

Giáo trình

677.3
v 600

Lập trình hợp ngữ (ASSEMBLY) và

Lập trình hợp ngữ
(ASSEMBLY)

và

MÁY VI TÍNH IBM-PC



YTHA YU & CHARLES MARUT

6T7.3

γ 600

LẬP TRÌNH HỢP NGỮ (ASSEMBLY)

VÀ

MÁY VI TÍNH IBM-PC

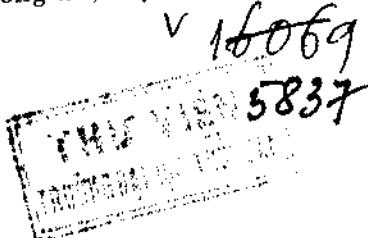
Biên dịch

Quách Tuấn Ngọc

Đỗ Tiến Dũng

Nguyễn Quang Khải

Giáo trình dùng cho sinh viên, kỹ sư các ngành
Công nghệ thông tin, Điện tử, Viễn thông, Tự động hóa ...



Chịu trách nhiệm xuất bản:
Giám đốc: PHẠM VĂN AN
Tổng biên tập: NGUYỄN NHƯ Ý

Biên dịch:
QUÁCH TUẤN NGỌC
ĐỖ TIẾN DŨNG
NGUYỄN QUANG KHẢI

Biên tập và sửa bản in
TUẤN NGỌC

6T7.3

MÃ SỐ: PIK19B8

— 232/1817-98

GD-98

MỞ ĐẦU

Cuốn sách này là sự đúc kết những kinh nghiệm giảng dạy tại trường đại học Hayward, bang California, Mỹ. Mục đích của chúng tôi là viết một tài liệu dễ đọc và bám sát các chuyên mục một cách đầy đủ. Chúng tôi trình bày các kiến thức theo một trình tự lô gíc và tìm hiểu tổ chức của IBM PC với các ví dụ mang tính chất thực tiễn và lý thú.

Ngôn ngữ assembly (hợp ngữ) thực chất chỉ là dạng ký hiệu của ngôn ngữ máy và do đó các chỉ thị (lệnh) của hợp ngữ giao tiếp với phần cứng của máy tính một cách hết sức chặt chẽ. Khi bạn học lập trình bằng hợp ngữ bạn cũng đồng thời nghiên cứu về tổ chức, cấu trúc của máy tính. Cũng bởi vì mỗi quan hệ chặt chẽ của chúng với phần cứng, các chương trình bằng hợp ngữ có thể *chạy nhanh hơn và chiếm ít bộ nhớ hơn* các chương trình ngôn ngữ bậc cao. Các điều kiện này là một mối quan tâm hàng đầu khi viết các chương trình như các chương trình trò chơi.

Do được viết với mục đích để sử dụng giảng dạy ở các trường đại học nên cuốn sách được trình bày theo văn phong sư phạm và bất cứ ai muốn học về IBM PC và muốn tận dụng tối đa những khả năng của nó đều có thể đọc nó dễ dàng. Các thầy giáo sẽ có được các chuyên mục trình bày dưới hình thức mang tính chất sư phạm cao với rất nhiều những ví dụ và bài tập.

Đối tượng đọc cuốn sách này

Bạn có thể là người mới bắt đầu học, không cần thiết phải có trước những hiểu biết về phần cứng máy tính cũng như về lập trình. Bạn hoàn toàn có thể đọc cuốn sách này. Tất nhiên sẽ rất có ích nếu như bạn đã từng lập trình bằng các ngôn ngữ bậc cao như PASCAL, Basic, hay Fortran.

Các yêu cầu về phần cứng và phần mềm :

Để có thể lập được các chương trình và chạy chúng bạn cần có:

1. Một máy IBM PC hoặc tương thích.
2. Hệ điều hành MS_DOS hay PC_DOS.
3. Một chương trình soạn thảo văn bản hoặc một chương trình xử lý văn bản.

4. Đồng thời bạn cũng sẽ phải làm việc với các phần mềm như: chương trình MASM và LINK của MICROSOFT hoặc là TASM và TLINK của Borland.

Giải pháp trình bày

Thế giới của IBM PC và các máy tính tương thích bao gồm rất nhiều loại máy tính khác nhau với các bộ xử lý và cấu trúc khác nhau. Tương tự như vậy có rất nhiều những phiên bản khác nhau của trình biên dịch Assembler và trình gõ rối Debugger. Chúng tôi đã tiến hành những giải pháp sau đây để trình bày:

1. Hướng trọng tâm vào kiến trúc và tập lệnh của các bộ vi xử lý 8086/8088 đồng thời lại có những chương riêng về các bộ vi xử lý cấp cao. Lý do ở đây là vì những công cụ đã học trong lập trình cho 8086/8088 thì cũng khá thông dụng cho toàn bộ họ vi xử lý 80x86 của INTEL bởi vì tập lệnh của các bộ vi xử lý cấp cao thực chất chỉ là sự mở rộng của tập lệnh 8086/8088. Các chương trình được viết cho 8086/8088 đều có thể chạy trên các bộ vi xử lý cấp cao mà không cần một sự thay đổi nào.
2. Các định nghĩa về đoạn (segment) đã được đơn giản hóa và được giới thiệu cùng với MASM 5.0 sẽ được sử dụng ở mọi thời điểm có thể.
3. Môi trường DOS được sử dụng vì nó vẫn còn là hệ điều hành thông dụng nhất trong họ PC.
4. DEBUG được sử dụng cho công việc gõ rối vì nó là một phần của DOS và các đặc điểm chung của nó cũng thông dụng với tất cả các trình gõ rối của Assembly. Chương trình CODEVIEW của MICROSOFT được trình bày trong phụ lục E.

Các đặc trưng của cuốn sách:

Toàn bộ những kiến thức đã được kiểm nghiệm trong các lớp học. Chúng tôi tin rằng những đặc trưng sau đã làm cuốn sách trở nên đặc biệt :

+ Sớm viết các chương trình.

Các bạn thường khao khát được bắt đầu viết các chương trình càng sớm càng tốt, tuy nhiên do các chỉ thị của hợp ngữ có liên quan đến phần cứng máy tính nên đầu tiên bạn cần phải có những hiểu biết cần thiết về kiến trúc máy tính và những kiến thức cơ bản về số nhị phân (hệ đếm cơ số 2) và số thập lục phân (hệ

đếm cơ số 16). Chương trình đầu tiên được trình bày ở chương 1, và đến chương 4 bạn sẽ có những công cụ cần thiết để viết các chương trình đơn giản nhưng lý thú.

+ Thực hiện Vào và Ra:

Việc vào/ra (INPUT/OUTPUT) trong hợp ngữ là việc khó khăn bởi vì tập lệnh quá cơ bản. Chúng ta hãy lập trình cho việc vào/ra bằng các lời gọi các hàm của DOS. Phương pháp này cho phép chúng ta sớm tạo ra các chương trình với các chức năng hoàn thiện.

+ Mã có cấu trúc

Tính ưu việt của lập trình có cấu trúc bằng ngôn ngữ bậc cao cũng có trong hợp ngữ. Chương 6 sẽ chỉ ra rằng các cấu trúc rẽ nhánh và lặp chuẩn của ngôn ngữ bậc cao có thể thực hiện bằng hợp ngữ như thế nào. Một loạt các chương trình được phát triển từ các lệnh giả (pseudocode) bậc cao theo lối từ cao xuống thấp (top-down).

+ Các thuật ngữ

Để có một sự hiểu biết sâu sắc về các ý tưởng của việc lập trình bằng hợp ngữ điều quan trọng là phải nắm chắc những thuật ngữ. Để làm đơn giản việc này, mỗi khi một thuật ngữ mới xuất hiện lần đầu tiên nó được trình bày dưới dạng chữ in đậm và sẽ có trong bảng tổng kết ở cuối chương.

+ Các ứng dụng tiến bộ

Một trong những việc hấp dẫn có thể dễ dàng thực hiện bằng hợp ngữ là thao tác với bàn phím và màn hình. Chúng tôi dành ra hai chương về vấn đề này. Điểm nổi bật đó là sự phát triển một trò chơi tương tự như trò PONG. Một ứng dụng thú vị khác là việc phát triển một chương trình thường trú để hiển thị và cập nhật thời gian.

+ Xử lý số

Các thao tác và chỉ thị của bộ xử lý số học cũng được miêu tả chi tiết.

+ Các bộ xử lý cao cấp

Cấu trúc và hoạt động của các bộ vi xử lý cao cấp được trình bày trong một chương riêng. Bởi vì DOS vẫn còn là hệ điều hành thống trị cho PC nên hầu hết các ví dụ là các ứng dụng cho DOS.

Các giảng viên cần chú ý:

Cuốn sách được chia làm hai phần.

Phần 1: phần cơ sở, bao gồm các chủ đề là cơ sở cho tất cả những ứng dụng của hợp ngữ.

Phần 2: phần nâng cao, bao gồm những chủ đề nâng cao. Bảng dưới đây chỉ ra những chương trong phần 2 liên quan như thế nào vào những chương đã viết trước đó:

| CHƯƠNG | SỬ DỤNG KIẾN THỨC CỦA CHƯƠNG: |
|--------|-------------------------------|
| 12 | 1-10 |
| 13 | 1-11, 12(vài bài tập) |
| 14 | 1-10 |
| 15 | 1-12, 14 |
| 16 | 1-15 |
| 17 | 1-10 |
| 18 | 1-10, 13 |
| 19 | 1-10 |
| 20 | 1-11, 13, 14 |

Các chương trong phần 1 nên được giảng dạy theo đúng trình tự. Nếu sinh viên đã có những cơ sở hiểu biết tốt về kỹ thuật máy tính thì chương 1 có thể giảng sơ qua hoặc giao cho sinh viên về nhà tự đọc. Trong một khoá học 10 tuần mà mỗi tuần gồm 4 giờ giảng, chúng tôi thường thực hiện được 4 chương đầu tiên trong hai tuần đầu, và xây dựng được chương trình đầu tiên vào cuối tuần thứ 2 hoặc đầu tuần thứ 3. Trong vòng 10 tuần chúng tôi thường giải quyết xong các chương từ 1-12 và sau đó tiếp tục chọn những chuyên mục trong các chương từ 13-16 tùy theo thời gian và mối quan tâm cho phép

Về bài tập

Mỗi chương đều kết thúc với hàng loạt những bài tập để nhấn mạnh những khái niệm và những nguyên tắc đã trình bày trong chương đó những bài tập được phân thành nhóm các bài tập thực hành và nhóm các bài tập lập trình

Phân 1

NHỮNG CƠ SỞ LẬP TRÌNH

BẮNG HỌP NGỮ

Chương 1

CÁC HỆ THỐNG MÁY VI TÍNH

Tổng quan

Chương này giới thiệu các kiểu kiến trúc của máy vi tính nói chung và đi sâu hơn vào IBM-PC. Các bạn sẽ học về cấu tạo chính của phần cứng máy tính: Bộ xử lý trung tâm, bộ nhớ và các thiết bị ngoại vi, các phần mềm liên quan và các chương trình. Chúng ta sẽ xem xét một cách chính xác hơn máy tính hoạt động như thế nào khi nó thi hành một lệnh và tranh luận về những ưu điểm chính (và cả các khuyết điểm) của lập trình bằng Hợp ngữ. Nếu bạn là một người sử dụng máy vi tính có kinh nghiệm, bạn sẽ thấy rất quen thuộc với hầu hết các ý kiến bàn luận ở đây. Nếu bạn chưa có kinh nghiệm chương này giới thiệu rất nhiều điều quan trọng cho các phần tiếp theo trong cuốn sách.

1. 1 Các thành phần của một hệ thống vi tính.

Một hệ thống máy vi tính điển hình bao gồm: khối hệ thống (system unit), bàn phím (keyboard), màn hình (display screen) và các ổ đĩa (disk drives). Khối hệ thống thường được xem xét như là “máy tính” bởi vì nó tập trung các bảng vi mạch của máy vi tính. Bàn phím, màn hình và các ổ đĩa được gọi là các thiết bị vào/ra (**I/O devices**) bởi lẽ chúng thực hiện các thao tác vào/ra dữ liệu của máy vi tính. Chúng còn được gọi là các thiết bị ngoại vi (**peripheral devices**).

Hiện nay người ta dùng các vi mạch (IC chip) để xây dựng các mạch điện tử của máy vi tính. Mỗi vi mạch này có thể bao gồm hàng trăm hay thậm chí hàng ngàn đèn bán dẫn (transistor). Các vi mạch này là các mạch số (**digital circuit**) bởi vì chúng thao tác với các mức tín hiệu điện áp rời rạc, mà điển hình là một mức điện áp thấp và một mức điện áp cao. Chúng ta sử dụng ký hiệu 0 và 1 đại diện cho tín hiệu điện áp thấp và cao tương ứng. Các ký hiệu này được

gọi là các chữ số nhị phân (**binary digit**) hay các **bit**. Tất cả các thông tin được xử lý trong máy tính đều được biểu diễn bằng các chuỗi chữ số '0' và '1', nghĩa là chuỗi các bit.

Về mặt chức năng, vi mạch máy tính bao gồm ba phần:

- đơn vị xử lý trung tâm (CPU),
- bộ nhớ
- và các mạch vào/ra.

Trong một máy vi tính, CPU là một bộ xử lý đơn chip, nó còn được gọi là bộ vi xử lý. CPU là "bộ não" của máy tính, nó điều khiển mọi hoạt động trong máy tính. Nó sử dụng bộ nhớ để lưu giữ thông tin, các vi mạch vào/ra để liên lạc với các thiết bị ngoại vi.

Bảng mạch hệ thống (System Board).

Bên trong khôi hệ thống là một bảng mạch chính, chứa bộ vi xử lý và vi mạch nhớ được gọi là bảng mạch hệ thống (**System Board**). Bảng mạch hệ thống còn được gọi là bảng mạch mẹ (**mother board**) bởi vì nó còn chứa các khe cắm (slot) mở rộng dùng để ghép thêm các vỉ vi mạch khác (add-in board). Các mạch vào/ra thường được đặt vào các vỉ ghép thêm.

1.1.1 Bộ nhớ.

Các byte và word.

Thông tin xử lý trong máy tính được lưu giữ trong bộ nhớ của nó. Mỗi phần tử vi mạch nhớ có thể chứa một bit dữ liệu. Thực ra các vi mạch nhớ thường được tổ chức thành các nhóm để có thể chứa được tám bit dữ liệu. Mỗi chuỗi tám bit dữ liệu được gọi là một **byte**. Mỗi mạch nhớ một byte hay gọi tắt là một byte nhớ, được xác định bởi một con số, gọi là địa chỉ (**address**), giống như số nhà trên đường phố. Byte nhớ đầu tiên có địa chỉ 0. Dữ liệu chứa trong byte nhớ gọi là nội dung (**content**). Khi mà nội dung của một byte nhớ được xử lí như là một số, ta thường dùng thuật ngữ giá trị (**value**) để chỉ chúng.

Điều quan trọng là phải hiểu được sự khác nhau giữa địa chỉ và nội dung. Địa chỉ của một byte nhớ là cố định và mỗi byte nhớ trong máy tính có một địa chỉ riêng của nó, tất nhiên các địa chỉ của các byte nhớ khác nhau là khác nhau. Còn nội dung của một byte nhớ lại là đại lượng có thể thay đổi được. Nội dung

byte nhớ chính là dữ liệu được lưu giữ tức thời trong bộ nhớ. Hình 1.3 chỉ ra sự tổ chức các byte nhớ, nội dung của chúng ở đây chỉ là một ví dụ.

Còn một sự khác biệt nữa giữa nội dung và địa chỉ. Đó là trong khi nội dung của một byte nhớ luôn là 8 bit thì số bit của một địa chỉ lại phụ thuộc vào bộ vi xử lý. Ví dụ bộ vi xử lý 8086 của hãng Intel thiết kế với địa chỉ gồm 20 bit, còn bộ vi xử lý 80286 cũng của hãng này lại sử dụng 24 bit địa chỉ. Số lượng bit địa chỉ quyết định số lượng các byte mà bộ xử lý có thể truy nhập.

| ĐỊA CHỈ | NỘI DUNG |
|---------|-----------------|
| ... | |
| 5 | 0 0 1 0 1 0 1 1 |
| 4 | 1 0 1 1 1 1 0 0 |
| 3 | 1 0 0 0 1 0 1 0 |
| 2 | 0 1 1 1 1 0 0 0 |
| 1 | 0 0 1 1 0 1 1 1 |
| 0 | 0 1 1 0 0 1 1 1 |

Hình 1.3: Bộ nhớ được biểu diễn bằng các Byte

Ví dụ 1:

Giả thiết bộ vi xử lý sử dụng 20 bit cho một địa chỉ. Hỏi rằng có thể truy nhập bao nhiêu byte bộ nhớ?

Trả lời:

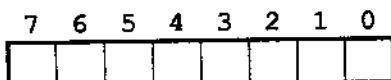
Một bit có thể có hai giá trị. Như vậy 20 bit địa chỉ tạo nên $2^{20} = 1.048.576$ giá trị khác nhau, với mỗi giá trị là một địa chỉ xác định của một byte bộ nhớ. Trong thuật ngữ máy tính, con số 2^{20} được gọi là 1 Mega, 2^{10} được gọi là một Kilo. Vì vậy 20 bit địa chỉ có thể dùng để đánh địa chỉ 1 Mêgabyte hay 1 MB.

Trong một máy vi tính tiêu biểu, hai byte tạo nên một từ (word). Để lưu trữ một từ dữ liệu, IBM PC cho phép một cặp 2 byte nhớ cạnh nhau được xem xét như là một đơn vị nhớ và được gọi là một từ nhớ. Địa chỉ thấp hơn trong hai byte nhớ được dùng làm địa chỉ của từ nhớ. Vì thế từ nhớ có địa chỉ là 5 sẽ được tạo nên từ các byte nhớ có địa chỉ là 5 và 6. Bằng một thông tin khác chứa trong mỗi lệnh, bộ vi xử lý luôn biết rằng một địa chỉ sẽ là địa chỉ chỉ đến một byte hay chỉ đến một từ. Trong cuốn sách này chúng tôi sử dụng thuật ngữ ô nhớ (memory location) để chỉ hoặc một byte nhớ, hoặc một từ nhớ.

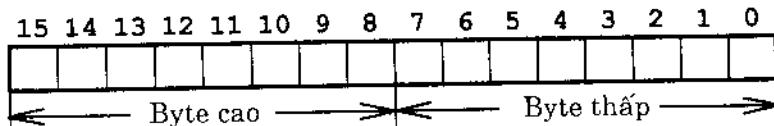
Vị trí của bit.

Hình 1.4 chỉ ra vị trí của bit trong một từ và một byte của bộ vi xử lý. Các vị trí được đánh số từ phải sang trái, bắt đầu từ 0. Trong một từ, các bit từ 0 đến 7 tạo thành byte thấp và các bit từ 8 đến 15 tạo nên byte cao. Để chứa một từ trong bộ nhớ byte thấp ghi vào byte nhớ với địa chỉ thấp, còn byte cao ghi vào byte nhớ với địa chỉ cao hơn.

Vị trí các bit trong byte :



Vị trí các bit trong từ :



Hình 1.4 vị trí các bit trong từ và trong byte

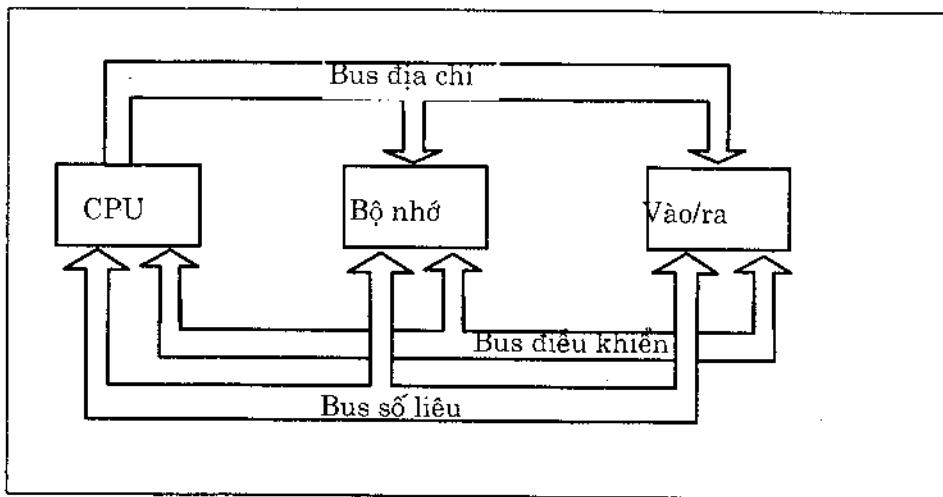
Các thao tác với bộ nhớ.

Bộ xử lý có thể thực hiện hai thao tác với bộ nhớ: Đọc (lấy) nội dung của ô nhớ và ghi (lưu trữ) dữ liệu vào ô nhớ. Trong thao tác đọc, bộ vi xử lý chỉ lấy ra bản sao của dữ liệu còn nội dung nguyên thuỷ của nó không đổi. Trong thao tác ghi, dữ liệu được viết vào sẽ trở thành nội dung mới của ô nhớ và dữ liệu nguyên thuỷ của nó bởi thế sẽ mất đi.

Bộ nhớ RAM và ROM

Có hai loại mạch nhớ: Bộ nhớ có thể ghi và đọc (**RAM**: Random Access Memory) và bộ nhớ chỉ đọc (**ROM**: Read Only Memory). Sự khác nhau giữa chúng là các ô nhớ RAM có thể đọc và viết lên được, còn các ô nhớ ROM như tên của nó đã chỉ ra: chỉ có thể đọc. Sở dĩ nhữ vậy là vì nội dung của các ô nhớ ROM một khi được nạp vào thì không thể thay đổi được.

Các lệnh và dữ liệu của chương trình thường được nạp vào bộ nhớ RAM. Nhưng do nội dung của các ô nhớ RAM bị mất đi khi tắt máy nên mọi giá trị trong RAM phải được ghi lại trên đĩa hay ra máy in sẵn sàng trước. Vì mạch ROM giữ nguyên nội dung của nó khi tắt điện. Vì thế ROM được các nhà sản xuất máy tính sử dụng chứa các chương trình hệ thống. Các chương trình đặt trên ROM này được xem như là phần mềm (firm-ware). Chúng có nhiệm vụ nạp các chương trình khởi động từ đĩa cũng như tự kiểm tra khi bật máy.



Hình 1.5 Các bus trong máy vi tính.

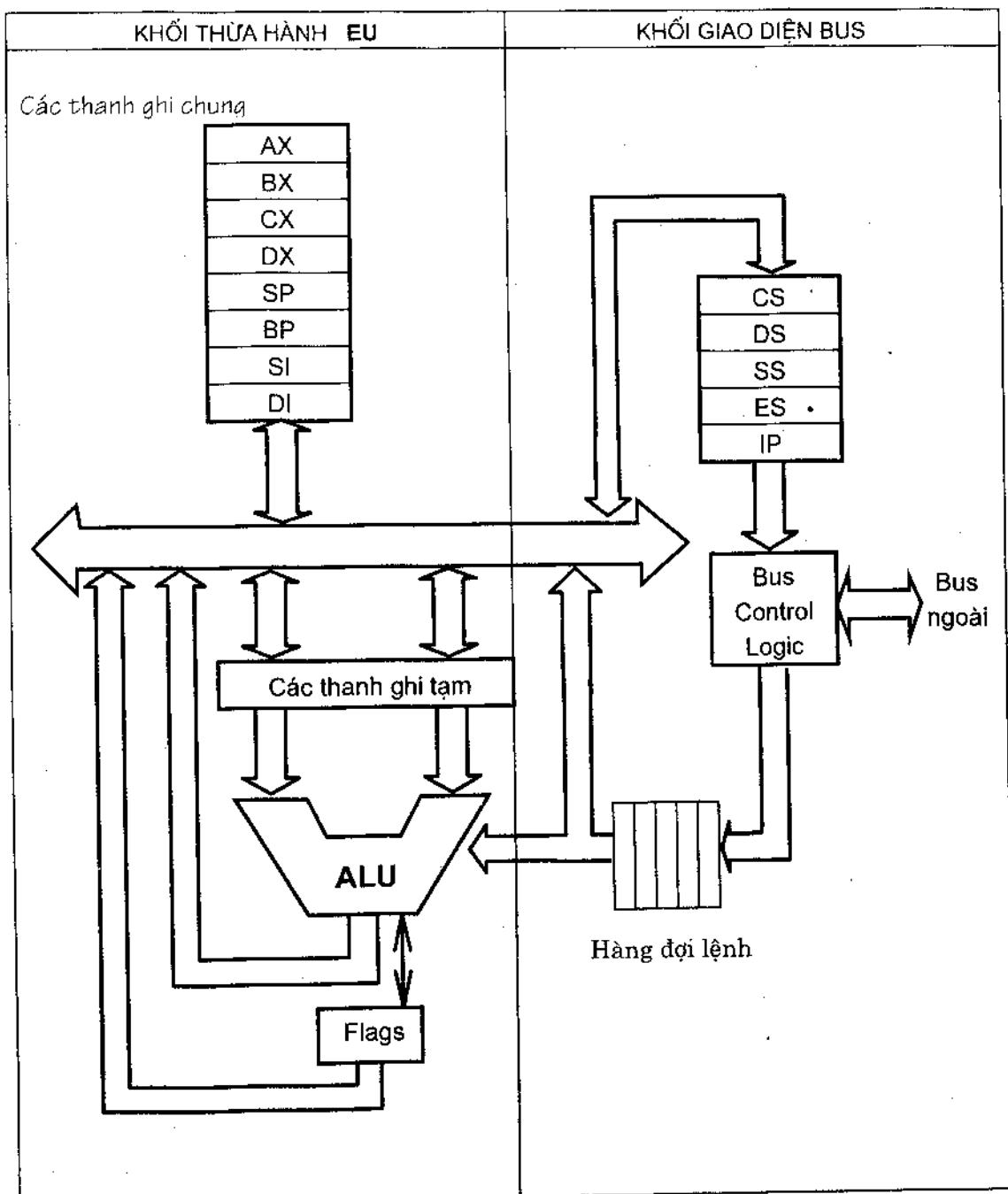
Các loại BUS.

Bộ xử lý liên lạc với bộ nhớ và các vi mạch vào/rã bằng các tín hiệu chạy dọc theo một hệ thống dây dẫn hay cáp nối được gọi là các bus. Các bus này dùng để nối các phần khác nhau. Có ba loại tín hiệu: tín hiệu địa chỉ, điều khiển và tín hiệu số liệu. Và có ba loại bus: **bus địa chỉ**, **bus số liệu** và **bus điều khiển**. Ví dụ để đọc nội dung của một ô nhớ, CPU gửi địa chỉ của ô nhớ trên bus địa chỉ, và nó nhận được số liệu gửi từ vi mạch nhớ trên bus số liệu. Một tín hiệu điều khiển được sử dụng để báo cho bộ nhớ thực hiện thao tác đọc. CPU gửi tín hiệu điều khiển trên bus điều khiển. Hình 1.5 là sơ đồ nối bus trong một máy vi tính.

1.1.2 Bộ xử lý trung tâm (CPU)

Như đã nói ở trên CPU là bộ não của máy tính. CPU điều khiển máy tính bằng cách thực hiện các chương trình chứa trong bộ nhớ. Chương trình đó có thể là một chương trình hệ thống hay một chương trình ứng dụng. Trong bất cứ trường hợp nào, mỗi lệnh mà CPU thi hành là một chuỗi các bit (với bộ vi xử lý

INTEL8086 các lệnh có chiều dài từ 1 đến 6 byte). Ngôn ngữ viết bằng các số 0 và 1 này được gọi là **ngôn ngữ máy**.



Hình 1.6: Cấu trúc mạch vi xử lý 8086

Mỗi CPU có một tập các lệnh đặc trưng duy nhất tạo thành **hệ lệnh** (instruction set). Để hạ giá thành, hệ lệnh được thiết kế đơn giản, ví dụ như

lệnh cộng 2 số hay lệnh di chuyển một số từ vị trí này đến vị trí khác. Một điều đáng ngạc nhiên là những công việc vô cùng phức tạp sẽ được máy tính thực hiện chỉ bằng một chuỗi các thao tác cơ bản.

Sau đây chúng ta sẽ sử dụng bộ vi xử lý INTEL8086 như là một ví dụ cho CPU. Hình 1.6 chỉ ra tổ chức của nó với 2 phần chính: khôi thừa hành (execution unit) và khôi giao diện bus (bus interface unit).

Khôi thừa hành(EU).

Như tên gọi đã chỉ ra, mục đích của khôi thừa hành là thực hiện các lệnh. Nó chứa một vi mạch gọi là **khối số học và lô gic** (ALU). ALU có thể thực hiện các phép tính số học (+, -, *, /) và lô gic (AND, OR, NOT). Trong CPU có một số ô nhớ được gọi là **thanh ghi** (registers), chúng chứa các dữ liệu cho các phép tính. Mỗi thanh ghi giống như một ô nhớ ngoại trừ một điều là chúng được đặt tên thay vì dùng số để chỉ địa chỉ. EU có 8 thanh ghi để chứa dữ liệu, chúng có tên là: AX, BX, CX, DX, SI, DI, BP và SP. Chúng ta sẽ trả lại vấn đề này trong chương 3. Ngoài ra EU còn chứa các thanh ghi tạm thời để lưu giữ các toán hạng cho ALU và một thanh ghi cờ (FLAGS) với các bit riêng rẽ phản ánh kết quả của mỗi phép tính.

Khôi giao diện BIU (Bus Interface Unit).

Khôi giao diện BIU làm đơn giản việc liên lạc giữa EU và bộ nhớ hay các vi mạch vào/ra. Nó có nhiệm vụ gửi các địa chỉ, số liệu và tín hiệu điều khiển vào các BUS. Các thanh ghi của nó có tên là CS, DS, ES, SS và IP. Chúng lưu giữ địa chỉ các ô nhớ. Thanh ghi IP (Instruction Pointer: **con trỏ lệnh**) chứa địa chỉ của lệnh tiếp theo sẽ được EU thi hành.

EU và BIU được nối bằng các bus nội bộ để làm việc với nhau. Khi EU đang thi hành một lệnh, BIU nạp 6 byte mã lệnh tiếp theo và đặt chúng vào hàng đợi lệnh. Thao tác này gọi là **nhận trước các lệnh**. Mục đích của nó là làm tăng tốc độ bộ vi xử lý. Nếu EU cần liên lạc với bộ nhớ hay các thiết bị ngoại vi, BIU sẽ tiến hành ‘treo’ các lệnh nhận trước và thực hiện các thao tác cần thiết.

1.1.3 Cổng vào/ra

Các thiết bị ngoại vi được nối với máy tính thông qua các mạch vào ra. Mỗi mạch này chứa vài thanh ghi gọi là **cổng Vào/Ra (I/O Port)**. Một số được dùng cho dữ liệu trong khi số khác được dùng cho các lệnh điều khiển. Giống như các

ô nhớ, các cổng Vào/Ra cũng có các địa chỉ và được nối với hệ thống BUS. Tuy nhiên các địa chỉ này chỉ được xem như địa chỉ Vào/Ra và chỉ có thể được sử dụng trong các lệnh Vào/Ra. Điều này cho phép CPU phân biệt cổng Vào/Ra với ô nhớ.

Các cổng Vào/Ra thực hiện chức năng là trung gian để trao đổi giữa CPU và thiết bị ngoại vi. Dữ liệu được nạp vào từ một thiết bị ngoại vi sẽ được gửi vào một cổng, tại đó chúng có thể được đọc bởi CPU. Khi xuất, CPU viết dữ liệu ra cổng, vì mạch vào/ra sau đó sẽ chuyển dữ liệu đến thiết bị ngoại vi.

Các cổng nối tiếp và song song.

Dữ liệu truyền giữa một cổng và một thiết bị ngoại vi có thể là từng bit một (nối tiếp) hay 8 hoặc 16 bit một lúc (song song). Một cổng song song yêu cầu nhiều dây nối hơn trong khi cổng nối tiếp thì lại chậm hơn. Các thiết bị chậm như bàn phím thường nối với cổng nối tiếp, ngược lại các thiết bị nhanh như ổ đĩa thường nối với cổng song song. Tuy nhiên có vài thiết bị chẳng hạn như máy in thì có thể nối với cả cổng nối tiếp và cổng song.

1.2 Việc thực hiện các lệnh.

Instruction: được dịch là lệnh hoặc chỉ thị, trong khi **Command** cũng được dịch là lệnh. Trong tiếng Việt, từ chỉ thị được hiểu theo nghĩa như là chỉ thị của thủ trưởng. Trong tiếng Anh, từ Command là lệnh máy tính do người sử dụng điều khiển máy tính qua bàn phím, hoặc chọn từ Menu (bảng chọn), hoặc từ con chuột ... Trong khi đó Instruction được hiểu là một câu lệnh viết trong một ngôn ngữ lập trình nào đó. Thí dụ các lệnh FORMAT, TYPE ... của DOS là loại Command.

Để hiểu xem cách CPU hoạt động, chúng ta hãy xem một lệnh (Instruction) được thực hiện ra sao. Trước tiên cần phải nhắc lại rằng một lệnh của máy tính gồm 2 phần: một phần **mã lệnh** (opcode) và một phần là **toán hạng** (Operands). Mã lệnh xác định kiểu của lệnh, còn các toán hạng thường là các địa chỉ bộ nhớ của dữ liệu để lệnh thực hiện. Các bước thực hiện một lệnh máy (một chu kỳ nhận lệnh và thực hiện) của CPU sẽ là như sau:

Nhận lệnh

1. Nhận một chỉ thị từ bộ nhớ.
2. Giải mã lệnh để xác định thao tác cần thực hiện.
3. Nhận dữ liệu từ bộ nhớ nếu cần thiết.

Thực hiện lệnh

4. Thực hiện thao tác trên dữ liệu.
5. Lưu trữ kết quả vào bộ nhớ nếu cần thiết.

Để xem quá trình này được thực hiện như thế nào chúng ta hãy duyệt qua việc thực hiện một chỉ thị điển hình của ngôn ngữ máy chẳng hạn chỉ thị cộng nội dung của thanh ghi AX với nội dung của từ nhớ có địa chỉ 0. Mã máy có dạng sau:

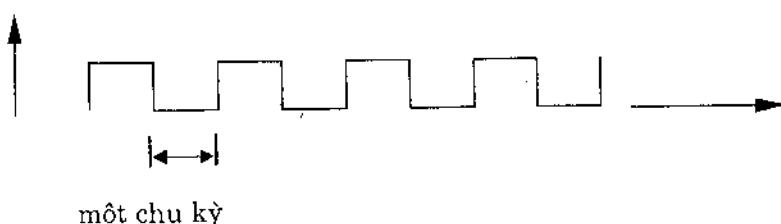
00000001 00000110 00000000 00000000

Chúng ta giả thiết rằng byte đầu tiên của lệnh đã được lưu tại vị trí ô nhớ có địa chỉ nằm trong con trỏ lệnh IP trước khi thực hiện lệnh.

1. **Nhận lệnh.** Vào đầu chu kỳ, BIU đặt yêu cầu đọc bộ nhớ vào bus điều khiển và địa chỉ của chỉ thị vào bus địa chỉ. Bộ nhớ trả lời bằng cách gửi nội dung của ô nhớ đã được chỉ ra (đó chính là mã lệnh cần thực hiện) qua bus dữ liệu. Vì mã lệnh dài 4 byte trong khi 8086 chỉ có thể đọc mỗi lần một word do đó cần có 2 thao tác đọc. Sau khi nhận dữ liệu, CPU thêm 4 vào nội dung thanh ghi con trỏ lệnh IP để nó trỏ đến lệnh tiếp theo.
2. **Giải mã lệnh.** Trong khi nhận lệnh, một vi mạch giải mã lệnh trong EU sẽ giải mã lệnh và xác định đó là thao tác cộng với từ nhớ tại địa chỉ 0.
3. **Nhận dữ liệu từ bộ nhớ.** EU báo cho BIU lấy nội dung của từ nhớ địa chỉ 0. BIU gửi địa chỉ 0 qua bus địa chỉ và yêu cầu đọc bộ nhớ một lần nữa lại được gửi đi qua bus điều khiển. Nội dung của từ nhớ địa chỉ 0 được gửi trở lại EU qua bus dữ liệu và được đặt vào thanh ghi giữ.
4. **Thực hiện thao tác.** Nội dung của thanh ghi giữ và nội dung của thanh ghi AX được gửi đến ALU. ALU sẽ thực hiện phép cộng và giữ lại kết quả.
5. **Lưu kết quả.** EU yêu cầu BIU lưu lại kết quả. Để thực hiện việc đó, BIU gửi yêu cầu ghi vào bộ nhớ qua bus điều khiển, địa chỉ 0 qua bus địa chỉ và tổng cần lưu qua bus dữ liệu. Giá trị mới là tổng mới tính được sẽ ghi đè vào nội dung trước đó của từ nhớ tại địa chỉ 0, tất nhiên nội dung cũ sẽ bị mất đi.
Chu kỳ bây giờ lại được lặp lại cho chỉ thị có địa chỉ chưa trong thanh ghi con trỏ lệnh IP.

Việc định thời gian (timing).

Ví dụ trên đã chỉ ra rằng mặc dù các chỉ thị máy rất đơn giản nhưng việc thực hiện chúng lại tương đối phức tạp. Để đảm bảo các bước được thực hiện theo đúng trình tự, một vi mạch đồng hồ điều khiển bộ vi xử lý bằng cách tạo ra một chuỗi **xung đồng hồ** hay **xung nhịp** (clock pulses) như chúng ta thấy trên hình 1.7. Khoảng thời gian giữa 2 xung gọi là **chu kỳ đồng hồ** hay chu kỳ xung nhịp và số xung nhịp trong khoảng thời gian một giây gọi là **tần số đồng hồ** hay **tốc độ đồng hồ** được tính bằng megahertz (MHz). Một MHz bằng một triệu chu kỳ (xung) trong thời gian một giây. Các máy tính IBM PC ban đầu có tần số đồng hồ 4,77MHz nhưng máy đời mới nhất có tần số đồng hồ là 100 MHz.



Hình 1.7: Chuỗi xung đồng hồ.

Các vi mạch của máy tính được kích hoạt bởi các xung nhịp, có nghĩa là chúng chỉ thực hiện các thao tác khi có xung nhịp xuất hiện. Mỗi bước trong chu kỳ nhận lệnh và thực hiện cần một chu kỳ xung nhịp hoặc hơn nữa. Chẳng hạn 8086 cần 4 chu kỳ xung nhịp để thực hiện thao tác đọc bộ nhớ còn phép nhân thì có thể cần đến hơn 70 chu kỳ xung nhịp. Nếu chúng ta tăng tốc độ của vi mạch đồng hồ, bộ vi xử lý có thể thao tác nhanh hơn nhưng mỗi bộ vi xử lý đều có tốc độ đồng hồ tối đa cho phép, mà nếu vượt qua đó tốc độ này thì máy có thể sẽ không còn hoạt động đúng nữa.

1.3 Các thiết bị ngoại vi

Các thiết bị ngoại vi dùng để đưa thông tin vào và ra khỏi máy tính. Các thiết bị ngoại vi cơ bản là đĩa từ, bàn phím, màn hình, máy in.

Các loại đĩa từ

Chúng ta đã thấy rằng nội dung của RAM sẽ bị mất khi tắt máy do đó đĩa từ được sử dụng để lưu trữ thông tin lâu dài. Có 2 loại đĩa từ: đĩa mềm (**floppy**)

✓ 16069

5837

disk hay **diskettes**) và đĩa cứng (**hard disk**). Thiết bị dùng để viết và đọc thông tin trên đĩa từ gọi là **ổ đĩa (disk drive)**.

Đĩa mềm có các loại kích thước đường kính $5\frac{1}{4}$ inch (inso) và $3\frac{1}{2}$ inch.

(1 inch=2,56 cm). Các đĩa mềm thường nhẹ và tiện vận chuyển. Bạn có thể cất đĩa vào nơi an toàn hay đem sử dụng nó ở máy tính khác. Dung lượng dữ liệu mà một đĩa mềm có thể lưu giữ phụ thuộc vào loại đĩa, vào khoảng từ 360 KB (KiloByte) đến 1.44 MB (MegaByte). 1 KiloByte(KB) bằng 2^{10} Byte.

Đĩa cứng và ổ đĩa của nó được đặt trong một hộp kín và không thể đưa ra khỏi máy tính và bởi vậy nó còn được gọi là đĩa cố định (fixed disk). Đĩa cứng có dung lượng lớn hơn nhiều đĩa mềm, tiêu biểu là các kích thước 20, 40, 120, 210, 340, 410, 500 MB cho đến 1, 2 GB. Các chương trình truy nhập thông tin trong đĩa cứng cũng nhanh hơn rất nhiều so với đĩa mềm. Các thao tác đĩa được trình bày trong chương 19

Bàn phím

Người sử dụng dùng bàn phím để đưa thông tin vào máy tính. Bàn phím có các phím thông thường của một máy chữ ngoài ra còn các phím số, phím điều khiển và phím chức năng. Bàn phím có hẳn một bộ vi xử lý riêng mà nó sẽ gửi các tín hiệu đã được mã hoá đến máy tính mỗi khi có phím được ấn hay được nhả.

Khi một phím được ấn, ký tự tương ứng thường xuất hiện trên màn hình, nhưng thực ra không hề có sự liên lạc trực tiếp giữa bàn phím với màn hình. Chương trình đang chạy sẽ nhận dữ liệu từ bàn phím và chương trình phải gửi dữ liệu ra màn hình để ký tự được hiển thị. Đến chương 12 chúng ta sẽ tìm hiểu cách điều khiển bàn phím.

Màn hình

Màn hình là thiết bị ra chuẩn của máy tính. Thông tin hiển thị trên màn hình được tạo ra bởi một vỉ mạch trong máy tính gọi là vỉ mạch ghép nối màn hình (video adapter). Hầu hết các vỉ mạch ghép nối đều có khả năng tạo ra cả các ký tự văn bản lẫn các hình ảnh đồ họa. Một số màn hình có khả năng hiển thị màu. Chúng ta sẽ tìm hiểu về các thao tác trong chế độ văn bản trong chương 12 và chế độ đồ họa trong chương 16.

Máy in

Mặc dù màn hình là một thiết bị đem lại sự phản hồi nhanh chóng bằng hình ảnh nhưng các thông tin trên màn hình không tồn tại lâu dài. Các máy in

mặc dù chậm nhưng cho ra các sản phẩm tồn tại lâu dài hơn. Sản phẩm của các máy in được xem như các bản sao cứng (hard copy).

Có 3 loại máy in phổ biến là máy in bằng các bánh xe tròn có khắc chữ (daisy wheel), máy in ma trận điểm (dot matrix) và máy in laze (lazer). Sản phẩm của các máy in dập đĩa (daisy wheel) tương đối giống như máy chũ. Máy in ma trận điểm in ra các ký tự là tổng hợp của các điểm. Tuỳ thuộc vào số điểm được sử dụng cho mỗi ký tự một số máy in có thể in ra các chữ có chất lượng khá cao. Ưu điểm của máy in ma trận điểm là nó có thể in ra các ký tự với các phông (font) chữ khác nhau cũng như các ký tự mở rộng.

Các máy in laze cũng in ra các ký tự là tổng hợp của các điểm nhưng với độ phân giải cực cao (300 trăm điểm trên một inch) do đó có thể in ra các chữ rất đẹp. Máy in laze đắt nhưng không thể thiếu được trong lĩnh vực chế bản. Một khuyết điểm của máy in laze là nó không gây tiếng ồn như các máy in thông thường.

1.4 Các ngôn ngữ lập trình

Phần mềm sẽ điều khiển các thao tác của máy tính. Khi máy tính được bật lên, nó luôn luôn ở trong quá trình thực hiện các lệnh. Để có thể hiểu cặn kẽ về các thao tác của máy tính chúng ta cần nghiên cứu các chỉ thị.

Ngôn ngữ máy

Các bộ vi xử lý chỉ có thể thực hiện các chỉ thị của ngôn ngữ máy mà như chúng ta đã thấy chúng là các chuỗi bit. Dưới đây là một chương trình ngắn bằng ngôn ngữ máy cho IBM PC :

| Chỉ thị máy | Thao tác |
|----------------------------|--|
| 10100001 00000000 00000000 | lấy nội dung của từ nhớ 0 và lưu nó vào thanh ghi AX |
| 00000101 00000100 00000000 | cộng 4 vào AX |
| 10100011 00000000 00000000 | lưu nội dung thanh ghi AX vào từ nhớ 0 |

Bạn có thể dễ thấy rằng viết các chương trình bằng ngôn ngữ máy là một công việc chán ngắt và rất dễ gây sai sót.

Hợp ngữ

Một ngôn ngữ tiện lợi hơn để sử dụng đó là hợp ngữ (ngôn ngữ Assembly). Trong hợp ngữ chúng ta có các tên biểu tượng (gợi nhớ) để biểu diễn các thao tác, các thanh ghi và các ô nhớ. Nếu ô nhớ địa chỉ 0 được biểu thị bởi chữ A thì chương trình trên đây viết lại bằng hợp ngữ cho IBM PC sẽ có dạng như sau:

Chỉ thị của hợp ngữ

MOV AX, A

Lời bình

;lấy nội dung của từ nhớ 0 và cất
;nó vào thanh ghi AX

ADD AX, 4

;cộng 4 vào AX

MOV A, AX

;gửi nội dung của thanh ghi AX
;vào từ nhớ 0

Các chương trình viết bằng hợp ngữ phải được dịch sang ngôn ngữ máy trước khi CPU có thể thực hiện chúng. Một chương trình biên dịch sẽ dịch mỗi dòng lệnh ở hợp ngữ sang một chỉ thị của ngôn ngữ máy.

Ngôn ngữ bậc cao

Mặc dù rằng viết một chương trình bằng hợp ngữ dễ hơn bằng ngôn ngữ máy nhưng đây vẫn là một công việc khó vì tập lệnh của nó vẫn còn quá đơn giản. Đó là nguyên nhân tại sao các ngôn ngữ bậc cao như FORTRAN, PASCAL, C ... được phát triển. Các ngôn ngữ bậc cao khác nhau được thiết kế cho các ứng dụng khác nhau nhưng nhìn chung chúng đều cho phép người lập trình viết các chương trình gần với ngôn ngữ tự nhiên hơn so với hợp ngữ.

Một chương trình biên dịch (**compiler**) sẽ dịch các chương trình bằng ngôn ngữ bậc cao ra mã máy. Công việc biên dịch phức tạp hơn việc hợp dịch bởi vì nó đòi hỏi cả việc dịch các biểu thức toán học phức tạp và các câu lệnh ở dạng ngôn ngữ tự nhiên sang các chỉ thị máy đơn giản. Một lệnh của ngôn ngữ bậc cao thông thường được dịch thành nhiều chỉ thị của ngôn ngữ máy.

Các ưu điểm của ngôn ngữ bậc cao

Có rất nhiều lý do để người lập trình thích chọn các ngôn ngữ bậc cao để viết chương trình hơn là chọn hợp ngữ.

- *Thứ nhất* là vì ngôn ngữ bậc cao gần với ngôn ngữ tự nhiên và dễ dàng đổi thuật toán ở dạng ngôn ngữ tự nhiên sang chương trình bằng ngôn ngữ bậc cao hơn là sang chương trình bằng hợp ngữ.
- *Thứ hai* là một chương trình bằng hợp ngữ thường chứa nhiều lệnh hơn là một chương trình tương tự nhưng viết bằng ngôn ngữ bậc cao do đó sẽ tốn nhiều thời gian để viết một chương trình bằng hợp ngữ hơn.
- *Thứ ba* là vì mỗi loại máy tính có một hợp ngữ duy nhất của riêng nó, vì thế chương trình bằng hợp ngữ có thể bị giới hạn trong một loại máy nhất định. Trong khi đó, chương trình bằng ngôn ngữ bậc cao có thể chạy tốt trên mọi loại máy mà có trình biên dịch của ngôn ngữ ấy.

Các ưu điểm của hợp ngữ.

Lý do chính để viết chương trình bằng hợp ngữ là tính hiệu quả: vì hợp ngữ rất gần gũi với ngôn ngữ máy nên một chương trình viết tốt bằng hợp ngữ sẽ tạo ra một chương trình *ngắn* và *chạy nhanh hơn* ở dạng ngôn ngữ máy. Mặt khác có một số thao tác chẳng hạn như đọc hoặc viết trực tiếp vào các ô nhớ hay các cổng vào ra có thể thực hiện dễ dàng bằng hợp ngữ nhưng có thể là không thể thực hiện được với một số ngôn ngữ bậc cao.

Thực ra không phải lúc nào lập trình viên cũng phải chọn lựa giữa hợp ngữ và ngôn ngữ bậc cao bởi vì rất nhiều ngôn ngữ bậc cao chấp nhận các chương trình con viết bằng hợp ngữ. Điều này có nghĩa là các phần quan trọng của chương trình có thể được viết bằng hợp ngữ và phần còn lại được viết bằng ngôn ngữ bậc cao.

Ngoài những lý do kể trên còn một lý do khác để học hợp ngữ. Đó là chỉ có thể bằng cách nghiên cứu hợp ngữ bạn mới thực sự nhận thức được cách ‘nghỉ’ của máy tính và tại sao lại có những việc nhất định xảy ra trong máy cũng như cách thực hiện chúng trong máy. Các ngôn ngữ bậc cao có khuynh hướng lờ đi những chi tiết của một chương trình đã biên dịch ra ngôn ngữ máy là chương trình mà máy tính thực sự thực hiện. Đôi khi một sự thay đổi nhỏ của chương trình có thể làm tăng đáng kể thời gian thực hiện chương trình hay sự tràn sổ học không mong đợi có thể xảy ra. Những điều đó có thể hiểu được dưới góc độ của hợp ngữ.

Mặc dù ở đây chúng ta nghiên cứu hợp ngữ cho IBM PC những kỹ thuật mà bạn sẽ học cũng là những kỹ thuật điển hình được sử dụng trong bất kỳ loại hợp ngữ nào. Việc nghiên cứu một hợp ngữ khác sẽ tương đối dễ dàng sau khi bạn đọc xong cuốn sách này.

1.5 Một chương trình bằng hợp ngữ.

Để có được một hình ảnh về một chương trình hợp ngữ chúng tôi đưa ra ở đây một ví dụ đơn giản. Chương trình sau đây cộng nội dung của 2 ô nhớ biểu thị bằng A và B. Tổng số sẽ được lưu vào ô nhớ SUM.

Chương trình PGM1_1.ASM

```
TITLE      PGM1_1: SAMPLE PROGRAM
.MODEL     SMALL
.STACK    100H
.DATA
A        DW      2
B        DW      5
SUM      DW      ?
.CODE
MAIN PROC
; khởi tạo DS
    MOV AX, @DATA
    MOV DS, AX
; cộng các số
    MOV AX, A          ;AX chứa A
    ADD AX, B          ;AX chứa A+B
    MOV SUM, AX         ;SUM=A+B
; trả về DOS
    MOV AX, 4C00H
    INT 21H
MAIN ENDP
END MAIN
```

Các chương trình bằng hợp ngữ chứa các dòng lệnh. Dòng lệnh có thể là một *chỉ thi* được thực hiện khi chương trình chạy hay là *dẫn hướng biên dịch* (*directive*) cho chương trình biên dịch. Ví dụ. MODEL SMALL là một dẫn hướng biên dịch xác định kích thước của chương trình. MOV AX, A là một chỉ thi. Mọi thứ ở sau dấu chấm phẩy (,) là các lời bình và trình biên dịch sẽ bỏ qua chúng. Chương trình trên đây gồm 3 phần hay 3 đoạn: đoạn ngắn xếp, đoạn dữ liệu và đoạn mã. Chúng được bắt đầu tương ứng bằng các dẫn hướng biên dịch :.STACK; .DATA; .CODE ..

Đoạn ngắn xếp được sử dụng để lưu trữ tạm thời các địa chỉ và dữ liệu. Nếu đoạn ngắn xếp không được khai báo, một thông báo lỗi sẽ được tạo ra, do đó phải có đoạn ngắn xếp cho dù chương trình không sử dụng chúng.

Các biến được khai báo trong đoạn dữ liệu. Mỗi biến sẽ được gán với một ô nhớ và có thể được khởi tạo. Ví dụ:

A DW 2

dành riêng ra một từ nhớ cho biến có tên gọi A và khởi tạo nó với trị ban đầu là 2 (DW có nghĩa là ‘Define Word’). Tương tự

B DW 5

dành riêng ra một từ nhớ cho biến B và khởi tạo nó với trị ban đầu là 5 (giá trị khởi tạo ban đầu là tùy ý).

SUM DW ?

dành riêng ra một từ nhớ cho biến SUM nhưng không khởi tạo nó.

Các lệnh của chương trình được đặt trong đoạn mã. Các lệnh thường được tổ chức thành các đơn vị gọi là các thủ tục (procedure). Chương trình trên chỉ có 1 thủ tục với tên gọi MAIN. Thủ tục MAIN được bắt đầu bằng dòng MAIN PROC và kết thúc bằng dòng

MAIN ENDP

Thủ tục chính bắt đầu và kết thúc với các chỉ thị cần thiết để khởi tạo thanh ghi DS và trở về hệ điều hành DOS. Mục đích của chúng sẽ được giải thích trong chương 4. Các chỉ thị dưới đây dùng để cộng A với B và lưu kết quả vào biến SUM:

| | |
|-------------|--------------|
| MOV AX, A | ;AX chứa A |
| ADD AX, B | ;AX chứa A+B |
| MOV SUM, AX | ;SUM=A+B |

Chỉ thị MOV AX, A sao nội dung của từ nhớ A sang thanh ghi AX.

Chỉ thị ADD AX, B cộng nội dung của B vào AX và bây giờ AX chứa tổng bằng 7.

Chỉ thị MOV SUM, AX lưu kết quả vào biến tổng.

Trước khi chạy chương trình trên máy tính, nó phải được biên dịch ra chương trình ngôn ngữ máy. Các bước sẽ được giải thích ở chương 4. Vì chương trình trên chưa có các lệnh xuất dữ liệu, chúng ta không thấy được kết quả trên màn hình; nhưng chúng ta vẫn có thể chạy từng bước chương trình bằng một chương trình gõ rồi chẵng hạn chương trình DEBUG.

Các thuật ngữ tiếng Anh

| | |
|--|---|
| Add_in board(card) | Bảng mạch thường chứa các vi mạch vào/ra hay phần bộ nhớ mở rộng và được nối với bảng mạch chính. |
| Address | Con số xác định vị trí ô nhớ. |
| Address bus | Tập hợp các đường dẫn điện tử cho các tín hiệu địa chỉ. |
| Arithmetic and Logic Unit (ALU) | Vi mạch xử lý trung tâm nơi thực hiện các thao tác số học và lô gic. |
| Assembler | Trình biên dịch sẽ dịch chương trình hợp ngữ ra chương trình ngôn ngữ máy. |
| Binary digit | Các ký hiệu chỉ nhận 2 giá trị 0 và 1. |
| Bit | Chữ số nhị phân. |
| Bus | Các đường dây nối bộ vi xử lý trung tâm (CPU) với bộ nhớ và các cổng vào/ra. |
| Bus Interface Unit (BIU) | Một phần của bộ vi xử lý trung tâm giúp đỡ việc thông tin giữa CPU bộ nhớ và các cổng vào/ra. |
| Byte | 8 bit. |
| Central Processing Unit (CPU) | Vi mạch xử lý chính trong máy tính. |
| Clock period | Khoảng thời gian giữa 2 xung đồng hồ. |
| Clock pulse | Xung dùng để đồng bộ các vi mạch trong máy tính. |
| Clock speed(Clock speed) | Số xung đồng hồ trong 1 giây được tính bằng Mê ga hec (MHz). |
| Compiler | Trình biên dịch các chương trình ngôn ngữ bậc cao ra ngôn ngữ máy. |
| Contents | Dữ liệu lưu trong thanh ghi hay một ô nhớ. |
| Control bus | Tập hợp các đường dẫn điện tử cho các tín hiệu điều khiển. |

| | |
|---------------------------------|---|
| Data bus | Tập hợp các đường dẫn điện tử cho các tín hiệu số liệu. |
| Digital circuit | Các vi mạch làm việc với các mức tín hiệu điện áp rời rạc. |
| Disk drive | Thiết bị dùng để đọc và ghi dữ liệu trên đĩa. |
| Execution Unit (EU) | Một phần của bộ vi xử lý trung tâm có nhiệm vụ thực hiện các lệnh. |
| Expansion slots | Các khe cắm trên bảng mạch mẹ ở đó các bảng mạch khác có thể ghép vào. |
| Fetch_execute cycle | Chu kỳ mà trong đó CPU thực hiện một chỉ thị |
| Firmware | <ul style="list-style-type: none"> • Các phần mềm của nhà sản xuất thường được chứa trong ROM. |
| Fixed disk | Đĩa cố định được chế tạo bằng kim loại. |
| Floppy disk | Đĩa có khả năng thay đổi và có độ mềm dẻo. |
| Hard copy | Sản phẩm của máy in. |
| Hard disk | Đĩa cứng (fixed disk). |
| I/O devices | Các thiết bị ngoại vi quản lý việc vào ra dữ liệu của máy tính, tiêu biểu là bàn phím, màn hình, và máy in. |
| I/O ports | Cổng vào/ra - các vi mạch làm nhiệm vụ trung gian giữa máy tính và thiết bị ngoại vi. |
| Instruction Pointer (IP) | Một thanh ghi của bộ vi xử lý trung tâm chứa địa chỉ của lệnh tiếp theo. |
| Instruction set | Tập các lệnh CPU có thể thực hiện. |
| Kylobyte, KB | 2^{10} hay 1024 byte |
| Machine language | Các lệnh được mã hoá như các chuỗi bit- ngôn ngữ của máy tính |
| Mega | Đơn vị biểu thị 1 triệu nhưng trong máy tính, thuật ngữ một mega bằng 2^{20} hay 1.048.576 |
| Megabyte, MB | 2^{20} hay 1.048.576 byte |
| Megahertz, MHz | 1.000.000 chu kỳ trong một giây. |
| Memory byte(circuit) | Vị mạch nhỏ có khả năng lưu được 1 byte dữ liệu. |
| Memory location | Vị trí một byte hay một word nhớ. |
| Memory word | Hai byte nhớ. |

| | |
|---------------------------------------|---|
| Microprocessor | Đơn vị xử lý đặt trên một mạch vi điện tử. |
| Motherboard | Bảng mạch chính của máy tính. |
| Opcode | Các mã số hay ký hiệu biểu thị kiểu thao tác của lệnh. |
| Operand | Dữ liệu của một chỉ thị. |
| Peripheral(device) | Các thiết bị ngoại vi. |
| Random Access Memory (RAM) | Bộ nhớ truy nhập ngẫu nhiên(có thể đọc hoặc ghi trên đó). |
| Read_Only Memory (ROM) | Bộ nhớ chỉ đọc. |
| Register | Các vi mạch dùng để lưu trữ thông tin. |
| System board | Bảng mạch chính (motherboard). |
| Video adapter | Bộ nối ghép điều khiển màn hình, nó thực hiện việc đổi số liệu của máy tính thành tín hiệu hình ảnh cho màn hình. |
| Word | 16 bit |

Bài tập

1. Giả sử các byte bộ nhớ từ 0 đến 4 có nội dung như sau:

| Địa chỉ | Nội dung |
|---------|----------|
| 0 | 01101010 |
| 1 | 11011101 |
| 2 | 00010001 |
| 3 | 11111111 |
| 4 | 01010101 |

a. Biết rằng một word bằng 2 byte, cho biết nội dung của:

- Từ nhớ ở địa chỉ 2 ?
- Từ nhớ ở địa chỉ 3 ?
- Từ nhớ mà byte cao của nó là byte ở địa chỉ 2

b. Cho biết

- Bit 7 của byte 2 ?
- Bit 4 của word 3 ?
- Bit 4 của byte 2 ?
- Bit 11 của word 2 ?

2. Một nibble bằng 4 bit, một byte bao gồm một nibble cao và một nibble thấp tương tự như byte cao và byte thấp của một word. Sử dụng số liệu của bài tập trên cho biết nội dung của:

- a. Nibble thấp của byte 1 ?
- b. Nibble cao của byte 4 ?

3. Có 2 loại bộ nhớ ROM và RAM cho biết loại bộ nhớ nào

- a. Chứa chương trình của người sử dụng ?
- b. Chứa chương trình để khởi động máy ?
- c. Có thể thay đổi bởi người sử dụng ?
- d. Giữ được số liệu kể cả khi tắt máy ?

4. Nêu chức năng của:

- a. Bộ vi xử lý ?
- b. Các loại bus ?

5. Trình bày chức năng 2 phần của CPU là EU và BIU.
6. Trong bộ vi xử lý trung tâm chức năng của IP là gì ? ALU có nhiệm vụ gì ?
7. Các cổng vào/ra dùng để làm gì ? Sự khác nhau giữa chúng với các ô nhớ ?
8. Chiều dài cực đại tính bằng byte của mỗi lệnh của các máy tính IBM PC có bộ vi xử lý trung tâm 8086 bằng bao nhiêu ?
9. Giả sử một chỉ thị máy chuyển nội dung của thanh ghi AX CPU vào một từ nhỏ.
Những gì sẽ xảy ra trong
 - a. Chu kỳ nhận lệnh
 - b. Chu kỳ thực hiện lệnh
10. Cho biết
 - a. Ba ưu điểm của việc lập trình bằng ngôn ngữ bậc cao
 - b. Các ưu điểm cơ bản của lập trình bằng hợp ngữ

Chương 2

PHƯƠNG PHÁP BIỂU DIỄN SỐ VÀ KÝ TỰ

Tổng quan

Như chúng ta đã thấy trong chương 1, các vi mạch của máy tính chỉ có thể xử lý thông tin dưới dạng mã nhị phân. Trong chương này chúng ta hãy xem các số được biểu diễn dưới dạng nhị phân như thế nào hay còn gọi là hệ thống số nhị phân. Đồng thời chúng tôi cũng giới thiệu một phương pháp ngắn gọn để biểu diễn thông tin nhị phân được gọi là hệ thống số cơ số 16 (số hexa-decimal hay gọi tắt là số Hex).

Việc chuyển đổi giữa các số dạng nhị phân, số hexa và số thập phân sẽ được trình bày trong phần 2.2. Phần 2.3 sẽ trình bày về phép cộng và phép trừ trong các hệ thống số này. Phần 2.4 sẽ cho chúng ta thấy các số âm được biểu diễn như thế nào và kích thước vật lý cố định của byte hay word có ảnh hưởng ra sao với việc biểu diễn các số. Chương 2 sẽ được kết thúc bằng việc nghiên cứu xem các ký tự được máy tính mã hoá và sử dụng ra sao.

2.1 Các hệ đếm

Trước khi tìm hiểu về việc biểu diễn các số dưới dạng nhị phân chúng ta hãy xem lại hệ đếm số quen thuộc: hệ thập phân. Đây là một ví dụ về hệ đếm phụ thuộc vị trí (positional number system), có nghĩa là mỗi chữ số gắn liền với một luỹ thừa của 10 tuỳ thuộc vào vị trí của nó trong số đó. Số thập phân 3932 sẽ bằng 3 nghìn, 9 trăm, 3 chục và 2 đơn vị, có thể viết:

$$3932 = 3 \cdot 10^3 + 9 \cdot 10^2 + 3 \cdot 10^1 + 2 \cdot 10^0$$

Trong hệ đếm phụ thuộc vị trí một số b nào đó được chọn làm *cơ số* và các ký hiệu biểu diễn số được gán bằng các số trong phạm vi từ 0 đến $b-1$. Ví dụ trong

hệ thập phân có 10 ký hiệu cơ bản (các chữ số) là: 0, 1, 2, 3, 4, 5, 6, 7, 8 và 9, cơ số ở đây là 10 và được biểu diễn là 10

Hệ đếm nhị phân

Trong hệ đếm nhị phân cơ số là 2 và chỉ có 2 chữ số là 0 và 1. Ví dụ chuỗi số nhị phân 10110 biểu diễn số:

$$1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = 22$$

Số 2 được biểu diễn trong hệ đếm 2 bằng 10 (xin đọc là một - không, đừng đọc là mười).

Hệ đếm thập lục phân (16)

Các số dưới dạng nhị phân thường dài và khó trình bày, khó nhớ. Ví dụ phải cần 16 bit để biểu diễn nội dung của một từ nhớ trong máy tính. Nhưng các số thập phân lại khó đổi sang dạng nhị phân. Khi chúng ta viết chương trình Hợp ngữ chúng ta thường dùng cả 2 hệ đếm: nhị phân, thập phân và cả một hệ đếm thứ 3 là hệ 16 còn gọi tắt là số hex. Việc sử dụng số hex có ưu điểm là việc chuyển đổi giữa số hex và số nhị phân tương đối dễ dàng. Hệ đếm hex là hệ đếm có cơ số bằng 16 nên các chữ số của nó là: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, và F. Vì hết các ký hiệu chữ số nên ta dùng thêm các chữ cái để biểu diễn: Các chữ cái từ A đến F tương ứng biểu diễn các số từ 10 đến 16. Tiếp theo chữ cái F (15_{10}) sẽ là cơ số của hệ đếm (tức là số 16_{10}) biểu diễn dưới dạng số hex sẽ là 10 (xin đọc là một-không, không đọc là mười vì mười là số trong hệ đếm mười trong khi 10 ở đây có giá trị là mười-sáu nếu tính trong hệ thập phân).

Vì 16 bằng luỹ thừa bậc 4 của 2 nên mỗi chữ số hex tương ứng với một số 4 bit như chúng ta thấy trong bảng 2.1. Điều này có nghĩa là nội dung của một byte 8 bit có thể biểu diễn ngắn gọn bằng 2 chữ số hex, chính điều này làm cho số hex trở nên tiện dụng trong máy tính với xu hướng sử dụng thông tin theo các byte.

Bảng 2.1 Các chữ số hex và giá trị nhị phân tương ứng

| CÁC CHỮ SỐ HỆ 16 (HEXA) | GIÁ TRỊ NHỊ PHÂN |
|-------------------------|------------------|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |
| A | 1010 |
| B | 1011 |
| C | 1100 |
| D | 1101 |
| E | 1110 |
| F | 1111 |

Bảng 2.2 Tương quan giữa các số thập phân, nhị phân và số hex.

| SỐ THẬP PHÂN | SỐ NHỊ PHÂN | SỐ HEX |
|--------------|-------------|--------|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 10 | 2 |
| 3 | 11 | 3 |
| 4 | 100 | 4 |
| 5 | 101 | 5 |
| 6 | 110 | 6 |
| 7 | 111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | A |
| 11 | 1011 | B |
| 12 | 1100 | C |
| 13 | 1101 | D |
| 14 | 1110 | E |
| 15 | 1111 | F |
| 16 | 10000 | 10 |
| 17 | 10001 | 11 |
| 18 | 10010 | 12 |
| 19 | 10011 | 13 |

| | | | |
|-------|--|--|------|
| 20 | 10100 | | 14 |
| 21 | 10101 | | 15 |
| 22 | 10110 | | 16 |
| 23 | 10111 | | 17 |
| 24 | 11000 | | 18 |
| 25 | 11001 | | 19 |
| 26 | 11010 | | 1A |
| 27 | 11011 | | 1B |
| 28 | 11100 | | 1C |
| 29 | 11101 | | 1D |
| 30 | 11110 | | 1E |
| 31 | 11111 | | 1F |
| 32 | 100000 | | . |
| . | . | | . |
| . | . | | . |
| 256 | 100000000 (bắt đầu dài quá, khó nhớ...) | | 100 |
| . | . | | . |
| . | . | | . |
| 1024 | . | | 400 |
| . | . | | . |
| . | . | | . |
| 32767 | . | | 7FFF |
| 32768 | . | | 8000 |
| . | . | | . |
| . | . | | . |
| 65535 | . | | FFFF |

Bảng 2.2 chỉ ra mối quan hệ giữa các số nhị phân, thập phân và số hex. Các bạn nên giành ra ít phút để ghi nhớ 16 dòng đầu tiên của bảng vì bạn sẽ thường xuyên phải biểu diễn các số nhỏ trong cả 3 hệ đếm.

$$\begin{aligned}
 1 \text{ Kylobyte (1 KB)} &= 1.024 = 400h \\
 64 \text{ Kylobyte (64 KB)} &= 65.536 = 10000h \\
 1 \text{ Mégabyte (1 MB)} &= 1.048.576 = 100000h
 \end{aligned}$$

Một vấn đề khi làm việc với các hệ đếm khác nhau là ý nghĩa của các ký hiệu được sử dụng. Ví dụ như bạn đã thấy 10 có nghĩa là mười trong hệ đếm thập phân, mười sáu trong hệ đếm 16 và hai trong hệ đếm nhị phân. Trong cuốn sách này chúng tôi sử dụng những qui ước sau ở những chỗ dễ nhầm lẫn: các số hex được kết thúc bằng chữ h ví dụ 123h, các số nhị phân được kết thúc bằng chữ b ví dụ 10b

Còn các số thập phân được kết thúc bằng chữ d ví dụ 179d.

2.2 Việc chuyển đổi giữa các hệ đếm

Khi làm việc với Hợp ngữ chúng ta thường phải xử lý các số ở hệ đếm này nhưng lại viết ra ở một hệ đếm khác.

Chuyển đổi các số nhị phân và số hex ra các số thập phân

Giả sử có một số hex 8A2D, có thể viết như sau:

$$\begin{aligned}8A2Dh &= 8 \cdot 16^3 + A \cdot 16^2 + 2 \cdot 16^1 + D \cdot 16^0 \\&= 8 \cdot 16^3 + 10 \cdot 16^2 + 2 \cdot 16^1 + 13 \cdot 16^0 = 35373d\end{aligned}$$

Tương tự số nhị phân 11101 có thể viết như sau :

$$11101b = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 29d$$

Trên đây là một phương pháp để đổi các số nhị phân và số hex ra số thập phân nhưng có một phương pháp nhanh hơn là thực hiện các phép nhân lồng nhau. Ví dụ:

$$\begin{aligned}8A2Dh &= 8 \cdot 16^3 + A \cdot 16^2 + 2 \cdot 16^1 + D \cdot 16^0 \\&= ((8 \cdot 16 + A) \cdot 16 + 2) \cdot 16 + D \\&= ((8 \cdot 16 + 10) \cdot 16 + 2) \cdot 16 + 13 \\&= 35373d\end{aligned}$$

Phương pháp này có thể thực hiện nhanh chóng bằng máy tính: Bạn chỉ việc nhân chữ số hex đầu tiên với 16 rồi cộng với chữ số hex thứ hai, nhân kết quả nhận được với 16 rồi cộng với chữ số hex thứ 3, lại nhân kết quả với 16 rồi cộng với chữ số hex tiếp theo và cứ như vậy.

Thủ tục chuyển đổi số nhị phân ra số thập phân cũng tương tự như vậy, chỉ thay vì nhân với 16 ở đây bạn nhân với 2.

Ví dụ 2.1 Đổi 11101b ra số thập phân

Trả lời :

$$\begin{array}{cccccc}1 & 1 & & 1 & 0 & 1 \\= 1 \cdot 2 + 1 \rightarrow 3 \cdot 2 + 1 \rightarrow 7 \cdot 2 + 0 \rightarrow 14 \cdot 2 + 1 = 29d.\end{array}$$

Ví dụ 2.2 Đổi 2BD4h ra số thập phân

Trả lời :

$$\begin{array}{ccccccc} & 2 & & B & & D & \\ & = 2 * 16 + 11 & \rightarrow & 43 * 16 + 13 & \rightarrow & 701 * 16 + 4 & = 11220d \end{array}$$

Đổi số thập phân ra số nhị phân và số hex

Giả sử chúng ta muốn đổi 11172 ra số hex. Đáp số là 2BA4h có thể nhận được bằng phương pháp sau: trước tiên chúng ta chia 11172 cho 16 nhận được thương số là 698 và số dư là 4. Bởi vậy:

$$11172 = 698 * 16 + 4$$

Số dư 4 chính là chữ số hàng đơn vị của số biểu diễn dạng số hex của 11172. Bây giờ chúng ta đem chia 698 cho 16 thương số là 43 và số dư là 10 = Ah. Do vậy:

$$698 = 43 * 16 + Ah$$

Số dư Ah cũng chính là một trong các chữ số biểu diễn dạng số hex của 11172. Chúng ta cứ tiếp tục quá trình trên mỗi lần đem chia thương số mới nhận được cho 16 cho đến khi thương số nhận được bằng 0. Số dư nhận được mỗi lần chính là các chữ số biểu diễn dạng số hex của 11172. Dưới đây là các phép tính:

$$\begin{aligned} 11172 &= 698 * 16 + 4 \\ 698 &= 43 * 16 + 10(Ah) \\ 43 &= 2 * 16 + 11(Bh) \\ 2 &= 0 * 16 + 2 \end{aligned}$$

Bây giờ chỉ việc chuyển các số dư sang dạng số hex và ghép chúng lại với nhau theo thứ tự ngược lại ta sẽ nhận được kết quả 2BA4h.

Bạn cũng có thể làm như trên để đổi số thập phân sang dạng nhị phân có điều là phải lặp lại việc chia cho 2 thay vì cho 16

Ví dụ 2.3 Đổi 95 thành số nhị phân

Trả lời :

$$95 = 47 * 2 + 1$$

$$47 = 23 * 2 + 1$$

$$\begin{aligned}
 23 &= 11 * 2 + 1 \\
 11 &= 5 * 2 + 1 \\
 5 &= 2 * 2 + 1 \\
 2 &= 1 * 2 + 0 \\
 1 &= 0 * 2 + 1
 \end{aligned}$$

Ghép các số dư nhận được theo thứ tự ngược lại ta có: $95d = 1011111b$.

Chuyển đổi giữa các số nhị phân và số hex

Để đổi số hex sang số nhị phân chúng ta chỉ việc biểu diễn các chữ số của nó dưới dạng nhị phân.

Ví dụ 2.4 Đổi $2B3Ch$ thành số nhị phân

Trả lời :

$$\begin{array}{cccc}
 2 & B & 3 & C \\
 = 0010 & 1011 & 0011 & 1100 \\
 = 0010101100111100
 \end{array}$$

Để đổi số nhị phân sang số hex, ta chỉ cần làm theo thứ tự ngược lại có nghĩa là nhóm 4 chữ số của số nhị phân lại theo thứ tự lần lượt từ phải qua trái. Sau cùng chỉ việc đổi mỗi nhóm thành chữ số hex tương ứng.

Ví dụ 2.5 Đổi $100110110100110b$ sang số hex.

Trả lời:

$$100110110100110 = 100 \quad 1101 \quad 1010 \quad 0110 = 4DA6h$$

2.3 Phép cộng và trừ trong các hệ đếm

Đôi khi bạn sẽ cần phải thực hiện các phép cộng và trừ với các số nhị phân và số hex. Bởi vì các thao tác này các bạn đã thuộc lòng đối với các số thập phân, chúng ta hãy xem xét lại quá trình này để xem những vấn đề liên quan đến các phép tính.

Phép cộng

Hãy xem phép cộng hai số thập phân dưới đây:

$$\begin{array}{r}
 3516 \\
 + 1972 \\
 \hline
 5488
 \end{array}$$

Để nhận được chữ số hàng đơn vị của tổng chúng ta chỉ việc tiến hành phép tính $6 + 2 = 8$. Để nhận được chữ số hàng chục ta tính $1 + 7 = 8$, tiếp theo ta có $5 + 9 = 14$ ta viết 4 vào hàng trăm của tổng và nhớ 1 sang cột tiếp theo, ở cột này chúng ta có $3 + 1 + 1 = 5$ và viết 5 vào tổng đến đây phép tính đã hoàn thành.

Sở dĩ chúng ta có thể thực hiện được phép cộng các số thập phân một cách dễ dàng như vậy là vì đã từ lâu chúng ta ghi nhớ bảng tính cộng các số nhỏ.

Bảng 2.3A là bảng tính cộng cho các số hex nhỏ. Chẳng hạn để tính $Bh + 9$ chúng ta chỉ tìm hàng có chứa B và cột có chứa 9 tại giao điểm của chúng ta đọc được 14h.

Bằng cách sử dụng bảng cộng có thể thực hiện các phép cộng số hex giống như đã làm với số thập phân. Chẳng hạn chúng ta cần tính phép cộng hai số hex sau đây:

$$\begin{array}{r}
 5B39h \\
 + 7AF4h \\
 \hline
 D62Dh
 \end{array}$$

Trong cột hàng đơn vị ta có $9h + 4h = 13d = Dh$. Trong cột tiếp theo $3h + Fh = 12h$ viết 2 và nhớ 1 sang cột bên trái, tại đây ta có $Bh + Ah + 1 = 16h$ viết 6 nhớ 1 sang cột cuối cùng ở đó ta có $5h + 7h + 1 = Dh$ và phép cộng đã thực hiện xong.

Phép cộng nhị phân cũng được tiến hành như vậy nhưng dễ hơn nhiều vì bảng cộng nhị phân rất nhỏ (bảng 2.3B). Chẳng hạn để tính tổng

$$\begin{array}{r}
 1001100110b \\
 + 1011b \\
 \hline
 1001110001b
 \end{array}$$

Với cột đơn vị ta có $0 + 1 = 1$, viết 1, ở cột tiếp theo $1 + 1 = 10$ viết 0 nhớ 1 sang cột tiếp theo ở đó ta có $0 + 1 + 1 = 10$ viết 0 nhớ 1 và quá trình cứ như vậy.

Bảng 2.3A Bảng cộng số hex

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 10 |
| 2 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 10 | 11 |
| 3 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 10 | 11 | 12 |
| 4 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 10 | 11 | 12 | 13 |
| 5 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 |
| 6 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 |
| 7 | 7 | 8 | 9 | A | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 8 | 8 | 9 | A | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 9 | 9 | A | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| A | A | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| B | B | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A |
| C | C | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B |
| D | D | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C |
| E | E | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D |
| F | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E |

Bảng 2.3B Bảng cộng số nhị phân

$$\begin{array}{r} & \begin{array}{r} 0 & 1 \end{array} \\ \begin{array}{r} 0 \\ 1 \end{array} & \left| \begin{array}{r} 0 & 1 \\ 1 & 10 \end{array} \right. \end{array}$$

Phép trừ

Chúng ta hãy bắt đầu với phép trừ số thập phân

$$\begin{array}{r} \text{bb} \\ 9145 \\ - 7283 \\ \hline 1862 \end{array}$$

Trong hàng đơn vị chúng ta có $5-3 = 2$ để tính toán cho hàng chục trước tiên chúng ta vay 1 từ hàng trăm (để nhớ điều này chúng ta có thể ghi b ở trên cột hàng trăm) sau đó tính được $14-8 = 6$.

Khi tính đến hàng trăm chúng ta một lần nữa lại phải vay 1 từ hàng nghìn và tính được $11-2-1 = 8$. Trong cột cuối cùng chúng ta có $9-7-1 = 1$.

Phép trừ số hex cũng có thể được thực hiện tương tự, chẳng hạn :

| |
|-------------|
| bb |
| D26F |
| <u>BA94</u> |
| 17DB |

Chúng ta bắt đầu với $Fh \cdot 4h = Bh$ để tiếp tục (cột 16) chúng ta phải vay 1 từ hàng cột thứ 3 và tính được $16h \cdot 9h = ?$ một phương pháp dễ dàng là xem ở hàng 9 của bảng 2.3A và nhận thấy rằng $16h$ xuất hiện ở cột D nghĩa là $9 + Dh = 16h$ và do đó $16h \cdot 9h = Dh$. Trong cột thứ 3 sau khi vay 1 chúng ta có $12h \cdot Ah \cdot 1h = 11h \cdot Ah$ tương tự như trên ta có $11h \cdot Ah = 7h$. Trong cột cuối cùng ta có $Ch \cdot Bh = 1$.

Bây giờ chúng ta hãy xem đến phép trừ số nhị phân, ví dụ:

| |
|-------------|
| bb |
| 1101 |
| <u>0111</u> |
| 0010 |

Trong cột đơn vị dễ dàng tính được $1-1 = 0$. Chúng ta cũng phải vay 1 để tính cột hai và có $10-1=1$. Để tính cột 4 (cột thứ 3) chúng ta lại phải vay 1 từ cột tiếp theo: $10-1-1=0$. Trong cột cuối cùng chúng ta có $0-0=0$.

2.4 Cách biểu diễn các số nguyên trong máy tính

Phần cứng của máy tính cần giới hạn kích thước của các số để có thể lưu nó trong các thanh ghi hay trong các ô nhớ. Trong phần này chúng ta hãy xem các số nguyên được lưu trữ ra sao trong một byte 8 bit hoặc một word 16 bit. Trong chương 18 chúng ta sẽ nghiên cứu về vấn đề các số thực được lưu trữ thế nào. Trong phần dưới đây chúng ta cần chú ý đến 2 bit đặc biệt của một byte hay một word: bit có trọng lượng cao nhất, còn gọi là bit nặng nhất (Most Significant Bit) hay msb, đó là bit nằm ở tận cùng bên trái. Trong một word, bit nặng nhất là bit 15 còn trong một byte đó là bit 7. Tương tự như vậy bit có trọng lượng thấp nhất hay còn gọi là bit nhẹ nhất (Least Significant Bit) hay lsb, đó là bit nằm ở tận cùng bên phải, bit 0.

2.4.1 Số nguyên không dấu

Số nguyên không dấu (unsigned integer) biểu diễn các số nguyên dương. Số nguyên không dấu rất thích hợp để biểu diễn các đại lượng luôn dương chẳng hạn địa chỉ của ô nhớ, bộ đếm hay mã ASCII của ký tự (chúng ta sẽ thấy sau này). Bởi vì các số nguyên không dấu được định nghĩa là các số dương nên không cần dùng bit nào của nó để biểu diễn dấu, do đó 8 bit của một byte hay 16 bit của một word đều được dùng để biểu diễn số.

Số nguyên không dấu lớn nhất có thể chứa trong một byte là 1111 1111 = FFh = 255

Đây rõ ràng không phải là con số lớn vì thế người ta thường dùng word để lưu các số nguyên không dấu. Số nguyên không dấu lớn nhất mà một word 16 bit có thể lưu được là 1111 1111 1111 1111 = FFFFh = 65535. Đây là con số đủ lớn cho hầu hết các mục đích. Trong trường hợp không đủ chúng ta có thể sử dụng hai word hoặc hơn nữa.

Chú ý rằng khi bit lsb của nguyên là 1 thì nó là một số lẻ ngược lại nếu lsb là 0 thì đó là một số chẵn.

2.4.2 Số nguyên có dấu

Số nguyên có dấu (signed integer) có thể là số dương hoặc số âm. Bit nặng nhất msb được dùng để biểu diễn dấu của số, msb bằng 1 biểu diễn số âm, msb bằng 0 biểu diễn số dương.

Các số âm được lưu trong máy tính dưới dạng số bù 2, để hiểu được khái niệm số bù 2, trước tiên chúng ta hãy cùng nhau tìm hiểu khái niệm số bù 1.

Số bù 1

Số bù 1 của một số nguyên nhận được bằng cách lấy phần bù của các bit của nó, có nghĩa là đảo bit: thay mỗi bit 0 bằng 1 và ngược lại, 1 bằng 0. Từ đây về sau chúng ta giả sử các số là các số 16 bit.

Ví dụ 2.6. Tìm số bù 1 của $5 = 0000\ 0000\ 0000\ 0101$.

Trả lời : Số bù một của 5 là 1111 1111 1111 1010

Chú ý rằng nếu ta cộng 5 vào số bù một của nó ta nhận được

1111 1111 1111 1111

Số bù 2

Để nhận được số bù 2 của một số nguyên ta chỉ việc cộng 1 vào số bù 1 của nó.

Ví dụ 2.7 Tìm số bù 2 của 5 ?

Trả lời Theo kết quả ở trên,

$$\begin{array}{r}
 \text{Số bù 1 của 5} \quad 1111 \ 1111 \ 1111 \ 1010 \\
 +1 \\
 \hline
 \text{Số bù 2 của 5} \quad 1111 \ 1111 \ 1111 \ 1011
 \end{array}$$

Bây giờ hãy cộng 5 với số bù 2 của nó

$$\begin{array}{r}
 \text{Số bù 2 của 5} \quad 1111 \ 1111 \ 1111 \ 1011 \\
 5 \quad + \ 0000 \ 0000 \ 0000 \ 0101 \\
 \hline
 \text{Tổng nhận được} \quad 10000 \ 0000 \ 0000 \ 0000
 \end{array}$$

Chúng ta nhận được một số 17 bit vì một word bộ nhớ chỉ có thể lưu được 16 bit, phần nhớ 1 nằm ở bit già nhất msb sẽ bị mất do đó kết quả 16 bit bằng 0. Do tổng của 5 và số bù 2 của nó bằng 0, số bù 2 của 5 rõ ràng biểu diễn giá trị -5.

Để thấy rằng tại sao số bù 2 của một số nguyên N lại bằng -N; Đơn giản là khi cộng N với số bù 1 của nó ta nhận được số gồm 16 bit 1 và khi cộng kết quả này với 1 ta nhận được 16 bit 0 và nhớ 1 sang bit thứ 17 nhưng bit thứ 17 này lại bị mất đi do đó kết quả cuối cùng bằng 0. Ví dụ sau đây sẽ cho ta thấy điều gì xảy ra khi lấy bù 2 của một số liên tiếp 2 lần.

Ví dụ 2.8 Tìm số bù 2 của số bù 2 của 5

Trả lời

Chúng ta cũng đoán được rằng sau khi lấy bù 2 của 5 hai lần liên tiếp, kết quả nhận được sẽ là 5. Để kiểm chứng điều này chúng ta hãy làm phép tính:

Theo kết quả phần trên, ta có :

$$\begin{array}{r}
 \text{Số bù 2 của 5} \quad 1111 \ 1111 \ 1111 \ 1011 \\
 \text{Số bù 1 của } 1111 \ 11111 \ 1111 \ 1011 \quad 0000 \ 0000 \ 0000 \ 0100 \\
 +1 \\
 \hline
 \text{Số bù 2 của } 1111 \ 11111 \ 1111 \ 1011 \quad 0000 \ 0000 \ 0000 \ 0101 = 5
 \end{array}$$

Ví dụ 2.9 Hãy biểu diễn số -97 dưới dạng số hex với
a.8 bit ?
b.16 bit ?

Trả lời

Sử dụng phương pháp biến đổi số thập phân thành số hex bằng cách lặp lại phép chia cho 16 ta nhận được :

$$6 \equiv 0 * 16 + 6$$

Do đó $97 = 61h$, và để biểu diễn -97 chúng ta phải biểu diễn $61h$ dưới dạng nhị phân rồi tìm số bù 2 của nó.

a. Dưới dạng 8 bit ta có:

| | |
|-----------------|-----------------|
| 61h | 0110 0001 |
| Số bù 1 của 61h | 1001 1110 |
| | +1 |
| Số bù 2 của 61h | 1001 1111 = 9Fh |

b. Dưới dạng 16 bit ta có:

| | |
|-----------------|-----------------------------|
| 61h | 0000 0000 0110 0001 |
| Số bù 1 của 61h | 1111 1111 1001 1110 |
| | +1 |
| Số bù 2 của 61h | 1111 1111 1001 1111 = FF9Fh |

Phương pháp trừ bằng cách cộng với số bù 2

Một ưu điểm của việc biểu diễn số âm bằng các số bù 2 trong máy tính là phép trừ có thể thực hiện bằng phương pháp lấy bù và phép cộng, và các vi mạch thực hiện phép cộng và bù các bit được thiết kế tương đối dễ dàng.

Vi du 2.10

Giả sử AX chứa 5ABCh và BX chứa 21FCh. Hãy tìm hiệu của AX và BX bằng phương pháp lấy bù và thực hiện phép cộng.

Trả lời:

| | | |
|---------------------------|---|------------------------------|
| AX chứa 5ABC _h | = | 0101 1010 1011 1100 |
| BX chứa 21FCh | = | 0010 0001 1111 1100 |
| Số bù 1 của 21FCh | = | 1101 1110 0000 0011 |
| Số bù 2 của 21FCh | = | 1101 1110 0000 0100 |
| Hiệu số | = | 10011 1000 1100 0000 = 38C0h |

Một đơn vị nhớ từ bit msb bị mất và kết quả nhận được là 38C0h có thể kiểm chứng lại bằng phép trừ số hex.

2.4.3 Biểu diễn số thập phân

Trong phần trước chúng ta đã thấy các số nguyên thập phân có dấu cũng như không dấu được biểu diễn ra sao trong máy tính. Vấn đề ngược lại là biểu diễn nội dung của một byte hay một word dưới dạng số thập phân có dấu và không có dấu như thế nào.

- *Biểu diễn số thập phân không dấu:*

Chỉ cần thực hiện phép biến đổi số nhị phân thành số thập phân, để đơn giản trước tiên nên đổi số nhị phân thành số hex rồi sau đó đổi số hex thành số thập phân.

- *Biểu diễn số thập phân có dấu:*

Nếu bit msb bằng 0, số là số dương và số thập phân có dấu trong trường hợp này giống như trường hợp không dấu. Nếu bit msb bằng 1, số là số âm vì thế chúng ta gọi nó là -N và để có N ta phải lấy bù 2 của số cần đổi rồi đổi nó sang dạng thập phân như đã làm trước đó.

Ví dụ 2.11 Giả sử thanh ghi AX chứa FE0Ch hãy tìm dạng biểu diễn thập phân có dấu và không dấu.

Trả lời Thực hiện phép biến đổi FE0Ch ra dạng thập phân chúng ta nhận được 65036, đó chính là dạng biểu diễn không dấu. Để tìm dạng biểu diễn có dấu chúng ta nhận thấy:

$$\text{FE0Ch} = 1111111000001100.$$

Vì bit dấu bằng 1 số này là số âm và chúng ta gọi nó là -N. Để tìm N chúng ta tìm số bù 2:

| | |
|-------------------|-------------------------------|
| FE0Ch | 1111 1110 0000 1100 |
| Số bù 1 của FE0Ch | 0000 0001 1111 0011 |
| Số bù 2 của FE0Ch | 0000 0001 1111 0100=01F4h=500 |

Chúng ta nhận được kết quả 01F4h hay bằng 500 do đó AX chứa -500

Các bảng 2.4A và 2.4B cho ta các biểu diễn thập phân có dấu và không dấu của các số hex 16 bit và 8 bit. Chú ý :

1. Bit msb của số nguyên dương có dấu là 0, chữ số hex đầu tiên của số nguyên dương có dấu là từ 0 đến 7; ngược lại các số nguyên có dấu bắt đầu với các chữ số từ 8 đến F có bit msb bằng 1 nên là các số âm.
2. Số nguyên dương có dấu 16 bit lớn nhất là 7FFFh = 32767, số nhỏ nhất là 8000h = -32768. Số nguyên dương có dấu 8 bit lớn nhất là 7Fh = 127, số nhỏ nhất là 80h = -128.
3. Giữa dạng biểu diễn số thập phân có dấu và không dấu của nội dung một word 16 bit có mối quan hệ sau đây:

Trong khoảng 0000h- 7FFFh, số thập phân có dấu bằng số thập phân không dấu

Trong khoảng 8000h- FFFFh, số thập phân có dấu bằng số thập phân không dấu trừ đi 65536.

Tương tự cũng có các quan hệ giữa dạng biểu diễn thập phân không dấu và có dấu của nội dung một byte nhớ 8 bit:

Trong khoảng 00h- 7Fh, số thập phân có dấu bằng số thập phân không dấu

Trong khoảng 80h- FFh, số thập phân có dấu bằng số thập phân không dấu trừ đi 256

Ví dụ 2.12 Sử dụng nhận xét 3 hãy làm lại ví dụ 2.11

Trả lời

Chúng ta nhận xét rằng dạng biểu diễn thập phân không dấu của FE0Ch bằng 65036. Vì chữ số đầu tiên số hex FE0Ch là F nên nó là số âm khi xét về phương diện số có dấu. Để biểu diễn nó ta chỉ việc trừ 65536 dạng biểu diễn thập phân không dấu và nhận được: $65036 - 65536 = -500$.

Bảng 2.4A Biểu diễn số thập phân có dấu và không dấu của word

| SỐ HEX | SỐ THẬP PHÂN KHÔNG DẤU | SỐ THẬP PHÂN CÓ DẤU |
|--------|------------------------|---------------------|
| 0000 | 0 | 0 |
| 0001 | 1 | 1 |
| 0002 | 2 | 2 |
| . | . | . |
| 0009 | 9 | 9 |
| 000A | 10 | 10 |
| . | . | . |
| 7FFE | 32766 | 32766 |
| 7FFF | 32767 | 32767 |
| 8000 | 32768 | -32768 |
| . | . | . |
| FFFE | 65534 | -2 |
| FFFF | 65535 | -1 |

Bảng 2.4B Biểu diễn thập phân có dấu và không dấu của byte

| SỐ HEX | SỐ THẬP PHÂN KHÔNG DẤU | SỐ THẬP PHÂN CÓ DẤU |
|--------|------------------------|---------------------|
| 00 | 0 | 0 |
| 01 | 1 | 1 |
| 02 | 2 | 2 |
| . | . | . |
| 09 | 9 | 9 |
| 0A | 10 | 10 |
| . | . | . |
| 7E | 126 | 126 |
| 7F | 127 | 127 |
| 80 | 128 | -128 |
| . | . | . |
| FE | 254 | -2 |
| FF | 255 | -1 |

2.5 Biểu diễn các ký tự

Bảng 2.5 Bảng mã ASCII

| DEC | HEX | CHAR |
|-----|-----|------|-----|-----|------|-----|-----|------|-----|-----|------|
| 0 | 00 | <CC> | 32 | 20 | SP | 64 | 40 | @ | 96 | 60 | |
| 1 | 01 | <CC> | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 02 | <CC> | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 03 | <CC> | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 04 | <CC> | 36 | 24 | \$ | 68 | 44 | D | 100 | 64 | d |
| 5 | 05 | <CC> | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 06 | <CC> | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 07 | <CC> | 39 | 27 | , | 71 | 47 | G | 103 | 67 | g |
| 8 | 08 | <CC> | 40 | 28 | (| 72 | 48 | H | 104 | 68 | h |
| 9 | 09 | <CC> | 41 | 29 |) | 73 | 49 | I | 105 | 69 | i |
| 10 | 0A | <CC> | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | 0B | <CC> | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | 0C | <CC> | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | 0D | <CC> | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | 0E | <CC> | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | 0F | <CC> | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | <CC> | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | <CC> | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | <CC> | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | <CC> | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | <CC> | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | <CC> | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | <CC> | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | <CC> | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | <CC> | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | <CC> | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | <CC> | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | <CC> | 59 | 3B | ; | 91 | 5B | [| 123 | 7B | { |
| 28 | 1C | <CC> | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | |
| 29 | 1D | <CC> | 61 | 3D | = | 93 | 5D |] | 125 | 7D | } |
| 30 | 1E | <CC> | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | <CC> | 63 | 3F | ? | 95 | 5F | | 127 | 7F | <CC> |

Chú thích:

<CC> : ký tự điều khiển(Control Character)

SP : ký tự khoảng trắng (blank space)

Các ký tự đặc biệt:

| DEC | HEX | CHAR | NỘI DUNG |
|-----|-----|------|-------------|
| 7 | 07 | BEL | Chuông |
| 8 | 08 | BS | Xoá ngược |
| 9 | 09 | HT | Tab ngang |
| 10 | 0A | LF | Xuống dòng |
| 12 | 0C | FF | Sang trang |
| 13 | 0D | CR | Về đầu dòng |

Mã ASCII

Không phải mọi số liệu mà máy tính xử lý đều là các con số, các thiết bị ngoại vi như màn hình hay máy in đều có xu hướng làm việc với các ký tự. Ngoài ra các chương trình như chương trình xử lý văn bản chuyên làm việc với dữ liệu kiểu ký tự. Cũng như tất cả mọi loại dữ liệu khác, các ký tự cần phải được mã hoá thành dạng nhị phân để máy tính có thể xử lý chúng. Một kiểu mã hoá thông dụng nhất cho các ký tự đó là mã ASCII (American Standard Code for Information Interchange: Mã chuẩn của Mĩ dùng để trao đổi thông tin). Trước kia mã ASCII được sử dụng trong thông tin với các thiết bị teletype, ngày nay mã ASCII được sử dụng trên tất cả các máy tính cá nhân.

Hệ thống mã ASCII sử dụng 7 bit để mã hoá mỗi ký tự, do đó có tổng cộng $2^7 = 128$ mã ASCII. Bảng 2.5 trình bày các mã ASCII và các ký tự tương ứng với chúng.

Chú ý rằng chỉ có 95 ký tự ứng với 95 mã ASCII từ 32 đến 126 là có khả năng in được, các mã ASCII từ 0 đến 31 và mã 127 được dùng với các mục đích điều khiển quá trình truyền thông, do đó không có khả năng in được. Hầu hết các máy vi tính chỉ sử dụng các ký tự in được và một số ký tự điều khiển như CR, LF, BS và BELL.

Vì mỗi ký tự ASCII được mã hoá bằng 7 bit nên mã của một ký tự độc lập chứa vừa vặn trong một byte với bit msb bằng 0. Các ký tự in được có thể hiển thị trên màn hình hoặc in ra máy in trong khi các ký tự điều khiển lại được sử dụng để điều khiển các thiết bị này. Chẳng hạn để hiển thị ký tự A trên màn hình chương trình sẽ gửi mã ASCII 41h đến màn hình, còn để lùi con trỏ trở lại đầu dòng, chương trình gửi mã ASCII 0Dh (mã ASCII của ký tự điều khiển CR) đến màn hình.

Ngoài ra máy tính cũng có thể hiển thị một số ký tự đặc biệt ứng với một số mã ASCII không in được. Như bạn sẽ thấy sau này bộ điều khiển màn hình của IBM PC có thể hiển thị một bộ ký tự mở rộng. Phụ lục A trình bày bộ 256 ký tự hiển thị của IBM PC.

Ví dụ 2.13

Hãy cho biết chuỗi ký tự 'RG 2z' được lưu trong bộ nhớ từ địa chỉ 0 như thế nào?

Trả lời :

Từ bảng 2.5 chúng ta có

| KÝ TỰ | MÃ ASCII (HEX) | MÃ ASCII (NHỊ PHÂN) |
|-------|----------------|---------------------|
| R | 52 | 0101 0010 |
| G | 47 | 0100 0111 |
| space | 20 | 0010 0000 |
| 2 | 32 | 0011 0010 |
| z | 7A | 0111 1010 |

Và do đó bộ nhớ có dạng sau:

| ĐỊA CHỈ | NỘI DUNG |
|---------|----------|
| 0 | 01010010 |
| 1 | 01000111 |
| 2 | 00100000 |
| 3 | 01111010 |

Bàn phím

Cũng rất có lý khi nghĩ rằng bàn phím xác định phím bằng cách tạo ra mã ASCII khi phím được ấn. Điều đó đúng cho loại bàn phím được xem là bàn phím ASCII của một số máy vi tính thế hệ cũ. Tuy nhiên các bàn phím hiện đại ngày nay ngoài các phím ký tự ASCII còn có rất nhiều phím điều khiển và phím chức năng. Do đó một phương án mã hoá khác được sử dụng. Với các máy IBM-PC, mỗi phím được gán cho một con số duy nhất gọi là mã quét (scan code). Khi phím được ấn, bàn phím gửi mã scan của phím đến máy tính. Những vấn đề về mã quét sẽ được trình bày trong chương 12.

TỔNG KẾT

Trong chương này chúng ta đã nghiên cứu những vấn đề sau:

- Các số được biểu diễn theo các phương pháp khác nhau tuỳ theo các ký hiệu cơ sở được sử dụng. Hệ đếm nhị phân sử dụng 2 chữ số, 0 và 1. Hệ đếm thập phân sử dụng các chữ số từ 0 đến 9, hệ đếm 16 sử dụng các chữ số từ 0 đến 9 và các chữ cái từ A đến F.
- Các số nhị phân và số hex có thể đổi sang số thập phân bằng cách thực hiện phép nhân lồng nhau.
- Một số hex có thể đổi sang số thập phân bằng quá trình chia lặp cho 16, tương tự một số nhị phân cũng có thể đổi sang số thập phân bằng quá trình chia lặp cho 2.
- Số hex có thể đổi sang số nhị phân bằng cách đổi từng chữ số của nó thành số nhị phân. Số nhị phân có thể đổi sang số hex bằng cách nhóm từng nhóm 4 chữ số của nó từ phải sang trái, sau đó đổi từng nhóm sang số hex.
- Quá trình thực hiện phép cộng và phép trừ các số nhị phân và số hex cũng tương tự như đổi với số thập phân và có thể dùng bảng cộng tương ứng.
- Các số âm được lưu dưới dạng số bù 2. Để lấy bù 2 của một số chỉ việc lấy phần bù của từng bit rồi cộng 1 vào kết quả nhận được.
- Nếu A và B chứa các số nguyên thì bộ xử lý tính hiệu A-B bằng cách cộng A với số bù 2 của B.
- Phạm vi của số nguyên không dấu chứa trong 1 byte là từ 0 đến 255 và trong một word là từ 0 đến 65535.
- Đối với các số có dấu, bit msb là bit dấu, 0 có nghĩa là dương và 1 có nghĩa là âm. Phạm vi của số nguyên có dấu chứa trong một byte là từ -128 đến 127 và trong một word là từ -32768 đến 32767.
- Có thể biểu diễn số thập phân không dấu chứa trong một word bằng cách đổi giá trị nhị phân ra thập phân. Nếu bit dấu bằng 0 đó đồng thời cũng là dạng biểu diễn của số thập phân có dấu. Nếu bit dấu bằng 1 số thập phân có dấu có thể nhận được bằng cách trừ đi 65536 từ dạng biểu diễn số thập phân không dấu.
- Bộ mã chuẩn cho các ký tự là mã ASCII.
- Mỗi ký tự dùng 7 bit để mã hoá do đó nó có thể chứa trong một byte.

- Bộ điều khiển màn hình của IBM PC có thể tạo ra bất kỳ ký tự nào có mã là một trong 256 số có thể chứa trong một byte.

Các thuật ngữ tiếng Anh

ASCII (American Standard Code for Information Interchange) codes

Bộ mã chuẩn cho các ký tự được sử dụng trên tất cả các máy tính cá nhân

Binary number system

Hệ đếm cơ số 2 trong đó chỉ dùng 2 chữ số 0 và 1

hexadecimal number system

Hệ đếm cơ số 16 trong đó dùng các chữ số từ 0 đến 9 và các chữ cái từ A đến F

Least significant bit, lsb

Bit bên phải nhất của byte hoặc word -bit 0

Most significant bit, msb

Bit trái nhất của byte hoặc word: bit 7 của byte, bit 15 của word

One's complement

Số bù 1 của một số nhị phân nhận được bằng cách lấy bù từng bit của số đó.

Scan code

Một số để xác định một phím trên bàn phím.

signed integer

Số nguyên có thể nhận giá trị dương hoặc âm.

Two's complement

Số bù 2, nhận được bằng cách cộng thêm 1 vào số bù 1

Unsigned integer

Số nguyên biểu thị trị số, có nghĩa là luôn nhận giá trị dương

Bài tập

1. Trong nhiều trường hợp nếu bạn nhớ được sự chuyển đổi giữa các số nhị phân, thập phân và số hex nhỏ sẽ tiết kiệm rất nhiều thời gian. Không xem bảng 2.2 hãy điền vào những ô trống trong bảng dưới đây:

| Nhị phân | Thập phân | Hex |
|----------|-----------|------|
| | 9 | |
| 1010 | | |
| | | D |
| | 12 | |
| 1110 | | |
| | | B |

2. Đổi những số nhị phân và số hex sau đây ra số thập phân :

- a. 1110
 - b. 100101011101
 - c. 46Ah
 - d. FAE2Ch

3. Đổi những số thập phân sau đây:

4. Đổi các số sau đây:

- | | | |
|----|-----------------|----------------|
| a. | 1001011 | ra số hex |
| b. | 100101010101110 | ra số hex |
| c. | A2Ch | ra số nhị phân |
| d. | B34Dh | ra số nhị phân |

- #### 5. Thực hiện các phép cộng sau:

- a. $100101b + 10111b$
 - b. $10011111001b + 100001111101b$
 - c. $B23CDh + 17912h$
 - d. $FFFFEhb + EBCADhb$

- #### 6. Thực hiện các phép tính sau

- a. 11011b-10110b
 - b. 10000101b-111011b
 - c. 5FC12h-3ABD1h

d. F001Eh-1FF3F

7. Biểu diễn các số nguyên thập phân dưới đây dưới dạng số hex 16 bit

- a. 234
- b. -16
- c. 31634
- d. -32216

8. Thực hiện phép trừ các số nhị phân và số hex dưới đây bằng cách cộng với số bù 2:

- a. 10110100-10010111
- b. 10001011-11110111
- c. FE0Fh-12ABh
- d. 1ABCh-B3EAh

9. Biểu diễn các số 16 bit và 8 bit dưới đây ra dạng số thập phân có dấu và không có dấu:

- a. 7FFEh
- b. 8543h
- c. FEh
- d. 7Fh

10. Hãy biểu diễn -120 thành dạng 16 bit và 8 bit

11. Cho biết mỗi số thập phân dưới đây có thể lưu được hay không trong (a) như một số 16 bit; (b) như một số 8 bit

- a. 32767
- b. -40000
- c. 65536
- d. 257
- e. -128

12. Hãy cho biết các số có dấu dưới đây là số dương hay âm:

- a. 1010010010001010b
- b. .78E3h
- c. .CB33h
- d. .807Fh
- e. .9AC4h

13. Giả sử chuỗi "S12.75" đang lưu trong bộ nhớ bắt đầu tại địa chỉ 0, cho biết nội dung của các byte từ 0 đến 5 dưới dạng số hex.

14. Hãy dịch bản thông điệp đã được mã hoá dưới dạng mã ASCII sau đây :

41 74 74 61 63 6B 20 61 77 6E

15. Giả sử một byte chứa mã ASCII của một ký tự chữ in, hỏi phải cộng thêm vào một số hex bao nhiêu để đổi nó thành dạng chữ thường.
 16. Giả sử một byte chứa mã ASCII của một chữ số thập phân hỏi phải trừ đi một số hex bao nhiêu để byte chứa các chữ số đó.
 17. Có một phương pháp cộng hoặc trừ các chữ số hex mà không phải dùng bảng cộng, chẳng hạn để tính $Ah + Eh$, trước hết bạn viết lại các chữ số hex như sau:

0 1 2 3 4 5 6 7 8 9 A B C D E F

Bây giờ bắt đầu tại Eh chuyển sang phải Ah= 10 vị trí, khi bạn ra khỏi dòng, hãy tiếp tục từ đầu dòng bên trái, mỗi khi vượt qua một chữ số hãy cộng thêm 1 vào nó.

10 11 12 13 14 15 16 17 18 9 A B C D E F
Kết thúc^ Bắt đầu^

Cuối cùng ban nhân được Eh + Ah = 18h

Tương tự ta cũng có thể thực hiện phép trừ, ví dụ để tính $15h - Ch$ hãy bắt đầu từ $15h$ chuyển sang trái $Ch = 12$ vị trí, khi bạn ra khỏi đầu dòng bên trái hãy tiếp tục từ bên phải:

10 11 12 13 14 15 16 17 18 9 A B C D E F
^Bắt đầu ^Kết thúc

Và bạn nhận được 15 - Ch=9h

Hãy làm lại bài tập 5(c) và 6(c) bằng phương pháp này.

Chương 3

TỔ CHỨC CỦA MÁY TÍNH CÁ NHÂN IBM PC

Tổng quan

Chương 1 đã mô tả tổ chức của một hệ thống máy vi tính điển hình, trong chương này chúng ta sẽ nghiên cứu kỹ hơn về các máy tính cá nhân IBM. Đó là hệ máy dựa trên cơ sở các bộ vi xử lý của họ vi xử lý INTEL 8086.

Sau khi khảo sát sơ qua về họ 8086 trong phần 3.1, phần 3.2 sẽ tập trung vào kiến trúc của 8088, chúng tôi sẽ giới thiệu các thanh ghi và nêu ra một số chức năng đặc biệt của chúng. Trong phần 3.2 chúng ta sẽ thảo luận về một vấn đề quan trọng đó là khái niệm bộ nhớ phân đoạn.

Trong phần 3.3 chúng ta sẽ xem xét một cách tổng hợp về cấu trúc của IBM PC bao gồm: tổ chức bộ nhớ, các cổng vào/ra và các chương trình của DOS và BIOS.

3.1 Họ vi xử lý INTEL 8086

Họ máy vi tính cá nhân IBM bao gồm các loại máy IBM PC, PC XT, PC AT, PS/1 và PS/2. Chúng đều dựa trên cơ sở các bộ vi xử lý của họ 8086 bao gồm 8086, 8088, 80186, 80188, 80286, 80386, 80386SX, 80486, 80486SX. Bộ vi xử lý 8088 được sử dụng trong các máy PC và PC XT, 80286 được sử dụng trong các máy PC AT và PS/1; 80186 được sử dụng trong các máy tính cỡ nhỏ trong khi máy PS/2 thì sử dụng cả 8086, 80286, 80386 cũng như 80486, 80586.

Các bộ vi xử lý 8086 và 8088

Hãng INTEL cho ra đời các bộ vi xử lý 8086 vào năm 1978, đó là bộ vi xử lý 16 bit đầu tiên của INTEL (bộ vi xử lý 16 bit là bộ vi xử lý có thể cùng lúc thao

tác với 16 bit dữ liệu). Bộ vi xử lý 8088 được cho ra đời vào năm 1979. Xét về bên trong, 8088 về cơ bản giống như 8086 nhưng xét về bên ngoài thì 8086 có bus dữ liệu 16 bit trong khi 8088 chỉ có bus dữ liệu 8 bit. 8086 còn có tốc độ đồng hồ cao hơn 8088 và bởi thế nó hoạt động nhanh hơn. Nhưng vì nguyên nhân kinh tế hãng IBM đã chọn 8088 cho các máy tính PC đầu tiên.

8086 và 8088 có cùng tập lệnh, tập lệnh này cũng được dùng cho các bộ vi xử lý khác trong họ INTEL 80XXX.

Các bộ vi xử lý 80186 và 80188

80186 và 80188 là các phiên bản cải tiến của 8086 và 8088. Ưu điểm của chúng là ở chỗ chúng có khả năng thực hiện tất cả các chức năng của 8086 và 8088 thêm vào đó chúng còn có các chip phụ trợ. Chúng có thể thực hiện một số lệnh mới gọi là tập lệnh mở rộng. Tuy nhiên những ưu điểm này cũng không mấy quan trọng và chúng nhanh chóng bị lu mờ bởi sự phát triển của bộ vi xử lý 80286.

Bộ vi xử lý 80286

Bộ vi xử lý 80286 được giới thiệu vào năm 1982, đó cũng là bộ vi xử lý 16 bit. Tuy nhiên nó có thể thao tác nhanh hơn 8086 (tần số đồng hồ của nó là 12.5MHz trong khi của 8086 là 10MHz), ngoài ra nó còn cung cấp những ưu điểm quan trọng sau đây so với người anh em đi trước.

1. Khả năng hoạt động ở 2 chế độ. 80286 có thể thao tác cả ở chế độ **địa chỉ thực** (*real address mode*) cũng như chế độ **địa chỉ ảo được bảo vệ** (*protected virtual address mode*). Trong chế độ địa chỉ thực 80286 được xem như 8086, mọi chương trình viết cho 8086 có thể chạy trong chế độ này mà không cần một thay đổi nào. Trong chế độ địa chỉ ảo được bảo vệ (còn gọi là chế độ bảo vệ- *protected mode*) 80286 cung cấp khả năng làm việc đa nhiệm (*multitasking*), đó là khả năng cùng lúc thực hiện vài chương trình (hay công việc) và khả năng **bảo vệ bộ nhớ** (*memory protection*), đó là khả năng bảo vệ bộ nhớ sử dụng bởi chương trình này khỏi các tác động của chương trình khác.
2. Khả năng định địa chỉ nhiều hơn. Trong chế độ bảo vệ 80286 có thể định địa chỉ được đến 16 megabyte bộ nhớ vật lý (so với 1megabyte của 8086 và 8088).
3. Khả năng sử dụng bộ nhớ ảo trong chế độ bảo vệ. 80286 có thể sử dụng bộ nhớ ngoài (như đĩa chẵng hạn) như là bộ nhớ vật lý và bởi vậy có thể thực

hiện các chương trình quá lớn để chứa trong bộ nhớ vật lý như những chương trình có kích thước đến 1 gigabyte(2^{30} byte).

Các bộ vi xử lý 80386 và 80386SX

INTEL cho ra đời bộ vi xử lý 32 bit đầu tiên- bộ vi xử lý 80386 (hay còn gọi là 386) vào năm 1985. Chúng hoạt động nhanh hơn nhiều bộ vi xử lý 80286 vì có bus dữ liệu 32 bit, tần số đồng hồ rất cao (lên đến 33MHz) đồng thời khả năng thực hiện các lệnh trong một số ít chu kỳ đồng hồ hơn 80286.

Giống như 80286, 80386 có thể hoạt động trong chế độ địa chỉ thực cũng như trong chế độ bảo vệ. Trong chế độ địa chỉ thực chúng được xem như 8086. Trong chế độ bảo vệ nó có thể giả lập 80286. Đồng thời nó còn có chế độ ảo 8086 được thiết kế để chạy đồng thời nhiều ứng dụng của 8086 dưới chế độ bộ nhớ bảo vệ. 80386 trong chế độ bảo vệ có thể định địa chỉ tới 4 Gigabyte của bộ nhớ vật lý và 64 Terabyte (2^{46} byte) bộ nhớ ảo. Bộ vi xử lý 386SX có cấu trúc bên trong về cơ bản giống như 386 nhưng nó chỉ có bus dữ liệu 16 bit.

Các bộ vi xử lý 80486 và 80486SX

Ra đời năm 1989, 80486 (hay 486) là một bộ vi xử lý 32 bit khác của INTEL. Cùng với các chức năng của 80386 nó có các chip phụ trợ khác bao gồm bộ xử lý số 80387 dùng để thực hiện các phép tính với dấu phẩy động và một bộ nhớ truy nhập nhanh (cache memory) 8 KB được sử dụng như vùng nhớ nhanh làm vùng đệm cho dữ liệu vào từ các đơn vị nhớ chậm. Với bộ xử lý số, bộ nhớ truy nhập nhanh và các thiết kế tiên tiến khác, 80486 chạy nhanh gấp 3 lần 80386 có cùng tốc độ đồng hồ. 80486SX cũng tương tự như 80486 nhưng không có bộ xử lý số dấu phẩy động.

3.2 Tổ chức của các bộ vi xử lý 8086/8088

Trong phần còn lại của chương này chúng ta sẽ tập trung vào tổ chức của 8086 và 8088. Các bộ vi xử lý này có cấu trúc đơn giản nhất, và hầu hết các lệnh mà chúng ta sẽ học là các lệnh của 8086/8088. Việc nghiên cứu này cũng cho ta cái nhìn thấu suốt đến tổ chức của các bộ vi xử lý cao cấp hơn mà sẽ được đề cập đến trong chương 20.

Vì 8086 và 8088 có cấu trúc bên trong về cơ bản là giống nhau, trong phần dưới đây tên gọi 8086 được hiểu là cả 8086 và 8088.

3.2.1 Các thanh ghi

Như đã nêu trong chương 1, thông tin được lưu giữ bên trong bộ vi xử lý trong các thanh ghi. Các thanh ghi được phân loại theo chức năng của chúng, thanh ghi dữ liệu chứa dữ liệu cho một thao tác, thanh ghi địa chỉ chứa địa chỉ của lệnh hay của dữ liệu và thanh ghi trạng thái thì lưu trạng thái hiện thời của bộ vi xử lý.

Bộ vi xử lý 8086 có 4 thanh ghi dữ liệu công dụng chung; các thanh ghi địa chỉ được chia ra làm các thanh ghi đoạn, thanh ghi con trỏ, thanh ghi chỉ số; thanh ghi trạng thái được gọi là thanh ghi cờ. Như vậy tổng cộng có 14 thanh ghi 16 bit (xem hình 3.1), dưới đây chúng tôi sẽ mô tả ngắn gọn chúng.

Chú ý bạn không cần phải ghi nhớ các chức năng đặc biệt của các thanh ghi ngay bây giờ, chúng ta sẽ dần dần quen thuộc với chúng trong quá trình sử dụng.

3.2.2 Các thanh ghi dữ liệu: AX, BX, CX, DX

Bốn thanh ghi này được người lập trình sử dụng cho các thao tác với dữ liệu. Mặc dù bộ vi xử lý có thể thao tác với dữ liệu trong bộ nhớ nhưng một lệnh như vậy sẽ được thực hiện nhanh hơn (cần ít chu kỳ đồng hồ hơn) nếu dữ liệu được lưu trong các thanh ghi. Đó cũng là nguyên nhân tại sao các bộ vi xử lý hiện đại lại có xu hướng có nhiều thanh ghi.

Các byte cao và thấp trong thanh ghi có thể truy nhập riêng biệt. Byte cao của thanh ghi AX gọi là AH, byte thấp gọi là AL. Tương tự như vậy các byte cao và byte thấp của các thanh ghi BX, CX, DX lần lượt là BH và BL, CH và CL, DH và DL. Nhờ điều này mà chúng ta có thêm nhiều thanh ghi khi làm việc với số liệu có kích thước byte.

Bốn thanh ghi này với ý nghĩa là các thanh ghi công dụng chung còn thực hiện các chức năng đặc biệt sau:

Thanh ghi AX (thanh ghi chứa - Accumulator register)

AX là thanh ghi được sử dụng nhiều nhất trong các lệnh số học, lô gic, và chuyển dữ liệu bởi vì việc sử dụng chúng tạo ra mã máy ngắn nhất.

Trong các thao tác nhân và chia một trong các số hạng tham gia phải được chứa trong AX hay AL. Các thao tác vào và ra cũng sử dụng thanh ghi AX hoặc AL.

THANH GHI SỐ LIỆU

| | | |
|----|----|----|
| AX | AH | AL |
| BX | BH | BL |
| CX | CH | CL |
| DX | DH | DL |

THANH GHI ĐOẠN

| | |
|----|--|
| CS | |
| DS | |
| ES | |
| SS | |

THANH GHI CON TRỎ VÀ CHỈ SỐ

| | |
|----|--|
| SI | |
| DI | |
| SP | |
| BP | |
| IP | |

THANH GHI CỜ

| |
|--|
| |
|--|

Hình 3.1 Các thanh ghi của 8086

Thanh ghi BX (thanh ghi cơ sở - Base register)

BX cũng đồng thời đóng vai trò là thanh ghi địa chỉ, một ví dụ là nó chứa địa chỉ bảng dữ liệu cho lệnh XLAT . (TRANSLATE)

Thanh ghi CX (thanh ghi đếm - Count register)

Việc xây dựng các chương trình lặp được thực hiện dễ dàng bằng cách sử dụng thanh ghi CX, trong đó CX đóng vai trò là bộ đếm số vòng lặp. Một ví dụ khác của việc sử dụng thanh ghi CX đó là trong lệnh REP (REPEAT), lệnh này điều khiển một lớp các lệnh đặc biệt chuyên về các thao tác chuỗi. CL cũng được sử dụng như là biến đếm trong các lệnh dịch hay quay các bit.

Thanh ghi DX (thanh ghi dữ liệu - Data register)

DX được sử dụng trong các thao tác nhân và chia, nó cũng được sử dụng trong các thao tác vào/ra.

3.2.3 Các thanh ghi đoạn CS, DS, ES, SS

Các thanh ghi địa chỉ lưu giữ địa chỉ của các lệnh và dữ liệu trong bộ nhớ. Bộ vi xử lý sử dụng các giá trị này để truy nhập bộ nhớ. Chúng ta hãy bắt đầu với tổ chức của bộ nhớ.

Chương 1 đã chỉ ra rằng bộ nhớ là tập hợp các byte. Mỗi byte bộ nhớ có một địa chỉ xác định, bắt đầu bằng địa chỉ 0. Bộ vi xử lý 8086 gán cho mỗi ô nhớ của nó một địa chỉ vật lý (physical address) 20 bit, bởi vậy nó có thể định địa chỉ được tới $2^{20} = 1048576$ byte (1 megabyte) bộ nhớ. 5 byte đầu tiên của bộ nhớ có địa chỉ như sau:

```

0000 0000 0000 0000 0000
0000 0000 0000 0000 0001
0000 0000 0000 0000 0010
0000 0000 0000 0000 0011
0000 0000 0000 0000 0100

```

Vì viết các địa chỉ dưới dạng nhị phân rất mệt nên chúng ta thường viết nó dưới dạng 5 chữ số hex như là:

```

00000
00001
00002

```

00009
0000A
0000B

Và cứ như vậy. Địa chỉ lớn nhất là FFFFh

Để giải thích chức năng của các thanh ghi đoạn, trước hết chúng tôi cần giới thiệu khái niệm các đoạn bộ nhớ. Đó chính là kết quả trực tiếp của việc sử dụng 20 bit địa chỉ trong bộ vi xử lý 16 bit. Bởi vì các địa chỉ quá lớn để đặt nó trong các thanh ghi hay từ nhữ 16 bit, 8086 giải quyết vấn đề này bằng cách chia bộ nhớ thành các đoạn (segments).

Đoạn bộ nhớ (memory segment)

Một đoạn bộ nhớ (memory segment) là một khối gồm 2^{16} byte (hay 64 KB) ô nhớ liên tiếp. Mỗi đoạn được xác định bằng một địa chỉ đoạn (segment number), bắt đầu bằng địa chỉ 0. Địa chỉ đoạn là một số 16 bit nên địa chỉ đoạn lớn nhất là FFFFh. Bên trong mỗi đoạn, các ô nhớ được xác định bằng một địa chỉ tương đối (offset), đó là số byte tính từ đầu đoạn. Với một đoạn 64 KB, offset là một số 16 bit. Byte đầu tiên trong một đoạn có offset 0, và byte cuối cùng trong đoạn có offset FFFFh.

Địa chỉ Segment:offset

Mỗi ô nhớ có thể được xác định bằng một địa chỉ đoạn và một offset, viết dưới dạng: **đoạn: vị trí tương đối** trong đoạn (segment:offset), nó được xem là địa chỉ lô gic. Ví dụ A4FB:4872h có offset 4872h trong đoạn A4FBh. Để nhận được một địa chỉ vật lý 20 bit bộ vi xử lý 8086 trước tiên dịch địa chỉ đoạn 4 bit về bên trái (tương đương với việc nhân với 10h) và sau đó cộng với offset. Như vậy địa chỉ vật lý của A4FB:4872h

$$\begin{array}{r} \text{A4FB0h} \\ \underline{4872h} \\ \text{A9822h} \end{array}$$

Sự bố trí các đoạn

Một việc không thể bỏ qua là phải xem sự bố trí các đoạn trong bộ nhớ. Đoạn 0 bắt đầu từ địa chỉ $0000:0000 = 00000h$ và kết thúc ở $0000:FFFF = OFFFFh$. Đoạn 1 bắt đầu từ địa chỉ $0001:0000 = 00010h$ và kết thúc ở $0001:FFFF = 1000Fh$. Như chúng ta đã thấy, có rất nhiều sự chồng nhau giữa các đoạn. Các đoạn bắt đầu từ các địa chỉ cách nhau $10h = 16$ byte và địa chỉ đầu của mỗi đoạn luôn kết thúc bằng chữ số 0. Chúng ta gọi 16 byte là một khúc (**paragraph**). Chúng ta gọi các địa chỉ chia hết cho 16 (các địa chỉ kết thúc bằng 0) là các biên giới khúc (paragraph boundary). Vì các đoạn có thể chồng nhau một địa chỉ **segment:offset** có thể không phải là duy nhất như các bạn thấy trong ví dụ dưới đây:

| ĐỊA CHỈ | NỘI DUNG |
|-------------------------|----------|
| ... | ... |
| 10021 | 00011110 |
| 10020 | 10100011 |
| Kết thúc đoạn 2 → 1001F | 11111111 |
| 1001E | 01011100 |
| ... | ... |
| 10010 | 10101010 |
| Kết thúc đoạn 1 → 1000F | 11001100 |
| 1000E | 01000111 |
| ... | ... |
| 10000 | 00011000 |
| Kết thúc đoạn 0 → OFFFF | 11111000 |
| OFFFE | 00011001 |
| | 00001111 |
| ... | ... |
| 00021 | 10011101 |
| Bắt đầu đoạn 2 → 00020 | 0000 |
| 0001F | 11111000 |
| ... | ... |
| 00011 | 11111111 |
| Bắt đầu đoạn 1 → 00010 | 00000000 |
| 0000F | 01100101 |
| ... | ... |
| 00003 | 00000111 |
| 00002 | 01110000 |
| 00001 | 01010101 |
| Bắt đầu đoạn 0 → 00000 | 10101000 |

Hình 3.2 Vị trí các đoạn bộ nhớ

Ví dụ 3.1 Cho một ô nhớ có địa chỉ vật lý là 1256Ah hãy cho biết địa chỉ dạng segment:offset với các đoạn 1256h và 1240h.

Trả lời Gọi X là offset trong segment 1256h và Y là offset trong segment 1240h chúng ta có:

$$1256Ah = 12560 + X \text{ và } 1256Ah = 12400 + Y$$

Do đó

$$X = 1256Ah - 12560h = Ah$$

$$\text{và } Y = 1256Ah - 12400h = 16Ah$$

$$\text{Như vậy } 1256Ah = 1256:000A = 1240:016A$$

Cũng có thể tính được địa chỉ đoạn khi biết địa chỉ vật lý và offset.

Ví dụ 3.2 Một ô nhớ có địa chỉ vật lý 80FD2h, ở trong đoạn nào thì nó có offset bằng BFD2h ?

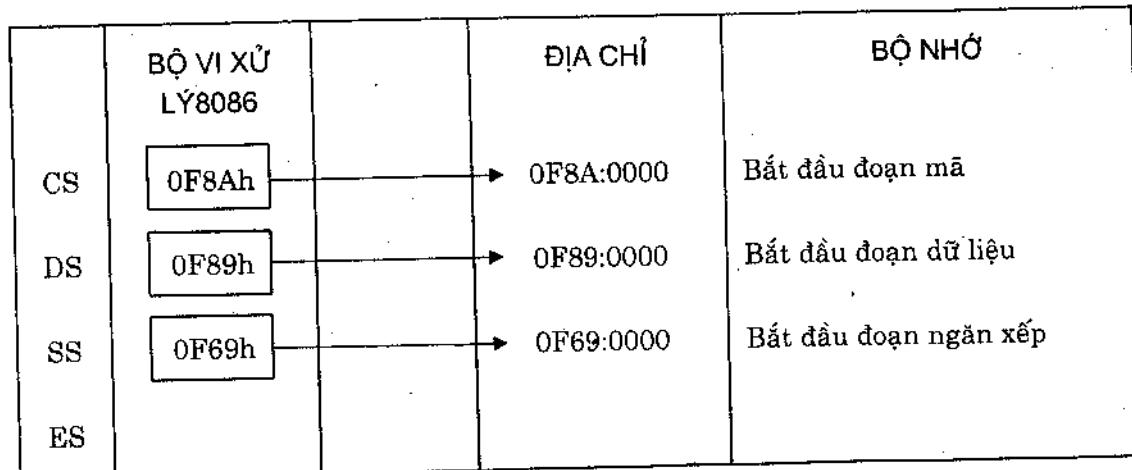
Trả lời Chúng ta biết rằng:

$$\text{địa chỉ vật lý} = \text{segment} * 10h + \text{offset}$$

Do đó $\text{segment} * 10h = \text{địa chỉ vật lý} - \text{offset}$
trong trường hợp này:

| | |
|----------------|--------|
| địa chỉ vật lý | 80FD2h |
| - offset | BFD2h |
| segment * 10h | 75000h |

Vậy địa chỉ đoạn cần tìm là 7500h.



Hình 3.3 Các thanh ghi đoạn

Các đoạn của chương trình

Bây giờ chúng ta hãy nói về các thanh ghi CS, DS, SS và ES. Một chương trình ngôn ngữ máy điển hình gồm các lệnh và dữ liệu. Ngoài ra còn có một cấu trúc dữ liệu khác gọi là ngăn xếp (stack) được bộ vi xử lý sử dụng để thực hiện các lời gọi thủ tục. Mã, dữ liệu và ngăn xếp của chương trình được nạp vào các đoạn bộ nhớ khác nhau, chúng ta gọi chúng là đoạn mã (code segment), đoạn dữ liệu (data segment) và đoạn ngăn xếp (stack segment).

Để theo dõi các đoạn khác nhau của chương trình, 8086 được cung cấp 4 thanh ghi đoạn để chứa các địa chỉ đoạn. Các thanh ghi CS, DS, SS lần lượt chứa địa chỉ của đoạn mã, đoạn dữ liệu và đoạn ngăn xếp. Nếu một chương trình cần truy nhập đến một đoạn dữ liệu thứ 2 nó có thể sử dụng thanh ghi thêm ES (extra segment).

Một chương trình không phải bao giờ cũng cần chiếm hết cả một đoạn 64 KB, đặc điểm chung nhau của các đoạn bộ nhớ cho phép các đoạn của một chương trình nhỏ hơn 64 KB có thể đặt gần lại với nhau. Hình 3.3 chỉ ra sự bố trí điển hình các đoạn của chương trình (địa chỉ đoạn và vị trí tương ứng các đoạn của chương trình là tùy ý).

Tại mọi thời điểm, chỉ những ô nhớ được định địa chỉ bởi 4 thanh ghi đoạn mới có thể được truy nhập, có nghĩa là chỉ có 4 đoạn bộ nhớ là hoạt động (active), tuy nhiên nội dung của các thanh ghi đoạn có thể được thay đổi bởi chương trình để truy nhập đến các đoạn khác nhau.

3.2.4 Các thanh ghi con trỏ và chỉ số

Các thanh ghi SP, BP, SI và DI thường trỏ tới các ô nhớ (nghĩa là chứa địa chỉ offset của các ô nhớ đó). Khác với các thanh ghi đoạn, các thanh ghi con trỏ và chỉ số được sử dụng trong các thao tác số học và một số thao tác khác nữa.

Thanh ghi SP (con trỏ ngăn xếp - stack pointer)

Thanh ghi SP được sử dụng kết hợp với thanh ghi SS để truy nhập đoạn ngăn xếp. Các thao tác có liên quan đến ngăn xếp được trình bày trong chương 8.

Thanh ghi BP (con trỏ cơ sở - base pointer)

Thanh ghi BP chủ yếu được sử dụng để truy nhập dữ liệu trong ngăn xếp. Tuy nhiên khác với thanh ghi SP, BP cũng có thể được sử dụng để truy nhập dữ liệu trong các đoạn khác.

Thanh ghi SI (chỉ số nguồn - source index)

Thanh ghi SI được sử dụng để trỏ tới các ô nhớ trong đoạn dữ liệu được định địa chỉ bởi thanh ghi DS. Bằng cách tăng nội dung của SI, chúng ta có thể truy nhập dễ dàng đến các ô nhớ liên tiếp.

Thanh ghi DI (chỉ số đích - destination index)

Thanh ghi DI thực hiện các chức năng tương tự như thanh ghi SI. Có một số lệnh gọi là các thao tác chuỗi sử dụng DI để truy nhập đến các ô nhớ được định địa chỉ đoạn bởi ES.

3.2.5 Thanh ghi con trỏ lệnh IP

Các thanh ghi bộ nhớ chúng ta vừa trình bày dùng để truy nhập dữ liệu, để truy nhập đến các lệnh, 8086 sử dụng các thanh ghi CS và IP. Thanh ghi CS chứa địa chỉ đoạn của lệnh tiếp theo, còn thanh ghi IP chứa offset lệnh đó. Thanh ghi IP được cập nhật mỗi khi có một lệnh được thực hiện để sao cho nó trỏ đến lệnh tiếp theo. Khác với các thanh ghi khác IP không thể bị tác động trực tiếp bởi các lệnh, do đó trong một lệnh thường không có mặt IP như một toán hạng.

3.2.6 Thanh ghi cờ

Mục đích của thanh ghi cờ là chỉ ra trạng thái của bộ vi xử lý. Để có thể làm được điều đó nó thiết lập các bit riêng biệt gọi là các cờ (flags). Có 2 loại cờ: cờ trạng thái (status flags) và cờ điều khiển (control flags). Cờ trạng thái phản ánh kết quả của một lệnh mà bộ xử lý thực hiện. Chẳng hạn khi một phép trừ cho kết quả bằng 0 cờ ZF (zero flag) được thiết lập 1 (true), chỉ thị tiếp theo có thể kiểm tra cờ ZF và rẽ nhánh đến các chỉ thị khác có nhiệm vụ liên quan đến kết quả 0 của chỉ thị trước.

Cờ điều khiển cho phép hoặc không cho phép một thao tác nào đó của bộ xử lý, chẳng hạn khi cờ IF (interrupt flag) bị xoá về 0, việc vào từ bàn phím bị bộ xử lý bỏ qua. Các cờ trạng thái được trình bày trong chương 5 còn các cờ điều khiển được trình bày trong chương 11 và 15.

3.3 Tổ chức của máy PC

Một hệ thống máy vi tính được tạo lên bởi cả phần cứng lẫn phần mềm, trong đó phần mềm dùng để điều khiển các thao tác phần cứng. Vì vậy để hiểu rõ các thao tác của máy vi tính, bạn cũng đồng thời phải nghiên cứu về phần mềm điều khiển nó.

3.3.1 Hệ điều hành

Một bộ phận quan trọng của phần mềm cho một máy vi tính là **hệ điều hành** (operating system). Chức năng của hệ điều hành là kết hợp các thao tác của tất cả các thiết bị tạo lên một hệ thống máy vi tính. Sau đây là một số chức năng của hệ điều hành :

1. Đọc và thực hiện các lệnh mà người sử dụng đưa vào
2. Thực hiện các thao tác vào/ra
3. Đưa ra các thông báo lỗi.
- 4 Quản lý bộ nhớ và các nguồn dữ liệu khác.

Hiện nay hệ điều hành thông dụng nhất cho các máy vi tính IBM PC là **hệ điều hành DOS** (Disk Operating System) còn được gọi là PC DOS hay MS DOS. DOS được thiết kế cho các máy tính dựa trên cơ sở các bộ vi xử lý 8086/8088. Bởi vậy nó chỉ có thể quản lý 1 Megabyte bộ nhớ và không có khả năng làm việc đa nhiệm. Tuy nhiên nó có thể sử dụng trong các máy 80286, 80386, 80486 khi chúng chạy trong chế độ địa chỉ thực.

Một trong vô vàn những chức năng của DOS là đọc và ghi dữ liệu trên đĩa. Các chương trình và các thông tin khác được lưu trên đĩa dưới dạng các **tệp** (files). Mỗi tệp có một **tên tệp** (file name), tên tệp được tạo bởi từ 1 đến 8 ký tự và sau là **phần mở rộng** tùy chọn gồm một dấu chấm và từ 1 đến 3 ký tự. Phần mở rộng thường được sử dụng để xác định kiểu của tệp. Ví dụ COMMAND.COM có tên tệp là COMMAND và phần mở rộng là .COM.

Có một số phiên bản (version) của DOS, với mỗi phiên bản mới lại có thêm những khả năng mới. Hầu hết các chương trình thương mại đều cần sử dụng DOS có version 2.1 trở lên. DOS không phải chỉ là một chương trình, nó chứa một loạt các chương trình phục vụ. Người sử dụng yêu cầu phục vụ bằng cách đánh vào các lệnh. Version mới nhất của DOS - DOS 6.0 cung cấp **giao diện đồ họa** với người sử dụng (Graphical User Interface - GUI), nó cho phép sử dụng con chuột trong môi trường DOS.

Một chương trình của DOS dùng để diễn đạt các lệnh của người sử dụng là COMMAND.COM. Nó có nhiệm vụ tạo ra dấu nhắc của DOS như là A>_ , và đọc các lệnh của người sử dụng. Có 2 loại lệnh của người sử dụng là lệnh trong (internal) và lệnh ngoài (external) còn được gọi là các lệnh **nội trú** và **ngoại trú**.

Các lệnh nội trú được thực hiện bởi các chương trình của DOS đã được nạp vào trong bộ nhớ còn các lệnh ngoại trú sử dụng các chương trình chưa được nạp vào bộ nhớ hay các chương trình ứng dụng. Trong các thao tác thông thường rất nhiều các chương trình của DOS không được nạp để tiết kiệm bộ nhớ.

Vì các chương trình của DOS nằm trên đĩa nên phải có một chương trình thực hiện việc đọc đĩa khi bật nguồn máy tính. Trong chương 1 chúng ta đã đề cập tới các chương trình hệ thống được chứa trong ROM mà không bị mất đi khi tắt máy. Trong các máy PC chúng được gọi là các chương trình **BIOS** (Basic Input/Output System)

BIOS

Các chương trình của BIOS thực hiện các thao tác vào/ra cho các máy PC. Khác với các chương trình của DOS, các chương trình của BIOS không hoạt động trên tất cả họ PC mà chỉ trên một loại máy nhất định. Mỗi mẫu máy PC có một cấu hình phần cứng riêng biệt và các chương trình BIOS của riêng nó. Các chương trình này tham chiếu đến các thanh ghi của các cổng vào/ra trong máy cho các thao tác vào và ra. Tất cả các thao tác vào/ra của DOS đều có thể thực hiện một cách hoàn thiện bằng các chương trình của BIOS.

Một chức năng quan trọng khác của BIOS là kiểm tra phần cứng của máy và nạp các chương trình của DOS. Trong phần 3.3.4 chúng ta sẽ bàn về vấn đề nạp các chương trình của DOS.

Để các chương trình của DOS và các chương trình khác có thể sử dụng các chương trình của BIOS, địa chỉ các chương trình của BIOS, còn gọi là các **vector ngắt** (interrupt vectors), được đặt trong bộ nhớ bắt đầu từ địa chỉ 00000h. Một số chương trình của DOS cũng có các địa chỉ của chúng đặt ở đó.

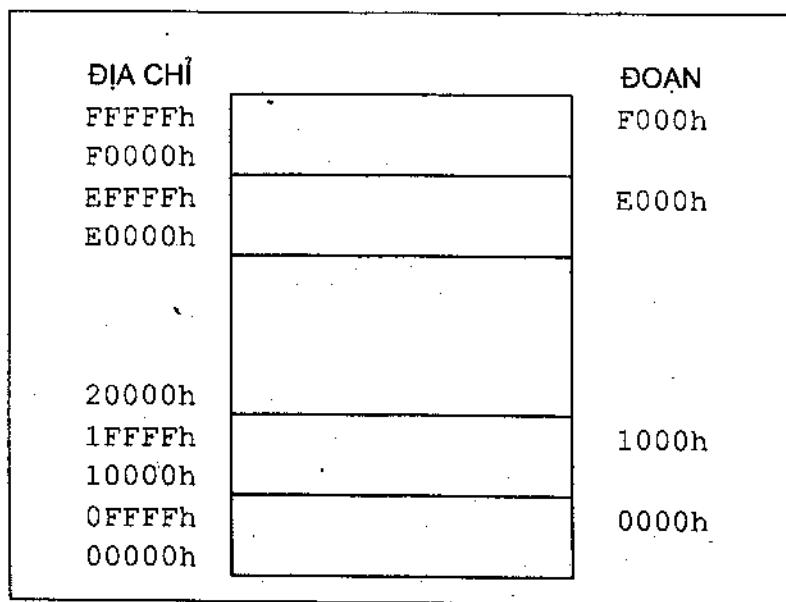
Vì IBM có các chương trình BIOS thuộc bản quyền của mình, các máy IBM tương thích cũng phải có những chương trình BIOS của riêng chúng. Mức độ tương thích được đánh giá bằng sự phù hợp của các chương trình của chúng với IBM BIOS.

3.3.2 Tổ chức bộ nhớ của PC

Như đã chỉ ra trong phần 3.2.3, các bộ vi xử lý 8086/8088 có khả năng định địa chỉ tới 1 Mégabyte bộ nhớ. Tuy nhiên không phải toàn bộ bộ nhớ đều giành cho các chương trình ứng dụng, một số vùng nhớ có ý nghĩa đặc biệt đối với bộ xử lý. Chẳng hạn, 1 KB đầu tiên được dùng cho bảng các vectơ ngắt.

Các vùng nhớ khác được IBM giành riêng cho các mục đích đặc biệt như chứa các chương trình của BIOS và **bộ nhớ màn hình** (video display memory). Bộ nhớ màn hình chứa các thông tin sẽ được hiển thị lên màn hình.

Để trình bày bản đồ bộ nhớ của IBM PC, sẽ rất có ích nếu chúng ta phân bộ nhớ thành các đoạn tách rời nhau. Chúng ta bắt đầu bằng đoạn 0, đoạn này kết thúc ở vị trí 0FFFFh, do đó đoạn tiếp theo phải bắt đầu tại địa chỉ 10000h = 1000:0000. Đoạn 1000h kết thúc tại 1FFFFh và đoạn tiếp theo bắt đầu tại 20000h = 2000:0000. Như vậy các đoạn rời nhau là 0000h, 1000h, 2000h, ...F000h có nghĩa là bộ nhớ được chia làm 16 đoạn rời nhau. Xem hình 3.4



Hình 3.4. Bộ nhớ được phân thành các đoạn rời nhau

Chỉ có 10 đoạn rời nhau đầu tiên được DOS sử dụng để nạp và thực hiện các chương trình ứng dụng. Mười đoạn này, từ 0000h đến 9000h cho chúng ta 64 KB bộ nhớ. Kích thước bộ nhớ của các máy vi tính trên cơ sở 8086/8088 được hiểu là những đoạn bộ nhớ này. Chẳng hạn một máy PC với bộ nhớ 512 KB chỉ có 8 đoạn bộ nhớ trong số 10 đoạn nêu trên.

Các đoạn A000h và B000h được sử dụng làm bộ nhớ hiển thị. Đoạn C000h và E000h được để giành. Đoạn F000h là đoạn đặc biệt vì nó là bộ nhớ ROM chủ

không phải RAM, trong đó có chứa các chương trình của BIOS và ROM BASIC. Hình 3.5 cho thấy sự bố trí của bộ nhớ.

| VÙNG NHỚ | ĐỊA CHỈ |
|---|---------|
| BIOS | F0000h |
| Để giành | E0000h |
| Để giành | D0000h |
| Để giành | C0000h |
| VIDEO | B0000h |
| VIDEO | A0000h |
| Vùng giành cho các chương trình ứng dụng | |
| DOS | |
| BIOS và dữ liệu của DOS | 00400h |
| Các vec tơ ngắt | 00000h |

Hình 3.5 Bản đồ bộ nhớ của PC

| Các địa chỉ cổng | Mô tả |
|------------------|--------------------------------|
| 20h - 21h | Bộ điều khiển ngắt |
| 60h - 63h | Bộ điều khiển bàn phím |
| 200h - 20Fh | Bộ điều khiển trò chơi |
| 2F8h - 2FFh | Cổng nối tiếp thứ 2 (COM2) |
| 320h - 32Fh | Đĩa cứng |
| 378h - 37Fh | Cổng máy in song song thứ nhất |
| 3C0h - 3CFh | EGA |
| 3D0h - 3DFh | CGA |
| 3F8h - 3FFh | Cổng nối tiếp thứ nhất (COM1) |

Bảng 3.1 Các cổng vào/ra thông dụng của PC

3.3.3 Các cổng vào/ra

8086/8088 cung cấp 64 KB cho các cổng vào/ra. Địa chỉ của các cổng thông dụng được trình bày trong bảng 3.1. Chúng tôi khuyên các bạn không nên lập trình trực tiếp đến các cổng vì việc sử dụng các địa chỉ cổng có thể khác nhau giữa các loại máy tính.

3.3.4 Thao tác khởi động

Khi máy tính được bật lên, bộ vi xử lý 8086/8088 được đặt vào trạng thái reset, thanh ghi CS được thiết lập giá trị FFFFh và thanh ghi IP được thiết lập giá trị 0000h. Như vậy chỉ thị được thực hiện đầu tiên đặt ở ô nhớ FF0h, ô nhớ này ở trong ROM và nó chứa chỉ thị sẽ chuyển điều khiển đến điểm đầu của các chương trình BIOS.

Các chương trình của BIOS đầu tiên kiểm tra hệ thống và bộ nhớ, sau đó nó khởi tạo bảng vectơ ngắt và vùng dữ liệu của BIOS. Cuối cùng BIOS nạp hệ điều hành từ đĩa hệ thống. Công việc này được thực hiện thông qua 2 bước: đầu tiên BIOS nạp một chương trình nhỏ gọi là **chương trình khởi động** (boot program) sau đó chương trình này nạp các chương trình của hệ điều hành thực sự. Chương trình khởi tạo có tên gọi như vậy bởi vì nó là một phần của hệ điều hành, việc sử dụng nó để nạp hệ điều hành cũng giống như máy tính tự khởi động nó bằng một **chương trình mồi** (bootstraps). Việc sử dụng chương trình khởi động tránh cho BIOS khỏi có những thay đổi gây ra bởi những version khác nhau của hệ điều hành, đồng thời cũng làm giảm kích thước của BIOS. Sau khi hệ điều hành được nạp, điều khiển được chuyển cho COMMAND.COM.

TỔNG KẾT

Trong chương này chúng ta đã nghiên cứu những vấn đề sau:

- Họ máy tính cá nhân IBM PC bao gồm các loại PC, PC XT, PC AT, PS/1 Và PS/2, chúng đều sử dụng bộ vi xử lý của họ INTEL 8086.
- Họ 8086 bao gồm các bộ vi xử lý 8086, 8088, 80186, 80188, 80286, 80386, 80386SX, 80486 và 80486SX.
- Các bộ vi xử lý 8086 và 8088 có cùng tập lệnh, tập lệnh này tạo thành tập lệnh cơ sở cho các bộ vi xử lý khác.
- Bộ vi xử lý 8086 có 14 thanh ghi. Các thanh ghi được chia thành các thanh ghi dữ liệu, thanh ghi đoạn, thanh ghi con trỏ và chỉ số và thanh ghi cờ.
- Các thanh ghi dữ liệu là AX, BX, CX và DX. Các thanh ghi này có thể sử dụng như những thanh ghi công dụng chung đồng thời chúng cũng có những chức năng đặc biệt. Các byte cao và thấp của chúng có thể truy xuất riêng biệt.
- Mỗi byte bộ nhớ có một địa chỉ 20 bit (5 chữ số hex), bắt đầu bằng địa chỉ 00000h.
- Một segment là một khối bộ nhớ 64 KB, các địa chỉ của bộ nhớ có thể được viết dưới dạng segment:offset. Địa chỉ vật lý nhận được bằng cách nhân địa chỉ đoạn với 10h rồi cộng với offset.
- Các thanh ghi đoạn là CS, DS, SS và ES. Khi một chương trình ngôn ngữ máy được thực hiện, các thanh ghi này chứa các địa chỉ đoạn của đoạn mã đoạn dữ liệu, đoạn ngăn xếp và đoạn dữ liệu phụ.
- Các thanh ghi con trỏ và chỉ số là SP, BP, SI, DI và IP, SP được sử dụng riêng cho đoạn ngăn xếp. BP có thể sử dụng để truy nhập đoạn ngăn xếp cũng như các đoạn dữ liệu khác. SI và DI có thể được sử dụng để truy nhập dữ liệu trong các mảng.
- IP chứa địa chỉ của lệnh tiếp theo sắp được thi hành.
- Thanh ghi cờ chứa các cờ trạng thái và cờ điều khiển. Các cờ trạng thái được thiết lập theo kết quả của các thao tác. Cờ điều khiển có thể được sử dụng để cho phép hoặc không cho phép một thao tác nào đó của bộ vi xử lý.
- DOS là tập hợp các chương trình liên kết những thao tác của máy tính. Chương trình thực hiện những lệnh của người sử dụng có tên là COMMAND.COM.

- Thông tin được lưu trên đĩa dưới dạng các tệp. Mỗi tệp có tên tệp và phần mở rộng.
- Các chương trình của BIOS dùng để thực hiện các thao tác vào/ra. Tính tương thích của các máy tính PC với IBM PC phụ thuộc vào một điều là các chương trình BIOS của chúng phù hợp với các chương trình BIOS của IBM PC không đến mức độ nào.
- Các chương trình của BIOS có trách nhiệm kiểm tra hệ thống và nạp hệ điều hành khi máy tính được bật lên.

Các thuật ngữ tiếng Anh

| | |
|--|---|
| Basic input/output system, BIOS | Các chương trình cơ bản quản lý việc vào/ra. |
| Boot program | Chương trình nạp hệ điều hành trong quá trình khởi động. |
| Code segment | Đoạn mã - đoạn bộ nhớ chứa các lệnh của chương trình ngôn ngữ máy. |
| COMMAND.COM | Bộ xử lý lệnh của DOS. |
| Control flags | Cờ điều khiển, được dùng để cho phép hoặc không cho phép một thao tác của bộ vi xử lý. |
| Data segment | Đoạn dữ liệu- đoạn bộ nhớ chứa dữ liệu của chương trình ngôn ngữ máy. |
| Disk Operating System,DOS | Hệ điều hành cho các máy tính IBM PC. |
| External command | Lệnh ngoại trú - lệnh tương ứng với các chương trình nằm trên đĩa. |
| File | Tệp- một tập hợp dữ liệu có tên gọi và tổ chức. Dữ liệu được xem như một khối riêng để lưu trữ trên các thiết bị như đĩa. |
| File extension | Phần mở rộng của tệp gồm từ 1 đến 3 ký tự theo sau một dấu chấm. Nó được dùng để xác định kiểu tệp. |
| File name | Tên của tệp gồm từ 1 đến 8 ký tự. |
| Flags | Các bit của thanh ghi cờ. |

| | |
|--|---|
| Graphic User Interface,GUI | Giao diện đồ họa với người sử dụng, gồm các con trỏ và các biểu tượng đồ họa. |
| Internal command | Lệnh nội trú- các lệnh của DOS thực hiện các chương trình có trong bộ nhớ. |
| Interrupt vectors | Các vectơ ngắt- chính là địa chỉ các chương trình của DOS và BIOS. |
| Logical address | Địa chỉ lô gic - địa chỉ được viết dưới dạng segment:offset. |
| Memory protection | Khả năng của bộ xử lý bảo vệ bộ nhớ sử dụng bởi một chương trình khỏi sự sử dụng của một chương trình khác. |
| Memory segment | Đoạn bộ nhớ - một khối bộ nhớ 64 KB. |
| Multitasking | Khả năng làm việc đa nhiệm- khả năng của máy tính có thể thực hiện vài chương trình cùng lúc. |
| Offset(of a memorylocation) | Địa chỉ tương đối- số byte ô nhớ tính từ địa chỉ đầu của một đoạn. |
| Operating system | Hệ điều hành- tập hợp các chương trình liên kết các thao tác của các thiết bị tạo lên một hệ thống máy vi tính. |
| Paragraph | Một đoạn 16 byte. |
| Paragraph boundary | Biên giới đoạn - địa chỉ dạng hex kết thúc bằng số 0 |
| Physical address | Địa chỉ vật lý- địa chỉ 20 bit của một ô nhớ trong các máy tính dựa trên cơ sở bộ vi xử lý 8086. |
| Protected(virtual address) mode | Chế độ bảo vệ- Một chế độ xử lý trong đó bộ nhớ sử dụng bởi chương trình này được bảo vệ khỏi sự truy nhập của chương trình khác. |
| Real address mode | Chế độ địa chỉ thực- Một chế độ xử lý trong đó các địa chỉ sử dụng trong chương trình tương ứng với địa chỉ của bộ nhớ vật lý |
| Segment number | Địa chỉ đoạn- Con số dùng để xác định một đoạn bộ nhớ. |

| | |
|-----------------------------|---|
| Stack | Ngăn xếp- Một cấu trúc dữ liệu được bộ xử lý sử dụng để thực hiện các lời gọi thủ tục. |
| Stack segment | Đoạn ngăn xếp- Đoạn bộ nhớ chứa ngăn xếp của chương trình ngôn ngữ máy. |
| Status flags | Cờ trạng thái- Các cờ phản ánh kết quả của một thao tác của bộ xử lý. |
| Video display memory | Bộ nhớ màn hình- Bộ nhớ sử dụng để lưu trữ dữ liệu sẽ được hiển thị trên màn hình. |
| Virtual memory | Bộ nhớ ảo - Khả năng của các bộ vi xử lý tiên tiến có thể sử dụng bộ nhớ ngoài như là bộ nhớ trong và bởi vậy có thể thực hiện được các chương trình quá lớn để có thể chứa ở bộ nhớ trong. |

Bài tập

1. Trình bày những điểm khác nhau cơ bản của các bộ vi xử lý 8086 và 80286 ?
 2. Trình bày những điểm khác nhau giữa thanh ghi và ô nhớ ?
 3. Nêu những chức năng đặc biệt của các thanh ghi dữ liệu: AX, BX, CX và DX ?
 4. Hãy xác định địa chỉ vật lý của ô nhớ có địa chỉ lô gic 0A51h:CD90h
 5. Một ô nhớ có địa chỉ vật lý là 4A37Bh hãy tính :
 - a. Địa chỉ offset của nó nếu địa chỉ đoạn là 40FFh ?
 - b. Địa chỉ đoạn của nó nếu địa chỉ offset là 123Bh ?
 6. Thế nào là một biên giới đoạn ?
 7. Làm sao để đánh giá độ tương thích của một máy tính tương thích IBM PC với một máy tính IBM PC thực sự ?
 8. Cho biết kích thước cực đại bộ nhớ mà DOS giành cho việc nạp và thực hiện chương trình nếu giả sử DOS chiếm tới byte 0FFFFh.
- Để làm các bài tập sau các bạn hãy tham khảo phụ lục B
9. Hãy cho biết các lệnh của DOS thực hiện các việc sau: (giả sử ổ đĩa A là ổ đĩa hiện hành)
 - a. Chép tệp FILE1 trong thư mục hiện hành sang FILE1A trên đĩa ổ B.
 - b. Chép tất cả các tệp có phần mở rộng .ASM sang đĩa trong ổ B.
 - c. Xoá tất cả các tệp có phần mở rộng .BAK
 - d. Liệt kê tất cả các tệp trong thư mục hiện hành có tên bắt đầu bằng chữ cái A
 - e. Thiết lập ngày 21 tháng 9 năm 1991 cho máy tính.
 - f. In tệp FILE5.ASM ra máy in
 10. Giả sử (a) thư mục gốc có các thư mục con A, B và C; (b) A có các thư mục con A1 và A2; (c) A1 có một thư mục con A1A. Viết các lệnh của DOS thực hiện các việc sau:
 - a. Tạo một thư mục đầu tiên của cây.
 - b. Biến A1A thành thư mục hiện hành.
 - c. Yêu cầu DOS hiển thị thư mục hiện hành.
 - d. Xoá thư mục đầu tiên của cây.

Chương 4

GIỚI THIỆU HỢP NGỮ CHO IBM PC.

Tổng quan.

Chương này bao gồm các bước cần thiết để tạo ra, hợp dịch và thực hiện một chương trình Hợp ngữ. Đến cuối chương, bạn đã có thể viết được các chương trình đơn giản nhưng khá thú vị, thực hiện một số công việc hữu ích và chạy chúng trên máy tính.

Cũng như với bất cứ ngôn ngữ nào khác, bước đầu tiên ta phải học cú pháp. Điều này đối với Hợp ngữ tương đối đơn giản. Tiếp theo, chúng tôi sẽ chỉ ra cách khai báo biến và giới thiệu các lệnh số học và dịch chuyển số liệu cơ bản. Cuối cùng chúng tôi sẽ trình bày cách tổ chức chương trình. Bạn sẽ thấy các chương trình Hợp ngữ bao gồm mã lệnh, số liệu và ngăn xếp giống như chương trình mã máy.

Bởi vì các lệnh của Hợp ngữ quá cơ bản nên thực hiện các thao tác vào/ra trong Hợp ngữ khó khăn hơn nhiều so với các ngôn ngữ bậc cao. Chúng tôi sử dụng các hàm của DOS cho các thao tác vào/ra vì chúng khá dễ dùng và đủ nhanh cho hầu hết các yêu cầu ứng dụng.

Một chương trình Hợp ngữ trước khi có thể thực hiện phải được chuyển sang dạng mã máy. Mục 4.10 sẽ giải thích các bước. Nó minh họa vài kỹ thuật lập trình chuẩn bằng Hợp ngữ và như là mẫu cho các bài tập.

4.1 Cú pháp của Hợp ngữ.

Các chương trình Hợp ngữ được dịch ra các chỉ thị máy bằng một chương trình biên dịch vì thế chúng phải được viết ra sao cho phù hợp với các khuôn mẫu của trình biên dịch đó. Trong cuốn sách này chúng tôi sẽ sử dụng trình biên dịch MICROSOFT MACRO ASSEMBLER (MASM). Mã lệnh Hợp ngữ nói chung

không phân biệt chữ hoa hay thường nhưng chúng tôi sử dụng chữ hoa để phân biệt mã lệnh với phần còn lại của chương trình.

Các dòng lệnh.

Các chương trình bao gồm các dòng lệnh, mỗi dòng lệnh trên một dòng. Một dòng lệnh là một lệnh mà trình biên dịch sẽ dịch ra mã máy hay là một hướng dẫn biên dịch để chỉ dẫn cho trình biên dịch thực hiện một vài nhiệm vụ đặc biệt nào đó, chẳng hạn dành chỗ cho một biến nhớ hay khai báo một chương trình con. Mỗi lệnh hay hướng dẫn biên dịch thường có 4 trường:

| Tên | Toán tử | Toán hạng | Lời bình |
|-----|---------|-----------|----------|
|-----|---------|-----------|----------|

Các trường phải được phân cách nhau bằng ít nhất một ký tự trống hạy TAB. Cũng không bắt buộc phải xắp xếp các trường theo cột nhưng chúng nhất định phải xuất hiện theo đúng thứ tự nêu trên.

Ví dụ một lệnh:

START: MOV CX, 5 ;khởi tạo bộ đếm.

Trong ví dụ này, trường tên là nhãn START, toán tử là MOV, toán hạng là CX và 5, lời bình là 'khởi tạo bộ đếm'.

Ví dụ về hướng dẫn biên dịch:

MAIN PROC

MAIN là tên và toán hạng là PROC. Dẫn hướng biên dịch này khai báo một chương trình con có tên PROC.

4.1.1 Trường tên.

Trường tên được sử dụng làm nhãn lệnh, các tên thủ tục và các tên biến. Chương trình biên dịch sẽ chuyển các tên thành các địa chỉ bộ nhớ.

Các tên có thể có chiều dài từ 1 đến 31 ký tự, có thể chứa các chữ cái, chữ số và các ký tự đặc biệt (? . @ _ \$ %). Không được phép chèn dấu trống vào giữa một tên. Nếu sử dụng dấu chấm (.) thì nó phải đứng đầu tiên. Các tên không được bắt đầu bằng một chữ số. Chương trình biên dịch không phân biệt chữ hoa và chữ thường trong tên.

Ví dụ các tên hợp lệ:

COUNTER1

```

@character
SUM_OF_DIGITS
$1000
DONE?
.TEST

```

Ví dụ các tên không hợp lệ:

| | |
|----------|---------------------------------------|
| TWO WORD | chứa một khoảng trắng |
| 2abc | bắt đầu bằng một chữ số |
| A45.28 | dấu chấm không phải là ký tự đầu tiên |
| YOU&ME | chứa một ký tự không hợp lệ |

4.1.2 Trường toán tử.

Trong một lệnh, trường toán tử chứa mã lệnh dạng tượng trưng. Chương trình biên dịch sẽ chuyển mã lệnh dạng tượng trưng sang mã lệnh của ngôn ngữ máy. Tượng trưng của mã lệnh thường biểu thị chức năng của các thao tác. Ví dụ như: MOV, ADD, SUB.

Trong một hướng dẫn biên dịch, trường toán tử chứa **toán tử giả** (pseudo-op). Các toán tử giả sẽ không được dịch ra mã máy mà đơn giản chúng chỉ báo cho trình biên dịch làm một việc gì đó. Chẳng hạn toán tử giả PROC được dùng để tạo ra một thủ tục.

4.1.3 Trường toán hạng.

Đối với một chỉ thị, trường toán hạng xác định dữ liệu sẽ được các thao tác động lên. Một chỉ thị có thể không có, có 1 hoặc 2 toán hạng. Ví dụ:

| | |
|--------------|---|
| NOP | không toán hạng, không làm gì cả. |
| INC AX | một toán hạng, cộng 1 vào nội dung AX. |
| ADD WORD1, 2 | hai toán hạng, cộng 2 vào từ nhớ WORD1. |

Trong một chỉ thị hai toán hạng, toán hạng đầu tiên gọi là toán hạng đích. Nó có thể là một thanh ghi hoặc một ô nhớ, là nơi chứa kết quả (lưu ý một số chỉ thị không lưu giữ kết quả). Toán hạng thứ hai là toán hạng nguồn. Các chỉ thị thường không làm thay đổi toán hạng nguồn.

Đối với một hướng dẫn biên dịch, trường toán hạng thường chứa thêm thông tin về việc dẫn hướng.

4.1.4 Trường lời giải thích.

Người lập trình thường sử dụng trường lời giải thích của một dòng lệnh để giải thích dòng lệnh đó làm cái gì. Mở đầu trường này là một dấu chấm phẩy (;) và trình biên dịch bỏ qua mọi cái được đánh vào sau dấu chấm phẩy này. Lời giải thích có thể tùy ý (có hoặc không) nhưng vì Hợp ngữ là ngôn ngữ bậc thấp cho nên ta hầu như không thể hiểu được một chương trình viết bằng Hợp ngữ khi không có lời bình. Trên thực tế điền các lời giải thích vào hầu hết các dòng lệnh là một phương pháp học lập trình tốt. Nghệ thuật chú giải sẽ được phát triển cùng với quá trình thực hành.

Không nên viết những điều đã quá rõ ràng như:

```
MOV      CX, 0           ; chuyển 0 vào CX.
```

Thay vào đó, ta nên sử dụng các lời giải thích để đặt các chỉ thị vào trong ngữ cảnh của chương trình:

```
MOV      CX, 0           ; CX đếm số vòng lặp, khởi tạo 0
```

Cũng có thể tạo nên cả một dòng ghi chú và dùng chúng để tạo ra các dòng trống trong chương trình:

```
;  
; khởi tạo các thanh ghi.  
;  
MOV      AX, 0  
MOV      BX, 0
```

4.2 Dữ liệu chương trình.

Bộ vi xử lý chỉ thao tác với dữ liệu nhị phân, vì thế trình biên dịch phải chuyển đổi tất cả các dạng dữ liệu khác nhau thành các số nhị phân. Tuy nhiên trong một chương trình Hợp ngữ chúng ta có thể biểu diễn dữ liệu dưới dạng các số nhị phân, thập phân, số hex và thậm chí cả các ký tự.

Các số.

Một số nhị phân được viết như là một chuỗi các bit kết thúc bằng chữ cái 'B' hay 'b'. Ví dụ 1011b.

Một số thập phân là chuỗi các chữ số thập phân kết thúc bằng chữ cái 'D' hay 'd' (hoặc không có).

Một số hex phải bắt đầu bằng một chữ số thập phân và kết thúc bằng chữ cái 'H' hay 'h'. Ví dụ 0ABC_H (với cách này, trình biên dịch có thể biết được ký hiệu 'ABC_H' biểu diễn một biến có tên 'ABC' hay số hex ABC).

Tất cả các số kể trên có thể có dấu tuỳ ý.

Sau đây là các ví dụ các số hợp lệ và không hợp lệ trong MASM:

| Số | Kiểu |
|--------|--|
| 1100 | thập phân |
| 1100b | nhi phân |
| 12345 | thập phân |
| -3568D | thập phân |
| 1,234 | không hợp lệ, chứa ký tự không là chữ số |
| 1B4DH | số hex |
| 1B4D | số hex không hợp lệ, không kết thúc là 'h' |
| FFFFH | số hex không hợp lệ, không bắt đầu bằng một chữ số thập phân |
| 0FFFFh | số hex |

Các ký tự.

Các ký tự và chuỗi các ký tự phải được bao trong dấu nháy đơn hay nháy kép. Ví dụ "A", 'hello'. Các ký tự được trình biên dịch dịch ra mã ASCII của chúng. Vì vậy chương trình không phân biệt giữa 'A' và 41h (là mã ASCII của 'A').

Bảng 4.1 Các toán tử giả định nghĩa số liệu.

| Toán tử giả | Biểu diễn |
|-------------|-----------------|
| DB | Định nghĩa byte |
| DW | Định nghĩa word |

| | |
|----|--|
| DD | Định nghĩa từ kép |
| DQ | Định nghĩa 4 word (4 từ liên tiếp) |
| DT | Định nghĩa 10 byte (10 byte liên tiếp) |

4.3 Các biến.

Trong Hợp ngữ các biến có vai trò giống như trong các ngôn ngữ bậc cao. Mỗi biến có một kiểu dữ liệu và được chương trình gán cho một địa chỉ bộ nhớ. Các toán tử giả định nghĩa số liệu và ý nghĩa của chúng được liệt kê trong bảng 4.1. Mỗi toán tử giả có thể được dùng để thiết lập một hay nhiều dữ liệu của kiểu đã được đưa ra.

Trong phần này chúng ta sử dụng DB và DW để định nghĩa các biến kiểu byte và các biến kiểu word. Các toán tử giả khác được dùng trong chương 18 có liên quan đến các thao tác với số có độ chính xác kép và số không nguyên.

4.3.1 Các biến kiểu byte.

Dẫn hướng định nghĩa một biến kiểu byte của trình biên dịch có dạng sau đây:

| | | |
|-----|----|------------------|
| Tên | DB | giá trị khởi tạo |
|-----|----|------------------|

Trong đó toán tử giả DB được hiểu là “định nghĩa byte”

Ví dụ:

| | | |
|-------|----|---|
| ALPHA | DB | 4 |
|-------|----|---|

Với dẫn hướng này, Hợp ngữ sẽ gán tên ALPHA cho một byte nhỏ và khởi tạo nó giá trị 4. Một dấu chấm hỏi (?) đặt ở vị trí của giá trị khởi tạo sẽ tạo nên một byte không được khởi tạo. Ví dụ:

| | | |
|------|----|---|
| BYTE | DB | ? |
|------|----|---|

Giới hạn thập phân của các giá trị khởi tạo nằm trong khoảng từ -128 đến 127 với kiểu có dấu và từ 0 đến 255 với kiểu không dấu. Các khoảng này vừa đúng giá trị của một byte.

4.3.2 Các biến kiểu word.

Dẫn hướng định nghĩa một biến kiểu word của trình biên dịch có dạng sau đây:

| | | |
|-----|----|------------------|
| Tên | DW | giá trị khởi tạo |
|-----|----|------------------|

Toán tử giá DW có nghĩa là "định nghĩa word". Ví dụ:

WRD DW -2

Giống như với biến kiểu byte một dấu chấm hỏi ở vị trí giá trị khởi tạo có nghĩa là word không được khởi tạo giá trị đầu. Giới hạn thập phân của giá trị khởi tạo được xác định từ -32768 đến 32767 đối với kiểu có dấu và từ 0 đến 65535 đối với kiểu không dấu.

4.3.3 Các mảng.

Trong ngôn ngữ hợp ngữ, mảng chỉ là một chuỗi các byte nhỏ hay từ nhớ. Ví dụ để định nghĩa mảng 3 byte có tên B_ARRAY với các giá trị khởi tạo là 10h, 20h, 30h chúng ta có thể viết:

B_ARRAY DB 10h, 20h, 30h

Tên B_ARRAY được gán cho byte đầu tiên, B_ARRAY+1 cho byte thứ hai và B_ARRAY+2 cho byte thứ ba. Nếu như trình biên dịch gán địa chỉ offset 0200h cho B_ARRAY thì bộ nhớ sẽ như sau:

| Phần tử | Địa chỉ | Nội dung |
|-----------|---------|----------|
| B_ARRAY | 0200h | 10h |
| B_ARRAY+1 | 0201h | 20h |
| B_ARRAY+2 | 0202h | 30h |

Mảng các word có thể được định nghĩa một cách tương tự. Ví dụ:

W_ARRAY DW 1000, 356, 248, 13

sẽ tạo nên một mảng có 4 phần tử với các giá trị khởi tạo là 1000, 356, 248, 13. Từ đầu tiên được gán với tên W_ARRAY, từ tiếp theo gán với W_ARRAY+2, rồi đến W_ARRAY+4 v.v. Nếu mảng bắt đầu tại 0300h thì bộ nhớ sẽ như sau:

| Phần tử | Địa chỉ | Nội dung |
|-----------|---------|----------|
| W_ARRAY | 0300h | 1000d |
| W_ARRAY+2 | 0302h | 356d |
| W_ARRAY+4 | 0304h | 248d |
| W_ARRAY+6 | 0306h | 13d |

Byte thấp và byte cao trong một word.

Đôi khi chúng ta muốn tham chiếu đến byte thấp và byte cao của biến word. Giả sử ta định nghĩa:

```
WORD1    DB      1234H
```

byte thấp của WORD1 sẽ chứa 34h còn byte cao chứa 12h. Byte thấp có địa chỉ ký hiệu là WORD1, còn byte cao có địa chỉ ký hiệu là WORD1+1.

Các chuỗi ký tự.

Một chuỗi ký tự có thể được khởi tạo bằng mảng các mã ASCII. Ví dụ:

```
LETTER    DB      'ABC'
```

tương đương với:

```
LETTER    DB      41h, 42h, 43h
```

Trong một chuỗi, trình biên dịch phân biệt các chữ hoa và chữ thường. Vì vậy chuỗi 'abc' được dịch ra 3 byte với các giá trị 61h, 62h và 63h.

Cũng có thể kết hợp các ký tự và các số trong một định nghĩa. Ví dụ:

```
MSG        DB      'HELLO', 0Ah, 0Dh, 'S'
```

tương đương với:

```
MSG        DB      48h, 45h, 4Ch, 4Ch, 4Fh, 0Ah, 0Dh, 24h
```

4.4 Các hằng có tên.

Để tạo ra các mã lệnh Hợp ngữ dễ hiểu, người ta thường dùng các tên tượng trưng để biểu diễn các hằng số.

EQU (EQUates: coi bằng như, coi như).

Để gán tên cho hằng, chúng ta có thể sử dụng toán tử giả EQU. Cú pháp:

```
Tên      EQU      hằng_số
```

Ví dụ:

```
LF       EQU      0Ah
```

sẽ gán tên LF cho 0Ah, là mã ASCII của ký tự xuống dòng. Tên LF bây giờ có thể được dùng để thay cho 0Ah tại bất cứ đâu trong chương trình. Trình biên dịch sẽ dịch các lệnh:

MOV DL, 0AH
và:

MOV DL, LF

ra cùng một chỉ thị máy.

