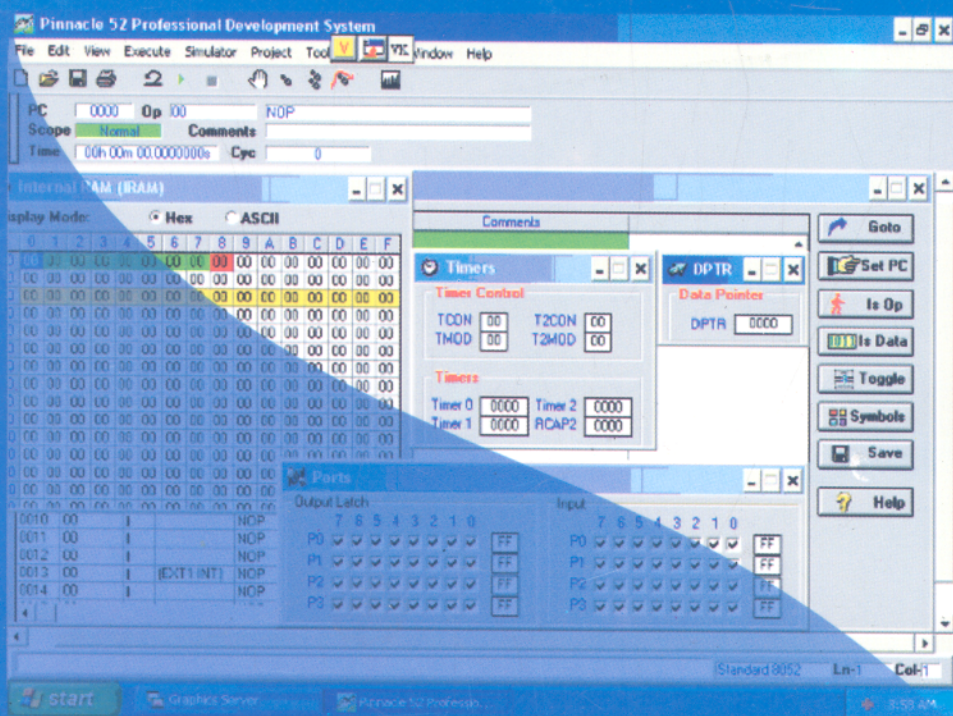


TS. NGUYỄN MẠNH GIANG

CẤU TRÚC-LẬP TRÌNH-GHÉP NỐI VÀ ỨNG DỤNG VI ĐIỀU KHIỂN

TẬP MỘT
CẤU TRÚC VÀ LẬP TRÌNH VĐK 8051/8052



NHÀ XUẤT BẢN GIÁO DỤC

TS. NGUYỄN MẠNH GIANG

CẤU TRÚC-LẬP TRÌNH-GHÉP NỐI VÀ ỨNG DỤNG VI ĐIỀU KHIỂN

TẬP MỘT
CẤU TRÚC VÀ LẬP TRÌNH VDK 8051/8052

NHÀ XUẤT BẢN GIÁO DỤC

Bản quyền thuộc HEVOBCO – Nhà xuất bản Giáo dục

192-2007/CXB/2-411/GD

Mã số: 7B677M7-DAI

LỜI NÓI ĐẦU

Vi xử lý (Microprocessor), bộ phận xử lý trung tâm để xử lý lệnh, đã ra đời lần đầu tiên (1971) với 4 bit dữ liệu, đã lần lượt được cải tiến thành 8 bit (8080, 8085, 8088 năm 1974), 16 bit (8086, 80286 năm 1978), 32 bit (80386, 80486, năm 1983) và 64 bit (80586, năm 1991).

Trên cơ sở các vi xử lý, thêm các khối nhớ chương trình chỉ đọc (PROM), khối nhớ đọc/ghi RAM, các cổng vào ra song song và nối tiếp, người ta đã chế tạo được các máy vi tính cá nhân PC (Personal Computer) như PC/XT (1985 với 8080, 8085), PC/AT (1987 với 80286), PC/AT cải tiến (1989 với 80386, 80486) và Pentium (1991 với 80586). Riêng Pentium, có Pentium I, Pentium II, Pentium III, Pentium MTX đa phương tiện, Pentium Pro hay IV (tốc độ 1,4GHz, 1995) và hiện nay là Pentium V (tốc độ tới trên 3,4 GHz).

Vi điều khiển (Microcontroller) là một vi mạch tích hợp lớn, kích thước nhỏ (rộng cỡ 2 cm, dài từ 3-8 cm, với số chân từ 18, 28, 40, 48, 64), bao gồm các bộ phận của một máy vi tính (bộ xử lý trung tâm CPU, nhớ chương trình PROM, nhớ đọc/ghi RAM, các cổng song song, cổng nối tiếp, bộ định thời/dếm (Timer/Counter) và điều khiển ngắt chương trình).

Vi điều khiển có khả năng như một máy vi tính, nhưng có ưu điểm đặc biệt về kích thước rất nhỏ, nên đã xâm nhập sâu hơn nữa vào các thiết bị đo lường, điều khiển tự động hóa và đặc biệt vào các thiết bị của đời sống hàng ngày như đồng hồ thời gian, đồng hồ báo giờ, đầu máy ghi và phát âm hay phát hình (video, VCD, DVD v.v...). Đỉnh cao của việc dùng VDK là dùng trong người máy (ROBOT) và thiết bị PLC để điều khiển logic chương trình hóa dây chuyền sản xuất tự động.

Vi điều khiển có bốn dòng họ lớn là Vi điều khiển 68HCxx của hãng MOTOROLA, MCS-51 của hãng INTEL, AVR của hãng ATMEL và PIC của hãng MICROCHIP.

Ngoài các bộ phận giống nhau về cấu trúc, các vi điều khiển trên đều có :

- *Các nhóm lệnh*: rẽ nhánh chương trình (điều khiển chương trình thay đổi thứ tự lệnh), dịch chuyển dữ liệu (giữa các thanh ghi trong bộ nhớ RAM), xử lý số học (cộng trừ, nhân, chia), xử lý logic (và, hoặc, loại trừ v.v...) và xử lý theo bit (xác lập, xóa, đảo v.v...).

- *Các hành động cơ bản*: trao đổi tin song song, trao đổi tin nối tiếp, định thời gian, đếm xung ngoài, điều khiển ngắt chương trình (do xung ngoài, do đếm tràn về 0, do trao đổi tin nối tiếp).

Hiện nay, có nhiều hãng điện tử tham gia chế tạo các VDK khác nhau nhưng theo cấu trúc và tính năng tương tự trên. Có các loại sau:

- *VDK dùng trong công nghiệp*: cấu tạo đơn giản, ít cổng vào/ra song song, có 18 chân như 2051 của hãng INTEL, PIC của MICROCHIP.

- *VDK thông dụng*: 8051, 8052 thuộc họ MCS-51 của hãng INTEL, AT89C51, AT89C52, AT90C52, AT89C535 của hãng ATMEL.

- *VDK tiên tiến*: có các tính năng mở rộng như có nhiều mức ngắt, có nhiều bộ định thời và cả Watch dog (chó canh cửa), đếm kiểu ma trận, ghép nối với thiết bị ngoài nối tiếp SPI v.v...

- *VDK có DAC, ADC bên trong vi mạch*: ADU812 (DAC, ADC có 12 bit), ADU 816 (DAC, ADC có 16 bit), ADU824 (DAC, ADC có 24 bit) của hãng Analog hay các hãng khác.

- *VDK dùng trong truyền thông trên mạng*: 87C51GB và xử lý số DSP 51000.

Tài liệu này được biên soạn với các tiêu chí sau:

- *Kết hợp phần cứng và phần mềm:* mô tả về cấu tạo kết hợp hoạt động và lập trình bằng hợp ngữ.

- *Kết hợp lý thuyết với thực hành:* mỗi chương đều có thực hành về ghép nối thiết bị với lập trình trên chương trình mô phỏng trên máy tính PC và cho chạy trên kit (giá) thực nghiệm.

- *Cấu trúc tài liệu từ cơ bản tới hiện đại, từ dễ tới khó:* để phù hợp với chương trình đào tạo liên thông từ học nghề lên cao đẳng, đại học và trên đại học.

- *Tạo điều kiện cho người tự học:* Với chương trình mô phỏng Pinnacle có trong máy tính, người tự học có thể quan sát được các bộ phận và hoạt động của chúng, của VDK. Các ví dụ về các chương trình của tài liệu là các chương trình mẫu, các ứng dụng cơ bản, người tự học chỉ cần sao chép lại vào chương trình mô phỏng Pinnacle trên máy tính là có thể hợp dịch và chạy được (từng bước và cả chương trình). Nếu có kit thực tập, người dùng có thể nạp chương trình vào bộ nhớ PROM bên trong hay bộ nhớ ROM bên ngoài VDK hoặc tải sang bộ nhớ RAM bên ngoài và cho chạy chương trình sau khi nhấn nút RESET. Với các bài thực hành, người dùng có thể tự học theo chỉ dẫn chi tiết.

- *Tạo điều kiện cho người nghiên cứu, thiết kế và chế tạo:* các ví dụ hay các chương trình mẫu chính là các vấn đề người nghiên cứu có thể dựa vào đó để thiết kế, chế tạo hệ dùng VDK riêng của mình và phát triển chương trình của mình.

- *Lập trình dạng lệnh gộp (macro lệnh):* để tiện dụng, vì lệnh gộp đặt trong thư viện chương trình, khi dùng, gọi và thay tham số bằng hướng dẫn biên dịch \$INCLUDE MYLIB.ASM.

Tài liệu "Cấu trúc, lập trình, ghép nối và ứng dụng vi điều khiển" này sắp tới sẽ ra mắt 2 tập như sau:

Tập một: *Cấu trúc và lập trình VDK 8051/8052* với 10 chương về VDK 8051/8052 và cũng là các vấn đề cơ bản của các VDK khác. Phần này trình bày về cấu trúc và hoạt động của vi xử lý, nhớ chương trình PROM bên trong, nhớ dữ liệu RAM bên trong, các cổng song song, cổng nối tiếp, bộ định thời/dếm (Timer/Counter) và điều khiển ngắt. Về phần mềm, trình bày cách lập trình (soạn thảo, dịch, kết nối và chạy thử trên chương trình mô phỏng Pinnacle), cách nạp chương trình vào PROM bên trong hay vào RAM ngoài để chạy chương trình trên kit VDK (giá thực hành có VDK, đèn hiển thị, cổng song song, bộ nhớ ROM, RAM mở rộng bên ngoài). Song song với mô tả cấu trúc và hoạt động của các bộ phận, các nhóm lệnh cũng được trình bày với các ví dụ điển hình.

Tập hai: *Ghép nối cơ bản và ứng dụng VDK* với 8 chương, trình bày cách ghép nối và chương trình phục vụ cho các đèn hiển thị (LED đơn, LED 7 đoạn, ma trận diot, màn hình tinh thể lỏng LCD), bàn phím, bộ nhớ ngoài (ROM, RAM) mở rộng, cổng song song mở rộng, cổng nối tiếp mở rộng (USB), hệ VDK với chương trình giám sát (monitor).

Người dùng tài liệu này có thể tự học hay hướng dẫn người khác học các kiến thức cơ bản và nâng cao về cấu trúc, hoạt động, lập trình và ứng dụng cho VDK với kit thực hành VDK và chương trình mô phỏng Pinnacle, proteus trong máy vi tính.

Tài liệu này được biên soạn cho ai muốn học về VDK, với mọi trình độ (học nghề, cao đẳng, đại học và sau đại học) với các ngành chuyên môn liên quan tới VDK (điện, điện tử, tin học, cơ-tin, vật lý kỹ thuật, tự động hóa v.v...).

Tác giả xin cảm ơn các Ban lãnh đạo và các đồng sự ở trường Đại học Bách khoa Hà Nội, Trường Đại học Sư phạm Kỹ thuật Nam Định, Viện Kỹ thuật Hạt nhân Hà Nội, nơi tác giả đã công tác và cộng tác về sự giúp đỡ và tạo điều kiện hoàn thành tài liệu này.

Tác giả mong được các độc giả đóng góp ý kiến cho tài liệu được hoàn thiện hơn. Thụ góp ý xin gửi về Công ty Cổ phần Sách Đại học - Dạy nghề, Nhà xuất bản Giáo dục, 25 Hàn Thuyên, Hà Nội. Điện thoại: 04 8.264974.

Tác giả

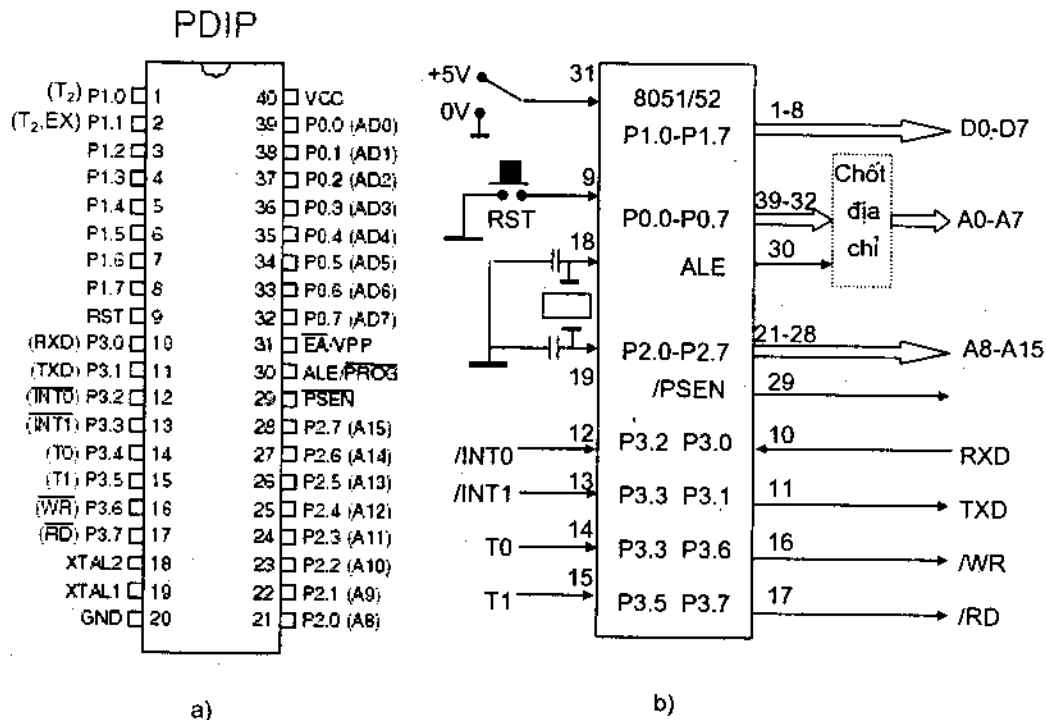
Chương 1

CẤU TRÚC VÀ HOẠT ĐỘNG CỦA VI ĐIỀU KHIỂN 8051/8052

Mục đích của chương:

- Giới thiệu về cấu trúc của VDK gồm CPU, bộ nhớ chương trình (PROM), bộ nhớ dữ liệu (RAM), cổng vào/ra song song, cổng vào/ra nối tiếp, Timer/Counter và bộ điều khiển ngắt.
- Giới thiệu về hoạt động của VDK: RESET, xung nhịp, dạng lệnh mã, quá trình thực hiện lệnh, các nhóm lệnh và các hành động cơ bản như vào/ra song song, vào/ra nối tiếp, định thời gian, đếm, điều khiển ngắt.
- Giới thiệu về cấu trúc của VDK qua chương trình mô phỏng Pinnacle.
- Giới thiệu về các kit thực hành VDK.
- Tổng quát về các tính năng của các VDK thuộc họ MCS-51.

1.1. CẤU TRÚC CỦA VDK 8051/8052



Hình 1-1. Sơ đồ chân theo số chân (a) và theo chức năng (b)

VĐK 8051 được hãng Intel chế tạo năm 1981. Hiện nay được dùng nhiều nhất vì giá rẻ và tiện dụng.

1.1.1. Sơ đồ chân

VĐK 8051/8052 là những vi mạch tích hợp mật độ lớn, có 40 chân. Sơ đồ chân của chúng như hình 1-1.

Số 40 chân trên chia thành các nhóm: nguồn nuôi, điều khiển, các tín hiệu địa chỉ, dữ liệu.

1. Nhóm chân nguồn nuôi

+ Nguồn nuôi 5 V (chân số 40): để nuôi vi mạch.

+ Nối đất (chân số 20): để nối đất.

2. Nhóm chân điều khiển

Nhóm này còn phân biệt các tín hiệu điều khiển vào và ra.

a) Nhóm tín hiệu vào điều khiển

+ XTAL1 (chân số 18) và XTAL2 (chân số 19): nối tinh thể thạch anh cho máy phát xung nhịp chu trình hay mắc với máy phát xung ngoài.

+ RST (RESET) (chân số 9): nối chuyển mạch (khi nhấn, nối với đất) để xóa về trạng thái ban đầu hay khởi động lại.

+ /EA/PP (chân số 31): chọn nhớ ngoài (nối đất) hay nhớ trong (nối nguồn nuôi 5 V).

+ T2 hay P1.0 (chân số 1): tín hiệu dữ liệu vào/ra hay tín hiệu vào đếm cho Timer2/Counter2 của 8052.

+ T2EX hay P1.1 (chân số 2): tín hiệu dữ liệu vào/ra hay tín hiệu vào ngắt ngoài2 cho 8052.

+ /INT0 hay P3.2 (chân số 12): tín hiệu vào gây ngắt ngoài0 cho VĐK 8051.

+ /INT1 hay P3.3 (chân số 13): tín hiệu vào gây ngắt1 cho VĐK 8051.

+ T0 hay P3.4 (chân số 14): tín hiệu vào đếm cho Timer0/Counter0.

+ T1 hay P3.5 (chân số 15): tín hiệu vào đếm cho Timer1/Counter1.

b) Nhóm tín hiệu ra điều khiển

+ ALE//PROG (chân số 30): dùng để đưa tín hiệu chốt địa chỉ (ALE) khi có nhớ ngoài hay điều khiển ghi chương trình /PROG.

+ /PSEN (chân số 29): dùng để đưa tín hiệu điều khiển đọc bộ nhớ chương trình ROM ngoài.

+ /WR hay P3.6 (chân số 16): để đưa tín hiệu dữ liệu vào/ra hay tín hiệu ghi dữ liệu vào bộ nhớ ngoài.

+ /RD hay P3.7 (chân số 17): để đưa tín hiệu dữ liệu vào/ra hay tín hiệu đọc dữ liệu từ bộ nhớ ngoài.

c) Nhóm các tín hiệu địa chỉ, dữ liệu

+ Cổng vào/ra địa chỉ, dữ liệu P0 hay P0.0-P0.7 (chân số 39-32): dùng để trao đổi tin về dữ liệu (D0-D7) vào/ra song song hoặc đưa ra các địa chỉ thấp (A0-A7) theo chế độ dồn kênh (kết hợp với tín hiệu chốt địa chỉ ALE).

+ Cổng vào/ra địa chỉ dữ liệu P2 hay P2.0-P2.7 (chân số 21-28): dùng để trao đổi tin song song về dữ liệu (D0-D7) hoặc đưa ra các địa chỉ cao (A8-A15).

+ Cổng vào/ra dữ liệu P1 hay P1.0-P1.7 (chân số 1-8): dùng để trao đổi tin song song về dữ liệu (D0-D7).

+ Cổng vào/ra P3 hay P3.0-P3.7:

- P3.0 (chân số 10): đưa vào tín hiệu nhận tin nối tiếp RXD.
- P3.1 (chân số 11): đưa ra tín hiệu truyền tin nối tiếp TXD.
- /INT0 hay P3.2 (chân số 12): tín hiệu vào gây ngắt0 INT0 của VĐK 8051/8052.
- /INT1 hay P3.3 (chân số 13): tín hiệu vào gây ngắt1 INT1 của VĐK 8051/8052.
- T0 hay P3.4 (chân số 14): tín hiệu vào đếm cho Timer0/Counter0 của VĐK 8051/8052.
- T1 hay P3.5 (chân số 15): tín hiệu vào đếm cho Timer1/Counter1 của VĐK 8051/8052.
- /WR hay P3.6 (chân số 16): để đưa ra tín hiệu ghi dữ liệu vào bộ nhớ ngoài.
- /RD hay P3.6 (chân số 17): để đưa ra tín hiệu đọc dữ liệu từ bộ nhớ ngoài.
- T2 hay P1.0 (chân số 1): tín hiệu vào đếm cho Timer2/Counter2 của 8052.
- T2EX (chân số 2): tín hiệu vào gây ngắt ngoài2 của VĐK 8052.

Ngoài các tín hiệu chuyên dùng trên, cổng vào/ra P3 này còn dùng để trao đổi tin về dữ liệu D7-D0.

1.1.2. Sơ đồ khối

VĐK 8051/8052 được chế tạo theo cấu trúc của một hệ vi tính gồm các khối (hình 1-2a) sau:

- Bộ xử lý trung tâm CPU (Central Processor Unit): dùng để điều khiển toàn vi mạch trong việc thực hiện lệnh và xử lý số học, logic.

- Bộ nhớ RAM bên trong: (128 byte cho 8051 và 256 byte cho 8052), dùng làm các thanh ghi thông dụng, đặc biệt SFR và xử lý bit để ghi nhớ dữ liệu cho chương trình.

- Bộ nhớ chương trình (ROM) bên trong: (4 kbyte cho 8051 hay 8 k cho 8052), dùng để ghi nhớ chương trình.

- Các cổng trao đổi tin vào/ra dữ liệu và địa chỉ P0-P2: dùng trao đổi tin song song về địa chỉ và dữ liệu (D0-D7) như sau:

+ Cổng P0: trao đổi tin về dữ liệu (D0-D7) hay khi có bộ nhớ ngoài mở rộng, trao đổi tin về địa chỉ thấp (A0-A7) (cùng với tín hiệu chốt địa chỉ ALE đưa qua vi mạch chốt).

+ Cổng P2: trao đổi tin về dữ liệu (D0-D7) hay địa chỉ cao (A8-A15) khi có bộ nhớ ngoài mở rộng.

+ Cổng P1: trao đổi tin về dữ liệu song song D7-D0.

- Cổng trao đổi tin nối tiếp: dùng trao đổi tin nối tiếp nhận vào từ chân RXD và đưa ra truyền tới chân TXD.

- Khối định thời gian/bộ đếm (Timer/Counter 0,1 và 2 cho 8052): dùng để định thời gian và đếm xung ngoài vào các chân T0, T1, T2.

- Khối điều khiển ngắt chương trình: dùng để ngắt chương trình khi có đếm tràn hay có xung ngoài tác động vào.

Như vậy, vi mạch VĐK nói chung và VĐK 8051/8052 nói riêng là một hệ vi xử lý, bao gồm các khối CPU, nhớ (ROM, RAM), cổng vào/ra (song song, nối tiếp), bộ định thời gian/đếm và điều khiển ngắt chương trình. Các chương sau, chúng ta sẽ nghiên cứu chi tiết từng khối trên cùng với hệ lệnh và lập trình phục vụ.

Sơ đồ khối chi tiết (hình 1-2b) mô tả tới từng khối nhỏ và từng thanh ghi của sơ đồ khối tổng quát, trong đó có :

- *Khối thực hiện lệnh*: để thực hiện lệnh gồm đưa địa chỉ lệnh (từ thanh ghi PC), đọc lệnh vào thanh ghi lệnh, giải mã lệnh và phát các tín hiệu điều khiển các thanh ghi.

- *Khối số học logic ALU*: để thực hiện các phép toán số học (cộng, trừ, nhân chia, tăng, giảm) và logic (Và, Hoặc, Loại trừ v.v...).

- *Các thanh ghi thông dụng*: A, B, PC, PSW, SP, các cổng vào/ra song song P0, P1, P2, P3.

Chúng ta sẽ xét chi tiết các thanh ghi và các khối trên trong các chương sau.

1.2. HOẠT ĐỘNG CỦA VĐK 8051/8052

1.2.1. Nguồn nuôi

VĐK 8051/8052 được nuôi bằng nguồn nuôi +5V. Thanh ghi PCON có hai bit để điều khiển nguồn nuôi ở chế độ đặc biệt:

- *Bit IDL (PCON.0) = 1*: khiến VĐK ở chế độ rỗng (không hoạt động), các tín hiệu nhịp bên trong không tác động lên CPU, nhưng vẫn tác động lên các bộ phận điều khiển ngắt, Timer và cổng nối tiếp. Chế độ này kết thúc khi có RESET.

- *Bit PD (PCON.1) = 1*: khiến VĐK ở chế độ nguồn nuôi thấp, bộ dao động bên trong ngừng phát xung, các chức năng không hoạt động, trừ RAM bên trong và các thanh ghi SFR còn duy trì dữ liệu. Ở chế độ này, nguồn nuôi Vcc có thể còn 2 V. Chế độ này kết thúc khi có RESET.

- *Bit GF0 (PCON.2) = 1*: bit cờ tổng quát.

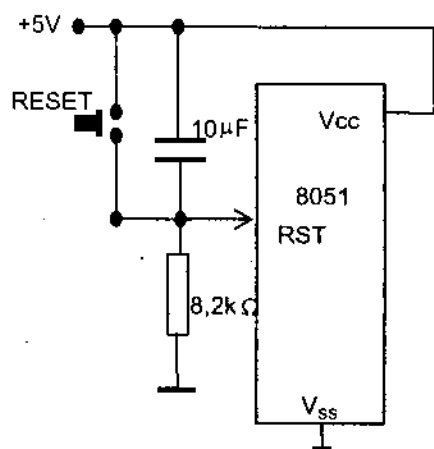
- *Bit GF1 (PCON.3) = 1*: bit cờ tổng quát.

- *Các bit (PCON.3-PCON.6)*: để dành, chưa dùng.

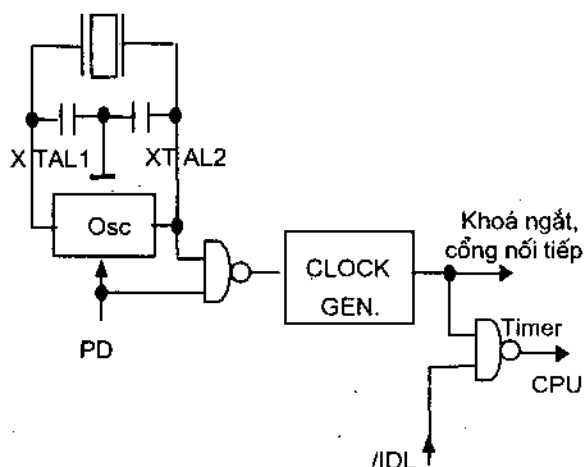
- *Bit SMOD (PCON.7)*: tốc độ baud nhân 2 cho trao đổi tin nối tiếp.

1.2.2. RESET

Trạng thái khởi động lại hay xóa khi bật nguồn nuôi hay ấn nút RESET để duy trì một xung âm nối đất vào chân RESET (chân số 9) trong thời gian 2 chu trình máy (cỡ 1μs cho tần số 12MHz) do mạch ở hình 1-3 khi có bộ nhớ ngoài mở rộng.



Hình 1-3. Mạch điện Reset



Hình 1-4. Máy phát xung nhịp

Sau khi bật nguồn nuôi hay ấn nút RESET thì :

- giá trị của thanh ghi SP = 07h;

- các giá trị của các cổng song song là 0FFH;
- các giá trị của các thanh ghi hay các ô nhớ là 00FFH;
- giá trị của thanh ghi PC = 00H nên địa chỉ của lệnh đầu tiên của chương trình chính là 00H.

1.2.3. Máy phát xung nhịp và chu trình

VĐK hay vi xử lý hoạt động được do chuỗi xung nhịp kích thích các mạch số. Do đó, mỗi VĐK phải có một máy phát xung nhịp với tần số thích hợp. VĐK 8051 cần tần số 11,0592 MHz. Máy phát xung nhịp có sẵn trong vi mạch VĐK, chỉ cần nối tinh thể thạch anh có tần số tương ứng vào các chân XTAL1, XTAL2 như hình 1-4. Có thể dùng máy phát xung nhịp bên ngoài, khi đó, chỉ cần đưa tín hiệu dao động ngoài vào chân XTAL1, còn chân XTAL2 để trống.

Các chu trình xung nhịp cho lệnh một byte và hai byte như hình 1-5.

Như vậy, chuỗi xung nhịp được chia thành:

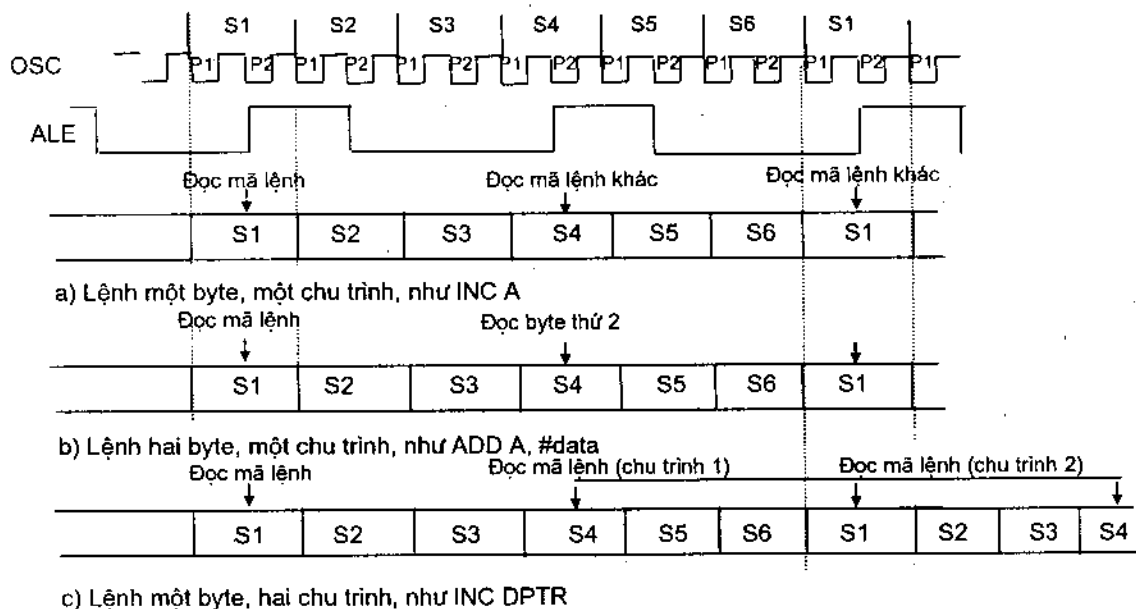
- *Chu trình* : gồm 6 nhịp, từ S1 tới S6.
- *Pha* : mỗi nhịp có hai pha P1, P2. Mỗi pha có độ kéo dài cỡ 1 microsec (tần số nhịp 12 MHz).

Tại các nhịp S1 và S4 được tiến hành đọc mã lệnh hoặc byte thứ hai khi có tín hiệu chốt địa chỉ ALE = 1.

Hình 1-5a ứng với lệnh INC A (tăng nội dung thanh ghi chứa A), là lệnh một byte, được thực hiện trong một chu trình (cỡ 12 μ sec).

Hình 1-5b ứng với lệnh ADD A, #data (cộng thanh ghi chứa A với dữ liệu trực tiếp data), là lệnh hai byte (byte mã lệnh và byte dữ liệu data) thực hiện trong một chu trình (6 nhịp).

Hình 1-5c ứng với lệnh INC DPTR (tăng thanh ghi địa chỉ nhớ ngoài) gồm 1 byte, thực hiện trong hai chu trình (12 nhịp).



Hình 1-5. Các chu kỳ xung nhịp cho lệnh một byte (a) và hai byte (b) một chu trình và một byte trong hai chu trình (c)

Ngoài các lệnh trên còn có lệnh hai byte, hai chu trình như MOVX, @DPTR, A (đưa nội dung thanh ghi A ra ô nhớ ngoài có địa chỉ là giá trị của thanh ghi DPTR).

Ở đây, VĐK chọn địa chỉ ô nhớ chương trình ngoài (PROM) hay RAM ngoài (EXRAM) bởi tín hiệu địa chỉ ALE (để chốt địa chỉ thấp A0-A7).

Tín hiệu đưa ra ở cổng P2 là địa chỉ cao (A8-A15), do thanh ghi đếm lệnh nửa cao PCH đưa ra (PCH OUT cho PROM) hay nửa cao của thanh ghi con trỏ địa chỉ DPH (cho EXRAM).

Cổng P1 đưa ra địa chỉ thấp A0-A7, do thanh ghi đếm lệnh thấp đưa ra (PCL OUT) hay nửa thấp của thanh ghi con trỏ địa chỉ DPL.

Khi có tín hiệu $/PSEN = 0$ bộ nhớ chương trình (PROM) đọc lệnh vào (ISNT IN) trên đường dây dữ liệu P0.

Khi có tín hiệu $/RD = 0$, ô nhớ RAM ngoài đọc dữ liệu vào (DATA IN) cũng trên đường dây dữ liệu P0.

1.2.4. Lệnh dạng ngôn ngữ mã (mã lệnh)

Giữa người điều hành (hay sử dụng) và VĐK "trao đổi" tin (dữ liệu, lệnh hành động) bằng loại ngôn ngữ chung dưới dạng mã (tập hợp các bit nhị phân 0, 1) quy ước do người thiết kế, chế tạo đề ra. Với ngôn ngữ dạng mã, VĐK có thể hiểu một cách trực tiếp, không cần thông dịch (intepreter - dịch và thực hiện theo từng lệnh) hay biên dịch (compiler - dịch cả văn bản) thành dạng mã. Một lệnh đầy đủ đều có bốn thành phần là nhãn, mã lệnh, các toán hạng (Operand) nguồn S và toán hạng đích D và chú thích (Comment).

Dạng một lệnh như sau: Label: OPCODE D, S ; Comment
trong đó:

- *Label*: gọi là nhãn hay ký hiệu, là một hay một số ký tự, không được trùng với các từ khoá hay tên lệnh, không bắt đầu bằng một chữ số, dùng để đánh dấu địa chỉ của lệnh khi chương trình cần nhảy tới khi dùng lệnh nhảy (JMP ABC) hay gọi chương trình con (CALL SUBR).

- *OPCODE*: là mã lệnh hay mã của hành động. Mã lệnh này (tổ hợp các bit 1 và 0) là do người thiết kế, chế tạo VĐK quy định dựa theo cấu trúc và hoạt động của VĐK. Mã lệnh này chứa trong một byte (8 bit). Do đó, một VĐK có thể có tới 256 lệnh với các số mã khác nhau.

Ví dụ: + VĐK 8051/8052 có lệnh NOP (NO operation-không hành động gì): có mã lệnh 00000000B = 00H. Gặp lệnh này, thanh ghi đếm lệnh PC tăng lên 1 và trễ thời gian một chu trình (1 μ s).

+ Lệnh INC A (tăng nội dung thanh ghi chứa A lên 1) có mã lệnh 00000100B = 04H.

+ Lệnh MOV A, #data (chuyển dữ liệu tức thì data vào thanh ghi A có mã 0111 0100B data hay 74H, data.

- *D (Destination)*: là toán hạng đích (nơi gửi dữ liệu sau khi hành động).

- *S (Source)*: là toán hạng nguồn (nơi chứa dữ liệu cần cho hành động).

Các hành động, toán hạng nguồn hay đích đều được quy ước dưới dạng nhị phân hay dạng mã.

Toán hạng là nơi chứa dữ liệu cho lệnh, có thể là dữ liệu tức thì (immediate - dữ liệu ghi ngay sau mã lệnh), trong thanh ghi R (phương pháp địa chỉ thanh ghi) hay trong ô nhớ (phương pháp địa chỉ trực tiếp hay gián tiếp thanh ghi hoặc chỉ số). Chúng ta sẽ đề cập tới các phương pháp địa chỉ ở chương ba.

- *Comment (chú thích)*: sau dấu chấm phẩy (;), dùng để ghi chú, giải thích thêm về lệnh và chương trình. Khi dịch chương trình, phần chú thích này được bỏ qua, không dịch và khi chạy chương trình, không thực hiện. Nếu quên dấu chấm phẩy, chương trình dịch sẽ báo lỗi vì sai cú pháp của câu lệnh.

Trong VĐK có các lệnh không có mã hành động, không có toán hạng hay chỉ có một toán hạng.

Ví dụ: - Lệnh NOP có mã 00000000B hay 00H, không thực hiện một hành động nào, chỉ tốn thời gian đọc lệnh là 1 chu trình máy (1 microsec).

- Lệnh INC A có mã là 00000100B hay 04H : lệnh không có toán hạng S hay D, làm tăng thanh ghi chứa A lên 1 đơn vị (A+1), không có toán hạng nguồn S và đích D.

- Lệnh MOV A, #data có toán hạng nguồn là data kèm sau mã lệnh, toán hạng đích là A.

- Lệnh ADD A, #data có mã là 0010 0100 (#data) hay 24H (#data), gồm hai byte mã lệnh và toán hạng nguồn 1 là data. Thanh ghi chứa A vừa là toán hạng nguồn 2 (cộng data với A) rồi lại gửi kết quả tổng vào thanh ghi A.

Tuỳ hành động, số toán hạng và phương pháp địa chỉ hay phương pháp tìm dữ liệu cho lệnh, ta có lệnh dài 1, 2 hay 3 byte và thời gian thực hiện cần 1, 2 chu trình máy.

1.2.5. Quá trình thực hiện một lệnh

Lệnh của chương trình được chứa trong bộ nhớ ROM (trong hay ngoài), còn dữ liệu hay các thanh ghi được chứa trong bộ nhớ RAM. VDK nói chung hay CPU nói riêng, khi thực hiện một lệnh chứa trong bộ nhớ theo các bước sau:

- CPU đưa ra nội dung thanh ghi đếm lệnh PC, tức địa chỉ ô nhớ ROM (trong hay ngoài) để đọc mã lệnh vào thanh ghi đếm lệnh IR.

- CPU đưa ra nội dung thanh ghi đếm lệnh PC+1 để đọc byte thứ hai của lệnh tức toán hạng nguồn S nếu lệnh cần toán hạng S.

- CPU đưa ra nội dung thanh ghi đếm lệnh PC+2 để đọc byte thứ ba của lệnh tức toán hạng nguồn D nếu lệnh cần toán hạng D.

- CPU giải mã lệnh để biết cần đưa các tín hiệu nào tới bộ phận tương ứng. Ví dụ lệnh INC A có mã 04H sẽ làm tăng nội dung thanh ghi chứa A lên 1 tức đưa một tín hiệu vào thanh ghi đếm A để nội dung A tăng lên 1.

Nếu lệnh có toán hạng nguồn S, dữ liệu sẽ được đọc vào thanh ghi đệm tạm thời (xem hình 1-2b) của sơ đồ khối chi tiết của VDK 8051/8052) và sau đó lệnh sẽ thao tác với toán hạng nguồn rồi gửi vào toán hạng đích D.

Ví dụ, lệnh ADD A, #data, thì thanh ghi chứa A sẽ được khối xử lý số học logic ALU cộng thanh ghi chứa A với dữ liệu data ở thanh ghi đệm, kết quả gửi lại vào A và đưa chỉ thị kết quả về trạng thái trên thanh ghi trạng thái cờ PSW.

1.2.6. Chương trình

Chương trình là một nhóm lệnh, có ký hiệu mở đầu và kết thúc chương trình, thực hiện một nhiệm vụ nào đó của VDK. Chương trình có thể viết dưới dạng:

- *Mã lệnh*: là chuỗi các lệnh dạng mã hay các con số dạng nhị phân (các bit 0, 1) hay dạng HEX. Dạng chương trình này khó viết, dễ nhầm lẫn.

- *Hợp ngữ*: là chuỗi các lệnh dạng chữ hay thuật nhớ (mỗi lệnh là tập hợp các chữ viết tắt bằng tiếng Anh đặc trưng cho hành động của VDK). Các lệnh và chương trình hợp ngữ này khắc phục nhược điểm của mã lệnh nhưng đòi hỏi chương trình biên dịch (Compiler hay Assembler) từ ngôn ngữ bậc cao hay hợp ngữ sang dạng mã. Cuốn giáo trình này sẽ đề cập đến lập trình hợp ngữ cho VDK họ MCS-51.

- *Ngôn ngữ bậc cao*: vì chương trình dạng hợp ngữ dài, phức tạp, tuy có gần với cấu tạo và hành động của VDK, nhưng còn xa với cách suy nghĩ và hành động của tư duy của con người và lại phụ thuộc vào từng loại VDK. Ngôn ngữ bậc cao ra đời khắc phục được nhược điểm trên, nhưng đòi

hồi chương trình biên dịch (Compiler) sang ngôn ngữ mã lệnh. Khi dùng ngôn ngữ bậc cao, đôi khi vẫn phải dùng hợp ngữ trong một chương trình ngôn ngữ bậc cao. Gần đây, cho VĐK họ MCS 51, người ta có các ngôn ngữ C51 và R51 cho một chương trình có cả hợp ngữ và ngôn ngữ bậc cao.

1.2.7. Các hành động cơ bản của VĐK 8051/8052

Từ các khối của sơ đồ khối (hình 1-2), ta thấy VĐK 8051/8052 có các hành động cơ bản sau:

- *Điều khiển thực hiện các lệnh của chương trình*: do bộ phận điều khiển của CPU và nhóm lệnh rẽ nhánh điều khiển.

- *Xử lý số học và logic*: do khối số học logic ALU và nhóm lệnh số học, logic điều khiển.

- *Định thời gian và đếm số xung vào*: do bộ phận Timer/Counter điều khiển.

- *Trao đổi tin song song*: do cổng vào/ra và nhóm lệnh dịch chuyển và xử lý bit điều khiển.

- *Trao đổi tin nối tiếp*: do cổng trao đổi tin nối tiếp điều khiển.

- *Điều khiển ngắt*: do khối điều khiển ngắt điều khiển.

Các chương sau sẽ lần lượt nghiên cứu các hành động cơ bản trên của VĐK 8051/52 cả về cấu trúc (phần cứng) và cách lập trình phục vụ (phần mềm).

1.3. HỆ LỆNH VÀ CÁC PHƯƠNG PHÁP ĐỊA CHỈ CỦA VĐK 8051/8052

1.3.1. Hệ lệnh

Tương tự vì xử lý họ 8086, VĐK 8051/8052 có một hệ lệnh với các nhóm lệnh sau:

- *Nhóm lệnh điều khiển thực hiện chương trình hay rẽ nhánh*: nhảy không điều kiện, nhảy có điều kiện, gọi chương trình con.

- *Nhóm lệnh xử lý số học*: cộng, trừ, nhân, chia, tăng, giảm, lấy bù 1, bù 2 v.v...

- *Nhóm lệnh xử lý logic*: Và, Hoặc, Loại trừ logic, Quay phải thanh ghi A, Quay trái thanh ghi A.

- *Nhóm lệnh xử lý bit*: thao tác trên các bit.

- *Nhóm lệnh dịch chuyển*: dịch chuyển dữ liệu giữa các thanh ghi và ô nhớ.

Chúng ta sẽ lần lượt nghiên cứu các nhóm lệnh này ở các chương sau.

1.3.2. Các phương pháp địa chỉ

Tùy theo toán hạng hay dữ liệu của lệnh mà ta có các phương pháp địa chỉ, độ dài lệnh cũng như thời gian thực hiện lệnh khác nhau. VĐK 8051/8052 có các phương pháp địa chỉ sau:

- *Địa chỉ tức thì*: ký hiệu #data, dữ liệu tức thì là data là một con số (dạng thập phân từ 0, 1, 2, ... 9), dạng 16 hay HEXADECIMAL (HEX) (từ 0, 1, 2, ... 9, A, B, C, D, E, F với chữ số 0 đứng trước các chữ A, B,...), được viết liền (tức thì) ngay trong lệnh.

Ví dụ, MOV A, #55H là gửi dữ liệu tức thì 55H vào thanh ghi A.

- *Địa chỉ thanh ghi*: ký hiệu Ri, dữ liệu trong thanh ghi Ri. Ví dụ MOV A, R0 là gửi dữ liệu trong thanh ghi R0 đưa vào thanh ghi chứa A.

- *Địa chỉ trực tiếp*: ký hiệu data, data là địa chỉ ô nhớ có chứa dữ liệu.

Ví dụ, MOV A, data là gửi dữ liệu của ô nhớ có địa chỉ là data (con số trực tiếp sau lệnh là data) vào thanh ghi A.

- *Địa chỉ gián tiếp thanh ghi*: ký hiệu @Ri, địa chỉ ô nhớ là nội dung thanh ghi Ri, với i = 0, 1 tức R₀, R₁).

Ví dụ, MOV A, @R0 là gửi dữ liệu là nội dung ô nhớ có địa chỉ là nội dung thanh ghi R₀.

- *Địa chỉ chỉ số*: phương pháp địa chỉ này dùng để đọc bảng dữ liệu có trong ô nhớ có địa chỉ là tổng số của thanh ghi A với thanh ghi đếm lệnh PC (8 bit) hoặc thanh ghi con trỏ địa chỉ DPTR (16 bit).

Ví dụ: MOV A, @A+PC; đọc nội dung ô nhớ có địa chỉ A+PC vào A.

hay MOVC A, @A+ DPTR; đọc nội dung ô nhớ chương trình có địa chỉ A+DPTR vào A.

Các lệnh này dùng để đọc bảng dữ liệu chứa trong bộ nhớ chương trình (MOVC).

Chúng ta sẽ xét chi tiết hơn về các phương pháp địa chỉ này ở chương 3.

1.4. CHƯƠNG TRÌNH MÔ PHỎNG (SIMULATOR) CỦA VĐK 8051/8052

1.4.1. Các chương trình mô phỏng

Người ta đã viết chương trình mô phỏng cấu trúc và hoạt động của VĐK trên máy vi tính PC - IBM để học tập và lập trình cho VĐK. Điều đó có nghĩa là:

- Mô tả cấu trúc và hoạt động của các khối của VĐK 8051/8052 bằng chương trình của máy tính cá nhân PC.

- Mô tả hoạt động của hệ lệnh (dạng hợp ngữ, dạng mã lệnh - ngôn ngữ máy) của VĐK sang hệ lệnh của PC. Nói khác đi, các lệnh của VĐK 8051/8052 được dịch chéo (Cross-Assembler) sang hệ lệnh của PC và thực hiện chúng trên PC. Do đó, chương trình viết cho VĐK 8051/52 được dịch, liên kết và chạy thử trên PC (theo từng bước và cả chương trình).

Như vậy, không cần VĐK 8051/8052, với chương trình mô phỏng VĐK 8051/8052, chúng ta có thể tìm hiểu về cấu trúc, hoạt động và lập trình cho VĐK 8051/8052 chỉ trên máy tính PC. Với chương trình cho VĐK đã hợp dịch và liên kết (đuôi .HEX) trên PC bằng chương trình mô phỏng, người ta có thể nạp vào VĐK (bằng bộ nạp chương trình-programmer) để chạy hay có thể truyền sang bộ nhớ RAM ngoài để chạy trên hệ VĐK.

Có hai loại chương trình mô phỏng cho VĐK 8051/8052, loại trên môi trường của hệ điều hành MS-DOS và trên môi trường của hệ điều hành Windows.

Loại trong môi trường MSDOS, thông dụng có hai chương trình SIM51 và AVSIM.

Loại trong môi trường WINDOWS có các chương trình 8051 IDE, 8052 SIMULATOR, PINNACLE và Proteus.

Chúng ta sẽ nghiên cứu chi tiết hai chương trình điển hình của hai loại trên là AVSIM51 và PINNACLE, còn chương trình Proteus ở phụ chương của tập II.

Đặc điểm của chương trình mô phỏng trong môi trường MSDOS là:

- Thực hiện trong môi trường DOS bằng lệnh MSDOS Prompt (vào C:\ tên chương trình mô phỏng (có trong đĩa C) <Enter>).

- Trong quá trình hợp dịch, liên kết, biến đổi dữ liệu phải gõ từ bàn phím từng tên chương trình và tên các tệp tương ứng. Trên màn hình có thông báo lỗi khi biên dịch chương trình nguồn dạng hợp ngữ (đuôi .ASM).

- Chương trình mô phỏng hiện lên màn hình toàn bộ hay từng bộ phận (các thanh ghi, các vùng nhớ, các Timer/Counter, các cổng song song, cổng nối tiếp v.v...) để người sử dụng theo dõi sự thay đổi nội dung của chúng trong quá trình thực hiện lệnh của chương trình đã biên dịch và kết nối.

Đặc điểm của chương trình mô phỏng trong môi trường WINDOWS là:

- Thực hiện trong môi trường WINDOWS nên không phải chuyển sang môi trường DOS bằng lệnh MSDOS Prompt.

- Trong quá trình hợp dịch, liên kết, biến đổi dữ liệu không phải gõ từ bàn phím từng tên chương trình và tên các tệp tương ứng mà chỉ cần nháy đúp chuột vào tên hay biểu tượng của chương trình cần thực hiện. Trên màn hình có thông báo lỗi khi biên dịch chương trình nguồn dạng hợp ngữ (đuôi .ASM).

- Phải gọi hiển thị từng bộ phận của VDK của chương trình mô phỏng lên màn hình. Ta không có một bức tranh toàn cảnh về cấu trúc của VDK. Điều này hạn chế và có khó khăn trong việc theo dõi kết quả thực hiện lệnh của chương trình. Tuy nhiên, chúng ta có thể theo dõi một cách tường minh được sự thay đổi của các thanh ghi và các bộ phận của VDK như định thời/dếm, trao đổi tin nối tiếp, ngắt chương trình.

Tuy có hạn chế này, chương trình mô phỏng trong WINDOWS vẫn được ưa chuộng hơn vì việc sử dụng đơn giản, thuận tiện và nhanh chóng, chỉ nháy chuột thay cho gõ từng ký tự của lệnh và tên tệp từ bàn phím như chương trình trong môi trường MSDOS. Trong các ví dụ của tài liệu này, chúng ta dùng chương trình trong WINDOWS là PINNACLE.

1.4.2. Chương trình mô phỏng AVSIM51

Chương trình mô phỏng này xuất hiện sau chương trình mô phỏng SIM và trước chương trình mô phỏng Pinnacle, nhưng hoạt động trên cơ sở của hệ điều hành MSDOS nên mỗi khi đưa lệnh điều khiển phải gõ phím từ bàn phím, nên hiện nay không thông dụng. Giao diện của chương trình mô phỏng AVSIM như hình 1-6 có nội dung các thanh ghi, bộ nhớ của VDK, tương tự chương trình mô phỏng PINNACLE.

LABEL OPERATION		8051/8751 AVSIM 8051 Simulator/Debugger V1.30			
RESET	no memory	CPU REGISTERS	FLAGS	SCL SPD DSP SKP	CURSOR
0001H	no memory	C Accumulator AC F0 OV P OFF HI ON OFF MENU			
0002H	no memory	0 00000000:00:	0 0 0 0	Cycles:	
0003H	no memory	addr data			
0004H	no memory	PC:0000 » FF FF FF FF Timers TH/TL TF/TR G/T/M1/M0			
0005H	no memory	SP: 07 » 00 00 00 00 T0: 00 00 0 0 0 0 0 0			
0006H	no memory	00 00 00 00 T1: 00 00 0 0 0 0 0 0			
0007H	no memory	DP:0000 » FF FF FF FF			
0008H	no memory	R0:00: » 00: RB:00 Ints A S TI X1 T0 X0 Edg IT IE			
0009H	no memory	R1:00: » 00: B:00 En 0 0 0 0 0 0 X0: 0 0			
000AH	no memory	R2:00 R4:00 R6:00 Pr 0 0 0 0 0 X1: 0 0			
000BH	no memory	R3:00 R5:00 R7:00 SBUF: In Out PCON:0xxxxxxx			
000CH	no memory	Data Space 00:00: SCON:00000000			
000DH	no memory	0000 00 00 00 00 00 00 00 00		Ports	
000EH	no memory	0008 00 00 00 00 00 00 00 00		P0 11111111	
000FH	no memory	0010 00 00 00 00 00 00 00 00		FF: 11111111	
0010H	no memory	0018 00 00 00 00 00 00 00 00		P1 11111111	
0011H	no memory	Data Space FF: 11111111			
0012H	no memory	0020 00 00 00 00 00 00 00 00		P2 11111111	
0013H	no memory	0028 00 00 00 00 00 00 00 00		FF: 11111111	
0014H	no memory	0030 00 00 00 00 00 00 00 00		P3 11111111	
0015H	no memory	0038 00 00 00 00 00 00 00 00		FF: 11111111	
>Select Command - or use arrow keys					
Dump Expression commandFile Help IO Load --space-- ESC to screen					

Hình 1-6. Giao diện của chương trình mô phỏng AVSIM

Hình 1-6. Giao diện của chương trình mô phỏng AVSIM

1.4.3. Chương trình mô phỏng PINNACLE 52

Chương trình PINNACLE 52 là một IDE (Integrated Equipment- thiết bị tích hợp) gồm:

- Chương trình soạn thảo (Edit) chương trình.

- Chương trình biên dịch (compiler), kết nối và xếp đặt lại (Relocatable) chương trình hợp ngữ (có đuôi .ASM).

- Chương trình biến đổi dạng HEX sang BIN và ngược lại.

- Chương trình mô phỏng (Simulator) để chạy thử trên máy vi tính PC (chạy từng bước, từng chương trình con và cả chương trình).

Giao diện của chương trình mô phỏng Pinnacle được giới thiệu trên hình 1-8 với các thực đơn ngang và dọc (sổ xuống). Chỉ cần nháy chuột, chúng ta có thể gọi được các chương trình của thực đơn ngang và dọc lên màn hình.

Do đó, sau khi gọi chương trình mô phỏng PINNACLE (trong WINDOWS), trên màn hình, ta có thực đơn ngang và kéo xuống với các đề mục sau:

- *FILE (tệp)*: NEW (để soạn thảo chương trình mới dạng .ASM), OPEN (để mở chương trình đã có), CLOSE (đóng chương trình đã mở), EXIT (thoát khỏi PINNACLE về WINDOWS).

- *VIEW (quan sát)*: REGISTER (các thanh ghi A, B, R0-R7, SP, IE, IP, PCON, PSW); DPTR (con trỏ địa chỉ), PORTS (các cổng P0, P1, P2, P3), TIMER (TCON, TMOD, T2CON, T2MOD); SFR Direct view; BIT field; CODE MEMORY (DISSASSEMBLY- nhớ mã lệnh), UART, I2C Device, SPI.

- *EXECUTE (thực hiện)*: RUN (F5- chạy cả chương trình); STOP (dừng chạy chương trình), RESTART (khởi động lại), SINGLE STEP (chạy từng bước); PROCEDURE STEP (chạy chương trình con); SKIP STEP (bỏ qua bước), FINISH SUBROUTINE (kết thúc chương trình con), FINISH INTERRUPT (kết thúc ngắt).

- *SIMULATOR (mô phỏng)*: CLEAR (xóa nhớ chương trình, nhớ trong, nhớ ngoài), LOAD MEMORY (nạp bộ nhớ ngoài, nhớ trong, nhớ chương trình), SAVE MEMORY (lưu trữ); EXECUTION/STACK HISTORY; CODE PROFILER, BREAKPOINTS; IMPORT LIST/MAP FILE.

- *PROJECT (dự án)*: NEW, OPEN, EDIT, PROJECT OPTION, CLOSE, COMPILE FILE (F2- biên dịch); COMPILE + LINK (CTRL+F2- biên dịch và kết nối); REBUILT ALL; STOP BUILD.

- *TOOL (công cụ)*: FILE CONVERSION (HEX-BIN, BIN-HEX); CODE WIZARD (baud Rate wizard).

- *OPTION (tùy chọn)*: IDE OPTION.

- *WINDOWS*: REGISTER PROGRAM.

- *HELP (trợ giúp)*: PINNACLE HELP; 8051 CONTROLLER; ABOUT PINNACLE.

Trong thư mục HELP, chúng ta có thể đọc các chi tiết và hướng dẫn sử dụng PINNACLE và cấu trúc của VĐK 8051.

Ở chương này, chúng ta lưu ý tới mô phỏng cấu trúc của Pinnacle qua thực đơn View với các đề mục sau:

- *Registers* : các thanh ghi A, B, R0-R7, SFR (IP, IE, IP, PCON, PSW, SCON).

- *DPTR*: con trỏ địa chỉ của bộ nhớ ngoài, 16 bit, từ 00H tới FFFFH.

- *Ports*: các cổng song song P0, P1, P2, P3.

- *Timer*: các bộ định thời gian Timer0, Timer1 và Timer2 với các thanh ghi TCON, TMOD, T2CON, T2MOD, TIMER0, 1, 2 và RCAP2H, RCAP2L.

- *Internal ram*: nhớ RAM trong có địa chỉ từ 00H tới FFH.

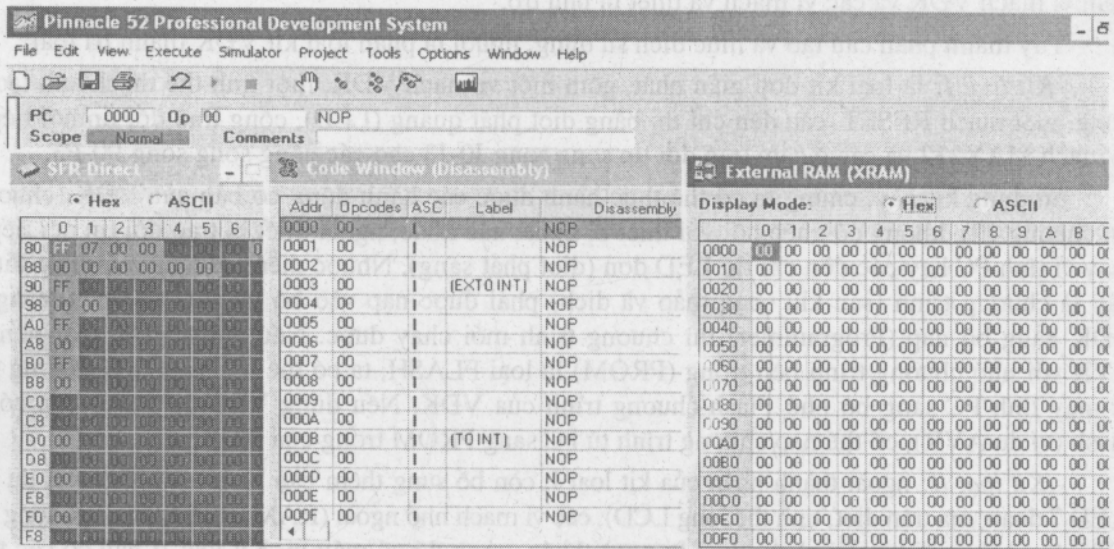
- *External ram*: nhớ RAM ngoài, có địa chỉ từ 00H tới FFFFH.

- *SFR direct view*: nhìn trực tiếp các thanh ghi đặc biệt ở vùng địa chỉ từ 80H tới FFH.

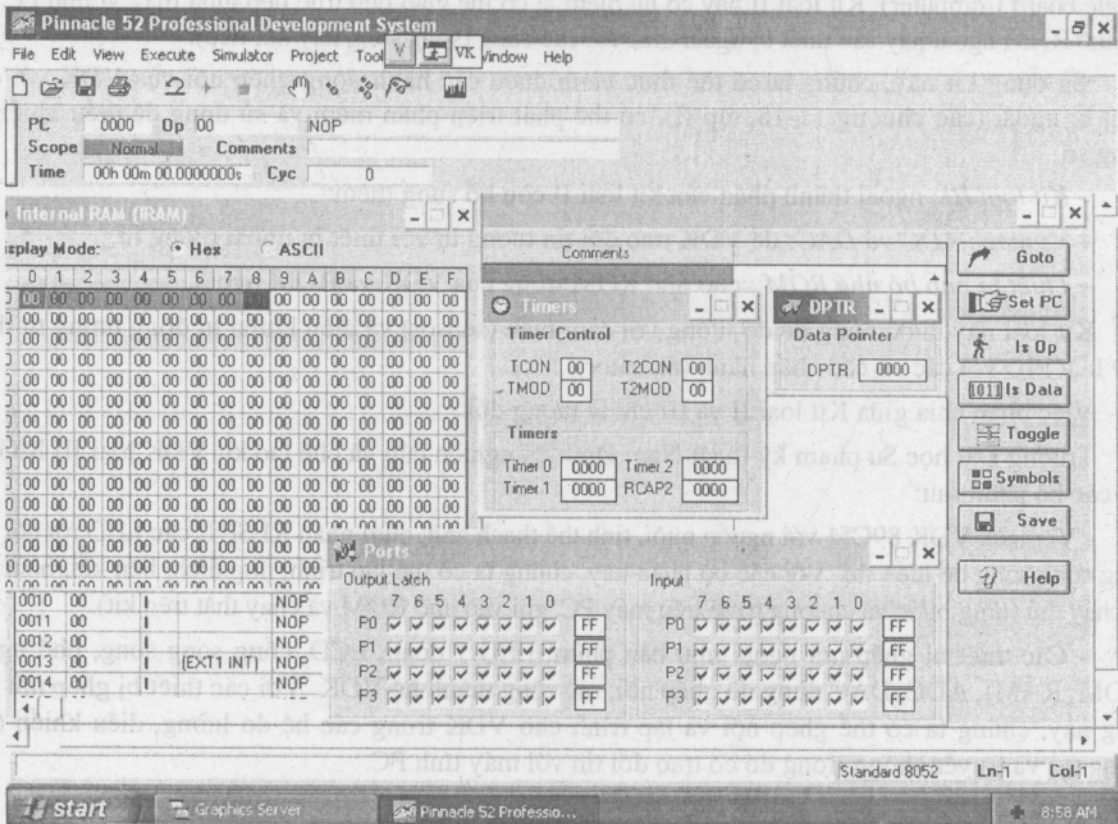
- *Bit field*: vùng địa chỉ bit, có địa chỉ từ 20H tới 2FH để chỉ bit trạng thái của chương trình.

Hình 1-7 giới thiệu các thực đơn con của thực đơn VIEW là SFR (bit FIELD), XRAM, CODE MEMORY.

Hình 1-8 giới thiệu thực đơn VIEW và các thực đơn con của nó: các thanh ghi Register, DPTR, Timer/Counter và IRAM.



Hình 1-7. Pinnacle với thực đơn VIEW : SFR (bit FIELD), XRAM, CODE MEMORY



Hình 1-8. Thực đơn VIEW/ REGISTER, DPTR, PORTS, và IRAM của Pinnacle

Chúng ta sẽ quan sát các bộ phận trên ở phần thực hành và sẽ sử dụng và quan sát chúng trong các hoạt động của VDK trong các chương sau.

1.5. CÁC KIT THỰC HÀNH VDK

Để thực hành VDK, người ta đã chế tạo các kit (giả thực nghiệm), đó chính là một hệ VDK, gồm vi mạch VDK và các vi mạch và thiết bị phụ trợ.

Tuỳ thành phần cấu tạo và mục đích sử dụng, người ta phân loại kit VDK thành ba loại:

- *Kit loại I:* là loại kit đơn giản nhất, gồm một vi mạch VDK, một tinh thể thạch anh tạo dao động, một mạch RESET, các đèn chỉ thị bằng diot phát quang (LED), cổng trao đổi tin nối tiếp với vi mạch MAX232 và các ổ cắm trao đổi tin song song J0-J3 cho các cổng song song P0-P3.

Sử dụng kit này, chúng ta có thể thực hành được các hành động cơ bản của VDK (chương I tới chương 10), không có ghép nối với thiết bị ngoài nào khác, ngoài thiết bị trao đổi tin nối tiếp với máy vi tính PC hay các đèn chỉ thị LED đơn (diot phát sáng). Nhược điểm của việc sử dụng kit loại này là chương trình (sau khi soạn thảo và dịch) phải được nạp vào bộ nhớ PROM bên trong của VDK bằng bộ nạp (Programmer) thì chương trình mới chạy được. Nếu dùng VDK loại tiên tiến AVR với nhớ chương trình bên trong (PROM) là loại FLASH, ta có thể nạp trực tiếp chương trình từ máy tính PC sang bộ nhớ Flash-chương trình của VDK. Nếu dùng VDK loại tiên tiến có nhớ Flash (ví dụ AVR), có thể nạp chương trình từ PC sang PROM trong của VDK.

- *Kit loại II:* ngoài thành phần của kit loại I, còn bổ sung thêm bàn phím, các khối chỉ thị khác (LED 7 đoạn, ma trận diot, tinh thể lỏng LCD), các vi mạch nhớ ngoài (RAM, ROM) và cổng song song 8255 để có thể nối kit với nhiều thiết bị ngoài khác, trong đó có máy in. Kit loại II này có các thành phần như của một hệ Vi xử lý hay một máy vi tính nên còn gọi là một máy vi tính đơn tấm (SBC – Single board Computer). Kit loại II này có ưu điểm là có thể giao tiếp trực tiếp giữa máy vi tính PC với bộ nhớ RAM ngoài hay các thiết bị ngoài mắc với các cổng (song song hay nối tiếp).

Sử dụng kit này, chúng ta có thể thực hành được các hành động ghép nối của VDK với các thiết bị ngoài (các chương 11-18, tập II), có thể phát triển phần mềm và sử dụng để điều khiển tự động số.

- *Kit loại III:* ngoài thành phần của kit loại II còn bổ sung thêm:

+ *Vi mạch ADC và DAC:* để VDK trao đổi tin tương tự với thiết bị ngoài tương tự.

+ *Thiết bị nạp bộ nhớ ROM:* cho nhớ ROM trong của VDK và ROM ngoài.

Kit loại này được dùng để đo lường với cảm biến (sensor) và điều khiển tự động tương tự theo quy luật PID với các cơ cấu chấp hành (actuator).

Việc phân chia giữa Kit loại II và III chỉ là tương đối.

Trường Đại học Sư phạm kỹ thuật Nam Định đã nghiên cứu và chế tạo kit VDK loại III MGH1 với các bộ phận sau:

- Vi mạch VDK 89C51 với nguồn nuôi, tinh thể thạch anh, nút nhấn RESET, các LED rời nối với cổng song song để hiển thị. Với các bộ phận này, chúng ta có thể lập trình, hợp dịch, liên kết, biến đổi và chạy thử (từng bước và chương trình trên máy PC, ghi vào nhớ ROM và chạy thật trên kit).

- Các thiết bị ghép nối ngoài như bàn phím, LED 7 đoạn, LCD, cổng song song, nhớ ngoài (ROM, RAM), ADC, DAC dùng để ghép nối, mở rộng thành hệ VDK. Với các thiết bị ghép nối mở rộng này, chúng ta có thể ghép nối và lập trình cho VDK trong các hệ đo lường, điều khiển (số, tương tự) và truyền thông trong đó có trao đổi tin với máy tính PC.

1.6. CÁC VDK THUỘC HỌ MCS-51

Ngoài các vi mạch VDK 8051, 8052 thông dụng, người ta đã chế tạo các vi mạch có tính năng tương tự 8051 (bảng 1-1). Có nhiều phiên bản thuộc họ VDK MCS-51 như sau:

- **VĐK 80C31**: có 126 Byte nhớ RAM, không có nhớ ROM bên trong, phải dùng ROM ngoài.
- **VĐK 80C51**: có 126 Byte nhớ RAM, 4kB nhớ ROM chỉ lập trình một lần, không lập trình lại được, có 2 bộ timer0,1.
- **VĐK 80C52**: có 8kB nhớ ROM, 256 Byte nhớ RAM và 3 bộ timer0,1,2 (thêm một timer2).
- **VĐK 87C51**: có 256 Byte nhớ RAM, 8kB nhớ EPROM có thể xóa bằng tia cực tím và ghi lại nhiều lần. Loại có đuôi XX52 có bộ nhớ ROM trong là 8k và RAM trong là 256 Byte và có 2 Timer0,1.
- **VĐK 89C51**: của hãng ATMEL có 256 Byte nhớ RAM, 4kB nhớ FLASH ROM có thể xóa và nạp bằng điện nhiều lần.

Bảng 1-1. Các VĐK thuộc họ MCS-51

Thiết bị	ROM/ EEPROM	RAM	Tốc độ MHz	Chân I/O	Timer/ counter	UART	Ngắt	Loại
Dòng 8051								
8031AH	LESS	128	12	32	2	1	5	HMOS
8051AH	4K	128	12	32	2	1	5	HMOS
8751H/BH	4K EPROM	128	12	32	2	1	5	HMOS
Dòng 8052								
8032	LESS	256	12	32	3	1	6	HMOS
8052AH	8K ROM	256	12	32	3	1	6	HMOS
Dòng 80C51								
80C51BH	4K ROM	128	12,16	32	2	1	5	CMOS
87C51	4K EPROM	128	12,24	32	2	1	5	CMOS
Dòng 83C51								
83C51FA	8K ROM	256	12,16	32	4	1	7	CMOS
83C152	8K EPROM	256	12,24	32	4	1	11	CMOS

Qua bảng 1-1 ta thấy:

- **Loại VĐK không có ROM trong**: không ghi được chương trình vào vi mạch VĐK mà phải dùng bộ nhớ ROM ngoài như 8031, 8031AH, 8032AH, 80C31BH, 80C51FA và 80C152.
- **Loại có đuôi XX51**: có bộ nhớ ROM trong là 4k và RAM trong là 128 Byte và có 2 Timer.
- **Loại có đuôi XX52**: có bộ nhớ ROM trong là 8k và RAM trong là 256 Byte và có 3 Timer0,1, riêng 83C51FA có 4 Timer.
- **Loại 83C51FA**: có các tính năng đặc biệt sau có bộ đếm chương trình hoá kiểu ma trận, có bộ so sánh/bắt giữ, bộ điều chế xung ra và Timer giám sát (Watchdog), kênh nối tiếp chương trình hoá (xem ở phần VĐK tiên tiến).
- **Loại 83C152**: là loại cải tiến của 80C51BH có thêm các tính năng sau:
 - + **Có kênh nối tiếp toàn bộ (GSC – Global Serial Channel)**: trao đổi tin nối tiếp với tốc độ tới 2Mbps theo hai thủ tục HDLC hay CSMA/CD (xem phần các VĐK tiên tiến).

- + Có hai kênh trao đổi tin trực tiếp khỏi nhớ DMA.
- + Có thêm cổng song song P4.
- + Có 11 mức ngắt.

Hiện nay, ngoài VĐK họ MCS-51, người ta còn dùng các họ khác có tính năng tương đương như của các hãng ATMEL (với AT 89C51/52, AT 90C51/52), AVR, Siemen, Analog, v.v...

CÂU HỎI VÀ THỰC HÀNH

I. Câu hỏi

1. Kể các thành phần cấu trúc một vi mạch vi điều khiển.
2. Hãy so sánh cấu trúc một hệ vi xử lý với một vi mạch VĐK.
3. Hãy so sánh các phiên bản của các VĐK thuộc họ MCS-51.
4. Kể các hoạt động cơ bản của một VĐK.
5. Kể các thành phần của một chương trình mô phỏng cho một VĐK.
6. Ta có thể tìm hiểu gì về cấu trúc của một VĐK thông qua một chương trình mô phỏng?
7. Quan sát chương trình mô phỏng SIM51 hay AVSIM, hãy so sánh với chương trình PINNACLE.

II. Thực hành

1. Quan sát và vẽ mạch nối ghép của các chân của vi mạch VĐK trên kit có sẵn tại phòng thí nghiệm.
2. Đo các điện thế của các chân sau khi bật nguồn nuôi và ấn nút RESET.
3. Quan sát dạng tín hiệu ra bằng dao động ký (nếu có) trên các chân của vi mạch VĐK của một kit VĐK có trong phòng thí nghiệm.
4. Gọi và đọc chương trình mô phỏng AVSIM51 trên máy tính PC, đọc qua thư mục Help và quan sát cấu trúc của VĐK trên màn hình.
5. Gọi và đọc chương trình PINNACLE trên máy tính PC, đọc qua thư mục Help (cả ba thành phần).
6. Quan sát trên màn hình thư mục con của VIEW để nhận biết về cấu trúc của VĐK. Ghi lại các kết quả hiện lên màn hình.

Chương 2

LẬP TRÌNH HỢP NGỮ CHO VDK HỌ MCS-51 VỚI PINNACLE

Mục đích của chương:

- Hiểu được yêu cầu và cách dùng hợp ngữ để lập trình cho VDK họ MCS-51.
- Hiểu được trình tự lập trình và các chương trình sử dụng để lập trình một chương trình hợp ngữ cho VDK thuộc họ MCS-51 trong các bước lập giải thuật, soạn thảo, biên dịch, liên kết, chạy thử với chương trình mô phỏng.
- Hiểu được cách viết và sử dụng lệnh gộp và chương trình con.
- Hiểu được cách chạy thử chương trình đã soạn thảo trên chương trình mô phỏng Pinnacle và trên kit VDK.

2.1. TỔNG QUÁT VỀ LẬP TRÌNH HỢP NGỮ CHO VDK

2.1.1. Yêu cầu sử dụng và đặc điểm của hợp ngữ

1. Yêu cầu sử dụng

Giữa người sử dụng và VDK có ngôn ngữ chung dạng mã (mã lệnh), gồm các ~~hệ~~ nhị phân (0, 1). Người dùng đưa lệnh dạng mã này (quy ước bởi nhà thiết kế và chế tạo) cho VDK, VDK thực hiện ngay, không cần thông dịch (dịch và thực hiện ngay từng lệnh) hoặc biên dịch (dịch văn bản của chương trình).

Nhưng mã lệnh này có nhược điểm là:

- Dạng con số nhị phân (BIN) hay 16 (HEX) phức tạp, khó nhớ, khó viết, dễ nhầm lẫn;
- Chương trình dài, khó đọc và khó sửa chữa.

Do đó, để khắc phục nhược điểm trên của lệnh dạng mã, người ta dùng hợp ngữ (ngôn ngữ tập hợp – Assembly) thay cho lệnh dạng mã. Lệnh dạng hợp ngữ là lệnh dạng ký tự (chữ viết) và con số để sao cho dễ viết, dễ nhớ, dễ kiểm tra và dễ sửa chữa. Nhưng nhược điểm cơ bản của hợp ngữ là phải có chương trình dịch hợp ngữ (ASSEMBLER) để dịch các lệnh hay cả chương trình từ dạng hợp ngữ sang dạng mã lệnh.

2. Đặc điểm của hợp ngữ

Hợp ngữ hay ngôn ngữ tập hợp, có các lệnh là tập hợp các chữ (ký tự) (từ A tới Z với chữ thường hay chữ hoa là như nhau) và các chữ số tự nhiên (0, 1, 2, 3 tới 9) và các ký hiệu (symbol) đặc biệt khác.

Hợp ngữ có các đặc điểm sau:

- Mỗi lệnh hay chỉ thị (instruction) có một tên gọi nhớ (mnemonic): tên lệnh này gắn với hành động của lệnh cho dễ nhớ, là chữ viết tắt của hành động bằng tiếng Anh.

Ví dụ: + Lệnh di chuyển dữ liệu từ nguồn S sang đích dữ liệu D là MOV(ement) D, S.

- + Lệnh dịch sang trái một vị trí của thanh ghi tích lũy A là RL A (Rotation left Accumulator).

- Có các lệnh điều khiển (control) và hướng dẫn biên dịch (directive) chương trình: để điều khiển việc dịch chương trình từ dạng hợp ngữ sang dạng mã lệnh. Các điều khiển và hướng dẫn biên dịch này không được dịch thành mã lệnh, không được VĐK thực hiện khi chạy chương trình, mà chỉ ảnh hưởng tới việc dịch chương trình sang dạng mã lệnh.

- Chương trình dạng hợp ngữ này (tập hợp các chữ cái A tới Z và các con số từ 0 tới 9, A, B, C, D, E, F) có ưu điểm là ngắn gọn, dễ nhớ các lệnh nhưng có nhược điểm là phải biên dịch sang chương trình dạng đối tượng (có đuôi .OBJ) bởi chương trình biên dịch (Compiler) và kết nối bởi chương trình kết nối (Linker) các vùng nhớ lệnh và dữ liệu thành một khối nhớ duy nhất gồm các byte dạng .HEX. Chương trình có thể biến đổi từ dạng có đuôi .HEX sang dạng có đuôi .BIN hay ngược lại bởi chương trình biến đổi tương ứng. Hệ vi xử lý hay máy vi tính chỉ chạy được chương trình có đuôi .EXE, .COM hay .BAT, còn VĐK chỉ hiểu và chạy chương trình có đuôi .HEX.

2.1.2. Các thành phần của hợp ngữ

Hợp ngữ gồm các thành phần sau:

- Các ký tự chữ cái: (alphabet) từ A tới Z (chữ hoa và chữ thường như nhau). Một cụm ký tự phải viết liền nhau (ví dụ ABC hay abc) hay nối với nhau bởi dấu gạch ngang dưới (_), ví dụ AB_C.

- Các con số tự nhiên từ 0 tới 9: cho số thập phân. Với số dạng 16 (HEX) thì các chữ phải có chữ số 0 đứng trước. Ví dụ, số 12D là 0CH.

- Các ký hiệu của bàn phím dạng mã ASCII: như \$, ' ', &, % v.v.

- Các toán tử số học: cộng (+), trừ (-), nhân (*) và chia (/).

- Các toán tử logic: AND, OR, XOR, NOT v.v...

- Các toán tử đặc biệt: SHR (dịch phải), dịch trái (SHL), byte cao (HIGH), byte thấp (LOW) v.v...

- Các toán tử quan hệ: EQU(= bằng), NE (<> không bằng), LT (< nhỏ hơn), LE (<= nhỏ hơn hoặc bằng), GT (> lớn hơn), GE (>= lớn hơn hoặc bằng).

Ta có thể dùng các toán tử trên dưới dạng chữ (EQU) hay dấu bằng hoặc ký hiệu (=).

2.1.3. Dạng một lệnh của hợp ngữ

Cũng như lệnh dạng mã, một lệnh của hợp ngữ đầy đủ (cho vi xử lý, máy vi tính hay VĐK) đều có dạng:

[NHAN] : hành động [Đích, Nguồn]; [chú thích]

[Label]: (Operation-opcode) D (Destination), S (Source); Comment

Các chữ trong dấu hai ngoặc vuông là tùy chọn (có thể có hoặc không).

- NHAN (Label): dùng để đánh dấu vị trí hay địa chỉ của lệnh hoặc dữ liệu trong bộ nhớ, để có thể lập lại lệnh hay dữ liệu đó trong chương trình hoặc gọi chương trình con có tên NHAN đó. NHAN này phải bắt đầu bằng một chữ cái, không thể bắt đầu bằng một chữ số thập phân (có thể kết thúc bằng số). NHAN phải viết ở tận cùng ngoài của dòng lệnh và theo sau là dấu hai chấm (:). Trong chương trình, NHAN được dùng ở một hay nhiều lệnh (hoặc dữ liệu) khác và phải ghi chính xác như lúc ký hiệu lần đầu tiên (về chữ hoa hay chữ thường); nếu không, chương trình dịch sẽ báo lỗi vì không tìm thấy để thay thế bằng địa chỉ của lệnh. Sau khi dịch chương trình, trong tệp có đuôi .LST hay .PRN, các nhãn này được thay thế bằng địa chỉ của lệnh trong bộ nhớ. Nếu lệnh (hay dữ liệu) không được gọi lại, lệnh không cần có NHAN đứng trước.

Ví dụ: LOOP: MOV A, #01H

- **HÀNH ĐỘNG (OPCODE)**: là tên hay hành động của lệnh, thường gồm ba hay bốn chữ cái, viết tắt của từ tiếng Anh, chỉ hành động của lệnh, ví dụ lệnh nhảy khi cờ Z bằng 0 là JNZ (Jump if Not Zero). Lệnh NOP được dịch thành mã lệnh 00H sẽ không yêu cầu VDK thực hiện hành động nào ngoài PC+1 và tốn một thời gian một chu trình lệnh (cỡ 1μsec).

Mỗi một vi xử lý hay VDK có một bộ lệnh dạng mã hay dạng hợp ngữ riêng và có một chương trình dịch riêng. Số lệnh dạng hợp ngữ này tùy thuộc số lệnh dạng mã, do người thiết kế vi xử lý hay VDK quy định, thường có tối đa là 256 lệnh vì là bộ mã của một byte (8 bit). Sau khi dịch, chương trình từ tệp nguồn (dạng mã ASCII) sang tệp có đuôi .LST, các mã lệnh đứng trước tên lệnh dạng hợp ngữ. Các hành động hay các lệnh này được nhóm lại theo nhóm theo loại hành động như dịch chuyển dữ liệu, số học (cộng, trừ, nhân, chia), logic (AND, OR, NO, XOR v.v...), điều khiển chương trình (nhảy, gọi chương trình con, nhảy có điều kiện v.v...), xử lý bit.

- **TOÁN HẠNG D, S**: lệnh hay hành động cần dữ liệu hay toán hạng (operand). Có hai loại toán hạng là:

+ **Toán hạng (operand) đích D (Destination)**: nơi gửi kết quả của hành động.

+ **Toán hạng nguồn S (Source)**: nơi lấy dữ liệu cho hành động.

Ví dụ, MOV A, B; gửi nội dung thanh ghi chứa B vào thanh ghi chứa A.

Có lệnh hay hành động không cần toán hạng (như lệnh NOP) hay chỉ cần một toán hạng như lệnh CLR A.

Ví dụ, lệnh xác lập bit SETB P3.3 (xác lập bit 3 của chân hay cổng P3 lên 1) chỉ cần một toán hạng là bit P3.3.

Trong các nhóm lệnh số học (cộng, trừ, nhân, chia) hay logic (AND, OR, XOR v.v...) đều cần hai toán hạng nguồn, do đó, toán hạng đích kiêm luôn cả toán hạng nguồn. Ví dụ ADD A, R0 (cộng thanh ghi chứa A với R0 (nguồn S) rồi gửi kết quả vào thanh ghi A (đích D)).

- **CHÚ THÍCH (comment)**: dùng để ghi chú về lệnh và chương trình, để người đọc lệnh hay chương trình dễ hiểu và dễ theo dõi. Chú thích phải sau dấu chấm phẩy (;), để khi gặp dấu này, chương trình dịch sẽ không dịch và giữ nguyên trong tệp .LST hay .PRN. Với người soạn thảo chương trình có kinh nghiệm, có thể bỏ qua chú thích, không cần ghi chú gì.

2.1.4. Khai báo dữ liệu của lệnh hợp ngữ

Vì toán hạng (D, S) hay dữ liệu của lệnh hợp ngữ có thể là *dữ liệu tức thì* (phương pháp địa chỉ tức thì) đưa từ ngoài vào, *nội dung các thanh ghi R* (phương pháp địa chỉ thanh ghi) hay *nội dung của ô nhớ M* (phương pháp địa chỉ trực tiếp – dữ liệu là địa chỉ ô nhớ hay gián tiếp thanh ghi @R), nên dữ liệu có thể là:

- **Hằng số**: một số có giá trị không đổi với các phép biểu diễn khác nhau (thập phân D, 16 (HEX), dấu phẩy động v.v...).

- **Biến số**: một số có giá trị thay đổi.

- **Ký tự hay số**: dạng mã ASCII.

Dữ liệu trên phải được khai báo trong chương trình, để chương trình dịch:

+ **Nhận biết được NHAN hay hằng số đã sử dụng**: nếu chưa khai báo hay gán giá trị hằng số cho các nhãn này, chương trình sẽ báo lỗi vì không tìm thấy giá trị của NHAN.

+ **Dành chỗ hay địa chỉ ô nhớ**: nếu không có khai báo này, chương trình dịch sẽ không dành chỗ cho các byte nhớ đệm để sẽ ghi giá trị của biến số theo byte (B), lời (W) và nhiều lời (DW, QW, TW).

+ *Dành chỗ hay địa chỉ cho bit nhớ*: nếu không có khai báo này, chương trình dịch sẽ không dành chỗ cho các bit nhớ đệm để sẽ ghi giá trị của biến số là bit.

+ *Ghi giá trị ban đầu của ô nhớ đệm*: nếu không có khai báo này, chương trình dịch sẽ không ghi giá trị ban đầu của biến số vào ô nhớ đệm.

Các khai báo trên tạo thành dữ liệu cho chương trình dưới nhiều dạng khác nhau như byte (B), lời (W), mảng (Array), bản ghi (Record) v.v...

1. Khai báo các hằng số hay nhãn

Các nhãn hay các hằng số này là các giá trị không đổi.

Người ta dùng chỉ dẫn EQU (hay dấu bằng =) nối giữa các hằng cần khai báo với giá trị (không đổi - hằng) của nó. Các giá trị này có thể là:

+ *Giá trị gán cho các ký hiệu hay nhãn dùng trong chương trình*.

Ví dụ, ký hiệu Pi được khai báo như sau: `PI EQU 3,1416`.

Ví dụ, ký hiệu Gio được gán cho ô nhớ 70H được khai báo như sau: `GIO EQU 70H`.

Trong chương trình, người ta dùng nhãn GIO thay cho ô nhớ có địa chỉ 70H như lệnh:

`MOV P0, GIO`; đưa ô nhớ GIO hay 70H ra hiển thị ở cổng P0.

+ *Địa chỉ nhớ gán cho các thanh ghi đặc biệt (SFR) hay thanh ghi dữ liệu đệm*.

Ví dụ: `P0 EQU 80H`; khai báo địa chỉ cho cổng P0.

`LCD_CM EQU 0E00H`; khai báo địa chỉ thanh ghi điều khiển của hiển thị tinh thể lỏng LCD.

Ví dụ: `BIT_DISP EQU 20H`.

`TF2 EQU 0CFH`.

2. Khai báo và dành chỗ các bit nhớ của các bit cờ

; Ví dụ 2-1: Chương trình có khai báo các bit và byte nhớ

<code>org 00h</code>	<code>; địa chỉ sau khi RESET</code>
<code>ljmp MAIN</code>	<code>; nhảy tới MAIN, là nhãn bắt đầu của chương trình chính</code>
<code>org 20h</code>	<code>; địa chỉ bắt đầu khai báo bit nhớ</code>
<code>flag ds 5</code>	<code>; dành 5 bit nhớ</code>
<code>oflag ds 1</code>	<code>; dành 1 bit nhớ</code>
<code>org 30h</code>	<code>; địa chỉ bắt đầu khai báo ô nhớ</code>
<code>kb_data ds 8</code>	<code>; dành 8 ô nhớ</code>
<code>o_data ds 2</code>	<code>; dành 2 ô nhớ</code>
<code>org 00h</code>	<code>; địa chỉ bắt đầu chương trình</code>
<code>MAIN: mov kb_data, #55h</code>	<code>; gửi #55H vào ô nhớ kb_data</code>
<code>setb flag</code>	<code>; xác lập cờ flag</code>
<code>mov a, kb_data</code>	<code>; gửi nội dung ô nhớ kb_data (#55H) vào A</code>
<code>mov p0, a</code>	<code>; đưa #55H ra P0</code>
<code>mov o_data, #0AAh</code>	<code>; gửi #0AAH vào ô nhớ</code>
<code>mov a, o_data</code>	<code>; gửi nội dung ô nhớ o_data (#0AAH) vào A</code>
<code>mov p1, a</code>	<code>; đưa #0AAH ra P1</code>
<code>setb oflag</code>	<code>; xác lập cờ flag</code>
<code>sjmp \$</code>	<code>; chờ tại chỗ</code>
<code>end</code>	<code>; kết thúc chương trình dịch</code>

Người ta dùng DS (Define Storage - định nghĩa tích lũy) để khai báo biến và dành chỗ cho các bit nhớ dành cho các bit cờ (flag) trạng thái, nằm trong vùng nhớ bit có địa chỉ từ 20H tới 2FH hay vùng BIT (BIT Field) của chương trình mô phỏng Pinnacle.

Ví dụ: `ORG 20H` ; bắt đầu từ địa chỉ nhớ 20H, dành cho các bit cờ.

`KEYFLAG DS 1` ; cờ bàn phím, dành 1 bit tại vùng nhớ địa chỉ 20H.

`SENDFLAG DS 1` ; cờ của dữ liệu gửi (truyền) nối tiếp.

Ví dụ 2-1 giới thiệu chương trình có khai báo các bit nhớ và các ô nhớ đệm.

3. Khai báo biến là các ô nhớ cho đệm dữ liệu

Người ta cũng dùng DS để khai báo và dành ô nhớ cho các dữ liệu đệm, giống như các bit cờ trạng thái, nhưng có khác là địa chỉ bắt đầu của vùng nhớ đệm là 30H.

Ví dụ: `ORG 30H`; bắt đầu từ địa chỉ nhớ 30H, dành cho các dữ liệu đệm.

`KB_DATA DS 1`; đệm bàn phím, có một ô nhớ.

4. Khai báo giá trị ban đầu của các ô nhớ dành cho đệm dữ liệu

Người ta dùng DB (Define Byte - xác định Byte), DW (Define Word) DD (Define Double Word), DT (Define Tetra Word) để định nghĩa và dành chỗ cho byte (8 bit), lời (16 bit), 32 bit và 10 byte để nạp giá trị 0 hay giá trị biến sau chữ DB, DW, DD, TD vào các ô nhớ.

Ví dụ: `KB_DATA DB 0` ; đệm bàn phím, giá trị đầu là 0

`RX_DATA DB 0` ; đệm dữ liệu nhận, giá trị ban đầu là 0

`MYDATA DB 'HANOI'` ; đệm dữ liệu với chuỗi ký tự 'HANOI'

Khai báo nhãn MYDATA và DB để gán giá trị ban đầu là chuỗi ký tự 'HANOI' cho bảng dữ liệu và được lệnh đọc bảng dữ liệu (nghiên cứu chi tiết ở chương 3).

2.1.5. Hướng dẫn và điều khiển biên dịch (directive and control) của Pinnacle

Ngoài lệnh, chương trình dạng hợp ngữ còn có chỉ dẫn (directive) và điều khiển (control) biên dịch. Các chỉ dẫn và điều khiển biên dịch chỉ có tác dụng trong quá trình biên dịch, không được dịch thành lệnh dạng mã của chương trình. Trừ khai báo dữ liệu tác động đến phân bố bộ nhớ cho chương trình, các hướng dẫn và điều khiển biên dịch này không có tác động gì tới chương trình mà chỉ tác động tới sự biên dịch chương trình.

Tuỳ loại chương trình hợp dịch, các chỉ dẫn và điều khiển biên dịch này có dạng khác nhau, nhưng không khác biệt nhau nhiều. Ở đây, chúng ta trình bày điều khiển và chỉ dẫn biên dịch của chương trình hợp dịch Pinnacle.

1. Các chỉ dẫn biên dịch

Các chỉ dẫn biên dịch được dùng để xác định các ký hiệu, dành các vùng nhớ, lấy ra các giá trị trong bộ nhớ chương trình, chọn các không gian nhớ khác nhau, xác lập đoạn hiện hành của bộ đếm vị trí và kết thúc của tệp chương trình nguồn.

Các chỉ dẫn biên dịch của VĐK 8051 cho chương trình dịch PINNACLE gồm:

- `ORG` : kèm theo sau là giá trị của ô nhớ bắt đầu của chương trình hay vùng dữ liệu.
- `END` : kết thúc chương trình dịch.
- Dành trước và ghi giá trị ban đầu của các ô nhớ (khai báo dữ liệu): `DB` (byte-8 bit), `DW` (lời-16 bit), `DS` (vùng nhớ nhiều byte).
- `BIT`: gán ký hiệu bit.
- Khai báo biến dùng cho nhiều chương trình (để liên kết các chương trình): `PUBLIC` (chung), `EXTRN` (ngoài), `PROGRAM` (chương trình), `NAME` (tên).

- SET: xác lập hay gán giá trị.
- PINNACLE: ký hiệu tiền định nghĩa.

2. Các điều khiển biên dịch

Các điều khiển biên dịch này dùng để điều khiển cho mã chương trình hợp dịch chéo (CROSS ASSEMBLER) lấy tệp chương trình nguồn vào (đuôi .ASM), chỗ lấy tệp đối tượng (đuôi .OBJ), tạo thành và đưa ra danh sách lệnh (đuôi .LST).

Chúng ta có thể đọc các điều khiển và chỉ dẫn này trong thực đơn Trợ giúp (Help) của Pinnacle và sẽ lần lượt sử dụng chúng trong các ví dụ ở trong chương này. Các chỉ dẫn biên dịch này như:

- IF... THEN... ELSE... ENDIF: hướng dẫn biên dịch có điều kiện.
- IFDF, IFNDF: hướng dẫn biên dịch xác định dùng nhãn.
- INCLUDE: hướng dẫn biên dịch tệp thư viện chương trình của hợp ngữ khi có lệnh gọi một hay nhiều tệp của thư viện chương trình.
- LIBRARY: hướng dẫn biên dịch dùng thư viện chương trình.
- MACRO ...ENDMAC: hướng dẫn biên dịch nhóm lệnh thành một lệnh gộp có tham số sẽ đưa vào khi gọi lệnh gộp.

2.2. SOẠN THẢO MỘT CHƯƠNG TRÌNH HỢP NGỮ CHO VĐK 8051/8052

2.2.1. Quá trình lập trình một chương trình hợp ngữ

Sau khi xác định nhiệm vụ và vẽ giải thuật của chương trình, chúng ta phải soạn thảo một văn bản của chương trình có khuôn dạng như ở phần 2.2.4, rồi hợp dịch bởi chương trình biên dịch và liên kết nó bởi chương trình liên kết. Quá trình soạn thảo, hợp dịch, liên kết và biến đổi một chương trình hợp ngữ với các chương trình hỗ trợ và cho các tệp như hình 2-1.

Vì có nhiều loại chương trình dịch và mô phỏng, nên cũng có nhiều loại hợp dịch và liên kết. Ở đây, chúng ta lại xét chương trình hợp dịch và liên kết với ASM51 (trong môi trường MSDOS) trước vì nó mô tả chi tiết quá trình dịch và liên kết, rồi tới Pinnacle trong môi trường Windows vì thông dụng. Với chương trình hợp dịch AVMAC51.EXE, chúng ta có thể tự làm lấy.

Bảng 2-1. Các tệp sau khi soạn thảo, hợp dịch, liên kết và biến đổi

Quá trình Chương trình	Soạn thảo	Hợp dịch	Liên kết	Biến đổi
ASM51.EXE	dùng EDIT.COM cho .ASM	dùng ASM51.EXE cho .OBJ, và .LST	dùng RL.EXE cho .MAP và .HEX	
PINNACLE	dùng Pinnacle cho .ASM	dùng PINNACLE cho .OBJ và .LST	dùng PINNACLE cho .MAP và .HEX	
AVMAC51.EXE	dùng EDIT.COM cho .ASM	dùng AVMAC51. cho .OBJ và .PRN	dùng AVLINK. cho .MAP và .HEX	dùng HEXTOCSM cho .HEX doc

Sau mỗi lần soạn thảo, hợp dịch, liên kết và biến đổi, chúng ta cần các chương trình vào và cho các chương trình ra như ở bảng 2-1. Từ bảng 2-1, chúng ta thấy có các đặc điểm sau:

Sau soạn thảo, đều có chương trình dạng mã ASCII với đuôi .ASM để đưa vào chương trình hợp dịch.

Sau hợp dịch, đều có tệp với các đuôi .OBJ (đối tượng) để đưa vào chương trình liên kết và cho tệp có đuôi .LST (cho Pinnacle hay ASM51.EXE) hay .PRN (cho AVMAC51.EXE) (xem ở dưới).

Có thể liên kết nhiều mô đun có đuôi .OBJ và dịch thành một chương trình có đuôi EXE hay .HEX.

Vì điều khiển chạy chương trình với đuôi .HEX.

Trong một số trường hợp, phải biến đổi tệp đuôi .HEX thành đuôi .BIN để tin truyền nối tiếp bằng các chương trình tương ứng. Với chương trình Pinnacle, có sẵn các chương trình biến đổi sang .HEX hay sang .BIN, chỉ cần nháy chuột vào chúng, sau khi đã gọi chương trình cần biến đổi.

Vì có nhiều loại chương trình dịch và mô phỏng, nên cũng có nhiều loại hợp dịch và liên kết. Ở đây, chúng ta lại xét chương trình hợp dịch và liên kết AVSIM trước và Pinnacle sau.

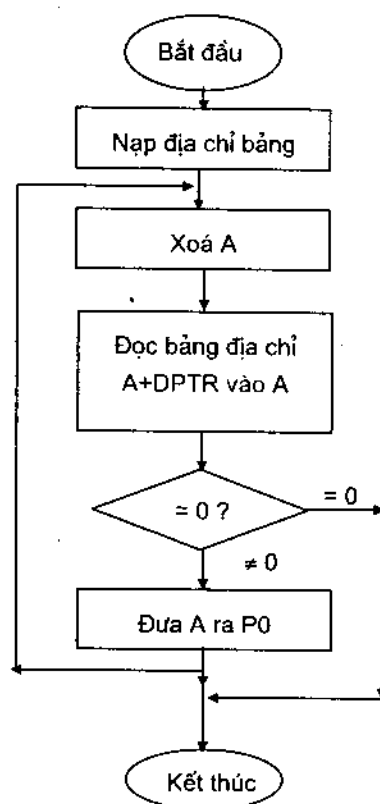
2.2.2. Vẽ giải thuật và lưu đồ

Giải thuật (Algorithm) là các bước lập luận của vấn đề hay bài toán của chương trình. Nói khác đi, giải thuật chính là cách thức giải bài toán hay cách lập luận để giải hoặc các khả năng của nghiệm số của bài toán. Giải thuật do nhiệm vụ của bài toán hay chương trình đặt ra. Có hai loại giải thuật:

- *Giải thuật đơn giản*: các bước hay tiến trình của bài toán diễn ra một cách độc lập với nhau, không có sự phụ thuộc lẫn nhau giữa các bước hay các nhiệm vụ nhỏ. Chương trình của giải thuật loại này không có các phép thử hay kiểm tra để quyết định có rẽ nhánh hay không rẽ nhánh sang các nhiệm vụ khác. Nói khác đi, chương trình của giải thuật này tiến hành theo thứ tự nối tiếp các nhiệm vụ nhỏ, các lệnh của chương trình được thực hiện nối tiếp từ trên xuống dưới (top-down), không có các lệnh so sánh và rẽ nhánh có hay không có điều kiện.

- *Giải thuật phức tạp*: các bước hay tiến trình của bài toán diễn ra một cách không độc lập, có sự phụ thuộc lẫn nhau giữa các bước hay nhiệm vụ nhỏ, tùy giá trị kết quả của các thông số trong các bước nhỏ ở trên. Chương trình của giải thuật này có các phép thử hay kiểm tra để quyết định có rẽ nhánh hay không sang các nhiệm vụ khác. Nói khác đi, chương trình của giải thuật này tiến hành không theo thứ tự nối tiếp các nhiệm vụ nhỏ, các lệnh không được thực hiện theo thứ tự từ trên xuống dưới (top-down), mà có các lệnh so sánh và rẽ nhánh có hay không có điều kiện. Trong loại giải thuật này, một số lệnh của chương trình có thể được thực hiện nhiều lần (lặp lại), một số lệnh có thể bỏ qua, không được thực hiện khi chạy chương trình.

Việc đầu tiên của lập trình là vẽ giải thuật, tức vẽ trình tự các nhiệm vụ, trong đó có các khâu quyết định (decision) để xem xét các điều kiện đặt ra, các điều kiện của các nhiệm vụ trước đó, để quyết định bước tiếp theo hay rẽ nhánh sang các nhiệm vụ khác tùy theo điều kiện của kết quả của lệnh của chương trình trước đó (các điều kiện và các lệnh rẽ nhánh). Nói khác đi, giải thuật là một hình vẽ với các hình bầu dục chỉ bắt đầu (ORG) và kết thúc (END) chương trình với các tiến trình (hình chữ nhật) và các so sánh để quyết định (hình thoi) với các nhánh với mũi tên trở tới các lệnh khác nhau cho các "tình huống" khác nhau. Hình 2-1 là giải thuật hay lưu đồ của chương trình của ví dụ 2-4 (trang 31).



Hình 2-1. Giải thuật của ví dụ 2-4

Sau khi vẽ giải thuật hay cấu trúc và lập luận của bài toán đặt ra của chương trình, người ta vẽ lưu đồ (flow chart) hay chiều diễn biến của chương trình. Lưu đồ này có cốt lõi là giải thuật nhưng bổ sung thêm các chi tiết, các hành động cụ thể dưới dạng sơ đồ khối các hành động. Nếu một chương trình dài của một bài toán có nhiều "tình huống" xảy ra, có một lưu đồ phức tạp với các điểm nối (connection) có đánh số có cùng một chỉ số để tránh vẽ nhiều đường thẳng nối các điểm nối các khối ở nhiều vị trí, nhiều trang giấy. Các điểm nối (connection) đánh số có cùng một chỉ số sẽ được nối với nhau bởi một đường thẳng, chỉ sự phát triển hay liên quan của chương trình.

Chương trình là những lệnh của các hành động của lưu đồ. Ví dụ 2-4 chính là chương trình của lưu đồ hay giải thuật trên hình 2-1.

2.2.3. Thiết kế chương trình

Sau khi vẽ lưu đồ và trước khi soạn thảo hay viết chương trình, chúng ta phải thiết kế chương trình. Thiết kế chương trình là căn cứ vào lưu đồ, chúng ta phải xác định:

- *Các lệnh gộp*: các hành động nhỏ được lặp lại trong chương trình cần các tham số khác nhau được viết dưới dạng lệnh gộp (xem mục 2-6).

- *Các chương trình con* (Subroutine): Trong ngôn ngữ bậc cao, các chương trình con này được gọi là các hàm (Function), các thủ tục (Procedure), chúng là một nhóm lệnh thực hiện các hành động lớn hơn lệnh gộp, được lặp lại trong chương trình, không cần các tham số khác nhau (xem mục 2.6). Ta cần chú ý bảo vệ các biến và các giá trị trả về của các biến khi gọi và kết thúc chương trình con. Các biến được dùng trong các chương trình con này có thể dùng riêng cho từng chương trình con (biến cục bộ với khai báo LOCAL) hay biến dùng chung cho mọi chương trình con (với khai báo PUBLIC) hay đã khai báo ở chương trình con khác (với khai báo EXTERN).

- *Chương trình chính*: chương trình chính này là bản danh sách lệnh, các lệnh gộp, các chương trình con. Tất cả các thành phần trên thực hiện các nhiệm vụ của sơ đồ khối của chương trình tức lưu đồ của chương trình. Trong chương trình chính cũng có các khai báo hằng và biến số cho các chương trình con.

2.2.4. Soạn thảo chương trình hợp ngữ

1. Cách soạn thảo một chương trình

Soạn thảo một văn bản của chương trình hay viết chương trình nguồn trên PC với đuôi .ASM, tức danh sách các lệnh dạng hợp ngữ của chương trình, ta phải:

- *Chọn môi trường soạn thảo*: có thể chọn môi trường MSDOS hay Windows.

Với môi trường MSDOS, chúng ta đưa máy tính PC về chế độ MSDOS với dấu nhắc C:> edit (ENTER). Với môi trường Windows chúng ta có thể nháy chuột vào NOTPAD hay PINNACLE với thực đơn NEW.

Các tệp của chương trình soạn thảo trong PINNACLE được tự động đặt trong thư mục MYDOCUMENT.

- *Chọn chương trình soạn thảo*: trong môi trường MSDOS, người ta dùng một trong các chương trình EDIT.EXE của MSDOS, NCEDIT.EXE hay NOTPAD.EXE của WINDOWS để tạo một tệp (file) văn bản mới bằng cách mở một tệp mới. Với chương trình mô phỏng Pinnacle, ta vào giao diện của Pinnacle rồi chọn thực đơn File\New.

- *Gõ các ký tự của văn bản chương trình*: từ bàn phím, gõ các ký tự của văn bản chương trình và với các phím lệnh soạn thảo và ký tự, chúng ta có một văn bản của chương trình.

- *Đặt tên tệp*: với lệnh SAVE AS, gõ tên tệp (có đuôi.ASM mặc định) và gõ Enter.

- *Đọc và sửa lại tệp cũ:* bằng lệnh OPEN và đuôi tương ứng (ALL files) hay .ASM có thể mở lại tệp đã có, sửa chữa lỗi của văn bản, gõ SAVE và Enter.

Với chương trình mô phỏng Pinnacle trong môi trường Windows, chúng ta có thể soạn thảo trực tiếp trên màn hình, sau khi nháy chuột vào thực đơn FILE \NEW giống như với Edit trong MSDOS.

2. Dạng chương trình cho chương trình hợp dịch Pinnacle

Một chương trình hợp ngữ phải có dạng quy ước theo chương trình hợp dịch, để chương trình dịch đúng, không báo lỗi. Với VDK họ MCS-51 có chương trình hợp dịch trong môi trường MSDOS (ASM51.EXE, AVMAC51.EXE) và trong môi trường Windows (Pinnacle). Hiện nay thông dụng là chương trình Pinnacle vì đơn giản, dễ sử dụng nên chúng ta xét trước, còn ASM51 sẽ xét sau, chỉ để tham khảo vì nó phức tạp và dài dòng hơn trong khai báo.

Pinnacle có các hướng dẫn biên dịch sau:

- ORG 00H: chỉ bắt đầu chương trình trong vùng nhớ, tại địa chỉ 00H.
- SJMP \$: ở chỗ kết thúc các chương trình chính, chỉ chờ tại chỗ, nếu không có lệnh này, khi chạy chương trình với Pinnacle sẽ báo lỗi (vì thực hiện quá 10 lệnh NOP sau chương trình).
- END : ở chỗ kết thúc các chương trình con, chỉ kết thúc chương trình dịch.

Có hai loại : dạng không có chương trình phục vụ ngắt và có chương trình phục vụ ngắt.

a) Dạng chương trình không có chương trình phục vụ ngắt và khai báo dữ liệu

Đây là dạng đơn giản và ngắn gọn nhất, vì không có chương trình phục vụ ngắt, không có khai báo dữ liệu, chương trình có dạng như hình 2-2. Chương trình dạng này, chỉ có các hướng dẫn biên dịch bắt đầu, chờ tại chỗ và kết thúc chương trình dịch.

Ví dụ 2-2 giới thiệu chương trình đưa dữ liệu ra cổng P₀, P₁.

ORG 00H	; khai báo bắt đầu vùng nhớ chương trình, PC = 00H
	; phần các mã lệnh của chương trình chính
	; phần các chương trình con với lệnh kết thúc RET
SJMP \$	; kết thúc chờ tại chỗ
END	; kết thúc chương trình dịch

Hình 2-2. Dạng chương trình không có ngắt và khai báo dữ liệu

; Ví dụ 2-2: Chương trình đưa dữ liệu ra cổng	
ORG 00H	; bắt đầu chương trình bằng địa chỉ nhớ 00h
MOV A, # 55h	; gán thanh ghi A bằng giá trị tức thì 55H
MOV P0, A	; đưa ra cổng P0
MOV A, #0AAH	; gán thanh ghi A bằng giá trị tức thì 0AAH
MOV P1, A	; đưa ra cổng P0
MOV A, #0FFH	; gán thanh ghi A bằng giá trị tức thì 0FFH
MOV P2, A	; đưa ra cổng P0
SJMP \$	; kết thúc, chờ tại chỗ
END	; kết thúc chương trình

b) Dạng chương trình không có chương trình phục vụ ngắt và có khai báo dữ liệu

Một chương trình loại này, ngoài khai báo bắt đầu và kết thúc chương trình (hình 2-3), còn có:

- Khai báo hằng số với EQU;
- Khai báo các bit cờ đệm;
- Các byte nhớ đệm với DS, DW, DQ, DT;
- Các giá trị ban đầu của ô nhớ đệm với DB.

Ví dụ 2-3 giới thiệu chương trình có khai báo các bit nhớ và các ô nhớ đệm.

```
org 00h
Ljmp Main
ORG 20H          ; khai báo các hằng số với EQU
                 ; khai báo các bit cờ Flag
ORG 30H
                 ; khai báo các ô nhớ đệm cho dữ liệu
ORG 100H         ; khai báo bắt đầu vùng nhớ chương trình, PC = 00H
Main:            ; phần các mã lệnh
                 ; phần các chương trình con với lệnh kết thúc RET
SJMP $           ; kết thúc chờ tại chỗ
END              ; kết thúc chương trình dịch
```

Hình 2-3. Dạng chương trình không ngắt và có khai báo dữ liệu

; Ví dụ 2-3: Chương trình có khai báo các bit cờ và byte nhớ

```
org 00h
Limp Main
org 20h          ; địa chỉ bắt đầu khai báo bit nhớ
flag ds 5        ; dành 5 bit nhớ
oflag ds 1       ; dành 1 bit nhớ
org 30h          ; địa chỉ bắt đầu khai báo ô nhớ
kb_data ds 8     ; dành 8 ô nhớ
o_data ds 2      ; dành 2 ô nhớ
org 100h         ; địa chỉ bắt đầu chương trình
Main: mov kb_data, #55h ; gửi #55H vào ô nhớ kb_data
      setb flag         ; xác lập cờ flag
      mov a, kb_data    ; gửi nội dung ô nhớ kb_data (#55H) vào A
      mov p0, a         ; đưa #55H ra P0
      mov o_data, #0AAh ; gửi #0AAH vào ô nhớ o_data
      mov a, o_data     ; gửi nội dung ô nhớ o_data (#0AAH) vào A
      mov p1, a         ; đưa #0AAH ra P1
      setb oflag        ; xác lập cờ oflag
      sjmp $           ; chờ tại chỗ
end                ; kết thúc dịch chương trình
```

Ví dụ 2-4 dùng khai báo nhãn MYDATA và DB để gán giá trị ban đầu là chuỗi ký tự 'HANOI' và dùng lệnh đọc bảng dữ liệu có giải thuật trên hình 2-1.

Ví dụ 2-4: Chương trình đọc bảng dữ liệu trong PROM trong

ORG 00H	
MOV DPTR, #MYDATA	; nạp địa chỉ đầu tiên của bảng
BACK: CLR A	; xoá A = 0
MOVC A, @A+DPTR	; đọc bảng
JZ KT	; nhảy tới KT nếu gặp 0 của chuỗi
MOV P0, A	; đọc bảng ra cổng P0
INC DPTR	; tăng địa chỉ bảng
SJMP BACK	; trở về Back để đọc lại
KT: SJMP KT	; kết thúc chờ
ORG 200H	; địa chỉ bắt đầu của bảng chữ "HA NOI"
MYDATA: DB 'HA NOI', 0	; dữ liệu của bảng, kết thúc là 0
END	; kết thúc dịch chương trình

3. Dạng chương trình có chương trình phục vụ ngắt và khai báo dữ liệu

Đây là dạng chương trình đầy đủ nhất, theo thứ tự từ trên xuống dưới, phải có:

- Khai báo các hằng và biến (gán nhãn hay địa chỉ, bit cờ, đệm dữ liệu);
- Khai báo các địa chỉ dành cho các ngắt (xem chương 3 và 10);
- Chương trình chính;
- Các chương trình con.

Hình 2-4 giới thiệu dạng chương trình có khai báo đầy đủ cả hằng, biến và các vùng nhớ dành cho các vectơ ngắt.

Chú ý: Trên đây là dạng chương trình khai báo đầy đủ cho các vectơ ngắt. Một số vectơ ngắt mà chương trình không sử dụng, chỉ có nhãn để chương trình nhảy đến và lệnh RETI để trở về từ ngắt. Trong thực tế, có thể bỏ các ngắt này, không cần khai báo vì không sử dụng.

Người ta hay dùng dạng chương trình cho chương trình dịch và mô phỏng Pinnacle vì:

- Có bắt đầu vùng nhớ bởi ORG và kết thúc chương trình dịch bởi END.
- Kê khai đơn giản hơn (chỉ cần ORG cho một đoạn nhớ bất kỳ).
- Có vùng bit cờ và byte nhớ dữ liệu đều kê khai với DS 1.
- Kê khai byte nhớ đệm dữ liệu với DB0H, không phải là DS0H như trong Pinnacle.

4. Dạng chương trình cho chương trình hợp dịch ASM51.EXE

Chương trình hợp dịch ASM51.EXE có dạng giống chương trình Pinnacle về bắt đầu (ORG), kết thúc chương trình dịch (END) và các khai báo hằng, biến, nhưng khác là có khai báo cả các thanh ghi đặc biệt (SFR) và các bit trạng thái. Chương trình mô phỏng AVSIM51 để lập trình cho chương trình hợp dịch ASM51.EXE như hình 2-5. Chương trình này có giao diện đen trắng. Nhưng ở trong môi trường MSDOS, nên đòi hỏi phải gõ phím khi gọi lệnh, nên sử dụng không thuận tiện bằng Pinnacle.

5. Dạng chương trình cho chương trình hợp dịch AVMAC51.EXE

Dạng chương trình cho chương trình hợp dịch AVMAC51 có những đặc điểm sau:

- Giống dạng chương trình dịch và mô phỏng Pinnacle là trình tự bố trí là kê khai hằng, biến, các vectơ ngắt, chương trình khởi phát, chương trình chính, các chương trình phục vụ ngắt và các chương trình con.

- Khác với dạng chương trình Pinnacle là kê khai các đoạn nhớ phức tạp hơn (xem ví dụ 2-5).

```

LCD_CM      EQU 0E000H
LCD_BSF     EQU 0E001H
LCD_WR      EQU 0E002H
LCD_RD      EQU 0E003H
DD_RAM_ADDR EQU 80H
LCD_INIT    EQU 38H
; vùng cờ RAM bên trong từ 20H
    org 20h
KEYFLAG     DS    1
CMPFLAG     DS    1
SENDFLAG    DS    1
RECEIFLAG   DS    1
EOFFLAG     DS    1
; vùng nhớ dữ liệu RAM bên trong từ 30H
    org 30h
KB_DATA     DS 0
RX_DATA     DS 0
TX_DATA     DS 0

ORG 00H      ; ấn RESET
JMP MAIN

ORG 03H      ; ngắt ngoài 0 cho bàn phím
JMP EXINTER0

ORG 13H      ; ngắt ngoài 1
JMP EXINTER 1

ORG 0BH      ; ngắt Timer0
JMP COUNTER0

ORG 1BH      ; ngắt Timer1
JMP COUNTER1

ORG 2BH      ; ngắt Timer2
JMP COUNTER2

ORG 23H      ; ngắt cổng nối tiếp
JMP SERIAL

```

Hình 2-4. Chương trình dạng có ngắt và khai báo dữ liệu (còn tiếp)

(tiếp theo)	
ORG 100H	; bắt đầu chương trình chính
MAIN:	; khởi phát IRQ0 cho bàn phím
;	Khởi phát LCD
;	-----
;	Khởi phát SP
;	-----
;	Các chương trình phục vụ các vectơ ngắt;
;	-----
EXINTER0:	
NOP	; chương trình phục vụ ngắt bàn phím
RETI	; trở về chỗ chương trình bị ngắt
EXINTER1:	; chương trình cho ngắt ngoài 2
NOP	
RETI	; trở về chỗ chương trình bị ngắt
COUNTER0:	; chương trình cho ngắt bộ đếm 0
NOP	
RETI	; trở về chỗ chương trình bị ngắt
COUNTER1:	; chương trình cho ngắt bộ đếm 1
NOP	
RETI	; trở về chỗ chương trình bị ngắt
COUNTER2:	; chương trình ngắt bộ đếm 2
NOP	
RETI	; trở về chỗ chương trình bị ngắt
SERIAL:	; chương trình phục vụ ngắt nối tiếp
NOP	
RETI	; trở về chỗ chương trình bị ngắt
;	-----
;	Các chương trình con
;	-----
SJMP \$	
END	; kết thúc chương trình dịch

Hình 2-4. Chương trình dạng có ngắt và khai báo dữ liệu (phần cuối)

Chú ý: - Chương trình trên là dạng cho chương trình hợp dịch AVMAC51.EXE với khai báo các đoạn nhớ phức tạp hơn trong Pinnacle.

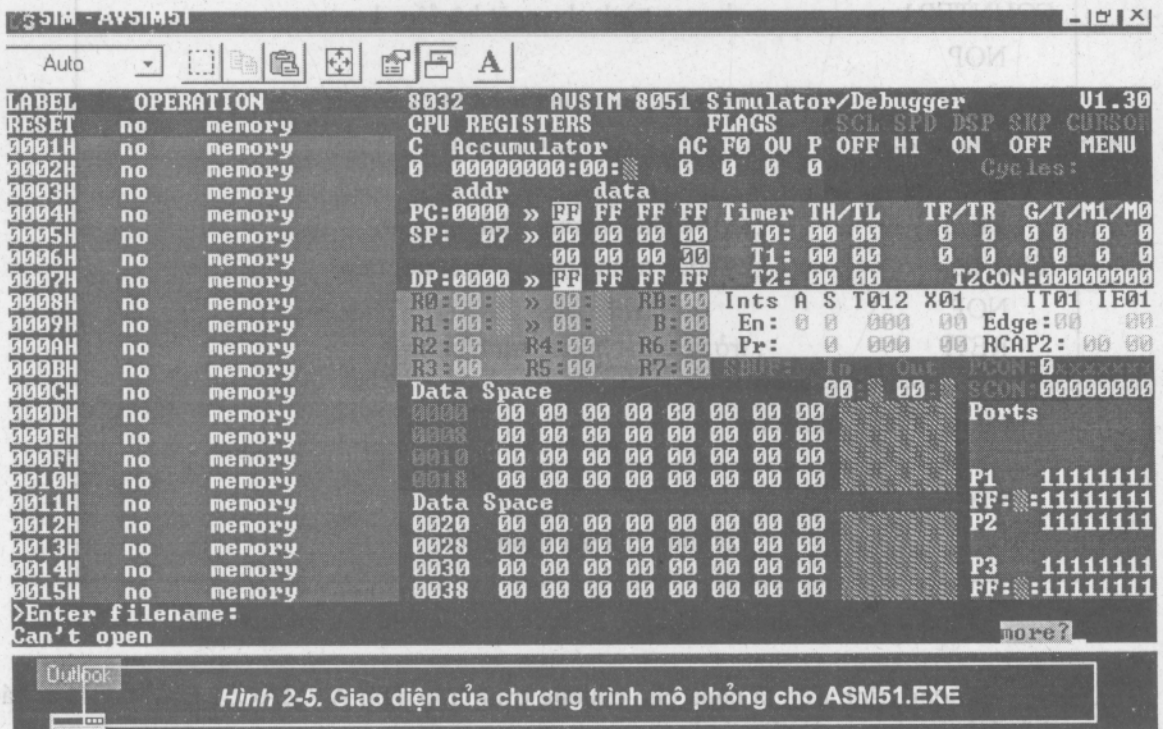
- Có thể đổi chương trình trên thành dạng chương trình cho chương trình hợp dịch Pinnacle nếu thay các đoạn nhớ bằng ORG 00H và ORG 4030H bằng cách bỏ dấu chấm phẩy (;) trước chúng và thêm dấu chấm phẩy trước các đoạn nhớ.

; Ví dụ 2-5: Chương trình bật tắt cổng P1 cho chương trình trong AVSIM51.EXE (bỏ dấu ;)

```
; P1 equ 90H ; gán địa chỉ cổng P1 cho Pinnacle và ASM51.EXE
DEFSEGP_START, START = 00H ; kê khai đoạn nhớ khi bật nguồn nuôi hay ấn RESET
SEGP_START
; ORG 00H ; bắt đầu chương trình cho Pinnacle và ASM51.EXE
JMP MAIN ; nhảy tới chương trình chính, có nhãn MAIN
DEFSEGMAINSEG, START = 4030H ; kê khai vùng nhớ chứa chương trình chính
SEGMAINSEG
; ORG 4030H ; bắt đầu chương trình chính cho Pinnacle và ASM51.EXE
MAIN: MOV P1, #0FFH ; đưa các bit 1 ra cổng P1
MOV P1, #00H ; đưa các bit 0 ra cổng P1
JMP MAIN ; trở về bắt đầu chương trình
END ; kết thúc chương trình dịch
```

- Có thể đổi thành dạng chương trình cho chương trình hợp dịch ASM51 nếu thay các đoạn nhớ bằng ORG 00H và ORG 4030H và kê khai cổng P1 EQU 90H.

