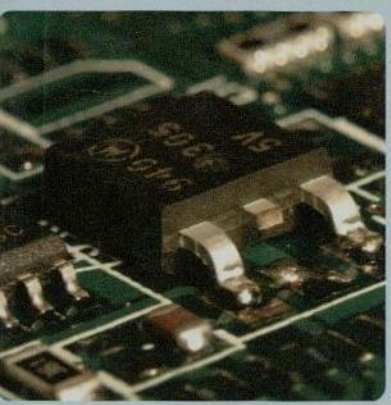
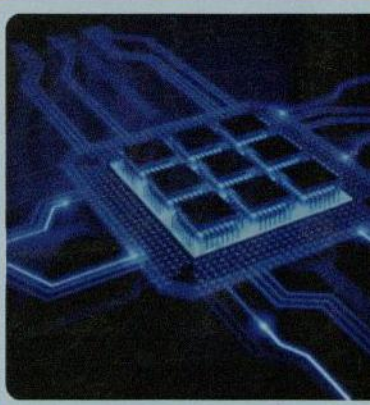
BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH

ISO 9001:2000

NGUYỄN ĐÌNH PHÚ - TRƯƠNG NGỌC ANH





GIÁO TRÌNH

VI XỬ LÝ

NHÀ XUẤT BẢN
ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT
THÀNH PHỐ HỒ CHÍ MINH

****

NGUYỄN ĐÌNH PHÚ
TRƯƠNG NGỌC ANH

GIÁO TRÌNH
VI XỬ LÝ

NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA
THÀNH PHỐ HỒ CHÍ MINH

LỜI NÓI ĐẦU

Bộ vi xử lý ngày càng phát triển hoàn thiện và được sử dụng hầu hết trong các hệ thống điều khiển trong công nghiệp cũng như trong các thiết bị dân dụng. Chính vai trò, chức năng của vi xử lý đã đem lại nhiều ưu điểm, nhiều tính năng đặc biệt cho các hệ thống điều khiển.

Các nhà nghiên cứu không ngừng nghiên cứu các hệ thống điều khiển và sử dụng vi xử lý để thay thế, và cũng chính vì thế đã thúc đẩy lĩnh vực vi xử lý phát triển ngày càng hoàn hảo thích nghi với yêu cầu điều khiển. Để đơn giản bớt sự phức tạp của phần cứng khi dùng vi xử lý, các nhà nghiên cứu đã tích hợp hệ vi xử lý thành một IC gọi là vi điều khiển.

Nội dung giáo trình này trình bày các khái niệm cơ bản của vi xử lý, sau đó đi vào nghiên cứu các kiến thức cơ bản của vi điều khiển. Do có nhiều họ vi điều khiển khác nhau mức độ tích hợp từ đơn giản đến phức tạp nên trong giáo trình này trình bày họ vi điều khiển tương đối đơn giản là AT89S52 của hãng ATMEL.

Giáo trình biên soạn chia thành 10 chương:

Chương 1: Giới thiệu về lịch sử phát triển của các thế hệ vi xử lý.

Chương 2: Trình bày đặc tính, cấu trúc, chức năng các port của vi điều khiển.

Chương 3: Trình bày về tổ chức bộ nhớ tích hợp bên trong vi điều khiển.

Chương 4: Trình bày về tập lệnh hợp ngữ.

Chương 5: Trình bày về ngôn ngữ lập trình C của vi điều khiển.

Chương 6: Trình bày cấu trúc các port và ứng dụng port.

Chương 7: Trình bày cấu trúc hoạt động của timer/counter.

Chương 8: Trình bày cấu trúc hoạt động chuyển đổi ADC, LM35 và các ứng dụng.

Chương 9: Trình bày cấu trúc hoạt động ngắt của vi điều khiển.

Chương 10: Trình bày cấu trúc hoạt động truyền dữ liệu của vi điều khiển.

Trong từng chương sau khi trình bày các kiến thức cơ bản và viết các ứng dụng cơ bản để người đọc dễ tiếp cận, có câu hỏi ôn tập, câu hỏi trắc nghiệm và bài tập để củng cố kiến thức, để có nhiều chương trình ứng dụng và bài tập bạn đọc có thể đọc thêm tài liệu thực hành trong đó có rất nhiều bài thực hành mẫu và bài tập đi kèm với bộ thí nghiệm do chúng tôi thiết kế.

Trong quá trình biên soạn không thể tránh được các sai sót nên rất mong các bạn đọc đóng góp xây dựng và xin hãy gởi về tác giả theo địa chỉ phu_nd@yahoo.com.

Tác giả xin cảm ơn các bạn bè đồng nghiệp đã đóng góp nhiều ý kiến, xin cảm ơn người thân trong gia đình cho phép tác giả có nhiều thời gian thực hiện biên soạn giáo trình này.

Tác giả

MỤC LỤC

LỜI NÓI ĐẦU 3
MỤC LỤC 5

CHƯƠNG 1. VI XỬ LÝ 13
I. CÁC KHÁI NIỆM 15
1. Vi xử lý – hệ thống vi xử lý 15
2. Khả năng ứng dụng vi xử lý 15
3. Lịch sử phát triển của các hệ vi xử lý 15
4. Chức năng của vi xử lý 17
5. Chức năng chương trình, bộ nhớ và ngoại vi 18
6. Máy vi tính (microcomputer) 18
7. Chiều dài từ dữ liệu 20
8. Khả năng truy xuất bộ nhớ 21
9. Tốc độ làm việc của vi xử lý 22
10. Các thanh ghi của vi xử lý 22
II. KHẢO SÁT VI XỬ LÝ 8 BIT 23
1. Sơ đồ khối 23
2. Khối ALU 24
3. Các thanh ghi 25
4. Chức năng các thanh ghi 26
III. LỆNH CỦA VI XỬ LÝ 31
1. Tập lệnh của vi xử lý 31
2. Từ gợi nhớ (mnemonics) 32
3. Các nhóm lệnh cơ bản của vi xử lý 33
4. Các kiểu truy xuất địa chỉ của một vi xử lý 34
IV. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP 34

1. Câu hỏi ôn tập 34
2. Câu hỏi mở rộng 35
3. Câu hỏi trắc nghiệm 35
4. Bài tập 37

CHƯƠNG 2. VI ĐIỀU KHIỂN 8 BIT – ĐẶC TÍNH, CẤU TRÚC, CHỨC NĂNG CÁC PORT 39

I. GIỚI THIỆU 40
II. KHẢO SÁT VI ĐIỀU KHIỂN ATMEL 41
1. Cấu hình của vi điều khiển ATMEL AT89S52 41
2. Sơ đồ cấu trúc của vi điều khiển AT89S52 41
3. Khảo sát sơ đồ chân vi điều khiển AT89S52 43
III. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP 48
1. Câu hỏi ôn tập 48
2. Câu hỏi mở rộng 48
3. Câu hỏi trắc nghiệm 48
4. Bài tập 50

CHƯƠNG 3. VI ĐIỀU KHIỂN 8 BIT – TỔ CHỨC BỘ NHỚ THANH GHI 51

I. GIỚI THIỆU 52
II. KIẾN TRÚC BỘ NHỚ 52
III. TỔ CHỨC BỘ NHỚ CỦA VI ĐIỀU KHIỂN ATMEL AT89S52 53
1. Tổ chức bộ nhớ 53
2. Khảo sát bộ nhớ ram 54
3. Khảo sát các thanh ghi có chức năng đặc biệt 57
IV. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP 64
1. Câu hỏi ôn tập 64

2. Câu hỏi mở rộng 64
3. Câu hỏi trắc nghiệm 65
4. Bài tập 66

CHƯƠNG 4. VI ĐIỀU KHIỂN 8 BIT – LỆNH HỢP NGỮ 67
I. GIỚI THIỆU 68
II. LỆNH HỢP NGỮ CỦA VI ĐIỀU KHIỂN MCS-52 69
1. Giới thiệu 69
2. Các kiểu định địa chỉ của vi điều khiển MCS-52 69
3. Khảo sát tập lệnh vi điều khiển MCS-52 72
III. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP 97
1. Câu hỏi ôn tập 97
2. Câu hỏi mở rộng 97
3. Câu hỏi trắc nghiệm 97
4. Bài tập 98

CHƯƠNG 5. VI ĐIỀU KHIỂN 8 BIT – NGÔN NGỮ LẬP TRÌNH C 99
I. GIỚI THIỆU 100
II. CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ C 100
1. Các kiểu dữ liệu của biến 100
2. Các toán tử 101
3. Các lệnh C cơ bản 106
III. TRÌNH BIÊN DỊCH C51 110
1. Phần mở rộng của trình biên dịch C51 110
2. Khai báo biến và hằng số 112
3. Các bit chức năng đặc biệt 112
4. Định nghĩa các biến 113
5. Con trỏ dữ liệu 113

6. Khai báo mảng 113
7. Khai báo chương trình con phục vụ ngắt 113
8. Cấu trúc chương trình C 114
9. Các thành phần của chương trình C 115
10. File thư viện cho họ AT89X52 116
IV. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP 123
1. Câu hỏi ôn tập 123
2. Câu hỏi mở rộng 124
3. Câu hỏi trắc nghiệm 124
4. Bài tập 126

CHƯƠNG 6. VI ĐIỀU KHIỂN 8 BIT – PORT XUẤT NHẬP 127
I. GIỚI THIỆU 128
II. CHỨC NĂNG CÁC PORT CỦA VI ĐIỀU KHIỂN 128
III. PORT CỦA VI ĐIỀU KHIỂN ATMEL AT89S52 128
1. Định cấu hình cho port 128
2. Lập trình truy xuất port dùng ngôn ngữ Assembly 129
3. Lập trình truy xuất port dùng ngôn ngữ Keil-C 129
IV. CÁC ỨNG DỤNG PORT CỦA VI ĐIỀU KHIỂN AT89S52 130
1. Ứng dụng AT89S52 điều khiển led đơn 130
2. Ứng dụng AT89S52 điều khiển led 7 đoạn trực tiếp 136
3. Ứng dụng AT89S52 điều khiển led 7 đoạn quét 144
4. Giao tiếp AT89S52 với nút nhấn, bàn phím 150
V. GIAO TIẾP VI ĐIỀU KHIỂN AT89S52 VỚI LCD 173
1. Giới thiệu LCD 173
2. Sơ đồ chân của LCD 174
3. Sơ đồ mạch giao tiếp vi điều khiển với LCD 175
4. Các lệnh điều khiển LCD 175

5. Địa chỉ của từng kí tự trên LCD 178
6. Các chương trình hiển thị trên LCD 179
VI. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP..... 185
1. Câu hỏi ôn tập 185
2. Câu hỏi mở rộng..... 185
3. Câu hỏi trắc nghiệm 185
4. Bài tập 188

CHƯƠNG 7. VI ĐIỀU KHIỂN 8 BIT – TIMER-COUNTER..... 189
I. GIỚI THIỆU 190
II. TIMER/COUNTER CỦA VI ĐIỀU KHIỂN ATMEL AT89S52 190
1. Khảo sát timer T0, T1 của AT89S52 190
2. Khảo sát timer T2 của AT89S52..... 195
3. Các thanh ghi, các bit của timer trong ngôn ngữ Keil-C..... 200
III. ỨNG DỤNG TIMER/COUNTER CỦA VI ĐIỀU KHIỂN ATMEL AT89S52 200
1. Định thời dùng timer của AT89S52 200
2. Đếm xung ngoại dùng counter của AT89S52 203
IV. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP..... 221
1. Câu hỏi ôn tập 221
2. Câu hỏi mở rộng..... 222
3. Câu hỏi trắc nghiệm 222
4. Bài tập 224

CHƯƠNG 8. VI ĐIỀU KHIỂN 8 BIT – CHUYỂN ĐỔI TƯƠNG TỰ SANG SỐ..... 225
I. GIỚI THIỆU 226
II. VI ĐIỀU KHIỂN ATMEL AT89S52 GIAO TIẾP ADC 0809 226
1. Khảo sát vi mạch ADC 0809..... 226

2. Ứng dụng đo nhiệt độ dùng AT89S52 và ADC 0809 230
III. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP..... 244
1. Câu hỏi ôn tập 244
2. Câu hỏi mở rộng..... 244
3. Câu hỏi trắc nghiệm 244
4. Bài tập 246

CHƯƠNG 9. VI ĐIỀU KHIỂN 8 BIT – NGẮT 247
I. GIỚI THIỆU 248
II. TỔNG QUAN VỀ NGẮT..... 248
III. NGẮT CỦA VI ĐIỀU KHIỂN ATMEL AT89S52 249
1. Các nguồn ngắt của AT89S52..... 249
2. Các thanh ghi ngắt của AT89S52..... 250
3. Khai báo ngắt của AT89S52 trong lập trình Keil-C..... 255
4. Ứng dụng ngắt của AT89S52..... 255
IV. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP..... 277
1. Câu hỏi ôn tập 277
2. Câu hỏi mở rộng..... 277
3. Câu hỏi trắc nghiệm 277
4. Bài tập 279

CHƯƠNG 10. VI ĐIỀU KHIỂN 8 BIT – TRUYỀN DỮ LIỆU UART 281
I. GIỚI THIỆU 283
II. TỔNG QUAN VỀ CÁC KIỂU TRUYỀN DỮ LIỆU 283
III. TRUYỀN DỮ LIỆU NỐI TIẾP ĐỒNG BỘ VÀ KHÔNG ĐỒNG BỘ..... 283
IV. TRUYỀN DỮ LIỆU NỐI TIẾP CỦA AT89S52..... 284
1. Truyền dữ liệu không đồng bộ của AT89S52 284

2. Chức năng các thanh ghi truyền dữ liệu của at89s5 285
3. Các kiểu truyền dữ liệu của AT89S52 286
4. Ứng dụng truyền dữ liệu của AT89S52 291
V. TRUYỀN DỮ LIỆU NỐI TIẾP SPI CỦA AT89S8252 299
1. Truyền dữ liệu SPI của AT89S8252 299
2. Chức năng các thanh ghi truyền dữ liệu SPI của AT89S8252 300
3. Dạng sóng truyền dữ liệu SPI 302
4. Ứng dụng truyền dữ liệu SPI của AT89S8252 304
VI. TRUYỀN DỮ LIỆU NỐI TIẾP I2C 307
1. Giới thiệu 307
2. Tổng quan về truyền dữ liệu I2C 307
3. Quy trình truyền dữ liệu chuẩn I2C 308
4. Dạng sóng truyền dữ liệu chuẩn I2C 309
5. Khảo sát realtime DS13B07 310
6. Ứng dụng realtime 313
VII. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP 320
1. Câu hỏi ôn tập 320
2. Câu hỏi mở rộng 320
3. Câu hỏi trắc nghiệm 320
4. Bài tập 323

TÀI LIỆU THAM KHẢO 325

Chương 1
TỔNG QUAN VỀ VI XỬ LÝ

❖ CÁC KHÁI NIỆM

- VI XỬ LÝ – HỆ THỐNG VI XỬ LÝ
- KHẢ NĂNG ỨNG DỤNG VI XỬ LÝ
- LỊCH SỬ PHÁT TRIỂN CỦA CÁC HỆ VI XỬ LÝ
- CHỨC NĂNG CỦA VI XỬ LÝ
- CHỨC NĂNG CHƯƠNG TRÌNH, BỘ NHỚ VÀ NGOẠI VI
- MÁY VI TÍNH (MICROCOMPUTER)
- CHIỀU DÀI TỪ DỮ LIỆU
- KHẢ NĂNG TRUY XUẤT BỘ NHỚ
- TỐC ĐỘ LÀM VIỆC CỦA VI XỬ LÝ
- CÁC THANH GHI CỦA VI XỬ LÝ

❖ KHẢO SÁT VI XỬ LÝ 8 BIT

- SƠ ĐỒ KHỐI
- KHỐI ALU
- CÁC THANH GHI
- CHỨC NĂNG CÁC THANH GHI
  - ✓ *Thanh ghi Accumulator*
  - ✓ *Thanh ghi bộ đếm chương trình PC (Program counter)*
  - ✓ *Thanh ghi trạng thái (Status Register)*
  - ✓ *Thanh ghi con trỏ ngăn xếp (Stack Pointer Register)*
  - ✓ *Thanh ghi địa chỉ bộ nhớ (memory address Register)*
  - ✓ *Thanh ghi lệnh (instruction Register)*
  - ✓ *Thanh ghi chứa dữ liệu tạm thời (Temporary data Register)*
  - ✓ *Khối điều khiển logic (control logic) và khối giải mà lệnh (instruction decoder)*

- ✓ *Bus dữ liệu bên trong vi xử lý*

❖ **LỆNH CỦA VI XỬ LÝ**

- TẬP LỆNH CỦA VI XỬ LÝ
- TỪ GỢI NHỚ (MNEMONICS)
- CÁC NHÓM LỆNH CƠ BẢN CỦA VI XỬ LÝ
- CÁC KIỂU TRUY XUẤT ĐỊA CHỈ CỦA MỘT VI XỬ LÝ

❖ **CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP**

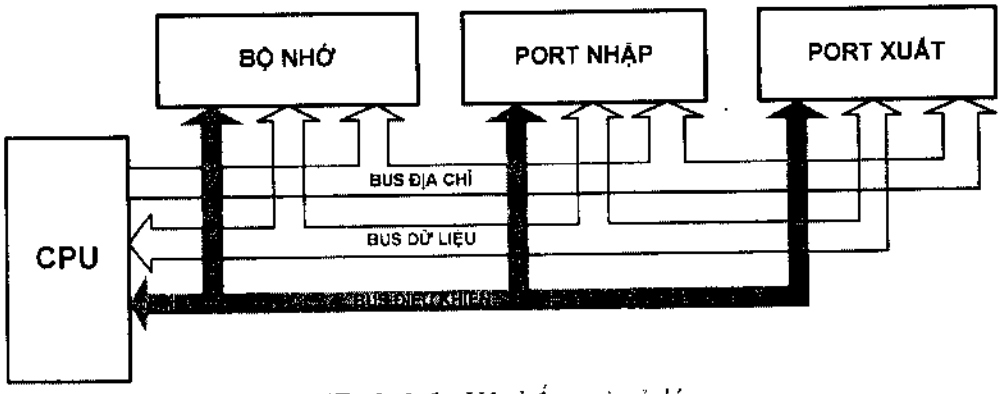
- CÂU HỎI ÔN TẬP
- CÂU HỎI MỞ RỘNG
- CÂU HỎI TRẮC NGHIỆM
- BÀI TẬP

I. CÁC KHÁI NIỆM

1. Vi xử lý – hệ thống vi xử lý

Vi xử lý là IC số chuyên về xử lý dữ liệu, tính toán dữ liệu và điều khiển theo chương trình.

Hệ thống vi xử lý gồm có vi xử lý, bộ nhớ và các thiết bị ngoại vi xuất nhập.



Hình 1-1: *Hệ thống vi xử lý.*

Chức năng của vi xử lý là xử lý dữ liệu và điều khiển theo chương trình, bộ nhớ có chức năng lưu trữ chương trình, các thiết bị ngoại vi có chức năng nhận dữ liệu từ bên ngoài đưa vào cho vi xử lý thực hiện xử lý dữ liệu, đồng thời nhận dữ liệu từ vi xử lý gởi ra điều khiển các đối tượng bên ngoài.

Tùy thuộc vào yêu cầu xử lý và điều khiển mà dung lượng bộ nhớ cũng như thiết bị ngoại vi nhiều hay ít.

2. Khả năng ứng dụng vi xử lý

Vi xử lý kết hợp với các thiết bị khác tạo ra các máy tính có khả năng tính toán rất lớn như máy vi tính và có thể tạo ra các sản phẩm khác các máy điện thoại, các tổng đài điện thoại, các hệ thống điều khiển tự động..., hình 1-2 cho thấy các ứng dụng có sử dụng vi xử lý.

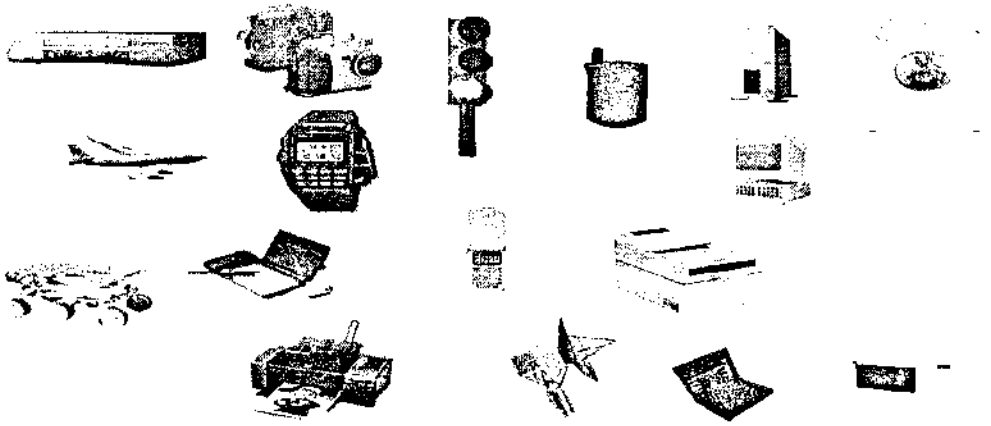
Có thể khẳng định vi xử lý ngày càng sử dụng hầu hết trong các thiết bị điều khiển cũng như các thiết dân dụng, trong các thiết bị giải trí, ...

3. Lịch sử phát triển của các hệ vi xử lý

Lịch sử phát triển của vi xử lý gắn liền với sự phát triển của các vi mạch điện tử vì vi xử lý là vi mạch điện tử chế tạo theo công nghệ LSI (large scale integrated) cho đến VLSI (very large scale integrated).

Với sự khám phá ra transistor và phát triển của công nghệ chế tạo vi mạch SSI, MSI, máy tính vẫn còn là một nhóm gồm nhiều IC kết hợp lại với nhau, cho đến thập niên 70, với sự phát triển của công nghệ LSI, cấu trúc máy tính được rút gọn bởi các nhà thiết kế và được chế tạo thành một IC duy nhất được gọi là vi xử lý (microprocessor).

Vi xử lý đầu tiên có khả năng xử lý 4 bit dữ liệu, các vi xử lý này có tốc độ xử lý rất chậm, các nhà thiết kế cải tiến thành vi xử lý 8bit, sau đó là vi xử lý 16 bit và 32 bit. Sự phát triển về dung lượng các bit của vi xử lý làm tăng thêm số lượng các lệnh điều khiển và các lệnh tính toán phức tạp.



***Hình 1-2:** Các thiết bị sử dụng vi xử lý.*

Lịch sử phát triển của vi xử lý gắn liền với hãng INTEL:

- Tháng 4 năm 1971, Intel cho ra vi xử lý 4 bit có mã số 4004 có thể truy xuất 4096 ô nhớ 4 bit và có 45 lệnh.
- Tháng 4 năm 1972, Intel cải tiến và cho ra vi xử lý 8 bit có mã số 8008 có thể truy xuất 16K ô nhớ 8 bit và có 48 lệnh.
- Tháng 4 năm 1974, Intel cải tiến vi xử lý 8008 thành vi xử lý 8080 có thể truy xuất 64Kbyte bộ nhớ và có nhiều lệnh hơn, chạy nhanh gấp 10 lần so với 8008.
- Tháng 4 năm 1976, Intel cải tiến vi xử lý 8080 thành vi xử lý 8085 có thêm mạch tạo xung clock được tích hợp bên trong, có nhiều ngắt trên chip phục vụ cho nhiều ứng dụng và tích hợp mạch điều khiển hệ thống trên chip.
- Tháng 6 năm 1978, Intel sản xuất vi xử lý 16 bit có mã số là 8086: có 20 đường địa chỉ cho phép truy xuất 1MB bộ nhớ và bus dữ liệu bên trong và bên ngoài đều là 16bit.

- Tháng 6 năm 1979, Intel sản xuất vi xử lý 16 bit có mã số là 8088 chủ yếu dựa vào vi xử lý 8086 nhưng khác với vi xử lý 8086 là bus dữ liệu bên ngoài chỉ có 8 bit nhưng bus dữ liệu bên trong vi xử lý là 16 bit, mục đích cải tiến này nhằm hạ giá thành hệ thống và trở thành vi xử lý trong máy tính IBM-PC/XT.
- Vào cuối năm 1981 và năm đầu 1982, Intel cho ra đời vi xử lý 80186 và phiên bản mở rộng của vi xử lý 8086 có hỗ trợ quản lý bộ nhớ theo phân đoạn và bảo vệ bộ nhớ, bus địa chỉ có 24 đường cho phép truy xuất 16Mbyte bộ nhớ.
- Tháng 2 năm 1982, Intel cho ra đời vi xử lý 80286 cũng là vi xử lý 16 bit và chủ yếu cũng phát triển từ vi xử lý 8086 có thêm nhiều chức năng như mạch định thời được tích hợp, mạch điều khiển DMA, mạch điều khiển ngắt và mạch chọn chip bộ nhớ được thiết kế riêng cho các ứng dụng nhúng với giá chip thấp.
- Tháng 10 năm 1985, Intel cho ra đời vi xử lý 80386 chính là vi xử lý 32bit, có quản lý bộ nhớ theo trang và phân đoạn bộ nhớ, bus dữ liệu bên trong và bên ngoài đều là 32 bit, tập thanh ghi được mở rộng.
- Tháng 4 năm 1989, Intel cho ra đời vi xử lý 80486, có cải thiện kiến trúc để tăng hiệu suất, cung cấp bộ nhớ cache trên board, đơn vị dấu chấm động trên board. Có thêm 6 lệnh so với vi xử lý 80386. Lệnh định thời được cải tiến để tăng hiệu suất.
- Tháng 3 năm 1993, Intel cho ra đời vi xử lý Pentium là vi xử lý 64 bit có đơn vị dấu chấm động hiệu suất cao. Lệnh định thời được cải tiến so với 80486.
- Tháng 3 năm 1995, Intel cho ra đời vi xử lý Pentium Pro có hai cấp cache có sẵn.
- Tháng 3 năm 1997, Intel cho ra đời vi xử lý Pentium II - Pentium Pro + MMX.
- Năm 1999, Intel cho ra đời vi xử lý Pentium III – IA64, mở rộng tạo luồng SIMD
- Năm 2000, Intel cho ra đời vi xử lý Pentium IV.
- Đến nay rất nhiều thế hệ vi xử lý mới ra đời.

4. Chức năng của vi xử lý

Công việc xử lý dữ liệu là chức năng chính của vi xử lý. Việc xử lý dữ liệu bao gồm tính toán và điều khiển dữ liệu. Việc tính toán được thực hiện

bởi các mạch điện logic được gọi là đơn vị xử logic số học (Arithmetic Logic Unit: ALU) có thể thực hiện các phép toán như Add, Subtract, And, Or, Compare, Increment, Decrement.

Để ALU có dữ liệu cho việc xử lý thì ngoài mạch điện ALU, vi xử lý còn có các mạch điện logic khác để điều khiển dữ liệu. Các mạch điện logic điều khiển dữ liệu sẽ di chuyển dữ liệu vào đúng vị trí để khối ALU xử lý dữ liệu. Sau khi thực hiện xong, khối điều khiển sẽ di chuyển dữ liệu đến bất cứ nơi nào mong muốn.

5. Chức năng chương trình, bộ nhớ và ngoại vi

Để xử lý dữ liệu, vi xử lý phải điều khiển các mạch logic, để vi xử lý điều khiển các mạch logic thì cần phải có chương trình. Chương trình là tập hợp các lệnh được lưu trữ trong bộ nhớ để vi xử lý thực hiện

Công việc thực hiện lệnh bao gồm các bước như sau: đón lệnh từ bộ nhớ, giải mã lệnh và sau cùng thì thực hiện các yêu cầu xử lý dữ liệu sau khi giải mã.

Do các lệnh lưu trữ trong bộ nhớ nên có thể thay đổi các lệnh nếu cần. Khi thay đổi các lệnh của vi xử lý là thay đổi cách thức xử lý dữ liệu, thay đổi chương trình điều khiển. Các lệnh lưu trữ trong bộ nhớ sẽ quyết định công việc mà vi xử lý sẽ làm.

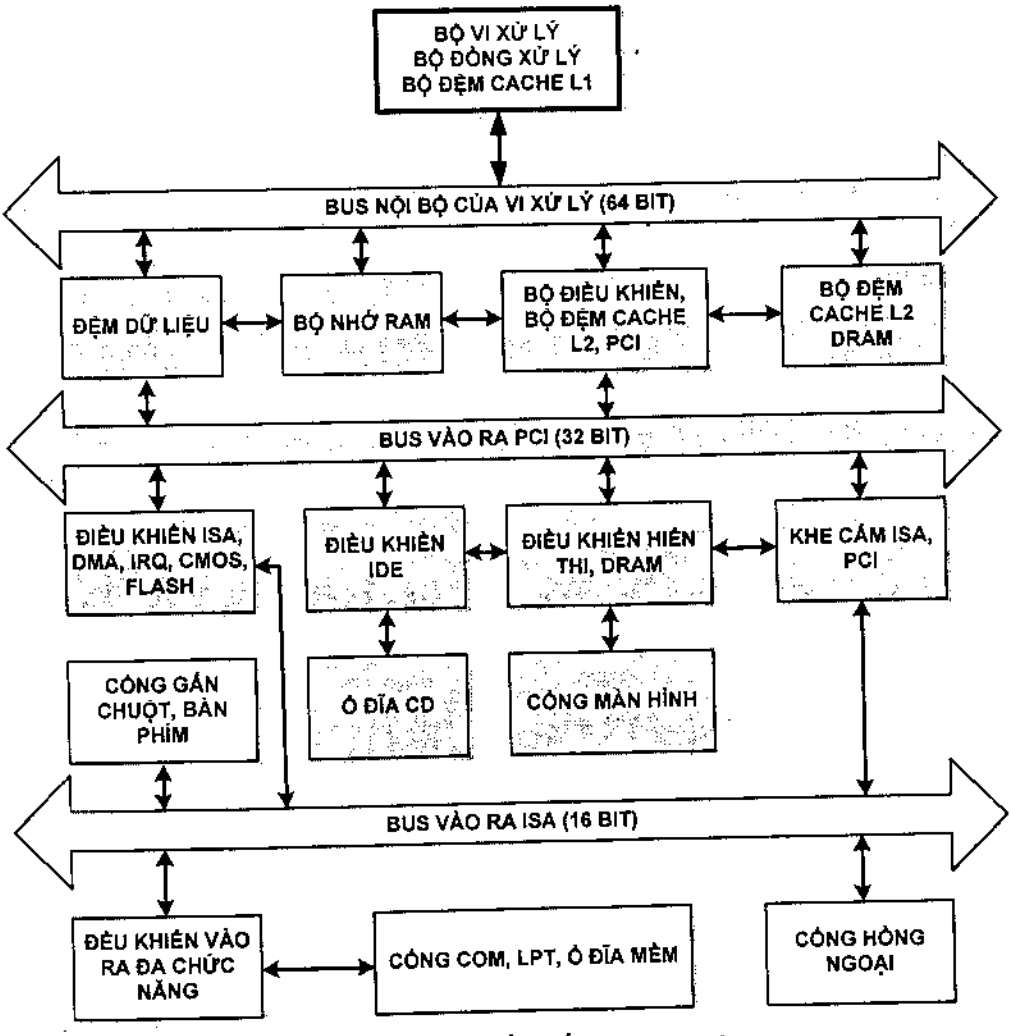
Nếu hệ thống điều khiển bằng mạch điện không có phần mềm thì khi thay đổi yêu cầu điều khiển phải thay đổi mạch điện hoặc thiết kế thêm hoặc thiết kế mới, còn hệ thống vi xử lý khi thay đổi yêu cầu trình tự điều khiển thì có thể chỉ thay đổi chương trình không cần phải thay đổi mạch điện phần cứng. Ví dụ hệ thống bán xăng dầu thì có thể thay đổi giá thành cho 1 lít xăng một cách dễ dàng hay chương trình điều khiển đèn giao thông có thể thay đổi thời gian điều khiển bằng cách lập trình lại.

Ngoài chức năng đón và thực hiện lệnh, các mạch logic điều khiển còn điều khiển các mạch điện giao tiếp bên ngoài kết nối với vi xử lý. Vi xử lý cần phải có sự trợ giúp của các mạch điện bên ngoài. Các mạch điện dùng để lưu trữ lệnh và dữ liệu gọi là ***bộ nhớ hay IC nhớ***, các mạch điện giao tiếp để di nhập dữ liệu từ bên ngoài vào bên trong vi xử lý và xuất dữ liệu từ bên trong vi xử lý ra bên ngoài được gọi là các ***thiết bị I/O*** hay các ***thiết bị ngoại vi***.

6. Máy vi tính (microcomputer)

Vi xử lý là một IC chuyên về xử lý dữ liệu và điều khiển còn máy vi tính là một hệ thống máy tính hoàn chỉnh được và có một bộ vi xử lý. Máy vi tính hoàn chỉnh bao gồm một vi xử lý, bộ nhớ và các cổng I/O.

Sơ đồ khối tổng quát của một hệ thống máy vi tính như hình 1-3:



Hình 1-3: Sơ đồ khối máy vi tính.

Máy vi tính tổ chức theo card bao gồm: card CPU, card bộ nhớ RAM, card điều khiển đĩa, card điều khiển màn hình, ngoài ra còn có màn hình video, bàn phím... xem hình 1-4 là bo mạch của máy vi tính.

Tất cả các card được kết nối với vi xử lý thông qua bus, bus bao gồm nhiều đường tín hiệu để phân biệt và xử lý các card khác nhau.

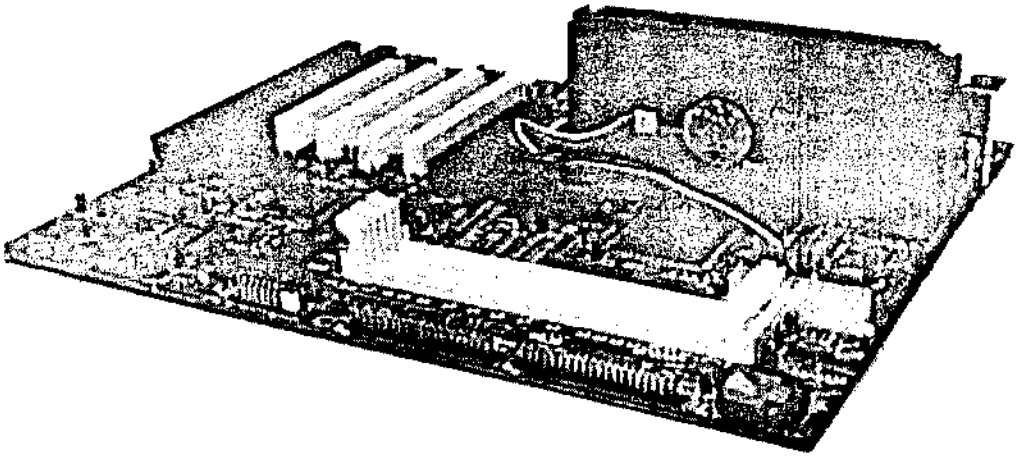
Trong card CPU có mạch tạo xung Clock để tạo ra tín hiệu clock cho vi xử lý. Card CPU còn có các IC giao tiếp để nâng cao khả năng giao tiếp của CPU.

Bộ nhớ ROM dùng để lưu trữ các lệnh của chương trình để cho phép nạp các chương trình từ đĩa mềm,..., card bộ nhớ RAM bao gồm các IC RAM để vi xử lý lưu trữ chương trình và dữ liệu khi xử lý. Trong card bộ nhớ có phần xuất nhập data nối tiếp UART (Universal Asynchronous Receiver – Transmitter), hai khối này có thể tách rời. UART có chức năng chuyển đổi dữ liệu song song thành nối tiếp để máy vi tính có thể giao tiếp với máy in, các modem, và các thiết bị điều khiển khác.

Để giao tiếp với màn hình video thì phải có card video, bên cạnh các IC giao tiếp với bus của vi xử lý còn có các IC điều khiển màn hình Video.

Màn hình Video dùng để hiển thị nội dung của một vùng nhớ đặc biệt trong bộ nhớ RAM do đó Card video có các IC RAM.

Khối nguồn cung cấp điện cho tất cả các hệ thống.



Hình 1-4: Main-board máy vi tính.

7. Chiều dài từ dữ liệu

Vi xử lý đầu tiên có chiều dài từ dữ liệu 4 bit, tiếp theo các vi xử lý 8 bit, 16 bit, 32 bit và 64 bit.

Mỗi vi xử lý có chiều dài từ dữ liệu khác nhau sẽ có một khả năng ứng dụng khác nhau, các vi xử lý có chiều dài từ dữ liệu lớn, tốc độ làm việc nhanh, khả năng truy xuất bộ nhớ lớn được dùng trong các công việc xử lý dữ liệu, điều khiển phức tạp, các vi xử lý có chiều dài từ dữ liệu nhỏ hơn, khả năng truy xuất bộ nhớ nhỏ hơn, tốc độ làm việc thấp hơn được sử dụng trong các công việc điều khiển và xử lý đơn giản, chính vì thế các vi xử lý này vẫn tồn tại.

Các vi xử lý 16 bit, 32 bit được sử dụng rất nhiều trong máy tính. Máy vi tính đầu tiên của IBM sử dụng vi xử lý 8088 vào năm 1981. Cấu trúc bên trong của vi xử lý 8088 có thể xử lý các từ dữ liệu 16 bit, nhưng bus dữ liệu giao tiếp bên ngoài chỉ có 8 bit. Do cấu trúc bên trong 16 bit nên các máy tính PC sử dụng bộ vi xử lý 8088 có thể tương thích với các máy tính mới sử dụng các vi xử lý 16 bit: 286, hoặc các vi xử lý 32 bit: 386, 486 và bộ vi xử lý Pentium.

Hầu hết các ứng dụng được điều khiển bởi máy tính tốt hơn nhiều so với vi xử lý và tùy theo yêu cầu điều khiển mà chọn điều khiển bằng máy tính hay điều khiển bằng vi xử lý. Các lĩnh vực điều khiển bằng vi xử lý như: công nghiệp, khoa học, y học,... Một lĩnh vực điều khiển phức tạp là robot khi đó các bộ vi xử lý 16 bit và 32 bit là thích hợp. Tùy theo yêu cầu độ phức tạp mà chọn bộ vi xử lý thích hợp.

Vi xử lý 32 bit là sự phát triển của vi xử lý 16 bit và ứng dụng đầu tiên của các vi xử lý 32 bit là các máy tính 32 bit. Các vi xử lý 32 bit có khả năng làm việc nhanh hơn vì mỗi lần lấy dữ liệu từ bộ nhớ vi xử lý có thể lấy một lần 4 byte, trong khi đó các vi xử lý 8 bit thì phải làm 4 lần, với vi xử lý 16 bit phải thực hiện 2 lần. Vậy nếu so với vi xử lý 8 bit thì vi xử lý 32 bit có tốc độ tăng gấp 4, với vi xử lý 16 bit thì tốc độ vi xử lý 32 bit tăng gấp đôi. Tăng tốc độ làm việc của vi xử lý là mục tiêu hàng đầu của các nhà chế tạo vi xử lý.

Vi xử lý có từ 4 bit đến 64 bit chia làm hai lĩnh vực ứng dụng: các vi xử lý mạnh, nhiều bit thì dùng trong các hệ thống máy tính để tính toán và xử lý các khối lượng dữ liệu lớn, phức tạp, các vi xử lý 8 bit, 16 bit hiện nay được sử dụng trong các hệ thống điều khiển công nghiệp vì khối lượng dữ liệu không lớn.

Khái niệm Bus dữ liệu: là đường truyền dữ liệu hai chiều để chuyển dữ liệu giữa vi xử lý và các thành phần khác của hệ thống như bộ nhớ, IC ngoại vi. Vi xử lý 8 bit thì bus dữ liệu sẽ là 8 bit, vi xử lý 16 bit thì bus dữ liệu giao tiếp cũng là 16 bit ngoại trừ vi xử lý 8088.

Vi xử lý 8 bit thì bus dữ liệu có 8 đường $D_0 \div D_7$, vi xử lý 16 bit thì bus dữ liệu có 16 đường $D_0 \div D_{15}$, vi xử lý 32 bit thì bus dữ liệu có 32 đường $D_0 \div D_{31}$, vi xử lý 64 bit thì bus dữ liệu có 64 đường $D_0 \div D_{63}$.

8. Khả năng truy xuất bộ nhớ

Dung lượng bộ nhớ mà vi xử lý có thể truy xuất là một phần trong cấu trúc của vi xử lý. Các vi xử lý đầu tiên bị giới hạn về khả năng truy xuất bộ nhớ: vi xử lý 4004 có 12 đường địa chỉ nên có thể truy xuất được $2^{12} = 4096$

ô nhớ, vi xử lý 8 bit có 16 đường địa chỉ nên có thể truy xuất được 2^{16} = 65536 ô nhớ, vi xử lý 16 bit có 20 đường địa chỉ nên có thể truy xuất 2^{20} = 1,024,000 ô nhớ, vi xử lý 32 bit như 386 hay 68020 có thể truy xuất 4G ô nhớ. ***Vi xử lý có khả năng truy xuất bộ nhớ càng lớn nên có thể xử lý các chương trình lớn***. Tùy theo ứng dụng cụ thể mà chọn một vi xử lý thích hợp.

➢ ***Khái niệm Bus địa chỉ*** là tất cả các đường địa chỉ của vi xử lý dùng để xác định địa chỉ của một ô nhớ hay một thiết bị ngoại vi trước khi thực hiện việc truy xuất dữ liệu.

Vi xử lý 16 bit địa thì bus địa chỉ có 16 đường $A_0 \div A_{15}$, vi xử lý 16 bit địa thì bus địa chỉ có 20 đường $A_0 \div A_{19}$, vi xử lý 32 bit địa thì bus địa chỉ có 32 đường $A_0 \div A_{31}$.

➢ ***Khái niệm Bus điều khiển*** là tất cả các đường mà vi xử lý dùng để điều khiển các đối tượng khác trong hệ thống như điều khiển đọc bộ nhớ, điều khiển ghi bộ nhớ, điều khiển đọc IO, điều khiển ghi IO.

9. Tốc độ làm việc của vi xử lý

Tần số xung clock cung cấp cho vi xử lý làm việc quyết định đến tốc độ làm việc của vi xử lý, vi xử lý có tốc độ làm việc càng cao thì khả năng xử lý lệnh càng nhanh. Tần số xung clock làm việc của các vi xử lý được cho bởi các nhà chế tạo.

10. Các thanh ghi của vi xử lý

Các thanh ghi là một phần quan trọng trong cấu trúc của vi xử lý.

Các thanh ghi bên trong của vi xử lý dùng để xử lý dữ liệu, có nhiều loại thanh ghi khác nhau cho các chức năng khác nhau trong vi xử lý, số lượng các thanh ghi đóng một vai trò rất quan trọng đối với vi xử lý và người lập trình.

Các vi xử lý khác nhau sẽ có số lượng và chức năng của các thanh cũng khác nhau.

Nếu vi xử lý có số lượng thanh ghi nhiều thì người lập trình có thể viết các chương trình điều khiển vi xử lý đơn giản hơn, làm tăng tốc độ xử lý chương trình. Nếu vi xử lý có số lượng thanh ghi ít thì chương trình sẽ phức tạp hơn, tốc độ xử lý chương trình chậm hơn.

Để hiểu rõ các thanh ghi bên trong của một vi xử lý cần phải khảo sát một vi xử lý cụ thể. Vậy số lượng các thanh ghi bên trong vi xử lý cũng ảnh hưởng đến tốc độ và khả năng xử lý chương trình.

II. KHẢO SÁT VI XỬ LÝ 8 BIT

Phần này giới thiệu cấu trúc bên trong của vi xử lý 8 bit, chức năng các thành phần cơ bản được xây dựng bên trong vi xử lý như khối ALU, khối thanh ghi, khối giải mã lệnh, khối xử lý, các bus địa chỉ, dữ liệu trong và ngoài, thanh ghi trạng thái...

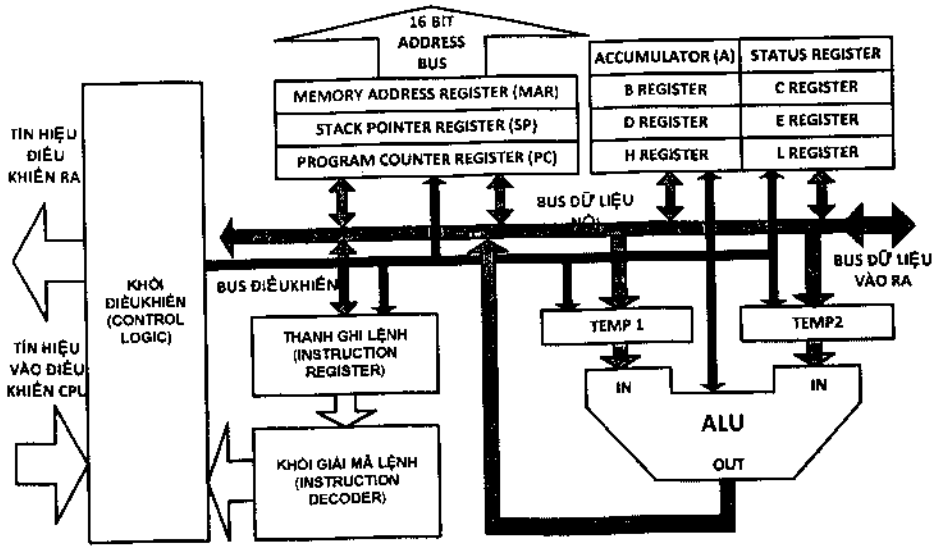
Trình tự hoạt động của các khối thông qua quá trình xử lý lệnh, giới thiệu về mã lệnh nhị phân, lệnh gợi nhớ, tập lệnh assembly và trình biên dịch Assembler.

Sau khi kết thúc phần này, người đọc sẽ biết được sơ đồ khối của vi xử lý 8 bit tổng quát, chức năng các khối bên trong, biết mã lệnh nhị phân, lệnh gợi nhớ, quá trình thực hiện lệnh của vi xử lý, các kiến thức cơ bản này sẽ được sử dụng ở các chương theo sau.

1. Sơ đồ khối

Cấu trúc của tất cả các vi xử lý đều có các khối cơ bản giống nhau như ALU, các thanh ghi, khối điều khiển là các mạch logic. Để nắm rõ nguyên lý làm việc của vi xử lý cần phải khảo sát nguyên lý kết hợp các khối với nhau để xử lý một chương trình.

Với mỗi vi xử lý đều có một sơ đồ cấu trúc bên trong và được cho trong các sổ tay của nhà chế tạo. Sơ đồ cấu trúc ở dạng khối rất tiện lợi và dễ trình bày nguyên lý hoạt động của vi xử lý. Hình 1-5 trình bày sơ đồ khối của vi xử lý 8 bit:



Hình 1-5: *Sơ đồ cấu trúc bên trong của vi xử lý.*

Trong sơ đồ khối của vi xử lý bao gồm các khối chính như sau: khối ALU, các thanh ghi và khối control logic. Ngoài ra sơ đồ khối còn trình bày các đường truyền tải tín hiệu từ nơi này đến nơi khác bên trong và bên ngoài hệ thống.

2. Khối ALU

ALU là khối quan trọng nhất của vi xử lý, khối ALU chứa các mạch điện tử logic chuyên về xử lý dữ liệu. Khối ALU có hai ngõ vào có tên là "IN" – là các ngõ vào dữ liệu cho ALU xử lý và một ngõ ra có tên là "OUT" – là ngõ ra kết quả dữ liệu sau khi ALU xử lý xong.

Dữ liệu trước khi vào ALU được chứa ở thanh ghi tạm thời (Temporarily Register) có tên là **TEMP1** và **TEMP2**. Bus dữ liệu bên trong vi xử lý được kết nối với hai ngõ vào "IN" của ALU thông qua hai thanh ghi tạm thời. Việc kết nối này cho phép ALU có thể lấy bất kỳ dữ liệu nào trên bus dữ liệu bên trong vi xử lý.

Thường thì ALU luôn lấy dữ liệu từ một thanh ghi đặc biệt có tên là Accumulator (A). Ngõ ra OUT của ALU cho phép ALU có thể gởi kết dữ liệu sau khi xử lý xong lên bus dữ liệu bên trong vi xử lý, do đó thiết bị nào kết nối với bus bên trong đều có thể nhận dữ liệu này. Thường thì ALU gởi dữ liệu sau khi xử lý xong tới thanh ghi A.

Ví dụ khi ALU cộng hai dữ liệu thì một trong hai dữ liệu được chứa trong thanh ghi A, sau khi phép cộng được thực hiện bởi khối ALU thì kết quả sẽ gởi trở lại thanh ghi A và lưu trữ ở thanh ghi này.

ALU xử lý một dữ liệu hay hai dữ liệu tùy thuộc vào lệnh hay yêu cầu điều khiển, ví dụ khi cộng hai dữ liệu thì ALU sẽ xử lý hai dữ liệu và dùng hai ngõ vào "IN" để nhập dữ liệu, khi tăng một dữ liệu nào đó lên một đơn vị hay lấy bù một dữ liệu, khi đó ALU chỉ xử lý một dữ liệu và chỉ cần một ngõ vào "IN".

Khối ALU có thể thực hiện các phép toán xử lý như sau:

Add	Complement	OR
Exclusive OR		
Subtract	Shift right	Increment
AND	Shift left	Decrement

Tóm Tắt: Chức năng chính của khối ALU là xử lý dữ liệu nhưng không lưu trữ dữ liệu. Để hiểu rõ thêm chức năng đặc biệt của ALU cần phải khảo sát một vi xử lý cụ thể.

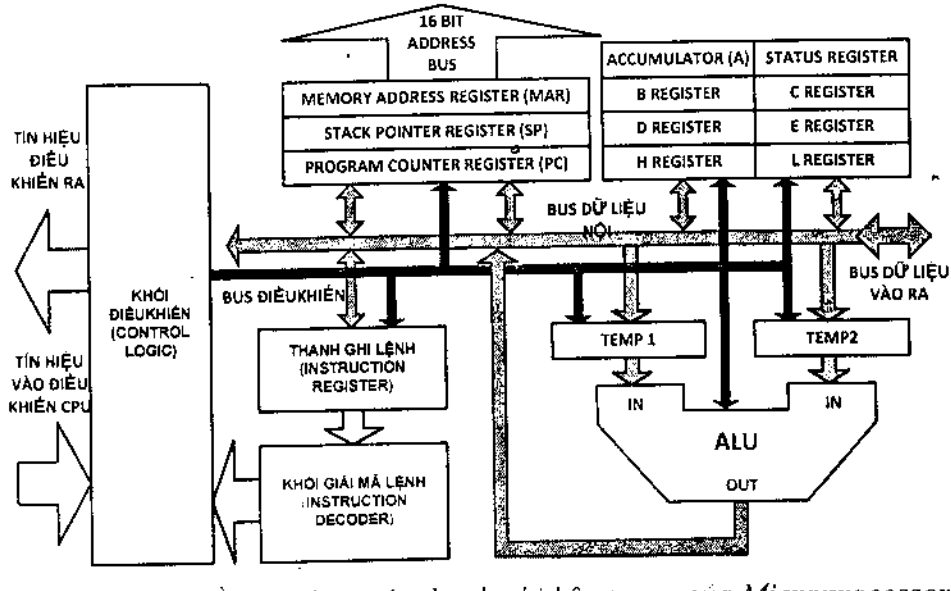
3. Các thanh ghi

Các thanh ghi bên trong có chức năng lưu trữ tạm thời các dữ liệu khi xử lý. Trong số các thanh ghi có một vài ***thanh ghi đặc biệt*** thực hiện các lệnh đặc biệt hay các chức năng đặc biệt, các thanh ghi còn lại gọi là các ***thanh ghi thông dụng***. Với sơ đồ khối minh họa ở trên, các thanh ghi thông dụng có tên Reg B, Reg C, Reg D, Reg E.

Các thanh ghi thông dụng rất hữu dụng cho người lập trình dùng để lưu trữ dữ liệu phục vụ cho công việc xử lý dữ liệu và điều khiển, khi viết chương trình chúng ta luôn sử dụng các thanh ghi này. Số lượng các thanh ghi thông dụng thay đổi tùy thuộc vào từng vi xử lý.

Số lượng và cách sử dụng các thanh ghi thông dụng tùy thuộc vào cấu trúc của từng vi xử lý, nhưng chúng có một vài điểm cơ bản giống nhau. Càng nhiều thanh ghi thông dụng thì vấn đề lập trình càng trở nên đơn giản.

Các thanh ghi cơ bản luôn có trong một vi xử lý là thanh ghi A (***Accumulator Register***), thanh ghi bộ đếm chương trình PC (***Program Counter register***), thanh ghi con trỏ ngăn xếp SP (***Stack pointer register***), thanh ghi trạng thái F (***Status register –Flag register***), các thanh ghi thông dụng, thanh ghi lệnh IR (***Instruction register***), thanh ghi địa chỉ AR (***Address Register***).



Hình 1-6: *Sơ đồ minh họa các thanh ghi bên trong của Microprocessor được tô đậm.*

4. Chức năng các thanh ghi

➢ ***Thanh ghi Accumulator***

Thanh ghi A là một thanh ghi quan trọng của vi xử lý có chức năng lưu trữ dữ liệu khi tính toán. Hầu hết các phép toán số học và các phép toán logic đều xảy ra giữa ALU và Accumulator.

Ví dụ khi thực hiện lệnh cộng dữ liệu X với dữ liệu Y, thì một dữ liệu phải chứa trong thanh ghi A giả sử là dữ liệu X, sau đó sẽ thực hiện lệnh cộng dữ liệu X (chứa trong A) với dữ liệu Y (có thể chứa trong ô nhớ hoặc trong một thanh ghi thông dụng), kết quả của lệnh cộng là dữ liệu Z sẽ được đặt trong thanh ghi A thay thế cho dữ liệu X trước đó.

Chú ý: Kết quả sau khi thực hiện ALU thường gởi vào thanh ghi A làm cho dữ liệu trước đó chứa trong A sẽ mất.

Thanh ghi A còn gọi là thanh ghi tích lũy có chức năng khi thực hiện phép toán cộng sẽ cộng dồn các dữ liệu: ví dụ cộng các dữ liệu X+Y+Z+V+W thì ta đưa dữ liệu đầu tiên là X vào thanh ghi A rồi cộng với dữ liệu Y, kết quả sau khi cộng lưu vào thanh ghi A, tiến hành cộng với dữ liệu Z, kết quả lưu vào thanh ghi A, rồi cộng tiếp cho đến khi hết dữ liệu.

Một chức năng quan trọng khác của thanh ghi A là lưu dữ liệu khi gởi ra các thiết bị IO bên ngoài và nhận dữ liệu đọc về từ IO.

Thanh ghi A còn nhiều chức năng quan trọng khác sẽ được thấy rõ qua tập lệnh của một vi xử lý cụ thể, số bit của thanh ghi A chính là đơn vị đo của vi xử lý, vi xử lý 8 bit thì thanh ghi A có độ dài 8 bit, vi xử lý 16 bit thì thanh ghi A có độ dài 16 bit.

➢ ***Thanh ghi bộ đếm chương trình PC (Program counter)***

Thanh ghi PC là một thanh ghi có vai trò quan trọng nhất của vi xử lý.

Chương trình là một chuỗi các lệnh nối tiếp nhau lưu trong bộ nhớ, các lệnh này sẽ yêu cầu vi xử lý thực hiện chính xác các công việc để giải quyết một vấn đề.

Bộ nhớ có thể lưu trữ nhiều chương trình, mỗi chương trình sẽ chiếm một vùng nhớ, nếu muốn vi xử lý thực hiện chương trình nào thì địa chỉ bắt đầu của chương trình đó sẽ nạp vào thanh ghi PC.

Địa chỉ bắt đầu của chương trình chính là địa chỉ của lệnh thứ nhất. Vi xử lý sẽ tiến hành đón mã lệnh của lệnh thứ nhất về để xử lý, trong quá trình xử lý lệnh thứ nhất thì nội dung của thanh ghi PC sẽ tăng lên để trỏ đến lệnh thứ hai, sau khi lệnh thứ nhất thực hiện xong thì sẽ đón mã lệnh thứ hai, trong quá trình xử lý lệnh thứ hai thì nội dung của thanh ghi PC sẽ tăng lên

để trỏ đến lệnh thứ ba, ... cứ thế thực hiện cho đến lệnh cuối cùng của chương trình.

Chức năng của thanh ghi PC là quản lý địa chỉ của lệnh đang thực hiện và lệnh tiếp theo hay cũng có thể nói thanh ghi PC quản lý địa chỉ của bộ nhớ chương trình.

Thanh ghi PC trong vi xử lý có chiều dài từ dữ liệu lớn hơn chiều dài từ dữ liệu của vi xử lý. Ví dụ đối với các vi xử lý 8 bit thì thanh ghi PC thường có chiều dài là 16 bit nên có thể truy xuất vùng nhớ bắt đầu từ ô nhớ thứ 0 đến ô nhớ thứ 65535.

Địa chỉ của lệnh được gởi đến bộ nhớ thông qua bus địa chỉ. Sau đó bộ nhớ sẽ đặt nội dung của ô nhớ lên bus dữ liệu, nội dung này chính là mã lệnh, quá trình này gọi là đón lệnh.

Một vài lệnh trong chương trình có thể nạp vào thanh ghi PC một giá trị mới, khi lệnh làm thay đổi thanh ghi PC sang giá trị mới được thực hiện thì lệnh kế có thể xảy ra ở một địa chỉ mới – đối với các lệnh nhảy hoặc lệnh gọi chương trình con.

➢ ***Thanh ghi trạng thái (Status Register)***

Thanh ghi trạng thái còn được gọi là thanh ghi cờ (Flag register) dùng để lưu trữ trạng thái kết quả của một số lệnh kiểm tra. Các trạng thái cho phép người lập trình thực hiện việc rẽ nhánh trong chương trình. Khi rẽ nhanh thì chương trình sẽ bắt đầu tại một vị trí mới. Khi rẽ nhánh có điều kiện thì chương trình chỉ rẽ nhánh khi thỏa điều kiện.

Các bit thường có trong một thanh ghi trạng thái được trình bày ở hình 1-7.

Z	N	C	AC	OV	P	-	-

Hình 1-7: Cấu trúc của một thanh ghi trạng thái.

Các lệnh xảy ra trong khối ALU thường ảnh hưởng đến thanh ghi trạng thái, ví dụ khi thực hiện một lệnh cộng hai dữ liệu 8 bit, nếu kết quả lớn hơn 11111111B thì bit carry sẽ mang giá trị là 1. Ngược lại nếu kết quả của phép cộng nhỏ hơn 11111111 B thì bit carry bằng 0. Ví dụ lệnh tăng hay giảm giá trị của một thanh ghi, nếu kết quả trong thanh ghi khác 0 thì bit Z luôn bằng 0, ngược lại nếu kết quả bằng 0 thì bit Z bằng 1.

Ý nghĩa của các bit trong thanh ghi trạng thái:

- Cờ tràn/mượn (Carry/borrow): khi thực hiện phép cộng nếu kết quả cộng tràn thì bit carry =1, ngược lại bit carry = 0. Khi thực

hiện phép toán trừ: nếu số bị trừ lớn hơn số trừ thì bit borrow = 0, ngược lại bit borrow =1. Bit carry hay bit borrow chỉ là 1 bit.

- Cờ Zero: khi kết quả của phép toán bằng 0 thì cờ Z bằng 1, khi kết quả khác 0 thì cờ Z=0.
- Cờ số âm (Negative): bit N = 1 khi bit MSB của thanh ghi có giá trị là 1, ngược lại N=0.
- Cờ tràn phụ (Auxiliary Carry): giống như bit Carry nhưng chỉ có tác dụng đối với phép cộng hay trừ 4 bit thấp của số BCD.
- Cờ tràn số có dấu (Overflow): cờ này bằng 1 khi cộng hai số có dấu bị tràn.
- Parity (cờ chẵn lẻ): nếu kiểm tra chẵn thì bit này có giá trị là 0 khi kết quả của phép toán là số chẵn, ngược lại là số lẻ thì bit P = 1, nếu kiểm tra lẻ thì ngược lại.

Số lượng các bit có trong thanh ghi trạng thái tùy thuộc vào từng vi xử lý. Trong một số vi xử lý có thể xóa hoặc đặt các bit của thanh ghi trạng thái.

➢ ***Thanh ghi con trỏ ngăn xếp (Stack Pointer Register)***

Thanh ghi con trỏ ngăn xếp là một thanh ghi quan trọng của vi xử lý, độ dài từ dữ liệu của thanh ghi SP bằng thanh ghi PC.

Chức năng của thanh ghi SP dùng để quản lý địa chỉ của bộ nhớ ngăn xếp, chức năng của bộ nhớ ngăn xếp là lưu dữ liệu trữ tạm thời.

Ví dụ khi muốn thực hiện cộng nội dung của hai thanh ghi B với C kết quả lưu vào thành ghi B, vi xử lý không cho phép cộng trực tiếp thanh ghi B với thanh ghi C mà phải thông qua thanh ghi A, trong khi đó A đang lưu một dữ liệu quan trọng khác vậy thì là m sao để thực hiện?

Để thực hiện phép cộng hai dữ liệu trên thì phải lưu tạm nội dung thanh ghi A vào bộ nhớ ngăn xếp, chuyển nội dung thanh ghi B sang thanh ghi A rồi cộng với thanh ghi C, chuyển kết quả cộng từ thanh ghi A sang thanh ghi B, cuối cùng thì lấy lại kết quả trong bộ nhớ ngăn xếp trả lại cho thanh ghi A. Dữ liệu trong thanh ghi A lưu tạm vào bộ nhớ ngăn xếp nên gọi là dữ liệu tạm thời.

Trong quá trình thực hiện chương trình, vi xử lý thường gọi chương trình con lưu ở một nơi khác trong bộ nhớ chương trình, trước khi thực hiện chương trình con thì địa chỉ của lệnh kế trong thanh ghi PC được lưu tạm vào bộ nhớ ngăn xếp, tiếp theo là nạp địa chỉ bắt đầu của chương trình con vào thanh ghi PC, chương trình con bắt đầu được thực hiện, cho đến khi gặp

lệnh trở về thì địa chỉ của lệnh kế trong bộ nhớ ngăn xếp sẽ trả lại cho thanh ghi PC và chương trình chính tiếp tục được thực hiện.

Thanh ghi SP phải trỏ đến một vùng nhớ RAM do người lập trình thiết lập, quá trình này gọi là khởi tạo con trỏ ngăn xếp. Nếu không khởi tạo, con trỏ ngăn xếp sẽ chỉ đến một ô nhớ ngẫu nhiên, khi đó dữ liệu cất vào ngăn xếp có thể ghi đè lên dữ liệu quan trọng khác làm chương trình xử lý sai hoặc thanh ghi SP trỏ đến vùng nhớ không phải là bộ nhớ RAM làm việc lưu dữ liệu không thực hiện được sẽ làm chương trình thực hiện không đúng. Tổ chức của ngăn xếp là vào sau ra trước **(LAST IN FIRST OUT: LIFO).**

➢ ***Thanh ghi địa chỉ bộ nhớ (memory address Register)***

Mỗi khi vi xử lý truy xuất bộ nhớ chương trình để đọc mã lệnh thì thanh ghi địa chỉ sẽ nhận địa chỉ từ thanh ghi PC và xuất địa chỉ ra bus địa chỉ để truy xuất bộ nhớ.