Phần tử bên phải EQU cũng có thể là một chuỗi. Ví dụ:

PROMPT EQU "TYPE YOUR NAME"

Sau đó thay vì:

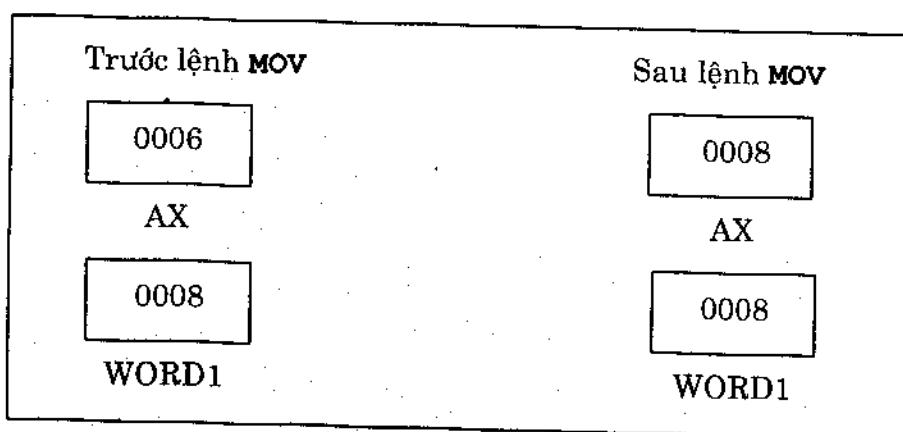
MSG DB "TYPE YOUR NAME"

Ta có thể viết:

MSG DB PROMPT

Chú ý: **Bộ nhớ không dành chỗ cho các hàng có tên** (khi biên dịch, nơi nào chứa tên hàng thì ở đó sẽ được thay đổi bởi giá trị của hàng).

Hình 4.1 Mov AX,WORD1



4.5 Vài lệnh cơ bản.

Hệ lệnh của bộ vi xử lý 8088 có đến hơn một trăm lệnh, trong đó có các lệnh được thiết kế dành riêng cho các bộ vi xử lý cao cấp (xem chương 20). Trong phần này chúng ta sẽ xem xét 6 lệnh tiện dụng nhất cho việc chuyển dữ liệu và thực hiện các phép tính số học. Các lệnh mà chúng tôi đưa ra ở đây có thể dùng được cho cả các toán hạng byte và word.

Trong phần sau đây, WORD1 và WORD2 là các biến kiểu word, BYTE1 và BYTE2 là các biến kiểu byte. Như đã nêu trong chương 3, AH là byte cao của thanh ghi AX; BL là byte thấp của BX.

Hình 4.2 XCHG AH, BL

| Trước XCHG | Sau XCHG | | | | |
|---|----------|----|---|----|----|
| <table border="1"><tr><td>1A</td><td>00</td></tr></table> | 1A | 00 | <table border="1"><tr><td>05</td><td>00</td></tr></table> | 05 | 00 |
| 1A | 00 | | | | |
| 05 | 00 | | | | |
| AH AL | AH AL | | | | |
| <table border="1"><tr><td>00</td><td>05</td></tr></table> | 00 | 05 | <table border="1"><tr><td>00</td><td>1A</td></tr></table> | 00 | 1A |
| 00 | 05 | | | | |
| 00 | 1A | | | | |
| BH BL | BH BL | | | | |

4.5.1 MOV và XCHG.

Lệnh MOV được sử dụng để chuyển dữ liệu giữa các thanh ghi, giữa một thanh ghi và một ô nhớ hoặc chuyển trực tiếp một số vào một thanh ghi hay ô nhớ. Cú pháp:

MOV đích, nguồn

Sau đây là một vài ví dụ:

MOV AX, WORD1

Lệnh này đọc là “chuyển WORD1 vào AX”. Nội dung của thanh ghi AX được thay bằng nội dung của ô nhớ WORD1. Nội dung của WORD1 không bị thay đổi. Nói một cách khác, một bản sao của WORD1 được gửi vào AX (Hình 4.1).

MOV AX, BX

AX lấy giá trị chứa trong BX, còn BX không bị thay đổi.

MOV AH, 'A'

Lệnh này chuyển số 41h (mã ASCII của 'A') vào thanh ghi AH. Giá trị trước đó của thanh ghi AH bị viết đè lên (thay bằng giá trị mới).

Lệnh XCHG (hoán chuyển) được dùng để hoán chuyển nội dung của hai thanh ghi, thanh ghi và một ô nhớ. Cú pháp là:

XCHG đích, nguồn

Ví dụ:

XCHG AH,BL

Lệnh này sẽ hoán chuyển nội dung của hai thanh ghi AH và BL, như vậy AH sẽ chứa nội dung trước đây của BL còn BL lại chứa nội dung trước đây của AH (Hình 4.2). Một ví dụ khác:

XCHG AX,WORD1

Lệnh sẽ hoán chuyển nội dung của thanh ghi AX và ô nhớ WORD1.

Bảng 4.2

Các khả năng kết hợp cho phép của các toán hạng trong lệnh MOV và XCHG

Chỉ thị MOV

| Toán hạng đích Toán hạng nguồn | Thanh ghi công dụng chung | Thanh ghi đoạn | Ô nhớ | Hằng số |
|---|------------------------------|-------------------|-------|---------|
| Thanh ghi công dụng chung | YES | YES | YES | NO |
| Thanh ghi đoạn | YES | NO | YES | NO |
| Ô nhớ | YES | YES | NO | NO |
| Hằng số | YES | NO | YES | NO |

Chỉ thị XCHG

| Toán hạng đích Toán hạng nguồn | Thanh ghi công dụng chung | Ô nhớ |
|--|---------------------------------|-------|
| Thanh ghi công dụng chung | YES | YES |
| Ô nhớ | YES | NO |

Các hạn chế của MOV và XCHG.

Vì lý do kỹ thuật, có một vài hạn chế khi sử dụng lệnh MOV và XCHG. Bảng 4.2 chỉ ra các khả năng kết hợp cho phép. Cần chú ý rằng các chỉ thị MOV và XCHG không hợp lệ trong trường hợp hai toán hạng cùng là các ô nhớ ví dụ :

MOV WORD1,WORD2 ;không hợp lệ

Nhưng chúng ta có thể giải quyết vấn đề này bằng cách sử dụng các thanh ghi, chẳng hạn:

MOV AX,WORD2
MOV WORD1,AX

4.5.2 Các chỉ thị ADD, SUB, INC và DEC.

Các chỉ thị ADD và SUB được sử dụng để cộng hoặc trừ nội dung của hai thanh ghi, một thanh ghi và một ô nhớ hoặc cộng (trừ) một số vào (từ) một thanh ghi hay một ô nhớ. Cú pháp:

ADD đích, nguồn
SUB đích, nguồn

Ví dụ:

ADD WORD1, AX

Chỉ thị này “cộng AX vào WORD1”, sẽ cộng nội dung của thanh ghi AX với nội dung của ô nhớ WORD1 và chứa tổng trong WORD1. AX không bị thay đổi (Hình 4.3).

SUB AX, DX

Trong ví dụ này “trừ DX từ AX”, giá trị của AX trừ đi giá trị của DX, kết quả được chứa trong AX, thanh ghi DX không bị thay đổi (Hình 4.4).

ADD BL, 5

Chỉ thị này cộng số 5 vào nội dung của thanh ghi BL.

Cũng giống như trường hợp MOV và XCHG, có một vài hạn chế khi kết hợp các toán hạng của ADD và SUB. Các trường hợp cho phép được tổng kết trong bảng 4.3. Phép cộng hay trừ trực tiếp giữa các ô nhớ là không hợp lệ. Ví dụ:

ADD BYTE1,BYTE2 ;không hợp lệ

Có một giải pháp là chuyển BYTE2 vào một thanh ghi trước khi cộng:

MOV AL,BYTE2 ;AL lấy giá trị BYTE2

ADD BYTE1,AL ;cộng vào BYTE1

INC (INCrement) được dùng để cộng 1 vào nội dung của một thanh ghi hay ô nhớ , DEC (DECrement) trừ 1 từ nội dung của một thanh ghi hay ô nhớ. Cú pháp:

INC đích
DEC đích

Ví dụ:

INC WORD1

cộng 1 vào nội dung của WORD1 (Hình 4.5).

DEC BYTE1

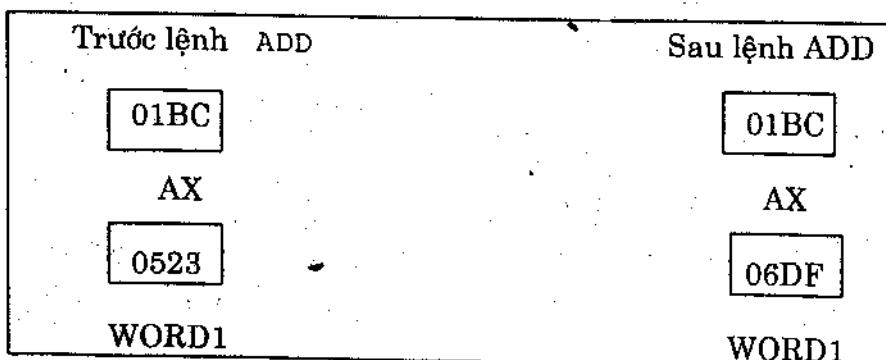
trừ 1 từ biến BYTE1 (Hình 4.6).

Bảng 4.3.

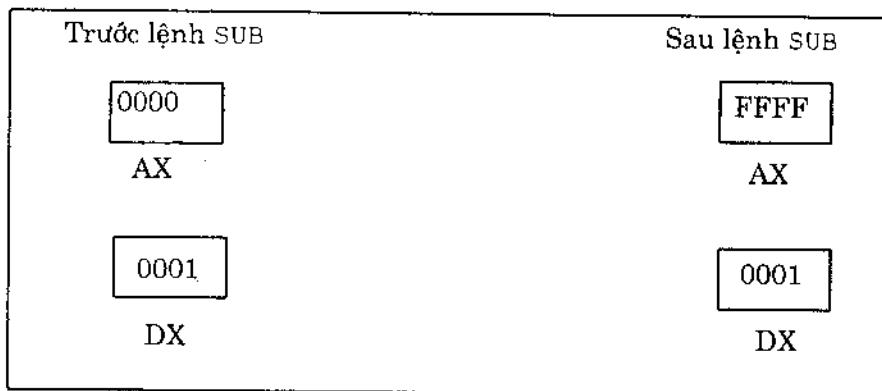
Các kết hợp cho phép của các toán hạng trong phép cộng và phép trừ.

| Toán hạng đích | | |
|---------------------------|---------------------------|-------|
| Toán hạng nguồn | Thanh ghi công dụng chung | Ô nhớ |
| Thanh ghi công dụng chung | yes | yes |
| Ô nhớ | yes | no |
| Hằng số | yes | yes |

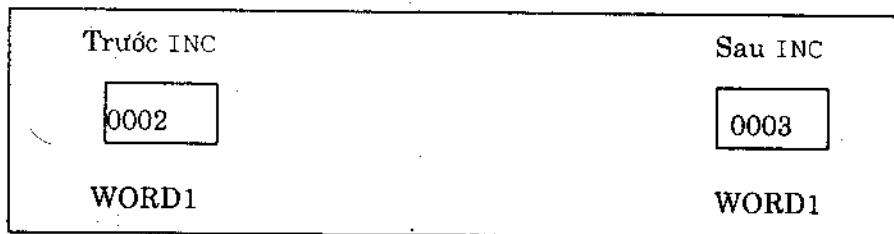
Hình 4.3. ADD WORD1,AX



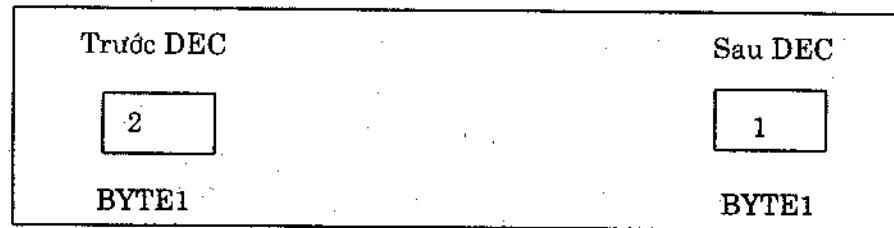
Hình 4.4. SUB AX, DX



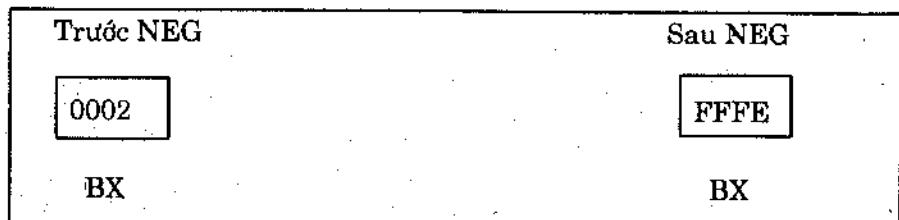
Hình 4.5. INC WORD1



Hình 4.6. DEC BYTE1



Hình 4.7. NEG BX



4.5.3 NEG

NEG dùng để phủ định nội dung của toán hạng đích. DEC sẽ thay thế nội dung bởi phần bù 2 của nó. Cú pháp:

NEG đích

Toán hạng đích có thể là một thanh ghi hay một ô nhớ. Ví dụ:

NEG BX

Sẽ phủ định nội dung của thanh ghi BX (Hình 4.7).

Kiểu quy ước của các toán hạng.

Các toán hạng của các lệnh hai toán hạng đã nêu phải có cùng kiểu, tức là cùng là byte hoặc word. Vì thế một lệnh như sau:

MOV AX,BYTE1 ;không hợp lệ

là không được phép. Tuy nhiên trình biên dịch chấp nhận cả hai lệnh sau đây:

MOV AH,'A'

và

MOV AX,'A'

Trong trường hợp đầu, trình biên dịch xét thấy do toán hạng đích AH là một byte nên toán hạng nguồn cũng phải là một byte và nó chuyển 41h vào AH. Đến trường hợp sau, vì toán hạng nguồn là một từ, nó giả thiết toán hạng đích cũng như vậy và chuyển 0041h vào AX.

4.6 Dịch từ ngôn ngữ bậc cao sang Hợp ngữ .

Để các bạn hiểu rõ hơn những lệnh đã nêu ở trên, chúng tôi sẽ dịch vài dòng lệnh gán của ngôn ngữ bậc cao sang Hợp ngữ. Chúng tôi chỉ sử dụng các lệnh MOV, ADD, SUB, INC, DEC và NEG mặc dù trong vài trường hợp nếu ta dùng các lệnh nêu sau này sẽ thực hiện tốt hơn. Trong các ví dụ, A và B là các biến word.

| Dòng lệnh | Dịch |
|-----------|---------------------------|
| B=A | MOV AX,A; chuyển A vào AX |
| | MOV B,AX; rồi sang B |

Như đã chỉ ra, chuyển trực tiếp giữa các ô nhớ là không hợp lệ, do vậy chúng ta phải chuyển nội dung của A sang một thang ghi trước khi chuyển sang B.

| | | |
|-------|----------|---------------|
| A=5-A | MOV AX,5 | ;đưa 5 vào AX |
| | SUB AX,A | ;AX chứa 5-A |
| | MOV A,AX | ;đưa nó vào A |

Ví dụ này minh họa một phương pháp dịch các lệnh gán: thực hiện các thao tác số học trong một thanh ghi (chẳng hạn AX), sau đó chuyển kết quả vào biến đích. Trong ví dụ này có một cách khác ngắn hơn:

| | |
|---------|--------|
| NEG A | ;A=-A |
| ADD A,5 | ;A=5-A |

Ví dụ tiếp theo sẽ chỉ ra phương pháp nhân với một hằng số.

| | | |
|---------|----------|--------------------|
| A=B-2*A | MOV AX,B | ;AX chứa B |
| | SUB AX,A | ;AX chứa B-A |
| | SUB AX,A | ;AX chứa B-2*A |
| | MOV A,AX | ;lưu kết quả vào A |

4.7 Cấu trúc chương trình.

Như chương 3 đã nêu, các chương trình bằng ngôn ngữ máy gồm có mã, dữ liệu và ngăn xếp. Mỗi phần chiếm một đoạn bộ nhớ. Chương trình bằng Hợp ngữ cũng có tổ chức như vậy. Trong trường hợp này, mã dữ liệu và ngăn xếp được cấu trúc như các đoạn chương trình. Mỗi đoạn chương trình sẽ được dịch thành một đoạn bộ nhớ bởi trình biên dịch.

Chúng ta sử dụng các định nghĩa đoạn đơn giản hóa mà đã được dùng cho MASM 5.0. Ta sẽ thảo luận kỹ hơn trong chương 14 cùng với các định nghĩa đoạn toàn phần.

4.7.1 Các chế độ bộ nhớ.

Kích thước của đoạn mã và dữ liệu trong một chương trình có thể được xác định bằng cách chỉ ra chế độ bộ nhớ nhờ sử dụng dẫn hướng biên dịch .MODEL Cú pháp:

.MODEL *kiểu_bộ_nhớ*

Các chế độ bộ nhớ thường sử dụng nhất là: SMALL, MEDIUM, COMPACT và LARGE. Chúng được trình bày ở hình 4.4. Trừ khi có rất nhiều mã lệnh hay số liệu, kiểu thích hợp nhất là SMALL. Dẫn hướng biên dịch .MODEL phải được đưa vào trước bất kỳ một định nghĩa đoạn nào.

Bảng 4.4. Các kiểu bộ nhớ.

| Kiểu | Miêu tả |
|------|---------|
|------|---------|

| | |
|-------|---|
| SMALL | Mã lệnh trong một đoạn Dữ liệu trong một đoạn. |
|-------|---|

| | |
|---------|---|
| MEDIUM | Mã lệnh chiếm nhiều hơn một đoạn. Dữ liệu trong một đoạn. |
| COMPACT | Mã lệnh trong một đoạn. Dữ liệu chiếm nhiều hơn một đoạn. |
| LARGE | Mã lệnh chiếm nhiều hơn một đoạn. Dữ liệu chiếm nhiều hơn một đoạn. Không có mảng nào lớn hơn 64 Kbyte. |
| HUGE | Mã lệnh chiếm nhiều hơn một đoạn. Dữ liệu chiếm nhiều hơn một đoạn. Các mảng có thể lớn hơn 64 Kbyte. |

4.7.2 Đoạn dữ liệu (Data segment).

Đoạn dữ liệu của một chương trình chứa tất cả các định nghĩa biến. Cũng như vậy các định nghĩa hằng cũng thường được tạo ra ở đây nhưng chúng cũng có thể được đưa vào một chỗ khác trong chương trình bởi vì không có ô nhớ nào liên quan đến nó. Để khai báo một đoạn dữ liệu chúng ta sử dụng dẫn hướng biên dịch **.DATA**, theo sau là các khai báo biến hay hằng. Ví dụ:

```
.DATA
WORD1 DW 2
WORD2 DW 5
MSG DB 'THIS IS A MESSAGE'
MASK EQU 10010010B
```

4.7.3 Đoạn ngăn xếp (Stack segment).

Mục đích của khai báo đoạn ngăn xếp là tạo ra một khối bộ nhớ (vùng ngăn xếp) để chứa ngăn xếp. Vùng ngăn xếp có thể đủ lớn để chứa ngăn xếp với kích thước lớn nhất của nó. Cú pháp khai báo như sau:

```
.STACK kích_thước
```

Trong đó kích thước là một số tuỳ ý, xác định kích thước của vùng ngăn xếp tính theo byte. Ví dụ:

```
.STACK 100h
```

sẽ tạo ra 100h byte cho vùng ngăn xếp (kích thước hợp lý cho hầu hết các chương trình ứng dụng). Nếu kích_thước bị bỏ qua, 1 Kbyte sẽ được thiết lập cho vùng ngăn xếp.

4.7.4 Đoạn mã (Code segment).

Đoạn mã chứa các lệnh của chương trình. Cú pháp khai báo là:

```
.CODE tên
```

Trong đó tên là một tên đoạn tùy ý (không cần thiết phải có tên trong một chương trình dùng kiểu bộ nhớ .SMALL bởi vì như vậy trình biên dịch sẽ phát sinh một lỗi).

Bên trong đoạn mã, các lệnh được tổ chức như các thủ tục. Một định nghĩa thủ tục đơn giản nhất là:

| | |
|-------------------|------|
| Tên_thủ_tục | PROC |
| ;thân của thủ tục | |
| Tên_thủ_tục | ENDP |

ở đây Tên_thủ_tục là tên của thủ tục; PROC và ENDP là các toán tử giả đánh dấu bắt đầu và kết thúc thủ tục.

Sau đây là một ví dụ định nghĩa đoạn mã:

```
.CODE  
MAIN PROC  
;các lệnh của chương trình chính  
MAIN ENDP  
;các thủ tục khác
```

4.7.5 Tổng hợp lại.

Giờ đây sau khi đã nghiên cứu tất cả các đoạn chương trình, chúng ta có thể chỉ ra khuôn mẫu chung cho một chương trình dùng kiểu bộ nhớ .SMALL. Chỉ với vài thay đổi nhỏ, mẫu này có thể dùng cho tất cả các chương trình ứng dụng:

```
.MODEL SMALL  
.STACK 100H  
.DATA  
;các định nghĩa số liệu ở đây
```

```

.CODE
MAIN    PROC
; các lệnh ở đây
MAIN    ENDP
; các thủ tục khác ở đây
END     MAIN

```

Dòng cuối cùng của chương trình phải là dẫn hướng biên dịch END, theo sau là tên của chương trình chính.

4.8 Các lệnh vào ra.

Trong chương 1 bạn đã biết rằng CPU liên lạc với các thiết bị ngoại vi thông qua các thanh ghi vào/ra hay còn được gọi là các cổng vào/ra. Có hai lệnh có thể truy nhập trực tiếp các cổng đó là IN và OUT. Các lệnh này được sử dụng khi yêu cầu tốc độ cao, ví dụ như trong các chương trình trò chơi. Tuy nhiên hầu hết các chương trình ứng dụng không dùng các lệnh IN và OUT bởi vì thứ nhất là các địa chỉ cổng thay đổi giữa các loại máy tính và sau nữa lập trình với các chương trình phục vụ được cung cấp bởi các nhà sản xuất dễ hơn nhiều.

Có hai loại chương trình phục vụ vào/ra: Các chương trình của DOS và BIOS (Basic Input Output System). Các chương trình BIOS được chứa trong ROM và tác động trực tiếp tới các cổng vào/ra. Trong chương 12, chúng ta sẽ dùng chúng để thực hiện các thao tác cơ bản với màn hình như di chuyển con trỏ hay cuộn màn hình. Các chương trình của DOS có thể thực hiện các công việc phức tạp hơn, ví dụ như in một chuỗi ký tự. Thực ra chúng sử dụng các chương trình của BIOS để thực hiện các thao tác vào/ra trực tiếp.

Lệnh INT.

Lệnh INT được dùng để gọi các chương trình ngắt của DOS và BIOS. Nó có dạng sau:

INT số_hiệu_ngắt

ở đây, số_hiệu_ngắt là một con số xác định một chương trình. Ví dụ: INT 16, sẽ gọi các phục vụ bàn phím của BIOS. Chương 15 sẽ trình bày về lệnh INT một cách chi tiết hơn. Sau đây chúng ta sẽ sử dụng một chương trình đặc biệt của DOS, phục vụ ngắt 21h.

4.8.1 Ngắt 21h

Ngắt 21h được dùng để gọi rất nhiều hàm của DOS (xem phụ lục C). Mỗi hàm được gọi bằng cách đặt số hàm vào trong thanh ghi AH và gọi INT 21h. Chúng ta hãy xem xét các hàm sau đây:

| Số hiệu hàm | Chương trình |
|-------------|---------------------------|
| 1 | Vào một phím |
| 2 | Đưa một ký tự ra màn hình |
| 9 | Đưa ra một chuỗi ký tự |

Các hàm của ngắt 21h nhận dữ liệu trong các thanh ghi nào đó và trả về kết quả trong các thanh ghi khác. Các thanh ghi này sẽ được liệt kê khi chúng tôi mô tả mỗi hàm.

Hàm 1:

Vào một phím.

Vào: AH=1

Ra: AL= Mã ASCII nếu một phím ký tự được ấn.

= 0 Nếu một phím điều khiển hay chức năng được nhấn

Để gọi phục vụ này, bạn hãy thực hiện các lệnh sau:

| | | |
|-----|-------|---------------------|
| MOV | AH, 1 | ; hàm vào một phím |
| INT | 21h | ; mã ASCII trong AL |

Bộ vi xử lý sẽ đợi người sử dụng ấn một phím nếu cần thiết. Nếu một phím ký tự được ấn, AL sẽ nhận mã ASCII và ký tự được hiện lên trên màn hình. Nếu một phím khác được ấn, chẳng hạn phím mũi tên, F1-F10..., thì AL sẽ chứa 0. Trong các lệnh tiếp theo INT 21h có thể kiểm tra AL và thực hiện tác vụ thích hợp.

Bởi vì hàm 1 của ngắt 21 không đưa ra thông báo để người sử dụng vào một phím nên bạn sẽ không biết được là máy tính đang đợi nhập số liệu hay đang làm các công việc khác. Hàm tiếp theo có thể được dùng để đưa ra các thông báo nhập số liệu:

Hàm 2:

Hiển thị một ký tự hay thi hành một chức năng điều khiển.

Vào: AH=2

DL= mã ASCII của ký tự hiển thị hay ký tự điều khiển.

Ra: AL= mã ASCII của ký tự hiển thị hay ký tự điều khiển.

Để dùng hàm này hiển thị một ký tự, ta đặt mã ASCII của nó trong DL. Ví dụ các lệnh sau đây sẽ làm xuất hiện dấu chấm hỏi trên màn hình:

```
MOV AH, 2  
MOV DL, '?'  
INT 21h
```

Sau khi ký tự được hiển thị, con trỏ màn hình dịch sang vị trí tiếp theo của dòng (nếu ở cuối dòng, con trỏ màn hình sẽ định chuyển sang đầu dòng tiếp theo).

Hàm 2 cũng có thể được dùng để thực hiện một chức năng điều khiển. Nếu như DL chứa mã ASCII của ký tự điều khiển, hàm này sẽ thi hành chức năng điều khiển đó. Các ký tự điều khiển quan trọng được chỉ ra sau đây:

| Mã ASCII(Hex) | Ký hiệu | Chức năng |
|---------------|---------|----------------------------|
| 7 | BEL | phát tiếng bip (beep). |
| 8 | BS | lùi lại một vị trí. |
| 9 | HT | tab. |
| A | LF | xuống dòng. |
| D | CR | xuống dòng và về đầu dòng. |

Khi thực hiện, AL nhận mã ASCII của ký tự điều khiển.

4.9 Chương trình đầu tiên.

Chương trình đầu tiên của chúng ta sẽ đọc một ký tự từ bàn phím và hiển thị nó ở đầu dòng tiếp theo.

Chúng ta bắt đầu bằng việc hiển thị một dấu chấm hỏi:

```
MOV AH, 2 ; hàm hiển thị ký tự  
MOV DL, '?' ; ký tự là "?"
```

INT 21h ; hiển thị ký tự

Lệnh thứ hai chuyển 3Fh (mã ASCII của '?') vào thanh ghi DL.

Tiếp theo ta hãy đọc một ký tự:

MOV AH, 1 ; hàm đọc một ký tự
INT 21h ; ký tự trong AL

Bây giờ chúng cần hiển thị ký tự ở dòng tiếp theo. Trước khi thực hiện điều này, ký tự phải được cất vào một thanh ghi khác (chúng tôi sẽ giải thích điều này trong chốc lát).

MOV BL, AL ; cất ký tự trong BL

Để dịch chuyển con trỏ màn hình đến vị trí đầu dòng tiếp theo chúng ta phải thực hiện các tác vụ xuống dòng và về đầu dòng. Chúng ta có thể thực hiện các hàm này bằng cách đưa mã ASCII của chúng vào DL và gọi ngắt 21h.

MOV AH, 2 ; hàm hiển thị ký tự
MOV DL, 0Dh ; về đầu dòng
INT 21h ; thực hiện về đầu dòng
MOV DL, 0Ah ; xuống dòng
INT 21h ; thực hiện xuống dòng

Lý do mà chúng ta phải đưa ký tự từ AL vào BL là hàm 2 của ngắt 21h làm thay đổi AL.

Cuối cùng chúng ta đã sẵn sàng hiển thị ký tự:

MOV DL, BL ; lấy ký tự
INT 21h ; và hiển thị nó

Sau đây là chương trình đầy đủ:

Chương trình PGM4_1.ASM

```
TITLE PGM4_1: SAMPLE PROGRAM
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
```

```

;hiển thị lời nhắc
    MOV     AH,2          ; hàm hiển thị ký tự
    MOV     DL,'?'        ; ký tự là "?"
    INT     21h            ; hiển thị ký tự

;vào một ký tự
    MOV     AH,1          ; hàm đọc một ký tự
    INT     21h            ; ký tự trong AL
    MOV     BL,AL          ; cất ký tự trong BL

;xuống dòng mới:
    MOV     AH,2          ; hàm hiển thị ký tự
    MOV     DL,0Dh          ; về đầu dòng
    INT     21h            ; thực hiện về đầu dòng
    MOV     DL,0Ah          ; xuống dòng
    INT     21h            ; thực hiện xuống dòng

;hiển thị ký tự:
    MOV     DL,BL          ; lấy ký tự
    INT     21h            ; và hiển thị nó

;trở về DOS
    MOV     AH,4CH          ; hàm thoát về DOS
    INT     21H             ; thoát về DOS
MAIN   ENDP
    END     MAIN

```

Do không dùng các biến nên ta bỏ qua đoạn dữ liệu.

Kết thúc một chương trình.

Hai dòng cuối cùng của chương trình MAIN cần có đôi lời giải thích. Khi một chương trình kết thúc, nó phải trả điều khiển về cho DOS. Chúng ta có thể thực hiện điều này bằng cách gọi hàm 4Ch của ngắt 21h.

4.10 Tạo lập và chạy một chương trình.

Bây giờ chúng ta đã sẵn sàng để xem xét các bước tạo lập và chạy một chương trình. Chương trình nêu ở trên như là một ví dụ để khảo sát. Có 4 bước cụ thể (hình 4.8):

1. Dùng một chương trình soạn thảo văn bản tạo ra một file chương trình nguồn (source program file).

2. Dùng một chương trình biên dịch tạo ra file đối tượng ngôn ngữ máy (object file).
3. Dùng chương trình LINK (sẽ trình bày chi tiết sau) liên kết một hay nhiều file đối tượng tạo ra các file chương trình (run file).
4. Cho chạy file chương trình.

Trong chỉ dẫn này, các file cần thiết (chương trình biên dịch và chương trình liên kết) có trong ổ đĩa C, các đĩa của người lập trình trong ổ đĩa A. Chúng ta đặt ổ đĩa A là ổ đĩa mặc định để các file tạo ra sẽ được chứa trong đĩa của người lập trình.

Bước 1. Tạo lập file chương trình nguồn.

Chúng ta đã sử dụng một chương trình soạn thảo văn bản tạo chương trình ở trên với tên là PGM4_1.ASM. Phần mở rộng .ASM được quy ước dùng để định nghĩa một file nguồn của Hợp ngữ.

Bước 2. Hợp dịch chương trình.

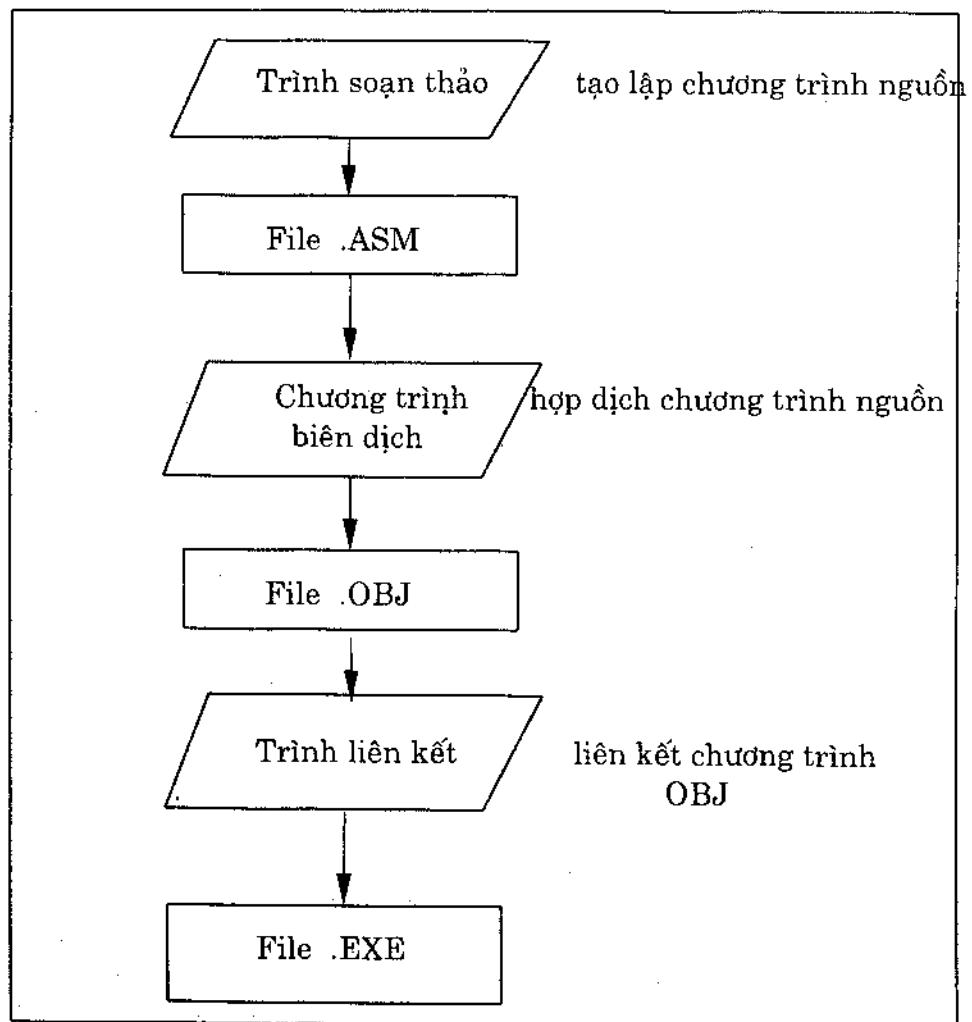
Chúng ta sử dụng MASM (Microsoft Macro Assembler) để định file nguồn PGM4_1.ASM sang file đối tượng ngôn ngữ máy PGM4_1.OBJ. Lệnh đơn giản nhất (câu trả lời của người sử dụng xuất hiện trong vùng đậm) là:

```
A>C:MASM PGM4_1;
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All right
reserved.

      50060 + 418673 Byte symbol space free
          0 Warning Errors
          0 Severe Errors
```

Sau khi in ra các thông tin bản quyền, MASM kiểm tra các lỗi cú pháp của file nguồn. Nếu tìm thấy một lỗi nào đó nó sẽ hiển thị số dòng của mỗi lỗi và một hướng dẫn ngắn gọn. Trong trường hợp này do không có lỗi nào, MASM sẽ dịch mã Hợp ngữ thành file đối tượng ngôn ngữ máy với tên PGM4_1.OBJ.

Hình 4.8 Các bước lập trình.



Dấu chấm phẩy theo sau câu lệnh có nghĩa là chúng ta không muốn phát sinh thêm các file khác nữa. Ta hãy bỏ nó đi và xem cái gì xảy ra:

```
A>C:MASM PGM4_1
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved.
Object filename [ PGM4_1.OBJ ] :
Source listing [ NUL.LST ] :      PGM4_1
Cross_reference [ NUL.CRF ] :      PGM4_1
```

```
50060 + 418673 Byte symbol space free
0 Warning Errors
0 Severe Errors
```

Lần này MASM in ra các tên file có thể được tạo nên và đợi chúng ta đưa vào tên mới. Các tên mặc định được viết trong dấu ngoặc vuông. Muốn chấp nhận một tên chỉ cần nhấn phím Return. Tên mặc định NUL có nghĩa là sẽ không tạo tên file đó nếu ta không đưa vào một tên, do vậy ở đây ta trả lời với tên PGM4_1.

File nguồn listing.

File nguồn listing (file .LST) là một file văn bản các dòng được đánh số hiển thị mã. Hợp ngữ bên cạnh mã máy tương ứng đồng thời đưa ra các thông tin khác về chương trình. Điều này đặc biệt có ích cho các mục đích gỡ rối bởi vì các lỗi của MASM đưa ra kèm theo số dòng.

File tham khảo ngang (Cross_reference file).

File tham khảo ngang (file .CRF) là một bảng liệt kê các tên xuất hiện trong chương trình và số thứ tự các dòng mà nó có mặt. Nó cần thiết khi xác định các biến và các nhãn trong một chương trình lớn.

Ví dụ về các file .LST và .CRF được chỉ ra trong phụ lục D cùng với các phần chọn khác của MASM.

Bước 3. Hợp dịch chương trình.

File .OBJ được tạo lập ở bước 2 là một file ngôn ngữ máy nhưng nó không thể thực hiện được bởi lẽ khuôn mẫu của nó chưa thích hợp với một File chương trình:

1. Nó không biết được nơi mà chương trình sẽ được nạp vào trong bộ nhớ để thi hành, và địa chỉ mã máy có thể chưa được điền vào.
2. Một vài tên dùng trong chương trình có thể chưa được định nghĩa. Ví dụ: trong một chương trình lớn có thể cần phải tạo ra vài file, và một thủ tục trong một file này có thể tham trỏ tới một tên được định nghĩa trong file khác.

Chương trình LINK sẽ gồm một hay nhiều file đối tượng, điền vào mọi địa chỉ còn thiếu và kết hợp các file đối tượng thành một file khả thi duy nhất (file .EXE). File này có thể được nạp vào trong bộ nhớ và chạy luôn.

Để liên kết chương trình bạn hãy đánh vào:

A>C:LINK PGM4_1;

Giống như trên, nếu không có dấu chấm phẩy, chương trình liên kết sẽ đưa ra thông báo để bạn vào các tên của các file phát sinh (xem phụ lục D).

Bước 4. Chạy một chương trình.

Để chạy một chương trình bạn chỉ cần đánh vào tên File chương trình có hoặc không có phần mở rộng .EXE.

A> PGM4_1

?A

A

Chương trình in ra dấu "?" và đợi chúng ra đưa vào một ký tự. Ta đánh vào A và chương trình lặp lại nó ở dòng tiếp theo.

4.11 Hiển thị một chuỗi.

Trong chương trình đầu tiên, ta dùng các hàm 1 và 2 của ngắt 21h để đọc và hiển thị một ký tự. Sau đây là một hàm khác của ngắt 21h được dùng để hiển thị một chuỗi các ký tự:

**Ngắt 21h, Hàm 9:
Hiển thị một chuỗi.**

Vào: DX=địa chỉ tương đối (offset) của chuỗi.
 Chuỗi phải kết thúc bằng ký tự '\$'.

Ký tự '\$' đánh dấu kết thúc chuỗi và không được hiển thị. Nếu như chuỗi chứa mã ASCII của ký tự điều khiển thì các chức năng điều khiển sẽ được thi hành.

Để làm ví dụ cho hàm này, chúng ta sẽ viết một chương trình in chuỗi "HELLO!" ra màn hình. Lời chào này được định nghĩa trong đoạn dữ liệu :

MSG DB "HELLO! \$"

Lệnh LEA.

Hàm `9` của ngắt `21h` yêu cầu địa chỉ tương đối của chuỗi ký tự chứa trong DX. Để thực hiện điều này chúng ta sẽ dùng một lệnh mới:

LEA dịch, nguồn

Trong đó đích là một thanh ghi công dụng chung, nguồn là một ô nhớ. LEA có nghĩa là “Load Effective Address” (nạp địa chỉ thực). Nó sẽ lấy ra và chép địa chỉ tương đối của nguồn sang đích. Ví dụ:

LEA DX, MSG

sẽ nhập địa chỉ tương đối của biến MSG vào DX.

Bởi vì chương trình thứ hai của chúng ta có chứa đoạn dữ liệu nên nó sẽ được bắt đầu với lệnh khởi tạo DS. Phần tiếp sau đây sẽ giải thích tại sao các lệnh đó là cần thiết.

Đoạn mở đầu chương trình.

Khi một chương trình được nạp vào bộ nhớ, DOS sẽ dành cho nó 256 byte đoạn mở đầu chương trình (PSP_Program Segment Prefix). PSP chứa các thông tin về chương trình vì thế chương trình có thể truy nhập vùng này. DOS sẽ đưa địa chỉ của PSP vào cả trong DS lẫn ES trước khi thi hành chương trình. Kết quả là DS không chứa địa chỉ của đoạn dữ liệu. Để chạy đúng, một chương trình có chứa đoạn dữ liệu sẽ được bắt đầu với hai lệnh sau:

MOV AX, @DATA
MOV DS, AX

@DATA là tên của đoạn dữ liệu được định nghĩa bởi .DATA. Chương trình biên dịch sẽ dịch @DATA thành địa chỉ. Ta phải dùng hai lệnh vì là một số (địa chỉ dữ liệu) không thể chuyển trực tiếp vào một thanh ghi đoạn. Với thanh ghi DS đã được khởi tạo, chúng ta có thể in ra lời chào "HELLO!" bằng cách đưa địa chỉ của nó vào thanh ghi DX rồi gọi ngắt 21h:

| | | |
|-----|---------|----------------------|
| LEA | DX, MSG | ; lấy thông báo |
| MOV | AH, 9 | ; hàm hiển thị chuỗi |
| INT | 21h | ; hiển thị chuỗi |

Sau đây là chương trình đầy đủ:

Chương trình PGM4_2.ASM

```
TITLE      PGM4_2: Chương trình in chuỗi ký tự.  
.MODEL      SMALL  
.STACK      100H  
.DATA  
MSG       DB      "HELLO!$"  
.CODE  
MAIN      PROC  
;khởi tạo DS  
          MOV     AX, @DATA  
          MOV     DS, AX  
;hiển thị thông báo  
          LEA     DX, MSG           ; lấy thông báo  
          MOV     AH, 9             ; hàm hiển thị chuỗi  
          INT     21h              ; hiển thị chuỗi  
;trở về DOS  
          MOV     AH, 4CH  
          INT     21H  
MAIN      ENDP  
END      MAIN
```

Và đây là kết quả khi chạy chương trình:

```
A> PGM4_2  
HELLO!
```

4.12 Một chương trình đổi chữ thường thành chữ hoa.

Bây giờ chúng ta sẽ tổng kết các kiến thức trình bày trong chương này vào một chương trình duy nhất. Chương trình này sẽ bắt đầu bằng việc nhắc người sử dụng đưa vào một chữ thường, trên dòng tiếp theo nó sẽ đưa ra một thông báo khác với chữ đã được đổi sang dạng in hoa. Ví dụ:

```
ENTER A LOWER CASE LETTER : a  
IN UPPER CASE IT IS : A
```

Chúng ta dùng các tên CR và LF để định nghĩa các hàng số 0DH và 0AH.

```
CR      EQU      0DH  
LF      EQU      0AH
```

Thông báo và ký tự nhập vào có thể được chứa trong đoạn dữ liệu như sau:

```
MSG1    DB        'ENTER A LOWER CASE LETTER : $ '  
MSG2    DB        CR,LF,' IN UPPER CASE IT IS :'  
CHAR    DB        ?, '$ '
```

Khi định nghĩa MSG2 và CHAR chúng ta sử dụng một mảnh khoá hữu hiệu: Bởi vì chương trình phải hiển thị thông báo thứ hai và một chữ cái (sau khi đã đổi thành chữ hoa) trên dòng tiếp theo, MSG2 bắt đầu với mã ASCII của ký tự xuống dòng và trở về đầu dòng. Khi MSG2 được hiển thị bằng hàm 9 của ngắt 21h các chức năng điều khiển này sẽ được thi hành và thông báo được hiển thị ở dòng tiếp theo. Do MSG2 không kết thúc bằng ký tự '\$', ngắt 21h sẽ hiển thị cả ký tự chứa trong CHAR.

Chương trình của chúng ta bắt đầu với việc hiển thị thông báo thứ nhất và đọc ký tự:

```
LEA     DX,MSG1          ; lấy thông báo đầu tiên  
MOV     AH,9              ; hàm hiển thị chuỗi  
INT     21h               ; hiển thị thông báo đầu tiên  
MOV     AH,1              ; hàm đọc một ký tự  
INT     21h               ; đọc một chữ thường vào AL
```

Sau khi đã đọc vào chữ thường, chương trình phải đổi nó sang dạng chữ hoa. Trong bảng mã ASCII, các chữ thường bắt đầu tại 61h và các chữ hoa bắt đầu tại 41h, vì vậy để đổi kiểu chữ chỉ cần lấy nội dung AL trừ đi 20h:

```
SUB     AL,20H           ; đổi thành chữ hoa  
MOV     CHAR,AL          ; và lưu trữ nó
```

Bây giờ chương trình sẽ hiển thị thông báo thứ hai và chữ ở dạng in hoa:

```
LEA     DX,MSG2          ; lấy thông báo thứ hai  
MOV     AH,9              ; hàm hiển thị chuỗi  
INT     21h               ; hiển thị thông báo thứ hai  
                                ; và chữ hoa
```

Sau đây là chương trình đầy đủ:

Chương trình PGM4_3.ASM

```
TITLE PGM4_3: SAMPLE PROGRAM
.MODEL SMALL
.STACK 100H
.DATA
CR EQU 0DH
LF EQU 0AH
MSG1 DB 'ENTER A LOWER CASE LETTER : $ '
MSG2 DB CR,LF,' IN UPPER CASE IT IS :'
CHAR DB ?, '$ '
.CODE
MAIN PROC
;khoi tao DS
    MOV AX, @DATA
    MOV DS, AX
;in dong nhanh nguoi su dung
    LEA DX, MSG1 ;lay thong bao dau tien
    MOV AH, 9 ;ham hien thi chuoi
    INT 21h ;hien thi thong bao dau tien
;vao mot ky tu va doi thanh chua hoa
    MOV AH, 1 ;ham doc mot ky tu
    INT 21h ;doc mot chua thuong vao AL
    SUB AL, 20H ;doi thanh chua hoa
    MOV CHAR, AL ;va luu tru no
;hien thi tren dong tiep theo
    LEA DX, MSG2 ;lay thong bao thu hai
    MOV AH, 9 ;ham hien thi chuoi
    INT 21h ;hien thi thong bao thu
;hai va chua hoa
;tro ve DOS
    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN
```

TỔNG KẾT:

- ◆ Các chương trình bằng Hợp ngữ được tạo nên từ các dòng lệnh. Mỗi dòng lệnh có thể là một lệnh sẽ được máy tính thi hành hay một dẫn hướng cho chương trình biên dịch.
- ◆ Các dòng lệnh bao gồm các trường tên, toán tử, (các) toán hạng và trường lối bình.
- ◆ Một tên có chứa đến 31 ký tự. Các ký tự có thể là các chữ cái, chữ số hay các ký hiệu đặc biệt nào đó.
- ◆ Các số có thể được viết ở dạng nhị phân, thập phân hay hex.
- ◆ Các ký tự và chuỗi ký tự phải được bao bọc bởi dấu ngoặc đơn hay ngoặc kép.
- ◆ Các dẫn hướng DB và DW được dùng để định nghĩa các biến byte và các biến word. EQU sử dụng khi muốn gán tên cho hằng.
- ◆ Nói chung một chương trình chứa một đoạn mã, một đoạn dữ liệu và một đoạn ngắn xếp.
- ◆ MOV và XCHG được dùng để chuyển số liệu. Có một vài hạn chế khi sử dụng các lệnh này, ví dụ chúng không thể thao tác trực tiếp giữa các ô nhớ.
- ◆ ADD, SUB, INC, DEC và NEG là các lệnh số học cơ bản.
- ◆ Có hai cách xuất và nhập dữ liệu đối với IBM PC : liên lạc trực tiếp với các thiết bị ngoại vi thông qua cổng và sử dụng các phục vụ ngắn của DOS và BIOS.
- ◆ Phương pháp trực tiếp khó lập trình và phụ thuộc vào các vi mạch phần cứng nhất định.
- ◆ Xuất và nhập các ký tự hay các chuỗi có thể thực hiện nhờ phục vụ ngắn 21h của DOS.
- ◆ Hàm 1 của ngắn 21h đọc một ký tự từ bàn phím vào thanh ghi AL.
- ◆ Hàm 2 của ngắn 21h hiển thị ký tự có mã ASCII chứa trong thanh ghi DL. Nếu DL chứa mã ASCII của một ký tự điều khiển thì chức năng điều khiển sẽ được thi hành.

- ♦ Hàm 9 của ngắt 21h hiển thị chuỗi có địa chỉ tương đối trong DX. Chuỗi phải được kết thúc bằng ký tự '\$'.

Thuật ngữ tiếng Anh

| | |
|-------------------------------------|---|
| Array | Một chuỗi các byte hay từ nhớ. |
| assembler directive | Dẫn hướng biên dịch: Dẫn hướng cho chương trình biên dịch thực hiện một vài nhiệm vụ đặc biệt. |
| CODE segment | Đoạn mã. Vùng chương trình chứa các lệnh. |
| .CRF file | File tạo nên bởi trình biên dịch trong đó liệt kê các tên xuất hiện trong chương trình và số của các dòng mà chúng xuất hiện. |
| DATA segment | Đoạn dữ liệu. Vùng chương trình chứa các biến. |
| destination operand | Toán hạng đích. Toán hạng đầu tiên của lệnh. Nó sẽ chứa kết quả trả về. |
| .EXE file | Giống như File chương trình. |
| instruction | Lệnh hay chỉ thị. Một dòng lệnh được chương trình biên dịch dịch ra mã máy. |
| .LST file | File có các dòng được đánh số tạo bởi trình biên dịch trong đó chứa mã Hợp ngữ, mã máy và các thông tin khác về chương trình. |
| memory MODEL | Kiểu bộ nhớ. Tổ chức của một chương trình cho biết tổng số các mã và dữ liệu. |
| object file | File đối tượng. File ngôn ngữ máy được chương trình biên dịch tạo ra từ file chương trình nguồn. |
| PSP (program segment prefix) | Đoạn mở đầu chương trình :Vùng 256 byte đứng trước chương trình trong bộ nhớ chứa các thông tin về chương trình. |
| pseudo_op | Toán tử giả. Dẫn hướng biên dịch. |
| run file | File chương trình. File ngôn ngữ máy khả thi tạo nên bởi chương trình LINK. |
| source operand | Toán hạng nguồn. Toán hạng thứ hai, thường không thay đổi khi thực hiện lệnh. |

| | |
|----------------------------|--|
| source program file | Tệp chương trình nguồn. Tệp văn bản chương trình tạo ra bằng một chương trình soạn thảo văn bản. |
| STACK segment | Đoạn ngăn xếp. Phần của chương trình chứa ngăn xếp hiện hành. |
| variable | Biến. Tên tượng trưng cho một ô nhớ chứa dữ liệu. |

Các lệnh mới

| | | |
|------------|------------|-------------|
| ADD | INT | NEG |
| DEC | LEA | SUB |
| INC | MOV | XCHG |

Các toán tử giả

mới

| | | |
|--------------|---------------|------------|
| .CODE | .MODEL | EQU |
| .DATA | .STACK | |

Bài tập:

1. Tên nào trong các tên sau đây là hợp lệ trong Hợp ngữ cho IBM PC ?
 - a. TWO_WORD
 - b. ?1
 - c. Two_word
 - d. .@?
 - e. \$145
 - f. LET'S_GO
 - g. T=.
2. Số nào trong các số sau đây là hợp lệ. Nếu chúng hợp lệ, hãy chỉ rõ chúng là số nhị phân, thập phân hay số hex.
 - a. 246
 - b. 246h
 - c. 1001
 - d. 1,101
 - e. 2A3h
 - f. FFF Eh
 - g. 0ah
 - h. bh
 - i. 1110b
3. Hãy nêu toán tử giả định nghĩa dữ liệu cho các biến và hằng sau đây nếu như chúng hợp lệ:
 - a. Một biến kiểu word được khởi tạo với giá trị 52.
 - b. Một biến kiểu word WORD1, không được khởi tạo.
 - c. Một biến kiểu byte B được khởi tạo với giá trị 52h.
 - d. Một biến kiểu byte C1 không được khởi tạo.
 - e. Một biến kiểu word WORD2 được khởi tạo với giá trị 65536.
 - f. Một mảng kiểu word được khởi tạo với 5 giá trị đầu nguyên (từ 1 đến 5).
 - g. Một hằng BEL bằng 07h.
 - h. Một hằng MSG bằng 'THIS IS A MESSAGE \$'.
4. Giả thiết rằng các số liệu sau đây được nạp vào bộ nhớ bắt đầu tại vị trí offset 0000h:

| | | |
|---|----|---------|
| A | DB | 7 |
| B | DW | 1ABCh |
| C | DB | 'HELLO' |

- a. Hãy cho biết các địa chỉ offset của các biến A, B và C.
 - b. Hãy cho biết nội dung của byte tại offset 0002h dưới dạng số hex.
 - c. Hãy cho biết nội dung của byte tại offset 0004h dưới dạng số hex.
 - d. Hãy cho biết địa chỉ offset của ký tự 'O' trong 'HELLO'.
5. Hãy cho biết mỗi lệnh dưới đây là hợp lệ hay không hợp lệ, trong đó W1 và W2 là các biến WORD; B1, B2 là các biến BYTE.
- a. MOV DS, AX
 - b. MOV DS, 100h
 - c. MOV DS, ES
 - d. MOV W1, DS
 - e. XCHG W1, W2
 - f. SUB 5, B1
 - g. ADD B1, B2
 - h. ADD AL, 256
 - i. MOV W1, B1
6. Chỉ dùng các lệnh MOV, ADD, SUB, INC, DEC và NEG hãy dịch các dòng lệnh gán của ngôn ngữ bậc cao sau đây sang Hợp ngữ với A, B, C là các biến kiểu word.
- a. $A = B - A$
 - b. $A = -(A + 1)$
 - c. $C = A + B$
 - d. $B = 3 * B + 7$
 - e. $B = B - A - 1$
7. Hãy viết các lệnh (không phải các chương trình đầy đủ) thực hiện các công việc sau đây:
- a. Đọc một ký tự và hiển thị nó ở vị trí tiếp theo trên cùng một dòng.
 - b. Đọc một chữ hoa (bỏ qua việc kiểm tra lỗi) và hiển thị nó ở vị trí tiếp theo trên cùng một dòng dưới dạng chữ thường.
8. Viết một chương trình thực hiện các công việc sau đây:
- a. Hiển thị dấu hỏi chấm (?)
 - b. Đọc hai chữ số thập phân có tổng nhỏ hơn 10.
 - c. Hiển thị các số đó với tổng của chúng với dòng thông báo tương ứng.

Ví dụ:

?27

Tổng của 2 và 7 là 9.

9. Hãy viết một chương trình thực hiện các công việc sau đây:

- a. Đưa ra thông báo cho người sử dụng.
- b. Đọc 3 chữ cái đầu của họ, tên đệm, tên của một người.
- c. Hiển thị chúng từ trên xuống trên lề trái.

Ví dụ:

Bạn hãy vào 3 chữ cái đầu: NTB

N

T

B

10. Viết một chương trình đọc một chữ số hex trong khoảng (A-F) rồi hiển thị nó trên dòng tiếp theo ở dạng nhị phân.

Ví dụ:

Bạn vào một chữ số hex: C

Dạng thập phân của nó là: 12

11. Viết một chương trình hiển thị một bảng 10x10 điền đầy dấu sao.

Gợi ý: Khai báo một chuỗi xác định hộp trong đoạn dữ liệu rồi hiển thị nó bằng hàm 9 của ngắt 21h.

12. Viết một chương trình:

- a. Hiển thị dấu "?".
- b. Đọc 3 chữ cái.
- c. Hiển thị chúng trong một bảng 11x11 được điền đầy các dấu sao.
- d. Phát tiếng kêu bip của máy tính.

Chương 5

TRẠNG THÁI CỦA BỘ XỬ LÝ VÀ THANH GHI CỜ

Tổng quan

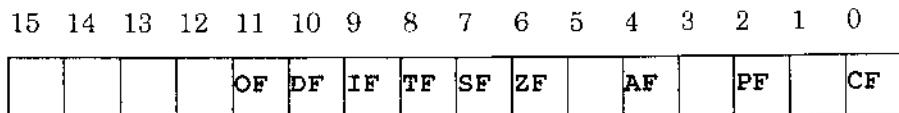
Một điểm khác biệt quan trọng giữa máy tính và các loại máy khác đó là máy tính có khả năng quyết định. Các mạch trong máy tính có khả năng thực hiện những quyết định đơn giản dựa trên trạng thái hiện tại của bộ xử lý. Đối với bộ vi xử lý 8086, trạng thái của bộ vi xử lý được thể hiện trong 9 bit riêng biệt gọi là các cờ. Mọi quyết định của bộ vi xử lý đều dựa trên giá trị của các cờ này.

Các cờ được đặt trong thanh ghi cờ và chúng được phân thành 2 loại cờ trạng thái và cờ điều khiển. Cờ trạng thái phản ánh kết quả của các phép tính. Trong chương này chúng ta sẽ thấy chúng bị ảnh hưởng ra sao bởi các chỉ thị máy. Trong chương 6 chúng ta sẽ nghiên cứu việc sử dụng chúng để xây dựng các chương trình có nhiều lệnh rẽ nhánh và vòng lặp. Cờ điều khiển được sử dụng để cho phép hoặc không cho phép một thao tác nào đó của bộ vi xử lý, chúng sẽ được mô tả ở các chương cuối.

Trong phần 5.1 chúng tôi sẽ giới thiệu chương trình DEBUG của DOS, chúng ta sẽ thấy cách sử dụng DEBUG để thực hiện từng lệnh trong chương trình của người sử dụng, hiển thị các thanh ghi, các cờ và các ô nhớ.

5.1 Thanh ghi cờ

Hình 5.1 cho thấy cấu trúc của thanh ghi cờ. Các cờ trạng thái nằm ở các bit 0, 2, 4, 6, 7 và 11 còn các cờ điều khiển nằm ở các bit 8, 9 và 10. Các bit khác không có ý nghĩa. Chú ý rằng không cần nhớ cờ nào nằm ở bit nào. Bảng 5.1 trình bày tên các cờ và ký hiệu của chúng. Trong chương này chúng ta sẽ tập trung vào các cờ trạng thái.



Hình 5.1 Thành ghi cờ

Các cờ trạng thái

Như đã nói trên, bộ xử lý sử dụng cờ trạng thái để phản ánh kết quả của một phép tính, chẳng hạn khi lệnh SUB AX,AX được thực hiện cờ ZF sẽ được thiết lập 1 nhờ vậy nó chỉ ra rằng kết quả bằng 0 đã được tạo ra. Bây giờ chúng ta hãy xem các cờ trạng thái.

Cờ nhớ (Carry Flag- CF)

Cờ CF được thiết lập 1 khi có nhỡ từ bit msb trong phép cộng hay có vay vào bit msb trong phép trừ. Ngược lại nó bằng 0. Cờ CF cũng bị ảnh hưởng bởi các lệnh quay và dịch (xem chương 7).

Cờ chẵn lẻ (Parity Flag- PF)

Cờ PF được thiết lập 1 nếu byte thấp của kết quả có số chẵn các bit 1 (parity chẵn). Nó bằng 0 nếu byte thấp có số lẻ bit 1 (parity lẻ). Ví dụ kết quả của một phép cộng các word là FFFEh, như vậy byte thấp có 7 bit 1 do đó PF=0

Cờ nhớ phụ (Auxiliary Flag- AF)

Cờ AF được thiết lập 1 nếu có nhỡ từ bit 3 trong phép cộng hoặc có vay vào bit 3 trong phép trừ. Cờ AF được sử dụng trong các thao tác với số thập phân mã hoá nhị phân (số BCD).

Cờ Zero (Zero Flag- ZF)

Cờ ZF được thiết lập 1 khi kết quả bằng 0 và ngược lại.

Cờ dấu (Sign Flag- SF)

Cờ SF được thiết lập 1 khi bit msb của kết quả bằng 1 có nghĩa là kết quả là âm nếu bạn làm việc với số có dấu. Ngược lại SF=0 nếu bit msb của kết quả bằng 0.

Cờ tràn (Overflow Flag- OF)

Cờ OF được thiết lập 1 khi xảy ra tràn ngược lại nó bằng 0, khái niệm tràn sẽ được giải thích sau đây.

| Cờ trạng thái | | |
|---------------|------------|---------|
| Bit | Tên gọi | Ký hiệu |
| 0 | cờ nhớ | CF |
| 2 | Cờ chẵn lẻ | PF |
| 4 | Cờ nhớ phụ | AF |
| 6 | Cờ ZERO | ZF |
| 7 | Cờ dấu | SF |
| 11 | Cờ tràn | OF |

| Cờ điều khiển | | |
|---------------|---------------|---------|
| Bit | Tên gọi | Ký hiệu |
| 8 | Cờ bẫy | TF |
| 9 | Cờ ngắt | IF |
| 10 | Cờ định hướng | DF |

Bảng 5.1 Tên cờ và các ký hiệu.

5.2 Hiện tượng tràn

Hiện tượng tràn gắn liền với một sự thật là phạm vi của các số biểu diễn trong máy tính có giới hạn.

Chương 2 đã chỉ ra rằng phạm vi của các số thập phân có dấu có thể biểu diễn bằng một word 16 bit là từ -32768 đến 32767, với một byte 8 bit thì phạm vi là từ -128 đến 127.

Đối với các số không dấu thì phạm vi là từ 0 đến 65535 cho một word và từ 0 đến 255 cho một byte. Nếu kết quả của một phép tính nằm ngoài phạm vi này thì hiện tượng tràn sẽ xảy ra và kết quả nhận được bị cắt bớt sẽ không phải là kết quả đúng.

Các ví dụ về hiện tượng tràn

Sự tràn có dấu và không có dấu là các hiện tượng độc lập nhau. Khi chúng ta thực hiện một thao tác số học như cộng hay trừ, có 4 khả năng xảy ra là: (1) không tràn, (2) chỉ tràn có dấu, (3) chỉ tràn không dấu, (4) tràn có dấu và không dấu đồng thời.

Đây là một ví dụ về hiện tượng tràn không dấu nhưng không tràn có dấu:

Giả sử AX chứa FFFFh, BX chứa 0001h và lệnh ADD AX,BX được thực hiện. Kết quả dạng nhị phân như sau:

$$\begin{array}{r} 1111\ 1111\ 1111\ 1111 \\ +\ 0000\ 0000\ 0000\ 0001 \\ \hline 1\ 0000\ 0000\ 0000\ 0000 \end{array}$$

Nếu chúng ta làm việc với các số không dấu, kết quả đúng phải là $100000h = 65536$, nhưng kết quả này nằm ngoài phạm vi biểu diễn của một word nên kết quả còn lại trong thanh ghi AX là 0h, đây là một kết quả sai, như vậy hiện tượng tràn không dấu đã xảy ra. Nhưng kết quả nhận được lại đúng với các số có dấu, FFFFh = -1 khi hiểu là số có dấu, trong khi đó 0001h = 1 vậy tổng của chúng bằng 0, rõ ràng hiện tượng tràn có dấu đã không xảy ra.

Bây giờ chúng ta hãy xem một ví dụ về hiện tượng tràn có dấu nhưng lại không tràn không dấu. Giả sử AX và BX cùng chứa 7FFFh, hãy thực hiện phép cộng ADD AX,BX

Kết quả dạng nhị phân như sau

$$\begin{array}{r} 0111\ 1111\ 1111\ 1111 \\ +\ 0111\ 1111\ 1111\ 1111 \\ \hline 1111\ 1111\ 1111\ 1110 \quad =FFFEh \end{array}$$

Trong cả 2 dạng có dấu và không có dấu 7FFFh đều bằng 32767, bởi vậy trong cả 2 phép cộng không dấu và có dấu đều cho kết quả là $32767 + 32767 = 65534$. Giá trị này nằm ngoài phạm vi của số có dấu, kết quả nhận được dạng có dấu là FFFEh = -2, như vậy xảy ra hiện tượng tràn có dấu. Tuy nhiên FFFEh lại bằng 65534 ở dạng không dấu do vậy không có hiện tượng tràn không dấu.

Có 2 vấn đề được đặt ra đó là làm sao CPU chỉ ra có hiện tượng tràn và làm sao nó biết được có hiện tượng tràn xảy ra ?

CPU đã chỉ ra có hiện tượng tràn như thế nào ?

Bộ xử lý lập cờ OF = 1 để báo có hiện tượng tràn có dấu và CF = 1 để báo có hiện tượng tràn không dấu. Thông báo này dùng cho chương trình để tiến hành những hành động thích hợp, nếu như không có hành động nào được thực hiện ngay lập tức thì kết quả của lệnh sau có thể xoá cờ báo tràn.

Khi xác định có hiện tượng tràn bộ xử lý không coi kết quả như là số không dấu cũng như số có dấu, thay vào đó nó sử dụng cả 2 cách hiểu có dấu và không có dấu cho mỗi thao tác đồng thời thiết lập các cờ CF hay OF báo tràn có dấu hay không có dấu một cách tương ứng.

Trách nhiệm thuộc về người lập trình, người có quy ước về kết quả. Nếu anh ta đang làm việc với số có dấu thì chỉ có cờ OF đáng quan tâm trong khi cờ CF có thể bỏ qua, ngược lại khi làm việc với số không dấu thì cờ quan trọng là CF chứ không phải là OF.

Làm cách nào CPU biết được có hiện tượng tràn xảy ra ?

Có rất nhiều lệnh có thể gây ra hiện tượng tràn, nhưng để đơn giản ở đây chúng ta chỉ nói về phép cộng và phép trừ.

Hiện tượng tràn không dấu

Khi thực hiện phép cộng, hiện tượng tràn xảy ra khi có nhỡ từ bit msb. Có nghĩa là kết quả đúng của phép tính lớn hơn số không dấu lớn nhất có thể biểu diễn, đó là FFFFh cho một word và FFh cho một byte. Khi thực hiện phép trừ hiện tượng tràn không dấu xảy ra khi có sự vay vào bit msb, có nghĩa là kết quả đúng nhỏ hơn 0.

Hiện tượng tràn có dấu

Trong phép cộng các số cùng dấu, hiện tượng tràn có dấu xảy ra khi tổng nhận được có dấu khác với dấu của 2 số hạng. Điều này đã xảy ra trong ví dụ trên khi chúng ta cộng 2 số dương 7FFFh nhưng lại nhận được kết quả là một số âm FFFEh.

Phép trừ các số khác dấu cũng giống như phép cộng các số cùng dấu, ví dụ

$A - (-B) = A + B$ và $-A - (+B) = -A + -B$. Hiện tượng tràn xảy ra khi kết quả có dấu khác với dấu chúng ta chờ đợi (xem ví dụ 5.3 trong phần sau).

Trong phép cộng 2 số khác dấu hiện tượng tràn là không thể xảy ra vì $A + (-B)$ chính bằng $A - B$ và bởi vì A và B là 2 số đủ nhỏ để chứa trong toán hạng

dịch thì hiệu của chúng tất nhiên không thể vượt ra ngoài phạm vi của nó được. Cũng vì lý do như vậy mà phép trừ 2 số cùng dấu cũng không thể gây ra hiện tượng tràn.

Thực ra bộ xử lý sử dụng phương pháp sau để thiết lập cờ OF:

Nếu việc nhớ vào bit msb và việc nhớ ra từ nó không đồng thời, có nghĩa là có nhớ vào msb nhưng không có nhớ ra từ nó, hay có nhớ ra từ nó mà không có nhớ vào thì hiện tượng tràn có dấu xuất hiện và OF được lập 1 (xem ví dụ 5.2 trong phần tiếp theo).

5.3 Sự ảnh hưởng của các lệnh đến các cờ

Nhìn chung mỗi khi bộ xử lý thực hiện một lệnh, các cờ được thay đổi để phản ánh kết quả. Tuy nhiên có một số lệnh không ảnh hưởng tới cờ, chỉ ảnh hưởng tới một số trong chúng hay có thể làm cho chúng không xác định. Vì lệnh nhảy sẽ nghiên cứu trong chương 6 phụ thuộc vào việc lập cờ, chúng ta cần biết mỗi lệnh ảnh hưởng tới cờ như thế nào. Chúng ta hãy trở lại với 7 lệnh cơ bản đã học trong chương 4, chúng ảnh hưởng tới cờ như sau:

| Chỉ thị | Các cờ bị ảnh hưởng |
|----------|---|
| MOV/XCHG | Không cờ nào |
| ADD/SUB | Tất cả các cờ |
| INC/DEC | Tất cả các cờ trừ cờ CF |
| NEG | Tất cả các cờ (CF = 1 trừ khi kết quả là 0, OF = 1 nếu toán hạng word là 8000h hay toán hạng byte là 80h) |

Để các bạn làm quen với việc các lệnh có ảnh hưởng như thế nào đến các cờ, chúng tôi sẽ nêu ra ở đây một vài ví dụ. Trong mỗi ví dụ chúng tôi sẽ trình bày một lệnh, nội dung của các toán hạng và dự đoán kết quả cũng như việc thiết lập các cờ CF, PF, ZF và OF (chúng ta bỏ qua AF vì nó chỉ được sử dụng cho các phép tính số học với số BCD).

Ví dụ 5.1: Thực hiện phép cộng AX,BX, trong đó AX và BX đều chứa FFFFh

Trả lời :

$$\begin{array}{r}
 \text{FFFFh} \\
 + \quad \text{FFFFh} \\
 \hline
 \text{1FFEh}
 \end{array}$$

Kết quả thu được chứa trong AX là FFFEh = 1111 1111 1111 1110.

SF = 1 vì msb = 1.

PF = 0 vì có 7 bit 1 (lẻ các bit 1) trong byte thấp của kết quả.

ZF = 0 vì kết quả thu được khác 0

CF = 1 vì có nhỡ từ bit msb trong phép cộng

OF = 0 vì dấu của tổng nhận được giống như dấu của các số hạng tham gia phép cộng (còn khi thực hiện phép cộng dưới dạng nhị phân bạn sẽ thấy có nhỡ vào bit msb đồng thời cũng có nhỡ từ msb).

Ví dụ 5.2 : Thực hiện phép cộng AL,BL trong đó AL và BL cùng chứa 80h

Trả lời:

$$\begin{array}{r}
 80h \\
 + \quad 80h \\
 \hline
 100h
 \end{array}$$

Kết quả nhận được trong AL bằng 0

SF = 0 vì msb = 0.

PF = 1 vì tất cả các bit của tổng bằng 0.

ZF = 1 vì kết quả thu được bằng 0

CF = 1 vì có nhỡ từ bit msb trong phép cộng

OF = 1 vì các số hạng tham gia phép cộng đều là các số âm nhưng kết quả nhận được là một số dương (còn khi thực hiện phép cộng dưới dạng nhị phân bạn sẽ thấy không có nhỡ vào bit msb nhưng lại có nhỡ từ bit msb).

Ví dụ 5.3 : Thực hiện phép trừ AX,BX trong đó AX chứa 8000h còn BX chứa 0001h

Trả lời :

$$\begin{array}{r}
 8000h \\
 + 0001h \\
 \hline
 7FFFh
 \end{array}$$

Kết quả nhận được trong AX là 7FFFh = 0111 1111 1111 1111

SF = 0 vì msb = 0.

PF = 1 vì có 8 bit 1 (chẵn các bit 1) trong byte thấp của kết quả.

ZF = 0 vì kết quả thu được khác 0

CF = 0 vì số không dấu nhỏ hơn bị trừ từ số lớn hơn.

Còn về cờ OF, xét về phương diện các số có dấu, chúng ta đã trừ một số dương từ một số âm, cũng giống như cộng 2 số âm. Tuy nhiên kết quả nhận được lại là số dương (sai dấu) do đó OF = 1.

Ví dụ 5.4: Tăng AL lên 1 khi AL chứa FFh

Trả lời:

$$\begin{array}{r}
 \text{FFh} \\
 + 1h \\
 \hline
 100h
 \end{array}$$

Kết quả nhận được trong AL là 00h, SF = 0, PF = 1, ZF = 1. Mặc dù có nhá CF không bị ảnh hưởng bởi lệnh INC. Có nghĩa là nếu trước đó CF = 0 thì cuối cùng CF vẫn bằng 0.

OF = 0 vì 2 số hạng có dấu khác nhau (có nhá vào msb đồng thời cũng có nhá từ msb).

Ví dụ 5.5 : Chuyển -5 vào AX

Trả lời :

Kết quả nhận được trong AX là -5 = FFFFh, không có cờ nào bị ảnh hưởng bởi lệnh MOV.

Ví dụ 5.6 : Nghịch đảo AX, trong đó AX chứa 8000h

Trả lời :

| | |
|-------------------|---------------------|
| 8000h | 1000 0000 0000 0000 |
| Số bù 1 của 8000h | 0111 1111 1111 1111 |
| | +1 |
| Số bù 2 của 8000h | 1000 0000 0000 0000 |

Kết quả nhận được trong AX là 1000 0000 0000 0000 = 8000h

SF = 1, PF = 1, ZF = 0.

CF = 1 vì trong lệnh NEG, CF luôn bằng 1 trừ khi kết quả bằng 0.

OF = 1 vì kết quả là 8000h, khi lấy nghịch đảo của một số kết quả nhận được phải có dấu ngược lại nhưng ở đây số bù 2 của 8000h lại chính bằng nó tức là dấu không thay đổi.

Trong chương tối chúng tôi sẽ giới thiệu một chương trình cho phép chúng ta thấy được sự thiết lập cờ thực sự.

5.4 Chương trình DEBUG

Chương trình DEBUG tạo ra một môi trường trong đó chương trình có thể được kiểm tra. Người sử dụng có thể từng bước duyệt qua chương trình đồng thời hiển thị và thay đổi nội dung các thanh ghi cũng như ô nhớ. Cũng có thể đưa vào trực tiếp các lệnh của Hợp ngữ mà sau đó chương trình DEBUG sẽ đổi nó sang dạng mã máy và lưu trong bộ nhớ. Những hướng dẫn chi tiết về DEBUG và CODEVIEW (một chương trình gõ rối phức tạp hơn) được trình bày trong phụ lục E.

Bây giờ chúng ta hãy sử dụng DEBUG để mô tả sự ảnh hưởng của các lệnh tới cờ, trước hết chúng ta tạo ra chương trình sau.

Chương trình PGM5_1.ASM

```

TITLE PGM5_1:CHECK FLAGS
;Chương trình được sử dụng trong DEBUG để kiểm tra việc lập
cờ
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
    MOV AX, 4000H          ;AX = 4000h
    ADD AX, AX              ;AX = 8000h

```

```

        SUB    AX,0FFFFH          ;AX = 8001h
        NEG    AX                 ;AX = 7FFFh
        INC    AX                 ;AX = 8000h
        MOV    AH,4CH
        INT    21H                ;Trở về DOS
MAIN    ENDP
END    MAIN

```

Chúng ta hãy hợp dịch và liên kết chương trình để tạo ra file PGM5_1.EXE nằm trong đĩa ở ổ A. Trong đoạn dưới đây những gì người sử dụng đánh vào được in đậm.

Chương trình DEBUG chứa trong đĩa DOS ở ổ C. Để vào DEBUG và mô tả chương trình của chúng ta, hãy đánh vào:

C>DEBUG A:PGM5_1.EXE

DEBUG sẽ đáp lại bằng dấu nhắc của nó: "-", và đợi lệnh đưa vào. Trước hết chúng ta có thể xem các thanh ghi bằng cách đánh vào lệnh "R".

-R

```

AX=0000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000
DI=0000 DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=0000 NV UP DI PL NZ
NA PO NC

OEE6:0000 B80040      MOV    AX,4000

```

Trên màn hình cho thấy nội dung của các thanh ghi ở dạng hex, tại dòng thứ 3 chúng ta thấy:

OEE6:0000 B80040 MOV AX,4000

OEE6:0000 Là địa chỉ của lệnh tiếp theo sẽ được thực hiện cho dưới dạng segment:offset.

B80040 là mã máy của lệnh đó. Đoạn OEE6 là nơi DOS quyết định nạp chương trình vào, nếu bạn thử làm ví dụ này có thể bạn sẽ nhận được địa chỉ khác. 8 cặp chữ xuất hiện bên phải của dòng thứ 2 là trạng thái hiện tại của các cờ. Các cờ xuất hiện theo trình tự: OF, DF, IF, SF, ZF, AF, PF và CF. Bảng 5.2 trình bày các ký hiệu DEBUG sử dụng cho các cờ. Các bạn có thể thấy các cờ

đã bị DEBUG xoá, ý nghĩa của các cờ điều khiển sẽ được giải thích ở chương 11 và chương 15.

Để từng bước thực hiện chương trình, chúng ta sử dụng lệnh "T" (Trace), trước khi làm điều đó hãy hiển thị lại nội dung các thanh ghi một lần nữa:

-R

```
AX=0000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000  
DI=0000 DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=0000 NV UP DI PL NZ  
NA PO NC  
  
0EE6:0000 B80040      MOV     AX,4000
```

Lệnh đầu tiên là MOV AX,4000h

-T

```
AX=4000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000  
DI=0000 DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=0003 NV UP DI PL NZ  
NA PO NC  
  
0EE6:0003 03C0 ADD     AX,AX
```

Việc thực hiện lệnh MOV AX, 4000h đưa 4000h vào thanh ghi AX, các cờ vẫn không thay đổi vì lệnh MOV không làm ảnh hưởng tới cờ. Vậy giờ chúng ta hãy thực hiện lệnh ADD AX, AX:

-T

```
AX=8000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000  
DI=0000 DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=0005 OV UP DI NG NZ  
NA PE NC  
  
0EE6:0005 2DFFFF      SUB     AX,FFFF
```

Bây giờ AX chứa $4000h + 4000h = 8000h$. SF trở thành 1(NG) để chỉ ra kết quả là số âm. Dấu hiệu tràn được chỉ ra bởi OF = 1 (OV) vì chúng ta cộng 2 số dương và nhận được một số âm. PF đặt bằng 1 (PE) vì byte thấp của kết quả không có bit 1 nào.

Bây giờ chúng ta thực hiện lệnh SUB AX,0FFFFh:

-T

```

AX=8001 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000
DI=0000 DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=0008 NV UP DI NG NZ
AC PO CY

OEE6:0008 F7D8 NEG AX

```

AX chứa 8000h- FFFFh = 8001h. OF đổi trở lại thành 0 (NV), vì chúng ta trừ 2 số cùng dấu không thể xảy ra hiện tượng tràn có dấu. Tuy nhiên CF = 1 chỉ ra rằng đã có hiện tượng tràn không dấu vì chúng ta đã trừ một số không dấu lớn hơn từ một số nhỏ hơn, khiến cho có sự vay vào bit msb. PF = 0 vì byte thấp của AX có một bit 1 duy nhất.

Bây giờ chúng ta tiếp tục với lệnh NEG AX:

-T

```

AX=7FFF BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000
DI=0000 DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=000A NV UP DI PL NZ
AC PE CY

OEE6:000A 40 INC AX

```

AX chứa số bù 2 của 8001h = 7FFFh. Đối với lệnh NEG CF luôn bằng 1 trừ khi kết quả bằng 0 nhưng trường hợp này lại không xảy ra ở đây. Cờ OF = 0 vì kết quả không phải là 8000h.

Cuối cùng chúng ta thực hiện lệnh INC AX:

-T

```

AX=8000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000
DI=0000 DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=000B OV UP DI NG NZ
AC PE CY

OEE6:000B B44C MOV AH, 4C

```

Cờ OF thay đổi trở lại thành 1 (OV) vì chúng ta cộng 2 số dương (7FFFh và 1) nhưng lại nhận được kết quả là một số âm. Mặc dù không có nhớ từ bit msb nhưng CF vẫn giữ nguyên bằng 1 vì lệnh INC không ảnh hưởng tới cờ này.

Để kết thúc việc thực hiện chương trình chúng đánh vào "G" (Go):

-G

Program terminated normally

Và để thoát khỏi DEBUG chúng ta đánh Q (Quit)

-Q

C>

Bảng 5.2 Các ký hiệu cờ của DEBUG

| Cờ trạng thái | Ký hiệu thiết lập (1) | Ký hiệu xoá(0) |
|---------------|-----------------------|--------------------|
| CF | CY (có nhớ) | NC (không nhớ) |
| PF | PE (chẵn) | PO (lẻ) |
| AF | AC (có nhớ phụ) | NA (không nhớ phụ) |
| ZF | ZR (zero) | NZ(không zero) |
| SF | NG (âm) | PL(đương) |
| OF | OV (tràn) | NV(không tràn) |
| Cờ điều khiển | | |
| DF | DN (giảm) | UP(tăng) |
| IF | EI (cho phép ngắt) | DI (cấm ngắt) |

TỔNG KẾT

Trong chương này chúng ta đã nghiên cứu những vấn đề sau

- ◆ Thanh ghi cờ là một thanh ghi của bộ vi xử lý 8086, 6 bit của nó gọi là cờ trạng thái và 3 bit khác gọi là cờ điều khiển.
- ◆ Các cờ trạng thái phản ánh kết quả của một thao tác, chúng là các cờ: cờ nhớ(CF), cờ chẵn lẻ(PF), cờ nhớ phụ(AF), cờ Zero(ZF), cờ dấu(SF) và cờ tràn(OF).
- ◆ Cờ CF được thiết lập 1 khi một thao tác cộng hay trừ gây ra nhớ hoặc vay ở bit msb, ngược lại nó bằng 0.
- ◆ Cờ PF được thiết lập nếu có một số chẵn các bit 1 trong byte thấp của kết quả, ngược lại nó bằng 0.
- ◆ Cờ AF được thiết lập 1 nếu có nhớ hoặc vay ở bit 3 trong kết quả, ngược lại chúng bằng 0.
- ◆ Cờ ZF được thiết lập 1 nếu kết quả bằng 0, ngược lại nó bằng 0.
- ◆ Cờ SF được thiết lập nếu bit msb trong kết quả bằng 1, ngược lại nó bằng 0.
- ◆ Cờ OF được thiết lập 1 nếu kết quả đúng của phép tính quá lớn để có thể chứa trong toán hạng đích, ngược lại nó bằng 0.
- ◆ Hiện tượng tràn xảy ra khi kết quả đúng của phép tính nằm ngoài phạm vi biểu diễn của máy tính. Hiện tượng tràn đó gọi là tràn có dấu nếu kết quả được xem xét như là số có dấu tương tự như vậy nó được gọi là hiện tượng tràn không dấu nếu kết quả được xem xét như là số không dấu.
- ◆ Bộ xử lý sử dụng các cờ CF và OF để báo tràn: CF = 1 nghĩa là đã xảy ra hiện tượng tràn không dấu, OF = 1 có nghĩa là xuất hiện hiện tượng tràn có dấu.
- ◆ Bộ xử lý thiết lập cờ CF nếu có nhớ từ bit msb trong phép cộng hoặc có vay vào bit msb trong phép trừ, trong trường hợp sau điều đó có nghĩa là chúng ta đã lấy một số không dấu trừ đi số lớn hơn nó
- ◆ Bộ xử lý thiết lập cờ OF nếu có nhớ vào bit msb nhưng không có nhớ từ đó hoặc ngược lại.
- ◆ Có một phương pháp khác để nhận ra hiện tượng tràn có dấu trong các phép cộng hay trừ. Trong phép cộng các số cùng dấu, hiện tượng tràn xảy ra nếu kết quả nhận được có dấu khác với dấu của các số hạng. Phép trừ các số

khác dấu cũng giống như phép cộng các số cùng dấu và hiện tượng tràn có dấu xảy ra khi kết quả nhận được có dấu khác với dấu của số bị trừ.

- ◆ Trong phép cộng các số khác dấu hay trừ các số cùng dấu hiện tượng tràn có dấu không thể xảy ra.
- ◆ Nói chung các lệnh đều có ảnh hưởng đến cờ, nhưng có một số lệnh không ảnh hưởng tới cờ hoặc chỉ ảnh hưởng tới một số cờ.
- ◆ Trạng thái của các cờ được cho thấy trong chương trình DEBUG.
- ◆ DEBUG có thể dùng để duyệt qua một chương trình, một số lệnh của nó là: "R" để hiển thị nội dung của các thanh ghi, "T" để thực hiện một lệnh của chương trình và "G" để thực hiện chương trình.

Các thuật ngữ tiếng Anh

| | |
|-----------------------|--|
| Control Flags | Cờ điều khiển- Các cờ dùng để cho phép hoặc không cho phép một thao tác nào đó của CPU |
| Flags | Các cờ- Các bit của thanh ghi cờ biểu diễn trạng thái của CPU |
| FLAGS register | Thanh ghi cờ- Thanh ghi trong CPU mà các bit của nó chính là các cờ |
| Status Flags | Cờ trạng thái- Các cờ phản ánh kết quả của một lệnh được CPU thực hiện |

Bài tập

1. Hãy cho biết nội dung mới của toán hạng đích và trạng thái mới của các cờ CF, SF, ZF, PF Và OF trong các lệnh sau (giả sử ban đầu các cờ đều bị xoá về 0)
 - a. ADD AX, AX Trong đó AX chứa 7FFFh và BX chứa 0001h
 - b. SUB AL, BL Trong đó AL chứa 01h và BL chứa FFh
 - c. DEC AL Trong đó AL chứa 00h
 - d. NEG AL Trong đó AL chứa 7Fh
 - e. XCHG AX, BX Trong đó AX chứa 1ABC_h và BX chứa 712A_h
 - f. ADD AL, BL Trong đó AL chứa 80h và BL chứa FFh
 - g. SUB AX, BX Trong đó AX chứa 0000h và BX chứa 8000h
 - h. NEG AX Trong đó AX chứa 0001h
2. a. Giả sử AX và BX đều chứa các số dương, lệnh ADD AX, BX được thực hiện, hãy chứng minh rằng hiện tượng có nhớ vào bit msb xảy ra nhưng không có nhớ ra từ nó khi và chỉ khi có hiện tượng tràn có dấu.
b. Giả sử AX và BX đều chứa các số âm, lệnh ADD AX, BX được thực hiện, hãy chứng minh rằng hiện tượng có nhớ từ msb nhưng không có nhớ vào nó xảy ra khi và chỉ khi có hiện tượng tràn có dấu.
3. Giả sử lệnh ADD AX, BX được thực hiện. Trong các phần sau đây số hạng thứ nhất được chứa trong AX, số hạng thứ 2 chứa trong BX, hãy cho biết kết quả trong thanh ghi AX và có hiện tượng tràn (có dấu và không có dấu) xảy ra không?
 - a.
$$\begin{array}{r} 512Ch \\ + 4185h \\ \hline \end{array}$$
 - b.
$$\begin{array}{r} FE12h \\ + 1ACBh \\ \hline \end{array}$$
 - c.
$$\begin{array}{r} E1E4h \\ + DAB3h \\ \hline \end{array}$$
 - d.

7132h

+ 7000h

e.

6389h

+ 1176h

4. Giả sử phép trừ SUB AX,BX được thực hiện. Trong các phần sau đây số hạng thứ nhất được chứa trong AX, số hạng thứ 2 chứa trong BX, hãy cho biết kết quả trong thanh ghi AX và có hiện tượng tràn (có dấu và không có dấu) xảy ra không ?

a.

2143h

- 1986h

b.

81FEh

- 1986h

c.

19BCh

- 81FEh

d.

0002h

- FEOFh

e.

8BCDh

- 71ABh

Chương 6

CÁC LỆNH ĐIỀU KHIỂN RẼ NHÁNH.

Tổng quan.

Để đảm bảo thực hiện được các công việc có ích, các chương trình bằng Hợp ngữ phải có các phương pháp chọn lựa và lặp lại các đoạn mã lệnh. Trong chương này, chúng tôi sẽ chỉ ra các phương pháp đó được thực hiện như thế nào nhờ các lệnh nhảy và lặp.

Các lệnh nhảy và lặp chuyển điều khiển cho các phần khác trong chương trình. Sự chuyển giao này có thể không có điều kiện, có thể phụ thuộc vào các tổ hợp riêng rẽ các cờ trạng thái.

Sau khi làm quen với các cấu trúc nhảy, chúng ta sẽ sử dụng chúng để thực hiện các cấu trúc chọn lựa và lặp của ngôn ngữ bậc cao. Các ứng dụng này sẽ làm cho việc đổi một thuật toán với các lệnh giả sang các lệnh Hợp ngữ dễ dàng hơn.

6.1 Một ví dụ về lệnh nhảy.

Để bạn có khái niệm về cách thức làm việc của cấu trúc nhảy, chúng tôi sẽ viết một chương trình hiển thị toàn bộ tập hợp các ký tự của IBM.

Chương trình PGM6_1.ASM

```
1: TITLE PGM6_1: Hiển thị các ký tự IBM.  
2: .MODEL SMALL  
3: .STACK 100H  
4: .CODE  
5: MAIN PROC  
6:     MOV AH, 2          ; hàm hiển thị ký tự
```

```

7:      MOV    CX, 256      ; số ký tự được hiển thị
8:      MOV    DL, 0       ; DL chứa mã ASCII ký tự NULL
9: PRINT_LOOP:
10:     INT   21h        ; hiển thị một ký tự
11:     INC   DL         ; tăng mã ASCII
12:     DEC   CX         ; giảm bộ đếm
13:     JNZ   PRINT_LOOP ; lặp lại nếu CX khác 0
14:; trở về DOS
15:     MOV   AH, 4Ch
16:     INT   21H
17: MAIN   ENDP
18:     END   MAIN

```

Tập hợp ký tự IBM bao gồm 256 ký tự. Các ký tự có mã từ 32 đến 127 là các ký tự ASCII chuẩn in được đã giới thiệu trong chương 2. IBM cũng cung cấp một hệ thống các ký tự đồ họa có mã từ 0 đến 32 và từ 128 đến 255.

Để hiển thị các ký tự chúng ta dùng một vòng lặp (từ dòng 9 đến dòng 13). Trước khi vào vòng lặp AH được khởi tạo giá trị 2 (hàm hiển thị một ký tự) và DL được đặt bằng 0 là mã ASCII ký tự đầu tiên. CX là bộ đếm vòng lặp, nó được đặt bằng 256 trước khi vào vòng lặp và giảm 1 sau khi mỗi ký tự được hiển thị.

Lệnh JNZ (Jump if Not Zero) điều khiển vòng lặp. Nếu kết quả của lệnh kế trước (DEC CX) khác 0, lệnh JNZ sẽ chuyển điều khiển đến lệnh có nhãn PRINT_LOOP. Cuối cùng khi DX bằng 0 chương trình tiếp tục thực hiện các lệnh trở về DOS. Hình 6.1 là kết quả khi chạy chương trình. Tất nhiên với các mã ASCII của các ký tự điều khiển như lùi một ký tự, về đầu dòng v.v. các chức năng điều khiển sẽ được thực hiện thay vì hiển thị chúng.

Lưu ý: PRINT_LOOP là nhãn dòng lệnh lần đầu tiên chúng ta sử dụng trong một chương trình. Các nhãn cần thiết khi một lệnh trả đến lệnh khác giống như trong trường hợp trên. Nhãn kết thúc bằng dấu hai chấm và để dễ nhận ra, chúng được đặt riêng một dòng. Các nhãn tham trả tới lệnh ngay sau chúng.

Hình 6.1. Kết quả chương trình PRG6_1.

```
C:\BIN>pgm6_1
♥♦♣↑↓→←!"#$%&'() *+,./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ
XYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}-❖ÜéâååçéééííííÅÉæððù
üýÖÜ¢£¥ fáióúñÑ°¿ ¬₩₩;«»
          ß      μ      ±      +  o..  ²
C:\BIN>
```

6.2 Các lệnh nhảy có điều kiện.

- JNZ là một ví dụ của lệnh nhảy có điều kiện. Cú pháp:

JNZ nhän đích

Nếu như điều kiện của lệnh nhảy được thoả mãn, lệnh có nhãn `nhãn_dịch` sẽ được thi hành. Lệnh này có thể ở trước hoặc sau lệnh nhảy. Nếu điều kiện không thoả mãn, lệnh ngay sau lệnh nhảy được thi hành. Với lệnh `JNZ` điều kiện là kết quả của lệnh trước nó khác 0.

Phạm vi của lệnh nhảy có điều kiện

Cấu trúc mã máy của lệnh nhảy có điều kiện đòi hỏi nhãn _dịch phải đứng trước lệnh nhảy không quá 126 byte hoặc đứng sau không quá 127 byte (chúng tôi sẽ chỉ ra cách để vượt qua giới hạn này sau).

CPU thực hiện một lệnh nhảy như thế nào?

Để thực hiện một lệnh nhảy, CPU nhìn vào thanh ghi cờ. Ta đã biết rằng thanh ghi phản ánh kết quả công việc gần nhất mà bộ vi xử lý thực hiện. Nếu điều kiện của lệnh nhảy (được phát biểu như một tổ hợp các sự kiện cờ trạng thái) thỏa mãn, CPU điều chỉnh IP trả đến nhãn đích và như thế lệnh ở sau nhãn này sẽ được thi hành tiếp theo. Nếu điều kiện nhảy không thỏa mãn, IP không bị sửa đổi. Điều này có nghĩa là lệnh trên dòng tiếp theo sẽ được thi hành.

Trong chương trình trên, CPU thi hành lệnh JNZ PRINT_LOOP phụ thuộc vào ZF. Nếu ZF=0 điều khiển sẽ được chuyển tới nhãn PRINT_LOOP; nếu ZF=1 chương trình tiếp tục thực hiện lệnh MOV AH,4Ch.

Bảng 6.1 chỉ ra các lệnh nhảy có điều kiện. Có ba loại đó là:

- (1) Các lệnh nhảy có dấu được dùng khi kết quả trả về là các số có dấu.
- (2) Các lệnh nhảy không dấu dùng với các số không dấu.
- (3) Các lệnh nhảy điều kiện đơn: điều kiện phụ thuộc vào một cờ riêng biệt.

Lưu ý: Các lệnh nhảy tự nó không ảnh hưởng tới các cờ.

Cột đầu tiên trong bảng 6.1 là mã lệnh nhảy. Rất nhiều lệnh nhảy có hai mã lệnh. Ví dụ: JG và JNLE. Cả hai mã lệnh này có cùng một mã máy. Khi thực hiện chúng cho kết quả hoàn toàn giống nhau.

Bảng 6.1. Các lệnh nhảy có điều kiện.

Các lệnh nhảy có dấu.

| Ký hiệu | Chức năng | Điều kiện nhảy |
|---------|---|----------------|
| JG/JNLE | nhảy nếu lớn hơn nhảy nếu không nhỏ hơn hay bằng | ZF=0 và SF=OF |
| JGE/JNL | nhảy nếu lớn hơn hay bằng nhảy nếu không nhỏ hơn | SF=OF |
| JL/JNGE | nhảy nếu nhỏ hơn nhảy nếu không lớn hơn hay bằng | SF<>OF |
| JLE/JNG | nhảy nếu nhỏ hơn hay bằng nhảy nếu không lớn hơn | ZF=1 hay SF=OF |

Các lệnh nhảy không dấu.

| Ký hiệu | Chức năng | Điều kiện nhảy |
|---------|---|----------------|
| JA/JNBE | nhảy nếu lớn hơn nhảy nếu không nhỏ hơn hay bằng | CF=0 và ZF=0 |
| JAE/JNB | nhảy nếu lớn hơn hay bằng nhảy nếu không nhỏ hơn | CF=0 |
| JB/JNAE | nhảy nếu nhỏ hơn nhảy nếu không lớn hơn hay bằng | CF=1 |

| | | |
|----------------|---|---------------|
| JBE/JNA | nhảy nếu nhỏ hơn hay bằng nhảy nếu không lớn hơn | CF=1 hay ZF=1 |
|----------------|---|---------------|

Các lệnh nhảy điều kiện đơn.

| Ký hiệu | Chức năng | Điều kiện nhảy |
|----------------|--|----------------|
| JE/JZ | nhảy nếu bằng nhảy nếu bằng 0 | ZF=1 |
| JNE/JNZ | nhảy nếu không bằng nhảy nếu khác 0 | ZF=0 |
| JC | nhảy nếu có nhỡ | CF=1 |
| JNC | nhảy nếu không nhỡ | CF=0 |
| JO | nhảy nếu nến tràn | OF=1 |
| JNO | nhảy nếu không tràn | OF=0 |
| JS | nhảy nếu dấu âm | SF=1 |
| JNS | nhảy nếu dấu dương | SF=0 |
| JP/JPE | nhảy nếu cờ chẵn | PF=1 |
| JNP/JPO | nhảy nếu cờ lẻ | PF=0 |

Lệnh CMP.

Các điều kiện nhảy thường được cung cấp bởi lệnh CMP (compare). Nó có dạng sau:

CMP đích, nguồn

Lệnh này so sánh toán tử đích với toán tử nguồn bằng cách lấy toán tử đích trừ đi toán tử nguồn. Kết quả không được lưu lại nhưng các cờ bị ảnh hưởng. Các toán hạng của lệnh CMP không thể cùng là các ô nhớ. Toán hạng đích không được phép là hằng số. Chú ý: CMP giống hệt như SUB ngoại trừ việc toán hạng đích không bị thay đổi.

Ví dụ. Giả thiết chương trình chứa hai dòng lệnh sau đây:

CPM AX, BX

JG BELOW

Ở đây AX=7FFFh, BX=0001. Kết quả so sánh AX và BX là 7FFFh-0001h=7FFEh.

Bảng 6.1 chỉ ra rằng điều kiện nhảy cho lệnh JG đã được thoả mãn bởi vì ZF=S傅=OF=0, và do đó điều khiển được chuyển đến nhãn BELOW.

Dịch các lệnh có điều kiện.

Trong ví dụ vừa nêu, khi xem xét các cờ sau lệnh CMP ta thấy rằng điều khiển sẽ được chuyển đến nhãn BELOW. Đó cũng là cách CPU thực hiện một lệnh nhảy có điều kiện. Người lập trình không cần thiết là phải suy nghĩ nhiều về các cờ, bạn có thể chỉ dùng tên của lệnh nhảy để quyết định việc chuyển điều khiển đến nhãn đích. Các lệnh sau đây:

CMP AX, BX

JG BELOW

Nếu như AX lớn hơn BX (coi là số có dấu) thì JG (jump if greater than) sẽ chuyển đến BELOW.

Đặc biệt cả khi nghĩ rằng lệnh CMP được thiết kế chỉ dùng cho các lệnh nhảy có điều kiện, chúng vẫn có thể đi kèm với các lệnh khác như trong chương trình PGM6_1. Một ví dụ khác:

DEC AX

JL THERE

trường hợp này nếu nội dung của AX (coi là số có dấu) nhỏ hơn 0, điều khiển sẽ được chuyển đến THERE.

So sánh các lệnh nhảy có dấu và không dấu.

Mỗi lệnh nhảy có dấu đều tương ứng với một lệnh nhảy không dấu. Ví dụ lệnh nhảy có dấu JG và lệnh nhảy không dấu JA. Dùng lệnh nhảy có dấu hay không dấu tuỳ thuộc vào kiểu số được đưa ra. Trên thực tế, như bảng 6.1 chỉ ra, các lệnh này thao tác với các cờ khác nhau: các lệnh nhảy có dấu sử dụng các cờ ZF,S傅 và OF trong khi các lệnh nhảy không dấu lại dùng các cờ ZF và CF. Sử dụng không đúng loại có thể đưa đến kết quả sai.

Ví dụ. giả thiết rằng chúng ta làm việc với các số không dấu. Nếu như AX=7FFFh và BX=8000h, khi ta thực hiện:

CMP AX, BX

JA BELOW

thậm chí 7FFFh > 8000h trong dạng có dấu, chương trình vẫn không nhảy đến nhãn BELOW. Nguyên nhân ở đây là 7FFFh < 8000h ở dạng không dấu và ở đây chúng ta lại dùng lệnh nhảy không dấu JA.

Làm việc với các ký tự.

Khi làm việc với tập hợp ký tự ASCII chuẩn cả các lệnh nhảy có điều kiện và không điều kiện đều có thể được sử dụng bởi lẽ bit dấu của byte chứa mã ký tự luôn là 0. Dù sao thì các lệnh nhảy không dấu phải được sử dụng khi so sánh các ký tự ASCII mở rộng (mã từ 80h đến FFh).

Ví dụ 6.1.

Giả sử AX và BX chứa các số có dấu. Hãy viết các lệnh để đưa số lớn nhất vào CX.

Trả lời:

```
MOV CX, AX      ; đưa AX vào CX
CMP BX, CX      ; BX lớn hơn?
JLE NEXT        ; không, tiếp tục
MOV CX, BX      ; đúng, đưa BX vào CX
NEXT:
```

6.3. Lệnh JMP.

Lệnh JMP (jump) dẫn đến việc chuyển điều khiển không điều kiện (nhảy không điều kiện). Cú pháp:

JMP đích

Ở đây đích phải là một nhãn trong cùng một đoạn với JMP (xem phụ lục F với chi tiết chi tiết hơn).

JMP có thể được dùng để khắc phục khoảng giới hạn của lệnh nhảy có điều kiện. Ví dụ: giả sử chúng ta muốn thực hiện vòng lặp:

```
TOP:
; thân vòng lặp
DEC CX          ; giảm bộ đếm
JNZ TOP         ; lặp nếu CX > 0
MOV AX, BX
```

nhưng thân vòng lặp lại chứa quá nhiều lệnh đến mức nhãn TOP nằm ngoài khoảng giới hạn của lệnh JNZ (nhiều hơn 126 byte trước JNZ TOP). Chúng ta có thể sửa lại:

```
TOP:
; thân vòng lặp
DEC CX          ; giảm bộ đếm
```

| | | | |
|---------|-----|--------|----------------|
| | JNZ | BOTTOM | lặp nếu CX > 0 |
| | JMP | EXIT | |
| BOTTOM: | JMP | TOP | |
| EXIT: | MOV | AX, BX | |

6.4. Các cấu trúc ngôn ngữ bậc cao.

Chúng tôi đã có lần nói rằng cấu trúc nhảy có thể được dùng để thực hiện các công việc rẽ nhánh và lặp. Tuy nhiên do các lệnh nhảy quá sơ khai nên rất khó mã hoá một thuật toán (có hoặc không có các dòng hướng dẫn) nhất là đối với những người mới lập trình.

Bởi vì đa số các bạn đã có chút ít kinh nghiệm về các cấu trúc của ngôn ngữ bậc cao như cấu trúc chọn lựa IF_THEN_ELSE hay các vòng lặp WHILE, chúng tôi sẽ nêu ra cách giả lập các cấu trúc này trong ngôn ngữ Hợp ngữ. Trong trường hợp đầu tiên chúng tôi sẽ đưa ra cấu trúc các toán tử giả của ngôn ngữ bậc cao.

6.4.1 Các cấu trúc rẽ nhánh.

Trong các ngôn ngữ bậc cao, các cấu trúc rẽ nhánh của một chương trình để chọn các đường dẫn khác nhau và phụ thuộc vào các điều kiện. Phần này chúng ta sẽ xem xét 3 cấu trúc.

IF_THEN

Cấu trúc IF_THEN có thể được khai báo dưới dạng toán tử giả như sau:

```

IF điều_kiện
  THEN
    nhánh_đúng
  END_IF

```

Xem hình 6.2.

Điều_kiện là một biểu thức có thể đúng hoặc sai. Nếu nó đúng, nhánh_đúng sẽ được thực hiện. Ngược lại, cấu trúc không thực hiện lệnh nào, chương trình tiếp tục với các lệnh theo sau.

Ví dụ 6.2. Thay số trong AX bằng giá trị tuyệt đối của nó.

Trả lời : Một thuật toán với mã lệnh giả:

```

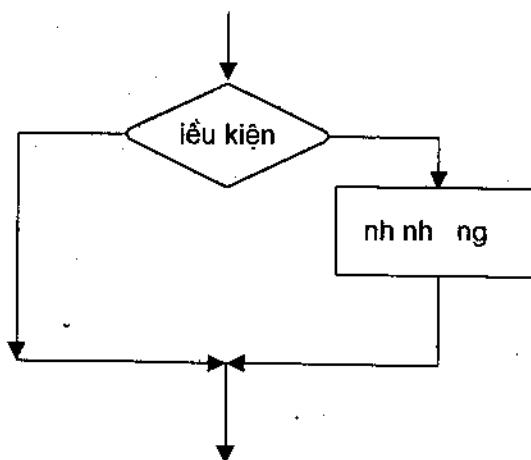
IF AX>0
  THEN
    thay AX bằng -AX
  END_IF

```

Nó có thể được mã hoá như sau:

```
; if AX<0
    CMP AX, 0           ; AX<0 ?
    JNL END_IF          ; không, thoát ra.
; then
    NEG AX              ; đúng, đổi dấu
END_IF:
```

Hình 6.2. IF THEN



Điều kiện $AX < 0$ được kiểm tra bởi lệnh $CMP AX, 0$. Nếu AX không nhỏ hơn 0, ta không phải làm gì cả, JNL được dùng để nhảy qua lệnh $NEG AX$. nếu điều kiện $AX < 0$ thỏa mãn, chương trình tiếp tục thực hiện lệnh $NEG AX$.

IF_THEN_ELSE.

IF điều_kiên

THEN

nhánh_đúng

ELSE

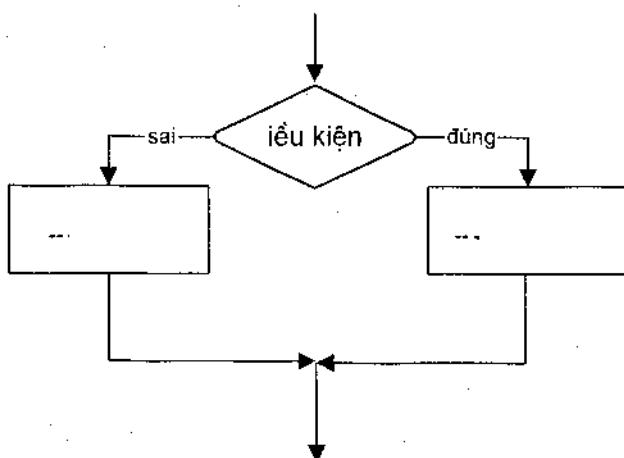
nhánh_sai

END_IF

Xem hình 6.3

Trong cấu trúc này nếu điều_kiên là đúng, nhóm lệnh nhánh_đúng sẽ được thi hành. Còn nếu điều_kiên sai, nhóm lệnh nhánh_sai sẽ được thi hành.

Hình 6.3. IF_THEN_ELSE



Ví dụ 6.3. Giả sử AL và BL chứa các ký tự ASCII mở rộng. Hãy hiển thị ký tự đúng trước trong bảng mã.

Trả lời :

```

IF AL<=BL
  THEN
    hiển thị ký tự trong AL
  ELSE
    hiển thị ký tự trong BL
END_IF
  
```

Ta có thể mã hoá nó như sau:

```

MOV AH, 2          ; chuẩn bị hiển thị
;if AL<=BL
  CMP AL, BL      ; AL<=BL ?
  JNBE ELSE_        ; không, hiển thị ký tự trong BL
;then
  MOV DL, AL       ; chuyển ký tự vào DL để hiển thị
  JMP DISPLAY      ; tới DISPLAY
;ELSE_:
  MOV DL, BL       ; chuyển ký tự vào DL để hiển thị
DISPLAY:
  INT 21h          ; hiển thị nó.
;END_IF
  
```

Chú ý: Ta dùng nhãn ELSE_ vì ELSE là từ dành riêng.

Điều kiện AL<=BL được kiểm tra bởi lệnh CMP AL,BL. Nếu nó sai, chương trình sẽ nhảy qua nhánh_đúng tới ELSE_. Chúng ta sử dụng lệnh nhảy không dấu JNBE bởi lẽ chúng ta đang so sánh các kí tự mở rộng.

Nếu AL<=BL thoả mãn, nhánh_đúng được thực hiện. Lưu ý rằng chỉ thị JMP DISPLAY là cần thiết để nhảy qua nhánh_sai. Điều này khác trong với ngôn ngữ bậc cao: nhánh_sai được tự động nhảy qua nếu nhánh-đúng được thực hiện.

CASE.

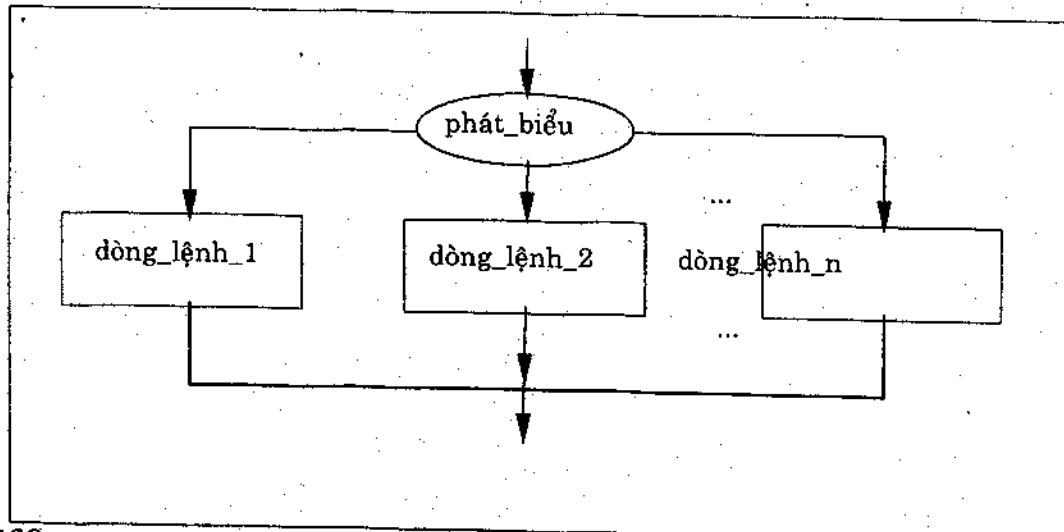
CASE là một cấu trúc đa nhánh, nó kiểm tra các thanh ghi, các biến hay các biểu thức với các giá trị riêng rẽ trong miền giá trị. Dạng tổng quát của nó là:

```
CASE phát_biểu
    giá_trị_1: dòng_lệnh_1
    giá_trị_2: dòng_lệnh_2
    .
    .
    .
    giá_trị_n: dòng_lệnh_n
END_CASE
```

Xem hình 6.4

Trong cấu trúc này phát_biểu được kiểm tra, nếu giá trị của nó bằng với giá_trị_i thì dòng_lệnh_i sẽ được thi hành. Ta giả thiết tập hợp giá_trị_1... giá_trị_n tách biệt nhau

Hình 6.4. CASE



Ví dụ 6.4. Nếu AX chứa một số âm, hãy nhập -1 vào BX, nếu AX chứa 0, cho BX bằng 0, nếu AX dương đổi BX thành 1.

Lời giải:

CASE AX

<0: gán BX bằng -1
=0: gán BX bằng 0
>0: gán BX bằng 1

END_CASE

Ta có thể mã hoá như sau:

;case AX

CMP AX, 0 ; kiểm tra AX
JL NEGATIVE ; AX<0
JE ZERO ; AX=0
JG POSITIVE ; AX>0

NEGATIVE:

MOV BX, -1 ; nhập -1 vào BX
JMP END_CASE ; rồi thoát

ZERO:

MOV BX, 0 ; nhập 0 vào BX
JMP END_CASE ; rồi thoát

POSITIVE:

MOV BX, 1 ; nhập 1 vào BX

END_CASE:

Các nhánh với điều kiện kép.

Đôi khi điều kiện nhánh của IF hay CASE có dạng:

điều_kiện_1 AND điều_kiện_2

hay

điều_kiện_1 OR điều_kiện_2

Ở đây điều_kiện_1 và điều_kiện_2 có thể đúng hoặc sai. Đầu tiên chúng ta hãy xem xét điều kiện AND (AND condition), sau đó đến điều kiện OR (OR condition).

Các điều kiện AND.

Điều kiện AND chỉ đúng khi cả hai điều kiện: điều_kiện_1 và điều_kiện_2 cùng đúng. Ngược lại nếu một trong chúng sai, điều kiện AND cũng sẽ sai.

Ví dụ 6.6. Đọc một ký tự. Nếu là chữ hoa thì hiển thị nó.

Lời giải:

Đọc một ký tự (vào DL)

IF ('A' <= ký_tự) và (ký_tự <= 'Z')

THEN

 hiển thị ký tự

END_IF

Để mã hoá, đầu tiên chúng ta kiểm tra xem ký tự trong AL có đứng sau 'A' trong bảng mã hay không, nếu sai ta có kết thúc. Nếu đúng, trước khi hiển thị ký tự ta vẫn còn phải kiểm tra ký tự có đứng trước 'Z' hay không. Sau đây là mã lệnh:

; đọc một ký tự

```
MOV AH,1          ; chuẩn bị đọc
INT 21h          ; ký tự vào AL
;if ( 'A' <= ký_tự ) và ( ký_tự <= 'Z' )
    CMP AL,'A'      ; ký_tự >='A' ?
    JNGE END_IF     ; không, thoát ra
    CMP AL,'Z'      ; ký_tự <='Z' ?
    JNLE END_IF     ; không, thoát ra
; then hiển thị ký tự
    MOV DL,AL        ; lấy ký tự
    MOV AH,2          ; chuẩn bị hiển thị
    INT 21h          ; hiển thị ký tự
END_IF:
```

Các điều kiện OR.

Điều_kiện_1 OR điều_kiện_2 là đúng khi điều_kiện_1 hoặc điều_kiện_2 đúng. Nó chỉ sai khi cả hai điều kiện thành phần cùng sai.

Ví dụ 6.7. Đọc một ký tự. Nếu là 'y' hay 'Y' thì hiển thị nó. Nếu ngược lại, kết thúc chương trình.

Lời giải:

```
Đọc một ký tự (vào AL).  
IF (ký_tự = 'y') hoặc (ký_tự = 'Y')  
THEN  
    hiển thị ký tự  
ELSE  
    kết thúc chương trình.  
END_IF
```

Để mã hoá, đầu tiên chúng ta kiểm tra ký_tự = 'y'? Nếu thoả mãn, điều kiện OR đúng và chúng ta có thể thực hiện dòng lệnh THEN. Ngược lại vẫn cơ hội để điều kiện OR đúng, đó là khi ký_tự bằng 'Y', và dòng lệnh THEN được thi hành. Nếu điều này vẫn sai, điều kiện OR là sai và chúng ta sẽ thực hiện dòng lệnh ELSE. Sau đây là mã lệnh:

```
; đọc một ký tự  
    MOV AH, 1          ; chuẩn bị đọc  
    INT 21h           ; ký tự trong AL  
; if ( ký_tự = 'y' ) hoặc ( ký_tự = 'Y' )  
    CMP AL, 'y'       ; ký_tự = 'y' ?  
    JE _THEN         ; đúng, chuyển đến hiển thị ký tự  
    CMP AL, 'Y'       ; ký_tự = 'Y'  
    JE _THEN         ; đúng, chuyển đến hiển thị ký tự  
    JMP ELSE_         ; sai, kết thúc  
  
_THEN:  
    MOV AH, 2          ; chuẩn bị hiển thị  
    MOV DL, AL          ; lấy ký tự  
    INT 21h           ; hiển thị nó  
    JMP END_IF         ; và thoát ra  
  
ELSE_:  
    MOV AH, 4Ch          ;
```

```
INT    21h          ; trả về DOS  
END_IF:
```

6.4.2 Các cấu trúc lặp.

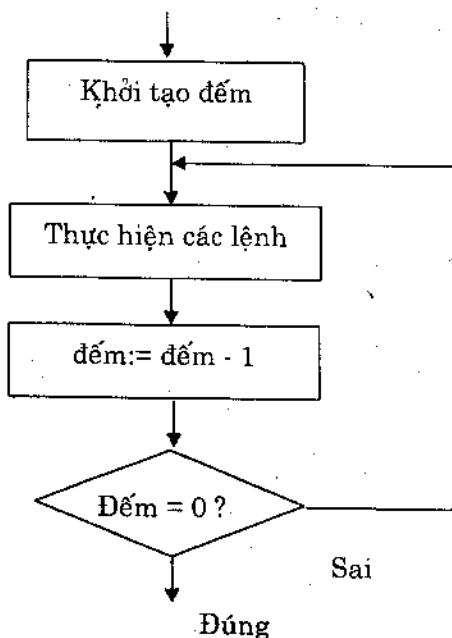
Một vòng lặp là một chuỗi các lệnh được lặp lại. Số lần lặp có thể đã xác định trước hoặc phụ thuộc vào các điều kiện.

Vòng lặp FOR.

Đây là một cấu trúc lặp mà số lần lặp lại các dòng lệnh đã biết trước (vòng lặp điều khiển bằng biến đếm). Dạng mã lệnh giả:

```
FOR  số_lần_lặp  DO  
    các dòng lệnh  
END FOR
```

Xem hình 6.5



Hình 6.5: Vòng FOR

Ta có thể sử dụng lệnh LOOP để thực hiện vòng lặp FOR. Lệnh này có dạng:

LOOP nhän_dich

Bộ đếm vòng lặp là thanh ghi CX được khởi tạo bằng số_lần_lặp. Mỗi lần thực hiện lệnh LOOP, thanh ghi CX tự động giảm đi 1 và nếu CX khác 0 thì điều khiển được chuyển tới nhãn đích. Nếu CX = 0, lệnh tiếp sau lệnh LOOP sẽ được thi hành. Nhãn đích phải ở trước lệnh lặp không quá 126 byte.

Vòng lặp FOR có thể được thực hiện nhờ lệnh LOOP như sau:

; khởi tạo CX bằng số_lần_lặp

TOP:

; thân vòng lặp

LOOP TOP

Ví dụ 6.8. Viết một vòng lặp điều khiển bằng biến đếm hiển thị một dòng 80 dấu sao.

Lời giải:

```
FOR 80 times DO  
    hiển thị '*'  
END_FOR
```

Mã lệnh là:

```
MOV CX, 80      ; số các dấu sao được hiển thị  
MOV AH, 2        ; hàm hiển thị ký tự  
MOV DL, '*'     ; ký tự hiển thị  
TOP:  
    INT 21h       ; hiển thị một dấu sao  
    LOOP TOP      ; lặp lại 80 lần
```

Bạn hãy lưu ý rằng vòng lặp FOR thực hiện bởi lệnh LOOP sẽ được thực hiện ít nhất một lần. Thực ra nếu CX bằng 0 khi vào vòng lặp, lệnh LOOP giảm CX thành OFFFFh và lệnh LOOP sẽ được thực hiện OFFFFh = 65535 lần nữa. Để khắc phục điều này, lệnh JCXZ (jump if CX is zero) được đặt trước vòng lặp. Cú pháp của nó là:

JCXZ nhän_dich

Nếu CX bằng 0, điều khiển sẽ được chuyển đến nhãn đích. Như vậy vòng lặp sẽ bị bỏ qua nếu CX bằng 0:

```
JCXZ SKIP  
TOP:  
    ;thân vòng lặp  
    LOOP TOP  
SKIP:
```

Vòng lặp WHILE.

Đây là vòng lặp phụ thuộc vào một điều kiện. Dạng mã lệnh giả:

```
WHILE điều_kiện DO  
    các dòng lệnh  
END WHILE
```

Xem hình 6.6

Điều_kiện được kiểm tra ở đầu vòng lặp. Nếu nó đúng thì các dòng lệnh sẽ được thi hành. Ngược lại nếu điều_kiện sai, chương trình tiếp tục thực hiện lệnh ở sau vòng lặp. Rất có thể ngay khi khởi đầu điều_kiện đã không thỏa mãn. Trong trường hợp này thân vòng lặp sẽ không được thực hiện lần nào. Vòng lặp tiếp tục được thực hiện khi điều_kiện còn đúng.

Ví dụ 6.9. Viết các lệnh để đếm số ký tự trong một dòng.

Lời giải:

```
khởi tạo bộ đếm bằng 0,  
đọc một ký tự  
WHILE ký_tự <> ký_tự_về_đầu_dòng DO  
    đếm = đếm + 1  
    đọc một ký tự  
END WHILE
```

Các lệnh là:

```
MOV DX, 0          ; DX đếm số ký tự  
MOV AH, 1          ; chuẩn bị đọc
```

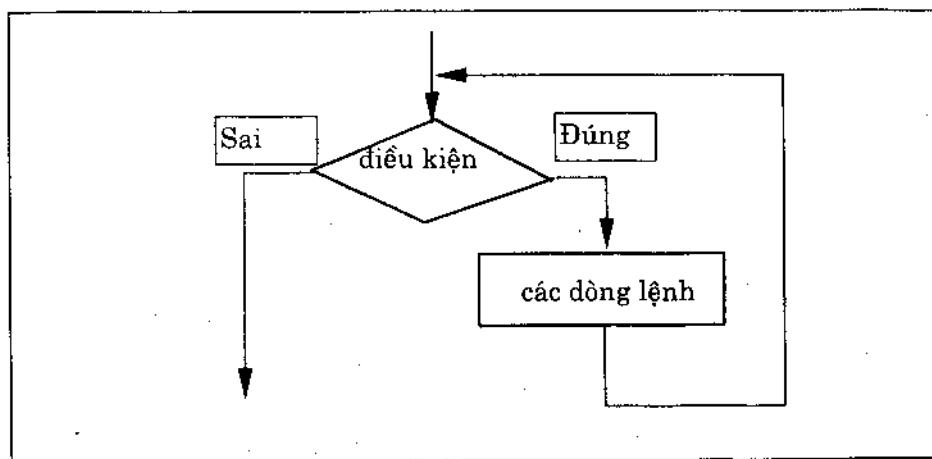
```

INT    21h          ; ký tự trong AL
WHILE_:
    CMP    AL, 0Dh      ; CR ?
    JE     END_WHILE   ; đúng, thoát ra
    INC    DX          ; không phải CR, tăng bộ đếm
    INT    21h          ; đọc một ký tự
    JMP    WHILE_       ; lặp lại
END_WHILE:

```

Lưu ý là do vòng lặp WHILE kiểm tra điều kiện kết thúc ở đầu vòng lặp nên bạn cần chắc chắn rằng bất cứ biến nào liên quan đến điều kiện vòng lặp đều phải được khởi tạo trước khi vào vòng lặp. Vì vậy bạn phải đọc một ký tự trước khi vào vòng lặp rồi lại phải đọc ký tự khác ở cuối nó. Ta dùng nhãn WHILE_ vì WHILE là từ dành riêng.

Hình 6.6. Vòng lặp WHILE



Vòng lặp REPEAT

Có một vòng lặp có điều kiện khác đó là vòng lặp REPEAT. Dạng mã lệnh giả của nó là:

```

REPEAT
    các dòng lệnh
UNTIL  điều_kiện

```

Xem hình 6.7.

Trong một vòng lặp REPEAT ...UNTIL, các dòng lệnh được thi hành sau đó mới kiểm tra điều_kiện. Nếu điều_kiện đúng, vòng lặp kết thúc, nếu nó sai điều khiển rẽ nhánh đến đầu vòng lặp.

Ví dụ 6.10. Viết các lệnh đọc vào các ký tự, kết thúc khi gặp một ký tự trắng.

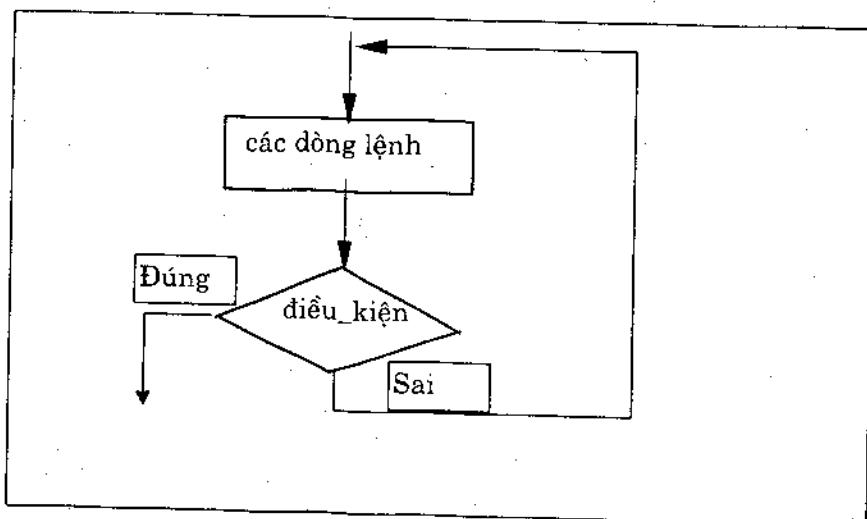
Lời giải:

REPEAT

 đọc một ký tự

UNTIL ký tự trắng

Hình 6.7 Vòng lặp REPEAT



Các lệnh là:

```
MOV    AH,1      ; chuẩn bị đọc  
REPEAT:  
    INT   21h      ; ký tự trong AL  
;intil  
    CMP   AL,' '   ; ký tự trắng ?  
    JNE   REPEAT   ; không, đọc tiếp
```

So sánh WHILE và REPEAT.

Trong nhiều trường hợp khi cần một vòng lặp có điều kiện, sử dụng vòng lặp WHILE hay REPEAT là tuỳ ý thích mỗi người. Ưu điểm của vòng lặp WHILE là vòng lặp có thể được bỏ qua khi điều kiện kết thúc khởi tạo với giá trị logic sai, trong khi đó các lệnh trong vòng lặp REPEAT luôn được thực hiện ít nhất một lần. Tuy nhiên các lệnh cho một vòng lặp REPEAT có vẻ ngắn hơn đôi chút bởi lẽ nó chỉ có một lệnh nhảy có điều kiện ở cuối trong khi vòng lặp WHILE có những hai: một lệnh nhảy có điều kiện ở đầu và lệnh JMP ở cuối.

6.5. Lập trình với các cấu trúc bậc cao.

Để chỉ rõ một chương trình có thể được phát triển từ các toán tử giả bậc cao thành các lệnh Hợp ngữ (Assembly) như thế nào, chúng ta hãy giải quyết vấn đề sau đây:

Yêu cầu:

Thông báo cho người sử dụng nhập vào một dòng văn bản. Hiển thị trên dòng tiếp theo chữ hoa đầu tiên và sau cùng tính theo thứ tự mã ASCII của chuỗi vừa nhập. Nếu không có chữ hoa nào được nhập vào thì hiển thị thông báo ‘Không có chữ hoa !’. Một ví dụ khi thực hiện chương trình:

Bạn hãy vào một dòng văn bản:

DONG CHAY THU CHUONG TRINH!

Chữ hoa đầu tiên = A Chữ hoa sau cùng = U

Để giải quyết vấn đề này chúng ta sẽ sử dụng phương pháp thiết kế chương trình top_down (từ trên xuống) mà bạn đã có thể gặp trong lập trình ngôn ngữ bậc cao. Trong phương pháp này, vấn đề nguyên thuỷ được chia thành một chuỗi các vấn đề con mà thực hiện mỗi trong chúng đơn giản hơn nhiều so với vấn đề ban đầu. Mỗi vấn đề con lại có thể được chia nhỏ hơn nữa ... Cứ tiếp tục như thế cho đến khi mỗi vấn đề con có thể được mã hoá trực tiếp. Việc sử dụng các chương trình con có thể phát triển phương pháp này

Sự phân chia đầu tiên:

1. Hiển thị thông báo ban đầu.
2. Đọc và xử lý dòng văn bản.

3. Hiển thị kết quả.

Bước 1. Hiển thị thông báo ban đầu.

Bước này có thể mã hoá ngay tức khắc:

```
MOV AH, 9          ; chức năng hiển thị chuỗi  
LEA DX, PROMPT   ; lấy thông báo ban đầu  
INT 21h           ; hiển thị nó
```

Thông báo có thể chứa trong đoạn dữ liệu như sau:

```
PROMPT DB 'Bạn hãy vào một dòng văn bản', 0Dh, 0Ah, '$'
```

Ta đưa vào cả ký tự xuống dòng và về đầu dòng để chuyển con trỏ xuống đầu dòng tiếp theo và như vậy người sử dụng có thể đánh vào toàn bộ một dòng văn bản.

Bước 2. Đọc và xử lý dòng văn bản.

Bước này thực hiện hầu hết các công việc của chương trình. Nó thực hiện nhập từ bàn phím, trả về các chữ cái thoả mãn (nó cũng đưa ra thông báo nếu không có chữ hoa nào được đọc vào). Sau đây là các bước thực hiện:

```
Đọc một ký tự  
WHILE không phải ký tự xuống dòng DO  
  IF ký tự là chữ hoa (*)  
    THEN  
      IF đứng trước chữ hoa đầu  
        THEN  
          chữ hoa đầu = ký tự  
        END_IF  
      IF đứng sau chữ hoa cuối  
        THEN  
          chữ hoa cuối = ký tự  
        END_IF  
    END_IF  
  đọc một ký tự  
END WHILE
```

Dòng (*) thực chất là một điều kiện AND:

```
IF ('A' <= ký tự) AND (ký tự <='Z')
```

Bước 2 có thể được mã hoá như sau:

```
MOV AH, 1           ; hàm đọc ký tự
INT 21h            ; ký tự trong AL
WHILE_:
; while không phải ký tự xuống dòng do
CMP AL, 0Dh        ; CR ?
JE END WHILE      ; đúng, thoát ra
;if ký tự là chữ hoa
CMP AL, 'A'         ; ký tự >= 'A' ?
JNGE END_IF        ; không phải chữ hoa
CMP AL, 'Z'         ; ký tự <= 'Z' ?
JNLE END_IF        ; không phải chữ hoa
;then
;if ký tự đứng trước chữ hoa đầu
CMP AL, FIRST       ; ký tự < FIRST ?
JNL CHECK_LAST      ; không, kiểm tra tiếp
;then chữ hoa đầu = ký tự
MOV FIRST, AL        ; FIRST = ký tự
;end_if
CHECK_LAST:
;if ký tự đứng sau chữ hoa cuối
CMP AL, LAST        ; ký tự > LAST ?
JNG END_IF          ; không, cho qua
;then chữ hoa cuối = ký tự
MOV LAST, AL         ; LAST = ký tự
;end_if
END_IF:
;đọc một ký tự
INT 21h             ; ký tự trong AL
JMP WHILE_           ; lặp lại
END WHILE:
```

Các biến FIRST và LAST phải được khởi tạo trước khi vào vòng lặp WHILE. Chúng có thể được định nghĩa trong đoạn dữ liệu như sau:

| | | |
|-------|----|-----|
| FIRST | DB | '' |
| LAST | DB | '@' |

Các giá trị khởi tạo '[' và '@' được chọn bởi vì '[' đứng sau 'Z' và '@' đứng trước 'A'. Theo cách này, chữ hoa đầu tiên được đưa vào sẽ làm thay đổi giá trị của cả hai biến.

Với bước 2 đã được mã hoá, chúng ta có thể tiếp tục đến bước cuối cùng.

Bước 3. Hiển thị kết quả:

```
IF không có chữ hoa
THEN
    hiển thị 'Không có chữ hoa ! '
ELSE
    hiển thị chữ hoa đầu và cuối.
END_IF
```

Bước này có thể hiển thị một trong hai thông báo: NOCAP_MSG nếu không có chữ hoa nào được đánh vào hay CAP_MSG nếu ngược lại. Chúng ta có thể khai báo chúng trong đoạn dữ liệu như sau:

| | |
|--------------|--------------------------|
| NOCAP_MSG DB | ' Không có chữ hoa ! \$' |
| CAP_MSG DB | ' Chữ hoa đầu tiên = ' |
| FIRST DB | '[' |
| | ' Chữ hoa sau cùng = ' |
| LAST DB | '@ \$ ' |

Khi CAP_MSG được hiển thị, nó sẽ đưa ra thông báo 'Chữ hoa đầu tiên = ', sau đó là giá trị của FIRST rồi đến 'Chữ hoa cuối cùng = ' và giá trị của LAST. Chúng ta đã sử dụng kỹ thuật này trong chương trình cuối cùng của chương 4.

Chương trình của chúng ta sẽ kiểm tra biến FIRST biết có chữ hoa nào được đọc vào hay không. Nếu FIRST chứa '[' là giá trị khởi đầu của nó thì có nghĩa là không đọc vào chữ hoa nào cả.

Bước 3 có thể được mã hoá như sau:

```
MOV AH, 9          ; hàm hiển thị chuỗi
;if không có chữ hoa
    CMP FIRST, '[' ; FIRST = '[' ?
    JNE CAPS        ; không, hiển thị kết quả
;then
    LEA DX, NOCAP_MSG
    JMP DISPLAY
CAPS:
```

```

    LEA    DX,CAP_MSG
DISPLAY:
    INT    21h          ; hiển thị thông báo
;end_if

```

Sau đây là chương trình đầy đủ:

Chương trình PGM6_2.ASM

```

TITLE      PGM6_2: Chữ hoa đầu và cuối.
.MODEL     SMALL
.STACK    100H
.DATA
PROMPT DB 'Bạn hãy vào một dòng văn bản',0Dh,0Ah,'$'
NOCAP_MSG DB 0DH,0AH,'Không có chữ hoa ! $'
CAP_MSG   DB 'Chữ hoa đầu tiên = '
FIRST     DB '['
                DB 'Chữ hoa sau cùng = '
LAST      DB '@ $ '
.CODE
MAIN      PROC
; khởi tạo DS
    MOV AX,@DATA
    MOV DS,AX
; hiển thị thông báo ban đầu
    MOV AH,9          ; chức năng hiển thị chuỗi
    LEA DX,PROMPT    ; lấy thông báo ban đầu
    INT 21h          ; hiển thị nó
; đọc và xử lý một dòng văn bản
    MOV AH,1          ; hàm đọc ký tự
    INT 21h          ; ký tự trong AL
WHILE_:
; while không phải ký tự xuống dòng do
    CMP AL,0Dh        ; CR ?
    JE END WHILE     ; đúng, thoát ra
; if ký tự là chữ hoa
    CMP AL,'A'        ; ký tự >= 'A' ?
    JNGE END_IF       ; không phải chữ hoa
    CMP AL,'Z'        ; ký tự <= 'Z' ?
    JNLE END_IF       ; không phải chữ hoa

```

```

;then
;if ký tự đứng trước chữ hoa đầu
    CMP    AL,FIRST      ; ký tự < FIRST ?
    JNL    CHECK_LAST   ; không, kiểm tra tiếp
;then  chữ hoa đầu = ký tự
    MOV    FIRST,AL      ; FIRST = ký tự
;end_if
CHECK_LAST:
;if ký tự đứng sau chữ hoa cuối
    CMP    AL,LAST       ; ký tự > LAST ?
    JNG    END_IF        ; không, cho qua
;then  chữ hoa cuối = ký tự
    MOV    LAST,AL       ; LAST = ký tự
;end_if
END_IF:
;đọc một ký tự
    INT    21h           ; ký tự trong AL
    JMP    WHILE_
END WHILE:
; hiển thị kết quả
    MOV    AH,9           ; hàm hiển thị chuỗi
;if không có chữ hoa
    CMP    FIRST,'['    ; FIRST = '[' ?
    JNE    CAPS          ; không, hiển thị kết quả
;then
    LEA    DX,NOCAP_MSG
    JMP    DISPLAY
CAPS:
    LEA    DX,CAP_MSG
DISPLAY:
    INT    21h           ; hiển thị thông báo
;end_if
;trở về DOS
    MOV    AH,4CH
    INT    21h
MAIN  ENDP
END  MAIN

```

TỔNG KẾT.

- ◆ Có hai loại lệnh nhảy: lệnh nhảy có điều kiện và lệnh nhảy không điều kiện. Lệnh nhảy có điều kiện được chia thành các lệnh nhảy có dấu, không dấu và các lệnh nhảy điều kiện đơn.
- ◆ Các lệnh nhảy có điều kiện được thực hiện dựa vào việc thiết lập các cờ trạng thái. Lệnh CMP (compare) chỉ dùng để thiết lập cờ trước các lệnh nhảy.
- ◆ Nhấn đích của lệnh nhảy có điều kiện phải đứng trước không quá 126 byte hoặc đứng sau không hơn 127 byte kể từ lệnh đó. Một lệnh JMP thường được dùng để nhảy qua giới hạn này.
- ◆ Trong một cấu trúc chọn lựa IF-THEN, nếu điều kiện kiểm tra là đúng, các dòng lệnh nhánh đúng sẽ được thực hiện. Trong trường hợp ngược lại, dòng lệnh theo sau cấu trúc được thực hiện.
- ◆ Trong một cấu trúc chọn lựa IF-THEN_ELSE, nếu điều kiện kiểm tra là đúng các dòng lệnh nhánh đúng sẽ được thực hiện. Trong trường hợp ngược lại các dòng lệnh nhánh sai sẽ được thực hiện. Cần phải có lệnh JMP ở cuối nhánh đúng để nhảy qua các dòng lệnh của nhánh sai.
- ◆ Trong một cấu trúc CASE, việc rẽ nhánh được điều khiển bởi một biểu thức. Các nhánh tương ứng với các giá trị có thể của biểu thức.
- ◆ Vòng lặp FOR được thực hiện với số lần lặp biết trước. Nó có thể được tạo lặp bởi lệnh LOOP. Trước khi vào vòng lặp, CX phải được khởi tạo bằng số lần lặp lại của vòng lặp.
- ◆ Trong một vòng lặp WHILE, điều kiện lặp được kiểm tra ở đầu vòng lặp. Các dòng lệnh của vòng lặp được lặp lại khi điều kiện đúng. Nếu điều kiện khởi đầu là sai, các dòng lệnh này sẽ không được thực hiện lần nào.
- ◆ Trong một vòng lặp REPEAT, điều kiện lặp được kiểm tra ở cuối vòng lặp. Các dòng lệnh của vòng lặp được lặp lại đến khi điều kiện đúng. Do điều kiện kiểm tra ở cuối vòng lặp nên các dòng lệnh của vòng lặp này sẽ được thực hiện ít nhất một lần.

Thuật ngữ tiếng Anh

| | |
|-----------------------------------|---|
| AND condition | Phép và lôgic hai điều kiện. |
| condition jump instruction | Lệnh nhảy có điều kiện. Một lệnh nhảy có điều kiện được thực hiện dựa vào việc thiết lập các cờ trạng thái. |
| loop | Vòng lặp. Một chuỗi các lệnh được lặp lại. |
| OR condition | Phép hoặc lôgic hai điều kiện. |
| signed jump | Nhảy có dấu. Lệnh nhảy có điều kiện dùng với các số có dấu. |
| single_flag jump | Nhảy điều kiện đơn. Lệnh nhảy có điều kiện thao tác dựa trên một cờ riêng biệt. |
| top_down program design | Thiết kế chương trình từ trên xuống. Khai triển chương trình bằng cách phân nhỏ một vấn đề lớn thành rất nhiều các vấn đề đơn giản hơn. |
| uncondition jump | Nhảy không điều kiện. Sự chuyển điều khiển không điều kiện. |
| unsigned jump | Nhảy không dấu. Lệnh nhảy có điều kiện dùng với các số không dấu. |

Các lệnh mới:

| | | |
|----------------|----------------|----------------|
| CMP | JCXZ | JLE/JNG |
| JA/JNBE | JE/JZ | JMP |
| JAE/JNB | JG/JNLE | JNC |
| JB/JNAE | JGE/JNL | JNE/JNZ |
| JC | JL/JNLE | LOOP |
| JBE/JNA | | |

Bài tập:

1. Viết các lệnh Hợp ngữ cho mỗi cấu trúc chọn lựa sau đây:

a. IF AX < 0

THEN

BX = -1

END_IF

b. IF AL < 0

THEN

AH = OFFh

ELSE

AH = 0

END_IF

c. Giả thiết DL chứa mã ASCII của một ký tự:

IF (DL >= 'A') AND (DL <= 'Z')

THEN

Hiển thị DL

END_IF

d. IF AX < BX

THEN

IF BX < CX

THEN

AX = 0

ELSE

BX = 0

END_IF

END_IF

e. IF (AX < BX) OR (BX < CX)

THEN

DX = 0

ELSE

DX = 1

END_IF

f. IF AX < BX

THEN

```

    AX = 0
    ELSE
        IF BX < CX
            THEN
                BX = 0
            ELSE
                CX = 0
        END_IF
    END_IF

```

2. Dùng cấu trúc CASE để mã hoá các công việc sau đây:

Đọc một ký tự.

Nếu là 'A', chuyển con trỏ về đầu dòng.

Nếu là 'B', chuyển con trỏ xuống dòng.

Nếu là một ký tự khác, trả về DOS.

3. Viết các lệnh thực hiện các công việc sau đây:

a. $AX = 1 + 4 + 7 + \dots + 148$.

b. $AX = 100 + 95 + \dots + 5$.

4. Dùng các lệnh LOOP thực hiện các công việc sau đây:

a. Nhập tổng của 50 phần tử đầu tiên của dãy số học: 1, 5, 9, 13 ... vào DX.

b. Đọc một ký tự và hiển thị nó 80 lần trên dòng tiếp theo.

c. Đọc một mật khẩu 5 ký tự và viết đè lên nó bằng cách trả về đầu dòng rồi hiển thị 5 chữ X. Bạn không cần phải lưu giữ lại các ký tự này..

5. Thuật toán sau đây có thể sử dụng để chia hai số dương bằng cách lặp lại phép trừ:

```

khởi động thương số bằng 0
WHILE số bị chia >= số chia DO
    tăng thương số
    trừ bớt số chia từ số bị chia
END WHILE

```

Hãy viết các lệnh thực hiện chia AX cho BX rồi cất thương trong CX.

6. Thuật toán sau đây có thể sử dụng để nhân hai số dương M và N bằng cách lặp lại phép cộng:

```
khởi động tích số bằng 0  
REPEAT  
    cộng M vào tích số  
    giảm N  
UNTIL N = 0
```

Hãy viết các lệnh thực hiện nhân AX với BX rồi cất tích trong CX. Bạn có thể bỏ qua trường hợp tràn.

7. Ta có thể tạo lập một vòng lặp điều khiển bởi biến đếm mà nó còn tiếp tục thi hành khi điều kiện được thỏa mãn. Các lệnh:

```
LOOPE nhän ; lặp khi bằng  
và  
LOOPZ nhän ; lặp khi bằng ZERO
```

sẽ làm giảm CX. Sau đó nếu CX $\neq 0$ và ZF = 1, điều khiển sẽ chuyển đến lệnh sau nhän đích. Nếu như CX = 0 hoặc ZF = 0, lệnh ở sau vòng lặp được thực hiện. Tương tự như vậy, các lệnh:

```
LOOPNE nhän ; lặp khi không bằng  
và  
LOOPNZ nhän ; lặp khi bằng khác ZERO
```

sẽ làm giảm CX. Sau đó nếu CX $\neq 0$ và ZF = 0, điều khiển sẽ chuyển đến lệnh sau nhän đích. Nếu như CX = 0 hoặc ZF = 1, lệnh ở sau vòng lặp được thực hiện.

a. Hãy sử dụng vòng lặp LOOPE viết các lệnh đọc các ký tự đến khi một ký tự khác ký tự trắng được đánh vào hoặc đã nhập đủ 80 ký tự.

b. Dùng vòng lặp LOOPNE viết các lệnh đọc các ký tự đến khi một ký tự về đầu dòng được đánh vào hoặc đã nhập đủ 80 ký tự.

Chương 7

CÁC LỆNH LOGIC, DỊCH VÀ QUAY

Tổng quan

Trong chương này chúng ta sẽ nghiên cứu về các chỉ thị dùng để thay đổi đến từng bit trong byte hay word. Khả năng thao tác với các bit thường không có ở trong các ngôn ngữ bậc cao (trừ ngôn ngữ C), và đó cũng là một lý do quan trọng để lập trình bằng Hợp ngữ .

Trong phần 7.1 chúng tôi sẽ giới thiệu các lệnh lôgic AND, OR, XOR và NOT. Các lệnh này có thể sử dụng để xoá, thiết lập và kiểm tra từng bit trong các thanh ghi hay các biến. Chúng ta sẽ sử dụng các lệnh này để thực hiện một số công việc đã quen thuộc như đổi chữ thường thành chữ hoa hay còn mới như xác định xem một thanh ghi chứa số chẵn hay lẻ.

Phần 7.2 sẽ trình bày về các lệnh dịch, các bit có thể được dịch phải hoặc trái trong thanh ghi hay trong một ô nhớ. Khi một bit bị dịch ra khỏi thanh ghi nó sẽ được chứa trong cờ CF. Vì dịch trái cũng có nghĩa là nhân đôi số và dịch phải có nghĩa là chia đôi nó, lệnh này cho phép chúng ta thực hiện phép nhân và chia cho một luỹ thừa của 2. Trong chương 9 chúng ta sẽ sử dụng các lệnh MUL và DIV để thực hiện các phép nhân và chia một cách tổng quát hơn, tuy nhiên các lệnh này chậm hơn nhiều so với các lệnh dịch.

Trong phần 7.3 chúng tôi sẽ giới thiệu các lệnh quay. Các lệnh này làm việc giống như các lệnh dịch ngoại trừ một điều là khi một bit bị ra khỏi một phía của toán hạng nó sẽ được đưa trở về phía kia của toán hạng đó. Các lệnh này có thể được sử dụng trong các trường hợp chúng ta muốn kiểm tra hoặc thay đổi các bit hay nhóm các bit.

Trong phần 7.4 chúng ta sẽ sử dụng các lệnh lôgic, dịch và lệnh quay để thực hiện các thao tác vào ra với số nhị phân và số hex. Khả năng đọc và viết các số cho phép chúng ta giải quyết rất nhiều vấn đề khác nhau.

7.1 Các lệnh lôgic

Như đã nói ở trên khả năng thao tác với từng bit riêng biệt là một trong những ưu điểm của hợp ngũ. Chúng ta có thể thay đổi từng bit trong máy tính bằng các lệnh lôgic. Các giá trị nhị phân 0 và 1 được xem như là các giá trị lôgic TRUE hoặc FALSE một cách tương ứng. Bảng 7.1 là bảng sự thật của các toán tử lôgic AND, OR, XOR(hoặc phủ định- exclusive OR) và NOT.

Khi lệnh lôgic được áp dụng với các toán hạng 8 và 16 bit kết quả nhận được bằng cách áp dụng chúng với từng bit một.

Ví dụ 7.1 Thực hiện các thao tác lôgic sau đây:

1. 10101010 AND 11110000
2. 10101010 OR 11110000
3. 10101010 XOR 11110000
4. NOT 10101010

Trả lời :

1.

$$\begin{array}{r} 10101010 \\ \text{AND } 11110000 \\ \hline = 10100000 \end{array}$$

2.

$$\begin{array}{r} 10101010 \\ \text{OR } 11110000 \\ \hline = 11111010 \end{array}$$

3.

$$\begin{array}{r} 10101010 \\ \text{XOR } 11110000 \\ \hline = 01011010 \end{array}$$

4.

$$\begin{array}{r} 10101010 \\ \text{NOT } 01010101 \\ \hline \end{array}$$

| a | b | a AND b | a OR b | a XOR b |
|---|---|---------|--------|---------|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

| a | NOT a |
|---|-------|
| 0 | 1 |
| 1 | 0 |

Bảng 7.1 Bảng sự thật của các lệnh lôgic AND, OR, XOR và NOT
(0 = TRUE, 1 = FALSE)

7.1.1 Các lệnh AND, OR và XOR

Các lệnh AND, OR và XOR thực hiện các thao tác lôgic như tên gọi của chúng. Cú pháp như sau:

| | |
|-----|-------------|
| AND | Đích, nguồn |
| OR | Đích, nguồn |
| XOR | Đích, nguồn |

Kết quả của thao tác được chứa trong toán hạng đích, nó phải là một thanh ghi hay một ô nhớ. Toán hạng nguồn có thể là hằng số, thanh ghi hay ô nhớ. Tuy nhiên các thao tác giữa 2 ô nhớ là không hợp lệ.

Ảnh hưởng tới cờ:

SF, ZF, PF phản ảnh kết quả của lệnh
AF không xác định
CF, OF = 0

Khi sử dụng các lệnh AND, OR và XOR chúng ta có thể thay đổi một cách có chọn lọc các bit của toán hạng đích. Để làm điều đó chúng ta tạo lên một mảng bit được gọi là mặt nạ (MASK). Các bit của mặt nạ được chọn để sao cho các bit tương ứng của toán hạng đích được thay đổi đúng như chúng ta mong muốn khi lệnh lôgic được thực hiện.

Để chọn các bit mặt nạ chúng ta sử dụng các tính chất sau đây của các lệnh AND, OR và XOR. Từ hình 7.1 chúng ta thấy, nếu b biểu diễn một bit (0 hoặc 1) thì:

$$\begin{aligned} b \text{ AND } 1 &= b, b \text{ OR } 0 = b, b \text{ XOR } 0 = b \\ b \text{ AND } 1 &= b, b \text{ OR } 1 = 1, b \text{ XOR } 1 = \sim b \end{aligned}$$

(bù của b)

Từ đó chúng ta rút ra kết luận sau:

1. Lệnh AND có thể sử dụng để xoá các bit nhất định của toán hạng đích trong khi giữ nguyên những bit còn lại. Bit 0 của mặt nạ xoá bit tương ứng, còn bit 1 của mặt nạ giữ nguyên bit tương ứng của toán hạng đích.
2. Lệnh OR có thể được dùng để thiết lập các bit xác định của toán hạng đích trong khi vẫn giữ nguyên những bit còn lại. Bit 1 của mặt nạ sẽ thiết lập bit tương ứng trong khi bit 0 của nó sẽ giữ nguyên bit tương ứng của toán hạng đích.
3. Lệnh XOR có thể dùng để đảo các bit xác định của toán hạng đích trong khi vẫn giữ nguyên những bit còn lại. Bit 1 của mặt nạ làm đảo bit tương ứng còn bit 0 giữ nguyên bit tương ứng của toán hạng đích.

Ví dụ 7.2 Xoá bit dấu của AL trong khi giữ nguyên các bit còn lại.

Trả lời:

Sử dụng lệnh AND với $0111111b = 7Fh$ làm mặt nạ : AND AL, 7Fh

Ví dụ 7.3 Thiết lập các bit trọng lượng cao nhất và thấp nhất của AL trong khi giữ nguyên các bit còn lại

Trả lời :

Sử dụng lệnh OR với $10000001b = 81h$ làm mặt nạ : OR AL, 81h

Ví dụ 7.4 Thay đổi bit dấu của DX

Trả lời :

Sử dụng lệnh XOR với mặt nạ 8000h chúng ta có: XOR DX, 8000h

*** Chú ý !:** Để tránh nhầm lẫn khi viết, tốt nhất các bạn nên biểu diễn mặt nạ dưới dạng số hex thay vì dưới dạng số nhị phân đặc biệt là khi sử dụng mặt nạ dài 16 bit.

Các lệnh logic đặc biệt có ích khi thực hiện các công việc dưới đây:

Đổi mã ASCII của một số thành số tương ứng.

Chúng ta đã thấy rằng khi một chương trình đọc một ký tự từ bàn phím, AL sẽ chứa mã ASCII của ký tự đó. Điều đó cũng đúng với các ký tự biểu diễn số. Chẳng hạn khi phím "5" được ấn, AL sẽ chứa 35h thay vì 5. Để nhận được 5 trong thanh ghi AL chúng ta có thể làm như sau:

```
SUB AL, 30h
```

Một phương pháp khác là sử dụng lệnh AND để xoá nửa byte cao của AL:

```
AND AL, 0Fh
```

Vì mã ASCII của các chữ số từ "0" đến "9" là từ 30h đến 39h, phương pháp này có thể dùng để đổi mã ASCII của bất cứ chữ số nào thành giá trị thập phân tương ứng.

Bằng cách sử dụng lệnh AND thay cho lệnh SUB chúng ta nhấn mạnh rằng chúng đang thay đổi những mẫu bit của AL, nhờ đó chương trình trở nên dễ đọc hơn.

Việc đổi ngược lại một số thập phân ra mã ASCII của nó để giành cho các bạn như là một bài tập.

Đổi chữ thường thành chữ hoa

Mã ASCII của các chữ thường từ "a" đến "z" nằm từ 61h đến 7Ah, mã ASCII của các chữ hoa từ "A" đến "Z" nằm từ 41h đến 5Ah. Bởi vậy chẳng hạn nếu DL chứa mã ASCII của một chữ thường chúng ta có thể đổi ra chữ hoa bằng cách thực hiện lệnh:

```
SUB DL, 20h
```

Phương pháp này được sử dụng trong chương 4, tuy nhiên nếu chúng ta so sánh mã ASCII dạng nhị phân của các chữ thường và chữ hoa tương ứng sẽ thấy rằng:

| Ký tự | Mã ASCII | Ký tự | Mã ASCII |
|-------|----------|-------|----------|
| a | 01100001 | A | 01000001 |
| b | 01100010 | B | 01000010 |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| z | 01111010 | Z | 01011010 |

Rõ ràng là để đổi từ chữ thường thành chữ hoa chúng ta chỉ cần xoá bit 5, điều này có thể thực hiện bằng cách dùng lệnh AND với mặt nạ 11011111b hay 0DFh. Do đó nếu một chữ thường chứa trong DL thì ta có thể đổi nó thành chữ in như sau:

AND DL, 0DFh

Việc đổi ngược lại từ chữ hoa ra chữ thường xem như bài tập cho các bạn.

Xoá một thanh ghi

Chúng ta đã biết 2 cách để xoá một thanh ghi, chẳng hạn để xoá thanh ghi AX chúng ta có thể làm như sau:

MOV AX, 0

hay

SUB AX, AX

Bằng cách sử dụng một tính chất là 1 XOR 1 = 0 và 0 XOR 0 = 0 chúng ta có một phương pháp thứ 3 như sau:

XOR AX, AX

Mã máy cho lệnh đầu tiên là 3 byte so với 2 byte của mỗi lệnh sau, như vậy các lệnh sau hiệu quả hơn. Tuy nhiên các thao tác giữa ô nhớ với ô nhớ là không hợp lệ nên khi cần xoá một ô nhớ thì chúng ta bắt buộc phải sử dụng lệnh thứ nhất.

Kiểm tra xem một thanh ghi có bằng 0 hay không

Vì 1 OR 1 = 1 và 0 OR 0 = 0 có vẻ như chúng ta sẽ phí thời gian nếu thực hiện lệnh sau:

OR CX, CX

Bởi vì chúng không làm thay đổi nội dung của CX. Tuy nhiên chúng lại tác động đến cờ ZF và SF do đó trong trường hợp đặc biệt khi CX chứa 0 thì ZF = 1 và vì vậy nó có thể được dùng thay cho lệnh :

CMP CX, 0

Để kiểm tra xem nội dung của một thanh ghi có bằng 0 hay không hoặc để kiểm tra dấu của số chứa trong thanh ghi.

7.1.2 Lệnh NOT

Lệnh NOT lấy số bù 1 của một toán hạng đích. Cú pháp như sau:

NOT toán hạng đích

Lệnh này không làm ảnh hưởng tới các cờ.

Ví dụ 7.5 Đảo các bit trong thanh ghi AX

Trả lời :

Dùng lệnh NOT:

NOT AX

7.1.3 Lệnh TEST

Lệnh TEST thực hiện thao tác và lôgic giữa toán hạng đích với nguồn nhưng không làm thay đổi toán hạng đích. Mục đích của lệnh TEST là thiết lập các cờ. Cú pháp :

TEST toán hạng đích, toán hạng nguồn

Các cờ bị tác động:

SF, ZF, PF phản ánh kết quả

AF không xác định

CF, OF = 0

Kiểm tra các bit

Lệnh TEST có thể sử dụng để kiểm tra các bit riêng biệt trong một toán hạng. Một nă có giá trị 1 ở các bit tương ứng với với các bit cần kiểm tra trong toán hạng đích và giá trị 0 ở các bit khác. Vì 1 AND b = b và 0 AND b = 0 kết quả của lệnh:

TEST toán hạng đích, mặt nạ

Sẽ có giá trị 1 ở các bit đã chọn khi và chỉ khi toán hạng đích cũng có giá trị 1 ở các vị trí đó, còn lại các bit khác có giá trị 0. Nếu toán hạng đích có giá trị 0 ở tất cả các bit được kiểm tra thì kết quả sẽ bằng 0 và cờ ZF được thiết lập 1.

Ví dụ 7.6 Viết đoạn lệnh thực hiện lệnh nhảy đến nhãn BELOW nếu AL chứa số chẵn.

Trả lời :

Vì các số chẵn có bit 0 bằng 0 nên mặt nạ là 00000001b = 1. Chúng ta có:

| | |
|------------|---------------------------|
| TEST AL, 1 | ; AL chứa số chẵn ? |
| JZ BELOW | ; Đúng ! , nhảy đến BELOW |

7.2 Các lệnh dịch

Các lệnh dịch và quay dịch các bit trong toán hạng đích sang trái hoặc phải một hoặc một số vị trí. Đối với lệnh dịch các bit bị dịch ra khỏi toán hạng sẽ bị mất, còn đối với lệnh quay các bit dịch ra khỏi một phía của toán hạng đích sẽ được đưa trở lại phía kia. Các lệnh này có 2 dạng sau :

Khi muốn dịch hoặc quay 1 vị trí chúng ta có:

Mã lệnh toán hạng đích, 1

Khi muốn dịch hoặc quay N vị trí chúng ta có:

Mã lệnh toán hạng đích, CL

Trong đó CL chứa N. Trong cả 2 trường hợp toán hạng đích là một thanh ghi 8 hoặc 16 bit hay là một ô nhớ. Chú ý rằng trong các bộ vi xử lý tiên tiến của INTEL, các lệnh quay hoặc dịch cho phép sử dụng các hàng số 8 bit. Như chúng ta sẽ thấy ngay sau đây các lệnh này có thể dùng để nhân hoặc chia cho luỹ thừa của 2, và chúng ta sẽ sử dụng chúng trong các chương trình vào ra với số nhị phân và số hex.

7.2.1 Các lệnh dịch trái

Lệnh SHL

Lệnh SHL (shift left) dịch các bit của toán hạng đích sang bên trái. Cú pháp

cho lệnh dịch một vị trí như sau:

SHL toán hạng dịch, 1

Một giá trị 0 sẽ được đưa vào vị trí bên phải nhất của toán hạng đích, còn bit msb của nó sẽ được đưa vào cờ CF (xem hình 7.2)

Nếu số đếm vị trí đích N khác 1 thì lệnh có dạng sau:

SHL toán hạng đích, CL

Trong đó CL chứa N. Trong trường hợp này N phép dịch trái đơn được thực hiện. Giá trị của CL vẫn giữ nguyên không thay đổi khi lệnh thực hiện xong.

Các cờ bị tác động:

SF, PF, ZF phản ánh kết quả

AF không xác định

CF chứa bit cuối cùng được dịch ra khỏi toán hạng

OF bằng 1 nếu kết quả bị thay đổi dấu trong phép dịch cuối cùng.

Ví dụ 7.7 Giả sử DH chứa 8Ah và CL chứa 3, cho biết giá trị của DH và CF sau khi lệnh: SHL DH , CL được thực hiện

Trả lời :

Giá trị nhị phân trong DH là 10001010. Sau 3 lần dịch CF sẽ chứa giá trị 0 còn nội dung của DH có thể nhận được bằng cách xoá đi 3 bit bên trái nhất của DH và thêm 3 bit 0 vào bên 0 phải nó. Bởi vậy chúng ta nhận được kết quả :01010000b = 50h.

Thực hiện phép nhân bằng cách dịch trái

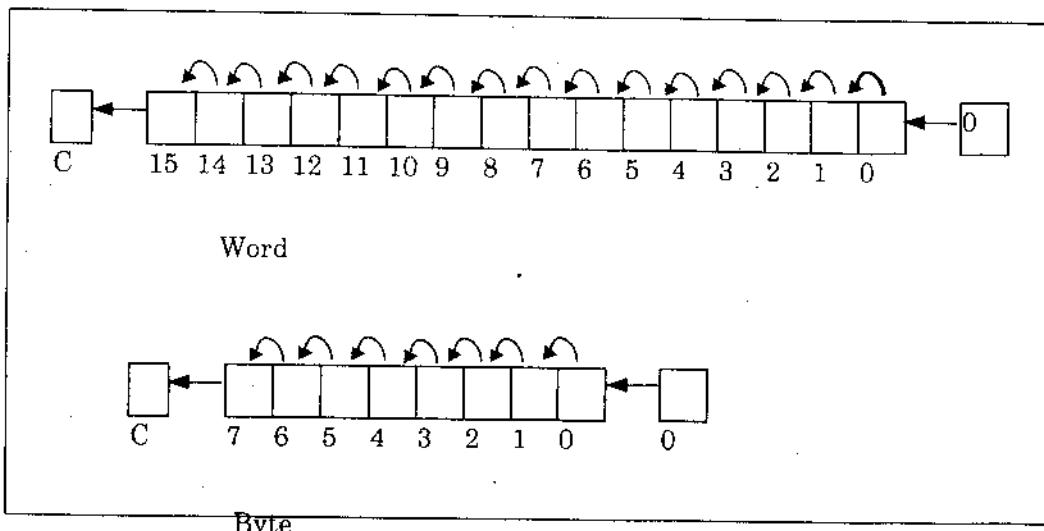
Giả sử có số thập phân 235, nếu mỗi chữ số của nó được dịch sang bên trái một vị trí và thêm vào bên phải chữ số 0 thì ta nhận được 2350 có nghĩa là bằng 235 nhân với 10.

Tương tự như vậy việc dịch trái một số nhị phân cũng như nhân nó với 2. Ví dụ AL chứa 5 = 00000101b. Sau khi dịch trái một bit chúng ta có : 00001010b = 10d nghĩa là bằng 2 giá trị của nó. Tiếp tục dịch trái chúng ta nhận được 00010100b = 20d nghĩa là lại nhân đôi nó một lần nữa.

Lệnh SAL

Như vậy lệnh SHL có thể dùng để nhân một toán hạng với luỹ thừa của 2. Tuy nhiên để làm nổi bật bản chất số học của thao tác, lệnh SAL (shift arithmetic left) thường được sử dụng để thay cho việc nhân các số. Cả 2 lệnh đều tạo ra cùng một mã máy.

Các số âm cũng có thể được nhân 2 bằng cách dịch trái. Ví dụ nếu AX chứa FFFFh (-1) chúng ta dịch trái nó 3 bit thì sẽ nhận được AX = FFF8h (-8).



Hình 7.2 Lệnh SHL và SAL

Sự tràn

Khi chúng ta sử dụng lệnh dịch trái để làm phép nhân hiện tượng tràn có thể xảy ra. Với lệnh dịch trái một bit CF và OF tương ứng chỉ ra một cách chính xác sự tràn không dấu và có dấu. Tuy nhiên cờ tràn không còn đáng tin cậy trong lệnh dịch trái nhiều bit. Sở dĩ như vậy là vì lệnh dịch nhiều bit thực ra là nhiều lệnh dịch một bit, và cờ CF, OF chỉ phản ánh kết quả của sự dịch cuối cùng. Ví dụ khi BL chứa 80h, CL chứa 2 và chúng ta thực hiện lệnh SHL

BL,CL thì CF = OF = 0 mặc dù cả 2 hiện tượng tràn có dấu và không dấu đều xảy ra.

Ví dụ 7.8 Viết một đoạn lệnh thực hiện phép nhân AX với 8, giả sử hiện tượng tràn không xảy ra.

Trả lời:

Để thực hiện phép nhân với 8 chúng ta cần thực hiện 3 lần dịch trái:

MOV CL, 3 ; Số lần dịch

SHL AX, CL ; Nhân với 8

7.2.2 Các lệnh dịch phải

Lệnh SHR

Lệnh SHR (shift Right) dịch các bit của toán hạng đích sang bên phải. Cú pháp cho lệnh dịch một vị trí như sau:

SHR toán hạng đích, 1

Một giá trị 0 sẽ được đưa vào bit msb của toán hạng đích, còn bit bên phải nhất của nó (lsb) sẽ được đưa vào cờ CF (xem hình 7.3)

Nếu số đếm vị trí dịch N khác 1 thì lệnh có dạng sau:

SHL toán hạng đích, CL

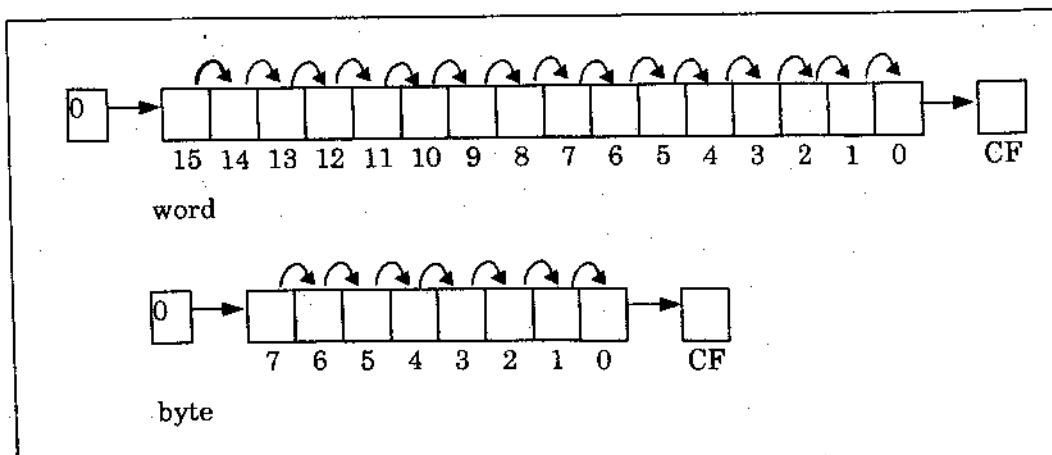
Trong đó CL chứa N. Trong trường hợp này N phép dịch phải đơn được thực hiện. Giá trị của CL vẫn giữ nguyên không thay đổi khi lệnh thực hiện xong.

Các cờ bị tác động cũng giống như trong trường hợp dịch trái

Ví dụ 7.9 Giả sử DH chứa 8Ah và CL chứa 2, cho biết giá trị của DH và CF sau khi lệnh: SHL DH, CL được thực hiện

Trả lời :

Giá trị nhị phân trong DH là 10001010. Sau 2 lần dịch CF sẽ chứa giá trị 1 còn nội dung của DH có thể nhận được bằng cách xoá đi 2 bit bên phải nhất của DH và thêm 2 bit 0 vào bên 0 phải nó. Bởi vậy chúng ta nhận được kết quả :00100010b = 22h.



Hình 7.3 Lệnh SHR

Lệnh SAR

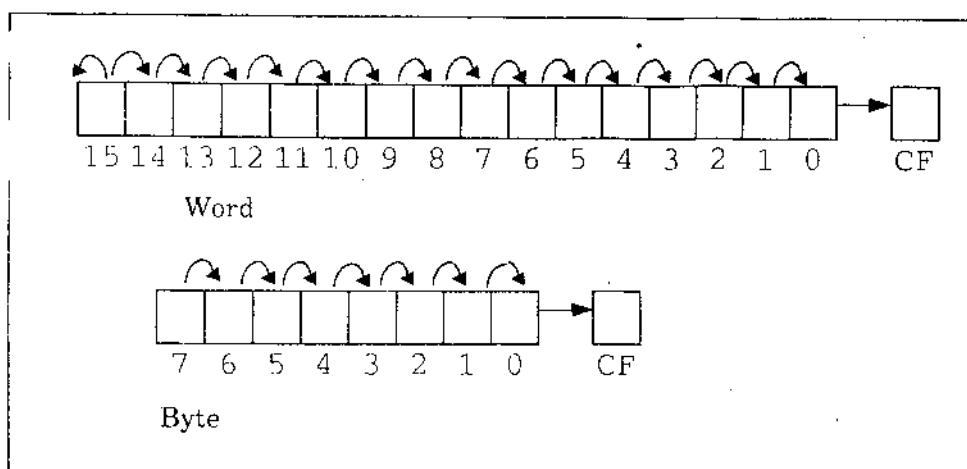
Lệnh SAR (shift arithmetic right) làm việc gần giống lệnh SHR với một điểm khác biệt là bit msb của toán hạng đích giữ nguyên giá trị ban đầu của nó. (xem hình 7.4). Cú pháp như sau:

SAR toán hạng đích,1

Và

SAR Toán hạng đích,CL

Các cờ bị tác động giống như trong lệnh SHR



Hình 7.4 Lệnh SAR

Thực hiện phép chia bằng cách dịch phải

Vì việc dịch trái nhân đôi giá trị của toán hạng đích cho nên cũng có lý khi đoán rằng việc dịch phải sẽ chia đôi nó. Điều đó đúng cho các số chẵn, đối với các số lẻ, dịch phải sẽ chia đôi nó và làm tròn xuống số nguyên gần nhất. Ví dụ nếu BL chứa

$00000101b = 5$ thì sau khi dịch phải BL sẽ chứa $0000010 = 2$.

Phép chia không dấu và có dấu

Trong phép chia bằng cách dịch phải chúng ta phải phân biệt 2 trường hợp đối với các số không dấu và có dấu. Nếu đang làm việc với các số không dấu chúng ta phải sử dụng lệnh SHR còn khi làm việc với số có dấu chúng ta phải

sử dụng lệnh SAR vì lệnh này giữ nguyên dấu.

Ví dụ 7.10 Sử dụng các phép dịch phải để thực hiện phép chia số không dấu 65143 cho 4, thương số chứa trong AX.

Trả lời :

Để chia cho 4 chúng ta cần dịch phải 2 lần, do số bị chia là số không dấu chúng ta sẽ sử dụng lệnh SHR. Đoạn lệnh như sau:

```
MOV AX, 65143 ;AX chứa số bị chia  
MOV CL, 2      ;CX chứa số lần dịch phải  
SHR AX, CL     ;Chia cho 4
```

Ví dụ 7.11 Giả sử AL chứa -15, hãy cho biết giá trị thập phân của AL sau khi thực hiện lệnh SAR AL,1.

Trả lời :

Lệnh SAR chia số cho 2 và làm tròn xuống. Chia -15 cho 2 chúng ta nhận được -7,5 sau khi làm tròn chúng ta nhận được -8. Dưới dạng số nhị phân chúng ta có $-15 = 11110001_b$. Sau khi dịch phải chúng ta có $11111000_b = -8$.

Các phép nhân và chia tổng quát hơn

Chúng ta đã thấy rằng việc nhân và chia cho luỹ thừa của 2 có thể thực hiện bằng các lệnh dịch trái và phải. Để nhân với một số bất kỳ như 10d chúng ta có thể kết hợp các lệnh dịch và cộng (xem chương 8).