- Khi cho chạy chương trình trên trên kit vi điều khiển, sau khi hợp dịch và liên kết, phải bổ sung các chương trình con TRE độ 1sec (xem chương 6) để mắt thường có thể thấy được sự sáng hay tắt các đèn chỉ thị mắc vào cổng P1.



Hình 2-5. Giao diện của chương trình mô phỏng cho ASM51.EXE

- Do khai báo các đoạn nhớ phức tạp hơn trong chương trình Pinnacle, nên ngày nay, người ta dùng chương trình Pinnacle vì hai lý do là khai báo đoạn nhớ đơn giản và dùng trong môi trường WINDOWS.

2.3. HỢP DỊCH VÀ LIÊN KẾT CHƯƠNG TRÌNH NGUỒN VỚI CHƯƠNG TRÌNH ASM51.EXE

2.3.1. Hợp dịch chương trình nguồn

Sau khi soạn thảo tệp.ASM (chương trình nguồn - Source), ta hợp dịch nó với chương trình hợp dịch ASM51.EXE theo cú pháp sau:

C:> ASM51.EXE vi_du2_1.ASM

Với chương trình ASM51.EXE và chương trình soạn thảo cùng trong ổ đĩa C. Nếu không cùng ổ đĩa, phải có đường dẫn tới tên tệp của chương trình.

Ví dụ: C:\ASM51 vi_du2_1.ASM <Enter>

Sau khi ấn Enter, trên màn hình hiện lên các ký tự sau:

8051 CROSS ASSEMBLER

VERSION 1.2

(C) copyright 1984, 1985, 1986, 1987, 1988, 1989, 1990

MetaLink Coporation

Source file drive and name [.ASM] : sample <CR>

First pass

Second pass

ASSEMBLY COMPLETE, 0 ERRORS FOUND

Điều đó có nghĩa là:

- Chương trình hợp dịch 8051 CROSS ASSEMBLER phiên bản 1.2 của công ty MetaLink.
- Dịch tệp nguồn có tên name.ASM theo hai bước : bước một (thống kê các nhãn và ký hiệu) và bước hai (hợp dịch thành các mã lệnh và diễn các nhãn bằng các địa chỉ các ô nhớ tương ứng trong chương trình) và việc dịch xong, không có lỗi.

Nếu việc dịch phát hiện lỗi về cú pháp (tên lệnh, tên nhãn không đúng), màn hình sẽ hiện lên số lỗi, dòng có lỗi và mã của lỗi.

Tra cứu mã của lỗi trong tài liệu cung cấp, ta sẽ biết được lỗi để sửa. Như vậy, sau khi hợp dịch, ta có hai tệp:

- + Tệp đối tượng (đuôi .OBJ).
- + Tệp danh sách (có đuôi .LST hay .PRN): tệp có địa chỉ ô nhớ, mã lệnh, số thứ tự dòng, và lệnh dạng hợp ngữ (hình 2-6 cho ví dụ 2-1.ASM). Xem tệp .LST này, ta có thể biết được lỗi ở dòng nào, điều này giúp ta sửa chương trình một cách dễ dàng.

2.3.2. Liên kết nhiều tệp. OBJ thành một tệp duy nhất

Nếu có nhiều tệp .OBJ hay chỉ một tệp .OBJ, cần liên kết và xếp đặt lại (Relocatable) các ký hiệu trong bản đồ nhớ bởi chương trình RL51.exe với cú pháp sau:

C:> RL51 vi_du2_1.OBJ <Enter>

Trong đó RL51.EXE là chương trình kết nối một hay nhiều tệp chương trình dịch có đuôi .OBJ.

Sau khi liên kết, ta được tệp với đuôi .MAP (hình 2-7) và đuôi HEX theo hàng ngang (hình 2-8 cho ví dụ 2-2). Tệp .MAP là một bản đồ nhớ của chương trình (đuôi .MAP) tức sự bố trí của chương trình trong vùng nhớ.

File	Edit	Search	View	Options	Help
+-----C:\avs\VD2-2.LST-----+					
; VD2-2.LST _					
	1	; Ví dụ 2-2: Chương trình đưa dữ liệu ra cổng P0, P1, P2			
0080	2	p0 equ 80h		_	
0080	3	p1 equ 80h		_	
00A0	4	p2 equ 0a0h		_	
	5			_	
0000	6	ORG 00h		_	
	7			_	
0000 020100	8	jmp MAIN		_	
0100	9	ORG100H		_	
0100	10	MAIN:		_	
	11			_	
0100 7455 12		MOV A,#55h ; gửi #55H vào thanh ghi A			
0102 F580 13		MOV P0,A ; gửi A ra cổng P0			
0104 74AA	14	MOV A,#0AAH; gửi #AAH vào thanh ghi A			
0106 F580 15		MOV P1,A ; gửi A ra cổng P1			
0108 74FF 16		MOV A,#0FFH; gửi #FFH vào thanh ghi A			
010A F5A0	17	MOV P2,A ; gửi A ra cổng P2			
010C 80FE	18	SJMP \$; nhảy tại chỗ			
	19	END ; kết thúc chương trình dịch			
F1=Help	Line:1	Col:1			

Hình 2-6. Tệp đuôi .LST của ví dụ 2-2.ASM

```
PINNACLE 52 LINKER MAP FILE
Output File: F:\PINNACLE2-1.HEX
RELOCATABLE SEGMENTS:
SEGMENTSTARTENDTOTAL
TYPENAMEADDRADDRLLENGTH
-----
```

```
PUBLIC SYMBOLS:
*VALUESYMBOL NAME
-----
```

Hình 2-7. Tập đuôi .MAP của ví dụ 2-2.ASM

```
FileEditSearchViewOptionsHelp
+----- C:\avs\VD2-2.HEX -----+
03000000020100FA_
0E0100007455F58074AAF58074FFF5A080FE9A_
00000001FF
```

Hình 2-8. Tập đuôi .HEX cho ví dụ 2-2.ASM

2.3.3. Biến đổi thành dạng đuôi .HEX hay .BIN

Sau khi liên kết, ta có tên tệp dạng .HEX theo hàng ngang (hình 2-8 cho Ví dụ 2-2). Để có thể nạp các mã lệnh vào bộ nhớ chương trình ROM, ta phải biến đổi tệp trên thành tệp có dạng đuôi .HEX theo hàng dọc (dùng chương trình HECTOMCS) hay .BIN với các chương trình tương ứng (HEXOBJ.EXE, HEX2OBJ.EXE). Ví dụ, biến đổi thành tệp đuôi HEX, với chương trình hợp dịch ASM51.EXE ta có cú pháp sau:

```
C:> OHvi_du2_1 <Enter>
```

Với OH.EXE là chương trình biến đổi chương trình có đuôi .EXE sang đuôi .HEX

Với chương trình hợp dịch AVMAC51.EXE, nếu dùng chương trình biến đổi HEXTOMCS.EXE, ta biến đổi tệp .EXE thành tệp HEX theo hàng dọc, thuận tiện cho việc nạp chương trình vào bộ nhớ.

Chúng ta có thể kết hợp cả hai bước hoặc ba bước hợp dịch (dùng AVMAC51.EXE), liên kết (dùng AVLINK.EXE) và biến đổi (dùng HEXTOMCS.EXE) trong một tệp chương trình có đuôi .BAT như chương trình C.BAT (hợp dịch và liên kết) hay CL.BAT (cả hợp dịch, liên kết và biến đổi) để hợp dịch chương trình, liên kết và biến đổi chương trình.

2.4. HỢP DỊCH VÀ BIẾN ĐỔI CHƯƠNG TRÌNH VỚI PINNACLE

2.4.1. Hợp dịch chương trình với PINNACLE

Với chương trình mô phỏng và hợp dịch PINNACLE, quá trình lập trình đơn giản hơn nhiều, ở chỗ:

- Chỉ cần nháy chuột vào thực đơn thay vì gõ phím phức tạp.
- Thực hiện chương trình PINNACLE.EXE tức gọi màn hình có các chức năng soạn thảo, hợp dịch, liên kết và chạy thử chương trình trên chương trình mô phỏng.

Ví dụ, ta đã soạn thảo xong VD2-2.ASM, hợp dịch và liên kết theo các bước sau:

1. *Gọi lại chương trình lên màn hình:* Nháy đúp vào chương trình Pinnacle, vào thực đơn FILE/OPEN, chọn chương trình có đuôi .ASM, tìm tên chương trình rồi nháy đúp.

2. *Hợp dịch hay hợp dịch và liên kết chương trình:* Vào thực đơn Project, chọn thực đơn con (đọc xuống), tìm thực đơn Compile để dịch hoặc Compile & Link để hợp dịch và liên kết.

Trên màn hình sẽ hiện kết quả hợp dịch, xem không có hay có lỗi.

3. *Xem chương trình dạng mã lệnh chứa trong bộ nhớ chương trình (PROM)*

Với thực đơn VIEW/Code Memory của Pinnacle chúng ta có thể xem được chương trình đã hợp dịch và liên kết chứa trong bộ nhớ chương trình (PROM) với các cột ADDR, OPCODE, ASC, LABEL, DISSASSEMBLY, COMMENT như hình 2-9.

4. *Xem tệp danh sách (đuôi .LST):*

Với thực đơn FLE/OPEN và ALL FILES, chúng ta có thể xem được tệp danh sách (đuôi .LST) của chương trình như hình 2-10.

5. *Xem sự bố trí chương trình trong bộ nhớ (đuôi .MAP):*

Với thực đơn FLE/OPEN và ALL FILES, chúng ta có thể xem được tệp đuôi .MAP của chương trình.

6. *Biến đổi tệp đuôi .HEX thành tệp dạng mã (Code Memory):*

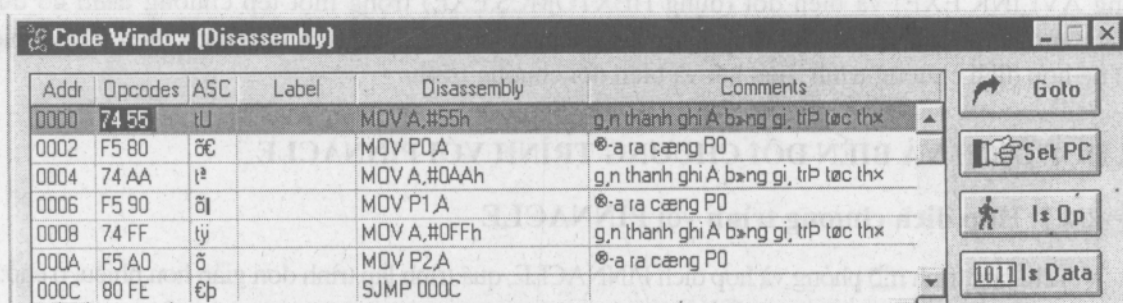
Với thực đơn FLE/OPEN và ALL FILES, có thể biến đổi được tệp đuôi .HEX thành tệp dạng mã của chương trình bằng cách:

- Vào thực đơn FILE/VIEW/VDU2-2.HEX để nạp vào bộ nhớ chương trình.
- Vào thực đơn FILE/VIEW/CODE DISSASSEMBLY để xem bộ nhớ chương trình.

Với phương pháp trên, ta có thể hồi phục và xem chương trình hợp ngữ cho VĐK khi chỉ biết chương trình dạng .HEX hay .BIN (qua biến đổi từ HEX sang BIN).

Chú ý: Khi soạn thảo chương trình, có thể có nhiều lỗi về hình thức và cú pháp của lệnh. Cần chú ý các lỗi thường gặp sau:

- Các nhãn phải gõ đúng như lần đầu sử dụng về chữ thường, chữ hoa và phải ở vị trí tận cùng bên trái của dòng lệnh, sau nhãn phải có dấu hai chấm (:).
- Các lệnh phải gõ đúng cú pháp về tên lệnh, thứ tự của toán hạng đích (D) và nguồn (S).
- Các dữ liệu phải đúng với dạng thập phân D (không cần có chữ D), HEXA (có chữ H) hay BIN (có chữ B) sau con số chỉ dữ liệu.
- Trước dữ liệu tức thì (giá trị cần nạp vào) phải có dấu thăng (#), nếu không, chương trình sẽ hiểu là địa chỉ của ô nhớ. Trước chú thích về lệnh, phải có dấu ;



Addr	Opcodes	ASC	Label	Disassembly	Comments
0000	74 55	U		MOV A, #55h	g.n thanh ghi A bằng gi. tức tốc thx
0002	F5 80	Ê		MOV P0, A	®-a ra cæng P0
0004	74 AA	U		MOV A, #0AAh	g.n thanh ghi A bằng gi. tức tốc thx
0006	F5 90	Ê		MOV P1, A	®-a ra cæng P0
0008	74 FF	U		MOV A, #0FFh	g.n thanh ghi A bằng gi. tức tốc thx
000A	F5 A0	Ê		MOV P2, A	®-a ra cæng P0
000C	80 FE	Ê		SJMP 000C	

Hình 2-9. Chương trình dạng mã (Code Window) trong bộ nhớ chương trình (PROM)

F:\ACTVDK\CHGTRI-1\VD21.LST

1	0000	; Ví dụ 2-2: Chương trình đưa dữ liệu ra cổng
2	0000	ORG 00H ; bắt đầu chương trình bằng địa chỉ nhớ 00H
3	0000	MOV A, # 55h ; gán thanh ghi A bằng giá trị tức thì 55H
4	0002	MOV PO, A ; đưa ra cổng PO
5	0004	MOV A, #0AAH ; gán thanh ghi A bằng giá trị tức thì 0AAH
6	0006	MOV P1, A ; đưa ra cổng PO
7	0008	MOV A, #OFFH ; gán thanh ghi A bằng giá trị tức thì OFFH
8	000A	MOV P2, A ; đưa ra cổng PO
9	000C	SJMP \$; chờ tại chỗ

Hình 2-10. Tập có đuôi .LST của ví dụ 2-2 trong Pinnacle

2.4.2. Chạy thử chương trình với PINNACLE

Với chương trình mô phỏng SIM51.EXE, AVSIM.EXE hay PINNACLE.EXE ta có thể cho chạy chương trình từng bước (từng lệnh), từng chương trình con, từng nhóm lệnh tới điểm ngắt hay cả chương trình.

Với chương trình PINNACLE, dùng thực đơn CODE MEMORY (DISASSEMBLY) và View để xem sự thay đổi địa chỉ nhớ của từng lệnh, từng thanh ghi, từng ô nhớ v.v.

Chương trình PINNACLE 52 là một IDE (Integrated equipment- thiết bị tích hợp) gồm:

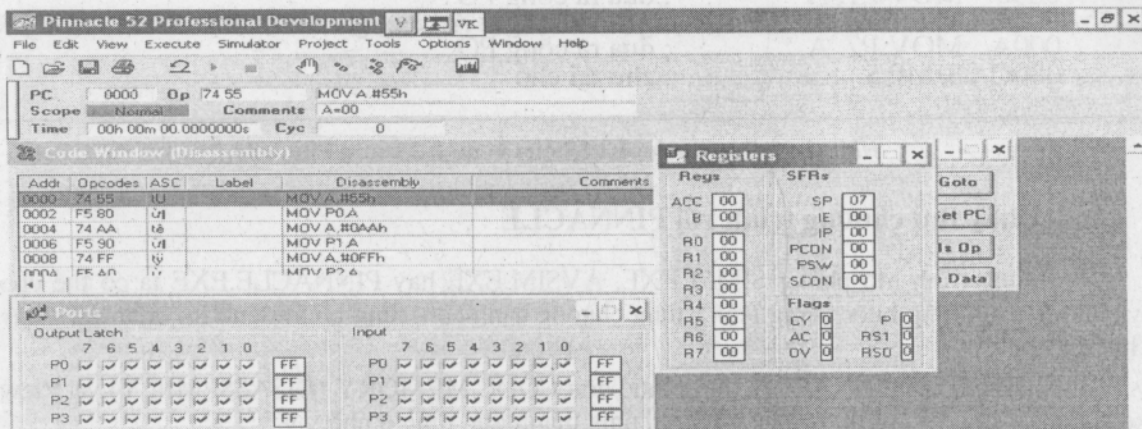
- Chương trình soạn thảo (Edit) chương trình.
- Chương trình biên dịch (compiler), kết nối và xếp đặt lại (Relocatable) chương trình hợp ngữ (có đuôi .ASM).
- Chương trình biến đổi dạng HEX sang BIN và ngược lại (trong thư mục OPTION).
- Chương trình mô phỏng (Simulator) để chạy thử trên máy vi tính PC (chạy từng bước, từng chương trình con và cả chương trình).

Cụ thể là, chạy với các biểu tượng trên màn hình (hình 2-11) theo trình tự sau :

- Mở tệp của chương trình cần chạy có đuôi .HEX (vào FILE/OPEN).
- Vào thư mục CODE MEMORY để có danh sách chương trình với địa chỉ, mã lệnh, tên lệnh dạng hợp ngữ tương tự tệp có đuôi .LST (hình 2-10).
- Vào thực đơn VIEW và các bộ phận phần cứng có liên quan, ví dụ, chương trình VDU2-1.ASM chỉ có Register và Port như ở hình 2-11 để xem sự thay đổi nội dung của chúng khi thực hiện từng lệnh của chương trình.
- Nháy vào hình “mũi tên ngược chiều kim đồng hồ màu đen” (RESTART hay RESET) để đưa về địa chỉ nhớ 00H (PC = 00H), có một vạch màu xanh trên lệnh có địa chỉ PC = 00H.
- Nháy vào hình “một dấu chân” (hay lệnh SINGLE trong EXECUTE) để chạy từng lệnh.
- Nháy vào hình “hai dấu chân” (hay lệnh Subroutine trong EXECUTE) để chạy từng chương trình con.
- Nháy vào hình “vòng đỏ xuôi chiều kim đồng hồ” (hay lệnh SKIP STEP trong EXECUTE) để bỏ qua chương trình con hay lệnh chờ.
- Nháy vào hình “tam giác xanh” (hay lệnh RUN) để chạy cả chương trình (xuất hiện dấu hình vuông đỏ).
- Nháy vào hình “vuông đỏ” (hay lệnh STOP) để dừng chạy chương trình tại một lệnh bất kỳ.
- Muốn chạy một lệnh hay chương trình từ một lệnh bất kỳ: nháy vào thanh ghi đếm lệnh PC và gõ vào địa chỉ cần, chương trình sẽ chỉ tại lệnh đó bằng vạch màu xanh trên lệnh đó.

Với thư mục VIEW và các thực đơn đọc, chúng ta có thể quan sát được sự thay đổi của các thanh ghi, các ô nhớ, các cổng song song, các Timer/Counter của vi điều khiển (xem chương 1).

Với cách chạy chương trình từng lệnh trên, chúng ta có thể theo dõi được từng bước diễn biến (từng lệnh hay từng chương trình con) và giải thuật của chương trình. Do đó có thể phát hiện sai sót về lệnh, về giải thuật, còn chương trình hợp dịch chỉ phát hiện lỗi về hình thức và cú pháp của lệnh.



Hình 2-11. Hiện CODE MEMORY, thanh ghi và cổng để chạy chương trình trong Pinnacle

2.5. LỆNH GỘP, CHƯƠNG TRÌNH CON, THƯ VIỆN CHƯƠNG TRÌNH VÀ LẬP TRÌNH VỚI LỆNH GỘP

2.5.1. Lệnh gộp (lệnh macro)

Tương tự chương trình con, thay cho một nhóm lệnh hay lặp lại trong chương trình chính, người ta có thể thay nhóm lệnh lặp lại trên bằng một lệnh gộp (lệnh macro - lệnh lớn). Với cách viết lệnh gộp này, người ta có thể:

- Mở rộng các lệnh gộp tiện ích cho VĐK để hoạt động của VĐK càng phong phú hơn.
- Lập trình với bộ lệnh gộp cho gần với ngôn ngữ bậc cao, tránh những rắc rối của hợp ngữ.

Các VĐK 8048, 8051 đã được trang bị các bộ lệnh gộp tương ứng để lập trình với lệnh gộp.

Các chương sau, trong mỗi chương, chúng ta sẽ viết những lệnh gộp cho VĐK 8051/52 và lập trình với lệnh gộp.

1. Định nghĩa

Lệnh gộp (lệnh Macro) là một nhóm lệnh có hướng dẫn biên dịch mở đầu MACRO và kết thúc ENDMAC và ở giữa là nhóm lệnh như sau:

```
MACRO @Tên lệnh <tham số>
; các lệnh
ENDMAC
```

Khai báo hay định nghĩa trên thích hợp cho chương trình dịch Pinnacle, có thể viết bằng chữ hoa hay chữ thường. Với chương trình dịch khác (cho vi xử lý 8086), có thể thay đổi, ví dụ, đặt tên lệnh trước.

Ký hiệu “a cộng” (@) luôn đứng trước tên lệnh. Như tên chương trình con (là một nhãn), tên lệnh gộp (có @ đứng trước) do người lập trình tự đặt, là một giả lệnh (Pseudo option), đặc trưng cho hành động của nhóm lệnh chứa trong thân của lệnh gộp. Nên đặt tên gợi nhớ (mnemonic), nói lên hành động của nhóm lệnh trên, để dễ nhớ.

Sau tên lệnh, có thể không có hay có tham số. Các tham số này sau tên lệnh, chỉ là tham số hình thức được ghi vào để biết được là lệnh gộp có tham số hay không. Do đó, khi khai báo, tham số sau tên lệnh có kèm theo dấu hỏi chấm (?), để lưu ý người sử dụng lệnh gộp là lệnh có bao nhiêu tham số và phải đưa giá trị cụ thể của tham số khi dùng lệnh gộp ở trong chương trình.

Trong khai báo, tại vị trí cần thay tham số, ta đánh dấu bằng dấu gạch đứng (!) và một con số tự nhiên 1, 2, 3, 4 v.v... chỉ số thứ tự của tham số ở tên của lệnh gộp khi gọi tới trong chương trình.

Các tham số trên phải được ghi giá trị bằng một con số cụ thể khi gọi tên lệnh gộp, tại vị trí trong chương trình. Khi thay tên lệnh gộp, các tham số này được thay vào các vị trí !1, !2, !3, !4 có tham số. Cần chú ý là, khai báo tham số theo đúng thứ tự định sẵn, để tránh diễn nhầm tham số.

Dữ liệu của tham số có thể là địa chỉ, hằng số (với kiểu khác nhau như HEX), để trống hay số thập phân D v.v.), nhãn, chỉ số và biến số. Nếu là biến, phải có khai báo kiểu dữ liệu ở đầu chương trình. Một lệnh gộp có thể không có tham số hay nhiều tham số tức có nhiều dữ liệu cho một hay nhiều lệnh của một lệnh gộp trong chương trình.

2. Gọi lệnh gộp trong chương trình chính

Trong một chương trình, lệnh gộp này nằm ở:

- Đầu chương trình với định nghĩa và coi như khai báo ở trên (giống khai báo dữ liệu).
- Vị trí cần thiết trong chương trình chính với tên lệnh gộp có chữ ‘a cộng’ (@) đứng trước.

Có thể có nhiều lệnh gộp trong một chương trình chính.

Khi lệnh gộp không có tham số, chỉ cần đặt @Tenlenhgop vào vị trí cần lệnh gộp.

Khi lệnh gộp có tham số, sau @ và tên lệnh gộp, phải có các giá trị cụ thể của các tham số với đơn vị của nó. Các tham số cách nhau bởi dấu phẩy.

Khi hợp dịch, chương trình dịch sẽ bỏ qua định nghĩa (hay khai báo) của lệnh gộp mà dịch từ phần chương trình chính. Khi gặp tên lệnh gộp, chương trình dịch sẽ tìm lệnh gộp đã khai báo hay định nghĩa ở phần đầu chương trình để:

- Chép toàn bộ các lệnh của lệnh gộp vào vị trí có tên lệnh gộp (thay gọi chương trình con).
- Lần lượt thay giá trị của tham số theo các số thứ tự (không có số 0) 1, 2, 3, 4 v.v... trong nhóm lệnh của lệnh gộp.
- Hợp dịch thành mã lệnh.
- Lệnh gộp ở đầu chương trình sẽ mất đi sau quá trình 1 của biên dịch (cẩn thận hơn, có khi người ta chèn lệnh gộp giữa hai điều khiển biên dịch có điều kiện IF1...ENDIF).

Như vậy, lệnh gộp khác với chương trình con ở chỗ:

+ Dùng lệnh gộp tiết kiệm được thời gian vì không có quá trình gọi chương trình con (bởi lệnh CALL) và trở về chương trình chính (bởi lệnh RET).

+ Lệnh gộp chèn nhóm lệnh của lệnh gộp vào chương trình chính tại vị trí có tên lệnh gộp ở giai đoạn 1 của hợp dịch (tương tự thay nhân của chương trình bằng địa chỉ của lệnh) và sẽ dịch ở giai đoạn 2 thành mã lệnh. Đây là nhược điểm lớn nhất của việc dùng lệnh gộp vì tốn bộ nhớ để ghi lại các lệnh của lệnh gộp mỗi khi gặp và tốn thời gian dịch.

+ Có thể truyền tham số cho lệnh gộp. Đây là ưu điểm của lệnh gộp so với chương trình con. Vì một chương trình chính có thể có nhiều lệnh gộp như nhau, nhưng tham số khác nhau. Với chương trình con, trước khi gọi, phải có lệnh gán tham số cho các lệnh của chương trình con.

Ví dụ 2-6 giới thiệu chương trình ghi, đọc ô nhớ lệnh gộp không tham số và tệp đuôi .LST như hình ở hình 2-12.

Ví dụ 2-6: Chương trình đọc ô nhớ 30H, 40H, 50H, P0, P1, P2, dùng lệnh gộp không tham số

```
MACRO @docnho          ; khai lệnh gộp docnho
    mov a, @r0          ; đọc ô nhớ ra thanh ghi A
ENDMAC
    org 00H
    mov r0, #30h        ; địa chỉ nhớ 30H
    @docnho
    mov P0, a           ; đưa A ra cổng P0
    mov r0, #40h        ; địa chỉ nhớ
    @docnho             ; đọc ô nhớ ra thanh ghi A
    mov P1, a           ; đưa A ra cổng P1
    mov r0, #50h        ; địa chỉ nhớ
    @docnho             ; đọc ô nhớ ra thanh ghi A
    mov P2, a           ; đưa A ra cổng P2
    sjmp $
end
```

Chương trình trên vẫn còn dài vì phải lặp lại ba lần địa chỉ nhớ và cổng Pi. Nếu dùng tham số, tham số 1 cho địa chỉ nhớ, tham số 2 cho tên cổng, ta có chương trình ngắn gọn như ví dụ 2-7 và chương trình dịch ở hình 2-13.

So với chương trình viết thẳng VDU2-2, chương trình dùng lệnh gộp không có tham số, chương trình dùng lệnh gộp có tham số này ngắn gọn và đơn giản hơn nhiều. Chương trình dịch có tham số với đuôi .LST (hình 2-12) của VDU2-6 có hàng ngang rộng. Sau khi đã chỉnh theo dòng, ta có hình 2-12 với các dấu hiệu MACLEVEL chỉ vị trí đã thay thế lệnh gộp MACRO @TEN.

Tương tự, ta có thể viết chương trình đọc dữ liệu từ cổng vào thanh ghi chứa A hay ghi vào địa chỉ nhớ (ví dụ 2-7).

Các lệnh gộp trên sẽ đưa vào thư viện các lệnh gộp để tiện dùng sau này.

1	0000 ; VDU2-6: Chương trình đọc ô nhớ không có tham số	
5	0000 org 00H	
6	0000 mov r0, #30h	; địa chỉ nhớ 30H
7	0002 MACLEVEL 00017	
	0002mov a, @r0	; đọc ô nhớ ra thanh ghi A7
	0003 MACLEVEL 00008	
	0003 mov P0, a	; đưa A ra cổng P0
9	0005mov r0, #40h	; địa chỉ nhớ
10	0007 MACLEVEL 000210	
	0007mov a, @r0	; đọc ô nhớ ra thanh ghi A10
	0008 MACLEVEL 000011	
	0008 mov P1, a	; đưa A ra cổng P1
12	000Amov r0, #50h	; địa chỉ nhớ
13	000C MACLEVEL 000313	
	000Cmov a, @r0	; đọc ô nhớ ra thanh ghi A13
	000D MACLEVEL 000014	
	000D mov P2, a	; đưa A ra cổng P2
15	000F sjmp \$	

Hình 2-12. Tập .LST của chương trình đọc ô nhớ không có tham số cho ví dụ 2-6

<i>; Ví dụ 2-7: Chương trình đọc các ô nhớ 30H, 40H, 50H có tham số</i>		
MACRO @docnho <diachi?>,<cong?> ; mở đầu lệnh gộp		
		; kê khai lệnh gộp docnho <diachi?>, <cong?>
	mov r0, #1	; tham số 1- địa chỉ ghi vào r0
	mov a, @r0	; đọc ô nhớ ra thanh ghi A
	mov l2, A	; tham số 2- cổng
	ENDMAC	; kết thúc lệnh gộp
	org 00H	
	@docnho 30H, P0	; đọc ô nhớ ra cổng P0
	@docnho 40H, P1	; đọc ô nhớ ra cổng P1
	@docnho 50H, P2	; đọc ô nhớ ra cổng P2
	sjmp \$	
	end	

Từ ví dụ 2-6 và hình 2-12, ta thấy:

- Khai báo hay định nghĩa các lệnh gộp @docnho và @ghinho không được dịch ở chỗ khai báo mà được dịch tại vị trí của lệnh gộp trong chương trình. Các dấu đề, tên lệnh và kết thúc chỉ là lệnh giả hay nhảm nên không được dịch, chỉ có các lệnh của lệnh gộp được dịch ra mã máy.

- Tập có đuôi .LST của chương trình có lệnh gộp khác với chương trình không có lệnh gộp:

+ Tại dòng có lệnh gộp, có ký tự MAC với các dòng lệnh của lệnh gộp.

+ Không theo thứ tự các dòng cho từng lệnh mà theo dòng ngang.

+ Căn chỉnh lại theo từng dòng lệnh, ta có hình 2-14.

Ví dụ 2-8 giới thiệu các lệnh gộp @ghicong <cong?>, <giá trị ghi> và @doccong <cong?> trong chương trình ghi dữ liệu (#57H) vào cổng P0 rồi đọc từ A ra cổng P1.

Code Window (Disassembly)

Addr	Opcodes	ASC	Label	Disassembly	Comments
0000	78 30	x0		MOV R0,#30h	
0002	E6	đ		MOV A,@R0	đọc ô nhớ ra thanh ghi A
0003	F5 80	ừ	(EXT0 INT)	MOV P0,A	
0005	78 40	x@		MOV R0,#40h	
0007	E6	đ		MOV A,@R0	đọc ô nhớ ra thanh ghi A
0008	F5 90	ừ		MOV P1,A	
000A	78 50	xP		MOV R0,#50h	
000C	E6	đ		MOV A,@R0	đọc ô nhớ ra thanh ghi A
000D	F5 A0	ừ		MOV P2,A	
000F	80 FE	ly		SJMP 000F	
0011	00	I		NOP	

Hình 2-13. Chương trình dịch của chương trình đọc ô nhớ có tham số cho ví dụ 2-7

; Ví dụ 2-8: Đọc cổng vào A, ghi a ra cổng

macro @doccong <cong?>, <dulieu?>

movl1, #l2 ; l1- tham số cổng; l2- tham số dữ liệu

mova, l1 ; l1- tham số cổng

endmac

macro @ghicong <?cong?>

mov l1, a ; l1- tham số cổng

endmac

org 00h

@doccongP0, 57h

@ghicong P1

sjmp \$

end

3. Các toán tử của lệnh gộp

Trong chương trình Pinnacle, lệnh gộp có 4 toán tử sau:

- *Toán tử %*: khi dấu phần trăm này đứng trước một ký hiệu trong danh sách của thông số, giá trị của ký hiệu được chuyển tới thân của lệnh gộp thay cho chính ký hiệu đó.

- *Toán tử !*: khi dấu chấm than này đứng trước một ký tự, thì ký tự được xem như một chữ (literal) và được chuyển vào thân của lệnh gộp với dấu chấm than đã mất đi.

Ví dụ: GENERATE_ISNT75, !, , , STK_VALUE thì thông số thứ hai sẽ là một dấu phẩy (,).

- *Toán tử &*: khi dấu này được dùng trong thân của lệnh gộp, các ký hiệu ở cả hai phía của nó được cùng xuất hiện (concatened) và mất dấu & (toán tử VA).

- *Toán tử ;*: khi có hai dấu chấm phẩy (;) được dùng trong lệnh gộp, thì chú thích đứng sau hai dấu chấm phẩy sẽ không được lưu giữ và như vậy sẽ không xuất hiện trong danh sách dù rằng lệnh gộp vẫn được đưa vào.

4. Đặc điểm của lệnh gộp

Cũng giống chương trình con, lệnh gộp có đặc tính lồng (nesting) nhau và nhãn (label).

Đặc tính lồng nhau là các lệnh gộp đan xen nhau: trong một lệnh gộp có thể có một lệnh gộp khác.

a) Lồng nhau

; Ví dụ 2-9: Cộng hai ô nhớ (lệnh gộp lồng nhau)

```
MACRO @docnho đích?
    MOV R0, #11
    MOV A, @R0
ENDMAC
MACRO @congno đích?, nguồn?
    @docnhodiachi?
    MOV B, A
    @docno diachi?
    ADDA, B
    MOV @I2, A
ENDMAC
ORG 00H ; chương trình chính
@ CONGNHO30H, 40H ; cộng hai ô nhớ địa chỉ 30H, 40H, rồi gửi kết quả vào 30H
SJMP $ ; chờ
END ; kết thúc dịch
```

; Ví dụ 2-10: Dịch có điều kiện (if... then... else)

```
a equ 20h ; gán các giá trị cho biểu thức sau IF
cequ 30h
if (a+c < 60h)
    mov A, #20H ; dịch nếu biểu thức đúng, vì đã cho a+c = 20h+30h = 50h
else
    mova, #50H ; dịch nếu biểu thức sai vì đã cho a+c = 20h+30h = 40h
endif
sjmp $
end
```

Ví dụ 2-9 giới thiệu chương trình lệnh gộp đọc và cộng hai byte nhớ trong. Chúng ta chú ý rằng lệnh gộp này chứa hai lệnh gộp docnho. Các tham số cho hai lệnh đọc nhớ cũng là tham số cho lệnh gộp lồng nhau @congno. Kết quả của phép cộng lại được ghi vào địa chỉ đích, cũng chính là địa chỉ của lệnh gộp @docnho. Sau khi đọc ô nhớ nguồn đầu tiên, phải gửi vào thanh ghi B, để cộng với thanh ghi A của lần đọc ô nhớ nguồn (S) thứ hai. Kết quả lại gửi vào ô nhớ đích (D) hay ô nhớ nguồn (S) thứ nhất.

b) Dịch có điều kiện

Lệnh gộp cũng như chương trình chính, có thể được dịch có điều kiện theo công thức:

Ví dụ 2-10 giới thiệu chương trình dịch có điều kiện với biểu thức $a+c < 60H$. Chúng ta có thể cho chương trình hợp dịch và liên kết để xem chương trình dịch. Có thể thay đổi điều kiện của biểu thức để xem chương trình dịch đoạn nào? (sau THEN hay sau ELSE?).

Tương tự, dịch có điều kiện có thể áp dụng cho VALUE đã được xác định hay định nghĩa như ở ví dụ 2-11 với công thức:

```
IF <biểu thức thoả mãn >
THEN <dịch đoạn lệnh này>
ELSE <dịch đoạn lệnh này nếu biểu thức không thoả mãn>
```

c) Sử dụng nhãn

Tương tự chương trình chính có sử dụng nhãn để lặp lại một số lệnh.

Nhãn này có thể :

- **Đặt sẵn:** Khi dùng hay gọi nhiều lần lệnh gộp, nhãn này tự động thay đổi chỉ số, ví dụ 2-12.

; Ví dụ 2-11: ifdef và ifndef

```
org 00h
value equ 20h
ifndef value
mov P0, #20h ; dịch với ifndef hay với ifndef nếu chưa có value (khai báo value sau dấu (;))
else
mov P1, #30H ; dịch với ifdef hay ifndef nhưng chưa có value (khai báo value sau dấu (;))
endif
sjmp $
end
```

; Ví dụ 2-12: Lệnh gộp có nhãn đặt sẵn

```
MACRO @MULTIPLE_SHIFT< A?, R0?>
    MOV A, #11
    MOV R0, #12
LOOP: RL A
    DJNZ R0, LOOP
ENDMAC
ORG 00H
@MULTIPLE_SHIFT, 55h, 8
@MULTIPLE_SHIFT, 0aah, 12
SJMP $
END
```

- **Chưa đặt hay gán sau:** coi như là một tham số. Trong ví dụ này, số lần lặp lại coi như một tham số và chúng ta có thể gán sau như ví dụ 2-13 và hình 2-15.

So sánh các nhãn của hai ví dụ trên, ta thấy mỗi lần gán nhãn, chương trình dịch sẽ tự động gán thêm chỉ số cho nhãn ở những lần gọi lệnh gộp.

; Ví dụ 2-13: Lệnh gộp có nhãn chưa đặt

```
MACRO @ MULTIPLE_SHIFTLOP?, N?
    MOV A, #3
14:  MOV R0, #13
12:  RL A                                ; nhãn bắt đầu ở số 2
    DJNZ R0, 12
ENDMAC
ORG 00H
@MULTIPLE_SHIFTLOP, 8, LEP
@MULTIPLE_SHIFTLOP, 4, LEPI
SJMP $
END
```

Việc thay đổi tên nhãn, gán nhãn sau, gán số lần lặp cho lệnh gộp là một ưu điểm vượt trội của lệnh gộp so với chương trình con.

Chúng ta cần lưu ý rằng mỗi lần gọi lệnh gộp, phải gán thông số nhãn với tên khác (Pinnacle không có từ điều khiển biên dịch LOCAL cho nhãn cục bộ của riêng lệnh gộp).

Code Window (Disassembly)

Addr	Opcodes	ASC	Label	Disassembly	Comments
0000	74 55	U		MOV A, #55h	
0002	78 08	xI		MOV R0, #08h	
0004	23	#	LOOP?1	RL A	
0005	D8 FD	ly		DJNZ R0, LOOP?1	
0007	74 AA	tê		MOV A, #0AAh	
0009	78 0C	xI		MOV R0, #0Ch	
000B	23	#	LOOP?2	RL A	
000C	D8 FD	ly		DJNZ R0, LOOP?2	
000E	80 FE	lv		SJMP 000E	

Hình 2-14: Chương trình có lệnh gộp với nhãn đặt sẵn của ví dụ 2-12

Code Window (Disassembly)

Addr	Opcodes	ASC	Label	Disassembly	Comments
0000	74 03	U		MOV A, #03h	
0002	78 04	xI		MOV R0, #04h	
0004	23	#	LOOP?1	RL A	nhãn bắt đầu ở số 2
0005	D8 FD	ly		DJNZ R0, LOOP?1	
0007	74 03	U		MOV A, #03h	
0009	78 06	xI		MOV R0, #06h	
000B	23	#	LOOP?2	RL A	nhãn bắt đầu ở số 2
000C	D8 FD	ly		DJNZ R0, LOOP?2	
000E	74 03	U		MOV A, #03h	
0010	78 06	xI		MOV R0, #06h	
0012	23	#	LOOP?3	RL A	nhãn bắt đầu ở số 2
0013	D8 FD	ly	(EXT1 INT)	DJNZ R0, LOOP?3	
0015	80 FE	lv		SJMP 0015	

Hình 2-15. Chương trình có lệnh gộp với nhãn chưa đặt của ví dụ 2-13

2.5.2. Chương trình con và thủ tục gọi

Chương trình con gồm một nhóm lệnh (số lượng lệnh nhiều hơn lệnh gộp, có thể chứa một hay nhiều lệnh gộp), được sử dụng lặp lại nhiều lần trong chương trình chính.

Để tiết kiệm khối nhớ, người ta viết nhóm lệnh lặp lại này thành một chương trình con để được hợp dịch và sử dụng chung, nằm dưới chương trình chính và sẽ nhảy tới (hay gọi) khi sử dụng.

Chương trình con không khai báo (hay định nghĩa) và gọi tên trong chương trình chính như lệnh gộp.

Nhưng cái giá phải trả cho việc tiết kiệm được bộ nhớ này là tốn thời gian để làm thủ tục ngắt chương trình chính khi gọi chương trình con và trở về chương trình chính khi gặp lệnh RET (RETurn - trở về) ở cuối chương trình con.

Với chương trình hợp dịch cho VĐK 8051/52, chương trình con gồm nhóm lệnh, với khai báo với tên như một nhãn và có lệnh RET (RETurn - trở về) hay RETI (trở về từ ngắt cho chương trình con phục vụ ngắt, xét ở chương 10) để kết thúc.

Chương trình con này thường nằm ở cuối chương trình chính, sau lệnh chờ tại chỗ SJMP \$ để dừng việc chạy chương trình chính. Có thể, chương trình con nằm ở đoạn nhớ khác với địa chỉ khai báo ORG XXXXH hoặc trong thư viện chương trình con (xem ở dưới) với chỉ dẫn biên dịch INCLUDE.

; Ví dụ 2-14: Chương trình con ghi và đọc ô nhớ rồi đưa ra cổng

org 00h	
mov a, #55h	; ghi #55H vào a
mov r0, #30H	; ghi #30H vào r0 làm địa chỉ nhớ
call ghinho	; gọi chương trình con ghi dữ liệu #55H vào ô nhớ địa chỉ 30H
mov a, #65h	; ghi #65H vào a
mov r0, #40H	; ghi #40H vào r0 làm địa chỉ nhớ
lcall ghinho	; gọi chương trình con ghi dữ liệu #65H vào ô nhớ địa chỉ 40H
mov r0, #30H	; ghi #30H vào r0 làm địa chỉ nhớ
call docnho	; gọi chương trình con đọc ô nhớ địa chỉ 30H vào a
mov P0, a	; đưa dữ liệu từ a ra P0
mov r0, #40H	; ghi #40H vào r0 làm địa chỉ nhớ
call docnho	; gọi chương trình con đọc ô nhớ địa chỉ 40H
mov P1, a	; đưa dữ liệu từ a ra P1
sjmp \$	; kết thúc chờ
ghinho:	; chương trình con ghi dữ liệu a vào ô nhớ địa chỉ trong r0
mov @r0, a	; ghi dữ liệu a vào ô nhớ địa chỉ trong r0
ret	; trở về chương trình chính
docnho:	; chương trình con đọc dữ liệu ở ô nhớ địa chỉ trong r0 vào a
mov a, @r0	; đọc dữ liệu ở ô nhớ địa chỉ trong r0 vào a
ret	; trở về chương trình chính
end	; kết thúc dịch

Trước khi thực hiện chương trình con, VĐK hay vi xử lý phải cất giữ nội dung của các thanh ghi đếm lệnh PC, thanh ghi chứa A và thanh ghi trạng thái PSW vào vùng nhớ ngăn xếp bởi các lệnh PUSH các thanh ghi mà địa chỉ các ô nhớ ngăn xếp là nội dung của thanh ghi Stack SP.

Sau khi kết thúc chương trình con, tức chương trình con gặp lệnh trở về chương trình chính (lệnh RET hay RETI), VDK nạp lại nội dung cũ của các thanh ghi PC và PSW bởi lệnh POP thanh ghi đã gửi trong ngăn xếp, để trở về chỗ gọi chương trình con, tức thực hiện tiếp lệnh đứng sau lệnh gọi chương trình con CALL addr. của chương trình chính. Nói khác đi, các lệnh của chương trình con được gọi này được thực hiện chèn vào việc thực hiện chuỗi lệnh của chương trình chính khi chương trình con được gọi.

Trong một chương trình chính có nhiều lệnh hay hành động lặp lại, khi thay nhóm lệnh lặp lại trên bằng một chương trình con, chương trình chính viết ngắn gọn hơn, nhưng phải thêm (trả giá) bằng:

- Một lệnh gọi (CALL tên_chương_trình_con) chương trình con.
- Một lệnh trở về (RET hay RETI) từ chương trình con.
- Thực hiện thủ tục gọi chương trình con (các lệnh PUSH các thanh ghi Acc và PSW) và trở về từ chương trình con (các lệnh POP các thanh ghi PSW và Acc).

Tuy nhiên, chương trình con không có thể gán tham số như lệnh gộp nên phải tốn lệnh để gán dữ liệu rồi gọi chương trình con.

Ví dụ 2-14 giới thiệu chương trình có gọi các chương trình con đưa dữ liệu từ thanh ghi A vào các ô nhớ và đưa ra các cổng P0, P1.

2.5.3. Thư viện chương trình

Các lệnh gộp, các chương trình con của một hệ VDK có thể để chung trong một thư viện và gọi một hay nhiều lệnh gộp hay chương trình con đó trong một chương trình nào đó mới viết. Với cách này, chúng ta tiết kiệm được thời gian và công sức lập trình lệnh gộp hay chương trình con.

Với chương trình hợp dịch Pinnacle, các lệnh gộp được đặt chung trong một tệp thư viện với tên gọi bất kỳ, ví dụ, MYLIB.ASM với đuôi .ASM hay .INC đều dịch đúng.

- Đặt tên thư viện trên sau ký hiệu \$INCLUDE.
- Gọi tên của một trong các lệnh gộp tại dòng của chương trình cần lệnh gộp đó (sau dấu @).
- Khi dịch chương trình, chương trình hợp dịch gộp tên của lệnh gộp sẽ tìm trong thư viện tên của lệnh gộp và chép lại khai báo (hay định nghĩa) lệnh gộp đó vào vị trí có tên lệnh gộp.
- Tiến hành dịch các lệnh của lệnh gộp.

Tương tự, ta cũng gọi chương trình con từ thư viện chương trình con theo cách trên. Khi gặp lệnh gọi chương trình con không có ở cuối chương trình chính, chương trình dịch sẽ:

- + Tìm tên chương trình con ở thư viện chương trình con đã khai báo với \$INCLUDE MYLIB.ASM (hay INC) ở đầu chương trình chính.
- + Tìm tên chương trình con trong thư viện.
- + Sao chép lại chương trình con đó vào cuối chương trình chính.
- + Dịch chương trình con cùng với chương trình chính.

Nhưng có điều khác với lệnh gộp là chương trình con chỉ được tìm, sao chép và dịch chỉ một lần khi gặp tên chương trình con, trong khi đó, lệnh gộp phải tìm, sao chép và dịch nhiều lần (nếu được gọi nhiều lần).

Ví dụ 2-15 giới thiệu thư viện lệnh gộp INCLUDE2.ASM với đuôi .ASM (hay .INC), chương trình Pinnacle đều dịch đúng.

Ví dụ 2-16 và hình 2-16 giới thiệu chương trình gọi lệnh gộp trong thư viện chương trình với lệnh INCLUDE2.ASM.

; Ví dụ 2-15: Thư viện các lệnh gộp MYLIB.ASM

```
macro@docnhodia chỉ?, cổng ?      ; kê khai lệnh gộp@docnho diachi?
    mov l2, l1                      ; tham số 1- địa chỉ nhớ, tham số 2- cổng
endmac                              ; kết thúc lệnh đọc ô nhớ
macro@ghinhodia chỉ?, dulieu?      ; kê khai lệnh gộp@ghinho
    mov r0, #1                      ; tham số 1- địa chỉ nhớ
    mov a, #2                       ; tham số 2- dulieu
    mov @r0, a                     ; ghi vào ô nhớ
endmac                              ; kết thúc lệnh gộp ghi vào ô nhớ
```

; Ví dụ 2-16: Gọi chương trình trong thư viện lệnh gộp MYLIB.ASM

\$INCLUDEMYLIB.ASM

```
@ghinho 53H, 60h      ; ghi #60H vào ô nhớ có địa chỉ 53H
@docnho 53h, P0        ; đọc ô nhớ có địa chỉ 53H ra cổng P0
sjmp $                 ; kết thúc chờ
end                    ; kết thúc dịch
```

 Code Window (Disassembly)

Addr	Opcodes	ASC	Label	Disassembly	Comments
0000	78 53	xS		MOV R0,#53h	tham số 1- địa chỉ nhớ
0002	74 60	t		MOV A,#60h	tham số 2- dulieu
0004	F6	ở		MOV @R0,A	ghi vào ô nhớ
0005	85 53 80	ISI		MOV P0,53h	tham số 1- địa chỉ nhớ, tham số 2- cổng
0008	80 FE	ly		SJMP 0008	

Hình 2-16. Chương trình hợp dịch của ví dụ 2-16

Như vậy, ở chương này, chúng ta đã định nghĩa và thư viện lệnh gộp MYLIB.ASM (hay.INC) có 4 lệnh gộp sau:

1. DOCCONG <cong?>; đọc dữ liệu ở <cổng ?> vào thanh ghi A.
2. GHICONG <cong?>, <dulieu?>; ghi <dulieu?> ra <cổng?>
3. DOCNHO<diachi?> ; đọc ô nhớ <diachi?> vào thanh ghi A
4. GHINHO<diachi?>, <dulieu?> ; ghi<dulieu?> vào ô nhớ <diachi?>

Với chương trình con, trong thư viện chương trình con MYLIBP.ASM ta mới có hai chương trình sau: 1. DOCNHO.ASM và 2. GHINHO.ASM. <dulieu?>.

2.5.4. Lập trình với lệnh gộp của VDK 8048 và 8051

1. Yêu cầu lập trình với lệnh gộp

Trong VDK, lệnh gộp có ưu điểm là có thể gán tham số, gán nhãn và sử dụng chung thư viện với hướng dẫn biên dịch \$INCLUDE. Do đó, nếu có bộ lệnh gộp về các hành động cơ bản của một VDK, ta có thể lập trình hợp ngữ với thư viện lệnh gộp. Khi đó, bộ lệnh gộp được đặt trong bộ nhớ PROM, có chương trình dịch các lệnh gộp hay gọi các lệnh hợp ngữ của lệnh gộp.

Các VDK 8048, 8051 đã được trang bị các bộ lệnh gộp tương ứng để lập trình với lệnh gộp.

2. Bộ lệnh gộp của VDK 8048 và 8051

VDK 8048 và 8051 đã được trang bị chương trình dịch MC-48 và MC-51 với bộ lệnh gộp có các nhóm lệnh sau:

- Nhóm lệnh nạp và lưu trữ.
- Nhóm lệnh nhập, xuất.
- Nhóm lệnh ghi và đọc các thanh ghi đặc biệt SFR.
- Nhóm lệnh số học, logic.
- Nhóm lệnh cho các chương trình con (thủ tục).
- Nhóm lệnh cho vòng lặp, đếm.
- Nhóm lệnh cho cấu trúc chương trình.

Trong các chương sau, chúng ta sẽ đề cập tới các lệnh của các nhóm lệnh này và sẽ viết các lệnh gộp tương tự cho các VDK 8051/8052.

3. Lập trình với lệnh gộp của VDK 8048 và 8051

Với bộ lệnh gộp trên, ta có thể lập trình với các lệnh gộp như ví dụ 2-17. Chương trình với lệnh gộp này gần giống với chương trình ngôn ngữ bậc cao. Do đó, việc lập trình đơn giản hơn vì chỉ cần nhớ các lệnh gộp gắn với cấu trúc và hành động của VDK, không cần nhớ chi tiết phần cứng như các thanh ghi (chế độ, đệm dữ liệu v.v...).

; Ví dụ 2-17: Chương trình với các lệnh gộp của 8048 và 8051	
Procedure xuất_ra	; chương trình con xuất_ra
A+1	; tăng thanh ghi A lên 1
WrP1	; đưa thanh ghi A ra cổng P1
EndProc	; kết thúc chương trình con
Begin	; bắt đầu chương trình chính
Loop xuất_ra	; gọi chương trình con xuất_ra
End	; kết thúc chương trình

2.6. CHẠY CHƯƠNG TRÌNH TRÊN KIT VI ĐIỀU KHIỂN

Sau khi đã chạy thử chương trình (từng bước và cả chương trình) trên chương trình mô phỏng của máy vi tính PC, ta cho chạy chương trình trên kit VDK. Có hai cách chạy chương trình trên kit vi điều khiển là nạp chương trình vào bộ nhớ PROM trong vi điều khiển hay nạp vào bộ nhớ RAM ngoài để chạy.

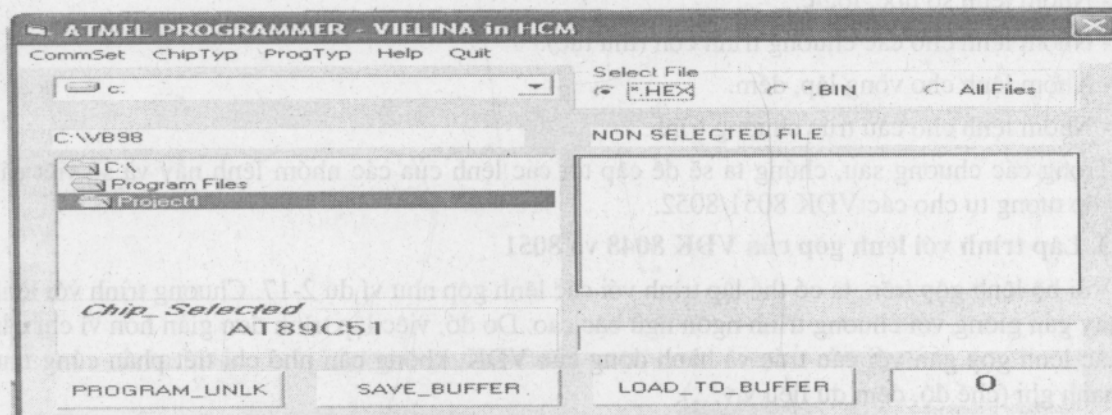
2.6.1. Chạy chương trình đã nạp trong PROM

Nạp chương trình từ máy tính PC vào bộ nhớ PROM bên trong của VDK

Dùng bộ nạp chương trình (Programmer) rời hay trên kit vi điều khiển (ví dụ MGH1), ta thực hiện các bước sau:

- Cắm vi mạch VDK vào bộ nạp chương trình PROGRAMMER và bật nguồn nuôi.
- Vào chương trình nạp (ghi sẵn trong PC) với giao diện như ở hình 2-17.
- Chọn cổng nối tiếp thích hợp của máy nạp nối với PC (COM1 hay COM2).
- Chọn loại VDK thích hợp (8051 hay 8052) từ thực đơn ChipType.

- Xóa bộ nhớ PROM trong VDK, với thực đơn ProgTyp.
- So sánh dữ liệu đã xóa với kết quả đã xóa bởi lệnh COMPARE trong thực đơn ProgTyp xem đã xóa hết chưa? Nếu xóa đúng, cửa sổ báo COMPARE COMPLET.
- Tải hay nạp tệp chương trình cần nạp (ví dụ 2-1.hex) từ máy vi tính vào bộ đệm (BUFER) của chương trình nạp.
- Chọn chế độ nạp UNLOCK hay LOCK (có bit khoá, chống sao chép) để nạp chương trình từ BUFER vào bộ nhớ PROM bên trong của VDK.
- Chọn lệnh COMPARE để so sánh có chương trình nạp với chương trình ghi xem có sai sót không. Chương trình sẽ báo đúng khi hiển thị COMPARE COMPLETE.



Hình 2-17. Giao diện chương trình nạp vào PROM của VDK

- Bật nguồn nuôi kit chứa VDK.
- Nhấn nút RESET để VDK trở về địa chỉ 0000H hay PC = 0000H để chạy lệnh đầu tiên của chương trình. Có thể, tại địa chỉ 0000H, VDK có lệnh LJMP MAIN để nhảy tới nhãn MAIN ở địa chỉ bắt đầu của chương trình chính (thường ở địa chỉ 100H), bỏ qua các vùng nhớ dành các vector ngắt (từ địa chỉ 03H tới 23H) cho các bit cờ (Flag) từ địa chỉ 20H và vùng nhớ đệm dữ liệu (từ địa chỉ 30H) và vùng nhớ ngăn xếp (từ địa chỉ 70H).

2.6.2. Chạy chương trình trên bộ nhớ RAM ngoài

Muốn chạy chương trình trên bộ nhớ RAM ngoài mở rộng, trên kit VDK, phải có:

- Kit VDK với bộ nhớ mở rộng RAM ngoài.
- Có chương trình trao đổi tin nối tiếp với VDK trên máy vi tính PC như TERM.EXE của NC hay SERIAL.EXE tự viết.
- VDK phải có chương trình trao đổi tin nối tiếp giữa PC, VDK và bộ nhớ RAM ngoài (chương trình RAM.EXE ở phụ lục D).
- Bật chuyển mạch hoạt động cho cổng nối tiếp trên VDK sang vị trí EXTERN để nạp chương trình cần chạy (đuôi .HEX hay .BIN tùy chương trình yêu cầu) trong bộ nhớ của PC vào bộ nhớ RAM ngoài thông qua các chương trình TERM.EXE của PC và chương trình trao đổi tin nối tiếp giữa PC, VDK và bộ nhớ RAM ngoài.
- Bật chuyển mạch hoạt động cho cổng nối tiếp trên VDK từ vị trí EXTERN sang vị trí INTERN trên kit VDK để VDK giao tiếp với bộ nhớ RAM ngoài.
- Ấn nút RESET để chạy chương trình đã nạp trên bộ nhớ RAM ngoài.

CÂU HỎI VÀ THỰC HÀNH

I. Câu hỏi

1. Hợp ngữ là gì? Nêu các thành phần của hợp ngữ. Hãy so sánh hợp ngữ với ngôn ngữ mã máy.
2. Nêu cấu tạo một lệnh dạng hợp ngữ. So sánh dạng lệnh mã với lệnh dạng hợp ngữ. Tại sao dùng lệnh dạng thuật ngữ?
3. Nêu các hướng dẫn và điều khiển biên dịch của hợp ngữ cho VDK họ MCS-51.
4. Chương trình dạng hợp ngữ là gì? Kể dạng của chúng cho các trường hợp khác nhau và chương trình hợp dịch khác nhau.
5. Chương trình hợp dịch dùng làm gì? Kể một số dạng chương trình hợp dịch cho VDK họ MCS-51.
6. Chương trình liên kết và biến đổi dùng làm gì cho VDK?
7. Lệnh gộp là gì? Hãy phân biệt lệnh gộp với chương trình con và với chương trình chính.
8. Hãy khai báo (hay định nghĩa) lệnh gộp không có và có tham số. Kể cách gọi và dịch lệnh gộp trong chương trình.
9. Kể các tính chất và ứng dụng của lệnh gộp.
10. Nêu định nghĩa, cách viết, tính chất và ứng dụng của chương trình con.
11. Nêu cách thành lập, cách gọi chương trình trong thư viện các lệnh gộp.
12. Nêu cách thành lập, cách gọi chương trình trong thư viện các chương trình con.
13. Chạy thử chương trình trên chương trình mô phỏng dùng để làm gì? Có thể kiểm tra được khâu nào trong quá trình lập trình hợp ngữ cho VDK?
14. Nêu cách đọc và nạp chương trình trong ổ đĩa của máy vi tính PC sang bộ nhớ PROM của VDK.
15. Dùng chương trình hợp dịch Pinnacle.exe có mấy cách chạy chương trình trên kit VDK? Ưu nhược điểm của từng cách.

II. Thực hành

1. Dùng chương trình hợp dịch ASM51.exe, soạn thảo, hợp dịch, kết nối và cho chạy thử (từng bước, cả chương trình) một chương trình đưa tin ra các cổng P0 và P2.
2. Dùng chương trình hợp dịch AVMAC51.exe, soạn thảo, hợp dịch, kết nối và cho chạy thử (từng bước, cả chương trình) một chương trình đưa tin ra các cổng P0 và P2.
3. Dùng chương trình hợp dịch Pinnacle.exe, soạn thảo, hợp dịch, kết nối và cho chạy thử (từng bước, cả chương trình) một chương trình đưa tin ra các cổng P0 và P2.
4. Dùng chương trình mô phỏng SIM51.EXE (nếu có) để chạy chương trình đưa tin ra các cổng P0, P2 ở trên.
5. Dùng chương trình hợp dịch Pinnacle.exe và các ô nhớ đệm địa chỉ từ 30H, soạn thảo, hợp dịch, kết nối và cho chạy thử (từng bước, cả chương trình) một chương trình đưa tin từ các ô nhớ trên ra các cổng P0 và P2.
6. Dùng chương trình hợp dịch Pinnacle.exe để chạy lại các ví dụ về lệnh gộp có trong chương này.
7. Dùng chương trình hợp dịch Pinnacle.exe để chạy lại các ví dụ về chương trình con có trong chương này.
8. So sánh chương trình, lệnh gộp và chương trình con. Lập trình bằng lệnh gộp và chương trình con có ưu điểm và nhược điểm gì? Nêu ứng dụng của từng trường hợp.

Chương 3

NHÓM LỆNH ĐIỀU KHIỂN CHƯƠNG TRÌNH

Mục đích của chương:

- Xét nhóm lệnh nhảy chương trình không và có điều kiện.
- Xét nhóm lệnh gọi chương trình con.
- Xét vòng lặp.
- Xét gọi chương trình con có điều kiện.
- Xét một số cấu trúc điển hình của chương trình.

Trong các chương trình ngắn và đơn giản, chương trình là chuỗi liên tiếp các lệnh và thực hiện từ trên xuống dưới (TOP-DOWN-đầu-cuối) hay nói khác đi, giá trị của thanh ghi đếm chương trình PC thay đổi tăng dần từ 00H cho tới khi kết thúc chương trình.

Ở chương này, chúng ta sẽ xét các chương trình diễn ra không theo thứ tự trên, mà:

- Nhảy (hay bỏ qua) một số lệnh, tới một nhãn có một địa chỉ mới. Địa chỉ nhớ này không nằm tiếp theo địa chỉ của lệnh trước đó. Đây là các lệnh rẽ nhánh hay nhảy chương trình không điều kiện.

- Với một điều kiện nào đó xảy ra, chương trình sẽ nhảy (hay bỏ qua) một số lệnh, tới một nhãn có một địa chỉ mới. Địa chỉ nhớ này không nằm tiếp theo địa chỉ của lệnh trước đó. Đây là các lệnh rẽ nhánh hay nhảy chương trình có điều kiện.

- Với một điều kiện nào đó xảy ra, chương trình sẽ nhảy trở về một địa chỉ trước đó hay lặp lại một số lệnh, tạo thành một vòng lặp (LOOP). Đây là các lệnh rẽ nhánh hay nhảy chương trình có điều kiện để trở về một trong các lệnh đã thực hiện, có thể lặp lại một số lệnh một số lần vô hạn hay hữu hạn.

- Gọi một chương trình khác (chương trình con) không có và có điều kiện ở một địa chỉ nhớ khác, rồi lại tiếp tục trở về thực hiện tiếp chương trình chính.

Qua nghiên cứu các lệnh rẽ nhánh, chúng ta xét các loại cấu trúc cơ bản của một chương trình.

Còn trường hợp, khi có một điều kiện nào đó xảy ra, việc thực hiện chương trình chính bị ngắt quãng, để thực hiện chương trình phục vụ ngắt, sau đó lại trở về thực hiện tiếp chương trình chính (lệnh RETI) sẽ được nghiên cứu ở chương 10 về điều khiển ngắt của vi điều khiển.

3.1. CÁC LỆNH RẼ NHÁNH CHƯƠNG TRÌNH KHÔNG ĐIỀU KIỆN

Thông thường, trong một chương trình đơn giản, các lệnh được thực hiện từ trên xuống dưới (top-down), theo thứ tự tăng dần của thanh ghi đếm lệnh PC (Program Counter). Các lệnh rẽ nhánh hay điều khiển việc thực hiện chương trình sẽ tác động hay thay thế thanh ghi đếm lệnh PC bằng một giá trị địa chỉ nào đó mà không có điều kiện kèm theo nào, để chương trình bỏ qua một số lệnh (vì PC thay đổi đột ngột). Giá trị PC mới có thể lớn hơn PC cũ, chương trình sẽ bỏ qua một số lệnh hay PC mới có thể nhỏ hơn PC cũ hay chương trình sẽ lặp lại một số lệnh.

Bảng 3-1 giới thiệu các lệnh rẽ nhánh (nhảy, gọi chương trình con không điều kiện).

Theo bảng 3-1, có hai lệnh rẽ nhánh không điều kiện là:

- *Lệnh nhảy JMP*: chuyển tới địa chỉ khác hay PC được nạp giá trị khác, lệnh nhảy tới có thể nằm sau lệnh hiện tại (bỏ qua một số lệnh) hay nhảy trở về lệnh cũ đã thực hiện, để lập lại một nhóm lệnh.

- *Lệnh gọi chương trình con CALL chương_trình_con*: PC gửi vào bộ nhớ ngăn xếp (có địa chỉ ở SP tăng dần) bởi lệnh PUSH PC, chuyển tới địa chỉ của chương trình con để chạy chương trình con và khi gặp lệnh RET khi kết thúc chương trình con, lại trở về chính chỗ gọi chương trình con, PC nạp lại giá trị cũ lúc gọi chương trình con bởi lệnh POP PC (SP giảm dần) để tiếp tục thực hiện chương trình cũ đã bị gián đoạn.

Bảng 3-1. Các lệnh rẽ nhánh không điều kiện

Lệnh	Hành động	Thời gian thực hiện (μ s)
JMP addr.	Nhảy tới lệnh ở ô nhớ có địa chỉ addr.	2
JMP @A+DPTR	Nhảy tới lệnh ở ô nhớ có địa chỉ A+DPTR	2
CALL addr.	Gọi chương trình con ở địa chỉ addr.	2
RET	Trở về từ chương trình con	2
RETI	Trở về từ chương trình con phục vụ ngắt	2
NOP	Không hành động	1

Chú ý: - Lệnh NOP: không hành động gì, chỉ tốn thời gian 1 chu trình máy (1 μ s).

- Lệnh RET: kết thúc chương trình con, trở về chỗ bị ngắt của chương trình chính.

- Lệnh IRET: kết thúc chương trình con phục vụ ngắt, trở về chỗ bị ngắt (để thực hiện chương trình phục vụ ngắt) của chương trình chính.

3.1.1. Các lệnh nhảy không điều kiện JMP addr

Lệnh nhảy không điều kiện này sẽ nhảy tới thực hiện lệnh nằm trong ô nhớ có địa chỉ ghi trực tiếp sau lệnh, dưới dạng một con số hay một nhân *addr* chỉ địa chỉ của lệnh.

Trong bảng 3-1 chỉ có một lệnh JMP addr. Nhưng thực chất có ba lệnh nhảy tới địa chỉ *addr*, tùy theo số bit của địa chỉ *addr* của ô nhớ chứa lệnh đầu tiên.

1. SJMP addr

Đây là *lệnh nhảy ngắn* (S-Short-ngắn), tương đối, địa chỉ nhảy tới có 8 bit (lệnh 2 byte) tức vùng địa chỉ trong phạm vi -128D tới +127D byte tương đối so với giá trị hiện hành của thanh ghi đếm chương trình PC. Lệnh này còn được dùng dưới dạng SJMP \$ hay nhảy tại chỗ, dùng để kết thúc chờ của một chương trình.

2. AJMP addr

Đây là *lệnh nhảy tuyệt đối* (A-Absolute-tuyệt đối) mà địa chỉ gồm 11 bit (lệnh 2 byte), vùng địa chỉ trong phạm vi 2 kByte.

3. LJMP addr

Đây là *lệnh nhảy dài* (L-Long-dài), tuyệt đối mà địa chỉ gồm 16 bit (lệnh 3 byte), vùng địa chỉ trong phạm vi 64 kB.

Cả ba lệnh nhảy trên có đặc điểm chung là: bộ đếm chương trình PC sẽ được thay bằng giá trị *addr* (giá trị địa chỉ nhảy tới) và chương trình sẽ tiếp tục thực hiện từ lệnh có địa chỉ *addr*.

Các lệnh nhảy chương trình này được dùng trong các trường hợp sau:

- *LJMP addr*: *addr* có giá trị 30H hay 100H để bỏ qua vùng nhớ của các vector ngắt của VDK và khai báo dữ liệu để thực hiện chương trình chính nằm ở sau vector ngắt (ví dụ 3-1).

- *AJMP MAIN*: nhảy về lệnh bắt đầu chương trình (ký hiệu là nhãn MAIN hay START) để lập lại chương trình nhiều lần, để quan sát việc thực hiện chương trình trên đèn chỉ thị hay trên dao

động ký (ví dụ 3-1). Trước đó, phải xác định địa chỉ nhãn bắt đầu chương trình bằng chỉ dẫn biên dịch **ORG** **DIACHI**, thường **DIACHI** là 00H (cho chương trình đơn giản không có ngắt và nhớ đệm) hay 30H, hay 100H (chương trình có ngắt và khai báo dữ liệu).

- **SJMP \$**: hay **SJMP HERE** (ví dụ 3-2) nhảy về chính nó để dừng hay kết thúc một chương trình.

4. Lệnh nhảy **JMP @A+DPTR**

Lệnh **JMP @A+DPTR** là lệnh nhảy tới một trong các thành phần của một bảng mà các địa chỉ là tổng số của thanh ghi A 8 bit với thanh ghi **DPTR** 16 bit (**A+DPTR**) trong vùng 64 kByte vì (PC) thay bằng [(A)+(DPTR)].

*; Ví dụ 3-1: Chương trình dùng lệnh **LJMP addr.** và **AJMP addr.** có vòng lặp vô hạn*

ORG 00H	; bắt đầu sau khi bật nguồn nuôi hay ấn nút RESET
LJMP MAIN	; nhảy dài tới nhãn MAIN:
ORG 100H	; bắt đầu chương trình MAIN
MAIN: MOV A, # 55H	; nạp giá trị 55h vào A
MOV P0, A	; gửi A ra cổng P0
MOV P1, A	; gửi A ra cổng P1
MOV P2, A	; gửi A ra cổng P2
SJMP MAIN	; trở về MAIN
END	; kết thúc chương trình dịch

*; Ví dụ 3-2: Chương trình dùng lệnh **JMP addr.** có kết thúc chờ*

ORG 00H	
LJMP START	; nhảy tới START tại 30H
ORG 30H	; bỏ vùng nhớ cho các vectơ ngắt
START: MOV A, #55H	; ghi #55h vào A
MOV P0, A	; đưa A ra cổng P0
MOV P1, A	; đưa A ra cổng P1
MOV P2, A	; đưa A ra cổng P2
SJMP \$	; hay HERE: SJMP HERE
END	; kết thúc dịch

*; Ví dụ 3-3: Chương trình dùng lệnh **JMP @A+DPTR***

ORG 00H	
MOV A, #00H	; xóa thanh ghi A
MOV DPTR, # JMP_TBL	; nạp nhãn JMP_TBL vào DPTR
JMP @A+DPTR	; đọc bảng
JMP_TBL: AJMP LABEL0	; đọc thành phần thứ 0 của bảng
AJMP LABEL1	; đọc thành phần thứ 1 của bảng
AJMP LABEL2	; đọc thành phần thứ 2 của bảng
AJMP LABEL3	; đọc thành phần thứ 3 của bảng
SJMP \$	
END	

Ví dụ 3-3 dùng lệnh **JMP @A+DPTR** để đọc bảng có địa chỉ **#JMP_TBL**. Khi đó chương trình sẽ lần lượt nhảy tới các nhãn **LABEL0**, **LABEL1**, **LABEL2**, **LABEL3** là các thành phần của bảng.

Theo ví dụ trên, A có giá trị chẵn lần lượt là 0, 2, 4, 6. Nếu A có giá trị là 04H thì lệnh nhảy trở tới lệnh **AJMP LABEL2**, tức chương trình sẽ nhảy tới nhãn **LABEL2**.

Lệnh này, dùng để tạo một thực đơn của chương trình, tương tự các lệnh **CJNE** và **MOV @A+DPTR** sẽ xét sau.

3.2. LỆNH GỌI (CALL) VÀ TRỞ VỀ (RET, RETI) TỪ CHƯƠNG TRÌNH CON

3.2.1. Lệnh gọi chương trình con CALL addr.

Trong bảng 3-1 chỉ có một lệnh CALL addr. Nhưng thực chất là hai lệnh:

- **ACALL addr.:** lệnh gọi chương trình con ở gần, tuyệt đối, có địa chỉ của lệnh đầu tiên là addr có 11 bit (trong vùng 2 kbyte).

- **LCALL addr.:** lệnh gọi chương trình con ở xa (L), gọi chương trình con có địa chỉ của lệnh đầu tiên là addr có 16 bit (trong vùng 64 kbyte).

Các lệnh gọi CALL addr. tuyệt đối (A) hay xa (L) này sẽ gọi chương trình con có lệnh đầu tiên ở địa chỉ addr. của bộ nhớ.

Thông thường, khi viết chương trình, người ta thay địa chỉ addr. bằng tên nhãn (hay tên chương trình con). Khi hợp dịch, chương trình hợp dịch sẽ tìm địa chỉ addr. của nhãn để thay vào vị trí của nhãn.

Ví dụ 3-4 là một chương trình có gọi chương trình con tạo trễ dùng lệnh DJNZ Ri, rel. (xét ở dưới).

Chú ý: Chương trình con có thể nằm trong đoạn nhớ khác với khai báo ORG addr khác.

; Ví dụ 3-4: Chương trình có gọi chương trình con tạo trễ	
ORG 00H	
LJMP MAIN	
ORG 100H	
MAIN: MOV P0, #55H	; đưa ra cổng P0 giá trị #55H
CALL DELAY	; gọi chương trình con tạo trễ
MOV P1, #0AAH	; đưa ra cổng P0 giá trị #AAH
CALL DELAY	; gọi chương trình con tạo trễ
MOV P2, #55H	; đưa ra cổng P2 giá trị #00H
CALL DELAY	; gọi chương trình con tạo trễ
LJMP MAIN	; trở về MAIN
DELAY: MOV R0, # 100	; nạp R0 giá trị 10D
DELY: MOV R1, # 250	; nạp R1 giá trị 25D
NHAN0: DJNZ R0, NHAN0	; giảm R0 và nhảy tại chỗ
DJNZ R1, DELY	; giảm R1, nhảy về DELAY
RET	; kết thúc chương trình con
END	; kết thúc hợp dịch

3.2.2. Lệnh RET (trở về từ chương trình con) hay RETI (trở về từ chương trình phục vụ ngắt)

Cuối mỗi chương trình con, đều có lệnh RET (RETurn – trở về hay RETI – trở về từ ngắt) để kết thúc chương trình con, trở về chỗ bị ngắt của chương trình chính, tức địa chỉ có lệnh gọi chương trình con (CALL Tên chương_ trình_ con) hay chương trình phục vụ ngắt. Khi thực hiện các lệnh RET hay RETI, VDK nạp lại giá trị của PC, PSW và Acc lúc gọi chương trình con (hay chương trình phục vụ ngắt) để VDK tiếp tục chạy chương trình chính. Cặp lệnh (CALL Tên chương_ trình_ con) và RET luôn đứng ở đầu và cuối các lệnh của một chương trình con.

3.3. CÁC LỆNH Rẽ NHÁNH CHƯƠNG TRÌNH THEO ĐIỀU KIỆN CỦA BIT

Khác với các lệnh rẽ nhánh không điều kiện ở trên, các lệnh rẽ nhánh này chỉ rẽ nhánh khi có điều kiện nào đó xảy ra. Điều kiện ở đây là giá trị của các bit CY hay Z của thanh ghi trạng thái PSW hay giá trị của một trong các bit của cổng vào Pi. nào đó.

Các lệnh này tạo vòng lặp một số lần, số lần lặp lại do chương trình quyết định.

Bảng 3-2 giới thiệu các lệnh rẽ nhánh có điều kiện.

Ở chương trước, chúng ta đã thấy VDK có thanh ghi trạng thái PSW để chỉ trạng thái VDK với các bit: - *CY (PSW.7)*: cờ nhớ, bằng 1 khi phép toán số học (cộng, trừ, nhân, chia) trong thanh ghi A có nhớ.

- *ZY*: cờ Zero, bằng 1 khi $A = 0$ (tuy PSW không có bit cờ Z như trong vi xử lý 8086, nhưng khi giá trị của thanh ghi A bằng 0 ta vẫn có $Z = 1$ và $A \neq 0$, bit $Z = 0$). Từ đó, có các lệnh rẽ nhánh hay nhảy chương trình với điều kiện của các bit C và Z.

Ngoài ra, trong VDK còn có các lệnh rẽ nhánh chương trình theo giá trị bit của cổng vào Pi nào đó (xét ở chương 6).

3.3.1. Các lệnh nhảy theo giá trị bit cờ trạng thái CY (CY-CARRY)

Các lệnh nhảy này sẽ thực hiện nếu có bit cờ nhớ C (Carry) bằng 0 hay bằng 1. Nói khác đi, phép toán số học trong thanh ghi A không có nhớ ($C = 0$) hay có nhớ ($C = 1$).

Có hai loại lệnh rẽ nhánh theo điều kiện của bit C, là:

- *JNC rel.*: nhảy tới địa chỉ tương đối rel. nếu $C = 0$.
- *JC rel.*: nhảy tới địa chỉ tương đối rel. nếu $C = 1$.

Bảng 3-2. Các lệnh rẽ nhánh có điều kiện

Lệnh	Hành động	Phương pháp địa chỉ				Thời gian thực hiện (μs)
		Tức thì	Trực tiếp	Gián tiếp	Thanh ghi	
JZ rel	Nhảy nếu $A = 0$	Chỉ cho thanh ghi A, địa chỉ tương đối				2
JNZ rel	Nhảy nếu $A \neq 0$	Chỉ cho thanh ghi A, địa chỉ tương đối				2
JC rel	Nhảy tới địa chỉ rel nếu $C = 1$	Chỉ cho thanh ghi A, địa chỉ tương đối				2
JNC rel	Nhảy tới địa chỉ rel nếu $C = 0$	Chỉ cho thanh ghi A, địa chỉ tương đối				2
JB bit, rel	Nhảy tới địa chỉ rel nếu bit = 1	Chỉ cho bit, địa chỉ tương đối				2
JNB bit, rel	Nhảy tới địa chỉ rel nếu bit = 0	Chỉ cho bit, địa chỉ tương đối				2
JBC	Nhảy tới địa chỉ rel và xóa bit về 0 nếu bit = 1	Chỉ cho bit, địa chỉ tương đối				2
DJNZ <byte>, rel	Giảm <byte> và nhảy tới rel. nếu chưa bằng 0.	X		X		2
CJNE A, <byte>, rel	Số sánh A với <byte>, nhảy nếu chưa bằng.	X			X	2
CJNE <byte>, #data, rel	Số sánh <byte> với #data, nhảy nếu chưa bằng.		X	X		2

3.3.2. Lệnh nhảy theo giá trị của bit Z ($Z = 1$ khi $A = 0$)

Các lệnh nhảy này sẽ thực hiện nếu giá trị của thanh ghi A bằng 0 ($Z = 1$) hay A bằng 1 ($Z = 0$).

Có hai loại lệnh rẽ nhánh theo điều kiện của bit Z là:

- *JNZ rel.*: nhảy tới địa chỉ tương đối rel. nếu A khác 0.
- *JZ rel.*: nhảy tới địa chỉ tương đối rel. nếu A bằng 0.

Ví dụ 3-5 minh họa chương trình rẽ nhánh theo điều kiện của bit Z.

3.3.3. Lệnh giảm và nhảy nếu khác 0 DJNZ <byte>, <rel.addr.>

Lệnh này giảm <byte> đi 1 và nhảy tới địa chỉ <rel.addr> nếu giá trị <byte> chưa bằng 0.

Thực chất lệnh này gồm hai lệnh *giảm <byte> tới 0* (D <byte>) và nhảy tới <rel.addr.> nếu không (chưa) bằng 0 (*JNZ*).

Lệnh này đại diện cho hai lệnh, cụ thể là :

- *DJNZ Ri, rel* (2 byte): giảm thanh ghi Ri đi 1, nhảy tới địa chỉ rel. khi Ri chưa bằng 0.

- *DJNZ direct, rel*: giảm nội dung ô nhớ có địa chỉ trực tiếp direct đi 1 và nhảy tới địa chỉ rel. khi nội dung ô nhớ chưa bằng 0.

; Ví dụ 3-5: Chương trình rẽ nhánh theo điều kiện của C và Z

```

ORG 00H
MOV A, #15D      ; nạp vào A số 15 thập phân
MOV B, #4D       ; nạp vào B số 4 thập phân
NHAN1: SUBB A, B  ; trừ 15D-4D
MOV P0, A        ; đưa hiệu số ở A ra cổng P0
MOV R0, A        ; gửi hiệu số ở A vào R0
JNC NHAN1        ; nhảy tới NHAN1 nếu C = 0 (trừ chưa nhớ)
MOV B, R0        ; gửi hiệu số (số âm) vào B
ADD A, B         ; hồi phục lại hiệu số cuối
NHAN2: MOV P1, A  ; đưa ra cổng P1 giá trị của A
DEC A           ; giảm hiệu số cuối ở A đi 1 đơn vị
JNZ NHAN2       ; nhảy tới NHAN2 nếu A chưa bằng 0
SJMP $         ; kết thúc chờ, khi A bằng 0
END            ; kết thúc chương trình dịch

```

Các giá trị địa chỉ rel. thường đứng trước địa chỉ của lệnh DJNZ do đó, lệnh DJNZ tạo một vòng lặp với số lần lặp là giá trị của Rn hay ô nhớ có địa chỉ direct (trực tiếp). Người ta dùng lệnh này để tạo thời gian trễ với độ trễ khác nhau:

- *Trễ có thời gian nhỏ*: như lệnh NHAN0: DJNZ R0, NHAN0 ở ví dụ 3-7, R0 giảm đi 25D lần, tức 25×2 chu trình = 50 μ sec

- *Trễ có thời gian lớn*: dùng hai vòng trễ lồng nhau như chương trình con tạo trễ DELAY trong ví dụ 3-5, và phần vòng lặp.

; Ví dụ 3-6: Chương trình tạo trễ dùng lệnh DJNZ

```

ORG 00h
MOV R0, #10      ; nạp R0 giá trị 10D
DELY: MOV R1, #25 ; nạp R1 giá trị 25D
NHAN0: DJNZ R0, NHAN0 ; giảm R0 và nhảy tại chỗ
      DJNZ R1, DELY  ; giảm R1, nhảy về DELY nếu khác 0
      END           ; kết thúc chương trình hợp dịch

```

Thời gian trễ được tính như sau:

25×10 chu trình + 1 (chu trình lệnh MOV Rn, #10) + 2 (chu trình lệnh NHAN0: DJNZ R0, NHAN0) + 2 (chu trình lệnh DJNZ R1, DELAY) + 2 (chu trình lệnh RET) = 250 + 7 = 257 chu

trình hay $257 \times 1,085 \mu \text{sec} \approx 260 \mu \text{sec}$. Muốn tăng thời gian trễ, ta có thể cho Rn, R1 tối đa là 255, tức thời gian trễ bằng $255 \times 255 + 7 = (65535 + 7) \times 1,085 \approx 70 \text{ msec}$. Muốn tăng lên nữa, ta phải thêm vòng lặp để có nhiều chu trình trễ.

Để có thời gian tạo trễ chính xác, người ta thêm lệnh NOP (1 chu trình) vào các vòng lặp.

3.3.4. Lệnh so sánh và nhảy nếu không bằng CJNE <dest-byte>, <src-byte>, <rel.addr.>

Lệnh này so sánh hai byte đích và nguồn, nếu chúng không bằng nhau, sẽ nhảy tới địa chỉ <rel.addr.>. Do đó, lệnh này thực chất là hai lệnh: so sánh (Compare) <dest-byte> (trong đó có A) với <src-byte> và nhảy tới địa chỉ <rel.addr.> nếu kết quả so sánh không bằng (CJNE).

Cú pháp trên là đại diện cho 4 lệnh với các phương pháp địa chỉ của toán hạng đích và nguồn khác nhau (bảng 3-2). Đó là các lệnh:

- CJNE A, direct, rel: (3 byte, 2 chu trình) so sánh thanh ghi chứa A với giá trị của ô nhớ có địa chỉ trực tiếp direct. Nếu không bằng nhau, nhảy trở về địa chỉ rel, nếu bằng nhau, thực hiện lệnh tiếp sau lệnh CJNE.

- CJNE A, #data, rel: so sánh thanh ghi chứa A với giá trị tức thì #data, nếu không bằng nhau, nhảy tới địa chỉ rel. Nếu bằng nhau, thực hiện lệnh tiếp theo lệnh này trong chương trình.

- CJNE Rn, #data, rel: so sánh thanh ghi Rn với giá trị tức thì #data, nếu không bằng nhau, nhảy tới địa chỉ rel. Nếu bằng nhau, thực hiện lệnh tiếp theo lệnh này trong chương trình.

- CJNE @Ri, #data, rel: so sánh ô nhớ có địa chỉ là nội dung của thanh ghi Ri với giá trị tức thì #data, nếu không bằng nhau, nhảy tới địa chỉ rel. Nếu bằng nhau, thực hiện lệnh tiếp theo lệnh này trong chương trình.

Lệnh CJNE này dùng để :

- Chờ một giá trị mong muốn #data gõ từ bàn phím: như lệnh (HERE: CJNE A, #data, HERE).
- Tìm và đọc ra giá trị cho trước: như #data trong một khối dữ liệu (ví dụ 3-7).

; Ví dụ 3-7: Chương trình dùng CJNE

ORG 00H	
MOV A, #05H	; nạp giá trị 05h vào A
NHAN: INC A	; tăng A lên 1
CJNE A, #10H, NHAN	; so A với 10H, nếu A < 10H, về NHAN
MOV P0, A	; khi A = 10H, gửi a ra cổng P0
SJMP \$	; kết thúc bằng chờ tại chỗ
END	; kết thúc dịch

- Chọn một trong hai hay nhiều giá trị #data: như trong thực đơn (ví dụ 3-8 và 3-9).

Các lệnh rẽ nhánh có điều kiện này được dùng để thoát khỏi vòng lặp (xem ví dụ ở vòng lặp) và tạo nên các cấu trúc có điều kiện của chương trình (IF ... then ... else ...) hay gặp trong các giải thuật của chương trình.

- So sánh để rẽ nhánh chương trình tiếp: với việc dùng thêm lệnh nhảy có điều kiện như JC, JNC (ví dụ 3-8). Trong ví dụ này, chương trình đọc giá trị của cổng vào P1 vào thanh ghi A và lựa chọn một trong ba phương án :

- + nếu A < 75 thì ghi vào R1;
- + nếu A = 75 thì đưa ra hiển thị ở cổng P2;
- + nếu A > 75 thì ghi vào R2.

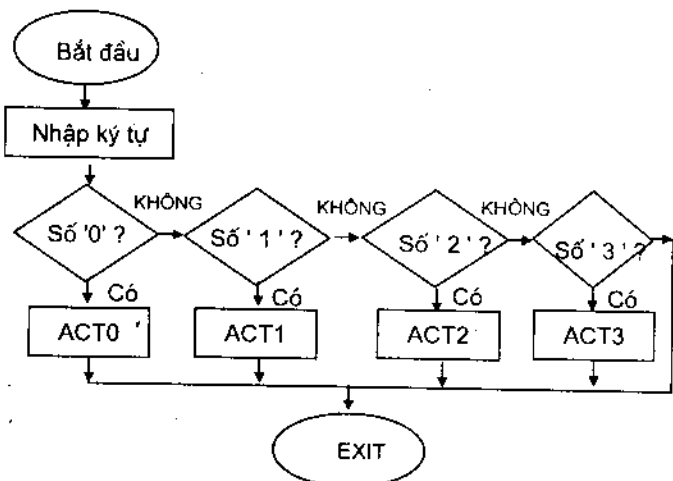
Trong ví dụ 3-9, lệnh CJNE A, #data dùng để lựa chọn một trong 4 phương án (ACT0, ACT1, ACT2, ACT3) tùy theo ký tự nhập vào là 0, 1, 2, 3 (giải thuật như hình 3-1). Còn lệnh JMP @A+DPTR cũng chọn từ một bảng cho sẵn. Với bảng này, ta có thể bổ sung vào tùy ý mà không chọn từ bàn phím.

3.3.5. Các lệnh rẽ nhánh chương trình theo giá trị của bit

Trong các thanh ghi bit hay cổng vào Pi, từng bit có thể xác lập (SETB) hay xóa (CLR bit) một cách riêng rẽ và có các lệnh nhảy hay rẽ nhánh chương trình tùy theo giá trị của các bit này.

Đó là các lệnh:

- JB bit, rel: chương trình nhảy tới địa chỉ rel nếu giá trị bit = 1.
- JNB bit, rel: chương trình nhảy tới địa chỉ rel nếu giá trị bit = 0.
- JBC bit, rel: chương trình nhảy tới địa chỉ rel nếu giá trị bit = 1 và xóa bit về 0.



Hình 3-1. Giải thuật của thực đơn của ví dụ 3-9

; Ví dụ 3-8: Chương trình rẽ nhánh dùng CJNE và JNC

```

ORG 00H
LJMP MAIN          ; nhảy tới chương trình chính
ORG 30H
MAIN: MOV P1, #OFFH ; khởi tạo cổng P1 làm cổng vào
      MOV A, P1      ; đọc cổng vào P1 vào A
      CJNE A, #75, OVER ; so sánh với #75, nếu không, tới OVER
      SJMP EXIT      ; nhảy tới EXIT
OVER: JNC NEXT       ; nếu CY = 0 khi A > 75 thì nhảy tới NEXT
      MOV R1, A       ; nếu CY = 1 khi A < 75 thì lưu A vào R1
      SJMP EXIT      ; nhảy tới EXIT
NEXT: MOV R2, A       ; lưu A vào R2 nếu > 75
EXIT: MOV P2, A       ; đưa hiển thị ra cổng P2 nếu A = 75
      SJMP $          ; kết thúc chờ
      END             ; kết thúc chương trình dịch
  
```

Chú ý: Có thêm lệnh rẽ nhánh tiếp JNC NEXT vì có ba phương án lựa chọn.

Các lệnh trên được dùng để rẽ nhánh chương trình khi có:

- bit cờ được xác lập: ví dụ, bit cờ tràn của Timer/Counter0, 1, 2 hay các bit yêu cầu ngắt của trao đổi tin nối tiếp (RI, TI).

Ta có thể chờ bit TF0 xác lập và sau đó xóa nó, ta dùng lệnh JBC TF0, HERE thay cho lệnh chờ và xóa CLR TF0.

- bit của cổng vào Pi: khi đọc cổng vào, đặc biệt là các lối vào /INT0, /INT1, /INT2, có thể có rẽ nhánh hay không rẽ nhánh chương trình tùy theo giá trị của bit đọc vào. Người ta dùng tính chất này để điều khiển (cưỡng bức) VĐK thực hiện hay không thực hiện một chương trình nào đó. Ở chương 10 về ngắt, chúng ta sẽ xét các ứng dụng này.

Ví dụ 3-9: Thực đơn dùng lệnh CJNE (cấu trúc chặt chẽ)

ORG 00h	
CALL INCH	; gọi chương trình đọc ký tự từ bàn phím
CJNE A, # '0', SKIP1	; so sánh với số '0'
ACT0:	; nhãn hình thức, không dùng
	; khối lệnh cho hành động 0
JMP EXIT	; nhảy tới EXIT
SKIP1: CJNE A, # '1', SKIP2	; so sánh với số '1'
ACT1:	; nhãn hình thức, không dùng
	; khối lệnh cho hành động 1
JMP EXIT	; nhảy tới EXIT
SKIP2: CJNE A, # '2', SKIP3	; so sánh với số '2'
ACT2:	; nhãn hình thức, không dùng
	; khối lệnh cho hành động 2
JMP EXIT	; nhảy tới EXIT
SKIP3: CJNE A, # '3', EXIT	; so sánh với số '3'
ACT3:	; nhãn hình thức, không dùng
	; khối lệnh cho hành động 3
EXIT: SJMP \$	
END	

Tương tự, ta có thể dùng lệnh JBC bit để xóa bit và rẽ nhánh.

Nếu thiết kế thực đơn, người ta hay dùng lệnh CJNE, còn lập và đọc bảng, người ta dùng lệnh JMP @A+DPTR vì có thể mở rộng được bảng một cách dễ dàng.

3.4. LỆNH GỘP ĐIỀU KHIỂN CHƯƠNG TRÌNH CÓ ĐIỀU KIỆN KÉP

Hệ lệnh của VĐK 8051/8052 chỉ có các lệnh nhảy chương trình có một điều kiện, không có các lệnh nhảy chương trình có điều kiện kép hay hai điều kiện xảy ra đồng thời (ví dụ, nhảy nếu lớn hơn hoặc bằng) hay gọi chương trình con có điều kiện kép, cũng như các lệnh lặp có điều kiện kép. Do đó, chúng ta phải viết các lệnh gộp có điều kiện kép để bổ sung vào những lệnh trên, làm cho bộ lệnh của VĐK cũng phong phú như bộ lệnh của vi xử lý của máy vi tính PC.

Với hệ lệnh nhảy có một điều kiện, các lệnh gộp điều khiển chương trình có điều kiện kép này và các lệnh gộp ứng dụng sẽ tự viết ở các chương sau, chúng ta có thể lập trình một cách dễ dàng cho VĐK bằng lệnh gộp, để chương trình ngắn gọn, gần với ngôn ngữ bậc cao, giải phóng cho người lập trình và người sử dụng khỏi những chi tiết và phức tạp của hợp ngữ.

3.4.1. Lệnh gộp nhảy chương trình với điều kiện kép

VĐK 8051/8052 không có lệnh nhảy chương trình với điều kiện kép hay hai điều kiện đồng thời phù hợp nhau (lớn hơn hay bằng GT hay $>=$, bằng hay nhỏ hơn LT hay $<=$, khác nhau ($<>$) v.v...)

Ngay điều kiện IF <biểu thức> với chỉ dẫn biên dịch:

IF <biểu thức> THEN phát biểu 1, ELSE phát biểu 2.

trong đó <biểu thức> chỉ có một giá trị là lớn hơn ($>$), bằng ($=$) hay nhỏ hơn ($<$).

Ở đây, chúng ta sẽ viết các lệnh gộp nhảy chương trình sẽ thỏa mãn các điều kiện kép tức là cả hai điều kiện, ví dụ, GT (Greater Than - lớn hơn), tức lớn hơn ($>$) hay bằng ($=$).

Chúng ta dựa vào các lệnh gộp của chương trình dịch lệnh gộp MCS-51 (bảng 3-3) để viết các lệnh gộp tương ứng.

Bảng 3-3 giới thiệu hệ lệnh gộp có điều kiện kép sẽ xây dựng theo định nghĩa của lệnh gộp.

a) Lệnh gộp nhảy chương trình có điều kiện đơn

Nguyên tắc viết lệnh gộp nhảy có điều kiện đơn JE, JNE, JLT, JGT trên là:

- Dùng thanh ghi chứa A để nạp giá trị cần so sánh với thông số <value?>.
- Dùng lệnh so sánh và nhảy "CJNE A, #1, \$+3" để phát hiện A khác giá trị của thông số <#value> đưa vào và nhảy tới lệnh tiếp theo (có địa chỉ \$+3).
- Dùng lệnh nhảy có điều kiện "JC/JNC nhan", "JZ/JNZ nhan" để phát hiện tiếp $A < \#value$ hay $A > \#value$.
- Dùng lệnh so sánh và chờ "CJNE A, #1, S" để chờ $A = \#value$ và "SJMP nhan" cho lệnh "JE nhan".
- "nhan": chứa lệnh thoả mãn điều kiện kép ở trên, có thể nằm ở bất kỳ vị trí nào trong chương trình.
- Lệnh tiếp theo sau lệnh gộp: là lệnh không thoả mãn điều kiện kép ở trên.

Ví dụ 3-10 JLE cho lệnh nhảy tới nhan nếu nhỏ hơn hay bằng.

Bảng 3-3. Hệ lệnh gộp có điều kiện kép

Ký hiệu	Tên lệnh	Hành động	MC-51
JE	Nhảy nếu bằng	Nhảy nếu $A = value$	$IF A = B$
JNE	Nhảy nếu khác	Nhảy nếu $A \neq value$	$IF A < > B$
JLT	Nhảy nếu nhỏ hơn	Nhảy nếu $A < value$	$IF A < B$
JGT	Nhảy nếu lớn hơn	Nhảy nếu $A > value$	$IF A > B$
JLE	Nhảy nếu nhỏ hay bằng	Nhảy nếu $A \leq value$	$IF A \leq B$
JGE	Nhảy nếu hơn hay bằng	Nhảy nếu $A \geq value$	$IF A \geq B$
JOR	Nhảy nếu ở ngoài dải	Nhảy nếu $valuenho < A$ hay $A > valuelon$	$IFA < valuenho$ hay $A > valuelon$
JIR	Nhảy nếu ở trong dải	Nhảy nếu $valuenho < A < valuelon$	$IFvaluenho < A$ và $A < valuelon$

b) Lệnh gộp nhảy chương trình có điều kiện kép

Nguyên tắc viết lệnh gộp nhảy có điều kiện kép JLE, JGE trên là dùng hai lệnh đều nhảy tới nhan, cụ thể là:

- Dùng lệnh so sánh đơn "JE nhan", "JLT nhan" hay "JGT nhan";
- Dùng lệnh nhảy tới nhan kết thúc trong lệnh gộp với điều kiện ngược với điều kiện đã cho (để không nhảy tới nhan khi xảy ra điều kiện ngược lại với điều kiện đã cho);
- Dùng thêm lệnh "SJMP nhan" cho điều kiện $A = \#value$.

Cụ thể, ta có các lệnh gộp sau:

- Lệnh JE: dùng lệnh so sánh và nhảy "CJNE A, #1, \$" dưới dạng thông số. Nếu giá trị A không bằng hay khác #value thì nhảy tại chỗ chờ. Nếu $A = \#value$ thực hiện lệnh tiếp theo là nhảy không điều kiện tới nhan (SJMP nhan).

- JGT NHAN: + Dùng lệnh so sánh và nhảy "CJNE A, #1, \$+3" tới lệnh tiếp theo.
+ Dùng lệnh "JNC nhan" để nhảy tới nhan nếu $A > \#value$.
- JLT NHAN: + Dùng lệnh so sánh và nhảy "CJNE A, #1, \$+3" tới lệnh tiếp theo.
+ Dùng lệnh "JC nhan" để nhảy tới nhan nếu $A > \#value$.
+ Dùng lệnh "JNC kt" để nhảy tới kt (nhan kết thúc lệnh gộp) nếu $A > \#value$.

- **JGE NHAN:** + Dùng lệnh so sánh và nhảy "CJNE A, #1, \$+3" tới lệnh tiếp theo.
 - + Dùng lệnh "JNC NHAN" để nhảy tới nhan nếu A > #value.
 - + Dùng lệnh "JC KT" để nhảy tới kt (nhãn kết thúc lệnh gộp) nếu A < #value.
 - + Dùng lệnh "SJMP NHAN" để nhảy tới nhan nếu A = #value
- **JLE NHAN:** + Dùng lệnh so sánh và nhảy "CJNE A, #1, \$+3" tới lệnh tiếp theo.
 - + Dùng lệnh "JNC NHAN" để nhảy tới NHAN nếu A < #value.
 - + Dùng lệnh "JNC KT" để nhảy tới KT (nhãn kết thúc lệnh gộp) nếu A > #value.
 - + Dùng lệnh "SJMP nhan" để nhảy tới nhan nếu A = #value.

c) Lệnh gộp nhảy chương trình có điều kiện kép so với dải giá trị

Dải giá trị đây là #valuenho và #valuelon trong đó #valuenho < #valuelon. Có hai trường hợp xảy ra là giá trị thanh ghi A ở ngoài dải (A < #valuenho hay A > #valuelon) hay giá trị thanh ghi A ở trong dải (#valuenho < A < #valuelon).

JOR: (A < valuenho hay A > valuelon), ở đây có hai sự so sánh riêng rẽ nhỏ hơn hay lớn hơn. Cả hai trường hợp, nếu thoả mãn, đều nhảy tới một "nhan" chung.

Ví dụ 3-11 cho lệnh nhảy JOR (A ở ngoài dải).

<i>; Ví dụ 3-10: Lệnh nhảy có điều kiện kép JLE</i>	<i>; Ví dụ 3-11: Lệnh nhảy có điều kiện kép ngoài dải JOR</i>
<pre>macro @JLE <value?> cjne a, #1, \$+3 ; A khác #value jc NHAN ; A > #value jnc KT ; nhảy tới KT nếu A . #value sjmp NHAN endmac org 00h @JLE 30H mov P0, #40H ; A < #value sjmp KT ; nhảy tới KT NHAN: ; A > #value mov a, #40H KT: sjmp \$; kết thúc chờ end</pre>	<pre>macro @JOR <valuenho?, valuelon?,> cjne a, #1, \$+3 ; A khác #valuenho jc NHAN ; A < #valuenho cjne a, #12, \$+3 ; A khác #valuelon jnc NHAN ; A > #valuelon jc KT ; A < #valuelon endmac org 00H @JOR 20H, 50H mov a, #0F0H ; trong dải mov P0, a sjmp KT ; kết thúc NHAN: Mov a, #00H ; ngoài dải mov P1, a KT: sjmp \$; kết thúc end</pre>

<i>; Ví dụ 3-12: Lệnh nhảy có điều kiện kép trong dải JIR</i>
<pre>macro @JIR <,valuenho? valuelon?,> cjne a, #12, \$+3 ; A khác #valuelon jnc NHAN1 ; A < #valuelon jc KT ; A > #valuelon NHAN1: cjne a, #1, \$+3 ; A khác #valuenho jc NHAN ; A < #valuenho jnc KT ; A > #valuenho endmac org 00H @JIR 50H, 20H mov a, #0F0H ; ngoài dải mov P0, a sjmp KT ; kết thúc NHAN: mov a, #00H ; trong dải mov P1, a KT: sjmp \$; kết thúc end</pre>

- **JIR**: Lệnh nhảy nếu ở trong dải ($valuenho < A < valuelon$), ở đây có hai sự so sánh liên tiếp (lớn hơn $valuenho$ và nhỏ hơn $valuelon$) và cùng thỏa mãn điều kiện để nhảy tới một "NHAN" chung.

Ví dụ 3-12 cho lệnh nhảy JIR (A ở trong dải).

Chú ý - Lệnh JIR phải so sánh liên tiếp hai lần với hai giá trị $valuelon?$ và $valuenho?$ để thỏa mãn đồng thời hai điều kiện " $valuenho? < A < valuelon?$ ".

- Tương tự, có thể viết lệnh JIER để thỏa mãn đồng thời hai điều kiện trong dải và cả giá trị giới hạn của dải, tức là $valuenho?$ lớn hơn hay bằng ($= < A$) và nhỏ hơn hay bằng ($=$) $valuelon?$. Dùng các lệnh "JGE $<valuenho?>$ nhan" và "JLE $<valuelon?>$ nhan" để so sánh liên tiếp hai điều kiện.

- Tương tự, có thể viết lệnh JOER để thỏa mãn một trong hai điều kiện ngoài dải và cả giá trị giới hạn của dải, tức là $valuenho?$ nhỏ hơn hay bằng ($= < A$) hay lớn hơn hay bằng ($=$) $valuelon?$ Khi đó, dùng các lệnh "JLE $<valuenho?>$ nhan" và "JGE $<valuelon?>$ nhan" để so sánh từng điều kiện một.

3.4.2. Viết lệnh gọi chương trình con có điều kiện

VĐK 8051/8052 không có lệnh gọi chương trình con có điều kiện. Do đó, muốn gọi chương trình con có điều kiện, ta phải dùng việc dịch có điều kiện lệnh gọi chương trình con (CALL chương_trình_con) hay dùng lệnh gộp nhảy tới chương trình con có điều kiện rồi nhảy trở về không điều kiện. Dưới đây, qua hai ví dụ 3-10 và 3-11, ta giới thiệu cách viết lệnh gộp và chương trình cho lệnh gộp gọi chương trình con GOILT theo hai cách khác nhau. Tương tự, dựa vào các lệnh nhảy có điều kiện, ta có thể viết được lệnh gọi chương trình con có điều kiện.

1. Dùng lệnh gọi (CALL) có điều kiện

Nhóm lệnh này cũng tương tự hướng dẫn dịch có điều kiện cho vòng lặp ở trên.

Ví dụ 3-13 giới thiệu nhóm lệnh gọi chương trình con "NHAN" với điều kiện của lệnh so sánh nếu nhỏ hơn ($cjne\ a, \#value, \$+3$ và $jc\ NHA$).

- Nếu điều kiện $A < 10H$ thì sẽ gọi chương trình con "NHAN"

- Nếu điều kiện $A > 10H$ thì sẽ nhảy tới thực hiện lệnh tại nhãn "NHA".

; Ví dụ 3-13: Chương trình GLT (gọi chương trình con dùng phép dịch lệnh gọi Call nhan)	
macro @GLT <value?>	; lệnh gộp gọi chương trình con nếu nhỏ hơn GLT
if	; nếu
cjne a, #10h, \$+3	; so sánh a với #10h, rồi nhảy tới lệnh jc có địa chỉ \$+3
	; (vì cjne có 3 byte)
jnc NHA	; nhảy tới "NHA" nếu lớn hơn (trừ không có nhớ)
call NHAN	; gọi NHAN nếu c = 1 hay nhỏ hơn (trừ có nhớ)
NHA: nop	; gọi chương trình con "NHA" nếu a > #10h
endif	; kết thúc nếu
endmac	; kết thúc định nghĩa lệnh gộp
org 00h	; chương trình chính
mov P0, #5ah	; đưa #5ah ra cổng P0
@gdk 20h	; lệnh gộp "gọi có điều kiện gdk" với a = 20h
mov p0, #50H	
sjmp \$	; kết thúc chờ
NHAN: mov P2, #0CCH	; chương trình con "NHAN", đưa #0CCH ra cổng P2
ret	; trở về chương trình chính
end	; kết thúc dịch

2. Dùng lệnh nhảy có điều kiện

Nếu dùng lệnh nhảy JLT ở trên và lệnh nhảy trở về không điều kiện, ta có thể tạo được lệnh gộp GLT như ở trên. Ví dụ 3-14 giới thiệu chương trình dùng lệnh gọi GOILT nếu nhỏ hơn (LT) dùng lệnh nhảy có điều kiện và trở về không điều kiện.

3.5. LỆNH GỘP CHO VÒNG LẶP

3.5.1. Tổng quát về vòng lặp

Vòng lặp là một số lệnh của chương trình được thực hiện lặp lại một số lần. Trong lập trình, đôi khi chương trình cần có vòng lặp để:

- Làm trễ việc thực hiện một hành động sắp tới.
- Lặp lại việc thực hiện một hành động đang diễn ra như lặp lại hiển thị để quan sát, lặp lại âm thanh cho tai nghe v.v...
- Lặp lại việc thực hiện một hành động cho nhiều thiết bị tương tự (như đưa hiển thị liên tiếp các bit cho các đèn chỉ thị LED rồi hay 7 đoạn v.v...).

1. Đặc điểm của vòng lặp

Một vòng lặp bao giờ cũng có:

- Một số lệnh chuẩn bị dữ liệu cho vòng lặp, nằm ở ngoài vòng lặp.
- Một nhãn tại lệnh đầu tiên của vòng lặp để chương trình trở về (không hay có điều kiện).
- Một lệnh nhảy trở về nhãn (không hay có điều kiện) ở cuối vòng lặp.
- Một biến đếm là giá trị một thanh ghi (Ri, A) để theo dõi số lần lặp, có giá trị giảm từ giá trị ban đầu R0 tới 0 ($Z = 1$) khi dùng lệnh (DJNZ Ri, nhãn) để kết thúc vòng lặp.
- Hay một biến đếm Ai thay đổi tới giá trị so sánh A0 khi dùng lệnh (CJNE Ai, A0, nhãn) để kết thúc vòng lặp.

; Ví dụ 3-14: Chương trình GOILT gọi chương trình con có điều kiện dùng lệnh nhảy

macro @GOILT <value?>	; lệnh gộp gọi chương trình con có điều kiện gdk
cjne a, #1, \$+3	; so sánh a với #10h, rồi nhảy tới lệnh jc có địa chỉ \$+3
	; (vì cjne có 3 byte)
jc NHA	; nhảy tới "NHA" nếu nhỏ hơn (trừ có nhớ)
endmac	; kết thúc định nghĩa lệnh gộp
org 00h	; bắt đầu chương trình chính
mov P0, #5ah	; đưa #5ah ra cổng P0
@GOILT 20h	; lệnh gộp "gọi có điều kiện gdk" với a = 20h
NHA: mov p0, #50H	; điểm trở về của lệnh sjmp
sjmp \$	; kết thúc chờ
NHA: mov P2, #0CCH	; chương trình con "nhân", đưa #0CCH ra cổng P2
sjmp NHA	; trở về chương trình chính
end	; kết thúc dịch

- Hay một biến đếm bit có giá trị 0 hay 1 để quyết định có trở về nhãn hay không khi dùng lệnh (JB bit nhãn) để nhảy tới nhãn nếu bit = 1, hay lệnh (JNB bit nhãn) để nhảy tới nhãn nếu bit = 0 hoặc (JBC bit nhãn) để nhảy tới nhãn và xóa bit nếu bit = 1. bit nhãn 0 để nhảy tới nhãn nếu bit = 1. Loại biến đếm bit này thường dùng để dừng hay cho phép hoặc tiếp tục vòng lặp.

- Một điểm thoát hay rời khỏi vòng lặp tại chỗ có lệnh kết thúc vòng lặp hay nhảy có điều kiện ở trên.

- Có thể kết thúc vòng lặp tự động (bằng biến đếm về 0, bằng biến đếm bit = 0 hay gặp giá trị 00H trong việc đọc chuỗi dữ liệu với lệnh nhảy JZ).

2. Các loại vòng lặp

Có các loại vòng lặp sau:

- *Vòng lặp vô hạn*: vòng lặp không có biến đếm và lệnh thoát, chỉ có lệnh nhảy không điều kiện trở về nhãn ở đầu vòng lặp. Loại vòng lặp này có thể dừng (hay kết thúc) hoặc tiếp tục vòng lặp nhờ nhận biết giá trị của bit điều khiển ở cổng vào VDK.

- *Vòng lặp hữu hạn*: vòng lặp có biến đếm, số lần lặp và lệnh thoát là lệnh nhảy có điều kiện ở cuối vòng lặp để trở về nhãn ở đầu vòng lặp.

3. Thoát khỏi vòng lặp

Kết thúc lặp hay thoát khỏi vòng lặp theo một trong bốn loại lệnh sau:

- Lệnh nhảy có điều kiện của bit trạng thái (JC, JNC, JZ, JNZ).
- Lệnh nhảy có điều kiện của bit (JB bit nhân, JNB bit nhân, JBC bit nhân).
- Lệnh giảm, so sánh và nhảy có điều kiện nếu thanh ghi R_i bằng 0 (DJNZ R_i, nhân).
- Lệnh so sánh và nhảy có điều kiện CJNE <byte>, <#data>, <rel addr.> (hay nhân) nếu <byte> không bằng giá trị cho trước #data.

3.5.2. Vòng lặp dùng lệnh gộp

Trong chương 2, chúng ta đã xét một số ví dụ về lệnh gộp là một vòng lặp, như:

- *Có thực hiện hay không thực hiện vòng lặp*: khi dịch hay không dịch đoạn chương trình có vòng lặp với công thức:

IF <biểu thức> (đoạn chương trình có vòng lặp) else (đoạn chương trình không có vòng lặp) lặp

- với :
- + *Biểu thức đúng*: dịch và chạy đoạn chương trình có vòng lặp sau IF.
 - + *Biểu thức sai*: dịch và chạy đoạn chương trình không có vòng lặp sau ELSE.
 - *Gán các nhãn với các chỉ số khác nhau*: nếu dùng nhiều lệnh gộp với nhãn có sẵn.
 - *Gán các nhãn khác nhau*: nếu dùng nhiều lệnh gộp chưa có nhãn.
 - *Gán số lần lặp khác nhau*: nếu dùng nhiều lệnh gộp chưa có số lần lặp.
 - Có thể viết các lệnh gộp mới cho vòng lặp và đếm.

Chúng ta có thể kể ra một số loại vòng lặp điển hình như: lặp lại vô hạn có điều khiển (dừng, tiếp tục), lặp lại hữu hạn (do biến đếm về 0 hay đọc số 00H), tìm dữ liệu mong muốn (con số, một hay chuỗi ký tự), phát chuỗi số theo quy luật mong muốn (giá trị đầu, bước, giá trị cuối) v.v.

1. Vòng lặp định thời gian

Người ta dùng vòng lặp vô hạn và có hạn để đếm thời gian cho đồng hồ.

Vòng lặp vô hạn (xem ví dụ 3-15 về đồng hồ) là vòng lặp:

- Không có sự thay đổi của biến đếm hay không có sự tăng hay giảm của thanh ghi.
- Có lệnh nhảy không điều kiện "SJMP NHAN" về nhãn ở cuối vòng lặp.

Vòng lặp vô hạn có điều khiển (xem ví dụ 3-14 về máy phát xung) là vòng lặp vô hạn khi có thêm:

- Lệnh JNB bit, \$: để dừng khi bit có giá trị 0 (tương đương lệnh Break ở ngôn ngữ bậc cao).
- Lệnh JB bit, \$+3: để tiếp tục khi bit có giá trị 1 (tương đương lệnh Continue ở ngôn ngữ bậc cao).

Ví dụ 3-15 giới thiệu chương trình đồng hồ đếm giây, phút, giờ dùng vòng lặp DELIS do các thanh ghi giảm dần với lệnh DJNZ. Các vòng lặp khác cho giây, phút, giờ là các vòng lặp tuần hoàn. Cuối mỗi vòng lặp cho phút (60 giây), giờ (60 phút) và ngày (24 giờ) đều có lệnh so sánh và nhảy CJNE A, #data (với data = 60 hay 24) để đếm tiếp ở đơn vị thấp hơn. Khi a = #data sẽ thoát khỏi vòng lặp để thực hiện tiếp chương trình, nghĩa là tiếp tục đếm thời gian ở hàng đơn vị cao hơn.

Các chỉ số về giây, phút và giờ được chỉ thị trên các cổng P2, P1, P0 bằng lệnh MOV P, R (xem chương 4).

Chú ý: - Chương trình trên có độ trễ DELIS chưa chính xác vì tính với tần số nhịp của VĐK là 12 MHz và chưa hiệu chỉnh độ trễ của các lệnh khác. Muốn đếm thời gian chính xác, phải dùng Timer để đếm thời gian (chương 8).

- Chương trình trên có thể viết thêm cho chục giây, chục phút và chỉ thị thành các thang đo nhỏ hơn.
- Chương trình trên có thể viết cho chỉ thị trên đèn LED 7 đoạn (xem tập II về ghép nối với đèn 7 đoạn).

2. Vòng lặp định thời gian trễ

a) Vòng lặp định thời gian trễ

Ở trên, đã giới thiệu chương trình tạo thời gian trễ (ví dụ 3-4). Ở đây, sẽ giới thiệu lệnh gộp viết cho thời gian trễ và các ứng dụng.

: Ví dụ 3-15: - Đồng hồ

```

ORG 00H
main: mov r0, #00h
giol: mov r1, #00h
phut1: mov r2, #00h
giay1: lcall ht          ; gọi hiển thị
      call del1s         ; gọi trễ 1 giây
      inc r2             ; tăng đếm giây
      mov a, r2          ; gửi r2 sang a
      cjne a, #60, gay1  ; so với 60, nếu chưa bằng, nhảy tới gay1
      inc r1             ; tăng đếm phút
      mov a, r1          ; gửi r1 sang a
      cjne a, #60, phut1 ; so với 60, nếu chưa bằng, nhảy tới phut1
      inc r0             ; tăng đếm giờ
      mov a, r0          ; gửi r0 sang a
      cjne a, #24, giol  ; so với 24, nếu chưa bằng, nhảy tới giol
      sjmp main          ; trở về main cho ngày hôm sau
ht:   mov P0, r0         ; hiển thị giờ
      mov P1, r1         ; hiển thị phút
      mov P2, r2         ; hiển thị giây
      ret
del1s: MOV R3, #250      ; trễ 250 × 200 × 20 = 1 giây
LAP:  MOV R4, #200
LAP1: MOV R5, #20
LAP2: DJNZ R5, LAP2
      DJNZ R4, LAP1
      DJNZ R3, LAP
      RET
      END                ; kết thúc dịch

```

Ví dụ 3-16 giới thiệu lệnh gộp có thông số cho thời gian trễ M phút, N giây.

Chú ý: Chương trình của lệnh gộp TREPS <M?, N?> trên có thể viết thêm cho thời gian trễ theo giờ với chương trình con DEL1G cho đơn vị thời gian trễ 1 giờ.

- Chương trình có chương trình con DELAY1P cho đơn vị thời gian trễ 1 phút.

- Trong lệnh gộp TREPS <M?, N?> có gọi chương trình con DEL1S và DEL1P.

b) Vòng lặp cho đồng hồ báo chuông

Ví dụ 3-17 giới thiệu chương trình cho đồng hồ báo chuông cho lớp học.

Vì có nhiều thời gian trễ khác nhau, nên ta dùng lệnh gộp có tham số M (phút) và N (giây). So với việc dùng chương trình con cho các thời gian trễ khác nhau, chương trình dùng lệnh gộp này ngắn gọn và thuận tiện hơn nhiều vì:

- Chỉ cần dùng một lệnh gộp TREPS <M?, N?> chung cho mọi thời gian trễ với các tham số M (phút) và N (giây) khác nhau.

- Các tham số M, N khác nhau cho số thời gian trễ của tiết 60 phút, tiết 70 phút, giải lao 5 phút, nghỉ trưa 30 phút v.v...

- Chương trình cần dùng chương trình con C1 để điều khiển chuông (hay đèn báo hiệu). Nếu chuông hay đèn cần dòng điều khiển lớn hơn 5 mA, ta phải mắc thêm tầng khuếch đại công suất để tránh làm hỏng VĐK.

3. Vòng lặp cho máy phát xung

Ví dụ 3-18 giới thiệu lệnh gộp cho máy phát xung tuần hoàn có điều khiển với P0.0 = 1 sẽ phát xung (Continue) khi P0.0 = 1 và dừng (Break) khi P1.1 = 1. Xung ra tại chân P1.0 với độ kéo dài 8 giây (khi P1.0 = 1) và thời gian nghỉ (khi P1.0 = 0) trong 5 giây. Thanh ghi R4 được dùng để đếm số xung đã phát và đưa chỉ thị ra cổng P2.

; Ví dụ 3-16: TREPPS - Chương trình tạo trễ N giây, M phút

```

MACRO @TREPPS <M?, N?>
    MOV R0, #2          ; thông số cho giây
    MOV R4, #1          ; thông số cho phút
    MOV R5, #00H        ; chỉ số phút đã trễ
    MOV R6, #00H        ; chỉ số giây đã trễ
    CJNE R4, #00H, DELMP ; nếu R4 đã đặt khác 0 (có trễ M phút), nhảy tới DELP
    SJMP DELNS          ; nếu R4 đã đặt bằng 0 (không có trễ phút), nhảy tới DELSN
DELM P: CALL DELAY1P    ; trễ cho 1 phút
    INC R5              ; tăng đếm phút
    MOV P0, R5          ; hiển thị số đếm phút
    DJNZ R4, DELMP      ; trở về DELMP nếu chưa đủ M phút
DELNS: CALL DELAY1S    ; trễ cho 1S
    INC R6              ; tăng đếm giây
    MOV P1, R6          ; hiển thị số đếm giây
    DJNZ R0, DELNS      ; trở về DELNS nếu chưa đủ N giây
ENDMACRO                ; kết thúc lệnh gộp
    ORG 00H            ; bắt đầu chương trình chính
    @TREPS 5, 30       ; tạo trễ 5 phút, 39 giây (tối đa 256 giây và 256 phút)
    SJMP $

DELAY1S: MOV R1, #250   ; trễ 250 × 200 × 20 = 1 giây
LAP: MOV R2, #200
LAP1: MOV R3, #20
LAP2: DJNZ R3, LAP2
    DJNZ R2, LAP1
    DJNZ R1, LAP
    RET
DELAY1P: MOV R7, #60    ; cho lặp lại 60 giây
LAPP: CALL DELAY1S
    DJNZ R7, LAPP
    RET
    END
    
```

Có thể dùng chương trình trên để phát xung có:

- Một độ trễ bất kỳ T, phải thêm lệnh gộp @TREPS T, 0 (trễ T phút);
- Độ kéo dài của xung phát ra bất kỳ;
- Thời gian nghỉ phát xung bất kỳ.