Mã lệnh được đón về lưu vào thanh ghi lệnh, tiến hành giải mã để cho biết lệnh bao nhiêu byte, lệnh xử lý dữ liệu lưu ở địa chỉ nào.

Khối điều khiển sẽ tiến hành thực hiện lệnh: thứ nhất là tăng thanh ghi PC tăng lên để chuẩn bị đón lệnh tiếp theo, thứ hai là nếu lệnh này có thêm byte thứ hai hoặc byte thứ hai byte thứ ba (nằm kế byte mã lệnh) thì khối điều khiển sẽ điều khiển tăng nội dung trong thanh ghi MAR lên 1 để đón byte thứ hai, nếu có byte thứ ba thì tiếp tục tăng lên 1 nữa để đón byte thứ ba3, sau khi có đầy đủ dữ liệu thì lệnh sẽ được xử lý và lưu kết quả, kết thúc quá trình thực hiện một lệnh rồi tiến hành thực hiện lệnh tiếp theo.

Vậy địa chỉ trong thanh ghi MAR có chức năng đón địa chỉ byte thứ hai hoặc byte thứ ba để xử lý lệnh.

Trong tất cả các vi xử lý, thanh ghi địa chỉ bộ nhớ có chiều dài bằng với thanh ghi PC.

➢ ***Thanh ghi lệnh (instruction Register)***

Thanh ghi lệnh dùng để chứa mã lệnh vi xử lý đang thực hiện.

Một chu kỳ lệnh bao gồm đón lệnh từ bộ nhớ, giải mã lệnh và thực hiện lệnh.

Đầu tiên là lệnh được đón từ bộ nhớ, sau đó PC trỏ đến lệnh kế trong bộ nhớ. Khi một lệnh được đón có nghĩa là mã lệnh trong ô nhớ đó được copy vào vi xử lý thông qua bus dữ liệu và lưu vào thanh ghi lệnh. Tiếp theo lệnh sẽ được giải mã: bộ giải mã lệnh đọc mã lệnh đang lưu trong thanh ghi

lệnh, bộ giải mã sẽ giải mã lệnh để báo cho khối điều khiển thực hiện chính xác công việc mà lệnh yêu cầu.

Chiều dài từ dữ liệu của thanh ghi lệnh tùy thuộc vào từng vi xử lý.

Thanh ghi lệnh do vi xử lý sử dụng người lập trình không sử dụng được thanh ghi này.

➢ ***Thanh ghi chứa dữ liệu tạm thời (Temporary data Register)***

Thanh ghi lưu trữ dữ liệu tạm thời dùng để ALU thực hiện các phép toán xử lý dữ liệu. Do ALU chỉ xử lý dữ liệu không có chức năng lưu trữ dữ liệu, bất kỳ dữ liệu nào đưa đến ngõ vào của ALU, lập tức sẽ xuất hiện ở ngõ ra.

Dữ liệu xuất hiện tại ngõ ra của ALU được quyết định bởi lệnh trong chương trình yêu cầu ALU thực hiện. ALU lấy dữ liệu từ bus dữ liệu bên trong vi xử lý, xử lý dữ liệu, sau đó đặt dữ liệu vừa xử lý xong trở lại thanh ghi Accumulator, do đó cần phải có thanh ghi lưu trữ dữ liệu tạm thời để ALU thực hiện. Người lập trình không được phép xử dụng các thanh ghi tạm thời. Số lượng các thanh ghi này tùy thuộc vào từng vi xử lý cụ thể.

➢ ***Khối điều khiển logic (control logic) và khối giải mà lệnh (instruction decoder)***

Chức năng của khối giải mã lệnh là nhận lệnh từ thanh ghi lệnh sau đó giải mã để gởi tín hiệu điều khiển đến cho khối điều khiển logic.

Chức năng của khối điều khiển logic (control logic) là nhận lệnh hay tín hiệu điều khiển từ bộ giải mã lệnh, sau đó sẽ thực hiện đúng các yêu cầu của lệnh.

Các tín hiệu điều khiển của khối điều khiển logic là các tín hiệu điều khiển bộ nhớ, điều khiển các thiết bị ngoại vi, các đường tín hiệu đọc-ghi,... và các tín hiệu điều khiển vi xử lý từ các thiết bị bên ngoài. Các đường tín hiệu này sẽ được trình bày cụ thể trong sơ đồ của từng vi xử lý.

Ngõ tín hiệu vào quan trọng nhất của khối điều khiển logic là tín hiệu clock cần thiết cho khối điều khiển logic hoạt động. Nếu không có tín hiệu clock thì vi xử lý không làm việc. Mạch tạo xung clock là các mạch dao động, tín hiệu được đưa đến ngõ vào clock của vi xử lý. Có nhiều vi xử lý tích hợp mạch tạo dao động ở bên trong, khi đó chỉ cần thêm tụ thạch anh ở bên ngoài.

➢ ***Bus dữ liệu bên trong vi xử lý***

Bus dữ liệu dùng để kết nối các thanh ghi bên trong và ALU với nhau, tất cả các dữ liệu di chuyển trong vi xử lý đều thông qua bus dữ liệu này.

Các thanh ghi bên trong có thể nhận dữ liệu từ bus hay có thể đặt dữ liệu lên bus nên bus dữ liệu này là bus dữ liệu hai chiều. Bus dữ liệu bên trong có thể kết nối ra bus bên ngoài khi vi xử lý cần truy xuất dữ liệu từ bộ nhớ bên ngoài hay các thiết bị IO. Bus dữ liệu bên ngoài cũng là bus dữ liệu hai chiều vì vi xử lý có thể nhận dữ liệu từ bên ngoài hay gởi dữ liệu ra.

III. LỆNH CỦA VI XỬ LÝ

1. Tập lệnh của vi xử lý

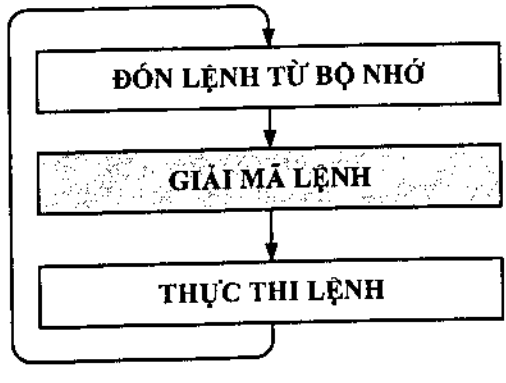
Lệnh của vi xử lý là một số nhị phân, khi vi xử lý đọc một lệnh thì từ dữ liệu nhị phân này sẽ yêu cầu vi xử lý làm một công việc đơn giản. Mỗi một từ dữ liệu tương đương với một công việc mà vi xử lý phải làm. Hầu hết các lệnh của vi xử lý là các lệnh chuyển dữ liệu và xử lý dữ liệu.

Tập lệnh của vi xử lý là tất cả các lệnh mà vi xử lý có thể hiểu và thực hiện được.

Độ dài của một lệnh bằng với độ dài từ dữ liệu của vi xử lý, đối với vi xử lý 8 bit thì độ dài của một lệnh là 8 bit, đối với vi xử lý 16 bit thì độ dài của một lệnh là 16 bit,...

Trong chu kỳ đón lệnh, mã lệnh sẽ được gởi đến thanh ghi lệnh, bộ giải mã lệnh, và bộ điều khiển logic của vi xử lý. Chức năng của các khối sẽ xác định lệnh này làm gì và sẽ gởi các tín hiệu điều khiển đến các mạch điện logic khác trong vi xử lý, các tín hiệu logic này sẽ thực hiện đúng chức năng mà lệnh yêu cầu.

Hình 1-8 minh họa chu kỳ thực hiện lệnh:



***Hình 1-8:** Chu kỳ thực hiện lệnh của vi xử lý.*

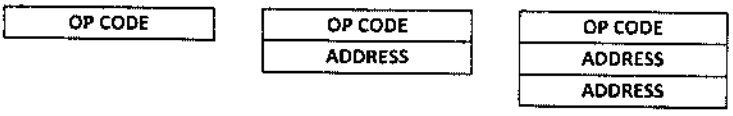
Một lệnh được thực hiện cần phải hội đủ hai thành phần:

➢ ***Thành phần thứ nhất*** sẽ yêu cầu vi xử lý thực hiện công việc gì. Ví dụ yêu cầu vi xử lý thực hiện một lệnh cộng: ADD, một lệnh dịch chuyển dữ liệu: MOV,... là những lệnh mà vi xử lý có thực hiện được.

➢ ***Thành phần thứ hai*** cho vi xử lý biết các thông tin địa chỉ tức là vị trí của các dữ liệu mà vi xử lý phải thực hiện. Ví dụ khi thực hiện một lệnh cộng nội dung hai thanh ghi A và B, hoặc nội dung thanh ghi A và dữ liệu chứa trong một ô nhớ. Thành phần thứ hai trong trường hợp này là các thanh ghi A và B, hoặc thanh ghi A và địa chỉ của ô nhớ.

Thành phần thứ nhất gọi là mã lệnh: op code (operation code) và thành phần thứ hai gọi là địa chỉ. Mã lệnh sẽ báo cho vi xử lý làm gì và địa chỉ sẽ cho vi xử lý biết vị trí của dữ liệu.

Sơ đồ hình 1-9 minh họa cho cấu trúc một lệnh.



Hình 1-9: *Cấu trúc của lệnh: 1 từ, lệnh 2 từ và lệnh 3 từ dữ liệu.*

Từ dữ liệu đầu tiên luôn là mã lệnh, các từ dữ liệu tiếp theo là địa chỉ. Đối với các lệnh chỉ có một từ dữ liệu thì địa chỉ đã được hiểu ngầm.

Do có nhiều cách chỉ cho vi xử lý biết địa chỉ của dữ liệu được gọi là các kiểu truy xuất bộ nhớ hay còn gọi là các kiểu định địa chỉ. Khi sử dụng một vi xử lý cần phải biết các kiểu truy xuất này.

2. Từ gợi nhớ (Mnemonics)

Lệnh của vi xử lý là các con số nhị phân. Đối với lệnh chỉ có một byte thì rất khó nhớ, nếu lệnh dài 2, 3, 4 hoặc nhiều hơn nữa thì không thể nào nhớ hết. Để giảm bớt sự phức tạp của số nhị phân có thể dùng số Hex để thay thế, khi đó các lệnh dễ viết và dễ đọc hơn nhiều nhưng cũng không thể nào giúp người sử dụng nhớ hết được và quan trọng nhất là khi viết chương trình cũng như lúc gỡ rối chương trình.

Bảng 1-1: *Mã lệnh nhị phân 3 lệnh của vi xử lý 8 bit 8085.*

Thứ tự	Mã lệnh nhị phân	Mã lệnh hex	Chức năng
1	1000 0111	87	(A) + (R) => (A)
2	1100 1101 Addr	CD addr	(SP) =PC PC =addr
3	0011 1100	3C	(R) + 1 => (R)

Để giải quyết vấn đề khó nhớ các mã lệnh nhị phân thì lệnh được viết thành các từ gợi nhớ rất gần với chức năng và ý nghĩa của các lệnh.

Các từ gợi nhớ của lệnh được rút gọn chỉ có khoảng 3 đến 4 ký tự. Ví dụ lệnh di chuyển dữ liệu có từ gợi nhớ là MOV, lệnh trừ là SUB,...

Bảng 1-2: *Lệnh gợi nhớ 3 lệnh của vi xử lý 8 bit 8085.*

Thứ tự	Lệnh gợi nhớ	Mã lệnh nhị phân	Mã lệnh hex	Chức năng
1	ADD R	1000 0111	87	(A) + (R) => (A)
2	CALL addr	1100 1101 addr	CD addr	(SP) =PC PC =addr
3	INR R	0011 1100	3C	(R) + 1 => (R)

Khi sử dụng lệnh gợi nhớ giúp người lập trình rất dễ nhớ tất cả các lệnh của vi xử lý, khi viết chương trình người lập trình dùng các từ gợi nhớ để viết chương trình, các từ gợi nhớ tạo thành một ngôn ngữ gọi là Assembly.

Khi viết chương trình bằng ngôn ngữ Assembly thì vi xử lý sẽ không hiểu – muốn vi xử lý hiểu và thực hiện chương trình thì phải sử dụng chương trình biên dịch Assembler để chuyển các lệnh viết dưới dạng ngôn ngữ Assembly thành các lệnh dạng số nhị phân tương ứng rồi nạp vào bộ nhớ thì vi xử lý mới thực hiện được.

Lệnh của vi xử lý còn gọi là mã máy, lệnh gợi nhớ gọi là hợp ngữ. Có thể lập trình cho vi xử lý bằng hợp ngữ nhưng khi chương trình yêu cầu phức tạp thì lập trình bằng hợp ngữ rất khó và dài, để giúp lập trình đơn giản thì có thể dùng ngôn ngữ lập trình cấp cao khá phổ biến là ngôn ngữ C.

Trong các chương theo sau thì chủ yếu sử dụng lập trình bằng ngôn ngữ C.

3. Các nhóm lệnh cơ bản của vi xử lý

Đối với hầu hết các vi xử lý tập lệnh được chia ra làm 9 nhóm lệnh cơ bản:

- Nhóm lệnh truyền dữ liệu: Data transfers.
- Nhóm lệnh trao đổi, truyền khối dữ liệu, lệnh tìm kiếm: Exchanges, Block transfers, Searches.
- Nhóm lệnh số học và logic: arithmetic and logic.
- Nhóm lệnh xoay và dịch: Rotates and shifts.
- Nhóm lệnh điều khiển CPU.

- Nhóm lệnh về bit: Bit set, bit reset, and bit test.
- Nhóm lệnh nhảy: Jumps.
- Nhóm lệnh gọi, trở về và nhóm lệnh bắt đầu lại: Calls, Return, and Restarts.
- Nhóm lệnh xuất nhập: Input and Output.

Các mã gợi nhớ và các mã nhị phân của tất cả các lệnh sẽ được cho trong các sổ tay của nhà chế tạo đối với từng vi xử lý cụ thể.

4. Các kiểu truy xuất địa chỉ của một vi xử lý

Như đã trình bày ở các phần trên, vi xử lý có thể truy xuất bộ nhớ bằng nhiều cách để lấy dữ liệu. Vi xử lý có nhiều cách truy xuất thì chương trình khi viết sẽ càng ngắn gọn rất có lợi cho người lập trình và làm giảm thời gian thực hiện chương trình.

Để biết vi xử lý có bao nhiêu cách truy xuất bộ nhớ cần phải khảo sát từng vi xử lý cụ thể. Các kiểu truy xuất được cho trong các sổ tay chế tạo.

Các kiểu truy xuất địa chỉ cơ bản của một vi xử lý (được gọi tắt là kiểu định địa chỉ):

- Kiểu định địa chỉ ngầm định (Implied Addressing Mode).
- Kiểu định địa chỉ tức thời (Immediate Addressing Mode).
- Kiểu định địa chỉ trực tiếp (Direct Addressing Mode).
- Kiểu định địa chỉ gián tiếp dùng thanh ghi (Register Indirect Addressing Mode).
- Kiểu định địa chỉ chỉ số (Indexed Addressing Mode).
- Kiểu định địa chỉ tương đối (Relative Addressing Mode).

IV. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP

1. Câu hỏi ôn tập

Câu số 1-1: Hãy nêu 5 ứng dụng của vi xử lý.

Câu số 1-2: Hãy vẽ sơ đồ khối hệ thống vi xử lý giao tiếp với bộ nhớ và thiết bị ngoại vi và chức năng của từng khối.

Câu số 1-3: Hãy trình bày các thế hệ phát triển vi xử lý của Intel từ 4 bit đến 32 bit và nêu tên vi xử lý cho mỗi thế hệ.

Câu số 1-4: Hãy trình các loại vi xử lý khác nhau trong thế hệ vi xử lý 8 bit và các cải tiến.

Câu số 1-5: Hãy cho biết chức năng của các thanh ghi PC, SP, A.

Câu số 1-6: Hãy cho biết chức năng của khối ALU và các bit trong thanh ghi trạng thái.

2. Câu hỏi mở rộng

Câu số 1-7: Hãy tìm hiểu vi xử lý 8085 và vi xử lý Z80.

Câu số 1-8: Hãy so sánh vi xử lý 8085 với vi xử lý Z80.

3. Câu hỏi trắc nghiệm

Câu 1-1: ALU không thực hiện chức năng nào:

(a) Add (b) Shift

(c) Complement (d) Lưu trữ dữ liệu.

Câu 1-2: ALU có hai ngõ vào, hai ngõ vào này được kết nối với:

(a) Program Counter (b) Bus dữ liệu bên trong

(c) Control logic (d) TG địa chỉ bộ nhớ.

Câu 1-3: Chức năng chính của khối ALU

(a) Lưu trữ dữ liệu (b) Giải mã lệnh

(c) Thực hiện các phép toán (d) Điều khiển.

Câu 1-4: Hầu hết các phép toán trong vi xử lý được thực hiện giữa ô nhớ hoặc thanh ghi với:

(a) Thanh ghi A (b) Thanh ghi PC

(c) Thanh ghi địa chỉ (d) Thanh ghi lệnh

Câu 1-5: Vi xử lý có thể truy xuất 2^{20} = 1.048.567 ô nhớ thì thanh ghi PC bao nhiêu bit:

(a) 4 (b) 8

(c) 16 (d) 20

Câu 1-6: Thanh ghi PC của vi xử lý là có chức năng lưu:

(a) Dữ liệu (b) Địa chỉ lệnh kế

(c) Mã lệnh (d) Trạng thái

Câu 1-7: Khi vi xử lý đang thực hiện lệnh thì thanh ghi PC trỏ đến:

(a) Lệnh vừa thực hiện
(b) Lệnh đang thực hiện
(c) Lệnh tiếp theo
(d) Ô nhớ tiếp theo

Câu 1-8: Lệnh mà vi xử lý hiểu và thực hiện là:

(a) Số nhị phân
(b) Mã BCD
(c) Lệnh gợi nhớ
(d) Lệnh cấp cao

Câu 1-9: Thanh ghi PC có chức năng quản lý địa chỉ của:

(a) Bộ nhớ dữ liệu
(b) Bộ nhớ chương trình
(c) Bộ nhớ ngăn xếp
(d) Tất cả các loại

Câu 1-10: Thanh ghi SP có chức năng quản lý địa chỉ của:

(a) Bộ nhớ dữ liệu
(b) Bộ nhớ chương trình
(c) Bộ nhớ ngăn xếp
(d) Tất cả các loại

Câu 1-11: Thanh ghi SP của vi xử lý 8 bit có số bit bằng:

(a) 8 bit
(b) 16 bit
(c) 32 bit
(d) 8 byte

Câu 1-12: Chức năng của bộ nhớ ngăn xếp là lưu:

(a) Mã lệnh
(b) Dữ liệu tạm thời
(c) Trạng thái của ALU
(d) Kết quả của ALU

Câu 1-13: Chức năng của bus địa chỉ của vi xử lý:

(a) Giao tiếp với các đường địa chỉ của bộ nhớ và ngoại vi
(b) Giao tiếp với các đường dữ liệu của bộ nhớ và ngoại vi
(c) Giao tiếp với các đường điều khiển của bộ nhớ và ngoại vi
(d) Tất cả đều đúng

Câu 1-14: Bus địa chỉ của vi xử lý là:

(a) Bus hai chiều
(b) Một chiều vào
(c) Một chiều ra
(d) Không xác định

Câu 1-15: Cờ tràn Carry C bằng 1 khi:

(a) Khi cộng có kết quả là số lẻ

(b) Khi cộng 2 số nhị phân có dấu

(c) Khi cộng 2 số nhị phân không dấu bị tràn

(d) Khi cộng có kết quả là số chẵn

Câu 1-16: Cờ zero Z bằng 1 khi:

(a) Khi kết quả bị tràn

(b) Khi kết quả bằng 0

(c) Khi kết quả khác không

(d) Khi kết quả bằng 1

Câu 1-17: Cơ sở thông tin để xác định vi xử lý bao nhiêu bit là căn cứ vào:

(a) Số đường địa chỉ

(b) Số đường dữ liệu

(c) Số đường điều khiển

(d) Dung lượng bộ nhớ

Câu 1-18: Lệnh mà vi xử lý hiểu và thực hiện là

(a) Lệnh gợi nhớ Assembly

(b) Lệnh ngôn ngữ cấp cao

(c) Lệnh của ngôn ngữ C

(d) Mã lệnh nhị phân

Câu 1-19: Lệnh Assembly có chức năng

(a) Giúp vi xử lý dễ hiểu

(b) Giúp con người dễ hiểu và dễ nhớ

(c) Làm vi xử lý thực hiện nhanh

(d) Làm vi xử lý thực hiện chậm

Câu 1-20: Trình biên dịch Aseember có chức năng

(a) Dịch mã lệnh nhị phân thành lệnh Assembly

(b) Dịch lệnh C thành lệnh Assembly

(c) Dịch lệnh Assembly thành lệnh nhị phân

(d) Làm vi xử lý thực hiện nhanh

Câu 1-21: Chức năng của thanh ghi IR là

(a) Lưu địa chỉ của mã lệnh

(b) Lưu mã lệnh nhị phân

(c) Lưu dữ liệu

(d) Lưu lệnh gợi nhớ

4. Bài tập

Chương 2
VI ĐIỀU KHIỂN AT89S52: ĐẶC TÍNH, CẤU TRÚC, CHỨC NĂNG CÁC PORT

- **GIỚI THIỆU**
- **KHẢO SÁT VI ĐIỀU KHIỂN ATMEL**
  - CẤU HÌNH CỦA VI ĐIỀU KHIỂN ATMEL AT89S52
  - SƠ ĐỒ CẤU TRÚC CỦA VI ĐIỀU KHIỂN AT89S52
  - KHẢO SÁT SƠ ĐỒ CHÂN VI ĐIỀU KHIỂN AT89S52
- **CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP**
  - CÂU HỎI ÔN TẬP
  - CÂU HỎI MỞ RỘNG
  - CÂU HỎI TRẮC NGHIỆM
  - BÀI TẬP

I. GIỚI THIỆU

Ở chương một đã giới thiệu về cấu trúc bên trong và chức năng của các khối cũng như trình tự hoạt động xử lý dữ liệu của vi xử lý.

Vi xử lý có rất nhiều loại bắt đầu từ 4 bit cho đến 64 bit, vi xử lý 4 bit hiện nay không còn nhưng vi xử lý 8 bit vẫn còn mặc dù đã có vi xử lý 64 bit.

Lý do sự tồn tại của vi xử lý 8 bit là phù hợp với một số yêu cầu điều khiển trong công nghiệp. Các vi xử lý 32 bit, 64 bit thường sử dụng cho các máy tính vì khối lượng dữ liệu của máy tính rất lớn nên cần các vi xử lý càng mạnh càng tốt.

Các hệ thống điều khiển trong công nghiệp sử dụng các vi xử lý 8 bit hay 16 bit như hệ thống điện của xe hơi, hệ thống điều hòa, hệ thống điều khiển các dây chuyền sản xuất,...

Khi sử dụng vi xử lý thì phải thiết kế một hệ thống gồm có: vi xử lý, có bộ nhớ, các ngoại vi.

Bộ nhớ dùng để lưu chương trình cho vi xử lý thực hiện và lưu dữ liệu cần xử lý, các ngoại vi dùng để xuất nhập dữ liệu từ bên ngoài vào xử lý và điều khiển trở lại. Các khối này liên kết với nhau tạo thành một hệ thống vi xử lý.

Yêu cầu điều khiển càng cao thì hệ thống càng phức tạp và nếu yêu cầu điều khiển đơn giản thì hệ thống vi xử lý cũng phải có đầy đủ các khối trên.

Để kết nối các khối trên tạo thành một hệ thống vi xử lý đòi hỏi người thiết kế phải rất hiểu biết về tất cả các thành phần vi xử lý, bộ nhớ, các thiết bị ngoại vi. Hệ thống tạo ra khá phức tạp, chiếm nhiều không gian, mạch in, và vấn đề chính là đòi hỏi người thiết kế hiểu thật rõ về hệ thống. Một lý do nữa là vi xử lý thường xử lý dữ liệu theo byte hoặc word trong khi đó các đối tượng điều khiển trong công nghiệp thường điều khiển theo bit.

Chính vì sự phức tạp nên các nhà chế tạo đã tích hợp bộ nhớ và một số các thiết bị ngoại vi cùng với vi xử lý tạo thành một IC gọi là vi điều khiển – Microcontroller.

Khi vi điều khiển ra đời đã mang lại sự tiện lợi là dễ dàng sử dụng trong điều khiển công nghiệp, việc sử dụng vi điều khiển không đòi hỏi người sử dụng phải hiểu biết một lượng kiến thức quá nhiều như người sử dụng vi xử lý.

Phần tiếp theo chúng ta sẽ khảo sát vi điều khiển để thấy rõ sự tiện lợi trong điều khiển.

Có rất nhiều hãng chế tạo được vi điều khiển, hãng sản xuất nổi tiếng là TI, Microchip, ATMEL,... tài liệu này trình bày vi điều khiển ATMEL.

II. KHẢO SÁT VI ĐIỀU KHIỂN ATMEL

Đến thời điểm hiện nay có rất nhiều loại vi điều khiển hãng Atmel, trong tài liệu sẽ giới thiệu về họ vi điều khiển MCS – 52 và cụ thể là khảo sát vi điều khiển AT89S52.

1. Cấu hình của vi điều khiển ATMEL AT89S52

Các vi điều khiển họ MCS-52 có các đặc điểm chung như sau:

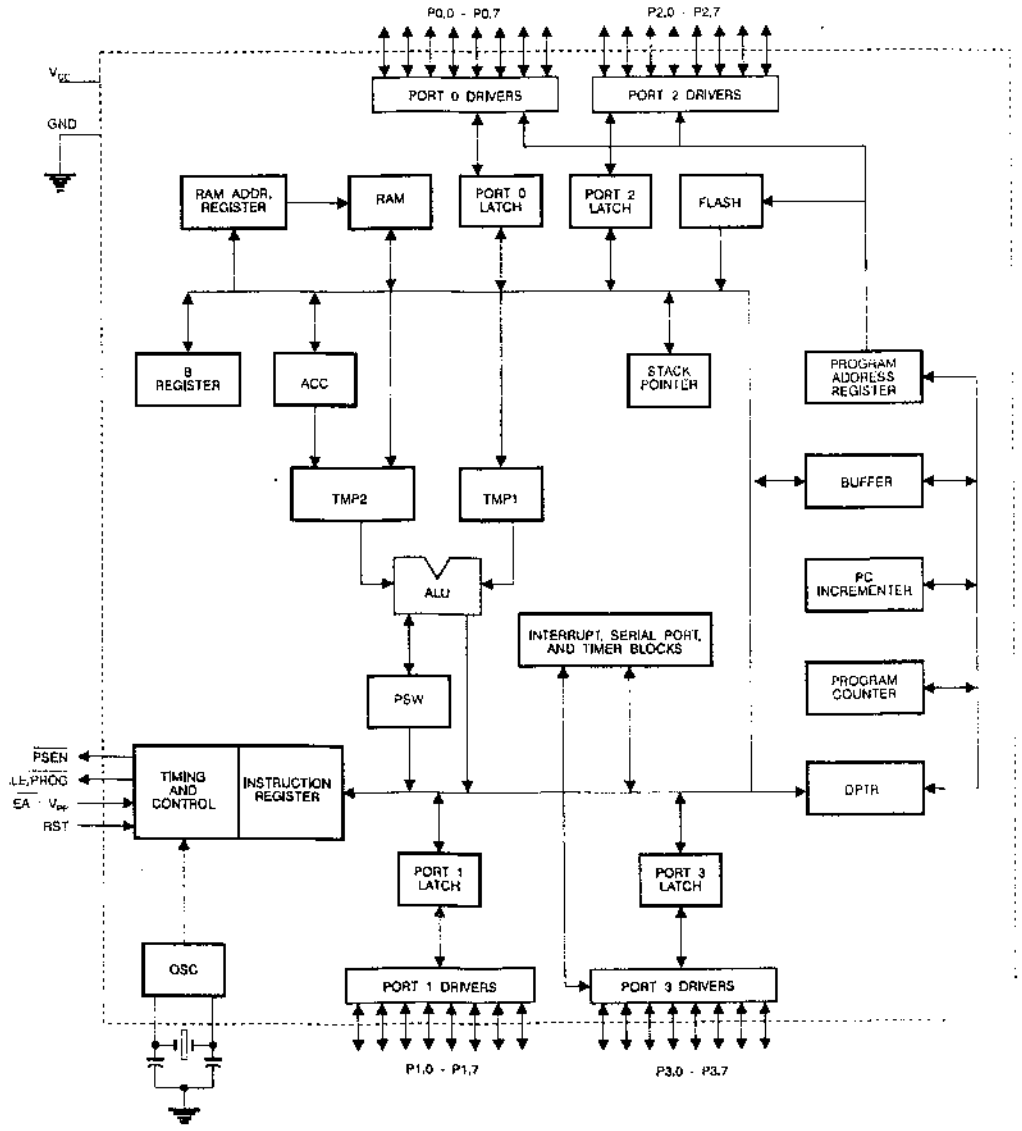
- Có 8 Kbyte bộ nhớ FLASH ROM bên trong dùng để lưu chương trình điều khiển.
- Có 256 Byte RAM nội và các thanh ghi có chức năng đặc biệt.
- 4 Port xuất/nhập (Input/Output) 8 bit.
- Có khả năng giao tiếp truyền dữ liệu nối tiếp.
- Có 6 nguồn ngắt.
- Có ba timer 16 bit.
- Cho phép lập trình nối tiếp theo chuẩn SPI với điện áp lập trình 5V.
- Có thể giao tiếp với 64 Kbyte bộ nhớ bên ngoài dùng để lưu ***chương trình*** điều khiển.
- Có thể giao tiếp với 64 Kbyte bộ nhớ bên ngoài dùng để lưu ***dữ liệu***.
- Có 210 bit có thể truy xuất từng bit. Có các lệnh xử lý bit.

2. Sơ đồ cấu trúc của vi điều khiển AT89S52

Sơ đồ cấu trúc vi điều khiển được trình bày ở hình 2-1.

Các khối bên trong vi điều khiển bao gồm:

- Khối ALU đi kèm với các thanh ghi temp1, temp2 và thanh ghi trạng thái PSW: có chức năng thực hiện các phép toán, các thanh ghi lưu dữ liệu cho khối ALU thực hiện, thanh ghi trạng thái lưu trạng thái của kết quả sau khi thực hiện xong phép toán.
- Bộ điều khiển logic (timing and control): có chức năng điều khiển các khối bên trong và bên ngoài để thực hiện lệnh.
- Vùng nhớ Flash Rom: dùng để lưu chương trình.



***Hình 2-1:** Cấu trúc bên trong của vi điều khiển.*

- Vùng nhớ RAM nội: dùng để lưu dữ liệu.
- Mạch tạo dao động nội kết hợp với tụ thạch anh bên ngoài để tạo dao động.
- Khối xử lý ngắt, truyền dữ liệu, khối timer/counter.
- Thanh ghi A, B, DPTR và 4 port0, port1, port2, port3 có chốt và đệm.

- Thanh ghi bộ đếm chương trình PC (program counter): có chức năng lưu địa chỉ của lệnh sẽ thực hiện, thanh ghi này có chức năng quản lý địa chỉ của bộ nhớ chương trình.
- Con trỏ dữ liệu DPTR (data pointer): dùng để lưu địa chỉ khi truy xuất bộ nhớ dữ liệu bên ngoài hay quản lý địa chỉ của bộ nhớ dữ liệu bên ngoài.
- Thanh ghi con trỏ ngăn xếp SP (stack pointer): quản lý địa chỉ của bộ nhớ ngăn xếp.
- Thanh ghi lệnh IR (instruction register): dùng để lưu mã lệnh đón về từ bộ nhớ.
- Ngoài ra còn có một số các thanh ghi hỗ trợ để quản lý địa chỉ bộ nhớ ram nội bên trong cũng như các thanh ghi quản lý địa chỉ truy xuất bộ nhớ bên ngoài.

3. Khảo sát sơ đồ chân vi điều khiển AT89S52

Sơ đồ chân của vi điều khiển AT89S52 được trình bày ở hình 2-2.

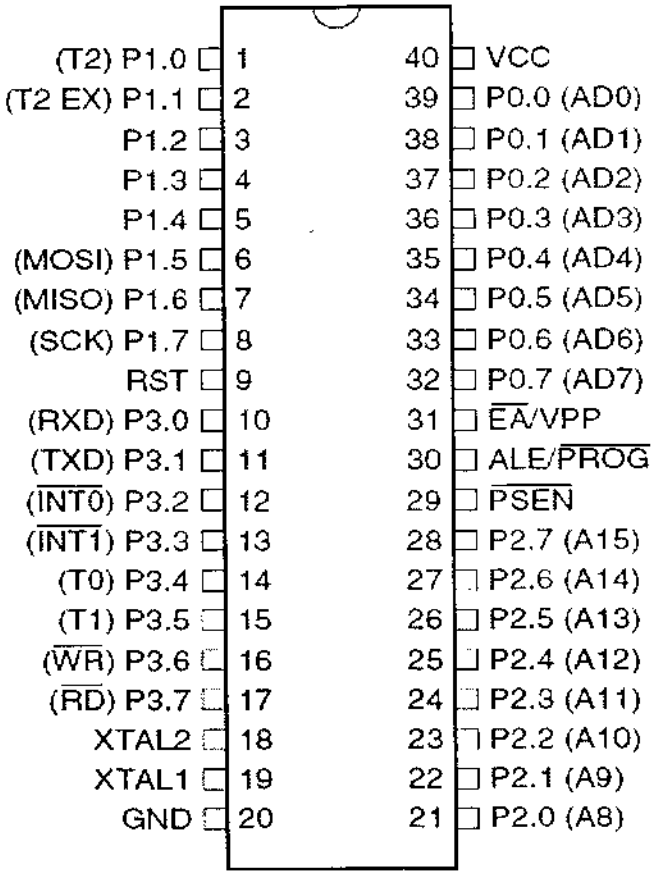
Vi điều khiển AT89S52 có 40 chân. Trong đó có nhiều chân có chức năng kép (một chân có hai chức năng), mỗi đường có thể hoạt động như đường xuất nhập điều khiển IO hoặc là thành phần của các bus dữ liệu và bus địa chỉ để tải địa chỉ và dữ liệu khi giao tiếp với bộ nhớ ngoài.

Chức năng các chân:

- **Port0:** có hai chức năng:

Trong các hệ thống điều khiển đơn giản sử dụng bộ nhớ bên trong không dùng bộ nhớ mở rộng bên ngoài, port0 được dùng làm các đường điều khiển IO (Input- Output).

Trong các hệ thống điều khiển lớn sử dụng thêm bộ nhớ mở rộng bên ngoài, port 0 có chức năng là bus địa chỉ và bus dữ liệu AD7 ÷ AD0. (Address: địa chỉ, Data: dữ liệu)



Hình 2-2: *Sơ đồ chân của AT89S52.*

➢ **Port1:** có 5 chân có hai chức năng: chân P1.0 và P1.1 phục vụ cho timer T2, ba chân P1.5, P1.6, P1.7 có chức năng nạp dữ liệu nối tiếp cho bộ nhớ chương trình nằm bên trong vi điều khiển. Sau khi nạp chương trình ba chân trên có thể làm xuất nhập dữ liệu để điều khiển.

Nên sử dụng ba chân này để xuất dữ liệu điều khiển để không làm ảnh hưởng đến chế độ nạp hoặc phải cách ly.

➢ **Port2:** có hai chức năng:

Trong các hệ thống điều khiển đơn giản không dùng bộ nhớ mở rộng bên ngoài, port2 được dùng làm các đường điều khiển IO (Input- Output).

Trong các hệ thống điều khiển lớn sử dụng thêm bộ nhớ mở rộng bên ngoài, port2 có chức năng là bus địa chỉ cao A8÷A15.

➢ **Port3:** có nhiều chức năng được liệt kê ở bảng 2-1:

Bảng 2-1: Chức năng các chân của port3:

Bit	Tên	Chức năng chuyển đổi
P3.0	RxD	Ngõ vào nhận dữ liệu trong truyền dữ liệu nối tiếp.
P3.1	TxD	Ngõ xuất dữ liệu trong truyền dữ liệu nối tiếp.
P3.2	$\overline{INT0}$	Ngõ vào ngắt cứng thứ 0.
P3.3	$\overline{INT1}$	Ngõ vào ngắt cứng thứ 1.
P3.4	T0	Ngõ vào nhận xung đếm của timer/counter thứ 0.
P3.5	T1	Ngõ vào nhận xung đếm của timer/counter thứ 1.
P3.6	$\overline{WR}$	Tín hiệu điều khiển ghi dữ liệu lên bộ nhớ ngoài.
P3.7	$\overline{RD}$	Tín hiệu điều khiển đọc dữ liệu từ bộ nhớ ngoài.

➢ **Tín hiệu $\overline{PSEN}$ (Program Store Enable):**

$\overline{PSEN}$ là tín hiệu ngõ ra có chức năng cho phép đọc bộ nhớ chương trình mở rộng – thường nối đến chân $\overline{OE}$ (Output enable hoặc $\overline{RD}$) của bộ nhớ EPROM bên ngoài cho phép đọc mã lệnh.

Khi có giao tiếp với bộ nhớ chương trình bên ngoài thì mới dùng đến $\overline{PSEN}$, nếu không có giao tiếp thì chân $\overline{PSEN}$ bỏ trống.

Tín hiệu $\overline{PSEN}$ ở mức thấp trong thời gian vi điều khiển AT89S52 lấy mã lệnh. Các mã lệnh của chương trình đọc từ ROM qua bus dữ liệu và được lưu vào thanh ghi lệnh bên trong vi điều khiển AT89S52 để giải mã lệnh. Khi vi điều khiển AT89S52 thi hành chương trình trong ROM nội thì $\overline{PSEN}$ ở mức logic 1.

➢ **Tín hiệu điều khiển ALE/PROG (Address Latch Enable/Program):**

Khi vi điều khiển AT89S52 truy xuất bộ nhớ bên ngoài, do port0 có chức năng là bus tải địa chỉ và dữ liệu [AD7 ÷ AD0] khi giao tiếp với bộ nhớ ngoài thì phải tách các đường địa chỉ và dữ liệu. Tín hiệu ra ALE dùng làm tín hiệu điều khiển IC chốt để giải đa hợp các đường địa chỉ và dữ liệu được trình bày ở hình 2-3.

Mỗi khi truy xuất bộ nhớ ngoài thì vi điều khiển sẽ xuất địa chỉ của ô nhớ cần truy xuất ra bus địa chỉ là port0 và có thể thêm port2, đồng thời xuất ra một xung ở chân ALE để chốt địa chỉ ở port0 vào IC chốt. Địa chỉ đã được lưu ở ngõ ra của IC chốt được đặt tên là A7 đến A0, port0 bây giờ sẽ có chức năng tải dữ liệu nên được đặt tên là D7 đến D0.

Xung chốt ALE có tần số bằng 1/6 tần số dao động thạch anh gắn vào vi điều khiển, ngoài chức năng chốt địa chỉ thì tín hiệu xung ALE có thể làm xung clock cung cấp cho các phần cứng khác của hệ thống.

Chức năng chính thứ hai của ALE là trong chế độ lập trình cho bộ nhớ nội của vi điều khiển thì chân ALE sẽ nhận xung từ bên ngoài để lập trình lưu mã vào bộ nhớ Flash Rom trong AT89S52 ở chế độ lập trình song – song, còn chế độ nạp nối tiếp thì không cần.

Nếu không dùng bộ nhớ mở rộng thì có thể thả nổi tín hiệu ALE.

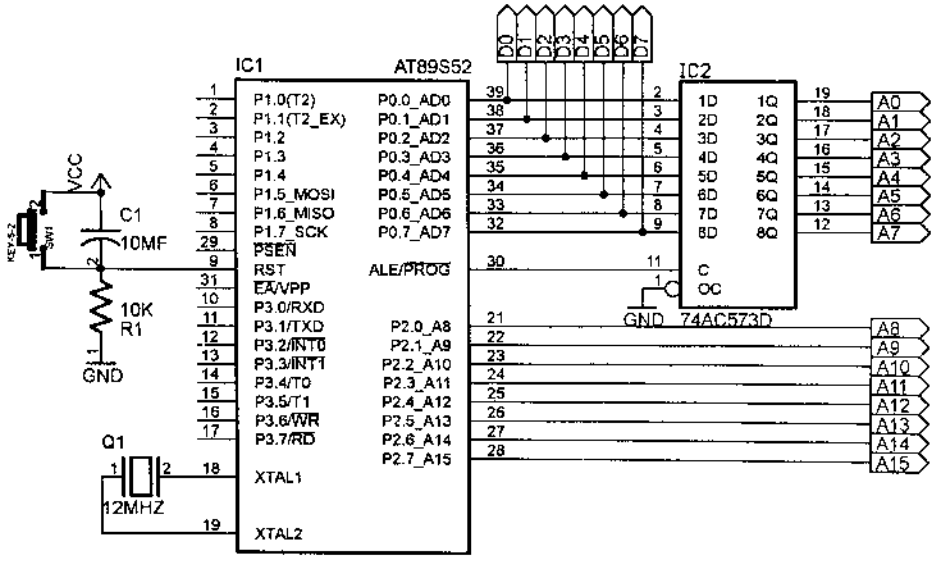
➢ **Tín hiệu $\overline{EA}$ /VPP (External Access):**

Tín hiệu vào $\overline{EA}$ thường nối với mức 1 hoặc mức 0.

Nếu nối $\overline{EA}$ với mức logic 1 (+5V) thì vi điều khiển sẽ thi hành chương trình từ bộ nhớ nội.

Nếu nối $\overline{EA}$ với mức logic 0 (0V) thì vi điều khiển sẽ thi hành chương trình từ bộ nhớ ngoại.

Khi lập trình nạp mã vào bộ nhớ nội dạng song – song, chân này sẽ nhận điện áp lập trình Vpp khoảng 12,5V.



***Hình 2-3:** Kết nối vi điều khiển với IC chốt.*

➢ **Tín hiệu RST (Reset):**

AT89S52 có ngõ vào RST ở chân 9. Sơ đồ kết nối mạch reset như hình 2-3. Khi cấp điện cho hệ thống hoặc khi nhấn nút reset, mạch sẽ reset

vi điều khiển. Tín hiệu reset phải ở mức cao ít nhất là hai chu kỳ máy, khi đó các thanh ghi bên trong được nạp những giá trị thích hợp để khởi động hệ thống.

Trạng thái của tất cả các thanh ghi sau khi reset hệ thống được tóm tắt như bảng 2-2.

Thanh ghi quan trọng nhất là thanh ghi bộ đếm chương trình PC = 0000H. Sau khi reset, vi điều khiển luôn bắt đầu thực hiện chương trình tại địa chỉ 0000H của bộ nhớ chương trình nên các chương trình viết cho vi điều khiển luôn bắt đầu viết tại địa chỉ 0000H.

Nội dung của RAM trong vi điều khiển không bị thay đổi bởi tác động của ngõ vào reset [có nghĩa là vi điều khiển đang sử dụng các thanh ghi để lưu trữ dữ liệu nhưng nếu vi điều khiển bị reset thì dữ liệu trong các thanh ghi vẫn không đổi].

Bảng 2-2: Các thanh ghi sau khi vi điều khiển bị reset.

Tên thanh ghi	Nội dung
Bộ đếm chương trình PC	0000H
Thanh ghi tích lũy A	00H
Thanh ghi B	00H
Thanh ghi trạng thái PSW	00H
Thanh ghi con trỏ SP	07H
DPTR	0000H
Port 0 đến port 3	FFH
IP	×××00000B
IE	0×000000B
Các thanh ghi định thời	00H
SCON	00H
SBUF	00H

➢ **Các ngõ vào bộ dao động Xtal1, Xtal2:**

Bộ dao động được tích hợp bên trong AT89S52, khi sử dụng AT89S52 người thiết kế chỉ cần kết nối thêm tụ thạch anh và các tụ như trong hình 2-3. Tần số tụ thạch anh thường sử dụng cho AT89S53 là 12Mhz ÷ 24Mhz.

➢ **Chân 40 (Vcc) được nối lên nguồn 5V, chân 20 GND nối mass.**

III. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM – BÀI TẬP

1. Câu hỏi ôn tập

Câu số 2-1: Hãy nêu cấu hình của vi điều khiển AT89S52.

Câu số 2-2: Hãy cho biết các loại bộ nhớ mà vi điều khiển AT89S52 tích hợp và mở rộng.

Câu số 2-3: Hãy trình bày tên và chức năng các port của vi điều khiển AT89S52.

2. Câu hỏi mở rộng

Câu số 2-4: Hãy tìm hiểu quá trình phát triển của họ vi điều khiển MCS51 và MCS52.

Câu số 2-5: Hãy tìm hiểu các port vi điều khiển AT89C52 và so sánh với vi điều khiển AT89S52.

Câu số 2-6: Hãy tìm hiểu cấu hình vi điều khiển AT89S8252 và so sánh với vi điều khiển AT89S52.

Câu số 2-7: Hãy tìm hiểu cấu hình vi điều khiển AT89C51RD2 và so sánh với vi điều khiển AT89S52.

Câu số 2-8: Hãy tìm hiểu cấu hình vi điều khiển AT89C2051 và so sánh với vi điều khiển AT89S52.

3. Câu hỏi trắc nghiệm

Câu 2-1: AT89S52 có bao nhiêu port:

(a) 3 (b) 4

(c) 5 (d) 6

Câu 2-2: Tín hiệu nào của AT89S52 có chức năng cho phép đọc bộ nhớ chương trình mở rộng:

(a) ALE (b) PSEN

(c) RST (d) EA

Câu 2-3: Port nào của AT89S52 có chức năng tải địa chỉ byte thấp:

(a) Port 0 (b) Port 1

(c) Port 2 (d) Port 3

Câu 2-4: Port nào của AT89S52 có chức năng tải dữ liệu khi giao tiếp bộ nhớ ngoại:

(a) Port 0 (b) Port 1

(c) Port 2 (d) Port 3

Câu 2-5: Tín hiệu nào của AT89S52 có chức năng chốt địa chỉ:

(a) ALE (b) PSEN

(c) RST (d) EA

Câu 2-6: Port nào của AT89S52 có chức năng tải địa chỉ byte cao:

(a) Port 0 (b) Port 1

(c) Port 2 (d) Port 3

Câu 2-7: Tín hiệu nào của AT89S52 có chức năng reset vi điều khiển:

(a) ALE (b) PSEN

(c) RST (d) EA

Câu 2-8: Khi reset vi điều khiển AT89S52 thì thanh ghi nào mang giá trị 07:

(a) ACC (b) PC

(c) SP (d) PS

Câu 2-9: Khi reset vi điều khiển AT89S52 thì thanh ghi PC bằng:

(a) FFFFH (b) 0007H

(c) 1FFFH (d) 0000H

Câu 2-10: Khi reset vi điều khiển AT89S52 thì thanh ghi nào bằng FFH:

(a) PC (b) DPTR

(c) Các port (d) SP

Câu 2-11: Tín hiệu nào làm AT89S52 thực hiện chương trình ở bộ nhớ nội hay ngoại:

(a) ALE (b) PSEN

(c) RST (d) EA

Câu 2-12: Sau khi tách địa chỉ và dữ liệu của AT89S52 thì bus địa chỉ có:

(a) 8 đường A7 ÷A0 (b) 16 đường A15 ÷A0

(c) 8 đường A15 ÷A8 (d) 8 đường D7 ÷D0

Câu 2-13: AT89S52 có hai ngõ vào nhận tín hiệu ngắt có tên là:

(a) T0, T1 (b) RxD, TxD

(c) $\overline{INT0}$, $\overline{INT1}$ (d) $\overline{RD}$, $\overline{WR}$

Câu 2-14: Port nào của AT89S52 có chức năng nhận xung cho timer T2:

(a) Port 0 (b) Port 1

(c) Port 2 (d) Port 3

Câu 2-15: Port nào của AT89S52 có chức năng giao tiếp mạch nạp nối tiếp:

(a) Port 0 (b) Port 1

(c) Port 2 (d) Port 3

4. Bài tập

Chương 3
VI ĐIỀU KHIỂN AT89S52: TỔ CHỨC BỘ NHỚ, THANH GHI

- **GIỚI THIỆU**
- **KIẾN TRÚC BỘ NHỚ**
- **TỔ CHỨC BỘ NHỚ CỦA VI ĐIỀU KHIỂN ATMEL AT89S52**
  - TỔ CHỨC BỘ NHỚ
  - KHẢO SÁT BỘ NHỚ RAM
  - KHẢO SÁT CÁC THANH GHI CÓ CHỨC NĂNG ĐẶC BIỆT
- **CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP**
  - CÂU HỎI ÔN TẬP
  - CÂU HỎI MỞ RỘNG
  - CÂU HỎI TRẮC NGHIỆM
  - BÀI TẬP

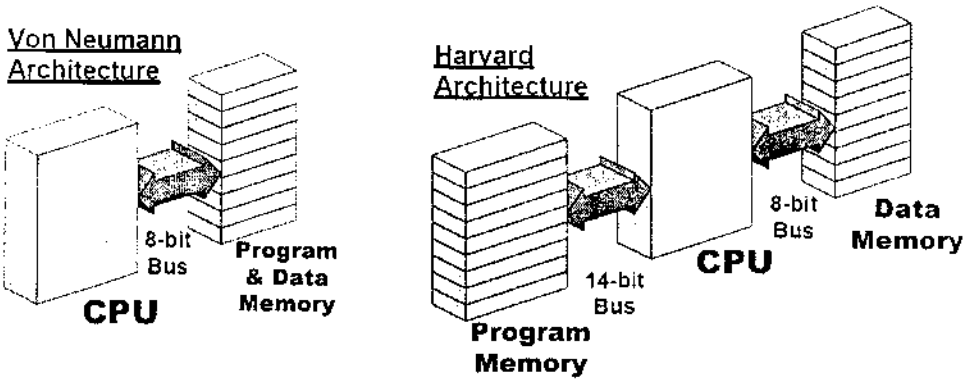
I. GIỚI THIỆU

Ở chương này khảo sát tổ chức bộ nhớ bên trong, các thanh ghi của vi điều khiển 8 bit. Sau khi kết thúc chương này thì người đọc có thể biết tổ chức bộ nhớ bên trong, chức năng của từng loại bộ nhớ, tên và chức năng của các thanh ghi đặc biệt.

II. KIẾN TRÚC BỘ NHỚ

Có hai loại kiến trúc bộ nhớ cơ bản là kiến trúc Von Neumann và Harvard.

Hình 3-1 trình bày hai kiến trúc:



Hình 3-1: Kiến trúc Von Neumann và Harvard.

➢ ***Kiến trúc Von Neumann***: với kiến trúc này thì bộ nhớ giao tiếp với CPU thông qua một bus dữ liệu 8 bit, bộ nhớ có các ô nhớ chứa dữ liệu 8 bit, bộ nhớ vừa lưu trữ chương trình và dữ liệu.

Ưu điểm: kiến trúc đơn giản.

Khuyết điểm: do chỉ có một bus nên tốc độ truy suất chậm, khó thay đổi dung lượng lưu trữ của ô nhớ.

➢ ***Kiến trúc Harvard***: với kiến trúc này thì bộ nhớ được tách ra làm hai loại bộ nhớ độc lập: bộ nhớ lưu chương trình và bộ nhớ lưu dữ liệu, CPU giao tiếp với hai bộ nhớ độc lập nên cần hai bus độc lập. Vì độc lập nên có thể thay đổi số bit lưu trữ của từng bộ nhớ mà không ảnh hưởng lẫn nhau.

Ưu điểm: do có hai bus nên tốc độ truy suất nhanh, tùy ý thay đổi số bit của ô nhớ.

Khuyết điểm: kiến trúc phức tạp.

III. TỔ CHỨC BỘ NHỚ CỦA VI ĐIỀU KHIỂN ATMEL AT89S52

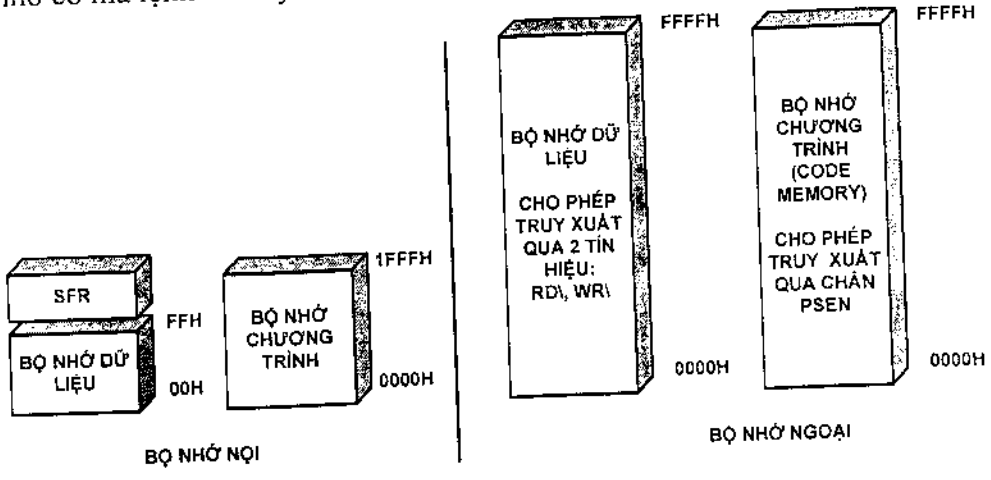
1. Tổ chức bộ nhớ

Vi điều khiển AT89S52 có ***bộ nhớ nội bên trong*** và có thêm khả năng giao tiếp với ***bộ nhớ bên ngoài*** nếu bộ nhớ bên trong không đủ khả năng lưu trữ chương trình.

Bộ nhớ nội bên trong gồm có hai loại bộ nhớ: bộ nhớ dữ liệu và bộ chương trình theo kiến trúc Harvard. Bộ nhớ dữ liệu có 256 byte, bộ nhớ chương trình có dung lượng 8kbyte.

Bộ nhớ mở rộng bên ngoài cũng được chia ra làm hai loại bộ nhớ: bộ nhớ dữ liệu và bộ nhớ chương trình. Khả năng giao tiếp là 64kbyte cho mỗi loại. Hình 3-2 minh họa khả năng giao tiếp bộ nhớ của vi điều khiển AT89S52.

Bộ nhớ chương trình chia thành nhiều trang, mỗi trang có dung lượng 2Kbyte, các lệnh gọi chương trình con, lệnh nhảy trong cùng một trang bộ nhớ có mã lệnh là 2 byte, vượt quá một trang thì mã lệnh là 3 byte.



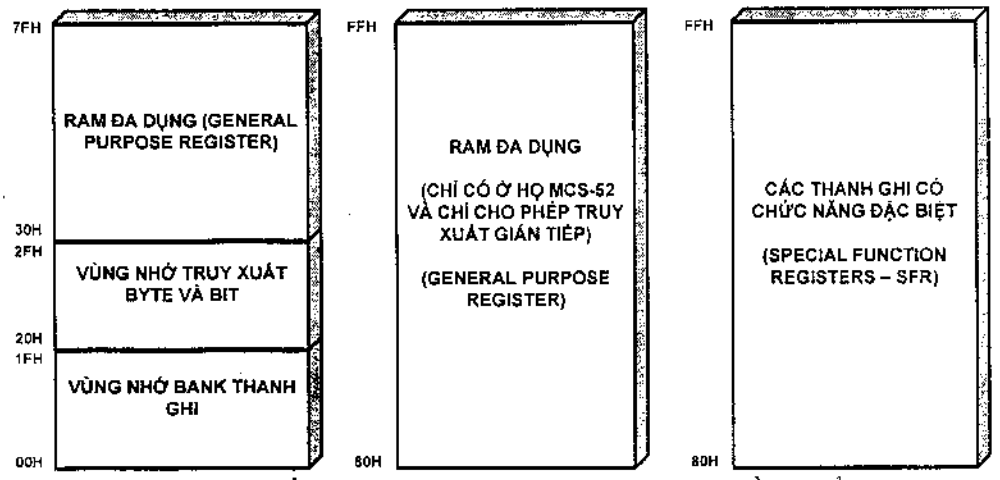
Hình 3-2: Các vùng nhớ AT89S52.

Ngoài chức năng phân chia bộ nhớ theo trang, bộ nhớ mở rộng bên ngoài và bộ nhớ chương trình bên trong không có gì đặc biệt – chỉ có chức năng lưu trữ dữ liệu và chương trình nên không cần phải khảo sát.

Bộ nhớ chương trình bên trong của vi điều khiển thuộc loại bộ nhớ Flash Rom cho phép xóa và lập trình lại bằng xung điện.

Bộ nhớ RAM nội bên trong là một bộ nhớ đặc biệt nên người sử dụng vi điều khiển cần phải nắm rõ các tổ chức và các chức năng đặc biệt của bộ nhớ này.

Sơ đồ cấu trúc bên trong của bộ nhớ này được trình bày như hình 3-3.



***Hình 3-3:** Cấu trúc bộ nhớ RAM bên trong vi điều khiển.*

RAM bên trong AT89S52 được phân chia như sau:

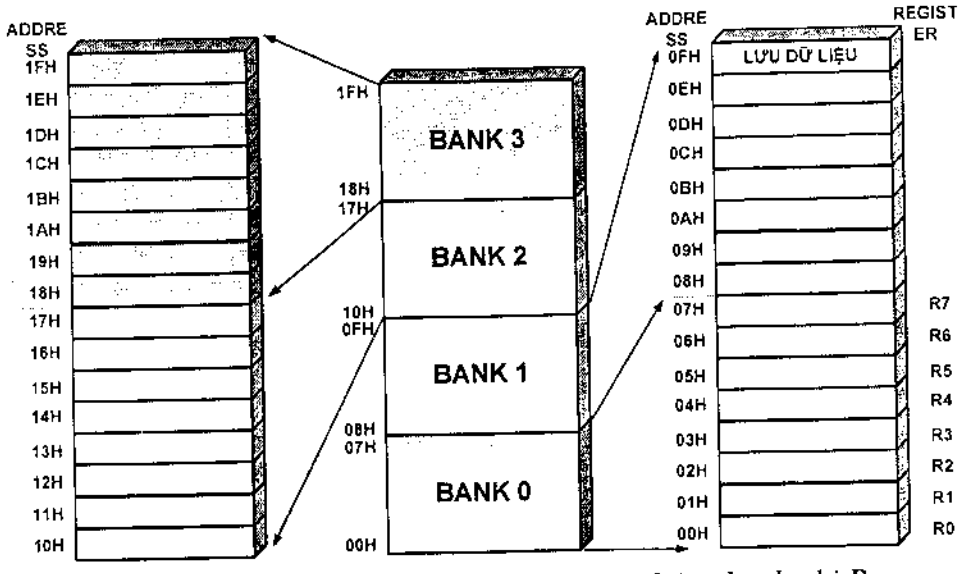
- Các bank thanh ghi có địa chỉ từ 00H đến 1FH.
- RAM truy xuất từng bit có địa chỉ từ 20H đến 2FH.
- RAM đa dụng từ 30H đến 7FH cho phép truy xuất trực tiếp hoặc gián tiếp.
- RAM đa dụng từ 80H đến FFH chỉ cho phép truy xuất gián tiếp.
- Các thanh ghi chức năng đặc biệt từ 80H đến FFH.

2. Khảo sát bộ nhớ RAM

➢ Vùng nhớ các bank thanh ghi :

Vùng nhớ này có 32 byte được thiết kế và phân chia thành 4 bank thanh ghi, cho phép truy xuất trực tiếp dùng địa chỉ, hoặc truy xuất gián tiếp.

Ngoài hai cách truy xuất trên thì tập lệnh AT89S52 hỗ trợ thêm kiểu truy xuất dùng thanh ghi từ R0 đến R7 và theo mặc định sau khi reset hệ thống thì các thanh ghi R0 đến R7 được gán cho 8 ô nhớ có địa chỉ từ 00H đến 07H được minh họa bởi hình 3-4, khi đó bank 0 có ba cách truy xuất bằng địa chỉ trực tiếp, gián tiếp và bằng thanh ghi R.



***Hình 3-4:** Vùng nhớ bank thanh ghi và nhóm thanh ghi R.*

Các lệnh dùng các thanh ghi R0 đến R7 sẽ có số lượng byte mã lệnh ít hơn và thời gian thực hiện lệnh nhanh hơn so với các lệnh có chức năng tương ứng nếu dùng kiểu địa chỉ trực tiếp.

Các dữ liệu dùng thường xuyên nên lưu trữ ở một trong các thanh ghi này.

Do có 4 bank thanh ghi nên tại một thời điểm chỉ có một bank thanh ghi được truy xuất bởi các thanh ghi R0 đến R7, để chuyển đổi việc truy xuất các bank thanh ghi ta phải thay đổi các bit chọn bank trong thanh ghi trạng thái.

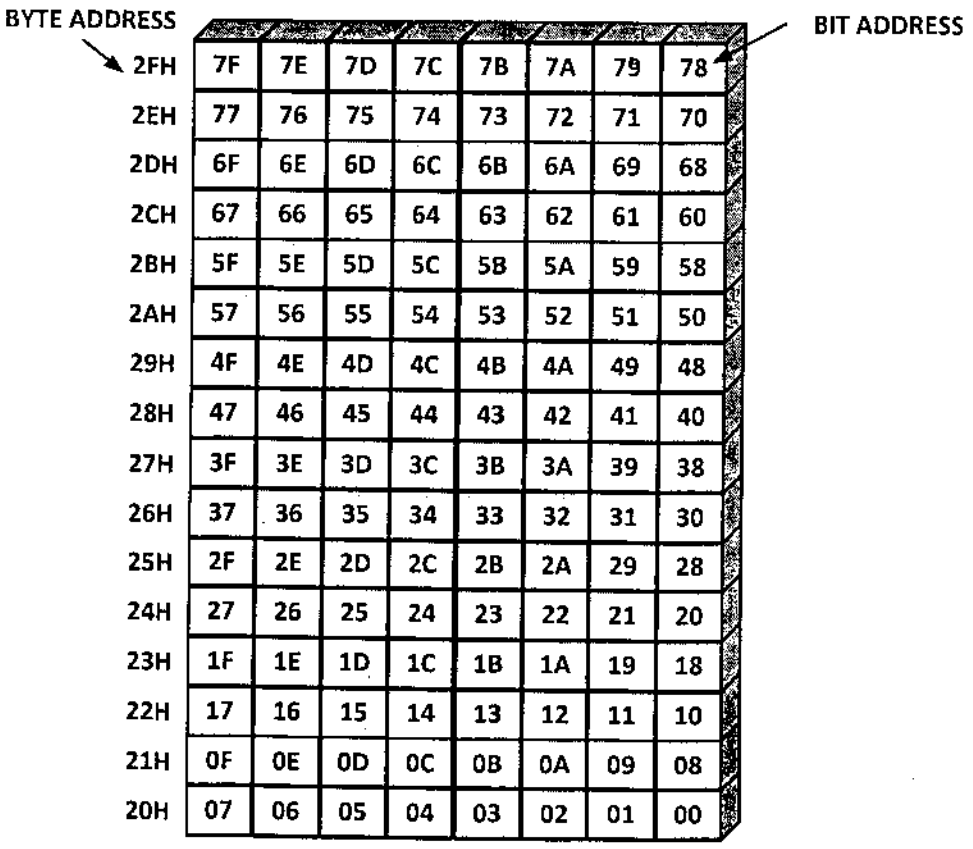
Người lập trình dùng vùng nhớ 4 bank thanh ghi để lưu trữ dữ liệu phục vụ cho việc xử lý dữ liệu khi viết chương trình.