Trong chương 9 chúng ta sẽ học các lệnh MUL và IMUL, DIV và IDIV. Chúng không bị giới hạn trong việc nhân hay chia các luỹ thừa của 2, nhưng lại chậm hơn nhiều so với các lệnh dịch

7.3 Các lệnh quay

Lệnh quay trái

Lệnh ROL (rotate left) dịch các bit sang bên trái. Bit msb được dịch vào bit bên phải nhất, đồng thời được đưa vào cờ CF. Các bạn có thể tưởng tượng các bit của toán hạng dịch tạo thành một vòng tròn với bit lsb theo sau bit msb, xem hình 7.5. Cú pháp :

ROL toán hạng dịch, 1

Và

ROL toán hạng dịch, CL

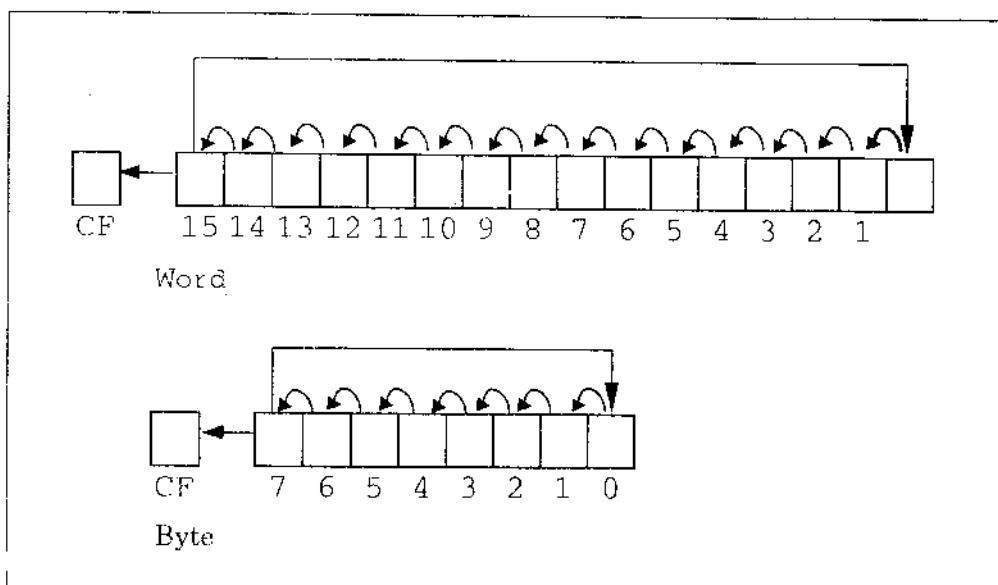
Lệnh quay phải

Lệnh ROR (rotate right) làm việc giống như lệnh ROL trừ một điểm khác biệt là các bit được dịch sang phải, bit bên phải nhất được dịch vào bit msb đồng thời cũng được đưa vào cờ CF. Cú pháp của lệnh:

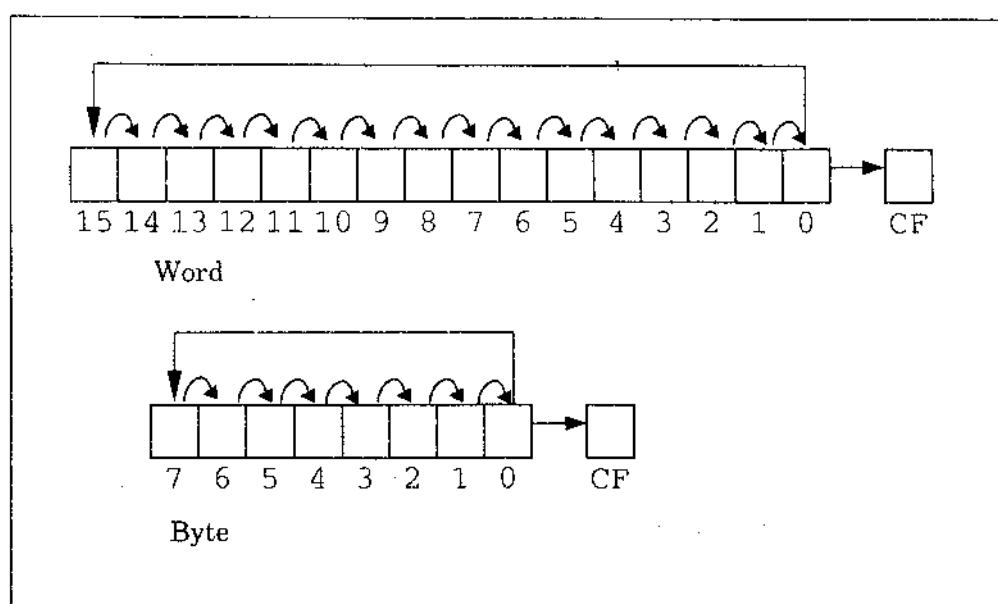
ROR toán hạng đích, 1

Và

ROR toán hạng đích, CL



Hình 7.5 Lệnh ROL



Hình 7.6 Lệnh ROR

Trong các lệnh ROL,ROR cờ CF chưa bit bị dịch ra khỏi toán hạng. Ví dụ sau sẽ trình bày một phương pháp để xác định các bit trong một byte hay word mà không làm thay đổi nội dung của chúng.

Ví dụ 7.12 Sử dụng lệnh ROL để đếm số bit 1 trong thanh ghi BX mà không làm thay đổi nội dung của nó, chứa kết quả trong AX.

Lời giải:

```
XOR  AX, AX      ;AX đếm số bit
MOV  CX, 16      ;Biến đếm vòng lặp
TOP:
    ROL  BX, 1      ;CF chưa bit bị đưa ra
    JNC  NEXT       ;bit 0 ?
    INC  AX         ;không phải !, tăng số bit 1
NEXT:
    LOOP TOP        ;lặp lại cho đến khi làm xong
```

Trong ví dụ trên đây, chúng ta đã sử dụng lệnh JNC (Jump if No Carry), lệnh này thực hiện việc nhảy nếu CF = 0. Trong phần 7.4 chúng ta sẽ sử dụng để đưa ra nội dung của một thanh ghi dưới dạng nhị phân.

Lệnh quay trái qua cờ nhớ

Lệnh RCL (Rotate Carry Left) dịch các bit của toán hạng đích sang trái. Bit msb được đặt vào cờ CF, giá trị cũ của CF được đưa vào bit phải nhất của toán hạng đích. Nói cách khác RCL làm việc giống như ROL ngoại trừ một điều là cờ CF cũng là một phần của vòng tròn tạo lên bởi các bit đang được quay (xem hình 7.7), cú pháp của lệnh:

RCL toán hạng đích,1

Và

RCL toán hạng đích,CL

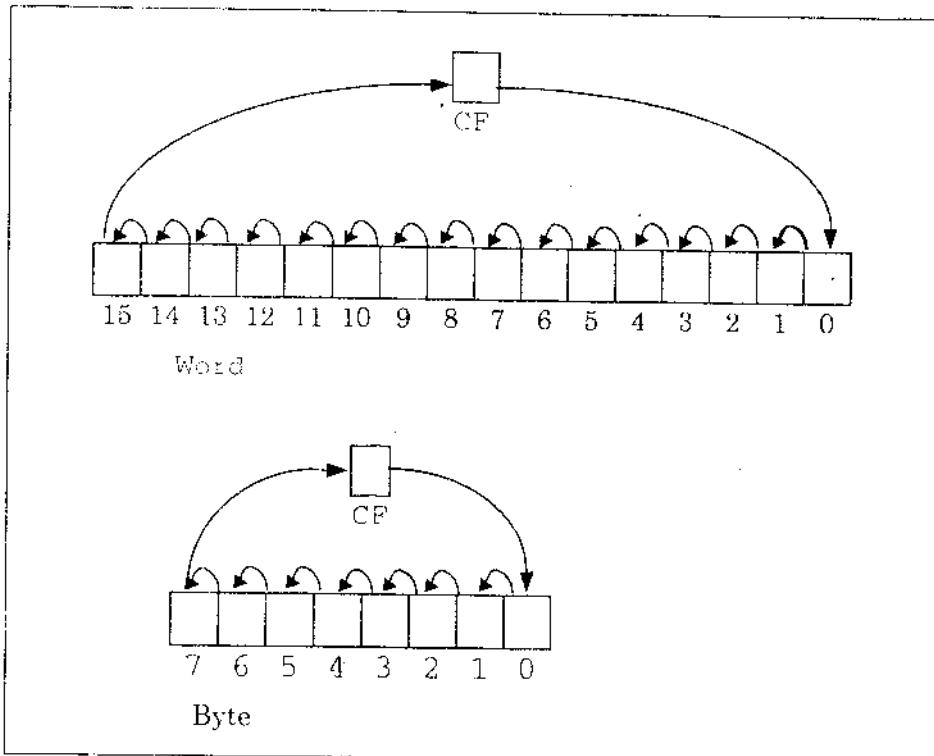
Lệnh quay phải qua cờ nhớ

Lệnh RCR (Rotate Carry Right) hoạt động giống như lệnh RCL nhưng các bit được quay sang phải (xem hình 7.8), cú pháp của lệnh như sau:

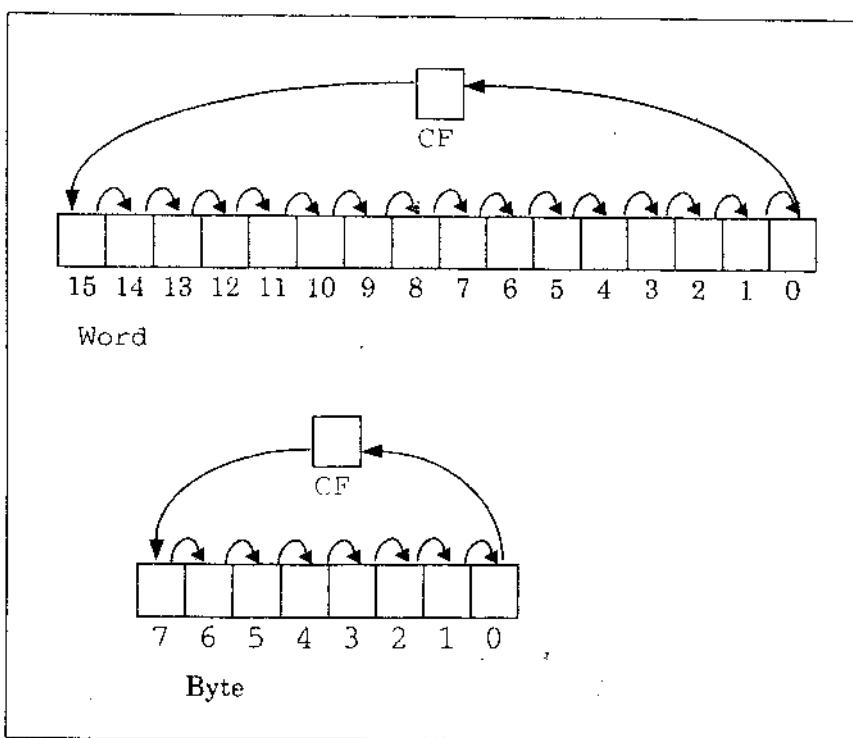
RCR toán hạng đích,1

Và

RCR toán hạng đích,CL



Hình 7.7 Lệnh RCL



Hình 7.8 Lệnh RCR

Ví dụ 7.13 Giả sử DH chứa 8Ah, CF = 1, và CL chứa 3. Cho biết nội dung của DH và CF khi thực hiện lệnh RCR DH,CL ?

Trả lời :

| | CF | DH |
|-----------------|----|----------|
| Các giá trị đầu | 1 | 10001010 |
| Sau 1 lần dịch | 0 | 11000101 |
| Sau 2 lần dịch | 1 | 01100010 |
| Sau 3 lần dịch | 0 | 10110001 |

Như vậy sau khi thực hiện lệnh DH chứa 10110001b = B1h

Tác động của các lệnh quay đến cờ

SF, PF, ZF phản ánh kết quả

AF không xác định

CF = bit cuối cùng bị dịch ra khỏi toán hạng

OF = 1 nếu kết quả đổi dấu trong lần quay cuối cùng.

Một ứng dụng: Đảo các mẫu bit

Chúng ta hãy xem xét vấn đề đảo các mẫu bit trong một byte hoặc một word để làm một ví dụ về ứng dụng của các lệnh quay và dịch. Chẳng hạn AL chứa 11011100 chúng ta muốn đổi lại thành 00111011. Một phương pháp đơn giản là sử dụng lệnh SHL để dịch các bit ra khỏi đầu bên trái của AL rồi dùng lệnh RCR để đưa nó vào đầu bên trái của một thanh ghi khác như BL chẳng hạn. Sau khi lặp lại công việc đó 8 lần BL sẽ chứa các mẫu bit đã đảo ngược của AL và có thể chép lại nội dung của nó vào AL. Đoạn lệnh như sau:

```
MOV CX, 8      ; Số lần cần thực hiện
REVERSE:
    SHL AL, 1    ; Lấy bit vào CF
    RCR BL, 1    ; Quay để đưa nó vào BL
    LOOP REVERSE ; lặp cho đến khi DO xong
    MOV AL, BL    ; AL chứa mẫu bit đảo ngược
```

7.4 Vào ra với các số nhị phân và số hex.

Một ứng dụng khá tiện lợi của các lệnh quay và dịch là thực hiện các thao tác

vào/ra với các số nhị phân và số hex.

Nhập các số nhị phân

Để nhập các số nhị phân chúng ta giả thiết rằng chương trình đọc vào các số nhị phân từ bàn phím kết thúc bằng phím ENTER. Các số thực sự ở dạng chuỗi các chữ số 0 và 1. Khi mỗi ký tự được nhập chúng ta phải đổi chúng ra giá trị của từng bit rồi kết hợp các bit vào trong thanh ghi. Thuật toán sau thực hiện việc đọc một số nhị phân từ bàn phím rồi sau đó lưu nó vào thanh ghi BX.

Thuật toán nhập số nhị phân

```
Xoá BX /* BX sẽ giữ kết quả */
Nhập một ký tự /* '0' hoặc '1' */
WHILE ký tự <> CR DO
    Đổi ký tự ra giá trị nhị phân
    Dịch trái BX
    Chèn giá trị nhận được vào bit lsb của BX
    Nhập ký tự
END WHILE
```

Biểu diễn với việc nhập 110

```
Xoá BX
BX=0000 0000 0000 0000
Nhập vào ký tự '1', đổi nó thành 1
Dịch trái BX
BX=0000 0000 0000 0000
Chèn giá trị nhận được vào bit lsb của BX
BX=0000 0000 0000 0001
Nhập vào ký tự '1', đổi nó thành 1
Dịch trái BX
BX=0000 0000 0000 0010
Chèn giá trị nhận được vào bit lsb của BX
BX=0000 0000 0000 0011
Nhập vào ký tự '0', đổi nó thành 0
Dịch trái BX
BX=0000 0000 0000 0110
Chèn giá trị nhận được vào bit lsb của BX
BX=0000 0000 0000 0110
BX chứa 110b
```

Thuật toán trên đây đã coi rằng các ký tự nhập vào chỉ là một trong 3 ký tự '1', '0' hay CR và có nhiều nhất 16 ký tự được nhập. Khi một chữ số mới được nhập vào, các bit có sẵn trong thanh ghi BX phải được dịch trái để giành chỗ cho nó.

Do đó có thể dùng lệnh OR để chèn bit mới vào BX. Các lệnh hợp ngữ như sau:

```
XOR    BX, BX      ;Xoá BX
MOV    AH, 1 ;Rờm con đọc ký tự từ bàn phím
INT    21H       ;Đọc ký tự
WHILE_:
CMP    AL, 0DH     ;CR?
JE     END_WHILE ;Đúng !, kết thúc
AND    AL, 0FH ;Không đổi ra giá trị nhị phân
SHL    BX, 1      ;Giành chỗ cho giá trị mới
OR     BL, AL     ;Chèn giá trị mới vào BX
INT    21H       ;Đọc tiếp ký tự
JMP    WHILE_    ;Lặp lại
END_WHILE:
```

Việc xuất các giá trị nhị phân

Việc đưa ra nội dung của BX dưới dạng nhị phân cũng dùng các lệnh dịch. Dưới đây chúng tôi chỉ đưa ra thuật toán, còn phần chương trình bằng hợp ngữ giành cho các bạn làm bài tập.

Thuật toán đưa ra số nhị phân

FOR 16 lần, DO:

```
Quay trái BX /* BX chứa giá trị đưa ra, bit msb đưa
               vào CF */
IF CF=1
  THEN
    Đưa ra '1'
ELSE
    Đưa ra '0'
END_IF
END_FOR
```

Nhập các số hex

Việc nhập các số hex bao gồm chữ số từ "0" đến "9" và các chữ cái từ "A" đến "F" kết thúc bằng ký tự CR. Để đơn giản chúng ta giả thiết :

- Chỉ sử dụng các chữ hoa
- Người sử dụng chỉ đưa vào tối đa 4 chữ số hex.

Quá trình đổi các ký tự thành trị nhị phân phức tạp hơn so với trường hợp nhập

số nhị phân, ngoài ra BX phải được dịch 4 lần để giành chỗ cho một giá trị hex.

Thuật toán nhập số hex

Xoá BX /*BX sẽ chứa giá trị nhập vào*/

Nhập ký tự hex

WHILE ký tự<>CR DO

 Đổi ký tự ra trị số nhị phân

 Dịch trái BX 4 lần

 Chèn giá trị mới vào 4 bit thấp của BX

 Nhập ký tự

END WHILE

Biểu diễn việc nhập 6ABh

Xoá BX

BX=0000 0000 0000 0000

Nhập vào "6", đổi thành 0110

Dịch trái BX 4 lần

BX=0000 0000 0000 0000

Chèn giá trị nhận được vào 4 bit thấp của BX

BX=0000 0000 0000 0110

Nhập vào "A", đổi thành Ah=1010

Dịch trái BX 4 lần

BX=0000 0000 0110 0000

Chèn giá trị nhận được vào 4 bit thấp của BX

BX=0000 0000 0110 1010

Nhập vào "B", đổi thành Bh=1011

Dịch trái BX 4 lần

BX=0000 0110 1010 0000

Chèn giá trị nhận được vào 4 bit thấp của BX

BX=0000 0110 1010 1011

BX chứa 06ABh.

Đoạn chương trình

XOR BX, BX ;Xoá BX

MOV CL, 4 ;Bộ đếm 4 lần dịch

MOV AH, 1 ;Hàm con nhập ký tự từ bàn phím

INT. 21H ;Nhập một ký tự

WHILE_:

CMP AL, 0DH ; CR ?

JE END WHILE ;Đúng!, kết thúc

;Đổi ký tự ra giá trị nhị phân

CMP AL, 39H ;Đó là chữ số ?

| | |
|---------------------|----------------------------------|
| JG LETTER | ;Không phải!, đó là một chữ cái |
| ;Nhập một số | |
| AND AL, 0FH | ;đổi chữ số ra giá trị nhị phân |
| JMP SHIFT | ;Đem chèn vào BX |
| LETTER: | ;Đổi chữ cái ra trị số nhị phân |
| SUB AL, 37H | |
| SHIFT: | |
| SHL BX, CL | ;Giành chỗ cho giá trị mới |
| ;đưa giá trị vào BX | |
| OR BL, AL | ;Chèn giá trị mới vào 4 bit thấp |
| INT 21H | ;của BX |
| JMP WHILE_ | ;Nhập tiếp ký tự từ bàn phím |
| | ;Lặp lại cho đến khi phím ENTER |
| | ;được ấn |

END WHILE:

Chú ý rằng chương trình không kiểm tra xem các ký tự nhập vào có hợp lệ hay không.

Đưa ra số hex

BX chứa số 16 bit bằng giá trị của số hex 4 chữ số. Để đưa ra nội dung của BX, chúng ta bắt đầu từ bên trái, lấy ra từng nhóm bit của mỗi chữ số rồi đổi nó thành chữ số hex tương ứng sau đó đưa ra.

Thuật toán đưa ra số hex

```

FOR 4 lần DO
    Chuyển BH vào DL /*BX chứa giá trị đưa ra*/
    Dịch DL về bên phải 4 lần
    IF DL<10
        THEN
            Đổi thành một trong các ký tự:
            "0", ... "9"
    ELSE
        Đổi thành một trong các chữ cái:
        "A" ... "F"
    END_IF
    Đưa ký tự ra
    Quay BX 4 lần về bên trái
END_FOR

```

Biểu diễn việc đưa ra số 4CA9h trong BX

BX=4CA9h=0100 1100 1010 1001
Chuyển BH vào DL
DL=0100 1100
Dịch phải DL 4 lần
DL=0000 0100
Đổi thành ký tự và đưa ra
DL=0011 0100=34h= 'A'
Quay BX 4 lần về bên trái
BX=1100 1010 1001 0100
Chuyển BH vào DL
DL=1100 1010
Dịch phải DL 4 lần
DL=0000 1100
Đổi thành ký tự và đưa ra
DL=0100 0011=43h= 'C'
Quay BX 4 lần về bên trái
BX=1010 1001 0100 1100
Chuyển BH vào DL
DL=1010 1001
Dịch phải DL 4 lần
DL=0000 1010
Đổi thành ký tự và đưa ra
DL=0100 0010=42h= 'B'
Quay BX 4 lần về bên trái
BX=1001 0100 1100 1010
Chuyển BH vào DL
DL=1001 0100
Dịch phải DL 4 lần
DL=0000 1001
Đổi thành ký tự và đưa ra
DL=0011 1001=39h= '9'
Quay BX 4 lần về bên trái
BX=0100 1100 1010 1001= Giá trị ban đầu.

Việc lập chương trình theo thuật toán này chúng tôi giành cho các bạn.

TỔNG KẾT

Trong chương này chúng ta đã học được:

- ◆ 5 lệnh logic là AND, OR, XOR, NOT và TEST
- ◆ Lệnh OR được sử dụng để thiết lập các bit riêng biệt của toán hạng đích hay để kiểm tra xem toán hạng đích có bằng 0 hay không.
- ◆ Lệnh XOR được dùng để lấy bù các bit riêng biệt của toán hạng đích hay để xoá nó về 0.
- ◆ Lệnh NOT dùng để lấy bù 1 của toán hạng đích.
- ◆ Lệnh TEST dùng để kiểm tra từng bit riêng biệt của toán hạng đích. Chẳng hạn nó có thể kiểm tra xem một số là chẵn hay lẻ.
- ◆ Các lệnh SHL và SAL dịch từng bit của toán hạng đích sang trái một vị trí. Bit msb được đưa vào cờ CF và một giá trị 0 được đưa vào bit lsb.
- ◆ Lệnh SHR dịch từng bit của toán hạng đích sang phải 1 vị trí, bit msb được đưa vào cờ CF và một giá trị 0 được đưa vào bit lsb.
- ◆ Lệnh SAR hoạt động tương tự như lệnh SHR nhưng bit msb được giữ nguyên giá trị.
- ◆ Các lệnh dịch có thể dùng để nhân hoặc chia cho 2. Các lệnh SHL và SAL nhân đôi giá trị trừ khi có hiện tượng tràn xảy ra. Các lệnh SHR và SAR chia đôi giá trị của toán hạng đích nếu nó là số chẵn. Trong trường hợp nó là số lẻ các lệnh này chia đôi giá trị của nó và làm tròn xuống số nguyên gần nhất. Lệnh SHR dùng cho số học không dấu và lệnh SAR dùng cho số học có dấu.
- ◆ Lệnh ROL dịch từng bit của toán hạng đích sang trái, bit msb được đưa vào vị trí của bit lsb. Đối với lệnh ROR từng bit dịch sang phải một vị trí, bit lsb được đưa vào vị trí của bit msb. Trong cả 2 trường hợp cờ CF chứa bit cuối cùng được dịch ra khỏi toán hạng đích.
- ◆ Các lệnh RCL và RCR hoạt động giống như các lệnh ROL và ROR nhưng bit bị quay ra khỏi toán hạng đích được đưa vào cờ CF còn nội dung của cờ CF được đưa vào toán hạng đích
- ◆ Cũng có thể thực hiện các phép dịch và quay nhiều lần cùng một lúc, khi đó CL phải chứa số lần dịch hoặc quay cần thực hiện
- ◆ Các lệnh dịch và quay được sử dụng rất hữu hiệu để vào/ra với các số nhị phân và số hex.

Các thuật ngữ tiếng Anh

| | |
|-------------------|---|
| Clear | Xoá - đổi giá trị thành 0 |
| Complement | Lấy bù - Đổi từ 0 về 1 và 1 về 0 |
| Mask | Mặt nạ - Một mẫu bit dùng trong các thao tác logic để xoá, thiết lập hay kiểm tra các bit xác định trong 1 toán hạng. |
| SET | Thiết lập - Đổi giá trị của bit thành 1 |

Các lệnh mới

| | | |
|------------|----------------|-------------|
| AND | RCR | SAR |
| NOT | ROL | SHR |
| OR | ROR | TEST |
| RCL | SAL/SHL | XOR |

Bài tập

1. Thực hiện các phép tính lôgic sau đây

- a. 10101111 AND 10001001
- b. 10110001 OR 01001001
- c. 01111100 XOR 11011010
- d. NOT 01011110

2. Viết các lệnh lôgic để thực hiện các công việc sau đây.

- a. Xoá các bit ở vị trí chẵn của AX, giữ nguyên các bit khác.
- b. Thiết lập các bit lsb và msb của BL trong khi giữ nguyên các bit khác.
- c. Đảo bit msb của BL, giữ nguyên các bit khác.
- d. Thay nội dung của biến word1 bằng số bù 1 của nó.

3. Dùng lệnh TEST để :

- a. Thiết lập cờ ZF nếu nội dung của AX bằng 0.
- b. Xoá cờ ZF nếu nội dung của là một số lẻ.
- c. Thiết lập SF nếu DX chứa số âm.
- d. Thiết lập ZF nếu DX chứa số dương hoặc 0.
- e. Thiết lập PF nếu BL chứa một số chẵn các bit 1.

4. Giả sử AL chứa 11001011b và CF=1, cho biết nội dung mới của AL khi mỗi lệnh sau được thực hiện. Điều kiện đầu đúng cho tất cả các phần của câu hỏi.

- a. SHL AL,1
- b. SHR AL,1
- c. ROL AL,CL trong đó CL chứa 2
- d. ROR AL,CL trong đó CL chứa 3
- e. SAR AL,CL trong đó CL chứa 2
- f. RCL AL,1
- g. RCR AL,CL trong đó CL chứa 3

5. Viết 1 hay nhiều lệnh để thực hiện các công việc sau, giả sử không có hiện tượng tràn xảy ra:

- a. Nhân đôi một biến kiểu byte có giá trị B5h
- b. Nhân nội dung của AL với 8.
- c. Chia 32142 cho 4 và lưu kết quả trong AX.
- d. Chia -2145 cho 16 và lưu kết quả trong BX.

6. Viết các lệnh thực hiện các công việc sau:

- a. Giả sử AL chứa giá trị nhỏ hơn 10, hãy đổi nó thành một chữ số thập phân.
- b. Giả sử DL chứa mã ASCII của một chữ hoa, hãy đổi nó thành chữ thường.

7. Viết các lệnh thực hiện các công việc sau:
- Nhân nội dung của BL với 10d, giả sử không có hiện tượng tràn.
 - Giả sử AL chứa số âm, hãy chia nó cho 8 và lưu số dư vào AH. (Gợi ý :dùng lệnh ROR)
- ### Các bài tập lập trình
8. Viết một chương trình thông báo cho người sử dụng vào một ký tự, in ra trên các dòng liên tiếp nhau mã ASCII của ký tự đó dưới dạng nhị phân, số các chữ số 1 trong mã ASCII dưới dạng nhị phân đó.
- Ví dụ:
- Đánh vào một ký tự : A
- Mã ASCII của A dưới dạng nhị phân là 01000001
- Số các bit 1 là 2
9. Viết chương trình thông báo cho người sử dụng đánh vào một ký tự và in ra mã ASCII của ký tự dưới dạng hex ở dòng tiếp theo. Lặp lại cho đến khi người sử dụng đánh ENTER.
- Ví dụ:
- Đánh vào một ký tự: Z
- Mã ASCII của Z dưới dạng hex là 5A
- Đánh vào một ký tự:
10. Viết chương trình thông báo cho người sử dụng đánh vào một số hex nhỏ hơn hay bằng 4 chữ số. Đưa ra số dưới dạng nhị phân ở dòng kế tiếp. Khi người sử dụng đưa vào một ký tự không hợp lệ, thông báo để họ vào lại. Chương trình chỉ nhận các chữ in hoa.
- Ví dụ:
- Đánh vào một số hex(0..FFFF): 1a
- Chữ số hex không hợp lệ, hãy vào lại :1ABC
- Dưới dạng nhị phân nó bằng: 0001101010111100
- Chương trình của bạn có thể bỏ qua các ký tự sau 4 ký tự đầu tiên.
11. Viết chương trình thông báo cho người sử dụng đánh vào một số nhị phân có nhỏ hơn hay bằng 16 chữ số. Đưa ra số dưới dạng hex ở dòng kế tiếp. Khi người sử dụng đưa vào một ký tự không hợp lệ, thông báo để họ vào lại.
- Ví dụ:
- Đánh vào một số nhị phân (nhiều nhất 16 chữ số):1110000001
- Dưới dạng hex nó bằng:E1
- Chương trình của bạn có thể bỏ qua các ký tự sau 16 ký tự đầu tiên.
12. Viết một chương trình thông báo cho người sử dụng đưa vào 2 số nhị phân, mỗi số có 8 chữ số, in ra màn hình ở dòng tiếp theo tổng của chúng dưới dạng nhị phân. Mỗi khi người sử dụng đánh vào một ký tự không hợp lệ sẽ có thông báo yêu cầu

vào lại. Mỗi số được nhận sau khi người sử dụng sử dụng đánh ENTER.

Ví dụ:

Đánh vào một số nhị phân 8 chữ số: 11001010

Đánh vào một số nhị phân 8 chữ số: 10011100

Tổng của chúng dạng nhị phân bằng: 101100110

13. Viết một chương trình thông báo cho người sử dụng đưa vào 2 số hex không dấu trong khoảng từ 0 đến FFFFh, in ra màn hình ở dòng tiếp theo tổng của chúng dưới dạng hex. Mỗi khi người sử dụng đánh vào một ký tự không hợp lệ sẽ có thông báo yêu cầu vào lại. Chương trình của bạn phải có khả năng kiểm soát hiện tượng tràn không dấu. Mỗi số được nhận sau khi người sử dụng sử dụng đánh ENTER.

Ví dụ:

Đánh vào một số hex, 0 - FFFF: 21AB

Đánh vào một số hex, 0 - FFFF: FE03

Tổng của chúng là 11FAE

14. Viết một chương trình thông báo cho người sử dụng đánh vào một chuỗi các chữ số thập phân kết thúc bằng phím ENTER, và in ra màn hình ở dòng kế tiếp tổng của chúng ở dạng hex. Nếu người sử dụng đánh vào một ký tự không hợp lệ phải thông báo để họ vào lại.

Ví dụ:

Đánh vào một chuỗi các chữ số thập phân: 1299843

Tổng của các chữ số dưới dạng hex là 0024

Chương 8

NGĂN XẾP VÀ CÁC THỦ TỤC

Tổng quan.

Đoạn ngắn xếp của chương trình dùng để lưu trữ tạm thời các dữ liệu và địa chỉ. Trong chương này chúng tôi trình bày về các thao tác với ngăn xếp và cách thức sử dụng ngăn xếp để khai báo các thủ tục.

Trong mục 8.1, chúng tôi giới thiệu về các lệnh để cất vào và lấy ra dữ liệu từ ngăn xếp: PUSH và POP. Vì từ cuối cùng cất vào ngăn xếp sẽ được lấy ra đầu tiên nên ngăn xếp có thể dùng để đảo ngược thứ tự một dãy dữ liệu. Tính chất này sẽ được khai thác trong mục 8.2.

Các thủ tục là phần hết sức quan trọng trong lập trình ngôn ngữ bậc cao, điều này cũng đúng với ngôn ngữ Hợp ngữ. Mục 8.3 và 8.4 sẽ trình bày những điểm cơ bản của các thủ tục trong ngôn ngữ Hợp ngữ. Về mặt kỹ thuật, chúng ta có thể biết chính xác một thủ tục được gọi và trả về chương trình gọi nó như thế nào. Trong mục 8.5, chúng tôi sẽ đưa ra ví dụ một thủ tục thực hiện phép nhân nhị phân bằng phương pháp cộng và dịch bit. Qua ví dụ này bạn sẽ học thêm được chút ít về chương trình DEBUG.

8.1. Ngăn xếp.

Ngăn xếp (LIFO) là cấu trúc dữ liệu một chiều. Các phần tử được cất vào và lấy ra từ một đầu của cấu trúc, tức là nó được xử lý theo phương thức ‘vào trước, ra sau’ (LIFO: Last-In, First-Out). Phần tử được cất vào cuối cùng gọi là đỉnh của ngăn xếp. Ta có thể hình dung ngăn xếp như một chồng đĩa; chiếc đĩa cuối cùng được xếp vào nằm trên đỉnh và chỉ có nó mới có thể được lấy ra đầu tiên.

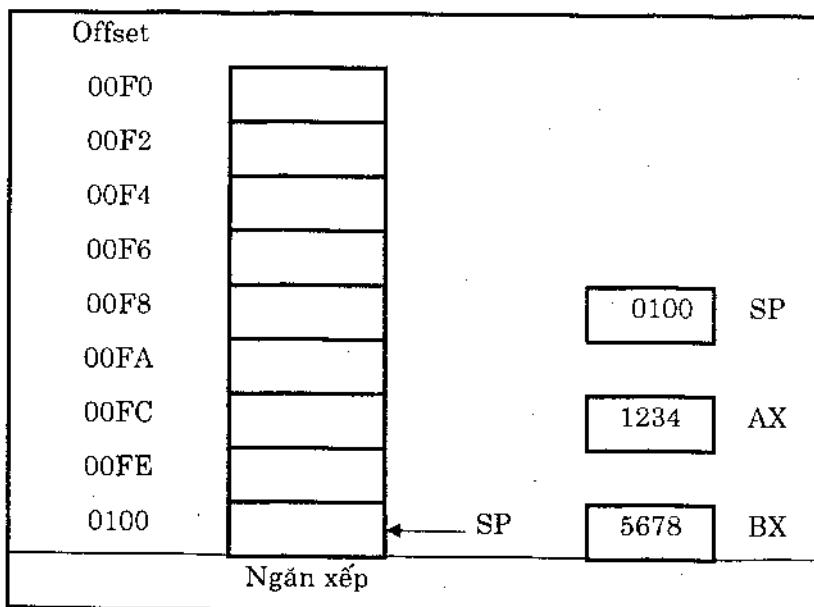
Mỗi chương trình phải dành ra một khối bộ nhớ để làm ngăn xếp. Chúng ta thực hiện điều này khi khai báo đoạn ngắn xếp. Ví dụ:

STACK 100h

Khi chương trình được biên dịch và nạp vào bộ nhớ, SS sẽ chứa địa chỉ đoạn ngắn xếp. Trong khai báo ngắn xếp nêu trên, con trỏ ngắn xếp SP được khởi tạo bằng 100h. Điều này làm phát sinh một vị trí ngắn xếp rỗng. Khi ngắn xếp không rỗng, SP chứa địa chỉ offset của đỉnh ngắn xếp.

PUSH và PUSHF.

Hình 8.1.a Ngắn xếp rỗng:



Lệnh PUSH được dùng để thêm một từ mới vào trong ngắn xếp. Cú pháp:

PUSH nguồn

Ở đây nguồn là một thanh ghi 16 bit hoặc một từ nhớ. Ví dụ:

PUSH AX

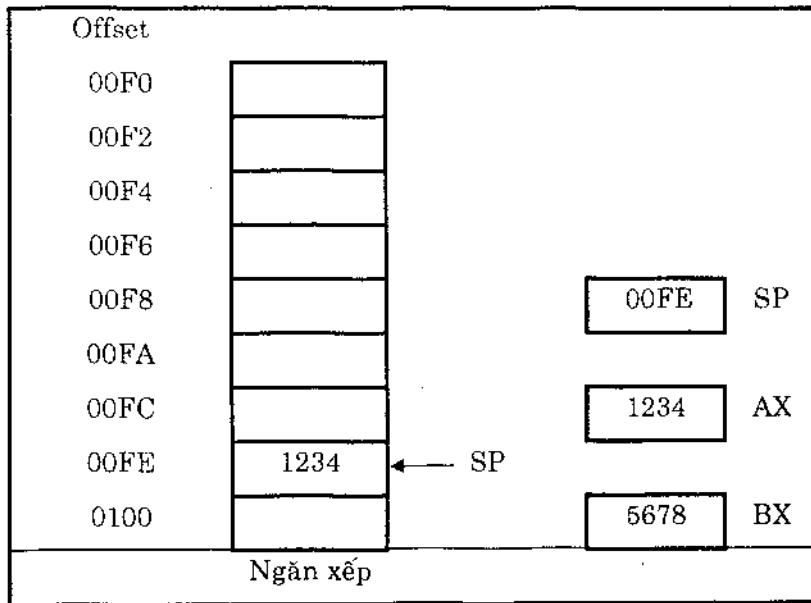
Lệnh PUSH thực hiện các công việc sau đây:

- Giảm SP đi 2.
- Một bản sao của nội dung của toán hạng nguồn được chuyển vào địa chỉ xác định bởi SS : SP. Toán hạng nguồn không bị thay đổi.

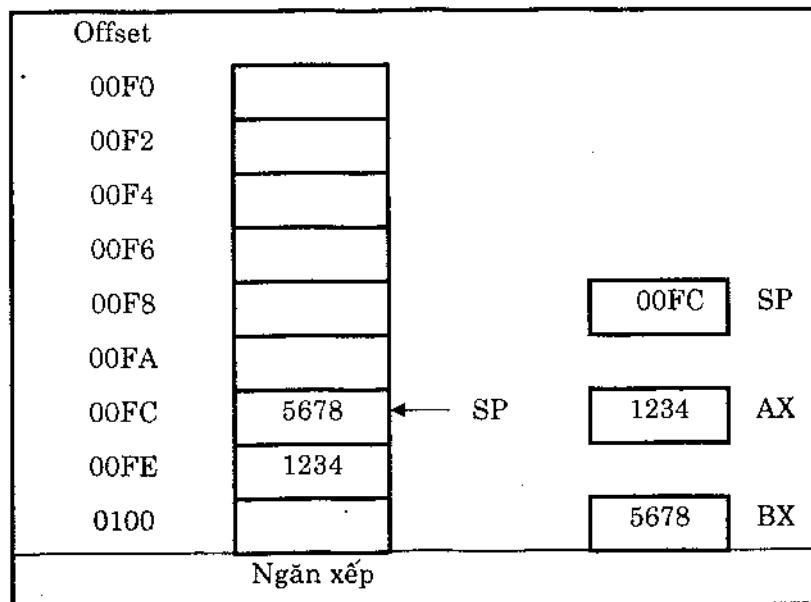
Lệnh PUSHF không có toán hạng cất nội dung của thanh ghi cờ vào ngắn xếp.

Ban đầu, SP chứa địa chỉ offset của ô nhớ theo sau đoạn ngắn xếp. Lệnh PUSH đầu tiên giảm SP đi 2 làm cho con trỏ chỉ đến từ cuối cùng của trong đoạn ngắn xếp. Bởi vì lệnh PUSH làm giảm SP nên ngắn xếp phát triển về phía đỉnh bộ nhớ. Hình 8.2 chỉ ra lệnh PUSH làm việc như thế nào.

Hình 8.1.b Sau lệnh PUSH AX



Hình 8.1.c Sau lệnh PUSH BX



POP và POPF

Lệnh POP được dùng để lấy ra phần tử đỉnh ngăn xén. Cú pháp:

POP-ÚJÍCH

trong đó toán tử `dich` là một thanh ghi 16 bit (trừ `IP`) hoặc là một từ 32 bit `U32`.

POP BY

Lệnh POP thực hiện các công việc sau đây:

1. Nội dung của ô nhớ SS : SP (đỉnh ngăn xếp) được chuyển tới toán tử đích.
 2. SP tăng lên 2.

Hình 8.2 chỉ ra một lệnh POP làm việc như thế nào

Lệnh POPF đưa vào thanh ghi cờ nội dung của định nghĩa xem

Các lệnh PUSH, PUSHE, POP và POPE đều không có biến số.

Lưu ý rằng các lệnh PUSH và POP chỉ thao tác với các WORD, vậy nên nếu dùng với các byte phụ sau:

PUSH DL ;không bao giờ

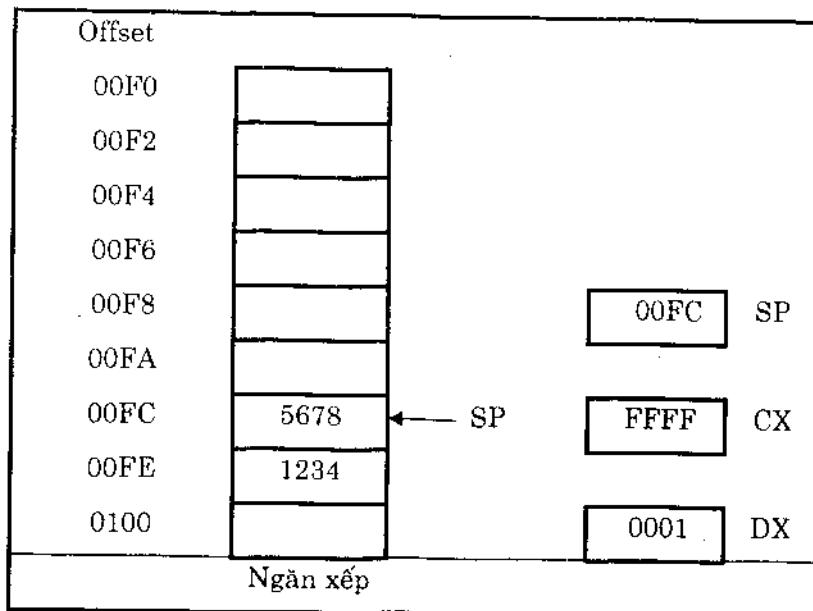
là không hợp lệ. Cũng như vậy với số liệu trực tiếp

PUSH 2 ;không bàn 10

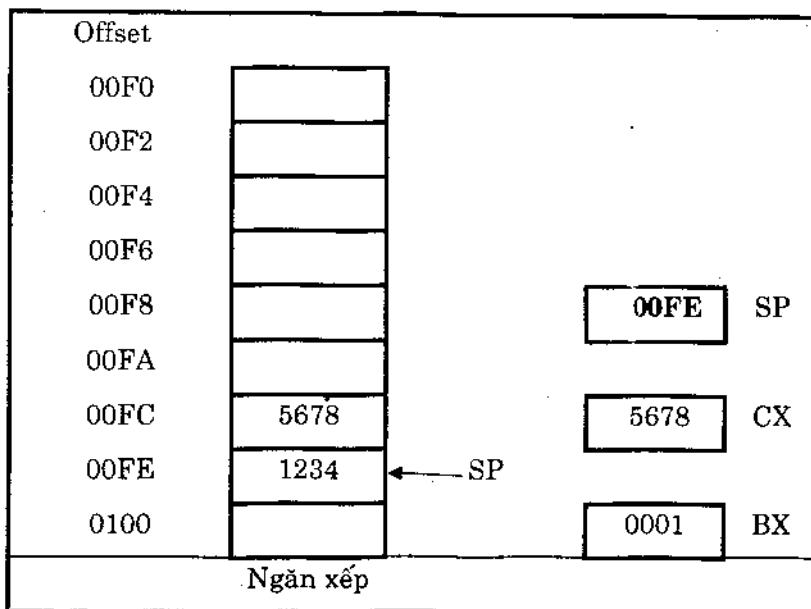
Chú ý: Cắt một số liệu trực tiếp là được phép đối với các bộ vi xử lý 80186, 80486. Các bộ vi xử lý này sẽ được trình bày ở chương 20.

Ngoài chương trình của người sử dụng, hệ điều hành cũng sử dụng ngăn xếp cho các mục đích của riêng nó. Ví dụ để thực hiện hàm INT 21h, DOS ghi lại mọi thanh ghi mà nó dùng đến vào ngăn xếp và phục hồi chúng khi phục vụ ngắt được hoàn thành. Người sử dụng không cần quan tâm đến vấn đề này bởi vì tất cả các giá trị mà DOS cất trong ngăn xếp sẽ được lấy ra hết trước khi trả điều khiển cho chương trình của người sử dụng.

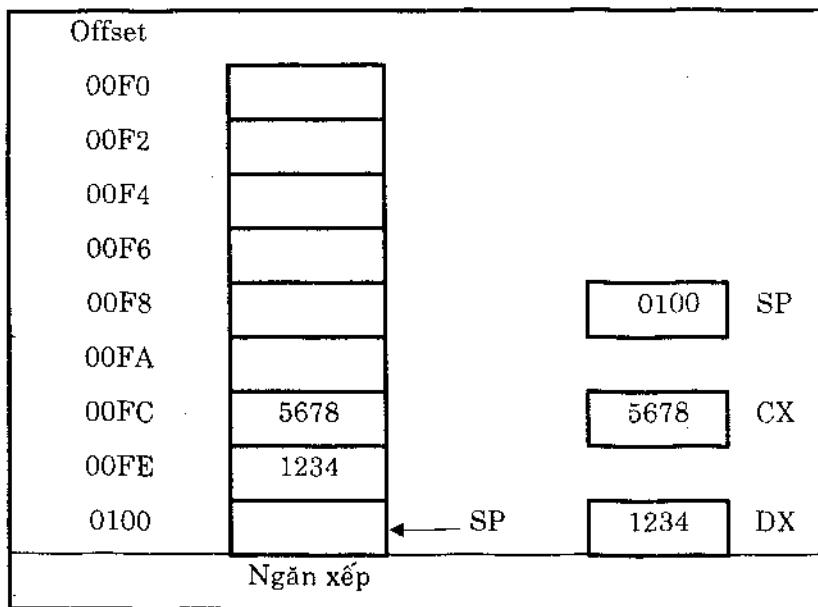
Hình 8.2A. Trước lệnh POP.



Hình 8.2B. Sau lệnh POP CX.



Hình 8.2C. Sau lệnh POP DX.



8.2. Một ứng dụng của ngăn xếp.

Bởi vì ngăn xếp hoạt động theo phương thức vào trước-ra sau nên thứ tự của các phần tử cất vào ngăn xếp sẽ bị đảo ngược khi lấy ra. Chương trình sau đây sẽ sử dụng tính chất này để đọc một chuỗi ký tự và hiển thị chúng theo thứ tự ngược lại trên dòng tiếp theo.

Thuật toán đảo ngược thứ tự.

Hiển thị dấu '?'

Khởi động bộ đếm bằng 0.

Đọc một ký tự.

WHILE không phải ký tự về đầu dòng DO

Cất ký tự vào ngăn xếp.

Tăng biến đếm.

Đọc một ký tự.

END WHILE

Chuyển con trỏ xuống dòng mới.

```

FOR biến đếm DO
    Lấy một ký tự từ ngăn xếp.
    Hiển thị nó.
END_FOR

```

Sau đây là chương trình:

Chương trình PGM8_1.ASM

```

1:      TITLE      PGM8_1:REVERSE INPUT
2:      .MODEL     SMALL
3:      .STACK     100H
4:      CODE
5:      MAIN      PROC
6:          ;hiển thị lời nhắc người sử dụng
7:          MOV      AH,2          ;chuẩn bị hiển thị
8:          MOV      DL,'?'        ;ký tự để hiển thị
9:          INT      21h          ;hiển thị '?'
10:         ;khởi động biến đếm ký tự
11:         XOR      CX,CX
12:         ;đọc một ký tự
13:         MOV      AH,1          ;chuẩn bị đọc
14:         INT      21h          ;đọc một ký tự
15:         ;while không phải ký tự xuống dòng do
16: WHILE_:
17:         CMP      AL,0Dh        ;CR?
18:         JE      END WHILE    ;đúng, thoát khỏi vòng lặp
19:         ;cất ký tự vào ngăn xếp và tăng biến đếm
20:         PUSH     AX          ;cất vào ngăn xếp
21:         INC      CX          ;đếm = đếm+1
22:         ;đọc một ký tự
23:         INT      21h          ;đọc một ký tự
24:         JMP      WHILE        ;trở về vòng lặp
25: END WHILE:
26:         ;xuống dòng tiếp theo
27:         MOV      AH,2          ;hàm hiển thị ký tự
28:         MOV      DL,0Dh        ;CR
29:         INT      21h          ;thi hành
30:         MOV      DL,0Ah        ;LF
31:         INT      21h          ;thi hành
32:         JCXZ    EXIT         ;thoát nếu đếm=0
33:         ;for biến đếm do

```

```

34:     TOP:
35:     ;lấy một ký tự từ ngăn xếp
36:             POP    DX          ; lấy một ký tự từ ngăn xếp
37:     ;hiển thị nó
38:             INT    21h        ;hiển thị nó
39:             LOOP   TOP
40:     ;end_for
41:     EXIT:
42:             MOV    AH, 4CH
43:             INT    21H
44:     MAIN    ENDP
45:             END    MAIN

```

Bởi lẽ số các ký tự nhập vào là không xác định trước, chương trình dùng CX để đếm nó. Sau đó CX điều khiển vòng lặp FOR hiển thị các ký tự theo thứ tự ngược lại.

Trong các dòng 16-24, chương trình dùng một vòng lặp WHILE để cất các ký từ ngăn xếp và đọc các ký tự mới cho đến khi ký tự về đầu dòng được đánh vào. Mặc dù các ký tự chỉ chứa trong AL, chúng ta vẫn phải cất cả AX vào ngăn xếp bởi vì toán tử trong lệnh PUSH phải là một WORD.

Khi chương trình thoát khỏi vòng lặp WHILE (dòng 25), tất cả các ký tự đều đã được cất vào ngăn xếp với byte thấp của đỉnh ngăn xếp chứa ký tự cất vào sau cùng, AL chứa mã ASCII của ký tự về đầu dòng.

Dòng 32 chương trình kiểm tra xem có ký tự nào được đọc vào hay không. Nếu không thì CX bằng 0 và chương trình nhảy trở về DOS. Nếu ít nhất một ký tự được đọc, chương trình đi vào vòng lặp FOR, trong đó các lệnh POP lặp lại sẽ đưa vào DX số liệu lấy từ ngăn xếp (do đó DL chứa mã của ký tự) và hiển thị các ký tự.

Một ví dụ khi chạy chương trình:

C>PRG8_1

?THIS IS A TEST
TSET A SI SIHT

C>PRG8_1

? (chỉ ký tự về đầu dòng được đánh vào)

C>

8.3. Các thuật ngữ của thủ tục.

Trong chương 6 chúng tôi đã đề cập đến ý tưởng lập trình từ trên xuống. Nội dung của ý tưởng này là phân tích vấn đề ban đầu thành chuỗi các vấn đề con dễ thực hiện hơn. Các ngôn ngữ bậc cao thường dùng các thủ tục để giải quyết các vấn đề con này và chúng ta cũng có thể làm tương tự đối với Hợp ngữ. Như vậy một chương trình Hợp ngữ có thể được tạo nên bằng cách kết hợp các thủ tục.

Trong số các thủ tục sẽ có một thủ tục chính, nó chứa điểm xuất phát của chương trình. Để thực hiện một nhiệm vụ, thủ tục chính gọi một trong số các thủ tục còn lại. Rất có thể các thủ tục này lại gọi các thủ tục khác hay một thủ tục gọi chính nó.

Khi một thủ tục gọi một thủ tục khác, điều khiển được chuyển đến thủ tục được gọi và các lệnh của nó được thi hành. Thủ tục được gọi luôn trả điều khiển về cho thủ tục gọi tại lệnh tiếp theo của dòng lệnh gọi (Hình 8.3). Trong các ngôn ngữ bậc cao cơ chế gọi và trả về là ẩn đối với người lập trình còn trong Hợp ngữ chúng ta có thể thấy rõ nó làm việc như thế nào (xem mục 8.4).

Khai báo thủ tục.

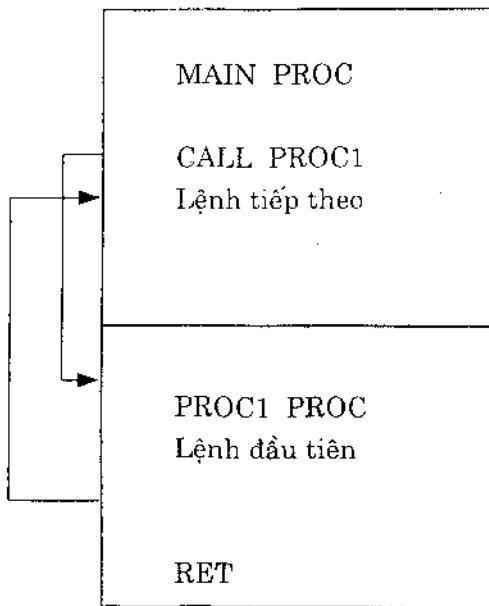
Cú pháp khai báo một thủ tục như sau:

```
name PROC type  
;thân thủ tục  
...  
RET  
name ENDP
```

Trong đó, name là tên của thủ tục định nghĩa bởi người sử dụng. Toán hạng tùy chọn type có thể là NEAR hay FAR (NEAR được ngầm định nếu bỏ qua type). NEAR có nghĩa là dòng lệnh gọi thủ tục ở cùng đoạn với thủ tục đó, ngược lại tùy chọn FAR có nghĩa dòng lệnh gọi ở trong một đoạn khác. Các phần sau đây ta giả thiết mọi thủ tục đều có kiểu NEAR. Các thủ tục FAR sẽ được trình bày trong chương 14.

RET

Để trả điều khiển về chương trình gọi ta sử dụng lệnh RET (return). Mọi thủ tục (ngoại trừ thủ tục chính) đều phải có một lệnh RET ở đâu đó. Thường thì nó là lệnh cuối cùng của thủ tục.



Liên lạc giữa các thủ tục

Một thủ tục phải có cách nào đó để nhận các giá trị và trả về kết quả cho thủ tục gọi nó. Khác với các thủ tục của ngôn ngữ bậc cao, các thủ tục của Hợp ngữ không có danh sách tham số, vì thế các lập trình viên phải nghĩ cách cho việc liên lạc giữa các thủ tục. Ví dụ nếu chỉ có vài số liệu vào, ra ta có thể chứa chúng trong các thanh ghi. Phương pháp chung cho việc liên lạc giữa các thủ tục sẽ được trình bày trong chương 14.

Chú giải các thủ tục.

Ngoài các yêu cầu về cú pháp, mỗi thủ tục cũng nên có những lời giải thích để bất kỳ ai đọc chương trình nguồn đều có thể hiểu được các thủ tục làm gì, nó lấy số liệu và trả về kết quả ở đâu. Trong cuốn sách này, thông thường chúng tôi chú giải các thủ tục như bằng khái lời bình như sau:

- ; (miêu tả thủ tục làm gì)
- ; Vào: (nơi lấy thông tin từ chương trình gọi)
- ; Ra: (nơi trả về thông tin cho chương trình gọi)
- ; Sử dụng: (Các chương trình mà thủ tục gọi)

8.4. CALL và RET.

Lệnh CALL được dùng để gọi một thủ tục. Có hai kiểu gọi thủ tục là gọi trực tiếp và gián tiếp. Cú pháp của lệnh gọi thủ tục trực tiếp:

Ở đây name là tên của thủ tục. Cú pháp của lệnh gọi thủ tục gián tiếp:

CALL address_expression

với address_expression là một thanh ghi hay ô nhớ chứa địa chỉ của thủ tục.

Lệnh CALL thực hiện các công việc sau đây:

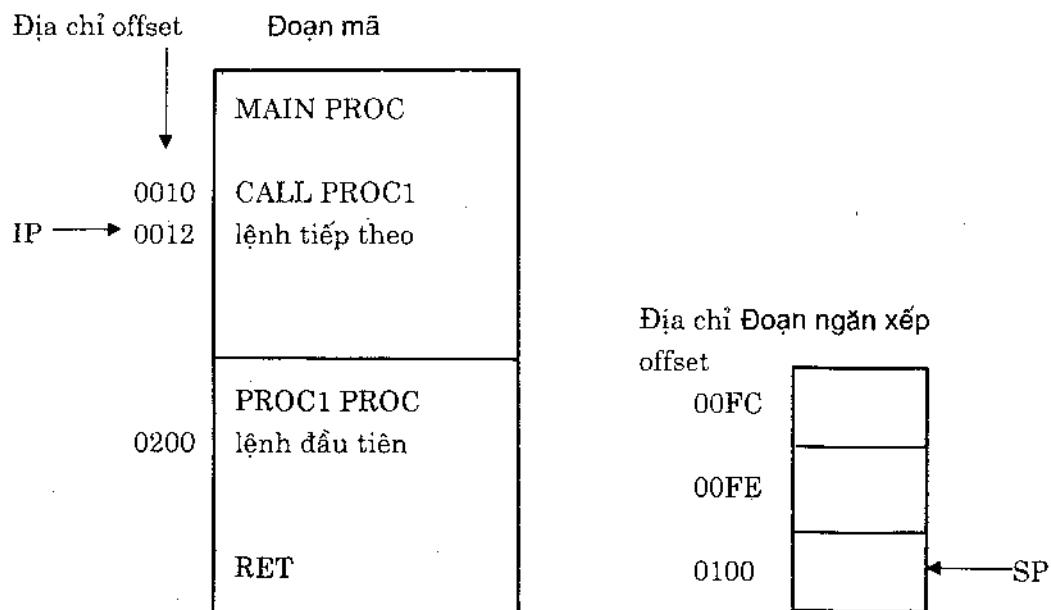
1. Địa chỉ trả về của chương trình gọi được cất vào ngăn xếp. Địa chỉ này là offset của lệnh ngay sau dòng lệnh CALL, dạng segment:offset của nó tại thời điểm thi hành lệnh CALL chứa trong CS : IP.
2. IP được gán bằng địa chỉ offset lệnh đầu tiên của thủ tục. Thao tác này chuyển điều khiển cho thủ tục. (Xem hình 8.4A và 8.4B)

Để trả về từ một thủ tục chúng ta dùng lệnh:

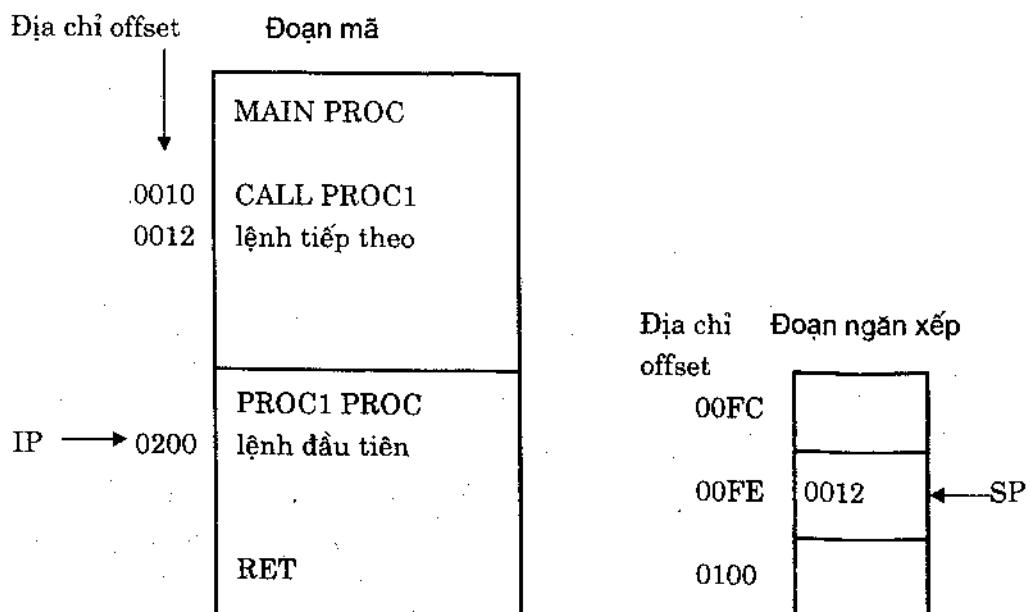
RET pop_value

Tham số nguyên pop_value là tùy chọn. Trong một thủ tục NEAR, lệnh RET đưa giá trị ở đỉnh ngăn xếp vào IP. Nếu pop_value xác định bằng N thì N sẽ được cộng vào SP. Điều này tương đương với việc lấy N byte khỏi ngăn xếp CS : IP lúc này chứa địa chỉ trả về dạng segment:offset và điều khiển được trả lại cho chương trình gọi (Xem hình 8.5A và 8.5B).

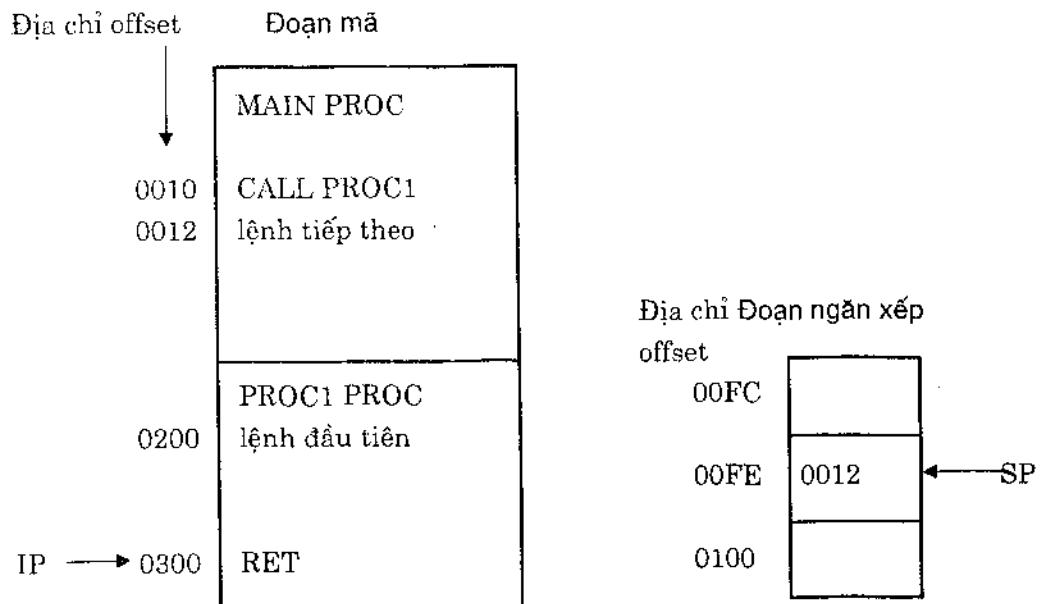
Hình 8.4A Trước lệnh CALL.



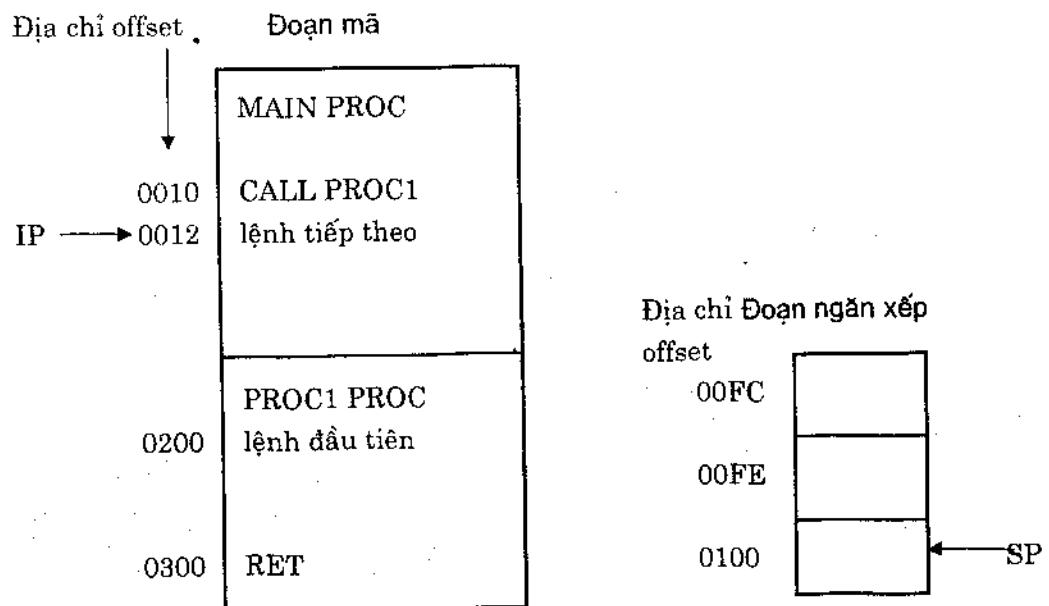
Hình 8.4. Sau lệnh CALL.



Hình 8.5A.Trước lệnh RET.



Hình 8.5B. Sau lệnh RET



8.5 Ví dụ về thủ tục

Để làm ví dụ chúng ta sẽ viết một thủ tục nhân 2 số nguyên dương A và B bằng cách cộng và dịch các bit, đây là một phương pháp để thực hiện phép nhân không dấu trong máy tính (trong chương 9 chúng tôi sẽ trình bày về lệnh nhân).

Thuật toán thực hiện phép nhân:

Tích = 0

REPEAT

 IF lsb của B bằng 1 (lsb là bit có trọng lượng thấp nhất)

 THEN

 Tích = Tích+ A

 END_IF

 Dịch trái A

 Dịch phải B

 UNTIL B = 0

Ví dụ nếu $A = 111_b = 7$ và $B = 1101_b = 13$

Tích = 0

Vì bit lsb của B bằng 1 nên Tích = Tích+A = 111b

Dịch trái A: $A=1110_b$

Dịch phải B: $B=110_b$

Vì bit lsb của B bằng 0 nên:

Dịch trái A: $A=11100_b$

Dịch phải B: $B=11_b$

Vì bit lsb của B bằng 1 nên $Tích=Tích+A=111b+11100b$
 $=100011b$

Dịch trái A: $A=111000_b$

Dịch phải B: $B=0$

Vì bit lsb của B bằng 1 nên $Tích=Tích+A=100011b+111000$
 $=1011011b$

Dịch trái A: $A=1110000_b$

Dịch phải B: $B=0$

Vì bit lsb của B bằng 0 nên :

Tích nhận được = 1011011b . 91d

Chú ý chúng ta cũng có thể nhận được kết quả như vậy bằng cách thực hiện quá trình nhân thập phân thông thường trên các số nhị phân như sau:

$$\begin{array}{r} & 111b \\ * & 1101b \\ \hline & 111 \\ & 000 \\ & 111 \\ \hline & 1011011b \end{array}$$

Trong chương trình sau đây, thuật toán nhân được thực hiện bởi chương trình MULTIPLY. Chương trình chính không có số liệu vào, ra. Chúng ta sẽ dùng chương trình DEBUG để thực hiện các thao tác vào/ra.

Chương trình PGM8_1.ASM

```
1:      TITLE    PGM8_2: MULTIPLICATION BY ADD AND SHIFT
2:      .MODEL   SMALL
3:      .STACK   100H
4:      .CODE
5:      MAIN     PROC
6:          ;đưa A vào AX và B vào BX nhờ DEBUG
7:          CALL    MULTIPLY
8:          ;DX sẽ chứa kết quả
9:          MOV     AH, 4Ch
10:         INT    21h
11:         MAIN    ENDP
12:         MULTIPLY PROC
13:         ;nhân hai số A và B bằng phép cộng và dịch
14:         ;vào: AX=A; BX=B; Các số nằm trong khoảng(0-FFh)
15:         ;ra: DX=kết quả
16:         PUSH   AX
17:         PUSH   BX
18:         XOR   DX, DX  ;tích=0
19:         REPEAT:
20:         ;if B lẻ
21:         TEST   BX, 1       ;B lẻ?
```

```

22:           JZ END_IF ;không, B chẵn
23:       ;then
24:           ADD DX, AX ;tích=tích+A
25:       END_IF:
26:           SHL AX, 1      ;dịch trái A
27:           SHR BX, 1      ;dịch phải B
28:       ;until
29:           JNZ REPEAT
30:           POP BX
31:           POP AX
32:           RET
33:       MULTIPLY ENDP
34:       END MAIN

```

Thủ tục MULTIPLY nhận các biến vào A và B thông qua các thanh ghi AX và BX. Người sử dụng nạp giá trị cho các thanh ghi này trong chương trình DEBUG. Kết quả của phép tính trả về thanh ghi DX. Để tránh tràn, A và B phải được giới hạn trong khoảng từ 0 đến FFh.

Mỗi thủ tục thường bắt đầu bằng việc cất tất cả các thanh ghi mà nó sử dụng trong ngăn xếp và khôi phục lại các thanh ghi này khi kết thúc. Thao tác này là cần thiết bởi lẽ chương trình có thể có các dữ liệu chứa trong các thanh ghi và các thủ tục có thể gây ra các hiệu ứng lề nếu không ghi lại chúng trước. thậm chí là không cần thiết đối với chương trình này, chúng tôi vẫn cất các thanh ghi AX và BX (dòng 16 và 17) và khôi phục lại chúng (dòng 30 và 31) để làm ví dụ minh họa. Các thanh ghi được lấy ra khỏi ngăn xếp theo thứ tự ngược lại với khi chúng được cất vào.

Sau khi xoá DX, thanh ghi sẽ chứa tích, chương trình tiến vào vòng lặp REPEAT (dòng 19-29). Dòng 22 chương trình kiểm tra bit trọng lượng thấp nhất (lsb) của BX. Nếu BX có lsb bằng 1, AX sẽ được cộng vào tích chứa trong DX và BX dịch phải. Vòng lặp kết thúc khi BX = 0. Khi thoát ra, thủ tục sẽ chứa tích trong DX.

Sau khi biên dịch và liên kết chương trình, ta gọi nó trong DEBUG (trong dòng lệnh sau đây, người sử dụng trả lời bằng chữ đậm):

```
C> DEBUG PGM8_2.EXE
```

DEBUG sẽ trả lời bằng dấu nhắc lệnh ". Để liệt kê chương trình ta dùng lệnh U (Unassemble):

-U

```
177F:0000 E80400      CALL  0007
177F:0003 B44C  MOV   AH, 4C
177F:0005 CD21  INT   21
177F:0007 50    PUSH  AX
177F:0008 53    PUSH  BX
177F:0009 33D2  XOR   DX, DX
177F:000B F7C3010C  TEST  BX, 0001
177F:000F 7402  JZ    0013
177F:0011 03D0  ADD   DX, AX
177F:0013 D1E0  SHL   AX, 1
177F:0015 D1EB  SHR   BX, 1
177F:0017 75F2  JNZ   000B
177F:0019 5B    POP   BX
177F:001A 58    POPF  AX
177F:001B C3    RET
177F:001C E3D1  JCXZ FFEF
177F:001E E38B  JCXZ FFAB
```

Lệnh U của DEBUG sẽ dịch nội dung bộ nhớ thành các lệnh ngôn ngữ máy. Mỗi lệnh được hiển thị địa chỉ dạng segmet:offset, mã máy và mã lệnh Hợp ngữ. Tất cả các số biểu diễn dưới dạng hex. Kết quả thực hiện cho ta thấy chương trình chính nằm từ địa chỉ offset 0000 đến 0005; thủ tục MULTIPLY bắt đầu tại 0007 và kết thúc với lệnh RET tại 001B. Các lệnh sau đó không thuộc chương trình.

Trước khi vào số liệu, ta hãy xem các thanh ghi.

-R

```
AX=0000  BX=0000  CX=001C  DX=0000  SP=0100  BP=0000  SI=0000
DI=0000
DS=176F ES=176F SS=1781 CS=177F IP=0000 NV UP EI PL NZ NA PO NC
177F:0000 E80400 CALL 0007
```

Giá trị khởi tạo SP bằng 100h có nghĩa là chúng ta dành ra 100h byte cho ngăn xếp. Để xem ngăn xếp, chúng ta có thể liệt kê bộ nhớ bằng lệnh D.

-DSS:F0 FF

```
1781:00f0 00 00 00 00 00 00 00 6f 17-A4 13 07 00 6F 17 00 00
```

Lệnh DSS:F0 FF hiển thị các byte nhớ từ địa chỉ SS:F0 đến SS:FF. Đây chính là 16 byte cuối cùng của ngăn xếp. Nội dung mỗi byte được hiển thị dưới dạng số hex hai chữ số. Vì là ngăn xếp rỗng, các số được hiển thị ở đây không có ý nghĩa.

Trước khi thực hiện chương trình, chúng ta phải đưa các số A và B vào các thanh ghi AX và BX (ở đây ta sẽ cho A=7 và B=13=Dh). Để vào A ta dùng lệnh R:

```
-RAX  
AX 0000:7
```

RAX có nghĩa là chúng ta muốn thay đổi nội dung của AX. DEBUG sẽ hiển thị giá trị hiện tại, theo sau là dấu hai chấm (:) và đợi chúng ta vào giá trị mới. Tương tự chúng ta có thể khởi tạo BX:

```
-RBX  
BX 0000:D
```

Bây giờ ta hãy xem lại các thanh ghi:

```
-R  
AX=0007 BX=000D CX=001C DX=0000 SP=0100 BP=0000 SI=0000 DI=0000  
DS=176F ES=176F SS=1781 CS=177F IP=0000 NV UP EI PL NZ NA PO NC  
177F:0000 E80400 CALL 0007
```

Lúc này AX và BX đã chứa các giá trị khởi đầu.

Để xem lệnh CALL ảnh hưởng đến các cờ như thế nào chúng ta sử dụng lệnh T (Trace). Nó sẽ thi hành một lệnh và hiển thị các thanh ghi:

```
-T  
AX=0007 BX=000D CX=001C DX=0000 SP=00FE BP=0000 SI=0000 DI=0000  
DS=176F ES=176F SS=1781 CS=177F IP=0007 NV UP EI PL NZ NA PO NC  
177F:0007 50 PUSH AX
```

Chúng ta hãy chú ý hai thanh ghi bị thay đổi:

1. IP chứa 0007, offset bắt đầu của thủ tục MULTIPLY.

2. SP giảm từ 100h xuống 00FE bởi vì lệnh CALL cất địa chỉ trở về của thủ tục MAIN trong ngăn xếp.

Ta xem lại 16 byte cuối cùng của ngăn xếp:

```
-DSS:F0 FF
```

```
1781:00f0 00 00 00 00 00 00 00 6f 17-A4 13 07 00 6F 17 03 00
```

Địa chỉ trở về là 0003 nhưng lại được hiển thị là 03 00 vì DEBUG hiển thị byte thấp trước byte cao.

Ba lệnh đầu tiên của thủ tục MAIN cất AX, BX vào ngăn xếp và xoá DX. Để thực hiện chúng, ta dùng lệnh G (go). Cú pháp của lệnh này như sau:

G offset

Lệnh này thực hiện chương trình và dừng lại tại offset xác định. Nhờ bản dịch ngược chúng ta có thể thấy ngay rằng sau lệnh XOR DX,DX là offset 000Bh.

```
-GB
```

```
AX=0007 BX=000D CX=001C DX=0000 SP=00FA BP=0000 SI=0000 DI=0000  
DS=176F ES=176F SS=1781 CS=177F IP=000B NV UP EI PL ZR NA PE NC  
177F:000B F7C30100 TEST BX,01
```

Ta thấy rằng hai lệnh PUSH làm giảm SP 4 đơn vị: từ 00FEh xuống 00FAh. Ngăn xếp bây giờ như sau:

```
-DSS:F0 FF
```

```
1781:00f0 00 00 00 00 00 00 00 6f 17-A4 13 0D 00 07 00 03 00
```

Lúc này ngăn xếp chứa ba từ: các giá trị của BX (000D), AX (0007) và địa chỉ trả về (0003). Chúng được hiển thị như sau: 0D 00 07 00 03 00.

Bây giờ chúng ta hãy xem chương trình làm việc ra sao. Trước hết, ta cho chương trình chạy đến cuối vòng lặp REPEAT tại offset 0017h:

-G17

```
AX=000E BX=0006 CX=001C DX=0007 SP=00FA BP=0000 SI=0000 DI=0000  
DS=176F ES=176F SS=1781 CS=177F IP=0017 NV UP EI PL NZ AC PE CY  
177F:0017 75F2 JNZ 000B
```

Bởi vì giá trị khởi đầu của BX là 0Dh=1101b nên lsb của nó bằng 1, như vậy AX được cộng vào tích trong DX:

$DX=DX+AX=0000h+0007h=0007h$. AX dịch trái 1, tức là được nhân đôi:
 $AX=14h=0110b$, BX dịch phải hay chia đôi nó: $BX=0006h=0110b$.

Để tái được định vòng lặp ta dùng lệnh T:

-T

```
AX=000E BX=0006 CX=001C DX=0007 SP=00FA BP=0000 SI=0000 DI=0000  
DS=176F ES=176F SS=1781 CS=177F IP=000B NV UP EI PL NZ AC PE CY  
177F:000B F7C30100 TEST BX,0001
```

Và ta lại thực hiện đến cuối vòng lặp:

-G17

```
AX=000E BX=0006 CX=001C DX=0007 SP=00FA BP=0000 SI=0000 DI=0000  
DS=176F ES=176F SS=1781 CS=177F IP=0017 NV UP EI PL NZ AC PE CY  
177F:0017 75F2 JNZ 000B
```

Bởi vì BX=0006h=110b, lsb của nó bằng 0 nên DX giữ nguyên giá trị. AX dịch trái thành 11100b = 1Ch và BX dịch phải thành 11b = 3h.

Sau hai lần lặp nữa chúng ta thu được tích trong DX. Bạn hãy xem các thanh ghi AX, BX, CX và DX thay đổi như thế nào:

-T

AX=001C BX=01C3 CX=001C DX=0007 SP=00FA BP=0000 SI=0000 DI=0000
DS=176F ES=176F SS=1781 CS=177F IP=000B NV UP EI PL NZ AC PE NC
177F:000B F7C30100 TEST BX,0001

-G17

AX=0038 BX=0001 CX=001C DX=0023 SP=00FA BP=0000 SI=0000 DI=0000
DS=176F ES=176F SS=1781 CS=177F IP=0017 NV UP EI PL NZ AC PE CY
177F:0017 75F2 JNZ 000B

105✓

-T

AX=0038 BX=0001 CX=001C DX=0023 SP=00FA BP=0000 SI=0000 DI=0000
DS=176F ES=176F SS=1781 CS=177F IP=000B NV UP EI PL NZ AC PE CY
177F:000B F7C30100 TEST BX,0001

-G17

AX=0070 BX=0000 CX=001C DX=005B SP=00FA BP=0000 SI=0000 DI=0000
DS=176F ES=176F SS=1781 CS=177F IP=0017 NV UP EI PL ZR AC PE CY
177F:0017 75F2 JNZ 000B

Lần dịch trái cuối cùng làm BX = 0, ZF = 1 và vòng lặp sẽ kết thúc. Tích là 91 = 5bh
chứa trong DX.

Để kết thúc chương trình, chúng ta duyệt qua lệnh JNZ và hai lệnh POP:

-T

AX=0070 BX=0000 CX=001C DX=005B SP=00FA BP=0000 SI=0000 DI=0000
DS=176F ES=176F SS=1781 CS=177F IP=0019 NV UP EI PL ZR AC PE CY
177F:0019 5B POP BX

-T

AX=0070 BX=000D CX=001C DX=005B SP=00FC BP=0000 SI=0000 DI=0000
DS=176F ES=176F SS=1781 CS=177F IP=0019 NV UP EI PL ZR AC PE CY
177F:001A 58 POP AX

-T

AX=0007 BX=000D CX=001C DX=005B SP=00FE BP=0000 SI=0000 DI=0000
DS=176F ES=176F SS=1781 CS=177F IP=0019 NV UP EI PL ZR AC PE CY
177F:001B C3 RET

Hai lệnh POP trả lại các thanh ghi các giá trị nguyên thuỷ của chúng. Ta hãy xem ngắn xếp:

```
-DSS:F0 FF  
1781:00f0 00 00 00 00 00 00 00 6f 17-A4 13 07 00 6F 17 03 00
```

Các giá trị 000Dh và 0007 không còn trong ngắn xếp nữa. Đây không phải là kết quả của lệnh POP mà do DEBUG cũng sử dụng ngắn xếp.

Cuối cùng ta chạy lệnh RET:

```
-T  
AX=0007 BX=000D CX=001C DX=005B SP=0100 BP=0000 SI=0000 DI=0000  
DS=176F ES=176F SS=1781 CS=177F IP=0019 NV UP EI PL ZR AC PE CY  
177F:0003 B44C MOV AH,4C
```

Lệnh RET nạp cho IP địa chỉ trở về chương trình MAIN; IP = 0003. SP cũng được trả về giá trị nguyên thuỷ của nó: SP = 100h. Để kết thúc chương trình chúng ta chỉ cần dùng lệnh G:

```
-G  
Program terminated normally.
```

và ta thoát khỏi DEBUG bằng lệnh Q (quit):

```
-Q  
>C
```

TỔNG KẾT:

- ◆ Ngăn xếp là vùng chứa dữ liệu tạm thời của cả chương trình ứng dụng lẫn hệ điều hành.
- ◆ Ngăn xếp là cấu trúc dữ liệu làm việc theo nguyên tắc vào trước-ra sau. SS:SP trả đến đỉnh ngăn xếp.
- ◆ PUSH, PUSHF, POP và POPF là các lệnh làm thay đổi ngăn xếp. Lệnh PUSH thêm một từ mới vào ngăn xếp còn POP lấy ra đỉnh ngăn xếp. PUSHF cất thanh ghi cờ vào ngăn xếp còn POPF lấy ra đỉnh ngăn xếp đưa vào thanh ghi cờ.
- ◆ SP giảm đi 2 khi thực hiện lệnh PUSH và PUSHF và tăng lên 2 khi thực hiện lệnh POP và POPF. SP được khởi tạo trả đến từ đầu tiên sau đoạn ngăn xếp khi chương trình nạp vào bộ nhớ.
- ◆ Thủ tục là một chương trình con. Một chương trình Hợp ngũ thường chia thành hai thủ tục. Một trong chúng là thủ tục chính, nó chứa điểm xuất phát của chương trình. Mỗi thủ tục có thể gọi các thủ tục khác hay có thể gọi chính nó.
- ◆ Có hai loại thủ tục: NEAR và FAR. Một thủ tục NEAR nằm trong cùng một đoạn với chương trình gọi nó còn thủ tục FAR nằm trong đoạn một khác.
- ◆ Lệnh CALL dùng để gọi các thủ tục. Đối với một thủ tục NEAR, lệnh CALL cất địa chỉ offset của lệnh tiếp theo đồng thời nạp cho IP địa chỉ offset của lệnh đầu tiên của thủ tục.
- ◆ Thủ tục kết thúc bằng lệnh RET. Lệnh này lấy giá trị của IP từ ngăn xếp và điều khiển được trả về chương trình gọi. Để có được địa chỉ trả về đúng, thủ tục phải chắc chắn rằng nó đang ở đỉnh ngăn xếp khi thi hành lệnh RET.
- ◆ Trong Hợp ngũ, các thủ tục thường trao đổi số liệu thông qua các thanh ghi.

Các thuật ngữ tin học:

| | |
|--------------------------------|---|
| direct procedure call | Gọi thủ tục trực tiếp bằng lệnh CALL name |
| FAR procedure | Thủ tục gọi được ở tầm xa: Một thủ tục có thể được gọi bởi thủ tục nằm ở bất kỳ đoạn nào. |
| indirect procedure call | Gọi thủ tục gián tiếp bằng lệnh CALL addr_exp |
| NEAR procedure | Thủ tục gọi gần: Một thủ tục chỉ có thể được gọi bởi thủ tục khác nằm trong cùng đoạn. |
| top of the stack | Đỉnh stack: Từ dữ liệu cuối cùng được cất vào ngăn xếp. |

Các lệnh mới học:

| | | |
|-------------|-------------|--------------|
| CALL | POPF | PUSHF |
| POP | PUSH | RET |

Bài tập:

1. Giả sử đoạn ngắn xếp được khai báo như sau:

.STACK 100h

- SP bằng bao nhiêu khi bắt đầu chương trình (dạng số hex) ?
- Ngắn xếp chứa được nhiều nhất bao nhiêu từ ?

2. Biết rằng AX = 1234h, BX = 5678h, CX = 9ABCCh và SP = 1000h. Hãy cho biết nội dung các thanh ghi AX, BX, CX và SP khi thực hiện các lệnh sau đây:

```
PUSH AX  
PUSH BX  
XCHG AX, CX  
POP CX  
PUSH AX  
POP BX
```

3. Khi ngắn xếp điền đầy vùng cung cấp cho nó thì SP = 0. Nếu có một từ nữa lại được cất vào ngắn xếp thì SP sẽ như thế nào ?, điều gì sẽ xảy ra với chương trình ?

4. Giả thiết chương trình chứa các dòng lệnh sau đây:

```
CALL PROC1  
MOV AX, BX
```

Biết rằng lệnh MOV nằm ở địa chỉ 08FD:0203 và PROC1 là thủ tục NEAR bắt đầu tại địa chỉ 08FD:0300, SP = 010Ah. Cho biết nội dung của IP và SP sau mỗi lệnh. Giá trị đindh ngắn xếp là bao nhiêu ?

5. Giả thiết SP = 0200h và giá trị đindh ngắn xếp là 012Ah. Cho biết nội dung của IP và SP:

- Sau khi thi hành lệnh RET (của một thủ tục NEAR).
- Sau khi thi hành lệnh RET 4 (của một thủ tục NEAR).

6. Viết các lệnh thực hiện:

- a. Đưa giá trị của đindh ngän xép vào AX, không làm thay đổi ngän xép.
 - b. Đưa từ nằm sau đindh ngän xép vào AX, không làm thay đổi ngän xép.
 - c. Đổi chỗ hai từ ở đindh ngän xép. Bạn có thể sử dụng AX và BX.
7. Ngän xép phải được thủ tục trả về chương trình gọi giống hệt như khi nó nhận được. Tuy nhiên cũng tốt nếu như có các thủ tục làm thay đổi ngän xép. Ví dụ chúng ta muốn viết một thủ tục NEAR_SAVE_REGS để cất các thanh ghi BX, CX, DX, SI, DI, BP, DS và ES vào ngän xép. Sau khi cất các thanh ghi, ngän xép như sau:

nội dung ES

nội dung CX

nội dung BX

địa chỉ trả về

Thật không may là bây giờ SAVE_REGS không thể trả về chương trình gọi bởi lẽ địa chỉ trả về không nằm ở đindh ngän xép.

- a. Hãy viết thủ tục SEVE_REGS khắc phục khó khăn nêu trên.
- b. Viết thủ tục RESTORE_REGS phục hồi các thanh ghi được cất bởi chương trình SAVE_SEGS.

Các bài tập lập trình:

8. Viết chương trình để nhập vào các dòng văn bản trong đó các từ ngăn cách nhau bởi các ký tự trắng, kết thúc bằng ký tự trả về đầu dòng và hiển thị chúng theo thứ tự nhưng các chữ trong mỗi từ được đảo ngược. Ví dụ: "this is a text" được đổi thành "siht si a txet". Gợi ý: sửa lại chương trình PGM8_2 trong mục 8.3.
9. Một biểu thức đại số có chứa các dấu ngoặc (như (); []; { }) được coi là hợp lệ nếu thỏa mãn:
- (a) số các dấu ngoặc trái và ngoặc phải của mỗi loại bằng nhau
 - (b) các dấu ngoặc tương ứng phải cùng loại.

Ví dụ:

(a + [b -{ c * (d - e) }] + f) là hợp lệ, nhưng:

(a + [b -{ c * (d - e) }] }+ f) không hợp lệ.

Công việc kiểm tra có thể thực hiện nhờ ngăn xếp. Biểu thức được quét từ trái qua phải. Nếu gặp dấu ngoặc trái ta cất nó vào ngăn xếp. Nếu gặp dấu ngoặc phải ta lấy ra từ ngăn xếp dấu ngoặc nữa (nếu ngăn xếp rỗng tức là có quá nhiều dấu ngoặc phải) và so sánh: nếu cùng loại ta “quét tiếp” còn nếu khác loại có nghĩa là biểu thức được điền các dấu ngoặc không tương ứng. Đến cuối biểu thức nếu ngăn xếp rỗng: biểu thức hợp lệ, nếu ngăn xếp còn dữ liệu tức là có quá nhiều dấu ngoặc trái.

Bạn hãy viết một chương trình để người sử dụng đánh vào một biểu thức đại số có chứa các dấu ngoặc (ngoặc tròn, vuông và nhọn) và kết thúc bằng ký tự về đầu dòng. Trong khi đánh vào biểu thức, chương trình kiểm tra mỗi ký tự. Nếu tại một thời điểm nào đó vào các dấu ngoặc không hợp lệ (quá nhiều dấu ngoặc phải hay các cặp dấu ngoặc tương ứng không cùng loại), chương trình sẽ báo cho người sử dụng bắt đầu lại. Khi ký tự về đầu dòng được đánh vào, nếu biểu thức hợp lệ chương trình sẽ hiển thị thông báo “Biểu thức hợp lệ”, nếu ngược lại, chương trình hiển thị thông báo “quá nhiều ngoặc trái”. Trong cả hai trường hợp chương trình đều hỏi xem người sử dụng có muốn tiếp tục hay không. Nếu trả lời là ‘Y’, chương trình sẽ được thực hiện lại.

Chương trình của bạn không cần phải ghi lại chuỗi vào, chỉ kiểm tra biểu thức có được điền các dấu ngoặc hợp lệ hay không.

Một ví dụ khi chạy chương trình:

Bạn vào một biểu thức đại số:

(a+b) quá nhiều dấu ngoặc phải. Bạn vào lại!

Bạn vào một biểu thức đại số:

[a+(b-c)*d]

Biểu thức hợp lệ.

Nhấn phím ‘Y’ nếu bạn muốn tiếp tục: Y

Bạn vào một biểu thức đại số:

(a+b*(c-d)-e] vào sai kiểu. Bạn vào lại!

Bạn vào một biểu thức đại số:

((a+[b-{c*(d-e)}]+f) có quá nhiều ngoặc trái. Bạn vào lại!)

Bạn vào một biểu thức đại số:

tôi đã vào đủ!

Biểu thức hợp lệ.

Nhấn phím 'Y' nếu bạn muốn tiếp tục:N

Phương pháp sau đây có thể dùng để tạo ra các số ngẫu nhiên trong khoảng từ 1 đến 32767:

Bắt đầu bằng một số bất kỳ trong miền giới hạn.

Dịch trái một bit.

Thay bit 0 bằng kết quả phép XOR bit 14 và bit 15.

Xoá bit 15.

Viết các thủ tục sau đây:

- a. Thủ tục READ để người sử dụng vào một số nhị phân và chứa nó trong AX. Bạn có thể dùng các lệnh nhập các số nhị phân được đưa ra trong mục 7.3.
- b. Thủ tục RANDOM nhận một số vào AX và trả về một số ngẫu nhiên trong AX.
- c. Thủ tục WRITE hiển thị AX dưới dạng nhị phân, Bạn thể sử dụng thuật toán đã nêu trong mục 7.4.

Cuối cùng, bạn hãy viết một chương trình hiển thị một dấu hỏi chấm (?), gọi READ để đọc một số nhị phân sau đó gọi RANDOM và WRITE để tính toán và hiển thị 100 số ngẫu nhiên. Các số được hiển thị thành các dòng 4 số và hai số trong cùng một dòng ngăn cách nhau bằng 4 ký tự trắng.

Chương 9

CÁC LỆNH NHÂN VÀ CHIA

Tổng quan

Trong chương 7 chúng ta đã thấy việc nhân và chia được thực hiện ra sao bằng cách dịch các bit trong một byte hay word. Chúng ta có thể sử dụng các phép dịch trái hoặc phải để nhân hoặc chia cho một luỹ thừa của 2. Trong chương này chúng tôi sẽ giới thiệu các lệnh dùng để nhân hoặc chia cho bất kỳ số nào.

Quá trình nhân và chia là khác nhau đối với các số có dấu và không có dấu, do đó cũng có những lệnh nhân và chia khác nhau cho các số không dấu và có dấu. Ngoài ra các lệnh cũng phân biệt dạng byte và word. Các phần từ 9.1 đến 9.4 sẽ trình bày chi tiết về các lệnh.

Một trong những ứng dụng có ý nghĩa nhất của việc nhân và chia là thực hiện công việc vào ra các số thập phân. Trong phần 9.5 chúng ta sẽ viết các thủ tục thực hiện các thao tác này, ứng dụng này sẽ mở rộng rất nhiều cho khả năng vào ra trong chương trình của bạn.

9.1 Các lệnh MUL và IMUL

Sự khác nhau giữa phép nhân không dấu và có dấu.

Trong phép nhân các số nhị phân, các số có dấu và không dấu phải được “đổi xử” khác nhau. Chẳng hạn chúng ta muốn nhân các số 8 bit 10000000 và 11111111 với nhau. Nếu coi chúng là các số không dấu, chúng lần lượt bằng 128 và 255. Tích số bằng :

$32640 = 011111110000000b$. Nhưng khi xem chúng là các số có dấu, chúng lại bằng -128 và -1 do đó tích số sẽ bằng $128 = 0000000010000000b$.

Vì phép nhân các số không dấu và có dấu dẫn đến những kết quả khác nhau cho nên có 2 lệnh nhân là: MUL (Multiply) cho các số không dấu và IMUL (Integer Multiply) cho các số có dấu. Các lệnh này làm việc với byte hoặc word. Kết quả của phép nhân 2 byte là một word và của 2 word là một từ kép (double word, 32 bit). Cú pháp của các lệnh này như sau:

Và **IMUL** toán hạng nguồn

Dạng byte của các lệnh nhân

Khi nhân các byte với nhau, một số được chứa trong toán hạng nguồn và số còn lại được giả thiết đã chứa trong AL. Toán hạng nguồn có thể là một thanh ghi hay một byte nhớ nhưng không thể là một hằng số.

Khi nhân các số dương (bit msb bằng 0) MUL và IMUL cho cùng một kết quả.

Tác động của MUL và IMUL đến các cờ trạng thái

SF, ZF, AF, PF Không xác định

CF/OF:

Sau lệnh MUL, CF/OF = 0 Nếu nửa cao của kết quả bằng 0

= 1 Trong các trường hợp khác

Sau lệnh IMUL, CF/OF= 0 Nếu nửa cao của kết quả là phần mở rộng dấu của nửa thấp (có nghĩa là các bit của nửa cao giống với bit dấu của nửa thấp).

= 1 Trong các trường hợp khác.

Trong cả 2 lệnh, CF/OF = 1 có nghĩa là kết quả quá lớn để chứa trong nửa thấp của toán hạng đích (AL trong phép nhân với byte và AX trong phép nhân với word).

Các ví dụ

Để mô tả các lệnh MUL và IMUL, chúng ta sẽ làm thử vài ví dụ. Vì thông thường việc nhân các số hex thường rất khó, chúng ta sẽ đoán trước kết quả

bằng cách đổi giá trị hex của số nhân và số bị nhân ra dạng thập phân, thực hiện phép nhân các số thập phân rồi đổi lại kết quả ra dạng hex.

Ví dụ 9.1 Giả sử AX chứa 1 và BX chứa FFFFh :

| Lệnh | Tích số dạng thập phân | Tích số dạng hex | DX | AX | CF/OF |
|---------|------------------------|------------------|------|------|-------|
| MUL BX | 65535 | 0000FFFF | 0000 | FFFF | 0 |
| IMUL BX | -1 | FFFFFFFF | FFFF | FFFF | 0 |

Trong lệnh MUL do DX = 0 nên CF/OF = 0. Trong lệnh IMUL do nội dung của BX dạng có dấu là -1 nên kết quả cũng bằng -1. CF/OF = 0 vì DX là sự mở rộng dấu của AX.

Ví dụ 9.2 Giả sử AX và BX cùng chứa FFFFh:

| Lệnh | Tích số dạng thập phân | Tích số dạng hex | DX | AX | CF/OF |
|---------|------------------------|------------------|------|------|-------|
| MUL BX | 4294836225 | FFFE0001 | FFFE | 0001 | 1 |
| IMUL BX | -1 | 00000001 | 0000 | 0001 | 0 |

Trong lệnh MUL CF/OF = 1 vì DX không bằng 0. Điều này phản ánh thực tế là kết quả FFFE0001 quá lớn để có thể chứa trong AX. Trong lệnh IMUL do AX và BX cùng chứa -1 nên kết quả là 1. DX là phần mở rộng dấu của AX nên CF/OF = 0.

Ví dụ 9.3 Giả sử AX chứa 0FFFh:

| Lệnh | Tích số dạng thập phân | Tích số dạng hex | DX | AX | CF/OF |
|---------|------------------------|------------------|------|------|-------|
| MUL AX | 16769025 | 00FFE001 | 00FF | E001 | 1 |
| IMUL AX | 16769025 | 00FFE001 | 00FF | E001 | 1 |

Vì bit msb của AX bằng 0 nên cả 2 lệnh cho cùng một kết quả. Vì kết quả quá lớn để chứa trong AX nên CF/OF = 1.

Ví dụ 9.4 Giả sử AX chứa 0100h và CX chứa FFFFh

| Lệnh | Tích số dạng thập phân | Tích số dạng hex | DX | AX | CF/OF |
|---------|------------------------|------------------|------|------|-------|
| MUL CX | 16776960 | 00FFFF00 | 00FF | FF00 | 1 |
| IMUL CX | -256 | FFFFFE00 | FFFF | FF00 | 0 |

Trong lệnh MUL tích FFFF00 nhận được bằng cách thêm 2 chữ số 0 vào giá trị của toán hạng nguồn FFFFh. Do tích số quá lớn để có thể chứa trong AX nên CF/OF = 1. Trong lệnh IMUL chứa 256 và CX chứa -1 nên kết quả là -256 và có thể biểu diễn dưới dạng 16 bit là FF00h. DX chứa phần mở rộng của AX nên CF/OF = 0.

Ví dụ 9.5 Giả sử AL chứa 80h còn BL chứa FFh:

| Lệnh | Tích số dạng thập phân | Tích số dạng hex | AH | AL | CF/OF |
|---------|------------------------|------------------|----|----|-------|
| MUL BL | 128 | 7F80 | 7F | 80 | 1 |
| IMUL BL | 128 | 0080 | 00 | 80 | 1 |

Đối với phép nhân các byte kết quả 16 bit được chứa trong AX.

Trong lệnh MUL tích số là 7F80, do 8 bit cao của tích khác 0 nên CF/OF= 1.

Trong lệnh IMUL đây là một trường hợp đáng ngờ. Ta có 80h = -128, FFh = -1 nên tích số là 128 = 0080h. AH không chứa phần mở rộng của AL nên CF/OF = 1. Điều này phản ánh sự thật là AL không chứa kết quả đúng về mặt dấu, vì 80h khi được hiểu là số thập phân có dấu thì bằng -128.

9.2 Các ứng dụng đơn giản của MUL và IMUL

Để các bạn làm quen với việc lập trình với các lệnh MUL và IMUL chúng tôi sẽ chỉ ra ở đây một số thao tác đơn giản được thực hiện ra sao với các lệnh này.

Ví dụ 9.6 Hãy dịch lệnh gán A = 5 * A - 12 * B của ngôn ngữ bậc cao ra các lệnh của hợp ngữ. Coi A và B là các biến kiểu word và giả sử rằng không có hiện tượng tràn xảy ra. Sử dụng lệnh IMUL cho các phép nhân.

Lời giải:

| | | |
|------|--------|--------------------|
| MOV | AX, 5 | ; AX=5 |
| IMUL | A | ; AX=5 * A |
| MOV | A, AX | ; A=5 * A |
| MOV | AX, 12 | ; AX=12 |
| IMUL | B | ; AX=12 * B |
| ADD | A, AX | ; A=5 * A + 12 * B |

Ví dụ 9.7 Viết một thủ tục mang tên FACTORIAL tính $N!$ với N là số nguyên dương. Thủ tục nhận giá trị N trong thanh ghi CX và trả lại kết quả trong thanh ghi AX. Giả thiết không có hiện tượng tràn.

Lời giải:

Ta đã biết định nghĩa: $N! = 1$ nếu $N = 1$

$$= N * (N-1) * (N-2) * \dots * 1 \text{ nếu } N > 1.$$

Dưới đây là thuật toán:

9.3 Các lệnh DIV và IDIV

Mỗi khi thực hiện phép chia chúng ta nhận được 2 kết quả : thương số và số dư. Cũng như đối với phép nhân, có những lệnh riêng biệt cho phép chia không dấu và có dấu. Lệnh DIV (Divide) được dùng cho phép chia không dấu còn lệnh IDIV (Integer Divide) dùng cho các phép chia có dấu. Cú pháp của lệnh như sau:

DIV Số chia
Và .

IDIV Số chia

Các lệnh này chia số 16 hoặc 32 bit cho số 8 bit hoặc 16 bit. Thương số và số dư có cùng kích thước với số chia.

Dạng Byte của các lệnh chia

Trong dạng này số chia là một thanh ghi 8 bit hay một byte bộ nhớ. Số bị chia 16 bit được giả định là đã chứa trong AX. Sau khi thực hiện phép chia thương số 8 bit chứa trong AL và số dư 8 bit chứa trong AH. Số chia không thể là hằng số.

Dạng Word của các tên chia

Trong trường hợp này số chia là thanh ghi 16 bit hay một từ nhớ (16 bit). Số bị chia giả định đã chứa trong DX:AX. Sau khi phép chia được thực hiện thương số 16 bit được chứa trong AX và số dư cũng 16 bit được chứa trong DX. Số chia không thể là hằng số.

Đối với phép chia có dấu, số dư có cùng kích thước với số bị chia. Nếu cả số bị chia và số chia đều dương các lệnh DIV và IDIV cho cùng một kết quả.

Tất cả các cờ trạng thái đều không xác định sau các lệnh DIV và IDIV.

Sự tràn trong phép chia

Rất có thể xảy ra trường hợp là thương số quá lớn để có thể chứa trong toán hạng đích xác định(AL hay AX). Điều này có thể xảy ra khi số chia nhỏ hơn nhiều so với số bị chia. Khi xảy ra điều này chương trình sẽ dừng lại (như các bạn sẽ thấy sau này) và hệ thống đưa ra thông báo “Divide Overflow”.

Ví dụ 9.8 Giả sử AX chứa 0005h, DX chứa 0000h và BX chứa 0002h.

| Lệnh | Thương số dạng thập phân | Số dư dạng thập phân | AX | DX |
|---------|-----------------------------|-------------------------|------|------|
| DIV BX | 2 | 1 | 0002 | 0001 |
| IDIV BX | 2 | 1 | 0002 | 0001 |

Chia 5 cho 2 chúng ta nhận được 2 và dư 1. vì cả số chia và số bị chia đều là các số dương nên 2 lệnh DIV và IDIV cho cùng một kết quả.

Ví dụ 9.9 Giả sử DX chứa 0000h, AX chứa 0005h và BX chứa FFFEh.

| Lệnh | Thương số dạng thập phân | Số dư dạng thập phân | AX | DX |
|---------|-----------------------------|-------------------------|------|------|
| DIV BX | 0 | 5 | 0000 | 0005 |
| IDIV BX | -2 | 1 | FFFE | 0001 |

Đối với lệnh DIV, số bị chia là 5 trong khi số chia là FFFEh = 65534. 5 chia cho 65534 được 0 và dư 5.

Đối với lệnh IDIV số bị chia là 5 trong khi số chia là FFFEh = -2. 5 chia cho -2 được -2 và dư 1.

Ví dụ 9.10 Giả sử DX chứa FFFFh, AX chứa FFFBh và BX chứa 0002.

| Lệnh | Thương số dạng thập phân | Số dư dạng thập phân | AX | DX |
|---------|-----------------------------|-------------------------|------|------|
| IDIV BX | -2 | -1 | FFFE | FFFF |
| IDIV BX | DIVIDE OVERFLOW | | | |

Trong lệnh IDIV, số bị chia là DX:AX = FFFFFFFFh = -5, số chia là BX = 2. -5 chia cho 2 được 2 và dư -1 = FFFFh

Trong lệnh DIV, số bị chia DX:AX = FFFFFFFFh = 4294967291 và số chia là 2 thương số chính xác bằng 2147483646 = 7FFFFFFEh. Số này quá lớn để có thể chứa trong AX, do đó máy tính thông báo DIVIDE OVERFLOW và chương

trình ngừng lại. Ví dụ trên cho thấy điều gì có thể xảy ra khi số chia nhỏ hơn nhiều so với số bị chia.

Ví dụ 9.11 Giả sử AX chứa 00FBh và BL chứa FFh.

| Lệnh | Thương số dạng thập phân | Số dư dạng thập phân | AX | AL |
|---------|--------------------------|----------------------|----|----|
| DIV BL | 0 | 251 | FB | 00 |
| IDIV BL | DIVIDE OVERFLOW | | | |

Trong phép chia cho byte, số bị chia chứa trong AX, Thương số đặt trong AL và số dư đặt trong AH. Đối với lệnh DIV, số bị chia là 00FBh = 251 và số chia là FFh = 256. 251 chia 256 được thương là 0 và dư 251 = FBh. Đối với lệnh IDIV, số bị chia là 00FBh = 251 còn số chia là FFh = -1. Chia 251 cho -1 nhận được thương là -251, số này quá lớn để có thể chứa trong AL do đó thông báo DIVIDE OVERFLOW được đưa ra màn hình.

9.4 Sự mở rộng dấu của số bị chia

Phép chia cho word

Trong phép chia cho word, số bị chia được đặt trong DX:AX ngay cả khi số bị chia có thể chứa vừa vặn trong AX. Trong trường hợp này DX phải được chuẩn bị như sau:

- Đối với lệnh DIV, DX phải được xoá.
- Đối với lệnh IDIV DX phải được làm thành phần dấu mở rộng của AX. Phép mở rộng được thực hiện bằng lệnh CWD (Convert Word to Double word).

Ví dụ 9.12 Chia -1250 cho 7.

Lời giải:

```

MOV AX,-1250    ;AX chứa số bị chia
CWD             ;mở rộng dấu vào DX
MOV BX,7        ;BX chứa số chia
IDIV BX         ;AX chứa thương số DX chứa số dư

```

```
INCLUDE A:PGM9_1.ASM
```

Khi MASM gặp dòng này lúc biên dịch nó sẽ tìm và lấy file PGM9_1.ASM trong đĩa và chèn nó vào trong chương trình tại vị trí của dãy hướng biên dịch INCLUDE.

Dưới đây là chương trình kiểm tra

Chương trình Nguồn PGM9_2.ASM

```
TITLE      PGM9_2:DECIMAL OUTPUT
.MODEL     SMALL
.STACK    100H
.CODE
MAIN      PROC
          CALL OUTDEC
          MOV AH, 4CH
          INT 21H           ;Trở về DOS
MAIN      ENDP
INCLUDE   A:PGM9_1.ASM
END      MAIN
```

Để kiểm tra chương trình chúng ta đánh vào DEBUG và chạy chương trình 2 lần, lần đầu cho AX = -25487 = 9C7h và lần thứ 2 cho AX = 654 = 2E8h:

```
C>DEBUG PGM9_2.EXE
-RAX
AX 0000
:9C71
-G
-25487          (kết quả đầu tiên)
Program terminated normally
-RAX
AX 9C71
:28E
-G
654            (kết quả thứ 2)
```

Chú ý rằng sau lần chạy đầu tiên DEBUG tự động đặt lại con trỏ lệnh IP vào đầu chương trình.

Nhập vào số thập phân

Để nhập vào các số thập phân chúng ta phải đổi chuỗi các chữ số ASCII thành dạng biểu diễn nhị phân của số thập phân. Chúng ta sẽ viết thủ tục INDEC để làm việc này.

Trong thủ tục OUTDEC để đưa ra nội dung của AX dưới dạng thập phân chúng ta đã lặp lại phép chia cho 10. Trong thủ tục INDEC chúng ta sẽ lặp lại phép nhân với 10. Dưới đây là những ý tưởng cơ bản:

Thuật toán nhập các số thập phân (version đầu tiên)

```
Tổng = 0  
Đọc một chữ số ASCII  
REPEAT  
    Đổi ký tự ra giá trị nhị phân  
    Tổng = Tổng*10+giá trị nhận được  
    Đọc ký tự  
UNTIL     Ký tự nhận được là CR
```

Ví dụ việc nhập vào số 123 được thực hiện như sau:

```
Tổng = 0  
Đọc '1'  
Đổi '1' thành 1  
Tổng = 10*0+1 = 1  
Đọc '2'  
Đổi '2' thành 2  
Tổng = 10*1+2 = 12  
Đọc '3'  
Đổi '3' thành 3  
Tổng = 10*12+3 = 123
```

Chúng ta sẽ thiết kế INDEC sao cho nó có thể kiểm soát các số thập phân trong phạm vi từ -32768 đến 32767. Chương trình sẽ in ra dấu hỏi, và để người sử dụng đánh vào dấu tuỳ ý theo sau là chuỗi các chữ số và kết thúc là ký tự CR. Nếu người sử dụng đánh vào một ký tự nằm ngoài khoảng từ "0" đến "9", thủ tục sẽ điều khiển con trỏ tới dòng mới và bắt đầu lại từ đầu. Với những yêu cầu đó, thuật toán trên đây được đổi lại như sau:

Thuật toán nhập các số thập phân (version thứ 2)

In ra dấu hỏi

```

Tổng = 0
âm = false
Đọc ký tự
CASE      ký tự OF
    '-' : âm = true
    Đọc ký tự
    '+' : Đọc ký tự
END_CASE
REPEAT
    IF      Ký tự không nằm trong khoảng giữa '0' và '9'
    THEN
        Nhảy về đầu*
    ELSE
        Đổi các ký tự thành giá trị nhị phân
        Tổng = 10*Tổng+Giá trị
    END_IF
    Đọc ký tự
UNTIL      Ký tự = CR
IF  âm = TRUE
THEN
    Tổng = -Tổng
END_IF

```

Chú ý: Việc nhảy như trên không mang tính chất của "lập trình cấu trúc". Tuy nhiên đôi khi cũng cần thiết phải phá vỡ các nguyên tắc về cấu trúc để đạt được tính hiệu quả, chẳng hạn khi có lỗi xuất hiện.

Từ thuật toán trên chúng ta có thể viết được chương trình như sau:

Chương trình nguồn PGM9_3.ASM

```

1: INDEC          PROC
2: ;Đọc vào một số trong phạm vi từ -32768 đến 32767
3: ;Vào: Không
4: ;Ra : AX = giá trị nhị phân tương đương của số
5:         PUSH BX           ;Cắt các thanh ghi
6:         PUSH CX
7:         PUSH DX
8: ;In ra dấu nhắc
9: @BEGIN:
10:            MOV AH,2
11:            MOV DL,'?'
12:            INT 21H           ;In ra dấu '?'
13: ;Tổng = 0
14:            XOR BX,BX       ;BX chứa Tổng

```

```

15: ;âm = false
16:         XOR CX, CX      ;CX chưa dấu
17: ;Đọc ký tự
18:         MOV AH, 1
19:         INT 21H      ;Ký tự trong AL
20: ;CASE ký tự OF
21:         CMP AL, '-'   ;Dấu âm?
22:         JE @MINUS    ;Đúng! thiết lập dấu
23:         CMP AL, '+'   ;Dấu dương?
24:         JE @PLUS     ;Đúng! Đọc ký tự tiếp theo
25:         JMP @REPEAT2  ;Bắt đầu xử lý các ký tự
26: @MINUS:
27:         MOV CX, 1      ;âm = TRUE
28: @PLUS:
29:         INT 21H      ;Đọc ký tự
30: ;END_CASE
31: @REPEAT2:
32: ;Nếu ký tự nằm trong phạm vi từ '0' đến '9'
33:         CMP AL, '0'    ;Ký tự>='0'?
34:         JNGE @NOT_DIGIT ;Không! ký tự không hợp lệ
35:         CMP AL, '9'    ;Ký tự<='9'?
36:         JNLE @NOT_DIGIT ;Không! ký tự không hợp lệ
37: ;THEN Đổi ký tự thành chữ số
38:         AND AX, 000FH  ;Đổi thành chữ số
39:         PUSH AX      ;Cất vào ngăn xếp
40: ;Tổng = 10*Tổng+Chữ số
41:         MOV AX, 10
42:         MUL BX       ;AX = Tổng*10
43:         POP BX       ;Phục hồi chữ số
44:         ADD BX, AX   ;Tổng = Tổng*10+Chữ số
45: ;Đọc ký tự
46:         MOV AH, 1
47:         INT 21H
48:         CMP AL, 0DH   ;Ký tự CR?
49:         JNE @REPEAT2  ;Không phải! tiếp tục
50: ;UNTIL CR
51:         MOV AX, BX   ;Lưu số vào AX
52: ;IF âm
53:         OR CX, CX   ;Số âm?
54:         JNE JE @EXIT    ;Không! kết thúc
55: ;THEN
56:         NEG AX      ;Đúng
57: ;END_IF
58: @EXIT:

```

Phép chia cho byte

Trong phép chia cho byte, số bị chia đặt trong AX. Trong trường hợp số bị chia thực sự chỉ là một byte, AH cần được chuẩn bị như sau:

- Đối với lệnh DIV, AH cần được xoá.
- Đối với lệnh IDIV, AH cần được làm thành phần dấu mở rộng của AL. Phép mở rộng này được thực hiện bằng lệnh CBW (Convert Byte to Word).

Ví dụ 9.13 Chia giá trị có dấu của biến kiểu byte XBYTE cho -7.

Lời giải:

```
MOV  AL, XBYTE    ;AL chứa số bị chia  
CBW            ;Mở rộng dấu vào AH  
MOV  BL, -7      ;BL chứa số chia  
IDIV BL        ;AL chứa thương số, AH chứa số dư.
```

Các lệnh CBW và CWD không làm ảnh hưởng tới cờ.

9.5 Các thủ tục vào ra với số thập phân

Mặc dù máy tính biểu diễn mọi thứ dưới dạng nhị phân, sẽ vẫn tiện lợi hơn cho người sử dụng khi làm các thao tác vào và ra với số thập phân. Trong phần này chúng ta sẽ viết các thủ tục quản lý việc vào/ra với số thập phân.

Trong thao tác vào, nếu chúng ta đánh 21543 thì thực chất chúng chỉ đưa vào một chuỗi ký tự. Chuỗi ký tự này phải được đổi ra (bằng chương trình) giá trị nhị phân tương đương của số nguyên thập phân 21453. Ngược lại trong thao tác ra, nội dung dạng nhị phân của một thanh ghi phải được đổi thành chuỗi ký tự biểu diễn số nguyên thập phân trước khi được in ra

Việc đưa ra các số thập phân

Chúng ta sẽ viết thủ tục OUTDEC để in ra nội dung của AX như là một số nguyên thập phân có dấu. Nếu AX >0, OUTDEC sẽ in ra dấu âm (-) rồi đổi AX thành - AX (để cho AX chứa số nguyên dương), và sau đó in ra nội dung của nó dạng thập phân. Trong trường hợp còn lại, vấn đề chỉ còn là in ra giá trị thập phân tương đương với một số dương dạng nhị phân. Dưới đây là thuật toán:

Thuật toán cho việc đưa ra số thập phân

1. IF AX<0 /*AX chứa giá trị cần đưa ra*/
2. THEN
3. In ra dấu âm
4. Thay nội dung của AX bằng số bù 2 của nó.
5. END_IF
6. Lấy dạng biểu diễn thập phân các chữ số trong AX
7. Đổi các chữ số này thành các ký tự rồi đưa chúng ra màn hình.

Để thấy được dòng 6 trong thuật toán cần những gì, chúng ta giả sử nội dung của AX dưới dạng thập phân là 24168. Để nhận được từng chữ số trong dạng biểu diễn thập phân chúng ta có thể làm như sau:

Chia 24168 cho 10. Thương số là 2416, số dư là 8

Chia 2416 cho 10. Thương số là 241, số dư là 6

Chia 241 cho 10. Thương số là 24, số dư là 1

Chia 24 cho 10. Thương số là 2, số dư là 4

Chia 2 cho 10. Thương số là 0, số dư là 2.

Như vậy những chữ số mà chúng ta muốn chính là những số dư trong phép chia lặp cho 10. Nhưng chúng lại xuất hiện theo thứ tự ngược lại, để đổi chúng ngược lại chúng ta có thể chia chúng trong ngăn xếp.

Dưới đây là dòng 6 được chia nhỏ ra thành các thao tác cơ bản:

Dòng 6

```
Đếm = 0 /* biến đếm số chữ số thập phân */  
REPEAT  
    Chia số bị chia cho 10  
    Cắt số dư vào ngăn xếp  
    Đếm = Đếm +1  
UNTIL   Thương số bằng 0
```

Trong đó giá trị ban đầu của thương số chính là nội dung của AX. Khi các chữ số đã ở trong ngăn xếp, những gì chúng ta còn phải làm là lấy chúng ra, đổi chúng thành các ký tự và in chúng ra màn hình. Dòng 7 có thể viết lại thành:

Dòng 7

```
FOR     Đếm lần DO  
        Lấy chữ số từ ngăn xếp
```

Đổi nó thành ký tự
 Đưa ra màn hình ký tự.
 END_FOR

Bây giờ chúng ta có thể viết thủ tục như sau

Chương trình Nguồn PGM9_1.ASM

```

1:     OUTDEC PROC
2:     ;in ra nội dung của AX dưới dạng số thập phân có dấu
3:     ;Vào: AX
4:     ;Ra : Không
5:             PUSH AX           ;Cắt các thanh ghi
6:             PUSH BX
7:             PUSH CX
8:             PUSH DX
9:     ;Nếu AX <0
10:    OR AX,AX            ;AX<0?
11:    JGE @END_IF1        ;NO,>0
12:    ;Thì
13:    PUSH AX             ;Cắt số cần đưa ra
14:    MOV DL,'-'          ;Lấy ký tự '-'
15:    MOV AH,2             ;Hàm con in ký tự ra màn hình
16:    INT 21H              ;In dấu '-'
17:    POP AX              ;Lấy lại AX
18:    NEG AX              ;AX=-AX
19:    @END_IF1:
20:    ;Lấy ra các chữ số thập phân
21:    XOR CX,CX           ;CX đếm các chữ số thập phân
22:    MOV BX,10D            ;BX chứa số chia
23:    @REPEAT1:
24:    XOR DX,DX           ;Chuẩn bị word cao cho số
                           ;bị chia
25:    DIV BX              ;AX=thương số, DX=số dư
26:    PUSH DX              ;Cắt số dư vào ngăn xếp
27:    INC CX              ;Đếm=Đếm+1
28:    OR AX,AX              ;Thương số=0?
29:    JNE @REPEAT1         ;Không!, tiếp tục
30:    ;Đổi các chữ số thành ký tự và đưa ra màn hình
31:    MOV AH,2              ;Hàm con in ký tự
32:    ;FOR Đếm lần DO
33:    @PRINT_LOOP:

```

```

35:           POP    DX          ;Chữ số trong DL
36:           OR     DL, 30H      ;Đổi nó thành ký tự
37:           INT    21H          ;In ra màn hình
38:           LOOP   @PRINT_LOOP;Lặp lại cho đến khi xong
39: ;END_FOR
40:           POP    DX          ;Phục hồi các thanh ghi
41:           POP    CX
42:           POP    BX
43:           POP    AX
44:           RET
45: OUTDEC  ENDP

```

Sau khi cất các thanh ghi chúng ta kiểm tra dấu của AX ở dòng 10 bằng cách hoặc AX với chính nó. Nếu AX >0, chương trình nhảy đến dòng 19, nếu AX <0, dấu âm được in ra và AX được thay bằng số bù 2 của nó. Cũng như trong trường hợp còn lại, tại dòng 19, AX chứa số dương.

Tại dòng 21, OUTDEC chuẩn bị cho phép chia. Vì việc chia cho hằng số là không hợp lệ chúng ta phải lưu số chia 10 vào một thanh ghi.

Vòng lặp REPEAT tại các dòng 23-30 sẽ nhận các chữ số và đưa nó vào ngăn xếp. Vì chúng ta làm việc với các số không dấu DX được xoá. Sau phép chia AX chứa thương số và DX chứa số dư (chính xác là số dư ở trong DL vì số dư nằm trong phạm vi từ 0 đến 9). Tại dòng 29, AX được kiểm tra xem đã bằng 0 chưa bằng cách dùng lệnh OR với chính nó. phép chia cho 10 được lặp lại cho đến khi thương số bằng 0

Vòng lặp FOR trong các dòng 34-38 lấy các chữ số trong ngăn xếp và in chúng. Trước khi in ra màn hình chúng phải được đổi ra các ký tự ASCII (dòng 36).

Toán tử giả INCLUDE

Chúng ta có thể kiểm tra lại OUTDEC bằng cách đặt nó bên trong một chương trình và chạy chương trình trong DEBUG. Để chèn OUTDEC vào chương trình mà không phải đánh lại chúng ta sử dụng toán tử giả INCLUDE. Nó có dạng sau:

INCLUDE Tên file

Trong đó tên file xác định file (có thể có cả ổ đĩa và đường dẫn). Ví dụ file chứa OUTDEC là PGM9_1.ASM do đó chúng ta có thể viết:

TỔNG KẾT

Trong chương này chúng ta đã nghiên cứu những vấn đề sau:

- ◆ Các lệnh nhân là MUL cho phép nhân các số không dấu và IMUL cho các số có dấu.
- ◆ Trong phép nhân giữa các byte, AL chứa một số hạng, số hạng còn lại có thể là một thanh ghi 8 bit hay một byte bộ nhớ. Trong phép nhân các word, AX chứa một số hạng, số hạng còn lại có thể là một thanh ghi 16 bit hay một từ bộ nhớ.
- ◆ Trong phép nhân giữa các byte tích số 16 bit được chứa trong AX, còn trong phép nhân các word tích số 32 bit chứa trong DX:AX.
- ◆ Các lệnh chia là DIV cho các số không dấu và IDIV cho các số có dấu.
- ◆ Số chia có thể là một thanh ghi, một byte hay một word. Trong phép chia cho một byte AX chứa số bị chia còn trong phép chia cho một word số bị chia được đặt trong DX:AX
- ◆ Sau phép chia cho byte AH chứa số dư còn AL, chứa thương số. Sau phép chia cho word AX chứa thương số còn DX chứa số dư.
- ◆ Đối với phép chia có dấu cho các word, nếu AX chứa số bị chia thì lệnh CWD được dùng để mở rộng dấu của AX vào DX. Tương tự như vậy trong phép chia có dấu cho các byte lệnh CBW sẽ mở rộng dấu của AL vào AH. Đối với phép chia không dấu cho các word, nếu AX chứa số bị chia thì DX phải được xoá cũng tương tự như vậy trong phép chia không dấu cho các byte, nếu AL chứa số bị chia thì AH phải được xoá.
- ◆ Các lệnh nhân và chia được sử dụng rất hữu hiệu trong việc thực hiện các thao tác vào/ra với số thập phân.
- ◆ Toán tử giả INCLUDE có thể sử dụng để chèn một đoạn văn bản (chương trình) từ một file bên ngoài vào trong chương trình.

Các lệnh mới học

| | | |
|-----|------|-------|
| CBW | DIV | I MUL |
| CWD | IDIV | MUL |

Toán tử giả mới INCLUDE

Bài tập

1. Hãy cho biết nội dung của DX, AX và các cờ CF/OF khi các lệnh sau là hợp lệ và được thực hiện.
 - a. MUL BX trong đó AX chứa 0008h và BX chứa 0003h
 - b. MUL BX trong đó AX chứa 00FFh và BX chứa 1000h
 - c. IMUL CX trong đó AX chứa 0005h và CX chứa FFFFh
 - d. IMUL WORD1 trong đó AX chứa 0005h và WORD1 chứa FFFFh.
 - e. IMUL 10h trong đó AX chứa FFE0h
2. Cho biết nội dung của AX và các cờ CF/OF sau mỗi lệnh sau:
 - a. MUL BX trong đó AL chứa ABh và BL chứa 10h
 - b. IMUL BX trong đó AL chứa ABh và BL chứa 10h
 - c. MUL AH trong đó AX chứa 01ABh
 - d. IMUL BYTE1 trong đó AL chứa 02h và BYTE1 chứa FBh
3. Cho biết nội dung mới của AX và DX sau mỗi lệnh sau, hiện tượng tràn có thể xảy ra hay không?
 - a. DIV BX trong đó DX chứa 0000h, AX chứa 0007h và BX chứa 0002h
 - b. DIV BX trong đó DX chứa 0000h, AX chứa FFFEh và BX chứa 0010h
 - c. IDIV BX trong đó DX chứa FFFFh, AX chứa FFFCh và BX chứa 0003h
 - d. DIV BX với các giá trị giống như câu c
4. Cho biết nội dung mới của AL, AH sau mỗi lệnh sau đây, hiện tượng tràn có thể xảy ra không?
 - a. DIV BL trong đó AX chứa 000Dh và BL chứa 03h
 - b. IDIV BL trong đó AX chứa FFFBh và BL chứa FEh
 - c. DIV BL trong đó AX chứa 00FEh và BL chứa 10h
 - d. DIV BL trong đó AX chứa FFE0h và BL chứa 02h
5. Cho biết nội dung của DX sau khi thực hiện lệnh CWD nếu AX chứa các giá trị:
 - a. 7E02h
 - b. 8ABC_h
 - c. 1ABC_h
6. Cho biết nội dung của AX sau khi thực hiện lệnh CBW nếu AL chứa:
 - a. F0h
 - b. 5Fh
 - c. 80h
7. Viết các lệnh hợp ngữ thực hiện các lệnh gán sau đây trong ngôn ngữ bậc cao. Giả thiết rằng A, B, C là các biến kiểu word và tất cả các tích số đều chứa vừa trong 16

bit. Sử dụng lệnh IMUL trong các phép nhân. Không cần giữ lại nội dung của các biến A, B, C.

a. $A = S * A - 7$
b. $B = (A-B) * (B+10)$
c. $A = 6 - 9 * A$
d. IF $A' + B' = C'$
THEN
 Thiết lập CF
ELSE
 Xóa CF
END_IF

Bài tập lập trình

Chú ý: Một số bài tập dưới đây yêu cầu bạn sử dụng IDEC hay OUTDEC hoặc cả 2 để thực hiện các thao tác vào ra. Hãy chú ý không được dùng những nhãn đã dùng trong các thủ tục này nếu không bạn sẽ nhận được thông báo lỗi dùng nhãn 2 lần. Điều này có thể thực hiện dễ dàng vì các nhãn của những thủ tục này đều bắt đầu bằng chữ "@".

8. Sửa lại thủ tục INDEC để nó có khả năng kiểm tra hiện tượng tràn.
9. Viết một chương trình cho phép người sử dụng đánh vào thời gian ở dạng giây nhỏ hơn hay bằng 65535, sau đó đưa ra thời gian dạng giờ, phút, giây. Sử dụng INDEC và UTDEC cho các thao tác vào/ra.
11. Viết một chương trình cho phép người sử dụng đánh vào một phân số dạng M/N($M < N$). Sau đó chương trình in ra thương số đến N chữ số thập phân theo thuật toán sau:
 1. In ra dấu ":"
 - Thực hiện các bước sau N lần:
 2. Chia 10^*M cho N, nhận được thương số Q và số dư R.
 3. In ra Q.
 4. Thay M bằng R và quay lại bước 2.Sử dụng INDEC để đọc vào M và N
12. Viết một chương trình tìm ước số chung lớn nhất (USCLN) của 2 số nguyên M và N theo thuật toán sau đây:
 1. Chia M cho N, nhận được thương số Q và số dư R.
 2. Nếu $R=0$ thì dừng lại. N là USCLN của M và N.
 3. Nếu $N > 0$, thay M bằng N, N bằng R và lặp lại bước 1.

Sử dụng INDEC để nhập các số M và N, OUTDEC để đưa ra màn hình USCLN của chúng.

Chương 10

MẢNG VÀ CÁC CHẾ ĐỘ ĐỊA CHỈ

Tổng quan.

Đôi khi trong các ứng dụng chúng ta cần phải làm việc với tập hợp các ký tự như là một nhóm. Ví dụ chúng ta cần đọc vào các điểm kiểm tra và in ra trị trung bình. Để làm việc này trước hết ta phải lưu trữ các điểm theo thứ tự tăng dần (đồng thời với việc nhập các điểm hoặc là chúng đã có sẵn trong bộ nhớ). Ưu điểm của việc sử dụng mảng để chứa dữ liệu là có thể đưa ra tên cho cả cấu trúc và mỗi phần tử có thể được truy nhập bằng cách cung cấp một chỉ số.

Mục 10.1 chúng tôi sẽ trình bày cách khai báo mảng một chiều trong Hợp ngữ. Để truy nhập các phần tử, mục 10.2 chúng tôi sẽ giới thiệu các phương pháp mới định nghĩa các toán hạng - các chế độ địa chỉ cơ sở, chỉ số và chế độ địa chỉ gián tiếp thông qua các thanh ghi. Đến mục 10.3 chúng tôi sẽ sử dụng các chế độ địa chỉ này để sắp xếp một mảng.

Mảng hai chiều là mảng một chiều mà mỗi phần tử lại là một mảng một chiều (mảng của các mảng). Trong mục 10.4 chúng tôi sẽ trình bày cách thức lưu trữ chúng. Các mảng này có hai chỉ số, chúng được tính toán thuận lợi hơn cả trong chế độ địa chỉ chỉ số cơ sở mà chúng tôi sẽ trình bày trong mục 10.5. Mục 10.6 sẽ cung cấp một ứng dụng đơn giản.

Mục 10.7 giới thiệu về lệnh XLAT (translate). Lệnh này có ích khi muốn đổi dữ liệu. Chúng ta sẽ sử dụng nó để mã hoá và giải mã một mật khẩu.

10.1. Mảng một chiều.

Mảng một chiều là một dãy có thứ tự các phần tử có cùng một kiểu.

“Thứ tự” ở đây có nghĩa là có phần tử thứ nhất, phần tử thứ hai, thứ ba v.v... Trong toán học các phần tử của mảng A thường được ký hiệu là A[1], A[2], A[3]... . Hình 10.1 biểu thị mảng A với 6 phần tử.

```

59:          POP   DX           ;Phục hồi các thanh ghi
60:          POP   CX
61:          POP   BX
62:          RET           ;Và trở về
63: ;Nếu không phải ký tự hợp lệ
64: @NOT_DIGIT:
65:          MOV   AH,2       ;Chuyển con trỏ đến dòng mới
66:          MOV   DL,0DH
67:          INT   21H
68:          MOV   DL,0AH
69:          INT   21H
70:          JMP   @begin     ;Làm lại từ đầu
71: INDEC    ENDP

```

Thủ tục bắt đầu bằng việc cất các thanh ghi và in ra dấu hỏi chấm. BX chưa tổng, ở dòng 14 được xoá đi để khởi tạo.

CX sử dụng để theo dõi dấu; 0 có nghĩa là số dương và 1 có nghĩa là số âm. Ban đầu chúng ta giả sử số là dương do đó CX được xoá ở dòng 16.

Ký tự đầu tiên được đọc vào ở dòng 18,19. Nó có khả năng là "+" hay "-" hay là một chữ số. Nếu nó là dấu, CX được thay đổi nếu cần thiết và ký tự khác được đọc (dòng 29) giả sử ký tự tiếp theo là một chữ số.

Tại dòng 31 INDEC bước vào vòng lặp REPEAT, vòng lặp này xử lý ký tự hiện thời và đọc vào ký tự tiếp theo cho đến khi người sử dụng đánh ENTER.

Tại dòng 33-36 INDEC kiểm tra xem ký tự hiện thời có phải là một chữ số không, nếu không phải thủ tục nhảy đến nhãn @NOT_DIGIT (dòng 64), chuyển con trỏ đến đầu dòng mới và nhảy đến nhãn @BEGIN. Điều này có nghĩa là người sử dụng không thể thoát khỏi chương trình mà không đánh vào một số hợp lệ.

Nếu ký tự hiện thời trong AL là một chữ số thập phân, nó được đổi sang trị nhị phân (dòng 38). Tiếp theo giá trị đó được cất vào ngăn xếp (dòng 39), vì AX được dùng đến trong phép nhân tổng với 10.

Tại dòng 41 và 42, tổng trong BX được nhân với 10. Tích số sẽ được chứa trong DX:AX . Tuy nhiên DX sẽ chứa 0 trừ khi số đánh vào nằm ngoài phạm vi (chúng ta sẽ nói thêm về chúng sau này). Tại dòng 43, giá trị đã cất vào ngăn xếp sẽ được đưa ra và cộng với 10 lần tổng.

Tại dòng 51, INDEC thoát khỏi vòng lặp REPEAT với số nhận được trong BX. Sau khi chuyển nó vào AX, INDEC kiểm tra dấu trong CX. Nếu CX chứa 1, AX được đảo dấu sau khi thủ tục kết thúc.

Kiểm tra INDEC

Chúng ta có thể kiểm tra INDEC bằng cách viết một chương trình sử dụng INDEC cho việc nhập và OUTDEC cho việc xuất.

Chương trình Nguồn PGM9_4.ASM

```
TITLE      PGM9_4 : DECIMAL I/O
.MODEL     SMALL
.STACK
.CODE
MAIN      PROC
;Nhập một số
        CALL INDEC      ;Số trong AX
        PUSH AX         ;Cất số vào ngăn xếp
;Chuyển con trỏ sang dòng mới
        MOV  AH, 2
        MOV  DL, 0DH
        INT  21H
        MOV  DL, 0AH
        INT  21H
;Đưa ra số đã nhập
        POP  AX         ;Phục hồi số đã nhập
        CALL OUTDEC
;Trở về DOS
        MOV  AX, 4CH
        INT  21H
MAIN      ENDP

INCLUDE  A:PGM9_1.ASM      ;Bao gồm cả thủ tục INDEC
INCLUDE  A:PGM9_3.ASM      ;Bao gồm cả thủ tục OUTDEC

END      MAIN
```

Ví dụ thực hiện

```
C>PGM9_4
?21345
21345Overflow
```

Hiện tượng tràn

Thủ tục INDEC có thể quản lý việc vào cổ những ký tự không hợp lệ, nhưng nó không thể quản lý việc đưa ra giá trị nằm ngoài khoảng -32768 đến 32767. Chúng ta gọi chúng là hiện tượng tràn khi nhập.

Hiện tượng tràn có thể xảy ra ở 2 chỗ trong thủ tục INDEC:

- * Khi Tổng được nhân với 10
- * Khi Giá trị mới được cộng vào Tổng.

Để lấy ví dụ cho trường hợp tràn đầu tiên người sử dụng có thể đánh vào 99999. Hiện tượng tràn sẽ xuất hiện khi Tổng = 9999 được nhân với 10. Để lấy ví dụ cho trường hợp tràn thứ 2 người sử dụng có thể đánh vào 32769, khi đó lúc Tổng = 32760, hiện tượng tràn xuất hiện khi 9 được cộng vào Tổng. Thuật toán có thể thay đổi để kiểm tra tràn như sau:

Thuật toán nhập số thập phân (version 3)

```
In ra dấu hỏi chấm
Tổng = 0
Âm = false
Đọc 1 ký tự
CASE ký tự OF
    '-' : Âm = TRUE
    '+' : Đọc ký tự
END_CASE
REPEAT
    IF ký tự không nằm trong khoảng giữa '0' và '9'
        THEN
            Quay lại từ đầu
        ELSE
            Đổi ký tự thành giá trị số
            Tổng = 10*Tổng
            IF overflow
                THEN
                    Quay lại từ đầu
                ELSE
                    Tổng = Tổng + Giá trị
                    IF overflow
                        THEN
                            Quay lại từ đầu
```

```
        END_IF  
    END_IF  
    Đọc ký tự  
UNTIL Ký tự = CR  
IF Âm = TRUE  
THEN  
    Tổng = -Tổng  
END_IF
```

Việc lập chương trình từ thuật toán này chúng tôi giành cho các bạn như là một bài tập.

Hình 10.1. Mảng một chiều A

Chỉ số

| | |
|---|------|
| 1 | A[1] |
| 2 | A[2] |
| 3 | A[3] |
| 4 | A[4] |
| 5 | A[5] |
| 6 | A[6] |

Trong chương 4 chúng ta đã sử dụng các toán tử giả DB và DW để khai báo các mảng byte và word. Ví dụ chuỗi 5 ký tự MSG:

MSG DB 'akudie'

hay một mảng word W gồm 6 số nguyên được khởi tạo với các giá trị 10, 20, 30, 40, 50, 60:

W DW 10, 20, 30, 40, 50, 60

Địa chỉ của biến mảng gọi là địa chỉ cơ sở của mảng (base address of the array). Nếu như địa chỉ offset gán cho W bằng 0200h thì nó sẽ được phân bố trong bộ nhớ như sau:

| Địa chỉ offset | Ký hiệu địa chỉ | Giá trị thập phân |
|----------------|-----------------|-------------------|
| 0200h | W | 10 |
| 0202h | W+2h | 20 |
| 0204h | W+4h | 30 |
| 0206h | W+6h | 40 |
| 0208h | W+8h | 50 |
| 020Ah | W+Ah | 60 |

Toán tử DUP.

Chúng ta có thể định nghĩa một mảng mà các phần tử có cùng một giá trị khởi đầu nhờ toán tử DUP (duplicate). Nó có dạng:

repeat_count DUP (value)

Với toán tử này, value sẽ được lặp lại một số lần xác định bởi repeat_count. Ví dụ:

GAMMA DW 100 DUP (0)

thiết lập một mảng 100 từ với giá trị khởi đầu cho mỗi phần là 0. Tương tự:

DELTA DB 212 DUP (?)

khai báo một mảng 212 phần tử không khởi tạo. Các toán tử DUP có thể lồng nhau. Ví dụ:

LINE DB 5, 4, 2 DUP (2, 3 DUP (0), 1)

tương đương với:

LINE DB 5, 4, 2, 0, 0, 0, 1, 2, 0, 0, 0, 1

Xác định vị trí các phần tử trong mảng.

Địa chỉ một phần tử của mảng có thể được xác định bằng cách cộng một hằng số vào địa chỉ cơ sở. Giả sử mảng A có S là số byte của một phần tử (đối với mảng byte S = 1 còn mảng word S = 2). Vị trí các phần tử của mảng A được xác định như sau:

| Số thứ tự | Vị trí |
|-----------|-------------------|
| 1 | A |
| 2 | A + 1 * S |
| 3 | A + 2 * S |
| . | . |
| N | A + (N - 1) * S |

Ví dụ 10.1. Hoán chuyển các phần tử thứ 10 và thứ 25 của mảng word W.

Lời giải:

$W[10]$ ở tại địa chỉ $W + 9 * 2 = W + 18$ và $W[25]$ tại $W + 24 * 2 = W + 48$, vì vậy ta có thể đổi chỗ như sau:

```
MOV      AX,W + 18      ; AX chứa W[10]
XCHG    W + 48,AX      ; AX chứa W[25]
MOV      W + 18,AX      ; hoàn thành việc hoán chuyển.
```

Trong rất nhiều ứng dụng chúng ta cần phải thực hiện các thao tác với từng phần tử của một mảng. Ví dụ mảng A có 10 phần tử và chúng cần tính tổng các phần tử của nó. Trong một ngôn ngữ bậc cao chúng ta có thể thực hiện như sau:

```
sum = 0
N = 1
REPEAT
    sum = sum + A[N]
    N = N + 1
UNTIL N > 10
```

Trong Hợp ngữ , để làm việc này chúng ta cần có một phương pháp di chuyển từ phần tử này đến phần tử tiếp theo của mảng. Trong mục tiếp theo chúng tôi sẽ chỉ ra điều đó được thực hiện như thế nào nhờ việc đánh địa chỉ gián tiếp.

10.2. Các chế độ địa chỉ.

Phương thức xác định toán hạng gọi là chế độ địa chỉ (addressing mode) của nó. Các chế độ địa chỉ chúng ta đã sử dụng đó là:

- (1) Chế độ địa chỉ thanh ghi (register mode): toán hạng là thanh ghi.
- (2) Chế độ địa chỉ tức thời (immediate mode): toán hạng là hằng số.
- (3) Chế độ địa chỉ trực tiếp (direct mode): toán hạng là biến nhớ.

Ví dụ:

```
MOV AX,0      ;toán hạng đích AX ở chế độ
               ;thanh ghi,toán hạnh nguồn 0
               ;ở chế độ tức thời.
ADD ALPHA,AX ;toán hạng đích ALPHA ở chế độ
               ;trực tiếp,toán hạnh nguồn AX
               ;ở chế độ thanh ghi.
```

Bộ vi xử lý 8086 có tổng cộng 4 chế độ địa chỉ, đó là các chế độ địa chỉ thanh ghi, cơ sở, chỉ số và chỉ số cơ sở. Các chế độ này được sử dụng để đánh địa chỉ ở những các toán hạng một cách gián tiếp. Trong phần này chúng tôi sẽ trình bày ba chế độ đầu, chúng hữu ích khi thao tác với các mảng một chiều. Chế độ địa chỉ chỉ số cơ sở có thể sử dụng cho mảng hai chiều, nó sẽ được trình bày trong mục 10.5.

10.2.1. Chế độ địa chỉ gián tiếp thanh ghi.

Trong chế độ này, địa chỉ offset của toán hạng được chứa trong một thanh ghi. Như vậy các thanh ghi đóng vai trò như một con trỏ trỏ đến các ô nhớ. Khuôn dạng của toán hạng ở chế độ này là:

[register]

Register có thể là BX, DI, SI hay BP. Với các thanh ghi BX, DI và SI, số hiệu đoạn của toán hạng được chứa trong DS. Với thanh ghi BP, SS chứa số hiệu đoạn.

Ví dụ SI chứa địa chỉ offset 0100h và word tại địa chỉ 0100h có giá trị 1234h. Khi thi hành:

MOV AX, [SI]

CPU sẽ kiểm tra SI để suy ra địa chỉ từ nhớ là DS : 0100h, sau đó nó chuyển nội dung của từ này, tức là 1234h vào AX. Điều đó khác hẳn với:

MOV AX, SI

chỉ đơn thuần chuyển giá trị của SI hay 100h vào AX.

Ví dụ 10.2. Giá sử:

BX chứa 100h
SI chứa 2000h
DI chứa 3000h

Offset 1000h chứa 1BACh
Offset 2000h chứa 20FBh
Offset 3000h chứa 031Dh

Ở đây các địa chỉ offset trên nằm trong đoạn dữ liệu đánh địa chỉ bởi DS.

Hãy cho biết lệnh nào sau đây là hợp lệ. Nếu hợp lệ hãy đưa ra địa chỉ offset của toán hạng nguồn và kết quả hoặc số được dịch chuyển.

a. MOV BX, [BX]

- b. MOV CX, [SI]
- c. MOV BX, [AX]
- d. ADD [SI], [DI]
- e. INC [DI]

Lời giải:

| | offset toán hạng nguồn | kết quả |
|----|--------------------------------------|-----------------------|
| a. | 1000h | 1BACh |
| b. | 2000h | 20FEh |
| c. | thanh ghi nguồn không hợp lệ | phải là BX, SI hay DI |
| d. | phép cộng không hợp lệ ô nhớ - ô nhớ | |
| e. | 3000h | 031Eh |

Bây giờ chúng ta hãy trả lại vấn đề tính tổng các phần tử của một mảng.

Ví dụ 10.3. Viết các lệnh đưa vào AX tổng của các phần tử của một mảng 10 phần tử W được định nghĩa như sau:

W DW 10,20,30,40,50,60,70,80,90,100

Lời giải:

Ta thiết lập một con trỏ trỏ đến địa chỉ cơ sở của mảng, cho nó dịch chuyển đến hết lượt các phần tử của mảng đồng thời cộng vào tổng các phần tử được tham trỏ tới.

| | | |
|---------|----------|---|
| XOR | AX, AX | ;AX chứa tổng |
| LEA | SI, W | ;SI trỏ đến mảng W |
| MOV | CX, 10 | ;CX chứa số các phần tử |
| ADDNOS: | | |
| ADD | AX, [SI] | ;tổng=tổng+phần tử |
| ADD | SI, 2 | ;chuyển con trỏ đến phần tử ;tiếp theo |
| LOOP | ADDNOS | ;lặp đến khi thực hiện xong |

Ta phải cộng 2 vào SI sau mỗi bước lặp bởi lẽ W là một mảng word (xem lại chương 4, lệnh LEA chuyển địa chỉ offset của toán hạng nguồn vào toán hạng đích).

Ví dụ tiếp theo sẽ chỉ ra chế độ địa chỉ gián tiếp thanh ghi được sử dụng như thế nào khi xử lý các mảng.

Ví dụ 10.4. Viết thủ tục REVERSE để đảo ngược một mảng N từ. Điều này có nghĩa là từ thứ N trở thành từ đầu tiên, từ thứ N-1 trở thành từ thứ hai ... và từ đầu tiên trở thành từ thứ N. Trước khi vào thủ tục, SI trỏ đến mảng, BX chứa số phần tử N.

Lời giải:

Mục đích của ta là đổi chỗ các từ thứ nhất và thứ N, từ thứ hai và thứ (N-1) v.v... Số lần hoán chuyển sẽ bằng $N/2$ (làm tròn xuống nếu như N lẻ). Trong mục 10 ta đã biết rằng phần tử thứ N của mảng A có địa chỉ $A + 2 * (N - 1)$.

Chương trình nguồn PGR10_1.ASM

```
REVERSE PROC
;đảo ngược một mảng
;vào:    SI=địa chỉ offset của mảng
;         BX=số phần tử
;ra:     đảo ngược mảng
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH SI
        PUSH DI
;cho DI trỏ đến phần tử thứ N
        MOV DI,SI ;DI trỏ đến phần tử đầu tiên
        MOV CX,BX      ;CX=N
        DEC BX          ;BX=N-1
        SHL BX,1        ;BX=2*(N-1)
        ADD DI,BX       ;DI trỏ đến phần tử thứ N
        SHR CX,1        ;CX=N/2=số lần hoán chuyển
                      ;hoán chuyển

XCHG_LOOP:
        MOV AX,[SI]      ;lấy một phần tử trong nửa
                          ;đầu của mảng
        XCHG AX,[DI]      ;chèn nó vào nửa sau
        MOV [SI],AX        ;hoàn thành hoán chuyển
        ADD SI,2           ;di chuyển con trỏ
        SUB DI,2           ;di chuyển con trỏ
        LOOP XCHG_LOOP   ;lặp đến khi hoán chuyển hết
        POP DI
```

```

    POP    SI
    POP    CX
    POP    BX
    POP    AX
    RET
REVERSE ENDP

```

10.2.2. Chế độ địa chỉ cơ sở và địa chỉ chỉ số.

Trong các chế độ địa chỉ này, địa chỉ offset của các toán hạng nhận được bằng cách cộng một số được gọi là độ dịch (displacement) với nội dung của một thanh ghi, trong đó độ dịch có thể là:

- địa chỉ offset của một biến.
- một hằng số (âm hoặc dương).
- địa chỉ offset của một biến cộng hoặc trừ với một hằng số.

Ví dụ các độ dịch có thể là (A là một biến):

```

A
-2
A+4

```

Toán hạng phải được viết tương đương với một trong các biểu thức sau đây:

```

[ thanh ghi + độ dịch ]
[ độ dịch + thanh ghi ]
[ thanh ghi ] + độ dịch
độ dịch + [ thanh ghi ]
độ dịch [ thanh ghi ]

```

Các thanh ghi phải là BX, BP, SI hay DI. Nếu ta dùng BX, SI hay DI, DS sẽ chứa số hiệu đoạn của địa chỉ toán hạng. Nếu BP được sử dụng, SS sẽ chứa số hiệu đoạn. Chế độ địa chỉ được gọi là chế độ địa chỉ cơ sở (based) nếu ta dùng BX (base register) hay BP (base pointer) và gọi là chế độ địa chỉ chỉ số nếu SI (source indexed) hay DI (destination indexed) được sử dụng.

Ví dụ. Giả sử W là một mảng word, BX chứa số 4. Trong lệnh sau đây:

```
MOV AX, W[BX]
```

độ dịch là địa chỉ offset của biến W. Lệnh MOV sẽ chuyển phần tử có địa chỉ W+4 vào AX. Đây chính là phần tử thứ 3 của mảng. Lệnh trên còn có thể được viết dưới các dạng sau :

```
MOV AX, [W+BX]
MOV AX, [BX+W]
MOV AX, W+[BX]
MOV AX, [BX]+W
```

Ta xét tiếp một ví dụ khác. Giả sử SI chứa địa chỉ của mảng word W. Trong lệnh:

```
MOV AX, [SI+2]
```

độ dịch bằng 2. Lệnh trên sẽ chuyển phần tử có địa chỉ W+2 vào AX. Đây chính là phần tử thứ hai của mảng. Ta có thể viết lại lệnh dưới các dạng sau đây:

```
MOV AX, [2+SI]
MOV AX, 2+[SI]
MOV AX, [SI]+2
MOV AX, 2[SI]
```

Ví dụ 10.5. Làm lại ví dụ 10.3 bằng cách sử dụng chế độ địa chỉ cơ sở.

Lời giải.

Phương hướng ở đây là xoá thanh ghi BX, cộng thêm vào nó 2 sau mỗi lần lặp:

```
XOR AX,AX      ;AX chứa tổng
XOR BX,BX      ;xoá thanh ghi cơ sở
MOV CX,10      ;CX chứa số phần tử
ADDNOS:
    ADD AX,W[BX]   ;tổng=tổng+phần tử
    ADD BX,2        ;chỉ số của phần tử tiếp theo
    LOOP ADDNOS    ;lặp đến khi làm xong
```

Ví dụ 10.6. Giả sử ALPHA được khai báo như sau:

```
ALPHA DW 0123h, 0456h, 0789h, 0ABCCh.
```

trong đoạn được đánh địa chỉ bởi DS. Biết rằng:

| | |
|-----------|------------------------|
| BX chứa 2 | Offset 0002 chứa 1084h |
| SI chứa 4 | Offset 0004 chứa 2BACH |
| DI chứa 1 | |

Hỏi rằng lệnh nào trong các lệnh sau đây là hợp lệ. Với các lệnh hợp lệ hãy cho biết địa chỉ offset của toán tử nguồn và con số được dịch chuyển.

- a. MOV AX, [ALPHA + BX]
- b. MOV BX, [BX + 2]
- c. MOV CX, ALPHA [SI]
- d. MOV AX, -2{SI}
- e. MOV BX, [ALPHA + 3 + DI]
- f. MOV AX, [BX]2
- g. ADD BX, [ALPHA + AX]

Lời giải:

| Offset toán hạng nguồn | Con số được dịch chuyển |
|--------------------------------------|-------------------------|
| a. ALPHA + 2 | 0456h |
| b. 2 + 2 = 4 | 2BACH |
| c. ALPHA + 4 | 0789h |
| d. -2 + 4 = 2 | 1084h |
| e. ALPHA + 3 + 1 = ALPHA + 4 | 0789h |
| f. dạng toán hạng nguồn không hợp lệ | |
| g. thành ghi nguồn không hợp lệ | |

Ví dụ tiếp theo sẽ minh họa cho việc xử lý mảng bằng chế độ địa chỉ chỉ số và địa chỉ cơ sở.

***Ví dụ 10.7.** Thay thế các chữ thường trong chuỗi sau đây bằng các chữ hoa tương ứng. Bạn hãy dùng chế độ địa chỉ chỉ số.

```
MSG    DB      ' this is a messsage '
```

Lời giải:

```

MOV  CX,17          ;số ký tự trong chuỗi
XOR  SI,SI          ;SI chỉ đến một ký tự
TOP:   CMP  MSG[SI], ' '   ;ký tự trắng?
       JE   NEXT        ;đúng! nhảy qua
       AND  MSG[SI], 0DFh ;không đổi thành chữ hoa
NEXT:  INC  SI          ;chỉ đến byte tiếp theo

```

LOOP TOP

;lặp đến khi hoàn thành

10.2.3. Các toán tử PTR và LABEL.

Hẳn các bạn còn nhớ trong chương 4 đã chỉ ra rằng các toán hạng trong một lệnh phải có cùng một kiểu, ví dụ cùng là byte hoặc word. Nếu một toán hạng là hằng số, chương trình biên dịch sẽ căn cứ vào toán hạng kia để suy ra dạng của nó. Ví dụ chương trình biên dịch xem lệnh:

MOV AX, 1

như một lệnh thao tác trên các word bởi vì AX là thanh ghi 16 bit. Tương tự nó coi:

MOV BH, 5

là lệnh thao tác với các byte. Tuy nhiên nó không thể biên dịch:

MOV [BX], 1 ;không hợp lệ

bởi lẽ nó không thể biết được BX chỉ đến byte hay word. Nếu như bạn muốn toán hạng đích là một byte, bạn có thể viết:

MOV BYTE PTR [BX], 1

còn nếu bạn lại muốn toán hạng đích là một word, bạn hãy viết như sau:

MOV WORD PTR [BX], 1

Ví dụ 10.8. Thay ký tự 't' của chuỗi đã cho trong ví dụ 7 bằng ký tự 'T'.

Lời giải 1:

Sử dụng chế độ địa chỉ gián tiếp thanh ghi.

LEA SI, MSG ;SI chỉ đến MSG
MOV BYTE PTR [SI], 'T'; thay 't' bằng 'T'

Lời giải 2:

Sử dụng chế độ địa chỉ chỉ số.

XOR SI, SI ;xoá SI
MOV MSG[SI], 'T' ;thay 't' bằng 'T'

Lời giải này không cần thiết phải dùng toán tử PTR bởi lẽ MSG đã được định nghĩa là một biến byte.

Dùng PTR để định lại kiểu.

Nói chung PTR có thể dùng để định lại kiểu đã khai báo của một biểu thức địa chỉ. Cú pháp như sau:

```
type PTR address_expression
```

Trong đó type có thể là BYTE, WORD hay DWORD (doubleword), còn biểu thức địa chỉ có kiểu DB, DW hay DD.

Ví dụ bạn có khai báo sau đây:

| | | |
|---------|----|-----|
| DOLLARS | DB | 1Ah |
| CENTS | DB | 52h |

và bạn muốn chuyển giá trị của DOLLARS vào AL, giá trị của CENTS vào AH chỉ bằng một lệnh duy nhất. Lúc này:

```
MOV AX, DOLLARS ; không hợp lệ
```

là không hợp lệ bởi vì toán hạng đích là một word trong khi toán hạng nguồn được định kiểu như là một byte.Trong trường hợp này bạn có thể định lại kiểu khai báo nhờ WORD PTR như sau:

```
MOV AX, WORD PTR DOLLARS ; AL=dollars, AH=cents
```

Lệnh này sẽ chuyển 521Ah vào AX.

Toán tử giả LABEL.

Thực chất đây là một phương pháp khác để giải quyết mâu thuẫn về kiểu trong ví dụ trước. Sử dụng toán tử giả LABEL, chúng ta có thể khai báo:

| | |
|---------|------------|
| MONEY | LABEL WORD |
| DOLLARS | DB 1Ah |
| CENTS | DB 52h |

Khai báo này ấn định MONEY là một biến word, DOLLARS và CENTS là các biến byte, và MONEY và DOLLARS được gán cho cùng một địa chỉ bởi trình biên dịch. Như vậy:

```
MOV AX, MONEY ; AL-dollars, AH-cents
```

là hợp lệ. Lệnh này cho kết quả giống hệt như:

```
MOV AL, DOLLARS  
MOV AH, CENTS
```

Ví dụ 10.9. Cho các dữ liệu đã được khai báo như sau:

```
.DATA  
A DW 1234H  
B LABEL BYTE  
DW 5678h  
C LABEL WORD  
C1 DB 9Ah  
C2 DB 0BC9h
```

Hỏi rằng lệnh nào trong các lệnh sau đây là hợp lệ. Với các lệnh hợp lệ hãy cho biết địa chỉ offset của toán tử nguồn và con số được dịch chuyển.

Các lệnh:

- a. MOV AX, B
- b. MOV AH, B
- c. MOV CX, C
- d. MOV BX, WORD PTR B
- e. MOV DL, WORD PTR C
- f. MOV AX, WORD PTR C1

Lời giải:

- a. không hợp lệ - sai kiểu.
- b. hợp lệ, 78h.
- c. hợp lệ, 0BC9Ah.
- d. hợp lệ, 5678h.
- e. hợp lệ, 9Ah.
- f. hợp lệ, 0BC9Ah

10.2.4. Phương pháp ghi rõ đoạn.

Trong chế độ địa chỉ gián tiếp thanh ghi, các con trỏ thanh ghi BX, SI và DI xác định địa chỉ offset ứng với địa chỉ đoạn trong DS. Nhưng chúng ta cũng có thể xác định một địa chỉ offset tương ứng với một thanh ghi đoạn khác một cách

rõ ràng, tường minh. Khuôn mẫu của một toán hạng để thực hiện điều này như sau:

```
segment_register:[pointer_register]
```

Ví dụ:

```
MOV AX, ES:[SI]
```

Nếu SI chứa 0100h, địa chỉ của toán hạng nguồn trong trường hợp này sẽ là ES : 0100h. Có thể bạn muốn thực hiện điều này trong một chương trình có hai đoạn dữ liệu và ES chứa số hiệu đoạn của đoạn dữ liệu thứ hai.

Phương pháp ghi rõ đoạn cũng có thể dùng được với chế độ địa chỉ chỉ số và địa chỉ cơ sở.

10.2.5. Truy nhập ngăn xếp.

Như chúng tôi đã lưu ý trước đây, khi BP xác định một địa chỉ offset trong chế độ địa chỉ gián tiếp thanh ghi thì SS sẽ chứa số hiệu đoạn. Điều này có nghĩa là BP có thể được dùng để truy nhập các phần tử của ngăn xếp.

Ví dụ 10.10. Chuyển 3 từ ở đỉnh ngăn xếp vào AX, BX và CX mà không làm thay đổi ngăn xếp.

Lời giải:

```
MOV BP, SP      ; SP trở đến đỉnh ngăn xếp  
MOV AX, [BP]    ; chuyển đỉnh ngăn xếp vào AX  
MOV BX, [BP+2]  ; chuyển từ thứ hai vào BX  
MOV CX, [BP+4]  ; chuyển từ thứ ba vào BX
```

Một ví dụ tương tự đó là dùng BP để truyền các giá trị cho thủ tục (xem chương 14).

10.3. Sắp xếp một mảng.

Khi một mảng đã được sắp xếp, chúng ta sẽ dễ dàng xác định vị trí các phần tử của nó. Có đến hàng tá các phương pháp sắp xếp. Phương pháp mà chúng tôi sẽ trình bày ở đây gọi là phương pháp chọn lựa. Nó là một trong những phương pháp sắp xếp đơn giản nhất.

Để sắp xếp một mảng N phần tử chúng ta thực hiện các bước sau đây:

Bước 1: Tìm phần tử lớn nhất của A[1] ... A[N]. Đổi chỗ nó và A[N]. Do phần tử lớn nhất đã được đặt ở vị trí N, chúng ta chỉ còn phải sắp xếp A[1] ... A[N-1].

Bước 2: Tìm phần tử lớn nhất của A[1] ... A[N-1]. Đổi chỗ nó và A[N-1]. Động tác này đặt phần tử “lớn thứ hai” vào đúng vị trí của nó.

Bước N-1: Tìm phần tử lớn nhất của A[1], A[2]. Đổi chỗ nó với A[2]. Lúc này A[2] ... A[N] đã được đặt ở vị trí thích hợp của chúng và A[1] cũng vậy. Như thế, mảng đã sắp xếp xong.

Ví dụ, mảng A chứa các số nguyên 21, 5, 16, 40, 7:

| Vị trí | 1 | 2 | 3 | 4 | 5 |
|-----------------|----|---|----|----|----|
| Giá trị ban đầu | 21 | 5 | 16 | 40 | 7 |
| Bước1 | 21 | 5 | 16 | 7 | 40 |
| Bước2 | 7 | 5 | 16 | 21 | 40 |
| Bước3 | 7 | 5 | 16 | 21 | 40 |
| Bước4 | 5 | 7 | 16 | 21 | 40 |

Thuật toán sắp xếp chọn lựa

```
i = N  
FOR N-1 DO  
    tìm vị trí k của phần tử lớn nhất của A[1]...A[i]  
    (*) đổi chỗ A[k] và A[i]  
    i = i - 1  
END_FOR
```

Bước (*) có thể được thực hiện nhờ thủ tục SWAP. Sau đây là chương trình (ta giả thiết mảng cần sắp xếp là mảng các byte):

Chương trình PGM10_2.ASM

```
1: SELECT PROC  
2: ;sắp xếp một mảng byte bằng phương pháp chọn lựa.  
3: ;vào:SI=địa chỉ offset của mảng  
4: ;BX=số phần tử của mảng ra  
5: ;ra:SI=offset của mảng đã sắp xếp
```

```

6: ;sử dụng:SWAP
7:         PUSH BX,
8:         PUSH CX,
9:         PUSH DX,
10:        PUSH SI,
11:        DEC BX;           ;N=N-1
12:        JE END_SORT;   ;thoát nếu còn một phần tử
13:        MOV DX,SI;      ;chép offset của mảng
14: ;for N-1 times do
15: SORT_LOOP:
16:         MOV SI,DX;      ;SI trỏ đến mảng
17:         MOV CX,BX;      ;số phép so sánh
18:         MOV DI,SI;      ;DI trỏ đến p.tử lớn nhất
19:         MOV AL,[DI];    ;AL chứa phần tử lớn nhất
20: ;xác định phần tử lớn nhất còn lại
21: FIND_BIG:
22:         INC SI;         ;SI trỏ đến p.tử tiếp theo
23:         CMP [SI],AL;    ;p.tử mới > p.tử lớn nhất?
24:         JNG NEXT;       ;không, tiếp tục
25:         MOV DI,SI;      ;đúng, trỏ đến nó
26:         MOV AL,[DI];    ;AL chứa p.tử lớn nhất
27: NEXT:
28:         LOOP FIND_BIG; ;lặp đến khi hoàn thành
29: ;đổi chỗ phần tử lớn nhất với phần tử cuối cùng
30:         CALL SWAP;      ;đổi chỗ với p.tử cuối cùng
31:         DEC BX;         ;N=N-1
32:         JNE SORT_LOOP; ;lặp lại nếu N>>0
33: END_SORT:
34:         POP SI;
35:         POP DX;
36:         POP CX;
37:         POP BX;
38:         RET;
39: SELECT ENDP;
40: SWAP PROC;
41: ;đổi chỗ hai phần tử của mảng
42: ;vào:SI=một phần tử
43: ;DI=phần tử kia
44: ;ra:đổi chỗ hai phần tử
45:         PUSH AX;        ;cất AX
46:         MOV AL,[SI];    ;lấy A[i]
47:         XCHG AL,[DI];  ;đưa vào A[k]
48:         MOV [SI],AL;    ;đưa A[k] vào A[i]
49:         POP AX;        ;khôi phục AX

```

```
50:          RET
51: SWAP      ENDP.
```

Trước khi vào thủ tục SELECT, SI chưa địa chỉ offset của mảng còn BX chưa số phần tử N. Thuật toán sắp xếp mảng có N-1 bước tương ứng với BX giảm dần. Nếu BX bằng 0, chúng ta chỉ còn mảng một phần tử. Đến đây chẳng còn gì để làm nữa và ta thoát khỏi thủ tục.

Trong trường hợp tổng quát, thủ tục tiến vào vòng lặp xử lý chính (dòng 15-32). Mỗi lần duyệt qua vòng lặp này, phần tử lớn nhất trong số các phần tử chưa sắp xếp còn lại được đưa vào vị trí thích hợp của nó.

Các dòng 21-28 là vòng lặp để tìm phần tử lớn nhất trong số các phần tử chưa được sắp xếp còn lại. Khi ra khỏi vòng lặp này, DI trỏ đến phần tử lớn nhất và SI trỏ đến phần tử cuối cùng của mảng. Dòng 30 gọi thủ tục SWAP để hoán chuyển các phần tử được SI và DI trỏ đến.

Ta có thể kiểm tra thủ tục bằng cách chèn nó vào một chương trình thử nghiệm.

Chương trình PGM10_3.ASM

```
TITLE      PGM10_3: Kiểm tra SELECT
.MODEL     SMALL
.STACK    100H
.DATA
ALA       DB      5, 2, 1, 3, 4
.CODE
MAIN      PROC
          MOV     Ax, @DATA
          MOV     DS, AX
          LEA     SI, ALA
          MOV     BX, 5
          CALL   SELECT
          MOV     AH, 4Ch
          INT    21h
MAIN      ENDP
;SELECT ở đây
END      MAIN
```

Sau khi biên dịch và liên kết, ta vào DEBUG và thi hành đến lời gọi thủ tục (địa chỉ chỉ ra sau đây đã được xác định trong DEBUG)

-GF

```
AX=100D BX=0005 CX=0049 DX=0000 SP=0100 BP=0000 SI=0004 DI=0000  
DS=100D ES=0FF9 SS=100E CS=1009 IP=000C NV UP EI PL NZ NA PO NC  
1009:000C E80400 CALL 00013
```

Trước khi gọi thủ tục ta hãy xem lại mảng chưa sắp xếp:

```
-D4 8  
100D:0000 05 02 01 03-04
```

Dữ liệu xuất hiện theo thứ tự 5, 2, 1, 3, 4. Bây giờ chúng ta hãy thực hiện SELECT:

```
-GF  
AX=1002 BX=0005 CX=0049 DX=0000 SP=0100 BP=0000 SI=0004 DI=0005  
DS=100D ES=0FF9 SS=100E CS=1009 IP=000F NV UP EI PL ZR NA PE NC  
1009:000F B44C MOV AH,4C
```

Và ta hãy xem lại mảng:

```
-D4 8  
100D:0000 01 02 03 04 05
```

Mảng của chúng ta bây giờ đã được sắp xếp theo thứ tự tăng dần.

10.4. Mảng hai chiều.

Mảng hai chiều (two-dimensional array) là mảng của các mảng có nghĩa là một mảng một chiều mà các phần tử của nó lại là các mảng một chiều. Ta có thể tưởng tượng các phần tử được xếp vào các hàng và các cột. Hình 10.2 chỉ ra một mảng hai chiều B với ba hàng, bốn cột (mảng 3x4). B[i,j] là phần tử thuộc hàng i và cột j.

Mảng hai chiều được lưu trữ trong bộ nhớ như thế nào?

Bởi vì bộ nhớ là mảng một chiều cho nên các phần tử của mảng hai chiều phải được lưu trữ liên tiếp nhau. Có hai phương pháp thường được sử dụng để lưu trữ mảng. Phương pháp thứ nhất là lưu trữ theo thứ tự hàng (row-major order), trong đó các phần tử hàng 1 được lưu trữ, tiếp theo là các phần tử hàng 2, rồi đến các phần tử hàng 3 và cứ như thế... Phương pháp thứ hai là lưu trữ theo thứ tự cột. Phương pháp này lưu trữ dữ liệu cột 1, tiếp theo là cột 2 rồi đến cột 3 v.v... Ví dụ, mảng B chứa 10, 20, 30, 40 trong hàng đầu tiên; 50, 60, 70, 80 trong hàng thứ hai và 90, 100, 110, 120 trong hàng thứ ba. Nó có thể được lưu trữ theo thứ tự hàng như sau:

B DB 10, 20, 30, 40
DB 50, 60, 70, 80
DB 90, 100, 110, 120

hay theo thứ tự cột:

B DB 10, 50, 90
DB 20, 60, 100
DB 30, 70, 110
DB 40, 80, 120

Hình 10.2A. Mảng hai chiều B

| | | Cột | | | |
|------|---|--------|--------|--------|--------|
| | | 1 | 2 | 3 | 4 |
| Hàng | 1 | B[1,1] | B[1,2] | B[1,3] | B[1,4] |
| | 2 | B[2,1] | B[2,2] | B[2,3] | B[2,4] |
| | 3 | B[3,1] | B[3,2] | B[3,3] | B[3,4] |
| | 4 | | | | |

Hầu hết các chương trình biên dịch ngôn ngữ bậc cao lưu trữ mảng hai chiều theo thứ tự hàng. Trong Hợp ngữ chúng ta có thể lưu trữ theo cả hai cách. Nếu như các phần tử của một hàng được xử lý liên tiếp nhau thì việc lưu trữ theo thứ tự hàng chiếm ưu thế hơn bởi vì các phần tử kề nhau trong một hàng

chiếm các ô nhớ kế tiếp nhau. Ngược lại, lưu trữ theo thứ tự cột sẽ tốt hơn khi xử lý các phần tử trong cùng một cột.

Xác định vị trí một phần tử trong mảng hai chiều.

Giả thiết ta có một mảng hai chiều lưu trữ trong bộ nhớ theo thứ tự hàng, trong đó kích thước mỗi phần tử là S (mảng byte S = 1, mảng word S = 2). Để tìm vị trí của A[i,j] ta xác định:

1. Vị trí bắt đầu cột i.
2. Vị trí của phần tử thứ j trong hàng.

Sau đây là bước thứ nhất. Hàng 1 bắt đầu tại vị trí A. Do có N phần tử trong mỗi hàng và mỗi phần tử có kích thước S byte, hàng 2 sẽ bắt đầu tại vị trí $A + N * S$. Tương tự, hàng 3 bắt đầu tại vị trí $A + 2 * N * S$. Một cách tổng quát ta có hàng i sẽ bắt đầu tại vị trí $A + (i - 1) * N * S$.

Và bây giờ đến bước thứ hai. Như chúng tôi đã trình bày về các mảng một chiều, phần tử thứ j của một hàng được lưu trữ ở vị trí $(j - 1) * S$ byte kể từ đầu dòng.

Cộng kết quả của bước 1 và bước 2 ta được kết quả cuối cùng:

Nếu một mảng A kích thước $M \times N$, mỗi phần tử chiếm S byte được lưu trữ theo theo thứ tự hàng thì:

$$A[i,j] \text{ có địa chỉ } A + ((i - 1) * N + (j - 1)) * S \quad (1)$$

Tương tự với các mảng lưu trữ theo theo thứ tự cột:

Nếu một mảng A kích thước $M \times N$, mỗi phần tử chiếm S byte được lưu trữ theo theo thứ tự cột thì:

$$A[i,j] \text{ có địa chỉ } A + ((i - 1) + (j - 1) * M) * S \quad (2)$$

Ví dụ 10.12. Giả sử A là một mảng word $M \times N$ được lưu trữ trong bộ nhớ theo theo thứ tự hàng. Hãy xác định:

1. Vị trí bắt đầu hàng i.
2. Vị trí bắt đầu cột j.
3. Số byte giữa các phần tử trong một cột.

Lời giải:

1. Hàng i bắt đầu tại $A[i,1]$, theo công thức (1) nó có địa chỉ:

$$A + (i - 1) * N * 2.$$

2. Cột j bắt đầu tại $A[1,j]$, theo công thức (1) nó có địa chỉ: $A + (j - 1) * 2$.

3. Do có N cột nên sẽ có $2 * N$ byte giữa hai phần tử kề nhau trong một cột bất kỳ.

10.5. Chế độ địa chỉ chỉ số cơ sở.

Trong chế độ này, địa chỉ offset của một toán hạng là tổng của:

1. Nội dung của một thanh ghi cơ sở (BX hay BP).
2. Nội dung của một thanh ghi chỉ số (SI hay DI).
3. (có thể) Địa chỉ offset của một biến byte.
4. (có thể) Một hằng số (âm hay dương).

Nếu chúng ta sử dụng BX, DS sẽ chứa số hiệu đoạn của địa chỉ toán hạng, còn nếu BP được sử dụng, SS sẽ chứa số hiệu địa chỉ đoạn. Toán hạng có thể được viết ở nhiều dạng khác nhau, sau đây là một số dạng thường dùng:

1. biến nhớ [thanh ghi cơ sở][thanh ghi chỉ số]
2. [thanh ghi chỉ số + thanh ghi cơ sở + biến nhớ + hằng số]
3. biến nhớ [thanh ghi chỉ số + thanh ghi cơ sở + hằng số]
4. hằng số [thanh ghi chỉ số + thanh ghi cơ sở + biến nhớ]

* Thứ tự các phần tử trong dấu ngoặc là tuỳ ý.

