúp giảm bộ nhớ khi nhiều ứng dụng sử dụng cùng một hàm chức năng ở cùng một thời điểm.

Liên kết động cung cấp một kỹ thuật để liên kết các ứng dụng với các thư viện của các hàm ở thời điểm hoạt động (Run Time) của các ứng dụng đó. Các thư viện được lưu trữ trong các file thực hiện được và không được sao vào file của ứng dụng như một liên kết tĩnh. Các thư viện này là liên kết động bởi vì chúng được liên kết với một ứng dụng khi ứng dụng đó được nạp và thực hiện. Khi một ứng dụng sử dụng một DLL, hệ điều hành nạp DLL vào bộ nhớ, thiết lập tham chiếu tới các hàm trong DLL sao cho ứng dụng có thể gọi được chúng, và DLL đó được loại bỏ khỏi bộ nhớ khi nó không còn cần thiết cho ứng dụng nữa.

DLL được thiết kế để cung cấp tài nguyên cho các ứng dụng. Nhiều ứng dụng có thể sử dụng mã trong một DLL, nghĩa là chỉ một bản sao của mã lệnh thường trú trong hệ thống. Ngoài ra cũng có thể nâng cấp hoặc sửa đổi DLL mà không cần sửa đổi ứng dụng nếu như các giao diện của các hàm trong DLL không thay đổi.

Những nhà phát triển phần mềm có thể mở rộng môi trường của Windows bằng cách tạo các DLL mới và thêm nó vào hệ thống để hỗ trợ các ứng dụng của Windows.

- ***Các thông báo hệ thống***

Windows quản lý và điều khiển các ứng dụng thông qua một tập hợp các thông báo hệ thống (system messages). Các thông báo hệ thống cung cấp cách thức để báo cho tất cả các ứng dụng cũng như các thành

phần khác của hệ thống những thay đổi có thể ảnh hưởng đến hoạt động và sự truy nhập tới các tài nguyên hệ thống của chúng.

Các ứng dụng và các thành phần khác của hệ thống cũng có thể định nghĩa các thông báo hệ thống của riêng chúng và sử dụng các thông báo đó để phục vụ cho việc thông báo về các kiểu sự kiện khác nhau trong hệ thống.

Mỗi một thông báo hệ thống bao gồm một định danh và hai tham số 32-bit, wParam và lParam. Định danh của thông báo hệ thống là một giá trị phân biệt dùng để chỉ định mục đích của thông báo đó. Các tham số cung cấp thêm các thông tin cần thiết, trong đó tham số wParam thường là một giá trị thông báo.

Một thông báo hệ thống có thể được gửi tới một hoặc nhiều thành phần trong hệ thống. Các thành phần này có thể là các ứng dụng, các trình điều khiển thiết bị, các trình điều khiển mạng của Windows hoặc các trình điều khiển thiết bị ở mức hệ thống.

Hầu hết các ứng dụng không gửi đi các thông báo hệ thống, mà chúng nhận và xử lý các thông báo hệ thống được gửi tới từ các thành phần khác của hệ thống. Hệ điều hành thường gửi các thông báo hệ thống phản ánh sự thay đổi của các trình điều khiển thiết bị ở mức hệ thống. Trình điều khiển thiết bị hoặc các thành phần liên quan thường sản sinh ra các thông báo hệ thống, sau đó gửi chúng tới các ứng dụng và các thành phần khác để báo cho chúng những thay đổi. Chẳng hạn, hệ thống con chịu trách nhiệm quản lý đĩa sản sinh và gửi các thông báo hệ thống khi trình điều khiển thiết bị cho đĩa mềm nhận biết sự thay đổi khi người dùng đưa đĩa vào ổ đĩa.

Các ứng dụng nhận được thông báo hệ thống thông qua thủ tục cửa sổ của cửa sổ làm việc chính của các ứng dụng đó. Thông báo hệ thống không được gửi tới các cửa sổ con của ứng dụng (đối với các ứng dụng có nhiều cửa sổ). Khi đó tùy thuộc vào nội dung của thông báo, mỗi ứng dụng có thể có những hành động tương ứng. Một vài thông báo hệ thống,

được gọi là các thông báo hàng đợi, đòi hỏi các ứng dụng phải trả lại hoặc giá trị TRUE hoặc BROADCAST_QUERY_DENY để chỉ định liệu hệ thống có nên tiếp tục gửi các thông báo tới những nơi nhận khác không.

Người lập trình có thể tạo các thông báo hệ thống của riêng mình để xác định các hành động giữa các ứng dụng và các thành phần khác của hệ thống. Điều này đặc biệt hữu ích nếu người lập trình phải tạo các trình điều khiển thiết bị hoặc các trình điều khiển thiết bị ở mức hệ thống. Những thống báo hệ thống mới đó có thể gửi hoặc nhận các thống tin giữa trình điều khiển thiết bị và các ứng dụng có sử dụng thiết bị của trình điều khiển đó.

Người lập trình cũng có thể gửi các thông báo do họ tạo ra tới các thành phần trong hệ thống. Các thông báo đó sẽ được gửi tới các nơi nhận theo thứ tự: Các trình điều khiển thiết bị mức hệ thống, các trình điều khiển mạng của Windows, các trình điều khiển thiết bị khác và các ứng dụng. Điều này có nghĩa là các trình điều khiển thiết bị mức hệ thống, nếu được chọn là nơi nhận, luôn luôn nhận được các thông báo đầu tiên. Trong cùng một kiểu những nơi nhận, không có trình điều khiển nào được đảm bảo sẽ nhận được các thông báo trước các trình điều khiển khác. Điều này có nghĩa là một thông báo muốn được gửi cho một trình điều khiển nhất định nào đó bắt buộc phải có một định danh phân biệt tổng thể để sao cho các trình điều khiển thiết bị khác không có ý định xử lý nó.

Windows có một cơ chế hàng đợi các thông báo cho phép chọn lựa những nơi nhận đã được trao quyền nhận thông báo để thực hiện một hành động nào đó. Người lập trình có thể tạo ra hàng đợi thông báo cho riêng mình. Với các hàng đợi này, người lập trình có thể chỉ ra thứ tự nhận các thông báo của các thành phần cần nhận thông báo (các nơi nhận) trong hệ thống. Mỗi nơi nhận thông báo của hàng đợi này phải trả lại giá trị để báo cho hàng đợi gửi thông báo cho nơi nhận tiếp theo hoặc kết thúc việc gửi thông báo. Người lập trình cũng có thể tạo các trình điều khiển để gửi và xử lý các thông báo.

4.2. QUẢN LÝ BỘ NHỚ CỦA HỆ ĐIỀU HÀNH WINDOWS

Hệ điều hành Windows cho phép truy xuất một không gian nhớ dung lượng nhiều GB. Trong chế độ bảo vệ, hệ điều hành điều khiển tác vụ nạp chương trình phải xác lập một bảng mô tả thông tin địa chỉ bộ nhớ. Cách xử lý chế độ bảo vệ tính toán một địa chỉ vật lý ở trên các CPU x86 cũng giống như ở trên CPU cơ sở 286. Bộ xử lý dùng nội dung của một thanh ghi phân đoạn như một chỉ dẫn tới một bảng mô tả, và việc nhập liệu vào bảng mô tả chứa đựng hầu hết những thông tin còn lại. Hầu như x86 cho phép hệ điều hành thi hành một sơ đồ bộ nhớ ảo phân trang hoàn hảo. Khi hệ điều hành có thể phân trang bộ nhớ, thông tin địa chỉ được rút ra từ bảng mô tả phải đi qua một cấp diễn giải cao hơn trước khi được sử dụng như một địa chỉ bộ nhớ thực.

4.2.1. Quản lý bộ nhớ

Quản lý bộ nhớ (memory management) trong Windows được phân thành hai mức độ khác nhau: một mức độ lập trình viên có thể nhìn thấy và một mức độ chỉ hệ điều hành nhìn thấy mà thôi.

Không gian địa chỉ hệ thống System address space (not addressable by application)	4GB
Không gian địa chỉ chia xẻ Shared address space	3GB
Không gian địa chỉ riêng Private address space	2GB

Hình 4.1. **Ánh xạ bộ nhớ ảo dành cho trình ứng dụng**

Hình 4.1 minh họa cách bố trí của bộ nhớ ảo trong trình ứng dụng Win32. Mỗi trình ứng dụng Win32 lại có một ánh xạ bộ nhớ tương ứng và mỗi không gian địa chỉ đều là duy nhất. Tuy nhiên, bộ nhớ vẫn không được bảo vệ trọn vẹn: bộ nhớ riêng được cấp phát cho một trình ứng dụng Win32 có thể bị trình ứng dụng khác gán địa chỉ tranh chấp. Không gian địa chỉ riêng của trình ứng dụng Win32 lại cũng là miền mà

trong đó hệ thống cấp phát bộ nhớ để thỏa mãn nhu cầu đòi hỏi của trình ứng dụng khi đang được thực thi trên máy.

Không gian địa chỉ hệ thống được dùng để ánh xạ những thư viện liên kết động (DLL) trong hệ thống vào địa chỉ không gian của trình ứng dụng. Những lệnh gọi tới những thư viện liên kết động của hệ thống trở thành những lệnh gọi tới vùng này. Những trình ứng dụng cũng có thể yêu cầu cấp phát bộ nhớ động bằng phương tiện của những địa chỉ ảo được ánh xạ vào vùng phân chia. Những địa chỉ ảo được ánh xạ vào không gian địa chỉ phân chia phục vụ nhu cầu kiểm soát sự phân chia bộ nhớ cho những trình ứng dụng khác nữa.

- ***Bộ nhớ ảo***

Bộ nhớ ảo là một phương pháp cho phép nhiều chương trình cùng chạy một lúc có thể dùng chung một bộ nhớ vật lý của một máy tính. Chế độ ảo ở đây muốn nói tới thao tác của bộ vi xử lý x86 ở trong chế độ ảo. Sự quản lý bộ nhớ ảo được hoàn toàn đặt dưới sự kiểm soát của hệ điều hành. Bất cứ một chương trình riêng rẽ nào đều cũng có thể truy xuất được bộ nhớ cần thiết vào bất cứ lúc nào. Thí dụ, chúng ta chạy hệ Widows trên một hệ thống có 4MB bộ nhớ và một đĩa cứng còn trống nhiều, ngay Windows với những thành phần như shell, bộ quản lý in ấn, v.v... đã chiếm một bộ nhớ hơn 1 MB. Còn ở trên đĩa là một chương trình xử lý văn bản mà người dùng định dùng tới. Một khi được nạp vào bộ nhớ, chương trình này phải chiếm 2 MB, và người dùng phải nạp vào bộ nhớ một tài liệu lớn bao gồm nhiều *font* chữ khác nhau. Như thế, tài liệu này chiếm tới 400 KB của MB còn lại trong bộ nhớ. Giờ đây, người dùng định nhập thêm một bảng số vào trong tài liệu đó. Những số liệu này nằm trong phần mềm bảng tính điện tử, và người dùng lại phải chạy một trình ứng dụng bảng tính điện tử để cắt, dán và sao chép vào tài liệu. Như thế Windows bắt buộc phải nạp trình ứng dụng bảng tính điện tử và dữ liệu vào trong bộ nhớ còn lại là 624 KB. Nếu như người dùng lại dùng phần mềm DELPHI thì phần mềm này và dữ liệu không thể chứa nổi vào bộ nhớ còn lại. Cứ theo cách nghĩ, khó lòng có thể chạy nổi chương trình trên theo như những điều đã mô tả. Song, hệ thống và cả

hai trình ứng dụng đó vẫn chạy suôn sẻ như thể bộ nhớ vẫn còn trống rất nhiều. Mọi chuyện xảy ra không phải ở bên trong 4 MB sẵn có của bộ nhớ vật lý, mà là xảy ra ở trong bộ nhớ ảo.

- ***Quản lý bộ nhớ ảo***

Bộ nhớ ảo của hệ thống được tạo lập nhờ RAM ở trong máy tímh và tập tin Windows tráo đổi trên đĩa cứng. Hệ điều hành quản lý tất cả bộ nhớ sẵn có bằng cách tráo đổi những phân đoạn chương trình và dữ liệu từ RAM vào tập tin tráo đổi. Những chỉ thị trong một phân đoạn mã đặc biệt được thi hành thì phân đoạn đó phải được nạp vào RAM, những phân đoạn mã khác vẫn có thể ở trên đĩa trong tập tin tráo đổi nếu không được dùng tới. Một vùng bộ đệm dữ liệu đĩa ở bên trong một phân đoạn dữ liệu phải ở trong RAM nếu chuyển đổi đĩa thành công. Mỗi khi một phân đoạn không nằm trong RAM thì hệ điều hành có thể đánh dấu vắng bóng phân đoạn đó bằng cách xóa bit Present nằm trong bộ mô tả phân đoạn thích hợp. Nếu truy xuất tới phân đoạn đó thì CPU x86 sẽ phát sinh một ngắt khiếm diện (not present interrupt) để báo cho hệ điều hành biết sự kiện này. Hệ thống sẽ dàn xếp để nạp đoạn bị thiếu đó vào một vùng sẵn có ở trong RAM, và rồi khởi động lại chương trình vốn đã gây ra ngắt đó.

Bộ xử lý từ 386 trở lên đã cải thiện mọi tác vụ trên bằng cách cho phép một kế hoạch phân trang bộ nhớ ảo khiến hệ điều hành có thể thi hành tất cả việc cấp phát bộ nhớ, giải phóng bộ nhớ và những tác vụ tráo đổi trong những đơn vị của những trang bộ nhớ. Trong bộ xử lý 386, một trang bộ nhớ là 4 KB, và mỗi phân đoạn bộ nhớ được tạo lập bằng một hay nhiều trang bộ nhớ 4 KB. Trong khi khởi tạo, hệ điều hành thoạt tiên chuyển bộ xử lý về chế độ bảo vệ, rồi làm tác vụ phân trang.. Khi đã phân trang xong, bộ xử lý 386 liền thay đổi các thông dịch địa chỉ 32 bit bằng cách thêm địa chỉ gốc từ bộ mô tả vào (offset) được chương trình phát sinh. Hệ điều hành giữ lấy toàn bộ cấu trúc bằng cách lưu trữ địa chỉ của thư mục bảng phân trang bộ nhớ dành cho chương trình hiện hành trong một thanh ghi đặc biệt tên là CR3. Mỗi lần chuyển đổi tác vụ, hệ điều hành lại nạp lại CR3 để báo cho chương trình mới biết thư

mục phân trang bộ nhớ (page directory). Để hỗ trợ những thao tác bộ nhớ ảo và hệ thống bảo vệ bộ nhớ x86, thì thư mục trang bộ nhớ và những nhập liệu bảng trang bộ nhớ phải chứa luôn cả một số bit cờ trạng thái. Ngoài ra, muốn để tương thích với những chương trình cũ viết bằng mã 16 bit, Windows lại dùng cả kỹ thuật *thunk* để chuyển đổi mã 32 bit thành 16 bit và ngược lại.

4.2.2. Hệ thống bảo vệ

Hệ điều hành Windows đã đề xuất những khả năng bảo vệ: bảo vệ dữ liệu cho người dùng, bảo vệ một chương trình này khỏi bị những chương trình khác lấn chiếm khi đang chạy trong máy, và bảo vệ những thiết bị vật lý không bị truy xuất trái phép.

- ***Bảo vệ bộ nhớ***

Mỗi khi một trình ứng dụng truy xuất một vị trí bộ nhớ không nằm trong ánh xạ bộ nhớ (memory map), bộ xử lý x86 liền phát sinh một ngắt và trao đổi cho hệ điều hành một tập thông tin liên hệ. Đôi khi, hệ điều hành phải dàn xếp để thêm một trang bộ nhớ thích hợp và ánh xạ bộ nhớ của trình ứng dụng đó. Với những phiên bản Windows 3.x, hệ điều hành cấp phát nhiều bộ nhớ cho trình ứng dụng. Nhưng thường nếu chạy nhiều trình ứng dụng tốn nhiều bộ nhớ thì hệ điều hành sẽ hiển thị một hộp thoại báo là bộ nhớ quá ít không thể tiếp tục thi hành, hoặc trình ứng dụng không thể làm được gì nữa. Vào trường hợp này, Windows 9x giải quyết bằng cách mở rộng số lượng những tài nguyên hệ điều hành sẵn có, đặc biệt khi những tài nguyên yêu cầu hệ thống được thỏa mãn do hệ điều hành cấp phát bộ nhớ từ vùng bộ nhớ động (memory pool) của chế độ bảo vệ 32 bit.

- ***Vành bảo vệ trong Windows***

Windows khai thác khả năng bộ xử lý Intel x86 để hỗ trợ những cấp độ ưu tiên phức tạp. Do tác vụ xử lý của những vành bảo vệ có khuynh hướng ảnh hưởng tới nhiều khía cạnh của cách thiết kế hệ thống, Windows xử lý bằng cách dùng những mức độ ưu tiên 0 và 3.

Những phần tử vành 0 là những gì người lập trình thường nghĩ tới như hệ điều hành. Phần mềm vành 0 là phần cơ sở của hệ điều hành và nó có mức ưu tiên cao nhất.

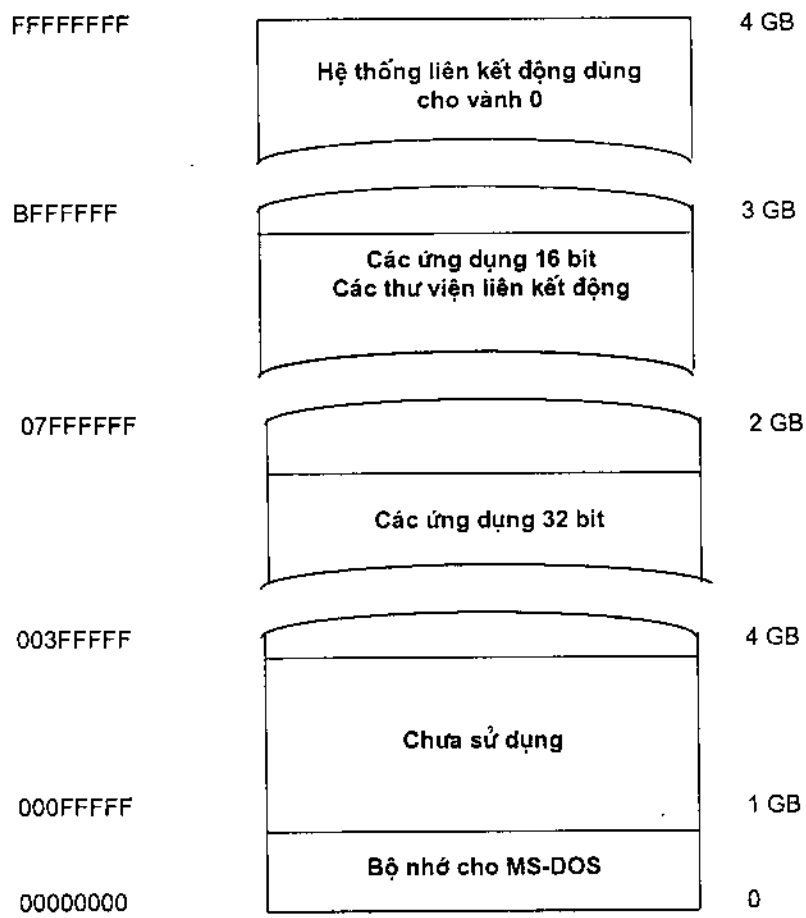
Những trình ứng dụng Windows và những trình ứng dụng hệ điều hành Microsoft luôn luôn chạy ở vành 3, do đó những quyền ưu tiên của nó thường bị hạn chế. Vì vậy, việc thâm nhập vào hệ thống rất khó mà phải có một số kỹ thuật lập trình đặc biệt với ngôn ngữ bậc thấp như Assembly hoặc ngôn ngữ bậc cao như C, Delphi... mới có thể thâm nhập được.

4.2.3. Ánh xạ bộ nhớ trong Windows

Bộ xử lý x86 quản lý được không gian địa chỉ ảo ltừ 1 GB trở lên. Trong không gian địa chỉ ảo này, những phần tử hệ thống khác nhau và những trình ứng dụng đều chiếm giữ những vùng có những giới hạn nhất định. Hình 4.2 cho ta thấy ánh xạ bộ nhớ cơ bản cho hệ thống là ánh xạ không gian địa chỉ ảo 4 GB vào trong bộ nhớ vật lý có sẵn.

Trong ánh xạ bộ nhớ hệ thống, vùng 1 MB thấp nhất của không gian địa chỉ ảo được dùng để thao tác máy ảo của hệ điều hành hiện hành. Mỗi một máy ảo cũng có riêng một ánh xạ bộ nhớ hợp thức trong vùng từ 2 tới 3 GB. Sự ánh xạ này cho phép hệ thống được gán địa chỉ cho một máy ảo bất kể máy đó hoạt động hay không. Nhưng khi một máy ảo của hệ điều hành Microsoft chạy, nó liền được ánh xạ vào đáy của 1 MB.

Trong không gian địa chỉ ảo của trình ứng dụng Windows 32 bit, những bộ công cụ chuẩn đều dùng 4 MB như là dung lượng mặc định và chúng được nạp ngay vào vùng từ 4 MB tới 2 GB. Địa chỉ nạp trình ứng dụng loại 4 MB sẽ làm phù hợp được với địa chỉ Windows NT được dùng để nạp những trình ứng dụng 32 bit ngay trong phiên bản sản phẩm đầu tiên. Vùng nhớ thấp nhất 16 KB của mỗi không gian địa chỉ của trình ứng dụng 32 bit (những địa chỉ ảo từ 0 tới 3FFFh) đều không hợp lệ.



Hình 4.2. **Tạo một máy ảo để tải file chạy vào bộ nhớ**

4.3. QUẢN LÝ FILE CỦA WINDOWS

- ***Vùng tiêu đề của file chạy (Portable Executable file - PE file)***

Điều quan trọng đầu tiên cần phải biết về vùng tiêu đề của file chạy (Portable Executable file - PE file) là một chương trình trên đĩa chính là hình ảnh của một module sau khi đã được Windows tải vào bộ nhớ. Ta sử dụng từ "module" để nói đến phần *code*, *data*, và *resource* của một file chạy hoặc thư viện liên kết động (DLL) được tải vào bộ nhớ. Trình nạp của Windows không cần phải làm gì nhiều để tạo ra một *process* từ file

trên đĩa, nó sử dụng cơ chế ánh xạ file trên bộ nhớ để ánh xạ các phần của file trên đĩa. Trong môi trường Windows 32 bit, phần bộ nhớ được sử dụng bởi một module là liên tục, ta chỉ cần phải xác định một điều là trình nạp sẽ ánh xạ file vào địa chỉ nào trên bộ nhớ.

Một vấn đề nữa cần chú ý là *địa chỉ ảo tương đối* (the Relative Virtual Address - RVA) vì trong nhiều trường file PE phải sử dụng RVA này. Một RVA đơn giản là một địa chỉ OFFSET của các đối tượng, địa chỉ OFFSET này có quan hệ với địa chỉ ánh xạ file trong bộ nhớ. Lấy ví dụ, ta nói trình nạp ánh xạ file trong bộ nhớ bắt đầu tại địa chỉ 0x10000 trong *vùng địa chỉ ảo*, nếu một thành phần nào đó của file bắt đầu tại địa chỉ 0x10464 thì RVA của nó sẽ là 0x464. Công thức tính toán địa chỉ này rất đơn giản:

Địa chỉ ảo (Virtual Address) - Địa chỉ cơ sở (Base Address) = RVA

0x10464 - 0x10000 = 0x00464

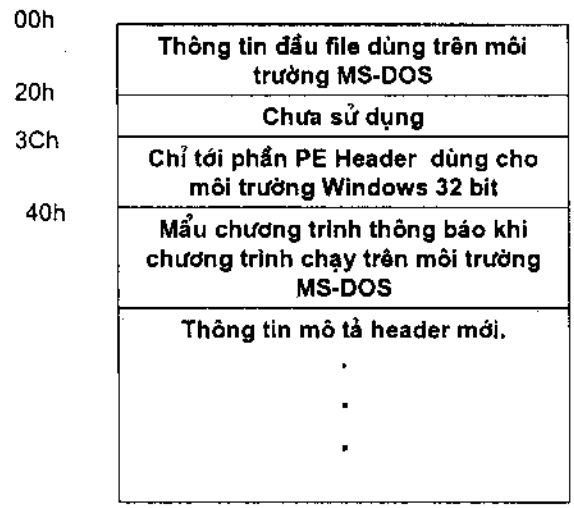
Địa chỉ cơ sở là địa chỉ bắt đầu của file EXE (hoặc của thư viện DLL) ánh xạ trong bộ nhớ. Trong Win32, gọi địa chỉ cơ sở của một module là HINSTANCE (instance handle) ở một mức nào đó có vẻ như không hợp lý bởi nó có nguồn gốc từ Win16. Mỗi một *instance* của ứng dụng trong Windows 16 bit có một mảng dữ liệu riêng biệt lập (gắn với nó là một handle toàn cục) để phân biệt với những *instance* khác của ứng dụng đó, do vậy xuất hiện cụm từ "instance handle". Các ứng dụng trong Win32 không cần phải được phân biệt giữa các instance bởi vì chúng không dùng chung vùng nhớ nào. Tuy nhiên, HINSTANCE được sử dụng để chỉ ra tính kế thừa từ Win16 của Win32. Đối với Win32, ta có thể dùng hàm *GetModuleHandle* cho bất cứ thư viện DLL nào mà chương trình sử dụng, do vậy có thể có được một con trỏ cho phép truy nhập đến các thành phần của module đó.

Trong file PE, một khái niệm cũng rất quan trọng là *Section*. Section có thể chứa hoặc là mã chương trình, hoặc là dữ liệu của chương trình. Không giống như các segment, section là một khối bộ nhớ liên tục không giới hạn về độ lớn. Một số section chứa mã hoặc dữ liệu mà chương trình trực tiếp sử dụng, một số khác được tạo ra do trình liên kết

(linker) chứa các thông tin cần thiết cho hệ điều hành. Trong một số tài liệu, section được đề cập đến như là các đối tượng.

Vùng tiêu đề PE không nằm ngay ở đầu file, khoảng 100 byte đầu tiên của một file PE chứa một mẩu chương trình MS-DOS nhỏ có nhiệm vụ in ra một thông tin ngắn như "*This program cannot be run in MS-DOS mode*" khi ta chạy chương trình Win32 trong môi trường không hỗ trợ Win32.

Hình 4.3 minh họa 4 phần đầu tiên của header file.



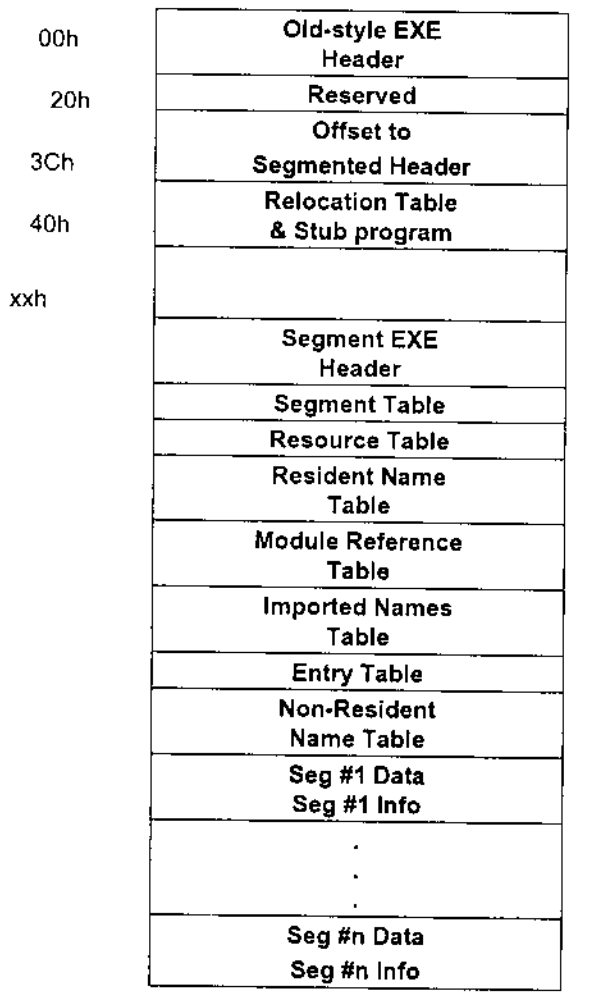
Hình 4.3. **Minh họa 4 phần đầu tiên của tiêu đề**

T rong phần tiêu đề MS-DOS, nếu word tại offset 18h có giá trị là 40h hoặc lớn hơn, thì giá trị của word ở offset 3Ch sẽ chỉ ra offset tới tiêu đề Windows. MS-DOS sẽ sử dụng mẩu chương trình để hiển thị thông báo khi thi hành chương trình trên môi trường MS-DOS. Các thông tin cụ thể về tiêu đề MS-DOS được trình bày ở phần tiêu đề của các file chạy trên môi trường MS-DOS.

Segment EXE Header bao gồm nhiều loại như PE, NE, LE, có nội dung miêu tả, kích thước, vị trí tải file lên bộ nhớ, số section trong file,... Tiếp theo là các section miêu tả về vị trí kích thước của section trên file và vị trí, kích thước của section khi tải lên bộ nhớ.

- ***Phần tiêu đề PE***

Phần header này chứa các thông tin về vị trí, kích thước của phần code và dữ liệu, hệ điều hành file cần để chạy,... Phần tiêu đề PE gồm có 3 phần: phần dấu hiệu nhận dạng (gồm 4 byte có giá trị "PE\0\0"), phần IMAGE_FILE HEADER và phần IMAGE_OPTIONAL_HEADER. Dưới đây là bảng liệt kê đầy đủ các trường trong phần tiêu đề PE, các vị trí được tính tương đối, bắt đầu từ byte đầu tiên của phần tiêu đề PE (không phải là byte đầu tiên tính từ đầu file). Tên các trường giữ nguyên theo định nghĩa của Microsoft để tiện tham khảo về sau.



Hình 4.4. Minh họa tiêu đề EXE

4 byte đầu của phần tiêu đề PE chứa chuỗi nhận dạng *"PE\0\0"*, nếu 4 byte này là *"NE\0\0"* thì có nghĩa là file chạy trong môi trường Windows 16 bit, nếu là *"LE\0\0"* chỉ ra đó là một VxD của Windows 3.x, còn *"LX\0\0"* dành cho file của OS/2 2.0. Tiếp theo chuỗi nhận dạng là phần IMAGE_FILE_HEADER, các trường trong phần này chứa những thông tin cơ bản nhất về file (đã được chỉ ra ở trên). Ngoài ra, còn một số lưu ý sau:

Offset	*Tên trường*	*Ý nghĩa*
00h - 03h	Signature	Dấu hiệu nhận dạng ("PE\0\0")
IMAGE_FILE_HEADER		
04h - 05h	Machine	Yêu cầu CPU cho file
06h - 07h	NumberOfSections	Số section của file
08h - 0Bh	TimeDateStamp	Thời điểm lúc file được tạo ra
0Ch - 0Fh	PointerToSymbolTable	Vị trí của bảng symbol COFF trên file, chỉ được dùng trong file OBJ và file PE với thông tin debug COFF. Thường không sử dụng
10h - 13h	NumberOfSymbols	Số symbol có trong bảng symbol COFF. Thường không sử dụng
14h - 15h	SizeOfOptionalHeader	Kích thước của phần Header mở rộng (Optional Header)
16h - 17h	Characteristics	Cờ chứa các thông tin về file
IMAGE_OPTIONAL_HEADER		
18h - 19h	Magic	Thường có giá trị 10Bh
1Ah	MajorLinkerVersion	
1Bh	MinorLinkerVersion	Version của trình liên kết

1Ch	SizeOfCode	Hầu hết các file chỉ có 1 code section, do đó trường này chứa kích thước của section.text
20h - 23h	SizeOfInitializedData	Tổng kích thước của các section (ngoại trừ code segment) chứa dữ liệu đã được khởi tạo khi chương trình bắt đầu
24h - 27h	SizeOfUninitializedData	Kích thước của các section mà trình nạp cho phép chiếm một khoảng trong bộ nhớ, tuy vậy chúng không chiếm 1 khoảng trong file trên đĩa. Các section này không cần phải có giá trị xác định khi chương trình khởi động. Thường các dữ liệu loại này nằm trong section.bss
28h - 2Bh	AddressOfEntryPoint	Địa chỉ (RVA) chứa câu lệnh đầu tiên được thực hiện
2Ch - 2Fh	BaseOfCode	Địa chỉ (RVA) nơi Code section bắt đầu
30h - 33h	BaseOfData	Địa chỉ (RVA) bắt đầu của Data section
34h - 37h	ImageBase	Địa chỉ (RVA) ánh xạ file trong bộ nhớ khi file được gọi (thường có giá trị là 0x400000)
38h - 3Bh	SectionAlignment	Khi đã ánh xạ file vào bộ nhớ, mỗi section của file được bảo đảm bắt đầu từ một địa chỉ là bội số của giá trị này (với mục đích phân trang, giá trị ngầm định là 0x1000)
3Ch - 3Fh	FileAlignment	Giá trị ngầm định là 0x200. Trong PE file, các section thường có kích thước là bội số của giá trị này
40h - 41h	MajorOperatingSystemVersion	
42h - 43h	MinorOperatingSystemVersion	Version tối thiểu của hệ điều hành để thực hiện file này
44h - 45h	MajorImageVersion	
46h - 47h	MinorImageVersion	Đây là trường do người sử dụng định nghĩa, có thể đặt giá trị cho trường này thông qua chỉ thị /VERSION của trình liên kết. Ví dụ: LINK /VERSION : 2.0 myobj.obj

48h - 49h	MajorSubsystemVersion	
4Ah - 4Bh	MinorSubsystemVersion	Version tối thiểu của hệ thống con chạy chương trình. Giá trị đặc trưng là 3.10 (có nghĩa là Windows NT 3.1)
4Ch - 4Fh	Reserved1	Dành riêng
50h - 53h	SizeOfImage	Tổng kích thước của file bắt đầu từ ImageBase cho đến hết section cuối cùng (kích thước của mỗi section được làm tròn dựa vào SectionAlignment, ví dụ nếu kích thước của section.data trong bộ nhớ là 230h thì kích thước của section trên file sẽ được làm tròn lên 400h nếu SectionAlignment = 200h)
54h - 57h	SizeOfHeaders	Kích thước của toàn bộ phần header và bảng các section
58h - 5Bh	CheckSum	CRC checksum của file
5Ch - 5Dh	Subsystem	
5Eh - 5Fh	DllCharacteristics	Cờ xác định trong trường hợp nào thì một hàm khởi tạo DLL (ví dụ như DllMain) được gọi
60h - 63h	SizeOfStackReserve	Giá trị ngầm định là 0x100000 (1MB). Số lượng bộ nhớ ảo dành cho ngăn xếp của luồng khởi tạo, không phải tất cả vùng nhớ này đều được chấp nhận
64h - 67h	SizeOfStackCommit	Giá trị ngầm định là 0x1000 (1 trang) đối với Microsoft Linker, còn đối với TLINK32 là 2 trang
68h - 6Bh	SizeOfHeapReserve	Số lượng bộ nhớ ảo dành cho vùng heap của tiến trình khởi tạo, có thể dùng hàm GetProcessHeap để tham khảo
6Ch - 6Fh	SizeOfHeapCommit	Giá trị ngầm định là 1 trang
70h - 73h	LoaderFlags	
74h - 77h	NumberOfRvaAndSizes	Số lượng các Entry trong mảng DataDirectory (= 10h)

- Đối với trường Mechine, ta có một số giá trị để xác định CPU:

Giá trị	*Yêu cầu CPU*
0x14d	Intel i860
0x14c	Intel I386 (same ID used for 486 and 586)
0x162	MIPS R3000
0x166	MIPS R4000
0x183	DEC Alpha AXP

- Đối với trường Characteristics, có một số giá trị quan trọng sau

Giá trị	*Ý nghĩa*
0x0001	Không có sự tái định vị trong file
0x0002	Là file chương trình chứ không phải là OBJ hay LIB
0x2000	Là thư viện liên kết động chứ không phải là file chương trình

Tiếp theo phần IMAGE_HEADER_FILE là phần tiêu đề PE mở rộng có cấu trúc IMAGE_OPTIONAL_HEADER, phần này cũng đã được mô tả ở trên. Trường cuối cùng của phần tiêu đề PE là NumberOfRvaAndSizes hiện nay luôn có giá trị ngầm định là 10h. Đây là số phần tử của mảng các phần tử có cấu trúc IMAGE_DATA_DIRECTORY. Khai báo của mảng này như sau:

IMAGE_DATA_DIRECTORY

DATADIRECTORY[IMAGE_NUMBEROF_DIRECTORY_ENTRIES]

Mảng này nằm ngay sau phần tiêu đề PE mở rộng. Các phần tử của mảng chứa RVA và kích cỡ của các thành phần quan trọng trong file, chúng có độ lớn là 8 byte. Một số phần tử ở phía cuối của mảng hiện nay chưa được sử dụng. Phần tử đầu tiên của mảng luôn chứa địa chỉ và kích thước của bảng các hàm Export (nếu có). Phần tử thứ hai chứa địa

chỉ và kích thước của bảng các hàm Import... Mảng phần tử này cho phép trình nạp nhanh chóng tìm ra các section riêng biệt của file mà không cần phải duyệt qua từng section. Dưới đây là bảng liệt kê 16 phần tử của mảng này :

Offset	*Nội dung*
78h - 7Fh	Export table address and size
80h - 87h	Import table address and size
88h - 8Fh	Resource table address and size
90h - 97h	Exception table address and size
98h - 9Fh	Certificate table address and size
A0h - A7h	Base relocation table address and size
A8h - AFh	Debugging information starting address and size
B0h - B7h	Architecture-specific data address and size
B8h - BFh	Global pointer register relative virtual address
C0h - C7h	Thread local storage (TLS) table address and size
C8h - CFh	Load configuration table address and size
D0h - D7h	Bound Import table address and size
D8h - DFh	Import address table address and size
E0h - E7h	Delay import descriptor address and size
E8h - Efh	COM runtime descriptor address and size
F0h - F7h	Reserved

Tiếp sau là bảng các section chứa các section header, mỗi section header có 40 byte có cấu trúc IMAGE_SECTION_HEADER, gồm các trường (tên trường giữ nguyên theo định nghĩa của Microsoft để tiện tham khảo về sau):

Vị trí	*Tên*	*Ý nghĩa*
00h - 07h	Name[IMAGE_SIZEOF_SHORT_NAME]	Tối đa là 8 ký tự, có thể đặt tên tùy ý, không có ký tự kết thúc
08h - 0Bh	VirtualSize	Kích thước section trên bộ nhớ
0Ch - 0Fh	VirtualAddress	Địa chỉ section trên bộ nhớ
10h - 13h	SizeOfRawData	Kích thước của section trên file
14h - 17h	PointerToRawData	Địa chỉ section trên file
18h - 1Bh	PointerToRelocations	Đối với các file EXE, có giá trị 0
1Ch - 1Fh	PointerToLinenumbers	Có giá trị 0 nếu không có số dòng COFF
20h - 21h	NumberOfRelocations	Có giá trị là 0 đối với file EXE
22h - 23h	NumberOfLinenumbers	
24h - 27h	Characteristics	Xác định các thuộc tính như Readable, Shared,...

- Đối với trường Characteristics, ta có một số giá trị quan trọng sau:

Giá trị	*Ý nghĩa*
0x00000020	Section chứa mã (luôn luôn được đặt cùng với cờ *Executable* (0x80000000)
0x00000040	Section chứa dữ liệu đã khởi tạo trước (hầu hết các section đều được đặt cờ này trừ section.*bss* và section thi hành)
0x00000080	Section chứa dữ liệu chưa được khởi tạo (section.bss)
0x00000200	Section chứa các ghi chú hoặc một vài thông tin khác
0x02000000	Section này có thể bị bỏ qua, do đó nó không cần thiết cho chương trình mỗi khi được tải vào bộ nhớ. Điển hình là section.*reloc*

0x10000000	Section này có thể dùng chung (*shareable*), Khi được sử dụng với 1 thư viện liên kết động, dữ liệu trong section này có thể được tất cả các chương trình đang sử dụng cùng 1 thư viện DLL sử dụng
0x20000000	Section là thi hành được (*Executable*). Thuộc tính này luôn được đặt bất cứ khi nào cờ "chứa mã" (0x00000020) được đặt
0x40000000	Section này có thể đọc (*Readable*). Cờ này thường được đặt cho các section trong file.EXE
0x80000000	Section này có thể ghi (*Writeable*)

Mỗi section header có kích thước là 40 byte, tạo thành một chuỗi liên tục trong file. Ta có một số section phổ biến sau:

- Code Section, thường có tên là *.code* hay *.text* chứa mã của chương trình. Section này bắt buộc đối với một file chương trình, nhưng lại không bắt buộc đối với một thư viện DLL. Trong file chương trình, section này luôn phải chứa đầu vào (EntryPoint) của chương trình.
- Data Section, thường được gọi là *.data* chứa các dữ liệu đã được khởi tạo (initialized data).
- Section chứa dữ liệu chưa được khởi tạo gọi là *.udata* hay *.bss*
- Section chứa dữ liệu chỉ đọc, có tên là *.rdata*, giống như section.data nhưng không có cờ Writable
- Section chứa tài nguyên, *.rscr*

4.4. LẬP TRÌNH HỆ THỐNG TRÊN HỆ ĐIỀU HÀNH ĐA NHIỆM WINDOWS

Lập trình có cấu trúc là tư duy lập trình logic làm cho chương trình được viết ra sáng sủa, dễ đọc, dễ hiểu. Phương pháp này đã và vẫn đang phát huy tác dụng. Song cùng với sự phát triển của công nghệ tin học, thì tư duy về lập trình có những bước phát triển mới, đặc biệt khi xây dựng các ứng dụng lớn với sự hợp tác của nhiều người khiến cho lượng dữ liệu trở nên lớn đến mức rất khó kiểm soát khi ghép nối các modul phần mềm riêng biệt với nhau. Một tư duy mới về lập trình ra đời giải

quyết khó khăn này đó là tư duy về lập trình hướng đối tượng. Lập trình hướng đối tượng thực chất là xây dựng một chương trình xử lý mối quan hệ giữa các đối tượng. Mỗi đối tượng được thừa kế từ một lớp sẽ mang đầy đủ đặc tính của lớp (Class) đó. Khái niệm về lớp là một khái niệm mới và khá trừu tượng. Có thể hiểu lớp là một tập hợp các đặc tính mà khi một đối tượng được định nghĩa nằm trong một lớp sẽ mang đầy đủ các đặc tính của lớp đó. Thông qua các thuộc tính (property), phương thức (method) và sự kiện (Event) gắn với đối tượng sẽ cho phép thao tác với các đối tượng đó. Trong lập trình hướng đối tượng có ba đặc trưng lớn là:

- Tính kế thừa (inheritance).
- Tính đóng gói (Encapsulation).
- Đa phương thức (Polymorphism).

Lập trình trên hệ điều hành đa nhiệm windows mang tư duy của lập trình hướng đối tượng cộng với kiến thức về quản trị hệ thống của windows. Chúng ta sử dụng một trình biên dịch cụ thể chạy trên nền của Win32API đó là delphi, song không làm mất tính tính tổng quát khi nêu ra những nguyên lý chung về lập trình trên windows khi dùng các ngôn ngữ khác như VISUAL C, VISUAL BASIC.

Nếu lập trình trên DOS, chương trình có thể trực tiếp can thiệp vào tài nguyên và gọi những hàm chức năng của hệ thống, thì trong lập trình hướng đối tượng trên windows việc cấp phát tài nguyên cũng như khi kích hoạt các hàm chức năng phải chịu sự kiểm soát nghiêm ngặt của hệ điều hành. Chính nhờ cơ chế kiểm soát này mà tài nguyên được bảo vệ, mặt khác nó giảm nhẹ công việc quản lý tài nguyên của chương trình. Cơ chế này chính là cơ chế đa nhiệm chạy trên chế độ địa chỉ ảo và các phép xử lý tiến hành theo theo sự kiện (tức là windows sẽ xử lý các sự kiện yêu cầu chứ không phân biệt là sự kiện của một ứng dụng hay nhiều ứng dụng). Để cấp phát bộ nhớ chẳng hạn thì chương trình sẽ thực hiện công việc theo trình tự sau:

- Chương trình sẽ gửi một yêu cầu cho windows;
- Yêu cầu này được xếp vào một hàng đợi chờ xử lý;

- Khi windows tiếp nhận được yêu cầu, nó sẽ gửi một thông báo cho chương trình biết là yêu cầu đó đã được giải quyết chưa (ở đây là yêu cầu về cấp phát bộ nhớ).

4.4.1. Lập trình Component trên DELPHI

Tài nguyên hệ thống của hệ điều hành đa nhiệm rất phong phú và được tổ chức thành các thư viện khác nhau như thư viên OWL, VCL... Nhờ thư viện các đối tượng trên Windows là OWL (Object Window Library) mà công việc lập trình trên Window trở nên đơn giản hơn. Thư viện OWL giúp cho lập trình viên giảm thiểu những công việc phải lặp lại nhiều lần trong lập trình. Thư viện các Coponent VCL được thiết kế đặc biệt để làm việc trong môi trường của các trình biên dịch hay thông dịch trên windows. Nhờ thư viện VCL này chúng ta rất dễ dàng thiết kế một ứng dụng nhờ khả năng thay đổi tùy ý các đặc tính của những thành phần tạo ra ứng dụng.

VCL chia thành hai mức :

- Mức phát triển các ứng dụng tạo ra những ứng dụng hoàn chỉnh nhờ việc tương tác giữa các thành phần của ứng dụng trong môi trường delph. Khi ở mức này chúng ta sẽ sử dụng VCL để thiết kế các giao diện với người dụng (UI) và các nguyên tố khác liên quan đến ứng dụng. Để làm được điều này chúng ta cần hiểu rõ tất cả những gì liên đến các đối tượng trong VCL như các thuộc tính, các sự kiện, hay các phương thức của chúng.

- Mức thứ hai là mức lập trình tạo ra các component mới. Các thư viện VCL có tính mở và do đó ta có thể mở rộng và làm phong phú thêm các thành phần của nó. Nhưng đây là mức lập trình khó hơn vì cần thiết phải nắm được kỹ thuật lập trình hướng đối tượng cũng như các hiểu biết khác về hệ thống.

Bản chất Component là những phần tử cơ bản sử dụng để thiết kế giao diện với người sử dụng và cung cấp một vài khả năng "không nhìn thấy" trong ứng dụng. Công việc thiết kế giao diện được mô phỏng theo chức năng yêu cầu, nghĩa là có thể nhặt những component trong thư

viện VCL từ một danh sách liệt kê và đặt lên một Form giao diện. Có thể thay đổi các thuộc tính cũng như thêm vào các sự kiện tạo cho component của thư viện phù hợp với mục đích của ứng dụng. Những component trong thư viện có hai loại. Loại thứ nhất là loại vẫn nhìn thấy khi dịch chương trình và một loại sẽ không nhìn thấy khi dịch chương trình (nonvisual component). Mức độ phức tạp của các component trong VCL không toàn toàn như nhau. Có component rất đơn giản chẳng hạn như "Tlabel" và cũng có những component rất phức tạp như "Ttable". Điểm mấu chốt để hiểu rõ về VCL là tìm hiểu các kiểu của component trong VCL như thế nào. Cần phải hiểu biết về những phần chung của các component trong VCL. Và cũng phải hiểu biết tính kế thừa giữa các component trong VCL cũng như mục đích của mỗi mức trong cây thừa kế.

Có bốn kiểu component của thư viện VCL: component chuẩn (standard), component tùy chọn (custom), component đồ họa (graphic) và component không hiển thị (nonvisual).

Component chuẩn như TrichEdit, TTrackBar và TListView và một vài component khác nữa... Chúng là những component chung hay được sử dụng trên Windows. Nếu muốn tham khảo mã nguồn của những component này có thể tham khảo tệp COMCTRLS.PAS. Việc tham khảo mã nguồn này sẽ giúp ích khi tìm hiểu cũng như khi sử dụng thư viện VCL đặc biệt khi muốn tổ chức các component cho thư viện VCL.

Component tùy chọn (custom) có thể kể tên một số như TcustomGrid, NoteBook và TPanel. Những component này không nêu ra những sự kiện và thuộc tính của nó mà chỉ đưa ra những sự kiện và thuộc tính được yêu cầu.

Component đồ họa (graphics) có ích khi muốn hiển thị một điều gì đó bằng phương thức của Windows. Vì không sử dụng yêu cầu thẻ sự kiện của Windows khi lập trình mà chúng ta có thể lấy lại quyền điều khiển (focus) khi cần thiết. Có thể lấy thí dụ một số component như Tlabel, Tshape, Timage, TBevel và TPaintBox là các component đồ hoạ.

Component không hiển thị (Nonvisual) không được hiển thị khi chương trình được biên dịch. Như vậy những component này đóng gói tất cả các chức năng trong đối tượng và cho phép chúng ta thay đổi các đặc tính của component trong VCL thông qua cửa sổ thuộc tính của đối tượng (object inspector) tại thời điểm thiết kế. Có thể thí dụ một số component không hiển thị như TOpenDialog, TTable và TTimer.

Cấu trúc của một component là những lớp đối tượng Pascal đóng gói toàn bộ các chức năng và thuộc tính của các phần tử mà người lập trình có thể thay đổi một cách mềm dẻo khi viết chương trình. Tất cả các component đều có một cấu trúc xác định gồm thuộc tính, phương thức và sự kiện.

- *Thuộc tính*

Thuộc tính của component cho phép giao tiếp với những trường lưu trữ bên trong một lớp. Khi sử dụng thuộc tính có thể đọc hoặc sửa đổi giá trị của một trường nào đó. Nếu không muốn cấm các thao tác truy nhập trực tiếp vào các trường lưu trữ bên trong chúng ta đặt các định nghĩa về lớp của component bên trong từ khóa *Private*.

Những thuộc tính cung cấp khả năng truy nhập đến các trường lưu trữ bên trong theo hai cách: hoặc là truy nhập trực tiếp đến những trường lưu trữ bên trong đó hoặc truy nhập thông qua các phương thức truy nhập (access method). Chúng ta xét một đoạn chương trình minh họa dưới đây:

```
TCustomEdit  = class(TWinControl)
Private
  FMaxLength:Integer;
Protected
  Procedure SetMaxLength(Value:Integer);
  ...
pulished
  property MaxLength: Integer read Fmaxlength write
   SetMaxLength default 0;
...
end;
```

Chúng ta thấy thuộc tính MaxLength truy nhập đến trường lưu trữ trong FMaxLength. Khi component định nghĩa một thuộc tính thì nó gồm có tên thuộc tính đó, kiểu thuộc tính, một miêu tả *read* một miêu tả *write* và một lựa chọn giá trị ngầm định. Miêu tả *read* đặc tả việc các trường lưu trữ trong đọc vào như thế nào. Thuộc tính maxlength trực tiếp đọc gía trị từ trường lưu trữ trong FMaxLength. Miêu tả *write* đặc tả phương thức mà nhờ nó trường lưu trữ được gán giá trị. Tóm lại trong thuộc tính MaxLength, một phương thức truy nhập SetMaxLength() thường gán giá trị tới trường lưu trữ FMaxLength.