Chức năng chính của 4 bank thanh ghi này là nếu trong hệ thống có sử dụng nhiều chương trình thì chương trình thứ nhất bạn có thể sử dụng hết các thanh ghi R0 đến R7 của bank0, khi chuyển sang chương trình thứ hai để xử lý một công việc gì đó và vẫn sử dụng các thanh ghi R0 đến R7 để lưu trữ cho việc xử lý dữ liệu mà không làm ảnh hưởng đến các dữ liệu R0 đến R7 trước đây và không cần phải thực hiện công việc cất dữ liệu thì cách nhanh nhất là gán nhóm thanh ghi R0 đến R7 cho bank1 là xong. Tương tự có thể mở thêm hai chương trình nữa và gán cho các bank 3 và 4.

➢ **RAM có thể truy xuất từng bit:**

Vi điều khiển AT89S52 có 210 ô nhớ bit có thể truy xuất từng bit, trong đó có 128 bit nằm ở các ô nhớ byte có địa chỉ từ 20H đến 2FH và các

bit còn lại chứa trong nhóm thanh ghi có chức năng đặc biệt. Sơ đồ vùng nhớ truy xuất từng bit như hình 3-5.



BYTE ADDRESS	BIT ADDRESS							
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00

Hình 3-5: *Vùng nhớ truy xuất từng bit.*

Các ô nhớ cho phép truy xuất từng bit và các lệnh xử lý bit là một thế mạnh của vi điều khiển. Các bit có thể được đặt, xóa, AND, OR bằng một lệnh duy nhất, trong khi đó để xử lý các bit thì ***vi xử lý*** vẫn có thể xử lý được nhưng phải sử dụng rất nhiều lệnh để đạt được cùng một kết quả vì ***vi xử lý*** thường xử lý byte.

Các port cũng có thể truy xuất được từng bit.

128 ô nhớ bit cho phép truy xuất từng bit và cũng có thể truy xuất byte phụ thuộc vào lệnh được dùng là lệnh xử lý bit hay lệnh xử lý byte. Chú ý địa chỉ của ô nhớ byte và bit trùng nhau.

Người lập trình dùng vùng nhớ này để lưu trữ dữ liệu phục vụ cho việc xử lý dữ liệu byte hoặc bit. Các dữ liệu xử lý bit nên lưu vào vùng nhớ này.

Chú ý: các ô nhớ nào mà chia ra làm 8 và có các con số bên trong là các ô nhớ vừa cho truy xuất byte và cả truy xuất bit. Những ô nhớ còn lại thì không thể truy xuất bit. Các số nằm bên trong từng ô bit là địa chỉ của từng bit.

➢ **RAM đa dụng cho phép truy xuất trực tiếp, gián tiếp:**

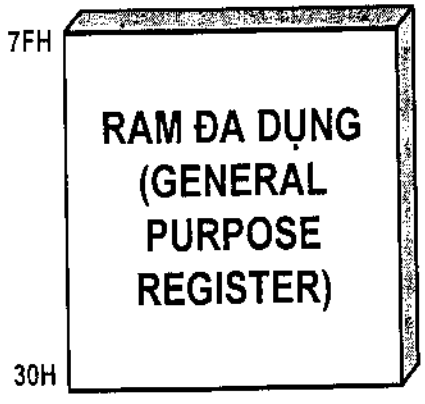
Vùng nhớ ram đa dụng gồm có 80 byte có địa chỉ từ 30H đến 7FH, xem hình 3-6.

Vùng nhớ này không có gì đặc biệt so với hai vùng nhớ trên. Vùng nhớ bank thanh ghi 32 byte từ 00H đến 1FH cũng có thể dùng làm vùng nhớ ram đa dụng mặc dù các ô nhớ này có chức năng như đã trình bày.

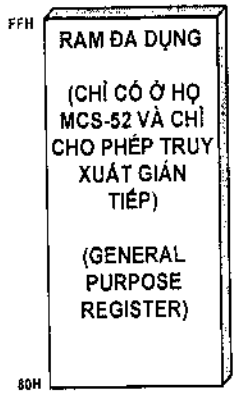
Mọi địa chỉ trong vùng RAM đa dụng đều có thể truy xuất tự do dùng kiểu định địa chỉ trực tiếp hoặc gián tiếp.

➢ **RAM đa dụng chỉ cho phép truy xuất gián tiếp:**

Vùng nhớ ram đa dụng gồm 128byte có địa chỉ từ 80H đến FFH, xem hình 3-7, vùng nhớ này không có gì đặc biệt chỉ dùng để lưu dữ liệu và chỉ cho phép truy xuất gián tiếp dùng thanh ghi.



Hình 3-6: Vùng nhớ đa dụng.



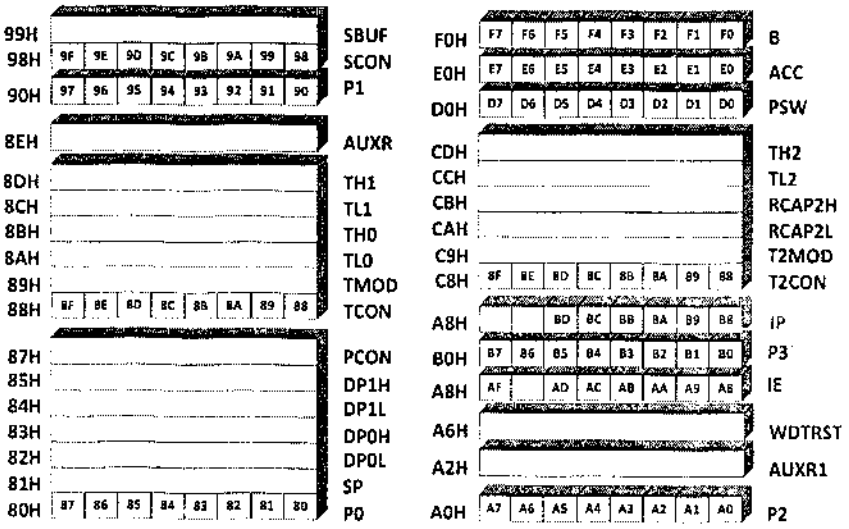
Hình 3-7: Vùng nhớ đa dụng.

3. Khảo sát các thanh ghi có chức năng đặc biệt

Các thanh ghi nội của AT89S52 được truy xuất ngầm bởi tập lệnh.

Các thanh ghi trong AT89S52 được định dạng như một phần của RAM trên chip vì vậy mỗi thanh ghi sẽ có một địa chỉ, các thanh ghi này chỉ có cho phép truy xuất trực tiếp, nếu truy xuất gián tiếp sẽ trùng với vùng nhớ RAM có địa chỉ từ 80H đến FFH.

Vi điều khiển AT89S52 có 32 thanh ghi có chức năng đặc biệt nằm ở vùng trên của RAM nội có địa chỉ từ 80H đến FFH như hình 3-8.



Hình 3-8: Các thanh ghi có chức năng đặc biệt.

Chú ý: 128 ô nhớ có địa chỉ từ 80H đến FFH thì chỉ có 32 thanh ghi có chức năng đặc biệt được xác định – còn các ô nhớ còn lại thì chưa thiết lập và trong tương lai sẽ được các nhà thiết kế vi điều khiển thiết lập thêm, khi đó sẽ có các vi điều khiển thế hệ mới hơn.

Trong phần này chỉ giới thiệu các phần tiếp theo sẽ trình bày chi tiết hơn về các thanh ghi này.

➢ Các ô nhớ có địa chỉ 80H, 90H, A0H, B0H

Là các Port của vi điều khiển AT89S52 bao gồm Port0 có địa chỉ 80H, port1 có địa chỉ 90H, port2 có địa chỉ A0H và port3 có địa chỉ B0H. Tất cả các port này đều có thể truy xuất từng bit nên rất thuận tiện trong điều khiển. Địa chỉ của các bit được đặt tên với ô bắt đầu chính là địa chỉ của port tương ứng ví dụ như bit đầu tiên của port0 là 80H cũng chính là địa chỉ bắt đầu của port0. Người lập trình không cần nhớ địa chỉ các bit trong các port vì phần mềm lập trình cho phép truy xuất bằng tên từng bit dễ nhớ như sau: P0.0 chính là bit có địa chỉ 80h của port0.

Ngoại trừ thanh ghi A có thể được truy xuất ngầm, đa số các thanh ghi có chức năng đặc biệt SFR (Special Function Register) có thể địa chỉ hóa từng bit hoặc byte.

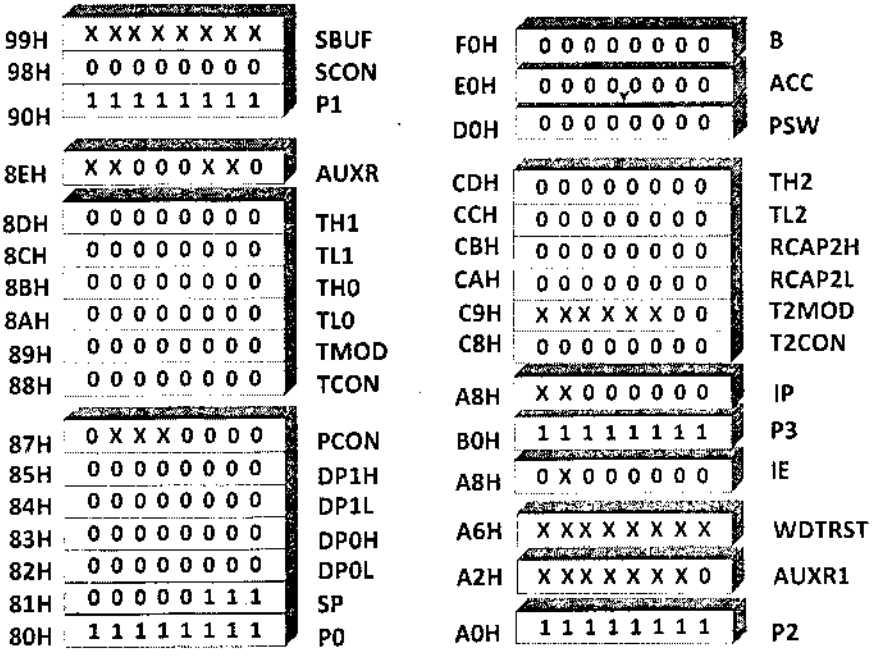
➢ Ô nhớ có địa chỉ 81H

Là thanh ghi con trỏ ngăn xếp SP (Stack Pointer) - có chức năng quản lý địa chỉ của bộ nhớ ngăn xếp. Bộ nhớ ngăn xếp dùng để lưu trữ tạm thời các dữ liệu trong quá trình thực hiện chương trình của vi điều khiển.

Bộ nhớ ngăn xếp của AT89S52 nằm trong RAM nội và chỉ cho phép truy xuất địa chỉ gián tiếp. Dung lượng bộ nhớ ngăn xếp lớn nhất là 256 byte ram nội của AT89S52.

Khi reset AT89S52, thanh ghi SP sẽ mang giá trị mặc định là 07H và dữ liệu đầu tiên sẽ được cất vào ô nhớ ngăn xếp có địa chỉ 08H.

Nếu phần mềm ứng dụng không khởi tạo SP một giá trị mới thì bank 1 và có thể cả bank 2, bank 3 sẽ không dùng được vì vùng nhớ này đã được dùng làm ngăn xếp.



***Hình 3-9:** Giá trị các thanh ghi chức năng sau khi reset.*

➢ Ô nhớ có địa chỉ 82H và 83H

Là hai thanh ghi: DP0L (byte thấp) có địa chỉ là 82H và DP0H (byte cao) có địa chỉ 83H. Hai thanh ghi này có thể sử dụng độc lập để lưu trữ dữ liệu và có thể kết hợp lại tạo thành một thanh ghi 16 bit có tên là DPTR (Data Pointer) – gọi là con trỏ dữ liệu - được dùng để lưu địa chỉ 16 bit khi truy xuất dữ liệu của bộ nhớ dữ liệu bên ngoài hay DPTR dùng để quản lý địa chỉ bộ nhớ dữ liệu bên ngoài.

➢ Ô nhớ có địa chỉ 84H và 85H

Là hai thanh ghi: DP1L (byte thấp) có địa chỉ là 84H và DP1H (byte cao) có địa chỉ 85H. Có chức năng giống như DP0L và DP0H.

➢ **Ô nhớ có địa chỉ 87H**

Là thanh ghi PCON (Power Control) có chức năng điều khiển công xuất khi vi điều khiển làm việc hay ở chế độ chờ. Khi không còn xử lý gì nữa thì người lập trình có thể lập trình cho vi điều khiển chuyển sang chế độ chờ để giảm bớt công suất tiêu thụ nhất là khi nguồn cung cấp cho vi điều khiển là pin.

➢ **Các ô nhớ có địa chỉ từ 88H đến 8DH**

Gồm 6 thanh ghi phục vụ cho hai timer/counter T1, T0.

Thanh ghi TCON (timer control): thanh ghi điều khiển timer/counter.

Thanh ghi TMOD (timer mode): thanh ghi lựa chọn mode hoạt động cho timer/counter.

Thanh ghi TH0 và TL0 kết hợp lại tạo thành một thanh ghi 16 bit có chức năng lưu trữ xung đếm cho timer/counter T0. Tương tự cho hai thanh ghi TH1 và TL1 kết hợp lại để lưu trữ xung đếm cho timer/counter T1. Khả năng lưu trữ số lượng xung đếm được là 65536 xung.

Chức năng của các thanh ghi này sẽ được trình bày rõ ở phần timer – counter.

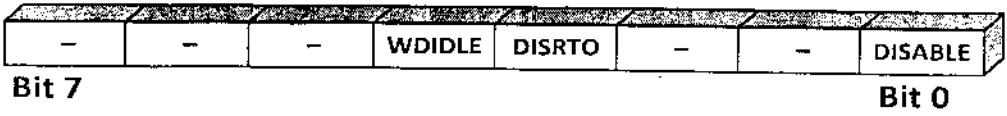
➢ **Các ô nhớ có địa chỉ 98H và 99H**

Gồm hai thanh ghi SCON và SBUF. SCON (series control): thanh ghi điều khiển truyền dữ liệu nối tiếp. Sbuf (series buffer): thanh ghi đệm dữ liệu truyền nối tiếp. Dữ liệu muốn truyền đi thì phải lưu vào thanh ghi SBUF và dữ liệu nhận về nối tiếp cũng lưu ở thanh ghi này. Khi có sử dụng truyền dữ liệu thì phải sử dụng hai thanh ghi này.

Chức năng của các thanh ghi này sẽ được trình bày rõ ở phần truyền dữ liệu.

➢ **Ô nhớ có địa chỉ 8EH**

Là thanh ghi AUXR 8 bit nhưng chỉ sử dụng 3bit: bit DISALE, bit DISRTO và bit WDIDLE và không cho phép truy xuất bit.



Hình 3-10: Thanh ghi AUXR.

Bit DISALE (Disable/Enable ALE – bit cấm/cho phép xuất tín hiệu ALE) khi bằng 0 thì cho phép chân ALE xuất tín hiệu chốt địa chỉ liên

tục có tần số bằng 1/6 tần số tụ thạnh anh, nếu bằng 1 thì chỉ tạo xung chốt khi truy xuất bộ nhớ ngoại hoặc bộ nhớ chương trình bằng hai lệnh MOVX và MOVC.

Bit DISRTO (Disable/Enable Reset Out – bit cấm/cho phép ngõ ra reset) khi bằng 0 thì cho phép reset CPU bởi bộ định thời Watchdog timer (WDT) khi đếm hết thời gian định thời, nếu bằng 1 thì chỉ cho phép reset CPU bởi ngõ vào reset ở chân RST số 9.

Bit WDIDLE (Disable/Enable WDT in IDLE mode – bit cấm/cho phép WDT đếm ở chế độ ngừng) khi bằng 0 thì WDT vẫn tiếp tục đếm khi CPU ở chế độ chờ IDLE, nếu bằng 1 thì WDT ngừng đếm khi CPU ở chế độ chờ IDLE.

➢ **Các ô nhớ có địa chỉ A8H và B8H**

Là hai thanh ghi IE và IP. Thanh ghi IE (Interrupt Enable): thanh ghi điều khiển cho phép / không cho phép ngắt. Thanh ghi IP (Interrupt Priority): thanh ghi điều khiển ưu tiên ngắt. Khi có sử dụng đến ngắt thì phải dùng đến hai thanh ghi này. Mặc nhiên các thanh ghi này được khởi tạo ở chế độ cấm ngắt.

Chức năng của các thanh ghi này sẽ được trình bày rõ ở phần ngắt.

➢ **Ô nhớ có địa chỉ A2H**

Là thanh ghi AUXR1 8 bit nhưng chỉ sử dụng một bit là DPS và không cho phép truy xuất bit.

Hình 3-11: Thanh ghi AUXR1.

Bit DPS khi bằng 0 thì thanh ghi DPTR sẽ dùng cặp thanh ghi DP0L và DP0H, nếu bằng 1 thì dùng cặp thanh ghi DP1L và DP1H.

➢ **Ô nhớ có địa chỉ A6H**

Là thanh ghi WDTRST 8 bit.

➢ **Các ô nhớ có địa chỉ từ C8H đến CDH**

Gồm 6 thanh ghi phục vụ cho timer/counter T2.

- T2CON (timer control): thanh ghi điều khiển timer/counter T2.
- T2MOD (timer mode): thanh ghi lựa chọn mode hoạt động cho timer/counter T2.

- RCAP2L (Register Capture Low): thanh ghi byte thấp lưu giá trị xung đếm ở chế độ Capture.
- RCAP2H (Register Capture high): thanh ghi byte cao lưu giá trị xung đếm ở chế độ Capture.
- Thanh ghi TH2 và TL2 kết hợp lại tạo thành một thanh ghi 16 bit có chức năng lưu trữ xung đếm cho timer/counter T2. Chức năng của các thanh ghi này sẽ được trình bày rõ ở phần timer – counter.

➢ **Thanh ghi trạng thái chương trình (PSW: Program Status Word)**

Thanh ghi trạng thái chương trình ở địa chỉ D0H được tóm tắt như bảng 3-3:

***Bảng 3-3:** Các bit trong thanh ghi trạng thái.*

Bit thứ	Kí hiệu	Địa chỉ bit	Mô tả
PSW.7	C	D7H	Carry Flag
PSW.6	AC	D6H	Auxiliary Carry Flag
PSW.5	F0	D5H	Flag 0 còn gọi là cờ Zero kí hiệu là Z
PSW4	RS1	D4H	Register Bank Select 1: bit lựa chọn bank thanh ghi.
PSW.3	RS0	D3H	Register Bank Select 0: bit lựa chọn bank thanh ghi.
PSW.2	OV	D2H	Overflow Flag: cờ tràn số nhị phân có dấu.
PSW.1	-	D1H	Reserved: chưa thiết kế nên chưa sử dụng được.
PSW.0	P	D0H	Even Parity Flag: sử dụng kiểm tra cờ chẵn.

Chức năng từng bit trạng thái:

- ***Cờ Carry C (Carry Flag):*** Cờ C được sử dụng cho các lệnh toán học:

C = 1 nếu phép toán cộng có tràn hoặc phép trừ có mượn.

C = 0 nếu phép toán cộng không tràn và phép trừ không có mượn.

- ***Cờ Carry phụ AC (Auxiliary Carry Flag)***

Khi cộng những giá trị BCD (Binary Code Decimal), cờ nhớ phụ AC =1 nếu kết quả 4 bit thấp lớn hơn 09H, ngược lại AC= 0. Cờ AC được dùng để chỉnh số BCD khi thực hiện lệnh cộng hai số BCD.

- *Cờ 0 (Flag 0)*

Cờ 0 (F0) còn gọi là cờ Z, cờ Z =1 khi kết quả xử lý bằng 0 và cờ Z = 0 khi kết quả khác 0.

- *Các bit chọn bank thanh ghi truy xuất*

Hai bit RS1 và RS0 dùng để thay đổi cách gán 8 thanh ghi R7 – R0 cho một trong 4 bank thanh ghi. Hai bit này sẽ bị xóa sau khi reset vi điều khiển và được thay đổi bởi chương trình của người lập trình.

Bảng 3-4: Các bit lựa chọn bank thanh ghi:

RS1	RS0	Bank thanh ghi được lựa chọn
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

- *Cờ tràn OV (Over Flag)*

Khi các số có dấu được cộng hoặc trừ với nhau, phần mềm có thể kiểm tra bit này để xác định xem kết quả có nằm trong vùng giá trị xác định hay không. Với số nhị phân 8 bit có dấu thì số dương từ 0 đến +127, số âm từ -128 đến – 1. Nếu kết quả cộng hai số dương lớn hơn +127 hoặc cộng hai số âm kết quả nhỏ hơn –128 thì kết quả đã vượt ra ngoài vùng giá trị cho phép thì khối ALU trong vi điều khiển sẽ làm bit OV = 1.

Khi cộng các số nhị phân không dấu thì không cần quan tâm đến bit OV.

- *Bit Parity (P)*

Bit P tự động được Set hay Clear ở mỗi chu kỳ máy để lập Parity chẵn với thanh ghi A. Đếm các bit 1 trong thanh ghi A cộng với bit Parity luôn luôn là số chẵn. Ví dụ thanh ghi A chứa nhị phân 10101101B thì bit P set lên một để cho biết tổng số bit 1 trong thanh ghi A và cả bit P tạo thành số chẵn.

Bit Parity thường được dùng kết hợp với những thủ tục truyền dữ liệu nối tiếp để tạo ra bit Parity cho dữ liệu trước khi truyền đi hoặc kiểm tra bit Parity sau khi nhận dữ liệu.

➢ **Thanh ghi ACC**

Thanh ghi ACC ở địa chỉ E0H là thanh ghi đặc biệt cùng với ALU thực hiện nhiều phép toán và nhiều lệnh chỉ có hiệu lực đối với thanh ghi ACC. Các lệnh bình thường thì dùng kí hiệu là thanh ghi A nhưng khi thực hiện lệnh Push và Pop thì phải dùng kí hiệu là ACC vì hai lệnh này chỉ dùng địa chỉ trực tiếp.

➢ **Thanh ghi B**

Thanh ghi B ở địa chỉ F0H cùng với thanh ghi A để thực hiện các phép toán nhân chia.

Lệnh MUL AB: sẽ nhân những giá trị không dấu 8 bit với 8 bit trong hai thanh ghi A và B, rồi trả về kết quả 16 bit trong A (byte cao) và B (byte thấp).

Lệnh DIV AB: lấy giá trị trong thanh ghi A chia cho giá trị trong thanh ghi B, kết quả nguyên lưu trong A, số dư lưu trong B.

Có thể dùng thanh ghi B để lưu dữ liệu.

IV. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP

1. Câu hỏi ôn tập

Câu số 3-1: Hãy cho biết các loại bộ nhớ mà vi điều khiển AT89S52 tích hợp và mở rộng.

Câu số 3-2: Hãy trình bày cấu trúc của bộ nhớ RAM nội của vi điều khiển AT89S52.

Câu số 3-3: Hãy trình bày tên và chức năng các thanh ghi đặc biệt của vi điều khiển AT89S52.

2. Câu hỏi mở rộng

Câu số 3-4: Hãy tìm hiểu tổ chức bộ nhớ vi điều khiển AT89C51RD2 và so sánh với AT89S52.

3. Câu hỏi trắc nghiệm

Câu 3-1: Bộ nhớ chương trình của AT89S52 có dung lượng là:

(a) 8K×Byte (b) 8K×14bit

(c) 8K×16bit (d) 368Byte

Câu 3-2: Bộ nhớ dữ liệu của AT89S52 có dung lượng là:

(a) 128×Byte (b) 256×Byte

(c) 1K×Byte (d) 368Byte

Câu 3-3: AT89S52 có số bank thanh ghi là:

(a) 2 (b) 3

(c) 4 (d) 5

Câu 3-4: Vùng nhớ các bank thanh ghi của AT89S52 có:

(a) 8 byte (b) 16 byte

(c) 32 byte (d) 64 byte

Câu 3-5: Vùng nhớ cho phép truy xuất bit của AT89S52 có:

(a) 8 byte (b) 16 byte

(c) 32 byte (d) 64 byte

Câu 3-6: Địa chỉ của vùng nhớ cho phép truy xuất bit của AT89S52:

(a) Từ 00H đến 7FH (b) Từ 00H đến 1FH

(c) Từ 20H đến 2FH (d) Từ 30H đến 7FH

Câu 3-7: Địa chỉ của vùng nhớ chương trình của AT89S52:

(a) Từ 0000H đến 1FFFH (b) Từ 0000H đến FFFFH

(c) Từ 00H đến FFH (d) Từ 80H đến FFH

Câu 3-8: Vùng nhớ ram nào của AT89S52 chỉ cho phép truy xuất gián tiếp:

(a) Từ 00H đến FFH (b) Từ 00H đến 7FH

(c) Từ 80H đến FFH (d) Từ 00H đến 1FH

Câu 3-9: Địa chỉ mặc nhiên bộ nhớ ngăn xếp của AT89S52 bắt đầu tại địa chỉ:

(a) 0000H (b) 20H

(c) 07H (d) 80H

Câu 3-10: Thanh ghi quản lý địa chỉ bộ nhớ ngăn xếp là:

(a) PC (b) SP

(c) DPTR (d) IP

Câu 3-11: Mỗi bank thanh ghi của AT89S52 có:

(a) 8 byte (b) 16 byte

(c) 32 byte (d) 64 byte

Câu 3-12: Bit lựa chọn bank thanh ghi của AT89S52 là:

(a) OV và Z (b) RS1 và RS0

(c) SR1 và SR0 (d) S1 và S0

Câu 3-13: Cờ tràn số có dấu của AT89S52 là:

(a) OV (b) RS1 và RS0

(c) C (d) Z

Câu 3-14: Cờ tràn số không dấu của AT89S52 là:

(a) OV (b) RS1 và RS0

(c) C (d) Z

Câu 3-15: AT89S52 có bao nhiêu thanh ghi con trỏ DPTR:

(a) 1 (b) 2

(c) 3 (d) 4

Câu 3-16: Vùng nhớ RAM nội của AT89S52 chia ra làm mấy vùng khác nhau:

(a) 1 (b) 2

(c) 3 (d) 4

Câu 3-17: Thanh ghi trạng thái của AT89S52 có tên là:

(a) FR (b) PSW

(c) SR (d) ACC

Câu 3-18: Các thanh ghi đặc biệt của AT89S52 có địa chỉ bắt đầu:

(a) 00H (b) 20H

(c) FFH (d) 80H

4. Bài tập

Chương 4
VI ĐIỀU KHIỂN AT89S52: LỆNH HỢP NGỮ

- **GIỚI THIỆU**
- **LỆNH HỢP NGỮ CỦA VI ĐIỀU KHIỂN MCS-52**
  - GIỚI THIỆU
  - CÁC KIỂU ĐỊNH ĐỊA CHỈ CỦA VI ĐIỀU KHIỂN MCS52
    - ✓ *Kiểu định địa chỉ dùng thanh ghi (Register Addressing):*
    - ✓ *Kiểu định địa chỉ trực tiếp (Direct Addressing):*
    - ✓ *Định địa chỉ gián tiếp (Indirect Addressing)*
    - ✓ *Định địa chỉ tức thời (Immediate Addressing)*
    - ✓ *Định địa chỉ tương đối*
    - ✓ *Định địa chỉ tuyệt đối (Absolute Addressing)*
    - ✓ *Định địa chỉ dài (Long Addressing)*
    - ✓ *Định địa chỉ chỉ số (Index Addressing)*
  - KHẢO SÁT TẬP LỆNH VI ĐIỀU KHIỂN MCS52
    - ✓ *Nhóm lệnh di chuyển dữ liệu (8 bit)*
    - ✓ *Nhóm lệnh số học*
    - ✓ *Nhóm lệnh logic*
    - ✓ *Nhóm lệnh chuyển quyền điều khiển*
    - ✓ *Nhóm lệnh xử lý bit*
- **CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP**
  - CÂU HỎI ÔN TẬP
  - CÂU HỎI TRẮC NGHIỆM
  - CÂU HỎI MỞ RỘNG
  - BÀI TẬP

I. GIỚI THIỆU

Ở chương này khảo sát tập lệnh hợp ngữ của vi điều khiển. Sau khi kết thúc chương này, bạn sẽ biết mã lệnh nhị phân, lệnh gợi nhớ, các kiểu định địa chỉ bộ nhớ của vi điều khiển, biết tập lệnh hợp ngữ của vi điều khiển

Vi điều khiển hay vi xử lý là các IC lập trình, khi bạn đã thiết kế hệ thống điều khiển có sử dụng vi xử lý hay vi điều khiển ví dụ như hệ thống điều khiển đèn giao thông cho một ngã tư gồm có các đèn Xanh, Vàng, Đỏ và các led 7 đoạn để hiển thị thời gian thì đó mới chỉ là phần cứng, muốn hệ thống vận hành thì bạn phải viết một chương trình điều khiển nạp vào bộ nhớ nội bên trong vi điều khiển hoặc bộ nhớ bên ngoài và gắn vào trong hệ thống để hệ thống vận hành và dĩ nhiên bạn phải viết đúng thì hệ thống mới vận hành đúng. Chương trình gọi là phần mềm.

Phần mềm và phần cứng có quan hệ với nhau, người lập trình phải hiểu rõ hoạt động của phần cứng để viết chương trình. Ở phần này sẽ trình bày chi tiết về tập lệnh của vi điều khiển giúp bạn hiểu rõ từng lệnh để bạn có thể lập trình được.

Các khái niệm về chương trình, lệnh, tập lệnh và ngôn ngữ gợi nhớ đã trình bày ở chương 1, ở đây chỉ tóm tắt lại.

Chương trình là một tập hợp các lệnh được tổ chức theo một trình tự hợp lí để giải quyết đúng các yêu cầu của người lập trình.

Người lập trình là người biết giải thuật để viết chương trình và sắp xếp đúng các lệnh theo giải thuật. Người lập trình phải biết chức năng tất cả các lệnh của vi điều khiển để viết chương trình.

Tất cả các lệnh có thể có của một ngôn ngữ lập trình còn gọi là ***tập lệnh***.

Lệnh của vi điều khiển là một số nhị phân 8 bit [còn gọi là mã máy]. 256 byte từ 0000 0000b đến 1111 1111b tương ứng với 256 lệnh khác nhau. Do mã lệnh dạng số nhị phân quá dài và khó nhớ nên các nhà lập trình đã xây dựng một ngôn ngữ lập trình Assembly cho dễ nhớ, điều này giúp cho việc lập trình được thực hiện một cách dễ dàng và nhanh chóng cũng như đọc hiểu và gỡ rối chương trình.

Khi viết chương trình bằng ngôn ngữ lập trình Assembly, vi điều khiển sẽ không thực hiện được mà phải dùng chương trình biên dịch Assembler để chuyển đổi các lệnh viết bằng Assembly ra mã lệnh nhị phân tương ứng rồi nạp vào bộ nhớ – khi đó vi điều khiển mới thực hiện được chương trình.

Ngôn ngữ lập trình Assembly do con người tạo ra, khi sử dụng ngôn ngữ Assembly để viết thì người lập trình vi điều khiển phải học hết tất cả các lệnh và viết đúng theo qui ước về cú pháp, trình tự sắp xếp dữ liệu để chương trình biên dịch có thể biên dịch đúng.

II. LỆNH HỢP NGỮ CỦA VI ĐIỀU KHIỂN MCS-52

1. Giới thiệu

Họ vi điều khiển MCS-52 đều có chung một tập lệnh, các vi điều khiển thế hệ sau chỉ phát triển nhiều về phần cứng còn lệnh thì ít mở rộng.

Tập lệnh họ MCS-52 có mã lệnh 8 bit nên có khả năng cung cấp 2^8= 256 lệnh.

Có lệnh có 1 hoặc 2 byte bởi dữ liệu hoặc địa chỉ thêm vào Opcode.

Trong toàn bộ tập lệnh của vi điều khiển có139 lệnh 1 byte, 92 lệnh 2 byte và 24 lệnh 3 byte.

2. Các kiểu định địa chỉ của vi điều khiển MCS52

Các kiểu định địa chỉ cho phép định rõ nơi lấy dữ liệu hoặc nơi nhận dữ liệu tùy thuộc vào cách thức sử dụng lệnh của người lập trình.

Vi điều khiển họ MCS-52 có 8 kiểu định địa chỉ như sau:

- Kiểu định địa chỉ dùng thanh ghi.
- Kiểu định địa chỉ trực tiếp.
- Kiểu định địa chỉ gián tiếp.
- Kiểu định địa chỉ tức thời.
- Kiểu định địa chỉ tương đối.
- Kiểu định địa chỉ tuyệt đối.
- Kiểu định địa chỉ dài.
- Kiểu định địa chỉ định vị.

➢ ***Kiểu định địa chỉ dùng thanh ghi (Register Addressing):***

Kiểu này thường được dùng cho các lệnh xử lý dữ liệu mà dữ liệu luôn lưu trong ***các thanh ghi***. Đối với vi điều khiển thì mã lệnh thuộc kiểu này chỉ có 1 byte.

Ví dụ 4-1: Mov A,R1; copy nội dung thanh ghi R1 vào thanh ghi A

➢ ***Kiểu định địa chỉ trực tiếp (Direct Addressing):***

Kiểu này thường được dùng để truy xuất dữ liệu của bất kỳ ô nhớ nào trong RAM nội.

Lệnh kiểu này có mã lệnh 2 byte: byte thứ nhất là mã lệnh, byte thứ hai là ***địa chỉ của ô nhớ.***

Ví dụ 4-2: Mov A, 05H; copy nội dung ô nhớ có địa chỉ 05H vào thanh ghi A

➢ ***Định địa chỉ gián tiếp (Indirect Addressing)***

Kiểu định địa chỉ gián tiếp dùng ký hiệu @ đặt trước các thanh ghi R0, R1 hay DPTR.

Hai thanh ghi R0 và R1 có thể hoạt động như một thanh ghi con trỏ, nội dung của thanh ghi cho biết địa chỉ của một ô nhớ trong RAM nội sẽ truy xuất.

Thanh ghi DPTR dùng để truy xuất ô nhớ ngoại. Các lệnh này chỉ có 1 byte.

Ví dụ 4-3: Mov A,@R1; copy nội dung ô nhớ có địa chỉ trong thanh ghi R1 vào A

➢ ***Định địa chỉ tức thời (Immediate Addressing)***

Kiểu định địa chỉ tức thời dùng ký hiệu # đặt trước một hằng số. Lệnh này dùng để nạp một hằng số ở byte thứ hai (hoặc byte thứ 3) vào thanh ghi hoặc ô nhớ.

Ví dụ 4-4: Mov A, #30H; nạp dữ liệu là con số 30H vào thanh ghi A

➢ ***Định địa chỉ tương đối***

Kiểu định địa chỉ tương đối chỉ sử dụng với những lệnh nhảy. Nơi nhảy đến có địa chỉ bằng địa chỉ đang lưu trong thanh ghi PC cộng với một giá trị 8 bit [còn gọi là giá trị lệch tương đối: relative offset] có giá trị từ – 128 đến +127 nên vi điều khiển có thể nhảy lùi [nếu số cộng với số âm] và nhảy tới [nếu số cộng với số dương]. Lệnh này có mã lệnh 2 byte, byte thứ hai chính là giá trị lệch tương đối.

Nơi nhảy đến thường được xác định bởi nhãn (label) và trình biên dịch sẽ tính toán giá trị lệch.

Định vị tương đối có ưu điểm là mã lệnh cố định khi thay đổi địa chỉ, nhưng khuyết điểm là chỉ nhảy ngắn trong phạm vi -128÷127 byte [256byte], nếu nơi nhảy đến xa hơn thì lệnh này không đáp ứng được – sẽ có lỗi.

Ví dụ 4-5: Sjmp X1; nhảy đến nhãn có tên là X1 nằm trong tầm vực 256 byte

➢ ***Định địa chỉ tuyệt đối (Absolute Addressing)***

Kiểu định địa chỉ tuyệt đối dùng với các lệnh ACALL và AJMP. Các lệnh này có mã lệnh 2 byte cho phép phân chia bộ nhớ theo trang - mỗi trang có kích thước đúng bằng 2kbyte so với giá trị chứa trong thanh ghi PC hiện hành. 11 bit địa chỉ A10÷A0 thay thế cho 11 địa chỉ thấp trong thanh ghi PC.

Định địa chỉ tuyệt đối có ưu điểm là mã lệnh ngắn (2 byte), nhưng khuyết điểm là mã lệnh thay đổi và giới hạn phạm vi nơi nhảy đến, gọi đến không quá 2 kbyte.

Ví dụ 4-6: Ajmp X1; nhảy đến nhãn có tên là X1 nằm trong tầm vực 2 kbyte

➢ ***Định địa chỉ dài (Long Addressing)***

Kiểu định địa chỉ dài dùng với lệnh LCALL và LJMP. Các lệnh này có mã lệnh 3 byte – trong đó có 2 byte (16bit) là địa chỉ của nơi đến. Cấu trúc mã lệnh là 3 byte như sau:

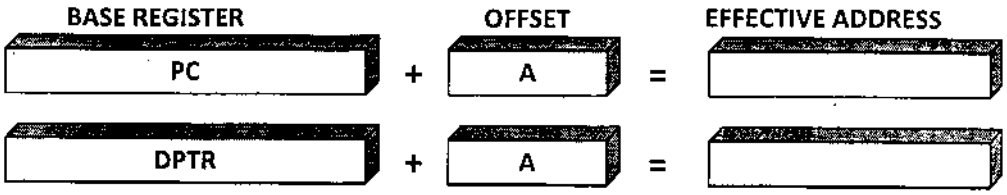
Ưu điểm của định địa chỉ dài là có thể gọi một chương trình con hoặc có thể nhảy đến bất kỳ vùng nhớ nào vùng nhớ 64K, nhược điểm là các lệnh kiểu này dài 3 byte và phụ thuộc vào vị trí đến – điều này sẽ bất tiện bởi không thể dời toàn bộ mã lệnh của chương trình từ vùng nhớ này sang các vùng nhớ khác – có nghĩa là khi chương trình đã viết nơi đến tại địa chỉ 1000h thì sau khi dịch ra mã lệnh dạng số nhị phân thì sau đó nạp vào bộ nhớ thì địa chỉ bắt đầu phải đúng với địa chỉ đã viết là 1000h; nếu nạp ở vùng địa chỉ khác địa chỉ 1000h thì chương trình sẽ thực hiện sai.

Ví dụ 4-7: Ljmp X1; nhảy đến nhãn có tên là X1 nằm trong tầm vực 64kbyte

➢ ***Định địa chỉ chỉ số (Index Addressing)***

Kiểu định địa chỉ chỉ số "dùng một thanh ghi cơ bản: là bộ đếm chương trình PC hoặc con trỏ dữ liệu DPTR" kết hợp với "một giá trị lệch (offset) còn gọi là giá trị tương đối [thường lưu trong thanh ghi]" để tạo ra địa chỉ của ô nhớ cần truy xuất hoặc là địa chỉ của nơi nhảy đến.

Việc kết hợp được minh họa như sau:



Ví dụ 4-8: MOVX A, @A + DPTR; lấy dữ liệu trong ô nhớ có địa chỉ bằng DPTR + A

Khi khảo sát tập lệnh một cách chi tiết thì chức năng của các thanh ghi và các kiểu truy xuất này sẽ được trình bày rõ ràng hơn.

3. Khảo sát tập lệnh vi điều khiển MCS52

Để khảo sát tập lệnh thì phải thống nhất một số qui định về các từ ngữ kí hiệu trong tập lệnh thường được sử dụng:

- Direct tượng trưng cho ô nhớ nội có địa chỉ bất kỳ từ 00H đến FFH.
- **Rn** tượng trưng cho các thanh ghi từ thanh ghi R0 đến thanh ghi R7.
- **@Ri** tượng trưng cho ô nhớ có địa chỉ lưu trong thanh ghi Ri và chỉ sử dụng hai thanh ghi: R0, R1.
- Các lệnh thường xảy ra giữa các đối tượng sau:
  - Thanh ghi A.
  - Thanh ghi Rn.
  - Ô nhớ có địa chỉ direct.
  - Ô nhớ có địa chỉ lưu trong thanh ghi @Ri.
  - Dữ liệu 8 bit **#data**.
- **Addr11** là địa chỉ 11 bit từ **A10÷A0**: địa chỉ này phục vụ cho lệnh nhảy hoặc lệnh gọi chương trình con trong phạm vi 2 kbyte.
- **Addr16** là địa chỉ 16 bit từ **A15÷A0**: địa chỉ này phục vụ cho lệnh nhảy và lệnh gọi chương trình con ở xa trong phạm vi 64 kbyte – đó chính là địa chỉ nhảy đến, hoặc địa chỉ của chương trình con.

Khi viết chương trình người lập trình có thể thay thế địa chỉ bằng nhãn (label) để khỏi phải tính toán các địa chỉ cụ thể. Nhãn (label) sẽ được đặt tại vị trí addr thay cho addr và phải có một nhãn đặt tại nơi muốn nhảy đến - gọi là một cặp nhãn cùng tên.

Có thể nhiều nơi nhảy đến cùng một nhãn. Không được đặt các nhãn cùng tên.

Các lệnh có ảnh hưởng đến thanh ghi trạng thái thì có trình bày trong lệnh, còn các lệnh không đề cập đến thanh ghi trạng thái thì có nghĩa là nó không ảnh hưởng.

➢ ***Nhóm lệnh di chuyển dữ liệu (8 bit)***

Bảng 4-1: *Tóm tắt cho nhóm lệnh di chuyển dữ liệu:*

CÚ PHÁP	MÃ LỆNH SỐ HEX	CHU KỲ	CỜ ẢNH HƯỞNG	CHỨC NĂNG
MOV A, Rn	E8 ÷ EF	1	Không	(A) ← (Rn)
MOV A, direct	E5 Byte 2	1	Không	(A) ← (direct)
MOV A,@RI	E6 ÷ E7	1	Không	(A) ← ((Ri))
MOV A, #data	74 Byte 2	1	Không	(A) ← #data
MOV Rn, A	F8 ÷ FF	1	Không	(Rn) ← (A)
MOV Rn, direct	A8 ÷ AF Byte 2	1	Không	(Rn) ← (direct)
MOV Rn #data	78 ÷ 7F Byte 2	1	Không	(Rn) ← #data
MOV direct, A	F5 Byte 2	1	Không	(direct) ← (A)
MOV direct, Rn	88 ÷ 8F Byte 2	2	Không	(direct) ← (Rn)
MOV direct, direct	85 Byte 2 Byte 3	2	Không	(direct) ← (direct) (source) (destination)
MOV direct,@Ri	86 ÷ 87 Byte 2	2	Không	(direct) ← ((Ri))

MOV direct, data	75 Byte 2 Byte 3	2	Không	(direct) ← #data
MOV @Ri, A	F6 ÷ F7	2	Không	((Ri)) ← (A)
MOV @Ri, direct	A6 ÷ A7 Byte 2	2	Không	((Ri)) ← (direct)
MOV @Ri, #data	76 ÷ 77 Byte 2	1	Không	((Ri)) ← (data)
MOV dptr,#data16	90 Byte 2 Byte 3	2	Không	(dptr) ← #data $_{15\text{-}0}$ (dpH) ← #data $_{15\text{-}8}$ (dpL) ← #data $_{7\text{-}0}$
MOVC A, @A + dptr	93	2	Không	(A) ← ((A)+(dptr)) bộ nhớ chương trình
MOVC A, @A + PC	83	2	Không	(A) ← ((A) + (PC)) bộ nhớ chương trình
MOVX A, @Ri	E2 ÷ E3	2	Không	(A)←((Ri)) bộ nhớ dữ liệu ngoài
MOVX A,@dptr	E0	2	Không	(A)←((dptr)) bộ nhớ dữ liệu ngoài
MOVX @Ri, A	F2 ÷ F3	2	Không	((Ri)) ← (A)
MOVX @ dptr, A	F0	2	Không	((dptr)) ← (A)
PUSH direct	C0 Byte 2	2	Không	(SP) ← (SP) + 1 ((SP)) ← (direct)
POP direct	D0 Byte 2	2	Không	(direct) ← ((SP)) (SP) ← (SP) - 1
XCH A, Rn	C8 ÷ CF	1	Không	(A) ↔ (Rn)
XCH A, direct	C5 Byte 2	1	Không	(A) ↔ (direct)
XCH A, @Ri	C6 ÷ C7	1	Không	(A) ↔ ((Ri))
XCHD A, @Ri	D6 ÷ D7	1	Không	($A_{4\text{-}0}$) ↔ (($Ri_{4\text{-}0}$))

❖ **Giải thích các lệnh:** do có nhiều lệnh với chức năng gần giống nhau nên chỉ giải thích một số lệnh.

➢ **Lệnh chuyển dữ liệu từ một thanh ghi vào thanh ghi A**

- Cú pháp: **MOV A, Rn**
- Mã lệnh:

1	1	1	0	1	n2	n1	n0

- **Chức năng:** copy nội dung thanh ghi Rn vào thanh ghi A.
- Do có 8 thanh ghi R0 đến R7 nên lệnh tổng quát này sẽ có 8 lệnh chi tiết cho 8 thanh ghi, khi sử dụng thanh ghi R0 thì cú pháp là "MOV A,R0" và mã lệnh là "11101111", tương tự cho các thanh ghi còn lại.

➢ **Lệnh chuyển dữ liệu từ ô nhớ trực tiếp vào thanh ghi A**

- Cú pháp: **MOV A, direct**
- Mã lệnh:

1	1	1	0	0	1	0	1
a7	a6	a5	a4	a3	a2	a1	a0

- **Chức năng:** copy nội dung của ô nhớ trong Ram nội có địa chỉ direct ở byte thứ hai vào thanh ghi A. Trực tiếp có nghĩa là địa chỉ của ô nhớ được ghi ở trong lệnh.

➢ **Lệnh chuyển dữ liệu từ ô nhớ gián tiếp vào thanh ghi A**

- Cú pháp: **MOV A,@RI**
- Mã lệnh 1 byte, thời gian thực hiện lệnh: 1 chu kỳ

1	1	1	0	0	1	1	i

- **Chức năng:** Copy nội dung ô nhớ trong Ram nội, có địa chỉ chứa trong thanh ghi Ri, vào thanh ghi A.

➢ **Lệnh nạp dữ liệu 8 bit vào thanh ghi A**

- Cú pháp: **MOV A, #data**
- Mã lệnh:

0	1	1	1	0	1	0	0
d7	d6	d5	d4	d3	d2	d1	d0

- **Chức năng:** Nạp dữ liệu 8 bit data (d0 đến d7) vào thanh ghi A.

➢ **Lệnh chuyển dữ liệu tức thời 16 bit vào thanh ghi con trỏ dữ liệu**

- Cú pháp: **MOV dptr, #data16**
- Mã lệnh:

1	0	0	1	0	0	0	0
d15	d14	d13	d12	d11	d10	d9	d8
d7	d6	d5	d4	d3	d2	d1	d0

- **Chức năng:** Nạp dữ liệu data 16 bit vào thanh ghi con trỏ dữ liệu dptr.

➢ **Lệnh chuyển dữ liệu từ ô nhớ có địa chỉ là Dptr + A vào thanh ghi A**

- Cú pháp: **MOVC A,@A+DPTR**
- Mã lệnh:

1	0	0	1	0	0	1	1

- **Chức năng**: chuyển nội dung của ô nhớ trong bộ nhớ chương trình (Code Memory), có địa chỉ chứa bằng dptr cộng với giá trị chứa trong A, chuyển vào thanh ghi A.

➢ **Lệnh chuyển dữ liệu từ ô nhớ có địa chỉ là PC + A vào thanh ghi A**

- Cú pháp: **MOVC A,@A+PC**
- Mã lệnh:

1	0	0	0	0	0	1	1

- Chức năng: chuyển nội dung của ô nhớ trong bộ nhớ chương trình (Code Memory) có địa chỉ chứa bằng PC cộng với giá trị chứa trong A được chuyển vào thanh ghi A.

➢ **Lệnh chuyển dữ liệu từ ô nhớ ngoài gián tiếp (8 bit địa chỉ) vào thanh ghi A**

- Cú pháp: **MOVX A, @Ri**
- Mã lệnh:

1	1	1	0	0	0	1	I

- **Chức năng:** copy nội dung ô nhớ ngoài có địa chỉ chứa trong thanh ghi Ri vào thanh ghi A.

➢ **Lệnh chuyển dữ liệu từ ô nhớ ngoài gián tiếp (16 bit địa chỉ) vào thanh ghi A**

- Cú pháp: **MOVX A,@DPTR**
- Mã lệnh:

1	1	1	0	0	0	0	0

- **Chức năng:** copy nội dung của ô nhớ ngoài có địa chỉ chứa trong thanh ghi dptr vào thanh ghi A.

➢ **Lệnh chuyển dữ liệu từ thanh ghi A vào ô nhớ ngoài gián tiếp (8 bit địa chỉ)**

- Cú pháp: **MOVX @Ri, A**
- Mã lệnh:

1	1	1	1	0	0	1	i

- **Chức năng:** chuyển nội dung của thanh ghi A ra ô nhớ ngoài có địa chỉ chứa trong thanh ghi Ri.

➢ **Lệnh chuyển dữ liệu từ thanh ghi A vào ô nhớ ngoài gián tiếp (16 bit địa chỉ)**

- Cú pháp: **MOVX @DPTR, A**
- Mã lệnh:

1	1	1	1	0	0	0	0

- **Chức năng:** Chuyển nội dung của thanh ghi A ra ô nhớ ngoài có địa chỉ chứa trong thanh ghi dptr.

➢ **Lệnh cất nội dung ô nhớ trực tiếp vào ngăn xếp**

- Cú pháp: **PUSH direct**
- Mã lệnh:

1	1	0	0	0	0	0	0
a7	a6	A5	a4	a3	a2	a1	a0

- **Chức năng:** cất nội dung của ô nhớ có địa chỉ direct vào ô nhớ ngăn xếp. Con trỏ ngăn xếp SP tăng lên 1 trước khi lưu nội dung.

➢ **Lệnh lấy dữ liệu từ ngăn xếp trả về ô nhớ trực tiếp**

- Cú pháp: **POP direct**
- Mã lệnh:

1	1	0	1	0	0	0	0
a7	a6	a5	a4	a3	a2	a1	a0

- **Chức năng:** lấy nội dung của ô nhớ ngăn xếp trả cho ô nhớ có địa chỉ direct. Con trỏ ngăn xếp SP giảm 1 sau khi lấy dữ liệu ra.

➢ **Lệnh trao đổi dữ liệu giữa thanh ghi với thanh ghi A**

- Cú pháp: **XCH A, Rn**
- Mã lệnh:

1	1	0	0	1	n2	n1	n0

- **Chức năng:** Trao đổi nội dung của thanh ghi Rn với thanh ghi A.

➢ **Lệnh trao đổi 4 bit dữ liệu giữa ô nhớ gián tiếp với thanh ghi A**

- Cú pháp: **XCHD Amie**
- Mã lệnh:

1	1	0	1	0	1	1	i

- **Chức năng:** Trao đổi dữ liệu 4 bit thấp của ô nhớ có địa chỉ chứa trong thanh ghi Ri với dữ liệu 4 bit thấp trong thanh ghi A.

➢ ***Nhóm lệnh số học***

Bảng 4-2: *Tóm tắt cho nhóm lệnh logic:*

CÚ PHÁP	MÃ LỆNH SỐ HEX	CHU KỲ	CỜ ẢNH HƯỞNG	CHỨC NĂNG
ADD A, Rn	28 ÷ 2F	1	Có	(A) ← (A) + (Rn)
ADD A, direct	25 Byte 2	1	Có	(A) ← (A) + (direct)
ADD A,@Ri	26 ÷ 27	1	Có	(A) ← (A) + ((Ri))
ADD A,#data	24 Byte 2	1	Có	(A) ← (A) + #data

ADDC A, Rn	38 ÷ 3F	1	Có	(A) ← (A) + (Rn) + (C)
ADDC A, direct	35 Byte 2	1	Có	(A) ← (A) + (direct) + (C)
ADDC A, @Ri	36 ÷ 37	1	Có	(A) ← (A) + ((Ri)) + (C)
ADDC A, #data	34 Byte 2	1	Có	(A) ← (A) + #data +(C)
SUBB A,Rn	98 ÷ 9F	1	Có	(A) ← (A) - (Rn) - (C)
SUBB A,direct	95 Byte 2	1	Có	(A) ← (A) - (direct) - (C)
SUBB A,@Ri	96 ÷ 97	1	Có	(A) ← (A) - ((Ri)) - (C)
SUBB A,#data	94 Byte 2	1	Có	(A) ← (A) - #data -(C)
INC A	04	1	Có	(A) ← (A) + 1
INC Rn	08 ÷ 0F	1	Có	(Rn) ← (Rn) + 1
INC direct	05 Byte 2	1	Có	(direct) ← (direct) + 1
INC @Ri	06 ÷ 07	1	Có	((Ri)) ← ((Ri)) + 1
INC dptr	A3	2	Có	(dptr) ← (dptr) + 1
DEC A	14	1	Có	(A) ← (A) - 1
DEC Rn	18 ÷ 1F	1	Có	(Rn) ← (Rn) - 1
DEC direct	15 Byte 2	1	Có	(direct) ← (direct) - 1
DEC @Ri	16 ÷ 17	1	Có	((Ri)) ← ((Ri)) - 1
MUL AB	A4	4	Có	$(B_{15-8}),(A_{7-0})$ ← (A)×(B)
DIV AB	84	4	Có	$(A_{15-8}),(B_{7-0})$ ← (A)/(B)
DA A	D4	4	Có	Nội dung A là BCD

❖ *Giải thích các lệnh:*

➢ **Lệnh cộng thanh ghi A với thanh ghi**

- Cú pháp: **ADD A,Rn**
- Mã lệnh:

0	0	1	0	1	n2	n1	n0

- **Chức năng:** cộng nội dung thanh ghi A với nội dung thanh ghi Rn, kết quả lưu trong thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

➢ **Lệnh cộng thanh ghi A với thanh ghi có bit carry**

- Cú pháp: **ADDC A,Rn**
- Mã lệnh:

0	0	1	1	1	n2	n1	n0

- **Chức năng:** cộng nội dung thanh ghi A với nội dung thanh ghi Rn với bit C, kết quả lưu trong thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

➢ **Lệnh trừ thanh ghi A với thanh ghi**

- Cú pháp: **SUBB A,Rn**
- Mã lệnh:

1	0	0	1	1	n2	n1	n0

- **Chức năng:** Trừ nội dung thanh ghi A với nội dung thanh ghi Rn và trừ cho cờ Carry, kết quả lưu trong thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

➢ **Lệnh tăng nội dung thanh ghi A**

- Cú pháp: **INC A (increment: tăng lên 1 đơn vị)**
- Mã lệnh:

0	0	0	0	0	1	0	0

- **Chức năng:** Tăng nội dung thanh ghi A lên 1.

➢ **Lệnh giảm nội dung thanh ghi A**

- Cú pháp: **DEC A**

- Mã lệnh:

0	0	0	1	0	1	0	0

- **Chức năng:** Giảm nội dung thanh ghi A xuống 1.

➢ **Lệnh nhân thanh ghi A với thanh ghi B**

- Cú pháp: **MUL AB**
- Mã lệnh:

1	0	1	0	0	1	0	0

- **Chức năng:** Nội dung của thanh ghi A nhân với nội dung của thanh ghi B, kết quả là một dữ liệu 16 bit, 8 bit thấp lưu trong thanh ghi A, 8 bit cao lưu trong thanh ghi B.

➢ **Lệnh chia thanh ghi A cho thanh ghi B**

- Cú pháp: **DIV AB**
- Mã lệnh:

1	0	0	0	0	1	0	0

- **Chức năng:** Nội dung của thanh ghi A chia cho nội dung của thanh ghi B, kết quả của phép chia lưu trong thanh ghi A, số dư lưu trong thanh ghi B. Lệnh ảnh hưởng đến thanh ghi trạng thái: Bit C và bit OV bị xóa về 0, nếu phép chia này mà dữ liệu trong thanh ghi B = 00H thì nội dung thanh ghi A không thay đổi, nội dung chứa trong thanh ghi B không xác định và bit OV = 1, bit Cy = 0.

➢ **Lệnh điều chỉnh thập phân nội dung thanh ghi A**

- Cú pháp: **DA A**
- Mã lệnh:

1	1	0	1	0	1	0	0

- **Chức năng:** Nếu 4 bit thấp A3A2A1A0>9 hoặc bit AC = 1thì A3A2A1A0 + 6, kết quả lưu trở lại trong A. Nếu 4 bit cao A7A6A5A4>9 hoặc bit Cy = 1 thì A7A6A5A4 + 6, kết quả lưu trở lại thanh ghi A. Kết quả sau cùng trong thanh ghi A là số BCD.

➢ ***Nhóm lệnh logic***

Bảng 4-3: *Tóm tắt cho nhóm lệnh logic:*

CÚ PHÁP	MÃ LỆNH SỐ HEX	CHU KỲ	CỜ ẢNH HƯỞNG	CHỨC NĂNG
ANL A,Rn	58 ÷ 5F	1		(A) ← (A) AND (Rn)
ANL A,direct	55 Byte 2	1		(A) ← (A) AND (direct)
ANL A,@Ri	56 ÷ 57	1		(A) ← (A) AND ((Ri))
ANL A,#data	54 Byte 2	1		(A) ← (A) AND #data
ANL direct, A	52 Byte 2	2		(direct)←(direct) and (A)
ANL direct, #data	53 Byte 2 Byte 3	2		(direct)←(direct) and #data
ORL A, Rn	48 ÷ 4F	1		(A) ← (A) OR (Rn)
ORL A, direct	45 Byte 2	1		(A) ← (A) OR (direct)
ORL A, @Ri	46 ÷ 47	1		(A) ← (A) OR ((Ri))
ORL A, #data	44 Byte 2	1		(A) ← (A) OR #data
ORL direct, A	42 Byte 2	2		(direct)←(direct) OR (A)
ORL direct, #data	43 Byte 2 Byte 3	2		(direct)←(direct) OR #data

XRL A, Rn	68 ÷ 6F	1		(A) ← (A) XOR (Rn)
XRL A, direct	65 Byte 2	1		(A) ← (A) XOR (direct)
XRL A, @Ri	66 ÷ 67	1		(A) ← (A) XOR ((Ri))
XRL A, #data	64 Byte 2	1		(A) ← (A) XOR #data
XRL direct, A	62 Byte 2	2		(direct)←(direct) XOR (A)
XRL direct, #data	63 Byte 2 Byte 3	2		(direct)←(direct) XOR #data
CLR A	E4	1		(A) ← 0
CPL A	F4	1		(A) ← ($\overline{A}$) lệnh not hay bù 1
RL A	23	1		Xoay nội dung thanh ghi A sang trái 1 bit
RLC A	33	1		Xoay nội dung thanh ghi A và cờ C sang trái 1 bit
RR A	03	1		Xoay nội dung thanh ghi A sang phải 1 bit
RRC A	13	1		Xoay nội dung thanh ghi A và cờ C sang phải 1 bit
SWAP A	C4	1		$(A_{4-0}) \leftrightarrow (A_{7-4})$

❖ **Giải thích các lệnh:**

➢ **Lệnh and thanh ghi A với thanh ghi**

- Cú pháp: **ANL A,Rn (and logic)**
- Mã lệnh:

0	1	0	1	1	n2	n1	n0

- **Chức năng:** Nội dung thanh ghi A and với nội dung thanh ghi Rn, kết quả lưu vào A.

➢ **Lệnh or thanh ghi A với thanh ghi**

- Cú pháp: **ORL A, Rn**
- Mã lệnh:

0	1	0	0	1	n2	n1	n0

- **Chức năng**: Nội dung thanh ghi A or với nội dung thanh ghi Rn, kết quả lưu vào A.

➢ **Lệnh ex-or thanh ghi A với thanh ghi**

- Cú pháp: **XRL A, Rn**
- Mã lệnh:

0	1	1	0	1	n2	n1	n0

- **Chức năng:** Nội dung thanh ghi A ex-or với nội dung thanh ghi Rn, kết quả lưu vào A.

➢ **Lệnh xóa nội dung thanh ghi A**

- Cú pháp: **CLR A (clear a)**
- Mã lệnh:

1	1	1	0	0	1	0	0

- **Chức năng:** Nội dung thanh ghi A bằng zero.

➢ **Lệnh bù nội dung thanh ghi A**

- Cú pháp: **CPL A (complement A)**
- Mã lệnh:

1	1	1	1	0	1	0	0

- **Chức năng:** Bù một nội dung thanh ghi A, kết quả chứa trong A.

➢ **Lệnh xoay trái nội dung thanh ghi A**

- Cú pháp: **RL A (rotate left)**
- Mã lệnh:

0	0	1	0	0	0	1	1

- **Chức năng:** Xoay nội dung thanh ghi A sang trái 1 bit minh họa như hình vẽ.

➢ **Lệnh xoay trái nội dung thanh ghi A và bit carry**

- Cú pháp: **RLC A**
- Mã lệnh:

0	0	1	1	0	0	1	1

- **Chức năng:** Xoay nội dung thanh ghi A và bit C sang trái 1 bit.

➢ **Lệnh xoay phải nội dung thanh ghi A**

- Cú pháp: **RR A** **(rotate right)**
- Mã lệnh:

0	0	0	0	0	0	1	1

- **Chức năng:** Xoay nội dung thanh ghi A sang phải 1 bit ngược với lệnh RL A.

➢ **Lệnh xoay phải nội dung thanh ghi A và bit carry**

- Cú pháp: **RRC A**
- Mã lệnh:

1	1	1	0	0	1	0	0

- **Chức năng:** Xoay nội dung thanh ghi A và bit C sang phải 1 bit ngược với lệnh RLC A.

➢ **Lệnh xoay thanh ghi A 4 bit**

- Cú pháp: **SWAP A**

- Mã lệnh:

1	1	0	0	0	1	0	0

- **Chức năng:** hoán chuyển 4 bit thấp và 4 bit cao trong thanh ghi A.

➢ ***Nhóm lệnh chuyển quyền điều khiển***

Nhóm lệnh này là nhóm lệnh chuyển quyền điều khiển có nghĩa là vi điều khiển đang thực hiện lệnh tại địa chỉ này thì có thể nhảy đến hoặc chuyển đến thực hiện lệnh tại một địa chỉ khác.

Trong nhóm này gồm có lệnh gọi chương trình con, lệnh kết thúc chương trình con trở về chương trình chính, lệnh nhảy không điều kiện và lệnh nhảy có điều kiện.

Các lệnh nhảy bao gồm lệnh nhảy tương đối, lệnh nhảy tuyệt đối, lệnh nhảy dài.

Các lệnh nhảy có điều kiện thì khi thỏa điều kiện thì lệnh sẽ nhảy còn nếu không thỏa điều kiện thì sẽ thực hiện lệnh kế ngay sau lệnh nhảy. Ở đây chỉ trình bày điều kiện thỏa còn điều kiện không thỏa thì ta hiểu ngầm.