Ví dụ W là một biến byte, BX chứa 2 và SI chứa 4. Khi đó lệnh:

MOV AX, W[BX][SI]

sẽ chuyển nội dung của $W + 2 + 4 = W + 6$ vào AX. Lệnh trên cũng có thể được viết dưới các dạng sau:

MOV AX, [W + BX + SI]

hay

MOV AX, W[BX + SI]

Chế độ địa chỉ chỉ số cơ sở đặc biệt hữu hiệu khi xử lý mảng hai chiều. Bạn sẽ thấy rõ điều này trong ví dụ sau đây.

Ví dụ 10.13. Giả sử A là một mảng word kích thước 5×7 , lưu trữ số liệu theo thứ tự hàng. Bạn hãy viết các lệnh sử dụng chế độ địa chỉ chỉ số cơ sở để:

1. Xoá hàng 3.

2. Xoá cột 4.

Lời giải:

1. Trong ví dụ 10.2 chúng ta đã biết rằng với A là một mảng word M x N phần tử thì hàng i sẽ bắt đầu tại địa chỉ $A + (i - 1) * N * 2$. Như vậy trong mảng 5 x 7, hàng 3 bắt đầu tại địa chỉ $A + (3 - 1) * 7 * 2 = A + 28$. Chúng ta có thể xoá hàng 3 như sau:

```
MOV BX, 28          ;BX chỉ đến hàng 3
XOR SI, SI          ;SI sẽ chỉ đến các cột
MOV CX, 7           ;số p.tử trong một hàng
CLEAR:
    MOV A[BX][SI], 0   ;xoá A[3,1]
    ADD SI, 2           ;trở đến cột kế tiếp
    LOOP CLEAR          ;lặp đến khi xong
```

2. Cũng trong ví dụ 10.2 chúng biết được cột j bắt đầu tại địa chỉ $A + (j - 1) * 2$ với A là mảng word M x N phần tử. Như vậy cột 4 sẽ bắt đầu tại địa chỉ $A + (4 - 1) * 2 = A + 6$. Do A có 7 cột và được lưu trữ theo thứ tự hàng nên để trả đến phần tử tiếp theo trong cột 4 ta phải cộng thêm địa chỉ offset với $7 * 2 = 14$. Chúng ta có thể xoá cột 4 như sau:

```
MOV SI, 6           ;SI chỉ đến cột 4
XOR BX, BX          ;BX sẽ chỉ đến các hàng
MOV CX, 5           ;số p.tử trong một cột
CLEAR:
    MOV A[BX][SI], 0   ;xoá A[1,4]
    ADD BX, 14          ;trở đến hàng kế tiếp
    LOOP CLEAR          ;lặp đến khi xong
```

10.6. Một ứng dụng: tính điểm trung bình.

Giả sử một trong lớp có 5 học sinh vừa thi hết học kỳ 1. Kết quả 4 môn thi của họ như sau:

| | Bài thi 1 | Bài thi 2 | Bài thi 3 | Bài thi 4 |
|------------|-----------|-----------|-----------|-----------|
| Ngọc Lan | 10 | 8 | 8 | 10 |
| Kiều Nga | 8 | 8 | 9 | 10 |
| Thu Hiền | 9 | 8 | 10 | 10 |
| Ngọc Thuý | 8 | 7 | 8 | 10 |
| Thanh Hằng | 10 | 8 | 6 | 10 |

Chúng ta sẽ viết một chương trình để tính điểm trung bình của mỗi môn thi. Để làm điều này, ta chỉ việc cộng các điểm trong mỗi cột rồi chia cho 5.

Thuật toán:

1. $j=4$
2. REPEAT
3. tính tổng các điểm trong cột j
4. $j=j-1$
5. UNTIL $j=0$

Bắt đầu tính tổng từ cột 4 làm cho chương trình của chúng ta ngắn hơn một chút. Bước 3 có thể được viết chi tiết hơn như sau:

```
sum[j]=0  
i=1  
FOR 5 DO  
    sum[j]=sum[j]+score[i,j]  
    i=i+1  
END_FOR
```

Chương trình PGM10_4.ASM

```
0: TITLE      PGM10_4: Tính điểm trung bình.  
1: .MODEL     SMALL  
2: .STACK     100H  
3: .DATA  
4: FIVE       DW      5  
5: SCORE      DW      10,8,8,10      ;ngọc lan  
6:           DW      8,8,9,10      ;kiều nga  
7:           DW      9,8,10,10     ;thu hiền  
8:           DW      8,7,8,10      ;ngọc thuý  
9:           DW      10,8,6,10     ;thanh hằng  
10:AVG        DW      4 DUP (0)  
11:.CODE  
12:MAIN       PROC  
13:           MOV     AX,@DATA  
14:           MOV     DS,AX      ;khởi tạo DS  
15:;J=4  
16:           MOV     SI,6.      ;khởi tạo chỉ đến cột 4  
17:REPEAT:  
18:           MOV     CX,5      ;số p.tử trong 1 cột  
19:           XOR     BX,BX     ;khởi tạo chỉ hàng 1
```

```

20:          XOR AX,AX      ;khởi tạo tổng =0
21: ;tính tổng các điểm trong cột j
22: FOR:
23:          ADD AX,SCORES[BX+SI];tổng=tổng+điểm
24:          ADD BX,8       ;chỉ đến hàng tiếp theo
25:          LOOP FOR      ;cộng hết một cột
26: ;end_for
27: ;tính trị trung bình cột j
28:          XOR DX,DX      ;xoá phần cao số bị chia
29:          DIV FIVE        ;AX=trị trung bình
30:          MOV AVG[SI],AX ;chứa vào mảng
31:          SUB SI,2       ;đến cột tiếp theo
32: ;intil j=0
33:          JNL REPEAT     ;lặp khi SI>=0
34: ;trở về DOS
35:          MOV AH,4Ch
36:          INT 21h
37: MAIN      ENDP
38: END MAIN

```

Các điểm thi được lưu trữ trong một mảng hai chiều (dòng 5-9).

Các dòng 22-25 tính tổng một cột và chứa kết quả vào trong mảng AVG. Chương trình lấy kết quả này chia cho 5 để tính điểm trung bình ở các dòng 28-30.

Các hàng và các cột của mảng SCORE được tham trỏ tương ứng bởi BX và SI. Chúng ta bắt đầu tính tổng từ cột 4 và cột này có địa chỉ bắt đầu là SCORE + 6 do đó SI được khởi tạo bằng 6. Sau khi tính tổng một cột, SI được giảm đi 2 (để trỏ đến cột tiếp theo) đến khi nó bằng 0 thì kết thúc.

Việc thi hành chương trình có thể được xem nhờ DEBUG. Chúng ta thi hành đến trước lệnh trở về DOS và liệt kê mảng AVG (các địa chỉ sử dụng ở đây đã được xác định trong lần làm việc trước với DEBUG).

-G29

```

AX=4C4B BX=0028 CX=0000 DX=0002 SP=0100 BP=0000 SI=FFFE DI=0000
DS=100B ES=0FF9 SS=100F CS=1009 IP=0029 NV UP EI NG ZR AC PO CY
100B:0029 CD21  INT 21

```

-D30 3D

```

100B:0030 09 00-07 00 08 00 08 00

```

Các điểm trung bình được lưu trữ: 0009h, 0007h, 0008h và 000Ah, hay viết dưới dạng nhị phân là 9, 7, 8 và 10.

10.7. Lệnh XLAT, MOV AL, [AL] [BX]

Đôi khi trong các ứng dụng chung ta phải chuyển dữ liệu từ một dạng này sang dạng khác. Ví dụ IBM PC dùng mã ASCII cho các ký tự nhưng các máy chủ IBM lại sử dụng EBCDIC (Extended Binary Coded Decimal Interchange Code). Để chuyển đổi một chuỗi ký tự từ mã ASCII sang mã EBCDIC chương trình của chúng ta phải thay mã ASCII của mỗi ký tự trong chuỗi bằng mã EBCDIC tương ứng.

XLAT (translate) là một lệnh không toán hạng, nó có thể được sử dụng để đổi giá trị của một byte sang một giá trị khác được lấy từ một bảng. Byte được đổi phải chứa trong AL và BX chứa địa chỉ offset của bảng chuyển đổi. Lệnh XLAT thực hiện các công việc sau: (1) Cộng nội dung AL với địa chỉ trong BX để nhận được một địa chỉ trong bảng và (2) thay thế nội dung của AL bằng giá trị tìm được ở địa chỉ trên.

Ví dụ: giả sử nội dung của AL trong khoảng từ 0 đến Fh và ta muốn thay nó bằng mã ASCII của chữ số hex tương ứng. Chẳng hạn 6h thay bởi 036h = '6' hay Bh thay bởi 042h = 'B'. Bảng chuyển đổi lúc này sẽ là

| | |
|-------|---|
| TABLE | DB 030h, 031h, 032h, 033h, 034h, 035h, 036h, 037h |
| | DB 038h, 039h, 041h, 042h, 043h, 044h, 045h, 046h |

Ví dụ, để đổi 0Ch thành 'C' ta có thể viết:

| | | |
|------|-----------|---------------------------|
| MOV | AL, 0Ch | ; số cần đổi |
| LEA | BX, TABLE | ; BX chứa offset của bảng |
| XLAT | | ; AL chứa 'C' |

Lệnh XLAT ở đây tính địa chỉ TABLE + Ch = TABLE + 12, sau đó nó thay nội dung của AL bởi giá trị tại địa chỉ vừa tìm được, tức là 043h = 'C'.

Trong ví dụ này nếu AL không chứa giá trị trong khoảng (0 - 15), XLAT sẽ thay thế nó bằng một giá trị vô nghĩa.

Ví dụ: mã hoá và giải mã một mật khẩu.

Trong ví dụ sau đây, chương trình sẽ nhắc người sử dụng đánh vào một dòng chữ, mã hoá nó và in ra dòng chữ đã mã hoá. Cuối cùng dịch ngược trở lại rồi in ra dòng giải mã.

Một ví dụ khi thực hiện chương trình:

ENTER A MESSAGE:

GATHER YOUR FORCES AND ATTACK AT DAWT (dòng chữ vào)
ZXKBGM WULM HUMPGN XJO XKKXPDXK OXSJ..(đã mã hoá)
GATHER YOUR FORCES AND ATTACK AT DAWT(đã giải mã)

Thuật toán:

in ra lời nhắc
đọc và mã hoá dòng chữ
xuống dòng mới
in ra dòng đã mã hoá
xuống dòng mới
giải mã và in ra kết quả

Chương trình PGM10_5.ASM

```
0: TITLE      PGM10_5: Mật khẩu
1: .MODEL     SMALL
2: .STACK     100H
3: .DATA
4: ;alphabet           ABCDEFGHIJKLMNOPQRSTUVWXYZ
5: CODE_KEY DB 65 DUP (' ')
               DB           'XQPOGHZBCADEIJUVFMNKLRSWY'
6:          DB 37 DUP (' ')
7: DECODE_KEY DB 65 DUP (' ')
               DB           'JHIKLQEFMNUTRSDCBVWXOOPYAZG'
8:          DB 37 DUP (' ')
9: CODED      DB 80 DUP ('$')
10: PROMPT    DB      'ENTER A MESSAGE', 0Dh, 0Ah, '$'
11: CRLF      DB      0Dh, 0Ah, '$'
12: .CODE
13: MAIN      PROC
14:           MOV AX, @DATA           ;khởi tạo DX
15:           MOV DS, AX
16: ;in ra lời nhắc
17:           MOV AH, 9             ;hàm hiển thị chuỗi
18:           LEA DX, PROMPT        ;DX trả đến lời nhắc
```

```

19:      INT 21h           ;in ra lời nhắc
20: ;đọc và mã hoá một dòng chữ
21:      MOV AH,1           ;hàm đọc ký tự
22:      LEA BX,CODE_KEY ;BX trả đến bảng mã hoá
23:      LEA DI,CODED     ;DI trả đến dòng mã hoá
24: WHILE:
25:      INT 21h           ;đọc một ký tự
26:      CMP AL,0Dh        ;CR?
27:      JE ENDWHILE      ;đúng,in ra dòng mã hoá
28:      XLAT              ;không,mã hoá ký tự
29:      MOV [DI],AL        ;cất vào dòng mã hoá
30:      INC DI            ;chuyển con trỏ
31:      JMP WHILE_        ;xử lý ký tự tiếp theo
32: END_WHILE:
33: ;xuống dòng mới
34:      MOV AH,9
35:      LEA DX,CRLF
36:      INT 21h           ;dòng mới
37: ;in ra dòng chữ đã mã hoá
38:      LEA DX,CODED     ;DX trả đến dòng mã hoá
39:      INT 21h           ;in ra dòng mã hoá
40: ;xuống dòng mới
41:      LEA DX,CRLF
42:      INT 21h           ;dòng mới
43: ;mã hoá dòng chữ và in ra kết quả
44:      MOV AH,2           ;hàm in ký tự
45:      LEA BX,DECODE_KEY;BX trả đến bảng giải mã
46:      LEA SI,CODED     ;SI trả đến dòng mã hoá
47: WHILE1:
48:      MOV AL,[SI]         ;lấy một ký tự
49:      CMP AL,'$'         ;ký tự kết thúc chuỗi
50:      JE ENDWHILE1      ;đúng,kết thúc
51:      XLAT              ;không,giải mã ký tự
52:      MOV DL,AL          ;đưa vào DL
53:      INT 21h           ;in ra
54:      INC SI            ;chuyển con trỏ
55:      JMP WHILE1        ;xử lý ký tự tiếp theo
56: ENDWHILE1
57:      MOV AH,4Ch
58:      INT 21h           ;trở về DOS
59: MAIN    ENDP
60:      END   MAIN

```

Có ba mảng được khai báo trong đoạn dữ liệu:

- CODE_KEY dùng để mã hoá.
- CODED chứa dòng chữ đã được mã hoá, được khởi tạo bằng một chuỗi các ký tự '\$'. Như vậy khi ta đưa chuỗi (chiều dài nhỏ hơn chuỗi khởi tạo) vào vị trí này, nó sẽ được kết thúc bằng ký tự '\$' và ta có thể in được bằng hàm 9 của ngắt 21h.
- DECODE_KEY dùng để chuyển đổi dòng chữ đã được mã hoá trở về dạng nguyên thuỷ của nó.

Các bạn sẽ dễ dàng nhận ra cách thức mã hoá và giải mã các ký tự nhờ bảng chữ cái trong dòng lời bình 4 được viết tương ứng với CODE_KEY và DECODE_KEY. Các dòng 24-32 đọc các ký tự rồi mã hoá chúng và kết thúc khi gặp ký tự về đầu dòng. AL nhận mã ASCII của ký tự được đánh vào, XLAT cộng giá trị này với địa chỉ của CODE_KEY trong BX để tìm ra địa chỉ của kết quả chuyển đổi trong bảng CODE_KEY.

Mảng CODE_KEY tạo thành từ 65 ký tự trắng, tiếp theo là các chữ cái, là các giá trị mã hoá của các ký tự từ 'A' đến 'Z' và kết thúc với 37 ký tự trắng nữa để được tổng cộng 128 byte (128 byte là cần thiết bởi vì giới hạn các ký tự ASCII chuẩn từ 0 đến 127). Ví dụ ta đánh vào chữ cái 'A', mã ASCII của nó là 65. XLAT sẽ đưa giá trị của byte có địa chỉ CODE_KEY mà ở đây này là 'X', vào AL. Đến dòng 29, giá trị này được chuyển vào mảng CODED. Một cách tương tự, 'B' đổi thành 'Q', 'C' thành 'P' ... 'Z' thành 'Y' (bảng mã hoá là tùy ý). Các ký tự không phải chữ hoa (bao gồm cả ký tự trắng) có mã ASCII trong khoảng từ 0 đến 64 hoặc từ 92 đến 127, được đổi thành ký tự trắng. Các dòng 38-39 in ra dòng chữ đã được mã hoá.

DECODE_KEY cũng bắt đầu và kết thúc tương ứng với 65 và 37 ký tự trắng. Vị trí các chữ cái trong mảng được xác định như sau (nằm dưới bảng chữ cái dòng 4): Do 'A' được mã hoá thành 'X' nên ký tự ở vị trí 'X' trong bảng giải mã phải là 'A'. Tương tự, bởi vì 'B' được mã hoá thành 'Q', chữ cái 'B' phải ở vị trí 'Q' và cứ như vậy...

Các dòng 47-56 dịch dòng chữ đã mã hoá. Sau khi đưa địa chỉ của DECODE_KEY và CODED một cách tương ứng vào BX và SI, chương trình chuyển một byte của dòng mã hoá vào AL. Nếu là ký tự '\$', có nghĩa dòng chữ đã được dịch xong và chương trình được thoát ra. Ngược lại, XLAT cộng AL với địa chỉ DECODE_KEY để nhận được một địa chỉ trong bảng giải mã, sau đó đưa ký tự tìm được vào AL. Dòng 52 chuyển ký tự này vào DL, như vậy nó có thể được in ra bằng hàm 2 của ngắt 21h.

TỔNG KẾT.

- ◆ Mảng một chiều là một chuỗi các phần tử có cùng kiểu. Các toán tử giả DB và DW được dùng để khai báo các mảng byte và word.
- ◆ Ta có thể xác định địa chỉ một phần tử của mảng bằng cách cộng một hằng số với địa chỉ cơ sở.
- ◆ Chế độ địa chỉ của một toán hạng là phương thức xác định địa chỉ của nó. Các chế độ địa chỉ gồm có chế độ địa chỉ thanh ghi, tức thời, trực tiếp, gián tiếp thanh ghi, cơ sở, chỉ số và chế độ địa chỉ chỉ số cơ sở.
- ◆ Trong chế độ địa chỉ gián tiếp thanh ghi, toán hạng có dạng [thanh_ghi], ở đây thanh_ghi có thể là BX, SI, DI hay BP. Offset của toán hạng được xác định bằng cách cộng độ dịch với nội dung của thanh_ghi. Với BX, SI và DI, DS chứa số hiệu đoạn; còn với BP, số hiệu đoạn trong SS.
- ◆ Các toán tử BYTE PTR và WORD PTR đứng trước một toán hạng dùng để định lại kiểu đã khai báo của toán hạng.
- ◆ Toán tử giả LABEL có thể dùng để xác định kiểu dữ liệu cho một biến.
- ◆ Mảng hai chiều là mảng một chiều mà các phần tử là các mảng một chiều. Mảng hai chiều có thể được lưu trữ “hàng nối tiếp hàng” (theo theo thứ tự hàng) hay “cột nối tiếp cột” (theo thứ tự cột).
- ◆ Trong chế độ địa chỉ chỉ số cơ sở, địa chỉ offset của toán hạng là tổng của: (1)BX hay BP; (2) SI hay DI; (3) Địa chỉ offset của một ô nhớ (...);(4) Một hằng số(...). Ví dụ một dạng hợp lệ: [thanh ghi cơ sở + thanh ghi chỉ số + ô nhớ +hằng số]. DS chứa số hiệu đoạn nếu ta sử dụng BX, còn nếu dùng BP, SS sẽ chứa số hiệu đoạn.
- ◆ Chế độ địa chỉ chỉ số cơ sở thường dùng để xử lý mảng hai chiều.
- ◆ Lệnh XLAT dùng để đổi giá trị của một byte sang một giá trị khác lấy từ một bảng. AL chứa giá trị được đổi và BX chứa địa chỉ của bảng. Lệnh này cộng AL với địa chỉ offset chứa trong BX để tìm ra được một địa chỉ trong bảng. Nội dung của AL sẽ được thay bằng giá trị tại địa chỉ này.

Các thuật ngữ tin học.

| | |
|--------------------------------|--|
| addressing mode | Chế độ địa chỉ: phương thức xác định địa chỉ toán hạng. |
| base address of array | Địa chỉ cơ sở của mảng; địa chỉ của biến mảng |
| based addressing mode | Chế độ địa chỉ cơ sở: là chế độ địa chỉ gián tiếp trong đó nội dung của BX hay BP được cộng với độ dịch tạo thành địa chỉ của toán hạng. |
| column-major order | Thứ tự cột: cột tiếp theo cột |
| direct mode | Chế độ địa chỉ trực tiếp: toán hạng là biến nhỏ. |
| displacement | Độ dịch: số được cộng vào nội dung của thanh ghi để xác định địa chỉ offset của toán hạng trong chế độ địa chỉ chỉ số hay địa chỉ cơ sở. |
| immediate mode | Chế độ địa chỉ tức thời: toán hạng là hằng số. |
| indexed addressing mode | Chế độ địa chỉ chỉ số: là chế độ địa chỉ gián tiếp trong đó nội dung của SI hay DI được cộng với độ dịch tạo thành địa chỉ offset của toán hạng. |
| one-dimensional array | Mảng một chiều: chuỗi các phần tử có cùng kiểu. |
| pointer | Con trỏ: thanh ghi chứa địa chỉ offset của một toán hạng. |
| register mode | Chế độ địa chỉ thanh ghi: toán hàng là một thanh ghi. |
| row-major order | Thứ tự hàng: hàng nối tiếp hàng. |
| two-dimensional array | Mảng hai chiều: Mảng một chiều mà các phần tử là các mảng một chiều. |

Các lệnh mới:

XLAT

Các toán tử giả mới:

DUP

LABEL

PTR

Bài tập:

1. Giả thiết:

| | |
|---------------|------------------------|
| AX chứa 0500h | offset 1000 chứa 0100h |
| BX chứa 1000h | offset 1500 chứa 0150h |
| SI chứa 1500h | offset 2000 chứa 0200h |
| DI chứa 2000h | offset 2500 chứa 0250h |
| | offset 3000 chứa 0300h |

và BETA là một biến word nằm ở địa chỉ offset 1000h.

Với mỗi trong các lệnh sau đây, nếu hợp lệ, hãy cho biết địa chỉ offset của toán hạng nguồn hay thanh ghi và kết quả được lưu trữ trong toán hạng đích.

- a. MOV DI, SI
- b. MOV DI, [DI]
- c. ADD AX, [SI]
- d. SUB BX, [DI]
- e. LEA BX, BETA[BX]
- f. ADD [SI], [DI]
- g. ADD BH, [BL]
- h. ADD AH, [SI]
- i. MOV AX, [BX+DI+BETA]

2. Cho các khai báo sau đây:

| | | |
|-----|------------|---------|
| A | DW | 1, 2, 3 |
| B | DB | 4, 5, 6 |
| C | LABEL WORD | |
| MSG | DB | 'ABC' |

và giả thiết BX chứa địa chỉ offset của C. Hãy cho biết lệnh nào trong các lệnh sau đây là hợp lệ. Với lệnh hợp lệ hãy cho biết con số được chuyển dịch.

- a. MOV AH, BYTE PTR A
- b. MOV AX, WORD PTR B
- c. MOV AX, C
- d. MOV AX, MSG
- e. MOV AH, BYTE PTR C

3. Sử dụng BP và chế độ địa chỉ cơ sở thực hiện các thao tác với ngăn xếp sau đây (Bạn có thể dùng một thanh ghi khác nhưng không được dùng các lệnh PUSH và POP):

- a. Thay nội dung của hai từ đỉnh ngăn xếp bằng 0.
- b. Sao chép một ngăn xếp gồm 5 từ sang một mảng word ST_A RR sao cho ST_ARR chứa đỉnh ngăn xếp, ST_ARR+2 chứa từ tiếp theo của đỉnh ngăn xếp, và cứ như vậy.

4. Viết các lệnh thực hiện các thao tác sau đây trên mảng word A 10 phần tử hay mảng byte B 15 phần tử:

- a. Chuyển A[i+1] vào vị trí i với $i = 1 \dots 9$ và A[1] đến vị trí 10.
- b. Dùng DX đếm số số 0 của mảng A.
- c. Giả sử mảng byte B chứa một chuỗi ký tự. Hãy kiểm tra lần xuất hiện đầu tiên của chữ cái 'E' trong B. Nếu tìm thấy, hãy đặt SI trả đến ô nhớ đó. Ngược lại, hãy lập cờ CF.

5. Viết thủ tục FIND_IJ trả về địa chỉ offset của phần tử trong hàng i, cột j của một mảng word hai chiều A có $M \times N$ phần tử, được lưu trữ theo thứ tự hàng. Thủ tục nhận i trong AX, j trong BX, N trong CX và địa chỉ offset của A trong DX. Nó trả về địa chỉ offset của phần tử trong DX. **Chú ý:** bạn có thể bỏ qua khả năng tràn.

6. Để sắp xếp một mảng A gồm N phần tử bằng phương pháp nổi bọt, chúng ta thực hiện các bước sau:

Bước1: Cho j chạy từ 2 đến N, nếu $A[j] < A[j-1]$ ta đổi chỗ $A[j]$ và $A[j-1]$.

Bước này sẽ đưa phần tử lớn nhất đến vị trí N.

Bước2: Cho j chạy từ 2 đến N-1, nếu $A[j] < A[j-1]$ ta đổi chỗ $A[j]$ và $A[j-1]$.

Bước này sẽ đưa phần tử lớn thứ hai đến vị trí N-1.

BướcN-1: Nếu $A[2] < A[1]$ ta đổi chỗ $A[2]$ và $A[1]$. Lúc này mảng đã được sắp xếp.

Ví dụ:

| Giá trị ban đầu | 7 | 5 | 3 | 9 | 1 |
|-----------------|---|---|---|---|---|
| Bước1 | 5 | 3 | 7 | 1 | 9 |
| Bước2 | 3 | 5 | 1 | 7 | 9 |
| Bước3 | 3 | 1 | 5 | 7 | 9 |
| Bước4 | 1 | 3 | 5 | 7 | 9 |

Bạn hãy viết thủ tục BUBLE để sắp xếp một mảng byte bằng thuật toán nổi bọt. Thủ tục nhận địa chỉ offset của mảng trong SI và số các phần tử trong BX. Sau đó bạn hãy viết một chương trình để người sử dụng đánh vào một chuỗi các số có một chữ số ngăn cách nhau bằng một ký tự trắng và gọi BUBLE để sắp xếp chúng, cuối cùng in ra chuỗi đã sắp xếp trên dòng tiếp theo. Ví dụ:

```
? 2 4 3 7 1 5 6  
1 2 3 4 5 6 7
```

Chương trình của bạn phải làm việc được với mảng chỉ có một phần tử.

7. Giả thiết các bản ghi trong ví dụ phần 10.4.3 được lưu trữ như sau:

CLASS

```
DW    'ngọc lan ', 10,8,8,10  
DW    'kiều nga ', 8,8,9,10  
DW    'thu hiền ', 9,8,10,10  
DW    'ngọc thuý ', 8,7,8,10  
DW    'thanh hằng', 10,8,6,10
```

Mỗi tên chiếm 12 byte. Bạn hãy viết một chương trình in ra tên của mỗi học sinh cùng với với điểm trung bình của cô ta (làm tròn thành số nguyên) trong 4 bài thi.

8. Viết một chương trình bắt đầu bằng một mảng byte không có giá trị ban đầu có kích thước tối đa là 100 và cho phép người sử dụng thêm vào các ký tự sao cho mảng luôn được lưu trữ theo thứ tự tăng dần. Chương trình phải in ra một dấu chấm hỏi, để người sử dụng đánh vào một ký tự và hiển thị mảng có ký tự mới được chèn vào. Kết thúc nhập khi người sử dụng đánh vào phím ESC. Ký tự lặp lại được bỏ qua.

Một ví dụ khi thi hành:

```
?A  
A  
?D  
AD  
?B  
ABD  
?a  
ABDa  
?D  
ABDa  
?<ESC>
```

9. Viết một chương trình sử dụng lệnh XLAT để (1) đọc vào một dòng văn bản và (2) in ra dòng tiếp theo với tất cả các chữ thường được đổi thành chữ hoa. Trong dòng vào có thể chứa các ký tự bất kỳ: chữ thường, chữ hoa, chữ số, dấu chấm câu và v.v... .
10. Viết thủ tục PRINTHEX sử dụng lệnh XLAT để hiển thị nội dung của BX dưới dạng số hex có 4 chữ số. Kiểm tra nó bằng một chương trình dùng thuật toán nhập số hex trong mục 7.4 để người sử dụng nhập vào một số hex có 4 chữ số, chứa vào BX và gọi PRINTHEX để in nó ra trên dòng tiếp theo.

Chương 11

CÁC LỆNH THAO TÁC CHUỖI

Tổng quan

Trong chương này chúng sẽ xem xét một nhóm lệnh đặc biệt gọi là các lệnh thao tác chuỗi. Trong Hợp ngữ 8086 khái niêm chuỗi bộ nhớ hay chuỗi đơn giản là các mảng byte hay word. Do đó các lệnh thao tác chuỗi được thiết kế cho các thao tác với dữ liệu kiểu mảng.

Dưới đây là một số ví dụ về các công việc có thể thực hiện bằng các lệnh thao tác chuỗi :

- ◆ Chép một chuỗi sang chuỗi khác.
- ◆ Tìm một byte hay một word xác định trong một chuỗi.
- ◆ Lưu các ký tự vào trong chuỗi.
- ◆ So sánh các chuỗi theo thứ tự ái pha bê của các ký tự.

Các công việc thực hiện bằng các lệnh thao tác chuỗi có thể sử dụng các chế độ địa chỉ gián tiếp thanh ghi mà chúng ta đã học trong chương 10. Tuy nhiên các lệnh thao tác chuỗi có một số ưu điểm bên trong. Chẳng hạn nó có thể tự động điều chỉnh thanh ghi con trỏ và cho phép có những thao tác giữa bộ nhớ với bộ nhớ.

11.1 Cờ định hướng

Trong chương 5 chúng ta đã biết thanh ghi cờ bao gồm 6 cờ trạng thái và 3 cờ điều khiển. Chúng ta cũng biết rằng các cờ trạng thái phản ánh kết quả của một thao tác mà bộ xử lý thực hiện trong khi đó cờ điều khiển được sử dụng để điều khiển các thao tác của bộ xử lý.

Một trong những cờ điều khiển đó là cờ định hướng (DF- Direction Flag). Công dụng của nó là để xác định hướng cho các thao tác chuỗi. Các thao tác này được thực hiện bằng 2 thanh ghi chỉ số SI và DI. Ví dụ, chẳng hạn có một chuỗi được khai báo như sau:

```
STRING1 DB      'ABCDE'
```

Và chuỗi này được lưu trong bộ nhớ bắt đầu tại offset 0200h:

| Địa chỉ offset | Nội dung | Ký tự ASCII |
|----------------|----------|-------------|
| 0200h | 041h | A |
| 0201h | 042h | B |
| 0202h | 043h | C |
| 0203h | 044h | D |
| 0204h | 045h | E |

Nếu DF = 0, SI và DI được xử lý theo chiều tăng của các địa chỉ bộ nhớ: từ trái qua phải trong chuỗi. Ngược lại nếu DF = 1, SI và DI được xử lý theo chiều giảm dần các địa chỉ bộ nhớ : từ phải qua trái trong các chuỗi.

Trong DEBUG DF = 0 được ký hiệu là UP còn DF = 1 được ký hiệu là DN.

Các lệnh CLD và STD

Để làm cho DF = 0, chúng ta sử dụng lệnh CLD(Clear Direction flag):

```
CLD ; xoá cờ định hướng.
```

Để làm cho DF = 1 chúng ta sử dụng lệnh STD (SeT Direction flag):

```
STD ; thiết lập cờ định hướng
```

Các lệnh CLD và STD không làm ảnh hưởng tới các cờ k khác.

11.2 Lệnh chuyển một chuỗi

Giả sử chúng ta đã định nghĩa 2 chuỗi như sau:

```
.DATA  
STRING1 DB      'HELLO'  
STRING2 DB      5 DUP (?)
```

Và bây giờ chúng ta muốn chuyển nội dung của chuỗi STRING1(chuỗi nguồn) vào chuỗi STRING2 (chuỗi đích). Thao tác này rất cần thiết cho các công việc xử lý chuỗi như là lặp chuỗi hay nối chuỗi (gắn một chuỗi vào cuối chuỗi khác).

Lệnh MOVSB

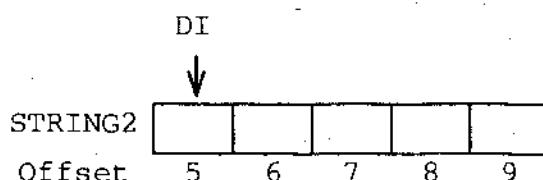
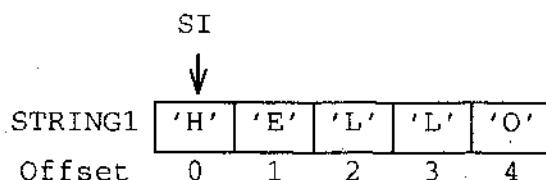
MOVSB ;chuyển một chuỗi các byte

sẽ chép nội dung của byte được định địa chỉ bởi DS:SI đến byte được định địa chỉ bởi ES:DI. Nội dung của byte nguồn không bị thay đổi. Sau khi byte được chuyển cả 2 thanh ghi SI và DI đều tự động tăng lên 1 nếu DF = 0 hay giảm đi 1 nếu DF = 1. Ví dụ nếu muốn chuyển 2 byte đầu tiên của STRING1 đến STRING2 chúng ta có thể thực hiện những lệnh sau:

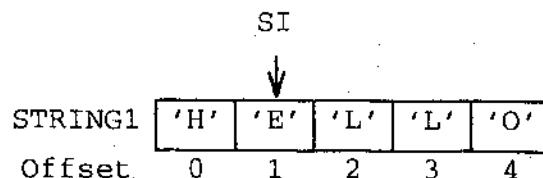
```
MOV AX, @DATA  
MOV DS, AX      ;Khởi tạo DS  
MOV ES, AX      ;và ES  
LEA SI, STRING1 ;SI trỏ đến chuỗi nguồn  
LEA DI, STRING2 ;DI trỏ đến chuỗi đích  
CLD             ;Xoá DF  
MOVSB           ;Chuyển byte đầu tiên  
MOVSB           ;rồi đến byte thứ 2
```

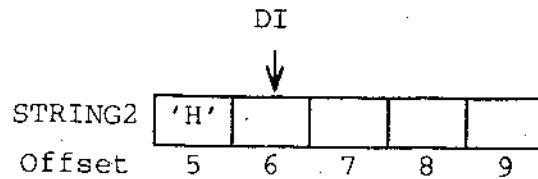
Xem hình 11.1

Trước lệnh MOVSB

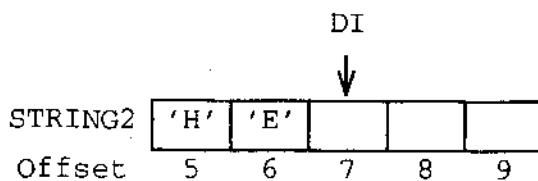
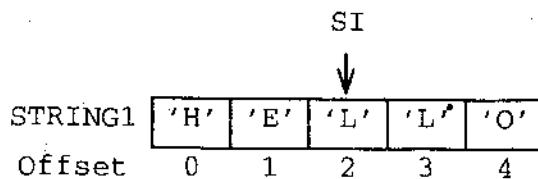


Sau lệnh MOVSB thứ nhất.





Sau lệnh MOVSB thứ hai.



Hình 11.1 Lệnh MOVSB

Lệnh MOVSB là lệnh đầu tiên cho phép thao tác với bộ nhớ mà chúng ta học, Đây cũng là lệnh đầu tiên liên quan đến thanh ghi ES.

Lệnh khởi tạo REP

Lệnh MOVSB chỉ chuyển 1 byte từ chuỗi nguồn đến chuỗi đích. Để chuyển cả chuỗi đầu tiên chúng ta phải khởi tạo CX với số N bằng số byte trong chuỗi nguồn và thực hiện lệnh sau:

REP MOVSB

Lệnh khởi tạo REP có tác dụng làm cho MOVSB được thực hiện N lần. Sau mỗi lệnh MOVSB, CX được giảm đi 1 cho đến khi nó bằng 0. Ví dụ để chuyển chuỗi STRING1 của ví dụ trước vào chuỗi STRING2, chúng ta có thể làm như sau:

```

CLD
LEA SI, STRING1
LEA DI, STRING2
MOV CX, 5      ; Số ký tự trong chuỗi STRING1

```

REP MOVSB

Ví dụ 11.1 Viết các lệnh chép chuỗi STRING1 trong phần trước vào chuỗi STRING2 theo thứ tự ngược lại

Lời giải:

Ý tưởng ở đây là cho SI trả đến cuối chuỗi STRING1 và DI trả đến đầu chuỗi STRING2 sau đó chuyển các ký tự trong khi SI di chuyển đọc theo chuỗi STRING1 từ trái qua phải.

```
LEA    SI, STRING1+4;SI trả đến cuối chuỗi STRING1  
LEA    DI, STRING2 ;DI trả đến đầu chuỗi STRING2  
STD          ;Xử lý từ phải qua trái  
MOV    CX, 5  
  
MOVE:  
    MOVSB           ;Chuyển từng byte  
    ADD    DI, 2  
    LOOP   MOVE
```

Ở trên chúng ta phải cộng thêm 2 vào DI sau mỗi lệnh MOVSB bởi vì khi DF = 1 thì sau mỗi lệnh MOVSB, SI và DI tự động giảm đi 1 trong khi chúng ta lại muốn tăng DI.

Lệnh MOVSW

Có dạng word cho lệnh MOVSB đó là:

MOVSW ;chuyển một chuỗi các word

Lệnh MOVSW chuyển từng word từ chuỗi nguồn đến chuỗi đích. Giống như lệnh MOVSB nó yêu cầu DS:SI trả đến chuỗi nguồn và ES:DI trả đến chuỗi đích. Sau khi một word của chuỗi được chuyển cả SI và DI cùng tăng lên 2 đơn vị nếu DF = 0 hoặc cùng giảm đi 2 nếu DF = 1.

MOVSB và MOVSW đều không làm ảnh hưởng tới cờ

Ví dụ 11.2 : Cho mảng sau đây:

ARR DW 10, 20, 40, 50, 60, ?

Hãy viết các lệnh để chèn 30 vào giữa 20 và 40 (giả thiết rằng DS và ES đã chứa địa chỉ đoạn dữ liệu).

Lời giải:

Chúng ta sẽ chuyển 40, 50 và 60 dịch lên một vị trí trong mảng rồi chèn 30 vào.

```
STD          ;Xử lý từ phải qua trái
LEA  SI,ARR+8h ;SI trở đến 60
LEA  DI,ARR+Ah ;DI trở đến ?
MOV  CX,3      ;Chuyển 3 phần tử
REP  MOVSW     ;Chuyển 40, 50 ,60
MOV  WORD PTR [DI],30 ;Chèn 30
```

Chú ý: toán tử PTR đã trình bày trong phần 10.2.3

11.3 Lệnh lưu chuỗi

Lệnh STOSB:

```
STOSB        ;Lưu chuỗi các byte
```

Có tác dụng chuyển nội dung của thanh ghi AL đến byte được định địa chỉ bởi ES:DI. Sau khi lệnh được thực hiện DI tăng 1 nếu DF = 0 hoặc giảm 1 nếu DF = 1. Tương tự như vậy lệnh STOSW:

```
STOSW        ;Lưu chuỗi các word
```

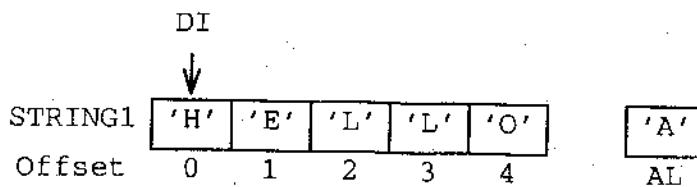
Chuyển nội dung của thanh ghi AX vào word được định địa chỉ bởi ES:DI và tăng hay giảm DI 2 đơn vị tùy theo trạng thái cờ DF.

Các lệnh STOSB và STOSW không ảnh hưởng tới cờ. Để làm ví dụ về lệnh STOSB, đoạn lệnh dưới đây sẽ lưu 2 chữ "A" vào chuỗi STRING1:

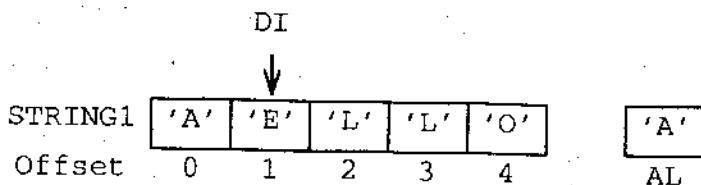
```
MOV  AX,@DATA
MOV  ES,AX      ;Khởi tạo ES
LEA  DI,STRING1 ;DI trở đến STRING1
CLD            ;Xử lý từ trái sang phải
MOV  AL,'A'     ;AL chứa ký tự cần lưu
STOSB         ;Lưu chữ cái "A"
STOSB         ;Lưu chữ thứ 2
```

Xem hình 11.2

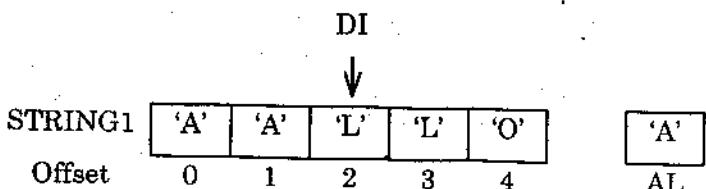
Trước lệnh STOSB



Sau lệnh STOSB thứ nhất.



Sau lệnh STOSB thứ hai.



Hình 11.2 Lệnh STOSB

Đọc và lưu một chuỗi ký tự

Hàm con số 1 của ngắt 21h đọc một ký tự từ bàn phím vào trong AL. Bằng cách lặp lại công việc này và kết hợp với lệnh STOSB chúng ta có thể đọc và lưu một chuỗi. Thêm vào đó chúng ta còn có thể xử lý các ký tự trước khi lưu chúng.

Thủ tục READ_STR dưới đây đọc và lưu các ký tự trong một chuỗi, cho đến khi người sử dụng đánh ENTER. Thủ tục được bắt đầu với địa chỉ offset của chuỗi ở trong DI. Sau khi thực hiện xong nó trả về địa chỉ offset chuỗi trong DI và số ký tự nhập vào trong BX. Nếu người sử dụng đánh nhầm một ký tự và dùng phím Backspace để xoá thì thủ tục cũng sẽ xoá ký tự đó khỏi chuỗi.

Thủ tục này tương tự như hàm 0Ah ngắt 21h của DOS (xem bài tập 11.11)

Thuật giải cho READ_STR

```
chars_read = 0
Đọc một ký tự
WHILE ký tự không phải CR DO
    IF ký tự là Backspace
        THEN
            chars_read = chars_read-1
            Xoá ký tự trước đó khỏi chuỗi
        ELSE
            Lưu ký tự vào chuỗi
            chars_read = chars_read+1
    END_IF
    Đọc tiếp ký tự
END WHILE
```

Chương trình nguồn PGM11_1.ASM

```
1: READ_STR PROC NEAR
2: ;đọc và lưu một chuỗi
3: ;Vào:DI chứa địa chỉ offset của chuỗi
4: ;RA :DI chứa địa chỉ offset của chuỗi
5: ;BX chứa số ký tự nhập được
6:     PUSH AX
7:     PUSH DI
8:     CLD           ;Xử lý từ bên trái sang
9:     XOR BX,BX    ;Số ký tự nhập được
10:    MOV AH,1      ;Hàm đọc một ký tự
11:    INT 21H       ;Đọc 1 ký tự vào AL
12: WHILE1:
13:     CMP AL,0DH   ;Ký tự CR?
14:     JE END_WHILE1 ;Đúng! kết thúc
15: ;Nếu ký tự là Backspace
16:     CMP AL,8H    ;Backspace?
17:     JNE ELSE1    ;Không! lưu nó vào chuỗi
18: ;Thì
19:     DEC DI       ;Đúng!, lùi con trỏ chuỗi
20:     DEC BX       ;Giảm bộ đếm số ký tự nhập được
21:     JMP READ     ;và đọc ký tự khác
22: ELSE1:
23:     STOSB        ;Lưu ký tự vào chuỗi
24:     INC BX       ;Tăng bộ đếm số KT
25: READ:
26:     INT 21H       ;Đọc ký tự vào AL
```

```

27:           JMP WHILE1      ;và tiếp tục vòng lặp
28: END_WHILE1:
29:           POP DI
30:           POP AX
31:           RET
32: READ_STR ENDP

```

Tại dòng 23 thủ tục sử dụng lệnh STOSB để lưu ký tự nhập được vào chuỗi. STOSB tự động tăng DI và tại dòng 24, bộ đếm số ký tự nhập được trong BX được tăng lên 1.

Thủ tục còn có khả năng quản lý các lỗi khi đưa ký tự vào. Nếu người sử dụng đánh phím Backspace thì tại dòng 19 thủ tục giảm DI và BX, bản thân ký tự Backspace cũng không được lưu. Khi ký tự tiếp theo được đọc nó sẽ thay thế ký tự bị lỗi trước đó. Chú ý rằng nếu phím Backspace được đánh ngay trước khi đánh ENTER thì ký tự lỗi vẫn nằm trong chuỗi nhưng số ký tự hợp lệ nhận được trong BX sẽ được điều chỉnh đúng.

Chúng ta sẽ sử dụng READ_STR cho việc nhập một chuỗi trong phần sau.

11.4 Nạp một chuỗi

Lệnh LODSB

LODSB ; nạp một chuỗi các byte

chuyển byte tại địa chỉ được chỉ ra bởi DS:SI vào AL. SI sau đó được tăng thêm 1 nếu DF = 0 và ngược lại bị giảm đi 1 nếu DF = 1. Dạng word của lệnh như sau:

LODSW ;nạp một chuỗi các word

Lệnh này chuyển word tại địa chỉ được chỉ ra bởi DS:SI vào AX. Sau đó SI được tăng hay giảm 2 tùy theo trạng thái cờ DF. Lệnh LODSB có thể sử dụng để kiểm tra các ký tự của một chuỗi như chúng ta sẽ thấy sau này.

Các lệnh LODSB và LODSW đều không tác động đến cờ. Để miêu tả lệnh LODSB chúng ta giả sử có chuỗi được định nghĩa như sau:

STRING1 DB 'ABC'

Đoạn lệnh dưới đây lần lượt nạp các byte thứ nhất và thứ 2 vào trong AL

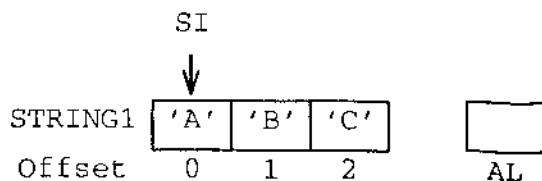
```

MOV AX, @DATA
MOV DS, AX ;Khởi tạo DS
LEA SI, STRING1 ;SI trỏ tới STRING1
CLD ;Xử lý từ trái sang phải
LODSB ;Nạp ký tự đầu tiên và AL
LODSB ;Nạp ký tự thứ 2 vào AL

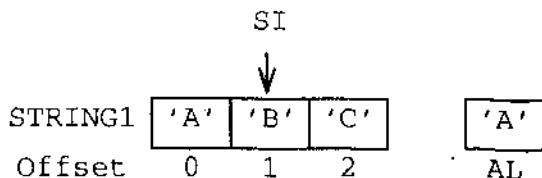
```

Xem hình 11.3

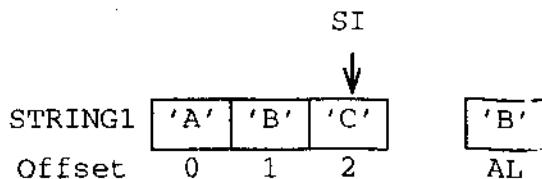
Trước lệnh LODSB



Sau lệnh LODSB thứ nhất.



Sau lệnh LODSB thứ hai.



Hình 11.3 Lệnh LODSB

Hiển thị một chuỗi ký tự

Thủ tục DISP_STR dưới đây hiển thị chuỗi được trả đến bởi SI với số ký tự trong BX. Thủ tục này có thể sử dụng để hiển thị toàn bộ hay một phần của chuỗi.

Thuật toán cho DISP_STR

```
FOR Đếm lần DO /*Đếm = số ký tự cần hiển thị*/  
    Nạp chuỗi ký tự vào AL  
    Chuyển vào DL  
    Hiển thị ký tự  
END_FOR
```

Chương trình nguồn PGM11_2.ASM

```
DISP_STR PROC NEAR  
;Hiển thị một chuỗi ký tự  
;Vào : SI = offset chuỗi  
;           BX = số các ký tự cần hiện thị  
;ra : none  
    PUSH AX  
    PUSH BX  
    PUSH CX  
    PUSH DX  
    PUSH SI  
    MOV CX,BX      ;khởi tạo bộ đếm trong DX  
    JCXZ P_EXIT ;Kết thúc nếu không có ký tự nào  
    CLD          ;Xử lý từ trái sang phải  
    MOV AH,2      ;Chuẩn bị đưa ký tự ra  
                ;thiết bị xuất lỗi chuẩn  
TOP:  
    LODSB        ;đưa 1 ký tự vào AL  
    MOV DL,AL     ;Chuyển nó vào trong DL  
    INT 21H       ;Hiển thị ký tự  
    LOOP TOP     ;Lặp lại cho đến khi  
                ;hiển thị hết  
P_EXIT:  
    POP SI  
    POP DX  
    POP CX  
    POP BX  
    POP AX  
    RET  
DISP_STR ENDP
```

Để xem READ_STR và DISP_STR hoạt động ra sao chúng ta viết một chương trình đọc một chuỗi ký tự (dài tối đa 80 ký tự) và hiển thị 10 ký tự đầu tiên của nó ở dòng tiếp theo.

Chương trình Nguồn PGM11_3.ASM

```
TITLE      PGM11_3:Kiểm tra 2 thủ tục READ_str Và
disp_str
.MODEL     SMALL
.STACK
.DATA
STRING    DB      80 DUP(0)
CRLF      DB      0DH,0AH,'$'
.CODE
MAIN      PROC
        MOV     AX,@DATA
        MOV     DS,AX
        MOV     ES,AX
;Đọc vào một chuỗi
        LEA     DI,STRING ;DI trả về tới chuỗi
        CALL   READ_STR ;BX = số ký tự nhập được
;Nhảy đến đầu dòng mới
        LEA     DX,CRLF ; SI trả về tới chuỗi
        MOV     AH,09H ; hiển thị 10 kí tự
        INT     21H
;Hiển thị chuỗi
        LEA     SI,STRING
        MOV     BX,10
        CALL   DISP_STR
;Trả về DOS
        MOV     AX,4CH
        INT     21H
MAIN      ENDP
;RAED_STR đặt ở đây
;DISP_STR đặt ở đây
        END   MAIN
```

Ví dụ chạy chương trình :

```
C>PGM11_3
THIS PROGRAM TETS TWO PROCEDURES
THIS PROGR
```

11.5 Lệnh duyệt chuỗi (Scan string)

Lệnh:

SCASB ; Duyệt một chuỗi các byte

có thể được sử dụng để tìm một byte (tạm gọi là byte đích- target byte) trong một chuỗi. Byte đích được chứa trong AL. Lệnh SCASB lấy nội dung của AL trừ đi từng byte trong chuỗi và sử dụng kết quả để thiết lập các cờ. Kết quả không được lưu lại và sau mỗi lần thực hiện phép trừ DI được tăng 1 nếu DF = 0 và giảm 1 nếu DF = 1.

Dạng word của lệnh như sau:

SCASW ; Duyệt chuỗi các word

Trong trường hợp này word đích đặt trong AX. SCASW trừ nội dung của AX cho từng word của chuỗi và thiết lập cờ. DI được tăng 2 nếu DF = 0 và giảm 2 nếu DF = 1.

Tất cả các cờ đều bị ảnh hưởng bởi 2 lệnh này.

Ví dụ nếu chuỗi :

STRING1 DB 'ABC'

được định nghĩa thì các lệnh sau sẽ kiểm tra xem 2 byte đầu của chuỗi có phải là chữ cái 'B' hay không

```
MOV AX, @DATA
MOV ES, AX          ; Khởi tạo ES
CLD                ; Xử lý từ trái sang
LEA DI, STRING1    ; DI trả đến chuỗi STRING1
MOV AL, 'B'         ; Ký tự đích
SCASB              ; Duyệt byte đầu tiên
SCASB              ; Duyệt byte thứ 2
```

Chúng ta hãy xem hình 11.4 chú ý rằng khi đích 'B' được tìm thấy, cờ ZF được thiết lập và do DI tự động được tăng thêm 1 nên DI trả tối kỵ tự đứng sau ký tự cần tìm.

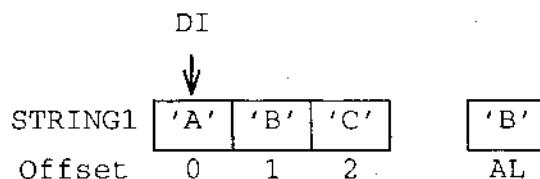
Khi tìm một byte đích trong chuỗi, chuỗi sẽ được duyệt cho đến khi byte được tìm thấy hoặc hết chuỗi. Nếu CX nhận giá trị đầu là số byte trong chuỗi,

REPNE SCASB ; lặp lại khi chưa bằng (chưa đến đích)

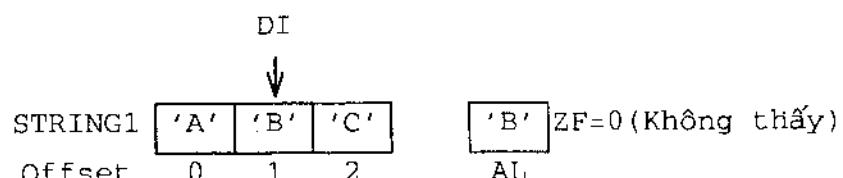
sẽ lặp lại phép trừ nội dung của AL đi từng byte trong chuỗi, điều chỉnh DI, giảm CX cho đến khi có kết quả 0 (khi tìm thấy byte đích) hoặc CX = 0 (khi chuỗi kết thúc).

Chú ý: Lệnh REPNZ (REPeat while Not Zero) tạo ra mã máy giống như lệnh REPNE (REPeat while Not Equal).

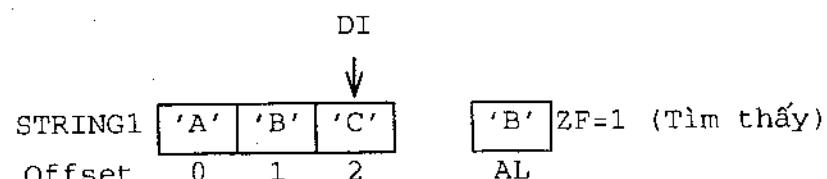
Trước lệnh SCASB



Sau lệnh SCASB thứ nhất.



Sau lệnh SCASB thứ hai.



Hình 11.4 Lệnh SCASB

Để làm ví dụ chúng hãy viết một chương trình đếm số nguyên âm và phụ âm trong một chuỗi.

Thuật toán đếm số nguyên âm và phụ âm của một chuỗi

Khởi tạo bộ đếm số nguyên âm (SNA) và số phụ âm (SPA) bằng 0

Đọc và lưu một chuỗi
 REPEAT
 Nạp một ký tự của chuỗi
 IF nó là nguyên âm
 THEN tăng SNA
 ELSE IF nếu nó là phụ âm
 THEN tăng SPA
 END_IF
 END_IF
 UNTIL hết chuỗi
 Hiển thị số nguyên âm
 Hiển thị số phụ âm

Chúng ta sẽ sử dụng thủ tục READ_STR để đọc một chuỗi. Thủ tục trả về với DI trả tới chuỗi và BX chứa số ký tự đọc được. Để hiển thị số nguyên âm và số phụ âm trong chuỗi chúng ta sẽ sử dụng thủ tục OUTDEC trong chương 9. Thủ tục này hiển thị nội dung của AX dưới dạng một số thập phân có dấu. Để đơn giản chúng ta giả thiết các ký tự được nhập đều là các chữ in hoa.

Chương trình Nguồn PGM11_4.ASM

```

0: TITLE      PGM11_4:Đếm số nguyên âm và phụ âm
1: .MODEL     SMALL
2: .STACK     100H
3: .DATA
4: STRING     DB    80 DUP(0)
5: NA          DB    'AEIOU'
6: PA          DB    'BCDFGHJKLMNPQRSTVWXYZ'
7: TB1         DB    0DH,0AH,'So nguyen am = $'
8: TB2         DB    ', So phu am = $'
9: SNA         DW    0
10: SPA        DW    0
11: .CODE.
12: MAIN       PROC
13:           MOV   AX,@DATA
14:           MOV   DS,AX      ;Khởi tạo DS
15:           MOV   ES,AX      ; và ES
16:           LEA   DI,STRING  ;DI trả đến chuỗi
17:           CALL  READ_STR  ;BX chứa số ký tự đọc được
18:           MOV   SI,DI      ;SI trả tới chuỗi
19:           CLD             ;Xử lý từ trái qua phải
20: REPEAT:
21: ;Nạp chuỗi ký tự
22: LODSB        ;Ký tự trong AL
  
```

```

23: ;Nếu nó là nguyên âm
24:         LEA    DI,NA ;DI trả tới chuỗi các nguyên âm
25:         MOV    CX,5      ;Có tất cả 5 nguyên âm
26:         REPNE SCASB;Ký tự có phải là nguyên âm?
27:         JNE    PHU_AM ;Không!, có thể là một phụ âm
28: ;Thì tăng số nguyên âm
29:         INC    SNA
30:         JMP    UNTIL
31: ;Hay nếu nó là phụ âm
32: PHU_AM:
33:         LEA    DI,PA ;DI trả tới chuỗi các phụ âm
34:         MOV    CX,21     ;Có tất cả 21 phụ âm
35:         REPNE SCASB     ;Ký tự có phải là phụ âm?
36:         JNE    UNTIL     ;Không!
37: ;Thì tăng số phụ âm
38:         INC    SPA
39: UNTIL:
40:         DEC    BX          ;BX chứa số ký tự còn lại
                           ;trong chuỗi
41:         JNE    REPEAT     ;Lặp lại nếu vẫn còn ký tự
                           ;trong chuỗi
42: ;Đưa ra số nguyên âm
43:         MOV    AH,9       ;Chuẩn bị đưa ra màn hình
44:         LEA    DX,TB1;Lấy thông báo về số nguyên âm
45:         INT    21H        ;Hiển thị nó
46:         MOV    AX,SNA       ;Lấy số nguyên âm
47:         CALL   OUTDEC     ;Hiển thị nó
48: ;Đưa ra số phụ âm
49:         MOV    AH,9       ;Chuẩn bị đưa ra màn hình
50:         LEA    DX,TB2     ;Lấy thông báo về số phụ âm
51:         INT    21H        ;Hiển thị nó
52:         MOV    AX,SPA       ;Lấy số phụ âm
53:         CALL   OUTDEC     ;Hiển thị nó
54: ;Trở về DOS
55:         MOV    AX,4CH
56:         INT    21H
57: MAIN    ENDP
58: ;READ_STR đặt ở đây
59: ;OUTDEC đặt ở đây
60:         END    MAIN

```

Vì chương trình sử dụng cả 2 lệnh LODSB để nạp byte tại địa chỉ DS:SI và SCASB để duyệt byte tại địa chỉ ES:DI, cả DS và ES đều phải khởi tạo để chúng chứa địa chỉ đoạn dữ liệu, BX được dùng như là bộ đếm vòng lặp và nó nhận giá

trị ban đầu bằng số ký tự trong chuỗi (CX được sử dụng ở nơi khác trong chương trình).

Dòng 22. Lệnh LODSB đưa ký tự vào AL và cho SI trỏ đến ký tự tiếp theo.

Dòng 26. Để xem ký tự trong AL có phải là nguyên âm hay không chương trình duyệt chuỗi NA bằng lệnh REPNE SCASB. Lệnh này trừ nội dung của AL cho mỗi byte của chuỗi NA và thiết lập cờ. Lệnh sẽ trả về cờ ZF = 1 nếu ký tự là nguyên âm và ZF = 0 nếu không phải.

Dòng 35. Nếu ký tự trong AL không phải nguyên âm, chương trình duyệt chuỗi PA theo cách giống như đã làm với chuỗi NA.

Ví dụ thực hiện

```
C>PGM11_4  
A,E,I,O,U LA CAC NGUYEN AM  
SO NGUYEN AM = 11, SO PHU AM = 7
```

11.6 Lệnh so sánh chuỗi

Lệnh CMPSB

CMPSB ; so sánh chuỗi các byte

trừ byte tại địa chỉ DS:SI cho byte tại địa chỉ ES:DI và thiết lập các cờ. Kết quả không được lưu lại. Và sau đó cả SI và DI cùng tăng 1 nếu DF = 0 hay giảm 1 nếu DF = 1.

Dạng word của lệnh CMPSB là:

CMPSW ; so sánh chuỗi các word

Lệnh này trừ word tại địa chỉ DS:SI cho word tại địa chỉ ES:DI và thiết lập các cờ. Nếu DF = 0 SI và DI được tăng 2 và ngược lại nếu DF = 1 SI và DI bị giảm đi 2. Lệnh CMPSW rất hữu hiệu khi dùng để so sánh các mảng số kiểu word.

Tất cả các cờ đều bị tác động bởi 2 lệnh CMPSB và CMPSW. Ví dụ chúng ta có:

```
.DATA  
STRING1    DB      'ACD'  
STRING2    DB      'ABC'
```

Đoạn lệnh dưới đây sẽ so sánh 2 byte đầu tiên của 2 chuỗi trên:

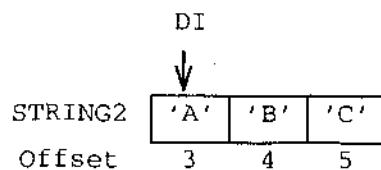
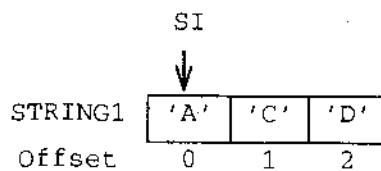
```

MOV AX, @DATA
MOV DS, AX ;Khởi tạo DS
MOV ES, AX; và ES
CLD ;Xử lý từ trái qua phải
LEA SI, STRING1 ;SI trỏ tới STRING1
LEA DI, STRING2 ;DI trỏ tới STRING2
CMPSB ;So sánh byte đầu tiên
CMPSB ;So sánh byte thứ 2

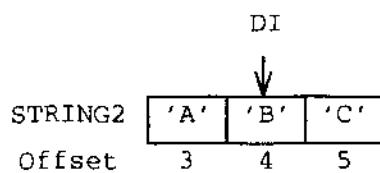
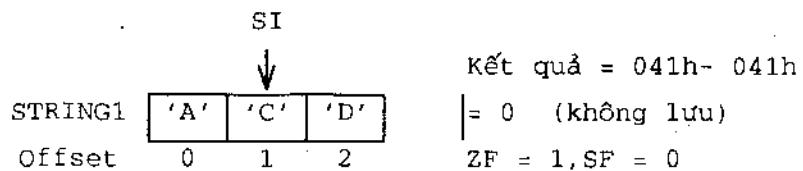
```

Xem hình 11.5

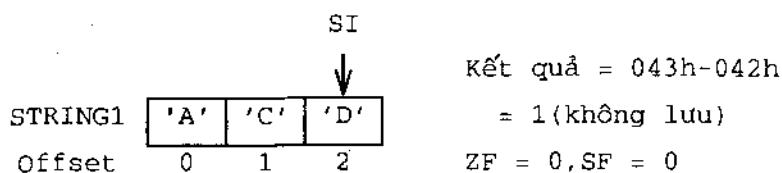
Trước lệnh CMPSB

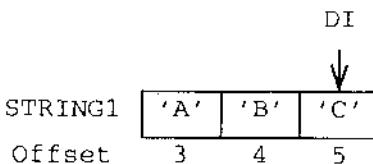


Sau lệnh CMPSB thứ nhất.



Sau lệnh CMPSB thứ hai.





Hình 11.5 Lệnh CMPSB

Các lệnh khởi tạo REPE và REPZ

Việc so sánh chuỗi có thể được thực hiện bằng cách gắn thêm lệnh khởi tạo REPE (Repeat While Equal) hay REPZ (Repeat While Zero) vào các lệnh CMPSB và CMPSW. CX nhận giá trị ban đầu là số byte trong chuỗi ngắn hơn sau đó ta có:

REPE CMPSB ; So sánh chuỗi các byte khi còn bằng nhau

hay

REPE CMPSW ; So sánh chuỗi các word khi còn bằng nhau

sẽ lặp lại việc thực hiện các lệnh CMPSB hay CMPSW rồi giảm CX cho đến khi

1. Có 2 byte tương ứng ở 2 chuỗi không bằng nhau
- hoặc 2. CX = 0.

Các cờ được thiết lập tùy theo kết quả của phép so sánh cuối cùng.

Lệnh CMPSB có thể sử dụng để so sánh 2 chuỗi xem chuỗi nào đứng trước theo thứ tự al pha bê, chúng có giống nhau hay chuỗi này có là chuỗi con của chuỗi kia không (có nghĩa là một chuỗi được chứa trong chuỗi còn lại như là một chuỗi các ký tự liên tiếp nhau).

Để làm ví dụ chúng ta giả sử STR1 và STR2 là 2 chuỗi có chiều dài 10. Đoạn lệnh sau đây sẽ đặt 0 vào AX nếu 2 chuỗi giống nhau, đặt 1 nếu chuỗi STR1 đứng trước chuỗi STR2 và đặt 2 nếu chuỗi STR2 đứng trước chuỗi STR1 (giả thiết rằng DS và ES đã được khởi tạo giá trị thích hợp).

```

MOV   CX,10      ;Chiều dài của các chuỗi
LEA   SI,STR1    ;SI trỏ đến STR1
LEA   DI,STR2    ;DI trỏ tới STR2
CLD
REPE  CMPSB     ;So sánh chuỗi các byte

```

```

JL      STR1_FIRST ;STR1 đứng trước STR2
JG      STR2_FIRST ;STR2 đứng trước STR1
;Đây là trường hợp 2 chuỗi giống nhau
MOV     AX, 0        ;Đặt 0 vào AX
JMP     EXIT         ;Và kết thúc
;Đây là trường hợp STR1 đứng trước STR2
STR1_FIRST:
MOV     AX, 1        ;Đặt 1 vào AX và
JMP     EXIT         ;Kết thúc
;Đây là trường hợp STR2 đứng trước STR1
STR2_FIRST:
MOV     AX, 2        ;Đặt 2 vào AX
EXIT:

```

11.6.1 Tìm một chuỗi con của một chuỗi

Có một số cách để xác định xem một chuỗi có phải là chuỗi con của chuỗi khác hay không. Phương pháp mà chúng tôi trình bày dưới đây có lẽ là một phương pháp đơn giản nhất. Giả sử chúng ta khai báo :

```

SUB1    DB    'ABC'
SUB2    DB    'CAB'
MAINST  DB    'ABABCA'

```

và chúng ta muốn biết SUB1 và SUB2 có phải là chuỗi con của MAINST hay không. Hãy bắt đầu từ chuỗi SUB1. Chúng ta có thể so sánh các ký tự tương ứng trong 2 chuỗi :

| | | | | | | |
|--------|---|---|---|---|---|---|
| SUB1 | A | B | C | | | |
| | | | + | | | |
| MAINST | A | B | A | B | C | A |

Vì có sự khác nhau trong phép so sánh thứ 3 chúng ta quay lại và thử so sánh SUB1 với phần của chuỗi MAINST từ vị trí MAINST+1 trở đi:

| | | | | | | |
|--------|---|---|---|---|---|---|
| SUB1 | A | B | C | | | |
| | + | | | | | |
| MAINST | A | B | A | B | C | A |

Vì có sự khác biệt ngay từ phép so sánh đầu tiên chúng ta lại bắt đầu lại từ đầu tại vị trí MAINST+2

| | | | | | | |
|--------|---|---|---|---|---|---|
| SUB1 | A | B | C | | | |
| | | | | | | |
| MAINST | A | B | A | B | C | A |

Lần này chúng ta đã thành công :SUB1 là chuỗi con của MAINST

Bây giờ hãy thử với chuỗi SUB2. Quá trình tìm kiếm cũng giống như trên cho đến khi chúng ta gặp trường hợp sau :

| | | | | | | |
|--------|---|---|---|---|---|---|
| SUB2 | C | A | B | | | |
| | + | | | | | |
| MAINST | A | B | A | B | C | A |

Có sự khác nhau trong phép so sánh nhưng ở đây chúng ta không cần làm tiếp vì nếu chúng ta cứ tiếp tục chúng ta sẽ phải so sánh 3 ký tự của chuỗi SUB2 với 2 ký tự còn lại của MAINST. Như vậy SUB2 không phải là chuỗi con của MAINST.

Thực tế chúng ta có thể đoán trước vị trí cuối cùng trong phép tìm kiếm. Đó là:

$$\begin{aligned} \text{STOP} &= \text{MAINST} + \text{chiều dài MAINST} - \text{chiều dài SUB2} \\ &= \text{MAINST} + 6 - 3 = \text{MAINST} + 3. \end{aligned}$$

Dưới đây là thuật toán và chương trình tìm chuỗi con SUBST trong chuỗi MAINST.

Thuật toán tìm chuỗi con

Thông báo cho người sử dụng đưa vào chuỗi con SUBST
Đọc SUBST

Thông báo cho người sử dụng đưa vào chuỗi MAINST
Đọc MAINST

IF (chiều dài của MAINST = 0)
OR (chiều dài của SUBST = 0) OR (SUBST dài
hơn MAINST)

THEN

 SUBST không phải là chuỗi con của MAINST

ELSE

 Tính vị trí dừng STOP

Vị trí bắt đầu START = offset MAINST
 ENDIF
 REPEAT
 So sánh các ký tự tương ứng trong MAINST bắt đầu
 từ START và SUBST
 IF tất cả các ký tự đều giống nhau
 THEN
 SUBST được tìm thấy trong MAINST
 ELSE
 START = START + 1
 END_IF
 UNTIL (SUBST được tìm thấy trong MAINST)
 OR (START > STOP)

Hiển thị kết quả

Sau khi đọc vào 2 chuỗi SUBST, MAINST, kiểm tra xem chúng khác rỗng và SUBST không dài hơn MAINST. Tại dòng 44 - 50 chương trình tính vị trí dừng STOP(vị trí trong MAINST để dừng lại việc tìm kiếm); và khởi tạo vị trí bắt đầu START(vị trí bắt đầu tìm kiếm) bằng đầu chuỗi MAINST.

Chương trình Nguồn PGM11_5.ASM

```

1: TITLE      PGM11_5:Tìm chuỗi con
2: .MODEL     SMALL
3: .STACK     100H
4: .DATA
5: MSG1       DB      'VAO CHUOI CON SUBST ',0AH,0DH,'$'
6: MSG2       DB      0AH,0DH,'VAO CHUOI CHINH MAINST '
               DB      0AH,0DH,'$'
7: MAINST    DB      80 DUP(0)
8: SUBST     DB      80 DUP(0)
9: STOP       DW      ?      ;Vị trí cuối cùng để tìm kiếm
10: START     DW      ?      ;Vị trí tiếp tục tìm kiếm
11: SUB_LEN   DW      ?      ;Chiều dài chuỗi con
12: YESMSG    DB      0AH,0DH,'SUBST la chuoi con cua' ''
               DB      'MAINST$'
13: NOMSG     DB      0AH,0DH,'SUBST khong la chuoi con cua' ''
               DB      'MAINST$'
14: .CODE
15: MAIN      PROC
16:           MOV AX,@DATA
17:           MOV DS,AX
18:           MOV ES,AX
19: ;Thông báo cho chuỗi con
  
```

```

20:      MOV AH,9          ;Hàm con in một chuỗi ký tự
21:      LEA DX,MSG1       ;Thông báo vào chuỗi con
22:      INT 21H
23: ;Đọc chuỗi SUBST
24:      LEA DI,SUBST     ;DI trả vào chuỗi con
25:      CALL READ_STR    ;BX chứa chiều dài chuỗi con
26:      MOV SUB_LEN,BX   ;Cất nó trong biến SUB_LEN
27: ;Thông báo cho chuỗi chính
28:      LEA DX,MSG2       ;Thông báo vào chuỗi MAINST
29:      INT 21H
30: ;Đọc chuỗi MAINST
31:      LEA DI,MAINST    ;DI trả tới chuỗi MAINST
32:      CALL READ_STR    ;BX chứa chiều dài chuỗi MAINST
33: ;Kiểm tra xem các chuỗi có rỗng hay SUBST có dài
;hơn MAINST không?
34:      OR BX,BX         ;MAINST rỗng?
35:      JE NO           ;Đúng! SUBST không phải là
;chuỗi con của MAINST
36:      CMP SUB_LEN,0    ;SUBST rỗng?
37:      JE NO           ;Đúng! SUBST không phải chuỗi con
38:      CMP SUB_LEN,BX   ;SUBST dài hơn MAINST?
39:      JG NO           ;Đúng! SUBST không phải chuỗi con
40: ;Kiểm tra xem SUBST có phải là chuỗi con của MAINST
;không
41:      LEA SI,SUBST     ;SI trả tới SUBST
42:      LEA DI,MAINST    ;DI trả tới MAINST
43:      CLD              ;Xử lý từ trái qua phải
44: ;Tính vị trí dừng STOP
45:      MOV STOP,DI       ;STOP chưa địa chỉ MAINST
46:      ADD STOP,BX       ;cộng thêm chiều dài MAINST
47:      MOV CX,SUB_LEN
48:      SUB STOP,CX       ;trừ đi chiều dài SUBST
49: ;Khởi tạo START
50:      MOV START,DI      ;Vị trí đầu tiên để tìm
51: REPEAT:
52: ;So sánh các ký tự
53:      MOV DI,START      ;Định lại DI
54:      LEA SI,SUBST      ;Định lại SI
55:      REPE CMPSB        ;So sánh các ký tự
56:      JE YES           ;Tìm thấy SUBST
57: ;Chưa tìm thấy SUBST
58:      INC START         ;Điều chỉnh lại START
59: ;Xem START còn nhỏ hơn STOP hay không
60:      MOV AX,START

```

```

62:           CMP   AX, STOP      ; START < STOP?
63:           JNLE NO          ; Không! kết thúc ngay
64:           JMP   REPEAT     ; Tiếp tục
65: ;Hiển thị kết quả
66: YES:
67:           LEA   DX, YESMSG    ; DX trả tới chuỗi YESMSG
68:           JMP   DISPLAY
69: NO:
70:           LEA   DX, NOMSG
71: DISPLAY:
72:           MOV   AH, 9
73:           INT   21H          ; Hiển thị kết quả
74: ;Trở về DOS
75:           MOV   AX, 4CH
76:           INT   21H
77: MAIN    ENDP
78: ;READ_STR đặt ở đây
79:           END   MAIN

```

Tại dòng 51 chương trình đi vào vòng lặp REPEAT trong đó các ký tự của SUBST được so sánh với một phần của MAINST từ vị trí START. Trong các dòng 53-56 CX được thiết lập bằng chiều dài của SUBST, SI trả tới SUBST, DI trả tới START và các ký tự tương ứng được so sánh bằng lệnh REPE CMPSB. Nếu ZF = 1, thì việc tìm kiếm thành công và chương trình nhảy tới dòng 66 tại đó nó hiển thị thông báo "SUBST LA CHUOI CON CUA MAINST". Nếu ZF = 0 nghĩa là đã có 2 ký tự khác nhau trong quá trình so sánh và START tăng thêm 1 ở dòng 59. Quá trình cứ tiếp tục như vậy cho đến khi SUBST giống một phần của MAINST hay START > STOP. Trong trường hợp sau thông báo "SUBST KHONG LA CHUOI CON CUA MAINST" được hiển thị.

Ví dụ thực hiện:

```

C>PGM11_5
VAO CHUOI CON SUBST
ABC
VAO CHUOI CHINH MAINST
XYZAABC
SUBST LA CHUOI CON CUA MAINST

```

```

C>PGM11_5
VAO CHUOI CON SUBST
ABD
VAO CHUOI CHINH MAINST
ABACADACD
SUBST KHONG LA CHUOI CON CUA MAINST

```

11.7 Dạng tổng quát của các lệnh thao tác chuỗi

Chúng ta hãy tổng kết các dạng byte và word của các lệnh thao tác chuỗi:

| Lệnh | Toán hạng đích | Toán hạng nguồn | Dạng byte | Dạng word |
|----------------|----------------|-----------------|-----------|-----------|
| Chuyển chuỗi | ES:DI | DS:SI | MOVSB | MOVSW |
| Số sánh chuỗi* | ES:DI | DS:SI | CMPSB | CMPSW |
| Lưu chuỗi | ES:DI | AL hay AX | STOSB | STOSW |
| Nạp chuỗi | AL hay AX | DS:SI | LODSB | LODSW |
| Duyệt chuỗi | ES:DI | AL hay AX | SCASB | SCASW |

*Kết quả không lưu lại.

Các toán hạng của các lệnh này là ngầm định bởi vậy bản thân chúng không còn là một phần của lệnh nữa. Tuy nhiên cũng có một số dạng của các lệnh thao tác chuỗi trong đó các toán hạng có mặt một cách rõ ràng. Đó là:

Lệnh

Ví dụ

| | | |
|-------|-------------------------|--------------|
| MOVS | chuỗi đích, chuỗi nguồn | MOVSB |
| CMPSS | chuỗi đích, chuỗi nguồn | CMPSB |
| STOS | chuỗi đích | STOS STRING2 |
| LODS | chuỗi nguồn | LODS STRING1 |
| SCAS | chuỗi đích | SCAS STRING2 |

Khi trình biên dịch gấp các dạng tổng quát này chúng sẽ kiểm tra xem:

1. Chuỗi nguồn có nằm trong đoạn định địa chỉ bởi DS và chuỗi đích có nằm trong đoạn định địa chỉ bởi ES hay không.
2. Trong trường hợp của lệnh MOVS và CMPS nếu các chuỗi có cùng kiểu nghĩa là cùng là chuỗi byte hay chuỗi word. Thì khi đó lệnh sẽ được mã hoá như là dạng byte (chẳng hạn MOVSB) hay dạng word (chẳng hạn MOVSW) tùy theo dạng dữ liệu mà các chuỗi được khai báo. Ví dụ DS và ES chứa địa chỉ đoạn của đoạn sau đây:

```
.DATA
STRING1 DB      'ABCDE'
STRING2 DB      'EFGH'
STRING3 DB      'IJKL'
STRING4 DB      'MNOP'
STRING5 DW      1,2,3,4,5
STRING6 DW      7,8,9
```

Thì những cặp lệnh sau đây là tương đương:

| | | |
|------|------------------|-------|
| MOVS | STRING1, STRING2 | MOVSB |
| MOVS | STRING6, STRING5 | MOVSW |
| LODS | STRING4 | LODSB |
| LODS | STRING5 | LODSW |
| SCAS | STRING1 | SCASB |
| STOS | STRING6 | STOSW |

Cần phải lưu ý các bạn một lần nữa là khi sử dụng dạng tổng quát vẫn cần phải đảm bảo cho DS:SI trả đến chuỗi nguồn và ES:DI trả đến chuỗi đích. Dạng tổng quát của các lệnh thao tác chuỗi có những ưu điểm cũng như những nhược điểm của nó. Ưu điểm của nó là ở chỗ do các toán hạng xuất hiện khá rõ ràng trong lệnh, chương trình trở lên dễ đọc hơn. Nhược điểm của nó là chỉ khi xác định dạng dữ liệu khai báo mới biết được dạng lệnh là byte hay word. Trong thực tế các toán hạng có mặt trong dạng tổng quát của lệnh thao tác chuỗi chưa chắc đã là các toán hạng được sử dụng khi lệnh được thực hiện! Để làm ví dụ các bạn hãy xem đoạn lệnh dưới đây:

```
LEA      SI, STRING1      ;SI trả đến STRING1
LEA      DI, STRING2      ;DI trả đến STRING2
MOVS   STRING3, STRING4
```

Mặc dù các toán hạng nguồn và đích đã được xác định là STRING3 và STRING4. Nhưng khi lệnh MOVS được thực hiện thì byte đầu tiên của chuỗi STRING1 được chuyển đến byte đầu tiên của chuỗi STRING2 vì trình biên dịch đã dịch lệnh MOVS STRING3, STRING4 thành mã máy của lệnh MOVSB trong khi SI và DI lại trả đến các byte đầu tiên của các chuỗi STRING1 và STRING2 một cách tương ứng.

TỔNG KẾT

Trong chương này chúng ta đã nghiên cứu những vấn đề sau:

- ◆ Các lệnh thao tác chuỗi là một nhóm đặc biệt của các lệnh xử lý dữ liệu kiểu mảng. Trạng thái cờ DF xác định chiều của thao tác chuỗi. Nếu DF = 0 thao tác được thực hiện từ trái qua phải trong chuỗi, và ngược lại nếu DF = 1 thao tác được thực hiện từ phải qua trái. Lệnh CLD xoá cờ DF(DF = 0) và lệnh STD thiết lập cờ DF(DF = 1).
- ◆ Lệnh MOVSB chuyển một byte tại địa chỉ DS:SI đến Bbyte tại địa chỉ ES:DI và sau đó DI, SI được điều chỉnh tùy theo giá trị cờ DF. Lệnh MOVSB có dạng word là MOVSW. Các lệnh này có thể sử dụng với tiền tố REP để lặp lại các thao tác nói trên CX lần.
- ◆ REPE và REPNE là các tiền tố điều kiện có thể dùng với các lệnh thao tác chuỗi. Với tiền tố REPE lệnh được lặp lại CX lần nếu ZF vẫn bằng 1. Với tiền tố REPNE lệnh được lặp lại CX lần nếu ZF vẫn bằng 0. REPZ và REPNZ cũng có tác dụng giống như REPE và REPNE.
- ◆ Lệnh STOSB chuyển nội dung của AL vào byte được định địa chỉ bởi ES:DI và sau đó điều chỉnh DI tùy theo trạng thái của cờ DF. STOSB có dạng word là STOSW. STOSB có thể sử dụng để đọc một chuỗi ký tự vào một mảng.
- ◆ Lệnh LODSB chuyển vào AL một byte được định địa chỉ bởi DS:SI và sau đó điều chỉnh SI tùy theo trạng thái của cờ DF. LODSB có dạng word là LODSW. LODSB có thể sử dụng để kiểm tra nội dung của một chuỗi ký tự.
- ◆ Lệnh SCASB trừ nội dung của AL cho byte được chỉ ra bởi ES:DI và sử dụng kết quả để thiết lập các cờ. Kết quả không được lưu lại và DI được điều chỉnh tùy theo trạng thái của DF. SCASW là dạng word của lệnh SCASB. Lệnh này trừ nội dung của AX cho word được chỉ ra bởi ES:DI, thiết lập các cờ và điều chỉnh DI tùy theo trạng thái của DF. Kết quả cũng không được lưu lại. Các lệnh này có thể sử dụng để tìm một byte hay một word đích (chứa trong AL hay AX) trong một chuỗi.
- ◆ Lệnh CMPSB trừ byte được chỉ ra bởi DS:SI cho byte được chỉ ra bởi ES:DI, thiết lập các cờ và điều chỉnh cả SI lẫn DI tùy theo trạng thái của DF. Kết quả không được lưu lại. CMPSB có dạng word là CMPSW. Các lệnh này có thể sử dụng để so sánh 2 chuỗi theo thứ tự al pha bê, để kiểm tra xem chúng có bằng nhau hoặc chuỗi này có là chuỗi con của chuỗi kia hay không.
- ◆ Các lệnh thao tác chuỗi có dạng tổng quát, trong đó các toán hạng được xác định rõ ràng. Trình biên dịch chỉ sử dụng các toán hạng để quyết định mã hoá lệnh theo dạng byte hay word.

Thuật ngữ tiếng Anh

(Memory) string Chuỗi - Một mảng các byte hay word

Các lệnh mới

| | | |
|-------|-------|-------|
| CLD | LODSW | SCASW |
| CMPS | MOVS | STD |
| CMPSB | MOVSB | STOS |
| CMPSW | MOVSW | STOSB |
| LODS | SCAS | STOSW |
| LODSB | SCASB | |

Các lệnh thao tác chuỗi :

| | | |
|------|--------|------|
| REP | REPNE | REPZ |
| REPE | REPNEZ | |

Bài tập

1. Giả sử

| | |
|---------------|--------------------|
| SI chứa 100h | byte 100h chứa 10h |
| DI chứa 200h | byte 101h chứa 15h |
| AX chứa 4142h | byte 200h chứa 20h |
| DF = 0 | byte 201h chứa 25h |

Hãy cho biết toán hạng đích, toán hạng nguồn và các giá trị được chuyển trong mỗi lệnh sau đây, đồng thời cho biết giá trị mới của SI và DI.

- a. MOVSB
- b. MOVSW
- c. STOSB
- d. STOSW
- e. LODSB
- f. LODSW

2. Giả sử có các khai báo sau đây:

```
STRING1 DB      'FGHIJ'  
STRING2 DB      'ABCDE'  
DB 5 DUP(?)
```

Hãy viết các lệnh chuyển STRING1 vào cuối STRING2 để tạo ra chuỗi "ABCDEFGHIJ".

3. Viết các lệnh đổi chuỗi STRING1 và chuỗi STRING2 trong bài tập 2. Các bạn có thể sử dụng 5 byte sau chuỗi STRING2 làm vùng nhớ trung gian.

4. Chuỗi ASCIIZ là một chuỗi kết thúc bằng byte 0, ví dụ:

```
STR  DB      'THIS IS A  ASCIIZ STRING', 0
```

Hãy viết thủ tục LENGTH nhận vào địa chỉ của một chuỗi ASCIIZ trong DX và trả về chiều dài của chuỗi trong CX.

5. Hãy sử dụng các chế độ địa chỉ trong chương 10 để viết những lệnh tương đương với các lệnh thao tác chuỗi dưới đây. Giả sử rằng ở những chỗ cần thiết SI đã chứa offset của chuỗi nguồn và DI đã chứa offset của chuỗi đích đồng thời DF = 0. Các bạn có thể dùng AL làm vùng nhớ trung gian. Đối với các lệnh SCASB và CMPSB các cờ phải phản ánh kết quả của phép so sánh.

- a. MOVSB
- b. STOSB
- c. LODSB
- d. SCASB
- e. CMPSB

6. Giả sử đã khai báo chuỗi sau đây:

```
STRING    DB      'TH*S* G*S* AR* b*ASTS'
```

Hãy viết các lệnh đổi các dấu hoa thị thành các chữ 'E'.

7. Giả sử các chuỗi sau đã được khai báo:

```
STRING1   DB      'T H I S I S A T E S T'  
STRING2   DB      11 DUP (?)
```

Viết các lệnh chép chuỗi STRING1 vào chuỗi STRING2 và bỏ đi những khoảng trống.

Các bài tập lập trình

8. Một "Palindrome" là một chuỗi ký tự mà khi đọc ngược hay xuôi đều giống nhau. Khi nói một chuỗi là "Palindrome" chúng ta đã bỏ qua những khoảng trống, các dấu và các chữ hoa. Ví dụ như "Madam, I'm Adam" hay "A man, a plan, Panama!"

Viết một chương trình :

1. Cho phép người sử dụng đưa vào một chuỗi
2. In nó ngược và xuôi không có các dấu và khoảng trống trên dòng tiếp theo.
3. Kiểm tra xem nó có phải là một "Palindrome" hay không và in ra kết luận.

9. Trong một số ứng dụng ..., rất cần phải hiển thị các số căn theo biên phải trong một khoảng nhất định. Ví dụ các số dưới đây được căn theo biên phải trong một khoảng bằng 10 ký tự:

```
12345  
23423345  
123
```

Hãy viết chương trình nhập 10 số có tối đa 10 chữ số và hiển thị chúng như trên đây.

10. Chuỗi STRING1 được gọi là đứng trước chuỗi STRING2 theo thứ tự al pha bê nếu một trong 3 khả năng sau xảy ra:

- a. Ký tự đầu tiên của STRING1 đứng trước ký tự đầu tiên của STRING2 theo thứ tự al pha bê

- b. N-1 ký tự đầu tiên của 2 chuỗi giống nhau nhưng ký tự thứ N của STRING1 đứng trước ký tự thứ N của STRING theo thứ tự al pha bê.
- c. STRING1 giống phần đầu tiên của STRING2 nhưng STRING 2 dài hơn.

Hãy viết chương trình cho phép người sử dụng đưa vào 2 chuỗi trên 2 dòng khác nhau. Kiểm tra xem chuỗi nào đứng trước theo thứ tự al pha bê hay 2 chuỗi có giống nhau không.

11. Hàm con 0Ah của ngắt 21h có thể dùng để nhập một chuỗi ký tự. Ký tự đầu tiên của mảng chứa chuỗi (bộ đệm chuỗi) phải được khởi tạo bằng số ký tự lớn nhất muốn nhập. Sau khi thực hiện hàm byte thứ 2 chứa số ký tự thực sự được nhập vào. Việc nhập kết thúc bằng ký tự CR, ký tự này cũng được lưu nhưng nó không được tính vào số các ký tự được nhập. Nếu người sử dụng đưa vào số ký tự nhiều hơn số ký tự mong đợi máy tính sẽ phát ra các tiếng bip.

Hãy viết chương trình in ra dấu "?"; đọc chuỗi nhiều nhất là 20 ký tự (sử dụng hàm 0Ah của ngắt 21h) và hiển thị chuỗi nhập được ở dòng tiếp theo. Khởi tạo bộ đệm như sau:

```

STRING      LABEL  BYTE
MAX_LEN    DB      20      ;Số ký tự lớn nhất mong
                           ;muốn
ACT_LEN    DB      ?       ;Số ký tự thực sự nhập
                           ;được
CHARS     DB      21 DUP (?) ;20 byte cho chuỗi và
                           ;1 byte cho ký tự CR
                           ; Return

```

12. Viết một thủ tục INSERT chèn chuỗi STRING1 vào chuỗi STRING2 tại một vị trí nhất định.

Vào

SI chứa địa chỉ offset của STRING1
 DI chứa địa chỉ offset của STRING2
 BX chứa chiều dài của STRING1
 CX chứa chiều dài của STRING2
 AX chứa địa chỉ offset tại đó STRING1 được chèn vào

Ra

DI chứa địa chỉ offset của chuỗi mới
 BX chứa chiều dài mới của chuỗi

Thủ tục có thể giả sử rằng không có chuỗi nào có chiều dài bằng 0 và địa chỉ trong AX nằm trong chuỗi STRING2.

Hãy viết một chương trình vào 2 chuỗi STRING1 và STRING2, một số không âm N ($0 \leq N \leq 40$), chèn chuỗi STRING1 vào chuỗi STRING2 sau N byte tính từ đầu chuỗi. Hiển thị chuỗi nhận được. Các bạn có thể giả thiết rằng $N \leq$ chiều dài của STRING2 và chiều dài của mỗi chuỗi nhỏ hơn 40.

13. Viết một thủ tục DELETE xoá N byte từ vị trí xác định trong một chuỗi và bỏ đi các chỗ trống mới xuất hiện(trong bộ nhớ).

Vào

- DI chứa địa chỉ offset của chuỗi.
- BX chứa chiều dài của chuỗi.
- CX chứa số ký tự cần xoá
- SI chứa địa chỉ offset mà các byte từ vị trí đó bị xoá đi.

Ra

- DI chứa địa chỉ offset của chuỗi mới.
- BX chiều dài của chuỗi mới.

Thủ tục có thể giả thiết rằng chuỗi có chiều dài khác 0, số byte cần xoá nhỏ hơn chiều dài của chuỗi, và địa chỉ cho trong SI nằm bên trong chuỗi.

Viết chương trình đọc một chuỗi STRING, số nguyên thập phân S chứa vị trí trong chuỗi STRING và số nguyên thập phân N các byte sẽ bị xoá khỏi chuỗi (cả 2 số đều nằm trong khoảng từ 0 đến 80). Chương trình gọi DELETE để xoá N byte từ vị trí S trong chuỗi STRING và hiển thị chuỗi nhận được. Bạn có thể giả thiết $0 \leq N \leq L - S$, trong đó L là chiều dài của STRING.

Chương 12

HIỂN THỊ KÝ TỰ VÀ LẬP TRÌNH BÀN PHÍM.

Tổng quan

Một trong những ứng dụng hữu ích và lý thú nhất của Hợp ngữ đó là việc điều khiển màn hình. Trong chương này, chúng ta lập trình thực hiện các thao tác như di chuyển con trỏ, cuộn cửa sổ trên màn hình và hiển thị con trỏ với thuộc tính bất kỳ. Đồng thời chúng tôi cũng chỉ ra cách thức lập trình với bàn phím để khi người sử dụng nhấn một phím, một chức năng điều khiển màn hình sẽ được thi hành. Để làm ví dụ, chúng tôi sẽ chỉ ra làm thế nào để thao tác với các phím mũi tên.

Sự hiển thị trên màn hình được xác định bằng dữ liệu chứa trong bộ nhớ. Chương này chúng ta bắt đầu với việc xem xét cách thức phát sinh sự hiển thị trên màn hình và điều khiển nó bằng cách truy nhập trực tiếp bộ nhớ hiển thị. Tiếp theo chúng tôi sẽ chỉ ra cách thức thao tác màn hình bằng các lời gọi hàm của BIOS. Các hàm này có thể dùng để kiểm tra các phím nhấn và để minh họa, chúng tôi sẽ viết một chương trình soạn thảo màn hình đơn giản.

12.1 Màn hình.

Màn hình của máy vi tính thao tác giống hệt như màn hình một vô tuyến thông thường. Một súng điện tử được dùng để bắn tia điện tử lên màn hình phết phơ tạo nên các điểm sáng. Tia điện tử quét qua màn hình theo các dòng và các điểm ảnh được tạo nên tuỳ thuộc vào sự bật tắt dòng điện tử đang dịch chuyển này. Một chu kỳ quét dòng bắt đầu với chùm tia ở góc trái, quét qua góc phải rồi được tắt đi và định vị lại tại vị trí bắt đầu của dòng tiếp theo. Quá trình

này được lắp lại đến tận dòng cuối cùng, lúc này tia điện tử sẽ được định vị đến góc trái trên và lắp lại sự quét từ dòng đầu tiên.

Có hai loại màn hình: màn hình đơn sắc và màn hình màu. Màn hình đơn sắc dùng một chùm tia điện tử và chỉ hiển thị một màu, thường là màu hổ phách hoặc màu xanh lá cây. Bằng cách thay đổi cường độ chùm tia điện tử có thể tạo ra các điểm ảnh có màu sắc khác nhau; điều này còn gọi là cân bằng độ xám (gray scale).

Màn hình màu được tráng bởi ba loại phốt pho tương ứng với ba màu cơ bản đỏ, xanh lá cây và xanh da trời. Ba chùm tia điện tử được sử dụng để viết các điểm ảnh trên màn hình: mỗi trong chúng được dùng để hiển thị các màu khác nhau. Thay đổi cường độ các chùm tia điện tử tạo nên các điểm ảnh đỏ, xanh lá cây và xanh da trời có cường độ khác nhau. Bởi lẽ ba điểm ảnh này rất gần nhau nên mắt người chỉ cảm nhận chúng như một điểm ảnh đồng nhất. Đây là nguyên tắc tạo ra các màu khác nhau của màn hình.

12.2 Bộ phối ghép màn hình và các chế độ hiển thị.

Các bộ phối ghép màn hình.

Việc hiển thị trên màn hình được điều khiển bởi một vi mạch của máy tính gọi là bộ phối ghép màn hình. Vi mạch này thường nằm trên một card phụ, trong đó có hai khối cơ bản: một bộ nhớ hiển thị (còn gọi là bộ đệm màn hình) và một bộ điều khiển màn hình.

Bộ nhớ màn hình chứa các thông tin để hiển thị. Nó có thể được truy nhập bởi cả CPU hay bộ điều khiển màn hình. Địa chỉ của bộ nhớ màn hình bắt đầu từ đoạn A000h, và trên, tuy nhiên địa chỉ này còn phụ thuộc vào từng bộ phối ghép màn hình cụ thể.

Bộ điều khiển màn hình đọc bộ nhớ hiển thị và phát sinh các tín hiệu video tương ứng đến màn hình. Để hiển thị màu, bộ phối ghép có thể phát sinh ba tín hiệu tách biệt cho các màu đỏ, xanh lá cây và xanh da trời hoặc đưa ra một tín hiệu chung được tổng hợp từ ba tín hiệu tách biệt đó. Một màn hình tổng hợp sử dụng các tín hiệu tổng hợp, còn một màn hình RGB sử dụng các tín hiệu tách biệt. Tín hiệu tổng hợp có chứa một tín hiệu bật màu. Khi không có tín hiệu này màn hình hiển thị với hai màu đen và trắng.

Các chế độ hiển thị.

Thông thường chúng ta thấy cả văn bản và các hình vẽ hiển thị được trên màn hình. Máy tính có các kỹ thuật và bộ nhớ khác nhau phục vụ cho việc hiển thị văn bản và đồ họa. Như vậy bộ phối ghép có hai chế độ hiển thị: văn bản và đồ họa. Trong chế độ đồ họa, màn hình được chia thành các cột và các dòng, thông thường là 80 cột và 25 hàng và các ký tự được hiển thị tại các vị trí xác định. Trong chế độ đồ họa, màn hình cũng được chia ra thành các cột và các dòng và mỗi vị trí trên màn hình gọi là điểm ảnh. Một bức tranh có thể được hiển thị bằng cách xác định màu cho mỗi điểm ảnh trên màn hình. Trong chương này chúng ta sẽ nghiên cứu chế độ văn bản, chế độ đồ họa được trình bày trong chương 16.

Chúng ta hãy xem xét tỉ mỉ hơn phương thức phát sinh ký tự trong chế độ văn bản. Mỗi ký tự trên màn hình được tạo nên từ một ma trận điểm gọi là ma trận ký tự. Bộ phối ghép sử dụng một vi mạch phát sinh ký tự tạo ra các ma trận điểm mẫu. Số điểm của mỗi ma trận phụ thuộc vào độ phân giải của bộ phối ghép-tham số quy định số các điểm có thể phát sinh trên màn hình. Bản thân màn hình cũng có độ phân giải riêng, và như vậy bắt buộc phải có sự tương thích giữa màn hình và bộ phối ghép màn hình.

Các bộ phối ghép màn hình.

Bảng 12.1 liệt kê các bộ phối ghép màn hình của IBM PC. Chúng khác nhau về độ phân giải và số màu hiển thị.

IBM PC cung cấp hai loại bộ phối ghép cho PC chuẩn: MDA (Monochrom Display Adapter) và CGA (Color Graphic Adapter). MDA chỉ có thể hiển thị chế độ văn bản và dùng cho các phần mềm thương mại như các chương trình xử lý văn bản và bảng tính điện tử là những chương trình không dùng chế độ đồ họa. MDA có độ phân giải khá cao với ma trận ký tự 9×14 điểm. CGA có thể hiển thị màu ở cả chế độ văn bản và đồ họa nhưng có độ phân giải thấp hơn. Trong chế độ văn bản, mỗi ma trận ký tự chỉ gồm 8×8 điểm.

Năm 1984 IBM đưa ra bộ phối ghép EGA (Enhanced Graphic Adapter) có độ phân giải cao và đồ họa màu. Ma trận ký tự 8×14 điểm.

Năm 1988 IBM giới thiệu các máy PS/2 được trang bị các bộ phối ghép VGA (Video Graphic Array) và MCGA (Multi-color Graphic Array). Nhưng bộ phối ghép này có độ phân giải cao nhất và có thể hiển thị nhiều màu hơn trong chế độ đồ họa so với EGA. Ma trận ký tự là 8×19 .

Bảng 12.1 Các bộ phối ghép màn hình.

| VIẾT TẮT | Ý NGHĨA |
|----------|---------------------------|
| MDA | Monochrom Display Adapter |
| CGA | Color Graphic Adapter |
| EGA | Enhanced Graphic Adapter |
| MCGA | Multi-color Graphic Array |
| VGA | Video Graphic Array |

Các chế độ.

Phụ thuộc vào bộ phối ghép hiện tại, một chương trình có thể chọn chế độ văn bản hay đồ họa. Mỗi chế độ được xác định bằng một số. Bảng 12.2 liệt kê các chế độ văn bản với các bộ phối ghép khác nhau.

Bảng 12.2 Chế độ văn bản của các bộ phối ghép màn hình.

| CHẾ ĐỘ | MÔ TẢ | CÁC BỘ PHỐI GHÉP |
|--------|----------------------------|---------------------|
| 0 | 40x25 văn bản 16 màu (xám) | CGA, EGA, MCGA, VGA |
| 1 | 40x25 văn bản 16 màu | CGA, EGA, MCGA, VGA |
| 2 | 80x25 văn bản 16 màu (xám) | CGA, EGA, MCGA, VGA |
| 3 | 80x25 văn bản 16 màu | CGA, EGA, MCGA, VGA |
| 7 | 80x25 văn bản đơn sắc | MDA, EGA, VGA |

Chú ý: Các chế độ 0 và 2 tắt tín hiệu bật màu với các màn hình tổng hợp, các màn hình RGB vẫn hiển thị 16 màu.

12.3 Lập trình trong chế độ văn bản.

Để tiện theo dõi, chúng ta quy ước làm việc với chế độ văn bản 80 cột, 25 dòng. Tuy nhiên các bộ phối ghép đồ họa màu vẫn có thể hiển thị 40 cột, 25 dòng.

Mỗi vị trí trên màn hình có thể xác định bằng cách đưa ra tọa độ (cột, dòng) của nó. Góc trái trên có tọa độ (0, 0). Trong chế độ hiển thị 80x25, các dòng có

giá trị 0–24, các cột có giá trị 0–79. Bảng 12.3 cho biết tọa độ của một vài vị trí trên màn hình.

Ký tự hiển thị tại một vị trí trên màn hình được xác định bởi nội dung của một word trong bộ nhớ hiển thị. Byte thấp của word chứa mã ASCII của ký tự, byte cao chứa thuộc tính (cho biết ký tự được hiển thị như thế nào: màu của nó, có chớp nháy hay gạch dưới không ? v.v..). Trên thực tế 256 vị trí của một byte đều tương ứng với các ký tự hiển thị (xem phụ lục A). Khái niệm thuộc tính sẽ được trình bày sau.

Bảng 12.3 Một vài vị trí trên màn hình 80×25.

| Vị trí | Dạng thập phân | | Dạng hex | |
|---------------|----------------|------|----------|------|
| | Cột | Dòng | Cột | Dòng |
| Góc trái trên | 0 | 0 | 0 | 0 |
| Góc trái dưới | 0 | 24 | 0 | 18 |
| Góc phải trên | 79 | 0 | 4F | 0 |
| Góc phải dưới | 79 | 24 | 4F | 18 |
| Tâm màn hình | 39 | 12 | 27 | C |

Các trang hiển thị.

Với MDA, bộ nhớ hiển thị chỉ lưu giữ được dữ liệu của một trang màn hình. Trong khi đó các bộ phối ghép đồ họa có thể chứa được dữ liệu của vài trang màn hình. Điều này là do hiển thị đồ họa yêu cầu nhiều bộ nhớ hơn, và như thế khối nhớ của bộ phối ghép đồ họa lớn hơn. Để sử dụng toàn bộ bộ nhớ hiển thị, bộ phối ghép đồ họa chia bộ nhớ hiển thị thành các trang hiển thị. Mỗi trang có thể chứa dữ liệu của một màn hình. Các trang được đánh số, bắt đầu từ 0. Số trang khả dụng phụ thuộc vào bộ phối ghép và chế độ được chọn. Nếu có nhiều hơn một trang khả dụng, chương trình có thể hiển thị trang này trong khi đang cập nhật trang khác.

Bảng 12.4 chỉ ra số trang hiển thị trong chế độ văn bản của MDA, CGA, EGA và VGA. Trong chế độ văn bản 80×25 mỗi trang hiển thị chiếm 4 KB. MDA chỉ có một trang hiển thị-trang 0 với địa chỉ bắt đầu B000:0000h. CGA có 4 trang hiển thị, bắt đầu tại địa chỉ B000:0000h. Trong chế độ văn bản EGA và VGA có thể giả lập MDA hoặc CGA.

Bảng 12.4 Số trang hiển thị trong chế độ văn bản

| Chế độ | Số trang lớn nhất | | |
|--------|-------------------|-----|-----|
| | CGA | EGA | VGA |
| 0-1 | 8 | 8 | 8 |
| 2-3 | 4 | 8 | 8 |
| 7 | NA | 8 | 8 |

Trang hoạt động.

Trang hoạt động là trang hiện thời đang được hiển thị. Đối với chế độ văn bản 80×25 , yêu cầu $80 \times 25 = 2000$ word = 4000 byte (như vậy không sử dụng hết cả 4 KB = 4096 byte của một trang). Bộ điều khiển màn hình hiển thị từ đầu tiên của trang hoạt động ở góc trái trên của màn hình. Từ tiếp theo được hiển thị tại cột 1, dòng 0. Nói chung trang hiển thị được hiển thị dòng nối tiếp dòng, nghĩa là màn hình có thể được xem như hình ảnh của một mảng hai chiều lưu trữ theo thứ tự dòng.

12.3.1 Byte thuộc tính.

Trong một trang hiển thị, byte cao của từ xác định ký tự hiển thị gọi là byte thuộc tính. Nó quy định màu và cường độ của ký tự, màu nền và ký tự có chớp nháy hay gạch dưới không.

Hiển thị 16 màu.

Byte thuộc tính đối với chế độ hiển thị văn bản 16 màu (chế độ 0-3) có dạng được chỉ ra trên hình 12.1. Mỗi vị trí bit là một đặc trưng của thuộc tính. Bit 0-2 xác định màu của ký tự (foreground color), bit 4-6 xác định màu nền tại vị trí ký tự. Ví dụ để hiển thị một ký tự đỏ trên nền xanh da trời ta phải có byte thuộc tính là 0001 0100 = 14h.

Hình 12.1 Byte thuộc tính.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---------|----|---|---|---------|----|---|---|---|
| | BL | R | G | B | IN | R | G | B |
| Màu nền | | | | Màu chữ | | | | |

BL=blinking (nhấp nháy) ; IN=intensity (độ sáng)

R=red; G=green; B=blue

Các màu cơ bản

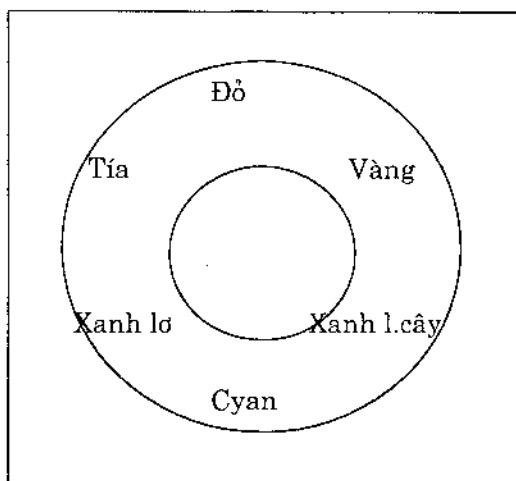
| | I | R | B | G | Màu |
|----|---|---|---|---|---------------|
| 1 | 0 | 0 | 0 | 0 | black |
| 2 | 0 | 0 | 0 | 1 | blue |
| 3 | 0 | 0 | 1 | 0 | green |
| 4 | 0 | 0 | 1 | 1 | cyan |
| 5 | 0 | 1 | 0 | 0 | red |
| 6 | 0 | 1 | 0 | 1 | magenta |
| 7 | 0 | 1 | 1 | 0 | brown |
| 8 | 0 | 1 | 1 | 1 | white |
| 9 | 0 | 0 | 0 | 0 | black |
| 10 | 1 | 0 | 0 | 0 | gray |
| 11 | 1 | 0 | 0 | 1 | light blue |
| 12 | 1 | 0 | 1 | 0 | light green |
| 13 | 1 | 0 | 1 | 1 | light cyan |
| 14 | 1 | 1 | 0 | 0 | light red |
| 15 | 1 | 1 | 0 | 1 | light magenta |
| 16 | 1 | 1 | 1 | 0 | yellow |
| 17 | 1 | 1 | 1 | 1 | intense white |

Các màu sáng

Các màu khác được tạo nên bằng cách kết hợp các màu đỏ, xanh lá cây và xanh da trời. Trong bánh xe cộng màu (hình 12.2), một màu bổ xung có thể được tạo ra bằng cách kết hợp các màu nguyên thuỷ cạnh nhau. Ví dụ màu tía là tổng của màu đỏ và xanh da trời. Để hiển thị một ký tự tía trên nền cyan, byte thuộc tính sẽ là 0011 1010 = 35h.

Nếu như bit độ sáng (bit3) bằng 1, màu nổi sẽ nhạt hơn. Nếu bit chớp nháy (bit 7) bằng 1, ký tự sẽ nhấp nháy. Bảng 12.5 chỉ ra các màu trong 16 màu hiển thị. Tất cả các màu có thể dùng làm màu của ký tự, màu nền chỉ được là một trong các màu cơ bản.

Hình 12.2 Bánh xe cộng màu.



Hiển thị đơn sắc.

Hiển thị đơn sắc chỉ có hai màu: trắng và đen. Đối với màu trắng, tất cả các bit RGB đều bằng 1; với màu đen các bit RGB đều là 0. Trong chế độ thường - trắng trên nền đen, byte thuộc tính là $0000\ 0111 = 7h$. Còn trong chế độ đảo video- đen trên nền trắng, byte thuộc tính trở thành $0111\ 0000 = 70h$.

Cũng giống như hiển thị màu, bit độ sáng có thể dùng để tăng độ sáng ký tự trắng và bit chớp nháy để làm ký tự nhấp nháy. Chỉ riêng đối với màn hình đơn sắc, có hai giá trị thuộc tính tạo nên ký tự gạch dưới. Đó là 01h cho chế độ gạch dưới thường và 09h cho chế độ gạch dưới sáng màu. Bảng 12.6 liệt kê các thuộc tính đơn sắc có thể.

Bảng 12.6 Các thuộc tính đơn sắc.

Byte thuộc tính

| DẠNG NHỊ PHÂN | DẠNG HEX | KẾT QUẢ |
|---------------|----------|--------------------------------|
| 0000 0000 | 00 | đen trên nền đen |
| 0000 0111 | 07 | thường (trắng trên nền đen) |
| 0000 0001 | 01 | gạch dưới thường |
| 0000 1111 | 0F | sáng (trắng sáng trên nền đen) |
| 0000 1001 | 09 | sáng gạch dưới |
| 0111 0000 | 70 | đảo video (đen trên nền trắng) |
| 1000 0111 | 87 | nháy thường |

| | | |
|-----------|----|----------------|
| 1000 1111 | 8F | sáng nháy |
| 1111 1111 | FF | sáng nháy |
| 1111 0000 | F0 | đảo video nháy |

12.3.2 Một ví dụ về trang hiển thị.

Để hiển thị một ký tự với thuộc tính tại một vị trí bất kỳ, ta phải chứa ký tự và thuộc tính vào từ tương ứng trong trang hiển thị hoạt động. Chương trình sau đây sẽ điền đầy màn hình với chữ "A" màu đỏ trên nền xanh da trời.

Program Listing PGM12_1.ASM

```

1: TITLE      PGM12_1:SCREEN_DISPLAY_1
2: .MODEL     SMALL
3: .STACK    100H
4: .CODE
5: MAIN      PROC
6: ;Cho DS trở tới trang hoạt động.
7:         MOV     AX,0B800h ;màu trang hiển thị
8:         MOV     DS,AX
9:         MOV     CX,2000   ;80x25=2000 Words
10:        MOV    DI,0       ;Khởi tạo DI
11: ;điền đầy trang hiển thị
12:FILL_BUF:
13:        MOV    DI,1441h ;A đỏ trên xanh da trời
14:        ADD    DI,2       ;Từ tiếp theo
15:        LOOP   FILL_BUF ;Lặp tiếp cho đến khi xong
16: ; Trở về DOS
17:        MOV    AH, 4CH
18:        INT    21H
19: MAIN      ENDP
20:        END    MAIN

```

Để hiển thị chữ "A" đỏ trên nền xanh da trời tại một vị trí trên màn hình, từ tương ứng của trang hiển thị phải chứa 14h trong byte cao và 41h trong byte thấp.

Chương trình bắt đầu bằng việc khởi tạo DS trở đến đoạn bộ đệm màn hình, tức là B800h đối với bộ phối ghép màu. Biến đếm vòng lặp CX được đặt bằng 2000 là số từ của trang hoạt động, DI khởi tạo bằng 0. Chương trình tiến vào vòng lặp ở dòng 13, nó lần lượt chuyển 1441h vào mỗi từ của bộ đệm màn hình.

Sau khi chạy chương trình, các vị trí trên màn hình giữ nguyên thuộc tính trừ khi một chương trình khác thay đổi chúng hoặc khởi động lại máy.

12.3.3 Ngắt 10H

Chúng ta đã biết rằng có thể hiển thị dữ liệu bằng cách chuyển trực tiếp chúng vào bộ nhớ màn hình, nhưng đây quả là một phương pháp khó khăn để điều khiển màn hình.

Thay vào đó ta có thể sử dụng các phục vụ màn hình của BIOS bằng lệnh INT 10h. Các hàm phục vụ màn hình được chọn bằng cách đưa số hiệu hàm vào thanh ghi AH.

Sau đây chúng tôi sẽ giới thiệu các hàm quan trọng nhất của ngắt 10h trong chế độ văn bản cùng với các ví dụ sử dụng chúng. Các hàm của ngắt này dùng cho chế độ đồ họa sẽ được trình bày trong chương 16. Nếu bạn muốn biết đầy đủ hơn, hãy xem phụ lục C.

Ngắt 10h, hàm 0:
Chọn chế độ hiển thị.

Vào: AH = 0
AL = kiểu (xem bảng 12.2)
Ra: Không có.

Ví dụ 12.1: Thiết lập chế độ văn bản màu cho bộ phôi ghép CGA.

Lời giải:

| | | |
|-----|--------|---------------------------|
| XOR | AH, AH | ;hàm chọn chế độ hiển thị |
| MOV | AL, 3 | ;chế độ văn bản màu 80x25 |
| int | 10h | ;chọn chế độ |

khi BIOS thiết lập chế độ hiển thị, nó đồng thời xoá luôn màn hình.

Ngắt 10h, hàm 1:
Thay đổi kích thước con trỏ màn hình.

Vào: AH = 1

CH = dòng quét đầu
 CL = dòng quét cuối
 Ra: Không có.

Con trỏ màn hình trong chế độ văn bản được hiển thị như một ma trận điểm nhỏ tại một vị trí màn hình (trong chế độ đồ họa không có con trỏ). Ma trận điểm này có 14 dòng (0–13) đối với các bộ phối ghép MDA và EGA và 8 dòng (0–7) với CGA. Thông thường chỉ có các dòng 6 và 7 của con trỏ CGA cũng như các dòng 12,13 của con trỏ MDA và EGA được bật sáng. Để thay đổi kích thước con trỏ, ta đưa số hiệu các dòng bắt đầu và kết thúc được bật sáng tương ứng vào CH và CL.

Ví dụ 12.2 Lập con trỏ với kích thước lớn nhất cho MDA.

Lời giải:

```

MOV AH, 1      ;hàm kích thước con trỏ
MOV CH, 0      ;dòng bắt đầu
MOV CL, 13     ;dòng kết thúc
INT 10h        ;thay đổi kích thước con trỏ
  
```

Ngắt 10h, hàm 2.
Dịch chuyển con trỏ.

Vào: AH = 2
 DH = dòng mới(0–24)
 DL = cột mới (0–79)trong chế độ 80x25,
 (0–39) trong chế độ 40x25)
 BH = số hiệu trang
 Ra: Không có.

Hàm trên cho phép chúng ta di chuyển con trỏ tới mọi chỗ trên màn hình. Trang được tác động đến không nhất thiết phải là trang hiện tại đang được hiển thị.

Ví dụ 12.3: Dịch chuyển con trỏ đến giữa màn hình 80x25 của trang 0.

Lời giải:

```
MOV AH, 2      ;hàm dịch chuyển con trỏ
```

```

XOX  BH,BH      ;trang 0
MOV  DX,0C27h    ;dòng=12,cột=39
INT  10h        ;dịch chuyển con trỏ

```

Ngắt 10h, hàm 3

Lấy vị trí và kích thước con trỏ hiện tại.

Vào: AH = 3
 BH = số hiệu trang
 Ra: DH = dòng
 DL = cột
 CH = dòng quét đầu
 CL = dòng quét cuối

Trong một số chương trình ứng dụng, chẳng hạn muốn di con trỏ lên một dòng, chúng ta cần phải biết được vị trí hiện tại của con trỏ.

Ví dụ 12.4: Di chuyển con trỏ trong trang 0 lên một dòng nếu nó không nằm ở dòng trên cùng.

Lời giải:

```

MOV  AH,3      ;hàm đọc vị trí con trỏ
XOX  BH,BH    ;trang 0
INT  10h      ;DH=dòng, DL=cột
OR   DH,DH    ;dòng trên cùng?
JZ   EXIT      ;đúng, nhảy qua
MOV  AH,2      ;hàm dịch chuyển con trỏ
DEC  DH        ;dòng=dòng-1
INT  10h      ;chuyển con trỏ

```

EXIT:

Ngắt 10h, hàm 5

Chọn trang hoạt động.

Vào: AH = 5

AL = trang hoạt động
 0 – 7 đối với kiểu 0,1
 0 – 3 đối với CGA kiểu 2,3
 0 – 7 đối với EGA, MCGA, VGA kiểu 2,3
 0 – 7 đối với EGA, VGA kiểu 7

Ra: Không có.

Hàm trên đây chọn trang hiển thị hoạt động.

Ví dụ 12.5: Chọn trang 1 cho CGA.

Lời giải:

```
MOV AH, 5      ;hàm chọn trang hiển thị  
MOV AL, 1      ;trang 1  
INT 10h        ;chọn trang
```

Ngắt 10h, hàm 6:

Cuốn màn hình hay cửa sổ lên.

Vào: AH = 6

AL = số dòng cuộn (AL = 0 có nghĩa là cuộn cả màn hình hay cửa sổ)

BH = thuộc tính của các dòng trống

CH, CL = dòng, cột góc trái trên của cửa sổ

DH, DL = dòng, cột góc phải dưới của cửa sổ

Ra: Không có.

Cuốn màn hình lên một dòng có nghĩa là dịch chuyển mỗi dòng hiển thị lên một hàng và điền thêm một dòng trống vào cuối. Dòng trên cùng biến mất khỏi màn hình.

Ta có thể cuốn cả màn hình hay chỉ một vùng hình chữ nhật (cửa sổ) và số dòng được cuốn chứa trong AL. Nếu AL = 0, tất cả các dòng được cuốn lên, đây cũng là một cách để xoá màn hình hay cửa sổ. CH và CL chứa dòng và cột góc trái trên của cửa sổ, DH và DL chứa dòng và cột góc phải dưới. BH chứa thuộc tính của những dòng trống điền vào.

Ví dụ 12.6: Xoá đen màn hình 80x25.

Lời giải:

```
MOV AH, 6      ;hàm cuốn màn hình lên  
XOX AL, AL    ;xoá cả màn hình  
XOX CX, CX    ;góc trái trên(0,0)  
MOV DX, 184Fh  ;góc phải dưới(4Fh, 18h)  
MOV BH, 7      ;thuộc tính video thường  
INT 10h        ;xoá màn hình
```

Ngắt 10h, hàm 7:

Cuốn cửa sổ hoặc màn hình xuống.

Vào: AH = 7

AL = số dòng cuộn (AL = 0 có nghĩa là cuộn cả màn hình hay cửa sổ)

BH = thuộc tính của các dòng trống

CH, CL = dòng, cột góc trái trên của cửa sổ

DH, DL = dòng, cột góc phải dưới của cửa sổ

Ra: Không có.

Cuộn màn hình hay cửa sổ xuống một dòng có nghĩa là mỗi dòng được chuyển xuống một hàng và một dòng trống được diền vào dòng trên cùng, dòng dưới cùng biến mất.

Ngắt 10h, hàm 8:

Đọc ký tự tại vị trí con trỏ.

Vào: AH = 8

BH = số hiệu trang

Ra: AH = thuộc tính của ký tự

AL = mã ASCII của ký tự

Đôi khi trong một số ứng dụng chúng ta cần biết ký tự tại vị trí con trỏ. Số hiệu trang trong BX không nhất thiết phải là trang đang được hiển thị. Sau khi thi hành, AL chứa mã ASCII của ký tự và AH chứa thuộc tính của nó. Lát nữa bạn sẽ thấy một ví dụ sử dụng hàm này. Trước hết chúng ta hãy ngó qua một hàm để viết ký tự.

Ngắt 10h, hàm 9:

Hiển thị ký tự tại vị trí con trỏ với thuộc tính bất kỳ

Vào: AH = 9

BH = số hiệu trang

AL = mã ASCII của ký tự

CX = số lần viết ký tự

BL = thuộc tính của ký tự

Ra: Không có.

Người lập trình có thể sử dụng hàm 9 để xác định thuộc tính của ký tự. CX chứa số lần hiển thị ký tự, bắt đầu từ vị trí con trỏ.

Khác với hàm 2 của ngắt 21h, con trỏ không được chuyển đến vị trí tiếp theo sau khi hiển thị ký tự. Cũng vậy, nếu AL chứa mã ASCII của một ký tự điều khiển, chức năng điều khiển không được thực hiện mà thay vào đó là hiển thị một biểu tượng.

Ví dụ sau đây chỉ ra phương pháp sử dụng các hàm 8, 9 để thay đổi thuộc tính của một ký tự.

Ví dụ 12.7: Đổi thuộc tính của ký tự tại vị trí con trỏ thuộc tính đảo video cho màn hình đơn sắc.

Lời giải:

```
MOV AH, 8      ;đọc ký tự
XOR BH, BH    ;trang 0
INT 10h       ;ký tự trong AL, thuộc tính trong AH
MOV AH, 9      ;hiển thị ký tự với thuộc tính
MOV CX, 1      ;hiển thị 1 ký tự
MOV BL, 70h    ;thuộc tính đảo video.
INT 10h       ;hiển thị ký tự
```

Ngắt 10h, hàm Ah:

Hiển thị ký tự tại vị trí con trỏ với thuộc tính hiện tại

Vào: AH = Ah
 BH = số hiệu trang
 AL = mã ASCII của ký tự
 CX = số lần viết ký tự

Ra: không có

Hàm này giống như hàm 9 ngoại trừ byte thuộc tính không thay đổi, nghĩa là ký tự được hiển thị với thuộc tính hiện tại.

Ngắt 10h, hàm 0Eh:

Hiển thị ký tự và dịch con trỏ.

Vào: AH = 0Eh

AL = mã ASCII của ký tự

BH = số hiệu trang

BL = màu nổi (chỉ trong chế độ đồ họa)

Ra: không có

Hàm này hiển thị ký tự trong AL và chuyển con trỏ đến vị trí tiếp theo trong cùng dòng hoặc đến vị trí bắt đầu dòng tiếp theo nếu nó đã ở cuối dòng. Nếu như con trỏ ở tại vị trí góc phải dưới màn hình thì màn hình được cuộn lên và con trỏ được thiết lập tại đầu dòng tiếp theo. Hàm BIOS này còn được hàm 2, ngắt 21h sử dụng để hiển thị ký tự. Khi nó gặp các ký tự điều khiển như ký tự reo chuông (07h), lùi một ký tự (08h), xuống dòng (0Ah) và trở về đầu dòng (0Dh), các chức năng điều khiển tương ứng sẽ được thực hiện.

Ngắt 10h, hàm Fh:

Lấy chế độ màn hình.

Vào: AH = 0Fh

Ra: AH = số cột màn hình

AL = chế độ hiển thị (xem bảng 12.2)

BH = trang hoạt động

Hàm này có thể được dùng với hàm 5 để chuyển đổi các trang hoạt động.

Ví dụ 12.8: Thay đổi trang hoạt động từ 1 sang 0 hoặc từ 0 sang 1.

Lời giải:

| | | |
|-----|---------|-----------------------|
| MOV | AH, 0Fh | ;lấy chế độ màn hình |
| INT | 10h | ;BH=trang hoạt động |
| MOV | AL, BH | ;chuyển vào AL |
| XOR | AL, 1 | ;bù bit 0 |
| MOV | AH, 5 | ;chọn trang hoạt động |
| INT | 10h | ;chọn trang mới |

12.3.4 Một ví dụ tổng hợp.

Để làm rõ một số hàm của ngắt 10h, chúng ta viết một chương trình thực hiện các công việc sau:

- Thiết lập chế độ hiển thị kiểu 3 (văn bản màu 80x25).
- Xoá cửa sổ có góc trái trên tại cột 26 dòng 8 và góc phải dưới tại cột 52 dòng 16 thành màu đỏ.
- Chuyển con trỏ đến cột 39, dòng 12.
- In ra một chữ A màu cyan, nhấp nháy tại vị trí con trỏ.

Nếu bạn có một màn hình và bộ phôi ghép màu, bạn có thể thấy kết quả khi chạy chương trình trong chương trình nguồn PGM12_2.ASM.

Program Listing PGM12_2.ASM

```
TITLE PGM12_2:SCREEN_DISPLAY_2
;Màn hình đỏ với chữ A màu cyan nhấp nháy
;giữa màn hình
.MODEL      SMALL
.STACK     100H
.CODE

MAIN        PROC
;thiết lập chế độ màn hình
    MOV AH,0          ;hàm chọn chế độ
    MOV AL,3          ;văn bản màu 80x25
    INT 10h           ;chọn chế độ
;xoá cửa sổ với màu đỏ
    MOV AH,6          ;hàm cuộn lên
    MOV CX,081Ah      ;góc trái trên(1Ah,08h)
    MOV DX,1034h       ;góc phải dưới(34h,10h)
    MOV BH,43h         ;ký tự cyan trên nền đỏ
    MOV AL,0          ;cuộn cả màn hình
    INT 10h           ;xoá cửa sổ
;dịch chuyển con trỏ
    MOV AH,2          ;hàm chuyển con trỏ
    MOV DX,OC27h       ;tâm màn hình
    XOR BH,BH          ;trang 0
    INT 10h           ;dịch chuyển con trỏ
;hiển thị kí tự với thuộc tính
    MOV AH,09          ;hàm hiển thị ký tự
```

```

    MOV  BH, 0           ;trang 0
    MOV  BL, 0C3h ;ký tự cyan nhấp nháy trên nền đỏ
    MOV  CX, 1           ;hiển thị một ký tự
    MOV  AL, "A"         ;chữ "A"
    INT  10h             ;hiển thị ký tự
;trở về DOS
    MOV  AH, 4Ch
    INT  21h
MAIN  ENDP
ENDL   MAIN

```

12.4 Bàn phím.

IBM PC sử dụng vài loại bàn phím khác nhau. Bàn phím chuẩn có 83 phím. Hiện nay có nhiều máy tính sử dụng bàn phím mở rộng với 101 phím. Nói chung chúng ta có thể phân chung chúng thành 3 nhóm chính:

1. Các phím ASCII. Đây là các phím tương ứng với các ký tự hiển thị ASCII và các ký tự điều khiển. Chúng bao gồm các chữ cái, chữ số, dấu chấm câu, dấu số học và một số biểu tượng đặc biệt khác, và các phím điều khiển: Esc (Escape), Enter (carriage return), Backspace và Tab.
2. Các phím dịch: phím dịch trái, dịch phải, Caps Lock, Ctrl, Alt, Num Lock và Scroll Lock. Các phím này thường được dùng kết hợp với các phím khác.
3. Các phím chức năng: F1 - F10 (F1 - F12 đối với bàn phím mở rộng), các phím mũi tên, Home, Pgup, Pgdn, End, Ins và Del. Ta gọi là các phím chức năng bởi vì chúng thường được dùng trong chương trình để thực hiện các chức năng đặc biệt.

Mã scan.

Mỗi phím của bàn phím được gán với một số duy nhất gọi là mã scan của phím đó. Khi một phím được nhấn, vi mạch bàn phím gửi mã scan tương ứng đến máy tính. Giá trị của các mã scan bắt đầu từ 1. Bảng 12.7 chỉ ra mã scan của các phím dịch và các phím chức năng. Một bảng đầy đủ các mã scan của bàn phím 101 phím được trình bày trong phụ lục H.

Bạn có thể tự hỏi máy tính nhận biết một tổ hợp các phím như thế nào, ví như tổ hợp Ctrl-Alt-Del để khởi động lại máy. Chắc chắn là phải có một cách nào đó để máy tính nhận biết được là một phím được nhấn nhưng chưa nhả ra.

Để bảo hiệu một phím được nhả ra, vì mạch bàn phím gửi một mã khác được tính toán từ mã scan bằng cách bật bit msb lên 1 (bản thân mã scan có thể xem là mã bắt đầu) gọi là mã kết thúc. Ví dụ mã bắt đầu của phím Esc là 1h, mã kết thúc của nó là 81h.

Máy tính không lưu trữ thông tin của một phím được nhấn nhưng chưa nhả trừ phím chức năng Ins và các phím dịch. Các thông tin được ghi lại trên các bit riêng lẻ gọi là *cờ bàn phím*, ở byte có địa chỉ 0040:0017.

Một chương trình có thể gọi một chương trình BIOS để đọc cờ này

Bảng 12.7 Mã scan của các phím dịch và các phím chức năng.

| DẠNG HEX | DẠNG THẬP PHÂN | PHÍM |
|----------|----------------|-------------|
| 1D | 29 | Ctrl |
| 2A | 42 | Left Shift |
| 38 | 56 | Alt |
| 3A | 58 | Caps Lock |
| 3B-44 | 59-68 | F1-F10 |
| 45 | 69 | Num Lock |
| 46 | 70 | Scroll Lock |
| 47 | 71 | Home |
| 48 | 72 | Up arrow |
| 49 | 73 | Pgup |
| 4B | 75 | Left arrow |
| 4C | 76 | Keypad5 |
| 4D | 77 | Right arrow |
| 4F | 79 | End |
| 50 | 80 | Dn arrow |
| 51 | 81 | Pgdn |
| 52 | 82 | Ins |
| 53 | 83 | Del |

Bộ đệm bàn phím.

Để tránh việc chương trình bỏ qua các phím nhấn, Máy tính sử dụng một khối nhớ 15 từ gọi là bộ đệm bàn phím để chứa các phím đã đánh vào nhưng chưa được đọc bởi chương trình. Mỗi phím nhấn được chứa trong một từ với byte cao chứa mã scan, byte thấp chứa mã ASCII nếu là phím ASCII hay 0 nếu là phím chức năng. Một phím dịch không được lưu lại trong bộ đệm. Khi một phím Shift trái hay phải, phím Ctrl hay Alt được nhấn, một vài phím khác có thể tạo nên một mã scan phím tổ hợp và được đưa vào bộ đệm (xem phụ lục H).

Nội dung của bộ đệm được lấy ra khi chương trình yêu cầu số liệu vào từ bàn phím. Các giá trị phím nhấn được chuyển cho chương trình có thứ tự giống hệt với khi nhập các phím, có nghĩa là theo kiểu "xếp hàng". Nếu có yêu cầu nhập phím mà bộ đệm rỗng, hệ thống sẽ đợi đến khi một phím được nhấn. Khi bộ đệm đã đầy, nếu người sử dụng nhấn một phím, máy tính sẽ phát một tiếng bíp.

Thao tác bàn phím.

Để tổng kết các kiến thức mới học, chúng ta hãy xem điều gì xảy ra khi bạn nhấn một phím và chương trình đang được thi hành đọc nó.

1. Bàn phím gửi một yêu cầu (ngắt 9) đến máy tính.
2. Phục vụ ngắt 9 nhận mã scan từ cổng vào/ra bàn phím và chứa nó vào một từ trong bộ đệm (byte cao = mã scan, byte thấp = mã ASCII với phím ASCII, 0 với phím chức năng).
3. Chương trình có thể dùng ngắt 21h, hàm 1 để đọc mã ASCII. Điều này cũng tạo nên sự hiển thị của mã ASCII trên màn hình.

Trong phần tiếp theo chúng tôi sẽ chỉ ra chương trình có thể xử lý các số liệu vào từ bàn phím bằng cách sử dụng ngắt 16h như thế nào. Để nhận được cả mã scan lẫn mã ASCII, một chương trình có thể truy nhập trực tiếp bộ đệm bàn phím hay sử dụng phục vụ ngắt 16h của BIOS.

INT 16H.

Ngắt 16h của BIOS cung cấp các phục vụ bàn phím. Cũng giống như ngắt 10h, một chương trình có thể yêu cầu một phục vụ bằng cách đưa số hiệu hàm vào thanh ghi AH trước khi gọi INT 16H. Sau đây chúng ta chỉ xem xét hàm 0.

Ngắt 16h, hàm 0

Đọc phím nhấn.

Vào: AH = 0

Ra:

AL = mã ASCII nếu một phím ASCII được nhấn
= 0 với các phím chức năng
AH = mã scan của phím nhấn

Hàm này chuyển giá trị phím nhấn đầu tiên trong bộ đệm bàn phím vào AX. Nếu bộ đệm trống, máy tính đợi người sử dụng nhấn một phím. Các phím ASCII không được hiển thị trên màn hình.

Hàm trên đây cung cấp một phương pháp kiểm tra phím chức năng. Nếu AL = 0 có nghĩa là một phím chức năng được nhấn, chương trình có thể lấy mã scan trong AH để nhận biết đó là phím chức năng nào.

Ví dụ 12.9: Dịch chuyển con trỏ đến góc trái trên nếu phím F1 được nhấn, góc trái dưới nếu là một phím chức năng khác. Chương trình sẽ bỏ qua nếu bạn nhấn một phím ký tự.

Lời giải:

```
MOV AH, 0          ;hàm đọc phím nhấn
INT 16h           ;AL=mã ASCII hoặc 0, AH=mã scan
OR AL, AL         ;AL=0(hàm chức năng)?
JNE EXIT          ;không, phím ký tự
CMP AH, 3Bh        ;mã scan F1?
JE F1             ;đúng, đến F1
;phím chức năng khác
MOV DX, 184Fh      ;góc phải dưới
JMP EXECUTE       ;đến dịch chuyển con trỏ
F1:
XOR DX, DX        ;góc trái trên
EXECUTE:
MOV AH, 2          ;hàm chuyển con trỏ
XOR BH, BH         ;trang 0
INT 10h            ;dịch chuyển con trỏ
EXIT:
```

12.5 Chương trình soạn thảo màn hình.

Để chỉ ra cách lập trình với các phím chức năng, chương trình của chúng ta sẽ thực hiện một vài thao tác cơ bản của một trình xử lý văn bản. Đầu tiên chương trình xoá màn hình và đưa con trỏ đến góc trái trên của màn hình. Sau đó nó cho phép người sử dụng đánh vào các dòng văn bản, thao tác với một số phím chức năng và kết thúc khi nhấn phím Esc.

Thuật giải soạn thảo màn hình:

```
Xoá màn hình.  
Chuyển con trỏ tới góc trái trên.  
Nhận một phím.  
WHILE không phải phím Esc DO  
    IF phím chức năng  
        THEN  
            thực hiện chức năng.  
        ELSE /* phím ký tự */  
            hiển thị ký tự  
    END_IF  
    Nhận một phím  
END WHILE
```

Chương trình có thể kiểm tra phím Esc bằng cách so sánh mã ASCII với 1Bh. Thủ tục DO_FUNTION viết cho các phím mũi tên, nó cho chúng ta thấy một phương pháp lập trình với các phím chức năng. Các thao tác của thủ tục như sau:

- ⇒ Up arrow (mũi tên lên): dịch con trỏ lên một dòng hoặc cuộn màn hình lên một dòng nếu con trỏ ở dòng trên cùng.
- ⇒ Down arrow (mũi tên xuống): dịch con trỏ xuống một dòng hoặc cuộn màn hình xuống một dòng nếu con trỏ ở dòng dưới cùng.
- ⇒ Right arrow (mũi tên sang phải): dịch con trỏ sang phải một cột hoặc chuyển xuống đầu dòng tiếp theo nếu con trỏ ở lề bên phải. Nếu con trỏ ở góc phải dưới, cuộn màn hình lên một dòng.
- ⇒ Left arrow (mũi tên sang trái): dịch con trỏ sang trái một cột hoặc chuyển xuống cuối dòng tiếp theo nếu con trỏ ở lề bên trái. Nếu con trỏ ở góc trái trên, cuộn màn hình xuống một dòng.

Thuật giải DO_FUNTION:

Lấy vị trí con trỏ;
Kiểm tra mã scan của phím nhấn cuối cùng;
CASE mã scan OF

Up arrow:

IF con trỏ ở dòng trên cùng
THEN
 cuốn màn hình xuống
ELSE
 chuyển con trỏ lên
END_IF

Down arrow:

IF con trỏ ở dòng dưới cùng
THEN
 cuốn màn hình lên
ELSE
 chuyển con trỏ xuống
END_IF

Left arrow:

IF con trỏ không ở đầu dòng /* cột 0 */
THEN
 dịch con trỏ sang trái
ELSE /* con trỏ tại vị trí bắt đầu dòng */
 IF con trỏ ở dòng 0
 THEN
 cuốn màn hình xuống
 ELSE
 chuyển con trỏ đến cuối
 dòng tiếp theo
 END_IF
END_IF

Right arrow:

IF con trỏ không ở cuối dòng /* cột 79 */
THEN
 dịch con trỏ sang phải
ELSE /* con trỏ tại vị trí cuối dòng */
 IF con trỏ ở dòng cuối cùng /* 24 */
 THEN
 cuốn màn hình lên
 ELSE
 chuyển con trỏ đến đầu
 dòng sát trên

```

        END_IF
    END_IF
END_CASE

```

Và sau đây là chương trình:

Program Listing PGM12_3.ASM

```

0: TITLE      PGM12_3:SCREEN EDITOR
1: .MODEL     SMALL
2: .STACK     100H
3: .CODE
4: MAIN       PROC
5: ;thiết lập chế độ và xoá màn hình
6:     MOV AH,0          ;hàm thiết lập chế độ
7:     MOV AL,3          ;văn bản màu 80x25
8:     INT 10h           ;thiết lập chế độ
9: ;chuyển con trỏ đến góc trái trên
10:    MOV AH,2          ;hàm chuyển con trỏ
11:    XOR DX,DX         ;vị trí(0,0)
12:    MOV BH,0          ;trang 0
13:    INT 10h           ;chuyển con trỏ
14: ;nhận phím nhấn
15:    MOV AH,0          ;hàm nhận số liệu vào từ bàn phím
16:    INT 16h           ;AH=mã scan;AL=mã ASCII
17: WHILE_:
18:     CMP AL,1Bh        ;Esc?
19:     JE END_WHILE     ;đúng, thoát ra
20: ;if phím chức năng
21:     CMP AL,0          ;phím chức năng?
22:     JNE ELSE_          ;không, phím ký tự
23: ;then
24:     CALL DO_FUNCTION;thi hành chức năng
25:     JMP NEXT_KEY      ;nhận phím tiếp theo
26: ;ELSE_
27:     MOV AH,2          ;hàm hiển thị ký tự
28:     MOV DL,AL          ;lấy kí tự
29:     INT 21h           ;hiển thị ký tự
30: NEXT_KEY:
31:     MOV AH,0          ;hàm nhận phím nhấn
32:     INT 16h           ;AH=mã scan,AL=mã ASCII
33:     JMP WHILE_
34: End WHILE
35: ;trở về DOS

```

```

36: MOV      AH, 4Ch
37: INT     21h
38: MAIN    ENDP
39:
40: DO_FUNCTION PROC
41: ;thao tác với các phím mũi tên
42: ;vào: AH=mã scan
43: ;ra: không có
44: PUSH BX
45: PUSH CX
46: PUSH DX
47: PUSH AX           ;cất mã scan
48: ;Xác định vị trí con trỏ
49: MOV      AH, 3       ;lấy vị trí con trỏ
50: MOV      BH, 0       ;trang 0
51: INT     10h          ;DH=dòng, DL=cột
52: POP      AX          ;khôi phục mã scan
53: ;CASE mã scan OF
54: CMP      AH, 72        ;mũi tên lên?
55: JE      CURSOR_UP      ;đúng, thi hành
56: CMP      AH, 75        ;mũi tên trái
57: JE      CURSOR_LEFT;đúng, thi hành
58: CMP      AH, 77        ;mũi tên phải
59: JE      CURSOR_RIGHT     ;đúng, thi hành
60: CMP      AH, 80        ;mũi tên xuống
61: JE      CURSOR_DOWN;đúng, thi hành
62: JMP      EXIT          ;phím chức năng khác
63: CURSOR_UP:
64: CMP      DH, 0          ;dòng 0?
65: JE      SCROLL_DOWN;đúng, cuộn màn hình xuống
66: DEC      DH            ;không, dòng=dòng-1
67: JMP      EXECUTE        ;gọi hàm dịch chuyển con trỏ
68: CURSOR_DOWN:
69: CMP      DH, 24         ;dòng cuối cùng
70: JE      SCROLL_UP        ;đúng, cuộn màn hình lên
71: INC      DH            ;không, dòng=dòng+1
72: JMP      EXECUTE
73: CURSOR_LEFT:
74: CMP      DL, 0          ;cột 0?
75: JNE      GO_LEFT        ;không, dịch trái
76: CMP      DH, 0          ;dòng 0?
77: JE      SCROLL_DOWN;đúng, cuộn màn hình xuống
78: DEC      DH            ;dòng=dòng-1
79: MOV      DL, 79          ;cột cuối cùng

```

```

80:    JMP EXECUTE
81: CURSOR_RIGHT:
82:    CMP DL, 79           ;cột cuối cùng?
83:    JNE GO_RIGHT        ;không, dịch phải
84:    CMP DH, 24           ;dòng cuối cùng
85:    JE SCROLL_UP        ;đúng, cuộn màn hình lên
86:    INC DH              ;dòng=dòng+1
87:    MOV DL, 0             ;cột 0
88:    JMP EXECUTE
89: GO_LEFT:
90:    DEC DL              ;cột=cột-1
91:    JMP EXECUTE
92: GO_RIGHT:
93:    INC DL              ;cột=cột+1
94:    JMP EXECUTE
95: SCROLL_DOWN:
96:    MOV AL, 1             ;cuốn 1 dòng
97:    XOR CX, CX           ;góc trái trên = (0,0)
98:    MOV DH, 24             ;hàng cuối cùng
99:    MOV DL, 79             ;cột cuối cùng
100:   MOV BH, 07            ;thuộc tính video thường
101:   MOV AH, 7              ;hàm cuộn xuống
102:   INT 10h               ;cuộn xuống 1 dòng
103:   JMP EXIT              ;thoát khỏi thủ tục
104: SCROLL_UP:
105:   MOV AL, 1             ;cuốn 1 dòng
106:   XOR CX, CX           ;góc trái trên = (0,0)
107:   MOV DX, 184Fh          ;góc phải dưới (4Fh, 18h)
108:   MOV BH, 07            ;thuộc tính video thường
109:   MOV AH, 6              ;hàm cuộn lên
110:   INT 10h               ;cuộn lên 1 dòng
111:   JMP EXIT              ;thoát khỏi thủ tục
112: EXECUTE:
113:   MOV AH, 2              ;hàm dịch chuyển con trỏ
114:   INT 10h               ;dịch chuyển con trỏ
115: EXIT:
116:   POP DX
117:   POP CX
118:   POP BX
119:   RET
120: DO_FUNCTION ENDP
121: END MAIN

```

Chương trình bắt đầu với việc thiết lập chế độ màn hình văn bản màu 80x25 (mode 3). Thao tác này cũng đồng thời xoá màn hình. Sau khi chuyển con trỏ đến góc trái trên, chương trình nhập một phím và đi vào vòng lặp WHILE ở dòng 17. AH chứa mã scan, AL chứa mã ASCII nếu là phím ký tự và 0 nếu là phím chức năng. Chương trình sẽ kết thúc nếu bạn bấm phím Esc (AL=1Bh), ngược lại chương trình sẽ kiểm tra xem có phải phím chức năng hay không (AL=0). Nếu là phím chức năng, thủ tục DO_FUNTION được gọi, nếu không thì đó phải là một phím ký tự và được hiển thị bởi hàm 2 của ngắt 21h. Hàm này tự động dịch con trỏ sau khi hiển thị ký tự. Phím tiếp theo được nhập tại cuối vòng lặp WHILE (dòng 30).

Chương trình tiến vào thủ tục DO_FUNTION với mã scan của phím nhấn cuối cùng trong AH. Mã này được cất trong ngăn xếp (dòng 47) khi thủ tục xác định vị trí con trỏ (dòng 49 - 51). Sau khi phục hồi mã scan trong AH (dòng 52), thủ tục kiểm tra xem nó có phải là phím mũi tên hay không (dòng 54 - 61). Nếu không, thủ tục sẽ kết thúc.

Nếu AH chứa mã scan của một phím mũi tên, thủ tục nhảy đến khối lệnh dịch chuyển con trỏ thích hợp. DH và DL tương ứng chứa dòng và cột của vị trí con trỏ.

DH và DL được tính toán lại nếu như con trỏ không ở tại lề của màn hình (dòng 0 hay 24, cột 0 hay 79). Để dịch lên hay xuống, ta chỉ việc tương ứng giảm hay tăng số hiệu dòng trong DH. Cũng như vậy, dịch trái hay phải con trỏ tương ứng với việc giảm hay tăng số hiệu cột trong DL. Sau khi tính toán lại DH và DL, chương trình nhảy đến dòng 112, tại đây hàm 2 của ngắt 10h có nhiệm vụ chuyển con trỏ đến vị trí mới.

Đối với phím mũi tên lên, nếu con trỏ ở tại dòng 0, thủ tục sẽ nhảy đến khối lệnh SCROLL_DOWN ở dòng 65 để cuộn màn hình xuống. Tương tự với phím mũi tên xuống, nếu con trỏ ở tại dòng 24, dòng 72 thủ tục sẽ nhảy đến khối lệnh SCROLL_UP để cuộn màn hình lên một dòng.

Đối với phím mũi tên trái, nếu con trỏ ở tại góc trái trên (0, 0), thủ tục sẽ nhảy đến SCROLL_DOWN (dòng 77). Nếu con trỏ ở lề trái nhưng không phải dòng 0, ta muốn dịch chuyển con trỏ đến cuối dòng trước đó. Để làm điều này, số hiệu dòng trong DH được giảm 1, DL gán bằng 79 và thủ tục nhảy đến dòng 112 để dịch chuyển con trỏ.

Tương tự đối với phím mũi tên phải. Nếu con trỏ ở tại góc phải dưới, thủ tục sẽ nhảy đến SCROLL_UP (dòng 85). Nếu con trỏ ở lề phải nhưng không phải dòng 24, ta muốn dịch chuyển con trỏ đến đầu dòng tiếp theo. Để làm điều này, số hiệu dòng trong DH được tăng 1, DL gán bằng 0 và thủ tục nhảy đến dòng 112 để dịch chuyển con trỏ.

Sau khi biên dịch và liên kết file PGM12_3.ASM bạn có thể chạy chương trình. Khi này bạn có thể thấy ngay các hạn chế của nó, ví dụ các dòng văn bản mất đi khi bị cuốn khỏi màn hình. Chương trình cho phép viết đè nhưng không xoá hay chèn văn bản được.

TỔNG KẾT.

- ◆ Bộ phôi ghép màn hình gồm có bộ nhớ và bộ điều khiển màn hình để chuyển đổi dữ liệu thành hình ảnh trên màn hình. Các bộ phôi ghép là MDA, CGA, EGA, MCGA và VGA. Chúng khác nhau ở độ phân giải và số màu hiển thị.
- ◆ Có hai loại chế độ hiển thị: chế độ văn bản và chế độ đồ họa. Trong chế độ văn bản, các ký tự được hiển thị tại mỗi vị trí trên màn hình. Chế độ đồ họa hiển thị theo các điểm ảnh.
- ◆ Trong chế độ văn bản, mỗi vị trí màn hình được xác định bởi toạ độ (dòng, cột) của nó. Một ký tự và thuộc tính của nó có thể được hiển thị tại một vị trí.
- ◆ Trong chế độ văn bản 80x25, Bộ nhớ của bộ phôi ghép màn hình được chia thành các khối 4 KB gọi là trang hiển thị. Số lượng trang hiển thị phụ thuộc vào từng bộ phôi ghép cụ thể. Tại một thời điểm, màn hình chỉ có thể hiển thị một trang, trang đang được hiển thị gọi là trang hoạt động.
- ◆ Sự hiển thị trên mỗi vị trí màn hình được xác định bởi một từ của trang hiển thị. Byte thấp của từ chứa mã ASCII của ký tự, byte cao chứa thuộc tính của nó.
- ◆ Byte thuộc tính xác định màu nổi (màu của ký tự) và màu nền tại mỗi vị trí màn hình. Các thuộc tính khác là nhấp nháy và gạch dưới (chỉ với MDA).
- ◆ Trong hiển thị đơn sắc, màu nổi và màu nền chỉ có thể là đen (các bit RGB đều bằng 0) hay trắng (các bit RGB bằng 1). Thuộc tính video thường là 07h, đảo video là 70h.
- ◆ Thường trình phục vụ ngắt 10h của BIOS thực hiện các thao tác màn hình. Số hiệu hàm phục vụ màn hình chứa trong AH.
- ◆ Hàm 0 ngắt 16h là hàm của BIOS để đọc phím nhấn. AH nhận mã scan và AL nhận mã ASCII nếu là phím ký tự. Với phím chức năng, AH chứa mã scan và AL=0.
- ◆ Một chương trình có thể sử dụng ngắt 16h hay ngắt 10h để lập trình các phím chức năng điều khiển màn hình hiển thị.

Các thuật ngữ tiếng Anh.

| | |
|----------------------------|---|
| active display page | Trang hiện tại đang được hiển thị trên màn hình. |
| attribute | Con số xác định thuộc tính ký tự. |
| attribute byte | Byte cao của từ xác định ký tự hiển thị, nó chưa thuộc tính của ký tự. |
| break code | Con số cho biết thời điểm một phím được nhả ra, tạo nên bằng cách đưa 1 vào msb của mã scan của phím. |
| CGA | Color Graphic Adapter. |
| character cell | Mã trận điểm dùng để tạo dạng ký tự trên màn hình. |
| display memory | Khối nhớ của bộ phối ghép màn hình. |
| display page | Một phần của bộ nhớ hiển thị chứa dữ liệu của một màn hình. |
| EGA | Enhanced Graphic Adapter. |
| function keys | Các phím không tương ứng với các ký tự ASCII hay các phím shift. |
| graphic mode | Chế độ có thể hiển thị các hình vẽ. |
| gray scale | Các mức độ sáng khác nhau trong hiển thị đơn sắc. |
| keyboard buffer | Khối nhớ 15 từ dùng để lưu nhận phím nhấn. |
| make code | Giống như mã scan. |
| MCGA | Multi-colorgraphic Array. |
| MDA | Monochrom Display Adapter. |
| mode number | Con số dùng để chọn chế độ hiển thị văn bản hay đồ họa. |
| normal video | Ký tự trắng trên nền đen. |
| resolution | Số các điểm mà bộ phối ghép màn hình có thể hiển thị. |
| reverse video | Ký tự đen trên nền trắng. |
| scan code | Con số dùng để định nghĩa một phím. |
| text mode | Chế độ chỉ hiển thị các ký tự |
| VGA | Video Graphics Array |
| video adapter | Ví mạch điều khiển hiển thị màn hình. |

| | |
|-------------------------|--|
| Video buffer | Bộ nhớ chứa dữ liệu hiển thị trên màn hình giống như bộ nhớ hiển thị |
| Video controller | Khối điều khiển của bộ phối ghép màn hình. |

Bài tập.

1. Để hiểu rõ hơn về bộ đệm màn hình, bạn hãy dùng DEBUG thực hiện các công việc sau đây:
 - a. Nếu máy của bạn có bộ phối ghép đơn sắc, dùng lệnh R đưa B000h vào DS; nếu là bộ phối ghép màu, đưa B800 vào DS.
 - b. Bây giờ ta có thể đưa số liệu một cách trực tiếp vào bộ đệm màn hình và xem luôn kết quả. Để thực hiện điều này bạn dùng lệnh E đưa dữ liệu vào bắt đầu tại offset 0. Ví dụ để hiển thị một chữ A đảo màu video nhấp nháy ở góc trái trên của màn hình, bạn đưa 41h vào byte 0 và F0h vào byte 1. Bạn tiếp tục đưa các giá trị ký tự : thuộc tính khác nhau vào các từ 2, 3 v.v... và quan sát sự thay đổi của dòng đầu tiên trên màn hình.
2. Viết các đoạn lệnh thực hiện các công việc sau đây (giả thiết làm việc với trang 0, hiển thị đơn sắc 80x25). Các phần của bài tập độc lập với nhau.
 - a. Chuyển con trỏ đến góc phải dưới của màn hình.
 - b. Xác định vị trí con trỏ và chuyển nó đến cuối dòng hiện tại.
 - c. Xác định vị trí con trỏ và chuyển nó đến dòng trên cùng trong cột hiện tại.
 - d. Chuyển con trỏ sang trái một vị trí nếu nó không ở vị trí đầu dòng.
 - e. Xoá dòng chứa con trỏ thành màu trắng.
 - f. Cuốn cột chứa con trỏ xuống một dòng (thuộc tính video thường).
 - g. Hiển thị 5 chữ A đảo màu video nhấp nháy bắt đầu từ góc trái trên của màn hình.

Các bài tập lập trình.

4. Viết các chương trình để:

- a. Xoá màn hình, tạo kích thước to nhất cho con trỏ và chuyển nó đến góc trái trên.
- b. Lập trình các phím chức năng sau:
 - Home: Chuyển con trỏ đến góc trái trên.
 - End: Chuyển con trỏ đến góc trái dưới.
 - Pgdn: Chuyển con trỏ đến góc phải dưới.

Esc: Kết thúc chương trình.
Các phím khác: Không làm gì cả.

5. Viết chương trình thực hiện:

- Xoá đen màn hình, chuyển con trỏ đến góc trái trên.
- Để người sử dụng đánh vào một tên.
- Xoá dòng vừa nhập và hiển thị tên dọc theo cột 40, bắt đầu từ dòng trên cùng. Bạn sử dụng chế độ hiển thị 80x25. Đối với MDA, hiển thị tên với thuộc tính đảo video. Với hiển thị màu, hiển thị nó với chữ xanh trên nền tía.

6. Viết một chương trình thực hiện các công việc sau đây:

- Xoá màn hình, chuyển con trỏ đến dòng 12, cột 0.
- Nếu người sử dụng đánh vào một ký tự, hiển thị ký tự tại vị trí con trỏ. Con trỏ không được dịch di sau khi hiển thị.
- Lập trình các phím chức năng sau:
 - Mũi tên phải: Nếu con trỏ không ở tại lề phải, chương trình chuyển con trỏ và ký tự sang phải một vị trí. Một khoảng trắng xuất hiện tại vị trí trước đó của con trỏ.
 - Mũi tên trái: Nếu con trỏ không ở tại lề trái, chương trình chuyển con trỏ và ký tự sang trái một vị trí. Một khoảng trắng xuất hiện tại vị trí trước đó của con trỏ.
 - Esc: Kết thúc chương trình.
 - Các phím chức năng khác không ảnh hưởng gì đến chương trình.

7. Viết một chương trình soạn thảo một dòng màn hình thực hiện các công việc sau:

- Xoá màn hình, định vị con trỏ tại đầu dòng 12.
- Để người sử dụng đánh vào các ký tự. Con trỏ dịch di sau khi hiển thị ký tự nếu nó không ở tại lề phải của màn hình.
- Phím mũi tên trái dịch con trỏ sang trái ngoại trừ nó đã ở vị trí đầu dòng. Nếu con trỏ không ở cuối dòng, phím mũi tên phải sẽ dịch nó sang phải một vị trí. Các phím mũi tên khác không có tác động.
- Phím Ins chuyển con trỏ và các ký tự bên phải nó (trong dòng chứa con trỏ) sang phải một vị trí. Một khoảng trống xuất hiện tại vị trí con trỏ trước đó. Ký tự cuối cùng trên dòng con trỏ bị đẩy khỏi màn hình.
- Phím del chuyển các ký tự bên phải con trỏ sang trái một vị trí và một khoảng trống được diền vào cuối dòng.
- Phím Esc kết thúc chương trình.

Chương 13

MACRO

Tổng quan

Trong các chương trước chúng tôi đã chỉ ra rằng bằng các thủ tục chúng ta có thể đơn giản hóa việc lập trình, trong chương này chúng tôi sẽ trình bày về một cấu trúc chương trình khá giống với các thủ tục, đó là macro. Giống như thủ tục, tên một macro đại diện cho một nhóm các lệnh. Mỗi khi cần các lệnh đó trong chương trình chúng ta chỉ việc đưa vào tên macro. Tuy nhiên các thủ tục và macro được thực hiện theo các phương pháp khác nhau. Một thủ tục được gọi vào thời điểm thực hiện chương trình, điều khiển được chuyển cho thủ tục để rồi sau đó được trả lại cho chương trình sau khi thực hiện xong các lệnh của nó. Macro được tham chiếu vào thời điểm hợp dịch chương trình. Trình biên dịch sao chép các lệnh của macro vào trong chương trình tại vị trí có lời gọi macro. Trong khi chương trình được thực hiện không xảy ra sự chuyển điều khiển.

Macro đặc biệt có ích khi cần thực hiện các công việc một cách thường xuyên. Chẳng hạn chúng ta có thể viết các macro để khởi tạo các thanh ghi ES và DS, in một chuỗi ký tự, kết thúc một chương trình v.v... Chúng ta cũng có thể dùng các macro để khắc phục hạn chế của các lệnh sẵn có; chẳng hạn toán hạng của lệnh MUL không thể là một hằng số nhưng chúng ta có thể viết một macro thực hiện phép nhân không có hạn chế trên.

13.1 Định nghĩa và tham chiếu đến các macro

Macro là một khối văn bản có tên. Khi trình biên dịch gặp tên đó trong khi biên dịch, nó sẽ chèn khối văn bản vào chương trình. Khối văn bản nói trên có thể bao gồm các lệnh, các toán tử giả, lời bình hay sự tham chiếu đến các macro khác.

Cú pháp của khai báo macro như sau:

macro_name MACRO d1, d2, ... dn

statements

ENDM

Trong đó macro_name là tên do người dùng định nghĩa cho macro. Các toán tử giả MACRO và ENDM chỉ ra sự bắt đầu và kết thúc của việc khai báo macro; d1, d2,...dn là danh sách tùy chọn các biến giả sử dụng trong macro.

Một trong các ứng dụng của macro là tạo ra các lệnh mới. Chẳng hạn như chúng ta đã biết các toán hạng của lệnh MOV không thể cùng là hai biến kiểu WORD, nhưng chúng ta có thể khắc phục hạn chế này bằng cách định nghĩa một macro chuyển một word vào một word

Ví dụ 13.1 Định nghĩa macro chuyển một word vào một word

| | | |
|------|-------|--------------|
| MOVW | MACRO | WORD1, WORD2 |
| | PUSH | WORD2 |
| | POP | WORD1 |
| | ENDM | |

Trong ví dụ này tên của macro là MOVW và WORD1, WORD2 là các biến giả.

Để sử dụng macro trong chương trình chúng ta tham chiếu nó theo cú pháp như sau:

macro_name a1, a2, ... an

Trong đó a1,a2,,an là danh sách các biến thực. Khi trình biên dịch gấp tên macro, nó sẽ “mở rộng macro” (expands macro) có nghĩa là nó sao toàn bộ các lệnh của macro vào trong chương trình tại vị trí tham chiếu đến macro giống hệt như là người sử dụng đã đánh vào các lệnh đó. Trong khi sao chép các lệnh, trình biên dịch thay thế các biến giả di bằng các biến thực tương ứng ai và tạo nên mã máy cho mọi lệnh.

Khai báo macro phải được thực hiện trước khi có thể tham chiếu đến nó trong chương trình nguồn. Để bảo đảm trình tự này việc định nghĩa các macro thường được đặt ở đầu chương trình. Cũng có thể tạo lên một thư viện các macro để sử dụng trong mọi chương trình và chúng ta sẽ đề cập đến vấn đề này trong chương sau.

Ví dụ 13.2 Tham chiếu macro MOVW để chuyển B vào A trong đó A và B là các biến word

Trả lời:

MOVW A, B

Để mở rộng macro này trình biên dịch sẽ chép các lệnh của macro vào chương trình tại vị trí của lời gọi, thay WORD1, WORD2 bằng A và B một cách tương ứng tại các vị trí xuất hiện của chúng. Kết quả nhận được như sau:

```
PUSH A  
POP B
```

Khi mở rộng macro trình biên dịch đơn giản thay thế các biến thực vào các biến giả tương ứng. Ví dụ các lời gọi macro MOVW dưới đây

```
MOVW A, DX  
và MOVW A+2, B
```

sẽ khiến cho trình biên dịch chèn những đoạn mã sau vào chương trình:

```
PUSH DX      và      PUSH B  
POP A          POP A+2
```

Các tham chiếu macro không hợp lệ

Có một số hạn chế về các biến cho một macro. Chẳng hạn các biến trong macro MOVW phải là từ nhớ hay thanh ghi 16 bit. Sự tham chiếu macro

```
MOVW AX, 1ABC
```

tạo ra mã lệnh như sau:

```
PUSH 1ABC  
POP AX
```

và bởi vì lệnh PUSH trực tiếp dữ liệu là không hợp lệ (đối với hệ vi xử lý 8086 và 8088), việc tham chiếu sẽ tạo ra lỗi hợp dịch. Một phương pháp để ngăn chặn tình trạng này là đưa các lời bình vào trong macro, chẳng hạn:

```
MOVW MACRO WORD1, WORD2
```

; các biến phải là từ nhớ hay thanh ghi 16 bit

```
PUSH WORD2  
POP WORD1  
ENDM
```

Phục hồi các thanh ghi

Một kinh nghiệm lập trình tốt là các thủ tục phải phục hồi những thanh ghi mà nó sử dụng trừ những thanh ghi chứa các kết quả từ thủ tục. Điều đó cũng đúng cho các macro. Để làm ví dụ hãy xem xét macro hoán đổi hai từ nhớ dưới đây. Vì macro này sử dụng AX để thực hiện việc hoán đổi, thanh ghi này phải được phục hồi lại khi kết thúc.

```
EXCH    MACRO WORD1,WORD2
        PUSH   AX
        MOV    AX,WORD1
        XCHG   AX,WORD2
        MOV    WORD1,AX
        POP    AX
ENDM
```

Mở rộng macro trong file .LST

File .LST là một trong các file được tạo ra khi chương trình được biên dịch(xem phụ lục D). File này chỉ ra mã lệnh hợp ngữ và mã máy tương ứng, các địa chỉ của biến và các thông tin khác về chương trình. File .LST cũng chỉ ra các macro được mở rộng ra sao. Để diễn tả điều đó chương trình sau đây chứa macro MOVW và 2 lần tham chiếu đến nó:

Chương trình nguồn PGM13_1.ASM

```
TITLE  PGM13_1:    MACRO DEMO
.MODEL  SMALL
MOVW    MACRO WORD1,WORD2
        PUSH   WORD2
        POP    WORD1
ENDM
.STACK 100H
.DATA
A       DW      1,2
B       DW      3
.CODE
MAIN   PROG
        MOV    AX,@DATA
        MOV    DS,AX
```

```

        MOVW    A, DX
        MOVW    A+2, B
; DOS  exit
        MOV     AH, 4CH
        INT     21H
MAIN    ENDP
END     MAIN

```

Hình 13.1 biểu diễn file PGM13_1.LST. Trong file này trình biên dịch in ra sự tham chiếu macro theo sau là sự mở rộng của nó (in bằng chữ đậm). Chữ số 1 xuất hiện ở mỗi dòng trong mở rộng macro mang ý nghĩa là các macro này được tham chiếu ở mức cao nhất, có nghĩa là bằng chính chương trình. Chúng tôi sẽ chỉ ra ở chương sau rằng một macro có thể tham chiếu đến một macro khác.

```

Microsoft (r)  macro assembler version 5.10
1/18/92      00:03:08
PGM13_1: MACRO DEMO          Page      1-1
                    TITLE PGM13_1: MACRO DEMO
                    .MODEL      SMALL
                    MOVW        MACRO WORD1,WORD2
                                PUSH WORD2
                                POP  WORD1
                    ENDM
                    .STACK      100H
                    .DATA
0000      0001 0002 A   DW   1,2
0004      0003           B   DW   3
                    .CODE
0000          MAIN      PROC
0000      B8__R           MOV      AX, @DATA
0003      8E D8            MOV      DS, AX
                                MOVW    A, DX
0005      52                 1     PUSH   DX
0006      8F 06 0000 R     1     POP    A
                                MOVW    A+2, B
000A      FF 36 0004 R     1     PUSH   B
000E      8F 06 0002 R     1     POP    A+2
                                ; DOS EXIT
0012      B4 4C             MOV    AH, 4CH
0014      CD 21             INT    21H
0016          MAIN      ENDP
                                END     MAIN

```

MICROSOFT (R) Macro Assembler version 5.10

```

1/18/92          00:03:08
PGM13_1: MACRO DEMO      SYMBOLS-1
Macros:
Name           Lines
MOVW, . . . . . 2
Segments and Groups:
Name       Length     Align     Combine     Class
DGROUP . . . . . GROUP
 _DATA . . . . . 0006 WORD PUBLIC   'DATA'
STACK . . . . . 0100 PARA STACK   'SATACK'
 _TEXT . . . . . 0016 WORD PUBLIC   'CODE'
Symbols:
Name       Type     Value     Attr
A . . . . . L WORD 0000 _DATA
B . . . . . L WORD 0004 _DATA
MAIN. . . . . N PROC 0000 _TEXT
Length = 0016
@CODE . . . . . . . TEXT _TEXT
@CODE SIZE. . . . . . . TEXT 0
@CPU . . . . . . . . . . . TEXT 0101h
@DATA SIZE. . . . . . . . . . . TEXT 0
@FILENAME . . . . . . . . . . . TEXT PGM13_1
@VERSION . . . . . . . . . . . TEXT 510
21    Source     Lines
25    Total Lines
21    Symbols
47930 + 4220033 Bytes symbols space free
0    Warning    Errors
0    Severe    Errors

```

Hình 13_1 PGM13_1.LST

Các phần chọn của file .LST

Ba dẫn hướng biên dịch quyết định mở rộng macro sẽ xuất hiện như thế nào trong file .LST. Các dẫn hướng biên dịch này thuộc về macro theo ngay sau nó trong chương trình.

- ⇒ Sau dẫn hướng biên dịch .SALL(suppress all), mã hợp ngữ trong mở rộng macro sẽ không được in ra. Bạn sử dụng tùy chọn này trong trường hợp có các macro lớn hoặc macro được tham chiếu nhiều lần trong chương trình.

- ⇒ Sau dẫn hướng biên dịch .XALL, chỉ những dòng trong chương trình nguồn tạo ra mã lệnh mới được in ra. Chẳng hạn các dòng bình luận sẽ không được in ra. Đây là tùy chọn ngầm định.
- ⇒ Sau dẫn hướng biên dịch .LALL(List all), tất cả các dòng trong chương trình nguồn được in ra, trừ những dòng được bắt đầu bằng 2 dấu chấm phẩy(;;).
- ⇒ Cần lưu ý các bạn rằng các tùy chọn này không ảnh hưởng tới mã máy được tạo ra trong phần tham chiếu macro mà chỉ ảnh hưởng tới kiểu thể hiện mở rộng macro trong file .LST.

Ví dụ 13.3 Giả sử macro MOVW được viết lại như sau:

```

MOVW      MACRO WORD1,WORD2
;Chuyển từ nguồn tới đích
;;Có sử dụng ngăn xếp
    PUSH WORD2
    POP  WORD1
ENDM

```

Hãy cho biết sự tham chiếu các macro sau đây sẽ thể hiện như thế nào trong file .LST:

```

.XALL
    MOVW DS,CS
.LALL
    MOVW P,Q
.SALL
    MOV  AX,[ SI]

```

Trả lời:

```

.XALL
    MOVW DS,CS
    PUSH CS
    POP  DS
.LALL
    MOVW P,Q
    ;Chuyển từ nguồn tới đích
    PUSH Q
    POP  P
.SALL
    MOVW AX,[ SI]

```

Xác định các lỗi biên dịch

Khi trình biên dịch phát hiện lỗi trong mở rộng macro, nó sẽ chỉ ra lỗi tại vị trí mà macro được tham chiếu trong chương trình. Tuy nhiên rất có thể là lỗi đó lại xuất hiện trong bản thân macro. Để xác định xem thực ra lỗi nằm ở đâu, các bạn cần rà soát phần mở rộng macro trong file .LST. File .LST đặc biệt có ích khi bạn có một macro mà nó tham chiếu đến một macro khác (xem phần sau).

13.2 Các nhãn cục bộ (Local labels).

Một macro với vòng lặp hay cấu trúc rẽ nhánh phải chứa một hoặc nhiều nhãn. Nếu một macro như vậy được tham chiếu nhiều hơn một lần trong chương trình, nhãn sẽ bị lặp và gây ra lỗi biên dịch. Chúng ta có thể tránh được vấn đề này bằng cách sử dụng các nhãn cục bộ trong macro. Để khai báo chúng, ta sử dụng toán tử giả LOCAL với cú pháp như sau:

```
LOCAL      List_of_labels
```

Trong đó List_of_labels là danh sách các nhãn, đặt cách nhau dấu phẩy. Mỗi khi macro được mở rộng, trình biên dịch sẽ gán những ký hiệu khác nhau vào các nhãn trong danh sách. Dẫn hướng biên dịch LOCAL phải được đặt ngay sau phát biểu MACRO, thậm chí ngay cả một lời bình cũng không được đặt trước nó.

Ví dụ 13.4: Viết một macro đưa từ lớn hơn trong hai từ vào AX

Trả lời:

```
GET_BIG MACRO WORD1,WORD2
         LOCAL EXIT
         MOV AX,WORD1
         CMP AX,WORD2
         JG EXIT
         MOV AX,WORD2
EXIT:
ENDM
```

Bây giờ hãy giả sử rằng FIRST, SECOND và THIRD là các biến word. Sự tham chiếu macro

```
GET_BIG     FIRST,SECOND
```

Được mở rộng như sau:

```
MOV AX, FIRST
CMP AX, SECOND
JG ??0000
MOV AX, SECOND
??0000:
```

Và lời gọi tiếp theo:

```
GET_BIG SECOND, THIRD
```

Được mở rộng như sau:

```
MOV AX, SECOND
CMP AX, THIRD
JG ??0001
??0001:
```

Sự tham chiếu liên tiếp macro này hay đến các macro khác với nhau sẽ khiến trình biên dịch chèn các nhãn ??0002, ??0003 và cứ như vậy vào trong chương trình các nhãn này là duy nhất và sẽ không bị xung đột với các nhãn mà người dùng chọn lựa.