4. Vòng lặp phát một chuỗi (chữ số, ký tự)

a) Phát chuỗi chữ số

Ví dụ 3-19 giới thiệu lệnh gộp phát một chuỗi số (lệnh FOR của ngôn ngữ bậc cao). Đây cũng là vòng lặp, dùng để phát một chuỗi dữ liệu (chữ số hay ký tự) với các tham số là số hạng đầu, số bước, số hạng cuối. Vòng lặp tự kết thúc khi số phát ra cuối cùng gần bằng số hạng cuối.

b) Phát chuỗi ký tự

Ví dụ 3-20 giới thiệu chương trình ghi vào bộ nhớ PROM và phát chuỗi ký tự 'this is a test' ra cổng P0. Chuỗi trên được kê khai trong chương trình, trong vùng nhớ PROM trong từ địa chỉ 30H. Khi dịch, chuỗi ký tự trên được dịch ra mã ASCII rồi ghi vào bộ nhớ IRAM. Chuỗi được đọc vào thanh ghi A bởi lệnh MOV A, @A+DPTR (xem chương 4) và đưa ra cổng P0 bởi lệnh MOV P0, A. Chương trình trên dùng để hiện lên bộ hiển thị chuỗi mã ASCII thông báo mong muốn.

<i>; Ví dụ 3-17: Chương trình báo chuông BAOCH</i> <pre> MACRO@TREPS <M?, N?>; M phút, N giây MOV R0, #12 ; thông số cho giây MOV R5, #1 ; thông số cho phút MOV R4, #00 ; cho giây MOV R6, #00 ; cho phút CJNE R5, #00H, DELP; khác 0, tới DELP SJMP DELS ; bằng 0, tới DELS DELP: CALL DELAY1P; trễ cho 1 phút INC R5 MOV P0, R5 DJNZ R5, DELP DELS: CALL DELAY1S; trễ cho 1 giây INC R4 MOV P1, R4 DJNZ R0, DELS ENDMAC ; kết thúc lệnh gộp ; chương trình báo chuông cho lớp học ORG 00h MOV R4, #2 ; cho hai buổi sáng, chiều CALL C1 ; gọi 3 hồi chuông vào lớp CALL C1 CALL C1 @TREPS 60, 0 ; chờ tiết 60 phút CALL C1 ; gọi 1 hồi chuông giải lao @TREPS 10, 0 ; giải lao 10 phút CALL C1 ; gọi 2 hồi chuông vào lớp CALL C1 @TREPS 10, 0 ; giải lao 10 phút CALL C1 ; gọi 2 hồi chuông vào lớp </pre>	<pre> CALL C1 @TREPS 70, 0 ; chờ tiết 70 phút CALL C1 ; gọi 1 hồi chuông giải lao @TREPS 10, 0 ; giải lao 10 phút CALL C1 ; gọi 2 hồi chuông vào lớp CALL C1 @TREPS 60, 0 ; chờ tiết 60 phút CALL C1 ; gọi 1 hồi chuông @TREPS 70, 0 ; chờ tiết 70 phút CALL C1 ; gọi 1 hồi chuông @TREPS 30, 0 ; chờ 30 phút nghỉ trưa DJNZ R4, LAP ; lặp lại cho buổi chiều SJMP \$; bắt đầu chương trình con DELAY1S: MOV R1, #5; trễ 250.200.20 = 1sec LAP: MOV R2, #2 LAP1: MOV R3, #2 LAP2: DJNZ R3, LAP2 DJNZ R2, LAP1 DJNZ R1, LAP RET DELAY1P: MOV R6, #6; trễ 1 phút LAPP: CALL DELAY1S DJNZ R6, LAPP RET C1: SETB P1.0 ; bật chuông nối với bit P1.0 MOV R2, #5 ; kêu 5 giây C10: CALL DELAY1S ; trễ 1 giây DJNZ R2, C10 ; lặp lại 5 lần RET END ; kết thúc dịch </pre>
---	---

3.6. CÁC LOẠI CẤU TRÚC CƠ BẢN CỦA CHƯƠNG TRÌNH

Chương trình là một chuỗi lệnh, thực thi một nhiệm vụ nào đó theo một thuật giải (hay luận lý - algorithm) nhất định. Tùy giải thuật, chương trình có cấu trúc khác nhau. Với các lệnh gọi chương trình con và rẽ nhánh không và có điều kiện, ta có 5 cấu trúc cơ bản của chương trình theo các dạng sau:

Các dạng cấu trúc cơ bản của chương trình là:

- *Cấu trúc tuần tự đơn giản (hình 3-2a):* chương trình được thực hiện từng lệnh, từ đầu (top) tới cuối (down). Trong chương trình này không có bất kỳ lệnh rẽ nhánh nào, hay nói khác đi, dòng lệnh thực hiện đơn điệu khắp tất cả các lệnh từ đầu tới cuối chương trình, không bỏ sót và lặp lại bất kỳ lệnh nào.

- *Cấu trúc lặp vô hạn (hình 3-2b):* chương trình được thực hiện từng lệnh, từ đầu (top) tới cuối (down) nhiều lần hay lặp lại vô hạn. Trong chương trình này, có lệnh nhảy về lệnh đầu tiên (SJMP MAIN) nằm ở cuối chương trình.

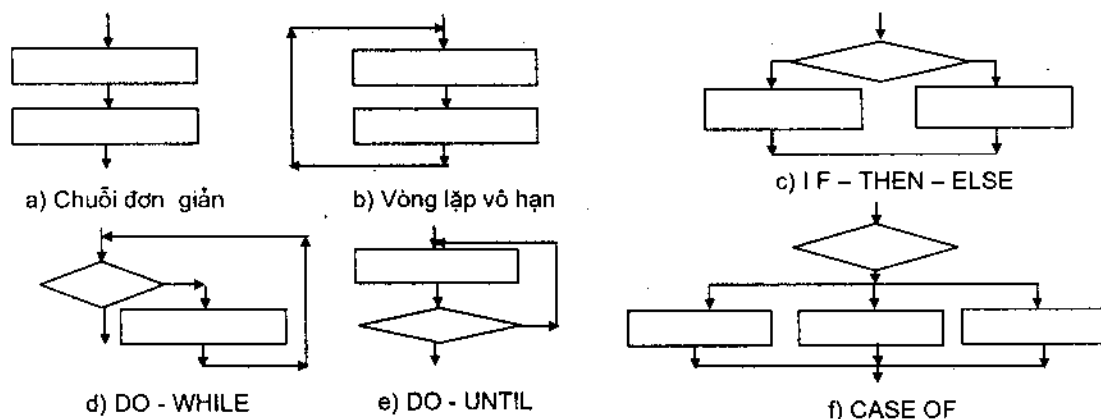
- *Cấu trúc IF-THEN-ELSE (NẾU- THÌ- KHÁC) (hình 3-2c):* chương trình gặp một quyết định (lệnh rẽ nhánh) hay lệnh JC/JNC và JZ/JNZ, JB/JNB v.v...

Chương trình có hai khả năng để lựa chọn:

: Ví dụ 3-18: FXVDK có điều khiển

```

MACRO @TRES <N?> ; lệnh gộp trễ N giây
    MOV R0, #1      ; ghi số giây N
DEL: CALL DELAY1S
    DJNZ R0, DEL
    SJMP KT
DELAY1S: MOV R1, #250 ; trễ 250 × 200 × 20 = 1 giây
LOOP:  MOV R2, #200
LOOP1: MOV R3, #20
LOOP2: DJNZ R3, LOOP2
        DJNZ R2, LOOP1
        DJNZ R1, LOOP
        RET
KT: NOP
ENDMAC
    ORG 00H          ; chương trình chính
    MOV R4, #00H     ; số xung đếm
    CLR P0            ; xoá cổng P0 để điều khiển phát xung
    CLR P1            ; xoá cổng P0 để điều khiển phát xung
    INC R4            ; đếm số xung đã phát ra
    MOV P2, R4        ; đưa số xung đã phát ra cổng P2
RE: JNB P0.0, $       ; điều khiển chạy (Continue) hay phát xung, khi P0.0 = 1 và dừng
                        ; (Break) khi P0.0 = 0
    @TRES 8           ; độ dài xung 5 giây
    CPL P1.0          ; đảo bit P1.0 để phát xung
    @TRES 8           ; độ dài nghỉ phát xung 5 giây
    INC R4            ; tăng số xung đã phát
    MOV P2, R4        ; đưa số xung ra cổng P2
    SJMP RE           ; lặp lại
    END              ; kết thúc dịch
    
```



Hình 3-2. Các cấu trúc cơ bản của chương trình

- + Nếu điều kiện xảy ra (ví dụ trạng thái của bit C, giá trị của A bằng hay khác 0 – bit z v.v...).
- + Nếu điều kiện không xảy ra.

- **Cấu trúc DO- WHILE** (hình 3-2d): cấu trúc lặp hay thoát khỏi vòng lặp khi có điều kiện xuất hiện như trạng thái của bit, giá trị dữ liệu v.v...). Chương trình có quyết định để lặp lại một số lần hay thoát khỏi vòng lặp bởi các lệnh JZ/JNZ hay JC/ JNC.

- Cấu trúc **DO- UNTIL** (hình 3-2e): cấu trúc lặp một số lần (điều kiện của chương trình), chương trình có quyết định và thoát khỏi vòng lặp bởi các lệnh DJNZ hay CJNE.

- Cấu trúc **CASE OF** (các trường hợp- hình 3-2f): chương trình có nhiều khả năng để lựa chọn. Cấu trúc này giống cấu trúc IF -THEN - ELSE (có hai khả năng xảy ra) nhưng có nhiều khả năng xảy ra.

; Ví dụ 3-19: phatso, chương trình phát một chuỗi số	; Ví dụ 3-20: phatchu, ghi, đọc, phát chuỗi 'this is a test' trong PROM
<pre> macro @phs < số đầu?, bước, số cuối?> mov r0, #1 ; nạp số đầu mov r1, #2 ; nạp bước mov r2, #3 ; nạp số cuối mov r4, #00H ; xoá số lần phát mov a, r0 mov r3, a ; lưu số đầu vào r0 lap: mov a, r3 ; gửi số phát vào a mov P0, r3 ; phát số ra P0 inc r4 ; tăng số lần phát add a, r1 ; số mới mov r3, a subb a, r2 ; so sánh với số cuối jc lap ; trở về lap nếu nhỏ hơn mov p0, r0 ; phát số đầu ra P0 mov P1, r3 ; phát số cuối ra P1 inc r4 ; cộng thêm số đầu mov P2, r4 ; đưa số lần phát ra P1 endmac ; kết thúc lệnh gộp org 00h ; bắt đầu chương trình @phatso 2, 3, 17H sjmp \$; kết thúc chờ end ; kết thúc </pre>	<pre> macro @phatchu mov DPTR, #data ; gán địa chỉ XRAM lap: mov A, #00h ; xoá A = 00H movc A, @A+DPTR; đọc trong XRAM? cjne a, #00h, hien ; kiểm tra hết chuỗi? jz kt ; kết thúc chuỗi nếu A = #00H hien: mov P0, A ; đưa ra hiển thị ở P0 inc DPTR ; tăng địa chỉ nhớ mp lap ; trở về lap endmac org 00h ; bắt đầu ljmp 100h org 30h data: db 'this is a test', 00h org 100h @phchk kt: sjmp \$; kết thúc chờ end ; kết thúc dịch </pre>

CÂU HỎI VÀ THỰC HÀNH

I. Câu hỏi

1. Hãy so sánh các lệnh rẽ nhánh không điều kiện và có điều kiện về bản chất, cách sử dụng trong từng trường hợp.
2. Hãy so sánh ứng dụng của hai lệnh JMP và CALL.
3. Hãy so sánh ứng dụng của hai lệnh DJNZ và CJNE.
4. Nêu các ứng dụng của lệnh DJNZ.
5. Nêu các ứng dụng của lệnh CJNE.
6. Hãy so sánh các lệnh rẽ nhánh có điều kiện JZ/JNZ, JC/JNC với các lệnh JB/JNB và JBC.
7. Nêu nguyên tắc tạo các lệnh gộp nhảy chương trình và gọi chương trình điều kiện kép.
8. Vòng lặp là gì ? Hãy so sánh vòng lặp hữu hạn và vô hạn.
9. Nêu cách kết thúc vòng lặp hữu hạn. Ứng dụng.
10. Nêu ứng dụng vòng lặp trong lệnh gộp.

II. Thực hành

1. Soạn thảo, hợp dịch, cho chạy thử các ví dụ 3-1 tới 3-9 với chương trình PINNACLE.
2. Nạp ví dụ 3-5 vào vi mạch VĐK và cho chạy chương trình trên kit.
3. Nạp ví dụ 3-6 vào vi mạch VĐK và cho chạy chương trình trên kit khi cho giá trị vào của cổng các giá trị < 75 , $= 75$, và > 75 .
4. Vẽ giải thuật và viết chương trình dùng cả hai lệnh DJNZ và CJNE.
5. Vẽ giải thuật và viết chương trình tạo trễ dùng cả hai lệnh DJNZ và CJNE.
6. Vẽ giải thuật và viết chương trình thực đơn với các chứa A, B, C và F (thoát).
7. Vẽ giải thuật và viết chương trình thực đơn như câu 3 nhưng dùng lệnh JMP @A+DPTR.
8. Soạn thảo, hợp dịch, liên kết và cho chạy thử các ví dụ về vòng lặp cho thời gian.
9. Soạn thảo, hợp dịch, liên kết và cho chạy thử các ví dụ về vòng lặp cho phát xung.
10. Soạn thảo, hợp dịch, liên kết và cho chạy thử các ví dụ về vòng lặp cho phát chuỗi số.

Chương 4

BỘ NHỚ TRONG, CÁC PHƯƠNG PHÁP ĐỊA CHỈ VÀ NHÓM LỆNH DỊCH CHUYỂN

Mục đích của chương:

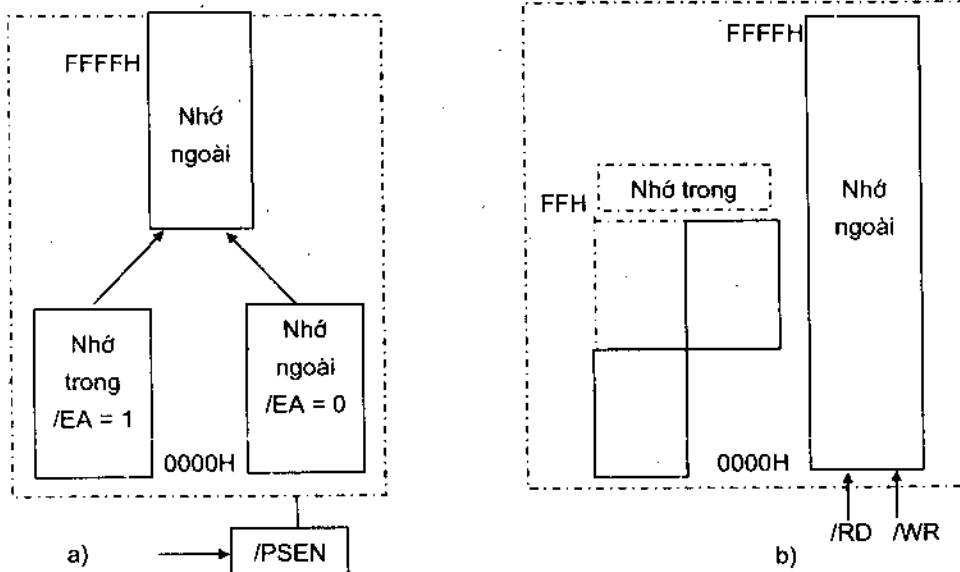
- Khảo sát bộ nhớ chương trình (PROM) bên trong của VDK.
- Khảo sát bộ nhớ đệm dữ liệu (RAM) bên trong của VDK gồm các thanh ghi thông dụng, các ô nhớ đệm dữ liệu, các thanh ghi đặc biệt SFR, trong đó có các thanh ghi xử lý bit.
- Khảo sát các phương pháp địa chỉ dành cho thanh ghi, bộ nhớ.
- Khảo sát hệ lệnh dịch chuyển dành cho thanh ghi.

4.1. BỘ NHỚ CHƯƠNG TRÌNH (PROM) BÊN TRONG CỦA VDK 8051/8052

VDK thuộc họ MCS-51 có thể không có (8031) hay có bộ nhớ PROM trong là:

- 128 Byte RAM và 4 kB ROM cho VDK 8051.
- 256 Byte RAM và 4 kB ROM cho VDK 8052.

Nói chung, các VDK đều có riêng bộ nhớ chỉ đọc (ROM) để nhớ chương trình (PROM) và bộ nhớ đọc-ghi để nhớ dữ liệu (RAM) ở bên trong và ngoài VDK như ở hình 4-1. Chúng có cùng một địa chỉ từ 000H tới FFFFH (địa chỉ phủ lên nhau (overlap), đều có dung lượng 64 kB nhưng cách ghi và đọc khác nhau do các lệnh khác nhau (nhớ ROM ghi bằng bộ ghi chương trình - Programmer, còn đọc bởi tín hiệu /PSEN), nhớ RAM ghi bởi tín hiệu /WR và đọc bởi tín hiệu /RD).



Hình 4-1. Bộ nhớ ROM (a) và RAM (b) ở bên trong và ngoài của VDK họ MCS-51

Ở chương này, ta chỉ xét bộ nhớ trong (ROM, RAM) của VDK họ MCS-51, còn bộ nhớ ngoài (ROM, RAM) dành cho tập II về ghép nối cơ bản của hệ VDK.

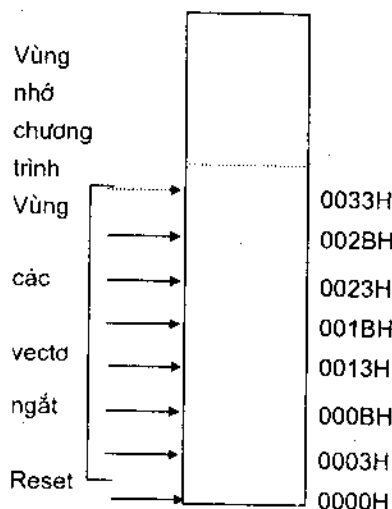
Bộ nhớ ROM bên trong này được dùng để đọc chương trình từ ROM vào CPU, khi chân /EA nối với nguồn +5 V (Vcc) tức chạy chương trình đã ghi nhớ sẵn trong PROM. Bộ nhớ PROM trong này có thể có dung lượng 4 kB (8051) hay 8 kB (8052). Hình 4-2 giới thiệu bộ nhớ ROM bên trong với các địa chỉ thấp là các vectơ ngắt (bảng 4-1).

Với bộ nhớ ROM 4 kB, có địa chỉ từ 0000H tới 0FFFFH, còn bộ nhớ ROM 8 kB có địa chỉ từ 0000H tới 1FFFFH.

Khi bật nguồn điện nuôi VDK hay ấn nút RESET, thanh ghi đếm chương trình PC được xóa về 0000H, thanh ghi con trỏ ngăn xếp SP = 07H, còn các cổng P0, P1, P2 có giá trị 0FFH. Nếu chưa ghi chương trình trong ROM, VDK không hoạt động.

Bảng 4-1. Bảng vectơ ngắt của PROM trong

Thứ tự	Tên vectơ ngắt	Địa chỉ	Chân
0	RESET (ấn nút xóa)	0000H	9
1	Ngắt phần cứng ngoài 0 (INT0)	0003H	12 (P3.2)
2	Ngắt Timer 0 (TF0)	000BH	
3	Ngắt phần cứng ngoài 1 (INT1)	0013H	13 (P3.13)
4	Ngắt Timer 1 (TF1)	001BH	
5	Ngắt nối tiếp (RI, TI)	0023H	
6	Ngắt Timer 2 (TF2)	0033H	



Hình 4-2. Bộ nhớ PROM bên trong

Khoảng cách địa chỉ của hai vectơ ngắt liên tiếp là 8 byte nhớ. Vectơ ngắt là địa chỉ bắt đầu của chương trình phục vụ ngắt. Do đó, nếu chương trình phục vụ ngắt có:

- *Chuỗi lệnh có độ dài nhỏ hơn 8 byte* (kể cả lệnh trở về từ ngắt RETI): ta có thể ghi trực tiếp chương trình phục vụ ngắt nằm trong vùng nhớ dành cho vectơ ngắt.

- *Chuỗi lệnh có độ dài lớn hơn 8 byte* (kể cả lệnh trở về từ ngắt RETI): để tránh đề lên chương trình phục vụ ngắt của vectơ ngắt khác, tại vectơ ngắt chỉ có lệnh nhảy dài (LJMP nhân) tới nhân của chương trình phục vụ ngắt tương ứng nằm trong vùng nhớ của chương trình chính (lớn hơn 0030H). Nếu chương trình không có chương trình phục vụ ngắt nào thì không cần ghi gì (cho chương trình chính sử dụng vùng nhớ hoặc chỉ có lệnh RETI (trở về từ ngắt) ở trong chương trình chính.

Ở chương 2 đã thấy, khi có chương trình đã soạn thảo và đã ghi vào PROM, tại địa chỉ của RESET (000H) có:

- Lệnh bắt đầu của chương trình chính, nếu không có chương trình phục vụ vectơ ngắt nào, chương trình chỉ có một khối là chương trình chính và các lệnh của chương trình có thể chiếm chỗ của các vectơ ngắt mà không sợ đề lên chương trình của các vectơ ngắt.

- Lệnh nhảy tới nhân của chương trình chính (MAIN) có địa chỉ từ 0030H (không có nhớ đệm và SP = 7H) hay 100H nếu có chương trình phục vụ vectơ ngắt và dùng nhớ đệm dữ liệu và SP = 70H.

Như vậy, sau khi bật nguồn điện nuôi hay ấn nút RESET, chương trình bắt đầu thực hiện từ địa chỉ hay giá trị của PC là 0000H. Nếu có lệnh nhảy LJMP MAIN, chương trình nhảy tới chương trình chính có nhân là MAIN tại địa chỉ 30H hoặc 100H.

Ví dụ 4-1 mô tả sơ đồ khối của chương trình khi có các chương trình phục vụ ngắt có độ dài địa chỉ lớn hơn 8 byte. Hình 4-2 giới thiệu các địa chỉ nhớ PROM theo Pinnacle. Khi chương trình phục vụ ngắt có độ dài nhỏ hơn 8 byte, chúng được viết trực tiếp trên vùng nhớ dành cho vectơ ngắt đó.

Tùy chương trình ứng dụng, sẽ sử dụng chương trình ngắt nào (xem chương 10) và khi đó, sẽ có các chương trình phục vụ ngắt tương ứng. Trường hợp đơn giản nhất, không có chương trình

phục vụ ngắt nào, chương trình chính bắt đầu từ địa chỉ 0000H và không có lệnh nhảy LJMP MAIN mà không sợ đè lên bất kỳ vùng nhớ dành cho chương trình phục vụ ngắt.

Ngoài ghi chương trình, bộ nhớ ROM bên trong VDK còn được ghi bằng dữ liệu và đọc bằng các lệnh đọc PROM như MOV_C A, @A+PC hay MOV_C A, @A+DPTR (sẽ xét ở mục sau).

; Ví dụ 4-1: Sơ đồ khối của chương trình ghi ở ROM trong	MAIN: ; lệnh bắt đầu của chương trình chính ; kết thúc chương trình chính ; các chương trình phục vụ ngắt và con EXINTER0: ; chương trình phục vụ ngắt bàn phím ; các lệnh của chương trình ngắt bàn phím RETI ; kết thúc chương trình ngắt TIMER0: ; chương trình phục vụ ngắt TIMER0 ; các lệnh của chương trình phục vụ RETI ; kết thúc chương trình phục vụ ngắt EXINTER1: ; EXINTER1 không dùng RETI TIMER1: RETI ; chương trình TIMER1 không dùng SERIAL: ; các lệnh của chương trình phục vụ SERIAL RETI ; kết thúc chương trình phục vụ ngắt ; các chương trình con END ; kết thúc chương trình dịch
ORG 00H ; bắt nguồn nuôi (RESET) LJMP MAIN ; nhảy tới chương trình chính ORG 03H ; ngắt ngoài 0 cho bàn phím LJMP EXINTER0 ; nhảy tới nhân ngắt ngoài ORG 0BH ; ngắt dành cho TIMER0 LJMP TIMER0 ; nhảy tới nhân của TIMER0 ORG 13H ; ngắt ngoài 1 dành cho hệ đo LJMP EXINTER1 ; nhảy tới EXINTER1 ORG 1BH ; ngắt dành cho TIMER1 LJMP TIMER1 ; nhảy tới nhân TIMER1 ORG 23H ; ngắt dành cho trao đổi nối tiếp LJMP SERIAL ; nhảy tới nhân SERIAL ORG 2BH ; ngắt dành cho TIMER2 LJMP TIMER2 ; nhảy tới nhân của TIMER2 ORG 30H ; địa chỉ chương trình chính	

4.2. BỘ NHỚ RAM BÊN TRONG (IRAM) VÀ CÁC THANH GHI CỦA VDK

4.2.1. Các vùng nhớ IRAM

Bộ nhớ RAM bên trong của VDK họ MCS-51 như hình 4-1b, chỉ có dung lượng tới 126 byte (8051) hay 256 byte (8052). Bộ nhớ RAM bên trong (IRAM) của 8051 được chia thành bốn vùng (hình 4-3 và 4-4).

- 32 byte thanh ghi thông dụng: chia làm 4 băng R0-R7, có địa chỉ từ 0000H tới 1FH, có thể gọi địa chỉ trực tiếp (bằng con số chỉ địa chỉ) và gián tiếp qua các thanh ghi R0, R1 (hình 4-5).

- 16 byte trạng thái: có địa chỉ từ 0020H tới 2FH.

- 80 byte đệm dữ liệu và ngăn xếp: có địa chỉ từ 0030H tới 07FH, dành cho các ô nhớ đệm dữ liệu và ngăn xếp (SP = 70h).

- 128 byte các thanh ghi đặc biệt SFR: có địa chỉ từ 0080H tới FFH, chứa các thanh ghi đặc biệt SFR (vùng màu đen, hình 4-4), có thể gọi địa chỉ trực tiếp (bằng số chỉ địa chỉ hay bằng tên thanh ghi).

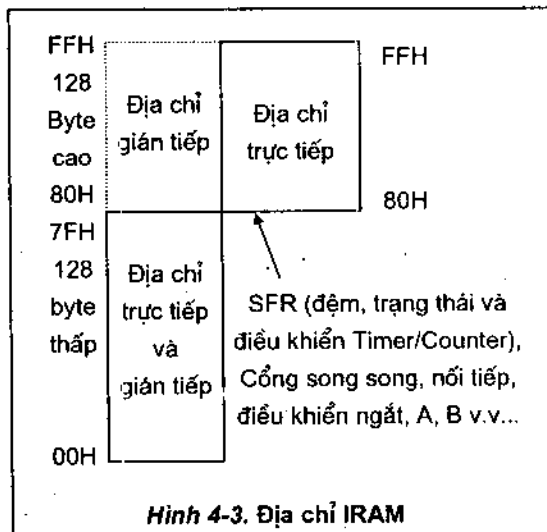
Riêng VDK 8052 còn có thêm 128 byte cao nữa có địa chỉ từ 0080H tới 7FH, gọi địa chỉ gián tiếp qua các thanh ghi R0, R1.

4.2.2. Các băng thanh ghi thông dụng

Có 4 băng thanh ghi thông dụng R0 tới R7, có địa chỉ từ 00H tới 1FH (từ 00D tới 31D). Việc chọn băng này do hai bit RS0, RS1 của thanh ghi trạng thái PSW quyết định với các giá trị như hình 4-6 và 4-9.

Công dụng của các băng thanh ghi này như sau:

- Dùng làm bộ nhớ nháp trong quá trình thực hiện chương trình. Nếu chương trình cần nhiều chương trình con, mỗi chương trình con này cần các thanh ghi dữ



Hình 4-3. Địa chỉ IRAM

liệu, người ta chọn mỗi băng (dải) thanh ghi này cho mỗi chương trình con.

- Các thanh ghi R0, R1 còn dùng để ghi địa chỉ của nhớ dữ liệu IRAM trong chế độ địa chỉ gián tiếp thanh ghi.

Internal RAM (IRAM)

Display Mode:

☒ Hex
☐ ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
80	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Hình 4-4. Phân bố vùng nhớ theo bit và SFR

4.2.3. Các byte thanh ghi địa chỉ theo bit

Có địa chỉ từ 20H tới 2FH dành cho các bit trạng thái (hình 4-5, 4-6 và 4-7).

Đây là các thanh ghi có thể sử dụng từng bit bởi các lệnh xử lý bit, dùng để chỉ trạng thái của thiết bị trong quá trình lập trình. Các bit trạng thái này sẽ quyết định hành động tiếp theo của chương trình như rẽ nhánh, trao đổi tin v.v...

Đây là các thanh ghi có thể sử dụng từng bit bởi các lệnh xử lý bit, dùng để chỉ trạng thái của thiết bị trong quá trình lập trình. Các bit trạng thái này sẽ quyết định hành động tiếp theo của chương trình như rẽ nhánh, trao đổi tin v.v...

Bit Fields

BIT 7 6 5 4 3 2 1 0

IRAM

00 08 10 18 20 28 30 38

20h 21h 22h 23h 24h 25h 26h 27h

00 00 00 00 00 00 00 00

BIT 7 6 5 4 3 2 1 0

IRAM

40 48 50 58 60 68 70 78

28h 29h 2Ah 2Bh 2Ch 2Dh 2Eh 2Fh

00 00 00 00 00 00 00 00

SFR Direct

Hex ASCII

0 1 2 3 4 5 6 7

80 FF 07 00 00 00 00 00

88 00 00 00 00 00 00 00

90 FF 00 00 00 00 00 00

98 00 00 00 00 00 00 00

A0 FF 00 00 00 00 00 00

A8 00 00 00 00 00 00 00

B0 FF 00 00 00 00 00 00

B8 00 00 00 00 00 00 00

C0 00 00 00 00 00 00 00

C8 00 00 00 00 00 00 00

D0 00 00 00 00 00 00 00

D8 00 00 00 00 00 00 00

E0 00 00 00 00 00 00 00

E8 00 00 00 00 00 00 00

User Bit Field 46h - IRAM 28.6

Standard 8052 Ln Col

Start Graphics Server Pinnacle 52 Professio... 12:05 AM

Hình 4-5. Các vùng nhớ bit và SFR của IRAM theo Pinnacle

4.2.4. Vùng nhớ RAM thông dụng và ngăn xếp

Vùng nhớ này có địa chỉ theo byte, từ 30H tới 7FH (hình 4-6), dùng để ghi và đọc dữ liệu đệm cho chương trình. Trong chương trình có dùng các vectơ ngắt, phải khởi phát cho thanh ghi con trỏ địa chỉ nhớ ngăn xếp SP có địa chỉ là 70H, tức dành cho ngăn xếp 16 ô nhớ có địa chỉ từ 70H tới 7FH. Một ô nhớ của vùng này được định địa chỉ trực tiếp và gián tiếp qua các thanh ghi R0, R1.

4.2.5. Vùng các thanh ghi đặc biệt SFR

Vùng nhớ này có địa chỉ từ 80H tới FFH, chứa 21 thanh ghi đặc biệt SFR (hình 4-6, 4-7 và 4-8) với bảng 4-2.

Các thanh ghi đặc biệt này gồm có:

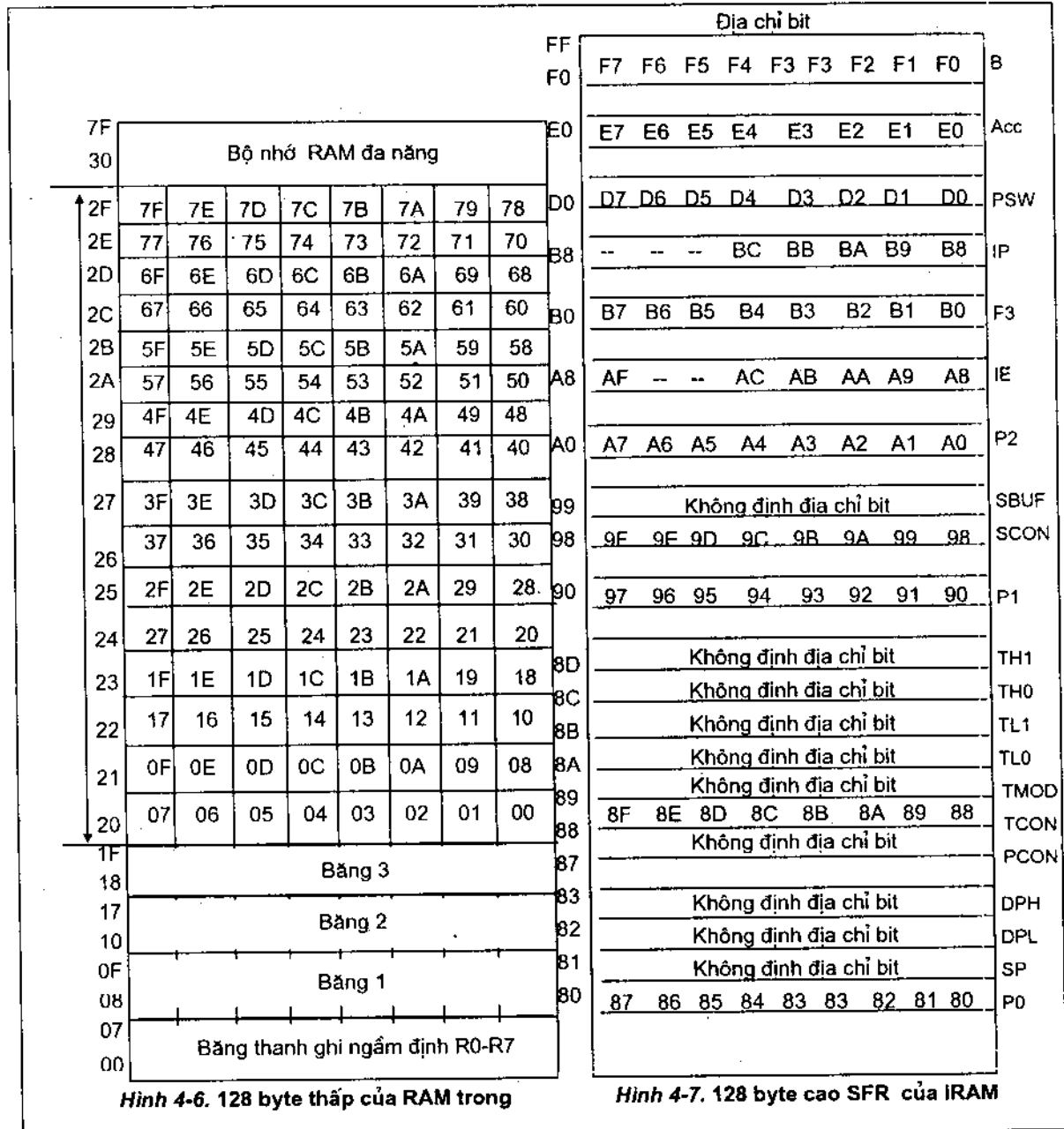
- Các thanh ghi tích lũy A và B (hay chứa – Accumulator): được dùng để thực hiện các lệnh số học, logic. Thanh ghi A còn là thanh ghi đệm dữ liệu để trao đổi tin giữa các thanh ghi thông

dụng, khối nhớ trong, khối nhớ ngoài, các cổng song song và nối tiếp. Thanh ghi B là thanh ghi đệm thông dụng và thanh ghi chứa toán hạng thứ hai trong các phép tính số học và logic.

- Các cổng song song (P0, P1, P2, P3): dùng để trao đổi tín hiệu song song, nối tiếp với các thiết bị ngoài về dữ liệu, trạng thái và điều khiển.

- Các thanh ghi điều khiển, trạng thái và dữ liệu cho Timer/Counter: TMOD, TCON, TH0, TL0, TH1, TL1, TH2, TL2, RCAP2L, RCAP2H. Các thanh ghi này có thể gọi địa chỉ trực tiếp bằng ký hiệu theo tên hay địa chỉ của thanh ghi.

- Thanh ghi trạng thái: thanh ghi này chỉ trạng thái của VDK, có các bit như hình 4-9. Thanh ghi này có các bit cờ nhớ CY, cờ nhớ phụ AC, cờ tràn dữ liệu OV và cờ chặn/lẻ P, cùng hai bit RS1, RS0 để xác định bằng nhớ các thanh ghi thông dụng R0-R7. Các bit trạng thái này do kết quả thực hiện các phép tính số học trong khối ALU của VDK (xét ở chương 5).



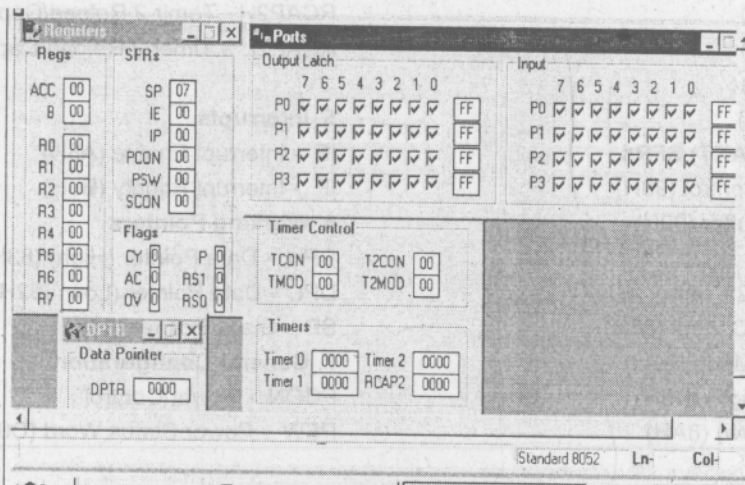
4.2.6. Vùng nhớ 128byte cao, địa chỉ gián tiếp theo byte

Tuy cùng giá trị địa chỉ như các thanh ghi đặc biệt SFR (địa chỉ 80H tới FFH) nhưng vùng nhớ này được gọi địa chỉ một cách gián tiếp và chỉ có VDK 8052 mới có, dùng để ghi dữ liệu cho chương trình (mở rộng vùng 30H tới 7FH của 8051).

Như vậy, VDK 8052 có địa chỉ nhớ RAM bên trong từ địa chỉ 30H tới FFH. Vùng này chỉ gọi địa chỉ theo byte.

- Các thanh ghi điều khiển và cho phép ngắt chương trình: IP, IE.
- Thanh ghi điều khiển nguồn nuôi: PCON.
- Thanh ghi trạng thái: PSW với các bit như hình 4-9.

Khi xét tới các hành động cơ bản ở phần sau, chúng ta sẽ xét kỹ các thanh ghi này.



Hình 4-8. Các thanh ghi đặc biệt theo Pinnacle

CY	AC	F0	RS1	RS0	OV	-	P
CY PSW.7	; cờ nhớ trong phép cộng, trừ và so sánh						
AC PSW.6	; cờ phụ hiệu chỉnh thập phân						
F0 PSW.5	; dành cho người dùng sau						
RS1 PSW.4	; bit 1 của chọn bằng thanh ghi thông dụng R0-R7						
RS0 PSW.3	; bit 0 của chọn bằng thanh ghi thông dụng R0-R7						
OV PSW.2	; cờ tràn khi dữ liệu vượt quá số bit của thanh ghi						
- PSW.1	; dành cho người dùng						
P PSW.0	; cờ chặn/lẻ						
RS1	RS0	Bảng thanh ghi		Địa chỉ			
0	0	0		00H-07H			
0	1	1		08H-0FH			
1	0	2		10H-17H			
1	1	3		18H-1FH			

Hình 4-9. Thanh ghi trạng thái PSW

Bảng 4-2. Các thanh ghi SFR

8052 Microcontroller Online Help	
Special Function Registers (SFRs)	
(số trong ngoặc là địa chỉ)	
1. Working Registers	
ACC - Accumulator (E0H)	
B- 'B' Auxillary Register (F0H)	TH1 - Timer 1 (High) (8DH)
R0-R7 - "R" Register (Register Banks)	TL1 - Timer 1 (Low) (8BH)
(00H - 1FH)	TH2 - Timer 2 (High)
	TL2 - Timer 2 (Low)
2. Input/Output SFRs	RCAP2H - Timer 2 Reload/Capture (High)
P0 - Port 0 (80H)	RCAP2L - Timer 2 Reload/Capture (Low)
P1 - Port 1 (90H)	
P2 - Port 2 (A0H)	
P3 - Port 3 (B0H)	
3. Serial Port (UART) SFRs	5. Interrupts
SCON - Serial Control (98H)	IE - Interrupt Enable (A8H)
SBUF - Serial Buffer (99H)	IP - Interrupt Priority (B8H)
4. Timer SFRs	6. Working Pointers
T2CON - Timer 2 Control (8052)	DPH - Data Pointer (High) (83H)
TCON - Timer Control (88H)	DPL - Data Pointer (Low) (82H)
TMOD - Timer Mode (89H)	SP - Stack Pointer (81H)
TH0 - Timer 0 (High) (8CH)	7. General Configuration
TL0 - Timer 0 (Low) (8AH)	PCON - Power Control
	PSW - Power Status Word (D0H)

4.3. CÁC PHƯƠNG PHÁP ĐỊA CHỈ CỦA TOÁN HẠNG

Các phương pháp địa chỉ là cách gọi địa chỉ hay cách tìm dữ liệu của toán hạng cho lệnh khi căn cứ vào phương pháp địa chỉ của toán hạng. Ở chương 1, ta đã thấy, một lệnh nói chung, có mã lệnh và các dữ liệu cho lệnh là các toán hạng nguồn (nơi lấy dữ liệu cho lệnh thực hiện) và toán hạng đích (nơi gửi dữ liệu kết quả sau khi thực hiện lệnh). Ở đây, chúng ta xét chi tiết các phương pháp địa chỉ trên cho nhóm lệnh dịch chuyển dữ liệu trong các thanh ghi và các ô nhớ (ROM, RAM ở cả bên trong và bên ngoài VDK).

VDK họ MCS51 có các phương pháp địa chỉ tức thì, thanh ghi, trực tiếp, gián tiếp, chỉ số, tuyệt đối, tương đối và dài. Về nguồn dữ liệu hay nơi chứa dữ liệu có ba nguồn là:

- *Tức thì (Imm)*: dữ liệu là con số ghi tức thì liền sau mã lệnh.
- *Nội dung thanh ghi R*: nội dung thanh ghi R0-R7 hay SFR chứa dữ liệu (Đ-đích hay S-nguồn) cho lệnh.
- *Nội dung ô nhớ*: dữ liệu chứa trong ô nhớ. Để gọi dữ liệu này, ta có các phương pháp gọi địa chỉ sau:

+ *Địa chỉ trực tiếp (Direct)*: địa chỉ ô nhớ là một con số trực tiếp sau tên lệnh (không có dấu thăng (#) như của dữ liệu tức thì -Imm).

+ *Địa chỉ gián tiếp thanh ghi (Indirect)*: dữ liệu trong ô nhớ có địa chỉ của ô nhớ là dữ liệu của thanh ghi. Loại địa chỉ này được biểu diễn dạng tên thanh ghi dùng làm con trỏ địa chỉ (R0, R1, PC, DPTR) sau chữ 'a công' (@).

Do đó, ở đây, chúng ta xét các dữ liệu của lệnh theo ba nguồn này. Các ví dụ về phương pháp gọi địa chỉ cũng là các ví dụ cho lệnh MOV D, S thuộc nhóm lệnh dịch chuyển. Các phương pháp địa chỉ trên cũng dùng cho các lệnh thuộc các nhóm khác.

4.3.1. Địa chỉ tức thì

Phương pháp địa chỉ tức thì này có dữ liệu là một con số, sau ký hiệu dấu thăng (#), nằm sau tên lệnh.

Ví dụ: MOV A, #57H ; tức nạp giá trị 57H vào thanh ghi A. Chú ý có ký hiệu # đứng trước dữ liệu, nếu không có #, chương trình dịch sẽ hiểu là nội dung của ô nhớ có địa chỉ là 57H được gửi vào A (địa chỉ trực tiếp).

Dạng tổng quát của lệnh dịch chuyển trên được viết như sau:

MOV A, #DATA

, trong đó DATA có thể là:

- Một con số cụ thể;
- Một ký hiệu đã được khai báo bằng EQU;
- Một nhãn của một hay nhiều dữ liệu (con số hay ký tự).

Ví dụ MOV A, #MYDATA, trong đó MYDATA đã được định nghĩa MYDATA EQU 57H ở đầu chương trình hay MYDATA là một nhãn của dữ liệu đã khai báo ở đầu hay cuối chương trình như: MYDATA: DB "CHÀO ANH"; là các mã ASCII của các ký tự C, H, A, O, A, N, H.

; Ví dụ 4-2: Các phương pháp địa chỉ tức thì và thanh ghi

```

ORG 00H
MOV A, #00H          ; nạp số #00H hay xóa thanh ghi A
MOV B, A              ; chuyển A cho B
LOOP: MOV P0, A        ; đưa giá trị của A ra cổng P0
MOV P1, A              ; đưa giá trị của A ra cổng P1
MOV P2, A              ; đưa giá trị của A ra cổng P2
MOV P3, A              ; đưa giá trị của A ra cổng P3
MOV R0, A              ; đưa giá trị của A vào R0
MOV R1, R0             ; đưa giá trị của R0 vào R1
MOV R2, R1             ; đưa giá trị của R1 vào R2
MOV R3, R2             ; đưa giá trị của R2 vào R3
MOV R4, R3             ; đưa giá trị của R3 vào R4
MOV R5, R4             ; đưa giá trị của R4 vào R5
MOV R6, R5             ; đưa giá trị của R5 vào R6
MOV R7, R6             ; đưa giá trị của R6 vào R7
INC A                  ; tăng A
CJNE A, #9H, LOOP      ; A = 9H chưa, về LOOP
SJMP $                 ; kết thúc chờ tại chỗ
END                     ; kết thúc dịch

```

4.3.2. Địa chỉ thanh ghi R (R0-R7)

Toán hạng là nội dung thanh ghi thông dụng R0-R7 của một trong 4 dải (băng) hay thanh ghi đặc biệt SFR.

Lệnh có dạng:

MOV A, Ri

; gửi nội dung thanh ghi thông dụng Ri vào A

hay:

MOV Ri, A

; gửi nội dung thanh ghi A vào thanh ghi Ri

Ví dụ: MOV A, R3; gửi nội dung thanh ghi R3 của dải hiện hành, đã xác định bởi RS1, RS0 của thanh ghi PSW vào thanh ghi A.

Ví dụ 4-2 giới thiệu chương trình dịch chuyển dữ liệu dùng lệnh MOV với các phương pháp địa chỉ tức thì và thanh ghi. Khi chạy chương trình này, chúng ta theo dõi được nội dung thay đổi của các thanh ghi.

4.3.3. Địa chỉ bộ nhớ (trực tiếp và gián tiếp thanh ghi)

Phương pháp địa chỉ này lấy toán hạng từ ô nhớ của bộ nhớ. Tùy theo cách đưa địa chỉ của ô nhớ, ta có phương pháp địa chỉ trực tiếp hay gián tiếp thanh ghi.

a) Phương pháp địa chỉ trực tiếp

Địa chỉ ô nhớ là một con số, được ghi trực tiếp liền sau mã lệnh, khác với địa chỉ tức thì là một con số nhưng không có dấu thăng (#).

Dạng lệnh là:

MOV A, con_so

; trong đó con_so là địa chỉ ô nhớ

Ví dụ : MOVA, 56H ; gửi nội dung ô nhớ có địa chỉ là 56H vào A.

Ví dụ : MOV 30H, 56H ; gửi nội dung ô nhớ có địa chỉ là 56H vào ô nhớ địa chỉ 30H.

b) Phương pháp địa chỉ gián tiếp thanh ghi

Chỉ có hai thanh ghi R0, R1 được dùng, nội dung của các thanh ghi này là địa chỉ của ô nhớ, đứng trước các thanh ghi R0, R1 là dấu a cộng (@).

Dạng lệnh là :

hay:	MOV A, @Ri	; gửi nội dung ô nhớ có địa chỉ là giá trị của Ri vào A
	MOV @Ri, A	; gửi A vào ô nhớ có địa chỉ là giá trị của Ri,
		; trong đó Ri là một trong hai thanh ghi R1 hay R2.
	MOV @R0, @R1	; gửi nội dung ô nhớ có địa chỉ là R1 vào ô nhớ có địa chỉ R0.

Ví dụ, gửi nội dung của ô nhớ có địa chỉ 57H vào thanh ghi A, có thể được viết bằng hai lệnh như sau: MOV R0, #57H ; gửi 57H vào thanh ghi R0 dùng làm địa chỉ ô nhớ.

MOV A, @R0 ; gửi ô nhớ có địa chỉ ở trong thanh ghi R0 (là 57H) vào thanh ghi A.

Nếu là bộ nhớ RAM ngoài, địa chỉ có thể vượt quá 8 bit, khi đó dùng thanh ghi con trỏ địa chỉ DPTR 16 bit chỉ địa chỉ. Trước khi viết lệnh cho DPTR, chúng ta phải gán cho DPTR giá trị của địa chỉ. Lệnh như sau:

hay: MOV DPTR, # con_so ; đọc nội dung ô nhớ ngoài có địa chỉ DPTR vào A

hay: MOV DPTR, # con_so ; ghi nội dung A vào ô nhớ ngoài có địa chỉ DPTR
MOVX @DPTR, A ; chú ý là lệnh MOVX, chữ X (eXtern – ngoài)

hay: MOVC A, @R1 ; đọc ô nhớ ROM (có chữ C) vào thanh ghi chứa A

Phương pháp địa chỉ gián tiếp thanh ghi này chạy chậm hơn địa chỉ trực tiếp vì phải ghi hai lệnh nhưng thuận tiện cho chương trình có sự tăng hay giảm liên tục địa chỉ nhớ (qua R0, R1 hay DPTR) của một vùng nhớ.

Ví dụ 4-3 mô tả chương trình xóa vùng nhớ có địa chỉ từ 60H tới 80H và ghi vào các ô nhớ.

c) Định địa chỉ chỉ số

Trong phép định địa chỉ chỉ số là địa chỉ gián tiếp thanh ghi, nhưng địa chỉ của ô nhớ là tổng số của : - Thanh ghi cơ sở hay thanh ghi nền: là bộ đếm chương trình PC hay con trỏ dữ liệu DPTR.

; Ví dụ 4-3 : Xóa vùng nhớ có địa chỉ từ 60H tới 80H và ghi vào các ô nhớ.

ORG 00H	
MOV R0, #60H	; nạp địa chỉ 60H cho R0
LOOP: MOV @R0, #00H	; xóa ô nhớ có địa chỉ 60H
INC R0	; tăng R0 hay địa chỉ
CJNE R0, #80H, LOOP	; so sánh R0 = 80H chưa, nếu chưa bằng, nhảy về LOOP
MOV A, #01H	; chuyển #01H vào A
MOV 60H, A	; chuyển #01H vào ô nhớ có địa chỉ 60H
INC A	; tăng A
MOV @R0, A	; chuyển #02H vào ô nhớ có địa chỉ R0 = 80H
INC A	; tăng A
MOV DPTR, A	; chuyển #02H vào thanh ghi DPTR
MOVX @DPTR, A	; chuyển #02H vào ô nhớ ngoài có địa chỉ là thanh ghi DPTR
MOVC 63H, A	; chuyển #02H vào ô nhớ ROM bên trong có địa chỉ 63H
SJMP \$	; kết thúc chờ tại chỗ
END	; kết thúc chương trình dịch

và lệnh là:

MOVC A, @A + DPTR
MOVC A, @A + PC

 ; đọc ô nhớ chương trình có địa chỉ là A+DPTR
; đọc ô nhớ chương trình có địa chỉ là A+PC.

Các lệnh trên dùng để đọc một bảng dữ liệu, thường là bảng biến đổi mã (xem ở dưới).

4.4. NHÓM LỆNH DỊCH CHUYỂN DỮ LIỆU

Có bốn loại nhóm lệnh cho các bộ nhớ: cho RAM trong, cho RAM ngoài, cho bảng dữ liệu và cho bộ nhớ ROM ngoài.

4.4.1. Nhóm lệnh dịch chuyển cho RAM trong

Bảng 4-3 giới thiệu nhóm lệnh dịch chuyển dữ liệu cho RAM trong, thấy có các nhóm khác nhau.

1. Các lệnh dịch chuyển MOV <dest>, <src>, MOV A, <src>; MOV <dest>, A

Đây là lệnh dịch chuyển dữ liệu điển hình, giữa toán hạng nguồn <src> vào toán hạng đích <dest>, trong đó:

- <src> có thể là toán hạng tức thì (Imm. dạng #data), thanh ghi A, thanh ghi Ri (Reg dạng Ri), địa chỉ trực tiếp (Dir dạng con số số học), gián tiếp thanh ghi (Ind. dạng @Ri).

- <dest> có thể là toán hạng, thanh ghi A, thanh ghi Ri (Reg dạng Ri), địa chỉ trực tiếp (Dir dạng con số số học), gián tiếp thanh ghi (Ind. dạng @Ri).

Từ lệnh MOV <dest>, <src> ta có 15 lệnh với tổ hợp của <dest> và <src> như bảng 4-3.

Bảng 4-3. Nhóm lệnh dịch chuyển dữ liệu cho RAM trong

Lệnh	Hành động	Các chế độ địa chỉ				Thời gian thực hiện (μs)
		Di	Ind.	Reg.	Imm.	
MOV A, <SRC>	A = <SRC>	X	X	X	X	1
MOV <dest>, A	<dest> = A	X	X	X		1
MOV <dest>, <SRC>	<dest> = <src>	X	X	X	X	2
MOV DPTR, #data	DPTR = 16 bit immediat constant	X				2
PUSH <SRC>	INC SP; MOV @SP, <src>	X				2
POP <dest>	MOV <dest>, @SP; DEC SP	X				2
XCH A, <byte>	ACC and <byte> exchange data	X	X	X		1
XCHD A, @Ri	ACC and @Ri exchange low nibbles		X			1

Bảng 4-4. Bảng các <dest> và <src>.

src \ Dest	#data	A	Ri	direct	@Ri
A	X	X	X	X	X
Ri	X	X	X	X	X
Direct	X	X	X	X	X
@Ri	X	X	X	X	X

Từ bảng trên ta thấy:

- Không có toán hạng <dest> là #data của lệnh MOV cho RAM trong.
- Không có lệnh mà <dest> và <src> đồng thời là thanh ghi A hay Ri.
- Không thể dịch chuyển dữ liệu trực tiếp giữa hai thanh ghi Ri mà phải qua thanh ghi A. không thể dịch chuyển dữ liệu trực tiếp giữa ô nhớ địa chỉ gián tiếp @Ri với thanh ghi Ri hay giữa hai ô nhớ địa chỉ gián tiếp @Ri (phải qua thanh ghi A làm trung gian).

Ví dụ 4-4 dùng kiểm tra các chế độ địa chỉ cho lệnh MOV.

Ví dụ 4-5 dùng để ghi dữ liệu tăng dần từ 00H vào nhớ trong, địa chỉ từ 30H tới 7FH.

; Ví dụ 4-4: Kiểm tra các trường hợp loại trừ của lệnh MOV	
ORG 00H	
MOV A, A	; gửi dữ liệu của A cho chính nó
MOV R0, # 20H	; gửi #20h vào R0
MOV R2, R0	; gửi R0 cho R2
MOV R2, @R0	; gửi dữ liệu của ô nhớ địa chỉ R0 cho R2
MOV @R1, R2	; gửi dữ liệu của R2 cho ô nhớ địa chỉ R1
MOV @R1, @R2	; gửi dữ liệu của ô nhớ địa chỉ R2 cho ô nhớ @R1
SJMP \$	
END	
; Ví dụ 4-5: Ghi dữ liệu tăng dần từ 00H vào nhớ trong, địa chỉ từ 30H tới 7FH	
ORG 00H	
MOV R0, #30H	; nạp địa chỉ ô nhớ cần ghi dữ liệu vào thanh ghi R0
MOV A, #00H	; xóa thanh ghi A
LOOP: MOV @R0, A	; ghi A vào ô nhớ có địa chỉ R0
INC A	; tăng A
INC R0	; tăng địa chỉ nhớ
CJNZ R0, # 7FH, LOOP	; so sánh R0 với 7FH, nếu chưa bằng 00H, nhảy về LOOP
SJMP \$	; kết thúc chờ tại chỗ
END	; kết thúc chương trình dịch

2. Lệnh gán địa chỉ 16 bit cho địa chỉ nhớ ngoài cho con trỏ địa chỉ DPTR

Lệnh này có dạng:

MOV DPTR, #data (16 bit)

Lệnh này thay cho hai lệnh dịch chuyển 8 bit là MOV DPH, #data1 và MOV DPL, #data2.

3. Các lệnh PUSH /POP trao đổi tin với ngăn xếp

Lệnh này dùng để lưu giữ nội dung thanh ghi vào ngăn xếp khi có gọi chương trình con và lấy tin ra từ ngăn xếp cho thanh ghi, trước khi thực hiện lệnh RET, để trở về chương trình chính.

Ngăn xếp là vùng nhớ đệm, có địa chỉ là nội dung thanh ghi SP. Khi bật nguồn nuôi hay ấn nút Xóa (RESET), thanh ghi SP có giá trị mặc định là 07H, tức vùng nhớ IRAM có địa chỉ từ 07H trở đi được dùng làm vùng nhớ đệm ngăn xếp. Trong chương trình, nếu có các vectơ ngắt dùng cho các chương trình phục vụ ngắt, thường khởi động bằng cách gán cho SP giá trị ban đầu là #70H tức các ô nhớ có địa chỉ từ #70H trở đi được dùng cho ngăn xếp.

Khi gửi nội dung thanh ghi R vào ngăn xếp, ta dùng lệnh:

PUSH Ri hay PUSH A

Khi đó, nội dung thanh ghi được gửi vào ô nhớ dành cho ngăn xếp có địa chỉ là nội dung của thanh ghi SP và địa chỉ ngăn xếp hay nội dung thanh ghi con trỏ ngăn xếp SP tự động tăng +1.

Nếu có nhiều lệnh PUSH Ri liên tiếp, nội dung thanh ghi SP tự động tăng, tức nội dung các thanh ghi gửi liên tiếp vào các ô nhớ đệm ngăn xếp có địa chỉ tăng dần.

Khi lấy tin ra từ ngăn xếp hay trả lại giá trị cũ cho thanh ghi đã gửi vào, ta dùng lệnh:

POP Ri hay POP A

Khi đó, nội dung thanh ghi nhận nội dung của ô nhớ dành cho ngăn xếp có địa chỉ là nội dung của thanh ghi SP và địa chỉ ngăn xếp hay nội dung thanh ghi con trỏ ngăn xếp SP tự động giảm -1.

Nếu có nhiều lệnh POP R_i liên tiếp, nội dung thanh ghi SP tự động giảm, tức nội dung các thanh ghi gửi liên tiếp vào các ô nhớ đệm xếp có địa chỉ giảm dần.

Trong chương trình, gửi nhiều nội dung của các thanh ghi vào ngăn xếp, rồi lấy chúng ra để trả về thanh ghi cũ vừa gửi vào, phải theo quy tắc sau:

- Gửi vào trước, lấy ra sau, theo nguyên tắc Last In, First Out (LIFO).
- Lấy ra sau, gửi vào trước, theo nguyên tắc First In, Last Out (FILO).

Trong chương trình, người ta dùng cặp lệnh PUSH/POP R_i này trong các trường hợp sau:

- Giữ nguyên nội dung thanh ghi trước và sau khi gọi chương trình con.
- Chuyển giao tham số giữa các thanh ghi khi không muốn dùng lệnh MOV.

Ví dụ 4-6 minh họa trường hợp sử dụng các lệnh PUSH/POP trong một chương trình con tạo trễ.

Ví dụ 4-6: Lệnh PUSH/POP trong chương trình con

ORG 00H	
LOOP: MOV A, #55H	: gửi #55H vào A
MOV P0, A	: đưa ra cổng P0
CALL DELAY	: gọi trễ
MOV A, #0AAH	: đưa #0AAH vào A
CALL DELAY	: gọi trễ
JMP LOOP	: trở về LOOP
DELAY: PUSH A	: lưu trữ A vào ngăn xếp có địa chỉ là SP = 08H
PUSH PSW	: lưu trữ PSW vào ngăn xếp có địa chỉ là SP = 09H
MOV R1, #100H	: gửi #100H vào R1
HERE: DJNZ R1, HERE	: giảm R1, nếu chưa bằng 0, lại lặp lại
POP PSW	: trả về giá trị PSW từ ngăn xếp có địa chỉ 09H
POP A	: trả về giá trị A từ ngăn xếp có địa chỉ 08H
RET	: trở về chương trình chính
END	: kết thúc chương trình dịch

4. Các lệnh hoán chuyển (trao đổi dữ liệu cho nhau)

Trong lập trình, đôi khi cần trao đổi dữ liệu 8 bit hay 4 bit thấp giữa hai thanh ghi, giữa ô nhớ và thanh ghi, để tránh dùng một thanh ghi trung gian và nhiều lệnh MOV, người ta có các lệnh hoán chuyển (eXCHange) như sau:

- Hoán chuyển giữa hai thanh ghi hay <byte> nhớ:

XCH A, <byte> ; XCH A, <byte> hay XCH <byte>, <byte> và A

thực hiện hoán chuyển dữ liệu giữa A và <byte>, <byte>, <byte>.

- Hoán chuyển giữa thanh ghi và ô nhớ: có địa chỉ gián tiếp thanh ghi R_i, ô nhớ địa chỉ trực tiếp (con số chỉ địa chỉ nhớ) hay gián tiếp (thanh ghi chứa địa chỉ nhớ dạng @R_i).

XCHD A, @R_i hay XCHD @R_i, A

- Hoán chuyển 4 bit thấp giữa thanh ghi A và ô nhớ: có địa chỉ trong R_i.

Ví dụ 4-7 minh họa các lệnh hoán chuyển XCH và XCHD.

4.4.2. Nhóm lệnh dịch chuyển cho RAM ngoài (XRAM)

Với bộ nhớ RAM ngoài, ta cũng có các lệnh đọc bằng mã và lệnh MOVX với con trỏ địa chỉ DPTR (bảng 4-5) và ví dụ 4-8.

4.4.3. Nhóm lệnh ghi/đọc bảng theo địa chỉ chỉ số

Bảng 4-6 giới thiệu hai lệnh đọc vào thanh ghi A các thành phần của bảng dữ liệu chứa trong bộ nhớ chương trình ROM, có địa chỉ tăng dần theo chỉ số là A+PC hay A+DPTR.

Hai lệnh này có dạng tổng quát là:

MOVC A, @ A+ <base-reg>

Trong đó <base-reg> là các thanh ghi cơ sở PC hay DPTR.

Ví dụ 4-9, đọc bảng dữ liệu trong bộ nhớ RAM ngoài.

Chú ý: - Có thể thay cho số 0 ở cuối chuỗi dữ liệu bằng chèn thêm lệnh nạp số lần đọc bảng vào R2 và dùng lệnh DJNZ R2, BACK thay cho lệnh SJMP BACK.

- Có thể khai báo bảng MYDATA ngay sau chương trình, không cần khai báo vùng nhớ địa chỉ 200H cho MYDATA.

- Có thể đọc bảng (có ít phần tử) vào các thanh ghi R0-R7.

- Chương trình trên đọc bảng dữ liệu của ROM trong (vì kê khai bảng trong chương trình), tuy rằng dùng A+DPTR làm địa chỉ cho dữ liệu của bảng. Nếu đổi lệnh đọc MOVC A, @A+DPTR thành lệnh MOVX A, @DPTR thì sẽ không đọc được bảng có địa chỉ MYDATA ở địa chỉ 200H ở bộ nhớ ROM trong mà sẽ đọc ở địa chỉ 200H của bộ nhớ RAM ngoài. Muốn đọc được bảng trên, chương trình phải nạp bảng dữ liệu vào bộ nhớ RAM ngoài trước đã.

: Ví dụ 4-7: Các lệnh hoán chuyển XCH, XCHD

```
ORG 00H
MOV A, #0A5H ; gửi #0A5H vào A
MOV R0, # 5AH ; gửi #05AH vào R0
MOV P0, A ; đưa ra cổng P0
XCH A, R0 ; hoán chuyển 8 bit của A với R0
MOV P1, A ; đưa ra cổng P1
XCH R0, A ; hoán chuyển 8 bit của R0 với A
MOV P0, A ; đưa ra cổng P0
MOV P1, A ; đưa ra cổng P1
XCH A, R0 ; hoán chuyển 8 bit của R0 với A
XCHD R0, A ; hoán chuyển 4 bit thấp của R0 với A
MOV P0, A ; đưa A ra cổng P0
MOV A, R0 ; chuyển R0 vào A
MOV P1, A ; đưa R0 ra cổng P1
SJMP $
END
```

Bảng 4-5. Nhóm lệnh đọc bộ nhớ RAM ngoài

Độ rộng địa chỉ	Lệnh	Hành động	Thời gian thực hiện
8bit	MOVX A, @Ri	Đọc ra từ nhớ ngoài, địa chỉ ở Ri	2
8bit	MOVX @Ri, A	Ghi vào nhớ ngoài, địa chỉ ở Ri	2
16bit	MOVX A, @DPTR	Đọc ra từ nhớ ngoài, địa chỉ ở DPTR	2
16bit	MOVX @DPTR, A	Ghi vào nhớ ngoài, địa chỉ ở DPTR	2

Bảng 4-6. Bảng lệnh đọc bảng trong ROM

Lệnh	Hành động	Thời gian thực hiện
MOVC A+@A+PC	Đọc nhớ chương trình ở địa chỉ A+PC vào A	2
MOVX A, @A+DPTR	Đọc nhớ chương trình ở địa chỉ A+DPTR vào A	2

Ví dụ 4-10, giới thiệu chương trình đọc bảng dữ liệu trong bộ nhớ chương trình PROM bên trong.

Đây chính là bảng biến đổi mã HEX hay BCD (từ 0 tới 9D) ra mã 7 đoạn để chỉ thị số thập phân 0 tới 9 ra các đèn LED 7 đoạn, thay cho vi mạch biến đổi mã 3-7 đoạn.

So với chương trình ở ví dụ 4-9, chương trình của ví dụ 4-10 này có những đặc điểm sau:

- Đọc (do lệnh MOVC A, @A+ PC) các dòng dữ liệu tùy theo giá trị tăng dần của PC thay cho việc đọc (do lệnh MOVX A, @A+ DPTR) một dòng hay một chuỗi ký tự, có địa chỉ do A và DPTR quyết định.

<i>; Ví dụ 4-8: Ghi dữ liệu tăng dần từ 00H vào nhớ ngoài, địa chỉ từ 1000H tới 1200H</i>	<i>; Ví dụ 4-9: Chương trình đọc bảng dữ liệu ở ROM trong</i>
<pre> ORG 00H MOV DPTR,#1000H ; nạp 1000H cho PTR MOV R0,#200H ; nạp số ô nhớ cần ghi MOV A, #00H ; xóa thanh ghi A LOOP:MOVX @DPTR,A; ghi A vào DPTR INC A ; tăng A INC DPTR ; tăng địa chỉ nhớ DJNZR0, LOOP SJMP \$; kết thúc chờ tại chỗ END ; kết thúc chương trình dịch </pre>	<pre> ORG 00H MOV DPTR,#MYDATA ; nạp địa chỉ bảng BACK: CLR A ; xóa A = 0 MOVC A, @A+DPTR; đọc bảng JZ KT ; nhảy tới kt nếu gặp 0 MOV P0, A ; đọc bảng ra cổng P0 INC DPTR ; tăng địa chỉ bảng SJMP BACK ; trở về Back để đọc lại KT: SJMP KT ; kết thúc chờ ORG 200H ; địa chỉ bắt đầu của bảng MYDATA: DB "HA NOI", 0; bảng, kết thúc 0 END ; kết thúc dịch chương trình </pre>

- Bảng dữ liệu này nằm trong chương trình, trong bộ nhớ PROM.
- Ngoài bảng biến đổi mã trên, trong chương trình, còn có bảng dữ liệu khác như bảng mã của chương trình, bảng dữ liệu cho sự biến đổi số sang tín hiệu tương tự cho vi mạch biến đổi số-tương tự DAC v.v...
- Có thể ghi bảng mã trên vào bộ nhớ RAM ngoài rồi đọc bằng lệnh MOVX A, @DPTR

4.5. LỆNH GỘP VÀ LẬP TRÌNH CHO BỘ NHỚ, THANH GHI

Với bộ nhớ, ta cũng có thể viết các lệnh gộp và lập trình với lệnh gộp cho chương trình đơn giản và tiện dụng.

4.5.1. Lệnh gộp cho thanh ghi và bộ nhớ

1. Lệnh gộp cho thanh ghi

Lệnh ghi cho thanh ghi có dạng như sau:

```

macro @ghithg <dulieu?, thanhghi?>
    mov a, #l1
    mov l2, a
endmac

```

Lệnh đọc cho thanh ghi có dạng như sau:

```

macro @docthg < thanhghi?>
    mov a, l1
endmac

```

Ví dụ 4-11 giới thiệu chương trình xóa, ghi và đọc thanh ghi nếu các lệnh gộp ghi, đọc thanh ghi có trong thư viện MYLIB.ASM.

2. Lệnh gộp cho bộ nhớ trong (IRAM) và thanh ghi đặc biệt SFR

Một ô nhớ trong hay thanh ghi thông dụng đều được đọc theo *phương pháp địa chỉ trực tiếp* (địa chỉ là con số sau lệnh, không có dấu #) hay *gián tiếp thanh ghi* (thanh ghi R0 hay R1 chứa địa chỉ ô nhớ hay SFR).

Ví dụ MOV A, 30H; gọi địa chỉ trực tiếp.

Hay: MOV R0, #30H; ghi địa chỉ vào R0

và MOV A, @R0; gọi địa chỉ gián tiếp thanh ghi R0 (hay R1).

Tương tự, ta viết được lệnh gộp.

; Ví dụ 4-10: Chương trình biến đổi 7 mã BCD (số 0-6) thành mã 7 đoạn	; Ví dụ 4-11: Dùng lệnh gộp chương trình ghi/đọc thanh ghi
<pre> ORG 00H MOV R0, #5 ; khoảng cách từ PC tới đầu bảng MOV R7, #7 ; số phần tử của bảng BDOI: MOV A, R0 ; nạp R0 vào A MOVC A, @A+PC ; đọc bảng vào A MOV P0, A ; đưa ra cổng P0 INC R0 ; tăng R0 DJNZ R7, BDOI ; giảm R7, về BIENDO DB 3FH ; mã 7 đoạn của số 0 BCD DB 06H ; mã 7 đoạn của số 1 BCD DB 5BH ; mã 7 đoạn của số 2 BCD DB 4FH ; mã 7 đoạn của số 3 BCD DB 66H ; mã 7 đoạn của số 4 BCD DB 6DH ; mã 7 đoạn của số 5 BCD DB 7DH ; mã 7 đoạn của số 6 BCD SJMP \$; chờ tại chỗ END </pre>	<pre> \$include mylib.asm org 00h @ghithg 55h, 00h ; ghi 55h vào R0 @docthg 01h ; đọc R1 @ghithg 00h, 80h ; xoá địa chỉ 80h (cổng P0) @docthg 80h ; đọc địa chỉ 80h (cổng P0) @ghithg 55h, 90h ; ghi 55h vào địa chỉ 90h (cổng P1) @docthg 90h ; đọc thanh ghi 90h hay cổng P1 @ghithg 0F5h 0a0h ; ghi 0F5h vào địa chỉ 0A0h (cổng P2) @docthg 0a0h ; đọc cổng P2 @ghithg 05h, 0B0h ; ghi 05h vào địa chỉ 0B0h (cổng P3) @docthg 0B0h ; đọc địa chỉ 0B0h (cổng P3) SJMP \$ END </pre>

a) Ghi dữ liệu (tham số 1) vào ô nhớ địa chỉ R0

Lệnh gộp là:

```

MACRO @GIIM <dữ liệu?, địa chỉ?>
    MOV A, #1 ; ghi dữ liệu vào A
    MOV @R0, A ; ghi dữ liệu theo phương pháp địa chỉ gián tiếp
ENDMAC

```

b) Đọc dữ liệu từ ô nhớ địa chỉ (tham số 1)

Lệnh gộp là:

```

MACRO @docM <dữ liệu?, địa chỉ?>
    MOV R0, #1 ; ghi địa chỉ vào R0
    MOV @R0, A ; ghi dữ liệu theo phương pháp địa chỉ gián tiếp
ENDMAC

```

Ví dụ 4-12 giới thiệu chương trình xoá, ghi, chuyển vùng nhớ rồi đọc khối nhớ theo địa chỉ gián tiếp thanh ghi.

; Ví dụ 4-12: Chương trình xoá ghi, chuyển vùng nhớ và đọc ra cổng P0	
<pre> org 00h ljmp main org 100h main: mov r0, #30h ; địa chỉ đầu mov r7, #10h ; số lần ghi/đọc lap1: mov a, #00h ; xoá ô nhớ mov @R0, a ; xoá r0 r7 lần inc r0 djnz r7, lap1 mov r0, #30h ; địa chỉ đầu để ghi vào ô nhớ lap2: mov r2, #45h mov a, r2 mov @r0, a ; ghi vào ô nhớ địa chỉ từ 30h 10h lần </pre>	<pre> inc r0 inc r2 ; dữ liệu tăng djnz r7, lap2 mov r0, #30h ; địa chỉ đầu để đọc và chuyển vùng nhớ mov r1, #50h ; địa chỉ truyền mov r7, #10h ; số lần ghi/đọc lap3: mov a, @r0 ; đọc r0 để đọc vùng nhớ ra p0 mov P0, a ; đưa ra mov P1, #00H ; đưa ra cổng P0 mov @R1, A ; ghi vào vùng nhớ có địa chỉ R0 inc r0 ; tăng r0 inc r1 ; tăng r1 djnz r7, lap3 ; giảm r7, nếu r7 khác 0, nhảy về lap3 sjmp \$ end </pre>

3. Lệnh gộp cho bộ nhớ PROM bên trong

a) Ghi dữ liệu

Với bộ nhớ chương trình bên trong hay ngoài, ghi dữ liệu (dạng chuỗi ký tự hay số, ví dụ bản biến đổi mã thập phân sang mã 7 đoạn (ví dụ 4-13) bằng khai báo chuỗi ký tự hay khai báo bảng với nhiều byte HEX chữ số.

Với chuỗi thông báo (ví dụ 4-9), đặt ngay trong chương trình, ngay cuối chương trình (sau lệnh SJMP \$ và trước chỉ dẫn END) và chỉ cần khai báo:

Nhan : db 'Hà Nội', 00H số 00H chỉ kết thúc chuỗi.

Với bảng dữ liệu có nhiều số mã ASCII cho cả ký tự và chữ số, người ta có thể khai báo riêng và đặt trong vùng nhớ PROM khác xa với vùng nhớ của chương trình vì có thể nạp vào và thay đổi sau khi viết chương trình.

Org xxxx
Nhan : db yy yy yy yy yy ,
db yy yy yy yy yy yy , 00H , 00H

Với xxxx là địa chỉ đầu của bảng. Địa chỉ này có thể nhỏ hơn 8 bit cho PROM trong hay 16 bit cho PROM ngoài. Khi đọc, để nhận biết kết thúc bảng, có thể đếm hết số phần tử bảng hay nhận biết nhiều mã của số 00H liên tiếp (ít nhất là hai chuỗi số 00H).

b) Đọc dữ liệu

Có thể đọc chuỗi hay bảng đều bằng lệnh MOVC A, @A+DPTR như ví dụ 4-10 hay 4-13. Khi nhận biết kết thúc bảng, việc đọc sẽ ngừng lại. Có thể kết hợp việc đọc và chuyển vùng nhớ sang nhớ RAM trong với địa chỉ thấp cho gần với chương trình như ở ví dụ 4-13. Trong các bộ đếm dùng đèn chỉ thị LED 7 đoạn, dữ liệu nhị phân trong A sẽ được biến đổi thành mã 7 đoạn bằng cách tìm trong bảng mã số mã tương ứng để đưa ra chỉ thị (xem ở dưới về giải mã).

<i>Ví dụ 4-13: Bảng biến đổi mã BDOIMA7, từ địa chỉ DPTR=230h sang RAM trong từ 40h</i> <pre> MACRO @@ bdoima7 <địa chỉ cao?, địa chỉ mã mới> MOV DPH, #02h MOV DPL, #30h MOV R0, #1 ; địa chỉ vùng mã mới BACK: MOV A, #00H MOVC A, @A+DPTR ; đọc phần tử của bảng vào A MOV @R0, A INC R0 INC DPTR ; tăng địa chỉ bảng CJNE A, #3FH, \$+3 CLR A ; xóa A = 0 MOVC A, @A+DPTR ; đọc phần tử của bảng vào A MOV @R0, A INC R0 INC DPTR ; tăng địa chỉ bảng CJNE A, #3FH, BACK ; xét kết thúc bảng = 0h, xét xem có 0H tiếp theo? </pre>	<pre> ENDMAC KT: SJMP \$; chờ tại chỗ hay kết thúc bản ORG 230h; địa chỉ bắt đầu của bảng biến đổi mã BANG: DB 3FH ; mã 7 đoạn của số 0 BCD DB 06H ; mã 7 đoạn của số 1 BCD DB 5BH ; mã 7 đoạn của số 2 BCD DB 4FH ; mã 7 đoạn của số 3 BCD DB 66H ; mã 7 đoạn của số 4 BCD DB 6DH ; mã 7 đoạn của số 5 BCD DB 7DH ; mã 7 đoạn của số 6 BCD DB 07H ; mã 7 đoạn của số 7 BCD DB 7FH ; mã 7 đoạn của số 8 BCD DB 6FH ; mã 7 đoạn của số 9 BCD DB 3FH ; mã 7 đoạn của số 0 BCD DB 3FH ; mã 7 đoạn của số 0 BCD org 00h @bdoima7 40h ; địa chỉ bắt đầu ; của bảng mã đã ghi lại ở RAM trong sjmp \$ end </pre>
---	---

4. Lệnh gộp cho bộ nhớ ngoài (RAM, PROM)

Với bộ nhớ ngoài (cả RAM hay PROM) đều dùng con trỏ địa chỉ DPTR 16 bit. Khi nạp địa chỉ vào DPTR, ta có thể nạp hai lần cho hai byte cao (DPH) và thấp (DPL). Việc ghi chương trình hay dữ liệu cũng tương tự PROM trong. Việc đọc RAM ngoài được tiến hành bởi lệnh:

MOVX A, @DPTR

Ví dụ 4-14 giới thiệu lệnh gộp và chương trình cho RAM ngoài.

<i>Ví dụ 4-14: Lệnh gộp ghi x và đọc x cho nhớ ngoài (RAM và ROM)</i>	MOV DPH, #1 ; địa chỉ đầu cao MOV DPL, #12 ; địa chỉ đầu thấp MOVX A, @DPTR MOV P0, A ENDMAC ORG 00h @GHIX 01, 40H, 58H @DOCX 01, 40H @DOCX 01, 41H @DOCX 01, 42H @DOCX 01, 43H @DOCX 01, 43H @DOCX 01, 43H SJMP \$ END
MACRO @GHIX <địa chỉ đầu cao? ; địa chỉ đầu thấp?, dữ liệu?, số ô nhớ?> MOV DPH, #1 ; địa chỉ đầu cao MOV DPL, #12 ; địa chỉ đầu thấp MOV A, #13 MOV R0, #14 LAP: MOVX @DPTR, A INC DPTR DJNZ R0, LAP ENDMAC MACRO @DOCX <địa chỉ đầu cao? ; địa chỉ đầu thấp?>	

4.5.2. Một số chương trình ứng dụng lệnh gộp cho thanh ghi và bộ nhớ

Với lệnh gộp cho thanh ghi và bộ nhớ, chúng ta có thể viết lại một số chương trình đã viết ở trên. Chúng ta sẽ thấy các chương trình với lệnh gộp này sẽ ngắn gọn vì có thể truyền tham số cho lệnh gộp ngay trong khai báo, còn khi dùng chương trình con, phải gán tham số trước rồi mới gọi chương trình con.

1. Chương trình cho đồng hồ đếm thời gian

Ở chương 3, khi giới thiệu vòng lặp, chúng ta đã ví dụ chương trình của đồng hồ đếm thời gian dùng thanh ghi. Tại chương này, chúng ta viết lại chương trình trên với các ô nhớ chứa số đếm về thời gian với lệnh gộp và cả giải mã và hiển thị.

Đơn vị thời gian cơ sở ở đây là giây và dùng phương pháp đếm trễ (DELAY). Có thể thay đổi thời gian đơn vị này nhỏ hơn cho các đồng hồ chính xác hơn hay thời gian nhỏ hơn 1 giây.

Các số đếm về giây, chục giây, phút, chục phút, giờ, chục giờ được nhớ trong các ô nhớ tương ứng, được tăng lên sau một thời gian tương ứng và được ghi lại vào ô nhớ dành cho chúng. Các vòng lặp lồng nhau về giây, phút, giờ sẽ lần lượt lặp lại nhiều lần. Chúng ta có thể theo dõi sự thay đổi của thời gian qua các ô nhớ RAM hay đưa ra các cổng P0, P1, P2 và P3.

Ví dụ 4-15 giới thiệu chương trình đồng hồ thời gian dùng lệnh gộp.

Vì các bộ đếm thời gian đều dùng chung lệnh gộp đếm < địa chỉ nhớ ?, giới hạn ?, nhãn?> vì có các điểm chung:

- Chứa trong các ô nhớ, nên tham số 1 là địa chỉ trực tiếp của ô nhớ tương ứng.

- Các bộ đếm đều có so sánh với 10 nên tham số 2 là chục (#10), với sáu chục (#6) cho giây và phút, với 4 và 2 chục cho giờ.

- Để nhảy trở về các nhãn tương ứng với tham số 3 để bắt đầu đếm lại từ 1 giây.

Chú ý: - Chương trình trên dùng đơn vị thời gian là 1 giây. Có thể dùng đơn vị thời gian nhỏ hơn (cỡ mili giây). Khi đó cần thời gian đếm trễ nhỏ hơn và nhiều ô nhớ hơn.

- Chương trình trên dùng các ô nhớ cho các đơn vị đếm giây hay chục giây v.v. riêng rẽ nên tốn ô nhớ. Để tiết kiệm ô nhớ, có thể dùng một ô nhớ cho cả hai bộ đếm giây và chục giây, khi đó, phải so sánh với 60 để chuyển sang bộ đếm phút và chục phút. Như vậy, một đồng hồ đếm thời gian tới 24 giờ chỉ cần 3 ô nhớ cho giờ, phút, giây thay cho 6 ô nhớ.

- Chương trình trên đưa trực tiếp số đếm gồm các chữ số thập phân (tới 9 cho hàng đơn vị và tới 6 cho hàng chục) ra các cổng P0 tới P1. Nếu dùng một ô nhớ cho hai số đếm hàng chục và hàng đơn vị, ta phải:

- + *Biến đổi số đếm dạng HEX sang dạng thập phân:* nếu đưa trực tiếp ra cổng song song phải qua bộ giải mã HEX sang mã BCD (xem chương 6).

- + *Biến đổi số đếm dạng HEX sang dạng mã 7 đoạn:* nếu đưa số đếm ra đèn LED 7 đoạn mắc với các cổng song song (xem chương 8), ta phải qua bộ giải mã 7 đoạn.

- Một đồng hồ đếm giây hoàn chỉnh phải có bộ đếm thời gian đơn vị dùng vòng lặp đếm trễ (kém chính xác) hay dùng bộ định thời Timer (chính xác hơn) và bộ giải mã và hiển thị trên các đèn LED 7 đoạn (xem tập II).

; Ví dụ 4-15: DHM - Đồng hồ đếm giây, chục giây, phút, chục phút, giờ, chục giờ	
MACRO @DEM < địa chỉ nhớ?, so sánh?, nhân?>	gio1: mov 32h, #00h
inc ll : tăng giá trị trong ô nhớ	chphut1: mov 33h, #00h
mov a, ll : gửi vào a	phut1: mov 34h, #00h
cjne a, #12, 13 : so sánh với dải, nhảy về nhãn	chgiay1: mov 35h, #00h
ENDMAC	giay1: lcall ht ; gọi hiển thị
org 00h ; bắt đầu chương trình	lcall delay1s ; gọi trễ 1 giây
ljmp main ; nhảy tới main	@DEM giay, #10, giay1 ; đếm đơn vị giây
chgio equ 30h ; gán ô nhớ cho chục giờ	@DEM chgiay, #6, chgiay1 ; đếm chục giây
gio equ 31h ; gán ô nhớ cho giờ	@DEM phut, #10, phut1 ; đếm đơn vị phút
chphut equ 32h ; gán ô nhớ cho chục phút	@DEM chphut, #6, chphut1 ; đếm chục phút
phut equ 33h ; gán ô nhớ cho phút	@DEM gio, #12, gio1 ; đếm đơn vị giờ
chgiay equ 34h ; gán ô nhớ cho chục giây	@DEM chgio, #3, chgio1 ; đếm chục giờ
giay equ 35h ; gán ô nhớ cho giây	sjmp main
org 100h ; bắt đầu chương trình chính	delay1s: mov r1, #10 ; trễ 1 giây
main: mov 30h, #00h ; xoá các ô nhớ	lap: mov r2, #100
chgio1: mov 31h, #00h	lap1: mov r3, #250
gio1: mov 32h, #00h	nop
chphut1: mov 33h, #00h	lap2: nop
phut1: mov 34h, #00h	djnz r3, lap2
chgiay1: mov 35h, #00h	djnz R2, lap1
giay1: lcall ht ; gọi hiển thị	djnz R1, lap
lcall delay1s ; gọi trễ 1 giây	ret
org 100h ; bắt đầu chương trình chính	ht: mov P0, gio ; hiển thị giờ
main: mov 30h, #00h ; xoá các ô nhớ	mov P1, chphut ; hiển thị chục phút
chgio1: mov 31h, #00h	mov P2, phut ; hiển thị phút
	mov P3, chgiay ; hiển thị chục giây
	ret
	end ; kết thúc dịch

Ví dụ 4-16: DII.7 - giới thiệu chương trình cho đồng hồ đếm tới ngày, có hiển thị trên LED 7 đoạn.

2. Chương trình tạo bảng dữ liệu

Ví dụ 4-13 giới thiệu chương trình tạo bảng dữ liệu trong đó có bảng các mã 7 đoạn cho các chữ số thập phân (từ 0 tới 9) để đưa ra hiển thị các con số thập phân trên đèn LED 7 đoạn.

Tuỳ loại đèn LED anốt chung (sáng khi các lối vào dữ liệu bằng 0) hay katốt chung (sáng khi lối vào dữ liệu bằng 1), ta có giá trị của các số (dạng HEX trong một byte) khác nhau. Trong chương trình của ví dụ 4-13 là bảng mã cho đèn katốt chung. Ta có thể đổi bảng mã này cho đèn LED 7 đoạn có anốt chung bằng cách lấy bù 15 cho các phần tử của bảng trên. Bảng mã được dịch và ghi vào bộ nhớ PROM ngay sau khi hợp dịch chương trình (quan sát được nhờ thực đơn View/Code Memory).

Bảng mã được dịch và ghi vào bộ nhớ PROM ngay sau khi hợp dịch chương trình (quan sát được nhờ thực đơn View/Code Memory).

Chương trình trên, ngoài tạo bảng mã, còn đọc bảng và ghi chuyển sang bộ nhớ RAM bên trong ở địa chỉ 40h. Ta có thể thay bảng mã 7 đoạn cho đèn LED katốt chung bằng chuỗi (ghi vào sau nhãn BANG) như sau: 0C0H, 0F9H, 0A4H, 0B0H, 99H, 92H, 82H, 0F8H, 80H, 90H.

Ví dụ 4-16 giới thiệu chương trình biến đổi mã trong đồng hồ hiện số trên LED 7 đoạn.

3. Chương trình hiển thị trên LED7 đoạn

Tuỳ loại đèn LED anốt chung (sáng khi các lối vào dữ liệu bằng 0) hay katốt chung (sáng khi lối vào dữ liệu bằng 1) ta có giá trị của các số (dạng HEX trong một byte) khác nhau. Trong chương trình dùng đèn katốt chung với mã tương ứng như bảng mã. Ta có thể đổi bảng mã này cho đèn LED7 đoạn có anốt chung bằng cách lấy bù 15 cho các phần tử của bảng trên.

Có bảng biến đổi mã, mỗi lần muốn đưa một số ra hiển thị trên đèn LED 7 đoạn, ta phải:

- Đọc số cần hiển thị vào thanh ghi chứa A.
- Đọc phần tử có địa chỉ A+ DPTR (bảng trong PROM).
- Đưa dữ liệu đã đọc ra cổng song song cho dữ liệu chung của LED 7 đoạn (hiển thị kiểu dồn kênh cho đỡ tốn giải mã).

<i>; Ví dụ 4-16: DHL7- Chương trình cho đồng hồ đếm giây tới ngày, hiện đèn LED 7 đoạn</i> <pre> macro @gma <ô nhớ chứa a đếm?, ô nhớ chứa, ; mã?> mov dptr, #ma7 mov a, 11 movc a, @a+dptr mov 12, a endmac org 00H ljmp main \$include MYLIB.ASM chgiao equ 30h gioequ 31h chphut equ 32h phut equ 33h chgiay equ 34h giay equ 35h org 100h main: mov 30h, #00h chgiao: mov 31h, #00h gio: mov 32h, #00h chphut: mov 33h, #00h phut: mov 34h, #00h chgiay: mov 35h, #00h giay: lcall gma lcall hthi lcall delay1s ; trễ 1 giây, đếm thời gian cjne r0, #59h, ht1 ret delay1s: mov r1, #10 ; trễ 1 giây lap: mov r2, #100 </pre>	<pre> lap1: mov r3, #250 nop @DEM giay, #10, giay1 @DEM chgiay, #6, chgiay1 @DEM phut, #10, phut1 @DEM chphut, #6, chphut1 @DEM gio, #4, gio1 ; @DEM chgio, #2, chgio1 ljmp chgio1 ma7: db 03fh, 06h, 5bh, 4fh, 66h, 6dh, 7dh, db 07h, 7fh, 6fh gma: nop @gma 35h, 55h @gma 34h, 54h @gma 33h, 53h @gma 32h, 52h @gma 31h, 51h @gma 30h, 50h ret hthi: mov a, #7fh mov r0, #50h ht1: mov P0, @r0 mov P1, a lcall delay1s ; trễ 1 giây để quan sát đèn LED inc r0 rr a lap2: nop djnz r3, lap2 djnz R2, lap1 djnz R1, lap ret end </pre>
--	---

- Đưa tín hiệu điều khiển LED 7 đoạn sáng trên một cổng song song khác (xem tập II về ghép nối LED 7 đoạn) với một thời gian trễ đủ để quan sát. Với thay đổi cách điều khiển trên cổng này, người ta có thể điều khiển các LED sáng đồng thời (quay vòng theo thời gian, mắt không phát hiện được), sáng kiểu chạy (các đèn từ trái qua phải hay ngược lại).

- Hay kiểu dồn (từ hai bên vào trong) hay kiểu phản tán (từ trong ra hai bên).

Ví dụ 4-16 giới thiệu chương trình điều khiển hiển thị LED 7 đoạn có katốt chung với tín hiệu điều khiển đưa ra mức 0 vào tranzito điều khiển LED sáng. Mỗi lần hiển thị, phải hiển thị lần lượt các LED cho các ô nhớ chứa tất cả các số đếm của các địa chỉ 35h, 34h, 33h, 32h, 31h, 30h (cho giây tới chục giờ), trong khi đó, các ô nhớ 55h, 54h, 53h, 52h, 51h, 50h chứa các mã 7 đoạn tương ứng. Như vậy, mắt người quan sát các đèn LED 7 đoạn, thấy các đèn đều sáng (vì quét với tần số lớn nên mắt người không phân biệt được sự sáng hay tắt liên tục). Chỉ các đèn có số đếm khác không mới chỉ số đếm, còn các đèn còn lại (ô nhớ tương ứng chưa tăng) vẫn chỉ số 0.

Nếu kết hợp cả việc định thời gian trễ hay báo chuông, ta có một đồng hồ hoàn chỉnh, vừa đếm được thời gian, vừa báo thức. Việc định thời gian trễ hay báo hiệu thời gian này phải đưa vào từ bàn phím (cho tiện dụng), chứ không đặt trước bằng chương trình. Ở chương ghép nối bàn phím (tập II), chúng ta sẽ đề cập vấn đề này.

4. Chương trình tìm kiếm dữ liệu

Trong VĐK không có những lệnh xử lý chuỗi (dịch chuyển, nạp vào lấy ra và quét chuỗi dữ liệu gồm ký tự và số mã) như trong máy vi tính. Nhưng với các lệnh MOV, CJNE và DJNZ, chúng ta có thể lập trình thành lệnh gộp xử lý chuỗi như các lệnh xử lý chuỗi tương ứng như trong vi xử lý của máy vi tính. Ví dụ, chúng ta có thể viết được các lệnh gộp để tìm kiếm một ký tự, một chuỗi ký tự hay so sánh dữ liệu của hai vùng nhớ đã cho.

a) Tìm một dữ liệu

Một dữ liệu hay một ký tự có thể được tìm thấy trong một khối dữ liệu nằm trong một vùng nhớ với địa chỉ đầu và cuối nào đó. Ví dụ 4-17 giới thiệu lệnh gộp và chương trình tìm một ký tự mong muốn.

Lệnh gộp có hai tham số: - địa chỉ đầu của vùng nhớ hay bảng dữ liệu (là 200h).

- ký tự cần tìm là chữ 'ơ'.

Lệnh gộp có dùng lệnh đọc nhớ PROM ngoài *MOVC A, @A+DPTR* và lệnh so sánh và rẽ nhánh chương trình *CJNE a, b, lap*. Mỗi lần gặp ký tự cần tìm, chương trình ghi lại số lần tìm thấy (trong R1), địa chỉ ký tự tìm thấy (trong các ô nhớ từ địa chỉ 30h). Chương trình trên, chỉ kết thúc khi kết thúc bảng khi gặp dữ liệu 0.

Trong ví dụ trên, để tìm ký tự trong một vùng dữ liệu, ta tạo một bảng dữ liệu với địa chỉ đầu là 200h với dữ liệu là câu 'ơ anh Bơ, đi chơi hồ Bơ?', 00 (xem cách tạo bảng ở trên). Khi chạy chương trình, nội dung thanh ghi r1 cho biết số lần tìm thấy (4) và sẽ nạp vào khối nhớ địa chỉ có chữ 'ơ' (1, 5, 0DH, 19H).

; Ví dụ 4-17: Tìm 1 ký tự (chữ trong câu 'ơ anh Bơ đi chơi hồ Bơ?', 00)		
macro @bg <địa chỉ bảng, dữ liệu của bảng>		mov @r0, a ; ghi lại địa chỉ thấp
org 1 ; lệnh gộp tạo bảng		mov a, dpl ; ghi lại địa chỉ thấp
bang: db 2 ; địa chỉ bảng		mov @r0, a ; ghi lại địa chỉ thấp
endmac ; dữ liệu bảng		inc r1 ; tăng số lần tìm thấy
		inc r0 ; tăng địa chỉ nhớ
macro @timlch <địa chỉ bảng?, dữ liệu tìm?>		lap1: inc dptr
mov dptr, # 1 ; địa chỉ bảng		sjmp lap ; trở về so sánh nếu chưa kt
mov a, # 2 ; chữ cần tìm		kt: nop ; kết thúc
mov b, a		endmac ; kết thúc lệnh gộp
lap: mov a, #00h ; xoá a		
movc a, @a+dptr ; đọc bảng		org 00h ; bắt đầu chương trình chính
cjne a, #00h, tiep ; so sánh với 0		mov r1, #00h ; xoá số lần tìm thấy
sjmp kt ; nhảy tới kết thúc		mov r0, #30h ; gán địa chỉ RAM trong
tiep: ; đọc tiếp nếu chưa gặp 0		@timlch 200h, 'ơ' ; lệnh tìm chữ 'ơ'
cjne a, b, lap1 ; so sánh a với b, về lap1		; trong bảng có địa chỉ là 200h ở ROM
mov P0, a ; nếu a = b, đưa ra P0 mã		sjmp \$
mov a, dpl ; ghi lại địa chỉ thấp		@bg 200h, 'ơ anh Bơ, đi chơi hồ Bơ?', 00
		; lệnh tạo bảng dữ liệu, địa chỉ 200h
		end ; kết thúc dịch chương trình

Nếu không cần tạo bảng làm ví dụ, mà tìm dữ liệu trong vùng nhớ có sẵn dữ liệu, ta cần bổ sung thêm tham số hay viết trong chương trình số lượng dữ liệu cần xét. Nếu lệnh gộp 'timlch < địa chỉ đầu bảng ?, ký tự cần tìm ?>' này có trong thư viện với kê khai \$include MYLIB.ASM thì chương trình chỉ cần 5 dòng lệnh. Điều này chứng minh ưu điểm của phương pháp lập trình bảng lệnh gộp.

b) Tìm một chuỗi dữ liệu

Một chuỗi dữ liệu, trong đó có chuỗi ký tự, được tìm ra nếu chúng ta dùng liên tiếp các lệnh gộp tìm một ký tự 'timlch' ở trên. Ví dụ 4-18 giới thiệu chương trình tìm một chuỗi ký tự, ví dụ 'Bơ đi chơi đâu đấy? hồ Bơ?'

Ta chú ý rằng: Lệnh gộp tìm một ký tự cho một chuỗi này phức tạp hơn tìm một ký tự với 4 tham số là <địa chỉ bảng?, dữ liệu tìm?, địa chỉ ghi mã cần tìm?, địa chỉ ghi địa chỉ chữ tìm thấy?>

Chuỗi ký tự cần tìm được viết dạng chữ hoa ('CHƠI').

Kết quả của chương trình là phát hiện chuỗi 'CHƠI' với mã liên tiếp ở địa chỉ nhớ RAM trong tương ứng.

c) So sánh hai bảng dữ liệu

Ví dụ 4-19 giới thiệu chương trình so sánh hai bảng dữ liệu. Do đó, phải tạo hai bảng có các dữ liệu giống và khác nhau. Nếu so sánh hai vùng nhớ có dữ liệu sẵn, không phải có lệnh tạo bảng. Có thể cho trước số ô nhớ (hay số phần tử của bảng), khi đó dùng một biến đếm thanh ghi và dùng lệnh DJNZ R, lap.

<i>Ví dụ 4-18: Tìm 1 chuỗi dữ liệu</i>	
<pre> macro @bg <địa chỉ, dữ liệu của bảng> org l1 ; địa chỉ bảng bang: db l2 ; dữ liệu bảng endmacro macro @tim1ch <địa chỉ bảng?, dữ liệu tìm? ; địa chỉ ghi mã tìm?, địa chỉ ghi địa chỉ chữ ; thấy? > mov dptr, #l1 ; địa chỉ bảng mov a, #l2 ; chữ cần tìm mov b, a mov r0, #l3 ; địa chỉ ghi mã cần tìm mov r1, #l4 ; địa chỉ ghi địa chỉ chữ thấy lap: mov a, #00h ; xóa a movc a, @a+dptr ; đọc bảng cjne a, #00h, tiep ; so sánh với 0 kết thúc sjmp kt ; nhảy tới kết thúc tiep: ; đọc tiếp nếu chưa gặp 0 cjne a, b, tiep1 ; so sánh a với b, về lặp jnc ghi </pre>	<pre> tiep1: inc dptr sjmp lap ghi: mov P0, a ; ra cổng P0 mã cần tìm mov @r0, a ; ghi lại chữ tìm thấy mov @r1, dpl; ghi địa chỉ chữ tìm thấy inc r7 ; tăng số lần tìm thấy inc r1 ; tăng địa chỉ tìm thấy inc r0 inc dptr sjmp lap ; trở về lặp kt: nop ; kết thúc endmacro ; kết thúc lệnh gộp org 00h ; chương trình chính @tim1ch 200h, 'C', 30H, 40H; tìm chữ C @tim1ch 200h, 'H', 31H, 41H; tìm chữ H @tim1ch 200h, 'O', 32H, 42H; tìm chữ 'O' @tim1ch 200h, 'T', 33H, 43H; tìm chữ 'T' sjmp \$ @bg 200h, 'Bơ, đi CHƠI đâu đấy? hồ Bơ?'. ; 00, 00, 00 ; tạo bảng dữ liệu, địa chỉ 200h end ; kết thúc dịch chương trình </pre>

Cách so sánh hai phần tử của bảng 1 và bảng 2 là:

- Đọc các phần tử của bảng 1 và gửi vào thanh ghi chứa b.
- Đọc các phần tử của bảng 2.
- Nếu kết quả đọc vào bằng 0, sẽ kết thúc so sánh.
- So sánh a với b. Nếu bằng nhau, trở về so sánh tiếp. Nếu kết quả phép trừ khác 0, ghi lại địa chỉ lỗi và đếm số lỗi sai.
- Tăng địa chỉ thấp của cả hai bảng và so sánh tiếp.

Khi cho chạy thử trên chương trình mô phỏng Pinnacle, ta có thể quan sát các thanh ghi, các bộ nhớ mã (PROM), nhớ trong (IRAM) và nhớ ngoài (XRAM) biết kết quả của chương trình.

<i>Ví dụ 4-19: So sánh hai bảng dữ liệu</i>	
<pre> macro @bg <địa chỉ của bảng?, dữ liệu của bảng?> ; lệnh gộp tạo bảng org l1 ; địa chỉ bảng bang: db l2 ; dữ liệu bảng endmacro macro @docbg ; lệnh gộp đọc một ô nhớ PROM ngoài MOV A, #00H movc a, @a+dptr ; đọc ô nhớ endmacro org 00h mov dph, #02h ; nạp địa chỉ cao mov r0, #00h ; con trỏ địa chỉ cho RAM trong để nhớ địa chỉ của dữ liệu sai mov r3, #00h ; số hai phần tử của bảng sai nhau mov r1, #30h ; con trỏ địa chỉ hai phần tử không trùng nhau lap: mov dpl, r0 ; gửi r0 trở lại dpl @docbg ; đọc phần tử của bảng 1 mov b, a ; gửi thanh ghi a vào thanh ghi b để so sánh cjne a, #00h, tiep ; xem đã kết thúc bảng chưa? (tận cùng là số 00H), nếu chưa, sẽ tiếp sjmp kt ; kết thúc nếu phần tử của bảng bằng 0 tiep: mov a, r0 ; nạp địa chỉ phần tử của bảng 2 (250H) add a, #50h ; cộng phần tử bảng 1 với 50H mov dpl, a ; gửi lại vào a @docbg ; đọc phần tử của bảng 2 cjne a, #00h, tiep1 ; xem đã kết thúc bảng chưa? (tận cùng là số 00H), nếu chưa, sẽ tiếp sjmp kt ; kết thúc nếu phần tử của bảng bằng 0. </pre>	

tiếp1: subb a, b	; trừ hai phần tử của hai bảng
jnz sai	; nhảy tới nhãn sai nếu kết quả trừ khác 0
inc r0	; tăng r0 nếu hai phần tử bằng nhau
sjmp lap	; trở về lặp để so sánh tiếp hai phần tử của hai bảng
sai: mov r2, dpl	; ghi lại kết quả so sánh khi hai phần tử khác nhau
inc r2; r2	; tăng số phần tử lỗi
mov a, r0	; gửi địa chỉ thấp của bảng 2 có lỗi vào a
mov @r1, a	; ô nhớ địa chỉ r1 chứa địa chỉ phần tử của bảng 2 sai so với bảng 1
inc r0	; tăng con trỏ địa chỉ thấp (dpl)
inc r1	; tăng địa chỉ ô nhớ chứa địa chỉ so sánh sai
sjmp lap	; trở về lặp để tiếp tục so sánh phần tử khác
kt: sjmp \$	; kết thúc chờ (phải đứng trước lệnh tạo bảng dữ liệu)
@bg 200h, 'tôi đi chơi', 00	; tạo bảng 1
@bg 250h, 'tô đi chơ ', 00	; tạo bảng 2
end	; kết thúc dịch

CÂU HỎI VÀ THỰC HÀNH

I. Câu hỏi

1. Kể tên và công dụng của các thanh ghi đa dụng của VDK thuộc họ MCS-51.
2. Kể đặc tính và công dụng của các thanh ghi đặc biệt SFR của VDK thuộc họ MCS-51.
3. Kể đặc tính và công dụng các thanh ghi xử lý bit của VDK thuộc họ MCS-51.
4. Kể đặc tính và công dụng của các ô nhớ RAM đa dụng của VDK thuộc họ MCS.
5. Kể đặc tính và công dụng của bộ nhớ chương trình ROM của VDK thuộc họ MCS-51.
6. Hãy so sánh đặc tính của nhớ ROM của VDK với nhớ ROM của vi xử lý.
7. Hãy so sánh đặc tính của nhớ RAM của VDK với nhớ RAM của vi xử lý.
8. Hãy so sánh đặc tính của các thanh ghi của VDK với các thanh ghi của vi xử lý.
9. Hãy so sánh các phương pháp địa chỉ của lệnh dịch chuyển dữ liệu trong VDK thuộc họ MCS-51.
10. Hãy so sánh cách sử dụng lệnh đọc bảng dữ liệu trong nhớ ROM với con trỏ địa chỉ DPTR và thanh ghi đếm lệnh PC.

II. Thực hành

1. Quan sát và thay đổi nội dung thanh ghi, bộ nhớ RAM trong bảng Pinnacle (thực đơn View/ Registers, SFR, Bit field, Code memory).
2. Soạn thảo, dịch và liên kết các chương trình của các ví dụ 4-1 tới 4-8. Cho chạy thử chương trình từng bước, theo dõi sự thay đổi của các thanh ghi và ô nhớ trong quá trình thực hiện chương trình.
3. Lập trình xoá 4 bảng thanh ghi thông dụng R0-R7 của VDK thuộc họ MCS-51.
4. Lập trình xoá các thanh ghi đặc biệt của VDK thuộc họ MCS-51.
5. Lập trình xoá các thanh ghi xử lý bit của VDK thuộc họ MCS-51.
6. Lập trình xoá vùng nhớ RAM đa dụng của VDK thuộc họ MCS-51.
7. Viết chương trình chuyển 20H byte dữ liệu của vùng nhớ ROM vào vùng nhớ RAM đa dụng khi biết các địa chỉ đầu của chúng.
8. Lập trình các ví dụ về lệnh gộp cho bộ nhớ như đồng hồ, giải mã, hiển thị các dữ liệu trong các ô nhớ của đồng hồ.
9. Lập trình tạo bảng dữ liệu trong bộ nhớ PROM đặt ở PROM trong (sau chương trình và ở PROM ngoài. Viết chương trình chuyển bảng dữ liệu vừa tạo ra từ PROM ngoài vào RAM trong.
10. Lập trình để tìm kiếm một ký tự và chuỗi ký tự (dịch và cho chạy các ví dụ tương ứng).
11. Lập trình để so sánh hai bảng nhớ (dịch và cho chạy ví dụ 4-19).

Chương 5

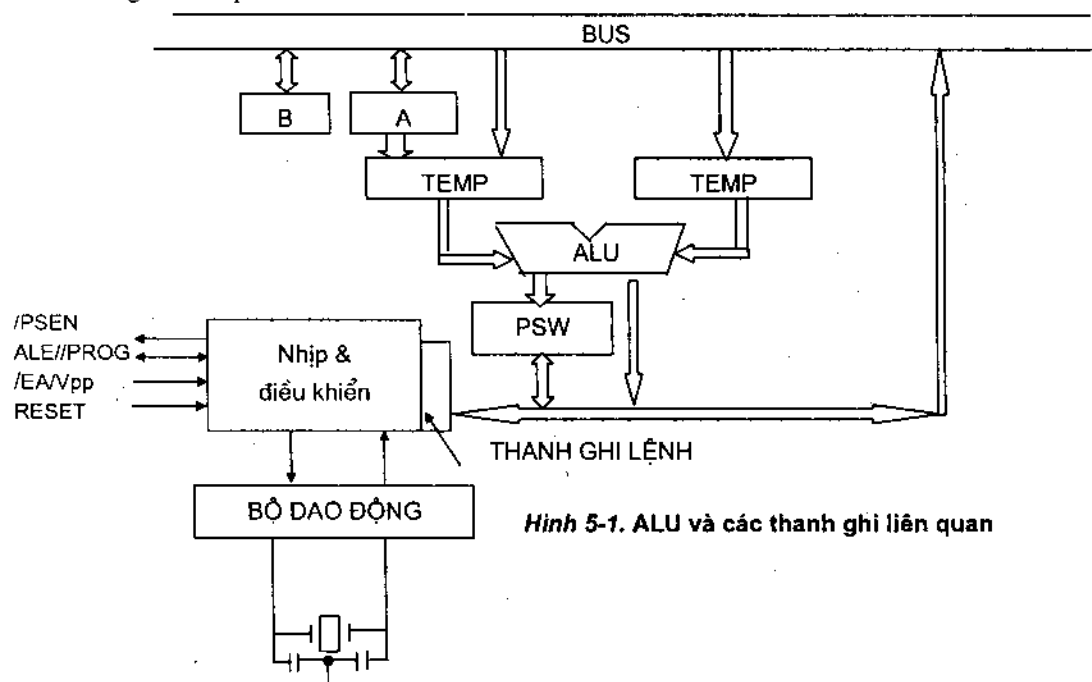
ALU VÀ CÁC LỆNH XỬ LÝ SỐ HỌC

Mục đích của chương:

- Hiểu được cấu trúc và hoạt động của bộ phận xử lý số học-logic (ALU) và các thanh ghi liên quan.
- Hiểu được các lệnh thuộc nhóm số học ứng dụng trong việc xử lý các phép toán số học như cộng trừ, nhân, chia các số không dấu và có dấu.

5.1. CẤU TRÚC ALU VÀ CÁC THANH GHI LIÊN QUAN CỦA VĐK

Từ hình vẽ sơ đồ khối chi tiết của VĐK thuộc họ MCS-51, ta thấy khối xử lý số học và logic và các thanh ghi liên quan như hình 5-1.



Hình 5-1. ALU và các thanh ghi liên quan

D7	D6	D5	D4	D3	D2	D1	D0
CY	AC	F0	RS1	RS0	OV	-	P

RS1	RS0		RS1	RS0	
0	0	dải 0	1	0	dải 2
0	1	dải 1	1	1	dải 3

Hình 5-2A. Các bit của thanh ghi trạng thái PSW

Khối ALU này của CPU của VĐK dùng để xử lý các phép toán về số học và logic với các nhóm lệnh số học (cộng, trừ, nhân, chia) và logic (và, hoặc, loại trừ) tương ứng.

Các thanh ghi liên quan với khối ALU là:

- Thanh ghi chứa A: dùng để ghi toán hạng nguồn và toán hạng đích của phép toán.
- Thanh ghi chứa B: dùng để ghi toán hạng thứ hai trong các phép toán nhân và chia.
- Thanh ghi trạng thái cờ PSW: dùng để chỉ thị cờ của kết quả của phép toán số học (cờ nhớ CY, cờ nhớ phụ AC, cờ trạng thái chung FO, cờ tràn OV, cờ chẵn lẻ P) và hai bit RS0, RS1 xác định một trong 4 dải nhớ của các thanh ghi R0-R7 thông dụng như hình 5-2A.

- Các thanh ghi đếm TEMP1, 2: dùng để ghi toán hạng cho các phép toán số học, logic.

Ngoài khối xử lý số học logic ALU (Arithmetic and Logic Unit), thanh ghi chứa A (hay Acc) và thanh ghi trạng thái cờ PSW là các thành phần cơ bản và quan trọng tới các lệnh xử lý số học (cộng, trừ, nhân, chia) và logic (AND, OR, XOR, dịch trái RL, dịch phải RR, v.v...).

Dưới đây, chúng ta sẽ nghiên cứu các lệnh số học, còn các lệnh logic dành cho chương 6.

5.2. NHÓM LỆNH SỐ HỌC CHO CÁC SỐ 8 BIT KHÔNG DẤU

5.2.1. Lệnh cộng

1. Bảng lệnh

Nhóm lệnh số học (cộng, trừ, nhân, chia) của VĐK thuộc họ MCS51 như ở bảng 5-1.

Bảng 5-1. Các lệnh số học

Lệnh	Hành động	Chế độ địa chỉ				Thời gian thực hiện (μs)
		Dir	Ind	Reg	Imm	
ADD A, <byte>	$A = A + \text{<byte>}$	X	X	X	X	1
ADDC A, <byte>	$A = A + \text{<byte>} + C$	X	X	X	X	1
SUBB A, <byte>	$A = A - \text{<byte>} - C$	X	X	X	X	1
INC A	$A = A + 1$	Chỉ thanh ghi chứa A				1
INC <byte>	$\text{<byte>} + 1$	X	X	X		1
INC DPTR	$\text{DPTR} + 1$	Chỉ Data Pointer				2
DEC A	$A = A - 1$	Chỉ thanh ghi chứa A				1
DEC <byte>	$\text{<byte>} - 1$	X	X	X		1
MUL AB	$B : A = A \times B$	Chỉ thanh ghi chứa A và B				1
DIV AB	$A = \text{Int}[A/B]$ $B = \text{Mod}[A/B]$	Chỉ thanh ghi chứa A và B				1
DA A	Deciml Adjust	Chỉ thanh ghi chứa A				1

Từ bảng trên, chúng ta có một số nhận xét sau:

- Trừ các lệnh INC DPTR và DEC <byte>, các lệnh số học khác đều có sự tham gia của thanh ghi chứa A với tư cách là nguồn dữ liệu thứ nhất và cũng là đích dữ liệu (nơi ghi kết quả của phép toán).

- Với phép tính cộng hoặc trừ, nguồn dữ liệu thứ hai là <byte> với 4 phương pháp địa chỉ (trực tiếp, thanh ghi, gián tiếp, thanh ghi gián tiếp).

- Với phép tính nhân và chia, chỉ dùng hai thanh ghi chứa A và B. Thanh ghi chứa A chứa nguồn dữ liệu thứ nhất (bị nhân, bị chia), chứa tích và thương số. Thanh ghi chứa B là nguồn dữ liệu thứ hai (số nhân, số chia), luôn viết liền với A (như là tích AB hay thương AB) và chứa số dư của phép chia.

- Dữ liệu nhập vào theo phương pháp địa chỉ tức thì, có thể ở dạng thập phân (không có hay có chữ D), dạng HEX (có chữ H) hay dạng nhị phân (có chữ B) thì chương trình sẽ tự động đổi thành dạng HEX.

- Trong VĐK, các phép tính số học (cộng, trừ, nhân, chia) luôn thực hiện dưới dạng HEX cho các dữ liệu 8 bit không dấu và đưa ra kết quả dưới dạng HEX.

- Các lệnh tăng 1 (INC) hay giảm thực hiện với thanh ghi A, thanh ghi Rn, thanh ghi con trỏ địa chỉ DPTR (không có lệnh giảm) và <byte> nhớ có địa chỉ trực tiếp và gián tiếp thanh ghi.

2. Các lệnh cộng các số 8 bit không dấu

Lệnh cộng các số 8 bit không dấu

a) Lệnh cộng các số 8 bit dạng hợp ngữ

Có hai lệnh cộng (không có nhớ C với $C = 0$ và có nhớ C với $C = 1$) là:

- ADD A, <byte> ; cộng thanh ghi chứa A với <byte> tức $A = A + \text{<byte>}$
- ADDC A, <byte> ; cộng có nhớ thanh ghi chứa A với <byte> tức $A = A + C + \text{<byte>}$

Toán hạng thứ 1 và kết quả hay tổng số luôn ở thanh ghi chứa A còn toán hạng thứ 2 ở <byte>. Ở đây, <byte> có thể là dữ liệu tức thì, thanh ghi thông dụng Ri, ở ô nhớ có địa chỉ trực tiếp hay gián tiếp thanh ghi. Như vậy, các lệnh cộng có bốn nguồn dữ liệu hay bốn phương pháp địa chỉ.

b) Lệnh gộp cộng các số 8 bit

Ví dụ 5-1 giới thiệu lệnh gộp và chương trình của phép cộng hai số 8 bit, chứa trong các ô nhớ (các tham số 1 và 2 gửi vào các ô nhớ liên tiếp có địa chỉ ban đầu là tham số 3) và kết quả ghi vào các ô nhớ liên tiếp có địa chỉ ban đầu là tham số 4. Ô nhớ đầu tiên của kết quả chứa bit cờ nhớ C vì phép cộng có thể tràn số.

Lệnh gộp trên có 4 tham số là các số hạng (thứ nhất, thứ hai), địa chỉ các ô nhớ chứa các số hạng và địa chỉ các ô nhớ chứa kết quả với ô nhớ có địa chỉ thấp chứa cờ nhớ CY, và địa chỉ cao chứa tổng số.

<i>Vi dụ 5-1: cong8 của hai số 8 bit trong các ô nhớ và ghi kết quả vào các ô nhớ</i>	<pre> JNC KT1 ; nhảy tới KT1 nếu C = 0 MOV @R1, #01h ; ghi byte cao vào @R1 INC R1 ; tăng địa chỉ nhớ MOV @R1, A ; ghi kết quả byte thấp SJMP KT ; tới kết thúc KT1: MOV @R1, #00H ; ghi byte cao INC R1 ; tăng địa chỉ nhớ MOV @R1, A ; ghi kết quả byte thấp KT: NOP ENDMAC ; kết thúc lệnh gộp ORG 00H @cong8 8H, 16H, 30H, 35H @cong8 0A8H, 0F0H, 40H, 45H @cong8 98H, 0B6H, 60H, 65H SJMP \$ END </pre>
<pre> MACRO @cong8 <số thứ 1?, số thứ 2?, ô nhớ ; ghi dữ liệu ?, ô nhớ ghi kết quả ?> CLR C ; xóa C MOV R0, #3 ; tham số #3, địa chỉ các số hạng MOV R1, #4 ; tham số #4, địa chỉ kết quả MOV @R0, #1 ; tham số #1, số thứ nhất INC R0 ; tăng địa chỉ nhớ MOV @R0, #2 ; tham số #2, số thứ hai DEC R0 ; giảm địa chỉ nhớ MOV A, @R0 ; đọc số thứ nhất INC R0 ; tăng địa chỉ nhớ ADD A, @R0 ; cộng A với số thứ nhất </pre>	

3. Chương trình cộng hai số 16 bit không dấu

Cộng các số 16 bit, gồm hai byte. Nguyên tắc cộng hai byte là:

- Dùng hai lệnh cong8 để cộng lần lượt hai cặp byte thấp và cặp byte cao.
- Cộng từ phải sang trái, cộng byte thấp trước, byte cao sau.
- Có thể phép cộng hai byte thấp có nhớ $C0 = 1$ của phép tính trước, nếu có.
- Cộng hai byte cao tuy dùng lệnh cộng có nhớ ADDC nhưng không có tác dụng mà phải cộng cờ nhớ C của phép cộng hai byte thấp bằng lệnh ngoài viết thêm (đọc cờ C ở ô nhớ 65h rồi cộng với ô nhớ chứa kết quả 66h, rồi ghi lại vào ô nhớ có địa chỉ 66h).
- Phép cộng hai byte cao có thể có nhớ C vì tràn số, nên kết quả C này phải được ghi vào ô nhớ đầu tiên (ví dụ trên là 65h như cộng hai số 8 bit).
- Kết quả cuối cùng là một số được đọc lần lượt từ kết quả của hai byte cao (có cả cờ nhớ C) và kết quả của hai byte thấp (không có cờ nhớ C).