Những phương thức truy nhập chỉ có duy nhất một tham số có kiểu giống như kiểu của thuộc tính. Mục đích của việc khai báo một phương thức truy nhập trong miêu tả *write* là gán giá trị của tham số tới trường lưu trữ bên trong mà thể hiện chính là giá trị của thuộc tính. Lý do của việc sử dụng lớp các phương thức để gán giá trị nhằm bảo vệ trường lưu trữ không nhận những giá trị sai, và thực hiện tốt các tác động khác nếu được yêu cầu. Có thể minh họa bằng một ví dụ dưới đây:

```
Procedure TCustomEdit.SetMaxLength(Value:Integer);
begin
  If FMaxLength <> Value then
    begin
      FMaxLength := Value;
      if HandleAllocated then
      SendMessage(Handle, EM_LIMITTEXT,Value,0);
    end;
end;
```

Như vậy, đầu tiên phương thức này kiểm tra xem giá trị đưa vào có bằng giá trị mà thuộc tính đang lưu giữ không. Nếu không bằng thì nó thực hiện việc gán cho trường lưu trữ trong FMaxLength và tiếp đó Sendmessage() đưa ra một thông báo của Windows EM_LIMITTEXT tới của sổ mà TCustomEdit đóng gói. Thông báo này bị hạn chế về số lượng ký tự mà người sử dụng có thể nhập vào một cửa sổ soạn thảo (Control Edit). Khi gọi Sendmessage() trong phương thức truy nhập *write* của

thuộc tính được hiểu như một hiệu ứng tác động khi gán những giá trị cho thuộc tính.

Một điểm thuận lợi khi sử dụng phương thức truy nhập tới những trường lưu trữ trong là nhờ đó mà khi tạo một component mới có thể bổ xung những trường mới mà không ảnh hưởng gì tới tính chất component, Mặc dù khi sử dụng những trường này ta không nhìn thấy trong hộp cuộn danh sách thuộc tính nhưng chúng ta vẫn có thể truy nhập đến các trường này nhờ các hàm truy nhập. Như vậy một phương thức truy nhập có thể thay đổi kiểu của giá trị trả về mà giá trị này có thể khác với kiểu của trường lưu trữ trong.

Một thuận tiện nữa khi sử dụng tham số thuộc tính của component là chúng có thể dễ dàng thay đổi giá trị của chúng khi thiết kế. Khi một thuộc tính được khai báo trong phần *Published* của component, thì nó sẽ xuất hiện trên cửa sổ cuộn danh sách các thuộc tính. Do vậy mà có thể thay đổi giá trị của những thuộc tính này từ chính cửa sổ hiển thị danh sách thuộc tính.

Kiểu dữ liệu của các đối tượng của Pascal dùng cho các thuộc tính tuân thủ các quy tắc chuẩn. Kiểu của chúng sẽ quyết định cách thức mà chúng ta nhập giá trị trong cửa sổ cuộn danh sách các thuộc tính. Thuộc tính có thể nhận một trong những kiểu theo bảng 4.1.

- ***Phương thức***

Component là những đối tượng mà gắn với chúng là những phương thức. Phương thức thực chất là những [illegible] thủ tục được đóng gói trong một lớp thao tác với dữ liệu được định nghĩa trong class đó.

- ***Sự kiện***

Là thể hiện của một hành động, điển hình là một tác động của hệ thống giống như việc bấm chuột hoặc nhấn một phím trên bàn phím. Những component chứa những thuộc tính đặc biệt gọi những sự kiện. Người sử dụng component có thể gán một đoạn mã cho một sự kiện để nó có thể được thực hiện khi sự kiện xuất hiện.

Nếu xem xét những sự kiện của component TEdit ta sẽ tìm thấy những sự kiện như OnChange, OnClick và OnDblClick. Khi xây dựng component, thì sự kiện (event) là những con trỏ thực sự tới các phương thức. Khi người sử dụng một component gán một đoạn mã chương trình cho một sự kiện, tức là chúng ta đã tạo ra một thẻ sự kiện (event handler). Thí dụ, khi chúng ta ấn đúp chuột trên một sự kiện trong trong hộp cuộn các sự kiện cho component, Delphi sinh ra một phương thức để đưa mã chương trình vào đó, như vậy đoạn mã sau đây cho sự kiện OnClick của component Tbutton.

***Bảng 4.1.* Những kiểu thuộc tính**

Kiểu thuộc tính	*Thể hiện trong hộp cuộn danh sách các thuộc tính*
simple	những thuộc tính có kiểu số, ký tự và chuỗi xuất hiện trong cửa sổ cuộn các thuộc tính như là những số, những ký tự và chuỗi tương ứng. Người sử dụng có thể gõ và soạn thảo giá trị cho những thuộc tính một cách trực tiếp.
Enumerated	Những thuộc tính nhận kiểu không thứ tự (gồm có cả kiểu boolean) sẽ hiển thị những gái trị được định nghĩa trước trong mã nguồn (source code). Người sử dụng có thể lựa chọn giá trị trong cột giá trị nhờ việc bấm đúp chuột. ở đây cũng có thể là một hộp cuộn danh sách liệt kê tất cả các giá trị có thể của kiểu không thứ tự này.
set	Những thuộc tính có kiểu tập hợp xuất hiện trong hộp cuộn danh sách các thuộc tính như là một tập hợp. Nhờ việc mở rộng các tập hợp, người sử dụng có thể coi mỗi phần tử trong tập hợp như giá trị logic. Là True nếu phần tử thuộc tập hợp và là False nếu không thuộc.
Object	Những thuộc tính mà bản thân chúng cũng là đối tượng thương có môi trường soạn thảo để nhúng nó. Tuy vậy nếu đối tượng là một thuộc tính cũng có những thuộc tính published thì trong hộp cuộn các thuộc tính cho phép người sử dụng mở rộng danh sách đối tượng và tự soạn thảo chúng. Những thuộc tính của đối tượng cùng thừa kế từ TPersistent.
Array	Những thuộc tính có kiểu mảng cần phải có những trình soạn thảo của nó. Trong thanh cuộn danh sách các thuộc tính không xây dựng để hỗ trợ cho việc soạn thảo mảng các thuộc tính.

```
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject);
begin
end;
```

- ***Bổ xung sự kiện khi dịch chương trình***

Khái niệm sự kiện (event) đã sáng tỏ khi chúng ta hiểu nó như những con trỏ đến các phương thức khi ta gán một thẻ sự kiện tới một sự kiện. Thí dụ việc liên kết những thẻ sự kiện tới sự kiện Onclick của component TButton, thì đầu tiên cần miêu tả và định nghĩa phương thức có ý định gán tới sự kiện OnClick của Button. Phương thức này có thể thuộc form chủ sở hữu của TButton như đoạn mã nguồn sau đây:

```
Tform1=class(TForm)
  Button1:Tbutton;
  ....
Private
  MyonclickEvent(Sender:TObject);//Phương thức do lập trình
                                  viên miêu tả
end;
....
{phương thức của lập trình viên được định nghĩa dưới dây}
```

```
Procedure TForm1.MyOnClickEvent(Sender:TObject);
begin
  {Đoạn mã chương trình đặt vào đây}
end;
```

Với đoạn chương trình trên ta thấy một phương thức được người dùng định nghĩa là MyOnClickEvent(), nó phục vụ như là một thẻ sự kiện cho button1.Onclick. Những dòng chương trình sau chỉ rõ cách thức gán phương thức này cho sự kiện Onclick của button1, và phương thức này luôn được gọi khi sự kiện xuất hiện.

```
Procedure TForm1.FormCreate(Sender:TObject);
begin
  Button1.OnClick:=MyOnClick;
end;
```

Kỹ thuật này có thể sử dụng thêm những thẻ sự kiện khác nhau trên cơ sở những điều kiện khác nhau trong đoạn mã chương trình. Ngoài ra, ta có thể hủy bỏ một thẻ sự kiện bằng việc gán *nil* tới sự kiện như dưới đây

```
Button1.OnClick:= nil;
```

Việc gán thẻ sự kiện tại thời điểm biên dịch chương trình thực chất tương tự với điều xảy ra khi ta tạo một thẻ sự kiện thông qua thanh cuộn danh sách các sự kiện. Delphi sẽ tự động sinh ra cách mô tả phương thức. Chúng ta không thể nào gán một phương thức bất kỳ cho một thẻ sự kiện đặc biệt vì những thuộc tính của sự kiện là những con trỏ phương thức. Thí dụ, một phương thức OnMouseDown là của kiểu TMouseEvent, một thủ tục được định nghĩa dưới dây:

```
TmouseEvent=procedure(Sender : TObject; Button: TMouseButton;
        Shift:TShiftState;x,y:Integer)   of object;
```

Vì vậy những phương thức này trở thành thẻ sự kiện cho sự kiện phải định nghĩa. Những phương thức đó cần phải giống về kiểu, số lượng và về trật tự của các tham số.

Giống như thuộc tính, các sự kiện trỏ tới những trường dữ liệu được khai báo trong từ khóa *Private* của component. Những trường dữ liệu này là những thủ tục kiểu như TmouseEvent. Thí dụ:

```
TControl =Class(TComponent)
private
  FOnMouseDown: TMouseEvent;
protected
  Property OnMouseDown: TMouseEvent read   FOnMouseDown
                                    write FOnMouseDown;
public
end;
```

Tóm tại chúng ta đã đưa ra cách thức tham chiếu đến những trường dữ liệu mang tính cục bộ trong một lớp của một component nào đó. Và ta cũng đã thấy như thế nào là sự kiện, thay đổi những thuộc tính, tham chiếu đến những những trường các con trỏ phương thức cục bộ của một component. *Điều quan trọng nhất cần nhấn mạnh đó là một sự kiện (event) được hệ thống chấp nhận xử lý tương đương như một ngắt được xử lý.*

- ***Khả năng phân luồng***

Một trong những đặc tính của component là chúng có khả năng phân luồng. Sự phân luồng (streaming) là cách thức lưu trữ một component và những thông tin về các giá trị của thuộc tính vào file. Sự phân luồng của Delphi là khả năng kiểm soát các vấn đề trên cho người lập trình. Trong thực tế những file có phần mở rộng.dfm được tạo bởi Delphi chính là những file tài nguyên chứa những thông tin về khung và những component của khung đó. Với người lập trình thì khi tổ chức component không cần quan tâm đến vấn đề này vì Delphi tự động làm việc đó.

Kiểu của một component nào đó có thể là chủ của component khác. Một chủ thể của một component nào đó được đặc tả nhờ thuộc tính owner. Khi một component là chủ thể của một component khác thì nó

phải thực hiện chức năng giaỉ phóng cho component mà nó sở hữu khi nó bị huỷ bỏ. Một trường hợp điển hình là một form là chủ sở hữu của các component xuất hiện trên nó. Khi ta đặt một component trên bề mặt một form khi thiết kế, thì form sẽ tự động trở thành chủ sở hữu của tất cả các component đặt trên nó. Khi ta tạo ra một component tại thời điểm dịch chương trình, thì cần phải chuyển quyền sở hữu của component đến constructor *create* của component. Đoạn chương trình sau sẽ chỉ ra các chuyển quyền sở hữu đó sử dụng một biến ẩn của form.

```
Mybutton : = TButton(self);
```

Khi một form bị huỷ bỏ, các thể hiện của lớp TButton là Mybutton cũng được huỷ bỏ.

Chúngta cũng có thể tạo ra một component không thuộc sở hữu bằng cách đưa giá trị *nil* đến phương thức Create() của component. Tuy vậy ta phải giải quyết một điều là phải tự thực hiện việc huỷ bỏ và việc cấp phát bộ nhớ cho component đó. Thí dụ:

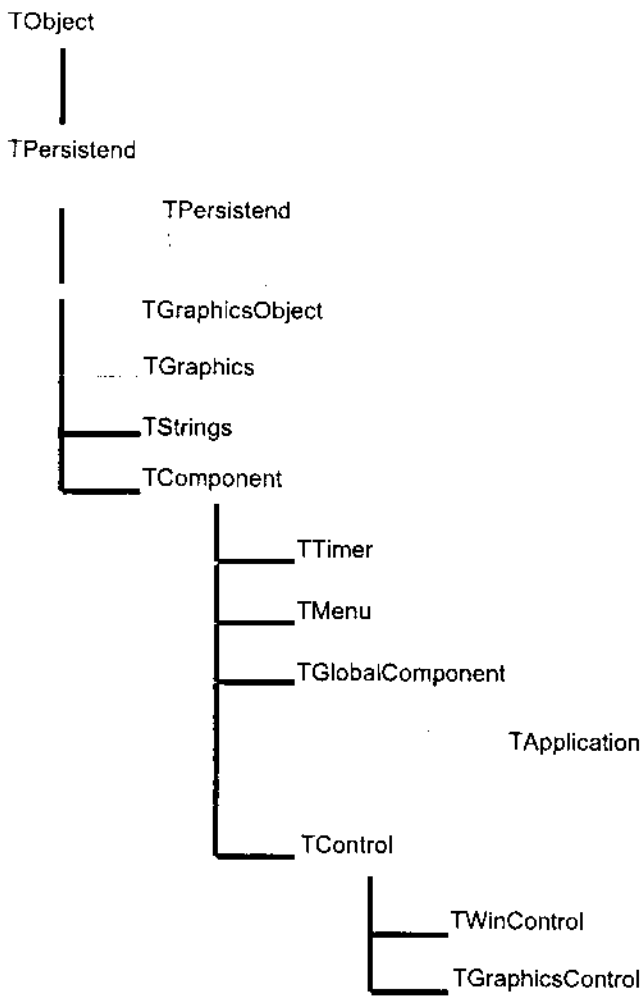
```
Mytable :=TTable.create (nil)
  try
  {một số công việc thực hiện với TTable}
  finally
  mytable.free;
end;
```

Khi sử dụng kỹ thuật này cần lưu ý một điểm là phải sử dụng mệnh đề try...finally để chắc chắn rằng ta đã giải phóng tất cả các tài nguyên được cấp phát.

- ***Tính thừa kế***

Không mâu thuẫn với tính sở hữu đó là tính thừa kế. Một component có thể là lớp cha của một component khác. Những component cha phải giải quyết việc gọi những phương thức của component con và bắt chúng phải vẽ lại bản thân chúng. Một component cha được đặc tả thông qua thuộc tính *parent*.

Một component cha không cần thiết phải có sở hữu chủ của nó. Khi một component có những component con bị huỷ bỏ thì phải gọi những phương thức của lớp con để giải phóng bộ nhớ cho nó.



***Hình 4.5.* Cây thừa kế của component**

Component thường không thừa kế trực tiếp từ TObject. Trong thư viện VCL luôn có những con cháu của Tobject mà component mới có thể dẫn xuất. Trong những lớp hiện có này cung cấp nhiều chức năng thỏa

mãn yêu cầu cho component mới. Chỉ khi không muốn thừa kế từ bất cứ lớp nào thì bắt buộc phải thừa kế từ Tobject.

Hình 4.5 là cây thừa kế của những component hiển thị khi chạy. Đây là cây thừa kế của những đối tượng trong thư viện VCL.

Phương thức Create() và Destroy() của Tobject thực hiện các công việc như cấp phát và giải phóng bộ nhớ cho một thể hiện của một đối tượng.

Trong thực tế thì constructor Tobject.create() trả về một tham chiếu đến đối tượng được khởi tạo. Tobject có một vài hàm đưa ra một vài thông tin có ích về đặc tả đối tượng VCL. Nó sử dụng hầu hết các phương thức nội tại của Tobject. Lập trình viên sẽ cơ bản sử dụng những phương thức create() và Free() của Tobject hoặc là những lớp con cháu của nó. Chương trình dưới đây sẽ minh họa về khả năng thừa kế của các component.

```
Unit Cây_Thừa_Kế;
Interface
Uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs;
Type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    TreeView1: TTreeView;
    Table1: TTable;
    Database1: TDatabase;
    PrintDialog1: TPrintDialog;
    FontDialog1: TFontDialog;
```

```
    ListBox1: TListBox;
    StringGrid1: TStringGrid;
    procedure Button1Click(Sender: TObject);
    procedure Edit1Enter(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
   Procedure WriteClassInfo(Sender:TObject);
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
Implementation
{$R *.DFM}
Procedure TForm1.WriteClassInfo(Sender:TObject);
var
  ParentClass:TClass;
begin
  With Listbox1.items do begin
      clear;
    add('Class name:'+Sender.ClassName);
     add('Ancestry');
     Parent.class:=Sender.Classparent;
    while ParentClass <> nil do begin
          Add('    '+ParentClass.ClassName);
          Parentclass :=ParentClass.ClassParent;
    end;
 end;
```

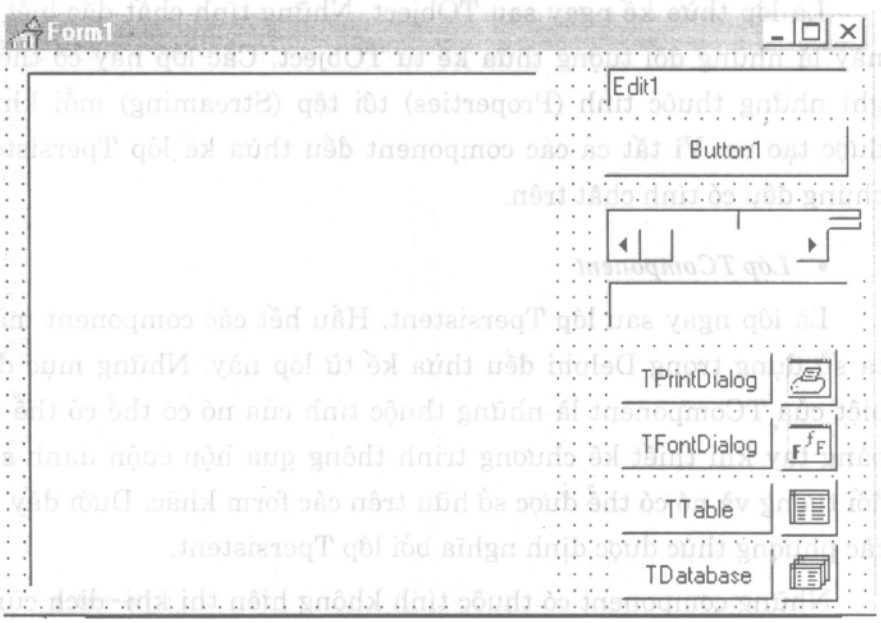
```
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
end;
procedure TForm1.Edit1Enter(Sender: TObject);
begin
  WriteClassInfo(Sender);
end;
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  WriteClassInfo(PrintDialog1);
end;
procedure TForm1.BitBtn2Click(Sender: TObject);
begin
  WriteClassInfo(FontDialog1);
end;
procedure TForm1.BitBtn3Click(Sender: TObject);
begin
  WriteClassInfo(Table1);
end;
procedure TForm1.BitBtn4Click(Sender: TObject);
begin
  WriteClassInfo(Database1);
end;
procedure TForm1.FormActivate(Sender: TObject);
begin
  WriteClassInfo(Edit1);
end;
end.
```

Trong từ khóa *private* của Form1 là thủ tục riêng WriteClassInfo(). Đây là một thủ tục hiển thị tên lớp của component và những ớp cha của chúng trong hộp TListBox trên Form1.

Biến cục bộ TClass được định nghĩa trong unit TYPINFO của Delphi. Unit này ta có thể thêm vào và có thể truy nhập đến đối tượng TClass khi đặt tên của unit sau mệnh đề *uses*.

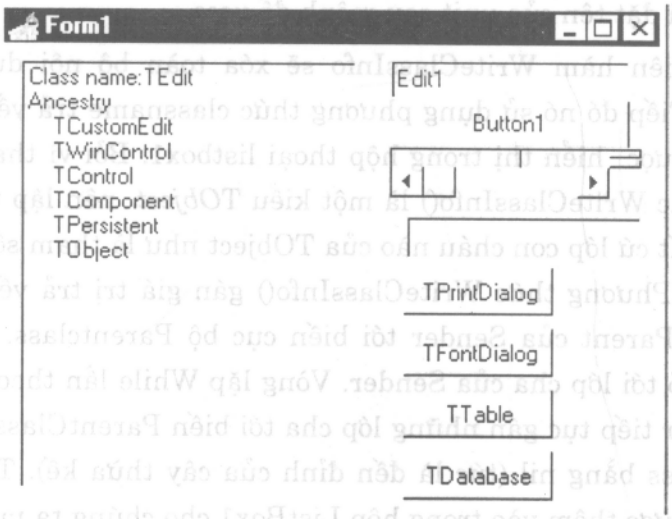
Đầu tiên hàm WriteClassInfo sẽ xóa toàn bộ nội dung của hộp listbox1. Tiếp đó nó sử dụng phương thức classname trả về tên của lớp. Tên này được hiển thị trong hộp thoại listbox1. Bởi vì tham số *sender* của thủ tục WriteClassInfo() là một kiểu T*Object* nên lập trình viên có thể đưa bất cứ lớp con cháu nào của TObject như là tham số của phương thức này. Phương thức WriteClassInfo() gán giá trị trả về của phương thức classParent của Sender tới biến cục bộ Parentclass. ParentClass bây giờ trỏ tới lớp cha của Sender. Vòng lặp While lần theo cây thừa kế của Sender tiếp tục gán những lớp cha tới biến ParentClass cho đến khi ParentClass bằng nil (tức là đến đỉnh của cây thừa kế). Tên của lớp ở mỗi mức được thêm vào trong hộp ListBox1 cho chúng ta một danh sách lớp cha ông của lớp đưa ra.

Khi chạy chương trình này sẽ cho kết quả như hình 4.6 và 4.7 thể hiện.



Hình 4.6. **Form của chương trình**

Khi chúng ta dịch chương trình trên kết quả sẽ được đưa ra như hình 4.4.



Hình 4.7. **Nội dung hiển thị trong cửa sổ chính**

- ***Lớp Tpersistent***

Là lớp thừa kế ngay sau TObject. Những tính chất đặc biệt của lớp này là những đối tượng thừa kế từ TObject. Các lớp này có thể đọc và ghi những thuộc tính (Properties) tới tệp (Streaming) mỗi khi chúng được tạo ra. Vì tất cả các component đều thừa kế lớp Tpersistent nên chúng đều có tính chất trên.

- ***Lớp TComponent***

Là lớp ngay sau lớp Tpersistent. Hầu hết các component mà chúng ta sử dụng trong Delphi đều thừa kế từ lớp này. Những mục đích đặc biệt của TComponent là những thuộc tính của nó có thể có thể kéo thả bằng tay khi thiết kế chương trình thông qua hộp cuộn danh sách các đối tượng và nó có thể được sở hữu trên các form khác. Dưới đây là bảng các phương thức được định nghĩa bởi lớp Tpersistent.

Những component có thuộc tính không hiên thị khi dịch cũng được thừa kế từ lớp này vì thế chúng cũng có khả năng kéo thả tại thời điểm thiết kế.

Những thuộc tính của TComponent minh họa trên bảng 4.4.

Trong lớp TComponent cũng định nghĩa một vài thuộc tính và phương thức đáng quan tâm.

Bảng 4.3. **Phương thức định nghĩa bởi Tpersistent**

Phương thức	*Mục đích*
Assign()	Đây là phương thức có tính Public cho phép một component dữ liệu do nó tự tổ chức cho một component khác.
Assignto()	Đây là một phương thức có tính Protected cho phép một component bỏ qua sự thực hiện của phương thức Assign(). Lớp Tpersistent bản thân nó tự có cơ chế kiểm tra lỗi và bảo vệ khi phương thức này được gọi. Ví dụ như lớp TClipboard là một đối tượng thực hiện gọi phương thức này.
DefineProperties()	Đây là phương thức có tính Protected cho phép người lập trình component cách thức định nghĩa những thuộc tính đặc biệt cũng như việc mở rộng lưu trữ của component. Ngầm định, một component tự động gán những thuộc tính của no là published.

Phương thức của lớp TComponent: Tcomponent định nghĩa một vài phương thức có thể thực hiện khả năng sở hữu các component khác và khả năng kéo thả bằng tay khi thiết kế:

- Trong lớp TComponent định nghĩa constructor Create() của component. Constructor này tạo ra một thể hiện của component và chuyển nó tới một chủ sở hữu trên cơ sở những tham số đưa tới nó.

- Destructor Tcomponent.Destroy() giải quyết việc giải phóng tài nguyên khi component bị huỷ bỏ.

- Phương thức Tcomponent.Destroying() giải quyết việc thiết lập một component và những sở hữu của component sang trạng thái cần thiết.

- Phương thức Tcomponent.DestroyComponents giải quyết việc huỷ bỏ các component. Sẽ không có khả năng khôi phục nội dung cũ khi sử dụng phương thức này.

- Phương thức TComponent.FindComponent() rất thuận tiện khi muốn trỏ đến một component mà ta chỉ biết tên. Có thể giả sử rằng trên mainform có một component TEdit có tên là Edit1. Khi ta không có một tham chiếu đến component này, ta có thể trả về một con trỏ thể hiên này bằng cách thực hiện đoạn mã sau:

```
EditInstance :=FindComponent.('edit1');
```

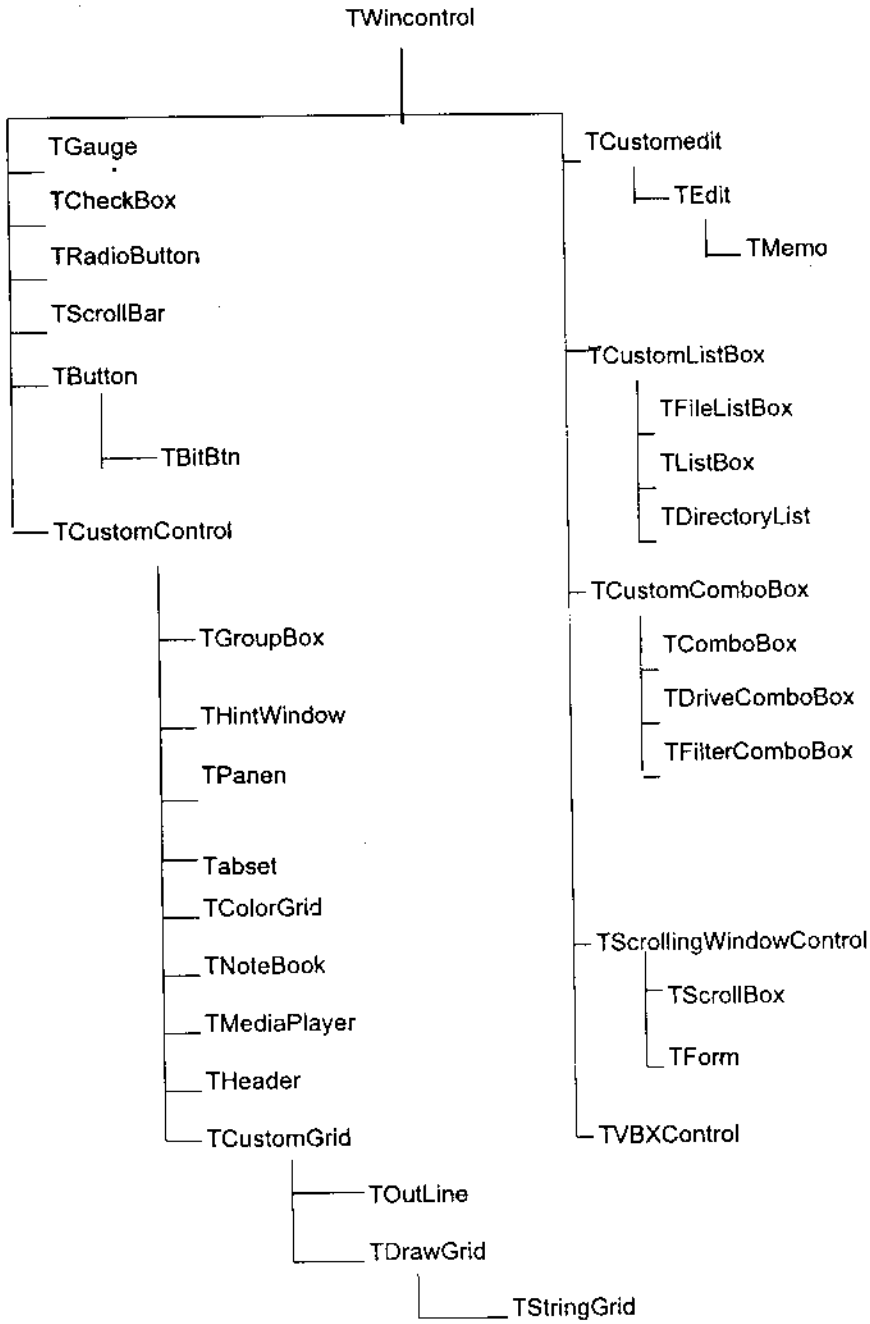
Trong thí dụ này thì EditInstancec có kiểu là TEdit. Lập trình viên có thể trỏ đến thể hiện trả về của phương thức FindComponent() như sau:

```
TEdit(FindComponent.('Edit1')).Text:='Hello';
```

- Phương thức TComponent.GetParentComponent() trả về một thể hiện tới component cha của component.

***Bảng 4.4.* Thuộc tính của TComponent**

Tên componet	*Mục đích*
Owner	Những con trỏ tới chủ sở hữu của component
ComponentCount	Lưu giữ số lượng component mà component sở hữu
ComponentIndex	Vị trí của component này trong danh sách chủ sở hữu nó. Component đầu tiên trong danh sách có thứ tự là Zero
ComponentState	Thuộc tính này lưu giữ trạng thái hiện tại của một component có kiểu TComponentState.
ComponentStyle	Khống chế những đặc tính khac nhau của component. CsInheritable và csCheckPropAvail là hai giá trị có thể gán cho thuộc tính này
Name	Lưu giữ tên của một component
Tag	Một thuộc tính có kiểu integer nó không có định nghĩa về ý nghĩa của nó vì vậy nó có thể ám chỉ là những con trỏ tới cấu trúc dữ liệu hoặc những thể hiện của đối tượng.
DesignInfo	Đượoc sử dụng bởi người thiết kế Form. Không thực hiện truy nhập tới thuộc tính này.



Hình 4.8. **Cây thừa kế từ TWincontrol**

- Phương thức TComponent.HasParent() trả về trị Boolean có hoặc không có component cha.

- Phương thức TComponent.InsertComponent() thêm một component nó nó được sở hữu bởi component gọi.

- Phương thức Tcomponent.RemoveComponent() gỡ bỏ một component bị sở hữu từ component gọi.

Lớp TControl định nghĩa nhiều thuộc tính, phương thức và các sự kiện. Lớp TControl gồm có những thuộc tính như Top, Left, Width, Height, ClientRect, ClientWidth, ClientHieght, Visible, Enabled, Color, Font, Text và Caption.

Lớp TControl cũng định nghĩa một số sự kiện chuẩn như OnClick, OnMouseDown, OnDblClick, OnMouseMove, OnMouseUp, OnDragOver, OnDragDrop và EndDrag.

Hầu hết các điều khiển chuẩn trên của Delphi đều được dẫn suất từ những thừa kế của lớp này : TWincontrol và TGraphicControl.

Lớp TWinControl: điều khiển chuẩn trên Windows thừa kế từ lớp TWincontrol. Những điều khiển chuẩn là những đối tượng ứng dụng trên Windows như edit, listbox, combo box và button. Delphi đóng gói những thuộc tính của những điều khiển chuẩn thay thế việc sử dụng những hàm của Windows API còn hình 4.8 là cây thừa kế từ TWincontrol.

- ***Những thuộc tính cơ bản của TWinControl***

TWincontrol định nghĩa một vài thuộc tính có thể áp dụng để thay đổi khả năng hội tụ và xuất hiện của điều khiển.

- TWinControl.Brush là thuộc tính sử dụng để vẽ những mẫu và hình dạng của component.

- TWincontrol.controls là mảng thuyộc tính của một danh sách các điều khiển mà có TWincontrol là lớp cha.

- TWincontrol.Controlcount là thuộc tính lưu trữ số đếm các điều khiển có TWincontrol là lớp cha.

TWincontrol.Ctl3D là thuộc tính đặc tả có hay không sử dụng hiệu ứng ba chiều.

TWincontrol.Handle tương ứng với một thẻ sự kiện của các đối tượng cửa sổ Windows mà được TWincontrol đóng gói. Những thẻ sự kiện này được các lập trình viên gửi cho hàm Windows API để yêu cầu tham số của một thẻ.

TWincontrol.HelpContext lưu giữ một số ngữ cảnh trợ giúp tương ứng với nội dung lưu trong một File.

TWincontrol.Showing chỉ ra điều khiển hiển thị.

TWincontrol.TabStop lưu giữ giá trị logíc về sử dụng Tab.

TWincontrol.TabOrder dặc tả vị trí trong danh sách điều khiển tab của các điều khiển hiện đang có mặt.

- ***Phương thức của TWincontrol***

TWincontrol cũng đưa ra một vài phương thức thực hiện việc tạo ra một của sổ, hội tụ điều khiển, thu nhận sự kiện. Thí dụ như là CreateParams(), CreateWnd(), CreateWindowHandle(), DestroyWnd(), DestroyWindowhandle(), và ReCreateWnd(), CanFocus(), Focused(), AlignControls(), EnableAlign(), DisableAlign() và ReAlign().

- ***Các sự kiện của TWincontrol***

Một số sự kiện liên quan đến các sự kiện bàn phím như : OnkeyDown, OnKeyPress, và OnKeyUp, OnEnter và OnExit.

Lớp TGraphicControl không giông như TWindowControl. Nó không dùng bất cứ thẻ sự kiện nào của window và do vậy nó không thể thực hiện việc thu nhận lại điều khiển. Chúng cũng không thể là lớp cha của bất cứ điều khiển nào. Điểm lợi của TGraphicControl là nó không chiếm tài nguyên của hệ thống. Nó thực hiện việc vẽ nhờ TgraphicControl nên nhanh hơn là sử dụng các điều khiển chuẩn của Window.

Lớp TCustomControl: Những điều khiển này có các hàm chức năng giống như những thừa kế từ TWincontrol trừ một số điểm đặc biệt và một số đặc tính tương tác.

Những lớp khác: Những lớp này đều có những đặc tính điển hình của các component khác và trực tiếp thừa kế từ lớp TPersistent.

LớpTcanvas: Thuộc tính canvas có kiểu TCanvas cung cấp điều khiển cho cửa sổ và diễn tả việc vẽ trên bề mặt của điều khiển.

- ***Đa phương thức***

Một trong những đặc tính qua trọng trong lập trình hướng đối tượng đó là tính đa phương thức (Polymorphism). Đa phương thức là khả năng một đối tượng đặt trên Form sẽ định nghĩa nó như bất kỳ lớp thừa kế nào của form.

4.4.2. Tạo Component mới

Trình tự xây dựng Componentmới

- ***Tạo component mới phải tuân thủ các bước sau***

- Chọn lớp kế thừa.

- Tạo Unit *component.*

- Bổ xung thuộc tính, phương thức và sự kiện (ngắt) mới cho component.

- Thử nghiệm component.

- Đăng ký vào thư viện tài nguyên hệ thống.

- Tạo file HELP cho component mới.

- ***Thí dụ minh họa - Tạo Component đồng hồ nhịp 1/2 phút***

Lớp kế thừa được chọn là lớp *TTIMER.*

Unit được tạo có tên HalfMin.

```
unit halfmin;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls;
type
```

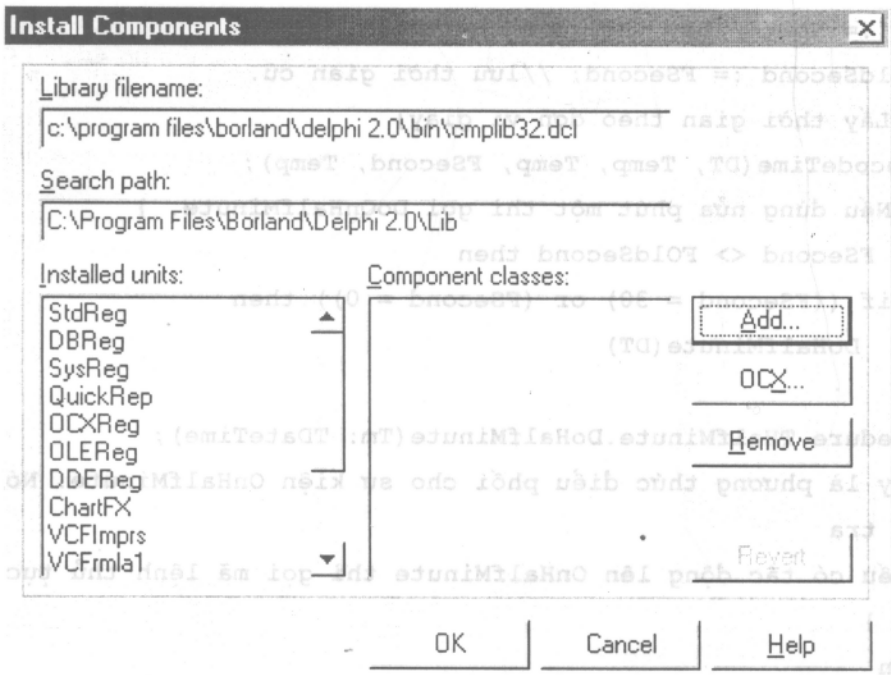
```
  { Định nghĩa thủ tục xử lý ngắt đồng hồ cứ sau  1/2  phút -
thẻ sự kiện}
 TTimeEvent = procedure(Sender: TObject; Tm: TDateTime) of
object;
  THalfMinute = class(TComponent)
 private
    FTimer: TTimer;   // THalfMinute sở hữu component TTimer.
    { Định nghĩa các trường trỏ tới sự kiện của người lập
trình }
    FOnHalfMinute: TTimeEvent;
    FOldSecond, FSecond: Word; // Các biến cần thiết
   {  Định nghĩa thủ tục FTimerTimer gán cho
FTimer.OnClick.
      Thủ tục phải có kiểu TNotifyEvent là kiểu của
TTimer.OnClick. }
procedure FTimerTimer(Sender: TObject);
protected
    { Định nghĩa phương thức điều phối cho OnHalfMinute
event. }
procedure DoHalfMinute(Tm: TDateTime);
public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
published
    { Định nghĩa thuộc tính hiển thi trong hộp Object
Inspector }
    property OnHalfMinute: TTimeEvent read FOnHalfMinute
             write FOnHalfMinute;
  end;
procedure Register;
implementation
constructor THalfMinute.Create(AOwner: TComponent);
```

```
{ Khởi tạo cấu trúc TTimer cho FTimer. Nó thiết lập mọi thuộc
tính cho
  FTimer, kể cả sự kiện OnTimer  thể hiện qua phương thức
THalfMinute's FTimerTimer(). Lưu ý rằng FTimer.Enabled =true
chỉ trong
giai đoạn component chạy còn = false trong giai đoạn thiết kế
component. }
begin
  inherited Create(AOwner);
 { Nếu trong giai đoạn thiết kế  thì cấm FTimer. }
  if not (csDesigning in ComponentState) then begin
    FTimer := TTimer.Create(self); // Tạo TTimer.
    FTimer.Enabled := True;
   { Thiết lập các thuộc tính khác, kể cả thẻ sự kiện
FTimer.OnTimer }
    FTimer.Interval := 500;
    FTimer.OnTimer := FTimerTimer;
   end;
end;
destructor THalfMinute.Destroy;
begin
  FTimer.Free;        // giải phóng FTimer
inherited Destroy; // gọi destructor
end;
procedure THalfMinute.FTimerTimer(Sender: TObject);
{ Phương thức này phục vụ như thẻ sự kiện FTimer.OnTimer và
được gán cho
  FTimer.OnTimer tại thời điểm chạy trong cấu trúc
THalfMinute's. Phương
thức này lấy thời gian hệ thống, tính ra phút và nửa phút.
Nếu một trong hai
giá trị trên xuất hiện nó sẽ gọi phương thức điều phối
OnHalfMinute và
```

```
DoHalfMinute. }
var
  DT: TDateTime;
  Temp: Word;
begin
  DT := Now; //Lấy thời gian hệ thống.
  FOldSecond := FSecond; //lưu thời gian cũ.
  { Lấy thời gian theo đơn vị giây}
  DecodeTime(DT, Temp, Temp, FSecond, Temp);
  { Nếu đúng nửa phút một thì gọi DoOnHalfMinute. }
  if FSecond <> FOldSecond then
    if ((FSecond = 30) or (FSecond = 0)) then
      DoHalfMinute(DT)
end;
procedure THalfMinute.DoHalfMinute(Tm: TDateTime);
{ Đây là phương thức điều phối cho sự kiện OnHalfMinute. Nó
kiểm tra
xemnếu có tác động lên OnHalfMinute thì gọi mã lệnh thủ tục
này. }
begin
  if Assigned(FOnHalfMinute) then
    FOnHalfMinute(Self, Tm);
end;
{Đăng ký vào thư viện tài nguyên hệ thống lớp có tên DDG}
procedure Register;
begin
  RegisterComponents('DDG', [THalfMinute]);
end;
end.
```

Bước tiếp theo là đăng ký cài đặt vào hệ thống. Muốn làm như vậy cần mở Menu ISTALL từ Menu chính COMPONENT. Một hộp thoại hiện ra (hình 4.9).

Nhấn nút ADD sẽ mở cửa sổ con nơi chứa tên Unit component THalfMinute cùng đường dẫn của thư mục nơi chứa THalfMinute. Đánh dấu vào Unit cần chọn và nhấn OPEN. Cuối cùng nhấn vào nút OK để hoàn tất việc cài đặt component mới.



Hình 4.9. **Cửa sổ Memu ISTALL component mới**

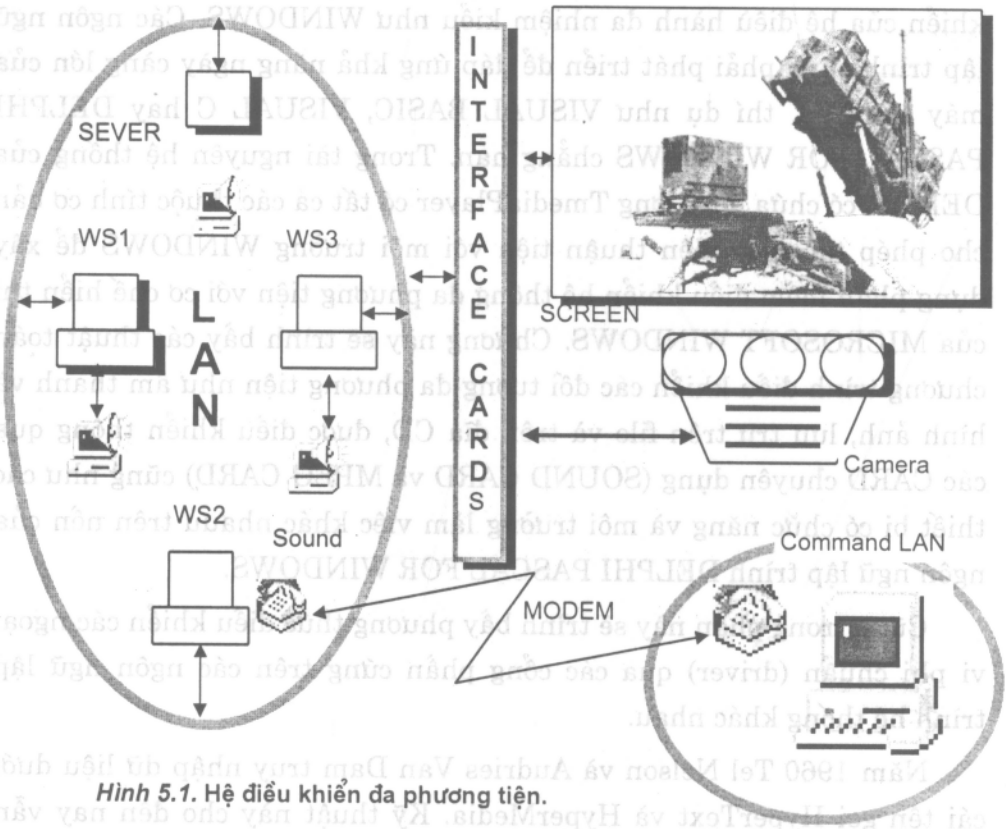
CHƯƠNG 5
LẬP TRÌNH ĐIỀU KHIỂN HỆ ĐA PHƯƠNG TIỆN

Các hệ đa phương tiện (multimedia systems) được đặt dưới sự điều khiển của hệ điêù hành đa nhiệm kiểu như WINDOWS. Các ngôn ngữ lập trình cũng phải phát triển để đáp ứng khả năng ngày càng lớn của máy tính PC, thí dụ như VISUAL BASIC, VISUAL C hay DELPHI PASCAL FOR WINDOWS chẳng hạn. Trong tài nguyên hệ thống của DELPHI có chứa đối tượng TmediaPlayer có tất cả các thuộc tính cơ bản cho phép tạo giao diện thuận tiện với môi trường WINDOWS để xây dựng phần mềm điều khiển hệ thống đa phương tiện với cơ chế hiển thị của MICROSOFT WINDOWS. Chương này sẽ trình bầy các thuật toán chương trình điều khiển các đối tượng đa phương tiện như âm thanh và hình ảnh, lưu trữ trên file và trên đĩa CD, được điều khiển thông qua các CARD chuyên dụng (SOUND CARD và MPEG CARD) cũng như các thiết bị có chức năng và môi trường làm việc khác nhauu trên nền của ngôn ngữ lập trình DELPHI PASCAL FOR WINDOWS.

Cũng trong phần này sẽ trình bầy phương thức điều khiển các ngoại vi phi chuẩn (driver) qua các cổng phần cứng trên các ngôn ngữ lập trình hệ thống khác nhau.

Năm 1960 Tel Nelson và Audries Van Dam truy nhập dữ liệu dưới cái tên gọi HyperText và HyperMedia. Kỹ thuật này cho đến nay vẫn được giữ nguyên tên và được sử dụng rộng tãi trong kỹ thuật tạo Web trên Internet. Năm 1968, Engle Band đã đưa ra được hệ thống sử dụng HypeText trên máy tính với cái tên NLS. Bộ quốc phòng Mỹ thành lập tổ chức DARPA (US Degerence Advanced Research Project Agency) để nghiên cứu về công nghệ Multimedia. Năm 1978 phòng thí nghiệm

khổng lồ MIT Media Laboratory chuyên nghiên cứu vè công nghệ Multimedia được thành lập. Chỉ sau một thời gian ngắn được thành lập và hoạt động, nhận thức được tầm quan trọng và ý nghĩa xã hội của Multimedia, người ta đã đầu tư rất lớn cho phòng thí nghiệm này. Một loạt các Công ty, các hãng lớn đã cho ra đời phòng thí nghiệm về Multimedia như AT&T, BELL,Oliverty. Những nỗ lực không ngừng của các nhà khoa học đã gặt hái được nhiều kết quả có tính chất nền móng cho lĩnh vực Multimedia và những kết quả này đã nhanh chóng được triển khai ứng dụng trong lĩnh vực truyền hình, viễn thông.



Hình 5.1. **Hệ điều khiển đa phương tiện.**

Hệ điều khiển đa phương tiện chính là trung tâm điều khiển và xử lý với nhiệm vụ phối hợp các đối tượng có môi trường hoạt động khác nhau cùng làm việc trong chế độ đồng bộ tuyệt đối (hình 5.1). Do mức độ phức tạp của các quá trình xử lý và tính toán nên hạt nhân của hệ thường là mạng máy tính LAN hoặc WAN.

Khái niệm đồng bộ trong Multimedia là một trong những khái niệm rất quan trọng có tính chất cốt lõi của Multimedia. Đồng bộ các "sự kiện" đó là sự sắp xếp các "sự kiện" theo trật tự thời gian sao cho ở cùng một thời điểm, các "sự kiện" có cùng trật tự thời gian cùng được xuất hiện (thuật ngữ "sự kiện" ở đây muốn nói đến các thay đổi hay biến đổi của các đối tượng - Object). Các đối tượng được xem xét trong lĩnh vực Multimedia có thể là các thiết bị vật lý, cơ học và cũng có thể là các đối tượng trìu tượng được xem xét trong lĩnh vực Multimedia có thể là âm thanh, ánh sáng, mầu sắc... và thậm chí có thể là các vận động cơ học của các thiết bị.

Hệ đa phương tiện hẹp là hệ làm việc với driver chuẩn như CD Player, còn hệ đa phương tiện mở (open multimedia system) bao gồm các driver chuẩn và các driver phi chuẩn. Chúng ta sẽ lần lượt xét từng hệ.

5.1. DỮ LIỆU MULTIMEDIA

Thông thường thông tin được thể hiện ở dạng văn bản, các văn bản này được mã hóa và lưu giữ trên máy tính và khi đó chúng ta có dữ liệu dạng văn bản. Nếu chúng ta thu nhận được thông tin ở dạng khác như âm thanh, hình ảnh thì dữ liệu của nó chínhi là dữ liệu Multimedia. Một cách đơn giản ta coi dữ liệu Multimedia là các dữ liệu của các dạng thông tin âm thanh (voice), hình ảnh (Image) hoặc văn bản (text). Nói ngắn gọn dữ liệu Multimedia là dữ liệu các dạng thông tin khác nhau.

Khi nghiên cứu các dạng dữ liệu trên, người ta nhận ra rằng cần phải phân chia dữ liệu Multimedia nhỏ hơn nữa. Bởi vì các dạng dữ liệu như âm thanh, hình ảnh trong quá trình "vận động" theo thời gian có những tính chất rất khác nhau so với dạng tĩnh. Điều này đòi hỏi công nghệ xử lý rất khác nhau. Vì vậy trong lĩnh vực công nghệ Multimedia người ta quan tâm nghiên cứu sử dụng các dạng dữ liệu sau: văn bản (Text), âm thanh (Sound, Void), âm điệu (Audio), hình ảnh tĩnh (Image), video (hình ảnh động - motion Image), animation (hình ảnh sử dụng theo nguyên tắc chiếu phim).

Dựa vào đặc thù của công nghệ và đối tượng nghiên cứu, người ta thống nhất cách phân chia công nghệ Multimedia thành hai lĩnh vực chính sau:

- ***Lĩnh vực các hệ thống thông tin Multimedia (Multimedia Information Systems)***

Mô hình hệ thống thông tin Multimedia (Model Information Multimedia). Trong lĩnh vực này người ta giải quyết các vấn đề sau:

- Các cấu trúc logic của tài liệu Multimedia (Logical Structure of media Document - Web page).

- Các phương thức để soạn thảo (Edit, Browse) các tài liệu Multimedia.

- Các quá trình tạo ra thông tin Multimedia.

- Các dạng (Forms) các công cụ (Tools) phục vụ cho xây lắp dữ liệu Multimedia.

Mô hình dữ liệu Multimedia phân tán (Multimedia Distributed Processing Model). Trong lĩnh vực này người ta quan tâm đến các mục tiêu sau:

- Các ngôn ngữ lập trình thao tác trên dữ liệu là các tài liệu Multimedia: kết hợp các chức năng cần thiết với các khái niệm lập trình các khái niệm cho phép lập trình truy nhập vào dữ liệu lưu trú trên thiết bị ngoại vi Multimedia (Media device control).

- Các dữ liệu dạng Multimedia và các dịch vụ cần trao đổi dữ liệu Multimedia (Interchange).

- Quản trị các dịch vụ viễn thông ở mức cao.

- Các mô hình dữ liệu Hypermedia, các máy chủ (Server) đáp ứng dịch vụ Hypermedia (Hypermedia Engine).

- Các hệ điều hành mạng đáp ứng dịch vụ Multimedia thời gian thực hiện.

- ***Lĩnh vực các hệ thống viễn thông Multimedia (Multimedia Communication Systems)***

Mô hình các dịch vụ Multimedia trên mạng (Multiservice Network Multimedia Model). Lĩnh vực này quan tâm nghiên cứu các vấn đề sau:

- Mạng đa dịch vụ (Multiservice) trên các hệ thống dữ liệu Multimedia phân tán.

- Các giao thức đáp ứng việc giao lưu giữa các mạng khác nhau có quản lý dữ liệu Multimedia.

- Trao đổi dữ liệu Multimedia trên Internet.

Mô hình hệ thống Multimedia hội nghị (Multimedia Conferencing Model). Đối tượng quan tâm của lĩnh vực này là làm thế nào kết hợp máy tính với các hệ thống viễn thông hiện có, tạo nên một hệ thống mạng không được thiết kế trước có khả năng đáp ứng các cuộc hội thảo, hội nghị theo gian thực.

Cuối cùng là sự pha trộn kết hợp của các lĩnh vực trên để tạo ra một mô hình ứng dụng đáp ứng thực tế yêu cầu.