13.3 Các macro tham chiếu đến macro khác.

Một macro có khả năng tham chiếu đến một macro khác. Ví dụ giả sử chúng ta có 2 macro thực hiện việc lưu và phục hồi các thanh ghi:

```
SAVE_REGS MACRO R1, R2, R3
    PUSH R1
    PUSH R2
    PUSH R3
    ENDM
RESTORE_REGS MACRO S1, S2, S3
    POP S1
    POP S2
    POP S3
    ENDM
```

Các macro này được tham chiếu bằng một macro khác trong ví dụ dưới đây:

Ví dụ 13.5: Viết một macro để copy chuỗi. Sử dụng các macro SAVE_REGS và RESTORE_REGS.

Trả lời:

```
COPY      MACRO SOURCE, DESTINATION, LENGTH
          SAVE_REGS CX, SI, DI
          LEA   SI, SOURCE
          LEA   DI, DESTINATION
          CLD
          MOV   CX, LENGTH
          REP   MOVSB
          RESTORE_REGS DI, SI, CX
          ENDM
```

Khi trình biên dịch gấp lời gọi macro

```
COPY      STRING1, STRING2, 15
```

nó sẽ sao chép đoạn mã sau vào chương trình:

```
PUSH  CX
PUSH  SI
PUSH  DI
LEA   SI, STRING1
LEA   DI, STRING2
CLD
MOV   CX, 15
REP   MOVSB
POP   DI
POP   SI
POP   CX
```

Chú ý một macro có thể tham chiếu đến chính nó, những macro như vậy được gọi là các macro đệ qui và chúng ta không đề cập tới chúng trong cuốn sách này.

13.4 Thư viện các macro

Các macro mà chương trình tham chiếu đến có thể chứa trong một file riêng biệt. Nhờ đó có thể tạo ra một file thư viện các macro có ích. Chẳng hạn, giả sử tên của file là macros và được đặt trong đĩa A thì khi trình biên dịch gấp dẫn hướng:

```
INCLUDE    A:MACROS
```

trong chương trình, nó sẽ sao toàn bộ các định nghĩa macro từ file MACROS vào chương trình tại vị trí của thông báo INCLUDE (dẫn hướng biên dịch INCLUDE được trình bày trong phần 9.5). Thông báo INCLUDE có thể xuất hiện tại bất cứ vị trí nào trong chương trình miễn là nó đứng trước mọi tham chiếu đến các macro trong file thư viện đó.

Toán tử điều kiện IF1

Khi một thư viện macro được ghép vào trong chương trình, tất cả các định nghĩa macro của nó sẽ xuất hiện trong file .LST, thậm chí ngay cả khi nó không được tham chiếu đến trong chương trình. Để tránh điều này chúng ta có thể chèn thêm vào câu lệnh sau:

```
IF1
INCLUDE      MACROS
ENDIF
```

Trong đó IF1 và ENDIF là các toán tử giả. Dẫn hướng IF1 khiến cho trình biên dịch truy nhập file MACROS trong lần biên dịch thứ nhất khi các macros được mở rộng nhưng không truy nhập đến nó trong lần biên dịch thứ 2 khi file .LST được tạo ra.

Chú ý các toán tử giả điều kiện được trình bày trong phần 13.6

Ví dụ về các macro tiện ích

Dưới đây là các ví dụ về macro mà sẽ rất có ích khi đặt nó trong một file thư viện.

Ví dụ 13.6: Viết một macro để trở về DOS.
Trả lời:

```
DOS_RTN MACRO
    MOV AH, 4CH
    INT 21H
ENDM
Cách dùng Macro
DOS_RTN
```

Ví dụ 13.7: Viết một macro thực hiện việc trả về đầu dòng và xuống dòng.

Trả lời:

```
NEW_LINE MACRO
    MOV AH, 02
    MOV DL, 0DH
    INT 21H
    MOV DL, 0AH
    INT 21H
ENDM
```

Cách dùng Macro

```
NEW_LINE
```

Ví dụ tiếp theo đây là một trong những macro khá thú vị.

Ví dụ 13.8: Viết một macro hiển thị một chuỗi ký tự trong đó chuỗi ký tự là một biến macro.

Trả lời:

```
DISP_STR MACRO STRING
    LOCAL START, MSG
;cắt các thanh ghi
    PUSH AX
    PUSH DX
    PUSH DS
    JMP START
    MSG DB STRING, '$'
START:
    MOV AX, CS
    MOV DS, AX ;cho DS hướng tới đoạn dữ liệu
    MOV AH, 09
    LEA DX, MSG
    INT 21H
;Phục hồi thanh ghi
    POP DS
    POP DX
    POP AX
ENDM
```

Chúng ta có thể tham chiếu đến macro này như sau:

```
DISP_STR 'THIS IS A STRING'
```

Khi macro được tham chiếu, biến chuỗi sẽ thay thế cho biến giả STRING. Vì chuỗi được lưu trong đoạn mã lên CS phải được chuyển vào DS, việc này cần phải dùng tới 2 lệnh vì không thể chuyển trực tiếp giữa 2 thanh ghi đoạn.

Ghép thư viện macro

Giả sử các macro trên đây đã có trong file MACROS chứa trong đĩa. Chúng ta sẽ sử dụng nó trong chương trình dưới đây. Chương trình này hiển thị một thông báo chuyển con trỏ xuống dòng dưới và hiển thị một thông báo khác.

Chương trình nguồn PGM13_2.ASM

```
TITLE PGM13_2: MACRO DEMO
.MODEL   SMALL
.STACK   100H
. IF1
INCLUDE  MACROS
ENDIF
.CODE
MAIN    PROC
        DISP_STR 'THIS IS THE FIRST LINE'
        NEW_LINE
        DISP_STR 'AND THIS IS THE SECOND LINE'
        DOS_RTN
MAIN    ENDP
END MAIN
```

Ví dụ thực hiện:

```
C>PGM13_2
THIS IS THE FIRST LINE
AND THIS IS THE SECOND LINE
```

Sự mở rộng macro được thể hiện trong file PGM13_2.LST(hình 13.2). Để giành chỗ phần mã máy không được in ra.

```
TITLE      PGM13_2:MACRO DEMO
.MODEL     SMALL
.STACK     100H
.CODE
MAIN      PROC
DISP_STR  'this is the first line'
1          PUSH AX
1          PUSH DX
```

```

1           PUSH   DS
1           JMP    ??0000
1     ??0001      DB      'THIS IS THE FIRST LINE','$'
1     ??0000:
1           MOV    AX,CS
1           MOV    DS,AX ;DX hướng tới đoạn mã
1           MOV    AH,09
1           LEA    DX,??0001
1           POP    DS
1           POP    DX
1           POP    AX

NEW_LINE
1           MOV    AH,02
1           MOV    DL,0DH
1           INT    21H
1           MOV    DL,0AH
1           INT    21H

DISP_STR 'AND THIS IS THE SECOND LINE'
1           PUSH   AX
1           PUSH   DX
1           PUSH   DS
1           JMP    ??0002
1     ??0003  DB  'AND THIS IS THE SECOND LINE','$'
1     ??0002:
1           MOV    AX,CS
1           MOV    DS,AX ;DS hướng tới đoạn mã
1           MOV    AH,09
1           LEA    DX,??0003
1           POP    DS
1           POP    DX
1           POP    AX

DOS_RTN
1           MOV    AH,4CH
1           INT    21H

MAIN    ENDP
END     MAIN

```

Hình 13.2 PGM13_2.LST

13.5 Các macro lặp

Macro **REPT** có thể sử dụng để lặp lại một khối các câu lệnh. Cú pháp của nó như sau:

```
REPT expression  
statements  
ENDM
```

Khi trình biên dịch gặp macro này, các câu lệnh được lặp lại một số lần cho bởi giá trị của biểu thức expression. Một macro **REPT** có thể được tham chiếu bằng cách đặt nó trong chương trình tại những nơi mà các câu lệnh của macro sẽ được lặp lại. Ví dụ để khai báo một mảng A các biến kiểu word với 5 giá trị 0 chúng ta có thể viết như sau trong đoạn dữ liệu:

```
A    LABEL WORD  
REPT      5  
DW      0  
ENDM
```

Chú ý: toán tử giả **LABEL** được trình bày trong phần 10.2.3. Trình biên dịch sẽ mở rộng macro như sau:

```
A    DW      0  
DW      0  
DW      0  
DW      0  
DW      0
```

Tất nhiên đây chỉ là một ví dụ tầm thường vì để làm công việc trên chúng ta chỉ cần viết:

```
A    DW      5 DUP (0)
```

Một cách khác để tham chiếu macro **REPT** là đặt nó trong một macro khác và tham chiếu đến macro đó.

Ví dụ 13.9: Viết một macro khởi tạo một khối nhớ với N số nguyên đầu tiên. Sau đó hãy tham chiếu nó để khởi tạo một mảng với 100 số nguyên đầu tiên.

Trả lời:

```
BLOCK    MACRO N  
          K=1  
          REPT  N  
          DW    K  
          K=K+1  
          ENDM  
          ENDM
```

Chú ý: Trong macro này chúng ta đã sử dụng toán tử = (equal). Giống như EQU, nó có thể sử dụng để để gán tên cho một hằng số. Biểu thức bên phải dấu bằng phải cho kết quả là một số. Khác với EQU hằng số được định nghĩa bằng dấu = có thể định nghĩa lại, chẳng hạn, K=K + 1. Cần nhớ rằng tất cả những công việc này đều diễn ra trong thời gian biên dịch chứ không phải trong thời gian thực hiện.

Để khai báo một mảng A các word và khởi tạo cho nó 100 số nguyên đầu tiên chúng ta có thể viết như sau trong đoạn dữ liệu:

```
A    LABEL WORD  
BLOCK 100
```

Việc tham chiếu macro BLOCK khởi tạo K bằng 1 và các lệnh bên trong macro REPT được biên dịch 100 lần. Trong lần đầu tiên lệnh DW 1 được tạo ra và K được tăng lên thành 2; trong lần thứ 2 DW 2 được tạo ra và K nhận giá trị 3,...cho đến lần thứ 100, DW 100 được tạo ra và K = 101. Kết quả cuối cùng như sau:

```
A    DW      1  
      DW      2  
      .  
      .  
      .  
      DW      100
```

Ví dụ 13.10: Viết một macro khởi tạo một mảng n word với các giá trị $1!$, $2!$, ..., $n!$ và chỉ ra cách tham chiếu đến nó.

Trả lời:

```
FACTORIALS      MACRO N  
                  M=1  
                  FAC=1  
REPT   N  
      DW      FAC  
      M=M+1  
      FAC=M* FAC  
ENDM  
ENDM
```

Để khai báo một mảng word B gồm 8 giao thừa đầu tiên chúng ta có thể viết như sau trong đoạn dữ liệu :

```
B    LABEL      WORD  
FACTORIALS 8
```

Vì $8! = 40320$ là giai thừa lớn nhất mà một từ nhớ 16 bit có thể chứa được sẽ không tính táo nếu tham chiếu macro này với giá trị N lớn hơn 8. Mở rộng macro có dạng như sau:

```
B    DW    1  
      DW    2  
      DW    6  
      DW    24  
      DW    120  
      DW    720  
      DW    5040  
      DW    40320
```

Macro IRP

Một macro lặp khác đó là macro **IRP** (indefinite macro). Nó có dạng sau:

```
IRP d, <a1,a2,...an>  
      statements  
ENDM
```

Chú ý: Cặp ngoặc nhọn trong khai báo trên là một phần của cú pháp. Khi được mở rộng, macro này làm cho các câu lệnh trong nó được biên dịch n lần; trong lần mở rộng thứ i mỗi khi có tham số d xuất hiện thì nó được thay thế bằng ai.

Ví dụ 13.11: Viết một macro cất và phục hồi một số tùy ý các thanh ghi.

Trả lời:

```
SAVE_REGS      MACRO_REGS  
              IRP D,<REGS>  
              PUSH D  
ENDM  
ENDM  
RESTORE_REGS   MACRO_REGS  
              IRP D,<REGS>  
              POP D  
ENDM  
ENDM
```

Để cất các thanh ghi AX, BX, CX, DX chúng ta có thể viết:

```
SAVE_REGS <AX, BX, CX, DX>
```

nó có khai triển như sau:

```
PUSH      AX  
PUSH      BX  
PUSH      CX  
PUSH      DX
```

Và để phục hồi các thanh ghi đó chúng ta chỉ cần viết:

```
RESTORE_REGS <DX,CX,BX,AX>
```

13.6 Một macro xuất

Để sử dụng những cấu trúc macro đã học đến nay chúng ta hãy viết một macro HEX_OUT để hiển thị nội dung của một word dưới dạng 4 chữ số hex. Thuật giải đưa ra một số hex (như đã trình bày trong chương 7) như sau:

Thuật giải đưa ra số hex (chứa trong BX)

```
1: FOR 4 lần DO  
2: chuyển BH vào DL  
3: dịch DL 4 lần sang phải  
4: IF     DL<10  
5:     THEN  
6:         đổi nội dung của DL thành ký tự trong  
         khoảng từ '0' đến '9'  
7:     ELSE  
8:         đổi nội dung của DL thành ký tự trong  
         khoảng từ 'A' đến 'F'  
9: END_IF  
10: đưa ra ký tự  
11: quay BX 4 lần về bên trái  
12: END_FOR
```

Phần dưới đây trình bày macro HEX_OUT và chương trình để kiểm tra nó. HEX_OUT tham chiếu đến 4 macro khác :(1) SAVE_REGS và (2) RESTORE_REGS đã nó đến trong Ví dụ 13.11;

(3) CONVERT_TO_CHAR là macro đổi nội dung của một byte ra một ký tự biểu diễn chữ số hex (dòng 4-9 trong thuật giải); và (4) DISP_CHAR là macro dùng để hiển thị một ký tự (dòng 10 trong thuật giải).

Chương trình nguồn PGM13_3.ASM

```
0: TITLE    PGM13_3:    HEX OUTPUT MACRO DEMO  
1: .MODEL   SMALL  
2:
```

```

3: SAVE_REGS      MACRO REGS
4:             IRP D,<REGS>
5:             PUSH D
6:             ENDM
7:             ENDM
8:
9: RESTORE_RES   MACRO REGS
10:            IRP D,<REGS>
11:            POP D
12:            ENDM
13:            ENDM
14:
15: CONVERT_TO_CHAR MACRO BYT
16:             LOCAL ELSE_, EXIT
17: ;đổi nội dung của BYT ra ký tự biểu diễn chữ số hex
18: ;IF
19:             CMP BYT,9 ; <=9?
20:             JNLE ELSE_ ;không, >= Ah
21: ;THEN
22:             OR BYT,30H;đổi ra ký tự chữ số
23:             JMP EXIT
24: ELSE_:
25:             ADD BYT,37H;đổi ra ký tự
26: EXIT:
27:             ENDM
28:
29: DISP_CHAR      MACRO BYT
30: ; hiển thị nội dung của BYT
31:             PUSH AX
32:             MOV AH,2
33:             MOV DL,BYT
34:             INT 21H
35:             POP AX
36:             ENDM
37:
38: HEX_OUTPUT     MACRO WRD
39: ;đổi nội dung của BYT thành chữ số hex
40:             SAVE_REGS <BX,CX,DX>
41:             MOV BX,WRD
42:             MOV CL,4;bộ đếm số lần dịch và quay
43:             REPT 4
44:             MOV DL,BH
45:             SHR DL,CL;dịch phải 4 lần

```

```

46:      CONVERT_TO_CHAR DL ;đổi thành ký tự biểu
                      ;diễn chữ số hex
47:      DISP_CHAR DL ;hiển thị nó
48:          ROL BX,CL ;quay trái 4 lần
49:          ENDM
50:      RESTORE_REGS <DX,CX,BX>
51:          ENDM
52:
53: .STACK
54: .CODE
55: ;chương trình để kiểm tra macro trên
56: MAIN PROC
57:     MOV AX,1AF4h ;dữ liệu để thử
58:     HEX_OUT AX
59: ;DOS exit
60:     MOV AH,4CH
61:     INT 21H
62: MAIN ENDP
63: END MAIN

```

Ví dụ thực hiện

C>PGM13_3

1AF4

Để mã hoá cho vòng lặp FOR trong thuật giải đưa ra số hex, HEX_OUT đã sử dụng

REPT..ENDM (dòng 43-49). Cách làm này chủ yếu phục vụ cho mục đích minh họa vì nó làm cho phần mã máy của mở rộng macro dài ra nhưng nó cũng có ưu điểm là không dùng đến thanh ghi CL và do đó CL được sử dụng tự do làm bộ đếm cho các lệnh quay và dịch.

Tại dòng 46 macro CONVERT_TO_CHAR được tham chiếu để đổi nội dung của DL thành ký tự biểu diễn chữ số hex. Macro này có 2 nhãn cục bộ được khai báo ở dòng 16. Tại dòng 47 macro DISP_CHR được tham chiếu để hiển thị nội dung của DL.

13.7 Các toán tử điều kiện

Các toán tử điều kiện có thể được sử dụng để biên dịch một số lệnh nào đó trong khi loại trừ ra một số lệnh khác. Chúng có thể được dùng ở mọi nơi trong chương trình hợp ngữ nhưng chủ yếu là bên trong các macro. Dưới đây là các dạng cơ bản:

```
conditional          và           conditional  
statements          statements1  
ENDIF               ELSE  
                     statements2  
                     ENDIF
```

Trong dạng thứ nhất nếu điều kiện (conditional) là đúng thì các câu lệnh(statements) được thực hiện và ngược lại sẽ không có câu lệnh nào được thực hiện nếu điều kiện sai.

Trong dạng thứ 2 nếu điều kiện là đúng thì các câu lệnh 1(statements1) được biên dịch trong trường hợp ngược lại các câu lệnh 2 (statements2) được biên dịch (ELSE và ENDIF là các toán tử giả).

| DẠNG | ĐIỀU KIẾN ĐÚNG |
|---------------------|--|
| IF exp | Hằng biểu thức khác 0 |
| IFE exp | Hằng biểu thức bằng 0 |
| IFB <arg> | Không có biến arg (blank), cần phải có cặp ngoặc nhọn |
| IFNB <arg> | Có biến arg (not blank) |
| IFDEF sym | Ký hiệu sym được định nghĩa trong chương trình (hay được khai báo EXTRN). Chú ý dẫn hướng biên dịch EXTRN được trình bày trong chương 14 |
| IFNDEF sym | Sym không được định nghĩa hay khai báo EXTRN |
| IFIDN<str1>,<str2> | Chuỗi str1 và str2 giống nhau. Cần phải có các cặp ngoặc nhọn |
| IFDIF <str1>,<str2> | Các chuỗi không giống nhau |
| IF1 | Trình biên dịch tiến hành lần duyệt thứ nhất |
| IF2 | Trình biên dịch tiến hành lần duyệt thứ 2 |

Bảng 13-1 Các toán tử điều kiện

Bảng 13.1 đưa ra dạng của những toán tử điều kiện tiện dụng nhất và điều kiện để chúng nhận giá trị đúng (TRUE).

Trong phần 13.4 chúng ta đã sử dụng toán tử điều kiện IF1 để ghép một thư viện macro vào trong chương trình. Các ví dụ tiếp theo sẽ chỉ ra các toán tử điều kiện khác được sử dụng như thế nào.

Macro ử dụng toán tử điều kiện IF

Ví dụ 13.12: Viết một macro định nghĩa một khối N từ nhớ chứa K số nguyên đầu tiên theo sau là N-K số 0. Sử dụng nó để khởi tạo một mảng từ nhớ với 10 giá trị 1, 2, 3, 4, 5, 0, 0, 0, 0 và 0.

Trả lời:

```
BLOCK      MACRO N, K
          I=1
          REPT  N
          IF    K+1-I
          DW    I
          I=  I+1
          ELSE
          DW    0
          ENDIF
          ENDM
          ENDM
```

Và macro này được tham chiếu để định nghĩa ma trận A như sau:

```
A      LABEL WORD
BLOCK 10, 5
```

Mở rộng macro khởi tạo giá trị 10 cho N, 5 cho K và 1 cho I sau đó biên dịch những câu lệnh bên trong REPT 10 lần. Sau 5 lần đầu tiên, các lệnh DW 1...DW 5 được tạo ra và I = 6. Sau đó bởi vì $K+1-I=5+1-6=0$, các câu lệnh sau ELSE - chính là các lệnh DW 0 được biên dịch. Kết quả cuối cùng tương tự như sau:

```
A      DW    1, 2, 3, 4, 5, 0, 0, 0, 0, 0
```

Macro ử dụng toán tử điều kiện IFNB

Hãy nhớ lại chương 11, bài tập 9 trong đó hàm 0AH ngắn 21h lưu một chuỗi mà người sử dụng đánh từ bàn phím vào một mảng byte. Địa chỉ offset của mảng này chứa trong DX. Byte đầu tiên của mảng chứa số lượng cực đại các ký

tự có thể đánh vào. DOS sẽ điền vào các byte sau đó những giá trị phù hợp tùy thuộc vào số ký tự đọc được.

Ví dụ 13.13: Viết một macro READ MACRO BUF,LEN có thể sử dụng hàm 0Ah ngắt 21h để đọc một chuỗi có độ dài LEN vào vùng đệm BUF(nếu có cả hai biến số) hay hàm con 01 của ngắt để đọc một ký tự vào trong AL(nếu không có cả hai biến số).

Trả lời:

```
READ      MACRO BUF,MAXCHARS
;BUF=địa chỉ vùng đệm chuỗi
;MAXCHARS=số cực đại các ký tự đọc vào
    IFNB <BUF>
        IFNB <MAXCHARS>
            MOV AH,0AH      ;hàm đọc chuỗi
            LEA DX,BUF      ;dx chứa địa chỉ chuỗi
            MOV BUF,MAXCHARS ;byte đầu tiên chứa
                               ;kích thước mảng
            INT 21H          ;đọc chuỗi
        ENDIF
        ELSE
            MOV AH,01        ;hàm đọc ký tự
            INT 21H          ;đọc ký tự
        ENDIF
    ENDM
```

Khi macro trên được tham chiếu bằng câu lệnh:

```
READ      MSG,10
```

thì do có cả 2 biến số mã lệnh sau sẽ được biên dịch:

```
MOV      AH,0AH
LEA      DX,MSG
MOV      MSG,10
INT      21H
```

Chuỗi MSG phải là một mảng đã khai báo có kích thước ít nhất là 13 byte, (1 byte cho số lượng cực đại các ký tự có thể đọc vào, 1 byte cho số ký tự thực tế đọc được, 10 byte cho các ký tự và 1 byte cho ký tự CR). Nếu macro được tham chiếu như sau:

READ

thì do không có cả 2 biến số, đoạn mã sau ELSE :

```
MOV      AH, 01  
INT      21H
```

sẽ được biên dịch. Trong trường hợp macro được tham chiếu không đúng, chẳng hạn chỉ có một biến số, thì không có mã lệnh nào được biên dịch.

Dẫn hướng biên dịch .ERR

Vì các macro có thể được tham chiếu trong rất nhiều tình huống khác nhau nên có thể xảy ra trường hợp chúng được tham chiếu không đúng. Dẫn hướng .ERR cho phép trình biên dịch thông báo với người sử dụng về vấn đề này. Nếu trình biên dịch gặp dẫn hướng này nó sẽ hiển thị thông báo "forced error" chỉ ra một lỗi hợp dịch nghiêm trọng.

Ví dụ 13.14: Viết một chương trình chứa macro hiển thị một ký tự. Macro phải tạo ra một thông báo lỗi hợp dịch nếu không có tham số của nó.

Trả lời:

Chương trình nguồn PGM13_4.ASM

```
TITLE    PGM13_4:           .ERR DEMO  
.MODEL   SMALL  
.STACK   100H  
DISP_CHAR MACRO CHAR  
    IFNB <CHAR>  
    MOV  AH, 02  
    MOV  DL, CHAR  
    INT  21H  
    ELSE  
    .ERR  
    ENDIF  
    ENDM  
.CODE  
MAIN    PROC  
    DISP_CHAR 'A'  ;lời gọi hợp lệ  
    DISP_CHAR        ;lời gọi không hợp lệ  
    ;DOS exit
```

```
        MOV     AH, 4CH
        INT     21H
MAIN      ENDP
        END     MAIN
```

```
C>MASM PGM13_4;
Microsoft (R) Macro Assembler Version 5.10.
Copyright (C) Microsoft Corp    1981, 1988. All rights reserved.

PGM13_4.ASM(15):      error A2089:Forced error
50050 + 418683 Bytes symbol space free
0 Warning Errors
1 Severe Errors
```

13.8 Các macro và thủ tục

Macro và thủ tục giống nhau về mặt chúng cùng được viết để thực hiện một công việc cho một chương trình. Nhưng trong một số trường hợp rất khó chọn lựa cấu trúc nào là tốt hơn trong tình huống đã cho. Dưới đây là một vài nhận xét:

- ◆ **Về thời gian biên dịch:**

Một chương trình chứa các macro thường cần nhiều thời gian biên dịch hơn là một chương trình tương tự chứa các thủ tục bởi vì nó cần thời gian để mở rộng macro. Điều này đặc biệt đúng trong trường hợp chương trình tham chiếu đến một thư viện các macro.

- ◆ **Về thời gian thực hiện:**

Mã lệnh tạo lên bởi mở rộng macro nói chung thực hiện nhanh hơn một lời gọi thủ tục, vì trong trường hợp sau còn liên quan đến việc cất giữ địa chỉ trả về, chuyển điều khiển cho thủ tục, chuyển dữ liệu vào trong thủ tục và cuối cùng là trả về từ thủ tục.

- ◆ **Về kích thước chương trình:**

Một chương trình với các macro nói chung lớn hơn một chương trình tương tự với các thủ tục, bởi vì mỗi lần tham chiếu đến macro đều khiến cho có một khối mã lệnh riêng biệt được chép vào trong chương trình. Trong khi đó thủ tục chỉ được mã hóa đúng một lần.

- ◆ **Về các mặt khác:**

Macro đặc biệt phù hợp cho các công việc nhỏ, xuất hiện thường xuyên, bởi việc sử dụng tự do các macro đó tạo ra một chương trình nguồn giống như chương

trình bằng ngôn ngữ bậc cao. Tuy nhiên các công việc lớn thì thường tốt nhất là nên quản lý bằng các thủ tục vì các macro lớn tạo ra một số lượng mã lệnh quá lớn nếu chúng được tham chiếu thường xuyên.

TỔNG KẾT

Trong chương này chúng ta đã nghiên cứu những vấn đề sau:

- ◆ Một macro là một khối văn bản có tên gọi. Nó có thể chứa các lệnh các toán tử giả hay tham chiếu đến các macro khác.
- ◆ Macro được tham chiếu trong thời gian biên dịch. Để mở rộng macro trình biên dịch chép toàn bộ văn bản macro vào trong chương trình tại vị trí tham chiếu đến nó, giống như người sử dụng đã đánh vào vậy. Nếu một macro có danh sách các biến giả thì các biến thực sẽ thay thế chúng. Trình biên dịch thay thế các lệnh bằng mã lệnh ngôn ngữ máy.
- ◆ Một công dụng quan trọng của macro là tạo ra các lệnh mới.
- ◆ Mở rộng macro có thể xem trong file .LST của chương trình. 3 dẫn hướng biên dịch sẽ quyết định mở rộng macro được trình bày như thế nào trong file .LST. Sau dẫn hướng .SALL phần mở rộng macro không được in ra. Sau dẫn hướng .XALL chỉ những dòng trong macro tạo lên mã nguồn mới được in ra. Và sau dẫn hướng .LALL, toàn bộ các dòng trong macro được in ra trừ những lời bình được đặt trước bằng 2 dấu chấm phẩy(;;).
- ◆ Có thể sử dụng các nhãn cục bộ trong macro. Mỗi lần macro được tham chiếu, một nhãn khác lại được tạo ra. Điều này giải quyết được vấn đề có các nhãn trùng nhau khi tham chiếu nhiều lần một macro.
- ◆ Một macro có thể tham chiếu các macro khác hay tham chiếu chính nó.
- ◆ Chúng ta có thể tạo ra một thư viện macro và bằng cách sử dụng toán tử giả INCLUDE các macro của thư viện có thể sử dụng trong chương trình.
- ◆ Macro lặp REPT được dùng để tạo ra một khối các câu lệnh. Nó có một tham số xác định số lần lặp các câu lệnh. Macro này có thể đặt tại mọi chỗ trong chương trình nơi cần lặp lại các câu lệnh hay trong bản thân một macro khác. Macro REPT không có trường tên.
- ◆ Macro IRP dùng để lặp lại một số lần tuỳ ý các câu lệnh.
- ◆ Bằng cách sử dụng các toán tử điều kiện chúng ta có thể buộc trình biên dịch biên dịch một số câu lệnh nhất định trong khi lại bỏ qua một số khác.
- ◆ Dẫn hướng biên dịch .ERR khiến cho trình biên dịch phải thông báo cho người sử dụng mỗi khi có macro được tham chiếu không đúng.

- ♦ Macro và thủ tục đều có những ưu điểm riêng của chúng. Các chương trình có sử dụng macro thường biên dịch lâu hơn tạo ra mã máy dài hơn nhưng lại thực hiện nhanh hơn. Các công việc nhỏ tốt nhất nên quản lý bằng các macro còn các công việc lớn thì nên giành cho thủ tục.

Các thuật ngữ tiếng Anh

| | |
|-------------------------------|---|
| Conditional pseudo_ops | Các toán tử điều kiện - các toán tử giả được sử dụng để biên dịch một số lệnh trong khi bỏ đi một số lệnh khác |
| Expand(a macro) | Mở rộng macro - khi trình biên dịch gặp tên macro nó sẽ thay thế tên macro bằng nội dung của nó |
| Invoke(a macro) | Tham chiếu macro - sử dụng tên macro trong chương trình |
| Local labels | Nhãn cục bộ - nhãn được định nghĩa với toán tử giả LOCAL bên trong macro. Mỗi khi macro được tham chiếu, trình biên dịch tạo ra một nhãn khác về mặt số mỗi khi nó gặp một nhãn cục bộ. |

Các toán tử giả mới

| | | |
|------------------------|------|-------|
| = | | |
| Các toán tử điều kiện* | ENDM | LOCAL |
| ELSE | .ERR | MACRO |
| ENDIF | IRP | REPT |

(*xem bảng 13-1)

Bài tập

1. Viết các macro sau đây. Tất cả các thanh ghi phải được lưu trữ trừ những thanh ghi dùng để trả lại kết quả.

- a. MUL_N MACRO N cho tích số 32 bit có dấu của AX và số N trong DX và AX.
- b. DIV_N MACRO N chia số chứa trong AX và số N và lưu thương số 16 bit có dấu trong AX. Bạn có thể giả sử N khác 0.
- c. MOD_MACRO M,N trả lại phần dư của phép chia M cho N. Chú ý rằng M và N có thể là các từ 16 bit hay các thanh ghi, bạn có thể giả sử N khác 0.
- d. POWER_MACRO N nhận số trong AX và luỹ thừa mũ N của nó, trong đó N là số nguyên dương. Kết quả được chứa trong AX, nếu kết quả quá lớn macro phải thiết lập cờ CF/OF.

2. Viết một macro C_TO_E nhận biến số C(biểu diễn nhiệt độ C) và đổi nó thành nhiệt độ F theo công thức $F = (\frac{9}{5} * C) + 32$. Để nhân với 9 và chia cho 5 bạn có thể tham chiếu các macro MUL_N và DIV_N trong bài tập 1. Kết quả nhận được cắt bỏ phần thập phân trả lại trong AX. Nếu xảy ra hiện tượng tràn trong phép nhân, cờ CF/OF phải được thiết lập .
3. Viết một macro CGD MACRO M,N tính USCLN của 2 biến số M và N. Thuật toán Euclide cho cách tính USCLN của 2 số M và N như sau:

```

WHILE N <> 0 DO
  M=M MOD N
  đổi M và N
  END WHILE
  RETURN M

```

MACRO của bạn có thể tham chiếu đến macro MOD trong bài tập 1(c).

- i. Các macro đặc biệt có ích trong các ứng dụng đồ họa. Hãy viết các macro thực hiện những công việc sau:

- Macro MOV_CURSOR MACRO R,C chuyển con trỏ tới dòng R cột C.
- Macro DISP_CHAR MACRO CHAR, ATTR hiển thị một lần ký tự CHAR với thuộc tính ATTR tại vị trí con trỏ.
- Macro CLEAR_WINDOW MACRO R1,C1,R2,C2,COLOR xoá một cửa sổ có toạ độ góc trái trên (R1,C1) và toạ độ góc phải dưới (R2,C2) với thuộc tính COLOR.
- Macro DRAW_BOX MACRO R1,C1,R2,C2 vẽ đường viền ngoài một hộp có toạ độ góc trái trên (R1,C1) và toạ độ góc phải dưới (R2,C2). Sử dụng các ký tự ASCII để vẽ các cạnh và các góc.

5. Dùng macro REPT để viết các macro dưới đây:

- Macro ALT MACRO N, trong đó N là một số nguyên dương chẵn, để khởi tạo N byte nhớ thay đổi lần lượt 0 và 1 bắt đầu từ 0. Tham chiếu macro để khởi tạo một mảng nhớ byte gồm 100 phần tử có tên là BYT.
- Macro ARITH MACRO B, I, N trong đó B,I,N là các số nguyên dương để khởi tạo một khối từ nhớ theo trình tự thuật giải sau:
B,B+I,B+2*I,...B+(N-1)*I. Cho biết cách tham chiếu macro để khởi tạo một mảng word WRD 100 phần tử mà 2 phần tử đầu tiên của nó là 10 và 12.
- Macro POWERS_OF_TWO MACRO N, trong đó N là số nguyên dương, dùng để khởi tạo một khối N từ nhớ với các giá trị 1, 2, 4,..., 2^{N-1} .

Cho biết cách tham chiếu macro để khởi tạo một mảng từ nhớ W có 10 phần tử.

d. Macro BIN MACRO N,K, trong đó N và K là các số nguyên không âm, để chuyển vào AX hệ số nhị thức $B(N,K)=N*(N-1)*(N-2)*\dots*(N-K+1)$.

6. Cho biết mã lệnh nào (nếu có) sẽ được biên dịch trong macro sau:

```
MAC1 MACRO M
    IF      M-1
    MOV     AX,M
    M=M-1
    IFE    M
    MOV     BX,M
    ENDIF
    ENDIF
ENDM
```

- a. Nếu tham chiếu: MAC1 1
- b. Nếu tham chiếu: MAC1 2

7. Cho biết mã lệnh nào (nếu có) sẽ được biên dịch trong macro sau:

```
MAC2 MACRO M, K
    REPT   M
    MOV    AX,M
    K=K+1
    IF     K-3
    MOV    BX,M
    ENDIF
    ENDM
ENDM
```

- a. Nếu tham chiếu: MAC2 5,1
- b. Nếu tham chiếu: MAC2 2,2

8. Dãy Fibonacci là dãy số 1, 1, 2, 3, 5, 8, 13, 21, 34... Viết một macro

FIB MACRO N mà sự tham chiếu nó khiến cho lệnh MOV AX,FN được biên dịch, trong đó FN là số thứ N của dãy Fibonacci. Chẳng hạn lời gọi FIB 8 sẽ khiến cho lệnh

MOV AX,21 được biên dịch. Dưới đây là thuật giải để tính số thứ N của dãy Fibonacci:

```
IF N=1
    THEN FN=1
    ELSE
        LO=0
        HI=1
```

REPEAT N-1 TIMES

X=LO

LO=HI

HI=X+LO

FN=HI.

Chương 14

QUẢN LÝ BỘ NHỚ

Tổng quan

Cho đến nay tất cả các chương trình của chúng ta đều có chứa đoạn mã, đoạn ngắn xếp và có thể cả đoạn dữ liệu. Nếu ngoài thủ tục chính ra còn có các thủ tục khác chúng sẽ được đặt trong đoạn mã, sau thủ tục chính. Trong chương này bạn sẽ thấy các chương trình có thể được cấu thành bằng các cách khác.

Trong phần 14.1 chúng ta sẽ nghiên cứu về dạng chương trình .COM trong đó mã lệnh, ngắn xếp, dữ liệu chứa trong một đoạn duy nhất. Các chương trình .COM có cấu trúc đơn giản và không chiếm nhiều bộ nhớ như các chương trình .EXE do đó các chương trình hệ thống thường được viết dưới dạng này.

Phần 14.2 sẽ trình bày về vấn đề các thủ tục có thể đặt trong các Mô đun khác nhau, được hợp dịch tách biệt và liên kết vào một chương trình duy nhất. Các Mô đun chứa các thủ tục này có thể có mã lệnh và dữ liệu của riêng nó. Khi các Mô đun được liên kết, các đoạn lệnh cũng như dữ liệu được ghép lại với nhau.

Trong phần 14.3 chúng tôi sẽ trình bày về cách định nghĩa đoạn toàn phần. Chúng cho phép quản lý hoàn toàn về trình tự, sự liên kết và vị trí của các đoạn chương trình.

Phần 14.5 nói thêm về cách định nghĩa đoạn đơn giản hóa, phương pháp mà chúng ta đã sử dụng trong cuốn sách này.

Các thủ tục chúng ta đã viết cho đến nay nói chung đều được truyền tham số thông qua các thanh ghi, phần 15 sẽ trình bày thêm một số cách khác để liên lạc với thủ tục.

14.1 Các chương trình .COM

Trong chương này chúng tôi sẽ trình bày về các chương trình mà trong đó các đoạn lệnh, dữ liệu, ngăn xếp trùng nhau. Các chương trình kiểu này còn có tên gọi là các chương trình dạng .COM vì đó là phần mở rộng của các file thực hiện. Các bạn sẽ thấy ưu điểm chính của chương trình là cấu trúc đơn giản của nó và một thực tế là chúng chiếm tương đối ít bộ nhớ. Nhược điểm của các chương trình .COM là tính không mềm dẻo và sự giới hạn về kích thước vì tất cả mọi thứ - mã lệnh, dữ liệu, ngăn xếp phải đặt chung trong một đoạn duy nhất.

Một vấn đề đối với các chương trình .COM là nếu có dữ liệu thì đặt nó ở đâu, vì chúng phải nằm chung một đoạn với mã lệnh. Chúng có thể được đặt cuối chương trình nhưng điều này yêu cầu phải khai báo đoạn toàn phần (phần 14.3). Chúng ta sẽ chọn phương án đặt dữ liệu ở đầu chương trình. Dưới đây là dạng của một chương trình .COM.

Dạng chương trình .COM

```
0: TITLE
1: .MODEL    SMALL
2: .CODE
3:          ORG      100H
4: START:
5:          JMP      MAIN
6: ;dữ liệu đặt ở đây
7: MAIN      PROC
8: ;các lệnh đặt ở đây
9: ;DOS exit
10:         MOV     AH, 4CH
11:         INT     21H
12: MAIN      ENDP
13: ;các thủ tục khác đặt ở đây
14: END      START
```

Hãy xem xét những điểm khác biệt giữa dạng chương trình trên và dạng chương trình chúng ta vẫn thường sử dụng (dạng chương trình .EXE). Trước hết chỉ có duy nhất một đoạn được định nghĩa .CODE vì dòng đầu tiên phải là một lệnh, thủ tục chính bắt đầu bằng lệnh nhảy JMP qua dữ liệu. Nhãn START chỉ ra điểm bắt đầu của chương trình, nhãn này cũng là toán hạng của dẫn hướng END trong dòng 14. Lý do phải có dẫn hướng biên dịch ORG 100h được giải thích dưới đây.

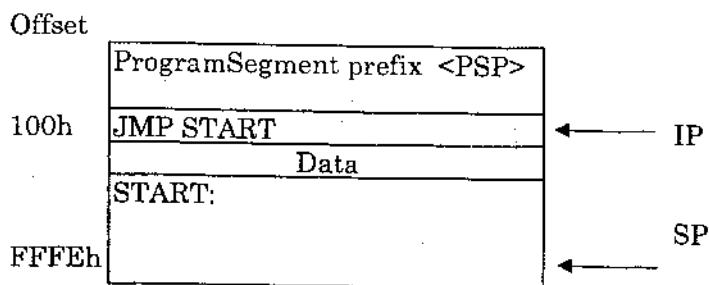
Dẫn hướng biếu dịch ORG

Trong chương 4 chúng tôi đã có nói một chương trình .EXE khi nạp vào bộ nhớ sẽ có một vùng thông tin 100h byte đặt phía trước nó, vùng này có tên gọi vùng trước chương trình (Program Segment Prefix- PSP). Điều này cũng đúng cho các chương trình dạng .COM và đối với chúng PSP chiếm 100h byte đầu tiên của đoạn.

Dẫn hướng ORG 100h gán 100h cho bộ đếm định vị (location counter) có nhiệm vụ theo dõi vị trí tương đối của câu lệnh đang được hợp dịch. Thông thường bộ đếm định vị được thiết lập giá trị 0 ở đầu chương trình, thay vào đó ORG 100h khiến cho nó nhận giá trị 100h. Bây giờ hãy giả thiết chương trình có một số dữ liệu, nếu không có dẫn hướng ORG 100h, trình biên dịch sẽ gán địa chỉ đầu của đoạn chương trình cho các biến và do đó chúng sẽ được đặt không đúng vào PSP. Bằng dẫn hướng ORG 100h các biến được gán đúng địa chỉ đầu chương trình- sau 100h byte địa chỉ bắt đầu của đoạn.

Ngăn xếp của chương trình .COM

Trong các chương trình .COM ngăn xếp đặt trong cùng một đoạn với mã lệnh và dữ liệu. Không giống các chương trình .EXE người lập trình không phải định nghĩa vùng ngăn xếp. Khi chương trình .COM nạp vào bộ nhớ, con trỏ ngăn xếp SP được khởi tạo với giá trị FFFEh- word cuối cùng trong đoạn. Vì ngăn xếp phát triển về phía đầu bộ nhớ ít khi xảy ra hiện tượng ngăn xếp va chạm với mã lệnh trừ khi ngăn xếp quá lớn hay mã lệnh quá dài. Hình 14.1 cho thấy sự bố trí của một chương trình .COM sau khi được nạp vào bộ nhớ (tất nhiên khi nó được định nghĩa như trên)



Hình 14.1 Chương trình .COM trong bộ nhớ

Ví dụ về một chương trình .COM

Dể làm ví dụ chúng ta hãy viết lại chương trình PGM4_2.ASM dưới dạng .COM. Chương trình hiển thị HELLO! Lên màn hình, để dễ so sánh chương trình PGM4_2.ASM được viết lại ở đây và được đánh số lại là PGM14_1.ASM

Chương trình nguồn PGM14_1.ASM

```
TITLE      PGM14_1:      HELLO
.MODEL     SMALL
.STACK    100H
.DATA
        MSG      DB  ' HELLO!', '$'
.CODE
MAIN      PROC
;khởi tạo DS
        MOV      AX, @DATA
        MOV      DS, AX      ;khởi tạo DS
;hiển thị lời chào
        LEA      DX, MSG    ;lấy địa chỉ MSG
        MOV      AH, 09      ;hàm hiển thị chuỗi
        INT      21H         ;hiển thị thông báo
;DOS exit
        MOV      AH, 4CH
        INT      21H
MAIN      ENDP
END      MAIN.
```

Và dưới đây là chương trình viết dưới dạng .COM.

Chương trình nguồn PGM14_2.ASM

```
TITLE      PGM14_2:      COM      DEMO
.MODEL     SMALL
.CODE
        ORG      100H
START:
        JMP      MAIN
MSG      DB      ' HELLO!$'
MAIN      PROC
        LEA      DX, MSG    ;lấy địa chỉ MSG
        MOV      AH, 09      ;hàm hiển thị chuỗi
        INT      21H         ;hiển thị 'HELLO'
        MOV      AH, 4CH      ;về DOS
        INT      21H
MAIN      ENDP
END      START
```

Chú ý rằng vì chỉ có một đoạn, các lệnh:

```
MOV AX, @DATA  
MOV DS, AX
```

cần phải có đổi với chương trình dạng .EXE có dữ liệu thì ở đây không cần thiết đổi với một chương trình .COM. Các bước hợp dịch và liên kết diễn ra như sau:

```
A>C:MASM PGM14_2;  
Microsoft (R) Macro Assembler Version 5.10.  
Copyright (C) Microsoft Corp 1981, 1988. All rights  
reserved.  
  
50116 + 418713 Bytes symbol space free  
0 Warning Errors  
0 Severe Errors  
A>C:LINK PGM14_2;  
Microsoft (R) Overlay linker version 3.64  
Copyright (C) Microsoft Corp 1981, 1988. All rights  
reserved.  
LINK :Warning L4021: no stack segment
```

Các bạn có thể yên tâm với lời cảnh báo ở đây vì chương trình .COM không có đoạn ngăn xếp riêng biệt. Đối với chương trình .COM, file .EXE được tạo ra bằng chương trình LINK không phải là file thực hiện. Nó phải được đổi thành file dạng .COM bằng một chương trình tiện ích của DOS với tên gọi EXE2BIN.

```
A>C:EXE2BIN PGM14_2 PGM14_2.COM
```

Biến đầu tiên của EXE2BIN là PGM14_2. Phần mở rộng ngầm định là .EXE. Tham số thứ 2 là PGM14_2.COM là tên của file tạo ra. File được tạo ra ở bước trước đó không cần thiết nữa và bạn nên xoá nó đi trước khi chạy chương trình. Để chạy chương trình chỉ cần đánh vào:

```
A>PGM14_2  
HELLO!  
A>
```

Như đã nói trước, một ưu điểm nổi bật của chương trình .COM là kích thước nhỏ của nó. Kích thước của file PGM14_1.EXE là 801 byte trong khi kích thước của PGM14_2.COM chỉ có 22 byte. Lý do của sự khác biệt này là một file .EXE có khối đầu 512 byte chứa các thông tin về kích thước của phần mã lệnh khả thi, nơi nó được đặt trong bộ nhớ và các dữ liệu khác. Một nguyên nhân nữa là chương trình .EXE chứa đoạn ngắn xếp riêng biệt.

14.2 Các Mô đun chương trình

Đối với các chương trình lớn với rất nhiều thủ tục thì thật tiện lợi nếu chúng ta đặt các thủ tục trong các file riêng biệt. Có 2 nguyên nhân chủ yếu để làm điều đó:

- ♦ Các thủ tục có thể được viết, hợp dịch, và kiểm tra riêng biệt bằng các lập trình viên khác nhau.
- ♦ Khi các thủ tục được hợp dịch tách biệt, chúng có thể sử dụng lại các tên cho các biến hay các nhãn lệnh. Sở dĩ có được điều đó là vì trình biên dịch cho phép các tên mang tính chất cục bộ đối với mỗi file và nó sẽ không xung đột với các tên giống như vậy trong một file khác.

Các Mô đun Assembly và đối tượng

Các thủ tục hợp dịch tách biệt phải chứa trong một Mô đun Assembly. Đây là một file .ASM chứa ít nhất một định nghĩa đoạn. Trình biên dịch lấy một Mô đun Assembly và tạo ra một file .OBJ gọi là Mô đun đối tượng (Object Module). Trình liên kết sau đó sẽ liên kết các Mô đun đối tượng vào một file khả thi dạng .EXE.

Các thủ tục NEAR và FAR

Trong phần 8.3 chúng tôi đã lưu ý các bạn về cú pháp khai báo một thủ tục:

procedure_name PROC type

trong đó type có thể là NEAR hay FAR (ngầm định là NEAR). Thủ tục NEAR là thủ tục mà bản thân nó và lời gọi nó nằm trong cùng một đoạn; ngược lại thủ tục FAR là thủ tục mà lời gọi nó nằm trong đoạn khác.

Vì thủ tục FAR nằm trong một đoạn khác, lệnh CALL trước tiên cất CS và tiếp theo là IP vào ngăn xếp, sau đó CS:IP nhận địa chỉ segment:offset của thủ tục. Khi trở về lệnh RET rút ra từ ngăn xếp 2 lần để phục hồi giá trị ban đầu của CS:IP.

Chúng ta sẽ thấy ngay sau đây một thủ tục có thể là NEAR ngay cả khi nó được biên dịch tách biệt. Một thủ tục phải có kiểu FAR chỉ khi nó và câu lệnh gọi nó không thể đặt chung trong một đoạn bộ nhớ hay khi nó được gọi từ ngôn ngữ bậc cao.

Toán tử EXTRN

Trong khi biên dịch một Mô đun, trình biên dịch phải được thông báo những tên được dùng trong Mô đun hiện thời nhưng lại được khai báo trong các Mô đun khác, nếu không chúng sẽ bị coi là các tên chưa khai báo, điều này được thực hiện bằng toán tử giả EXTRN có cú pháp như sau:

```
EXTRN external_name_list
```

trong đó external_name_list là danh sách các biến có dạng:

```
name:type
```

Ở đây name là một tên ngoài (external name), và type là một trong các dạng sau: NEAR, FAR, BYTE, WORD hay DWORD. Đối với các thủ tục khai báo ngoài, type nhận các giá trị NEAR hay FAR. Các kiểu BYTE, WORD và DWORD được sử dụng cho các biến. Ví dụ để thông báo cho trình biên dịch có thủ tục NEAR với tên gọi PROC1 và thủ tục FAR với tên gọi PROC2 chúng ta có thể viết như sau:

```
EXTRN      PROC1:NEAR,  PROC2:FAR
```

bây giờ giả sử trình biên dịch gấp câu lệnh :

```
CALL      PROC1
```

trình biên dịch biết được thông qua danh sách EXTRN rằng PROC1 nằm trong một Mô đun Assembly khác và nó giành cho thủ tục một địa chỉ không xác định, địa chỉ này sẽ được điền vào khi các Mô đun được liên kết.

Toán tử giả EXTRN có thể đặt ở mọi nơi trong Mô đun miễn là nó đứng trước tham chiếu đầu tiên đến bất cứ tên nào trong danh sách các tên khai báo ngoài. Chúng ta sẽ đặt nó ở đầu Mô đun.

Toán tử giả PUBLIC

Một thủ tục hay biến phải được khai báo với toán tử giả PUBLIC nếu nó được sử dụng trong các Mô đun khác nhau. Cú pháp như sau:

```
PUBLIC      name_list
```

Toán tử giả PUBLIC có thể đặt tại mọi nơi trong Mô đun nhưng chúng ta thường đặt nó ở gần đầu Mô đun.

Liên kết các Mô đun đối tượng

Chương trình Link liên kết các Mô đun đối tượng vào một chương trình ngôn ngữ máy khả thi. Nó sẽ cố gắng tìm các tên khai báo trong dãy hướng EXTRN trong các Mô đun khác với khai báo PUBLIC. Nó liên kết các đoạn dữ liệu và đoạn lệnh trong các Mô đun khác nhau theo các khai báo của những đoạn này (xem phần 14.3). Do biết được vị trí tương đối của các lệnh và dữ liệu, trình biên dịch có thể điền những địa chỉ còn chưa xác định.

Để làm ví dụ chúng ta viết lại chương trình PGM4_3.ASM hiển thị một thông báo, đọc vào các chữ thường và đổi thành các chữ hoa. Có 2 Mô đun Assembly, Mô đun thứ nhất chứa thủ tục chính, nó hiển thị thông báo, cho người sử dụng đánh vào một chữ thường và gọi thủ tục CONVERT đổi chữ cái thành chữ hoa và hiển thị nó với một thông báo khác. Thủ tục CONVERT nằm trong Mô đun thứ 2.

Chương trình nguồn PGM14_3.ASM: First module

```
0: TITLE      PGM14_3:      CASE CONVERSION
1: EXTRN      CONVERT:NEAR
2: .MODEL     SMALL
3: .STACK    100H
4: .DATA
5: MSG        DB      ' ENTER A LOWERCASE LETTER:$'
6: .CODE
7: MAIN       PROC
8:           MOV     AX, @DATA
```

```

9:      MOV     DS, AX      ;khởi tạo DS
10:     MOV     AH, 09      ;hàm hiển thị chuỗi
11:     LEA     DX, MSG    ;nạp địa chỉ MSG
12:     INT     21H
13:     MOV     AH, 01      ;hàm đọc ký tự
14:     INT     21H
15:     CALL    CONVERT   ;đổi ra chữ hoa
16: ;DOS exit
17:     MOV     AH, 4CH
18:     INT     21H
19: MAIN  ENDP
20:     END     MAIN

```

Mô đun đầu tiên chứa các đoạn dữ liệu đoạn mã và đoạn ngắn xếp. Sau khi khởi tạo DS ở dòng 8 và 9, chương trình in ra thông báo 'ENTER A LOWERCASE LETTER:' rồi gọi thủ tục CONVERT. Sự tồn tại của CONVERT như một thủ tục trong một Mô đun khác được thông báo cho trình biên dịch thông qua dẫn hướng biên dịch EXTRN ở dòng 1. Mô đun đầu tiên kết thúc bằng dẫn hướng biên dịch END ở dòng 19 với điểm bắt đầu chương trình MAIN.

Chương trình nguồn PGM14_3A.ASM: Second module

```

0: TITLE    PGM14_3A:  CONVERT
1: PUBLIC   CONVERT
2: .MODEL   SMALL
3: .DATA
4: MSG      DB      0DH, 0AH, 'IN UPPERCASE IT IS'
5: CHAR     DB      -20H, '$'
6: .CODE
7: CONVERT  PROC    NEAR
8: ;đổi ký tự trong AL thành chữ hoa
9:         PUSH   BX
10:        PUSH   DX
11:        ADD    CHAR, AL ;đổi thành chữ hoa
12:        MOV    AH, 09      ;hàm hiển thị chuỗi
13:        LEA    DX, MSG    ;lấy địa chỉ MSG
14:        INT    21H
15:        POP    DX
16:        POP    BX
17:        RET
18: CONVERT ENDP
19: END

```

Mô đun chứa thủ tục CONVERT chứa các đoạn dữ liệu và lệnh của riêng nó. Khi các Mô đun được liên kết, các đoạn dữ liệu được ghép lại thành một đoạn

duy nhất tương tự như vậy đối với các đoạn lệnh (chúng ta sẽ thấy nguyên nhân của việc này trong phần 14.3).

Tại dòng 1, CONVERT được khai báo PUBLIC cho phép nó được gọi từ Mô đun thứ nhất. Tại dòng 7 thủ tục được khai báo NEAR vì các đoạn lệnh của 2 Mô đun sẽ liên kết với nhau làm một. Vì các đoạn dữ liệu cũng được liên kết, không cần thiết phải khởi tạo DS trong Mô đun thứ 2, công việc này đã được thực hiện trong Mô đun thứ nhất.

Mô đun thứ 2 được kết thúc bằng dẫn hướng biên dịch END, khác với Mô đun trước, nó không có toán hạng.

Sau khi đã cất các thanh ghi sẽ sử dụng, CONVERT bắt đầu tại dòng 11 bằng việc cộng chữ thường trong AL với -20h chứa trong biến CHAR nhờ đó chữ được đổi thành chữ hoa (giả thiết người sử dụng đánh vào một chữ thường). Tại dòng 12-14 thủ tục đưa ra thông báo sau cùng. Chú ý rằng tên MSG được sử dụng trong cả 2 Mô đun. Nay giờ chúng ta hãy biên dịch và liên kết các Mô đun. MASM và LINK nằm trong ổ đĩa C còn các file nguồn trong ổ đĩa A, A là ổ đĩa hiện thời.

```
A>C:MASM PGM14_3;
Microsoft (R) Macro Assembler Version 5.10.
Copyright (C) Microsoft Corp 1981, 1988. All rights
reserved.

49984 + 390317 Bytes symbol space free
0 Warning Errors
0 Severe Errors
```

```
A>C:MASM PGM14_3A;
Microsoft (R) Macro Assembler Version 5.10.
Copyright (C) Microsoft Corp 1981, 1988. All rights
reserved.

49976 + 390325 Bytes symbol space free
0 Warning Errors
0 Severe Errors
```

MASM đã tạo ra mô đun đối tượng PGM14_3.OBJ và PGM14_3A.OBJ để liên kết chúng, ta đánh

```
C:LINK PGM14_3 + PGM14_3A;
Microsoft (R) Overlay linker version 3.64
Copyright (C) Microsoft Corp 1981, 1988. All rights
reserved

Run File[ PGM14_3.EXE] :
List File [ NUL..MAP] :PGM14_3
Libraries File [ .LIB] :
```

File .EXE có tên giống như Mô đun đầu tiên được liên kết. Máy yêu cầu chúng ta cho tên để tạo ra file .MAP, thông qua file ta có thể biết được kích thước và sự bố trí của đoạn lệnh, dữ liệu và ngăn xếp. Nội dung file .MAP được liệt kê trong hình 14.2.

Trong file .MAP chúng ta sẽ thấy các đoạn lệnh của 2 Mô đun được ghép lại và cũng như vậy đối với các đoạn dữ liệu. Chương trình LINK đã chọn thứ tự mã lệnh, dữ liệu và ngăn xếp. Chúng tôi sẽ chỉ ra sau này cách thay đổi trình tự các đoạn. Nhưng trong phần lớn các chương trình hợp ngữ đơn thì trình tự không mấy quan trọng.

Ví dụ thực hiện:

```
C>PGM14_3
ENTER A LOWERCASE LETTER:r
IN UPPERCASE IT IS R
```

Các thư viện

Thay vì liên kết các Mô đun đối tượng riêng biệt để tạo thành một file .EXE như đã trình bày ở trên, chúng ta có thể tạo ra một file thư viện của các Mô đun đối tượng chứa các thủ tục. Khi thư viện được xác định trong quá trình liên kết, chương trình LINK lấy ra bất cứ thủ tục nào mà chương trình cần từ thư viện và tạo ra file .EXE. Chương trình LIB được dùng để quản lý các thư viện. Chẳng hạn để tạo lên một thư viện với tên gọi MYLIB, chúng ta có thể viết như sau:

```
LIB MYLIB;
```

| Start | Stop | Length | Name | Class |
|----------------------------------|--------|--------|-------|-------|
| 00000H | 00028H | 00029H | -TEXT | CODE |
| 0002AH | 0005DH | 00034H | -DATA | DATA |
| 00060H | 0015FH | 00100H | STACK | STACK |
| Origin | | Group | | |
| 0002:0 | | DGROUP | | |
| Program entry point at 0000:0000 | | | | |

Hình 14.2 PGM14_3.MAP

Câu lệnh này tạo ra một file thư viện MYLIB.LIB. Để thêm một Mô đun đối tượng vào trong thư viện chẳng hạn PGM14_3A.OBJ chúng ta có thể viết:

```
LIB MYLIB + PGM14_3A;
```

Trong trường hợp MYLIB.LIB chưa tồn tại câu lệnh trên sẽ tạo ra nó và gán PGM14_3A.OBJ vào trong thư viện. Nếu có một chương trình PROC.ASM cần một thủ tục trong thư viện MYLIB.LIB trước hết chúng ta biên dịch PROC.ASM thành PROC.OBJ và sau đó đánh vào như sau:

```
LINK PROC + MYLIB.LIB;
```

Ví dụ chúng ta có thể lấy thủ tục CONVERT từ thư viện và ghép nó vào PGM14_3.OBJ bằng cách đánh vào như sau:

```
LINK PGM14_3 + MYLIB.LIB;
```

Câu lệnh này tạo ra file PGM14_3.EXE, khi chạy chương trình sẽ cho kết quả như sau:

```
C>PGM14_3
```

```
ENTER A LOWERCASE LETTER:r
```

```
IN UPPERCASE IT IS R
```

Chúng ta cũng có thể xem danh sách các thủ tục trong thư viện bằng cách sử dụng chương trình LIB. Ví dụ:

```
C>LIB MYLIB
Microsoft (R) Library Manager Version 3.10
Copyright (C) Microsoft Corp 1983~1988. Allright reserved

operations:
List file :Mylib
```

Khi chương trình LIB yêu cầu cho tên file danh sách chúng ta trả lời MYLIB. Kết quả là một file danh sách có tên MYLIB như sau được tạo lên:

```
C>TYPE MYLIB
CONVERT.....pgm14_3a

pgm14_3a Offset: 00000010H Code and Data size: 29H
CONVERT
```

Danh sách trình bày tên các Mô đun và các thủ tục chứa trong chúng. Trong trường hợp này chỉ có duy nhất một Mô đun trong thư viện đó là PGM14_3A.OBJ và nó chứa duy nhất một thủ tục CONVERT. Để hiểu thêm về chương trình tiện ích LIB bạn hãy tham khảo thêm cuốn Microsoft Codeview and Utilities manual.

14.3 Các định nghĩa đoạn toàn phần

Phương pháp định nghĩa đoạn đơn giản hóa mà chúng ta vẫn thường sử dụng từ trước đến nay đã đáp ứng được hầu hết các mục tiêu. Trong phần này chúng ta xem xét cách định nghĩa đoạn toàn phần. Dưới đây là các nguyên nhân chính để sử dụng phương pháp này:

- Với các trình biên dịch version 5.0 trở về trước chúng ta chỉ có thể sử dụng phương pháp định nghĩa đoạn toàn phần.
- Với phương pháp định nghĩa đoạn toàn phần, người lập trình có thể điều khiển được thứ tự các đoạn, việc kết hợp chúng với nhau và sự liên kết giữa chúng trong bộ nhớ.

Dẫn hướng biên dịch Segment

Dạng đầy đủ của dẫn hướng biên dịch Segment như sau:

```
name      SEGMENT      align combine    'class'
```

Các toán hạng align, combine và class có các kiểu tùy chọn và sẽ được đề cập tới trong phần tiếp theo. Để kết thúc một đoạn, chúng ta có:

```
name      ENDS
```

Ví dụ chúng ta có thể định nghĩa một đoạn có tên gọi D_SEG như sau:

```
D_SEG      SEGMENT  
;dữ liệu đặt ở đây  
D_SEG      ENDS
```

Bây giờ chúng ta hãy xem xét các toán hạng của dẫn hướng biên dịch segment.

Kiểu ALIGN

Kiểu align của khai báo đoạn quyết định địa chỉ bắt đầu của đoạn được chọn như thế nào khi chương trình được nạp vào bộ nhớ. Bảng 14.1 đưa ra những phần chọn.

Ví dụ sau đây sẽ làm sáng tỏ ý nghĩa kiểu align của đoạn. Chẳng hạn 2 đoạn SEG1 và SEG2 được khai báo như sau:

```
SEG1      SEGMENT      PARA  
DB        11H DUP (1)  
SEG1      ENDS  
SEG2      SEGMENT      PARA  
DB        10H DUP (2)  
SEG2      ENDS
```

Giả sử 2 đoạn được nạp liên tiếp vào bộ nhớ, với SEG1 được gán địa chỉ đoạn là 1010h. 11h byte của SEG1 sẽ kéo dài từ địa chỉ 1010:0000h đến 1010:0010h. Bây giờ vì SEG2 có kiểu align là PARA, nó bắt đầu từ biên giới đoạn sẵn có tiếp theo bắt đầu tại 1010:0000h chính bằng 1010:0020h. Dưới đây là bộ nhớ được hiển thị bằng trình DEBUG:

```

1010:0000 01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 01
1010:0010 01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
1010:0020 02 02 02 02 02 02 02 02-02 02 02 02 02 02 02 02

```

Như chúng ta thấy có một phần trống Fh=15 byte (nhận giá trị 00) từ cuối đoạn SEG1 đế đầu đoạn SEG2. Phần trống này là phần lãng phí vì nó không phải là dữ liệu của đoạn nào cả.

| | |
|-------------|---|
| PARA | segment bắt đầu tại đoạn săn có tiếp theo (chữ số hex hàng đơn vị của địa chỉ vật lý=0) |
| BYTE | segment bắt đầu tại byte săn có tiếp theo |
| WORD | Segment bắt đầu tại word săn có tiếp theo(bit lsb của địa chỉ vật lý=0) |
| PAGE | Segment bắt đầu tại trang tiếp theo(2 chữ số hex trọng lượng thấp nhất của địa chỉ vật lý đều bằng 0) <lưu ý PARA là kiểu align ngầm định> |

Bảng 14.1 Các kiểu align

Bây giờ giả sử các đoạn được khai báo như sau:

```

SEG1      SEGMENT      PARA
DB        11H DUP (1)
SEG1      ENDS
SEG2      SEGMENT      BYTE
DB        10H DUP (2)
SEG2      ENDS

```

Trong đó SEG2 được khai báo có kiểu align là BYTE, nếu các đoạn được nạp lần lượt thì bộ nhớ sẽ như sau:

```

1010:0000 01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 01
1010:0010 01 02 02 02 02 02 02 02-02 02 02 02 02 02 02 02
1010:0020 02 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

```

Các đoạn được ghép vào một đoạn bộ nhớ duy nhất và không bỏ phí một khoảng trống nào.

Kiểu Combine

Nếu một chương trình chứa các đoạn cùng tên, kiểu combine sẽ giúp chúng được ghép lại khi chương trình được nạp vào bộ nhớ. Bảng 14.2 chỉ ra các lựa chọn được sử dụng thường xuyên.

Trình biên dịch sẽ có một thông báo lỗi nếu đoạn ngắn xếp không có kiểu combine STACK. Đối với các đoạn khác, nếu kiểu combine không được chỉ ra thì đoạn chương trình sẽ được nạp vào chính đoạn bộ nhớ của nó.

Kiểu combine PUBLIC được sử dụng thường xuyên để ghép các đoạn lệnh cùng tên trong các Mô đun khác nhau vào một đoạn mã lệnh duy nhất. Điều này có nghĩa là tất cả mọi thủ tục đều có thể có kiểu NEAR. Tương tự các đoạn dữ liệu PUBLIC có thể ghép lại vào một đoạn dữ liệu duy nhất. Ưu điểm của nó là DS chỉ phải khởi tạo một lần và không phải thay đổi để truy nhập dữ liệu, điều này chúng ta đã thấy trong PGM14_3 khi các đoạn dữ liệu được ghép lại.

Các đoạn dữ liệu ở những Mô đun khác nhau có thể có cùng tên và kiểu combine COMMON, nhờ đó các biến trong Mô đun này có thể dùng chung ô nhớ với các biến trong Mô đun khác. Để minh họa hoạt động của COMMON giả sử chúng ta khai báo như sau:

```
D SEG      SEGMENT      COMMON
A          DB           11H DUP  (1)
D SEG      ENDS
```

trong FIRST.ASM và

```
D SEG      SEGMENT      COMMON
B          DB           10H DUP  (2)
D SEG      ENDS
```

trong SECOND.ASM. Bây giờ giả sử các Mô đun được biên dịch và liên kết như sau:

```
C>LINK FIRST + SECOND;
```

Khi đó chúng sẽ chồng lên nhau trong bộ nhớ và các biến A, B có cùng địa chỉ. Kích thước của đoạn dữ liệu chung sẽ bằng kích thước của đoạn dài hơn (11h byte). Tuy nhiên các giá trị của byte sẽ là những giá trị trong SECOND vì nó là Mô đun sau cùng được đưa ra trong dòng lệnh của lệnh LINK. Bộ nhớ sẽ có dạng như sau:

```
1010:0000 02 02 02 02 02 02 02 02-02 02 02 02 02 02 02 02  
1010:0010 01 00 00 00 00 03 00 00-00 00 00 00 00 00 00 00
```

Bảng 14.2 Các kiểu Combine

| | |
|---------------------|--|
| PUBLIC | Các đoạn cùng tên được ghép lại với nhau tạo thành một khối nhớ liên tục và duy nhất |
| COMMON | Các đoạn cùng tên bắt đầu tại cùng một vị trí trong bộ nhớ, và như vậy sẽ chồng lên nhau |
| STACK | Có cùng tác dụng như PUBLIC ngoại trừ một điều là tất cả các địa chỉ offset của các lệnh và số liệu đều tính tương đối với thanh ghi SS, SP được khởi tạo trả đến cuối đoạn. |
| AT paragraph | Chỉ ra rằng segment phải bắt đầu tại một đoạn xác định. |

Kiểu class

Kiểu class trong khai báo đoạn xác định trình tự các đoạn, được nạp vào bộ nhớ. Khai báo kiểu class phải được đặt trong cặp ngoặc đơn.

Nếu có 2 hoặc nhiều hơn 2 đoạn có cùng kiểu class, chúng sẽ được nạp theo trình tự liên tiếp nhau đoạn này sau đoạn kia. Nếu các kiểu class không xác định trong khai báo đoạn, các đoạn được nạp theo trình tự xuất hiện của chúng trong chương trình nguồn. Chẳng hạn nếu chúng ta khai báo:

```
C_SEG SEGMENT 'CODE'  
;đặt thủ tục chính ở đây  
C_SEG ENDS
```

trong FIRST.ASM và:

```
C1_SEG SEGMENT 'CODE'  
;các thủ tục khác đặt ở đây  
C1_SEG ENDS
```

trong SECOND.ASM và chỉ có 2 đoạn này có kiểu class 'CODE' thì khi các Mô đun được biên dịch và liên kết :

```
C>LINK FIRST + SECOND
```

thì C1_SEG sẽ nằm sau C_SEG trong bộ nhớ. Tuy nhiên vẫn có thể có khoảng trống giữa hai đoạn, để tránh điều đó, C1_SEG có thể khai báo kiểu align BYTE.

14.3.1 Dạng chương trình .EXE với kiểu định nghĩa đoạn toàn phần

Dạng của một chương trình .EXE với kiểu định nghĩa đoạn toàn phần có một chút khác biệt khi định nghĩa đoạn đơn giản hóa. Dưới đây là dạng chuẩn:

```
S_SEG      SEGMENT STACK
            DB      100H      DUP    (? )
S_SEG      ENDS
D_SEG      SEGMENT
;đữ liệu đặt ở đây
D_SEG      ENDS
C_SEG      SEGMENT
            ASSUME CS:C_SEG, SS:S_SEG, DS:D_SEG
MAIN      PROC
;khởi tạo DS
            MOV     AX,D_SEG
            MOV     DS,AX
;các lệnh khác đặt ở đây
;trở về DOS
            MOV     AH,4CH
            INT     21H
MAIN      ENDP
;các thủ tục khác có thể đặt ở đây
C_SEG      ENDS
END      MAIN
```

Các tên đoạn ở đây là tùy ý. Có một dẫn hướng biên dịch mới là ASSUME nên chúng ta sẽ giải thích vai trò của nó dưới đây.

Dẫn hướng biên dịch ASSUME

Khi chương trình được biên dịch, trình biên dịch phải biết đoạn nào là đoạn lệnh, đoạn dữ liệu, đoạn ngăn xếp. Mục đích của dẫn hướng ASSUME là gắn các thanh ghi CS, DS, SS và thậm chí cả ES với các đoạn phù hợp. Với cách định nghĩa đoạn đơn giản hóa, các thanh ghi đoạn tự động được gắn với các đoạn một cách phù hợp do đó không cần dẫn hướng ASSUME. Tuy nhiên với

các chương trình có dữ liệu chúng ta vẫn phải chuyển địa chỉ đoạn dữ liệu vào DS vì như đã lưu ý các bạn trong chương 4, ban đầu DS chưa địa chỉ đoạn của PSP.

14.3.2 Sử dụng phương pháp định nghĩa đoạn toàn phần

Để xem phương pháp định nghĩa đoạn toàn phần làm việc ra sao chúng ta sử dụng chúng để viết lại chương trình PGM14_3A.ASM và PGM14_3.ASM. chúng ta sẽ làm điều này theo 2 cách, trong phiên bản thứ nhất chúng ta sử dụng các toán hạng mặc định cho các dẫn hướng biên dịch về đoạn.

Chương trình nguồn PGM14_4.ASM: First Module

```
0: TITLE      PGM14_4:      CASE CONVERSION
1: EXTRN      CONVERT:FAR
2: S_SEG      SEGMENT STACK
3:           DB      100 DUP  (0)
4: S_SEG      ENDS
5: D_SEG      SEGMENT
6: MSG        DB      'ENTER A LOWERCASE LETTER:$'
7: D_SEG      ENDS
8: C_SEG      SEGMENT
9:           ASSUME CS:C_SEG,DS:D_SEG,SS:S_SEG
10: MAIN      PROC
11:           MOV     AX,D_SEG
12:           MOV     DS,AX      ;khởi tạo DS
13:           MOV     AH,09      ;hàm hiển thị chuỗi
14:           LEA     DX,MSG      ;nạp địa chỉ MSG vào DX
15:           INT     21H      ;hiển thị lời nhắc
16:           MOV     AH,1      ;hàm đọc ký tự từ bàn phím
17:           INT     21H      ;nhập ký tự
18:           CALL    CONVERT   ;đổi nó thành chữ hoa
19:           MOV     AH,4CH
20:           INT     21H      ;về DOS
21: MAIN      ENDP
22: C_SEG      ENDS
23:           END     MAIN
```

Chương trình nguồn PGM14_4A.ASM Second Module

```
24:TITLE      PGM14_4A:      CONVERT
25:PUBLIC     CONVERT
26:D_SEG      SEGMENT
27:MSG        DB      0DH,0AH,' IN UPPERCASE IT IS'
28:CHAR       DB      -20H,'$'
29:D_SEG      ENDS
```

```

30: C_SEG      SEGMENT
31:          ASSUME CS:C_SEG, DS:D_SEG
32: CONVERT    PROC    FAR
33: ;đổi ký tự trong AL thành chữ hoa
34:          PUSH   DS      ;cất DS
35:          PUSH   DX      ;và DX
36:          MOV    DX,D_SEG ;reset lại DS cho trở đến đoạn
37:          MOV    DS,DX   ;dữ liệu cục bộ
38:          ADD    CHAR,AL ;đổi thành chữ hoa
39:          MOV    AH,09   ;hàm hiển thị chuỗi
40:          LEA    DX,MSG  ;lấy địa chỉ MSG
41:          INT    21H
42:          POP    DX      ;phục hồi DX và
43:          POP    DS      ;DS
44:          RET
45: CONVERT    ENDP
46: C_SEG      ENDS
47:          END

```

Các bạn hãy lưu ý những điểm sau:

- Chúng ta đã chọn tên C_SEG cho cả 2 đoạn lệnh trong 2 Mô đun, vì chúng không có kiểu combine là PUBLIC nên chúng sẽ nằm trong các đoạn bộ nhớ riêng biệt khi các Mô đun được biên dịch và liên kết. Điều này có nghĩa là thủ tục CONVERT phải có kiểu FAR (dòng 1,32)
- Vì các đoạn dữ liệu cũng không có kiểu PUBLIC, chúng cũng chiếm những đoạn bộ nhớ phân biệt. Vì vậy thủ tục CONVERT phải đổi DS để truy nhập dữ liệu trong Mô đun thứ 2 (dòng 36,37). Chúng ta sử dụng DX thay vì AX để chuyển địa chỉ đoạn vào DS vì CONVERT nhận giá trị vào của nó trong AL.

Sau khi biên dịch và liên kết các Mô đun hãy xem file .MAP (hình 14.3). Các đoạn xuất hiện theo trình tự xuất hiện của chúng trong chương trình nguồn. Và vì các đoạn được khai báo với kiểu align là mặc định (PARA), có những khoảng trống giữa chúng.

Bây giờ chúng ta hãy viết lại các Mô đun trên để lợi dụng những ưu điểm của các dẫn hướng biên dịch đối với đoạn. Dưới đây là các yêu cầu:

- Các đoạn lệnh của 2 chương trình phải được ghép lại thành một đoạn duy nhất cũng như các đoạn dữ liệu.
- Không được phép có các khoảng trống giữa các đoạn.
- Trình tự của các đoạn trong chương trình cuối cùng là ngăn xếp, dữ liệu, mã lệnh.

| Start | Stop | Length | Name | Class |
|--------|--------|--------|-------|-------|
| 00000H | 00063H | 00064H | S_SEG | |
| 00070H | 0008AH | 0001BH | D_SEG | |
| 00090H | 000A9H | 0001AH | C_SEG | |
| 000B0H | 000C7H | 00018H | D_SEG | |
| 000D0H | 000E5H | 00016H | C_SEG | |

Program entry point at 0009:0000

Hình 14.3 PGM14_4.MAP

Chương trình nguồn PGM14_5.ASM:First module

```

0: TITLE      PGM14_5:    CASE CONVERSION
1: EXTRN      CONVERT:NEAR
2: S_SEG      SEGMENT STACK
3:           DB      100 DUP  (0)
4: S_SEG      ENDS
5: D_SEG      SEGMENT BYTE      PUBLIC ' DATA'
6: MSG        DB      ' ENTER A LOWERCASE LETTER:$'
7: D_SEG      ENDS
8: C_SEG      SEGMENT BYTE      PUBLIC ' CODE'
9: ASSUME    CS:C_SEG,DS:D_SEG,SS:S_SEG
10:MAIN      PROC
11:          MOV     AX,D_SEG
12:          MOV     DS,AX      ;khởi tạo DS
13:          MOV     AH,09      ;hàm hiển thị chuỗi
14:          LEA     DX,MSG    ;nap địa chỉ MSG vào DX
15:          INT     21H      ;hiển thị lời nhắc
16:          MOV     AH,1      ;hàm đọc ký tự từ bàn phím
17:          INT     21H
18:          CALL    CONVERT   ;đổi nó thành chữ hoa
19:          MOV     AH,4CH
20:          INT     21H      ;về DOS
21:MAIN      ENDP
22:C_SEG      ENDS
23:          END     MAIN

```

Chương trình nguồn PGM14_5A.ASM: Second module

```

1: TITLE    PGM14_5A:  CONVERT
2: PUBLIC   CONVERT
3: D_SEG    SEGMENT BYTE      PUBLIC 'DATA'
4: MSG      DB      0DH,0AH,'IN UPPERCASE IT IS'
5: CHAR     DB      -20H,'$'
6: D_SEG    ENDS
7: C_SEG    SEGMENT BYTE      PUBLIC 'CODE'
8:          ASSUME CS:C_SEG,DS:D_SEG
9: CONVERT  PROC  NEAR
10: ;đổi ký tự trong AL thành chữ hoa
11:         PUSH   DX      ; cất DX
12:         ADD    CHAR,AL  ;đổi thành chữ hoa
13:         MOV    AH,09      ;hàm hiển thị chuỗi
14:         LEA    DX,MSG1  ;lấy địa chỉ MSG1
15:         INT    21H
16:         POP    DX      ;phục hồi DX
17:         RET
18: CONVERT ENDP
19: C_SEG   ENDS
20: END

```

Giống như trước chúng ta biên dịch và liên kết các Mô đun. Hình 14.4 cho thấy file .MAP. Nó chỉ ra rằng các đoạn dữ liệu và các đoạn lệnh của 2 Mô đun được ghép lại thành một đoạn duy nhất và không có khoảng trống giữa chúng. Dưới đây là các toán hạng của dẫn hướng biên dịch SEGMENT đã sử dụng:

1. Bằng cách sử dụng các tên trùng nhau cho các đoạn mã và dữ liệu trong 2 Mô đun, và sử dụng kiểu combine PUBLIC chúng ta tạo ra một chương trình chỉ chứa 3 đoạn. Hơn nữa khoảng trống giữa chúng lại bị loại bỏ vì chúng ta đã sử dụng kiểu align BYTE. Vì kiểu combine PUBLIC khiến cho các đoạn cùng tên được nối lại với nhau, việc sử dụng kiểu class 'CODE' và 'DATA' là thừa.
2. Vì dữ liệu trong cả 2 Mô đun bây giờ tạo thành một đoạn duy nhất, không cần thiết phải thay đổi DS trong thủ tục CONVERT, và CONVERT không cần cất thanh ghi DS. Đây chính là nguyên nhân chủ yếu để ghép các đoạn dữ liệu.
3. Vì bây giờ chỉ có một đoạn mã lệnh, chúng ta có thể cho CONVERT thuộc tính NEAR.

| Start | Stop | Length | Name | Class |
|--------|--------|--------|-------|-------|
| 00000H | 00063H | 00064H | S_SEG | |
| 00064H | 00096H | 00033H | D_SEG | DATA |
| 00097H | 000BDH | 00027H | C_SEG | CODE |

Program entry point at 0009:0000

Hình 14.4 PGM14_5.MAP

14.4 Đôi điều nói thêm về phương pháp định nghĩa đoạn đơn giản hoá

Bây giờ khi đã thấy các định nghĩa đoạn toàn phần chúng ta hãy nói thêm một chút về các đặc điểm của định nghĩa đoạn đơn giản hoá mà chúng ta đã sử dụng trong suốt cuốn sách này.

Trước tiên, như chúng ta thấy trong phần 4.7.1, chế độ bộ nhớ phải được xác định khi các định nghĩa đoạn đơn giản hoá được sử dụng. Các chế độ bộ nhớ được lựa chọn tùy theo có bao nhiêu đoạn lệnh và đoạn dữ liệu. Cú pháp như sau:

```
.MODEL memory_model
```

trong đó memory_model là một trong các kiểu được trình bày trong bảng 14.3. Trừ trường hợp có rất nhiều mã lệnh hay số liệu, kiểu .SMALL là đủ dùng cho hầu hết các chương trình hợp ngữ.

Thứ hai, đối với kiểu SMALL bảng 14.4 cho thấy các định nghĩa đoạn đơn giản hoá, tên cũng như các kiểu align, combine và class mặc định của chúng. Thêm vào các đoạn .CODE, .DATA, .STACK chúng ta vẫn thường dùng, dữ liệu chưa khởi tạo có thể khai báo trong một đoạn riêng biệt .DATA?, và dữ liệu mà chương trình không thể thay đổi có thể đặt trong đoạn .CONST. Ví dụ:

```
.MODEL SMALL
.STACK 100H
.DATA
X DW 5
.DATA?
Y DW ?
```

```

.CONST
MSG      DB      ' HELLO$'
.CODE
MAIN     PROC
...
MAIN     ENDP
END     MAIN

```

Dưới đây là các câu lệnh khởi tạo thường dùng:

```

MOV      AX, @DATA
MOV      DS, AX

```

những câu lệnh này cho phép chương trình truy nhập vào các đoạn .DATA, .DATA?, .CONST. Sở dĩ có điều này vì chương trình LINK thực chất đã liên kết những đoạn chương trình này vào một đoạn bộ nhớ duy nhất.

Thứ ba là đối với kiểu SMALL, khi .CODE được sử dụng để định nghĩa các đoạn lệnh trong các Mô đun hợp dịch tách biệt, các đoạn này có cùng một tên mặc định là _TEXT và kiểu combine PUBLIC. Nhờ đó khi các Mô đun được liên kết, các đoạn mã lệnh được ghép vào một đoạn mã duy nhất; cũng như vậy các đoạn dữ liệu định nghĩa bằng .DATA được ghép vào một đoạn dữ liệu duy nhất. Chúng ta đã thấy điều này trong chương trình PGM14_3.ASM.

| | |
|----------------|--|
| SMALL | Mã lệnh nằm trong một đoạn, dữ liệu nằm trong một đoạn |
| MEDIUM | Mã lệnh lớn hơn một đoạn, dữ liệu nằm trong một đoạn |
| COMPACT | Mã lệnh nằm trong một đoạn, dữ liệu lớn hơn một đoạn |
| LARGE | Mã lệnh lớn hơn một đoạn Dữ liệu lớn hơn một đoạn Không có mảng lớn hơn 64kb |
| HUGE | Mã lệnh lớn hơn một đoạn Dữ liệu lớn hơn một đoạn Mảng có thể lớn hơn 64kb |

Bảng 14.3 Các chế độ bộ nhớ

| Đoạn | Tên ngầm định | Align | Combine | class |
|--------|---------------|-------|---------|---------|
| .CODE | TEXT | WORD | PUBLIC | 'CODE' |
| .DATA | DATA | WORD | PUBLIC | 'DATA' |
| .DATA? | BSS | WORD | PUBLIC | 'BSS' |
| .STACK | STACK | PAKA | STACK | 'STACK' |
| .CONST | CONST | WORD | PUBLIC | 'CONST' |

Bảng 14.4 Các đoạn trong chế độ SMALI.

14.5 Truyền dữ liệu giữa các thủ tục

Trong phần 8.3 chúng ta đã trình bày vấn đề truyền dữ liệu giữa các thủ tục. Vì các thủ tục của hợp ngữ không có các danh sách tham số đi kèm như các thủ tục của ngôn ngữ bậc cao, trách nhiệm của người lập trình là phải nghĩ ra một phương pháp để truyền dữ liệu giữa chúng. Cho đến nay chúng ta đã tiến hành truyền dữ liệu thông qua các thanh ghi.

14.5.1 Các biến toàn cục

Chúng ta đã sử dụng các dẫn hướng biến dịch EXTRN và PUBLIC để chỉ ra rằng một thủ tục khai báo trong một Mô đun có thể gọi được từ một Mô đun khác. Chúng ta cũng dùng chúng để khai báo dữ liệu trong một Mô đun và tham chiếu đến nó trong một Mô đun khác. Theo khái niệm của ngôn ngữ bậc cao những biến này được gọi là biến toàn cục (Global variables). Một ưu điểm của biến toàn cục là các thủ tục không cần phải thêm các lệnh để chuyển dữ liệu giữa chúng.

Để làm ví dụ chương trình sau sẽ in ra dấu nhắc người sử dụng, đọc vào 2 chữ số thập phân có tổng nhỏ hơn 10, in chúng và tổng của chúng ở dòng tiếp theo. Đây chính là nội dung của bài tập 4.7.

Chương trình nguồn PGM14_6.ASM:First module

```

0: TITLE    PGM14_6:    ADD DIGITS
1: EXTRN    ADDNO$: NEAR
2: PUBLIC   DIGIT1, DIGIT2, SUM
3: .MODEL   SMALL
4: .STACK   100H
5: .DATA
6: MSG      DB      'ENTER TWO DIGITS:$'

```

```

7: MSG1      DB      0DH, 0AH 'THE SUM OF '
8: DIGIT1    DB      ?
9:          DB      'AND'
10: DIGIT2   DB      ?
11:          DB      'IS'
12: SUM       DB      -30H, ' $'
13: .CODE
14: MAIN      PROC
15: ;khởi tạo DS
16:          MOV     AX, @DATA
17:          MOV     DS, AX
18: ;thông báo cho người sử dụng
19:          MOV     AH, 09 ;hàm hiển thị chuỗi
20:          LEA     DX, MSG ;lấy địa chỉ chuỗi
21:          INT     21H ;hiển thị
22: ;đọc vào 2 số
23:          MOV     AH, 01 ;hàm đọc từ bàn phím
24:          INT     21H ;ký tự biểu diễn số trong AL
25:          MOV     DIGIT1, AL;cất vào biến DIGIT1
26:          INT     21H ;ký tự biểu diễn số trong AL
27:          MOV     DIGIT2, AL;cất vào biến DIGIT2
28: ;cộng 2 số
29:          CALL    ADDNOS
30: ;hiển thị kết quả
31:          LEA     DX, MSG1
32:          MOV     AH, 09
33:          INT     21H
34:          MOV     AH, 4CH ;về DOS
35:          INT     21H
36: MAIN      ENDP
37:          END     MAIN

```

Các chữ số và tổng của chúng chứa trong các biến DIGIT1, DIGIT2 và SUM được khai báo trong Mô đun thứ nhất. Tại dòng 2 chúng được khai báo PUBLIC để thủ tục ngoài ADDNOS có thể truy nhập chúng.

Chương trình nguồn PGM14_6A.ASM: Second module

```

0: TITLE    PGM14_6A: ADDNOS
1: EXTRN    DIGIT1:BYTE, DIGIT2:BYTE, SUM:BYTE
2: PUBLIC   ADDNOS
3: .MODEL   SMALL
4: .CODE
5: ADDNOS   PROC    NEAR
6: ;tính tổng 2 chữ số
7: ;Vào :các biến byte DIGIT1, DIGIT2 trong PGM1
8: ;Ra  :biến byte SUM trong PGM14_6
9:          PUSH    AX

```

```

10:      MOV      AL, DIGIT1
11:      ADD      AL, DIGIT2
12:      ADD      SUM, AL
13:      POP     AX
14:      RET
15: ADDNOS  ENDP
16:      END

```

Các biến DIGIT1, DIGIT2, SUM xuất hiện trong danh sách EXTRN của Mô đun thứ 2 (dòng 1). Thủ tục tiến hành cộng chúng (thực chất là cộng mã ASCII của các ký tự chữ số), sau đó cộng tổng với 30h chứa trong biến SUM. Thao tác này đưa mã ASCII của tổng vào trong biến SUM.

Ví dụ thực hiện:

```

C> PGM14_6
ENTER TWO DIGITS:26
THE SUM OF 2 AND 6 IS 8

```

14.5.2 Truyền địa chỉ của dữ liệu.

Một phương pháp thứ 2 để truyền dữ liệu cho thủ tục là truyền địa chỉ của dữ liệu. Phương pháp này còn được gọi là “call by reference” (tạm dịch gọi bằng cách tham chiếu), nó đặc biệt có ích khi làm việc với các mảng. Phương pháp này khác với phương pháp có tên gọi “call by value” (gọi bằng trị), trong đó giá trị chính bản thân giá trị của dữ liệu được truyền cho thủ tục được gọi. Cả 2 phương pháp có thể sử dụng trong cùng một thủ tục. Ví dụ thủ tục sắp xếp có chọn lựa viết trong phần 10.3 nhận địa chỉ của mảng cần sắp xếp trong SI (“call by reference”) và số phần tử trong BX (“call by value”).

Dưới đây là chương trình cộng 2 chữ số sử dụng phương pháp “call by reference”

Chương trình nguồn PGM14_7.ASM :First module

```

0: TITLE    PGM14_7: add digits
1: EXTRN    ADDNOS: NEAR
2: .MODEL   SMALL
3: .STACK   100H
4: .DATA
5: MSG      DB      'ENTER TWO DIGITS:$'
6: MSG1     DB      0DH, 0AH, 'THE SUM OF'
7: DIGIT1   DB      ?
8:          DB      ' AND'
9: DIGIT2   DB      ?

```

```

10:           DB      ' IS'
11: SUM        DB      '-30H,' $'
12: .CODE
13: MAIN       PROC
14: ;khởi tạo DS
15:           MOV     AX,@DATA
16:           MOV     DS,AX
17: ;thông báo cho người sử dụng
18:           MOV     AH,09      ;hàm hiển thị chuỗi
19:           LEA     DX,MSG    ;lấy địa chỉ chuỗi
20:           INT     21H      ;hiển thị
21: ;đọc vào 2 số
22:           MOV     AH,01      ;hàm đọc từ bàn phím
23:           INT     21H      ;ký tự biểu diễn số trong AL
24:           MOV     DIGIT1,AL;đặt vào biến DIGIT1
25:           INT     21H      ;ký tự biểu diễn số trong AL
26:           MOV     DIGIT2,AL  ;đặt vào biến DIGIT2
27: ;cộng 2 số
28:           LEA     SI,DIGIT1 ;SI chứa offset của DIGIT1
29:           LEA     DI,DIGIT2 ;DI chứa offset của DIGIT2
30:           LEA     BX,SUM    ;BX chứa offset của SUM
31:           CALL   ADDNOS
32: ;hiển thị kết quả
33:           LEA     DX,MSG1
34:           MOV     AH,09      ;hàm hiển thị chuỗi
35:           INT     21H      ;đưa ra kết quả
36:           MOV     AH,4CH    ;về DOS
37:           INT     21H
38: MAIN       ENDP
39:           END    MAIN

```

Chương trình nguồn PGM14_7A.ASM: Second module

```

0: TITLE    PGM14_7A: ADDNOS
1: PUBLIC   ADDNOS
2: .MODEL   SMALL
3: .CODE
4: ADDNOS  PROC    NEAR
5: ;tính tổng 2 chữ số
6: ;Vào :SI=địa chỉ của DIGIT1
7: ;          DI= địa chỉ của DIGIT2
8: ;          BX= địa chỉ của SUM
9: ;Ra   :[ BX] = tổng của DIGIT1 và DIGIT2
10:          PUSH   AX
11:          MOV    AL,[ SI]  ;AL chứa DIGIT1
12:          ADD    AL,[ DI]  ;AL chứa tổng của DIGIT1 và
                           ;DIGIT2
13:          ADD    [ BX],AL  ;Cộng vào SUM
14:          POP    AX

```

```

15:          RET
16: ADDNOS   ENDP
17:          END

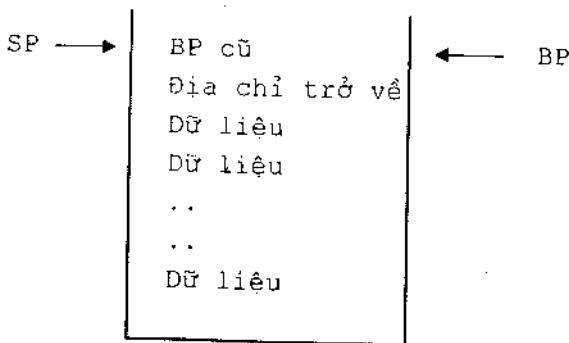
```

Trong dòng 11 và 12 thủ tục ADDNOS đã sử dụng chế độ địa chỉ gián tiếp để đưa tổng của 2 chữ số vào trong AL. Trong dòng 13, chế độ địa chỉ gián tiếp lại được sử dụng để cộng tổng tìm được với -30h trong biến SUM.

14.5.3 Sử dụng ngăn xếp.

Thay vì sử dụng thanh ghi, một thủ tục có thể đặt giá trị số liệu và các địa chỉ vào trong ngăn xếp trước khi gọi một thủ tục khác. Thủ tục được gọi khi đó sẽ sử dụng thanh ghi BP để truy nhập bằng phương pháp địa chỉ gián tiếp vào dữ liệu trong ngăn xếp (cần nhắc lại rằng khi BP được dùng trong chế độ địa chỉ gián tiếp thanh ghi thì SS chứa địa chỉ đoạn của toán hạng). Phương pháp này được dùng trong các ngôn ngữ bậc cao để truyền dữ liệu cho các thủ tục viết bằng hợp ngữ. Chúng ta cũng sẽ sử dụng phương pháp này để viết các thủ tục đệ quy (các thủ tục gọi lại chính nó).

Vì lệnh CALL trước tiên khiến cho địa chỉ trả về được đặt vào đỉnh ngăn xếp, thủ tục được gọi phải bắt đầu bằng việc cất BP vào ngăn xếp sau đó nó chuyển SP vào BP nhờ đó BP trả tới đỉnh ngăn xếp. Ngăn xếp lúc đó sẽ như sau:



Ngăn xếp

Bây giờ chúng ta có thể sử dụng BP với phương pháp địa chỉ gián tiếp để truy nhập dữ liệu (sở dĩ chúng ta phải sử dụng BP vì không thể sử dụng SP trong chế độ địa chỉ gián tiếp). Để trả về chương trình gọi, BP được lấy ra khỏi ngăn xếp và lệnh RET N được thực hiện với N là số byte dữ liệu mà thủ tục gọi đã cất vào ngăn xếp. Lệnh này khôi phục lại giá trị của CS:IP và chuyển N byte khỏi ngăn xếp nhờ đó đưa ngăn xếp trở về trạng thái ban đầu của nó.

Dưới đây là chương trình cộng 2 chữ số có sử dụng phương pháp trên:

Chương trình nguồn PGM14_8.ASM:First module

```
0: TITLE      PGM14_8:    ADD DIGITS
1: EXTRN      ADDNOS: NEAR
2: .MODEL     SMALL
3: .STACK     100H
4: .DATA
5: MSG        DB      'ENTER TWO DIGITS:$'
6: MSG1       DB      0DH, 0AH, 'THE SUM OF'
7: DIGIT1     DB      ?
8:           DB      'AND'
9: DIGIT2     DB      ?
10:          DB      ' IS'
11:SUM        DB      -30H,'$'
12:.CODE
13:MAIN       PROC
14:;khởi tạo DS
15:          MOV     AX, @DATA
16:          MOV     DS, AX
17:;thông báo cho người sử dụng
18:          MOV     AH, 09 ;hàm hiển thị chuỗi
19:          LEA     DX, MSG ;lấy địa chỉ chuỗi
20:          INT     21H ;hiển thị
21:;đọc vào 2 số
22:          MOV     AH, 01 ;hàm đọc từ bàn phím
23:          INT     21H ;ký tự biểu diễn số trong AL
24:          MOV     DIGIT1, AL;cắt vào biến DIGIT1
25:          PUSH   AX ;cắt vào ngăn xếp
26:          INT     21H ;ký tự biểu diễn số trong AL
27:          MOV     DIGIT2, AL;cắt vào biến DIGIT2
28:          PUSH   AX ;cắt vào ngăn xếp
29:;cộng 2 số
30:          CALL   ADDNOS;Ax chứa tổng
31:          ADD    SUM, AL ;lưu kết quả
32:;hiển thị kết quả
33:          LEA     DX, MSG1
34:          MOV     AH, 09 ;hàm hiển thị chuỗi
35:          INT     21H ;đưa ra kết quả
36:          MOV     AH, 4CH ;về DOS
37:          INT     21H
38:MAIN       ENDP
39:          END     MAIN
```

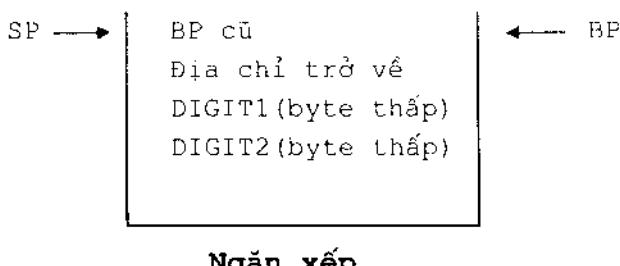
Tại dòng 24_28 hai số được đọc vào, chứa trong các biến nhớ và được cắt vào ngăn xếp (vì lệnh PUSH yêu cầu toán hạng phải có dạng word chúng ta phải viết PUSH AX). Tại dòng 30 thủ tục ADDNOS được gọi để thực hiện phép cộng

2 số, thủ tục này trả về tổng của 2 số trong AL và giá trị này lại được cộng vào với -30h chứa trong biến SUM.

Chương trình nguồn PGM14_8A.ASM: Second module

```
0: TITLE      PGM14_8A:    ADDNOS
1: PUBLIC     ADDNOS
2: .MODEL    SMALL
3: .CODE
4: ADDNOS    PROC    NEAR
5: ;tính tổng 2 chữ số
6; ngăn xếp lúc vào: địa chỉ trả về (đỉnh ngăn xếp),
7; chữ số thứ nhất, chữ số thứ 2
8; Ra: AX=tổng
9:          PUSH     BP      ;Cắt BP
10:         MOV      BP,SP   ;BP trả tới đỉnh ngăn xếp
11:         MOV      AX,[BP+6] ;AL chứa chữ số thứ nhất
12:         ADD      AX,[BP+4] ;AL chứa tổng
13:         POP      BP      ;Khôi phục lại giá trị của BP
14:         RET      4      ;Khôi phục lại ngăn xếp và trả
về
15: ADDNOS    ENDP
16: END
```

Tại dòng 9 ngăn xếp lúc đó sẽ như sau:



DIGIT1 và DIGIT2 là các byte thấp của các word trong ngăn xếp. Sau khi tiến hành cộng chúng ADDNOS đưa BP ra khỏi ngăn xếp và thực hiện lệnh RET 4, lệnh này bỏ ra 2 word dữ liệu từ ngăn xếp.

TỔNG KẾT

Trong chương này chúng ta đã nghiên cứu những vấn đề sau:

- ◆ Trong một chương trình .COM, toàn bộ ngắn xếp, dữ liệu và mã lệnh chứa gọn trong một đoạn duy nhất. Chương trình .COM chiếm ít chỗ trên đĩa hơn so với một chương trình .EXE tương ứng nhưng do ngắn xếp, lệnh và dữ liệu của nó phải chia chung trong một đoạn duy nhất nên sẽ làm hạn chế tính linh hoạt của chương trình.
- ◆ Có 2 dạng thủ tục:dạng NEAR và dạng FAR. Thủ tục NEAR là thủ tục nằm trong cùng đoạn mã với thủ tục gọi nó trong khi thủ tục FAR nằm ở đoạn khác. Khi thủ tục FAR được gọi cả CS và IP được lưu vào ngắn xếp.
- ◆ Toán tử giả EXTRN được dùng để thông báo cho trình biên dịch các thủ tục và biến được định nghĩa trong một Môđun hợp ngữ khác.
- ◆ Một thủ tục phải nằm trong một Môđun hợp ngữ có chứa ít nhất một định nghĩa đoạn, trình biên dịch sẽ dịch Môđun hợp ngữ thành một Môđun đối tượng(.OBJ) dạng ngôn ngữ máy.
- ◆ Toán tử giả PUBLIC được dùng để thông báo cho trình biên dịch biết những tên sẽ được tham chiếu đến từ một Môđun khác.
- ◆ Chương trình LINK kết nối tất cả các Môđun đối tượng thành một chương trình khả thi bằng ngôn ngữ máy. Chương trình này sẽ phối hợp các khai báo EXTRN trong các Môđun đối tượng với các khai báo PUBLIC trong các Môđun đối tượng khác.
- ◆ Chương trình LIB có thể sử dụng để tạo ra và duy trì một file chứa Môđun đối tượng.
- ◆ Dẫn hướng biên dịch SEGMENT có thể có các kiểu align, combine hay class.
- ◆ Kiểu align quyết định địa chỉ bắt đầu của các đoạn sẽ được chọn ra sao khi chương trình được nạp vào bộ nhớ.
- ◆ Kiểu combine quyết định các đoạn cùng tên được ghép lại như thế nào trong bộ nhớ.
- ◆ Nếu có 2 (hoặc nhiều hơn) đoạn có cùng kiểu class, chúng sẽ được nạp lần lượt vào bộ nhớ.
- ◆ Các thủ tục trong các Môđun khác nhau có thể liên lạc với nhau thông qua các biến toàn cục. Cũng có thể dùng các phương pháp khác như gọi bằng trị(call by value) và gọi bằng cách tham chiếu(call by reference). Các thủ tục gọi có thể sử dụng các phương pháp này bằng cách đặt dữ liệu, địa chỉ vào các thanh ghi hay đưa chúng vào ngắn xếp.

Các thuật ngữ tiếng Anh

| | |
|--------------------------|---|
| Assembly module | Mô đun hợp ngữ- file .ASM chứa ít nhất một định nghĩa đoạn. |
| Call by reference | Gọi bằng cách tham chiếu- phương pháp liên lạc với thủ tục bằng cách truyền cho nó chính dữ liệu mà nó cần. |
| Call by value | Gọi bằng trị- phương pháp liên lạc với một thủ tục bằng cách truyền cho nó địa chỉ các biến chứa dữ liệu mà thủ tục cần. |
| .COM program | Chương trình dạng .COM- chương trình mà tất cả các đoạn lệnh, dữ liệu và ngăn xếp được đặt chung trong một đoạn duy nhất. |
| Globalvariable | Biến toàn cục- biến được khai báo PUBLIC do đó có thể truy nhập đến từ các Mô đun khác. |
| Object module | Mô đun đối tượng- file .OBJ mà MASM tạo ra bằng cách hợp dịch một Mô đun hợp ngữ. |

Một số toán tử giả mới

| | | |
|--------|-------|---------|
| ASSUME | EXTRN | PUBLIC |
| .CONST | ORG | SEGMENT |
| .DATA | | |

Bài tập

1. Giả sử có một chương trình chứa những dòng sau:

```
CALL PROC1  
MOV AX, BX
```

và (a) lệnh MOV AX,BX được chứa tại địa chỉ 08FD:0200h, (b) PROC1 là một thủ tục FAR bắt đầu tại địa chỉ 1000:0200h, và (c) SP bằng 010Ah.

Hãy cho biết nội dung của CS, IP và SP ngay sau khi lệnh CALL PROC1 được thực hiện. Cho biết nội dung của word nằm tại đỉnh ngăn xếp.

2. Giả sử SP= 00FAh, CS=1000h, đỉnh ngăn xếp = 0200h, từ tiếp theo trong ngăn xếp= 08FDh. Hãy cho biết nội dung của CS, IP và SP trong các trường hợp sau:

- Sau khi lệnh RET trong một thủ tục dạng NEAR được thực hiện.
- Sau khi lệnh RET trong một thủ tục dạng FAR được thực hiện.
- Sau khi lệnh RET 4 trong một thủ tục dạng NEAR được thực hiện.

Các bài tập lập trình

3. Gia sử có một chương trình thực hiện những công việc sau:

- Thủ tục chính MAIN hiển thị thông báo "INSIDE MAIN PROGRAM", gọi thủ tục PROC1 và trở về DOS
- PROC1 hiển thị thông báo "INSIDE PROC1" tại ở dòng tiếp theo, gọi thủ tục PROC2 và trở về MAIN.
- PROC2 hiển thị thông báo "INSIDE PROC2" ở dòng tiếp theo và trở về PROC1

Hãy viết chương trình này theo các cách sau đây:

- a. Dạng .COM
- b. Dạng .EXE trong đó PROC1 và PROC2 là các thủ tục NEAR chứa trong các Mô đun được hợp dịch tách biệt. Mỗi Mô đun của thủ tục tương ứng chứa thông báo mà thủ tục cần hiển thị.
- c. Dạng .EXE trong đó PROC1 và PROC2 là các thủ tục FAR chứa trong các Mô đun được hợp dịch tách biệt. Mỗi Mô đun của thủ tục tương ứng chứa thông báo mà thủ tục cần hiển thị.
- d. Dạng .EXE trong đó các thông báo chứa trong Mô đun của thủ tục MAIN và được khai báo PUBLIC. Các thủ tục là các thủ tục NEAR chứa trong các Mô đun hợp dịch tách biệt, mỗi thủ tục tham chiếu đến thông báo tương ứng thông qua dẫn hướng biên dịch EXTRN.
- e. Dạng .EXE trong đó các thông báo chứa trong Mô đun của thủ tục MAIN. Các thủ tục PROC1 và PROC2 là các thủ tục NEAR được hợp dịch tách biệt. Trước khi gọi PROC1, MAIN đưa địa chỉ của các thông báo " INSIDE PROC1" và " INSIDE PROC2 " vào SI và DI một cách tương ứng.
- f. Dạng .EXE trong đó các thông báo chứa trong Mô đun của thủ tục MAIN. Các thủ tục PROC1 và PROC2 là các thủ tục NEAR được hợp dịch tách biệt. Trước khi gọi PROC1, MAIN đưa địa chỉ của các thông báo " INSIDE PROC1" và "INSIDE PROC2 " vào ngăn xếp.

4. Vị trí của một chuỗi con trong một chuỗi là số byte tính từ đầu chuỗi đến đầu chuỗi con. Hãy viết thủ tục hợp dịch tách biệt dạng NEAR có tên gọi FIND_SUBST nhận địa chỉ offset của chuỗi thứ nhất trong SI, chuỗi thứ 2 trong DI và xác định chuỗi thứ 2 có phải là chuỗi con của chuỗi thứ nhất hay không, nếu đúng như vậy, FIND_SUBST sẽ trả về vị trí của chuỗi con trong AX. Nếu chuỗi thứ 2 không phải là chuỗi con của chuỗi thứ nhất thủ tục sẽ trả về giá trị âm trong AX.

Hãy viết chương trình thử thủ tục FIND_SUBST, chương trình thử sẽ đọc vào 2 chuỗi, gọi FIND_SUBST và hiển thị kết quả. Đây là một dạng khác của PGM11_5.ASM.

Chương 15

CÁC NGẮT CỦA DOS VÀ BIOS.

Tổng quan

Trong các chương trước chúng ta đã sử dụng lệnh INT (interrupt) để gọi các chương trình của hệ thống. Đến chương này, chúng ta sẽ thảo luận về các loại ngắt khác nhau và xem xét một cách cụ thể các thao tác của lệnh INT. Trong các mục 15.2 và 15.3, chúng ta sẽ thảo luận về các phục vụ cung cấp bởi các chương trình phục vụ ngắt đa dạng của DOS và BIOS.

Để bạn hiểu rõ cách sử dụng các ngắt, chúng ta sẽ viết một chương trình hiển thị thời gian trên màn hình. Có tất cả ba version: version đầu tiên chỉ đơn giản hiển thị thời gian rồi kết thúc, version thứ hai hiển thị thời gian và cập nhật lại sau mỗi giây. Version thứ 3 là một chương trình thường trú bộ nhớ, chúng ta có thể gọi nó khi đang chạy một chương trình khác.

15.1 Phục vụ ngắt.

Ngắt cứng.

Về cơ bản các ngắt được hiểu là cho phép các thiết bị phần cứng ngắt các thao tác của CPU. Ví dụ khi ta nhấn một phím bất kỳ, 8086 phải được báo để đọc một mã phím vào bộ đệm bàn phím. Các ngắt cứng (hardware) nói chung xảy ra như sau: (1) thiết bị phần cứng cần phục vụ gửi tín hiệu yêu cầu ngắt (**interrupt request signal**) đến bộ xử lý; (2) 8086 treo công việc đang thực hiện và chuyển điều khiển cho chương trình phục vụ ngắt (**interrupt routine**); (3) chương trình phục vụ ngắt phục vụ ngắt cứng với việc thực hiện một vài thao tác vào ra; (4) điều khiển được trả về cho chương trình đang thi hành tại thời điểm bị treo.

Có một vài câu hỏi được đặt ra là 8086 nhận biết thiết bị yêu cầu ra sao ?
Bằng cách gì 8086 biết được chương trình phục vụ ngắt cần thi hành ? và nó tiếp tục nhiệm vụ trước như thế nào ?.

Bởi lẽ có thể có tín hiệu ngắt bất cứ lúc nào, 8086 kiểm tra chúng sau khi thi hành mỗi lệnh. Khi phát hiện thấy tín hiệu ngắt, 8086 chấp nhận nó bằng cách gửi đi tín hiệu chấp nhận ngắt (**interrupt acknowledge signal**). Thiết bị yêu cầu ngắt gửi trả lời bằng một số 8 bit trên bus địa chỉ gọi là số hiệu ngắt (**interrupt number**). Mỗi thiết bị sử dụng một số hiệu ngắt khác nhau để xác định phục vụ ngắt của nó. Phương thức xử lý bằng cách gửi các tín hiệu điều khiển qua lại gọi là móc nôi (**hand_shaking**), phương pháp này cần thiết dùng để xác định các thiết bị yêu cầu ngắt. Ta nói rằng ngắt N xảy ra khi một thiết bị sử dụng số hiệu ngắt N ngắt 8086.

Thực hiện một chương trình phục vụ ngắt tương tự như gọi một thủ tục. Trước khi chuyển điều khiển cho phục vụ ngắt, đầu tiên 8086 ghi địa chỉ của lệnh tiếp theo (địa chỉ trở về) vào ngăn xếp. 8086 còn cất thanh ghi cờ trong ngăn xếp để đảm bảo phục hồi lại trạng thái của chương trình bị ngắt. Các chương trình phục vụ ngắt còn phục hồi mọi thanh ghi khác mà chúng sử dụng.

Trước khi bàn luận về việc 8086 sử dụng số hiệu ngắt để xác định các phục vụ ngắt ra sao, chúng ta hãy xem xét một loại ngắt khác.

Ngắt mềm.

Ngắt mềm được các chương trình sử dụng để yêu cầu các phục vụ của hệ thống. Một ngắt mềm (**software interrupt**) xảy ra khi chương trình gọi một phục vụ ngắt bằng lệnh INT. Dạng của lệnh INT như sau:

INT interrupt_number

8086 coi số hiệu ngắt này cũng giống như số hiệu ngắt phát sinh bởi một thiết bị phần cứng. Chúng ta đã đưa ra một số thí dụ khi thực hiện công việc vào ra bằng INT 21H.

Ngắt cứng nội bộ

Có một loại ngắt thứ 3 được gọi là ngắt cứng nội bộ (**processor exception**). Một ngắt cứng nội bộ xảy ra khi có một tình huống này sinh bên trong bộ vi xử lý và yêu cầu có một xử lý đặc biệt chẳng hạn như xảy ra tràn khi chia. Mỗi trạng thái tương ứng với một loại ngắt duy nhất. Ví dụ, phép chia tràn tương ứng với ngắt 0 và như thế khi một phép chia bị tràn. 8086 tự động thi hành ngắt 0 xử lý tình huống tràn.

Tiếp theo chúng ta sẽ xem xét cách tính toán địa chỉ cho các chương trình phục vụ ngắt.

15.1.1 Véc tơ ngắt.

Các số hiệu ngắt của bộ xử lý 8086 là các số kiểu byte không dấu. Như vậy chúng có thể xác định được tổng cộng 256 ngắt. Không phải mọi số hiệu ngắt đều tương ứng với một phục vụ ngắt. Các nhà sản xuất máy tính cung cấp một số chương trình phục vụ ngắt các thiết bị phần cứng trong ROM, đó là các phục vụ ngắt của BIOS. Các phục vụ ngắt hệ thống bậc cao, chẳng hạn INT 21h, là một phần của DOS, được nạp vào bộ nhớ khi khởi động máy. Một vài số hiệu ngắt phụ được IBM dự trù cho các mục đích tương lai; các số hiệu ngắt tiếp theo có thể được dùng bởi người sử dụng. Xem bảng 15.1.

8086 không tạo ra địa chỉ của phục vụ ngắt trực tiếp từ số hiệu ngắt vì nếu như thế, mỗi phục vụ ngắt riêng biệt phải được đưa vào các vị trí giống hệt nhau trong mọi máy tính. Điều này không thể thực hiện được do số lượng rất lớn các loại máy tính và version của các chương trình luôn thay đổi. Thay vào đó, 8086 sử dụng số hiệu ngắt để tính toán địa chỉ của ô nhớ chứa địa chỉ thực của phục vụ ngắt. Điều này có nghĩa là phục vụ ngắt có thể ở bất cứ đâu khi mà địa chỉ của nó gọi là véc tơ ngắt (**interrupt vector**) được chứa trong một ô nhớ xác định từ số hiệu ngắt.

Tất cả các véc tơ ngắt được chứa trong một bảng chiếm 1 KB đầu tiên của bộ nhớ gọi là bảng véc tơ ngắt (**interrupt vector table**). Mỗi véc tơ ngắt có dạng segment:offset và chiếm 4 byte; 4 byte đầu tiên của bộ nhớ chứa véc tơ ngắt 0. Xem hình 15.1.

Để tìm một véc tơ cho một phục vụ ngắt, ta chỉ việc nhân số hiệu ngắt với 4. Kết quả cho biết vị trí ô nhớ chứa offset của phục vụ ngắt, địa chỉ đoạn của nó ở trong từ tiếp theo. Ví dụ với phục vụ ngắt bàn phím INT 9, địa chỉ offset của nó được chứa tại vị trí $9 \times 4 = 36 = 00024h$ và địa chỉ đoạn tại $00024h + 2 = 00026h$. BIOS khởi tạo các véc tơ ngắt của nó khi bật máy, các véc tơ ngắt của DOS được khởi tạo khi nạp DOS.

Hình 15.1 Bảng véc tơ ngắt.

| ĐỊA CHỈ | TÙ NHỚ |
|---------|--------------------|
| 003FE | segment của INT FF |
| 003FC | offset của INT FF |
| | |
| 00006 | segment của INT 1 |
| 00004 | offset của INT 1 |
| 00002 | segment của INT 0 |
| 00000 | offset của INT 0 |

Bảng 15.1 Các loại ngắt.

| | |
|-------------------|-------------------|
| Các ngắt 0-1Fh: | Các ngắt của BIOS |
| Các ngắt 20h-3Fh: | Các ngắt của DOS |
| Các ngắt 40h-7Fh: | Dự trù |
| Các ngắt 80h-F0h: | ROM BASIC |
| Các ngắt F1h-FFh: | Không sử dụng |

Các phục vụ ngắt.

Chúng ta hãy xem 8086 thi hành một lệnh INT như thế nào. Đầu tiên nó cất thanh ghi cờ vào ngăn xếp. Sau đó nó xoá các cờ điều khiển IF (interrupt flag) và TF (trap flag), lý do của các thao tác này sẽ giải thích sau. Tiếp theo, 8086 cất địa chỉ hiện tại bằng cách đưa giá trị của CS và IP vào ngăn xếp. Cuối cùng nó dùng số hiệu ngắt để lấy véc tơ ngắt từ bộ nhớ và chuyển điều khiển cho phục vụ ngắt bằng cách nạp véc tơ ngắt vào CS:IP. Cũng tương tự như vậy, 8086 trao điều khiển cho các phục vụ ngắt cứng hoặc cho phục vụ ngắt nội bộ theo cách này.

Khi thực hiện xong, các phục vụ ngắt thi hành lệnh IRET (interrupt return) phục hồi các thanh ghi IP, CS và thanh ghi cờ.

Các cờ điều khiển IF và TF.

Các cờ điều khiển IF và TF có vai trò rất quan trọng trong xử lý ngắt. Khi TF được lập, 8086 phát sinh ngắt loại 1 (ngắt cứng nội bộ). DEBUG sử dụng ngắt này khi thi hành lệnh T (trace). Để chạy từng bước một lệnh, đầu tiên DEBUG lập cờ TF rồi mới chuyển điều khiển cho lệnh đó. Sau khi lệnh được thi hành, bộ vi xử lý sẽ phát sinh một ngắt do TF được lập. DEBUG sử dụng chính phục vụ ngắt này để lấy quyền điều khiển từ bộ vi xử lý.

IF được dùng để điều khiển các ngắt phần cứng bên ngoài. Nếu như IF được thiết lập, các ngắt phần cứng có thể ngắt 8086. Khi xoá IF, các ngắt bên ngoài không còn tác dụng nữa (bị che). Thực ra vẫn có một ngắt cứng không che được, đó là NMI (NonMaskable Interrupt).

Trước khi bộ vi xử lý trao điều khiển cho một phục vụ ngắt nó xoá cả IF và TF, như vậy phục vụ ngắt đó sẽ không bị ngắt. Tất nhiên một phục vụ ngắt có thể đổi cờ để cho phép ngắt khi nó đang thi hành.

15.2 Các ngắt của BIOS.

Như đã chỉ ra trong bảng 15.1, các ngắt từ 0 đến 1Fh là các ngắt của BIOS. Đó là các phục vụ thường trú trong ROM, đoạn F000h.

Các ngắt 0-7.

Các ngắt 0-7 được Intel dành riêng trong đó các ngắt 0-4 đã được định nghĩa trước. IBM sử dụng ngắt 5 để in màn hình. Các ngắt 6 và 7 không được sử dụng.

Ngắt 0 - phép chia tràn. Một ngắt 0 được phát sinh khi có tràn trong thao tác DIV hay IDIV. Phục vụ ngắt này hiển thị dòng chữ "DIVIDE OVERFLOW" và trả điều khiển về cho DOS.

Ngắt 1 - chạy từng bước. Như đã thảo luận trong phần trên, một ngắt 1 phát sinh khi TF được thiết lập.

Ngắt 2 - ngắt không che được. Ngắt 2 là ngắt cứng không thể che được bằng cách xoá cờ IF. IBM PC sử dụng ngắt này để báo các lỗi bộ nhớ hay lỗi parity vào ra, chúng chỉ ra rằng có các chip hỏng.

Ngắt 3 - điểm gãy. Lệnh INT 3 là chỉ thị ngắt một byte duy nhất (mã lệnh CCh), các chỉ thị ngắt khác đều là các lệnh hai byte. Ta có thể chèn lệnh INT 3 vào bất cứ đâu trong chương trình bằng cách thay thế một mã lệnh tại đó bằng mã lệnh của nó. Chương trình DEBUG sử dụng tính chất này để thiết lập các điểm dừng cho lệnh G (go).

Ngắt 4 - tràn. Một ngắt 4 phát sinh bởi lệnh INTO (interrupt if overflow) khi OF được thiết lập. Các lập trình viên có thể viết chương trình phục vụ ngắt của chính mình để quản lý các khả năng tràn không mong muốn.

Ngắt 5 - in màn hình. Phục vụ ngắt 5 của BIOS gửi các thông tin của màn hình ra máy in. Một lệnh INT 5 phát sinh bởi phục vụ ngắt bàn phím (ngắt 9) khi ta nhấn phím PrtSc (print screen).

Các ngắt 8h - Fh.

8086 chỉ có chân nhận tín hiệu ngắt cứng. Để cho phép nhiều thiết bị hơn có thể ngắt 8086, IBM dùng một bộ điều khiển ngắt, chip Intel 8259, có thể mọc nối với 8 thiết bị. Các ngắt 8h-Fh phát sinh bởi các thiết bị phần cứng được nối với 8259. Các thế hệ đầu tiên của PC chỉ sử dụng các ngắt 8, 9 và Eh.

Ngắt 8 - thời gian. IBM PC có một mạch thời gian phát sinh một ngắt chu kỳ 54,92 milligiây (khoảng 18,2 lần một giây). Thường trình ngắt 8 của BIOS phục vụ cho mạch thời gian này. Nó dùng các tín hiệu thời gian (các nhịp) để duy trì thời gian trong ngày.

Ngắt 9 - bàn phím. Ngắt này được phát sinh bởi bàn phím mỗi khi có một phím được nhấn hay nhả ra. Phục vụ ngắt 9 của BIOS đọc mã scan và chứa nó trong bộ đệm bàn phím.

Ngắt E - lỗi đĩa mềm. Phục vụ ngắt Eh của BIOS quản lý các lỗi đĩa mềm.

Bảng 15.2. Kiểm tra thiết bị.

| | |
|-------|--|
| 15-14 | số máy in được cài đặt |
| 13 | =1 nếu modem nội bộ được cài đặt |
| 12 | =1 nếu bộ phôi ghép trò chơi được cài đặt |
| 11-9 | số các cổng RS-232 (nối tiếp) được cài đặt |
| 8 | không sử dụng |
| 7-6 | số lượng các ổ đĩa mềm (nếu bit 0 = 1) 00 - 1 ổ đĩa 01 - 2 ổ đĩa 10 - 3 ổ đĩa 11 - 4 ổ đĩa |
| 5-4 | chế độ màn hình khởi tạo 00 - không sử dụng 01 - văn bản màu 40x25 10 - văn bản màu 80x25 11 - văn bản đen trắng 80x25 |
| 3-2 | kích thước RAM hệ thống (với PC nguyên thuỷ) 00 - 16 KB 01 - 82 KB 10 - 48 KB 11 - 64 KB |
| 1 | =1 nếu bộ đồng xử lý toán học được cài đặt |
| 0 | =1 nếu có ổ đĩa mềm được cài đặt |

Các ngắt 10h - 1Fh.

Các phục vụ ngắt 10h-1Fh có thể được gọi bởi các chương trình ứng dụng để thực hiện các thao tác vào ra phức tạp hay kiểm tra các trạng thái.

Ngắt 10h - màn hình. Phục vụ ngắt 10h của BIOS là trình điều khiển màn hình. Nó được trình bày chi tiết trong các chương 12 và 16.

Ngắt 11h - kiểm tra thiết bị. Phục vụ ngắt 11h của BIOS trả về cấu hình thiết bị của các máy PC riêng biệt. Mã trả về được chứa trong AX. Bảng 15.2 cho biết ý nghĩa của các bit trả về trong AX.

Ngắt 12h - kích thước bộ nhớ. Phục vụ ngắt 12h của BIOS trả về trong AX tổng dung lượng bộ nhớ quy ước của máy tính. Bộ nhớ quy ước chỉ các mạch nhớ có địa chỉ nhỏ hơn 640K. Đơn vị của giá trị trả về tính bằng KB.

Ví dụ 15.1: Giả sử máy tính có 512 KB bộ nhớ quy ước. Hãy cho biết giá trị trả về trong AX nếu thi hành lệnh INT 12h.

Lời giải: $512 = 200h$, như vậy $AX = 200h$.

Ngắt 13h - ghi/dọc đĩa. Phục vụ ngắt 13h của BIOS là trình điều khiển ổ đĩa, nó cho phép các chương trình ứng dụng thực hiện các thao tác ghi đọc đĩa.

Ngắt 14h - liên lạc. Phục vụ ngắt 14h của BIOS là trình điều khiển liên lạc thông qua các cổng nối tiếp.

Ngắt 15h - cassette. Ngắt này dùng cho PC nguyên thuỷ trong việc giao tiếp với băng từ và dùng cho các phục vụ hệ thống khác nhau trong các máy PC AT và PS/2.

Ngắt 16h - vào ra bàn phím. Phục vụ ngắt 16h của BIOS là trình điều khiển bàn phím. Các thao tác với bàn phím đã được trình bày trong chương 12.

Ngắt 17h - vào ra máy in. Phục vụ ngắt 17h của BIOS là trình điều khiển máy in. Nó cung cấp ba hàm: 0-2. Hàm 0 đưa một ký tự ra máy in; các giá trị vào là AH=0, AL=ký tự, DX=số hiệu máy in. Hàm 1 khởi tạo một cổng máy in; các giá trị vào là AH=1, DX=số hiệu máy in. Hàm 2 lấy trạng thái máy in; các giá trị vào là AH=2, DX=số hiệu máy in. Với cả ba hàm trên, trạng thái được trả về trong AH. Bảng 15.3 chỉ ra ý nghĩa của các bit trả về trong AH.

Bảng 15.3. Trạng thái máy in.

| BIT TRONG AH | Ý NGHĨA |
|--------------|-------------------------|
| 7 | =1 máy in không bận |
| 6 | =1 chấp nhận in |
| 5 | =1 không có giấy |
| 4 | =1 máy in được chọn |
| 3 | =1 lỗi vào ra |
| 2 | không sử dụng |
| 1 | không sử dụng |
| 0 | =1 máy in quá thời gian |

Ví dụ 15.2. Viết các lệnh in ra một số 0.

Lời giải: Chúng ta sử dụng hàm 0 để in. Bởi lẽ các máy in có chứa các bộ đếm dữ liệu, số 0 sẽ không được in ra chừng nào một ký tự về đầu dòng và xuống dòng chưa được gửi đến. Ta có đoạn lệnh:

```
MOV AH, 0          ;hàm 0,in ký tự
MOV AL, '0'        ;ký tự 0
MOV DX, 0          ;máy in 0
INT 17H            ;AH chứa mã trả về
MOV AH, 0          ;hàm 0,in ký tự
MOV AL, 0AH         ;xuống dòng
INT 17H
```

Ngắt 18h - BASIC. Phục vụ ngắt 18h của BIOS chuyển điều khiển cho ROM BASIC.

Ngắt 19h - khởi động hệ thống. Phục vụ ngắt 19h của BIOS khởi động lại hệ thống.

Ngắt 1Ah - thời gian trong ngày. Phục vụ ngắt 1Ah của BIOS cho phép chương trình lấy và thiết lập giá trị bộ đếm nhịp thời gian. Trong trường hợp là máy PC AT hay PS/2, ngắt này còn cho phép chương trình lấy và thiết lập thời gian và ngày tháng cho vi mạch đồng hồ.

Ngắt 1Bh - Ctrl-Break. Ngắt này được gọi bởi phục vụ ngắt 9 khi ta nhấn phím Ctrl-Break. Phục vụ ngắt 1Bh của BIOS chỉ chứa một lệnh IRET. Người sử dụng có thể viết chương trình của chính mình để quản lý phím Ctrl-Break.

Ngắt 1Ch - nhịp thời gian. INT 1Ch được gọi bởi phục vụ ngắt INT 8 mỗi lần vi mạch thời gian ngắt 8086. Phục vụ ngắt 1Ch của BIOS chỉ chứa một lệnh IRET. Người sử dụng có thể viết chương trình phục vụ ngắt để thực hiện các thao tác định thời. Trong phần 15.5 chúng tôi sẽ sử dụng nó để cập nhật và hiển thị thời gian.

Các ngắt 1Dh-1Fh. Các véc tơ ngắt này chỉ đến dữ liệu thay vì chỉ đến các lệnh. Các véc tơ ngắt 1Dh, 1Eh và 1Fh tương ứng chỉ đến các tham số khởi tạo màn hình, các tham số đĩa mềm và các ký tự đồ họa màn hình.

15.3 Các ngắt của DOS.

Các phục vụ **ngắt 20-3Fh** của DOS cung cấp các phục vụ bậc cao cho phần cứng cũng như các tài nguyên phần mềm như các file và các thư mục. Trong đó hữu ích nhất là ngắt 21h, nó cung cấp rất nhiều chức năng làm việc với bàn phím, màn hình và các thao tác file.

Ngắt 20h - kết thúc chương trình. Một chương trình có thể sử dụng ngắt 20h để trả điều khiển về cho DOS. Bởi vì CS phải được thiết lập trở đến vùng nháp chương trình trước khi sử dụng INT 20h nên tiện lợi hơn chúng ta thường thoát khỏi chương trình bằng hàm 4CH, ngắt 21H.

Ngắt 21h - gọi các hàm. Số lượng các hàm khác nhau nhiều theo các version của DOS. DOS 1.x có các hàm 0-2Eh. DOS 2.x đưa vào thêm các hàm 2Fh-57h và DOS 3.x tiếp tục có thêm các hàm mới 58h-5Fh. Các hàm này có thể được phân loại thành các hàm vào ra ký tự, truy nhập file, quản lý bộ nhớ, truy nhập đĩa, mạng và nhiều loại hàm khác nữa. Để biết chi tiết hơn bạn hãy xem phụ lục C.

Các ngắt 22h-26h. Các phục vụ ngắt 22h-26h quản lý Ctrl-Break, các lỗi nghiêm trọng (critical errors) và truy nhập trực tiếp đĩa.

Ngắt 27h - kết thúc chương trình và ở lại thường trú. Ngắt 27h cho phép chương trình ở lại thường trú trong bộ nhớ sau khi kết thúc. Chúng ta sẽ đề cập đến nó trong phần 15.6.

15.4. Một chương trình hiển thị thời gian.

Để làm ví dụ sử dụng các phục vụ ngắt, chúng ta sẽ viết một chương trình hiển thị thời gian hiện thời. Có tất cả ba version, chúng được viết phức tạp dần. Trong phần này chúng tôi sẽ chỉ ra version đầu, nó chỉ đơn thuần hiển thị thời gian hiện tại dưới dạng giờ, phút, giây. Trong phần 15.5 chúng ta sẽ viết version thứ hai, nó hiển thị thời gian và cập nhật lại sau mỗi giây. Version thứ ba viết trong phần 15.6 là một chương trình thường trú bộ nhớ, nó có thể hiển thị thời gian khi đang chạy một chương trình khác.

Khi bật máy, thời gian hiện tại có thể được người sử dụng đưa vào hay lấy từ mạch đồng hồ thời gian thực được nuôi bằng nguồn pin. Giá trị thời gian này

được giữ trong bộ nhớ và được cập nhật bởi một mạch thời gian sử dụng ngắt 8. Một chương trình có thể gọi ngắt 21h, hàm 2Ch để truy nhập thời gian.

Ngắt 21h, hàm 2Ch:

Thời gian trong ngày.

| | |
|------|--------------------------|
| Vào: | AH = 2Ch |
| Ra: | CH = giờ (0-23) |
| | CL = phút (0-59) |
| | DH = giây (0-59) |
| | DL = 1/100 giây (0-99) |

Chương trình hiển thị thời gian của chúng ta bao gồm các bước sau đây: (1) lấy thời gian hiện tại; (2) đổi giờ, phút, giây sang dạng chữ số ASCII (chúng ta bỏ qua phần lẻ của giây) và (3) hiển thị các chữ số ASCII.

Chương trình được tổ chức thành một chương trình chính trong program listing PGM15_1.ASM và hai chương trình con GET_TIME và CONVERT trong program listing PGM15_1A.ASM.

Một bộ đếm thời gian, TIME_BUF, được khởi tạo với chuỗi 00:00:00. Đầu tiên thủ tục MAIN gọi GET_TIME cắt thời gian hiện tại vào trong bộ đếm thời gian. Sau đó nó dùng hàm 9 của ngắt 21h in ra chuỗi trong bộ đếm thời gian.

Thủ tục GET_TIME gọi ngắt 21h, hàm 2Ch để lấy giá trị thời gian, tiếp theo nó gọi CONVERT để đổi giờ, phút và giây sang các ký tự ASCII. Bước đầu tiên của thủ tục CONVERT là chia giá trị vào trong AL cho 10. Thao tác này sẽ nhận được chữ số hàng chục trong AL và chữ số hàng đơn vị trong AH (lưu ý rằng giá trị vào nhỏ hơn 60). Bước thứ hai đổi các chữ số sang dạng ASCII.

Program listing PGM15_1.A SM

```
TITLE PGM15_1:TIME_DISPLAY_VER_1
; lập trình hiển thị thời gian hiện tại
;
EXTRN    GET_TIME:NEAR
.MODEL   SMALL
.STACK  100h
.DATA
TIME_BUF DB '00:00:00$'           ;bộ đếm thời gian
;hr:min:sec
.CODE
MAIN     PROC
```

```

        MOV      AX, @DATA
        MOV      DX, AX          ;khởi tạo DS
;lấy và hiển thị thời gian
        LEA      BX, TIME_BUF    ;BX trỏ đến TIME_BUF
        CALL    GET_TIME         ;đưa thời gian hiện tại
                                ;vào DX
        LEA      DX, TIME_BUF    ;DX trỏ đến TIME_BUF
        MOV      AH, 09H          ;hiển thị thời gian
        INT      21H
;trở về DOS
        MOV      AH, 4CH
        INT      21H          ;trở về DOS
MAIN     ENDP
END      MAIN

```

Program listing PGM15_1A.ASM

```

TITLE PGM15_1A:GET AND CONVERT TIME TO ASCII
PUBLIC   GET_TIME
.MODEL   SMALL
.CODE
GET_TIME PROC
;lấy thời gian trong ngày và chứa các chữ số ASCII
;trong bộ đệm thời gian
;vào: BX=địa chỉ bộ đệm thời gian
        MOV      AH, 2CH          ;lấy thời gian
        INT      21H          ;CH=hr, CL=min, DH=sec
;đổi giờ sang dạng ASCII và lưu trữ
        MOV      AL, CH          ;giờ
        CALL    CONVERT          ;đổi sang dạng ASCII
        MOV      [ BX], AX        ;cắt
;đổi phút sang dạng ASCII và lưu trữ
        MOV      AL, CL          ;phút
        CALL    CONVERT          ;đổi sang dạng ASCII
        MOV      [ BX+3], AX; cắt
;đổi giây sang dạng ASCII và lưu trữ
        MOV      AL, DH          ;giây
        CALL    CONVERT          ;đổi sang dạng ASCII
        MOV      [ BX+6], AX; cắt
        RET
GET_TIME ENDP
;
CONVERT PROC
;đổi số kiểu byte (0-59) sang các chữ số ASCII
;vào: AL = số
;ra : AX = các chữ số ASCII, AL=chữ số hàng chục
;AH=chữ số hàng đ.v
        MOV      AH, 0          ;xoá AH
        MOV      DL, 10         ;chia AX cho 10

```

```

        DIV      DL      ;AH chứa số dư, AL chứa thương
        OR       AX, 3030H   ;đổi sang dạng ASCII
        RET
CONVERT END
;
END

```

Chương trình hiển thị thời gian và kết thúc.

15.5. Các thủ tục ngắt của người sử dụng.

Để tạo ra một chương trình hiển thị thời gian thú vị hơn chúng ta hãy viết version thứ hai, hiển thị thời gian và cập nhật lại sau mỗi giây.

Có một phương pháp cập nhật thời gian một cách liên tục, đó là thi hành vòng lặp bao gồm lấy thời gian bằng ngắt 21h, hàm 2Ch và hiển thị nó. Vấn đề đặt ra là phải tìm cách để kết thúc chương trình.

Thay vì phương pháp trên chúng ta sẽ viết một chương trình mới cho ngắt 1Ch. Như đã nhắc tới ở trên, ngắt này phát sinh bởi INT 8, ngắt được mạch thời gian kích hoạt khoảng 18,2 lần mỗi giây. Khi phục vụ ngắt của chúng ta được gọi, nó sẽ truy nhập và hiển thị thời gian.

Chương trình của chúng ta sẽ chứa thủ tục MAIN thiết lập phục vụ ngắt mới và khi một phím được nhấn, nó trả lại phục vụ ngắt ban đầu và kết thúc.

Thiết lập vector ngắt.

Để thiết lập một chương trình phục vụ ngắt mới, chúng ta cần phải; (1) cất vector ngắt hiện tại; (2) đưa vector của chương trình con phục vụ ngắt của người sử dụng vào bảng vector ngắt và (3) phục hồi lại vector cũ trước khi kết thúc chương trình.

Chúng ta dùng ngắt 21h, hàm 35h để lấy vector cũ và hàm 25h để thiết lập vector ngắt mới.

Ngắt 21h, hàm 25h:

Thiết lập vector ngắt.

;đưa vector ngắt vào trong bảng vector

| | |
|------|---------------------|
| Vào: | AH = 25h |
| | AL = số hiệu ngắt |
| | DS:DX = vector ngắt |
| Ra: | không |

Ngắt 21h, hàm 35h:

Lấy vector ngắt.

;nhận vector ngắt từ bảng vector.

| | |
|------|-----------------------------|
| Vào: | AH = 35h AL=số hiệu ngắt |
|------|-----------------------------|

| | |
|-----|---------------------|
| Ra: | ES:BX = vector ngắt |
|-----|---------------------|

Thủ tục SETUP_INT trong program listing PGM15_2ASM cắt vector cũ và thiết lập vector ngắt mới. Nó nhận số hiệu ngắt trong AL, bộ đệm để cắt vector cũ tại DS:DI và bộ đệm chứa vector ngắt mới tại DS:SI. Bằng việc hoán đổi hai bộ đệm, SETUP_INT có thể được dùng để phục hồi lại vector cũ.

Program listing PGM15_2A.ASM

```

TITLE PGM15_2A:SET INTERRUPT VECTOR
PUBLIC SETUP_INT
.MODEL SMALL
.CODE
SETUP_INT PROC
;cắt vector cũ và thiết lập vector mới
;vào:AL = số hiệu ngắt
;        DI = địa chỉ bộ đệm vector cũ
;        SI = địa chỉ bộ đệm chứa vector mới
;cắt vector cũ
    MOV AH,35H      ;hàm 35h,lấy vector ngắt
    INT 21H          ;ES:BX = vector
    MOV [DI],BX      ;cắt offset
    MOV [DI+2],ES;cắt địa chỉ đoạn
;thiết lập vector mới
    MOV DX,[SI]      ;DX chứa offset
    PUSH DS          ;cắt DS
    MOV DS,[SI+2];DS chứa số hiệu đoạn
    MOV AH,25H      ;hàm 25h,thiết lập vector ngắt
    INT 21H
    POP DS          ;phục hồi DS
    RET
SETUP_INT END
END

```

Điều khiển con trỏ.

Mỗi khi hiển thị thời gian bằng hàm 9 ngắt 21h, con trỏ bị dịch đi. Lần hiển thị sau sẽ xuất hiện tại một vị trí khác trên màn hình. Như vậy để thấy thời gian cập nhật lại ở cùng một vị trí trên màn hình, chúng ta phải phục hồi con trỏ tại vị trí nguyên thuỷ của nó trước khi hiển thị thời gian. Muốn thực hiện điều này, trước hết chúng ta phải xác định vị trí hiện tại của con trỏ, tiếp đó sau mỗi thao tác in chuỗi chúng ta lại chuyển con trỏ trở về vị trí ban đầu của nó.

. Chúng ta sẽ sử dụng các hàm 3 và 2 của ngắt 10h để cất vị trí của con trỏ và di chuyển con trỏ đến vị trí ban đầu của nó sau mỗi thao tác in chuỗi.

Ngắt 10h, hàm 2:

Lấy vị trí con trỏ.

| | |
|------|--------------------|
| Vào: | AH = 2 |
| | BH = số hiệu trang |
| | DH = số hiệu dòng |
| | DL = số hiệu cột |
| Ra: | không |

Ngắt 10h, hàm 3:

Đặt vị trí con trỏ.

| | |
|------|---------------------------------|
| Vào: | AH = 3 |
| | BH = số hiệu trang |
| Ra: | DH = số hiệu dòng |
| | DL = số hiệu cột |
| | CH = dòng quét đầu của con trỏ |
| | CL = dòng quét cuối của con trỏ |

Thủ tục ngắt.

Khi một thủ tục ngắt được kích hoạt, nó không thể xem rằng thanh ghi DS chứa địa chỉ đoạn dữ liệu của chương trình. Vì thế nếu muốn sử dụng bất cứ biến nào nó phải khởi tạo thanh ghi DS. Thanh ghi DS phải được phục hồi lại trước khi kết thúc chương trình con phục vụ ngắt với lệnh IRET.

Program listing PGM15_2.ASM chứa thủ tục MAIN và thủ tục ngắt TIME_INT. Các bước của thủ tục MAIN gồm (1) cất vị trí con trỏ hiện tại, (2)

thiết lập vector ngắt cho TIME_INT, (3) đợi vào một phím và (4) phục hồi lại vector cũ và kết thúc.

Thực hiện bước 2, chúng ta sử dụng các toán tử giả OFFSET và SEG để lấy offset và địa chỉ đoạn của thủ tục TIME_INT, vector này sau đó được lưu giữ trong bộ đệm NEW_VEC. Thủ tục SETUP_INT được gọi để thiết lập vector cho ngắt nhịp thời gian 1Ch. Ngắt 16h, hàm 0 được dùng trong bước 3 để vào một phím. Thủ tục SETUP_INT lại được gọi một lần nữa trong bước 4, lần này SI trả đến vector cũ và DI trả đến vector của thủ tục TIME_INT.

Các bước của thủ tục TIME_INT gồm: (1) thiết lập DS, (2)lấy thời gian mới, (3) hiển thị thời gian, (4) phục hồi vị trí con trỏ và (5) phục hồi DS.

Chương trình được tiến hành như sau: Sau khi thiết lập con trỏ và các vector ngắt, thủ tục MAIN đợi một phím nhấn. Khi ấy thủ tục ngắt TIME_INT cập nhật thời gian mới mỗi khi có xung nhịp của mạch thời gian. Sau khi một phím được nhấn, vector cũ được phục hồi và chương trình kết thúc.

Program listing PGM15_2.ASM

```
TITLE PGM15_2:DISPLAY_TIME_VER_2
;Lập trình hiển thị thời gian hiện tại
;và cập nhật thời gian 18.2 lân mỗi giây
    EXTRN GET_TIME:NEAR, SETUP_INT:NEAR
.MODEL SMALL
.STACK 100h
.DATA
TIME_BUF DB '00:00:00$' ;bộ đệm thời gian
;hr:min:sec
CURSOR_POS DW ? ;vị trí con trỏ(row:col)
NEW_VEC DW ?,? ;vec tor ngắt mới
OLD_VEC DW ?,? ;vec tor cũ
;
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX ;khởi tạo DS
;
;cắt vị trí con trỏ
    MOV AH, 3 ;hàm 3,lấy vị trí con trỏ
    MOV BH, 0 ;trang 0
    INT 10H ;DH=dòng, DL=cột
    MOV CURSOR_POS, DX ;cắt vị trí con trỏ
;thiết lập thủ tục ngắt bằng cách
;đưa segment:offset của TIME_INT vào NEW_VEC
    MOV NEW_VEC, OFFSET TIME_INT ;offset
    MOV NEW_VEC+2, SEG TIME_INT ;segment
    LEA DI, OLD_VEC ;DI trả đến bộ đệm vector cũ
```

```

        LEA      SI, NEW_VEC      ;SI trả đến vector mới
        MOV      AL, 1Ch          ;ngắt thời gian
        CALL    SETUP_INT;thiết lập vector mới
;đọc bàn phím
        MOV      AH, 0
        INT      16H
;phục hồi vector ngắt cũ
        LEA      DI, NEW_VEC    ;DI trả đến bộ đệm vector
        LEA      SI, OLD_VEC    ;SI trả đến vector cũ
        MOV      AL, 1Ch          ;ngắt thời gian
        CALL    SETUP_INT;phục hồi vector cũ
;trở về DOS
        MOV      AH, 4CH
        INT      21H              ;trở về DOS
MAIN     ENDP
;
TIME_INT PROC
;thủ tục ngắt
;kích hoạt bởi mạch thời gian
        PUSH    DS ;cất giá trị tức thời của DS
        MOV     AX, @DATA
        MOV     DS, AX      ;DS trả đến đoạn dữ liệu
;lấy thời gian mới
        LEA      BX, TIME_BUF   ;BX trả đến bộ đệm
                           ;thời gian
        CALL    GET_TIME ;cất thời gian trong bộ đệm
;hiển thị thời gian
        LEA      DX, TIME_BUF   ;DX trả đến TIME_BUF
        MOV     AH, 09H          ;hiển thị chuỗi
        INT      21H
;phục hồi vị trí con trỏ
        MOV     AH, 2      ;hàm 2, di chuyển con trỏ
        MOV     BH, 0      ;trang 0
        MOV     DX, CURSOR_POS ;vị trí con trỏ,
                           ;DH=hàng, DL=cột
        INT      10H
        POP     DS           ;phục hồi DS
        IRET
TIME_INT ENDP          ;kết thúc thủ tục ngắt
;
END      MAIN

```

Lệnh LINK gồm có các modul PGM15_2, PGM15_1A và PGM15_2A.

15.6. Chương trình thường trú bộ nhớ.

Chúng ta sẽ viết version thứ ba của DISPLAY_TIME như một chương trình kết thúc và ở lại thường trú TSR (terminate and stay resident). Thông thường khi kết thúc một chương trình, bộ nhớ mà nó chiếm giữ được DOS sử dụng để nạp chương trình khác. Tuy nhiên khi một chương trình TSR kết thúc, bộ nhớ không bị thu hồi lại. Chính vì thế một chương trình TSR còn được gọi là chương trình thường trú bộ nhớ (memory resident program).

Để trở về DOS, một chương trình TSR kết thúc bằng cách sử dụng ngắt 27h hay ngắt 21h hàm 31h. Chương trình của chúng ta sẽ dùng INT 27h.

| Ngắt 27h: | |
|------------------------------|---|
| Kết thúc và ở lại thường trú | |
| Vào: | DS:DX = địa chỉ byte sau vùng được giữ lại thường trú |
| Ra: | không |

Chúng ta sẽ viết một chương trình .COM bởi một lẽ là để sử dụng ngắt 27h chúng ta cần phải xác định số byte được ở lại thường trú bộ nhớ. Cấu trúc của một chương trình .COM khiến việc tính toán dễ dàng hơn bởi vì nó chỉ có một đoạn chương trình. Một lý do khác của việc sử dụng chương trình .COM là vì kích thước của nó. Như đã thấy trong chương 14, một chương trình .COM có kích thước nhỏ hơn chương trình EXE tương đương. Vì thế để tiết kiệm bộ nhớ, các chương trình TSR thường được viết dạng .COM.

Một khi đã kết thúc, chương trình TSR không hoạt động. Nó phải được kích hoạt bởi các tác động ngoài chẳng hạn một tổ hợp phím nào đó hay mạch thời gian. Ưu điểm của chương trình thường trú là chúng có thể kích hoạt khi đang chạy một chương trình khác. Chương trình của chúng ta sẽ được kích hoạt khi nhấn đồng thời các phím Ctrl và Shift phải.

Để chương trình gọn nhẹ, chúng tôi sẽ không cập nhật thời gian. Viết một chương trình TSR cập nhật lại thời gian sau mỗi giây coi như bài tập dành cho bạn đọc.

Chương trình gồm có hai phần, phần khởi tạo thiết lập vector ngắt và phục vụ ngắt. Thủ tục INITIALIZE khởi tạo vector ngắt 9 với địa chỉ của thủ tục ngắt MAIN và sau đó gọi INT 27h để kết thúc. Địa chỉ truyền cho INT 27h là địa chỉ bắt đầu của thủ tục INITIALIE vì thủ tục INITIALIZE được chỉ ra trong program listing PGM15_3.ASM.

Program listing PGM15_3A.ASM

```

TITLE PGM15_3A:SET UP TSR PROGRAM
        EXTRN  MAIN:NEAR,SETUP_INT:NEAR
        EXTRN  NEW_VEC:DWORD,OLD_VEC:DWORD
PUBLIC  INITIALIZE
C_SEG   SEGMENT PUBLIC
        ASSUME CS:C_SEG
INITIALIZE      PROC
;thiết lập vector ngắt
        MOV     NEW_VEC,OFFSET MAIN ;lưu giữ địa chỉ
        MOV     NEW_VEC+2,CS    ;Segment
        LEA     DI,OLD_VEC    ;DI trả đến vector cũ
        LEA     SI,NEW_VEC    ;SI trả đến vector mới
        MOV     AL,09H         ;ngắt bàn phím
        CALL    SETUP_INT;thiết lập vector ngắt
;trở về DOS
        LEA     DX,INITIALIZE
        INT    27H           ;kết thúc và ở lại thường trú
INITIALIZE      ENDP
C_SEG   ENDS
        END    INITIALIZE

```

Có rất nhiều cách để các chương trình nhận biết một tổ hợp phím nhất định. Cách đơn giản nhất là nhận biết các phím điều khiển và dịch bằng việc kiểm tra các cờ bàn phím. Khi được kích hoạt bằng một phím nhấn, thường trình ngắt gọi phục vụ ngắt bàn phím cũ để quản lý việc nhập phím. Để nhận biết các phím điều khiển và dịch, chương trình có thể kiểm tra các cờ bàn phím tại vùng dữ liệu của BIOS 0000:0417h hay sử dụng ngắt 16h, hàm 2.

Ngắt 16h, hàm 2:

Lấy các cờ bàn phím.

| | |
|------------|-----------------------|
| Vào: | AH = 2 |
| Ra: | AL = cờ các phím. |
| <u>bit</u> | <u>ý nghĩa</u> |
| 7 = 1 | Insert on |
| 6 = 1 | Caps Lock on |
| 5 = 1 | Num Lock on |
| 4 = 1 | Scroll Lock on |
| 3 = 1 | Phím Alt được nhấn |
| 2 = 1 | Phím Ctrl được nhấn |
| 1 = 1 | Left Shift được nhấn |
| 0 = 1 | Right Shift được nhấn |

Chúng ta sẽ sử dụng tổ hợp phím Ctrl và Shift phải để kích hoạt và kết thúc sự hiển thị đồng hồ. Khi kích hoạt, thời gian hiện tại sẽ được hiển thị tại góc phải trên của màn hình. Đầu tiên chúng ta phải chép lại dữ liệu màn hình để sau khi kết thúc hiển thị đồng hồ ta có thể phục hồi lại màn hình cũ.

Thủ tục SET_CURSOR thiết lập vị trí con trỏ tại hàng 0 và cột trong DL. Thủ tục SAVE_SCREEN chép dữ liệu màn hình vào bộ đệm SS_BUF, thủ tục RESTOR_SCREEN trả lại dữ liệu cho bộ đệm màn hình. Cả ba thủ tục được chỉ ra trong program listing PGM15_3B.

Program listing PGM15_3B.ASM

```

TITLE PGM15_3B: SAVE SCREEN AND CURSOR
EXTRN SS_BUF:BYTE
PUBLIC SAVE_SCREEN,RESTORE_SCREEN,SET_CURSOR
C SEGMENT PUBLIC
    ASSUME CS:C SEG
;
SAVE_SCREEN    PROC
;lưu giữ 8 ký tự ở góc phải trên của màn hình
    LEA    DI,SS_BUF;bộ đệm màn hình
    MOV    CX,8      ;lặp 8 lần
    MOV    DL,72     ;cột 72
    CLD              ;xoá DF cho thao tác chuỗi
SS_LOOP:
    CALL SET_CURSOR ;thiết lập con trỏ tại
                    ;hàng 0,cột DL
    MOV    AH,08h    ;đọc ký tự trên màn hình
    INT    10H       ;AH=thuộc tính,AL=ký tự
    STOSW           ;cất ký tự và thuộc tính
    INC    DL        ;cột tiếp theo
    LOOP   SS_LOOP
    RET
SAVE_SCREEN    ENDP
;
RESTORE_SCREEN PROC
;phục hồi màn hình đã cất
    LEA    SI,SS_BUF;SI trỏ đến bộ đệm
    MOV    DI,8      ;lặp 8 lần
    MOV    DL,72     ;cột 72
    MOV    CX,1      ;mỗi lần một ký tự
RS_LOOP:
    CALL  SET_CURSOR ;di chuyển con trỏ
                    ;AL=ký tự,AH=thuộc tính
    LODSW           ;thuộc tính trong BL
    MOV    BL,AH
    MOV    AH,09H    ;hàm 9,viết ký tự và thuộc tính
    MOV    BH,0      ;trang 0
    INT    10H

```

```

        INC      DL      ;vị trí ký tự tiếp theo
        DEC      DI      ;đủ số lần chưa?
        JG       RS_LOOP ;chưa, lặp lại
        RET
RESTOR_SCREEN    END
;
SET_CURSOR        PROC
;chuyển con trỏ đến hàng 0 ,cột DL
;vào DL=số hiệu cột
        MOV      AH,02    ;hàm 2, thiết lập con trỏ
        MOV      BH,0     ;trang 0
        MOV      DH,0     ;cột 0
        INT      10H
        RET
SET_CURSOR        END
;
C_SEG      ENDS
END

```

Bây giờ chúng ta đã sẵn sàng để viết chương trình ngắn. Để xác định xem sự hiển thị thời gian có được kích hoạt hay không ta sử dụng cờ biến đổi được ON_FLAG. Cờ này được thiết lập bằng 1 khi thời gian được hiển thị. MAIN là thủ tục ngắn, các bước của nó gồm có: (1) cất tất cả các thanh ghi sẽ sử dụng và thiết lập các thanh ghi DS và ES, (2) gọi chương trình ngắn bàn phím cũ để quản lý việc nhập phím, (3) kiểm tra xem cả Ctrl và Shift phải có được nhấn không, nếu không: thoát ra, (4) kiểm tra cờ ON_FLAG để xác định trạng thái, nếu ON_FLAG bằng 1, phục hồi lại màn hình và thoát, (5) cất vị trí con trỏ hiện tại đồng thời hiển thị thông tin màn hình và (6) lấy và hiển thị thời gian rồi thoát.

Trong bước 1 chúng ta sử dụng CS để thiết lập các thanh ghi DS và ES. Hắn các bạn nghĩ rằng có thể dùng C_SEG thay cho CS, thực ra không thể sử dụng giá trị đoạn trong một chương trình .COM. Trong bước 2 ta cần phải cất thanh ghi cờ để lời gọi thủ tục giống như một lời gọi ngắn. Trong bước 6 chúng ta sử dụng ngắn 10h của BIOS thay vì ngắn 21h, hàm 9 để hiển thị thời gian bởi lẽ kinh nghiệm cho thấy ngắn 21h, hàm 9 tỏ ra không được tin cậy cho lắm trong một chương trình TSR.

```

Program listing PGM15_3.ASM
TITLE PGM15_3: TIME_DISPLAY_VER_3
;chương trình thường trú bộ nhớ chỉ ra thời gian
;hiện tại trong ngày
;kích hoạt bằng tổ hợp phím Ctrl và Shift phải
;

```

```

        EXTRN  INITIALIZE:NEAR, SAVE_SCREEN:NEAR
        EXTRN  RESTOR_SCREEN:NEAR, SET_CURSOR:NEAR
        EXTRN  GET_TIME:NEAR
        PUBLIC  MAIN
        PUBLIC  NEW_VEC,OLD_VEC,SS:C_BUF

C_SEG  SEGMENT
        ASSUME CS:C_SEG, DS:C_SEG, SS:C_SEG
        ORG    100H

START:   JMP    INITIALIZE

;
SS_BUF  DB     16 DUP(?) ;bộ đệm lưu giữ màn hình
TIME_BUF DB    '00:00:00$' ;bộ đệm thời gian
;hr:min:sec
CURSOR_POS DW ?      ;vị trí con trỏ
ON_FLAG  DB 0       ;l=chạy thủ tục ngắt
NEW_VEC   DW ?,?    ;chứa vector mới
OLD_VEC   DW ?,?    ;chứa vector cũ
;

MAIN     PROC
;thủ tục ngắt
;cắt các thanh ghi
        PUSH   DS
        PUSH   ES
        PUSH   AX
        PUSH   BX
        PUSH   CX
        PUSH   DX
        PUSH   SI
        PUSH   DI

        MOV    AX,CS    ;thiết lập DS
        MOV    DS,AX
        MOV    ES,AX    ;và ES cho đoạn hiện hành
;gọi thủ tục ngắt bàn phím cũ
        PUSHF   ;cắt thanh ghi cờ
        CALL   OLD_VEC
;lấy các cờ bàn phím
        MOV    AX,CS    ;thiết lập lại DS
        MOV    DS,AX
        MOV    ES,AX    ;và ES cho đoạn hiện hành
        MOV    AH,02    ;hàm 2,các cờ bàn phím
        INT    16H     ;AL chứa các bit cờ
        TEST   AL,1     ;Shift phải?
        JE     I_DONE   ;không, thoát
        TEST   AL,100h   ;Ctrl?
        JE     I_DONE   ;không, thoát
;đúng,xử lý
        CMP    ON_FLAG,1 ;thủ tục đang được

```

```

;kích hoạt?
JE      RESTOR          ;đúng, khử kích hoạt
MOV     ON_FLAG, 1 ;không, kích hoạt
;cắt vị trí con trỏ và thông tin màn hình
MOV     AH, 03           ;lấy vị trí con trỏ
MOV     BH, 0             ;trang 0
INT     10H              ;DH=hàng, DL=cột
MOV     CURSOR_POS, DX  ;lưu vị trí con trỏ
CALL    SAVE_SCREEN      ;lưu màn hình hiển thị
                                ;thời gian
;định vị con trỏ đến góc trên bên phải
MOV     DL, 72            ;cột 72
CALL   SET_CURSOR        ;định vị con trỏ dòng
                                ;0, cột 72
LEA     BX, TIME_BUF    ;lấy thời gian hiện tại
CALL   GET_TIME
;hiển thị thời gian
LEA     SI, TIME_BUF
MOV     CX, 8             ;8 ký tự
MOV     BH, 0             ;trang 0
MOV     AH, 0Eh            ;viết ký tự
M1:    LODSB              ;ký tự trong AL
INT     10H              ;con trỏ được chuyển đến
                                ;cột tiếp theo
LOOP   M1                ;lặp lại 8 lần
JMP    RES_CURSOR

RESTOR:
;phục hồi màn hình
MOV     ON_FLAG, 0 ;xoá cờ
CALL   RESTORE_SCREEN
;phục hồi vị trí con trỏ đã lưu trữ
RES_CURSOR:
MOV     AH, 02            ;thiết lập con trỏ
MOV     BH, 0
MOV     DX, CURSOR_POS
INT    10H
;phục hồi các thanh nghi
I_DONE:
POP    DI
POP    SI
POP    DX
POP    CX
POP    BX
POP    AX
POP    ES
POP    DS
IRET
;trở về từ ngắt
MAIN
ENDP
;

```

```

C_SEG      ENDS
END        START      ;lệnh khởi đầu

```

Vì rằng chương trình được viết như một chương trình .COM, chúng ta cần phải viết lại file chứa thủ tục GET_TIME với các dẫn hướng đoạn đầy đủ. File PGM15_3C gồm có các thủ tục GET_TIME, CONVERT và SETUP_INT.

Program listing PGM15_3C.ASM

```

TITLE PGM15_3C: GET AND CONVERT TIME TO ASCII
        PUBLIC GET_TIME,SETUP_INT
C_SEG      SEGMENT PUBLIC
        ASSUME CS:C_SEG
;
GET_TIME PROC
;lấy thời gian trong ngày và chứa các chữ số ASCII
;trong bộ đệm thời gian
;vào: BX=địa chỉ bộ đệm thời gian
        MOV     AH,2CH          ;lấy thời gian
        INT     21H
;CH=gìờ,CL=phút,DH=giây
;đổi giờ sang dạng ASCII và lưu trữ
        MOV     AL,CH            ;giờ
        CALL    CONVERT           ;đổi sang dạng ASCII
        MOV     [ BX],AX          ;cắt
;đổi phút sang dạng ASCII và lưu trữ
        MOV     AL,CL            ;phút
        CALL    CONVERT           ;đổi sang dạng ASCII
        MOV     [ BX+3],AX;đổi sang dạng ASCII và lưu trữ
;đổi giây sang dạng ASCII và lưu trữ
        MOV     AL,DH            ;giây
        CALL    CONVERT           ;đổi sang dạng ASCII
        MOV     [ BX+6],AX;đổi sang dạng ASCII và lưu trữ
        RET
GET_TIME ENDP
;
CONVERT PROC
;đổi số kiểu byte (0-59) sang các chữ số ASCII
;vào: AL = số
;ra : AX = các chữ số ASCII,
;      AL=chữ số hàng chục,AH=chữ số hàng đơn vị
        MOV     AH,0              ;xoá AH
        MOV     DL,10             ;chia AX cho 10
        DIV     DL                ;AH chứa số dư,AL=chữ thương
        OR     AX,3030H           ;đổi sang dạng ASCII
        RET

```

```

CONVERT END
;
SETUP_INT PROC
;cắt vector cũ và thiết lập vector mới
;vào:AL = số hiệu ngắt
;        DI = địa chỉ bộ đệm vector cũ
;        SI = địa chỉ bộ đệm chứa vector mới
;cắt vector cũ
    MOV     AH,35H      ;hàm 35h,lấy vector ngắt
    INT     21H          ;ES:BX = vector
    MOV     [DI],BX       ;cắt offset
    MOV     [DI+2],ES;cắt địa chỉ đoạn
;thiết lập vector mới
    MOV     DX,[SI]       ;DX chứa offset
    PUSH   DS            ;cắt DS
    MOV     DS,[SI+2];DS chứa số hiệu đoạn
    MOV     AH,25H ;hàm 25h,thiết lập vector ngắt
    INT     21H
    POP     DS            ;phục hồi DS
    RET
SETUP_INT ENDP
;
C_SEG ENDS
END

```

Lệnh LINK của chúng ta như sau: LINK PGM15_3 + PGM15_3B + PGM15_3C + PGM15_3A. Chú ý rằng PGM15_3A được liên kết cuối cùng để thủ tục INITIALIZE được đưa vào cuối chương trình. Viết các chương trình TSR rất phức tạp, nếu như đã có một chương trình TSR khác trong hệ thống, chương trình của bạn có thể không còn thực hiện đúng nữa.

TỔNG KẾT

- ◆ Một ngắt có thể được yêu cầu bởi một thiết bị phần cứng hay một chương trình sử dụng lệnh INT hay cũng có thể được phát sinh ngay bên trong bộ vi xử lý.
- ◆ Lệnh INT sử dụng số hiệu ngắt để gọi một chương trình con phục vụ ngắt.
- ◆ 8086 cung cấp 256 ngắt và vector ngắt (địa chỉ của các thủ tục), các vector ngắt này được lưu giữ trong 1KB đầu tiên của bộ nhớ.
- ◆ Các ngắt 0-1Fh gọi là các chương trình phục vụ ngắt của BIOS, các vec tơ ngắt của chúng được BIOS thiết lập khi bật máy.
- ◆ Các ngắt 20h-3Fh là các thường trình phục vụ ngắt của DOS.
- ◆ Người sử dụng có thể viết chương trình của chính mình để thực hiện các Nhiệm vụ khác nhau.
- ◆ Một chương trình thường trú bộ nhớ có thể được kích hoạt bởi một tổ hợp phím nhấn.

Các thuật ngữ tiếng anh.

| | |
|-------------------------------------|--|
| Conventional memory | 640 KB đầu tiên của bộ nhớ |
| hand-shaking | Một thủ tục liên lạc của các thiết bị. |
| hardware interrupt | Một thiết bị phần cứng ngắt bộ vi xử lý |
| interrupt acknowledge signal | Một tín hiệu phát sinh bởi bộ vi xử lý khi nó chấp nhận một yêu cầu ngắt |
| interrupt number | Con số xác định một ngắt |
| interrupt request signal | Tín hiệu một thiết bị phần cứng gửi đến bộ vi xử lý yêu cầu phục vụ |
| interrupt routine | Thủ tục được gọi bởi một ngắt |
| interrupt vector | Địa chỉ của một thường trình phục vụ ngắt |
| interrupt vector table | Tập hợp tất cả các vector ngắt |

memory resident program Một chương trình TSR

NMI (non maskable interrupt)

Một ngắt cứng không che được bằng cách xoá cờ IF

processor exception Tình huống trong của bộ vi xử lý đòi hỏi một sự xử lý đặc biệt.

TSR (terminate and stay resident) program Chương trình còn ở lại trong bộ nhớ sau khi kết thúc.

software interrupt Lệnh INT

Các lệnh mới

IRET

Các toán tử giả mới

OFFSET SEG

Bài tập

1. Tính toán vị trí của vector ngắt của ngắt 20h.
2. Dùng DEBUG tìm giá trị của vector ngắt của ngắt 0
3. Viết các lệnh sử dụng ngắt 17h của BIOS để in dòng chữ "Hello".
4. Viết các lệnh sử dụng ngắt 21h, hàm 9 để hiển thị ngày hiện tại

Các bài tập lập trình.

5. Viết một chương trình cứ nửa giây lại đưa ra dòng chữ "Hello" trên màn hình.
6. Sửa chương trình PGM15_2.ASM để nó gọi hàm 9 ngắt 21h hiển thị thời gian chỉ khi nào con số chỉ giây thay đổi.
7. Viết một chương trình thường trú bộ nhớ tương tự PGM15_3.ASM sử dụng ngắt 21h, hàm 31h.

Chương 16

ĐỒ HOẠ MÀU

Tổng quan

Trong chương 12 chúng tôi đã chỉ ra màn hình làm việc như thế nào trong chế độ văn bản. Trong chương này chúng ta sẽ bàn về các chế độ đồ họa của PC. Có 3 bộ phôi ghép đồ họa màu phổ biến cho các máy PC là :CGA(Color Graphic Adapter- bộ phôi ghép đồ họa màu), EGA (Enhanced Graphic Adapter- bộ phôi ghép đồ họa nâng cao), VGA (Video Graphic Array- bộ phôi ghép đồ họa kiểu ma trận điểm). Chúng tôi sẽ miêu tả hoạt động và cách lập trình với chúng đồng thời cũng giúp các bạn viết một chương trình trò chơi video tương tác.

16.1 Các chế độ đồ họa.

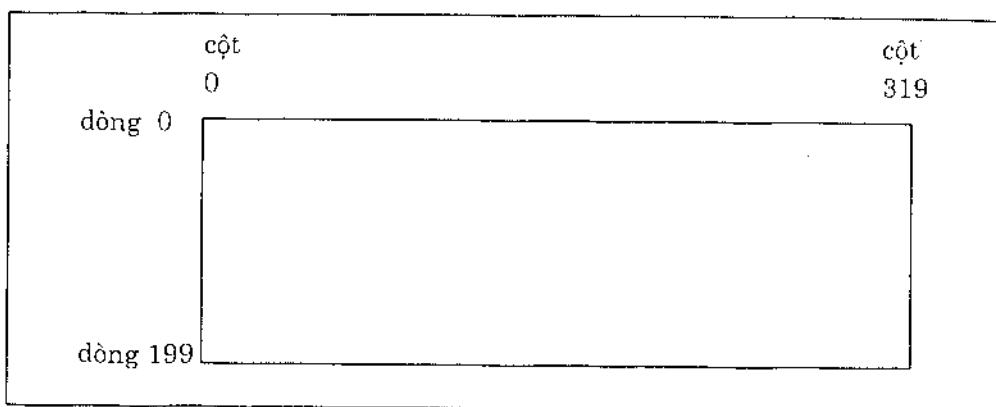
Như đã nhắc đến trong chương 12, màn hình hiển thị là tổ hợp các dòng được chùm tia điện tử quét lên. Các dòng này được gọi là các dòng quét (scan lines). Một điểm ảnh được tạo lên khi tia điện tử bật hay tắt trong khi quét. Các điểm tạo lên các ký tự cũng như các hình ảnh trên màn hình. Tín hiệu video điều khiển việc quét được tạo lên bởi vi mạch phôi ghép màn hình bên trong máy tính.

Bộ phôi ghép màn hình có thể thay đổi số điểm trên một dòng bằng cách thay đổi kích thước các điểm. Một số bộ phôi ghép màn hình còn có khả năng thay đổi cả số dòng quét.

Các điểm ảnh (pixels).

Khi làm việc ở các chế độ đồ họa, màn hình được chia làm các cột và các dòng, và mỗi vị trí trên màn hình xác định bởi tọa độ dòng và cột được gọi là một điểm ảnh (picture element, pixel). Số cột và số dòng thể hiện độ phân giải (resolution) của chế độ đồ họa tương ứng, ví dụ độ phân giải 320×200 có nghĩa là 320 cột và 200 hàng. Các cột được đánh số từ trái sang phải bắt đầu từ 0 còn các hàng được đánh số từ trên xuống dưới cũng bắt đầu từ 0. Chẳng hạn trong

chế độ 320x200, góc phải trên của màn hình có tọa độ là cột 319 dòng 0 còn điểm ảnh ở góc phải dưới tương ứng với cột 319, dòng 199. Xem hình 16.1.



Hình 16.1 Tọa độ các điểm ảnh ở độ phân giải 320 × 200.

Tùy thuộc vào ánh xạ từ các cột và các hàng đến các dòng quét và các điểm quét (dots) mà mỗi điểm ảnh có thể có một hoặc hơn một điểm quét. Ví dụ trong chế độ phân giải thấp của CGA, có 160 cột và 100 hàng nhưng CGA tạo ra 320 điểm quét và 200 dòng do đó mỗi điểm ảnh được tạo lên bởi một tập hợp 2×2 điểm quét. Chế độ đồ họa gọi là **APA** (All Points Addressable) nếu như mỗi điểm ảnh của nó tương ứng với một điểm quét duy nhất.

Bảng 16.1 trình bày các chế độ đồ họa APA của các bộ phôi ghép màn hình CGA, EGA, VGA. để giữ tính tương thích EGA được thiết kế để hiển thị mọi chế độ của CGA và VGA thì được thiết kế để có thể hiển thị mọi chế độ của EGA.

Chọn mode

Màn hình thường được thiết lập chế độ văn bản do đó thao tác đầu tiên để hiển thị ở chế độ đồ họa là phải thiết lập lại chế độ hiển thị thích hợp. Chúng tôi đã trình bày trong chương 12 ngắt 10h của BIOS quản lý tất cả các hàm màn hình, hàm 0 dùng để thiết lập chế độ màn hình.

Ngắt 10h Hàm 0:

Thiết lập chế độ màn hình

| | |
|------|--------------------|
| Vào: | AH=0 AL=số mode |
|------|--------------------|

| | |
|-----|-------|
| Ra: | không |
|-----|-------|

Ví dụ 16.1: Thiết lập chế độ hiển thị 640 × 200, 2 màu.

Trả lời:

Từ bảng 16.1 chúng ta xác định được số mode là 06h và có các lệnh thực hiện công việc trên như sau:

```

MOV AH, 0          ;hàm 0
MOV AL, 06H        ;mode 6
INT 10H           ;chọn chế độ

```

| Số mode (dạng hex) | Bộ phối ghép CGA |
|-----------------------|---------------------|
| 4 | 320x200 4 màu |
| 5 | 320x200 4 màu (xám) |
| 6 | 620x200 2 màu |
| | Bộ phối ghép EGA |
| D | 320x200 16 màu |
| E | 640x200 16 màu |
| F | 640x350 đơn sắc |
| 10 | 640x350 16 màu |
| | Bộ phối ghép VGA |
| 11 | 640x480 2 màu |
| 12 | 640x480 16 màu |
| 13 | 320x200 256 màu |

Bảng 16.1 Chế độ hiển thị của các bộ phối ghép đồ họa

16.2 Bộ phối ghép CGA

Bộ phối ghép CGA cung cấp 3 chế độ phân giải khác nhau: độ phân giải thấp 160×100 , độ phân giải trung bình 320×200 và độ phân giải cao 640×200 . Phục vụ 10h của BIOS chỉ cho phép các độ phân giải trung bình và cao. Các chương trình sử dụng chế độ phân giải thấp phải truy nhập trực tiếp vào chip điều khiển màn hình.