***Bảng 4-4:** Tóm tắt cho nhóm lệnh logic:*

CÚ PHÁP	MÃ LỆNH SỐ HEX	CHU KỲ	CỜ ẢNH HƯỞNG	CHỨC NĂNG
ACALL addr 11	Byte 1 Byte 2	2		(PC) ← (PC) + 2, (SP) ← (SP) + 1 ((SP)) ← (PC_{7-0}), (SP) ← (SP) + 1 ((SP)) ← (PC_{15-8}), (PC) ← page address
LCALL addr 16	12 Byte 2 Byte 3	2		(PC) ← (PC) + 3, (SP) ← (SP) + 1 ((SP)) ← (PC_{7-0}), (SP) ← (SP) + 1 ((SP)) ← (PC_{15-8}), (PC) ← $addr_{15-0}$

RET	22	2		$(PC_{15-8}) \leftarrow ((SP))$, $(SP) \leftarrow (SP) - 1$ $(PC_{7-0}) \leftarrow ((SP))$, $(SP) \leftarrow (SP) - 1$
RETI	32	2		$(PC_{15-8}) \leftarrow ((SP))$, $(SP) \leftarrow (SP) - 1$ $(PC_{7-0}) \leftarrow ((SP))$, $(SP) \leftarrow (SP) - 1$
AJMP addr 11	Byte 1 Byte 2	2		$(PC) \leftarrow (PC) + 2$ $(PC) \leftarrow$ page address
LJMP addr 16	02 Byte 2 Byte 3	2		$(PC) \leftarrow addr_{15-0}$
SJMP rel	80 Byte 2	2		$(PC) \leftarrow (PC) + 2$ $(PC) \leftarrow (PC) +$ rel
JMP @A + dptr	73	2		$(PC) \leftarrow (A) +$ (dptr)
JZ rel	60 Byte 2	2	Có	$(PC) \leftarrow (PC) + 2$ IF (A) = 0 then $(PC) \leftarrow (PC) +$ rel
JNZ rel	70 Byte 2	2	Có	$(PC) \leftarrow (PC) + 2$ IF (A) $\neq$ 0 then $(PC) \leftarrow (PC) +$ rel
JC rel	40 Byte 2	2	Có	$(PC) \leftarrow (PC) + 2$ IF (C) = 1 then $(PC) \leftarrow (PC) +$ rel
JNC rel	50 Byte 2	2	Có	$(PC) \leftarrow (PC) + 2$ IF (C) = 0 then $(PC) \leftarrow (PC) +$ rel
JB bit, rel	20	2	Có	$(PC) \leftarrow (PC) + 3$

	Byte 2 Byte 3			IF (bit) =1 then (PC)←(PC) + rel
JNB bit, rel	30 Byte 2 Byte 3	2	Có	(PC) ← (PC) + 3 IF (bit) = 0 then (PC)←(PC) + rel
JBC bit, rel	10 Byte 2 Byte 3	2	Có	(PC) ← (PC) + 3 IF (bit)= 1 then (bit) ← 0 (PC)←(PC) + rel

CJNE A, direct, rel	B5
	Byte 2
	Byte 3
		2	Có	(PC) ← (PC) + 3
				IF (direct) < (A)
				then (C)←0 and
				(PC)←(PC) + rel
				IF (direct) > (A)
				then (C)←1 and
				(PC)←(PC) + rel

CJNE A, #data, rel	B4
	Byte 2
	Byte 3
		2	Có	(PC) ← (PC) + 3
				IF #data < (A) then
				(C)←0 and
				(PC)←(PC) + rel
				IF #data > (A) then
				(C)←1 and
				(PC)←(PC) + rel

CJNE Rn,#data, rel	B8 ÷ BF Byte 2 Byte 3	2	Có	(PC) ← (PC) + 3 IF #data < (Rn) then (C)←0 and (PC)←(PC) + rel IF #data > (Rn) then (C)←1 and (PC)←(PC) + rel
CJNE @Ri,#data, rel	B6 ÷ B7 Byte 2	2	Có	(PC) ← (PC) + 3 IF #data <((Ri))

	Byte 3			then (C)←0 and (PC)←(PC) + rel IF #data >((Ri)) then (C)←1 and (PC)←(PC) + rel
DJNZ Rn, rel	D8 ÷ DF Byte 2	2	Có	(PC) ← (PC) + 2 (Rn) ← (Rn) – 1 IF ((Rn)) ≠ 0 then (PC)←(PC) + rel
DJNZ direct, rel	D5 Byte 2 Byte 3	2	Có	(PC) ← (PC) + 3 (direct) ← (direct) – 1 IF (direct) ≠ 0 then (PC)←(PC) + rel
NOP	00	1	Không	(PC) ← (PC) + 1

❖ **Giải thích các lệnh:**

➢ **Lệnh gọi chương trình con dùng địa chỉ tuyệt đối**

- Cú pháp: **ACALL addr11**
- Mã lệnh:

a10	a9	a8	1	0	0	0	1
a7	a6	a5	a4	a3	a2	a1	a0

- **Chức năng:** Khi lệnh này được thực hiện, vi điều khiển sẽ thực hiện chương trình con tại địa chỉ addr11. Chương trình con không được cách lệnh gọi quá 2 kbyte. Addr11 của chương trình con có thể thay bằng nhãn (tên của chương trình con).
- ***Chú ý:*** Trước khi nạp địa chỉ mới vào thanh ghi PC, địa chỉ của lệnh kế trong chương trình chính được cất vào bộ nhớ ngăn xếp.

➢ **Lệnh gọi chương trình con dùng địa chỉ dài 16 bit**

- Cú pháp: **LCALL addr16**

- Mã lệnh:

0	0	0	1	0	0	1	0
a15	a14	a13	a12	a11	a10	a9	a8
a7	a6	a5	a4	a3	a2	a1	a0

- **Chức năng**: Khi lệnh này được thực hiện thì vi điều khiển sẽ thực hiện chương trình con tại địa chỉ addr16.
- 16 bit địa chỉ A15 – A0 được nạp vào PC, vi điều khiển sẽ thực hiện chương trình con tại địa chỉ vừa nạp vào PC. Chú ý: Trước khi nạp địa chỉ vào thanh ghi PC thì địa chỉ của lệnh kế trong chương trình chính được cất vào bộ nhớ ngăn xếp.

➢ **Lệnh trở về từ chương trình con**

- Cú pháp: **RET**
- Mã lệnh:

0	0	1	0	0	0	1	0

- **Chức năng:** Lệnh này sẽ kết thúc chương trình con, vi điều khiển sẽ trở lại chương trình chính để tiếp tục thực hiện chương trình.
- Chú ý: lệnh này sẽ lấy địa chỉ của lệnh kế đã lưu trong bộ nhớ ngăn xếp (khi thực hiện lệnh gọi) trả lại cho thanh ghi PC để tiếp tục thực hiện chương trình chính. ***Khi viết chương trình con thì phải luôn luôn kết thúc bằng lệnh ret.***

➢ **Lệnh trở về từ chương trình con phục vụ ngắt**

- Cú pháp: **RETI**
- Mã lệnh:

0	0	1	1	0	0	1	0

- **Chức năng:** Lệnh này sẽ kết thúc chương trình phục vụ ngắt, vi điều khiển sẽ trở lại chương trình chính để tiếp tục thực hiện chương trình.

➢ **Lệnh nhảy dùng địa chỉ tuyệt đối**

- Cú pháp: **AJMP addr11**

- Mã lệnh:

a10	a9	a8	0	0	0	0	1
a7	a6	a5	a4	A3	a2	a1	a0

- **Chức năng:** vi điều khiển sẽ nhảy đến địa chỉ addr11 để thực hiện chương trình tại đó. Addr11 có thể thay thế bằng nhãn. Nhãn hay địa chỉ nhảy đến không quá 2 kbyte.
- 11 bit địa chỉ A10 – A0 được nạp vào PC, các bit cao của PC không thay đổi, vi điều khiển sẽ nhảy đến thực hiện lệnh tại địa chỉ PC mới vừa nạp.
- Lệnh này khác với lệnh gọi chương trình con là không cất địa chỉ trở về. Nơi nhảy đến không quá 2 kbyte so với lệnh nhảy.

➢ **Lệnh nhảy dùng địa chỉ 16 bit**

- Cú pháp: **LJMP addr16**
- Mã lệnh:

0	0	0	0	0	0	1	0
a15	a14	a13	a12	a11	a10	a9	a8
a7	a6	a5	a4	a3	a2	a1	a0

- **Chức năng:** vi điều khiển sẽ nhảy đến địa chỉ addr16 để thực hiện chương trình tại đó. Nơi nhảy đến tùy ý nằm trong vùng 64 kbyte.

➢ **Lệnh nhảy tương đối**

- Cú pháp: **SJMP rel**
- Mã lệnh:

1	0	0	0	0	0	0	0
r7	r6	r5	r4	r3	r2	r1	r0

- **Chức năng:** vi điều khiển sẽ nhảy đến lệnh có địa chỉ tương đối (rel) để thực hiện tiếp. Có thể thay thế rel bằng nhãn.
- Lệnh này chỉ nhảy trong tầm vực 256 byte: có thể nhảy tới 128 byte và có thể nhảy lùi 128 byte. Khi tầm vực nhảy xa hơn ta nên dùng lệnh AJMP hay LJMP.

➢ **Lệnh nhảy gián tiếp**

- Cú pháp: **JMP @A + DPTR**
- Mã lệnh:

0	1	1	1	0	0	1	1

- **Chức năng:** lệnh sẽ nhảy đến nơi có địa chỉ bằng nội dung của A cộng với dptr.

➢ **Lệnh nhảy nếu cờ Z =1 (nội dung thanh ghi A bằng 0)**

- Cú pháp: **JZ rel (jump zero)**
- Mã lệnh:

0	1	1	0	0	0	0	0
r7	r6	r5	r4	r3	r2	r1	r0

- **Chức năng:** nhảy đến nhãn rel nếu cờ Z = 1.

➢ **Lệnh nhảy nếu cờ Z = 0 (nội dung thanh ghi A khác 0)**

- Cú pháp: **JNZ rel**

➢ **Lệnh nhảy nếu bit carry = 1**

- Cú pháp: **JC rel**
- Mã lệnh:

0	1	0	0	0	0	0	0
r7	r6	r5	r4	r3	r2	r1	r0

- **Chức năng:** nhảy đến nhãn rel nếu cờ C = 1.

➢ **Lệnh nhảy nếu bit carry = 0**

- Cú pháp: **JNC rel**

➢ **Lệnh nhảy nếu bit = 1**

- Cú pháp: **JB bit, rel**
- Mã lệnh:

0	0	1	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
r7	r6	r5	r4	r3	r2	r1	r0

- **Chức năng:** nhảy đến nhãn rel nếu nội dung của bit có địa chỉ bit [được xác định bởi byte thứ hai] bằng 1.

➢ **Lệnh nhảy nếu bit = 0**

- Cú pháp: **JNB bit, rel**
- Mã lệnh:

0	0	1	1	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
r7	r6	r5	r4	r3	r2	r1	r0

- Chức năng: nhảy đến nhãn rel nếu nội dung của bit có địa chỉ bit [được xác định bởi byte thứ hai] bằng 0.

➢ **Lệnh nhảy nếu bit = 1 và xóa bit**

- Cú pháp: **JBC bit, rel**
- Mã lệnh:

0	0	0	1	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
r7	r6	r5	r4	r3	r2	r1	r0

- **Chức năng:** nhảy đến nhãn rel nếu nội dung của bit có địa chỉ bit [được xác định bởi byte thứ hai] bằng 1 và xóa bit đó về 0.

➢ **Lệnh so sánh ô nhớ trực tiếp với nội dung thanh ghi A**

- Cú pháp: **CJNE A, direct, rel (compare jump if not equal)**
- Mã lệnh:

1	0	1	1	0	1	0	0
a7	a6	a5	a4	a3	a2	a1	a0
r7	r6	r5	r4	r3	r2	r1	r0

- **Chức năng:** lệnh này ảnh hưởng đến cờ C và thực hiện việc nhảy như sau:
  + Nếu nội dung của A ≥ nội dung của ô nhớ có địa chỉ direct thì bit C = 0.
  + Nếu nội dung của A < nội dung của ô nhớ có địa chỉ direct thì bit C = 1.

+ Nếu nội dung của A khác nội dung ô nhớ có địa chỉ direct thì lệnh sẽ nhảy đến rel.
+ Nếu nội dung của A bằng nội dung của ô nhớ có địa chỉ direct thì làm lệnh kế.

➢ **Lệnh giảm thanh ghi và nhảy**

- Cú pháp: **DJNZ Rn, rel (decrement and jump if not zero)**
- Mã lệnh:

1	1	0	1	1	n2	n1	n0
r7	r6	r5	r4	r3	r2	r1	r0

- **Chức năng:** Giảm nội dung thanh ghi Rn đi 1 và nhảy nếu Rn sau khi giảm khác 0.

➢ **Lệnh Nop**

- Cú pháp: **NOP**
- Mã lệnh:

0	0	0	0	0	0	0	0

- **Chức năng:** Nội dung của PC tăng lên 1 và tiếp tục thực hiện lệnh tiếp theo.

➢ ***Nhóm lệnh xử lý bit***

Bảng 4-5: *Tóm tắt cho nhóm lệnh logic:*

CÚ PHÁP	MÃ LỆNH SỐ HEX	CHU KỲ	CỜ ẢNH HƯỞNG	CHỨC NĂNG
CLR C	C3	1	Có	(C) ← 0
CLR bit	C2 Byte 2	1	Không	(bit) ← 0
SETB C	D3	1	Có	(C) ← 1
SETB bit	D2 Byte 2	1	Không	(bit) ← 1
CPL C	B3	1	Có	(C) ← ($\overline{C}$)

CPL bit	B2 Byte 2	1		(bit) ← ($\overline{bit}$)
ANL C,bit	82 Byte 2	1	Có	(C) ← (C) AND (bit)
ANL C,/bit	B0 Byte 2	1	Có	(C) ←(C) AND ($\overline{bit}$)
ORL C,bit	72 Byte 2	1	Có	(C) ← (C) OR (bit)
ORL C,/bit	A0 Byte 2	1	Có	(C) ←(C) OR ($\overline{bit}$)
MOV C,bit	A2 Byte 2	1	Có	(C) ← (bit)
MOV bit,C	92 Byte 2	1	Có	(bit) ← (C)

❖ **Giải thích các lệnh:**

➢ **Lệnh xóa bit carry**

- Cú pháp: **CLR C**

➢ **Lệnh xóa bit**

- Cú pháp: **CLR bit**

➢ **Lệnh đặt bit carry**

- Cú pháp: **SETB C**

➢ **Lệnh đặt bit**

- Cú pháp: **SETB bit**

➢ **Lệnh bù bit carry**

- Cú pháp: **CPL C**

➢ **Lệnh bù bit**

- Cú pháp: **CPL bit**

➢ **Lệnh and bit carry với bit**

- Cú pháp: **ANL C, bit**
- Mã lệnh:

1	0	0	0	0	0	1	0
b7	b6	b5	b4	b3	b2	b1	b0

- **Chức năng:** Bit C and với bit có địa chỉ được xác định bởi byte thứ hai, kết quả chứa ở bit C.

➢ **Lệnh and bit carry với bù bit**

- Cú pháp: **ANL C, /bit**
- Mã lệnh:

1	0	1	1	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0

- **Chức năng:** Bit C and với bù bit có địa chỉ được xác định bởi byte thứ hai, kết quả lưu vào C.

➢ **Lệnh or bit carry với bit**

- Cú pháp: **ORL C, bit**
- Mã lệnh:

0	1	1	1	0	0	1	0
b7	b6	b5	b4	b3	b2	b1	b0

- **Chức năng:** Bit C or với bit có địa chỉ được xác định bởi byte thứ hai, kết quả lưu vào C.

➢ **Lệnh or bit carry với bù bit**

- Cú pháp: **ORL C, /bit**
- Mã lệnh:

1	0	1	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0

- **Chức năng:** Bit C or với bù bit có địa chỉ được xác định bởi byte thứ hai, kết quả lưu vào C.

➢ **Lệnh di chuyển bit vào bit carry**

- Cú pháp: **MOV C, bit**
- Mã lệnh:

1	0	1	0	0	0	1	0
b7	b6	b5	b4	b3	b2	b1	b0

- **Chức năng:** Bit có địa chỉ được xác định bởi byte thứ hai được chuyển vào bit C.

➢ **Lệnh di chuyển bit carry vào bit**

- Cú pháp: **MOV bit, C**
- Mã lệnh:

1	0	0	1	0	0	1	0
b7	b6	b5	b4	b3	b2	b1	b0

- **Chức năng**: Bit C được chuyển vào bit có địa chỉ được xác định bởi byte thứ hai.

III. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP

1. Câu hỏi ôn tập

Câu số 4-1: Hãy cho biết các kiểu định địa chỉ của vi điều khiển AT89S52.

Câu số 4-2: Hãy cho biết các nhóm lệnh của vi điều khiển AT89S52.

2. Câu hỏi mở rộng

Câu số 4-3: Hãy tìm hiểu lệnh so sánh cấu trúc CISC và RISC.

Câu số 4-4: Hãy tìm cấu trúc đường ống lệnh.

3. Câu hỏi trắc nghiệm

Câu 4-1: Kí hiệu "Direct" trong tập lệnh AT89S52 là:

(a) Địa chỉ trực tiếp của ô nhớ RAM ngoại

(b) Địa chỉ trực tiếp của ô nhớ RAM nội

(c) Dữ liệu ô nhớ

(d) Địa chỉ trực tiếp của ô nhớ ROM nội

Câu 4-2: Kí hiệu "@Ri" trong tập lệnh AT89S52 là:

(a) Địa chỉ gián tiếp của ô nhớ RAM ngoại

(b) Địa chỉ gián tiếp của ô nhớ ROM nội

(c) Nội dung của thanh ghi Ri

(d) Địa chỉ gián tiếp của ô nhớ RAM nội

Câu 4-3: Kí hiệu "@Ri" trong tập lệnh AT89S52 chỉ dùng cho các thanh ghi:

(a) R0 đến R7 (b) R0, R1 (c) R0 đến R8 (d) R0, R1, R2

Câu 4-4: Kí hiệu "Rn" trong tập lệnh AT89S52 chỉ dùng cho các thanh ghi:

(a) R0 đến R7 (b) R0, R1 (c) R0 đến R8 (d) R0, R1, R2

Câu 4-5: Kí hiệu "#data" trong tập lệnh AT89S52 tượng trưng cho:

(a) Nội dung ô nhớ (b) Nội dung thanh ghi

(c) Dữ liệu (d) Địa chỉ

Câu 4-6: Trong tập lệnh AT89S52 thì lệnh nào nạp dữ liệu vào thanh ghi A:

(a) MOV A,R0 (b) MOV A,30H

(c) MOV A,#30H (d) MOV A,@R0

Câu 4-7: Trong tập lệnh AT89S52 thì lệnh nào cất dữ liệu vào bộ nhớ ngăn xếp:

(a) XCH A, R0 (b) POP 00H

(c) MOV SP,#20H (d) PUSH ACC

Câu 4-8: Trong tập lệnh AT89S52 thì lệnh nào lấy dữ liệu từ bộ nhớ ngăn xếp:

(a) XCH A, R0 (b) POP 00H

(c) MOV SP, #20H (d) PUSH ACC

Câu 4-9: Trong tập lệnh AT89S52 thì lệnh nào so sánh:

(a) DJNZ R0, REL (b) SJMP REL

(c) CJNE A,#DATA,REL (d) JNC REL

Câu 4-10: Trong tập lệnh AT89S52 thì lệnh nào giảm và nhảy nếu chưa bằng 0:

(a) DJNZ R0, REL (b) SJMP REL

(c) DEC R0, REL (d) JNC REL

4. Bài tập

Chương 5
VI ĐIỀU KHIỂN AT89S52: NGÔN NGỮ LẬP TRÌNH C

- **GIỚI THIỆU**
- **CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ C**
  - CÁC KIỂU DỮ LIỆU CỦA BIẾN
  - CÁC TOÁN TỬ
  - CÁC LỆNH C CƠ BẢN
  - CẤU TRÚC CỦA CHƯƠNG TRÌNH C
- **TRÌNH BIÊN DỊCH C51**
  - PHẦN MỞ RỘNG CỦA TRÌNH BIÊN DỊCH C51
  - KHAI BÁO BIẾN VÀ HẰNG SỐ
  - CÁC BIT CHỨC NĂNG ĐẶC BIỆT
  - ĐỊNH NGHĨA CÁC BIẾN
  - CON TRỎ DỮ LIỆU
  - KHAI BÁO MẢNG
  - KHAI BÁO CHƯƠNG TRÌNH CON PHỤC VỤ NGẮT
- **CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP**
  - CÂU HỎI ÔN TẬP
  - CÂU HỎI MỞ RỘNG
  - CÂU HỎI TRẮC NGHIỆM
  - BÀI TẬP

I. GIỚI THIỆU

Ngôn ngữ lập trình C là một ngôn ngữ lập trình được sử dụng phổ biến, là ngôn ngữ tạo mã hiệu quả, các phần tử lập trình có cấu trúc, và một tập hợp phong phú các toán tử.

Ngôn ngữ C một ngôn ngữ lập trình thuận tiện và hiệu quả, nhiều ứng dụng có thể được giải quyết dễ dàng hơn và hiệu quả hơn bằng ngôn ngữ C so với các ngôn ngữ chuyên biệt khác.

Ở chương này giới thiệu các ngôn ngữ lập trình C cho các loại vi điều khiển và các lệnh C cơ bản để phục vụ lập trình cho các ứng dụng. Do có nhiều họ vi điều khiển của nhiều hãng khác nhau nên các phần mềm lập trình C cho vi điều khiển cũng khác nhau, phần này chỉ trình bày những kiến thức lập trình C chung và cơ bản nhất và tùy thuộc vào từng phần mềm biên dịch mà các bạn tìm hiểu thêm.

Sau khi kết thúc phần này sẽ giúp các bạn biết cấu trúc một chương trình, biết các lệnh C cơ bản để lập trình, biết khai báo các kiểu dữ liệu cho các biến, biết viết chương trình.

II. CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ C

1. Các kiểu dữ liệu của biến

Trong chương trình thường khai báo biến để lưu dữ liệu và xử lý dữ liệu, tùy thuộc vào loại dữ liệu mà ta phải chọn loại dữ liệu cho phù hợp. Các biến của vi xử lý bao gồm bit, byte, word và long word tương ứng với dữ liệu 1 bit, 8 bit, 16 bit và 32 bit. Các kiểu dữ liệu cơ bản như sau:

***Bảng 5-1:** Các kiểu dữ liệu*

TT	Kiểu dữ liệu	Số bit	Giới hạn
1	bit	1	0÷1
2	signed char	8	-128÷127
3	unsigned char	8	0÷255
4	signed int	16	-32768÷32767
5	unsigned int	16	0÷65535
6	signed long	32	-2147483648 to 2147483647
7	unsigned long	32	0 to 4294967295
8	float	32	±1.175494E-38 to ±3.402823E+38

9	pointer	24/16/8	
10	sbit	1	0÷1
11	sfr	8	0÷255
12	Sfr16	16	0÷65535

***Ví dụ 5-1:** Khai báo các biến*

bit TT; //khai báo biến trạng thái thuộc kiểu dữ liệu bit.

unsigned char dem; //khai báo biến đếm (DEM) thuộc kiểu kí tự không dấu 8 bit.

sfr16 T2= 0xCC; //khai báo T2 là hai thanh ghi T2L và T2H nằm ở hai ô nhớ có địa chỉ liên tiếp là 0xCC và 0xCD.

2. Các toán tử

Các toán tử là thành phần quan trọng trong lập trình, để lập trình thì chúng ta cần phải hiểu rõ ràng chức năng của các loại toán tử.

***Bảng 5-2:** Các toán tử trong ngôn ngữ C bao gồm:*

TT	Toán tử	Chức năng	Ví dụ
1	+	Toán tử cộng	
2	+=	Toán tử cộng và gán.	x+=y tương đương với x=x+y
3	&=	Toán tử and và gán.	x&=y tương đương với x=x&y
4	&	Toán tử and	
5	^=	Toán tử ex-or và gán.	x^=y tương đương với x=x^y
6	^	Toán tử ex-or	
7	|=	Toán tử or và gán.	x|=y tương đương với x=x|y.
8	|	Toán tử or nhiều đại lượng với nhau thành 1.	Ví dụ or nhiều bit trong 1 byte với nhau
9	--	Giảm	
10	/=	Toán tử chia và gán.	x/=y tương đương với x=x/y
11	/	Toán tử chia	
12	==	Toán tử bằng dùng để so sánh	

13	>	Toán tử lớn hơn	
14	>=	Toán tử lớn hơn hay bằng	
15	++	Tăng	
16	*	***Toán tử truy xuất gián tiếp, đi trước con trỏ***	
17	!=	Toán tử không bằng	
18	<<=	Toán tử dịch trái và gán	x << = y tương đương với x=x<<y
19	<	Toán tử nhỏ hơn	
20	<<	Toán tử dịch trái	
21	<=	Toán tử nhỏ hơn hay bằng	
22	&&	Toán tử and	
23	!	Toán tử phủ định (not)	
24	||	Toán tử or	
25	%=	Toán tử chia lấy số dư và gán	x % = y tương đương với x=x%y
26	%	Toán tử module	
27	*=	Toán tử nhân và gán	x * = y tương đương với x=x*y
28	*	Toán tử nhân	
29	~	Toán tử bù 1	
30	>>=	Toán tử dịch phải và gán	x >> = y tương đương với x=x>>y
31	>>	Toán tử dịch phải	
32	->	Toán tử con trỏ cấu trúc	
33	-=	Toán tử trừ và gán	x - = y tương đương với x=x-y
34	-	Toán tử trừ	
35	sizeof	Xác định kích thước theo byte của toán tử	

➢ ***Toán tử gán (=)***

Dùng để gán một giá trị nào đó cho một biến

Ví dụ 5-2: A = 5; Gán biến A bằng 5

Ví dụ 5-3: A = 2+ (B = 5);

Có chức năng gán biến B bằng 5 rồi cộng với 2 và gán cho biến A, kết quả B = 5 và A = 7.

➢ ***Toán tử số học (+, -, *, /, %)***

Có 5 toán tử để thực hiện các phép toán cộng, trừ, nhân, chia và chia lấy phần dư.

Ví dụ 5-4: A = 24; B = A % 5;

Gán A bằng 24, B gán số dư của A chia cho 5, kết quả B bằng 4

Ví dụ 5-5:

```
A=123;    X = A%10;//X=3
A= A/10;          //A = 12
Y = A%10;         //Y=2
Z = A/10;         //Z=1
```

Ví dụ này có chức năng tách từng con số đơn vị, chục, trăm gán cho ba biến X, Y, Z.

Ví dụ 5-6:

```
A=1234;   X = A%10;//X=4
A= A/10;          //A = 123
Y = A%10;         //Y=3
A= A/10;          //A = 12
Z = A%10;         //Z=2
V = A/10;         //V=1
```

Ví dụ này có chức năng tách từng con số đơn vị, chục, trăm, ngàn gán cho 4 biến X, Y, Z, V.

➢ ***Toán tử gán phức hợp (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)***

Tổng quát cho toán tử gán phức hợp: biến += giá_trị tương đương với biến = biến + giá_trị.

Ví dụ 5-7:

```
A+=5;      tương đương với A = A +5;
A /= 5;    tương đương A = A / 5;
B *= X + 1; tương đương B = B * (X+1);
```

➢ ***Toán tử tăng và giảm (++, --)***

Tổng quát cho toán tử gán phức hợp: biến += giá_trị tương đương với biến = biến + giá_trị.

Ví dụ 5-8: A=5; Gán A bằng 5

A++; tương đương với A = A +1 hay A+=1; kết quả A bằng 6

Ví dụ 5-9: B = 3; gán B bằng 3

A=++B; kết quả A bằng 4, B bằng 4

Ví dụ 5-10: B = 3; gán B bằng 3

A=B++; kết quả A bằng B và bằng 3, B tăng lên 1 bằng 4

Sự khác nhau là "++" đặt trước thì tính trước rồi mới gán, đặt sau thì gán trước rồi mới tính.

➢ ***Toán tử quan hệ (==, !=, >, <, >=, <=)***

Các toán tử quan hệ dùng để so sánh các biểu thức, kết quả so sánh là đúng hoặc sai.

Các toán tử trên tương ứng là bằng, khác, lớn hơn, nhỏ hơn, lớn hơn hay bằng, nhỏ hơn hay bằng.

Ví dụ 5-11: if (X<100) X+=1;

Lệnh trên kiểm tra nếu X còn nhỏ hơn 100 thì tăng X lên 1.

➢ ***Toán tử logic các điều kiện (!, &&,\|\|)***

Các toán tử trên tương đương là NOT, AND và OR.

Ví dụ 5-12:

!true sẽ cho kết quả là false

((5==5) && (6>4)) and hai điều kiện lại với nhau và kết quả là true.

➢ ***Toán tử xử lý bit (&, \|, ^, ~, <<, >>)***

Các toán tử xử lý bit với bit, các toán tử trên tương đương là AND, OR, XOR, NOT, SHL (dịch trái), SHR (dịch phải).

Ví dụ 5-13:

A=0x77; //gán A = 0111 0111B

B=0XC9; //gán B = 1100 1001B

X = A & B; // X bằng A and với B, kết quả X = 0100 0001B = 0X41

Y = A | B; // Y bằng A or với B, kết quả Y = 1111 1111B = 0XFF

Z = A ^ B; // Z bằng A xor với B, kết quả Z = 1011 1110B = 0XBE

W = ~A; // W bằng not A, kết quả W = 1000 1000B = 0X88

Ví dụ 5-14:

A=0x01; //gán A = 0000 0001B

A= (A <<1); //dịch A sang trái 1 bit, kết quả A = 0000 0010B.

A= (A <<1); //dịch A sang trái 1 bit, kết quả A = 0000 0100B.

Khi dịch trái thì bit bên trái mất, bit 0 thêm vào bên phải.

Ví dụ 5-15:

A=0x81; //gán A = 1000 0001B

A= (A >>1); //dịch A sang phải 1 bit, kết quả A = 0100 0000B.

Khi dịch phải thì bit bên phải mất, bit 0 thêm vào bên trái.

Ví dụ 5-16:

A=0x00; //gán A = 0000 0000B

A= (A <<1) +0x01; //dịch A sang trái 1 bit rồi cộng với 1, kết quả A = 0000 0001B.

A= (A <<1) +0x01; //dịch A sang trái 1 bit rồi cộng với 1, kết quả A = 0000 0011B.

Khi dịch trái và cộng với 1 thì có chức năng đẩy số 1 thêm vào bên phải, với dữ liệu 8 bit thì sau 8 lần sẽ lấp đầy 8 bit 1.

Ví dụ 5-17:

A=0x00; //gán A = 0000 0000B

A= (A >>1) +0x80; //dịch A sang phải 1 bit rồi cộng với 1000 0000, kết quả A = 1000 0000B.

A= (A >>1) +0x80; //dịch A sang phải 1 bit rồi cộng với 1000 0000, kết quả A = 1100 0000B.

Khi dịch phải và cộng với 0x80 thì có chức năng đẩy số 1 thêm vào bên trái, với dữ liệu 8 bit thì sau 8 lần sẽ lấp đầy 8 bit 1.

Ví dụ 5-18:

unsigned char X, Y; // X, Y là biến 8 bit

```
unsigned int A;   // A là biến 16 bit
A=0x1234;
X=A;              //gán X=0X34 là gán byte thấp
Y=A>>8;           //dịch A sang phải 8 bit rồi gán Y, kết quả Y=0X12
```

có nghĩa là gán byte cao.

➢ ***Toán tử lấy kích thước chuỗi theo bye ()***

Ví dụ 5-19:

```
A = sizeof (charac);    kết quả là A sẽ chứa số byte của chuỗi charac
```

3. Các lệnh C cơ bản

Thành phần quan trọng thứ ba trong lập trình C là các lệnh của ngôn ngữ C, phần tiếp theo sẽ khảo sát các lệnh cơ bản.

➢ ***Lệnh if và else:***

Chức năng: kiểm tra điều kiện nếu thỏa thì làm.

Cú pháp:

```
if (điều_kiện)
        {   Lệnh a1;
            Lệnh a2;
            ...
        }
else
        {   Lệnh b1;
            Lệnh b2;
            ...
        }
```

Ví dụ 5-20:

```
if (x==50)
    x=1;
else
    x=x+1;
```

➢ ***Lệnh lặp while:***

Chức năng: lặp lại một thao tác với một số lần nhất định hoặc khi còn thỏa một điều kiện nào đó.

Cú pháp: while (điều_kiện)

```
{   Lệnh 1;
    Lệnh 2;
    ...
}
```

Ví dụ 5-21:

```
while (x > 0)
    {x = x - 1 ;}
```

➢ ***Lệnh lặp do while:***

Chức năng: làm các lệnh trong dấu ngoặc và thoát nếu điều kiện theo sau lệnh while không đúng.

Cú pháp: do

```
{   Lệnh 1;
    Lệnh 2;
    ...
}
while (điều_kiện)
```

Ví dụ 5-22:

```
do
    {x=x+10;}
while (x < 100)
```

Thực hiện lệnh x bằng x cộng với 10, làm cho đến khi x bằng hay lớn hơn 100.

➢ ***Lệnh lặp for:***

Chức năng: làm các lệnh trong dấu ngoặc một số lần nhất định.

Cú pháp: for (giá_trị_bắt_đầu; điều_kiện_kết_thúc; tăng_giá_trị)

```
{   Lệnh 1;
    Lệnh 2;
    ...
}
```

Ví dụ 5-23:

```
for (int n = 0; n <100; n++)
    {x=x+10;}
```

Vòng lặp thực hiện với biến n bằng 0 cho đến khi n bằng 100 thì ngừng.

➢ ***Lệnh rẽ nhánh break:***

Chức năng: Lệnh này dùng để thoát khỏi vòng lặp dù cho điều kiện để kết thúc chưa thỏa mãn.

➢ ***Lệnh continue:***

Chức năng: Lệnh này dùng để kết thúc phần còn lại của vòng lặp để làm lần lặp tiếp theo.

➢ ***Lệnh rẽ nhánh goto:***

Chức năng: Lệnh này dùng để nhảy đến nhãn trong chương trình.

➢ ***Lệnh switch case:***

Chức năng: thực hiện một công việc tùy thuộc vào hàm.

Cú pháp: switch (cmd)

```
{
    Case constant1:  Lệnh a1;
                     Lệnh a2;
                     ...
                     Break;

    Case constant2:  Lệnh b1;
                     Lệnh b2;
                     ...
                     Break;
```

```
        Default:    Lệnh c1;
                    Lệnh c2;
                    ...
                    Break;
    }
```

Ví dụ 5-24:

```
switch (cmd)
    {
        Case 0:     printf("cmd 0");
                    Break;

        Case 1:     printf("cmd 1");
                    Break;

        Default:    printf("bad cmd ");
                    Break;
    }
```

III. TRÌNH BIÊN DỊCH C51

Trình biên dịch C51 dùng để biên dịch tập tin mã nguồn C cho họ vi điều 8051, để lập trình C chúng ta có thể sử dụng phần mềm biên dịch Keil-C (trong Keil-C sử dụng C51 để biên dịch), cách cài đặt, lập trình, biên dịch và sử dụng bạn có thể tham khảo ở tài liệu thực hành vi điều khiển.

Với những kiến thức cơ bản đã trình bày thì bạn có thể lập trình được cho vi điều khiển, tuy nhiên do mỗi vi điều khiển khác nhau nên ta cần phải tìm hiểu thêm các tính năng khác như phần trình bày tiếp theo.

1. Phần mở rộng của trình biên dịch C51

Trình biên dịch C51 cung cấp một số phần mở rộng cho ngôn ngữ C chuẩn ANSI. Các phần mở rộng này cung cấp hỗ trợ trực tiếp cho các thành phần của kiến trúc vi điều khiển 8051 bao gồm:

- Các vùng nhớ của vi điều khiển 8051
- Các kiểu mô hình bộ nhớ
- Các chỉ dẫn loại bộ nhớ
- Các chỉ dẫn loại biến dữ liệu bộ nhớ
- Các biến thuộc dữ liệu bit và dữ liệu cho phép truy xuất bit
- Các thanh ghi đặc biệt
- Con trỏ
- Các thuộc tính hàm

➢ ***Từ khóa:***

Để tránh lỗi phát sinh trong lập trình nên phần mềm C51 cung cấp các từ khóa như sau:

_at_	**idata**	**sfr**
alien	**interrupt**	**sfr16**
bdata	**large**	**small**
bit	**pdata**	**_task_**
code	**_priority_**	**using**
compact	**reentrant**	**xdata**
data	**sbit**	

➢ ***Bộ nhớ chương trình ROM:***

Bộ nhớ này chỉ dùng để lưu chương trình, không có chức năng lưu dữ liệu, ngoại trừ các hằng số thì cho phép lưu bằng các khai báo sau trong chương trình:

```
unsigned char code constant_1 =   0x03;
unsigned char code array_1[3] =   {'1', '2', '3'};
```

➢ ***Bộ nhớ chương trình SMALL:***

Bộ nhớ này có kích thước khoảng 2kbyte, dùng để khi biên dịch thì các lệnh CALL và lệnh JMP sẽ được dịch thành lệnh ACALL và AJMP để có byte mã lệnh ít hơn và chạy nhanh hơn.

➢ ***Bộ nhớ chương trình COMPACT:***

Bộ nhớ này có kích thước khoảng 64kbyte, dùng để khi biên dịch thì lệnh CALL sẽ được dịch thành lệnh LCALL, còn lệnh JMP thì vẫn là AJMP.

➢ ***Bộ nhớ chương trình LARGE:***

Bộ nhớ này có kích thước khoảng 64kbyte, dùng để khi biên dịch thì các lệnh CALL và lệnh JMP sẽ được dịch thành lệnh LCALL và LJMP để cho phép gọi và nhảy xa hơn.

➢ ***Bộ nhớ dữ liệu RAM SMALL:***

Bộ nhớ này nằm trong vi điều khiển nên truy xuất nhanh nhất. Tất cả các biến đều lưu trong bộ nhớ dữ liệu, tuy nhiên bộ nhớ này có dung lượng giới hạn, chỉ có 256 byte bao gồm luôn bank thanh ghi và vùng nhớ ngăn xếp.

Trong 256 byte thì 128 byte đầu tiên là cho phép truy xuất trực tiếp, kiểu này có tên là DATA, 128 byte cao còn lại thì chỉ cho phép truy xuất gián tiếp, kiểu này có tên là IDATA

➢ ***Bộ nhớ dữ liệu RAM COMPACT:***

Bộ nhớ này là bộ nhớ dữ liệu ngoại, kiểu bộ nhớ này chỉ có 256 byte dùng port0 để giao tiếp. Kiểu này có tên là PDATA. Khi truy xuất thì dùng hai thanh ghi là R0 và R1 để lưu địa chỉ.

➢ ***Bộ nhớ dữ liệu RAM LARGE:***

Bộ nhớ này là bộ nhớ dữ liệu ngoại, kiểu bộ nhớ này chỉ có dùng lượng 2^{16} byte dùng port0 và port2 để giao tiếp. Kiểu này cũng có tên là XDATA. Khi truy xuất thì dùng thanh ghi DPTR để lưu địa chỉ 16 bit.

2. Khai báo biến và hằng số

Ví dụ 5-25: *Khai báo các hằng trong bộ nhớ chương trình*

```
unsigned char code      dem =       0x03;
unsigned char code ma_ascii[10] = {'0','1', '2', '3','4','5', '6', '7', '8', '9'};
```

Ví dụ 5-26: *Khai báo biến trong bộ nhớ dữ liệu RAM bên ngoài*

```
#pragma   SMALL
unsigned char xdata speed ;
```

signed char xdata big_array[200];

Ví dụ 5-27: *Khai báo biến và khởi gán giá trị cho biến*

unsigned int speed = 3;

3. Các BIT chức năng đặc biệt

Trong vi điều khiển có vùng nhớ cho phép truy xuất từng bit và cho phép truy xuất trực tiếp và vùng nhớ này có tên là BDATA.

Ví dụ 5-28: *Khai báo các biến byte và bit*

char bdata test; //khai báo biến test là kí tự byte thuộc vùng dữ liệu cho phép truy xuất bit.

sbit sign = test^7;//khai báo biến sign là bit thứ 7 của biến test.

sfr P1 = 0x90; //gán P1 là ô nhớ có địa chỉ 0x90.

4. Định nghĩa các biến

Ví dụ 5-29: *định nghĩa các biến*

#define HANG P0 ; // định nghĩa biến HANG là port0 của vi điều khiển AT89xx

sbit HANG_0 HANG^0; // định nghĩa bit HANG_0 là bit thứ 0 của vi điều khiển AT89xx

5. Con trỏ dữ liệu

Con trỏ dữ liệu trong ngôn ngữ C chính là kiểu truy xuất gián tiếp dùng thanh ghi, địa chỉ của ô nhớ cần truy xuất lưu trong thanh ghi.

Ví dụ 5-30: *khai báo con trỏ, gán địa chỉ con trỏ và lưu dữ liệu:*

unsigned char *pointer0 ; //khai báo con trỏ

pointer0 = mem_address; //gán địa chỉ bắt đầu của vùng nhớ cho con trỏ

*pointer0 = 0xFF; //lưu giá trị 0xFF vào ô nhớ có địa chỉ lưu trong con trỏ.

Đối với vùng nhớ ngoại thì xem ví dụ sau

Ví dụ 5-31: *khai báo con trỏ, gán địa chỉ con trỏ và lưu dữ liệu:*

unsigned char *asb_pointer ; //khai báo con trỏ

asb_pointer = (char*) 0x8000; //gán địa chỉ bắt đầu của vùng nhớ là 0x8000 cho con trỏ

* asb_pointer = 0xFF; //lưu giá trị 0xFF vào ô nhớ có địa chỉ lưu trong con trỏ.

* asb_pointer++; //tăng địa chỉ con trỏ đến ô nhớ kế

6. Khai báo mảng

Ví dụ 5-32: *khai báo mảng:*

unsigned char ma7doan[] = {0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90} ;

char chuoi_hien_thi = { "HELLO ! "} ;

7. Khai báo chương trình con phục vụ ngắt

Ví dụ 5-33: *khai báo chương trình con phục vụ ngắt*

```
Timer0_int()      interrupt 1 using 2
{
}
```

Khai báo tên chương trình con phục vụ ngắt và sử dụng ngắt thứ nhất đúng với ngắt của timer0 và dùng bank thanh ghi thứ hai cho chương trình con phục vụ ngắt.

8. Cấu trúc của chương trình C

Tất cả các chương trình viết bằng ngôn ngữ C bao gồm các chỉ dẫn, các khai báo biến, các định nghĩa biến, các biểu thức, các lệnh và hàm.

Chương trình C đơn giản như sau

```
/* my first C program*/
#include <AT89X52.h>
unsigned char MA7D[10] =
{0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90};
int GIAY,CHUC,DONVI;
void DELAY_TIMER0()
{     int BDN;
```

```
        TR0 = 1;
        for (BDN = 0; BDN<20;BDN++)
        {       do      {}
                while(TF0==0);
                TF0 = 0;TH0 = 0X3C;TL0 = 0XB0;
        }
        TR0 = 0;
}
void MAIN ()
{
        TMOD = T0_M0_ ;     TH0 = 0X3C;     TL0 = 0XB0;
 while(1)
                {
                for (GIAY = 0; GIAY <60; GIAY ++)
                        {       CHUC = GIAY /10;            DONVI = GIAY %10;
                                P0 = MA7D[DONVI];           P1 = MA7D[CHUC];
                                DELAY_TIMER0();
                        }
                }
}
```

"/* my first C program*/" là chú thích cho chương trình nằm trong cặp dấu "/* .... */" hoặc nằm sau "//".

Lệnh "#include < AT89X52.h >" có chức năng báo cho trình biên dịch biết chương trình dùng các thành phần của vi điều khiển có trong file "AT89X52.h".

Tiếp theo là khai báo biến toàn cục, chương trình con và các lệnh của chương trình con nằm trong dấu { }.

"void main ()" cho biết bắt đầu chương trình chính và các lệnh tiếp theo của chương trình chính nằm trong cặp dấu { }.

9. Các thành phần của chương trình C

➢ ***Chỉ dẫn tiền xử lý***

Có 2 chỉ dẫn là #define và #include.

Chỉ dẫn #define có chức năng thay thế một đoạn chuỗi bằng một chỉ định đặc biệt nào đó.

Chỉ dẫn # include có chức năng chèn vào một đoạn lệnh từ một file khác vào trong chương trình.

Ví dụ 5-34:

```
#define portled    P1;
#include <AT89X52.H>;
```

Chỉ dẫn thứ nhất có chức năng định nghĩa biến portled có giá trị là port1

Chỉ dẫn thứ hai có chức năng khai báo chương trình thư viện của vi điều khiển AT89X52.H

➢ ***Khai báo***

Khai báo biến bao gồm tên và thuộc tính, khai báo hàm, khai báo hằng. Khai báo biến toàn cục nằm bên ngoài hàm, khai báo biến cục bộ thì nằm bên trong hàm.

➢ ***Định nghĩa giá trị cho biến***

Dùng để thiết lập giá trị cho một biến hoặc một hàm.

➢ ***Biểu thức***

Là sự kết hợp của toán tử và tác tố để cho ra một kết quả duy nhất.

➢ ***Hàm***

Một hàm bao gồm các khai báo biến, định nghĩa giá trị, các biểu thức và các lệnh để thực hiện một chức năng đặc biệt nào đó.

10. File thư viện cho họ AT89X52

Khi dùng phần mềm Keil lập trình cho vi điều khiển AT89S52 thì phải khai báo thư viện <AT89X52.h>. Trong file này đã định nghĩa tên các thành phần của vi điều khiển họ AT89X52, nội dung của file như sau:

```
/*-------------------------------------------------------------------------

AT89X52.H
```

```
Header file for the low voltage Flash Atmel AT89C52 and AT89LV52.
Copyright (c) 1988-2002 Keil Elektronik GmbH and Keil Software, Inc.
All rights reserved.
------------------------------------------------------------------------*/
#ifndef __AT89X52_H__
#define __AT89X52_H__
/*-------------------------------------------
Byte Registers
-------------------------------------------*/
sfr     P0      =     0x80;
sfr     SP      =     0x81;
sfr     DPL     =     0x82;
sfr     DPH     =     0x83;
sfr     PCON    =     0x87;
sfr     TCON    =     0x88;
sfr     TMOD    =     0x89;
sfr     TL0     =     0x8A;
sfr     TL1     =     0x8B;
sfr     TH0     =     0x8C;
sfr     TH1     =     0x8D;
sfr     P1      =     0x90;
sfr     SCON    =     0x98;
sfr     SBUF    =     0x99;
sfr     P2      =     0xA0;
sfr     IE      =     0xA8;
sfr     P3      =     0xB0;
sfr     IP      =     0xB8;
sfr     T2CON   =     0xC8;
```

```
sfr     T2MOD   =   0xC9;
sfr     RCAP2L  =   0xCA;
sfr     RCAP2H  =   0xCB;
sfr     TL2     =   0xCC;
sfr     TH2     =   0xCD;
sfr     PSW     =   0xD0;
sfr     ACC     =   0xE0;
sfr     B       =   0xF0;
/*-------------------------------------------
P0 Bit Registers
-------------------------------------------*/
sbit    P0_0    =   0x80;
sbit    P0_1    =   0x81;
sbit    P0_2    =   0x82;
sbit    P0_3    =   0x83;
sbit    P0_4    =   0x84;
sbit    P0_5    =   0x85;
sbit    P0_6    =   0x86;
sbit    P0_7    =   0x87;
/*-------------------------------------------
PCON Bit Values
-------------------------------------------*/
#define IDL_        0x01
#define STOP_       0x02
#define PD_         0x02 /* Alternate definition */
#define GF0_        0x04
#define GF1_        0x08
#define SMOD_       0x80
```

```
/*-------------------------------------------
TCON Bit Registers
-------------------------------------------*/
sbit    IT0     =   0x88;
sbit    IE0     =   0x89;
sbit    IT1     =   0x8A;
sbit    IE1     =   0x8B;
sbit    TR0     =   0x8C;
sbit    TF0     =   0x8D;
sbit    TR1     =   0x8E;
sbit    TF1     =   0x8F;
/*-------------------------------------------
TMOD Bit Values
-------------------------------------------*/
#define T0_M0_      0x01
#define T0_M1_      0x02
#define T0_CT_      0x04
#define T0_GATE_    0x08
#define T1_M0_      0x10
#define T1_M1_      0x20
#define T1_CT_      0x40
#define T1_GATE_    0x80

#define T1_MASK_    0xF0
#define T0_MASK_    0x0F

/*-------------------------------------------
P1 Bit Registers
```

```
-----------------------------------------------*/
sbit    P1_0    =    0x90;
sbit    P1_1    =    0x91;
sbit    P1_2    =    0x92;
sbit    P1_3    =    0x93;
sbit    P1_4    =    0x94;
sbit    P1_5    =    0x95;
sbit    P1_6    =    0x96;
sbit    P1_7    =    0x97;

sbit  T2   =    0x90; /* External input to Timer/Counter 2, clock out */
sbit T2EX  = 0x91; /* Timer/Counter 2 capture/reload trigger & dir ctl */
/*-----------------------------------------------
SCON Bit Registers
-----------------------------------------------*/
sbit    RI          =       0x98;
sbit    TI          =       0x99;
sbit    RB8         =       0x9A;
sbit    TB8         =       0x9B;
sbit    REN         =       0x9C;
sbit    SM2         =       0x9D;
sbit    SM1         =       0x9E;
sbit    SM0         =       0x9F;
/*-----------------------------------------------
P2 Bit Registers
-----------------------------------------------*/
sbit    P2_0        =       0xA0;
sbit    P2_1        =       0xA1;
```

```
sbit    P2_2    =    0xA2;
sbit    P2_3    =    0xA3;
sbit    P2_4    =    0xA4;
sbit    P2_5    =    0xA5;
sbit    P2_6    =    0xA6;
sbit    P2_7    =    0xA7;
/*-------------------------------------------
IE Bit Registers
-------------------------------------------*/
sbit    EX0     =    0xA8; /* 1=Enable External interrupt 0 */
sbit    ET0     =    0xA9; /* 1=Enable Timer 0 interrupt */
sbit    EX1     =    0xAA; /* 1=Enable External interrupt 1 */
sbit    ET1     =    0xAB; /* 1=Enable Timer 1 interrupt */
sbit    ES      =    0xAC; /* 1=Enable Serial port interrupt */
sbit    ET2     =    0xAD; /* 1=Enable Timer 2 interrupt */
sbit    EA      =    0xAF; /* 0=Disable all interrupts */
/*-------------------------------------------
P3 Bit Registers (Mnemonics & Ports)
-------------------------------------------*/
sbit    P3_0    =    0xB0;
sbit    P3_1    =    0xB1;
sbit    P3_2    =    0xB2;
sbit    P3_3    =    0xB3;
sbit    P3_4    =    0xB4;
sbit    P3_5    =    0xB5;
sbit    P3_6    =    0xB6;
sbit    P3_7    =    0xB7;
```

```
sbit    RXD     =   0xB0; /* Serial data input */
sbit    TXD     =   0xB1; /* Serial data output */
sbit    INT0    =   0xB2; /* External interrupt 0 */
sbit    INT1    =   0xB3; /* External interrupt 1 */
sbit    T0      =   0xB4; /* Timer 0 external input */
sbit    T1      =   0xB5; /* Timer 1 external input */
sbit    WR      =   0xB6; /* External data memory write strobe
*/
sbit    RD      =   0xB7; /* External data memory read strobe
*/
/*-----------------------------------------------
IP Bit Registers
-----------------------------------------------*/
sbit    PX0     =   0xB8;
sbit    PT0     =   0xB9;
sbit    PX1     =   0xBA;
sbit    PT1     =   0xBB;
sbit    PS      =   0xBC;
sbit    PT2     =   0xBD;
/*-----------------------------------------------
T2CON Bit Registers
-----------------------------------------------*/
sbit    CP_RL2  =   0xC8; /* 0=Reload, 1=Capture select */
sbit    C_T2    =   0xC9; /* 0=Timer, 1=Counter */
sbit    TR2     =   0xCA; /* 0=Stop timer, 1=Start timer */
sbit    EXEN2   =   0xCB; /* Timer 2 external enable */
sbit    TCLK    =   0xCC; /* 0=Serial clock uses Timer 1
overflow, 1=Timer 2 */
```

```
sbit    RCLK    =    0xCD; /* 0=Serial clock uses Timer 1
overflow, 1=Timer 2 */
sbit    EXF2    =    0xCE; /* Timer 2 external flag */
sbit    TF2     =    0xCF; /* Timer 2 overflow flag */
/*-----------------------------------------------
T2MOD Bit Values
-----------------------------------------------*/
#define  DCEN_  0x01 /* 1=Timer 2 can be configured as up/down counter
*/
#define  T2OE_          0x02 /* Timer 2 output enable */
/*-----------------------------------------------
PSW Bit Registers
-----------------------------------------------*/
sbit    P       =    0xD0;
sbit    F1      =    0xD1;
sbit    OV      =    0xD2;
sbit    RS0     =    0xD3;
sbit    RS1     =    0xD4;
sbit    F0      =    0xD5;
sbit    AC      =    0xD6;
sbit    CY      =    0xD7;
/*-----------------------------------------------
Interrupt Vectors:
Interrupt Address = (Number * 8) + 3
-----------------------------------------------*/
#define  IE0_VECTOR     0 /* 0x03 External Interrupt 0 */
#define  TF0_VECTOR     1 /* 0x0B Timer 0 */
#define  IE1_VECTOR     2 /* 0x13 External Interrupt 1 */
#define  TF1_VECTOR     3 /* 0x1B Timer 1 */
```

```
#define  SIO_VECTOR    4 /* 0x23 Serial port */
#define  TF2_VECTOR    5 /* 0x2B Timer 2 */
#define  EX2_VECTOR    5 /* 0x2B External Interrupt 2 */
#endif
```

IV. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP

1. Câu hỏi ôn tập

Câu số 5-1: Hãy cho biết các kiểu bộ nhớ trong lập trình C của vi điều khiển AT89S52.

Câu số 5-2: Hãy cho biết loại dữ liệu trong ngôn ngữ lập trình C cho vi điều khiển.

Câu số 5-3: Hãy cho biết các toán tử trong trong ngôn ngữ lập trình C.

2. Câu hỏi mở rộng

Câu số 5-4: Hãy tìm hiểu các kiểu bộ nhớ trong ngôn ngữ lập trình C của vi điều ATMEGA128.

Câu số 5-5: Hãy tìm trình biên dịch Mikro-C.

3. Câu hỏi trắc nghiệm

Câu 5-1: Kiểu dữ liệu "signed char" là kiểu:

(a) 8 bit (b) 16 bit

(c) 24 bit (d) 32 bit

Câu 5-2: Giới hạn của kiểu dữ liệu "signed char" là:

(a) Từ 0 đến 255 (b) Từ 0 đến 65535

(c) Từ -256 đến 255 (d) Từ -128 đến 127

Câu 5-3: Kiểu dữ liệu "unsigned char" là kiểu:

(a) 8 bit (b) 16 bit

(c) 24 bit (d) 32 bit

Câu 5-4: Giới hạn của kiểu dữ liệu "unsigned char" là:

(a) Từ 0 đến 255 (b) Từ 0 đến 65535

(c) Từ -256 đến 255 (d) Từ -128 đến 127

Câu 5-5: Kiểu dữ liệu "unsigned int" là kiểu:

(a) 8 bit (b) 16 bit

(c) 24 bit (d) 32 bit

Câu 5-6: Giới hạn của kiểu dữ liệu "unsigned int" là:

(a) Từ 0 đến 255 (b) Từ 0 đến 65535

(c) Từ -256 đến 255 (d) Từ -128 đến 127

Câu 5-7: Toán tử nào là toán tử gán:

(a) "=" (b) "%"

(c) "/" (d) "&"

Câu 5-8: Toán tử nào là chia lấy số dư:

(a) "=" (b) "%"

(c) "/" (d) "&"

Câu 5-9: Toán tử nào là chia lấy kết quả nguyên:

(a) "=" (b) "%"

(c) "/" (d) "&"

Câu 5-10: Toán tử nào là so sánh xem có bằng hay không:

(a) "=" (b) "!="

(c) "==" (d) "<="

Câu 5-11: Toán tử nào là so sánh không bằng:

(a) "=" (b) "!="

(c) "==" (d) "<="

Câu 5-12: Toán tử nào là so sánh nhỏ hơn hoặc bằng:

(a) ">=" (b) "!="

(c) "==" (d) "<="

Câu 5-13: Toán tử nào là AND 2 điều kiện với nhau:

(a) "!" (b) "&&"

(c) "&" (d) "||"

Câu 5-14: Kiểu dữ liệu nào là 32 bit:

(a) int (b) unsigned long

(c) unsigned int (d) signed int

Câu 5-15: Toán tử nào là OR 2 điều kiện với nhau:

(a) "!" (b) "&&"

(c) "&" (d) "||"

Câu 5-16: Toán tử nào là phủ định dữ liệu:

(a) "!" (b) "~"

(c) "&" (d) "|"

Câu 5-17: Toán tử nào là phủ định điều kiện:

(a) "!" (b) "&&"

(c) "&" (d) "||"

Câu 5-18: Toán tử nào là AND 2 dữ liệu:

(a) ">>" (b) "~"

(c) "&" (d) "|"

Câu 5-19: Toán tử nào là dịch trái dữ liệu:

(a) "<<" (b) "~"

(c) "&" (d) ">>"

Câu 5-20: Toán tử lấy kích thước theo byte:

(a) "size" (b) "sizeof"

(c) "sizeoff" (d) "sizeon"

Câu 5-21: Vòng lặp while (điều_kiện) sẽ làm khi điều kiện:

(a) Lớn hơn 1 (b) Nhỏ hơn 1

(c) Sai (d) Đúng

Câu 5-22: Vòng lặp for(i=1;i<10;i++) sẽ thực hiện các lệnh trong vòng lặp:

(a) 10 lần (b) 100 lần

(c) 9 lần (d) 99 lần

Câu 5-23: Chỉ dẫn "#define" có chức năng:

(a) Chèn một đoạn lệnh từ một file khác vào

(b) Khai báo thư viện của vi điều khiển

(c) Định nghĩa một đoạn chuỗi

(d) Khai báo thư viện của C

Câu 5-24: Chỉ dẫn "#include" có chức năng:

(a) Chèn một đoạn lệnh từ một file khác vào

(b) Khai báo thư viện của vi điều khiển

(c) Khai báo thư viện

(d) Tất cả đều đúng

4. Bài tập

Chương 6
VI ĐIỀU KHIỂN AT89S52: PORT XUẤT NHẬP

- **GIỚI THIỆU**
- **CHỨC NĂNG CÁC PORT CỦA VI ĐIỀU KHIỂN**
- **PORT CỦA VI ĐIỀU KHIỂN ATMEL AT89S52**
  - ĐỊNH CẤU HÌNH CHO PORT
  - LẬP TRÌNH TRUY XUẤT PORT DÙNG NGÔN NGỮ ASSEMBLY
  - LẬP TRÌNH TRUY XUẤT PORT DÙNG NGÔN NGỮ KEIL-C
- **CÁC ỨNG DỤNG PORT CỦA VI ĐIỀU KHIỂN AT89S52**
  - ỨNG DỤNG AT89S52 ĐIỀU KHIỂN LED ĐƠN
  - ỨNG DỤNG AT89S52 ĐIỀU KHIỂN LED 7 ĐOẠN TRỰC TIẾP
  - ỨNG DỤNG AT89S52 ĐIỀU KHIỂN LED 7 ĐOẠN QUÉT
  - GIAO TIẾP VĐK AT89S52 VỚI NÚT NHẤN, BÀN PHÍM
- **GIAO TIẾP VI ĐIỀU KHIỂN AT89S52 VỚI LCD**
  - GIỚI THIỆU LCD
  - SƠ ĐỒ CHÂN CỦA LCD
  - SƠ ĐỒ MẠCH GIAO TIẾP VI ĐIỀU KHIỂN VỚI LCD
  - CÁC LỆNH ĐIỀU KHIỂN LCD
  - ĐỊA CHỈ CỦA TỪNG KÍ TỰ TRÊN LCD
  - CÁC CHƯƠNG TRÌNH HIỂN THỊ TRÊN LCD
- **CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP**
  - CÂU HỎI ÔN TẬP
  - CÂU HỎI MỞ RỘNG
  - CÂU HỎI TRẮC NGHIỆM
  - BÀI TẬP

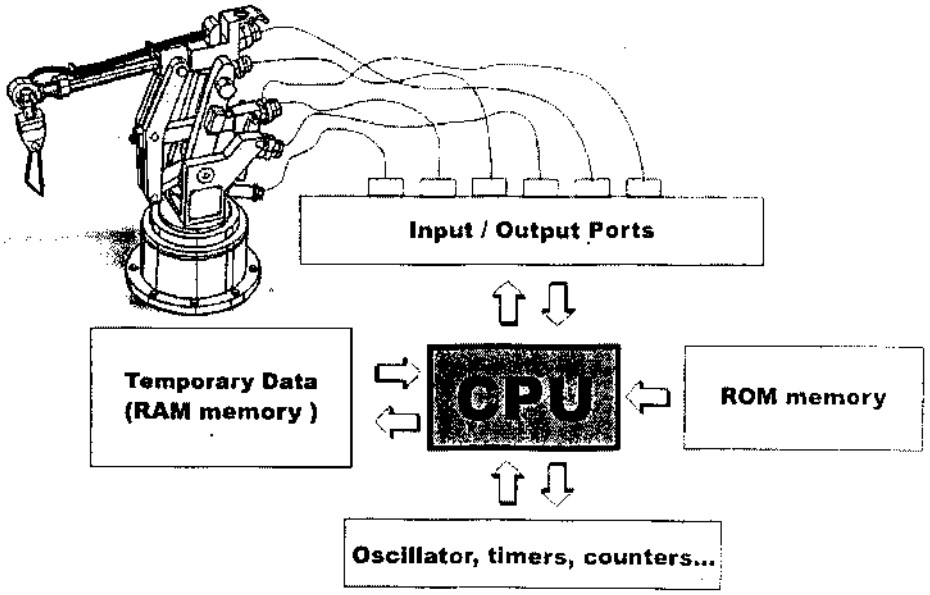
I. GIỚI THIỆU

Ở chương này khảo sát các port xuất nhập của các vi điều khiển, các lệnh để định cấu hình cho port và các lệnh xuất nhập dữ liệu cho port, ứng dụng các port để điều khiển led đơn, led 7 đoạn, LCD, nút nhấn, bàn phím ma trận.

Sau khi kết thúc chương này thì người đọc có thể sử dụng được các port để điều khiển.

II. CHỨC NĂNG CÁC PORT CỦA VI ĐIỀU KHIỂN

Vi điều khiển có các port để xuất nhập dữ liệu giao tiếp với các đối tượng điều khiển. Tín hiệu điều khiển từ CPU gởi ra port để điều khiển, đồng thời có các port nhận dữ liệu về để xử lý. Trong một hệ thống luôn có các tín hiệu vào ra ví như hệ thống điều khiển robot như hình sau:



Hình 6-1: Sơ đồ kết nối port với đối tượng điều khiển.

Các port chỉ tạo ra các mức logic tương thích với chuẩn TTL, nếu điều khiển các đối tượng công suất lớn thì phải thêm mạch giao tiếp.