5.2. ĐIỀU KHIỂN HỆ ĐA PHƯƠNG TIỆN HẸP

5.2.1. Điều khiển file

- ***Sử dụng file âm thanh WAV***

Trong các file âm thanh kiểu MIDI (Musical Instrument Digital Interface), WAV(WaveForm)... thì các file WAV là đơn giản nhất vì nội dung của nó phản ánh chính xác cấu trúc bản nhạc mà nó ghi lại, còn các file MIDI lại chứa các câu lệnh dùng để điều khiển SOUND CARD chứ không phải với chính bản nhạc. Để điều khiển dạng file này chỉ cần sử dụng hàm PlaySound () có trong MMSystem Unit:

```
function PlaySound (pszSound:Pchar; hmod:HMODULE; fdwSound:
                                                    WORD): BOOL;stdcall;
```

Tham số pszSound là biến kiểu Pchar dùng để mô tả file hoặc làm con trỏ để trỏ tới file WAV chứa trong bộ nhớ. Tham số thứ hai hmod dùng để mô tả cơ chế điều khiển việc nạp file. Tham số thứ ba fdwSound- chính là các cờ điều khiển dùng để biểu diễn cách chơi của file âm thanh.

Để dừng file WAV ở bất cứ thời điểm nào chỉ cần gọi hàm PlaySound() với các tham số được gán bằng Nil hoặc 0:

```
PlaySound(Nil,0, 0); hoặc
PlaySound(Nil,0, snd_Purge);
```

- *Điều khiển file video*

Audio và Video Interface (AVI) là một khuôn dạng file thông dụng dùng để chuyển đổi chức năng giữa file âm thanh và file hình ảnh. Lưu ý rằng AVI file chỉ chạy trên cửa sổ riêng.

Nếu muốn hiển thị khung của file AVI trên cửa sổ trước khi cho nó chạy thì phải đặt thuộc tính FRAMES của TmediaPlayer lên 1 rồi gọi trình phương thức STEP(). Thuộc tính FRAMES thông báo cho TmediaPlayer biết rằng cần bao nhiêu khung phải chuyển khi trình Step() hoặc Back() được gọi. Trình Step() thao tác theo trình tự:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 if OpenDialog1.Execute then
 with MediaPlayer1 do begin
 Filename:= OpenDialog1.Filename;
 Open;
  Frames:=1;     // thuộc tính khung=1
  step;
  Notify:=True;  //cờ thông báo = có
 end;
end;
```

Thủ tục này sẽ kích hoạt khung hiện hành. Thuộc tính về kích thước ảnh DisplayRec có thể gán giá trị tùy ý để phần diện tích ảnh chiếm đúng như ý đồ của người lập trình. Thí dụ, nếu ta muốn AVI file chỉ nằm trong giới hạn của PANEL1 thì chỉ cần gán DisplayRec cho kích thước của Panel:

```
MediaPlayer1.DisplayRect:=Rect(0,0,Panel1.Width,Panel1.Height);
```

Sử dụng câu lệnh này cho thủ tục kích hoạt chức năng:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
  if OpenDialog1.Execute then begin
    MediaPlayer1.Filename:= OpenDialog1.Filename;
    MediaPlayer1.Open;
    MediaPlayer1.DisplayRect:=Rect(0, 0, Panel1.Width,
                                   Panel1.Height);
    end;
end;
```

Sử dụng các sự kiện (event) OnPostClick và OnNotify trong điều khiển chế độ VIDEO là việc làm cần thiết để tạo giao diện thuận lợi cho người sử dụng. Sự kiện OnPostClick giống như sự kiện OnClick, ngoại trừ trường hợp là OnClick có tác động ngay còn OnPostClick phải sau một số thao tác mới có tác dụng. Nếu ta ấn phím PLAY trong thời gian TmediaPlayer đang chạy thì OnClick phản ứng ngay, còn sự kiện OnPostClick chỉ phản ứng sau khi thiết bị MEDIA thực hiện thao tác chạy điều khiển.

Sự kiện OnNotify đặt là TRUE thì các thao tác điều khiển khác như BACK, CLOSE, EJECT, NEXT, OPEN, PAUSE, PAUSEONLY, PLAY, PREVIOUSE, RESUME, STEP, STOP... mới có tác dụng. Các tham số này được truyền bằng thủ tục:

```
procedure TForm1. MediaPlayer1Notify(Sender: TObject);
begin
  MessageDlg('Media control method executed',mtInformation,
             [mbOK], 0);
end;
```

Cần nhớ là sự kiện này phải được đặt là TRUE trong thủ tục điều khiển:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
    with MediaPlayer1 do begin
```

```
        Filename:= OpenDialog1.Filename;
        Open;
        DisplayRect:=Rect(0, 0, Panel1.Width, Panel1.Height);
        Notify:=True;
      end;
end;
```

Bây giờ ta có đủ điều kiện để xây dựng chương trình điều khiển WAV file và AVI file cho hệ Multimedia. Chương trinh nguồn có dạng như sau:

Modul thứ nhất là File.DPR có dạng:

```
program EasyMM;
uses
Forms,
Main in 'Main.pas' {Form1};
{$R *.RES}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Modul thứ hai có dạng:

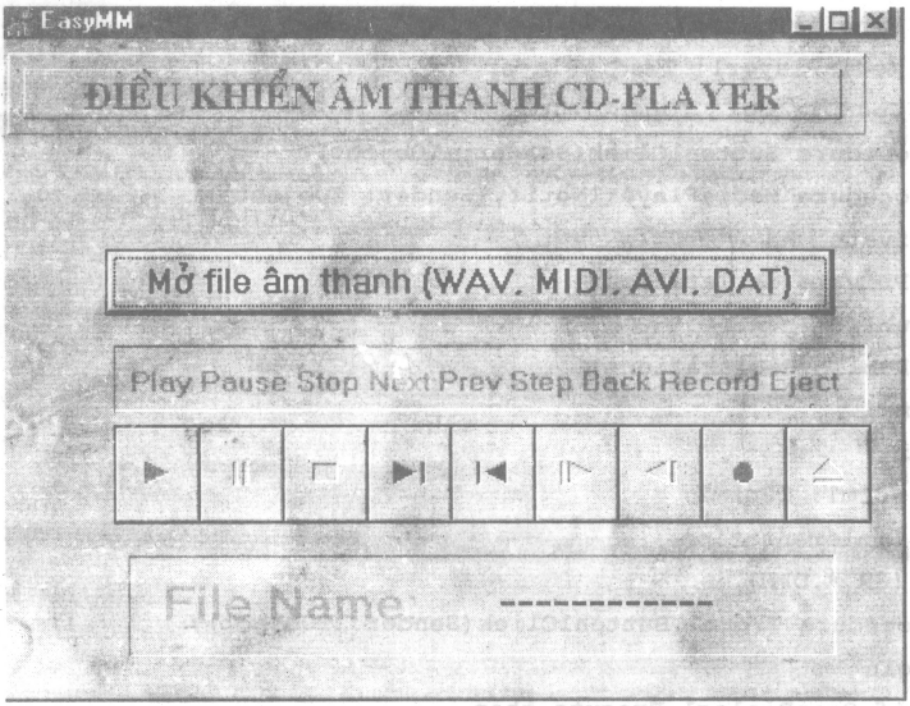
File unit.PAS có dạng:

```
unit Main;
interface
uses
Wintype, Winprocs, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs, MPlayer, StdCtrls;
type
  TForm1 = class(TForm)
  Button1: TButton;
```

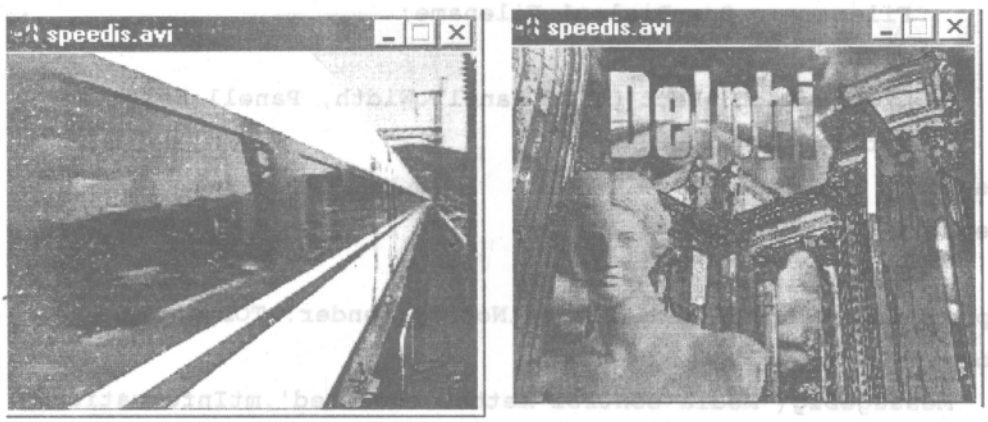
```
    Panel1:Tbutton;
    MediaPlayer1: TMediaPlayer;
    OpenDialog1: TOpenDialog;
procedure Button1Click(Sender: TObject);
procedure MediaPlayer1Notify(Sender: TObject);
private
 {Private declarations}
 public
 {Public declarations}
 end;
var
    Form1: TForm1;
    implementation
    {$R *.DFM}
procedure TForm1.Button1Click(Sender: TObject);
begin
    if OpenDialog1.Execute then
      with MediaPlayer1 do begin
      Filename:= OpenDialog1.Filename;
      Open;
      DisplayRect:=Rect(0, 0, Panel1.Width, Panel1.Height);
      Notify:=True;
end;
end;

procedure TForm1. MediaPlayer1Notify(Sender: TObject);
begin
    MessageDlg('Media control method executed',mtInformation,
                    [mbOK], 0);
end;
end.
```

Khi chương trình EasyMM chạy, một cửa sổ giao diện như hình 5.2 sẽ hiện ra. Nếu CDROM tồn tại thì thanh dụng cụ sẽ rõ nét. Trong trường hợp ngược lại, nó sẽ bị mờ. Trên hình 5.3 là cửa sổ của file AVI khi chạy.



Hình 5.2. Hộp thoại với nút mở file



Hình 5.3. Cửa sổ file AVI

5.2.2. Điều khiển CD player

Chỉ sau một vài năm thâm nhập vào lĩnh vực sản phẩm chất lượng cao (Hi-Fi) và với sự bùng nổ thị trường sau đó, công nghiệp đĩa laser

ngày nay đã tấn công thị trường PC. Phải công nhận rằng một ổ đĩa CD-ROM và một PC là một sự kết hợp thú vị, vì đĩa laer cung cấp cho PC một bộ nhớ khối, ngày nay mới chỉ là đọc được, nhưng nó lại thay được và cho phép lưu trữ trên rãnh duy nhất của nó hàng trăm thậm chí hàng ngàn MB dữ liệu, văn bản hay hình thành ảnh, âm thanh... CD-ROM đã mở ra một khả năng mới cho kỹ thuật điều khiển và công nghệ thông tin. Trong lĩnh vực Hi-Fi, ta đã thấy ở Hoa Kỳ và các nước phát triển khác, dưới dạng CD-ROM, các quyển sách với giá tương đối rẻ như danh bạ điện thoại, thư mục sách, sách cẩm nang hay cả bản dịch tiếng anh của báo Pravda (Báo sự thật Liên - xô cũ). Chúng ta cũng thấy dưới dạng số trên CD-ROM các bản đồ địa lý, thư viện ảnh, thư viện dữ liệu y học...

Tính ưu việt so với các sách in theo cách truyền thống là rõ ràng: các thông tin đã được số hóa có thể được xử lý lại trên máy tính. Các khả năng ở đây có thể nói là không hạn chế vì ta có thể dễ dàng tổ hợp các thông tin, đánh giá chúng kiểm tra chúng dưới các góc độ khác nhau bằng kỹ thuật xử lý phần mềm.

Cấu trúc phần cứng của PC hoàn toàn phù hợp để có thể ghép một ổ đĩa như vậy vào hệ thống, vấn đề còn lại lại là phần mềm, vì bản thân hệ điều hành chưa có chuẩn bị gì để đảm bảo cho một thiết bị loại này.

Hướng giải quyết chính của vấn đề này là làm sao, thông qua các driver và các phần mềm bổ sung nào để có thể dùng ổ đĩa CD-ROM như là một ổ đĩa mềm (tất nhiên là chỉ đọc). Như chúng ta đã chỉ ra ở các chương trước, các driver thiết bị thực hiện việc liên lạc giữa hệ điều hành và các thiết bị ngoài như màn hình, máy in và các ổ đĩa mềm, đĩa cứng. DOS phân biệt hai loại driver thiết bị, các driver khối và driver ký tự, ổ đĩa CD-ROM là một bộ nhớ khối, các thông tin được đọc ra theo từng khối, ta có thể cho rằng nó sẽ được điều khiển bởi một driver khối. Nhưng điều này lại không đúng, và ở đây bắt đầu xuất hiện các phức tạp, vì hệ điều hành yêu cầu các thiết bị khối một số điều kiện mà ổ đĩa CD-ROM không có.

Dung lượng của một CD-ROM không cho phép sát nhập nó thông qua một driver khối, vì rằng dung lượng của một thiết bị khối bị hạn chế (trước thế hệ 4.0 là 32 MB). Ngoài ra, CD-ROM không có File Allocation Table (bảng phân bố file) vì nó không thể ghi vào các sector và không có sector nào được dành cho FAT. Thay vào đó, CD-ROM có một bảng các file chứa địa chỉ bắt đầu của các thư mục con và các file.

Để viết một driver CD-ROM, có lẽ nên dùng đến driver ký tự vì đối với các driver này, hệ điều hành không đặt bất kỳ điều kiện gì về cấu trúc thiết bị. Nhưng mặt khác, các driver này sẽ gặp khó khăn trong việc liên lạc với ổ đĩa CD-ROM, vì chúng không chuyển số liệu theo khối mà theo từng byte và không mang ký hiệu thiết bị mà mang một tên file như "CON" hay 'NUL". Dù sao chăng nữa thì driver CD-ROM đối với hệ điều hành vẫn phải là một driver ký tự, để tránh việc đọc FAT vốn không tồn tại trong CD-ROM. Driver sẽ không mang tên thiết bị, mà lấy tên của mình trong file cấu hình CONFIG.SYS. Vì driver CD-ROM chủ yếu là do các hãng sản xuất CD-ROM cung cấp, nên nó mang tên theo kiểu SONY.SYS hay HITACH.SYS, và lời gọi nó trong file CONFIG.SYS được viết theo cách sau:

```
DEVICE = HITACHI.SYS / D: CDR1
```

Driver thiết bị chọn tên CDR1 như là tên của ổ đĩa CD-ROM.

Sau khi thủ tục khởi tạo đã cài đặt xong, driver mới thể hiện đúng mình là một driver khối, nó được bổ sung thêm một số hàm đặc biệt để đảm bảo hoạt động cho CD-ROM. Nhưng đối với hệ điều hành, nó vẫn là một driver ký tự và không thể kiểm tra thư mục của CD-ROM hoặc truy nhập đến các file.

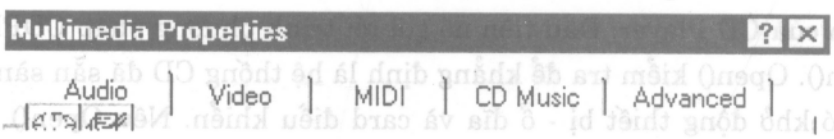
Để vượt qua trở ngại này, một chương trình nội trú (TSR) gọi là MSCDEX (Microsoft CD-ROM Extention) được cung cấp để đảm bảo hoạt động cho ổ đĩa CD-ROM. Chương trình này được gọi trong file Autoexec.bat. Trong dòng lệnh, tên của driver CD-ROM phải được truyền như là tham số cho chương trình:

```
MSCDEX /D: CDR1
```

Trước tiên, MSCDEX mở driver thông qua hàm 0pen của hệ điều hành, cung cấp cho nó một ký hiệu thiết bị. MSCDEX làm cho DOS tin rằng CD-ROM là một ổ đĩa ở xa trong mạng.

Các ổ đĩa của mạng được DOS xử lý như là file lớn, các file này có thể chứa đến hơn 32 MB. Ngoài ra, các thiết bị này không được DOS gọi một cách trực tiếp, nó không qua bộ đổi hướng. Phần nội chú của MSCDEX được cài trong bộ đổi hướng. Bộ đổi hướng này thông dịch mọi lời gọi. Nếu xuất hiện một lời gọi hướng tới ổ đĩa CD-ROM, nó không được truyền cho bộ đổi hướng theo cách truyền thống, mà mỗi lệnh được chuyển thành lời gọi các hàm của driver CD-ROM. Như vậy, mối liên hệ giữa DOS và ổ đĩa CD-ROM được thực hiện một cách hoàn hảo và sự truy nhập đến các thư mục con, các file có thể cho phép vào bất kỳ lúc nào.

Khi chương trình điều khiển CD Player chạy, các thực đơn chính được đặt trên thanh dụng cụ (hình 5.4). Đó là các thực đơn: Audio, Video, MIDI, Advanced. Mỗi khi cần một chức năng gì, chỉ cần ấn trỏ chuột vào thực đơn tương ứng. Khi ấn trỏ chuột vào thực đơn chính là ta khởi động chương trình con xử lý chức năng đó. Chúng ta sẽ lần lượt xây dựng các chương trình điều khiển này.



Hình 5.4. **Thực đơn điều khiển hoạt động của hệ đa phương tiện**

- ***Khởi tạo Audio cd player***

Khi ổ CD chay, nó sử dụng từng 2 giây một để nạp dữ liệu và có khi phải mất nhiều giây cho khởi tạo TmediaPlayer. Điều đó là do WINDOWS phải mất thời gian để nạp hệ điều hành cho Multimedia. Để tránh sự khó chịu khi phải chờ đợi như vậy, ta cài vào khoảng thời gian này chương trình SPLASH SCREEN (tạo pháo hoa màn hình).

```
unit Splash;
  interface
```

```
  uses   Wintype, Winprocs, Classes, Graphics, Controls,
                  Forms, StdCtrls,   ExtCtrls;
type
 TSplashScreen = class(TForm)
  StatusPanel1:=TPanel;
var
 SplashScreen:=TSplashScreen;
implementation
{$R *.DFM}
begin
  SplashScreen:=TSplashScreen.Create(Application);
  SplashScreen.Show;
  SplashScreen.Update;
end.
```

Khác với các form thông thường, SPLASH SCREEN được thiết lập và hiển thị trong khối khởi tạo của unit. Vì khối khởi tạo cho mọi unit được chạy trước khối chương trình chính trong DPR file, nên form này được thực hiện trước khi chương trình chính được thực hiện. Lưu ý rằng không được dùng Application.CreateForm() để thiết lập splash screen.

Để bắt đầu với CD Player ta cung cấp sự kiện điều khiển cho trình OnCreate. Trong trình này ta sẽ mở được và kích hoạt được chương trình của CD Player. Đầu tiên nó goi tới trình phương pháp CDPlayer's Open(). Open() kiểm tra để khẳng định là hệ thống CD đã sẵn sàng, sau đó nó khở động thiết bị - ổ đĩa và card điều khiển. Nếu Open() không thực hiện thành công nó sẽ thông báo lỗi kiểu EMCIDeviceEror. Ta có thể kết thúc bằng Application.Terminate:

```
  try;      {cố gắng mở}
     cdPlayer.open;  { mở}
  except    { không mở được thì...}
       on EMCIDeviceEror do begin
       MessageDlg('Error CD Player',mtError, [mbOK],0);
       Application.Terminate;
  end;
 end;
```

Sau khi CDPlayer hoạt động, ta có thể lập thuộc tính EnableButtons để đảm bảo là các button thích hợp được cho phép. Để thực hiện việc kiểm tra trạng thái của thiết bị CD chúng ta cần kiểm tra thuộc tính CDPlayer's Mode:

```
case CDPlayer.Mode of
mpPlaying:CDPlayer.EnabledButtons:=[btPause, btStop, btNext,
btPrev];
mpStopped,
mpPaused:CDPlayer.EnabledButtons:=[btPlay, btNext, btPrev];
end;
```

Thủ tục sau là mã nguồn của trình phương thức TCDPlayerForm. FormCreate(). Chú ý là ở thủ tục này thực hiện lời gọi tới hàng loạt các trình phương thức sau khi mở thành công CDPlayer. Mục đích của trình này là cập nhật (update) các thông số khác nhau của CD như số lượng TRACK đang có hiện hành.

```
procedure TCDPlayerForm.FormCreate(Sender: TObject);
     {thủ tục này được gọi khi form được khơi tạo.
     Nó mở và khơi tạo player}
var
 buf: array[0..15] of char;//tạo buffer đệm kiểu char kích
thước 16 byte
begin
 try
 CDPlayer.Open; //    mở CDPlayer
 {nếu CD đã chạy thì chỉ ra trạng thái chơi}
 if CDPlayer.Mode = mpPlaying then
 StatusLabel.Caption:= 'Playing';
 GetCDTotals; // lấy thời gian toàn bộ
        và số lượng tracks trên CD
 ShowTrackNumber; // chỉ ra track hiện hành
 ShowTrackTime; // thời gian (phút, giây) của track hiện hành
 ShowCurrentTime;
```

```
 ShowPlayerStatus; // cập nhật trạng thái
 except
 {nếu có lỗi khi khởi tạo thì thông báo}
 on EMCIDeviceError do begin
 MessageDlg('Error Initializing CD Player. Program will now
exit.',
 mtError, [mbOk], 0);
 Application.Terminate;
 end;
 end;
 {kiểm tra mode của CDplayer và kích hoạt các phím tương
ứng.}
 case CDPlayer.Mode of
 mpPlaying: CDPlayer.EnabledButtons:= [btPause, btStop,
btNext, btPrev];
 mpStopped,
 mpPaused: CDPlayer.EnabledButtons:= [btPlay, btNext,
btPrev];
 end;
 SplashScreen.Release; // đóng và giải phóng màn hình
end;
```

Lưu ý là dòng lệnh cuối cùng sẽ đóng form của thủ tục SPLASH SCREEN.

- *Cập nhật (Update) thông tin cho CDPlayer*

Khi cần cập nhật thông tin trên form CD phải sử dụng component TTimer. Mỗi khi có sự kiện nào xẩy ra ta có thể kích hoạt trình phương thức cập nhật. Nhấp chuột 2 lần Timer1 sẽ truy nhập tới sự kiện OnTimer. Mã nguồn sau sử dụng cho sự kiện đó:

```
procedure TCDPlayerForm.UpdateTimerTimer(Sender: TObject);
  {đây là trình phương thức của CD Player. Nó cập nhật các
   thông tin tại mọi khoảng thời gian.}
var
```

```
 CurTrack: Byte;
begin
 if CDPlayer.EnabledButtons = [btPause, btStop, btNext,
btPrev] then begin
 CDPlayer.TimeFormat:= tfMSF;
 DiskDoneGauge.Progress:= (mci_msf_minute(CDPlayer.Position)
*
          60 + mci_msf_second(CDPlayer.Position));
 CDPlayer.TimeFormat:= tfTMSF;
 ShowTrackNumber;  // chỉ số hiệu track
 ShowTrackTime;   //thời gian toàn bộ cho track
 ShowCurrentTime;  //chỉ ra thời gian hiện hành
 end;
end;
```

Lưu ý là thuộc tính TimeFormat có các dạng: tfHMS (giờ, phút, giây), tfMilliseconds (ms)...còn chuyển đổi thời gian thì WINDOWS API cung cấp các dịch vụ cho phép lấy thông tin thời gian từ bất kỳ dạng nào: mci_hms_Hour() (giờ) làm việc với tfHMS, mci_hms_Minute() (phút) làm việc với tfHMS, mci_hms_Second() (giây), mci_TMSF_Track() (rãnh của đĩa CD) làm việc với tfTMSF...

- ***Các trình phương thức để cập nhật (Update) cho CDPlayer***

Mục đích đầu tiên của các trình phương thức sử dụng là cập nhật cho nhãn nằm phía trên form CDplayer và cập nhật giá trị trên nó. Trình phương thức GetCDTotals() dùng để nhận giá trị về độ dài và số lượng TRACK của đĩa CD hiện hành. DiskDoneGauge dùng các thông tin này để cập nhật một loạt nhãn. AdjustSpeedButtons() cho phép điều chỉnh tốc độ CD Track:

```
procedure TCDPlayerForm.GetCDTotals;
{Trình phương thức này tìm thời gian và track và cho CD
Player chạy.}
var
```

```
 TimeValue: longint;
begin
 CDPlayer.TimeFormat:= tfTMSF; // đặt khuôn thời gian
 TimeValue:= CDPlayer.Length; // độ dài CD
 TotalTracks:= mci_Tmsf_Track(CDPlayer.Tracks);
                                              // tổng số track
 TotalLengthM:= mci_msf_Minute(TimeValue); // thời gian(phút)
 TotalLengthS:= mci_msf_Second(TimeValue); // thời gian(giây)
 {đặt caption choTotal Tracks label}
 TotTrkLabel.Caption:= TrackNumToString(TotalTracks);
 {đặt caption cho Total Time label}
 TotalLenLabel.Caption:= IntToStr(TotalLengthM) + 'm' + ' ' +
 IntToSTr(TotalLengthS) + 's';
 DiskDoneGauge.MaxValue:= (TotalLengthM * 60) + TotalLengthS;
 {đặt giá trị đúng cho speedbutton}
 AdjustSpeedButtons;
end;
```

Trình phương thức ShowCurrentTime() dùng để nhận thông tin về số phút và số giây của rãnh đang chơi và cập nhật các điều khiển cần thiết:

```
procedure TCDPlayerForm.ShowCurrentTime;
{hiển thị thời gian của rãnh đang chơi}
begin
 {phút}
 m:= mci_Tmsf_Minute(CDPlayer.Position);
 {giây}
 s:= mci_Tmsf_Second(CDPlayer.Position);
 {cập nhật cho nhãn track time}
 TrackTimeLabel.Caption:= IntToStr(m)+ 'm' + ' ' +
IntToStr(s) + 's';
 {cập nhật cho track gauge}
 TrackDoneGauge.Progress:= (60 * m) + s;
end;
```

ShowtrackTime() dùng để thu được thời gian tổng thể tính theo phút và giây của track hiện hành và cập nhận nhãn điều khiển:

```
procedure TCDPlayerForm.ShowTrackTime;
{Hiển thị thời gian tổng của track được chọn.}
var
 Min, Sec: Byte;
begin
 {không cập nhật thông tin nếu player vẫn đang ở track cũ}
 if CurrentTrack <> OldTrack then begin
 Min:= mci_msf_Minute(CDPlayer.TrackLength[mci_Tmsf_Track
 (CDPlayer.Position)]);
 Sec:= mci_msf_Second(CDPlayer.TrackLength[mci_Tmsf_Track
 (CDPlayer.Position)]);
 TrackDoneGauge.MaxValue:= (60 * Min) + Sec;
 TrackLenLabel.Caption:= IntToStr(Min) + 'm' + ' ' +
IntToStr(Sec) + 's';
 end;
 OldTrack:= CurrentTrack;
end;
```

- *Xây dựng chương trình điều khiển CDPlayer*

Bây giờ chúng ta có đủ điều kiện để xây dựng chương trình tổng thể cho điều khiển CDPlayer. Đầu tiên là trình điều phối.DPR có dạng:

```
program Mplyr;
uses
 Forms,
 Splash in 'Splash.pas'          {SplashScreen},
 Mplymain in 'Mplymain.pas' {CDPlayerForm};
begin
 Application.CreateForm(TCDPlayerForm, CDPlayerForm);
 Application.Run;
end.
```

Tiếp theo là module chương trình chính.PAS:

```
unit Mplymain;
interface
uses
 WinTypes, WinProcs, Classes, Graphics, Forms,
 Controls, MPlayer, StdCtrls, Menus, MMSystem,
 Gauges, SysUtils, Messages, Buttons, Dialogs,
 ExtCtrls, Splash;
type
 TCDPlayerForm = class(TForm)
 UpdateTimer: TTimer;
 MainScreenPanel: TPanel;
 StatusLabel: TLabel;
 Label2: TLabel;
 CurTrkLabel: TLabel;
 Label4: TLabel;
 TrackTimeLabel: TLabel;
 Label7: TLabel;
 Label8: TLabel;
 TotTrkLabel: TLabel;
 TotalLenLabel: TLabel;
 Label12: TLabel;
 TrackLenLabel: TLabel;
 Label15: TLabel;
 CDInfo: TPanel;
 Panel2: TPanel;
 Panel1: TPanel;
 CDPlayer: TMediaPlayer;
 SpeedButton1: TSpeedButton;
 SpeedButton2: TSpeedButton;
 SpeedButton3: TSpeedButton;
 SpeedButton4: TSpeedButton;
 SpeedButton5: TSpeedButton;
```

```
  SpeedButton6: TSpeedButton;
  SpeedButton7: TSpeedButton;
  SpeedButton8: TSpeedButton;
  SpeedButton9: TSpeedButton;
  SpeedButton10: TSpeedButton;
  SpeedButton11: TSpeedButton;
  SpeedButton12: TSpeedButton;
  SpeedButton13: TSpeedButton;
  SpeedButton14: TSpeedButton;
  SpeedButton15: TSpeedButton;
  SpeedButton16: TSpeedButton;
  SpeedButton17: TSpeedButton;
  SpeedButton18: TSpeedButton;
  SpeedButton19: TSpeedButton;
  SpeedButton20: TSpeedButton;
  TrackDoneGauge: TGauge;
  DiskDoneGauge: TGauge;
  Label1: TLabel;
  Label3: TLabel;
  procedure UpdateTimerTimer(Sender: TObject);
  procedure CDPlayerPostClick(Sender: TObject; Button:
TMPBtnType);
  procedure FormCreate(Sender: TObject);
  procedure SpeedButton1Click(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action:
TCloseAction);
  private
  {khai báo cục bộ}
  OldTrack, CurrentTrack: Byte;
  TestStatus: Boolean;
  m, s: Byte;
  TotalTracks: Byte;
  TotalLengthM: Byte;
```

```
 TotalLengthS: Byte;
 procedure GetCDTotals;
 procedure ShowTrackNumber;
 procedure ShowTrackTime;
 procedure ShowCurrentTime;
 procedure ShowPlayerStatus;
 procedure AdjustSpeedButtons;
 procedure HighlightTrackButton;
function TrackNumToString(InNum: Byte): String;
 public
 {các thủ tục, hàm dùng chung}
 end;
var
 CDPlayerForm: TCDPlayerForm;
implementation
{$R *.DFM}
const
 {mảng xâu ký tự biểu diễn các số từ 1--->20}
 NumStrings: array[1..20] of String[10] = ('One', 'Two',
'Three', 'Four', 'Five', 'Six', 'Seven', 'Eight',
'Nine','Ten', 'Eleven', 'Twelve', 'Thirteen', 'Fourteen',
'Fifteen', 'Sixteen', 'Seventeen', 'Eighteen', 'Nineteen',
'Twenty');
function TCDPlayerForm.TrackNumToString(InNum: Byte): String;
{dổi integer----> string (1--> 20).}
begin
 if (InNum > High(NumStrings)) or (InNum < Low(NumStrings))
then
 Result:= IntToStr(InNum) {nếu nằm ngoài mảng thì trả về
number}
 else
 Result:= NumStrings[InNum]; {trả về string từ NumString}
end;
```

```
procedure TCDPlayerForm.AdjustSpeedButtons;
{hiệu chỉnh phím nóng}
var
 i: integer;
begin
 {duyệt các thành phần mảng Component}
 for i:= 0 to ComponentCount - 1 do
 if Components[i] is TSpeedButton then begin //đúng là phím
nóng?
 if TSpeedButton(Components[i]).Tag <= TotalTracks then
 TSpeedButton(Components[i]).Enabled:= True
 else
 TSpeedButton(Components[i]).Enabled:= False;
 end;
end;
procedure TCDPlayerForm.GetCDTotals;
{lấy giá trị thời gian và số lượng track rồi cho CD chạy}
var
 TimeValue: longint;
begin
 CDPlayer.TimeFormat:= tfTMSF; // đặt khung thời gian
 TimeValue:= CDPlayer.Length; // độ dài
 TotalTracks:= mci_Tmsf_Track(CDPlayer.Tracks); // số lượng
track
 TotalLengthM:= mci_msf_Minute(TimeValue); // thời gian(phút)
 TotalLengthS:= mci_msf_Second(TimeValue); // thời gian(giây)
 {đặt caption cho label track}
 TotTrkLabel.Caption:= TrackNumToString(TotalTracks);
 {đặt caption cho label Thời gian}
 TotalLenLabel.Caption:= IntToStr(TotalLengthM) + 'm' + ' ' +
 IntToSTr(TotalLengthS) + 's';
 DiskDoneGauge.MaxValue:= (TotalLengthM * 60) + TotalLengthS;
 {đặt giá trị đúng cho speedbutton}
```

```
 AdjustSpeedButtons;
end;

procedure TCDPlayerForm.ShowPlayerStatus;
{hiển thị trạng thái của Player.}
begin
 if CDPlayer.EnabledButtons = [btPause, btStop, btNext,
btPrev] then
 with StatusLabel do begin
 case CDPlayer.Mode of
 mpNotReady: Caption:= 'Not Ready';
 mpStopped: Caption:= 'Stopped';
 mpSeeking: Caption:= 'Seeking';
 mpPaused: Caption:= 'Paused';
 mpPlaying: Caption:= 'Playing';
 end;
 end
 {nếu các nút này đang được kích hoạt thi CD Player phải
dừng}
 else if CDPlayer.EnabledButtons = [btPlay, btNext, btPrev]
then
 StatusLabel.Caption:= 'Stopped';
end;

procedure TCDPlayerForm.ShowCurrentTime;
{hiển thị thời gian của track hiện hành}
begin
 {thời gian tính theo phút}
 m:= mci_Tmsf_Minute(CDPlayer.Position);
 {thời gian tính theo giây}
 s:= mci_Tmsf_Second(CDPlayer.Position);
 {cập nhật thời gian cho label}
```

```
 TrackTimeLabel.Caption:= IntToStr(m)+ 'm' + ' ' +
IntToStr(s) + 's';
 TrackDoneGauge.Progress:= (60 * m) + s;
end;
procedure TCDPlayerForm.ShowTrackTime;
{chuyển sang hiển thị thời gian tổng thể của track được
chọn.}
var
 Min, Sec: Byte;
begin
 {không cập nhật thông tin nếu player còn trên cùng một
track}
 if CurrentTrack <> OldTrack then begin
Min:= mci_msf_Minute(CDPlayer.TrackLength[mci_Tmsf_Track
 (CDPlayer.Position)]);
Sec:= mci_msf_Second(CDPlayer.TrackLength[mci_Tmsf_Track
 (CDPlayer.Position)]);
TrackDoneGauge.MaxValue:= (60 * Min) + Sec;
TrackLenLabel.Caption:= IntToStr(Min) + 'm' + ' ' +
IntToStr(Sec) + 's';
end;
OldTrack:= CurrentTrack;
end;

procedure TCDPlayerForm.HighlightTrackButton;
{tô mầu đỏ cho nút hiện hành trong khi các nút khác tô mầu
xanh}
var
 i: longint;
begin
 {duyệt các component của form}
 for i:= 0 to ComponentCount - 1 do
 {có phải là nút?}
```

```
 if Components[i] is TSpeedButton then
 if TSpeedButton(Components[i]).Tag = CurrentTrack then
 {đổi màu đỏ nếu là tract hiện hành}
 TSpeedButton(Components[i]).Font.Color:= clRed
 else
 {đổi màu xanh nếu không phải là tract hiện hành}
 TSpeedButton(Components[i]).Font.Color:= clNavy;
end;

procedure TCDPlayerForm.ShowTrackNumber;
{hiển thị số hiệu track hiện hành.}
var
 t: byte;
begin
 t:= mci_Tmsf_Track(CDPlayer.Position); // lấy track hiện
hành
 CurrentTrack:= t;
 CurTrkLabel.Caption:= TrackNumToString(t); // gán nhãn
 HighlightTrackButton; // làm sáng nút hiện hành.
end;

procedure TCDPlayerForm.UpdateTimerTimer(Sender: TObject);
{Đây là thủ tục quan trọng. Nó cập nhật thông tin
 trên mỗi khoảng thời gian.}
var
 CurTrack: Byte;
begin
 if CDPlayer.EnabledButtons = [btPause, btStop, btNext,
btPrev] then begin
CDPlayer.TimeFormat:= tfMSF;
DiskDoneGauge.Progress:= (mci_msf_minute(CDPlayer.Position) *
60 + mci_msf_second(CDPlayer.Position));
 CDPlayer.TimeFormat:= tfTMSF;
```

```
ShowTrackNumber;
 ShowTrackTime;
 ShowCurrentTime;
 end;
end;

procedure TCDPlayerForm.CDPlayerPostClick(Sender: TObject;
                  Button: TMPBtnType);
{hiển thị trạng thái nút ấn khi một trong các nút đó bị kích
hoạt}
begin
 Case Button of
 btPlay: begin
 CDPlayer.EnabledButtons:= [btPause, btStop, btNext, btPrev];
 StatusLabel.Caption:= 'Playing';
 end;
 btPause: begin
 CDPlayer.EnabledButtons:= [btPlay, btNext, btPrev];
 StatusLabel.Caption:= 'Paused';
 end;
 btStop: begin
 CDPlayer.Rewind;
 CDPlayer.EnabledButtons:= [btPlay, btNext, btPrev];
 CurTrkLabel.Caption:= 'One';
 TrackTimeLabel.Caption:= '0m 0s';
 TrackDoneGauge.Progress:= 0;
 DiskDoneGauge.Progress:= 0;
 StatusLabel.Caption:= 'Stopped';
 end;
 btPrev, btNext: begin
 CDPlayer.Play;
 CDPlayer.EnabledButtons:= [btPause, btStop, btNext, btPrev];
 StatusLabel.Caption:= 'Playing';
```

```
 end;
 end;
end;

procedure TCDPlayerForm.FormCreate(Sender: TObject);
       {thủ tục này được gọi khi form đã được tạo ra.
        Nó mở và khởi tạo CD player}
var
 buf: array[0..15] of char;
begin
 try
 CDPlayer.Open; // mở CD player Open.
   {nếu khi khởi động mà CD player đã chạy
   thì hiển thị trạng thái của nó.}
 if CDPlayer.Mode = mpPlaying then
 StatusLabel.Caption:= 'Playing';
 GetCDTotals; // thời gian và số lượng track của CD player
 ShowTrackNumber; // track hiện hành
 ShowTrackTime; // phút, giây cho track hiện hành
 ShowCurrentTime; // vị trí hiện hành của CD player
 ShowPlayerStatus; // cập nhật trạng thái CD Player
 except
 {nếu có lỗi thì thông báo và xử lý lỗi.}
 on EMCIDeviceError do begin
 MessageDlg('Error Initializing CD Player. Program will now
exit.',
 mtError, [mbOk], 0);
 Application.Terminate;
 end;
 end;
      {kiểm tra mode của CD-ROM và tích cực hoá
      nút điều khiển tương ứng.}
 case CDPlayer.Mode of
```

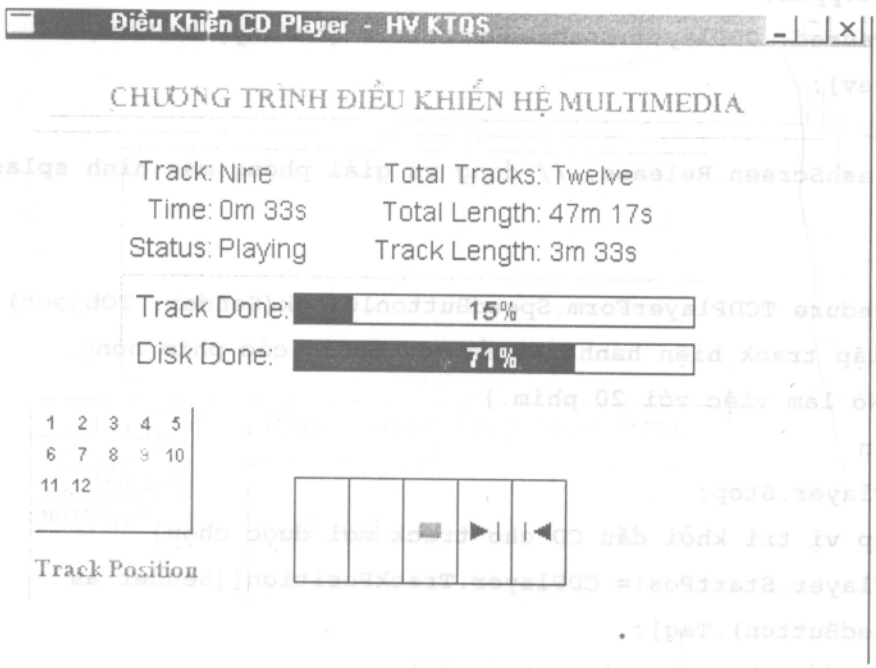
```
 mpPlaying: CDPlayer.EnabledButtons:= [btPause, btStop,
btNext, btPrev];
 mpStopped,
 mpPaused: CDPlayer.EnabledButtons:= [btPlay, btNext,
btPrev];
 end;
 SplashScreen.Release; // đóng và giải phóng màn hình splash
end;

procedure TCDPlayerForm.SpeedButton1Click(Sender: TObject);
   {lập track hiện hành khi ấn một trong các phím nóng.
    Nó làm việc với 20 phím.}
begin
 CDPlayer.Stop;
 {lập vị trí khởi đầu CD cho track mới được chọn}
 CDPlayer.StartPos:= CDPlayer.TrackPosition[(Sender as
TSpeedButton).Tag];
 {bắt đầu chạy CD tại vị trí mới}
 CDPlayer.Play;
 CDPlayer.EnabledButtons:= [btPause, btStop, btNext, btPrev];
 StatusLabel.Caption:= 'Playing';
end;

procedure TCDPlayerForm.FormClose(Sender: TObject;
             var Action: TCloseAction);
begin
 CDPlayer.Close;
end;
end.
```

Khi chương trình này chạy, một cửa sổ thông báo sẽ hiện ra có dạng như hình 5.5. Như vậy với cách nắm nguyên tắc làm việc của TMediaPlayer chúng ta có thể xây dựng các trình điều khiển hệ thống

MULTIMEDIA với các đối tượng WAVE, AVI và đĩa âm thanh CD một cách thuận tiện.



Hình 5.5. **Hộp thoại của chương trình Mplyr.**

5.3. ĐIỀU KHIỂN HỆ ĐA PHƯƠNG TIỆN MỞ

Hệ đa phương tiện mở bao gồm các thành phần chuẩn và các thành phần phi chuẩn. Chương trình điều khiển hệ thống đa phương tiện MULTIMEDIA với các đối tượng chuẩn là CD Player đã được xét ở phần trên. Trong phần này sẽ trình bầy khả năng điều khiển các đối tượng phi chuẩn bất kỳ thông qua các cổng IO bằng các ngôn ngữ lập trình khác nhau.

Khi xây dựng một phần mềm làm việc với các ứng dụng điều khiển các thiết bị phần cứng thông qua các cổng ngoại vi cũng như các cổng phần cứng của máy tính chúng ta thường gặp phải 2 vấn đề chính:

Tổ chức việc truy nhập trực tiếp tới các cổng phần cứng, nghĩa là khả năng đọc/ghi trực tiếp các giá trị 1 hoặc 2 byte từ các cổng phần

cứng trong môi trường các hệ điều hành khác nhau như DOS, WINDOWS...

Quản lý việc truy nhập tới các cổng phần cứng đảm bảo chế độ thời gian thực. Thông thường việc quản lý truy nhập trực tiếp các cổng có thể được thực hiện thông qua ba chế độ:

Bám liên tục, nghĩa là việc đọc/ ghi các cổng phần cứng liên tục theo thời gian. Để thực hiện chế độ này, người lập trình có thể thể hiện trong một vòng lặp của chương trình hoặc thực hiện thông qua thủ tục phục vụ của ngắt thời gian với khoản thời gian giữa hai lần kích hoạt ngắt rất nhỏ (có thể coi như không đáng kể).

Chế độ truy nhập theo thời gian định trước. Có nghĩa là việc đọc/ghi các cổng phần cứng được thực hiện theo một nhịp thời gian đều đặn với khoản thời gian giữa các lần đọc/ghi được định trước. Người lập trình có thể thực hiện chế độ truy nhập này bằng thủ tục phục vụ ngắt thời gian.

Chế độ truy nhập ngẫu nhiên (do tác động của một ngắt cứng hoặc tín hiệu trạng thái). Có nghĩa là việc đọc/ghi các cổng phần cứng (ngắt phát sinh do các thiết bị phần cứng của máy tính) hoàn toàn ngẫu nhiên khi có tác động của một ngắt cứng hoặc có xác nhận cuả các tín hiệu trạng thái. Người lập trình có thể thực hiện chế độ truy nhập này thông qua thủ tục phục vụ ngắt của các ngắt cứng tương ứng đã được chọn trước làm tác động ngẫu nhiên.

Từ phân tích trên, chúng ta thấy rằng việc thực hiện đối với cả ba chế độ truy nhập các cổng phần cứng đều có thể thực hiện thông qua các thủ tục phục vụ ngắt của các ngắt cứng hoặc ngắt mềm. Do vậy, dưới đây chúng ta sẽ xem xét và phân tích kỹ hơn khả năng hỗ trợ của các ngôn ngữ lập trình đối với việc đọc/ghi trực tiếp các cổng phần cứng và việc sử dụng các ngắt trên các môi trường hệ điều hành khác nhau.

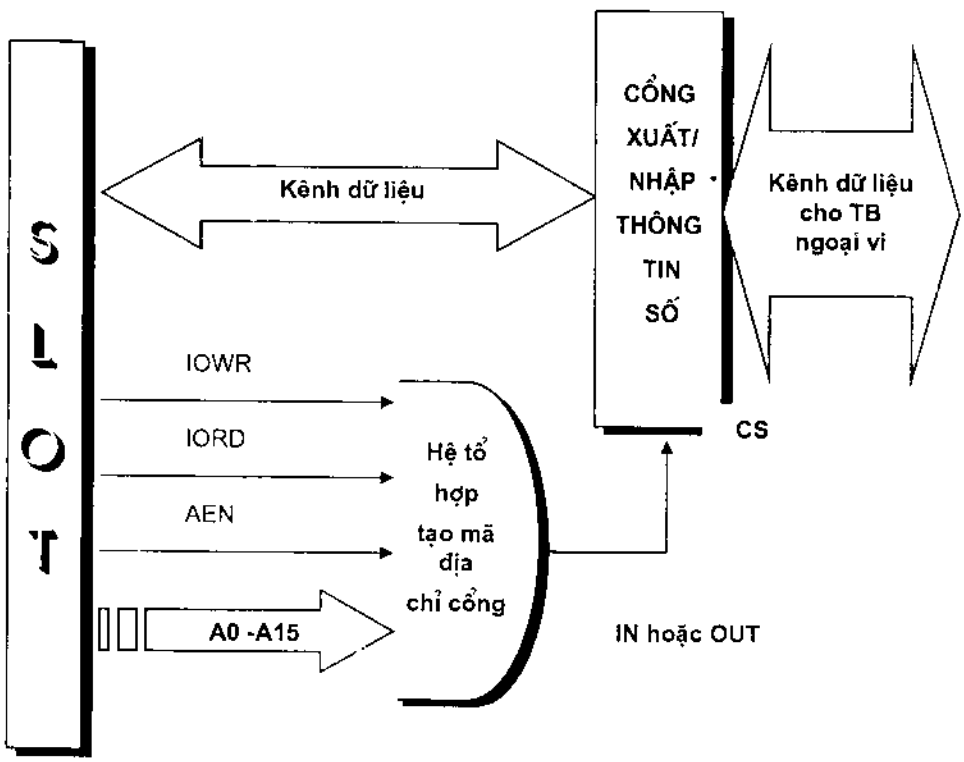
Để tổ chức xuất nhập thông tin số qua các cổng có điều khiển từ máy tính thì ngoài các cổng chuẩn được tổ chức sẵn như cổng COM, cổng máy in, PC cho phép tổ chức thêm các cổng có địa chỉ từ 300h đến 3FFh.

Nguyên tắc sử dụng SLOT để tổ chức một cổng được minh họa trên hình 5.6. Cổng xuất nhập thông tin số có thể là một bộ chốt thông

thường kiểu 74373 (hay 74374) hay chip thông minh kiểu 8255. Bộ tạo mã địa chỉ cho cổng được tổ chức trên cơ sở của hệ tổ hợp với các tín hiệu đầu vào:

Bảng 5.1. **Bảng địa chỉ tổ chức trên SLOT**

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Địa chỉ
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0300h
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0301h
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	03FFh



Hình 5.6. **N guyên tắc tổ chức một cổng xuất/nhập thông tin trên SLOT**

- *Nhóm tín hiệu điều khiển:* IOWR, IORD, AEN (đôi khi cả ALE) dùng để định hướng truyền thông tin;

- *Nhóm tín địa chỉ:* giá trị của kênh địa chỉ để xác định địa chỉ của cổng cần điều khiển. Do vùng địa chỉ cho các ứng dụng bị giới hạn từ 300h đến 3FFh nên chỉ các bit từ A0 đến A7 được quyền thay đổi còn các bit từ A8 đến A15 phải giữ không thay đổi. Bảng 5.1 là bảng địa chỉ tổ chức trên SLOT.

Trên hệ điều hành 16 bits (MS-DOS, MS WINDOWS 3.1,...) dùng cho các máy tính cá nhân thì hầu hết các ngôn ngữ lập trình bậc thấp cũng như bậc cao (ASSEMBLER, BASIC, PASCAL, C,...) đều hỗ trợ rất đầy đủ các thủ tục và hàm phục vụ cho việc truy nhập (đọc/ghi) trực tiếp các cổng phần cứng. Các lệnh đọc ghi trực tiếp ra cổng trên hệ điều hành MS-DOS (minh họa bằng các ngôn ngữ khác nhau):

Đọc một byte từ cổng:

Minh bằng assembler:

```
mov dx, portID; gán địa chỉ cổng portID cho dx
in al, dx; đọc nội dung cổng vào al
```

Minh họa bằng Pascal:

```
biến:= port[portID];
```

Trong đó: portid là số hiệu cổng.

Thí dụ: Giả sử có một biến kiểu byte là read_val;

Đọc một byte từ cổng 3f8h: read_val = port[$3f8];

Minh họa bằng ngôn ngữ C:

Đọc một byte từ cổng phần cứng:

```
int inp(int portid);
```

hoặc:

```
unsigned char inportb (portid);
```

Trong đó: portid là số hiệu cổng.

Thí dụ: Giả sử có một biến kiểu byte là read_val;

Đọc một byte từ cổng 3f8h: read_val = inp(3f8);

Đọc một từ (2 byte) từ cổng:

Minh họa bằng Pascal:

```
integer portw[portid: integer];
```

Trong đó: portid là số hiệu cổng.

Thí dụ: Giả sử có một biến kiểu 2 byte là read_val;

Đọc 2 byte từ cổng 3f8h: read_val = portw[$3f8];

Minh họa bằng ngôn ngữ C:

Đọc một từ (2 byte) từ cổng phần cứng:

```
int inport(int portid);
```

Trong đó: portid là số hiệu cổng.

Thí dụ: Giả sử có một biến kiểu 2 byte là read_val;

Đọc 2 byte từ cổng 3f8h: read_val = inport(3f8);

Xuất 1 byte ra cổng:

Minh bằng assembler:

```
mov dx, portID; gán địa chỉ cổng portID cho dx
out dx, al; dọc nội dung cổng vào al
```

Minh họa bằng Pascal:

```
port[portID]:= biến;
```

Thí dụ: Giả sử có một biến kiểu byte là write_val;

Phát một byte ra cổng 3f8h: port[$3f8]:= write_val;

Minh họa bằng C:

```
int outp(int portid, int byte_value);
```

hoặc:

```
void outportb (int portid, unsigned char byte_value);
```

Trong đó: portid là số hiệu cổng

byte_value là giá trị cần phát.