Ví dụ 5-2 giới thiệu chương trình cộng hai số 16 bit.

Chú ý: Với phép cộng nhiều số 8 bit không dấu, ta phải:

- Đặt các số 8 bit (tham số 1, 2) vào các ô nhớ liên tiếp theo tham số địa chỉ nhập (tham số 3).
- Cộng lần lượt hai số một theo lệnh @cong8, rồi đặt kết quả vào các ô nhớ (tham số 4).
- Cộng hai kết quả với nhau theo lệnh cong8 với các tham số quy định trên.
- Tổng số là kết quả của phép cộng sau cùng của hai kết quả trung gian cuối.

; Ví dụ 5-2: Chương trình cộng hai số 16 bit dùng lệnh @cong8

ORG 00H	
\$INCLUDE MYLIB.ASM	; gọi thư viện có chứa lệnh @cong8
@cong8 18H, 96H, 30H, 35H	; cộng 8 bit hai byte thấp của số 2 byte thứ nhất
@cong8 0A0H, 85H, 40H, 45H	; cộng 8 bit hai byte cao của số 2 byte thứ nhất
@cong8 8FH, 0A6H, 50H, 55H	; cộng 8 bit hai byte thấp của số 2 byte thứ hai
@cong8 0D0H, 9AH, 60H, 65H	; cộng 8 bit hai byte cao của số 2 byte thứ hai
MOV A, 55H	; nhớ C của phép cộng byte thấp gửi vào a
ADD A, 66H	; cộng với kết quả byte cao
MOV 66H, A	; gửi lại vào ô nhớ chứa kết quả của byte cao
SJMP \$	; kết thúc chờ
END	; kết thúc chương trình dịch

5.2.2. Các lệnh trừ các số 8 bit không dấu

1. Lệnh trừ hai số 8 bit không dấu SUBB A, B

a) Lệnh trừ 8 bit dạng hợp ngữ

Chỉ có một lệnh:

SUBB A, <byte>

trừ A với <byte> và C0, kết quả gửi lại vào A và cờ nhớ C1. Ở đây C0 là cờ nhớ ở phép tính trước, còn C1 là cờ nhớ của phép trừ hai số này, tức $A - \text{<byte>} - C$

Với phép trừ, tương tự phép cộng, nhưng có những đặc điểm sau:

- Luôn dùng lệnh trừ có nhớ SUBB A, <byte> với toán hạng theo 4 phương pháp địa chỉ (bảng 5-1).
- Kết quả phép trừ có thể có nhớ (bit cờ nhớ C1 = 1) vì số bị trừ trong A nhỏ hơn <byte> được ghi ở địa chỉ đầu tiên của kết quả (cho dễ đọc).

b) Lệnh gộp trừ hai số 8 bit @tru8

Ví dụ 5-3 mô tả lệnh gộp và chương trình trừ hai số 8 bit dùng lệnh gộp.

Lệnh gộp trừ hai số 8 bit trên có đặc điểm sau:

<i>; Ví dụ 5-3: Lệnh gộp @tru8 và chương trình cho phép trừ dùng lệnh gộp</i>	
MACRO @TRU <SO A?, SO B?, địa chỉ nhớ ghi kết quả?>	MOV @R0, B ; ghi kết quả vào ô nhớ
MOV A, #1 ; tham số #1 là số hạng thứ nhất	SJMP KT ; tới kết thúc
MOV B, #2 ; tham số #2 là số hạng thứ hai	XULY: MOV A, B ; gửi kết quả âm sang a để
MOV R0, #3 ; tham số #3 là địa chỉ nhớ ghi	; chỉnh số âm nếu có nhớ (C = 1)
; kết quả	CPL A ; lấy đảo a hay bù 1 của a
CLR C ; xóa C	ADD A, #1 ; cộng 1 hay lấy bù 2 của kết quả
SUBB A, B ; trừ hai số trong A và B	INC R0 ; tăng địa chỉ nhớ
MOV B, A ; chuyển hiệu số sang B	INC R0
MOV A, #00H ; xóa A	MOV @R0, A ; kết quả dương vào ô nhớ
RLC A ; dịch trái để xét CY	DEC R0 ; giảm địa chỉ nhớ
MOV R1, A ; gửi C vào r1	DEC R0 ; giảm địa chỉ cho cờ nhớ
RRC A ; dịch phải C sang A	SJMP TIEP ; trở về tiếp
JC XULY ; nhảy tới tiếp nếu có nhớ (C = 1)	KT: NOP ; kết thúc
TIEP: MOV A, R1 ; gửi C vào A	ENDMAC ; kết thúc macro lệnh
MOV @R0, A ; ghi nhớ C vào ô nhớ	ORG 00H
INC R0 ; tăng địa chỉ nhớ	@TRU 57H, 38H, 40H ; trừ không có nhớ
	@TRU 57H, 89H, 50H ; trừ có nhớ
	SJMP \$
	END

- Giống phép cộng là có ghi kết quả của C (còn gọi là B-Borrow-mượn) và hiệu số vào các ô nhớ liên tiếp.

- Khác phép cộng là phải xử lý kết quả là số âm dạng HEX khi số bị trừ < số trừ. Do đó, phải đổi thành số âm với giá trị tuyệt đối dương (đảo bit, cộng 1) và bit cờ nhớ C = 1 (ghi vào ô nhớ đầu tiên). Dấu của hiệu số là dấu dương khi C = 0 và dấu âm khi C = 1.

2. Chương trình trừ hai số 16 bit không dấu

Nguyên tắc trừ hai số 16 bit (hai byte) cũng tương tự phép cộng hai byte chứa trong khối nhớ, nhưng có điểm khác là khi trừ hai byte cao tuy dùng lệnh trừ có nhớ SUBB nhưng không có tác dụng mà phải trừ cờ nhớ C1 bằng lệnh ngoài viết thêm (đọc cờ C ở ô nhớ 65h rồi trừ với ô nhớ chứa kết quả 66h, rồi lại ghi lại vào ô nhớ có địa chỉ 66h).

Ví dụ 5-4 giới thiệu chương trình trừ hai số 16 bit hay hai byte.

: Ví dụ 5-4: Chương trình trừ hai số 16 bit dùng lệnh @tru8	
ORG 00H	
\$INCLUDE MYLIB.ASM	: gọi thư viện có chứa lệnh gộp @tru8
@tru8 8FH, 0A6H, 50H, 55H	: trừ 8 bit hai byte thấp của số 2 byte thứ hai
@tru8 0D0H, 9AH, 60H, 65H	: trừ 8 bit hai byte cao của số 2 byte thứ hai
MOV A, 55H	: nhớ C của phép cộng byte thấp gửi vào A
ADD A, 66H	: trừ với kết quả byte cao
MOV 66H, A	: gửi lại vào ô nhớ chứa kết quả của byte cao
MOV A, #00H	: gửi 00H vào A
MOV R0, 50H	: gán địa chỉ cờ nhớ của hai byte thấp
MOV @R0, A	: xoá cờ nhớ của trừ hai byte thấp
SJMP \$	: kết thúc chờ
END	: kết thúc chương trình dịch

3. Lệnh nhân hai số 8 bit không dấu

a) Lệnh nhân dạng hợp ngữ

Lệnh nhân hai số 8 bit không dấu này là:

MUL A.B

Hai thừa số hay số bị nhân và số nhân được đặt trong các thanh ghi chứa A và B.

Kết quả là:

- Tích số đặt trong A.
- Các bit cao (nếu tràn số ở A) đặt trong thanh ghi chứa B.

b) Lệnh gộp @nhan8

Ví dụ 5-5 giới thiệu lệnh gộp và chương trình nhân hai số 8 bit. Lệnh gộp này tương tự lệnh @cong8, nhưng có những điểm khác sau:

- Lệnh @nhan8 thay cho lệnh @cong8.
- Thanh ghi chứa B chứa các bit của byte cao của tích số, khác lệnh cộng chỉ có bit cờ nhớ C1.

: Ví dụ 5-5: Lệnh @nhan8 nhân hai số 8 bit	
MACRO @NHAN8 <SO A, SO B?, ?, địa chỉ 1, địa chỉ nhớ ghi kết quả?>	
MOV A, #1	: tham số #1 là số hạng thứ nhất
MOV R0, #3	: tham số #3 là địa chỉ nhớ ghi số thứ nhất
MOV @R0, A	: ghi vào ô nhớ có địa chỉ trong r0
MOV B, #2	: tham số #2 là số hạng thứ hai
INC R0	: tăng r0
MOV @R0, B	: ghi vào ô nhớ có địa chỉ trong r0 +1
MUL AB	: nhân hai số trong A và B
MOV R1, #4	: tham số #4 là ghi địa chỉ của tích số
MOV @R0, B	: ghi tích byte cao vào địa chỉ r1 của RAM
INC R0	: tăng địa chỉ nhớ
MOV @R0, A	: ghi tích byte thấp vào địa chỉ r1+1 của RAM
ENDMAC	
ORG 00H	
@nhan 07H, 08H, 30H, 40H	
@nhan 37H, 98H, 50H, 60H	
SJMP \$	
END	

5.2.3. Phép nhân hai số không dấu

Các lệnh nhân và chia chỉ thao tác với thanh ghi A (số bị nhân hay bị chia) và thanh ghi B (số nhân hay số chia).

Với phép nhân, tích số được ghi vào thanh ghi chứa A (ví dụ 5-5). Khi tích số vượt quá 8 bit, kết quả là các bit cao ghi sang thanh ghi B.

Ngoài nhân trực tiếp bằng lệnh MUL AB, người ta có thể nhân hai số gián tiếp theo các phương pháp sau:

- Cộng nhiều lần một thừa số với chính nó mà số lần bằng thừa số kia (ví dụ 5-6).
- Nhân bằng cách dịch trái cho nhân 2 số bị nhân và cộng thêm với một số lần số bị nhân (ví dụ nhân với $9 = 8 + 1 = 2 \text{ mũ } 3 + 1$, xem chương 6).

1. Chương trình nhân hai số 8 bit bằng phương pháp cộng

Phép nhân thực chất là phép cộng số bị nhân với chính nó mà số số hạng bằng số nhân. Ví dụ 5-6 giới thiệu lệnh gộp và chương trình của phép nhân kiểu cộng dồn @nhancg. Theo phép nhân này, số bị nhân được cộng với chính nó với số lần là [số nhân - 1].

2. Chương trình nhân hai số 16 bit

Có thể nhân hai số 16 bit bằng phương pháp cộng dồn thừa số thứ nhất với chính nó với số lần là thừa số thứ hai-1. Phương pháp này đơn giản nhưng tốn thời gian của VDK. Thường dùng lệnh gộp nhân hai số 16 bit kiểu nhân.

	X2	X1		45	0FH
	×			×	
	Y2	Y1		50	F1H
		C1	X1.Y1		
+	C2	X2.Y1		0	0F.F1H
	C3	X1.Y2		0	50.0FH
C4	X2.Y2			0	45.50
(C4)	(X2.Y2+C2+C3)(X2.Y1+X1.Y2+C1) (X1.Y1)			16	21 44 1EH
	a)				b)

Hình 5-2B. Nhân hai số 16 bit không dấu : quy tắc (a) và ví dụ (b)

Quy tắc nhân hai số 16 bit như hình 5-2b với :

- Số thứ nhất là Y1 X1, số Y1 là byte cao 8 bit, số X1 là byte thấp 8 bit.
- Số thứ hai là Y2 X2, số Y2 là byte cao 8 bit, số X2 là byte thấp 8 bit.
- Phép nhân như nhân số học tức nhân các byte với nhau và cộng các tích số riêng phần với các số tràn C0, C1, C2.

- Phép nhân hai byte cao X1.Y1 có thể có số tràn C3 và là byte cuối cùng của tích số.

Tích số là tổng số của các tích riêng phần Y0X1, Y1X0, và Y1X1 có chú ý đến thứ bậc trong phép nhân (xem hình vẽ 5-2) và C3 là byte cao nhất. Như vậy, tích số của hai số có 16 bit hay hai byte có thể có tích số chứa trong 4 byte.

; Ví dụ 5-6: Lệnh @nhancg kiểu cộng dồn

MACRO @nhancg8 <1, 2, 3, 4>	
CLR C	; xóa C
MOV A, #1	; tham số #1 là thừa số thứ nhất vào A
MOV B, #2	; tham số #2 là thừa số thứ hai vào B
MOV R0, #13	; tham số #3 là địa chỉ nhớ của các thừa số
MOV R1, #14	; tham số #4 là địa chỉ nhớ của tích số
MOV R2, #00H	; đếm số lần cộng byte thấp
MOV R3, ##00H	; đếm số lần cộng byte cao
MOV @r0, a	; ghi số a vào r0
INC R0	
MOV @r0, b	; ghi số b vào r0
DEC B	; giảm B đi một lần vì A sẽ cộng với chính nó

NHAN: ADD A, #1	: cộng A với chính nó (tham số #1)
INC R2	: tăng số lần cộng hay tích số
JNC TIEP	: nếu R2 chưa tràn, tiếp tục cộng
INC R3	: tăng R3 (byte cao của tích số) vì R2 tràn, có nhớ C = 1
TIEP: DJNZ B, NHAN	: giảm số lần cộng, nếu B = 0, về NHAN cộng tiếp
MOV A, R3	: đọc R3
MOV @R1, A	: gửi byte kết quả cao vào ô nhớ có địa chỉ R1
INC R1	: tăng địa chỉ nhớ
MOV A, R2	: đọc byte thấp R2 của tích số
MOV @R1, A	: ghi vào khối nhớ
ENDMAC	: kết thúc lệnh gộp
ORG 00H	
@nhancg8 07H, 05H, 30H, 35H	: kết quả @35 = 7H × 5H = 23H = 35D
@nhancg8 17H, 05H, 40H, 45H	: kết quả @45 = 17H × 5H = 73H = 115D
@nhancg8 0FFH, 0FBH, 50H, 55H	: kết quả @55 = 1AH × 5BH
SJMP \$	
END	

5.2.4. Phép chia hai số không dấu

1. Lệnh chia hai số 8 bit dạng hộp ngữ

Với phép chia hai số ở trong các thanh ghi chứa A và B theo lệnh:

DIV AB

kết quả là : - số thương (hay Int [A/B] ở trong thanh ghi A),

- số dư (hay Mod[A/B] ở trong thanh ghi B).

Lệnh này luôn xóa cờ CY = 0 và OV = 0 nếu số chia B (ở mẫu số) khác 0.

Nếu số chia bằng 0, phép chia bị tràn số, cờ tràn OV = 1 và CY = 0.

Ví dụ 5-7: Nhân hai số 16 bit	
MACRO @nhan <so a?, so b?, địa chỉ nhớ ghi kết quả?>	ORG 00H ; cho số a = 1234h, b = 5678h
MOV A, #1 ; tham số #1 là số hạng thứ nhất	@nhan 34H, 67H, 40H; Z3
MOV B, #2 ; tham số #2 là số hạng thứ hai	@nhan 12H, 67H, 42H; Z1
MOV R0, #3 ; tham số #3 là địa chỉ nhớ ghi kết quả	@nhan 34H, 45H, 44H; Z2
	@nhan 12H, 45H, 46H; Z0
MOV P0, #00H ; xoá cổng ghi CY	MOV R0, #50H
MOV P1, #00H ; xoá cổng ghi tổng	MOV A, 44H ; gửi z0
MUL AB ; cộng hai số trong a và b	MOV @r0, a ; vào 50h
MOV @r0, b ; ghi tích bậc cao vào RAM	MOV A, 42H
MOV P0, B ; ghi tích bậc cao ra P0	MOV B, 44H
INC R0 ; tăng địa chỉ nhớ	ADD A, B
MOV P1, A ; ghi tích bậc thấp ra P1	INC R0
MOV @r0, a ; ghi tích bậc thấp vào RAM	MOV @r0, a; gửi z1+z2 ; vào 51h
ENDMAC	INC R0
	MOV A, 40H ; gửi z0
	MOV @r0, a ; vào 52h
	SJMP \$
	END

2. Lệnh gộp chia hai số 8 bit @chia8

Ví dụ 5-8 giới thiệu lệnh gộp và chương trình của phép chia hai byte 8 bit, chứa trong các ô nhớ địa chỉ liên tiếp trong thanh ghi R0, kết quả chứa trong ô nhớ có địa chỉ trong R1. Vì phép chia hai số nguyên luôn có số thương < số chia và bị chia và không bao giờ có nhớ. Các tham số 1, 2 là nội dung của hai số trong ô nhớ có địa chỉ trong R0 có tham số 3 và kết quả chứa liên tiếp số thương, số dư trong bộ nhớ có địa chỉ trong R1, là tham số 4.

3. Chương trình chia hai số dùng phương pháp trừ

Ngoài chia theo lệnh DIV AB, tương tự phép nhân kiểu cộng nhiều lần, còn có thể chia theo phương pháp trừ nhiều lần liên tiếp số bị chia với số chia tới khi có nhớ, tức số dư nhỏ hơn số chia,

phải dùng trừ và hồi phục lại số dư. Do đó, cần dùng phép trừ khi số bị chia < số chia hay phép trừ có nhớ (với C = 1) và hồi phục số dư hay số thương bằng cách cộng số dư mới với số chia (vì đã trừ).

Ta có: - Số thương là số lần thực hiện phép trừ trừ đi 1.

- Số dư là kết quả của phép cộng số dư với số chia trong phép hồi phục số dư dương.

Ví dụ 5-9 giới thiệu chương trình của phép chia theo phép trừ liên tiếp.

Ví dụ 5-8: Lệnh gộp @chia và chương trình chia hai số 8 bit

```
MACRO @chia 8 <so a?, so b?, địa chỉ nhớ ghi kết quả>
    MOV A, #1      ; tham số #1 là số hạng thứ nhất
    MOV B, #2      ; tham số #2 là số hạng thứ hai
    MOV R0, #3     ; tham số #1 là địa chỉ nhớ ghi các thừa số
    MOV R1, #4     ; tham số #4 là địa chỉ nhớ ghi kết quả
    MOV @r0, a     ; địa chỉ nhớ ghi số bị chia
    INC R0         ; tăng địa chỉ nhớ r0
    MOV @r0, b     ; địa chỉ nhớ ghi số chia
    DIV AB         ; chia hai số a và b
    MOV @r1, a     ; ghi thương vào địa chỉ r1 của nhớ RAM
    INC R1         ; ghi số dư địa chỉ r1+1 của nhớ RAM
    MOV @r1, b     ; ghi số dư vào RAM
ENDMAC
ORG 00H
@chia 27H, 08H, 30H, 40H
@chia 87H, 18H, 50H, 60H
SJMP $
END
```

Ví dụ 5-9: Chương trình của phép chia theo phương pháp trừ liên tiếp

```
MACRO @chiatru <số bị chia?, số chia?, địa chỉ nhớ các thừa số r0, địa chỉ kết quả r1>
    MOV A, #1      ; tham số #1 là số hạng thứ nhất
    MOV B, #2      ; tham số #2 là số hạng thứ hai
    MOV R0, #3     ; tham số #3 là địa chỉ 1
    MOV R1, #4     ; tham số #4 là địa chỉ 2
    MOV R2, #00H   ;
    MOV @r0, a     ; gửi số thứ nhất vào ô nhớ địa chỉ r0
    INC R0         ; tăng địa chỉ nhớ
    MOV @r0, b     ; gửi số thứ hai vào ô nhớ địa chỉ r0+1
LAP: SUBB A, B     ; trừ hai số
    INC R2         ; tăng số lần trừ (số thương)
    JNC LAP        ; trở về lap để trừ tiếp nếu chưa có nhớ c = 1 (số (d)ư > số trừ)
    ADD A, B       ; cộng với b để hồi phục số dư
    MOV @R1, R2    ; gửi số thương vào ô nhớ địa chỉ r1
    INC R1         ; tăng địa chỉ nhớ
    MOV @r1, a     ; gửi số (d)ư vào ô nhớ địa chỉ r1+1
ENDMAC
ORG 00H
@chiatru8 97H, 35H, 30H, 40H
@chiatru8 0F7H, 0A9H, 50H, 60H
SJMP $
END
```

5.2.5. Các lệnh tăng và giảm 1 đơn vị

Lệnh tăng hay giảm 1 đơn vị chỉ thao tác trên thanh ghi A và <byte> với địa chỉ thanh ghi, địa chỉ khối nhớ với trực tiếp và gián tiếp (bảng 5-1). Riêng thanh ghi con trỏ địa chỉ cho bộ nhớ ngoài DPTR chỉ có lệnh tăng.

Các lệnh tăng và giảm đi 1 này thường được dùng để thay đổi địa chỉ nhớ khi ghi nhớ liên tiếp vào các ô nhớ.

Các lệnh tăng và giảm đi 1 này được dùng kết hợp với các lệnh khác để đếm và điều khiển số lần lặp lại của nhiều vòng lặp của chương trình.

Lệnh *DJNZ Ri, nhan* là kết hợp lệnh giảm thanh ghi Ri đi 1 và lệnh nhảy JZ tới nhan.

5.3. CÁC PHÉP TOÁN SỐ HỌC VỚI SỐ BCD

5.3.1. Biểu diễn số BCD và lệnh hiệu chỉnh thập phân

1. Biểu diễn số BCD

Thông thường, trong máy vi tính hay VĐK, các phép tính số học được tiến hành dưới dạng nhị phân (B-Binary) và biểu diễn dưới dạng HEX. Dạng này có ưu điểm là dễ biến đổi thành dạng thập phân và tiết kiệm ô nhớ khi lưu trữ. Nhưng dạng HEX này, khi đưa ra hiển thị khó nhận biết, không thuận lợi cho người dùng.

Trong kỹ thuật, người ta dùng dạng biểu diễn thập phân mã hóa nhị phân BCD (Binary Coded Decimal) cho các số để tính toán trong VĐK và đưa hiển thị.

Dạng này bao gồm:

- Các con số tự nhiên từ 0 tới 9 biểu diễn dưới dạng mã nhị phân giống như trong phép biểu diễn số HEX nhưng không có các con số từ 0AH (10D) tới 0FH (15D).
- Các con số lớn hơn 10D được biểu diễn dưới dạng thập phân, tức các con số cạnh nhau, số đứng trước lớn hơn số đứng sau 10D đơn vị theo hàng chục, hàng trăm, hàng ngàn, hàng vạn v.v. như thực tế đời sống thường dùng.
- Có hai dạng biểu diễn số BCD :
 - + *Dạng BCD không nén*: mỗi byte 8 bit biểu diễn một con số BCD. Như vậy, dạng này để trống 4 bit không sử dụng nên ít được dùng.
 - + *Dạng BCD nén*: mỗi byte 8 bit biểu diễn hai con số BCD. Như vậy, dạng này tận dụng cả 8 bit để biểu diễn hai chữ số của một con số hàng chục và hàng đơn vị.
- Các con số lớn hơn hàng chục được biểu diễn dưới dạng nhiều byte kế tiếp, hàng bên trái lớn hơn hàng bên phải 10 lần. Như vậy, con số 1234D dùng hai byte để biểu diễn 4 chữ số thập phân là 1D, 2D ở byte cao, còn 3D và 4D ở byte thấp. Khi lưu trữ trong bộ nhớ, lại lưu trữ hàng đơn vị trước, hàng chục, trăm, nghìn v.v... sau.

Dùng số BCD có ưu điểm là các con số dễ đọc và gần với thực tế đời sống thường ngày.

Nhưng số BCD có những nhược điểm sau:

- *Không tiết kiệm được dung lượng bộ nhớ*: khi lưu trữ trong bộ nhớ của VĐK hay trong máy vi tính. Ví dụ, hai byte nhớ chỉ lưu trữ được tối đa các con số từ 0000D tới 9999D (10000D con số) trong khi đó dưới dạng nhị phân (B) hay HEX được 65535 con số.
- *Khó khăn trong tính toán số học*: trong tính toán, chúng ta phải biến đổi kết quả ở dạng HEX sang BCD nén. Vì trong vi xử lý của máy tính hay VĐK đều tính toán và đưa ra kết quả dưới dạng nhị phân B hay HEX nên ta phải biến đổi kết quả dạng HEX sang BCD nén bằng:
 - + Chương trình biến đổi HEX sang BCD (xét ở chương 6).
 - + Dùng lệnh DA A để đổi một chữ số dạng HEX từ AH tới FH thành dạng thập phân là 10D tới 15D khi có cờ nhớ phụ (Auxiliary Carry) AC = 1 (xác lập khi bit thứ ba D3 > 9).

2. Lệnh hiệu chỉnh thập phân DA A

Một byte (8 bit) biểu diễn được hai chữ số dạng BCD có nén, tức chữ số hàng chục (hai chữ số) hoặc đơn vị (một chữ số). Khi cộng hai số BCD nén, hai chữ số dạng HEX của hàng đơn vị và hàng chục được cộng lại với nhau. Mỗi chữ số hàng đơn vị hay hàng chục của tổng số có thể > 9, tức từ 0AH tới 0FH. Khi đó phải hiệu chỉnh thành dạng thập phân để trở thành 10D tới 15D. Cách dùng lệnh hiệu chỉnh thập phân DA (Decimal Adjust) cho dữ liệu chứa trong thanh ghi A để biến đổi nó thành các chữ số dạng thập phân (từ 0D tới 9D của dạng thập phân).

Lệnh DA A này tác dụng lên số dạng HEX chứa trong thanh ghi A và có đặc điểm sau:

- Không có tác động gì khi thanh ghi A chứa số < 9H;
- Chỉ xảy ra khi chữ số BCD ở 4 bit thấp của byte hay bit thứ ba (D3) của byte vượt quá 9H (từ 0AH hay 10D tới 0FH dưới dạng HEX), bit cờ nhớ phụ (AC) của thanh ghi trạng thái PSW xác lập lên 1.

- Bit thứ ba D3 được cộng thêm 6D để số đó vượt quá 16D và chỉ giữ lại phần dư sau khi trừ đi 16D.

- Bit thứ tư D4 đứng trước trong byte được cộng thêm 1 (nhớ 1 của AC) hay đơn vị thập phân đứng trước tăng thêm 1 đơn vị.

Lệnh hiệu chỉnh thập phân DA A này dùng để hiệu chỉnh kết quả sau khi cộng hai số dạng nhị thập phân BCD có nén.

Với các số BCD có nhiều chữ số chứa trong nhiều byte, phải dùng nhiều lệnh DA A và chúng thực hiện việc cộng thêm 60D, 600D, 6000D vào các bit D3 của các byte cao trong phép biểu diễn BCD nhiều byte.

5.3.2. Cộng hai số BCD có nén

1. Cộng hai số BCD có nén 8 bit

Cộng hai số BCD có nén tiến hành theo hai bước:

- Cộng số HEX cho hai byte như phép cộng số HEX ở trên.

- Hiệu chỉnh số HEX sang số BCD bằng lệnh DA A.

Ví dụ, cộng hai số 37H (thực chất là 37D) và 35H (thực chất là 35D), tổng hai số là 6CH (dạng HEX), dùng thêm lệnh DA A sau lệnh cộng để biến đổi 6CH thành 72H hay 72D trong phép biểu diễn BCD có nén.

Do đó, lệnh gộp @cong8 (xem ví dụ 5-10) chính là lệnh gộp @cong8 có thêm lệnh hiệu chỉnh DA A.

Ví dụ 5-10 giới thiệu lệnh gộp @cong8 và chương trình cộng hai số BCD có nén.

Chú ý: - Các dữ liệu vào tuy là số thập phân nhưng phải viết kèm đơn vị là số HEX (chữ H sau hai chữ số cho một byte), nếu không, chương trình sẽ tự động biến thành dạng số HEX và kết quả tổng sẽ sai.

- Tổng số tuy là dạng HEX (có chữ H sau hai chữ số) nhưng là kết quả thập phân tức con số hàng chục và hàng đơn vị cho một byte.

- Lệnh DA A còn hiệu chỉnh cả con số hàng chục và kết quả là bit C xác lập, tức nhớ 1 sang hàng trăm.

Ví dụ 5-10: Lệnh @ cong8 hai số BCD có nén

```
MACRO @cong8 <1, 2, 3, 4>
    MOV A, #1          ; nạp tham số #1, số thứ nhất vào A
    MOV B, #2          ; nạp tham số #2, số thứ 2 vào B
    MOV R0, #13        ; nạp tham số #13, địa chỉ các số hạng
    MOV R1, #14        ; nạp tham số #14, địa chỉ kết quả
    MOV @R0, A
    INC R0
    MOV @R0, B
    ADD A, B           ; cộng hai số A và B
    DA A              ; hiệu chỉnh thập phân
    INC R1             ; tăng địa chỉ nhớ kết quả
    MOV @R1, A         ; ghi tổng số byte thấp
    DEC R1             ; giảm địa chỉ nhớ cho hàng chục
    MOV A, #00h        ; xoá A
    RLC A              ; chuyển bit C vào A
    MOV @R1, A         ; nhớ số hàng trăm
ENDMAC
ORG 00H
@cong8 19H, 98H, 30H, 32H
@cong8 93H, 89H, 40H, 42H
SJMP $               ; kết thúc chờ
END                  ; kết thúc dịch
```

Với phép trừ hai số có nén, ta cũng tiến hành tương tự như cộng hai byte BCD có nén.

Với phép nhân và chia, ta phải:

- Biến đổi dữ liệu vào để nhân và chia từ dạng BCD sang dạng HEX trước khi nhân và chia (xem chương 6).

- Tiến hành các phép tính nhân hay chia hai số dưới dạng HEX.

- Biến đổi kết quả từ dạng HEX sang dạng BCD (xem chương 6).

Nói chung, dạng BCD chỉ thuận tiện cho nhập dữ liệu vào, đưa dữ liệu ra và cộng hay trừ hai hay nhiều số dạng BCD, còn nhân và chia, dạng này không thích hợp vì phải đổi sang dạng HEX.

2. Cộng hai số BCD có nén 16 bit

Cộng hai byte 16 bit, chứa được hai số thập phân hàng nghìn, theo quy tắc sau:

- Giống phép cộng hai byte của hai số dạng HEX là lần lượt cộng hai byte thấp rồi hai byte cao.

- Xử lý bit nhớ C ở phép cộng hai byte thấp bằng cách cộng với kết quả của phép cộng hai byte cao, xóa bit nhớ trên khỏi ô nhớ tương ứng.

- Kết quả của phép cộng này lần lượt đọc ra gồm bit C của cộng hai byte cao, kết quả cộng hai byte cao, kết quả cộng hai byte thấp.

Ví dụ 5-11 giới thiệu chương trình cộng hai số BCD hàng ngàn với lệnh @cong8 đã có trong thư viện MYLIB.ASM.

Nếu cần cộng hai số lớn hơn hai byte (trên hàng ngàn, ta cũng cộng tiếp hai byte một rồi cộng bit nhớ C sang kết quả của byte cao kế nó).

; Ví dụ 5-11: Chương trình cộng hai số BCD 16 bit

```
ORG 00H
INCLUDE MYLIB.ASM
@cong8 19H, 98H, 30H, 32H
@cong8 098H, 089H, 40H, 42H
MOV A, 32H
ADD A, 43H
MOV 43H, A
MOV A, #00H
MOV 32H, A
SJMP $      ; kết thúc chờ
END         ; kết thúc dịch
```

5.3.3. Trừ hai số BCD dương có nén 8 bit

Tương tự phép trừ trong biểu diễn số nhị phân hay HEX, ta phải cộng số bị trừ với dạng bù 10 của số trừ, trong dạng BCD nén.

1. Dạng bù 10 của số trừ BCD có nén

Cách biến đổi số bù 10 như sau:

- Lấy số bù 9 của số trừ: bằng cách lấy các số 9 trừ đi các chữ số của số trừ thập phân. Ví dụ, số 15D có số bù 9 là $99-15 = 84$.

- Lấy số bù 10 của số trừ: lấy số bù 9 + 1. Ví dụ, số bù 10 của 15 D là $84+1 = 85$.

2. Trừ hai số BCD 8 bit có nén cho kết quả dương

Ở đây chỉ xét trừ hai số BCD có nén cho kết quả dương (số bị trừ > số trừ), còn ngược lại, sẽ xét ở mục trừ hai số có dấu.

Cách tính hiệu số của hai số BCD là cộng số bị trừ với số bù 10 của số trừ, sau đó hiệu chỉnh thập phân bằng lệnh DA A.

Ví dụ 5-12 giới thiệu lệnh gộp @trud8 và chương trình tính trừ hai số BCD 8 bit.

Các phép tính nhân và chia hai số BCD 8 bit có nén hay 16 bit có nén ta phải:

- Biến đổi thành số dạng HEX (xem chương 6).

- Tiến hành cộng, trừ, nhân và chia với số HEX.

- Biến đổi ngược lại từ HEX sang BCD có nén.

; Ví dụ 5-12: Lệnh gộp @trud8	
MACRO @trud8 <1, 2, 3, 4>	DEC R0 ; giảm địa chỉ nhớ
CLR C	MOV A, @r0 ; ghi vào ô nhớ
MOV A, #1 ; nạp số 11 vào a	ADD A, B ; cộng số thứ nhất với bù 10 của
MOV B, #2 ; nạp số 12 vào b	DA A ; số thập phân
MOV R0, #3 ; nạp địa chỉ nhớ số bị trừ và số trừ	MOV @r1, a ; ghi kết quả vào ô nhớ
MOV R1, #4 ; nạp địa chỉ nhớ kết quả	ENDMAC ; kết thúc lệnh gộp
MOV @R0, A ; ghi số thứ nhất vào địa chỉ r0	ORG 00H ; bắt đầu chương trình chính
INC R0	@trud8 85h, 13h, 30h, 32h; trừ 85h-13h,
MOV @r0, b ; ghi số thứ hai vào ô nhớ tiếp theo	@trud8 76h, 49h, 40h, 42h; trừ 76h-49h
MOV A, #99H ; lấy bù 9 của số thứ hai trong b	; ghi ở 40h, 41h, kết quả ở 42h
SUBB A, B ; trừ 99h-b	SJMP \$; kết thúc chờ
ADD A, #1 ; lấy bù 10	END ; kết thúc dịch
MOV B, A ; b chứa bù 10 của số thứ nhất	

5.4. LẬP TRÌNH CÁC PHÉP TOÁN SỐ HỌC VỚI SỐ CÓ DẤU

5.4.1. Cách biểu diễn số có dấu

Với các byte 8 bit hay lời 16 bit, người ta có thể biểu diễn số có dấu theo quy ước sau:
 - dành một bit ở tận cùng bên trái của một byte số nhị phân để biểu diễn dấu của số với quy ước: 1 là dấu âm, 0 là dấu dương (mặc định).

Ví dụ (+)15D = 00010101B

và (-)5D = 10001010B hay là 85H.

phần giá trị của số chỉ có 7 bit (cho byte) hay 15 bit (cho lời).

Nói khác đi, giá trị của:

+ Một byte từ -127D (hay -2^7) tới +127 (hay $+2^7$).

+ Một lời từ (- 32767) hay (-2^{15}) tới (+ 32767) hay ($+2^{15}$).

- Dành các bit còn lại ở bên phải cho giá trị tuyệt đối của số có dấu.

Ví dụ (-) 5D = 10001010B hay là 85H thì số 8 là dấu âm còn số 5H là giá trị tuyệt đối.

5.4.2. Chương trình tính số bù 2 của một số

Khi cộng với số có dấu âm, người ta cộng với số bù 2 của nó (xem ví dụ 5-10). Do đó, chúng ta phải viết chương trình biến đổi số đó thành số bù 2 của nó.

1. Quy tắc lấy bù 2 của một số nhị phân

Một số nhị phân biến đổi một số dấu âm nhị phân thành dạng bù 2 theo các bước sau:

+ Lấy bù 1 bằng cách đảo các bit (1 thành 0, 0 thành 1) của số đó.

+ Cộng với 1 (+1).

Ví dụ -5D hay 10001001B có số bù 2 là 01110110B+1B = 01110111B = 77H.

Ví dụ 5-13 minh họa phép biến đổi số bù 2 của số âm một byte cho +15D với (-5D).

; Ví dụ 5-13: Cộng số âm cho (+)15D + (-)5D = (+)10D			
Dạng nhị phân của 5D là:	0000101B		
Đảo hay bù 1 của 5D là :	1111010B	+15D = 00001111B	
Cộng 1	+ 1B	+ -5D = 1111011B	
		+ 15D+(-5D)=00001010B=0AH=(+)10D	
Bù của 5D là	1111011B		

2. Chương trình lấy bù 2 của số có dấu 8 bit

Các ví dụ 5-14 và 5-15 biến đổi số bù 2 của số 8 bit. Ví dụ 5-15 lưu ý tới hiện tượng tràn số đối với số âm (giá trị tuyệt đối có 7 bit) bởi cờ tràn OV.

Các thanh ghi nội được dùng cho các chương trình số học như ở bảng 5-2.
Cổ bốn bit địa chỉ hóa được dùng là ba bit đầu XS, YS và ZS và bit tràn số ZOV.

Bảng 5-2. Các thanh ghi nội cho các chương trình số học có dấu

Thanh ghi	Các tên					
1	X	XL	XA	XAL		
2			XT	XTL		
3				XTH		
4	Y	YL	YA	YAL		
5		XH		XAH		
6		YH		YAH		
7	Z	ZL	ZA	ZAL	Z0	ZA0
8		ZH		ZAH	Z1	ZA1
9				-	Z2	ZA2
10					Z3	ZA3

Các chương trình trên có:

- Các thanh ghi vào XT, tích lũy byte có dấu X ở dạng bù 2 và giá trị tuyệt đối của X.
- Các thanh ghi ra XA, tích lũy byte có dấu X ở dạng bù 2 và giá trị tuyệt đối của X.
- Bit dấu XS tích dấu của X. $S = 1$ nếu số âm ($X < 0$).
- Với số 16 bit, dùng các thanh ghi XTH và XTL để chứa byte cao và thấp của số 16 bit có dấu ở dạng bù 2.
- Tương tự, các thanh ghi XAH, XAL chứa byte cao và thấp ở dạng tuyệt đối có dấu.
- Các chương trình biến đổi dạng bù 2 TCMS8 và TCM16 thành độ lớn/dấu cho 8 và 16 bit.
- Các chương trình biến đổi độ lớn/dấu MS2TC8, TC2MS16 thành dạng bù 2 cho 8 và 16 bit.

3. Chương trình lấy bù 2 của số có dấu 16 bit

Các ví dụ 5-16 và 5-17 giới thiệu chương trình biến đổi xuôi và ngược một số có dấu 16 bit thành dạng bù 2 của nó. So với các chương trình ở ví dụ cho số có dấu 8 bit, các chương trình này có những đặc điểm sau:

- Trị số tuyệt đối gồm hai byte (8 bit và 7 bit), thích hợp cho phép nhân và chia.
- Tiến hành lấy bù 2 cho cả 15 bit biểu diễn độ lớn của số, thích hợp cho phép cộng và trừ.
- Tiến hành biến đổi dấu trên bit 16 thay cho bit 8.
- Tiến hành xét tràn số khi tới bit 16 thay cho bit 8.

5.4.3. Chương trình cho số có dấu 8 bit

1. Cộng trừ số có dấu 8 bit

Các phép tính cộng đều thực hiện theo quy tắc sau:

Nếu hai số dương, cách cộng như cộng hai số không có dấu.

Nếu một số âm, cách cộng bằng cộng số dương với số bù 2 của số âm như ví dụ 5-13.

Nếu cộng hai số âm, kết quả dấu âm, trị số tuyệt đối là tổng của hai trị số tuyệt đối.

Với chương trình hợp dịch Pinnacle, số âm đưa vào được tự động biến đổi thành số bù hai. Do đó, phép cộng hay trừ hai số có dấu theo nguyên tắc cộng hai số không dấu, sau đó phải đổi kết quả dạng bù 2 thành số có giá trị tuyệt đối dương. Ví dụ 5-16 giới thiệu lệnh gộp và chương trình cộng hai số 8 bit có dấu trong mọi trường hợp.

Trong các phép cộng hay trừ số có dấu, chúng ta cần lưu ý rằng:

- Nếu hai số có dấu dưới dạng bù 2 tổng của chúng $Z = X + Y$ cũng tìm thấy khi dùng lệnh ADD.
- Bit cờ tràn OV trong thanh ghi trạng thái PSW sẽ xác lập lên 1 khi tổng của hai số âm thành một số dương.
- Nếu một trong hai số X và Y hay cả hai ở dưới dạng tuyệt đối (có dấu), trước tiên phải biến đổi thành dạng bù 2 rồi mới tiến hành phép cộng.
- Hiệu của hai số có dấu $Z = X - Y$ được tiến hành bởi lệnh trừ SUBB nếu cả hai dưới dạng bù 2.

- Bit cờ tràn OV cũng xác lập lên 1 nếu kết quả tràn khỏi dải. Ví dụ (10)-(-120) cho 130 lớn hơn 127. Khi đó, phải dùng phép trừ hai byte 16 bit.

; Ví dụ 5-14: Chương trình biến đổi số bù 2/dấu	; Ví dụ 5-15: Chương trình biến đổi số bù 2/bit dấu
TC2MS8: MOV A, XT ; gửi XT vào A JB ACC.7, Xneg ; nếu số âm, nhảy tới Xneg MOV XA, XT ; giá trị tuyệt đối vào A CLR XS ; xóa dấu về 0 cho số dương RET ; kết thúc Xneg: CPL A ; đảo bit cho A INC A ; +1 MOV XA, A ; gửi vào XA SETB XS ; xác lập dấu âm (1) RET ; kết thúc	MS2TC8: MOV A, XT ; gửi XT vào A JNB ACC.7, rangeOK ; xét khi số dương JNE #80H, rangeERROR ; 80H tới ERROR JB XS, rangeOK ; XS = 1, tới rangeOK rangeERROR: SETB XOVI ; OV = 1, chỉ tràn RET rangeOK: CLR XOVI ; xóa cờ tràn OV = 0 JB XS, Xneg1 ; XS = 1, tới Xneg1 MOV XT, XA ; dương, XT = XA RET Xneg1: MOV A, XA ; nếu âm, xét XA CPL A ; đảo A INC A ; +1 MOV XT, A ; gửi A vào XT RET ; kết thúc
; Ví dụ 5-16: Chương trình biến đổi bù 2	; Ví dụ 5-17: Chương trình biến đổi số bù 2
TC2MS16: MOV A, XTH ; gửi XT vào A JB ACC.7, Xneg2 ; nếu âm, tới Xneg2 MOV XAH, XTH ; nạp giá trị t/dối XTH MOV XAL, XTL ; nạp giá trị t/dối XTL CLR XS ; xóa dấu về 0 cho số dương RET ; kết thúc Xneg2: SETB XS ; xác lập dấu âm MOV A, XTL ; nạp XTL vào A CPL A ; đảo bit cho A INC A ; +1 MOV XAL, A ; nạp A vào XAL JNZ XL_OK ; khác 0, nhảy tới XL_OK MOV A, XTH ; nạp XTH vào A CPL A ; đảo bit cho A INC A ; +1 MOV XAH, A ; nạp A vào XAH RET ; kết thúc XL_OK: MOV A, XTH ; nạp XTH vào A CPL A ; đảo bit cho A MOV XAH, A ; nạp A vào XAH RET ; kết thúc SETB XS ; xác lập dấu âm (1) RET ; kết thúc	MS2TC16: MOV A, XAH ; gửi XAH vào A JNB ACC.7, rangeOK1 ; khi âm tới rangeOK1 CLR ACC.7 ; xóa dấu ORL A, XAL ; Acc = 0 chỉ khi X = 8000 JNZ rangeERROR1 ; khác đi là lỗi JB XS, range_OK1 ; lỗi khi X là số dương rangeERROR1: SETB XOVI ; cờ tràn OV = 1 RET ; kết thúc rangeOK1: CLR XOVI ; xóa cờ tràn JB XS, Xneg2 ; nếu âm, kiểm tra dấu MOV XTH, XAH ; nạp XAH vào XTH MOV XTL, XAL ; nạp XAL vào XTL RET ; kết thúc Xneg3: MOV A, XAL ; nếu X âm, gửi vào A CPL A ; đảo A INC A ; +1 MOV XTL, A ; gửi A vào XTL JNZ XL_OK1 ; nếu khác 0, tới XL_OK1 MOV A, XTH ; gửi XTH vào A CPL A ; đảo A INC A ; +1 MOV XTH, A ; nạp A vào XTH RET ; kết thúc XL_OK: MOV A, XAH ; nạp XAH vào A CPL A ; đảo A MOV XTH, A ; nạp A vào XTH RET ; kết thúc

2. Nhân và chia hai số có dấu 8 bit

Nhân và chia hai số có dấu theo quy tắc sau:

- Giá trị tuyệt đối của chúng sẽ nhân hay chia với nhau;
- Dấu của kết quả theo quy tắc: + Hai số cùng dấu, có dấu dương;
+ Hai số trái dấu, có dấu âm.

Các ví dụ 5-19, 5-20 và 5-21 giới thiệu lệnh gộp và chương trình nhân và chia hai số có dấu 8 bit.

Khi nhân hay chia số có dấu, có thể có các trường hợp sau:

- *Cả hai số ở dưới dạng tuyệt đối:* dấu của kết quả được xác định từ dấu của các toán hạng, theo quy tắc cùng dấu cho dấu kết quả dương, trái dấu cho dấu của kết quả âm. Giá trị tuyệt đối là tích hay thương của các toán hạng theo các lệnh nhân MUL hay chia DIV.

- *Cả hai số ở dưới dạng bù 2:* phải đổi sang dạng tuyệt đối bằng các chương trình biến đổi rồi tiến hành nhân hay chia cho hai số dưới dạng tuyệt đối như trên.

<i>; Ví dụ 5-18: Lệnh gộp và chương trình cộng hai số có dấu 8 bit</i>	
MACRO @cgdaus <số thứ 1?, số thứ 2?, ô ; nhớ ghi dữ liệu ? ; ô nhớ ghi kết quả ?> CLR C ; xóa C MOV R0, #13 ; ghi địa chỉ nhớ các số hạng ; của dữ liệu MOV R1, #14 ; ghi địa chỉ nhớ kết quả MOV @R0, #1 ; tham số ghi số thứ nhất INC R0 ; tăng địa chỉ nhớ MOV @R0, #2 ; tham số ghi số thứ hai DEC R0 MOV A, @R0 INC R0 ADD A, @R0 ; cộng A với số thứ nhất JB ACC.7, xneg INC R1 ; ghi kết quả dương	MOV @r1, a DEC r1 MOV @r1, #00h SJMP kt XNEG: cpl a ; xử lý kết quả âm INC a INC r1 ; ghi kết quả âm MOV @r1, a DEC r1 MOV a, #01h MOV @r1, a KT: NOP ENDMAC ; kết thúc lệnh gộp ORG 00H @cgdaus 8H, -16H, 30H, 35H @cgdaus 14H, -20H, 50H, 55H @cgdaus 15H, 20H, 60H, 65H @cgdaus -15H, -20H, 70H, 75H SJMP \$ END

Về phần giá trị tuyệt đối, hai toán hạng tuân theo các lệnh nhân (MUL) và chia (DIV) hai số 8

Gọi XA và YA là các giá trị tuyệt đối của các toán hạng, còn XS và YS là các bit dấu. Khi nhân hai số 8 bit, tích số có thể là 16 bit với bit dấu ở byte cao.

(Tiến hành xét tràn số khi tới bit 16 thay cho bit 8).

Về phần giá trị tuyệt đối, hai toán hạng tuân theo các lệnh nhân (MUL) và chia (DIV) hai số 8 bit không dấu (chỉ kể tối đa 7 bit).

5.4.4. Chương trình cho số có dấu 16 bit

Các phép tính ở đây cho hai cặp thanh ghi 8 bit là XH, XL, YH, YL cho kết quả trong ZL chứa hai byte 16 bit. Cũng tiến hành xét dấu của các số nguyên X, Y và Z. Bit ZOV cũng xác lập lên 1 khi có tràn tức kết quả Z vượt quá dài.

Cũng có hai dạng biểu diễn là tuyệt đối (có dấu) và bù 2 cho số âm và cũng có thể tiến hành biến đổi và tính toán theo một trong hai dạng biểu diễn số như số 8 bit. Ở dạng tuyệt đối, các phép tính nhân hay chia có thể tiến hành nhân hay chia hai giá trị tuyệt đối và xét dấu sau.

1. Cộng và trừ số có dấu 16 bit

Tương tự phép cộng và trừ cho số 8 bit nhưng ở đây:

- Tiến hành hai phép cộng hay trừ cho hai byte thấp và cao của hai số 16 bit.
- Tiến hành xét dấu cho bit thứ 16 ở tận cùng bên trái của byte cao.

; Ví dụ 5-19: Chương trình con nhân hai số 8 bit có dấu	; Ví dụ 5-20: Chương trình con chia hai số 8 bit có dấu
SMUL8: MOV A, XA; nạp giá trị tuyệt đối XA vào A MOV B, XB ; nạp giá trị tuyệt đối XB vào B MUL AB ; nhân hai số MOV ZH, B ; nạp B vào ZH MOV ZL, A ; nạp A vào ZL JB XS, SM8_2 ; xét dấu XS, nếu = 1 tới SM8_2 JB YS, SM8_1 ; xét dấu XS, nếu = 1 tới SM8_1 CLR ZS ; xóa ZS RET ; kết thúc SM8_1: SETB ZS ; xác lập ZS = 1 RET ; kết thúc SM8_2: JB YS, SM8_3 ; xét dấu YS, nếu YS = 1, tới SM8_3 SETB ZS ; xác lập ZS = 1 RET ; kết thúc SM8_3: CLR ZS ; xóa ZS = 0 RET ; kết thúc	SDIVL8: MOV A, XA ; nạp giá trị tuyệt đối XA vào A MOV B, XB ; nạp giá trị tuyệt đối XB vào B DIV AB ; nhân hai số MOV C, OV ; nạp bit OV vào C MOV ZOV, C ; gửi OV vào ZOV MOV ZAQ, A ; nạp A vào thương số ZAQ MOV ZAR, B ; nạp B vào số dư ZAR JB XS, SD8_2 ; xét dấu XS, nếu = 1 tới SM8_2 JB YS, SD8_1 ; xét dấu XS, nếu = 1 tới SM8_1 CLR ZS ; xóa ZS = 0 RET ; kết thúc SD8_1: SETB ZS ; xác lập ZS = 1 RET ; kết thúc SD8_2: JB YS, SD8_3 ; xét dấu YS, nếu YS = 1, tới SD8_3 SETB ZS ; xác lập ZS = 1 RET ; kết thúc SD8_3: CLR ZS ; xóa ZS = 0 RET ; kết thúc

; Ví dụ 5-21: Lệnh gộp và chương trình nhân hai số 8 bit có dấu	
macro @nhda8 <a?, b?, r0, r1?> MOV R2, #00H ; giá trị tuyệt đối XB vào B MOV R0, #13 ; địa chỉ ghi dữ liệu vào R0 MOV R1, #14 MOV A, #1 ; giá trị tuyệt đối XA vào A MOV B, #12 CALL DAU0 ; xử lý dấu của A MOV A, B CALL DAU0 ; xử lý dấu của B MOV a, @R0 DEC R0 MOV b, @R0 MUL AB CALL dau1 ; xử lý dấu của tích tuyệt đối AB MOV A, B CALL DAU1 CALL DAUKQ SJMP KT DAU0: jb acc.7, xneg ; ghi kết quả dương INC R0 MOV @R0, A RET XNEG: INC R2 ; tăng lần xử lý giá trị tuyệt đối âm CPL A ; đảo bit của a (Bù 1) INC A ; +1 (Bù 2) INC R0 ; ghi Số Âm MOV @R0, A	RET DAU1: b acc.7, xneg1 ; xử lý giá trị tuyệt đối của tích INC R1 MOV @R1, A RET XNEG1: CPL A ; xử lý kết quả âm INC A INC R1 ; ghi kết quả âm MOV @R1, A RET DAUKQ: MOV A, R2 JZ DUONG CJNE A, #02, AM DUONG: MOV A, #00h MOV @R1, A RET ; địa chỉ ghi kết quả AM: MOV A, #01h MOV @R1, A RET ENDMAC ORG 00H @nhda8 8H, 5H, 30H, 35H; kết quả dấu dương (0) @nhda8 -8H, 75H, 40H, 45H; kết quả dấu âm (1) @nhda8 0A8H, -5H, 50H, 55H; 0a8H là số Âm, nên tích có dấu dương (0) @nhda8 -8H, -5H, 60H, 65H; kết quả dương (0) JMP \$ END

Các ví dụ 5-22 và 5-23 cho các chương trình cộng và trừ hai số có dấu 16 bit.

Nhân và chia hai số có dấu 16 bit được thực hiện một cách đơn giản như nhân và chia hai số không dấu rồi cộng thêm dấu như xét cho nhân và chia hai số có dấu 8 bit cho giá trị tuyệt đối. Nếu số có dạng bù 2, trước tiên phải đổi chúng thành dạng tuyệt đối. Kết quả của chúng có thể biến đổi lại thành dạng bù 2.

; Ví dụ 5-22: Chương trình con cộng hai số 16 bit có dấu

```
SADD8: MOV A, XL ; nạp byte thấp XL vào A
        ADD A, YL ; cộng XL với YL
        MOV ZL, A ; nạp A vào ZL
        MOV A, XH ; nạp byte cao XH vào A
        ADDC A, YH ; cộng XH với YH
        MOV ZH, A ; nạp A vào ZH
        MOV C, OV ; nạp OV vào C
        MOV ZOV, C ; nạp OV vào ZOC
        RET
```

; Ví dụ 5-23: Chương trình con trừ hai số 16 bit có dấu

```
SSUB 16: MOV A, XL ; nạp XL vào A
          CLR C ; xóa C
          SUBB A, YL ; trừ hai số
          MOV ZL, A ; nạp A vào ZL
          MOV A, XH ; nạp byte cao XH vào A
          SUBB A, YH ; trừ XH với YH
          MOV ZH, A ; nạp A vào ZH
          MOV C, OV ; nạp OV vào C
          MOV ZOV, C ; nạp OV vào ZOC
          RET ; kết thúc
```

2. Nhân và chia hai số có dấu 16 bit

Nhân và chia hai số 16 bit có dấu cũng tương tự nhân và chia hai số có dấu 8 bit. Chỉ có điều khác là bit dấu (0 cho số dương, 1 cho số âm) đặt ở bit tận cùng D7 của byte cao của số. Do đó, khi nhập số vào, ta chỉ xét bit E7 này, còn bit D7 của byte thấp không cần xét, vì nó là giá trị tuyệt đối.

5.5. LẬP TRÌNH CHO CÁC PHÉP TOÁN SỐ HỌC VỚI SỐ CÓ DẤU PHẪY ĐỘNG

1. Biểu diễn số dấu phẩy động

Các số nguyên hay BCD hay số có dấu phẩy cố định có phần nguyên cố định, có nhược điểm là:

- Không biểu diễn được số thập phân (phần lẻ sau dấu phẩy).
 - Chiếm nhiều bộ nhớ.
 - Khó biểu diễn được số rất lớn hay rất bé như $6,02 \cdot 10^{23}$ và $8,76 \cdot 10^{-14}$.
 - Các phép toán cộng, trừ, nhân, chia khó thực hiện với các số có độ lớn chênh lệch nhau nhiều.
- Do đó, người ta đặt ra số dấu phẩy động, có phần thập phân thay đổi được (động) với quy ước:
- Mỗi số là một hàm số mũ cơ số A (có thể là 2- nhị phân hay 10 - thập phân).
 - Mỗi số là tích của hai phần:
- + *Phần mũ E*: mũ +E (cho số nguyên) hay mũ -E (cho số thập phân có dấu phẩy đứng trước).
 - + *Phần định trị M*: phần chỉ giá trị tuyệt đối hay chữ số có nghĩa của số.

Như vậy, một số dấu phẩy động có dạng:

$$F = M \times A^E$$

Trong đó: - Phần mũ E chiếm 1 byte với bit dấu S1 và 7 bit mũ E (số dấu phẩy động thông dụng).

- Định trị M chiếm 3 byte với bit dấu S2 và 16 bit M (số dấu phẩy động thông dụng).

$$f = (EEEEEEEE.S MMM MMMM MMMM MMMM)$$

$$\text{hay } f = (-1)^S \times 1.MMM MMMM MMMM MMMM \times 2^{(EEEEEEEE) - 127}$$

Như vậy, số dấu phẩy động thông dụng chiếm tổng cộng 4 byte, 24 bit và chứa trong bộ nhớ theo thứ tự sau:

- *byte thấp nhất*: ô nhớ thứ N, chứa byte cuối cùng của phần định trị M.
- *byte thấp thứ 2*: ô nhớ thứ N+1, chứa byte giữa của phần định trị M.
- *byte thấp thứ 3*: ô nhớ thứ N+2, chứa byte cao có dấu S2 và 7 bit của định trị M.
- *byte cao nhất*: ô nhớ thứ N+3, chứa bit dấu S2 và 7 bit của phần mũ E (± 127).

Do đó, đối với dấu phẩy thông dụng dạng mũ 2 có:

Số nhỏ nhất là 1.2^{-127} và số lớn nhất là $2^{23} \times 2^{\pm 127}$

Phần định trị M có 16 bit với bit lớn nhất là 1 số = 0 khi mũ = 0.

Phần mũ không ở dưới dạng bù 2 mà dạng âm với độ lệch -127.

Ví dụ: $101.0E+0 = 5.2^0 = 5.1 = 5.$

$101.E+3 = 0,625.2^3 = 5/8 \times 8 = 5.$

Khi tính toán với số dấu phẩy động, ta cần lưu ý:

- *Phép tính cộng hay trừ*: phải đưa hai số có phần mũ như nhau (cùng bậc). Khi đó, chỉ cần cộng hay trừ số nguyên của phần định trị M.

- *Phép tính nhân hay chia*: thao tác trên cả hai phần:

+ Nhân hay chia hai phần định trị với nhau.

+ Lấy tổng hay hiệu số của hai phần mũ của hai số.

2. Chuẩn hóa dấu phẩy động

Số dấu phẩy động phải được chuẩn hóa, tức đưa về dạng của chuẩn để có dạng thống nhất chung cho các số và có độ chính xác cao nhất.

Dạng chuẩn có:

- Bit lớn nhất của nó có giá trị bằng 1 về bên phải liền ngay sau dấu phẩy nhị phân. Nếu bit cao nhất này không phải là 1, việc chuẩn hóa sẽ dịch phần định trị về bên trái, cho tới khi bằng 1 liền sau dấu phẩy nhị phân.

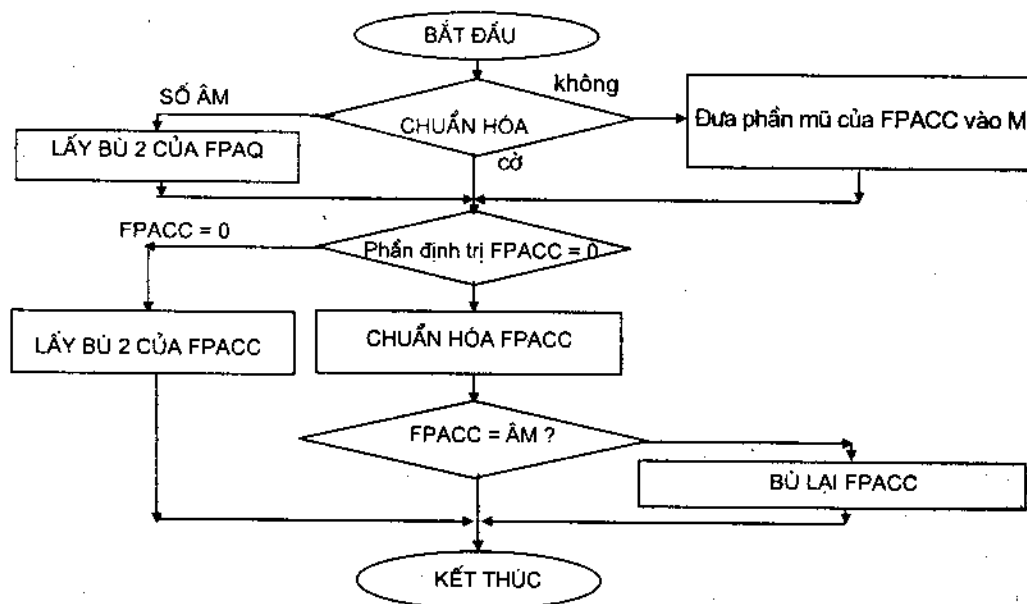
- Mỗi lần dịch sang trái một vị trí, giá trị của phần số mũ phải giảm đi 1 giá trị để giá trị của số luôn luôn không đổi.

- Giá trị kết quả của phần định trị sẽ là một số lớn hơn hoặc bằng 0,5 và nhỏ hơn 1.

Ví dụ: số chưa chuẩn hóa là 0.0001 1011 1000 1100 0011 010 E + 0.

số đã chuẩn hóa là 0.1101 1100 0110 0001 1010 000 E - 3.

Hình 5-3 là thuật toán của chương trình chuẩn hóa số dấu phẩy động.



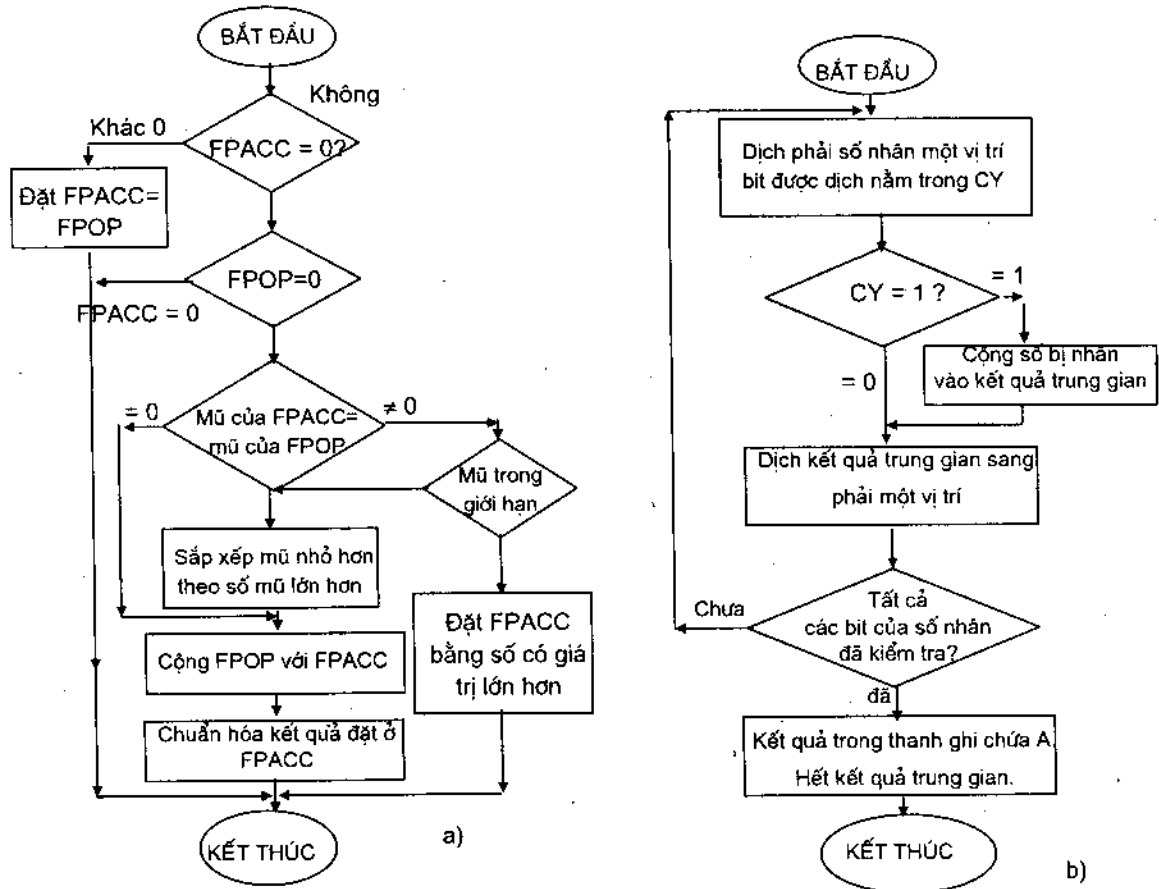
Hình 5-3. Giải thuật của chương trình chuẩn hoá dấu phẩy động

3. Cộng trừ hai số dấu phẩy động

Giải thuật của chương trình cộng hai số dấu phẩy động như hình 5-4a.

Giải thuật của chương trình nhân hai số dấu phẩy động như hình 5-4b.

Các ví dụ 5-24 và 5-25 cho các chương trình nhân và chia hai số dấu phẩy động.



Hình 5-4. Giải thuật của chương trình cộng (a) và nhân (b) hai số dấu phẩy động

; Ví dụ 5-24: Nhân hai số dấu phẩy động			
FMUL:	; xét dấu	RET	
MOV A, X1	; xét dấu X1	NOUNF:	
MOV C, ACC.7		JZ NOOVF	
MOV XS, C	; lưu bit dấu	SETB ZOV	
MOV A, Y1	; xét dấu	RET	
MOV C, ACC.7		NOOVF:	
MOV YS, C		MOV A, X1	; xét dấu X1
JB XS, FMUL_2		SETB ACC.7	
JB YS, FMUL_1		MOV X1, A	
CLR ZS		MOV A, Y1	
ASJMP FMUL_4		SETB ACC.7	
FMUL_1:	; xác lập dấu ZS	MOV Y1, A	
SETB ZS	; ZS = 1	LCALL MULIO	; nhân phần định trị
AJMP FMUL_4		MOV A, Z3	
FMUL_2:	; xác lập dấu YS		; nếu MSB của XY là 0 thì định trị M
JB YS, FMUL_3			; bằng 1 và tăng mũ E
SETB ZS : YS = 1		JB ACC.7, XYOK	
		INC R2	

AJMP FMUL_4 FMUL3: CLR ZS FMUL4: MOV R2, X2 ; xét dấu X2 MOV R3, Y2 CJNE R2, #0, XNZ MOV X2, #0 MOV Y2, #0 MOV R2, #0 RET XNZ: CJNE R3, #0, YNZ MOV Z2, #0, YNZ MOV Z1, #0 MOV Z0, #0 RET YNZ: MOV A, R2 ADD A, R3 MOV B, C CLR C SUBB A, #128 MOV R2, A MOV A, B SUBB A, #0 JNC NOUNT MOV Z2, #0 MOV Z1, #0 MOV Z0, #0	CJNE R2, #0, EXP_0K SETB ZOV RET EXP_0K: CLR C MOV A, Z1 RLC A MOV A, Z2 RLC A MOV Z0, A MOV A, Z3 RLC A MOV C, ZS MOV ACC.7, C MOV Z1, A MOV Z2, R2 RET XYOK: MOV Z0, Z2 ; byte thấp của M MOV A, Z3 ; byte cao của M MOV C, ZS MOV ACC.7, C ; bit dấu thay chỗ ở Acc.7 MOV Z1, A MOV Z2, R2 ; lấy mũ RET
--	---

: Ví dụ 5-25: Chia hai số dấu phẩy động	
FDIV: ; lấy mũ số bị chia MOV A, Y2 ; xét dấu Y2 JNZ FD_1 ; tiếp tục nếu Y khác 0 SETB ZOV ; xác lập cờ tràn nếu Y = 0 RET ; kết thúc FD_1: ; xác lập dấu ZS MOV A, X1 ; xét dấu X1 của M MOV C, ACC.7 ; đọc dấu vào C MOV XS, C ; gửi C vào XS SETB ACC.7 ; xác lập ACC.7 lên 1 MOV X1, A ; gửi vào X1 MOV A, Y1 ; xét dấu Y1 MOV C, ACC.7 ; đọc bit dấu MOV YS, C SETB ACC.7 MOV Y1, A JB XS, FD_2 JB YS, FD_2 CLR ZS SJMP FD_5 FD_2: SETB ZS ; xác lập ZS SJMP FD_5 ; nhảy tới FD_5 FD_3: JB YS, FD_4 ; nếu YS = 1, nhảy tới FD_4 SETB ZS SJMP FD_5	FD_4: CLR ZS FD_5: MOV R7, X2 MOV R6, X1 MOV R5, X0 MOV R4, Y2 MOV R3, Y1 MOV R2, Y0 ACLL SUB16 MOV C, ZOV MOV ZADJ, C MOV A, R7 MOV B, #0 JB ZADJ, FD_6 ADD A, #127 MOV B.0, C SIMP FD_7 FD_6: ADD A, #126 MOV B.0, C FD_7: CLR C ; xóa C SUBB A, R4 ; trừ mũ của Y MOV ZEXP, A ; lưu mũ vào ZEXP MOV A, B ; lấy tổng byte cao SUBB A, #0 ; hoàn thành phép trừ JNC FC_8 ; tràn nếu C = 1 MOV Z0, #0 ; cho kết quả bằng 0

RET	; kết thúc	FD_9:	
FD_8:	; mũ trong ZEXP, dấu trong ZS và	MOV A, Z4	; hoặc xác lập C
	; thương trong M	MOV C, ACC.0	
MOV X3, R6	; tạo X là số 32 bit	MOV A, Z3	; dịch Z2, Z3 có C
MOV X2, R5	; nhân X với 2^{16}	RRC A	
MOV X1, #0		MOV Z3, A	
MOV X0, #0		MOV A, Z2	
MOV Y1, R3		RRC A	
MOV Y0, R2		MOV Z0, A	; M byte thấp
LCALL DIV32	; gọi phép chia 32	MOV Z3, A	; lấy M byte cao
JNB ZADJ, FD_9	; nếu ≥ 1 , Z4 = 1	MOV ACC.7, C	; đặt dấu
MOV Z0, Z2		MOV Z1, A	; cấu trúc M byte cao
MOV A, Z3	; lấy M byte cao	MOV Z2, ZEXP	
MOV C, ZS	; lấy dấu	RET	
MOV ACC.7, C			
MOV Z1, A	; cấu trúc bit cao của M		
MOV Z2, ZEXP	; lấy số mũ		
RET			

CÂU HỎI VÀ THỰC HÀNH

I. Câu hỏi

1. Hãy so sánh giải thuật của phép tính cộng và nhân.
2. Hãy so sánh giải thuật của phép tính trừ và chia.
3. Hãy vẽ giải thuật và lập trình cho phép tính cộng và trừ hai số 16 bit (2 byte).
4. Hãy vẽ giải thuật và lập trình cho phép tính nhân hai số 8 bit bằng phương pháp cộng nhiều lần số bị nhân.
5. Hãy vẽ giải thuật và lập trình cho phép tính nhân hai số 8 bit bằng phương pháp trừ nhiều lần số bị chia.
6. Hãy vẽ giải thuật và lập trình cho phép tính nhân hai số 8 bit bằng phương pháp dịch trái số bị nhân và cộng nhiều lần số bị nhân theo sự phân tích số nhân thành tổng của lũy thừa cơ số 2 và cộng với số tự nhiên.
7. Hãy vẽ giải thuật và lập trình cho phép tính nhân hai số 8 bit bằng phương pháp dịch trái số bị nhân và cộng nhiều lần số bị nhân theo sự kiểm tra bit 1 của số nhân.
8. Thử tìm giải thuật cho phép chia bằng phương pháp dịch phải số bị chia và trừ nhiều lần số chia.
9. Hãy vẽ giải thuật và lập trình cho phép tính nhân hai số 8 bit bằng phương pháp dịch trái số bị nhân và cộng nhiều lần số bị nhân theo sự phân tích số nhân thành tổng của lũy thừa cơ số 2 và cộng với số tự nhiên.
10. Hãy vẽ giải thuật và lập trình cho phép tính nhân hai số 8 bit bằng phương pháp dịch trái số bị nhân và cộng nhiều lần số bị nhân theo sự kiểm tra bit 1 của số nhân.
11. Thử tìm giải thuật cho phép chia bằng phương pháp dịch phải số bị chia và trừ lần số chia.

II. Thực hành

1. Quan sát các thanh ghi liên quan tới ALU (A, B, PSW v.v...) trên chương trình PINNACLE.
2. Lập trình cho phép cộng, trừ cho các số không dấu (thực hiện các ví dụ liên quan).
Soạn thảo, dịch và cho chạy chương trình cộng và trừ (cho chạy các ví dụ có trong chương).
3. Lập trình cho phép cộng, trừ cho các số có dấu (thực hiện các ví dụ liên quan).
Soạn thảo, dịch và cho chạy chương trình cộng và trừ (cho chạy các ví dụ có trong chương).
4. Lập trình cho phép nhân, chia cho các số không dấu 8 bit và 16 bit và có dấu.
Soạn thảo, dịch và cho chạy chương trình nhân và chia (cho chạy các ví dụ có trong chương).
5. Lập trình để nhân hai số 15D với 7D bằng phương pháp dịch trái và cộng liên tiếp nhiều lần.

Chương 6

NHÓM LỆNH LOGIC VÀ PHÉP CHUYỂN ĐỔI MÃ

Mục đích của chương:

- Nghiên cứu các hành động xử lý logic cho một byte 8 bit (VÀ-ANL, HOẶC-ORL, LOẠI TRỪ-XRL và DỊCH BIT) của khối xử lý số học-logic ALU của VĐK.
- Nghiên cứu các hành động xử lý logic cho một bit với cờ nhớ C (ANL C, bit, ORL C, bit).
- Nghiên cứu ứng dụng của hai nhóm lệnh logic trên trong các phép toán nhân và chia nhị phân và biến đổi mã của dữ liệu.

6.1. NHÓM LỆNH XỬ LÝ LOGIC

Ngoài tính toán số học (cộng, trừ, nhân, chia), khối ALU của CPU của VĐK còn xử lý các phép tính logic (VÀ-AND, HOẶC-OR và LOẠI TRỪ-XOR) cho dữ liệu một byte và bit.

1. Nhóm lệnh logic cho dữ liệu theo byte

Với dữ liệu một byte (8 bit), VĐK thuộc họ MCS 51 có các lệnh logic như ở bảng 6-1.

Bảng 6-1. Nhóm lệnh logic cho byte

Lệnh	Hành động	Các phương pháp địa chỉ				Thời gian
		Dir.	Ind.	Reg.	Imm.	
ANL A, <BYTE>	A = A. AND. <byte>	X	X	X	X	1
ANL<BYTE>, A	<byte> = <byte>. AND. A	X				1
ANL<BYTE>, # data	<byte> = <byte>. AND. # data	X				2
ORL A, <byte>	A = A. OR. <byte>	X	X	X	X	1
ORL<byte>, A	<byte> = <byte>. OR. A	X				1
ORL<byte>, # data	<byte> = <byte>. OR. # data	X				2
XRL A, <byte>	A = A. XOR. <byte>	X	X	X	X	1
XRL<byte>, A	<byte> = <byte>. XOR. A	X				1
XRL<byte>, # data	<byte> = <byte>. XOR. # data	X				2
CPL A	A = 00H				Chỉ trong A	1
CPL A	A = NOT. A				Chỉ trong A	1
RL A	quay A về trái 1 bit không qua cờ C				Chỉ trong A	1
RL CA	quay A về phải 1 bit qua cờ nhớ C				Chỉ trong A	1
RR A	quay A về trái 1 bit không qua cờ C				Chỉ trong A	1
RR CA	quay A về bên trái 1 bit qua cờ nhớ C				Chỉ trong A	1
SWAP A	hoán vị 4 bit cao với 4 bit thấp trong A				Chỉ trong A	1

Đặc điểm chung của các lệnh như sau:

- Có ba lệnh AND (ANL), OR (ORL) và XOR (XRL) của thanh ghi chứa A và <byte>, trong đó <byte> có 4 chế độ địa chỉ là trực tiếp (DIR), gián tiếp (IND.), thanh ghi (REG.) và tức thì (IMM.)
- Có hai lệnh AND (ANL), OR (ORL) và XOR (XRL) của <byte> (địa chỉ trực tiếp – DIR) với A và #data (địa chỉ tức thì).

Các lệnh trên đều có hai toán hạng, trong đó chủ yếu là trong thanh ghi A và trong <byte> với địa chỉ nhớ gián tiếp (IND.) hoặc thanh ghi (REG).

- Có hai lệnh XÓA (CLR) và ĐẢO bit (CPL) cho thanh ghi chứa A.
- Có 4 lệnh dịch bit về bên trái (RL) hay về bên phải (RR) không qua bit cờ nhớ C (RL A, RR A), có qua bit cờ nhớ C (RLC, RRC A) chỉ với thanh ghi chứa A.
- Có một lệnh SWAP A (hoán chuyển) chỉ với thanh ghi A, để trao đổi 4 bit thấp với 4 bit cao của A (4 bit cao thành 4 bit thấp và 4 bit thấp thành 4 bit cao).

a) Lệnh VÃ - ANL

Lệnh VÃ này lấy VÃ logic giữa các bit tương ứng của hai byte, giữa byte của thanh ghi A với <byte> (có địa chỉ gián tiếp hay dữ liệu tức thì). Cụ thể là các lệnh sau:

- ANL A, <byte> : cho A = A .AND. <byte> với <byte> theo 4 phương pháp địa chỉ.
- ANL <byte>, A : cho <byte> = <byte> .AND. A với <byte> theo địa chỉ trực tiếp.
- ANL <byte>, #data : cho <byte> = <byte> .AND. #data theo địa chỉ trực tiếp.

Các bit kết quả của lệnh này chỉ có giá trị là 1 khi và chỉ khi cả hai bit nhị phân tương ứng của cả hai toán hạng bằng 1.

Người ta dùng lệnh này để :

- Che các bit không để ý đến (ANL với các bit 0) trong đó thường che 4 bit cao hay 4 bit thấp của một byte.
- Lọc các bit cần chú ý tới (ANL với các bit 1) như cần kiểm tra một bit trạng thái của thiết bị ngoài (ví dụ 6-1).

b) Lệnh HOẶC- ORL

Lệnh này lấy HOẶC các bit tương ứng của <byte> với A hay #data. Phép lấy HOẶC dùng để xác lập bit kết quả lên 1 khi một trong hai bit tương ứng của toán hạng bằng 1.

Ví dụ 6-1 dùng toán tử VÃ- ANL để che hay lọc bit của một số bất kỳ.

Ví dụ 6-2 dùng toán tử HOẶC-ORL để xác lập một số bit hay biến đổi một số BCD của số tự nhiên thành mã ASCII.

Ví dụ: A = 10110001B #data = 11011010B ANL A, #data = 10010000B	Ví dụ: A = 10110001B #data = 11011010B ORL, #data = 1111011B
; Ví dụ 6-1: Dùng VÃ để lọc bit	; Ví dụ 6-2: Dùng HOẶC để đổi sang ASCII
<pre>ORG 00H MOV A, #0D7H ; nạp 0D7H vào A ANL A, #0F0H ; VÃ với #0F0h (che 4bit thấp) SWAP A ; biến đổi thành 4bit thấp SJMP \$ END</pre>	<pre>ORG 00H MOV A, # 07H ; nạp # 07H vào thanh ghi A ORL A, #30H ; Hoặc với #30H thành 37H SJMP \$ END</pre>

c) Lệnh LOẠI TRỪ - XRL

Lệnh này thực hiện việc loại trừ (hay khử nhau) giữa các bit của thanh ghi A với bit tương ứng của <byte> hay với #data. Khi hai bit giống nhau, kết quả bằng 0, hai bit khác nhau, kết quả bằng 1.

Nếu XOR của A với chính nó, kết quả trong A có các bit bằng 0, thường dùng để xóa thanh ghi (XRL A,A).

Người ta hay dùng lệnh này để :

- Xóa một thanh ghi khi XOR với chính nó.

Ví dụ: XRL A, A cho kết quả A = 0.

- Kiểm tra hai thanh ghi hay một thanh ghi và một giá trị có bằng nhau không.

Ví dụ: XRL A, R0 cho kết quả A = 0 thì A = R0.

XRL A, R0 cho kết quả A ≠ 0 thì A ≠ R0.

Ví dụ 6-3 giới thiệu chương trình dùng lệnh XRL A, R3.

2. Các lệnh xóa CLR A và đảo (CPL) thanh ghi chứa A

Hai lệnh này chỉ thao tác trên thanh ghi A, xóa các bit của thanh ghi A về 0 và đảo các bit của A hay bù 1 của A, tức là: CLR A cho A = # 00H, các bit trở về 0.

CPL A (hay /A) hoặc lấy bù 1 của A. Đây chính là hàm NO logic của A.

Ví dụ, nếu A = 11001010B thì

CPL A = 00110101B

Chúng ta đã dùng lệnh này khi biến đổi một dữ liệu trong A thành dạng bù 1 (đảo bit) và bù 2 (bù 1+1) để tiến hành phép trừ ở chương 5.

3. Các lệnh dịch bit theo vòng kín về bên trái (RL A, RLC A) hay phải (RR A, RRC A)

Các lệnh này còn có tên là quay bit (Rotate) theo một vòng kín không qua bit C (RL A, RR A) hay qua bit C (RLC A, RRC A) như hình 6-1.

Về chiều dịch, có hai loại:

- Dịch sang trái là RL A và RLC A (hình 6-1a).
- Sang phải là RR A và RRC A (hình 6-1b).

Về thể dịch: - Không qua bit cờ nhớ C như RL A và RR A.

- Qua cờ nhớ C như RLC A và RRC A.

Đặc điểm chung của các lệnh này là:

- + Chỉ tác động lên các bit của thanh ghi A.
- + Mỗi lần chỉ thực hiện dịch 1 vị trí bit, các bit trống ở hai đầu được điền bởi bit 0.
- + Khi dịch qua C, thì bit C tạo với 8 bit của A thành một vòng kín (hình 6-1) và C nhận và truyền giá trị bit của nó với các bit lân cận của A.

Người ta dùng các lệnh quay bit này trong các trường hợp sau:

- Nhân giá trị trong thanh ghi A với 2 sau mỗi lần dịch trái hay chia cho 2 sau mỗi lần dịch phải. Như vậy, sau n lần dịch, giá trị của A được nhân với lũy thừa 2 lên n lần (RL A) hay chia cho 2^n (RRA), (ví dụ 6-4).

Ví dụ 6-3: Chương trình dùng lệnh XRL để kiểm tra giá trị đọc vào từ cổng song song

```
ORG 00H
MOV P0, #00H ; xóa P0
MOV P1, #0FFH ; P1 là cổng vào
MOV R3, #75H ; nạp #75H vào R3
MOV A, P1 ; đọc cổng vào P1
XRL A, R3; kiểm tra giá trị đọc vào = #75H
JNZ EXIT ; tới EXIT nếu khác #75H
MOV P0, 99H ; đưa 99H ra cổng 99H
EXIT: SJMP $ ; kết thúc tại chỗ
END ; kết thúc dịch
```

Ví dụ: A = 11010100B thì:

RL A = 10101000B

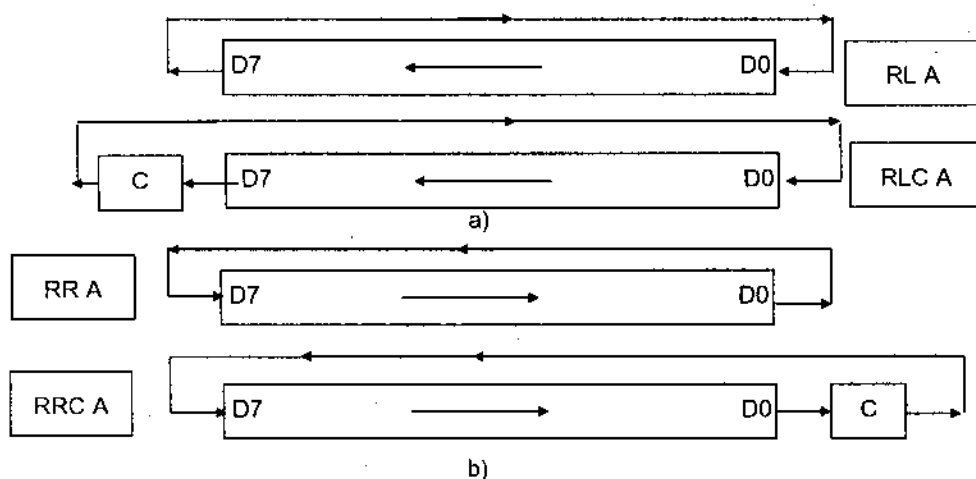
RR A = 01101010B

C = 0 và RLC A = 10101000B và C = 1

C = 1 và RLC A = 10101001B và C = 1

C = 0 và RRC A = 01101010B và C = 0

C = 1 và RRC A = 01101010B và C = 0



Hình 6-1. Các lệnh quay trái (a) và quay phải (b)

- Nhân trực tiếp bằng phép cộng với số nhân đã dịch bit: kiểm tra số nhân xem có bit 1 không, nếu có, cộng số bị nhân với số nhân đã dịch đi từng ấy vị trí (xem ở dưới).
- Kiểm tra bit : kiểm tra bit D7 của thanh ghi A (lệnh RLC A) hay D0 (lệnh RRC A) (ví dụ 3-7)
- Kiểm tra bit 1 của phím nhấn trong một hàng (xem chương 14 tập II).
- Điều khiển hiển thị kiểu dôn kênh hay điều khiển mô tơ bước.
- Biến đổi dạng số BCD không nén thành BCD nén và ngược lại.

Ví dụ 6-4 giới thiệu chương trình dùng lệnh RL A.

Ví dụ 6-5 giới thiệu chương trình dùng lệnh RLC A.

Ta có thể dùng ví dụ 6-4, 6-5 cho các lệnh RR A và RRC A bằng cách thay bằng các lệnh RL A và RLC A.

<i>; Ví dụ 6-4: Dùng RL A để nhân hay chia A (RR A) cho 2</i>	<i>; Ví dụ 6-5: Dùng RLC hay RRC để kiểm tra bit D7 hay D0 của A</i>
<pre> ORG 00H MOV A, #07H ; nạp số 07D vào A MOV B, A ; nạp số 07D vào B RL A ; dịch trái A (thành 14D = 0EH) RL A ; dịch trái A (thành 28D = 1CH) ADD A, B; B × (2²+1) A = 07D × 5 = 35D = 23H SJMP \$ END </pre>	<pre> ORG 00H MOV A, #35H ; nạp số 35H (mã ASCII của ; 5D) RLC A ; bit D7 = C = 0 RLC A ; bit D6 = C = 1 MOV A, #35H ; nạp số 35H (mã ASCII của 7D) RRC A ; bit D0 = C = 1 RRC A ; bit D1 = C = 0 SJMP \$; kết thúc chờ END </pre>

4. Các lệnh logic theo bit

Với VĐK họ MCS-51 có hai nhóm lệnh logic theo bit C và bit hay /bit. Đó là:

- *ANL C, bit*: logic AND của C với bit của thanh ghi có khả năng gọi địa chỉ bit.
- *ANL C, /bit*: logic AND của C với bit đảo (/bit) của thanh ghi có khả năng gọi địa chỉ bit.
- *ORL C, bit*: logic OR của C với bit của thanh ghi có khả năng gọi địa chỉ bit.
- *ORL C, /bit*: logic OR của C với bit đảo (/bit) của thanh ghi có khả năng gọi địa chỉ bit.

Lệnh ANL C, bit dùng để kiểm tra bit C nếu biết trước giá trị của bit hoặc ngược lại.

Lệnh ORL C, bit dùng xác lập các bit còn lại của thanh ghi A, ngoài các bit đã xác lập từ trước.

Chúng ta sẽ nghiên cứu chi tiết hơn các lệnh logic theo bit trên ở chương 7.

6.2. CÁC PHÉP TÍNH NHÂN VÀ CHIA NHỊ PHÂN VỚI CÁC LỆNH LOGIC

Ở chương 5, chúng ta đã lập trình xử lý số học (cộng, trừ, nhân, chia) với các dữ liệu HEX, BCD nén. Ở đây, với các lệnh dịch bit (trái, phải) và cộng hay trừ nhị phân, chúng ta có thể thực hiện các phép tính nhân và chia hai số nhị phân.

1. Nhân nhị phân

Có hai số nhị phân, số bị nhân và số nhân, bằng lập trình, thực hiện phép nhân chúng với nhau nhưng không dùng lệnh MUL mà chỉ dùng lệnh dịch (trái, phải) và cộng.

Muốn vậy, người ta thực hiện phép nhân như ở hình 6-2 cho nhân số 23D với 5D theo kiểu nhị phân mà không dùng lệnh MUL AB. Ví dụ 6-6 giới thiệu chương trình nhân 23Dx5D = 115D.

2. Chia nhị phân

Có hai số nhị phân, số bị chia và số chia, bằng lập trình, thực hiện phép chia chúng với nhau nhưng không dùng lệnh DIV AB mà chỉ dùng lệnh dịch (trái, phải) và trừ.

Muốn vậy, người ta thực hiện phép chia như ở hình 6-3 cho chia hai số 23D với 5D theo kiểu nhị phân mà không dùng lệnh DIV AB.

Ví dụ 6-7 giới thiệu chương trình chia 210D : 5D = 115D.

6.3. BIẾN ĐỔI MÃ ASCII - HEX

Dữ liệu đưa vào máy vi tính hay VĐK thường từ bàn phím và đưa ra màn hình đều dưới dạng mã ASCII 8 bit. Chúng ta phải đổi dạng mã này sang dạng HEX và ngược lại. Với các lệnh logic theo byte, chúng ta có thể lập trình để biến đổi chúng.

1. Biến đổi mã ASCII sang mã HEX của số BCD có nén

Mã ASCII cho các số HEX từ 00H tới 0FH theo bảng 6-2.

Từ bảng 6-2 ta thấy:

- Các số tự nhiên hay dạng HEX từ 0D tới 9D có mã ASCII từ 30H tới 39H (hơn 30H).

- Các số HEX từ 0AH tới 0FH có mã từ 42H tới 46H.

- Các số HEX từ 0aH tới 0fh có mã từ 62H tới 66H (hơn 20H).

Ví dụ 6-8 là chương trình con biến đổi mã ASCII sang mã HEX của số BCD có nén. Chỉ các ký tự là số tự nhiên từ 0D (mã từ 30H) tới 9D (mã 39H) và các ký tự A tới F (mã từ 41H tới 46H) hay a (mã 61H tới f (mã 66H mới biến đổi được thành mã ASCII).

; Ví dụ 6-6: Nhân $r2 \times r1$, kết quả trong $r4, r5$

```
ORG 00H ; r1 = số nhân, r2 = bị nhân, ; kết quả: r3 = byte
; cao, r4 = byte thấp

MOV R0, #8 ; số bit
MOV R2, #00010111B ; số bị nhân bằng 23D
MOV R1, #101B ; số nhân bằng 5D
MOV R3, #101B ; gửi i số nhân vào r3
NEXTBIT: MOV A, R3 ; số nhân vào A
RRC A ; dịch phải số nhân
MOV R3, A ; số nhân đã dịch vào r3
JNC NOADD ; nhảy tới noadd nếu C = 0
MOV A, R4 ; byte cao của kết quả trung gian
ADD A, R2 ; cộng với số bị nhân
MOV R4, A ; gửi vào r4 chứa kết quả trung gian
NOADD: MOV A, R4 ; để dịch số bị nhân 1 bit khi C = 0
RRC A ; dịch phải
MOV R4, A ; byte cao của tích cất vào r4
MOV A, R5 ; byte thấp của tích
RRC A ; dịch phải
MOV R5, A ; đưa lại vào R5
DJNZ R0, NEXTBIT ; trở về nextbit nếu r0 khác 0
SJMP $ ; kết thúc chờ
END ; kết thúc dịch
```

$$\begin{array}{r} 23D = 17H = 00010111B \\ 5D = 05H = \quad \times 101B \end{array}$$

$$\begin{array}{r} 00010111B \\ 00000000B \\ \hline 00010111B \end{array}$$

$$\begin{array}{r} 23D \times 5D = 0001110011B \\ 115D = 73H \end{array}$$

a)

$$\begin{array}{r} 23D = 17H = 00010111B \\ 5D = 05H = \quad \times 101B \end{array}$$

$$\begin{array}{r} 00010111B \\ 00000000B \\ \hline 00010111B \end{array}$$

$$\begin{array}{r} 23D \times 5D = 00001110011B \\ 115D = 73H \end{array}$$

b)

Hình 6-2. Phép nhân nhị phân dịch phải (a) và trái (b)

Bảng 6-2. Các mã ASCII của các số HEX (0H-FH)

Số tự nhiên	Số HEX	Mã ASCII	Số tự nhiên	Số HEX	Mã ASCII
0D	00H	30H	8D	38H	38H
1D	01H	31H	9D	39H	39H
2D	02H	32H	10D	0AH ; 0aH	41H ; 61H
3D	03H	33H	11D	0BH ; 0bH	42H ; 62H
4D	04H	34H	12D	0CH ; 0cH	43H ; 63H
5D	05H	35H	13D	0DH ; 0dH	44H ; 64H
6D	06H	36H	14D	0EH ; 0eH	45H ; 65H
7D	07H	37H	15D	0FH ; 0fH	46H ; 66H

2. Biến đổi mã HEX sang mã ASCII

Từ bảng 6-2 ở trên, ta thấy:

- Với các số tự nhiên từ 0 tới 9 ta có mã ASCII từ 30H tới 39H, nên mã ASCII được tính theo công thức:

Mã ASCII bằng tổng của mã HEX với 30H

- Trong chương trình, mã HEX của một số nào đó được gửi vào thanh ghi A rồi OR logic với 30H, ta sẽ có mã ASCII của số đó trong thanh ghi A.

	<i>; Ví dụ 6-7: chia r2/r1, r3- số dư, r4- số thương</i> <pre> ORG 00H MOV R0, #8 ; số bit MOV R1, #00000101B ; số chia bằng 5D MOV R2, #00010111B ; số bị chia = 16D+7D = 23D NEXTBIT: MOV A, R2; nạp bị chia vào A RLC A ; dịch trái qua C MOV R2, A ; bị chia đã dịch MOV A, R3 ; bị chia 1 phần vào A RLC A ; đưa bit C vào bit thấp nhất của A SUBB A, R1 ; trừ số chia JNC NOADD ; nếu C = 0, tới noadd ADD A, R1 ; hồi phục số dư ở r1 NOADD: MOV R3, A ; đưa trở lại r3 CPL C ; bù C MOV A, R4 ; r4 số thương RLC A ; dịch trái qua C MOV R4, A ; chuyển C vào bit thấp nhất của thương DJNZ R0, NEXTBIT ; giảm R0, nếu chưa bằng 0, ; trở về nexbit SJMP \$; kết thúc chờ END ; kết thúc dịch </pre>
--	--

Hình 6-3. Phép chia nhị phân

<i>; Ví dụ 6-8: Chương trình con biến đổi ASCII – HEX</i> <pre> ASCHEX: CJNE A, #'0', AH1 ; so với 30H AH1: JC AHBAD CJNE A, #'9'+1, AH2 ; so với 3AH AH2: JC AHVAL09 CJNE A, #'A', AH3 ; so với 41H AH3: JC AHBAD CJNE A, #'F'+1, AH4 ; so với 47H AH4: JC AHVALAF CJNE A, #'a', AH5 ; so với 61H AH5: JC AHBAD ; tới AHBAD nếu C = 1 </pre>	<pre> CJNE A, #'f'+1, AH6 ; so với 67H AH6: JNC AHBAD ; tới AHBAD, C = 0 CLR C ; xóa C SUBB A, #27H ; trừ đi 27H SJMP AHVAL09 ; tới AHVAL09 AHBAD: CLR HEXFLAG ; xóa cờ SJMP AHHEX ; tới AHHEX AHVAL09: CLR C ; xóa C SUBB A, #'0' ; trừ đi 30H SETB HEXFLAG ; xác lập cờ AHEX: NOP RET </pre>
--	---

6.4. BIẾN ĐỔI MÃ BCD

Trong chương 5, chúng ta đã đề cập tới số BCD là mã HEX của các số tự nhiên từ 0D tới 9D. Số BCD này thuận tiện để đọc các chữ số thập phân với các chữ số hàng vạn, hàng ngàn, hàng trăm, hàng chục và hàng đơn vị. Ví dụ 65535D. Số thập phân hay BCD này thuận tiện cho người dùng vì dễ đọc và đưa ra hiển thị trên đèn LED 7 đoạn. Nhưng trong máy vi tính hay VDK, lại dùng số dạng HEX để tính toán. Do đó, chúng ta phải biết chuyển đổi từ.

- Dạng mã ASCII sang dạng BCD hay thập phân.

- Dạng BCD sang dạng ASCII.

- Dạng BCD sang dạng HEX.
- Dạng HEX sang BCD hay thập phân.

1. Biến đổi hai mã BCD nén sang một byte mã HEX

Mỗi mã BCD nén chỉ cần 4 bit để lưu trữ. Do đó một byte 8 bit có thể biểu diễn được hai mã BCD nén.

Chương trình biến đổi hai mã BCD nén (vào từ bàn phím) thành một byte 8 bit dạng HEX (hay BIN) như ví dụ 6-9.

Mã BCD nén thứ nhất vào chứa trong 4 bit thấp sẽ dịch chuyển lên 4 bit cao rồi gửi vào thanh ghi B để dành chỗ cho mã BCD nén thứ hai ở 4 bit thấp.

Thanh ghi chứa A chứa mã BCD nén ở 4 bit thấp sẽ cộng với thanh ghi B thành hai mã BCD nén 8 bit chứa trong thanh ghi A, rồi ghi nhớ trong ô nhớ.

Ví dụ 6-9 giới thiệu lệnh gộp và chương trình biến đổi hai byte mã BCD sang một byte mã HEX rồi ghi nhớ trong bộ nhớ. Trong chương trình, ta cho chạy thử 4 trường hợp là nhỏ hơn mã 30H, trong khoảng 30H tới 46H (từ số 0H tới FH) và ngoài khoảng trên. Chỉ có mã BCD nén trong khoảng của các số tự nhiên (từ 0D tới 9D) mới được biến đổi và ghi vào các ô nhớ tương ứng. Cho chạy chương trình, quan sát bộ nhớ trong (IRAM), ta thấy được các kết quả biến đổi trên.

; Ví dụ 6-9: BCDHEX, biến đổi hai byte mã BCD nén sang một byte mã HEX	
<pre>MACRO @BCDHEX < 1?, 2?, 3?> MOV R0, #1 ; nạp địa chỉ nhớ đầu tiên INC R0 MOV A, #12 ; nạp số mã ASCII thứ nhất MOV @R0, A ; ghi vào ô nhớ số thứ nhất INC R0 MOV B, #13 ; nạp số mã ASCII thứ hai MOV @R0, B ; ghi vào ô nhớ số thứ hai DEC R0 MOV A, @R0 ; đọc số thứ nhất trong ô nhớ CALL BCDIN ; gọi BCDIN RL A ; dịch trái 4 lần RL A RL A RL A MOV B, A ; chuyển sang B INC R0; MOV A, @R0 ; đọc số thứ hai trong ô nhớ CALL BCDIN ; gọi BCDIN ADD A, B ; cộng A và B DEC R0 DEC R0</pre>	<pre>MOV @R0, A ; ghi kết quả vào ô nhớ SJMP KT ; tới kết thúc BCDIN: ; chương trình con nhận mã CLR C PUSH ACC SUBB A, #30H ; nhận mã A SCII của 0 tới 9 POP ACC JC KT ; nhảy tới KT nếu A < 30H PUSH ACC SUBB A, #47H ; mã ASCII của từ 0 tới 0FH POP ACC JNC KT ; nhảy tới KT nếu A > số FH YES: ANL A, #0FH RET KT: NOP ENDMAC ; kết thúc lệnh gộp ORG 00H ; chương trình chính @BCDHEX 30h, 32H, 23H; số nhỏ hơn 0D @BCDHEX 40h, 36H, 35H; số từ 0D tới 9D @BCDHEX 50h, 36H, 44H; số từ 0A tới 0F @BCDHEX 60h, 36H, 49H; số lớn hơn 0FH SJMP \$ END</pre>

2. Biến đổi 5 số BCD nén sang hai byte HEX

Một byte nhị phân hay HEX có thể chứa tới giá trị 255D hay ba chữ số BCD. Còn hai byte HEX có thể chứa tới số 65535 hay 5 chữ số BCD nén. Do đó, có thể biến đổi 5 chữ số BCD nén sang hai byte nhị phân hay HEX.

Nguyên tắc biến đổi là các số BCD nén (hay mã ASCII trừ đi 30H):

- Chỉ nhận chữ số ở trong dải từ 0D tới 9D (hay mã ASCII ở trong dải 30H tới 39H).
- Theo thứ tự, các số nhập vào được nhân với 10000D, 1000D, 100D, 10D và 1. Nói khác đi, mỗi số nhập vào được nhân liên tiếp với 10D tùy theo vị trí chữ số đó ở hàng chục ngàn (nhân với 10000D), hàng ngàn (nhân với 1000D), hàng trăm (nhân với 100D), hàng chục (nhân với 10D) và hàng đơn vị (không nhân).
- Biến đổi từ byte thấp sang byte cao.

Ví dụ 6-10 giới thiệu lệnh gộp và chương trình biến đổi với lệnh gộp. Đặc điểm của lệnh gộp và chương trình như sau là có nhiều thông số là địa chỉ nhớ ghi các số biến đổi, số các chữ số, các giá trị của các chữ số cần biến đổi.

; Ví dụ 6-10: BCDBIN, Biến đổi 5 số BCD sang hai byte BIN	
MACRO@BCDHEX <số chữ số?, địa chỉ nhớ?, các số?> MOV SP, #70h ; địa chỉ nhớ ngăn xếp MOV R5, #1 ; số chữ số (5) MOV R0, #12 ; địa chỉ nhớ đầu MOV A, #13 ; số thứ 1 MOV @R0, A ; ghi nhớ vào ô nhớ INC R0 ; tăng địa chỉ MOV A, #14 ; số thứ 2 MOV @R0, A INC R0 MOV A, #15 ; số thứ 3 MOV @R0, A INC R0 MOV A, #16 ; số thứ 4 MOV @R0, A INC R0 MOV A, #17 ; số thứ 5 MOV @R0, A MOV R0, #12 ; địa chỉ nhớ đầu BCD_0: CALL MUL10 ; nhân 10 CLR C MOV A, LBYTE ADD A, @R0 ; cộng với LBYTE MOV LBYTE, A ; gửi vào LBYTE JC BCD_1 BCD_2: INC R0 ; tăng địa chỉ nhớ DJNZ R5, BCD_0 ; giảm R5 tới 0 JMP KT ; về KT nếu R5 = 0 BCD_1: MOV A, HBYTE; đọc HBYTE INC A ; tăng HBYTE MOV HBYTE, A ; gửi vào HBYTE JMP BCD_2 ; trở về BCD_2	MUL10: CLR C; nhân 10D cho hai byte MOV A, LBYTE MOV B, #0AH MUL AB MOV LBYTE, A MOV A, HBYTE MOV R7, B MOV B, #0AH MUL AB ADD A, R7 ; cộng với byte cao MOV HBYTE, A RET KT: NOP ENDMAC ORG 00H LJMP MAIN ORG 30H ; ô nhớ ghi kết quả HBYTE DS 1 LBYTE DS 1 ORG 100H ; chương trình chính MAIN: @BCDBIN 5, 32h, 0, 0, 1, 2, 3; số 00123 BCD @BCDBIN 5, 42h, 0, 1, 2, 3, 4; số 01234 BCD @BCDBIN 5, 62h, 1, 5, 5, 3, 4; số 15534 BCD @BCDBIN 5, 62h, 2, 5, 5, 3, 4; số 25534 BCD @BCDBIN 5, 62h, 3, 5, 5, 3, 4; số 35534 BCD @BCDBIN 5, 62h, 4, 5, 5, 3, 4; số 45534 BCD @BCDBIN 5, 62h, 5, 5, 5, 3, 4; số 55534 BCD @BCDBIN 5, 62h, 6, 5, 5, 3, 4; số 65534 BCD SJMP \$ END

3. Biến đổi một byte dạng HEX sang ba số BCD

Một byte nhị phân hay HEX có thể chứa tới giá trị 255D hay ba chữ số BCD. Do đó, có thể biến đổi một byte nhị phân sang con số nhỏ hơn 255D. Nguyên tắc biến đổi là:

- Liên tiếp chia hay trừ đi 100D, 10D và tăng số lần trừ cho số hàng trăm hay số hàng chục.
- Số dư là số đơn vị.

Ví dụ 6-11 giới thiệu lệnh gộp và chương trình biến đổi trên. Các số nhị phân chỉ chứa trong 1 byte (tới 0FFH). Chương trình ví dụ cho biến đổi ba số 08H (nhỏ hơn 10D), 1DH (lớn hơn 10D, nhỏ hơn 100D) và 2FH (lớn hơn 100D và nhỏ hơn 256D).

Chú ý là chương trình trên có thể biến đổi cho các số BCD, khi nhập vào không có chữ H sau chữ số.

4. Biến đổi hai byte dạng HEX sang năm số hay BCD

Trong thực tế kỹ thuật, thường phải biến đổi nhiều số BCD nén sang hai byte dạng HEX. Ví dụ 6-12 giới thiệu lệnh gộp và chương trình biến đổi hai byte dạng HEX sang các số BCD (tới 5 chữ số và tới đa tới số 65535D cho FFFH).

Khác với lệnh gộp biến đổi một byte nhị phân là:

- Phải biến đổi hai byte nhị phân.

- Số dư hay số đơn vị ở byte trước phải chuyển sang byte sau bằng cách:
- + Đổi đơn vị cho byte sau bằng cách nhân với 256D (hay FFH).
- + Cộng với giá trị của byte sau rồi biến đổi bằng cách chia (hay trừ liên tiếp với 100D, 10D) như cách biến đổi ở byte thứ nhất.

: Ví dụ 6-11: Biến đổi một byte dạng HEX sang ba số BCD

MACRO @BIDEC8 <địa chỉ nhớ?, số?> MOV R0, #1 ; tham số địa chỉ nhớ MOV A, #12 ; tham số dữ liệu MOV @R0, A INC R0 ; tăng địa chỉ nhớ CLR C ; xóa C SUB100: SUBB A, #100 ; trừ 100D JC ADD100 ; tới ADD100 nếu nhỏ INC @R0 ; tăng R0 cho hàng trăm JMP SUB100 ; trở về SUB100 ADD100: ADD A, #100 ; cộng 100D cho số dư INC R0 ; tăng địa chỉ nhớ SUB10: SUBB A, #10 ; trừ 10D	JC UNIT ; tới UNIT nếu nhỏ INC @R0 ; tăng R1 cho hàng chục JMP SUB10 ; về SUB10 UNIT: ADD A, #10 ; cộng 10D INC R0 ; tăng địa chỉ nhớ MOV @R0, A ; gửi vào R2 cho đơn vị INC @R0 ; tăng đơn vị ENDMAC ORG 00H @BIDEC8 30H, 8H @BIDEC8 40H, 1DH @BIDEC8 50H, 0F2H SJMP \$ END
--	--

Vì hai byte nhị phân tức 16 bit có thể chứa tới số 65535D thập phân, tức 5 con số BCD (hay HEX).

Chương trình ở ví dụ 6-12 giới thiệu cách biến đổi các con số hàng chục ngàn (TTHOU), hàng ngàn (THOU), hàng trăm (HENDER), hàng chục (TEN) và hàng đơn vị (UNIT) thành hai byte nhị phân (HBYTE, LBYTE).

Trong chương trình ở ví dụ 6-12, ta biến đổi các số 1, 2, 3 ; số 00123 BCD; số 01234 BCD; số 15534 BCD; số 25534 BCD; số 35534 BCD; số 45534 BCD; số 55534 BCD và số 65534 BCD.

: Ví dụ 6-12: BCDHEX, Biến đổi 5 số BCD sang hai byte dạng HEX

MACRO @BCDBIN <số chữ số?, địa chỉ nhớ?, ; các số?> MOV SP, #70h PUSH ACC PUSH PSW MOV LBYTE, #00H MOV HBYTE, #00H MOV R5, #1 MOV R0, #12 ; địa chỉ nhớ MOV a, #13 ; ghi các số BCD MOV @r0, a INC r0 MOV a, #14 MOV @r0, a INC r0 MOV a, #15 MOV @r0, a INC r0 MOV a, #16 MOV @r0, a INC R0 MOV A, #17 MOV @r0, a MOV R0, #12 BCD_0 ; bắt đầu biến đổi CALL MUL10 ; biến đổi byte thấp CLR C	MOV HBYTE, A JMP BCD_2 MUL10: CLR C ; nhân 10D cho cả hai byte MOV A, LBYTE MOV B, #0AH MUL AB MOV LBYTE, A MOV A, HBYTE MOV R7, B MOV B, #0AH MUL AB ; nhân với 10D ADD A, R7 ; cộng với kết quả byte cao ; của phép nhân byte thấp MOV HBYTE, A RET KT: POP ACC POP PSW ENDMAC ORG 00H LJMP MAIN ORG 30H ; gán ô nhớ ghi kết quả HBYTE DS 1; ô nhớ đệm, lưu kết quả LBYTE DS 1 ORG 100H ; chương trình chính MAIN: @BCDBIN 5, 32h, 0, 0, 1, 2, 3 ; số 00123 BCD @BCDBIN 5, 42h, 0, 1, 2, 3, 4 ; số 01234 BCD
--	---

MOV A, LBYTE ADD A, @R0 ; cộng với số sau MOV LBYTE, A JC BCD_1 BCD_2: INC R0 DJNZ R5, BCD_0 JMP KT BCD_1: MOV A, HBYTE ; biến đổi byte cao INC A	@BCDBIN 5, 62h, 1, 5, 5, 3, 4 ; số 15534 BCD @BCDBIN 5, 62h, 2, 5, 5, 3, 4 ; số 25534 BCD @BCDBIN 5, 62h, 3, 5, 5, 3, 4 ; số 35534 BCD @BCDBIN 5, 62h, 4, 5, 5, 3, 4 ; số 45534 BCD @BCDBIN 5, 62h, 5, 5, 5, 3, 4 ; số 55534 BCD @BCDBIN 5, 62h, 6, 5, 5, 3, 4 ; số 65534 BCD SJMP \$ END
---	--

CÂU HỎI VÀ THỰC HÀNH

I. Câu hỏi

- Hãy so sánh hai lệnh cộng số học và lệnh cộng logic. Ứng dụng của lệnh cộng logic.
- Hãy so sánh hai lệnh trừ số học và lệnh XOR logic. Ứng dụng của lệnh XOR logic.
- Nêu bản chất của lệnh OR logic và ứng dụng.
- Hãy so sánh hai lệnh cộng logic và OR logic.
- Nêu các lệnh logic cho xử lý bit. Tại sao không có lệnh XOR logic cho bit ? Có thể dùng lệnh nào thay thế khi ta muốn loại trừ hai bit.
- Hãy so sánh các cặp lệnh dịch bit và ứng dụng của chúng.
- Phép nhân nhị phân dùng các lệnh dịch.
- Phép chia nhị phân dùng các lệnh dịch.
- Nêu cách kiểm tra một bit, trong đó có bit c, bit dấu của một số có dấu.
- Nêu các loại mã dữ liệu? So sánh chúng và kể các cách sử dụng trong kỹ thuật VĐK.
- Nêu cách thu nhận và biến đổi sang các mã khác và sử dụng mã ASCII.
- Nêu cách biến đổi sang các mã khác và sử dụng mã HEX hay BCD có nén.
- Nêu cách biến đổi sang các mã khác và sử dụng mã nhị phân B.
- Nêu cách biến đổi sang các mã khác và sử dụng mã thập phân D.

II. Thực hành

- Dịch, liên kết và cho chạy tất cả các ví dụ 6-1 tới 6-12 .
- Lập trình một chương trình nhân hai số dưới dạng mã BCD rồi đưa kết quả ra cổng song song.
- Lập trình một chương trình chia hai số dưới dạng mã BCD rồi đưa kết quả ra cổng song song.

Chương 7

CỔNG SONG SONG, NHÓM LỆNH XỬ LÝ BIT VÀ ĐIỀU KHIỂN SỐ

Mục đích của chương:

- Hiểu rõ cấu trúc, hoạt động và công dụng của các cổng vào ra song song (P0-P3) của VDK 8051/8052.
- Hiểu rõ ý nghĩa và công dụng của nhóm lệnh xử lý bit (SETB, CLR, CPL, JB, JNB, JBC) của VDK 8051/8052.
- Ứng dụng của các cổng, các thanh ghi xử lý theo bit và các lệnh xử lý bit trong lập trình VDK để phát xung và điều khiển.

7.1. CẤU TRÚC VÀ HOẠT ĐỘNG CỦA CÁC CỔNG SONG SONG

7.1.1. Đặc điểm chung

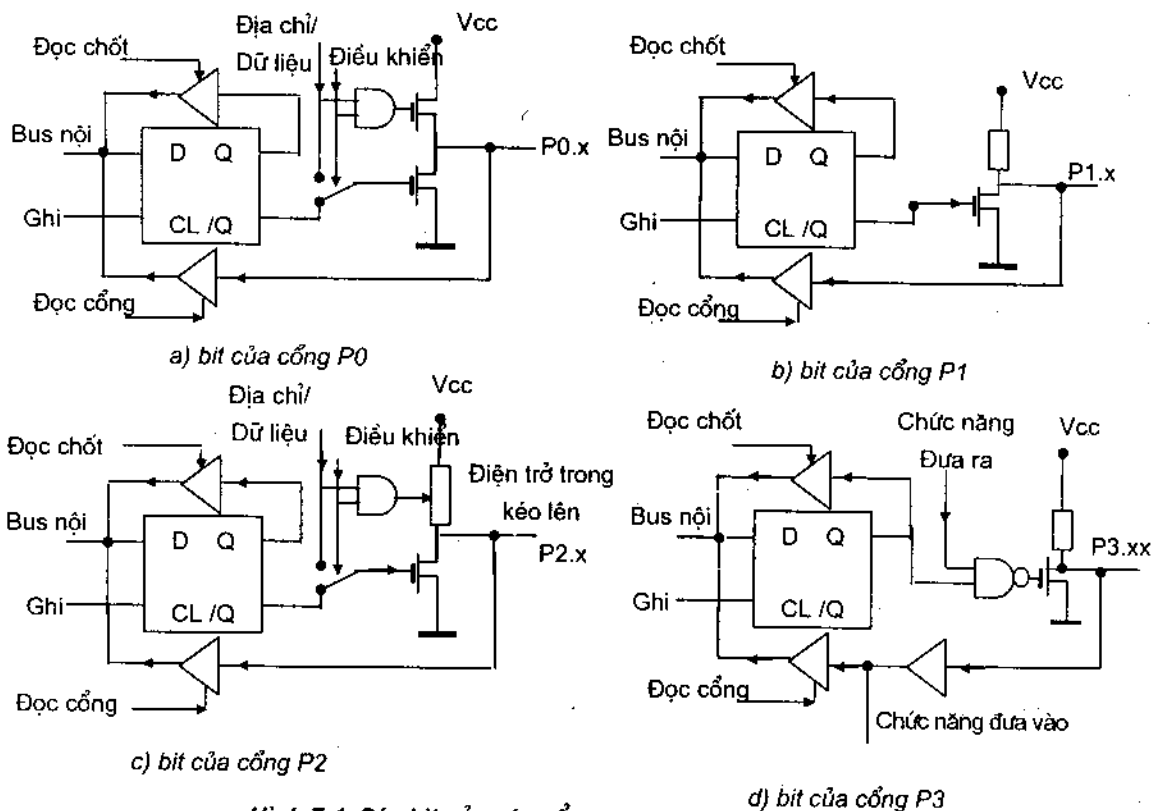
VDK 8051/8052 có 40 chân (hình 1-1, chương 1). Ngoài các chân nguồn điện nuôi (+5V-chân 40), 0V (GND-chân 20), Reset (RST-chân 9), nối tinh thể thạch anh (XTAL1-chân 19, XTAL2-chân 18), dùng nhớ trong hay ngoài (/EA/CPM-chân 31 nối đất), điều khiển đọc địa chỉ (ALE//PROG-chân 30) và điều khiển đọc nhớ chương trình bên ngoài (/PSEN-chân 29), còn lại là 32 chân của các cổng vào ra song song P0, P1, P2, P3.

Mỗi cổng vào/ra song song trên đều có 8 bit, có thể được xử lý theo byte (8 bit) hay từng bit và có cấu trúc bit hơi khác nhau cho các cổng khác nhau (hình 7-1).

Tuy các bit của chúng khác nhau, nhưng có các điểm chung giống nhau như sau:

- Là các thanh ghi đặc biệt, có địa chỉ liên tiếp là 80H, 90H, 0A0h và 0B0H.
 - Có khả năng gọi địa chỉ và xử lý theo từng bit với các địa chỉ liên tiếp, ví dụ, cổng P0 có các địa chỉ là 80H (P0.0), 81H (P0.1), 82H (P0.2), 83H (P0.3), 84H (P0.4), 85H (P0.5), 86H (P0.6) và 87H (P0.7).
 - đều có trigger D để chốt dữ liệu vào (điều khiển bởi Write to latch - ghi vào chốt từ CPU) và đọc ra dữ liệu vào (điều khiển bởi tín hiệu Read latch - đọc chốt từ CPU) và đọc dữ liệu ra (điều khiển bởi tín hiệu Read pin - chân đọc từ CPU).
 - đều có bộ khuếch đại đọc vào hay ra để tăng dòng chịu tải cho cổng.
 - đều có điện trở treo mức tín hiệu ra ở bên trong (bit cổng P1, P2, P3) hay điện trở treo mắc thêm ở bên ngoài (cổng P0).
 - Cổng P0 và P2 có điểm giống nhau ở:
 - + Sự dồn kênh (multiplexer) địa chỉ và dữ liệu, được điều khiển bởi tín hiệu Control (điều khiển).
 - + Đưa địa chỉ khi có đồng thời tín hiệu địa chỉ và điều khiển (ADDR.AND. CONTROL).
- Do đó, hai cổng này ngoài công dụng truyền dữ liệu vào/ra còn dùng để đưa tín hiệu địa chỉ thấp (A0-A7) trên P0 và địa chỉ cao (A8-A15) trên cổng P2 cho bộ nhớ ngoài.
- Cổng P3 còn có lối vào, chức năng vào tuần tự (alternate input function) để đưa các tín hiệu vào và ra điều khiển.

Khả năng chịu tải của bộ đệm ra của các cổng P1, P2 và P3 có thể điều khiển 4 lối vào của vi mạch loại 74LS TTL, còn cổng P0 có thể điều khiển 8 lối vào của vi mạch loại 74LS TTL.



Hình 7-1. Các bit của các cổng song song

7.1.2. Cổng P0 (P0.0-P0.7: chân 39-32)

Đây là cổng dồn kênh để trao đổi tín hiệu song song cho dữ liệu D0-D7 hay địa chỉ thấp (A0-A7).

- Cổng P0 dùng trao đổi tín hiệu về dữ liệu

Bình thường, đây là cổng ra, để vừa là đầu ra, vừa là đầu vào, phải có điện trở treo mắc bên ngoài (cỡ 10K ôm nối với nguồn nuôi +5V) và là dạng cực máng hở (tranzito trường) hay colectơ hở (tranzito thường). Do đó, sau khi khởi động, cổng P0 là cổng ra. Muốn P0 là cổng vào, phải lập trình để các bit P0 ở mức cao (bởi các lệnh MOV A, #0FFH và MOV P0, A).

Dữ liệu trao đổi tín hiệu (ra hay vào) có thể qua thanh ghi chứa A (bởi lệnh MOV A, P0 hay MOV P0, A) hay có thể đưa dữ liệu tức thì qua cổng P0 (bởi lệnh MOV P0, #data).