III. PORT CỦA VI ĐIỀU KHIỂN ATMEL AT89S52

1. Định cấu hình cho port

Vi điều khiển AT89S52 có 4 port 0, 1, 2 và 3, các port khi reset thì ngõ ra bằng 1, dòng ra vào của các port rất nhỏ: port 1, 2, 3 khi ở mức thấp

thì nhận dòng vào khoảng 1,6mA, port 0 thì 3,2mA, khi ở mức cao thì dòng ra khoảng 60μA.

Do dòng cấp nhỏ nên khi sử dụng các port phải dùng điện trở kéo lên và dùng IC đệm để mở rộng khả năng cấp dòng.

Các port của vi điều khiển AT89S52 có thể điều khiển vào ra tùy ý, không cần định cấu hình, khi làm ngõ vào thì nên xuất dữ liệu FFh ra port trước khi tiến hành đọc dữ liệu vào.

2. Lập trình truy xuất port dùng ngôn ngữ Assembly

➢ **Lệnh truy xuất port 8 bit:**

Các port khi truy xuất dùng địa chỉ trực tiếp nên các lệnh xử lý dùng kiểu định địa chỉ trực tiếp đều có thể gởi dữ liệu ra port hoặc đọc dữ liệu về từ port.

Ví dụ 6-1: Lệnh MOV P0, #0FH có chức năng gởi dữ liệu 8bit 0Fh ra port0.

Ví dụ 6-2: Lệnh MOV P1, A có chức năng gởi dữ liệu trong thanh ghi A ra port1.

Ví dụ 6-3: Lệnh MOV R1, P2 có chức năng đọc dữ liệu từ port2 vào thanh ghi R1.

➢ **Lệnh truy xuất bit của port:**

Các port khi truy xuất từng bit dùng các lệnh xử lý bit.

Ví dụ 6-4: Lệnh SETB P1.2 có chức năng làm bit P1.2 lên mức 1.

Ví dụ 6-5: Lệnh MOV C, P2.0 có chức năng đọc dữ liệu từ P2.0 vào cờ C.

3. Lập trình truy xuất port dùng ngôn ngữ Keil-C

➢ **Lệnh truy xuất port 8 bit: Ngôn ngữ lập trình Keil-C cho phép truy xuất port rất đơn giản:**

Ví dụ 6-6: Lệnh P1 = 0x0F; có chức năng gởi dữ liệu 0Fh ra port1

Ví dụ 6-7: Lệnh X = P2; có chức năng đọc dữ liệu port2 gán cho biến X.

➢ **Lệnh truy xuất bit của port:**

Ví dụ 6-8: Lệnh P1_0 = 0; làm bit thứ 0 của port1 bằng 0

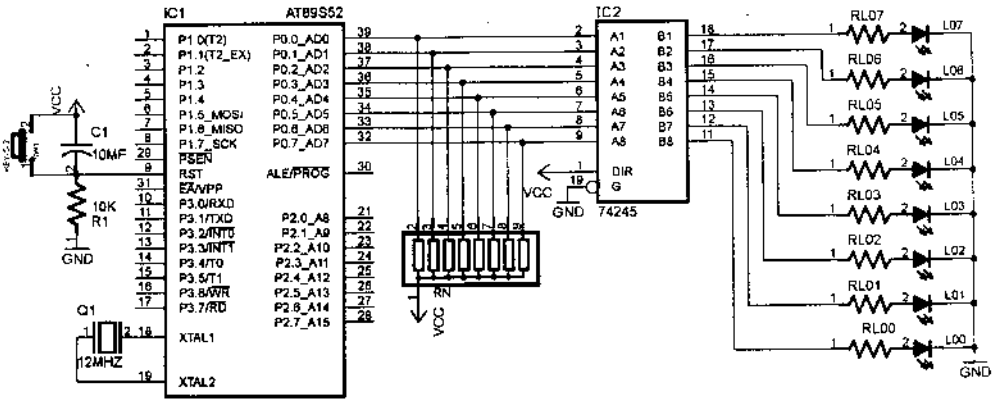
Ví dụ 6-9: Lệnh P2_2 = 1; làm bit thứ 2 của port2 bằng 1.

IV. CÁC ỨNG DỤNG PORT CỦA VI ĐIỀU KHIỂN AT89S52

1. Ứng dụng AT89S52 điều khiển led đơn

Bài 6-1: Dùng port 0 của AT89S52 điều khiển 8 led đơn sáng tắt.

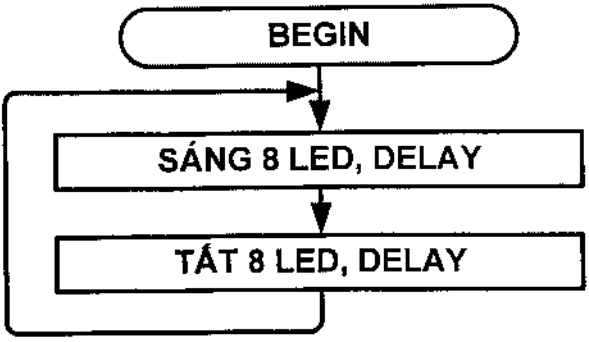
➢ ***Sơ đồ mạch:***



Hình 6-2: *Sơ đồ kết nối port0 điều khiển led đơn qua IC đệm.*

Mạch dùng port0 kết nối với IC đệm 74245 để tăng khả năng cấp dòng điều khiển led. Mức logic 0 làm led tắt, mức logic 1 làm led sáng. Điện trở thanh RN treo port0 lên nguồn Vcc có giá trị 4,7kΩ, điện trở hạn dòng cho led là 330Ω.

➢ ***Lưu đồ:***



Hình 6-3: *Lưu đồ điều khiển led đơn chớp tắt.*

➢ **Chương trình viết bằng hợp ngữ:**

```
        ORG     0000H
LB:     MOV     P0,#00H
```

```
        CALL    DELAY
        MOV     P0,#0FFH
        CALL    DELAY
        JMP     LB
$INCLUDE(TV_DELAY.ASM)
        END
```

❖ **Giải thích chương trình hợp ngữ:**

➢ ***Chương trình chính:***

"ORG 0000H" là chỉ dẫn khai báo địa chỉ bắt đầu của chương trình, khi reset vi điều khiển thì địa chỉ của thanh ghi PC bằng 0000H nên sẽ thực hiện chương trình bắt đầu tại địa chỉ này.

"LB: MOV P0,#00H" gồm nhãn để nhảy đến có tên là LB và lệnh nạp dữ liệu 00 cho port0 làm tắt 8 led.

"CALL DELAY" là lệnh gọi chương trình con delay để cho chúng ta nhìn thấy led tắt.

"MOV P0,#0FFH" là lệnh nạp dữ liệu FF cho port0 làm sáng 8 led.

"CALL DELAY" là lệnh gọi chương trình con delay để cho chúng ta nhìn thấy led sáng.

"JMPLB" là lệnh nhảy về nhãn LB để làm lại từ đầu.

➢ ***Chương trình con delay:***

"$INCLUDE (TV_DELAY.ASM)" là chỉ dẫn khai báo thư viện delay chứa các chương trình con delay viết sẵn.

Cuối cùng là chỉ dẫn báo kết thúc.

➢ ***Chương trình Keil-C:***

```
#include <AT89X52.H>
void delay (unsigned    int x)
{ unsigned int y;
        for (y=0; y<x; y++)     { }
}
```

```
void MAIN ()
{
 while(1)
            {     P0 = 0x00; delay (10000);    P0 = 0xFF; delay (10000);
      }
}
```

➢ ***Giải thích chương trình:***

Hàng lệnh đầu tiên là khai báo thư viện của chương trình.

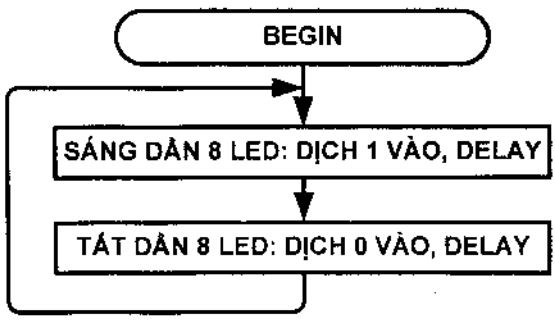
Tiếp theo là khai báo chương trình con delay: chương trình con có tên là "void delay (unsigned int x)", trong tên chương trình có khai báo biến "x" là kiểu số nguyên 16 bit không dấu, khi gọi chương trình con sẽ nhận thông số gán cho biến x, khai báo biến y và lệnh for cho biến y tăng từ 0 đến nhỏ hơn x sẽ tạo ra khoảng thời gian trễ đáng kể để bạn có thể nhìn thấy led tắt hoặc sáng.

Chương trình chính có tên là "void MAIN ()", trong chương trình có lệnh "while (1)" luôn thỏa điều kiện nên những lệnh trong dấu ngoặc sẽ được thực hiện lặp lại không giới hạn. Lệnh gán "P0 = 0x00;" làm tắt 8 led, gọi delay để làm trễ, lệnh gán "P0 = 0xFF;" làm sáng 8 led, gọi delay để làm trễ rồi lặp lại. Có thể điều chỉnh thông số trong hàm delay để tăng giảm thời gian trễ, chưa quan tâm đến thời gian trễ chính xác bằng bao nhiêu.

Bài 6-2: Dùng port 0 của AT89S52 điều khiển 8 led đơn sáng dần rồi tắt – gọi tắt là sáng tắt dần từ phải sang trái.

➢ ***Sơ đồ mạch: giống như bài 6-1.***

➢ ***Lưu đồ:***



Hình 6-4*: Lưu đồ điều khiển 8 led đơn sáng tắt dần phải sang trái.*

➢ ***Chương trình Keil-C:***

```
#include <AT89X52.H>
unsigned char I;
void delay (unsigned   int x)
{ unsigned int y;
            for (y=0; y<x; y++)     { }
}
void MAIN ()
{      P0=0x00;
       while(1)
            {      for (I=0;I<8;I++)
                   {      P0 = (P0<<1)+0x01;    delay (10000); }

                   for (I=0;I<8;I++)
                   {      P0 = (P0<<1)     ;       delay (10000); }
}}
```

❖ **Giải thích chương trình:**

Do có 8 led nên mỗi vòng lặp for thực hiện 8 lần.

Vòng lặp for thứ nhất sẽ tiến hành dịch dữ liệu của P0 sang trái và cộng với số 1 để đẩy số 1 vào rồi delay. Quá trình này sẽ làm 8 led sáng dần lên.

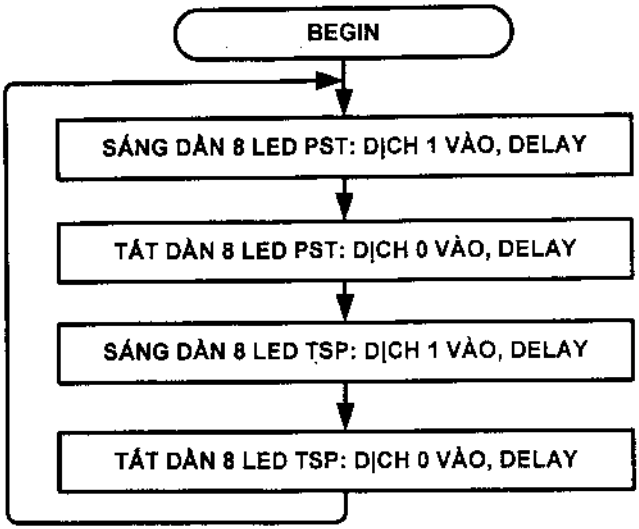
Vòng lặp for thứ hai sẽ tiến hành dịch dữ liệu của P0 sang trái để đẩy số 0 vào rồi delay. Quá trình này sẽ làm 8 led tắt dần.

Sau khi kết thúc thì làm lại từ đầu.

Bài 6-3: Dùng port 0 của AT89S52 điều khiển 8 led đơn sáng tắt dần từ phải sang trái rồi từ trái sang phải.

➢ ***Sơ đồ mạch: giống như bài 6-1.***

➢ ***Lưu đồ:***



Hình 6-5: Lưu đồ điều khiển 8 led đơn sáng tắt dần phải sang trái, trái sang phải.

➢ ***Chương trình Keil-C:***

```
#include <AT89X52.H>
unsigned char I;
void delay (unsigned    int x)
{ unsigned int y;
            for (y=0; y<x; y++)      { }
}
void MAIN ()
{       P0=0x00;
        while(1)
            {       for (I=0;I<8;I++)
                    {       P0 = (P0<<1)+0x01;    delay (10000); }
                    for (I=0;I<8;I++)
                    {       P0 = (P0<<1)      ;       delay (10000); }
                    for (I=0;I<8;I++)
                    {       P0 = (P0>>1)+0x80;    delay (10000); }
```

```
                for (I=0;I<8;I++)
                {     P0 = (P0>>1)     ;     delay (10000); }
}}
```

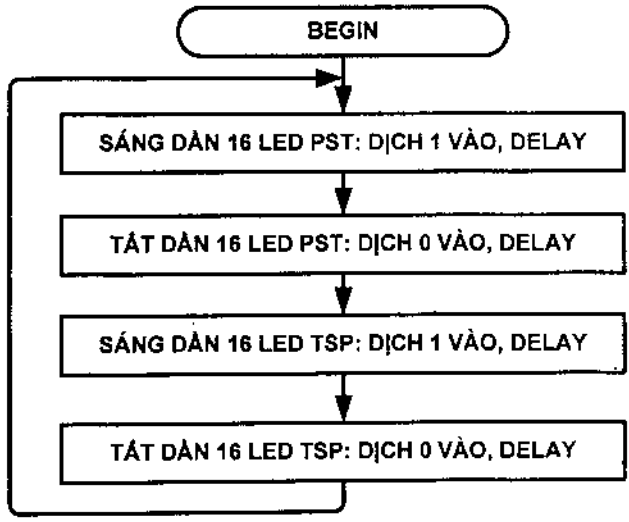
❖ **Giải thích chương trình:**

Giống như bài 6-2 chỉ thêm vào hai vòng lặp for để dịch chiều ngược lại.

Bài 6-4: Dùng hai port 0 và 1 của AT89S52 điều khiển 16 led đơn sáng tắt dần từ phải sang trái rồi từ trái sang phải,.

➢ ***Sơ đồ mạch: giống như bài 6-1 nhưng thêm port 1.***

➢ ***Lưu đồ:***



Hình 6-6: *Lưu đồ điều khiển 16 led đơn sáng tắt dần phải sang trái, trái sang phải.*

➢ ***Chương trình Keil-C:***

```
#include <AT89X52.H>
unsigned char I;
unsigned int Z;

void delay (unsigned    int x)
{ unsigned int y;
```

```
            for (y=0; y<x; y++)     { }     }

void XUATPORT ()
{     P0 = Z;              P1 = Z>>8; }

void MAIN ()
{     Z=0x0000;
 while(1)
      {for (I=0;I<16;I++)
            {Z = (Z<<1)+0x01;        XUATPORT();      delay (5000); }
      for (I=0;I<16;I++)
            {Z = (Z<<1);             XUATPORT();      delay (5000); }
        for (I=0;I<16;I++)
            {Z = (Z>>1)+0x8000;      XUATPORT();      delay (5000); }
        for (I=0;I<16;I++)
            {Z = (Z>>1);             XUATPORT();      delay (5000); }
}}
```

❖ **Giải thích chương trình:**

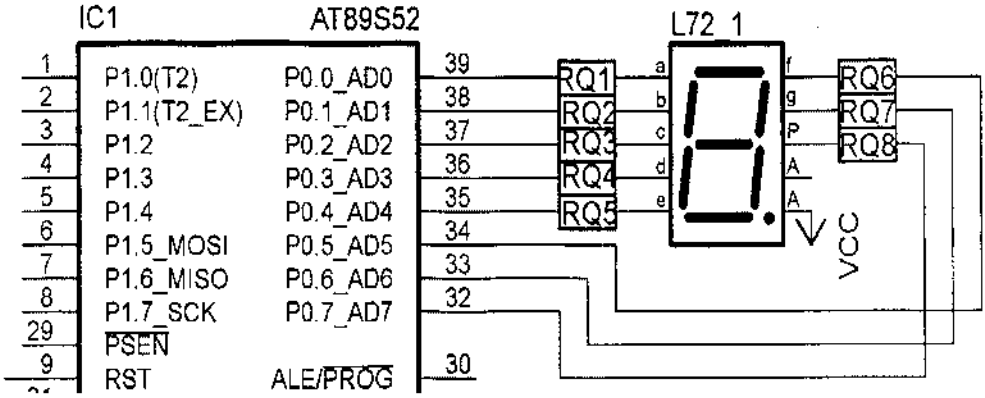
Do có 16 led tương đương 16 bit nên chương trình sử dụng biến Z là 16 bit và tiến hành dịch mức 1 vào 16 lần rồi dịch mức 0 vào 16 lần, sau mỗi lần dịch thì tiến hành tách dữ liệu 16 bit thành 2 byte gán cho hai port 0 và 1 để điều khiển led sáng dần rồi tắt dần.

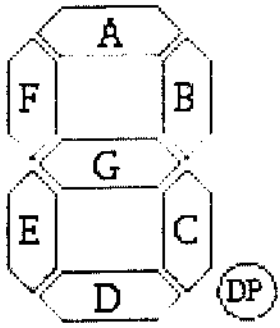
2. Ứng dụng AT89S52 điều khiển led 7 đoạn trực tiếp

Bài 6-5: Dùng vi điều khiển AT89S52 kết nối với một led 7 đoạn anode chung và viết chương trình đếm từ 0 đến 9.

➢ ***Sơ đồ mạch:***

Mạch dùng port0 kết nối trực tiếp với led 7 đoạn. Mức logic 1 làm led tắt, mức logic 0 làm led sáng. Trong sơ đồ mạch này thì không cần dùng điện trở kéo lên vì điện trở và led là tải đã kéo lên nguồn.





Hình 6-7: *Sơ đồ kết nối port0 điều khiển led 7 đoạn.*

Hình 6-8: *Hình led 7 đoạn.*

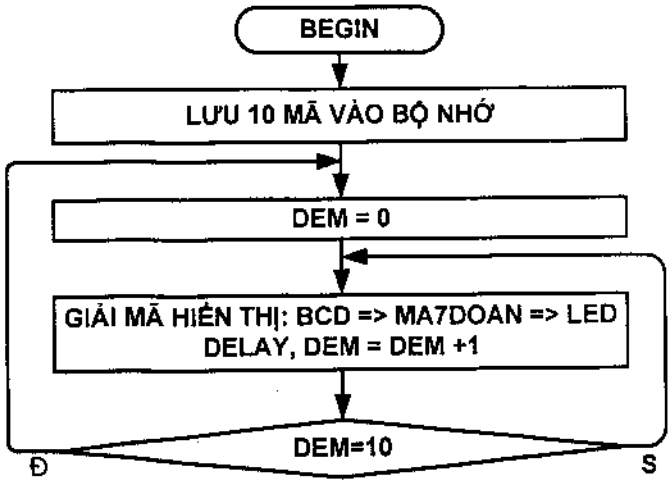
➢ ***Mã 7 đoạn:***

Led 7 đoạn và tên các đoạn như hình 6-8:

Bảng 6-1: *Mã 7 đoạn cho các số thập phân như sau:*

TP	SỐ NHỊ PHÂN								HEX	TP	SỐ NHỊ PHÂN								HEX
	7	6	5	4	3	2	1	0			7	6	5	4	3	2	1	0	
	DP	G	F	E	D	C	B	A			DP	G	F	E	D	C	B	A	
0	1	1	0	0	0	0	0	0	C0	5	1	0	0	1	0	0	1	0	92
1	1	1	1	1	1	0	0	1	F9	6	1	0	0	0	0	0	1	0	82
2	1	0	1	0	0	1	0	0	A4	7	1	1	1	1	1	0	0	0	F8
3	1	0	1	1	0	0	0	0	B0	8	1	0	0	0	0	0	0	0	80
4	1	0	0	1	1	0	0	1	99	9	1	0	0	1	0	0	0	0	90

➢ ***Lưu đồ***: hình 6-9.



Hình 6-9: *Lưu đồ đếm từ 0 đến 9.*

➢ ***Giải thích lưu đồ:***

10 mã 7 đoạn của 10 số thập phân từ 0 đến 9 được lưu vào bộ nhớ.

Cho biến "DEM" bắt đầu từ 0.

Chuyển đổi giá trị của biến "DEM" dạng số nhị phân thành số BCD hàng đơn vị và hàng chục, chuyển mã BCD thành mã 7 đoạn rồi gởi ra port điều khiển led sáng đúng số thập phân.

Thực hiện delay để trì hoãn sau đó tăng biến "DEM" lên 1.

So sánh biến "DEM" xem bằng 10 hay chưa: nếu chưa thì quay lại và làm lại như đã trình bày, nếu bằng 10 thì kết thúc xem như làm lại từ đầu với biến "DEM" bằng 0.

➢ ***Chương trình Keil-C:***

```
#include <AT89X52.h>
unsigned char MA7D [10] =
{0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90};
signed char DEM;
void delay(unsigned int x)
{     unsigned int y;
      for(y=0; y<x; y++)      {}
```

```
}
void MAIN ()
{
 while(1)
            {
                for (DEM = 0; DEM <10; DEM++)
                        {  P0 = MA7D[DEM];          delay(20000);}
              }
}
```

❖ **Giải thích chương trình:**

Sau phần khai báo thư viện là khai báo mảng 10 kí tự của 10 mã 7 đoạn từ 0 đến 9, tên của mảng là "MA7D", kiểu kí tự không dấu.

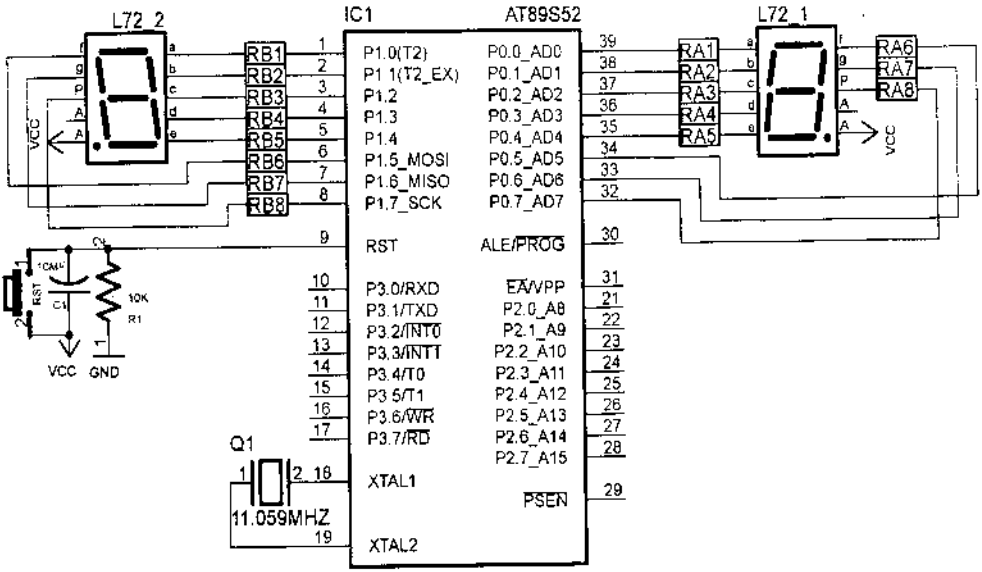
Tiếp theo là khai báo biến "DEM" kiểu số nguyên có dấu.

Chương trình con delay có truyền tham số để có thể thay đổi thời gian tùy ý từ nơi gọi hàm delay, trong chương trình chính có gọi hàm delay với tham số truyền là 20000.

Chương trình chính cho vòng lặp for với biến đếm chạy từ 0 đến nhỏ hơn 10, mỗi lần tăng lên 1 đơn vị và thực hiện việc lấy mã 7 đoạn trong mảng "MA7D" với chỉ số là biến "DEM". Nếu biến "DEM" bắt đầu bằng 0 thì lấy phần tử thứ 0 của mảng là "0xC0", sau đó biến "DEM" tăng lên 1 thì lấy phần tử thứ 1 của mảng là "0xF9", tương tự cho đến khi bằng 9 thì lấy phần tử thứ 9 của mảng là "0x90". Các phần tử của mảng sau đó gán cho port0 sẽ điều khiển led sáng lần lượt từ số 0 đến số 9.

Bài 6-6: Dùng vi điều khiển AT89S52 kết nối với hai led 7 đoạn anode chung và viết chương trình đếm từ 00 đến 99.

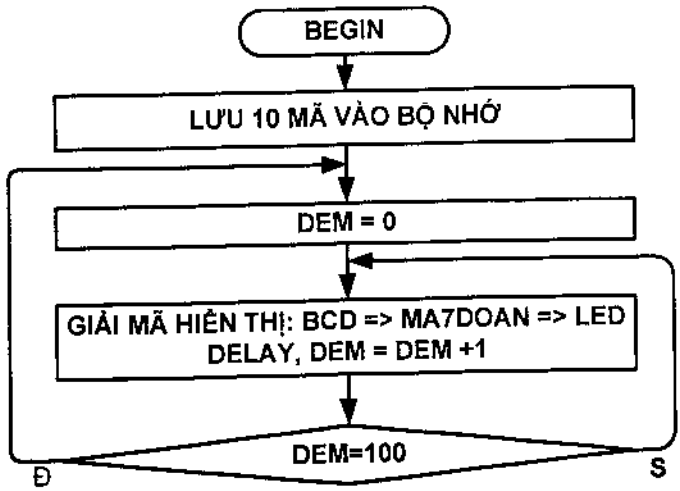
Sơ đồ mạch:



Hình 6-10: *Sơ đồ kết nối hai port điều khiển hai led 7 đoạn.*

Mạch dùng port0 và port1 kết nối trực tiếp với led 7 đoạn. Mức logic 1 làm led tắt, mức logic 0 làm led sáng.

➢ ***Lưu đồ:***



Hình 6-11: *Lưu đồ đếm từ 00 đến 99.*

➢ ***Giải thích lưu đồ:***

10 mã 7 đoạn của 10 số thập phân từ 0 đến 9 được lưu vào bộ nhớ.

Cho biến "DEM" bắt đầu từ 0.

Chuyển đổi giá trị của biến "DEM" dạng số nhị phân thành số BCD hàng đơn vị và hàng chục, chuyển mã BCD thành mã 7 đoạn rồi gởi ra port điều khiển led sáng đúng số thập phân.

Thực hiện delay để trì hoàn sau đó tăng biến "DEM" lên 1.

So sánh biến "DEM" xem bằng 100 hay chưa: nếu chưa thì quay lại và làm lại như đã trình bày, nếu bằng 100 thì kết thúc xem như làm lại từ đầu với biến "DEM" bằng 0.

➢ ***Chương trình Keil-C:***

```
#include < AT89X52.h>
unsigned char MA7D[10] =
{0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
unsigned char DEM,CHUC,DONVI;
void delay (unsigned int x)
{     unsigned int y;
      for (y=0; y<x; y++)     {}
}
void MAIN ()
{ while (1)
      {     for(DEM = 0; DEM < 100; DEM++)
                  {     DONVI = DEM % 10;       CHUC = DEM /10;
                        P0 = MA7D[DONVI];       P1 = MA7D[CHUC];
                        delay (10000);
                  }
}     }
```

❖ **Giải thích chương trình:**

Vòng lặp for cho biến "DEM" tăng giá trị từ 0 lên đến nhỏ hơn 100, mỗi lần tăng 1 rồi tiến hành thực hiện các lệnh:

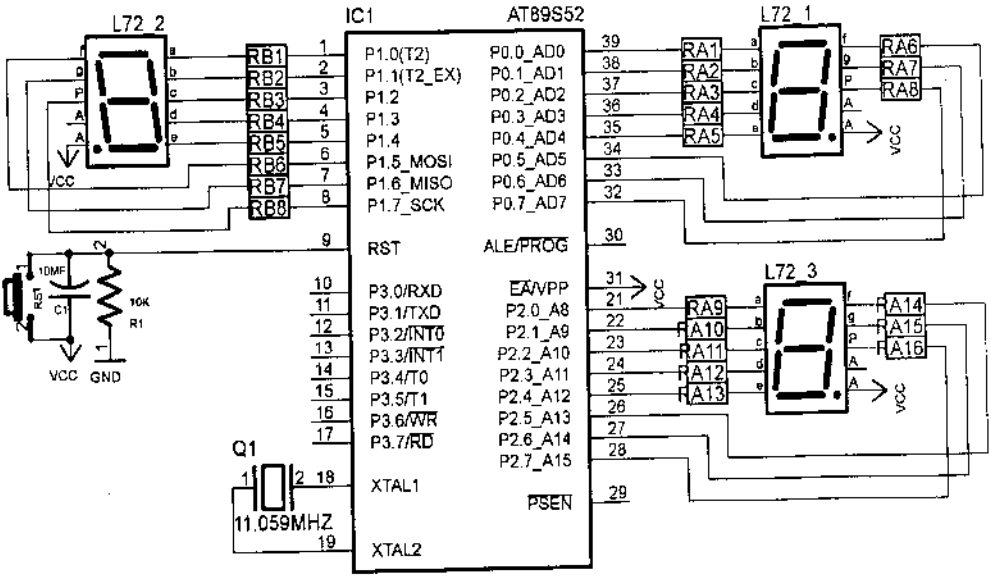
Lệnh "DONVI = DEM % 10;" sẽ lấy giá trị của biến "DEM" chia cho 10 và gán số dư cho biến "DONVI". Lệnh "CHUC = DEM /10;" sẽ lấy giá trị của biến "DEM" chia cho 10 và gán kết quả cho biến "CHUC". Ví dụ cho biến "DEM = 15" thì sau khi thực hiện hai lệnh trên thì "DONVI = 5", "CHUC = 1".

Hai lệnh trên có chức năng tách hàng chục và hàng đơn vị của biến "DEM" thành hai số độc lập để giải mã 7 đoạn.

Hai lệnh tiếp theo tiến hành giải mã 7 đoạn và gán cho hai port để hiển thị trên hai led.

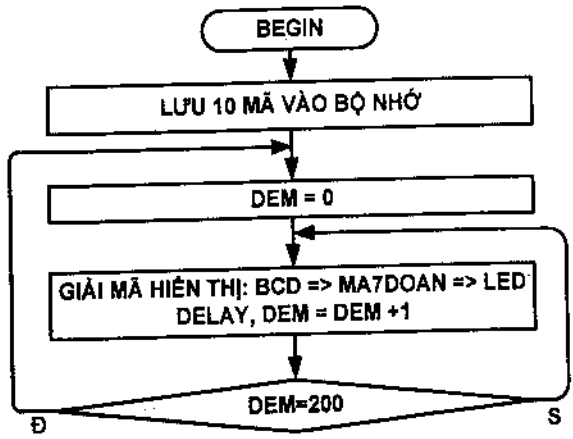
Bài 6-7: Dùng vi điều khiển AT89S52 kết nối với hai led 7 đoạn anode chung và viết chương trình đếm từ 000 đến 199.

Sơ đồ mạch:



Hình 6-12: *Sơ đồ kết nối 3 port điều khiển 3 led 7 đoạn.*

➢ ***Lưu đồ:***



Hình 6-13: *Lưu đồ đếm từ 000 đến 199.*

➢ *Chương trình Keil-C:*

```
#include <AT89X52.h>
const unsigned    char
MA7D[10]={0XC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
signed char DEM;
unsigned char DVI,CHUC,TRAM;

void delay (unsigned int x)
{      unsigned int y;
       for (y=0; y<x; y++)      {;}
}
void GIAIMA_HIENTHI(int DEMT)
{      DVI=DEMT%10;              DEMT=DEMT/10;
       CHUC=DEMT%10;             TRAM=DEMT/10;
       P0=MA7D[DVI];             P1=MA7D[CHUC];
       P2=MA7D[TRAM];
}
void main ()
{ while (1)
       {for (DEM=0;DEM<200;DEM++)
              { GIAIMA_HIENTHI(DEM);      delay (5000);}
}      }
```

❖ **Giải thích chương trình:**

Bài 6-7 là mở rộng của bài 6-6, giá trị đếm từ 0 đến 199 gồm 3 số hàng đơn vị, hàng chục, hàng trăm, vòng lặp for thực hiện 200 lần, mỗi lần gọi chương trình con giải mã hiển thị vào trao giá trị của biến DEM gán cho biến DEMT để tiến hành giải mã và hiển thị.

3. Ứng dụng AT89S52 điều khiển led 7 đoạn quét

Khi số led 7 đoạn cần nhiều hơn ví dụ là 8 led thì ta không thể kết nối trực tiếp một port điều khiển một led vì không đủ port. Có nhiều phương

pháp mở rộng để điều khiển được nhiều led nhưng ở đây trình bày phương pháp quét 8 led dùng hai port.

Bài 6-8: Dùng vi điều khiển AT89S52 kết nối với 8 led 7 đoạn anode chung theo phương pháp quét và viết chương trình hiển thị 8 số từ số 0 đến số 7 trên 8 led.

Sơ đồ mạch: xem hình 6-9.

Mạch dùng port2 kết nối với các đoạn "a, b, c, d, e, f, g, dp" và port1 kết nối điều khiển 8 transistor đóng ngắt.

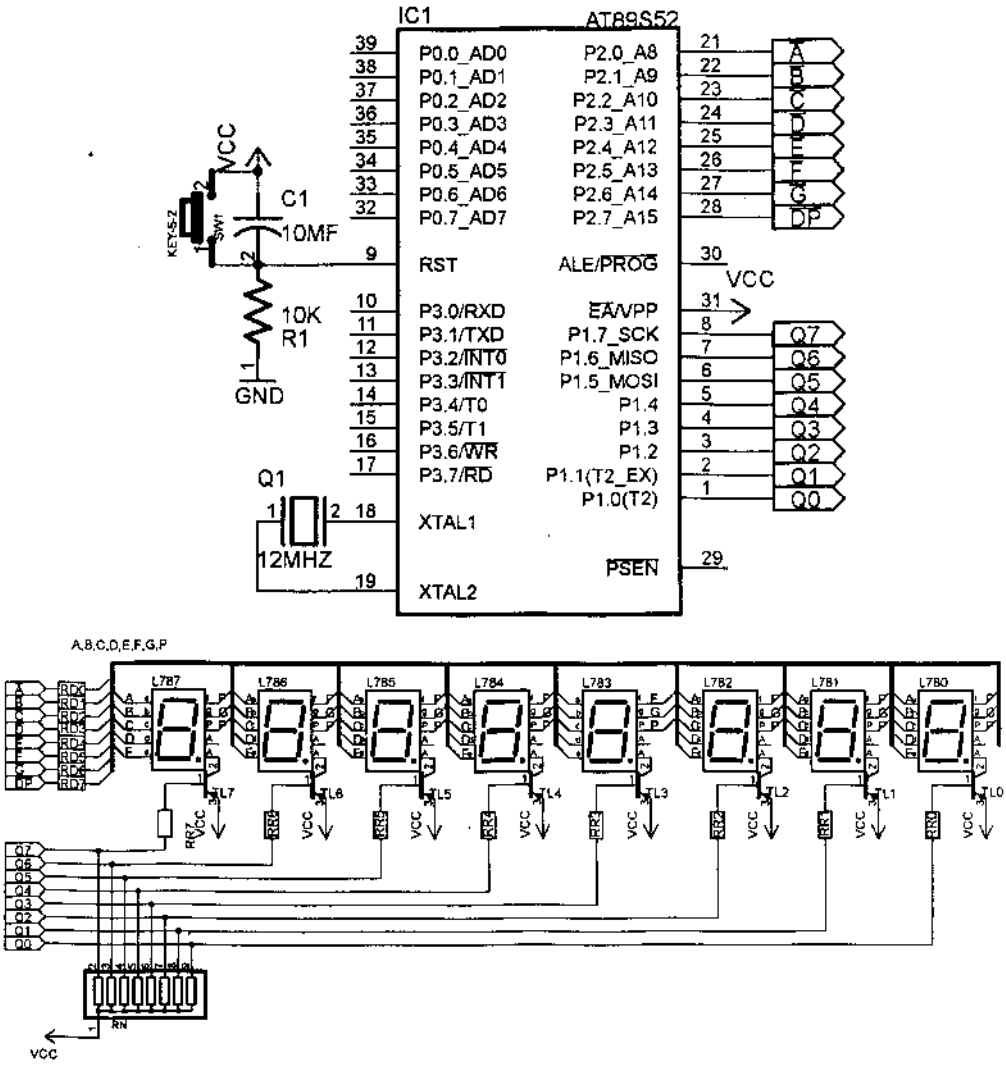
➢ ***Nguyên lý quét 8 led:***

Để điều khiển led L780 sáng số '0' thì mã số 0 là "0xC0" được gởi ra port2 và điều khiển port1 cho một transistor TL0 dẫn, các transistor còn lại tắt, khi đó led L780 sáng số 0.

Sau một khoảng thời gian 1ms thì tắt transistor đã dẫn, số led L780 tắt.

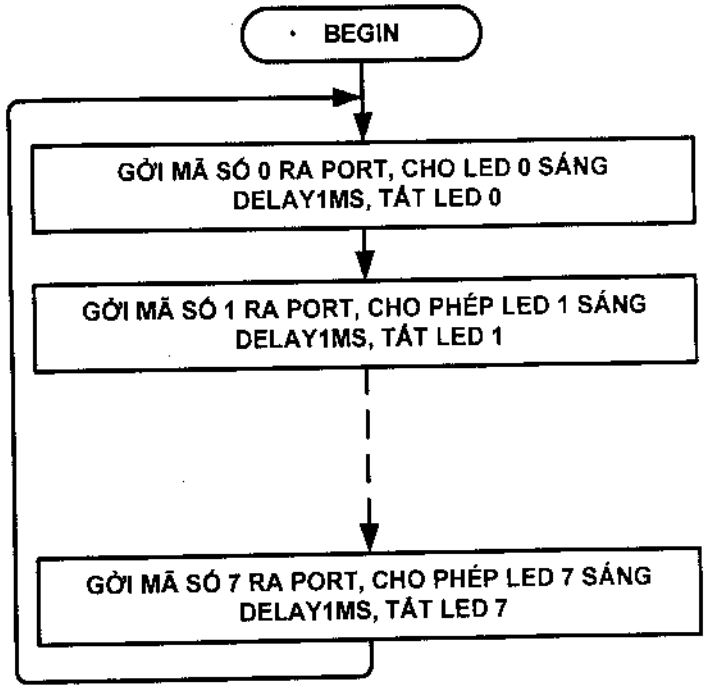
Thực hiện tương tự cho led L781 và cho đến led cuối cùng là L787 rồi lặp lại cho led L780.

Mỗi thời điểm chỉ cho một led sáng nhưng do tốc độ quét nhanh hơn đáp ứng của mắt nên bạn không nhìn thấy led tắt mà nhìn thấy 8 led sáng cùng một lúc. Chức năng của transistor là điều khiển cấp dòng hoặc không cấp dòng cho led và transistor phải đủ dòng cấp cho led khi led sáng. Mỗi đoạn là 10mA và sáng cực đại là 8 đoạn thì dòng tổng là 80mA nên có thể chọn transistor có dòng làm việc là 100mA là phù hợp.



Hình 6-14: *Sơ đồ kết nối hai port điều khiển 8 led 7 đoạn quét.*

➢ ***Lưu đồ:***



***Hình 6-15:** Lưu đồ điều khiển 8 led quét hiển thị 8 số.*

➢ ***Giải thích lưu đồ:***

Gởi mã số 0 ra port điều khiển các đoạn của led, điều khiển cho transistor dẫn để cấp dòng cho led sáng, trì hoãn khoảng 1ms sau đó tắt transistor để tắt led và chuyển sang led kế.

➢ ***Chương trình Keil-C:***

```
#include < AT89X52.h>
unsigned int tg;
void delay(unsigned int x)
{ unsigned int y;
      for (y= 0; y<x; y++)     {;}
}
void MAIN ()
{     tg = 200;
 while(1)
```

```
        {
        P2 = 0xC0;      P1_0 = 0;  delay (tg);  P1_0 = 1;
        P2 = 0xF9;      P1_1 = 0;  delay (tg);  P1_1 = 1;
        P2 = 0xA4;      P1_2 = 0;  delay (tg);  P1_2 = 1;
        P2 = 0xB0;      P1_3 = 0;  delay (tg);  P1_3 = 1;
        P2 = 0x99;      P1_4 = 0;  delay (tg);  P1_4 = 1;
        P2= 0x92;       P1_5 = 0;  delay (tg);  P1_5 = 1;
        P2 = 0x82;      P1_6 = 0;  delay (tg);  P1_6 = 1;
        P2 = 0xF8;      P1_7 = 0;  delay (tg);  P1_7 = 1;
        }
}
```

❖ **Giải thích chương trình:**

Lệnh "P2 = 0xC0;" có chức năng gán mã số 0 cho port2.

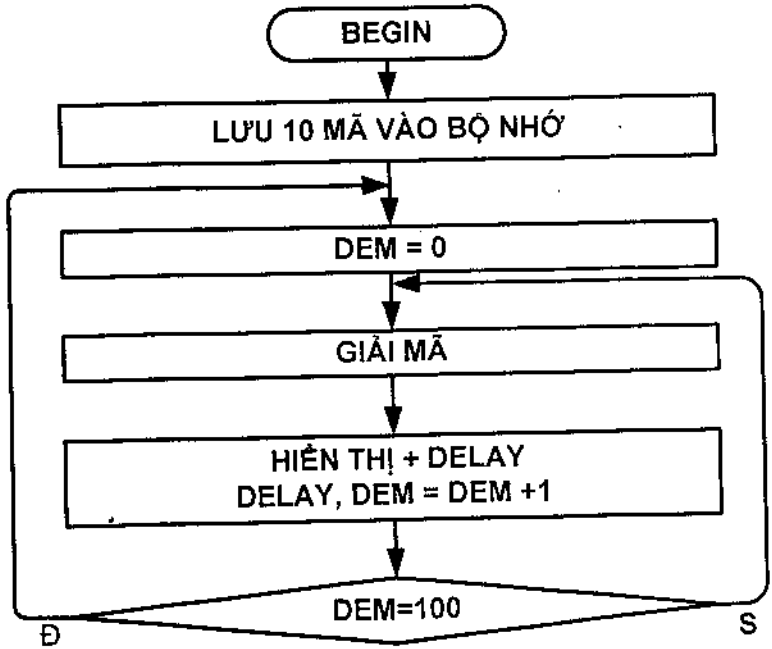
Lệnh "P1_0 = 0;" có chức năng làm bit thứ 0 của port1 xuống mức 0 để mở transistor cấp dòng cho led, lệnh delay trì hoãn nhỏ hơn 1ms sau đó tắt transistor để tắt led.

Lặp lại cho led tiếp theo, thời gian delay nhỏ khoảng 1ms thì ta sẽ nhìn thấy 8 led sáng, thời gian delay lớn thì led sẽ sáng chập chờn hoặc ta sẽ thấy từng led sáng. Có thể tăng thời gian delay để thấy từng led sáng.

Bài 6-9: Viết chương trình đếm từ 00 đến 99 hiển thị trên hai led 7 đoạn quét.

Sơ đồ mạch: giống bài 6-8.

Lưu đồ:



Hình 6-16: Đếm từ 00 đến 99 hiển thị trên hai led quét.

➢ ***Chương trình Keil-C:***

```
#include < AT89X52.h>
unsigned char DEM,J,DONVI,CHUC;
unsigned char MDONVI,MCHUC;
unsigned char MA7D[10] =
{0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
void delay(unsigned int x)
{ unsigned int y;
      for (y= 0; y<x; y++)    {;}}
void HIENTHI_DELAY()
{     for (J=0;J<100;J++)
            { P2 = MDONVI;       P1_0 = 0;  delay (100);      P1_0 = 1;
              P2 = MCHUC;        P1_1 = 0;  delay (100);      P1_1 = 1;}
}
void GIAIMA()
```

```
{     DONVI = DEM %10;              CHUC = DEM/10;
      MDONVI = MA7D[DONVI];         MCHUC = MA7D[CHUC];}
void MAIN ()
{     while(1)
            { for (DEM=0;DEM<100;DEM++)
                  { GIAIMA();     HIENTHI_DELAY();}
            }
}
```

4. Giao tiếp VĐK AT89S52 với nút nhấn, bàn phím

.Nút nhấn, bàn phím dùng để giao tiếp giữa con người và mạch điện tử để điều khiển, ví dụ: bàn phím máy tính, bàn phím điện thoại, bàn phím máy bán xăng dầu dùng nhập số tiền cần bán, số lít cần bán ... máy giặt tự động có bàn phím để chỉnh chế độ giặt, chọn mực nước ...

Có hai dạng giao tiếp vi điều khiển với bàn phím, nút nhấn:

- Dạng giao tiếp ít phím: ví dụ đk động cơ bằng ba phím: start, stop, inv, đồng hồ có ba đến bốn phím để chỉnh thời gian.
- Dạng giao tiếp nhiều phím: bàn phím máy tính, bàn phím điện thoại,

➢ ***Hệ thống ít phím***

Để hiểu giao tiếp vi điều khiển với bàn phím ta khảo sát các bài ứng dụng theo sau.

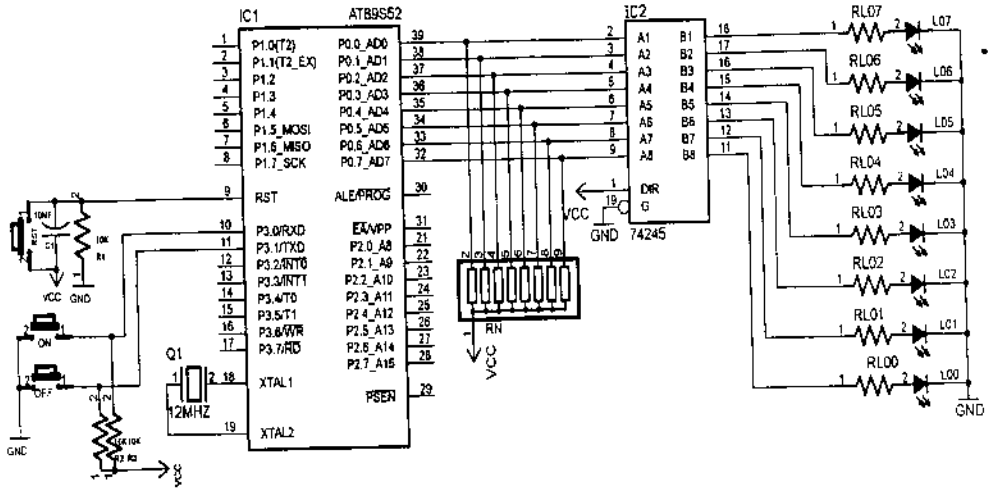
Bài 6-10: Dùng vi điều khiển AT89S52 giao tiếp với 8 led đơn và hai nút nhấn ON, OFF. Khi cấp điện thì 8 led tắt, khi nhấn ON thì 8 led sáng, khi nhấn OFF thì 8 led tắt.

➢ ***Sơ đồ mạch:*** xem hình 6-11.

Hoạt động của hai nút nhấn: theo sơ đồ kết nối trên thì bình thường hai nút nhấn hở, hai ngõ vào có hai điện trở kéo lên nguồn nên mức logic ngõ vào là 1.

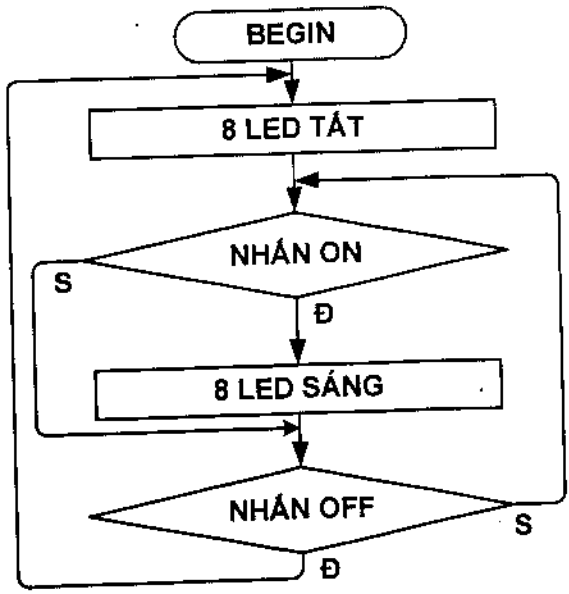
Khi ta có nhấn phím, ngắn mạch ngõ vào xuống mass làm ngõ vào ở mức logic 0, khi nhả phím, ngõ vào trở về lại mức logic 1.

Khi lập trình ta kiểm tra xem có nhấn phím hay không bằng cách kiểm tra mức logic nếu bằng 1 thì không nhấn, bằng 0 thì có nhấn.



Hình 6-17: *Sơ đồ điều khiển led và hai nút nhấn.*

➢ ***Lưu đồ:***



Hình 6-18: *Lưu đồ điều khiển led bằng nút nhấn ON-OFF.*

➢ ***Chương trình Keil-C:***

```
#include <AT89X52.h>
#define ON      P3_0
#define OFF     P3_1
```

```
void MAIN ()
{     P0 = 0x00;
      while(1)
            {     if (ON == 0)      {P0 = 0xFF;}
                  if (OFF == 0)     {P0 = 0x00;}
            }
}
```

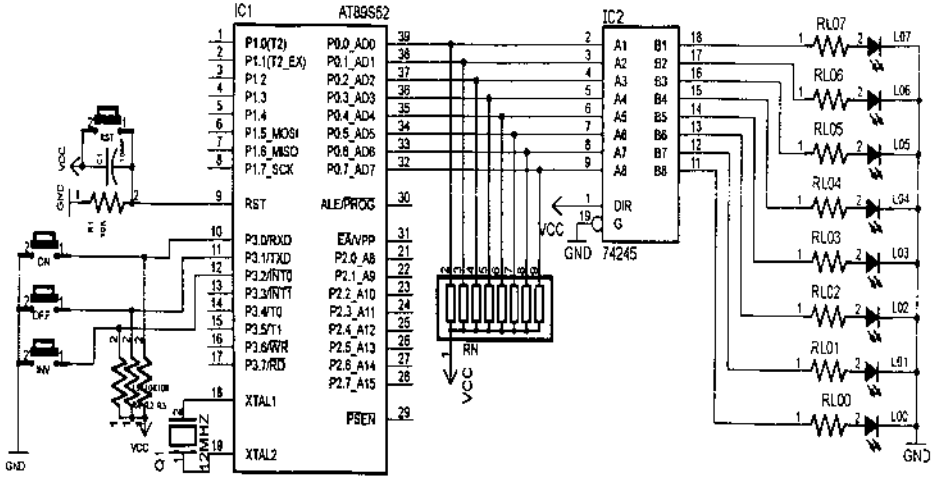
❖ **Giải thích chương trình:**

Lệnh if thứ nhất kiểm tra có nhấn ON hay không? Nếu có thì gán port0 bằng FF nên 8 led sáng.

Lệnh if thứ hai kiểm tra có nhấn OFF hay không? Nếu có thì gán port0 bằng 00 nên 8 led tắt.

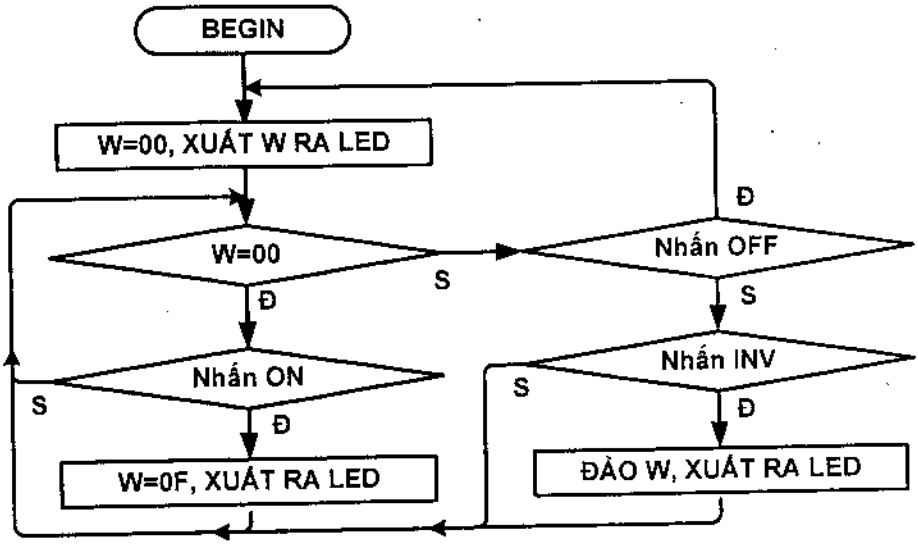
Bài 6-11: Dùng vi điều khiển AT89S52 giao tiếp với 8 led đơn và 3 nút nhấn ON, OFF, INV. Khi cấp điện thì 8 led tắt, khi nhấn ON thì 4 led thấp sáng, khi nhấn INV thì đảo trạng thái của 8 led: 4 sáng thành tắt, 4 tắt thành sáng, khi nhấn OFF thì 8 led tắt.

➢ ***Sơ đồ mạch:***



Hình 6-19: *Sơ đồ điều khiển led và ba nút nhấn.*

➢ ***Lưu đồ:***



Hình 6-20: *Lưu đồ điều khiển led bằng 3 nút nhấn ON-OFF-INV.*

➢ ***Chương trình Keil-C:***

```
#include <AT89X52.h>
#define ON      P3_0
#define OFF     P3_1
#define INV     P3_2
char W;
void MAIN ()
{     W = 0x00; P0 = W;
      while(1)
            {     if (W == 0x00)
                        {if (ON == 0)     {W = 0x0F;      P0 = W;}}
                  else
                        {     if (OFF == 0) {W = 0x00;    P0 = W;}
                              if (INV == 0) {W = ~ W;     P0 = W;}
                        }
            }
}
```

❖ **Giải thích chương trình:**

Lệnh if thứ nhất kiểm tra xem W = 00 hay không? Nếu W = 00 thì mới kiểm tra có nhấn phím ON không? Nếu nhấn ON thì gán biến W = 0F và xuất ra port làm 4 led thấp sáng, 4 led cao tắt. Nếu W khác 00 tức là đã nhấn ON rồi thì không cần kiểm tra nhấn ON nữa, tiến hành kiểm tra xem có nhấn OFF không?

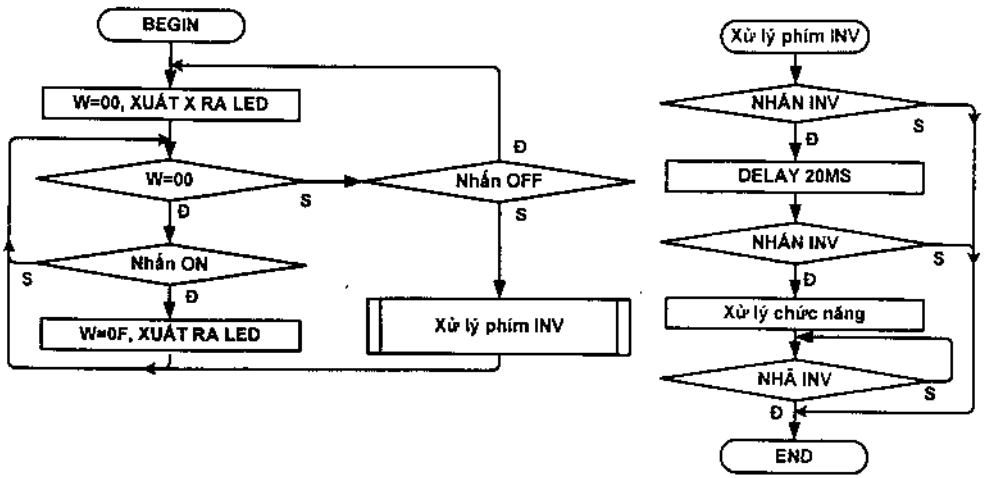
Nếu có nhấn OFF thì gán biến W bằng 0x00 và xuất ra port làm tắt led. Nếu không nhấn OFF thì kiểm tra xem có nhấn phím đảo chiều INV hay không? Nếu có thì đảo giá trị của W và xuất ra port làm 4 led tắt thành sáng, 4 led sáng thành tắt. Nếu không nhấn INV thì quay lại kiểm tra OFF.

Trong chương trình này, khi chạy thực tế, do tốc độ của vi điều khiển quá nhanh, khi nhấn đảo chiều và do thời gian nhấn phím dài nên vi điều khiển thực hiện đảo led liên tục, ta sẽ nhìn thấy 8 led sáng luôn cho đến khi ta buông phím, hoặc ta nhấn nhanh thì trạng thái đảo của led không thể xác định rõ ràng.

Để điều khiển chính xác thì phải chống dội phím nhấn, rồi xử lý chức năng và kiểm tra buông phím.

Trong bài điều khiển chỉ có phím INV gây ra ảnh hưởng nên lưu đồ và chương trình xử lý phần chống dội và chờ buông phím INV như sau:

Lưu đồ:



- ***Hình 6-21:** Lưu đồ điều khiển led có chống dội phím INV.*

➢ ***Chương trình Keil -C:***

```
#include <AT89X52.h>
```

```
#define ON      P3_0
#define OFF     P3_1
#define INV     P3_2
unsigned char W;
void delay (unsigned int x)
{     unsigned int y;
      for (y=0; y<x; y++)     {;}
}
void CHONGDOI_INV ()
{     if (INV == 0)
            {     delay (10000);
                  if (INV == 0)
                        {     W = ~ W;  P0 = W;
                              do    {}
                              while (INV == 0);
}           }           }
void MAIN ()
{     W = 0x00;  P0 = W;
      while(1)
            {     if (W == 0x00)
                        {if (ON == 0)     {W = 0x0F;       P0 = W;}}
                  else
                        {     if (OFF == 0) {W = 0x00;     P0 = W;}
                              CHONGDOI_INV ();
}           }           }
```

❖ **Giải thích chương trình:**

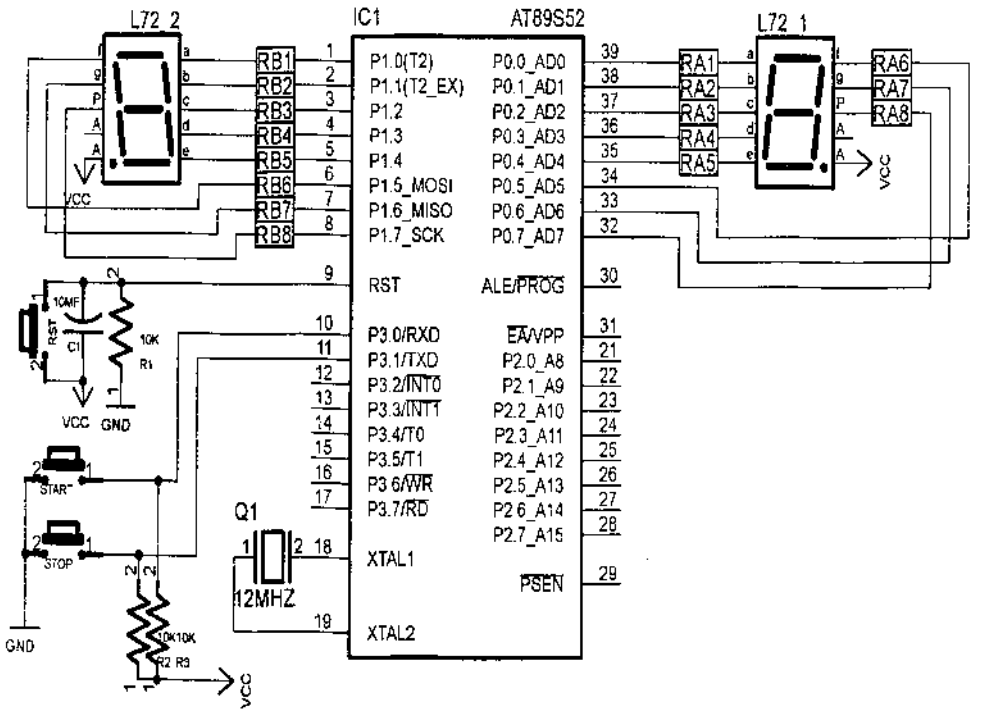
Trong chương trình chính có gọi chương trình con có tên là "CHONGDOI_INV()", ở chương trình con này sẽ kiểm tra xem có nhấn phím INV không, nếu có thì gọi hàm delay, sau đó kiểm tra lần 2 còn nhấn

INV hay không, nếu còn thì xử lý đảo led và thực hiện vòng lặp do while chờ nhả phím mới thoát. Nếu không nhấn INV thì thoát ngay lần đầu hoặc sau delay mà không còn nhấn thì cũng thoát.

Kết quả chạy thực tế khi nhấn INV rất chính xác.

Bài 6-12: Mạch đếm thời gian từ 00 đến 99, dùng vi điều khiển AT89S52 kết nối với hai led 7 đoạn anode chung và hai nút nhấn Start, Stop. Viết chương trình thực hiện chức năng: khi cấp điện thì led hiển thị 00, khi nhấn Start thì mạch đếm từ 00 đến 99, nếu nhấn Stop thì ngừng tại giá trị đang đếm, nhấn Start thì đếm tiếp.

Sơ đồ mạch:

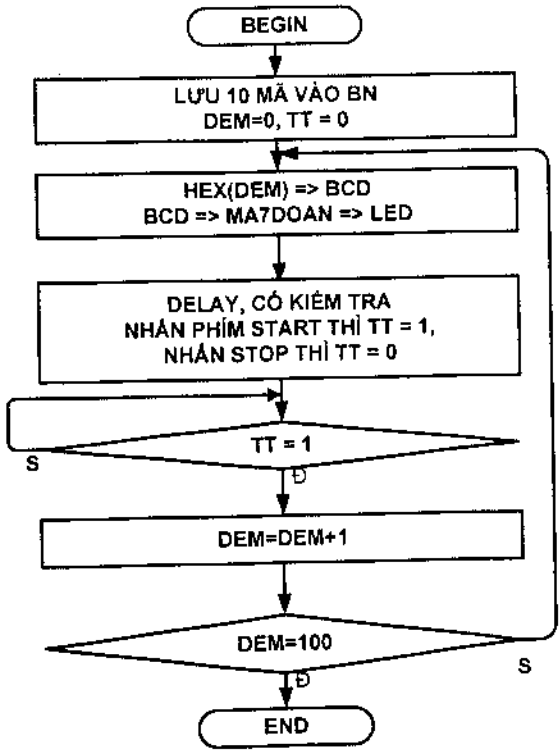


Hình 6-22: *Sơ đồ kết nối hai port điều khiển hai led 7 đoạn, hai nút nhấn.*

➢ ***Giải thích lưu đồ:***

Bài này giống bài ở trước, chỉ thêm phần kiểm tra nút nhấn trong chương trình delay, biến trạng thái TT ban đầu gán bằng 0 tương ứng với chế độ dừng, chương trình delay chỉ thoát khi nhấn Start làm biến TT bằng 1 và cho phép tăng giá trị đếm. Nếu nhấn nút Stop sẽ làm biến trạng thái TT= 0, mạch ngừng tại giá trị đang đếm.