Thí dụ: Giả sử có một biến kiểu byte là write_val;

Phát một byte ra cổng 3f8h: outp(3f8, write_val);

Xuất 2 byte ra cổng:

Minh họa bằng Pascal:

```
portw [portid: integer]:= biến_2byte;
```

Trong đó: portid là số hiệu cổng

biến_2byte là giá trị cần phát (2 byte).

Thí dụ: Giả sử có một biến kiểu 2 byte là write_val;

Phát một từ (2 byte) ra cổng 3f8h:portw [$3f8]:= write_val;

Minh họa bằng C:

```
void outport (int portid, int value);
```

Trong đó: portid là số hiệu cổng

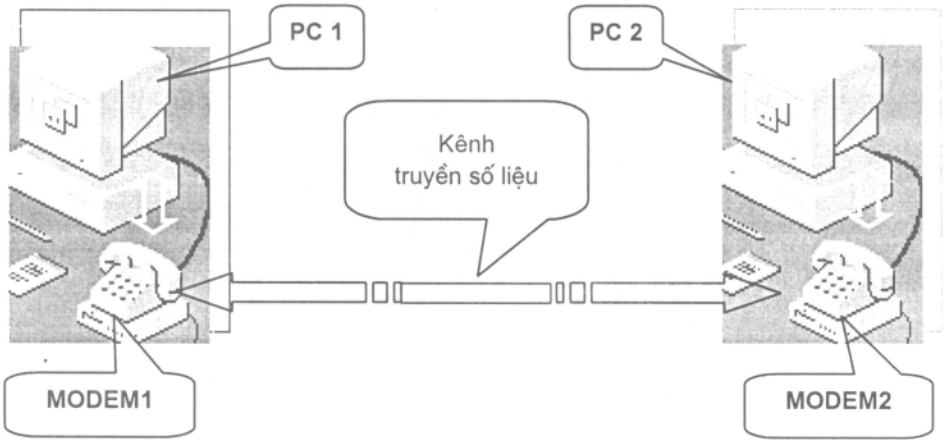
value là giá trị cần phát (2 byte).

Thí dụ: Giả sử có một biến kiểu 2 byte là write_val;

Phát một từ (2 byte) ra cổng 3f8h: outport (3f8, write_val);

CHƯƠNG 6
LẬP TRÌNH ĐIỀU KHIỂN HỆ TRUYỀN TIN MODEM

Trong chương này sẽ đề cập tới một hệ thống công nghệ viễn thông phổ dụng là hệ thống thu phát dữ liệu qua thiết bị MODEM với máy tính PC hay mạng máy tính thực hiện chức năng xử lý thông tin và điều khiển cơ chế truyền dẫn thông tin (hình 6.1). MODEM là thiết bị trung gian rất quan trọng vì nhờ nó mà máy tính có thể ghép nối và hòa nhập được với mạng truyền thông truyền thống cả bằng kỹ thuật vô tuyến và kỹ thuật hữu tuyến.



Hình 6.1. **Tổ chức chương trình truyền số liệu qua MODEM**

6.1. NGUYÊN TẮC LÀM VIỆC CỦA MODEM

Modem là thiết bị truyền tin quan trọng có nhiệm vụ điều chế (MOdulation) tín hiệu số từ máy tính thành các dạng tín hiệu truyền

thông tiêu chuẩn và giải điều chế (DEModulation) từ tín hiệu [illegible] thông trên đường truyền thành tín hiệu số đưa vào máy tính.. Nhờ Modem mà các thông tin số trong máy tính được truyền dẫn đi xa, hòa nhập vào mạng truyền tin truyền thống, tạo thành một cấu trúc thống nhất của công nghệ thông tin hiện đại. Để có thể sử dụng Modem, chúng ta bắt đầu từ nguyên tắc làm việc của nó.

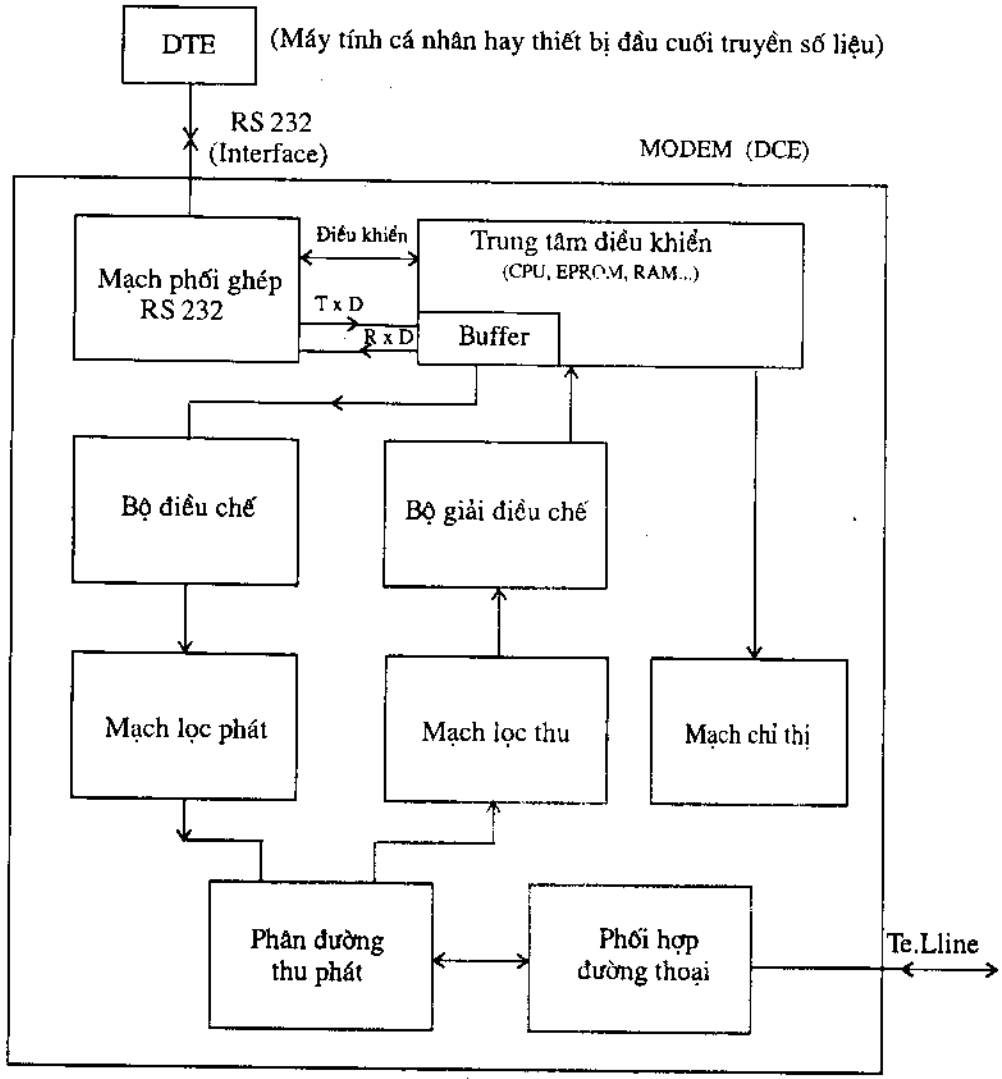
6.1.1. Sơ đồ chức năng của MODEM

- ***Mạch phối ghép RS 232***

MODEM với chức năng điều chế và giải điều chế tín hiệu được coi là thiết bị truyền số liệu (DCE - Data Communication Equipment) thường nối với thiết bị đầu cuối truyền số liệu (DTE - Data Terminal Equipment), ví dụ với máy tính phải thông qua giao diện RS232 là một chuẩn ghép nối rất cơ bản.

Chuẩn RS 232 quy ước phương pháp phép nối giữa đầu cuối số liệu và đầu cuối truyền dòng dữ liệu nhị phân. Chuẩn RS 232 gồm 25 đường tín hiệu (tuy nhiên máy tính chỉ cần một số đường là đủ hoạt động). Các đường tín hiệu đó là:

1. PGND (Protet Ground) - đất bảo vệ.
2. TxD (Transmited Data) - số liệu phát.
3. RxD (Received Data) - số liệu thu.
4. RTS (Request to Send) - yêu cầu gửi.
5. CTS (Clear to Send) - xóa để gửi.
6. DSR (Data Set Ready) - số liệu sẵn sàng.
7. SGND (Signal Ground) - đất của dây tín hiệu.
8. RLSD (Received - Line Signal Director) - bộ phát hiện tín hiệu thu đường dây.
9. (Reserved - for Data Set Tesning) - dành riêng cho thử dữ liệu.
11. Unassigned - không xác định.
12. SRLSD (Secondary Received Line Signal Detector) - bộ phát tín hiệu thu thứ cấp.



Hình 6.2. **Sơ đồ khối chức năng của MODEM**

13. SCTS - thứ cấp của 5.
14. STD - thứ cấp của 2.
15. TSET (Trasmission Signal Element Timing) - định thời phần tử tín hiệu phát
16. RSD - thứ cấp của 3.
17. RSET (Received Signal Element Timing) - định thời phần tử tín hiệu nhận.

18. Unassigned - không xác định.
19. SRTS - thứ cấp của 4.
20. DTR (Data Terminal Ready) - Thiết bi đầu cuối (PC) sẵn sàng.
21. SQD (Signal Quality Detector) -bộ phát hiện tín hiệu chất lượng.
22. RI (Ring Indicator) - chỉ dẫn báo chuông.
23. DSRS (Data Signal Rate Selector) - chọn tốc độ.
24. TSET (Transmit Signal Element Timing) - định thời cho phần tử của tín hiệu truyền.
25. (Unassignal) - không xác định.

Các tín hiệu điều khiển cơ bản của giao diện RS 232 giữa PC và MODEM được mô tả kỹ hơn trên hình 6.3 cả về chức năng lẫn hướng đi của tín hiệu. Cụ thể:

TxD (chân số 2), RxT (chân số 3): TxD là đường truyền số liệu từ máy tính ra modem (transmit data) còn RxT là đường thu số liệu từ modem vào máy tính (Receive Data). Dãy tín hiệu nhị phân được truyền nối tiếp, không đồng bộ (asyncronous), logic 1 ứng với mức điện thế - 12V và mức logic 0 ứng với mức điện thế +12V.

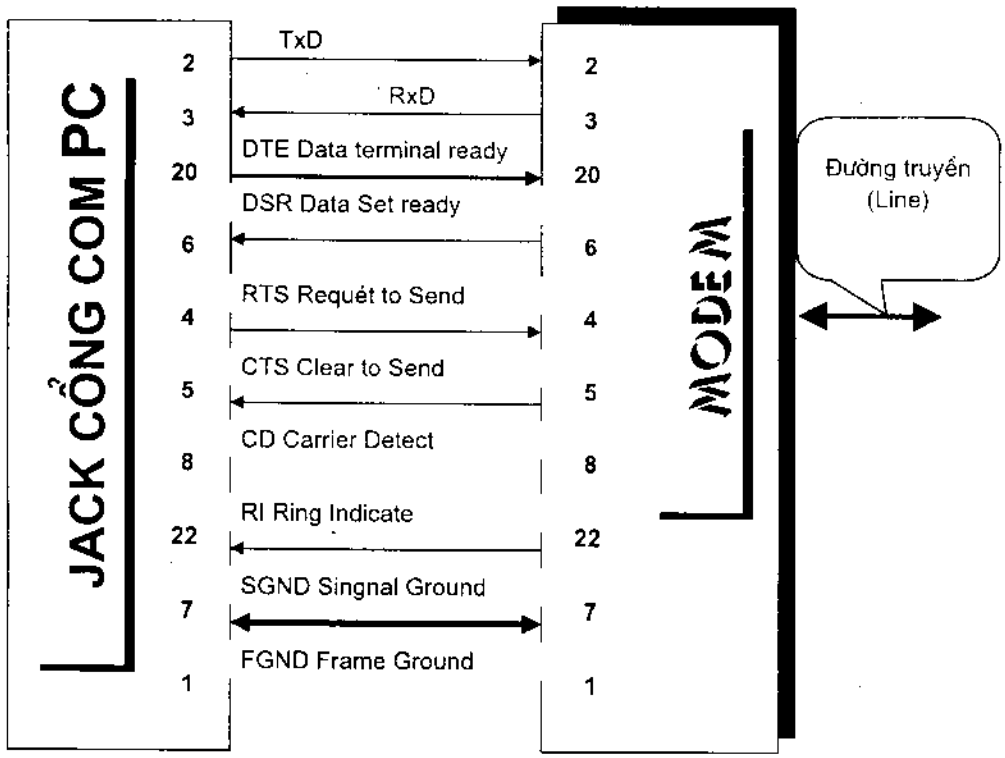
RTS (chân số 4) và CTS (chân số 5): RTS (Ready To Send) là tín hiệu máy tính báo cho modem biết có yêu cầu truyền số liệu. CTS (Clear to Send) Modem báo cho máy tính biết rằng nó đã sẵn sàng phát.

DSR (chân số 6). Modem báo cho máy tính biết rằng đã sẵn sàng làm việc

CD (chân số 8). Modem báo rằng đã phát hiện được sóng mang (đường truyền thiết lập).

RI (chân số 22). Modem báo có tín hiệu gọi tới (chuông).

Các tín hiệu trên giao diện RS232 có mức điện áp giữa - 15V và + 15V (tín hiệu lưỡng cực). Mức logic 1 ứng với điện áp từ - 5 V đến - 15 V, mức logic 0 ứng với điện áp từ + 5 V đến + 15 V. Việc sử dụng mức điện áp cao hơn mức TTL như thế đảm bảo công suất tín hiệu khi truy xuất các thiết bị ngoại vi ở khoảng cách tương đối xa và tăng khả năng chịu tải cho máy tính.



***Hình 6.3.* Chuẩn RS 232 C**

Khi PC ở trạng thái sẵn sàng nó thông báo bằng chân 20 là DTE=1 thì MODEM báo lại rằng đang ở trạng thái sẵn sàng bằng chân 6 là DSR=1. Lúc này MODEM sẽ kiểm tra đường dây (Line) xem đã có tín hiệu báo nhận từ xa chưa, nếu có nó sẽ báo cho PC biết bằng tín hiệu CD=1 tại chân 8. PC lập tức báo cho MODEM yêu cầu được gửi số liệu bằng tín hiệu RTS=1 tại chân 4. Thao tác cuối cùng là MODEM báo trạng thái sẵn sàng bằng tín hiệu CTS=1 tại chân 5.

Kết nối được thiết lập thông qua việc người sử dụng quay số gọi máy tính ở xa và đợi trả lời. Máy tính ở phía đó sẽ nghe được chuông rung và nếu không bận sẽ thực hiện kết nối. Chuông dừng và người sử dụng nghe được trả lời. Phía bên người sử dụng, nếu ấn nút DATA thì máy tính tại chỗ sẽ được nối lên đường dây (DTR=1) và MODEM tại chỗ trả lời sẵn sàng (DSR=1). Lúc này đèn báo kết nối đã được thiết lập sẽ bật sáng.

Khi có cuộc gọi, MODEM được gọi sẽ reo chuông (RI=1). Giả sử máy tính được gọi đã sẵn sàng (DTR=1) thì nó thông báo bằng cách đặt RTS=1. Điều này có hai hiệu ứng:

MODEM được gọi gửi tín hiệu sóng mang (CARRIER) đến MODEM gọi để báo cuộc gọi đã được chấp nhận.

Sau thời gian trễ MODEM được gọi đặt CTS=1 để máy tính có thể gửi số liệu. Máy tính gửi lời mời tới bên kia rồi chuẩn bị nhận trả lời của người sử dụng bằng cách đặt tín hiệu chưa sẵn sàng (RTS Off). Kết quả là chuông sẽ ngừng reo (Tone Off).

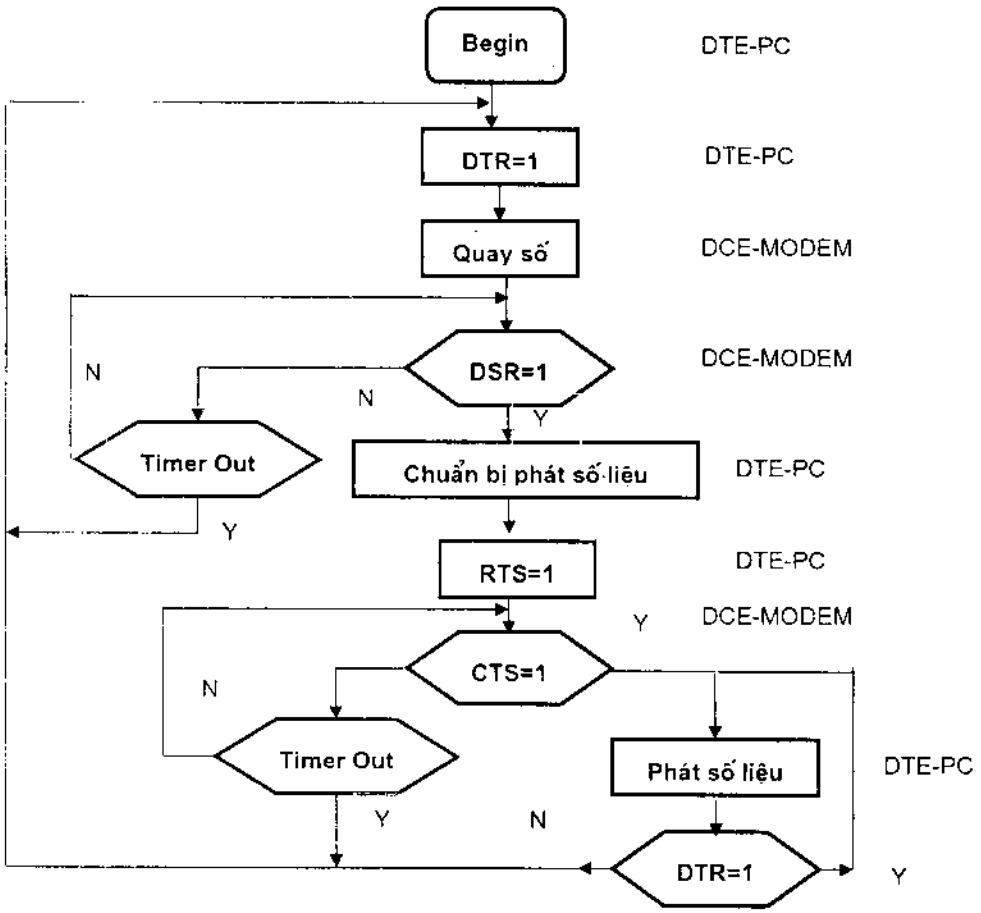
Khi MODEM gọi kiểm tra thấy mất tín hiệu sóng mang nó sẽ đặt CD Off, máy tính gửi yêu cầu được chuyển số liệu (RTS=1) và MODEM đồng ý chuyển số liệu (CTS=1). Người sử dụng gõ bản tin cần chuyển đi. Khi quá trình thu/ phát số liệu được hoàn tất, cả hai tín hiệu sóng mang sẽ được ngắt (Switch Off), cuộc gọi kết thúc. Thuật toán của quá trình này được thể hiện trên hình 6.4.

- ***Khối trung tâm điều khiển***

Khối này có nhiệm vụ điều khiển hoạt động của MODEM - mức độ thông minh của thiết bị chủ yếu phụ thuộc khối này - sự thông minh nhờ việc sử dụng hệ vi xử lý để tạo ra các khả năng phát hiện và sửa lỗi tự kiểm tra tại chỗ và từ xa, hay cho phép nhiều tùy chọn thuận tiện cho người sử dụng. Bộ đệm Buffer của MODEM dùng để lưu trữ số liệu với dung lượng hợp lý, cho phép phối hợp tốc độ giữa DTE và MODEM, mặt khác có thể tổ chức triển khai các thuật toán nâng cao độ đáng tin của tin tức như sử dụng thủ tục điều khiển dòng dữ liệu (Flow Control Protocol).

- ***Khối điều chế***

Tùy theo thiết bị sử dụng phương pháp điều chế nào, khối này thực hiện biến đổi dãy tín hiệu số thành tín hiệu tương tự có tần số nằm trong giải tần của kênh thông tin và có dạng sóng đảm bảo ít bị tổn hao trên đường truyền và ít bị tác động của can nhiễu. Tín hiệu thích hợp nhất trong trường hợp này là tín hiệu hình sin.



Hình 6.4. **Lưu đồ thuật toán của quá tình thu/ phát**

- ***Khối giải điều chế***

Chức năng khối này ngược với khối điều chế - biến đổi tín hiệu tương tự nhận được từ kênh thông tin, thành dãy tín hiệu số để chuyển vào khối trung tâm xử lý. Việc biến đổi như thế nào, tùy thuộc vào phương pháp điều chế mà hệ thống sử dụng (AFK, FSK, BPSK hay QPSK).

- ***Mạch lọc phát***

Mạch lọc phát đảm bảo hạn chế băng tần của tín hiệu sau điều chế. Trong các MODEM điều tần, sử dụng chế độ song công, tại MODEM gọi,

hài bậc hai của tín hiệu phát rơi vào giải thông của bộ lọc thu, nếu không có bộ lọc sẽ dẫn tới xuyên nhiễu làm méo tín hiệu thu.

- ***Mạch lọc thu***

Bao gồm các khâu lọc tích cực, đảm bảo việc loại trừ ảnh hưởng của can nhiễu tác động trên đường truyền và qua các khâu của thiết bị.

- ***Phân đường thu phát***

Trong chế độ song công, tại một thời điểm MODEM phải thực hiện truyền tín hiệu sau bộ lọc phát ra đường thoại và thu tín hiệu từ MODEM xa gửi tới trên một đôi dây. Mạch phân đường thu phát chính là một bộ khuếch đại vi sai, nó cấm tín hiệu phát quay trở về bộ lọc thu và ngược lại - tín hiệu thu từ đường dây không đưa vào mạch lọc phát.

- ***Phối hợp đường thoại***

Nhiệm vụ của khối này là phối hợp trở kháng cổng MODEM với trở kháng đường dây (600ohm), phối hợp mức điện áp giữa MODEM và điện áp của đường dây và đảm bảo tính đối xứng của đường dây (Telephone Line).

6.1.2. Các chế độ hoạt động của SMART MODEM

SMARTMODEM là MODEM thông minh có thể hoạt động ở các chế độ sau:

- Chế độ trực tiếp (Direct MODE).
- Chế độ bình thường (Normal MODE).
- Chế độ an toàn (Reliable MODE).
- Chế độ an toàn tự động (Auto - Reliable MODE).

Dùng lệnh AT/N để đặt các chế độ này và sau đây chúng ta xét từng chế độ làm việc của MODEM.

- ***Chế độ trực tiếp***

Trong chế độ liên lạc trực tiếp, tốc độ thông tin lớn nhất sẽ bằng tốc độ đường truyền. Nếu tốc độ MODEM là 1200bps thì tốc độ đầu cuối

(DTE) cũng phải là 1200bps, và tốc độ thông tin không bao giờ vượt quá 1200 bps. Lệnh (AT/Q) bị bỏ qua trong chế độ này, tốc độ của cổng nối tiếp phải điều chỉnh tương đương với tốc độ của MODEM.

- ***Chế độ bình thường***

Trong chế độ này MODEM sử dụng bộ đệm (Buffer) trong thao tác thu/ phát, điều đó cho phép tốc độ đầu cuối (DTE - PC) khác với tốc độ đường truyền. Tuy nhiên tốc độ dữ liệu không thể vượt qua tốc độ đường truyền. Nếu tốc độ DTE là 9600bps, tốc độ đường truyền là 2400bps, thì trong trường hợp bộ đệm chưa bị đầy, DTE vẫn có thể gửi dữ liệu ra MODEM với tốc độ 9600bps. Mật độ thông tin không thể vượt quá 2400bps. Các phương pháp Flow Control phải được sử dụng để tránh trường hợp mất dữ liệu.

- ***Chế độ an toàn***

Trong chế độ an toàn, MODEM cho phép sử dụng bộ đệm dữ liệu, thêm vào đó nó còn cho phép sử dụng giao thức V.42/MNP với chức năng phát hiện và xử lý lỗi. Mật độ thông tin lớn nhất tùy thuộc mức độ đàm phán liên lạc an toàn và kiểu dữ liệu được truyền, nhưng không bao giờ vượt quá tốc độ DTE thấp nhất. Cổng nối tiếp RS232 phải sử dụng phương pháp Flow Control để tránh mất dữ liệu.

Khi hai MODEM với giao thức V.42/MNP kết nối với nhau lần đầu, chúng tiến hành đàm thoại với nhau về sự liên kết và tất cả các tùy chọn có thể. Cụ thể nếu một MODEM có khả năng V.42 và MODEM kia có khả năng MNP Class 4, thì kết nối sẽ sử dụng thủ tục Class 4. Nếu một MODEM được đặt ở chế độ an toàn và MODEM kia không có khả năng sử dụng một thủ tục an toàn nào cả, thì sau khi phát hiện sai sót, MODEM sẽ ngắt đường truyền và trả lại mã NO CARRIER (không có sóng mang).

- ***Chế độ an toàn tự động***

Trong nhiều trường hợp MODEM phải liên lạc với MODEM xa hoặc không có V.42/MNP, khi đó có thể sử dụng chế độ hoàn toàn tự động.

Trong chế độ này, MODEM sẽ tìm kiếm các ký tự của giao thức V.42/MNP từ MODEM xa. Nó phát hiện trong vòng 4 Sec và cố gắng thiết lập kết nối an toàn.

Nếu không phát hiện ra các ký tự của V.42/MNP và nếu chế độ tự động điều chỉnh tốc độ là ON, đường liên lạc trực tiếp sẽ được thiết lập. Nếu tự động điều chỉnh tốc độ là OFF, MODEM sẽ trả về chế độ liên lạc bình thường và sử dụng Flow Control nếu nó được kích hoạt. Cổng nối tiếp cũng phải sử dụng Flow Control để tránh mất dữ liệu.

6.1.3. Các thủ tục truyền số liệu

Trước khi bắt đầu truyền tin, một môi trường vật lý truyền dẫn được thiết lập giữa hai đầu thu phát. Ví dụ hai máy tính sử dụng MODEM để liên lạc với nhau qua mạng điện thoại hay qua hệ thống thông tin vệ tinh. Ngoài các vấn đề đối với đường truyền, nếu không tuân theo một thủ tục chuẩn nào, thì không có gì đảm bảo rằng số liệu thu được một cách hoàn hảo. Ví dụ kí tự đi vào từ đường truyền sẽ bị mất, nếu tại thời điểm đó đang bận các thao tác khác mà không có thủ tục đặc biệt để xử lý ghi nhận kí tự đó lại, hay khi truyền dữ liệu tới máy in, do tốc độ in chậm, nếu không phối hợp tốc độ hợp lý sẽ xảy ra tình trạng tràn bộ đệm của máy in, gây sai sót.

Sau đây là một số thủ tục truyền số liệu cơ bản được xét

- ***Thủ tục điều khiển dòng dữ liệu (Flow Control Protocol)***

Flow Control cho phép đặt ON hoặc OFF dòng thông tin giữa các thiết bị tham gia truyền tin. Giao diện giữa DTE (PC) và DCE (MODEM) là cổng RS232, giao diện giữa DCE và kênh thông tin là cổng của MODEM.

Phần lớn các máy tính và DTE có khả năng Flow Control, trường hợp không có thì thủ tục này phải bị cấm trên cổng MODEM và cần phải điều chỉnh tốc độ trên cổng RS232 bằng tốc độ cổng MODEM.

MODEM có thể truyền và nhận dữ liệu trên cổng RS232 có tốc độ khác tốc độ cổng MODEM. Nếu tốc độ truyền trên cổng RS232 ra

MODEM lớn hơn tốc độ cổng MODEM thì bộ đệm của MODEM sẽ bị đầy, lúc này nếu có thủ tục Flow Control thì tránh được mất dữ liệu.

Có hai thủ tục Flow Control là Software Flow Control và Hardware Flow Control.

- *Software Flow Control* (Điều khiển dòng dữ liệu bằng phần mềm): Thủ tục này gửi ký tự XOFF (tương ứng Ctrl-S) khi muốn dừng dòng dữ liệu và gửi XON (tương ứng Ctrl-Q) khi tiếp tục truyền dữ liệu. XOFF và XON có thể điều khiển cả 2 cổng là RS 232 và cổng MODEM.

Khi bộ đệm phát của MODEM A bị đầy (do đường truyền xấu hay MODEM B không kịp xử lý dữ liệu) thì nó truyền cho PC A 1 ký tự XOFF yêu cầu dừng việc đưa dữ liệu ra. Quá trình phát chỉ tiếp tục khi PC A nhận được ký tự XON.

Khi bộ đệm thu của MODEM A bị đầy (do dòng dữ liệu từ MODEM B đến quá nhanh), nó gửi cho MODEM B một ký tự XOFF để tạm dừng. Khi nào có thể tiếp tục nhận, nó sẽ gửi đi ký tự XON cho MODEM B.

Hardware RTS/CTS Flow Control (Điều khiển dòng dữ liệu bằng phần cứng): Thủ tục này sử dụng các chân tín hiệu của RS 232 là CTS và RTS để điều khiển dòng thông tin.

Trong điều khiển dòng thông tin một chiều (Undirectinoal Flow Control), khi mức tín hiệu trên CTS bị hạ xuống 0, tương ứng với việc gửi XOFF đến DTE và khi mức CTS nâng lên 1 tương ứng với việc gửi XON. Đối với DTE phải có khả năng trả lời tín hiệu CTS. Kiểu điều khiển này chỉ điều khiển dòng dữ liệu từ DTE đến MODEM, còn chiều ngược lại sẽ không được kiểm tra.

Trong điều khiển dòng dữ liệu 2 chiều (Bidirectional Flow Control). DTE đáp lại tín hiệu CTS như trong trường hợp điều khiển dòng một chiều, thêm vào đó, RTS ở mức thấp tương ứng với việc truyền XOFF tới MODEM và RTS ở mức cao tương ứng với việc truyền XON. Khi này cả hai chiều của dòng dữ liệu đều được điều khiển.

- ***Thủ tục kiểm tra lỗi và nén dữ liệu***

Các thế hệ MODEM V.42 và V.42bis có tùy chọn cho việc kiểm tra lỗi và nén dữ liệu. Khi phát hiện sai sót, MODEM sẽ tự động yêu cầu

truyền lại các dữ liệu bị lỗi. Việc nén các dữ liệu cho phép tăng mật độ luồng thông tin, tăng hiệu suất sử dụng kênh.

- Chuẩn V.42 và V.42bis:

V.42 là chuẩn kiểm tra lỗi của CCITT, bằng cách dùng tổng kiểm tra hay mã kiểm tra CRC với cách biểu diễn các từ mã bằng các đa thức cho phép sử dụng mã Cyclic để kiểm tra lỗi..

Chuẩn V.42bis thiết kế cho việc nén dữ liệu với tỷ lệ 4:1 dựa trên thuật toán của British Telecom Lempelzip, trong đó một chuỗi ký tự được mã hóa bằng một từ mã có độ dài cố định.

- Thủ tục MNP (Micrcom Networking Protocol):

Thủ tục này cho phép phát hiện và sửa lỗi trong dòng số liệu không đồng bộ và trong các thủ tục truyền file NMP tương ứng với tầng liên kết dữ liệu trong mô hình tham chiếu OSI; cho phép bảo đảm truyền dữ liệu - điểm rất an toàn. MNP đã trở thành chuẩn trong công nghiệp.

Có 4 Class MNP mà thuộc tính cơ sở của nó là mỗi một Class có thể làm việc với các Class mức thấp hơn. Khi mối liên kết MNP được thiết lập, hai thiết bị sẽ đàm thoại, sau đó hoạt động với Class cao nhất cho phép. Các thiết bị MNP cũng có thể làm việc với các thiết bị không MNP. Tất cả các MNP Class sẽ đóng gói các dữ liệu không đồng bộ thành các gói liên kết HDLC trước khi truyền.

+ MNP Class 1: MNP Class1 sử dụng phương pháp truyền khối Byte bán song công, không đồng bộ. Hiệu suất của các MODEM sử dụng giao thức này khoảng 70%.

+ MNP Class 2: MNP Class 2 được bổ xung thêm khả năng song công cho MNP Class 1. Hiệu suất của các MODEM sử dụng giao thức này khoảng 84%. Thực tế, MODEM MNP Class 2, tốc độ 2400bps chỉ đạt tốc độ 2000bps.

+ MNP Class 3: MNP Class 3 truyền các gói tin Synchronous Bit oriented Half Duplex có format trong đó giảm bớt việc truyền các bit Start, Stop. Hiệu suất của các MODEM MNP Class 3 khoảng 108%. Như vậy 1 MODEM vận tốc 2400bps có thể đạt trong thực tế 2600bps.

+ MNP Class 4: MNP Class 4 có thêm khả năng đóng gói tin và khả năng tối ưu format dữ liệu pha để cải thiện hiệu suất của MODEM. Trong quá trình truyền dữ liệu MNP theo dõi tính an toàn của môi trường truyền dẫn. Nếu kênh truyền hoàn toàn không có lỗi, MNP sẽ tăng kích thước của trường dữ liệu trong mỗi gói tin - tăng mật độ dữ liệu trên kênh. Nếu dữ liệu có sai sót MNP sẽ chuyển sang đóng gói các gói tin nhỏ hơn, lúc này mật độ dữ liệu trên kênh giảm và số các gói tin phải truyền cũng giảm bớt. Giao thức MNP Class 4 tối ưu format dữ liệu bằng cách giảm bớt các thông tin quản lý thừa ở phần đầu của các khung. Hiệu suất của MODEM khoảng 120%. MODEM tốc độ 2400bps có thể đạt tới 2900bps. Các thiết bị sử MNP Class 4 nên đặt tốc độ 4800bps để tận dụng hết khả năng của giao thức này.

- ***Các thủ tục truyền file***

Khi truyền một file dữ liệu có thể truyền lần lượt từng byte cho đến khi gặp ký tự EOF (End of file) thủ tục này đơn giản dễ thực hiện, nó bỏ qua công đoạn kiểm tra phát hiện lỗi, sử dụng truyền tin khi yêu cầu độ đáng tin không cao.

Để tăng độ đáng tin có thể thực hiện phát lặp file dữ liệu với một số lần nhất định, ở đầu thu tiến hành lựa chọn để có bản tin ít lỗi nhất. Thủ tục này đơn giản, nhưng làm giảm hiệu suất sử dụng kênh và khi kênh tốt, việc phát lặp là thừa, còn khi kênh xấu, thì chưa đủ độ tin cậy. Về nguyên tắc, có thể sử dụng mã sửa sai để phát hiện và sửa các lỗi, nhưng việc thêm nhiều phần tử kiểm tra vào chuỗi tin làm giảm hiệu suất của kênh.

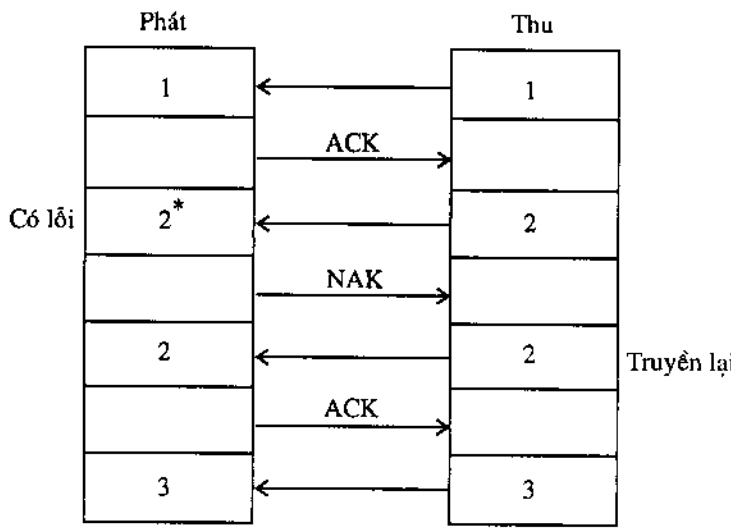
Thực tế thường tổ chức thông tin thành các gói (Packet) bắt đầu bằng các tiếp đầu (Header) và kết thúc bằng các byte kiểm tra cho phép phát hiện lỗi trong gói tin, nếu có lỗi yêu cầu bên phát truyền lại khối tin đó. Các thủ tục dựa theo nguyên tắc này vừa đảm bảo độ tin cậy cần thiết của tin tức, vừa đảm bảo hiệu suất kênh truyền vì chỉ phải truyền lại những khối tin bị lỗi và số các phần tử kiểm tra không lớn lắm, chỉ cần phát hiện, không cần sửa lỗi. Sau đây là một số thủ tục truyền file chuẩn.

- *Thủ tục yêu cầu tự động lặp lại* (Automatic Repeat Request - ARQ).

Bản chất của thủ tục này là các gói tin bị lỗi hoặc không có xác nhận từ phía thu thì được tự động truyền lại.

+ Thủ tục ARQ truyền và đợi (Send and Wait ARQ).

Sau khi truyền xong một khối tin, nếu phía thu nhận đúng sẽ báo về bên phát, tín hiệu xác nhận đúng ACK (Ackmowledge) thì tiếp tục truyền khối tin mới, nếu khối tin có lỗi tín hiệu báo lại là NAK, bên phát sẽ tự động phát lại khối tin đó, cho tới khi bên thu nhận được một cách hoàn hảo.



+ Thủ tục ARQ truyền liên tục.

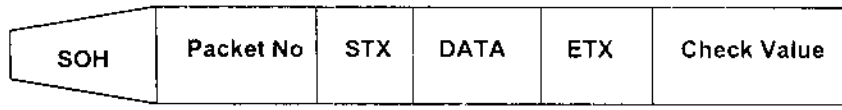
Bên phát truyền liên tục các khối tin, bên thu cũng tiến hành kiểm tra và báo về các thông tin ACK, NAK và kèm theo NAK là số thứ tự của khối tin bị lỗi. Sau khi truyền xong toàn bộ dữ liệu, bên phát truyền lại các khối tin bị lỗi đó.

Các gói dữ liệu được hình thành từ các trường điều khiển và trường dữ liệu, sau đây là một số cấu trúc điển hình của gói dữ liệu.

Gói dữ liệu giới hạn, các kí tự điều khiển.

Trong các file văn bản, chứa các ký tự mã ASCII cộng với 6 kí tự điều khiển: BS, HT, LF, VT, FF, CR. Vì thế trong trường hợp dữ liệu

không được có các kí tự điều khiển khác ngoài 6 kí tự đó. Format (Khuân mẫu) gói này như sau:



SOH (Start of Header): bắt đầu khối.

Packet No: số thứ tự của khối.

STX (Start of Text): Byte đánh dấu đầu trường văn bản.

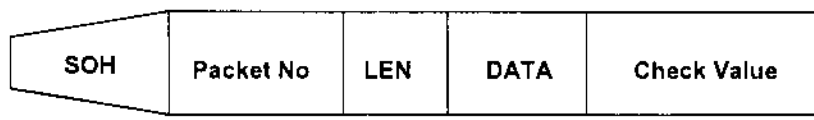
EXT (End of Text): Byte đánh dấu kết thúc văn bản,

Check Value: 1 hay nhiều byte dùng để kiểm tra lỗi.

Gói dữ liệu có giới hạn kích thước.

Khi cần truyền mảng dữ liệu không kiển thì không thể dùng kí tự đặc biệt để đánh dấu đầu và cuối.

Trường hợp dữ liệu, khối dữ liệu như sau:



SOH: bắt đầu khối.

Packet No: số thứ tự khối.

LEN: độ dài trường dữ liệu.

Date: trường dữ liệu.

Check Value: trường kiểm tra.

- *Thủ tục XMODEM.*

Thủ tục XMODEM được sử dụng rộng rãi trong các hệ thống truyền số liệu, thực chất là thủ tục ARQ.

Truyền và đợi dùng gói dữ liệu có độ dài cố định, thường là 128byte, các trường phục vụ khác có độ dài một byte.

Gói dữ liệu này có trường thứ 3 là phần bù của trường thứ 2, với tác dụng kiểm tra số thứ tự của gói tin. Nếu có sai sót ở đây thì không thể sắp xếp các gói thành file dữ liệu.

Quá trình truyền file có thể chia thành 3 giai đoạn. Ta xét các giai đoạn theo đầu thu (phát).

+ Đầu phát XMODEM. (Hình6.5). Với thủ tục XMODEM, công việc ở đầu phát đơn giản hơn ở đầu thu.

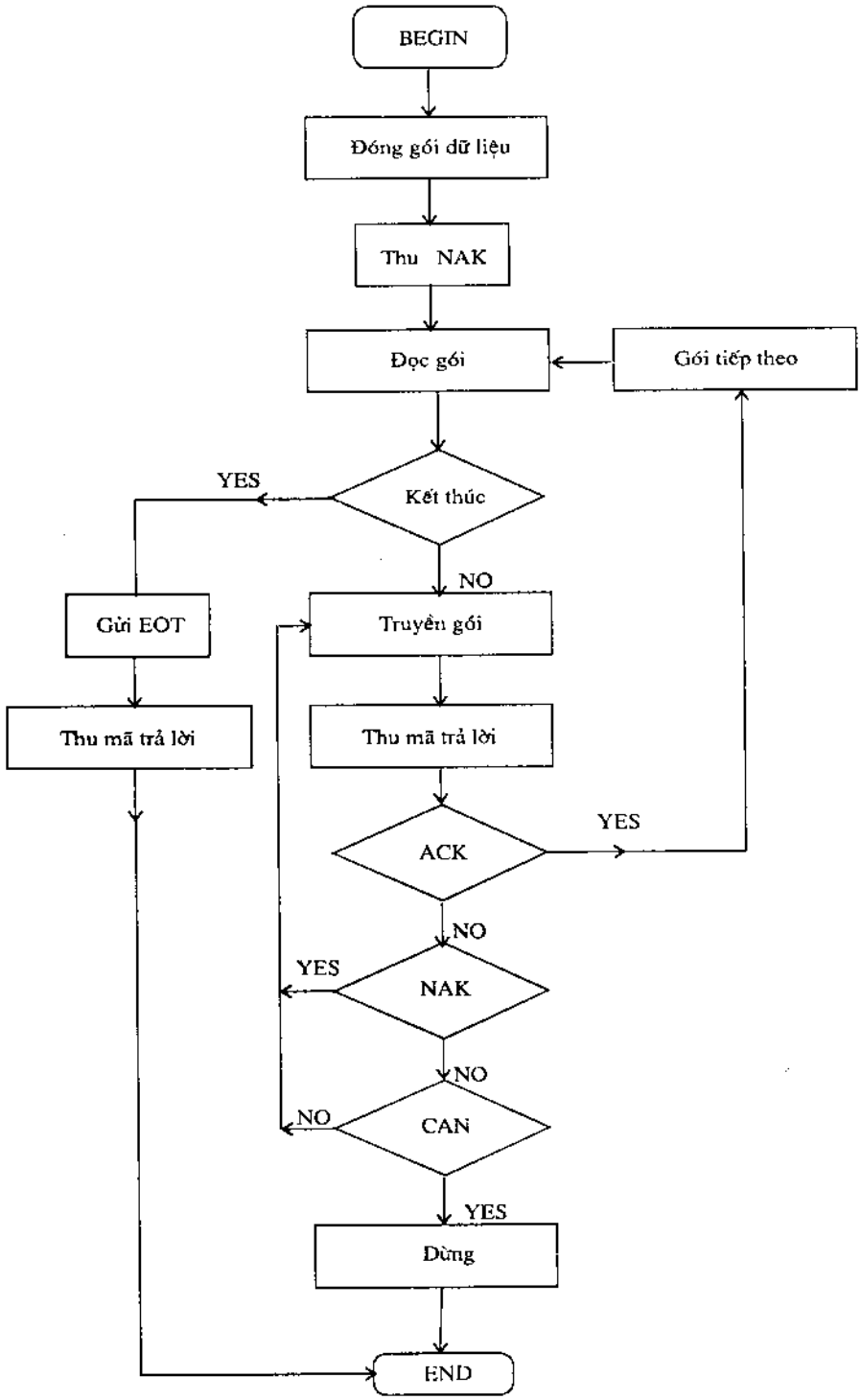
Giai đoạn 1: Trước khi truyền phải thiết lập đường truyền và tiến hành bắt tay giữa thu và phát. Việc kiểm soát truyền do một phía thực hiện, trong trường hợp này là phía thu. Đầu thu sẽ kiểm soát, điều khiển dòng các gói dữ liệu. Đầu tiên bên phát chờ một mã NAK từ phía thu, sau khi nhận được, nghĩa là phía thu đã sẵn sàng, bắt đầu truyền gói dữ liệu đầu tiên.

Giai đoạn 2: Sau khi nhận được một mã NAK khởi đầu, bên phát truyền đi một gói dữ liệu và chờ trả lời. Nếu bên thu gửi về mã ACK, nghĩa là nhận được gói tin không bị lỗi, hãy truyền gói dữ liệu tiếp theo. Nếu là mã NAK, nghĩa là gói tin vừa truyền bị lỗi, cần phát lại. Nếu là mã CAN có nghĩa là yêu cầu dừng cuộc truyền. Giai đoạn này kết thúc khi đã truyền hết file dữ liệu.

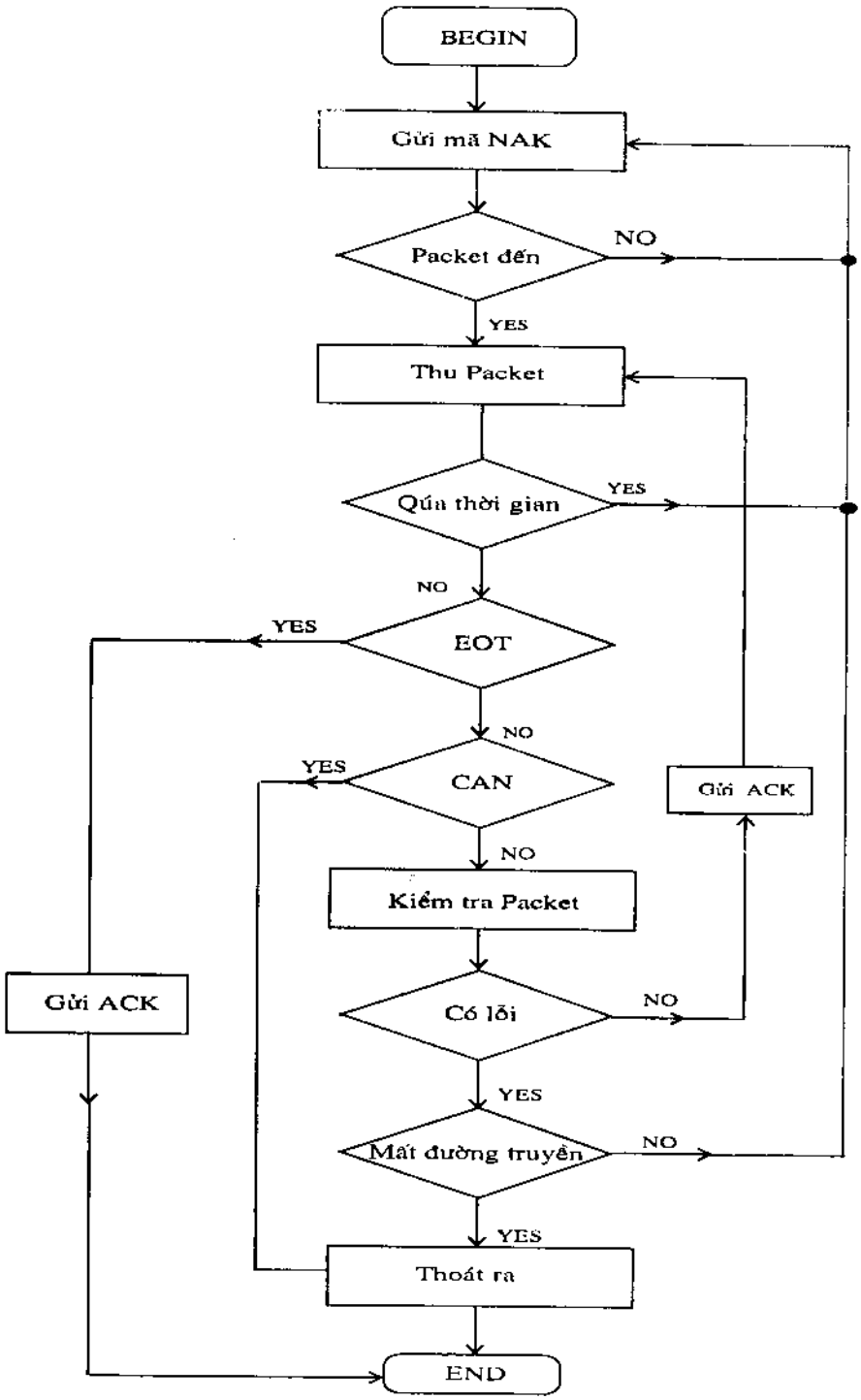
Giai đoạn 3: Nếu giai đoạn hai kết thúc bình thường, bên phát gửi mã EOT (End of Transmission) báo cho bên thu biết đã truyền hết dữ liệu, bên thu trả lời bằng một mã ACK, có thể tiến hành đóng file, giải phóng bộ nhớ... Nếu giai đoạn hai kết thúc không bình thường, chẳng hạn bằng mã CAN, thì bên phát không gửi đi mã EOT.

+ Đầu thu XMODEM. (Hình 6.6): Đầu thu không đơn thuần là thu các dữ liệu, mà còn phải căn cứ vào các trường phục vụ để có những quyết định phù hợp cho quá trình truyền. Qúa trình thu cũng chia làm ba giai đoạn.

Giai đoạn 1: Sau khi làm các việc chuẩn bị thu (khởi tạo bộ đệm thu, mở file...) bên thu gửi một mã NAK báo rằng đã sẵn sàng chuyển sang giai đoạn hai.



Hình 6.5. Lưu đồ thuật toán XMODEM phát



Hình 6.6. Lưu đồ thuật toán XMODEM đầu thu

Giai đoạn 2: Khởi động dòng thu và chờ Packet, nếu sau 10 giây mà chưa thấy thì gửi tiếp một mã NAK nữa. Nếu phát hiện có tín hiệu thì bên thu thực hiện các bước sau:

° Xét mã SOH - mã đánh dấu đầu khối dữ liệu, nếu là khối cuối cùng thì đó lại là mã EOT, lúc này cần kết thúc cuộc truyền.

° Kiểm tra số thứ tự của khối bằng cách thực hiện phép XOR giữa trường thứ hai và thứ ba, nếu kết quả khác không, nghĩa là có lỗi thì gửi đi một mã NAK để yêu cầu phát lại khối và quay lại từ đầu, nếu kết quả bằng không thì tiếp tục bước sau.

° Kiểm tra số thứ tự khối, nếu phát hiện có sự nhầm lẫn về trật tự thì gửi mã CAN yêu cầu kết thúc cuộc truyền. Nếu số thứ tự là liên tục thì tiến hành thu dữ liệu, tính tổng kiểm tra theo thuật toán quy định, so sánh với tổng kiểm tra bên phát gửi sang, nếu trùng khớp, nghĩa là dữ liệu không có lỗi, thì gửi mã ACK, nếu không trùng khớp thì gửi mã NAK, yêu cầu phát lại khối dữ liệu đó.

Giai đoạn 3: Nếu cuộc truyền kết thúc hoàn hảo, khi nhận được mã EOT, bên thu gửi mã trả lời ACK và làm các bước như đóng file, giải phóng bộ nhớ... kết thúc liên lạc.

6.2. TỔ CHỨC CỔNG TRUYỀN THÔNG (COM) CỦA PC

Đại đa số các máy vi tính được thiết kế có các cổng vào/ra (I/0 Ports) để trao đổi số liệu với các thiết bị ngoại vi. Khi ở cự ly gần, số liệu có thể truyền song song các bít để nâng cao tốc độ, cổng giao tiếp để truyền số liệu theo cách này gọi là cổng song song (parallel port), điển hình là cổng nối với máy in.

Khi truyền số liệu đi xa, không thể truyền song song các bít mà phải truyền nối tiếp theo phương thức đồng bộ hay không đồng bộ (Serial Sycronous hay Serial Asyncronous).