- Cổng P0 dùng trao đổi tín hiệu về địa chỉ thấp (A0-A7)

Khi P0 ghép nối với bộ nhớ ngoài, nó đưa ra các địa chỉ thấp (A0-A7) của bộ nhớ qua vi mạch chốt (ví dụ 74245). Tín hiệu chốt địa chỉ ALE = 1 (Address Ligne Enable) cho phép đường dây địa chỉ ghi các bit địa chỉ thấp này vào vi mạch chốt để đưa ra đồng thời với các địa chỉ cao (A8-A15) trên cổng P2 vào bộ nhớ ngoài (dung lượng 64Kbyte). Tín hiệu ALE = 1 này được CPU phát ra khi CPU thực hiện lệnh trao đổi tín hiệu với bộ nhớ ngoài bởi lệnh MOVX. Khi ALE = 0, cổng P0 cấp dữ liệu D0-D7.

7.1.3. Cổng P2 (P2.0-P2.7: chân 21-28)

Cổng này giống cổng P0, cổng P2 này cũng làm nhiệm vụ đưa dữ liệu (D0-D7) và địa chỉ cao (A8-A15) khi ghép nối với bộ nhớ ngoài.

- Cổng P2 dùng trao đổi tín hiệu về địa chỉ cao (A8-A15)

P2 nối trực tiếp với các địa chỉ cao (A8-A15) của bộ nhớ, không qua vi mạch chốt địa chỉ. CPU của VDK phát ra các tín hiệu địa chỉ cao này khi nó thực hiện lệnh đọc bộ nhớ ngoài MOVX.

- Cổng P2 dùng trao đổi tin về dữ liệu

Khác với cổng P0, cổng P2 này có điện trở kéo lên ở bên trong, nên không phải mắc điện trở kéo lên bên ngoài. Bình thường, sau khi Reset, đây là cổng ra và muốn là cổng vào, phải lập trình để các bit của P2 ở mức cao (bởi các lệnh MOV A, #0FFH và MOV P2, A).

7.1.4. Cổng P1 (P1.0-P1.7: chân 1-8)

Đây là cổng đơn kênh, chỉ trao đổi tin về dữ liệu (D0-D7). Giống cổng P2, cổng P1 này có điện trở kéo bên trong, sau khi Reset, P1 là cổng ra, muốn là cổng vào, phải lập trình để các bit của P1 ở mức cao (bởi các lệnh MOV A, #0FFH và MOV P1, A).

7.1.5. Cổng P3 (P3.0-P3.7: chân 10-17)

Cổng này giống cổng P1, có thể làm đầu vào hay đầu ra để trao đổi tin về dữ liệu. Nhưng cổng P3 này thường dùng làm nhiệm vụ khác (bảng 7-1). Từ bảng 7-1, ta thấy cổng P3 này có thể:

- Trao đổi tin nối tiếp: nhận dữ liệu RXD vào (chân P3.0 - số 10) hay truyền dữ liệu ra TXD (chân P3.1-số 11).

- Nhận các tín hiệu ngắt chương trình từ bên ngoài (ngắt ngoài) là INT0 (chân P3.2 - số 12), INT1 (chân P3.3 - số 13) và T2.EX (chân P1.1 - số 2) cho 8052.

- Nhận các tín hiệu đếm từ bên ngoài: T0 (P3.4 - chân số 14), T1 (P3.5 - chân số 15) và T2 (chân P1.0 - số 1) cho 8052.

- Phát các tín hiệu điều khiển ghi (/WR - P3.6 - chân 16) và đọc (/RD - P3.7 - chân 17): cho bộ nhớ ngoài khi CPU của VDK khi thực hiện các lệnh MOVX.

Bảng 7-1. Các tín hiệu của cổng P3

Bit P3	Chức năng	Chân
P3.0	Nhận dữ liệu nối tiếp (RXD)	10
P3.1	Phát dữ liệu nối tiếp (TXD)	11
P3.2	Nhận tín hiệu ngắt ngoài 0	12
P3.3	Nhận tín hiệu ngắt ngoài 1	13
P3.4	Nhận tín hiệu đếm T0	14
P3.5	Nhận tín hiệu đếm T1	15
P3.6	Phát tín hiệu Ghi (/WR)	16

7.2. CÁC THANH GHI VÀ Ô NHỚ XỬ LÝ THEO BIT

Như đã trình bày ở trên, các cổng song song có thể trao đổi tin theo byte (8 bit) và theo từng bit với các lệnh xử lý bit. Ngoài ra, bộ nhớ trong của VDK họ MCS51 có các thanh ghi xử lý theo bit như các thanh ghi đặc biệt SFR (hình 7-3) và khối nhớ RAM với địa chỉ từ 20H tới 2FH.

7.2.1. Các cổng song song xử lý bit

Các cổng song song P0 tới P3 có các bit có các địa chỉ như hình 7-3. Các bit này có thể đọc tin (bit vào) và đưa tin ra (bit ra) và tuân theo các lệnh xử lý bit (xem ở dưới).

7.2.2. Các thanh ghi thông dụng xử lý bit

Hình 7-2 giới thiệu cả các thanh ghi thông dụng xử lý theo cả byte (8 bit) và từng bit. Đó là:

- Các thanh ghi chứa A và B dùng để làm thanh ghi đệm dữ liệu, thực hiện các lệnh số học (A, và B trong các lệnh nhân và chia) và logic (chỉ thanh ghi A).

- Thanh ghi trạng thái PSW trong đó có các bit RS0, RS1 để xác định bảng (hay dải) thanh ghi đệm thông dụng R0-R7, bit C (cờ nhớ), bit AC (cờ nhớ phụ cho hiệu chỉnh số thập phân (cho số BCD nén) và bit tràn số OV (PSW.2).

- Thanh ghi điều khiển nguồn nuôi PCON, các thanh ghi ưu tiên ngắt IP và cho phép ngắt IE (xem chương 10).

- Thanh ghi điều khiển bộ định thời/đếm (Timer/Counter) TCON, TMOD và đệm dữ liệu TH0, TL0, TH1, TL1 cho VDK 8051 và T2CON, T2MOD và đệm dữ liệu TH2, TL2, RCAP2H, RCAP2L cho VDK 8052 (xem chương 8).

Địa chỉ byte

7F	Bộ nhớ RAM thông dụng							
30								
2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	77	76	75	74	73	72	71	70
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00
1F	Bảng 3 thanh ghi ngầm định R0-R7							
18								
17	Bảng 2 thanh ghi ngầm định R0-R7							
10								
	Bảng 1 thanh ghi ngầm định R0-R7							
0F								
08	Bảng 0 thanh ghi ngầm định R0-R7							
07								
00								

Hình 7-2. Các thanh ghi nhớ RAM địa chỉ bit

FF									
F0	F7	F6	F5	F4	F3	F2	F1	F0	B
E0	E7	E6	E5	E4	E3	E2	E1	E0	Acc
D0	D7	D6	D5	D4	D3	D2	D1	D0	PSW
B8	--	--	--	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
A8	AF	--	--	AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	Không định địa chỉ bit								SBUF
98	9F	9E	9D	9C	9B	9A	99	98	SCON
90	97	96	95	94	93	92	91	90	P1
8D	Không định địa chỉ bit								TH1
8C	Không định địa chỉ bit								TH0
8B	Không định địa chỉ bit								TL1
8A	Không định địa chỉ bit								TL0
89	Không định địa chỉ bit								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	Không định địa chỉ bit								PCON
83	Không định địa chỉ bit								DPH
82	Không định địa chỉ bit								DPL
81	Không định địa chỉ bit								SP
80	87	86	85	84	83	82	81	80	P0

Hình 7-3. Các thanh ghi thông dụng địa chỉ bit

- Thanh ghi điều khiển trao đổi tin nối tiếp SCON và đếm dữ liệu SBUF (xem chương 9).

7.2.3. Các bit của ô nhớ dành cho trạng thái

Dải nhớ RAM bên trong có địa chỉ từ 20H tới 2FH (dải màu vàng của bộ nhớ IRAM bên trong, quan sát được nhờ thực đơn VIEW \ IRAM) dành cho địa chỉ theo bit (từ 00H tới 7H ở hình 7-2) và dùng để ghi nhớ các bit trạng thái của chương trình (xem hình 4-6 và 4-7).

7.3. NHÓM LỆNH XỬ LÝ BIT (XỬ LÝ THEO ĐẠI SỐ BOOL)

Khác với vi xử lý, VĐK có nhóm lệnh xử lý theo từng bit. Họ MCS-51 có các lệnh xử lý theo từng bit như bảng 7-2.

Bảng 7-2. Các lệnh xử lý theo bit

	Lệnh	Hành động	Thời gian thực hiện
CLR C ; xoá bit C = 0	MOV C, bit	C = bit	1
CLR bit ; xoá bit = 0	MOV bit, C	bit = C	2
SETB C ; xác lập bit C = 1	ANL C, bit	C = C.AND.bit	2
SETB bit ; xác lập bit = 1	ANL C, /bit	C = C.AND.NOT.bit	2
CPL C ; lấy đảo bit C (C = /C)	ORL C, bit	C = C. OR. Bit	2
CPL bit ; lấy đảo bit (bit = /bit)	ORL C, /bit	C = C. OR.NOT.bit	2
JC rel ; nhảy tới địa chỉ rel ; khi C = 1	CLR C	C = 0	1
JNC rel ; nhảy tới địa chỉ rel ; khi C = 0	CLR bit	bit = 0	1
JB bit, rel ; nhảy tới địa chỉ rel ; khi bit = 1	SETB C	C = 1	1
JNB bit, rel; nhảy tới địa chỉ rel ; khi bit = 0	SETB bit	bit = 1	1
JBC bit, rel; nhảy tới địa chỉ rel ; khi bit = 1 và xoá ; bit = 0	CPL C	C = .NOT. C	1
	CPL bit	bit = .NOT. bit	1
	JC rel	Jump if C = 1	2
	JC rel	Jump if C = 0	2
	JB bit, rel	Jump if bit = 1	2
	JNB bit, rel	Jump if bit = 0	2
	JBC bit, rel	Jump if bit = 1; CLR bit	2

Nhóm lệnh xử lý bit này có các nhóm nhỏ là dịch chuyển bit, xoá, xác lập và lấy bù bit và rẽ nhánh chương trình theo giá trị của bit.

Nhóm lệnh xử lý bit này có những đặc điểm chung như sau:

- *Bit cờ nhớ C (bit PSW.7):* có vai trò tương tự thanh ghi tích lũy A (trong các phép tính số học và logic theo byte). Bit C này là một toán hạng, còn bit (của cổng song song, thanh ghi thông dụng theo bit, ô nhớ theo bit) là toán hạng thứ hai trong các lệnh xử lý bit như dịch chuyển bit (MOV), logic theo bit (ANL, XRL).

- *Toán hạng trong các lệnh xử lý bit:* là bit C và bit của các cổng song song P0-P3, các thanh ghi thông dụng theo bit và các bit nhớ IRAM có địa chỉ nhớ thuộc dải 20H-2FH.

7.3.1. Các lệnh dịch chuyển bit

Các lệnh này dịch chuyển giữa bit C và bit, có hai lệnh:

MOV bit, C ; chuyển bit vào bit C.

MOV C, bit ; chuyển bit C vào bit

Các lệnh trên để trao đổi bit với bit C, để thực hiện việc lưu giữ, các lệnh logic và nhảy chương trình theo bit.

7.3.2. Các lệnh xoá, xác lập và lấy bù hay đảo (hàm NOT) bit

- *Lệnh xoá bit:* đó là lệnh xoá bit C (CLR C cho C = 0) và xoá bit bất kỳ (CLR bit cho bit = 0) có địa chỉ theo bit.

- *Lệnh xác lập bit:* đó là lệnh xác lập bit C (SETB C cho C = 1) và xác lập bit bất kỳ (SETB bit cho bit = 1) có địa chỉ theo bit.

- *Lệnh đảo bit:* đó là lệnh đảo bit C (CPL C cho C = /C) và đảo bit bất kỳ (CPL bit cho bit = /bit) có địa chỉ theo bit.

Các lệnh này tương tự xóa thanh ghi A (MOV A, #00H), xác lập A (MOV A, #0FFH) và đảo A (CPL A) để xác lập giá trị ban đầu của bit C và bit bất kỳ có địa chỉ bit.

7.3.3. Các lệnh logic theo bit

Với bit, chỉ có hai loại lệnh logic VÀ (ANL) và HOẶC (ORL) cho hai toán hạng là bit C, bit hay /bit (bit đảo). Đó là:

- *ANL C, bit*: AND (VÀ) giữa hai bit C và bit, kết quả là bit C có giá trị 0 hay 1. Lệnh này dùng để kiểm tra hay so sánh bit C và bit xem có cùng là 1 (kết quả 1) hay có một bit 0 (kết quả là 0).

- *ORL C, bit*: OR (HOẶC) giữa hai bit C và bit, kết quả là một bit ghi lại trong C (giá trị 0 hay 1). Lệnh này dùng để xác lập bit lên 1 nếu một trong bit C hay bit là 1 hoặc = 0 khi cả hai bit = 0.

- *ORL C, /bit*: OR (HOẶC) giữa hai bit C và /bit (đảo bit), kết quả là một bit ghi lại trong C (giá trị 0 hay 1). Lệnh này dùng để xác lập bit C lên 1 nếu một trong bit C hay /bit là 1 hoặc = 0 khi cả hai bit = 0.

Các lệnh logic cho bit này không có lệnh loại trừ bit (như A.XOR. R). Muốn thực hiện lệnh loại trừ giữa hai bit1 và bit2 (C = bit1.XOR.bit2), ta phải thực hiện nhóm lệnh sau:

MOV C, bit1 ; gửi bit1 vào bit C

JNB bit2, OVER ; chờ bit2 = 0, nhảy tới OVER

CPL C ; đảo bit C, tức kết quả C = bit1, như vậy các lệnh từ nhãn OVER sẽ thực

OVER: (tiếp tục) ; hiện khi các bit1 và bit2 có giá trị trái nhau (nếu bằng nhau, kết quả bằng 0 ; nếu trái nhau, kết quả bằng 1).

7.3.4. Nhóm lệnh rẽ nhánh chương trình theo giá trị của bit

Ngoài các lệnh rẽ nhánh chương trình JC/JNC theo bit C, JZ/JNZ theo kết quả của A bằng 0 hay không bằng 0 đã xét ở chương ba, ở đây có thêm những lệnh rẽ nhánh chương trình theo giá trị của bit (xem bảng 7-2). Đó là các lệnh JB bit, JNB bit, JBC bit (trong đó có bit C), chúng chỉ rẽ nhánh chương trình khi điều kiện giá trị của bit được thỏa mãn là:

- *JB bit*: nhảy hay rẽ nhánh khi bit = 1.

- *JNB bit*: nhảy hay rẽ nhánh khi bit = 0.

- *JBC bit*: nhảy hay rẽ nhánh khi bit = 1 và xóa bit = 0.

Chúng ta sẽ gặp các lệnh trên trong những ứng dụng dưới đây về điều khiển đèn hiển thị, điều khiển thiết bị ngoài, trong đó có động cơ bước, điều khiển phát tín hiệu xung và nhận lệnh điều khiển. Lệnh JB bit, \$ nhảy hay rẽ nhánh tại chỗ khi bit = 1, được dùng để chờ giá trị của bit = 1.

7.4. LẬP TRÌNH ĐIỀU KHIỂN ĐÈN HIỂN THỊ

Trừ cổng P3 thường dùng riêng cho các tín hiệu vào/ra đặc biệt và điều khiển ghi/đọc khối nhớ ngoài và khi không có ghép nối bộ nhớ ngoài (không dùng các cổng P0, P2 để đưa địa chỉ), các cổng P0, P1, P2 dùng để trao đổi tín hiệu song song theo từng byte và theo từng bit. Với các lệnh xử lý theo bit, ta có ứng dụng khác nhau của cổng song song. Ở đây, các cổng song song có thể nối ghép cả 8 lối ra với các đèn chỉ thị LED rời để chỉ trạng thái 0 hay 1 của các bit của dữ liệu hoặc ghép nối với từng bit của cổng với các LED rời hay loa chỉ thị âm thanh hoặc lối vào dao động ký để biết được trạng thái 0 hay 1 của bit của một cổng.

7.4.1. Điều khiển đèn LED theo giá trị đơn bit

Các đèn diot phát sáng (LED – Luminescent emitted diot) được nối trực tiếp hay gián tiếp qua các vi mạch cổng logic colectơ hở và điện trở hạn chế dòng điện R sẽ phát sáng hay tắt khi có dòng điện chạy qua (cỡ miliampe) tùy thuộc vào trạng thái logic của các bit của các cổng song song (P0 tới P3). Bằng chương trình, ta có thể điều khiển sự hiển thị của các đèn, một số đèn hay một đèn.

Có thể mắc 8 đèn theo một byte với một cổng song song hay nhiều đèn với nhiều cổng song song. Khi đưa tín hiệu 1 ra một bit của cổng, đèn sáng và tín hiệu ra = 0, đèn tắt. Với các lệnh của chương trình, ta có thể:

- Cho tắt cả các đèn sáng: đưa tín hiệu 1 ra cổng.

- Cho tắt cả các đèn tắt: đưa tín hiệu 0 ra cổng.

- Cho một đèn sáng (hay tắt) và chạy từ phải sang trái (dùng lệnh dịch RL A).

- Cho một đèn sáng (hay tắt) và chạy từ trái sang phải (dùng lệnh dịch RR A).
- Cho hai đèn sáng (hay tắt) và chạy từ hai đầu vào giữa hay từ giữa ra (dùng lệnh dịch RL A và RR A, ORL A).

- Thay đổi thời gian chớp/tắt bằng thay đổi thời gian trễ DELAY.

Với các động tác cơ bản trên, ta có thể kết hợp để có thể điều khiển sáng của nhiều đèn theo nhiều cách thức mong muốn.

Ví dụ 7-1 giới thiệu lệnh gộp và chương trình điều khiển 8 đèn LED rời. Trong chương trình có dùng: - Lệnh gộp @WRITEPs để đưa dữ liệu (tham số 1) ra cổng P12 (tham số 2).

- Lệnh gộp @WRITEP để đưa dữ liệu từ thanh ghi chứa A ra cổng P11 (tham số 1).

- Chương trình con tạo trễ DELAY để để quan sát thời gian đèn sáng hay tối.

- Các lệnh dịch trái (RL A) và dịch phải (RR A) các giá trị của thanh ghi A để dịch chuyển một đèn sáng về bên trái hay về bên phải.

; Ví dụ 7-1: Điều khiển 8 đèn LED rời	
MACRO @WRITEPS < 1, 2> ; lệnh gộp ghi dữ liệu	DJNZ R7, LAP2 ; giảm r7 trở về lap2
MOV A, #1 ; dữ liệu	MOV A, #00H ; xoá a
MOV P12, A ; ghi ra cổng P12	SETB ACC.7 ; cho a = 80h
ENDMAC ; kết thúc lệnh gộp	MOV R6, #8H 8 ; số lần dịch chuyển
MACRO @WRITEP < 1> ; lệnh gộp ghi dữ liệu	LAP3: NOP
MOV P11, A ; ghi ra cổng P11	@WRITEP 3 ; ghi a ra cổng P3
ENDMAC ; kết thúc lệnh gộp	CALL DELAY ; gọi trễ
ORG 00H	RR A ; dịch a sang phải
LAP0: MOV R7, #8 ; số lần dịch chuyển	DJNZ R6, LAP3 ; giảm r6, lặp lại
@WRITEPS 55H, 0 ; ghi 55h ra P0	@WRITEPS 0AAH, 0 ; ghi 0aah ra P0
CALL DELAY ; gọi trễ	CALL DELAY ; gọi trễ
@WRITEPS 0AAH, 1 ; ghi 0aah ra P1	@WRITEPS 55H, 1 ; ghi 55h ra P1
CALL DELAY ; gọi trễ	CALL DELAY ; gọi trễ
MOV A, #00H ; xoá a	SJMP LAP0 ; lặp lại từ lap0
SETB ACC.0 ; xác lập bit cho a = 01h	DELAY: MOV R0, #250 ; tạo trễ
LAP2: NOP	DEL1: MOV R1, #250
@WRITEP 2 ; ghi a ra cổng P2	DEL2: DJNZ R1, DEL2
CALL DELAY ; gọi trễ	DJNZ R0, DEL1
RL A ; dịch a sang trái	RET
	END ; kết thúc dịch

Cho chạy chương trình của ví dụ 7-1, ta thấy:

- Các cổng P0 và P1 lần lượt sáng hay tắt những đèn số lẻ (đưa ra số 55H) hay số chẵn (đưa ra số 0AAH).

- Cổng P2 sáng một đèn chạy từ phải sang trái.

- Cổng P2 sáng một đèn chạy từ trái sang phải.

7.4.2. Điều khiển đèn LED đơn sáng dần và tắt dần

Chương trình của ví dụ 7-2 cho:

- Cổng P0 có các đèn sáng tăng dần từ bên phải sang trái.

- Cổng P1 có một đèn tắt chạy từ bên trái sang bên phải.

- Cổng P2 tắt một đèn chạy từ phải sang trái.

- Cổng P3 các đèn tắt dần chạy từ trái sang phải.

7.4.3. Điều khiển đèn theo bảng dữ liệu

Các ví dụ 7-1 và 7-2 điều khiển các đèn LED đơn sáng hay tắt theo các quy luật nhất định. Do đó, giải thuật của chương trình cũng tương đối phức tạp và chương trình dài vì có nhiều lệnh. Nếu điều khiển đèn hiển thị theo bảng dữ liệu ghi sẵn trong bộ nhớ (IRAM hay ROM trong) chương trình sẽ:

- Đơn giản về giải thuật vì chỉ có ghi bảng dữ liệu vào bộ nhớ, đọc từng dữ liệu và đưa ra cổng.
- Dữ liệu có thể có cấu trúc mong muốn nào đó, khó có thể dùng giải thuật để tạo ra bộ dữ liệu này.

Ví dụ 7-2: Sáng và tắt theo kiểu dồn

<pre> MACRO @WRITEP < 1> MOV P1, A ; ghi a ra cổng P1 ENDMAC ; kết thúc lệnh gộp MACRO @WRITEPS < 1, 2> MOV A, #1 ; ghi vào a MOV P1, A ; ghi từ a ra cổng P1 ENDMAC ; kết thúc lệnh gộp ORG 00H ; bắt đầu chương trình LAP0: MOV R7, #8 ; lặp lại 8 lần MOV A, #01H ; xác lập bit Acc.0 MOV R5, A ; lưu vào r5 LAP1: NOP @WRITEP 0 ; ghi ra P0 CALL DELAY ; gọi trễ MOV B, A ; lưu sang b MOV A, R5 ; lấy giá trị cũ RL A ; quay a sang trái MOV R5, A ; lưu vào r5 ADD A, B ; cộng bởi b DJNZ R7, LAP1 ; giảm r7, lặp lại 8 lần MOV R7, #8 MOV A, #00H SETB ACC.7 ; xác lập Acc.7 MOV R5, A LAP2: NOP @WRITEP 1 ; ghi ra cổng P1 CALL DELAY ; gọi trễ MOV B, A MOV A, R5 RR A ; quay a sang phải MOV R5, A ADD A, B DJNZ R7, LAP2 </pre>	<pre> MOV R7, #8 LAP3: NOP @WRITEP 2 ; ghi ra cổng P2 CALL DELAY ; gọi trễ MOV R5, A MOV A, B RL A ; quay s sang trái MOV B, A MOV A, R5 SUBB A, B ; trừ a với b DJNZ R7, LAP3 ; lặp lại 8 lần từ b MOV R7, #8 MOV A, #0FFH LAP4: NOP @WRITEP 3 CALL DELAY MOV R5, A MOV A, B RR A ; quay a sang phải MOV B, A MOV A, R5 SUBB A, B DJNZ R7, LAP4 @WRITEPS 00H, 3 ; ghi 00h vào cổng P3 CALL DELAY SJMP LAP0 DELAY: MOV R0, #2 ; tính trễ DEL1: MOV R1, #250 DEL2: MOV R2, #250 DEL3: DJNZ R2, DEL3 DJNZ R1, DEL2 DJNZ R0, DEL1 RET END </pre>
---	---

Ví dụ 7-3 giới thiệu chương trình điều khiển đèn hiển thị LED đơn bằng phương pháp đọc bảng dữ liệu trong bộ nhớ.

Chương trình cho :

- Cổng P0 sáng một đèn, chạy từ bên phải sang bên trái.
- Cổng P1 sáng một đèn, chạy từ bên trái sang bên phải.
- Cổng P2 sáng dần nhiều đèn (kiểu dồn) từ bên phải sang bên trái.
- Cổng P3 sáng dần nhiều đèn (kiểu dồn) từ bên trái sang bên phải.

Với kiểu sáng theo ma trận này, chúng ta có thể biểu diễn các ký tự hay các hình vẽ bất kỳ. Vì chỉ có 4 cổng ra cho $4 \times 8 = 32$ điểm nên ký tự hoặc hình vẽ không đẹp và khó biểu diễn. Với các đèn LED 7 đoạn hay ma trận diot, ma trận LED, ta có thể biểu diễn được các ký tự và hình vẽ bất kỳ. Đây là nguyên tắc tạo ký tự và hình vẽ của vi xử lý và vi điều khiển.

Ví dụ 7-3: Hiện thị theo bảng dữ liệu

MACRO @DOCBG ; lệnh gộp LAP: MOV DPTR, #BANG1 ; nạp địa chỉ bảng1 LB1: CLR A ; xoá thanh ghi A MOVC A, @A+DPTR ; đọc bảng vào A MOV P0, A ; đưa A ra cổng P0 LCALL DELAY ; gọi trễ INC DPTR ; tăng địa chỉ bảng CJNE A, #0FFH, LB1 ; so với 0FFh, về lb1 MOV DPTR, #BANG2; nạp địa chỉ bảng2 LB2: CLR A ; xoá thanh ghi A MOVC A, @A+DPTR ; đọc bảng vào A MOV P1, A ; đưa A ra cổng P1 LCALL DELAY ; gọi trễ INC DPTR ; tăng địa chỉ bảng CJNE A, #0FFH, LB2 ; so với 0FFh, về lb2 MOV DPTR, #BANG3 ; nạp địa chỉ bảng3 LB3: CLR A ; xoá thanh ghi A MOVC A, @A+DPTR; đọc bảng vào A MOV P2, A ; đưa A ra cổng P2 LCALL DELAY ; gọi trễ INC DPTR ; tăng địa chỉ bảng CJNE A, #0FFH, LB3 ; so với 0FFh, về lb3 MOV DPTR, #BANG4 ; nạp địa chỉ bảng4 LB4: CLR A ; xoá thanh ghi A MOVC A, @A+DPTR; đọc bảng vào A	MOV P3,A ; đưa A ra cổng P2 LCALL DELAY ; gọi trễ INC DPTR ; tăng địa chỉ bảng CJNE A, #0FFH, LB4 ; so với 0FFh, về lb1 LJMP LAP ; về lap ORG 200H ; địa chỉ bảng BANG1: DB 00H,01H, 02H, 04H, 08H,10H, 20H, DB 40H, 80H,0FFH BANG2: DB 80H, 40H, 20H,10H, 08H, 04H, 02H, DB 01H, 00H, 0FFH BANG3: DB 01H, 03H, 07H, 0FH, 1FH,3FH,7FH, 0FFH BANG4: DB 80H, 0C0H, 0E0H, 0F0H, 0F8H, 0FCH, DB 0FEH, 0FFH ENDMAC ; kết thúc lệnh gộp ORG 00H ; chương trình chính LAP0: NOP @DOCBG ; đọc bảng SJMP \$; chờ DELAY: MOV R0, #2 ; chương trình trễ DEL1: MOV R1, #250 DEL2: MOV R2, #250 DEL3: DJNZ R2, DEL3 DJNZ R1, DEL2 DJNZ R0, DEL1 RET ; trở về từ chương trình con END ; kết thúc dịch
---	---

7.4.4. Điều khiển đèn giao thông

Ở đường ngã tư, ngã năm v.v... trong thành phố, có các đèn giao thông dùng để điều khiển xe cộ qua lại theo các trục đường giao nhau. Ví dụ, tại một đường ngã tư, có hai trục đường giao nhau, đèn giao thông phải sáng, tắt theo quy luật sau:

- Đèn xanh được đi qua, đèn vàng chuẩn bị và đèn đỏ dừng lại.
- Hai đèn vàng cùng sáng và cùng tắt trong thời gian ngắn (cỡ 5-10 giây), giữa hai đèn xanh và đỏ sáng để các xe chuẩn bị dừng hay chuẩn bị đi qua ngã tư.
- Hai đèn xanh và đỏ ở hai chiều đường A hay B phải sáng, tắt ngược nhau, để xe chỉ đi được một đường, đường kia phải dừng lại và kéo dài trong thời gian lớn hơn (cỡ 1 phút).

Một số ngã tư, còn trang bị thêm đồng hồ đếm giây (gắn đèn chỉ thị ở cột cho mỗi ngã đường) và chỉ thị số đếm lùi từ 5 phút tới 0 để người tham gia giao thông biết thời gian chờ sắp hết và chuẩn bị đi qua ngã tư đường (xem phần đếm ở chương 8).

Để điều khiển tổ hợp 6 đèn (xanh, đỏ, vàng cho hai trục đường) trên, ta lập bảng 7-3.

Bảng 7-3. Trạng thái đèn giao thông ở đường ngã tư

Xanh	Vàng	Đỏ	Đỏ	Vàng	Xanh	
1	0	0	1	0	0	10000100B = 84H
0	1	0	1	0	0	01000100B = 44H
0	0	1	0	0	1	00100001B = 21H
0	0	1	0	1	0	00100010B = 22H
D7	D6	D5	D2	D1	D0	các bit

Các đèn trên được điều khiển bởi một VĐK, mỗi đèn nối với một bit của một cổng song song nào đó, qua một mạch khuếch đại dòng công suất.

Có thể gắn song song hai đèn cho hai bên đường của mỗi ngã đường. Để tiết kiệm đèn, ta có thể chỉ cần một đèn màu vàng chung cho hai ngã đường vì chúng sáng và tắt đồng thời cho hai ngã (chuẩn bị dừng lại và chuẩn bị đi qua đường). Điều khiển sáng hay tắt các đèn theo dữ liệu trên bảng 7-3, phải tránh các đèn xanh hay đỏ sáng hay tắt đồng thời. Lệnh gộp và chương trình điều khiển đèn như ví dụ 7-3. Với các tham số 1, ta chọn cổng tùy ý và tham số 3, ta chọn thời gian mong muốn. Có thể nối đèn màu khác nhau với bit tùy chọn của cổng, ta phải thay đổi giá trị đưa ra cổng để điều khiển đèn (xem chi tiết ở tập II).

CÂU HỎI VÀ THỰC HÀNH

I. Câu hỏi

1. Nêu đặc điểm của các cổng song song của VDK.
2. Hãy so sánh đặc điểm của cổng song song P0 với các cổng P1 của VDK.
3. Hãy so sánh đặc điểm của cổng song song P2 với các cổng P1 của VDK.
4. Hãy so sánh đặc điểm của cổng song song P3 với các cổng khác của VDK.
5. Hãy so sánh sự trao đổi tin ra và vào của các cổng P0, P1, P2, P3 của VDK.
6. Kể tên các lệnh đưa tin ra và vào qua các cổng song song theo byte.
7. Kể tên các lệnh xử lý bit qua các cổng song song và so sánh với nhóm lệnh logic cho byte nhớ.
8. Hãy so sánh các lệnh nhảy theo giá trị của bit với các lệnh nhảy theo giá trị của thanh ghi A.
9. Hãy so sánh các ứng dụng của việc đưa tin ra cổng song song trong hiển thị, phát xung và điều khiển của VDK.
10. Các ứng dụng của việc đọc tin vào và xử lý tin theo các bit.
11. Nêu ứng dụng của sự điều khiển theo bit của VDK.
12. Nêu ứng dụng của điều khiển theo byte của VDK.

II. Thực hành

1. Quan sát các cổng vào ra song song với chương trình mô phỏng Pinnacle.
2. Chạy các chương trình đưa tin ra hiển thị trên các LED rồi qua các ví dụ trong chương.
3. Chạy các chương trình đưa tin ra điều khiển động cơ bước ở các ví dụ trong chương.
4. Chạy các chương trình đưa tin ra điều khiển máy in trong chương.
5. Chạy các chương trình phát tín hiệu vuông góc, chữ nhật và nhiều xung trễ liên tiếp trong chương.

III. Thực hành lập trình nâng cao

1. Viết chương trình đưa tin ra các đèn LED đơn theo phương thức sau: sáng lần lượt các đèn từ trái sang phải hay từ phải sang trái.
2. Viết chương trình đưa tin ra đèn LED đơn theo phương thức sau: sáng một đèn chạy từ trái sang phải hay từ phải sang trái.
3. Viết chương trình đưa tin ra các đèn LED đơn theo phương thức sau: sáng dồn từ đèn ở giữa chạy ra hai đầu.
4. Viết chương trình đưa tin ra các đèn LED đơn theo phương thức sau: sáng dồn từ đèn ở giữa chạy ra hai đầu.
5. Viết chương trình đưa tin ra các đèn LED đơn theo phương thức sau: sáng dồn từ các đèn ở hai đầu vào giữa.

Chương 8

BỘ ĐỊNH THỜI/BỘ ĐẾM (Timer/Counter)

Mục đích của chương:

- Hiểu rõ cấu tạo và lập trình cho hoạt động của Timer ở 4 chế độ khác nhau.
- Hiểu rõ cấu tạo và lập trình cho hoạt động của Counter ở 4 chế độ khác nhau.
- Hiểu rõ các ứng dụng của Timer và Counter ở chế độ không có ngắt chương trình.

8.1. TIMER/COUNTER0,1 CỦA 8051

8.1.1. Timer0,1

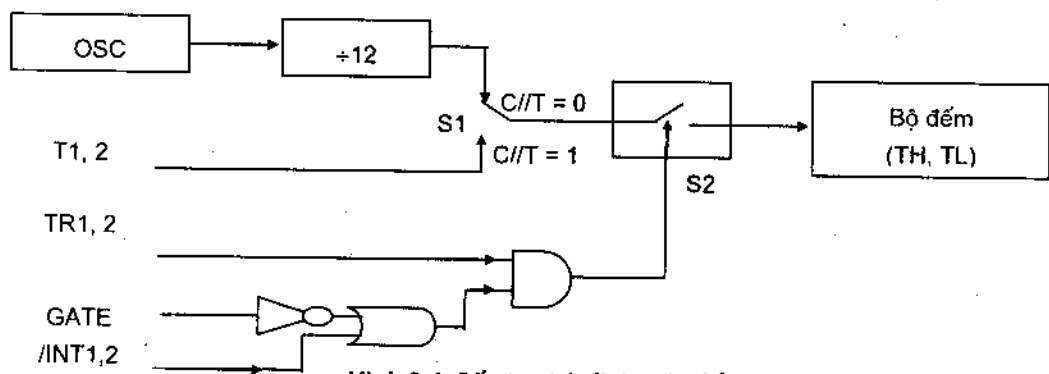
1. Cấu tạo và hoạt động của Timer0,1

Cũng như vi mạch 8053 của hệ vi xử lý hay máy vi tính, trong vi mạch VDK có một bộ đếm xung thời gian bên trong (định thời -Timer) và đếm xung ngoài (Counter). Đó là một bộ đếm 16 bit, có 4 chế độ hoạt động (do thanh ghi TMOD điều khiển), có các cờ báo đếm tràn (từ giá trị FFFFH về 0 ở thanh ghi TCON).

Bộ đếm trên có thể hoạt động ở chế độ định thời (Timer) hay ở chế độ đếm (Counter).

Bộ đếm trên, ở chế độ hoạt động định thời (Timer) dùng để đếm chuỗi xung nhịp của vi điều khiển với tần số chia cho 12. Ở chế độ đếm xung ngoài (Counter), bộ đếm trên đếm số xung ngoài vào các lối vào T0 (bit P3.4- chân số 14), T1 (bit P3.5- chân 15), T2 (bit P1.0- chân 1) của 8052 (hình 8-1).

a) Cấu trúc chung



Hình 8-1. Cấu tạo bộ định thời/đếm

Các VDK 8051 và 8052 đều có hai bộ Timer/Counter0,1. Mỗi bộ có cấu trúc như hình 8-1' gồm nguồn xung đếm, hai bộ đếm 8 bit TH, TL và cơ cấu điều khiển gồm các thanh ghi chế độ TMOD và điều khiển/trạng thái TCON. Riêng 8052 có thêm bộ Timer/Counter2 (xét ở dưới).

Với thực đơn VIEWTIMER, REGISTER của Pinnacle, ta có thể quan sát được các thanh ghi của Timer/Counter.

b) Thanh ghi đếm

Mỗi một Timer/Counter có ba thanh ghi: thanh ghi đếm dữ liệu và đếm TL (8 bit), TH (8 bit); thanh ghi chế độ TMOD (8 bit) và thanh ghi điều khiển TCON (bảng 8-1).

Tùy chế độ khác nhau, ta sẽ dùng bộ đếm 16 bit của các thanh ghi TH, TL này tạo nên các bộ đếm dung lượng khác nhau.

Bảng 8-1. Các thanh ghi của Timer/Counter

Thanh ghi (SFR)	Tên gọi	Địa chỉ	Địa chỉ bit
TCON	Điều khiển	88H	Có
TMOD	Chế độ	89H	Không
TL0	Byte thấp của Timer/Counter0	8AH	Không
TH0	Byte cao của Timer/Counter0	8BH	Không
TL1	Byte thấp của Timer/Counter1	8CH	Không
TH1	Byte cao của Timer/Counter1	8DH	Không

c) Thanh ghi điều khiển-trạng thái TCON

Thanh ghi này có các bit như hình 8-2 và đặc tính của các bit như ở bảng 8-2.

Thanh ghi này là thanh ghi xử lý bit, có thể xóa (bởi lệnh CLR) hay xác lập (bởi lệnh SETB) từng bit.

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Hình 8-2. Các bit của thanh ghi TCON của Timer/Counter

Như vậy, 4 bit cao (TF1, TR1, TF0, TR0) dành để điều khiển Timer/Counter hoạt động ở chế độ không có ngắt, còn 4 bit thấp dành cho hoạt động khi có ngắt ngoài với sườn xung xuống (xét chi tiết ở chương 10).

Bảng 8-2. Các bit của thanh ghi điều khiển-trạng thái TCON

Ký hiệu	Vị trí	Ý nghĩa	Ký hiệu	Vị trí	Ý nghĩa
TF1	TCON.7	Bit trạng thái cờ đếm tràn của Timer1. Xác lập bởi phần cứng khi đếm tràn về 0000H. Xóa bởi lệnh hay xử lý ngắt chương trình.	IE1	TCON.3.	Bit trạng thái cờ ngắt sườn xung 1. Xác lập bởi phần cứng khi phát hiện ngắt ngoài INT1. Xóa khi xử lý ngắt.
TR1	TCON.6	Bit điều khiển Timer1. Xác lập/xóa bởi lệnh để khởi động hay dừng đếm cho Timer1.	IT1	TCON.2	Bit điều khiển Ngắt ngoài 1. Xác lập/xóa bởi lệnh để chỉ sườn xuống/mức thấp của ngắt ngoài.
TF0	TCON.5	Bit trạng thái cờ đếm tràn của Timer0. Xác lập bởi phần cứng khi đếm tràn về 0000H. Xóa bởi lệnh hay xử lý ngắt chương trình.	IE0	TCON.1	Bit trạng thái cờ ngắt sườn xung 0. Xác lập bởi phần cứng khi phát hiện ngắt ngoài INT0. Xóa khi xử lý ngắt.
TR0	TCON.4	Bit điều khiển Timer0. Xác lập/xóa bởi lệnh để khởi động hay dừng đếm cho Timer0.	IT0	TCON.0	Bit điều khiển ngắt ngoài 0. Xác lập/xóa bởi lệnh để chỉ sườn xuống/mức thấp của ngắt ngoài.

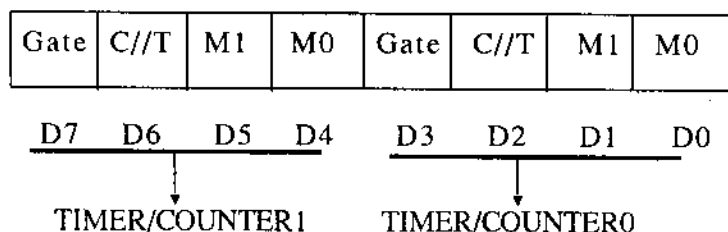
Các bit TR1, TR0 dùng để điều khiển bộ đếm của Timer bắt đầu đếm (TR = 1) hay dừng đếm (TR = 0). Sự xác lập hay xoá các bit này bởi lệnh SETB TR hay CLR TR của chương trình.

Các bit trạng thái TF được xác lập bởi mạch (phần cứng) khi bộ đếm tràn từ FFFFH về 0000H và được xoá về 0 khi chương trình nhảy tới chương trình phục vụ ngắt (xem chương 10) hoặc bởi lệnh CLR TF trong chương trình không có nhảy tới chương trình phục vụ ngắt.

Bốn bit thấp IE1, IT1, IE0, IT0 được dùng ở chế độ ngắt chương trình (xem chương 10).

d) Thanh ghi chế độ TMOD

Cả hai bộ Timer/Counter0,1 của VDK họ MCS-51 chỉ có một thanh ghi điều khiển chế độ TMOD như hình 8-3 và các bit như bảng 8-3.



Hình 8-3. Các bit của thanh ghi TMOD của Timer/Counter

Từ hình 8-3 ta thấy:

- Thanh ghi TMOD có hai nửa 4 bit dành cho hai Timer1 (4 bit cao) và Timer0 (4 bit thấp).
- Bit GATE dùng để ghi điều khiển bộ định thời/dếm.
- + GATE = 0 đưa xung của máy phát xung nhịp với tần số bằng tần số xung nhịp /12 vào đếm cho bộ định thời.
- + GATE = 1 để đưa tín hiệu ngoài /T0,1 vào đếm cho bộ định thời.
- Hai bit M1, M0 xác định chế độ cho bộ định thời như bảng 8-4.

Bảng 8-3. Các bit của thanh ghi TMOD

Bit	Tên	Timer	Mô tả
7	GATE	1	Khi Gate = 1, Timer chỉ làm việc khi INT0
6	C//T	1	Bit điều khiển đếm xung nhịp (= 0) cho Timer hay đếm xung ngoài (= 1) cho Counter.
5	M1	1	Bit chọn chế độ của Timer1
4	M0	1	Bit chọn chế độ của Timer1
3	GATE	0	Khi Gate = 1, Timer chỉ làm việc khi INT1
2	C//T	0	Bit điều khiển đếm xung nhịp (= 0) cho Timer hay đếm xung ngoài (= 1) cho Counter.
1	M1	0	Bit chọn chế độ của Timer0
0	M0	0	Bit chọn chế độ của Timer0

Bảng 8-4. Các chế độ của thanh ghi TMOD

M1	M0	Chế độ	Đặc điểm
0	0	0	Hai thanh ghi TH, TL là bộ đếm 13 bit
0	1	1	Hai thanh ghi TH, TL là bộ đếm 16 bit
1	0	2	TL là hai bộ đếm 8 bit còn thanh ghi TH lưu giá trị nạp lại cho TL
1	1	3	Thanh ghi TH, TL là hai bộ đếm độc lập 8 bit

2. Các chế độ

Chọn chế độ hoạt động hay cấu hình của bộ đếm, tức chọn các giá trị các bit M1, M0 của thanh ghi TMOD. Có 4 chế độ để lựa chọn là:

+ **Chế độ 0:** bộ đếm 13 bit (5 bit cao của TL + 8 bit TH) (hình 8-4a).

+ **Chế độ 1:** bộ đếm 16 bit (8 bit TL + 8 bit TH). Ở chế độ 1 này, dung lượng đếm là tối đa (tới 65535).

+ **Chế độ 2:** bộ đếm 8 bit trên TL hay chế độ tự nạp lại giá trị của TH vào TL khi đếm tràn (hình 8-4c).

+ **Chế độ 3:** chỉ cho TL0 và TH0 (hình 8-4d), có hai bộ đếm 8 bit TL0, TH0 riêng rẽ là:

Bộ đếm TL0: giống bộ đếm ở chế độ 2 nhưng không nạp lại giá trị TH0 và có cờ tràn TF0.

Bộ đếm TH0: bộ đếm 8 bit trên TH0 nhưng chỉ lấy xung vào đếm từ bộ dao động thạch anh với tần số chia cho 12 và xác lập cờ tràn TF1 khi đếm tràn.

Từ các hình 8-4 và các bảng 8-3 và 8-4, ta có những nhận xét chung sau:

Có 4 chế độ đếm (do giá trị của M1M0 quyết định) hay 4 chế độ sử dụng hai thanh ghi đếm 8 bit TH và TL như sau:

+ Trừ chế độ 2 cho bộ đếm TH0 chỉ nhận xung đếm từ bộ dao động thạch anh bên trong, còn các chế độ khác đều có hai nguồn xung là bộ dao động thạch anh (ở chế độ Timer-định thời) hay xung ngoài vào (ở chế độ Counter-đếm).

+ Các bộ đếm sẽ đếm từ giá trị ban đầu (do ghi vào bằng lệnh của chương trình) tới giá trị tràn (đếm tới cực đại và trở về 00H) và xác lập cờ tràn TF = 1. Ở chế độ ngắt chương trình, sự xác lập cờ tràn này sẽ gây ra ngắt chương trình (xét ở chương 10).

Từ thanh ghi TMOD, ta có các giá trị TMOD cho các bộ định thời (Timer) và đếm (Counter) cho các chế độ khác nhau như bảng 8-5.

Bảng 8-5. Lỗi TMOD của Timer0,1/Counter0,1

Chế độ	Timer0	Timer1	Counter0	Counter1	Ghi chú
0	00h	00h	0Ch	0C0h	đếm 8+5 = 13 bit
1	01h	10h	0Dh	0D0h	đếm 16 bit
2	02h	20h	0Eh	0E0h	đếm 8 bit lặp lại
3	03h	30h	0Fh	0F0h	đếm 8 và 16 bit

Chú ý: - Tuy chỉ dùng một Timer hay Counter nhưng lời chế độ TMOD này luôn ghi chung cho cả hai Timer hay Counter vì luôn có số 0 cho Timer hay Counter còn lại không sử dụng, nhưng chỉ có một Timer hay Counter hoạt động vì chỉ ghi có một TRi vào thanh ghi điều khiển TCON.

- Ở chế độ 3, theo chương trình mô phỏng Pinnacle, chỉ có Timer0 hoạt động, còn Timer1 không hoạt động vì ở chế độ 3 của Timer0 này, thanh ghi TH1, TL1 là bộ đếm độc lập thứ hai (không phải TH0 như trong hình vẽ của các tài liệu).

- Chúng ta sẽ sử dụng bảng này để gán tham số cho các lệnh gộp @timer chung cho cả Timer và Counter khi lập trình với lệnh gộp cho Timer0,1/Counter0,1.

3. Nguồn xung vào đếm

Nguồn tín hiệu đưa vào đếm do đóng chuyển mạch S1 (hình 8-1). Do đó, Bộ định thời/đếm có thể chọn một trong ba nguồn xung vào:

- **Tín hiệu của bộ dao động thạch anh OSC:** (tần số 11,9MHz hay 12 MHz) chia cho 12. Như vậy, chu kỳ của xung đếm cỡ 1 micro giây. Khi dùng nguồn xung đếm này, Timer/Counter hoạt động ở chế độ định khoảng thời gian hay định thời.

- **Tín hiệu ngoài vào đếm:** đưa vào các lối vào T0 (hay P3.4, chân số 14) hay T1 (hay P3.5, chân số 15). Khi dùng nguồn xung đếm này, Timer/Counter hoạt động ở chế độ đếm xung ngoài.

Việc chọn một trong hai nguồn xung đếm hay chọn chế độ định thời hay đếm xung ngoài do chuyển mạch S1 điều khiển. Chuyển mạch này được điều khiển bởi giá trị của bit C/T.

+ Nếu bit C/T = 0, chuyển mạch ở vị trí trên hay đếm xung của bộ dao động, khi đó Timer/Counter ở chế độ định thời (T = 0).

+ Nếu bit C/IT = 1, chuyển mạch ở vị trí dưới hay ở chế độ đếm xung ngoài (C = 1).

- Tín hiệu ngoài vào kết thúc đo khoảng thời gian: ở chế độ định thời (IT = 0), nếu cho tín hiệu /INT0 (P3.2 chân 12) hay /INT1 (P3.3 chân 13) thì bộ định thời sẽ ngừng đếm và các thanh ghi đếm dữ liệu TH, TL sẽ ghi số đếm của thời khoảng giữa thời điểm tác động tín hiệu TR tới thời khoảng tác động /INT0,1.

4. Khởi động/dừng đếm

Chuyển mạch S2 điều khiển bắt đầu đếm hay dừng đếm. Việc đóng (bắt đầu đếm) hay mở (dừng đếm) do các tín hiệu điều khiển TR0,1, GATE và /INT0,1 (chân P3.2) quyết định.

a) Đóng mở chuyển mạch S2 do TR0,1

Ở chế độ mặc định (sau khi bật nguồn nuôi hay ấn nút REST), các thanh ghi được xóa về 0, do đó các bit GATE (bit D7, D3) của thanh ghi TMOD được xóa về 0, và lối vào /INT0,1 (các chân P3.2 (số 12) và P3.3 (số 13) có giá trị 1. Hàm số $GATE0,1 \text{ OR } /INT0,1 = 1$ cho ở lối ra của vi mạch OR thể thấp (L-LOW) có mức logic 1 (hình 8-1). Hàm ra của sơ đồ AND thể cao (hình 8-1) cho lối ra là 0 hay 1 khi TR0,1 là 0 hay 1. Nói khác đi, khi không có các tín hiệu GATE1,2 và /INT0,1, việc đóng mở chuyển mạch S2 chỉ do TR0,1 quyết định (TR0,1 = 1, bộ đếm bắt đầu đếm và TR0,1 = 0, bộ đếm dừng).

b) Đóng mở chuyển mạch S2 do GATE0,1 và /INT0,1

Nếu cho $GATE0,1 = 1$, lối vào vi mạch AND thể L (hình 8-1) sẽ có mức 0 và lối vào.

+ Khi $/INT0,1 = 0$: Hàm số $GATE0,1 \text{ AND } /INT0,1 = 0$ cho ở lối ra của vi mạch AND thể thấp (L-LOW) có mức logic 0. Hàm ra của sơ đồ AND thể cao cho lối ra luôn luôn là 0 dù TR0,1 là 0 hay 1. Nói khác đi, chuyển mạch S2 luôn luôn mở.

+ $/INT0,1 = 1$, lối vào vi mạch AND thể thấp L sẽ có mức 1 và lối ra của nó có mức H hay bằng 1 (hàm OR thể L). Do đó lối ra của hàm AND thể cao H luôn có mức logic 1 khi TR0,1 = 1. Nói khác đi, tín hiệu TR0,1 điều khiển chuyển mạch S2 đóng mạch, cho tín hiệu /INT0,1 = 1 đi qua. Sơ đồ đếm được điều khiển bởi cả hai tín hiệu TR0,1 và /INT0,1 hay hàm $TR0,1 \text{ AND } /INT0,1 = 1$. Người ta sử dụng trường hợp này để đo độ rộng xung vào dương tác động lên chân /INT0,1 (xét ở dưới).

+ Xóa bit TR0 để ngừng đếm và xóa cờ tràn TF0.

5. Hoạt động định thời của TIMER0,1

Như đã trình bày tổng quát ở trên, bây giờ chúng ta áp dụng cụ thể cho từng hành động định thời (đếm xung bên trong VĐK) hay đếm xung ngoài của bộ Timer/Counter với các chương trình cụ thể. Để khởi động bộ đếm TH, TL ta thực hiện các hành động theo trình tự sau:

- Ghi vào thanh ghi chế độ TMOD:

+ Ghi chế độ cho bộ đếm: chọn một trong 4 chế độ bằng cách ghi giá trị M (bằng 0, 1, 2, 3) vào hai bit M1M0 của thanh ghi TMOD.

+ Ghi nguồn xung vào đếm cho bộ đếm: ghi IT = 0 (GATE = 0 do mặc định) để chọn chế độ hoạt động định thời tức nguồn xung vào đếm từ bộ dao động thạch anh.

- Ghi vào thanh ghi điều khiển - trạng thái TCON:

+ Ghi lệnh bắt đầu đếm xung: bằng lệnh SETB TR0,1 sẽ cho TR0,1 = 1 (TR0 cho Timer0/Counter0, TR1 cho Timer1/Counter1).

+ Ghi lệnh ngừng đếm: bằng lệnh CLR TR0,1 sẽ cho TR = 0.

+ Đọc trạng thái kết thúc của bộ đếm: khi đếm tràn, cờ tràn TF (TF0, TF1) sẽ xác lập lên 1. Đọc cờ trạng thái này, ta biết được bộ đếm đã đếm từ giá trị TH, TL ghi vào ban đầu tới #0FFFFH rồi trở về #0000H.

Ví dụ 8-1 giới thiệu lệnh gộp @timer <1.chế độ?, 2.timer ?, 3.TH, 4.TL> và chương trình định thời cho TMER0 theo các bước trên của Timer.

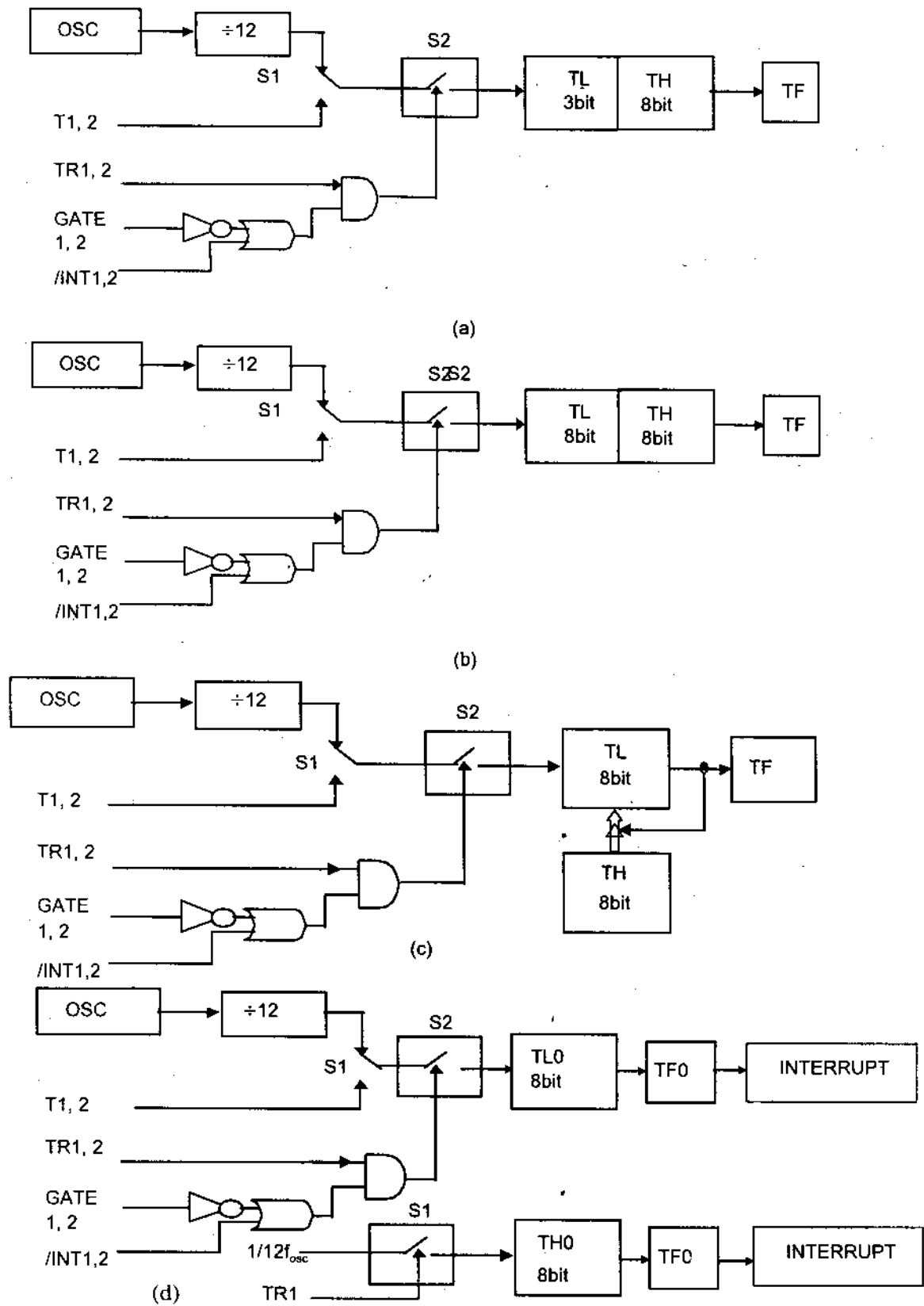
- Ghi chế độ vào thanh ghi TMOD cho TIMER0.

- Ghi các giá trị ban đầu cho bộ đếm TH0TL0 là #00H.

- Ghi lệnh bắt đầu đếm vào bit TR0 của thanh ghi TCON.

- Chờ đếm tràn.

- Xóa bit TR0 để ngừng đếm và xóa cờ tràn TF0.



Hình 8-4. Các chế độ của Timer: chế độ 0 (a), 1 (b), 2 (c) và 3 (d)

Hình 8-5 giới thiệu chương trình dịch của và chương trình của ví dụ 8-1.

Với chương trình ở ví dụ 8-1 trên, ta có thể:

- Hợp dịch và cho chạy (từng bước hay chương trình) trên chương trình mô phỏng Pinnacle trong máy tính PC để thấy:

+ *Chế độ 0*: cho bộ đếm 11 bit (8 bit của TH0, 5 bit thấp của TL0) như hình 8-4a.

+ *Chế độ 1*: cho bộ đếm 16 bit như hình 8-4b.

+ *Chế độ 2*: cho bộ đếm 8 bit có nạp lại TH0 cho TL0 như hình 8-4c.

+ *Chế độ 3*: cho bộ đếm 8 bit của Timer0, còn Timer1 đếm 16 bit hình 8-4d.

Ví dụ 8-2 chương trình định thời dùng lệnh gộp cho TMER1. Chương trình này có đặc điểm sau: - dùng lệnh gộp chung @timer <1.chd?, 2.timer?, 3. TH, 4.TL> ở ví dụ 8-1 (đã ghi vào thư viện MYLIB.ASM).

; Ví dụ 8-1: Lệnh gộp cho timer 0, 1	
macro @Timer <1.chd?, 2.timer?, 3. TH, 4.TL>	jc cd2 ; nhảy tới cd0 (chế độ 0)
mov tmod, #1	mov c, P0.3 ; nếu P0.0 = 1
mov thl2, #13 ; giá trị TH	jc cd3 ; nhảy tới cd0 (chế độ 0)
mov tll2, #14	ljmp menu ; trở về thực đơn
setb trl2	cd0: ; (chế độ 0)
back: jnb tfl2, back	@timer 00h, 0, 0FCh, 00h ; chế độ 0,
clr trl2	; đếm 13 bit
clr tfl2	ljmp menu ; trở về thực đơn
endmac	cd1: ; (chế độ 1)
org 00h ; chế độ 0, 1, 2, 3 cho timer0	@timer 01h, 0, 0ffh, 0f4h ; chế độ 1, 16 bit
mov P3, #00h	ljmp menu ; trở về thực đơn
mov P0, #00h	cd2: ; (chế độ 2)
menu: jnb P3.2, \$; thực đơn	@timer 02h, 0, 0f5h, 0f4h ; chế độ 2,
mov c, P0.0 ; nếu P0.0 = 1	; đếm TL 8 bit nạp lại
jc cd0 ; nhảy tới cd0 (chế độ 0)	ljmp menu ; trở về thực đơn
mov c, P0.1 ; nếu P0.0 = 1	cd3: ; (chế độ 3)
jc cd1 ; nhảy tới cd0 (chế độ 0)	@timer 03h, 0, 00h, 0f4h ; chế độ 3,
mov c, P0.2 ; nếu P0.0 = 1	; đếm 8 bit riêng rẽ, Timer1 đếm 16 bit
	ljmp menu ; t trở về thực đơn
	end ; kết thúc dịch

- Các chế độ 0, 1 và 2 cũng tương tự chương trình cho Timer0.

- Nhưng chế độ 3 không hoạt động.

Nếu hợp dịch và cho chạy trên chương trình mô phỏng Pinnacle, chúng ta sẽ thấy những kết quả đã trình bày ở trên.

6. Hoạt động đếm của Timer/Counter

Ngoài hoạt động định thời gian trễ, bộ định thời Timer0,1 của VDK 8051 còn được dùng làm bộ đếm các tín hiệu ngoài đưa vào từ các chân T0 (P3.4 chân số 14) và T1 (P3.5 chân số 15).

a) Xác định chế độ đếm

Ở chế độ đếm, phải ghi bit C/T (bit D6, D2) của thanh ghi TMOD phải có giá trị 1 (C = 1) để cho phép nhận xung ngoài vào từ T0 hay T1.

Bộ đếm này cũng được ghi chế độ vào thanh ghi TMOD, ghi các giá trị đếm ban đầu TH, TL và đếm tới 1FFFH (chế độ 0), FFFFH (chế độ 1) hay FFH (chế độ 2, 3) và sẽ tràn về 0, xác lập cờ tràn TF lên 1.

b) Chương trình cho bộ đếm

Ví dụ 8-3 giới thiệu chương trình đếm dùng bộ đếm COUNTER0 ở chế độ 1 và xung ngoài phải có sườn âm (từ mức logic 1 tới 0) đưa vào qua T0. Do đó, trước đó phải xóa lỗi vào T0 bằng lệnh.

CLR P3.4 vì mặc định, cổng P3 xác lập lên 1. Phải có lệnh JNB P3.4, \$ để chờ sườn xung âm ngoài vào đếm. Khi có xung từ nguồn ngoài vào hay kích cho P3.4 xuống 0, bộ đếm sẽ đếm lên 1. Bộ đếm này cũng kích thích bằng tín hiệu TR0,1 = 1 vì mặc định, GATE01 = 0 và /INT0 (P3.2) = 1.

Code Window (Disassembly)

Addr	Opcodes	ASC	Label	Disassembly	Comments
0000	75 80 00	u'I		MOV P3,#00h	
0003	75 80 00	u'II	(EXT0 INT)	MOV P0,#00h	
0006	30 B2 FD	0'y	menu	JNB P3.2,menu	
0009	A2 80	A'I		MOV C,P0.0	
000B	40 0F	@I	(T0 INT)	JC cd0	
000D	A2 81	A'I		MOV C,P0.1	
000F	40 20	@		JC cd1	
0011	A2 82	A'I		MOV C,P0.2	
0013	40 31	@I	(EXT1 INT)	JC cd2	
0015	A2 83	A'I		MOV C,P0.3	
0017	40 42	@B		JC cd3	
0019	02 00 06	III		LJMP menu	
001C	75 89 00	u'II	cd0	MOV TMOD,#00h	
001F	75 8C FC	u'ly		MOV TH0,#0FCh	giá trị TH
0022	75 8A 00	u'II		MOV TL0,#00h	
0025	D2 8C	éI		SETB TR0	
0027	30 8D FD	0'ly	back?1	JNB TF0,back?1	
002A	C2 8C	A'I		CLR TR0	
002C	C2 8D	A'I		CLR TF0	
002E	02 00 06	III		LJMP menu	
0031	75 89 01	u'II	cd1	MOV TMOD,#01h	

Addr	Opcodes	ASC	Label	Disassembly	Comments
0037	75 8A F4	u'ly		MOV TL0,#0F4h	
003A	D2 8C	éI		SETB TR0	
003C	30 8D FD	0'ly	back?2	JNB TF0,back?2	
003F	C2 8C	A'I		CLR TR0	
0041	C2 8D	A'I		CLR TF0	
0043	02 00 06	III		LJMP menu	
0046	75 89 20	u'I	cd2	MOV TMOD,#20h	
0049	75 8D F5	u'ly		MOV TH1,#0F5h	giá trị TH
004C	75 8B F4	u'ly		MOV TL1,#0F4h	
004F	D2 8E	éI		SETB TR1	
0051	30 8F FD	0'ly	back?3	JNB TF1,back?3	
0054	C2 8E	A'I		CLR TR1	
0056	C2 8F	A'I		CLR TF1	
0058	02 00 06	III		LJMP menu	
005B	75 89 03	u'II	cd3	MOV TMOD,#03h	
005E	75 8C 00	u'II		MOV TH0,#00h	giá trị TH
0061	75 8A F4	u'ly		MOV TL0,#0F4h	
0064	D2 8C	éI		SETB TR0	
0066	30 8D FD	0'ly	back?4	JNB TF0,back?4	
0069	C2 8C	A'I		CLR TR0	
006B	C2 8D	A'I		CLR TF0	
006D	02 00 06	III		LJMP menu	
0070	00	I		NOP	

Hình 8-5. Chương trình dịch của chương trình ở ví dụ 8-1

; Ví dụ 8-2: Chương trình cho Timer1

```

org 00h ; chế độ 0, 1, 2, 3 cho timer1
$INCLUDE MYLIB.ASM
mov P3, #00h
mov P0, #00h
cd0: ; (chế độ 0)
@timer 00h, 0, 0FCh, 00h ; chế độ 0,
; đếm 13 bit
ljmp menu ; trở về thực đơn
cd1: ; (chế độ 1)
menu:jnb P3.2, $ ; thực đơn
mov c, P0.0 ; nếu P0.0 = 1
jc cd0 ; nhảy tới cd0 (chế độ 0)
mov c, P0.1 ; nếu P0.0 = 1
jc cd1 ; nhảy tới cd0 (chế độ 0)
mov c, P0.2 ; nếu P0.0 = 1

```

```

jc cd2 ; nhảy tới cd0 (chế độ 0)
mov c, P0.3 ; nếu P0.0 = 1
jc cd3 ; nhảy tới cd0 (chế độ 0)
ljmp menu ; trở về thực đơn
@timer 10h, 0, 0FFh, 0F4h; chế độ 1, 16 bit
ljmp menu ; trở về thực đơn
cd2: ; (chế độ 2)
@timer 20h, 0, 0F5h, 0F4h ; chế độ 2,
; đếm TL 8 bit nạp lại
ljmp menu ; trở về thực đơn
cd3: ; (chế độ 3)
@timer 30h, 0, 00h, 0F4h ; chế độ 3,
; đếm 8 bit riêng rẽ, Timer1 đếm 16 bit
ljmp menu ; trở về thực đơn
end ; kết thúc dịch

```

Ví dụ 8-3 sẽ đếm từ giá trị 50D lên 256D rồi xoá về 0 để xác lập cờ tràn TF0. Nếu chạy chương trình đếm 8-3 trên bảng chương trình Pinnacle, ta phải tạo một xung âm giả bằng cách:

- Kích (bằng chuột) lối vào P3.4 (cho Timer0) hay P3.5 (cho Timer1) xuống 0 để đưa xung sườn âm vào đếm.

- Kích (bằng chuột) lối vào P3.4 (cho Timer0) hay P3.5 (cho Timer1) ở lần xung thứ hai lên 1 rồi xuống 0 để tạo một xung khác vào bộ đếm.

Với các chế độ đếm khác (0, 2, 3) ta cũng có chương trình tương tự như bộ định thời.

: Ví dụ 8-3: Lệnh gộp và chương trình đếm ở các chế độ	
<pre>macro @dem <1.chế độ?, 2.timer?, 3.TH?, 4.TL?> MOV TMOD, #1 ; Counter1 MOV TH12, #13 ; nạp vào TH12 MOV TL12, #14 ; vào TL12 SETB TR12 ; xác lập TR12 JNB TF12, \$; chờ cờ tràn TF12 = 1 CLR TF12 ; xoá cờ tràn TF12 CLR TR12 ; xóa TR12 endmacro org 00H mov P3, #0ffh mov P0, #00h menu: jb P3.2, \$ mov c, P0.0 jc cd0 mov c, P0.1 jc cd1 mov c, P0.2 jc cd2 mov c, P0.3 jc cd3 ljmp menu cd0: nop @dem 0ch, 0, 0fch, 0f0h; timer0, dem, M = 0 ljmp menu cd1: nop @dem 0dh, 0, 0fch, 0f0h; timer0, dem, M = 1 ljmp menu cd2: nop @dem 0e0h, 0, 0fah, 0f0h; timer0, dem, M = 2 ljmp menu cd3: nop @dem 0f0h, 0, 0fah, 0f0h; timer0, dem, M = 3 ljmp menu end ; kết thúc chương trình dịch</pre>	

8.2. TIMER2 CỦA VĐK 8052

Vì điều khiển 8052 ngoài hai Timer/Counter0,1 còn có thêm một Timer/Counter2. Cũng giống Timer/Counter0,1, nó có thể hoạt động hoặc như một Timer hoặc như một Counter bằng cách chọn bit C/T2 = 1 trong thanh ghi điều khiển/trạng thái T2CON (hình 8-6). Bảng 8-6 giới thiệu tên và công dụng của các bit của T2CON.

Timer2 có các chế độ hoạt động như bảng 8-8. Có ba chế độ hoạt động cho Timer2 là "bắt giữ" (hình 8-7), "tự nạp lại" (hình 8-8) và "máy phát tốc độ baud", được chọn bởi các bit của thanh ghi T2CON như hình 8-6 và bảng 8-6.

D7	D6	D5	D4	D3	D2	D1	D0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	C/RL2

Hình 8-6. Các bit của thanh ghi T2CON

Tuỳ tổ hợp các bit của T2CON, Timer2 có các chế độ hoạt động như bảng 8-7 và lời điều khiển như bảng 8-8.

Bảng 8-6. Các bit của thanh ghi T2CON

Bit	Ký hiệu	Vị trí	Ý nghĩa
7	TF2	T2CON.7	Cờ tràn Timer2, xác lập bởi đếm tràn Timer2 và bị xoá bởi lệnh. TF2 không xác lập bởi RCLK = 1 hay TCLK = 1
6	EXF2	T2CON.6	Cờ Time r2 ngoài, xác lập khi hoặc bị bắt giữ hay nạp lại gây ra bởi một dịch chuyển âm trên T2EX và EXEN2 = 1. Khi ngắt Timer2 cho phép, EXF2 = 1 sẽ gây ra cho CPU trở tới chương trình phục vụ ngắt Timer2. EXF2 bị xoá bởi lệnh.
5	RCLK	T2CON.5	Cờ nhịp nhận. Khi xác lập, gây ra cổng nối tiếp dừng các xung tràn Timer2 cho nhịp nhận tin nối tiếp ở chế độ 1 và 3. RCLK = 0 gây ra tràn Timer1 được dùng cho nhịp nhận tin nối tiếp.
4	TCLK	T2CON.4	Cờ nhịp truyền. Khi xác lập, gây ra cổng nối tiếp dừng các xung tràn Timer2 cho nhịp truyền tin nối tiếp ở chế độ 1 và 3. RCLK = 0 gây ra tràn Timer1 được dùng cho nhịp truyền tin nối tiếp.
3	EXEN2	T2CON.3	Cờ cho phép ngắt Timer2 ngoài. Khi xác lập, cho phép bắt hoặc nạp lại để tạo ra một dịch chuyển âm trên T2EX nếu Time r2 không được dùng để nhịp cho cổng nối tiếp. EXEN2 = 0 gây ra Time r2 không biết sự kiện ở T2EX.
2	TR2	T2CON.2	Điều khiển khởi động/dừng cho Timer2. Một logic 1 khởi phát Timer2.
1	C/IT2	T2CON.1	Chọn chế độ Timer (= 0) để đếm xung bên trong hay Counter (= 1) để đếm xung ngoài vào.
0	CP/RL2	T2CON.0	Cờ Capture (bắt)/Reload (nạp lại). Khi xác lập, việc bắt giữ sẽ xảy ra ở dịch chuyển âm ở T2EX nếu EXEN = 1. Khi bị xoá, việc nạp lại tự động sẽ xảy ra hoặc với tràn của Timer2 hoặc với dịch chuyển âm ở T2EX khi EXEN2 = 1. Khi RCLK = 1 hay TCLK = 1, bit này sẽ không được biết đến và Timer bị cưỡng bức tới tự động nạp lại khi Timer2 tràn.

Có ba chế độ hoạt động cho Timer2 (bảng 8-7) là:

- *Chế độ "bắt giữ"* (hình 8-7): dữ liệu của TH2, TL2 được bắt giữ (hay ghi nhớ) vào hai thanh ghi RCAPL2L, RCAP2H để đọc ra. Việc "bắt giữ" này xảy ra khi có dịch chuyển sườn xung âm của tín hiệu vào chân T2EX và đã ghi EXEN2 vào T2CON. Cờ TF2 xác lập khi đếm tràn và có xung T2EX vào khi EX2 = 1. Cờ EXF2 = 1 khi ghi EX2 = 1.

- *Chế độ "tự nạp lại"* (hình 8-8): dữ liệu từ các thanh ghi RCAP2L, RCAP2H được nạp lại vào TL2, TH2 khi có đếm tràn (giống chế độ 2 của Timer1 nhưng có 16 bit) hay có sườn xung âm vào lối vào T2EX và đã ghi EXEN2. Cờ TF2 xác lập khi đếm tràn hay khi có xung T2EX vào nếu đã ghi EX2 = 1. Cờ EXF2 = 1 khi ghi EX2 = 1.

- *Chế độ "máy phát tốc độ baud"* (xem chương 9): tạo ra các tín hiệu nhịp nhận Rx Clock và truyền TX CLOCK cho cổng nối tiếp từ các tín hiệu tràn của Timer1 hay Timer2 chia cho 16.

Để tiện sử dụng, người ta thành lập các bảng (bảng 8-7 và 8-8) cho các lời điều khiển (các bit của T2CON) ở các chế độ Timer2 và Counter2

Ghi chú: 1. Bắt giữ chỉ xảy ra khi có sườn xung âm ở lối vào T2EX.

2. Tự nạp chỉ xảy ra khi tràn Timer/Counter hay một dịch chuyển sườn xung âm (từ 1 về 0) trên chân T2EX (P1.1).

3. Timer2 thường được dùng cho :

- Bộ đếm hay định thời ở chế độ tự nạp lại với 16 bit (trong khi Timer0,1 chế độ 2 chỉ có 8 bit).
- Bộ đếm hay định thời ở chế độ bắt giữ để ghi TH2/TL2 ra RCAP2 để lưu trữ (dùng so sánh với số đếm đã ghi trước) và đọc ra. Ví dụ 8-4 giới thiệu chương trình định thời 1sec dùng Timer2.
- Cờ TF2 = 1 khi có đếm tràn 16 bit trên TH2, TL2.
- Máy phát tốc độ baud cho trao đổi tin qua cổng nối tiếp.

Bảng 8-7. Các chế độ hoạt động của Timer2

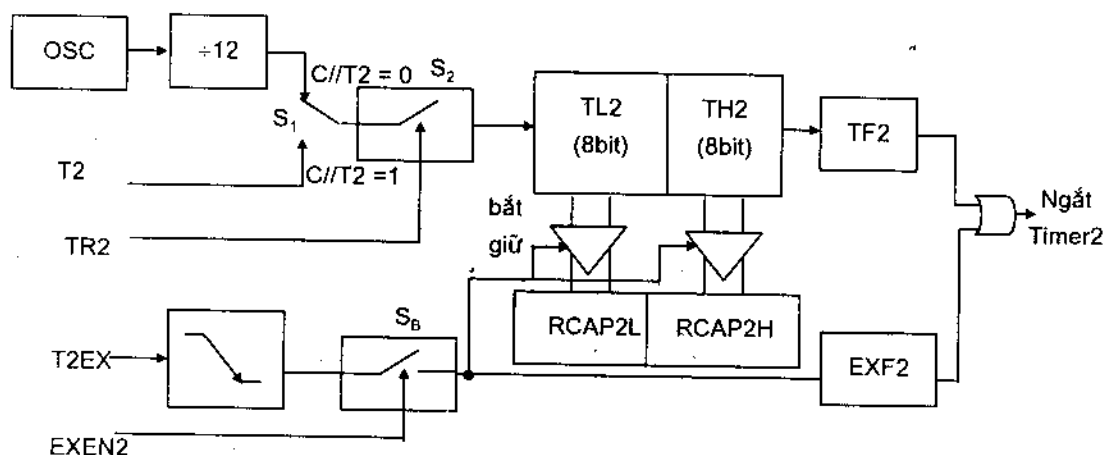
RCLK+TCLK	CP//RL2	TR2	Chế độ	Ghi chú
0	0	1	Tự nạp 16 bit	Nạp RCAP2 vào TH2TL2
0	1	1	Bắt giữ 16 bit	Ghi TH2TL2 ra RCAP2
1	x	1	Máy phát tốc độ baud	Dùng cho cổng nối tiếp
X	x	0	Tắt	Không dùng

Ví dụ 8-4 giới thiệu chương trình phát tốc độ baud dùng TIMER2.

Ví dụ 8-5 giới thiệu chương trình phát tốc độ baud cho cổng nối tiếp dùng lệnh gộp.

Bảng 8-8. Lệnh điều khiển chế độ Timer của Timer2

Chế độ	T2CON Điều khiển trong (1)	T2CON Điều khiển ngoài (2)
Tự nạp 16 bit	00H	08H
Bắt giữ 16 bit	01H	09H
Máy phát tốc độ nhận truyền cùng vận tốc	34H	36H
Máy phát tốc độ chỉ nhận	24H	26H
Máy phát tốc độ chỉ truyền	14H	16H



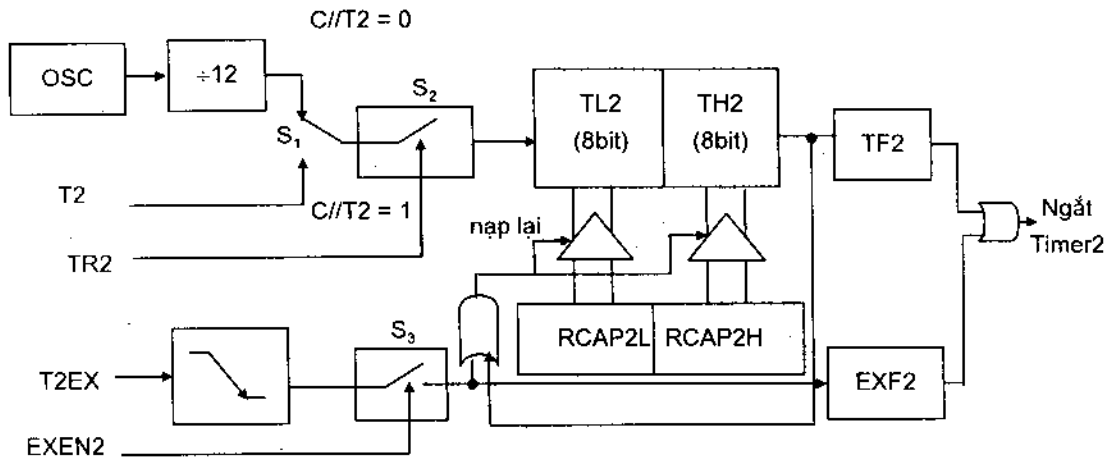
Hình 8-7. Chế độ bắt giữ của Timer2

8.3. ỨNG DỤNG TIMER/COUNTER TRONG ĐỊNH THỜI GIAN TRỄ

Bộ định thời gian trễ TIMER được dùng để tạo thời gian trễ mong muốn. Ta có thể dùng TIMER để:

- . Tạo thời gian trễ cho nhiều hành động liên tiếp của chương trình.
- . Tạo và khống chế thời gian cho một hành động khác: ví dụ đếm trong một khoảng thời gian cho phép (xem ở dưới).
- . Tạo độ kéo dài của một tín hiệu cho máy phát xung.

Sở với sự định trễ DELAY theo thời gian thực hiện lệnh (chương 3), việc định trễ do bộ định thời này có đặc điểm sau:



Hình 8-8. Chế độ tự nạp lại của Timer2

; Ví dụ 8-4: Chương trình phát tốc độ baud dùng TIMER2

```

ORG 00H                                ; bắt đầu chương trình từ địa chỉ 00H
LOOP: MOV RCAP2H, #0FFH                 ; tốc độ 9830 baud
      MOV RCAP2L, #-5                    ; thêm đủ cho 9600 baud
      MOV TH2, #0FFH                    ; nạp TH2 = #0FFH
      MOV TL2, #-5                       ; nạp TL2 = #-5
      MOV T2CON, #00110100B             ; bit RCLK = TCLK = TR2 = 1
      SJMP LOOP                          ; nhảy về LOOP
END                                      ; kết thúc chương trình dịch

```

; Ví dụ 8-5. TIMER2: Chương trình phát tốc độ cho nối tiếp dùng lệnh gộp

```

macro @Timer2                          ; dùng timer2, chế độ 2
    MOV T2CON, #00h
    MOV TH2, #1                          ; tham số #1 ghi dữ liệu vào thanh ghi TH2
    MOV TL2, #2                          ; tham số #2 ghi dữ liệu vào thanh ghi TL2
    MOV RCAP2H, #13                      ; tham số #3 ghi dữ liệu vào thanh ghi RCAP2H
    MOV RCAP2L, #14                      ; tham số #4 ghi dữ liệu vào thanh ghi RCAP2L
    SETB TR2                             ; bắt đầu đếm
endmacro

macro @TRA <1.CHE DO?>
    MOV SCON, #1                         ; chế độ nối tiếp
    MOV SBUF, A                          ; đưa A ra truyền
    JNB TI, $                            ; chờ cờ báo truyền
    CLR TI                                ; xoá TI
endmacro

org 00h
@baud2 high(-13), low(-13), high(-13), low(-13); tốc độ 9600 baud
sjmp $
end

```

- Không những dùng đơn vị thời gian nhỏ nhất là thời gian thực hiện lệnh một chu kỳ (cỡ 1 micro giây) mà còn dùng đơn vị thời gian của máy phát xung ngoài vào các lối vào T0,1 nên dài trẻ rộng hơn (nhiều giây tới nhiều giờ).

- Có thể thực hiện song song với những hành động khác của chương trình. Khi hết thời gian định trễ, có thể báo hiệu bằng cờ tràn và ngắt chương trình (ngắt do lệnh hay cơ chế ngắt).

1. Tính giá trị cần nạp vào TH, TL theo thời gian trễ

Bộ đếm của Timer sẽ đếm chuỗi xung nhịp với tần số nhỏ hơn 12 lần tần số xung nhịp.

Nếu tần số xung nhịp là 12 MHz thì tần số của Timer là $12 \text{ MHz} / 12 = 1 \text{ MHz}$ hay độ kéo dài của một xung đếm là 1 μsec . Nếu tần số xung nhịp là 11,0592 MHz thì tần số của Timer là $11.0592 / 12 \text{ MHz} = 0,9216 \text{ MHz}$ hay độ kéo dài của một xung là 1,085 μsec .

Nếu cho trước số đếm nạp vào TL, TH, ta có thể tính được số xung đếm tràn từ giá trị của THTL tới FFFFH+1 để có tín hiệu tràn TF.

Ví dụ, áp dụng cho Timer0 ở chế độ 1 (chế độ đếm 16 bit, số đếm ban đầu nạp vào cho các thanh ghi đếm TH0TL0 là FAF2H nên số đếm tới khi có cờ tràn TF0 là:

$$(FFFFH - FAF2H) + 1 = 50DH + 1 = 50EH = 256 \times 5 + 16 \times 0 + 14 = 1294D$$

Như vậy, phải sau 1294 xung nhịp vào đếm, thanh ghi đếm TH0, TL0 mới đếm tràn về 0 và cờ tràn TF0 mới xác lập lên 1 và quá trình định thời mới kết thúc. Lệnh JNB TF0, HERE hay JNB TF0, \$ sẽ chờ đếm tràn và khi đó mới thực hiện lệnh ngừng đếm CLR TR0 và xoá cờ tràn CLR TF0.

Chú ý: Chương trình trên có thể thay đổi thành chế độ 0, 2 và 3 thì bộ đếm thực hiện việc đếm khác và số xung như sau:

- Ở chế độ 0 (ghi 0H vào TMOD): bộ đếm TH0 chỉ có 5 bit, nên số đếm được dùng trong TH0 là 1AH và số đếm ban đầu là 1AF2H và số đếm tràn là $(1FFFH - 1AF2H) + 1 = 50DH + 1 = 50EH$.

Người ta ít dùng chế độ này vì dung lượng đếm nhỏ (có 11 bit) mà dùng chế độ 1 (16 bit).

- Ở chế độ 1 (ghi 1H vào TMOD): bộ đếm TH0 và TL0 có 8 bit, nên số đếm được dùng trong TH0 TL0 ban đầu là FAF2H và số đếm tràn là $(FFFFH - FAF2H) + 1 = 50DH + 1 = 50EH$.

- Ở chế độ 2 (ghi 2 vào TMOD): bộ đếm TH0 và TL0 có 8 bit, nhưng TH0 dùng để ghi số đếm được nạp lại cho TL0 nếu bộ đếm được tiếp tục đếm (ví dụ 8-4). Chương trình ở chế độ 2 khác chế độ 1 là: + Bộ đếm TL chỉ có 8 bit và được nạp giá trị của TH.

+ Luôn có lệnh lặp lại vì phát chuỗi xung liên tục.

- Ở chế độ 3 (ghi 3 vào TMOD): có hai bộ đếm phân biệt, bộ đếm TL0 với cờ tràn TF0 và TH0 với cờ tràn TF1 (xem hình 8-2 về các chế độ). Ở chế độ này, đã biến Timer0 thành hai Timer với hai bộ đếm 8 bit và dùng hai cờ tràn TF0 và TF1 của Timer1.

Nếu cho trước thời gian trễ dạng HEX, người ta có thể tính được số đếm ban đầu cần ghi vào các thanh ghi TH, TL ở các chế độ khác nhau theo thứ tự sau:

- Đổi thời gian trễ ra dạng nhị phân hay dạng HEX.

- Lấy hiệu $(FFFFH - \text{thời gian trễ}) + 1$ để ghi vào THTL.

; Ví dụ 8-6: Chương trình tạo thời gian trễ ở chế độ 2.

ORG 00H	; bắt đầu chương trình
MOV TMOD, #20H	; Timer1 ở chế độ 2 (đếm 8 bit)
MOV TH1, #0FDH	; nạp TH0 cho tốc độ baud 9,6 kHz
MOV TL1, TH1	; nạp TL1 = TH1 = #0FDH
LOOP: MOV P0, #0AAH	; đưa ra cổng P0 số AAH
SETB TR1	; bắt đầu đếm định thời gian trễ
HERE: JNB TF0, HERE	; chờ cờ tràn TF0 = 1
CLR TR1	; ngừng đếm
CLR TF1	; xoá cờ tràn
SJMP LOOP	; nhảy về LOOP để lặp lại
END	; kết thúc chương trình dịch

Nếu thời gian trễ cho trước dưới dạng thập phân, ví dụ, chữ số NNNNN trình tự tính như sau:

- Lấy hiệu số của (65536 - NNNNN).
- Đổi hiệu số thập phân trên thành dạng HEX.
- Ghi các số dạng HEX vào THTL.

Với chế độ 2, chỉ cần ghi một số HEX vào TH thay cho giá trị cần nạp vào dưới dạng HEX, ta có thể ghi trực tiếp số thời gian trễ với dấu âm (-) vào TH. Chương trình dịch sẽ biến đổi số âm thành số bù 2 của thời gian trễ dạng HEX như ví dụ 8-6, 8-7.

; Ví dụ 8-7: Chương trình tạo thời gian trễ đưa vào TH	
ORG 00H	; bắt đầu chương trình
MOV TMOD, 02H	; Timer0 ở chế độ 2 (đếm 8 bit)
LOOP: MOV TH0, # -150	; thời gian trễ 150 μ sec bù 2 là 6 AH
MOV TL0, TH0	; nạp TH0 vào TL0
SETB TR0	; bắt đầu đếm
DL1: JNB TF0, DL1	; chờ cờ tràn TF0 = 1 sau 5000 μ sec
CLR TF0	; xoá cờ tràn
SJMP LOOP	; trở về LOOP
END	

2. Hiệu chỉnh thời gian trễ do thực hiện lệnh của chương trình

Ngoài thời gian đếm trễ của các thanh ghi đếm THTL, các chương trình tạo trễ thời gian DELAY) và DELAY1 còn tốn thời gian thực hiện các lệnh của chương trình. Do đó, thời gian trễ chính xác phải kể thêm thời gian thực hiện các lệnh trên.

Ví dụ, chương trình DELAY0 có các thời gian trễ tính theo các chu kỳ máy thực hiện lệnh như hình 8-9.

Lệnh	Số chu kỳ máy
SETB P0.3	1
ACALL DELAY0	2
MOV TL0, #0F2H	2
MOV TH0, #0FAH	2
SETB TR0	1
L0: JNB TF0, L0	1
CLR TR0	1
CLR TF0	1
RET	1
	12

Hình 8-9. Hiệu chỉnh thời gian trễ

; Ví dụ 8-8: Chương trình tạo thời gian trễ 1 phút = 60 sec	
ORG 00H	; bắt đầu chương trình
MOV R0, #20	; nạp R0 = 20 lần
MOV R1, #60	; nạp R0 = 60 lần
MOV TMOD, #01H	; Timer0 ở chế độ 1 (đếm 16 bit)
DL: MOV DPTR, #15535	; cho đếm 5000 μ sec
MOV TL0, DPL	; nạp TL0 = #535D
MOV TH0, DPH	; nạp TH0 = # 15D
SETB TR0	; bắt đầu đếm định thời gian trễ
JNB TF0, \$	; chờ cờ tràn TF0 = 1 sau 5000 μ sec
CLR TF0	; xoá cờ tràn
DJNZ R0, DL	; nhảy về DL và lặp 20 lần cho 1 sec
DJNZ R1, DL	; nhảy về DL 60 lần cho đủ 60 sec
SJMP \$	; kết thúc chờ
END	; kết thúc chương trình dịch

Do đó thời gian bổ sung là $12 \times 2 \times 1,085 \mu\text{sec}$ và độ kéo dài xung do DELAY0 là $(5 \times 16 + 13 + 12) \times 1,085 \mu\text{sec} = 105 \times 1,085 \mu\text{sec} = 113,925 \text{ msec}$ (với tần số xung nhịp 11,0579 MHz).

Với thời gian nghỉ Delay1 của xung cũng tính tương tự và thêm thời gian thực hiện lệnh LJMP LOOP là 2 chu kỳ máy và ta phải bổ sung thêm 14 chu kỳ máy.

3. Chương trình tạo thời gian trễ

a) Tạo thời gian trễ ngắn

Thời gian trễ nhỏ nhất là độ kéo dài của 1 xung là 1 μsec (tần số xung nhịp 12 MHz) hay 1,085 μsec (tần số xung nhịp 11,059 MHz). Muốn có thời gian trễ nhỏ hơn, ta phải dùng VDK với tần số xung nhịp lớn hơn.

b) Tạo thời gian trễ dài

Thời gian trễ lớn nhất ở chế độ 1 đếm 16 bit là đếm FFFFH xung hay 65536D xung, tức 65536 μsec (tần số xung nhịp 12 MHz). Muốn có thời gian trễ lớn hơn nữa, người ta dùng kết hợp với phương pháp vòng lặp khi giảm dần một thanh ghi bằng lệnh DJMP Ri, LOOP. Ví dụ 8-8 giới thiệu chương trình trễ 1 phút = 60 giây bằng phương pháp lặp dùng lệnh DJMP Ri, LOOP. Với việc thêm lệnh DJNZ Ri, LOOP này, người ta có thể tạo nên thời gian trễ bất kỳ (giây, phút, giờ, ngày, tháng, năm v.v.) và đó là nguyên tắc của đồng hồ điện tử dùng VDK để đếm thời gian.

8.4. ỨNG DỤNG TIMER/COUNTER TRONG PHÁT XUNG

8.4.1. Phát một xung với độ trễ khác nhau

Dùng tạo trễ, có thể duy trì mức điện áp ra (logic 0 hay 1) tại chân của một bit của một cổng song song nào đó, ta có xung vuông ra với thời gian kéo dài DELAY1 và thời gian nghỉ DELAY2.

Nếu: - Xung phát ra là một xung hay xung của một đơn hài và có thể có thời gian trễ DELAY (kể từ thời điểm phát xung) và độ kéo dài DELAY1 tùy ý.

- Xung phát ra là một chuỗi xung, hay xung của một đa hài, có thể có thời gian trễ DELAY, độ kéo dài DELAY1 và thời gian lặp lại DELAY2 tùy ý.

- Với $DELAY1 = DELAY2$: ta có các xung vuông có độ kéo dài bằng thời gian lặp lại.

- Với $DELAY1 \neq DELAY2$: ta có xung có độ kéo dài khác với thời gian lặp lại.

- Với $DELAY1, DELAY2$ nhỏ: ta có chuỗi xung độ kéo dài nhỏ, tần số lớn.

; Ví dụ 8-9: Chương trình tạo xung vuông dùng hai DELAY.

```
ORG 00H          ; bắt đầu chương trình từ địa chỉ 00H
MOV TMOD, #21H   ; Timer0 ở chế độ 1, Timer1 ở chế độ 2
LOOP: SETB P0.0   ; xác lập bit P0.3
      ACALL DELAY1 ; gọi trễ DELAY1
      CLR P0.3     ; xoá bit P0.3
      ACALL DELAY2 ; gọi trễ DELAY2
      LJP LOOP     ; lặp lại LOOP
DELAY1: MOV TL0, #0F2H ; nạp TL0 = #0F2H
      MOV TH0, #0FAH  ; nạp TH0 = #0FAH
      SETB TR0        ; bắt đầu đếm thời gian trễ
L0: JNB TF0, L0       ; chờ cờ tràn TF0 = 1
      CLR TR0         ; ngừng đếm TIMER0
      CLR TF0         ; xoá cờ tràn
      RET            ; kết thúc chương trình con
DELAY2: MOV TH1, #55H ; nạp TH1 = #55H
      MOV TL1, TH1    ; nạp TL1 = TH1 = #55H
      SETB TR1        ; bắt đầu đếm trễ
L1: JNB TF1, L1       ; chờ cờ tràn TF1 = 1
      CLR TR1         ; ngừng đếm TIMER0
      CLR TF1         ; xoá cờ tràn
      RET            ; kết thúc chương trình con
END                ; kết thúc chương trình dịch
```

Ví dụ 8-9 giới thiệu chuỗi xung vuông phát ra từ chân P0.0 với độ kéo dài DELAY1 và thời gian nghỉ DELAY2 khác nhau.

Nhược điểm của phương pháp phát xung này là tốn bộ định thời mà số bộ định thời trong một VDK chỉ có 2 tới 3.

8.4.2. Phát tốc độ baud cho cổng nối tiếp

Trong trao đổi tin nối tiếp (chương 9), cần các xung nhịp $R \times C$, $T \times C$ để phát hay thu các tín hiệu phát $T \times D$ hay nhận $R \times D$. Khi đó thường dùng:

- Bộ định thời Timer0 hay Timer1 hoạt động ở chế độ 2 (ví dụ 8-2) cho VDK 8051.
- Bộ định thời Timer2 hoạt động ở chế độ thu nhận (ví dụ 8-4) cho VDK 8052.

8.5. ỨNG DỤNG TRONG ĐẾM THỜI GIAN

8.5.1. Nguyên tắc chung

Với máy phát xung và bộ định thời, bộ đếm, ta có thể dùng VDK để đếm và đo thời gian.

Nguyên tắc của đếm thời gian là:

- *Chọn một thời gian đơn vị*: đơn vị thời gian nhỏ nhất trong VDK là 1 (cho tần số nhịp 12MHz) hay 1,085 micro giây (cho 11,059 MHz). Tùy độ chính xác yêu cầu, ta có thể chọn đơn vị thời gian là mili giây hay 1/100 tới 1/10 giây (đồng hồ dùng trong thể thao) hoặc giây (đồng hồ hiện giờ). Dùng bộ định thời (chế độ Timer với bit /T = 0 của thanh ghi TMOD) để đếm các xung nhịp (sau khi chia 12) của bộ dao động thạch anh của VDK làm đơn vị đo thời gian (ký hiệu DELAY0) tùy độ chính xác yêu cầu (msec hay sec). Trong chương trình, có thể:

+ *Chờ chờ tràn TF*: rồi tiếp tục các lệnh khác. Đây là đồng hồ thời gian không dùng ngắt định thời (Timer0,1, Timer2).

+ *Chờ ngắt chương trình do Timer0,1, Timer2*: trong khi chờ chờ tràn TF = 1 (xem chương 10) bằng cách chuyển sang chương trình phục vụ ngắt, chương trình có thể chạy các lệnh khác như hiển thị số đếm của thời gian. Cách này tiết kiệm được thời gian thực hiện lệnh (chạy song song hai hay nhiều hành động) nhưng phải sử dụng ngắt về thời gian.

+ *Tăng số biến đếm về thời gian*: sau khi đếm xong một đơn vị thời gian DELAY0, sẽ tăng số biến đếm về đơn vị nhỏ nhất lên 1 cho tới giá trị cực đại của thang đo thời gian.

+ *So sánh với giá trị cực đại của thang đo*: sau khi tăng biến đếm, phải có lệnh so sánh biến đếm với giá trị cực đại của thang đo thời gian đó, ví dụ giá trị #60D cho giây, phút, giờ hay 12 hoặc 24 cho ngày. Nếu chưa bằng, lại tiến hành đếm và tăng đơn vị thời gian bằng lệnh CJNE BIEN, #BIEN max, NHAN với NHAN là địa chỉ của lệnh bắt đầu đếm của thang đo.

+ *Hiển thị thời gian*: thời gian có thể hiển thị bằng đèn hiển thị (LED 7 đoạn, ma trận diot, LCD) để người dùng biết được thời gian hiện tại (xem tập II về hiển thị).

+ *Báo hiệu thời gian*: có thể dùng các phương tiện báo hiệu như đèn nhấp nháy, loa, chuông để báo hiệu thời gian cần báo. Đây là đồng hồ báo thời gian hay báo thức.

+ *Điều khiển bắt đầu/dừng đếm thời gian*: với các chuyển mạch START/STOP, có thể tạo các giá trị 1 hay 0 cho các bit của cổng song song (xem chương 7) để điều khiển đếm thời gian. Đây là nguyên tắc của một đồng hồ dùng trong thi đấu thể thao.

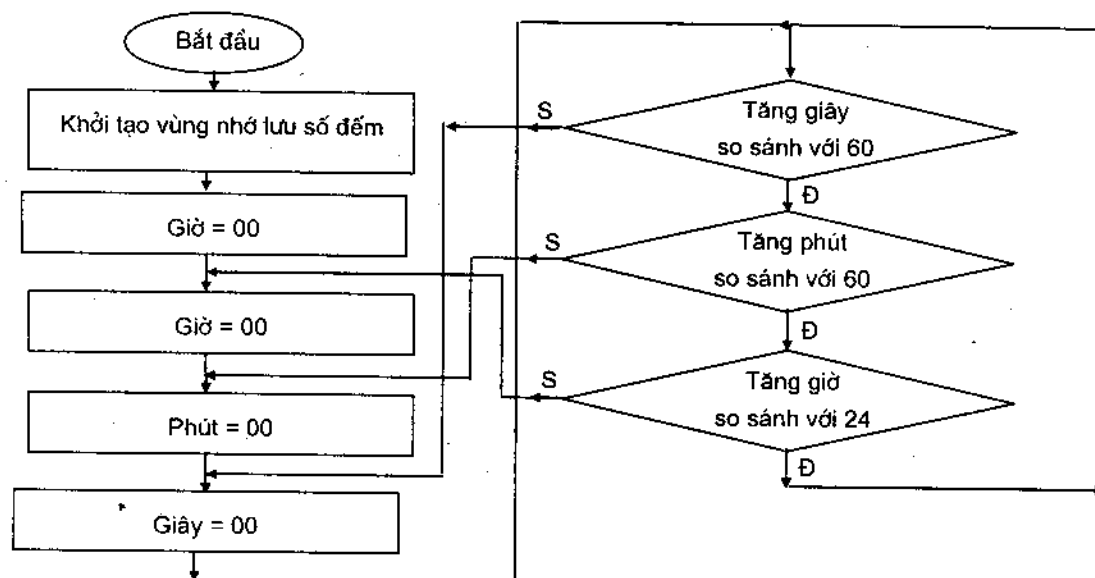
+ *Chỉnh thời gian sai lệch*: sau một thời gian, đếm thời gian có chênh lệch so với thời gian thực, người ta phải sử dụng ngắt để hiệu chỉnh (xem chương 10).

8.5.2. Đồng hồ đếm giây

Đây là đồng hồ đếm từ mili giây tới đơn vị giây. Ví dụ 8-10 giới thiệu chương trình đếm từ 0 tới 60 giây, không có điều khiển START/STOP. Chúng ta chưa xét tới việc hiển thị thời gian. Ký hiệu DELAY0 là đơn vị thời gian cơ bản và viết dưới dạng một chương trình con. Ta dùng các ô nhớ để ghi các giá trị thời gian như chục, trăm, ngàn mili giây (hay giây), đơn vị giây, chục giây.

Chú ý: DELAY0 là trễ 10ms vì D817H là số bù 2 của FFFFH-D817H = 27E8H = 10.

Nếu thêm hai lệnh chờ ấn phím Start (vào bắt đầu chương trình, sau MAIN) và Stop (vào vị trí thích hợp - tùy yêu cầu đếm), ta sẽ có một đồng hồ thể thao từ 10 ms.



Hình 8-10. Giải thuật của đồng hồ điện tử dùng VDK

8.5.3. Đồng hồ chỉ giờ, phút, giây

; Ví dụ 8-10: Chương trình đếm mili giây tới giây, kết quả trong các ô nhớ

<pre> ORG 00H LJMP MAIN ORG 100H MAIN: TMOD, #001H TMOD, #001H MS4: MOV 10H, #00H ; ô nhớ chứa 10 s MS3: MOV 11H, #00H ; ô nhớ chứa 1 s MS2: MOV 12H, #00H ; ô nhớ chứa tới 100 ms MS1: MOV 13H, #00H ; ô nhớ chứa 10 ms MS0: LCALL DELAY0 ; gọi trễ 10 ms INC 13H ; tăng đếm 10 ms MOV A, 13H CJNE A, #10, MS0 ; so sánh với 100 ms INC 12H ; tăng hàng 100 ms MOV A, 12H CJNE A, #10, MS1 ; so sánh với 1000 ms INC 11H MOV A, 11H CJNE A, #10, MS2 </pre>	<pre> INC 10H MOV A, 10H CJNE A, #10, MS3 INC 10H MOV A, 10H CJNE A, #10, MS4 SJMP \$ DELAY0: MOV R0, #10H DEL: CLR TR0 MOV TH0, #00H MOV TL0, #00 SETB TR0 JNB TF0, \$ CLR TF0 DJNZ R0, DEL RET END </pre>
--	---

Đồng hồ này có đặc điểm sau:

Chọn đơn vị thời gian DELAY0 là 50 ms.

Chỉ thị thời gian tới giây, phút và tới 24 giờ.

Hình 8-10 giới thiệu giải thuật của chương trình ở ví dụ 8-11. Các biến đếm giây, phút, giờ và BDN (ms) được gán cho các ô nhớ 32H, 33H, 34H và 31H.

; Ví dụ 8-11: Đồng hồ đếm 50 ms, giây, phút tới 24 giờ, ghi ra số BCD trong các ô nhớ	
GIAY EQU 32H ; gán GIAY PHUT EQU 33H ; gán PHUT GIO EQU 34H ; gán GIO BDN EQU 31H ; gán 50 ms ORG 00H LJMP MAIN ORG 100H MAIN: MOV TMOD, #01H ; Timer0, chế độ 1 CLR TF0 ; xóa cờ tràn MOV TH0, #HIGH (-50000); đếm 50 ms MOV TL0, #LOW (-50000) SETB TR0 ; bắt đầu đếm MAIN4: MOV GIO, #00H ; xóa ô nhớ chứa giờ MAIN3: MOV PHUT, #00H; xóa ô nhớ chứa phút MAIN0: MOV GIAY, #00H; xóa ô nhớ chứa giây MAIN1: MOV BDN, #00H ; xóa ô nhớ chứa 50 ms	MAIN2: INC BDN MOV A, BDN CJNE A, #20h, MAIN2; xét = 1 s ? MOV A, GIAY ; gọi giây ADD A, #1 ; tăng giây DA A ; chỉnh BCD MOV GIAY, A ; ghi lại giây CJNE A, #60h, MAIN1; xét = 60 s ? MOV A, PHUT ; gọi phút ADD A, #1 ; tăng phút DA A ; chỉnh BCD MOV PHUT, A ; ghi lại phút CJNE A, #60h, MAIN0; xét = 1 giờ ? MOV A, GIO ; gọi giờ ADD A, #1 ; tăng giờ DA A; chỉnh BCD ; chỉnh BCD MOV GIO, A ; ghi lại giờ CJNE A, #24h, MAIN3; xét = 24 giờ? SJMP MAIN4 ; trở về đếm tiếp END ; kết thúc dịch

Nếu cần hiển thị số đếm thời gian, ta có thể đưa các ô nhớ trên ra các đèn chỉ thị (LED đơn, LED 7 đoạn, ma trận diot hay màn tinh thể lỏng LCD) (xem tập II về hiển thị).

Nếu cần báo số phút sau 15 phút, sau 1 giờ, ta cần mắc thêm bộ phận báo hiệu bằng âm thanh (xem chương 7).

Nếu cần báo thức, ta đặt sẵn số giờ, đồng hồ sẽ báo hiệu khi hai số trùng nhau.

8.6. ỨNG DỤNG TRONG ĐO THỜI GIAN

8.6.1. Nguyên tắc và mạch đo một khoảng thời gian

Muốn đo một khoảng thời gian dùng VDK, ta phải đếm các xung nhịp trong độ kéo dài của khoảng thời gian đó bằng một bộ đếm. Như vậy, ta phải có:

- Một bộ Timer để đếm các xung đơn vị tần số cao (của bộ dao động thạch anh hay xung ngoài vào các lối vào T0, T1).

- Tín hiệu Start của thời điểm Bắt đầu của khoảng thời gian để khởi phát bộ đếm.

- Tín hiệu Stop của thời điểm Kết thúc đếm.

Người ta đã dùng một bộ định thời (timer0, 1, 2) của VDK để đếm các xung nhịp với tần số thích hợp, còn các tín hiệu Start, Stop có thể được thực hiện bằng mạch (phần cứng) hay bằng chương trình (phần mềm) trong các mạch đo cụ thể như sau:

- Tín hiệu cần đo khoảng thời gian là xung dương:

Tín hiệu cần đo khoảng thời gian được đưa vào lối vào /INT0 (chân P3.2) hay /INT1 (chân P3.3) của VDK 8051 (hình 8-1) và phải xóa các bit trên về mức thấp (logic 0).

Nếu đã ghi bit GATE0 (hay GATE1) = 1 vào thanh ghi TMOD, thì sau khi thực hiện lệnh SETB TR0 (hay TR1), chuyển mạch S1 sẽ đóng và chuyển mạch S2 còn để hở. Do đó, chưa có các

tín hiệu xung đưa vào bộ đếm nên bộ đếm chưa đếm (hình 8-1) vì các lối vào /INT0 (hay /INT1), chân P3.2 (hay P3.3) ở mức thấp (logic 0) và lối ra của vi mạch AND mức cao có mức logic 0.

Khi xuất hiện tín hiệu cần đo khoảng thời gian t_{do} , lối vào này lên cao (logic 1), có một mức logic 1 ở lối ra vi mạch AND mức cao và chuyển mạch S2 đóng, có chuỗi xung vào bộ đếm.

Khi xung cần đo khoảng thời gian kết thúc, lối vào /INT0 (hay /INT1) trở về mức thấp, bộ đếm xung ngừng.

Số đếm trong các thanh ghi TH, TL chính là số đếm N của các xung nhịp. Nếu chu kỳ xung nhịp là t_0 , ta có khoảng thời gian đo được là:

$$t_{do} = N \times t_0$$

Chương trình đo một khoảng thời gian có những đặc điểm sau:

- Giống chương trình đếm thời gian (ví dụ 8-10) nhưng có những điểm khác là chỉ đếm thời gian khi có xung vào dương (ghi GATE = 1 và khi P3.2 = 1).

- Bắt đầu đếm khi có xung dương với kiểm tra bit P3.2 = 1 bởi lệnh chờ JNB P3.2, \$.

- Sau mỗi lần đếm (50ms, giây, phút, giờ) phải có những lệnh kiểm tra sự kết thúc xung vào bằng kiểm tra bit P3.2 gửi qua bit C. Nếu có kết thúc xung, P3.2 = 0, bit đảo của C = 1 và lệnh JC KT sẽ kết thúc đếm hay đo độ rộng xung.

- Số thời gian đo (hay đếm) được cũng ghi vào các ô nhớ để hiển thị (ra cổng cho LED đơn, LED 7 đoạn, LCD, ma trận diot).

- Thời gian đo có thể tới 24 giờ hoặc hơn (thêm n tăng biến thanh ghi Rn khi quá 24 giờ).

Xung cần đo phải đưa vào một bit bất kỳ của cổng song song và trạng thái ban đầu ở mức logic 0. Bằng lệnh JB Px.y, \$, ta phát hiện sườn xung xuống logic 0 để đóng chuyển mạch S2 và kích thích bộ đếm bắt đầu đếm. Sau đó, bằng lệnh JNB Px.y, \$, ta phát hiện thời điểm kết thúc xung cần đo khoảng thời gian, ta đưa lệnh CLR TR0 (hay TR1) để kết thúc đếm.

: Ví dụ 8-12: Đo độ kéo dài xung âm, ghi số BCD trong các ô nhớ 31H, 32H, 33H, 34H

GIAY EQU 32H	; gán GIAY	MOV A, GIAY	; gọi giây
PHUT EQU 33H	; gán PHUT	ADD A, #1	; tăng giây
GIO EQU 34H	; gán GIO	DA A	; chỉnh BCD
BDN EQU 31H	; gán 50 ms	MOV GIAY, A	; ghi lại giây
		MOV C, P3.4	
ORG 00H		JC KT	
LJMP MAIN		CJNE A, #60h, MAIN1; xét = 60s ?	
ORG 100H		MOV A, PHUT	; gọi phút
MAIN:		ADD A, #1	; tăng phút
MOV TMOD, #09H	; Timer0, chế độ 1	DA A	; chỉnh BCD
CLR TF0	; xóa cờ tràn	MOV PHUT, A	; ghi lại phút
SETB P3.4		MOV C, P3.4	
MOV TH0, #HIGH (-50000); đếm 50 ms		JC KT	
MOV TL0, #LOW (-50000)		CJNE A, #60h, MAIN0; xét = 1 giờ ?	
MAIN4: MOV GIO, #00H	; xóa nhớ chứa giờ	MOV A, GIO	; gọi giờ
MAIN3: MOV PHUT, #00H	; xóa nhớ chứa phút	ADD A, #1	; tăng giờ
MAIN0: MOV GIAY, #00H	; xóa nhớ chứa giây	DA A; chỉnh BCD	; chỉnh BCD
MAIN1: MOV BDN, #00H	; xóa nhớ chứa 50 ms	MOV GIO, A	; ghi lại giờ
JB P3.4, \$		MOV C, P3.4	
SETB TR0	; bắt đầu đếm	JC KT	
MAIN2: INC BDN		CJNE A, #24h, MAIN3; xét = 24 giờ?	
MOV A, BDN		KT: NOP	
MOV C, P3.4	; kiểm tra kết thúc xung	CLR TR0	
JC KT	; nếu P3.4 = 1, kết thúc	SJMP \$	
CJNE A, #20h, MAIN2; xét = 50 ms ?		END	; kết thúc dịch

Ví dụ 8-12 giới thiệu chương trình đo khoảng thời gian của một xung âm.

Chương trình này có đặc điểm là:

- Lối vào xung âm là chân P3.4.
- Lệnh chờ xung âm là JB P3.4, \$.
- Lệnh kiểm tra kết thúc xung âm là MOV C, P3.4 và JC KT để nhảy tới nhãn KT của lệnh dừng đếm.

8.6.2. Đo nhiều khoảng thời gian nối tiếp

Trong thực tế, nhiều khi có yêu cầu đo nhiều khoảng thời gian nối tiếp nhau của nhiều quá trình liên tiếp như đo nhiều khoảng thời gian của một quá trình nào đó như đo nhiều thời gian bay của một hạt qua nhiều cảm biến (detector) để biết được năng lượng của hạt.

Đo nhiều khoảng thời gian nối tiếp là đo nhiều khoảng thời gian đơn lẻ theo nguyên tắc sau:

- Mỗi khoảng thời gian được đo bởi một timer riêng. Có thể dùng hai timer thay đổi nhau để đo nhiều khoảng thời gian.
- Các timer kế tiếp nhau đo các khoảng thời gian khác nhau: timer này tắt để ghi số đếm thì timer kia phải bật để đo khoảng thời gian kế tiếp.
- Nếu cần biết thời gian tổng cộng cho các khoảng thời gian thành phần, ta lấy tổng số các khoảng thời gian thành phần.
- Từ các dữ liệu của các khoảng thời gian đo được, ta có thể vẽ được sự phân bố của các khoảng thời gian hay tính khoảng thời gian trung bình của phép đo.

8.7. ỨNG DỤNG TRONG ĐẾM XUNG

Trong thực tế, có nhiều yêu cầu về đếm các sự kiện như:

- Đếm số người hay phương tiện giao-thông đi qua một chiếc cầu, một cửa khẩu hay một băng truyền.
- Đếm số sản phẩm chế tạo được sau một thời gian sản xuất.
- Đếm số hạt bức xạ từ một nguồn phát xạ, ví dụ nguồn phóng xạ hạt nhân.

Người ta dùng Timer/Counter của VĐK và lập trình để đếm các xung vào các lối vào T0, T1 (của 8051 hay cả T2 của 8052). Khi đó, các vật cần đếm đi qua một cửa hẹp phải biến đổi thành một tín hiệu điện có độ kéo dài nhỏ.

Có thể có các loại đếm như sau:

- Đếm lên từ giá trị ban đầu TH, TL tới FFFFH rồi sang 0000H để chỉ thị một cờ tràn TF (xem ví dụ 8-1).
- Đếm lên từ giá trị 0000H tới giá trị TC nào đó, trong thời gian TP (Time Preset đặt sẵn), từ đó đọc được giá trị đếm TC.
- Đếm lên từ giá trị 0000H tới giá trị TC đặt trước, trong thời gian TP, từ đó đọc được thời gian đếm TC.
- Đếm liên tiếp nhiều kênh.

8.7.1. Đếm trong thời gian đặt trước

Muốn đếm kiểu này, ta phải có:

- Một bộ đếm để đếm từ giá trị ban đầu (thường là TH = TL = 00H) với thanh ghi TMOD có bit C/T = 1 và các bit chế độ ở M = 1 (đếm 16 bit). Có thể dùng chế độ M = 3 để biến Timer0 thành hai bộ đếm 8 bit để đếm xung và xác định thời gian đếm. Bộ đếm xung ngoài vào bit T0 (P3.4) này phải có xung có sườn âm nên chương trình mô phỏng phải thêm lệnh chờ các mức logic 1 và 0 bằng các lệnh liên tiếp JB P3.4, \$, JNB P3.4, \$ và tạo xung vào đếm bằng cách kích chuột vào bit P3.4 của cổng vào trong chương trình mô phỏng Pinnacle.

- Một bộ định thời để đếm thời gian từ giá trị ban đầu (thường là TH = TL = 00H) với thanh ghi TMOD có bit C/T = 0 và các bit chế độ ở M = 1 (đếm 16 bit), bộ đếm thời gian này phải được đặt tới giá trị TPH và TPL định sẵn.

Ta phải khởi phát và dừng hai bộ đếm trên ở cùng một thời điểm bằng cách ghi các bit TR0, TR1 = 1 hay 0 với lệnh MOV vào thanh ghi TMOD.

Có thể khởi phát và dừng hai bộ đếm trên một cách liên tiếp bằng cách dùng các lệnh SETB TR0, SETB TR1 và CLR TR0, CLR TR1 ở trong chương trình.

Một bàn phím để đặt thời gian TP (xem tập II về bàn phím).

- Có chương trình con so sánh giá trị thời gian đặt trước TPH, TPL với thời gian đếm TCH = TH1 và TCL = TL1 của bộ đếm. Khi so sánh, phải dừng bộ đếm thời gian Timer1. Khi TC = TP, có lệnh xác lập cờ CMPTPTC để dừng cả hai bộ đếm. Đọc các kết quả về số đếm (đặt trước) và thời gian đếm ra các cổng P0, P1, P2, P3.

Ví dụ 8-13. Chương trình đếm trong thời gian đặt trước TP.

; Ví dụ 8-13: Chương trình đếm trong thời gian TPH, TPL	
<pre> TPH EQU 30H TPL EQU 31H TCL EQU 32H TCH EQU 33H T2_REG EQU 34H ORG 20H CMPFLAG DS 1 ORG 00H ; bắt đầu MOV TMOD, #15H ; timer1,0 ở chế độ 1 MOV TL0, #00H MOV TH0, #00H ; nạp TH0 vào TL0 MOV TL1, #00H MOV TH1, #00H ; nạp TH0 vào TL0 MOV TPL, #0F0H MOV TPH, #0FFH ; nạp TH0 vào TL0 JB P3.4, \$ JNB P3.4, \$ SETB TR0 SETB TR1 ; bắt đầu đếm LOOP: MOV TCH, TH1 MOV TCL, TL1 CLR TR1 CALL COMPARTPTC ; so sánh thời gian JBC CMPFLAG, KT SETB TR1 JMP LOOP </pre>	<pre> KT: CLR TR0 CLR TR1 MOV P0, TH0 MOV P1, TL0 MOV P2, TH1 MOV P3, TL1 SJMP \$ COMPARTPTC: PUSH ACC PUSH PSW MOV TCH, TH1 MOV TCL, TL1 MOV A, TPH CJNE A, TCH, NOT_EQH CLR C MOV A, TPL CJNE A, TCL, NOT_EQL JMP TOVER NOT_EQH: JNC DONE TOVER: SETB CMPFLAG JMP DONE NOT_EQL: JC TOVER DONE: POP PSW POP ACC RET END ; kết thúc chương trình dịch </pre>

Ta phải khởi phát và dừng hai bộ đếm trên ở cùng một thời điểm bằng cách ghi các bit TR0, TR1 = 1 hay 0 với lệnh MOV vào thanh ghi TMOD.

Có thể khởi phát và dừng hai bộ đếm trên một cách liên tiếp bằng cách dùng các lệnh SETB TR0, SETB TR1 và CLR TR0, CLR TR1 ở trong chương trình.

Cần có một bàn phím để đặt thời gian TP (xem tập II về bàn phím).

- Có chương trình con so sánh giá trị thời gian đặt trước TPH, TPL với thời gian đếm TCH = TH1 và TCL = TL1 của bộ đếm. Khi so sánh, phải dừng bộ đếm thời gian Timer1. Khi TC = TP, có lệnh xác lập cờ CMPTPTC để dừng cả hai bộ đếm. Đọc các kết quả về số đếm (đặt trước) và thời gian đếm ra các cổng P0, P1, P2, P3.

8.7.2. Đếm với số đếm đặt trước

Tương tự việc đếm có thời gian đặt trước, đếm với số đếm đặt trước có đặc điểm như sau:

- Cũng có một bộ đếm xung ngoài vào (ví dụ chân T0 của bit P3.4) và bộ đếm thời gian (ví dụ Timer1).

- Cũng có chương trình con so sánh số đếm hiện hành TC = TH0, TL0 với số đếm đặt trước TP = TPH, TPL và xác lập cờ ngắt CMPIPTC để kết thúc chương trình.

- Khi hai số đếm bằng nhau, cờ so sánh xác lập, chuyển tới kết thúc chương trình. Đọc các kết quả về số đếm (đặt trước) và thời gian đếm ra các cổng P0, P1, P2, P3.

Ví dụ 8-14. giới thiệu chương trình đếm với số đếm đặt trước TP = TPH, TPL = 0005.