Bộ phối ghép CGA có bộ nhớ hiển thị 16KB nằm ở đoạn B800h địa chỉ bộ nhớ là từ B800:0000 đến B800:3FFF. Mỗi điểm ảnh chiếm một hoặc nhiều bit tùy thuộc vào mode. Chẳng hạn, trong chế độ phân giải cao mỗi điểm ảnh chiếm 1 bit trong khi với chế độ phân giải trung bình lại là 2 bit. Giá trị của bit biểu diễn điểm ảnh xác định màu sắc của nó.

Bảng 16.2 : 16 màu chuẩn của bộ phôi ghép CGA

| I R G B | COLOR |
|---------|------------------|
| 0 0 0 0 | Black |
| 0 0 0 1 | Blue |
| 0 0 1 0 | Green |
| 0 0 1 1 | Cyan |
| 0 1 0 0 | Red |
| 0 1 0 1 | Magenta (purple) |
| 0 1 1 0 | Brown |
| 0 1 1 1 | White |
| 1 0 0 0 | Gray |
| 1 0 0 1 | Light Blue |
| 1 0 1 0 | Light Green |
| 1 0 1 1 | Light Cyan |
| 1 1 0 0 | Light Red |
| 1 1 0 1 | Light magenta |
| 1 1 1 0 | Yellow |
| 1 1 1 1 | Intense White |

Chế độ phân giải trung bình

CGA có khả năng hiển thị 16 màu, bảng 16.2 mô tả các màu của CGA. Trong chế độ phân giải trung bình, màn hình có thể thể hiện 4 màu cùng một lúc. Sở dĩ có hạn chế này là vì bộ nhớ hiển thị có kích thước hạn chế. Do độ phân giải là 320×200 nên sẽ có $320 \times 200 = 64.000$ điểm ảnh. Để hiển thị 4 màu mỗi điểm ảnh được mã hoá bằng 2 bit và bộ nhớ cần thiết là $64.000 \times 2 = 128.000$ bit hay 16000 byte. Chính vì thế bộ nhớ hiển thị 16KB của CGA chỉ cung cấp được 4 màu trong chế độ này.

Để cho phép các nhóm 4 màu khác nhau trong chế độ phân giải trung bình, CGA sử dụng **2 bảng màu (palette)**. Một bảng màu là tập hợp các màu có thể thể hiện cùng lúc. Mỗi bảng màu chứa 3 màu cố định và một màu nền (**background color**) có thể chọn từ 16 màu chuẩn. Màu nền là màu mặc định của tất cả các điểm ảnh, do đó màn hình sẽ thể hiện màu nền nếu không có dữ liệu viết lên nó. Bảng 16.3 trình bày 2 bảng màu.

Bảng màu mặc định là bảng màu 0, nhưng các chương trình có thể chọn bất cứ bảng màu nào để hiển thị. Giá trị của điểm ảnh (0-3) xác định màu trong bảng màu hiện thời. Nếu chúng ta thay đổi bảng màu, tất cả các điểm ảnh trên màn hình sẽ đổi màu. Chúng ta có thể sử dụng hàm con OBh của ngắt 10h để chọn bảng màu hay màu nền.

INT 10h, Function 0Bh**Chọn bảng màu hay màu nền****Hàm con 0: chọn màu nền**

Vào: AH=0BH
BH=0
BL=Số màu (0-15)

Ra: không

Hàm con 1: chọn bảng màu

Vào: AH=0Bh
BH=1
BL=số bảng màu (0 hay 1)

Ra: không

| Bảng màu | Trị số điểm ảnh | Màu |
|----------|-----------------|-------------|
| 0 | 0 | Nền |
| | 1 | Xanh lá cây |
| | 2 | Đỏ |
| | 3 | Nâu |
| | | |
| 1 | 0 | Nền |
| | 1 | Xanh Cyan |
| | 2 | Tía |
| | 3 | Trắng |

Bảng 16.3 Các bảng 4 màu của CGA**Ví dụ 16.2:** Viết các lệnh chọn bảng màu 1 và màu nền xanh nhạt.**Trả lời:**

Màu xanh nhạt có số hiệu màu là 9 do đó ta có:

```

MOV AH, 0BH      ;hàm 0Bh
MOV BH, 00H      ;chọn màu nền
MOV BL, 09       ;màu xanh nhạt

```

```

INT 10H
MOV BH, 01      ; chọn bảng màu
MOV BL, 1       ; bảng màu 1
INT 10H

```

Đọc và ghi các điểm ảnh

Để đọc và ghi các điểm ảnh chúng ta phải xác định nó thông qua toạ độ cột và dòng. Các hàm 0Dh và 0Ch tương ứng dùng để đọc và ghi các điểm ảnh.

INT 10H, Hàm 0Ch:

Ghi điểm ảnh

| | |
|------|--|
| Vào: | AH=0CH AL=Trị số điểm ảnh BH=Số trang(với CGA giá trị này được bỏ qua) CX=Toạ độ cột DX=Toạ độ dòng |
| Ra: | không |

INT 10H, Hàm 0Dh:

Đọc điểm ảnh

| | |
|------|--|
| Vào: | AH=0DH BH=Số trang(với CGA giá trị này được bỏ qua) CX=Toạ độ cột DX=Toạ độ dòng |
| Ra: | AL=Trị số điểm ảnh |

Ví dụ 16.3: Chép điểm ảnh tại dòng 199 cột 50 đến điểm ảnh tại dòng 40 cột 20

Trả lời:

Trước tiên chúng ta đọc điểm ảnh tại dòng 199 cột 50, sau đó ghi vào điểm ảnh tại dòng 40 cột 20:

```

MOV AH, 0DH      ;đọc điểm ảnh .
MOV CX, 50       ;cột 50

```

```

MOV DX, 199      ;dòng 199
INT 10H          ;AL chứa trị số điểm ảnh
MOV AH, 0CH      ;ghi điểm ảnh, AL đã được thiết lập
MOV CX, 20       ;cột 20
MOV DX, 40       ;dòng 40
INT 10H

```

Chế độ phân giải cao

Trong chế độ phân giải cao; CGA có thể hiển thị 2 màu, mỗi điểm ảnh có giá trị 0 hoặc 1; 0 ứng với màu đen và 1 ứng với màu trắng. Chúng ta cũng có thể chọn màu nền bằng hàm OBh ngắt 10h, khi màu nền được chọn thì điểm ảnh có giá trị 0 sẽ có màu trùng màu nền còn điểm ảnh có giá trị 1 có màu trắng.

Bây giờ chúng ta sẽ xem một chương trình đồ họa hoàn chỉnh.

Ví dụ 16.4: Hãy viết một chương trình vẽ một đường thẳng trên dòng 100 từ cột 301 đến cột 600 trong chế độ phân giải cao.

Trả lời:

Tổ chức của chương trình sẽ như sau: (1) thiết lập chế độ hiển thị là 6 (CGA phân giải cao), (2) vẽ đường thẳng, (3) đọc vào một phím và (4) thiết lập lại chế độ hiển thị trở lại chế độ 3 (chế độ văn bản). Sở dĩ chúng ta phải thêm vào cả bước 3 vì nếu không chúng ta không thể kịp nhìn đường thẳng.

Chương trình nguồn PGM16_1.ASM

```

TITLE PGM16_1: CGA LINE DRAWING
;Vẽ đường thẳng nằm ngang trong chế độ phân
;giải cao trên dòng 100 từ cột 301 đến cột 600.
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
;thiết lập chế độ đồ họa
    MOV AX, 6 ;chọn mode6 độ phân giải cao
    INT 10H
;vẽ đường thẳng
    MOV AH, 0CH      ;viết điểm ảnh
    MOV AL, 1         ;màu trắng
    MOV CX, 301      ;cột bắt đầu
    MOV DX, 100      ;tại dòng 100
L1:  INT 10H
    INC CX          ;cột tiếp theo
    CMP CX, 600      ;đã đến cột 600?
    JLE L1           ;chưa! tiếp tục
;đọc vào từ bàn phím

```

```

MOV AH, 0
INT 16H
; thiết lập lại chế độ văn bản
MOV AX, 3      ; chọn mode 3, chế độ văn bản
INT . 10H
; trả về DOS
MOV AH, 4CH      ; quay về DOS
INT 21H
MAIN ENDP
END MAIN

```

Viết trực tiếp vào bộ nhớ

Khi chúng ta muốn thay đổi nhanh màn hình, như trong các trò chơi video, chúng ta có thể không dùng các phục vụ của BIOS mà thay vào đó có thể viết trực tiếp vào bộ nhớ màn hình CGA. Để làm được điều đó chúng ta phải hiểu tổ chức bộ nhớ hiển thị của phôi ghép CGA. Bộ nhớ hiển thị 16KB của CGA được chia làm 2 nửa. Các điểm ảnh nằm ở dòng chẵn được chứa trong 8 KB đầu tiên (B800:0000 đến B800:1FFF), còn các điểm ảnh nằm trên dòng lẻ thì được chứa trong 8 KB thứ 2 (B800:2000 đến B800:3FFF). Mỗi hàng chiếm 50 byte. Hình 16.2 chỉ ra mối quan hệ giữa địa chỉ bộ nhớ màn hình và màn hình.

Để định vị một bit cho một điểm ảnh xác định trước tiên chúng ta phải xác định byte bắt đầu của hàng chứa điểm ảnh và sau đó xác định offset của điểm ảnh trong hàng. Bây giờ chúng ta hãy xét một ví dụ.

Ví dụ 16.5: Giả sử chế độ màn hình đang là chế độ 4. Hãy xác định địa chỉ byte và vị trí bit cho điểm ảnh trong dòng 5 cột 10.

Trả lời:

Dòng 5 là dòng lẻ thứ 3 do đó byte đầu tiên cho dòng 5 có địa chỉ offset là $2000h + 2 \times 50 = 20A0h$. Trong mode 4 mỗi điểm ảnh là 2 bit do đó mỗi byte chứa được 4 điểm ảnh. Cột 10 là cột thứ 11 trong hàng do đó điểm ảnh sẽ là điểm ảnh thứ 3 trong byte thứ 3. Địa chỉ byte sẽ là $20A0h + 2 = 20A2h$. Các điểm ảnh được lưu trong byte từ trái sang do đó điểm ảnh thứ 3 có vị trí bit là 3 và 2. **Ví dụ 16.6:** Giả sử chế độ màn hình đang là mode4. Viết một điểm ảnh có trị số 10b vào vị trí cột 10 dòng 5.

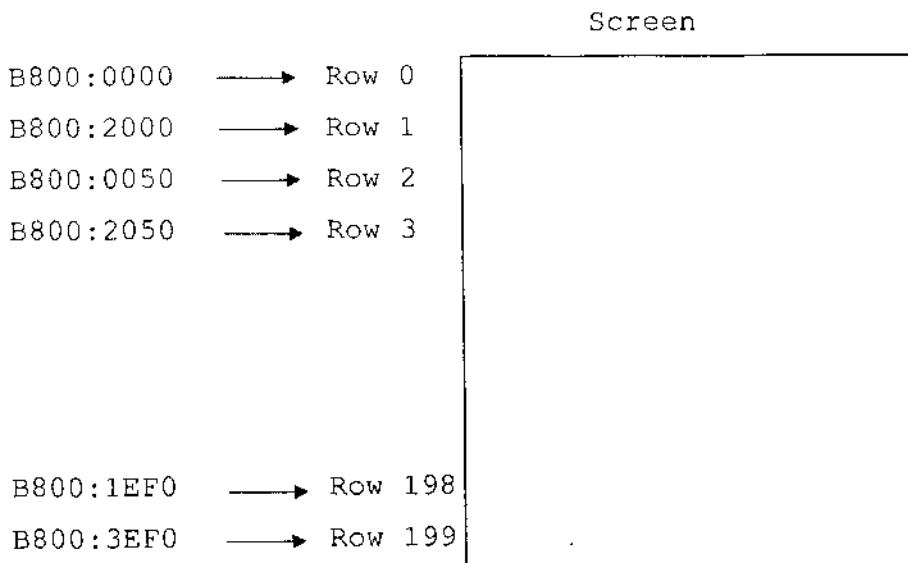
Trả lời:

Chúng ta sẽ sử dụng địa chỉ đã tính được trong ví dụ trước, để viết điểm ảnh trước tiên chúng ta sẽ đọc byte có chứa điểm ảnh, thay đổi bit thích hợp rồi ghi trả lại. Sở dĩ cần phải đọc trước khi viết là để tránh việc làm thay đổi giá trị của những điểm ảnh chứa trong cùng byte. Để thay đổi các bit, đầu tiên chúng ta sẽ xoá chúng bằng lệnh AND sau đó ghi giá trị mới bằng lệnh OR.

```

MOV AX, 0B800H ;địa chỉ đoạn của bộ nhớ màn hình
MOV ES, AX ;đặt nó vào ES
MOV DI, 20A2H ;địa chỉ offset của byte
MOV AL, ES:[DI] ;chuyển byte vào AL
AND AL, 11110011B ;xoá các bit cần thay đổi
OR AL, 1000B ;ghi 10b vào vị trí 3 và 2 trong byte
STOSB ;ghi lại byte vào bộ nhớ màn hình

```



Hình 16.2 Bộ nhớ hiển thị của CGA

Hiển thị văn bản

Chúng ta cũng có thể hiển thị văn bản trong chế độ đồ họa. Các ký tự văn bản trong chế độ đồ họa không phải được tạo ra từ mạch tạo ký tự như trong chế độ văn bản. Thay vào đó các ký tự được tạo lên từ các font chữ chứa trong bộ nhớ. Một điểm khác biệt nữa giữa chế độ văn bản và chế độ đồ họa là trong chế độ đồ họa con trỏ không hiện lên trên màn hình. Tuy nhiên chúng ta vẫn có thể định vị con trỏ thông qua hàm 2 của ngắt 10h.

Ví dụ 16.7: Viết các lệnh hiển thị chữ "A" màu đỏ ở góc phải trên màn hình. Sử dụng mode 4 với nền màu xanh da trời.

Trả lời:

Khi chúng ta hiển thị các ký tự trong chế độ đồ họa chúng ta sử dụng tọa độ văn bản. Với độ phân giải 320×200 , chỉ có thể hiển thị được 40 cột văn bản trên màn hình(xem bảng 16.4). Vì vậy tọa độ dòng và cột ứng với góc phải trên màn hình là 0 và 39. Để hiển thị chữ đỏ trên nền xanh chúng ta dùng bảng màu 0 với màu nền xanh da trời.

Các bước của công việc sẽ như sau: (1) thiết lập mode 4, (2) thiết lập màu nền xanh, (3) định vị con trỏ và (4) hiển thị chữ cái "A" màu đỏ.

```

MOV AH, 0          ;thiết lập chế độ
MOV AL, 04H        ;mode 4
INT 10H
MOV AH, 0BH        ;hàm con 0BH
MOV BH, 00H        ;chọn màu nền
MOV BL, 3          ;xanh da trời
INT 10H
MOV AH, 02          ;định vị con trỏ
MOV BH, 0          ;trang 0
MOV DH, 0          ;dòng 0
MOV DL, 39         ;cột 39
INT 10H
MOV AH, 09          ;hàm ghi ký tự
MOV AL, 'A'         ;ghi chữ "A"
MOV BL, 2          ;màu đỏ
MOV CX, 1          ;ghi 1 ký tự
INT 10H

```

| Độ phân giải | Cột văn bản | Hàng văn bản |
|---------------------|--------------------|---------------------|
| 320 x 200 | 40 | 25 |
| 640 x 200 | 80 | 25 |
| 640 x 350 | 80 | 25 |
| 640 x 480 | 80 | 29 |

Bảng 16.4 Hàng và cột văn bản trong chế độ đồ họa

16.3 Bộ phối ghép EGA

Bộ phối ghép EGA có thể tạo được 200 hay 350 dòng quét. Để hiển thị ở chế độ phân giải cao cần có một màn hình ECD(Enhanced Color Display). Bộ phối ghép EGA có 16 thanh ghi bảng màu, các thanh ghi này chứa các màu hiện thời. Mỗi thanh ghi bảng màu có 6 bit màu, mỗi cặp 2 bit biểu diễn một màu cơ bản. Do đó mỗi thanh ghi có khả năng chứa được bất cứ màu nào trong 64 màu và EGA có thể thể hiện 16 màu bất kỳ trong tổng số 64 màu cùng lúc. Trong chế độ EGA 16 màu mỗi trị số điểm ảnh chọn một thanh ghi bảng màu. Ban đầu 16 thanh ghi bảng màu được nạp 16 màu chuẩn CGA. Để hiển thị các màu khác trên màn hình chương trình có thể thay đổi các thanh ghi bảng màu bằng cách sử dụng ngắt 10h phục vụ 10h hàm con 00h (xem phụ lục C).

Bộ phổi ghép EGA có thể giả lập chế độ đồ họa của CGA do đó các chương trình viết cho CGA có thể chạy trên EGA với cùng màu. Bộ nhớ hiển thị của EGA có thể cấu hình hoá bằng phần mềm. Tuỳ thuộc vào chế độ hiển thị, bộ nhớ hiển thị có thể bắt đầu từ địa chỉ A0000h, B0000h hay B8000h. Khi hiển thị với chế độ CGA, bộ nhớ hiển thị bắt đầu tại địa chỉ B8000h để giữ tính tương thích với bộ nhớ hiển thị của CGA.

Trong các chế độ hiển thị EGA, bộ nhớ màn hình có cấu trúc như sau: nó nằm ở đoạn A000h và sử dụng đến 256KB. Để đặt 256KB trong một đoạn EGA sử dụng 4 mảng 64KB. Mỗi mảng này được gọi là các mặt bit (bit plane). Các mặt bit này có chung 64K địa chỉ bộ nhớ, mỗi địa chỉ chỉ đến 4 byte tương ứng với 4 mặt bit. 8086 không truy nhập trực tiếp được đến các mặt bit mà phải thông qua các thanh ghi của EGA.

Chúng ta có thể thấy rằng với một dung lượng nhớ lớn đến như vậy, bộ nhớ màn hình có thể chứa được lượng dữ liệu của hơn một màn hình. Trong chế độ EGA bộ nhớ màn hình được chia thành nhiều trang, mỗi trang có kích thước bằng lượng dữ liệu của một màn hình. Chẳng hạn trong mode D (320x 200 với 16 màu) có 64000 điểm ảnh, mỗi điểm ảnh chiếm 4 bit và như vậy bộ nhớ cần thiết cho một màn hình là 32000 byte. Nếu bộ nhớ màn hình có dung lượng 256 KB thì nó có thể chứa tối 8 trang màn hình (0 đến 7). Trong trường hợp bộ nhớ màn hình nhỏ hơn tất nhiên sẽ có ít trang hơn.

Khi chúng ta đọc và ghi các điểm ảnh bằng các hàm 0Ch và 0Dh, số trang được xác định trong BH. Các hàm này có thể sử dụng với bất kỳ trang nào không phụ thuộc vào trang đang được hiển thị.

Ví dụ 16.8: Giả sử chúng ta đang sử dụng một bảng màu 16 màu, hãy viết một điểm ảnh màu xanh lá cây lên trang 2 tại dòng 0 cột 0.

Trả lời:

Chúng ta sử dụng hàm 0Ch và trị số màu là 2.

```
MOV AH, 0CH ;hàm ghi một điểm ảnh
MOV AL, 2      ;màu xanh lá cây
MOV BH, 2      ;tại trang 2
MOV CX, 0      ;cột 0
MOV DX, 0      ;dòng 0
INT 10H
```

Khi chế độ đồ họa được chọn đầu tiên, trang màn hình hoạt động tự động chọn là trang 0. Chúng ta có thể chọn trang màn hình hoạt động khác bằng cách sử dụng hàm 05h.

Ví dụ 16.9: Hãy chọn trang 1 làm trang màn hình hoạt động

Trả lời:

```
MOV     AH, 05H      ; hàm chọn trang hoạt động  
MOV     AL, 01        ; chọn trang 1  
INT     10H
```

Bằng cách chuyển các trang màn hình hoạt động có thể tạo lên một hoạt cảnh đơn giản. Giả sử chúng vẽ một hình ở trang 0 sau đó vẽ một hình giống như thế ở một vị trí hơi khác trong trang 1, và cứ như vậy. Sau đó bằng cách chuyển đổi nhanh màn hình hoạt động chúng ta có thể thấy hình vẽ chuyển động trên màn hình. Sự chuyển động này bị giới hạn của số trang màn hình sẵn có. Chúng tôi sẽ trình bày một kỹ thuật hoạt hình thực tế hơn trong phần 16.5.

Ngắt 10h, hàm 5:

chọn trang màn hình hoạt động

| | |
|------|--------------|
| Vào: | AH=5 |
| | AL= số trang |
| Ra: | không |

16.4 Bộ phối ghép VGA

Bộ phối ghép VGA có độ phân giải cao hơn so với EGA, nó có thể hiển thị với độ phân giải 640×480 trong mode 12h. Ngoài ra VGA còn có khả năng hiển thị với nhiều màu hơn, bộ phối ghép VGA có khả năng tạo ra 64 mức độ khác nhau của mỗi màu cơ bản(red, green, blue). Các tổ hợp màu của các màu trên tạo ra 64^3 bằng 256K màu khác nhau. Số lượng màu cực đại có thể thể hiện cùng lúc là 256. Màu đang được hiển thị được chứa trong 256 thanh ghi màu đồ họa DAC (Digital to Analog Circuit). Mỗi thanh ghi màu có 18 bit, mỗi bộ 6 bit dùng cho một màu cơ bản. Để hiển thị tất cả các màu đó chúng ta phải có một màn hình tương tự (analog monitor).

Bộ phối ghép VGA có thể giả lập các chế độ đồ họa EGA và CGA. Trong chế độ VGA bộ nhớ màn hình được tổ chức thành các mặt bit cũng giống như trong chế độ EGA.

Chúng ta hãy xem xét chế độ 13h của VGA, chế độ này cho phép hiển thị 256 màu. Trong chế độ này, mỗi điểm ảnh chiếm 1 byte, và byte này chọn thanh ghi màu. Các thanh ghi màu được nạp ban đầu với các trị số mặc định. Chúng ta có

khả năng thay đổi các thanh ghi màu nhưng trước hết hãy xem các màu mặc định.

Ví dụ 16.10: Viết các lệnh hiển thị 256 điểm với 256 màu mặc định tại dòng 100.

Trả lời:

Trước hết chúng ta sẽ chọn mode 13h, sau đó tạo ra một vòng lặp ghi giá trị trong AL vào các cột từ 0 đến 255, giá trị trong AL sẽ tự động tăng trong vòng lặp từ 0 đến 255.

```
MOV AH,0          ; thiết lập mode
MOV AL,13H        ; 13h
INT 10H
; hiển thị 256 điểm ảnh tại dòng 100
MOV AH,0CH        ; hàm ghi điểm ảnh
MOV AL,0          ; bắt đầu với điểm ảnh có màu 0
MOV BH,0          ; trang 0
MOV CX,0          ; cột 0
MOV DX,100        ; dòng 100
L1: INT 10H        ; ghi điểm ảnh
    INC AL          ; màu tiếp theo
    INC CX          ; cột tiếp theo
    CMP CX,256      ; hết?
    JL L1           ; chưa! tiếp tục
```

Chúng ta có thể thiết lập màu trong các thanh ghi màu bằng hàm 10h.

Ngắt 10h, hàm 10h, hàm con 10h:

Đặt thanh ghi màu

| | |
|------|---|
| Vào: | AH=10H AL=10H BX=số hiệu thanh ghi màu CH=gia trị cho màu xanh lá cây CL=gia trị cho màu da trời DH=gia trị cho màu đỏ |
|------|---|

| | |
|-----|-------|
| Ra: | không |
|-----|-------|

Ví dụ 16.11: Đưa trị số màu: 30 cho màu đỏ, 20 cho xanh lá cây và 10 cho xanh da trời vào thanh ghi màu số 5.

Trả lời:

```

MOV AH,10H      ;hàm đặt thanh ghi màu
MOV AL,10H
MOV BX,5        ;thanh ghi màu số 5
MOV DH,30       ;trị số cho màu đỏ
MOV CH,20       ;trị số cho màu xanh lá cây
MOV CL,10       ;trị số cho màu xanh da trời
INT 10H

```

Chúng ta cũng có thể thiết lập màu cho một khối các thanh ghi màu bằng một hàm;(xem phụ lục C).

16.5 Làm hình chuyển động

Chuyển động của một vật thể được giả lập bằng cách xoá hình ảnh hiện có và hiển thị nó tại một vị trí khác. Chúng ta sẽ minh họa kỹ thuật làm hình chuyển động với một quả bóng nhỏ.

Để hiển thị quả bóng chúng ta phải chọn chế độ đồ họa, màu quả bóng và màu nền. Vì tất cả các bộ phổi ghép đều cung cấp các chế độ của CGA nên chúng ta hãy chọn mode 4. Nếu chọn bảng màu 1 với màu nền xanh lá cây chúng ta có thể vẽ một quả bóng trắng chuyển động trên nền xanh. Quả bóng sẽ được biểu diễn bằng một hình vuông 4 điểm ảnh, vị trí của nó được cho bởi toạ độ của điểm ảnh ở góc trái trên.

Hiển thị quả bóng

Chúng ta sẽ giữ quả bóng trong vùng giới hạn bởi các cột 10 đến 300 và các dòng 10 đến 189. Đường biên giới được vẽ màu lục. Ban đầu chúng ta hãy đặt quả bóng ở lề bên phải nghĩa là vị trí của quả bóng là cột 298 dòng 100.

Thủ tục SET_DISPLAY_MODE thiết lập chế độ hiển thị về mode 4, chọn bảng màu 1 và màu nền xanh sau đó vẽ đường biên giới màu lục. Đường viền được vẽ bằng 2 macro DRAW_ROW và DRAW_COLUMN. Thủ tục DISPLAY_BALL hiển thị quả bóng tại cột CX dòng DX với màu trong AL. Cả 2 thủ tục đều viết trong chương trình nguồn PGM16_2A.ASM.

Chương trình nguồn PGM16_2A.ASM

```

TITLE PGM16_2A:
PUBLIC      SET_DISPLAY_MODE, DISPLAY_BALL
.MODEL      SMALL

```

```

DRAW_ROW MACRO X
    LOCAL L1
    ;vẽ một đường thẳng tại dòng X từ cột 10 đến cột 300
    MOV AH,0CH      ;vẽ điểm ảnh
    MOV AL,1        ;màu lục
    MOV CX,10       ;cột 10
    MOV DX,X        ;dòng X
L1:   INT 10H
    INC CX         ;cột tiếp theo
    CMP CX,301     ;qua cột 300?
    JL  L1         ;chưa, lặp lại!
ENDM
;
DRAW_COLUMN MACRO Y
    LOCAL L2
    ;vẽ một đường thẳng tại cột Y từ dòng 10 đến dòng 189
    MOV AH,0CH      ;vẽ điểm ảnh
    MOV AL,1        ;màu lục
    MOV CX,Y        ;cột Y
    MOV DX,10 ;dòng 10
L2:   INT 10H
    INC DX         ;dòng tiếp theo
    CMP DX,190     ;đã qua dòng 189
    JL  L2         ;chưa, lặp lại!
ENDM
.CODE
SET_DISPLAY_MODE PROC
;thiết lập chế độ hiển thị và vẽ đường viền
    MOV AH,0          ;thiết lập chế độ
    MOV AL,04         ;mode 4, 320x200 4 màu
    INT 10H
    MOV AH,0BH        ;chọn bảng màu
    MOV BH,1
    MOV BL,1          ;bảng màu 1
    INT 10H
    MOV BH,0          ;chọn màu nền
    MOV BL,2          ;xanh lá cây
    INT 10H
;vẽ đường viền
    DRAW_ROW 10      ;vẽ dòng 10
    DRAW_ROW 189     ;vẽ dòng 189
    DRAW_COLUMN 10   ;vẽ cột 10
    DRAW_COLUMN 300  ;vẽ cột 300
    RET
SET_DISPLAY_MODE ENDP
;
DISPLAY_BALL PROC
;hiển thị quả bóng ở cột CX dòng DX có màu cho trong AL

```

```

;
; Vào: AL=màu của quả bóng
;           CX= toạ độ cột
;           DX=toạ độ dòng
; Ra:  không
    MOV  AH,0CH      ;viết điểm ảnh
    INT  10H
    INC  CX          ;viết tiếp ở cột sau
    INT  10H
    INC  DX          ;dòng dưới
    INT  10H
    DEC  CX          ;cột trước
    INT  10H
    DEC  DX          ;phục hồi lại DX
    RET
DISPLAY_BALLENDP
END

```

Chú ý rằng để xoá quả bóng chúng ta chỉ cần vẽ lại nó với màu trùng với màu nền tại vị trí hiện thời của nó. Như vậy chúng ta có thể sử dụng thủ tục DISPLAY_BALL vừa để vẽ vừa để xoá quả bóng.

Để giả lập chuyển động của quả bóng chúng ta sẽ định nghĩa tốc độ của nó bằng 2 thành phần VEL_X và VEL_Y, cả 2 đều là biến word. Khi VEL_X dương quả bóng sẽ chuyển động về bên phải và khi VEL_Y dương quả bóng sẽ di chuyển xuống dưới. Vị trí của quả bóng được cho trong CX (cột) và DX (dòng). Sau khi hiển thị quả bóng tại một vị trí nào đó, chúng ta xoá nó và tính toạ độ mới bằng cách cộng VEL_X vào CX và VEL_Y vào DX. Quả bóng sẽ được hiển thị ở vị trí mới và quá trình cứ như vậy được lặp lại.

Các lệnh sau đây hiển thị quả bóng tại cột CX dòng DX, xoá nó và hiển thị lại tại vị trí mới xác định bởi vận tốc.

```

MOV  AL,3          ;màu 3 trong bảng màu hiện thời
                   ;là màu trắng
CALL DISPLAY_BALL ;vẽ quả bóng trắng
MOV  AL,0          ;màu 0 là màu nền
CALL DISPLAY_BALL ;xoá quả bóng
ADD  CX,VEL_X     ;cột mới
ADD  DX,VEL_Y     ;dòng mới
MOV  AL,3          ;màu trắng
CALL DISPLAY_BALL ;vẽ quả bóng tại vị trí mới

```

Vì máy tính có thể thực hiện các lệnh với tốc độ cực cao, quả bóng sẽ chuyển động quá nhanh trên màn hình đến nỗi chúng ta sẽ không kịp quan sát thấy. Có một cách để giải quyết việc này là sử dụng vòng lặp trễ điều khiển bằng một

bộ đếm mỗi khi vẽ xong quả bóng. Nhưng do sự khác về tốc độ của các máy PC, vòng lặp như vậy không thể cho một thời gian trễ cố định. Giải pháp tốt hơn là sử dụng mạch định thời. Như chúng tôi đã nói, bộ định thời tạo ra 18,2 nhịp trong một giây.

Để tính giờ chúng ta cần viết một thủ tục ngắt thời gian, thủ tục này thực hiện những công việc sau: mỗi khi bộ định thời gõ nhịp nó sẽ đặt biến TIMER_FLAG bằng 1. Thủ tục chuyển quả bóng sẽ kiểm tra biến này để xem bộ định thời đã gõ nhịp chưa, nếu rồi thủ tục sẽ di chuyển quả bóng và xoá cờ TIMER_FLAG về 0. Thủ tục ngắt thời gian được cho trong chương trình nguồn PGM16_2.ASM.

Chương trình nguồn PGM16_2B.ASM

```
TITLE PGM16_2B:  TIMER_TICK
;thủ tục ngắt thời gian
EXTRN TIMER_FLAG:BYTE
PUBLIC    TIMER_TICK
.MODEL     SMALL
.CODE
;ngắt thời gian mới
TIMER_TICK PROC
;cắt các thanh ghi
    PUSH DS      ;cắt DS
    PUSH AX
;
    MOV AX, SEG TIMER_FLAG ;lấy địa chỉ đoạn của
                            ;biến TIMER_FLAG
    MOV DS, AX        ;chứa nó trong DS
    MOV TIMER_FLAG, 1 ;thiết lập cờ
;phục hồi các thanh ghi
    POP  AX
    POP  DS        ;khôi phục lại chương trình
    IRET
TIMER_TICK ENDP      ;kết thúc chương trình
END
```

Quả bóng dội

Nếu chúng ta cứ chuyển quả bóng theo một phương cố định, cuối cùng quả bóng sẽ ra ngoài đường biên. Để giữ quả bóng trong vùng xác định, chúng ta sẽ vẽ nó dội ra khi chạm vào đường biên. Trước hết chúng ta phải kiểm tra mỗi vị trí mới trước khi vẽ quả bóng. Nếu vị trí mới nằm ngoài đường biên, chúng ta chỉ đơn giản đặt vị trí của quả bóng ở ngay trên đường biên đồng thời, chúng ta đổi dấu một thành phần của vận tốc sẽ làm cho quả bóng chuyển động ra ngoài.

Nhờ đó quả bóng sẽ chuyển động ngược lại đường như nó bị dội ra từ đường biên. Thủ tục CHECK_BOUNDARY trong chương trình nguồn PGM16_2C.ASM kiểm tra các điều kiện biên của quả bóng và thay đổi các thành phần của vận tốc một cách phù hợp.

Cùng với thủ tục kiểm tra biên, chúng ta viết thủ tục MOV_BALL để đợi một nhịp đồng hồ và di chuyển quả bóng. Thủ tục MOV_BALL trước tiên sẽ xoá quả bóng tại vị trí hiện thời cho bởi CX,DX; sau đó nó tính toạ độ mới của quả bóng bằng cách cộng tốc độ và gọi thủ tục CHECK_BOUNDARY để kiểm tra vị trí mới. Cuối cùng nó kiểm tra cờ TIMER_FLAG xem đã qua một nhịp đồng hồ chưa để hiển thị quả bóng ở vị trí mới. Thủ tục MOV_BALL được viết trong chương trình nguồn PGM16_2C.ASM.

Chương trình nguồn PGMS16_2C.ASM

```
TITLE PGM16_2C:  
;các thủ tục MOV_BALL và CHECK_BOUNDARY  
    EXTRN DISPLAY_BALL:NEAR  
    EXTRN TIMER_FLAG:BYTE, VEL_X:WORD, VEL_Y:WORD  
    PUBLIC      MOV_BALL  
.MODEL      SMALL  
.CODE  
MOV_BALL    PROC  
;xoá quả bóng tại vị trí hiện thời và  
;vẽ nó ở một vị trí mới  
;  
;      Vào: CX=toạ độ cột  
;              DX=toạ độ dòng  
;xoá quả bóng  
    MOV AL,0           ;màu 0 là màu nền  
    CALL DISPLAY_BALL ;xoá quả bóng  
;xác định vị trí mới  
    ADD CX,VEL_X  
    ADD DX,VEL_Y  
;kiểm tra biên  
    CALL CHECK_BOUNDARY  
;đợi một nhịp đồng hồ  
    CMP TIMER_FLAG,1   ;đã qua một nhịp?  
    JNE TEST_TIMER    ;chưa, tiếp tục kiểm tra!  
    MOV TIMER_FLAG,0   ;rồi, thiết lập lại cờ!  
    MOV AL,3           ;màu trắng  
    CALL DISPLAY_BALL ;hiển thị quả bóng  
    RET  
MOV_BALL    ENDP  
  
CHECK_BOUNDARY    PROC
```

```

;kiểm tra xem quả bóng có ra ngoài đường biên
;hay không nếu có, chuyển nó ngược lại và đổi hướng
;chuyển động
;
;      Vào: CX=toa độ cột của quả bóng
;              DX=toa độ dòng của quả bóng
;      Ra:   CX=toa độ cột mới của quả bóng
;              DX=toa độ dòng mới của quả bóng
;kiểm tra giá trị cột
    CMP CX,11          ;nằm ngoài lề trái ?
    JG   L1             ;không, kiểm tra lề phải!
    MOV  CX,11          ;đúng, đặt lại bằng 11
    NEG  VEL_X          ;đổi hướng chuyển động
    JMP  L2             ;kiểm tra giá trị hàng
L1:   CMP  CX,298       ;ngoài lề phải?
    JL   L2             ;không, kiểm tra giá trị hàng
    MOV  CX,298          ;đúng, đặt lại toa độ cột bằng 298
    NEG  VEL_X          ;đổi hướng chuyển động
    ;kiểm tra các giá trị hàng
L2:   CMP  DX,11          ;nằm ngoài đường biên trên?
    JG   L3             ;không, kiểm tra đường biên dưới
    MOV  DX,11          ;đặt lại bằng 11
    NEG  VEL_Y          ;đổi hướng chuyển động
    JMP  DONE            ;xong
L3:   CMP  DX,187         ;nằm dưới đường biên dưới?
    JL   DONE            ;không, kết thúc
    MOV  DX,187          ;thiết lập lại bằng 187
    NEG  VEL_Y          ;đổi hướng chuyển động
DONE:
    RET
CHECK_BOUNDARY ENDP
END

```

Bây giờ chúng ta đã sẵn sàng để viết thủ tục chính. Chương trình của chúng ta sẽ sử dụng thủ tục SETUP_INT trong chương trình nguồn PGM15_2A.ASM trong chương 15 để đổi vector ngắn. Các bước của thủ tục chính sẽ như sau: (1) thiết lập chế độ đồ họa và thủ tục ngắn TIMER_TICK, (2) vẽ quả bóng ở lề phải với vận tốc sẽ làm cho quả bóng chuyển động lên phía trên và sang bên trái, (3) đợi một nhịp đồng hồ, (4) gọi thủ tục MOV_BALL để chuyển quả bóng, (5) đợi một nhịp đồng hồ tiếp theo để quả bóng ở lại trên màn hình một thời gian và (6) quay trở lại bước 3. Thủ tục chính được viết trong chương trình nguồn PGM16_2.ASM.

Chương trình nguồn PGM16_2.ASM

```

TITLE PGM16_2A: BOUNCING BALL
EXTRN SET_DISPLAY_MODE:NEAR, DISPLAY_BALL:NEAR

```

```

        EXTRN      MOV_BALL:NEAR,
        EXTERN SETUP_INT:NEAR,TIMER_TICK:NEAR
        PUBLIC     TIMER_FLAG,VEL_X,VEL_Y
.MODEL    SMALL
.STACK    100H
;
.DATA
NEW_TIMER_VEC    DW    ?,?
OLD_TIMER_VEC    DW    ?,?
TIMER_FLAG DB   0
VEL_X      DW    -6
VEL_Y      DW    -1
;
.CODE
MAIN PROC
        MOV AX,@DATA
        MOV DS,AX           ;khởi tạo DS
;
;thiết lập chế độ đồ họa và vẽ đường viền
        CALL SET_DISPLAY_MODE
;thiết lập thủ tục ngắt thời gian mới
        MOV NEW_TIMER_VEC, OFFSET TIMER_TICK ;offset
        MOV NEW_TIMER_VEC+2,CS      ;segment
        MOV AL,1CH                ;số hiệu ngắt
        LEA DI,OLD_TIMER_VEC      ;DI trả tới vùng đệm
                                    ;vector
        LEA SI,NEW_TIMER_VEC      ;SI trả tới
                                    ;vector ngắt mới
        CALL SETUP_INT
;quả bóng bắt đầu tại cột 298 dòng 100
        MOV CX,298
        MOV DX,100
        MOV AL,3                 ;quả bóng màu trắng
        CALL DISPLAY_BALL
;đợi một nhịp đồng hồ trước khi di chuyển quả bóng
TEST_TIMER:
        CMP TIMER_FLAG,1          ;đã qua một nhịp?
        JNE TEST_TIMER            ;chưa, tiếp tục kiểm tra!
        MOV TIMER_FLAG,0          ;rồi, xoá cờ!
        CALL MOV_BALL             ;chuyển đến vị trí mới
;trễ một nhịp đồng hồ
TEST_TIMER_2:
        CMP TIMER_FLAG,1;đã qua một nhịp đồng hồ?
        JNE TEST_TIMER2           ;chưa, tiếp tục đợi!
        MOV TIMER_FLAG,0          ;rồi, xoá cờ!
        JMP TEST_TIMER             ;tiếp tục quá trình
MAIN ENDP
END MAIN

```

Để chạy chương trình chúng ta cần liên kết các file object PGM16_2 + PGM15_2A + PGM16_2A + PGM16_2B + PGM16_2C. Cần nhắc các bạn một điều, chương trình của chúng ta vẫn không có cách nào để kết thúc, do đó có thể các bạn sẽ phải khởi động lại hệ thống nếu muốn kết thúc chương trình. Trong phần 16.6.2 chúng ta sẽ bàn đến phương pháp kết thúc chương trình.

16.6 Trò chơi video tương tác

Trong các phần sau chúng tôi sẽ xây dựng chương trình quả bóng dội thành một chương trình trò chơi video tương tác. Trước tiên trong phần 16.6.1 chúng tôi sẽ thêm âm thanh vào trong chương trình, mỗi khi quả bóng chạm vào tường thì một âm sắc sẽ được phát ra. Tiếp theo đó trong phần 16.6.2 chúng tôi sẽ thêm vào một cái vợt để người chơi có thể đánh quả bóng. Để đơn giản vợt sẽ chỉ trượt lên và xuống dọc theo đường biên trái và nó được điều khiển bằng các phím mũi tên lên và xuống. Nếu cây vợt đánh trượt quả bóng và để nó chạm vào lề trái trò chơi sẽ kết thúc. Trò chơi cũng có thể kết thúc bằng cách ấn phím ESC.

16.6.1 Thêm vào âm thanh

Máy PC có bộ tạo âm thanh mà chúng ta có thể định được tần số âm và thời gian kéo dài của âm. Tần số của âm có thể xác định bằng vi mạch định thời.

Vì mạch định thời được điều khiển bằng một mạch đồng hồ có tần số 1.193MHz. Tần số này nằm ngoài khoảng tần số mà tai người có thể nghe được. Tuy nhiên bộ định thời có thể tạo ra các tín hiệu với tần số nhỏ hơn. Để làm được điều đó nó tạo ra một xung mỗi khi có một số xác định N xung vào, trong N có thể thay đổi bằng chương trình. Đầu tiên số N được nạp vào bộ đếm sau đó mỗi khi đếm đủ N xung vào, mạch sẽ tạo ra một xung ra. Quá trình cứ như vậy được lặp lại cho đến khi một số đếm khác được đặt vào bộ đếm. Ví dụ bằng cách đặt giá trị 1193 vào bộ đếm, ở đầu ra chúng ta sẽ nhận được 1000 xung mỗi giây hay đầu ra có tần số 1000Hz.

Mặt khác cũng có thể định được thời gian kéo dài của âm được tạo ra. Để bắt đầu một âm chúng ta bật bộ định thời sau một thời gian vừa ý chúng ta phải tắt nó đi. Để tính giờ chúng ta có thể sử dụng thủ tục ngắn TIMER_TICK. Vì thủ tục TIMER_TICK được gọi 55ms một lần chúng ta sẽ tạo được thời gian trễ nửa giây bằng cách đợi 9 nhịp đồng hồ.

Để truy nhập mạch định thời chúng ta phải sử dụng các lệnh vào ra IN và OUT. Các lệnh này cho phép chuyển dữ liệu giữa các cổng vào ra và các thanh ghi AL hay AX. Để đọc một cổng 8 bit chúng ta có thể viết:

IN AL, Port

trong đó port là số hiệu của một cổng. Tương tự để viết ra một cổng vào ra 8 bit chúng ta sử dụng lệnh:

OUT port, AL

Có 3 cổng cần dùng đến ở đây: cổng 42h để nạp bộ đếm, cổng 43h để định các thao tác định thời và cổng 61h để khởi động mạch định thời.

Trước khi nạp số đếm vào cổng 42h chúng ta nạp cổng 43h với mã lệnh B6h. Lệnh này sẽ quy định mạch định thời tạo ra các xung vuông và cổng 42h sẽ được nạp số liệu từng byte một byte thấp trước byte cao sau. Các bit ở vị trí 0 và 1 trong cổng 61h điều khiển bộ định thời và đầu ra của nó. Bằng cách đặt chúng bằng 1, loa được kích hoạt.

Thủ tục tạo âm thanh, BEEP, tạo ra âm ở tần số 1000Hz trong nửa giây. Các bước thực hiện như sau: (1) nạp 1193 vào bộ đếm(cổng 42h), (2) kích hoạt loa, (3)cho tiếng bip kéo dài khoảng 500ms và (4) tắt loa. Thủ tục BEEP được viết trong chương trình nguồn PGM16_3A.ASM.

Chương trình nguồn PGM16_3A.ASM

```
TITLE PGM16_3A: BEEP
;thủ tục tạo âm thanh
    EXTRN TIMER_FLAG:BYTE
    PUBLIC      BEEP
.MODEL      SMALL
.CODE
BEEP PROC
;tạo tiếng kêu bip
    PUSH CX          ;cắt thanh ghi CX
; khởi tạo nhịp thời gian
    MOV  AL,0B6H      ;định chế độ làm việc
    OUT  43H,AL       ;viết ra cổng 43h
;nạp bộ đếm
    MOV  AX,1193      ;số đếm để tạo ra tần số 1000Hz
    OUT  42H,AL       ;byte thấp
    MOV  AL,AH         ;byte cao
    OUT  42H,AL
;kích hoạt loa
    IN   AL,61H        ;đọc cổng điều khiển
    MOV  AH,AL         ;giữ giá trị ban đầu trong AH
    OR   AL,11B         ;đặt các bit điều khiển
    OUT  61H,AL         ;kích hoạt loa
;vòng lặp trễ 500ms
    MOV  CX,9           ;lặp 9 lần
B_1: CMP   TIMER_FLAG,1      ;kiểm tra cờ định thời
    JNE   B_1             ;chưa thiết lập, quay lại!
```

```

    MOV  TIMER_FLAG, 0      ;cờ đã thiết lập, xoá nó!
    LOOP B_1                 ;lặp lại với nhịp tiếp theo
;tắt tiếng kêu
    MOV  AL,AH                ;trả lại giá trị điều khiển
                                ;vào AL
    OUT  61H,AL               ;tắt loa
    POX  CX
    RET
BEEP ENDP
END

```

Bây giờ chúng ta hãy viết một thủ tục di chuyển quả bóng mới có sử dụng thủ tục tạo âm thanh BEEP. Mỗi khi quả bóng chạm vào đường biên, thủ tục BEEP sẽ được gọi để tạo ra tiếng bip. Các thủ tục mới có tên gọi MOV_BALL_A và CHECK_BOUNDARY_A chúng được viết trong chương trình nguồn PGM16_3B.ASM.

Chương trình nguồn PGM16_3B.ASM

```

TITLE PGM16_3B: BALL MOVEMENT
EXTRN DISPLAY_BALL:NEAR, BEEP:NEAR
EXTRN TIMER_FLAG:BYTE, VEL_X:WORD, VEL_Y:WORD
PUBLIC MOVE_BALL_A
.MODEL SMALL
.CODE
MOVE_BALL_A PROC
; xoá quả bóng tại vị trí hiện thời và hiển thị nó
;tại vị trí mới
    MOV  AL,0
    CALL DISPLAY_BALL
    ADD  CX,VEL_X
    ADD  DX,VEL_Y
    CALL CHECK_BOUNDARY_A
TEST_TIMER_1:
    CMP  TIMER_FLAG,1
    JNE  TEST_TIMER_1      ;chưa, kiểm tra
    MOV  TIMER_FLAG,0       ;rồi xoá cờ
    MOV  AL,3                ;màu trắng
    CALL DISPLAY_BALL       ;hiển bóng
    RET
MOVE_BALL_A ENDP
;
CHECK_BOUNDARY_A PROC
;kiểm tra xem quả bóng có ra ngoài đường biên
;hay không. Nếu có, chuyển nó ngược lại và đổi
;hướng chuyển động

```

```

;
; Vào: CX=toa độ cột của quả bóng
;       DX=toa độ dòng của quả bóng
; Ra:   CX=toa độ cột mới của quả bóng
;       DX=toa độ dòng mới của quả bóng
; kiểm tra giá trị cột
    CMP CX,11           ;nằm ngoài lề trái ?
    JG   L1              ;không, kiểm tra lề phải!
    MOV  CX,11           ;đúng, đặt lại bằng 11
    NEG  VEL_X          ;đổi hướng chuyển động
    CALL BEEP
    JMP  L2              ;kiểm tra giá trị hàng
L1:   CMP  CX,299        ;ngoài lề phải?
    JL   L2              ;không, kiểm tra giá trị hàng
    MOV  CX,299          ;đúng, đặt lại toa độ cột bằng 299
    NEG  VEL_X          ;đổi hướng chuyển động
    CALL BEEP
; kiểm tra các giá trị hàng
L2:   CMP  DX,11          ;nằm ngoài đường biên trên?
    JG  L3               ;không, kiểm tra đường biên dưới
    MOV  DX,11           ;đặt lại bằng 11
    NEG  VEL_Y          ;đổi hướng chuyển động
    CALL BEEP
    JMP  DONE             ;xong
L3:   CMP  DX,188         ;nằm dưới đường biên dưới?
    JL  DONE             ;không, kết thúc
    MOV  DX,188          ;thiết lập lại bằng 188
    NEG  VEL_Y          ;đổi hướng chuyển động
    CALL BEEP             ;tiếng beep
DONE:
    RET
CHECK_BOUNDARY_A ENDP
END

```

16.6.2 Thêm vào chiếc vợt

Bây giờ chúng ta hãy thêm vào chương trình một chiếc vợt. Chiếc vợt sẽ di chuyển lên xuống dọc theo đường biên dọc khi người sử dụng ấn các phím mũi tên lên và xuống.

Do chương trình không biết được khi một phím được ấn, chúng ta phải viết một thủ tục ngắt mới cho ngắt 9-ngắt bàn phím. Thủ tục ngắt này khác với thủ

tục trong chương 15 ở chỗ nó truy nhập cổng vào ra bàn phím và giữ lại mã scan.

Chúng ta cần truy nhập đến 3 cổng vào ra. Khi bàn phím tạo ngắn, bit 5 của cổng vào ra 20h được thiết lập và mã quét được đưa vào cổng 60h. Bit 7 của cổng 61h được dùng để cho phép bàn phím. Như vậy thủ tục ngắn của chúng ta sẽ đọc dữ liệu ở cổng 60h, cho phép bàn phím bằng cách thiết lập bit 7 của cổng 61h và xoá bit 5 của cổng 20h.

Thủ tục ngắn bàn phím có tên gọi KEYBOARD_INT. Khi thủ tục này nhận mã scan, trước hết nó sẽ kiểm tra xem đó là mã ấn phím hay mã nhả phím. Nếu đó là mã ấn phím nó sẽ thiết lập cờ KEY_FLAG và đưa mã scan vào biến SCAN_CODE. Trong trường hợp đó là mã nhả phím, các biến sẽ không bị thay đổi. Thủ tục KEYBOARD_INT được viết trong chương trình nguồn PGM16_3C.ASM.

Chương trình nguồn PGM16_3C.ASM

```
TITLE PGM16_3C: Keyboard interrupt
EXTRN SCAN_CODE:BYTE, KEY_FLAG:BYTE
PUBLIC      KEYBOARD_INT
.MODEL      SMALL
.CODE
KEYBOARD_INT    PROC
;thường trình phục vụ ngắn bàn phím mới
;cắt các thanh ghi
    PUSH DS
    PUSH AX
;thiết lập DS
    MOV AX, SEG SCAN_CODE
    MOV DS, AX
;nhận mã scan
    IN AL, 60H      ;đọc mã scan
    PUSH AX          ;cắt nó đi
    IN AL, 61H      ;giá trị điều khiển cổng
    MOV AH, AL        ;cắt vào AH
    OR AL, 80H       ;thiết lập bit cho phép bàn phím
    OUT 61H, AL      ;ghi trở lại
    XCHG AH, AL      ;lấy ra giá trị điều khiển ban đầu
    OUT 61H, AL      ;thiết lập lại điều khiển cổng
    POP AX           ;phục hồi mã scan
    MOV AH, AL        ;chứa nó vào AH
    TEST AL, 80H      ;có phải mã nhả phím?
    JNE KEY_0         ;đúng, xoá cờ, nhảy đến KEY_0
;mã ấn phím
    MOV SCAN_CODE, AL ;cắt vào biến
```

```

        MOV    KEY_FLAG, 1      ;thiết lập cờ
KEY_0:
        MOV    AL, 20H      ;thiết lập lại ngắt
        OUT   20H, AL
;phục hồi các thanh ghi
        POP   AX
        POP   DS
        IRET
KEYBOARD_INT ENDP      ;kết thúc thủ tục ngắt
END

```

Bây giờ chúng ta sẽ thêm chiếc vợt vào cột 11, và sử dụng các phím mũi tên lên và xuống để di chuyển nó. Nếu quả bóng đến cột 11 mà vợt không ở vị trí để có thể đánh quả bóng thì chương trình sẽ kết thúc. Chiếc vợt được tạo lên bởi 10 điểm ảnh, vị trí ban đầu là từ dòng 45 đến dòng 54. Chúng ta sử dụng 2 biến PADDLE_TOP và PADDLE_BOTTOM để theo dõi vị trí hiện thời của vợt.

Chúng ta cần 2 thủ tục DRAW_PADDLE để vẽ và xoá vợt, MOVE_PADDLE để di chuyển vợt lên xuống. Cả 2 thủ tục đều viết trong chương trình nguồn PGM16_3D.ASM.

Chương trình nguồn PGM16_3D.ASM

```

TITLE PGM16_3D: PADDLE CONTROL
;các thủ tục MOVE_PADDLE và DRAW_PADDLE
        EXTRN PADDLE_TOP:WORD, PADDLE_BOTTOM:WORD
        PUBLIC      DRAW_PADDLE, MOVE_PADDLE
.MODEL      SMALL
.CODE
DRAW_PADDLE PROC
;vẽ vợt tại cột 11
;Vào: AL chứa trị số điểm ảnh, 2 (đỏ) khi
;vẽ và 0 (xanh) khi xoá
;cắt các thanh ghi
        PUSH CX
        PUSH DX
        MOV AH, 0CH      ;hàm ghi điểm ảnh
        MOV CX, 11       ;tại cột 11
        MOV DX, PADDLE_TOP ;dòng định
.DP1:
        INT 10H
        INC DX           ;dòng tiếp theo
        CMP DX, PADDLE_BOTTOM ;xong?
        JLE DP1          ;chưa, tiếp tục!
;khôi phục lại thanh ghi
        POP DX
        POP CX

```

```

        RET
DRAW_PADDLE      ENDP
;
MOVE_PADDLE PROC
;di chuyển chuyển vợt lên và xuống
;Vào: AX=2 chuyển vợt xuống 2 điểm ảnh
;      AX=-2 chuyển vợt lên 2 điểm ảnh
;
MOV BX,AX ;chép giá trị này vào BX
;kiểm tra hướng
CMP AX,0
JL UP           ;âm, chuyển lên phía trên
;chuyển xuống dưới, kiểm tra vị trí của vợt
CMP PADDLE_BOTTOM,188 ;vợt nằm ở đáy?
JGE DONE         ;đúng, không thể di chuyển được nữa
JMP UPDATE       ;không, cập nhật vị trí vợt
;chuyển vợt lên trên, kiểm tra xem nó có nằm ở đỉnh
UP: CMP PADDLE_TOP,11 ;ở đỉnh?
JLE DONE         ;đúng, không thể di chuyển
;di chuyển vợt
UPDATE:
;xoá vợt
MOV AL,0          ;màu xanh lá cây
CALL DRAW_PADDLE
;thay đổi vị trí của vợt
ADD PADDLE_TOP, BX
ADD PADDLE_BOTTOM,BX
;vẽ vợt ở vị trí mới
MOV AL,2          ;màu đỏ
CALL DRAW_PADDLE
DONE: RET
MOVE_PADDLE      ENDP
END

```

Thủ tục MOVE_PADDLE có thể chuyển vợt lên trên cũng như xuống dưới 2 điểm ảnh tùy thuộc vào giá trị của AX là âm hay dương. Tuy nhiên khi vợt nằm ở trên cùng nó sẽ không thể di lên được nữa và khi nó ở dưới cùng nó cũng không thể di xuống được nữa. Đến giờ chúng ta đã có thể viết thủ tục chính.

Chương trình nguồn PGM16_3.ASM

```

TITLE PGM16_3:    PADDLE_BALL
EXTRN SET_DISPLAY_MODE:NEAR, DISPLAY_BALL:NEAR
EXTRN MOV_BALL_A:NEAR,DRAW_PADDLE:NEAR
EXTRN MOVE_PADDLE:NEAR
EXTRN KEYBOARD_INT:NEAR, TIMER_TICK:NEAR
PUBLIC     TIMER_FLAG,KEY_FLAG, SCAN_CODE
PUBLIC     PADDLE_TOP, PADDLE_BOTTOM, VEL_X, VEL_Y
;

```

```

.MODEL      SMALL
.STACK     100H
.DATA
;
NEW_TIMER_VEC    DW    ?,?
OLD_TIMER_VEC    DW    ?,?
NEW_KEY_VEC DW    ?,?
OLD_KEY_VEC DW    ?,?
SCAN_CODE   DB    0
KEY_FLAG     DB    0
TIMER_FLAG    DB    0
PADDLE_TOP    DW    45
PADDLE_BOTTOM DW    54
VEL_X        DW    -6
VEL_Y        DW    -1
;các mã scan
UP_ARROW=72
DOWN_ARROW=80
ESC_KEY=1
;
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX ;khởi tạo DS
;thiết lập chế độ đồ họa
    CALL SET_DISPLAY_MODE
;vẽ vợt
    MOV AL, 2           ;màu đỏ
    CALL DRAW_PADDLE
;tạo vector ngắt thời gian mới
    MOV NEW_TIMER_VEC, OFFSET TIMER_TICK ;offset
    MOV NEW_TIMER_VEC+2, CS      ;segment
    MOV AL, 1CH            ;số hiệu ngắt
    LEA DI, OLD_TIMER_VEC
    LEA SI, NEW_TIMER_VEC
    CALL SETUP_INT
;tạo ngắt bàn phím mới
    MOV NEW_KEY_VEC, OFFSET KEYBOARD_INT ;offset
    MOV NEW_KEY_VEC+2, CS      ;segment
    MOV AL, 09H            ;số hiệu ngắt
    LEA DI, OLD_KEY_VEC
    LEA SI, NEW_KEY_VEC
    CALL SETUP_INT
;ban đầu quả bóng ở cột 298 dòng 100
    MOV CX, 298           ;cột
    MOV DX, 100           ;dòng
    MOV AL, 3             ;màu trắng
    CALL DISPLAY_BALL
;kiểm tra cờ bàn phím
TEST_KEY:
    CMP KEY_FLAG, 1       ;kiểm tra cờ bàn phím
    JNE TEST_TIMER        ;chưa thiết lập, kiểm
                           ;tra cờ định thời
    MOV KEY_FLAG, 0        ;đã thiết lập, xoá nó và
                           ;kiểm tra xem
    CMP SCAN_CODE, ESC_KEY

```

```

        JNE  TK_1           ;có phải phím ESC không
                                ;không, kiểm tra xem có
                                ;phải phím mũi tên không
        JMP  DONE           ;đúng, kết thúc

TK_1:
        CMP  SCAN_CODE, UP_ARROW   ;mũi tên lên
        JNE  TK_2           ;không, kiểm tra xem có phải
                                ;mũi tên xuống
        MOV  AX, -2          ;đúng là mũi tên lên, chuyển
                                ;lên 2 điểm ảnh
        CALL  MOVE_PADDLE
        JMP  TEST_TIMER      ;di kiểm tra cờ định thời

TK_2:
        CMP  SCAN_CODE, DOWN_ARROW ;mũi tên xuống?
        JNE  TEST_TIMER       ;không, kiểm tra cờ định thời
        MOV  AX, 2            ;đúng, chuyển xuống 2 điểm ảnh
        CALL  MOVE_PADDLE
;kiểm tra cờ định thời
TEST_TIMER:
        CMP  TIMER_FLAG, 1      ;cờ đã thiết lập?
        JNE  TEST_KEY         ;không, kiểm tra cờ bàn phím
        MOV  TIMER_FLAG, 0      ;đúng! Xoá nó
        CALL  MOV_BALL_A       ;chuyển quả bóng đến
                                ;vị trí mới
;kiểm tra xem vợt có trượt bóng?
        CMP  CX, 11            ;bóng ở cột 11?
        JNE  TEST_KEY         ;không, kiểm tra cờ bàn phím
        CMP  DX, PADDLE_TOP    ;đúng, kiểm tra vị trí vợt
        JL   CP_1              ;trượt, bóng ở trên vợt
        CMP  DX, PADDLE_BOTTOM
        JG   CP_1              ;trượt, bóng ở dưới vợt
;bóng chạm vợt, đợi một nhịp đồng hồ sau đó
;chuyển bóng và vẽ lại vợt
DELAY:
        CMP  TIMER_FLAG, 1      ;đã qua một nhịp?
        JNE  DELAY             ;chưa, đợi!
        MOV  TIMER_FLAG, 0      ;đúng, thiết lập lại cờ
        CALL  MOV_BALL_A       ;vẽ vợt màu đỏ
        CALL  DRAW_PADDLE
        JMP  TEST_KEY          ;kiểm tra cờ bàn phím
;vợt trượt bóng, xoá bóng và kết thúc
CP_1:
        MOV  AL, 0              ;xoá bóng
        CALL  DISPLAY_BALL
;phục hồi vector ngắt thời gian
DONE:
        LEA   DI, NEW_TIMER_VEC
        LEA   SI, OLD_TIMER_VEC
        MOV  AL, 1CH
        CALL  SETUP_INT
;phục hồi vector ngắt bàn phím
        LEA   DI, NEW_KEY_VEC
        LEA   SI, OLD_KEY_VEC
        MOV  AL, 09H

```

```

CALL  SETUP_INT
;đợi ấn một phím
MOV  AH,0
INT  16H
;thiết lập lại chế độ văn bản
MOV  AH,0
MOV  AL,3
INT  10H
;DOS exit
MOV  AH,4CH
INT  21H
MAIN  ENDP
END  MAIN

```

Trong thủ tục chính chúng ta đã thay đổi việc kiểm tra cờ bàn phím và cờ định thời. Khi cờ bàn phím được thiết lập, chúng ta kiểm tra mã scan: (1) phím ESC sẽ làm kết thúc chương trình, (2) phím mũi tên lên sẽ chuyển vẹt lên phía trên, (3) phím mũi tên xuống sẽ chuyển vẹt xuống và (4) các phím khác sẽ bị bỏ qua. Nếu cờ định thời thiết lập chúng ta sẽ gọi thủ tục MOV_BALI_A để chuyển quả bóng đến vị trí mới và nếu bóng đến cột 11 nhưng vẹt trượt bóng chúng ta sẽ kết thúc chương trình.

Để kết thúc chương trình trước tiên chúng ta phục hồi các vector ngắn và đợi ấn một phím bất kỳ. Khi có một phím được ấn, chúng ta thiết lập lại chế độ văn bản và quay trở về DOS.

TỔNG KẾT

Trong chương này chúng ta đã nghiên cứu những vấn đề sau:

- ◆ Các phần tử màn hình trong chế độ đồ họa được gọi là các điểm ảnh
- ◆ Các bộ phối ghép màn hình phổ biến của IBM PC là CGA, EGA và VGA.
- ◆ Ngắt 10h dùng để quản lý tất cả các thao tác với màn hình
- ◆ CGA có chế độ phân giải trung bình với 320×200 điểm và chế độ phân giải cao với 640×200 điểm.
- ◆ EGA có tất cả các chế độ của CGA thêm vào đó là chế độ với độ phân giải 640×350 .
- ◆ VGA có tất cả các chế độ của EGA thêm vào đó là chế độ 640×480 điểm. Nó còn có thể hiển thị 256 màu với độ phân giải 320×200 .
- ◆ Kỹ thuật làm hình chuyển động chính là xoá đi một vật thể và vẽ lại nó tại vị trí mới.
- ◆ Có thể tạo âm thanh bằng cách thao tác với các cổng vào ra.
- ◆ Lập trình một trò chơi video tương tác cần phải hoán đổi ngắt bàn phím.

Các thuật ngữ tiếng Anh

Analog monitor

Màn hình tương tự - màn hình có thể nhận các tín hiệu màu với nhiều mức.

APA(All Points Addressable) Chế độ đồ họa trong đó một điểm ảnh tương ứng với một điểm quét trên màn hình.

Background color

Màu nền - màu mặc định của các điểm ảnh.

Bit planes

Các mặt bit - các mô đun bộ nhớ có cùng địa chỉ.

ECD(Enhanced Color Display) monitor

Màn hình ECD - màn hình có khả năng hiển thị tất cả các chế độ của EGA

Palette

Bảng màu - tập tất cả các màu có thể thể hiện cùng lúc.

Pixel(Picture Element)

Điểm ảnh.

Scan lines

Dòng quét - các dòng trên màn hình được chùm tia điện tử quét lên.

Các lệnh mới:

IN

OUT

Bài tập

1. Viết các chỉ thị chọn chế độ đồ họa 320×200 với 16 màu.
2. Viết các chỉ thị chọn bảng màu 0 với màu nền trắng trong chế độ phân giải trung bình của CGA.
3. Viết các chỉ thị vẽ một hình chữ nhật màu xanh lá cây kích thước 10×10 có góc trái trên nằm ở cột 150 dòng 100. Sử dụng chế độ phân giải trung bình của CGA với màu nền trắng.
4. Viết các chỉ thị đổi hình chữ nhật nói trên thành hình chữ nhật màu lục trên nền trắng.

Các bài tập lập trình.

5. Hãy thêm một chiếc vợt tại cột 299 vào chương trình trò chơi trong chương này nhờ đó biến thành trò chơi cho 2 người.
6. Hãy sửa chương trình trò chơi trong chương này để quả bóng sẽ chuyển động với vận tốc giảm khi nó gặp đường viền nhưng sẽ chuyển động với vận tốc tăng khi nó chạm vào vợt.

Chương 17

ĐỆ QUY

Tổng quan

Thủ tục đệ quy là một thủ tục gọi chính nó. Các thủ tục đệ quy có vai trò rất quan trọng trong các ngôn ngữ bậc cao, nhưng cách thức hệ thống sử dụng ngắn xếp để thực hiện các thủ tục này vẫn là bí ẩn đối với người sử dụng. Trong Hợp ngữ, người sử dụng phải lập trình cho các thao tác ngắn xếp. Điều này tạo cơ hội cho ta thấy được thực chất sự đệ quy diễn ra như thế nào.

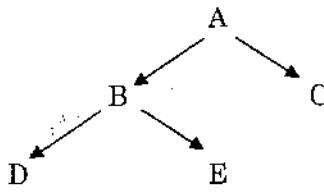
Bởi vì bạn có thể chỉ có một chút ít kinh nghiệm về đệ quy, các phần 17.1-17.2 sẽ trình bày các ý tưởng cơ sở. Phần 17.3 chỉ ra có thể dùng ngắn xếp để truyền dữ liệu cho thủ tục như thế nào; đề tài này cũng đã được thảo luận trong chương 14. Đến các phần 17.4-17.5 chúng ta ứng dụng phương pháp đó để thực hiện các thủ tục đệ quy gọi chính nó một lần. Cuối chương chúng ta sẽ tìm hiểu về các thủ tục có các lời gọi đệ quy nhiều lần.

17.1. Ý tưởng đệ quy

Một quá trình được gọi là đệ quy nếu nó được định nghĩa trong chính lúc thực hiện. Để làm ví dụ, chúng ta xem xét định nghĩa cây nhị phân sau đây:

Một cây nhị phân hoặc là rỗng hoặc gồm có một phần tử đơn gọi là gốc và các phần tử còn lại của nó được chia thành hai nhánh riêng rẽ (nhánh trái và nhánh phải) mà mỗi trong chúng lại là một cây nhị phân.

Chúng ta hãy dùng định nghĩa để chứng tỏ cây T sau đây là một cây nhị phân:



Chọn A là gốc của T. Cây T1 bao gồm B, D và E là nhánh trái của A và cây T2 chứa C là nhánh phải. Chúng ta phải chứng minh rằng các cây T1 và T2 là các cây nhị phân.

Chọn B là gốc của T1. Cây T1a chứa D và cây T1b chứa E là các nhánh trái và phải. Chúng ta phải chứng minh rằng các cây T1a và T1b là các cây nhị phân.

Chọn D là gốc của T1a. Các nhánh trái và phải của D là rỗng. Do cây rỗng là một cây nhị phân nên T1a là một cây nhị phân. Tương tự ta có T1b là một cây nhị phân. Vì T1a và T1b là các cây nhị phân nên T1 phải là một cây nhị phân.

Bây giờ đến T2. Nó có nhánh C có các nhánh trái và phải là rỗng do đó nó cũng là một cây nhị phân.

Ta đã chỉ ra được T1 và T2 là các cây nhị phân, từ đây suy ra T cũng là một cây nhị phân.

Ví dụ trên minh họa các tính chất chính của các quá trình đệ quy:

1. Vấn đề chính (T là một cây nhị phân) chia ra thành các vấn đề nhỏ hơn (T1 và T2 là các cây nhị phân) và cách thức giải quyết các vấn đề con này giống hệt như vấn đề chính.
2. Phải có một điểm dừng (cây rỗng là cây nhị phân) để kết thúc đệ quy.
3. Một khi vấn đề con được giải quyết (T1 được chứng tỏ là cây nhị phân), công việc tiếp tục với bước tiếp theo của vấn đề nguyên thuỷ (chứng tỏ T2 là một cây nhị phân).

17.2. Các thủ tục đệ quy

Một thủ tục đệ quy lại gọi thi hành chính nó. Cũng giống như ví dụ đầu tiên, ta hãy xem xét cách tính giai thừa của một số nguyên dương. Nó được định nghĩa một cách không đệ quy như sau:

```

FACTORIAL(1) = 1
FACTORIAL(N) = N × (N-1) × (N-2) × ... × 2 × 1
với N > 1.
  
```

Từ đó:

$$\text{FACTORIAL}(N-1) = (N-1) \times (N-2) \times \dots \times 2 \times 1$$

Chúng ta có thể viết định nghĩa đệ quy sau đây:

$$\begin{aligned} \text{FACTORIAL}(1) &= 1; \\ \text{FACTORIAL}(N) &= N \times \text{FACTORIAL}(N-1) \end{aligned} \quad \text{với } N > 1.$$

Chúng ta hãy viết lại định nghĩa trên như là thuật toán của thủ tục đệ quy FACTORIAL:

```
1: PROCEDURE FACTORIAL ( vào: N, ra: RESULT )
2: IF N = 1
3:   THEN
4:     RESULT = 1
5:   ELSE
6:     call FACTORIAL ( vào: N-1,      ra:RESULT)
7:     RESULT = N x RESULT
8: END_IF
9: RETURN
```

Giá trị RESULT ở bên phải dòng 7 là giá trị trả về bởi lời gọi thủ tục FACTORIAL ở dòng 6.

Với $N = 4$ chúng ta có thể chạy từng bước thủ tục như sau:

```
call FACTORIAL(4, RESULT) /* bắt đầu lần gọi đầu tiên */
call FACTORIAL(3, RESULT) /* bắt đầu lần gọi thứ hai */
call FACTORIAL(2, RESULT) /* bắt đầu lần gọi thứ ba */
call FACTORIAL(1, RESULT) /* bắt đầu lần gọi thứ tư */
RESULT = 1
RETURN                      * kết thúc lần gọi thứ tư */
```

Lần gọi thứ tư là điểm thoát. Khi nó kết thúc, lần gọi thứ ba được tiếp tục ở dòng 7:

RESULT = N x RESULT

Về phải có $N = 2$ và RESULT bằng 1 là giá trị được tính toán trong lần gọi thứ tư. Ta tính toán:

RESULT = 2 x 1 = 2.

và lần gọi này kết thúc. Thủ tục tiếp tục với lần gọi thứ hai ở dòng 7. Trong lần gọi này $N = 3$, như vậy:

$$\text{RESULT} = 3 \times \text{RESULT} = 3 \times 2 = 6.$$

phép tính trên đã kết thúc lần gọi thứ hai. Cuối cùng thủ tục tiếp tục lần gọi thứ nhất ở dòng 7. Trong lần gọi này $N = 4$, như vậy kết quả sẽ là:

$$\text{RESULT} = 4 \times \text{RESULT} = 4 \times 6 = 24.$$

Đây là giá trị trả về của thủ tục.

Thủ tục trên đây có các tính chất của một quá trình xử lý đệ quy mà chúng ta đã chỉ ra trong ví dụ cây nhị phân. Mỗi lần gọi thủ tục FACTORIAL làm việc như một version đơn giản hơn của vấn đề ban đầu (tính giai thừa của một số nhỏ hơn), có một điểm thoát (giai thừa của 1) và mỗi khi lần gọi được hoàn thành, công việc tiếp tục với lần gọi trước đó.

Tương tự như ví dụ thứ hai, chúng ta hãy xem xét vấn đề tìm số lớn nhất của một mảng A gồm N số nguyên. Với $N = 1$, số lớn nhất chính là số nguyên duy nhất của mảng A[1]. Nếu $N > 1$, số lớn nhất hoặc là A[N] hoặc là số lớn nhất trong các số A[1] ... A[N-1]. Sau đây là thuật toán:

```
1: PROCEDURE FIND_MAX ( vào: N, ra: MAX )
2: IF N = 1
3: THEN
4:     MAX = A[ 1 ]
5: ELSE
6:     call FIND_MAX ( N-1, MAX )
7:     IF A[ N ] > MAX
8:     THEN
9:         MAX = A[ N ]
10:    ELSE
11:        MAX = MAX
12:    END_IF
13: END_IF
14: RETURN
```

Trong các dòng 7 và 11, giá trị của MAX ở vế phải là giá trị trả về của lần gọi ở dòng 6.

Chúng ta hãy chạy từng bước thủ tục cho mảng A có 4 phần tử: 10, 50, 20, 4.

```

call FIND_MAX ( 4, MAX )          /* lần gọi đầu tiên */
call FIND_MAX ( 3, MAX )          /* lần gọi thứ hai */
call FIND_MAX ( 2, MAX )          /* lần gọi thứ ba */
call FIND_MAX ( 1, MAX )          /* lần gọi thứ tư */

```

Giống như trong ví dụ tính giai thừa, lần gọi thứ tư là điểm thoát. Nó trả về giá trị MAX = A[1] = 10 và kết thúc.

Bây giờ tiếp tục đến lần gọi thứ ba ở dòng 7. Bởi vì A[2] (= 50) > MAX (= 10), giá trị trả về của lần gọi này sẽ là MAX = 50.

Bước tiếp theo là tiếp tục lần gọi thứ hai ở dòng 7. Bởi vì A[3] (= 20) < MAX (= 50), giá trị trả về của lần gọi này vẫn là MAX = 50.

Cuối cùng chúng ta quay về lần gọi đầu tiên ở dòng 7. Bởi vì A[4] (= 4) < MAX (= 50), lần gọi này trả về MAX = 50. Đây chính là giá trị thủ tục trả về chương trình gọi nó.

17.3. Truyền các tham số bằng ngăn xếp

Như là chúng ta sẽ xem sau đây, các thủ tục đệ quy được thực hiện trong Hợp ngũ bằng cách truyền các tham số thông qua ngăn xếp. Để thấy rõ điều đó được tiến hành như thế nào chúng ta hãy xem xét một chương trình đơn giản sau đây. Chương trình đưa nội dung của hai từ nhớ vào trong ngăn xếp và gọi thủ tục ADD_WORD để trả về tổng của chúng trong AX.

Chương trình nguồn PGM17_1.ASM

```

0:   TITLE PGM17_1: ADD_WORD
1:   .MODEL SMALL
2:   .STACK 100h
3:   .DATA
4:   WORD1 DW 2
5:   WORD2 DW 5
6:   .CODE
7:   MAIN PROC
8:       MOV AX, @DATA
9:       MOV DS, AX
10:      PUSH WORD1
11:      PUSH WORD2
12:      CALL ADD_WORD
13:      MOV AH, 4CH
14:      INT 21H           ;trở về DOS
15: ;

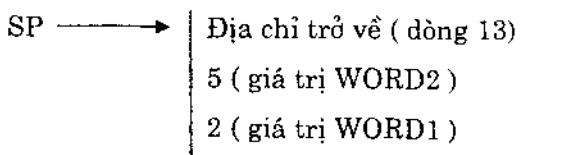
```

```

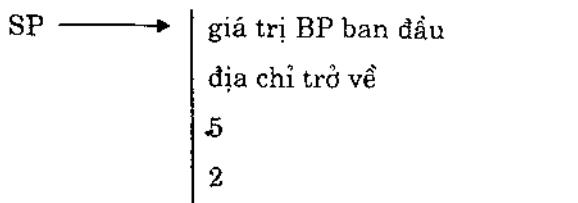
16: ADD_WORD PROC NEAR
17: ; cộng hai từ nhớ
18: ; số liệu vào trong ngăn xếp: địa chỉ trả
; về (định), word1, word2
19: ; ra      :AX=tổng
20:         PUSH   BP
21:         MOV    BP, SP
22:         MOV    AX,[ BP+6]
23:         ADD    AX,[ BP+4]      ; AX chứa tổng
24:         POP    BP            ; phục hồi BP
25:         RET    4             ; thoát
26: ADD_WORD    ENDP
27: END    MAIN

```

Sau khi khởi tạo DS, chương trình đưa nội dung câu WORD1 và WORD2 và ngăn xếp, sau đó gọi ADD_WORD. Khi vào thủ tục ngăn xếp như sau:



Tại các dòng 20-21, thủ tục cất giá trị ban đầu của BP vào trong ngăn xếp và thiết lập BP trả đến định ngăn xếp. Kết quả là:



Bây giờ dữ liệu có thể truy nhập bằng địa chỉ gián tiếp. BP được dùng bởi hai lý do: (1) khi sử dụng BP trong chế độ địa chỉ gián tiếp, SS là thanh ghi đoạn mặc định; và (2) bản thân SP có thể không dùng được trong chế độ địa chỉ gián tiếp. Tại dòng 22, địa chỉ thu được của toán hạng nguồn trong lệnh:

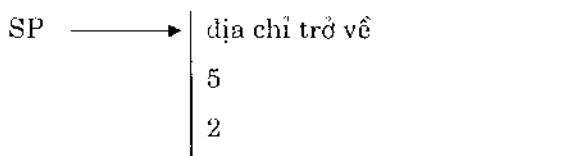
MOV AX,[BP+6]

bằng offset định ngăn xếp cộng với 6. Đây là vị trí của giá trị WORD1 (2). Tương tự ở dòng 23, toán hạng nguồn trong:

ADD AX,[BP+4]

là giá trị của WORD2 (5).

Sau khi phục hồi giá trị ban đầu cho BP ở dòng 24, ngăn xếp trở thành:



Để thoát khỏi thủ tục và phục hồi ngăn xếp với trạng thái ban đầu của nó ta sử dụng:

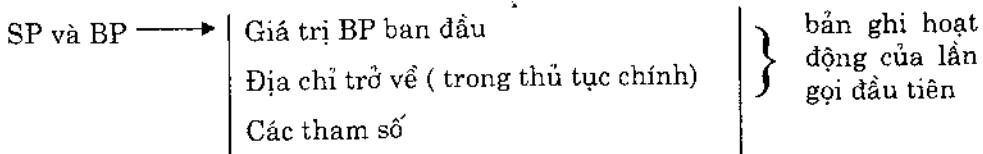
RET 4

Lệnh này nạp địa chỉ trả về vào IP đồng thời loại bỏ khỏi ngăn xếp 4 byte.

17.4. Bản ghi hoạt động

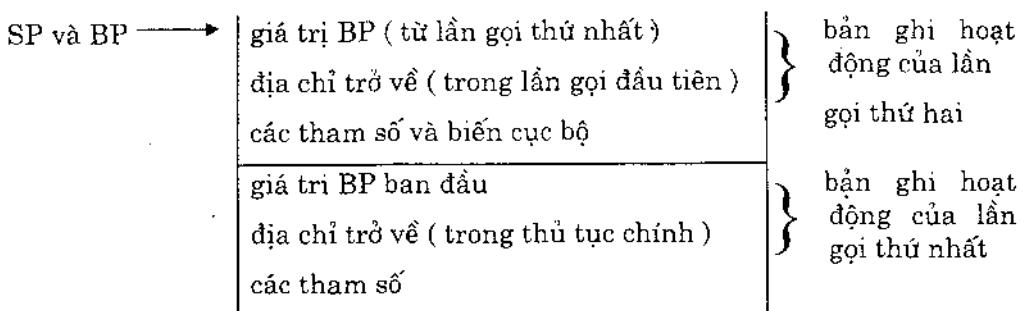
Trước khi bắt tay vào viết một thủ tục đệ quy, có một vấn đề cần được giải quyết. Các tham số (và các biến cục bộ nếu có) của thủ tục được khởi tạo lại mỗi lần thủ tục được gọi. Trong cả hai ví dụ phần 17.2, thủ tục đầu tiên được gọi với $N = 4$, sau đó là $N = 3$ rồi $N = 2$ và $N = 1$. Khi một lần gọi được hoàn thành, thủ tục tiếp tục lần gọi trước đó tại điểm mà nó bị ngắt. Để thực hiện phải có một cách nào đó ghi nhớ điểm ngắt cũng như giá trị của các tham số và biến cục bộ của lần gọi đó. Các giá trị này được gọi là **bản ghi hoạt động** của lần gọi.

Để minh họa, giả thiết rằng chúng ta có một thủ tục được gọi một lần từ thủ tục chính và sau đó gọi chính nó hai lần nữa. Trước khi khởi tạo lần gọi đầu tiên, thủ tục chính đưa bản ghi hoạt động ban đầu vào ngăn xếp và gọi thủ tục. Thủ tục cất BP và thiết lập nó trả đến định ngăn xếp giống như đã làm trong ví dụ ở phần 17.3. Ngăn xếp như sau:

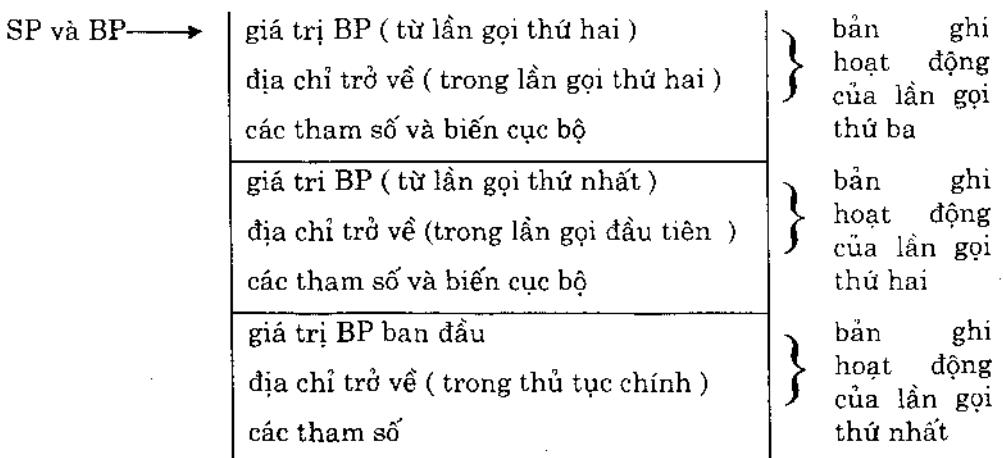


Thủ tục thi hành các lệnh của nó bằng cách sử dụng BP truy nhập các tham số và các biến cục bộ. Trước khi gọi đệ quy, nó đưa bản ghi hoạt động cho lần gọi tiếp theo vào ngăn xếp. Địa chỉ trả về mà lần gọi đệ quy cất vào ngăn xếp là địa

chỉ của lệnh tiếp theo được thực hiện trong thủ tục. Khi lần gọi thứ hai bắt đầu, thủ tục một lần nữa cất BP và thiết lập nó trở đến đỉnh ngăn xếp. Kết quả là:



Bây giờ trong lần gọi đầu tiên, thủ tục sử dụng BP để truy nhập dữ liệu cho lần gọi thứ hai. Trước khi khởi tạo lần gọi thứ ba, bản ghi hoạt động của nó được đưa vào trong ngăn xếp. Lần gọi thứ ba cất BP và thiết lập nó trở đến đỉnh ngăn xếp. Ngăn xếp trở thành:



Chúng ta hãy giả thiết rằng lần gọi thứ ba là điểm thoát. Kết quả tính toán của nó có thể đưa vào một thanh ghi hay một ô nhớ để lần gọi thứ hai có thể sử dụng khi được tiếp tục. Sau khi lần gọi thứ ba hoàn thành, lần gọi thứ hai có thể được tiếp tục bằng cách lấy BP từ ngăn xếp để phục hồi giá trị trước đó của nó và thi hành các câu lệnh trả về. Giá trị trả về trong IP là địa chỉ của lệnh được thi hành tiếp theo trong lần gọi thứ hai. Khi trả về, các tham số và biến cục bộ của lần gọi thứ ba được loại bỏ khỏi ngăn xếp giống như đã thực hiện trong ví dụ phần trước đây. Ngăn xếp trở thành:

| | | |
|------------|---|---|
| SP và BP → | giá trị BP (từ lần gọi thứ nhất) địa chỉ trả về (trong lần gọi đầu tiên) các tham số | } bản ghi hoạt động của lần gọi thứ hai |
| | giá trị BP ban đầu địa chỉ trả về (trong thủ tục chính) các tham số | |

Bây giờ lần gọi thứ hai tiếp tục. Nó lấy kết quả của lần gọi thứ ba và hoàn thành công việc. Khi nó kết thúc và chưa lại kết quả, một lần nữa BP được lấy khỏi ngăn xếp và điều khiển được trả về cho lần gọi thứ nhất. Giống như trước, các số liệu của lần gọi thứ hai bị loại bỏ. Bây giờ ngăn xếp như sau:

| | | |
|------------|---|--|
| SP và BP → | giá trị BP ban đầu địa chỉ trả về (trong thủ tục chính) các tham số | } bản ghi hoạt động của lần gọi thứ nhất |
| | | |

Khi lần gọi đầu tiên được hoàn thành, thủ tục phục hồi BP với giá trị ban đầu của nó và điều khiển được truyền cho thủ tục chính. Giống như trước, các tham số đều bị loại bỏ. Thủ tục lưu giữ kết quả cuối cùng vào nơi mà thủ tục chính có thể truy nhập được.

17.5. Thực hiện các thủ tục đệ quy

Trong phần này chúng tôi sẽ chỉ ra các thủ tục đệ quy được thực hiện như thế nào trong Hợp ngữ.

Ví dụ 17.1.: Viết thủ tục FACTORIAL trong phần 17.2. Gọi nó trong một chương trình để tính 3 giai thừa.

Lời giải: Để tiện theo dõi, ta viết lại thuật toán ở đây:

```

1: PROCEDURE FACTORIAL ( vào: N, ra: RESULT )
2: IF N = 1
3:   THEN
4:     RESULT = 1

```

```

5:      ELSE
6:          call FACTORIAL ( vào:N-1,ra:RESULT )
7:          RESULT = N * RESULT
8: END_IF
9: RETURN

```

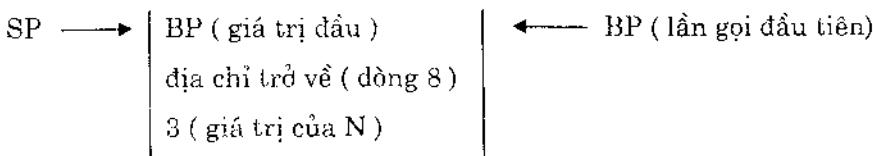
Chương trình nguồn PGM17_2.ASM

```

0   TITLE PGM17_2: FACTORIAL PROGRAM
1   .MODEL      SMALL
2   .STACK      100h
3   .CODE
4   MAIN PROC
5       MOV AX, 3           ;N = 3
6       PUSH AX            ;cất N vào ngăn xếp
7       CALL FACTORIAL    ;AX bằng 3 giai thừa
8       MOV AH, 4CH          ;trở về DOS
9       INT 21H
10  MAIN ENDP
11  FACTORIAL PROC NEAR
12 ;tính N giai thừa
13 ;vào:ngăn xếp-dịa chỉ trả về (đỉnh), N
14 ;ra :AX
15       PUSH BP            ;cất BP
16       MOV BP,SP          ;BP trả đến đỉnh ngăn xếp
17 ;if
18       CMP WORD PTR[BP+4], 1 ;N=1?
19       JG END_IF           ;không, gọi đệ quy
20 ;then
21       MOV AX, 1            ;đúng, kết quả = 1
22       JMP RETURN
23 END_IF:
24       MOV CX,[BP+4]         ;lấy N
25       DEC CX              ;N-1
26       PUSH CX              ;cất vào ngăn xếp
27       CALL FACTORIAL      ;gọi đệ quy
28       MUL WORD PTR[BP+4]    ;RESULT=N x RESULT
29 RETURN:
30       POP BP               ;phục hồi BP
31       RET 2                ;trở về và loại bỏ N
32 FACTORIAL ENDP
33 END MAIN

```

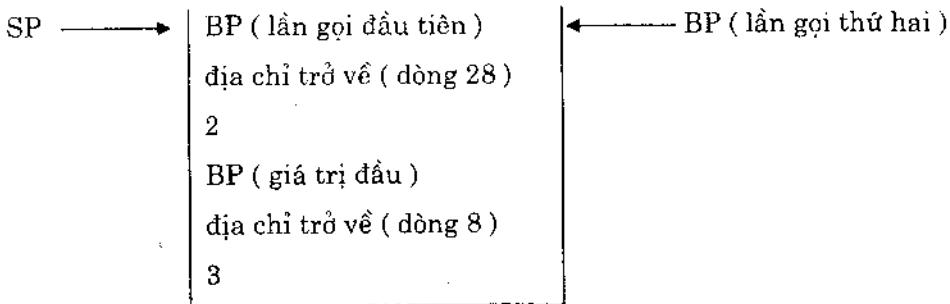
Chương trình kiểm tra đưa 3 vào ngăn xếp và gọi FACTORIAL. Tại các dòng 15 và 16, thủ tục cắt BP và thiết lập trở đến đỉnh ngăn xếp. Ngăn xếp lúc này như sau:



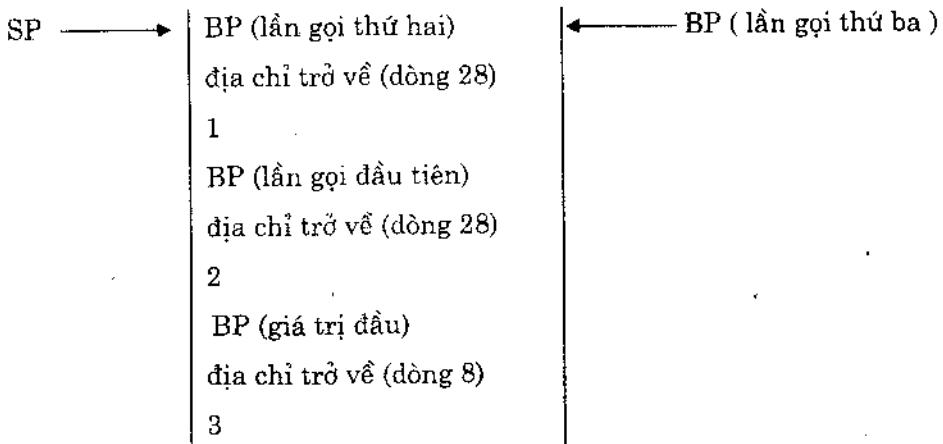
Bây giờ đến dòng 18, kiểm tra giá trị hiện tại của N. Chúng ta phải dùng lệnh CMP WORD PTR [BP+4],1 thay vì CMP [BP+4],1 bởi lẽ chương trình biên dịch không biết được từ toán hạng nguồn 1 đây là lệnh byte hay word.

Do N <> 1, dữ liệu cho lần gọi tiếp theo được chuẩn bị bằng cách phục hồi giá trị hiện tại của N, giảm đi 1 và cất vào ngăn xếp.

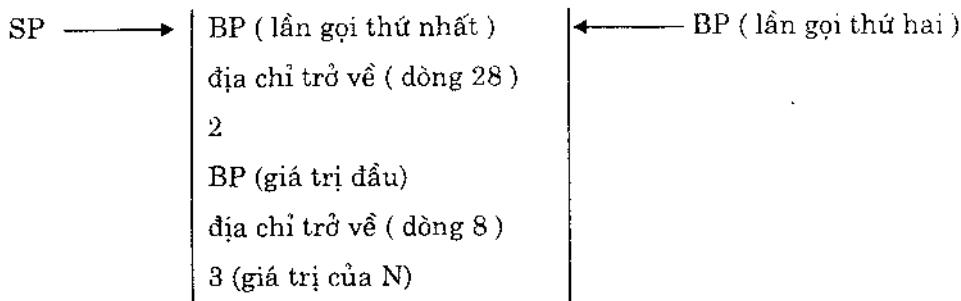
Tại dòng 27 phát sinh lần gọi thứ hai ($N = 2$). Một lần nữa tại các dòng 15 và 16, BP được cắt và thiết lập trở đến đỉnh ngăn xếp. Ngăn xếp trở thành:



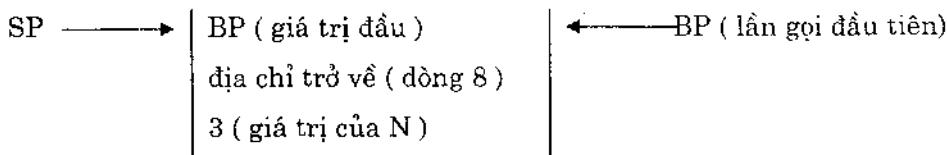
Vì N vẫn khác 1, thủ tục gọi chính nó một lần nữa và ngăn xếp như sau:



Do lúc này N là 1, sự đệ quy kết thúc. Tại dòng 21, chương trình đưa kết quả bằng 1 vào AX, phục hồi lại BP giá trị của nó trong lần gọi thứ hai và trở về. Lệnh RET 2 ở dòng 31 đưa địa chỉ trả về trong lần gọi thứ hai vào IP, đồng thời loại bỏ tham số N khỏi ngăn xếp. Ngăn xếp trở thành:



Bây giờ tiếp tục thi hành lần gọi thứ hai ở dòng 28. Vì kết quả của lần gọi thứ ba trong AX, thủ tục có thể nhân nó với giá trị hiện tại của N, thu được $RESULT = 2 \times 1 = 2$. Kết quả mới này được lưu lại trong AX. Khi lần gọi này hoàn thành, BP được phục hồi và lần gọi đầu tiên được tiếp tục ở dòng 28. Ngăn xếp bây giờ như sau:



Giống như trước, kết quả mới nhất được nhân với N, ta thu được $RESULT = 3 \times 2 = 6$. Điều khiển được truyền cho thủ tục chính ở dòng 8 với kết quả cuối cùng trong AX.

Ví dụ 17.2: Viết thủ tục FIND_MAX trong phần 17.2 và kiểm tra nó trong một chương trình.

Lời giải:

Ta viết lại thuật toán của thủ tục:

```

1: PROCEDURE FIND_MAX (vào:N, ra: MAX)
2: IF N = 1
3:   THEN
4:     MAX = A[1]
5:   ELSE
  
```

```

6:          call FIND_MAX ( N-1, MAX )
7:          IF A[ N] > MAX
8:          THEN
9:              MAX = A[ N]
10:         ELSE
11:             MAX = MAX
12:         END_IF
13: RETURN

```

Chương trình nguồn PGM17_3.ASM

```

0   TITLE PGM17_2: FIND_MAX
1   .MODEL      SMALL
2   .STACK      100h
3   .DATA
4       A      DW      10,50,20,4
5   .CODE
6   MAIN PROC
7       MOV AX,@DATA
8       MOV DS,AX      ;khởi tạo DS
9       MOV AX,4      ;số phần tử trong mảng
10      PUSH AX      ;tham số trong ngăn xếp
11      CALL FIND_MAX ;trả giá trị max trong AX
12      MOV AH,4CH
13      INT 21H      ;trở về DOS
14 MAIN ENDP
15 FIND_MAX PROC NEAR
16 ;tìm phần tử lớn nhất trong mảng A gồm N phần tử
17 ;vào:ngăn xếp-địa chỉ trả về(đỉnh),N
18 ;ra :AX chứa phần tử lớn nhất
19     PUSH BP      ;cất BP
20     MOV BP,SP      ;BP trở đến đỉnh ngăn xếp
21 ;if
22     CMP WORD PTR[BP+4],1      ;N=1?
23     JG ELSE_        ;không, thiết lập lần gọi
                           ;tiếp theo
24 ;then
25     MOV AX,A      ;MAX=A[ 1]
26     JMP END_IF
27 ELSE_:
28     MOV CX,[BP+4]    ;lấy N
29     DEC CX      ;N-1
30     PUSH CX      ;cất vào ngăn xếp
31     CALL FIND_MAX ;trả về MA X trong AX

```

```

32 ;if
33     MOV    BX,[ BP+4]           ;lấy N
34     SHL    BX,1                ;2N
35     SUB    BX,2                ;2(N-1)
36     CMP    A[ BX] ,AX          ;A(N)>MAX ?
37     JLE    END_IF             ;không, trở về
38 ;then
39     MOV    AX,A[ BX]           ;đúng, lập MAX=A[ N]
40 END_IF:
41     POP    BP                 ;phục hồi BP
42     RET    2                  ;trở về và loại bỏ N
43 FIND_MAX ENDP
44     END   MAIN

```

Sự sắp xếp các bản ghi hoạt động trong khi gọi đệ quy của ví dụ này tương tự như ví dụ 17.1 và nó không được nêu ra ở đây (xem như bài tập).

Đến dòng 32, thủ tục bắt đầu chuẩn bị so sánh A[N] với giá trị lớn nhất hiện tại trong AX. Trong chương 10 ta đã biết offset của phần tử thứ N trong một mảng word A có giá trị $A + 2x(N - 1)$. Các dòng 33-35 đưa $2x(N - 1)$ vào BX và dòng 36 sử dụng chế độ địa chỉ cơ sở để so sánh. Nếu như MAX > A[N] chúng ta có thể giữ nguyên nó trong AX, có nghĩa là câu lệnh ELSE tại dòng 11 của thuật toán không cần mã hoá.

17.6. Một chương trình đệ quy phức tạp hơn

Trong các ví dụ trước đây, các thủ tục đệ quy chỉ có chứa một lời gọi đệ quy. Ví dụ như lời gọi duy nhất trong thủ tục FACTORIAL (N) biến nó thành FACTORIAL (N-1). Tuy nhiên mã lệnh của một thủ tục đệ quy có thể bao gồm các lời gọi đệ quy phức tạp.

Ví dụ chúng muốn viết một thủ tục tính các tổ hợp chập k của n: C(n,k). Chúng là các hệ số xuất hiện trong biểu thức $(x+y)^n$. Biểu thức có dạng sau:

$$(x+y)^n = C(n,0)x^n y^0 + C(n,1)x^{n-1}y^1 + C(n,2)x^{n-2}y^2 + \dots + C(n,n-1)x^1y^{n-1} + C(n,n)x^0y^n$$

Các hệ số này còn được dùng để xây dựng tam giác Pascal. Với n = 4, tam giác Pascal của chúng ta như sau:

| | | | | |
|--------|--------|--------|--------|--------|
| | C(0,0) | | | |
| C(1,0) | C(1,1) | | | |
| C(2,0) | C(2,1) | C(2,2) | | |
| C(3,0) | C(3,1) | C(3,2) | C(3,3) | |
| C(4,0) | C(4,1) | C(4,2) | C(4,3) | C(4,4) |

Các hệ số thoả mãn quan hệ sau đây:

$$C(n,n) = C(n,0) = 1$$

$$C(n,k) = C(n-1,k) + C(n-1,k-1) \quad \text{nếu } n > k > 0$$

Điều này có nghĩa là các phần tử nằm trên các cạnh đều là 1 và các phần tử bên trong bằng tổng các phần tử bên trái và phải ở hàng sát trên. Như vậy tam giác sau khi tính toán trở thành:

| | | | | |
|---|---|---|---|---|
| | 1 | | | |
| 1 | 1 | | | |
| 1 | 2 | 1 | | |
| 1 | 3 | 3 | 1 | |
| 1 | 4 | 6 | 4 | 1 |

Chúng ta hãy sử dụng định nghĩa trên để tính $C(3,2)$.

$$C(3,2) = C(2,2) + C(2,1)$$

$$C(2,2) = 1$$

$$C(2,1) = C(1,1) + C(1,0)$$

$$C(1,1) = 1$$

$$C(1,0) = 1$$

$$\text{Như vậy } C(2,1) = 1 + 1 = 2$$

$$\text{và } C(3,2) = 1 + 2 = 3$$

Sau đây là một thuật toán của thủ tục tính $C(n,k)$:

```

PROCEDURE    BINOMIAL (vào: N, K; ra: RESULT)
IF (K = N) OR (K = 0)
THEN
    RESULT = 1
ELSE
    CALL BINOMIAL (N-1, K, RESULT1)
    CALL BINOMIAL (N-1, K-1, RESULT2)
    RESULT = RESULT1 + RESULT2
RETURN

```

Ví dụ 17.3: Viết các mã lệnh cho thủ tục BINOMIAL và gọi nó trong một chương trình để tính C(3, 2).

Lời giải:

Chương trình nguồn PGM17_4.ASM

```
0: TITLE PGM17_4: BINOMIAL COEFICIENTS
1: .MODEL      SMALL
2: .STACK      100h
3: .CODE
4: MAIN PROC
5:     MOV AX, 2           ; k=2
6:     PUSH AX
7:     MOV AX, 3           ; n=3
8:     PUSH AX
9:     CALL BINOMIAL       ; AX=RESULT
10:    MOV AH, 4CH
11:    INT 21H             ; trả về DOS
12: MAIN ENDP
13: BINOMIAL PROC NEAR
14:     PUSH BP
15:     MOV BP, SP
16:     MOV AX, [BP+6]       ; lấy K
17: ;if
18:     CMP AX, [BP+4]       ; K=N?
19:     JE THEN             ; đúng, không đệ quy
20:     CMP AX, 0            ; K=0?
21:     JNE ELSE_             ; không, đệ quy
22: THEN:
23:     MOV AX, 1             ; RESULT=1
24:     JMP RETURN
25: ELSE_:
26: ;tính C(N-1, K)
27:     PUSH [BP+6]           ; cất K
28:     MOV CX, [BP+4]         ; lấy N
29:     DEC CX                ; N-1
30:     PUSH CX                ; cất N-1
31:     CALL BINOMIAL         ; RESULT1 trong AX
32:     PUSH AX                ; cất RESULT1
33: ;tính C(N-1, K-1)
34:     MOV CX, [BP+6]           ; lấy K
35:     DEC CX                ; K-1
36:     PUSH CX                ; cất K-1
```

```

37:     MOV    CX,[ BP+4]      ;lấy N
38:     DEC    CX              ;N-1
39:     PUSH   CX              ;cất N-1
40:     CALL   BINOMIAL       ;RESULT2 trong AX
41: ;tính C(N,K)
42:     POP    BX              ;lấy RESULT1
43:     ADD    AX,BX           ;RESULT=RESULT1+RESULT2

44: RETURN:
45:     POP    BP              ;phục hồi BP
46:     RET    4               ;trở về và loại bỏ N,K
47: BINOMIAL    ENDP
48: END MAIN

```

Thủ tục BINOMIAL khác với các thủ tục trong các ví dụ 17.1 và 17.2 ở các điểm sau:

1. Có hai điểm thoát $k = n$ và $k = 0$; trong cả hai trường hợp, lời gọi trả về 1 trong AX.
2. Trong trường hợp tổng quát, việc tính toán $C(n,k)$ bao gồm hai lời gọi đệ quy để tính $C(n-1,k)$ và $C(n-1,k-1)$.

Tất cả các lần gọi thủ tục BINOMIAL đều trả về kết quả trong AX. Sau khi tính $C(n-1,k)$ (dòng 31), kết quả (RESULT1 trong thuật toán) được đưa vào ngăn xếp (dòng 32). Dòng 40 tính $C(n-1,k-1)$ và kết quả (RESULT2 trong thuật toán) được chứa trong AX. Các dòng 42-43 lấy RESULT1 từ ngăn xếp đưa vào BX và cộng với RESULT2, như vậy AX sẽ chứa $C(n,k) = C(n-1,k) + C(n-1,k-1)$.

Để hiểu một cách cặn kẽ thủ tục BINOMIAL làm việc như thế nào, bạn hãy theo dõi kết quả chạy từng bước của thủ tục trong ngăn xếp như đã làm trong ví dụ 17.1.

TỔNG KẾT.

- Việc giải quyết vấn đề đệ quy có các tính chất sau đây: (1) Vấn đề chính được chia thành các vấn đề đơn giản hơn, mỗi trong chúng có cách giải quyết tương tự như vấn đề chính; (2)có một trường hợp không đệ quy và (3) khi một vấn đề con được giải quyết, công việc tiếp tục với bước tiếp theo của vấn đề ban đầu.
- Trong Hợp ngữ, các thủ tục đệ quy được thực hiện như sau: Chương trình gọi đưa bản ghi hoạt động cho lần gọi đầu tiên vào ngăn xếp và gọi thủ tục. Thủ tục dùng BP để truy nhập dữ liệu mà nó cần trong ngăn xếp. Trước khi khởi tạo một lời gọi đệ quy, thủ tục chứa bản ghi hoạt động cho lời gọi vào ngăn xếp và gọi chính nó. Khi một lời gọi hoàn thành, BP được phục hồi, địa chỉ trả về được nạp vào IP và số dữ liệu cho lời gọi này được lấy ra khỏi ngăn xếp.
- Các mã lệnh của thủ tục có thể có nhiều hơn một lần gọi đệ quy. Kết quả trung gian có thể được cất vào ngăn xếp và lấy lại khi lần gọi ban đầu tiếp tục.

Các thuật ngữ tiếng anh.

Activition record

Giá trị của các tham số, các biến cục bộ và địa chỉ trả về của một lần gọi thủ tục.

recursive process

Một quá trình được định nghĩa trong khi thực hiện nó.

Bài tập.

- Viết một định nghĩa đệ quy cho an, trong đó n là số nguyên không âm.
- Một hàm được định nghĩa như sau với m, n là các số nguyên không âm:
$$\begin{aligned}A(0,n) &= n+1 \\ A(m,0) &= A(m-1,1) \\ A(m,n) &= A(m-1,A(m,n-1))\end{aligned}$$
 với $m, n > 0$

Dùng định nghĩa chứng minh rằng $A(2,2) = 7$.

- Chạy từng bước ví dụ 17.2 (PGM17_3.ASM) và cho biết ngăn xếp:

- a) Tại dòng 20 khi khởi tạo lần gọi (đầu tiên) thủ tục FIND_MAX.
- b) Tại dòng 20 trong lần gọi thủ tục FIND_MAX lần thứ hai.
- c) Tại dòng 20 trong lần gọi thủ tục FIND_MAX lần thứ ba.
- d) Tại dòng 20 trong lần gọi thủ tục FIND_MAX lần thứ tư. Đây là điểm thoát.
- e) Tại dòng 42 khi hoàn thành thủ tục FIND_MAX lần thứ ba (sau khi thi hành RET 2). Bạn hãy cho biết giá trị trong AX
- f) Tại dòng 42 khi hoàn thành thủ tục FIND_MAX lần thứ hai (sau khi thi hành RET 2). Bạn hãy cho biết giá trị trong AX
- g) Tại dòng 42 khi hoàn thành thủ tục FIND_MAX lần thứ tư (sau khi thi hành RET 2). Bạn hãy cho biết giá trị trong AX. Đây là giá trị thủ tục trả về.

Các bài tập lập trình.

4. Viết một thủ tục bằng Hợp ngữ để tính tổng các phần tử của một mảng word. Viết một chương trình để kiểm tra thủ tục của bạn với một mảng 4 phần tử.
5. Chuỗi Finonacci 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... có thể được định nghĩa một cách đệ quy như sau:

$$\begin{aligned} F(0) &= F(1) = 1 \\ F(n) &= F(n-1) + F(n-2) \quad \text{với } n > 1. \end{aligned}$$

Viết một thủ tục đệ quy bằng Hợp ngữ để tính $F(n)$, và gọi nó trong một chương trình kiểm tra để tính $F(7)$.

Chương 18

CÁC PHÉP TOÁN SỐ HỌC NÂNG CAO

Tổng quan

Các chương trình thường phải làm việc với dữ liệu lớn hơn 16 bit hay có chứa phần thập phân hoặc có những cách mã hoá đặc biệt. Trong 3 phần đầu của chương này chúng ta hãy xem xét các thao tác số học với các số có độ chính xác kép, số BCD (số thập phân được mã hoá nhị phân - Binary-Coded Decimal) và các số dấu phẩy động. Trong phần 18.4 chúng ta sẽ bàn đến các thao tác của bộ đồng xử lý số 8087.

18.1 Các số có độ chính xác kép

Chúng ta đã biết các số lưu trữ trong máy tính dựa trên cơ sở bộ vi xử lý 8086 có thể có 8 hay 16 bit. Nhưng ngay cả đối với các số 16 bit thì giới hạn của chúng vẫn chỉ là từ 0 đến 65535 cho các số không dấu và từ -32768 đến 32767 cho các số có dấu. Để mở rộng giới hạn này thì một kỹ thuật khá phổ biến là sử dụng 2 word cho mỗi số. Các số như vậy được gọi là các số có độ chính xác kép (double_precision numbers). Và phạm vi biểu diễn của chúng là từ 0 đến $2^{32} - 1$ hay 4.294.967.295 cho các số không dấu và từ -2.147.483.648 đến +2.147.483.647 cho các số có dấu.

Các số độ chính xác kép có thể chiếm 2 thanh ghi hay từ nhớ. Ví dụ một số 32 bit được chứa trong 2 từ nhớ là A và A+2 (sẽ được viết A+2:A) thì 16 bit cao của nó ở trong A+2 và 16 bit thấp ở trong A. Nếu số là số có dấu thì bit msb của A+2 sẽ là bit dấu của số. Các số âm được biểu diễn dưới dạng số bù 2.

Do 8086/8088 chỉ có thể tính toán với các số 8 hay 16 bit, các thao tác với số có độ chính xác kép phải được giả lập bằng phần mềm. Trong phần 18.4 chúng ta sẽ thấy bộ đồng xử lý 8087 làm việc như thế nào với các số có độ chính xác kép.

18.1.1 Phép cộng, trừ, phủ định với các số có độ chính xác kép.

Để cộng hoặc trừ các số 32 bit, trước tiên chúng ta phải tiến hành cộng hay trừ 16 bit thấp rồi sau đó tiến hành cộng hay trừ 16 bit cao của chúng. Tuy nhiên đáp số sẽ không còn đúng nữa nếu phép cộng hay trừ đầu tiên có nhớ hay mượn.

Một phương pháp để giải quyết vấn đề này là dùng các lệnh để kiểm tra cờ và điều chỉnh kết quả. Một phương pháp tốt hơn là dùng các lệnh mới của 8086. Lệnh ADC (Add with carry) cộng toán hạng nguồn và CF với toán hạng đích còn lệnh SBB (Subtract with borrow) trừ toán hạng nguồn và CF từ toán hạng đích. Cú pháp của các lệnh này như sau:

```
ADC destination, source  
SBB destination, source
```

Ví dụ 18.1: Viết các lệnh thực hiện việc cộng 2 số 32 bit A+2:A và B+2:B.

Trả lời:

Chúng ta phải chuyển số thứ nhất vào thanh ghi trước khi có thể tiến hành cộng chúng.

```
MOV AX,A ;AX chứa 16 bit thấp trong A  
MOV DX,A+2 ;DX chứa 16 bit cao trong A+2  
ADD B,AX ;cộng 16 bit thấp vào B  
ADC B+2,DX ;cộng 16 bit cao và CF vào B+2
```

Khi 32 bit của tổng được chứa trong B+2:B, các cờ có thể bị thiết lập không đúng. Đặc biệt là các cờ mà đáng ra nó được thiết lập tùy theo giá trị của B+2 và B như ZF và PF thì nay chúng chỉ được thiết lập theo giá trị của B+2. Trong trường hợp cần phải thiết lập đúng các cờ chúng ta phải sử dụng thêm các lệnh khác.

Thủ tục DADD trong chương trình nguồn PGM18_1.ASM thực hiện phép cộng 2 số có độ chính xác kép và đặt lại các cờ với các giá trị đúng (dường như là bộ vi xử lý có lệnh cộng 32 bit). Chúng ta giả sử 2 số được chứa trong DX:AX và CX:BX và tổng sẽ được chứa trong DX:AX.

Chương trình nguồn PGM18_1.ASM

```
; thủ tục cộng 2 số có độ chính xác kép
; Có điều chỉnh PF và ZF
DADD PROC
;           Vào: CX:BX = toán hạng nguồn
;                   DX:AX = toán hạng đích
;           Ra: DX:AX = tổng
;cắt thanh ghi SI
    PUSH SI          ;thủ tục sử dụng SI để
                      ;chứa các thanh ghi cờ
    ADD  AX,BX      ;cộng 16 bit thấp
    ADC  DX,CX      ;cộng 16 bit cao có nhớ
    PUSHF            ;cắt thanh ghi cờ vào ngăn xếp
    POP   SI          ;và chuyển nội dung ngăn
                      ;xếp vào SI
;kiểm tra ZF
    JNE   CHEK_PF   ;nếu DX khác 0 thi ZF
                      ;đúng, kiểm tra PF
    TEST AX,0FFFFH  ;DX=0, kiểm tra xem AX=0?
    JE    CHEK_PF   ;đúng, vậy ZF đúng
    AND   SI,0FFBFH  ;AX khác 0, xoá bit ZF
                      ;trong SI
;kiểm tra PF
CHEK_PF:
    OR    SI,100B    ;lắp parity chẵn cho SI
    TEST AL,0FFH    ;kiểm tra parity cho AL
    JP    RESTORE   ;AL có parity chẵn, SI đúng
    XOR   SI,100B    ;AL có parity lẻ, đảo bit PF
                      ;trong SI
RESTORE:
    PUSH SI          ;đặt các cờ vào ngăn xếp
    POPF             ;cập nhật thanh ghi cờ
;phục hồi SI
    POP   SI
    RET
DADD ENDP
```

Thanh ghi SI được dùng để thao tác các bit của thanh ghi cờ. Chúng ta chuyển thanh ghi cờ vào SI bằng cách đưa nó vào ngăn xếp rồi đưa nội dung của ngăn xếp vào SI vì nội dung của thanh ghi cờ không thể chuyển trực tiếp vào SI. Để điều chỉnh ZF chúng ta kiểm tra cả DX và AX, còn để kiểm tra PF chúng ta kiểm tra AL. Sau khi đã kiểm tra và hiệu chỉnh xong chúng ta chép lại nội dung của SI vào thanh ghi cờ vẫn bằng cách sử dụng ngăn xếp. Nếu bạn muốn sử dụng DADD trong chương trình của mình, các bạn có thể dùng dẫn hướng biên dịch INCLUDE để ghép PGM18_1.ASM vào.

Để nhận được đảo của một số có độ chính xác kép chúng ta cần nhớ lại rằng số bù 2 của một số bất kỳ là số bù một của nó cộng với 1.

Ví dụ 18.2: Viết các lệnh tạo ra số bù 2 của số A+2:A.

Trả lời:

Trước hết chúng ta đi tìm số bù 1 bằng cách sử dụng lệnh NOT rồi sau đó cộng số tìm được với 1.

| | | |
|-----|--------|---------------------------|
| NOT | A+2 | ;lấy bù 1 |
| NOT | A | ;lấy bù 1 |
| INC | A | ;cộng thêm 1 |
| ADC | A+2, 0 | ;để phòng khả năng có nhớ |

Đối với phép trừ chúng ta cũng tiến hành trừ 16 bit thấp trước, rồi sau đó trừ tiếp 16 bit cao với nhau cùng với nhớ có thể tạo ra từ phép trừ đầu.

Ví dụ 18.3 Viết các lệnh trừ số 32 bit trong B+2:B cho số 32 bit trong A+2:A.

Trả lời:

| | | |
|-----|---------|-------------------------------|
| MOV | AX, A | ;AX chứa 16 bit thấp trong A |
| MOV | DX, A+2 | ;DX chứa 16 bit cao trong A+2 |
| SUB | B, AX | ;trừ 16 bit thấp |
| SBB | B+2, DX | ;trừ B+2 cho DX và CF |

Để thiết lập đúng các cờ chúng ta có thể sử dụng các kỹ thuật giống như đối với phép cộng, chúng tôi giành lại cho các bạn như một bài tập.

18.1.2 Phép nhân và phép chia các số có độ chính xác kép.

Phép nhân và phép chia các số có độ chính xác kép cho luỹ thừa của 2 có thể thực hiện bằng các thao tác dịch như đã làm trong chương 7. Để nhân chúng ta tiến hành phép dịch trái và để chia chúng ta tiến hành phép dịch phải.

Ví dụ 18.4: Viết các lệnh thực hiện thao tác dịch trái số trong A+2:A.

Trả lời:

Chúng ta bắt đầu với lệnh dịch trái từ thấp, msb của từ thấp sẽ được chuyển vào CF. Tiếp theo bằng lệnh RCL chúng ta sẽ dịch trái từ cao với CF được dịch vào nó. Các lệnh như sau:

| | | |
|-----|--------|---------------------|
| SHL | A, 1 | ;dịch từ thấp |
| RCL | A+2, 1 | ;dịch CF vào từ cao |

Một lần nữa các cờ OF, ZF và PF có thể bị đặt không đúng. Ví dụ sau sẽ biểu diễn phép nhân với 2^{10} số trong A+2:A.

Ví dụ 18.5: Viết các lệnh để dịch trái 10 lần số trong A+2:A.

Trả lời:

Có một cách tưởng như khá hay là đặt 10 vào CL và sử dụng nó như một bộ đếm trong lệnh dịch. Nhưng cách làm này sẽ khiến cho 9 bit của số bị mất. Trong phép dịch các số có độ chính xác kép chúng ta chỉ được dịch một bit một lần. CX có thể sử dụng làm bộ đếm vòng lặp.

```
MOV CX, 10      ; khởi tạo bộ đếm
L1:
    SAL A, 1      ; dịch từ thấp
    RCL A+2, 1    ; dịch từ cao
    LOOP L1       ; lặp lại nếu bộ đếm chưa bằng 0
```

Các phép dịch và quay khác chúng tôi để lại cho các bạn như là các bài tập.

Khi số nhân không phải là luỹ thừa của 2, chúng ta có thể thực hiện phép nhân và bằng một loạt các phép cộng. Chẳng hạn để nhân 2 số có độ chính xác kép M và N chúng ta có thể tiến hành phép cộng số M N lần. Một phương pháp có hiệu quả hơn để nhân và chia các số có độ chính xác kép là sử dụng các chỉ thị của bộ đồng xử lý số sẽ được trình bày trong phần 18.4.

18.2 Số thập phân mã hoá nhị phân

Hệ thống số BCD (số thập phân được mã hoá nhị phân) sử dụng 4 bit để mã hoá một chữ số thập phân, từ 0000 đến 1001. Các tổ hợp còn lại từ 1010 đến 1111 là không hợp lệ đối với các số BCD. Ví dụ dạng biểu diễn BCD của số thập phân 957 là 1001 0101 0111. Sở dĩ người ta sử dụng các số BCD là vì việc chuyển đổi giữa các số thập phân và các số BCD tương đối dễ dàng. Trong phần 18.4 chúng tôi sẽ trình bày một thủ tục để đổi giữa số thập phân và số BCD.

Như chúng ta đã thấy ở chương 9, để thực hiện các thao tác vào ra với số thập phân cần phải thực hiện các phép nhân và chia. Đây là các thao tác khá chậm chạp. Đối với một số chương trình dùng trong công việc kinh doanh thực hiện nhiều thao tác vào ra nhưng chỉ làm các công việc tính toán đơn thuần thì các số sẽ được lưu trữ dưới dạng các số BCD nhằm giảm tối thiểu thời gian cần thiết. Có lẽ không cần thiết phải nói thêm rằng các bộ vi xử lý phải làm cho các thao tác số học với số BCD trở lên dễ dàng hơn đối với chương trình nếu muốn tiết kiệm được lượng thời gian đáng kể.

Trước hết chúng ta hãy xem xét 2 phương pháp lưu trữ các số BCD trong bộ nhớ.

18.2.1 Các số BCD dạng nén và không nén

Do chỉ cần 4 bit để biểu diễn một chữ số BCD nên 2 chữ số có thể đặt trong một byte. Các số này được gọi là số BCD dạng nén(packed BCD form). Trong dạng BCD không nén(unpacked BCD form), chỉ có một chữ số chứa trong một byte. 8086 có các lệnh cộng và trừ đối với cả 2 loại nhưng các phép nhân và chia thì chỉ có BCD dạng không nén được chấp nhận.

Ví dụ 18.6: Cho biết dạng nhị phân, BCD dạng nén và không nén biểu diễn số thập phân 59.

Trả lời:

$59 = 3Bh = 00111011$ đó chính là dạng biểu diễn nhị phân. Ta có $5 = 0101$ và $9 = 1001$ nên dạng biểu diễn BCD nén là 01011001 , dạng biểu diễn BCD không nén là

00000101 00001001.

Trong các phần dưới đây chúng ta sẽ nghiên cứu các lệnh thao tác số học với các số BCD không nén.

18.2.2 Phép cộng các số BCD và lệnh AAA

Khi tính toán với số BCD chúng ta làm việc với từng chữ số một. Rất có thể tổng của 2 số BCD lại là một số không phải dạng BCD. Chẳng hạn chúng ta cộng BL có chứa 7 với AL có chứa 6 thì tổng của chúng là 13 chứa trong AL bây giờ lại không còn là số BCD nữa. Để điều chỉnh chúng ta trừ AL cho 10 và đặt 1 vào AH khi đó AX sẽ chứa kết quả đúng

| | | | |
|----|-----------------|------------|---|
| AH | 00000000 | AL | 00000110 |
| | | BL | + 00000111 |
| | | AL | <u>00001101</u> ; không phải là số BCD |
| | + 1 | - 00001010 | ; điều chỉnh bằng cách trừ AL cho ; 10d và cộng 1 vào AH |
| AH | <u>00000001</u> | AL | 00000011 ; kết quả là 1 trong AH và 3 ; trong AL |

Chúng ta cũng nhận được kết quả như vậy nếu đem cộng AL với 6 và xoá đi nửa byte cao (các bit 4-7) của AL (tất nhiên vẫn phải cộng thêm 1 vào AH). Nguyên nhân của hiện tượng này là do kết quả của phép tính là 13 lớn hơn 10, khi ta

cộng thêm vào AL 6 kết quả nhận được sẽ lớn hơn 16. Xoá đi nửa byte cao của AL tương đương với việc trừ nó cho 16.

| | |
|-------------|---|
| AH 00000000 | AL 00001101 ;không phải là số BCD |
| + 1 | + 00000110 ;điều chỉnh bằng cách cộng thêm AL |
| AH 00000001 | AL 00010011 ;với 6 và AH với 1 |
| AH 00000001 | AL 00000011 ;xoá đi nửa byte cao của AL ;kết quả là 1 trong AH và 3 trong AL |

Lệnh AAA

8086 không có lệnh cộng số BCD nhưng nó lại có lệnh để thực hiện việc điều chỉnh trên : đó là lệnh AAA (ASCII Adjust for Addition).

Lệnh AAA không có toán hạng. nó được sử dụng để điều chỉnh giá trị BCD trong AL. Lệnh này kiểm tra nibble thấp của AL và cờ AF (auxiliary flag). Nếu nibble thấp lớn hơn 9 hay cờ AF được thiết lập thì AL được cộng thêm 6, nibble cao của nó bị xoá đi và AH được cộng thêm 1.Cả 2 cờ AF và CF đều được thiết lập nếu có sự điều chỉnh. Các cờ khác không xác định.

Cũng có thể cộng 2 ký tự biểu diễn chữ số và sử dụng lệnh AAA điều chỉnh kết quả để nhận được một số BCD. Nhờ đó chương trình có thể nhập vào 2 chữ số, cộng chúng và lưu tổng dưới dạng BCD. Ví dụ AL chứa 36h (ứng với chữ số 6) và BL chứa 37h (ứng với chữ số 7). Chúng ta sẽ cộng BL vào AL và sử dụng lệnh AAA để điều chỉnh kết quả.

| | |
|-------------|--|
| AH 00000000 | AL 00110110 |
| | BL +00110111 |
| AH 00000000 | AL 01101101 ;nibble thấp không phải số BCD |
| +1 | + 00000110 ;điều chỉnh bằng cách cộng AL với 6 |
| | ————— ;và cộng 1 vào AH |
| AH 00000001 | AL 01110011 ;sau đó xoá đi nibble cao của AL |
| AH 00000001 | AL 00000011 |

một ví dụ khác, giả sử AL chứa 39h (ứng với chữ số 9) và BL chứa 37h (ứng với chữ số 7):

```

AH 00000000 AL 00111001
          BL +00110111
AH 00000000 AL 01110000 ; nibble thấp số BCD nhưng AF được
                           thiết lập
+1   + 00000110 ; điều chỉnh bằng cách cộng AL với 6
----- ; và cộng 1 vào AH
AH 00000001 AL 01110110 ; sau đó xoá đi nibble cao của AL
AH 00000001 AL 00000110

```

Ví dụ 18.7: Viết các lệnh thực hiện phép cộng thập phân các số BCD không nén trong BL và AL.

Trả lời:

Thao tác đầu tiên là phải xoá AH rồi cộng và điều chỉnh kết quả.

```

MOV AH, 0      ; để phòng khả năng có nhớ
ADD AL, BL    ; cộng nhị phân
AAA           ; điều chỉnh kết quả, AX chứa kết quả.

```

Ví dụ 18.8: Viết các lệnh thực hiện phép cộng các số BCD có 2 chữ số trong B+1:B và A+1:A. Giả thiết rằng kết quả chỉ có 2 chữ số.

Trả lời:

Chúng ta tiến hành cộng chữ số thấp rồi đến chữ số cao.

```

MOV AH, 0      ; để phòng khả năng có nhớ
MOV AL, A      ; nạp chữ số BCD
ADD AL, B      ; cộng nhị phân
AAA           ; điều chỉnh kết quả, AX chứa kết quả
MOV A, AL      ; lưu chữ số
MOV AL, AH      ; chuyển nhớ vào AL
ADD AL, A+1    ; cộng chữ số cao vào A giả sử không
               ; cần điều chỉnh kết quả
ADD AL, B+1    ; cộng với chữ số cao của B, giả
               ; sử không cần điều chỉnh kết quả
MOV A+1, AL

```

Phép cộng các số BCD có nhiều chữ số giành cho các bạn như bài tập.

18.2.3 Phép trừ các số BCD và lệnh AAS.

Phép trừ các số BCD cũng được thực hiện từng chữ số một. Khi trừ một chữ số BCD cho một chữ số BCD có thể có nhỡ. Chẳng hạn khi chúng ta trừ 26 cho 7, chúng ta đặt 7 trong BL, 2 trong AH và 6 trong AL. Khi trừ AL cho BL thì sẽ nhận được kết quả sai. Để điều chỉnh chúng ta phải trừ AL đi 6, xoá nibble cao của nó và trừ AH đi 1. Điều này giống như chúng ta đã vay 1 từ AH và cộng thêm AL với 10.

| | | | |
|-----|----------|------------|--|
| AH | 00000010 | AL | 00000110 |
| | | BL | - 00000111 |
| AH | 00000010 | AL | 11111111 ;không phải số BCD |
| - 1 | | - 00000110 | ;điều chỉnh bằng cách trừ AL đi ;6 và trừ AH đi 1 |
| AH | 00000001 | AL | 11111001 ;sau đó xoá đi nibble cao của AL |
| AH | 00000001 | AL | 00001001 ;kết quả trong AH:AL là 19 |

Lệnh AAS

Lệnh AAS (ASCII Adjust for Subtraction) thực hiện việc điều chỉnh kết quả trong AL của phép trừ các số BCD. Nếu nibble thấp của AL lớn hơn 9 (không là chữ số BCD) hay cờ AF thiết lập thì AAS sẽ trừ AL đi 6, AH đi 1 và xoá nibble cao của AL.

Ví dụ 18.9: Viết các lệnh thực hiện phép trừ các số BCD có 2 chữ số trong B+1:B và A+1:A. Giả thiết rằng số bị trừ A+1:A lớn hơn.

Trả lời:

Chúng ta sẽ tiến hành trừ chữ số thấp trước.

| | | |
|-----|--------|--------------------------------|
| MOV | AH,A+1 | ;nạp chữ số BCD cao trong A |
| MOV | AL,A | ;nạp chữ số thấp |
| SUB | AL,B | ;trừ đi chữ số thấp của B |
| AAS | | ;điều chỉnh kết quả nếu có nhỡ |
| SUB | AH,B+1 | ;trừ đi chữ số cao của B |
| MOV | A+1,AH | ;lưu chữ số cao |
| MOV | A,AL | ;lưu chữ số thấp của hiệu. |

Trong phép trừ những chữ số cao, chúng ta có thể dùng thanh ghi AH bởi vì chúng ta giả sử không cần một sự điều chỉnh nào, AL sẽ được dùng như 1 kết quả điều chỉnh với AAS. Với phép trừ 3 chữ số, cũng bắt đầu từ số thấp nhất đến số cao nhất, sau đó cần 3 lần AAS. Phần chi tiết coi như bài tập.

18.2.4 Phép nhân các số BCD và lệnh AAM.

Trong phần này chúng tôi chỉ trình bày về phép nhân một chữ số BCD, trong phần 18.4 chúng tôi sẽ cho các bạn thấy có thể sử dụng 8087 để thực hiện phép nhân các số BCD có nhiều chữ số như thế nào. Hai chữ số BCD có thể được nhân với nhau để tạo ra một số BCD có 1 hay 2 chữ số. Chúng ta sẽ đặt số sẽ bị nhân trong AL và số nhân trong một thanh ghi hay một byte nhớ. Kết thúc phép nhân, AX sẽ chứa kết quả.

Chẳng hạn để nhân 8 với 9 chúng ta có thể đặt 8 trong AL và 9 trong BL. Sau khi thực hiện các bước của phép nhân BCD, thanh ghi AH:AL sẽ chứa tích 07 02.

Bước đầu tiên của phép nhân các số BCD là nhân các chữ số bằng phép nhân nhị phân thông thường. Tích nhị phân nhận được trong AL. Bước tiếp theo là đổi số nhị phân trong AL thành số BCD tương ứng trong AX.

Với 8 trong AL và 9 trong BL, bước đầu tiên chúng ta dùng lệnh MUL BL. Kết quả là AX có chứa 0048h = 72. Kết quả này cần được điều chỉnh để AH chứa 07 và AL chứa 02.

Lệnh AAM

Lệnh AAM (ASCII Adjust for Multiply) thực hiện bước thứ 2 của quá trình nhân. Nó chia nội dung của AL cho 10. Thương số nhận được - chính là chữ số hàng chục được đặt vào AH còn số dư - chính là chữ số hàng đơn vị được đặt vào AL. Tóm lại để nhân các số BCD trong AL và BL sau đó đưa kết quả vào AX chúng ta làm như sau:

```
MUL BL      ;phép nhân 8 bit nhị phân  
AAM          ;điều chỉnh kết quả, lưu lại trong AX.
```

18.2.5 Phép chia các số BCD và lệnh AAD

Trong phần này chúng ta sẽ nghiên cứu phép chia một số BCD có 2 chữ số cho một số BCD có một chữ số. Thương số được lưu dưới dạng một số BCD có 2 chữ số (chữ số đầu có thể bằng 0). Chúng ta đặt số bị chia trong AX và số chia

trong một thanh ghi hay một ô nhớ. Sau khi kết thúc phép chia AX sẽ chứa thương số dạng BCD. Ví dụ chúng ta đem chia 97 cho 5. Trước khi thực hiện phép chia AH:AL chứa 09 07. Số chia 5 có thể được chứa trong BL. Vì thương số là 19 nên sau phép chia BCD AH:AL = 01 09. Dưới đây là 3 bước trong phép chia BCD:

1. Đổi số bị chia trong AX từ dạng số BCD thành dạng nhị phân tương ứng.
2. Thực hiện phép chia nhị phân thông thường. Thương số nhận được ở trong AL và số dư nhận được ở trong AH.
3. Đổi thương số nhận được dưới dạng nhị phân trong AL thành dạng BCD tương ứng trong AX.

Lệnh AAD

Lệnh AAD (ASCII Adjust for Division) thực hiện bước thứ nhất trong quá trình trên. Nó đem nhân AH với 10, cộng kết quả vào AL sau đó xoá AH. Trong ví dụ của chúng ta AH chứa 09 được nhân lên thành 90=5Ah và cộng vào 07 trong AL ta có AL=61h=01100001.

Nếu số chia chứa trong BL thì bước 2 được thực hiện bằng lệnh DIV BL AL nhận thương số 13h=19 và AH nhận số dư là 02h.

Bước 3 được thực hiện bằng lệnh AAM, lệnh này đổi 13h trong AL thành 01 09 trong AH:AL. Tóm lại để thực hiện phép chia số BCD trong AX cho số BCD trong BL chúng ta có thể viết như sau:

| | |
|--------|---|
| AAD | ; đổi số bị chia BCD trong AX thành ; dạng nhị phân. |
| DIV BL | ; chia nhị phân |
| AAM | ; AX chứa thương số dạng BCD |

18.3 Các số dấu phẩy động.

Bằng cách sử dụng các số dấu phẩy động chúng ta có thể biểu diễn các số rất lớn và các phần thập phân rất nhỏ trong một dạng duy nhất. Trước khi xem xét các số dấu phẩy động chúng ta hãy xem phần thập phân được biểu diễn như thế nào dưới dạng nhị phân.

18.3.1 Đổi phần thập phân thành dạng nhị phân

Chẳng hạn chúng ta có phần thập phân $0,d_1d_2d_3...d_n$ có dạng biểu diễn nhị phân là $0,b_1b_2b_3..b_n$. Bit b_1 tương ứng với phần nguyên của tích $0,d_1d_2d_3...d_n \times 2$. Sở dĩ như vậy là vì khi chúng ta nhân đôi $0,b_1b_2..b_n$ chúng ta nhận được $b_1, b_2b_3..b_n$ do 2 tích phải bằng nhau nên phần nguyên tương ứng của chúng cũng phải bằng nhau. Nếu chúng ta nhân tiếp phần thập phân của tích chúng ta sẽ được một tích mới có phần nguyên là b_2 . Chúng ta có thể lặp lại công việc đó cho đến khi nhận được b_n . Dưới đây là thuật toán:

Thuật toán để đổi phần thập phân ra dạng nhị phân M chữ số :

```
X chứa phần thập phân cần đổi  
FOR      i=1 step 1 until m DO  
    Y= X × 2  
    X= phần thập phân của Y  
    Bi= phần nguyên của Y  
END
```

Bây giờ chúng ta hãy xem một vài ví dụ:

Ví dụ 18.10: Đổi phần thập phân 0,75 ra dạng nhị phân.

Trả lời:

Step 1 X= 0,75; Y=0,75×2=1,5, do đó $b_1=1$.

Step 2 X=0,5; Y=0,5×2=1,0 do đó $b_2=1$.

Do phần thập phân mới bằng 0 nên chúng ta dừng lại ở đây. Vậy dạng biểu diễn nhị phân của 0,75 là 0,11.

Ví dụ 18.11 : Đổi số thập phân 4,9 ra dạng nhị phân.

Trả lời:

Chúng ta thực hiện làm 2 bước. Trước tiên chúng ta đổi phần nguyên ra dạng nhị phân và nhận được kết quả là 100b. Tiếp theo chúng ta đổi phần thập phân :

Step 1 X= 0,9; Y=0,9×2=1,8 do đó $b_1=1$.

Step 2 X=0,8; Y=0,8×2=1,6 do đó $b_2=1$.

Step 3 X=0,6; Y=0,6×2=1,2 do đó $b_3=1$.

Step 4 X=0,2 Y=0,2×2=0,4 do đó $b_4=0$.

Step 5 X=0,4 Y=0,4×2=0,8 do đó $b_5=0$.

Bây giờ giá trị trong X lại bằng 0,8 chúng ta có thể thấy rằng việc tính toán là có chu kỳ và dạng biểu diễn nhị phân của 4,9 sẽ là 100,1110011001100...

18.3.2 Các số dấu phẩy động

Trong dạng biểu diễn dấu phẩy động, mỗi số được biểu diễn bằng 2 phần: phần định trị (**mantissa**) chứa các bit của số và phần mũ (**exponent**) dùng để điều chỉnh lại vị trí của dấu phẩy nhị phân trong số đó. Ví dụ số 2,5 trong dạng nhị phân là 10,1b và dạng biểu diễn dấu phẩy động của nó có phần định trị là 1,01 và phần mũ là 1 sở dĩ có như vậy là vì 10,1b có thể viết thành $1,01 \times 2^1$. Đối với các số khác 0 phần định trị được lưu như là một trị số lớn hơn 1 và -2. Phần định trị như vậy được gọi là phần định trị đã được chuẩn hoá. Một số dạng biểu diễn dấu phẩy động không chứa phần nguyên. Các số âm không được lấy bù, chúng được lưu trữ dưới dạng "độ lớn có dấu" (signed-magnitude).

Đối với các số nhỏ hơn 1, nếu chúng ta chuẩn hoá phần định trị thì phần mũ của chúng sẽ âm. Chẳng hạn số 0,0001b có dạng biểu diễn dấu phẩy động là 1×2^{-4} . Phần mũ âm không được biểu diễn như số có dấu, thay vào đó các số có tên gọi là **bias** sẽ được cộng thêm vào để tạo ra một số dương. Chẳng hạn nếu chúng ta chọn 8 bit cho phần mũ thì số $2^7-1=127$ sẽ được chọn làm bias. Để biểu diễn số 0,0001 chúng ta có phần định trị là 1 và phần mũ là -4, sau khi đã cộng thêm 127 chúng ta nhận được phần mũ là $123=01111101$. Hình 18.1 mô tả xấp xếp của một số dấu phẩy động 32 bit, nó bắt đầu bằng một bit dấu tiếp theo là 8 bit phần mũ và cuối cùng là 23 bit định trị. Chúng tôi sẽ đưa ra ví dụ trong phần 18.4.1.

| | | | | |
|----|----------|----------|----|---|
| 31 | 30 | 23 | 22 | 0 |
| S | Exponent | Mantissa | | |

Hình 18.1 số dấu phẩy động

18.3.3 Các thao tác với số dấu phẩy động.

Để thực hiện các thao tác số học với số dấu phẩy động trước tiên phần định trị và phần mũ phải được tách riêng ra, sau đó mới tiến hành các thao tác số học lần lượt lên chúng. Chẳng hạn để nhân 2 số thực chúng ta cần phải cộng các phần mũ và nhân các phần định trị. Tuy nhiên khi cộng 2 số thực thì số có phần

mũ nhỏ hơn phải được dịch phai sao cho phần mũ của nó tăng lên bằng phần mũ của số kia. Sau đó phần định trị được cộng lại và được chuẩn hoá. Cần phải nói rằng tất cả các thao tác nói trên đều tốn rất nhiều thời gian nếu giả lập bằng phần mềm. Các thao tác số học với số dấu phẩy động có thể thực hiện nhanh hơn nhiều bằng một vi mạch được thiết kế chuyên dụng.

18.4 Bộ đồng xử lý số 8087

Chip 8087 được thiết kế để thực hiện các thao tác số học trong hệ thống cơ sở 8086 hay 8088. Nó có thể thao tác trên các dạng số liệu độ chính xác kép, BCD hay dấu phẩy động.

18.4.1 Các dạng số liệu

8087 cung cấp 3 dạng số nguyên có dấu: kiểu word integer(16 bit), short integer (32 bit) và long integer (64 bit).

8087 còn cung cấp dạng số BCD nén 10 byte với một byte dấu và 9 byte theo sau chứa 18 chữ số BCD dạng nén. Dấu dương được lấy là 0h và dấu âm được lấy là 80h.

Ngoài ra còn 3 loại số dấu phẩy động:

Short real - 4 byte số liệu với 8 bit phân mũ và 24 bit định trị. Không chứa phần nguyên.

Long real - 8 byte số liệu với 11 bit phân mũ và 53 bit định trị. Cũng không chứa phần nguyên.

Temporary real - 10 byte số liệu với 15 bit phân mũ và 64 bit phân định trị. Tất cả các bit phân định trị gồm cả phần nguyên được lưu trữ.

Hình 18.2 mô tả các kiểu dữ liệu của 8087. Chúng tôi xin đưa ra một vài ví dụ.

Ví dụ 18.12 : Biểu diễn số -12345 dưới dạng số BCD nén của 8087.

Trả lời:

Đối với các số BCD àm byte dấu của nó bằng 80h. Có tổng cộng 18 chữ số BCD vậy dạng biểu diễn BCD nén là: 800000000000000012345h.

Ví dụ 18.13 Biểu diễn số 4,9 dưới dạng Short real.

Trả lời:

Áp dụng kết quả ví dụ 18.11 chúng ta có dạng biểu diễn nhị phân của 4,9 là 100,1110011001100.... Sau khi chuẩn hoá ta có phần định trị 24 bit là 1,0011100110011001100110 và phần mũ là 2. Cộng thêm bias vào phần mũ chúng ta nhận được $129 = 10000001$ b. Do phần nguyên không được lưu nên cuối cùng ta có số cần tìm là:

0 10000001 0011100110011001100 hay bằng 409CCCCCh.

Ví dụ 18.14 : Biểu diễn số -0,75 dưới dạng Short real.

Trả lời :

Áp dụng kết quả ví dụ 18.12 chúng ta có dạng biểu diễn nhị phân của 0,75 là 0,11b do đó của -0,75 sẽ là -0,11b. Phần định trị chuẩn hoá sẽ bằng 1,1 và phần mũ sẽ bằng -1. Cộng với bias bằng 127 chúng ta được phần mũ là $126 = 01111110$ b. Phần nguyên không được lưu nên chúng ta có dạng short real của -0,75 là:

1 01111110 10000000000000000000000000000000 hay bằng BF400000h.

| Dạng số liệu | phạm vi | độ chính xác | byte có trọng lượng cao nhất | | | | | | | | | | | |
|----------------|-----------------|-----------------|------------------------------|-----------------|-----------------|----------------|-----------------|-----------------|---|---|---|----------------|----------------|--------------------------|
| | | | 7 | 0 | 7 | 0 | 7 | 0 | 7 | 0 | 7 | 0 | 7 | 0 |
| Word integer | 10^4 | 16 bit | I ₁₅ | | I ₀ | Bù 2 | | | | | | | | |
| Short integer | 10^9 | 32 bit | I ₃₁ | | I ₀ | Bù 2 | | | | | | | | |
| Long integer | 10^{18} | 64 bit | I ₆₃ | | | | I ₀ | Bù 2 | | | | | | |
| Packed BCD | 10^{18} | 18 chữ số | S | D ₁₇ | D ₁₆ | | | | | | | D ₁ | D ₀ | |
| Short real | $10^{\pm 38}$ | 24 bit | S | E ₇ | E ₀ | F ₁ | F ₂₃ | | | | | | | F ₀ ngầm định |
| Long real | $10^{\pm 308}$ | 53 bit | S | E ₁₀ | E ₀ | F ₁ | | F ₆₂ | | | | | | F ₀ ngầm định |
| Temporary real | $10^{\pm 4932}$ | 64 bit | S | E ₁₄ | E ₀ | F ₀ | | | | | | | | F ₆₃ |

Integer : I

Packed BCD: (-1)^S(D₁₇..D₀)

Real: (-1)^S(2^{E-bias})(F₀F₁...)

Bias = 127 với Short real

1023 với Long real

16383 với Temp real

Hình 18.2 Các dạng dữ liệu của 8087

18.4.2 Các thanh ghi của 8087

8087 có 8 thanh ghi dữ liệu 80 bit và chúng làm việc như một ngăn xếp. Dữ liệu có thể đưa vào hay lấy ra từ ngăn xếp. Định của ngăn xếp được định địa chỉ là ST hay ST(0). Thanh ghi nằm ngay dưới đỉnh được gọi là ST(1). Một cách tổng quát thanh ghi thứ i trong ngăn xếp được định địa chỉ là ST(i) trong đó i là một hằng số.

Dữ liệu được giữ trong các thanh ghi dưới dạng Temporary real. Dữ liệu trong bộ nhớ cũng có thể được nạp vào trong ngăn xếp, trong trường hợp đó dữ liệu phải được đổi thành dạng Temporary real. Tương tự như vậy khi lưu trữ dữ liệu vào bộ nhớ nó cũng phải được đổi từ dạng Temporary real thành các dạng dữ liệu nhất định tuỳ theo cấu trúc của lưu trữ.

18.4.3 Các lệnh của 8087

Tập lệnh của 8087 bao gồm các lệnh: cộng, trừ, nhân, chia, so sánh, nạp, lưu trữ, căn bậc hai, lấy tang và luỹ thừa. Khi thực hiện các thao tác phức tạp với các số dấu phẩy động 8087 có thể làm việc nhanh gấp 100 lần so với một chương trình giả lập của 8086. Liên hệ của 8086 và 8087 như sau. 8086 có trách nhiệm nhận lệnh từ bộ nhớ. 8087 luôn giám sát quá trình này nhưng nó sẽ không làm gì cho đến khi gặp một lệnh của 8087. Các lệnh của 8087 được 8086 bỏ qua trừ khi nó chứa các toán hạng bộ nhớ. Trong các trường hợp này 8086 truy nhập dữ liệu và đặt nó lên bus dữ liệu. Đó cũng chính là cách 8087 truy nhập dữ liệu trong bộ nhớ.

Trong phần này chúng ta sẽ xem xét một vài ví dụ đơn giản về các thao tác nạp, lưu trữ, cộng, trừ, nhân và chia. Phụ lục F có những thông tin chi tiết hơn về các lệnh này và trong các phần dưới đây chúng ta cũng xem xét một số chương trình thí dụ.

Nạp và lưu trữ

Lệnh nạp sẽ nạp nội dung của toán hạng nguồn vào đỉnh của ngăn xếp 8087. Có 3 lệnh nạp :**FLD** (nạp số thực), **FILD** (nạp số nguyên) và **FBLD** (nạp số BCD nén). Cú pháp như sau:

```
FLD    source  
FILD   source  
FBLD   source
```

trong đó source là một ô nhớ.

Kiểu dữ liệu được xác định thông qua khai báo. Chẳng hạn để nạp một từ kiểu nguyên trong từ nhớ NUMBER chúng ta dùng lệnh FILD NUMBER. Nếu biến DNUM được khai báo kiểu DD (Define Double word) thì lệnh FILD DNUM sẽ nạp số có kiểu Short integer. Lệnh FLD cũng có thể dùng để nạp thanh ghi của 8087 vào ngăn xếp chẳng hạn FLD ST(3).

Một khi dữ liệu đã được nạp vào ngăn xếp chúng ta có thể đổi chúng thành kiểu bất kỳ đơn giản bằng cách lấy nó ra khỏi ngăn xếp và lưu trữ vào bộ nhớ. Đó là một cách đơn giản để sử dụng 8087 cho việc đổi kiểu dữ liệu. Bây giờ chúng ta hãy xem xét các lệnh lưu trữ dữ liệu.

Khi lưu trữ dữ liệu tại đỉnh ngăn xếp vào bộ nhớ, đỉnh ngăn xếp có thể bị đưa ra cũng có thể không. Các lệnh **FST** (store real) và **FIST** (integer store) không đưa dữ liệu khỏi đỉnh ngăn xếp, trong khi đó các lệnh **FSTP** (store real and pop), **FISTP** (integer store and pop) và **FBSTP** (packed BCD store and pop) lại đưa dữ liệu ra khỏi ngăn xếp sau khi lưu trữ. Cú pháp:

```
FST    destination  
FIST   destination  
FSTP   destination  
FISTP  destination  
FBSTP  destination
```

Trong đó destination là một ô nhớ. Kiểu dữ liệu được lưu trữ tùy thuộc vào kiểu toán hạng bộ nhớ được khai báo.

Ví dụ 18.15 : Viết các lệnh đổi biến kiểu Short integer trong DNUM thành kiểu Long real và lưu nó vào biến 4 từ có tên QNUM.

Trả lời: Chúng ta sử dụng lệnh nạp số nguyên và lệnh lưu trữ số thực.

```
FILD  DNUM      ;nạp số kiểu Short integer  
FSTP  QNUM      ;lưu trữ dưới dạng long real và đưa  
                  ;ra khỏi ngăn xếp
```

Các phép cộng, trừ, nhân và chia.

Chúng ta có thể cộng, trừ, nhân, hay chia một toán hạng bộ nhớ hay một thanh ghi của 8087 với đỉnh của ngăn xếp. Các lệnh cho toán hạng kiểu thực như sau: **FADD**(add real), **FSUB** (subtract real), **FMUL**(multiply real) và

FDIV(divide real). Mỗi lệnh có thể không có toán hạng nào hay có từ 1 đến 2 toán hạng. Lệnh không toán hạng coi ST(0) là toán hạng nguồn và ST(1) là toán hạng đích. Các lệnh này còn đồng thời đưa dữ liệu ra khỏi ngăn xếp. Chẳng hạn lệnh FADD không có toán hạng sẽ thực hiện phép cộng ST(0) vào ST(1) và đưa ra khỏi ngăn xếp. Trong các lệnh có một toán hạng, toán hạng đó xác định một ô nhớ còn toán hạng đích ngầm định là ST(0). Ví dụ để trừ một số kiểu Short real trong một biến 2 từ DWORD từ ST(0) chúng ta viết: FSUB DWORD.

Các lệnh có 2 toán hạng xác định ST(0) và ST(1) là các toán hạng của nó. Trong trường hợp này dữ liệu không đưa ra khỏi ngăn xếp. Chẳng hạn lệnh FMUL ST(0), ST(1) sẽ nhân ST(0) vào ST(1) và lệnh FDIV ST(0), ST(1) sẽ chia ST(0) cho ST(1). Cú pháp của các lệnh như sau:

```
FADD      { [ destination,] source]
FSUB      [ [ destination,] source]
FMUL      [ [ destination,] source]
FDIV      [ [ destination,] source]
```

trong đó các toán hạng trong ngoặc vuông là tùy chọn.

Ngoài ra còn các lệnh cho các số nguyên, đó là: **FIADD** (integer add), **FISUB** (integer subtract), **FIMUL** (integer multiply), **FIDIV** (integer divide). Cú pháp của các lệnh trên như sau:

```
FIADD      source
FISUB      source
FIMUL      source
FIDIV      source
```

Ví dụ 18.16: Viết các lệnh cộng 2 biến short real trong NUM1 và NUM2, lưu trữ kết quả trong NUM3.

Trả lời: Chúng ta nạp toán hạng thứ nhất, cộng với toán hạng thứ 2 rồi lưu trữ tổng trong toán hạng thứ 3.

```
FLD  NUM1          ;nạp số thứ nhất
FADD NUM2          ;cộng với số thứ 2
FSTP NUM3          ;cất tổng và đưa dữ liệu khỏi
                   ;ngăn xếp
```

18.4.4 Vào ra các số nguyên có độ chính xác cao.

Số có độ chính xác cao là số được lưu trữ trong nhiều từ nhớ. Trong phần 18.4.1 chúng ta đã được thấy một trường hợp đặc biệt đó là các số có độ chính xác kép. Thông thường việc chuyển đổi giữa các số có độ chính xác cao với dạng biểu diễn nhị phân, thập phân của nó tốn rất nhiều thời gian. Chúng ta có thể dùng 8087 để nâng cao tốc độ của quá trình này. Để vào một số thập phân có độ chính xác cao và đổi nó thành dạng nhị phân, trước tiên chúng ta lưu trữ nó dưới dạng số BCD. Sau đó 8087 sẽ đổi từ dạng BCD thành dạng nhị phân. Để đưa ra một số nhị phân có độ chính xác cao trước tiên dùng 8087 đổi nó thành dạng BCD rồi đưa ra các chữ số BCD. Thuật toán để đọc các chữ số và đổi nó thành dạng BCD nén như sau:

Thuật toán để đổi các ký tự ASCII biểu diễn chữ số thành dạng BCD nén

```
Đọc chữ số đầu tiên
CASE   '-'      : thiết lập bit dấu của BCD
       '0',...,'9' : đổi thành dạng nhị phân và
                      đưa vào ngăn xếp
WHILE ký tự <> CR
    đọc ký tự
    CASE '0'...,'9' : đổi thành dạng nhị phân
                      và đưa vào ngăn xếp
END WHILE
REPEAT:
    Lấy dữ liệu ra khỏi ngăn xếp
    ghép 2 chữ số vào 1 byte
UNTIL tất cả các chữ số được lấy ra.
```

Thuật toán được viết thành chương trình trong thủ tục READ_INTEGER. Thủ tục này còn thực hiện việc đổi các số BCD thành dạng temporary real. Chúng ta có thể đổi các số thành các dạng nhị phân khác bằng cách thay đổi các lệnh lưu trữ. Thủ tục READ_INTEGER được viết trong chương trình nguồn PGM18_2.ASM. Bộ đệm vào có kích thước 10 byte và được khởi tạo bằng các giá trị 0. Chúng ta cũng giả thiết rằng số nhập vào có tối đa 18 chữ số.

Chương trình nguồn PGM18_2.ASM

```
READ_INTEGER PROC
;đọc và lưu trữ một số nguyên, độ chính xác kép như
;số thực
;Vào:    BX = địa chỉ vùng đệm của 10 byte 0
        XOR    BP,BP ;BP=số đếm các chữ số đọc được
        MOV    SI,BX      ;chép vào con trỏ
;đọc các chữ số và đưa vào ngăn xếp
        MOV    AH,01      ;đọc chữ số
        INT    21H
;kiểm tra xem có phải số âm?
        CMP    AL,'-'
        JNE    RI_LOOP1 ;không phải số âm
;đúng là số âm, thiết lập byte dấu bằng 80h
        MOV    BYTE PTR [BX+9],80H;đọc ký tự tiếp theo
        INT    21H
;kiểm tra xem có phải ký tự CR?
RI_LOOP1:
        CMP    AL,0DH ;CR?
        JE    RI_1       ;đúng,nhảy đến RI_1
;là chữ số, đổi thành dạng nhị phân và đưa vào ngăn xếp
        AND    AL,0FH      ;đổi từ dạng mã ASCII
                           ;thành dạng nhị phân
        INC    BP          ;tăng bộ đếm
        PUSH   AX          ;cất vào ngăn xếp
        MOV    AH,01      ;đọc ký tự tiếp theo
        INT    21H
        JMP    RI_LOOP1 ;lặp lại
;lấy số ra khỏi ngăn xếp và lưu nó như số BCD nén
RI_1:
        MOV    CL,4       ;bộ đếm để dịch trái
RI_LOOP2:
        POP    AX          ;chữ số thấp
        MOV    [BX],AL     ;lưu lại
        DEC    BP          ;còn chữ số nào nữa không?
        JE    RI_4       ;hết, kết thúc vòng lặp
        POP    AX          ;còn, lấy ra chữ số cao
        SHL    AL,CL      ;dịch lên nibble cao
        OR    [BX],AL     ;lưu lại
        INC    BX          ;byte tiếp theo
        DEC    BP          ;còn chữ số nào nữa không
        JG    RI_LOOP2 ;còn, lặp lại
;đổi thành dạng số thực
RI_4:
```

```

FBLD    TBYTE PTR [ SI] ;nạp số BCD vào ngăn
                    ;xếp của 8087
FSTP    TBYTE PTR [ SI] ;lưu số thực vào bộ nhớ
RET
READ_INTEGER ENDP

```

Một khi số đã được đổi thành dạng nhị phân, chúng ta có thể cộng, trừ, nhân, chia chúng. Nếu không có hiện tượng tràn xảy ra chúng ta có thể lưu chúng như là số BCD và in ra kết quả dưới dạng số thập phân bằng thuật toán dưới đây.

Thuật toán in ra các số BCD nén

```

IF bit dấu được lập THEN in ra dấu '-'
Lấy byte cao
FOR 9 lần DO
    đổi chữ số BCD cao thành dạng Mã ASCII và đưa ra
    đổi chữ số BCD thấp thành dạng Mã ASCII và đưa ra
    Lấy byte tiếp theo

```

Thuật toán được viết thành thủ tục PRINT_BCD trong chương trình nguồn PGM18_3.ASM.

Chương trình nguồn PGM18_3.ASM

```

PRINT_BCD PROC
;in ra số BCD trong vùng đệm
;Vào:    BX=địa chỉ vùng đệm 10 byte
TEST    BYTE PTR [ BX+9],80H ;kiểm tra bit dấu
        JE    PB_1           ;số dương, bỏ qua
        MOV   DL,'-'          ;số âm, in ra dấu âm
        MOV   AH,2
        INT   21H

PB_1:
        ADD   BX,8            ;bắt đầu với chữ số có
                            ;trọng lượng cao nhất
        MOV   CH,9            ;9 byte cả thảy
        MOV   CL,4            ;bộ đếm để dịch 4 lần

PB_LOOP:
        MOV   DL,[ BX]         ;lấy byte ra
        SHR   DL,CL            ;lấy nibble cao
        OR    DL,30H            ;đổi thành mã ASCII
        MOV   AH,2            ;in ra
        INT   21H
        MOV   DL,[ BX]         ;lấy byte ra một lần nữa
        AND   DL,0FH            ;xoá nibble cao

```

```

        OR     DL, 30H; đổi chữ số thấp thành mã ASCII
        MOV    AH, 2           ;in ra
        INT    21H
        DEC    BX             ;BYTE tiếp theo
        DEC    CH             ;còn nữa không?
        JG    PB_LOOP         ;còn, tiếp tục
        RET
PRINT_BCD ENDP

```

Khi kết hợp các lệnh của 8086 và 8087 trong chương trình chúng ta cần phải bảo đảm 8086 không truy nhập dữ liệu là kết quả các thao tác của 8087 khi 8087 chưa kết thúc các thao tác đó và lưu trữ kết quả. Để đồng bộ 8086 và 8087 chúng ta sử dụng lệnh **FWAIT**, lệnh này sẽ tạm treo 8086 cho đến khi 8087 thực hiện xong các thao tác của mình.

Chương trình nguồn PGM18_4.ASM đọc vào 2 số có độ chính xác cao và đưa ra tổng, hiệu, tích, thương của chúng.

Chương trình nguồn PGM18_4.ASM

```

TITLE      PGM18_4:          MULTIPLE PRECISION ARITHMETIC
; vào 2 số thực
; ra tổng, hiệu, tích, thương
.MODEL    SMALL
.8087
.STACK
NUM1      DT  0
NUM2      DT  0
SUM       DT ?
DIFFRENCE DT ?
PRODUCT   DT ?
QUOTIENT DT ?
CR        EQU  0DH
LF        EQU  0AH
;
NEW_LINE MACRO ;in ra các ký tự điều khiển CR và LF
                MOV   DL,CR
                MOV   AH,02
                INT   21H
                MOV   DL,LF
                INT   21H
ENDM
;
DISPLAY  MACRO X           ;in ra X
                MOV   DL,X
                MOV   AH,2
                INT   21H
ENDM

```

```

.CODE
;ghép các thủ tục vào ra
INCLUDE PGM18_2.ASM
INCLUDE PGM18_3.ASM
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX ;khởi tạo DS
    MOV ES, AX ;khởi tạo ES
    DISPLAY '?'
    LEA BX, NUM1 ;BX trỏ tới vùng đệm
    CALL READ_INTEGER ;đọc vào số thứ nhất
    NEW_LINE
    DISPLAY '?'
    LEA BX, NUM2 ;BX trỏ tới vùng đệm
    CALL READ_INTEGER ;đọc vào số thứ 2
    NEW_LINE
;tính tổng
    FLD NUM1 ;nạp số thứ nhất
    FLD NUM2 ;nạp số thứ 2
    FADD ;tính tổng
    FBSTP SUM ;lưu trữ và đưa ra khỏi ngăn xếp
    FWAIT ;đồng bộ 8086 và 8087
    LEA BX, SUM ;BX trỏ tới SUM
    CALL PRINT_BCD ;in ra tổng
    NEW_LINE
;tính hiệu
    FLD NUM1 ;nạp số thứ nhất
    FLD NUM2 ;nạp số thứ 2
    FSUB ;tính hiệu
    FBSTP DIFFERENCE ;lưu trữ và đưa ra khỏi ngăn xếp
    FWAIT ;đồng bộ 8086 và 8087
    LEA BX, DIFFERENCE ;thiết lập con trỏ
    CALL PRINT_BCD ;in ra hiệu
    NEW_LINE
;tính tích
    FLD NUM1 ;nạp số thứ nhất
    FLD NUM2 ;nạp số thứ 2
    FMUL ;tính tích
    FBSTP PRODUCT ;lưu trữ và đưa ra khỏi ngăn xếp
    FWAIT ;đồng bộ 8086 và 8087
    LEA BX, PRODUCT ;thiết lập con trỏ
    CALL PRINT_BCD ;in ra tích
    NEW_LINE
;tính thương
    FLD NUM1 ;nạp số thứ nhất
    FLD NUM2 ;nạp số thứ 2
    FDIV ;tính thương
    FBSTP QUOTIENT ;lưu trữ và đưa ra khỏi ngăn xếp

```

```

FWAIT           ;đồng bộ 8086 và 8087
LEA     BX, QUOTIENT ;thiết lập con trỏ
CALL    PRINT_BCD   ;in ra tổng
NEW_LINE
;DOS exit
MOV     AH, 4CH
INT    21H
MAIN    ENDP
END     MAIN

```

18.4.5 Vào ra các số thực

Các số với phần thập phân gọi là các số thực. Thuật toán đọc vào các số thực cũng giống như đối với các số nguyên. Các chữ số được đọc vào như là các số BCD sau đó được đổi thành dạng số có dấu phẩy động và chỉnh lại. Để thực hiện việc chỉnh lại, một bộ đếm sẽ đặt bằng số các chữ số sau dấu phẩy.

Thuật toán đọc các số thực.

```

REPEAT: đọc ký tự
        CASE '-' : đặt bit dấu của vùng đệm BCD
        '.' : Thiết lập cờ
        '0'...'9' : đổi thành dạng nhị phân
                      và đưa vào ngăn xếp, nếu cờ được
                      thiết lập thì tăng bộ đếm
UNTIL CR
RPEAT:
        lấy số liệu ra khỏi ngăn xếp
        ghép 2 chữ số vào 1 byte
UNTIL lấy hết tất cả các chữ số
       nạp số BCD vào ngăn xếp của 8087
       chia cho số đếm khác 0
       lưu trả lại bộ nhớ như là số thực.

```

Thuật toán được viết thành thủ tục READ_FLOAT trong chương trình nguồn PGM18_5.ASM

Chương trình nguồn PGM18_5.ASM

```

READ_FLOAT      PROC
; đọc và lưu trữ số thực
; Vào: Bx= địa chỉ của vùng đệm 10 byte 0

```

```

        XOR  DX,DX      ;DH=1 với dấu phẩy thập phân
              ;DL=số các chữ số nằm sau dấu phẩy
        XOR  BP,BP      ;BP=số đếm các chữ số đọc được
        MOV  SI,BX      ;chép vào con trỏ
;đọc các chữ số và đưa vào ngăn xếp
        MOV  AH,01      ;đọc chữ số
        INT  21H
;kiểm tra xem có phải số âm?
        CMP  AL,'-'
        JNE  RF_1       ;không phải số âm, kiểm tra '.'
;đúng là số âm, thiết lập byte dấu bằng 80h
        MOV  BYTE PTR [ BX+9],80H
        JMP  RF_LOOP1 ;đọc ký tự tiếp theo
RF_1:
        CMP  AL,'.'    ;Dấu phẩy thập phân?
        JNE  RF_2       ;không, kiểm tra xem có phải CR?
;đúng là dấu phẩy thập phân, đặt DH=1
        INC  DH
        JMP  RF_LOOP1 ;đọc ký tự tiếp theo
;kiểm tra xem có phải ký tự CR?
RF_2:
        CMP  AL,0DH    ;CR?
        JE   RF_3       ;đúng, nhảy đến RF_3
;là chữ số, đổi thành dạng nhị phân và đưa vào ngăn xếp
        AND  AL,0FH    ;đổi từ dạng mã ASCII thành dạng
                          ;nhị phân
        INC  BP        ;tăng bộ đếm
        PUSH AX        ;cất vào ngăn xếp
        CMP  DH,0       ;đã thấy dấu phẩy thập phân chưa?
        JE   RF_LOOP1 ;chưa, đọc tiếp ký tự
        INC  DL        ;rồi, tăng bộ đếm
        JMP  RF_LOOP1 ;đọc ký tự tiếp theo
;lấy số ra khỏi ngăn xếp và lưu nó như số BCD nén
RF_3:
        MOV  CL,4       ;bộ đếm để dịch trái
RF_LOOP2:
        POP  AX        ;chữ số thấp
        MOV  [ BX],AL   ;lưu lại trong bộ đệm
        DEC  BP        ;còn chữ số nào nữa không?
        JE   RF_4       ;hết, kết thúc vòng lặp
        POP  AX        ;còn, lấy ra chữ số cao
        SHL  AL,CL     ;dịch lên nibble cao
        OR   [ BX],AL   ;lưu lại
        INC  BX        ;byte tiếp theo
        DEC  BP        ;còn chữ số nào nữa không ?
        JG   RF_LOOP2 ;còn, lặp lại
;đổi thành dạng số thực
RF_4:

```

```

        FB LD T BYTE PTR [ SI] ;nạp số BCD vào ngăn xếp
                                ;của 8087
        FWAIT                 ;đồng bộ 8086 và 8087
        CM P DL, 0            ;có chữ số phần thập phân?
        JE      RF_5          ;không, không cần hiệu chỉnh,
                                ;nhảy đến RF_5
        XOR    CX, CX
        MOV    CL, DL          ;số các chữ số sau phần thập
                                ;phân trong CX
        MOV    AX, 1            ;chuẩn bị để tạo thành
        MOV    BX, 10           ; các luỹ thừa của 10

RF_LOOP3:
        IMUL   BX             ;nhân với 10
        LOOP   RF_LOOP3 ;CX lần
        MOV   [ SI], AX ;Cắt hệ số điều chỉnh
        FIDIV WORD PTR [ SI];chia cho hệ số điều chỉnh

RF_5:
        FSTP   T BYTE PTR [ SI];lưu số thực vào bộ nhớ
        FWAIT                 ;đồng bộ 8086 và 8087
        RET

READ_FLOAT    ENDP

```

Trong trường hợp này chúng ta đã giả sử số các chữ số sau dấu phẩy nhỏ hơn 5, nhờ đó hệ số hiệu chỉnh có thể được lưu như một số nguyên một từ có dấu.

Để đưa ra số thực, trước hết chúng ta đem nhân nó với hệ số hiệu chỉnh, sau đó chúng lưu số thực dưới dạng BCD, rồi in ra với dấu phẩy thập phân ở vị trí thích hợp. Chúng ta chỉ in ra 4 chữ số sau dấu phẩy, do đó hệ số hiệu chỉnh là 10000.

Thuật toán in ra số thực với 4 chữ số thập phân.

Nhân số thực với 10000
 lưu nó như là số BCD
 In ra số BCD với dấu '.' chèn vào trước 4 chữ số
 cuối cùng.

Thuật toán được viết thành thủ tục PRINT_FLOAT trong chương trình nguồn PGM18_6.ASM.

Chương trình nguồn PGM18_6.ASM

```

PRINT_FLOAT    PROC
;in ra định của ngăn xếp 8087
;          Vào:Bx=địa chỉ của bộ đệm
;

```

```

MOV WORD PTR [ BX],10000;hệ số hiệu chỉnh cho 4
                                ;chữ số sau dấu phẩy
FIMUL WORD PTR [ BX]      ;nhân lên với hệ số
                                ;hiệu chỉnh
FBSTP TBYE PTR [ BX]      ;lưu lại như là số BCD
FWAIT                      ;đồng bộ 8086 với 8087
TEST  BYTE PTR [ BX+9],80H ;kiểm tra bit dấu
JE    PF_1                  ;bằng 0, nhảy đến PF_1
MOV   DL,'-'                ;in ra dấu '-'
MOV   AH,2
INT   21H

PF_1:
ADD   BX,8                  ;trở đến byte cao
MOV   CH,7                  ;có 14 chữ số trước dấu phẩy
MOV   CL,4                  ;dịch 4 lần
MOV   DH,2                  ;lặp 2 lần

PF_LOOP:
MOV   DL,[ BX]              ;lấy ra chữ số BCD
SHR   DL,CL                ;lấy ra nibble cao bằng
                            ;cách dịch nó vào nibble thấp
OR    DL,30H                ;đổi thành mã ASCII
INT   21H                  ;in ra
MOV   DL,[ BX]              ;chuyển byte ra một
                            ;lần nữa
AND   DL,0FH                ;xoá nibble cao
OR    DL,30H                ;đổi thành mã ASCII
INT   21H                  ;in ra
DEC   BX                    ;byte tiếp theo
DEC   CH                    ;giảm bộ đếm
JG    PF_LOOP              ;tiếp tục nếu vẫn còn byte
                            ;lần thứ 2 (đưa ra các
                            ;chữ số sau dấu phẩy chưa)
                            ;rồi, kết thúc
DISPLAY '.'                 ;Chưa, hiển thị dấu '.'
MOV   CH,2                  ;còn 4 chữ số sau dấu phẩy
JMP   PF_LOOP              ;lặp lại công việc in các
                            ;chữ số ra màn hình

PF_DONE:
RET
PRINT_FLOAT ENDP

```

Chương trình để kết nối các thủ tục trên đây xem như bài tập.

TỔNG KẾT

Trong chương này chúng ta nghiên cứu những vấn đề sau:

- ◆ Bằng cách sử dụng các số có độ chính xác kép, phạm vi biểu diễn của các số nguyên tăng lên.
- ◆ Các lệnh ADC và SBB được dùng trong thao tác cộng và trừ các số có độ chính xác kép.
- ◆ Nhân và chia các số có độ chính xác kép cho các luỹ thừa của 2 có thể thực hiện bằng các lệnh dịch hay quay.
- ◆ Trong hệ thống số BCD, các chữ số thập phân được biểu diễn bằng 4 bit. Một số được lưu dưới dạng nén nếu một byte chứa đến 2 chữ số BCD, còn trong dạng không nén chỉ có một chữ số BCD trong mỗi byte.
- ◆ Ưu điểm của các số BCD là dễ dàng đổi các ký tự biểu diễn chữ số thập phân thành các số BCD và ngược lại. Nhược điểm của nó là đối với máy tính các thao tác số học với số thập phân phức tạp hơn nhiều so với số nhị phân.
- ◆ Lệnh AAA dùng để điều chỉnh kết quả trong AL sau khi thực hiện phép cộng.
- ◆ Lệnh AAS dùng để điều chỉnh hiệu trong AL sau khi thực hiện phép trừ.
- ◆ Lệnh AAM nhận tích của 2 chữ số BCD trong AL và đổi thành số 2 chữ số BCD trong AH:AL.
- ◆ Lệnh AAD đổi số bị chia có 2 chữ số BCD trong AH:AL thành giá trị nhị phân tương đương của nó trong AL.
- ◆ Dạng biểu diễn số dấu phẩy động bao gồm bit dấu, phần mũ và phần định trị.
- ◆ 8087 có thể thực hiện rất nhiều các thao tác khác nhau trên các dữ liệu kiểu nguyên, BCD hay kiểu thực.