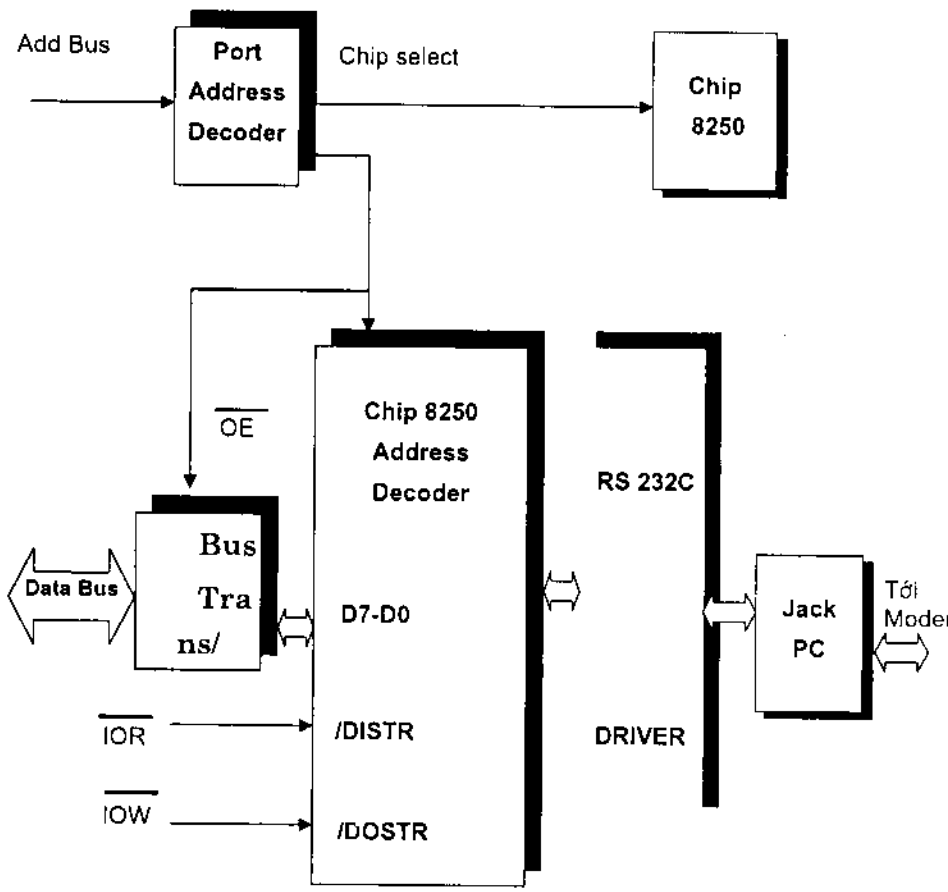
Họ máy tính PC thường được trang bị hai cổng nối tiếp, gọi là COM1, COM2, mỗi cổng gồm một mạch điều khiển và phối hợp tín hiệu mà phần tử quan trọng nhất là mạch tích hợp ISN 8250 thực hiện toàn

bộ các chức năng truyền tin nối tiếp không đồng bộ như biến đổi số liệu song song thành số liệu nối tiếp và ngược lại, điều khiển tốc độ truyền vv... Với chức năng như thế người ta còn gọi IC 8250 là UART (Universal asyncronous Receiver / Transniter).

Giao diện giữa UART với thiết bị truyền dẫn (DCE) là các chuẩn của CCITT như RS232, RS422, RS423, X21, V24, V28...

Chuẩn RS 232 ra đời sớm nhất gồm có 25 tín hiệu (như đã mô tả ở phần trên) sau đó được cải tiến thành RS 232-C chỉ sử dụng 10 tín hiệu của RS 232.

Sơ đồ khối cổng COM được mô tả trên hình 6.7.



Hình 6.7. Cổng COM của PC

- ***Các thanh ghi của ISN 8250***

ISN 8250 có một trường thanh ghi mà ta có thể can thiệp vào nội dung của chúng bằng các lệnh vào ra theo địa chỉ cho trước. Các thanh ghi của 8250 dành cho cổng COM1 là từ $3F8 đến $3FE, với cổng COM2 từ $2F8 đến $2FE. Các thanh ghi đều có độ dài là 8 bit.

Thí dụ muốn đọc nội dung thanh ghi RxD (đệm thu) có địa chỉ $3F8 vào biến Char, ta dùng một lệnh của Pascal:

```
Char:= Port [$3F8];
```

hay nạp nội dung thanh ghi MXR có địa chỉ $3Fc bằng giá trị của biến a, ta dùng lệnh: port [$3FC]: = a;

Mỗi thanh ghi được sử dụng vào các chức năng khác nhau, giúp người lập trình điều khiển một cách mềm dẻo quá trình truyền số liệu. Nội dung, ý nghĩa các thanh ghi đó như sau:

a. Thanh ghi TxD, RxD (thanh ghi đệm phát và thanh ghi đệm thu)

- Hai thanh ghi này có cùng địa chỉ $3F8 với COM1 và $2F8 với COM2.

- Khi phát, ta nạp byte dữ liệu vào thanh ghi TxD, ISN 8250 sẽ lần lượt truyền các bit của nó ra đường TxD của RS 232 cùng với bit Start và stop.

- Khi thu được một ký tự, ISN 8250 chứa vào RxD để ta đọc vào và xử lý.

b. Thanh ghi IER (intrrupt Enable Register - cho phép ngắt)

Địa chỉ $3F9 ($2F9).

Bốn bít cao của thanh ghi này luôn bằng 0, mỗi bit trong bốn bit thấp khi bằng 1 thì cho phép một ngắt tương ứng:

- Bit0 = 1: Cho phép ngắt thanh ghi nhận RxD đã đầy.
- Bit1 = 1: Cho phép ngắt thanh ghi truyền TxD rỗng.
- Bít 2 = 1: Cho phép ngắt từ trạng thái đường truyền.
- Bít 3 = 1: Cho phép ngắt từ trạng thái modem

c. Thanh ghi IR (Intrrupt Identification Register - định danh ngắt)

Địa chỉ $3FA ($2FA).

Thanh ghi này được đọc mỗi khi có yêu cầu ngắt từ 8250 để xem nguyên nhân nào gây ngắt.

d.Thanh ghi LCR (Line Control Register - Điều khiển đường truyền)

Địa chỉ $3FB ($2FB)

Ý nghĩa các bit như sau:

Bit	D1	D0	
	0	0	: 5 data bits
	0	1	: 6 data bits
	1	0	: 7 data bits
	1	1	: 8 data bits
Bit	D2 = 0		: Dùng 1 bit stop
	= 1		: Tùy độ dai data bit sẽ có 1 hoặc 2 stopbit.
Bit	D3 = 1		: Có 1 parity bit ở cuối ký tự 9 nếu data bit <8)
Bít	D4 = 1		: Có Parity chẵn
	= 0		: Parity lẻ
Bit	D6 =1		: Ngừng cuộc truyền (Ctrl - Break)
Bit	D7 = 1		: Chỉ rằng thanh ghi TxD và IER chứa số chia (2 byte) của tần số nhịp của 8250 để tính tốc độ truyền. Bình thường bit này luôn bằng 0. Bit này còn gọi là DLAB bit.

e. Thanh ghi MCR (Modem Control Register - Điều khiển modem)

Địa chỉ $3FC ($2FC)

Bit 0 = 0 : chân DTR (pin 20) của RS232 sẽ lên mức lôgic 1 (=-12)

= 1 : chân DTR (pin 20) của RS232 sẽ về mức lôgic 0 (+12)

Bit 1 = 0 : chân RTS (pin 4) của RS232 sẽ lên mức lôgic 1 (=-12)

=1 : chân RTS (pin 4) của RS232 sẽ lên mức lôgic 0 (=+12)

Bit 4 = 1 : Loop back control, đầu ra của TxD nối với đầu vào của RxD để tạo mạch vòng kiểm tra.

Bit 5, 6, 7 luôn bằng 0.

d. Thanh ghi LSR (line Status Register-Trạng thái đường truyền).

Địa chỉ $3FD ($3FD)

Bit 0 = 1 : Đã nhận xong 1 ký tự và đng lưu tại RxD.

Bit 1 = 1 : Báo lỗi Overrun, một ký tự trong RxD chưa kịp đọc đã bị ký tự sau ghi đè lên.

Bit 2 = 1 : Báo có lỗi Parity

Bit 3 = 1 : Lỗi khung (Framing error)

Bit 4 = 1 : Break interrupt (khi gặp tín hiệu 0 dài quá một ký tự)

Bit 5 = 1 : Transmite Register Empty (ký tự cuối cùng đã truyền đi, có thể gửi ra ký tự tiếp theo)

Bit 6 = 1 : Transmite Shift Register Empty (Thanh ghi chuyển đổi song song --> nối tiếp dã rỗng, ý nghĩa gần giống bit 5).

Bit 7 luôn bằng 0.

e. *Thanh ghi MSR* (Modem Status Register: Trạng thái modem)

Địa chỉ $3FE ($2FE)

Từ bit 0 đến bit 3 được xóa về 0 sau khi đọc thanh ghi. Các bit này lập lên mức 1 khi có sự thay đổi trạng thái của chân RS232

Tương ứng : Bit 0: Delta Clear To Send

Bit 1: Delta Set Ready

Bit 2: Ring (có chuông)

Bit 0: Delta Rx line signal detect

Từ bit 4 đến bit 7 dùng trong chế độ test vòng, phản ánh trạng thái các chân RS 232:

bit 4 = CTS

Bit 5 = DSR

Bit 4 = RI

Bit 4 = DCD

6.3. XÂY DỰNG CHƯƠNG TRÌNH ĐIỀU KHIỂN MODEM

Chương trình điều khiển Modem được viết trên ngôn ngữ lập trình hướng đối tượng DELPHI để chạy trên điều hành đa nhiệm WINDOWS. Trong phần mềm này có hai thành phần. Thành phần thứ nhất là chương trình Driver điều khiển cơ chế ngắt cổng COM của PC. Chương trình này phải được xây dựng dưới dạng một Component của hệ thống và cài đặt vào thư viện tài nguyên của hệ thống. Thành phần thứ hai là chương trình điều khiển mọi chức năng thu phát thông tin của hệ truyền tin MODEM này. Bây giờ sẽ xem xét từng thành phần của phần mềm cần xây dựng.

6.3.1.Xây dựng chương trình driver TCommPortDriver

Chương trình driver điều khiển cho cổng COM của PC được xây dựngdưới dạng một Component của hệ thống. Phương pháp xây dựng các Component và cài đặt chúng trong DELPHI đã được trình bầy ở chương 4. Tại đây chỉ đề cập tới cấu trúc của Component điều khiển cho cổng COM. Đặt tên Component này là ComPortDRIVER, nó phải có cấu trúc cơ bản như sau:

- ***Thuộc tính của ComPortDRIVER***

Trên cửa sổ thuộc tính của hộp thoại Object inspecter của cửa sổ soạn thảo chính phải có các trường như sau:

property ComPort: chọn cổng COM

property ComPortDataBits: chọn số lượng bit dữ liệu.

property ComPortHwHandshaking:	chọn chế độ giao tiếp (không sử dụng hoặc sử dụng chế độ bắt tay bằng phần cứng)
property ComPortInBufSize:	chọn dung lượng bộ đệm vào (byte).
property ComPortOutBufSize:	chọn dung lượng bộ đệm ra (byte).
property ComPortParity:	chọn kiểu Parity (không sử dụng hoặc sử dụng chẵn hoặc sử dụng lẻ...).
property ComPortPollingDelay:	chọn thời gian trễ Polling (ms).
property ComPortSpeed:	chọn tốc độ truyền
property ComPortStopBits:	chọn số lượng bit STOP
property ComPortSwHandshaking:	chọn chế độ giao tiếp (không sử dụng hoặc sử dụng chế độ bắt tay bằng phần mềm).
property EnableDTROnOpen:	cho phép DTE sẵn sàng.
property Name:	tên của Component (CommPortDriver):
property OutputTimeout:	thời gian đợi (ms).

Trên cửa sổ sự kiện (EVEN) của hộp thoại Object inspecter của cửa sổ soạn thảo chính phải có trường như sau:

property OnReceiveData: tạo ngắt thu cho cổng COM hiện hành.

- ***Các modul chức năng***

- Modul thiết lập cổng COM đặt tên là SetComPort có chức năng gán giá trị vào trường FComPort của hộp thoại Object Ispecter để kết nối với cổng COM tương ứng.

- Modul thiết lập tốc độ truyền tin cho cổng COM đặt tên là SetComPortBaudRate có chức năng gán giá trị vào trường FComPortBaudRate của hộp thoại Object Ispecter.

- Modul thiết lập số lượng bit DATA trong khung tin cho cổng COM đặt tên là SetComPortDataBits có chức năng gán giá trị vào trường FComPortDataBits của hộp thoại Object Ispecter.

- Modul thiết lập số lượng bit STOP trong khung tin cho cổng COM đặt tên là SetComPortStopBits có chức năng gán giá trị vào trường FComPortStopBits của hộp thoại Object Ispecter.

- Modul thiết lập số lượng bit PARITY trong khung tin cho cổng COM đặt tên là SetComPortParity có chức năng gán giá trị vào trường FComPortParity của hộp thoại Object Ispecter.

- Modul thiết lập chế độ bắt tay bằng phần cứng cho cổng COM đặt tên là SetComPortHwHandshaking có chức năng gán giá trị vào trường FComPortHwHandshaking của hộp thoại Object Ispecter.

- Modul thiết lập chế độ bắt tay bằng phần mềm cho cổng COM đặt tên là SetComPortSwHandshaking có chức năng gán giá trị vào trường FComPortSwHandshaking của hộp thoại Object Ispecter.

- Modul đặt bộ đệm vào cho cổng COM đặt tên là SetComPortInBufSize có chức năng gán giá trị vào trường FComPortInBufSize của hộp thoại Object Ispecter và cấp phát bộ đệm này với dung lượng là giá trị trong FComPortInBufSize.

- Modul đặt bộ đệm ra cho cổng COM đặt tên là SetComPortOutBufSize có chức năng gán giá trị vào trường FComPortOutBufSize của hộp thoại Object Ispecter.

- Modul đặt thời gian giữ chậm cho cổng COM đặt tên là SetComPortPollingDelay có chức năng gán giá trị vào trường FComPortPollingDelay của hộp thoại Object Ispecter và dùng hàm SetTimer để khởi động lại đồng hồ.

- Modul ApplyCOMSettings là modul thiết lập chế độ làm việc cho cổng COM với việc sử dụng một loạt các hàm chức năng làm việc với cấu trúc chuẩn TDCB (Device Control Block). Trước tiên là hằng Win16BaudRates được khai báo với các tốc độ 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200bps.

Sử dụng hàm GetCommState(FComPortID, dcb) để lấy ấy trạng thái hiện hành của cổng COM. Sau đó xử lý các thông số cho TDCB. TDCB có cấu trúc được khai báo theo kiểu record như sau:

```
TDCB = record
 Id: Byte;              {Đặc tả thiết bị nội tại - ID }
 BaudRate: Word;        {Tốc độ baud/sec }
 ByteSize: Byte;        {Số lượng bit/byte, 4-8 }
 Parity: Byte;          {0-4=None,Odd,Even,Mark,Space}
 StopBits: Byte;        {0,1,2 = 1, 1.5, 2 }
 RlsTimeout: Word;      {Timeout cho RLSD}
 CtsTimeout: Word;      {Timeout cho CTS}
 DsrTimeout: Word;      {Timeout cho DSR}
 Flags: Word;

(* Phần chú giải chức năng
 Byte fBinary: 1;       {Binary Mode: kiểm tra EOF}
 Byte fRtsDisable:1;    {Không xác nhận RTS tại thời điểm
                             khởi tạo}
 Byte fParity: 1;       {Cho phép kiểm tra parity}
 Byte fOutxCtsFlow:1;   {Bắt tay kiểu CTS cho output }
 Byte fOutxDsrFlow:1;   {Bắt tay kiểu DSR cho output }
 Byte fDummy: 2;        {dự trữ }
 Byte fDtrDisable:1;    {Không xác nhận DTR tại thời điểm
                             khởi tạo}
 Byte fOutX: 1;         {Cho phép output X-ON/X-OFF }
 Byte fInX: 1;          {Cho phép input X-ON/X-OFF }
 Byte fPeChar: 1;       {Cho phép Parity Err}
 Byte fNull: 1;         {Cho phép Null stripping }
 Byte fChEvt: 1;        {Cho phép ký tự sự kiện Rx }
 Byte fDtrflow: 1;      {Bắt tay kiểu DTR ở đầu input }
 Byte fRtsflow: 1;      {Bắt tay kiểu RTS ở đầu input }
 Byte fDummy2: 1;
*)
```

```
    XonChar: char;        {Tx và Rx X-ON character }
    XoffChar: char;       {Tx và Rx X-OFF character }
    XonLim: Word;         {phát X-ON ngưỡng }
    XoffLim: Word;        {phát X-OFF ngưỡng}
    PeChar: char;         {Lỗi Parity}
    EofChar: char;        {Ký tự kết thúc End of Input}
    EvtChar: char;        {ký tự sự kiện thu }
    TxDelay: Word;        {thời gian giữa các ký tự}
    end;
  {-------------------------------------------------}
```

Như vậy, cấu trúc DCB sẽ xác định cách thiết lập điều khiển cho thiết bị truyền tin nối tiếp.

Id: Thiết bị truyền tin. Giá trị Id được chính thiết bị gán vào. Nếu bit có trọng số cao nhất được lập thì cấu trúc DCB sẽ dùng cho thiết bị song song.

BaudRate: Nếu high-order byte= 0xFF, thì low-order byte = chỉ số tốc độ truyền tin. Chỉ số này có các giá trị:

CBR_110 CBR_14400

CBR_4400 CBR_19200

CBR_9200 CBR_38400

CBR_8400 CBR_56000

CBR_6000 CBR_128000

CBR_28000 CBR_256000

CBR_9600

Nếu high-order byte <> 0xFF, thì tham số đó chính là tốc độ thật của thiết bị truyền tin.

ByteSize: Số lượng bit/ ký tự:từ 4 đến 8.

Parity:

EVENPARITY	Even
MARKPARITY	Mark

NOPARITY No parity

ODDPARITY Odd

StopBits:

ONESTOPBIT 1 stop bit

ONE5STOPBITS 1.5 stop bits

TWOSTOPBITS 2 stop bits

RlsTimeout: là thời gian cực đại (ms) mà thiết bị có thể đợi tín hiệu RLSD (receive-line-signal-detect). RLSD có thể hiểu như là tín hiệu CD (carrier-detect).

CtsTimeout: là thời gian cực đại (ms) mà thiết bị có thể đợi tín hiệu CTS (clear-to-send).

DsrTimeout: là thời gian cực đại (ms) mà thiết bị có thể đợi tín hiệu DSR (data-set-ready).

fBinary: Binary mode. Trong mode này, tín hiệu EofChar được nhận diện ở đầu vào input và là dấu hiệu báo kết thúc dữ liệu.

fRtsDisable: Nếu được lập thì RTS=0 (thụ động). Nếu nó =0 thì RTS được sử dụng để yêu cầu thiết bị đã sẵn sàng.

fOutxCtsFlow: CTS được dùng làm cờ chỉ dẫn cho dòng dữ liệu truyền đi. Nếu nó được lập và CTS=0, output sẽ bị treo cho tới khi CTS =1.

fOutxDsrFlow: DSR được dùng làm cờ chỉ dẫn cho dòng dữ liệu truyền đi. Nếu nó được lập và DSR=0, output sẽ bị treo cho tới khi DSR =1.

fDtrDisable: Nếu được lập thì DTR =0 (thụ động). Nếu nó =0 thì DTR được sử dụng để yêu cầu thiết bị đã sẵn sàng.

fOutX: XON/XOFF được sử dụng khi truyền tin. Nếu nó được lập, truyền tin sẽ dừng khi nhận được ký tự XoffChar và khởi động lại khi nhận được ký tự XonChar.

fInX:	XON/XOFF được sử dụng khi thu tin. Nếu nó được lập, thì XonChar được gửi khi thu được XoffLim (báo đầy) và XonChar được gửi khi thu được XonLim (báo rỗng).
fPeChar:	Báo lỗi parity xuất hiện.
fNull:	Thu được các ký tự null - dấu hiệu hủy.
fChEvt:	eventflag xuất hiện.
fDtrflow:	Tín hiệu DTR (data-terminal-ready) được sử dụng cho việc kiểm soát luồng dữ liệu thu. Nếu chế độ này được lập, DTR =0 (off) khi luồng dữ liệu thu lấp đầy ký tự XoffLim và nó được gửi đi khi luồng dữ liệu thu có các ký tự XonLim rỗng.
fRtsflow:	Tín hiệu RTS (ready-to-send) được sử dụng cho việc kiểm soát luồng dữ liệu thu. Nếu chế độ này được lập, RTS=0 (off) khi luồng dữ liệu thu lấp đầy ký tự XoffLim và nó được gửi đi khi luồng dữ liệu thu có các ký tự XonLim rỗng.
fDummy2	Dự trữ
	XonChar Giá trị ký tự XON sử dụng cho cả thu và phát.
XoffChar	Giá trị ký tự XOFF sử dụng cho cả thu và phát.
XonLim	Số lượng ký tự cực tiểu cho phép trong hàng đợi thu trước khi ký tự XON được phát đi.
XoffLim	Số lượng ký tự cực đại cho phép trong phần thu trước khi ký tự XOFF được phát đi. Giá trị XoffLim bị trừ từ dung lương của phần thu theo byte để tính số lượng ký tự cực đại cho phép.
PeChar	Giá trị ký tự sử dụng để xấp đặt ký tự thu được có lỗi parity.
EofChar	Giá trị ký tự sử dụng để thông báo kết thúc dữ liệu.
EvtChar	Giá trị ký tự sử dụng để thông báo có sự kiện xuất hiện.
TxDelay	Không sử dụng.

Các hàm hệ thống được sử dụng trong driver này gồm:

- ***function WriteComm(Cid: Integer; Buf: PChar; Size: Integer): Integer;***

Hàm WriteComm ghi vào thiết bị truyền thông được chỉ định.Cid chỉ rõ thiết bị thu byte. Hàm OpenComm trả lại chính giá trị này.Buf trỏ tới bộ đệm chứa các byte đợc ghi.Size chỉ rõ số lượng byte được ghi.

Hàm trả về giá trị chỉ rõ số lượng byte được ghi nếu thực hiện thành công. Nếu không, nó trả về giá trị <0 khi gặp lỗi.

Để xác định dạng lỗi cần sử dụng hàm GetCommError để lấy lại trạng thái và giá trị lỗi.

Đối với cổng nối tiếp, hàm WriteComm xóa dữ liệu trong hàng đợi phát nếu ở đây không đủ chỗ cho hàng đợi của các byte bổ xung. Trước khi gọi hàm này, các ứng dụng cần kiểm tra không gian hàng đợi phát bằng hàm OpenComm để kiểm tra kích thước của hàng đợi phát không dược nhỏ hơn kích thước cực đại của xâu ký tự phát.

- ***function GetTickCount: LongInt;***

Hàm GetTickCount trả về số lượng mili giây trôi qua kể từ khi Window khởi động. Hàm GetTickCount được coi là của riêng hàm GetCurrentTime.

- ***function GetCurrentTime: LongInt;***

Hàm GetCurrentTime trả về số lượng mili giây trôi qua kể từ khi Window khởi động.

- ***function SetTimer(Wnd: HWnd; IDEvent: Integer; Elapse: Word; TimerFunc: TFarProc): Word;***

Hàm SetTimer khởi tạo đồng hồ hệ thống. Giá trị thời gian time-out được định rõ và mỗi lần time-out xuất hiện thì hệ thống gửi thông báo WM_TIMER tới hàng đợi hoặc tham chiếu thông báo tới hàm TimerProc.

Wnd nhận diện việc Window kết hợp với timer. IDEvent chỉ rõ đồng hồ khác không. Elapse chỉ rõ giá trị time-out value (miliSec).

TimerFunc chỉ rõ địa chỉ thủ tục của hàm gọi ngược callback để xử lý thông báo WM_TIMER. Hàm trả về thời gian cho đồng hồ. Thông báo chỉ thị khoảng thời gian timeout đã trôi qua. HWND là thẻ 16-bit của window.

- ***function KillTimer(Wnd: HWnd; IDEvent: Integer): Bool;***

The KillTimer loại bỏ timer được chỉ định. Các tam số giống như hàm SetTimer. IDEvent đặc tả việc đồng hồ bị loại bỏ. Nếu ứng dụng gọi SetTimer với tham số Wnd được đặt bằng zero, tham số này phải là thời gian. Nếu thực hiện thành công nó trả về giá trị khác không. Còn nếu nó bằng 0 nghĩa là hàm không tìm thấy timer này.

- ***function StrPCopy(Dest: PChar; Source: string): PChar;***

Hàm StrPCopy copy xâu ký tự Pascal vào vùng đệm đích Dest. StrPCopy không kiểm tra độ dài xâu. Vùng đệm đích Dest có dung lượng bằng Length(Source)+1.

Ví dụ:

```
uses SysUtils;
var   A: array[0..79] of Char;
begin
   S:= 'Đây là chương trình TEST hàm StrPCopy.';
   StrPCopy(A, S);
   Canvas.TextOut(10, 10, StrPas(A));
end;
```

- ***function CloseComm(Cid: Integer): Integer;***

Hàm CloseComm đóng thiết bị truyền thông và giải phóng bộ đệm thu và phát. Tất cả các ký tự ở hàng đợi phát được phát hết trước khi thiết bị bị đóng. Cid chỉ rõ thiết bị phải đóng (OpenComm cho giá trị Cid này). Hàm trả về 0 nếu nó thực hiện thành công.

- ***function AllocateHWnd(Method: TWndMethod): HWND;***

Hàm AllocateHWnd tạo cửa sổ ảo trong Windows và trả về thẻ cho cửa sổ ảo. AllocateHWnd cần cho component

- ***procedure DeallocateHWnd(Wnd: HWND);***

Hàm DeallocateHWnd thu hồi cửa sổ ảo trong Windows. Tham số Wnd là thẻ cho cửa sổ ảo được trả về bởi AllocateHWnd.

- ***property ComponentState: TComponentState;***

Thuộc tính ComponentState là thuộc tính chỉ đọc (read-only) thể hiện trạng thái hiện hành của component. Component sử dụng thuộc tính này để kiểm tra trạng thái mà ở đó nó cần để ngăn ngừa một kiểu tác động cụ thể nào đó. Xét kiểu này ta thấy:

```
type
TComponentState=set of (csLoading, csReading, csWriting,
                        csDestroying, csDesigning);
```

csDesigning	là mode thiết kế.
csDestroying component	sắp bị hủy.
csLoading	tải từ bộ lọc..
csReading	đọc giá trị thuộc tính từ luồng.
csWriting	ghi giá trị thuộc tính vào luồng.

- ***function BuildCommDCB(Def: PChar; var DCB: TDCB): Integer;***

Hàm BuildCommDCB truyền xâu ký tự được định nghĩa tới khối điều khiển thiết bị truyền nối tiếp tương ứng.

Def trỏ tới ký tự kết thúc đặc tả thông tin điều khiển thiết bị. Ký tự này phải có khuôn dạng giống như tham số dùng trong chế độ lệnh trên DOS. DCB trỏ tới cấu trúc TDCB là nơi nhận ký tự đã truyền đi. Cấu trúc này định nghĩa việc thiết lập cơ chế điều khiển cho thiết bị truyền tin.

Kết quả trả lại zero nếu thành công. Ngược lại, nó bằng -1.

Hàm BuildCommDCB có chức năng chỉ chất đầy bộ đệm bufer. Để lập cổng, các ứng dụng cần sử dụng hàm SetCommState.

Hàm BuildCommDCB ở chế độ mặc định không dùng chế độ bắt tay mềm và cứng. Để dùng chế độ này, ứng dụng phải đặt trong cấu trúc TDCB.

- ***function SetCommState(var DCB: TDCB): Integer;***

Hàm SetCommState đặt thiết bị truyền thông vào trạng thái đặc tả bởi DCB. DCB trỏ tới TDCB nơi chứa các thông số thiết lập cần thiết cho thiết bị. Thành phần ID của TDCB phải nhận diện được thiết bị.

Hàm trả lại zero nếu thành công. Ngược lại, nó sẽ nhỏ hơn 0.

Hàm này khởi tạo lại phần cứng và cơ cấu điều khiển như định nghĩa trong TDCB, nhưng nó không làm rỗng hàng đệm phát và thu.

- ***function GetCommState(Cid: Integer; var DCB: TDCB): Integer;***

Hàm GetCommState khôi phục lại nội dung DCB của thiết bị được chỉ ra. CID định rõ thiết bị phải được kiểm tra. Chức năng OpenComm sẽ trả lại giá trị này. DCB trỏ tới TDCB nơi nhận nội dung DCB hiện hành. TDCB định nghĩa các thông số điều khiển cho thiết bị. Hàm trả lại zero nếu thực hiện thành công. Ngược lại, nó sẽ nhỏ hơn zero.

- ***function GetCommError(Cid: Integer; var Stat: TComStat): Integer;***

Hàm GetCommError trả về giá trị hầu hết các lỗi và trạng thái hiện hành cho thiết bị được chỉ ra. Khi lỗi xuất hiện, Windows khóa cổng truyền thông cho tới khi GetCommError xóa được các lỗi.

Cid	định rõ thiết bị phải kiểm tra. Hàm OpenComm trả lại giá trị này. Stat trỏ tới TCOMSTAT là nơi nhận được trạng thái của thiết bị. CE_BREAK.
Hardware	phát hiện điều kiện ngừng (break).
CE_CTSTO	CTS (clear-to-send) quá thời gian.
CE_DNS	thiết bị song song không được lựa chọn.
CE_DSRTO	DSR (data-set-ready) quá thời gian.
CE_FRAME	phần cứng phát hiện lỗi khung.
CE_IOE	lỗi I/O khi truyền với thiết bị song song.
CE_MODE	mode yêu cầu không được hỗ trợ.
CE_OOP	thiết bị song song hết giấy.
CE_OVERRUN	lỗi chạy vượt.

CE_PTO	quá thời gian khi cố liên lạc với thiết bị song song.
CE_RLSDTO	quá thời gian RLSD (receive-line-signal-detect).
CE_RXOVER	hàng thu bị vượt quá.
CE_RXPARITY	lỗi parity.
CE_TXFULL	hàng đợi phát đầy.

- ***function OpenComm(ComName: PChar; InQueue, OutQueue: Word): Integer;***

Hàm OpenComm để mở thiết bị truyền thông. ComName trỏ tới xâu ký tự chỉ rõ thiết bị trong dạng COMn hay or LPTn, ở đây n là số hiệu thiết bị. InQueue chỉ rõ kích thước (bytes) của hàng đợi thu. OutQueue. chỉ rõ kích thước (bytes) của hàng đợi phát.

Hàm trả lại giá trị đặc trưng cho việc mở thiết bị thành công. Ngược lại, nó nhỏ hơn 0.

- ***function SetCommBreak(Cid: Integer): Integer;***

Hàm SetCommBreak đình chỉ ký tự phát và đặt vào đó trạng thái hủy (break). Kết quả là zero nếu thành công.

- ***function ClearCommBreak(Cid: Integer): Integer;***

Hàm này lưu ký tự phát đặt vào đó trạng thái bình thường (nonbreak).

- ***function CloseComm(Cid: Integer): Integer;***

Hàm CloseComm dùng để đóng thiết bị được chỉ ra. Các ký tự hàng đợi phát được phát hết trước khi thiết bị bị đóng. Cid chỉ rõ thiết bị phải đóng. Hàm trả về zero nếu thành công.

- ***function FlushComm(Cid, Queue: Integer): Integer;***

Hàm FlushComm xóa hết bộ đệm thu và phát của thiết bị. Cid chỉ rõ thiết bị bị xoá. Hàm OpenComm trả lại kết quả này.Queue chỉ rõ hàng đợi phải xoá. Nếu tham số này bằng 0, hàng đợi phát đã xoá. Nếu nó bằng 1 hàng đợi thu đã được xoá. Hàm trả về zero nếu thành công.

Kiểu TComStat có dạng:

```
TComStat = record
Flags: Byte;
(*
 Byte fCtsHold: 1; {phát dựa trên CTS hold }
 Byte fDsrHold: 1; {phát dựa trên DSR hold }
 Byte fRlsdHold: 1; {phát dựa trên RLSD hold }
 Byte fXoffHold: 1; {handshake thu }
 Byte fXoffSent: 1; {handshake phát }
 Byte fEof: 1; {tìm thấy ký tự Eof }
 Byte fTxim: 1; {ký tự đã được phát}
*)
 cbInQue: Word; {đếm ký tự trong hàng đợi thu}

 cbOutQue: Word; {đếm ký tự trong hàng đợi phát}
 end;
```

The TCOMSTAT structure contains information about a communications device.

Cờ CSTF_CTSHOLD	chỉ rõ việc phát phải chờ tín hiệu CTS được gửi đi.
Cờ F_DSRHOLD	chỉ rõ việc phát phải chờ tín hiệu DSR được gửi đi.
Cờ CSTF_RLSDHOLD	chỉ rõ việc phát phải chờ tín hiệu RLSD (receive-line-signal-detect) được gửi đi.
Cờ CSTF_XOFFHOLD	Specifies whether transmission is waiting as a result of the XOFF character being received.
CSTF_XOFFSENT	Specifies whether transmission is waiting as a result of the XOFF character being transmitted. Transmission halts when the XOFF character is transmitted and

	used by systems that take the next character as XON, regardless of the actual character.
CSTF_EOF	Specifies whether the end-of-file (EOF) character has been received.
CSTF_TXIM	Specifies whether a character is waiting to be transmitted.
cbInQue	Specifies the number of characters in the receive queue.
cbOutQue	Specifies the number of characters in the transmit queue.

Trước hết ta xây dựng chương trình Driver điều khiển cho cổng COM của PC. Khi ghép thành công vào hệ điều hành của máy tính thì xây dựng tiếp chương trình điều khiển truyênf tin qua MODEM. Trong tài nguyên của hệ thống, ta sử dụng các unit WinTypes, WinProcs, Messages, sysUtils,Classes, Forms...;

Unit SysUtils khai báo các lớp và dịch vụ ngoại lệ: classes, string, date, time và dịch vụ tiện ích. Unit ExtCtrls khai báo component trên trang chuẩn và trang bổ xung của bảng màu Component. Unit StdCtrls khai báo sự xuất hiện components trên trang chuẩn của bảng màu Component mà các objects, types và constants kết hợp.

```
unit ComDrv16;
interface
uses
 WinTypes, WinProcs, Messages, SysUtils, Classes, Forms;
type {Định nghĩa các kiểu thông số cho cổng COM}
 {COM Port Baud Rates}
 TComPortBaudRate = (br110, br300, br600, br1200, br2400,
   br4800, br9600, br14400, br19200, br38400, br57600);
 {COM Port Numbers}
 TComPortNumber = (pnCOM1, pnCOM2,
         pnCOM3, pnCOM4);
```

```
{COM Port Data bits}
TComPortDataBits = (db5BITS, db6BITS, db7BITS, db8BITS);
{COM Port Stop bits}
TComPortStopBits = (sb1BITS, sb1HALFBITS, sb2BITS);
{COM Port Parity}
TComPortParity = (ptNONE, ptODD, ptEVEN,
        ptMARK, ptSPACE);
{COM Port Hardware Handshaking}
TComPortHwHandshaking = (hhNONE, hhRTSCTS);
{COM Port Software Handshaing}
TComPortSwHandshaking = (shNONE, shXONXOFF);

TComPortReceiveDataEvent = procedure(Sender:
    TObject; DataPtr: pointer; DataSize: integer) of object;

TCommPortDriver = class(TComponent)
protected
FComPortID: integer; {COM Port Device ID - 0..x}

FComPort: TComPortNumber;
          {cổng COM được sử dụng (1..4)}
FComPortBaudRate: TComPortBaudRate;
    {COM Port speed (brXXXX)}
FComPortDataBits: TComPortDataBits;
          {số lượng bit dữ liệu (5..8)}
FComPortStopBits: TComPortStopBits;
        {số lượng bit stop(1,1.5,2)}
FComPortParity: TComPortParity;
      {kiểu parity (none,odd,even,mark,space)}
FComPortHwHandshaking: TComPortHwHandshaking;
      {kiểu handshaking Hw}
FComPortSwHandshaking: TComPortSwHandshaking;
        {kiểu handshaking sw}
```

```
FComPortInBufSize: word; {dung lượng buffer vào}
FComPortOutBufSize: word; {dung lượng buffer ra}
FComPortReceiveData: TComPortReceiveDataEvent;
          {sự kiện xuất hiện dữ liệu thu}
FComPortPollingDelay: word; {ms of delay between
            COM port pollings}
FEnableDTROnOpen: boolean; {cho phép/cấm kết
              nối đường DTR}
FOutputTimeout: word; {output timeout - millisec}
FNotifyWnd: HWND; {sử dụng cho timer}
FTempInBuffer: pointer;
{các phương thức xử lý dưới dạng hàm và thủ tục}
procedure SetComPortID(Value: integer);
procedure SetComPort(Value: TComPortNumber);
procedure SetComPortBaudRate(Value: TComPortBaudRate);
procedure SetComPortDataBits(Value: TComPortDataBits);
procedure SetComPortStopBits(Value: TComPortStopBits);
procedure SetComPortParity(Value: TComPortParity);
procedure SetComPortHwHandshaking(Value:
        TComPortHwHandshaking);
procedure SetComPortSwHandshaking(Value:
        TComPortSwHandshaking);
procedure SetComPortInBufSize(Value: word);
procedure SetComPortOutBufSize(Value: word);
procedure SetComPortPollingDelay(Value: word);
procedure ApplyCOMSettings;
procedure TimerWndProc(var msg: TMessage);

public
constructor Create(AOwner: TComponent); override;
destructor Destroy; override; {chỉ dẫn override được sử dụng
            để định nghĩa lại phương thức}
```

```
 function Connect: boolean; {hàm kết nối đúng}
 procedure Disconnect; {thủ tục hủy kết nối}
 function Connected: boolean; {hàm đã kết nối}
 procedure FlushBuffers(inBuf, outBuf: boolean);
 {giải phóng output buffer}
 function OutFreeSpace: word;

 {phát data}
 function SendData(DataPtr: pointer; DataSize: integer):
integer;
 {phát ký tự}
 function SendString(s: string): boolean;
 function SendZString(s: pchar): boolean;
          {đặt đường DTR lên cao (onOff=TRUE)
           hoặc xuống thấp(onOff=FALSE)}
 procedure ToggleDTR(onOff: boolean);
          {đặt đường RTS lên cao (onOff=TRUE)
           hoặc xuống thấp(onOff=FALSE)}
 procedure ToggleRTS(onOff: boolean);
 property ComPortID: integer read FComPortID write
SetComPortID;
 published
 {gán cổng COM}
 property ComPort: TComPortNumber read FComPort
          write SetComPort default pnCOM1;
 {gán tốc độ cho cổng}
 property ComPortSpeed: TComPortBaudRate
          read FComPortBaudRate
          write SetComPortBaudRate default br9600;
 {số bits dữ liệu (5..8, cho chip 8250)}
 property ComPortDataBits: TComPortDataBits
        read FComPortDataBits
        write SetComPortDataBits default db8BITS;
```

```
{số bits Stop là (1, 1.5, 2)}
property ComPortStopBits: TComPortStopBits
        read FComPortStopBits
        write SetComPortStopBits default sb1BITS;
{bit Parity là (none, odd, even, mark, space)}
property ComPortParity: TComPortParity
        read FComPortParity
        write SetComPortParity default ptNONE;
{sử dụng kiểu Hardware Handshaking:
cdNONE no handshaking
cdCTSRTS cả cdCTS và cdRTS
(** thường là sử dụng phương pháp này**)}
property ComPortHwHandshaking: TComPortHwHandshaking
   read FComPortHwHandshaking
     write SetComPortHwHandshaking default hhNONE;
{sử dụng kiểu Software Handshaking:
cdNONE no handshaking
cdXONXOFF XON/XOFF handshaking}
property ComPortSwHandshaking: TComPortSwHandshaking
   read FComPortSwHandshaking
     write SetComPortSwHandshaking default shNONE;
{kích thước bộ đệm vào -Buffer}
property ComPortInBufSize: word read FComPortInBufSize
      write SetComPortInBufSize default 2048;
{kích thước bộ đệm ra- Buffer}
property ComPortOutBufSize: word read FComPortOutBufSize
       write SetComPortOutBufSize default 2048;
{giữ chậm (ms) giữa các vòng kiểm tra}
property ComPortPollingDelay: word read FComPortPollingDelay
         write SetComPortPollingDelay default 50;
  {Đặt TRUE để cho phép kết nối đường DTR
  và hủy khi chưa kết nối. Đặt FALSE để
  cấm kết nối đường DTR.}
```

```
 property EnableDTROnOpen: boolean
       read FEnableDTROnOpen
       write FEnableDTROnOpen default true;
 {Output timeout (ms)}
 property OutputTimeout: word read FOutputTimeOut
        write FOutputTimeout default 4000;
 {sự kiện có dữ liệu vào}
 property OnReceiveData: TComPortReceiveDataEvent
          read FComPortReceiveData
          write FComPortReceiveData;
 end;

procedure Register;

implementation      {phần công cụ của unit sử dụng các
procedures
 và functions đã khai báo trong phần interface}
constructor TCommPortDriver.Create(AOwner: TComponent);
     {constructor định nghĩa các hiệu ứng liên kết với
      các đối tượng được tạo ra. Nó được khai báo dưới
      dạng từ khóa constructor}
begin
 inherited Create(AOwner);
 {khởi tạo các giá trị mặc định}
 FComPortID:= -1; {chưa kết nối}
 FComPort:= pnCOM2; {COM 2}
 FComPortBaudRate:= br9600; {9600 bauds}
 FComPortDataBits:= db8BITS; {8 bits dữ liệu}
 FComPortStopBits:= sb1BITS; {1 bit stop}
 FComPortParity:= ptNONE; {không parity}
 FComPortHwHandshaking:= hhNONE;
  {không sử dụng chế độ bắt tay cứng hardware handshaking}
 FComPortSwHandshaking:= shNONE;
```

```
   {không sử dụng chế độ bắt tay mềm software handshaking}
  FComPortInBufSize:= 2048;
         {bộ đệm input buffer = 2048 bytes}
  FComPortOutBufSize:= 2048;
         {bộ đệm output buffer = 2048 bytes}
  FComPortReceiveData:= nil; {no data handler}
  FComPortPollingDelay:= 50;
           {poll COM port every 50ms}
  FOutputTimeout:= 4000; {output timeout - 4000ms}
  FEnableDTROnOpen:= true; {DTR cao khi kết nối}
         {Tổ chức bộ đệm cho dữ liệu thu}
  GetMem(FTempInBuffer, FComPortInBufSize);
      {tạo cửa sổ quản lý để nắm bắt các thông báo thời gian}
  if not (csDesigning in ComponentState) then
  FNotifyWnd:= AllocateHWnd(TimerWndProc);

end;

Destructor TCommPortDriver.Destroy;
begin
  {chắc chắn rằng COM device đã thoát}
  Disconnect;
  {giải phóng vùng buffer đệm}
  FreeMem(FTempInBuffer, FComPortInBufSize);
  {hủy bỏ timer's window}
  DeallocateHWnd(FNotifyWnd);
  inherited Destroy;
end;
   {COM port ID được dùng chung. Điều này cho phép
    kết nối với các cổng ngoài. ComPortID = -1--->hủy}
procedure TCommPortDriver.SetComPortID(Value: integer);
begin
  {nếu cùng cổng COM thì không làm gì}
```

```
 if FComPortID = Value then
 exit;
       {nếu = 65535 thì dừng kiểm tra cổng COM
        nhưng không đóng nó}
 if Value = 65535 then
 begin
 if Connected then
 {dừng đồng hồ}
 if Connected then
 KillTimer(FNotifyWnd, 1);
 FComPortID:= -1;
 end
 else
 begin
 {hủy liên kết}
 Disconnect;
 {nếu < 0 thì thoát}
 {(ComPortID < 0 ---->hủy)}
 if Value < 0 then
 exit;

 {Set COM port ID}
 FComPortID:= Value;

 {khởi động đồng hồ (sử dụng hỏi vòng)}
 SetTimer(FNotifyWnd, 1, FComPortPollingDelay, nil);
 end;
end;
procedure TCommPortDriver.SetComPort(Value:
             TComPortNumber);
begin
 {chắc chắn rằng không sử dụng bất kỳ cổng COM nào}
 if Connected then
```

```
 exit;
 {đổi cổng COM}
 FComPort:= Value;
end;
procedure TCommPortDriver.SetComPortBaudRate(Value:
              TComPortBaudRate);
begin
 {đặt tốc độ mới cho cổng COM}
 FComPortBaudRate:= Value;
 if Connected then
 ApplyCOMSettings;
end;
procedure TCommPortDriver.SetComPortDataBits(Value:
                TComPortDataBits);
begin
 {đặt data bits}
 FComPortDataBits:= Value;
 if Connected then
 ApplyCOMSettings;
end;
procedure TCommPortDriver.SetComPortStopBits(Value:
                TComPortStopBits);
begin
 {đặt lại số lượng stop bit}
 FComPortStopBits:= Value;
 if Connected then
 ApplyCOMSettings;
end;
procedure TCommPortDriver.SetComPortParity(Value:
                TComPortParity);
begin
 { đặt lại số lượng bit parity}
 FComPortParity:= Value;
```

```
 if Connected then
 ApplyCOMSettings;
end;

procedure TCommPortDriver.SetComPortHwHandshaking(Value:
                    TComPortHwHandshaking);
begin
 {đặt lại chế độ hardware handshaking}
 FComPortHwHandshaking:= Value;
 if Connected then
 ApplyCOMSettings;
end;
procedure TCommPortDriver.SetComPortSwHandshaking(Value:
TComPortSwHandshaking);
begin
 {đặt lại chế độ software handshaking}
 FComPortSwHandshaking:= Value;
 {đặt lại giá trị thay đổi}
 if Connected then
 ApplyCOMSettings;
end;
procedure TCommPortDriver.SetComPortInBufSize(Value: word);
begin
 {không làm gì nếu đã kết nối}
 if Connected then
 exit;
 {giải phóng bộ đệm vào buffer}
 FreeMem(FTempInBuffer, FComPortInBufSize);
 {đặt lại kích thước bộ đệm vào}
 FComPortInBufSize:= Value;
 {tổ chức bộ đệm vào}
 GetMem(FTempInBuffer, FComPortInBufSize);
end;
```

```
procedure TCommPortDriver.SetComPortOutBufSize(Value: word);
begin
 {không làm gì nếu đã kết nối}
 if not Connected then
 exit;
 {đặt kích thước bộ đệm ra}
 FComPortOutBufSize:= Value;
end;
procedure TCommPortDriver.SetComPortPollingDelay(Value:
word);
begin
 {nếu thời gian giữ chậm không bằng giá trị cũ...}
 if Value <> FComPortPollingDelay then
 begin
 {dừng timer}
 if Connected then
 KillTimer(FNotifyWnd, 1);
 {lưu giá trị giữ chậm mới}
 FComPortPollingDelay:= Value;
 {khởi động lại timer}
 if Connected then
 SetTimer(FNotifyWnd, 1, FComPortPollingDelay, nil);
 end;
end;
const
 Win16BaudRates: array[br110..br115200] of longint =
 (110, 300, 600, 1200, 2400, 4800,
 9600, 14400, 19200, 38400, 57600,
 57600+1{=115200bps});

{Đặt lại cổng COM}
procedure TCommPortDriver.ApplyCOMSettings;
var dcb: TDCB;
```

```
begin
 {không làm gì nếu đã kết nối}
 if not Connected then
 exit;

 {lấy trạng thái hiện hành}
 GetCommState(FComPortID, dcb);
 dcb.BaudRate:= Win16BaudRates[FComPortBaudRate];
     {tốc độ}
 dcb.Flags:= dcb_Binary;
    {cho phép mode truyền (cấm kiểm tra EOF).}

 if not EnableDTROnOpen then
 {cấm DTR line khi device hở}
 dcb.Flags:= dcb.Flags or dcb_DtrDisable;

 case FComPortHwHandshaking of {kiểu hw handshaking}
 hhNONE:; {không dùng hardware handshaking}
 hhRTSCTS:
      {RTS/CTS (request-to-send/clear-to-send)-
      kiểu hardware handshaking}
 dcb.Flags:= dcb.Flags or dcb_OutxCtsFlow or dcb_Rtsflow;
 end;
 case FComPortSwHandshaking of {kiểu sw handshaking}
 shNONE:; {không dùng software handshaking}
 shXONXOFF: {XON/XOFF handshaking}
 dcb.Flags:= dcb.Flags or dcb_OutX or dcb_InX;
 end;
 dcb.XONLim:= FComPortInBufSize div 4;
        {ấn định số byte tối thiểu được chứa
         trong input buffer trước khi gửi ký tự
         XON (hoặc đặt CTS=cao)}
 dcb.XOFFLim:= 1; {ấn định số byte tối đa được chứa
```

```
        trong input buffer trước khi gửi ký tự
        XOFF (hoặc đặt CTS=thấp)}
 dcb.ByteSize:= 5 + ord(FComPortDataBits);
           {data bits ?}
 dcb.Parity:= ord(FComPortParity);    {kiểu parity}
 dcb.StopBits:= ord(FComPortStopbits);
      {stop bits ?}
 dcb.XONChar:= #17; {XON ASCII - DC1, Ctrl-Q, ASCII 17}
 dcb.XOFFChar:= #19; {XOFF ASCII - DC3, Ctrl-S, ASCII 19}

 {Đặt mới}
 SetCommState(dcb);
 {làm rỗng buffers}
 FlushBuffers(true, true);
end;
function TCommPortDriver.Connect: boolean;
var comName: array[0..4] of char;
begin
 {không làm gì khi đã kết nối}
 Result:= Connected;
 if Result then
 exit;
 {mở cổng COM}
 StrPCopy(comName, 'COM');
 comName[3]:= chr(ord('1') + ord(FComPort));
 comName[4]:= #0;
 FComPortID:= OpenComm(comName,
     FComPortInBufSize, FComPortOutBufSize);
 Result:= Connected;
 if not Result then
 exit;
 {Đặt giá trị cho COM}
 ApplyCOMSettings;
```

```
 {khởi động timer (cho kiểm tra vòng)}
 SetTimer(FNotifyWnd, 1, FComPortPollingDelay, nil);
end;
procedure TCommPortDriver.Disconnect;
begin
 if Connected then
 begin
 {dừng timer (cho kiểm tra vòng)}
 KillTimer(FNotifyWnd, 1);
 {giải phóng cổng COM}
 CloseComm(FComPortID);
 {không kết nối nữa}
 FComPortID:= -1;
 end;
end;
function TCommPortDriver.Connected: boolean;
begin
 {COM port IDs bắt đầu từ 0.
 Thay đổi từ "FComPortID > 0"
   tới "FComPortID >= 0")}
 Result:= FComPortID >= 0;
end;
procedure TCommPortDriver.FlushBuffers(inBuf,
            outBuf: boolean);
begin
 if not Connected then
 exit;
 {làm rỗng buffer dữ liệu tới}
 if outBuf then
 FlushComm(FComPortID, 0);
 if inBuf then
 FlushComm(FComPortID, 1);
end;
```

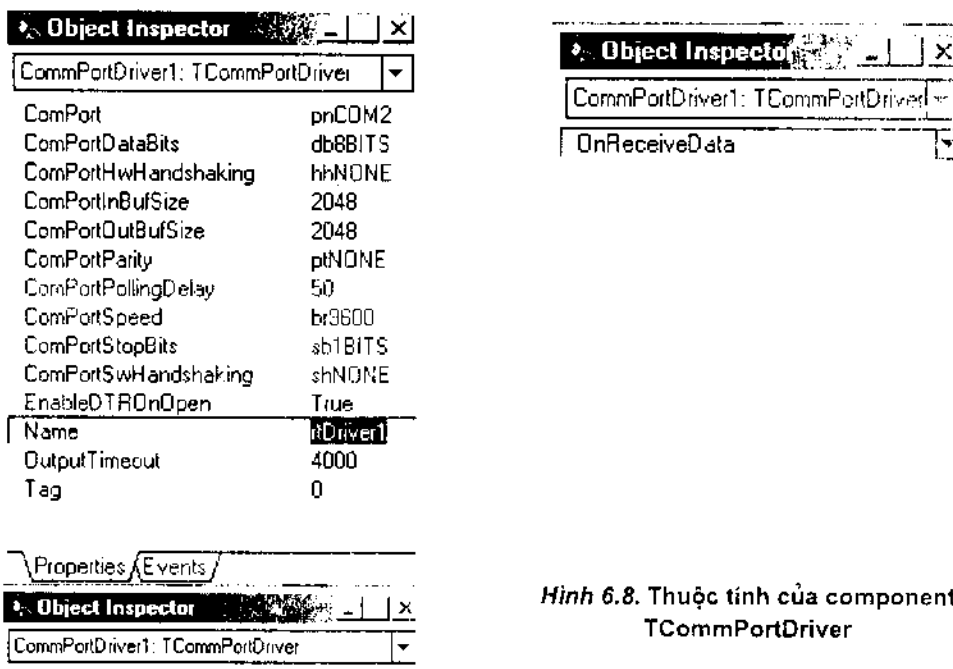
```
     {trả lại bộ đệm ra 65535 byte được giải phóng
      nếu không kết nối}
function TCommPortDriver.OutFreeSpace: word;
var stat: TCOMSTAT;
begin
 if not Connected then
 Result:= 65535
 else
 begin
 GetCommError(FComPortID, stat);
 Result:= FComPortOutBufSize - stat.cbOutQue;
 end;
end;
       {gửi data (nếu các gói dữ liệu không
        hợp lệ cho bộ đệm ra thì hủy)}
function TCommPortDriver.SendData(DataPtr: pointer; DataSize:
integer): integer;
var nToSend, nsent: integer;
 t1: longint;
begin
 {0 bytes sent}
 Result:= 0;
 {không làm gì khi chưa kết nối}
 if not Connected then
 exit;
 {thời gian hiện hành}
 t1:= GetTickCount;
      {lặp cho tới khi tất cả dữ liệu
      đã phát hết hoặc xuất hiện timeout}
 while DataSize > 0 do
 begin
 {lấy vùng tự do của bộ đệm ra}
```

```
 nToSend:= OutFreeSpace;
 {nếu bộ đệm ra còn vùng rỗng...}
 if nToSend > 0 then
 begin
 {không gửi thêm}
 if nToSend > DataSize then
 nToSend:= DataSize;
 {phát}
 nsent:= WriteComm(FComPortID, DataPtr, nToSend);
 {cập nhật số lượng byte đã phát đi}
 Result:= Result + abs(nsent);
 {giảm số byte phát}
 DataSize:= DataSize - abs(nsent);
 {lấy thời gian hiện hành}
 t1:= GetTickCount;
 {Tiếp tục. kiểm tra thời gian còn không? (không
  dừng phát nếu FOutputTimeout ở mức thấp thấp)}
 continue;
 end;
 {bộ đệm đầy. Nếu chờ quá lâu thì thoát ra}
 if (GetTickCount-t1) > FOutputTimeout then
 begin
 Result:= -Result;
 exit;
 end;
 end;
end;
{phát ký tự pascal}
function TCommPortDriver.SendString(s: string): boolean;
var len: integer;
begin
 len:= length(s);
 Result:= SendData(pchar(@s[1]), len) = len;
```

```
end;
function TCommPortDriver.SendZString(s: pchar): boolean;
var len: integer;
begin
 len:= strlen(s);
 Result:= SendData(s, len) = len;
end;
{đặt DTR line=cao (onOff=TRUE) hay thấp (onOff=FALSE).
 không được dùng HW handshaking.}
procedure TCommPortDriver.ToggleDTR(onOff: boolean);
 const funcs: array[boolean] of integer = (CLRDTR,SETDTR);
begin
 if Connected then
 EscapeCommFunction(FComPortID, funcs[onOff]);
end;
{đặt RTS line =cao (onOff=TRUE) hay(onOff=FALSE).
 không được dùng HW handshaking.}
procedure TCommPortDriver.ToggleRTS(onOff: boolean);
 const funcs: array[boolean] of integer = (CLRRTS,SETRTS);
begin
 if Connected then
 EscapeCommFunction(FComPortID, funcs[onOff]);
end;

{thủ tục polling cho cổng COM}
procedure TCommPortDriver.TimerWndProc(var msg: TMessage);
 var nRead: longint;
begin
 if (msg.Msg = WM_TIMER) and Connected then
 begin
 {đọc khối FComPortInBufSize bytes. nRead để nhận số lượng
thực}
```

```
 nRead:= ReadComm(FComPortID, FTempInBuffer,
FComPortInBufSize);
 {nếu dữ liệu sai thì gọi quản trị}
 if abs(nRead)>0 then
 if Assigned(FComPortReceiveData) then
 FComPortReceiveData(Self, FTempInBuffer, abs(nRead));
 end;
end;
procedure Register;
begin
 {ghi lại component này và chỉ rõ trong lớp 'System'
 của hệ thống}
 RegisterComponents('System', [TCommPortDriver]);
end;
end.
```



Hình 6.8. Thuộc tính của component TCommPortDriver

Để cài đặt Driver này phải vào Menu OPTION rồi vào Menu con INSTALL COMPONENTS và theo các trình tự các hướng dẫn để cài

component TCommPortDriver vào lớp SYSTEM của hệ thống. Sau khi cài đặt xong, một biểu tượng RS232 sẽ có mặt trong lớp này và khi viết chương trình điều khiển sẽ lấy nó vào như lấy các component khác. Thuộc tính của TCommPortDriver được thể hiện như hình 6,8.