Chú ý: Hai chương trình ở các ví dụ 8-14, 8-15 có số đo thời gian nhỏ, tối đa với số đếm 16 bit (cỡ 70ms cho tần số 11,059 MHz). Muốn thời gian lớn hơn, phải dùng thêm các vòng lặp trễ với việc giảm các thanh ghi (xem phân tạo trễ Delay với thời gian lớn ở trên) hay dùng nhiều bộ đếm thời gian nối tiếp nhau.

; Ví dụ 8-14: Chương trình đếm có so sánh số đếm bằng 5	
<pre> TPH EQU 30H TPL EQU 31H TCL EQU 32H TCH EQU 33H ORG 20H CMPFLAG DS 0 ORG 00H MOV TMOD, #05H ; Counter0 MOV TL0, #00H ; MOV TH0, #00H ; nạp TH0 vào TL0 MOV TPL, #05H ; MOV TPH, #00H ; nạp TH0 vào TL0 SETB TR0 ; đóng mạch bộ đếm JB P3.4, \$; chờ sườn xung âm SETB TR1 ; bắt đầu đếm thời gian LOOP: JNB P3.4, \$; chờ kết thúc xung vào CLR TR1 ; ngừng đếm thời gian CALL COMPARTPTC; so sánh số đếm JBC CMPFLAG, KT; nếu bằng, tới KT JB P3.4, \$; chờ xung vào SETB TR1 ; bắt đầu đếm thời gian JMP LOOP ; trở về LOOP KT: CLR TR0 ; kết thúc, dừng đếm xung CLR TR1 ; dừng đếm thời gian SJMP \$; kết thúc chờ </pre>	<pre> MOV P0, TH0 ; đọc số đếm MOV P1, TL0 MOV P2, TH1 ; đọc thời gian đếm MOV P3, TL1 COMPARTPTC: ; so sánh số đếm với TP PUSH ACC PUSH PSW CLR C MOV TCH, TH0 MOV TCL, TL0 MOV A, TPH CJNE A, TCH, NOT_EQH CLR C MOV A, TPL CJNE A, TCL, NOT_EQL JMP TOVER NOT_EQH: JNC DONE TOVER: SETB CMPFLAG JMP DONE NOT_EQL: JC TOVER DONE: POP PSW POP ACC RET END ; kết thúc chương trình dịch </pre>

- Có thể dùng Timer2 thay cho Timer0,1 để đếm thời gian và đếm xung vào.

- Có thể dùng các Timer ở chế độ ngắt chương trình (chương 10) để ấn định thời gian hoặc số đếm. Khi đó hai chương trình đếm xung và đếm thời gian độc lập nhau và để viết chương trình với thời gian đếm lớn. Còn đếm xung, cũng có thể mở rộng bộ đếm lớn hơn 16 bit.

- Có thể đếm số xung trung bình trong một khoảng thời gian lớn bằng cách đếm tổng số xung rồi chia cho số thời gian đã đếm.

- Có thể đếm số xung định trước rồi tính thời gian trung bình cho 1 xung hay khoảng thời gian trung bình giữa hai xung vào đếm để suy ra cường độ xung (số xung phát ra trong một đơn vị thời gian).

CÂU HỎI VÀ THỰC HÀNH

I. Câu hỏi

1. Hãy phân biệt chế độ đếm và chế độ định thời của bộ Timer/Counter.
2. Nêu các chế độ 0, 1, 2, 3 của bộ Timer/Counter.
3. Nêu thời gian tối thiểu của tối đa của bộ định thời ở chế độ khác nhau của Timer.
4. Nêu dung lượng đếm tối đa của bộ đếm ở chế độ khác nhau và cách tăng thời gian định thời.
5. Nêu ứng dụng của định thời trong VDK 8051/52.
6. Nêu ứng dụng của bộ đếm trong VDK 8051/52.
7. Hãy so sánh hoạt động của Timer0,1 với Timer2.
8. Nêu ứng dụng của Timer trong trao đổi tin nối tiếp.

II. Thực hành

1. Quan sát các thanh ghi của các timer/counter của chương trình mô phỏng Pinacle.
Chúng ta có thể quan sát được Timer và Counter của chương trình mô phỏng PINNACLE theo trình tự sau:

- Vào thư mục PINNACLE trên máy vi tính PC;
- Vào thực đơn View;
- Vào thực đơn Timer.

Ta sẽ có cửa sổ Timer, Timer2, sẽ hiển thị các giá trị của các thanh ghi của Timer0,1, Timer2 như TCON, T2CON, TMOD, T2MOD, TL0, TH0, TL1, TH1, TL2, TH2 RCAP2.

2. Vào thực đơn File/new để soạn thảo chương trình cho Timer0, 1 như ví dụ 8-1 với một trong các chế độ 0, 1, 2, 3. Dịch, kết nối và cho chạy chương trình từng bước và cả chương trình. Theo dõi sự thay đổi giá trị các thanh ghi TL, TH và các cờ tràn TF0, TF1.

3. Vào thực đơn File/new để soạn thảo chương trình cho Timer0,1 như ví dụ 8-2 với chế độ 2 rồi chế độ 3. Dịch, kết nối và cho chạy chương trình từng bước và cả chương trình. Theo dõi sự thay đổi giá trị các thanh ghi TL, TH và các cờ tràn TF0, TF1.

4. Vào thực đơn File/new để soạn thảo chương trình cho Timer0, 1 như ví dụ 8-3, 8-4, 8-5 và 8-6 về các thời gian trễ khác nhau. Dịch, kết nối và cho chạy chương trình từng bước và cả chương trình. Theo dõi sự thay đổi giá trị các thanh ghi TL, TH và các cờ tràn TF0, TF1.

5. Vào thực đơn File/new để soạn thảo chương trình cho Timer0, 1 như ví dụ 8-7 cho chế độ đếm. Đưa một máy phát xung và chân T0 hoặc T1 (hoặc tạo xung bằng kích chuột vào cổng vào). Dịch, kết nối và cho chạy chương trình (từng bước và cả chương trình). Theo dõi sự thay đổi giá trị các thanh ghi TL, TH và các cờ tràn TF0, TF1.

6. Vào thực đơn File/new để soạn thảo chương trình cho Timer2 như ví dụ 8-8. Dịch, kết nối và cho chạy chương trình từng bước và cả chương trình. Theo dõi sự thay đổi giá trị các thanh ghi TL, TH và các cờ tràn TF2.

7. Thực hiện dịch, kết nối và cho chạy (từng bước, cả chương trình) cho các ví dụ 8-8, 8-9, 8-10, 8-11, 8-12, 8-13.

8. Thực hiện dịch, kết nối và cho chạy (từng bước, cả chương trình) cho các ví dụ về đếm xung ở các ví dụ 8-14, 8-15.

Chương 9

CỔNG VÀO/RA NỐI TIẾP

Mục đích của chương:

- Khảo sát cấu trúc và hoạt động của cổng nối tiếp của VĐK ở các chế độ 0, 1, 2, 3 khác nhau.
- Lập trình trao đổi tin nối tiếp khi nhận tin và phát tin (chữ, chuỗi) ở các chế độ 0, 1, 2, 3.

9.1. CẤU TRÚC VÀ HOẠT ĐỘNG CỦA CỔNG NỐI TIẾP

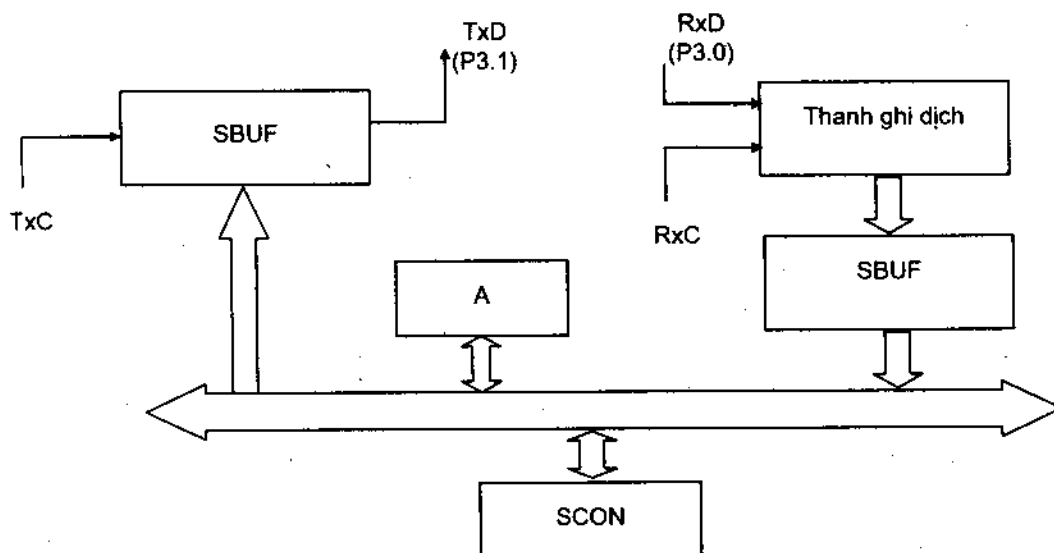
9.1.1. Cấu trúc cổng nối tiếp

Bên trong VĐK nói chung hay VĐK 8051/8052 nói riêng, đều có một cổng trao đổi tin nối tiếp hai chiều thuận nghịch (song công) để phát tin nối tiếp từ thanh ghi đệm truyền nối tiếp SBUF ra chân TXD (chân P3.1- số 11) và nhận tin nối tiếp từ chân RxD (chân P3.0 - số 10) vào thanh ghi đệm nhận nối tiếp SBUF. VĐK trao đổi tin song song với các thanh ghi đệm truyền hay nhận nối tiếp qua thanh ghi chứa A (hay thanh ghi tích lũy Acc).

1. Cấu trúc chung

Sơ đồ cấu tạo cổng nối tiếp như hình 9-1, gồm:

- Hai thanh ghi đệm dữ liệu SBUF: (địa chỉ 99H chung cho cả hai thanh ghi), địa chỉ theo byte, dùng để nhận tin song song từ thanh ghi A, rồi đưa tin ra nối tiếp trên lối ra TXD và nhận tin vào nối tiếp từ lối vào RxD rồi đưa tin song song vào thanh ghi A.



Hình 9-1. Cấu tạo tổng quát của cổng nối tiếp của VĐK 8051/8052

- Thanh ghi điều khiển/trạng thái trao đổi tin nối tiếp SCON : (địa chỉ 98H và có thể địa chỉ theo bit). Thanh ghi điều khiển/trạng thái này có dạng như hình 9-2 với các bit như bảng 9-1.

- *Bộ tạo tốc độ nhịp cho tín hiệu nối tiếp*: VDK dùng xung nhịp với tốc độ baud của một trong các bộ Timer (0, 1 hay 2).

- *Thanh ghi dịch*: dịch dữ liệu nối tiếp thành song song.

Bảng 9-1. Các bit của thanh ghi SCON

Bit	Ký hiệu	Công dụng	Tốc độ baud
7	SM0 SM1	<i>Chế độ nối tiếp 0</i> : chế độ thanh ghi dịch, tốc độ baud cố định	$f_{osc}/12$
	0 0	<i>Chế độ nối tiếp 1</i> : 8 bit UART (nối tiếp không đồng bộ), baud thay đổi.	tốc độ thay đổi
	0 1		
6	1 0	<i>Chế độ nối tiếp 2</i> : 9 bit UART (truyền nối tiếp không đồng bộ) tốc độ $f_{osc}/32$ hay $f_{osc}/64$	$f_{osc}/64$
	1 1	<i>Chế độ nối tiếp 3</i> : 9 bit UART (truyền nối tiếp không đồng bộ) tốc độ.	hay $f_{osc}/32$ tốc độ baud thay đổi
5	SM2	Cho phép hành động truyền thông nhiều vi xử lý ở chế độ 2 và 3. Ở chế độ 0, SM2 sẽ bằng 0. Ở chế độ 2 và 3, SM2 xác lập lên 1. RI sẽ không bị tác động nếu bit thứ 9 (RB8) nhận được là 0. Ở chế độ 2 và SM2 = 1, RI sẽ không bị tác động nếu một bit stop hợp lệ không được nhận.	
4	REN	Cho phép nhận. Cho phép nhận nối tiếp. Xác lập bởi lệnh để cho phép nhận. Xóa bởi lệnh để từ chối nhận.	
3	TB8	Bit truyền 8. Bit dữ liệu thứ 9, sẽ được truyền ở chế độ 2 và 3. Xác lập và xóa bởi lệnh như mong muốn (truyền 9 bit tin, kể cả bit chẵn lẻ). Ở chế độ 0, TB8 không dùng.	
2	RB8	Bit nhận 8. Bit dữ liệu thứ 9, sẽ được nhận ở chế độ 2 và 3. Ở chế độ 1, nếu SM2 = 0, RB8 là bit stop mà nó nhận. Ở chế độ 0, RB8 không dùng.	
1	TI	Cờ ngắt truyền. Xác lập bởi phần cứng ở cuối của bit thứ 8 ở chế độ 0 hay ở bắt đầu của bit stop ở các chế độ khác trong bất kỳ truyền nối tiếp nào. Phải bị xóa bằng lệnh (phần mềm).	
0	RI	Cờ ngắt nhận. Xác lập bởi phần cứng ở cuối của bit thứ 8 ở chế độ 0 hay ở trong thời gian của bit stop ở các chế độ khác trong bất kỳ truyền nối tiếp nào. Phải bị xóa bằng lệnh (phần mềm).	

Địa chỉ	9FH	9EH	9DH	9CH	9BH	9AH	99H	98H
Bit	D7	D6	D5	D4	D3	D2	D1	D0
Tên bit	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Hình 9-2. Thanh ghi điều khiển nối tiếp SCON

2. Thanh ghi điều khiển/trạng thái nối tiếp SCON

Thanh ghi này có các bit sau:

- *Các bit chế độ SM0, SM1*: xác định chế độ trao đổi tin như sau:

+ *Chế độ 0 (SM0, SM1 = 0)*: truyền, nhận 8 bit (không có các bit Start, Stop), bit thấp nhất truyền trước, tốc độ baud = $1/12 f_{osc}$

+ *Chế độ 1 (SM0 = 1, SM1 = 0)*: truyền nhận 10 bit (8 bit dữ liệu + 2 bit Start, Stop), tốc độ baud thay đổi được do lập trình.

+ *Chế độ 2 (SM0 = 0, SM1 = 1)*: truyền nhận 11 bit (8 bit dữ liệu + 1 bit chẵn/lẻ (ghi vào RB8, TB8 bằng lệnh) + 2 bit start, stop), tốc độ baud lập trình bằng $1/32$ hay $1/64 f_{đồng}$.

+ *Chế độ 3 (SM0 = SM1 = 1)*: giống chế độ 2, khác là tốc độ baud thay đổi.

- *Bit chế độ SM2*: dành cho trao đổi tin nhiều VDK.

- Bit *REN*: cho phép nhận tin vào nối tiếp.
- Bit *TB8, RB8*: bit tin thứ 9 của các chế độ 2 và 3 (không dùng ở chế độ 0).
- Các bit *TI, RI*: các cờ ngắt khi đã truyền tin (*TI*) hay đã nhận tin (*RI*) xong.

9.1.2. Các chế độ trao đổi tin

Tuỳ giá trị các bit *SM0, SM1* và *SM2* của thanh ghi *SCON* mà ta ghi vào, ta có các chế độ truyền tin nối tiếp khác nhau.

1. Chế độ 0 (*SM0* = 0, *SM1* = 0, *SM2* = 0) hay đồng bộ 8 bit

Ở chế độ này, thanh ghi đệm *SBUF* là một thanh ghi dịch 8 bit, dịch các bit ra và vào ở chân *RxD* (bit có ý nghĩa nhỏ nhất được dịch đầu tiên). Lối ra *TxD* đưa ra các xung nhịp tốc độ baud. Việc phát dữ liệu được bắt đầu sau khi ghi dữ liệu vào thanh ghi *SBUF* và nhịp bởi xung nhịp của chu kỳ máy có mức âm ở thời điểm *S3P1* và trở lại mức cao ở thời điểm *S6P1*. Ở chế độ này, tin truyền hay nhận không có các bit Start, hay Stop. Tốc độ baud là cố định bằng 1/12 tần số của xung nhịp của VDK (*fosc*).

Việc thu dữ liệu được khởi động khi bit cho phép thu *REN* ở mức logic 1 và cờ ngắt thu *RI* ở mức logic 0. Khi bit *RI* được xoá bằng lệnh, các xung clock dịch bit được xuất hiện ra chân *TxD* (chứ không phải dữ liệu truyền) và dữ liệu được dịch vào chân *RxD* theo nhịp xung ra ở chân *TxD* (hình 9-3).

2. Chế độ 1 (*SM0* = 1, *SM1* = 0, *SM2* = 0) hay truyền không đồng bộ 8 bit

Ở chế độ này, truyền tin qua chân *TxD*, và nhận tin qua chân *RxD* nối tiếp không đồng bộ UART với 10 bit tin (một bit start mức logic 0, 8 bit dữ liệu dạng mã ASCII 7 bit, một bit chẵn/lẻ *P* và một bit stop (mức logic 1), có tốc độ thay đổi tuỳ máy phát tốc độ.

Bit chẵn/lẻ *P* này dùng để kiểm tra độ tin cậy của tin truyền: tính tổng các bit có mức logic 1 phát đi và tính tổng các bit 1 nhận được, nếu các tổng này (ở nơi phát và nơi nhận tin) là như nhau, các bit tin 1 không bị mất. Tin truyền đúng, không bị lỗi.

Chế độ 1 này là chế độ thông dụng nhất của trao đổi tin nối tiếp.

Ở chế độ này, dùng các xung đếm tràn của *Timer1* (*Timer0* dành cho định thời) hay *Timer2*:

+ *Nguồn xung từ đếm tràn của Timer1*: Ở chế độ này, các bit *TCLK, RCLK* = 0 (không dùng *Timer2*). Nếu *SMOD* (của thanh ghi *PCON*) = 1 thì tốc độ baud = tốc độ đếm tràn chia cho 32, còn khi *SMOD* = 0, tốc độ baud bằng tốc độ đếm tràn chia cho 16.

+ *Nguồn xung từ đếm tràn của Timer2 (của VDK 8052)*: muốn dùng *Timer2* này làm máy phát tốc độ baud, phải ghi các giá trị của các bit *TCLK, RCLK* (của thanh ghi *T2CON*) = 1 và tốc độ baud bằng 1/16 tốc độ đếm tràn của *Timer2*.

Có thể dùng cả hai *Timer* cho tốc độ baud, một cho tốc độ truyền dữ liệu (qua *TxD*), một cho tốc độ thu dữ liệu (qua *RxD*). Khi đó, cần lưu ý giá trị các bit *TCLK, RCLK* (bằng 0 khi dùng *Timer1* và bằng 1 khi dùng *Timer2*).

Khi truyền, tin được bắt đầu bởi việc ghi dữ liệu từ thanh ghi *A* vào thanh ghi đệm *SBUF*. Lệnh ghi *SBUF* cũng nạp 1 vào vị trí bit thứ 9 của thanh ghi dịch truyền và báo hiệu cho bộ điều khiển truyền có yêu cầu truyền. Việc truyền bắt đầu ở thời điểm *S1P1* của xung nhịp đưa ra sau bộ chia 16. Việc truyền bắt đầu khi có bit start ở *TxD* và kích hoạt tín hiệu */SEND* và sau đó là các bit dữ liệu. Khi bit 1 của stop bắt đầu nạp vào vị trí thứ 9, tín hiệu */SEND* mất đi và xác lập cờ ngắt *TI*.

Khi nhận, tin được bắt đầu bởi việc phát hiện dịch chuyển từ 1 tới 0 (của bit Start) ở chân *RxD*. Khi bit start dịch tới vị trí cuối cùng của thanh ghi dịch (là thanh ghi 9 bit ở chế độ 1), nó chỉ cho bộ điều khiển nhận nạp *SBUF* và *RB8* vào thanh ghi *T2CON* và xác lập cờ *RI* để yêu cầu VDK đọc dữ liệu từ *SBUF* vào thanh ghi *A*. Điều kiện này chỉ xảy ra khi cả hai điều kiện sau được thoả mãn:

1. *RI* = 0 và
2. Hoặc *SM2* = 0 hay bit stop nhận = 1.

3. Chế độ 2 (*SM0* = 0, *SM1* = 1, *SM2* = 0) và 3 (*SM0* = 1, *SM1* = 1, *SM2* = 0)

Chế độ này đều truyền và nhận nối tiếp không đồng bộ UART 11 bit (một bit start mức logic 0, 8 bit dữ liệu dạng mã ASCII, một bit thứ 9 (*RB8, TB8*) ghi bởi chương trình, 1 bit chẵn lẻ *P*, một

bit stop mức logic 1), có tốc độ baud không đổi (chế độ 2, bằng 1/32 hay 1/64 fosc) hay tốc độ thay đổi (chế độ 3, tùy tần số tràn của timer1 hay timer2).

Khi truyền, bit thứ 9 được ghi ở TB8, còn khi nhận, bit thứ 9 được ghi ở bit RB8 của thanh ghi T2CON.

Việc truyền và nhận cũng giống như ở chế độ 1 nhưng điều kiện thứ hai là hoặc SM2 = 0 hoặc bit dữ liệu thứ 9 = 1.

Các chế độ 2 và 3 này được dùng trong trao đổi tin nhiều VDK.

4. Chế độ trao đổi tin nhiều VDK

Khi bit SM2 = 1, nhiều VDK mắc nối tiếp có thể trao đổi tin nối tiếp giữa một VDK chủ (hay chính) dùng để điều khiển trao đổi tin với một trong nhiều VDK tớ (phụ) khác mắc theo kiểu nối song song theo đường dây nối tiếp. Ở chế độ trao đổi tin này, các chế độ 2 hay 3 được dùng và bit dữ liệu thứ 9 (được ghi vào RB8) sau đó đến bit stop.

Mỗi lần trao đổi tin, VDK chủ (chính) phải truyền 2 byte liên tiếp:

- Byte địa chỉ của VDK tớ có bit thứ 9 có giá trị 1 tới VDK tớ có địa chỉ chỉ rõ trong byte truyền.
- Byte dữ liệu cần truyền có bit thứ 9 có giá trị 0 tới VDK vừa truyền địa chỉ.

VDK tớ đã nhận địa chỉ của mình, sẽ xóa bit SM2 của nó về 0 và nhận dữ liệu truyền từ VDK chủ truyền tới.

Còn các VDK tớ khác không được gọi địa chỉ, sẽ xác lập bit SM2 của mình lên 1 và không nhận dữ liệu.

Bit SM2 không có ảnh hưởng ở chế độ 0, nhưng có thể dùng để kiểm tra bit stop ở chế độ 1.

Khi SM2 = 1, ngắt nhận không tác dụng cho tới khi nhận một bit stop.

9.1.3. Lập trình cho cổng nối tiếp

1. Trình tự lập trình cho trao đổi (truyền và nhận) một byte tin nối tiếp

Lập trình trao đổi tin qua cổng nối tiếp phải qua hai giai đoạn là khởi phát (ghi chế độ, ghi tốc độ baud, lệnh bắt đầu tạo xung baud) và trao đổi tin. Hình 9-3 giới thiệu các bước của truyền và nhận tin nối tiếp của cổng nối tiếp ở chế độ 0. Hai quá trình truyền (hình 9-3a) và nhận tin (hình 9-3b) có các điểm khác nhau như sau:

- *Quá trình truyền tin:* sau khi khởi động và ghi tin vào SBUF, tin được truyền. VDK phải chờ cho truyền tin xong bằng lệnh kiểm tra cờ TI xác lập lên 1. Khi cờ TI xác lập lên 1, ta phải xóa cờ truyền TI về 0 (bằng lệnh CLR TI) và có thể để có thể truyền một byte dữ liệu mới.

- *Quá trình nhận tin:* khi khởi phát ngoài ghi chế độ 1 (bit SM1 = 1) phải có bit REN = 1 và sau khi khởi phát, phải chờ xác lập cờ nhận RI = 1 (báo đã nhận vào SBUF xong) để đọc SBUF vào thanh ghi A và xóa cờ nhận RI về 0 (bằng lệnh CLR RI) để có thể nhận một byte tin vào tiếp.

2. Khởi phát

Quá trình khởi phát có nhiệm vụ là:

- Ghi các dữ liệu cho timer để phát tốc độ baud cần thiết cho trao đổi tin (xem ví dụ 9-1).
- Ghi chế độ truyền nối tiếp vào các bit SM0, SM1, SM2 của thanh ghi SCON.
- Ghi cho phép nhận vào bit REN của thanh ghi SCON.
- Ghi bit chặn lẽ P RB8 vào thanh ghi SCON nếu truyền ở chế độ 2 và 3.

3. Lập trình cho truyền một byte tin nối tiếp ở chế độ 0

Ở chế độ này, chỉ truyền 8 bit tin, có thể là 7 bit của mã ASCII và một bit chặn/lẻ, với tốc độ baud bằng 1/12 tần số của bộ dao động thạch anh (cỡ 1 micro giây). Do đó, nếu hai VDK 8051/52 trao đổi tin cho nhau, sẽ cùng một tốc độ baud, và được gọi là trao đổi tin đồng bộ.

Hình 9-4 giới thiệu các bước lập trình của cổng nối tiếp của VDK ở chế độ trao đổi tin M = 0.

Ví dụ 9-1 và 9-2 giới thiệu các chương trình truyền một byte tin ở chế độ 0 dùng bộ dao động thạch anh của của VDK.

4. Lập trình để nhận một byte tin nối tiếp ở chế độ 0

Ở chế độ này, chỉ nhận 8 bit tin, có thể là 7 bit của mã ASCII và một bit chặn/lẻ P, với tốc độ baud bằng 1/12 tần số của bộ dao động thạch anh (cỡ 1 micro giây). Do đó, nếu hai VDK 8051/52 trao đổi tin cho nhau, sẽ cùng một tốc độ baud, và được gọi là trao đổi tin đồng bộ.

1. Ghi chế độ 2 vào thanh ghi TMOD của Timer1 hoặc Timer2 2. Ghi giá trị tốc độ baud vào thanh ghi TH. 3. Ghi chế độ trao đổi tin vào thanh ghi SCON. 4. Khởi động Timer bằng TR = 1 5. Ghi byte dữ liệu vào SBUF 6. Chờ xác lập bit cờ TI 7. Xoá bit cờ TI về 0 8. Tiếp tục truyền byte dữ liệu mới từ bước 5.	1. Ghi chế độ 2 vào thanh ghi TMOD của Timer1 hoặc Timer2 2. Ghi giá trị tốc độ baud vào thanh ghi TH. 3. Ghi chế độ trao đổi tin vào thanh ghi SCON. 4. Khởi động Timer bằng TR = 1 5. Chờ xác lập bit cờ TI 6. Đọc byte dữ liệu từ SBUF vào A 7. Xoá bit cờ RI về 0 8. Tiếp tục nhận byte dữ liệu mới từ bước 5
--	--

a)

b)

Hình 9-3. Các bước truyền (a) và nhận tin nối tiếp (b)

<i>; Ví dụ 9-1: Chương trình truyền một byte tin</i>	<i>; Ví dụ 9-2: Chương trình nhận một byte tin</i>
<pre> ORG 00H MOV SCON, #00H ; chế độ nối tiếp 0 MOV A, #'A' ; gửi ký tự "A" BACK: MOV SBUF, A HERE: JNB TI, HERE ; chờ cờ báo truyền CLR TI ; xoá TI INC A ; tăng A CJNE A, #8BH, BACK SJMP \$; kết thúc chờ END ; kết thúc dịch </pre>	<pre> ORG 00H MOV SCON, #01H ; chế độ 0, cho nhận HERE: JNB RI, HERE ; chờ cờ báo nhận MOV A, SBUF ; đọc SBUF vào A MOV P1, A ; đưa A ra cổng P1 CLR RI ; xoá RI SJMP HERE ; trở lại truyền tiếp SJMP \$; kết thúc chờ END ; kết thúc dịch </pre>

Khi VĐK nhận tin, sau khi khởi phát, phải xác lập bit cho phép nhận REN = 1 của thanh ghi TCON. Quá trình nhận tin bắt đầu khi cờ nhận tin RI = 1, khi đó, tin nhận đã ghi vào thanh ghi đệm SBUF và tin nhận vào thanh ghi chứa A bởi lệnh MOV A, SBUF. Sau đó, cờ RI phải được xoá về 0 bởi lệnh CLR RI để có thể nhận tin tiếp theo.

Ví dụ 9-1 giới thiệu chương trình truyền một byte tin ở chế độ 0 dùng bộ dao động thạch anh của VĐK. Ví dụ 9-2 giới thiệu chương trình nhận một byte tin ở chế độ 0.

Chú ý: - Muốn truyền chỉ có 7 bit của thanh ghi A (7 bit mã ASCII) và bit chẵn/lẻ P; trước khi ghi dữ liệu từ thanh ghi A vào SBUF, ta phải đảo bit P ở bit thứ 7 của Acc bằng các lệnh sau:

MOV C, P; nạp bit P vào C

CPL C; lấy bù C vì có thêm bit P, tính chẵn lẻ của P thay đổi

MOV Acc.7, C; gửi C vào bit thứ 7 của A

- Muốn thu chỉ có 7 bit của thanh ghi A (7 bit mã ASCII), sau khi đọc dữ liệu từ SBUF vào thanh ghi A, ta phải đảo bit P ở bit thứ 7 của Acc bằng các lệnh trên.

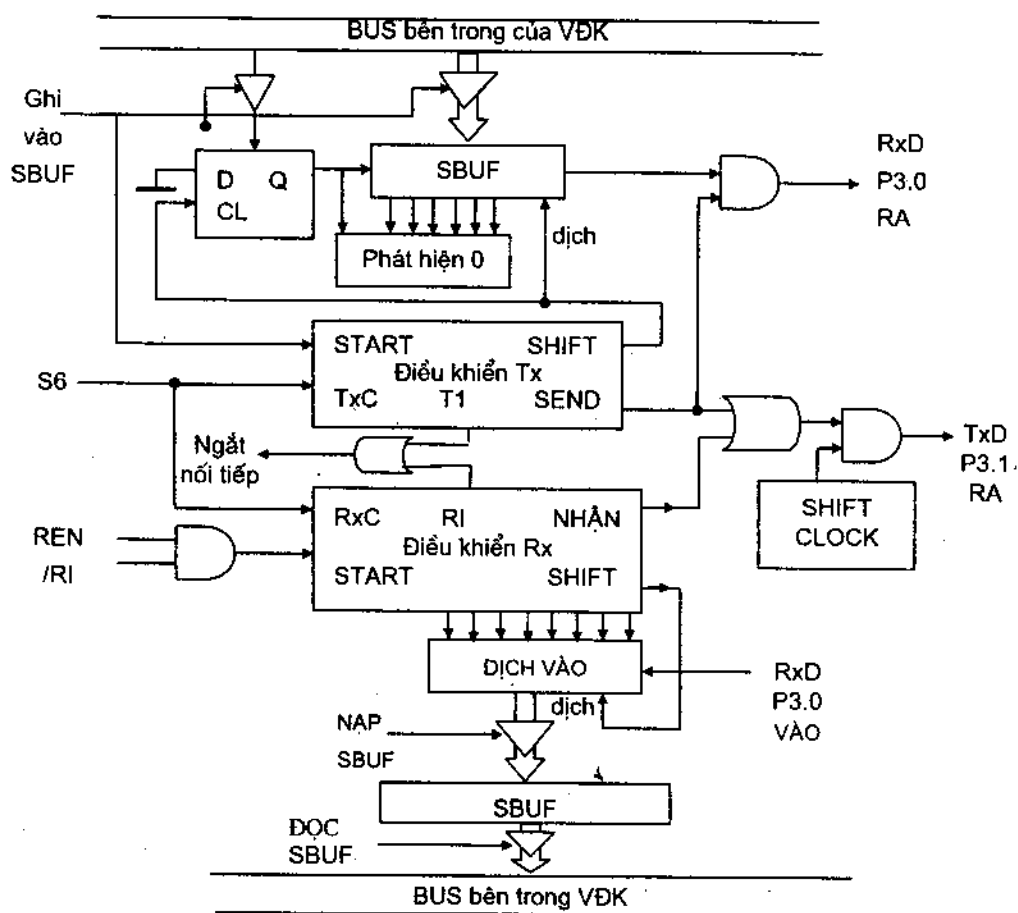
- Muốn xác lập bit SMOD = 1 của thanh ghi PCON, ta bổ sung lệnh:

MOV PCON, #80H

- Các chương trình truyền, nhận trên có thể đổi thành chương trình cho các chế độ 3 bằng thay giá trị nạp vào cho SCON là 00H (truyền) và 01H (nhận) và phần khởi phát thêm tốc độ baud.

- Các chương trình truyền, nhận trên có thể đổi thành chương trình cho các chế độ 0 và 2 với giá trị của SCON tương ứng và không cần khởi phát các Timer1 hay Timer2 vì chúng dùng xung baud của máy phát xung nhịp sau khi chia tần số (xem ở trên).

- Có thể dùng chương trình Pinnacle để kiểm tra việc truyền một hay nhiều byte tin với thực đơn VIEW/UART, còn không thể kiểm tra được chương trình nhận tin.



Hình 9.4. Cấu tạo cổng nối tiếp ở chế độ 0

9.2. PHÁT TỐC ĐỘ BAUD

Các bit tin của cổng nối tiếp được đưa ra truyền hay nhận vào theo nhịp của chuỗi xung tốc độ baud. Xung nhịp này lấy từ máy phát xung nhịp của VDK và có hai loại tốc độ:

- **Tốc độ cố định:** tần số được chia với hệ số chia cố định. Đó là ở các chế độ 0 và 2.
- **Tốc độ thay đổi:** tùy máy phát baud của bộ định thời Timer1 hay Timer2, ta có tốc độ khác nhau cho các chế độ 1 và 3.

Tuỳ chế độ trao đổi tin (0, 1, 2, 3) ta có tốc độ trao đổi tin khác nhau (hình 9-5) và các bảng 9-2 và 9-3.

Cụ thể là:

- **Tốc độ baud ở chế độ 0** (thanh ghi dịch 8 bit): có tần số bằng tần số xung nhịp của VDK chia cho 12 (hình 9-5a) theo công thức:

$$\text{Tốc độ baud} = 1/12 \times \text{tần số dao động fosc}$$

- **Tốc độ baud ở chế độ 2** (trao đổi tin UART 8 bit, không có bit chẵn, lẻ, tốc độ cố định): có tần số theo công thức:

$$\text{Tốc độ baud} = 1/64 \times 2^{\text{SMOD}} \times \text{tần số dao động fosc}$$

trong đó SMOD là bit thứ 7 của thanh ghi PCON và có một trong hai giá trị 0 hay 1 (hình 9-5b).

+ Nếu SMOD = 0 thì tốc độ baud là 1/64 tần số xung nhịp.

+ Nếu SMOD = 1 thì tốc độ baud là 1/32 tần số xung nhịp (tốc độ tăng gấp 2 so với trước).

<i>; Ví dụ 9-3: Truyền byte tin, chế độ 0,1</i> <pre> macro @truyen01 <1.chế độ?, 2.dữ liệu? MOV SCON, #1 ; chế độ nối tiếp với 11 MOV SBUF, A ; đưa ra A để truyền JNB TI, \$; chờ cờ báo truyền xong CLR TI ; xoá cờ TI ENDMAC ORG 00H mov a, #'A' ; chuyển 'A' vào thanh ghi a lap1: ; lặp lại @truyen01 0h ; chế độ 0, thanh ghi dịch, ; tốc độ bằng fosc/12 inc a ; tăng a cjne a, #'L', lap1 ; truyền tới 'L' mov a, #'0ah' ; truyền CR ; (tiếp ở cột bên phải) </pre>	<pre> @truyen01 0h (tiếp theo cột bên trái) mov a, #0dh @truyen01 0h ; truyền LF JB P3.2, \$; chờ ấn phím P3.2 (= 1) mov a, #'L' ; truyền 'L' lap2: @truyen01 40h ; chế độ 1, UART 8 bit, tốc ; độ thay đổi inc a ; tăng a cjne a, #'Z', lap2 ; truyền tới Z mov a, #'0ah' ; truyền CR @truyen01 0h mov a, #0dh ; truyền LF @truyen 0h sjmp \$; kết thúc chờ end ; kết thúc dịch </pre>
<i>; Ví dụ 9-4: nhan0; nhan byte tin, chế độ 0</i> <pre> macro @nhan01 <1.chế độ? MOV SCON, #1 ; chế độ nối tiếp với tham số #1 SETB REN ; cho phép nhận JNB RI, \$; chờ cờ báo nhận (nhận xong) MOV A, SBUF ; nhận vào A CLR RI ; xoá cờ nhận RI ENDMAC ORG 00H @nhan0 00h ; chế độ 0, thanh ghi dịch 8 bit, tốc độ không đổi sjmp \$; kết thúc chờ end ; kết thúc dịch </pre>	

- *Tốc độ baud ở chế độ 1, 3* (trao đổi tin UART 9 bit, có bit chẵn lẻ, tốc độ cố định) : có tốc độ baud (hình 9-5c) theo công thức sau:

+ Nếu SMOD = 0 ta có tốc độ baud = $1/32 \times$ tần số đếm tràn của Timer.

+ Nếu SMOD = 1 ta có tốc độ baud = $1/16 \times$ tần số đếm tràn của Timer.

Cụ thể: + Nếu dùng Timer1 ở chế độ 2 (tự nạp), ghi dữ liệu vào thanh ghi đếm TH1 và các bit TCLK = RCLK = 1 vào thanh ghi T2CON ta có:

$$\text{Tốc độ baud} = 1/32 \times 2^{\text{SMOD}} \times \text{tần số xung nhịp}/12 \times (256 - \text{TH1})$$

+ Nếu dùng Timer2 ở chế độ 2 (tự nạp), ghi dữ liệu vào thanh ghi đếm RCAP2H RCAP2L và các bit TCLK = RCLK = 1 vào thanh ghi T2CON, ta có:

$$\text{Tốc độ baud} = 1/32 \times 2^{\text{SMOD}} \times \text{tần số xung nhịp}/65536 - \text{RCAP2H} \text{ RCAP2L}$$

Với Timer1, 0 ở chế độ 2 (tự nạp lại) tần số xung nhịp khác nhau, các giá trị TH1 nhau như bảng 9-2.

Với Timer2, ta có bảng 9-3 với các giá trị của thanh ghi bất giữ RCAP2H, RCAP2L.

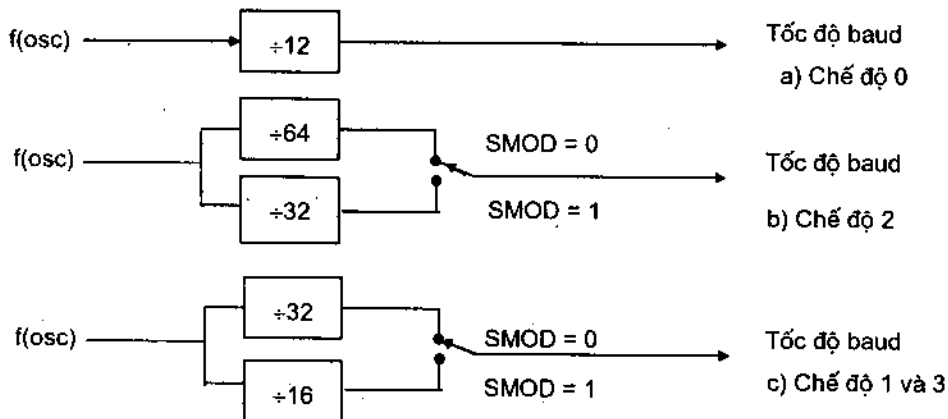
Ví dụ 9-5 giới thiệu lệnh gộp và chương trình trao đổi tin nối tiếp ở chế độ M = 1, dùng Timer1 làm máy phát tốc độ baud ở chế độ M = 2 (tự nạp lại từ TH1 vào TL1).

Ví dụ 9-6 giới thiệu lệnh gộp và chương trình trao đổi tin nối tiếp ở chế độ M = 1, dùng Timer2 làm máy phát tốc độ baud ở chế độ M = 2 (tự nạp lại RCAP2H, RCAP2L vào TH2, TL2).

Tất nhiên là dải tốc độ baud dùng Timer2 có dải rộng hơn Timer1.

Bảng 9-2. Bảng tốc độ baud cho Timer1,0

Tốc độ baud	Tần số xung nhịp	SMOD	C/T	MOD	TH1
62,3 kbaud	12,0 MHz	1	0	2	FFH
19,5 kbaud	11,059 MHz	1	0	2	FDH
9,6 kbaud	11,059 MHz	0	0	2	FDH
4,8 kbaud	11,059 MHz	0	0	2	FAH
2,4 kbaud	11,059 MHz	0	0	2	F4H
1,2 kbaud	11,059 MHz	0	0	2	F8H
110 baud	6,0 MHz	0	0	2	72H
110 baud	1,2 MHz	0	0	1	FE8H



Hình 9-5. Các tốc độ baud ở các chế độ

; Ví dụ 9-5: baud1 - Tốc độ baud với SMOD = 1	; Ví dụ 9-6: baud2 - Tốc độ baud với TIMER2
<pre> macro @baud1 ; <1.TH1?, 2. TL1; dùng ; timer1, chế độ 2 MOV TMOD, #20H ; chế độ 2 MOV TH1, #1 ; ghi thành ghi TH1 MOV TL1, #2 ; ghi thành ghi TL1 SETB TR1 ; bắt đầu đếm endmac macro @TRA <1.CHE DO?> MOV SCON, #1 ; chế độ nối tiếp MOV SBUF, A ; đưa A ra truyền JNB TI, \$; chờ cờ báo truyền CLR TI ; xóa TI endmac org 00h mov a, #'a' ; ghi mã chữ A @BD1 high(-3), low(-3); chế độ 2; TIMER1 lap1: @TRA 40h ; truyền UART, chế độ 1 inc a ; tăng A cjne a, #'z', lap1; so với mã của chữ z sjmp \$ end </pre>	<pre> macro @Timer2 ; dùng Timer1, chế độ 2 MOV T2CON, #00h MOV TH2, #1 ; ghi thành ghi TH2 MOV TL2, #2 ; ghi thành ghi TL2 MOV RCAP2H, #3 ; ghi thành ghi TH2 MOV RCAP2L, #4 ; ghi thành ghi TL2 SETB TR2 ; bắt đầu đếm endmac macro @TRA <1.CHE DO?> MOV SCON, #1 ; chế độ nối tiếp MOV SBUF, A ; đưa A ra truyền JNB TI, \$; chờ cờ báo truyền CLR TI ; xóa TI endmac org 00h mov a, #'0' ; ghi mã số 0 @baud2 high(-13), low(-13), high(-13), low(-13) ; tốc độ 9600 baud lap1: @TRA 40h ; truyền UART, chế độ 1 inc a ; tăng A cjne a, #'9', lap1 ; so với mã của số 9 (39H) sjmp \$ end </pre>

Bảng 9-3. Bảng tốc độ và giá trị của RCAP2

Tốc độ	12 MHz	16 MHz	20 MHz	11,0592
110	-3094	-4545	-5682	-3142
150	-2500	-3333	-4167	-2304
300	-1250	-1667	-2083	-1152
600	-625	-833	-1042	-576
1200	-312	-417	-521	-288
2400	-156	-208	-260	-144
4800	-78	-104	-130	-72
9600	-39	-52	-65	-36
19200		--	-26	-33
38400		13	-16	-9

9.3. TRAO ĐỔI TIN NỐI TIẾP Ở CHẾ ĐỘ 1

Chúng ta đã xét trao đổi tin nối tiếp đồng bộ, 8 bit ở chế độ 0 (không có các bit Start ở đầu byte tin và bit Stop ở cuối byte tin) với tốc độ baud không đổi, cỡ 12 Mbaud.

Ở đây, chúng ta xét chế độ 1, trao đổi tin không đồng bộ UART (Universal Asynchrone Transceiver and Receiver) với:

- 10 bit tin, gồm 1 bit Start (bắt đầu), 8 bit tin và một bit Stop (dừng).
- Tốc độ thay đổi nhờ lập trình cho bộ định thời Timer1 hay Timer2.

9.3.1. Thủ tục trao đổi tin không đồng bộ UART

Trao đổi tin đồng bộ dùng thanh ghi dịch (chế độ 0) là loại trao đổi tin nối tiếp đơn giản, nhanh nhưng không tin cậy vì có nhiễu nhiễu của môi trường xung quanh và không thể truyền tin đi xa. Do đó, người ta quy định trao đổi tin nối tiếp không đồng bộ (UART- Universal Asynchrone Receive and Transmiste) để khắc phục những nhược điểm của trao đổi tin đồng bộ.

Trao đổi tin không đồng bộ quy định về phương pháp truyền tin, dạng tin, các tín hiệu và cách trao đổi tin; mức tín hiệu của tin. Ở đây, chúng ta cần chú ý tới cách trao đổi tin UART trên cho VĐK.

1. Phương pháp truyền tin

Trong trao đổi tin nối tiếp, về chiều truyền tin, có thể có:

+ *Trao đổi tin đơn công*: là trao đổi tin theo một chiều, trên một đường dây từ VĐK hay máy vi tính tới thiết bị thu hoặc ngược lại, từ thiết bị phát tin tới VĐK hay máy vi tính, ví dụ, từ máy vi tính ra máy in hay từ bàn phím vào máy vi tính. Loại trao đổi tin này chỉ dùng khi thiết bị ngoài chỉ có thể trao đổi tin một chiều (hoặc thu hoặc nhận tin) với VĐK hay máy vi tính.

+ *Trao đổi tin bán song công (một chiều, mỗi thời điểm truyền theo một chiều)*

Đó là trường hợp giữa thiết bị phát và thu tin chỉ có một đường dây truyền tin, nhưng truyền theo hai chiều, lúc đưa tin ra, lúc đưa tin vào và một đường dây đất.

+ *Trao đổi tin song công hoàn toàn (hai chiều)*

Đó là trường hợp giữa thiết bị phát và thu tin có hai đường dây truyền tin, một đường dành cho tin phát và một đường dành cho tin thu và đồng thời truyền theo hai chiều, cả phát và thu tin và một đường dây đất.

2. Khung tin

Tin truyền, ngoài byte dữ liệu (8 bit, 9 bit, 10 bit) còn có thêm hai bit:

- *Bit Start (bắt đầu)*: có mức logic 0, được gửi đi đầu tiên, để báo cho máy thu biết bắt đầu của một byte tin. Máy thu nhận biết được bit tin này sẽ chuẩn bị thu tin và phát xung nhịp nhận RxC.

- **Bit Stop (dừng):** có mức logic 1, được gửi đi cuối cùng, để báo cho máy thu biết kết thúc của một byte tin. Máy thu nhận biết được bit tin này sẽ kết thúc việc thu tin và ngừng phát xung nhịp nhận RxC.

- **Bit chẵn hay lẻ:** để kiểm tra độ tin cậy của truyền tin, người ta tính tổng các giá trị 1 và cả giá trị của bit chẵn ở nơi phát và gửi đi trong bit chẵn lẻ (bit P của thanh ghi trạng thái của VĐK). Ở nơi thu, cũng tính giá trị tương tự và so sánh giá trị gửi đi và giá trị thu. Nếu hai giá trị bằng nhau, sự trao đổi tin là đúng, tin cậy, nếu sai, phát hiện lỗi và yêu cầu truyền lại.

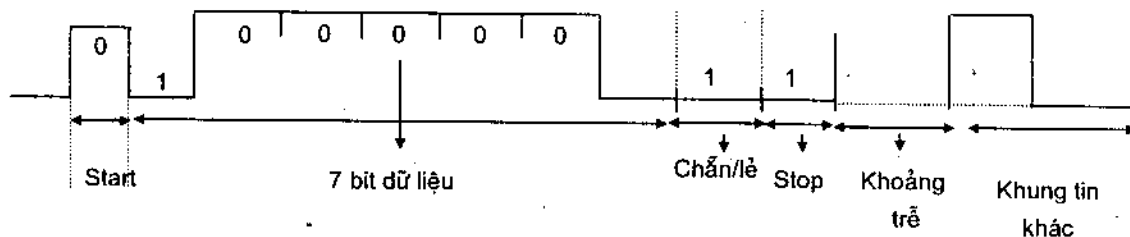
Như vậy, một byte tin (có thể là mã ASCII 7 bit hay 9 hoặc 10 bit và một bit chỉ đặc tính chẵn hay lẻ P của các bit 1 và cả bit chẵn/lẻ) được đóng trong một khung tin, có các bit Start ở đầu và bit Stop ở cuối (hình 9-6).

Nhờ khung tin này, việc trao đổi tin tuy tốn thời gian nhưng tin cậy, không bị mất tin.

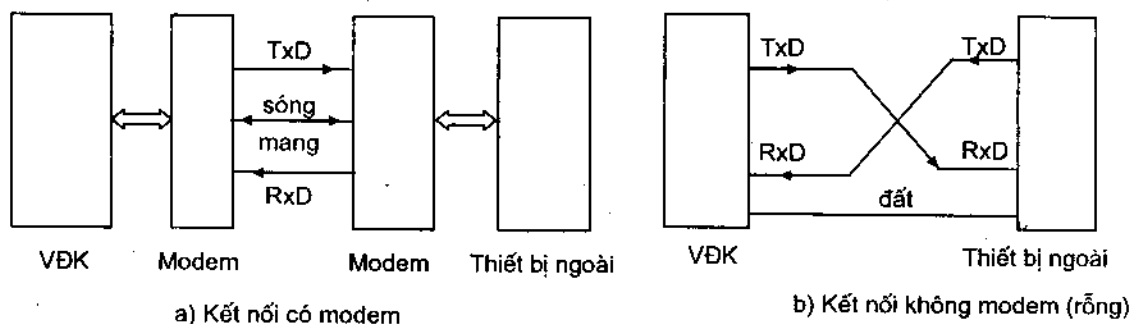
3. Modem và các tín hiệu hội thoại

Giữa hai thiết bị trao đổi tin còn được ghép nối với nhau qua modem (điều chế và giải điều chế với một sóng mang cao tần) để chống nhiễu và truyền tin đi xa trên đường dây điện thoại (hình 9-7) với mức tín hiệu cao (theo chuẩn RS232A là ± 12 V) để chống nhiễu. Giữa thiết bị trao đổi tin và modem, có các tín hiệu hội thoại như hình 9-8a và bảng 9-4 giữa nguồn phát tin và nhận tin. Nếu không có modem (modem rỗng, các tín hiệu hội thoại được nối như hình 9-8b).

Giữa VĐK hay máy vi tính nối với thiết bị ngoài (qua modem hay không có modem) đều qua một ổ cắm DB 25 chân (với máy in Centronic) hay ổ cắm DB-9 (9 chân) với số chân nối các tín hiệu như ở bảng 9-4.



Hình 9-6. Các bit của một khung tin UART với 7 bit dữ liệu

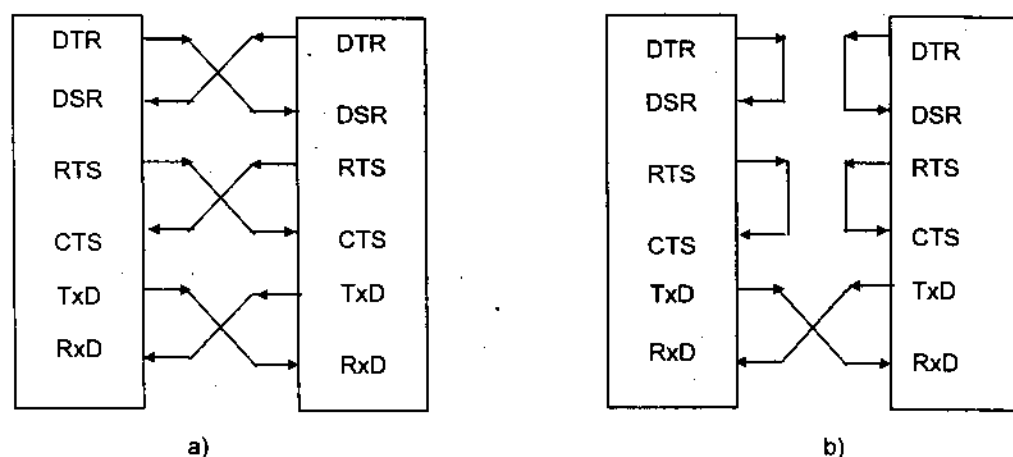


Hình 9-7. Kết nối VĐK với thiết bị ngoài

4. Đường dây và mức tín hiệu

Chuẩn RS232C mắc với đường dây điện thoại đường dây điện thoại (ba dây Tx, Rx, đất) có mức tín hiệu -12 V (logic 1) và $+12$ V (logic 0). Các mức tín hiệu này có ưu điểm là chênh lệch nhau lớn (tới 24 V), nên phân biệt rõ ràng với mức tín hiệu nhiễu của môi trường tác động lên

đường dây. Nhưng ghép nối các thiết bị số (VĐK, vi xử lý) với mức TTL (0-5 V) phải qua một bộ chuyển mức TTL-RS232 bằng vi mạch, ví dụ MC 1488, 1489, MAX 232 (hình 9-9) hay MAX 233.



Hình 9-8. Kết nối các tín hiệu hội thoại của modem

Bảng 9-4. Các tín hiệu của RS232

Chân số	Tên tín hiệu	Ý nghĩa
1	Data Carrier Detect (DCD)	Tách tín hiệu mang dữ liệu
2	Received Data (RxD)	Dữ liệu nhận
3	Transmitted Data (TxD)	Dữ liệu truyền
4	Data Terminal Ready (DTR)	Thiết bị đầu cuối dữ liệu sẵn sàng
5	Signal Ground (GND)	Đất của tín hiệu
6	Data Set Ready (DSR)	Dữ liệu sẵn sàng
7	Request To Send (RTS)	Yêu cầu gửi
8	Clear To Send (CTS)	Xóa để gửi
9	Ring Indicator (RI)	Báo chuông

Ngoài chuẩn RS232, người ta còn dùng cổng USB (Universal Serial Bus) với các chuẩn RS423 (không cân bằng và RS422 cân bằng (hình 9-10) và bảng 9-5 (xem chi tiết ở tập II).

Từ bảng 9-5 ta thấy: Các chuẩn cân bằng (dùng khuếch đại thuật toán) tuy phải dùng hai đường dây cho mỗi tín hiệu truyền hay nhận (TD+, TD-) nhưng có ưu điểm là mức logic TTL, tính chống nhiễu cao, tốc độ trao đổi tin lớn, khoảng cách truyền tin lớn.

9.3.2. Đặc điểm chung của trao đổi tin ở chế độ 1

Ở chế độ 1 này, sự trao đổi tin có đặc điểm sau:

- Trao đổi tin theo khung tin không đồng bộ UART (có thêm các bit Start (logic 0) và Stop (logic 1)).
- Tốc độ trao đổi tin thay đổi theo tốc độ baud của máy phát nhịp là Timer1 hay Timer2 bằng lập trình.

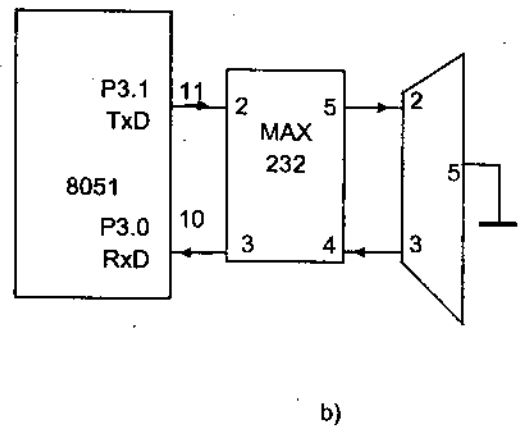
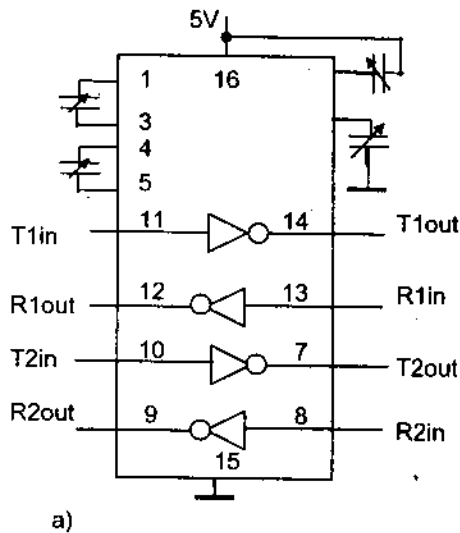
- Cũng có sự kiểm tra trạng thái cho phép truyền TI = 1 hay đã truyền xong byte dữ liệu trước (SBUF rỗng) hoặc cho phép nhận RI = 1 hay đã nhận tin xong vào SBUF (SBUF đầy).

So với chế độ trao đổi tin 0, chế độ trao đổi tin này có những đặc điểm chung sau:

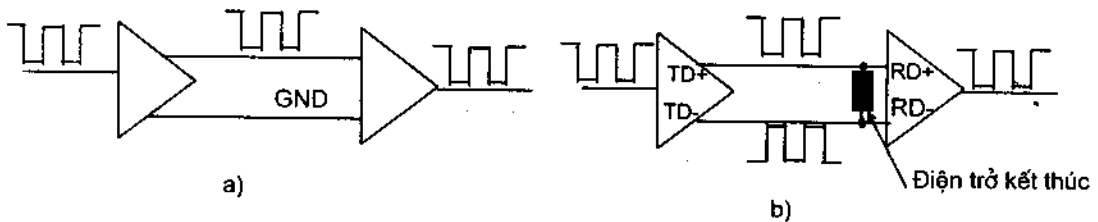
- Tốc độ trao đổi tin chậm hơn vì có các bit Start, Stop.
- Tốc độ trao đổi tin có thể thay đổi được nhờ lập trình tốc độ baud.
- Chuẩn RS232C được thông dụng trong trao đổi tin giữa một VĐK (hay vi xử lý) và thiết bị ngoài. Hiện nay, trong các thiết bị hiện đại, người ta dùng cổng trao đổi tin theo USB thay cho RS232C.

Bảng 9-5. Đặc điểm của các chuẩn RS

EIA	RS-232C	RS-423A	RS-422A	RS-485
ITU	V.28	V.10/X26	V.11/X27	
Tốc độ truyền số liệu	20 kbps	300 kbps	10 Mbps	10 Mbps
Khoảng cách cực đại	17-20 m	1200 m	1200 m	1200 m
Loại nối tải	không cân bằng	không cân bằng vi sai	cân bằng vi sai	cân bằng vi sai
Số cặp thu phát	1	1	1	32
Điện thế điều khiển	$\pm 3\text{ V} \div \pm 15\text{ V}$	$\pm 2\text{ V} \div \pm 6\text{ V}$	$\pm 2\text{ V} \div \pm 5\text{ V}$	$\pm 2\text{ V} \div \pm 5\text{ V}$
Số đường dây cho mỗi tín hiệu	1	2	2	2



Hình 9-9. Cấu trúc MAX232 (a) và ghép nối không modem với 8051



Hình 9-10. Đường truyền ghép nối không cân bằng (a) và cân bằng (b)

Ví dụ 9-7: Chương trình truyền 1 byte dùng Timer1

```

ORG 00H
MOV TMOD, #20H ; timer1
MOV TH1, #-3H ; 9600 baud
MOV SCON, #70H ; chế độ 1
SETB TR1 ; bật Timer1
MOV A, #41H ; gửi mã ký tự A
LOOP: MOV C, P ; gửi P vào C
MOV TB8, C ; gửi C vào TB8

```

```

MOV SBUF, A ; gửi A vào SBUF
JNB TI, $ ; chờ truyền xong
CLR TI ; xóa TI
INC A ; tăng A
CJNE A, #4AH, LOOP; về LOOP
CLR TR1 ; xóa TR1
SJMP $ ; kết thúc chờ
END ; kết thúc dịch

```


; Ví dụ 9-8: Chương trình truyền các byte tin dùng Timer2

ORG 00H	SETB TR2 ; bật Timer2
MOV RCAP2H, #0FFH ; timer2	MOV SBUF, A ; gửi A vào SBUF
MOV RCAP2L, #-3H ; 9600 baud	JNB TI, \$; chờ TI
MOV SCON, #50H ; chế độ 1	CLR TI ; xoá TI
MOV T2CON, #10H ; khởi động timer2	INC A ; tăng A
MOV A, #'A' ; gửi ký tự 'A' vào A	CJNE A, #70H, LOOP; trở về LOOP
LOOP: MOV C, P ; gửi P vào C	SJMP \$; kết thúc chờ
MOV TB8, C ; gửi C vào TB8	END ; kết thúc dịch

9.3.3. Truyền byte tin

1. Truyền byte tin

Truyền một hay nhiều byte tin ở chế độ 1 cũng tương tự chế độ 0, chỉ có khác là, trước khi trao đổi, phải lập trình cho máy phát tốc độ baud.

- Bit TI (bit DO) của thanh ghi SCON phải xác lập lên 1 để sẵn sàng truyền tin.
- Phải chờ bit TI xác lập lên 1 (bằng lệnh JNB TI, \$) để biết tin cũ đã truyền xong.
- Xóa bit TI = 0 (bằng lệnh CLR TI) để tiếp tục truyền byte tin mới.
- Ghi tin cần truyền vào thanh ghi thanh ghi chứa A.
- Ghi tin từ thanh ghi thanh ghi chứa A vào thanh ghi đệm dữ liệu SBUF. Tin ở SBUF sẽ tự động dịch nối tiếp từ SBUF ra chân TxD theo nhịp của tốc độ baud.

Ví dụ 9-7 giới thiệu chương trình truyền 10 ký tự từ A tới J với tốc độ 9600bps dùng timer1.

Ví dụ 9-8 giới thiệu chương trình truyền các ký tự từ A tới ký tự có mã ASCII là #70H với tốc độ 9600bps dùng timer2 (với bit TCLK của thanh ghi SCON = 1).

; Ví dụ 9-9: Chương trình truyền nhiều byte tin, chữ 'YES'

ORG 00H	MOV SBUF, A ; gửi A vào SBUF
MOV TMOD, #20H ; timer1	JNB TI, \$; chờ truyền xong
MOV TH1, #-3 ; 9600 baud	CLR TI ; xoá TI
MOV SCON, #70H ; chế độ 1,	MOV A, #'S' ; gửi mã ký tự 'S'
SETB TR1 ; bật Timer1	MOV SBUF, A ; gửi A vào SBUF
MOV A, #'Y' ; gửi mã ký tự 'Y'	JNB TI, \$; chờ truyền xong
MOV SBUF, A ; gửi A vào SBUF	CLR TI ; xoá TI
JNB TI, \$; chờ truyền xong	CLR TR1 ; xoá TR1
CLR TI ; xoá TI	SJMP \$; kết thúc chờ
MOV A, #'E' ; gửi mã ký tự 'E'	END ; kết thúc dịch

; Ví dụ 9-10: Chương trình truyền chuỗi tin 'Bài thí nghiệm 9'

ORG 00H	JZ STOP ; tới Stop khi A = 0
MOV RCAP2H, #0FFH ; timer2	MOV SBUF, A ; gửi A vào SBUF
MOV RCAP2L, #-5H ; 9600 baud	JNB TI, \$; chờ TI
MOV SCON, #52H ; chế độ 1	CLR TI ; xoá TI
MOV T2CON, #14 ; khởi động timer2	INC DPTR ; tăng địa chỉ nhớ
MOV DPTR, #TBAO ; gán địa chỉ nhớ	JMP SEND ; trở về SEND
SETB TR2 ; bật Timer2	STOP: SJMP \$; kết thúc chờ
SEND: CLR A ; xoá A	TBAO: DB 'BAI THI NGHIEM 9', 0
MOVC A, @A+DPTR ; đọc bảng nhớ	END ; kết thúc dịch

- Theo nhiều lần từng byte tin, ví dụ 9-9 truyền chữ 'YES', truyền các ký tự 'Y', 'E', 'S'. Cách truyền này đơn giản nhưng tốn lệnh và thời gian thực hiện.

- Theo một chuỗi, nhiều byte tin ghi nhớ trong bộ nhớ kết thúc bởi số 0. Ví dụ 9-10 truyền thông báo 'Bài thí nghiệm 9' dùng lệnh đọc chuỗi tin MOV A, @A+DPTR, trong đó DPTR là địa chỉ vùng nhớ chứa chuỗi tin. Chương trình truyền này ngắn, nhưng chuỗi ghi trong các ô nhớ liên tiếp.

Nếu cho chạy cả hai chương trình trên với chương trình mô phỏng Pinnacle và quan sát trên thực đơn VIEW\UART, ta quan sát được các mã ASCII đã truyền.

9.3.4. Nhận byte tin

1. Nhận byte tin

Nhận một hay nhiều byte tin ở chế độ 1 cũng tương tự chế độ 0, chỉ có khác là, trước khi trao đổi, phải lập trình cho máy phát tốc độ baud.

So với truyền một byte tin, việc nhận một byte tin có những điểm khác là:

- Bit REN (bit D4) của thanh ghi SCON phải xác lập lên 1 để cho phép nhận tin.

- Bit RI (bit D0) của thanh ghi SCON phải xác lập lên 1 để sẵn sàng nhận tin.

- Phải chờ bit RI xác lập lên 1 (bằng lệnh JNB RI, \$) để biết đã nhận xong tin (lần đầu không phải chờ).

- Tin từ chân RxD (P3.0) sẽ tự động dịch nối tiếp vào thanh ghi đệm dữ liệu vào SBUF theo nhịp của tốc độ baud..

- Ghi tin từ thanh ghi đệm dữ liệu SBUF vào thanh ghi chứa A.

- Xóa bit RI = 0 (bằng lệnh CLR RI) để tiếp tục nhận byte tin mới.

Ví dụ 9-11a, b giới thiệu chương trình nhận một byte tin với tốc độ 9600bps dùng Timer1.

Chú ý: - Chúng ta không thể dùng thực đơn VIEW\UART của chương trình mô phỏng Pinnacle để kiểm tra quá trình nhận tin này mà phải kết hợp với chương trình truyền tin.

- Chương trình 9-11b nhận một byte tin dùng Timer2, trong thanh ghi T2CON phải có bit RCLK = 1 (bit D5) thay cho TCLK = 1 (bit D4) ở chương trình truyền một byte tin.

<i>; Ví dụ 9-11a: Nhận một byte tin dùng Timer1</i>	<i>; Ví dụ 9-11b: Nhận một byte dùng Timer2</i>
ORG 00H	ORG 00H
MOV TMOD, #20H ; timer1	MOV RCAP2H, #0FFH ; timer2
MOV TH1, #-3 ; 9600 baud	MOV RCAP2L, #-3H ; 9600-baud
MOV SCON, #70H ; chế độ 1	MOV SCON, #42H ; chế độ 1
SETB TR1 ; bật Timer1	MOV T2CON, #20H ; timer2
GETCHAR: JNB RI, \$; chờ nhận xong	SETB TR2 ; bật Timer2
CLR RI ; xóa RI	JNB RI, \$; chờ RI
SJMP \$; kết thúc chờ	CLR RI ; xóa RI
END ; kết thúc dịch	MOV A, SBUF ; gửi SBUF vào A
	SJMP \$; kết thúc chờ
	END ; kết thúc dịch

2. Nhận chuỗi nhiều byte tin

Khác với truyền chuỗi nhiều byte tin, chương trình nhận chuỗi nhiều byte tin có đặc điểm là:

- Bị động nhận, số lượng byte tin do thiết bị truyền quyết định. Tin ngừng nhận khi thiết bị truyền chủ động ngừng.

- Tin nhận vào A phải lưu vào các ô nhớ trong (địa chỉ gián tiếp thanh ghi R0 hay R1) hay nhớ ngoài (địa chỉ gián tiếp theo con trỏ địa chỉ DPTR).

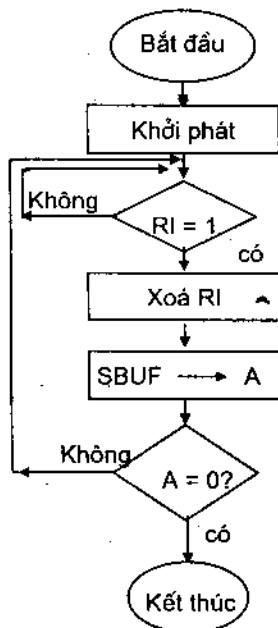
Ví dụ 9-12a, b giới thiệu chương trình nhận chuỗi tin dùng timer1, timer2 làm máy phát tốc độ baud với giải thuật ở hình 9-11.

9.4. TRAO ĐỔI TIN NỐI TIẾP TRONG HỆ NHIỀU VĐK Ở CHẾ ĐỘ 2, 3

9.4.1. Đặc điểm chung

Trao đổi tin nối tiếp ở chế độ 2 và 3 này có đặc điểm chung sau:

- Trao đổi tin 9 bit (9 bit tin mã ASCII+1 bit chẵn/lẻ) trong thanh ghi đệm dữ liệu SBUF và bit thứ 8 (RB8 hay TB8) trong thanh ghi SCON.
- Tốc độ baud có thể cố định bằng $1/32 f_{osc}$ (khi SMOD = 1 của PCON) hay bằng $1/64 f_{osc}$ (SMOD = 0 của PCON) của máy phát xung nhịp của VĐK (chế độ 2) hay thay đổi theo tần số đếm tràn của Timer1 hay Timer2 (chế độ 3).
- Có thể trao đổi với một VĐK (khi SM2 = 0) hay nhiều VĐK (khi SM2 = 1).



Hình 9-11. Giải thuật của chương trình nhận chuỗi tin

; Ví dụ 9-12a: Chương trình nhận nối tiếp 1 chuỗi tin

```

ORG 00H
ACALL INIT      ; gọi khởi phát
MOV DPTR, #ADDR ; gán địa chỉ ghi chuỗi
BACK: ACALL GETCHAR ; gọi trình nhận 1 byte
JZ STOP        ; nhảy tới stop nếu A = 0
MOV @DPTR, A    ; ghi tin vào ô nhớ
INC DPTR        ; tăng địa chỉ nhớ
SJMP BACK       ; trở về nhận tiếp
STOP: SJMP $     ; kết thúc chờ

INIT: MOV TMOD, #20H ; chế độ M = 2 ; timer1
MOV TH1, #3        ; nạp TH1 cho 9600 baud
MOV SCON, #50H     ; chế độ 1, cho phép nhận
SETB TR1           ; khởi động timer1
RET               ; trở về chương trình gọi

GETCHAR:
JNB RI, $         ; chờ cờ báo nhận
CLR RI           ; xoá RI
MOV A, SBUF       ; đọc tin vào A
RET              ; trở lại chương trình gọi
END              ; kết thúc dịch
    
```

; Ví dụ 9-12b: NCT2 - Chương trình nhận chuỗi tin dùng Timer2

ORG 00H		RECEIV: JNB RI, \$	; chờ RI
MOV DPTR, #200H	; địa chỉ nhớ	CLR RI	; xoá RI
MOV RCAP2H, #0FFH	; timer2	MOV A, SBUF	; gửi SBUF vào A
MOV RCAP2L, #-3H	; 9600 baud	MOVX @DPTR, A	
MOV SCON, #42H	; chế độ 1	INC DPTR	; tăng địa chỉ nhớ
MOV T2CON, #20H	; timer2	JMP RECEIV	; trở về RECEIV
SETB TR2	; bật timer2	SJMP \$	; kết thúc chờ
		END	; kết thúc dịch

9.4.2. Trao đổi tin với một VĐK ở chế độ 2 và 3

Ta có thể dùng lệnh gộp @TRA hay @NHA để truyền hay nhận tin nối tiếp ở chế độ 2 (tốc độ baud không đổi) hay chế độ 3 (tốc độ baud thay đổi theo sự đếm tràn của Timer1 hay Timer2).

Nhưng trước khi truyền, ta phải:

- Ghi bit chẵn/lẻ P vào bit thứ 8 của thanh ghi chứa A (Acc.7).

- Ghi bit TB8 vào thanh ghi SCON.

Sau khi nhận tin, phải:

- Đọc bit chẵn/lẻ P từ Acc.7 sang P của PSW.

- Đọc bit RB8 từ thanh ghi SCON.

Trong hệ VDK mắc nối tiếp nhiều VDK (hình 9-12), tin có thể trao đổi nối tiếp với nhau qua một đường dây nối tiếp chung và dưới sự điều khiển của một VDK chủ, đóng vai trò chủ động trong trao đổi tin.

9.4.3. Trao đổi tin nối tiếp trong hệ nhiều VDK ở các chế độ 2 và 3

Tổ chức trao đổi tin này có những đặc điểm sau:

- Chế độ trao đổi tin là 2 (bit SM2 = 1 của thanh ghi SCON).

- Các bit TB8 phải là 1 khi truyền và RB8 = 1 khi nhận.

- Ở chế độ 2, tốc độ baud không đổi, dùng xung nhịp của bộ dao động thạch anh của VDK với tần số là 1/32 hay 1/64 tần số của thạch anh.

- Ở chế độ 3, tốc độ baud thay đổi được, bằng lập trình, dùng timer1 ở chế độ tự nạp lại với công thức là:

$$\text{Tốc độ baud} = \frac{2\text{SMOD}}{32} \times \frac{\text{fosc}}{12d \times [256 - (\text{TH1})]}$$

với fosc = 11,059 MHz
cho VDK 8051/8052

- Ở chế độ 3, tốc độ baud thay đổi được, bằng lập trình, dùng timer2 ở chế độ bất giữ, với công thức là:

$$\text{Tốc độ baud} = \frac{1}{32} \times \frac{\text{fosc}}{[65536 - (\text{RCAP2H}, \text{RCAP2L})]}$$

- Trao đổi tin giữa một VDK chủ với một VDK tớ trong hệ theo trình tự sau:

+ VDK chủ đưa địa chỉ của VDK tớ với dấu hiệu là bit thứ 9 (RB8 hay TB8) = 1, còn byte dữ liệu là 0.

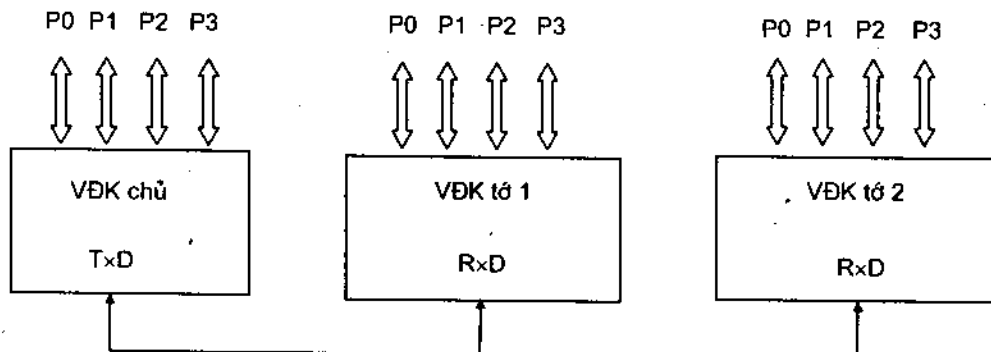
+ VDK tớ nhận tin về địa chỉ riêng của mình, xóa bit SM2 = 0 của mình và chuẩn bị nhận các byte dữ liệu theo sau với bit thứ 9 (TB8, RB8 = 0).

+ Tin về dữ liệu chỉ trao đổi giữa VDK chủ (có bit SM2 = 1) với VDK tớ (có bit SM2 = 0), còn các VDK tớ khác không nhận địa chỉ vẫn có bit SM2 = 1 nên không nhận tin về dữ liệu.

- Bằng lập trình, bit Stop của tin nhận sẽ ghi vào bit RB8 của VDK nhận tin và tạo ra một ngắt chương trình và VDK đó trao đổi tin với VDK chủ.

- Khi kết thúc trao đổi tin, bit SM2 của VDK tớ phải được xác lập lên 1 để trao đổi tin lần sau.

- Các VDK không được gọi địa chỉ vẫn thực hiện chương trình của riêng chúng.



Hình 9-12. Hệ trao đổi tin nối tiếp nhiều VDK

9.5. TRAO ĐỔI TIN NỐI TIẾP GIỮA VDK VỚI MÁY VI TÍNH PC

9.5.1. Yêu cầu và giải pháp trao đổi tin giữa máy vi tính và VDK

1. Yêu cầu trao đổi tin

Giữa máy vi tính và VDK cần trao đổi tin với nhau vì:

- VDK cần nạp chương trình đã soạn thảo, biên dịch và chạy thử từ bộ nhớ của máy vi tính PC-IBM sang bộ nhớ PROM bên trong của VDK hay RAM ngoài để thực hiện.
- Máy vi tính PC-IBM cần truyền lệnh, dữ liệu cho bộ nhớ RAM bên ngoài của VDK để thực hiện và sử dụng.
- VDK cần gửi dữ liệu hay các kết quả đo, điều khiển sang máy vi tính để hiển thị, xử lý và lưu trữ trên các ổ đĩa mềm, ổ đĩa cứng.

2. Giải pháp

Tổ chức trao đổi tin nối tiếp cho tiết kiệm đường dây giữa máy vi tính và VDK như sau:

- Máy vi tính và VDK nối ghép với nhau qua các ổ cắm và đường dây nối tiếp DB9 để trao đổi tin không đồng bộ UART theo chuẩn RS232 không hay có modem. Ngoài ra, giữa máy tính và VDK còn trao đổi tin theo cổng USB (xem tập II).

- Máy vi tính phải có chương trình trao đổi tin (truyền và nhận) với thực đơn trên màn hình có ba chức năng truyền, nhận và thoát. Có thể kể các chương trình TERM.EXE, TERM95.EXE của chương trình NC (Norton Commander), chương trình RAM.EXE, chương trình Serial.EXE v.v... tự viết cho máy vi tính PC.

- VDK phải có chương trình trao đổi tin (truyền và nhận) để nhận tin nối tiếp về chương trình, lệnh, dữ liệu từ máy vi tính PC để ghi vào bộ nhớ chương trình hay vào bộ nhớ RAM ngoài để thực hiện. Ngược lại VDK còn truyền sang máy vi tính các kết quả đo, điều khiển của hệ VDK về dữ liệu và trạng thái của hệ.

Hai chương trình trao đổi tin của máy vi tính PC và của VDK, tuy dưới dạng ngôn ngữ khác nhau (dạng C giống nhau, hợp ngữ khác nhau) nhưng đều có đặc điểm chung là đều ở chế độ hội thoại (bắt tay hay kiểm tra trạng thái sẵn sàng) hay ngắt chương trình (xem chương 10).

9.5.2. Kết nối PC-VDK

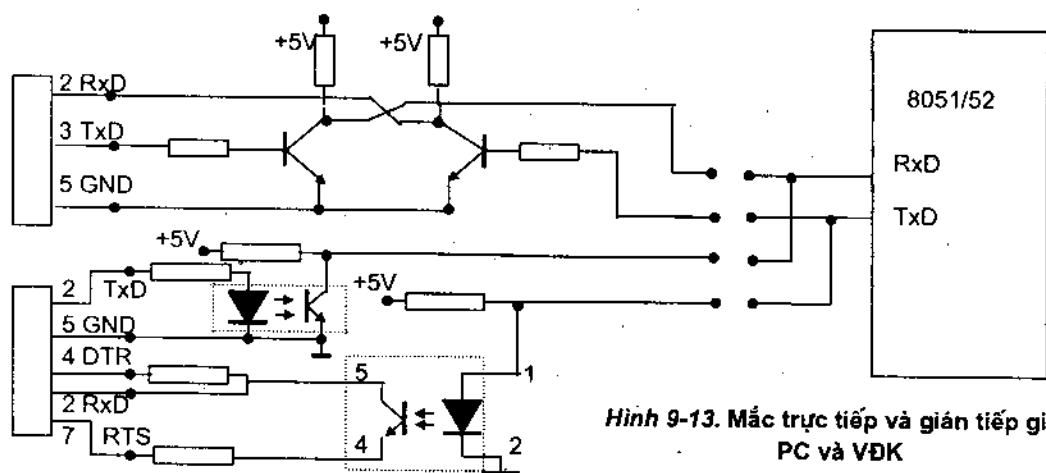
Giữa máy vi tính PC và VDK được kết nối với nhau qua cổng (chân DB9) và đường dây nối tiếp. Việc kết nối này phải theo chuẩn RS-232A về:

- Mức tín hiệu RS-232 là ± 12 V.
- Thủ tục trao đổi tin với các tín hiệu tin nhận RxD, tín truyền TxD theo đơn công (một chiều) hay song công (hai chiều đồng thời).

Vì VDK hay Vi xử lý của PC chỉ có mức tín hiệu TTL (+5 V và 0 V) nên phải có sự chuyển mức RS-232 hay ± 12 V sang (+5 V và 0 V) hay ngược lại nên phải có các sơ đồ chuyển mức tín hiệu. Với PC, ta phải dùng các vi mạch MC 1488 và MC1489, còn với VDK ta phải dùng một trong ba giải pháp sau:

- Dùng vi mạch MAX 232 (hình 9-9).
- Dùng điện trở và diot để ghim điện áp 5V ở lối vào VDK. Phương pháp này đơn giản và rẻ tiền nhất.
- Dùng tranzito chuyển mức tín hiệu theo phương pháp trực tiếp (hình 9-13).
- Dùng diot và tranzito phát quang ghép theo phương pháp gián tiếp (hình 9-13).

Phương pháp này rẻ tiền hơn MAX 232 (ghép một cổng nối tiếp) hay 233 (ghép hai cổng nối tiếp).



Hình 9-13. Mắc trực tiếp và gián tiếp giữa PC và VDK

9.5.3. Truyền nhận tin của PC

Máy vi tính phải có chương trình có những hành động sau:

- Giao diện với thực đơn với các hành động:
 - A. Truyền tin từ PC sang VDK
 - B. Nhận tin từ VDK vào PC
 - T. Thoát khỏi thực đơn

- Mã hay tệp truyền từ PC sang VDK;

- Mã hay tệp nhận từ VDK sang PC.

Người dùng phải chọn một trong các hành động bằng cách nhấn một trong các phím tương ứng (A, B, T), tìm tên hay mã của chương trình truyền hay nhận vào bộ nhớ của PC rồi nhấn Enter (CR). Khi nhận tin từ VDK, trước đó phải đặt tên tệp ghi tin nhận.

Một số chương trình truyền nhận bao gồm cả giao diện với thực đơn thông dụng như:

- Chương trình TERM. EXE của NC trong PC.
- Chương trình MPE. EXE của hệ ghép nối máy vi tính của thiết bị thực tập về giao diện với máy vi tính (phần Viện Tự động hóa, thành phố Hồ Chí Minh).
- Chương trình RAM2. EXE của hệ VDK của Thiết bị thực tập về vi xử lý (phần Viện Tự động hóa, thành phố Hồ Chí Minh).
- Chương trình SERIAL.EXE của Viện Kỹ thuật Hạt nhân Hà Nội.

Máy vi tính PC sẽ truyền các tệp chương trình (đuôi HEX hay BIN) cho VDK sau khi đã soạn thảo, hợp dịch, liên kết và chạy thử (từng bước, cả chương trình) cho VDK để nạp vào EEPROM trong VDK, nhớ chương trình ngoài hay RAM ngoài. Các chương trình này có thể dịch và liên kết và chạy thử với bất cứ chương trình dịch và mô phỏng nào (ASM51, PINNACLE v.v...).

9.5.4. Trao đổi tin nối tiếp của máy vi tính PC với bộ nhớ EEPROM

Bộ nhớ EEPROM là bộ nhớ ghi chương trình, có thể xóa và ghi lại bằng điện. Bộ nhớ EEPROM này có thể ở trong VDK (thường từ 4kB tới 8kB) hay nằm ngoài VDK. Việc ghi chương trình từ máy vi tính sang EEPROM theo các nguyên tắc sau:

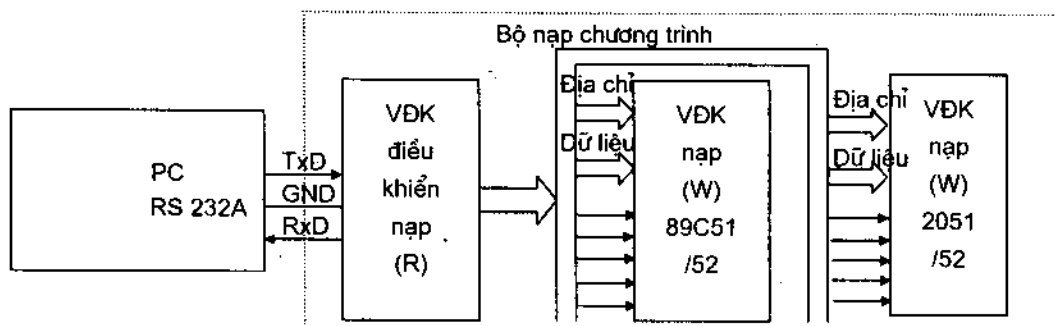
- PC đưa tin về chương trình theo cách nối tiếp sang VDK hay điều khiển nạp chương trình của bộ nạp (Programmer).

- Tin về chương trình được ghi song song từ PC sang VDK hoặc bộ nhớ EEPROM ngoài.

1. Trao đổi tin của PC với EEPROM của VDK

Máy vi tính PC đưa chương trình đã biên dịch và kết nối (dạng .HEX hay .BIN) sang nạp vào bộ nhớ EEPROM của VDK 2051 hay 89C51/52 theo sơ đồ khối như hình 9-14, sơ đồ nguyên lý như hình 9-15 và giao diện như hình 9-16.

Bộ nạp chương trình trên rất cần thiết cho những người nghiên cứu về VDK để nạp chương trình giám sát (Monitor) (xem tập II) và ứng dụng, trong đó có chương trình con trao đổi tin nối tiếp với máy vi tính PC để nạp chương trình đã soạn thảo, biên dịch, liên kết và chạy thử trên chương trình mô phỏng (AVSIM, Pinnacle v.v...). Chúng ta sẽ nghiên cứu chi tiết hơn về bộ nạp cùng chương trình nạp này ở chương Mở rộng khối nhớ ngoài (tập II). Ở đây, chúng ta chỉ quan tâm tới chương trình con giao tiếp với PC.



Hình 9-14. Sơ đồ khối của bộ nạp chương trình cho VDK

Trong chương trình ghi trong VDK của bộ nạp phải có:

- Chương trình trao đổi tin nối tiếp với máy tính PC theo nguyên tắc hỏi đáp (hội thoại) để nhận tin vào lối vào RxĐ nối với TxĐ của PC hay truyền tin trên TxĐ nối với RxĐ của PC.
- Chương trình ghi tin song song của chương trình vào các VDK nạp (W) với các tín hiệu sau:
 - + Địa chỉ: có địa chỉ thấp trên các chân P0 (A0-A7) và địa chỉ cao (A8-A12) trên P2.0 tới P2.
 - + Dữ liệu: D0-D7 trên các chân P1.0 tới P1.7.
 - + Điều khiển: điều khiển nguồn nuôi 5V/12V, ghi chương trình (/PGM) và tổ hợp các tín hiệu P2.6, P2.7, P3.6, P3.7 để có thể ghi, đọc và kiểm tra bộ nhớ EEPROM.

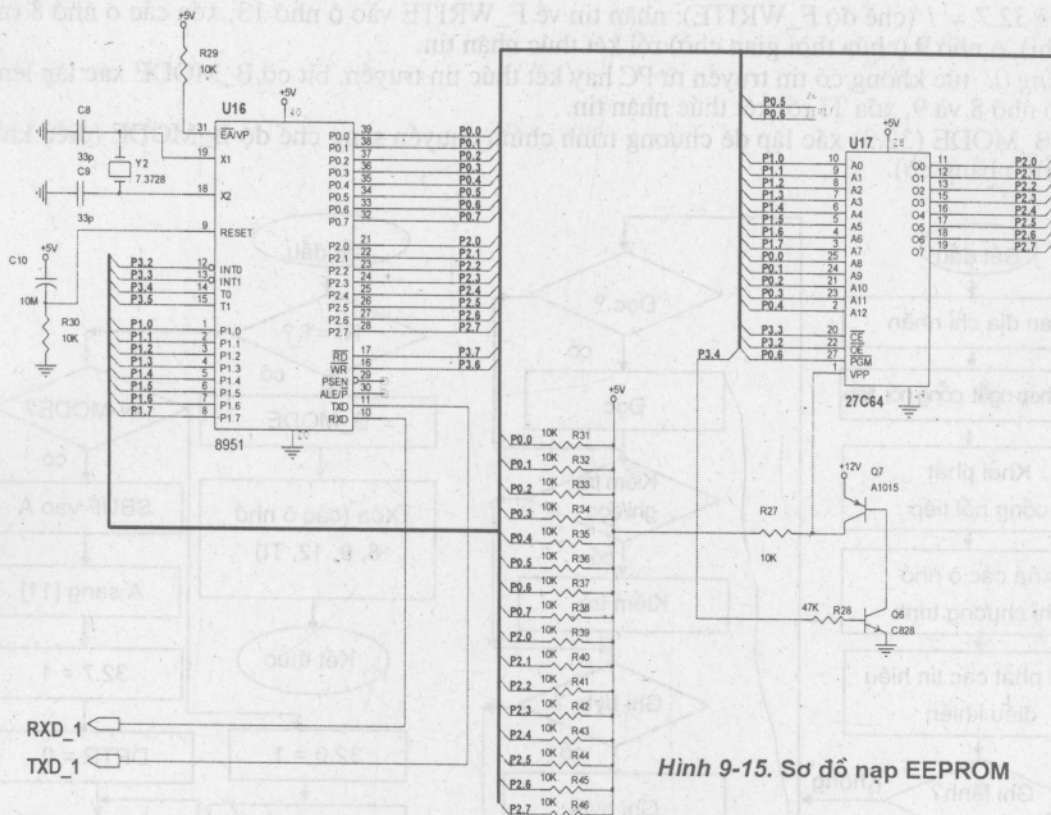
Chúng ta sẽ thấy rõ hơn về cách kết nối và chương trình nạp, trong đó có chương trình con trao đổi tin nối tiếp với PC qua chương trình nạp cho bộ nhớ EEPROM ngoài.

2. Trao đổi tin nối tiếp bộ nhớ chương trình EEPROM ngoài

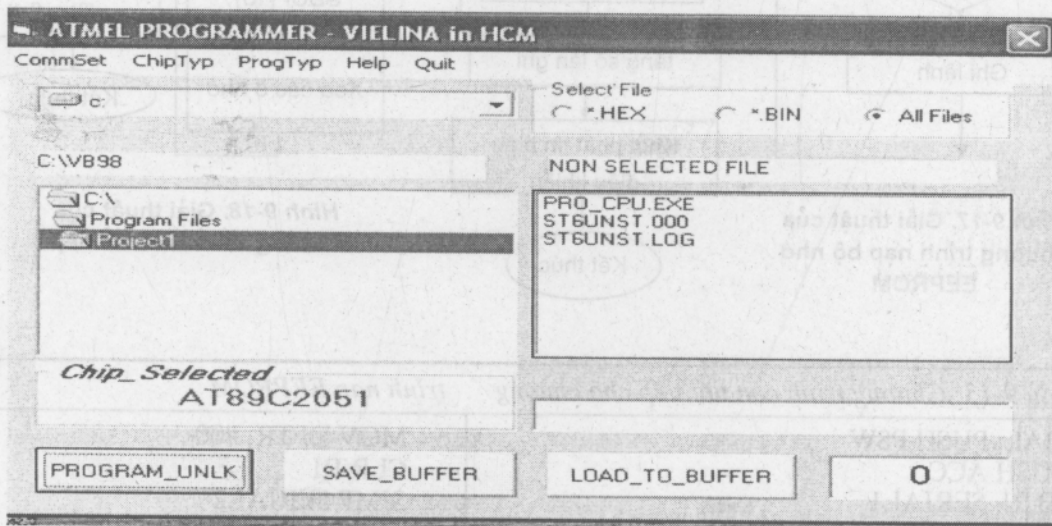
Bộ nhớ EEPROM ngoài được nạp chương trình có sơ đồ khối như hình 9-15, được ghi bởi địa chỉ từ các cổng P1 (địa chỉ thấp), P0 (địa chỉ cao) và dữ liệu từ cổng P2. Bộ nhớ EEPROM được ghi bởi nguồn +12 V hay +5 V (nhớ Flash) vào chân VPP và các tín hiệu điều khiển P3.4 (cho +12 V), P3.3 (cho /CE-chọn vi mạch), P3.2 (cho /OE- cho phép đọc ra), P0.6 (cho /PGM- ghi chương trình).

Chương trình toàn bộ của bộ nạp (xem ở phụ chương E) có thực đơn như hình 9-16 và chương trình khởi phát và chương trình con trao đổi tin nối tiếp với PC như ở ví dụ 9-13 với giải thuật ở hình 9-17. Thực đơn hay giải thuật của chương trình nạp cho EEPROM 2764 có các hành động sau:

- Khởi phát: gán nhãn, khai báo nhớ đệm, khai báo cờ (flag), cho phép ngắt, khởi phát cổng nối tiếp, xóa ô nhớ, khởi phát các tín hiệu điều khiển (nguồn nuôi 5V/12V, P3.4-T0 = 0, P3.3-/CE = 1, P0.6-/PGM = 1, P3.2-/OE = 1).
- Thực đơn: có các hành động ghi lệnh (COMMANd), đọc (READ), kiểm tra trắng (Blank check), ghi dữ liệu (Write) và kết thúc.



Hình 9-15. Sơ đồ nạp EEPROM



Hình 9-16. Giao diện của chương trình nạp EEPROM

Chương trình của ví dụ 9-13 chỉ nhận tín hiệu nối tiếp từ máy vi tính PC. Khi bit cờ nhận RI có giá trị: - bằng 1: tức có tín hiệu truyền từ PC sang, bộ nhớ của VĐK nhận tín hiệu, có một trong hai khả năng xảy ra do cờ 32.7 (W_MODE) quyết định:

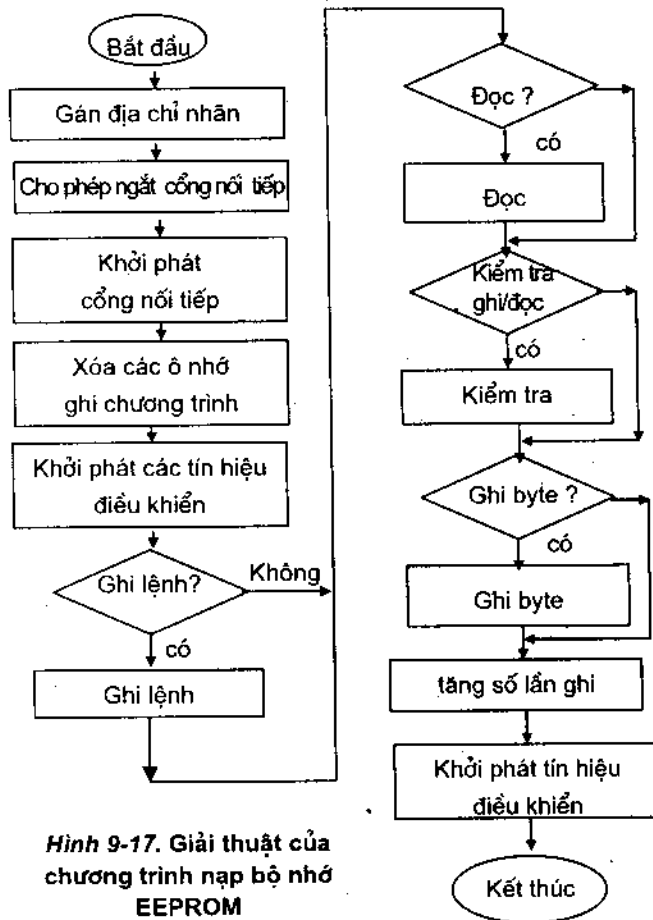
Chương trình của ví dụ 9-13 chỉ nhận tín hiệu nối tiếp từ máy vi tính PC. Khi bit cờ nhận RI có giá trị bằng 1 tức có tín hiệu truyền từ PC sang, bộ nhớ của VĐK nhận tín hiệu, có một trong hai khả năng xảy ra do cờ 32.7 (W_MODE) quyết định:

+ Cờ 32.7 = 0 (chế độ COMMAND khi 32.4 = 1): nhận tín hiệu về lệnh vào ô nhớ 11 chứa lệnh, rồi xác lập B_MODE, xóa địa chỉ DPTR về 0 rồi kết thúc nhận tín hiệu.

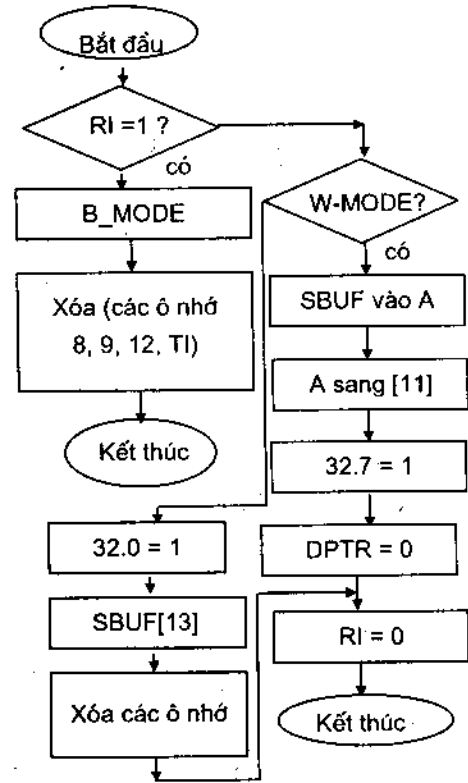
+ Cờ 32.7 = 1 (chế độ F_WRITE): nhận tin về F_WRITE vào ô nhớ 13, xóa các ô nhớ 8 (nhớ dữ liệu ghi), ô nhớ 9 (chứa thời gian chờ) rồi kết thúc nhận tin.

- Bảng 0: tức không có tin truyền từ PC hay kết thúc tin truyền, bit cờ B_MODE xác lập lên 1, xóa các ô nhớ 8 và 9, xóa TI rồi kết thúc nhận tin.

Cờ B_MODE (32.3) xác lập để chương trình chính chuyển sang chế độ B_MODE (điều khiển chương trình bằng cờ).



Hình 9-17. Giải thuật của chương trình nạp bộ nhớ EEPROM



Hình 9-18. Giải thuật của chương trình con nối tiếp

Ví dụ 9-13: Chương trình con nối tiếp cho chương

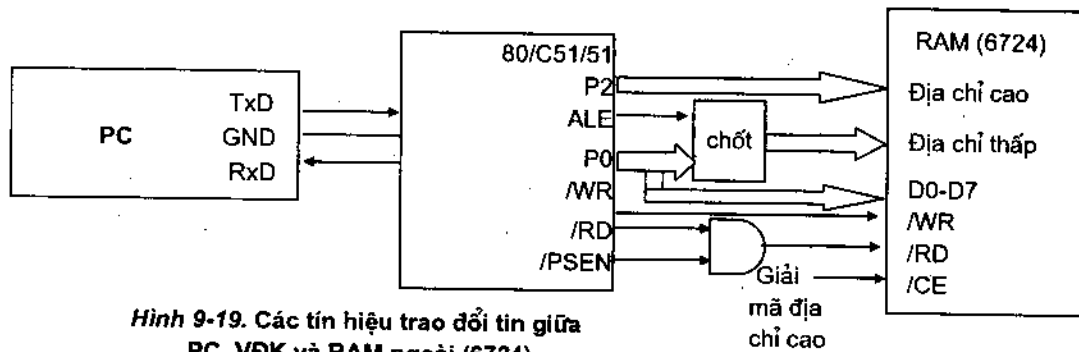
trình nạp EEPROM

```

SERIAL: PUSH PSW
        PUSH ACC
        JB RI, SERIAL1
        SETB 32.3
        MOV 8, #0
        MOV 9, #0
        MOV 12, #0
        CLR TI
        SJMP SERIAL2
SERIAL1: JB 32.7, SERIAL3
        MOV A, SBUF
        MOV 11, A
        SETB 32.4
    
```

```

MOV DPTR, #00
CLR RI
SJMP SERIAL2
SERIAL3: SETB 32.0
        MOV 13, SBUF
        MOV 8, #0
        MOV 9, #0
        CLR RI
SERIAL2: POP ACC
        POP PSW
        RET
    
```



Hình 9-19. Các tín hiệu trao đổi tin giữa PC, VDK và RAM ngoài (6724)

9.5.5. Trao đổi tin nối tiếp của máy vi tính PC với bộ nhớ RAM ngoài

Nhớ RAM ngoài nằm ngoài VDK, dùng để ghi nhớ chương trình và dữ liệu trong hệ VDK có mở rộng bộ nhớ. Dùng nhớ RAM ngoài thay cho nhớ EEPROM ngoài có đặc điểm là:

- Chương trình từ máy vi tính PC có thể nạp trực tiếp trên RAM ngoài để chạy chương trình thông qua VDK trong hệ VDK có RAM ngoài. Muốn vậy, VDK nhận tin nối tiếp từ PC qua cổng nối tiếp RS-232A và vi mạch MAX232 hay MAX233 (có hai mạch đổi mức tín hiệu RS-232 sang TTL và ngược lại).

- Chương trình trên được VDK ghi song song vào các ô nhớ của bộ nhớ RAM theo sơ đồ ở hình 9-19.

- Chương trình trên RAM được VDK đọc song song vào các thanh ghi đếm lệnh và đếm lệnh (PC) để thực hiện từng lệnh. Các tín hiệu hội thoại của hệ như hình 9-20.

- Chương trình trên sẽ bị mất khi tắt nguồn nuôi +5V của VDK.

Người ta dùng RAM ngoài này để chạy thử chương trình trước khi nạp vào EEPROM của VDK hay vi mạch EEPROM bên ngoài. Nếu dùng NVRAM hay nhớ flag, chương trình được nạp nối tiếp ngay lần đầu tiên với nguồn nuôi +5 V.

Quá trình ghi và thực hiện chương trình trên RAM tiến hành như sau:

- *VDK ghi chương trình trên RAM trong:* chuyển mạch EXT/INT đặt ở vị trí INT, tức chân /EA/VP (chân số 31) của VDK nối với +5 V, VDK nhận tin nối tiếp từ PC về chương trình vào bộ nhớ RAM bên trong (IRAM).

- *VDK ghi chương trình lên RAM ngoài và chạy chương trình:* theo các bước:

- + Nhấn và giữ nút RESET và chuyển mạch EXT/INT đặt ở vị trí EXT, tức chân /EA/VP (chân số 31) của VDK nối với 0V.

- + Nhấn và giữ nút RESET và công tắc ROM/RAM để ở vị trí RAM.

- + Nhả nút RESET.

- *VDK chạy chương trình:* chương trình trên RAM có thể chạy sau khi ấn nút RESET nếu cuối chương trình có kết thúc chờ và địa chỉ 00H có lệnh nhảy tới lệnh bắt đầu của chương trình tại địa chỉ bắt đầu của bộ nhớ RAM (ví dụ 6000H). Có thể cho chương trình chạy ngay sau khi nạp nếu cuối chương trình có lệnh LJMP DIA_CHI_DAU của chương trình (xem ví dụ 9-11).

Ví dụ 9-14 giới thiệu chương trình trao đổi tin nối tiếp đơn giản không có hội thoại của VDK với PC để nhận chương trình cho bộ nhớ RAM.

Hình 9-20 giới thiệu giải thuật của chương trình trao đổi tin nối tiếp có hội thoại của VDK với PC (xem phụ chương) để nhận chương trình cho bộ nhớ RAM.

Chú ý: chương trình trên có những đặc điểm sau:

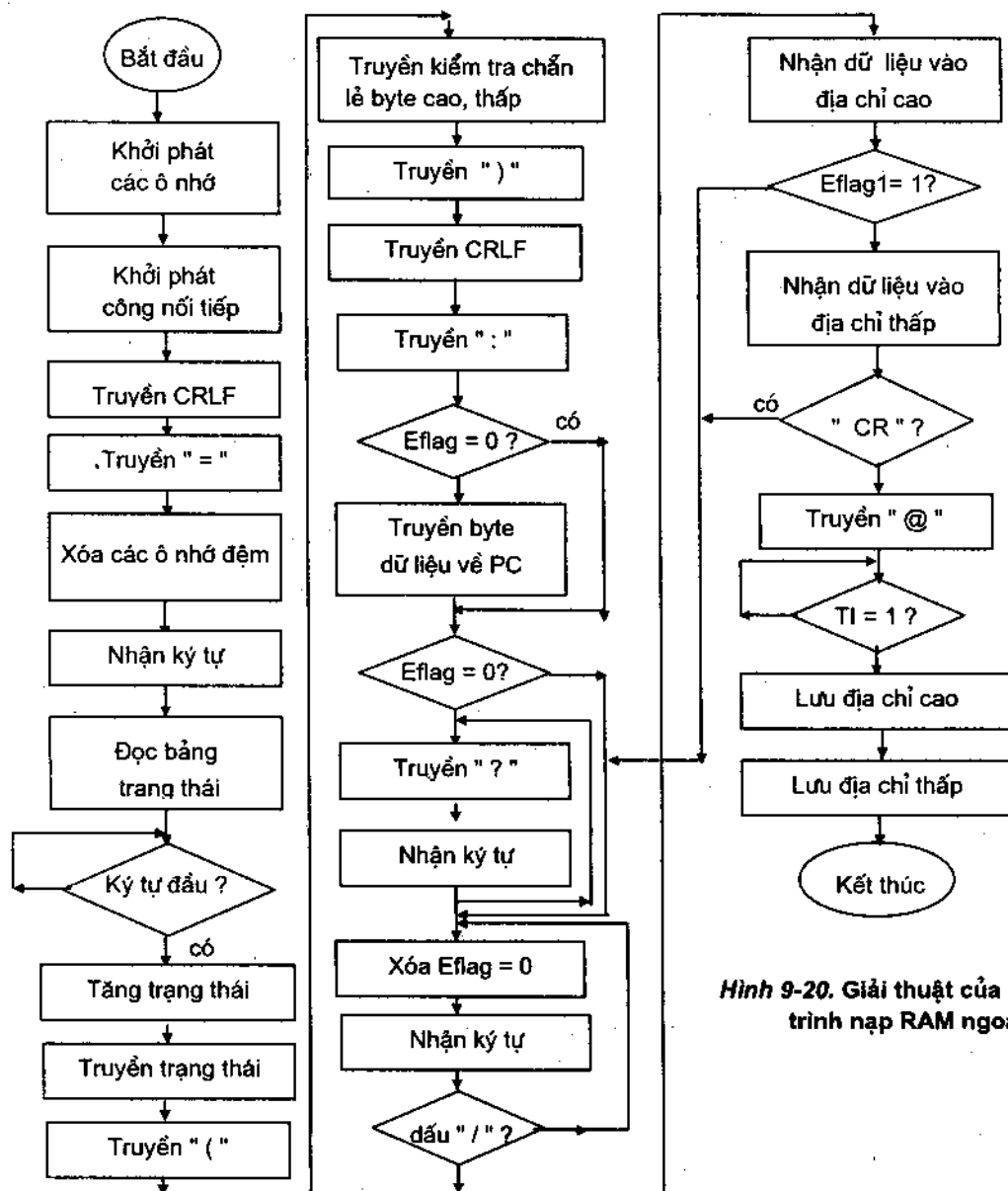
- Ghi nhận tin nối tiếp từ PC rồi ghi ra bộ nhớ ngoài RAM ở địa chỉ đầu là 6000H.

- Nhận 26×250 byte từ PC.

- Kết thúc nhận bởi nháy đèn nối với bit P1.0 10 lần.

- Chạy chương trình ngay sau khi nhận bởi lệnh LJMP 6000H.

- Chương trình không có khả năng phát hiện lỗi truyền tin từ PC sang VDK hay khi ghi vào RAM để yêu cầu PC truyền lại.



Hình 9-20. Giải thuật của chương trình nạp RAM ngoài

Chương trình có giải thuật ở hình 9-20 có những đặc điểm sau:

- Đầu tiên VDK truyền "CRLF" cho máy vi tính PC để xuống dòng (CR) và đưa về đầu dòng (LF) của màn hình.
- VDK truyền dấu "=" để yêu cầu PC truyền tin.
- VDK nhận tin dạng HEX theo các bước sau:
 - + Xóa các bộ nhớ đệm cho dữ liệu và các bit cờ (flag).
 - + Nhận tin một ký tự từ PC, biến đổi dạng ASCII (để truyền) sang dạng HEX, lưu trữ trong bộ nhớ đệm.
 - + Gọi trạng thái trong bảng có 16 trạng thái. Xét trạng thái của ký tự đầu tiên. Nếu là ký tự đầu tiên sẽ tăng trạng thái. Với các trạng thái khác, sẽ gọi các chương trình con tương ứng.

- + Nếu chưa có cờ Doneflag, tiếp tục nhận ký tự.
- + Truyền, dữ liệu về trạng thái, " (kiểm tra chẵn lẻ byte cao, byte thấp)" và " CRLF " (báo kết thúc việc truyền) cho PC.

VĐK truyền lỗi bằng dấu " : " và xét lỗi, nếu không có lỗi, sẽ thoát. Khi có lỗi truyền, lại truyền byte dữ liệu về trạng thái cho PC.

VĐK xét cờ lỗi Eflag, nếu không lỗi nhảy tới HEXOK, nếu có lỗi truyền dấu " ? " và nhận ký tự về trạng thái và lại lặp lại xét lỗi.

- VĐK chấp nhận byte HEX (HEXOK), xóa cờ Eflag và nhận dữ liệu. Nếu không có dấu " / ", chương trình lại trở về HEXOK để chờ PC xóa Eflag và gửi dữ liệu tiếp.

- VĐK nhận dữ liệu. Nếu có lỗi Errflag1, lại trở về HEXOK để nhận dữ liệu. Nếu không có lỗi, dữ liệu được ghi vào địa chỉ cao.

- VĐK nhận dữ liệu. Nếu có lỗi Errflag1, lại trở về HEXOK để nhận dữ liệu. Nếu không có lỗi, dữ liệu được ghi vào địa chỉ thấp.

- VĐK nhận dữ liệu. Nếu không là "CR" (chỉ kết thúc tin về dữ liệu), lại trở về HEXOK để nhận dữ liệu tiếp.

- VĐK trả lời bằng cách truyền " @ " cho PC để xác nhận đã nhận hai byte tin về dữ liệu, ghi vào hai địa chỉ cao và thấp.

- VĐK chờ xác lập cờ truyền TI, lưu trữ các địa chỉ thấp và cao và kết thúc chương trình chính.

Qua giải thuật trên, ta thấy:

- PC và VĐK trao đổi tin nối tiếp theo ký tự dưới dạng mã ASCII, mỗi mã ASCII là một byte dữ liệu. Nếu truyền một byte dạng mã HEX, phải biến đổi thành hai byte dạng ASCII bởi chương trình con HEXASC (xem ví dụ 9-12 cho chương trình PRByte). Nếu nhận hai byte dạng ASCII, phải biến đổi thành một byte dạng mã HEX bởi chương trình con ASCHEX (xem ví dụ 9-12 cho chương trình GETBYTE và các chương trình xử lý 4 bit cao StLeft và xử lý 4 bit thấp StRight).

- PC và VĐK trao đổi tin nối tiếp theo hội thoại (hình 9-18) như sau:

- + Bắt đầu chương trình, VĐK truyền "CRLF" và " = " cho PC.

- + Khi nhận dữ liệu bởi chương trình con HEXIN, VĐK truyền cho PC trạng thái (trong bảng) kèm theo ký hiệu "(byte cao, byte thấp của kiểm tra chẵn lẻ)" cùng "CRLF" (chỉ kết thúc truyền trạng thái).

- + VĐK báo lỗi cho PC bằng cách truyền ký hiệu " : " bởi chương trình con ERRPrt và nếu không có lỗi, sẽ truyền Byte dữ liệu của chương trình con PrByte.

- + VĐK báo lỗi cho PC bằng cách truyền ký hiệu " ? " để khẳng định rằng chờ nhận dữ liệu, nếu có lỗi truyền lại truyền lại dấu " ? ".

- + PC truyền ký hiệu " / " để báo tiếp tục truyền dữ liệu. Nếu chưa có ký hiệu trên, lại trở về chương trình HEXOK.

- Mỗi lần nhận một ký tự về trạng thái VĐK nhảy tới một trong các chương trình con tương ứng với số mã từ 0 tới 12 của bảng trạng thái sau:

- + StWait: chờ ký tự bắt đầu.

- + StLeft: biến đổi và ghi 4 bit cao của byte trạng thái vào dữ liệu và tăng trạng thái.

- + StGetCnt: biến đổi và ghi 4 bit thấp, để có một byte hoàn chỉnh, gửi vào ByteCount và tăng trạng thái.

- + StLeft: biến đổi và ghi 4 bit cao của byte địa chỉ 1 (cao).

- + StGetAd1: nhận địa chỉ byte 1.

- + StLeft: biến đổi và ghi 4 bit thấp của byte địa chỉ 2 (thấp).

- + StGetAd2: nhận địa chỉ byte 2.

- + StLeft: biến đổi và ghi 4 bit cao của bản ghi.

- + StGetRec: nhận loại bản ghi.

- + StLeft: biến đổi và ghi 4 bit cao của byte dữ liệu.

- + StGetDat: nhận byte dữ liệu.

- + StLeft: biến đổi và ghi 4 bit cao của kiểm tra chẵn lẻ (tổng).

- + *StGetChk*: nhận kiểm tra tổng.
- + *StSkip*: bỏ qua dữ liệu sau khi có lỗi.
- + *BadState*: trạng thái xấu, không bao giờ nhận ở đây.
- + *BadState*: trạng thái xấu, không bao giờ nhận ở đây.

; Ví dụ 9-14: Chương trình của VDK với máy tính PC và khối nhớ

<pre> org 00h \$INCLUDE MYLIB.ASM @BAUD2 high(-3), low(-3), high(-3), low(-3) ; tốc độ 9600 baud diachinh equ 110h menu: jb ti, truyen jb ri, nhan jmp menu truyen: mov dptr, #diachitr lap1: mov a, #00h movc a, @a+dptr jz kt @truyen0 40h </pre>	<pre> setb ti inc dptr sjmp lap1 kt: sjmp menu nhan: mov dptr, #diachinh @nhan0 jz kt lap2: movx @dptr, a inc dptr sjmp lap2 sjmp menu org 100h diachitr: db 'Microcontroller 8051/52', 0 end </pre>
---	--

9.5.6. Các chương trình con trao đổi nối tiếp giữa máy vi tính PC và VDK

VDK trao đổi tin theo từng ký tự và theo chuỗi ký tự dạng mã ASCII với các chương trình con *Puchar* (truyền) hay *Getchar* (nhận) của VDK như đã xét ở đầu chương.

Ở đây, chúng ta giới thiệu các chương trình trao đổi tin nối tiếp với PC với các dạng dữ liệu khác như dạng BCD không nén, BCD nén, dạng nhị phân và dạng ASCII có cả bit kiểm tra (tổng các bit 1 hay chẵn lẻ).

1. Chương trình trao đổi tin dạng BCD không nén

Dữ liệu dạng BCD không nén là các mã nhị phân cho các số tự nhiên từ 0 tới 9 được chứa trong một byte nhớ 8 bit. Như vậy, các số này chính là các số dạng HEX cho các số tự nhiên từ 0 tới 9. Khi trao đổi tin nối tiếp với PC, các số này phải đổi thành dạng ASCII, tức các số dạng HEX này phải cộng với 30H để thành các số mã ASCII của các số tự nhiên từ 0 tới 9 là 30H tới 39H.

Ví dụ 9-15 giới thiệu chương trình truyền nhiều byte HEX không nén của các số hàng vạn, ngàn, trăm, chục, đơn vị được chứa trong các ô nhớ đệm *TTHOU*, *THOU*, *HUN*, *TEN* và *UNIT*.

<i>; Ví dụ 9-15: Chương trình trao đổi tin số</i>	<i>dạng BCD không nén</i>
<pre> SENDBUF: MOV A, TTHOU; gửi chục ngàn ADD A, #30H; đổi thành mã ASCII CALL SEND; gửi ký tự MOV A, THOU; hàng ngàn ADD A, #30H CALL SEND MOV A, HUN; hàng trăm ADD A, #30H CALL SEND MOV A, TEN; hàng chục ADD A, #30H CALL SEND </pre>	<pre> MOV A, UNIT; hàng đơn vị ADD A, #30H; CALL SEND SEND:; chương trình gửi một ký tự cho PC PUDH PSW; lưu giữ PSW PUSH ACC; lưu giữ Acc MOV SBUF, A; gửi ký tự trong A SEN0: JNB SENDFLAG, SEN0; chờ gửi xong CLR SENDFLAG; xóa cờ gửi POP ACC; hồi phục A POP PSW; hồi phục trạng thái RET; trở về END; kết thúc dịch </pre>

2. Chương trình trao đổi tin dạng BCD nén

Dạng BCD nén là dạng một byte (8 bit) chứa hai số BCD hay HEX (mỗi số 4 bit hay một nible) cho số tự nhiên từ 0 tới 9.

Khi truyền nối tiếp trên đường dây RS 232 phải đổi chúng thành dạng BCD không nén tức một byte dạng BCD nén thành hai byte không nén và đổi thành mã ASCII. Ví dụ 9-12 giới thiệu chương trình truyền một số BCD nén trong một byte với tên là PrByte.

- Nhận hai lần hai ký tự, biến đổi sang dạng HEX (bởi ASCHEX) rồi gửi vào nhớ đệm CH.
- Xử lý 4 bit cao và thấp (bởi StLeft và StRight) để trở thành một byte dạng HEX.

Ví dụ 9-16 giới thiệu chương trình nhận một số BCD nén trong một byte với tên là GetByte.

Ví dụ 9-17 giới thiệu chương trình truyền một số BCD nén trong một byte với tên là PrByte.

; Ví dụ 9-16: Chương trình nhận byteBCD nén (đổi ASCII sang một byte HEX)	
GetByte: ACALL GetChar; nhận một ký tự ACALL ASCHEX ; đổi ASCII sang HEX MOV CH, A ; gửi vào nhớ đệm ACALL StLeft ; xử lý 4 bit cao ACALL GetChar ; nhận một ký tự ACALL ASCHEX ; đổi ASCII sang HEX MOV CH, A ; gửi vào nhớ đệm ACALL StRight ; xử lý 4 bit thấp RET ASCHEX: RET HEXASCII; HEX sang ASCII (chương trước) RET PUTCHAR ; truyền 1 byte ký tự mã ASCII JNB TI, \$ CLR TI MOV SBUF, A RET	StLeft: MOV A, CH ; gọi nhớ đệm ANL A, #0FH ; lấy 4 bit thấp SWAP A ; trao A MOV DataByte, A ; gửi vào DataByte RET ; kết thúc StRight: MOV A, CH ; gọi nhớ đệm ANL A, #0FH ; lấy 4 bit thấp ORL A, DataByte ; cộng với 4 bit thấp MOV DataByte, A ; gửi vào DataByte RET ; kết thúc GetChar: JNB RI, \$; nhận ký tự CLR RI ; xóa RI MOV A, SBUF ; nhận vào A RET

; Ví dụ 9-17: Chương trình truyền một byte BCD nén (sang 2 byte không nén, sang ASCII)	
PrByte: PUSH ACC ; lưu giữ A SWAP A ; hoán đổi nible cao sang thấp CALL HEXASCII ; đổi HEX sang ASCII CALL PUTCHAR POP A CALL HEXASCII CALL PUTCHAR RET	

3. Chương trình truyền một số hai byte có 6 chữ số tự nhiên

Một lời 2 byte (16 bit) dạng nhị phân có thể chứa tới 65535 dạng thập phân, nếu kể cả dấu, có tới 6 con số. Dữ liệu của VDK truyền về cho PC có thể tới 6 con số này trong hai byte truyền. Muốn vậy, phải:

- Biến đổi số nhị phân 2 byte sang BCD bởi chương trình BIN2BCD (xem chương Các lệnh logic). Các con số được ghi trong thanh ghi R7 (một chữ số hàng chục ngàn), R6 (hai chữ số hàng ngàn và hàng trăm) và R5 (hai chữ số hàng chục và đơn vị).