Các thuật ngữ tiếng Anh.

| | |
|--|---|
| BCD (Binary-Coded Decimal) system | Hệ thống số BCD, trong đó mỗi chữ số thập phân được mã hoá bằng 4 bit nhị phân |
| Bias | Một số cộng thêm vào phần mũ để chúng lớn hơn 0. |
| Double-precision numbers | Số có độ chính xác kép - Các số chiếm 2 từ bộ nhớ của máy tính. |
| Exponent | Phần mũ - một phần trong dạng biểu diễn của số có dấu phẩy động. |
| Mantissa | Phần định trị - một phần trong dạng biểu diễn của số có dấu phẩy động. Chứa các chữ số có nghĩa của nó. |
| Multiple-precision numbers | Số có độ chính xác cao - các số chiếm nhiều từ nhớ. |
| Packed BCD form | Số BCD nén - số BCD mà 2 chữ số của nó nằm trong một byte. |
| Unpacked BCD form | Số BCD không nén - số BCD mà mỗi byte chỉ chứa 1 chữ số. |

Các lệnh mới

| | | |
|-------|-------|-------|
| AAA | FDIV | FLD |
| AAD | FIADD | FMUL |
| AAM | FIDIV | FST |
| AAS | FILD | FSTP |
| ADC | FIMUL | FSUB |
| FADD | FIST | FWAIT |
| FBLD | FISTP | SBB |
| FBSTP | FISUB | |

Toán tử giả mới

.8087

Bài tập

Trong các bài tập từ 1 đến 6 chỉ dùng các lệnh của 8086.

1. Viết thủ tục DSUB thực hiện việc trừ số có độ chính xác kép với số trong CX:BX từ số trong DX:AX. Hiệu trả lại trong DX:AX. DSUB phải thiết lập đúng các cờ.
2. Viết thủ tục DCMP thực hiện việc so sánh các số có độ chính xác kép trong CX:BX và DX:AX, không được làm thay đổi các thanh ghi, các cờ phải được thiết lập đúng.
3. Viết các chỉ thị thực hiện các thao tác sau đây với các số có độ chính xác kép. Giả sử rằng số nằm trong DX:AX. Thực hiện các phép quay và dịch một vị trí:
 - a. SHR
 - b. SAR
 - c. ROR
 - d. ROL
 - e. RCR
 - f. RCL
4. Một số có độ chính xác bậc 3 là số chiếm 3 từ nhớ (48 bit). Viết các lệnh thực hiện các thao tác sau trên 2 số có độ chính xác bậc 3 trong A+4:A+2:A và B+4:B+2:B.
 - a. Cộng số thứ 2 vào số thứ nhất.
 - b. Trừ số thứ nhất cho số thứ 2.
5. Viết các lệnh thực hiện phép dịch phải số học số có độ chính xác bậc 3 trong BX:DX:AX.
6. Giả sử 2 số BCD không nén có 3 chữ số chứa trong A+2:A+1:A và B+2:B+1:B. Viết các lệnh thực hiện những công việc sau:
 - a) Cộng số thứ 2 vào số thứ nhất, giả sử tổng chỉ có 3 chữ số.
 - b) trừ số thứ nhất cho số thứ 2, giả sử số thứ nhất lớn hơn.
7. Biểu diễn số -0,0014 dưới dạng short real của 8087.
8. Biểu diễn số -29544683 dưới dạng số BCD nén của 8087.
9. Viết các lệnh thao tác với các số dấu phẩy động thực hiện những công việc sau:
 - a. Cộng biến nguyên X vào đinh ngan xếp.
 - b. Chia số dạng short real Y cho số nằm tại đinh ngan xếp.
 - c. Lưu và lấy ra khỏi ngan xếp số BCD Z.

Các bài tập lập trình.

10. Viết một chương trình đọc vào 2 số thập phân từ bàn phím và đưa ra tổng của chúng. Các số có thể âm và có tới 20 chữ số. Chú ý không sử dụng các chỉ thị của 8087.
11. Viết một chương trình đọc vào 2 số thực có tới 4 chữ số sau dấu phẩy và đưa ra tổng, hiệu, tích và thương của chúng.

Chương 19

CÁC THAO TÁC ĐĨA VÀ TẬP TIN

Tổng quan

Cho đến nay chúng ta đã sử dụng đĩa từ như một nơi chứa các file hệ thống và chương trình của người sử dụng. Các file trên đĩa còn có thể dùng để lưu các dữ liệu vào và ra cho một chương trình. Các ví dụ phổ biến là các cơ sở dữ liệu và các bảng tính điện tử. Trong chương này chúng ta sẽ nghiên cứu cấu trúc của đĩa, các thao tác đĩa và cách quản lý các file.

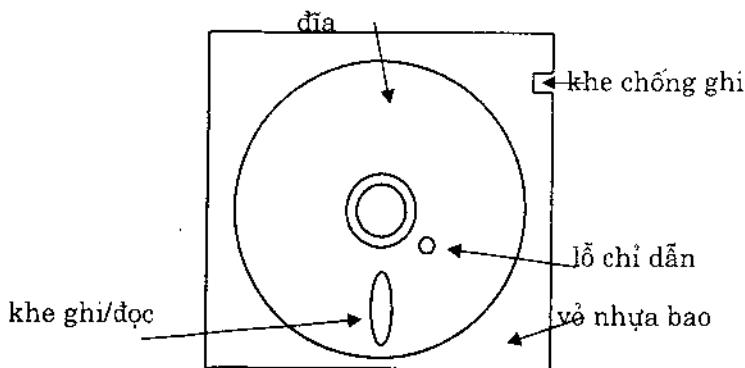
19.1 Các loại đĩa

Có 2 loại đĩa: đĩa mềm và đĩa cứng. Đĩa mềm được chế tạo từ chất dẻo và khá mềm chính vì vậy mà nó có tên gọi trên. Đĩa cứng được chế tạo bằng kim loại và rất cứng. Bề mặt của đĩa được phủ một lớp oxit kim loại và thông tin được lưu trên đĩa dưới dạng các rãnh từ hoá.

Thao tác của đĩa cứng và đĩa mềm khá giống nhau. Đơn vị điều khiển đĩa đọc và ghi dữ liệu lên đĩa bằng các đầu từ đọc và ghi. Các đầu từ này di chuyển trên bề mặt đĩa đọc theo bán kính của nó trong khi đĩa đang quay. Mỗi đầu từ lướt trên một đường vòng tròn gọi là rãnh trên mặt đĩa (**track**). Sự chuyển động của đầu từ đọc ghi cho phép nó truy nhập tới các rãnh khác nhau.

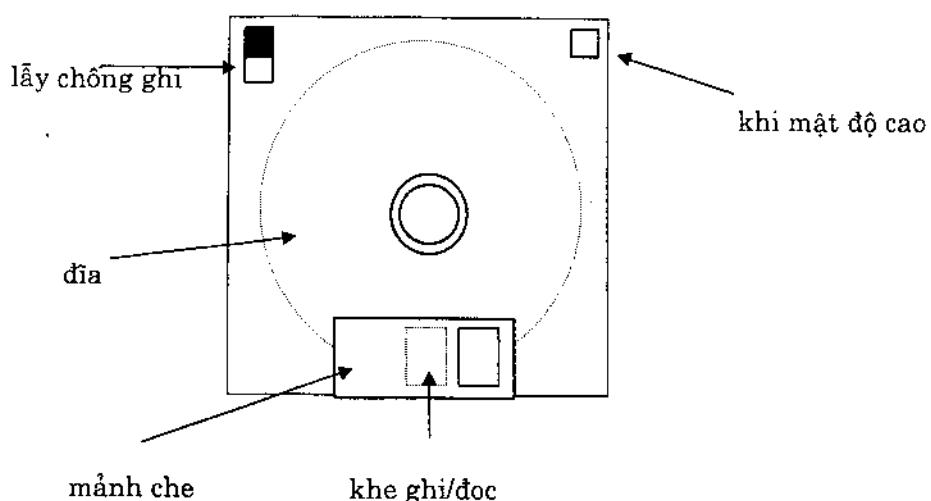
Đĩa mềm

Đĩa mềm được chứa trong một vỏ nhựa bảo vệ và có các loại khác nhau với kích thước đường kính là $3\frac{1}{2}$ và $5\frac{1}{4}$ inch. Vỏ của đĩa $5\frac{1}{4}$ inch được làm bằng nhựa dẻo và có 4 phần trống như hình vẽ 19.1. (1) Phần trống ở giữa để ổ đĩa có thể áp sát vào đó và quay đĩa; (2) một lỗ trống hình ô val để đầu từ đọc ghi có thể truy nhập dữ liệu trên đĩa; (3) một lỗ nhỏ hình tròn sẽ được bố trí với một lỗ chỉ số trên đĩa để ổ đĩa xác định vị trí bắt đầu một rãnh và (4) một khe chống ghi nếu khe để hở, đĩa có thể đọc và ghi, nếu khe bị dán kín, máy chỉ có thể đọc đĩa.



Hình 19.1 A5 $\frac{1}{2}$ inch Floppy Disk

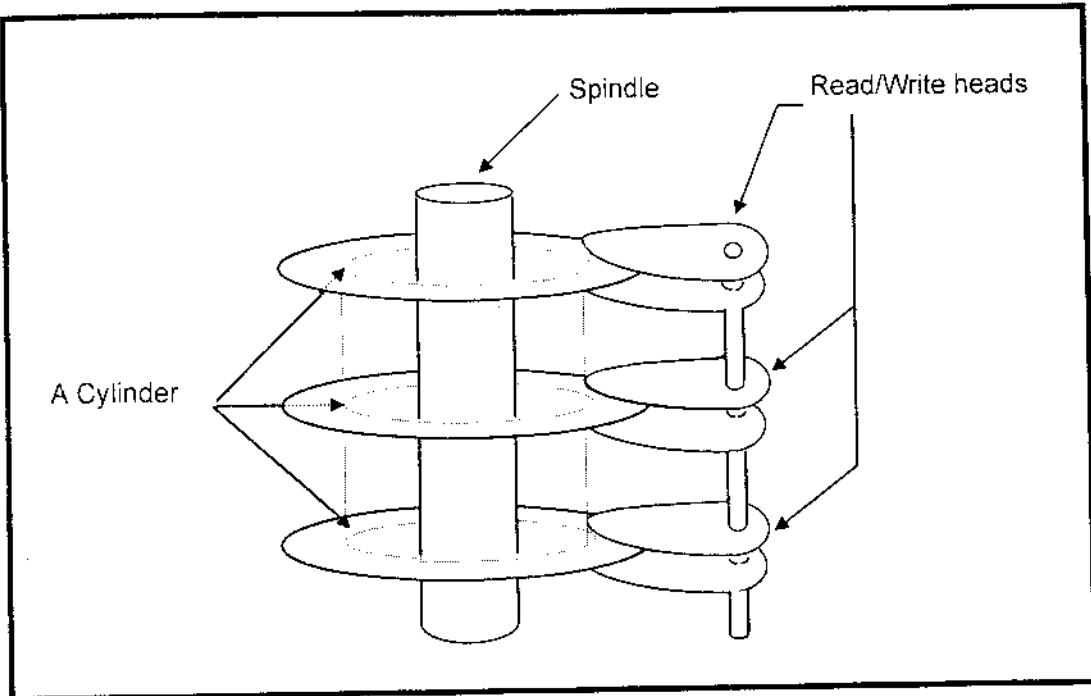
Đĩa $3\frac{1}{2}$ có cấu trúc cứng hơn. Vỏ bảo vệ của nó được làm bằng nhựa cứng nhờ đó đĩa trở lên cứng hơn. Nó có một moay ổ bằng kim loại nhờ đó sử dụng được lâu bền hơn. Ngoài ra nó còn một vỏ trượt bằng kim loại để mở ra cho đầu từ đọc ghi truy nhập dữ liệu. Lỗ chống ghi hoạt động khác với đĩa $5\frac{1}{4}$, đĩa sẽ không thể ghi nếu lỗ để trống. Đĩa $3\frac{1}{2}$ không có lỗ chỉ số. (xem hình 19.2)



Hình 19.2 A3 $\frac{1}{2}$ inch Floppy Disk

Đĩa cứng

Đĩa cứng bao gồm nhiều đĩa phẳng được ráp vào một trục quay chung. Cả 2 mặt của mỗi đĩa đều có thể sử dụng để ghi thông tin. Mỗi mặt của một đĩa có một đầu từ đọc ghi riêng. Tất cả các đầu từ đều gắn trên một đơn vị chuyển động chung. Xem hình 19.3.



Khác với đĩa mềm các đầu từ đọc ghi của đĩa cứng chỉ lướt trên mặt đĩa mà không bao giờ thực sự chạm vào chúng trong khi làm việc. Khoảng trống giữa đầu từ và bề mặt đĩa nhỏ tới mức mà chỉ cần một hạt bụi nhỏ cũng làm cho đầu từ cào lên mặt đĩa và làm hỏng đĩa. Chính vì thế cả đĩa cứng và ổ đĩa đều phải được đặt trong một hộp bọc kín.

Đĩa cứng được truy nhập nhanh hơn nhiều so với đĩa mềm vì mấy nguyên nhân sau: (1) đĩa cứng luôn luôn quay do đó không mất thời gian để khởi động nó, (2) đĩa cứng quay nhanh hơn nhiều so với đĩa mềm, khoảng 3600 vòng trên một phút trong khi tốc độ quay của đĩa mềm chỉ vào khoảng 300 vòng trên một phút và (3) do đĩa cứng có bề mặt cứng hơn nhiều lại được đặt trong môi trường không có bụi nên mật độ thông tin trên đĩa cứng lớn hơn rất nhiều mật độ thông tin trên đĩa mềm.

19.2 Cấu trúc của đĩa

Thông tin trên đĩa được lưu trên các rãnh (track). Khi đĩa được định dạng (format), các track được chia thành các phần 512 byte được gọi là các cung từ (sector). DOS đánh số các track bắt đầu từ 0. Trong một track, các sector lại được đánh số, bắt đầu từ 1. Số rãnh trên đĩa, số cung trên một rãnh tùy thuộc vào từng loại đĩa.

Trụ (Xy_lanh **cylinder**) tập hợp các rãnh có cùng số hiệu. Ví dụ trụ số 0 của đĩa mềm chứa rãnh 0 trên mỗi mặt đĩa. Đối với đĩa cứng trụ số 0 chứa các rãnh 0 trên cả 2 mặt của mỗi đĩa. Sở dĩ các trụ (cylinder) có tên gọi như vậy là vì các rãnh có cùng số hiệu nằm vuông góc với nhau và dường như tạo thành một trụ xy lanh vật lý. Số trụ (cylinder) trên đĩa bằng số rãnh trên mỗi mặt đĩa.

DOS cũng đánh số mặt đĩa bắt đầu từ 0. Đĩa mềm có 2 mặt 0 và 1 trong khi đĩa cứng có thể có nhiều mặt vì nó có thể chứa nhiều đĩa bên trong.

Dung lượng đĩa.

Dung lượng tính bằng byte của đĩa có thể được tính như sau:

Dung lượng đĩa (bytes) = số mặt × số track trên một mặt × số sector trên một track × số byte trên một sector.

Ví dụ đĩa $5^{1/4}$ có dung lượng:

Dung lượng = 2 mặt × 40 track/mặt × 9 sector/track × 512 byte/sector = 368.640 byte.

Các bảng 19.1A và 19.1B cho biết số cylinder, sector/track, số mặt và dung lượng của một số loại đĩa đang được sử dụng hiện nay.

Mật độ thông tin trên đĩa tuỳ thuộc vào kỹ thuật ghi. Các kỹ thuật ghi khá phổ biến là mật độ kép và mật độ cao. Một ổ đĩa mật độ cao có đầu từ hẹp và nó có thể truy nhập các đĩa mật độ kép. Tuy nhiên ổ đĩa mật độ kép lại không đọc được đĩa mật độ cao.

| Loại đĩa | Số cylinder | Sector/track | Dung lượng |
|----------------------|-------------|--------------|----------------|
| $5^{1/4}$ mật độ kép | 40 | 9 | 368.640 byte |
| $5^{1/4}$ mật độ cao | 80 | 15 | 1.228.800 byte |
| $3^{1/2}$ mật độ kép | 80 | 9 | 737.2800byte |
| $3^{1/2}$ mật độ cao | 80 | 18 | 1.474560 byte |

Bảng 19.1A: Một số loại đĩa mềm thông dụng.

| Loại đĩa | Số cylinder | Sector/track | Số mặt | Dung lượng |
|----------|-------------|--------------|--------|-----------------|
| 10MB | 306 | 17 | 4 | 10.653.696 byte |
| 20MB | 615 | 17 | 4 | 21.411.840 byte |
| 30MB | 615 | 17 | 6 | 32.117.760 byte |
| 60MB | 940 | 17 | 8 | 65.454.080 byte |

Bảng 19.1B: Dung lượng một số đĩa cứng.

19.2.2 Truy nhập đĩa.

Phương pháp truy nhập đĩa mềm và đĩa cứng là giống nhau. Ở đĩa nằm dưới sự điều khiển của mạch điều khiển đĩa. Mạch điều khiển đĩa có nhiệm vụ chuyển dịch đầu từ, đọc và ghi dữ liệu trên đĩa. Dữ liệu luôn được truy nhập theo từng sector.

Đầu tiên để truy nhập dữ liệu thì đầu từ phải được đặt vào đúng track cần truy nhập. Muốn vậy phải chuyển dịch cả hệ thống đầu từ - một công việc khá chậm. Khi đầu từ đã đặt vào đúng track nó sẽ đợi cho sector cần tìm di tới, do đó lại phải tốn thêm một số thời gian nữa. Vì tất cả các track trong cylinder có thể truy nhập mà không phải di chuyển thêm đầu từ nên khi DOS ghi dữ liệu nó thường ghi đầy một cylinder rồi mới chuyển sang cylinder tiếp theo.

19.2.3 Sự phân bố các file

Để theo dõi dữ liệu ghi trên đĩa, DOS sử dụng cấu trúc thư mục. Các track và sector đầu tiên của đĩa chứa thông tin về cấu trúc file của đĩa. Chúng ta sẽ tập trung vào cấu trúc của đĩa mềm mật độ kép 5^{1/4} inch. đĩa này được tổ chức như sau:

| Mặt | Track | Sectors | Thông tin |
|-----|-------|---------|------------------------------------|
| 0 | 0 | 1 | Boot record(sử dụng khi khởi động) |
| 0 | 0 | 2-5 | Bảng phân bổ file (FAT) |
| 0 | 0 | 6-9 | Thư mục |
| 1 | 0 | 1-3 | Thư mục |
| 1 | 0 | 4-9 | Dữ liệu (khi cần) |
| 0 | 1 | 1-9 | dữ liệu (khi cần) |

Thư mục file.

DOS tạo ra một điểm nhập 32 byte cho mỗi file trong thư mục. Cấu trúc của mỗi điểm nhập như sau:

| Byte | Function |
|-------|---|
| 0-7 | Tên file (byte 0 được dùng làm byte trạng thái) |
| 8-10 | Phản mở rộng |
| 11 | Thuộc tính(xem dưới) |
| 12-21 | Để giành |
| 22-23 | Giờ tạo lập (Giờ:phút:giây) |
| 24-25 | Ngày tạo lập(năm:tháng:ngày) |

- 26-27 Số hiệu cluster đầu tiên (xem phần mô tả bảng FAT)
 28-31 Kích thước file tính bằng byte.

Có 7 sector cho một thư mục, mỗi điểm nhập chiếm 32 byte do đó thư mục có thể chứa tối $7 \times 512/32 = 112$ điểm nhập. Tuy nhiên các điểm nhập file có thể chứa trong các thư mục con.

Thư mục được tổ chức như các cây với một thư mục gốc chính là gốc của cây, và các thư mục con như là các nhánh cây.

Trong mỗi điểm nhập của thư mục byte 0 được gọi là byte trạng thái (**status byte**). Chương trình FORMAT gán 0 cho byte này, điều này có nghĩa là điểm nhập đó chưa được sử dụng. Giá trị E5H có nghĩa là file đã bị xoá, và 2EH chỉ ra rằng đó là một thư mục con. Ngoài các trường hợp trên byte 0 chứa ký tự đầu tiên của tên file.

Khi có một file mới được tạo, DOS dùng trường đầu tiên còn chưa được sử dụng để lưu trữ thông tin về file đó.

Byte 11 trong điểm nhập của thư mục là byte thuộc tính, mỗi bit trong byte mang một thuộc tính của file (xem hình 19.4).

Một file ẩn (**hidden**) là file mà tên của nó không hiện lên trong lệnh DIR của DOS. Làm ẩn một file là một biện pháp để bảo vệ file khi một số người cùng sử dụng chung một máy tính. File ẩn không thể thực hiện được với DOS 2.0 nhưng có thể thực hiện được với DOS 3.0. tuy nhiên chúng ta có thể thay đổi thuộc tính của file(xem phần 19.2.8) và có thể chạy được tất cả các file nếu muốn.

Bit lưu (**archive bit**) - bit 5 được thiết lập mỗi khi file được tạo. Nó được lệnh BACKUP sử dụng để lưu các file. Khi một file được lưu bằng lệnh BACKUP bit lưu bị xoá nhưng khi chúng ta thay đổi file thì bit lưu lại được thiết lập lại. Nhờ đó chương trình BACKUP biết được file nào đã được cất giữ.

Thuộc tính của file được tạo ra mỗi khi file được tạo, nhưng như đã nói ở trên, chúng ta có thể thay đổi thuộc tính của file. Thông thường khi một file được tạo nó có byte thuộc tính là 20h (nghĩa là tất cả các bit đều xoá trừ bit lưu).

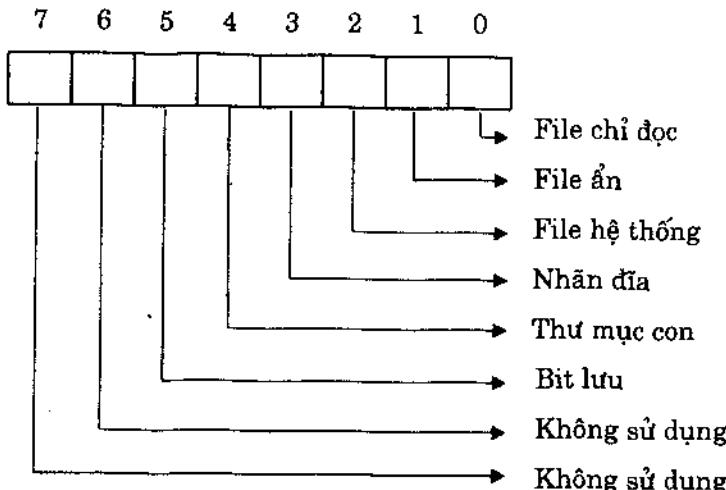
Ví dụ về điểm nhập file trong thư mục được cho trong phần 19.3.

Cluster (liên cung)

DOS lưu các file theo các liên cung (cluster). Đối với mỗi loại đĩa nhất định liên cung là một số nhất định các sector (2 cho đĩa 5^{1/4}2 mặt mật độ kép). Trong mọi trường hợp số sector trong một cluster luôn là lũy thừa của 2.

Các liên cung được đánh số với liên cung 0 là 2 cung (sector) cuối cùng của thư mục. Các byte 26 và 27 trong điểm nhập của thư mục chứa số hiệu của cluster đầu tiên của file tương ứng. File dữ liệu đầu tiên của đĩa bắt đầu tại cluster 2.

Ngay cả khi kích thước của file nhỏ hơn kích thước một cluster (1024 byte cho đĩa 5^{1/4} hai mặt mật độ kép), DOS vẫn để giành cho file đó cả một cluster. Điều đó có nghĩa là trên đĩa vẫn còn chỗ chưa sử dụng ngay cả khi DOS cho thông báo rằng đĩa đã đầy.



Hình 19.4 : byte thuộc tính

Bảng FAT

Mục đích của bảng FAT (File Allocation Table) là để tạo ra một bản đồ các file trên đĩa. Đối với các đĩa mềm và đĩa cứng 10MB mỗi điểm nhập của bảng FAT dài 12 bit, trong các đĩa cứng dung lượng lớn hơn mỗi điểm nhập của FAT dài 16 bit. Byte đầu tiên của bảng FAT được dùng để chỉ ra loại đĩa (xem bảng 19.2). Đối với FAT 12 bit, 2 byte tiếp theo chứa giá trị FFh.

| Loại đĩa | Byte đầu tiên(dạng hex) |
|----------------------------------|-------------------------|
| 5 ^{1/4} inch mật độ kép | FD |
| 5 ^{1/4} inch mật độ cao | F9 |
| 3 ^{1/2} inch mật độ kép | F9 |
| 3 ^{1/2} inch mật độ cao | F0 |
| đĩa cứng | F8 |

Bảng 19.2: Byte đầu tiên của FAT đối với một số loại đĩa.

DOS đọc một file như thế nào ?

Để xem FAT được tổ chức ra sao, chúng ta hãy lấy một ví dụ về việc DOS sử dụng FAT để đọc file như thế nào (xem hình 19.5).

1. DOS nhận số hiệu cluster đầu tiên từ thư mục, giả sử đó là 2.
2. DOS đọc cluster từ đĩa và chứa nó trong một vùng bộ nhớ gọi là vùng chuyển dữ liệu (**Data Transfer Area - DTA**), chương trình thực hiện việc đọc sẽ nhận dữ liệu từ DTA khi cần.
3. Vì điểm nhập thứ 2 chứa giá trị 4, cluster tiếp theo của file có số hiệu là 4. Nếu chương trình cần thêm dữ liệu, DOS sẽ đọc cluster vào DTA.
4. Điểm nhập 4 trong FAT chứa giá trị FFFh, giá trị này chỉ ra rằng đó là cluster cuối cùng trong file. Tóm lại quá trình lấy số hiệu cluster trong FAT là liên tục đọc dữ liệu vào DTA cho đến khi điểm nhập trong FAT chứa giá trị FFFh.

| Điểm nhập | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|-----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| | FDF | FFF | 004 | 005 | FFF | 006 | 007 | 008 | FFF | 000 |

Hình 19.5: Ví dụ về bảng FAT

Một ví dụ khác là trong hình 19.5 chúng ta thấy có một file chiếm các cluster 3,5,6,7 và 8.

DOS lưu trữ các file như thế nào?

Để lưu trữ các file DOS thực hiện những công việc sau đây:

1. DOS xác định một điểm nhập chưa sử dụng trong thư mục và lưu vào đó tên file, thuộc tính file, ngày và giờ tạo lập.
2. DOS tìm trong bảng FAT điểm nhập đầu tiên đánh dấu một cluster chưa sử dụng (giá trị 000 có nghĩa là cluster chưa sử dụng) và chứa số hiệu cluster đầu tiên của tập tin lấy trong thư mục vào đó. Chúng ta hãy giả sử nó tìm thấy giá trị 000 ở điểm nhập 9.
3. Nếu dữ liệu chưa vừa trong một cluster, DOS chứa nó trong cluster 9 và đặt giá trị FFF vào điểm nhập thứ 9 của FAT. Nếu vẫn còn dữ liệu, DOS tiếp tục tìm cluster chưa được sử dụng tiếp theo trong FAT. Ví dụ nó tìm thấy điểm nhập Ah, nó sẽ lưu dữ liệu vào cluster Ah và đặt giá trị 00A vào điểm nhập 9 của FAT. Qua trình tìm các cluster chưa được sử dụng trong FAT, chứa dữ liệu vào đó, cho điểm nhập của FAT trở tới cluster tiếp theo sẽ tiếp

tục cho đến khi dữ liệu được lưu trữ hết. Điểm nhập cuối cùng của file trong FAT sẽ chứa giá trị FFFh.

19.3 Xử lý file.

Trong phần này chúng ta sẽ nghiên cứu một nhóm các hàm của ngắt 21h được gọi là các hàm quản lý file. Các hàm này có từ DOS 2.0 và nhờ đó các thao tác file trở lên dễ dàng hơn nhiều so với cách quản lý file bằng các khối điều khiển file (File Control Block - FCB) vẫn thường dùng trước đó. Với phương pháp dùng các FCB người sử dụng phải tự tạo lên một bảng thông tin về các file mở, bằng các hàm quản lý file DOS theo dõi các file dữ liệu mở bằng các bảng bên trong của chính nó, nhờ đó giải phóng con người sử dụng khỏi một công việc vất vả. Một ưu điểm khác của các hàm quản lý file là người sử dụng có thể chỉ định tên file với đầy đủ đường dẫn, đây là điều không thể thực hiện được khi dùng các FCB.

Trong các phần sau đây, **đọc** một file có nghĩa là chép toàn bộ hay một phần dữ liệu trong đó ra bộ nhớ, **ghi** một file là chép dữ liệu trong bộ nhớ vào file và **ghi lại** một file là thay nội dung của file bằng dữ liệu mới.

19.3.1 Thẻ file (file handle).

Khi một file được mở hay được tạo lập trong chương trình, DOS gán cho nó một số xác định duy nhất gọi là thẻ file (file handle). Con số này dùng để xác định file do đó chương trình phải lưu nó.

Có 5 thẻ file được định nghĩa trước là:

| | |
|---|---------------------------------|
| 0 | Bàn phím |
| 1 | Màn hình |
| 2 | Thông báo lỗi đầu ra - màn hình |
| 3 | Thiết bị phụ |
| 4 | Máy in |

Ngoài các file này ra DOS cho phép người sử dụng định nghĩa 3 file được mở (có khả năng tăng số file được mở của người sử dụng (xem tài liệu tham khảo về DOS)).

19.3.2 Lỗi thao tác file.

Có rất nhiều khả năng gây ra lỗi trong các hàm quản lý file của ngắt 21h. DOS xác định mỗi lỗi thông qua mã lỗi. Trong các hàm chúng tôi trình bày ở đây, mỗi khi có lỗi, cờ CF được lập và mã lỗi tương ứng sẽ được đặt trong AX. Dưới đây là mã của một số lỗi thường gặp.

| Mã lỗi (dạng hex) | Ý nghĩa |
|-------------------|-------------------------------------|
| 1 | Số hàm không hợp lệ |
| 2 | Không tìm thấy file |
| 3 | Không tìm thấy đường dẫn |
| 4 | Không còn thẻ file |
| 5 | Từ chối truy nhập |
| 6 | Thẻ file không hợp lệ |
| C | Mã truy nhập không hợp lệ |
| F | Đĩa không hợp lệ |
| 10 | Đang tìm cách xoá thư mục hiện thời |
| 11 | Không cùng thiết bị |
| 12 | Không tìm được thêm file nào. |

Trong các phần dưới đây, chúng ta sẽ nghiên cứu về các hàm quản lý file của DOS. Khi sử dụng các hàm vào ra của DOS, chúng ta đặt số hàm vào thanh ghi AH và gọi ngắt 21h.

19.3.3 Đóng mở file.

Trước khi sử dụng một file, chúng ta phải mở nó. Để tạo một file mới hay ghi lại một file cũ, người sử dụng cho tên file và thuộc tính của file; DOS sẽ trả lại thẻ file.

INT 21h, Hàm 3Ch:

Mở một file mới/Ghi lại một file

| | |
|------|--|
| Vào: | AH=3Ch DS:DX=địa chỉ của chuỗi ASCII (chuỗi tên file kết thúc bằng byte 0) CL= thuộc tính |
| Ra: | Nếu thành công, AX=thẻ file Nếu CF được đặt thì có lỗi, mã lỗi trong AX (3,4 hay 5) |

Tên file có thể bao gồm cả đường dẫn, ví dụ A:PROGS\PGM18_2.ASM. Các lỗi có thể có với hàm này là 3 (đường dẫn không tồn tại), 4 (hết thẻ file) và 5 (từ chối truy nhập nghĩa là thư mục đầy hay file có thuộc tính chỉ đọc).

Ví dụ 19.1: Viết các chỉ thị mở một file mới có thuộc tính chỉ đọc tên là FILE1.

Trả lời:

Giả sử tên file được lưu như sau:

| | | |
|--------|----|------------|
| FNAME | DB | 'FILE1', 0 |
| HANDLE | DW | ? |

Chuỗi FNAME chứa tên file và phải kết thúc bằng một byte 0. Biến HANDLE sẽ chứa thẻ file.

```
MOV AX, @DATA
MOV DS, AX      ;khởi tạo DS
MOV AH, 3CH     ;hàm mở file
MOV CL, 1       ;thuộc tính chỉ đọc
LEA DX, FNAME   ;DX chứa địa chỉ tên file
INT 21H         ;mở file
MOV HANDLE, AX  ;lưu thẻ file hay mã lỗi
JC OPEN_ERROR   ;nhảy nếu có lỗi
```

Trong trường hợp có lỗi, chương trình sẽ nhảy đến nhãn OPEN_ERROR ở đó chúng ta có thể hiển thị thông báo lỗi.

Để mở một file có sẵn chúng ta dùng hàm 3Dh:

INT 21h, Hàm 3Dh:

Mở một file cũ.

| | |
|------|--|
| Vào: | AH=3Dh DS:DX=địa chỉ của chuỗi ASCII (chuỗi tên file kết thúc bằng byte 0) AL=mã truy nhập 0:mở để đọc 1:mở để ghi 2:mở để đọc và ghi |
|------|--|

| | |
|-----|--|
| Ra: | Nếu thành công, AX=thẻ file Nếu CF được đặt thì có lỗi, mã lỗi trong AX (2,4,5,12) |
|-----|--|

Sau khi làm việc với một file chúng ta phải đóng nó lại để giải phóng thẻ file cho các file khác. Khi một file đã được ghi, tất cả dữ liệu còn lại trong bộ nhớ sẽ được ghi vào file, ngày giờ tạo lập và kích thước của file sẽ được cập nhật vào thư mục.

INT 21h, Hàm 3Eh:

Đóng một file đã mở

Vào: AH=3Eh
BX=thẻ file

Ra: Nếu CF được đặt thì có lỗi, mã lỗi trong AX (6)

Ví dụ 19.2: Viết các lệnh để đóng một file, giả sử thẻ file chứa trong biến có tên HANDLE.

Trả lời:

```
MOV AH, 3EH      ;hàm đóng một file
MOV BX, HANDLE   ;Bx chứa thẻ file
INT 21H          ;đóng file
JC CLOSE_ERROR    ;nhảy nếu có lỗi
```

Lỗi duy nhất có thể xảy ra với hàm này là không có file nào tương ứng với thẻ file trong BX (mã lỗi 6 - thẻ file không hợp lệ).

19.3.4 Đọc file.

Hàm sau đây đọc một số xác định các byte từ một file và lưu nó vào bộ nhớ.

INT 21h, Hàm 3Fh:

Đọc một file

Vào: AH=3Fh
BX=thẻ file
CX=số byte cần đọc
DS:DX=địa chỉ bộ đệm

Ra: AX= số byte đọc được, nếu AX=0 hay AX< CX, file đã kết thúc

Nếu CF được đặt thì có lỗi, mã lỗi trong AX(5,6)

Ví dụ 19.3: Viết các lệnh đọc một sector 512 byte từ một file.

Trả lời: Trước tiên chúng ta phải tạo ra một vùng đệm để nhận dữ liệu.

```
.DATA
HANDLE DW ?
BUFFER DB 512 DUP(?)
MOV AX, @DATA
MOV DS, AX ;khởi tạo DS
MOV AH, 3FH ;hàm đọc file
MOV CX, 512 ;số byte cần đọc
MOV BX, HANDLE ;BX chứa thẻ file
MOV CX, 512 ;đọc 512 byte
INT 21H ;đọc file
JC READ_ERROR ;nhảy nếu có lỗi
```

Trong một số ứng dụng chúng ta có thể đọc và xử lý các sector cho đến khi kết thúc một file (gặp dấu hiệu EOF). Chương trình có thể kiểm tra EOF bằng cách so sánh AX và CX.

```
CMP AX, CX ;kết thúc file?
JL EXIT ;đúng, kết thúc chương trình
JMP READ_LOOP ;chưa, tiếp tục đọc
```

19.3.5 Ghi một file.

Hàm 40h ghi một số byte xác định ra file hay thiết bị.

INT 21h, Hàm 40h:

Ghi file

Vào: AH=40h
BX=thẻ file
CX=số byte cần ghi
DS:DX=địa chỉ của vùng đệm dữ liệu

Ra: AX số byte ghi được, nếu AX<CX, có lỗi (đĩa đầy)
Nếu CF được đặt thì có lỗi, mã lỗi trong AX (5,6)

Có một khả năng xảy ra là không còn chỗ trống trên đĩa để có thể nhận dữ liệu. DOS không coi đó là một lỗi do đó chương trình phải tự kiểm tra bằng cách so sánh AX và CX.

Hàm 40h dùng để ghi dữ liệu vào file nhưng nó cũng có thể dùng để đưa dữ liệu ra màn hình hay máy in (tương ứng các thẻ file 1 và 4).

Ví dụ 19.4:

Viết các lệnh dùng hàm 40h để hiện một thông báo lên màn hình.

Trả lời:

Chúng ta hãy giả thiết là thông báo được lưu như sau:

```
.DATA  
MSG      DB      'DISPLAY A MESSAGE'
```

Các lệnh như sau:

```
MOV      AX, @DATA  
MOV      DS, AX      ;khởi tạo DS  
MOV      AH, 40H     ;hàm ghi file  
MOV      BX, 1       ;thẻ file của màn hình  
MOV      CX, 20      ;chiều dài thông báo  
LEA      DX, MSG    ;địa chỉ của MSG  
INT      21H        ;hiển thị thông báo
```

19.3.6 Chương trình đọc và hiển thị nội dung một file.

Để minh họa hoạt động của các hàm quản lý file, chúng ta hãy viết một chương trình cho phép người sử dụng đánh vào một tên file, chương trình sẽ đọc và hiển thị nội dung file lên màn hình.

Thuật toán để hiển thị nội dung file.

```
Nhận tên file từ người sử dụng  
Mở file  
IF có lỗi mở file  
  THEN  
    Hiển thị thông báo lỗi và kết thúc  
  ELSE  
    REPEAT  
      Đọc từng sector vào vùng đệm  
      Hiển thị vùng đệm  
    UNTIL Kết thúc file  
    Đóng File  
ENDIF
```

Chương trình nguồn PGM19_1.ASM

```
0: TITLE      PGM19_1: DISPLAY FILE
1: .MODEL     SMALL
2:
3: .STACK    100H
4:
5: .DATA
6: PROMPT    DB      'FILENAME:$'
7: FILENAME   DB      30 DUP(0)
8: BUFFER     DB      512 DUP(0)
9: HANDLE     DW      ?
10: OPENERR   DB      0DH,0AH, 'OPEN ERROR - CODE'
11: ERRCODE   DB      30H, '$'
12:
13: .CODE
14: MAIN      PROC
15:           MOV AX, @DATA
16:           MOV DS, AX          ;khởi tạo DS và
17:           MOV ES, AX          ;ES
18:           CALL GET_NAME       ;đọc vào tên file
19:           LEA DX, FILENAME    ;DX chứa offset tên file
20:           MOV AL, 0            ;mã truy nhập 0: chỉ đọc
21:           CALL OPEN           ;mở file
22:           JC OPEN_ERROR       ;kết thúc nếu có lỗi
23:           MOV HANDLE, AX        ;lưu thê file
24: READ_LOOP:
25:           LEA DX, BUFFER        ;DX trả tới vùng đệm
26:           MOV BX, HANDLE        ;lấy thê file
27:           CALL READ            ;đọc file, AX = số byte đọc được
28:           OR AX, AX             ;kết thúc file?
29:           JE EXIT              ;đúng, kết thúc
30:           MOV CX, AX            ;CX chứa số byte đọc được
31:           CALL DISPLAY          ;hiển thị file
32:           JMP READ_LOOP         ;lặp lại
33: OPEN_ERROR:
34:           LEA DX, OPENERR        ;lấy thông báo lỗi
35:           ADD ERRCODE, AX        ;đổi mã lỗi thành mã ASCII
36:           MOV AH, 9              ;hiển thị thông báo lỗi
37:           INT 21H
38: EXIT:
39:           MOV BX, HANDLE        ;lấy thê file
40:           CALL CLOSE            ;đóng file
```

```

41:             MOV     AH, 4CH
42:             INT     21H
43: MAIN      ENDP
44:

45: GET_NAME PROC NEAR
46: ;đọc vào và lưu lại tên file
47: ; Vào: không
48: ; Ra: tên file được lưu như một chuỗi ASCIIIZ
49: PUSH    AX           ;lưu các thanh ghi sẽ dùng đến
50: PUSH    DX
51: PUSH    DI
52: MOV     AH, 09        ;hàm hiển thị chuỗi
53: LEA     DX, PROMPT
54: INT     21H          ;hiển thị lời nhắc
55: CLD
56: LEA     DI, FILENAME ;DI trả tới tên file
57: MOV     AH, 1           ;hàm đọc ký tự từ bàn phím
58: READ_NAME:
59:         INT     21H        ;đọc ký tự
60:         CMP     AL, 0DH   ;có phải CR?
61:         JE      DONE       ;đúng, kết thúc
62:         STOSB
63:         JMP     READ_NAME ;không, lưu nó vào trong chuỗi
64:         DONE:
65:         MOV     AL, 0           ;tiếp tục đọc vào
66:         STOSB
67:         POP     DI           ;lưu byte 0
68:         POP     DX           ;phục hồi các thanh ghi
69:         POP     AX
70:         RET
71: GET_NAME ENDP
72:

73: OPEN     PROC NEAR
74: ;mở một file
75: ;Vào: DS:DX=tên file
76: ; Al:mã truy nhập
77: ;Ra    nếu thành công, AX chứa thẻ file
78: ;      nếu không thành công, CF=1, AX chứa mã lỗi
79: MOV     AH, 3DH        ;hàm mở file
80: MOV     AL, 0           ;chỉ đọc
81: INT     21H          ;mở File
82: RET
83: OPEN     ENDP
84:

```

```

85: READ      PROC NEAR
86: ;đọc một sector file
87: ;          Vào:   BX=thẻ file
88: ;          CX=số byte cần đọc (512)
89: ;          DS:DX địa chỉ vùng đệm
90: ;          Ra    nếu thành công, sector ở trong vùng đệm
91: ;                  AX chứa số byte đọc được
92: ;                  nếu không thành công, CF=1
93:         PUSH  CX
94:         MOV   AH, 3FH .       ;hàm đọc file
95:         MOV   CX, 512        ;số byte cần đọc
96:         INT   21H
97:         POP   CX
98:         RET
99: READ     ENDP
100:
101: DISPLAY  PROC  NEAR
102: ;hiển thị bộ nhớ lên màn hình
103: ;          Vào:   BX=thẻ file (1)
104: ;                  CX=số byte cần hiển thị
105: ;                  DS:DX=địa chỉ vùng đệm dữ liệu
106: ;          Ra:    AX=số byte đã hiển thị
107:         PUSH  BX
108:         MOV   AH, 40H       ;hàm ghi File
109:         MOV   BX, 1         ;thẻ File cho màn hình
110:         INT   21H         ;đóng File
111:         POP   BX
112:         RET
113: DISPLAY ENDP
114:
115: CLOSE    PROC  NEAR
116: ;đóng một file
117: ;          Vào:   BX=thẻ file
118: ;          Ra:    Nếu CF=1, mã lỗi trong AX
119:         MOV   AH, 3EH        ;hàm đóng file
120:         INT   21H         ;đóng File
121:         RET
122: CLOSE   ENDP
123:
124: END      MAIN

```

Tại dòng 18, thủ tục GET_NAME được gọi để nhập tên file do người sử dụng đánh vào và lưu nó trong biến FILENAME như một chuỗi ASCII. Sau khi địa

chỉ offset của FILENAME được chuyển vào DX, thủ tục OPEN được gọi ở dòng 21 để mở file. Lỗi có nhiều khả năng xảy ra nhất là đường dẫn hoặc tên file không tồn tại. Trong cả 2 trường hợp OPEN đều đặt cờ CF và mã lỗi 2 hay 3 trong AL. Chương trình đổi số này ra mã ASCII bằng cách cộng nó với 30h trong biến ERRCODE (dòng 35) và hiển thị thông báo lỗi tương ứng với mã lỗi. Chú ý rằng đánh nhầm cũng bị coi là lỗi.

Nếu file được mở tốt đẹp, AX sẽ chứa 5, thẻ file săn có đầu tiên sau 5 thẻ file đã định nghĩa trước.

Tại dòng 24, chương trình vào vòng lặp chính. Trước tiên thủ tục được gọi READ sẽ đọc một sector vào vùng đệm BUFFER. CF được đặt nếu có lỗi, nhưng các lỗi có thể với hàm này (từ chối truy nhập hay thẻ file không hợp lệ) lại không thể xảy ra trong chương trình của chúng ta, do đó AX sẽ chứa số byte thực tế đọc được. Nếu số này bằng 0 nghĩa là lần đọc trước đã đọc hết dữ liệu trong file và chương trình gọi thủ tục CLOSE để đóng file.

Nếu AX khác 0, số byte đọc được được chuyển vào CX(dòng 30) và thủ tục DISPLAY được gọi để hiển thị các byte lên màn hình.

Ví dụ thực hiện:

```
C>PGM19_1
FILENAME: A:\A.TXT
THIS IS A SMALL TEST FILE

C>PGM19_1
FILENAME: A:\B.TXT
OPEN ERROR - CODE 2 (NONEXISTENT FILE)

C>PGM19_1
FILENAME: A:\PROGS\A.TXT
OPEN ERROR - CODE 3 (ILLEGAL PATH)
```

19.3.7 Con trỏ file (file pointer).

Con trỏ file được dùng để định vị trong file. Mỗi khi file được mở, con trỏ file được đặt ở đầu file. Sau một thao tác đọc, con trỏ file chỉ đến byte tiếp theo sẽ được đọc. Sau khi ghi một file mới con trỏ file chỉ đến cuối file (EOF). Để di chuyển con trỏ file chúng ta có thể sử dụng hàm sau:

INT 21h, Hàm 42h:**Di chuyển con trỏ file**

Vào: AH=42h

AL=mã dịch chuyển: 0 dịch chuyển tương đối với đầu file
 1 dịch chuyển tương đối với vị trí hiện thời của
 con trỏ

2 dịch chuyển tương đối với cuối file.

BX=thẻ file

CX:DX=số byte dịch chuyển (có dấu)

Ra: DX:AX vị trí mới của con trỏ file tính bằng byte từ đầu file
 Nếu CF =1, mã lỗi ở trong AX (1,6)

CX:DX chứa số byte để di chuyển con trỏ, được biểu diễn như một số có dấu (dương nghĩa là chuyển về cuối file, âm nghĩa là chuyển về đầu file). Nếu AL=0, sự dịch chuyển tính từ đầu file, AL=1 sự dịch chuyển tính từ điểm hiện thời và AL=2 sự dịch chuyển tính từ cuối file.

Nếu CX:DX quá lớn con trỏ file có thể di chuyển ra ngoài điểm cuối cùng của file. Bản thân nó không phải là một lỗi nhưng nó sẽ gây ra lỗi cho các công việc đọc hay ghi file tiếp theo.

Đoạn lệnh sau đây chuyển con trỏ file đến cuối file và xác định kích thước của file.

```

MOV     AH, 42H      ;hàm chuyển con trỏ file
MOV     BX, HANDLE   ;BX chứa thẻ file
XOR     CX, CX
XOR     DX, DX      ;dịch chuyển 0 byte
MOV     AL, 2         ;tính từ cuối file
INT     21H          ;chuyển con trỏ tới cuối file
                  ;DX:AX = kích thước file
JC      MOVE_ERROR  ;có lỗi nếu CF=1.

```

Ứng dụng :Thêm vào file một bản ghi

Chương trình sau đây tạo ra các file chứa tên người. Nó thông báo cho người sử dụng đánh vào các tên dài tối đa 20 ký tự. Mỗi khi có một tên được đánh vào, chương trình thêm nó vào file và xoá dòng nhập liệu trên màn hình. Người sử dụng chỉ định kết thúc nhập liệu bằng cách ấn CTRL_Z.

Thuật toán cho chương trình chính

```
Open file NAMES
Chuyển con trỏ file tới cuối file
Hiển thị dấu nhắc nhập số liệu
WHILE <CTRL_Z> chưa đánh vào DO
    nhập tên từ người sử dụng và lưu nó trong mảng byte
    NAMFLD
    viết tên vào file NAMES
ENDWHILE
Close file NAMES .
```

Chương trình chính sẽ gọi thủ tục GET_NAME để nhập một tên từ bàn phím.

Thuật toán cho thủ tục GET_NAME.

```
Đặt các dấu trống vào 20 byte đầu tiên của NAMEFLD
(2 byte cuối cùng là CR, LF)
REPEAT
    đọc một ký tự
    IF ký tự là CTRL_Z (1Ah)
    THEN
        Đặt CF và kết thúc
    ELSE
        IF ký tự không là <CR>
        THEN
            lưu ký tự vào NAMEFLD
        EDNIF
    ENDIF
UNTIL ký tự là <CR>
Xoá dòng nhập liệu trên màn hình.
```

Chương trình nguồn PGM19_2.ASM

```
0: TITLE PGM19_2: APPEND RECORD
1: .MODEL SMALL
2:
3: .STACK 100H
4:
5: .DATA
6: PROMPT DB      'NAMES:', 0DH, 0AH, '$'
7: NAMEFLD DB     20 DUP('0'), 0DH, 0AH
8: FILE    DB     'NAMES', 0
```

```

9:     HANDLE DW      0
10:    OPENERR DB      ODH,0AH,'OPEN ERROR $'
11:    WRITERR DB      ODH,0AH,'WRITE ERROR $'
12:
13:    .CODE
14:    MAIN    PROC
15:        MOV     AX, @DATA
16:        MOV     DS,AX      ;khởi tạo DS
17:        MOV     ES,AX      ;và ES
18:    ;mở file NAMES
19:        LEA     DX,FILE      ;lấy địa chỉ tên file
20:        CALL    OPEN         ;mở file
21:        JC     OPEN_ERROR   ;thoát nếu có lỗi
22:        MOV     HANDLE,AX    ;lưu thê file
23:    ;chuyển con trỏ file đến cuối file
24:        MOV     BX,HANDLE    ;lấy thê file
25:        CALL    MOVE_PTR     ;chuyển con trỏ
26:    ;in ra màn hình dấu nháć
27:        MOV     AH,09        ;hàm hiển thị chuỗi
28:        LEA     DX,PROMPT    ;"NAMES:"
29:        INT     21H          ;hiển thị dòng nháć
30:    READ_LOOP:           ;đọc vào các tên
31:        LEA     DI,NAMEFLD   ;DI trả tới tên
32:        CALL    GET_NAME     ;đọc vào tên
33:        JC     EXIT         ;CF=1 nếu kết thúc nhập liệu
34:    ;thêm tên vào file
35:        MOV     BX,HANDLE    ;lấy thê file
36:        MOV     CX,22        ;20 byte tên và 2 byte CR, LF
37:        LEA     DX,NAMEFLD   ;lấy địa chỉ vùng đệm tên
38:        CALL    WRITE        ;ghi vào file
39:        JC     WRITE_ERROR   ;kết thúc nếu có lỗi
40:        JMP     READ_LOOP    ;đọc tên tiếp theo
41:    OPEN_ERROR:          ;lấy thông báo lỗi
42:        LEA     DX,OPENERR    ;lấy thông báo lỗi
43:        MOV     AH,09
44:        INT     21H          ;hiển thị thông báo lỗi
45:        JMP     EXIT
46:    WRITE_ERROR:          ;lấy thông báo lỗi
47:        LEA     DX,WRITERR    ;lấy thông báo lỗi
48:        MOV     AH,09
49:        INT     21H          ;hiển thị thông báo
50:    EXIT:               ;lấy thê file
51:        MOV     BX,HANDLE    ;đóng file NAMES
52:

```

```

53:           MOV      AH, 4CH
54:           INT      21H          ;về DOS
55: MAIN     ENDP
56:
57: GET_NAME    PROC NEAR
58: ;đọc vào và lưu lại một tên
59: ;      Vào:    DI = offset của NAMEFLD
60: ;      Ra:     NAMEFLD chứa tên
61:           CLD
62:           MOV      AH, 1        ;hàm đọc ký tự từ bàn phím
63: ;xoá NAMEFLD
64:           PUSH     DI          ;lưu con trỏ tới NAMEFLD
65:           MOV      CX, 20      ;tên có thể có tới 20 ký tự
66:           MOV      AL, ' '
67:           REP      STOSB      ;lưu các khoảng trắng
68:           POP      DI          ;phục hồi con trỏ tới NAMEFLD
69: READ_NAME:
70:           INT      21H          ;đọc một ký tự
71:           CMP      AL, 1AH      ;kết thúc nhập liệu?
72:           JNE      NO          ;không, tiếp tục
73:           STC
74:           RET          ;đúng, thiết lập cờ CF
                           ;và trả về
75: NO:
76:           CMP      AL, 0DH      ;nhập xong tên?
77:           JE      DONE         ;đúng, kết thúc
78:           STOSB
79:           JMP      READ_NAME   ;chưa, lưu nó vào chuỗi
                           ;tiếp tục đọc vào
80: ;xoá dòng nhập liệu
81: DONE:
82:           MOV      AH, 2        ;hàm đưa ra màn hình ký tự
83:           MOV      DL, 0DH
84:           INT      21H          ;về đầu dòng
85:           MOV      DL, ' '
86:           MOV      CX, 20
87: CLEAR:
88:           INT      21H
89:           LOOP    CLEAR        ;xoá dòng nhập liệu
90:           MOV      DL, 0DH      ;về đầu dòng
91:           INT      21H
92:           RET
93: GET_NAME  ENDP
94:
95: OPEN     PROC    NEAR
96: ;mở một file

```

```

97: ;          Vào:    DS:DX=tên file
98: ;                      AL:mã truy nhập
99: ;          Ra      nếu thành công, AX chứa thẻ file
100: ;         nếu không thành công, CF=1, AX chứa mã lỗi
101:     MOV     AH, 3DH      ;hàm mở file
102:     MOV     AL, 1        ;chỉ ghi
103:     INT     21H        ;mở file
104:     RET
105: OPEN   ENDP
106:
107: WRITE  PROC NEAR
108: ;ghi vào một file
109: ;          Vào:    BX=thẻ file
110: ;                      CX=số byte cần ghi
111: ;                      DS:DX=địa chỉ vùng đệm dữ liệu
112: ;          Ra:      AX=số byte ghi được
113: ;         nếu không thành công, CF=1, AX=mã lỗi
114:     MOV     AH, 40H      ;hàm ghi file
115:     INT     21H        ;ghi file
116:     RET
117: WRITE  ENDP
118:
119: CLOSE  PROC NEAR
120: ;đóng một file
121: ;          Vào:    BX=thẻ file
122: ;          Ra:      Nếu CF=1, mã lỗi trong AX
123:     MOV     AH, 3EH      ;hàm đóng file
124:     INT     21H        ;đóng file
125:     RET
126: CLOSE  ENDP
127:
128: MOVE_PTR PROC NEAR
129: ;chuyển con trỏ tới cuối file
130: ;          Vào:    BX=thẻ file
131: ;          Ra:      DX:AX= vị trí con trỏ tính từ đầu file
132:     MOV     AH, 42H      ;hàm chuyển con trỏ file
133:     XOR     CX,CX       ;chuyển 0 byte
134:     XOR     DX,DX       ;tính từ cuối file
135:     MOV     AL, 2        ;mã chuyển
136:     INT     21H
137:     RET
138: MOVE_PTR ENDP
139:
140: END     MAIN

```

Chương trình bắt đầu bằng việc sử dụng hàm 3Dh ngắt 21h để mở file NAMES. Vì hàm này chỉ được dùng để mở các file đã có sẵn, một file rỗng NAMES phải được tạo lập trước khi chương trình chạy lần đầu tiên. Để tạo ra file như vậy chúng ta sử dụng trình DEBUG với các bước như sau:

1. Sử dụng lệnh N để đặt tên file (đánh vào N NAMES)
2. Đặt 0 vào BX và CX (xác định kích thước dài file)
3. Ghi file vào đĩa (Đánh W)

Sau khi chạy chương trình có thể dùng lệnh TYPE của DOS để xem file.

Ví dụ thực hiện: Chú ý! Các tên đánh vào thực tế sẽ xuất hiện trên cùng một dòng, nhưng để tiện theo dõi chúng tôi trình bày trên các dòng riêng biệt.

```
C>PGM19_2
NAMES:
GEORGE WASHINGTON
JOHN ADAMS
<CTRL_Z>
C>TYPE NAMES
GEORGE WASHINGTON
JOHN ADAMS
C>PGM19_2
NAMES:
THOMAS JEFERSON
HARRY TRUMAN
BIN CLINTON
<CTRL_Z>
C>TYPE NAMES
GEORGE WASHINGTON
JOHN ADAMS
THOMAS JEFERSON
HARRY TRUMAN
BIN CLINTON
```

19.3.8 Thay đổi thuộc tính file.

Trong phần 19.1.2 chúng ta đã thấy rằng mỗi khi file được tạo ra (hàm 3Ch) thì thuộc tính của nó được xác lập. Các hàm sau đây dùng để thay đổi thuộc tính file.

INT 21h, Hàm 43h:

Lấy/Thay đổi thuộc tính file

Vào: AH=43h
DS:DX=địa chỉ của chuỗi ASCII
(chuỗi tên file kết thúc bằng byte 0)
AL=0 để lấy thuộc tính file
AL=1 để thay đổi thuộc tính file
CX= thuộc tính file mới (nếu AL=1)

Ra: Nếu thành công, CX=thuộc tính hiện thời(khi AL=0)
Nếu CF được đặt thì có lỗi, mã lỗi trong AX (2,3 hay 5)

Chúng ta cũng có thể dùng hàm này để thay đổi các bit nhãn đĩa hay thư mục con trong byte thuộc tính (bits 3 và 4).

Ví dụ 19.5: Thay đổi thuộc tính một file thành ẩn.

Trả lời:

```
MOV AH, 43H ;hàm lấy/thay đổi thuộc tính file
MOV AL, 1    ;tuỳ chọn thay đổi thuộc tính
LEA DX, FILENAME ;lấy đường dẫn
MOV CX, 1    ;thuộc tính ẩn
INT 21H      ;đổi thuộc tính
JC ATTR_ERROR ;thoát nếu có lỗi, mã lỗi
                ;trong AX.
```

19.4 Các thao tác trực tiếp đĩa

Cho đến nay chúng ta đã nói về các thao tác file sử dụng các hàm quản lý file của ngắt 21h. Có 2 ngắt khác của DOS cho phép đọc và ghi trực tiếp các sector của đĩa.

19.4.1 Ngắt 25h và 26h.

Các ngắt của DOS để đọc và ghi trực tiếp các sector đĩa tương ứng là ngắt 25h và 26h. Trước khi gọi các ngắt này chúng ta phải khởi tạo các thanh ghi sau đây:

| | | |
|-------|---|---|
| AL | = | số hiệu ổ đĩa (0=ổ A, 1=ổ B và cứ như vậy) |
| DS:BX | = | địa chỉ segment:offset của vùng đệm bộ nhớ |
| CX | = | số sector muốn đọc/ghi |
| DX | = | số hiệu sector logic đầu tiên(xem phần sau) |

Khác với ngắt 21h, không có số hàm nào phải đặt vào trong AH. Thường trình phục vụ ngắt đặt thanh ghi cờ FLAGS vào ngăn xếp và nó sẽ được đưa ra khi tiếp tục chương trình. Nếu CF=1 thì đã có lỗi xảy ra và AX chứa mã lỗi.

| Mặt đĩa | Track | Sector | Sector Logic | Thông tin |
|---------|-------|--------|----------------|------------------------|
| 0 | 0 | 1 | 0 | Boot record |
| 0 | 0 | 2-5 | 1-4 | FAT |
| 0 | 0 | 6-9 | 5-8 | Thư mục file |
| 1 | 0 | 1-3 | 9-11(9h-Bh) | Thư mục file |
| 1 | 0 | 4-9 | 12-17(Ch-11h) | Dữ liệu (theo nhu cầu) |
| 0 | 1 | 1-9 | 18-26(12h-1Ah) | Dữ liệu (theo nhu cầu) |

Bảng 19.3: Các sector logic

Số hiệu sector logic

Trong phần 19.2 chúng ta xác định một vị trí trên đĩa bằng số hiệu mặt, track và sector. DOS gán một số hiệu logic cho mỗi sector, bắt đầu bằng sector 0. Số hiệu logic của các sector tăng dần trong track ở mặt 0 và tiếp theo trong track có cùng số hiệu ở mặt 1. Bảng 19.3 cho thấy tương quan giữa số hiệu sector logic và tọa độ mặt-track-sector trong phần đầu của đĩa mềm 5^{1/4} inch.

Đọc một sector.

Để làm ví dụ cho các thao tác đĩa trực tiếp, chương trình sau đây đọc sector đầu tiên của thư mục(sector logic 5) ổ đĩa A.

Chương trình nguồn PGM19_3.ASM

```
0. TITLE      PGM19_3:      READ SECTOR
1. .MODEL     SMALL
2.
3. .STACK     100H
4.
5. .DATA
6. BUFF        DB      512 DUP(0)
7. ERROR_MSG   DB      'ERROR!$'
8.
9. .CODE
10.MAIN      PROC
11.          MOV      AX, @DATA
12.          MOV      DS, AX ; khởi tạo DS
13.          MOV      AL, 0  ; ổ đĩa A
14.          LEA      BX, BUFF ; địa chỉ vùng đệm
15.          MOV      CX, 1  ; đọc 1 sector
16.          MOV      DX, 5  ; bắt đầu từ sector 5
17.          INT      25H    ; đọc sector
18.          POP     DX     ; phục hồi ngăn xếp
19.          JNC      EXIT   ; nhảy nếu không có lỗi
20. ;nếu có lỗi
21.          MOV      AH, 09 ; hàm hiển thị chuỗi
22.          LEA      DX, ERROR_MSG; hiển thị chuỗi
23.          INT      21H
24.EXIT:
25.          MOV      AH, 4CH ; về DOS
26.          INT      21H
27.MAIN      ENDP
28.
29.          END      MAIN
```

Để minh họa, chúng ta đặt đĩa chứa các file A.TXT và B>TXT vào ổ đĩa A và chạy chương trình trong DEBUG. Trong môi trường này chương trình thực hiện chức năng giống lệnh L (load) của DEBUG.

C> DEBUG PGM19_3.EXE

-G13 (thực hiện cho đến dòng 18 ở trên)
AX=0100 BX=0000 CX=0000 DX=0005 SP=0062 BP=7420 SI=01B6
DI=0001
DS=0F12 ES=0EFB SS=0F0B CS=0F33 IP=0013 NV UP EI ZR NA PE NC
0F33:0013 5A POP DX

-D0 (hiển thị vùng đệm)

0F12:0000 41 20 20 20 20 20 20-54 58 54 20 00 00 00 00 A TXT

```

CF12:0010 00 0C 00 00 00 00 BD 19-22 16 02 00 80 00 00 00 .....=.".....
OF12:0020 42 20 20 20 20 20 20 20-54 54 58 20 00 00 00 00 B .....TXT .....
OF12:0030 0C 00 00 00 00 00 23 24_8A 16 03 00 80 00 00 00 .....#S.....
OF12:0040 00 F6 F6 F6 F6 F6 F6-F6 F6 F6 F6 F6 F6 F6 F6 .vvvvvvvvvvvvvvvvv
OF12:0050 F6 F6 F6 F6 F6 F6 F6-F6 F6 F6 F6 F6 F6 F6 F6 vvvvvvvvvvvvvvvvvv
OF12:0060 0C F6 F6 F6 F6 F6 F6-F6 F6 F6 F6 F6 F6 F6 F6 .vvvvvvvvvvvvvvvvv
OF12:0070 F6 F6 F6 F6 F6 F6 F6-F6 F6 F6 F6 F6 F6 F6 F6 vvvvvvvvvvvvvvvvvv

```

Từ kết quả hiển thị chúng ta có thể lấy ra những trường tương ứng của điểm nhập ứng với file A.TXT trong thư mục.

| OFFSET | THÔNG TIN | BYTE |
|--------|--------------------------|---------------------------|
| 0-7 | Tên file | 41 20 20 20 20 20 20 20 A |
| 8-A | Phần mở rộng | 54 58 54 TXT |
| B | Thuộc tính | 20 |
| C-15 | Để giành | |
| 16-17 | Giờ tạo lập | BD 19 |
| 18-19 | Ngày tạo lập | 22 16 |
| 1A-1B | Số hiệu cluster đầu tiên | 02 |
| 1C-1D | Kích thước file | 80 |

Dạng của giờ tạo lập giờ:phút:giây là :hhhhmmmmmmssss. Đối với file này ta có:

$$19Bdh = 00011\ 001101\ 11101 \\ = 3:13:29$$

Dạng của ngày tạo lập năm:tháng:ngày là yyyyymmddddd trong đó số hiệu năm được tính tương đối từ năm 1980 đối với DOS version này. Chúng ta nhận được:

$$1622h=0001011\ 0001\ 00010 = 11:1:2 \text{ (thực tế là 91:1:2)}$$

Kiểm tra bảng FAT

Để làm ví dụ thứ 2 chúng ta có thể đặt đĩa chứa vài file vào ổ và hiển thị phần đầu của bảng FAT bắt đầu từ sector logic 1. Nếu chúng ta thay đổi dòng 17 trong chương trình và chạy thì kết quả sẽ như sau:

-D0

```
0F12:0000 FD FF FF FF 4F 00 05 60-00 07 F0 FF 09 A0 00 0B  
0F12:0010 C0 00 0D E0 00 0F 00 01-11 20 01 13 40 01 15 60  
0F12:0020 01 17 80 01 19 A0 01 1B-C0 01 1D E0 01 1F 00 02  
0F12:0030 21 20 02 23 40 02 25 60-02 27 80 02 29 A0 02 2B  
0F12:0040 C0 02 2D E0 02 2F 00 03-31 20 03 33 40 03 35 60  
0F12:0050 03 37 80 03 39 A0 03 3B-C0 03 3D E0 03 3F 00 04  
0F12:0060 41 20 04 43 40 04 45 60-04 47 80 04 49 A0 04 4B  
0F12:0070 C0 04 4D E0 04 4F 00 05-51 20 05 53 40 05 55 60
```

Rất khó đọc bảng FAT ở dạng này vì các điểm nhập của FAT dài 12 bit = 3 chữ số hex, để giải mã, chúng ta phải tạo ra một cặp điểm nhập (2 số hex có 3 chữ số) bằng cách lần lượt (1) lấy 2 chữ số hex từ một byte và chữ số bên phải nhất của byte tiếp theo và (2) lấy chữ số còn lại của byte đó (chữ số bên trái nhất) với 2 chữ số của byte tiếp theo.

Thực hiện các thao tác này với kết quả cho ở trên chúng ta nhận được kết quả như sau:

| cluster | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | ... |
|----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| nội dung | FFD | FFF | FFF | C04 | 005 | 006 | 007 | FFF | 009 | 00A | 00B | ... |

Dữ liệu bắt đầu từ cluster 2. Điểm nhập tại đó là FFF do đó file cũng kết thúc tại cluster này. File tiếp theo bắt đầu tại cluster 3 và kết thúc tại cluster 7. File tiếp theo nữa bắt đầu tại cluster 8 và cứ như vậy.

TỔNG KẾT

Trong chương này chúng ta đã nghiên cứu những vấn đề sau:

- ◆ Chương trình FORMAT phân chia mỗi mặt của đĩa thành những vòng tròn đồng tâm gọi là các rãnh (track). Mỗi rãnh lại được chia nhỏ thành các cung (sector) chứa 512 byte. Số các rãnh và cung tùy thuộc vào từng loại đĩa. Đĩa mềm 51/4 2 mặt mật độ kép có 40 track mỗi mặt và 9 sector trên mỗi track.
- ◆ Khi lưu dữ liệu DOS điền đầy vào một rãnh ở mặt này rồi mới tiếp tục ở rãnh của mặt kia.
- ◆ Thông tin về các file được lưu trong thư mục. Một điểm nhập của thư mục bao gồm tên, phần mở rộng, thuộc tính, ngày, giờ, liên cung (cluster) đầu tiên và kích thước của file.
- ◆ Thuộc tính của tệp được gán khi chúng ta tạo lập nó. Thuộc tính tệp xác định tệp có phải là tệp chỉ đọc, ẩn, hệ thống, nhãn đĩa, thư mục con hay đã được sửa hay chưa. Thuộc tính thông thường của một tệp là 20h.
- ◆ DOS giành vùng nhớ cho các file thông qua các cluster. Một cluster chứa một số xác định các sector tuỳ thuộc vào loại đĩa (2 cho các đĩa mềm). Dữ liệu được ghi vào đĩa bắt đầu từ cluster 2.
- ◆ Bảng FAT (File Allocation Table) cho ta một bản đồ các file lưu trong đĩa. Mỗi điểm nhập của FAT dài 12 bit. Trong mỗi điểm nhập của thư mục có chứa số hiệu cluster đầu tiên N1 của tập tin. Điểm nhập N1 của FAT chứa số hiệu N2 của cluster tiếp theo của tập tin và cứ như vậy, điểm nhập cuối cùng của tập tin trong FAT chứa giá trị FFFh.
- ◆ Các hàm quản lý file của ngắt 21h cho ta một phương pháp tiện dụng để thực hiện các thao tác file. Với phương pháp này, các tập tin được gán một số gọi là thẻ file khi nó được mở và chương trình có thể làm việc với file thông qua con số này.
- ◆ Các hàm quản lý file được gọi bằng cách đặt số hiệu hàm vào AH và gọi ngắt 21h. Các hàm cơ bản là 3Ch để mở một tập tin mới có, 3Dh để mở một tập tin có sẵn, 3Eh để đóng một tập tin, 3F để đọc một tập tin, 40h để ghi một tập tin, 42h để chuyển con trỏ file và 43h để thay đổi thuộc tính tập tin.
- ◆ Các ngắt 25h và 26h của DOS có thể dùng để đọc và ghi các sector đĩa.

Các thuật ngữ tiếng Anh

| | |
|-----------------------------------|--|
| Archive bit | Bit lưu - dùng để chỉ ra lần cập nhật tập tin gần đây nhất. |
| Attribute byte | Byte thuộc tính - xác định thuộc tính của file. |
| Cluster | Một số xác định các sector tuỳ thuộc vào từng loại đĩa. |
| Cylinder | Các track có cùng số hiệu nhưng ở các mặt đĩa khác nhau. |
| Data Transfer Area(DTA) | Vùng chuyển dữ liệu - vùng bộ nhớ DOS dùng để lưu trữ dữ liệu từ file. |
| File Allocation Table(FAT) | Bảng phân bổ file - bảng đồ sự phân vùng các file trong bộ nhớ. |
| File handle | Thẻ file - một con số sẽ được INT 21h sử dụng để xác định một file. |
| File pointer | Con trỏ file - dùng để định vị trong file. |
| Hidden file | File ẩn - file mà tên của nó không hiện trong lệnh kiểm tra thư mục đĩa (DIR). |
| Read a file | Đọc file - chép toàn bộ hay một phần của file ra bộ nhớ. |
| Rewrite a file | Ghi lại file - thay nội dung của file bằng dữ liệu mới. |
| Sector | Một vùng 512 byte trong một track. |
| Status byte | Byte trạng thái - byte 0 trong điểm nhập của thư mục. |
| Track | Một vùng nhớ hình tròn trên đĩa. |
| Write a file | Ghi file - chép dữ liệu từ bộ nhớ vào file. |

Bài tập

- Hãy kiểm chứng lại rằng đĩa mềm 5 ½ có 80 cylinder và 15 sector trên một track có dung lượng là 1.228.800 byte.
- Giả sử các điểm nhập của FAT là số dài 12bit = 3 chữ số hex. Hơn nữa giả sử rằng đĩa chứa 3 tập tin là FILE1, FILE2, FILE3 và FAT bắt đầu như sau:

| Điểm nhập | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| cluster | 004 | 009 | 00B | FFF | 00A | FFF | FFF | 000 | 000 | 000 | 000 | 000 | 000 | 000 |

- a. Nếu FILE1, FILE2, FILE3 tương ứng bắt đầu tại các cluster 2, 3 và 7 hãy cho biết mỗi file chiếm những cluster nào.
- b. Mỗi khi một file bị xoá, các điểm nhập của nó trong FAT bị đặt thành 000. Hãy cho biết nội dung của FAT sau những trường hợp sau (giả thiết các thao tác thực hiện theo đúng trình tự của câu hỏi)
 - Xoá FILE1.
 - Tạo ra FILE4 có độ dài 1500 byte.
 - Xoá FILE2.
 - Tạo ra FILE5 có độ dài 500 byte.
 - Tạo ra FILE6 có độ dài 1500 byte.
- Viết các lệnh thực hiện các thao tác sau. Giả sử thẻ file chứa trong biến WORD có tên HANDLE.
 - Chuyển con trỏ file 100 byte tính từ đầu file.
 - Chuyển con trỏ file ngược lại 1 byte tính từ vị trí hiện thời.
 - Đưa vị trí hiện thời của con trỏ file vào DX:AX
- Từ kết quả hiển thị thư mục file trong chương trình DEBUG ở phần 19.4.1, xác định ngày giờ tạo lập và kích thước của file B.TXT.

Bài tập lập trình

- Viết một chương trình chép một file nguồn đến một file đích, thay tất cả các chữ thường bằng chữ hoa. Dùng lệnh TYPE của DOS để hiển thị file nguồn và file đích.
- Viết một chương trình đọc 2 file và hiển thị chúng bên cạnh nhau trên màn hình. Bạn có thể giả sử chiều dài dòng của mỗi file nhỏ hơn nửa chiều rộng màn hình.

7. Hãy sửa chương trình PGM19_2.ASM trong phần 19.3.7 để nó thông báo cho người sử dụng đánh vào một tên, nó sẽ xác định xem tên đó có trong file NAMES không, nếu có sẽ đưa ra vị trí của nó ở dạng hex.
8. Sửa chương trình PGM19_2.ASM trong phần 19.3.7 để nó thông báo cho người sử dụng đánh vào một tên. Nếu tên đó có mặt trong NAMES, chương trình sẽ tạo ra một bản sao của NAMES không có tên đó. Sử dụng lệnh TYPE của DOS để hiển thị file nguồn và file đã thay đổi.

Chương 20

CÁC BỘ VI XỬ LÝ TIÊN TIẾN CỦA INTEL

Tổng quan

Cho đến nay chúng ta vẫn tập trung vào các bộ vi xử lý 8086/8088. Trong chương này chúng ta sẽ xem xét một số bộ vi xử lý tiên tiến của INTEL. Những bộ vi xử lý này ngày nay đã trở lên vô cùng quen thuộc. Các bạn sẽ thấy chúng cũng tương thích với 8086 và có thể thực hiện chương trình viết cho 8086. Ngoài ra chúng còn có các đặc tính riêng nhờ đó có khả năng cung cấp chế độ bộ nhớ bảo vệ và làm việc đa nhiệm.

Trong phần 20.1 chúng ta sẽ nói về 80286. Hệ điều hành cần thiết để sử dụng chế độ bộ nhớ bảo vệ của 80286 được trình bày trong phần 20.2. Trong phần 20.3 chúng ta sẽ xem xét các bộ vi xử lý 80386 và 80486.

20.1 Bộ vi xử lý 80286

Giống như 8086, 80286 cũng là một bộ vi xử lý 16 bit. Nó có tất cả các thanh ghi của 8086 và có thể thực hiện tất cả các lệnh của 8086. Nó được thiết kế để đảm bảo tính tương thích với 8086 đồng thời có khả năng làm việc đa nhiệm. Để có thể làm được điều đó, 80286 phải có 2 chế độ làm việc: chế độ địa chỉ thực (**real address mode**) còn gọi tắt là chế độ thực (**real mode**) và chế độ địa chỉ ảo được bảo vệ (**protected virtual address mode**) còn gọi tắt là chế độ bảo vệ (**protected mode**).

Trong chế độ địa chỉ thực, 80286 có thể xem như 8086 và có khả năng thực hiện mọi chương trình viết cho 8086 không cần phải có điều chỉnh gì. Ngoài tập lệnh của 8086 nó còn có khả năng thực hiện một số lệnh khác nữa gọi là tập lệnh mở rộng (**extended instruction set**).

Trong chế độ bảo vệ, 80286 cung cấp khả năng làm việc đa nhiệm và có thể thực hiện một số lệnh cần thiết cho mục đích này. Ngoài ra cũng có thêm một số thanh ghi khác được sử dụng trong chế độ này.

Chúng ta hãy bắt đầu với tập lệnh mở rộng.

20.1.1 Tập lệnh mở rộng

Tập lệnh mở rộng có chứa một số lệnh của 8086 với các kiểu toán hạng khác và một số lệnh mới. Tập lệnh mở rộng có thể được chia thành các nhóm lệnh nạp và lấy dữ liệu từ ngăn xếp, nhân, quay, dịch, vào ra theo chuỗi và các lệnh bậc cao.

PUSH và POP

80286 cho phép thực hiện lệnh PUSH với toán hạng là hằng số. Dạng lệnh như sau:

PUSH immediate

Với lệnh này bây giờ chúng ta không phải đưa một hằng số vào một thanh ghi rồi đưa thanh ghi vào ngăn xếp như trước nữa, ví dụ chúng ta có thể viết PUSH 25 thay vì 2 lệnh:

MOV AX, 25 và PUSH AX.

Ngoài ra còn có lệnh nạp và lấy tất cả các thanh ghi công dụng chung. Lệnh PUSHA (push all) nạp tất cả các thanh ghi công dụng chung vào ngăn xếp theo trình tự sau: AX, CX, DX, BX, SP, BP, SI và DI. Lệnh POPA (pop all) lấy ra khỏi ngăn xếp tất cả các thanh ghi công dụng chung theo trình tự sau: DI, SI, BP, SP, BX, DX, CX, AX. Các lệnh này đặc biệt có ý nghĩa khi một chương trình cần lưu tất cả các thanh ghi. Cú pháp của lệnh như sau:

PUSHA
POPA

Lệnh nhân (MULTIPLY)

80286 có 3 dạng mới của lệnh IMUL cho phép có nhiều toán hạng:

IMUL reg16, immed

```
IMUL    reg16,reg16,immed  
IMUL    reg16,mem16,immed
```

Trong đó immed là hằng số, reg16 là thanh ghi 16 bit và mem16 là một từ nhớ. Dạng lệnh thứ nhất định nghĩa hằng số immed là toán hạng nguồn và thanh ghi công dụng chung 16 bit là toán hạng đích. Dạng lệnh thứ 2 và thứ 3 có 3 toán hạng, toán hạng thứ nhất là một thanh ghi 16 bit - nơi chứa tích, toán hạng thứ 2 và thứ 3 chứa số nhân và số bị nhân. Dưới đây là các ví dụ:

1. IMUL BX, 20 ; BX được nhân với 20,
; tích số chứa trong BX
2. IMUL AX, BX, 20 ; BX và 20 nhân với nhau,
; tích số chứa trong AX
3. IMUL AX, WDATA, 20 ; Nhân WDATA với 20 và
; chứa tích trong AX.

Chú ý rằng chỉ có 16 bit thấp của tích là được lưu, CF và OF sẽ xoá nếu tích có thể lưu được như một số có dấu 16 bit. Ngược lại chúng sẽ được đặt, các cờ khác là không xác định.

Các lệnh quay và dịch (Rotate & Shift).

80286 thực hiện nhiều lệnh quay và dịch cùng lúc sử dụng bộ đếm là một hằng số kiểu byte. Do đó sẽ không cần dùng thanh ghi CL. Chẳng hạn chúng ta có thể viết SHR AX, 4 thay vì 2 lệnh:

```
MOV     CL, 4  và     SHR   AX, CL.
```

Các lệnh vào ra theo chuỗi (STRING I/O).

80286 cho phép các thao tác vào ra thực hiện được với cùng lúc nhiều byte. Lệnh vào có dạng INSB (input string byte) và INSW (input string word). Lệnh INSB (hay INSW) chuyển một byte (hay một word) từ cổng có địa chỉ trong DX vào ô nhớ được định địa chỉ bởi ES:DI. Sau đó DI sẽ được tăng hay giảm tùy theo trạng thái cờ DF cũng giống như các thao tác với chuỗi. Tiên tố REP có thể dùng để nhập vào nhiều byte hay word một lúc.

Các lệnh ra là OUTSB (out string byte) và OUTSW (out string word). Lệnh OUTSB (hay OUTSW) chuyển một byte hay một word từ ô nhớ có địa chỉ ES:SI ra cổng có địa chỉ trong DX, sau đó SI sẽ được tăng hay giảm tùy theo trạng thái của cờ DF giống như các thao tác chuỗi. Một lần nữa tiên tố REP có thể thêm vào để thực hiện thao tác với nhiều byte hay word cùng lúc.

Các lệnh bậc cao.

Các lệnh bậc cao cho phép các ngôn ngữ bậc cao có cấu trúc khối kiểm tra giới hạn của mảng, và tạo không gian bộ nhớ trong ngăn xếp cho các biến cục bộ. Đó là các lệnh **BOUND**, **LEAVE** và **ENTER**. Do các lệnh này chủ yếu được các trình biên dịch sử dụng nên chúng ta sẽ không đề cập tới chúng ở đây.

20.1.2 Chế độ địa chỉ thực.

Sự hình thành một địa chỉ.

Một nhược điểm chính của 8086 đó chính là khả năng sử dụng địa chỉ 20 bit của nó, điều này khiến cho không gian bộ nhớ bị giới hạn trong một MEGA byte. Một Mb bộ nhớ này còn bị giới hạn thêm nữa bởi PC do việc PC giành ra phần địa chỉ trên 640KB cho màn hình và các mục đích khác. 80286 sử dụng 24 bit địa chỉ do đó nó có không gian bộ nhớ địa chỉ đến 2^{24} hay 16MB.

Nếu nhìn thoáng qua thì có vẻ như 80286 có thể giải quyết nhiều vấn đề về sự hạn chế bộ nhớ. Nhưng khi kiểm tra kỹ lại chúng ta thấy rằng dù sao các chương trình chạy dưới DOS không thể sử dụng vùng nhớ mở rộng thêm. DOS được thiết kế cho 8086/8088 tương ứng với chế độ thực trong 80286. Để tương thích với 8086, chế độ địa chỉ thực của 80286 tạo ra địa chỉ vật lý theo cách giống như 8086. Nghĩa là 16 bit địa chỉ đoạn được dịch trái 4 bit để rồi cộng với địa chỉ offset. Con số địa chỉ 20 bit tạo ra trở thành 20 bit thấp của một địa chỉ vật lý 24 bit. 4 bit cao bị xoá đi. Như vậy chúng ta có giới hạn trong 1MB.

Trên thực tế, 80286 có thể truy nhập tối hơn 1MB một chút trong chế độ địa chỉ thực. Để minh họa chúng ta hãy sử dụng địa chỉ đoạn FFFFh và địa chỉ offset FFFFh. Địa chỉ tính được là FFFF0h + FFFFh = 10FFEFh. Trong 8086 bit ngoài cùng bị bỏ đi và địa chỉ vật lý nhận được là OFFEFh. Nhưng đối với 80286 do nó có 24 đường địa chỉ lên ô nhớ 10FFEFh là có thể địa chỉ hoá. Để dàng thấy rằng đối với đoạn FFFFh các byte có địa chỉ offset từ 10h đến FFFFh sẽ có địa chỉ vật lý 21 bit. Bởi vậy ở chế độ địa chỉ thực 80286 có khả năng truy nhập hơn 8086 đến 64Kb. Vùng bộ nhớ nằm trên 1MB này được DOS 5.0 dùng để nạp một số ứng dụng của nó nhờ vậy giành ra thêm một lượng bộ nhớ cho các ứng dụng khác. Chú ý rằng trong rất nhiều máy PC, bit địa chỉ thứ 21 phải được kích hoạt bằng phần mềm trước khi phần bộ nhớ cao hơn có thể sử dụng.

Các chương trình chạy trên nền của DOS.

Dưới sự kiểm soát của DOS, 80286 phải làm việc trong chế độ thực. Tất cả mọi chương trình viết cho 8086 đều có thể chạy trên máy 80286 với hệ điều hành DOS. Một chương trình cho 80286 có thể chứa các lệnh mở rộng. Để biên dịch một chương trình với các lệnh mở rộng chúng ta phải sử dụng dãy hướng biên dịch .286 để tránh thông báo lỗi biên dịch.

Để làm ví dụ về tập lệnh mở rộng, chúng ta hãy viết một thủ tục đưa ra nội dung của BX ở dạng hex. Thuật toán được viết trong chương 7.

```
.286
HEX_OUT      PROC
;đưa ra nội dung của BX dưới dạng hex.
    PUSHA          ;lưu tất cả các thanh ghi
    MOV   CX, 4    ;CX đếm số thứ tự của các chữ số hex
;lặp lại 4 lần
REPEAT:
    MOV   DL, BH   ;lấy byte cao
    SHR  DL, 4    ;bỏ đi chữ số hex thấp
    CMP  DL, 9    ;là chữ số hay chữ
    JG   LETTER   ;nhảy đến letter nếu >9
    OR   DL, 30H   ;<=9, đổi thành mã ASCII
    JMP  PRINT    ;đưa ra màn hình
LETTER:
    ADD  DL, 37H   ;đổi thành chữ
PRINT:
    MOV  AH, 2    ;hàm in ra ký tự trên màn hình
    INT  21H      ;đưa ra số Hex
    SHL  BX, 4    ;dịch chữ số tiếp theo vào vị trí
;cao nhất
    LOOP REPEAT
    POPA          ;phục hồi các thanh ghi
    RET
    HEX_OUT      ENDP
```

20.1.3 Chế độ bảo vệ.

Để tận dụng hoàn toàn sức mạnh của 80286 chúng ta phải sử dụng nó trong chế độ bảo vệ. Khi làm việc trong chế độ bảo vệ, 80286 cung cấp khả năng địa chỉ ảo, nó cho phép chương trình có kích thước lớn hơn rất nhiều kích thước bộ nhớ vật lý của máy tính. Một đặc điểm của chế độ bảo vệ nữa là nó cung cấp khả năng làm việc đa nhiệm, nghĩa là cho phép một số chương trình cùng chạy một lúc. 80286 được thiết kế để làm việc trong chế độ địa chỉ thực ngay khi được

bật lên, chuyển nó sang chế độ bảo vệ thường là công việc của hệ điều hành. Trong phần 20.2 chúng ta sẽ xem một số phần mềm chạy trong chế độ bảo vệ.

Địa chỉ ảo.

Các chương trình ứng dụng trong chế độ bảo vệ vẫn sử dụng địa chỉ segment và offset để chỉ đến các ô nhớ. Tuy nhiên địa chỉ đoạn bây giờ không tương ứng với một đoạn bộ nhớ nhất định nào. Thay vào đó bây giờ nó được gọi là từ chọn đoạn (**segment selector**) và được hệ thống sử dụng để định vị một đoạn vật lý có thể nằm ở mọi nơi trong bộ nhớ. Hình 20.1 mô tả một từ chọn đoạn.

Để theo dõi các đoạn bộ nhớ vật lý được các chương trình sử dụng, hệ điều hành duy trì một tập hợp các bảng gọi là bảng mô tả đoạn (**segment descriptor table**). Mỗi chương trình có một bảng mô tả đoạn cục bộ (**local descriptor table**) chứa các thông tin về các đoạn của chương trình.Thêm vào đó còn có một bảng mô tả toàn cục (**global descriptor table**) chứa các thông tin về các đoạn bộ nhớ mà tất cả các chương trình có thể truy nhập tới. Từ chọn đoạn được dùng để truy nhập tới cái gọi là mô tả đoạn chứa trong bảng mô tả đoạn. Như chúng ta thấy trên hình 20.2, mô tả đoạn xác định kiểu và kích thước đoạn, xác định xem đoạn có mặt hay không và địa chỉ cơ sở 24 bit của đoạn trong bộ nhớ.

| | |
|-------|---------|
| 15 | 3 2 1 0 |
| INDEX | TI RPL |

RPL (Requested Privilege Level) = mức ưu tiên mong muốn.

TI (Table Indicator) = 0, sử dụng bảng mô tả toàn cục
= 1, sử dụng bảng mô tả cục bộ

INDEX = chỉ số để chọn trường mô tả đoạn trong bảng mô tả

Hình 20.1: Từ chọn đoạn.

Quá trình đổi địa chỉ đoạn mà chương trình sử dụng thành địa chỉ vật lý 24 bit diễn ra như sau. Trước hết bit TI trong từ chọn đoạn được sử dụng để chọn bảng mô tả đoạn. TI=0 có nghĩa là chọn bảng toàn cục và TI=1 có nghĩa là chọn bảng mô tả cục bộ. Vị trí của bảng mô tả toàn cục chứa trong thanh ghi GDTR (Global Descriptor Table Register) và một thanh ghi khác, thanh ghi

LDTR (Local Descriptor Table Register) chứa địa chỉ của bảng mô tả cục bộ của chương trình đang chạy. Tiếp theo, một trường mô tả đoạn được xác định bằng 13 bit chỉ số trong trong từ chọn đoạn sẽ được truy nhập trong bảng mô tả đã chọn để nhận được 24 bit địa chỉ đoạn. Cuối cùng địa chỉ offset được cộng với địa chỉ đoạn để tạo thành địa chỉ vật lý 24 bit của một ô nhớ.

| | | | | | | |
|-----------------------|-----|-----|-----------------------|------|---|-----------------------|
| 15 | 8 7 | 0 | | | | |
| Để giành, phải bằng 0 | | | | | | |
| +6 | P | DPL | S | TYPE | A | BASE ₁₃₋₁₆ |
| +4 | | | BASE ₁₅₋₁₆ | | | |
| +2 | | | LIMIT ₁₅₋₀ | | | |
| 0 | | | | | | |

P = Present

DPL = mô tả mức ưu tiên

S = mô tả đoạn

A = truy nhập

BASE = địa chỉ bộ nhớ vật lý của đoạn

LIMIT = cỡ đoạn

Một bảng mô tả có thể có kích thước tối 64KB. Do mỗi trường mô tả đoạn có kích thước là 8 byte nên mỗi bảng mô tả đoạn có thể chứa tối 8K (2^{13}) trường mô tả. Mỗi trường mô tả xác định một đoạn của một chương trình. Một chương trình có thể vừa chọn bảng cục bộ của nó vừa có thể chọn bảng toàn cục, do đó nó có thể định nghĩa tới 16K đoạn. Do kích thước cực đại của mỗi đoạn là 64KB, một chương trình có thể dùng tới $16K \times 64$ KB bằng 2^{30} hay một GB (giga byte) bộ nhớ. Bộ nhớ đó được gọi là bộ nhớ ảo (**virtual memory**) vì 80286 chỉ có 16MB bộ nhớ vật lý.

Các đoạn bộ nhớ ảo của một chương trình được chứa trong ổ đĩa. Hệ điều hành sẽ nạp chúng vào bộ nhớ khi cần. Hệ điều hành sử dụng bit P trong trường mô tả để theo dõi xem đoạn tương ứng đã được nạp vào bộ nhớ hay chưa. Khi một đoạn ảo chưa được nạp vào bộ nhớ thì bit P trong trường mô tả tương ứng sẽ bị xoá. Một ví dụ là khi chương trình lớn hơn rất nhiều kích thước bộ nhớ vật lý. Nó phải được nạp dần vào bộ nhớ. Mỗi khi có một chỉ thị định đến một địa chỉ đoạn chưa được nạp vào trong bộ nhớ, hệ điều hành sẽ được phần cứng thông báo thông qua một ngắt. Khi đó hệ điều hành sẽ nạp đoạn vào bộ nhớ và thực hiện lại chỉ thị. Tất nhiên có thể cần phải cắt một đoạn bộ nhớ vào đĩa để lấy chỗ cho đoạn mới này.

Các nhiệm vụ (Tasks).

Công việc cơ bản khi làm việc trong chế độ bảo vệ được gọi là task, nó tương tự như việc thực hiện một chương trình trong chế độ địa chỉ thực. Mỗi công cụ có một bảng mô tả cục bộ. Tại mỗi thời điểm chỉ có một tác vụ được thực hiện, nhưng hệ điều hành có thể chuyển đổi qua lại giữa các tác vụ bằng các ngắt. Ngoài ra mỗi tác vụ còn có thể gọi một tác vụ khác. Để bảo đảm tính an toàn thì mỗi tác vụ được gán một mức ưu tiên (**privilege level**). Có 3 mức ưu tiên từ 0 đến 3. Mức 0 được coi là được ưu tiên nhất và mức 3 là mức kém nhất. Hệ điều hành hoạt động ở mức 0 còn các chương trình ứng dụng làm việc ở mức 3. Có một số lệnh được ưu tiên như nạp các thanh ghi bảng mô tả chỉ có thể thực hiện bằng các tác vụ ở mức ưu tiên 0. Một tác vụ hoạt động ở một mức ưu tiên này không thể truy nhập dữ liệu của tác vụ ở một mức ưu tiên cao hơn, nó cũng không thể gọi thủ tục của tác vụ ở mức ưu tiên thấp hơn.

20.1.4 Bộ nhớ mở rộng (extended memory).

Như chúng ta đã thấy, 80286 không thể truy nhập tới bộ nhớ tiềm tàng của nó khi hoạt động trong chế độ địa chỉ thực. Điều này cũng đúng cho các bộ vi xử lý 80386 hay 80486. Phần bộ nhớ ở trên 1MB, gọi là bộ nhớ mở rộng (**extended memory**), thông thường không dùng được với các chương trình ứng dụng chạy trên nền DOS. Tuy nhiên một chương trình vẫn có thể truy nhập tới vùng nhớ mở rộng thông qua ngắt 15H. Hai hàm làm việc với bộ nhớ mở rộng là 87h và 88h. Một chương trình có thể sử dụng hàm 88h để xác định kích thước bộ nhớ mở rộng khả dụng và dùng hàm 87h để chuyển dữ liệu đến bộ nhớ mở rộng hay từ bộ nhớ mở rộng về bộ nhớ quy ước. Cần nhắc các bạn một điều khi sử dụng ngắt 15h để thao tác với bộ nhớ mở rộng là một phần của nó có thể đã được một chương trình khác sử dụng như RAMDRIVE.SYS, và do đó chương trình của bạn có thể làm loạn bộ nhớ. Một giải pháp tốt hơn để truy nhập bộ nhớ mở rộng là gọi các chương trình quản lý bộ nhớ mở rộng.

| |
|---|
| Ngắt 15h Hàm 87h: chuyển một khối nhớ mở rộng |
| Vào: AH=87h CX=số word cần chuyển ES:SI=địa chỉ bảng mô tả toàn cục |
| Ra: AH=0 nếu thành công |

Mỗi khi hàm 87h được gọi, trình phục vụ ngắt tạm thời chuyển bộ vi xử lý về chế độ bảo vệ. Sau khi dữ liệu đã được chuyển xong, bộ vi xử lý lại được chuyển về chế độ địa chỉ thực. Đó chính là lý do tại sao phải cần có bảng mô tả toàn cục.

Ngắt 15h Hàm 88h:

Lấy kích thước bộ nhớ mở rộng

Vào: AH=88h

Ra: AX= kích thước bộ nhớ mở rộng (KB)

Chương trình PGM20_1.ASM chép dữ liệu trong mảng SOURCE đến bộ nhớ mở rộng tại địa chỉ 110000h và chép thông tin tại đó ngược trở lại bộ nhớ quy ước vào mảng DESTINATION. Vì chương trình không thực hiện việc vào ra nào, bộ nhớ có thể được kiểm tra bằng chương trình DEBUG.

Chương trình nguồn PGM20_1.ASM

TITLE PGM20_1: COPY EXTENDED MEMORY

```
.MODEL      SMALL
.286
.STACK
.DATA
SOURCE      DB      'HI, THERE!'
DESTINATION DB      10 DUP(0)
GDT          DB      48 DUP(?)           ;bảng mô tả toàn cục
SRC_ADDR    DB      ?,?,?             ;24 bit địa chỉ nguồn
DST_ADDR    DB      ?,?,?             ;24 bit địa chỉ đích
.CODE
MAIN     PROC
        MOV  AX, @DATA
        MOV  DS, AX
        MOV  ES, AX
;chuyển 24 bit địa chỉ nguồn vào SRC_ADDR
        MOV  WORD PTR SRC_ADDR, DS ;lấy địa chỉ đoạn
        SHL  WORD PTR SRC_ADDR, 4  ;dịch sang trái 4 bit
        MOV  AX, DS                ;lấy 4 bit cao nhất
        SHR  AX, 4
        MOV  SRC_ADDR+2, AH
        LEA  SI, SOURCE            ;địa chỉ offset của nguồn
        ADD  WORD PTR SRC_ADDR, SI ;cộng với
                                ;địa chỉ đoạn
```

```

        ADC    SRC_ADDR+2,0           ;để phòng có nhỡ
;chuyển địa chỉ đích vào DST_ADDR
        MOV    DST_ADDR,0
;địa chỉ đích là 110000h
        MOV    DST_ADDR+1,0
        MOV    DST_ADDR+2,11H
;chuẩn bị các thanh ghi
        LEA    SI,SRC_ADDR          ;địa chỉ nguồn
        LEA    DX,DST_ADDR          ;địa chỉ đích
        MOV    CX,5                  ;số từ cần chuyển
        LEA    DI,GDT                ;bảng mô tả toàn cục
;chuyển dữ liệu đến bộ nhớ mở rộng
        CALL   COPY_EMEM
;chuẩn bị địa chỉ nguồn mới
        MOV    SRC_ADDR,0           ;địa chỉ nguồn bây giờ là
        MOV    SRC_ADDR+1,0          ;110000h
        MOV    SRC_ADDR+2,11H
;chuẩn bị địa chỉ đích
        MOV    WORD PTR DST_ADDR,DS ;lấy địa chỉ đoạn
        SHL    WORD PTR DST_ADDR,4  ;dịch sang trái 4 bit
        MOV    AX,DS                 ;lấy 4 bit cao nhất
        SHR    AX,4
        MOV    DST_ADDR+2,AH
        LEA    SI,DESTINATION       ;địa chỉ offset đích
        ADD    WORD PTR DST_ADDR,SI ;cộng vào
                                ;địa chỉ đoạn
        ADC    DST_ADDR+2,0           ;để phòng có nhỡ
;chuẩn bị các thanh ghi
        LEA    SI,SRC_ADDR          ;địa chỉ nguồn
        LEA    DX,DST_ADDR          ;địa chỉ đích
        MOV    CX,5                  ;số từ được chuyển
        LEA    DI,GDT                ;bảng mô tả toàn cục
        CALL   COPY_EMEM            ;chép đến DESTINATION
        MOV    AH,4CH                 ;DOS exit
        INT    21H
MAIN   ENDP
;
COPY_EMEM PROC
;chuyển khối nhớ tới và từ bộ nhớ mở rộng
;Vào: ES:DI=địa chỉ của bộ đệm 48 byte được dùng
;      như bảng mô tả toàn cục
;      CX=số từ cần chuyển
;      SI=địa chỉ nguồn (24 bit)
;      DI=địa chỉ đích (24bit)

```

```

;khởi tạo bảng mô tả toàn cục và 6 trường mô tả
    PUSHAD           ;cất các thanh ghi
;trường mô tả đầu tiên bằng 0,
;có nghĩa là 8 byte bằng 0
    MOV   AX,0
    SOTSW
    SOTSW
    SOTSW
    SOTSW
;trường mô tả thứ hai bằng 0,có nghĩa là 8 byte bằng 0
    SOTSW
    SOTSW
    SOTSW
    SOTSW
;trường mô tả thứ 3 là đoạn nguồn
    SHL   CX,1      ;đổi thành số byte
    DEC   CX
    MOV   AX,CX      ;kích thước đoạn tinh bằng byte
    SOTSW
    MOVSXB          ;địa chỉ nguồn, 3 byte
    MOVSXB
    MOVSXB
    MOV   AL,93H ;byte quyền truy nhập
    STOSB
    MOV   AX,0
    STOSW
;trường mô tả thứ 4 là đoạn đích
    MOV   AX,CX      ;kích thước đoạn tinh bằng
    STOSW          ;byte
    MOV   SI,DX      ;địa chỉ đích, 3 byte
    MOVSXB
    MOVSXB
    MOVSXB
    MOV   AL,93H      ;byte quyền truy nhập
    STOSB
    MOV   AX,0
    STOSW
;trường mô tả thứ 5 đặt bằng 0
    STOSW
    STQSW
    STOSW
    STOSW
;trường mô tả thứ 6 đặt bằng 0
    STOSW

```

```

        STOSW
        STOSW
        STOSW
;phuc hồi các thanh ghi
        POPA
;chuyển dữ liệu
        MOV    SI,DI      ;ES:SI trả tới bảng mô tả GDT
        MOV    AH,87H
        INT    15H
        RET
COPY_EMEM    ENDP
;
        END    MAIN

```

Việc chép số liệu được thực hiện bởi thủ tục COPY_EMEM. Thủ tục này nhận trong CX số từ cần chuyển, trong SI ô nhớ chứa địa chỉ nguồn 24 bit và trong DI ô nhớ chứa địa chỉ đích 24 bit. Bộ đệm nguồn và đích có thể ở mọi nơi trong 16MB bộ nhớ vật lý của 80286. Trước tiên 80286 chuẩn bị bảng mô tả toàn cục trong đó có chứa các bộ đệm nguồn và đích như là các đoạn của chương trình, sau đó nó sử dụng hàm 87h ngắt 15h để tiến hành việc chuyển số liệu.

20.2 Các hệ thống làm việc ở chế độ bảo vệ.

Bây giờ khi đã có một số khái niệm về phần cứng làm việc ra sao trong chế độ bảo vệ, chúng ta hãy chuyển sang các phần mềm. Cho đến nay vẫn chưa có một hệ điều hành đa nhiệm chuẩn nào cho PC. Chúng ta hãy cùng nhau xem xét Windows 3 và OS/2. Trước hết hãy bàn về vấn đề đa nhiệm.

Đa nhiệm (multitasking).

Trong môi trường đơn nhiệm như DOS, một chương trình nắm quyền điều khiển CPU và chỉ trả quyền điều khiển khi nó muốn. Tất nhiên có một ngoại lệ đối với các ngắt. Trong các môi trường đa nhiệm như Windows và OS/2, hệ điều hành xác định chương trình nào đang nắm quyền điều khiển và một vài chương trình có thể cùng chạy một lúc. Thực tế mỗi chương trình có được một khoảng thời gian nhỏ để thực hiện và khi hết khoảng thời gian đó đến lượt của chương trình khác. Bằng cách thực hiện việc quay vòng đó rất nhanh giữa một số chương trình, máy tính tạo cho ta cảm giác tất cả các chương trình dường như đang thực hiện cùng lúc.

20.2.1 Windows và OS/2.

Windows 3

Windows 3 là một giao diện đồ họa với người sử dụng (**graphical user interface - gui**) khá quen thuộc trên máy PC. Mỗi tác vụ được trình bày trong một hộp trên màn hình gọi là các cửa sổ (windows). Một cửa sổ có thể làm rộng ra để chiếm toàn màn hình hay co lại thành một phần tử đồ họa gọi là một biểu tượng (**icon**). Một chương trình ứng dụng của Windows 3 có thể cung cấp các phục vụ của nó thông qua các thực đơn (**menu**) cho người sử dụng. Để chọn một mục bất kỳ người sử dụng chỉ việc đặt con trỏ màn hình bằng con chuột (**mouse**) đúng vào mục đó và nhấn chuột (**click**).

Windows 3 có thể làm việc trong 3 chế độ: *chế độ chuẩn, chế độ thực và chế độ 386 nâng cấp*. Khi Windows 3 chạy trên một máy 8086 hay trong chế độ địa chỉ thực của một máy có vi xử lý tiên tiến khác thì nó hoạt động trong chế độ thực. Một chương trình phải kết thúc trước khi một chương trình khác có thể thực hiện.

Chế độ chuẩn của Windows 3 tương ứng với chế độ bảo vệ của 80286. Windows 3 sử dụng khả năng đa nhiệm của 80286 để cung cấp các đa ứng dụng Windows 3. Nó cũng có thể thực hiện các chương trình viết cho DOS, tuy nhiên để chạy các chương trình đó nó phải chuyển bộ vi xử lý trở về chế độ địa chỉ thực. Trong trường hợp này về cơ bản các chương trình khác không thể chạy được nữa. Windows 3 cần phải có ít nhất 192KB bộ nhớ mở rộng để chạy trong chế độ này. Nếu không nó chỉ có thể chạy trong chế độ thực.

Chế độ 386 nâng cấp của Windows 3 tương ứng với chế độ bảo vệ của 80386. Trong phần tới chúng ta sẽ thấy 80386 có thể thực hiện nhiều ứng dụng của 8086 cùng lúc trong chế độ bảo vệ. Chính vì vậy, trong chế độ 386 nâng cấp, Windows 3 có thể thực hiện được sự đa nhiệm các ứng dụng của nó cũng như của DOS. Máy tính phải có bộ vi xử lý 80386 hay 80486 với bộ nhớ mở rộng ít nhất 1MB để chạy Windows 3 trong chế độ này.

Windows 3 vẫn không phải là một hệ điều hành đầy đủ vì nó vẫn cần dùng DOS trong rất nhiều thao tác với tập tin. Để chạy Windows 3 chúng ta phải khởi động máy bằng DOS rồi sau đó mới thực hiện chương trình Windows 3.

OS/2

Khác với Windows 3, OS/2 là một hệ điều hành hoàn chỉnh. OS/2 phiên bản 1.0 được thiết kế cho chế độ bảo vệ của 80286, nó yêu cầu ít nhất 2MB bộ nhớ mở rộng. OS/2 2.0 đáp ứng được chế độ bảo vệ của 80386.

Trên nền OS/2, một chương trình có thể thực hiện một số công việc cùng lúc. Chẳng hạn chương trình có thể vừa hiển thị một tệp trên màn hình đồng thời

chép một tệp khác vào đĩa. Bản thân chương trình được gọi là **process** và mỗi công việc nói trên được gọi là một **luồng việc (thread)**. Mỗi luồng là một đơn vị cơ bản của công việc trên OS/2 và chúng ta có thể thấy rằng nó tương ứng với một tác vụ được cung cấp bởi phần cứng. Một luồng có thể gọi một luồng khác bằng các chương trình phục vụ của hệ thống.

Nói chung một process bao gồm một hay nhiều luồng công việc với một số tài nguyên của hệ thống như các file hay thiết bị mở mà nó dùng chung cho tất cả các luồng trong process. Khái niệm process giống như khái niệm về file thực hiện trong DOS.

20.2.2 Lập trình

Chúng tôi sẽ chỉ đưa ra ở đây một vài chương trình đơn giản cho OS/2 để làm ví dụ minh họa. Các chương trình ứng dụng phức tạp hơn của OS/2 và Windows 3 nằm ngoài phạm vi của cuốn sách này.

Một khác nhau đáng kể giữa DOS và OS/2 đối với người lập trình là ở chỗ, để thực hiện các lời gọi hệ thống và các công việc vào ra trong OS/2, chương trình phải thực hiện những lời gọi xa đến các thủ tục của hệ thống thay vì thực hiện các lệnh INT. Các tham số được đưa vào ngăn xếp trước khi tiến hành các lời gọi. Điều này được thực hiện để tối ưu hoá việc giao tiếp với ngôn ngữ bậc cao. Các thủ tục của hệ thống có thể được nối vào các chương trình ứng dụng bằng cách ghép vào các thư viện hệ thống. Thực tế các thư viện hệ thống chỉ chứa các lời gọi đến các thủ tục chứ không phải các đoạn lệnh. Các thủ tục của hệ thống chứa trong các file .DLL và nó sẽ được liên kết mỗi khi chương trình được nạp. Các lời gọi hàm của OS/2 được gọi là giao diện với chương trình ứng dụng (**application program interface-API**).

Chương trình "HELLO"

Để làm ví dụ đầu tiên, chúng tôi trình bày chương trình in ra dòng chữ "HELLO". Chương trình được viết trong chương trình nguồn PGM20_2.ASM.

Chương trình nguồn PGM20_2.ASM

```
TITLE      PGM20_2:      PRINT HELLO
.286
.MODEL     SMALL
.STACK
;
.DATA
```

```

MSG      DB      'HELLO!'
NUM_BYTES DW          0
;
.CODE
        EXTRN DOSWRITE:FAR, DOSExit:FAR
MAIN    PROC
;đưa các biến cho DOSWRITE vào ngăn xếp
        PUSH 1           ;thẻ file cho màn hình
        PUSH DS          ;địa chỉ của thông báo: đoạn
        PUSH offset MSG ;offset
        PUSH 5            ;độ dài thông báo
        PUSH DS          ;địa chỉ của số byte đã
                         ;viết:đoạn
        PUSH offset NUM_BYTES ;offset
        CALL DOSWRITE   ;viết ra màn hình
; đưa các biến cho DOSExit vào ngăn xếp
        PUSH 1           ;mã lệnh 1=kết thúc mọi luồng việc
        PUSH 0           ;mã trả về 0
        CALL DOSExit    ;kết thúc
MAIN    ENDP
END    MAIN

```

Chú ý rằng chúng ta không cần khởi tạo DS. Khi chương trình được nạp OS/2 tự động đặt DS trả tới đoạn dữ liệu, và nó cũng không tạo ra khối PSP cho chương trình. Thêm một điểm nữa OS/2 chỉ cung cấp các file .EXE.

Chúng ta đã sử dụng 2 hàm API, DOSWRITE để ghi ra màn hình và DOSExit để kết thúc chương trình. Hàm DOSWRITE có chức năng ghi ra một file, các biến số cần thiết là thẻ file, địa chỉ vùng đệm, chiều dài vùng đệm và địa chỉ của biến số byte được viết ra. Thẻ file cho màn hình là 1. Biến số byte viết ra nhận số byte viết được vào file, giá trị này có thể sử dụng để kiểm tra lỗi. Các biến phải được đưa vào ngăn xếp theo đúng trình tự trên để gọi thực hiện DOSWRITE.

DOSExit có thể dùng để kết thúc một luồng hay tất cả các luồng trong process. Các biến cần thiết là (1) mã lệnh thực hiện để kết thúc một luồng hay toàn bộ các luồng và (2) mã trả về sẽ được chuyển cho hệ thống tạo lên process. Biến số để kết thúc thông thường gồm mã lệnh thực hiện bằng 1 và mã trả về bằng 0.

Trong OS/2, các thủ tục được gọi có trách nhiệm xoá khỏi ngăn xếp các biến được gửi cho chúng khi chúng kết thúc, do đó chúng ta không cần dùng các lệnh POP trong chương trình.

Các hàm DOSWRITE và DOSExit được định nghĩa trong file thư viện có tên DOSCALL.LIB để ghép với chương trình chúng ta dùng lệnh:

LINK PGM20_2,,,DOSCALL

Chương trình tạo hiển thị phím ấn.

Để làm ví dụ thứ 2 chúng tôi trình bày một chương trình hiển thị những phím được đánh vào từ bàn phím.

Chương trình nguồn PGM20_3.ASM

```
TITLE PGM20_3: ECHO PROGRAM
.286
.MODEL      SMALL
.STACK
;
.DATA
BUFFER       DB    20 DUP(0)
NUM_CHARS    DW    0
NUM_BYTES    DW    0
;
.CODE
        EXTRN DOSREAD:FAR, DOSWRITE:FAR, DOSExit:FAR
MAIN PROC
;đưa các biến cho DOSREAD vào ngăn xếp
        PUSH 0          ;thẻ file cho bàn phím
        PUSH DS         ;địa chỉ của bộ đếm :đoạn
        PUSH offset BUFFER ;offset
        PUSH 20          ;chiều dài bộ đếm
        PUSH DS         ;địa chỉ của thông báo: đoạn
        PUSH offset NUM_CHARS ;offset
        CALL DOSREAD   ;đọc vào từ bàn phím
; đưa các biến cho DOSWRITE vào ngăn xếp
        PUSH 1          ;thẻ file cho màn hình
        PUSH DS         ;địa chỉ của thông báo :đoạn
        PUSH offset BUFFER ;offset
        PUSH NUM_CHARS ;chiều dài thông báo
        PUSH DS         ;địa chỉ số byte được viết:đoạn
        PUSH offset NUM_BYTES ;offset
        CALL DOSWRITE  ;viết ra màn hình
; đưa các biến cho DOSExit vào ngăn xếp
        PUSH 1          ;mã lệnh 1=kết thúc mọi luồng
        PUSH 0          ;mã trả về 0
        CALL DOSExit   ;kết thúc
```

```
MAIN    ENDP  
END        MAIN
```

Chúng ta đã sử dụng hàm API DOSREAD để đọc vào từ bàn phím. DOSREAD đọc vào từ một file và nó cần các tham số: thẻ file, địa chỉ vùng đệm, chiều dài vùng đệm và địa chỉ số byte đọc được. Thẻ file cho bàn phím có giá trị là 0, DOSREAD sẽ đọc vào các phím cho đến khi vùng đệm đầy hoặc phím ENTER được ấn. Số các byte đọc được sẽ trả về trong biến chứa số byte đọc được.

Chúng ta đã coi màn hình và bàn phím như các file để gọi DOSREAD và DOSWRITE. Ngoài ra cũng có các hàm VIO (Video) và KBD (Keyboard) API có thể thực hiện nhiều công việc khác về màn hình và bàn phím.

Các ví dụ trên đây chỉ là sự giới thiệu sơ lược về OS/2, để có sự hiểu biết đầy đủ cần tham khảo một cuốn sách khác.

20.3 Các bộ vi xử lý 80386 và 80486.

80386 và 80486 đều là các bộ vi xử lý 32 bit. Như đã nói trong chương 3 chúng rất giống nhau ngoại trừ một điểm là 80486 có chứa mạch xử lý số dấu phẩy động. Trong các phần dưới đây chúng ta tập trung vào 80386 vì 80486 có thể xem như một bộ vi xử lý 80386 nhanh hơn với sự tham gia của mạch xử lý số dấu phẩy động.

80386 có thể thao tác trên cả 2 chế độ: chế độ địa chỉ thực và chế độ bảo vệ giống như 80286.

20.3.1 Chế độ địa chỉ thực.

80386 có 8 thanh ghi công dụng chung 32 bit là EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP. Mỗi thanh ghi lại chứa các thanh ghi 16 bit tương ứng của 8086, chẳng hạn 16 bit thấp của EAX chính là thanh ghi AX. Có 6 thanh ghi đoạn 16 bit là CS, DS, ES, FS, GS trong đó DS, ES, FS, GS là các thanh ghi đoạn số liệu. Thanh ghi 32 bit EFLAGS chứa trong nó 26 bit của thanh ghi cờ 16 bit FLAGS và thanh ghi 32 bit EIP chứa thanh ghi 16 bit IP. Ngoài ra còn có thanh ghi debug, thanh ghi điều khiển và thanh ghi kiểm tra.Thêm vào đó còn có các thanh ghi để quản lý và bảo vệ trong chế độ bảo vệ.

Trong chế độ thực, 80386 có thể thực hiện tất cả các chỉ thị của 80286 ở chế độ địa chỉ thực. Do đó đối với người lập trình, chế độ địa chỉ thực của 80386 tương tự như 8086 với tập lệnh và các thanh ghi được mở rộng.

80386 sử dụng 32 bit địa chỉ nhưng trong chế độ thực nó tạo địa chỉ giống như 8086 do đó nó chỉ có thể định địa chỉ tối đa 1Mb thêm vào 64Kb giống như 80286.

20.3.2 Chế độ bảo vệ.

80386 trong chế độ bảo vệ có thể thực hiện được tất cả các lệnh của 80286. Khi sử dụng hệ điều hành với 80286 ở chế độ bảo vệ cho 80386, mỗi trường mô tả đoạn chứa 24 bit địa chỉ cơ sở do đó chỉ có 16 MB bộ nhớ khả dụng. Thực tế nó có truy nhập 16 MB cộng thêm 64 KB.

80386 ở trong chế độ bảo vệ cho phép trường mô tả đoạn chứa đến 32 địa chỉ cơ sở và địa chỉ offset cũng có 32 bit do đó kích thước của một đoạn bộ nhớ là 2^{32} hay 4 GB. Nó cũng là kích thước của khoảng không gian địa chỉ hóa trên bộ nhớ vật lý. Một chương trình vẫn có thể sử dụng 2^{14} đoạn do đó kích thước của bộ nhớ ảo lên đến $2^{32} \times 2^{14} = 2^{46}$ hay 64 terabyte. Rõ ràng đây là kích thước đủ lớn cho mọi chương trình ứng dụng trong tương lai xa.

Bộ nhớ ảo phân trang.

Cũng có thể tổ chức bộ nhớ ảo thành các trang, hệ điều hành có thể thiết lập một bit trong thanh ghi điều khiển để thông báo sử dụng bảng phân trang. Khi đó 32 bit địa chỉ trong trường mô tả đoạn sẽ được xem như số chọn trang để chọn một trong các bảng 1K trang từ thư mục trang, và một số trang trong bảng đã chọn với một địa chỉ offset trong trang. Thư mục trang có chứa 1K các bảng và mỗi bảng chứa 1K trang còn mỗi trang chứa 4KB. Vì vậy tổng cộng địa chỉ có thể có tới 2^{32} hay toàn bộ 4 gigabyte bộ nhớ vật lý.....

Chế độ 8086 ảo.

80386 cung cấp khả năng thực hiện nhiều chương trình 8086 cùng lúc trong môi trường 80386 chế độ bảo vệ. Bộ vi xử lý hoạt động trong chế độ 8086 ảo khi bit VM (**virtual machine**) trong thanh ghi cờ được thiết lập. Trong chế độ V86, các thanh ghi đoạn được dùng giống như trong chế độ địa chỉ thực, nghĩa là địa chỉ được tạo lên bằng địa chỉ offset cộng với địa chỉ đoạn được dịch trái 4 bit. Địa chỉ tuyến tính này có thể ánh xạ đến mọi địa chỉ vật lý bằng cách sử dụng phương pháp phân trang.

20.3.3 Lập trình với 80386

Lập trình 16 bit

Các lệnh của 80386 ở chế độ địa chỉ thực với các toán hạng 16 bit về cơ bản giống như tập lệnh của 80286. Ngoài ra có một số lệnh mới các bạn có thể tham khảo trong phụ lục F.

Lập trình 32 bit

Khi lập trình 32 bit, cả toán hạng và địa chỉ offset đều có 32 bit. Mã máy cho các lệnh 32 bit của 80386 thực tế giống như đối với các lệnh 16 bit. Chúng ta đã biết rằng 80386 có 2 chế độ hoạt động: chế độ 16 bit và chế độ 32 bit. Do mã lệnh cho 16 bit và 32 bit giống nhau nên kiểu toán hạng phải tùy thuộc vào chế độ hoạt động của 80386. Chế độ hoạt động không ảnh hưởng tới các toán hạng có kích thước byte.

Khi làm việc trong chế độ bảo vệ, 80386 có thể hoạt động trong cả chế độ 16 bit và 32 bit. Chế độ hoạt động được xác định trong trường mô tả đoạn của mỗi tác vụ. Tuy nhiên nó chỉ có thể hoạt động trong chế độ 16 bit khi làm việc trong chế độ địa chỉ thực.

Kết hợp các chỉ thị 16 bit và 32 bit.

Có thể kết hợp các chỉ thị 16 bit và 32 bit trong cùng một chương trình. Phần tiền chương trình (prefix) chồng kích thước đoạn (66h) có thể đặt trước các chỉ thị để thay đổi kích thước ngầm định của toán hạng. Trong chế độ 32 bit phần tiền tố đổi kích thước toán hạng thành 16 bit còn trong chế độ 16 bit cũng vẫn phần tiền tố đó sẽ đổi kích thước toán hạng thành 32 bit. Mỗi chỉ thị cần một tiền tố.

Ngoài ra còn có phần tiền tố chồng kích thước địa chỉ offset (67h). Nó cũng được sử dụng tương tự như phần tiền tố chồng kích thước toán hạng, và cả hai có thể sử dụng trong cùng một chỉ thị.

Chúng ta sẽ minh họa bằng một chương trình cho DOS (chế độ địa chỉ thực 16 bit) sử dụng các toán hạng 32 bit. Chương trình được cho trong chương trình nguồn PGM20_4.ASM đọc vào 2 số không dấu có độ chính xác kép và đưa ra tổng của chúng. Phép cộng được thực hiện bằng các thanh ghi 32 bit.

Chương trình nguồn PGM20_4.ASM

```
TITLE PGM20_4: 32_BIT OPERATION
;nhập vào 2 số và đưa ra tổng của chúng.
;Sử dụng các thao tác 32 bit
```

```

.386
.MODEL      SMALL
S_SEG SEGMENT USE16 STACK
        DB      256 DUP(?)
S_SEG ENDS
D_SEG SEGMENT USE16
FIRST DD    0      ;nơi lưu số 32 bit đầu tiên
D_SEG ENDS
;
NEW_LINE MACRO
;chuyển con trỏ tới đầu dòng mới
    MOV AH,2
    MOV DL,0AH
    INT 21H
    MOV DL,0DH
    INT 21H
    ENDM
PROMPT MACRO
;đưa ra dấu nhắc
    MOV DL,'?'
    MOV AH,2
    INT 21H
    ENDM
C_SEG SEGMENT USE16
ASSUME CS:C_SEG, DS:D_SEG, SS:S_SEG
MAIN PROC
    MOV AX,D_SEG
    MOV DS,AX          ;khởi tạo DS
;đưa ra dấu nhắc
    PROMPT
;xoá EBX
    MOV EBX,0
;đọc số đầu tiên
L1:
    MOV AH,1
    INT 21H
;kiểm tra xem có phải CR
    CMP AL,0DH
    JE NEXT           ;đúng, nhập tiếp số thứ 2
;chuyển số thứ nhất vào EBX
    AND AL,0FH          ;đổi thành dạng nhị phân
    IMUL EBX,10          ;nhân EBX với 10
    MOVZX ECX,AL          ;chuyển AL vào ECX và thêm
                           ;Vào các chữ số 0

```

```

        ADD    EBX,ECX      ;cộng các chữ số.
;Lặp lại
        JMP    L1
;cắt số thứ nhất đi
NEXT:
        MOV    FIRST,EBX
;sang đầu dòng mới
        NEW_LINE
;đưa ra dấu nhắc
        PROMPT
;lại xoá EBX
        MOV    EBX,0
;đọc vào các ký tự
L2:
        MOV    AH,1
        INT    21H
;có phải CR?
        CMP    AL,0DH
        JE     SUMUP          ;đúng, tính tổng của chúng
;không phải đặt chữ số vào trong EBX
        AND    AL,0FH          ;đổi thành dạng nhị phân
        IMUL   EBX,10          ;nhân EBX với 10
        MOVZX  ECX,AL          ;chuyển AL vào ECX và thêm
                                ;Vào các chữ số 0
        ADD    EBX,ECX          ;cộng các chữ số.
;Đọc tiếp
        JMP    L2
;tính tổng
SUMUP:
        ADD    EBX,FIRST         ;EBx chứa tổng
;đổi thành dạng thập phân
        MOV    EAX,EBX
        MOV    EBX,10          ;EAX chứa tổng
                                ;hệ số chia là 10
        MOV    CX,0           ;khởi tạo bộ đếm
L3:   MOV    EDX,0
        DIV    EBX
        PUSH   DX              ;chia ECX:EAX cho EBX
                                ;DX chứa số dư, EAX chứa
                                ;thương số
        INC    CX              ;tăng bộ đếm
        CMP    EAX,0          ;đã xong?
        JG     L3              ;chưa, tiếp tục
;sang đầu dòng mới
        NEW_LINE
        MOV    AH,2            ;hàm viết ra màn hình

```

; đưa tổng số ra màn hình

L4:

```
    POP    DX      ;lấy chữ số ra khỏi ngăn xếp
    OR     DL,30H ;chuyển thành mã ASCII
    INT    21H      ;và in ra
    LOOP   L4
; DOS  exit
    MOV    AH,4CH    ;trở về DOS
    INT    21H
MAIN  ENDP
C_SEG ENDS
END    MAIN
```

Chúng ta sử dụng hàm **MOVZX** của 80386 để chuyển toán hạng nguồn nhỏ hơn vào toán hạng đích và chèn vào các chữ số 0 ở đầu trước. Để hợp dịch các lệnh của 80386 chúng ta phải dùng dẫn hướng biên dịch .386. Tuy nhiên khi dẫn hướng biên dịch .386 được sử dụng trình biên dịch coi chế độ làm việc là 32 bit. Khi chạy chương trình dưới chế độ địa chỉ thực của 80386 chúng ta phải định nghĩa chế độ mặc định là 16 bit. Điều này chỉ có thể thực hiện được bằng các dẫn hướng biên dịch đoạn toàn phần. Một đoạn có thể định nghĩa kiểu **USE** chẳng hạn để định nghĩa kiểu USE 16 bit chúng ta đã viết D_SEG

SEGMENT USE16 trong chương trình. Kiểu USE16 định nghĩa cả kích thước toán hạng và kích thước địa chỉ là 16 bit và tất cả các thanh ghi đoạn của chúng ta đều có kiểu USE16.

TỔNG KẾT

Trong chương này chúng ta đã nghiên cứu những vấn đề sau:

- ◆ 80286 có thể làm việc trong chế độ địa chỉ thực cũng như trong chế độ bảo vệ.
- ◆ Trong chế độ địa chỉ thực, 80286 làm việc như 8086.
- ◆ 80286 sử dụng 24 bit địa chỉ do đó cho phép dung lượng bộ nhớ lên tới 16 MB. Tuy nhiên trong chế độ địa chỉ thực, 80286 chỉ có thể truy nhập tới 1MB.
- ◆ Trong chế độ bảo vệ, 80286 có thể sử dụng tới 1GB bộ nhớ ảo.
- ◆ Windows 3 có thể hoạt động ở 3 chế độ, chế độ thực, chế độ chuẩn và chế độ 386 nâng cấp.
- ◆ OS/2 version 1 có thể làm việc trong chế độ bảo vệ của 80286 và version 2 có thể làm việc trong chế độ bảo vệ của 80386.
- ◆ Các phục vụ hệ thống của OS/2 được mã hoá như những hàm kiểu FAR.
- ◆ 80486 có thể xem như 80386 với đơn vị xử lý số dấu phẩy động.
- ◆ Trong chế độ thực, 80386/80486 hoạt động giống như 80286.
- ◆ Trong chế độ bảo vệ, 80386/80486 có khả năng phân trang bộ nhớ và làm việc trong chế độ 8086 ảo. Đồng thời chúng cũng có khả năng thực hiện được tất cả các lệnh của 80286.

Các thuật ngữ tiếng Anh.

Dynamic linking

Liên kết động - các module được liên kết vào thời điểm chúng được nạp.

Extended instruction set

Tập lệnh mở rộng - các lệnh mới ban đầu được sử dụng trong các bộ vi xử lý 80186 và 80288 và sau cũng được dùng trong các bộ vi xử lý 80286, 80386, 80486.

Global descriptor table

Bảng mô tả toàn cục - bảng mô tả đoạn có chứa thông tin về các đoạn mã có thể được tất cả các tác vụ truy nhập tới.

Graphical user interface - gui

Giao diện đồ họa với người sử dụng - một giao diện mà người sử dụng dùng con trỏ và các biểu tượng đồ họa nhất định để ra lệnh.

| | |
|---|--|
| Local descriptor table register (LDTR) | Thanh ghi bảng mô tả toàn cục - thanh ghi chứa địa chỉ của bảng mô tả toàn cục. |
| Menu | Thực đơn - tập hợp các lệnh có thể chọn lựa được hiển thị trong cửa sổ. |
| Mouse | Chuột - thiết bị dùng để điều khiển vị trí con trỏ trên màn hình. |
| Multitasking | Đa nhiệm - kỹ thuật cho phép nhiều chương trình có thể thực hiện cùng lúc. |
| Privilege level | Mức ưu tiên - đơn vị dùng để xác định khả năng của chương trình có thể thực hiện các lệnh đặc biệt. |
| Process | Việc thực hiện một chương trình. |
| Protected (address) mode | Chế độ bảo vệ - chế độ trong đó các bộ vi xử lý tiến hành hoạt động cho phép bộ nhớ được bảo vệ giữa các chương trình với nhau. |
| Real (address) mode | Chế độ địa chỉ thực - Chế độ hoạt động trong đó, địa chỉ chứa trong lệnh tương ứng với địa chỉ thực của bộ nhớ. |
| Segment descriptor | Trường mô tả đoạn - một điểm nhập trong bảng mô tả mà chứa các thông tin mô tả đoạn của chương trình. |
| Segment descriptor table | Bảng mô tả đoạn - bảng có chứa các trường mô tả đoạn. Có 2 loại bảng mô tả đoạn, bảng mô tả toàn cục và bảng mô tả cục bộ. |
| Segment selector | Số chọn đoạn - giá trị của thanh ghi đoạn, khi bộ vi xử lý chạy trong chế độ bảo vệ, số này sẽ xác định một đoạn trong bảng mô tả. |
| Task | Tác vụ - một đơn vị chương trình với những đoạn của riêng nó. |
| Thread | Tác vụ con của Process. |
| Virtual memory | Bộ nhớ ảo - Bộ nhớ ngoài được hệ điều hành sử dụng để lưu những đoạn của các tác vụ hiện thời không cần thiết. |
| Windows | Cửa sổ - vùng hình chữ nhật trên màn hình |

Các lệnh mới

| | | |
|-------|-------|-------|
| BOUND | INSW | OUTSB |
| ENTER | LEAVE | OUTSW |
| INS | MOVZX | POPA |
| INSB | OUTS | PUSHA |

Các dẫn hướng biên dịch mới.

.286 .386

Bài tập

1. Viết một chương trình cho OS/2 nhập vào một chuỗi và đưa chuỗi ra màn hình 10 lần trên 10 dòng khác nhau.
2. Sử dụng các lệnh của 80386 để thực hiện việc nhân 2 số 32 bit.
3. Sử dụng các lệnh của 80386 cho trong phụ lục F để viết một thủ tục đưa ra vị trí của bit 1 bên trái nhất trong thanh ghi BX.

Bài tập lập trình

4. Sửa chương trình PGM20_4.ASM để nó đưa ra tổng của 2 số có dấu độ chính xác kép.

Phụ lục A

CÁC MÃ HIỂN THỊ CỦA IBM.

IBM PC sử dụng hệ thống các ký tự ASCII mở rộng cho việc hiển thị trên màn hình. Bảng A.1 chỉ ra các ký tự ASCII. Các ký tự điều khiển: BS (backspace), HT (tab), CR (carriage return), ESC (escape), SP (space) tương ứng với các phím Backspace, Tab, Enter, Esc và phím trắng; LF (line feed) nhảy con trỏ đến dòng tiếp theo; BEL (bell) phát một tiếng bip và FF (form feed) chuyển con trỏ máy in đến trang tiếp theo.

Bảng A.2 chỉ ra hệ thống 256 ký tự hiển thị mở rộng. Khi viết một mã hiển thị vào trang hoạt động của bộ nhớ hiển thị, ký tự tương ứng sẽ được hiển thị trên màn hình. Để viết vào bộ nhớ hiển thị, chúng ta có thể sử dụng các hàm 9h, 0Ah, 0Eh và 13h của ngắt 10h. Hàm 9h và 0Ah viết được mọi ký tự vào bộ nhớ hiển thị. Hàm 0Eh và 13h phát hiện các mã ký tự điều khiển 07h (bell), 08h (backspace), 0Ah (line feed) và 0Dh (carriage return) và thực hiện các chức năng điều khiển thay vì viết các mã này vào bộ nhớ hiển thị.

BẢNG MÃ ASCII với 128 kí tự đầu tiên

| Hexa-decimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|--------------|-------------|-------------|----------|---------|---------|---------|----------|--------------|
| 0 | <NULL> 0 | <DLE> 16 | | 0 48 | @ 64 | P 80 | ` 96 | p 112 |
| 1 | | <DC1> 17 | ! 33 | 1 49 | A 65 | Q 81 | a 97 | q 113 |
| 2 | | <DC2> 18 | " 34 | 2 50 | B 66 | R 82 | b 98 | r 114 |
| 3 | * | <DC3> 19 | # 35 | 3 51 | C 67 | S 83 | c 99 | s 115 |
| 4 | ♦ | <DC4> 20 | \$ 36 | 4 52 | D 68 | T 84 | d 100 | t 116 |
| 5 | ♣ | | % 37 | 5 53 | E 69 | U 85 | e 101 | u 117 |
| 6 | ♠ | <SYN> 22 | & 38 | 6 54 | F 70 | V 86 | f 102 | v 118 |
| 7 | <BEL> 7 | | | 7 55 | G 71 | W 87 | g 103 | w 119 |
| 8 | <BS> 8 | <CAN> 24 | (40 | 8 56 | H 72 | X 88 | h 104 | x 120 |
| 9 | <HT> 9 | <END> 25 |) 41 | 9 57 | I 73 | Y 89 | i 105 | y 121 |
| A | <LF> 10 | | * | : | J 74 | Z 90 | j 106 | z 122 |
| B | <VT> 11 | <ESC> 27 | + | ; | K 75 | [91 | k 107 | { 123} |
| C | <FF> 12 | <SO> 28 | , | < 44 | L 76 | \ 92 | l 108 | 124 |
| D | <CR> 13 | | - | = 45 | M 77 |] 93 | m 109 | } 125 |
| E | <FS> 14 | | . | > 46 | N 78 | ^ 94 | n 110 | ~ 126 |
| F | <STX> 15 | | / | ? | O 79 | — 95 | o 111 | 127 |

NỬA SAU CỦA BẢNG MÃ

| Hexa-decimal | 8 | 9 | A | B | C | D | E | F |
|--------------|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | ç | é | á | | | | α | ≡ |
| | 128 | 144 | 160 | 176 | 192 | 208 | 224 | 240 |
| 1 | ü | æ | í | | | | β | ± |
| | 129 | 145 | 161 | 177 | 193 | 209 | 225 | 241 |
| 2 | é | æ | ó | | | | γ | ≥ |
| | 130 | 146 | 162 | 178 | 194 | 210 | 226 | 242 |
| 3 | â | ô | ú | | | | π | ≤ |
| | 131 | 147 | 163 | 179 | 195 | 211 | 227 | 243 |
| 4 | ä | ö | ñ | | | | Σ | ʃ |
| | 132 | 148 | 164 | 180 | 196 | 212 | 228 | 244 |
| 5 | à | ò | ñ | | | | σ | ј |
| | 133 | 149 | 165 | 181 | 197 | 213 | 229 | 245 |
| 6 | å | û | á | | | | μ | ÷ |
| | 134 | 150 | 166 | 182 | 198 | 214 | 230 | 246 |
| 7 | ç | ù | º | | | | τ | ≈ |
| | 135 | 151 | 167 | 183 | 199 | 215 | 231 | 247 |
| 8 | ê | ÿ | è | | | | Φ | ● |
| | 136 | 152 | 168 | 184 | 200 | 216 | 232 | 248 |
| 9 | ë | ö | | | | | θ | · |
| | 137 | 153 | 169 | 185 | 201 | 217 | 233 | 249 |
| A | è | ú | ¬ | | | | Ω | - |
| | 138 | 154 | 170 | 186 | 202 | 218 | 234 | 250 |
| B | ï | ç | ½ | | | | δ | √ |
| | 139 | 155 | 171 | 187 | 203 | 219 | 235 | 251 |
| C | î | £ | ¼ | | | | ∞ | ʌ |
| | 140 | 156 | 172 | 188 | 204 | 220 | 236 | 252 |
| D | ì | ¥ | ¡ | | | | φ | 2 |
| | 141 | 157 | 173 | 189 | 205 | 221 | 237 | 253 |
| E | À | | « | | | | € | ■ |
| | 142 | 158 | 174 | 190 | 206 | 222 | 238 | 254 |
| F | Ã | f | » | | | | ⌚ | |
| | 143 | 159 | 175 | 191 | 207 | 223 | 239 | 255 |

Phụ lục B

CÁC LỆNH CỦA DOS.

Trong phụ lục này chúng tôi sẽ chỉ ra một số lệnh thường dùng của DOS.

Chú ý: Trong các phần sau đây, chúng tôi có thể sử dụng hai ký tự đặc biệt trong các tên file hay phần mở rộng. Ký tự "?" đại diện cho một ký tự bất kỳ còn ký tự "*" đại diện cho một hoặc nhiều ký tự trong tên file hay phần mở rộng.

BACKUP

Tạo ra một bản sao của các file trên đĩa.

Ví dụ: BACKUP C: A:

Chép các file trong thư mục C hiện tại tạo ra một bản sao trong đĩa A.

CLS (Clear screen)

Xoá màn hình hiển thị và chuyển con trỏ đến góc trái trên.

Ví dụ: CLS

COPY

Chép các file từ một đĩa hay thư mục sang đĩa hay thư mục khác.

Ví dụ 1: COPY A:FILE1.TXT B:

Chép FILE1 từ đĩa A sang đĩa B. Ở đĩa hiện hành không cần chỉ ra trong lệnh.
Có thể đưa ra một tên khác cho file tạo ra.

Ví dụ 2: COPY FILE1.TXT B:FILE2.TXT

Chép FILE1.TXT từ đĩa trong ở hiện hành thành FILE2.TXT trên đĩa trong ở B.

Ví dụ 3: COPY A:/*.* B:

Chép tất cả các file từ đĩa A sang đĩa B.

DATE

Thay đổi ngày hệ thống. Ngày này được ghi như một điểm nhập thư mục với mọi file mà bạn tạo nên. Dạng của nó là mm-dd-yy.

Ví dụ: DATE 07-14-90

DIR (Directory)

Liệt kê các điểm nhập thư mục.

Ví dụ 1: DIR

Liệt kê tất cả các điểm nhập thư mục của ổ đĩa hiện tại. Mỗi điểm nhập có tên file, kích thước và ngày tháng. Có thể liệt kê các điểm nhập thư mục trong một thư mục hay trong một ổ đĩa khác bằng cách chỉ ra tên của ổ đĩa hay thư mục.

Ví dụ 2: DIR C*.*

Liệt kê tất cả các điểm nhập thư mục và file bắt đầu bằng chữ C với phần mở rộng bất kỳ.

ERASE (hay DEL)

Xoá một file.

Ví dụ 1: ERASE FILE1.TXT

Xoá file FILE1.TXT trong thư mục và ổ đĩa hiện tại.

Ví dụ 2: ERASE * .OBJ

Xoá tất cả các file có phần mở rộng là .OBJ trong ổ đĩa hiện tại.

FORMAT

Khởi tạo một đĩa.

Ví dụ: FORMAT A:

Tạo dạng đĩa trong ổ A. Chú ý: FORMAT huỷ bỏ mọi nội dung trước đó có trên đĩa. Một đĩa mới phải được "FORMAT" trước khi sử dụng.

PRINT

In các file bằng máy in.

Ví dụ: PRINT MYFILE.TXT

In file có tên MYFILE.TEXT trong ổ đĩa A.

RENAME (hay REN)

Đổi tên của một file.

Ví dụ: REN FILE1.TXT MYFILE.TXT

Đổi tên File FILE1.TXT thành MYFILE.TXT

RESTORE

Phục hồi một file từ đĩa sao.

Ví dụ: RESTORE A: C:

Chép bản sao các file từ đĩa A sang đĩa C.

TIME

Thay đổi thời gian của hệ thống. Thời gian được ghi như một điểm vào thư mục của mọi file mà bạn tạo nên. Dạng của nó là hh:mm:ss. Phạm vi của giờ là 0-23.

Ví dụ: TIME 16:47:00

TYPE

Hiển thị nội dung của một file trên màn hình hiển thị.

Ví dụ: TYPE MYFILE.TXT

Hiển thị file có tên MYFILE.

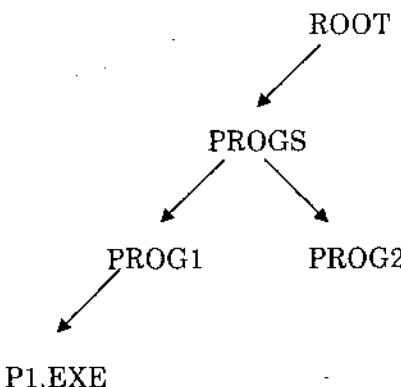
B.1 Các thư mục cấu trúc cây.

Các phiên bản của DOS từ version 2.1 trở về sau cung cấp khả năng đưa các file trên đĩa có liên quan vào trong các thư mục của chúng.***

Khi một đĩa được format, một thư mục đơn được gọi là thư mục gốc (root directory) được tạo nên. Nó có thể chứa đến 112 file đối với một đĩa mềm 5^{1/4} hai mặt mật độ kép.

Thư mục gốc có thể chứa các tên hay các thư mục khác gọi là các thư mục con. Các thư mục con được xem như các file bình thường; chúng có tên 1-8 ký tự và phần mở rộng từ 1 đến 3 ký tự.

Để minh họa cho các lệnh sẽ nêu ở sau, chúng tôi sử dụng thư mục có cấu trúc sau đây như một ví dụ.



Trong đó PROGS là một thư mục con của thư mục gốc. PROG1 và PRG2 là các thư mục con của PROGS. P1.EXE là một file trong thư mục PROG1.

Đường dẫn của một file chứa một chuỗi các tên thư mục con ngăn cách bằng dấu gạch chéo (\) và kết thúc bằng tên file. Nếu chuỗi bắt đầu bằng '\' thì đường dẫn bắt đầu tại thư mục gốc; nếu không, nó bắt đầu từ thư mục hiện thời.

CHDIR (hay CD)

Thay đổi thư mục hiện tại.

Ví dụ 1: CD\

Thiết lập thư mục gốc trở thành thư mục hiện hành.

Ví dụ 2: CD\ PROGS

PROGS là thư mục hiện hành.

Ví dụ 3: CD PRO1

Sau ví dụ 2, biến PRO1 thành thư mục hiện hành.

Ví dụ 4: CD\ PRO1

DOS sẽ trả lời "invalid directory" bởi vì PRO1 không phải là một thư mục con của thư mục gốc.

Ví dụ 5: CD

Hiển thị đường dẫn của thư mục hiện thời. Như vậy sau ví dụ 3, nếu ta đang thao tác trên ổ đĩa C, DOS sẽ trả lời: C:\PROGS\PRO1.

MKDIR (hay MD)

Tạo lập một thư mục con trên đĩa.

Ví dụ chúng ta sẽ tạo lập cấu trúc cây thư mục ở trên đĩa C:

```
C>CD\  
C>MD\ PROGS  
C>MD\ PROGS\ PRO1  
C>MD\ PROGS\ PRO2
```

RMDIR (hay RD)

Xoá một thư mục con trên đĩa. Thư mục con cần xoá phải rỗng. Thư mục con cuối cùng của đường dẫn chỉ ra là thư mục bị xoá.

Ví dụ chúng ta sẽ xoá file P1A.EXE và tất cả các thư mục đã nêu trên trong đĩa C:

C>ERASE\ PROGS\ PRO1\ P1A.EXE

C>RM\ PROGS\ PRO1

C>RM\ PROGS\ PRO2

C>RM\ PROGS

Phục lục C

CÁC NGẮT CỦA DOS VÀ BIOS.

C.1. Giới thiệu.

Trong phục lục này, chúng tôi sẽ chỉ ra một số ngắt thường dùng của DOS và BIOS. Chúng ta sẽ bắt đầu với ngắt 10h. Các ngắt 0-Fh ít được sử dụng trong các chương trình ứng dụng, tên của chúng được đưa ra trong bảng C.1.

Bảng C.1. Các ngắt từ 0 đến Fh.

| CÁC NGẮT | CÁCH DÙNG |
|----------|--------------------------|
| 0h | Chia cho 0 |
| 1h | Thực hiện từng bước |
| 2h | NMI |
| 3h | Điểm dừng |
| 4h | Tràn |
| 5h | In màn hình |
| 6h | Dự trữ |
| 7h | Dự trữ |
| 8h | Nhịp thời gian |
| 9h | Bàn phím |
| Ah | Dự trữ |
| Bh | Liên lạc nối tiếp (COM2) |
| Ch | Liên lạc nối tiếp (COM1) |
| Dh | Đĩa cứng |
| Eh | Đĩa mềm |
| Fh | Máy in song song |

C.2. Các ngắt của BIOS.

Ngắt 10: Video

Hàm 0h:

Chọn chế độ hiển thị.

Chọn chế độ hiển thị màn hình.

Vào: AH = 0

AL = kiểu (xem bảng 12.2)

Ra: Không.

Hàm 1h

Thay đổi kích thước con trỏ màn hình.

Chọn dòng quét bắt đầu và kết thúc của con trỏ.

Vào: AH = 1h

CH = dòng quét đầu

CL = dòng quét cuối

Ra: Không.

Hàm 2h

Dịch chuyển con trỏ.

Định vị con trỏ.

Vào: AH = 2h

DH = dòng

DL = cột

BH = trang

Ra: Không.

Hàm 3h

Lấy vị trí và kích thước tức thời của con trỏ.

Nhận vị trí tức thời và kích thước của con trỏ

Vào: AH = 3h

BH = trang

Ra: DH = dòng

DL = cột

CH = dòng quét đầu

CL = dòng quét cuối

Hàm 5h

Chọn trang hoạt động.

Vào: AH = 5h

AL = trang

DH = dòng

DL = cột

Ra: Không.

Hàm 6h

Cuốn màn hình hay cửa sổ lên.

Cuốn cả màn hình hay cửa sổ lên một số dòng xác định.

Vào: AH = 6h

AL = số dòng cuộn (AL = 0 có nghĩa là cuộn cả màn hình hay cửa sổ)

BH = thuộc tính của các dòng trống

CH, CL = dòng, cột góc trái trên của cửa sổ

DH, DL = dòng, cột góc phải dưới của cửa sổ

Ra: Không.

Hàm 7h

Cuốn cửa sổ và màn hình xuống.

Cuốn cả màn hình hay cửa sổ xuống một số dòng xác định.

Vào: AH = 7h

AL = số dòng cuộn (AL = 0 có nghĩa là cuộn cả màn hình hay cửa sổ)

BH = thuộc tính của các dòng trống

CH, CL = dòng, cột góc trái trên của cửa sổ

DH, DL = dòng, cột góc phải dưới của cửa sổ

Ra: Không.

Hàm 8h

Đọc ký tự và thuộc tính tại vị trí con trỏ.

Nhận ký tự ASCII tại vị trí con trỏ và thuộc tính của nó

Vào: AH = 8h

BH = trang

Ra: AH = thuộc tính

AL = ký tự

Hàm 9h

Hiển thị ký tự và thuộc tính tại vị trí con trỏ

Viết một ký tự ASCII và thuộc tính của nó tại vị trí con trỏ.

Vào: AH = 9h

BH = trang

AL = ký tự

CX = số lần viết ký tự

BL = thuộc tính (chế độ văn bản) hay màu (chế độ đồ họa)

Ra: Không.

Hàm Ah

Viết ký tự tại vị trí con trỏ

Viết một ký tự ASCII tại vị trí con trỏ. Ký tự có thuộc tính của ký tự trước đó ở vị trí này.

Vào: AH = OAh

BH = số hiệu trang

AL = mã ASCII của ký tự

CX = số lần viết ký tự

Ra: không

Hàm 0Bh

Thiết lập bảng màu, nền và viền.

Chọn bảng màu, màu nền và màu viền.

Vào: Chọn màu viền và màu nền

AH = 0Bh

BH = 0

BL = màu

Chọn bảng trộn màu (chế độ 4 màu 320x200)

AH = 0Bh

BH = 1

BL = bảng màu

Ra: không

Hàm 0Ch

Viết điểm ảnh

Vào: AH = 0Ch

AL = giá trị điểm ảnh

BH = trang

CX = cột

DX = hàng

Ra: không

Hàm 0Dh

Đọc điểm ảnh

Nhận một giá trị điểm ảnh.

Vào: AH = 0Dh

BH = trang

CX = cột

DX = hàng

Ra: AL = giá trị điểm ảnh

Hàm 0Eh

Viết ký tự trong chế độ đánh telex.

Viết một ký tự ASCII tại vị trí con trỏ sau đó dịch chuyển con trỏ đến vị trí tiếp theo.

Vào: AH = 0Eh

AL = mã ASCII của ký tự

BH = trang

BL = màu nổi (chế độ đồ họa)

Ra: không

Chú ý: không ấn định được thuộc tính của ký tự.

Hàm Fh

Lấy chế độ màn hình.

Nhận chế độ hiển thị hiện tại.

Vào: AH = 0Fh

Ra: AH = số cột màn hình

AL = chế độ hiển thị

BH = trang hoạt động

Hàm 10h, hàm con 10h:

Thiết lập thanh ghi màu.

Thiết lập các thanh ghi màu VGA.

Vào: AH = 10h

AL = 10h

BX = thanh ghi màu

CH = giá trị màu xanh lá cây

CL = giá trị màu xanh da trời

DH = giá trị màu đỏ

Ra: không

Hàm 10h, hàm con 12h:

Thiết lập khối thanh ghi màu.

Thiết lập nhóm các thanh ghi màu VGA

Vào: AH = 10h

AL = 12h

BX = thanh ghi màu thứ nhất

CX = số lượng thanh ghi màu.

ES:DX = địa chỉ segment:offset của bảng màu.

Ra: không

Chú ý: Bảng màu bao gồm các điểm nhập ba byte tương ứng với các giá trị màu đỏ, xanh lá cây và xanh da trời cho mỗi thanh ghi màu.

Hàm 10h, hàm con 15h:

Lấy thanh ghi màu.

Nhận các giá trị màu đỏ, xanh lá cây và xanh da trời của một thanh ghi màu VGA.

Vào: AH = 10h

AL = 15h

BX = thanh ghi màu

Ra: CH = giá trị màu xanh lá cây

CL = giá trị màu xanh da trời

DH = giá trị màu đỏ

Hàm 10, hàm con 17h:

Lấy khối thanh ghi màu.

Nhận các giá trị màu đỏ, xanh lá cây và xanh da trời trong một nhóm các thanh ghi màu VGA.

Vào: AH = 10h

AL = 17h

BX = thanh ghi màu thứ nhất

CX = số lượng thanh ghi màu

ES:DX = địa chỉ segment:offset của bộ đệm nhận bảng màu

Ra: ES:DX = địa chỉ segment:offset của bộ đệm

Chú ý: Bảng bao gồm các điểm nhập ba byte tương ứng với các giá trị màu đỏ, xanh lá cây và xanh da trời cho mỗi thanh ghi màu.

Ngắt 11h: lấy cấu hình thiết bị.

Nhận từ mã hoá danh sách thiết bị.

Vào: không

Ra: AX = từ mã hoá danh sách thiết bị.

(các bit 14-15 = số máy in được cài đặt,

13 = modem bên trong,

12 = bộ phổi ghép trò chơi,

9-11 = số cổng nối tiếp,

8 dự trữ,
6-7 = số ổ đĩa mềm,
4-5 = chế độ màn hình khởi tạo,
2-3 = kích thước RAM hệ thống với PC nguyên thuỷ,
2 dùng cho PS/2,
1 = bộ đồng xử lý toán học,
0 = đĩa mềm được cài đặt)

Ngắt 12h: Lấy kích thước bộ nhớ quy ước.

Trả về tổng kích thước của bộ nhớ quy ước.

Vào: không

Ra: AX = kích thước bộ nhớ (theo KB)

Ngắt 13h: Vào ra đĩa.

Hàm 2h: Đọc sector.

Đọc một hay nhiều sector

Vào: AH = 2h

AL = số sector

CH = xi-lanh

CL = sector

DH = đầu từ

DL = ổ đĩa (0-7Fh = đĩa mềm, 80h-ffh = đĩa cứng)

ES:BX = địa chỉ segment:offset của bộ đệm

Ra: Nếu thành công:

CF = xoá

AH = 0

AL = số sector được đọc

Nếu không thành công:

CF = thiết lập

AH = mã lỗi

Hàm 3h: Viết sector.

Viết một hay nhiều sector.

Vào: AH = 3h
AL = số sector
BX = thanh ghi màu thư nhất
CH = xi-lanh
CL = sector
DH = đầu từ
DL = ổ đĩa (0-7Fh = đĩa mềm, 80h-FFh = đĩa cứng)
ES:BX = địa chỉ segment:offset của bộ đệm

Ra: Nếu thành công:

CF = xoá
AH = 0
AL = số sector được viết

Nếu không thành công:

CF = thiết lập
AH = mã lỗi

Ngắt 15h: Ghi đọc băng từ và các đặc tính cao cấp của AT và PS/2.

Hàm 87h:

Chuyển khối bộ nhớ mở rộng.

Truyền số liệu giữa bộ nhớ mở rộng và bộ nhớ quy ước.

Vào: AH = 87h

CX = số word dịch chuyển
ES:SI = địa chỉ bảng mô tả toàn cục.

Ra: Nếu thành công:

CF = xoá
AH = 0
AL = số sector được dịch chuyển.

Nếu không thành công:

CF = thiết lập
AH = mã lỗi

Hàm 88h:

Lấy kích thước bộ nhớ mở rộng.

Nhận kích thước tổng cộng của bộ nhớ rộng.

Vào: AH = 88h

Ra: AX = kích thước bộ nhớ mở rộng (theo KB)

Ngắt 16h: Bàn phím

Hàm 0h

Đọc ký tự từ bàn phím

Vào: AH=0h

Ra: AH=keyboard scan code

AL = ký tự ASCII

Hàm 2h:

Lấy các cờ bàn phím.

Vào: AH = 2h

Ra: AL = các cờ

| Bit | Nếu được thiết lập |
|-----|---------------------------|
| 7 | Insert on |
| 6 | Caps Lock on |
| 5 | Num Lock on |
| 4 | Scroll Lock on |
| 3 | Phím Alt được nhấn |
| 2 | Phím Ctrl được nhấn |
| 1 | Phím Shift trái được nhấn |
| 0 | Phím Shift phải được nhấn |

Hàm 10h:

Đọc ký tự từ bàn phím bậc cao.

Vào: AH = 0h

Ra: AH = mã scan bàn phím

AL = ký tự ASCII

Chú ý: Có thể sử dụng hàm này để trả về mã scan của các phím điều khiển giống như F11 và F12.

Ngắt 17h: máy in

Hàm 0h:

Viết ký tự ra máy in.

Vào: AH = 0h

AL = ký tự

DX = số hiệu máy in.

Ra: AH = trạng thái

Bit Nếu được thiết lập.

7 máy in không bận

6 máy in chấp nhận

5 hết giấy

4 máy in được chọn

3 lỗi vào/ra

2 không sử dụng

1 không sử dụng

0 quá thời gian

C.3. Các ngắt của DOS.

Ngắt 21h

Hàm 0h: Kết thúc chương trình

Kết thúc việc thi hành một chương trình.

Vào: AH = 0h

CS = địa chỉ đoạn của PSP

Ra: không

Hàm 1h: vào từ bàn phím.

Đợi đọc một ký tự từ thiết bị vào chuẩn (trừ khi đã có một ký tự sẵn sàng), sau đó đưa ký tự tới thiết bị ra chuẩn và trả về mã ASCII trong DL.

Vào: AH =01h
Ra: AL = ký tự từ thiết bị vào chuẩn.

Hàm 2h: Hiển thị.

Đưa ký tự trong DL tới thiết bị ra chuẩn.

Vào: AH =02h
DL = ký tự
RA: không

Hàm 5h: In ra.

Đưa ký tự trong DL ra thiết bị in chuẩn.

Vào: AH =5h
DL = ký tự.
Ra: không

Hàm 9h: In chuỗi.

Đưa chuỗi ký tự ra tới thiết bị ra chuẩn.

Vào: AH =09h
DS:DX = con trỏ đến chuỗi ký tự kết thúc bằng '\$'
Ra: không

Hàm 2Ah: Lấy ngày tháng.

Trả về ngày trong tuần, tháng, năm và ngày trong tháng.

Vào: AH =2Ah
Ra: AL = ngày trong tuần (0 = C.nhật, 6 = T.bảy).
CX = năm (1980-2099)
DH = tháng (1-12)
DL = ngày (1-31)

Hàm 2Bh: Thiết lập ngày tháng.

Vào: AH =2Bh
CX = năm (1980-2099)
DH = tháng (1-12)
DL = ngày (1-31)
Ra: AL =00h, nếu ngày hợp lệ,
FFh, nếu ngày không hợp lệ.

Hàm 2Ch: Lấy thời gian.

Trả về thời gian: giờ, phút, giây và phần trăm của giây.

Vào: AH =2Ch
Ra: CH = giờ (0-23)

CL = phút (0-59)
DH = giây (0-59)
DL = phần trăm của giây (0-99)

Hàm 2Dh: Thiết lập thời gian.

Vào: AH = 2Dh
CH = giờ (0-23)
CL = phút (0-59)
DH = giây (0-59)
DL = phần trăm của giây (0-99)

Ra: AL = 00h nếu thời gian hợp lệ,
FFh nếu thời gian không hợp lệ.

Hàm 30h: Lấy số hiệu version của DOS.

Trả về số hiệu version của DOS.

Vào: AH = 30h
Ra: BX = 0000h
CX = 0000h
AL = số chính của version (trước dấu chấm)
AH = số phụ của version (sau dấu chấm)

Hàm 31h: Kết thúc chương trình và ở lại thường trú.

Kết thúc chương trình hiện tại và để dành ra một khối nhớ tính bằng paragraph.

Vào: AH = 31h
AL = mã trả về
DX = kích thước bộ nhớ theo paragraph.
Ra: không

Hàm 33h: Kiểm tra Ctrl-Break.

Thiết lập và lấy trạng thái BREAK (kiểm tra Ctrl-break)

Vào: AH = 33h
AL = 00h để lấy trạng thái hiện tại
01h để thiết lập trạng thái hiện tại
DL = 00h thiết lập trạng thái hiện tại OFF
= 01h thiết lập trạng thái hiện tại ON
Ra: DL = trạng thái hiện hành (00h=OFF, 01h=ON)

Hàm 35h: Lấy vector.

Nhận địa chỉ của một vector ngắt.

Vào: AH = 35h
AL = số hiệu ngắt

Ra: ES:BX = con trỏ trỏ đến thường trình quản lý ngắn.

Hàm 36: Lấy dung lượng còn trống trên đĩa.

Trả về dung lượng còn trống trên đĩa (số cluster chưa dùng, số clustor/đĩa, số byte/sector).

Vào: AH =36h
DL = ổ đĩa (0=mặc định, 1=A ...)

Ra: BX = số cluster khả dụng.
DX = số cluster/đĩa
CX = số byte/sector
AX = FFFFh nếu ổ đĩa trong DL không hợp lệ,
các trường hợp khác bằng số sector trong một cluster

Hàm 39h: Tạo lập thư mục con (MKDIR).

Tạo lập thư mục con xác định.

Vào: AH =39h
DS:DX = con trỏ đến một chuỗi ASCIIZ.
Ra: AX = mã lỗi nếu cờ carry được lập.

Hàm 3Ah: Xoá thư mục con (RMDIR).

Xoá thư mục con cho trước.

Vào: AH =3Ah
DS:DX = con trỏ đến một chuỗi ASCIIZ.
Ra: AX = mã lỗi nếu cờ nhớ được lập

Hàm 3Bh: Thay đổi thư mục hiện tại (CHDIR).

Đổi thư mục cho trước thành thư mục hiện tại.

Vào: AH =3Bh
DS:DX = con trỏ đến một chuỗi ASCIIZ.
Ra: AX = mã lỗi nếu cờ carry được lập.

Hàm 3Ch: Tạo lập một file (CREAT).

Tạo ra một file mới hoặc cắt bỏ file cũ cho có độ dài bằng 0 để chuẩn bị viết.

Vào: AH =3Ch
DS:DX = con trỏ trỏ đến một chuỗi ASCIIZ
CX = thuộc tính của file
Ra: AX = mã lỗi nếu cờ nhớ được lập.
Thẻ 16 bit nếu cờ nhớ không được lập.

Hàm 3Dh: Mở một file.

Mở một file cho trước.

- Vào: AH =3Dh
 DS:DX = con trỏ trỏ đến một tên đường dẫn ASCIIIZ.
 AL = mã truy nhập
- Ra: AX =mã lỗi nếu cờ nhỡ được lập.
 Thẻ16 bit nếu cờ nhỡ không được lập.

Hàm 3Eh: Đóng một thẻ file.

Đóng một thẻ file cho trước.

- Vào: AH =3Eh
 BX = thẻ file trả về bởi thao tác mở hay tạo lập file
- Ra: AX = mã lỗi nếu cờ nhỡ được lập.
 không có nghĩa nếu cờ nhỡ không được lập

Hàm 3Fh: Đọc từ một file hay thiết bị.

Chuyển một số cho trước các byte từ một file vào bộ đệm cho trước.

- Vào: AH =3Fh
 BX =thẻ file
 DS:DX = địa chỉ bộ đệm
 CX = số byte đọc.
- Ra: AX = số byte đọc được
 mã lỗi nếu cờ nhỡ được lập.

Hàm 40h: Viết vào một file hay thiết bị.

Chuyển một số cho trước các byte từ một bộ đệm vào một file cho trước.

- Vào: AH =40h
 BX =thẻ file
 DS:DX = địa chỉ của dữ liệu để viết
 CX = số byte được viết.
- Ra: AX = số byte viết
 mã lỗi nếu cờ nhỡ được lập.

Hàm 41h: Xoá một file trong một thư mục nhất định (UNLINK).

Xoá một điểm nhập thư mục tương ứng với tên file

- Vào: AH =41h
 DS:DX = địa chỉ của một ASCIIIZ.
- Ra: AX = mã lỗi nếu cờ nhỡ được lập.
 không có nghĩa nếu cờ nhỡ không được lập.

Hàm 42h: Di chuyển con trỏ đọc/viết file (LSEEK).

Di chuyển con trỏ đọc/viết theo phương thức cho trước

Vào: AH =42h
DS:DX = khoảng cách (offset) di chuyển theo byte
AL = phương thức di chuyển (0,1, 2)
BX = thẻ file
Ra: AX = mã lỗi nếu cờ nháy được lập.
DX:AX = vị trí con trỏ mới nếu cờ nháy không được lập.

Hàm 47h: Lấy thư mục hiện thời.

Đưa vào tên đường dẫn đầy đủ (bắt đầu từ thư mục gốc) của thư mục hiện tại trong ổ đĩa cho trước vào vùng có địa chỉ DS:SI.

Vào: AH =47h
DS:SI = con trỏ trả về vùng nhớ 64 byte của người sử dụng.
DL = số hiệu ổ đĩa (0=mặc định, 1=A, ...)
Ra: DS:SI = tên đường dẫn đầy đủ từ gốc nếu cờ nháy không được lập
AX = mã lỗi nếu cờ nháy được lập.

Hàm 48h: Cấp phát bộ nhớ.

Cấp phát bộ nhớ theo số paragrph yêu cầu.

Vào: AH =48h
BX = số paragraph nhớ yêu cầu.
Ra: AX:0 = địa chỉ khôi nhớ cấp phát.
AX = mã lỗi nếu cờ nháy được lập.
BX = kích thước của khôi lớn nhất của bộ nhớ khả dụng nếu sự cấp phát không thành công.

Hàm 49: Giải phóng bộ nhớ đã cấp phát.

Giải phóng bộ nhớ cấp phát cho trước.

Vào: AH =49h
ES = segment của khôi nhớ được giải phóng.
Ra: AX = mã lỗi nếu cờ nháy được lập.
không có nghĩa nếu cờ nháy không được lập.

Hàm 4Ch: Kết thúc chương trình (EXIT)

Kết thúc chương trình hiện tại vào trả điều khiển cho chương trình gọi nó.

Vào: AH = 4Ch
AL = mã trả về.
Ra: không

Ngắt 25h: Đọc tuyệt đối đĩa

Vào: AL = số hiệu ổ đĩa
CX = số sector đọc

DX = số hiệu sector logic bắt đầu.
DS:BX = địa chỉ chuyển đến.
Ra:
 Nếu thành công CF = 0
 Nếu không thành công CF = 1 và AX chứa mã lỗi.

Ngắt 26h: Ghi tuyệt đối đĩa.

Vào: AL = số hiệu ổ đĩa
 CX = số sector ghi
 DX = số hiệu sector logic bắt đầu.
 DS:BX = địa chỉ chuyển đi.
Ra:
 Nếu thành công CF = 0
 Nếu không thành công CF = 1 và AX chứa mã lỗi.

Ngắt 27h: Kết thúc chương trình và ở lại thường trú.

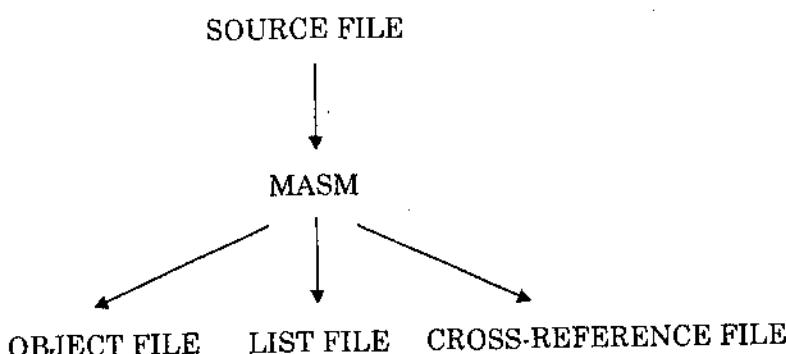
Vào: DX = offset bắt đầu của vùng nhớ được giải phóng(không giữ lại thường trú), segment là segment của PSP.
Ra: không

Phục lục D

CÁC TÙY CHỌN CỦA MASM VÀ LINK

D.1. MASM

Chương trình biên dịch MASM chuyển một file nguồn viết bằng Hợp ngữ thành file đối tượng ngôn ngữ máy. Nó tạo ra ba file như sau:



File OBJ (object file: tệp đối tượng, tệp vật thể) chứa bản dịch mã máy của các mã lệnh trong chương trình nguồn Assembly và các thông tin cần thiết để tạo nên một file chương trình.

File list (list file) là một file văn bản đưa ra các mã lệnh assembly và mã máy tương ứng, danh sách các tên dùng trong chương trình, các thông báo lỗi và các số liệu thống kê khác. File này rất có ích khi gỡ rối chương trình.

File tham khảo chéo (cross-reference file) liệt kê các tên sử dụng trong chương trình và các dòng mà chúng ta xuất hiện. Nhờ nó chúng ta dễ dàng theo dõi các chương trình lớn hơn. Dạng nguyên thuỷ của file này rất khó theo dõi; chúng ta có thể dùng chương trình tiện ích CREF để đổi nó sang dạng dễ đọc hơn.

Dòng lệnh MASM.

Đối với MASM version 5.0, dạng dòng lệnh phổ biến nhất là:

MASM options source_file,object_file,list_file,cross_ref_file

MASM 4.0 cũng có dòng lệnh tương tự ngoại trừ các tùy chọn (options) được đặt ở cuối cùng.

Phần mở rộng mặc định cho file Object là .OBJ, cho file list là .LST và file tham khảo chéo là .CRF.

Ví dụ, giả sử ta có MASM trong đĩa C, file nguồn FIRST.ASM trong đĩa A và C là ổ đĩa hiện hành. Để tạo ra file đối tượng FIRST.OBJ, file list FIRST.LST và file tham khảo chéo FIRST.CRF trên đĩa A ta đánh vào:

```
C>MASM A:FIRST.ASM,A:FIRST.OBJ,A:FIRST.LST,A:FIRST.CRF
```

Một cách ngắn gọn hơn để có kết quả tương tự là:

```
C>MASM A:FIRST,A:,A:,A:
```

Một dấu chấm phẩy thay cho dấu phẩy trong dòng lệnh MASM báo cho chương trình biên dịch không tạo ra các file tiếp sau nữa. Ví dụ, nếu ta đánh vào:

```
C>MASM A:FIRST,A:;
```

MASM sẽ chỉ tạo ra file FIRST.OBJ. Còn nếu đánh vào:

```
C>MASM A:FIRST,A:,A:,
```

Chúng ta sẽ nhận được các file FIRST.OBJ và FIRST.LST nhưng FIRST.CRF thì không.

Cũng có thể để MASM nhắc vào các file bạn muốn. Ví dụ bạn chỉ cần tạo ra các file .OBJ và .CRF:

```

C>MASM A:FIRST

Microsoft (R) Macro Assembler Version 5.0
Copyright (c) Microsoft Corp 1981, 1988. All rights reserved.

Object filename [ FIRST.OBJ] : A:<Enter>
Source listing [ NUL.LST] : <Enter>
Cross-reference [ NUL.CRF] : A:FIRST <Enter>

50140 + 234323 Byte symble space free
          0 Warning Errors
          0 Severe Errors

```

Câu trả lời đầu tiên có nghĩa là chúng ta chấp nhận tên FIRST.OBJ cho file đối tượng. Câu trả lời thứ hai có nghĩa chúng ta không muốn có một file list (NUL có nghĩa là không tạo file). Câu trả lời thứ ba có nghĩa chúng ta muốn một file tham khảo chéo có tên FIRST.CRF.

Các tùy chọn.

Các tùy chọn của MASM điều khiển các thao tác của chương trình biên dịch và tạo dạng các file tạo thành. Bảng D.1 đưa ra danh sách của một số tùy chọn thường dùng. Nếu muốn có danh sách đầy đủ hơn, bạn hãy xem cuốn Microsoft Programer's Guide.

Ta có thể xác định vài tùy chọn trong cùng một dòng lệnh. Ví dụ:

```
C>MASM /D /W2 /Z /ZI FIRST;
```

Bảng D.1. Vài tùy chọn của MASM.

| Tùy chọn | Ý nghĩa |
|----------|--|
| /A | Xắp xếp các đoạn nguồn theo thứ tự an_pha_B. |
| /C | Tạo lập một file tham khảo chéo |

| | |
|-----------|--|
| /D | Tạo lập danh sách trong lần duyệt thứ nhất |
| /ML | Phân biệt chữ hoa và chữ thường trong tên |
| /R | Chấp nhận các lệnh dấu phẩy động 8087 |
| /S | Sắp các đoạn nguồn theo thứ tự nguyên thuỷ |
| /W{0 1 2} | Thiết lập mức độ hiển thị lỗi (mặc định = 0): 0 = các dòng lệnh không hợp lệ. 1 = các dòng lệnh không rõ nghĩa 2 = các dòng lệnh có thể tạo ra các mã lệnh không có hiệu lực. |
| /Z | Hiển thị các dòng có lỗi |
| /Z{ | Viết các thông tin về các ký hiệu dùng trong chương trình nguồn vào file đối tượng (dùng cho CODEVIEW). |

Một ví dụ sử dụng MASM.