6.3.2. Xây dựng chương trình điều khiển MODEM

Bây giờ chúng ta xây dựng chương trình điều khiển modem cũng dưới dạng một unit - unit MainFrm. Trong chương trình này sử dụng chương trình điều khiển driver TCommPortDriver là Unit COMdrv16 nên trong mục USES phải ghép thêm COMdrv16 vào.

```
unit MainFrm;
interface
uses
 WinTypes, WinProcs, Messages, SysUtils, Classes, Graphics,
 Controls, Forms, Dialogs, ExtCtrls, StdCtrls, ComDrv16,
 Buttons, Menus;
type  {khai báo kiểu}
 TMainForm = class(TForm) {TMainForm là một lớp của TForm}
 CommPortDriver: TCommPortDriver;
 TopPanel: TPanel;
 TxMemo: TMemo;
 ConnectBtn: TButton;
 DisconnectBtn: TButton;
 QuitBtn: TButton;
 ComPortRG: TRadioGroup; {tổ hợp nút kiểu dảo mạch}
 BaudRateRG: TRadioGroup; {tổ hợp nút kiểu dảo mạch}
 DataBitsRG: TRadioGroup;
 ParityRG: TRadioGroup;
 HandshakingRG: TRadioGroup;{tổ hợp nút kiểu dảo mạch}
 RxMemo: TMemo;
 Panel2: TPanel;
 Panel3: TPanel;
 StatusBar: TPanel;
```

```
    ClrRxBtn: TSpeedButton;
    ClrTxBtn: TSpeedButton;
    SetDTRBtn: TSpeedButton;
    ClearDTRBtn: TSpeedButton;
    SetRTSBtn: TSpeedButton;
    SpeedButton2: TSpeedButton;
    {bắt đầu các trình phương thức của TMainForm}
    procedure QuitBtnClick(Sender: TObject);
    procedure TxMemoKeyPress(Sender: TObject; var Key: Char);
    procedure CommPortDriverReceiveData
      (Sender: TObject; DataPtr: Pointer; DataSize: Integer);
    procedure ConnectBtnClick(Sender: TObject);
    procedure DisconnectBtnClick(Sender: TObject);
    procedure BaudRateRGClick(Sender: TObject);
    procedure DataBitsRGClick(Sender: TObject);
    procedure ParityRGClick(Sender: TObject);
    procedure HandshakingRGClick(Sender: TObject);
    procedure ComPortRGClick(Sender: TObject);
    procedure ClrRxBtnClick(Sender: TObject);
    procedure ClrTxBtnClick(Sender: TObject);
    procedure SetDTRBtnClick(Sender: TObject);
    procedure ClearDTRBtnClick(Sender: TObject);
    procedure SetRTSBtnClick(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure THot1Click(Sender: TObject);
    private {thủ tục riêng}
    procedure ApplyCommSettings;
    public
    end;
var
    MainForm: TMainForm;
    {MainForm là biến kiểu TMainForm}
implementation
```

```
{$R *.DFM}
  {Chỉ ra tên file nguồn cần ghép vào ứng dụng hay vào thư
viện}
procedure TMainForm.QuitBtnClick(Sender: TObject);
          {Thủ tục đặt phím thoát}
begin
 Close;
end;
procedure TMainForm.TxMemoKeyPress(Sender:
               TObject; var Key: Char);
{Thủ tục phát data khi có phím được ấn}
var s: string;
begin
 if CommPortDriver.Connected then
 {nếu đã kết nối thì phát data khi có phím được ấn}
 case Key of
 #13: if TxMemo.Lines.Count>0 then {CR bộ đếm}
 begin
 s:= TxMemo.Lines[TxMemo.Lines.Count-1];
 CommPortDriver.SendData(pchar(@s[1]), length(s));
 CommPortDriver.SendData(@Key, 1);
 end
 else CommPortDriver.SendData(@Key, 1);
 end;
end;
procedure TMainForm.CommPortDriverReceiveData(Sender:
TObject;
 DataPtr: Pointer; DataSize: Integer);
var p: pchar;
 s: string;
begin
 if RxMemo.Lines.Count > 0 then
 s: = RxMemo.Lines[RxMemo.Lines.Count-1]
```

```
 else
 s: = '';
 p: = DataPtr;
 while DataSize > 0 do
 begin
 case p^ of
 #10:; {về đầu dòng LF}
 #13: {xuống dòng CR}
 begin
 if RxMemo.Lines.Count <> 0 then
 RxMemo.Lines[RxMemo.Lines.Count-1]:= s
 else
 RxMemo.Lines.Add(s);
 RxMemo.Lines.Add('');
 s:= '';
 end;
 #8:            {là phím Backspace}
 delete(s, length(s), 1);
 else {ký tự bất kỳ}
 s: = s + p^;
 end;
 dec(DataSize);
 inc(p);
 end;
 if (s<>'') then
 if RxMemo.Lines.Count > 0 then
 RxMemo.Lines[RxMemo.Lines.Count-1]: = s
 else
 RxMemo.Lines.Add(s);
end;
procedure TMainForm.ConnectBtnClick(Sender: TObject);
begin
 {Đặt lại}
```

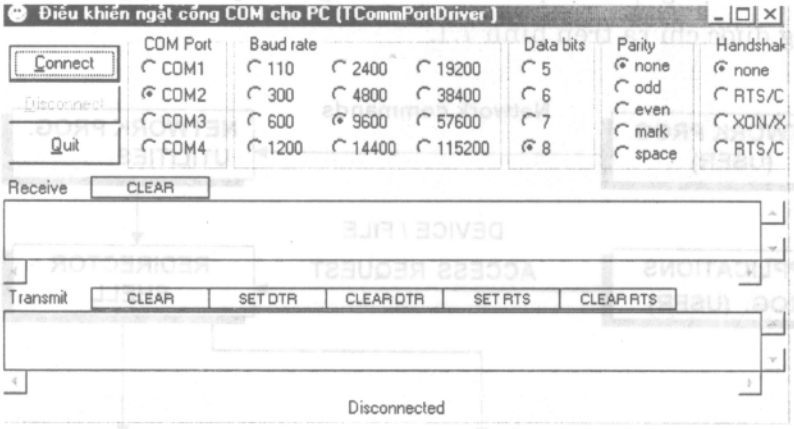
```
 ApplyCommSettings;
 {kết nối}
 if CommPortDriver.Connect then
 begin
 ConnectBtn.Enabled:= false; {cấm ấn nút ConnectBtn do đã
nối}
 DisconnectBtn.Enabled:= true; {cho phép ấn nút DisconnectBtn
              để sẵn sàng hủy kết nối}
 TxMemo.SetFocus;
 StatusBar.Caption:= 'Connected';
 end
 else            {Lỗi !}
 StatusBar.Caption:= 'Error: could not connect.
     Check COM port settings and try again.';
end;
procedure TMainForm.DisconnectBtnClick(Sender: TObject);
begin
 StatusBar.Caption:= 'Disconnected';
 CommPortDriver.Disconnect;
 DisconnectBtn.Enabled:= false;
 ConnectBtn.Enabled:= true;
end;
procedure TMainForm.BaudRateRGClick(Sender: TObject);
begin
 CommPortDriver.ComPortSpeed:=
   TComPortBaudRate(BaudRateRG.ItemIndex);
end;
procedure TMainForm.DataBitsRGClick(Sender: TObject);
begin
   CommPortDriver.ComPortDataBits:=
   TComPortDataBits(DataBitsRG.ItemIndex);
end;
procedure TMainForm.ParityRGClick(Sender: TObject);
```

```
begin
 CommPortDriver.ComPortParity:= TComPortParity
            (ParityRG.ItemIndex);
end;
procedure TMainForm.HandshakingRGClick(Sender: TObject);
begin
 case HandshakingRG.ItemIndex of
 0: {không}
 begin
 CommPortDriver.ComPortHwHandshaking:= hhNone;
 CommPortDriver.ComPortSwHandshaking:= shNone;
 end;
 1: {là RTS/CTS}
 begin
 CommPortDriver.ComPortHwHandshaking:= hhRTSCTS;
 CommPortDriver.ComPortSwHandshaking:= shNone;
 end;
 2: {là XON/XOFF}
 begin
 CommPortDriver.ComPortHwHandshaking:= hhNone;
 CommPortDriver.ComPortSwHandshaking:= shXONXOFF;
 end;
 3: {là RTS/CTS + XON/XOFF}
 begin
 CommPortDriver.ComPortHwHandshaking:= hhRTSCTS;
 CommPortDriver.ComPortSwHandshaking:= shXONXOFF;
 end;
 end;
end;
procedure TMainForm.ComPortRGClick(Sender: TObject);
begin
 {Đặt lại}
end;
procedure TMainForm.ApplyCommSettings;
var wasConnected: boolean;
```

```
begin
 wasConnected:= CommPortDriver.Connected;
 {nếu liên lạc đã kết nối thì....}
 if wasConnected then
 DisconnectBtnClick(nil);
  CommPortDriver.ComPort:=
  TComPortNumber(ord(ComPortRG.ItemIndex));
 BaudRateRGClick(nil);
 DataBitsRGClick(nil);
 ParityRGClick(nil);
 HandshakingRGClick(nil);
 {Kết nối liên lạc lại}
 if wasConnected then
 ConnectBtnClick(nil);
end;
procedure TMainForm.ClrRxBtnClick(Sender: TObject);
begin
 RxMemo.Lines.Clear;
end;
procedure TMainForm.ClrTxBtnClick(Sender: TObject);
begin
 TxMemo.Lines.Clear;
end;
procedure TMainForm.SetDTRBtnClick(Sender: TObject);
begin
 CommPortDriver.ToggleDTR(true);
end;
procedure TMainForm.ClearDTRBtnClick(Sender: TObject);
begin
 CommPortDriver.ToggleDTR(false); {chuyển đổi trạng thái DTR}
end;
procedure TMainForm.SetRTSBtnClick(Sender: TObject);
begin
 CommPortDriver.ToggleRTS(true); {chuyển đổi trạng thái RTS}
end;
```

```
procedure TMainForm.SpeedButton2Click(Sender: TObject);
begin
 CommPortDriver.ToggleRTS(false); {chuyển đổi trạng thái RTS}
end;
procedure TMainForm.THot1Click(Sender: TObject);
begin
  close;
end;
END.
```

Khi chương trình này chạy, một cửa sổ giao diện được mở ra (hình 6.9). Các Menu chức năng cơ bản được thể hiện trên thanh dụng cụ: connecting, connected, number, disconnect, quit, seting nằm trên thanh dụng cụ phía trên. Cửa sổ con phía trên là cửa sổ thu tin, cửa sổ con phía dưới là cửa phát tin. Mỗi khi tác động vào một nút thì thủ tục tương ứng với chức năng của tên nút đó sẽ được kích hoạt và đi vào thao tác.

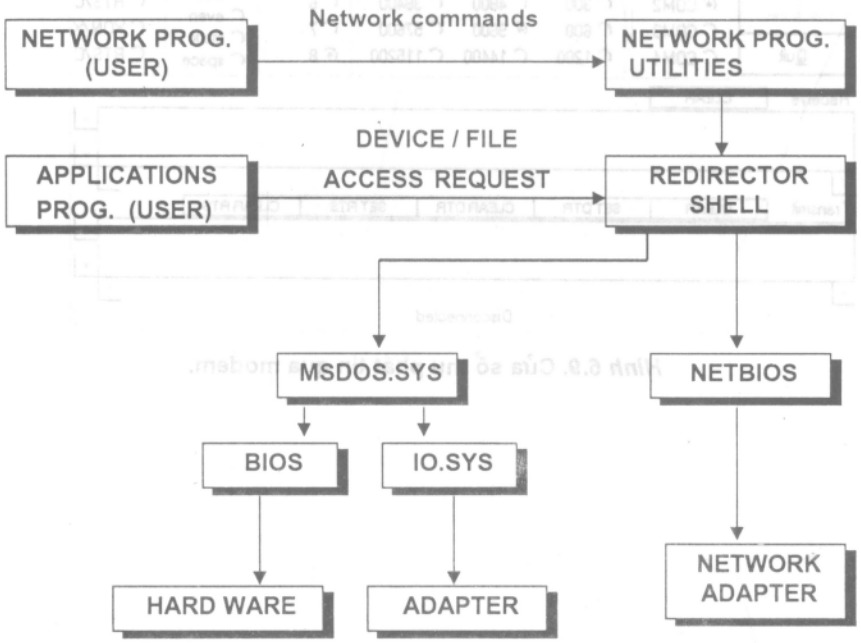


Hình 6.9. **Cửa sổ thu phát tin qua modem.**

CHƯƠNG 7
LẬP TRÌNH NETBIOS

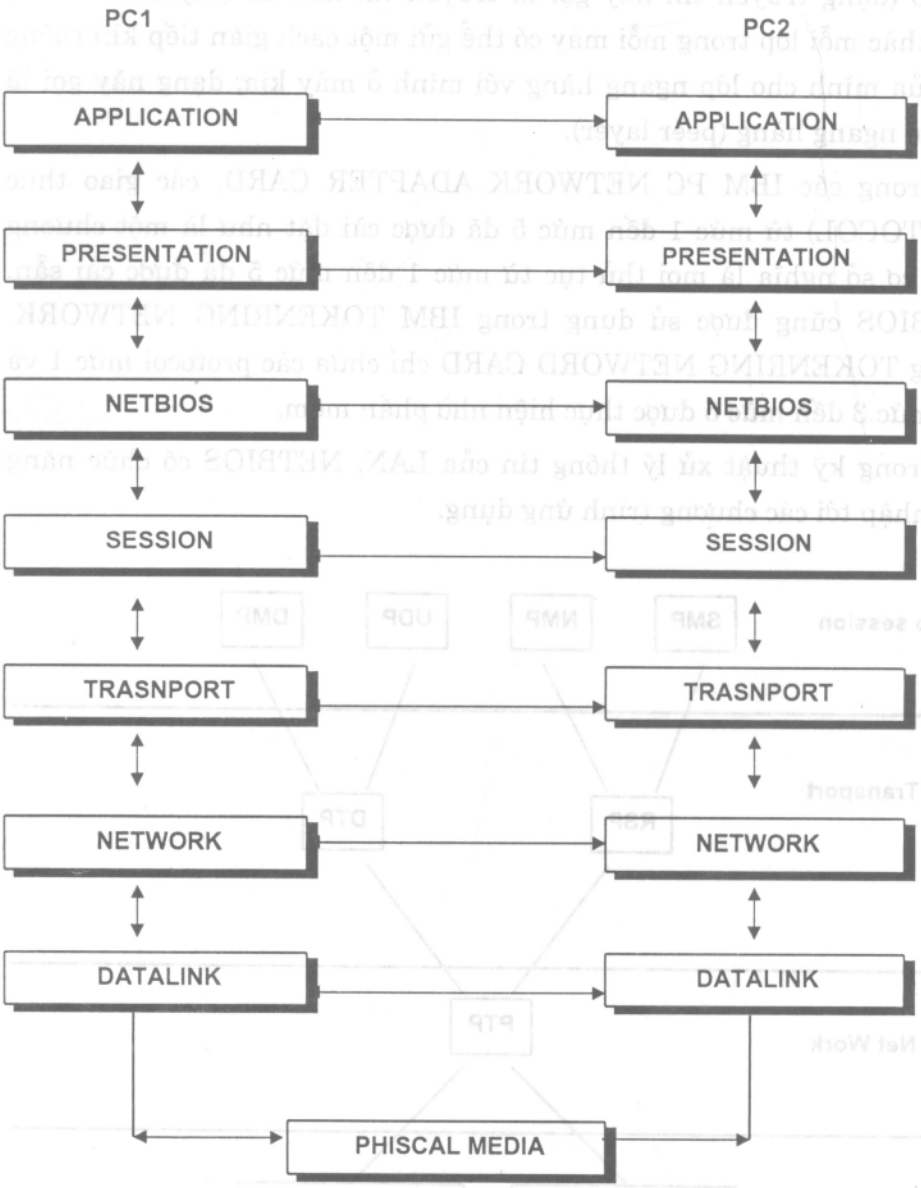
7.1. NGUYÊN TẮC LÀM VIỆC CỦA NETBIOS

Netbios là chương trình cơ sở nhằm cung cấp các dịch vụ Network để xây dựng các chương trình các ứng dụng chạy trên hệ điều hành đơn nhiệm và hệ điều hành đa nhiệm. NETBIOS cung cấp các dịch vụ giữa chương trình ứng dụng và phần cứng của LAN tương tự như IO.SYS và BIOS cung cấp các dịch vụ giữa các ứng dụng chương trình và các phần cứng của thiết bị ngoại vi. Vị trí của nó trong mối quan hệ giữa các phần mềm mạng được chỉ ra trên hình 7.1.



Hình 7.1. Mối quan hệ NETBIOS với các phần mềm mạng

Trong hình 7.1 Redirector là một phần mền đảm bảo chuyển hướng đối với yêu cầu của chương trình sử dụng phần mềm hệ thống PC. Trong chuẩn OSI NETBIOS nằm giữa mức 5 và mức 6 (hình 7.2).

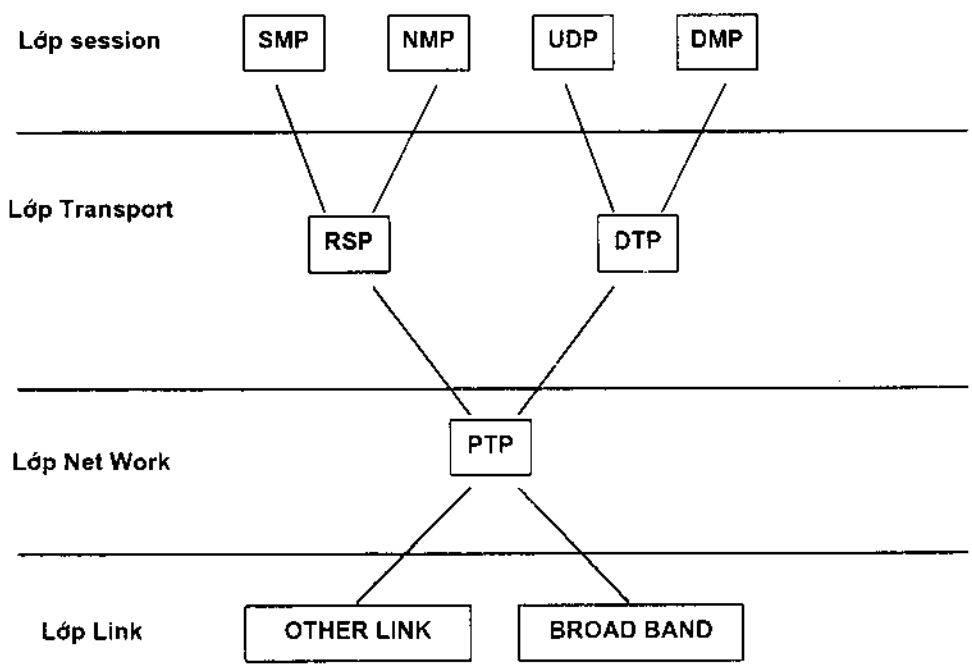


Hình 7.2. Vị trí của NETBIOS trong chuẩn OSI

Trên hình 7.2 chúng ta thấy quá trình truyền dẫn thông tin từ một ứng dụng này đến một ứng dụng khác xảy ra như thế nào trong mô hình OSI. Mỗi lớp trong PC cần gửi trực tiếp các thông tin đến các lớp liền kề với nó (dạng truyền tin này gọi là truyền tin liền kề (adjacent layre), mặt khác mỗi lớp trong mỗi máy có thể gửi một cách gián tiếp khi thông báo của mình cho lớp ngang hàng với mình ở máy kia, dạng này gọi là truyền ngang hàng (peer layer).

Trong các IBM PC NETWORK ADAPTER CARD, các giao thức (PROTOCOL) từ mức 1 đến mức 5 đã được cài đặt như là một chương trình cơ sở nghĩa là mọi thủ tục từ mức 1 đến mức 5 đã được cài sẵn. NETBIOS cũng được sử dụng trong IBM TOKENRING NETWORK, nhưng TOKENRING NETWORD CARD chỉ chứa các protocol mức 1 và 2 từ mức 3 đến mức 5 được thực hiện nhờ phần mềm.

Trong kỹ thuật xử lý thông tin của LAN, NETBIOS có chức năng truy nhập tới các chương trình ứng dụng.



Hình 7.3. Các giao thức cục bộ NET/PC

SMP: SESSION MANAGEMENT PROTOCOL (giao thức quản trị phiên trục)

NMP: NAME MANAGEMENT PROTOCOL (giao thức quản trị tên)

UDP: USER DATAGRAM PROTOCOL (giao thức chương trình ứng dụng)

DMP: DIAGNOSTIC PROTOCOL(giao thức dự đoán)

RSP: RELIABLE STREAM PROTOCOL (giao thức luồng tin cậy)

DTP: DATAGRAM TRANSPORT PROTOCOL (giao thức truyền dữ liệu)

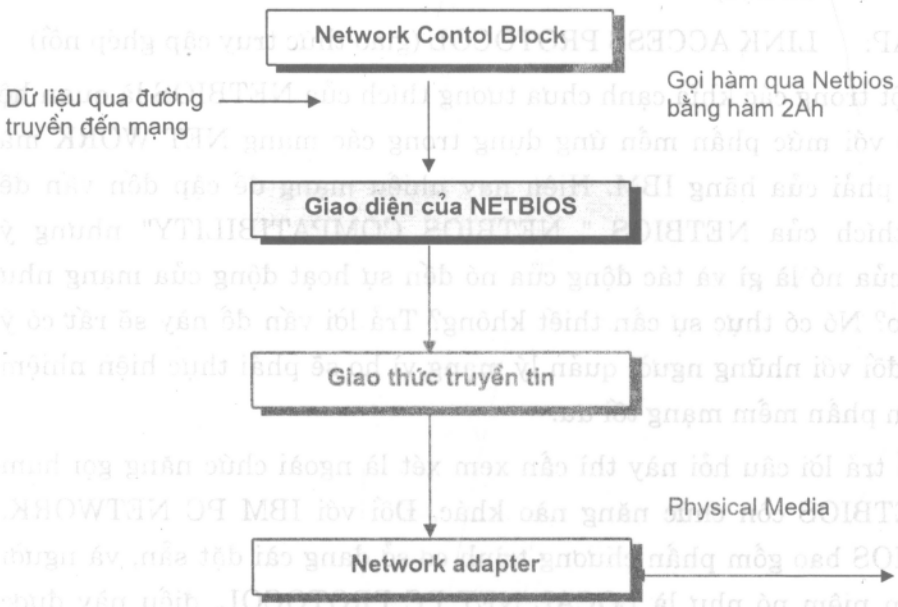
LAP: LINK ACCESS PROTOCOL (giao thức truy cập ghép nối)

Một trong các khía cạnh chưa tương thích của NETBIOS là quan hệ của nó với mức phần mềm ứng dụng trong các mạng NET WORK mà không phải của hãng IBM. Hiện nay nhiều mạng đề cập đến vấn đề ương thích của NETBIOS " NETBIOS COMPATIBILITY" nhưng ý nghĩa của nó là gì và tác động của nó đến sự hoạt động của mạng như thế nào? Nó có thực sự cần thiết không? Trả lời vấn đề này sẽ rất có ý nghĩa đối với những người quản lý mạng vì họ sẽ phải thực hiện nhiệm vụ chọn phần mềm mạng tối ưu.

Để trả lời câu hỏi này thì cần xem xét là ngoài chức năng gọi hàm thì NETBIOS còn chức năng nào khác. Đối với IBM PC NETWORK, NETBIOS bao gồm phần chương trình cơ sở đang cài đặt sẵn, và người ta quan niệm nó như là LOCAL NET PC PROTOCOL, điều này được minh họa ở hình 7.3. Hình 7.3 mô tả các lớp từ 1 đến 5 đã được thực hiện nhờ card mạng (đối với IBM PC NETWORK ADAPTER - CARD), riêng với IBM TOKEN RING NETWORK, lớp 1 và lớp 2 có khác hơn, còn các lớp còn lại hoàn toàn tương đương.

Đa số người sử dụng dùng NETBIOS để thực hiện các lệnh nào đó được hỗ trợ bằng NETBIOS PC NETWORK. Dạng chương trình ứng dụng khác là có thể truy xuất đến các dịch vụ do NETBIOS hỗ trợ qua 1 hoặc 2 lệnh gọi là hợp ngữ. Một lệnh gọi là ngắt mềm 5Ch, gọi địa chỉ

trực tiếp của IBM PC ADAPTER CARD; lệnh còn lại là ngắt mềm 2Ah, lệnh này không phụ thuộc vào IBM PC ADAPTER CARD hiện thời. Một chương trình mô phỏng NETBIOS, chúng ta thấy rất rõ trong các mạng không phải của IBM, có thể hỗ trợ các ứng dụng phần mềm giống như các dịch vụ của NETBIOS. Nếu chương trình ứng dụng được viết và sử dụng ngắt 2Ah, chương trình mô phỏng sẽ bảo đảm tiêu chuẩn xử lý tuần tự, chương trình ứng dụng cung cấp các yếu tố dữ liệu cần thiết, các yếu tố này được cài đặt vào trong NCB (NETWORK CONTROL BLOCK). Khi lệnh gọi ngắt 2AH được thực hiện, bộ mô phỏng sẽ xử lý các đòi hỏi xử lý của LAN.

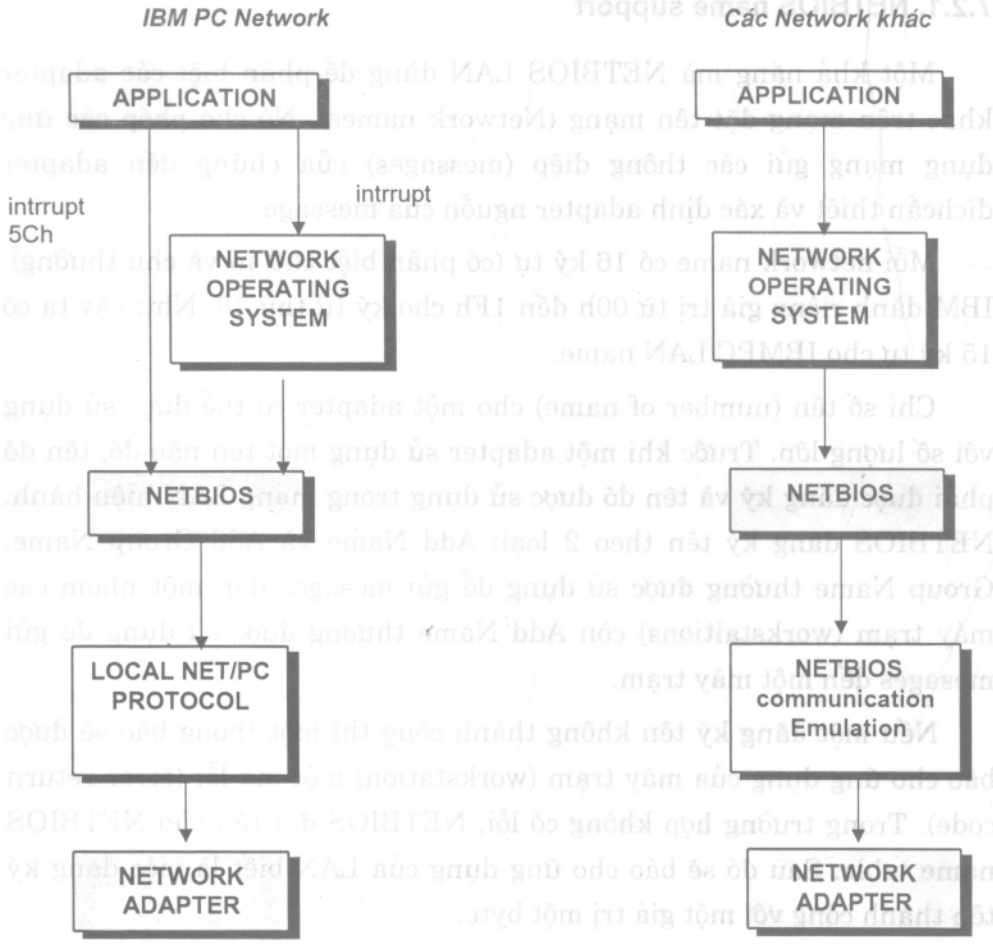


Hình 7.4 **Mô tả các tác động của NETBIOS**

Thuận lợi của chương trình này là người lập trình ứng dụng có thể sử dụng NCB tiêu chuẩn để cài đạt LAN như một thiết bị vào ra. Để làm việc đó, NETBIOS tổ chức đặt tên cho các máy trạm WORK STATION hoặc máy chủ FILE SERVER có trong mạng. Tên được gán một cách logic đến một đối tượng được kích hoạt. Ví dụ các thư mục dùng chung có các tên của mình, các máy in được kết nối cũng có tên... Mỗi WORK

STATION hoặc FILE SERVER có tên riêng của mình. IBM PC NETWORK ADAPTER CARD có thể gán 16 tên cho một thiết bị, một trong số 16 tên này hỗ trợ cho chính thiết bị.

Vai trò của NETBIOS đối với mạng IBM và không IBM được mô tả trên hình 7.5.



Hình 7.5. **Vai trò của NETBIOS trong mạng LAN**

7.2. CÁC HỖ TRỢ CỦA NETBIOS

NETBIOS cung cấp 4 hỗ trợ cơ bản như sau:

- NETBIOS name support.
- NETBIOS datagram support.
- NETBIOS session support.
- NETBIOS general support.

7.2.1. NETBIOS name support

Một khả năng mà NETBIOS LAN dùng để phân biệt các adapter khác trên mạng đặt tên mạng (Network names). Nó cho phép các ứng dụng mạng gửi các thông điệp (messages) của chúng đến adapter đíchcần thiết và xác định adapter nguồn của message.

Mỗi network name có 16 ký tự (có phân biệt chữ in và chữ thường). IBM dành riêng giá trị từ 00h đến 1Fh cho ký tự thứ 16. Như vậy ta có 15 ký tự cho IBMPC LAN name.

Chỉ số tên (number of name) cho một adapter có thể được sử dụng với số lượng lớn. Trước khi một adapter sử dụng một tên nào đó, tên đó phải được đăng ký và tên đó được sử dụng trong mạng LAN hiện hành. NETBIOS đăng ký tên theo 2 loại: Add Name và Add Group Name. Group Name thường được sử dụng để gửi mesages đến một nhóm các máy trạm (workstaitions) còn Add Name thường được sử dụng để gửi mesages đến một máy trạm.

Nếu một đăng ký tên không thành công thì một thông báo sẽ được báo cho ứng dụng của máy trạm (workstation) một mã lỗi (error return code). Trong trường hợp không có lỗi, NETBIOS đặt tên vào NETBIOS name table. Sau đó sẽ báo cho ứng dụng của LAN biết là việc đăng ký tên thành công với một giá trị một byte.

Byte không dấu này được gọi là NetBIOS name number. Name number này là một chỉ số, được sử dụng trong các lệnh của NetBIOS dùng để liên hệ với name của nó. NetBIOS gán giá trị cho name number tăng dần, module 255 và quay vòng. Giá trị 00 và 255 không được gán.

NetBIOS name table là một bảng tạm thời ở trong RAM và nó được tạo lại mỗi khi hệ thống khởi động hay khởi động adapter. Nếu một

NetBIOS module cung cấp nhiều hơn một LAN adapter trong một máy trạm workstation, mỗi adapter có một NetBIOS name table độc lập nhau.

7.2.2. Datagram và session support

Khi một adapter được phép hoạt động trong một mạng, các chương trình ứng dụng trong một máy trạm workstion có thể sử dụng NetBIOS để kết nối với các ứng dụng khác trong cùng một workstation hay trên các workstation khác. Các ứng dụng có thể sử dụng datagram hay session.

- *Datagram support*

Datagram là một thông báo ngắn có kích thước thay đổi tùy theo NetBIOS. Nó không đảm bảo là các thông báo có đến nơi an toàn không vì không có một xác nhận nào được cung cấp bởi NetBIOS. Trạng thái của máy nhận có thể là:

- Không hoạt động.
- Nguồn bị tắt.
- Không sẵn sàng chờ một datagram nào.

Một datagram có thể không bao giờ được nhận bởi bất kỳ một worrkstation nào. Kết nối datagram là kết nối kiểu" gửi và mời". Thuận lợi của kết nối datagram là nó ít tốn tài nguyên hơn kết nối session.

Có hai kiểu kết nối datagram: "broadcast datagram" và "plain datagram". Trong cả hai trường hợp, lệnh NetBIOS truyền datagram đều tham chiếu đến number name NetBIOS cục bộ. Plain datagram được truyền đến các group names, và broadcast datagram có độ an toàn dữ liệu rất thấp vì nó có thể bị cản trở bởi một tác động rất nhỏ.

Broadcast datagram không phân biệt các datagram được truyền với lệnh NetBIOS send Broadcast datagram. Bất kỳ một adapter nào, kể cả adapter truyền tin, đều có thể nhận datagram nếu trước đó lệnh NetBIOSNetBIOS Receive Broadcast datagram được phát đi. Nên tránh truyền Broadcast datagram, bởi vì hai ứng dụng trong cùng một

workstation có thể dễ dàng nhận Broadcast datagram của ứng dụng khác.

Plain datagram thì phân biệt các datagram được truyền với lệnh NetBIOS send datagram. Không như lệnh NetBIOS send Broadcast datagram, các ứng dụng đặc tả NetBIOS name nhận trong lệnh NetBIOS send datagram.

Nếu một ứng dụng đặc tả name number là Ffh, ứng dụng này có thể nhận một datagram của bất kỳ name trong NetBIOS name table. Điều này được gọi là Receive any-datagram.

- *Session support*

NETBIOS session support tạo sự kết nối để truyền dữ liệu hai chiều giữa hai ứng dụng trong một thời gian dài.

Khởi tạo session:

Các sessiondược tạo khi một ứng dụng phát hiện NetBIOS Listen tham chiếu đến một tên trong NetBIOS name table của nó. Các trình ứng dụng có thể sử dụng một tên có trong NetBIOS name table hay là thêm tên vào trong bảng đó. Trình ứng dụng thứ hai phát hiện NetBIOS Call với tên tham chiếu là tên mà ứng dụng thứ nhất đang đợi. Nếu phù hợp thì hai lệnh Listen và Call thực hiện thành công, session đã được khởi tạo.

Mỗi ứng dụng sẽ nhận thông báo session đã thiết lập bằng một giá trị một byte không dấu được gọi là NetBIOS Local Session Number (LSN) để adapter liên hệ với session. NetBIOS gán giá trị LSN tăng dần, Module 255 quay vòng. giá trị 00 và 255 không bao giờ được gắn vì đó là cận trên và dưới.

Đặc tính của lệnh Receive:

Sau khi thiết lập session, hai bên có thể phát lệnh NetBIOS Send và Receive để truyền dữ liệu. Nếu một tên cho trước được dùng để tạo nhiều session, ứng dụng có thể phát hiện NetBIOS Receive-Any-for-a-Specfied-name (Receive-Any) để nhận dữ liệu của bất kỳ session đã liên hệ với tên được đặc tả. Một ứng dụng có thể phát hiện NetBIOS Receive-

Any-for-Any-Name (Receive-Any-Any) để nhận dữ liệu của bất kỳ session hiện hành.

Thứ tự ưu tiên như sau:

1. Receive
2. Receive - Any-For-a-Specified- Name
3. Receive - Any-For - Any - Name

Đặc tính của lệnh gửi:

Các ứng dụng phát hiện NetBIOS Send để truyền dữ liệu đến các ứng dụng khác. Lệnh Send cho phép các ứng dụng gửi các thông báo có kích thước từ 0 byte đến 64 K trừ 1 byte dữ liệu. Các dữ liệu phải liên tục trong bộ nhớ. ứng dụng có thể phát lệnh NetBIOS Chain Send cho phép dữ liệu ở trong 2 Buffer nằm ở 2 vùng lưu trữ khác nhau.

Với lệnh Chain Send, dữ liệu ở trong mỗi buffer phải được bố trí liên tục trong bộ nhớ còn giữa hai buffer này không được bố trí liên tục. Hơn nữa, mỗi khối dữ liệu có kích thước 0 byte đến 64 K trừ một byte. Như vậy, cho phép 128K trừ 2 bytes dữ liệu có thể được truyền với lệnh Chain Send.

Lưu ý về lệnh Send và Receive:

Chú ý đầu tiên là có lệnh NetBIOS Chain Send, nhưng không có lệnh NetBIOS Chain Receive. NetBIOS cho phép các ứng dụng nhận một phần của của lượng tin và phát hiện NetBIOS Receive để nhận phần còn lại. Điều này đúng cho các thông báo bắt nguồn từ cả hai lệnh Send và Chain Send. Ngược lại, một lệnh đơn NetBIOS Receive thường thường nhận các thông báo được truyền với lệnh Chain Send nếu thông báo không quá lớn. Trong bất kỳ sự kiện nào, ứng dụng nhận không thể biết được thông báo được truyền với lệnh Send hay Chain Send, trừ khi nó có kích thước vượt quá 64K trừ một byte. Bởi vì trong lệnh Chain Send, dữ liệu khởi đầu trong 2 buffer riêng biệt và luôn luôn đến không liền một khối và không có một sự hiển thị nào về ranh giới của các buffer nguồn.

Các ứng dụng nhận từng phần của thông báo và không được giữ chậm quá lâu để nhận toàn bộ thông báo. Nói rõ hơn, khi một session được thiết lập, mỗi bên cần đặc tả thời hạn time-out cho lệnh Send và Receive. Nếu việc gửi vượt quá ngưỡng giới hạn trước khi thông báo được nhận hoàn tất, lệnh gửi sẽ bị time-out và session bị kết thúc bởi adapter gửi. Trong trường hợp này, cả hai bên sẽ được thông báo về hậu quả của việc gửi không thành công này.

- ***Chấm dứt bình thường Session***

Session được chấm dứt bởi một hay cả hai bên phát lệnh NetBIOS Hang Up và chỉ rõ số LSN của session được chấm dứt, ứng dụng khác được thông báo về session chấm dứt khi nó phát lệnh session đến sau. Một ứng dụng có thể phát lệnh NetBIOS Session để biết trạng thái của một session, đang hiện hành hay đã bị hủy bỏ.

7.2.3. General suppport

Lệnh NetBIOS General cung cấp các hỗ trợ NetBIOS sau:

- Reset
- Adapter Status.
- Cancel hay Unlink.
- Find Name.
- Trace.

- ***Lệnh RESET***

Lệnh RESET đưa adapter về trạng thái khởi tạo. Chấm dứt tất cả session và xóa tất cả các tên trong NetBIOS name table ngoại trừ Permanent Node name. Lệnh Reset sẽ đặc tả số lượng tối đa cho phép các lệnh của NetBIOS gắn vào cùng một thời điểm, và số lượng tối đa các session hiện hành có trong adapter.

- ***Lệnh ADAPTER STATUS***

Lệnh Adapter cho phép ta biết được các thông tin về trạng thái của adapter cũng như xác định những lỗi của LAN và NetBIOS name table.

- *Lệnh CANCEL và UNLINK*

Lệnh NetBIOS Cancel cho phép các ứng dụng hủy bỏ các lệnh chưa được hoàn tất. Lệnh NetBIOS Unlink cho phép PC NetBIOS LAN Adapter không nối kết nữa.

- *Lệnh FIND NAME*

Lệnh NetBIOS Find Name sẽ xác định các adapter có các symbolic name được đặc tả trong lệnh NetBIOS Find Name.

- *Lệnh TRACE*

Lệnh Trace thực hiện từng bước tất cả các lệnh đã được đưa ra cho giao diện NetBIOS. Điều này giúp cho việc phân tích các chương trình.

7.2.4. Phát các lệnh của NetBIOS

Các ứng dụng phát lệnh NetBIOS cần có một vùng đệm 64 bytes trong bộ nhớ. Trình ứng dụng sử dụng vùng này để xây dựng NetBIOS Control Block (NCB). Sau khi hoàn thành việc điền các Field của NCB thì có thể phát lệnh NetBIOS.

Sau khi điền đầy đủ vào NCB, ứng dụng sử dụng cặp register ES:BX để chỉ vào vị trí của NCB, và phát yêu cầu qua ngắt 5Ch.

7.3. NETWORK CONTROL BLOCK (NCB).

7.3.1. Cấu trúc của NCB

Các ứng dụng truy xuất các hỗ trợ NetBIOS cần sử dụng NetBIOS Control Block. NCB có 64 bytes, với 13 fields và 14 bytes danh dự trữ. Các field của NCB được mô tả trong bảng 7.1.

Trong ngôn ngữ ASSEMBLER, NCB được mô tả như sau:

```
NCB STRUC
    Ncb_Command         db 00h
    Ncb_RetCod          db 00h
    Ncb_Lsn             db 00h
```

```
Ncb_Num          db 00h
Ncb_BuferOff     dw 0000h
Ncb_BuferSeg     dw 0000h
Ncb_Length       dw 0000h
Ncb_CallName     db 16 dup(0)
Ncb_Name         db 16 dup(0)
Ncb_Rto          db 00h
Ncb_Sto          db 00h
Ncb_PostOff      dw 0000h
Ncb_PostSeg      dw 0000h
Ncb_Lan_Num      db 00h
Ncb_Cmd_Cplt     db 00h
Ncb_Reserved     db 14 dup(0)
NCB Ends
```

***Bảng 7.1.* Các trường (Fields) của NCB.**

Offset	*Tên trường (Field Name)*	*Độ dài (bytes)*
00h	Command	1
01h	Return Code	1
02h	Local Session Number	1
03h	Name Number	1
04h	Buffer Address	4
08h	Buffer Length	2
0Ah	Call Name	16
1Ah	Name (local)	16
2Ah	Receive Time out	1
2Bh	Send Tim,e out	1
2Ch	Post Routine Address	4
30h	LANA Number	1
31h	Comand Complete Flag	1
32h	Resetved Field	14

7.3.2. NCB Command

Field NcbCommand là một byte dùng để đặc tả mã lệnh NetBIOS. Nếu bit cao nhất của mã lệnh là zero, NetBIOS chấp nhận yêu cầu và trở về ứng dụng khi lệnh hoàn tất. Nó được tham chiếu đến như là lựa chọn wait (wait option). Tại một thời điểm chỉ có một lệnh với lựa chọn wait có thể được gán vào.

Một vài lệnh như Reset, Cancel, Unlink phải đảm bảo được hoàn thành, các lệnh khác chỉ được thực hiện với vài điều kiện. Nếu một lệnh không hoàn tất được thì NetBIOS không bao giờ trở về và máy bị treo ngay. Để tránh tình trạng này, các ứng dụng có thể đặt bit cao nhất của mã lệnh trong field Command là 1 đối với các lệnh, ngoại trừ lệnh Reset, Cancel, Unlink. Điều này được tham chiếu như là lựa chọn No-wait. Trong trường hợp này, NetBIOS cung cấp một mã trả về (return code) chứa trong AL (đối với IBM PC) ngay sau khi lệnh được khởi tạo. Mã trả về thứ hai, gọi là mã trả về cuối cùng (final return code) sẽ được trả về khi lệnh hoàn tất.

Mã trả về thứ nhất không phải là 00h thì NetBIOS không chấp thuận lệnh và không xử lý.

- ***RETURN Code***

Field return code là một byte, chứa giá trị của mã trả về cuối cùng (final return code) khi NetBIOS hoàn tất lệnh.

Nếu lựa chọn no-wait được sử dụng, cả hai field Ncb command complete và Ncb return code chứa final return code sau khi lệnh hoàn tất:

- Mã trả về có giá trị 00h, xác nhận lệnh hoàn tất thành công.
- Mã trả về có giá trị 1 đến 254 (Feh) xác nhận lệnh chấm dứt không thành công.

- ***Local Sesion Number***

Field NCB local session number gồm 1 byte, chứa chỉ số session cục bộ liên hệ với lệnh NetBIOS. Nó được gán bằng một giá trị tăng dần với

modulo 255 và quay vòng. Lưu ý là giá trị 00 và 255 không bao giờ được gắn.

- ***Name number***

Field Ncb Name Number gồm một byte, chứa name number trong NetBIOS name table liên hệ với lệnh. NetBIOS gán giá trị cho name number tăng dần, modulo 255, quay vòng. Giá trii 00 và 255 không bao giờ được gán. Name number one luôn luôn đượcgán cho chỉ số của permanent node name.

Giá trị Ffh của Field name number đặc tả với lệnh Receive - Any và Receive datagram để xác định rằng dữ liệu có thể được tiếp nhận đối với bất kỳ session hiện hành hay name đã được đăng ký.

- ***Bufer address***

Field Ncb buferr address gồm 4 byte, chứa pointer chỉ đến buffer dữ liệu. Trong IBMPC, địa chỉ có dạng OFFSET: SEGMENT.

Đối với lệnh Adapter Status và Session status, field này chứa địa chỉ nơi NetBIOS có thể đặt thông tin trả về. Đối với lệnh Cancel, field này trỏ đến một Ncb chỉ định cho việc hủy bỏ. Đối với lệnh Chain Send, Send. Send Datagram và send Broadcast Datagram thì field này đặc tả địa chỉ nơi dữ liệu được chứa và được truyền đi. Đối với lệnh Receive, Receive - Any, Receive Datảgam và Receive Broadcast Datagram, field này chỉ định kích thước của bufer nhận dữ liệu.

- ***Buferr length***

Field Ncb bufer length gồm 2 byte, xác định kích thước của buffer được chỉ định trong field Ncb buffer address. Đối với lệnh Send, Chain Send, Send datagram và Send Broadcast Datagram, field này chỉ định kích thước của buffer nhận dữ liệu.

- ***Call (remote) name***

Field Ncb call name gồm 16 byte, chứa remote name liên hệ với yêu cầu:

Đối với lệnh Call, Listen và Send Datagram, field này chứa tên ứng dụng muốn nối kết. Tên luôn luôn là remote name, tuy thế, tên có thể là local name đối với local session. Trong trường hợp lệnh Listen, ký tự asterisk (*) ở vị trí đầu tiên của field này xác định rằng Listen có thể thỏa mãn yêu cầu của lệnh Call bất kỳ gọi tới tên được đặc tả trong field local name. Đối với lệnh Chain Send, field này dùng để đặc tả kích thước (2 bytes đầu) và địa chỉ (4 bytes tiếp theo) của buffer thứ hai.

- *Local name*

Field Ncb (local) name gồm 16 byter, chứa local name liên quan với yêu cầu. Ký tự đầu tiên không được đặt là dấu asterisk (*).