- Truyền 5 lần cho 5 chữ số hay 5 mã BCD hay HEX của con số sau khi đổi sang mã ASCII.

Ví dụ 9-18 giới thiệu chương trình truyền một số hai byte nhị phân có 6 con số tự nhiên.

; Ví dụ 9-18: Chương trình truyền nối tiếp một số có 6 chữ số

SEND6DIG: CALL BIN2BCD ; biến đổi 2 byte BIN ra BCD MOV A, R7 ; lấy chữ số hàng chục ngàn ANL A, #0FH ; lấy 4 bit cao ADD A, #'0' ; đổi sang mã ASCII CALL Prchar ; truyền MOV A, R6 ; lấy chữ số hàng ngàn SWAP A ; tráo A ANL A, #0FH ; lấy 4 bit thấp ADD A, #'0' ; đổi sang mã ASCII CALL Prchar ; truyền MOV A, R6 ; lấy chữ số hàng trăm SWAP A; ; tráo A ANL A, #0FH ; lấy 4 bit cao ADD A, #'0' ; đổi sang mã ASCII CALL Prchar ; truyền	MOV A, R5 ; lấy chữ số hàng chục SWAP A ; tráo A ANL A, #0FH ; lấy 4 bit thấp ADD A, #'0' ; đổi sang mã ASCII CALL Prchar ; truyền MOV A, R5 ; lấy chữ số hàng đơn vị SWAP A ; tráo A ANL A, #0FH ; lấy 4 bit thấp ADD A, #'0' ; đổi sang mã ASCII CALL Prchar ; truyền RET BIN2BCD: ; chương trình con BIN2 ra BCD RET ; (xem chương 6) Prchar: ; chương trình con truyền một ký tự RET ; (xem ở trên)
--	--

CÂU HỎI VÀ THỰC HÀNH

I. Câu hỏi

1. Nêu cấu tạo của cổng nối tiếp của VĐK 8051/52.
2. Nêu yêu cầu và các lệnh của việc khởi phát cổng nối tiếp của VĐK 8051/52.
3. Nêu các chương trình con của truyền tin và nhận tin nối tiếp, so sánh đặc điểm của chúng.
4. Hãy so sánh trao đổi tin nối tiếp trong hệ nhiều VĐK và một VĐK.
5. Hãy so sánh trao đổi tin nối tiếp với trao đổi tin nối tiếp qua một bit của cổng song song.
6. Hãy so sánh đặc điểm trao đổi tin song song và nối tiếp trong một VĐK.
7. Muốn trao đổi tin nối tiếp giữa một máy tính PC với VĐK phải có những điều kiện gì ? Nêu đặc điểm của chương trình cho VĐK và cho PC.
8. Hãy so sánh việc sử dụng tốc độ baud của máy phát nhíp và qua Timer.
9. Tại sao người ta hay dùng timer làm máy phát tốc độ baud ?
10. Tại sao người ta hay dùng timer2 làm máy phát tốc độ baud thay cho timer1?

II. Thực hành

1. Quan sát các thành phần của cổng nối tiếp với PINNACLE:

Chúng ta có thể quan sát các thanh ghi và hoạt động liên quan tới cổng nối tiếp trên màn hình của chương trình mô phỏng PINNACLE như sau:

- Vào chương trình PINNACLE.
- Vào thực đơn VIEW.
- + Xem Register : các thanh ghi PCON, SCON, PSW
- + Xem UART khi chạy chương trình cổng nối tiếp.

- Vào thực đơn TOOL: vào Code Wizards để xác lập tốc độ baud cho chương trình cho timer1 và timer2 (phải vào VĐK 8052).

2. Tính và lập trình tốc độ baud với timer1 và timer2, kiểm tra lại bằng Code Wizard.
3. Soạn thảo chương trình truyền tin cho các chế độ.
4. Hợp dịch, liên kết và cho chạy thử (từng bước và cả chương trình) cho các ví dụ từ ví dụ 9-1 tới 9-16.

III. Thực hành lập trình nâng cao

1. Trao đổi tin nối tiếp và hiển thị trên đèn LED 7 đoạn một ký tự.
2. Trao đổi tin nối tiếp và hiển thị trên đèn LED 7 đoạn một chuỗi ký tự theo kiểu đưa từng ký tự.
3. Lập trình cho trao đổi tin nối tiếp một chuỗi ký tự theo kiểu đưa cả chuỗi ký tự.
4. Lập trình cho trao đổi tin nối tiếp một ký tự ra và vào cổng nối tiếp khi nối lối ra TxD với lối vào RxD.
5. Lập trình trao đổi tin nối tiếp cho một byte dữ liệu dạng HEX thành các con số hiển thị trên đèn LED 7 đoạn.
6. Lập trình trao đổi tin nối tiếp trên một bit của cổng song song với tốc độ baud 9600.

Chương 10

ĐIỀU KHIỂN NGẮT

Mục đích của chương:

- Hiểu rõ cơ chế ngắt chương trình của VĐK 8051/8052.
- Hiểu rõ các ngắt thời gian (TIMER0, TIMER1, TIMER2), ngắt ngoài (INT0, INT1, EXF2) và ngắt nối tiếp (RI, TI) của VĐK 8051/8052.
- Hiểu rõ ứng dụng các ngắt trong lập trình cho VĐK.

10.1. TỔNG QUÁT VỀ NGẮT CỦA VĐK 8051/8052

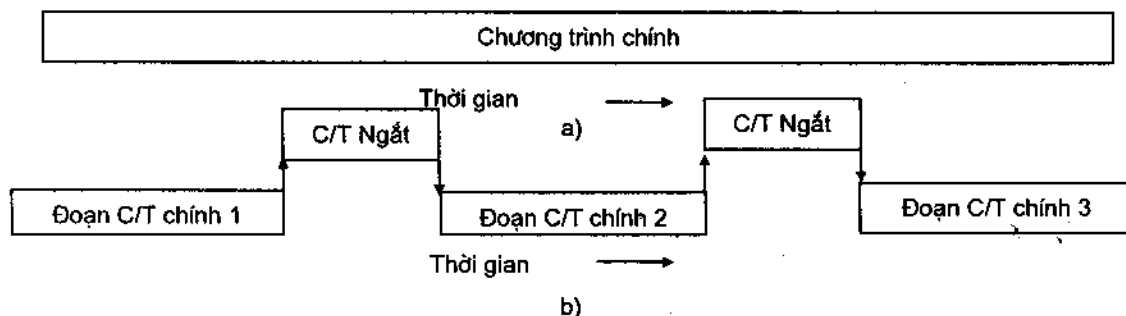
10.1.1. Yêu cầu ngắt chương trình đang chạy

Chúng ta đã xét các hành động cơ bản của một VĐK như trao đổi tin song song, trao đổi tin nối tiếp, định thời, đếm số xung vào. Khi hoạt động, ở một thời điểm, VĐK chỉ thực hiện một lệnh hay một hành động duy nhất của một chương trình.

Trong thực tế, nếu một hệ vi xử lý hay một máy vi tính cần trao đổi tin hay thực hiện nhiều chương trình khác nhau, nó có thể thực hiện theo nguyên tắc chia sẻ thời gian: các lệnh của các chương trình sẽ lần lượt được thực hiện nối tiếp nhau theo trình tự thời gian và phân phối đều cho các hành động, do đó các chương trình đều được thực hiện sau một thời gian chia sẻ lẫn nhau.

Với hệ VĐK, khi cần chạy nhiều chương trình, không có sự chia sẻ thời gian mà chỉ hoạt động theo nguyên tắc ngắt chương trình: chương trình đang thực hiện bị ngắt (hay dừng) để thực hiện chương trình mới có yêu cầu thực hiện. Khi thực hiện xong chương trình ngắt mới, VĐK lại trở về thực hiện tiếp chương trình đã bị ngắt, tại chỗ bị ngắt. Có thể, có trường hợp, chương trình ngắt đang chạy lại bị chương trình ngắt mới khác làm ngắt (ngắt trong ngắt), thì chương trình ngắt mới lại được thực hiện và chương trình ngắt cũ bị dừng và lại được thực hiện sau khi chương trình ngắt mới kết thúc và chương trình đầu tiên được thực hiện tiếp sau cùng.

Hình 10-1 minh họa việc thực hiện chương trình chính (chương trình đầu tiên) và các chương trình ngắt khi không có ngắt và có ngắt chương trình.



Hình 10-1. Thực hiện chương trình không ngắt (a) và chương trình có ngắt (b)

Tổ chức ngắt chương trình này đáp ứng được việc thực hiện nhiều chương trình của nhiều hành động khi có yêu cầu, nhưng theo nguyên tắc nối tiếp về thời gian, tại mỗi một thời điểm, chỉ một chương trình được thực hiện. Nếu có nhiều yêu cầu cần thực hiện, các yêu cầu được ghi lại để thực

hiện lần lượt, nhưng phải tổ chức thực hiện theo sự ưu tiên: nếu có hai yêu cầu ngắt chương trình xảy ra đồng thời, chương trình nào có quyền ưu tiên hơn sẽ được thực hiện trước, chương trình kia sẽ thực hiện sau.

10.1.2. Tổ chức ngắt chương trình của VĐK 8051/8052

VĐK 8051 có cấu tạo với 5 ngắt và 8052 có 6 ngắt (thêm ngắt INTER2) cho 6 hành động cơ bản (bảng 10-1) là xác lập (Reset), định thời/đếm (2), ngắt ngoài (2) và trao đổi tin nối tiếp (1) (cả truyền và nhận tin). Các ngắt này có địa chỉ của chương trình phục vụ ngắt tương ứng (ISR-Interrupt Service Routine) tại các địa chỉ nhớ cố định trong bộ nhớ chương trình PROM ở bên trong VĐK như bảng 10-1, được gọi là các vector ngắt. Các ngắt Reset, ngắt ngoài do các tín hiệu ngoài đưa vào các chân của cổng P3 như ở bảng 10-1.

Bảng 10-1. Các vector ngắt của 8051/8052

Ngắt	Địa chỉ trong ROM	Cờ ngắt	Bit cờ của SFR	Chân
Reset	0000H	Không	Không	9
Ngắt ngoài INT0 (phần cứng)	0003H	IE0	TCON.1	12 (P3.2)
Ngắt Timer 0 (TF0)	000BH	TF0	TCON.5	
Ngắt ngoài INT1 (phần cứng)	0013H	IE1	TCON.3	13 (P3.3)
Ngắt Timer 1 (TF1)	001BH	TF1	TCON.7	
Ngắt nối tiếp (RI, TI)	0023H	(RI+TI)	SCON.1	
Ngắt Timer 2 (TF2)	002BH	TF2	T2CON.7	

Trừ Reset được dành 2 địa chỉ nhớ, còn khoảng cách địa chỉ giữa hai vector ngắt liên tiếp có 8 địa chỉ nhớ dành để ghi chương trình phục vụ ngắt. Các địa chỉ này dành để :

- Ghi lệnh nhảy dài tới địa chỉ bắt đầu của chương trình chính (cho Reset).
- Ghi trực tiếp chương trình phục vụ ngắt nếu độ dài nhỏ hơn 8 byte.
- Ghi lệnh nhảy dài tới nhãn của lệnh bắt đầu chương trình phục vụ ngắt (nằm ở sau chương trình chính). Nếu trong chương trình chính không có chương trình phục vụ ngắt nào đó, tại các nhãn của chương trình phục vụ ngắt chỉ có mỗi lệnh RETI (trở về từ ngắt).

Trong một chương trình chính có chương trình phục vụ ngắt (ví dụ 10-1), chúng ta phải tuân theo:

- Tại vector Reset, phải có lệnh nhảy tới chương trình chính (LJMP MAIN), để nhảy qua vùng nhớ dành cho các vector ngắt (từ 03H tới 23H+8H). Nếu không có lệnh nhảy này, chương trình chính sẽ ghi vào các ô nhớ của các vector ngắt và không có việc thực hiện các chương trình phục vụ ngắt.

Thường chương trình chính bắt đầu ở địa chỉ 30H hay 100H với kê khai :

ORG 30H (hay ORG 100H)

MAIN: ; để lệnh LJMP MAIN tại địa chỉ 00H của RESET nhảy tới đây.

- Có lệnh xác lập địa chỉ ban đầu cho thanh ghi SP là #70H vì sau khi ấn Reset, SP = 7H. Nếu không có lệnh này, vùng nhớ dành cho ngăn xếp sẽ đè lên các vector ngắt.

- Có các lệnh nhảy tới nhãn của lệnh đầu tiên của các chương trình phục vụ ngắt nằm sau chương trình chính với lệnh kết thúc là RETI (trở về chương trình chính sau khi ngắt). Chương trình phục vụ ngắt, thực chất là một chương trình con, nhưng có lệnh kết thúc là RETI, khác với chương trình con là RET.

Chú ý: - Trong mọi chương trình, thường có lệnh nhảy tới chương trình chính ở địa chỉ 30H hay 100H.

- Trong chương trình của người dùng khi không có chương trình phục vụ ngắt nào, thì không phải kê khai địa chỉ vector ngắt và lệnh nhảy tới nhãn của chương trình phục vụ ngắt, mà chương trình bắt đầu từ địa chỉ 00H.

- Các chương trình phục vụ ngắt, chương trình con hoặc vùng dữ liệu, có thể nằm ở vùng nhớ khác với vùng nhớ của chương trình chính, khi đó, trước nhãn của chúng, phải có kê khai vùng nhớ với ORG XXXXH, với XXXXH là địa chỉ nhớ cần dùng.

10.1.3. Cho phép và cấm ngắt - thanh ghi IE

Trừ Reset, các ngắt khác phải được cho phép mới được quyền ngắt chương trình. Ở mặc định, VDK bị cấm ngắt chương trình. Thanh ghi cho phép ngắt IE (Interrupt Enable) có các bit như hình 10-2, sẽ cho phép ngắt khi các bit tương ứng với ngắt và bit cho phép ngắt toàn bộ EA (Enable All) được xác lập lên 1 và cấm ngắt khi có giá trị 0. Đây là thanh ghi xử lý bit, nó có thể xác lập (SETB) hay xoá (CLR) từng bit mà không ảnh hưởng tới các bit khác của thanh ghi.

Ở chương trình chính, khi khởi phát, chúng ta phải xác lập bit EA = 1 và các bit cho phép ngắt tương ứng với ngắt mong muốn bằng lệnh SETB IE.n (n là chỉ số của bit cho phép ngắt tương ứng).

	D7	D6	D5	D4	D3	D2	D1	D0
	EA	-	ET2	ES	ET1	EX1	ET0	EX0
Ký hiệu	EA	-	ET2	ES	ET1	EX1	ET0	EX0
Vị trí	IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0
Chức năng	Cho phép (EA = 1) hay cấm (EA = 0) tất cả các ngắt	Chưa sử dụng	Cho phép (ET2 = 1) hay cấm (ET2 = 0) ngắt TIMER2	Cho phép (ES = 1) hay cấm (ES = 0) ngắt trao đổi tin nối tiếp	Cho phép (ET1 = 1) hay cấm (ET1 = 0) ngắt do TIMER1	Cho phép (EX1 = 1) hay cấm (EX1 = 0) ngắt do ngắt ngoài INT1	Cho phép (ET0 = 1) hay cấm (ET0 = 0) ngắt do TIMER0	Cho phép (EX0 = 1) hay cấm (EX0 = 0) ngắt do ngắt ngoài INT0

Hình 10-2. Thanh ghi cho phép ngắt IE

10.1.4. Cờ ngắt

Trừ Reset, các ngắt xảy ra sẽ xác lập cờ ngắt tương ứng (xem bảng 10-1). Khác với cho phép/cấm ngắt có thanh ghi điều khiển IE riêng, cờ ngắt không có thanh ghi trạng thái riêng mà các bit cờ ngắt này nằm chung với các bit điều khiển của các thanh ghi điều khiển/trạng thái của Timer 0,1 (TCON), của Timer2 (T2CON.7) và của SCON cho ngắt nối tiếp.

Các cờ ngắt này được tự động xác lập lên 1 bằng phần cứng khi có ngắt xảy ra (do Timer đếm tràn, do ngắt ngoài, do có yêu cầu ngắt nối tiếp). Khi nhảy về vectơ ngắt, các cờ ngắt tự động xoá về 0, trừ ngắt nối tiếp với các cờ ngắt TI, RI phải được xoá bằng lệnh CLR bit (phần mềm).

Hình 10-3 mô tả các ngắt với các cờ ngắt tương ứng.

Chú ý: - Các ngắt ngoài INT0, INT1: (đưa tín hiệu vào chân P3.2 và P3.3 với một trong hai trường hợp sau:

+ **Đưa tín hiệu ngoài mức âm:** khi thanh ghi cho phép ngắt đã ghi IT0, IT1 = 1, tức ngắt do mức âm của tín hiệu ngoài.

+ **Đưa tín hiệu ngoài có sườn xung âm** (chân P3.2, P3.3 ở mức ban đầu là logic 1, xung vào mức logic 0) khi thanh ghi cho phép ngắt đã ghi ET0, ET1 = 1, tức ngắt do sườn xung âm của tín hiệu ngoài.

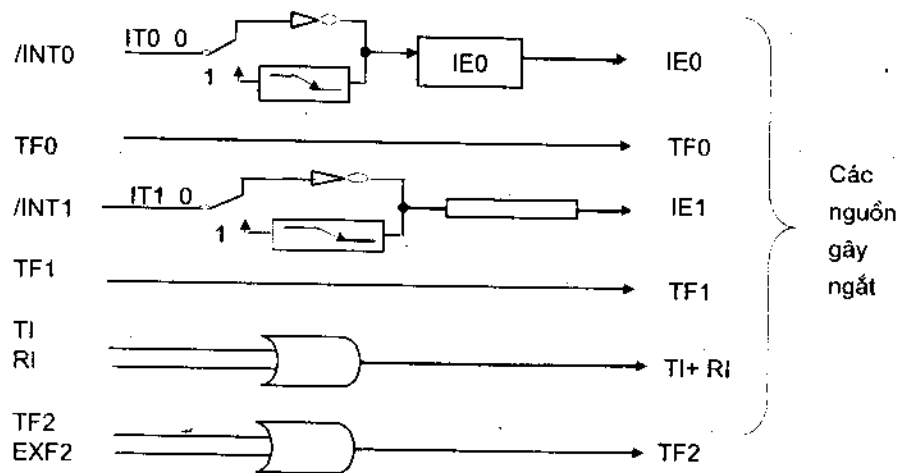
- **Các ngắt nối tiếp TI, RI** : chỉ gây ra một cờ ngắt nối tiếp nên chương trình phải xét tới do tín hiệu RI hay TI để nhảy tới phần chương trình nhận hay truyền của chương trình phục vụ ngắt nối tiếp.

- **Các ngắt TF2, EXF2:** chỉ gây ra một cờ ngắt do Timer2 đếm tràn TH2, TL2 hay do Counter2 đếm tràn các tín hiệu ngoài T2 vào chân P1.0 của bộ đếm và do tín hiệu ngoài T2.EX vào chân P1-1.

Các cờ ngắt này được dùng để:

+ Chỉ thị có ngắt tương ứng đã xảy ra và tự động nhảy tới chương trình con phục vụ ngắt tương ứng (do phần cứng của VDK).

+ Kiểm tra và nhảy tới chương trình phục vụ ngắt bằng lệnh nhảy chương trình (bằng phần mềm).



Hình 10-3. Các nguồn gây ngắt (hay các cờ ngắt)

10.1.5. Ưu tiên ngắt

Trong lập trình kiểm tra trạng thái thiết bị ngoài sẵn sàng theo nguyên tắc hỏi vòng trạng thái, thiết bị ngoài nào được kiểm tra trạng thái trước, nếu đã có yêu cầu (hay trạng thái đã sẵn sàng), sẽ được VDK (hay vi xử lý) phục vụ trước, tức ưu tiên hơn.

Trong VDK, nếu đồng thời có hai ngắt xảy ra, VDK sẽ:

- Ghi nhận các ngắt hay các yêu cầu ngắt chương trình.
- Lần lượt thực hiện các chương trình con phục vụ ngắt theo mức ưu tiên tự nhiên như ở bảng 10-2 với mức ưu tiên cao từ trên xuống dưới.

Bảng 10-2. Thứ tự ưu tiên ngắt

Mức ưu tiên từ cao xuống thấp	Địa chỉ trong ROM
RESET	0000H
Ngắt ngoài INT0 (phần cứng)	0003H
Ngắt Timer0 (TF0)	000BH
Ngắt ngoài INT1 (phần cứng)	0013H
Ngắt Timer1 (TF1)	001BH
Ngắt nối tiếp (RI, TI)	0023H
Ngắt Timer2 (TF2)	002BH

Chú ý: - Mức ưu tiên này cũng theo thứ tự kê khai của bảng các vector ngắt (bảng ghi địa chỉ các lệnh đầu tiên của chương trình con phục vụ ngắt) và theo trật tự từ cao xuống thấp.

- Ngoài sự ưu tiên mặc định như bảng 10-2 ở trên (theo một thứ tự cho trước khi chế tạo VDK), người ta có thể thiết lập một mức ưu tiên mới do người lập trình ghi vào thanh ghi ưu tiên IP (Interrupt Priority) để thay đổi mức ưu tiên như mặc định (do chế tạo).

Thanh ghi ưu tiên ngắt IP có địa chỉ B8H và có thể gọi địa chỉ theo bit để dễ dàng thiết lập mức ưu tiên cho từng ngắt. Hình 10-4 giới thiệu các bit của thanh ghi ưu tiên ngắt và hình 10-5 giới thiệu cấu trúc ngắt với điều khiển ưu tiên.

Thanh ghi ưu tiên ngắt IP này cũng tương tự thanh ghi cho phép ngắt IE về trật tự các bit, chỉ khác thay tên IE thành IP. Khi một bit nào đó ghi 1, nó có quyền ưu tiên hơn các ngắt khác, còn ở mặc định, các bit ưu tiên này có giá trị bằng 0. Khi không có bit nào của thanh ghi IP này ghi 1, sự ưu tiên theo thứ tự mặc định như bảng 10-2. Nếu ghi một bit IP nào đó lên 1 nhưng ngắt tương ứng lại xảy ra sau ngắt khác, thì ngắt khác vẫn xảy ra trước theo mặc định. Quyền ưu tiên này chỉ xảy ra khi có hai ngắt cùng phát sinh tại cùng một thời điểm. Khi có nhiều bit ưu tiên ngắt cùng ghi 1 ở thanh ghi IP và các ngắt cùng phát sinh tại một thời điểm, thì quyền ưu tiên cũng theo mặc định như bảng 10-2 hay theo thứ tự quyết của lệnh kiểm tra cờ ngắt của chương trình (hỏi vòng).

D7	D6	D5	D4	D3	D2	D1	D0
--	--	PT2	PS	PT1	PX1	PT0	PX0

----	IP.7	dự trữ
----	IP.6	dự trữ
PT2	IP.5	bit ưu tiên ngắt cho timer2 khi đếm tràn
PS	IP.4	bit ưu tiên ngắt cho cổng nối tiếp
PT1	IP.3	bit ưu tiên ngắt cho timer1
PX1	IP.2	bit ưu tiên ngắt cho ngắt ngoài INT1
PT1	IP.3	bit ưu tiên ngắt cho timer0
PX0	IP.2	bit ưu tiên ngắt cho ngắt ngoài INTO

Hình 10-4. Các bit của thanh ghi điều khiển ưu tiên ngắt IP

Chú ý là, trong nhiều ngắt chỉ có một ngắt ưu tiên trước, sau đó, sự ưu tiên theo:

- Sự xảy ra ngắt trước nếu hai ngắt xảy ra không đồng thời.
- Ưu tiên theo mặc định nếu hai ngắt xảy ra đồng thời và chưa ghi mức ưu tiên cho ngắt nào.
- Nếu hai ngắt xảy ra đồng thời và đã ghi mức ưu tiên cho ngắt có ưu tiên theo mặc định thấp hơn thì ngắt thấp hơn này được xử lý trước ngắt theo mặc định có mức ưu tiên cao hơn.
- Với các bit cho phép ngắt và ưu tiên ngắt riêng phần cho từng ngắt, ta có thể thử sự cho phép và cấm ngắt trong một chương trình cho trước (xem các ví dụ ở dưới).

10.2. CHƯƠNG TRÌNH CÓ NGẮT

Ở các chương trước (1 tới 9) chúng ta mới giới thiệu các chương trình chính, chương trình con không có ngắt. Khi VĐK có ngắt, chương trình chính sẽ thay đổi vì kê khai các vectơ ngắt, khởi động các phần thiết bị liên quan tới ngắt và có thêm chương trình phục vụ ngắt.

10.2.1. Chương trình chính có ngắt

Chương trình chính có ngắt có dạng tổng quát như ví dụ 10-1, trong đó có:

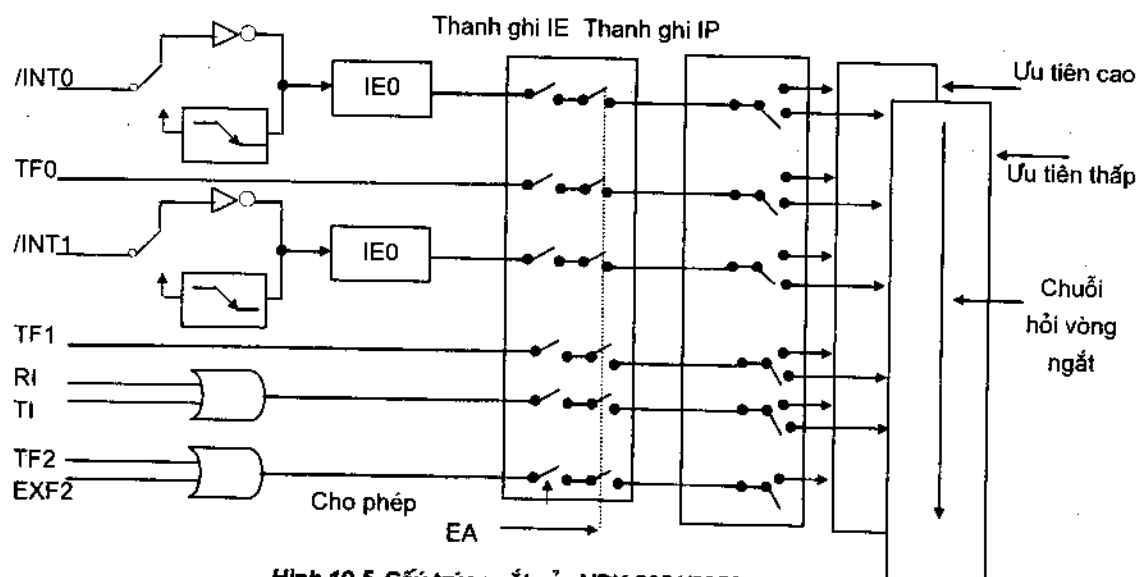
- *Kê khai vùng nhớ sau khi RESET* (bật nguồn nuôi hay ấn nút RESET): với địa chỉ gốc là 00H bằng ORG 00H và lệnh nhảy tới lệnh bắt đầu của chương trình chính với nhãn MAIN (hay Start) ở địa chỉ 100H (hay 30H) với lệnh LJMP MAIN.

- *Kê khai các vectơ ngắt*: hay các địa chỉ dành riêng và cố định cho các ngắt như trong bảng 10-1 hay ví dụ 10-1 với ORG XXH, trong đó XX là địa chỉ nhớ quy ước của ngắt.

Nếu chương trình không có ngắt nào, chúng ta không phải kê khai ngắt đó.

- *Kê khai vùng nhớ bắt đầu chương trình chính*: chương trình chính bắt đầu từ địa chỉ nhớ 100H (hay 30H) với ORG 100H (hay 30H) và nhãn MAIN (hay Start). Việc kê khai này mục đích là chương trình chính bỏ qua vùng vectơ ngắt, tránh ghi đè lên các các vectơ ngắt và vùng nhớ dành cho ngăn xếp bắt đầu từ địa chỉ 70H (với lệnh MOV SP, #70H).

- *Ghi các lệnh khởi phát*: đây là các lệnh chuẩn bị cho các hoạt động của ngăn xếp (ghi SP = #70H), của Timer0, 1, 2 (TMOD, TCON, TH0, TL0, TH1, TL1, TH2, TL2, RCAP2H, RCAP2L), của cổng trao đổi tin nối tiếp (SCON, tốc độ baud) và ngắt như cho phép ngắt (ghi vào thanh ghi TCON, T2CON) và ưu tiên ngắt (ghi vào thanh ghi IP) v.v... Ngoài ra, còn các lệnh khởi động Timer (TR0, TR1, TR2). Ví dụ 10-1 giới thiệu tổng quát các lệnh khởi phát của chương trình chính với các lệnh khởi phát các thanh ghi cho các hoạt động của Timer/Counter, trao đổi tin nối tiếp và ngắt. Nếu trong chương trình không có đầy đủ mọi hành động của VĐK và các ngắt, chỉ cần giữ lại các khai báo và các lệnh liên quan tới hành động của VĐK.



Hình 10-5. Cấu trúc ngắt của VDK 8051/8052

Ví dụ 10-1: Chương trình có các ngắt (ngoài 0, 1; timer0, 1, 2; nối tiếp)

```

ORG 00h      ; reset
LJMP MAIN    ; nhảy tới MAIN
ORG 03H      ; ngắt ngoài 0
LJMP EXINT0
ORG 0Bh      ; ngắt timer0
LJMP INTER0
ORG 13h      ; ngắt ngoài 1
LJMP EXINT1
ORG 1Bh      ; ngắt timer1
LJMP INTER1
ORG 23h      ; ngắt nối tiếp
LJMP SERINT
ORG 2Bh      ; ngắt timer2
LJMP INTER2
ORG 100H     ; chương trình chính
MAIN:
MOV TMOD, #22H ; chế độ 2
MOV TCON, #00H
MOV T2CON, #00H
MOV IE, #0BFh ; cho phép các ngắt
SETB TR0      ; đóng mạch timer0
SETB TR1      ; đóng mạch timer1
SETB TR2      ; đóng mạch timer2
SJMP $        ; chờ ngắt
EXINT0: JB P3.2, $ ; chờ tín hiệu ngoài 0
CPL P0.0      ; đảo bit P0.0
RETI          ; trở về chương trình chính
INTER0: CLR TR0; ngắt timer0 khi TF0 = 1
CPL P1.0
RETI          ; trở về chương trình chính
EXINT1: JB P3.3, $ ; chờ tín hiệu ngoài 1
JNB IE1, $
CPL P1.1
RETI          ; trở về chương trình chính
INTER1: CLR TR1; ngắt timer1 khi TF1 = 1
CPL P1.2; đảo bit
RETI          ; trở về chương trình chính
SERINT: JB RI, NHAN ; RI = 1, tới NHAN
JB TI, TRUY    ; TI = 1, tới TRUY
NHAN: CLR RI
MOV A, SBUF    ; nhận từ SBUF
MOVX @DPTR, A ; ghi vào @DPTR
JMP KT        ; tới KT
TRUYEN: MOV A, #41H ; truyền ký tự 'A'
MOV SBUF, A    ; ghi vào đệm
CLR TI         ; xóa cờ truyền
KT: RETI       ; trở về chương trình chính
INTER2: CLR TF2; ngắt timer2 khi TF2 = 1
CPL P2.0      ; đảo bit P2.0
RETI          ; trở về chương trình chính
END           ; kết thúc chương trình

```

- Kết thúc chương trình chính: kết thúc chương trình chính bao giờ cũng có lệnh Chờ tại chỗ (SJMP \$) để chờ ngắt xảy ra nếu không có lệnh quay vòng (LJMP LOOP) lại nhóm các lệnh của chương trình. Ngoài ra, còn lệnh END để kết thúc chương trình dịch.

10.2.2. Chương trình phục vụ ngắt

Chương trình phục vụ ngắt là một chương trình con với lệnh kết thúc là RETI (trở về từ ngắt) thay cho lệnh RET của chương trình con. Lệnh RETI này trở về đúng chỗ bị ngắt của chương trình chính và sẽ thực hiện nốt các lệnh của chương trình chính đã bị ngắt hay gián đoạn bởi chương trình con phục vụ ngắt. Chương trình con phục vụ ngắt này có thể nằm ở trong các ô nhớ dành cho vector ngắt hay ở cuối chương trình chính với nhãn tương ứng.

1. Chương trình phục vụ ngắt nằm trong các ô nhớ dành cho vector ngắt

Sau khai báo của địa chỉ ban đầu ORG XXH của ngắt tương ứng, nếu độ dài lệnh của chương trình phục vụ ngắt nhỏ hơn 8 địa chỉ nhớ để chương trình phục vụ ngắt này không tràn lên vector ngắt khác kế bên.

Ví dụ 10-2 giới thiệu chương trình phát xung nhờ ngắt của timer0 và chương trình phục vụ ngắt nằm trong vùng nhớ dành cho vector ngắt.

; Ví dụ 10-2: Chương trình có ngắt do Timer0 để phát xung

ORG 00H ; Reset	MAIN: MOV TMOD, #02H ; timer0, chế độ 2
LJMP MAIN ; nhảy tới MAIN	MOV TH0, #-50 ; kéo dài 50 microsec
ORG 0BH ; chương trình ngắt Timer0	MOV IE, #82H ; cho phép ngắt Timer0
CPL P1.0 ; đảo BIT P1.0	SETB TR0 ; đóng mạch Timer0
RETI ; trở về từ ngắt	SJMP \$; chờ ngắt
ORG 30H ; bắt đầu chương trình chính	END ; kết thúc chương trình dịch

2. Chương trình phục vụ ngắt nằm trong các ô nhớ sau chương trình chính

Khi độ dài địa chỉ của các lệnh của chương trình phục vụ ngắt lớn hơn 8 ô nhớ, chương trình này phải nằm sau chương trình chính và lệnh trong vector ngắt là lệnh nhảy tới nhãn là tên của chương trình phục vụ ngắt. Sau khai báo của vector ngắt ORG XXH phải có lệnh nhảy dài tới nhãn của chương trình phục vụ ngắt tương ứng. Khi có ngắt xảy ra (cờ ngắt xác lập lên 1), VDK tự động xoá cờ ngắt (trừ ngắt nối tiếp) rồi trở về địa chỉ dành cho chương trình phục vụ ngắt. Nếu chương trình phục vụ ngắt ngắn (dưới 8 địa chỉ nhớ), VDK thực hiện ngay chương trình phục vụ ngắt trong vùng vector ngắt. Nếu chương trình phục vụ ngắt dài hơn 8 địa chỉ nhớ, VDK thực hiện lệnh nhảy dài tới nhãn của lệnh đầu tiên của chương trình phục vụ ngắt nằm sau chương trình chính.

Ví dụ 10-3 giới thiệu chương trình phát xung có độ dài lớn, nằm sau chương trình chính.

Chương trình có hai ngắt của Timer0 (nằm trong vùng của vector ngắt) và Timer1 (nằm sau chương trình chính). Cả hai Timer đều ở chế độ 2 (lập lại). Hai Timer thay nhau xác lập và xoá bit P1.0. Như vậy, chương trình trên khiến VDK phát chuỗi xung hình chữ nhật có thời gian kéo dài (bit P1.0 xác lập) và thời gian nghỉ (bit P1.0 xoá về 0).

; Ví dụ 10-3: Phát một xung vuông dùng hai ngắt của hai timer0,1

ORG 00H ; reset	MOV TH1, #-200 ; kéo dài 200 microsec
LJMP MAIN ; nhảy tới MAIN	; tần số 200 kHz
ORG 0BH ; chương trình ngắt do Timer 0	MOV P1, #00H ; xoá cổng P1
CLR TR0 ; ngừng đếm TIMER0	SETB P1.0 ; xác lập bit P1.0
CLR P1.0 ; xoá bit P1.0	SETB TR0 ; đóng mạch Timer0
SETB TR1 ; đóng mạch Timer1	SJMP \$; chờ ngắt
RETI ; trở về từ ngắt	TISR: ; chương trình ngắt do Timer1
ORG 01BH ; chương trình ngắt do Timer1	CLR TR1 ; ngắt mạch Timer1
LJMP TISR ; nhảy tới chương trình ngắt	SETB P1.0 ; đảo bit P1.6, phát xung vuông
ORG 30H ; bắt đầu chương trình chính	; tần số 500 Hz
MAIN: MOV IE, #8AH ; cho phép ngắt 0, 1	SETB TR0 ; đóng mạch Timer0
MOV TMOD, #22H ; timer0,1 chế độ 2	RETI ; trở về từ ngắt
MOV TH0, #-100 ; kéo dài 100 microsec	END ; kết thúc chương trình dịch
; tần số 10kHz	

10.2.3. Xử lý ngắt của VDK

Khi có một nguyên nhân gây ngắt chương trình (ấn phím tạo ngắt ngoài ở chân INT0-P3.2 hay INT1-P3.3, đếm tràn các xung thời gian từ máy phát xung nhịp hay xung ngoài vào các chân T0-P3.4, T1-P3.5 hay nhận tín nối tiếp vào chân RXD-P3.0 hoặc truyền tín nối tiếp từ chân TXD-P3.1) và chương trình khởi phát đã cho phép ngắt, VDK sẽ xử lý các yêu cầu ngắt chương trình trên theo trình tự sau: *ghi lại các yêu cầu ngắt*: nếu chúng xảy ra ở cùng thời điểm để xử lý tất cả các yêu cầu ngắt lần lượt theo thứ tự ưu tiên (theo mặc định như bảng 10-1 hay theo ghi ưu tiên nhất ở thanh ghi IP).

Nếu + VDK đang rỗi: tức không thực hiện chương trình nào, VDK chuyển sang thực hiện chương trình phục vụ ngắt.

+ VDK đang chạy một chương trình: quá trình thực hiện chương trình ngắt diễn ra theo trình tự sau:

Kết thúc lệnh đang chạy.

VDK tiến hành thủ tục ngắt chương trình chính: nếu chương trình chính đang chạy, bằng cách:

+ Kết thúc lệnh đang thực hiện.

+ Ghi nhớ trạng thái của VDK ở thời điểm ngắt chương trình vào bộ nhớ ngăn xếp (địa chỉ trong SP tăng lên) như giá trị thanh ghi đếm chương trình PC, thanh ghi trạng thái PSW, thanh ghi tích lũy A.

+ Nhảy tới địa chỉ của ngắt tương ứng trong vùng vector ngắt, nạp địa chỉ này cho thanh ghi đếm chương trình PC và thực hiện chương trình phục vụ ngắt trong vùng vector ngắt (nếu chương trình ngắn hơn 8 ô nhớ) hay nhảy tới nhãn của chương trình phục vụ ngắt sau chương trình chính.

+ Tiếp tục chạy các chương trình phục vụ ngắt khác: đã ghi ở bộ xử lý ngắt, nếu có.

+ Trở về chương trình chính: khi gặp lệnh RETI ở cuối chương trình phục vụ ngắt sau cùng, VDK làm thủ tục trở về chỗ bị ngắt như sau:

+ Nhận lại các giá trị trạng thái lúc ngắt chương trình (PC, PSW và Acc) đã ghi trong ngăn xếp cho các thanh ghi trên.

+ Tiếp tục chạy chương trình chính ở chỗ bị ngắt.

Có thể có hiện tượng là chương trình phục vụ ngắt 1 chưa kết thúc, đã có yêu cầu ngắt của chương trình phục vụ ngắt 2 có mức ưu tiên hơn (ngắt trong ngắt), VDK tiến hành ngắt chương trình phục vụ ngắt 1, để thực hiện chương trình phục vụ ngắt 2, rồi tiếp tục thực hiện chương trình phục vụ ngắt 1 và chương trình chính.

Có thể có sự thực hiện chương trình ngắt một cách cưỡng bức, nghĩa là chưa có yêu cầu ngắt chương trình từ phần cứng (có tín hiệu ngắt ngoài, có cờ đếm tràn TF = 1 v.v...), nhưng cờ ngắt vẫn xác lập do lệnh xác lập cờ tràn, ví dụ SETB TF0. Khi đó, cần phải huỷ nguyên nhân gây ngắt cho tới thời điểm tác động và chuyển ngay sang chương trình phục vụ ngắt muốn ngắt cưỡng bức.

Chú ý: Chương trình trên có thể viết thành hai chương trình phục vụ ngắt ở sau chương trình chính.

10.3. NGẮT DO CÁC BỘ ĐỊNH THỜI/ĐẾM

Các ngắt này do các bộ đếm trong Timer/Counter 0, 1, 2 đếm các xung nhịp của bộ dao động thạch anh tràn từ 0FFFH trở về 0000H để tạo nên các cờ ngắt TF0, TF1, TF2. Với VDK 8051 chỉ có hai Timer0,1, còn VDK 8052 có thêm Timer2. Timer2 còn có thêm ngắt ngoài với lối vào T2EX (P1.1), còn ở chế độ đếm xung ngoài, đưa tín hiệu vào chân T2 (P1.0).

10.3.1. Đặc điểm của ngắt do Timer0,1

Các bộ định thời Timer0, Timer1 đều có chung các đặc điểm như nguồn phát xung đếm, chế độ đếm, giá trị đếm ban đầu, cho phép/cấm ngắt, ưu tiên ngắt, điều khiển bắt đầu đếm và cờ ngắt.

1. Nguồn xung đếm

Các bộ định thời Timer0,1 đều đếm các xung của máy phát xung nhịp thạch anh của VDK với tần số nhịp/12, cờ gần 1 MHz hay chu kỳ cỡ 1 μ sec (vì tần số 11,059 MHz hay 12 MHz). Theo mặc định, các bit C/T của thanh ghi TMOD bị xóa về 0, các bộ định thời này đếm các xung vào từ bộ dao động với tần số bằng tần số nhịp/12.

2. Các chế độ đếm

Tùy giá trị của các bit M0, M1 của thanh ghi TMOD, ta có 4 chế độ đếm khác nhau:

- **Chế độ 0:** Chế độ đếm 13 bit (5 bit thấp của thanh ghi TL, 8 bit của thanh ghi TH) của VDK 8048. Chế độ này ít dùng vì dung lượng đếm nhỏ.

- **Chế độ 1:** Chế độ đếm 16 bit của cả hai thanh ghi TL, TH. Chế độ này thường dùng cho Timer0,1.

- **Chế độ 2:** Chế độ đếm 8 bit ở thanh ghi TL và nạp lại giá trị của thanh ghi TH khi TL đếm tràn về 0. Người ta dùng chế độ này trong các trường hợp:

+ Đếm thời gian nhỏ hơn 256 μ sec (tối đa 8 bit) và tự nạp lại giá trị ban đầu.

+ Timer1 phát nhịp hay tốc độ baud cho trao đổi tin nối tiếp (xem chi tiết ở phần sau).

- **Chế độ 3:** Chế độ đếm 8 bit tách biệt dùng các thanh ghi TH0, TL0 để Timer0 trở thành hai bộ đếm tách biệt, độc lập với các cờ tràn TF0, TF1 (cho TH). Người ta dùng chế độ đếm này để tăng số lượng bộ đếm 8 bit (hai Timer0 thành hai Timer 8 bit với các ngắt TF0, TF1) và một Timer1 (16 bit) không có cờ tràn báo ngắt.

3. Giá trị ban đầu của TH1, TL1, TH0, TL0

Trước khi đếm, ta phải ghi giá trị ban đầu trên vào các thanh ghi đếm. Các bộ đếm sẽ đếm từ giá trị này tới FFFFH và trở về 0 với cờ báo tràn TF0, TF1 để báo ngắt chương trình. Do đó, số đếm là $N = (FFFFH - \text{giá trị ban đầu trên TH0, TL0, TH1, TL1})$. Để dễ tính số đếm cũng như thời gian trễ từ giá trị ban đầu tới 0000H, trong chương trình, người ta ghi số thời gian cần định trễ với dấu âm (-) đứng trước (xem ví dụ).

4. Cho phép/cấm ngắt

Theo mặc định, sau khi RESET (bật nguồn nuôi hay ấn nút RESET) các thanh ghi đều có giá trị 0000H (trừ các cổng P0-P3 có giá trị FFFFH). Do đó, sẽ cấm các ngắt, trong đó có ngắt do các bộ định thời Timer0,1. Muốn cho phép các Timer này ngắt chương trình khi đếm tràn về 0000H, ta phải: - **Xác lập bit EA:** của thanh ghi cho phép ngắt IE lên 1 bằng lệnh SETB EA hay lệnh MOV IE, #80H.

- **Xác lập các bit ET0 (bit D1), ET1 (bit D3):** của thanh ghi cho phép ngắt IE lên 1 bằng lệnh SETB ET0, SETB ET1 hay lệnh MOV IE, #8AH (chung cho cả EA = 1).

5. Cờ ngắt TF0, TF1 và địa chỉ ngắt

Khi đếm từ giá trị ban đầu tràn về 0000H, mỗi timer sẽ xác lập cờ tràn TF0, TF1. Các cờ tràn này sẽ gây ra ngắt chương trình chính khi đã ghi cho phép ngắt toàn bộ (EA = 1) và timer tương ứng (ET0 = ET1 = 1). Khi có cờ ngắt TF0 hay TF1 = 1, VDK làm thủ tục ngắt chương trình chính (nếu đang chạy), tự xóa cờ ngắt TF0, TF1 về 0 và nhảy tới địa chỉ dành cho các ngắt tương ứng:

0BH dành cho ngắt của Timer0

1BH dành cho ngắt của Timer1

VĐK sẽ nạp các lệnh bắt đầu của chương trình phục vụ ngắt ở những địa chỉ trên để thực hiện chương trình phục vụ ngắt. Khi gặp lệnh RETI (trở về từ ngắt) ở cuối chương trình phục vụ ngắt, VĐK làm thủ tục trở về chương trình chính, tại chỗ bị ngắt.

Việc ngắt chương trình có thể xảy ra trước khi đếm tràn nếu cờ ngắt được xác lập lên 1 bằng lệnh SETB TF0 hay SETB TF1 (ngắt cưỡng bức bằng lệnh hay ngắt mềm).

Ví dụ 10-4 giới thiệu chương trình phát hai xung vuông bằng ngắt do Timer0 và Timer1.

6. Ưu tiên ngắt

Theo mặc định, sau khi RESET (bật nguồn nuôi hay ấn nút RESET) thanh ghi ưu tiên ngắt IP bị xóa về 0. Các ngắt, nếu có các cờ được xác lập đồng thời sẽ có mức ưu tiên như ở bảng 10-2. Từ bảng trên, ta có ngắt do Timer0 có mức ưu tiên hơn ngắt do Timer1. Nếu muốn Timer1 có mức ưu tiên hơn Timer0, ta phải ghi bit IP3 hay PT1 lên 1 bằng lệnh SETB PT1.

Khi cho chạy chương trình của ví dụ 10-4, ta có thể kiểm tra sự ưu tiên ngắt của hai Timer0,1 theo mặc định và ghi sự ưu tiên vào thanh ghi IP khi cho hai số đếm TH0, TL0 = TH1, TL1 để đồng thời xảy ra hai ngắt của hai Timer0,1.

7. Điều khiển bắt đầu đếm xung

Các Timer0,1 chỉ bắt đầu đếm khi các bit TR0, TR1 của thanh ghi TCON được xác lập lên 1 bằng lệnh SETB. Do điều khiển các bit này, ta có thể cho hai Timer hoạt động đồng thời (bằng lệnh MOV TCON, #50H) hay liên tiếp (bằng lệnh CLR TR0, SETB TR1 hay ngược lại).

; Ví dụ 10-4: Phát hai xung vuông dùng hai ngắt của hai Timer0,1		MOV P0, #00H	; xoá cổng P0
		MOV P1, #00H	; xoá cổng P1
		SETB P1.0	; xác lập bit P1.0
		SETB TR0	; đóng mạch Timer0
		SETB TF1	; ngắt cường bức Time r1
		SJMP \$	; chờ ngắt
		TOISR: CPL P0.0	; tần số 100 kHz
		RETI	
		T1ISR:	
		MOV TH1, #HIGH (-1000)	; kéo dài
			; 1000 microsec,
			; tần số 1 kHz
		MOV TH1, #LOW (-1000)	;
		SETB TR1	; đóng mạch Timer1
		CPL P1.0	; phát ở P1.0
		RETI	; trở về từ ngắt
		END	; kết thúc dịch
ORG 00H ; reset			
LJMP MAIN ; nhảy tới MAIN			
ORG 0BH ; chương trình ngắt Timer0			
LJMP TOISR			
ORG 01BH			
LJMP T1ISR ; trở về từ ngắt			
ORG 30H ; bắt đầu chương trình chính			
MAIN: MOV TMOD, #12H			
MOV IE, #8AH ; cho phép ngắt Timer0,1			
MOV TMOD, #12H; timer1 chế độ 1,			
			; timer0 chế độ 2
MOV TH0, #-100 ; kéo dài 71 microsec,			
			; tần số 100 kHz

Chương trình trên có đặc điểm sau:

- Dùng hai ngắt do Timer0, Timer1 để phát hai xung vuông tại hai chân P0.0 và P1.0.
- Chế độ khác nhau: + Timer0 chế độ 2 với số đếm tràn là 200 micro sec hay tần số 50 kHz.
+ Timer1 chế độ 1 với số đếm tràn là 1000 microsec hay tần số 1 kHz.
- Timer1 có ngắt cường bức vì xác lập TF1 bởi lệnh SETB TF1 để ghi số đếm ban đầu vào TH1, TL1.
- Timer0 được ghi số đếm ban đầu khi khởi phát vào TH0.
- Hai Timer cùng hoạt động, lệnh chờ Sjmp \$ sẽ chờ timer0 sẽ ngắt 5 lần trong khi Timer1 mới ngắt chương trình 1 lần.

10.3.2. Đặc điểm của ngắt do Timer2

Timer2 là bộ định thời bổ sung thêm của VĐK 8052, ngoài hai Timer0,1. So với Timer0,1, Timer2 có những điểm tương tự sau:

- Cũng có các thanh ghi TH2, TL2 để ghi số đếm ban đầu.
- Cũng có riêng thanh ghi điều khiển T2CON với các bit TF2 (D7), EXF2, RCLJ, TCLK, EXEN2, TR2, C/T2 và CP/RL2C (D0) trong đó có các bit giống TCON là cờ ngắt TF2, cho phép ngắt EXEN2 và điều khiển ngắt TR2 và điều khiển đếm/định thời CT/T2.
- Các bit điều khiển phát xung nhịp baud cho cổng nối tiếp RCLK, RCLK: Timer2 làm máy phát xung nhịp baud truyền và nhận khi RCLK = TCLK = 1.
- Bit cờ ngắt ngoài EXF2: xác lập khi có tín hiệu ngoài vào lối vào T2X (P1.1). Bit này coi như một ngắt ngoài bổ sung cho Timer2, tương tự cờ ngắt ngoài khi có những tín hiệu vào /INT0, /INT1 của Timer0,1.
- Bit điều khiển thu nhận và nạp lại:
 - + Khi bit này bằng 0 (mặc định): tức ở chế độ nạp lại, dữ liệu từ các thanh ghi RCA2H, RCA2L được nạp lại vào TH2, TL2 khi đếm tràn hoặc trạng thái từ 1 sang 0 ở T2EX (P1.1) nếu EXEN = 1, tương tự chế độ 2 của Timer0,1 (với TH nạp lại cho TL), nhưng ở đây là 16 bit.
 - + Khi bit này bằng 1 (CP = 1): tức ở chế độ thu nhận, dữ liệu từ các thanh ghi TH2, TL2 được thu nhận vào các thanh ghi RCA2H, RCA2L (để đọc) khi đếm tràn trong TH2, TL2 từ FFFFH sang 0000H hay tín hiệu ở T2EX (P1.1) chuyển trạng thái từ 1 sang 0, nếu EXEN = 1.
- Trong thanh ghi cho phép ngắt IE cũng có bit cho phép ngắt ET2 dành cho Timer2.
- Trong thanh ghi ưu tiên ngắt IP cũng có bit cho ưu tiên ngắt PT2 dành cho Timer2.

Quá trình tạo ngắt cũng tương tự Timer0,1 nhưng với địa chỉ của chương trình phục vụ ngắt là 2BH. Nếu theo mặc định, mức ngắt của Timer2 này có mức ưu tiên thấp nhất (sau cả ngắt nối tiếp). Khi đã ghi PT2 và cho phép ngắt (EA, ET2), Timer2 này có mức ưu tiên cao nhất.

Ví dụ 10-5 giới thiệu chương trình có ba ngắt của Timer0,1 và Timer2. Bằng chạy thử chương trình với việc thay đổi các lệnh sau dấu chấm phẩy (;), ta có thể hiểu rõ thêm về ba ngắt của ba bộ định thời.

Chú ý: - Ví dụ 10-5 dùng lệnh gộp để dễ dàng thay các thông số số đếm ban đầu cho các timer 0, 1, 2. - Chương trình chính là vòng lặp để ghi số lần tăng R3.

- Các chương trình phục vụ ngắt ghi số lần tăng R0, R1, R2 tương ứng.

- Có thể chạy từng bước, cho các số đếm ban đầu THi, TLi như nhau và dùng các lệnh SETB PTi để kiểm tra thứ tự ưu tiên của các timer như lý thuyết.

; Ví dụ 10-5: Ngắt Timer0,1,2	
<pre> macro @baud01; <1.TH0?, 2.TL0? 3.TL1, 4.TH1 MOV TMOD, #22H ; chế độ 2 MOV TL0, #11 ; ghi thanh ghi TL1 MOV TH0, #12 ; ghi thanh ghi TH1 MOV TL1, #13 ; ghi thanh ghi TL1 MOV TH1, #14 ; ghi thanh ghi TH1 endmac macro @baud2 ; dùng timer2, chế độ 2 MOV T2CON, #00h MOV TL2, #1 ; ghi thanh ghi TL2 MOV TH2, #12 ; ghi thanh ghi TH2 MOV RCAP2L, #13 ; ghi thanh ghi TL2 MOV RCAP2H, #14 ; ghi thanh ghi TH2 endmac ORG 00h LJMP MAIN ORG 0BH LJMP NGTM0 ORG 1BH LJMP NGTM1 ORG 2BH LJMP NGTM2 ORG 100H MAIN: SETB ET0 ; cho phép ngắt Timer0 SETB ET1 ; cho phép ngắt Timer1 SETB ET2 ; cho phép ngắt Timer2 SETB EA ; cho phép ngắt toàn bộ ; SETB PT0 ; SETB PT1 ; ưu tiên ngắt cho timer1 SETB PT2 ; ưu tiên ngắt cho timer2 MOV R0, #00h MOV R1, #00h MOV R2, #00h </pre>	<pre> @baud01 low(-10), low (-10), low(-10), low (-10) ; TIMER0,1 chế độ 2 @baud2 low(-10), high(-10), low(-10), high(-15), 00h; LAP: MOV TCON, #50H ; TR0, TR1 = 1 SETB TR2 ; TR2 = 1 INC R3 MOV A, R3 MOV P3, A JMP LAP NGTM0: CLR TR0 CLR TR1 CLR TR2 INC R0 MOV A, R0 MOV P0, A RETI NGTM1: CLR TR1 CLR TR0 CLR TR2 INC R1 MOV A, R1 MOV P1, A RETI NGTM2: CLR TR2 CLR TR1 CLR TR0 INC R2 MOV A, R2 MOV P2, A RETI END </pre>

10.3. 3. Đặc điểm của các bộ đếm (Counter)

Các bộ đếm này cũng chính là các thanh ghi đếm TH, TL nhưng:

- Ghi bit C/T = 1 vào thanh ghi TMOD (cho Timer0,1) hay vào thanh ghi T2CON(cho Timer2).
- Các xung ngoài vào đếm đưa vào các lối vào Y0, T1, T2 tương ứng.

Các chi tiết khác của bộ đếm cũng giống như của Timer như đếm tràn tạo cờ ngắt TF0, TF1 (tự xóa khi nhảy tới ngắt), TF2 (không thể tự xóa khi nhảy tới ngắt), cho phép ngắt bằng xác lập các bit EA, ET0, ET1, ET2), ưu tiên ngắt (xác lập các bit PT1, PT2) v.v...

Điều khiển Timer bắt đầu đếm xung bằng xác lập các bit TR0, TR1, TR2 (bằng lệnh SETB TR).

Điều khiển Timer kết thúc đếm xung bằng xóa các bit TR0, TR1, TR2 (bằng lệnh CLR TR) hay đưa vào tín hiệu ngoài /INT0 hay /INT1 nếu trước đó đã ghi bit GATE = 1 vào thanh ghi TMOD (cho Timer0,1) hay T2CON (cho Timer2).

10.3.4. Các ứng dụng của các ngắt do Timer/Counter

Ở chương Timer/Counter, chúng ta đã xét một số ứng dụng của Timer/Counter không có ngắt. Ở đây, chúng ta xét thêm một số ứng dụng các Timer/Counter khi có ngắt. Điều đó có nghĩa là chúng ta sẽ sử dụng nhiều ngắt của Timer/Counter và cả các ngắt ngoài nữa.

1. Làm trễ một hành động

Vì một Timer có thể làm trễ một thời gian tùy ý (từ 1 μ sec tới ngày, tháng, năm) nên được dùng để làm trễ hành động của chương trình chính.

Muốn vậy: - Ở chương trình chính, người ta cho thời gian trễ ban đầu t_0 bằng cách ghi vào các thanh ghi TH, TL một giá trị tương ứng với trễ t_0 và đóng mạch Timer.

- Tại chỗ muốn làm trễ hành động, người ta cho thực hiện lệnh chờ ngắt (JNB TF, \$).

- Khi có TF = 1, chương trình nhảy tới chương trình trễ nào đó cho hành động tiếp theo do ngắt một thời gian nào đó với thời gian trễ t_r .

2. Tạo thời gian kéo dài cho hành động

Muốn vậy: - chương trình chính khởi phát Timer với thời gian ấn định cho hành động và cho thực hiện hành động tại thời điểm mong muốn bằng SETB TR rồi chờ thời gian kết thúc.

- Khi hết thời gian kéo dài, chương trình chính nhảy tới chương trình phục vụ ngắt do Timer và thực hiện lệnh dừng hành động của chương trình chính.

Có thể đặt sẵn thời gian kéo dài hành động TP từ bàn phím rồi so sánh thời gian của hành động TC, khi TC = TC, cờ so sánh CMFLAG xác lập, hành động được kết thúc (xem ví dụ 10-10).

Ví dụ 10-6 giới thiệu chương trình đếm số xung của Timer0 sau thời gian là 15 msec và trong thời gian 30 msec của Timer2. Ví dụ trên có sự so sánh thời gian ở Timer2 với thời gian định trước là 10 msec.

Ví dụ 10-6: Đếm xung của timer0, trễ và trong thời gian định sẵn cỡ msec ở Timer2

<pre> macro @NMSEC ; lệnh gộp Nmsec MOV RCAP2L, #low(-1000) ; giá trị nạp lại MOV RCAP2H, #high(-1000); T2 đếm 1 msec MOV TL2, #low(-1000) ; giá trị ban đầu MOV TH2, #high(-1000) ; MOV R0, #1 ; R0 tăng 1x1 msec LAP0: SETB TR2 ; khởi phát đếm msec JNB TF2, \$; chờ cờ tràn TF2 CLR TF2 ; xóa cờ tràn TF2 CLR TR2 ; dừng đếm timer2 INC A ; tăng số lần đếm MOV P0, A ; đưa số lần Nmsec ra P0 DJNZ R0, LAP0 ; giảm R0 tới 0, về LAP0 endmac ORG 00H ; Reset LJMP MAIN ; nhảy tới MAIN ORG 02BH ; chương trình ngắt Timer2 LJMP TIM2 ; nhảy tới TIM2 </pre>	<pre> ORG 100H ; bắt đầu chương trình chính MAIN: MOV TMOD, #01H; TIMER0 chế độ 1 MOV TH0, #00H MOV TLO, #00H @NMSEC 15 ; chờ trễ 15msec SETB TR0 ; khởi phát đếm cho timer0 @NMSEC 30 ; đếm trong 30msec SETB EA ; cho phép ngắt toàn cục SETB ET2 ; cho phép ngắt timer2 SETB TF2 ; tạo ngắt cưỡng bức timer2 SJMP \$; chờ đếm xong TIM2: CLR TR0 ; ngừng đếm timer0 CLR TF2 ; xóa TF2 MOV P1, TH0 ; đưa số đếm byte cao ra P1 MOV P2, TLO ; đưa số đếm byte thấp ra P2 RETI END ; kết thúc chương trình dịch </pre>
---	---

Muốn thời gian trễ và độ kéo dài lớn cỡ sec, ta phải tạo đơn vị thời gian cỡ sec bằng lệnh gộp dùng Timer2 kết hợp với vòng lặp dùng thanh ghi như ví dụ 10-7.

; Ví dụ 10-7: Đếm xung của timer0 trong thời gian định sẵn cỡ sec ở Timer2	
<pre> macro @NMSEC ; lệnh gộp N msec MOV RCAP2L, #low(-50000); MOV RCAP2H, #high(-50000); MOV TL2, #low(-50000); MOV TH2, #high(-50000); MOV R1, #1 ; R0 tăng 11x1msec LAP1: MOV R0, #21 ; tăng 50x20x1msec LAP0: SETB TR2 ; khởi phát đếm 50msec JNB TF2, \$; chờ cờ tràn TF2 CLR TF2 ; xoá cờ tràn TF2 CLR TR2 ; dừng đếm timer2 INC A ; tăng số lần đếm MOV P0, A ; đưa số lần Nmsec ra P0 DJNZ R0, LAP0; gim R0 tới 0, về LAP0 DJNZ R1, LAP1; gim R1 tới 0, về LAP1 endmac ORG 00H ; Reset LJMP MAIN ; nhảy tới MAIN </pre>	<pre> ORG 02BH ; chương trình ngắt Timer2 LJMP TIM2 ; nhảy tới TIM2 ORG 100H ; bắt đầu chương trình chính MAIN: MOV TMOD, #01H; TIMER0 chế độ 1 MOV TH0, #00H MOV TL0, #00H SETB TR0 ; khởi phát đếm cho timer0 @NMSEC 30 ; đếm trong 30msec SETB EA ; cho phép ngắt toàn cục SETB ET2 ; cho phép ngắt timer2 SETB TF2 ; tạo ngắt cường bức timer2 SJMP \$; chờ đếm xong TIM2: CLR TR0; ngừng đếm timer0 CLR TF2 ; xoá TF2 MOV P1, TH0 ; đưa số đếm byte cao ra P1 MOV P2, TL0 ; đưa số đếm byte thấp ra P2 RETI ; trở về từ ngắt END ; kết thúc chương trình dịch </pre>

3. Đồng hồ thời gian dùng ngắt

Chúng ta đã xét đồng hồ thời gian dùng Timer ở chương 8 về Timer/Counter. Ở chương này, chúng ta dùng Timer làm đồng hồ kết hợp với các ngắt để điều khiển như điều khiển Start/Stop, xóa số đếm, điều chỉnh phút, điều chỉnh giờ và báo hiệu thời gian.

Đặc điểm chung của mọi loại đồng hồ là:

- Dùng một Timer0 để đếm thời gian đơn vị (thường là 50 ms cho đồng hồ số hay 10 msec cho đồng hồ thể thao).
- Dùng một Timer0 để đếm thời gian đơn vị (thường là 50 ms cho đồng hồ số hay 10 msec cho đồng hồ thể thao).
- Dùng các vòng lặp với các biến đếm tăng (đếm tăng) hay giảm (đếm lùi) là các thanh ghi thông dụng (R0-R7) của cả 4 dải nhớ RAM bên trong (địa chỉ 00H- 32H) hay các ô nhớ đệm (địa chỉ từ 30H tới 70H), để đếm tăng lên 1sec, 10 sec, 60 sec, 10 phút, 60 phút và tới 24 giờ.
- Dùng các lệnh so sánh với #10 hay #60 để quyết định sự lặp lại hay tiếp tục cho các biến đếm.
- Dùng các ngắt ngoài /INT0, /INT1 để điều khiển khi ấn nút cho một tín hiệu âm vào các chân /INT0 (P3.2), /INT1 (P3.3). Tùy từng loại đồng hồ, ta có thể dùng các ngắt ngoài điều khiển.

a) Đồng hồ thể thao

Đồng hồ thể thao là thiết bị đếm thời gian chuẩn của Timer có:

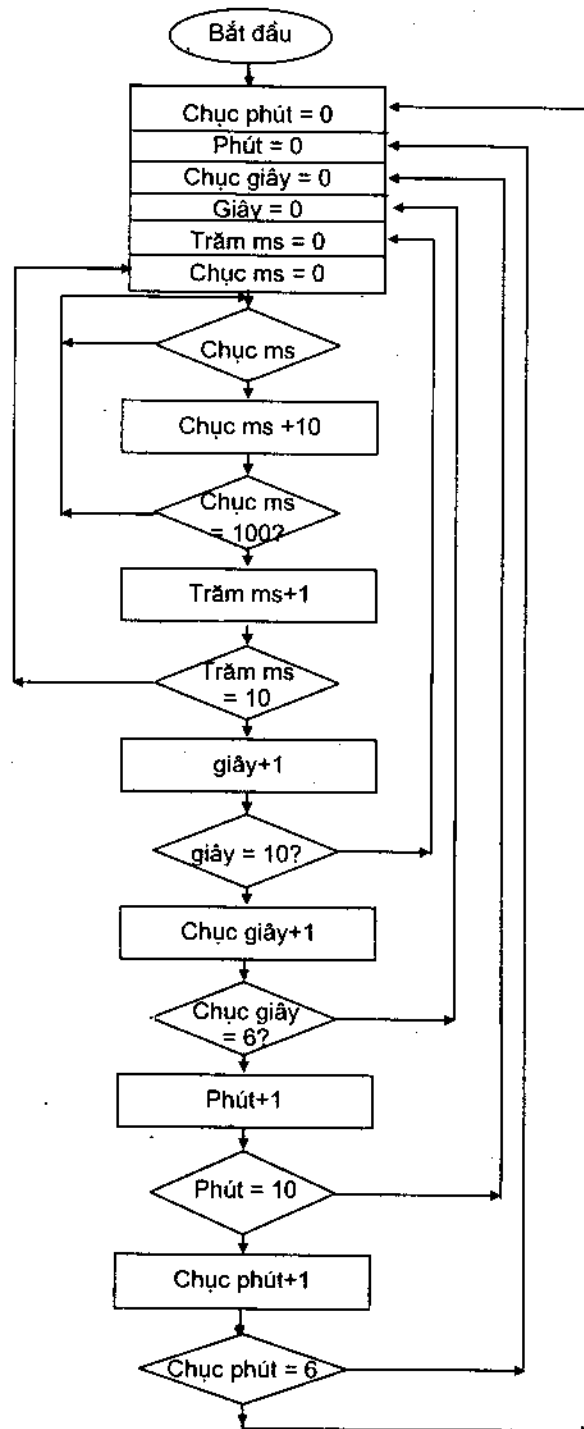
- Nút Start/Stop (bắt đầu chạy/dừng) dùng /INT0 và /INT1 để xóa số đếm đồng hồ thể thao.
- Ví dụ 10-8. giới thiệu chương trình cho đồng hồ thể thao với giải thuật ở hình 10-6 có những đặc điểm sau:

- Đơn vị thời gian chính xác hơn, tới 10msec.
- Dùng các bộ nhớ đệm làm các biến đếm cho chục ms, trăm ms, giây, phút, chục phút.
- Dùng thêm các ngắt ngoài /INT0 (điều khiển Start/Stop) và /INT1 (xóa số đếm).
- Dùng bit trạng thái 00H để điều khiển (= 1-chạy, 0-dùng đếm Timer0)
- Đồng hồ này khác với đồng hồ không dùng ngắt là:
 - + Trong khi chờ đếm tràn thời gian, đồng hồ giải mã và hiển thị số đếm trước đó, khi đếm tràn, có ngắt để tăng số đếm chục msec.
 - + Ngắt ngoài 0 khi có bấm nút mới bắt đầu đếm thời gian.
 - + Ngắt ngoài 1 khi có bấm nút để dừng đếm thời gian và xóa số đếm ghi trong các ô nhớ.

b) Đồng hồ số

Chương trình của đồng hồ số ở ví dụ 10-9 với giải thuật ở hình 10-7 có những đặc điểm sau:

- Dùng ngắt của Timer0 thay cho Timer2 (làm ví dụ) để đếm đơn vị thời gian là 1 sec, khác với đồng hồ thể thao có đơn vị thời gian là 10 msec.



Hình 10-6. Giải thuật của chương trình đồng hồ thể thao

; Ví dụ 10-8: Đồng hồ thể thao, Timer0 - đếm 10 msec

<pre> MACRO @DEM <1.ONH?, 2. GTRI?, 3.NHAN?> MOV A, 1 INC A DA A MOV 1, A CJNE A, # 2, 3 ENDMAC MACRO @GMA <ONHOGOC?, ONHOCAT?> MOV A, 1 ; ô nhớ MOVC A, @A+DPTR MOV 2, A ; địa chỉ nhớ cất mã ENDMAC ORG 00H LJMP MAIN ORG 0BH LJMP T0ISR ORG 30H CHUCPHUT EQU 35H PHUT EQU 34H CHUCGIAY EQU 33H GIAY EQU 32H TRMSEC EQU 31H CHMSEC EQU 30H QUET EQU P2 LED7 EQU P0 ORG 100H MAIN: MOV DPTR, #MA7DOAN MOV TMOD, #01H CLR TF0 MOV IE, #82H MOV TH0, #HIGH(-10000) MOV TL0, #LOW(-10000) SETB TR0 CHPHT: MOV CHUCPHUT, #00H PHT: MOV PHUT, #00H CHGY: MOV CHUCGIAY, #00H GY: MOV GIAY, #00H TRMS: MOV TRMSEC, #00H CHMS: MOV CHMSEC, #00H </pre>	<pre> LAP0 :LCALL GMA LCALL HTHI MOV A, CHMSEC CJNE A, #11, LAP0 @DEM TRMSEC, 10H, CHMS; trăm msec @DEM GIAY, 10H, TRMS ; sec @DEM CHUCGIAY, 6H, GY ; chục sec @DEM PHUT, 10H, CHGY ; phút @DEM CHUCPHUT, 6H, PHT; chục phút SJMP CHPHT T0ISR: INC CHMSEC MOV TH0, #HIGH(-10000) MOV TL0, #LOW(-10000) CLR TF0 RETI GMA: @GMA CHMSEC, 40H @GMA TRMSEC, 41H @GMA GIAY, 42H @GMA CHUCGIAY, 43H @GMA PHUT, 44H @GMA CHUCPHUT, 45H RET HTHI: MOV B, #01111111B ; mã quét MOV R0, #40H HTH1: MOV LED7, @R0 MOV A, B MOV QUET, A LCALL DELAY MOV QUET, #0FFH INC R0 RR A MOV B, A MOV A, R0 CJNE A, #45H, HTH1 RET DELAY: MOV R7, #0FH DJNZ R7, \$ RET MA7DOAN: DB 0C0H, 0F9H, 0A4H, 0B0H, 99H, DB 92H, 82H, 0F8H, 80H, 90H END </pre>
--	--

- Dừng ngắt đếm tràn của Timer0, thay cho nhiều lệnh đếm của Timer0 của chương trình không dùng ngắt (chương 8).

- Có thể thêm các ngắt ngoài INT0, INT1 để chỉnh phút và giờ (xem các ví dụ ở dưới cho ngắt ngoài). Mỗi lần tác động tín hiệu ngoài (bằng ấn nút nối với các chân P3.2 và P3.3), sẽ tạo ra ngắt ngoài để chạy các chương trình ở ví dụ 10-11 hay ví dụ 10-12.

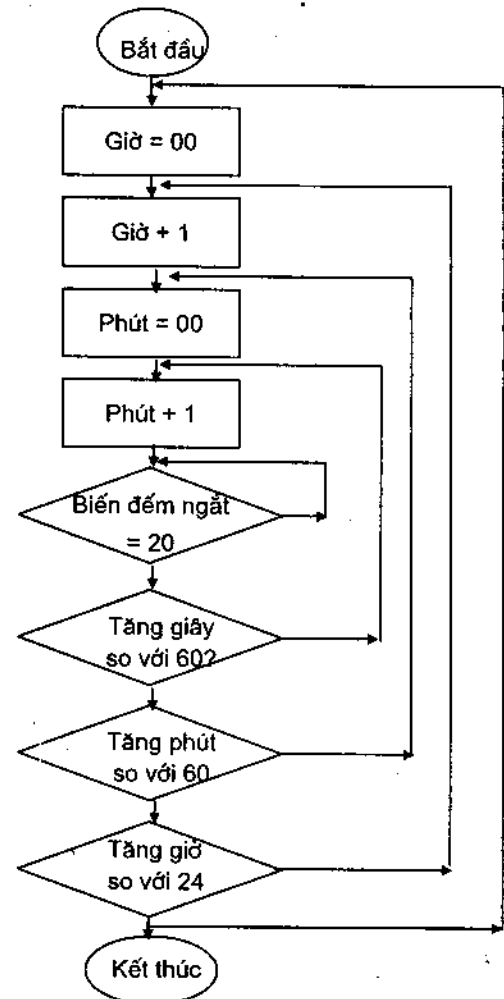
Các chương trình này là những chương trình đếm phút hay giờ, dùng để đặt lại (chỉnh) phút hay giờ nếu đồng hồ chạy sai so với thời gian thực. Mỗi lần ấn nút để chỉnh, chỉ có thể làm tăng

thời gian lên 1 phút hay 1 giờ. Số phút hay giờ mới này sẽ thay thế số phút hay giờ cũ trong các ô nhớ tương ứng. Nếu cần chỉnh nhiều phút hay nhiều giờ, ta phải nhấn bấy nhiêu lần. Nếu muốn chỉnh số phút hay số giờ giảm đi, ta phải ấn nút chạy tăng lên tới 60 rồi tăng tới số phút hay giờ cần chỉnh.

Ví dụ 10-9: Đồng hồ số với giây (đơn vị, hàng chục), phút (đơn vị, hàng chục), giờ

```
MACRO @DEM <1.ONH?, 2.GTR?, NHAN?>
    MOV A, |1
    INC A
    DA A
    MOV |1, A
    CJNE A, #|2, |3
ENDMAC
MACRO @GMA <ONHOGOC?, ONHOCAT?>
    MOV A, |1 ; ô nhớ
    MOVC A, @A+DPTR
    MOV |2, A ; địa chỉ nhớ cất mã
ENDMAC
ORG 00H
LJMP MAIN
ORG 0BH
LJMP T0ISR
ORG 30H
CHUCGIO EQU 36H
GIO EQU 35H
CHUCPHUT EQU 34H
PHUT EQU 33H
CHUCGIAY EQU 32H
GIAY EQU 31H
MSEC EQU 30H
QUET EQU P2
LED7 EQU P0
ORG 100H
MAIN: MOV DPTR, #MA7DOAN
MOV TMOD, #01H
CLR TF0
MOV IE, #82H
MOV TH0, #HIGH(-50000)
MOV TL0, #LOW(-50000)
SETB TR0
CHGIO: MOV CHUCGIO, #00H
G: MOV GIO, #00H
CHPHT: MOV CHUCPHUT, #00H
PHT: MOV PHUT, #00H
CHGY: MOV CHUCGIAY, #00H
GY: MOV GIAY, #00H
LAP1: MOV MSEC, #00H
LAP0: LCALL GMA
LCALL HTHI
```

```
MOV A, MSEC
CJNE A, #21, LAP0
@DEM GIAY, 10H, LAP1 ; sec
@DEM CHUCGIAY, 6H, GY ; chục sec
@DEM PHUT, 10H, CHGY ; phút
@DEM CHUCPHUT, 6H, PHT ; chục phút
@DEM GIO, 6H, CHPHT ; giờ
@DEM CHUCGIO, 3H, G ; chục giờ
SJMP CHPHT
T0ISR: INC CHMSEC
MOV TH0, #HIGH(-10000)
MOV TL0, #LOW(-10000)
CLR TF0
RETI (xem tiếp ở phần tương ứng của ví dụ 10-8)
```



Hình 10-7. Giải thuật của đồng hồ số

- Có thể thêm các chương trình giải mã và hiển thị trên đèn LED 7 đoạn (xem chương 11 hay chương 12 và 13) để chỉ thị nội dung các ô nhớ chứa thời gian (giây, phút, giờ) trong khi chờ đếm tràn.
- Có thể chia nhỏ thang đo thời gian theo đơn vị giây, chục giây, đơn vị phút, chục phút, đơn vị giờ.
- Có thể thêm ô nhớ chứa ngày, thứ trong tuần, tháng, năm để có một lịch điện tử.
- Có thể thêm dùng ngắt ngoài /INT0 để chỉnh phút, /INT1 để chỉnh giờ sau một thời gian sai lệch và đặt lại giờ.

Các ví dụ chỉnh phút (ví dụ 10-11) và chỉnh giờ (ví dụ 10-12) thực chất là chương trình đếm tăng một phút hay một giờ và thay thế số phút hay giờ trong ô nhớ tương ứng của chúng. Các chương trình này được thực hiện mỗi khi nhấn nút tương ứng để tạo ra các ngắt ngoài INTO (đưa tín hiệu ngoài vào chân P3.2 cho chỉnh phút) hay INT1 (đưa tín hiệu ngoài vào chân P3.3 cho chỉnh giờ). Sau khi chỉnh, chương trình chính lại trở về chỗ bị ngắt của chương trình đồng hồ. Có thể, trong khi đang chỉnh thời gian, có ngắt đếm thời gian của Timer0, chương trình ngắt chỉnh thời gian này lại bị ngắt để trở về đếm thời gian vì có mức ưu tiên hơn và sau đó lại tiếp tục chỉnh.

c) Đồng hồ báo thời gian

Đồng hồ báo thời gian là đồng hồ số có thêm bộ phận ghi nhớ, so sánh thời gian đặt trước và thời gian thực của đồng hồ và báo hiệu khi hai thời gian bằng nhau bằng chuông hay loa. Ta có thể đặt sẵn thời gian cần báo hiệu với GIAY, PHUT, GIO. Khi thời gian đếm TCH, TCL bằng thời gian đặt TPH, TPL đã ghi nhớ trong hai byte đồng hồ sẽ xác lập cờ ngắt COMFLAG để chạy chương trình báo hiệu, phát chuỗi xung ra loa hay reo chuông điện tử.

Chương trình con so sánh thời gian đặt TPH, TPL với thời gian đếm TCH, TCL như ví dụ 10-10 và giải thuật như hình 10-7.

Một số loại đồng hồ báo hiệu (báo thức) có bán trên thị trường, có những đặc điểm sau:

* *Về cấu tạo:* - Có lịch chỉ thị cả thứ trong tuần (từ thứ hai, ba tới chủ nhật).

- Có lịch chỉ thị cả ngày, tháng năm.

- Có pin hay ắc quy nuôi đồng hồ thời gian trong bộ nhớ RAM hay nhớ NVRAM để khi mất điện nuôi VDK, đồng hồ vẫn chạy mà không chỉ thị thời gian. Khi có điện nuôi, sẽ chỉ thị thời gian thực.

Như vậy, ta phải đếm thêm ngày, thứ, tháng, năm với các bộ nhớ đếm tương ứng (giống ví dụ 10-9).

* *Về điều khiển:* có các nút ấn điều khiển sau:

1. *Điều khiển chuyển các phím:* hay thực đơn có 7 nút: điều khiển chung (nút 1), chuyển loại thời gian (nút 2), chuyển dữ liệu (nút 3), điều khiển ghi dữ liệu mới (nút 4), điều khiển chỉnh thời gian (nút 4), điều khiển ghi thời gian báo hiệu (nút 6), điều khiển trao đổi tin với máy vi tính PC hay ghi chương trình (nút 7).

2. *Điều khiển chuyển thời gian hay địa chỉ của các ô nhớ dữ liệu:* về phút (2 địa chỉ), giờ (2 địa chỉ), ngày (2 địa chỉ), thứ (1), tháng (2 địa chỉ) tổng cộng 9 địa chỉ.

3. *Điều khiển chuyển dữ liệu:* cho các giá trị từ 0 tới 9 theo thứ tự số lần ấn nút.

4. *Điều khiển chỉnh thời gian:* thay giá trị mới vào giá trị cũ có trong ô nhớ tương ứng.

5. *Điều khiển xác lập dữ liệu về thời gian cần báo hiệu (báo thức):* xác lập các giá trị về thời gian cần báo hiệu vào ô nhớ đếm.

6. *Điều khiển ghi dữ liệu về thời gian cần báo hiệu (báo thức):* ghi thời gian báo hiệu trên vào thời gian cần so sánh để báo hiệu. Có thể ghi nhiều thời gian báo hiệu trong một ngày, tháng.

7. *Điều khiển ghi chương trình từ máy vi tính PC sang VDK:* khi ấn nút này, có sự trao đổi tin nối tiếp giữa PC và VDK.

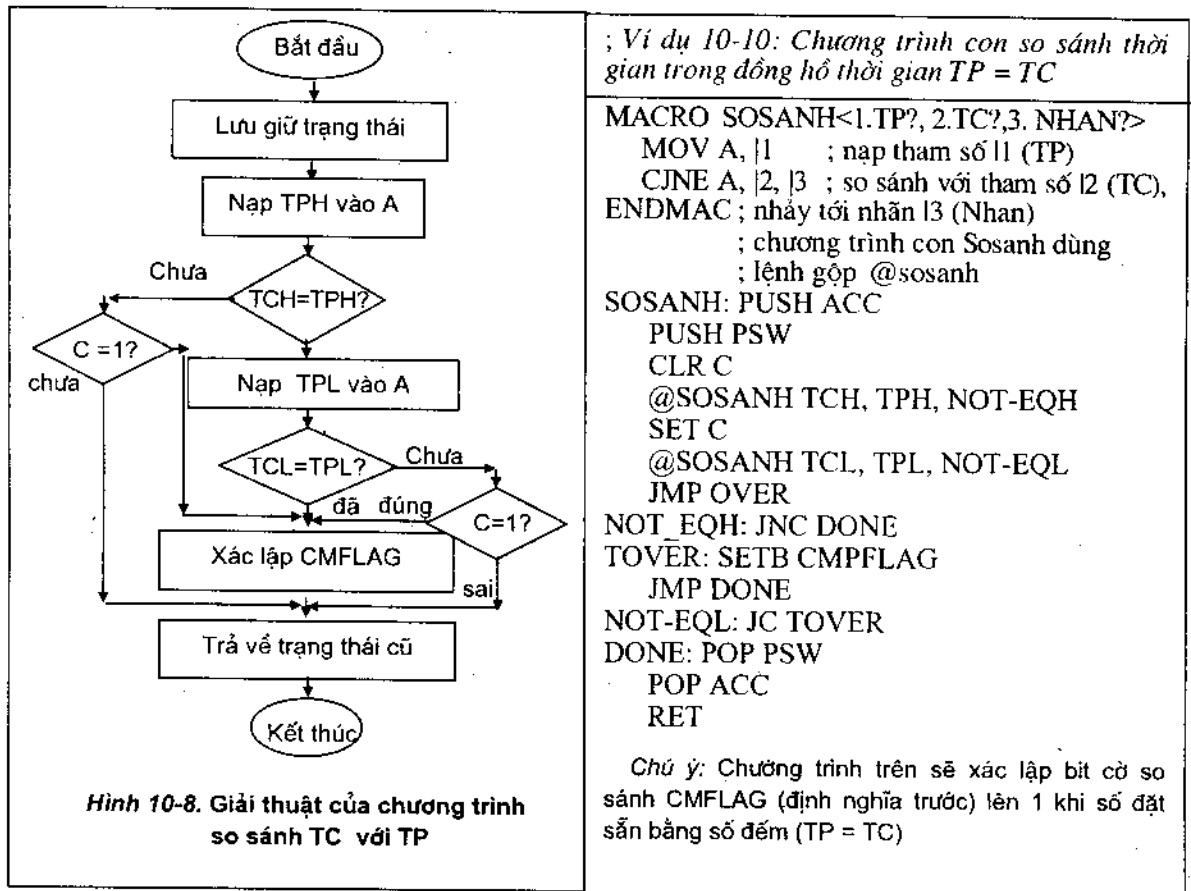
* *Về chương trình:* có các loại chương trình sau:

- *Chương trình thời gian cơ bản:* chương trình tạo thời gian đơn vị 20microsec hay 50 microsec dùng Timer0, ghi trên RAM ngoài (nuôi bằng pin hay ắc quy) hay NVRAM.

- *Chương trình thời gian:* cho giây (đơn vị, hàng chục), phút (đơn vị, hàng chục), giờ (đơn vị, hàng chục), ngày (đơn vị, hàng chục), thứ (đơn vị), tháng (đơn vị, hàng chục).

- *Chương trình giải mã và hiển thị:* chương trình giải mã 7 đoạn và hiển thị đèn LED 7 đoạn theo kiểu dồn kênh.

- *Chương trình thực đơn:* phục vụ phím điều khiển chung.



Hình 10-8. Giải thuật của chương trình so sánh TC với TP

- Chương trình phục vụ các phím: để tạo dữ liệu bằng số lần ấn nút điều khiển, điều khiển ghi nhớ dữ liệu đếm, chương trình nhận biết báo hiệu bằng so sánh thời gian đặt với thời gian thực, chương trình phát tín hiệu ra loa v.v... Ví dụ 10-13 giới thiệu một số chương trình con cần cho đồng hồ báo hiệu trên.

Chú ý: Chương trình trên dùng chương trình phục vụ ngắt thời gian Timer2 để đếm tới 1 sec cho thời gian đếm TC (TCH, TCL) để so sánh với thời gian đặt sẵn TP (TPH, TPL). Trong chương trình chính, các dữ liệu dành cho đếm bộ Timer2 ở chế độ nạp lại (từ RCAP2H, RCAP2L sang TH2, TL2) với thời gian tràn 10 msec. Mỗi lần đếm tràn được 1 msec và cần tới 100 lần, TCH, TCL cũng tăng 100 lần.

- Chương trình so sánh CMPTPTC so sánh số thời gian định sẵn TP với thời gian đếm TC. Khi chúng bằng nhau, cờ so sánh CMPFLAG xác lập và nhảy tới kết thúc (khác với chương 8 không dùng cờ). Khi kết thúc, đọc các giá trị thời gian, số đếm NNNNH ra các cổng P0-P3.

- Có thể đổi chương trình trên thành chương trình đếm xung ngoài theo số đếm NNNN định sẵn. Khi đó, có sự so sánh byte cao của số đếm định sẵn với TH0 và byte thấp với TL0. Khi có cờ so sánh xác lập, sẽ dùng cả hai bộ đếm và đọc thời gian đếm trong TH2, TL2 của Timer2. Chương trình con so sánh CMPNTHTL này sẽ đơn giản hơn (xem chương 8).

4. Phát xung

Chúng ta đã xét việc phát xung không dùng ngắt và Timer. Ở đây, chúng ta xét sự phát xung dùng nhiều ngắt do Timer tạo ra. Khác với phát xung ở trên là ở đây có thể phát nhiều dạng xung với độ trễ, độ kéo dài, độ rộng khác nhau vì dùng nhiều Timer điều khiển.

Với cách phát xung dùng các ngắt của Timer của VĐK này, ta có thể phát được các dạng xung của vi mạch 8253 ở các chế độ và các dạng xung hay nhiều xung mong muốn khác.

Có thể kể: - *Phát xung với độ trễ và độ kéo dài khác nhau:*

Muốn vậy, chương trình chính dùng một Timer (ví dụ Timer0 trong ví dụ 10-2).

Chương trình phục vụ ngắt dùng Timer1 để tạo độ kéo dài xung t_0 .

<i>: Ví dụ 10-11: Chương trình chỉnh phút, chục phút dùng ngắt ngoài INT0</i>	<i>; Ví dụ 10-12: Chương trình chỉnh giờ, chục giờ dùng ngắt ngoài INT1.</i>
NGATTO: MOV IE, #80H INC 15H MOV A, 15H CJNE A, #10, NGAT1 MOV 15H, #00H INC 14H MOV A, 14H CJNE A, #6, NGAT1 MOV 14H, #00H NGAT1: LCALL GIAIMA LCALL HIENITHI CLR TFO SETB TR0 RETI GIAIMA: ; chương trình con giải mã RET HIENITHI: ; chương trình con hiển thị RET END	NGATT1: MOV IE, #80H; cấm ngắt INT1 INC 13H ; tăng đơn vị giờ MOV A, 13H ; gửi vào A CJNE A, #03, NGAT11 ; so với 3, tới NGAT11 MOV 12H, #00H ; xóa chục giờ MOV 13H, #1 ; tăng giờ NGAT11: CJNE A, #10, NGAT13; so 10 giờ MOV 12H, #1 ; tăng chục giờ MOV 13H, #00 ; xóa giờ NGAT13: LCALL GIAIMA; gọi giải mã LCALL HIENITHI ; gọi hiển thị CLR TFO ; xóa cờ TFO SETB TR0 ; xác lập TR0 RETI GIAIMA: ; chương trình con giải mã RET HIENITHI: ; chương trình con hiển thị RET END

- *Xung tuần hoàn (đa hài)*: với độ trễ khác nhau, độ kéo dài bằng độ rộng (thời gian nghỉ) và cho chương trình lặp lại theo vòng LOOP bằng lệnh LJMP LOOP.

Ví dụ 10-14 là chương trình phát xung vuông tần số 50 Hz (hình 10-14) và sau thời gian đếm 23FFH (có thể thực hiện việc phát xung vuông này ở chương trình chính). Đây là chương trình ngắt có kích thước nhỏ, nằm trong chính 8 byte nhớ của vector ngắt dành cho ngắt của Timer.

Từ chương trình này, ta có thể tạo:

- + *Xung tuần hoàn (đa hài)*: với thời gian trễ, độ kéo dài t_0 và thời gian rộng khác nhau.
- + *Xung đơn (đơn hài)*: với độ trễ khác nhau (thay đổi giá trị của TH0, TH1) và độ rộng khác nhau (thay đổi giá trị TH1, TL1) khi cho chương trình chạy một lần.

Khi đó, ta phải dùng thêm Timer2 để định thời gian nghỉ cho xung phát ra.

Với các ví dụ trên, ta có thể viết chương trình phát nhiều xung với các đặc tính khác nhau như:

- Chuỗi xung đơn có cùng độ rộng và trễ liên tiếp với nhau ở các chân của VDK trong máy phân tích số.
- Chuỗi xung đơn có cùng thời điểm phát, nhưng với độ rộng khác nhau.
- Chuỗi xung tuần hoàn có cùng thời điểm phát, nhưng với độ rộng khác nhau, tần số lặp lại khác nhau.
- Xung đơn hài, xung tuần hoàn có điều khiển bằng tín hiệu ngoài /INT0 đưa vào chân P3.2.

5. Ngắt do đếm xung

Bộ đếm chính là bộ định thời Timer, nhưng có hai điểm khác là:

- Ghi vào thanh ghi điều khiển TMOG với bit GATE = 1 (điều khiển xung ngoài vào) và C/T = 1 cho chế độ đếm cho timer T0,1 và bit C/T2 vào T2CON cho timer2.
- Xung vào đếm vào một trong hai lối vào đếm T0 (chân P3.4-số 14), T1 (chân P3.5-số 15) cho Timer0,1 và T2 (chân P1.0-số 1) cho Timer2.

Hoạt động ngắt của bộ đếm cũng như của bộ định thời. Khi đếm tràn (các thanh ghi TH, TL về 0) sẽ xác lập TFO, TF1, TF2 để gây ngắt chương trình nếu đã cho phép ngắt và đọc số đếm hay thời gian đếm. Có thể có bốn loại đếm:

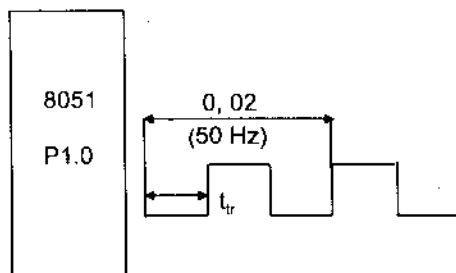
- *Đếm với thời gian cho trước ghi từ bàn phím vào bộ nhớ đệm*: ghi sẵn vào bộ đệm TPH, TPL để so sánh với số đếm TCH, TCL. Khi hai thời gian TP = TC, bộ đếm dừng đếm.

; Ví dụ 10-13: Một số chương trình con cho đồng hồ báo hiệu

GIAIMA: ; chương trình con giải mã 7 đoạn MOV DPTR, #BANGMA; gán đ/c bảng mã MOV A, GIAY ; GIAY- ô nhớ chứa giây ANL A, #0FH ; giữ 4 bit thấp MOVC A, @A+DPTR; đọc bảng mã 7 đoạn MOV 27H, A ; gửi vào ô nhớ 27h MOV A, GIAY ; GIAY- ô nhớ chứa giây ANL A, #0FH0 ; giữ 4 bit cao SWAP A ; đổi sang 4 bit thấp MOVC A, @A+DPTR ; đọc bảng mã MOV 26H, A ; gửi vào ô nhớ 27h RET BANGMA: DB 0C0H, 0F9H, 0A4H, 0B0H, DB 99H, 92H, 82H, 0F8H, 0F8H, 80H, 90H HTHI: ; chương trình con hiển thị MOV A, #01111111B; xóa D7 MOV R0, #27H ; gửi địa chỉ nhớ 27h HT1: MOV LED7, @R0; đưa ô nhớ ra LED7 MOV QUET, A ; đưa A ra QUET LCALL DELAY ; gọi trễ MOV QUET, #0FFH ; tắt các LED DEC R0 ; giảm R0, ô nhớ 26H RR A ; dịch phải CJNE R0, #25H, HT1; hiển thị ô nhớ 26H RET DELAY: MOV R7, #0FH DJNZ R7, \$ RET THUCDON: ; thực đơn với các số '0, 1, 2,...9' JNB KEYFLAG ; chờ ấn phím CLR KEYFLAG ANL KB_DATA, #0FH CALL KB_CONVER MOV A, KB_DATA XRL A, '0' JZ NUM1 MOV A, KB_DATA XRL A, '1' JZ NUM1 MOV A, KB_DATA XRL A, '2' JZ NUM2 MOV A, KB_DATA	XRL A, '3' JZ NUM3 NUM1: ; chương trình con phục vụ núm 1 ; các lệnh của chương trình con NUM1 RET NUM2: ; chương trình con phục vụ núm 2 ; các lệnh của chương trình con NUM2 RET NUM3: ; chương trình con phục vụ núm 3 ; các lệnh của chương trình con NUM3 RET COMPTPC: ; so số báo hiệu TP và số đếm TC PUSH ACC ; nếu bằng nhau, xác lập cờ PUSH PSW ; CMPFLAG CLR C ; xóa C MOV A, TPH; gửi số báo hiệu TPH vào A CJNEA, TCH, NOTEQH; nhảy nếu chưa bằng CLR C ; xóa C MOV A, TPL; gửi số báo hiệu TPL vào A CJNE A, TCL, NOTEQL; nhảy nếu chưa bằng JMP TOVER ; tới TOVER NOTEQH: JNC DONE; nhảy nếu C = 0 TOVER: SETB CMPFLAG; xác lập cờ bằng JMP DONE ; nhảy tới DONE NOTEQL: JC TOVER; nhảy nếu C = 1 DONE: POP PSW POP ACC RET ; chương trình ghi giá trị núm đã nhấn vào ô ; nhớ dành cho giá trị TTHOU (chục ngàn) KB_SET: ANL KB_DATA, #0FH; lấy 4 bit CALL KB_CONVER ; gọi biến đổi núm MOV A, KB_DATA ; đọc dữ liệu núm MOV TTHOU, A ; gửi giá trị núm RET ; chương trình hiển thị dữ liệu của núm nhấn SET: JNB KEYFLAG, SET ; chờ ấn núm CLR KEY_FLAG ; xóa cờ ấn núm CALL KB_SET ; gọi xác lập núm CALL GIAIMA ; gọi giải mã CALL HTHI ; gọi hiển thị RET END
---	---

- Đếm với thời gian cho trước, ghi vào bộ định thời khác: thời gian được ấn định trước cho hành động đếm. Khi tới thời gian ấn định trước, sẽ gây ngắt làm bộ đếm dừng. Trường hợp này, bộ định thời sẽ gây ngắt chương trình.

- Đếm với số đếm định sẵn ghi vào các thanh ghi TH, TL: trường hợp này cũng tương tự timer, bộ đếm sẽ đếm từ các giá trị high (-số đếm) và low (-số đếm) tới các giá trị 0000H.



Chú ý: Chương trình trên có thể dùng một Timer0 hay Timer1 nếu cho $t_r = t_0$ (độ kéo dài bằng thời gian trễ). Khi đó không phải dùng thêm chương trình ngắt do Timer.

Hình 10-9: Phát xung vuông ở chân P1.0 của VDK 8051

Ví dụ 10-14: Chương trình tạo xung vuông 50 Hz

```

ORG 00H           ; Reset
LJMP MAIN         ; nhảy tới MAIN
ORG 000BH         ; chương trình ngắt Timer 0
CPL P1.2          ; đảo bit P1.2
MOV TL0, #00H     ; nạp TL0
MOV TH0, #0DCH    ; nạp TH0
RETI              ; trở về từ ngắt
ORG 100H          ; bắt đầu chương trình chính
MAIN: MOV TMOD, #1H; Mod 1
      MOV TL0, #0DCH; nạp TL0
      MOV IE, #82H; cho phép ngắt Timer 0
      SETB TR0     ; đóng mạch Timer 0
      SJMP $       ; chờ ngắt
END               ; kết thúc chương trình dịch

```

- *Đếm với số đếm định sẵn ghi vào các bộ nhớ đệm TPH, TH1*: khi đếm luôn có lệnh so sánh TP với TC như ở timer. Khi TP = TC sẽ xác lập cờ so sánh CMFLAG hay cưỡng bức xác lập cờ TF để gây ngắt do bộ đếm.

Ví dụ 10-15 giới thiệu chương trình ngắt đo đếm xung vào bộ đếm 0 với chương trình con so sánh TP với TC.

Nếu dùng VDK làm bộ đếm một vật (hay người) đi qua thiết bị đếm mắc với VDK, ta phải có bộ phát hiện vật (dùng bức xạ hạt nhân, ánh sáng hồng ngoại, ánh sáng thường, sóng siêu âm v.v...), bộ tạo dạng xung để đưa một xung vào đếm khi có một vật (hay người) đi qua vùng đếm.

10.4. NGẮT DO TRAO ĐỔI TIN NỐI TIẾP

Ngắt trao đổi tin nối tiếp có mức ưu tiên thấp nhất trong các ngắt. Ngắt này là chung cho cả hai cờ ngắt:

- *Cờ ngắt nhận tin RI*: Xác lập khi bộ đệm nhận SBUF đầy tin (đã nhận tin xong) và yêu cầu VDK nhận tin vào thanh ghi chứa A để nhận tin cho lời tin tiếp theo. Cờ này có mức ưu tiên cao hơn cờ TI và phải được phép ngắt khi ghi bit REN = 1 ở thanh ghi SCON.

- *Cờ ngắt truyền tin TI*: Xác lập khi bộ đệm truyền rỗng (đã truyền tin xong) và yêu cầu VDK đưa tin từ thanh ghi A ra bộ đệm truyền SBUF để truyền tin cho lời tin tiếp theo. Cờ này có mức ưu tiên thấp hơn RI nhưng lại do VDK xác lập sau khi tự chủ truyền tin cho thiết bị ngoài mắc nối tiếp với VDK 8051/8052.

Khác với các ngắt khác, ngắt trao đổi tin này có những đặc điểm sau:

- *Các cờ ngắt RI (cho nhận tin) và TI (cho truyền tin) chỉ có chung một mức ngắt, một địa chỉ vectơ ngắt (23H)*: nên chương trình phục vụ ngắt phải có thêm phần kiểm tra xem ngắt nào trong RI (cho nhận tin) hay TI (cho truyền tin) để nhảy tới đoạn chương trình con phục vụ ngắt tương ứng.

- *Các cờ ngắt TI, RI sẽ không được VDK tự động xóa*: khi nhảy tới chương trình phục vụ ngắt mà phải xóa bằng lệnh CLR TI hay CLR RI trước khi nhảy tới chương trình phục vụ ngắt.

- Trao đổi tin nối tiếp luôn cần máy phát nhịp hay tốc độ baud (chuẩn là 9600 baud), có thể dùng Timer0, Timer1 ở chế độ 2 (tự nạp lại) hay Timer2 ở chế độ tự nạp lại (mặc định với bit CP//RL = 0 ở thanh ghi T2CON để giá trị thanh ghi 16 bit RCAP2 tự nạp vào TH2, TL2 khi đếm tràn).

Ngắt trao đổi tin nối tiếp này có thể được dùng kết hợp với các hành động khác của VDK như đưa tin vào-ra song song, định thời, đếm số xung vào và ngắt ngoài. Ngắt trao đổi tin nối tiếp này được dùng để hệ VDK trao đổi tin nối tiếp khi chưa biết thiết bị ngoài khác yêu cầu trao đổi tin khi nào. Nếu dùng chương trình trao đổi tin nối tiếp như ở chương 9, hệ VDK luôn phải chờ thiết bị ngoài đưa yêu cầu trao đổi tin hay xác lập cờ TI hay RI. Nếu dùng ngắt trao đổi tin nối tiếp này, hệ VDK không tốn thời gian chờ các cờ xác lập mà có thể thực hiện chương trình chính hay chương trình con phục vụ khác cho tới khi có cờ trao đổi tin nối tiếp xác lập mới nhảy tới chương trình phục vụ ngắt trao đổi tin nối tiếp.

; Ví dụ 10-15: Chương trình đếm xung của Timer0 trong 100sec dùng Timer2	
<pre> ORG 00H LJMP MAIN ORG 2BH ; vectơ ngắt Timer2 LJMP COUNTER2 ORG 20H ; vùng nhớ flag CMPFLAG DS 1 ORG 30H; vùng nhớ đệm T2_REG DS 1 TPH DS 1 TPL DS 1 TCH DS 1 TCL DS 1 ORG 100H ; vùng nhớ chương trình MAIN: MOV TMOD, #05H; Timer0 đếm, chế độ 1 SETB EA ; cho phép ngắt tổng quát SETB ET2 ; cho phép ngắt Timer2 SETB PT2 ; ưu tiên ngắt Timer2 MOV RCAP2L, #0F0H ; nạp dữ liệu MOV RCAP2H, #0D8H MOV TL2, #0F0H MOV TH2, #0D8H MOV T2_REG, #00H ; xóa nhớ đệm MOV TCL, #00H MOV TCH, #00H MOV TLO, #00H MOV TH0, #00H SETB TR0 ; bắt đầu đếm xung ngoài SETB TR2 ; bắt đầu đếm thời gian STAT0: CALL COMPARTPTC; so sánh TP = TC JBC CMPFLAG, STOP; nếu CMPTPTC = 1 JMP STAT0 ; trở về STAT0 STOP: CLR TR2 ; xóa TR2 CLR TR0 ; xóa TR2 MOV P0, TLO ; đọc TLO MOV P1, TH0 ; đọc TH0 MOV P2, TPH ; đọc TPH MOV P3, TPL ; đọc TPL SJMP \$; kết thúc chờ </pre>	<pre> MOV P0, TLO MOV P1, TH0 MOV P2, TPH MOV P3, TPL SJMP \$ COUNTER2: ; ch. trình con so sánh TP = TC PUSH PSW ; lưu giữ PSW PUSH ACC ; lưu giữ ACC INC T2_REG ; tăng nhớ đệm MOV A, T2_REG XRL A, #100 ; xét nhớ đệm = 100 ? JNZ CT2; nếu <100, tới CT2 MOV T2_REG, #00H; xóa T2_REG INC TCL ; tăng TCL MOV A, TCL XRL A, #00H ; xét TCL = 0? INC TCH ; tăng TCH CT2: CLR TF2 ; xóa TR2 CLR TR2 ; xóa TF2 POP ACC ; trả về ACC POP PSW ; trả về PSW RETI ; ; trở về chỗ bị ngắt CMPTPTC: ; chương trình so sánh TP = TC? PUSH ACC PUSH PSW CLR C ; xóa cờ C MOV A, TPH ; gửi TPH vào A CJNE A, TCH, NOT_EQH; so với TCH CLR C ; xóa cờ C MOV A, TPL ; gửi TPL vào A CJNE A, TCL, NOT_EQL; so với TCL JMP TOVER ; nhảy tới TOVER NOT_EQH: JNC DONE; nếu C = 0, tới DONE TOVER: SETB CMPFLAG; xác lập cờ = JMP DONE ; về so sánh xong NOT_EQL: JC TOVER; nếu C = 1, tới DONE DONE: POP PSW; trả về PSW POP ACC ; trả về ACC RET ; trở về chương trình chính END ; kết thúc chương trình dịch </pre>

Ví dụ 10-16 là chương trình đọc dữ liệu từ cổng P1, đưa ra hiển thị trên cổng P2. Khi có cờ ngắt nối tiếp, nếu TI = 1 truyền byte của tin đọc vào từ P1 ra chân TxĐ của cổng nối tiếp. Nếu cờ nhận RI = 1, VDK nhận tin từ chân RxĐ của cổng nối tiếp vào thanh ghi A rồi đưa ra cổng P2.

Ví dụ 10-17 giới thiệu chương trình gồm một hành động chính và 3 ngắt:

- *Hành động chính*: khởi động các Timer, cổng nối tiếp, đọc tin từ cổng P1 rồi đưa ra cổng P2.
- *Phát tốc độ baud*: dùng timer1 phát với tốc độ 4800 baud.
- *Ngắt timer0*: phát xung vuông tại chân P1.0 của cổng P1 với cờ ngắt TF0.

Ví dụ 10-16: Chương trình ngắt truyền, nhận tin nối tiếp

<pre> ORG 00H LJMP MAIN ORG 0023H LJMP SERIAL ; tới chương trình ngắt ORG 100H MAIN: MOV P1, #0FFH; P1 là cổng vào MOV TMOD, #20H ; Timer 1, chế độ 2 MOV TH1, #0FDH ; tốc độ 9600 MOV IE, #90H ; cho phép các ngắt nối tiếp SETB TR1 ; bật Timer 1 BACK: MOV A, P1 ; đọc cổng P1 MOV P2, A ; gửi ra P2 </pre>	<pre> SETB TI ; truyền cưỡng bức SJMP BACK ; trở về BACK SERIAL: JB TI, TRAN ; truyền nếu TI = 1 JB RI, \$ + 3 ; nhận nếu RI = 1 RECEIV: MOV A, SBUF; nhận tin nối tiếp CLR RI ; xoá cờ nhận JMP KT ; trở về từ ngắt TRAN: CLR TI ; xoá cờ truyền MOV SBUF, A ; truyền tin nối tiếp KT: RETI ; trở về từ ngắt END ; kết thúc dịch </pre>
---	--

Ví dụ 10-17: Ngắt truyền nhận nối tiếp, tạo xung vuông

<pre> ORG 00H LJMP MAIN ORG 00BH CPL P0.1 RETI ORG 0023H LJMP SERIAL ; nhảy tới ngắt nối tiếp ORG 002BH LJMP T2ISR ; nhảy tới ngắt T2ISR ORG 100H MAIN: MOV P1, #00H ; P1 là cổng vào MOV TMOD, #22H ; Timer 0, 1 ở chế độ 2 MOV TH1, #0F6H ; tốc độ baud 4800 MOV TL1, #0F6H MOV SCON, #50H ; 8 bit, 1stop, REN = 1 MOV TH0, #HIGH(-92); tần số 5kHz MOV TH0, #LOW(-92); tần số 5kHz MOV TH2, #00H ; số đếm ban đầu MOV TL2, #00H ; số đếm ban đầu MOV RCAP2H, #00H ; số đếm ban đầu MOV RCAP2L, #00H ; số đếm ban đầu MOV T2CON, #03H ; timer2 đếm xung ; vào chân T2 (chân P1.0, số 1) MOV IE, #0BAH ; cho phép các ngắt SETB TR0 ; bật Timer 0 </pre>	<pre> SETB TR1 ; bật Timer 1 SETB TR2 ; bật Timer2 BACK: MOV A, P1 ; đọc cổng P1 MOV P2, A ; gửi ra P2 SJMP BACK ; trở về BACK SERIAL: JB TI, TRAN ; truyền nếu TI = 1 JB RI, \$ + 3 ; chờ cờ nhận RI = 1 RECEIV: CLR RI ; xoá cờ nhận MOV A, SBUF ; nhận tin nối tiếp RETI TRAN: CLR TI ; xoá cờ truyền SETB TF2 ; xác lập TF2 RETI ; trở về từ ngắt T2ISR: CLR TR0 ; ngừng phát xung CLR TR1 ; ngừng phát tốc độ CLR TR2 ; ngừng đếm xung vào T0 MOV IE, #00H ; cấm các ngắt MOV A, RCAP2H ; đọc số đếm byte cao MOV SBUF, A ; truyền byte cao CLR TI ; xoá cờ truyền MOV A, RCAP2L ; đọc số đếm byte thấp MOV SBUF, A ; truyền byte thấp CLR TI ; xoá cờ truyền RETI ; trở về từ ngắt END ; kết thúc chương trình </pre>
---	--

- Ngắt đếm Counter2: dùng timer2 làm bộ đếm.

- Ngắt nối tiếp: + ngắt cưỡng bức bởi lệnh SETB TI để truyền tin ở A đọc từ cổng vào P1 cho máy vi tính PC. + nhận tin từ PC nếu có cờ RI xác lập lên 1.

Chương trình trên luôn phát tốc độ baud 4800 baud và xung vuông ở chân P1.0 cho tới khi bộ đếm Timer2 dừng đếm.

Khi có yêu cầu trao đổi tin từ máy vi tính PC thì:

- Nếu PC có yêu cầu VDK nhận tin, RI = 1, VDK nhận tin vào thanh ghi A.

; Ví dụ 10-18: Chương trình trao đổi tin nối tiếp với máy tính PC

ORG 20h ; kê khai cờ KEYFLAG DS 1 ; cờ nhấn phím SENDFLAG DS 1 ; cờ gửi dữ liệu RECEIFLAG DS 1 ; cờ nhận dữ liệu ORG 30H ; kê khai ô nhớ đếm KB_DATA DS 1 ; đếm của phím nhấn RX_DATA DS 1 ; đếm nhận TX_DATA DS 1 ; đếm gửi ORG 00H ; RESET LJMP MAIN ; chương trình ngắt nối tiếp ORG 0023H LJMP SERIAL ; nhảy tới ngắt nối tiếp ; chương trình chính ORG 100H MAIN: SETB SCON.6 ; SM1 = 1 CLR SCON.7 ; SM0 = 0 SETB T2CON.5 ; RCLK = 1 SETB T2CON.4 ; TCLK = 1 CLR T2CON.1 ; CP/RL = 0, tự nạp MOV RCAP2H, #0FFH ; baud = 9600 MOV RCAP2L, #0D9H SETB T2CON.2 ; TR2 = 1 MOV IE, #91H ; cho phép ngắt nối tiếp SE_MENU: JNB KEYFLAG, SE_MENU CLR KEYFLAG ; chờ ấn phím, xóa cờ ANL KB_DATA, #0FH ; lấy 4 bit thấp MOV A, KB_DATA ; gửi vào A XRL A, #'A' ; chữ 'A' ; xét chữ 'A'? JZ SENDING ; gửi ký tự MOV A, KB_DATA ; gửi vào A XRL A, #'B' ; chữ 'B' ; xét chữ 'A'? JZ RECEIVER ; nhận ký tự MOV A, KB_DATA ; gửi vào A XRL A, #'F' ; chữ 'F' ; xét chữ 'A'? JZ MAIN ; trở về đầu chương trình JMP SE_MENU ; về thực đơn RECEIVER: JMP RECEIVING SENDING: MOV A, #' ' ; gửi dấu ' ' CALL SEND ; gọi gửi MOV A, #0DH ; mã của 'LF' CALL SEND ; gọi gửi SEND1: JNB KEYFLAG, SEND1 ; chờ cờ CLR KEYFLAG ; xóa cờ bàn phím ANL KB_DATA, #0FH ; chọn 4 bit thấp MOV A, KB_DATA ; gửi vào A	XRL A, #0AH ; mã của 'CR' ? JZ SEND1 RECEIVING: ; nhận vào các ô nhớ @DPTR SETB SCON.4 ; REN = 1 RECEI1: JNB RECEIFLAG, RECEI1 ; chờ CLR RECEIFLAG ; xóa cờ nhận MOV A, RX_DATA ; gửi dữ liệu nhận MOVX @DPTR, A ; ghi vào nhớ ngoài CALL SEND ; gọi gửi INC DPTR ; tăng địa chỉ JNB KEYFLAG, RECEI1 ; trở về RECEI1 CLR KEYFLAG ; xóa cờ bàn phím JMP SE_MENU ; về thực đơn SERIAL: ; chương trình ngắt nối tiếp PUSH PSW PUSH ACC MOV A, SCON ; đọc trạng thái ANL A, #01 ; xét bit RI JNZ RITE ; nếu RI = 1, tới RITE MOV A, SCON ; đọc trạng thái ANL A, #02 ; xét bit TI JNZ TITE ; nếu TI = 1, tới TITE SERIAL1: POP ACC POP PSW RETI RITE: SETB RECEIFLAG ; xác lập cờ nhận MOV RX_DATA, SBUF ; nhận dữ liệu CLR RI ; xóa cờ nhận CLR SCON.2 ; xóa RB8 JMP SERIAL1 ; về SERIAL1 TITE: SETB SENDFLAG ; xác lập cờ Gửi CLR TI ; xóa TI CLR SCON.3 ; xóa TB8 JMP SERIAL1 ; về SERIAL1 SEND: ; chương trình con gửi dữ liệu trong A PUSH PSW PUSH ACC MOV SBUF, A ; A gửi ra SBUF SEN0: JNB SENDFLAG, SEN0 ; chờ cờ gửi CLR SENDFLAG ; xóa cờ POP ACC POP PSW RET END
--	--

- Nếu PC có yêu cầu VDK truyền tin, chương trình xác lập TF2, gây ngắt cưỡng bức cho bộ đếm Timer2, làm ngừng đếm, ngừng phát tốc độ baud, ngừng phát xung vuông, đọc dữ liệu của bộ đếm từ các thanh ghi RCAP2H và RCAP2L (chế độ thu nhận với bit CP = 1) rồi truyền cho PC.

Nếu chương trình trên có thêm lệnh chờ ngắt ngoài (INT0, INT1) để khởi phát chương trình, ta có chương trình với đầy đủ 4 ngắt của VĐK.

Ví dụ 10-18 giới thiệu chương trình trao đổi tin nối tiếp giữa VĐK và máy vi tính PC.

Chương trình này có thể truyền tin và nhận tin từ máy vi tính PC (xem chương 9). Chương trình gồm hai phần là khởi động trao đổi tin nối tiếp và ngắt trao đổi tin nối tiếp.

- *Khởi động trao đổi (truyền, nhận) tin nối tiếp*: gồm các hành động cho phép ngắt nối tiếp, đặt tốc độ truyền, nhận (dùng Timer2 dùng chế độ tự nạp lại), thực đơn trao đổi tin nối tiếp với các chữ khi nhấn phím từ bàn phím:

+ *Chữ A*: truyền tin nối tiếp.

+ *Chữ B*: nhận tin nối tiếp.

+ *Chữ F*: thoát trở về chương trình chính.

10.5. NGẮT DO TÍN HIỆU NGOÀI

VĐK 8051 có hai ngắt ngoài INT0 (chân P3.2- số 12), INT1(chân P3.2- số 12) và VĐK 8052 có thêm ngắt ngoài T2EX (chân P1.1- số 2). Khi có tín hiệu ngoài vào một trong ba lối vào ngắt ngoài trn và chương trình đã cho phép ngắt (ghi vào thanh ghi IE) thì các cờ ngắt IE0, IE1, EXF2 sẽ xác lập, chương trình sẽ nhảy tới vectơ ngắt ngoài tương ứng (03H, 13h và 2BH) để thực hiện chương trình ngắt ngoài ứng với các vectơ ngắt. Ta cần lưu ý rằng Timer2 có ngắt ngoài T2EXF2 có chung một vectơ ngắt với TF2 là 2BH.

10.5.1. Đặc điểm của ngắt ngoài của VĐK 8051

1. Nguồn gây ngắt và các loại ngắt

Do tín hiệu ngoài đưa vào các chân P3.2 (/INT0) hay P3.2 (/INT1). Tín hiệu này có thể là mức âm (logic 0) hay sườn xung từ mức dương (logic 1) xuống mức âm (logic 0).

- *Với tín hiệu vào theo mức âm*: Theo mặc định, các bit IT0, IT1 của thanh ghi TCON có giá trị 0 và chân các lối vào của tín hiệu ngắt ngoài (P3.2, P3.3) ở mức cao (logic 1). Khi có tín hiệu ngắt ngoài ở mức logic 0, nếu đã ghi cho phép ngắt, VĐK sẽ trở về địa chỉ của vectơ ngắt tương ứng (03h hay 13h) để thực hiện chương trình phục vụ ngắt ngoài INT0 hay INT1. Sau đó, tín hiệu vào phải trở về mức dương, nếu không, quá trình ngắt chương trình vẫn xảy ra cho tới khi tín hiệu vào trở về mức dương. Người ta lợi dụng hiện tượng này để duy trì báo động (ví dụ 10-21) khi lối vào có tín hiệu mức âm.

- *Với tín hiệu vào theo sườn xung âm* (từ mức dương xuống mức âm): trong chương trình (ví dụ 10-19), ta phải dùng lệnh SETB IT0 hay SETB IT1 để có IT0 hay IT1 = 1 ở thanh ghi TCON (hình 10-10). Khác với trường hợp trên, nếu tín hiệu ngoài vào chưa trở về mức dương, sẽ không có quá trình ngắt xảy ra.

Chúng ta có thể kiểm nghiệm lại các nhận xét trên bằng ví dụ 10-19 với các giá trị IT (hình 10-9) tương ứng.

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Hình 10-10. Các bit của thanh ghi TCON

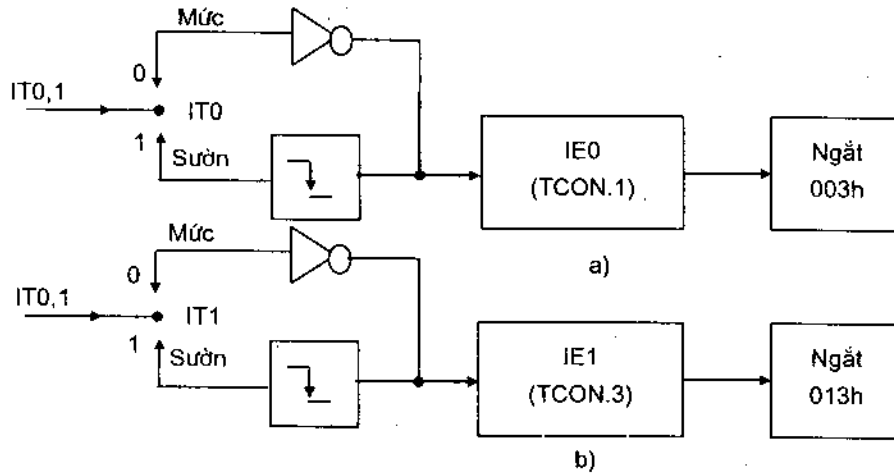
2. Cấm và cho phép ngắt ngoài

Theo mặc định, các ngắt ngoài cũng như các ngắt khác bị cấm ngắt vì mọi bit của thanh ghi cho phép ngắt IE có giá trị 0 (hình 10-10).

Việc ngắt do các tín hiệu ngoài /INT0 hay /INT1 chỉ xảy ra khi:

- Bit cho phép ngắt tổng quát hay toàn bộ EA = 1 (ghi bởi lệnh SETB EA) của thanh ghi cho phép ngắt IE.

- Bit cho phép ngắt ngoài EX0 = 1 (ghi bởi lệnh SETB EX0), sẽ cho phép ngắt ngoài/INT0 ngắt chương trình.



Hình 10-11. Kích hoạt INT0 (a) và INT1 (b)

- Bit cho phép ngắt ngoài $EX1 = 1$ (ghi bởi lệnh SETB EX1), sẽ cho phép ngắt ngoài /INT1 ngắt chương trình.