Để thấy được MASM tạo ra các file có dạng như thế nào, chúng ta sẽ biên dịch chương trình hoán chuyển nội dung của hai từ nhớ sau đây:

Program listing PGMD_1.ASM

```

TITLE PGMD_1:SWAP WORDS

.MODE      SMALL
.STACK 100h
.DATA
WORD1 DW    10
WORD2 DW    20
.CODE
MAIN PROC
        MOV AX, @DATA
        MOV DS, AX
        MOV AX, WORD1
        XCHG AX, WORD2
        MOV WORD1, AX
        MOV AH, 4Ch
        INT 21H
        ENDP
END MAIN

```

C>MASM A:PGMD_1,A:,A:,A:

Microsoft (R) Macro Assembler Version 5.0
 Copyright (c) Microsoft Corp 1981, 1988. All rights reserved.

```
47358 + 390893 Byte symble space free  
0 Warning Errors  
0 Severe Errors
```

File list được chỉ ra trong hình D.1

```
C>TYPE A:PGMD_1.LST
```

Đọc theo lề trái của file .LST là các số hiệu dòng. Cột tiếp theo viết địa chỉ offset (dạng hex) tương ứng với các đoạn ngắn xếp, dữ liệu và đoạn lệnh. Sau đó là mã máy (dạng hex) của các lệnh.

Biên dịch hai lần và bảng ký hiệu.

MASM duyệt file nguồn hai lần. Trong lần duyệt đầu, nó kiểm tra các lỗi cú pháp và tạo lập một bảng ký hiệu chứa các tên và các vị trí tương đối theo của chúng trong một đoạn. Để theo dõi các vị trí, MASM sử dụng một bộ đếm định vị (location counter). Bộ đếm định vị được thiết lập bằng 0 khi bắt đầu một đoạn. Khi gặp một lệnh, bộ đếm định vị được tăng lên một số bằng số byte mã máy của lệnh đó. Khi gặp một tên, MASM đưa nó kèm theo giá trị của bộ đếm định vị vào trong bảng ký hiệu. Bảng ký hiệu xuất hiện ở gần cuối file .LST. Trong ví dụ nêu trên, các ký hiệu là MAIN, WORD1 và WORD2. Tuỳ chọn /D của MASM tạo ra file .LST chứa các thông báo lỗi của lần duyệt thứ nhất. Lần duyệt thứ hai sẽ xác định chúng có phải là các lỗi thực sự hay không.

Trong lần duyệt thứ hai, MASM hoàn thành việc soát lỗi và chuyển các lệnh sang dạng mã máy trừ các lệnh tham trả các tên xuất hiện trong các modul đối tượng khác. File .LST cũng được tạo lập xong sau lần duyệt này.

Lý do mà MASM cần tới hai lần duyệt để biên dịch một chương trình là các lệnh có thể tham trả các tên xuất hiện ở sau trong chương trình nguồn. Các lệnh chỉ có thể chuyển sang dạng mã máy được sau khi vị trí tương đối của chúng được xác định trong bảng ký hiệu.

File đối tượng (PGMD.OBJ) mà MASM tạo nên không thể thi hành được. Địa chỉ cuối cùng của các biến cần phải được xác định bởi chương trình LINK (trình bày ở sau). Trong file .LST, các địa chỉ này được đánh dấu bằng chữ "R" (relocatable) (các dòng 9, 10, 11, 12, 13).

Hình D.1. PGMD.LST

Microsoft (R) Macro Assemller Version 5.10 9/6/91 00:43:35
PGMD_1: SWAP WORDS Page-1

```
1           TITLE PGMD_1:SWAP WORDS
2           .MODE SMALL
3           .STACK    100h
4           .DATA
5 0000 000A          WORD1 DW    10
6 0002 0014          WORD2 DW    20
7           .CODE
8 0000          MAIN PROC
9 0000 B8 - R        MOV AX, @DATA
10 0003 8E D8         MOV DS, AX
11 0005 A1 0000 R      MOV AX, WORD1
12 0008 87 06 0002 R     XCHG AX, WORD2
13 000C A3 0000 R      MOV WORD1, AX
14 000F B4 4C         MOV AH, 4Ch
15 0011 CD 21         INT 21H
16 0013          MAIN ENDP
17          END MAIN
```

Microsoft (R) Macro Assemller Version 5.10 9/6/91 00:43:35
PGMD_1: SWAP WORDS Symbols-1

Segment and Groups:

| Name | Length | Align | CombineClass |
|-------------|--------|-------|---------------|
| DGROUP..... | GROUP | | |
| _DATA..... | 0004 | WORD | PUBLIC 'DATA' |
| STACK..... | 0100 | PARA | STACK 'STACK' |
| _TEXT..... | 0013 | WORD | PUBLIC 'CODE' |

Symbols:

| Name | Type | Value | Attr |
|----------------|--------|-------|-------------------|
| MAIN..... | N PROC | 0000 | _TEXT Length=0013 |
| WORD1..... | L WORD | 0000 | _DATA |
| WORD2..... | L WORD | 0002 | _DATA |
| @CODE..... | TEXT | | _TEXT |
| @CODESIZE..... | TEXT | 0 | |
| @CPU..... | TEXT | 0101h | |
| @DATASIZE..... | TEXT | 0 | |
| @FILENAME..... | TEXT | | PGMD_1 |

| | | |
|---|------|-----|
| @VERSION..... | TEXT | 510 |
| 17 Source Lines | | |
| 17 Total Lines | | |
| 20 Symbols | | |
| 47358 + 390893 Bytes symbols space free | | |
| 0 Warning Errors | | |
| 0 severe Errors | | |

File tham khảo chéo.

File tham khảo chéo (cross-reference file) mà ở đây là PGMD_1.CRF chứa thông tin của các tên: nơi chúng được định nghĩa và các dòng mà chúng xuất hiện trong file .LST. File CRF không thể hiển thị trên màn hình; chương trình CREF trong đĩa DOS đổi nó thành file .REF có dạng ASCII:

```
C>CREF A:PGMD_1;  
  
Microsoft (R) Cross-Reference Utility Version 5.0  
Copyright (c) Microsoft Corp 1981, 1987. All rights reserved.  
  
11 Symbols
```

Ta thu được file PGMD_1.REF có thể hiển thị trên màn hình bằng lệnh type của DOS (hình D.2).

```
C>TYPE A:PGMD_1.REF
```

Hình D.2 PGMD_1.REF

```
Microsoft Cross-Reference Version 5.10 Fri Sep 06 01:33:52 1991  
PGMD_1:SWAP WORDS  
Symbols Cross Reference (# definition, +modification) Cref-1  
  
@CPU. . . . . . . . . . . 1#  
@VERSION. . . . . . . . . . 1#  
  
CODE. . . . . . . . . . . 7  
  
DATA. . . . . . . . . . . 4  
DGROUP. . . . . . . . . . 9
```

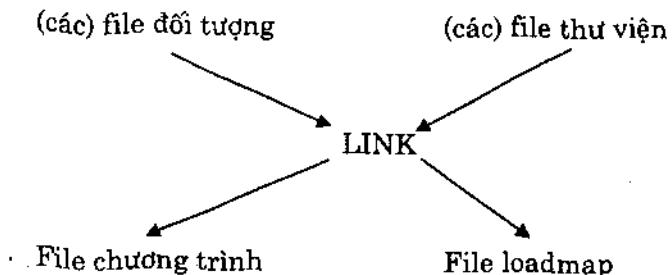
| | | | |
|--------------------------------|----|-----|-----|
| MAIN. | 8# | 16 | 17 |
| STACK | 3 | #3 | |
| WORD1 | 5# | 11 | 13+ |
| WORD2 | 6# | 12+ | |
| <u>_DATA</u> | 4# | | |
| <u>_TEXT</u> | 7# | | |

11 Symbols

D.2. LINK.

Chương trình LINK có nhiệm vụ liên kết các file đối tượng (và có thể cả các file thư viện) thành một file khả thi duy nhất. Muốn như vậy nó phải tham khảo được các tên dùng trong modul này nhưng lại được định nghĩa trong một modul khác. Cơ chế của việc làm này đã được giải thích trong chương 14. Ta vẫn cần phải sử dụng chương trình LINK thậm chí nếu chỉ có một file đối tượng.

Đầu vào chương trình LINK là một hay nhiều file đối tượng và thư viện, đầu ra của nó là file chương trình và có thể một file loadmap như chỉ ra sau đây:



File chương trình là một chương trình khả thi ngôn ngữ máy. File loadmap chứa kích thước và các vị trí tương đối của các đoạn chương trình.

Dòng lệnh LINK.

Đối với LINK 5.0, dòng lệnh chung nhất có dạng:

LINK options object_file_list run_file loadmap_file Library_list

Tuỳ chọn duy nhất mà bạn hay sử dụng đó là /CO, nó sẽ ghép thêm các thông tin phụ cho chương trình CODEVIEW.

Object_file_list là danh sách các file đối tượng cần liên kết. Nó bắt đầu bằng tên của file đối tượng chứa chương trình chính. Các file đối tượng khác thường chứa các thủ tục được chương trình chính gọi hay gọi lẫn nhau. Các tên file được ngăn cách bằng khoảng trắng hay dấu "+".

File chương trình có phần mở rộng là .EXE. Nó là một file khả thi trừ khi chương trình có dạng .COM; trong trường hợp này cần phải có thêm một bước nữa để tạo nên một chương trình thi hành được. Các chương trình .COM đã được thảo luận trong chương 14.

Library_list chứa các file thư viện (ngăn cách bằng dấu trắng hay "+") nếu có. Các file thư viện hay có phần mở rộng là .LIB, chúng thường chứa các thủ tục chuẩn dùng cho nhiều chương trình như các thủ tục phục vụ vào/ra. Chúng ta đã có một ví dụ về file thư viện trong chương 14.

Ví dụ. Giả sử ta có LINK trên đĩa C và các file cần liên kết trên đĩa A. File đối tượng chính là FIRST.OBJ, các file đối tượng khác gồm có SECOND.OBJ và THIRD.OBJ. Để tạo ra file chương trình FIRST.EXE và file loadmap FIRST.MAP, chúng ta đánh vào:

```
C>LINK A:FIRST+SECOND+THIRD,A:FIRST,A:FIRST;
```

hoặc chỉ cần:

```
C>LINK A:FIRST+SECOND+THIRD,A:,A:;
```

Dấu chấm phẩy ở cuối có nghĩa là không có các file thư viện. Giống như MASM, ta có thể chạy LINK trong chế độ hỏi đáp:

```
C>LINK FIRST+SECOND+THIRD
```

```
Microsoft (R) Overlay linker Version 3.64  
Copyright (c) Microsoft Corp 1981-1988. All rights reserved.
```

```
Run File [ FIRST.EXE] : <Enter>  
List File [ NUL.MAP] : A:first <Enter>  
Libraries [ .LIB] : <Enter>
```

Câu trả lời đầu tiên có nghĩa là ta chấp nhận tên FIRST.EXE cho file chương trình. Câu trả lời thứ hai có nghĩa chúng ta muốn file loadmap có tên FIRST.MAP. Câu trả lời thứ ba có nghĩa không có file thư viện nào.

Một ví dụ sử dụng chương trình LINK.

Chúng ta hãy liên kết chương trình PGMD_1 ở trên:

```
C>LINK A:PGMD_1,A:,A:;
```

```
Microsoft (R) Overlay linker Version 3.64
Copyright (c) Microsoft Corp 1981, 1988. All rights reserved.
C>
```

Sau đây là file loadmap:

```
C>TYPE A:PGMD_1:MAP
```

| Start | Stop | Length | Name | Class |
|--------|--------|--------|-------|--------|
| 00000H | 00012H | 00013H | _TEXT | CODE |
| 00014H | 00017H | 00004H | _DATA | DATA |
| 00020H | 0011FH | 00100H | STACK | STACK |
| origin | | | | Group |
| 0001:0 | | | | DGROUP |

Program entry point at 0000:0000

File cho biết kích thước và vị trí tương đối của các đoạn chương trình.

Phụ lục E

CÁC TRÌNH GỠ RỐI DEBUG VÀ CODEVIEW

E.1 Lời giới thiệu

Phụ lục này trình bày về các trình gỡ rối DEBUG và CODEVIEW. Chương trình DEBUG có kèm trong đĩa DOS còn trình CODEVIEW kèm theo trình biên dịch Microsoft Assembler version 5.0 hay mới hơn. Trình DEBUG khá cũ nhưng nó lại là một trình gỡ rối khá tiện ích với kích thước nhỏ và tập lệnh dễ học. CODEVIEW là một chương trình phức tạp có thể gỡ rối các chương trình bằng ngôn ngữ PASCAL, C, FORTRAN, BASIC hay chương trình bằng hợp ngữ. Người sử dụng có thể đồng thời quan sát chương trình nguồn, các thanh ghi, các cờ và các biến được chọn.

E.2 Chương trình DEBUG.

Vì hầu hết các lệnh của DEBUG đều có thể sử dụng trong CODEVIEW, các bạn nên đọc hết cả phần này cho dù cuối cùng chúng ta có thể chỉ dùng CODEVIEW. Bảng E.1 tổng kết các lệnh tiện dụng nhất của DEBUG, các bạn có thể tham khảo thêm tài liệu về DOS để có tập lệnh đầy đủ và chi tiết.

Biểu diễn các lệnh của DEBUG.

Để biểu diễn các lệnh của DEBUG chúng ta sẽ sử dụng chương trình PGM14_1.ASM, đó là chương trình hiển thị lời chào "HELLO!" trên màn hình.

Chương trình nguồn PGM14_1.ASM

```
TITLE PGM14_1:           HELLO
.MODEL SMALL
.STACK 100H
.DATA
MSG    DB      'HELLO!', '$'
.CODE
MAIN   PROC
```

```

; khởi tạo DS
    MOV AX, @DATA
    MOV DS, AX ; khởi tạo DS
; hiển thị lời chào
    LEA DX, MSG ; lấy thông báo
    MOV AH, 09 ; hàm hiển thị
    INT 21H ; hiển thị thông báo
; DOS exit
    MOV AH, 4CH
    INT 21H ; trả về DOS
MAIN ENDP
END MAIN

```

Sau khi hợp dịch và liên kết chương trình chúng ta đưa nó vào trong môi trường DEBUG (các chữ do người sử dụng đánh vào được in đậm).

C>DEBUG PGM14_1.EXE

DEBUG xuất hiện với dấu nhắc lệnh ".". Để xem các thanh ghi chúng ta đánh "R"

R
AX=0000 BX=0000 CX=0017 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=0EFB ES=0EFB SS=0F0B CS=0F1C IP=0000 NV UP DI PL NZ NA PO NC
0F1C:0000 B81B0F MOV AX,0F1B

Màn hình trình bày nội dung của các thanh ghi dưới dạng hex. Tại dòng thứ 3 chứa địa chỉ segment:offset của lệnh đầu tiên trong chương trình kèm theo nó là lệnh ở dạng mã máy và dạng hợp ngữ. Các cặp chữ ở cuối dòng thứ 2 là trạng thái hiện thời của các cờ (bao gồm cả cờ trạng thái và cờ điều khiển). Dưới đây là các ký hiệu được sử dụng trong DEBUG để hiển thị trạng thái của các cờ:

| Cờ | Ký hiệu xóa (0) | Ký hiệu thiết lập (1) |
|--------------------------------|--------------------|--------------------------|
| Cờ tràn (Overflow Flag) | NV | OV |
| Cờ định hướng (Direction Flag) | UP | DN |
| Cờ ngắt (Interrupt Flag) | DI | EI |
| Cờ dấu (Sign Flag) | PL | NG |
| Cờ Zero (Zero Flag) | NZ | ZR |
| Cờ nhỡ phụ (Auxiliary Flag) | NA | AC |
| Cờ chẵn lẻ (Parity Flag) | PO | PE |
| Cờ nhỡ (Carry Flag) | NC | CY |

Bảng E.1 Các lệnh của DEBUG

Các tham số tùy chọn được viết trong cặp ngoặc nhọn. Tất cả các hằng số được biểu diễn dưới dạng hex.

| Lệnh | Công việc |
|---|--|
| D{start{end}}{range} | Hiển thị các byte dưới dạng hex |
| Ví dụ: | |
| D 100 chỉ | Hiển thị 80h byte bắt đầu tại địa DS:100h |
| D CS:100 120 | Hiển thị các byte từ CS:100h đến CS:120h |
| D | Hiển thị 80h byte bắt đầu tại DS:last+1 trong đó last là byte mới được hiển thị. |
| E start{list} | Nhập số liệu trong list bắt đầu từ start. |
| Ví dụ: | |
| E DS:0 A B C | Nhập các byte Ah, Bh, Ch vào các vị trí DS:0, DS:1, DS:2. |
| E ES:100 1 2 'ABC' | Nhập 1h vào ES:100h, 2 vào ES:101h 41h vào ES:102h, 42h vào ES:103h, 43h vào ES:104h. |
| E 25 | Nhập các byte có lần lượt từ DS:25. Dùng phím bước trống để chuyển sang byte tiếp theo, phím ENTER để kết thúc. |
| G(=start){addr1 addr2...addrn} | Thực hiện bắt đầu từ start đến các điểm dừng tại addr1 addr2...addrn. |
| Ví dụ: | |
| G | Thực hiện lệnh từ CS:IP cho đến cuối chương trình. |
| G=100 | Thực hiện từ CS:100h cho đến cuối chương trình. |
| G 100 200 300 | Thực hiện từ CS:IP cho đến khi gặp điểm dừng đầu tiên trong số CS:100h, CS:200h hay CS:300h. |
| G=100 150 | Thực hiện bắt đầu từ CS:100h cho đến CS:150h. |
| I address{drive start_sector end_sector} | nap các sector tuyệt đối của đĩa hay chương trình được đặt tên (xem lệnh N). Ổ đĩa được xác định bằng các chữ số 0 cho ổ A, 1 cho ổ B ... |
| Ví dụ: | |

| | |
|-----------------------------|--|
| L DS:100 0 C 1B | Nạp các sector từ Ch cho đến 1Bh trong ổ đĩa A tại địa chỉ DS:100h. |
| L 8FD0:0 1 2A 3B | Nạp các sector từ 2A đến 3B trong ổ đĩa B tại địa chỉ 8FD0h. |
| L DS:100 | Nạp file được đặt tên tại địa chỉ DS:100h. |
| N filename | Đặt tên hiện thời cho các file dùng trong lệnh L và W. |
| Ví dụ: N myfile | Đặt tên file để đọc ghi là myfile. |
| Q | Thoát khỏi DEBUG và trở về DOS. |
| R{register} | Hiển thị/Thay đổi nội dung thanh ghi |
| Ví dụ: R | Hiển thị nội dung thanh ghi và các cờ. |
| RAX | Hiển thị nội dung thanh ghi AX và thay đổi nếu muốn. |
| T{=start}{value} | Duyệt "value" lệnh bắt đầu từ start. |
| Ví dụ: T | Duyệt lệnh tại CS:IP |
| T =100 | Duyệt lệnh tại CS:100h |
| T =100 5 | Duyệt 5 lệnh bắt đầu tại CS:100h |
| T 4 | Duyệt 4 lệnh bắt đầu từ CS:IP |
| U{start(end)}{range} | Dịch ngược dữ liệu thành dạng lệnh. |
| Ví dụ: U 100 | Dịch ngược khoảng 32 bte từ CS:100h |
| U CS:100..110 | Dịch ngược dữ liệu từ CS:100h đến CS:110h. |
| U 200 L 20 | Dịch ngược 20 lệnh bắt đầu từ CS:200h |
| U | Dịch ngược khoảng 32 byte bắt đầu từ CS:last trong đó last là byte cuối cùng đã được dịch ngược. |
| W{start} | Viết BX:CX byte vào file(xem lệnh N). |
| Ví dụ: W 100 | Viết BX:CX byte vào file tại vị trí CS:100h. |

Để đổi nội dung của một thanh ghi - ví dụ DX với 1ABCh chúng ta đánh:

```
-RDX  
DX 0000  
:1ABC
```

DEBUG đáp lại bằng việc hiển thị nội dung hiện thời của DX, sau đó nó hiển thị dấu 2 chấm và đợi chúng ta đánh vào nội dung mới của DX. Chúng ta đánh vào 1ABC và đánh phím ENTER (DEBUG ngầm định rằng tất cả các số người sử dụng đánh vào đều biểu diễn dưới dạng hex nên ở đây không cần phải đánh thêm chữ "h"). Để giữ nguyên nội dung của DX chúng ta chỉ việc đánh ENTER ngay sau dấu hai chấm. Để kiểm chứng lại việc thay đổi chúng ta có thể hiển thị lại một lần nữa nội dung của các thanh ghi:

```
-R  
AX=0000 BX=0000 CX=0121 DX=1ABC SP=0100 BP=0000 SI=0000 DI=0000  
DS=0EFB ES=0EFB SS=0F0B CS=0F1C IP=0000 NV UP DI PL NZ NA PO NC  
0F1C:0000 B81B0F MOV AX,0F1B
```

```
-T  
AX=0F1B BX=0000 CX=0121 DX=1ABC SP=0100 BP=0000 SI=0000 DI=0000  
DS=0EFB ES=0EFB SS=0F0B CS=0F1C IP=0003 NV UP DI PL NZ NA PO NC  
0F1C:0003 8ED8 MOV DS,AX  
-T  
AX=0F1B BX=0000 CX=0121 DX=0002 SP=0100 BP=0000 SI=0000 DI=0000  
DS=0F1B ES=0EFB SS=0F0B CS=0F1C IP=0009 NV UP DI PL NZ NA PO NC  
0F1C:0009 B409 MOV AH,09
```

Chú ý rằng dường như DEBUG bỏ qua lệnh LEA DX,MSG. Thực tế lệnh đó đã được thực hiện, chúng ta có thể nói như vậy là vì DX đã có nội dung mới. Đôi khi DEBUG thực hiện một lệnh mà không tạm dừng lại để hiển thị nội dung của các thanh ghi.

```
-T  
AX=091B BX=0000 CX=0121 DX=0002 SP=0100 BP=0000 SI=0000 DI=0000  
DS=0F1B ES=0EFB SS=0F0B CS=0F1C IP=000B NV UP DI PL NZ NA PO NC  
0F1C:000B CD21 INT 21
```

Nếu bây giờ chúng ta lại đánh T lần nữa thì DEBUG sẽ tiến hành từng lệnh trong chương trình phục vụ ngắt 21h mà đó là điều chúng ta không mong muốn.

Từ lần hiển thị thanh ghi cuối cùng chúng ta thấy rằng INT 21h là một lệnh 2 byte. Do giá trị hiện nay của IP là 000Bh, lệnh tiếp theo nằm tại địa chỉ 000Dh, và chúng ta có thể tạo điểm ngắt tại đó:

-GD

HELLO!

AX=0924 BX=0000 CX=0121 DX=0002 SP=0100 BP=0000 SI=0000 DI=0000
DS=0F1B ES=0EFB SS=0F0B CS=0F1C IP=000D NV UP DI PL NZ NA PO NC
0F1C:000D B44C MOV AH, 4C

Hàm 09 của ngắt 21h hiển thị 'HELLO!' và việc thực hiện dừng lại ở điểm gãy 000Dh. Để kết thúc việc thực hiện chương trình chúng ta chỉ việc đánh 'G':

-G

Program terminate normally.

Dòng nhắc này thông báo cho ta biết rằng chương trình đã hoàn thành. Chương trình phải được nạp để có thể chạy lại do đó chúng ta hãy thoát khỏi DEBUG.

-Q
C>

Để thử lệnh U hãy quay lại DEBUG và dùng nó để liệt kê chương trình của chúng ta:

C>DEBUG PGM14_1.EXE

-U

| | |
|--------------------|--------------------|
| 0F1C:0000 B81B0F | MOV AX, 0F1B |
| 0F1C:0003 8ED8 | MOV DS, AX |
| 0F1C:0005 8D160200 | LEA DX, [0002] |
| 0F1C:0009 B409 | MOV AH, 09 |
| 0F1C:000B CD21 | INT 21 |
| 0F1C:000D B44C | MOV AH, 4C |
| 0F1C:000F CD21 | INT 21 |
| 0F1C:0011 015BE8 | ADD [BP+DI-18], BX |
| 0F1C:0014 3BEE | CMP BP, SI |
| 0F1C:0016 E88AF3 | CALL F3A3 |
| 0F1C:0019 E97E08 | JMP 089A |
| 0F1C:001C 8D1E8E09 | LEA BX, [098E] |

DEBUG đã dịch ngược khoảng 32 bytes; nó đã dịch nội dung của các byte này như là các lệnh. Chương trình kết thúc tại 000Fh và phần còn lại là các ký tự đi kèm theo mã assembly do chương trình dịch DEBUG tạo ra. Để hiển thị chương trình đánh:

- U O F

| | |
|------------------|--------------|
| 0F1C:0000 B81B0F | MOV AX, 0F1B |
| 0F1C:0003 8ED8 | MOV DS, AX |

```

OF1C:0005 8D160200 LEA      DX, [0002]
OF1C:0009 B409      MOV      AH, 09
OF1C:000B CD21      INT      21
OF1C:000D B44C      MOV      AH, 4C
OF1C:000F CD21      INT      21

```

Trong phần liệt kê chưa biên dịch DEBUG thay thế những tên bằng các địa chỉ segment và offset được gán cho những tên đó. Chẳng hạn thay vì MOV AX,@DATA chúng ta có

MOV AX,01FB. LEA DX,MSG trở thành LEA DX,[0002] vì 0002h chính là offset gán cho MSG trong segment .DATA.

Để biểu diễn lệnh D chúng ta hãy liệt kê nội dung vùng nhớ có chứa lời chào "HELLO!". Trước tiên chúng ta thực hiện 2 lệnh khởi tạo DS:

-G5

```

AX=0F1B BX=0000 CX=0121 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=0F1B ES=0EFB SS=0F0B CS=0F1C IP=0005 NV UP DI PL NZ NA PO NC
0F1C:0005 8D160000 LEA DX,[0000] DS:0000=4548

```

Bây giờ chúng ta hãy hiển thị nội dung của bộ nhớ bắt đầu từ DS:0

-D0

```

0F1B:0000 21 00 48 45 4C 4C 4F 21-24 C4 02 8B 1E 46 43 D1 !.HELLO!$....FC.
0F1B:0010 E3 D1 E3 8B 87 BC 3D 8B-97 BE 3D 89 86 7C FF 89 .....=...=...|...
0F1B:0020 96 7E FF 05 0C 00 52 50-E8 7D 6A 83 C4 04 50 E8 ~.....RP.) j...P.
0F1B:0030 6C FB 83 C4 02 0A C0 75-03 E9 F6 FE C6 06 D9 37 l.....u.....?
0F1B:0040 FF 8B 1E 46 43 D1 E3 8B-87 A0 3C A3 60 3E 8B 1E ...FC....<.'>..
0F1B:0050 46 43 8A 87 E6 3C 2A E4-A3 5A 3C D1 E3 D1 E3 8B FC...<*.Z<.....
0F1B:0060 87 FC 31 0B 87 FE 31 75-03 E8 9E FD B8 FF FF 8B ..1...1u.....
0F1B:0070 E5 5D C3 90 55 8B EC 83-EC 08 56 C6 06 0C 42 FF .]..U....V...B.

```

DEBUG hiển thị 80 byte bộ nhớ. Nội dung của mỗi ô nhớ được biểu diễn bằng 2 chữ số hex. Chẳng hạn nội dung hiện thời của ô nhớ 0003h là 48h. Đọc theo hàng thứ nhất chúng ta thấy nội dung của các byte từ 0 đến 7, dấu gạch chéo rồi đến nội dung của các byte từ 8 đến 0Fh. Nội dung của các byte từ 10 đến 1Fh được trình bày ở dòng thứ 2 và cứ như vậy. Phần bên phải của hiển thị là nội dung của ô nhớ được dịch thành các ký tự (các ký tự không in ra được thì đánh dấu bằng một chấm).

Để hiển thị chỉ riêng lời chào "HELLO!" chúng ta đánh:

-D2 8

```

0F1B:0000     48 45 4C 4C 4F 21 24      HELLO!$ 

```

Trước khi tiếp tục chúng ta hãy chú ý đến một đặc trưng của nội dung bộ nhớ. Chúng ta thường viết nội dung của một word theo trình từ byte cao rồi đến byte thấp. Thế nhưng DEBUG lại hiển thị nội dung của ô nhớ theo trình từ byte thấp rồi đến byte cao, chẳng hạn word tại địa chỉ 1 có nội dung là 4845 nhưng DEBUG hiển thị nó thành 4845. Điều này có thể gây ra nhầm lẫn khi chúng dịch nội dung của ô nhớ như các từ.

Bây giờ hãy đổi thông báo "HELLO!" thành "GOODBYE!" bằng lệnh E:

```
-E2 'GOODBYE!$'
```

Để kiểm chứng lại sự thay đổi chúng ta hãy hiển thị nội dung bộ nhớ:

```
-d0 F  
0F1B:0000 21 00 47 4F 4F 44 42 59-45 21 24 8B 1E 46 42 D1 !.GOODBYE!$..FC
```

Bây giờ chúng ta hãy thực hiện chương trình:

```
-G  
GOODBYE!  
Program terminate normally.
```

Lệnh E còn có thể dùng để thay đổi nội dung của ô nhớ một cách tương tác. Chẳng hạn chúng ta muốn thay đổi nội dung của các byte từ 200-204h. Trước khi làm điều đó hãy xem nội dung hiện thời của chúng:

```
-D 200 204  
0F1B:0200 0C FF 5A E9 48 ..Z.H
```

Bây chúng ta hãy đưa các giá trị 1,2,3,4,5 vào các vị trí đó:

```
-E 200  
0F1B:0200 0C.1 FF.2 5A.3 E9.4 48.5 F3.
```

DEBUG bắt đầu bằng việc hiển thị nội dung hiện thời của byte 0200h chính là 0Ch và đợi chúng ta đưa vào giá trị mới. Chúng ta đánh 1 và ấn phím bước trống. Tiếp theo DEBUG hiển thị nội dung của byte 0201h là FFh và một lần nữa lại đợi chúng ta đưa vào giá trị mới. Chúng ta đánh vào 2 và ấn phím bước trống để chuyển sang byte tiếp theo. Cứ như vậy cho đến khi chúng ta đã đưa vào giá trị 5 cho byte 0204h, DEBUG hiển thị nội dung của byte 0205h là F3, do

chúng ta không muốn vào tiếp số liệu nên chúng ta đánh ENTER để trả về dấu nhắc của DEBUG. Nay giờ hãy xem lại nội dung của bộ nhớ:

```
-D 200 204  
0F1B:0200 01 02 03 04 05 .....
```

Trong quá trình vào số liệu chúng ta có thể muốn giữ lại nội dung của một ô nhớ, muốn vậy chúng ta chỉ cần nhấn phím bước trống để chuyển sang byte tiếp theo hay ấn ENTER để trả về dấu nhắc của DEBUG.

E3. Chương trình CODEVIEW

CODEVIEW là một chương trình gỡ rối khá mạnh cho phép người sử dụng thấy được mã nguồn của cả chương trình ở ngôn ngữ bậc cao lẫn hợp ngữ trong quá trình gỡ rối. Có 2 chế độ làm việc với CODEVIEW là chế độ cửa sổ và chế độ tuân tự. Trong chế độ tuân tự, CODEVIEW có thể xem gần giống với DEBUG. Chúng ta phải sử dụng CODEVIEW trong chế độ tuân tự nếu máy của chúng ta không phải là tương thích IBM PC hay chương trình của chúng ta được biên dịch và liên kết không có chọn lựa nào. Trong chế độ cửa sổ chúng ta có thể dùng được mọi khả năng sẵn có của CODEVIEW và đó cũng chính là lý do tại sao chúng ta sẽ chỉ xem xét CODEVIEW trong chế độ này. Vì CODEVIEW là một chương trình lớn với rất nhiều tính năng, chúng ta sẽ không cố gắng nghiên cứu toàn diện về chúng.

Chuẩn bị với chương trình

Để có thể gỡ rối trong chế độ cửa sổ, đoạn mã của chương trình phải có kiểu class "CODE" ví dụ:

```
C SEGMENT 'CODE'
```

Chú ý: dẫn hướng biên dịch trong kiểu định nghĩa đoạn đơn giản hóa .CODE sẽ tạo ra đoạn mã lệnh với kiểu lass mặc định là 'CODE'.

Khi hợp dịch và liên kết chương trình chúng ta phải dùng các tùy chọn /ZI và /CO, ví dụ:

```
MASM /ZI MYPROG;  
LINK /CO MYPROG;
```

Các phần chọn này khiến cho các thông tin ký hiệu cho CODEVIEW được ghép vào file .EXE. Vì điều này làm cho file tạo ra lớn lên rất nhiều nên sau khi đã gỡ rối chúng ta phải hợp dịch lại nó ở dạng thông thường.

Vào môi trường CODEVIEW.

Dòng lệnh để vào môi trường CODEVIEW là :

CV{ options} filename

Trong đó filename phải là tên file khả thi. Các phần chọn sẽ điều khiển các kiểu khởi tạo của CODEVIEW.

| Tuỳ chọn | Ý nghĩa |
|----------|--|
| /D | Bạn sử dụng một máy tương thích IBM không cung cấp các chức năng ngắt do IBM xác định. |
| /I | Bạn sử dụng máy không tương thích IBM và muốn dùng các tổ hợp phím CTRL-C và CTRL-break để kết thúc chương trình.. |
| /M | Bạn có chuột nhưng không muốn dùng nó. |
| /P | Bạn có màn hình EGA không theo chuẩn IBM và gặp rắc rối khi chạy CODEVIEW. |
| /S | Bạn có máy không tương thích IBM và muốn có khả năng xem màn hình ngoài. |
| /W | Bạn có một máy tương thích IBM và muốn chạy CODEVIEW trong chế độ cửa sổ. |

Chúng ta có thể chọn một số tùy chọn cùng lúc, chẳng hạn:

CV /D /M /W Myprog

Chú ý rằng khi làm việc với CODEVIEW khác với DEBUG không cần chỉ ra phần mở rộng của file.

Chế độ cửa sổ

Để mô tả một vài tính năng của CODEVIEW, chúng ta sẽ hợp dịch và liên kết chương trình đã dùng để minh họa hoạt động của DEBUG (PGM14_1.ASM) trong môi trường CODEVIEW.

```
C>MASM /ZI A:PGM14_1;
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All right reserved.

50094 + 289327 Bytes symbol space free

0 Warning Errors
0 Severe Errors
```

```
C>LINK /CO A:PGM14_1;
Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988 .All right reserved.
```

```
C>CV A:PGM14_1
```

Hình E1 cho thấy màn hình hiển thị trong chế độ cửa sổ. Chúng ta thấy có 3 cửa sổ trên màn hình là cửa sổ hiển thị (**display window**) ở phía trên, cửa sổ đối thoại (**dialog window**) ở phía dưới và cửa sổ thanh ghi (**register window**) ở bên phải.

The screenshot shows the Microsoft CodeView interface with the following details:

- Menu Bar:** File, View, Search, Run, Watch, Options, Language, Calls, Help | F8=Trace F5=Go
- Title Bar:** PgM4_2.ASM
- Assembly Code:**

```

1:     MODEL    SMALL
2:     STACK    100H
3:     DATA
4:     MSG      DB      'HELLO!$'
5:     CODE
6:     MAIN     PROC
7:     ; initialize
8:     MOV      AX,@DATA
9:     MOV      DS,AX      :INITIALIZE DS
10:    ;display MESSAGE
11:    LEA      DX,MSG      : GET MESSAGE
12:    MOV      AH,9      : DISPLAY STRING FUNCTION
13:    INT      21h      : DISPLAY MESSAGE
14:    ; return to DOS
15:    MOV      AH,4CH
16:    INT      21h      : DOS EXIT
17:    MAIN    ENDP
18:    END      MAIN

```
- Registers Column:** Shows register values for AX, BX, CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, IP, NV UP, EI PL, NZ NA, PO NC.
- Bottom Status Bar:** Microsoft (R) CodeView (R) Version 2.2
(C) Copyright Microsoft Corp. 1986-1988. All rights reserved

Hình E.1 Hiển thị kiểu cửa sổ CODEVIEW

Cửa sổ hiển thị cho thấy chương trình nguồn với lệnh hiện thời ở chế độ video đảo hay ở một màu khác. Các dòng trước đó được thiết lập làm các điểm dừng được làm nổi bật.

Cửa sổ hội thoại là nơi bạn có thể vào các lệnh, nhưng như chúng ta sẽ thấy các phím chức năng có thể dùng để thực hiện rất nhiều lệnh.

Cửa sổ thanh ghi trình bày nội dung của các thanh ghi và các cờ. Ký hiệu các cờ giống như trong chương trình DEBUG.

Chúng ta cũng có thể kích hoạt một cửa sổ gọi là **watch window** để hiển thị các biến và tình trạng nhất định.

Điều khiển cửa sổ hiển thị.

Nội dung của cửa sổ hiển thị có thể điều khiển bằng bàn phím hay con chuột. Bảng E.2 mô tả các phím và các tổ hợp phím. Các bạn hãy xem thêm tài liệu tham khảo của Microsoft để biết thêm về các thao tác với con chuột.

Điều khiển việc thực hiện chương trình.

Bảng D.3 trình bày các phím chức năng có thể dùng để xoá hay thiết lập các điểm dừng, thực hiện toàn bộ chương trình hay thực hiện đến điểm dừng nhất định.

Chọn lựa thông qua thực đơn.

Menu bar ở đỉnh màn hình có 9 tiêu đề. 2 lệnh cuối (TRACE và GO) có thể dùng chuột để chọn.

1. Để mở một menu ấn ALT và chữ đầu tiên của tiêu đề, chẳng hạn bạn có thể ấn ALT-F để mở File menu khi đó một menu box sẽ xuất hiện.
2. Sử dụng các phím mũi tên để chọn lựa. Khi mục bạn muốn chọn đã được làm nổi bật lên bạn có thể ấn ENTER để chọn nó.
3. Hầu hết các chọn lựa đều được thi hành ngay lập tức tuy nhiên có một số lựa chọn cần phải có sự hỏi đáp.
4. Khi cần thiết sự hỏi đáp cửa sổ hội thoại được mở ra và bạn có thể đánh vào những thông tin cần thiết.

Có thể dùng phím escape để xoá bỏ một menu đã chọn. Khi một menu được mở, có thể dùng các phím mũi tên sang trái và sang phải để chuyển giữa menu này với menu khác.

Bảng E.2 Các lệnh hiển thị.

| Phím | Chức năng |
|---------------|--|
| F1 | Hiển thị cửa sổ giúp đỡ on-line. |
| F2 | Chuyển sang cửa sổ thanh ghi. |
| F3 | Chuyển giữa các chế độ nguồn, kết hợp và chế độ hợp ngữ. Chế độ nguồn trình bày chương trình nguồn trong cửa sổ hiển thị. Chế độ hợp ngữ trình bày các lệnh hợp ngữ còn chế độ kết hợp trình bày cả 2. |
| F4 | Chuyển tới cửa sổ đầu ra. Cửa sổ đầu ra trình bày kết quả thực hiện chương trình. Nhấn phím bất kỳ để trở về cửa sổ hiển thị. |
| F6 | Chuyển con trỏ giữa cửa sổ hiển thị và cửa sổ hội thoại. |
| CTRL-G | Tăng kích thước cửa sổ chứa con trỏ. |
| CTRL-T | Giảm kích thước cửa sổ chứa con trỏ. |
| Mũi tên lên | Chuyển con trỏ lên một dòng. |
| Mũi tên xuống | Chuyển con trỏ xuống một dòng. |
| PgUp | Cuốn lên một trang màn hình. |
| PgDn | Cuốn xuống một trang màn hình. Dừng lại ở cuối file trong chế độ nguồn, trong các chế độ khác nó làm việc giống như lệnh U của chương trình DEBUG. |
| HOME | Cuốn lên đầu file nếu con trỏ đang trong cửa sổ hiển thị hoặc tới dỉnh vùng đệm lệnh nếu con trỏ đang trong cửa sổ hội thoại. |

Bảng E.3 Các phím chức năng thực hiện lệnh.

| Phím | Chức năng |
|-------------|--|
| F5 | Thực hiện đến điểm dừng tiếp theo hay đến cuối chương trình nếu không gặp điểm dừng nào. |
| F7 | Thiết lập điểm dừng tạm thời tại dòng có con trỏ và thực hiện chương trình đến dòng đó nếu như không gặp điểm dừng khác hay điểm kết thúc chương trình. |
| F8 | Thực hiện dòng lệnh tiếp theo trong chương trình nguồn nếu trong chế độ nguồn, hay lệnh tiếp theo trong chế độ hợp ngữ. Nếu một dòng lệnh của chương trình nguồn là một lời gọi nó sẽ đưa vào thường trình được gọi. Chú ý: nó sẽ thực hiện qua các lời gọi hàm của DOS. |
| F9 | Thiết lập hoặc xoá bỏ các điểm dừng tại dòng có |

con trỏ. Nếu dòng không có điểm dừng nó sẽ thiết lập điểm dừng, nếu dòng đã có điểm dừng nó sẽ xoá điểm dừng đó.

F10

Thực hiện bước tiếp theo của chương trình. Giống như F8 nhưng các hàm và thủ tục được thực hiện qua chứ không được duyệt từng lệnh một.

Menu RUN

Menu này chứa các chọn lựa để chạy chương trình. Bảng E.4 trình bày các chọn lựa đó.

Bảng E.4 Các chọn lựa của menu RUN

| Chọn lựa | Nội dung |
|-------------------|--|
| Start | Chạy chương trình từ đầu. Chương trình sẽ chạy cho đến khi gặp điểm dừng hay lệnh quan sát (xem dưới). |
| Restart | Khởi động lại chương trình nhưng không thực hiện lại từ đầu nó. Tất cả các điểm dừng hay các lệnh quan sát vẫn còn hiệu lực. |
| Excute | Thực hiện chậm từ lệnh hiện thời, để dừng lại có thể ấn phím bất kỳ hay nháy chuột. |
| Clear breakpoint: | Xoá tất cả các điểm dừng. Không làm ảnh hưởng đến các lệnh quan sát. |

Các lệnh quan sát.

Một trong các tính năng tiện lợi của CODEVIEW là khả năng kiểm soát các biến và các biểu thức. Các lệnh quan sát sẽ được mô tả sau này khi xác định các biến và các biểu thức để quan sát.

Các lệnh quan sát có thể vào khi đang ở trong cửa sổ xem (watch window), nhưng chúng ta có thể vào chúng dễ dàng hơn trong cửa sổ hội thoại như những lệnh thông thường. Hơn nữa trong cửa sổ hội thoại chúng ta có thể xác định phạm vi các biến để quan sát.

Quan sát bộ nhớ.

Lệnh để xem bộ nhớ như sau:

W{ type} range

Trong đó range có thể là start_address end_address hay
L count

Với count là số các giá trị sẽ được hiển thị. Type nhận một trong các giá trị sau:

| TYPE | MEANING |
|----------|-------------------------|
| Không có | Mặc định |
| B | Byte dạng hex |
| A | ASCII |
| I | Tử thập phân có dấu. |
| U | Tử thập phân không dấu. |
| W | Tử dạng hex |
| D | Tử kép dạng hex |
| S | Short real |
| L | Long real |
| T | 10-byte real. |

Kiểu mặc định là kiểu cuối cùng định nghĩa bằng các lệnh DUMP, ENTER, WATCH hay TRACEPOINT, trong các trường hợp khác nó là B.

Ví dụ mảng A được khai báo như sau:

A DW 37,12,18,96,45,3

Và DS được khởi tạo đến 4A7Dh là địa chỉ đoạn của đoạn .DATA.

Ta có cửa sổ hội thoại:

```
>W A
>WI A
>WI A L6
>WI A 4
>W 100 104
```

Tạo ra cửa sổ xem như sau:

| | | | | | | | | | |
|------------------------|------|------|-------|------|------|------|------|--|--|
| 0) A : 4A7D:0000 | 25 % | 37 | | | | | | | |
| 1) A : 4A7D:0000 | | 12 | 18 | 96 | 45 | 3 | | | |
| 2) A L6 : 4A7D:0000 | | 37 | | | | | | | |
| 3) A L6 : 4A7D:0000 | 00 | 25 | 0000C | 0012 | 0060 | 002D | 0003 | | |
| 4) A 4 : 4A7D:0000 | | 37 | 12 | 18 | | | | | |
| 5) 100 104 : 4A7D:0000 | 6400 | 8448 | 6912 | | | | | | |

Trong dòng (0) CODEVIEW hiển thị cả giá trị dạng hex và dạng ASCII của biến A. Tại dòng (1) chúng ta yêu cầu hiển thị A như số nguyên có dấu. Trong dòng (2) chúng ta xem mảng A gồm 6 từ biểu diễn dưới dạng các số nguyên. Tại dòng (3) chúng ta xem mảng A dưới dạng hex. Trong dòng 4 chúng ta hiển thị dưới dạng thập phân các biến trong phạm vi: địa chỉ đầu = A = 0000h, địa chỉ kết thúc = 0004h. Tại dòng (5) chúng ta khai báo phạm vi là DS:0100h đến DS:0104h. Hiển thị ở dạng thập phân vì kiểu là mặc định theo dòng (4).

Quan sát ngăn xếp.

Chúng ta có thể xem xét ngăn xếp như trường hợp đặc biệt của một khoảng bộ nhớ. Chẳng hạn nếu SS:SP = 4A6C:000Ah và BP = 000Ch để quan sát 6 từ của ngăn xếp chúng ta đánh:

```
>WI SP L6
```

và cửa sổ quan sát hiện ra như sau:

| | | | | | | | |
|----|------------------|------|------|------|------|------|------|
| 0) | SP 16 :4A6C:000A | 1813 | 5404 | 2009 | 5404 | 2741 | 5404 |
|----|------------------|------|------|------|------|------|------|

chúng ta cũng có thể dùng BP như con trỏ ngăn xếp, chẳng hạn:

```
>WI BP L6
```

và cửa sổ quan sát hiện ra như sau:

| | | | | | | | |
|----|------------------|------|------|------|------|------|------|
| 1) | BP 16 :4A6C:000C | 5404 | 2009 | 5404 | 2741 | 5404 | 3085 |
|----|------------------|------|------|------|------|------|------|

Quan sát biểu thức.

Cửa sổ quan sát có thể dùng để quan sát các giá trị của các biểu thức. Cú pháp của nó như sau:

WP? Expression{ ,format}

Trong đó Expression có thể là một biến đơn hay một biểu thức phức tạp có liên quan đến nhiều biến và hằng số. Phần tuỳ chọn format là một chữ cái để xác định các biểu thức sẽ được hiển thị như thế nào, dưới đây là một vài khả năng:

| Format | Dạng đầu ra |
|---------------|--------------------------------|
| d | Số nguyên thập phân có dấu. |
| i | Số nguyên thập phân có dấu. |
| u | Số nguyên thập phân không dấu. |
| x | Số nguyên dạng hex |
| c | Ký tự đơn |

Dưới đây là một số ví dụ sử dụng mảng A đã được khai báo trước đó, giả sử rằng AX=1 và BX=4.

```
>W? A
>W? A, d
>W? AX+BX
>W? A+2*AX
```

và cửa sổ quan sát sẽ như sau:

- 0) A : 0x0025
- 1) A, d: 37
- 2) AX+BX : 0x0005
- 3) A+2*AX: 0x0027

Trong dòng (0) biểu thức được biểu diễn chỉ là biến A. Nó xuất hiện dưới dạng 0x0025 (ký hiệu 0x... là ký hiệu của ngôn ngữ C để biểu diễn các chữ số dạng hex). Trong dòng (1) chúng ta yêu cầu biểu diễn A dưới dạng số thập phân. Tại dòng (2) chúng ta nhận được tổng của 2 thanh ghi AX và BX. Tại dòng (3) chúng ta nhận được tổng của A và 2 lần nội dung thanh ghi AX.

Định địa chỉ gián tiếp thanh ghi

Trong thực tế đôi khi chúng ta muốn theo dõi các byte hay word được trỏ tới bằng một thanh ghi, chẳng hạn [BX] hay [BP+4]. CODEVIEW không cho phép các ký hiệu con trỏ dạng dấu ngoặc vuông nhưng lại cho phép sử dụng các ký hiệu sau để thay thế:

Ký hiệu trong hợp ngữ

BYTE PTR [register]
WORD PTR [register]
DWORD PTR [register]

Ký hiệu dùng trong CODEVIEW

BY register
WO register
DW register

Chẳng hạn BX chứa 0100h và nội dung bộ nhớ như sau:

| Offset | Nội dung |
|---------------|-----------------|
| 0100h | ABh |
| 0101h | CDh |
| 0102h | EFh |

0103h 00h

Các lệnh quan sát sau đây :

```
>W? BY BX  
>W? WO BX  
>W? WO BX+2
```

Sẽ tạo ra cửa sổ quan sát như sau:

- ```
0) BY BX :0x00ab
1) WO BX :0xcdab
2) WO BX+2 :0x00ef
```

### Xoá các dòng từ cửa sổ quan sát

Để xoá các dòng từ cửa sổ quan sát, chúng ta có thể dùng lệnh Y (Yank). Cú pháp của lệnh như sau:

Y number

Trong đó number là số hiệu của dòng cần xoá. Lệnh Y \* xoá tất cả các dòng.

### Các điểm dừng

Chúng ta có thể khai báo các biến hay phạm vi các biến như các điểm dừng. Khi biến bị thay đổi, chương trình sẽ dừng thực hiện. Cú pháp của lệnh như sau:

```
TP? Expression{ ,format}
hay
TP{ type} range
```

Trong đó format, type range có ý nghĩa giống như lệnh W. CODEVIEW sẽ hiển thị các biểu thức, biến hay phạm vi các biến ở dạng giống như lệnh W ngoại trừ một điều là hiển thị ở dạng sáng. Chẳng hạn chúng ta có thể đánh:

```
TPI A L6
```

Và CODEVIEW sẽ hiển thị:

```
0) A L6 : 4A7D:0000 37 18 96 45 3
```

trong cửa sổ quan sát. Nếu như có một phần tử của mảng A thay đổi chương trình sẽ dừng việc thực hiện.

### Các điểm quan sát (Watch point).

Một điểm quan sát sẽ dừng việc thực hiện chương trình khi biểu thức khác 0 (đúng). Dòng lệnh để thiết lập một điểm quan sát như sau:

WP? Expression{ , format}

Trong đó Expression là một biểu thức quan hệ gồm các biến và các hằng.

Ví dụ nếu A được định nghĩa như sau:

A DW 25h

Và nội dung hiện thời của AX và BX tương ứng là 5 và 2, chúng ta có các lệnh hội thoại:

```
>WP? AX>BX
>WP? AX-BX-3
>WP? A>25
>WP? A=25
```

Sẽ tạo ra cửa sổ quan sát như sau:

- |    |         |   |        |
|----|---------|---|--------|
| 0) | AX>BX   | : | 0x0001 |
| 1) | AX-BX-3 | : | 0x0000 |
| 2) | A>25    | : | 0x0000 |
| 3) | A=25    | : | 0x0025 |

Hiển thị sau (0) chỉ ra rằng việc thực hiện chương trình sẽ bị dừng nếu AX>BX là đúng. Vì nội dung hiện thời của AX và BX là 2 và 5 nên biểu thức có giá trị đúng và chương trình lập tức dừng thực hiện. CODEVIEW chỉ ra biểu thức đúng bằng dấu hiệu 0x0001.

Trong dòng (1) việc thực hiện sẽ ngừng lại khi AX-BX-3 khác 0. Hiện thời nó bằng 0 như vậy biểu thức nhận giá trị sai, CODEVIEW chỉ ra điều đó bằng dấu hiệu 0x0000.

Trong dòng (2), việc thực hiện sẽ ngừng lại khi A>25, hiện thời biểu thức nhận giá trị sai. Trong dòng (3) việc thực hiện sẽ ngừng lại khi A=25 biểu thức này

đúng nên việc thực hiện chương trình cũng lập tức bị dừng lại. CODEVIEW trình bày nội dung hiện thời của A là 0x0025.

### Các lệnh của DEBUG trong CODEVIEW.

Hầu hết các lệnh của DEBUG có thể dùng trong cửa sổ hội thoại của CODEVIEW. Khi làm việc trong chế độ nguồn, có thể sử dụng các ký hiệu biến trong dòng lệnh. Ví dụ nếu BELOW là một nhãn của chương trình thì

G BELOW

Sẽ khiến cho việc thực hiện bị dừng lại khi gặp nhãn BELOW. Trong các lệnh D hay E có thể định nghĩa kiểu. Cú pháp cho lệnh E như sau:

E { type} address { list}

Lệnh D cũng có cú pháp tương tự. Type có cùng các giá trị như trong lệnh W. Ví dụ:

E I A 17 -1 456 8900 -29

Sẽ cho phép người sử dụng vào 5 số nguyên thập phân đầu tiên của mảng A.

## Phụ lục F

# TẬP LỆNH HỢP NGỮ

Trong phụ lục này chúng tôi sẽ trình bày dạng mã hoá nhị phân của các lệnh 8086 tiêu biểu và đưa ra một bảng tổng kết về các lệnh thường dùng của 8086, 8087, 8286 và 80386.

### F1. Dạng của các lệnh 8086 điển hình.

Một lệnh máy cho 8086 chiếm từ 1 đến 6 byte. Trong hầu hết các lệnh, byte đầu tiên chứa mã lệnh, byte thứ 2 chứa chế độ địa chỉ hoá của các toán hạng còn các byte khác chứa hoặc là thông tin địa chỉ của dữ liệu hay chính bản thân dữ liệu. Một lệnh 2 toán hạng điển hình có dạng như hình F.1.

Trong byte đầu tiên chúng ta thấy 6 bit mã lệnh xác định thao tác của lệnh. Mã lệnh này dùng cho cả thao tác 16 bit cũng như 8 bit. Kích thước của toán hạng cho trong bit W, W=0 có nghĩa là dữ liệu 8 bit và W=1 có nghĩa là dữ liệu 16 bit.

Trong các thao tác thanh ghi với thanh ghi, thanh ghi với bộ nhớ, và bộ nhớ với thanh ghi, trường REG trong byte thứ 2 chứa số hiệu của thanh ghi và bit D xác định thanh ghi trong trường REG là toán hạng nguồn hay đích, D=0 có nghĩa là nguồn và D=1 có nghĩa là đích. Trong các thao tác dạng khác, trường REG chứa 3 bit mở rộng của mã lệnh.

| Byte1                                                                             | Byte2                                             | Byte3 | Byte4 | Byte5 | Byte6 |
|-----------------------------------------------------------------------------------|---------------------------------------------------|-------|-------|-------|-------|
| 7 6 5 4 3 2   1 0   7 6 5 4 3 2 1 0   low disp   high disp   low data   high data | Mã lệnh   D W   MOD REG   R/M   or data   or data |       |       |       |       |

Hình F.1 Dạng mã lệnh.

| MOD = 11 |     |     | Tính địa chỉ |                   |                  |                   |
|----------|-----|-----|--------------|-------------------|------------------|-------------------|
| R/M      | W=0 | W=1 | R/M          | MOD=00            | MOD=01           | MOD=10            |
| 000      | AL  | AX  | 000          | (BX) + (SI)       | (BX) + (SI) + D8 | (BX) + (SI) + D16 |
| 001      | CL  | CX  | 001          | (BX) + (DI)       | (BX) + (DI) + D8 | (BX) + (DI) + D16 |
| 010      | DL  | DX  | 010          | (BP) + (SI)       | (BP) + (SI) + D8 | (BP) + (SI) + D16 |
| 011      | BL  | BX  | 011          | (BP) + (DI)       | (BP) + (DI) + D8 | (BP) + (DI) + D16 |
| 100      | AH  | SP  | 100          | (SI)              | (SI) + D8        | (SI) + D16        |
| 101      | CH  | BP  | 101          | (DI)              | (DI) + D8        | (DI) + D16        |
| 110      | DH  | SI  | 110          | Địa chỉ trực tiếp | (BP) + D8        | (BP) + D16        |
| 111      | BH  | DI  | 111          | (BX)              | (BX) + D8        | (BX) + D16        |

MOD=11 - chế độ thanh ghi

MOD=00 - chế độ bộ nhớ không có phần dịch, trừ khi R/M=110 thì phần dịch dài 16 bit theo sau.

MOD=01 - chế độ bộ nhớ với 8 bit dịch theo sau (D8)

MOd=10 - chế độ bộ nhớ với 16 bit dịch theo sau (D16)

### Hình F.2 Các trường MOD và R/M.

Bảng sự kết hợp của bit W và trường REG có thể định nghĩa tới 16 thanh ghi, xem bảng F.1.

Toán hạng thứ 2 được xác định bằng các trường MOD và R/M. Hình F.2 trình bày các chế độ khác nhau.

Trường SEG dùng để xác định các thanh ghi đoạn. Bảng F.2 chỉ ra sự mã hoá các thanh ghi đoạn.

**Bảng F.1 Mã hoá các thanh ghi.**

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

Bảng F.2 Mã hoá các thanh ghi đoạn

| SEG | Thanh ghi |
|-----|-----------|
| 00  | ES        |
| 01  | CS        |
| 10  | SS        |
| 11  | DS        |

## F.2 Tập lệnh của 8086

Các lệnh của 8086 được trình bày theo trình tự a b c. Trong đó:

- ♦ (register) biểu diễn nội dung của thanh ghi.
- ♦ (EA) biểu diễn nội dung của ô nhớ có địa chỉ EA.
- ♦ Các cờ bị ảnh hưởng là các cờ bị thay đổi theo kết quả thực hiện lệnh.
- ♦ Các cờ không xác định là các cờ mà giá trị của nó không biểu thị kết quả của lệnh.
- ♦ disp\_lowdisp\_high biểu diễn phần dịch 16 bit.

### AAA (ASCII Adjust for Addition)

Điều chỉnh kết quả trong AL khi cộng 2 số BCD không nén hay 2 chữ số ASCII.

*Dạng lệnh:* AAA

*Thao tác:* nếu nibble thấp của AL lớn hơn 9 hay cờ AF đặt thì tăng AL lên 6, tăng AH lên 1 và thiết lập 1 cờ AF. Lệnh này luôn xoá nibble cao của AL và chép cờ AF vào CF.

*Cờ:* Bị ảnh hưởng: AF, CF

Không xác định PF, SF, OF, ZF.

*Mã lệnh:* 00110111

## AAD (ASCII Adjust for Division)

Điều chỉnh số bị chia dạng BCD không nén trong AX khi thực hiện phép chia.

*Dạng lệnh:* AAD

*Thao tác:* Số BCD không nén trong AX được đổi thành dạng nhị phân và giữ lại trong AL bằng cách nhân AH với 10 và cộng vào AL sau đó xoá AH.

*Cờ:* Bị ảnh hưởng - PF, SF, ZF

Không xác định - AF, CF, OF.

*Mã lệnh* 11010101 00001010

D5 0A

## AAM (ASCII Adjust for Multiplication)

Đổi kết quả của phép nhân 2 chữ số BCD thành dạng BCD không nén. Có thể dùng để đổi các số nhỏ hơn 100 thành dạng BCD không nén.

*Dạng lệnh:* AAM

*Thao tác:* Nội dung của AL được đổi thành 2 chữ số BCD không nén và được đặt trong AX bằng cách chia AL cho 10, thương số nhận được đặt trong AH còn số dư đặt trong AL.

*Cờ:* Bị ảnh hưởng - PF, SF, ZF

Không xác định - AF, CF, OF.

*Mã lệnh* 11010100 00001010

D4 0A

## AAS (ASCII Adjust for Subtraction)

chỉnh lại kết quả của phép trừ 2 số BCD không nén.

*Dạng lệnh:* AAS

*Thao tác:* Nếu nibble thấp của AL lớn hơn 9 hay cờ AF được đặt thì giảm AL đi 6, giảm AH đi 1 và thiết lập cờ AF. Lệnh này luôn xoá nibble cao của AL và chép nội dung cờ AF sang cờ CF.

|                |                                  |
|----------------|----------------------------------|
| <i>Còn:</i>    | Bị ảnh hưởng - AF, CF            |
|                | Không xác định - PF, SF, ZF, OF. |
| <i>Mã lệnh</i> | 00111111                         |
|                | 3F                               |

### **ADC (ADd with Carry)**

Còn nhớ được cộng vào tổng của toán hạng nguồn và toán hạng đích.

*Dạng lệnh:* ADC Destination, Source

*Thao tác:* Nếu CF=1 thì (dest) = (source) + (dest) +1

Nếu CF=0 thì (dest)=(source)+(dest)

*Còn:* Bị ảnh hưởng - AF, CF, OF, PF, SF, ZF

*Mã lệnh:* Cộng ô nhớ hay thanh ghi với thanh ghi:

000100dw mod reg r/m

Cộng trực tiếp số hạng vào thanh chứa:

0001010w data

Cộng trực tiếp vào ô nhớ hay thanh ghi:

100000sw mod 010 r/m data

(trong đó s được thiết lập nếu một byte số liệu được cộng vào ô nhớ hay thanh ghi 16 bit).

### **ADD (ADDition)**

*Dạng lệnh:* ADD destination, source

*Thao tác:* (dest)=(source)+(dest)

*Còn:* Bị ảnh hưởng - AF, CF, OF, PF, SF, ZF

*Mã lệnh:* Cộng ô nhớ hay thanh ghi với thanh ghi:

000000dw mod reg r/m

Cộng trực tiếp số hạng vào thanh chứa:

0000010w data

Cộng trực tiếp vào ô nhớ hay thanh ghi:

100000sw mod 000 r/m data

(trong đó s được thiết lập nếu một byte số liệu được cộng vào ô nhớ hay thanh ghi 16 bit).

## AND(Logic AND)

*Dạng lệnh:* AND destination, source

*Thao tác:* Mỗi bit của toán hạng nguồn được lấy và logic với bit tương ứng của toán hạng đích, kết quả được lưu trong toán hạng đích. Cờ CF và OF bị xoá.

*Cờ:* Bị ảnh hưởng - CF, OF, PF, SF, ZF

Không xác định - AF

*Mã lệnh:* Và ô nhớ hay thanh ghi với thanh ghi:

001000dw mod reg r/m

Và trực tiếp với thanh chứa:

0010010w data

Và trực tiếp với ô nhớ hay thanh ghi:

100000w mod 100 r/m data

## CALL (Gọi thủ tục).

*Dạng lệnh:* CALL target.

*Thao tác:* Địa chỉ offset của lệnh tiếp theo trong tiến trình thực hiện được đưa vào ngăn xếp, điều khiển được chuyển cho toán hạng target. Địa chỉ của target được tính như sau:

(1) chế độ địa chỉ trực tiếp trong nội bộ một đoạn .  
offset=IP+độ dịch,

(2) chế độ địa chỉ gián tiếp trong nội bộ một đoạn .  
offset=(EA),

(3) chế độ địa chỉ trực tiếp ngoài đoạn segment:offset cho  
trong lệnh

và (4) chế độ địa chỉ gián tiếp ngoài đoạn  
segment=(EA+2), offset=(EA).

*Cờ:* Bị ảnh hưởng - không

*Mã lệnh:* Chế độ địa chỉ trực tiếp trong một đoạn:

11101000 disp\_low disp\_high

Chế độ địa chỉ gián tiếp trong một đoạn:

11111111 mod 010 r/m

Chế độ địa chỉ trực tiếp ở ngoài đoạn:

10011010 offset\_low offset\_high seg\_low seg\_high

Chế độ địa chỉ gián tiếp ở ngoài đoạn:

11111111 mod 011 r/m.

### **CBW(Convert Byte to Word)**

Đổi số có dấu 8 bit trong AL thành số có dấu 16 bit trong AX.

*Dạng lệnh:* CBW

*Thao tác:* Nếu bit 7 của AL bằng 1 thì AH=FFh.

Nếu bit 7 của AL bằng 0 thì AH=00h

*Cờ:* Không ảnh hưởng - không

*Mã lệnh:* 10011000

98

### **CLC (CLear Carry flag)**

*Dạng lệnh:* CLC

*Thao tác:* Xoá CF.

*Cờ:* Không ảnh hưởng - CF.

*Mã lệnh:* 11111000

F8

### **CLD (CLear Direction flag)**

*Dạng lệnh:* CLD

*Thao tác:* Xoá DF.

*Cờ:* Không ảnh hưởng - DF.

*Mã lệnh:* 11111100

FC

## **CLI (CLear Interrupt flag)**

Cấm các ngắt ngoài che được.

*Dạng lệnh:* CLI

*Thao tác:*Xoá IF.

*Cờ:* Bị ảnh hưởng - IF.

*Mã lệnh:* 11111010

FA

## **CMC (Complement Carry flag - lấy bù cờ nhớ)**

*Dạng lệnh:* CMC

*Thao tác:*Lấy bù CF.

*Cờ:* Bị ảnh hưởng - CF.

*Mã lệnh:* 11110100

F5

## **CMP (CoMPare - so sánh)**

So sánh 2 toán hạng bằng cách thực hiện phép trừ nhưng không lưu kết quả.

*Dạng lệnh:* CMP destination, source

*Thao tác:*Trừ toán hạng đích cho toán hạng nguồn và thiết lập các cờ theo kết quả của phép tính, các toán hạng vẫn không thay đổi.

*Cờ:* Bị ảnh hưởng - AF, CF, OF, PF, SF, ZF.

*Mã lệnh:* Ô nhớ hay thanh ghi với thanh ghi:

001110dw mod reg r/m

Trực tiếp với thanh chứa

0011110w data

Trực tiếp với ô nhớ hay thanh ghi

1000000sw mod 111 r/m

## **CMP/CMPSB/CMPSW : So sánh chuỗi byte hay word.**

So sánh 2 toán hạng bộ nhớ. Nếu có tiền tố REP có thể so sánh chuỗi các byte hay word.

*Dạng lệnh:* CMS source\_string, dest\_string  
hay CMSB  
hay CMPSW

*Thao tác:* Trừ các thành phần của chuỗi source\_string (được chỉ số bằng DS:SI) cho các thành phần của chuỗi dest\_string (được chỉ số bằng ES:SI). Các cờ trạng thái được thiết lập theo kết quả của phép tính. Nếu cờ định hướng DF=0 thì cả SI và DI đều tăng sau mỗi phép tính (1 khi so sánh byte và 2 khi so sánh word) trong trường hợp còn lại cả SI và DI đều giảm (cũng 1 khi so sánh byte và 2 khi so sánh word).

*Cờ:* Bị ảnh hưởng - AF, CF, SF, ZF, PF, OF.

*Mã lệnh:* 1010011w.

## **CWD (Convert WORD to Double word) đổi word thành double word.**

Đổi số 16 bit có dấu trong AX thành số 32 bit có dấu trong DX:AX.

*Dạng lệnh:* CWD

*Thao tác:* Nếu bit 15 của AX =1 thì DX nhận giá trị FFFFh trong trường hợp còn lại DX nhận giá trị 0000h

*Cờ:* Bị ảnh hưởng - không

*Mã lệnh:* 10011001

99

## **DAA (Decimal Adjust for Addition)**

Điều chỉnh tổng trong AL của 2 toán hạng BCD nén.

*Dạng lệnh:* DAA

*Thao tác:* nếu nibble thấp của AL lớn hơn 9 hay cờ AF đặt thì tăng AL lên 6, thiết lập 1 cờ AF. Nếu AL lớn hơn 9Fh hay CF được thiết lập 1 thì cộng 60h vào AL và thiết lập cờ CF.

*Cờ:* Bị ảnh hưởng:AF, CF, PF, ZF, SF

Không xác định OF.

*Mã lệnh:* 00100111

27

### DAS (Decimal Adjust for Subtraction)

Điều chỉnh hiệu trong AL của 2 toán hạng BCD nén.

*Dạng lệnh:* DAS

*Thao tác:* nếu nibble thấp của Al lớn hơn 9 hay cờ AF đặt thì trừ AL đi 60h và thiết lập 1 cờ CF.

*Cờ:* Bị ảnh hưởng: AF, CF, PF, ZF, SF

*Mã lệnh:* 00101111

2F

### DEC (DECrement).

*Dạng lệnh:* DEC destination.

*Thao tác:* Giảm toán hạng đích destination đi 1.

*Cờ:* Bị ảnh hưởng - AF, OF, PF, SF, ZF.

*Mã lệnh:* giảm thanh ghi 16 bit:

01001 reg

Giảm ô nhớ hay thanh ghi:

1111111w mod 001 r/m

### DIV (DIVide).

Thực hiện phép chia không dấu.

*Dạng lệnh:* Số chia là toán hạng nguồn có thể là một ô nhớ hay một thanh ghi. Trong phép chia cho byte (toán hạng nguồn 8 bit) số bị chia trong AX còn trong phép chia cho word (toán hạng nguồn 16 bit) số bị chia trong DX:AX. Thương số chứa trong AL (AX trong trường hợp chia cho word) và số dư trả về trong AH (DX trong phép chia cho word) và ngắt INT 0 được tạo ra.

*Cờ:* Không xác định - AF, OF, CF, ZF, PF, SF.

*Mã lệnh:* 1111011w mod 110 r/m

## **ESC (ESCAPE)**

Cho phép các bộ xử lý khác chẵng hạn 8087 thực hiện các thao tác của chúng. Bộ vi xử lý 8086 không thực hiện thao tác nào ngoại trừ việc lấy các toán hạng bộ nhớ cho các bộ xử lý khác.

*Dạng lệnh:* ESC external\_opcode, source

*Cờ:* Không

*Mã lệnh:* 11011xxx mod xxx r/m

(trong đó xxx biểu diễn mã lệnh của bộ xử lý khác)

## **HLT (Halt)**

Đưa bộ vi xử lý vào trạng thái dừng để đợi một ngắt ngoài.

*Dạng lệnh:* HLT

*Cờ:* Không.

*Mã lệnh:* 11110100

F4

## **IDIV (Integer Divide)**

Thực hiện phép chia có dấu.

*Dạng lệnh:* IDIV source.

*Thao tác:* Số chia là toán hạng nguồn có thể là một ô nhớ hay một thanh ghi. Trong phép chia cho byte (toán hạng nguồn 8 bit) số bị chia trong AX còn trong phép chia cho word (toán hạng nguồn 16 bit) số bị chia trong DX:AX. Thương số chứa trong AL (AX trong trường hợp chia cho word) và số dư trả về trong AH (DX trong phép chia cho word). Nếu thương số lớn hơn 8 bit (16 bit trong phép chia cho word) thì ngắt INT 0 được tạo ra.

*Cờ:* Không xác định - AF, OF, CF, ZF, PF.

*Mã lệnh:* 1111011w mod 111 r/m

## **IMUL (Integer MULtiplication)**

Thực hiện phép nhân có dấu.

*Dạng lệnh:* IMUL source.

*Thao tác:* Số nhân là toán hạng nguồn có thể là một ô nhớ hay một thanh ghi. Trong phép nhân với byte (toán hạng nguồn 8 bit) số bị nhân trong AL còn trong phép nhân với word (toán hạng nguồn 16 bit) số bị nhân trong AX. Tích số chứa trong AX (DX:AX trong trường hợp nhân với word). Các cờ CF và OF được thiết lập nếu nửa cao của tích không phải là phần mở rộng dấu của nửa thấp.

*Cờ:* Bị ảnh hưởng - CF, OF.

*Mã lệnh:* 1111011w mod 101 r/m.

## **IN - nhập vào từ cổng 1 byte hay word.**

*Dạng lệnh:* IN thanh chứa,cổng.

*Thao tác:* Nội dung của thanh chứa được thay bằng nội dung của cổng vào ra có chỉ định. Toán hạng cổng có thể là hằng số (cho một cổng cố định) hay DX (cho số hiệu cổng thay đổi).

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* Cổng cố định:

1110010w port

Cổng thay đổi:

1110110w.

## **INC (INCrement)**

*Dạng lệnh:* INC destination

*Thao tác:* Tăng toán hạng đích lên 1.

*Cờ:* Bị ảnh hưởng - AF, SF, PF, ZF, OF.

*Mã lệnh:* Tăng thanh ghi 16 bit:

01000 reg

Tăng ô nhớ hay thanh ghi:

1111111w mod 000 r/m.

## **INT (INTerrupt)**

Chuyển điều khiển cho một trong 256 chương trình phục vụ ngắt.

*Dạng lệnh:* INT interrupt\_type  
*Thao tác:* thanh ghi FLAGS được đưa vào ngăn xếp, TF và IF bị xoá, nội dung của CS được đưa vào ngăn xếp và thay bằng từ cao của vector ngắt, nội dung của IP cũng được đưa vào ngăn xếp và thay bằng từ thấp của vector ngắt.

*Cờ:* Bị ảnh hưởng - IF, TF.

*Mã lệnh:* Ngắt 3

11001100

Các ngắt khác:

11001101 số hiệu ngắt.

### **INTO (INTerrupt if Overflow)**

Tạo ra ngắt 4 nếu cờ OF được thiết lập.

*Dạng lệnh:* INTO

*Thao tác:* Nếu OF=1 thì tạo ngắt 4, ngược lại không có gì xảy ra.

*Cờ:* Nếu OF=1 thì OF và TF bị xoá.

Nếu OF=0 thì không cờ nào bị ảnh hưởng.

*Mã lệnh:* 11001110

CE

### **IRET (Interrupt RETurn)**

Trở về từ thường trình phục vụ ngắt.

*Dạng lệnh:* IRET

*Thao tác:* Lấy ra khỏi ngăn xếp nội dung của CS, IP và thanh ghi FLAGS.

*Cờ:* Bị ảnh hưởng - tất cả.

*Mã lệnh:* 11001111

CF

**J (điều kiện): nhảy gần nếu thoả mãn điều kiện.**

*Dạng lệnh:* J(điều kiện) short\_label

*Thao tác:* Nếu điều kiện đúng, nhảy tới nhãn. Nhãn phải nằm trong phạm vi từ -128 byte đến 127 byte tính từ lệnh tiếp theo.

Cờ:

Bị ảnh hưởng - không.

| Lệnh | Nhảy nếu              | Điều kiện           | Mã lệnh |
|------|-----------------------|---------------------|---------|
| JA   | Above                 | CF=0 và ZF=0        | 77 disp |
| JAE  | Above or Equal        | CF=0                | 73 disp |
| JB   | Below                 | CF=1                | 72 disp |
| JBE  | Below or Equal        | CF=1 hay ZF=1       | 76 disp |
| JC   | Carry                 | CF=1                | 72 disp |
| JCXZ | CX =0                 | (CF or ZF) =0       | E3 disp |
| JE   | Equal                 | ZF=1                | 74 disp |
| JG   | Greater               | ZF=0 và SF=OF       | 7F disp |
| JGE  | Greater or Equal      | ZF=OF               | 7D disp |
| JL   | Less                  | (SF xor OF)=1       | 7C disp |
| JLE  | Less or Equal         | (SF xor OF) or ZF=1 | 7E disp |
| JNA  | Not Above             | CF=1 hay ZF=1       | 76 disp |
| JNAE | Not Above or Equal    | CF=1                | 72 disp |
| JNB  | Not Below             | CF=0                | 73 disp |
| JNBE | Not Below or Equal    | CF=0 hay ZF=0       | 77 disp |
| JNC  | Not Carry             | CF=0                | 73 disp |
| JNE  | Not Equal             | ZF=0                | 75 disp |
| JNG  | Not Greater           | (SF xor OF) or ZF=1 | 7E disp |
| JNGE | Not Greater nor Equal | (SF xor OF)=1       | 7C disp |
| JNL  | Not less              | SF = OF             | 7D disp |
| JNLE | Not less nor Equal    | ZF=0 và SF=OF.      | 7F disp |
| JNO  | Not Overflow          | OF=0                | 71 disp |
| JNP  | Not Parity            | PF=0                | 7B disp |
| JNS  | Not Sign              | SF=0                | 79 disp |
| JNZ  | Not Zero              | ZF=0                | 75 disp |
| JO   | Overflow              | OF=1                | 70 disp |
| JP   | Parity                | PF=1                | 7A disp |
| JPE  | Parity Even           | PF=1                | 7A disp |
| JPO  | Parity Odd            | PF=0                | 7B disp |
| JS   | Sign                  | SF=1                | 78 disp |
| JZ   | Zero                  | ZF=1                | 74 disp |

## JMP (JuMP)

*Dạng lệnh:* JMP target

*Thao tác:* điều khiển được chuyển tới nhãn target.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* Chế độ địa chỉ trực tiếp trong đoạn

11101001 disp\_low disp\_high  
Chế độ địa chỉ trực tiếp bên trong đoạn(nhảy gần)

11101011 disp

Chế độ địa chỉ trực tiếp ngoài đoạn:

11101010

Chế độ địa chỉ gián tiếp ngoài đoạn

11111111 mod 101 r/m

Chế độ địa chỉ gián tiếp trong đoạn:

11111111 mod 100 r/m

## LAHF (Load AH from Flags)

*Dạng lệnh:* LAHF

*Thao tác:* 8 bit thấp của thanh ghi FLAGS được chuyển vào AH.

*Cờ:* Bị ảnh hưởng - không

*Mã lệnh:* 10011111

9F

## LDS (Load Data Segment register)

Nạp vào thanh ghi DS địa chỉ đoạn và thanh ghi công dụng chung địa chỉ offset để có thể truy nhập dữ liệu.

*Dạng lệnh:* LDS destination,source

*Thao tác:* Toán hạng nguồn là một ô nhớ có độ dài 2 word. Từ thấp được đặt vào thanh ghi đích còn từ cao đặt vào thanh ghi DS.

*Cờ:* Bị ảnh hưởng - không

*Mã lệnh:* 11000101 mod reg r/m.

## **LEA (Load Effective Address).**

Nạp địa chỉ offset của một ô nhớ vào thanh ghi.

*Dạng lệnh:* LEA destination,source

*Thao tác:* Địa chỉ offset của toán hạng nguồn được đặt vào toán hạng đích là một thanh ghi công dụng chung.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* 11000100 mod reg r/m.

## **LES (Load Extra Segment register)**

Nạp vào thanh ghi ES địa chỉ đoạn và một thanh ghi công dụng chung địa chỉ offset để có thể truy nhập dữ liệu.

*Dạng lệnh:* LES destination,source

*Thao tác:* Toán hạng nguồn là một ô nhớ có độ dài 2 word. Từ thấp được đặt vào thanh ghi đích còn từ cao đặt vào thanh ghi ES.

*Cờ:* Bị ảnh hưởng - không

*Mã lệnh:* 11000100 mod reg r/m.

## **LOCK khoá bus.**

Khoá bus trong môi trường có nhiều bộ xử lý.

*Dạng lệnh:* LOCK

*Thao tác:* LOCK có thể dùng như một tiền tố đặt trước bất cứ lệnh nào. Bus sẽ bị khoá trong thời gian thực hiện lệnh để ngăn không cho các bộ xử lý khác truy nhập bộ nhớ.

*Cờ:* Không.

*Mã lệnh:* 11110000

F0

## **LODS/LODSB/LODSW nạp chuỗi byte hay word.**

Chuyển nội dung của byte hay word bộ nhớ có chỉ số trong SI vào thanh chứa.

*Dạng lệnh:* LODS source\_string

hay

LODSB

hay

LODSW

*Thao tác:* Byte (hay word) nguồn được nạp vào AL (hay AX). SI tăng lên 1 (hay 2) nếu DF=0, ngược lại SI giảm đi 1 (hay 2) nếu DF=1.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* 1010110w.

## LOOP

Lặp lại cho đến khi hết số đếm.

*Dạng lệnh:* LOOP short\_label

*Thao tác:* CX được giảm đi 1 và nếu kết quả khác 0, điều khiển được chuyển tới lệnh có nhãn short\_label, trong trường hợp còn lại điều khiển được chuyển cho lệnh nằm sau lệnh LOOP.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* 11100010 disp

E2.

## LOOPE/LOOPZ (LOOP if Equal/LOOP if Zero).

Vòng lặp được điều khiển bằng bộ đếm và cờ ZF.

*Dạng lệnh:* LOOPE short\_label

hay

LOOPZ short\_label

*Thao tác:* CX được giảm 1, nếu CX khác 0 và cờ ZF=1 thì điều khiển được chuyển cho lệnh có nhãn short\_label, trong trường hợp khác, điều khiển được chuyển cho lệnh tiếp theo.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* 1110001 disp

E1

## **LOOPNE/LOOPNZ (LOOP if Not Equal/LOOP if Not Zero).**

Vòng lặp được điều khiển bằng bộ đếm và cờ ZF.

*Dạng lệnh:*      LOOPNE short\_label

hay

LOOPNZ short\_label

*Thao tác:*      CX được giảm 1, nếu CX khác 0 và cờ ZF=0 thì điều khiển được chuyển cho lệnh có nhãn short\_label, trong trường hợp khác, điều khiển được chuyển cho lệnh tiếp theo.

*Cờ:*              Bị ảnh hưởng - không.

*Mã lệnh:* 1110000 disp

E0

## **MOV (MOVE).**

Chuyển dữ liệu.

*Dạng lệnh:*      MOV destination,source

*Thao tác:* Chép nội dung của toán hạng nguồn sang toán hạng đích.

*Cờ:*              Bị ảnh hưởng - không.

*Mã lệnh:* Từ thanh chứa vào ô nhớ:

1010001w addr\_low addr\_high

Từ ô nhớ vào thanh chứa:

1010000w addr\_low addr\_high.

Từ ô nhớ hay thanh ghi vào thanh ghi đoạn:

10001110 mod 0 seg r/m

Từ thanh ghi đoạn vào thanh ghi hay ô nhớ:

10001100 mod 0 seg r/m

Từ bộ nhớ hay thanh ghi vào thanh ghi/từ thanh ghi vào bộ nhớ:

100010d1 mod reg r/m (addr\_low addr\_high)

Từ dữ liệu trực tiếp vào thanh ghi:

1011w reg data (data\_high)

Từ dữ liệu trực tiếp vào thanh ghi hay ô nhớ:

1100011w mod 000 r/m data (data\_high)

## **MOVS/MOVSB/MOVSW (MOVe Byte or Word String).**

Chuyển dữ liệu trong bộ nhớ tại địa chỉ SI vào ô nhớ có địa chỉ ES:DI. Bằng cách dùng tiền tố REP có thể chuyển đồng thời nhiều byte hay word.

*Dạng lệnh:* MOVS des\_string,source\_string

hay

MOVSB

hay

MOVSW

*Thao tác:* Chuỗi byte (hay word) được chuyển tới toán hạng đích. Cả SI và DI đều tăng 1 (hoặc 2 với chuỗi word) nếu DF=0 và giảm đi 1 (hoặc 2 với chuỗi word) nếu DF=1.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* 1010010w

## **MUL (MULTiply).**

Thực hiện phép nhân không dấu.

*Dạng lệnh:* MUL source

*Thao tác:* Số nhân là toán hạng nguồn có thể là một ô nhớ hay một thanh ghi. Trong phép nhân với byte (toán hạng nguồn 8 bit) số bị nhân trong AL còn trong phép nhân với word (toán hạng nguồn 16 bit) số bị nhân trong AX. Tích số chứa trong AX (DX:AX trong trường hợp nhân với word). Các cờ CF và OF được thiết lập nếu nửa cao của tích khác 0.

*Cờ:* Bị ảnh hưởng - CF, OF.

Không xác định - AF, PF, ZF, SF.

*Mã lệnh:* 1111011w mod 100 r/m.

## **NEG (NEGate).**

Lấy số bù 2.

*Dạng lệnh:* NEG destination

*Thao tác:* Toán hạng đích bị trừ đi từ số gồm toàn chữ số 1 (0FFh với byte và 0FFFFh với word). Sau đó kết quả chứa trong toán hạng đích được cộng thêm 1.

*Cờ:* Bị ảnh hưởng - AF, CF, OF, PF, SF, ZF.

*Mã lệnh:* 111011w mod 011 r/m

### NOP (NO oPeration).

*Dạng lệnh:* NOP

*Thao tác:* không thực hiện thao tác nào.

*Cờ:* Bị ảnh hưởng - không

*Mã lệnh:* 10010000

90

### NOT (Logical Not)

*Dạng lệnh:* NOT destination

*Thao tác:* Tạo dạng bù 1 của toán hạng đích.

*Cờ:* Bị ảnh hưởng - không

*Mã lệnh:* 1111011w mod 010 r/m

### OR (Logical Inclusive Or)

*Dạng lệnh:* OR destination, source

*Thao tác:* Thực hiện phép hoặc logic trên từng bit của các toán hạng và đặt kết quả trong toán hạng đích.

*Cờ:* Bị ảnh hưởng - CF, OF, PF, SF, ZF

Không xác định - AF.

*Mã lệnh:* Ô nhớ hay thanh ghi với thanh ghi:

000010dw mod reg r/m

Dữ liệu trực tiếp với thanh chứa:

0000110w data

Dữ liệu trực tiếp với thanh ghi hay ô nhớ:

## OUT (Output Byte or Word)

*Dạng lệnh:* OUT thanh chứa, port.

*Thao tác:* Nội dung của cổng vào ra có chỉ định được thay thế bằng nội dung của thanh chứa. port có thể là một hàng số (cổng cố định) hay DX (số hiệu cổng thay đổi).

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* cổng cố định:

1110011w port

cổng thay đổi:

0001000w

## POP (Pop Word Off Stack to Destination)

Lấy dữ liệu ra khỏi ngăn xếp và đưa vào toán hạng đích.

*Dạng lệnh:* POP destination

*Thao tác:* Nội dung của toán hạng đích được thay bằng từ nằm ở đỉnh ngăn xếp. Con trỏ ngăn xếp tăng lên 2.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* Toán hạng đích là thanh ghi công dụng chung:

01011 reg

Thanh ghi đoạn:

000 seg 111

Ô nhớ hay thanh ghi:

10001111 mod 000 r/m

## POPF (Pop Flags Off Stack)

Lấy thanh ghi cờ ra khỏi ngăn xếp.

*Dạng lệnh:* POPF

*Thao tác:* Chuyển các bit cờ ở đỉnh ngăn xếp vào thanh ghi FLAGS sau đó tăng con trỏ ngăn xếp lên 2.

*Cờ:* Bị ảnh hưởng - tất cả.

*Mã lệnh:* 10011101

9D

### **PUSH (Push Word onto Stack)**

Chuyển một từ vào ngăn xếp.

*Dạng lệnh:* PUSH source

*Thao tác:* Giảm SP thêm 2 và chuyển từ dữ liệu từ source vào đỉnh mới của ngăn xếp.

*Cờ:* Không.

*Mã lệnh:* Toán hạng đích là thanh ghi công dụng chung:

01010 reg

Thanh ghi đoạn:

000 seg 110

Ô nhớ hay thanh ghi:

11111111 mod 110 r/m

### **PUSHF (Push Flags onto Stack)**

Chuyển thanh ghi cờ vào ngăn xếp.

*Dạng lệnh:* PUSHF

*Thao tác:* Giảm SP đi 2 và chuyển các bit cờ vào đỉnh ngăn xếp.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* 10011100

9C

### **RCL (Rotate Left through Carry)**

Quay toán hạng đích qua cờ CF một hoặc nhiều lần.

*Dạng lệnh:* RCL destination, 1

hay

RCL destination, CL

*Thao tác:* Dạng đầu tiên quay trái toán hạng đích qua cờ CF, kết quả là bit msb của nó được đặt vào CF và nội dung cũ của CF được chuyển vào bit lsb . Khi quay nhiều hơn một lần, số lần quay

được đặt trong CL. Khi bộ đếm bằng 1 và 2 bit trái nhất của toán hạng đích cũ bằng nhau thì cờ OF bị xoá, nếu chúng không bằng nhau thì cờ OF được thiết lập 1. Nếu bộ đếm khác 1, OF không xác định. CL không thay đổi.

*Cờ:* Bị ảnh hưởng - CF, OF.

*Mã lệnh:* 110100vw mod 010 r/m

nếu v=0, bộ đếm bằng 1

nếu v=1, bộ đếm bằng nội dung của CL.

## **RCR (Rotate Right through Carry)**

Quay toán hạng đích sang phải qua cờ CF một hoặc nhiều lần.

*Dạng lệnh:* RCR destination, 1

hay

RCR destination, CL

*Thao tác:* Dạng đầu tiên quay phải toán hạng đích qua cờ CF, kết quả là bit lsb của nó được đặt vào CF và nội dung cũ của CF được chuyển vào bit msb. Khi quay nhiều hơn một lần, số lần quay được đặt trong CL. Khi bộ đếm bằng 1 và 2 bit trái nhất của toán hạng đích cũ bằng nhau thì cờ OF bị xoá, nếu chúng không bằng nhau thì cờ OF được thiết lập 1. Nếu bộ đếm khác 1, OF không xác định. CL không thay đổi.

*Cờ:* Bị ảnh hưởng - CF, OF.

*Mã lệnh:* 110100vw mod 011 r/m

nếu v=0, bộ đếm bằng 1

nếu v=1, bộ đếm bằng nội dung của CL.

## **REP/REPZ/REPE/REPNE/RPENZ (Repeat String Operation)**

Lặp lại thao tác chuỗi. Thao tác chuỗi theo sau được lặp lại trong khi CX còn khác 0.

*Dạng lệnh:* REP/REPZ/REPE/REPNE/RPENZ string\_instruction

*Thao tác:* Thao tác chuỗi được thực hiện cho đến khi CX giảm xuống đến 0. Đối với các lệnh CMPS hay SCAS, cờ ZF cũng được dùng để kết công việc. Đối với các lệnh REP/ REPZ/REPE các lệnh CMPS hay SCAS được lặp lại nếu CX khác 0 và ZF=1. Với các

lệnh REPNE/REP NZ, các lệnh CMPS hay SCAS được lặp lại khi CX khác 0 và ZF=0.

*Cờ:* Bị ảnh hưởng - xem các lệnh thao tác chuỗi có liên quan.

*Mã lệnh:* REP/REPE/REPZ            11110011

REPNE/REP NZ 11110010

## RET (RETurn from procedure)

Trả lại điều khiển khi thủ tục được gọi thực hiện xong.

*Dạng lệnh:* RET (pop\_value)

*Thao tác:* Nếu lệnh RET trong một thủ tục NEAR, nó được dịch thành một lệnh trả về trong cùng đoạn, lệnh này cập nhật thanh ghi IP bằng cách lấy một từ ra khỏi ngăn xếp. Nếu lệnh RET trong một thủ tục FAR nó được dịch thành một lệnh trả về bên ngoài đoạn, lệnh này cập nhật cả 2 thanh ghi CS và IP bằng cách lấy 2 từ ra khỏi ngăn xếp. Con số tùy chọn pop\_value xác định số byte sẽ bị xoá đi khỏi ngăn xếp. Đó chính là số các tham số truyền cho thủ tục.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* Trong một đoạn:

11000011

Bên trong 1 đoạn với pop\_value:

11000010

Ngoài đoạn:

11001011

Ngoài đoạn với pop\_value:

11001010

## ROL (Rotate Left)

Quay trái toán hạng đích một hoặc nhiều lần.

*Dạng lệnh:* ROL destination, 1  
hay

ROL destination, CL

*Thao tác:* Dạng đầu tiên quay trái toán hạng đích một lần, CF nhận được bit msb. Khi quay nhiều hơn một lần, số lần quay được đặt trong CL. Khi bộ đếm bằng 1 và 2 bit trái nhất của toán hạng

dịch cũ bằng nhau thì cờ OF bị xoá, nếu chúng không bằng nhau thì cờ OF được thiết lập 1. Nếu bộ đếm khác 1, OF không xác định. CL không thay đổi.

*Cờ:* Bị ảnh hưởng - CF, OF.

*Mã lệnh:* 110100vw mod 000 r/m

nếu v=0, bộ đếm bằng 1  
nếu v=1, bộ đếm bằng nội dung của CL.

### **ROR (Rotate Right)**

Quay phải toán hạng đích một hoặc nhiều lần.

*Dạng lệnh:* ROR destination, 1

hay

ROR destination, CL

*Thao tác:* Dạng đầu tiên quay phải toán hạng đích một lần, CF nhận được bit lsb. Khi quay nhiều hơn một lần, số lần quay được đặt trong CL. Khi bộ đếm bằng 1 và 2 bit trái nhất của toán hạng đích cũ bằng nhau thì cờ OF bị xoá, nếu chúng không bằng nhau thì cờ OF được thiết lập 1. Nếu bộ đếm khác 1, OF không xác định. CL không thay đổi.

*Cờ:* Bị ảnh hưởng - CF, OF.

*Mã lệnh:* 110100vw mod 001 r/m

nếu v=0, bộ đếm bằng 1  
nếu v=1, bộ đếm bằng nội dung của CL.

### **SAHF (Store AH in FLAGS register)**

*Dạng lệnh:* SAHF

*Thao tác:* Lưu 5 bit thấp của AH vào byte thấp của thanh ghi FLAGS. Chỉ những bit tương ứng với cờ mới được chuyển. Các cờ trong bit thấp của thanh ghi FLAGS là SF=bit7, ZF=bit6, AF=bit4, PF=bit 2 và CF=bit0.

*Cờ:* Bị ảnh hưởng - AF, CF, PF, SF, ZF.

*Mã lệnh:* 10011110

## **SAL/SHL (Shift Arithmetic Left/SHift logical Left).**

*Dạng lệnh:* SAL/SHL destination, 1  
hay

SAL/SHL destination, CL

*Thao tác:* Dạng đầu dịch trái toán hạng đích một lần, CF nhận được bit msb và 0 được chuyển vào bit lsb. Để dịch nhiều hơn một lần, số lần dịch được đặt trong CL. Khi bộ đếm bằng 1 và giá trị mới của CF khác bit msb thì cờ OF được thiết lập, nếu chúng bằng nhau thì cờ OF bị xoá. Nếu bộ đếm khác 1, OF không xác định. CL không thay đổi.

*Cờ:* Bị ảnh hưởng - CF, OF, PF, SF, ZF.  
Không xác định - AF.

*Mã lệnh:* 110100vw mod 100 r/m

nếu v=0, bộ đếm bằng 1  
nếu v=1, bộ đếm bằng nội dung của CL.

## **SAR (Shift Arithmetic Right).**

*Dạng lệnh:* SAR destination, 1  
hay  
SAR destination, CL

*Thao tác:* Dạng đầu dịch phải toán hạng đích một lần, CF nhận được bit lsb và bit msb được giữ nguyên (không đổi dấu). Để dịch nhiều hơn một lần, số lần dịch được đặt trong CL. Khi bộ đếm bằng 1 thì cờ OF bị xoá. Nếu bộ đếm khác 1, OF không xác định. CL không thay đổi.

*Cờ:* Bị ảnh hưởng - CF, OF, PF, SF, ZF.  
Không xác định - AF.

*Mã lệnh:* 110100vw mod 111 r/m

nếu v=0, bộ đếm bằng 1  
nếu v=1, bộ đếm bằng nội dung của CL.

## **SBB (SuBtract with Borrow)**

Trừ có nhớ.

*Dạng lệnh:* SBB destination,source

*Thao tác:* Trừ toán hạng đích cho toán hạng nguồn và nếu CF=1 thì trừ kết quả nhận được đi 1. Kết quả chứa trong toán hạng đích.

*Cờ:* Bị ảnh hưởng - AF, CF, OF, PF, SF, ZF.

*Mã lệnh:* Trừ ô nhớ hay thanh ghi cho thanh ghi:

000110dw mod reg r/m

Trừ thanh chứa cho toán hạng trực tiếp:

0001110w data

Trừ ô nhớ hay thanh ghi cho toán hạng trực tiếp:

100000sw mod 011 r/m data

(s=1 nếu toán hạng trực tiếp là số 8 bit khi toán hạng đích là ô nhớ hay thanh ghi 16 bit.

### **SCAS/SCASB/SCASW (SCAn Byte or Word String).**

So sánh nội dung của ô nhớ với nội dung của thanh chứa. Có thể dùng tiền tố REP để so sánh nhiều ô nhớ với một giá trị nào đó.

*Dạng lệnh:* SCAS dest-string

hay

SCASB

hay

SCASW

*Thao tác:* Trừ AL (hay AX) đi byte (hay word) địa chỉ bởi DI, các cờ bị ảnh hưởng nhưng kết quả không được lưu. DI sẽ tăng (nếu DF=0) hay giảm (nếu DF=1) 1(với chuỗi byte) và 2 (với chuỗi word).

*Cờ:* Bị ảnh hưởng - AF, CF, OF, PF, SF, ZF.

*Mã lệnh:* 1010111w

### **SHR (SHift logical Right).**

*Dạng lệnh:* SHR destination, 1

hay

SHR destination, CL

*Thao tác:* Dạng đầu dịch phải toán hạng đích một lần, CF nhận được bit lsb và 0 được dịch vào bit msb. Để dịch nhiều hơn một lần, số lần dịch được đặt trong CL. Khi bộ đếm bằng 1 và 2 bit trái nhất của toán hạng đích bằng nhau thì cờ OF bị xoá. Nếu chúng khác nhau, OF được thiết lập 1. Khi bộ đếm khác 1, OF không xác định. CL không thay đổi.

*Cờ:* Bị ảnh hưởng - CF, OF, PF, SF, ZF.

Không xác định - AF.

*Mã lệnh:* 110100vw mod 101 r/m

nếu v=0, bộ đếm bằng 1

nếu v=1, bộ đếm bằng nội dung của CL.

### **STC (SeT Carry flag)**

*Dạng lệnh:* STC

*Thao tác:* CF được thiết lập 1.

*Cờ:* Bị ảnh hưởng - CF

*Mã lệnh:* 11111001

F9

### **STD (SeT Direction flag)**

*Dạng lệnh:* STD

*Thao tác:* DF được thiết lập 1.

*Cờ:* Bị ảnh hưởng - DF

*Mã lệnh:* 11111101

FD

### **STI (SeT Interrupt flag)**

*Dạng lệnh:* STI

*Thao tác:* IF được thiết lập 1.

*Cờ:* Bị ảnh hưởng - IF

*Mã lệnh:* 11111011

**STOS/TOSB/STOSW (STOre Byte or Word String).**

Lưu nội dung thanh chứa vào bộ nhớ. Có thể dùng tiền tố REP để lưu vào các ô nhớ với cùng một giá trị.

*Dạng lệnh:* STOS dest\_string

hay

STOSB

hay

STOSW

*Thao tác:* chứa nội dung của AL (AX) vào trong byte (hay word) đích có địa chỉ trong DI. DI được tăng (nếu DF=0) hay giảm (nếu DF=1) 1 (với chuỗi byte), 2 (với chuỗi word).

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* 1010101w

**SUB (SUBtract)**

*Dạng lệnh:* SUB destination, source

*Thao tác:* Trừ toán hạng nguồn cho toán hạng đích. Kết quả được đặt trong toán hạng đích.

*Cờ:* Bị ảnh hưởng - AF, CF, OF, PF, SF, ZF.

*Mã lệnh:* Trừ ô nhớ hay thanh ghi cho thanh ghi:

100010dw mod reg r/m

Trừ thanh chứa cho toán hạng trực tiếp:

0010110w data

Trừ ô nhớ hay thanh ghi cho toán hạng trực tiếp:

100000sw mod 101 r/m data

(s=1 nếu toán hạng trực tiếp là số 8 bit khi toán hạng đích là ô nhớ hay thanh ghi 16 bit).

**TEST: Test (Logical Compare)**

So sánh logic.

*Dạng lệnh:* TEST destination, source

*Thao tác:* 2 toán hạng được và logic với nhau làm ảnh hưởng tới cờ, các toán hạng vẫn không thay đổi.

*Cờ:* Bị ảnh hưởng - CF, OF, PF, SF, ZF.

Không xác định - AF.

*Mã lệnh:* Ô nhớ hay thanh ghi với thanh ghi

1000010w mod reg r/m

toán hạng trực tiếp với thanh chứa:

1010100w data

Toán hạng trực tiếp với ô nhớ hay thanh ghi:

1111011w mod 000 r/m data.

## **WAIT**

*Dạng lệnh:* WAIT

*Thao tác:* bộ vi xử lý ở trạng thái chờ cho đến khi được kích hoạt bằng một ngắt ngoài.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* 10011011

9B

## **XCHG (EXCHanGe).**

*Dạng lệnh:* XCHG destination, source

*Thao tác:* Toán hạng đích và toán hạng nguồn được đổi lẩn cho nhau.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* Thanh ghi với thanh chứa:

10010 reg

Ô nhớ hay thanh ghi với thanh ghi:

1000011w mod reg r/m.

## **XLAT (Translate)**

*Dạng lệnh:* XLAT source\_table

*Thao tác:* BX phải chứa địa chỉ offset của bảng nguồn có chiều dài tối đa là 256 byte. AL phải chứa chỉ số của các phần tử bảng. Lệnh sẽ thay thế nội dung của AL bằng nội dung của các phần tử bảng được định địa chỉ bằng BX và AL.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* 110101111

D7

## **XOR (eXclusive OR).**

*Dạng lệnh:* XOR destination,source

*Thao tác:* XOR thực hiện phép hoặc ngoại trừ các toán tử nguồn và đích. Kết quả được lưu trong toán hạng đích. CF và OF bị xoá.

*Cờ:* Bị ảnh hưởng - CF, OF, Pf, SF, ZF.

Không xác định - AF.

*Mã lệnh:* ô nhớ hay thanh ghi với thanh ghi

001100dw mod reg r/m  
toán hạng trực tiếp với thanh chứa:

0011010w data

Toán hạng trực tiếp với ô nhớ hay thanh ghi:  
1000000w mod 110 r/m data.

## **F3. Tập lệnh của 8087.**

8087 sử dụng một số kiểu dữ liệu. Khi chuyển dữ liệu đến hoặc từ bộ nhớ, các khai báo kiểu dữ liệu bộ nhớ quyết định dạng số liệu 8087 thao tác. Bảng F.3 trình bày sự liên hệ giữa các dạng dữ liệu của 8087 và các khai báo dữ liệu bộ nhớ. Trong phần này chúng tôi chỉ trình bày những lệnh thao tác số học đơn giản của 8087. Các bạn có thể xem thêm tài liệu tham khảo về 8087 để biết thêm các lệnh khác.

| Dạng số liệu   | kích thước (bit) | Khai báo bộ nhớ | kiểu con trỏ |
|----------------|------------------|-----------------|--------------|
| Word integer   | 16               | DW              | WORD PTR     |
| Short integer  | 32               | DD              | DWORD PTR    |
| Long integer   | 64               | DQ              | QWORD PTR    |
| Packed decimal | 80               | DT              | TBYTE PTR    |
| Short real     | 32               | DD              | DWORD PTR    |
| Long real      | 64               | DQ              | QWORD PTR    |
| Temporary real | 80               | DT              | TBYTE PTR    |

Bảng F.3 Các kiểu dữ liệu của 8087.

### FADD (Add Real)

Cộng số thực.

*Dạng lệnh:* FADD hay

FADD source hay

FADD destination,source

*Thao tác:*

Cộng toán hạng nguồn vào toán hạng đích. Trong dạng thứ nhất, toán hạng nguồn là đinh ngăn xếp và toán hạng đích là ST(1). Đinh ngăn xếp bị đưa ra khỏi ngăn xếp và giá trị của nó được cộng vào đinh mới. Trong dạng thứ 2, toán hạng nguồn có thể là số kiểu short real hay long real trong bộ nhớ, toán hạng đích là đinh ngăn xếp. Trong dạng thứ 3, một trong các toán hạng là đinh ngăn xếp và toán hạng còn lại là một thanh ghi ngăn xếp, đinh ngăn xếp vẫn không bị lấy ra khi thực hiện lệnh.

### FBLD (Packed Decimal Load)

Nạp số thập phân dạng nén.

*Dạng lệnh:* FBLD source

*Thao tác:* Nạp số thập phân dạng nén vào đinh ngăn xếp. Toán hạng nguồn có kiểu DT (10 byte).

## **FBSTP (Packed BCD Store and Pop)**

Đưa ra khỏi ngăn xếp số BCD nén và lưu vào bộ nhớ.

*Dạng lệnh:* FBSTP destination

*Thao tác:* Đổi định ngăn xếp thành dạng BCD nén và lưu nó vào trong bộ nhớ, sau đó lấy ra khỏi ngăn xếp (pop).

## **FDIV (Divide Real)**

Chia số thực.

*Dạng lệnh:* FDIV

hay

FDIV source

hay

FDIV destination, source

*Thao tác:* Chia toán hạng nguồn cho toán hạng đích. Trong dạng thứ nhất, toán hạng nguồn là định ngăn xếp và toán hạng đích là ST(1). Định ngăn xếp bị đưa ra khỏi ngăn xếp và giá trị của nó được đem chia cho định mới. Trong dạng thứ 2, toán hạng nguồn có thể là số kiểu short real hay long real trong bộ nhớ, toán hạng đích là định ngăn xếp. Trong dạng thứ 3, một trong các toán hạng là định ngăn xếp và toán hạng còn lại là một thanh ghi ngăn xếp, định ngăn xếp vẫn không bị lấy ra khi thực hiện lệnh.

## **FIADD (Integer Add)**

Cộng số nguyên.

*Dạng lệnh:* FIADD source

*Thao tác:* Cộng toán hạng nguồn (có thể là short integer hay word integer) vào định ngăn xếp.

## **FIDIV (Interger Divide)**

Chia số nguyên.

*Dạng lệnh:* FIDIV source

*Thao tác:* Chia toán hạng nguồn (có thể là short integer hay word integer) cho định ngăn xếp.

## **FILD (Integer Load)**

Nạp số nguyên.

*Dạng lệnh:* FILD source

*Thao tác:* Nạp toán hạng nguồn (có thể là short integer, word integer hay long integer) vào đinh ngắn xếp.

## **FIMUL (Integer Multiply)**

Nhân số nguyên.

*Dạng lệnh:* FIMUL source

*Thao tác:* Nhân toán hạng nguồn (có thể là short integer hay word integer) với đinh ngắn xếp.

## **FIST (Integer Store)**

Lưu số nguyên.

*Dạng lệnh:* FIST destination

*Thao tác:* Làm tròn đinh ngắn xếp thành dạng số nguyên và lưu nó vào một ô nhớ. Toán hạng đích có thể là short integer hay word integer. Ngắn xếp không bị lấy ra (pop).

## **FISTP (Integer Store and Pop)**

Lưu số nguyên và lấy ra khỏi ngắn xếp.

*Dạng lệnh:* FIST destination

*Thao tác:* Làm tròn đinh ngắn xếp thành dạng số nguyên và lưu nó vào một ô nhớ sau đó ngắn xếp bị lấy ra (pop). Toán hạng đích có thể là short integer, long integer hay word integer.

## **FISUB (Intiger Subtract)**

Trừ số nguyên.

*Dạng lệnh:* FISUB source

*Thao tác:* Trừ đinh ngắn xếp cho toán hạng nguồn (có thể là short integer hay word integer).

## **FLD (Load Real)**

Nạp số thực.

*Dạng lệnh:* FLD source

*Thao tác:* Nạp một toán hạng kiểu thực vào đindh ngän xếp. Toán hạng nguồn có thể là một thanh ghi ngän xếp ST(i) hay một ô nhớ. Trong trường hợp là ô nhớ nó có thể có bất cứ kiểu thực nào.

## **FMUL (Multiply Real)**

Nhân số thực.

*Dạng lệnh:* FMUL

hay

FMUL source

hay

FMUL destination, source

*Thao tác:* Nhân toán hạng nguồn với toán hạng đích. Trong dạng thứ nhất, toán hạng nguồn là đindh ngän xếp và toán hạng đích là ST(1). Đindh ngän xếp bị đưa ra khỏi ngän xếp và giá trị của nó được đem nhân với đindh mới. Trong dạng thứ 2, toán hạng nguồn có thể là số kiểu short real hay long real trong bộ nhớ, toán hạng đích là đindh ngän xếp. Trong dạng thứ 3, một trong các toán hạng là đindh ngän xếp và toán hạng còn lại là một thanh ghi ngän xếp, đindh ngän xếp vẫn không bị lấy ra khi thực hiện lệnh.

## **FST (Store Real)**

Lưu số thực.

*Dạng lệnh:* FST destination

*Thao tác:* Lưu đindh của ngän xếp vào một ô nhớ hay một thanh ghi khác của ngän xếp. Toán hạng đích trong trường hợp là một ô nhớ có thể có kiểu short real ( double word), long real (quad word) hay temporary real (10 byte). Đindh ngän xếp không bị pop.

## **FSTP (Store Real and Pop)**

Lưu số thực và lấy ra khỏi ngän xếp.

- Dạng lệnh:* FSTP destination
- Thao tác:* Lưu đỉnh của ngăn xếp vào một ô nhớ hay một thanh ghi khác của ngăn xếp. Đỉnh ngăn xếp sau đó được lấy ra (pop). Toán hạng đích trong trường hợp là một ô nhớ có thể có kiểu short real (double word), long real (quad word) hay temporary real (10 byte). Đỉnh ngăn xếp không bị pop.

### FSUB (Subtract Real)

Trừ số thực.

- Dạng lệnh:* FSUB  
hay  
FSUB source  
hay  
FSUB destination, source

- Thao tác:* Trừ toán hạng đích đi toán hạng nguồn. Trong dạng thứ nhất, toán hạng nguồn là đỉnh ngăn xếp và toán hạng đích là ST(1). Đỉnh ngăn xếp bị đưa ra khỏi ngăn xếp và giá trị của nó bị trừ đi bởi đỉnh mới. Trong dạng thứ 2, toán hạng nguồn có thể là số kiểu short real hay long real trong bộ nhớ, toán hạng đích là đỉnh ngăn xếp. Trong dạng thứ 3, một trong các toán hạng là đỉnh ngăn xếp và toán hạng còn lại là một thanh ghi ngăn xếp, đỉnh ngăn xếp vẫn không bị lấy ra khi thực hiện lệnh.

## F.4 Các lệnh của 80286.

Tập lệnh của 80286 trong chế độ địa chỉ thực bao gồm tất cả các lệnh của 8086 cộng thêm tập lệnh mở rộng. Tập lệnh mở rộng bao gồm 5 nhóm lệnh chính là (1) nhân với toán hạng trực tiếp (IMUL), (2) Xuất và nhập các chuỗi (INS và OUTS), (3) các thao tác ngăn xếp (POPA, PUSH toán hạng trực tiếp và PUSHA), (4) quay và dịch với số đếm ở dạng toán hạng trực tiếp và (5) Các lệnh để dịch cấu trúc của ngôn ngữ bậc cao (BOUND, ENTER). Chúng ta chỉ xem xét các lệnh trong nhóm từ 1-4.

## **IMUL (Integer Immediate MULtiply).**

Nhân trực tiếp số nguyên

*Dạng lệnh:*      IMUL destination, immediate

hay

                IMUL destination, source, immediate

*Thao tác:*      Trong dạng thứ nhất toán hạng trực tiếp phải là một byte được nhân với toán hạng đích phải là một thanh ghi 16 bit. 16 bit thấp của tích được lưu trong thanh ghi. Với dạng thứ 2, toán hạng trực tiếp dạng 8 hay 16 bit được nhân với toán hạng nguồn có thể là thanh ghi 16 bit hay một từ nhớ. 16 bit thấp của tích được lưu trong toán hạng đích phải là một thanh ghi 16 bit. Các cờ CF và OF được thiết lập nếu phần cao của tích không phải là mở rộng dấu của phần thấp.

*Cờ:*              Bị ảnh hưởng - CF, OF.

Không xác định - AF, PF, SF, ZF

*Mã lệnh:* 011010sl mod reg r/m data (data nếu s=0).

## **INS/INSB/INSW (INput from port to String)**

Chuyển các phần tử của chuỗi byte hay word từ cổng vào bộ nhớ. Có thể dùng tiền tố REP để chuyển đồng thời nhiều byte hay word.

*Dạng lệnh:*      INS destination\_string, port

hay

      INSB

hay

      INSW

*Thao tác:*      Một byte hay word được chuyển từ cổng vào ra được chỉ định bởi DX đến ô nhớ được định địa chỉ bởi ES:DI. Sau đó DI được tăng 1 (2 đối với chuỗi word) nếu DF=0. Nếu không, DI bị giảm 1 (2 với chuỗi word).

*Cờ:*              Bị ảnh hưởng - không.

*Mã lệnh:* 0110110w

## **OUTS/OUTSB/OUTSW (OUTput String to port)**

Chuyển các phần tử của chuỗi byte hay word từ bộ nhớ ra cổng. Có thể dùng tiền tố REP để chuyển đồng thời nhiều byte hay word.

*Dạng lệnh:* OUTS destination\_string, port

hay

OUTSB

hay

OUTSW

*Thao tác:* Một byte hay word được chuyển từ ô nhớ được định địa chỉ bởi DS:SI ra cổng vào ra được chỉ định bởi DX. Sau đó SI được tăng 1 (2 đối với chuỗi word) nếu DF=0. Nếu không, SI bị giảm 1 (2 với chuỗi word).

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* 0110111w

## **POPA (POP All general register).**

*Dạng lệnh:* POPA

*Thao tác:* Các thanh ghi được lấy ra khỏi ngăn xếp theo trình tự: DI, SI, BP, SP, BX, DX, CX, AX.

*Mã lệnh:* 01100001

61

## **PUSH**

Nạp một toán hạng trực tiếp vào ngăn xếp.

*Dạng lệnh:* PUSH data

*Thao tác:* Dữ liệu có thể có 8 hay 16 bit. Trong trường hợp là số 8 bit nó được mở rộng dấu thành 16 bit trước khi nạp vào ngăn xếp.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* 011010s0 data (data nếu s=0).

## PUSHA (PUSH All general register).

Dạng lệnh: PUSHA

Thao tác: Các thanh ghi được đưa vào ngăn xếp theo trình tự: AX, CX, DX, BX, giá trị ban đầu của SP, BP, SI và DI.

Cờ: Bị ảnh hưởng - không.

Mã lệnh: C110000C

60

Dạng chung của các lệnh dịch và quay với bộ đếm là toán hạng trực tiếp như sau:

opcode destination, immediate

Trong đó opcode là một trong số RCL, RCR, ROL, ROR, SAL, SAR, SHL và SHR. Nếu giá trị immediate là 1 thì lệnh trở thành lệnh tương ứng của 8086. Với các giá trị trực tiếp khác 1 (từ 2 đến 31) lệnh làm việc giống như lệnh tương ứng của 8086 với CL chứa giá trị trực tiếp. 80286 không cho phép giá trị trực tiếp lớn hơn 31.

Mã lệnh của chúng là:

RCL

1100000w mod 010 r/m

RCR

1100000w mod 011 r/m

ROL

1100000w mod 000 r/m

ROR

1100000w mod 001 r/m

SAL/SHL

1100000w mod 100 r/m

SAR

1100000w mod 111 r/m

SHR

1100000w mod 101 r/m

## F.5 Tập lệnh của 80386.

Tập lệnh của 80386 trong chế độ địa chỉ thực bao gồm tập lệnh của 80286 ở chế độ địa chỉ thực thêm vào đó là phần mở rộng 32 bit đồng thời còn có thêm 6 nhóm lệnh mới là (1) bit scan, (2) bit test, (3) chuyển dữ liệu có mở rộng, (4) thiết lập byte theo điều kiện, (5) dịch số có độ chính xác kép và (6) chuyển từ hoặc chuyển tới các thanh ghi đặc biệt. Chúng ta sẽ chỉ xem xét các lệnh trong nhóm từ 1-5.

### Các lệnh quét bit (bit scan)

Các lệnh quét bit là BSF (Bit Scan Forward) và BSR (Bit Scan Reverse). Chúng được dùng để tìm trong toán hạng bit thiết lập đầu tiên và chúng chỉ khác nhau ở hướng quét.

*Dạng lệnh:*      BSF destination, source  
                        hay  
                        BSR destination, source

*Thao tác:*      Toán hạng đích phải là một thanh ghi trong khi toán hạng nguồn có thể là một thanh ghi hay một ô nhớ. Chúng phải đồng thời có kiểu word hay doubleword. Toán hạng nguồn được quét để xác định bit thiết lập đầu tiên, nếu tất cả các bit của nó đều bằng 0, cờ ZF bị xoá. Trong trường hợp còn lại ZF được thiết lập và toán hạng đích chứa vị trí của bit thiết lập đầu tiên trong toán hạng nguồn. Trong lệnh BSF toán hạng nguồn được quét từ bit 0 đến bit msb còn trong lệnh BSR nó được quét từ bit msb đến bit 0.

*Cờ:*              Bị ảnh hưởng - ZF

*Mã lệnh:* BSF  
  
00001111 10111101 mod reg r/m  
BSR  
00001111 10111101 mod reg r/m

### Các lệnh kiểm tra bit (bit test).

Các lệnh kiểm tra bit là BT (Bit Test), BTC (Bit Test and Complement), BTR (Bit Test and Reset) và BTS (Bit Test and Set). Chúng được dùng để chép bit vào cờ CF nhờ đó có thể dùng các lệnh JC hay JNC để kiểm tra bit.

*Dạng lệnh:* BT destination, source  
hay  
BTC destination, source  
hay  
BTR destination, source  
hay  
BTS destination, source

*Thao tác:* Toán hạng nguồn xác định vị trí bit trong toán hạng đích sẽ được chép vào cờ CF. BT chỉ đơn giản thực hiện việc sao chép, BTC chép bit và đảo nó ở toán hạng đích, BTR chép và thiết lập lại nó trong toán hạng đích, BTS chép và thiết lập nó trong toán hạng đích. Toán hạng nguồn có thể là thanh ghi 16 hay 32 bit hoặc là một hằng số 8 bit. Toán hạng đích có thể là thanh ghi hay ô nhớ 16 hoặc 32 bit. Nếu toán hạng nguồn là thanh ghi thì toán hạng nguồn và toán hạng đích phải có cùng kích thước.

*Cờ:* Bị ảnh hưởng - CF

*Mã lệnh:* Toán hạng nguồn là dữ liệu trực tiếp 8 bit:

BT  
00001111 10111010 mod 100 r/m  
BTC  
00001111 10111010 mod 111 r/m  
BTR  
00001111 10111010 mod 110 r/m  
BTS  
00001111 10111010 mod 101 r/m  
toán hạng nguồn là thanh ghi  
BT  
00001111 10100011 mod reg r/m  
BTC  
00001111 10111011 mod reg r/m  
BTR  
00001111 10110011 mod reg r/m  
BTS  
00001111 10101011 mod reg r/m

## Các lệnh chuyển dữ liệu có mở rộng

Các lệnh chuyển dữ liệu có mở rộng là MOVSX (Move with sign extend) và MOVZX (Move with zero extend). Các lệnh này chuyển toán hạng nguồn vào

toán hạng đích có kích thước lớn hơn và mở rộng ra nữa cao của toán hạng đích với dấu hay các byte 0.

*Dạng lệnh:* MOVSX destination, source

hay

MOVZX destination, source

*Thao tác:* Toán hạng đích phải là một thanh ghi, toán hạng nguồn có thể là một thanh ghi hay một ô nhớ. Nếu toán hạng nguồn là byte hay word thì toán hạng đích là word hay doubleword. Lệnh MOVSX chép và mở rộng dấu toán hạng nguồn vào toán hạng đích. Lệnh MOVZX chép và mở rộng với các byte 0 vào toán hạng đích.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* MOVSX

00001111 1011111w mod reg r/m

MOVZX

00001111 1011011w mod reg r/m

### Các lệnh thiết lập byte theo điều kiện

Các lệnh thiết lập byte theo điều kiện thiết lập byte đích thành 1 nếu điều kiện đúng và xoá nó nếu điều kiện sai.

*Dạng lệnh:* SET (condition) destination

*Thao tác:* toán hạng đích có thể là thanh ghi hay ô nhớ 8 bit. Nó được thiết lập thành 1 nếu điều kiện đúng và bị xoá thành 0 nếu điều kiện sai.

*Cờ:* Bị ảnh hưởng - không.

*Mã lệnh:* 00001111 opcode mod 000 r/m

(byte opcode được cho trong bảng dưới đây ở dạng hex)

| LỆNH  | THIẾT LẬP NẾU  | ĐIỀU KIỆN    | MÃ LỆNH |
|-------|----------------|--------------|---------|
| SETA  | Above          | CF=0 và ZF=0 | 97      |
| SETAE | Above or Equal | CF=0         | 93      |
| SETB  | Below          | CF=1         | 92      |

|        |                       |                    |    |
|--------|-----------------------|--------------------|----|
| SETBE  | Below or Equal        | CF=1 hay ZF=1      | 96 |
| SETC   | Carry                 | CF=1               | 92 |
| SETE   | Equal                 | ZF=1               | 94 |
| SETG   | Greater               | ZF=0 và SF=OF      | 9F |
| SETGE  | Greater or Equal      | ZF=OF              | 9D |
| SETL   | Less                  | (SF xor OF)=1      | 9C |
| SETLE  | Less or Equal         | SF xor OF)or ZF=1  | 9E |
| SETNA  | Not Above             | CF=1 hay ZF=1      | 96 |
| SETNAE | Not Above or Equal    | CF=1               | 92 |
| SETNB  | Not Below             | CF=0               | 93 |
| SETNBE | Not Below or Equal    | CF=0 hay ZF=0      | 97 |
| SETNC  | Not Carry             | CF=0               | 93 |
| SETNE  | Not Equal             | ZF=0               | 95 |
| SETNG  | Not Greater           | (SF xor OF)or ZF=1 | 9E |
| SETNGE | Not Greater nor Equal | (SF xor OF)=1      | 9C |
| SETNL  | Not less              | SF = OF            | 9D |
| SETNLE | Not less nor Equal    | ZF=0 và SF=OF      | 9F |
| SETNO  | Not Overflow          | OF=0               | 91 |
| SETNP  | Not Parity            | PF=0               | 9B |
| SETNS  | Not Sign              | SF=0               | 99 |
| SETNZ  | Not Zero              | ZF=0               | 95 |
| SETO   | Overflow              | OF=1               | 90 |
| SETP   | Parity                | PF=1               | 9A |
| SETPE  | Parity Even           | PF=1               | 9A |
| SETPO  | Parity Odd            | PF=0               | 9B |
| SETS   | Sign                  | SF=1               | 98 |
| SETZ   | Zero                  | ZF=1               | 94 |

### Các lệnh dịch số có độ chính xác kép.

Các lệnh dịch số có độ chính xác kép là SHLD (Double\_ precision SHift Left) và SHRD (Double\_ precision SHift Right).

Dạng lệnh: SHLD destination, source, count

hay

SHRD destination, source, count

*Thao tác:* Toán hạng đích có thể là thanh ghi hay ô nhớ. Toán hạng nguồn phải là một thanh ghi và chúng phải có cùng kích thước (16 hay 32 bit). Số đếm có thể là một hằng số 8 bit hay CL. Số đếm xác định số lần dịch toán hạng đích. Thay vì các bit 0 được dịch vào như đối với số có độ chính xác đơn trong trường hợp này các bit được dịch vào toán hạng đích từ toán hạng nguồn. Tuy nhiên toán hạng nguồn vẫn không bị thay đổi. Các cờ SF, PF và ZF được thiết lập theo kết quả. CF được thiết lập theo bit cuối cùng bị dịch ra khỏi toán hạng đích. OF và AF không xác định.

*Cờ:*  
Bị ảnh hưởng - PF, SF, ZF  
Không xác định - AF, OF.

*Mã lệnh:* Số đếm là dữ liệu trực tiếp:

SHLD  
00001111 10100100 mod reg r/m [ disp] data  
SHRD  
00001111 10101100 mod reg r/m [ disp] data  
Số đếm là CL:  
SHLD  
00001111 10100101 mod reg r/m [ disp].  
SHRD  
00001111 10101101 mod reg r/m [ disp]

## Phục lục G.

# CÁC DẪN HƯỚNG BIÊN DỊCH.

Trong phụ lục này chúng tôi sẽ mô tả các dẫn hướng biên dịch quan trọng nhất. Để giải thích cú pháp, chúng tôi sẽ sử dụng các ký hiệu sau đây:

- | phân cách các chọn lựa
- { } bao quanh các mục không bắt buộc
- [ ] lặp lại các phần được bao quanh 0 hay một vài lần.

Nếu cú pháp không được đưa ra có nghĩa là dẫn hướng biên dịch không yêu cầu hoặc không có các tham số chọn lựa.

### .ALPHA

Báo cho chương trình biên dịch sắp xếp các đoạn theo thứ tự A\_n\_pha\_B. Dẫn hướng này đặt trước các định nghĩa đoạn.

### ASSUME

#### Cú pháp:

```
ASSUME segment_register:name[,segment_register:name]
```

Báo cho chương trình biên dịch liên kết một thanh ghi đoạn với một tên.

#### Ví dụ:

```
ASSUME CS:C_SEG, DS:D_SEG, SS:S_SEG, ES:E_SEG
```

Chú ý: Tên NOTHING loại bỏ liên kết hiện thời của thanh ghi đoạn. Đặc biệt, ASSUME NOTHING loại bỏ liên kết thanh ghi đoạn được tạo nên bởi câu lệnh ASSUME trước đó.

## **.CODE**

**Cú pháp:**

```
.CODE { name}
```

Một dẫn hướng đoạn ngắn gọn (MASM 5.0) để định nghĩa một đoạn lệnh.

## **.COMM**

**Cú pháp:**

```
.COMM definition [,definition]
```

Ở đây definition có cú pháp

```
NEAR|FAR label:size{ :count}
```

label là một tên biến,

size có thể là BYTE, WORD, DWORD, QWORD hay TBYTE,

count là số các phần tử chứa trong biến (mặc định =1).

Dẫn hướng dùng để định nghĩa một biến công cộng. Một biến như thế có cả hai thuộc tính PUBLIC và EXTRN và vì vậy ta có thể sử dụng nó trong các module Assembly khác nhau.

**Ví dụ:**

```
COMM NEAR WORD1:WORD
```

```
COMM FAR ARR1:BYTE:10,ARR2:BYTE:20
```

## **COMMENT**

**Cú pháp:**

```
COMMENT delimiter { text}
{ text}
delimiter { text}
```

Ở đây mốc (delimiter) là ký tự khác trống đầu tiên sau dẫn hướng COMMENT. Thường được sử dụng để định nghĩa một lời bình. Chương trình biên dịch sẽ lờ đi đoạn văn bản giữa các mốc thứ nhất và thứ hai. Văn bản trong cùng dòng với mốc thứ hai cũng bị bỏ qua.

**Ví dụ:**

```
COMMENT * Uses an asterisk as the delimiter. All
this text is ignored *
COMMENT * This text and the following instruction
is ignored too *
 MOV AX, BX
```

## .CONST

Một dẫn hướng sử dụng trong kiểu định nghĩa đoạn đơn giản hoá để định nghĩa một đoạn chứa các dữ liệu không bị thay đổi bởi chương trình. Chủ yếu được dùng trong các chương trình viết bằng Hợp ngữ được gọi bởi một ngôn ngữ bậc cao.

## .CREF và .XCREF

Cú pháp:

```
.CREF { name [,name] }
.XCREF { name [,name] }
```

.CREF dẫn hướng phát sinh file tham khảo ngang (.CRF) chứa các tên trong một chương trình. .CREF không có tham số tạo ra file tham khảo ngang chứa tất cả các tên. Đây là dẫn hướng mặc định.

.XCREF ngừng việc tham khảo ngang nói chung hoặc chỉ ngừng với các tên xác định.

Ví dụ:

```
.XREF ;ngừng tham khảo ngang
```

```
CRE F ;tiếp tục việc tham khảo ngang
```

```
.XREF NAME1,NAME2 ;ngừng việc tham khảo ngang với
;NAME1 và NAME2
```

## .DATA và .DATA?

Các dẫn hướng sử dụng trong kiểu định nghĩa đoạn đơn giản hoá để định nghĩa các đoạn dữ liệu. .DATA định nghĩa một đoạn dữ liệu khởi tạo trước và .DATA? định nghĩa một đoạn dữ liệu không khởi tạo. Dữ liệu không khởi tạo bao gồm các biến được định nghĩa với ?. .DATA? chủ yếu được dùng trong các chương

trình viết bằng Hợp ngữ được gọi từ một ngôn ngữ bậc cao. Trong một chương trình Assembly thuần nhất, đoạn .DATA có thể chứa các dữ liệu không khởi tạo.

### Các dẫn hướng định nghĩa dữ liệu.

| Dẫn hướng | ý nghĩa                                                      |
|-----------|--------------------------------------------------------------|
| DB        | định nghĩa byte                                              |
| DD        | định nghĩa doubleword (4 byte)                               |
| DF        | định nghĩa farword (6 byte); chỉ dùng với bộ vi xử lý 80386. |
| DQ        | định nghĩa quadword (8 byte)                                 |
| DT        | định nghĩa tenbyte (10 byte)                                 |
| DW        | định nghĩa word (2 byte)                                     |

### Cú pháp:

```
{ name} derective initializer [,initializer]
```

Ở đây, name là một tên biến. Nếu tên bị bỏ qua, ô nhớ được dành ra nhưng không có tên nào gán được cho nó. Initializer là một hằng số, biểu thức hằng hay ?. Các giá trị lặp lại có thể được định nghĩa bằng cách dùng toán hạng DUP. Bạn hãy xem chương 10.

## DOSSEG

Báo cho chương trình biên dịch chấp nhận quy ước về trình tự các đoạn. Trong một chương trình kiểu SMALL, trình tự này là : đoạn lệnh, đoạn dữ liệu và đoạn ngắn xếp. Dẫn hướng biên dịch này cần đặt trước mọi định nghĩa đoạn.

## ELSE

Sử dụng trong một khối điều kiện. Cú pháp là:

Condition

statements1

ELSE

statement2

ENDIF

nếu điều kiện (condition) là đúng, các dòng lệnh statement được biên dịch; nếu điều kiện sai, các dòng lệnh statement2 được biên dịch. Về dạng thức của điều kiện bạn hãy xem chương 13.

## **END**

### **Cú pháp:**

`END { start_address}`

Kết thúc một chương trình nguồn. Start\_address là một tên tại vị trí bắt đầu thi hành khi chương trình được nạp vào bộ nhớ. Đối với một chương trình chỉ có một module nguồn, địa chỉ đầu (start\_address) thường là tên của thủ tục chính hay một nhãn xác định câu lệnh đầu tiên. Đối với một chương trình gồm có nhiều module, mỗi module phải có một dấu hướng END nhưng chỉ một trong chúng được phép xác định một địa chỉ đầu.

## **ENDIF**

Kết thúc khối điều kiện. Xem chương 13.

## **ENDM**

Kết thúc một macro hay một khối lặp. Xem MACRO và REPT.

## **ENDP**

Kết thúc một thủ tục. Xem PROC.

## **ENDS**

Kết thúc một đoạn hay cấu trúc. Xem SEGMENT và STRUC.

## **EQU**

### **Cú pháp:**

Có hai dạng, đẳng thức dạng số và đẳng thức dạng chuỗi. Một đẳng thức dạng số có dạng:

`name EQU numeric_expression`

Một dãy thức dạng chuỗi có dạng:

```
name EQU <string>
```

Dãy hướng EQU gán biểu thức sau EQU cho hằng số name. Biểu thức số (numeric\_expression) phải tương đương với một con số. Những nơi có mặt name trong chương trình sẽ được trình biên dịch thay thế bởi biểu thức số hay chuỗi (string). Không có ô nhớ nào được dành ra cho các tên (name) và chúng ta không thể định nghĩa lại các tên này.

**Ví dụ:**

```
MAX EQU 32767
MIN EQU MAX-10
PROMPT EQU <'type a line of text $'>
ARG EQU <[DI+2] >
```

Dùng trong chương trình:

```
.DATA
 MSG DB PROMPT
.CODE
MAIN PROC
 MOV AX,MIN ;tương đương với MOV AX,32757
 MOV BX,ARG ;tương đương với MOV BX,[DI+]
```

**=(bằng)**

**Cú pháp:**

```
name = expresion
```

trong đó biểu thức (expression) là một số nguyên, biểu thức hằng hay một hằng chuỗi một hoặc hai ký tự,

Dãy hướng '=' làm việc giống như EQU ngoại trừ các tên định nghĩa với '=' có thể định nghĩa lại sau đó trong chương trình.

**Ví dụ:**

```
CTR = 1
 MOV AX,CTR ;được dịch thành MOV AX,1
Ctr = CTR + 5
 MOV BX,CTR ;được dịch thành MOV BX,6
```

Dẫn hướng '=' thường được dùng trong các macro. Xem chương 13.

## Các dẫn hướng .ERR

Đây là các dẫn hướng lỗi có điều kiện. Để phục vụ cho việc gỡ rối, chúng ta có thể sử dụng chúng để buộc chương trình biên dịch hiển thị một thông báo lỗi khi biên dịch. Chương trình biên dịch hiển thị thông báo "Forced error" cùng với một số xác định. Bạn hãy xem lại chương 13.

| DẪN HƯỚNG              | SỐ | BÁO LỖI NẾU                |
|------------------------|----|----------------------------|
| .ERR1                  | 87 | gặp khi biên dịch lần 1    |
| .ERR2                  | 88 | gặp khi biên dịch lần 2    |
| .ERR                   | 89 | gặp lỗi                    |
| .ERR expression        | 90 | biểu thức sai (0)          |
| .ERRNZ expression      | 91 | biểu thức đúng (khác 0)    |
| .ERRNDEF name          | 92 | name không được định nghĩa |
| .ERRDEF name           | 93 | name đã được định nghĩa    |
| .ERRB <argument>       | 94 | tham số bỏ trống           |
| .ERRNB <argument>      | 95 | tham số không bị bỏ trống  |
| .ERRIDN <arg1>, <arg2> | 96 | arg1 và arg2 bằng nhau     |
| .ERRDIF <arg1>, <arg2> | 97 | arg1 và arg2 khác nhau     |

## EVEN

Tăng bộ đếm định vị đến địa chỉ chẵn tiếp theo.

## EXITM

Sử dụng trong một macro hay khối lặp. Báo cho chương trình biên dịch kết thúc một macro hay khối lặp.

## EXTRN

Cú pháp:

EXTRN name:type[ , name:type]

Báo cho chương trình biên dịch một tên ngoài; có nghĩa là một tên được định nghĩa trong một module khác. Kiểu (type) của tên phải tương ứng với kiểu đã

được khai báo trong module kia. Các kiểu có thể là NEAR, FAR, PROC, BYTE, WORD, DWORD, FWORD, QWORD, TBYTE hay ABS. Xem chương 14.

## .FARADATA và .FARADATA?

**Cú pháp:**

```
.FARADATA { name}
.FARADATA? { name}
```

Chủ yếu dùng với trình biên dịch để định nghĩa các đoạn dữ liệu phụ ngoài.

## GROUP

**Cú pháp:**

```
name GROUP segment [,segment]
```

Một nhóm các đoạn được chọn được liên kết với cùng một địa chỉ bắt đầu. Các nhãn và biến trong các đoạn của nhóm được gán cho các địa chỉ tương đối so với điểm bắt đầu của nhóm thay vì so với điểm bắt đầu của đoạn mà chúng được định nghĩa. Điều này tạo ra khả năng chỉ với việc khởi tạo một thanh ghi đoạn ta có thể tham chiếu tất cả các dữ liệu của nhóm.

**Chú ý:** Ta có thể nhận được kết quả tương tự bằng cách đưa cùng một tên và thuộc tính PUBLIC cho tất cả các đoạn.

## Các dẫn hướng IF

Các dẫn hướng này được sử dụng để cung cấp các cho phép biên dịch nhằm biên dịch các dòng lệnh theo sau các dẫn hướng tuỳ thuộc vào các điều kiện. Danh sách các dẫn hướng này đã được chỉ ra trong chương 13.

## INCLUDE

**Cú pháp:**

```
INCLUDE filespec
```

trong đó filespec xác định một file chứa các dòng lệnh viết bằng Hợp ngữ. Ngoài một tên file, filespec có thể chứa tên ổ đĩa và đường dẫn.

Dẫn hướng báo cho chương trình biên dịch chèn nội dung của file tại vị trí INCLUDE vào file nguồn và bắt đầu xử lý các dòng lệnh của file mới chèn này.

### **Ví dụ:**

```
INCLUDE MACLIB
INCLUDE C:\BIN\PROG1.ASM
```

## **LABEL**

### **Cú pháp:**

```
name LABEL type
```

Ở đây type có thể là BYTE, WORD, DWORD, QWORD, TBYTE hay một tên có cấu trúc đã định nghĩa trước.

Dẫn hướng này cung cấp một phương pháp định nghĩa hay định nghĩa lại kiểu đã gán cho một tên.

### **Ví dụ:**

```
WORD_ARR LABEL WORD
BYTE_ARR DB 100 DUP (0)
```

Ở đây WORD\_ARR định nghĩa là một mảng 50 word, và BYTE\_ARR định nghĩa là một mảng 100 byte. Cả hai biến được gán cho cùng một địa chỉ.

## **.LALL**

Báo cho chương trình biên dịch liệt kê tất cả các dòng lệnh trong mở rộng macro trừ khi chúng được viết trước bằng hai dấu chấm phẩy liên tiếp.

## **.LIST và .XLIST**

.LIST báo cho chương trình biên dịch ghép các dòng lệnh theo sau nó vào trong chương trình nguồn. .XLIST khử bỏ các dòng lệnh theo sau nó.

## **LOCAL**

### **Cú pháp:**

```
LOCAL name [,name]
```

Dùng bên trong một macro. Mỗi lần chương trình biên dịch gặp một tên (name) khi mở rộng macro, nó thay thế tên đó bởi một tên duy nhất có dạng số ???. Bằng cách này ta tránh được các tên lặp lại khi gọi macro vài lần trong một chương trình. Xem chương 13.

## MACRO và ENDM

### Cú pháp:

```
name MACRO { parameter [,parameter] }
```

Dùng để định nghĩa một macro.

### Ví dụ:

```
EXCHANGE MACRO WORD1,WORD2
 PUSH WORD1
 PUSH WORD2
 POP WORD1
 POP WORD2
ENDM
```

Xem chương 13.

## .MODEL

### Cú pháp:

```
.MODEL memory_model
```

Một dẫn hướng đoạn ngắn gọn để định nghĩa kiểu bộ nhớ. Các kiểu bộ nhớ có thể được liệt kê sau đây:

|         |                                                                                                     |
|---------|-----------------------------------------------------------------------------------------------------|
| TINY    | mã lệnh và dữ liệu trong cùng một đoạn                                                              |
| SMALL   | mã lệnh trong một đoạn, dữ liệu trong một đoạn                                                      |
| MEDIUM  | mã lệnh trong nhiều hơn một đoạn, dữ liệu trong một đoạn                                            |
| COMPACT | mã lệnh trong một đoạn, dữ liệu trong nhiều hơn một đoạn                                            |
| LARGE   | mã lệnh trong nhiều hơn một đoạn, dữ liệu trong nhiều hơn một đoạn, không có mảng nào lớn hơn 64 KB |
| HUGE    | mã lệnh trong nhiều hơn một đoạn, dữ liệu trong nhiều hơn một đoạn, các mảng có thể lớn hơn 64 KB   |

## ORG

### Cú pháp:

```
ORG expression
```

trong đó biểu thức (expression) phải tương đương với một số hai byte.

Thiết lập bộ đếm định vị với giá trị của biểu thức. Ví dụ, trong một chương trình .COM, dẫn hướng ORG 100h thiết lập bộ đếm định vị bằng 100h, như vậy các

bien sẽ được gán địa chỉ tính tương đối với điểm bắt đầu của chương trình chứ không phải điểm đầu của đoạn nháp 100 byte đứng trước chương trình trong bộ nhớ. Một cách khác sử dụng ORG là để định nghĩa vùng dữ liệu có thể gán cho nhiều biến. Ví dụ:

```
.DATA
 WORD1_ARR DW 100 DUP (?)
 ORG 0
 WORD2_ARR DW 50 DUP (?)
 WORD3_ARR DW 50 DUP (?)
```

Định nghĩa này tạo ra WORD2\_ARR và 50 từ đầu tiên của WORD1\_ARR chiếm cùng một chỗ trong bộ nhớ. Tương tự WORD3\_ARR và 50 từ sau cùng của WORD1\_ARR cũng chiếm cùng chỗ bộ nhớ.

## %OUT

**Cú pháp:**

```
%OUT text
```

trong đó text là một dòng các ký tự ASCII.

Dùng để hiển thị một thông báo tại một vị trí xác định trong một assembly listing. Dẫn hướng này thường được dùng khi biên dịch có điều kiện.

**Ví dụ:**

```
IFNDEF SUM
 SUM DW ?
%OUT SUM is define here
ENDIF
```

Nếu SUM chưa được định nghĩa, nó sẽ được định nghĩa ở đây và thông báo được hiển thị.

## PAGE

**Cú pháp:**

```
PAGE {length},width
```

trong đó length có giá trị 10-255; width có giá trị 60-132. Các giá trị ngầm định là length = 50 và width = 80.

Dùng để nhảy sang trang mới hay xác định số lớn nhất các dòng trong một trang và số lớn nhất các ký tự trong một dòng trong một chương trình nguồn.

### Ví dụ:

|            |                                                                            |
|------------|----------------------------------------------------------------------------|
| PAGE       | ; sang trang mới                                                           |
| PAGE 50,70 | ; thiết lập chiều dài tối đa của trang<br>; là 50; chiều rộng tối đa là 70 |
| PAGE ,132  | ; thiết lập chiều rộng tối đa là 132                                       |

### PROC và ENDP

#### Cú pháp:

```
name PROC distance
 name ENDP
```

trong đó distance là REAR hoặc FAR. Ngầm định là NEAR.

Dùng để bắt đầu và kết thúc một định nghĩa thủ tục. Xem chương 8.

### Các dẫn hướng cho bộ xử lý và bộ đồng xử lý.

Các dẫn hướng sau đây định nghĩa hệ lệnh được tổ chức bởi MASM. Các dẫn hướng này phải nằm ngoài các khai báo đoạn. Trong bảng sau đây, 8086 được hiểu là gồm cả 8088, 8087, 80287 và 80387 là các bộ đồng xử lý.

| DẪN HƯỚNG | CHO PHÉP CÁC LỆNH CỦA CÁC BỘ XỬ LÝ VÀ ĐỒNG XỬ LÝ       |
|-----------|--------------------------------------------------------|
| .8086     | 8086, 8087                                             |
| .186      | 8086, 8087 và các lệnh thêm của 80186                  |
| .286      | 8086, 80287 và các lệnh không ưu tiên của 80286        |
| .286P     | giống như .286 cộng với các lệnh ưu tiên của 80286     |
| .386      | 8086, 80387, 80286 và các lệnh không ưu tiên của 80386 |
| .386P     | giống như .386 cộng với các lệnh ưu tiên của 80386     |
| .287      | 8087 và các lệnh thêm của 80287                        |
| .387      | 8087, 80287 và các lệnh thêm của 80387                 |

## PUBLIC

### Cú pháp:

```
PUBLIC name [,name]
```

Ở đây name là một biến, nhãn hay một đăng thức số định nghĩa trong module chứa dẫn hướng PUBLIC.

Dùng để cho phép các tên trong module này có thể sử dụng được trong các module khác. Chú ý không nhầm lẫn với kiểu combine PUBLIC dùng để kết hợp các đoạn. Xem chương 14.

## PURGE

### Cú pháp:

```
PURGE macroname [,macroname]
```

trong đó macroname là tên của một macro.

Dùng để xoá các macro trong bộ nhớ khi biên dịch. Điều này có thể cần thiết khi hệ thống không đủ bộ nhớ để lưu giữ mọi macro mà chương trình cần tại cùng một thời điểm.

### Ví dụ:

```
MAC1 ; mở rộng macro MAC1
PURGE MAC1 ; không cần nó trong bộ nhớ nữa.
```

## .RADIX

### Cú pháp:

```
.RADIX base
```

trong đó base là một số thập phân trong khoảng 2-16.

Xác định cơ số mặc định cho các hàng số nguyên. Điều này có nghĩa là khi không có một ký tự "b", "d" hay "h" theo sau một số nguyên, chương trình biên dịch sẽ coi cơ số của nó được xác định bởi dẫn hướng. Cơ số ngầm định là 10 (số thập phân).

### Ví dụ:

```
.DATA
.RADIX 16 ; số hex
 A DW 1101 ; được dịch bằng 1101h
.RADIX 2
 B DW 1101 ; được định bằng 1101b
```

## **RECORD**

Dùng để định nghĩa một biến kiểu record. Đây là một byte hay word trong đó một cách hình thức ta có thể truy nhập các trường bit xác định. Bạn có thể xem thêm thêm cuốn Microsoft Macro Assembler Programer's Guide.

## **REPT và ENDM**

**Cú pháp:**

```
REPT expression
statements
ENDM
```

(trong đó expression phải có giá trị là một số không dấu 16 bit).

Định nghĩa một khôi lặp. Các dòng lệnh trong khôi sẽ được biên dịch một số lần bằng với giá trị của expression. Chúng ta có thể sử dụng khôi lặp để lặp lại các dòng lệnh hay dùng nó trong một macro. Xem chương 13.

## **.SALL**

Báo cho chương trình biên dịch loại bỏ danh sách các mở rộng macro.

## **SEGMENT và ENDS**

**Cú pháp:**

```
name SEGMENT { align}{ combine}{ 'class'}
```

trong đó align có thể là PARA, BYTE, WORD hay PAGE,

combine có thể là PUBLIC, COMMON, STACK hay AT,

class là một tên bao trong dấu nháy đơn.

Dẫn hướng này định nghĩa một đoạn chương trình. Align, combine và class xác định một đoạn được xếp trong bộ nhớ và kết hợp với các đoạn khác như thế nào, thứ tự của nó với các đoạn còn lại ra sao. Xem chương 14.

## **.SEQ**

Hướng dẫn chương trình biên dịch xắp xếp các đoạn theo thứ tự nguyên thuỷ của nó. Cho cùng kết quả như .ALPHA.

## **.STACK**

**Cú pháp:**

STACK { size}

Ở đây size là một số nguyên dương.

Một dẫn hướng biên dịch sử dụng trong kiểu định nghĩa đoạn đơn giản hoá để định nghĩa một đoạn ngắn xếp với kích thước tính bằng byte. Kích thước ngầm định là 1 KB.

## **STRUC và ENDS**

Dùng để khai báo một cấu trúc. Đây là một tập hợp các đối tượng dữ liệu mà một cách hình thức có thể truy nhập chúng như một đối tượng đơn lẻ. Bạn có thể xem thêm cuốn Microsoft Macro Assembler Programer's Guide.

## **SUBTTL**

**Cú pháp:**

SUBTTL { text}

Đưa một tiêu đề phụ (1-60 ký tự) vào dòng thứ ba của mỗi trang trong một danh sách biên dịch. Dẫn hướng này có thể dùng một hay nhiều lần.

## **TITLE**

**Cú pháp:**

TITLE { text}

Đưa một tiêu đề vào mỗi trang trong một danh sách biên dịch. Dẫn hướng này chỉ được dùng một lần.

## **.XALL**

Báo cho chương trình biên dịch liệt kê tất cả cá dòng lệnh mà tạo ra các mã lệnh trong một mảng rộng macro. Các lời bình bị bỏ qua.

## **.XREF**

Xem .CREF

## **.XLIST**

Xem .LIST

## **Phụ lục H**

# **CÁC MÃ QUÉT BÀN PHÍM.**

## **(KEYBOARD SCAN CODES)**

Bàn phím liên lạc với bộ vi xử lý thông qua các mã quét được gán cho các phím. Khi một phím được nhấn, bàn phím gửi một mã nhấn đến máy tính và khi một phím được nhả ra, một mã nhả được gửi đi. Bảng H.1 chỉ ra các mã nhấn của bàn phím IBM nguyên thuỷ 83 phím. Từ mã nhấn ta có thể nhận được mã nhả qua phép OR nó với 80h.

Phục vụ ngắt 9 có trách nhiệm nhận mã quét từ cổng vào/ra, sau đó đưa một mã ASCII và mã quét vào bộ đệm bàn phím. Chúng ta có thể phân loại các phím thành các phím ASCII, các phím chức năng và các phím dịch. Các phím ASCII bao gồm các phím ký tự, phím trống và các phím điều khiển: Esc, Enter, Backspace, Tab. Các phím này có các mã ASCII tương ứng. Các phím chức năng bao gồm các phím F1-F10, các phím mũi tên, PgUp, PgDn, Home, End, Ins và Del. Các phím này không có mã ASCII và trong bộ đệm bàn phím, một số 0 được dùng để xác định một phím chức năng. Các phím dịch gồm có các phím Shift trái, Shift phải, Caps Lock, Ctrl, Alt, Num Lock và Scroll Lock. Các mã quét của các phím dịch không được đưa vào bộ đệm bàn phím. Thay vào đó, phục vụ ngắt 9 dùng byte các cờ bàn phím (địa chỉ 0000:0417) để theo dõi các phím này. Ta có thể nhận được cờ bàn phím thông qua hàm 2 của ngắt 16h.

Khi một phím dịch nào đó được nhấn cùng với các phím khác, phục vụ ngắt 9 đưa một mã quét khác vào bộ đệm bàn phím để xác định các tổ hợp phím. Bảng H.2 chỉ ra mã quét của các tổ hợp phím.

Bàn phím nâng cao 101 phím sử dụng một hệ thống mã nhả và mã nhấn khác. Tuy vậy, để giữ tính tương thích, ngoại trừ các phím mới F11 và F12, phục vụ ngắt 9 vẫn phát sinh 83 mã quét đến bộ đệm bàn phím. Bảng H.3 chỉ ra các mã quét mới phát sinh bởi phục vụ ngắt 9 cho bàn phím nâng cao. Ta cũng thấy có một số mã quét tổ hợp mới. Các mã quét mới này chỉ có thể nhận được thông qua ngắt 16h, hàm 10h và 11h.

Khi phím Ctrl được nhấn, ngắt 9 sẽ phát sinh mã ASCII khác cho các phím chữ cái. Bảng H.4 chỉ ra mã quét và mã ASCII của các tổ hợp phím.

**Bảng H.1. Các mã quét bàn phím 83 phím.**

| MÃ QUÉT<br>DANG HEX | PHÍM       | MÃ QUÉT<br>DANG HEX | PHÍM       | MÃ QUÉT<br>DANG HEX | PHÍM            |
|---------------------|------------|---------------------|------------|---------------------|-----------------|
| 01                  | Esc        | 1D                  | Ctrl       | 39                  | Space bar       |
| 02                  | 1          | 1E                  | A          | 3A                  | Caps Lock       |
| 03                  | 2          | 1F                  | S          | 3B                  | F1              |
| 04                  | 3          | 20                  | D          | 3C                  | F2              |
| 05                  | 4          | 21                  | F          | 3D                  | F3              |
| 06                  | 5          | 22                  | G          | 3E                  | F4              |
| 07                  | 6          | 23                  | H          | 3F                  | F5              |
| 08                  | 7          | 24                  | J          | 40                  | F6              |
| 09                  | 8          | 25                  | K          | 41                  | F7              |
| 0A                  | 9          | 26                  | L          | 42                  | F8              |
| 0B                  | 0          | 27                  | ;          | 43                  | F9              |
| 0C                  | -          | 28                  | ' "        | 44                  | F10             |
| 0D                  | =+         | 29                  | ' ~        | 45                  | Num Lock        |
| 0E                  | Back Space | 2A                  | Shift trái | 46                  | Scroll Lock     |
| 0F                  | Tab        | 2B                  | \          | 47                  | 7 Home          |
| 10                  | Q          | 2C                  | Z          | 48                  | 8 Mũi tên lên   |
| 11                  | W          | 2D                  | X          | 49                  | 9 PgDn          |
| 12                  | E          | 2E                  | C          | 4A                  | - (num)         |
| 13                  | R          | 2F                  | V          | 4B                  | 4 Mũi tên trái  |
| 14                  | T          | 30                  | B          | 4C                  | 5 (num)         |
| 15                  | Y          | 31                  | N          | 4D                  | 6 Mũi tên phải  |
| 16                  | U          | 32                  | M          | 4E                  | + (num)         |
| 17                  | I          | 33                  | , <        | 4F                  | 1 End           |
| 18                  | O          | 34                  | , >        | 50                  | 2 Mũi tên xuống |
| 19                  | P          | 35                  | / ?        | 51                  | 3 PgDn          |
| 1A                  | [ {        | 36                  | Shift phải | 52                  | 0 Ins           |
| 1B                  | } ]        | 37                  | Prt Sc     | 53                  | Del             |
| 1C                  | Enter      | 38                  | Alt        |                     |                 |

**Bảng H.2. Mã quét các phím tổ hợp**

| Mã quét<br>dạng hex | Phím      | Mã quét<br>dạng hex | Phím       | Mã quét<br>dạng hex | Phím      |
|---------------------|-----------|---------------------|------------|---------------------|-----------|
| 54                  | Shift F1  | 65                  | Ctrl F8    | 75                  | Ctrl End  |
| 55                  | Shift F2  | 66                  | Ctrl F9    | 76                  | Ctrl PgDn |
| 56                  | Shift F3  | 67                  | Ctrl F10   | 77                  | Ctrl Home |
| 57                  | Shift F4  | 68                  | Alt F1     | 78                  | Alt 1     |
| 58                  | Shift F5  | 69                  | Alt F2     | 79                  | Alt 2     |
| 59                  | Shift F6  | 6A                  | Alt F3     | 7A                  | Alt 3     |
| 5A                  | Shift F7  | 6B                  | Alt F4     | 7B                  | Alt 4     |
| 5B                  | Shift F8  | 6C                  | Alt F5     | 7C                  | Alt 5     |
| 5C                  | Shift F9  | 6D                  | Alt F6     | 7D                  | Alt 6     |
| 5D                  | Shift F10 | 6E                  | Alt F7     | 7E                  | Alt 7     |
| 5E                  | Ctrl F1   | 6F                  | Alt F8     | 7F                  | Alt 8     |
| 5F                  | Ctrl F2   | 70                  | Alt F9     | 80                  | Alt 9     |
| 60                  | Ctrl F3   | 71                  | Alt F10    | 81                  | Alt 0     |
| 61                  | Ctrl F4   | 72                  | Ctrl PrtSc | 82                  | Alt -     |
| 62                  | Ctrl F5   | 73                  | Ctrl ←     | 83                  | Alt =     |
| 63                  | Ctrl F6   | 74                  | Ctrl →     | 84                  | Ctrl PgUp |
| 64                  | Ctrl F7   |                     |            |                     |           |

**Bảng H.3. Mã quét cho bàn phím nâng cao.**

| Mã quét<br>dạng<br>hex | Phím         | Mã quét<br>dạng hex | Phím         | Mã quét<br>dạng hex | Phím            |
|------------------------|--------------|---------------------|--------------|---------------------|-----------------|
| 85                     | F11          | 90                  | Ctrl + (num) | 9B                  | Alt ←           |
| 86                     | F12          | 91                  | Ctrl ↓       | 9D                  | Alt →           |
| 87                     | Shift F11    | 92                  | Ctrl Ins     | 9F                  | Alt End         |
| 88                     | Shift F12    | 93                  | Ctrl Del     | A0                  | Alt             |
| 89                     | Ctrl F11     | 94                  | Ctrl Tab     | A1                  | Alt PgDn        |
| 8A                     | Ctrl F12     | 95                  | Ctrl / (num) | A2                  | Alt Ins         |
| 8B                     | Alt F11      | 96                  | Ctrl * (num) | A3                  | Alt Del         |
| 8C                     | Alt F12      | 97                  | Alt Home     | A4                  | Alt / (num)     |
| 8D                     | Ctrl ↑       | 98                  | Alt ↑        | A5                  | Alt Tab         |
| 8E                     | Ctrl - (num) | 99                  | Alt PgUp     | A6                  | Alt Enter (num) |
| 8F                     | Ctrl 5 (num) |                     |              |                     |                 |

(num) có nghĩa là các phím của bàn phím số ( bên phải bàn phím )

**Bảng H.4. Các mã ASCII của các phím tổ hợp.**

| Phím   | Mã ASCII<br>dạng hex | Mã quét<br>dạng hex | Phím   | Mã ASCII<br>dạng hex | Mã quét<br>dạng hex |
|--------|----------------------|---------------------|--------|----------------------|---------------------|
| Ctrl A | 1                    | 1E                  | Ctrl N | E                    | 31                  |
| Ctrl B | 2                    | 30                  | Ctrl O | F                    | 18                  |
| Ctrl C | 3                    | 2E                  | Ctrl P | 10                   | 19                  |
| Ctrl D | 4                    | 20                  | Ctrl Q | 11                   | 10                  |
| Ctrl E | 5                    | 12                  | Ctrl R | 12                   | 13                  |
| Ctrl F | 6                    | 21                  | Ctrl S | 13                   | 1F                  |
| Ctrl G | 7                    | 22                  | Ctrl T | 14                   | 14                  |
| Ctrl H | 8                    | 23                  | Ctrl U | 15                   | 16                  |
| Ctrl I | 9                    | 17                  | Ctrl V | 16                   | 2F                  |
| Ctrl J | A                    | 24                  | Ctrl W | 17                   | 11                  |
| Ctrl K | B                    | 25                  | Ctrl X | 18                   | 2D                  |
| Ctrl L | C                    | 26                  | Ctrl Y | 19                   | 15                  |
| Ctrl M | D                    | 32                  | Ctrl Z | 1A                   | 2C                  |

# MỤC LỤC

|                                                   |    |
|---------------------------------------------------|----|
| <b>Mở đầu</b>                                     | 3  |
| <b>Phần I: Những cơ sở lập trình bằng Hợp ngữ</b> | 7  |
| Chương 1: CÁC HỆ THỐNG MÁY VI TÍNH                | 8  |
| Tổng quan                                         | 8  |
| 1.1 Các thành phần của một hệ thống vi tính       | 8  |
| 1.2 Việc thực hiện các lệnh                       | 15 |
| 1.3 Các thiết bị ngoại vi                         | 17 |
| 1.4 Các ngôn ngữ lập trình                        | 19 |
| 1.5 Một chương trình bằng Hợp ngữ                 | 22 |
| Các thuật ngữ tiếng Anh                           | 24 |
| Bài tập                                           | 27 |
| Chương 2: PHƯƠNG PHÁP BIỂU DIỄN SỐ VÀ KÍ TỰ       | 29 |
| Tổng quan                                         | 29 |
| 2.1 Các hệ đếm                                    | 29 |
| 2.2 Việc chuyển đổi giữa các hệ đếm               | 33 |
| 2.3 Phép cộng và trừ trong các hệ đếm             | 35 |
| 2.4 Cách biểu diễn các số nguyên trong máy tính   | 38 |
| 2.5 Biểu diễn các ký tự                           | 45 |
| Tổng kết                                          | 48 |
| Các thuật ngữ tiếng Anh                           | 49 |
| Bài tập                                           | 50 |
| Chương 3: TỔ CHỨC CỦA MÁY TÍNH CÁ NHÂN IBM-PC     | 53 |
| Tổng quan                                         | 53 |
| 3.1 Bộ vi xử lý INTEL 8086                        | 53 |
| 3.2 Tổ chức của các bộ vi xử lý 8086/8088         | 55 |
| 3.3 Tổ chức của máy PC                            | 64 |
| Tổng kết                                          | 69 |
| Các thuật ngữ tiếng Anh                           | 70 |
| Bài tập                                           | 73 |
| Chương 4: GIỚI THIỆU HỢP NGỮ CHO IBM-PC           | 74 |
| Tổng quan                                         | 74 |
| 4.1 Cú pháp của Hợp ngữ                           | 74 |
| 4.2 Dữ liệu chương trình                          | 77 |

|                                                              |     |
|--------------------------------------------------------------|-----|
| 4.3 Các biến                                                 | 79  |
| 4.4 Các hằng có tên                                          | 81  |
| 4.5 Vài lệnh cơ bản                                          | 82  |
| 4.6 Dịch từ ngôn ngữ bậc cao sang Hợp ngữ                    | 88  |
| 4.7 Cấu trúc chương trình                                    | 89  |
| 4.8 Các lệnh vào/ra                                          | 92  |
| 4.9 Chương trình đầu tiên                                    | 94  |
| 4.10 Tạo lập và chạy một chương trình                        | 96  |
| 4.11 Hiển thị một chuỗi                                      | 100 |
| 4.12 Một chương trình đổi chữ thường thành chữ hoa           | 112 |
| <br>Tổng kết                                                 | 105 |
| Các thuật ngữ tiếng Anh                                      | 106 |
| Bài tập                                                      | 108 |
| <br><b>Chương 5: TRẠNG THÁI CỦA BỘ XỬ LÍ VÀ THANH GHI CỜ</b> | 111 |
| Tổng quan                                                    | 111 |
| 5.1 Thanh ghi cờ                                             | 111 |
| 5.2 Hiện tượng tràn                                          | 113 |
| 5.3 Sự ảnh hưởng của các lệnh các cờ                         | 118 |
| 5.4 Chương trình DEBUG                                       | 119 |
| <br>Tổng kết                                                 | 124 |
| Các thuật ngữ tiếng Anh                                      | 125 |
| Bài tập                                                      | 126 |
| <br><b>Chương 6: CÁC LỆNH ĐIỀU KHIỂN RẼ NHÁNH</b>            | 128 |
| Tổng quan                                                    | 128 |
| 6.1 Một ví dụ về lệnh nhảy                                   | 128 |
| 6.2 Các lệnh nhảy có điều kiện                               | 130 |
| 6.3 Lệnh JMP                                                 | 134 |
| 6.4 Các cấu trúc ngôn ngữ bậc cao                            | 135 |
| 6.5 Lập trình với các cấu trúc bậc cao                       | 147 |
| <br>Tổng kết                                                 | 153 |
| Các thuật ngữ tiếng Anh                                      | 154 |
| Bài tập                                                      | 155 |
| <br><b>Chương 7: CÁC LỆNH LOGIC, DỊCH VÀ QUAY</b>            | 158 |
| Tổng quan                                                    | 158 |
| 7.1 Các lệnh logic                                           | 159 |
| 7.2 Các lệnh dịch                                            | 165 |
| <br>Tổng kết                                                 | 180 |

|                                              |            |
|----------------------------------------------|------------|
| Các thuật ngữ tiếng Anh                      | 181        |
| Bài tập                                      | 182        |
| <br>                                         |            |
| <b>Chương 8: NGĂN XẾP VÀ CÁC THỦ TỤC</b>     | <b>185</b> |
| Tổng quan                                    | 185        |
| 8.1 Ngăn xếp                                 | 185        |
| 8.2 Một ứng dụng của ngăn xếp                | 190        |
| 8.3 Các thuật ngữ của thủ tục                | 193        |
| 8.4 CALL và RET                              | 294        |
| 8.5 Ví dụ các thủ tục                        | 298        |
| <br>                                         |            |
| Tổng kết                                     | 207        |
| Các thuật ngữ tiếng Anh                      | 208        |
| Bài tập                                      | 209        |
| <br>                                         |            |
| <b>Chương 9: CÁC LỆNH NHÂN VÀ CHIA</b>       | <b>213</b> |
| Tổng quan                                    | 213        |
| 9.1 Các lệnh MUL và IMUL                     | 213        |
| 9.2 Các ứng dụng đơn giản của MUL và IMUL    | 216        |
| 9.3 Các lệnh DIV và IDIV                     | 218        |
| 9.4 Sự mở rộng dấu của số bị chia            | 220        |
| 9.5 Các thủ tục vào ra với số thập phân      | 221        |
| <br>                                         |            |
| Tổng kết                                     | 233        |
| Các thuật ngữ tiếng Anh                      | 234        |
| Bài tập                                      | 234        |
| <br>                                         |            |
| <b>Chương 10: MẢNG VÀ CÁC CHẾ ĐỘ ĐỊA CHỈ</b> | <b>236</b> |
| Tổng quan                                    | 236        |
| 10.1 Mảng một chiều                          | 236        |
| 10.2 Các chế độ địa chỉ                      | 239        |
| 10.3 Sắp xếp một mảng                        | 249        |
| 10.4 Mảng hai chiều                          | 253        |
| 10.5 Chế độ địa chỉ chỉ số cơ sở             | 256        |
| 10.6 Một ứng dụng: tính điểm trung bình      | 257        |
| 10.7 Lệnh XLAT                               | 260        |
| <br>                                         |            |
| Tổng kết                                     | 264        |
| Các thuật ngữ tiếng Anh                      | 265        |
| Bài tập                                      | 266        |
| <br>                                         |            |
| <b>Chương 11: CÁC LỆNH THAO TÁC CHUỖI</b>    | <b>270</b> |
| Tổng quan                                    | 270        |
| 11.1 Cờ định hướng                           | 270        |

|                                                 |     |
|-------------------------------------------------|-----|
| 11.2 Lệnh chuyển một chuỗi                      | 271 |
| 11.3 Lệnh lưu chuỗi                             | 275 |
| 11.4 Nạp một chuỗi                              | 278 |
| 11.5 Lệnh duyệt chuỗi                           | 282 |
| 11.6 Lệnh so sánh chuỗi                         | 286 |
| 11.7 Dạng tổng quát của các lệnh thao tác chuỗi | 294 |
| Tổng kết                                        | 296 |
| Các thuật ngữ tiếng Anh                         | 297 |
| Bài tập                                         | 298 |

## **Phần II: Các chủ đề nâng cao**

|                                                         |     |
|---------------------------------------------------------|-----|
| <b>Chương 12 HIỂN THỊ VĂN BẢN VÀ LẬP TRÌNH BÀN PHÍM</b> |     |
| Tổng quan                                               | 302 |
| 12.1 Màn hình                                           | 302 |
| 12.2 Bộ phổi ghép màn hình và các chế độ hiển thị       | 303 |
| 12.3 Lập trình ở chế độ văn bản                         | 305 |
| 12.4 Bàn phím                                           | 319 |
| 12.5 Chương trình soạn thảo màn hình                    | 323 |
| <br>Tổng kết                                            | 330 |
| Các thuật ngữ tiếng Anh                                 | 331 |
| Bài tập                                                 | 332 |
| <br><b>Chương 13 MACRO (Lệnh gộp)</b>                   | 334 |
| Tổng quan                                               | 334 |
| 13.1 Định nghĩa và tham chiếu macro                     | 334 |
| 13.2 Các nhãn cục bộ                                    | 341 |
| 13.3 Macro tham chiếu đến Macro khác                    | 342 |
| 13.4 Thư viện Macro                                     | 343 |
| 13.5 Các macro lặp                                      | 347 |
| 13.6 Một macro xuất                                     | 351 |
| 13.7 Các toán tử điều kiện                              | 354 |
| 13.8 Macro và thủ tục                                   | 358 |
| <br>Tổng kết                                            | 359 |
| Các thuật ngữ tiếng Anh                                 | 360 |
| Bài tập                                                 | 360 |
| <br><b>Chương 14 : QUẢN LÍ BỘ NHỚ</b>                   | 364 |
| Tổng quan                                               | 364 |
| 14.1 Chương trình *.COM                                 | 365 |
| 14.2 Modun chương trình                                 | 369 |
| 14.3 Các định nghĩa đoạn toàn phần                      | 376 |
| 14.4 Thêm về các định nghĩa đoạn đơn giản hóa           | 386 |
| 14.5 Chuyển dữ liệu giữa các thủ tục                    | 388 |
| Tổng kết                                                | 395 |
| Các thuật ngữ tiếng Anh                                 | 396 |
| Bài tập                                                 | 396 |
| <br><b>Chương 15 NGẮT DOS VÀ BIOS</b>                   | 398 |
| Tổng quan                                               | 398 |
| 15.1 Các dịch vụ ngắt                                   | 398 |
| 15.2 Ngắt của BIOS                                      | 414 |
| 15.3 Ngắt của DOS                                       | 407 |

|                                             |     |
|---------------------------------------------|-----|
| 15.4 Chương trình hiện thời gian            | 407 |
| 15.5 Các thủ tục ngắn của người sử dụng     | 410 |
| 15.6 Chương trình thường trú                | 415 |
| Tổng kết                                    | 423 |
| Các thuật ngữ tiếng Anh                     | 423 |
| Bài tập                                     | 424 |
| <br>Chương 16 ĐỒ HOẠ MÀU                    | 425 |
| Tổng quan                                   | 425 |
| 16.1 Các chế độ đồ họa                      | 425 |
| 16.2 Đồ họa CGA                             | 427 |
| 16.3 Đồ họa EGA                             | 431 |
| 16.4 Đồ họa VGA                             | 436 |
| 16.5 Làm hình chuyển động                   | 438 |
| 16.6 Làm trò chơi hoạt hình tương tác       | 445 |
| <br>Tổng kết                                | 455 |
| Các thuật ngữ tiếng Anh                     | 455 |
| Bài tập                                     | 456 |
| <br>Chương 17 ĐỆ QUI                        | 457 |
| Tổng quan                                   | 457 |
| 17.1 Ý tưởng về đệ qui.                     | 457 |
| 17.2 Các thủ tục đệ qui.                    | 458 |
| 17.3 Truyền các tham số bằng ngăn xếp.      | 461 |
| 17.4 Bản ghi.                               | 463 |
| 17.5 Thực hiện các thủ tục đệ qui.          | 465 |
| 17.6 Thêm nữa về đệ qui.                    | 470 |
| Tổng kết                                    | 474 |
| Các thuật ngữ tiếng Anh                     | 474 |
| Bài tập                                     | 474 |
| <br>Chương 18 CÁC PHÉP TÍNH SỐ HỌC NÂNG CAO | 476 |
| Tổng quan                                   | 476 |
| 18.1 Các số có độ chính xác kép             | 476 |
| 18.2 Các số thập phân mã hoá nhị phân       | 480 |
| 18.3 Các số dấu phẩy động                   | 486 |
| 18.4 Bộ đồng xử lí 8087                     | 489 |
| Tổng kết                                    | 503 |
| Các thuật ngữ tiếng Anh                     | 504 |
| Bài tập                                     | 505 |

|                                                        |            |
|--------------------------------------------------------|------------|
| <b>Chương 19 CÁC THAO TÁC ĐĨA VÀ TỆP TIN</b>           | <b>507</b> |
| Tổng quan                                              | 507        |
| 19.1 Các loại đĩa                                      | 507        |
| 19.2 Cấu trúc đĩa                                      | 509        |
| 19.3 Xử lí file                                        | 515        |
| 19.4 Các thao tác trực tiếp với đĩa.                   | 531        |
| <br>Tổng kết                                           | 536        |
| Các thuật ngữ tiếng Anh                                | 537        |
| Bài tập                                                | 538        |
| <br><b>Chương 20 CÁC BỘ VI XỬ LÍ CAO CẤP CỦA INTEL</b> | <b>540</b> |
| Tổng quan                                              | 540        |
| 20.1 Bộ vi xử lý 80286                                 | 540        |
| 20.2 Các hệ thống ở chế độ bảo vệ.                     | 551        |
| 20.3 Các bộ vi xử lý 80386 và 80486                    | 556        |
| Tổng kết                                               | 562        |
| Các thuật ngữ tiếng Anh                                | 562        |
| Bài tập                                                | 564        |
| <br><b>PHỤ LỤC</b>                                     |            |
| Phụ lục A: Các mã hiển thị của IBM                     | 565        |
| Phụ lục B: Các lệnh của DOS                            | 568        |
| Phụ lục C: Các ngắt của BIOS và DOS                    | 574        |
| Phụ lục D: MASM và LINK                                | 591        |
| Phụ lục E: DEBUG và COVIEW.                            | 601        |
| Phụ lục F: Bộ lệnh assembly.                           | 621        |
| Phụ lục G: Các hướng dẫn dịch.                         | 665        |
| Phụ lục H: Các mã quét của bàn phím.                   | 681        |

---

*in 1000<sup>c</sup>, khổ 17,5cm x 25,5cm Tại số 2 Phạm Ngũ Lão  
XN in 15. Số XB : 232 / CXB - 204  
in xong và nộp lưu chiểu quý 4 năm 1998*