Đối với lệnh Add Name và Add Group name, field này chứa tên mà ứng dụng yêu cầu NetBIOS đăng ký vào NetBIOS name table. Đối với lệnh Delete thì field này chứa tên mà ứng dụng yêu cầu xóa khỏi NetBIOS name table. Đối với lệnh Call hay Listen thì field này chứa tên localnetwork mà ứng dụng nuốn sử dụng để xây dựng session.

- *Receive time out*

Field Ncb receive time out gồm một byte, được sử dụng trong lệnh Call và Listen. Nó xác định số lượng nửa giây một mà lệnh Receive cần phải chờ trước khi time out và trả về một lỗi. Nếu field này có giá trị là 00 thì không có ngưỡng time - out đối với lệnh Receive.

Thời hạn time out có thể khác nhau đối với mỗi session, nhưng khi đã được ấn định cho session nào thì không có khả năng thay đổi khi session đã được thiết lập.

Khi hết hạn time out của lệnh Receive - Any hay Receive, session không bị kết thúc.

- *Send time out*

Field Ncb send time out gồm một byte, được sử dụng trong lệnh Call và Listen. Nó xác định số nửa giây mà lệnh Send có thể chờ được khi time out và trả về một mã lỗi. Nếu lệnh Send time - out, session sẽ kết thúc. Nếu Filed này có giá trị là 00 thì không có ngưỡng time out.

- ***Post routine address***

Field ncb post routine address gồm 4 bytes, chứa con trỏ chỉ đến routine sẽ được thực hiện khi lệnh hoàn tất. NetBIOS chỉ xem xét field này khi lệnh được đặc tả lựa chọn no - wait.

Nếu địa chỉ này là 0--0, NetBIOS không gọi Post Routine. Nếu địa chỉ của Post Routine khác 0 thì NetBIOS sẽ gọi Post Routine như là một ngắt. NetBIOS sử dụng thanh ghi AL để lưu mã trở về. Thanh ghi CS:IP trỏ đến Post Routine và ES:BX trỏ đến NCB đã hoàn thành. Post Routine sẽ trở về NetBIOS bằng lệnh IRET.

- ***Lana adapter number***

Field Ncb LANA number gồm một byte, xác định adapter sẽ xử lý lệnh.

- ***Command complete flag***

Field Ncb command complete flag gồm một byte được xác định khi lệnh được đặc tả với lựa chọn no - wait hoàn tất. nếu field này có giá trị là Ffh thì lệnh chưa được hoàn thành. Nếu khác, field này chứa final command return code.

7.4. CÁC LỆNH CỦA NETBIOS

Lệnh của NETBIOS được thực hiện:

Khi một ứng dụng phát một lệnh đến NetBIOS, NetBIOS sẽ cung cấp một mã trả về cho yêu cầu của ứng dụng. Thành phần này phụ thuộc vào lệnh đặc tả lựa chọn wait hay no - wait. Nếu bit cao nhất của mã lệnh là 1 thì no - wait được lựa chọn. Ngược lại thì wait được chọn.

Nếu lệnh đặc tả chọn wait, quyền điều khiển không trả về ứng dụng cho đến khi adapter hoàn thành lệnh. Khi lệnh hoàn thành, final retum code sẽ được chứa trong thanh ghi AL cũng như trong field NCB Return code và field NCB command Complete.

Nếu lệnh lựa chọn no - wait, NetBIOS đưa ra từ mã trả về. Sau khi khởi tạo NCB, quyền điều khiển trả về sau ngắt yêu cầu NetBIOS với

mã trả về chứa trong thanh ghi AL. Nếu mã trả về khác 00h nghĩa là adapter không thành công khi thi hành lệnh. Nếu mã trả về là 00h, adapter xếp hàng yêu cầu và cung cấp mã final return code khi adapter hoàn thành lệnh.

Khi ứng dụng kiểm tra field NCB command Complete mà giá trị đó khác Ffh thì xác nhận lệnh hoàn tất và giá trị này là final return code. Trình ứng dụng không nên sử dụng vòng lặp dựa trên field NCB Return Code để chấm dứt vòng lặp. Bởi vì NCB Return Code được đặt trước tiên, sau khi hoàn thành NCB Return Code, NCB có thể được đặt trong hàng đợi xử lý bên trong NetBIOS POST. Field NCB Command Complete không được đặt cho đến khi NCB ra khỏi hàng đợi xử lý POST.

- ***Lệnh adapter status: 33h: wait - B3h: no-wait***

Lệnh adapter status yêu cầu NetBIOS cho biết trạng thái của local hay remote adapter.

Ncb CallName đặc tả adapter. Field này có thể là Permanent node name, group name hay unique name. Thông tin trạng thái được tả về trong buffer được chỉ định trong field Ncb Buffen@. Kích thước tối thiểu của fuffer là 60 bytes. Kích thước tối đa là 60 nhân với 18 lần tối đa số name.

	In	*Out*	*Mã Return Code*
Lệnh Ncb Command	X		00h 03h
Ncb RetCode			21h 22h 23h
Ncb Lsn		X	4xh
Ncb Num			50h-FEh
Ncb Buffer @	X		
Ncb Length	X	X	
Ncb CallName	X		Final Returrn Code
Ncb Rto			
Ncb Sto			00h 01h 03h 05h 06h
Ncb Post@	?		0Bh
Ncb LanaNum	X		19h
Ncb CmdCplt		X	21h 22h 23h
Ncb Reserve		?	4Xh

Thông báo trạng thái không được đầy đủ (06h), có hai lý do làm xuất hiện thông báo này:

1. Buffer không đủ lớn để chứa thông tin

2. Thông tin trạng thái cho bởi lệnh remote adapter status vượt quá kích thước tối đa của datagram.

- ***Lệnh add group name: 36h:wait - B6h: no-wait***

	In	Out	Mã Return Code
Ncb Command	X		00h 03h
Ncb RetCode		X	21h 22h 23h
Ncb Lsn			4xh
Ncb Num		X	50h-FEh
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrn Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

Lệnh ADD Group name thêm tên không phải là unique name vào NetBIOS name table. Adapter trả về number name vào field Ncb Num. Số này được sử dụng trong datagram support và trong lệnh Receive - Any.

- ***ADD name: 30h: wait - B0h: no-wait***

	In	Out	Immediate Return Code
Ncb Command	X		00h 03h
Ncb RetCode		X	21h 22h 23h
Ncb Lsn			4xh
Ncb Num		X	50h-FEh
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrn Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

Lệnh ADD name thêm unit name vào NetBIOS name table của adapter. Đó là tên mà adapter biết và phải là duy nhất trong mạng Adapter trả về number name trong field Ncb Num. Chỉ số này được dùng trong datagram support đối với lệnh Receive - Any.

- ***Lệnh Call: 10h: wait - 90h: no-wait***

	In	Out	Mã Return Code
Ncb Command	X		00h 03h
Ncb RetCode		X	21h 22h 23h
Ncb Lsn			4xh
Ncb Num		X	50h-FEh
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Return Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

Lệnh Call mở một session với tên nơi đến được đặc tả trong field Ncb Call Name. Lệnh này sử dụng local name được đặc tả trong Ncb Name. Adapter của tên nơi đứn phải có lệnh listen gắn vào. Khi lệnh Call hoàn tất, NetBIOS gắn local session number (SLN) đến session vừa được thành lập.

- ***Lệnh cancel: 35h:wait***

	In	Out	Mã Return Code
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Return Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

Lệnh Cancel hủy bỏ các lệnh của NetBIOS của NCB có địa chỉ được cho bởi NCb Buffer@. Ta không được hủy bỏ các lệnh: Add Group Name, Add Name, Cancel, Delete Name, Reset, Send Datagram, Send Broadcast Datagram, Séion Status, Unlink.

- ***Lệnh Chain send: 17h: wai-97h: no-wait***

Lệnh chain Send gửi dữ liệu trên session được đặc tả bởi NCB Lsn. Dữ liệu trong 2 buffer độc lập được chỉ ra bởi Ncb Buffer@ và Ncb CallName. NetBIOS nối dữ liệu của buffer thứ hai vào dữ liệu của buffer thứ nhất, và gửi dữ liệu tập hợp được như là một gói tin. Mỗi buffer có thể chứa 0 - 64K trừ một byte.

	In	*Out*	*Mã Return Code*
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrn Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

- ***Lệnh delete name: 31h:wait - B1h: no-wait***

	In	*Out*	*Mã Return Code*
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrn Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

Lệnh delete xóa tên ra khỏi NetBIOS name table.

- ***Lệnh Hang up: 12h: wait - 92h: - wait***

Lệnh Hang Up đóng session được đặc tả trong field Ncb Lsn. Nếu lệnh Send và Chain Send đang được gắn với session thì lệnh Hang Up được hoãn lại cho đến khi lệnh Send và Chain Send hoàn tất.

	In	*Out*	*Mã Return Code*
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrn Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

- ***Lệnh Listen: 11h: wait - 91h: no-wait.***

	In	*Out*	*Mã Return Code*
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrn Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

Lệnh Listen mở một session với tên được đặc tả trong field Ncb Callname. Lệnh này sử dụng local name được đặc tả trong Ncb Name.

- ***Lệnh Receive: 15h: wait - 95h: no--wait***

Lệnh Receive tiếp nhanạ dữ liệu từ thành phần của session đã được đặc tả và có lệnh Send, Chain Send gửi dữ liệu đó. Nếu hơn một lệnh có thể tiếp nhận dữ liệu trên session được chỉ định, chúng được xử lý theo ưu tiên sau:

- Receive
- Receive - Any-for-a Speciield - Name
- Receive - Any- fò- Any.

Nếu buffer của Receive không đủ lớn để tiếp nhận thông báo, thông báo message incomplete status (06h) được trả về. Để tiếp nhận phần còn lại trước khi Send time out xảy ra, ứng dụng có thể phát lệnh. Receive hay Receive - Any khác.

- ***Lệnh Receive - any: 16h: wait - 96h: no-wait***

	In	*Out*	*Mã Return Code*
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrn Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

Lệnh Receive - Any tiếp nhận dữ liệu đến từ session bất kỳ. Giá trị time - out được đặc tả trong lệnh Call và Listen không ảnh hưởng trong lệnh này.

- ***Lệnh Receive broadcast datảgam: 23h: wait - A3h: no-wait***

Lệnh Receive Broadcast Datagram tiếp nhận Datagram từ bất kỳ datagram gọi lệnh Send Broadcast Datagram. Không có time-out đối với lệnh này.

	In	*Out*	*Mã Return Code*
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrn Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

- ***Lệnh Receive datagram: 21h:wait - mA1h: no-wait.***

	In	*Out*	*Mã Return Code*
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrn Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

Lệnh Receive Datagram tiếp nhận datagram. Lệnh này không tiếp nhận broadcast datagram. Không có time-out đối với lệnh này.

- ***Lệnh Reset: 32h: wait***

Lệnh Reset sẽ reset adapter được đắc ta. Nó xóa NetBIOS name và session table và hủy bỏ bất kỳ sesion nào đang hiện hữu của adapter.

	In	*Out*	*Mã Return Code*
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrn Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

- ***Lệnh send: 14h: wait - 94h: no-wait.***

	In	*Out*	*Mã Return Code*
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrn Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

Lệnh Send gửi thông báo đến thành phần của session được đặc tả bởi giá trị Ncb Lsn. Lệnh Send không đi tương xứng với lệnh Receive sẽ làm lãng phí tài nguyên của NetBIOS.

- ***Lệnh send broadcát datagram: 22h: wait - A2h: no-wait.***

	In	Out	Mã Return Code
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrr Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

Lệnh Send Broadcast Datagram gửi datagram đến mọi adapter trong mạng. Các Remote adapter không có lệnh Receive Broadcast Datagram gắn vào sẽ không nhận được datagram. Nếu một adapter gọi lệnh Send Broadcast Datagram và có Receive Broadcast Datagram gắn vào thì sẽ nhận được datagram của chính nó. Nếu một adapter có vài lệnh Receive Broadcast Datagram gắn vào thì lệnh Send Broadcast Datagram sẽ thỏa mãn tất cả lệnh Receive Broadcast Dataram.

- ***Lệnh Send datagram: 20h: wait - A0h: no-wait.***

	In	Out	Mã Return Code
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Return Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

Lệnh Send Datagram gửi datagram đến network name đã được tả.

Những adapter có tên được đặc tả nhưng không có lệnh Receive Datagram gắn vào, thì lệnh Send Datagram chỉ thỏa mãn cho một lệnh Receive Datagram gắn vào.

- ***Lệnh Session status 34h: wait - B41: no-wait***

Lệnh session Status sẽ thông báo trạng thái của một hay nhiều session liên quan đến local name. Kích thước tối thiểu hợp lệ của buffer là 4 bytes. Trạng thái kích thước không hợp lệ của buffer (01h) được trả về nếu Ncb Length nhỏ hơn 4. Trạng thái message incomplete (06h) được trả về nếu Ncb Length nhỏ hơn dữ liệu trạng thái gửi về.

Lệnh Remote Adapter Satus thất bại nếu nó không thành công trước khi time-out của hệ thống hết hạn.

	In	*Out*	*Mã Return Code*
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrn Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

- ***Lệnh unlink: 70w***

Lệnh unlink cung cấp tương thích với PC Network LANA NetBIOS dùng để tháo bỏ nối kết ra khỏi RPL Server.

Chỉ có giá trị 00h là hợp lệ trong Ncb Lana Num.

Lệnh Unlink cung cấp tương thích với PC Network LANA NetBIOS dùng để tháo bỏ kết nối ra khỏi RPL Server.

Chỉ có giá trị 00h là hợp lệ trong Ncb LanaNUM.

	In	*Out*	*Mã Return Code*
Ncb Command	X		
Ncb RetCode		X	
Ncb Lsn			
Ncb Num		X	
Ncb Buffer @			
Ncb Length			
Ncb CallName			Final Returrn Code
Ncb Name	X		00h 03h 0Fh 0Eh
Ncb Rto			15h 16h 19h
Ncb Sto		X	21h 22h 23h
Ncb Post@	?	?	4xh
Ncb LanaNum	X		50h - FEh
Ncb CmdCplt			
Ncb Reserve			

Mã trả về và cách khắc phục:

00h Thực hiện thành công. Trở về tốt.

Ý nghĩa: Lệnh thực hiện không có lỗi.

Yêu cầu hành động: Không.

01h, Kích thước buffer không hợp lệ.

Ý nghĩa: Lệnh Send Datagram haySend Broadcast.

Datagram không thể gửi hơn 512 bytes.

Yêu cầu hành động: Đặc tả lại hợp lý kích thước của buffer và cố gắng phát lệnh lần nữa.

03h, Lệnh không hợp lệ.

Ý nghĩa: Mã lệnh sử dụng không hợp lệ.

Yêu cầu hành động: Phát lại lệnh hợp lệ.

05h, Lệnh time - out.

Yêu cầu hành động: Đối với lệnh Send, session đã bị kết thúc. Thiết lập session khác.

06h, Thông báo nhận chưa đầy đủ.

Ý nghĩa: Một phần thông báo đã được nhận, vì kích thước buffer không đủ lớn nên không nhận được đầy đủ thông báo.

Yêu cầu hành động: Phát lệnh Receive khác để nhận phần còn lại.

08h, Local Session Number không hợp lệ

Ý nghĩa: Đặc tả session number của session không hoạt động.

Yêu cầu hành động: Phát lại lệnh với đặc tả session hoạt động.

09h Không đủ tài nguyên.

Ý nghĩa: Session không thể thiết lập với chương ứng dụng từ xa vì không còn chỗ trống trong session table.

Yêu cầu hành động: Phát lại lệnh vào một thời gian sau.

0Dh Trùng tên trong local NetBIOS name table.

Ý nghĩa: Lệnh Add Name đã đặc tả một tên đã ó trong local name table.

Yêu cầu hành động: Phát lại lệnh với đặc tả tên khác.

0Eh NetBIOS name table đã đầy

Yêu cầu hành động: Chờ cho đến khi có một tên bị xoá.

0Fh Tên có trong session hoạt động, bây giờ bị xóa đăng ký.

Ý nghĩa: Tên được đặc tả trong lệnh Delete có trong một session, nhưng bây giờ bị đánh dấu không đăng ký. Tên đó không còn khả năng sử dụng trong bất kỳ session mới nào khi vẫn còn session đó và chiếm chỗ trong bảng.

Yêu cầu hành động: Đóng tất cả các sessio sử dụng tên đó đến khi tên bị xoá.

11h Session từ chối mở vì không có lệnh Listen.

Ý nghĩa: Không có lệnh Listen gắn vào giao diện NetBIOS từ xa.

Yêu cầu hành động: Chờ cho đến khi có lệnh Listen đượ phát ra trong giao diện NetBIOS từ xa.

13h Name Number không hợp lệ.

Ý nghĩa: Tên được đặc tả không còn hiện hữu hay không được đặc tả.

Yêu cầu hành động: Sử dụng chỉ số hợp lệ đã được gán cho tên.

14h Không tìm thấy tên được gọi hay không có trả lời.

Ý nghĩa: Không nhận được đáp ứng lệnh Call.

Yêu cầu hành động: Cố gắng gọi lại lệnh Call sau đó.

15h Không tìm thấy tên.

Ý nghĩa: Tên được đặc tả không có trog NetBIOS name table.

Yêu cầu hành động: Cố gắng với một tên hợp lệ.

16h Tên được sử dụng trên remote adapter

Ý nghĩa: Tên được đặc tả đã được đăng ký tại một bảng khác.

Yêu cầu hành động: Đặc tả một tên khác hay dùng một tên đã bị xoá.

17h Tên đã bị xoá

Ý nghĩa: Tên đã bị xóa và không được sử dụng

Yêu cầu hành động: thêm tên đó vào NetBIOS name table và phát lại lệnh.

18h Kết thúc session không bình thường

Ý nghĩa: Lệnh Send chấm dứt vì hết hạn time out hay có vấn đề về phần cứng.

Yêu cầu hành động: Phát lệnh remote adapter status xem trạng thái của các adapter khác và kiểm tra lại dây cáp. Thiết lập lại session đồng bộ mới.

Return Code	*Ý nghĩa*
00h	Thực hiện lệnh thành công, trở về tốt.
03h	Kích thước buffer không hợp lệ.
04h	Lệnh không hợp lệ.
05h	Lệnh hết hạn time-out.
06h	Thông báo nhận chưa đầy đủ.
07h	Lệnh Local No-ack thất bại.
08h	Local Session Number không hợp lệ.
09h	Không có tài nguyên.
0Ah	Session đã đóng.
0Bh	Lệnh đã được hủy bỏ.
0Dh	Trùng tên trong NetBIOS nam table.
0Eh	NetBIOS name table đã đầy.
0Fh	Name đã kích oạt session và bây giờ không đăng ký.
11h	Net BIOS local session table đã đầy.
12h	Từ chối mở session vì không có Listen.
13h	Name number không hợp lệ.
14h	Không tìm thấy tên được gọi hay không đáp ứng.
15h	Không tìm thấy tên.
16h	Tên được sử dụng trên remote adapter.
17h	Tên đã được xoá.
18h	Session kết thúc không bình thường.
19h	Tên liên kết bị đụng độ.
1Ah	Thiết bị remote không tương thích.
21h	Giao diện bận.
22h	Quá nhiều lệnh được gán vào.
23h	Chỉ số trong field cb Lana Num không hợp lệ.
24h	Lệnh đã hoàn thành trong khi Cancel xảy ra.
25h	Tên dành riêng được đặc tả đối với Add Group Name.
26h	Lệnh không hợp lệ để hủy bỏ.
40h	Lỗi hệ thống.
50h-F6h	Hàm của adapter không tốt.
Fah	Kiểm tra adapter.
Fdh	Không mong đợi adapter đóng.
Ffh	Trạng thái lệnh đang gắn vào.

1Ah Thiết bị remote không tương thích.

Ý nghĩa: Không mong đợi gói protocol vừa nhận được.

21h Giao diện bận.

Yêu cầu hành động: Cố gắng lệnh sau đó.

40h Lỗi hệ thống.

Ý nghĩa: Không có tài liệu.

41h Quá tai từ remote adapter.

Ý nghĩa: Remote adapter đã quá tải.

Yêu cầu hành động: Hủy bỏ adapter ra khỏi mạng và tắt máy trước khi cố gắng sử dụng mạng lần nữa.

50h-F6h Hàm của adapter không tốt.

Yêu cầu hành động: Sử dụng adapter khác.

Fbh Chương trình NetBIOS không được nạp vào PC.

Yêu cầu hành động: Nạp và bắt đầu chương trình IBM.

LAN Suppport và phát lại lệnh hay điều chỉnh khối control.

Fdh Không mong đợi adapter đóng.

Ý nghĩa: Adapter bị đóng trong khi giao diện NetBIOS đang thực hiện lệnh.

Yêu cầu hành động: Phát lệnh Reset.

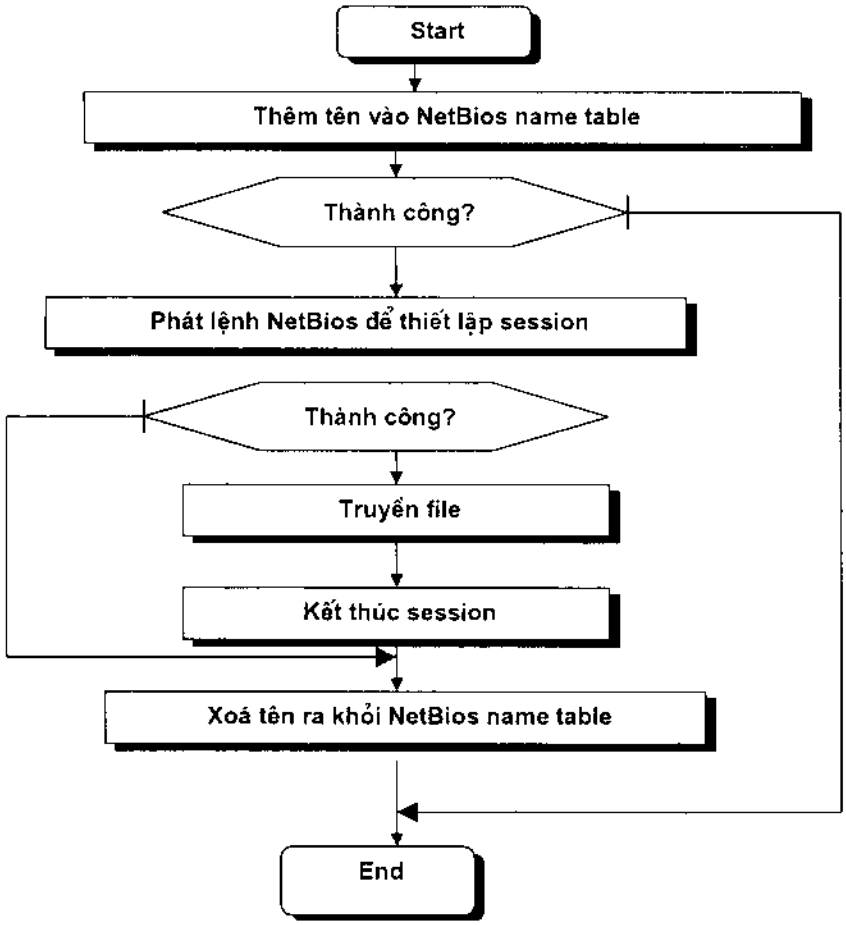
7.5. TRUYỀN FILE BẰNG SESSION

Đây là một ví dụ về truyền file giữa hai ứng dụng sử dụng khả năng của NetBIOS thông qua SESSSION SUPPORT. Trong ví dụ này, một ứng dụng sẽ thự hiện chương trình SEND để truyền file và ứng dụng kia thực hiện chương trình RECEIVE để nhận file.

7.5.1. Chương trình chuyển file Send

Thuật toán chính:

Để thực hiện truyền một file qua session support của BetBIOS, đầu tiên ta cần thiết lập một session với tên cho trước vào hệ thống (thêm tên đó vào NetBIOS name table). Thuật toán chính được minh họa theo lưu đồ 7.6.



Hình 7.6. **Lưu đồ thuật toán của chương trình Send.**

Nếu hệ thống không chấp thuận việc đưa tên này thì chương trình kết thúc và việc truyền file không thực hiện được. Nếu hệ thống chấp nhận thì chương trình sẽ khởi tạo session. Nếu khởi tạo thành công ta sẽ nhận được số LSN. Nếu không thành công thì phải hủy bỏ session và kết thúc. Trong trường hợp thành công thì ta bắt đầu bằng việc xử lý file cần truyền và thực hiện truyền file, Sau khi truyền xong chương trình sẽ kết thúc session (HANUP) và xóa tên khỏi NETBIOS name table.

- ***Xử lý chức năng truyền file***

Chương trình phải nhập vào tên file cần truyền, mở file cần truyền để đọc, nếu hợp lệ thì tiếp tục. Tiếp đó, nó đọc một khối dữ liệu từ file vào buffer (kích thước tối đa 8K) và trả về số bytes đã đọc được. Sau đó

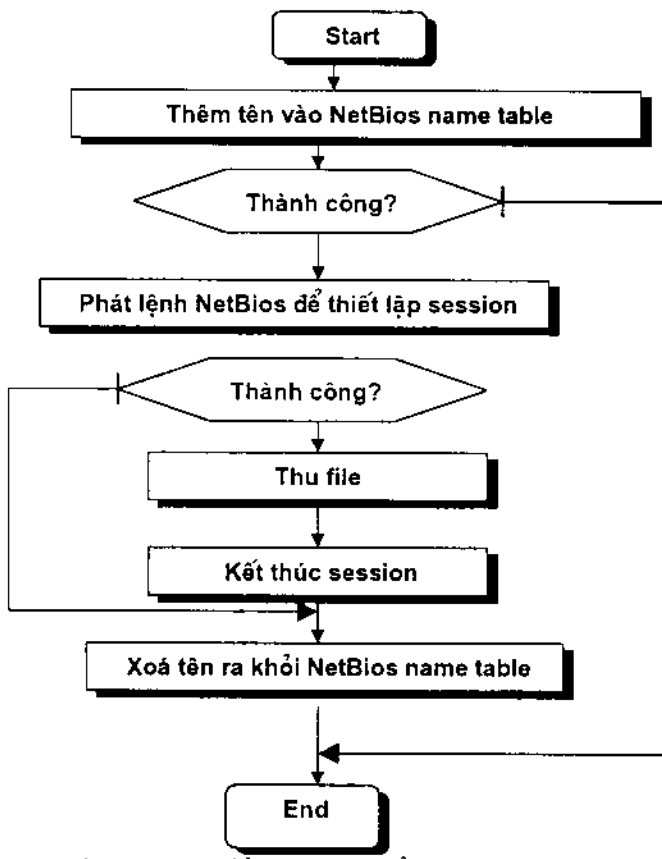
gọi hàm NetBIOS - SEND để gửi tin đi. Nếu việc gửi quá thời hạn time out thì trạng thái command - Time - out được trả về và session bị kết thúc.

Cứ như vậy, chương trình tiếp tục thực hiện cho đến khi truyền file xong.

7.5.2. Chương trình thu file Receiver

Thuật toán chính:

Quá trình thu file là quá trình ngược lại với quá trình chuyển file. Lưu đồ thuật toán chính được thể hiện trên hình 7.7.



Hình 7.7. Lưu đồ thuật toán của chương trình Receiver

Trên cơ sở của lưu đồ thuật toán cho chức năng thu và chức năng phát đã xây dựng ở trên, ta có thể dùng các ngôn ngữ lập trình khác để viết các ứng dụng trao đổi thông tin qua NetBios, và như vậy ta đã can thiệp sâu vào cấu trúc hệ thống, tận dụng được tài nguyên và ưu điểm của hệ thống.

PHỤ LỤC

PHỤ LỤC 1
CÁC VECTOR NGẮT CỦA HỆ ĐIỀU HÀNH

P1.1. Các vector ngắt của ROM BIOS

1024 byte đầu tiên trong bộ nhớ RAM (từ 0 đế 400 hex) chứa 256 ngắt 4-byte tham trỏ đến các thủ tục hay dữ liệu đã được đặt ở những nơi khác trong bộ nhớ. Mỗi bộ 4-byte này được đánh số từ 0 đến FF hex và gọi tên là các vectơ ngắt. Một số các vectơ ngắt này có thể được phần cứng truy xuất trực tiếp. Khi ta đánh vào một phím, một tín hiệu sẽ tạm thời dừng công việc của CPU. CPU sẽ tìm đến vectơ gắt số 9 (của bàn phím). Vectơ này ở tại 24 h (vì 9* 4 byte = 36 = 24 hex) và CPU sẽ lấy ra địa chỉ của thủ tục chuyên phục vụ bàn phím (đã có sẵn trong ROM BIOS). Nó sẽ thực hiện thủ thục này, xong mới quay trở về làm tiếp công việc đang bị dở dang. Máy in cũng có thể dừng CPU khi nó cần nhận một ký tự mới. Bảng 1.1 liệt kề các vectơ ngắt trỏ đến các thủ tục có sẵn trong ROM BIOS của IBM PC.

Chúng ta vừa thấy các thiết bị cứng (như bàn phím, máy in ...) có thể ngắt CPU lúc nào chúng thấy cần được phục vụ. Tuy nhiên cũng có thể dùng các vectơ ngắt thông qua phần mền. Vì các vectơ nay đã được định vị trong bộ nhớ, nên chương trình nào cũng có thể dùng chúng để tìm đến thủ tục mong muốn. thực ra lệnh INTx của bộ vi xử lý (trong đó x là số của ngắt) tự động sinh ra một lời gọi đến thủ tục ứng với vectơ ngắt x này. Lấy thí dụ, các thủ tục phục vụ màn hình của ROM BIOS bắt đầu tại địa chỉ F000: F045. Địa chỉ này sẽ được lưu trong vectơ ngắt

số 10 hex (ở tại 40 đến 43 hex). Để dùng các thủ tục này, ta có thể viết một chương trình gọi đến F000: F045. Tuy nhiên, dùng chỉ thị INT 10h sẽ đơn giản hơn. Hơn nữa, ROM BIOS có thể thay đổi từ kiểu máy này đến máy kia. Nhưng địa chỉ chứa trong vectơ ngắt số 10h sẽ luôn luôn trỏ đúng thủ tục cần tìm.

Bảng 1.1. **Các vectơ ngắt của BIOS trong IBM PC**

Vec tơ	*Địa chỉ*	*Chức năng*
5	14	In màn hình ra máy in
8	20	Đo thời gian
9	24	Mã scan từ bàn phím
10	40	Màn hình
11	44	Danh sách các thiết bị
12	48	Kích thước bộ nhớ
13	4C	Phục vụ đĩa
14	50	Chuyển nối tiếp
15	54	Máy cassette và Topview
16	58	Ký tự từ bàn phím
17	5C	Máy in
18	60	ROM- BASIC
19	64	Khởi động
1A	68	Thời gian

- ***Đổi các vectơ ngắt***

Đôi khi chúng ta muốn thay đổi hoạt động của các thủ tục trong ROM BIOS. Tất nhiên không thể dùng đến ROM BIOS vì đây là phần bộ nhớ chỉ đọc. Nhưng chúng ta lại có thể chặn đầu các vectơ ngắt. Lấy tthí dụ, ta muốn hoán đổi hai phím. Để làm việc điều này, phải thay đổi vectơ ngắt số 9 (dành cho bàn phím) để nó trỏ đến một chương trình của ta. Và trong chương trình, ta sẽ kiểm tra xem phím vừa nhận có phải là phím cần được hoán đổi. Sau đó mới gọi thủ tục phục vụ bàn phím trong ROM BIOS .

Cũng có thể làm mất tác dụng một vectơ ngắt. Lấy thí dụ, khi ta nhấn cùng lúc hai phím Shift và Prtsc thì máy in sẽ in ra những gì có trên màn hình. Nhưng nếu chưa gắn máy in thì máy tính sẽ đứng ngay. Một giải pháp là thay đổi vectơ ngắt cho máy in để nó trỏ ngay đến lệnh IRET (trở về) trong ROM BIOS. Như vậy nếuta vô tình nhấn Shift-Prtsc trong khi chưa gắn máy in thì quyền điều khiển sẽ lại tức khắc quay về chương trình của ta.

Ngắt 5: PRINT SCREEN: In từ màn hình ra máy in.

Nếu ta giữ phím Shift và đồng thời nhấn phím Prtsc, thông tin đang có trên hình sẽ được in ra máy in. Ngắt 5 được dùng vào chính mục đích này. Như vậy ta có thể kích hoạt Prtsc từ chương trình của mình bằng cách dùng lệnh:

```
INT 5h
```

Ta cũng có thể chặn không cho dùng phím này bằng cách cho vectơ số 5 này trỏ đến chỉ thị IRET (quay trở về).

Ngắt 9: Mã SCAN của bàn phím.

Như đã thấy, khi một phím được nhấn, thì bàn phím của IBM gửi đến CPU mã scan của phím, thay vì gửi mã ASCII thông thường. Ngắt 9 được dùng vào mục đích này. Ta có thể chặn mã scan bằng cách cho ngắt này trỏ đến chương trình của mình.

Ngắt 10 hex : Các phục vụ màn hình của ROM BIOS

Ngắt 9h rất đơn giản, nó chỉ có một phục vụ duy nhất. Ngược lại, ngắt số 10 hex chứa một loạt các phục vụ được đánh số bắt đầu từ 0. Các phục vụ màn hình quan trọng hơn cả sẽ được tóm tắt ngay sâu dây. Ta chỉ cần đặt số hiệu của phục vụ vào thanh ghi AH, còn dữ liệu cần thiết cho phục vụ thì đặt vào các thanh ghi khác, sau đó gọi INT 10h là đủ. Chúng ta đã xét đến 8 chế độ màn hình (liệt kê trong bảng 1.2). Bây giờ xem cách đổi các chế độ này.

- **Phục vụ 0:** ***Đặt chế độ màn hình***

Như đã thấy, IBM PC có thể dùng với kiểu hiển thị đơn sắc, kiểu hiển thị đồ hoạ. Nếu ta chỉ có kiểu kiểu hiển đơn sắc, thì chế độ là 7.

Nhưng nếu có kiểu kiểu hiển đồ hoạ, ta có thể dùng phục vụ 0 này để đổi sang một trong 7 chế độ (0-6). Còn nếu ta có cả hai kiểu kiểu hiển thì không thể chỉ dùng phục vụ 0 để đổi từ kiểu kiểu hiển này sang kiểu kiểu hiển kia. Phải dùng một chút kỹ xảo.

Để đổi sang chế độ màn hình mới, bạn hãy đặt số hiệu của chế độ mới vào thanh ghi AL. Rồi dùng chỉ thị INT 10h . Màn hình sẽ bị xoá, dù cho chế độ mới trùng với chế độ cũ.

- **Phục vụ 1:** ***Đổi kích thước con trỏ*** **(cursor)**

Một con trỏ nhấp nháy trên màn hình cho chúng ta biết nơi mà ký tự tiếp theo sẽ xuất hiện. Ký tự trên màn hình đơn sắc dựa trên một ma trận 14× 9 điểm sáng, còn màn hình đồ hoạ dùng ma trận 8× 8 điểm sáng. Lúc khởi đầu con trỏ được tạo bởi 2 dòng cuối của ma trận. Đó là các dòng số 12, 13 cho màn hình đơn sắc và các dòng 6, 7 cho màn hình đồ thị (dòng thứ nhất được đánh số 0). Tuy nhiên ta có thể thay đổi con trỏ để nó bắt đầu và kết thúc trên bất kỳ dòng nào của ma trận.

Để con trỏ trở thành một khối chữ nhật lớn và vững chắc, hãy bắt đầu bằng dòng 0 và kết thúc bằng dòng 13 hoặc 7. Ta có thể tạo một con trỏ gồm 2 phần, một phần phía trên và một phía dưới ký tự. Để làm điều này ta cho dòng bắt đầu lớn hơn dòng kết thúc. Thí dụ dòng bắt đầu là 10 còn dòng kết thúc là 2 sẽ tạo nên một con trỏ gồm 2 phần trong màn hình đơn sắc.

Để đặt kích thước con trỏ, hãy đặt CH = dòng bắt đầu, CL = dòng kết thúc, AH = 1 rồi gọi INT 10h.

- **Phục vụ 2:** ***Đổi vị trí con trỏ***

Mỗi màn hình đều có con trỏ chỉ nơi ký tự tiếp theo sẽ xuất hiện. Màn hình đơn sắc chỉ có một trang văn bản, như vậy chỉ có một con trỏ. Còn màn hình đồ thị có đến 4 trang ở chế độ 80 cột và 8 trang ở chế độ 40 cột. Mỗi trang phải có một con trỏ riêng.

Bảng 1.2. **Các phục vụ quan trọng của ROM BIOS**

AH	Input	Output	Phục vụ
0	AL = chế độ		Đặt chế độ mới cho màn hình
1	CH = dòng bắt đầu CL = dòng kết thúc		Đổi kích thước cho trỏ
2	BH = trang DH = dòng DL = cột		Đổi vị trí con trỏ
3	BH = trang	BH = trang DH = dòng DL = cột CH = dòng bắt đầu CL = dòng kết thúc	Tìm vị trí con trỏ
5	AL = số của trang		Chọn trang màn hình
6	AL = số của dòng lên CL = cột trái CH = dòng đầu DL = cột phải DH = dòng cuối BH = thuộc tính		Cuộn lên
7	AL = số dòng lên CH = cột trái CH = dòng đầu DL = cột phải DH = dòng cuối BH = thuộc tính		Cuộn xuống
8	BH = trang	AL = ký tự AH = thuộc tính	Đọc ký tự và thuộc tính
9	AL = ký tự BL = thuộc tính BH = trang CX = số lần		In ký tự và thuộc tính
Ah	AL = ký tự BL = thuộc tính BH = trang CX = số lần		In ký tự
Fh		AL = chế độ AH = độ rộng BH = trang	Xác định chế độ màn hình

Để đổi vị trí con trỏ, hay đặt BH = số thứ tự của trang DH = số thứ tự của dòng mới, DL = số thứ tự của cột mới. Để ý góc trên bên trái của màn hình là dòng 0 cột 0. Số thứ tự của dòng là từ 0 đến 24; số thứ tự của cột từ 0 đến 79 ở chế độ 80 cột và từ 0 đến 39 ở chế độ 40 cột. Sau đó đặt AH = 2 rồi gọi INT 10h.

- **Phục vụ 3:** ***Đọc vị trí con trỏ***

Phục vụ này cho phép xác định vị trí hiện thời của con trỏ. Để làm điều này hãy đặt BH = số thứ tự của trang (thường là 0). AH = 3 rồi gọi INT 10h. Lúc ngắt kết thúc thì DH sẽ chứa số thứ tự của dòng và DL số thứ tự của cột. Ngoài ra hình dạng con trỏ cũng được nhận diện : CH chứa dòng bắt đầu và CL dòng kết thúc của nó.

- **Phục vụ 5:** ***Chọn tran màn hình***

Màn hình đồ thị có 4 trang ở chế độ 80 cột và 8 trang ở chế độ 40 cột. Thông thường chúng ta dùng trang 0. Tất nhiên ta có thể đổi sang trang khác. Hãy đặt AL = số thứ tự của trang và AH = 5 rồi gọi INT 10h.

Trong chế độ 80 cột, có thể tạo ra 4 hình ảnh khác nhau cho 4 trang màn hình. Sau đó ta hãy chuyển trang để tạo ra một hoạt cảnh.

- **Phục vụ 6:** ***Cuộn lên (Scroll Up)***

Có thể cuộn toàn bộ hay một phần của màn hình. Ta hãy đặt AL = số thứ tự các dòng cần cuộn (0 đến 24). Nếu chọn AL = 0 thì sẽ cuộn toàn bộ màn hình); CL = cột trái, CH = dòng trên, DL = cột phải, DH = dòng dưới, khi văn bản được cuộn lên, các dòng trống sẽ xuất hiện phía dưới. Thuộc tính cho các dòng trống này sẽ được đặt vào BH. Như vậy BH = 0 sẽ cho dòng trống bình thường còn BH = 70 hex cho các dòng trống đảo mầu. Cuối cùng cho AH = 7 và gọi INT 10h.

- **Phục vụ 7:** ***Cuộn xuống (Scroll down)***

Phục vụ 7 giống như phục vụ 6, chỉ khác ở chỗ các dòng sẽ cuộn xuống. Ta hãy chú ý có thể cuộn toàn bộ hay chỉ một phần của màn hình.

- **Phục vụ 8:** ***Đọc một ký tự và thuộc tính***

Bộ nhớ màn hình chứa xen kẽ các ký tự và các thuộc tính của ký tự. Ta có thể nhận dạng được ký tự lẫn thuộc tính của nó bằn phục vụ này. Trước tiên, ta phải chuyển con trỏ đến vị trí cần đọc bằng phục vụ số 2. Sau đó đặt BH = số của trang (thường là 0), AH = 8 rồi gọi INT 10h. Khi kết thúc ngắt thì AL = mã ASCII của ký tự cần đọc và AL = thuộc tính.

- **Phục vụ 9:** ***in ký tự và thuộc tính ra màn hình***

Chúng ta có thể in một ký tự một hay nhiều lần lên màn hình nhờ phục vụ này. Trước tiên, ta phải chuyển con trỏ đến vị trí mong muốn bằng phục vụ số 2. Sau đó đặt AL = ký tự cần in, BL = thuộc tính (so sánh với phục vụ 8), BH = số của trang CX = số lần in ký tự. Sau đó đặt AH = 9 rồi gọi INT 10h.

Phục vụ này không tự động chuyển con trỏ sang phải sau khi ký tự được in. Ta phải tự đổi con trỏ bằng phục vụ 2 mỗi khi in ra màn hình.

- **Phục vụ A hex :** ***In ký tự***

Phục vụ này cũng giống phục vụ 9, chỉ khác ở chỗ nó không thay đổi thuộc tính; chúng ta có thể in một ký tự nhiều lần. Ta hãy đặt AH = A hex rồi gọi INT 10h.

- **Phục vụ F hex :** ***Đọc chế độ màn hình***

Phục vụ này là phần bù của phục vụ 0. Hãy đặt AH = F hex rồi gọi INT 10h. Khi ngắt kết thúc thì AL sẽ chứa chế độ hiện tại của màn hình. Ngoài ra BH = số thứ tự của trang màn hình và AH = độ rộng màn hình (40 hay 80).

- **Ngắt 11 HEX:** ***Danh sách thiết bị***

- **Ngắt 13 HEX :** ***Phục vụ đĩa mềm.***

Phục vụ này của ROM BIOS thực hiện các thao tác cơ bản cho việc đọc và ghi lên đĩa mềm. Chúng ta sẽ không dùng chúng, mà sẽ dùng các phục vụ mạnh hơn của DOS.

Khi ngắt này chạy nó sẽ trả về số hiệu ổ đĩa vào thanh ghi DL.

***Bảng1.3.* Các bit của danh sách các thiết bị**

Bit	*Ý nghĩa*
F,E	Số cổng song song
D	(không dùng cho PC)
C	Đặt nếu có bộ phối ghép trò chơi
B, A, 9	Số cổng nối tiếp
8	(không dùng cho PC)
7,6	Số lượng các ổ đĩa mềm
5,4	Chế độ màn hình 01 = 25 × 40 đồ thị 10 = 25 × 80 đồ thị 11 = đơn sắc
3,2	Dung lượng RAM hệ thống
1	(không dùng cho PC)
0	Đặt nếu có ổ đĩa mềm

- **Ngắt 15 HEX:** ***Bảng cassette và topview.***

Ta có thể có kiểu IBM PC đầu tiên mà không cần mua đĩa. Trong cấu hình này, một băng cassette để nghe nhạc có thể được dùng để lưu trữ dữ liệu.

P1.2. Các ngắt của DOS

Trong mục trên, chúng ta thấy các ngắt của ROM BIOS được đánh số từ 5 đến 1A hex. Còn các ngắt của DOS được đánh số từ 20h đến 27 hex. Dùng các ngắt nàyta có thể gọi các chức năng rất tiện dụng của DOS. Ngắt quan trọng nhất là 21 hex. Chúng ta sẽ xét đến các ngắt nàycủa DOS.

- **Ngắt 20 HEX:** ***Kết thúc chương trình.***

Chúng ta thường kết thúc chương trình bằng ngắt 20 hex vì nó thật dễ dùng với lệnh chỉ chiếm 2 byte trong bộ nhớ:

INT 20h thì điều khiển sẽ được trả lại về DOS. Nhưng INT 20 h chỉ dùng được cho chương trình COM. Không thể kết thúc một chương trình EXE bằng lệnh này.

- **Ngắt 23 HEX:** ***Chặn CTRL BREAK***

Có thể kết thúc sớm một chương trình bằng cách giữ phím Ctrl và nhấn phím Break. Đôi khi Ctrl - C cũng làm y như thế. Tổ hợp phím này làm CPU rẽ nhánh đến ngắt 23 hex, rồi sau đó đến một thủ tục của DOS để kết thúc chương trình. Tuy nhiên ta có thể ngăn không cho chương trình kết thúc sớm hoặc in ra một thông báo hướng dẫn hoặc đặt ra một câu hỏi cho người sử dụng mỗi khi ta nhấn Ctrl - Break, bằng cách cho ngắt 23 hex trỏ đến chương trình của mình. Ta có thể kết thúc chương trình bằng lệnh INT 23h .

Sau khi đổi một vectơ ngắt, đôi lúc ta muốn vectơ mới này phát huy hiệu lực đến khi tắt máy, lúc khác lại muốn phục hồi lại giá trị ban dầu của nó trước khi chương trình mình kết thúc. Riêng đối với ngắt 23 hex. DOS luôn luôn phục hồi lại giá trị ban đầu của nó khi chương trình kết thúc.

- **Ngắt 27 HEX:** ***Kết thúc và đặt thường trú.***

Hai ngắt trước rất dễ sử dụng. Ta chỉ cần dùng một lệnh 2 byte. Ngắt này cho phép đặt thường trú một phần chương trình. Phần này sẽ được lưu lại trong bộ nhớ đến khi tắt máy. Thông thường, phần thường trú này lại chặn một ngắt khác và làm thay đổi hoạt động của nó. Hoặc nó có thể điều khiển một thiết bị. Các bộ điều khiển thiết bị (device drier), các chương trình điều khiển đĩa ảo hoặc cho phép in đồng thời có thể cần đến ngắt này.

Để sử dụng ngắt 27 hex, chúng ta cho chạy một chương trình gồm 2 phần. Một phần sẽ phụ trách việc cài đặt chophần kia thường trú. Chúng ta đặt DX = địa chỉ của phần thường trú rồi cho gọi INT 27 hex. Khi phần cài đặt kết thúc thì điều khiển sẽ được trả lại cho DOS. Tất nhiên DOS không giải phóng tất cả bộ nhớ đã dành cho chương trình.

Để bảo vệ phần thường trú của chương trình. DOS phải chuyển địa chỉ nạp của các chương trình sẽ nạp sau này lên cao hơn trong bộ nhớ.

Chúng ta thường dùng ngắt 27 hex để hoán đổi hai phím, thay đổi các ký tự của máy in, và thu hẹp bộ nhớ xuống dưới 512K byte (một số chương trình đòi hỏi điều nầy).

P1.3. Các hàm của DOS

Chúng ta đã xét đến một số ngắt chỉ thực iện một một phục vụ duy nhất. Bây giờ xét đến một ngắt phức tạp có thể thực hiện hàng chục chức năng khác nhau, đó là ngắt 21 hex. Hàm cần thiết được đặt vào thanh ghi AH. Các thanh ghi khác cũng có thể đặt lại. Với ngắt này, có thể đọc bàn phím, in ra màn hình và thực hiện các thao tác đĩa.

Ta cũng nhận thấy có nhiều cặp hàm thực hiện chung một công việc. Thật ra có một nhóm hàm dành cho DOS version 1 dùng các khối điều khiển file (FCB). Nhóm thứ hai dành cho DOS version 2 và cao hơn. Nhóm này dùng các thẻ file. Chúng ta sẽ sử dụng nhóm thứ hai vì nó thuận tiện hơn. Do đó các hàm dùng FCB sẽ không được đứ ra trong phụ lục này. Một số hàm của DOS được trình bày trong bảng 1.4 .

- **Hàm 1:** ***Đọc một ký tự từ bàn phím và in ra màn hình***

Hàm này dợi ta nhấn một phím. Nếu ký tự tương ứng là ASCII thì nó sẽ hiển thị lên màn hình. Nếu nhấn Tab thì con trỏ sẽ bị đẩy sang phải 8 cột. Giữ Ctrl và nhấn Break sẽ kết thúc công việc.

Hãy đặt AH = 1 rồi gọi INT 21h. Ký tự đọc vào sẽ nằm trong AL nếu là ASII. Nếu phím là phím đặc biệt có mã mở rộng thì AL = 0. Và phải gọi hàm này một lần nữa để lấy mã mở rộng. Nếu lúc này có AL <84 thì AL chứa mã scan của phím đặc biệt này. Lấy thí dụ AL = 30, có nghĩa là tổ hợp phím Alt-A được nhấn vì mã sacn của A là 30, AL = 81 là phím PgDn, còn AL = 59 là phím F1. Nếu AL > 83 thì hãy tra catalog. Lấy thí dụ tổ hợp phím Shift-F1 cho AL = 84, còn Alt-1 cho AL = 120.

Các hàm tương tự là 6, 7 và 8. Ta có thể xác định xem còn ký tự nào đang chờ được đọc hay không bằng hàm 0B hex. Có thể đọc từ một thiết

bị khác hay từ tập tin trên đĩa bằn cách đổi hướng thiết bị vào chuẩn (Standard input, mặc định là bàn phím).

***Bảng 1.4.* Các hàm của DOS trong ngắt 21 hex**

AH (hex)	*Hàm*
1	Đọc từ bàn phím, cho hiện
2	Đưa một ký tự ra màn hình
3	Đọc vào từ cổng nối tiếp
4	Đưa ra cổng nối tiếp
5	Đưa a máy in
6	Vào/ra trực tiếp bàn phím/màn hình
7	Đọc từ bàn phím, không hiện
8	Đọc từ bàn phím, không có tab
9	Đưa một xâu ra màn hình
A	Đọc vào vùng đệm của bàn phím
B	Kiểm tra trạng thái vào
25	Đặt vectơ ngắt
30	Lấy vectơ của DOS
31	Kết thúc và ở lại thường trú
35	Lấy vectơ ngắt
3C	Tạo tập tin
3D	Mở tập tin
3E	Đóng tập tin
3F	Đọc từ tập tin hoặc thiết bị
40	ghi lên tập tin hoặc thiết bị
41	Xoá tập tin
43	Lấy/đặt thuộc tính tập tin
4A	Thay đổi kích thước 1 khối nhớ đã cấp phát
4B	Nạp/thực hiện chương trình
4C	Kết thúc chương tình
4D	Xác định mã lỗi
56	Đổi tên tập tin

- **Hàm 2:** ***Đưa một ký tự ra màn hình***

Hàm này in ra màn hình ký tự chứa trong thanh ghi DL. Nó rất tiện dụng để hiển thị 1 hay 2 ký tự. (hãy dùng hàm 9 nếu muốn in ra một xây ký tự). Hàm này in ra được ký tự dollar ($), điều mà hàm 9 không thể làm nổi. Hàm này thường đi đôi với hàm 8 (đọc từ bàn phím nhưng không cho hiển thị). Ký tự Backspace chuyển con trỏ về phía trái nhưng không xoá ký tự ở đây. Có thể đưa ký tự ra một thiết bị khác hay lên tập tin trên đĩa bằng cách dùng ký tự lớn hơn (>) để đổi hướng thiết bị ra chuẩn (Standard) output, mặc định là màn hình).

- **Hàm 3:** ***Đọc vào từ cổng nối tiếp***

Hàm 3 đọc một ký tự từ thiết bị phụ chuẩn (standard auxilliary input), có tên là AUX hay COM1. Tuy nhiên có thể đổi cổng phụ chuẩn thành COM2 bằng cách dùng chương trình MODE của DOS. Hàm này đợi đến khi có byte được đọc vào từ cổng. Hãy đặt AH = 3 rồi gọi INT 21h. Ký tự sẽ chứa trong AL.