➢ *Lưu đồ:*



***Hình 6-23**: Lưu đồ đếm có điều khiển bằng nút nhấn Start-Stop.*

➢ ***Chương trình Keil-C:***

```
#include <AT89X52.h>
#define START   P3_0
#define STOP    P3_1
unsigned char CHUC,DONVI;      signed char DEM;      bit    TT;
unsigned char MA7D[10] =
{0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90};
void delay(unsigned int x)
{     unsigned int y;
      for(y=0; y<x; y++)
            {     if (START == 0) {TT = 1;}
                  if (STOP == 0)   {TT = 0;}
```

```
                }
}
void MAIN ()
{       TT = 0;
        while(1)
                {       for(DEM = 0; DEM < 100; DEM++)
                        {       DONVI = DEM % 10;       CHUC = DEM /10;
                                P0 = MA7D[DONVI];       P1 = MA7D[CHUC];
                                do
                                        {delay(20000);}
                                while(TT==0);
                        }
                        }
                }
```

❖ **Giải thích chương trình:**

Ban đầu gán biến trạng thái TT = 0 nên mạch ngừng đếm, chương trình chính gọi chương trình con delay và chỉ thoát để tăng giá trị đếm khi biến TT = 1, muốn TT = 1 thì ta phải nhấn nút Start, nếu nhấn Stop thì gán TT = 0, mạch ngừng đếm.

Bài 6-13: Một vi điều khiển AT89S52 kết nối với một led 7 đoạn anode dùng port0 và và hai nút nhấn UP, DOWN. Viết chương trình thực hiện chức năng: khi cấp điện thì led hiển thị 0, khi mỗi lần nhấn UP rồi nhả thì giá trị hiển thị trên led 7 đoạn tăng 1 đơn vị nếu bằng 9 mà tăng nữa thì về 0, nếu mỗi lần nhấn DOWN thì giá trị hiển thị trên led 7 đoạn giảm 1 đơn vị nếu bằng 0 mà giảm nữa thì về 9.

➢ ***Sơ đồ mạch:*** hình 6-24.

➢ ***Lưu đồ:*** hình 6-25.

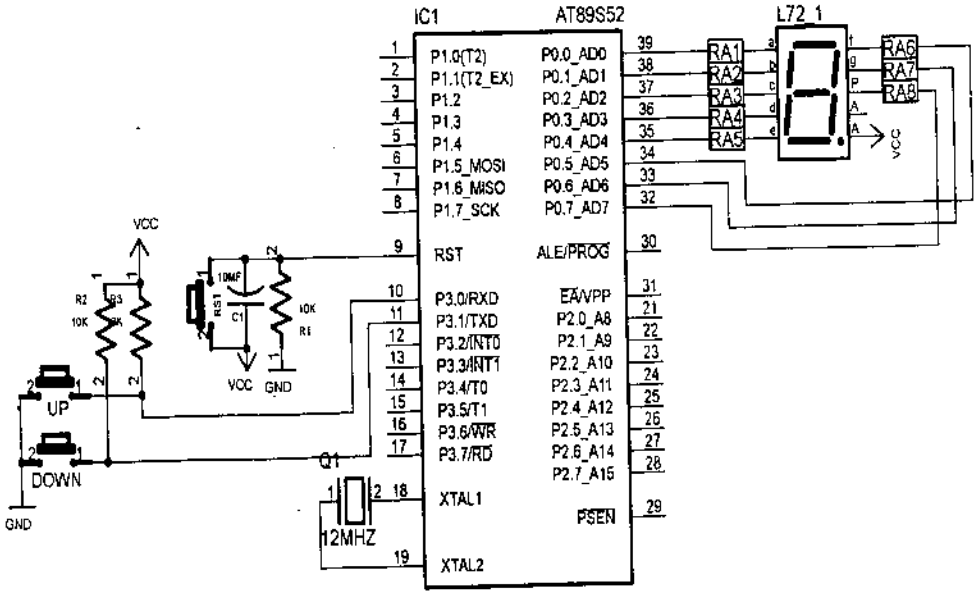
➢ ***Giải thích lưu đồ:***

Chương trình chính sau khi khởi tạo các thông số thì gọi hàm kiểm tra xử lý phím UP, rồi gọi hàm kiểm tra xử lý phím DOWN.

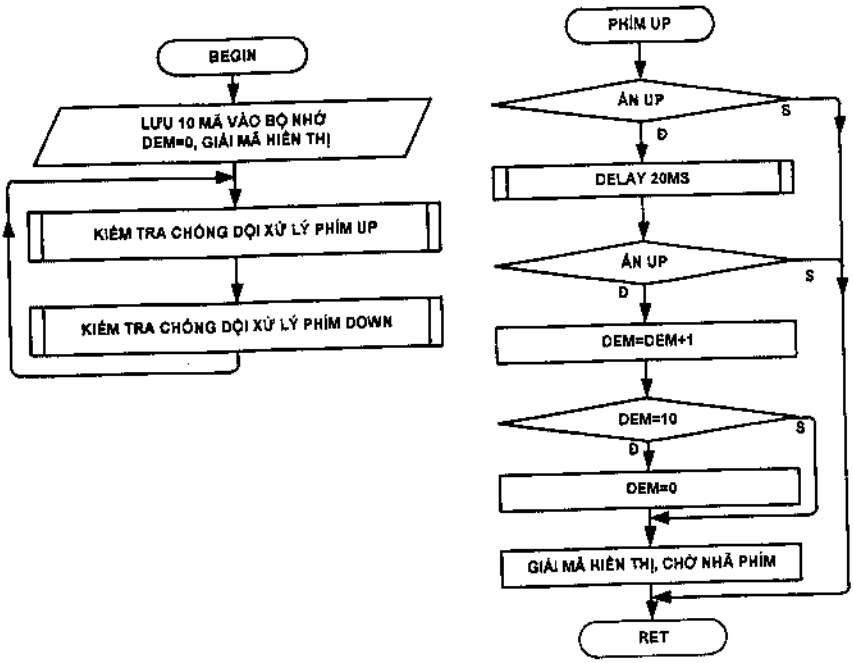
Hàm xử lý phím UP tiến hành kiểm tra xem có nhấn phím UP hay không, nếu không thì thoát, nếu có thì delay 20ms rồi kiểm tra lần 2, nếu còn

thì tiến hành tăng giá trị của biến DEM, kiểm tra biến DEM nếu bằng 10 thì cho biến đếm về 0, giải mã hiển thị ra led, kiểm tra buông phím mới thoát.

Hàm xử lý phím DOWN cũng tương tự.



Hình 6-24: *Sơ đồ kết nối port0 điều khiển một led 7 đoạn, hai nút nhấn.*



Hình 6-25: *Lưu đồ tăng giảm giá trị trên led bằng hai phím UP-DOWN.*

➢ ***Chương trình Keil-C:***

```
#include <AT89X52.h>
#define UP      P3_0
#define DOWN    P3_1
unsigned char MA7D[10] =
{0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90};
signed char DEM;
void delay(unsigned int y)
{     unsigned int i;
 for(i=0;i<y;i++){;}
}
void CHONGDOI_UP ()
{     if (UP == 0)
            {     delay (10000);
                  if (UP == 0)
                        {     DEM++;
                              if (DEM==10) DEM =0;
                              P0 = MA7D[DEM];
                              do    {}
                              while (UP == 0);
}           }           }
void CHONGDOI_DOWN ()
{     if (DOWN == 0)
            {     delay (10000);
                  if (DOWN == 0)
                        {     DEM--;
                              if (DEM==-1) DEM =9;
                              P0 = MA7D[DEM];
```

```
                    do    { }
                    while (DOWN == 0);
}}}

void MAIN ()
{     DEM=0;          P0 = MA7D[DEM];
      while(1)
            {     CHONGDOI_DOWN ();     CHONGDOI_UP ();}
}
```

Trong chương trình trên, mỗi lần nhấn và nhả phím thì giá trị trên led tăng hoặc giảm 1, nếu muốn khi nhấn và giữ thì giá trị tăng hoặc giảm cho đến khi nhả phím thì ngừng, chương trình được hiệu chỉnh lại như sau:

```
#include <AT89X52.h>
#define UP        P3_0
#define DOWN  P3_1
unsigned char MA7D[10] =
{0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90};
signed char DEM;
void delay(unsigned int y)
{     unsigned int i;
 for(i=0;i<y;i++){;}
}
void CHONGDOI_UP ()
{     if (UP == 0)
            {     delay (10000);
                  if (UP == 0)
                        {     DEM++;
                              if (DEM==10) DEM =0;
                              P0 = MA7D[DEM];
```

```
                        delay (20000);
}       }       }
void CHONGDOI_DOWN ()
{   if (DOWN == 0)
        {   delay (10000);
            if (DOWN == 0)
                {   DEM--;
                    if (DEM==-1) DEM =9;
                    P0 = MA7D[DEM];
                    delay (20000);
}}}

void MAIN ()
{   DEM=0;      P0 = MA7D[DEM];
    while(1)
        {   CHONGDOI_DOWN ();   CHONGDOI_UP ();}
}
```

Trong chương trình này bỏ đi phần chờ nhả phím và thêm vào chương trình con delay để làm chậm lại.

➢ ***Hệ thống nhiều phím***

Với cách 1 thì mỗi phím sử dụng một ngõ vào để kết nối, 16 phím sẽ dùng 16 ngõ vào – tốn nhiều đường tín hiệu.

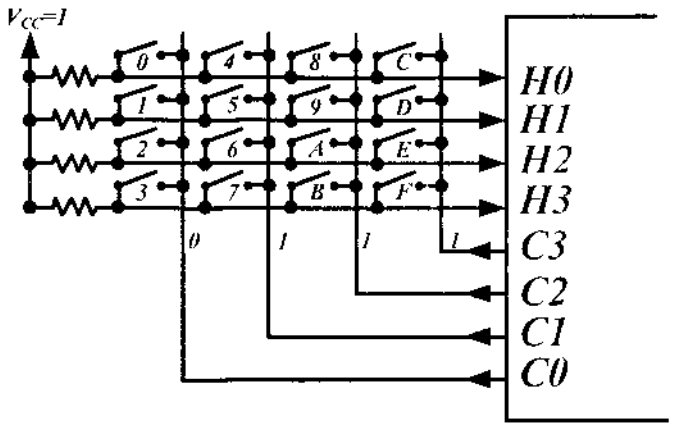
Cách kết nối dạng ma trận thì 16 phím chỉ dùng 8 tín hiệu: 4 cho hàng và 4 cho cột – gọi là ma trận 4×4 = 16 phím, ví dụ ma trận 8×8 = 64 phím.

Tổng quát ma trận bàn phím m×n thì số đường tín hiệu bằng "m+n", số phím bằng "m×n"

- Ưu điểm của cách kết nối theo dạng ma trận là tiết kiệm đường điều khiển.
- Khuyết điểm: chương trình phức tạp.

➢ ***Khảo sát bàn phím ma trận 4×4 = 16 phím:***

Sơ đồ mạch bàn phím như hình 6-26:



Hình 6-26: *Bàn phím ma trận 4×4.*

Trong ma trận 4×4 thì có 4 hàng và 4 cột, hàng được chọn là tín hiệu vào – cột được chọn là tín hiệu ra, hàng thì treo lên nguồn Vcc qua điện trở – nên mức logic của hàng luôn là mức 1.

Các phím nhấn thường hở nên 4 hàng luôn ở mức 1 hay $H_3H_2H_1H_0$ =1111.

Cột là tín hiệu ra nên chúng ta điều khiển xuất dữ liệu ra cột tùy ý.

Để phân biệt các phím thì mỗi phím có một tên được đánh theo số thập lục phân từ '0' đến 'F'.

Để xem có phím nào nhấn hay không ta tiến hành quét từng cột bằng cách cho một cột ở mức 0, 3 cột còn lại ở mức 1 và kiểm tất cả các tra hàng, nếu tất cả các hàng vẫn ở mức logic 1 tức là không có nhấn phím, nếu có một hàng xuống mức 0 thì đã có nhấn phím. Cụ thể như sau:

➢ ***Quét cột thứ 0:***

Xuất dữ liệu ra cột là: $C_3C_2C_1C_0$ =1110 như hình 6-26, kiểm tra hàng $H_3H_2H_1H_0$ xem có bằng 1111 hay không?

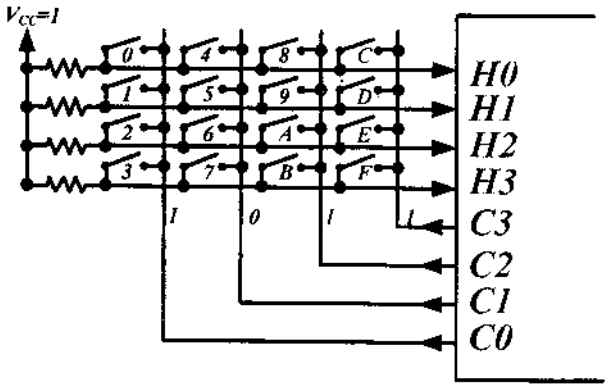
- Nếu bằng 1110 thì đã nhấn phím số '0'.
- Nếu bằng 1101 thì đã nhấn phím số '1'.
- Nếu bằng 1011 thì đã nhấn phím số '2'.
- Nếu bằng 0111 thì đã nhấn phím số '3'.

- Nếu bằng 1111 thì không có phím nhấn nào ở cột C_0 được nhấn thì phải quét cột tiếp theo.

➢ ***Quét cột thứ nhất:***

Xuất dữ liệu ra cột là: $C_3C_2C_1C_0$ =1101 như hình 6-27, kiểm tra hàng $H_3H_2H_1H_0$ xem có bằng 1111 hay không?

- Nếu bằng 1110 thì đã nhấn phím số '4'.
- Nếu bằng 1101 thì đã nhấn phím số '5'.
- Nếu bằng 1011 thì đã nhấn phím số '6'.
- Nếu bằng 0111 thì đã nhấn phím số '7'.
- Nếu bằng 1111 thì không có phím nhấn nào ở cột C_1 được nhấn thì phải quét cột tiếp theo.

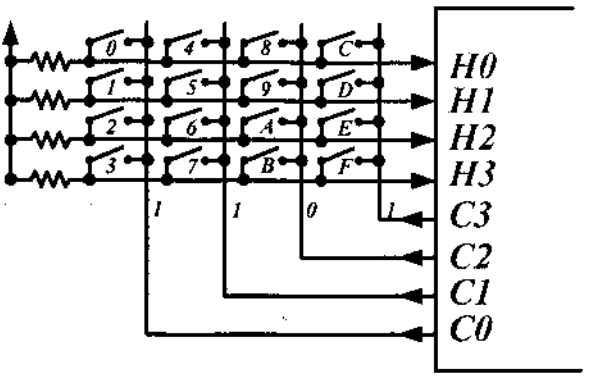


Hình 6-27: *Bàn phím ma trận 4×4 với cột C_1 bằng 0.*

➢ ***Quét cột thứ hai:***

Xuất dữ liệu ra cột là: $C_3C_2C_1C_0$ =1011 như hình 6-28, kiểm tra hàng $H_3H_2H_1H_0$ xem có bằng 1111 hay không?

- Nếu bằng 1110 thì đã nhấn phím số '8'.
- Nếu bằng 1101 thì đã nhấn phím số '9'.
- Nếu bằng 1011 thì đã nhấn phím số 'A'.
- Nếu bằng 0111 thì đã nhấn phím số 'B'.
- Nếu bằng 1111 thì không có phím nhấn nào ở cột C_2 được nhấn thì phải quét cột tiếp theo.

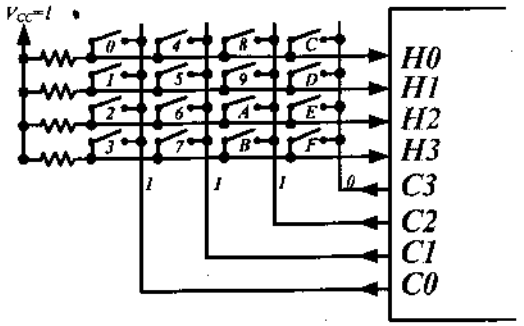


Hình 6-28: *Bàn phím ma trận 4×4 với cột C_2 bằng 0.*

➢ ***Quét cột thứ ba:***

Xuất dữ liệu ra cột là: $C_3C_2C_1C_0 = 0111$ như hình 6-29, kiểm tra hàng $H_3H_2H_1H_0$ xem có bằng 1111 hay không?

- Nếu bằng 1110 thì đã nhấn phím số 'C'.
- Nếu bằng 1101 thì đã nhấn phím số 'D'.
- Nếu bằng 1011 thì đã nhấn phím số 'E'.
- Nếu bằng 0111 thì đã nhấn phím số 'F'.
- Nếu bằng 1111 thì không có phím nhấn nào ở cột C3 đến đây xem như kết thúc.

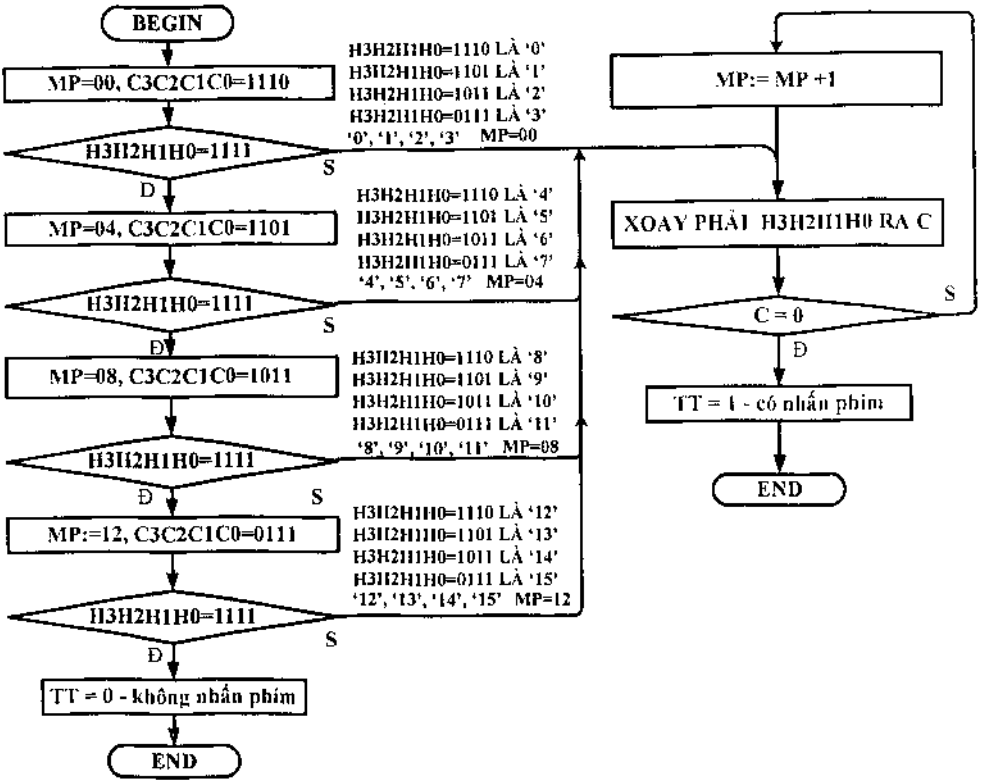


Hình 6-29: *Bàn phím ma trận 4×4 với cột C_3 bằng 0.*

Tên các phím để chúng ta phân biệt, còn chương trình phân biệt các phím bằng mã, mỗi phím có một mã phím do phần mềm lập trình xây dựng, để đơn giản mỗi phím được đặt mã như sau:

Phím số '0' có mã là 00H, phím số '1' có mã là 01H, ... phím 'F' có mã là 0FH.

Hoạt động quét bàn phím được thực hiện theo lưu đồ như hình 6-30.



***Hình 6-30:** Lưu đồ quét bàn phím ma trận 4×4.*

➢ ***Giải thích lưu đồ:***

Cho mã phím (MP) bắt đầu bằng 00H, cho cột C_0= 0, kiểm tra 4 hàng xem có bằng 1111 không?

Nếu có thì đã nhấn một trong bốn phím từ '0' hoặc '1' hoặc '2' hoặc '3' và rẽ nhánh theo hướng tìm xem hàng nào bằng không để biết phím nào đã nhấn.

Nếu không nhấn thì gán mã phím bằng 04H, gán bằng 04H là do các phím từ '0' đến '3' đã sử dụng các mã từ 00H đến 03H. Cho cột C_1= 0 để quét các phím tiếp theo.

Tương tự quét cho đến khi hết cột mà không có phím nào nhấn thì gán biến trạng thái TT bằng 0 để cho biết không có phím nhấn.

Khi có nhấn phím, tiến hành xoay hàng sang phải qua cờ C, sau đó kiểm tra C có bằng 0 hay không, nếu chưa bằng 0 thì quay lại tăng mã phím

lên 1 và xoay tiếp cho đến khi C bằng 0 thì gán biến TT bằng 1 cho biết có nhấn phím và thoát với mã phím chứa trong biến MP.

Ví dụ 6-10: Giả sử khi quét phím và có nhấn phím số '3' thì mã hàng là $H_3H_2H_1H_0 = 0111$, mã phím MP bằng 00.

Bảng 6-2: Quá trình tìm mã phím nhấn phím số '3'.

	$H_3H_2H_1H_0$	Cờ C	Mã phím MP
Giá trị ban đầu	0111	x	00
Xoay lần 1	X011	1	Tăng mã phím lên 1, MP = 01
Xoay lần 2	XX01	1	Tăng mã phím lên 1, MP = 02
Xoay lần 3	XXX0	1	Tăng mã phím lên 1, MP = 03
Xoay lần 4	XXXX	0	Kết thúc với mã phím MP = 03

Ví dụ 6-11: Giả sử khi quét phím và có nhấn phím số '6' thì mã hàng là $H_3H_2H_1H_0 = 0111$, mã phím MP bằng 04.

Bảng 6-3: Quá trình tìm mã phím nhấn phím số '6'.

	$H_3H_2H_1H_0$	Cờ C	Mã phím MP
Giá trị ban đầu	1011	x	04
Xoay lần 1	X101	1	Tăng mã phím lên 1, MP = 05
Xoay lần 2	XX10	1	Tăng mã phím lên 1, MP = 06
Xoay lần 3	XXX1	0	Kết thúc với mã phím MP = 06

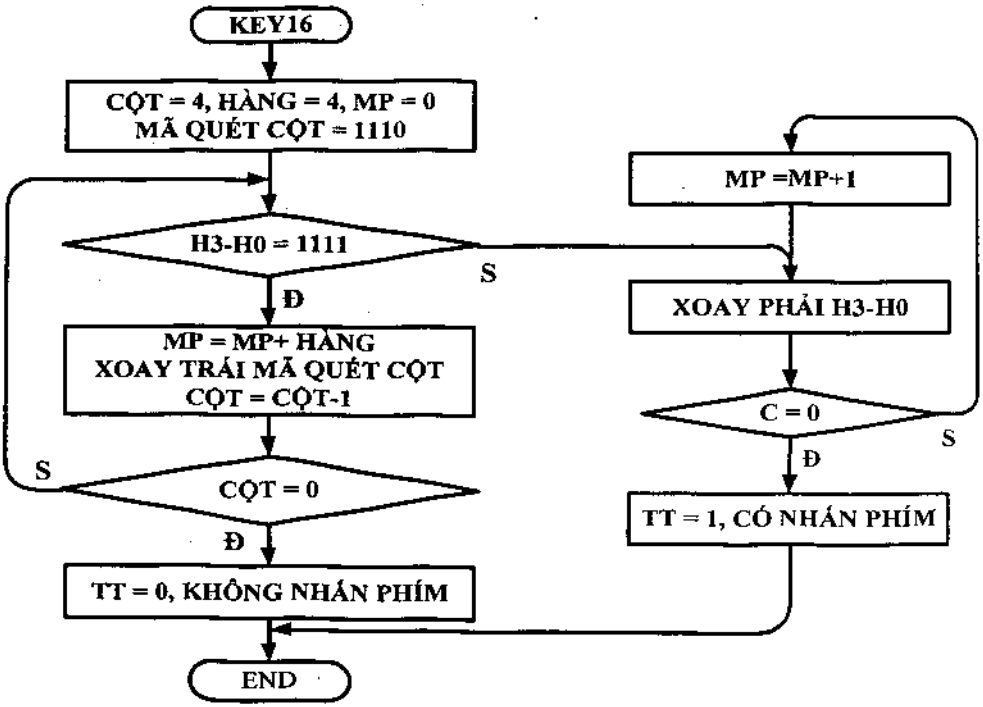
Lưu đồ trên giúp các bạn đọc hiểu và lưu đồ sau sẽ rút gọn hơn và phù hợp khi viết chương trình.

➢ ***Giải thích lưu đồ:***

Do ma trận bàn phím là 4×4 nên gán biến cột bằng 4 và biến hàng bằng 4, mã phím xuất phát là 00H, có 4 cột nên mã quét cột bằng 1110.

Kiểm tra hàng H3-H0 có bằng 1111 hay không?

Nếu có thì tìm mã phím giống như đã trình bày, nếu không có phím nhấn thì tăng mã phím lên 4 bằng với số hàng đã gán, xoay mã quét cột sang trái để quét cột tiếp theo, giảm biến cột đi 1 và kiểm tra xem đã quét hết chưa, nếu chưa thì quay lại quét tiếp, nếu hết thì kết thúc.



Hình 6-31: *Lưu đồ quét bàn phím ma trận 4×4 rút gọn.*

Ví dụ 6-12: Hãy hiệu chỉnh lưu đồ trên để quét bàn phím ma trận 8×8.

Với bàn phím ma trận 8×8 thì số lượng đường tín hiệu bằng 8 + 8 bằng 16, số lượng phím là 8×8 bằng 64 phím, các thay đổi như sau:

- Tên chương trình nên đặt là 'KEY64'.
- Biến cột gán bằng 8, biến hàng gán bằng 8.
- Biến lưu mã quét cột có giá trị bắt đầu là "11111110".
- Tên của 8 hàng là H7-H0 và giá trị so sánh là "11111111".

Ví dụ 6-13: Hãy hiệu chỉnh lưu đồ trên để quét bàn phím ma trận 6×4.

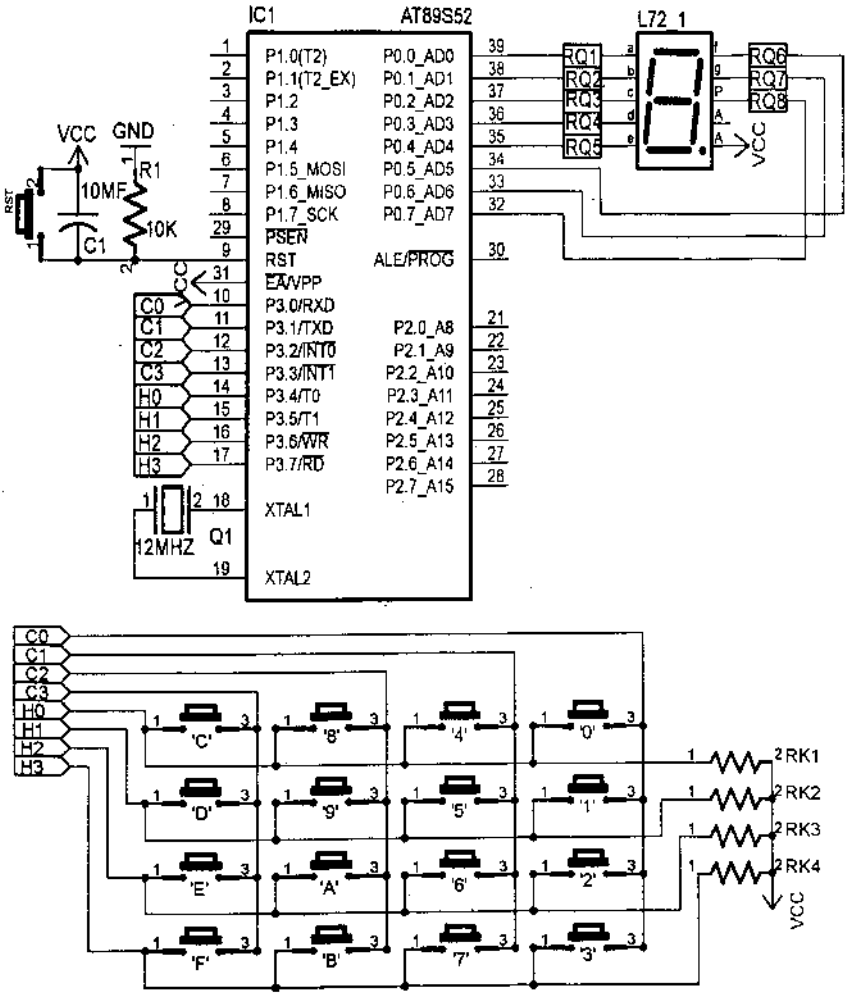
Với bàn phím ma trận 6×4 thì số lượng đường tín hiệu bằng 6 + 4 bằng 12, số lượng phím là 6×4 bằng 24 phím, các thay đổi như sau:

- Tên chương trình nên đặt là 'KEY24'.
- Biến cột gán bằng 4, biến hàng gán bằng 6.
- Biến lưu mã quét cột có giá trị bắt đầu là "1110".
- Tên của 6 hàng là H5-H0 và giá trị so sánh là "111111".

Để hiểu chương trình quét bàn phím ma trận ta khảo sát bài ứng dụng theo sau.

Bài 6-14: Dùng vi điều khiển AT89S52 giao tiếp với một led 7 đoạn và ma trận phím 4x4. Viết chương trình quét bàn phím ma trận, hiển thị tên của phím trên led 7 đoạn.

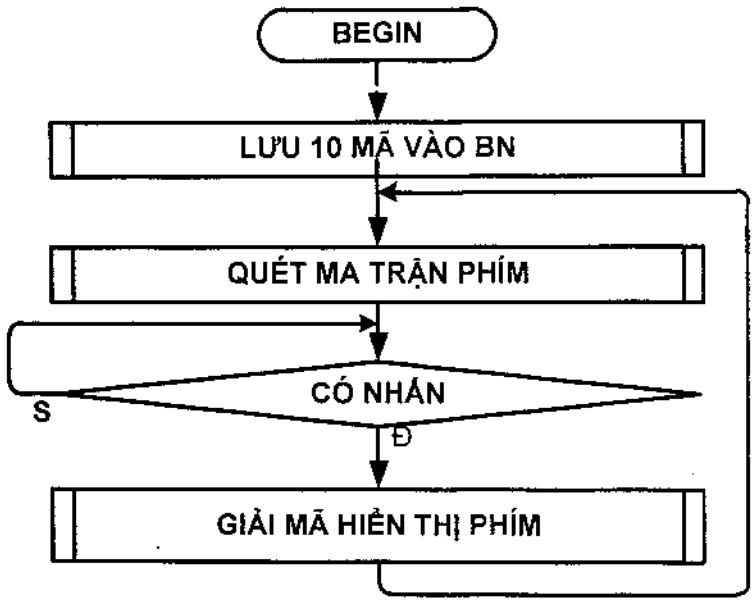
Sơ đồ mạch:



Hình 6-31: *Vi điều khiển giao tiếp bàn phím ma trận.*

Trong sơ đồ dùng port3 giao tiếp với bàn phím ma trận 4x4 và port0 điều khiển một led 7 đoạn.

Lưu đồ:



Hình 6-32: Lưu đồ quét hiển thị mã của bàn phím ma trận.

➢ **Chương trình Keil-C:**

```
#include <AT89X52.h>
#define KEYPORT P3
sbit HANG0 =    KEYPORT^4;
sbit HANG1 =    KEYPORT^5;
sbit HANG2 =    KEYPORT^6;
sbit HANG3 =    KEYPORT^7;

#define LEDPORT P1
int    MAQUETCOT,MAPHIM,HANG,COT;
bit    TT_KEY,TT_NHA;
char X;
unsigned char MA7D[16] =
{0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90,0x88, 0x83,
0xC6, 0xA1, 0x86, 0x8E};
```

```
void KEY_4X4();
void KEY_NHAN();
void KEY_NHA();
void KEY_NHA()
{
do   {TT_NHA = HANG0 & HANG1 & HANG2 & HANG3;}
while (TT_NHA ==0);
}
void KEY_4X4()
{ KEY_NHAN();
if (TT_KEY==1)
{          MAPHIM = COT*4 + HANG; LEDPORT = MA7D[MAPHIM];
           KEY_NHA();                    //CHO NHA PHIM NHAN
}
}
void KEY_NHAN()
{ MAQUETCOT = 0xFE; COT=0;       TT_KEY=0;
do
{ KEYPORT=MAQUETCOT;
if         (HANG0==0) {HANG=0;   TT_KEY=1;      break;}
else if    (HANG1==0) {HANG=1;   TT_KEY=1;      break;}
else if    (HANG2==0) {HANG=2;   TT_KEY=1;      break;}
else if    (HANG3==0) {HANG=3;   TT_KEY=1;      break;}
else
     { MAQUETCOT = (MAQUETCOT<<1) + 0x1;
     COT++; }
} while (COT!=4);
}
```

```
void MAIN ()
{           LEDPORT=0x7F;
while (1)
{ KEY_4X4(); }
}
```

❖ **Giải thích chương trình:**

Lệnh "LEDPORT=0x7F;" có chức năng làm dấu chấm của led 7 đoạn sáng để báo hiệu.

Lệnh "KEY_4X4();" có chức năng gọi chương trình con quét phím ma trận.

Trong chương trình con KEY_4X4() có gọi chương trình con "KEY_NHAN();".

Lệnh "if (TT_KEY==1) ... }" có chức năng kiểm tra biến trạng của phím có nhấn hay không, nếu có thì tiến hành tạo mã phím bằng cách lấy cột nhân với 4 rồi cộng với hàng gán cho mã phím. Lệnh tiếp giải mã 7 đoạn cho mã phím và xuất ra port để hiển thị trên led. Sau đó tiến hành chống dội bằng cách kiểm tra nhả phím đã nhấn hay chưa bằng cách gọi chương trình con "KEY_NHA();".

Giả sử ta nhấn phím số 7 thì cột sẽ bằng 1, hàng bằng 3, khi đó mã phím sẽ bằng 1 nhân 4 cộng với 3 nên bằng 7.

➢ ***Chương trình con quét phím: KEY_NHAN()***

Lệnh "MAQUETCOT = 0xFE;" có chức năng gán dữ liệu "FE = 11111110" cho biến mã quét cột.

Lệnh "COT=0;" có chức năng gán biến cột bắt đầu từ cột thứ 0.

Lệnh "TT_KEY=0;" có chức năng gán biến trạng thái của phím ban đầu bằng 1.

Thực hiện vòng lặp do while có điều kiện chấm dứt khi có nhấn phím hoặc khi đã quét hết 4 cột.

Lệnh "KEYPORT=MAQUETCOT;" có chức năng xuất mã quét cột ra port giao tiếp với bàn phím ma trận.

Lệnh "if (HANG0 == 0) {HANG=0;TT_KEY=1;break;}" có chức năng kiểm tra hàng thứ 0 có bằng 0 hay không, nếu bằng 0 thì đã nhấn

phím, tiến hành gán biến hàng (HANG) bằng 0, gán biến trạng thái phím (TT_KEY) bằng 1, ngừng kiểm tra để thoát khỏi chương trình.

Nếu không có phím nào của hàng thứ 0 thì kiểm tra phím của hàng thứ 1. Nếu có nhấn thì các lệnh cũng làm tương tự, chỉ khác là gán biến hàng bằng 1.

Tương tự cho hai hàng 2 và 3.

Nếu cả 4 hàng không có phím nào nhấn thì xoay mã quét cột sang cột kế bằng lệnh "MAQUETCOT= (MAQUETCOT<<1) + 0x1;" khi xoay thì đẩy số 0 vào, do chỉ cho phép 1 cột bằng 0 nên ta phải cộng với 1 để bit 0 mới xoay vào sẽ bằng 1.

Ví dụ sau khi xoay thì dữ liệu sẽ là "1100", cộng với 1 để trở thành "1101".

Tăng biến cột lên 1 và làm tiếp cho đến khi nhấn phím hoặc hết cột thì thoát.

➢ ***Chương trình con kiểm tra nhả phím: KEY_NHA()***

Thực hiện vòng lặp do while, trong vòng lặp tiến hành and các bit của hàng với nhau, khi có nhấn phím và còn nhấn thì làm hàng xuống mức 0 nên khi and các hàng với nhau thì kết quả bằng 0, vòng lặp do while tiếp tục thực hiện cho đến khi kết quả and với nhau bằng 1, có nghĩa là đã nhả phím đã nhấn.

Sau khi nhả phím đã nhấn ta mới được phép nhấn phím tiếp theo.

Chương trình quét phím khác với lưu đồ ở phần kiểm tra có nhấn phím hay không, trong chương trình thì kiểm tra từng hàng, còn lưu đồ thì kiểm tra 4 hàng và xoay tìm hàng nào bằng 0.

V. GIAO TIẾP VI ĐIỀU KHIỂN AT89S52 VỚI LCD

1. Giới thiệu LCD

Giao tiếp với led 7 đoạn có hạn chế vì chỉ hiển thị được các số từ 0 đến 9 hoặc số hex từ 0 đến F – không thể nào hiển thị được các thông tin kí tự khác, nhưng chúng sẽ được hiển thị đầy đủ trên LCD.

LCD có rất nhiều dạng phân biệt theo kích thước từ vài kí tự đến hàng chục kí tự, từ một hàng đến vài chục hàng. LCD 16×2 có nghĩa là có hai hàng, mỗi hàng có 16 kí tự. LCD 20×4 có nghĩa là có bốn hàng, mỗi hàng có 20 kí tự.

LCD 16×2 có hình dáng như hình 6-33:

Hình 6-33. Hình của LCD

2. Sơ đồ chân của LCD

LCD có nhiều loại và số chân của chúng cũng khác nhau nhưng có hai loại phổ biến là loại 14 chân và loại 16 chân, sự khác nhau là các chân nguồn cung cấp, còn các chân điều khiển thì không thay đổi, khi sử dụng loại LCD nào thì phải tra datasheet của chúng để biết rõ các chân.

Bảng 6-4: Các chân của LCD:

Thứ tự	Tên tín hiệu	I/O	Mô tả
1	V_{SS}	Nguồn	GND
2	V_{DD}	Nguồn	+5V
3	Vo	Điện áp	Điều khiển ánh sáng nền
4	RS	INPUT	Register select
5	R/W	INPUT	Read/Write
6	E	INPUT	Enable (strobe)
7	D0	I/O	DATA LSB
8	D1	I/O	DATA
9	D2	I/O	DATA
10	D3	I/O	DATA
11	D4	I/O	DATA
12	D5	I/O	DATA
13	D6	I/O	DATA
14	D7	I/O	DATA MSB
15	A	I	Nguồn dương +5V
16	K	I	GND

Trong 16 chân của LCD được chia ra làm 3 dạng tín hiệu như sau:

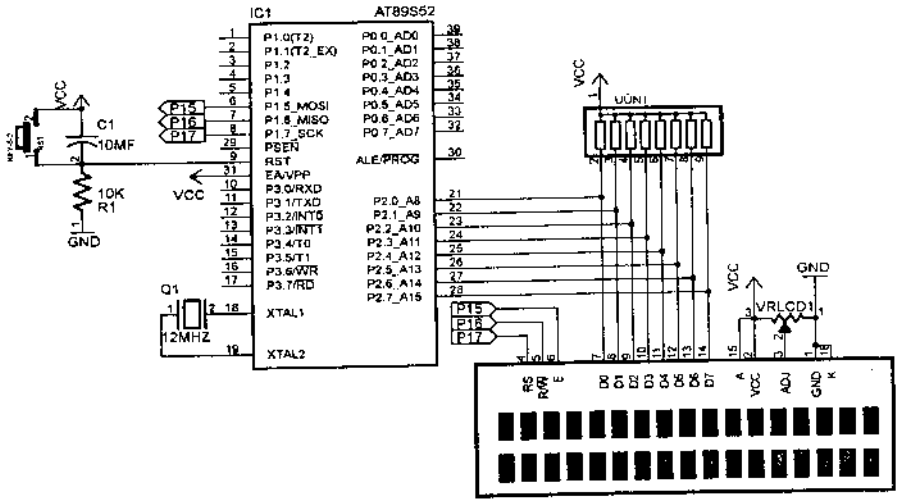
Các chân cấp nguồn: Chân số 1 là chân nối mass (0V), chân thứ hai là Vdd nối với nguồn +5V. Chân thứ ba dùng để chỉnh contrast thường nối với biến trở. Hai chân A và K dùng để cấp nguồn cho đèn nền để có thể nhìn thấy vào ban đêm.

Các chân điều khiển: Chân số 4 là chân RS dùng để điều khiển lựa chọn thanh ghi. Chân R/W dùng để điều khiển quá trình đọc và ghi. Chân E là chân cho phép dạng xung chốt.

Các chân dữ liệu D7÷D0: Chân số 7 đến chân số 14 là 8 chân dùng để trao đổi dữ liệu giữa thiết bị điều khiển và LCD.

3. Sơ đồ mạch giao tiếp vi điều khiển với LCD

Trong phần này sẽ trình bày phần giao tiếp vi điều khiển AT89S52với LCD như hình 6-34:



Hình 6-34. *Giao tiếp vi điều khiển AT89S52 với LCD.*

Chú ý: chỉnh biến trở sao cho các kí tự hiển thị trên LCD thì dừng lại.

4. Các lệnh điều khiển LCD

Để điều khiển LCD thì có các IC chuyên dùng được tích hợp bên dưới LCD có mã số 447801 đến các IC 447809. Trong IC này có bộ nhớ RAM dùng để lưu trữ dữ liệu cần hiển thị và thực hiện việc điều khiển LCD hiển thị.

Bảng 6-5: Các lệnh điều khiển bao gồm các lệnh được liệt kê như sau:

Lệnh xoá màn hình "Clear Display": khi thực hiện lệnh này, LCD sẽ bị xoá và bộ đếm địa chỉ được xoá về 0.

Lệnh di chuyển con trỏ về đầu màn hình "Cursor Home": khi thực hiện lệnh này, bộ đếm địa chỉ được xoá về 0, phần hiển thị trở về vị trí gốc đã bị dịch trước đó. Nội dung bộ nhớ RAM hiển thị DDRAM không bị thay đổi.

Lệnh thiết lập lối vào "Entry mode set": lệnh này dùng để thiết lập lối vào cho các kí tự hiển thị, bit ID = 1 thì con trỏ tự động tăng lên 1 mỗi khi có 1 byte dữ liệu ghi vào bộ hiển thị, khi ID = 0 thì con trỏ sẽ không tăng: dữ liệu mới sẽ ghi đè lên dữ liệu cũ. Bit S = 1 thì cho phép dịch chuyển dữ liệu mỗi khi nhận 1 byte hiển thị.

LỆNH	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	Mô tả	clock
NOP	0	0	0	0	0	0	0	0	0	0	No operation	0
Clear display	0	0	0	0	0	0	0	0	0	1	Clear display & sets address counter to zero	165
Cursor home	0	0	0	0	0	0	0	0	1	0	Sets address counter to zero, returns shifted display to original position. DDRAM contens remain unchanged.	3
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction, and specifies automatic shift.	3
Display control	0	0	0	0	0	0	1	D	C	B	Turns display (D), cursor on/off (C) or cursor blinking (B).	3
Cursor/ display shift	0	0	0	0	0	1	S/C	R/L	0	0	Move cursor and shift display. DDRAM contents remain unchanged.	3
Function set	0	0	0	0	1	DL	N	F	0	0	Sets interface data width (DL), number of display	3

											lines (N) and font (F).	
Set CGRAM addr	0	0	0	1	Character Generator RAM						Sets CGRAM addres	3
Set DDRAM addr	0	0	1	Display data ram address							Sets DDRAM addres	3
Buzy flag & Addr	0	0	BF	Address counter							Reads buzy flag & address counter	0
Read data	1	0	Read data								Reads data from CCGRAM or DDRAM	3
Read data	1	1	Write data								Write data to CCGRAM or DDRAM	3

Lệnh điều khiển con trỏ hiển thị "Display Control": lệnh này dùng để điều khiển con trỏ (cho hiển thị thì bit D = 1, tắt hiển thị thì bit D = 0), tắt mở con trỏ (mở con trỏ thì bit C = 1, tắt con trỏ thì bit C = 0), và nhấp nháy con trỏ (cho nhấp nháy thì bit B = 1, tắt thì bit B = 0).

Lệnh di chuyển con trỏ "Cursor /Display Shift": lệnh này dùng để điều khiển di chuyển con trỏ hiển thị dịch chuyển (SC = 1 cho phép dịch chuyển, SC = 0 thì không cho phép), hướng dịch chuyển (RL = 1 thì dịch phải, RL = 0 thì dịch trái). Nội dung bộ nhớ DDRAM vẫn không đổi.

Lệnh thiết chức năng "Function set": lệnh này dùng để thiết lập chức năng giao tiếp, bit DL (data length) = 1 thì cho phép giao tiếp 8 đường data D7 ÷ D0, nếu bằng 0 thì cho phép giao tiếp 4 đường D7 ÷ D4 để tiết kiệm tín hiệu giao tiếp. Bit N (number of line) = 1 thì cho phép hiển thị hai hàng, nếu bằng 0 thì cho phép hiển thị 1 hàng. Bit F (font) = 1 thì cho phép hiển thị với ma trận 5×10, nếu bằng 0 thì cho phép hiển thị với ma trận 5×7.

Lệnh thiết lập địa chỉ cho bộ nhớ RAM phát kí tự "Set CGRAM Addr": lệnh này dùng để thiết lập địa chỉ cho bộ nhớ RAM phát kí tự.

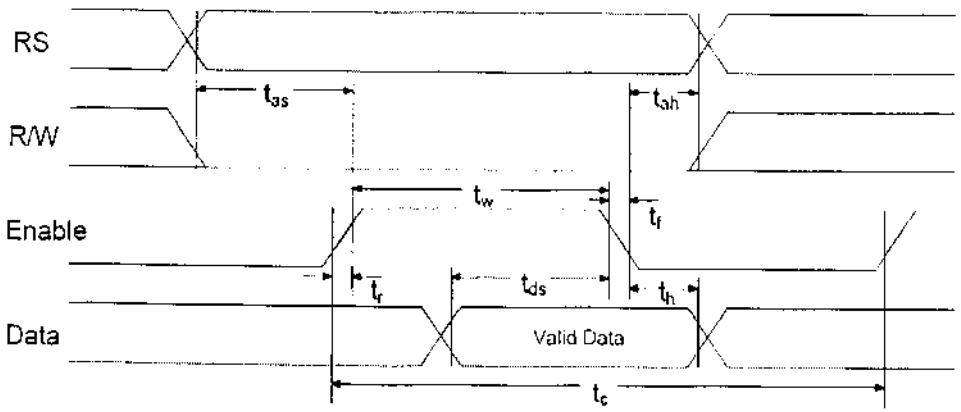
Lệnh thiết lập địa chỉ cho bộ nhớ RAM hiển thị "Set DDRAM Addr": lệnh này dùng để thiết lập địa chỉ cho bộ nhớ RAM lưu trữ các dữ liệu hiển thị.

Hai lệnh cuối cùng là lệnh đọc và lệnh ghi dữ liệu LCD.

Dạng sóng các tín hiệu khi thực hiện ghi dữ liệu vào LCD như hình 6-35:

Nhìn vào dạng sóng ta có thể thấy được trình tự điều khiển như sau:

- Điều khiển tín hiệu RS.
- Điều khiển tín hiệu R/W xuống mức thấp.
- Điều khiển tín hiệu E lên mức cao để cho phép.
- Xuất dữ liệu D7÷D0.
- Điều khiển tín hiệu E về mức thấp.
- Điều khiển tín hiệu R/W lên mức cao trở lại.



Hình 6-35. *Dạng sóng điều khiển của LCD.*

5. Địa chỉ của từng kí tự trên LCD

LCD16x2 có hai hàng mỗi hàng 16 kí tự.

Hàng thứ nhất: kí tự tận cùng bên trái có địa chỉ là 0x80, kí tự kế là 0x81, kí tự cuối cùng là 0x8F. Hàng thứ hai: kí tự tận cùng bên trái có địa chỉ là 0xC0, kí tự kế là 0xC1, kí tự cuối cùng là 0xCF.

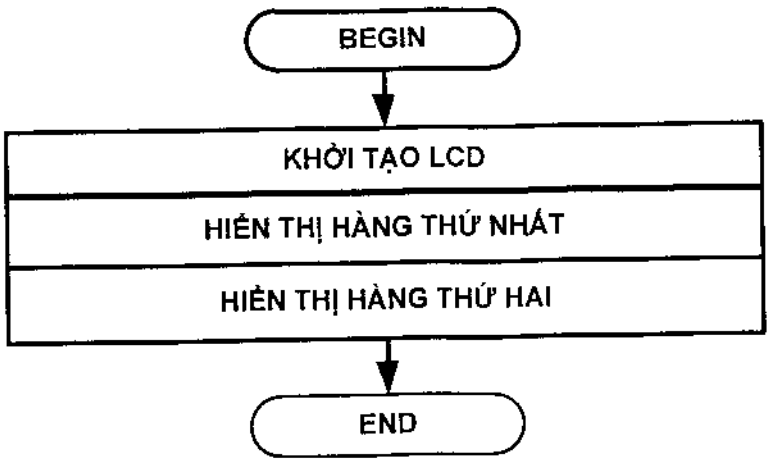
Bảng 6-6: *địa chỉ của từng kí tự:*

Địa chỉ	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
Kí tự hàng 1		D	H		S	P		K	Y		T	H	U	A	T	
Kí tự hàng 2		T	P		H	O		C	H	I		M	I	N	H	
Địa chỉ	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF

6. Các chương trình hiển thị trên LCD

Bài 6-15: Sử dụng mạch giao tiếp vi điều khiển AT89S52 với LCD 16×2 ở hình 6-29, hãy viết chương trình hiển thị hai hàng thông tin như trong bảng 6-6.

- Sơ đồ mạch: giống như hình 6-36.
- Lưu đồ:



Hình 6-36*: Khởi tạo LCD và hiển thị thông tin trên hai hàng.*

- ***Chương trình Keil-C:***

```
#include <AT89X52.H>
#define LCD_E          P1_5
#define LCD_RW         P1_6
#define LCD_RS         P1_7
#define LCD_DB         P2
#define function_set          0x38
#define display_control       0x0F
#define clear_display         0x01
#define entry_mode            0x06
#define addr_line1            0x80
#define addr_line2            0xC0
unsigned char x, j;
```

```
const unsigned char HANG1[16] = {" DH SP KY THUAT "};
const unsigned char HANG2[16] = {" TP HO CHI MINH "};
void delay(unsigned int x)
{     unsigned int y;
      for (y=0;y<x;y++) {· }
}
void COMMAND_WRITE(unsigned char CMD)
      { LCD_DB=CMD;     LCD_RS=0;     LCD_E=1;     LCD_E=0;
      delay(10);  }
void DATA_WRITE ( unsigned char DTDPL)
      { LCD_DB=DTDPL;  LCD_RS=1;     LCD_E=1;     LCD_E=0;
      delay(10);}

void SETUP_LCD( )
      {     LCD_E=1;       LCD_RW=0;
            COMMAND_WRITE(function_set);     delay(5000);
            COMMAND_WRITE(display_control);
            COMMAND_WRITE(clear_display);    delay(1000);
            COMMAND_WRITE(entry_mode);
            }

void main ( )
      {     SETUP_LCD( );
            COMMAND_WRITE(addr_line1);       delay(10);
            for( j=0;j<16;j++)       {DATA_WRITE(HANG1[j]);}

            COMMAND_WRITE(addr_line2);       delay(10);
            for( j=0;j<16;j++)       {DATA_WRITE(HANG2[j]);}
            while (1){ }
}
```

Để các chương trình có dùng LCD cho đơn giản thì các chương trình con khởi tạo sẽ viết thành một file riêng với tên file là "THUVIEN_LCD16X2.C" và lưu vào một thư mục của phần mềm Keil: "C:\Keil\C51\INC\Atmel".

Nội dung của file như sau:

```
#ifndef     LCD_E
#define     LCD_E    P1_5
#endif
#ifndef     LCD_RW
  #define   LCD_RW  P1_6
#endif
#ifndef     LCD_RS
  #define   LCD_RS   P1_7
#endif
#ifndef     LCD_DB
  #define   LCD_DB   P2
#endif

#define function_set          0x38
#define display_control       0x0F
#define clear_display         0x01
#define entry_mode            0x06
#define shift_left            0x18
#define shift_right           0x1C
#define addr_line1            0x80
#define addr_line2            0xC0
#define addr_line3            0x94
#define addr_line4            0xd4
void delay(unsigned int x)
{     unsigned int y;
```

```
        for (y=0;y<x;y++) { }
}
void COMMAND_WRITE(unsigned char CMD)
    { LCD_DB=CMD;   LCD_RS=0;   LCD_E=1;   LCD_E=0;
    delay(10); }
void DATA_WRITE ( unsigned char DTDPL)
    { LCD_DB=DTDPL; LCD_RS=1;   LCD_E=1;   LCD_E=0;
    delay(10);}
void SETUP_LCD( )
    {   LCD_E=1; LCD_RW=0;
        COMMAND_WRITE(function_set);    delay(50);
        COMMAND_WRITE(display_control);
        COMMAND_WRITE(clear_display);   delay(1000);
        COMMAND_WRITE(entry_mode);
        }
```

Sau khi tạo xong file thư viện, chương trình hiển thị hai hàng theo yêu cầu của bài 6-10 còn lại như sau:

```
#include<AT89X52.H>
#include< THUVIEN_LCD16X2.C>
unsigned char x, j;
const unsigned char HANG1[16] = {"DAI HOC SU PHAM "};
const unsigned char HANG2[16] = {" TP HO CHI MINH "};
void main ( )
    {   SETUP_LCD( );
        COMMAND_WRITE(addr_line1);      delay(10);
        for( j=0;j<16;j++)      {DATA_WRITE(HANG1[j]);}
        COMMAND_WRITE(addr_line2);      delay(10);
        for( j=0;j<16;j++)      {DATA_WRITE(HANG2[j]);}
        while (1){ }
}
```

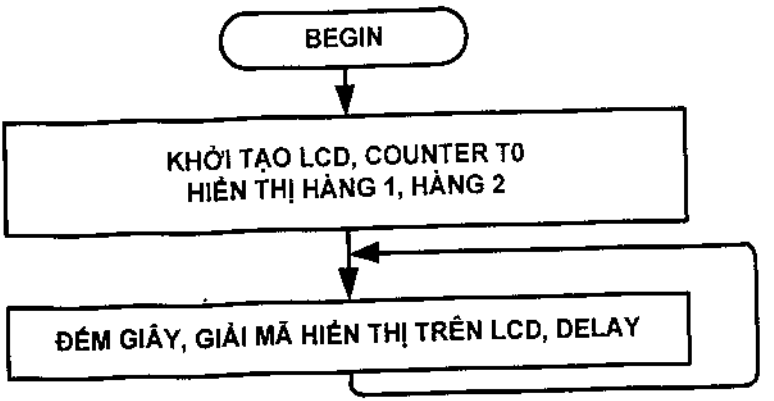
Bài 6-16: Sử dụng mạch giao tiếp vi điều khiển AT89S52 với LCD 16x2 ở hình 6-29, hãy viết chương trình hiển thị hai hàng thông tin như trong bảng 6-7. Trong đó hai ô cuối của hàng 1 sẽ hiển thị thời gian đếm giây chưa cần chính xác.

Bảng 6-7: *thông tin hiển thị hai hàng kí tự:*

Địa chỉ	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
Kí tự hàng 1	D	O	N	G		H	O	:							S	S
Kí tự hàng 2		T	P		H	O		C	H	I		M	I	N	H	
Địa chỉ	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF

Sơ đồ mạch: giống như hình 6-27.

➢ ***Lưu đồ:***



Hình 6-37: *Hiển thị thông tin trên hai hàng và đếm giây.*

➢ ***Chương trình Keil-C:***

```
#include<AT89x52.H>
#include<THUVIEN_LCD16X2.C>
```

```
unsigned char giay,x,y,j, madvigiay,machucgiay;
const unsigned char HANG1[16] = {"DONG HO:"};
const unsigned char HANG2[16] = {" TP HO CHI MINH "};
void GIAIMA_ASCII()
{       x=giay % 10;                y=giay / 10;
        madvigiay = x + 0x30;       machucgiay = y + 0x30;
}
void HIENTHI_LCD()
{       COMMAND_WRITE(0x8E);
        DATA_WRITE(machucgiay); DATA_WRITE(madvigiay);
}
void main ( )
{   SETUP_LCD( );
    COMMAND_WRITE(addr_line1);      delay(10);
    for( j=0;j<8;j++) {DATA_WRITE(HANG1[j]);}
    COMMAND_WRITE(addr_line2);      delay(10);
    for( j=0;j<16;j++)      {DATA_WRITE(HANG2[j]);}
    while (1)
        {for(giay=0; giay<60; giay++ )
            {   GIAIMA_ASCII();     HIENTHI_LCD();
    delay(10000);}
        }}
```

Các thông tin hiển thị trên LCD là mã ASCII, giá trị của giây được tách ra thành số BCD và cộng thêm với 0x30 sẽ thành mã ASCII, khi đó mới hiển thị trên LCD.

Nếu không hiển thị các thông tin mà chỉ hiển thị thời gian đếm giây thì chương trình còn lại như sau:

```
#include<AT89x52.H>
#include<THUVIEN_LCD16X2.C>
```

```
unsigned char giay,x,y,j, madvigiay,machucgiay;
void GIAIMA_ASCII()
{           x=giay % 10;                y=giay / 10;
            madvigiay = x + 0x30;       machucgiay = y + 0x30;
}
void HIENTHI_LCD()
{           COMMAND_WRITE(0x8E); DATA_WRITE(machucgiay);
            DATA_WRITE(madvigiay); }
void main ( )
{     SETUP_LCD( );
      while (1)
            {for(giay=0; giay<60; giay++ )
                  {     GIAIMA_ASCII();     HIENTHI_LCD();
      delay(10000);}
            }}
```

VI. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP

1. Câu hỏi ôn tập

Câu số 6-1: Hãy nêu các chức năng port0, 1, 2 và 3 của vi điều khiển AT89S52.

2. Câu hỏi mở rộng

Câu số 6-2: Hãy tìm hiểu các port của vi điều khiển AT89C51RD2 và so sánh với AT89S52.

Câu số 6-3: Hãy tìm hiểu các thông tin của LCD 20×4.

3. Câu hỏi trắc nghiệm

Câu 6-1: Khai báo biến "INT x" thì biến x là:

(a) 1 bit (b) 8 bit

(c) 16 bit (d) 32 bit

Câu 6-2: Khai báo biến "INT16 y" thì biến y là:

(a) 1 bit (b) 8 bit

(c) 16 bit (d) 32 bit

Câu 6-3: Khai báo biến "unsigned int y" thì biến y là số nguyên dương:

(a) Có giá trị từ 0 đến 255 (b) Có giá trị từ 0 đến 256

(c) Có giá trị từ -128 đến 127 (d) Có giá trị từ -128 đến 255

Câu 6-4: Khai báo biến "signed int y" thì biến y là số nguyên dương:

(a) Có giá trị từ 0 đến 255 (b) Có giá trị từ 0 đến 256

(c) Có giá trị từ -128 đến 127 (d) Có giá trị từ -128 đến 255

Câu 6-5: Kí hiệu "x = y/z" là lệnh có chức năng:

(a) X bằng y chia z lấy số dư (b) X bằng y chia z lấy kết quả

(c) X bằng z chia y lấy số dư (d) X bằng z chia y lấy kết quả

Câu 6-6: Kí hiệu "x = y%z" là lệnh có chức năng:

(a) X bằng y chia z lấy số dư (b) X bằng y chia z lấy kết quả

(c) X bằng z chia y lấy số dư (d) X bằng z chia y lấy kết quả

Câu 6-7: Khai báo nào là khai báo kí tự:

(a) INT (b) Char

(b) FLoat (d) Long

Câu 6-8: Vi điều khiển AT89S52 thì các port:

(a) Chỉ truy xuất bit (b) Truy xuất bit và byte

(c) Truy xuất byte (d) Truy xuất 16 bit

Câu 6-9: Trong mạch quét 8 led thì transistor có chức năng:

(a) Khuếch đại dòng (b) Khuếch đại áp

(c) Khuếch đại dòng, áp (d) Điều khiển led

Câu 6-10: Trong mạch quét 8 led thì mỗi thời điểm có:

(a) 2 led sáng (b) 1 led sáng

(c) 3 led sáng (d) 8 led sáng

Câu 6-11: Trong mạch quét 8 led nếu chương trình không quét thì:

(a) 8 led vẫn sáng (b) 8 led sáng mờ

(c) 8 led tắt (d) 1 led sáng

Câu 6-12: Nếu mở rộng mạch quét 8 led thành 16 led thì dùng tối thiểu:

(a) 16 đường I/O (b) 32 đường I/O

(c) 8 đường I/O (d) 24 đường I/O

Câu 6-13: Trong mạch quét 8 led thì thời gian led sáng mỗi led là:

(a) 1/8 chu kỳ (b) 7/8 chu kỳ

(c) 1/4 chu kỳ (d) 8/7 chu kỳ

Câu 6-14: Trong mạch quét 8 led thì thời gian led tắt mỗi led là:

(a) 1/8 chu kỳ (b) 7/8 chu kỳ

(c) 1/4 chu kỳ (d) 8/7 chu kỳ

Câu 6-15: Khi AT89S52 giao tiếp với LCD dùng bao nhiêu IO:

(a) 8 đường I/O (b) 11 đường I/O

(c) 3 đường I/O (d) 16 đường I/O

Câu 6-16: Mã 7 đoạn số 9 của led anode chung là:

(a) 0xF9 (b) 0x90

(c) 0x80 (d) 0xF0

Câu 6-17: Mã 7 đoạn số hex F của led anode chung là:

(a) 0x8E (b) 0xE8

(c) 0xF8 (d) 0Xfe

Câu 6-18: Dữ liệu hiển thị trên LCD thuộc dạng:

(a) Mã BCD (b) Mã GRAY

(c) Mã ASCII (d) Mã quá 3

Câu 6-19: Địa chỉ bắt đầu hàng thứ nhất của LCD là:

(a) 80 (b) 0x8F

(c) 0x80 (d) 0xC0

Câu 6-20: Địa chỉ bắt đầu hàng thứ hai của LCD là:

(a) 80 (b) 0x8F

(c) 0x80 (d) 0xC0

4. Bài tập

Bài tập 6-1: Dùng vi điều khiển AT89S52 giao tiếp với 32 led đơn dùng 4 port, hãy viết lưu đồ và chương trình điều khiển với các yêu cầu như sau:

+ Chương trình 1: 32 led sáng dần tắt dần từ phải sang trái với thời gian delay là 1s.
+ Chương trình 2: 32 led sáng dần tắt dần từ trái sang phải với thời gian delay là 1s.
+ Chương trình 3: 32 led sáng tắt (chớp tắt) 5 lần với thời gian delay là 1s.

Bài tập 6-2: Dùng vi điều khiển AT89S52 giao tiếp với hai led 7 đoạn loại anode chung, hãy viết lưu đồ và chương trình đếm thời gian 1 giây với yêu cầu đếm từ 00 đến 59 rồi đếm xuống 00.

Bài tập 6-3: Dùng vi điều khiển AT89S52 giao tiếp với 8 led 7 đoạn loại anode chung dùng phương pháp quét, hãy viết lưu đồ và chương trình đếm giây.

Bài tập 6-4: Dùng vi điều khiển AT89S52 giao tiếp với 8 led 7 đoạn loại anode chung dùng phương pháp quét, hãy viết lưu đồ và chương trình đếm phút giây.