3. Cờ ngắt

Thanh ghi TCON (hình 10-10) có hai bit IE0 và IE1 chỉ cờ ngắt ngoài INT0, INT1 của VĐK 8051. Nếu có tín hiệu ngoài gây ngắt vào các chân P3.2 hay P3.3 mà chương trình chưa có lệnh cho phép ngắt ($EA = 1$ hay $EX0$ hay $EX1 = 1$), cờ ngắt (bit IE0 hay IE1) của thanh ghi TCON đã xác lập. Khi đã ghi cho phép ngắt ($EA = 1$, $EX0 = 1$ hay $EX1 = 1$) chương trình sẽ trở về vectơ ngắt và tự động xóa cờ ngắt này. Khi cần ngắt ngoài cưỡng bức (không có tín hiệu ngoài gây ngắt, ta có thể ghi các lệnh cho phép ngắt ($EA = 1$, $EX0$ hay $EX1 = 1$) và SETB IE0 hay IE1. Khi đó vẫn có ngắt ngoài xảy ra và chương trình vẫn thực hiện chương trình phục vụ ngắt ngoài.

4. Mức ưu tiên

Theo mặc định hay quy ước của phần cứng, ngắt ngoài /INT0 có mức ưu tiên cao nhất sau tín hiệu RESET, còn /INT1 phải sau ngắt của Timer0. Giữa hai ngắt ngoài, thì theo mặc định, /INT1 có mức ưu tiên sau /INT0 nếu cả hai tín hiệu vào đồng thời các lối vào P3.2, P3.3. Do đó, trong hệ VĐK, người ta dùng ngắt ngoài /INT0 cho bàn phím và khi ấn phím của bàn phím, sẽ tạo ngắt ngoài /INT0.

Chúng ta có thể tạo cho tín hiệu /INT1 gây ngắt trước tín hiệu /INT0 nếu trong chương trình có thêm lệnh SETB PX1 tức ghi mức ưu tiên $PX1 = 1$ cho thanh ghi ưu tiên ngắt IP. Khi chạy chương trình của ví dụ 10-19, ta dùng lệnh trên để /INT1 được ngắt chương trình chính trước ngắt do tín hiệu /INT0 (bằng bỏ dấu chấm phẩy (;) để dùng lệnh SETB PX1).

10.5.2. Chương trình cho ngắt ngoài của VĐK 8051

1. Chương trình chính

Chương trình chính có các lệnh khởi phát ngắt là:

- Các kê khai vectơ ngắt tương ứng và dữ liệu.
- Các lệnh cho phép ngắt: ngắt tổng quát (SETB EA) và lệnh cho phép ngắt của từng ngắt ngoài (SETB EX0, SETB EX1).
- Các lệnh xác định tín hiệu ngắt theo sườn xung: SETB IT0 cho INT0 hay SETB IT1 cho INT1. Nếu ngắt theo mức logic 0, theo mặc định, không cần ghi thêm một lệnh nào.
- Lệnh ưu tiên ngắt: cho tín hiệu có mức ưu tiên thấp hơn theo mặc định nhưng lại cần ưu tiên trước như SETB PX1 (cho $IP.1 = 1$) để ngắt INT1 có mức ưu tiên hơn ngắt INT0.
- Lệnh chờ các tín hiệu vào ngắt: *sjmp \$*.

2. Chương trình phục vụ ngắt

Chương trình này ghi ở vectơ ngắt (nếu độ dài nhỏ hơn 8 ô nhớ) hay ghi ở chương trình riêng, nằm sau chương trình chính. Ở vectơ ngắt, phải có lệnh nhảy dài tới nhãn của chương trình phục vụ ngắt. Cả hai trường hợp trên, cuối chương trình phục vụ ngắt, phải có lệnh RETI (trở về từ ngắt).

Ví dụ 10-19 minh họa cấu trúc chương trình cho hai ngắt ngoài. Các lệnh sau dấu chấm phẩy (;) chưa dùng. Người sử dụng có thể dùng các lệnh này bằng cách xóa dấu chấm phẩy. Khi cho biên dịch và chạy thử từng bước trong chương trình mô phỏng Pinnacle, ta phải kích chuột vào các lối vào P3.2, P3.3 của cổng vào trên các cổng mô phỏng (vào VIEW\PORT).

; Ví dụ 10-19: Các ngắt ngoài	
ORG 00H	MOV P1, #00H
LJMP MAIN	SETB EA ; cho phép ngắt toàn bộ
ORG 03H ; ngắt ngoài 0	SETB EX0 ; cho phép ngắt INTO
EX0ISR: SETB P1.7	SETB EX0 ; cho phép ngắt INT
RETI	; SETB IT0 ; sườn xung xuống của P3.2
ORG 13H ; ngắt ngoài 1	; SETB IT1 ; sườn xung xuống của P3.3
EX1ISR: CLR P1.7	; SETB PX1 ; ưu tiên ngắt INT1 hơn INTO
RETI	; MOV P3, #00H; tạo hai ngắt đồng thời
ORG 30H	SJMP \$; chờ ngắt
MAIN: ; mov tcon, #00h	END ; kết thúc dịch

Theo dõi trình tự chương trình trong thực đơn VIEW/CODE DISASSEMBLER và các thanh ghi REGISTERS, TCON và IP ta có thể thấy được cờ ngắt ngoài (IE0, IE1) ở thanh ghi TCON xác lập và chương trình phục vụ các ngắt ngoài (địa chỉ nhớ 03H và 13H).

Cụ thể là: - Chưa dùng lệnh sau dấu chấm phẩy để khảo sát ngắt ngoài với mức logic âm.

- Dùng các lệnh SETB IT0, SETB IT1 bằng cách bỏ dấu chấm phẩy (;) để khảo sát ngắt ngoài với sườn xung âm (P3.2 hay P3.3 từ logic 1 sang logic 0).

- Dùng lệnh SETB PX1 và MOV P3, #00h bằng cách bỏ dấu chấm phẩy (;) để khảo sát INT1 ưu tiên hơn INTO vì theo mặc định INTO ưu tiên hơn.

3. Mở rộng ngắt ngoài

Đặc điểm chính của ngắt ngoài là khi nhận biết tín hiệu vào có mức âm hay sườn âm vào các chân P3.2, P3.3, VDK nhảy tới các vectơ ngắt ngoài /INT0, /INT1 định sẵn. Vì chỉ có hai ngắt ngoài INT0, INT1, nhưng trong thực tế, nhiều khi đòi hỏi dùng nhiều ngắt ngoài, ví dụ, điều khiển một chiếc xe cần 5 phím cho các hành động khởi động, tiến, lùi, rẽ trái, rẽ phải (xem chương 8). Từ đó, ta cần mở rộng thêm nhiều ngắt ngoài bằng cách:

- Ngắt ngoài INT0 hay INT1 làm ngắt khởi động, vào thực đơn dùng nhiều lối vào của một cổng song song nào đó. Trong ví dụ 10-20 dùng cổng P0.

- Chọn các bit của một trong các cổng vào làm bit nhận các tín hiệu ngoài vào.

- Chương trình đọc và nhận biết tín hiệu vào để nhảy tới nhãn của các đoạn chương trình dành cho mở rộng ngắt nằm trong chương trình phục vụ ngắt ngoài.

- Trước khi nhảy tới lệnh đầu tiên của đoạn chương trình phục vụ này, phải ghi nhớ nội dung các thanh ghi PSW, Acc (bằng các lệnh PUSH) vào đoạn nhớ ngăn xếp.

- Trước khi kết thúc phải tiến hành hồi phục trạng thái các thanh ghi trên (bằng các lệnh POP) với trật tự ghi vào trước, đọc ra sau (LIFO-Last In First Out).

- Có thể chọn một bit (trong vùng địa chỉ từ 20H) làm cờ cho chương trình mở rộng ngắt ngoài và phải kiểm tra cờ này để nhảy tới nhãn (lệnh JC nhãn) của chương trình phục vụ phím ấn.

Ví dụ 10-20 là chương trình mở rộng ngắt ngoài. Khi có ngắt ngoài INT0, nhảy tới chương trình phục vụ ngắt ngoài mở rộng với các bit P0.0, P0.1 riêng rẽ và tổ hợp P0.0+P0.1, P0.2 cho bốn ngắt ngoài mở rộng với 4 chương trình phục vụ ngat0, ngat1, ngat2 và ngat3 tương ứng. Cổng P0 có thể cho tới 2⁸ giá trị của P0I, tức 2⁸ ngắt ngoài mở rộng.

Với ngắt ngoài INT1, ta cũng có thể mở rộng cho nhiều ngắt ngoài với nhiều bit của một cổng khác.

10.5.3. Ngắt ngoài của 8052

Timer2 của VĐK 8052, ngoài hai ngắt ngoài INT0, INT1 như của 8051, còn có ngắt ngoài với lối vào T2EX (chân P1.1- số 2). Thanh ghi T2CON (hình 10-10) có các bit cờ ngắt ngoài EXF2 nếu được phép bởi bit E XEN2 = 1 ở cả hai chế độ (tự nạp lại và thu nhận (hình 10-11a, b)

Ngoài ra, còn các bit: - *ET2* : cho phép ngắt của thanh ghi cho phép ngắt IE.

- *PT2* : ưu tiên ngắt cho INT2.

Hoạt động ngắt chương trình của ngắt ngoài do Timer2 gây ra cũng tương tự của VĐK 8051, chỉ có điều khác là vectơ ngắt có địa chỉ 2BH.

Ví dụ 10-21 giới thiệu ngắt ngoài 2 của Timer2 của VĐK 8052.

Chú ý: - Theo sơ đồ ở hình 10-13, ngắt ngoài EX2 có đặc điểm sau:

- Chỉ xảy ra khi đã ghi cho phép vào bit EXEN2.

- Xác lập cờ EX2 và TF2.

- Bắt giữ hay đưa ra các thanh ghi RCAP2H, RCAP2L các giá trị của các thanh ghi TH2, TL2.

Nhưng chương trình mô phỏng Pinnacle không thực hiện các điều trên hay nói cách khác, không chạy ngắt ngoài INT2 cho Timer2.

; Ví dụ 10-20: Ngắt ngoài mở rộng	
<pre> ORG 00H LJMP MAIN ORG 03H LJMP NGAT00 ORG 100H MAIN: MOV SP, #70H SETB EA SETB EX0 MOV P0, #00H SJMP \$ NGAT00: LAP: MOV A, P0 JZ NGAT0 CJNE A, #01H, NGAT JMP NGAT1 JMP LAP NGAT: CJNE A, #02, NGAT0 JMP NGAT2 JMP LAP NGAT0: CJNE A, #03, NGAT1 JMP NGAT3 </pre>	<pre> NGAT1: JMP LAP NGAT0: MOV P2, #00H JMP LAP NGAT1: PUSH PSW PUSH ACC MOV P2, #01H POP ACC POP PSW JMP LAP NGAT2: PUSH PSW PUSH ACC MOV P2, #02H POP ACC POP PSW LJMP LAP NGAT3: PUSH PSW PUSH ACC MOV P2, #03H POP ACC POP PSW RETI END </pre>

D7	D6	D5	D4	D3	D2	D1	D0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/I/T2	CP//RL2

Hình 8-12. Các bit của thanh ghi T2CON

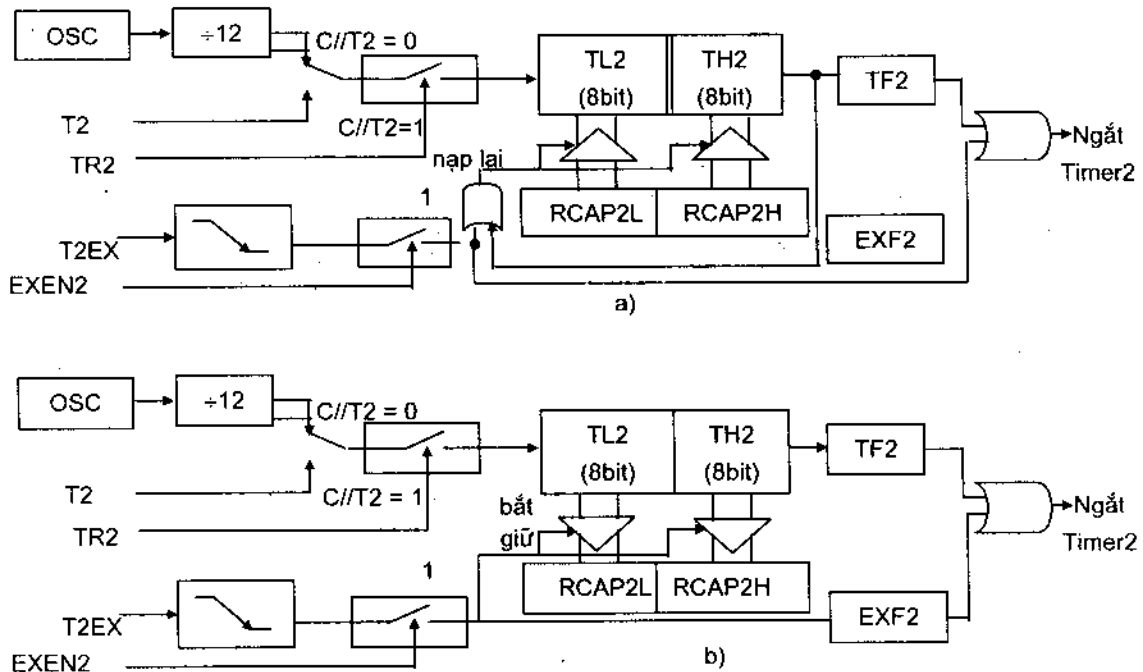
10.5.4. Ứng dụng các ngắt ngoài

Thông thường, trong một hệ VĐK thường dùng các ngắt ngoài với mục đích sau:

- *Ngắt ngoài /INT0*: dành cho chương trình ngắt bàn phím, khi có một phím của bàn phím được nhấn, chương trình chính nhảy tới chương trình ngắt bàn phím để đọc mã phím nhấn và chuyển tới chương trình con phục vụ cho phím đó.

Muốn tạo ngắt bàn phím, có thể:

- Nối các lối ra của các phím nhấn với lối vào của cổng logic Hoặc và nối lối ra của cổng này với /INT0 để tạo một ngắt cứng.



Hình 10-13. Các chế độ tự nạp (a) và bắt giữ (b) của Timer 2

; Ví dụ 10-21: Chương trình cho ngắt ngoài INT2	
<pre> ORG 00H LJMP MAIN ORG 02BH ; ngắt ngoài 2 LJMP EX2ISR ORG 30H MAIN: MOV T2CON, #01H; chế độ bắt giữ MOV T2MOD, #02H ; cho phép đưa ra SETB EA ; cho phép ngắt toàn bộ SETB ET2 ; cho phép ngắt timer2 SETB PT2 ; ưu tiên ngắt pt2 SETB EXEN2 ; cho phép ngắt ngoài </pre>	<pre> MOV TH2, #0FFH ; giá trị th2 đưa ra MOV TL2, #0FAH ; giá trị tl2 đưa ra MOV RCAP2L, #00H ; giá trị bắt giữ th2 MOV RCAP2H, #00H ; giá trị bắt giữ tl2 SJMP \$; chờ ngắt EX2ISR: CLR TF2 ; chương trình ngắt MOV P2, #55H RETI ; trở về từ ngắt END ; kết thúc dịch </pre>

- Dùng lệnh SETB INT0 để tạo ngắt mềm hay ngắt cường bức thay cho một phím được nhấn (giải pháp mềm). Chương trình chính kiểm tra cờ bàn phím này để nhảy tới chương trình ngắt bàn phím.

- Ngắt ngoài thứ hai INT1 trong một hệ đo có VDK: thường được dùng để ngắt chương trình và nhảy tới chương trình phục vụ việc tích lũy tin cho hệ đo. Khi có yêu cầu trao đổi tin với hệ đo, có tín hiệu yêu cầu INT1 vào chân /INT1 và hệ nhảy tới chương trình phục vụ ngắt để thu thập tin từ cảm biến qua một khối biến đổi tương tự-số ADC rồi ghi vào bộ nhớ ngoài hoặc gửi tin vào máy vi tính.

- Ngắt ngoài thứ hai INT1 trong một hệ điều khiển có VDK: thường dùng để điều khiển chương trình, điều khiển thiết bị ngoài.

1. Điều khiển chương trình

a) Điều khiển chương trình chính

Người ta có thể điều khiển chương trình của VDK bắt đầu chạy (thực hiện) bằng chèn lệnh JBP3.2 (hay P3.3), \$ vào đầu một chương trình chính nào đó. Khi chưa có tín hiệu ngoài vào từ một

phím chuyển mạch, khi đó P3.2 (hay P3.3 = 1 theo mặc định, chương trình chờ tại chỗ. Khi nhấn phím, cho một tín hiệu âm (logic 0) vào một trong các chân ngắt ngoài (P3.2, P3.3, T2EX), lệnh JB P3.2 (hay JB P3.3 hoặc JB P1.1), VDK tiếp tục thực hiện các lệnh đứng sau lệnh JB P3.2 (hay JB P3.3 hoặc JB P1.1).

b) Điều khiển chương trình con phục vụ ngắt

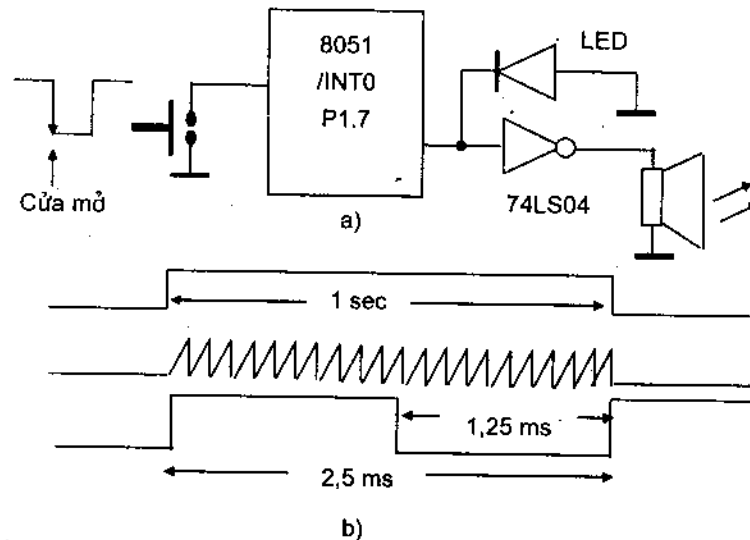
Chương trình con phục vụ ngắt ngoài được thực hiện khi có ngắt ngoài tác động lên các chân ngắt ngoài (P3.2, P3.3, P1.1). Khi đó, trong chương trình chính phải có lệnh cho phép ngắt toàn bộ (SETB EA) và ngắt ngoài tương ứng (SETB EX0, EX1, EXEN2). Chương trình phục vụ ngắt ngoài có thể có hành động bất kỳ mong muốn tùy lập trình viên. Ví dụ, chương trình chỉnh phút (ví dụ 10-11) hay chỉnh giờ (ví dụ 10-12) trong đồng hồ số hay đơn giản là đưa dữ liệu #55h ra cổng P trong ví dụ 10-21.

2. Điều khiển báo động

Nếu có một thiết bị báo động gắn ở cửa như hình 10-14. Khi cửa mở, chương trình báo động ở ví dụ 10-22 sẽ làm loa kêu với tần số 440 Hz trong một giây. Có thể thay loa bằng một đèn LED đơn. Đèn sẽ nhấp nháy trong một giây khi cửa mở.

Chương trình báo động bằng cho loa như ví dụ 10-22 với :

- Ngắt ngoài lấy tín hiệu vào/INT0 từ bộ cảm biến quang để phát hiện vật che chắn với chương trình ngắt ngoài EX0ISR.
- Loa kêu với tần số âm thanh 440 Hz trong 1 sec với ngắt thời gian T0ISR



Hình 10-14. Sơ đồ (a) của thiết bị báo động và giản đồ thời gian của tín hiệu ra loa

3. Điều khiển lò nung

Sơ đồ mắc như hình 10-15 của sự khống chế nhiệt độ của lò nung.

Một lò nung được điều khiển bởi chân P1.7 của VDK 8051/52 với các lối vào /INT0, /INT1 mắc với hai nhiệt kế chỉ nóng (/HOT) và lạnh (/COLD) như hình 10-15a. Lò nung được điều chỉnh ở nhiệt độ < 25°C và > 19°C như giản đồ xung ở hình 10-13b.

Khi nhiệt độ > 21°C lối vào /HOT có một xung âm vào /INT0, gây ngắt chương trình, khiến lối ra P1.7 không có xung, làm nhả rơle, ngắt mạch điện nuôi lò nung, nhiệt độ lò sẽ giảm đi.

Khi nhiệt độ < 19°C lối vào /COLD có một xung âm vào /INT1, gây ngắt chương trình, khiến lối ra P1.7 có tín hiệu = 1, làm đóng rơle, đóng mạch điện nuôi lò nung, nhiệt độ lò sẽ tăng lên.

Như vậy, thiết bị và chương trình ở ví dụ 10-23 đã tự động khống chế lò nung với hai ngắt ngoài vào từ các bộ tạo dạng tín hiệu của các nhiệt kế điện tử số.

<i>; Ví dụ 10-22: Chương trình báo động bằng loa</i> <pre> ORG 00H LJMP MAIN ORG 03H LJMP EX0ISR ORG 00BH LJMP T0ISR ORG 01BH LJMP T1ISR ORG 30H MAIN: SETB IT0 MOV TMOD, #11H ; timer mode 1 MOV IE, #81H ; cho phép ngắt ngoài 0 SJMP \$ EX0ISR: MOV R7, #20 SETB TF0 ; ngắt cường bức TF0 SETB TF1 ; ngắt cường bức TF1 SETB ET0 ; cho phép ngắt timer0 </pre>	<pre> SETB ET1 ; cho phép ngắt timer1 T0ISR: CLR TR0 DJNZ R7, SKIP CLR ET0 CLR ET1 CLR TR1 LJMP EXIT SKIP: MOV TH0, #HIGH(-50000) MOV TL0, #LOW (-50000) SETB TR0 EXIT: RETI ; trở về từ ngắt T1ISR: CLR TR1 MOV TH1, #HIGH (-1250) MOV TL1, #LOW (-1250) CPL P1.7 ; phát âm SETB TR1 RETI ; trở về từ ngắt END </pre>
<i>; Ví dụ 10-23: Chương trình điều khiển lò nung</i> <pre> ORG 00H LJMP MAIN ORG 03H ; ngắt /INT0 CLR P1.7 ; xoá P1.7 RETI ; trở về từ ngắt ORG 13H SETB P1.7 ; xác lập P1.7 RETI ; trở về từ ngắt </pre>	<pre> ORG 30H ; chương trình chính MAIN: MOV IE, #85H ; cho phép ngắt ngoài 0, 1 SETB IT0 ; ngắt sườn xung /INT0 SETB IT1 ; ngắt sườn xung /INT1 SETB P1.7 ; đóng rơ le của nguồn nuôi JB P3.2, SKIP ; chờ P3.2 = 1 CLR P1.7 SKIP: SJMP \$ END </pre>

4. Đo thời gian dùng ngắt ngoài

Ở chương 8, chúng ta đã xét vấn đề đo một khoảng thời gian bằng Timer nhưng không dùng ngắt mà dùng tín hiệu ngoài cần đo độ rộng xung đưa vào các chân INT0 (P3.2), INT1 (P3.3).

Ở mục này, chúng ta xét việc đo thời gian dùng ngắt (ngắt do Timer, ngắt ngoài).

Đo độ rộng xung là so sánh độ kéo dài xung cần đo với độ kéo dài của N độ kéo dài xung hẹp hơn được dùng làm đơn vị thời gian. Nếu gọi độ kéo dài của xung đơn vị đo là t_0 , thì độ kéo dài xung đo là :

$$T_x = N \times t_0$$

Tùy độ lớn của t_x , ta chọn đơn vị đo t_0 . Nếu t_x càng lớn, t_0 càng nhỏ để phép đo càng chính xác, nhưng số đếm N càng lớn. Với timer ở chế độ 1, số đếm tối đa là 65535 (của 16 bit).

Với thời gian nhỏ cỡ microsec, đơn vị thời gian t_0 là microsec của máy phát xung nhịp vào timer và số đếm tối đa là 65535.

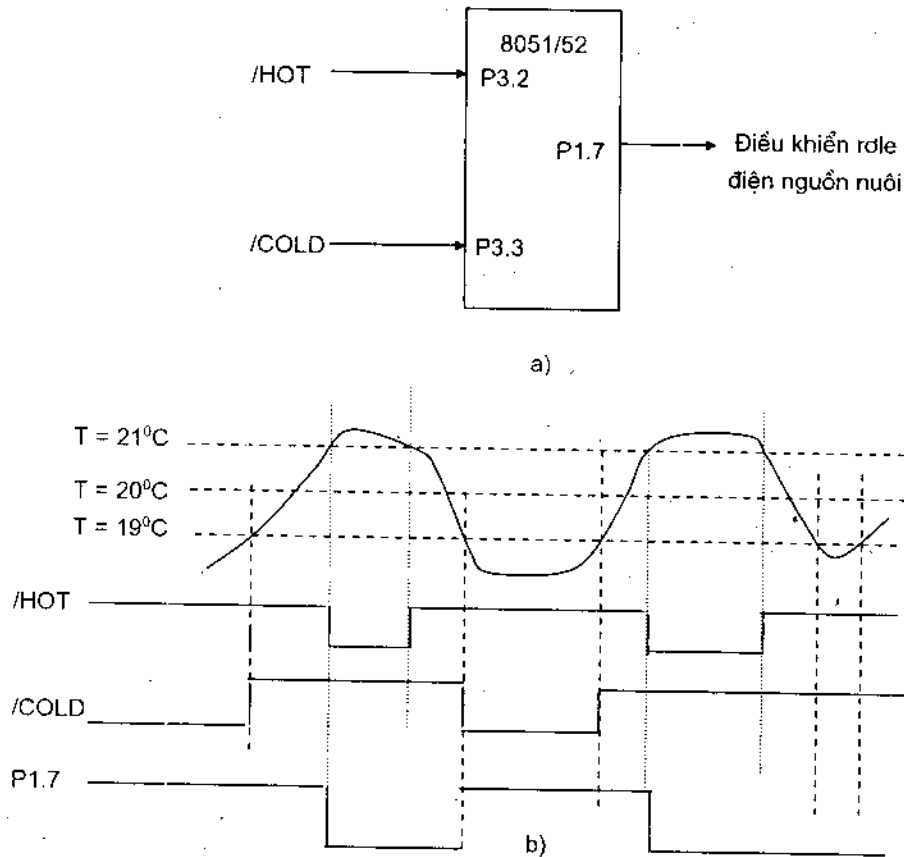
Với thời gian lớn, đơn vị đo (hay t_0) là giây, số đếm N lớn có thể tới giây, phút, giờ như trong đồng hồ, khi đó phải dùng nhiều ô nhớ để đếm giây, phút, giờ như trong đồng hồ số.

Có hai cách đo độ rộng xung: đo độ rộng xung âm và đo kiểu bắt đầu (Start) và kết thúc (Stop) của một xung.

a) Đo thời gian với hai tín hiệu xung bắt đầu/kết thúc (Start/Stop)

Quá trình đo thời gian như sau:

- Khi có ngắt ngoài /INT0: khi có xung Start (mức 0 hay sườn âm) vào lối vào /INT0 (chân P3.2) sẽ có ngắt ngoài EXINT0, chương trình sẽ chuyển tới chương trình phục vụ ngắt ngoài EXINT0 để khởi phát bộ định thời phát và đếm thời gian đơn vị t_0 .



Hình 10-15. Sơ đồ mắc (a) và giản đồ xung (b) của các tín hiệu điều khiển lò nung

- Khi có ngắt ngoài /INT1: khi có xung Stop (mức 0 hay sườn âm) vào lối vào /INT1 (chân P3.3) sẽ có ngắt ngoài EXINT1, chương trình sẽ chuyển tới chương trình phục vụ ngắt ngoài EXINT1 để kết thúc đếm ở bộ định thời t_0 .

Còn ngắt thời gian được dùng để tạo đơn vị thời gian đo t_0 , ta có thể:

+ Dùng chu kỳ thời gian 1 μsec của máy phát nhịp: Khi đó, khoảng thời gian đo được tối đa là 65535 μsec tức 65,535 msec.

+ Dùng bội số một khoảng thời gian cố định của máy phát nhịp: ta có chương trình con tạo ngắt với một biến đếm ngắt (bđn) dùng thanh ghi cho tăng (hay giảm) từ 2 tới giá trị nào đó.

Chương trình đo khoảng thời gian dùng ngắt cũng tương tự chương trình đo khoảng thời gian không dùng ngắt, chỉ có điều khác là dùng ngắt của Timer để tạo thời gian đơn vị (10 microsec, 50 microsec v.v...).

Ví dụ 10-24 giới thiệu chương trình đo thời gian dùng hai tín hiệu Start/Stop đưa vào hai lối vào của ngắt ngoài P3.2, P3.3.

Thanh ghi R1 chỉ số giây đo được. Thanh ghi R0 chỉ số 50msec, còn các thanh ghi TH0, TL0 chỉ hai byte cao và thấp của số bù 65535 của microsec (số microsec byte cao = 65535 - TH0 và byte thấp = 65535 - TL0). Các dữ liệu trên được đưa ra các cổng song song P0 - P3. Nếu muốn chỉ thị ra đèn LED 7 đoạn, ta phải thêm chương trình con giải mã 7 đoạn và điều khiển LED 7 đoạn (xem ví dụ về đồng hồ số).

Chương trình trên có thể dùng để đo độ kéo dài một xung âm. Khi đó, xung âm phải tạo dạng thành hai xung âm ở thời điểm bắt đầu và kết thúc xung (qua mạch vi phân và đảo tạo dạng).

b) Đo độ kéo dài của một xung dương

Nếu đo thời gian kéo dài của một xung dương ta phải:

; Ví dụ 10-24: Chương trình đo thời gian	
<pre> ORG 00H LJMP MAIN ORG 03H ; ngắt /INT0 LJMP EX0ISR ; nhảy tới ngắt ngoài 0 ORG 13H LJMP EX1ISR ; nhảy tới ngắt ngoài 1 ORG 30H ; chương trình chính MAIN: MOV IE, #85H ; cho phép ngắt ngoài 0, 1 SETB IT0 ; ngắt sườn xung /INT0 SETB IT1 ; ngắt sườn xung /INT1 SETB PX1 MOV TMOD, #01H ; chế độ đếm 16 bit CLR TF0 JNB TF0, \$; chờ tín hiệu tràn tf0 LAP0: MOV R0, #21; giảm đi 20 lần cho 1 sec LAP: MOV TH0, #HIGH(-50000) ; số đếm đầu MOV TL0, #LOW(-50000) ; số đếm ban đầu JNB TF0, \$; chờ tín hiệu tràn tf0 CLR TF0 DJNZ R0, LAP ; trở về đếm số 50 msec </pre>	<pre> INC R1 ; tăng thanh ghi đếm số giây JNB P3.3, KT ; nếu P3.3 = 1 tới kt SJMP LAP0 ; trở về đếm r1 sec KT: SJMP \$; kết thúc EX0ISR: SETB TR0 ; phục vụ ngắt 0 RETI ; trở về từ ngắt EX1ISR: CLR TR0 ; tắt timer0 để dừng đếm MOV P0, R1 ; chỉ thị số giây MOV A, #21 MOV B, R0 SUBB A, B MOV P1, A ; chỉ thị số 50microsec MOV A, #0FFH MOV B, TH0 SUBB A, B MOV P2, A ; byte cao của microsec MOV A, #0FFH MOV B, TL0 SUBB A, B MOV P3, A ; byte thấp của microsec RETI ; trở về từ ngắt END ; kết thúc dịch </pre>

- Biến đổi một xung dương thành hai xung âm, độ kéo dài nhỏ để có hai xung Start và Stop ở hai sườn xung xuống và lên bằng mạch vi phân và đảo cực tính xung cho cả hai xung. Sau đó, có thể đo bằng phương pháp dùng hai ngắt ngoài 0, 1 như đã mô tả ở trên.

; Ví dụ 10-25: Chương trình đo độ rộng xung Tx	
<pre> ORG 00H LJMP MAIN ORG 03H LJMP EX0ISR ORG 30H ; chương trình chính MAIN: SETB EA SETB EX0 SETB IT0 MOV TMOD, #08H ; chế độ đếm 16 bit CLR TF0 SETB TR0 CLR P3.2 LAP0: MOV R0, #21; giảm đi 20 lần cho 1 sec LAP: MOV TH0, #HIGH(-50000) ; số đếm đầu MOV TL0, #LOW(-50000) ; số đếm đầu JNB TF0, \$; chờ tín hiệu tràn tf0 CLR TF0 DJNZ R0, LAP ; trở về đếm số 50 msec </pre>	<pre> inc r1 ; tăng thanh ghi đếm giây sjmp lap0 ; trở về đếm r1 sec ex0isr: clr tf0 mov ie, #00h mov P0, r1 mov a, #21 mov b, r0 subb a, b mov P1, a mov a, #0FFh mov b, Th0 subb a, b mov P2, a mov a, #0FFh mov b, Tl0 subb a, b mov P3, a sjmp \$ end ; kết thúc dịch </pre>

- Đưa xung cần đo vào một trong hai lối vào của ngắt ngoài, nhưng timer0, 1 phải ghi ở chế độ đếm với bit C = 1 ở thanh ghi TCON như ở chương 8 về Timer.
+ Xóa bit P3.2 hay P3.3 để chờ xung dương vào kích thích đếm.

+ Xác lập TR0 hay TR1, các timer0,1 vẫn chưa đếm vì chưa có xung vào lối vào đếm T0 (chân P3.4-số 14) hay T1 (chân P3.5-số 15).

- Có xung dương cần đo độ rộng xung t_x , bit P3.2 hay P3.3 lên mức cao, chuỗi xung nhịp từ bộ dao động thạch anh vào bộ đếm Timer0 hay Timer1 (xem chương 8 về Timer). Khi kết thúc xung, lối vào P3.2 hay P3.3 lên cao. Chuyển mạch hồ, không có chuỗi xung vào đếm.

- Ngắt ngoài /INT0 hay /INT1 tạo ra để ghi các kết quả đo t_x của các thanh ghi và các bộ đếm TH, TL.

Ví dụ 10-25 giới thiệu chương trình đo độ rộng xung dương với ngắt ngoài 0.

Chương trình này tương tự ví dụ 10-24 nhưng đơn giản hơn. Bộ đếm timer0 ở chế độ đếm (bit C=1). Khi có xung dương INT0 vào P3.2, bộ đếm bắt đầu đếm. Khi kết thúc xung, P3.2 xuống logic 0, bộ đếm ngừng. Nội dung thanh ghi R1 chỉ số giây đưa ra cổng P0, nội dung 21-R0 chỉ số 50 mic-rosec đưa ra cổng P1. Số bù 0FFh của TH0, TL0 chỉ byte cao và thấp của số microsec đưa ra các cổng P2, P3.

c) Đo độ kéo dài của một xung âm

Ở chương 8, ta đã đề cập đến việc đo độ kéo dài của một xung âm. Khi đó, ta phải:

- Xác lập Timer ở chế độ đếm với bit C = 1 ở thanh ghi TCON để đếm xung.

- Xác lập TR = 1 để sẵn sàng đếm chuỗi xung đơn vị t_0 từ máy phát xung nhịp.

- Xung âm cần đo độ rộng xung đưa vào lối vào ngắt ngoài /INT tương ứng. Khi có xung âm vào, lối vào /INT có sườn âm, chuyển mạch S2 (xem hình 8-1) đóng mạch, bộ đếm TH, TL bắt đầu đếm chuỗi xung t_0 .

- Khi kết thúc xung, mức của ngắt ngoài /INT lên 1, chuyển mạch S2 mở, bộ đếm TH, TL ngừng đếm. Giá trị của TH, TL chính là số đếm Nx của thời gian đơn vị t_0 .

CÂU HỎI VÀ THỰC HÀNH

I. Câu hỏi

1. Ngắt chương trình là gì ? Nêu những ưu và nhược điểm của ngắt chương trình.
2. Nêu các biện pháp xử lý ngắt chương trình : cho phép/cấm ngắt, ưu tiên ngắt, chuyển thực hiện chương trình con phục vụ ngắt và trở về chương trình chính bị ngắt.
3. Nêu sự khác nhau giữa chương trình con và chương trình phục vụ ngắt.
4. Ngắt cưỡng bức là gì ? Hãy so sánh ngắt cứng với ngắt mềm.
5. Có thể mở rộng nhiều ngắt mềm được không ? Cách lập trình.
6. Nêu các ứng dụng của ngắt do định thời.
7. Nêu các ứng dụng của ngắt ngoài. Hãy so sánh ngắt ngoài với ngắt do định thời.
8. Nêu các phép đo độ rộng xung dương, xung âm.
9. Nêu ứng dụng của ngắt nối tiếp. Hãy so sánh ngắt nối tiếp với ngắt ngoài.
10. Hãy thiết kế hệ VDK có nhiều ngắt ngoài để nhận nhiều tín hiệu vào điều khiển.

II. Thực hành

1. Quan sát các thanh ghi và các cờ ngắt liên quan tới ngắt của VDK 8051/52.
Mở chương trình mô phỏng PINNACLE, mở thực đơn VIEW để xem các thanh ghi, các Timer và kết quả thực hiện việc truyền, nhận tín hiệu nối tiếp.
2. Chép các ví dụ 10-1 tới ví dụ 10-25 để hợp dịch, liên kết, cho chạy từng bước để quan sát sự thay đổi của các thanh ghi, các cờ, tín thu hay tín phát trên cổng nối tiếp v.v... để hiểu rõ hoạt động của các ngắt. Đặc biệt lưu ý khi sắp xảy ra ngắt của Timer. Với ngắt ngoài, phải cho tín hiệu vào bằng cách kích chuột vào các bit P3.2, P3.3 (/INT0, /INT1). Với đếm xung ngoài, phải cho xung vào các lối vào T0, T1.
3. Ghi một số chương trình đã hợp dịch và chạy thử tốt vào nhớ PROM bên trong của VDK rồi cho chạy chương trình trên KIT VDK.
4. Quan sát cách chuyển chương trình khi xác lập các bit cờ TF0, TF1, TF2, RI, TI và kích thích các lối vào P3.2, P3.3 cho có logic 0.
5. Lập lại câu 4 với hai cờ cùng xuất hiện và cấm bit cho phép ngắt IE, ET0, ET1, ET2, EXEN2.
6. Kiểm tra bảng thứ tự ưu tiên của các ngắt và các lệnh ưu tiên ngắt cho các ngắt bằng cách cho các ngắt xảy ra đồng thời.

III. Thực hành nâng cao

1. Vẽ giải thuật của một số chương trình ví dụ.
2. Kết hợp hai hay nhiều hành động của VDK trong một chương trình ở chương trình chính và chương trình phục vụ ngắt. Dịch và cho chạy từng bước trên PINNACLE để quan sát quá trình diễn biến của chương trình.
3. Chép các ví dụ 8-1 tới ví dụ 8-8 để hợp dịch, liên kết, cho chạy từng bước để quan sát sự thay đổi của các thanh ghi, các cờ, tín thu hay phát trên cổng nối tiếp v.v. để hiểu rõ hoạt động của các ngắt. Đặc biệt lưu ý khi sắp xảy ra ngắt của Timer. Với ngắt ngoài, phải cho tín hiệu vào bằng cách kích với logic 0 vào các chân /INT0 (P3.2) hay /INT1 (P3.3). Với đếm xung ngoài, phải kích logic 0 vào các chân T0 (P3.4) hay T1 (P3.5).

PHỤ LỤC

PHỤ LỤC A

Bảng mã ASCII

Dec	Hex	Ch	Code	Dec	Hex	Ch	Dec	Hex	Ch	Dec	Hex	Ch
0	0		NUL	33	21	!	65	41	A	97	61	a
1	1		SOH	34	22	"	66	42	B	98	62	b
2	2		STX	35	23	#	67	43	C	99	63	c
3	3		ET	36	24	\$	68	44	D	100	64	d
4	4		EOT	37	25	%	69	45	E	101	65	e
5	5		ENQ	38	26	&	70	46	F	102	66	f
6	6		ACK	39	27	'	71	47	G	103	67	g
7	7		BEL	40	28	(	72	48	H	104	68	h
8	8		BS	41	29	)	73	49	I	105	69	i
9	9		HT	42	2A	*	74	50	J	106	6A	j
10	0A		LF	43	2B	+	75	51	K	107	6B	k
11	0B		VT	44	2C	,	76	4C	L	108	6C	l
12	0C		FF	45	2D	-	77	4D	M	109	6D	m
13	0D		CR	46	2E	.	78	4E	N	110	6E	n
14	0E		SO	47	2F	/	79	4F	O	111	6F	o
15	0F		SI	48	30	0	80	50	P	112	70	p
16	10		DLE	49	31	1	81	51	Q	113	71	q
17	11		DC1	50	32	2	82	52	R	114	72	r
18	12		DC2	51	33	3	83	53	S	115	73	s
19	13		DC3	52	34	4	84	54	T	116	74	t
20	14		DC4	53	35	5	85	55	U	117	75	u
21	15		NAK	54	36	6	86	56	V	118	76	v
22	16		SYN	55	37	7	87	57	W	119	77	w
23	17		ETB	56	38	8	88	58	X	120	78	x
24	18		CAN	57	39	9	89	59	Y	121	79	y
25	19		EM	58	3A	:	90	5A	Z	122	7A	z
26	1A		SUB	59	3B	;	91	5B	[	123	7B	{
27	1B		ES	60	3C	<	92	5C	]	124	7C	
28	1C		FS	61	3D	=	93	5D	\	125	7D	}
29	1D		GS	62	3E	>	94	5E	^	126	7E	~
30	1E		RS	63	3F	?	95	5F	_	127	7F	^
31	1F		US	64	40	@	96	60	`			
32	20		SP									

PHỤ LỤC B. CÁC THANH GHI SFR

Ký hiệu	Tên thanh ghi	Địa chỉ
ACC	Thanh ghi tích lũy A	0E0H
B	Thanh ghi B	0F0H
PSW	Lời trạng thái	0D0H
SP	Con trỏ ngăn xếp	81H
DPTR	Con trỏ địa chỉ (hyte byte)	82H-83H
DPL	Byte thấp của con trỏ địa chỉ	82H
DPH	Byte cao của con trỏ địa chỉ	83H
P0	Cổng song song 0	80H
P1	Cổng song song 1	90H
P2	Cổng song song 2	0A0H
P3	Cổng song song 3	0B0H
IP	Điều khiển ưu tiên ngắt	0B8H
IE	Điều khiển cho phép ngắt	0A8H
TMOD	Điều khiển chế độ bộ định thời/dếm 0,1	89H
TCON	Điều khiển/trạng thái của bộ định thời/dếm 0,1	88H
T2MOD	Điều khiển chế độ bộ định thời/dếm 2	0C8H
T2CON	Điều khiển/trạng thái của bộ định thời/dếm 2	0C9H
TH0	Byte cao của bộ đếm định thời/dếm 0	8CH
TLO	Byte thấp của bộ đếm định thời/dếm 0	8AH
TH1	Byte cao của bộ đếm định thời/dếm 1	8DH
TL1	Byte thấp của bộ đếm định thời/dếm 1	8BH
TH2	Byte cao của bộ đếm định thời/dếm 2	0CDH
TL2	Byte thấp của bộ đếm định thời/dếm 2	0CCH
RCAP2H	Byte cao của thanh ghi bắt giữ của bộ định thời/dếm 2	0CBH
RCAP2L	Byte thấp của thanh ghi bắt giữ của bộ định thời/dếm 2	0CAH
SCON	Thanh ghi điều khiển/trạng thái của cổng nối tiếp	98H
SBUF	Bộ đệm của cổng nối tiếp	99H
PCON	Thanh ghi điều khiển nguồn	87H

D7	D6	D5	D4	D3	D2	D1	D0
CY	AC	F0	RS1	RS0	OV	-	P

CY	PSW.7	; cờ nhớ trong phép cộng, trừ và so sánh		
AC	PSW.6	; cờ phụ hiệu chỉnh thập phân		
F0	PSW.5	; dành cho người dùng sau		
RS1	PSW.4	; bit 1 của chọn băng thanh ghi thông dụng R0-R7		
RS0	PSW.3	; bit 0 của chọn băng thanh ghi thông dụng R0-R7		
OV	PSW.2	; cờ tràn khi dữ liệu vượt quá số bit của thanh ghi		
-	PSW.1	; dành cho người dùng		
P	PSW.0	; cờ chắn/lẻ		
RS1		RS0	Băng thanh ghi	Địa chỉ
0		0	0	00H-07H
0		1	1	08H-0FH
1		0	2	10H-17H
1		1	3	18H-1FH

Hình B-1. Các bit của thanh ghi trạng thái PSW

D7	D6	D5	D4	D3	D2	D1	D0
EA	-	ET2	ES	ET1	EX1	ET0	EX0

Ký hiệu	Vị trí	Chức năng
EA	IE.7	Cho phép (EA = 1) hay cấm (EA = 0) tất cả các ngắt
-	IE.6	Chưa sử dụng
ET2	IE.5	Cho phép (ET2 = 1) hay cấm (ET2 = 0) ngắt TIMER2
ES	IE.4	Cho phép (ES = 1) hay cấm (ES = 0) ngắt nối tiếp
ET1	IE.3	Cho phép (ET1 = 1) hay cấm (ET1 = 0) ngắt do TIMER1
EX1	IE.2	Cho phép (EX1 = 1) hay cấm (EX1 = 0) ngắt do ngắt ngoài INT1
ET0	IE.1	Cho phép (ET0 = 1) hay cấm (ET0 = 0) ngắt do TIMER0
EX0	IE.0	Cho phép (EX0 = 1) hay cấm (EX0 = 0) ngắt do ngắt ngoài INT0

Hình B-2. Thanh ghi cho phép/cấm ngắt IE

D7	D6	D5	D4	D3	D2	D1	D0
--	--	PT2	PS	PT1	PX1	PT0	PX0

----	IP.7	dự trữ
----	IP.6	dự trữ
PT2	IP.5	bit ưu tiên ngắt cho timer2 khi đếm tràn
PS	IP.4	bit ưu tiên ngắt cho cổng nối tiếp
PT1	IP.3	bit ưu tiên ngắt cho timer1
PX1	IP.2	bit ưu tiên ngắt cho ngắt ngoài INT1
PT1	IP.3	bit ưu tiên ngắt cho timer0
PX0	IP.2	bit ưu tiên ngắt cho ngắt ngoài INT0

Hình B-3. Các bit của thanh ghi cho ưu tiên ngắt IP

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Ký hiệu	Vị trí	Ý nghĩa	Ký hiệu	Vị trí	Ý nghĩa
TF1	TCON.7	Bit trạng thái cờ đếm tràn của Timer1. Xác lập bởi phần cứng khi đếm tràn về 0000H. Xóa bởi lệnh hay xử lý ngắt chương trình.	IE1	TCON.3	Bit trạng thái cờ ngắt sườn xung 1. Xác lập bởi phần cứng khi phát hiện ngắt ngoài INT1. Xóa khi xử lý ngắt
TR1	TCON.6	Bit điều khiển Timer1. Xác lập/xóa bởi lệnh để khởi động hay dừng đếm cho Timer1.	IT1	TCON.2	Bit điều khiển ngắt ngoài 1. Xác lập/xóa bởi lệnh để chỉ sườn xuống/mức thấp của ngắt ngoài.
TF0	TCON.5	Bit trạng thái cờ đếm tràn của Timer0. Xác lập bởi phần cứng khi đếm tràn về 0000H. Xóa bởi lệnh hay xử lý ngắt chương trình.	IE0	TCON.1	Bit trạng thái cờ Ngắt sườn xung 0. Xác lập bởi phần cứng khi phát hiện ngắt ngoài INT0. Xóa khi xử lý ngắt.
TR0	TCON.4	Bit điều khiển Timer1. Xác lập/xóa bởi lệnh để khởi động hay dừng đếm cho Timer1.	IT0	TCON.0	Bit điều khiển ngắt ngoài 1. Xác lập/xóa bởi lệnh để chỉ sườn xuống/mức thấp của ngắt ngoài.

Hình B-4. Các bit của thanh ghi điều khiển-trạng thái TCON

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Bit	Tên	Timer	Mô tả
7	GATE	1	Khi Gate = 1, Timer chỉ làm việc khi INT0
6	C//T	1	Bit điều khiển đếm xung nhịp (= 0) cho Timer hay đếm xung ngoài (= 1) cho Counter.
5	M1	1	Bit chọn chế độ của Timer1
4	M0	1	Bit chọn chế độ của Timer1
3	GATE	0	Khi Gate = 1, Timer chỉ làm việc khi INT1
2	C//T	0	Bit điều khiển đếm xung nhịp (= 0) cho Timer hay đếm xung ngoài (= 1) cho Counter.
1	M1	0	Bit chọn chế độ của Timer0
0	M0	0	Bit chọn chế độ của Timer0

M1	M0	Chế độ	Đặc điểm
0	0	0	Hai thanh ghi TH, TL là bộ đếm 13 bit
0	1	1	Hai thanh ghi TH, TL là bộ đếm 16 bit
1	0	2	TL là hai bộ đếm 8 bit còn thanh ghi TH lưu giá trị nạp lại cho TL
1	1	3	Thanh ghi TH, TL là hai bộ đếm độc lập 8 bit

Hình B-5. Các bit của thanh ghi THOD

D7	D6	D5	D4	D3	D2	D1	D0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C//T2	C//RL2

Bit	Ký hiệu	Vị trí	Ý nghĩa
7	TF2	T2CON.7	Cờ tràn Timer2, xác lập bởi đếm tràn Timer2 và bị xoá bởi lệnh. TF2 không xác lập bởi RCLK = 1 hay TCLK = 1
6	EXF2	T2CON.6	Cờ Timer2 ngoài, xác lập khi hoặc bị bắt giữ hay nạp lại gây ra bởi một dịch chuyển âm trên T2EX và EXEN2 = 1. Khi ngắt Timer2 cho phép, EXF2 = 1 sẽ gây ra cho CPU trở tới chương trình phục vụ ngắt Timer2. EXF2 bị xoá bởi lệnh.
5	RCLK	T2CON.5	Cờ nhịp nhận. Khi xác lập, gây ra cổng nối tiếp dùng các xung tràn Timer2 cho nhịp nhận tin nối tiếp ở chế độ 1 và 3. RCLK = 0 gây ra tràn Timer1 được dùng cho nhịp nhận tin nối tiếp.
4	TCLK	T2CON.4	Cờ nhịp truyền. Khi xác lập, gây ra cổng nối tiếp dùng các xung tràn Timer2 cho nhịp truyền tin nối tiếp ở chế độ 1 và 3. RCLK = 0 gây ra tràn Timer1 được dùng cho nhịp truyền tin nối tiếp.
3	EXEN2	T2CON.3	Cờ cho phép ngắt Timer2 ngoài. Khi xác lập, cho phép bắt hoặc nạp lại để tạo ra một dịch chuyển âm trên T2EX nếu Timer2 không được dùng để nhịp cho cổng nối tiếp. EXEN2 = 0 gây ra Timer2 không biết sự kiện ở T2EX.
2	TR2	T2CON.2	Điều khiển khởi động/dừng cho Timer2. Một logic 1 khởi phát Timer2.
1	C//T2	T2CON.1	Chọn chế độ Timer (= 0) để đếm xung bên trong hay Counter (= 1) để đếm xung ngoài vào.
0	CP//RL2	T2CON.0	Cờ Capture (bắt)/Reload (nạp lại). Khi xác lập, việc bắt giữ sẽ xảy ra ở dịch chuyển âm ở T2EX nếu EXEN = 1. Khi bị xoá, việc nạp lại tự động sẽ xảy ra hoặc với tràn của Timer2 hoặc với dịch chuyển âm ở T2EX khi EXEN2 = 1. Khi RCLK = 1 hay TCLK = 1, bit này sẽ không được biết đến và Timer bị cưỡng bức tới tự động nạp lại khi Timer2 tràn.

Hình B-6. Các bit của thanh ghi T2CON

Bảng B-1. Các bit của thanh ghi SCON

Địa chỉ bit	9FH D7	9EH D6	9DH D5	9CH D4	9BH D3	9AH D2	99H D1	98H D0
Tên bit	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Bit	Ký hiệu	Công dụng	Tốc độ baud
7	SM0 SM1 0 0 0 1	Chế độ nối tiếp 0: chế độ thanh ghi dịch, tốc độ baud cố định Chế độ nối tiếp 1: 8 bit UART (nối tiếp không đồng bộ), tốc độ baud thay đổi	fosc/12 tốc độ thay đổi
6	1 0 1 1	Chế độ nối tiếp 2: 9 bit UART (truyền nối tiếp không đồng bộ) tốc độ fosc/32 hay fosc/64 hay fosc/32 Chế độ nối tiếp 3: 9bit UART (truyền nối tiếp không đồng bộ) tốc độ thay đổi.	fosc/64 hay fosc/32 tốc độ baud thay đổi
5	SM2	Cho phép hành động truyền thông nhiều vi xử lý ở chế độ 2 và 3. Ở chế độ 0, SM2 sẽ bằng 0. Ở chế độ 2 và 3, SM2 xác lập lên 1. RI sẽ không bị tác động nếu bit thứ 9 (RB8) nhận được là 0. Ở chế độ 2 và SM2 = 1, RI sẽ không bị tác động nếu một bit stop hợp lệ không được nhận.	
4	REN	Cho phép nhận nối tiếp. Xác lập bởi lệnh để cho phép nhận. Xóa bởi lệnh để từ chối nhận.	
3	TB8	Bit truyền 8. Bit dữ liệu thứ 9 sẽ được truyền ở chế độ 2 và 3. Xác lập và xóa bởi lệnh như mong muốn (truyền 9 bit tin, kể cả bit chẵn lẻ). Ở chế độ 0, RB8 không dùng.	
2	RB8	Bit nhận 8. Bit dữ liệu thứ 9, sẽ được nhận ở chế độ 2 và 3. Ở chế độ 1, nếu SM2 = 0, RB8 là bit stop mà nó nhận. Ở chế độ 0, RB8 không dùng.	
1	TI	Cờ ngắt truyền. Xác lập bởi phần cứng ở cuối của bit thứ 8 ở chế độ 0 hay ở bắt đầu của bit stop ở các chế độ khác trong bất kỳ truyền nối tiếp nào. Phải bị xóa bằng lệnh (phần mềm).	
0	RI	Cờ ngắt nhận. Xác lập bởi phần cứng ở cuối của bit thứ 8 ở chế độ 0 hay ở trong thời gian của bit stop ở các chế độ khác trong bất kỳ truyền nối tiếp nào. Phải bị xóa bằng lệnh (phần mềm).	

Hình B-7. Thanh ghi điều khiển nối tiếp SCON

PHỤ LỤC C

Hệ lệnh của VDK 8051/8052

Ghi chú	Giải thích
Ký hiệu	được thay thế bởi
←	nội dung của
()	dữ liệu được trữ bởi
(())	một trong 8 thanh ghi R0-R7
rrr	các bit dữ liệu
ddddddd	các bit địa chỉ
aaaaaaaa	địa chỉ của một bit
bbbbbbbb	nội dung ở nhớ có địa chỉ là giá trị của R0 hay R1
@R0 hay @R1	
ACALL addr11	
Chức năng	Gọi chương trình con, có địa chỉ tuyệt đối là 11 bit
Số byte	2
Số chu kỳ	2
Mã lệnh	aaa10001 aaaaaaaa
ADD A <src-byte>	
Chức năng	cộng thanh ghi A với <byte src>, kết quả gửi vào A và chỉ thị lên thanh ghi trạng thái PSW
Hoạt động	(A) <-- (A) +(src-byte)
ADD A, Rn	
Chức năng	cộng thanh ghi chứa A với nội dung của thanh ghi Rn, kết quả gửi vào A và chỉ thị lên thanh ghi trạng thái PSW
Số byte	1
Số chu kỳ	1
Mã lệnh	00101rrr
Hoạt động	(A) <-- (A) +(R.n.)
ADD A, direct	
Chức năng	cộng thanh ghi chứa A với nội dung của ô nhớ có địa chỉ là con số direct, kết quả gửi vào A và chỉ thị lên thanh ghi trạng thái PSW
Số byte	2
Số chu kỳ	1
Mã lệnh	00100101 aaaaaaaa
Hoạt động	(A) <-- (A) +(direct)
ADD A, @Ri	
Chức năng	cộng thanh ghi chứa A với ô nhớ có địa chỉ gián tiếp là nội dung của thanh ghi Ri (với i = 0 hay 1), kết quả gửi vào A và chỉ thị lên thanh ghi trạng thái PSW
Số byte	1
Số chu kỳ	1
Mã lệnh	0010011i
Hoạt động	(A) <-- (A) +(Ri)
ADD A, #data	
Chức năng	cộng thanh ghi chứa A với dữ liệu tức thì là data, kết quả gửi vào A và chỉ thị lên thanh ghi trạng thái PSW
Số byte	2
Số chu kỳ	1
Mã lệnh	00100100 dddddddd
Hoạt động	(A) <-- (A) + #data
ADDC A, <scr-byte>	
Chức năng	cộng có nhớ thanh ghi chứa A với nội dung <scr-byte>, kết quả gửi vào A và chỉ thị lên thanh ghi trạng thái PSW
ADDC A, Rn	
Chức năng	cộng có nhớ (cộng thêm bit cờ nhớ C) nội dung thanh ghi chứa A với nội dung của thanh ghi Rn, kết quả gửi vào A và chỉ thị lên thanh ghi trạng thái PSW
Số byte	1

Số chu kỳ	1
Mã lệnh	00110rrr
Hoạt động	$(A) \leftarrow (A) + (C) + (Rn)$
ADDC A, direct	
Chức năng	cộng có nhớ thanh ghi chứa A với nội dung của ô nhớ có địa chỉ là con số direct, kết quả gửi vào A và chỉ thị lên thanh ghi trạng thái PSW
Số byte	2
Số chu kỳ	1
Mã lệnh	00110101 aaaaaaaa
Hoạt động	$(A) \leftarrow (A) + (C) + (\text{direct})$
ADDC A, @Ri	
Chức năng	cộng có nhớ thanh ghi chứa A với ô nhớ có địa chỉ gián tiếp là nội dung của thanh ghi Ri (với i = 0 hay 1), kết quả gửi vào A và chỉ thị lên thanh ghi trạng thái PSW
Số byte	1
Số chu kỳ	1
Mã lệnh	0011011i
Hoạt động	$(A) \leftarrow (A) + (C) + (Ri)$
ADDC A, #data	
Chức năng	cộng có nhớ thanh ghi chứa A với dữ liệu tức thì là data, kết quả gửi vào A và chỉ thị lên thanh ghi trạng thái PSW
Số byte	2
Số chu kỳ	1
Mã lệnh	00110100 dddddddd
Hoạt động	$(A) \leftarrow (A) + (C) + \#data$
AJMP addr11	
Chức năng	nhảy đến địa chỉ tuyệt đối 11 bit
Số byte	2
Số chu kỳ	2
Mã lệnh	aaa00001 aaaaaaaa
ANL <dst-byte>, <src-byte>	
Chức năng	AND hai toán hạng <dst-byte>, <src-byte>
ANL A, Rn	
Chức năng	AND hai toán hạng <A>, <Rn>
Số byte	1
Số chu kỳ	1
Mã lệnh	01011rrr aaaaaaaa
Hoạt động	$(A) \leftarrow (A) \text{ AND } (Rn)$
ANL A, direct	
Chức năng	AND hai toán hạng <A> với nội dung ô nhớ có địa chỉ là con số direct
Số byte	2
Số chu kỳ	1
Mã lệnh	01010101 aaaaaaaa
Hoạt động	$(A) \leftarrow (A) \text{ AND } (\text{direct})$
ANL A, @Ri	
Chức năng	AND hai toán hạng <A>, <Rn>
Số byte	1
Số chu kỳ	1
Mã lệnh	010101i aaaaaaaa (với i = 0, 1)
Hoạt động	$(A) \leftarrow (A) \text{ AND } ((Ri))$
ANL A, #data	
Chức năng	AND hai toán hạng <A>, #data
Số byte	2
Số chu kỳ	1
Mã lệnh	01010100 dddddddd
Hoạt động	$(A) \leftarrow (A) \text{ AND } \#data$

ANL direct, #data	
Chức năng	AND hai toán hạng <A>, #data
Số byte	3
Số chu kỳ	1
Mã lệnh	01010011 dddddddd
Hoạt động	(direct) <-- (direct) AND #data
ANL C, <src-bit>	
Chức năng	AND hai bit <C>, <src-bit>
Số byte	2
Số chu kỳ	1
Mã lệnh	01011rrr aaaaaaaa
Hoạt động	(C) <-- (C) AND (src-bit)
ANL C, bit	
Chức năng	AND hai bit <C>, <src-bit>
Số byte	1
Số chu kỳ	1
Mã lệnh	01011rrr aaaaaaaa
Hoạt động	(C) <-- (C) AND (bit)
ANL C, /bit	
Chức năng	AND hai bit <C>, </bit> (đảo của bit hay CPL bit)
Số byte	2
Số chu kỳ	2
Mã lệnh	10110000 bbbbbbb
Hoạt động	(C) <-- (C) AND NOT (bit)
LCALL	tương tự ACALL nhưng địa chỉ là 16 bit
CJNE <dest-byte>, <src-byte>, rel	
Chức năng	so sánh <dest-byte> với <src-byte>, nếu không bằng, nhảy tới địa chỉ rel
CJNE A, direct, rel	
Chức năng	so sánh thanh ghi A với nội dung ô nhớ có địa chỉ direct, nếu không bằng, nhảy tới địa chỉ rel
Số byte	3
Số chu kỳ	2
Mã lệnh	10110101 aaaaaaaa eeeeeeee
Hoạt động	(PC) <-- (PC) + 3 IF (A) <> (direct) THEN (PC) <-- (PC) + địa chỉ tương đối rel IF (A) < (direct) THEN (C) <-- 1 ELSE (C) <-- 0
CJNE Rn, #data, rel	
Chức năng	so sánh thanh ghi Rn với dữ liệu tức thì #data, nếu không bằng, nhảy tới địa chỉ rel
Số byte	3
Số chu kỳ	2
Mã lệnh	10111rrr dddddddd eeeeeeee
Hoạt động	(PC) <-- (PC) + 3 IF (Rn) <> #data THEN (PC) <-- (PC) + địa chỉ tương đối rel IF (Rn) < #data THEN (C) <-- 1 ELSE (C) <-- 0
CJNE Rn, #data, rel	
Chức năng	so sánh thanh ghi Rn với dữ liệu tức thì #data, nếu không bằng, nhảy tới địa chỉ rel
Số byte	3
Số chu kỳ	2
Mã lệnh	10111rrr dddddddd eeeeeeee
Hoạt động	(PC) <-- (PC) + 3 IF (Rn) <> #data THEN (PC) <-- (PC) + địa chỉ tương đối rel IF (Rn) < #data THEN (C) <-- 1 ELSE (C) <-- 0
CJNE @Ri, #data, rel	
Chức năng	so sánh thanh ghi Rn với dữ liệu tức thì #data, nếu không bằng, nhảy tới địa chỉ rel
Số byte	3
Số chu kỳ	2
Mã lệnh	10111011i eeeeeeee (với i = 0 hay 1)

<i>Hoạt động</i>	(PC) <-- (PC) +3 IF ((Ri)) <> #data THEN (PC) <-- (PC) + địa chỉ tương đối rel IF ((Ri)) < #data THEN (C) <-- 1 ELSE (C) <-- 0
CLR A	xoá thanh ghi chứa A
<i>Chức năng</i>	1
<i>Số byte</i>	1
<i>Số chu kỳ</i>	1
<i>Mã lệnh</i>	11100100
<i>Hoạt động</i>	(A) <-- 0
CLR bit	xoá bit
<i>Chức năng</i>	bit <-- 0
<i>Hoạt động</i>	
CLR C	xoá thanh ghi chứa A
<i>Chức năng</i>	1
<i>Số byte</i>	1
<i>Số chu kỳ</i>	1
<i>Mã lệnh</i>	11000011
<i>Hoạt động</i>	(C) <-- 0
CLR bit	xoá thanh ghi chứa A
<i>Chức năng</i>	2
<i>Số byte</i>	1
<i>Số chu kỳ</i>	1
<i>Mã lệnh</i>	11000010
<i>Hoạt động</i>	(bit) <-- 0
CPL A	lấy bù (hay đảo - NOT) thanh ghi chứa A
<i>Chức năng</i>	1
<i>Số byte</i>	1
<i>Số chu kỳ</i>	1
<i>Mã lệnh</i>	11110100
<i>Hoạt động</i>	(A) <-- NOT (A)
CPL bit	lấy bù (hay đảo- NOT) bit
<i>Chức năng</i>	1
<i>Số byte</i>	1
<i>Số chu kỳ</i>	1
<i>Mã lệnh</i>	10110010 bbbbbbbb
<i>Hoạt động</i>	(bit) <-- NOT (bit)
CPL C	lấy bù (hay đảo- NOT) bit C
<i>Chức năng</i>	1
<i>Số byte</i>	1
<i>Số chu kỳ</i>	1
<i>Mã lệnh</i>	110110011
<i>Hoạt động</i>	(C) <-- NOT (C)
DA A	hiệu chỉnh thập phân nội dung của thanh ghi A sau phép cộng hay đổi số dạng HEX sang dạng BCD (thập phân mã 2)
<i>Chức năng</i>	1
<i>Số byte</i>	1
<i>Số chu kỳ</i>	1
<i>Mã lệnh</i>	11010100
<i>Hoạt động</i>	Nếu A < 9 thì A+0 ---> A (để nguyên). Nếu A > 9 (là Ah, Bh, Ch, Dh, Eh, Fh) Thì A+6 và số hàng đơn vị ---> (A), 1 --> số hàng chục
DEC byte	giảm byte đi 1
<i>Chức năng</i>	byte <-- byte - 1
<i>Hoạt động</i>	
DEC A	giảm thanh ghi chứa A đi 1
<i>Chức năng</i>	1
<i>Số byte</i>	1
<i>Số chu kỳ</i>	1
<i>Mã lệnh</i>	00010100
<i>Hoạt động</i>	(A) <-- (A) - 1

DEC Rn

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

DEC direct

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

DEC @Ri

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

DIV AB

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

DJNZ <byte>, <rel-addr>

Chức năng

DJNZ Rn, rel

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

INC byte

Chức năng

INC A

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

INC Rn

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

INC direct

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

INC @Ri

Chức năng

giảm thanh ghi Rn đi 1

1

1

00011rrr

(Rn) <-- (Rn) - 1

giảm thanh ghi Rn đi 1

2

1

00010101 aaaaaaaa

(direct) <-- (direct) - 1

giảm nội dung ô nhớ có địa chỉ là nội dung của thanh ghi Ri (i = 0 hay 1) đi 1

1

1

0001011i

(Rn) <-- (Rn) - 1

chia thanh ghi A cho thanh ghi B. Số thương trong thanh ghi A, số dư trong thanh ghi B. Cờ nhớ C và cờ tràn OV bị xoá.

1

4

10000100

(A) <-- số thương (hay MODUN của A/B), (B) <-- số dư của A/B.

giảm <byte> đi 1, nếu <byte> chưa bằng 0, nhảy chương trình tới địa chỉ <rel- addr>

giảm nội dung của thanh ghi Rn đi 1, nếu Rn chưa bằng 0, nhảy chương trình tới địa chỉ rel.

3

2

11010101 aaaaaaaa eeeeeeee

(PC) <-- (PC)+2

(direct) <-- (direct)-1 IF (direct) <> 0 THEN (PC) <-- (PC) + byte_2

tăng nội dung của byte lên 1

tăng nội dung của byte lên 1

1

1

00000100

(A) <-- (A)+1

tăng nội dung của byte lên 1

1

1

00001rrr

(Rn) <-- (Rn)+1

tăng nội dung của byte lên 1

2

1

00000101 aaaaaaaa

(direct) <-- (direct)+1

tăng nội dung ô nhớ có địa chỉ Ri (với i = 0, hay 1) lên 1

<i>Số byte</i>	1	
<i>Số chu kỳ</i>	1	
<i>Mã lệnh</i>	0000111i	
<i>Hoạt động</i>	((Ri)) <-- ((Rn))+1	
INC DPTR		tăng nội dung của con trỏ địa chỉ nhớ ngoài DPTR lên 1
<i>Số byte</i>	1	
<i>Số chu kỳ</i>	2	
<i>Mã lệnh</i>	10100011	
<i>Hoạt động</i>	(DPTR) <-- (DPTR)+1	
JB bit, rel		nhảy chương trình tới địa chỉ rel nếu bit được xác lập lên 1
<i>Chức năng</i>	3	
<i>Số byte</i>	2	
<i>Số chu kỳ</i>	00100000 bbbbbbbb eeeeeeee	
<i>Mã lệnh</i>	(PC) <-- (PC)+3 IF (bit) = 1 THEN (PC) <-- (PC) + byte_2	
<i>Hoạt động</i>		nhảy chương trình tới địa chỉ rel nếu bit được xác lập lên 1 rồi xóa bit
JBC bit, rel	3	
<i>Chức năng</i>	2	
<i>Số byte</i>	00010000 bbbbbbbb eeeeeeee	
<i>Số chu kỳ</i>	(PC) <-- (PC)+3 IF (bit) = 1 THEN (PC) <-- (PC) + byte_2 và (bit) = 0	
<i>Mã lệnh</i>		nhảy chương trình tới địa chỉ rel nếu bit C được xác lập lên 1.
<i>Hoạt động</i>	2	
JC bit, rel	2	
<i>Chức năng</i>	01000000 eeeeeeee	
<i>Số byte</i>	(PC) <-- (PC)+2 IF (C) = 1 THEN (PC) <-- (PC) + byte_2.	
<i>Số chu kỳ</i>		
<i>Mã lệnh</i>		
<i>Hoạt động</i>		
JMP <dest> xem SJMP, AJMP hay LJMP		
JMP @A+DPTR		nhảy gián tiếp chương trình tới địa chỉ A+DPTR.
<i>Chức năng</i>	1	
<i>Số byte</i>	2	
<i>Số chu kỳ</i>	01110011 eeeeeeee	
<i>Mã lệnh</i>	(PC) <-- (PC)+(A)+(DPTR).	
<i>Hoạt động</i>		nhảy gián tiếp chương trình tới địa chỉ rel nếu bit = 0 (không được xác lập = 1).
JNB bit, rel	3	
<i>Chức năng</i>	2	
<i>Số byte</i>	00110000 bbbbbbbb eeeeeeee	
<i>Số chu kỳ</i>	(PC) <-- (PC)+3 IF (bit) = 0 THEN (PC) <-- (PC) + byte_2	
<i>Mã lệnh</i>		nhảy gián tiếp chương trình tới địa chỉ rel nếu bit C = 0 (không được xác lập = 1).
<i>Hoạt động</i>	2	
JNC rel	2	
<i>Chức năng</i>	01010000 eeeeeeee	
<i>Số byte</i>	(PC) <-- (PC)+2 IF (C) = 0 THEN (PC) <-- (PC) + byte_2	
<i>Số chu kỳ</i>		nhảy gián tiếp chương trình tới địa chỉ rel nếu thanh ghi chứa A khác 0.
<i>Mã lệnh</i>	2	
<i>Hoạt động</i>	01110000 eeeeeeee	
JNZ rel	(PC) <-- (PC)+2 IF (A) <> 0 THEN (PC) <-- (PC) + byte_2	
<i>Chức năng</i>		nhảy gián tiếp chương trình tới địa chỉ rel nếu thanh ghi chứa A bằng 0.
<i>Số byte</i>	2	
<i>Số chu kỳ</i>	2	
<i>Mã lệnh</i>	01100000 eeeeeeee	

Hoạt động	
LCALL addr16	(PC) <-- (PC)+2 IF (A) = 0 THEN (PC) <-- (PC) + byte_2
Chức năng	gọi dài một chương trình ở địa chỉ 16 bit (byte 2 chứa các bit A15 tới A8, byte 3 chứa các bit A7-A0)
Số byte	3
Số chu kỳ	2
Mã lệnh	00010010 aaaaaa aaaaaa
Hoạt động	(PC) <-- (PC)+3 (PC) <-- addr 15- addr0
JMP addr16	
Chức năng	nhảy dài một chương trình tới địa chỉ 16 bit (byte 2 chứa các bit A15 tới A8, byte 3 chứa các bit A7-A0)
Số byte	3
Số chu kỳ	2
Mã lệnh	00010010 aaaaaa aaaaaa
Hoạt động	(PC) <-- addr 15- addr0
MOV <dest-byte>, <src-byte>	
Chức năng	địch chuyển dữ liệu từ byte nguồn, <src-byte> tới byte đích <dest-byte>
MOV A, Rn	
Chức năng	địch chuyển dữ liệu từ thanh ghi Rn tới thanh ghi A
Số byte	1
Số chu kỳ	1
Mã lệnh	11101rrr aaaaaa
Hoạt động	(A) <-- (Rn)
MOV A, direct	
Chức năng	địch chuyển dữ liệu từ ô nhớ có địa chỉ trực tiếp direct tới thanh ghi A
Số byte	2
Số chu kỳ	1
Mã lệnh	11100101 aaaaaa
Hoạt động	(A) <-- (direct)
MOV A, @Ri	
Chức năng	địch chuyển dữ liệu từ ô nhớ có địa chỉ gián tiếp là nội dung thanh ghi Ri (i = 0 hay 1) tới thanh ghi A
Số byte	1
Số chu kỳ	1
Mã lệnh	1110011i
Hoạt động	(A) <-- ((Ri))
MOV A, #data	
Chức năng	địch chuyển dữ liệu tức thì #data tới thanh ghi A
Số byte	2
Số chu kỳ	1
Mã lệnh	01110100 dddddddd
Hoạt động	(A) <-- #data
MOV Rn, A	
Chức năng	địch chuyển dữ liệu từ thanh ghi A tới thanh ghi Rn
Số byte	1
Số chu kỳ	1
Mã lệnh	11111rrr aaaaaa
Hoạt động	(Rn) <-- (A)
MOV Rn, direct	
Chức năng	địch chuyển dữ liệu từ ô nhớ có địa chỉ trực tiếp direct tới thanh ghi Rn
Số byte	2
Số chu kỳ	2
Mã lệnh	10101rrr aaaaaa
Hoạt động	(Rn) <-- (direct)
MOV Rn, #data	
Chức năng	địch chuyển dữ liệu từ ô nhớ có địa chỉ trực tiếp direct tới thanh ghi A
Số byte	2

Số chu kỳ	1
Mã lệnh	01111rrr aaaaaaa
Hoạt động	(Rn) <-- (#data)
MOV direct, A	
Chức năng	địch chuyển dữ liệu từ thanh ghi A tới ô nhớ có địa chỉ trực tiếp direct.
Số byte	2
Số chu kỳ	1
Mã lệnh	11110101 aaaaaaa
Hoạt động	(direct) <-- (A)
MOV direct, Rn	
Chức năng	địch chuyển dữ liệu từ thanh ghi A tới ô nhớ có địa chỉ trực tiếp direct.
Số byte	2
Số chu kỳ	2
Mã lệnh	10001rrr aaaaaaa
Hoạt động	(direct) <-- (Rn)
MOV direct, direct	
Chức năng	địch chuyển dữ liệu từ thanh ghi A tới ô nhớ có địa chỉ trực tiếp direct.
Số byte	3
Số chu kỳ	2
Mã lệnh	10000101 aaaaaaa (nguồn) aaaaaaa (đích)
Hoạt động	(direct) <-- (direct)
MOV direct, @Ri (i = 0 hay 1)	
Chức năng	địch chuyển dữ liệu từ ô nhớ gián tiếp có địa chỉ trong thanh ghi Ri tới ô nhớ có địa chỉ trực tiếp direct.
Số byte	2
Số chu kỳ	2
Mã lệnh	1000011i aaaaaaa
Hoạt động	(direct) <-- ((Ri))
MOV direct, #data	
Chức năng	địch chuyển dữ liệu tức thì #data tới ô nhớ có địa chỉ trực tiếp direct.
Số byte	3
Số chu kỳ	2
Mã lệnh	01110101 aaaaaaa dddddddd
Hoạt động	(direct) <-- #data
MOV @Ri, direct	(i = 0 hay 1)
Chức năng	địch chuyển dữ liệu từ ô nhớ có địa chỉ trực tiếp direct vào ô nhớ có địa chỉ là nội dung thanh ghi Ri.
Số byte	2
Số chu kỳ	2
Mã lệnh	1010011i aaaaaaa
Hoạt động	((Ri)) <-- (direct)
MOV @Ri, #data	
Chức năng	địch chuyển dữ liệu từ thanh ghi A tới ô nhớ có địa chỉ trực tiếp direct.
Số byte	2
Số chu kỳ	1
Mã lệnh	0111011i dddddddd
Hoạt động	((Ri)) <-- #data
MOV <dest-bit>, <src-bit>	
Chức năng	địch chuyển bit nguồn <src-bit> tới bit đích <dest-bit>.
MOV C, bit	
Chức năng	địch chuyển bit nguồn tới bit đích C.
Số byte	2
Số chu kỳ	1
Mã lệnh	10100010 bbbbbbbb
Hoạt động	(C) <-- (bit)
MOV bit, C	
Chức năng	địch chuyển bit nguồn C tới bit đích bit.
Số byte	2

Số chu kỳ	2
Mã lệnh	10010010 bbbbbbbb
Hoạt động	(bit) <-- (C)
MOV DPTR, #data16	
Chức năng	dịch chuyển dữ liệu #data16 (16 bit) tới thanh ghi con trỏ địa chỉ nhớ ngoài DPTR.
Số byte	2
Số chu kỳ	2
Mã lệnh	10010010 bbbbbbbb
Hoạt động	(DPTR) <-- #data16
MOVC A, @A+ <base-reg>	
Chức năng	dịch chuyển dữ liệu của ô nhớ có địa chỉ là tổng của thanh ghi A+ <base-reg> tới thanh ghi chứa A.
MOVC A, @A+ DPTR	
Chức năng	dịch chuyển dữ liệu của ô nhớ có địa chỉ là tổng của thanh ghi A+ DPTR tới thanh ghi chứa A.
Số byte	1
Số chu kỳ	2
Mã lệnh	10010011
Hoạt động	A, <-- ((A+ (DPTR)))
MOVC A, @A+ PC	
Chức năng	dịch chuyển dữ liệu của ô nhớ có địa chỉ là tổng của thanh ghi A+ DPTR tới thanh ghi chứa A.
Số byte	1
Số chu kỳ	2
Mã lệnh	10000011
Hoạt động	A, <-- ((A+ (PC)))
MOVX <dest-byte>, <src-byte>	
Chức năng	dịch chuyển dữ liệu của ô nhớ ngoài có địa chỉ là <src-byte> tới ô nhớ ngoài có địa chỉ là <dest-byte>
MOVX A, @Ri	
Chức năng	dịch chuyển dữ liệu của ô nhớ ngoài có địa chỉ là nội dung của thanh ghi Ri tới thanh ghi chứa A
Số byte	1
Số chu kỳ	2
Mã lệnh	1110011i
Hoạt động	A, <-- ((Ri))
MOVX A, @DPTR	
Chức năng	dịch chuyển dữ liệu của ô nhớ ngoài có địa chỉ là nội dung thanh ghi DPTR tới thanh ghi chứa A.
Số byte	1
Số chu kỳ	2
Mã lệnh	11100000
Hoạt động	(A), <-- ((DPTR))
MOVX @Ri, A	
Chức năng	dịch chuyển dữ liệu của ô nhớ ngoài có địa chỉ là nội dung thanh ghi DPTR tới thanh ghi chứa A.
Số byte	1
Số chu kỳ	2
Mã lệnh	111i 0011
Hoạt động	((Ri)), <-- (A)
MOVX @DPTR, A	
Chức năng	dịch chuyển dữ liệu của thanh ghi chứa A tới ô nhớ ngoài có địa chỉ là nội dung thanh ghi DPTR.
Số byte	1
Số chu kỳ	2
Mã lệnh	1111 0000
Hoạt động	((DPTR)), <-- (A)

MUL AB

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

Nhân A với B

1

4

10100100

(B) <-- byte cao của tích AB

(A) <-- byte thấp của tích AB

NOP

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

Không làm gì

1

1

00000000

(PC) <--(PC)+1

ORL <dest-byte>, <src-byte>

Chức năng

OR logic toán hạng nguồn <src-byte> với toán hạng đích <dest-byte>, kết quả ghi trong toán hạng đích <dest-byte>.

ORL A, Rn

Chức năng

OR logic toán hạng nguồn <src-byte> với toán hạng đích <dest-byte>, kết quả ghi trong toán hạng đích <dest-byte>.

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

2

1

01001rrr aaaaaaaa

(A) <--(A) OR (Rn)

ORL A, direct

Chức năng

OR logic toán hạng nguồn A với toán hạng đích là ô nhớ có địa chỉ direct, kết quả ghi trong toán hạng đích A.

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

2

1

01000101 aaaaaaaa

(A) <--(A) OR (direct)

ORL A, @Ri

Chức năng

OR logic toán hạng nguồn A với toán hạng đích là ô nhớ có địa chỉ Ri, kết quả ghi trong toán hạng đích A.

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

2

1

0100011i aaaaaaaa

(A) <--(A) OR ((Ri)).

ORL A, #data

Chức năng

OR logic toán hạng nguồn A với toán hạng đích là ô nhớ có địa chỉ tức thì #data, kết quả ghi trong toán hạng đích A.

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

2

1

01000100 dddddddd

(A) <--(A) OR #data

ORL direct, A

Chức năng

OR logic toán hạng nguồn là ô nhớ có địa chỉ direct, với toán hạng đích A, kết quả ghi trong toán hạng đích là ô nhớ có địa chỉ direct.

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

2

1

01000010 aaaaaaaa

(direct) <--(direct) OR (A)

ORL direct, #data

Chức năng

OR logic toán hạng nguồn là giá trị tức thì #data với ô nhớ có địa chỉ direct, kết quả ghi trong toán hạng đích là ô nhớ có địa chỉ direct.

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

3

2

01000011 aaaaaaaa dddddddd

(direct) <--(direct) OR #data

ORL C, <src-bit>*Chức năng*

OR logic toán hạng nguồn là bit C với toán hạng đích <src-bit>, kết quả ghi trong toán hạng đích là bit C.

ORL C, bit*Chức năng*

OR logic toán hạng nguồn là bit C với toán hạng đích là bit, kết quả ghi trong toán hạng đích là bit C.

Số byte

2

Số chu kỳ

2

Mã lệnh

01110010 bbbbbbbb

Hoạt động

(C) <--(C) OR (bit)

ORL C, /bit*Chức năng*

OR logic toán hạng nguồn là bit C với toán hạng đích là /bit, kết quả ghi trong toán hạng đích là bit C.

Số byte

2

Số chu kỳ

2

Mã lệnh

10100000 bbbbbbbb

Hoạt động

(C) <--(C) OR NOT (bit)

POP direct*Chức năng*

lấy nội dung ô nhớ ngăn xếp (stack) có địa chỉ là nội dung thanh ghi SP vào ô nhớ có địa chỉ là direct.

Số byte

2

Số chu kỳ

2

Mã lệnh

11010000 aaaaaaaa

Hoạt động

(direct) <--((SP))

(SP) <-- (SP) - 1

PUSH direct*Chức năng*

chất nội dung ô nhớ có địa chỉ là direct vào ngăn xếp có địa chỉ là thanh ghi con trỏ ngăn xếp SP.

Số byte

2

Số chu kỳ

2

Mã lệnh

11000000 aaaaaaaa

Hoạt động

(SP) <-- (SP) - 1

((SP)) <-- (direct).

RET*Chức năng*

trở về từ chương trình con

Số byte

1

Số chu kỳ

2

Mã lệnh

00100010 aaaaaaaa

Hoạt động

(PC15- PC8) <-- ((SP))

(SP) <-- (SP) - 1.

(PC7- PC0) <-- ((SP))

(SP) <-- (SP) - 1.

RETI*Chức năng*

trở về từ chương trình con phục vụ ngắt

Số byte

1

Số chu kỳ

2

Mã lệnh

00110010 aaaaaaaa

Hoạt động

(PC15- PC8) <-- ((SP))

(SP) <-- (SP) - 1.

(PC7- PC0) <-- ((SP))

(SP) <-- (SP) - 1.

RL A*Chức năng*

địch (quay) các bit của thanh ghi A về bên trái

Số byte

1

Số chu kỳ

1

Mã lệnh

00100011 aaaaaaaa

Hoạt động

(An+1) <-- (An), n = 0-6

(A0) <-- A7

RLC A

Chức năng
Số byte
Số chu kỳ
Mã lệnh
Hoạt động

địch (quay) các bit của thanh ghi A về bên trái qua bit C

1
1
00110011 aaaaaaaa
(An+1) <-- (An), n = 0-6
(A0) <-- (C)
(C) <-- (A7)

RR A

Chức năng
Số byte
Số chu kỳ
Mã lệnh
Hoạt động

địch (quay) các bit của thanh ghi A về bên phải

1
1
00000011 aaaaaaaa
(An) <-- (An+1), n = 0-6
(A7) <-- (A0)

RRC A

Chức năng
Số byte
Số chu kỳ
Mã lệnh
Hoạt động

địch (quay) các bit của thanh ghi A về bên phải qua bit C

1
1
00010011 aaaaaaaa
(An) <-- (An+1), n = 0-6
(A7) <-- (C)
(C) <-- (A0)

SETB <bit>

Chức năng

xác lập bit lên 1

SETB C

Chức năng

xác lập bit C lên 1

Số byte

1

Số chu kỳ

1

Mã lệnh

11010011

Hoạt động

(C) <-- 1

SETB bit

Chức năng

xác lập bit lên 1

Số byte

2

Số chu kỳ

1

Mã lệnh

11010010 bbbbbbbb

Hoạt động

(bit) <-- 1

SJMP rel

Chức năng

nhảy chương trình ngắn tới địa chỉ rel

Số byte

2

Số chu kỳ

2

Mã lệnh

10000000 eeeeeeee

Hoạt động

(PC) <-- (PC)+2

(PC) <-- (PC)+ byte_2

SUBB A, <src-byte>

Chức năng

trừ có mượn B của thanh ghi A với <src-byte>

SUBB A, Rn

Chức năng

trừ có mượn B của thanh ghi A với <src-byte>

Số byte

1

Số chu kỳ

1

Mã lệnh

10011rrr

Hoạt động

(A) <-- (A)- (C)- (Rn)

SUBB A, direct

Chức năng

trừ có mượn B của thanh ghi A với nội dung ở nhớ có địa chỉ direct

Số byte

2

Số chu kỳ

1

Mã lệnh

10010101 aaaaaaaa

Hoạt động

(A) <-- (A)- (C)- (direct)

SUBB A, @Ri

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

SWAP A

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

XCH A, <byte>

Chức năng

XCH A, Rn

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

XCH A, direct

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

XCHD A, @Ri

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

XRL <dest-byte>, <src-byte>

Chức năng

XRL A, Rn

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

XRL A, direct

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

XRL A, @Ri

Chức năng

Số byte

Số chu kỳ

Mã lệnh

Hoạt động

trừ có mượn B của thanh ghi A với nội dung ô nhớ có địa chỉ là Ri với i = 0 hay 1.

1

1

10010111i

(A) <-- (A) - (C) - ((Ri))

trao đổi 4 bit cao với 4 bit thấp của thanh ghi A

1

1

11000100

(A3- A0) <-- ----> (A7- A4)

trao đổi nội dung của thanh ghi A với byte

1

1

trao đổi nội dung của thanh ghi A với byte

1

1

1101rrr

(A) <-- ----> (Rn)

trao đổi nội dung của thanh ghi A với nội dung ô nhớ có địa chỉ direct.

2

1

11000101 aaaaaaaa

(A) <-- ----> (direct)

trao đổi nội dung nửa thấp của thanh ghi A với nội dung nửa thấp của ô nhớ có địa chỉ Ri.

1

1

1101011i aaaaaaaa

(A3-A0) <-- ----> (Ri3-Ri0)

logic XOR (loại trừ) của <dest-byte> với <src-byte>, kết quả đặt trong <dest-byte>.

1

1

logic XOR (loại trừ) của A với thanh ghi Rn, kết quả đặt trong A.

1

1

01101rrr

(A) <-- (A) XOR (Rn)

logic XOR (loại trừ) của A với ô nhớ có địa chỉ direct, kết quả đặt trong A.

1

1

01100101 aaaaaaaa

(A) <-- (A) XOR (direct)

logic XOR (loại trừ) của A với ô nhớ có địa chỉ là thanh ghi Ri (với i = 0 hay 1), kết quả đặt trong A.

1

1

0110011i

(A) <-- (A) XOR ((Ri))

XRL A, #data*Chức năng**Số byte**Số chu kỳ**Mã lệnh**Hoạt động***XRL direct, A***Chức năng**Số byte**Số chu kỳ**Mã lệnh**Hoạt động***XRL direct, #data***Chức năng**Số byte**Số chu kỳ**Mã lệnh**Hoạt động*

logic XOR (loại trừ) của A với dữ liệu tức thì #data, kết quả đặt trong A.

2

1

01100100 dddddddd

(A) <-- (A) XOR #data

logic XOR (loại trừ) của nội dung ô nhớ có địa chỉ direct với thanh ghi A, kết quả đặt trong ô nhớ có địa chỉ direct.

2

1

01100010

(direct) <-- (A) XOR (direct)

logic XOR (loại trừ) của nội dung ô nhớ có địa chỉ direct với dữ liệu tức thì #data, kết quả đặt trong ô nhớ có địa chỉ direct.

3

2

01100011

(direct) <-- (direct) XOR #data

PHỤ LỤC D

Chương trình RAM.LST

1 0000 \$Title	41 00F6 ExInt1 EQU 613h
2 0000 \$Date	42 00F6 T1Int EQU 61Bh
3 0000 \$MOD51	43 00F6 SerInt EQU 623h
4 0000 LF EQU 0AH	45 00F6 org 0000h
5 0000 CR EQU 0DH	46 0000 ljmp Start
6 0000 ESC EQU 1BH	48 0003 org 0003h
7 0000 StartChar EQU ':'	49 0003 ljmp ExInt0
8 0000 Slash EQU '/'	50 0006 reti
9 0000 Skip EQU 13	52 0007 org 000Bh
11 0000 Ch DS 0Fh	53 000B ljmp T0Int
12 000F State DS 10H	54 000E reti
13 001F DataByte DS 11H	56 000F org 0013h
14 0030 ByteCount DS 12H	57 0013 ljmp ExInt1
15 0042 HighAddr DS 13H	58 0016 reti
16 0055 LowAddr DS 14H	60 0017 org 001Bh
17 0069 RecType DS 15H	61 001B ljmp T1Int
18 007E ChkSum DS 16H	62 001E reti
19 0094 HASave DS 17H	64 001F org 0023h
20 00AB LASave DS 18H	65 0023 ljmp SerInt
21 00C3 FilChkHi DS 19H	66 0026 reti
22 00DC FilChkLo DS 1AH	68 0027 Start: mov IE, #0
24 00F6 Flags EQU 20H	69 002A mov SP, #5Fh
25 00F6 HexFlag BIT Flags.0	70 002D acall SerStart
26 00F6 EndFlag BIT Flags.1	71 002F acall CRLF
27 00F6 DoneFlag BIT Flags.2	72 0031 mov a, #'='
28 00F6 DatSkipFlag BIT Flags.3	73 0033 acall PutChar
30 00F6 EFlags EQU 21H	74 0035 acall HexIn
31 00F6 ErrFlag1 BIT EFlags.0	75 0037 acall ErrPrt
32 00F6 ErrFlag2 BIT EFlags.1	77 0039 mov A, EFlags
33 00F6 ErrFlag3 BIT EFlags.2	78 003B jz Hexok
34 00F6 ErrFlag4 BIT EFlags.3	80 003D ErrLoop: MOV A, #'?'
35 00F6 ErrFlag5 BIT EFlags.4	81 003F acall PutChar
36 00F6 ErrFlag6 BIT EFlags.5	82 0041 acall GetChar
37 00F6 ; khai báo nhớ ngoài	83 0043 sjmp ErrLoop
39 00F6 ExInt0 EQU 603h	85 0045 Hexok: mov EFlags, #0
40 00F6 T0Int EQU 60Bh	86 0048 acall GetChar

87 004A cjne a, #Slash, Hexok ; chờ "/"	131 0094 MOV A, FilChkHi
89 004D acall GetByte	132 0096 ACALL PrByte
90 004F j b ErrFlag1, Hexok	133 0098 MOV A, FilChkLo
91 0052 mov HighAddr, DataByte	134 009A ACALL PrByte
93 0055 acall GetByte	135 009C MOV A, #')'
94 0057 j b ErrFlag1, Hexok	136 009E ACALL PutChar
95 005A mov LowAddr, DataByte	137 00A0 ACALL CRLF
97 005D acall GetChar ; chờ CR	138 00A2 RET
98 005F cjne A, #CR, Hexok	139 00A3 ; tìm, chạy trạng thái "state"
100 0062 mov a, #'@'; truyền lệnh GO là @	141 00A3 GoState: MOV A, State
101 0064 acall PutChar	142 00A5 ANL A, #0FH
102 0066 jnb TI, \$	143 00A7 RL A
103 0069 push LowAddr	144 00A8 MOV DPTR, #StateTable
104 006B push HighAddr	145 00AB mp @A+DPTR
105 006D ret	147 00AC StateTable: AJMP StWait
106 006E ; kết thúc chương trình chính	148 00AE AJMP StLeft
107 006E ; SUBROUTINE Input	149 00B0 AJMP StGetCnt
109 006E HexIn:	150 00B2 AJMP StLeft
110 006E CLR A	151 00B4 AJMP StGetAd1
111 006F MOV State, A	152 00B6 AJMP StLeft
112 0071 MOV Flags, A	153 00B8 AJMP StGetAd2
113 0073 MOV HighAddr, A	154 00BA AJMP StLeft
114 0075 MOV LowAddr, A	155 00BC AJMP StGetRec
115 0077 MOV HASave, A	156 00BE AJMP StLeft
116 0079 MOV ChkSum, A	157 00C0 AJMP StGetDat
117 007B MOV FilChkHi, A	158 00C2 AJMP StLeft
118 007D MOV FilChkLo, A	159 00C4 AJMP StGetChk
119 007F MOV EFlags, A	160 00C6 AJMP StSkip
120 0081 SETB ErrFlag4	161 00C8 AJMP BadState
122 0083 StateLoop: ACALL GetChar	162 00CA AJMP BadState
; kiểm tra trạng thái của ký tự	164 00CC ; chờ bắt đầu
123 0085 ACALL AscHex	165 00CC StWait: MOV A, Ch
124 0087 MOV Ch, A	166 00CE CJNE A, #StartChar, SWEX
125 0089 ACALL GoState	167 00D1 INC State
126 008B JNB DoneFlag, StateLoop	168 00D3 SWEX: RET
128 008E ACALL PutChar	170 00D4 ; xử lý nibble đầu của byte hex bất kỳ
129 0090 MOV A, #'('	171 00D4 StLeft: MOV A, Ch; thay ký tự vào
130 0092 ACALL PutChar	172 00D6 JNB HexFlag, SLERR

173 00D9	ANL A, #0Fh	216 0115	RET
174 00DB	SWAP A	218 0116	; thử tệp bản ghi cho dòng
175 00DC	MOV DataByte, A	219 0116	StGetRec: ACALL StRight
176 00DE	INC State	220 0118	MOV A, DataByte
177 00E0	RET	221 011A	MOV RecType, A
178 00E1		222 011C	JZ SGRDat
179 00E1	SLERR: SETB ErrFlag1; lỗi	223 011E	CJNE A, #1, SGRERR
180 00E3	SETB DoneFlag	224 0121	SETB EndFlag
181 00E5	RET	225 0123	SETB DatSkipFlag
183 00E6	; xử lý nibble thứ hai của byte hex	226 0125	MOV State, #11
184 00E6	StRight: MOV A, Ch	227 0128	SJMP SGREX
185 00E8	JNB HexFlag, SRERR	228 012A	SGRDat: INC State
186 00EB	ANL A, #0Fh	229 012C	SGREX: RET
187 00ED	ORL A, DataByte	231 012D	SGRErr: SETB ErrFlag2
188 00EF	MOV DataByte, A		; sai, loại bản ghi sai
189 00F1	ADD A, ChkSum	232 012F	SETB DoneFlag
190 00F3	MOV ChkSum, A	233 0131	RET
191 00F5	RET	235 0132	; nhận một byte dữ liệu
193 00F6	SRERR: SETB ErrFlag1; không phải	236 0132	StGetDat: ACALL StRight
194 00F8	SETB DoneFlag	237 0134	JB DatSkipFlag, SGD1
195 00FA	RET	238 0137	ACALL Store
197 00FB	; đếm byte dữ liệu cho một dòng	239 0139	MOV A, DataByte
198 00FB	StGetCnt: ACALL StRight	240 013B	ADD A, FilChkLo
199 00FD	MOV A, DataByte	241 013D	MOV FilChkLo, A
200 00FF	MOV ByteCount, A	242 013F	CLR A
201 0101	INC State	243 0140	ADDC A, FilChkHi
202 0103	RET	244 0142	MOV FilChkHi, A
204 0104	; thử byte địa chỉ cao cho dòng	245 0144	MOV A, DataByte
205 0104	StGetAd1: ACALL StRight	246 0146	SGD1: DJNZ ByteCount, SGDEX
206 0106	MOV A, DataByte	247 0149	INC State
207 0108	MOV HighAddr, A	248 014B	SJMP SGDEX2
208 010A	INC State	249 014D	SGDEX: DEC State
209 010C	RET	250 014F	SGDEX2: RET
211 010D	; thử byte địa chỉ thấp cho dòng	252 0150	; GET CHECKSUM
212 010D	StGetAd2: ACALL StRight	253 0150	StGetChk: ACALL StRight
213 010F	MOV A, DataByte	254 0152	JNB EndFlag, SGC1
214 0111	MOV LowAddr, A	255 0155	SETB DoneFlag
215 0113	INC State	256 0157	SJMP SGCEX

258	0159	SGC1: MOV A, ChkSum	299	0199	; SerStart
259	015B	JNZ SGCErr	300	0199	; GetChar
260	015D	MOV ChkSum, #0	301	0199	; GetByte
261	0160	MOV State, #0	302	0199	; PutChar
262	0163	MOV LASave, LowAddr	303	0199	; AscHex
263	0166	MOV HASave, HighAddr	304	0199	; ErrPrt
264	0169	SGCEX: RET	305	0199	; CRLF
266	016A	SGCErr: SETB ErrFlag3	306	0199	; PrtByte
		; lỗi kiểm tra dòng	308	0199	SerStart:
267	016C	SETB DoneFlag	309	0199	MOV A, PCON
268	016E	RET	310	019B	CLR Acc.7
270	016F	; trạng thái này dùng để bỏ qua	311	019D	mov PCON, a
		sự gửi dữ liệu thêm vào, không biết nó	312	019F	mov TH1, #0FDh
271	016F	StSkip: RET	313	01A2	mov TL0, #0FDh
273	0170	; chờ nếu có trạng thái sai	314	01A5	mov TMOD, #20h
274	0170	BadState: MOV State, #Skip	315	01A8	mov TCON, #40h
275	0173	RET	316	01AB	mov SCON, #52h
277	0174	; tích dữ liệu vào RAM	317	01AE	ret
278	0174	Store: MOV DPH, HighAddr	319	01AF	GetByte:
279	0177	MOV DPL, LowAddr	320	01AF	acall GetChar
280	017A	MOV A, DataByte	321	01B1	acall AscHex
281	017C	MOVX @DPTR, A	322	01B3	mov Ch,a
282	017D	MOVX A, @DPTR	323	01B5	acall StLeft
283	017E	CJNE A, DataByte, StoreErr	324	01B7	acall GetChar
285	0181	CLR ErrFlag4	325	01B9	acall AscHex
286	0183	INC DPTR	326	01BB	mov Ch, A
287	0184	MOV HighAddr, DPH	327	01BD	acall StRight
288	0187	MOV LowAddr, DPL	328	01BF	ret
289	018A	CLR A	330	01C0	GetChar:
290	018B	CJNE A, HighAddr, StoreEx	331	01C0	jnb RI, \$
		; kiểm tra tràn địa chỉ	332	01C3	clr RI
291	018E	CJNE A,LowAddr, StoreEx	333	01C5	mov A, SBUF
292	0191	SETB ErrFlag5; xác lập cờ tràn	334	01C7	cjne a, #ESC, GCEX
293	0193	StoreEx: RET	335	01CA	ljmp Start
294	0194	StoreErr: SETB ErrFlag6	336	01CD	GCEX: ret
295	0196	SETB DoneFlag	338	01CE	PutChar:
296	0198	RET	339	01CE	jnb TI, \$
298	0199	; SUBROUTINES	340	01D1	CLR TI

341 01D3 mov SBUF,A	380 0214 call PutChar
342 01D5 ret	381 0217 mov A, EFlags
343 01D6 ; biến đổi Asc sang Hex	382 0219 jz ErrPrEx
344 01D6 AscHex:	383 021B call PrByte
345 01D6 cjne A, #'0', AH1	384 021E ErrPrEx: ret
346 01D9 AH1: jc AHBAd	386 021F ; CRLF
347 01DB cjne A, #'9'+1,AH2	388 021F CRLF:
348 01DE AH2: jc AHVal09	389 021F mov a, #CR
349 01E0 cjne A, #'A', AH3	390 0221 call PutChar
350 01E3 AH3: jc AHBAd	391 0224 mov A, # LF
351 01E5 cjne A, #'F'+1,AH4	392 0226 call PutChar
352 01E8 AH4: jc AHValAF	393 0229 ret
353 01EA cjne A, #'a', AH5	395 022A ; gửi byte ASCII ra cổng nối tiếp
354 01ED AH5: jc AHBAd	396 022A PrByte: push Acc
355 01EF cjne A, #'f'+1,AH6	397 022C Swap A
356 01F2 AH6: jnc AHBAd	398 022D call HexAsc
357 01F4 clr c	399 0230 call PutChar
358 01F5 subb A, #27h	400 0233 pop Acc
359 01F7 sjmp AHVal09	401 0235 call HexAsc
360 01F9 AHBAd: clr HexFlag	402 0238 call PutChar
361 01FB sjmp AHEX	403 023B ret
362 01FD AHValAF: clr c	405 023C ORG 603H
363 01FE subb a, #7	406 0603 MOV P0, #55H
364 0200 AHVal09: clr C	407 0606 RETI
365 0201 subb A, #'0'	409 0607 ORG 60BH
366 0203 setb HexFlag	410 060B MOV P1, #0AAH
367 0205 AHEX: ret	411 060E RETI
369 0206 ; biến đổi Hex - Asc	413 060F ORG 613H
370 0206 HexAsc:	414 0613 MOV P2, #0F5H
371 0206 ANL A, #0Fh	415 0616 RETI
372 0208 cjne A, #0Ah,HA1	
373 020B HA1: jc HAVal09	
374 020D ADD A, #7	
375 020F HAVal09: ADD A, '0'	
376 0211 ret	
377 0212 ; đưa mã lỗi về máy chủ	
378 0212 ErrPr:	
379 0212 mov A, #:'	

PHỤ LỤC E

Bảng các lệnh gộp

Chương	Tên	Ký hiệu	Công dụng	Thông số	Trang
(1)	(2)	(3)	(4)	(5)	(6)
2	Đọc ô nhớ	@docnho	Đọc ô nhớ	Không	42,43
2	Đọc cổng	@doccong	Đọc cổng vào A	Cổng	44
2	Ghi ra cổng	@ghicong	Ghi A ra cổng	Cổng, (d)ữ liệu	44
2	Cộng hai ô nhớ	@congkho	Cộng hai ô nhớ	Địa chỉ đích, nguồn	45
2	Dịch nhiều lần	@MULTIPLE_SHIFT	Dịch trái nhiều lần	Nhân, số lần	47
3	Nhảy nhỏ hơn hay bằng	@JLE	Nhảy nhỏ hơn hay bằng	Value	64
3	Nhảy ngoài dải	@JOR	Nhảy nếu ở ngoài dải	Value lớn, Value nhỏ	64
3	Nhảy trong dải	@JIR	Nhảy trong dải	Value lớn, Value nhỏ	64
3	Gọi nếu nhỏ hơn	@GLT	Gọi chương trình con có điều kiện	Value	66
3	Gọi nếu nhỏ hơn	@GOILT	Gọi chương trình con có điều kiện	Value	66
3	Tạo trễ	@TREPPS	Trễ phút, giây	M phút, N giây	69
3	Báo chuông	@BAOCH	Báo chuông	M phút, N giây	70
3	Phát xung	@FXVDK	Phát xung vuông	N giây	71
3	Phát số	@phatso	Phát chuỗi số	Số đầu, cuối, bước	72
3	Phát chữ	@phatchu	Phát một chuỗi chữ	Chuỗi chữ	72
4	Đọc thanh ghi	@docthg	Đọc thanh ghi SFR	Thanh ghi	87
4	Ghi dữ liệu vào thanh ghi	@ghithg	Ghi dữ liệu vào thanh ghi	Dữ liệu, thanh ghi	87
4	Ghi dữ liệu vào M	@GHIM	Ghi dữ liệu vào bộ nhớ M	Dữ liệu, địa chỉ nhớ	88
4	Ghi dữ liệu vào M ngoài	@GHIX	Ghi dữ liệu vào bộ nhớ M ngoài	Địa chỉ cao Địa chỉ thấp Dữ liệu	90
4	Đọc dữ liệu từ M ngoài	@DOCX	Đọc dữ liệu từ bộ nhớ M ngoài	Địa chỉ cao Địa chỉ thấp	90
4	Đếm	@DEM	Đếm thời gian cho đồng hồ	Địa chỉ nhớ Giá trị so sánh nhấn	91

(1)	(2)	(3)	(4)	(5)	(6)
4	Bảng biến đổi mã	@BDOIM--A7	Biến đổi mã BCD thành mã 7 đoạn	Địa chỉ vùng mã 7 đoạn mới	89
4	Bảng biến đổi mã	@BDMA7	Bảng biến đổi mã BCD thành mã 7 đoạn	Địa chỉ cao Địa chỉ thấp Địa chỉ mã mới	89
4	Giải mã một ô nhớ	@gma	Giải mã cho dữ liệu của một ô nhớ	Ô nhớ chứa số đếm Ô nhớ chứa số mã	92
4	Tạo bảng	@bg	Tạo một bảng chữ	Địa chỉ bảng Dữ liệu của bảng	93
5	Cộng 8 bit	@cong8	Cộng hai số 8 bit	Số thứ nhất Số thứ hai Ô nhớ ghi dữ liệu Ô nhớ ghi kết quả	98
5	Trừ 8 bit	@tru8	Trừ hai số 8 bit	Số thứ nhất Số thứ hai Ô nhớ ghi kết quả	99
5	Nhân 8 bit	@nhan8	Nhân hai số 8 bit	Số thứ nhất Số thứ hai Ô nhớ ghi dữ liệu Ô nhớ ghi kết quả	100
5	Nhân 8 bit kiểu cộng dồn	@nhancong8	Nhân hai số 8 bit kiểu cộng dồn	Số thứ nhất Số thứ hai Ô nhớ ghi dữ liệu Ô nhớ ghi kết quả	101
5	Chia 8 bit	@chia8	Chia hai số 8 bit	Số thứ nhất Số thứ hai Ô nhớ ghi kết quả	103
5	Chia 8 bit kiểu trừ	@chia tru	Chia hai số 8 bit kiểu trừ dần	Số thứ nhất Số thứ hai Ô nhớ ghi dữ liệu Ô nhớ ghi kết quả	103
5	Cộng 8 bit số BCD nén	@cong8d	Cộng hai số BCD nén	Số thứ nhất Số thứ hai Ô nhớ ghi dữ liệu Ô nhớ ghi kết quả	105
5	Trừ 8 bit số BCD nén	@trud8	Trừ hai số BCD nén	Số thứ nhất Số thứ hai Ô nhớ ghi dữ liệu Ô nhớ ghi kết quả	107

(1)	(2)	(3)	(4)	(5)	(6)
5	Cộng hai số có dấu 8 bit	@cgdaud8	Cộng hai số có dấu 8 bit	Số thứ nhất Số thứ hai Ô nhớ ghi dữ liệu Ô nhớ ghi kết quả	110
5	Nhân hai số có dấu 8 bit	@nhda8	Nhân hai số có dấu 8 bit	Số thứ nhất Số thứ hai Ô nhớ ghi dữ liệu Ô nhớ ghi kết quả	111
6	Biến đổi BCDHEX	@BCDHEX	Biến đổi mã BCD-HEX	Ô nhớ ghi dữ liệu Số thứ nhất, hai	124
7	Ghi dữ liệu ra cổng	@WritePs	Ghi dữ liệu ra cổng Pi	Dữ liệu Cổng Pi	134
7	Ghi A ra cổng Pi	@WriteP	Ghi A ra cổng Pi	Cổng Pi	135
7	Đọc bảng	@docbg	Đọc bảng dữ liệu ra cổng Pi	Không	136
8	Định thời timer0, 1	@timer	Định thời gian trễ	Chế độ, timer, TH, TL	145
8	Đếm	@dem	Đếm xung ngoài	Chế độ, timer, TH, TL	147
8	Định thời timer2	@timer2	Định thời gian trễ	TH2, TL2, RCAP2H, RCAP2L	150
9	Truyền byte tin	@truyen01	Truyền nối tiếp một byte tin	Chế độ, dữ liệu	168
9	Nhận byte tin	@nhan01	Nhận nối tiếp một byte tin	Chế độ	168
9	Tốc độ baud 1	@baud1	Phát tốc độ nối tiếp	TH1, TL1, timer, Chế độ	169

TÀI LIỆU THAM KHẢO

1. Ngô Diên Tập, 1998. *Vi điều khiển trong đo lường và điều khiển tự động*. NXB Khoa học và Kỹ thuật, Hà Nội.
2. Lê Văn Doanh, Phạm Khắc Chương. 1998, *Vi điều khiển*. NXB Khoa học và Kỹ thuật, Hà Nội.
3. Nguyễn Tăng Cường, Phan Quốc Thắng. 2004, *Cấu trúc và lập trình cho Vi điều khiển*. NXB Khoa học và Kỹ thuật, Hà Nội.
4. Nguyễn Đình Phú, (2002). *Cấu trúc và lập trình cho Vi điều khiển*. Tài liệu Đại học tại chức, Sư phạm kỹ thuật Tp. Hồ Chí Minh.
5. Tống Văn ON, 2001. *Họ vi điều khiển 8051*. NXB Lao động - Xã hội, Hà Nội.
6. Nguyễn Mạnh Giang, 1997. *Kỹ thuật ghép nối máy vi tính*. NXB Giáo dục, Hà Nội.
7. Nguyễn Mạnh Giang, 2001. *Lập trình Assembly cho máy vi tính PC-IBM*. NXB Giáo dục, Hà Nội.
8. Nguyễn Ngọc Lâm và tập thể tác giả, tháng 10-1999. *Thiết bị thực tập về vi xử lý*. Phân viện nghiên cứu điện tử tin học - tự động hóa Tp. Hồ Chí Minh.
9. Nguyễn Phúc và các đồng sự khác, 1999. *Nghiên cứu ứng dụng linh kiện lập trình, linh kiện chất lượng cao vào chế tạo một vài thiết bị hạt nhân điển hình*. Đề tài nghiên cứu cấp bộ 1997, 1998, 1999. Viện Khoa học và Kỹ thuật hạt nhân Hà Nội.
10. Myke Predko, 1999. *Programming and Customizing the 8051Microcontroller*. Mc Graw-Hill, New York.
11. Dusan Ponikvar, 2001. *Microprocessor*. University of Ljubjana.
12. *PINNACLE 52 IDE*, <http://www.vaultbbs.com/pinnacle>.

MỤC LỤC

	<i>Trang</i>
Lời nói đầu	3
Chương 1. Cấu trúc và hoạt động của vi điều khiển 8051/8052	
1.1. Cấu trúc của VDK 8051/8052	5
1.2. Hoạt động của VDK 8051/8052	9
1.3. Hệ lệnh và các phương pháp địa chỉ của VDK 8051/8052	13
1.4. Chương trình mô phỏng (SIMULATOR) của VDK 8051/8052	14
1.5. Các Kit thực hành VDK	18
1.6. Các VDK thuộc họ MCS-51	18
Chương 2. Lập trình hợp ngữ cho VDK họ MCS-51 với PINNACLE	
2.1. Tổng quát về lập trình hợp ngữ cho VDK	21
2.2. Soạn thảo một chương trình hợp ngữ cho VDK 8051/8052	26
2.3. Hợp dịch và liên kết chương trình nguồn với chương trình ASM51.EXE	35
2.4. Hợp dịch và biến đổi chương trình với Pinnacle	37
2.5. Lệnh gộp, chương trình con, thư viện chương trình và lập trình với lệnh gộp	40
2.6. Chạy chương trình trên kit vi điều khiển	51
Chương 3. Nhóm lệnh điều khiển chương trình	
3.1. Các lệnh rẽ nhánh chương trình không điều kiện	54
3.2. Lệnh gọi (call) và trở về (Ret, Reti) từ chương trình con	57
3.3. Các lệnh rẽ nhánh chương trình theo điều kiện của bit	57
3.4. Lệnh gộp điều khiển chương trình có điều kiện kép	62
3.5. Lệnh gộp cho vòng lặp	65
3.6. Các loại cấu trúc cơ bản của chương trình	70
Chương 4. Bộ nhớ trong, các phương pháp địa chỉ và nhóm lệnh dịch chuyển	
4.1. Bộ nhớ chương trình (PROM) bên trong của VDK 8051/8052	74
4.2. Bộ nhớ RAM bên trong (IRAM) và các thanh ghi của VDK	76
4.3. Các phương pháp địa chỉ của toán hạng	80
4.4. Nhóm lệnh dịch chuyển dữ liệu	83
4.5. Lệnh gộp và lập trình cho bộ nhớ, thanh ghi	87
Chương 5. ALU và các lệnh xử lý số học	
5.1. Cấu trúc ALU và các thanh ghi liên quan của VDK	96
5.2. Nhóm lệnh số học cho các số 8 bit không dấu	97
5.3. Các phép toán số học với số BCD	104
5.4. Lập trình các phép toán số học với số có dấu	107
5.5. Lập trình cho các phép toán số học với số có dấu phẩy động	112
Chương 6. Nhóm lệnh logic và phép chuyển đổi mã	
6.1. Nhóm lệnh xử lý logic	118
6.2. Các phép tính nhân và chia nhị phân với các lệnh logic	121
6.3. Biến đổi mã ASCII-HEX	121
6.4. Biến đổi mã BCD	123

Chương 7. Cổng song song, nhóm lệnh xử lý bit và điều khiển số	
7.1. Cấu trúc và hoạt động của các cổng song song	128
7.2. Các thanh ghi và ô nhớ xử lý theo bit	130
7.3. Nhóm lệnh xử lý theo bit (xử lý theo địa số BOOL)	131
7.4. Lập trình điều khiển đèn hiển thị	133
Chương 8. Bộ định thời/bộ đếm (Timer/Counter)	
8.1. Timer/Counter0,1 của 8051	139
8.2. Timer2 của VDK 8052	147
8.3. Ứng dụng Timer/Counter trong định thời gian trễ	149
8.4. Ứng dụng Timer/Counter trong phát xung	153
8.5. Ứng dụng trong đếm thời gian	154
8.6. Ứng dụng trong đo thời gian	156
8.7. Ứng dụng trong đếm xung	158
Chương 9. Cổng vào/ra nối tiếp	
9.1. Cấu trúc và hoạt động của cổng nối tiếp	162
9.2. Phát tốc độ baud	167
9.3. Trao đổi tin nối tiếp ở chế độ 1	170
9.4. Trao đổi tin nối tiếp trong hệ nhiều VDK ở chế độ 2,3	176
9.5. Trao đổi tin nối tiếp giữa VDK và máy vi tính PC	178
Chương 10. Điều khiển ngắt	
10.1. Tổng quát về ngắt của VDK 8051/8052	190
10.2. Chương trình có ngắt	194
10.3. Ngắt do các bộ định thời/đếm	197
10.4. Ngắt do trao đổi tin nối tiếp	210
10.5. Ngắt do tín hiệu ngoài	214
PHỤ LỤC A BẢNG MÃ ASCII	225
PHỤ LỤC B CÁC THANH GHI SFR	226
PHỤ LỤC C HỆ LỆNH CỦA VDK 8051/8052	231
PHỤ LỤC D CHƯƠNG TRÌNH RAM.LST	245
PHỤ LỤC E BẢNG CÁC LỆNH GỘP	250
TÀI LIỆU THAM KHẢO	253

Chịu trách nhiệm xuất bản:

Chủ tịch HĐQT kiêm Tổng Giám đốc NGÔ TRẦN ÁI
Phó Tổng Giám đốc kiêm Tổng biên tập NGUYỄN QUÝ THAO

Tổ chức bản thảo và chịu trách nhiệm nội dung:

Chủ tịch HĐQT kiêm Giám đốc CTCP sách ĐH-DN TRẦN NHẬT TÂN

Biên tập nội dung và sửa bản in:

DƯƠNG VĂN BẰNG

Trình bày bìa:

HOÀNG MẠNH DỨA

Chế bản:

NGUYỄN LÊ BẢO NGỌC

CẤU TRÚC – LẬP TRÌNH – GHÉP NỐI VÀ ỨNG DỤNG VI ĐIỀU KHIỂN – TẬP MỘT

Mã số : 7B677M7 – DAI

In 1.500 bản, khổ 19 x 27 cm, tại Nhà in Đại học Quốc Gia Hà Nội.

Giấy phép xuất bản số : 192 – 2007/CXB/2 – 411/GD.

In xong và nộp lưu chiểu tháng 5 năm 2007.



CÔNG TY CỔ PHẦN SÁCH ĐẠI HỌC – DẠY NGHỀ
HEVOBCO
25 HÀN THUYỀN – HÀ NỘI
Website : www.hevobco.com.vn

**TÌM ĐỌC SÁCH THAM KHẢO
CỦA NHÀ XUẤT BẢN GIÁO DỤC**

Bộ sách về ngành tin học

- | | |
|---|----------------------------|
| 1. Lập trình bằng ngôn ngữ Assembly cho máy tính PC - IBM | TS. Nguyễn Mạnh Giang |
| 2. Kỹ thuật ghép nối máy vi tính tập 1 và tập 2 | TS. Nguyễn Mạnh Giang |
| 3. Cấu trúc - lập trình - ghép nối và ứng dụng vi điều khiển. Tập 1. Cấu trúc - lập trình VDK 8051/8052 | TS. Nguyễn Mạnh Giang |
| 4. Cấu trúc máy vi tính | PGS. TS. Trần Quang Vinh |
| 5. Cơ sở đồ họa máy vi tính | TS. Phan Hữu Phúc |
| 6. AutoCAD cho tự động hoá thiết kế | Nguyễn Văn Hiến |
| 7. Ngôn ngữ lập trình C ** | PGS. TS. Nguyễn Việt Hương |

Bạn đọc có thể mua tại các Công ty Sách – Thiết bị trường học ở địa phương hoặc các Cửa hàng của Nhà xuất bản Giáo dục :

Tại Hà Nội : 25 Hàn Thuyên ; 187B Giảng Võ ; 23 Tràng Tiền ; 232 Tây Sơn.

Tại Đà Nẵng : 15 Nguyễn Chí Thanh ; 62 Nguyễn Chí Thanh.

Tại Thành phố Hồ Chí Minh : 240 Trần Bình Trọng – Quận 5 ;

104 Mai Thị Lựu – Quận 1 ;

Cửa hàng 451B, 453 Hai Bà Trưng – Quận 3.

Tại Thành phố Cần Thơ : Số 5/5, đường 30/4.



Giá: 36.000 đ