- **Hàm 4:** ***Đưa ra cổng nối tiếp***

Hàm 4 đưa byte chứa trong thanh ghi DL ra cổng máy in chuẩn AUX hay COM1. Hãy đặt AH = 4 rồi gọi INT 21h. Có thể chọn cổng phụ COM2 bằng chương trình MODE của DOS.

- **Hàm 5:** ***Đưa ra máy in***

Hàm 5 đưa ký tự chứa trong thanh ghi DL ra cổng máy in chuẩn PRN hay LPT1. Tuy nhiên có thể chọn LPT2 hay LPT3 với chương trình MODE. Hãy đặt AH = 5 rồi gọi INT 21h.

- **Hàm 6:** ***Vào trực tiếp từ bàn phím và ra màn hình***

Hàm 6 có thể thực hiện cả vào lẫn ra: nó cũng xác định được trạng thái vào. Hàm này không đợi chúng ta nhập ký tự vào như các hàm trước. Chú ý ký tự nhập vào không tự động hiển thị ra màn hình và tổ hợp phím Ctrl-Break không kết thúc được hàm này.

Như thường lệ, thanh ghi DL chứa ký tự cần in ra còn AL chứa ký tự nhập vào. Ta nhập vào bằng cách cho DL = FF hex. Hãy đặt AH = 6

rồi gọi INT 21 hex. Khi hàm này kết thúc, cờ zero sẽ được đặt nếu không có ký tự nào sẵn sáng. Nếu cờ zero được xoá thì nghĩa là AL đang chứa ký tự mới nhập vào. Nếu DL chứa một ký tự khác với FF hex thì nó sẽ được chuyển đến thiết bị ra chuẩn. Hãy so sánh với các hàm 6 và 8.

- **Hàm 7:** ***Đọc ký tự từ bàn phím, không hiện, không CTRL - BREAK***

Hàm 7 giống hàm 1 ở chỗ nó đợi cho đến khi một ký tự được đọc vào. Nếu đó là ký tự ASCII như một chữ cái, chữ số, hay ký tự điều khiển thì nó sẽ chứa trong AL. Các ký tự đồ hoạ của IBM có thể được nhập vào bằng cách giữ phím Alt rồi đánh giá trị thập phân của nó bằng nhóm phím số ở bên phải của bàn phím. Chữ số nhập vào sẽ chứa trong thanh ghi AL. Tuy nhiên nếu đọc vào ký tự mở rộng như phím chức năng (F1 đến F12), phím mũi tên, hay các tổ hợp phím có Alt thì AL sẽ chứa 0. Phải gọi hàm này lần thứ 2 để xác định ký tự. Khác với hàm 1, hàm này không đưa ra màn hình ký tự đọc vào. Hơn nữa, Ctrl - Break không làm kết thúc công việc của hàm này. Hãy đặt AH = 7 rồi gọi INT 21 hex. Hãy so sánh với các hàm 6 và 8.

- ***Hàm 8***

Đọc ký tự từ bàn phím, không hiển thị nhưng có tác động của Ctrl-Break. Hàm 8 giống hàm 1 ở chỗ nó đợi đến khi một ký tự được đọc vào và kết thúc khi có Ctrl-Break. Nhưng ký tự này lại không hiển thị ra màn hình. Hãy đặt AH = 8 rồi gọi INT 21 hex. Hãy so sánh với các hàm 6 và 7.

- **Hàm 9:** ***Đưa ra màn hình một xâu ký tự***

Hàm này dùng để đưa ra màn hình một xâu ký tự. Các hàm 2 và 6 chỉ in ra được 1 ký tự. Tất nhiêm một số ký tự không in ra được như ký tự về đầu dòng (carriage return), xuống dòng (line feed) hay ESC cũng có thể đặt trong xâu ký tự.

Để dùng hàm này, hãy đặt xâu tại một nơi nào đó trong bộ nhớ và kết thúc xâu bằng ký tự dollar ($). (Tất nhiên không thể in ra ký tự $ này, phải dùng hàm 2 nếu cần). Địa chỉ của xâu được đặt vào cặp thanh ghi DS:DX. Hãy đặt AH = 9 rồi gọi INT 21hex.

- **Hàm A Hex:** ***Đọc vào vùng đệm bàn phím***

Các hàm 1, 6, 7 và 8 mỗi lần chỉ đọc vào được một ký tự. Nhưng đôi khi chúng ta muốn đọc toàn bộ một dòng vào. Như thế người sử dụng có thể sửa chữa dòng này trước khi nó được nhập vào máy. Hàm A hex được dùng vào mục đích này. Các ký tự nhập vào sẽ được chuyển đến vùng đệm của bàn phím. Lúc này ta có thể sử dụng được tất cả các phím soạn thảo của DOS như mũi tên trái, mũi tên phải, Ins, Del. Khi nhấn phím Enter thì dòng nhập sẽ được chuyển đến vùng đệm nhập.

Trước khi dùng hàm này, phải tạo ra một vùng đệm trong RAM. Ngay đầu vùng đệm này là 2 byte đặc tả. Byte thứ nhất định ra chiều dài tối đa của văn bản, byte thứ hai cho biết tổng số các ký tự mà ta đã nhập. Chúng ta tự chọn lấy giá trị của byte thứ nhất, còn DOS sẽ ghi vào byte thứ hai khi đã nhập xong. DOS cho thêm một ký tự về đầu dòng ở cuối xâu vừa nhập. Do đó phải đặt chiều dài tối đa bằng chiều dài văn bản tối đa cần nhập cộng thêm 1 byte.

Để dùng hàm này, hãy đặt DS:DX = địa chỉ của byte đặc tả thứ nhất trong vùng đệm nhập, đặt AH = A hex rồi gọi INT 21 hex.

- **Hàm B Hex:** ***Trạng thái bàn phím***

Chúng ta thấy các hàm 1, 7, và 8 đợi đến khi một phím được nhấn. Để tránh một sự chờ đợi vô ích, ta kiểm tra trước trạng thái đọc với hàm B hex. Đặt thanh ghi AH = B hex rồi gọi INT 21 hex. Khi ngắt này thực hiện xong thì thanh ghi AL = 0 nếu không có ký tự nào đang đợi. Một giá trị khác trong AL báo cho biết có một ký tự đang đợi được máy đọc vào.

Hàm này không đọc ký tự này được mà phải dùng các hàm 1,7 hay 8.

Các hàm cho khối điều khiển FILE (FCB)

Nhóm hàm tiếp theo thực hiện các thao tác đĩa dùng FCB. Tuy nhiên đây là một phương pháp đã lỗi thời nên chúng tôi sẽ không dùng đến. Thay vào đó, chúng ta sẽ sử dụng một nhóm hàm tương đương

dùng thẻ file. Để ý ta sẽ thấy là khi muốn xoá một tập tin thì có thể gọi hàm 13 hex, dùng FCB, hoặc hàm 41 hex, dùng thẻ file.

- **Hàm 25 Hex:** ***Đổi vectơ ngắt***

Chúng ta đã biết ở đầu bộ nhớ là các vectơ ngắt, mỗi vectơ gồm 4 byte chứa địa chỉ một thủ tục của ROM BIOS hay của DOS. Có thể thay đổi cách hoạt động của thủ tục này bằng cách cho vectơ ngắt tương ứng trỏ đến một thủ tục khác (do chúng ta viết). Thí dụ, ta có thể làm mất tác dụng phím PrtSc bằng cách đổi vectơ ngắt số 5, ở tại địa chỉ 0:14 hex trong bộ nhớ, để nó trỏ ngay đến chỉ thị IRET (kết thúc ngắt à quay trở về).

Về nguyên tắc, có thể thay đổi một vectơ ngắt bằng cách ghi vào nó một giá trị mới. Tuy nhiên phương pháp này có thể dẫn đến hậu quả không lường trước được nhất là khi ngắt lại được kích hoạt ngay lúc đang ghi gía trị mới vào vectơ của nó. Do đó, nên nhờ DOS ghi lên vectơ ngắt cho an toàn, bằng cách gọi hàm 25 hex. Hãy đặt DS:DX = giá trị mới (= địa chỉ thủ tục mới) của vectơ ngắt và AL = số hiệu của ngắt. Sau đó đặt AH = 25 hex rồi gọi INT 21 hex. Hàm 35 hex (lấy vectơ ngắt).

- **Hàm 30 Hex:** ***Lấy VERSION của DOS***

DOS version cao giới thiệu thêm nhiều hàm mới khá tiện dụng. Nếu muốn dùng đến các hàm mới trong chương trình thì trước hết phải gọi hàm 30 hex để xem version của DOS trong máy bạn. Nếu thấy là version 1 thì phải cho chương trình kết thúc sớm và in ra thông báo cần version từ 2 trở về sau.

Hãy đặt AH = 30 hex rồi gọi INT 21 hex. Khi ngắt kết thúc thì AL = phần chính của version hoặc = 0 nếu version trước 2. Còn AH sẽ chứa phần phụ của version (phần sau dấu chấm). Lấy thí dụ nếu version là 3.1 thì AL = 3 và AH = 1, còn version1.1 sẽ cho AL = 0.

- **Hàm 31 hex :** ***Kết thúc và ở lại thường trú***

Các bộ điều khiển thiết bị (device driver) và các bộ xử lý ngắt (interrupt handler) có thể được cài đặt bằng INT 27 hex. Hàm 31 hex

cũng thực hiện công việc tương tự. Hơn thế nữa, hàm này còn cho lại một mã lỗi truy cập được bằng lệnh ERRORLEVEL trong một tập tin. BAT. Ngoài ra, cũng có thể dùng hàm 4D hex để lấy ra mã lỗi này.

Chúng ta đặt thanh ghi DX = kích thước (tính bằng Paragraph = 16 byte) của phần muốn giữ thường trú trong chương trình. Sau đó đặt AH = 31 hex rồi gọi INT 21 hex.

- **Hàm 35 Hex :** ***Lấy vectơ ngắt.***

Có thể thay đổi hoạt động của một thủ tục của ROM BIOS hay DOS bằng cách gán 1 giá trị mới cho vectơ ngắt tương ứng với hàm 25 hex. Nhưng trước tiên nên lấy giá trị cũ ra và cất đi, để còn có thể phục hồi nó lại vào một thời điểm thích hợp.

Hãy đặt AL = số hiệu của ngắt. Sau đó đặt AH =35 hex rồi gọi INT 21 hex. Sau khi ngắt kết thúc thì cặp thanh ghi ES: BX = địa chỉ thủ tục tương ứng với vectơ ngắt này.

Các tập tin trên đĩa và các thẻ file.

Các hàm tiếp theo của DOS thao tác trên các tập tin thông qua các thẻ file. Chúng chỉ dùng được với version 2 trở về sau. Có thể dùng thẻ file để tạo, đọc, ghi, đổi tên, và xóa tập tin trên đĩa.

Với DOS version 1 thì việc đặc tả một tập tin trên đĩa chỉ bao gồm ổ đĩa, phần tên (dài 8 byte) và phần mở rộng (dài 3 byte). Thí dụ như

```
C: sortfast. ASM
```

DOS version 2 giới thiệu thêm khái niệm thư mục con (subdirectory). Từ đây có thể thêm một hay nhiều tên thư mục con., với dấu xổ chéo (\) đứng sau, trong đặc tả của một tập tin trên đĩa . Ví dụ như

```
C: ASSEMBLER\SYSTEMS\SORTFAST.ASM
```

và gọi là một đường tên (path name) của tập tin. Quan niệm về thẻ file được DOS giới thiệu để làm đơn giản hoá việc đặc tả một tập tin (vì đường tên có thể quá dài). Ta thêm số 0 đằng sau đường tên(sẽ được một xâu lấy tên là ASCIIZ), rồi cho cặp thanh ghi DS:DX trỏ đến đầu đường tên, đặt CX = thuộc tính của tập tin (thường là 0) rồi gọi hàm

thích hợp. DOS sẽ trả lại một thẻ file hoặc một mã lỗi trong thanh ghi AX. Cờ cary (cờ nhớ) được đặt là dấu hiệu có lỗi. thuộc tính có trong bảng 1.5. còn các mã lỗi chuẩn ở trong bảng 1.6.

Số lượng tối đa các thẻ file mà một chương trình có thể sử dụng là 8. tuy nhiên có thể tăng con số này lên 20 bằng lệnh: FILES = 20 trong tập tin CONFIG.SYS .

***Bảng 1.5.* Các thuộc tính của tập tin**

Thuộc tính	*Ý nghĩa*
0	Bình thường
1	Chỉ đọc
2	Ẩn
4	Hệ thống

***Bảng 1.6.* Các mã lỗi chuẩn khi xử lý tập tin**

Mã Hex	*Ý nghĩa*
1	Hàm không hợp lệ
2	Tập tin không có
3	Không có đường đến tập tin
4	Hết thẻ file
5	Không truy cập được (tập tin ẩn)
6	Thẻ file không hợp lệ
7	Khối điều khiển bộ nhớ (MCB) bị huỷ
8	Thiếu bộ nhớ
9	Địa chỉ không hợp lệ
A	Môi trường không hợp lệ
B	Khuôn dạng không hợp lệ
C	Kiểu truy cập không hợp lệ
D	Dữ liệu không hợp lệ
F	ổ đĩa không hợp lệ
10	Có ý định xoá thư mục hiện hành
11	Không cùng thiết bị
12	Không còn tập tin nào

Ngoài ra DOS tự động gán 5 thẻ file (từ 0 đến 4) cho 3 thiết bị ngoại vi là CON, AUX và PRN (CON dùng 3 thẻ file). Tuy nhiên chương trình của chỉ phải chịu 3 thẻ file thôi. Như vậy bạn còn lại 5 thẻ và có thể tăng con số này lên 17 với lệnh FILES = 20. Bảng 1.7 cho thấy 4 thẻ file đầu. Như thế, thẻ file của tập tin đầu tiên mà DOS gọi sẽ bằng 5.

***Bảng 1.7.* Năm thẻ file do DOS đặt**

Thiết bị thẻ file	*Công dụng*
CON 0	Vào chuẩn (thường là bàn phím)
CON1	Ra chuẩn (thường là màn hình)
CON 2	Lỗi ra màn hình
AUX 3	Vào và ra phụ
PRN 4	Ra máy in (LPT1 hay PRN)

- **Hàm 3C Hex:** ***Tạo tập tin mới***

Hàm 3C hex tạo ra một tập tin mới dựa vào đặc tả trong đường tên. Nếu trên đĩa đã có một tập tin trùng tên thì nó sẽ bị xoá. Đây là điều mà thông thường chúng ta muốn. Tuy nhiên, nếu muốn tránh việc xoá đi một tập tin đã có, cần phải duyệt qua thư mục trước.

Như đã thấy trong mục trước, chúng ta đặt thêm 0 ở cuối đường tên và dùng cặp thanh ghi DS:DX để trỏ đến đầu của đường tên, còn CX chứa thuộc tính tập tin. DOS luôn đặt lại CX = 0 (thuộc tính bình thường) khi bạn tạo một tập tin mới. Hãy đặt AH = 3C hex rồi gọi INT 21 hex. Khi ngắt kết thúc ta kiểm tra cờ nhớ xem nó được đặt hay không? Nếu được đặt là có lỗi. Vì mã lỗi là chung cho tất cả các hàm dùng để truy cập các tập tin trên đĩa nên có thể viết ra một thủ tục chuẩn để xử lý lỗi. Còn nếu không có lỗi, hãy cất thẻ file (đang chứa trong AX) lại. Và sẽ dùng lại nó mỗi khi cần truy cập đến tập tin tương ứng trên đĩa.

- **Hàm 3D Hex:** ***Mở một tập tin đã có trên đĩa***

Nếu muốn đọc một tập tin đã có trên đĩa, thì trước hết phải mở nó. Cũng giống trước, hãy đặt thêm 0 vào cuối đường tên và cho DS: DX trỏ đến đầu đường tên. Đặt AL = kiểu truy cập tập tin (xem bảng 1.8, thường thì AL = 0). Sau đó đặt AH = 3D hex rồi gọi INT 21 hex. Khi ngắt kết thúc, hãy chú ý đến cờ nhớ xem có lỗi hay không ? Nếu không có lỗi thì hãy cất thẻ file đang ở trong AX.

Bảng 1.8. **Các kiểu truy cập khi mở một tập tin**

Kiểu	Ý nghĩa
0	Chỉ đọc
1	Chỉ ghi
2	Đọc và ghi

- **Hàm 3F Hex:** ***đọc từ tập tin hoặc thiết bị***

Sau khi một tập tin trên đĩa đã được khởi tạo, ghi lên rồi đóng lại, chúng ta vẫn có thể đọc được dữ liệu chứa trong nó. Tất nhiên phải mở lại tập tin bằng hàm 3D hex thì sẽ có một thẻ file. Hãy đặt BX = thẻ file rồi cho CX = số byte cần đọc vào. Có thể chọn một số bất kỳ, nhưng càng lớn càng tốt. Ngoài ra nên chọn là bội số của kích thước một sector - 512 hay 1024 để truy cập nhanh hơn. Cặp thanh ghi DS:DX trỏ đến vùng bộ nhớ mà dữ liệu sẽ được chuyển đến. Hãy đặt AH = 3C hex rồi gọi INT 21 hex. Khi ngắt kết thúc, cần nhớ kiểm tra cờ carry xem có lỗi hay không? Nếu không thì AX sẽ chứa số byte đã được đọc vào từ đĩa. Nếu thấy AX < CX thì có nghĩa đã đọc hết tập tin. Nếu chưa hết phải gọi lại hàm 3C hex để đọc tiếp. Không cần đóng tập tin nếu chỉ đọc nó, tuy nhiên việc đóng tập tin sẽ giải phóng bớt một thẻ file và để dành nó cho công việc khác sau này.

Vì DOS tự động gán thẻ file cho các thiết bị ngoại vi, ta có thể dùng hàm này để đọc từ bàn phím. Trong trường hợp này thẻ file chứa trong BX bằng 0.

- **Hàm 40 Hex:** ***Ghi tên tập tin hoặc thiết bị***

Sau khi tạo ra một tập tin với hàm 3C hex, có thể ghi dữ liệu lên tập tin này với hàm 40 hex. Hãy đặt BX = thẻ file, cho cặp DS:DX trỏ đến vùng đang chứa dữ liệu trong bộ nhớ, còn CX = số byte cần ghi lên đĩa. Sau đó đặt AH = 40 hex rồi gọi INT 21 hex. Khi ngắt kết thúc, nhớ kiểm tra cờ carry xem có lỗi hay không? Nếu không thì AX sẽ chứa số byte đã ghi lên đĩa. Nếu thấy AX < CX nghĩa là có lỗi (thường do đầy đĩa). Nếu AX = CX thì có thể còn phải ghi tiếp. Hãy nhớ đóng tập tin lại bằng hàm 3E hex.

Vì DOS tự động gán thẻ file cho các thiết bị ngoại vi nên có thể dùng hàm này để ghi lên màn hình. Trong trường hợp này thẻ file chứa trong BX bằng 1.

- **Hàm 41 Hex:** ***Xoá tập tin trên đĩa***

Hàm 41 Hex dùng để xoá một tập tin trên đĩa: Thực ra nó chỉ đánh dấu xoá tên tập tin này trong bảng thư mục. Do đó có thể cứu vớt lại một tập tin vô tình bị xoá. Ta cũng đã thấy hàm 3C hex tự động xoá một tập tin có tên trùng với tập tin đang được khởi tạo.

Hãy cho thêm 0 ở cuối đường tên và cho cặp DS:DX trỏ đến đầu đường tên. Sau hãy đặt AH = 41 hex rồi gọi INT 21 hex. Khi ngắt kết thúc, hãy kiểm tra cờ nhớ xem có lỗi hay không? Không thể xoá một tập tin đang mở hoặc chỉ đọc.

- **Hàm 42 Hex:** ***Chuyển con trỏ file***

Hàm này dùng để thay đổi vị trí logic read/ write ở một file. Để thực hiện hàm này phải nạp thẻ file vào BX, vị trí con trỏ mới vào AL, số lượng byte vào CX:DX.

Vị trí đầu trong AL được gọi là mã phương pháp. Nếu AL=0 sẽ tính từ đầu file và con trỏ được chuyển đi CX:DX byte từ vị trí này. Nếu AL=1 sẽ tính từ vị trí hiện hành. Nếu AL=2 sẽ tính từ cuối file hiện hành. Trong trường hợp cuối cùng này ta thường gán 0 cho CX:DX để tìm ra kích thước hiện thời của file. Nếu đặt 0 và yêu cầu phương pháp 0 thì sẽ trở về đầu file.

Sau khi thực hiện hàm này, DX:AX sẽ chứa con trỏ file hiện thời tính bằng byte kể từ đầu file.

Mã lỗi=1 (sai số hiệu hàm), =6 (thẻ không hợp lệ).

- **Hàm 43 Hex:** ***Thuộc tính tập tin***

Hàm này dùng để xác định hoặc thay đổi thuộc tính của một tập tin. Hãy cho thêm 0 ở cuối đường tên và cho cặp DS:DX trỏ đến đầu đường tên. Để xác định thuộc tính hãy đặt AL = 0, sau đặt AH = 43 hex rồi gọi INT 21 hex. Khi ngắt kết thúc, hãy kiểm tra cờ nhớ xem có lỗi hay không? Nếu không thì các thuộc tính đang ở trong CX. Nếu AX = FFFF hex nghĩa là không có lỗi cho đĩa cứng.

Còn nếu đặt AL = 1 và CX = thuộc tính mới thì có thể thay đổi thuộc tính cũ. Có thể đặt lại thuộc tính thứ 2, 3, 4 hay tất cả. Lấy thí dụ CX = 7 (111 nhị phân) sẽ gán cho tập tin thuộc tính chỉ đọc, ẩn vào hệ thống. còn CX = 0 sẽ gán thuộc tính bình thường.

- **Hàm 4A hex:** ***Thay đổi kích thước 1 khối nhớ đã cấp phát***

Khi một chương trình bắt đầu chạy, nó được cấp phát toàn bộ phần còn lại của bộ nhớ. Sau này chúng ta sẽ viết một chương trình gọi được một chương trình khác bằng hàm 4B hex. Để làm điều bày, trước tiên phải giải phóng khối nhớ đã được cấp phát cho chương trình đầu, để chừa chỗ nạp chương trình sau vào. Hàm 4A hex được dùng vào mục đích này.

Thanh ghi ES trỏ đến khối nhớ cần thay đổi kích thước. Đối với các chương trình của chúng ta (kiểu COM) thì có ES = CS. Còn BX = kích thước mới, tính bằng paragraph (16 byte). Thường thì cho BX = số paragraph tối thiểu cần cho chương trình đầu = (độ lớn chương trình đầu)/ 16 + 1. Hãy đặt AH = 4A hex rồi gọi INT 21 hex. Khi ngắt kết thúc, hãy kiểm tra cờ nhờ xem có lỗi hay không?

- **Hàm 4B hex:** ***Nạp thực hiện một chương trình***

Hàm 4B hex có thể thực hiện hai phục vụ khác nhau. Có thể nạp dữ liệu vào bộ nhớ rồi sử dụng nó hoặc là nạp một chương trình khác rồi

thực hiện nó. Trước khi có thể nạp một chương trình khác, phải giải phóng bớt khối nhớ đã được cấp phát cho chương trình đầu bằng hàm 4A hex. Để nạp một chương trình khác, hãy cho thêm 0 ở cuối đường tên và cho DS:DX trỏ đến đầu đường tên. Hãy đặt AL = 0 nếu muốn nạp chương trình và thực hiện nó, còn đặt AL = 3 nếu chỉ muốn nạp chương trình vào bộ nhớ nhưng không thực hiện. Cặp ES:BX phải trỏ đến khối tham số (parameter block) chứa thông tin về chương trình thứ hai này. (Bảng 1.9 nói về khối tham số cần cho việc nạp và thực hiện). Khối này gồm 7 word (= 14 byte). Word đầu tiên là địa chỉ đoạn của xâu môi trường, địa chỉ được chứa tên của ổ đĩa có tập tin COMMAND.COM (nơi mà DOS sẽ tìm mỗi khi phần tạm thời của COMMAND bị xoá trong bộ nhớ). Xâu này cũng chứa biến PATH (dựa vào đây DOS tìm các tập tin thực hiện được) và biến PROMPT (cho biết dạng của dấu nhắc của DOS) nếu như hai biến này đã được đặt lại trong tập tin AUTOEXEC.BAT .

Ba double word còn lại trỏ đến ba địa chỉ cần truyền cho chương trình sau: Đầu tiên là địa chỉ của dòng lệnh (command line: tại offset 82 hex trong chương trình đầu), tiếp theo là hai địa chỉ của hai khối điều khiển file (File control block: FCB, ở tại các offset 5C và 6C hex). Chúng ta có thể dùng dòng lệnh hay hai khối FCB để truyền các tham số cho chương trình sau. Nếu tham số là một đường tên (có các thư mục con) thì phải dùng dòng lệnh, nếu không thì có thể dùng khối FCB. Tiếp theo hãy đặt AH = 4B hex rồi gọi INT 21 hex. Khi ngắt này kết thúc, hãy kiểm tra lại cờ nhớ xem có lỗi hay không ?.

***Bảng 1.9.* Khối tham số để nạp và thực hiện với hàm 4B hex**

Địa chỉ	*Số byte*	*Ý nghĩa*
0	2	Đoạn chứa xâu môi trường
2	4	Địa chỉ của dòng lệnh
6	4	Địa chỉ của FCB thứ nhất
10	4	Địa chỉ của FCB thứ hai

Nếu muốn chép một tập tin vào bộ nhớ nhưng không thực hiện nó, thì phải dùng một khối tham số khác, chỉ dài 2 word thôi. Word thứ nhất chứa địa chỉ đoạn mà tập tin sẽ được nạp vào, và word thứ hai là hằng số định vị lại (relocation factor) cần thiết cho một tập tin . EXE. Khối tham số này được cho trong bảng 1.10.

Bảng 1.10. **Khối tham số để nạp với hàm 4B hex**

Địa chỉ	*Số byte*	*Ý nghĩa*
0	2	Đoạn mà tập tin được nạp vào
2	2	Hằng số định vị lại cho loại .EXE

- **Hàm 52 hex:*Trả về địa chỉ của vùng DIB***

Địa chỉ của vùng DIB nơi có chứa địa chỉ của MCB đầu tiên ở trường thứ nhất của DIB. Lưu ý là địa chỉ của vùng DIB chứa trong ES:BX trỏ tới trường thứ hai của vùng DIB nên muốn có địa chỉ của MCB đầu tiên phải giảm nội dung ES:BX đi 4 đơn vị.

PHỤ LỤC 2
TẬP LỆNH CHO MODEM SMARTLINK

Khi khởi động, modem ở chế độ command Mode. Ta có thể đặt cấu hình cho Modem bằng cách sử dụng các lệnh AT chuẩn. Lúc này chương trình điều khiển ở chế độ Intractiv (tương tác) hoặc chế độ Terminal. Sau khi nhận mỗi lệnh chỉ định hoạt động, Modem sẽ cho lại mã kết quả để thông báo với người sử dụng. Các lệnh có thể được đưa ra dưới dạng một chuỗi có hoặc không có dấu trắng (Space) cho dễ đọc. Còn khi thực hiện, Modem sẽ bỏ qua các dấu cách này. Trong khi nhập lệnh có thể dùng các phím xoá (delete), hoặc phím Backsspace để sửa lỗi. Các ký tự trong câu lệnh có thể ở dạng chữ hoa hay chữ thường. Tất cả các lệnh đưa ra cho Modem đều bắt đầu bằng hai chữ AT (mã chú ý bắt đầu cho mỗi dòng lệnh, ngoại trừ lệnh A/ và tổ hợp lệnh thoát (+ + + 0 Mã chú ý này được sử dụng để Modem xác định tốc độ bit và cấu trúc của dữ liệu (bit/char, start, Stop). Trong một dòng có thể đặt một hay nhiều lệnh, giữa các lệnh có thể đặt dấu cách hoặc không. Dòng lệnh phải được kết thúc bằng ký tự CR (Carriage Return - về đầu dòng) mà giá trị của nó là #13. Các câu lệnh không có mã kết thúc sẽ bị bỏ qua. Khi nhận xong mã kết thúc, các lệnh nối tiếp sau AT sẽ được tuần tự thực hiện và Modem trả lại mã kết quả tương ứng. Mỗi dòng lệnh không được dài quá 40 ký tự, nếu không sẽ bị lỗi.

- ***Lệnh trả lời (Answer command)***

Modem thực hiện lệnh này tương tự như ta nhấc tổ hợp của máy điện thoại (OFF - HOOK) để trả lời Modem gọi tới. Nếu trong khoảng thời gian được chỉ định trong thanh ghi S7 mà không phát hiện thấy sóng mang (carrier) hoặc khi huỷ bỏ lệnh này bằng cách nhấn một phím bất kỳ, thì kết quả trả lại là "NO CARRIER" và Modem quay về command mode. Nếu phát hiện có sóng mang, Modem trả lại mã " CONNECT" và chuyển sang chế độ dữ liệu (DATA MODE). Trong chế độ dữ liệu, Modem trở về chế độ lệnh khi phát hiện tổ hợp mã thoát (Escape code sequence), thường tổng hợp mã thoát là " + + + ".

- ***Lệnh nhắc lại (Repeat Command)***

Khi nhận lệnh này, Modem thực hiện lại lệnh cuối cùng được lưu trong bộ đệm lệnh. Lệnh này có thể dùng để quay lại số nếu đường bị bận (BUSY). Lệnh này không cần câu lệnh AT ở đầu và không cần kết thúc bằng mã về đầu dòng (CR).

- ***Lệnh lựa chọn chuẩn Bell/CCITT (Bell/CCITT Selection Command)***

BO: Chọn chuẩn CCITT V22.

B1 : Chọn chuẩn Bell 212 - A.

B2 : Chọn chuẩn CCITT V23 (ở chế độ này khi gọi đi, modem truyền với tốc độ 75 bps và nhận với tốc độ 1200 bps. Trong chế độ trả lời sẽ truyền với tốc độ 1200 bps và nhận với tốc độ 75 bps).

- ***Lệnh quay số (Dialing Command)***

Các tham số có thể bao gồm các chữ số từ 0 - 9, touch-tone từ A đến D, *, #, P, R, T, W,!, @, /, dấu ';" và dấu ";". Các dấu ngoặc, dấu chấm, dấu nối, dấu gạch chéo và dấu trắng có thể sử dụng cho dễ đọc, khi thực hiện lệnh modem sẽ bỏ qua chúng. Sau đây là các lệnh cụ thể :

P hoặc T : Lựa chọn việc tạo số bằng chùm xung (PULSE) hay là bằng mã đa tần (TOUCH- TONE). Số được tạo sau khoảng thời gian xác định trong thanh ghi S6. Kiểu quay số mặc định là tạo xung.

, Khi gặp dấu phẩy, modem tạm dừng quay số trong khoảng thời gian định nghĩa trong thanh ghi S8 (sử dụng trong trường hợp từ tổng đài cơ quan (PBX) gọi ra ngoài, tạm nghỉ trước khi tiếp tục quay số ngoài).

; Dấu phẩy đặt cuối cùng dòng lệnh quay số khiển modem chuyển về chế độ lệnh sau khi quay số. Modem sẽ không nối với bất kỳ modem nào, nhưng nó sẽ giữ đường và đợi các lệnh tiếp theo.

@: Bắt modem đợi trong khoảng thời gian chỉ định trong S7 cho ít nhất một lần chuông rung, theo sau là 5 sec im lặng trước klhi thực hiện các lệnh tiếp theo trong dãy gọi. Nếu không phát hiện hồi âm chuông sau thời gian chỉ định trong S7, Modem trả lại mã " NO ANSWER".

! : Nhận dấu này, modem chuyển sang NO - HOOK trong 1/2 sec, sau đó sẽ nối lại.

S=n: Tự động quay số phone number đã được chứa trong NCRAM bằng lệnh &Zn= x, trong đó n = 0,1,2 hoặc 3.

R: Thiết lập việc gọi theo chiều ngược lại, nghĩa là sau khi quay số thành công, modem chuyển ngay sang chế độ trả lời.

W: Cho phép modem chờ tín hiệu mời quay số. Thời gian đợi chỉ định trong S7.

/: Lệnh này cho phép modem đợi trong 0.125 sec.

- ***Lệnh tiếng vọng (Echo Command)***

E1: Modem giử lại DTE các ký tự nhận được trong chế độ lệnh.

EO: Modem không giử lại các ký tự nhận được trong chế độ lệnh.

- ***(H) Điều khiển nhấc đặt máy (Switch - Hook control)***

H1: Điều khiển nhấc máy (OFF Hook).

H0: Điều khiển đặt máy (ON Hook)

Không nên có thêm lệnh nào trong cùng một dòng lệnh tiếp theo sau lệnh (H)

- ***Yêu cầu mã sản phẩm và Checksum (Product code & Checksum)***

IO : Yêu cầu mã sản phẩm. Modem sẽ trả lại một giá trị, như là 240.

11: Lệnh cho modem tính lại checksum trong ROM và trả lại dưới dạng 3 ký tự số ASCII và theo sau là một ký tự về đầu dòng và xuống dòng (CR và LF).

12: Luôn trả lại mã OK.

13: Yêu cầu ngày sản xuất của EPROM. Một kết quả như là 5/1/91-9.10 REVISION sẽ được hiển thị.

- *Âm lượng của loa*

Cho phép người sử dụng nghe thấy tín hiệu của sự bắt tay, của quá trình gọi (hand shake) và quá trình truyền dữ liệu với âm lượng thích hợp.

L, L0, L1: Âm lượng thấp

L2: Âm lượng trung bình

L3: Âm lượng cao.

Ngầm định là L2.

- *Đặt loa ON/OFF*

N.M0: Loa luôn luôn OFF

N1: Loa ON cho tới khi sóng mang được phát hiện

N2: Loa luôn luôn ON

N3: Loa On cho tới khi phát hiện sóng mang, nhưng OFF trong khi quay số .

- *ON - Line*

Lệnh này dùng để đưa modem quay về modem on - line, sau khi trở về mode on - line command bằng tổ hợp mã thoát (+ + +). Điều này cho phép đặt lại cấu hình modem sau khi đã kết nối liên lạc.

O1: Yêu cầu như trên trong mode V22 bis.

- *Mã kết quả*

Lệnh này kích hoạt bộ mã kết quả trong quá trình thực hiện các lệnh.

Q0: Cho phép trả lại mã kết quả.

q1: Không trả lại mã kết quả sau khi thực hiện các lệnh. Các ngoại lệ không chịu ảnh hưởng của Q1 gồm : Các lệnh đặt,

đọc các thanh ghi s, mã đặc trưng, kết quả checksum và kết quả của các mode test với sự tự kiểm tra luôn được trả lại.

Giá trị mặc định là Q0.

(Sr) Các lệnh thanh ghi trực tiếp

Sr?: Đọc giá trị trong thanh ghi sr, kết quả trả lại dưới dạng mã cơ số 10.

Sr =n: Nạp vào thanh ghi sr giá trị bằng n (n từ 0 đến 255). Các giá trị cấu hình của modem được lưu trong thanh ghi s.

- ***Mã kết quả dạng chữ hay dạng số (Verbal/mumeric Result codde)***

Lệnh này chọn kiểu mã kết quả trả lại bởi modem sau khi thực hiện các lệnh.

V0: Mã kết quả dạng số (dạng ngắn)

V1: Mã kết quả dạng chữ (dạng dài).

- ***Cho phép sử dụng mã trả lại mở rộng (Enable Extended Result code)***

X, X0: Báo tin cuộc nối - CONNECT.
Quay số - Không cẩn thận.
Phát hiện bận - Không.

Modem đợi một thời gian xác định trong s6 và sau đó quay số không và để ý đến việc tồn tại hay không tồn tại tín hiệu mời quay số (quay số không cẩn thận).

X1: Báo tin cuộc nối - Bằng thông báo đầy đủ (CONNECT 1200).
Quay số - Không cẩn thận.
Phát hiện bận - Không.

X2: Báo tin cuộc nối - Bằng thông báo đầy đủ.
Quay số - Đợi đến khi có dial - tone.
Phát hiện bận - Không.

Mã trả lại " NO DIAL TONE" sẽ được trả lại nếu trong vòng 5 sec vẫn chưa phát hiện thấy tín hiệu mời quay số (Dial tone).

X3: Báo tín cuộc nối - Bằng thông báo đầy đủ.
Quay số - Không cẩn thận.
Phát hiện bận - có.

Nếu modem phát hiện đường bận, nó trả lại mã " BYSY" và dập máy. Modem đợi hồi âm chuông trong khoảng thời gian xác định trong S7, nếu có hồi âm chuông hoặc hết thời gian trong S7, nó sẽ đợi thêm một khoảng S7 nữa. Mã trả lại dạng đầy đủ.

X4: Báo tin cuộc nối - Bằng thông báo đầy đủ.
Quay số - Đợi khi có dial tone.
Phát hiện bận - Có.

- ***Enable long space Disconnect***

Y1: Cho phép modem ngắt liên lạc khi nhận được một dấu trắng kéo dài 1.6 sec từ modem xa.

Y0: Không cho phép như trên.

- ***Lệnh Reset***

Reset tất cả các giá trị mặc định của modem chứa trong bộ nhớ, trả về mã OK. Modem có hai profile dành cho người sử dụng, ta lựa chọn bằng lệnh ZO hoặc Z1. mọi lệnh trong cùng dòng lệnh sau Z sẽ bị bỏ qua.

Bảng mã kết qủa trả lại được trình bày trên bảng 6.1.

(+ + +) Tổ hợp thoát (Escape sequence)

Đưa modem từ mode dữ liệu về mode lệnh mà không dập máy, vẫn duy trì đường liên lạc. Ba dấu + phải nhập liên tiếp và phải có thời gian nghỉ trước và sau tổ hợp này.

(&C) các tuỳ chọn DCD.

: Duy trì trạng thái ON bền vững cho DCD và bỏ qua mọi trạng thái thực của sóng mang từ modem xa.

&C1: DCD kiểm tra trạng thái của sóng mang dữ liệu từ modem xa và tạo ra tín hiệu tương thích với định nghĩa của CCITT.

Giá trị mặc định là &C1.

(&D) các tuỳ chọn DTR.

&C, &DO : Modem bỏ qua tín hiệu DTR và luôn cho bằng ON

&D1: Modem chuyển từ mode dữ liệu sang mode lệnh nếu DTR chuyển trạng thái từ on sang OFF.

Bảng 6.1

Dạng ngắn	*Dạng dài*	*Ý nghĩa*
0	OK	Dòng lệnh thực hiện không gặp lỗi
1	CONNECT	Đã nối với tốc độ 300 bps
2	RING	Chuông reo tại chỗ
3	NO CARIER	Mất hoặc không thấy sóng mang
4	ERROR	Lỗi trong dòng lệnh
5	CONNECT 1200	Đã nối được với tốc độ 1200 bps
5	CONNECT V23	Nối theo chuẩn V23 1300/75 bps
6	NO DIAL TONE	Không có tín hiệu dial tone trong giới hạn thời gian cho phép
7	BUSY	Gọi tới máy đang bận
8	NO ANSWER	Đối tượng bị gọi không trả lời sau khoảng thời gian cho phép
10	CONNECT 2400	Đã nối với tốc độ 2400 bps
30	CONNECT 4800	Đã nối với tốc độ 4800 bps
32	CONNECT 9600	Đã nối với tốc độ 9600 bps
32	CONNECT 19200	Đã nối với tốc độ 19200 bps
22	CONNECT 1200/RE4	Liên kết MNP class 4
22	CONNECT 1200/REL5	Liên kết MNP class 5
23	CONNECT 2400/REL4	Liên kết MNP class 4
23	CONNECT 2400/REL4	Liên kết MNP class 5
22	CONNECT 1200/V.42	Liên kết V.42
22	CONNECT 1200/V.42.bis	Liên kết V.42 bis
23	CONNECT 2400/V.42	Liên kết V.42
23	CONNECT 2400/V.42 bis	Liên kết MNP class 4

&D2: Việc chuyển trạng thái DRR từ ON sang OFF sẽ đưa modem thành đặt máy, cấm AUTO- ANSWER và chuyển sang trạng thái đợi lệnh.

&D#: Modem áp dụng trạng thái khởi tạo nếu phát hiện ra sự chuyển tín hiệu DTR từ ON sang OFF.

Mọi sự thay đổi trạng thái của DTR phải kéo dài hơn khoảng thời gian xác định trong S25 để modem có thể nhận biết được.

(&F) Đưa về cấu hình của nhà máy:

Các thanh ghi s và các tham số hoạt động của modem được nạp các giá trị mặc định được lưu trong ROM.

&F, &F1: Hoạt động 2400 trực tiếp (đặt modem vào chế độ nối trực tiếp 2400 bps). Nó đặt các tham số sau:

\GO : Huỷ bỏ việc flow cỏntol tại cổng modem.

\J1 : Cho phép điều chỉnh tốc độ bps.

\ N1 : Hoạt động trực tiếp.

\ QO : Huỷ bỏ flow control tại cổng nối tiếp.

\ VO : Chọn bộ mã trả lại chuẩn.

&F2: Đặt cấu hình hoạt động của modem theo MNP class 5 có sử dụng kiểm tra lỗi và nén dữ liệu đồng thời đặt các tham số sau:

\G1 : Cho phép flow control tại cổng Modem.

\ JO : Huỷ bỏ việc điều chỉnh tốc độ bps.

\ N3 : Hoạt động mnp tự động.

\ Q3 : Cho phép flow control tại cổng nối tiếp.

\ V1 : Chọn bộ mã trả lại mở rộng.

&F3: Đặt modem hoạt động theo chuẩn V.42/V.42 bis; kiểm tra lỗi và nén dữ liệu ngược theo MNP 5 và đặt các tham số sau:

\ G1 : Cho phép flow control tại cổng Modem.

\ JO : Huỷ bỏ việc điều chỉnh tốc độ.

\ N7 : Hoạt động theo LAPM/MP tự động.

\ Q3 : Cho phép flow control tại cổng nối tiếp.

\ V1 : Lựa chọn bộ mã trả lại mở rộng.

(&G) Tần số trực.

&G, &GO: Không có tần số trực (guard tone).

&G1: Tần số trực 550 Hz.

&G2: Tần số trực 1800 Hz. Tần số trực sẽ không được phát trong chế độ Bell 212A và 103. Giá trị mặc định là &GO.

(&J) Lựa chọn đầu nối telephone.

&J, &J0: Chọn kiểu Jack RJ-11/RJ-41/RJ-45s.

&J1: Chọn kiểu RJ12/RJ13 cho đường thoại 4 dây.

Giá trị mặc định là &JO.

(&J) Lựa chọn flow control.

&KO: Ngăn cấm flow control

&K3: RTS/CTS flow control.

&K4: XON/XOFF flow contrl

&K5: Bỏ qua XON/XOFf (pass through).

(&L) Lựa chọn đường thuê bao / đường quay số.

&L, &LO : Chọn đường quay số.

&L1: Chọn đường thuê bao.

(&M) Lựa chọn chế độ đồng bộ/ không đồng bộ.

&M,&M): Chọn chế độ truyền không đồng bộ.

&M1: Sử dụng với các terminal có thể hoạt động trong cả hai mode đồng bộ và không đồng bộ, sau khi thực hiện gọi ở mode không đồng bộ, modem chuyển mode đồng bộ. Nếu tín hiệu DTR chuyển từ ON sang OFF, modem sẽ quay trở lại mode không đồng bộ.

&M2: Số phone được lưu trong vùng 0 sẽ được quay khi DTR chuyển từ OFF sang ON (số này lưu bằng lệnh &Z0). Modem quay về mode không đồng bộ khi DTR chuyển từ ON sang OFF.

&M3: Dùng cho quay số bằng tay. Quay số khi DTR bằng OFF, sau đó đưa tín hiệu DTR về On và hạ ống nói xuống. Modem phải nhận biết tín hiệu DTR On trước khi đặt ống nói, nếu không sẽ bị mất đường liên lạc.

(&P) Lựa chọn tốc độ xung (Make/Break Pulse Dia1 Rate).

&PO: Quay số theo US, 10 pps.

&P1: Quay số theo UK, 10pps.

(&Q) Lựa chọn mode hoạt động.

&QO: Modem trực tiếp.

&Q1: Giống như &M1.

&Q2: Giống như & M2.

&Q5: Mode kiểm tra lỗi.

&Q6: Mode bình thường.

(&R): Tuỳ chọn RTS/CTS.

Trong mode không đồng bộ, CTS luôn là ON trong mode lệnh và on-line.

Trong mode đồng bộ, có thể chọn các khả năng sau:

&R, &RO : Đặt CTS để đáp lại RTS. Nên RTS chuyển từ OFF sang ON thì chuyển sang ON.

Việc nhận dữ liệu đồng bộ sẽ bị bỏ qua khi CTS là OFF.

- ***($R) Tuỳ chọn RTS/CTS***

Trong mode không đồng bộ, CTS luôn là ON trong mode lệnh và on - line.

Trong mode đồng bộ, có thể chọn các khả năng sau:

&R, &RO : Đặt CTS để đáp lại RTS. Nêu RTS chuyển từ OFF sang ON thì chuyển sang ON.

Việc nhận dữ liệu đồng bộ sẽ bị bỏ qua khi CTS là OFF.

&R1: Cho phép modem bỏ qua tín hiệu RTS. Lệnh này đặt CTS thành ON khi ở trạng thái on-line và sẵn sàng nhận dữ liệu đồng bộ. CTS giữ On cho đến khi modem hạ máy.

- *(&S) Tuỳ chọn DSR*

&S, &SO : Đặt DSR luôn ON khi modem bật lên.

&S1: DSR hoạt động theo đúng khuyến cáo của CCITT V.42 bis/ V22.

(&T) Các lệnh kiểm tra

&T, &TO: Kết thúc mọi quá trình test đang chạy. Lệnh &T phải là lệnh cuối cùng trong dòng lệnh.

&T1: Khởi tạo analog loopback tại chỗ. Dùng để kiểm tra đường liên lạc giữa modem và DTE. Thanh ghi S18 xác định thời gian test .

&T3: Khởi tạo digital loopback tại chỗ, Test này cho phép dữ liệu gửi từ modem xa đến được quay vòng trong số (digital section) của modem tại chỗ và giử ngược lại modem xa. Như vậy mode này cho phép modem xa kiểm tra tín hiệu số hồi liên từ xa (remote digital loopback test). Hai modem phải được nối trước khi thực hiện test này.

&T5: Cấm modem tại chỗ ưng thuận yêu cầu từ modem xa thực hiện digital loopbak từ xa.

&T6: Cho phép test thiết bị dữ liệu đầu cuối tại chỗ, mo đem tại chỗ và mo dem xa và mạch điện thoại. Terminal tại chỗ gửi message test tới modem xa. Nếu nối được, modem xa sẽ quay vòng dãy thông tin ngược trở lại. Terminal tại chỗ kiểm tra bằng cách so sánh dãy thông tin đó với dãy thông tin gốc và thông báo kết quả test.

&T7: Khởi tạo digital loopback từ xa với quá trình tự kiểm tra tương ứng với khuyến cáo của CCITT V.54. Trong quá trình test, các phần tử dữ liệu được modem kích hoạt bao gồm các tập hợp bit 0 và 1 theo tốc độ lựa chọn áp dụng cho scrrambler (bộ trộn). Khả năng phát hiện các lỗi đặc trưng được nối trực tiếp tới đầu ra của descramler (bộ giải trộn). tại cuối cuộc test một số đếm lỗi 3 chữ số (từ 0 đến 255) được hiển thị. Kết quả 000 báo rằng modem và đường thoại đã được kiểm tra tốt.

&T8: Khởi tạo anolog loopback tại chỗ với chế độ tự động kiểm tra ứng với khuyến cáo CCITT V.54. Trong qúa trình test, các phần tử dữ liệu bao gồm 0 và 1 được khuyếch đại tại chỗ theo tốc độ lựa chọn áp dụng cho scrambler. Khả năng phát hiện các lỗi đặc trưng được nối trực tiếp tới đầu ra của descramler. Nếu modem đang trong tráng thái on - line khi &T8 bắt đầu, sóng mang sẽ bị mất. Test này có ích trong việc kiểm tra mạch thu/ phát của modem tại chỗ.

(&V): Liệt kê cấu hình

Lệnh này cho hiển thị cả hai profile đang tích cực và được cất giữ.

(&W) ghi cấu hình vào vùng RAM không mất (No volatile memory).

Có thể lưu giữ hai cấu hình bằng ệnh &W0 hoặc &W1 trong nội dung một số thanh ghi S (S0, S10, S14, S21, S22, S23, S25, S26 và S27). Các giá trị này được tự động lấy ra khi bật nguồn cho mode, hoặc dùng lệnh zn.

(&X) Nguồn tín hiệu đồng hồ truyền đồng bộ (Synchrronous Transmit Clock source).

Khi truyền đồng bộ, modem yêu cầu dữ liệu truyền (TXD) phải đồng bộ với tín hiệu đồng hồ. Tín hiệu đồng hồ này có thể được cung cấp bởi modem hoặc DTE..

&X, &X0 : Dùng nguồn tín hiệu đồng hồ do modem tạo ra

&X1: Tín hiệu đồng hồ được cung cấp bởi DTE thông qua chân 24 của jac RS 232.

&X2: Tín hiệu đồng hồ được tái tạo từ sóng mang của các tín hiệu nhận được.

(&Y): Lựa chọn profile của người sử dụng

&Y0: Lựa chọn cấu hình 0

&Y1: Lựa chọn cấu hình 1.

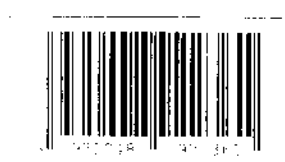
- ***(&Z) Lưu giữ số telephone***

Lệnh &zn = x dùng để lưu giữ số telephone để khi gọi dùng lệnh DS = n Modem có thể lưu giữ 4 số telephone mỗi số dài tối đa 36 chữ số hoặc ký tự. Các chữ số từ 0 đến 9; các ký tự có thể là A, B, C, D, #, * (dùng cho mã đa tần).

TÀI LIỆU THAM KHẢO

1. *Jeffrey Richter.*
   Advanced Windows. Microsoft Press 2000.
2. *Xavier Pacheco.*
   Delphi 2. Developer's Guide. SAMS 1996.
3. *Jeferay Richer.*
   ADVANCED WINDOWS. The Developer's Guide
4. To the WIN32 API for Windows NT and Windows 95.
5. *W.Davis Sch11aderer.*
   G Programer's Guide to NETBIOS. Howard
6. W.SAMS & COMPANY-1989.
7. *Alan R. Miller.*
   Lập trình assembler cho DOS. Nxb ĐH và GD chuyên nghiệp-1992.
8. *Michael Tischer.*
   Cẩm nang lập trình hệ thống. Nxb Thống kê-1992.
9. *F. Halsall.*
   Data communication, computer neywork and open system. Addison Wesley-1992.
10. Windows NT & Windows 95. Nxb Thống kê-1997.
11. *R.Perlman.*
    Interconnection-Bridge and Router. Addison Wesley-1992.
12. *Richard J.Vaccaro.*
    Digital Control. AState-Space-appoarch Internation Edition-1995.
13. SmartLink 1200S MODEM User's Manual. Hayes Microcomputer Products Inc 1995.
14. Fred Cohen, Computer viruses - Theory and Experiments, Online Book at all.net - 2000.
15. Microsoft Coporation, MSDN Collection, 8-2000
16. *Đỗ Xuân Tiến.*
    Kỹ thuật Vi xử lý. Học viện KTQS-1991.
17. *Đỗ Xuân Tiến.*
    Kỹ thuật Số & Kỹ thuật Vi xử lý. Học viện KTQS-1996.
18. *Đỗ Xuân Tiến.*
    Kỹ thuật Vi xử lý và lập trình Assembly cho hệ vi xử lý. NXB Khoa học và Kỹ thuật - 2001.

202236



Giá: 45000đ