Bài tập 6-5: Dùng vi điều khiển AT89S52 giao tiếp với 8 led 7 đoạn loại anode chung dùng phương pháp quét, hãy viết lưu đồ và chương trình đếm giờ phút giây.

Bài tập 6-6: Dùng vi điều khiển AT89S52 giao tiếp với 8 led 7 đoạn loại anode chung dùng phương pháp quét, hãy viết lưu đồ và chương trình đếm giờ phút giây, có 3 nút nhấn mod, up, dn để chỉnh thời gian.

Bài tập 6-7: Dùng vi điều khiển AT89S52 giao tiếp với LCD, hãy viết lưu đồ và chương trình đếm giờ phút giây.

Bài tập 6-8: Dùng vi điều khiển AT89S52 giao tiếp với LCD, hãy viết lưu đồ và chương trình đếm giờ phút giây, có 3 nút nhấn mod, up, dn để chỉnh thời gian.

Chương 7
VI ĐIỀU KHIỂN AT89S52: TIMER - COUNTER

- **GIỚI THIỆU**
- **TIMER/COUNTER CỦA VI ĐIỀU KHIỂN ATMEL AT89S52**
  - KHẢO SÁT TIMER T0, T1 CỦA AT89S52
    - ✓ *Khảo sát thanh ghi TMOD của AT89S52*
    - ✓ *Khảo sát thanh ghi TCON của AT89S52*
    - ✓ *Timer của AT89S52 hoạt động ở chế độ 0 hay mode 0*
    - ✓ *Timer của AT89S52 hoạt động ở chế độ 1 hay mode 1*
    - ✓ *Timer của AT89S52 hoạt động ở chế độ 2 hay mode 2*
    - ✓ *Timer của AT89S52 hoạt động ở chế độ 2 hay mode 2*
  - CÁC THANH GHI, CÁC BIT CỦA TIMER TRONG NGÔN NGỮ KEIL-C
- **ỨNG DỤNG TIMER/COUNTER CỦA VI ĐIỀU KHIỂN ATMEL AT89S52**
  - ĐỊNH THỜI DÙNG TIMER CỦA AT89S52
  - ĐẾM XUNG NGOẠI DÙNG COUNTER CỦA AT89S52
- **CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP**
  - CÂU HỎI ÔN TẬP
  - CÂU HỎI MỞ RỘNG
  - CÂU HỎI TRẮC NGHIỆM
  - BÀI TẬP

I. GIỚI THIỆU

Ở chương này khảo sát các timer/counter của các vi điều khiển, các timer/counter có chức năng đếm xung nội có chu kỳ đã biết để định thời điều khiển thiết bị theo thời gian, có thể đếm xung ngoại để đếm sự kiện như đếm số vòng dây quấn, đếm sản phẩm, đếm tiền, ...

Timer có nhiều tiện ích trong điều khiển nên hầu hết các vi điều khiển đều tích hợp, ở chương này chúng ta sẽ khảo sát timer/counter và các ứng dụng.

Sau khi kết thúc chương này, bạn có thể sử dụng được timer/counter của các vi điều khiển.

II. TIMER/COUNTER CỦA VI ĐIỀU KHIỂN ATMEL AT89S52

Vi điều khiển AT89S52 có 3 timer/counter T0, T1 và T2. Timer/counter T0 và T1 xuất hiện ở các thế hệ họ MCS51 và T2 xuất hiện ở họ MCS52. Hai timer T0 và T1 sử dụng chung hai thanh ghi TMOD và TCON, còn các thanh ghi của timer T2 là độc lập.

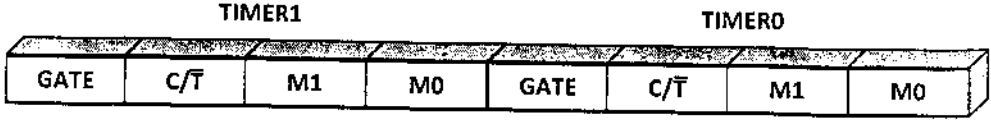
1. Khảo sát timer T0, T1 của AT89S52

Hai timer T0 và T1 có 6 thanh ghi như bảng 7-1:

Tên	Chức năng
TCON	Control: điều khiển
TMOD	Mode: lựa chọn kiểu hoạt động
TL0	Byte thấp của timer T0
TL1	Byte thấp của timer T1
TH0	Byte cao của timer T0
TH1	Byte cao của timer T1

➢ ***Khảo sát thanh ghi TMOD của AT89S52***

Thanh ghi TMOD gồm hai nhóm 4 bit: 4 bit thấp dùng để thiết lập các chế độ hoạt động cho T0 và 4 bit cao thiết lập các chế độ hoạt động cho T1.



Hình 7-1: *Thanh ghi TMOD.*

Các bit của thanh ghi TMOD được tóm tắt như bảng 7-2.

Bảng 7-2: Các bit trong thanh ghi TMOD:

Bit	Tên	Timer	Chức năng
7	GATE	1	Nếu GATE = 1 thì Timer 1 chỉ làm việc khi INT1= 1 GATE = 0 thì timer hoạt động bình thường.
6	C/$\overline{T}$	1	Bit lựa chọn counter hay timer:
			C/$\overline{T}$ = 1: đếm xung từ bên ngoài đưa đến ngõ vào T1.
			C/$\overline{T}$ = 0: định thời đếm xung nội bên trong.
5	M1	1	Bit chọn mode của Timer 1.
4	M0	1	Bit chọn mode của Timer 1.
3	GATE	0	Nếu GATE = 1 thì Timer 0 chỉ làm việc khi INT0= 1
2	C/$\overline{T}$	0	Bit lựa chọn counter hay timer: giống như trên.
1	M1	0	Bit chọn mode của Timer 0.
0	M0	0	Bit chọn mode của Timer 0.

Hai bit M0 và M1 tạo ra bốn trạng thái tương ứng với bốn kiểu làm việc khác nhau của T0 hoặc của T1 như bảng 7-3.

Bảng 7-3: Các bit chọn mode trong thanh ghi TMOD:

M1	M0	Kiểu	Chức năng
0	0	0	Kiểu đếm 13 bit tương thích với vi điều khiển 8048.
0	1	1	Kiểu đếm 16 bit.
1	0	2	Kiểu đếm 8 bit tự động nạp lại.
1	1	3	Kiểu chia tách Timer: ***Timer0***: được tách ra làm hai timer 8 bit gồm có: • Timer 8 bit TL0 được điều khiển bởi các bit của T0. • Timer 8 bit TH0 được điều khiển bởi các bit của T1. ***Timer1***: không được hoạt động ở mode 3.

➢ ***Khảo sát thanh ghi TCON của AT89S52***

Thanh ghi điều khiển TCON chứa các bit trạng thái và các bit điều khiển cho T0 và T1.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Hình 7-2: Thanh ghi TCON.

Hoạt động của từng bit của thanh ghi tcon được tóm tắt như bảng 7-4:

Bảng 7-4: Các bit trong thanh ghi TCON:

Bit	Kí hiệu	Chức năng
7	TF1	Cờ tràn Timer 1.
6	TR1	Bit điều khiển Timer 1 đếm / ngừng đếm: TR1 = 1 thì timer 1 được phép đếm khi có xung. TR1 = 0 thì timer 1 không được phép đếm (ngừng).
5	TF0	Cờ tràn Timer 0 (hoạt động tương tự TF1)
4	TR0	Bit điều khiển Timer 0 (giống TR1)
3	IE1	Cờ báo ngắt INT1.
2	IT1	Bit lựa chọn ngắt INT1 tác động bằng mức hay bằng cạnh. IT1 = 0 thì ngắt INT1 tác động bằng mức. IT1 = 1 thì ngắt INT1 tác động bằng cạnh xuống.
1	IE0	Giống như IE1 nhưng phục vụ cho ngắt INT0
0	IT0	Giống như IT1 nhưng phục vụ cho ngắt INT0

Khi timer bị tràn, cờ tràn TFx = 1, ta có thể xóa cờ bằng phần mềm hoặc tự động xóa khi vi điều khiển thực hiện chương trình con phục vụ ngắt timer.

Khi có ngắt xảy ra ở ngõ vào INTx sẽ làm cờ IEx lên mức 1.

Khi vi điều khiển thực hiện chương trình con phục vụ ngắt INTx, tự động xóa luôn cờ báo ngắt IEx.

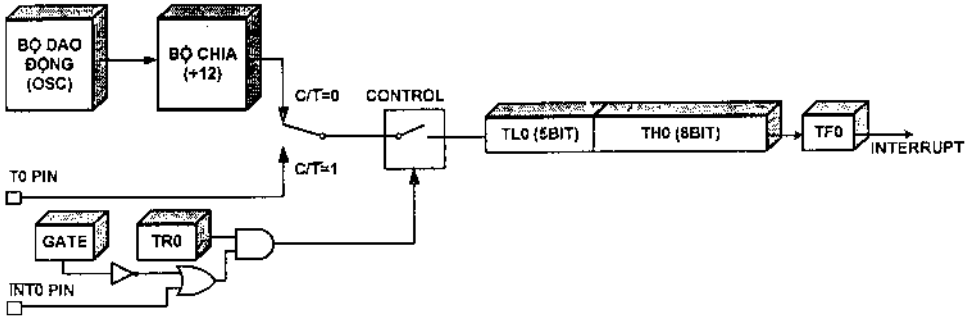
Chú ý: "x" là 0 hoặc 1 nói chung cho cả T0 và T1 hoặc INT0 và INT1.

Như đã trình bày ở trên các timer có bốn kiểu hoạt động, phần này sẽ khảo sát chi tiết các kiểu hoạt động của timer.

➢ ***Timer của AT89S52 hoạt động ở chế độ 0 hay mode 0***

Mode 0 là mode đếm 13 bit: trong đó 8 bit cao sử dụng hết 8 bit của thanh ghi THx, 5 bit còn lại chỉ sử dụng 5 bit trọng số thấp của thanh ghi TLx, còn 3 bit cao của TLx không dùng như hình 7-3.

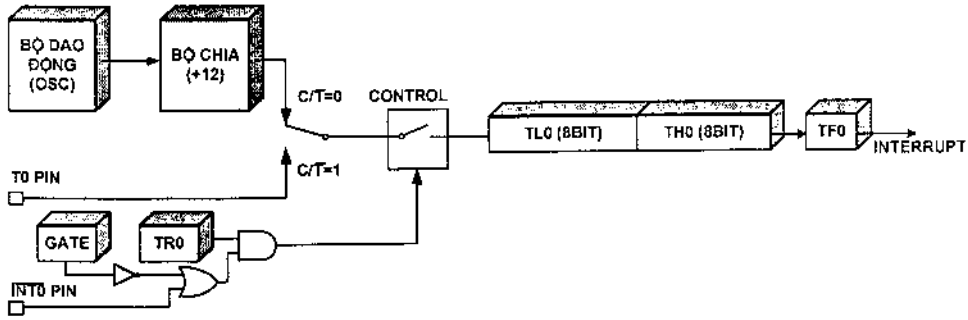
Ở mode 0 thì giá trị đếm lớn nhất là 2^{13} = 8192 tức đếm từ "0 0000 0000 0000B" đến "1 1111 1111 1111B" và nếu có thêm một xung nữa thì bộ đếm sẽ tràn và cờ tràn lên 1. Nếu cho phép timer ngắt thì ngắt sẽ tác động.



Hình 7-3: *Timer 0 hoạt động ở mode 0.*

➢ ***Timer của AT89S52 hoạt động ở chế độ 1 hay mode 1***

Mode 1 là mode đếm 16 bit, xem hình 7-4. Ở mode này giá trị đếm là lớn nhất là 2^{16}. Timer hoạt động ở mode 1 giống như mode 0 chỉ khác là đếm 16 bit.

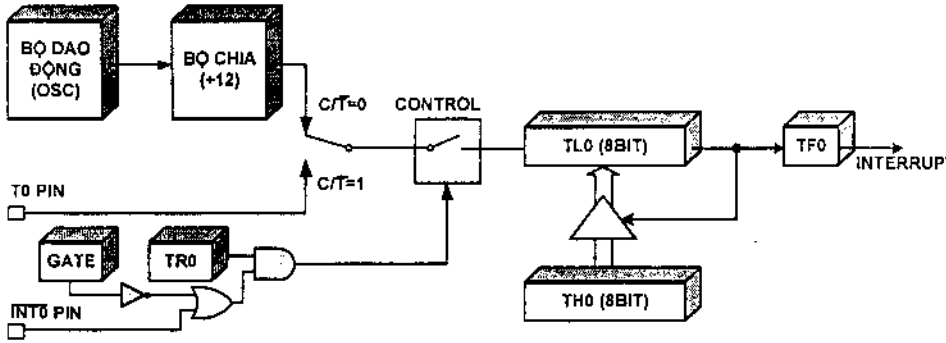


Hình 7-4: *Timer 0 hoạt động ở mode 1.*

➢ ***Timer của AT89S52 hoạt động ở chế độ 2 hay mode 2***

Mode 2 là mode đếm 8 bit tự động nạp lại, byte thấp TLx của Timer hoạt động như một Timer 8 bit trong khi byte cao THx của Timer dùng để lưu trữ giá trị để nạp lại cho thanh ghi TLx.

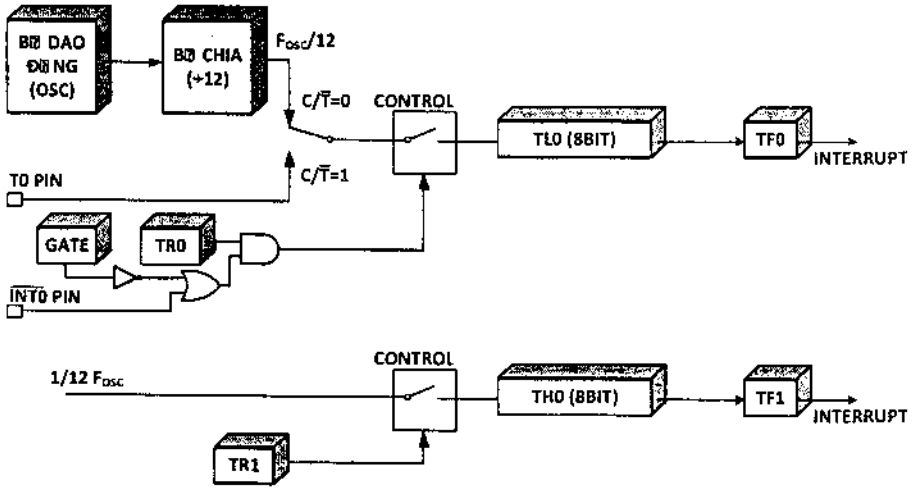
Khi bộ đếm TLx đếm và chuyển trạng thái từ FFH sang 00H: thì cờ tràn lên 1 và giá trị lưu trong THx được nạp vào TLx và bộ đếm tiếp tục đếm từ giá trị vừa nạp vào TLx từ THx lên và cho đến khi có chuyển trạng thái từ FFH sang 00H kế tiếp và cứ thế tiếp tục. Sơ đồ minh họa cho timer hoạt động ở mode 2 như hình 7-5.



Hình 7-5: Timer 1 hoạt động ở mode 2.

➢ ***Timer của AT89S52 hoạt động ở chế độ 3 hay mode 3***

Mode 3 là mode Timer0 tách ra làm 2 timer cùng với timer 1 tạo thành 3 timer.



Hình 7-6:Timer 0 hoạt động ở mode 3.

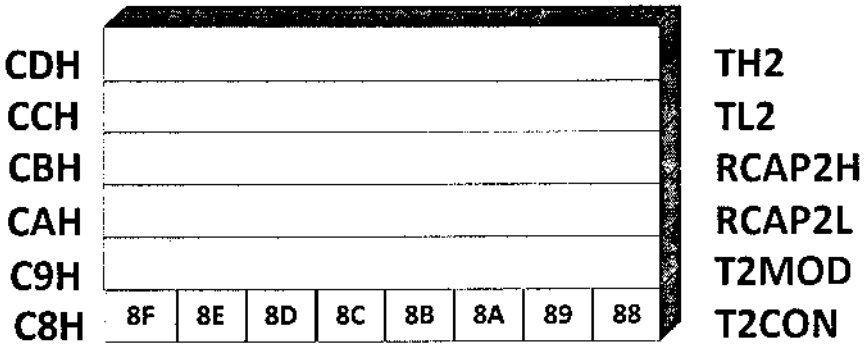
Khi định ở cấu hình mode 3 thì timer0 được chia là 2 timer 8 bit TL0 và TH0 hoạt động như những timer độc lập và sử dụng các bit TF0 và TF1 làm các bit cờ tràn tương ứng như hình 7-6.

Timer 1 không sử dụng ở mode 3 nhưng có thể hoạt động ở các mode khác và không có báo tràn vì cờ tràn TF1 đã dùng để báo tràn cho timer TH0.

Khi timer 0 hoạt động ở Mode 3 sẽ cung cấp thêm 1 timer 8 bit thứ ba. Khi Timer 0 ở mode 3 thì Timer 1 có thể hoạt động như là một bộ dao động thiết lập tốc độ Baud phục vụ cho Port nối tiếp để truyền và nhận dữ liệu, hoặc có thể dùng trong các ứng dụng mà không cần báo tràn và báo ngắt. Sơ đồ minh họa cho timer hoạt động ở mode 3 như hình 7-6.

2. Khảo sát timer T2 của AT89S52

Các thanh ghi của timer/counter T2 bao gồm: thanh ghi TL2, TH2, thanh ghi điều khiển T2CON, thanh ghi RCAP2L và RCAP2H.



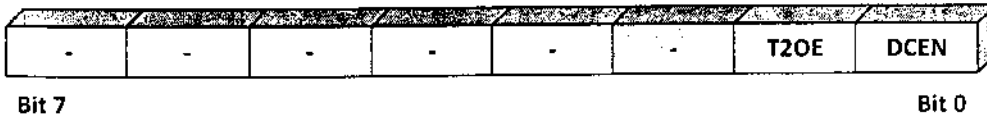
Hình 7-7: Các thanh ghi của timer T2.

Timer/counter T2 có thể dùng để định thời timer hoặc dùng như bộ đếm counter để đếm xung ngoài đưa đến ngõ vào P1.0.

Timer/counter T2 có ba kiểu hoạt động: đếm tự động nạp lại, chế độ Capture và thiết lập tốc độ baud để phục vụ cho truyền dữ liệu.

➢ ***Khảo sát thanh ghi T2MOD***

Cấu trúc của thanh ghi T2MOD như hình sau:

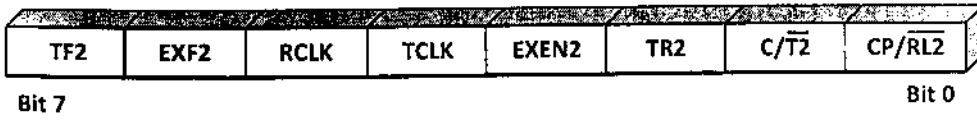


Hình 7-8: Thanh ghi T2MOD.

- Bit DCEN (Down Counter Enable): khi bằng 0 thì chỉ đếm lên, khi bằgng 1 thì T2 được định cấu hình như bộ đếm lên/xuống.
- Bit T2OE: là bit cho phép timer T2 xuất.

➢ ***Khảo sát thanh ghi T2CON***

Cấu trúc của thanh ghi T2CON như hình sau:



Hình 7-9: Thanh ghi T2CON.

Chức năng của thanh ghi điều khiển T2CON như bảng 7-5:

Bit	Kí hiệu	Chức năng
7	TF2	Cờ tràn Timer 2, (TF2 sẽ không được thiết lập lên mức 1 nếu bit TCLK hoặc RCLK ở mức 1).
6	EXF2	Cờ báo có xung ngoài đưa đến T2XE của timer T2: lên 1 khi xảy ra chế độ capture hoặc nạp lại dữ liệu bởi chuyển trạng thái từ 1 sang 0 ở ngõ vào T2EX và EXEN2 = 1. Khi cho phép T2 ngắt, EXF2=1 thì CPU sẽ thực hiện chương trình con phục vụ ngắt T2, bit EXF2 phải xóa bằng phần mềm. Bit EXF2 không tạo ra ngắt ở chế độ đếm lên/xuống.
5	RCLK	Bit cho phép nhận lựa chọn xung clock để nhận dữ liệu nối tiếp: • Khi bit RCLK=1 thì T2 tạo tốc độ baud cho port nối tiếp để nhận dữ liệu còn timer T1 sẽ cung cấp tốc độ baud cho port nối tiếp để phát dữ liệu đi. • Khi bit RCLK=0 thì T1 sẽ tạo tốc độ baud để nhận dữ liệu.
4	TCLK	Bit cho phép nhận lựa chọn xung clock để phát dữ liệu nối tiếp: • Khi TCLK=1 thì T2 tạo tốc độ baud cho port nối tiếp để nhận dữ liệu về. • Khi bit TCLK=0 thì T1 sẽ tạo tốc độ baud để nhận dữ liệu.
3	EXEN2	Bit cho phép tác động ra bên ngoài. • Khi EXEN2 = 1 thì cho phép xuất xung báo hiệu ở ngõ ra T2EX khi hoạt động ở chế độ capture hoặc chế độ nạp lại với điều kiện là T2 không được dùng để tạo tốc độ baud. • Khi EXEN2 = 0 thì không cho phép xuất tín hiệu ra ngoài.
2	TR2	Bit điều khiển Timer 1 đếm / ngừng đếm: • TR2 = 1 thì timer 1 được phép đếm xung. • TR2 = 0 thì timer 1 không được phép đếm xung (ngừng).

1	$C/\overline{T2}$	Bit lựa chọn counter hay timer: (tích cực cạnh xuống) $C/\overline{T2}$ = 1: đếm xung từ bên ngoài đưa đến ngõ vào T2. $C/\overline{T2}$ = 0: định thời đếm xung nội bên trong.
0	$CP/\overline{RL2}$	Bit lựa chọn chế độ Capture hay tự động nạp lại: • $CP/\overline{RL2}$ = 1: thì hoạt động Capture và khi kết thúc thì chuyển trạng thái từ 1 sang 0 ở ngõ ra T2EX khi bit EXEN2=1. • $CP/\overline{RL2}$ = 0: thì hoạt động đếm và tự động nạp lại và khi tràn hoặc khi ngõ vào T2EX chuyển trạng thái từ 1 sang 0 nếu bit EXEN2=1. Khi bit RCLK hoặc TCLK = 1 thì bit này xem như bỏ.

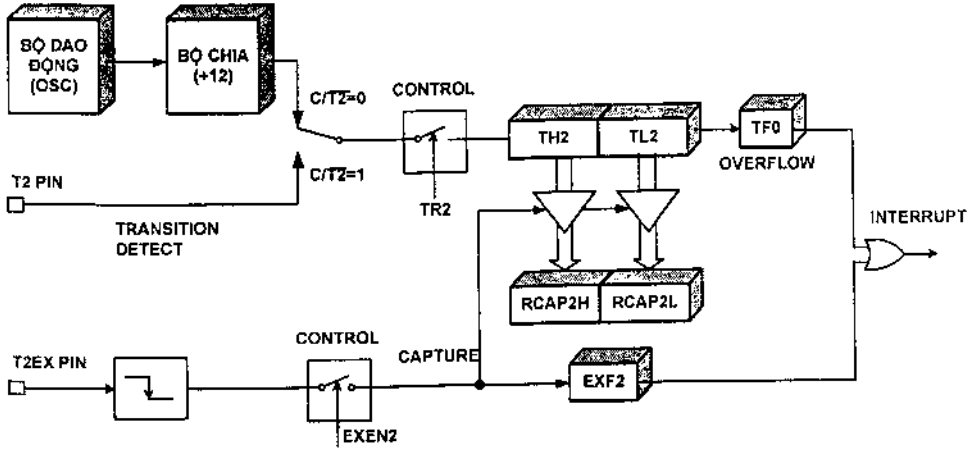
➢ ***Chế độ Capture:***

Khi $CP/\overline{RL2}=1$ thì timer hoạt động ở chế độ capture.

Có hai lựa chọn tùy thuộc vào bit EXEN2 trong thanh ghi T2CON.

Nếu bit EXEN2 = 0 thì T2 là timer hay counter 16 bit và khi đếm tràn thì làm cờ tràn lên 1 và phát sinh ngắt nếu cho phép.

Nếu bit EXEN2 = 1 thì T2 hoạt động đếm bình thường nhưng nếu có chuyển trạng thái của xung bên ngoài đưa đến ngõ vào T2EX (cạnh xuống) thì giá trị đếm của cặp thanh ghi TH2 và TL2 sẽ nạp sang cặp thanh ghi RCAP2H và RCAP2L tương ứng, đồng thời làm bit EXF2 trong T2CON lên 1, cờ EXF2 có thể tạo yêu cầu ngắt giống như TF2.

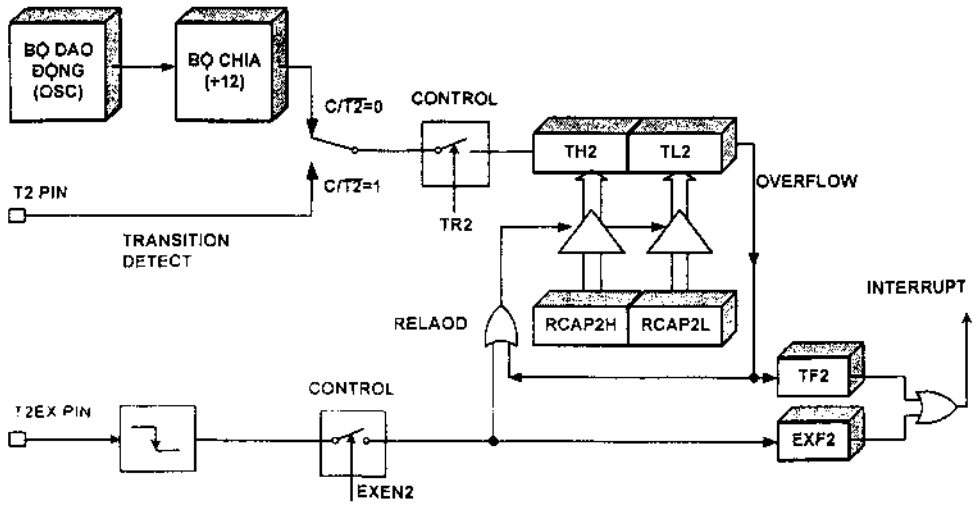


Hình 7-10: *Hoạt động của timer T2 ở chế độ Thu nhận dữ liệu.*

➢ ***Khảo sát chế độ đếm tự động nạp lại:***

Khi $CP/\overline{RL2} = 0$, timer hoạt động ở chế độ đếm tự động nạp lại. Ở chế độ này thì T2 có thể lập trình để đếm lên hoặc xuống phụ thuộc vào bit DCEN.

Khi reset, bit DCEN bằng 0 nên chế độ đếm mặc nhiên là đếm lên. Khi lập trình làm bit DCEN bằng 1 thì hoạt động chế độ ***đếm lên hoặc đếm xuống phụ thuộc vào giá trị của bit ở ngõ vào T2EX.***



Hình 7-11: *Hoạt động của T2 ở chế độ tự động nạp lại đếm lên với DCEN = 0.*

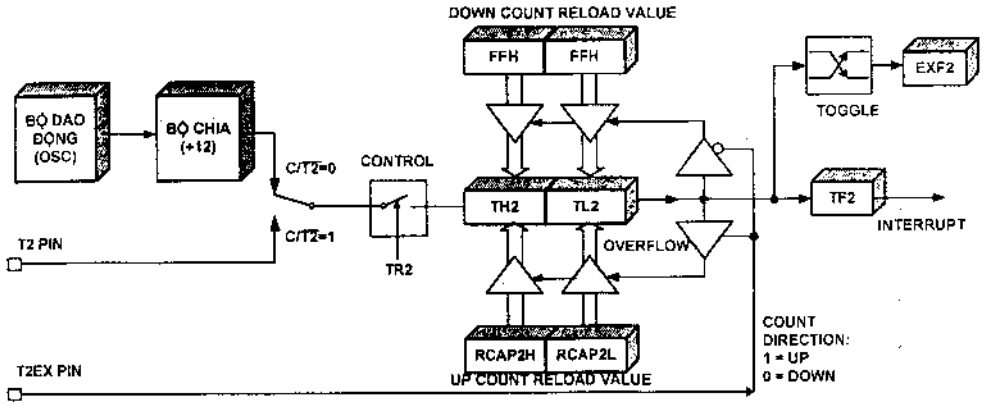
Hoạt động đếm lên khi bit DCEN bằng 0 như hình sau:

Ở chế độ này, có hai tùy chọn phụ thuộc vào bit EXEN2:

- Nếu EXEN2 = 0 thì T2 đếm lên vượt quá FFFFH thì tràn làm TF2 lên 1 đồng thời nạp lại giá trị lưu trữ trong hai thanh ghi RCAP2L và RCAP2H cho TL2, TH2.
- Nếu EXEN2 = 1 thì T2 đếm lên và nạp lại giá trị khi bị tràn hoặc khi có chuyển trạng thái cạnh xuống của xung đưa đến ngõ vào T2EX.

Cả hai đều phát sinh yêu cầu ngắt.

Khi cho bit DCEN bằng 1 thì T2 hoạt động đếm lên hoặc xuống như hình sau:



Hình 7-12: Hoạt động của T2 ở chế độ tự động nạp lại đếm lên/xuống với DCEN = 1.

Ở chế độ này nếu ngõ vào T2EX bằng 1 thì T2 sẽ đếm lên, T2 sẽ tràn khi giá trị đếm lớn hơn FFFFH và cờ tràn bằng 1, phát sinh ngắt, đồng thời nạp lại giá trị đặt trước trong cặp thanh ghi RCAP2L và RCAP2H cho TL2, TH2.

Nếu ngõ vào T2EX bằng 0 thì T2 sẽ đếm xuống, nếu giá trị đếm trong cặp thanh ghi TL2, TH2 nhỏ hơn RCAP2L, RCAP2H thì cờ tràn bằng 1, phát sinh ngắt, đồng thời nạp lại giá trị FFFFH cho TL2, TH2.

Bit EXF2 đảo trạng thái khi T2 tràn trên (đếm lên) hoặc tràn dưới (đếm xuống) thì có thể dùng như là bit thứ 17 vì trong chế độ hoạt động này thì bit EXF2 không phải là cờ báo ngắt.

3. Các thanh ghi, các bit của Timer trong ngôn ngữ Keil-C

Trong ngôn ngữ lập trình Keil-C đã định nghĩa sẵn các tên thanh ghi và các bit của các thanh ghi, bạn cần phải biết các tên thanh ghi và các bit để lập trình.

Bảng 7-6: Thư viện đã định nghĩa các thanh ghi và các bit timer/counter gồm:

TCON Bit Registers	*TMOD Bit Values*
sbit IT0 = 0x88;	*#define T0_M0_ 0x01*
sbit IE0 = 0x89;	*#define T0_M1_ 0x02*
sbit IT1 = 0x8A;	*#define T0_CT_ 0x04*
sbit IE1 = 0x8B;	*#define T0_GATE_ 0x08*
sbit TR0 = 0x8C;	*#define T1_M0_ 0x10*
sbit TF0 = 0x8D;	*#define T1_M1_ 0x20*

sbit TR1 = 0x8E; *sbit TF1 = 0x8F;*	*#define T1_CT_ 0x40* *#define T1_GATE_ 0x80*
	#define T1_MASK_ 0xF0 *#define T0_MASK_ 0x0F*

Thanh ghi TCON và các bit có địa chỉ xác định ví dụ "sbit TF0 = 0x8F" là bit tràn của timer T1 có địa chỉ là 0x8F, thanh ghi TMOD không cho phép truy xuất bit, mỗi bit có 1 giá trị tương ứng, ví dụ bit "T0_CT_" được gán có giá trị là 0x04 tương ứng với chế độ được chọn là counter, mặc nhiên không chọn thì hoạt động ở chế độ timer.

Ví dụ 7-1: Thiết lập từ chế độ hoạt động cho timer0 là counter, chế độ 1.

Giải: TMOD = T0_CT_ + T0_M0_;

Với lệnh này thì giá trị gán cho thanh ghi TMOD bằng 0x04 + 0x01 = 0x05.

Ví dụ 7-2: Thiết lập từ chế độ hoạt động cho timer0 là timer, chế độ 2.

Giải: TMOD = T0_M1_;

Với lệnh này thì giá trị gán cho thanh ghi TMOD bằng 0x02.

Ví dụ 7-3: Thiết lập từ chế độ hoạt động cho timer1: là timer, chế độ 1, timer0: là counter, chế độ 1

Giải: TMOD = T1_M0_ + T0_CT_ + T0_M0_;

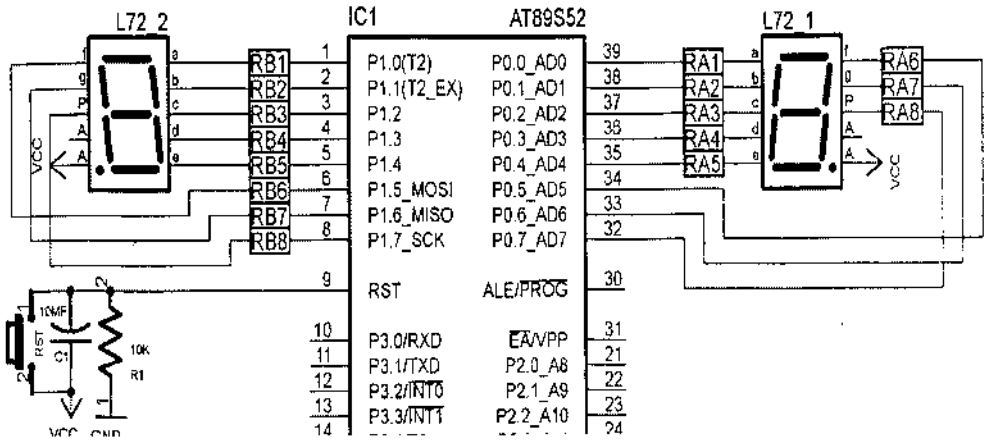
Với lệnh này thì giá trị gán cho thanh ghi TMOD bằng 0x10 + 0x04 + 0x01 = 0x15.

III. ỨNG DỤNG TIMER/COUNTER CỦA VI ĐIỀU KHIỂN ATMEL AT89S52

1. Định thời dùng Timer của AT89S52

Bài 7-1: Dùng vi điều khiển AT89S52 đếm giây sử dụng timer T0 để định thời chính xác.

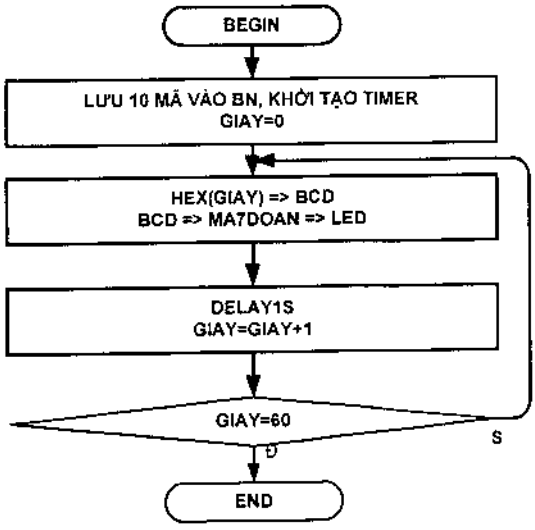
➢ ***Sơ đồ mạch:***



Hình 7-13: *Đếm giây dùng timer.*

Kết quả đếm giây hiển thị trên 2 led nối với port0 và port1.

➢ ***Lưu đồ:***



Hình 7-14: *Lưu đồ đếm giây dùng timer.*

➢ ***Chương* trình Keil-C:**

```
#include <AT89X52.h>
unsigned char MA7D[10] =
{0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90};
int GIAY,CHUC,DONVI;
void DELAY_TIMER0()
```

```
{     int BDN;
      TR0 = 1;
      for (BDN = 0; BDN<20;BDN++)
      {     do    {}
            while(TF0==0);
            TF0 = 0;TH0 = 0X3C;TL0 = 0XB0;
      }
      TR0 = 0;
}
void MAIN ()
{
      TMOD = T0_M0_ ;    TH0 = 0x3C;    TL0 = 0xB0;
 while(1)
            {
            for (GIAY = 0; GIAY <60; GIAY ++)
                  {     CHUC = GIAY /10;          DONVI = GIAY %10;
                        P0 = MA7D[DONVI];         P1 = MA7D[CHUC];
                        DELAY_TIMER0();
                  }
            }
}
```

❖ **Giải thích chương trình:**

Chương trình chính có chức năng khởi tạo timer T0 hoạt động định thời chế độ 1, giá trị bắt đầu đếm cho timer là 15536 = 3CB0H gán cho cặp thanh ghi TH0 và TL0 để đếm 50000 xung có chu kỳ 1μs, thạch anh sử dụng là 12MHz.

Cho biến "GIAY" chạy từ 0 đến 59, tiến hành tách hàng chục và hàng đơn vị để giải mã và gởi ra hai port để hiển thị trên hai led 7 đoạn.

Sau đó gọi chương trình con delay 1 giây dùng timer.

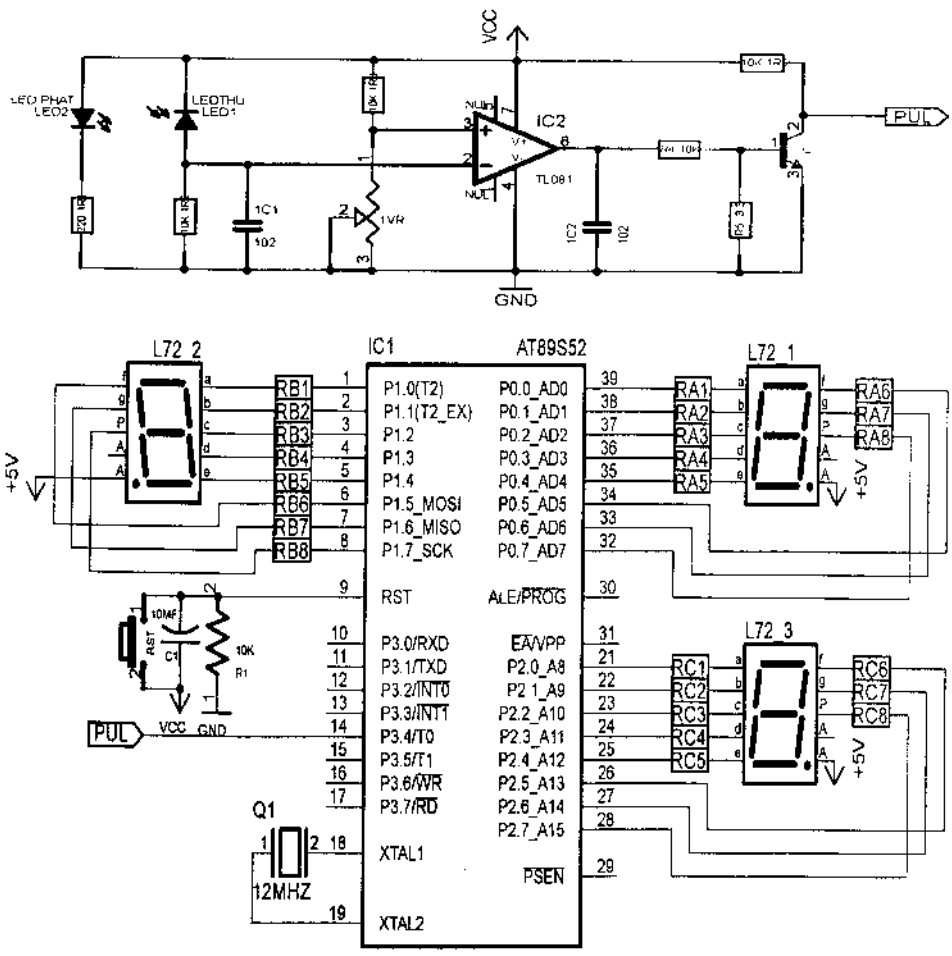
Chương trình con delay dùng timer có tên là "DELAY_TIMER0()" thực hiện các lệnh:

Gán TR0 bằng 1 để cho timer bắt đầu đếm, cho biến đếm ngắt chạy từ 0 đến 19 là 20 lần, mỗi lần làm lệnh "do while (TF0==0)" – lệnh này kiểm tra cờ tràn TF0 của timer T0, nếu chưa tràn thì tiếp tục chờ, khi tràn thì tiến hành xóa cờ tràn bằng lệnh "TF0 = 0" và khởi gán lại giá trị xuất phát để đếm 50000 xung tiếp theo, cho đến khi BDN bằng 20 thì kết thúc, ngừng timer và thời gian delay được 1 giây, trở lại chương trình chính để thực hiện tiếp.

2. Đếm xung ngoại dùng COUNTER của AT89S52

Bài 7-2: Dùng vi điều khiển AT89S52 đếm xung ngoại dùng timer T0, kết quả đếm hiển thị trên ba led kết nối trực tiếp với ba port, giới hạn đếm từ 0 đến 120, khi bằng 120 thì quay về 0.

➢ ***Sơ đồ mạch:***



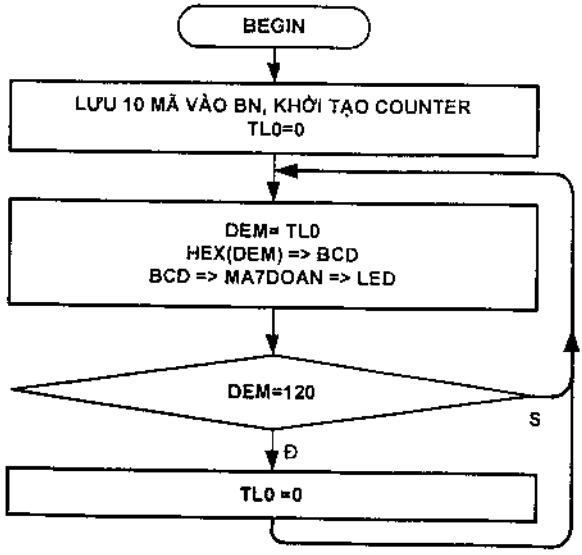
Hình 7-15: *Đếm sản phẩm dùng counter.*

➢ ***Giải thích nguyên lý hoạt động của mạch:***

Mạch tạo xung để phát hiện sản phẩm gồm led phát hồng ngoại và led thu hồng ngoài, khi không có sản phẩm thì ánh sáng hồng ngoại chiếu thẳng vào led thu hồng ngoại làm điện trở của led giảm, điện áp ngõ vào trừ của opamp gần bằng điện áp nguồn Vcc, lớn hơn điện áp ngõ vào cộng đã được cân chỉnh bằng biến trở khoảng 4V, qua mạch so sánh dùng opamp sẽ làm ngõ ra bằng 0V, làm transistor tắt, điện áp ngõ ra "pul" bằng Vcc tương đương mức logic '1'.

Khi có sản phẩm đi ngang qua giữa hai led phát và thu làm led thu không nhận được ánh sáng hồng ngoại, điện trở của led thu tăng lên rất lớn, điện áp ngõ vào trừ xuống gần bằng 0V nhỏ hơn điện áp của ngõ vào cộng, qua mạch so sánh làm ngõ ra lên mức '1', transistor dẫn, ngõ ra "pul" xuống mức '0', khi sản phẩm đi qua thì lên mức '1' trở lại, vậy cứ mỗi sản phẩm đi qua sẽ tạo ra 1 xung, xung được đưa đến mạch đếm counter T0 để đếm.

➢ ***Lưu đồ:***



Hình 7-16: *Lưu đồ đếm sản phẩm từ 0 đến 120.*

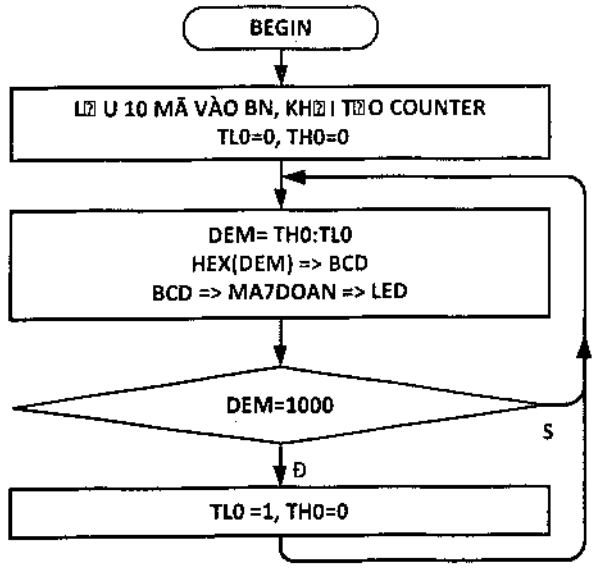
➢ ***Chương trình Keil-C:***

```
#include<AT89X52.H>
const unsigned char
MA7D[10]={0xc0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
int KQD;
```

```
unsigned char DVI,CHUC,TRAM;

void giaima_hienthi( )
{
 DVI=KQD%10; KQD=KQD/10; CHUC= KQD%10; TRAM=KQD/10;
 P0= MA7D[DVI]; P1= MA7D[CHUC]; P2=MA7D[TRAM];
}
void main ( )
{       TMOD=0x05;      TR0=1;
        while(1)
        {       KQD = TL0;          giaima_hienthi( );
                if(TL0==120)        {TL0=0;}
 }
 }
```

Bài 7-3: Dùng vi điều khiển AT89S52 đếm xung ngoại dùng timer T0, kết quả đếm hiển thị trên ba led kết nối trực tiếp với ba port giống sơ đồ hình 7-15, giới hạn đếm từ 000 đến 999, khi bằng 1000 thì quay về 1.

➢ ***Lưu đồ:***



Hình 7-17: *Lưu đồ đếm sản phẩm từ 0 đến 999.*

Lưu đồ khởi tạo 10 mã 7 đoạn, timer hoạt động đếm xung ngoại, cho giá trị đếm bắt đầu từ 0, xử lý giá trị 2 byte TH0 và TL0 thành 1 giá trị 16 bit gán cho biến "DEM", chuyển sang số BCD, giải mã và hiển thị.

➢ ***Chương trình Keil-C:***

```
#include <REGX52.h>
unsigned char MA7D[10] =
{0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90};
void MAIN ()
{     int DEM,TRAM,CHUC,DONVI;
      TMOD = T0_CT_+T0_M0_;
      TR0 = 0;          TL0 = 0;
while(1)
          {     DEM = (TH0<<8) + TL0;
                {
                      DONVI = DEM %10;
                      DEM = DEM/10;       CHUC = DEM %10;
                      DEM = DEM/10;       TRAM = DEM %10;

                      P0 = MA7D[DONVI]; P1 = MA7D[CHUC];
                      P2 = MA7D[TRAM];
                      if (DEM ==1000)              {TL0 = 1; TH0 = 0;}
                }
          }
}
```

❖ **Giải thích chương trình:**

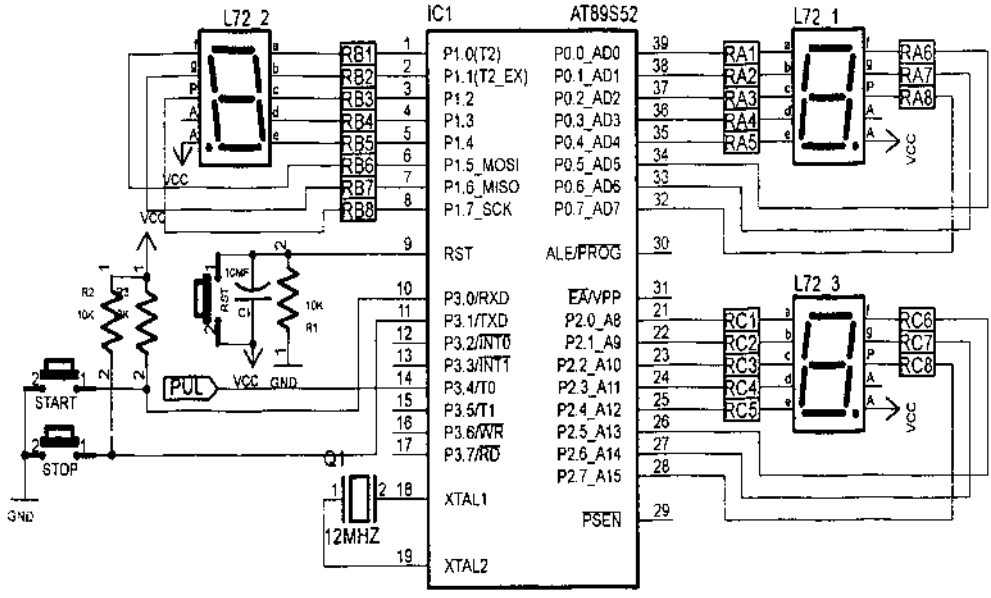
Chương trình khởi tạo timer hoạt động chế độ 1, đếm xung ngoại, cho giá trị đếm ban đầu bằng 0.

Vòng lặp while tiến hành xoay 8 bit của thanh ghi TH0 sang trái 8 bit để đưa lên 8 bit cao từ bit thứ 8 đến bit thứ 15, cộng với giá trị của thanh ghi TL0, kết quả gán cho biến "DEM" là 16 bit. Tiếp theo là tách từng số đơn vị, chục, trăm để giải mã và gởi ra port để hiển thị

Bài 7-4: Dùng vi điều khiển AT89S52 đếm xung ngoại dùng timer T0, kết quả đếm hiển thị trên ba led kết nối trực tiếp với ba port, giới hạn

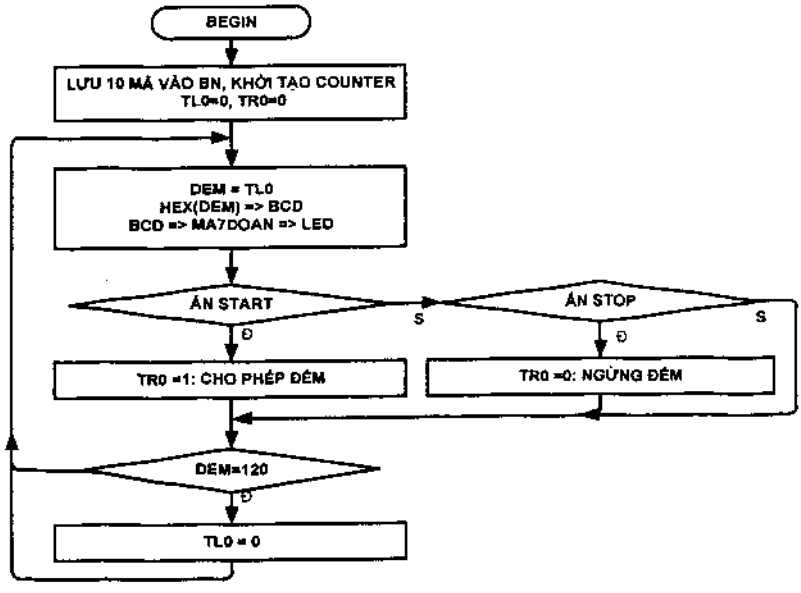
đếm từ 0 đến 120, khi bằng 120 thì quay về 0, có hai nút nhấn START và STOP. Khi nhấn START thì cho phép đếm, nhấn STOP thì ngừng.

Sơ đồ mạch: mạch tạo xung giống như hình 7-15.



Hình 7-18: *Đếm sản phẩm dùng counter có 2 nút điều khiển.*

➢ ***Lưu đồ:***



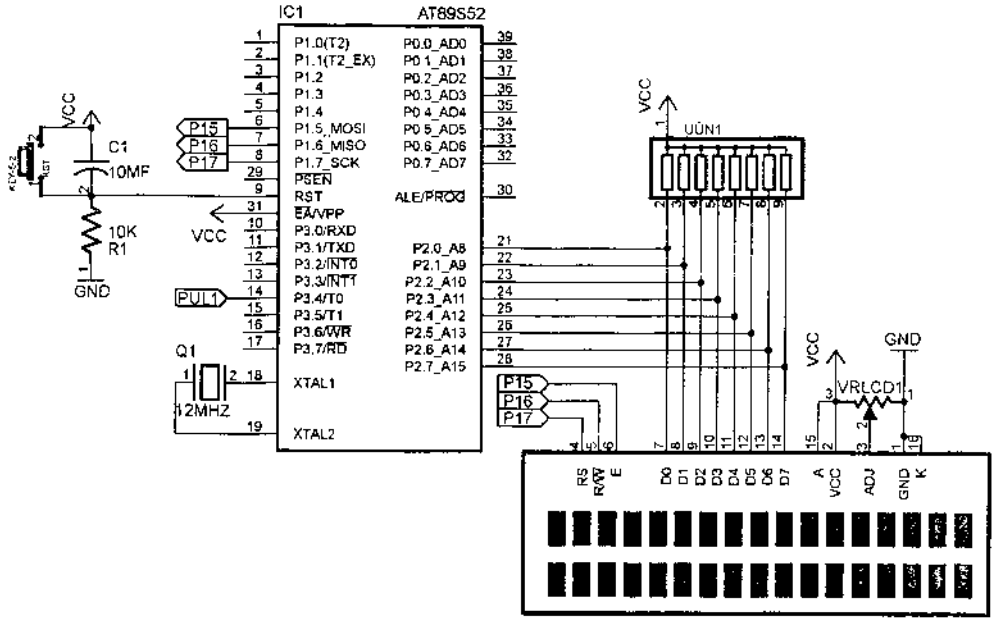
Hình 7-19: *Lưu đồ đếm sản phẩm từ 0 đến 120 có 2 phím start, stop.*

➢ ***Chương trình Keil-C:***

```
#include<AT89X52.H>
#define START        P3_0
#define STOP         P3_1
unsigned char MA7D[10] =
{0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90};
signed int KQD;
unsigned char DVI,CHUC,TRAM;
void giaima_hienthi()
{
 DVI=KQD%10; KQD=KQD/10; CHUC= KQD%10; TRAM=KQD/10;
 P0= MA7D[DVI]; P1= MA7D[CHUC]; P2=MA7D[TRAM];
}
void MAIN ()
{     TMOD = 0X50;  TR1 = 0;
 while(1)
            {
                KQD = (TH1<<8) + TL1;
                if (KQD ==301) {TL1 = 1; TH1 = 0;}
                giaima_hienthi();
                if (START==0)        {TR1=1;}  //CHO PHEP DEM
                if (STOP ==0)        {TR1=0;}  //NGUNG DEM
                }
}
```

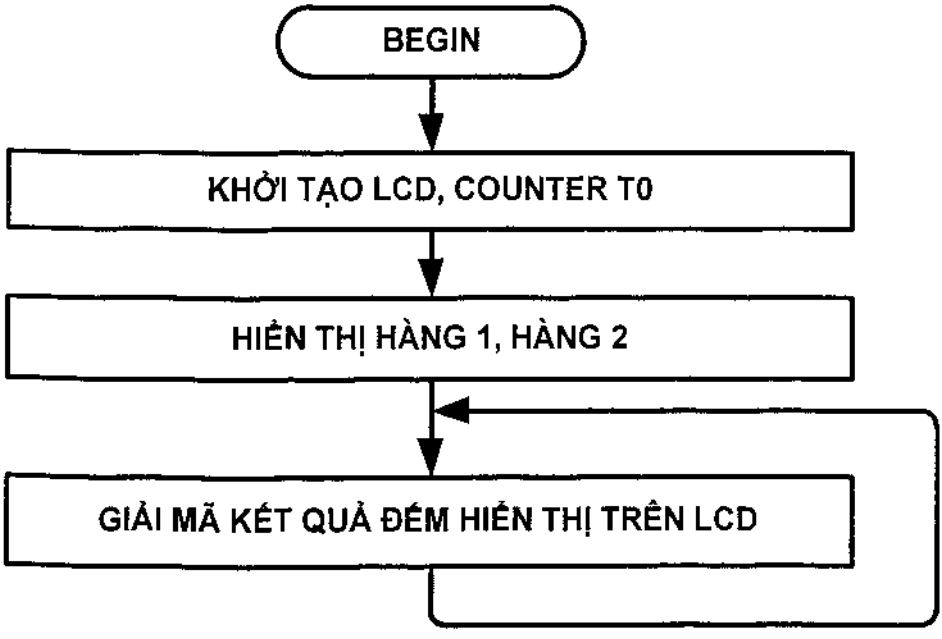
Bài 7-5: Dùng vi điều khiển AT89S52 đếm xung ngoại 1 kênh dùng T0 kết quả đếm hiển thị trên LCD 16×2, giới hạn đếm từ 0 đến 200, khi bằng 200 thì quay về 0. Giá trị đếm T0 hiển thị hàng 2.

➢ ***Sơ đồ mạch:*** Trong sơ đồ mạch này, vi điều khiển kết nối với LCD giống như đã trình bày và mạch cảm biến tạo xung cũng giống như đã trình bày.



Hình 7-20: *Đếm sản phẩm 1 kênh T0 hiển thị trên LCD.*

Lưu đồ: Lưu đồ giống như hiển thị trên led đơn, ở bài này chỉ khác là hiển thị trên LCD.



Hình 7-21: *Lưu đồ đếm sản phẩm từ 0 đến 120, hiển thị LCD.*

➢ **Chương trình Keil-C:**

```
#include <AT89X52.h>
#include <THUVIEN_LCD16X2.C>
unsigned char j;
unsigned char KQD,KQDT,TRAM,CHUC,DONVI;
const unsigned char HANG1[16]={" DH SP KY THUAT "};
const unsigned char HANG2[16]={"SAN PHAM:      "};

void GIAIMA_ASCII ()
{     DONVI=KQD%10;                KQD=KQD/10;
      CHUC=KQD%10;                 TRAM=KQD/10;

      DONVI=DONVI+0x30;  CHUC=CHUC+0x30;
      TRAM=TRAM+0x30;
      if (TRAM==0X30)
            { TRAM=' ';
                  if (CHUC==0X30)     { CHUC=' ';          }
}           }
void HIENTHI_LCD()
{           COMMAND_WRITE(0xCD);
      DATA_WRITE(TRAM);
            DATA_WRITE(CHUC);
      DATA_WRITE(DONVI);
}
void main ( )
{     SETUP_LCD( );
      COMMAND_WRITE(addr_line1);          delay(10);
      for( j=0;j<8;j++) {DATA_WRITE(HANG1[j]);}
```

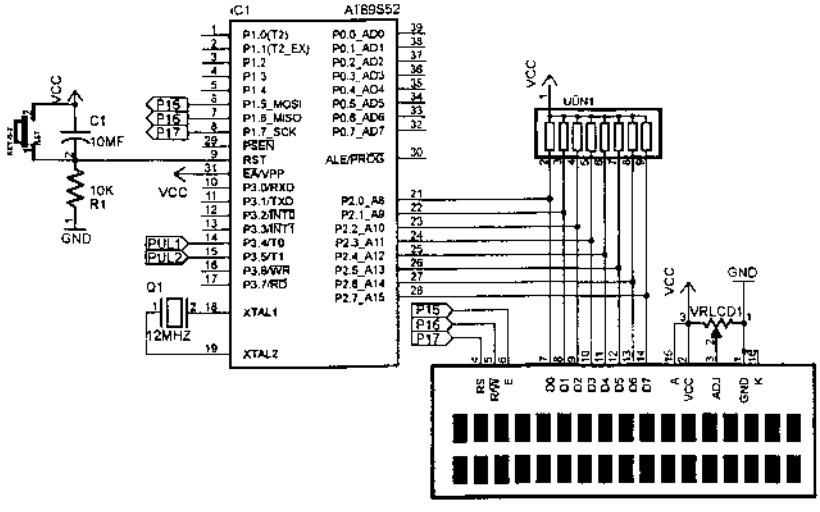
```
    COMMAND_WRITE(addr_line2);          delay(10);
    for( j=0;j<16;j++)          {DATA_WRITE(HANG2[j]);}
    TMOD=T0_CT_+T0_M0_;          TR0=1;     KQD=TL0;
        while(1)
     {          KQDT=KQD;          GIAIMA_ASCII();
                HIENTHI_LCD();
                do {KQD=TL0;          if (KQD == 200)     TL0=0;}
                while(KQD==KQDT);
     }
}
```

❖ **Giải thích chương trình:**

Trong chương trình chính lấy kết quả từ thanh ghi TL0 gán cho biến KQD và KQDT, tiến hành giải mã và hiển thị trên LCD. Thực hiện đọc kết quả gán cho biến KQD và so sánh bằng 200 thì cho xóa về 0 và vòng lặp do while chỉ thoát khi có kết quả khác với kết quả hiển thị.

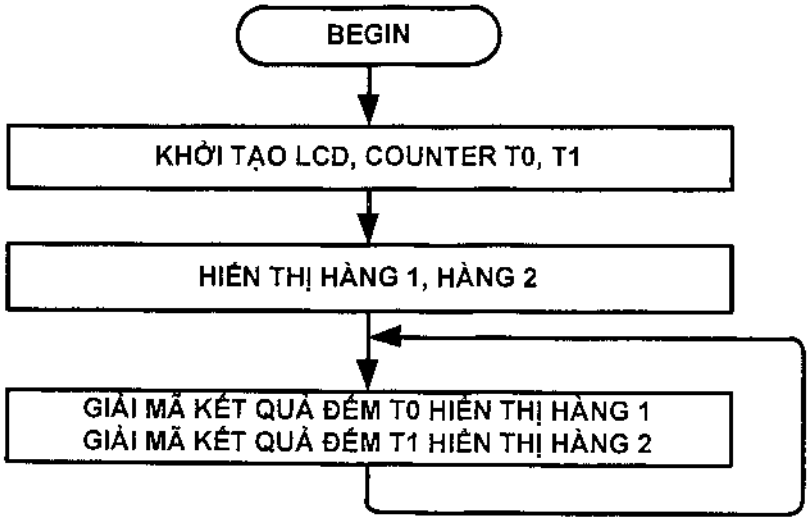
Bài 7-6: Dùng vi điều khiển AT89S52 đếm xung ngoại 1 kênh dùng T0 kết quả đếm hiển thị trên LCD 16x2, giới hạn đếm từ 0 đến 200, khi bằng 200 thì quay về 0. Giá trị đếm T0 hiển thị hàng 2.

Sơ đồ mạch: mạch cảm biến tạo xung cũng giống như đã trình bày.



Hình 7-22: *Mạch đếm sản phẩm hai kênh T0, T1 hiển thị trên LCD.*

➢ *Lưu đồ:*



Hình 7-23: *Lưu đồ đếm sản phẩm 2 kênh hiển thị trên LCD.*

➢ ***Chương trình Keil-C:***

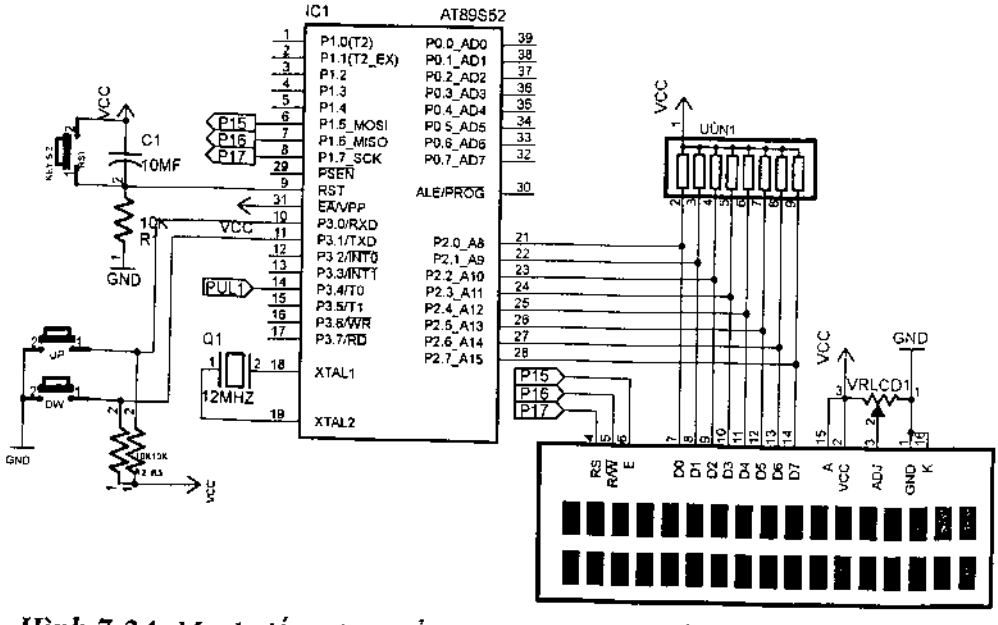
```
#include <AT89X52.h>
#include <THUVIEN_LCD16X2.C>
unsigned char j;
unsigned char KQDTC0,TRAM0,CHUC0,DONVI0;
unsigned char KQDTC1,TRAM1,CHUC1,DONVI1;
const unsigned char HANG1[16]={"COUNTER T0:    "};
const unsigned char HANG2[16]={"COUNTER T1:    "};

void GIAIMA_ASCII_C0 (unsigned char KQDC0)
{     DONVI0=KQDC0%10;   KQDC0=KQDC0/10;
      CHUC0=KQDC0%10;              TRAM0=KQDC0/10;
      DONVI0=DONVI0+0x30;          CHUC0=CHUC0+0x30;
      TRAM0=TRAM0+0x30;
      if (TRAM0==0X30)
            { TRAM0=' ';      if (CHUC0==0X30)          CHUC0=' '; }
}
```

```
void GIAIMA_ASCII_C1 (unsigned char KQDC1)
{     DONVI1=KQDC1%10;              KQDC1=KQDC1/10;
      CHUC1=KQDC1%10;               TRAM1=KQDC1/10;
      DONVI1=DONVI1+0x30;           CHUC1=CHUC1+0x30;
      TRAM1=TRAM1+0x30;
      if (TRAM1==0X30)
            { TRAM1=' ';      if (CHUC1==0X30)          CHUC1=' '; }
}
void HIENTHI_LCD_C0 ()
{           COMMAND_WRITE(0x8D);
      DATA_WRITE(TRAM0);
            DATA_WRITE(CHUC0);
      DATA_WRITE(DONVI0);
}
void HIENTHI_LCD_C1 ()
{     COMMAND_WRITE(0xCD);      DATA_WRITE(TRAM1);
      DATA_WRITE(CHUC1);        DATA_WRITE(DONVI1);
}
void main ( )
{     SETUP_LCD( );
      COMMAND_WRITE(addr_line1);          delay(10);
      for( j=0;j<16;j++)          {DATA_WRITE(HANG1[j]);}
      COMMAND_WRITE(addr_line2);          delay(10);
      for( j=0;j<16;j++)          {DATA_WRITE(HANG2[j]);}
      TMOD = T1_CT_+T1_M0_+ T0_CT_+T0_M0_;
      TR1=1;     TR0=1;
      GIAIMA_ASCII_C0 (TL0);      HIENTHI_LCD_C0();
      GIAIMA_ASCII_C1 (TL1);      HIENTHI_LCD_C1();
      KQDTC0=TL0;           KQDTC1=TL1;
      while(1)
```

```
    {
        if(KQDTC0!=TL0)  {      KQDTC0=TL0;
        GIAIMA_ASCII_C0(TL0); HIENTHI_LCD_C0();}
        if(KQDTC1!=TL1)  {      KQDTC1=TL1;
        GIAIMA_ASCII_C1(TL1); HIENTHI_LCD_C1();}
    }
}
```

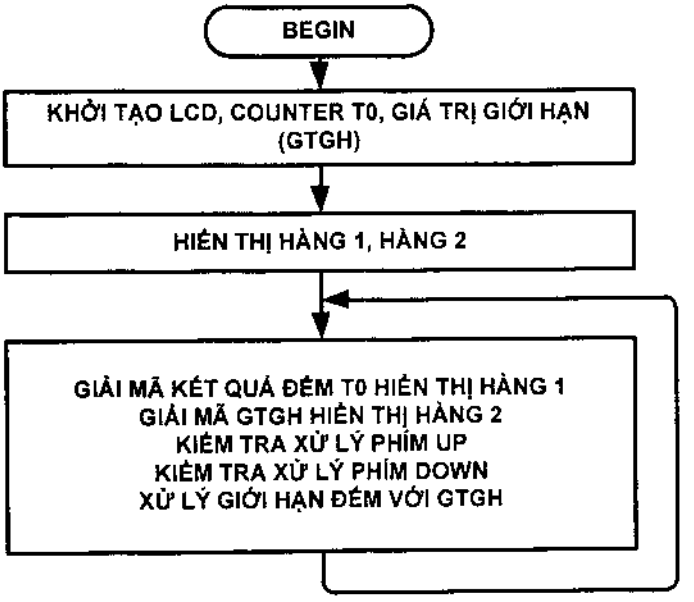
Bài 7-7: Dùng vi điều khiển AT89S52 đếm xung ngoại 1 kênh dùng T0 kết quả đếm hiển thị trên LCD 16×2 ở hàng 1, giới hạn đếm hiển thị ở hàng 2 được thiết lập bằng 2 nút nhấn UP, DOWN, giới hạn nhỏ nhất là 10 và lớn nhất là 250, mặc nhiên hiển thị giá trị 10.

Sơ đồ mạch: mạch cảm biến tạo xung cũng giống như đã trình bày.



Hình 7-24: *Mạch đếm sản phẩm 1 kênh T0, có thiết lập giá trị giới hạn, hiển thị trên LCD.*

➢ ***Lưu đồ:***



Hình 7-25: *Lưu đồ đếm sản phẩm 1 kênh, có cài đặt giá trị giới hạn, hiển thị trên LCD.*

➢ ***Chương trình Keil-C:***

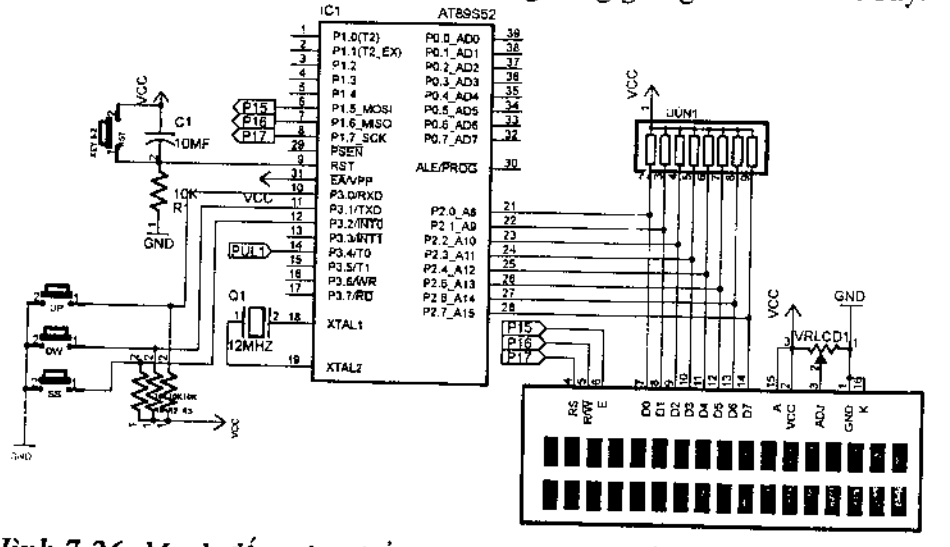
```
#include <AT89X52.h>
#include <THUVIEN_LCD16X2.C>
#define UP        P3_0
#define DOWN   P3_1

unsigned char j;
unsigned char KQD,KQDT,TRAM,CHUC,DONVI;
unsigned char GTGH,TRAMGH,CHUCGH,DONVIGH;
const unsigned char HANG1[16]={"SAN PHAM DEM:   "};
const unsigned char HANG2[16]={"GIA TRI CAI :   "};

void GIAIMA_ASCII_C0 (unsigned char KQDC0)
{     DONVI=KQDC0%10;              KQDC0=KQDC0/10;
      CHUC=KQDC0%10;               TRAM=KQDC0/10;
      DONVI=DONVI+0x30;  CHUC=CHUC+0x30;
TRAM=TRAM+0x30;
```

```
    if (TRAM==0X30)
        { TRAM=' ';     if (CHUC==0X30)          CHUC=' '; }
}

void GIAIMA_ASCII_GH (unsigned char TAM)
{   DONVIGH=TAM%10;          TAM=TAM/10;
    CHUCGH=TAM%10;           TRAMGH=TAM/10;

    DONVIGH=DONVIGH+0x30;    CHUCGH=CHUCGH+0x30;
    TRAMGH=TRAMGH+0x30;
    if (TRAMGH==0X30)
        { TRAMGH=' ';
            if (CHUCGH==0X30) { CHUCGH=' ';    }
}       }
void HIENTHI_LCD_C0 ()
{   COMMAND_WRITE(0x8D);     DATA_WRITE(TRAM);
     DATA_WRITE(CHUC);       DATA_WRITE(DONVI);
}
void HIENTHI_LCD_GH ()
{   COMMAND_WRITE(0xCD);     DATA_WRITE(TRAMGH);
    DATA_WRITE(CHUCGH);      DATA_WRITE(DONVIGH);
}
void CHONGDOI_UP ()
{   if (UP == 0)
        {   delay (10000);
            if (UP == 0)
                {       GTGH++;
                        if (GTGH==251) GTGH =250;
                        GIAIMA_ASCII_GH(GTGH);
                        HIENTHI_LCD_GH ();
                        delay(20000);
}       }       }
```

```
void CHONGDOI_DOWN ()
{     if (DOWN == 0)
            {     delay (10000);
                  if (DOWN == 0)
                        {     GTGH--;
                              if (GTGH==9) GTGH =10;
                              GIAIMA_ASCII_GH(GTGH);
                              HIENTHI_LCD_GH ();
                              delay(20000);
}}}
void main ( )
{     SETUP_LCD( );
      COMMAND_WRITE(addr_line1);          delay(10);
      for( j=0;j<16;j++)          {DATA_WRITE(HANG1[j]);}

      COMMAND_WRITE(addr_line2);          delay(10);
      for( j=0;j<16;j++)          {DATA_WRITE(HANG2[j]);}

      TMOD=T0_CT_+T0_M0_;          TR0=1;     KQD=TL0;
      GTGH =10;       GIAIMA_ASCII_GH(GTGH);
                      HIENTHI_LCD_GH ();
      while(1)
       {
                  KQDT=KQD;  GIAIMA_ASCII_C0 (TL0);
                  HIENTHI_LCD_C0 ();
                  do {KQD=TL0;  CHONGDOI_DOWN ();
CHONGDOI_UP ();
                        if (KQD >= GTGH+1) TL0=TL0-GTGH;}
                  while(KQD==KQDT);
       }
}
```

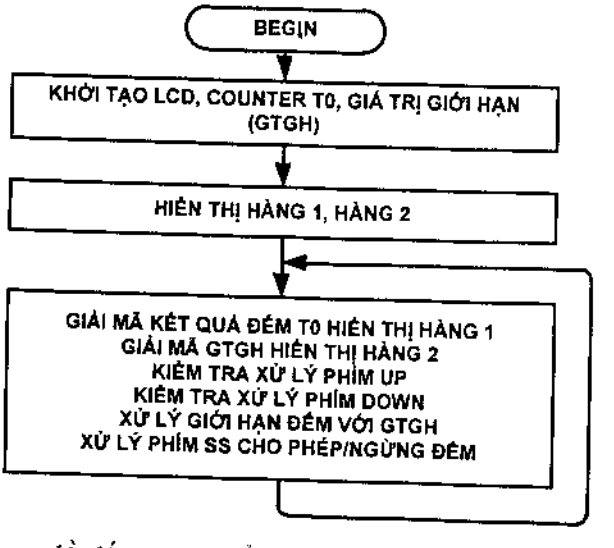
Bài 7-8: Dùng vi điều khiển AT89S52 đếm xung ngoại 1 kênh dùng T0 kết quả đếm hiển thị trên LCD 16x2 ở hàng 1, giới hạn đếm hiển thị ở hàng 2 được thiết lập bằng hai nút nhấn UP, DOWN, giới hạn nhỏ nhất là 10 và lớn nhất là 250, mặc nhiên hiển thị giá trị 10, có 1 nút SS có chức năng khi nhấn thì cho phép counter đếm, đang đếm thì ngừng.

➢ ***Sơ đồ mạch:*** mạch cảm biến tạo xung cũng giống như đã trình bày.



Hình 7-26: Mạch đếm sản phẩm 1 kênh T0, có thiết lập giá trị giới hạn, đếm-ngừng, hiển thị trên LCD.

➢ ***Lưu đồ:***



Hình 7-27: Lưu đồ đếm sản phẩm 1 kênh, có cài đặt giá trị giới hạn, đếm ngừng, hiển thị trên LCD.

➢ **Chương trình Keil-C:**

```
#include <THUVIEN_LCD16X2.C>
#define UP      P3_0
#define DOWN    P3_1
#define SS      P3_2
unsigned char j;
unsigned char KQD,KQDT,TRAM,CHUC,DONVI;
unsigned char GTGH,TRAMGH,CHUCGH,DONVIGH;
const unsigned char HANG1A[16]={"DANG DEM:     "};
const unsigned char HANG1B[16]={"NGUNG DEM:    "};
const unsigned char HANG2[16]={"GIA TRI CAI :  "};
void GIAIMA_ASCII_C0 (unsigned char KQDC0)
{   DONVI=KQDC0%10;              KQDC0=KQDC0/10;
    CHUC=KQDC0%10;               TRAM=KQDC0/10;
    DONVI=DONVI+0x30;  CHUC=CHUC+0x30;
    TRAM=TRAM+0x30;
    if (TRAM==0X30)
        { TRAM=' ';      if (CHUC==0X30)          CHUC=' '; }
}
void GIAIMA_ASCII_GH (unsigned char TAM)
{   DONVIGH=TAM%10;              TAM=TAM/10;
    CHUCGH=TAM%10;               TRAMGH=TAM/10;
    DONVIGH=DONVIGH+0x30;  CHUCGH=CHUCGH+0x30;
    TRAMGH=TRAMGH+0x30;
    if (TRAMGH==0X30)
        { TRAMGH=' ';
            if (CHUCGH==0X30) { CHUCGH=' ';    }
}       }
void HIENTHI_LCD_C0 ()
{       COMMAND_WRITE(0x8D);DATA_WRITE(TRAM);
        DATA_WRITE(CHUC);   DATA_WRITE(DONVI);}
```

```
void HIENTHI_LCD_GH ()
{           COMMAND_WRITE(0xCD); DATA_WRITE(TRAMGH);
            DATA_WRITE(CHUCGH); DATA_WRITE(DONVIGH);}
void CHONGDOI_UP ()
{     if (UP == 0)
            {     delay (10000);
                  if (UP == 0)
                        {     GTGH++;
                              if (GTGH==251) GTGH =250;
                              GIAIMA_ASCII_GH(GTGH);
                              HIENTHI_LCD_GH ();
                              delay(20000);
}           }           }
void CHONGDOI_DOWN ()
{     if (DOWN == 0)
            {     delay (10000);
                  if (DOWN == 0)
                        {     GTGH--;
                              if (GTGH==9) GTGH =10;
                              GIAIMA_ASCII_GH(GTGH);
                              HIENTHI_LCD_GH ();
                              delay(20000);
}}}
void CHONGDOI_SS ()
{     if (SS == 0)
            {     delay (10000);
                  if (SS == 0)
                        {     TR0=~TR0;
                              if (TR0==0){ COMMAND_WRITE(0x80);
                              for (j=0;j<12;j++)
                              { DATA_WRITE (HANG1B[j]);}}
```

```
                        else    {COMMAND_WRITE(0x80);
                        for (j=0;j<12;j++)
                        {  DATA_WRITE (HANG1A[j]);}}
                        do{}
                        while(SS==0);
}}}
void main ( )
{     SETUP_LCD( );
      COMMAND_WRITE(addr_line1);          delay(10);
      for( j=0;j<16;j++)        {DATA_WRITE(HANG1B[j]);}

      COMMAND_WRITE(addr_line2);          delay(10);
      for( j=0;j<16;j++)        {DATA_WRITE(HANG2[j]);}
      TMOD=T0_CT_+T0_M0_;       TR0=1;    KQD=TL0;
      GTGH =10;       GIAIMA_ASCII_GH(GTGH);
                      HIENTHI_LCD_GH ();
      while(1)
       {
           KQDT=KQD;        GIAIMA_ASCII_C0 (TL0);
                HIENTHI_LCD_C0 ();
                do {KQD=TL0;  CHONGDOI_DOWN ();
                CHONGDOI_UP ();
                     CHONGDOI_SS ();
                     if (KQD >= GTGH+1) TL0=TL0-GTGH;}
                while(KQD==KQDT);
       }
}
```

IV. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP

1. Câu hỏi ôn tập

Câu số 7-1: Hãy cho biết tên các thanh ghi liên quan đến timer T0 và T1 của vi điều khiển AT89S52.

Câu số 7-2: Hãy cho biết chức năng của các bit trong thanh ghi TMOD của vi điều khiển AT89S52.

Câu số 7-3: Hãy cho biết chức năng của các bit trong thanh ghi TCON của vi điều khiển AT89S52.

Câu số 7-4: Hãy giải thích hoạt động timer ở chế độ mode 0 của vi điều khiển AT89S52.

Câu số 7-5: Hãy giải thích hoạt động timer ở chế độ mode 1 của vi điều khiển AT89S52.

Câu số 7-6: Hãy giải thích hoạt động timer ở chế độ mode 2 của vi điều khiển AT89S52.

2. Câu hỏi mở rộng

Câu số 7-7: Hãy tìm hiểu các timer của vi điều khiển AT89C51RD2 và so sánh với AT89S52.

3. Câu hỏi trắc nghiệm

Câu 7-1: Trong AT89S52 thì các thanh ghi có liên quan đến Timer:

(a) TH0, TL0, SBUF

(b) TCON, SCON

(c) TH0, TL0, TCON, TMOD, TH1, TL1

(d) DPTR, TMOD, TH1, TL1

Câu 7-2: Các timer của AT89S52 gồm có:

(a) T0

(b) T0, T1

(c) T0, T1, T2

(d) T0, T1, T2, T4

Câu 7-3: Trong AT89S52 thì các timer là:

(a) 8 bit

(b) 10 bit

(c) 12 bit

(d) 16 bit

Câu 7-4: Trong AT89S52 thì timer T0 có thể hoạt động ở:

(a) 1 mode

(b) 3 mode

(c) 2 mode

(d) 4 mode

Câu 7-5: Trong AT89S52 thì mod nào hoạt động đếm 16 bit:

(a) Mode 0

(b) Mode 1

(c) Mode 2

(d) Mode 3

Câu 7-6: Trong AT89S52 thì bit nào lựa chọn mode hoạt động:

(a) $C/\overline{T}$ (b) M_1M_0

(c) Gate (d) TR0

Câu 7-7: Trong AT89S52 thì bit nào lựa chọn hoạt động timer hay counter:

(a) $C/\overline{T}$ (b) M_1M_0

(c) Gate (d) TR0

Câu 7-8: Trong AT89S52 thì bit nào cho phép timer đếm:

(a) $C/\overline{T}$ (b) M_1M_0

(c) TF0 (d) TR0

Câu 7-9: Trong AT89S52 thì bit nào báo timer tràn:

(a) $C/\overline{T}$ (b) M_1M_0

(c) TF0 (d) TR0

Câu 7-10: Trong AT89S52 khi T1 hoạt động ở mode 2 thì thanh ghi nào chứa giá trị bắt đầu đếm:

(a) TH1 (b) TL1

(c) TH0 (d) TL0

Câu 7-11: Trong AT89S52 thì mod 0 đếm bao nhiêu bit:

(a) 8 bit (b) 16 bit

(c) 13 bit (d) 32 bit

Câu 7-12: Trong AT89S52 thì mod 2 đếm bao nhiêu bit:

(a) 8 bit (b) 16 bit

(c) 13 bit (d) 32 bit

Câu 7-13: Timer T0 đếm xung nội, mod 1 với giá trị bắt đầu là 50000 thì timer tràn khi đếm thêm:

(a) 50000 xung (b) 15536 xung

(c) 15000 xung (d) 65536 xung

Câu 7-14: Timer nào của AT89S52 có thể đếm lên đếm xuống:

(a) T0 (b) T0 và T1

(c) T1 (d) T2

Câu 7-15: Timer nào của AT89S52 có thể đếm hoạt động ở chế độ capture:

(a) T0 (b) T0 và T1

(c) T1 (d) T2

4. Bài tập

Bài tập 7-1: Một mạch đếm sản phẩm dùng vi điều khiển AT89S52 giao tiếp với ba led 7 đoạn anode chung, mạch tạo xung khi phát hiện sản phẩm đưa đến counter T1.

Hãy vẽ mạch, viết lưu đồ và chương trình đếm xung với giới hạn đếm từ 0 đến 200.

Bài tập 7-2: Mạch đếm sản phẩm hai kênh dùng vi điều khiển AT89S52 giao tiếp với 6 led 7 đoạn anode chung theo phương pháp quét, hai mạch tạo xung đưa đến counter T0, T1.

Hãy vẽ mạch, viết lưu đồ và chương trình đếm xung hai kênh với giới hạn đếm từ 0 đến 200.

Bài tập 7-3: Một mạch đếm sản phẩm dùng vi điều khiển AT89S52 giao tiếp với ba led 7 đoạn anode chung, mạch tạo xung khi phát hiện sản phẩm đưa đến counter T1, có ba nút nhấn RUN, STOP, CLEAR.

Hãy vẽ mạch, viết lưu đồ và chương trình đếm xung với giới hạn đếm từ 0 đến 200 và hoạt động như sau: khi cấp điện thì mạch ngừng đến hiển thị số 0, khi nhấn RUN thì mạch bắt đầu đếm khi có xung, khi nhấn STOP thì mạch ngừng tại giá trị đang đếm, khi nhấn CLEAR thì xóa giá trị đếm về 0 và ngừng luôn.

Bài tập 7-4: Một mạch đếm sản phẩm dùng vi điều khiển AT89S52 giao tiếp với ba led 7 đoạn anode chung, mạch tạo xung khi phát hiện sản phẩm đưa đến counter T1.

Hãy vẽ mạch, viết lưu đồ và chương trình đếm xung với giới hạn đếm từ 200 đến 0, kết quả hiển thị dạng đếm xuống.

Chương 8
VI ĐIỀU KHIỂN AT89S52: GIAO TIẾP ADC

- **GIỚI THIỆU**
- **VI ĐIỀU KHIỂN ATMEL AT89S52 GIAO TIẾP ADC 0809**
  - KHẢO SÁT VI MẠCH ADC 0809
    - ✓ *Các thông số của vi mạch ADC 0809*
    - ✓ *Sơ đồ chân và sơ đồ cấu trúc bên trong của vi mạch ADC 0809*
    - ✓ *Chức năng các tín hiệu của vi mạch ADC 0809*
    - ✓ *Dạng sóng thực hiện quá trình chuyển đổi của vi mạch ADC 0809*
  - ỨNG DỤNG ĐO NHIỆT ĐỘ DÙNG AT89S52 VÀ ADC 0809
    - ✓ *Yêu cầu*
    - ✓ *Thiết kế sơ đồ mạch*
    - ✓ *Tính toán độ phân giải*
    - ✓ *Lưu đồ chuyển đổi*
    - ✓ *Chương trình chuyển đổi*
- **CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP**
  - CÂU HỎI ÔN TẬP
  - CÂU HỎI MỞ RỘNG
  - CÂU HỎI TRẮC NGHIỆM
  - BÀI TẬP

I. GIỚI THIỆU

Ở chương này khảo sát vi điều khiển giao tiếp với vi mạch chuyển đổi tương tự sang số (ADC).

Sau khi kết thúc chương này, bạn có thể kết nối vi điều khiển không có tích hợp bộ chuyển đổi ADC với các vi mạch ADC, sử dụng được vi điều khiển có tích hợp ADC, biết trình tự thực hiện quá trình chuyển đổi ADC, biết tính toán độ phân giải, chuyển đổi và tính trung bình kết quả.

II. VI ĐIỀU KHIỂN ATMEL AT89S52 GIAO TIẾP ADC 0809

Vi điều khiển AT89S52 không tích hợp bộ chuyển đổi tín hiệu tương tự sang số (ADC), đối với các ứng dụng cần xử lý tín hiệu tương tự thì phải kết nối thêm vi mạch chuyển đổi ADC. Tùy thuộc vào yêu cầu như độ phân giải, giá trị chuyển đổi lớn nhất, thời gian đáp ứng, số lượng kênh tín hiệu tương tự mà chọn vi mạch ADC cho phù hợp.

Trong tài liệu này sử dụng vi mạch ADC 0809 có số bit là 8 và có 8 kênh tương tự.

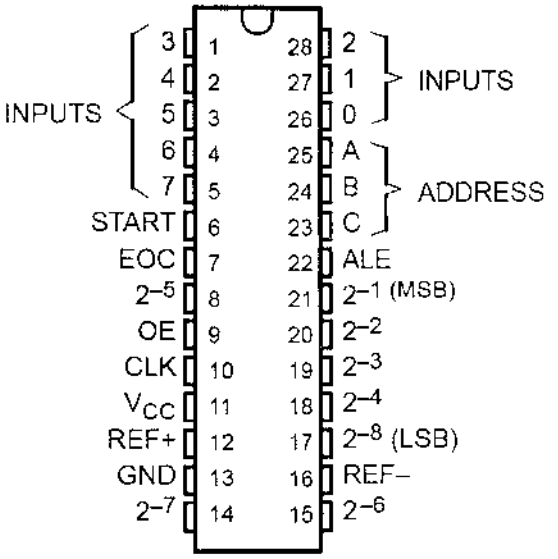
1. Khảo sát vi mạch ADC 0809

➢ ***Các thông số của vi mạch ADC 0809***

- ADC 8 bit.
- Thời gian chuyển đổi 100μs.
- Dễ giao tiếp với vi xử lý hoặc vi điều khiển.
- Các ngõ ra ba trạng thái có chốt.
- Các ngõ vào địa chỉ có chốt.
- Dùng nguồn 5V.

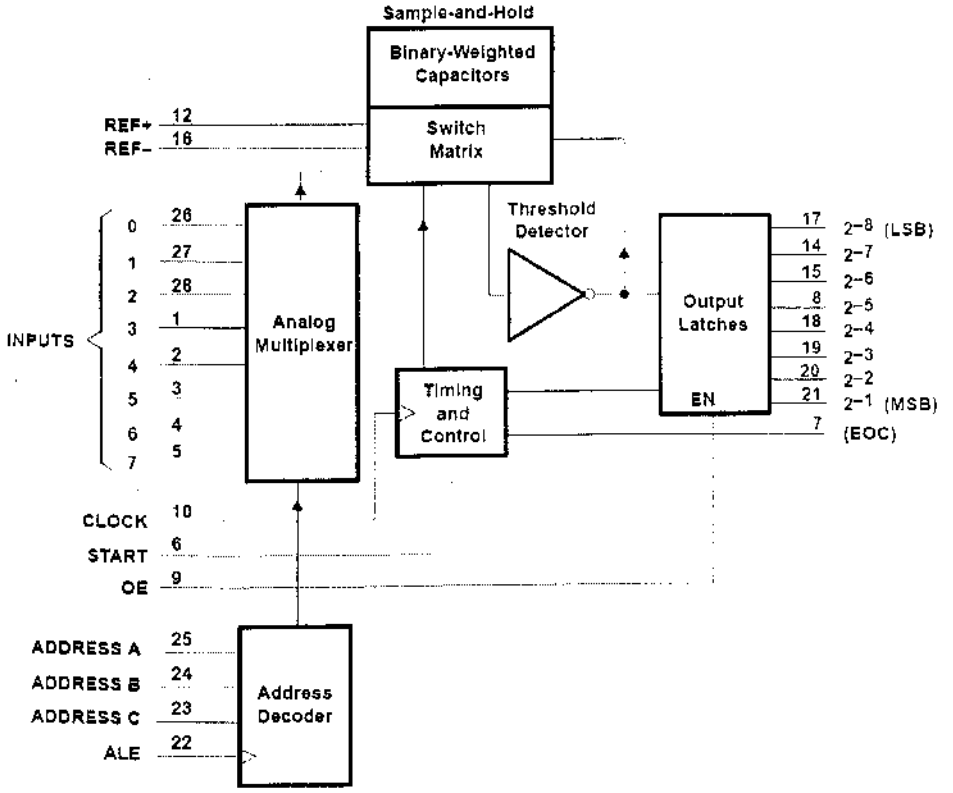
➢ ***Sơ đồ chân và sơ đồ cấu trúc bên trong của vi mạch ADC 0809***

Vi mạch có sơ đồ chân như hình 8-1:



Hình 8-1: *Sơ đồ chân IC ADC 0809.*

Sơ đồ cấu trúc bên trong của vi mạch ADC 0809 như hình 8-2:



Hình 8-2: *Sơ đồ khối bên trong IC ADC 0809.*

➢ ***Chức năng các tín hiệu của vi mạch ADC 0809:*** Vi mạch có 8 kênh tương tự vào có đánh số từ 0 đến 7 qua mạch đa hợp với ba tín hiệu là A, B, C để một chọn kênh cần chuyển đổi, tín hiệu ALE (Address latch enable) dùng để chốt ba tín hiệu chọn kênh.

Tín hiệu clock dùng để nhận xung clock cấp cho mạch hoạt động hay thực hiện chuyển đổi.

Tín hiệu Start dùng để điều khiển thực hiện quá trình chuyển đổi.

Tín hiệu EOC (End of conversion) dùng để báo hiệu quá trình chuyển đổi hoàn tất.

Tín hiệu OE (Output enable) dùng để cho phép xuất dữ liệu sau khi chuyển đổi xong.

Dữ liệu số 8 bit sẽ xuất ở 8 ngõ ra.

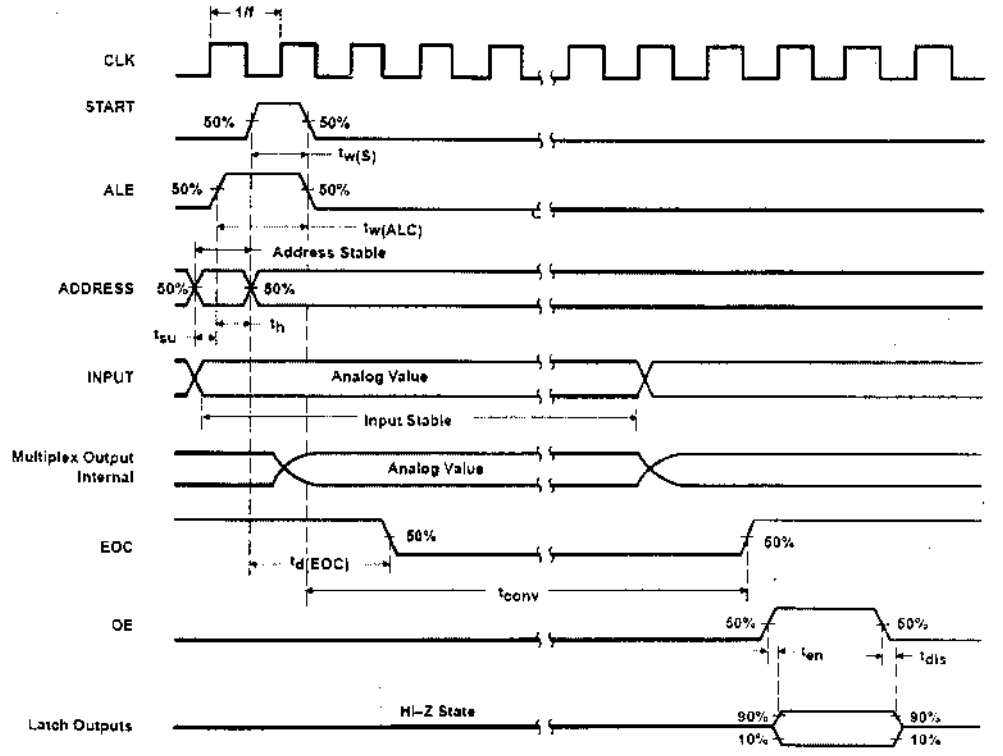
Mạch có nguồn điện áp tham chiếu Vref+ và Vref- để thiết lập độ phân giải.

Bảng 8-1: Trạng thái IC ADC 0809 như sau:

Ngõ vào				**Chọn kênh thứ**
Địa chỉ				
C	B	A	ALE	
0	0	0	↑	0
0	0	1	↑	1
0	1	0	↑	2
0	1	1	↑	3
1	0	0	↑	4
1	0	1	↑	5
1	1	0	↑	6
1	1	1	↑	7

➢ ***Dạng sóng thực hiện quá trình chuyển đổi của vi mạch ADC 0809***

Dạng sóng chuyển đổi được cung cấp bởi nhà tạo như hình 8-3:



Hình 8-3: *Dạng sóng chuyển đổi của ADC 0809.*

Để thực hiện quá trình chuyển đổi thì phải có xung clock, quá trình chuyển đổi như sau:

- Tín hiệu tương tự cần chuyển đổi được đưa đến ngõ vào tương tự.
- Tiến hành chọn kênh tương tự bằng ba đường địa chỉ chọn kênh.
- Điều khiển tín hiệu ALE từ 0 lên 1 để chốt địa chỉ.
- Điều khiển tín hiệu START từ 0 lên 1.
- Điều khiển cả ALE và START từ 1 về 0 và quá trình chuyển đổi bắt đầu.
- Sau 1 khoảng thời gian thì tín hiệu EOC chuyển trạng thái từ 1 về 0 để cho biết quá trình chuyển đổi đang xảy ra, khi chuyển đổi xong thì tín hiệu EOC chuyển sang mức 1. Mạch điều khiển kiểm tra tín hiệu EOC để biết quá trình chuyển đổi kết thúc và tiến hành nhận dữ liệu.
- Để cho phép xuất dữ liệu số thì tín hiệu OE phải ở mức 1.

Thời gian chuyển đổi được tính từ khi có cạnh xuống của tín hiệu START cho đến khi tín hiệu EOC lên 1 trở lại.

Theo tài liệu vi mạch, tần số hoạt động của xung clock lớn nhất là 1280kHz và thời gian chuyển đổi nhỏ nhất là 90μs và dài nhất là 116μs.

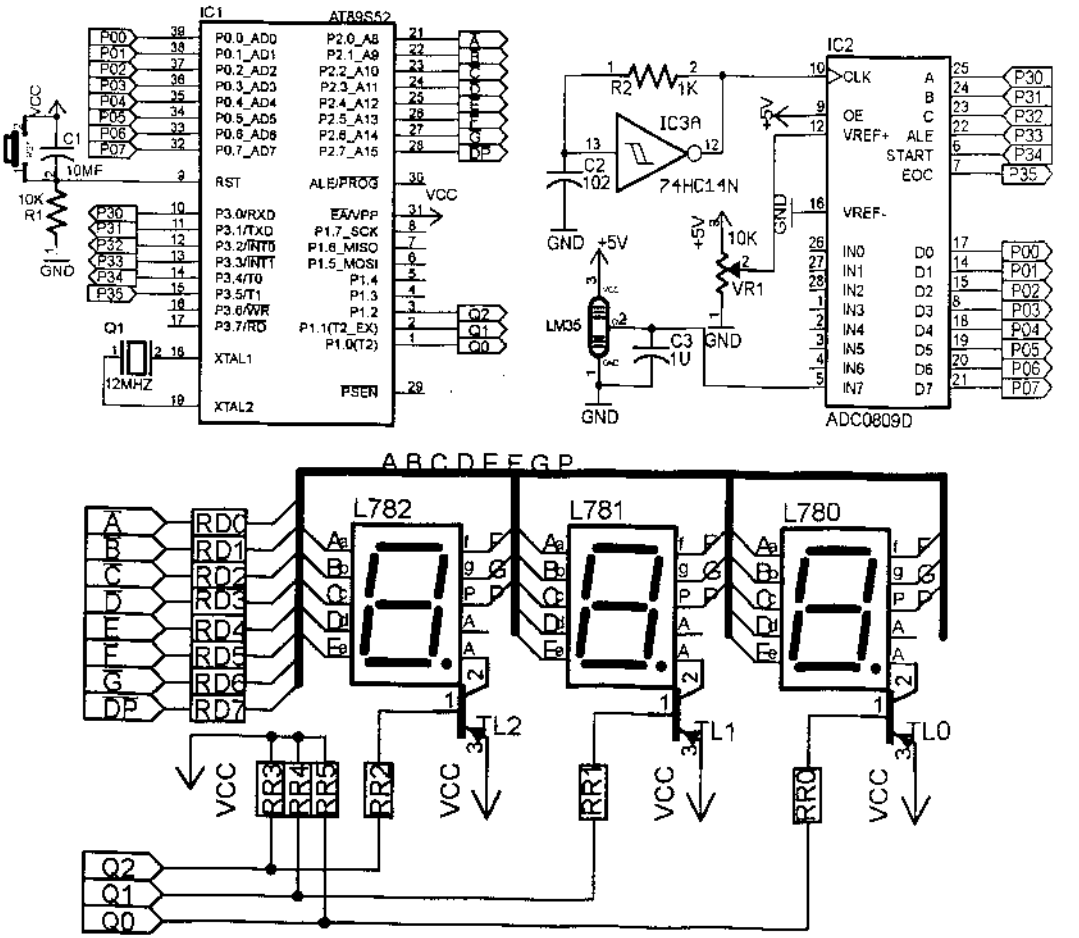
2. Ứng dụng đo nhiệt độ dùng AT89S52 và ADC 0809

Bài 8-1: Dùng vi điều khiển AT89S52 giao tiếp với ADC 0809, cảm biến nhiệt LM35 và ba led 7 đoạn theo phương pháp quét để hiển thị nhiệt độ.

➢ **Sơ đồ mạch:**

Vi điều khiển AT89S52 có bốn port có thể chọn tùy ý các port giao tiếp với ADC 0809, tuy nhiên nếu sử dụng chế độ nạp nối tiếp ISP thì không được dùng các chân MOSI, MISO, SCK để nhận tín hiệu từ ADC do xung đột tín hiệu.

Mạch được thiết kế như hình 8-4:



Hình 8-4: *Mạch giao tiếp vi điều khiển AT89S52 với ADC 0809.*

Cảm biến LM35 nối với ngõ vào kênh tương tự IN7, tụ C3 dùng để giảm bớt độ nhạy của cảm biến.

Ngõ vào Vref- nối mass và Vref+ nối với biến trở để tạo độ phân giải cho ADC tương thích với tín hiệu đo.

Cổng Not cùng với tụ C2 và R2 tạo thành mạch dao động cấp xung cho ADC hoạt động.

Dùng port0 nhận dữ liệu số từ ADC và port3 điều khiển chọn kênh và chuyển đổi ADC.

Port 1 và 2 điều khiển led 7 đoạn quét led để hiển thị kết quả đo.

➢ **Tính toán độ phân giải**

Độ phân giải của ADC được tính theo công thức:

$$SS = \frac{V_{REF+} - V_{REF-}}{(2^n - 1)}$$

Kí hiệu độ phân giải là SS (Step size), n là số bit của ADC.

Theo thông số của cảm biến LM35 thì hệ số chuyển đổi nhiệt độ sang điện áp tương tự là 10mV/1°C, tầm nhiệt độ đo được từ -50 °C đến 150 °C.

Để đo và hiển thị đúng nhiệt độ đo thì độ phân giải của ADC phải trùng với độ phân giải của cảm biến LM35 nên ta chọn SS = 10mV.

Từ công thức (8-1) ta tính được:

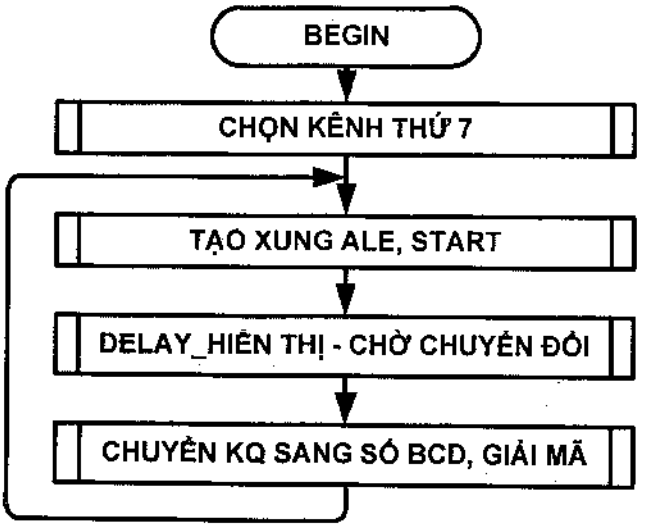
$$V_{REF+} - V_{REF-} = SS \times (2^n - 1) = 10mV \times 255 = 2550mV$$

Cho Vref- = 0V thì tính được $V_{REF+} = 2550mV$

Trong mạch có sử dụng biến trở VR1 để hiệu chỉnh tạo ra đúng giá trị điện áp trên.

➢ **Lưu đồ chuyển đổi**

Theo sơ đồ mạch đã kết nối và trình tự chuyển đổi của ADC thì lưu đồ chuyển đổi như hình 8-5:



Hình 8-5: Lưu đồ chuyển đổi của ADC 0809.

❖ **Chương trình chuyển đổi**

```
#include < AT89X52.h>
#define CHANNEL_A       P3_0
#define CHANNEL_B       P3_1
#define CHANNEL_C       P3_2
#define ALE             P3_3
#define START           P3_4
unsigned char KQADC ;
unsigned char TRAM,CHUC,DONVI,k;
unsigned char MTRAM,MCHUC,MDONVI;
unsigned char MA7D[10] =
{0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90};
void delay(unsigned int x)
    { unsigned int y;
        for(y=0;y<x;y++)    {}
    }
void CHUYEN_DOI_ADC()
```

```
    {    START = 1;    ALE = 1;  START = 0;    ALE = 0;   }
void HEX_BCD_GIAIMA()
    {
        DONVI = KQADC %10;          KQADC = KQADC/10;
        TRAM = KQADC /10;           CHUC = KQADC %10;
        MDONVI = MA7D[DONVI];       MCHUC = MA7D[CHUC];
        MTRAM = MA7D[TRAM];
        }
void DELAY_HTHI_QUET_3LED()
    {   for (k=0;k<5;k++)
        {
            P2=MDONVI;  P1_0 = 0;  delay (100);    P1_0 = 1;
            P2=MCHUC;   P1_1 = 0;  delay (100);    P1_1 = 1;
            P2=MTRAM;   P1_2 = 0;  delay (100);    P1_2 = 1;
        }
    }
void MAIN ()
{   CHANNEL_A = 1;   CHANNEL_B = 1;   CHANNEL_C = 1;
    while(1)
        {   CHUYEN_DOI_ADC();      delay (50);
            {   KQADC = P0;
                HEX_BCD_GIAIMA();
                DELAY_HTHI_QUET_3LED();
            }
        }
}
```

Chương trình trên chạy thực tế thì kết quả đo còn nhấp nháy, để giảm nhấp nháy và đo chính xác hơn thì ta tiến hành đo 10 lần rồi chia kết quả cho 10 sẽ được nhiệt độ đo trung bình và kết quả hiển thị không nhấp nháy, chương trình như sau:

```
#include < AT89X52.h>
#define CHANNEL_A       P3_0
#define CHANNEL_B       P3_1
#define CHANNEL_C       P3_2
#define ALE             P3_3
#define START           P3_4
unsigned int KQADC ;
unsigned char TRAM,CHUC,DONVI,k,i;
unsigned char MTRAM,MCHUC,MDONVI;
unsigned char MA7D[10] =
{0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90};
void delay(unsigned int x)
    { unsigned int y;
        for(y=0;y<x;y++){ }
    }
void CHUYEN_DOI_ADC()
    {   START = 1;    ALE = 1;  START = 0;    ALE = 0;  }
void HEX_BCD_GIAIMA()
    {
        DONVI = KQADC %10;          KQADC = KQADC/10;
        TRAM = KQADC /10;           CHUC = KQADC %10;
        MDONVI = MA7D[DONVI];       MCHUC = MA7D[CHUC];
        MTRAM = MA7D[TRAM];
        }

void DELAY_HTHI_QUET_3LED()
    {   for (k=0;k<5;k++)
        {
```

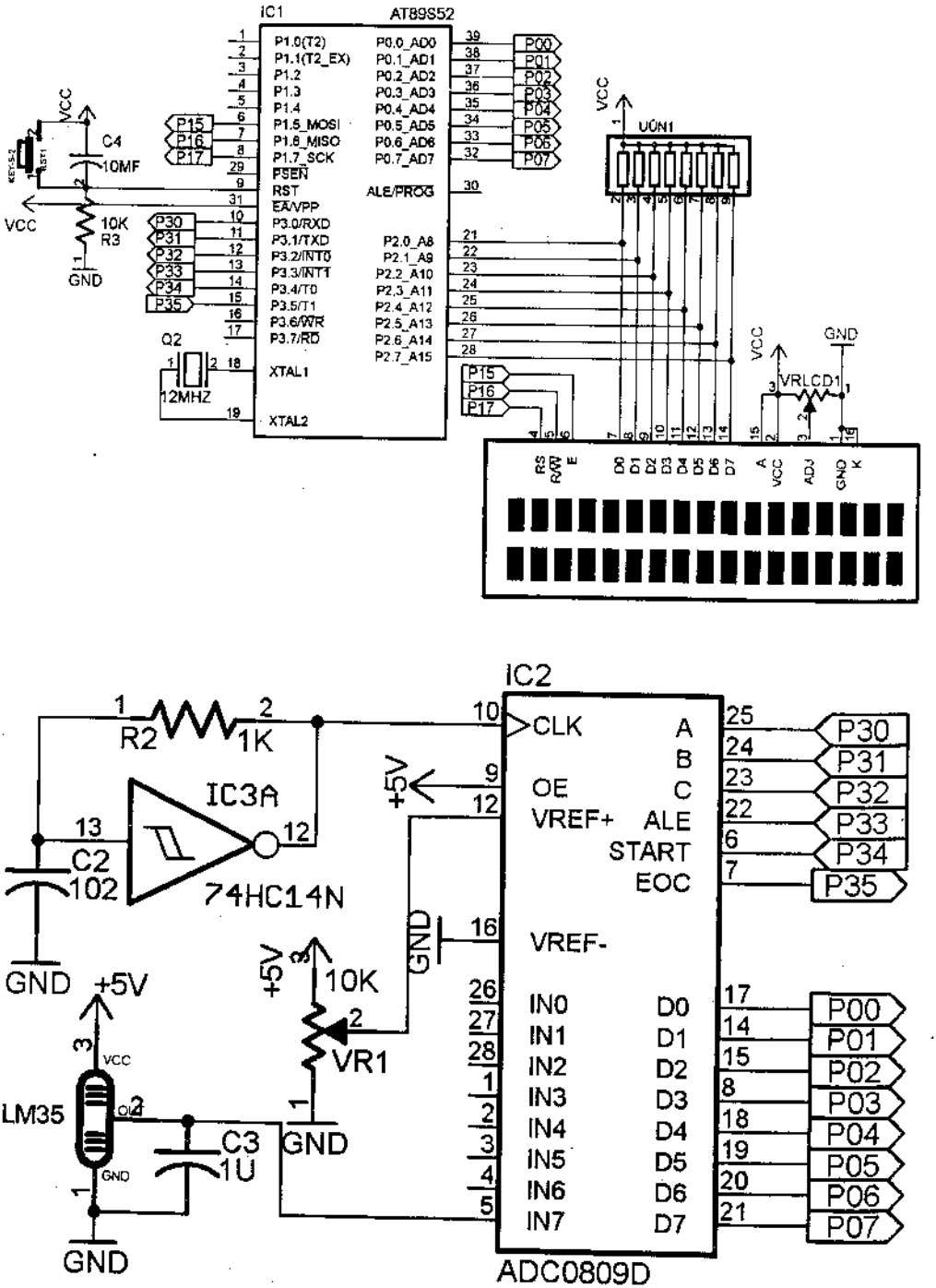
```
                P2=MDONVI;   P1_0 = 0;  delay (100);     P1_0 = 1;
                P2=MCHUC;    P1_1 = 0;  delay (100);     P1_1 = 1;
                P2=MTRAM;    P1_2 = 0;  delay (100);     P1_2 = 1;
                }
    }
void MAIN ()
{   CHANNEL_A = 1;    CHANNEL_B = 1;    CHANNEL_C = 1;
  while(1)
        {   KQADC =0;
            for (i=0;i<10;i++)
            {   CHUYEN_DOI_ADC();
                DELAY_HTHI_QUET_3LED();
                KQADC = KQADC+P0;
            }
            KQADC = KQADC/10;       HEX_BCD_GIAIMA();
        }
}
```

Chương trình này ta thực hiện chương trình con chuyển đổi 10 lần, trong quá trình chờ chuyển đổi ta gọi chương trình con delay có luôn quét hiển thị trên led, sau đó cộng dồn giá trị sau chuyển đổi cho biến KQADC, chia kết quả cho 10 và tiến hành chuyển tách số hàng trăm, hàng chục và đơn vị, giải mã để hiển thị ở lần kế tiếp.

Bài 8-2: Dùng vi điều khiển AT89S52 giao tiếp với ADC 0809, cảm biến nhiệt LM35 và LCD để hiển thị nhiệt độ.

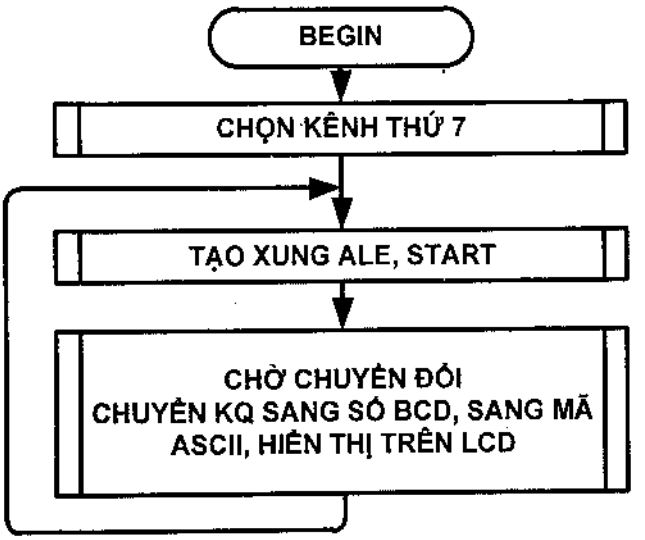
➢ ***Sơ đồ mạch:***

Mạch được thiết kế như hình 8-6, chỉ khác là thay led 7 đoạn bằng LCD:



Hình 8-6: *Mạch giao tiếp vi điều khiển AT89S52 với ADC 0809, LM35, LCD.*

➢ *Lưu đồ chuyển đổi:*



Hình 8-7: Lưu đồ chuyển đổi của ADC 0809, đo nhiệt độ hiển thị LCD.

➢ **Chương trình chuyển đổi**

```
#include <AT89X52.h>
#include <THUVIEN_LCD16X2.C>
#define CHANNEL_A       P3_0
#define CHANNEL_B       P3_1
#define CHANNEL_C       P3_2
#define START           P3_4
#define ALE             P3_3

unsigned int KQADC;
unsigned char j,i;
unsigned char TRAM,CHUC,DONVI;
const unsigned char HANG1[16]={" DH SP KY THUAT "};
const unsigned char HANG2[16]={"NHIET DO DO:    "};
```

```
void GIAIMA_ASCII ()
{     DONVI=KQADC%10;        KQADC=KQADC/10;
      CHUC=KQADC%10;         TRAM=KQADC/10;
      DONVI=DONVI+0x30;      CHUC=CHUC+0x30;
      TRAM=TRAM+0x30;
      if (TRAM==0X30)
            { TRAM=' ';
                  if (CHUC==0X30)          CHUC=' '; }
}

void HIENTHI_LCD()
{           COMMAND_WRITE(0xCD);     DATA_WRITE(TRAM);
            DATA_WRITE(CHUC);        DATA_WRITE(DONVI);
}

void CHUYEN_DOI_ADC()
{     START = 1;     ALE = 1;     START = 0;     ALE = 0;}
void main ()
{     SETUP_LCD( );
      COMMAND_WRITE(addr_line1);       delay(10);
      for( j=0;j<16;j++)       {DATA_WRITE(HANG1[j]);}
      COMMAND_WRITE(addr_line2);       delay(10);
      for( j=0;j<16;j++)       {DATA_WRITE(HANG2[j]);}
      CHANNEL_A = 1;     CHANNEL_B = 1;     CHANNEL_C = 1;
      while(1)
       {    KQADC =0;
            for (i=0; i<10; i++)
            {     CHUYEN_DOI_ADC();        delay(100);
```

```
                KQADC = KQADC+P0;      }
                KQADC = KQADC/10;
                GIAIMA_ASCII ();     HIENTHI_LCD ();
            }
        }
```

Chương trình trên chạy thực tế thì kết quả đo còn nhấp nháy, để giảm nhấp nháy và đo chính xác hơn thì ta tiến hành đo 100 lần rồi chia kết quả cho 100 sẽ được nhiệt độ đo trung bình và kết quả hiển thị không nhấp nháy, chương trình như sau:

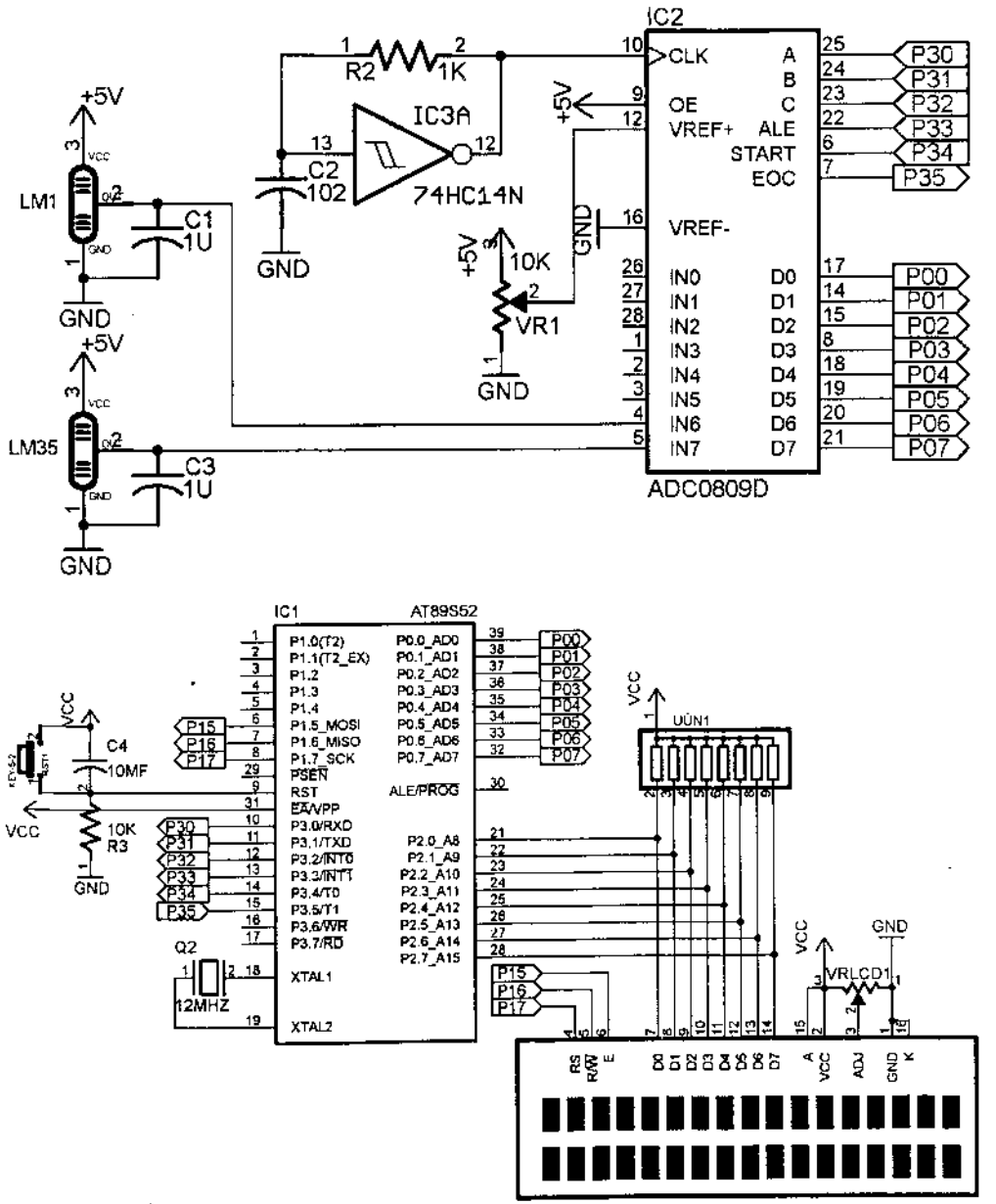
```
#include <AT89X52.h>
#include <THUVIEN_LCD16X2.C>
#define CHANNEL_A       P3_0
#define CHANNEL_B       P3_1
#define CHANNEL_C       P3_2
#define START           P3_4
#define ALE             P3_3
unsigned int KQADC;
unsigned char j,i;
unsigned char TRAM,CHUC,DONVI;
const unsigned char HANG1[16]={" DH SP KY THUAT "};
const unsigned char HANG2[16]={"NHIET DO DO:   "};
void GIAIMA_ASCII ()
{   DONVI=KQADC%10;       KQADC=KQADC/10;
    CHUC=KQADC%10;        TRAM=KQADC/10;
    DONVI=DONVI+0x30;     CHUC=CHUC+0x30;
    TRAM=TRAM+0x30;
    if (TRAM==0X30)
        { TRAM=' ';
            if (CHUC==0X30)          CHUC=' '; }
```

```
}
void HIENTHI_LCD()
{     COMMAND_WRITE(0xCC);     DATA_WRITE(TRAM);
      DATA_WRITE(CHUC);        DATA_WRITE('.');
DATA_WRITE(DONVI);
}
void CHUYEN_DOI_ADC()
{     START = 1;     ALE = 1;        START = 0;     ALE = 0;}

void main ()
{     SETUP_LCD( );
      COMMAND_WRITE(addr_line1);      delay(10);
      for( j=0;j<16;j++)      {DATA_WRITE(HANG1[j]);}
      COMMAND_WRITE(addr_line2);      delay(10);
      for( j=0;j<16;j++)      {DATA_WRITE(HANG2[j]);}

      CHANNEL_A = 1;     CHANNEL_B = 1;     CHANNEL_C = 1;
      while(1)
       {    KQADC =0;
            for (i=0;i<100;i++)
            {     CHUYEN_DOI_ADC();      delay(100);
                  KQADC = KQADC+P0;      }
                  KQADC = KQADC/10;
                  GIAIMA_ASCII ();    HIENTHI_LCD ();
            }
       }
```

Bài 8-3: Dùng vi điều khiển AT89S52 giao tiếp với ADC 0809, 2 cảm biến nhiệt LM35 nối với kênh IN7 và IN6, kết quả đo nhiệt độ hiển thị trên LCD, hàng 1 hiển thị nhiệt độ kênh IN6, hàng 2 hiển thị nhiệt độ kênh IN7.

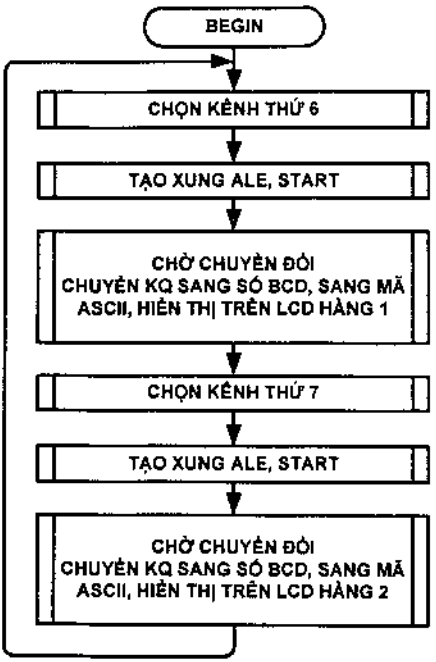
➢ ***Sơ đồ mạch:***

Mạch được thiết kế như hình 8-8:



Hình 8-8: *Mạch giao tiếp vi điều khiển AT89S52 với ADC 0809, 2 LM35, LCD.*

➢ *Lưu đồ chuyển đổi:*



Hình 8-9: *Lưu đồ chuyển đổi của ADC 0809, đo nhiệt độ hai kênh hiển thị LCD.*

❖ ***Chương trình chuyển đổi***

```
#include <AT89X52.h>
#include <THUVIEN_LCD16X2.C>
#define CHANNEL_A       P3_0
#define CHANNEL_B       P3_1
#define CHANNEL_C       P3_2
#define START           P3_4
#define ALE             P3_3
unsigned int KQADC;
unsigned char j,i;
unsigned char TRAM,CHUC,DONVI;
const unsigned char HANG1[16]={"NHIET DO K6:   "};
```

```
const unsigned char HANG2[16]={"NHIET DO K7:    "};
void GIAIMA_ASCII ()
{     DONVI=KQADC%10;        KQADC=KQADC/10;
      CHUC=KQADC%10;         TRAM=KQADC/10;
      DONVI=DONVI+0x30;      CHUC=CHUC+0x30;
      TRAM=TRAM+0x30;
      if (TRAM==0X30)
            { TRAM=' ';
                  if (CHUC==0X30)          CHUC=' '; }
}
void HIENTHI_LCD()
{           DATA_WRITE(TRAM);    DATA_WRITE(CHUC);
            DATA_WRITE('.');     DATA_WRITE(DONVI);    }
void HIENTHI_LCD_K6 ()
{     COMMAND_WRITE(0x8C);       HIENTHI_LCD ();      }
void HIENTHI_LCD_K7 ()
{     COMMAND_WRITE(0xCC);       HIENTHI_LCD ();      }
void CHUYEN_DOI_ADC()
{     START = 1;      ALE = 1;          START = 0;     ALE = 0;}
void main ()
{     SETUP_LCD( );
      COMMAND_WRITE(addr_line1);       delay(10);
      for( j=0;j<16;j++)       {DATA_WRITE(HANG1[j]);}
      COMMAND_WRITE(addr_line2);       delay(10);
      for( j=0;j<16;j++)       {DATA_WRITE(HANG2[j]);}
      CHANNEL_B = 1;     CHANNEL_C = 1;
      while(1)
       {    KQADC =0;      CHANNEL_A = 1;
```

```
        for (i=0;i<100;i++)
        {   CHUYEN_DOI_ADC();      delay(100);
KQADC = KQADC+P0; }
            KQADC = KQADC/10;
            GIAIMA_ASCII();   HIENTHI_LCD_K6 ();
        KQADC =0;      CHANNEL_A = 0;
        for (i=0;i<100;i++)
        {   CHUYEN_DOI_ADC();      delay(100);
KQADC = KQADC+P0; }
            KQADC = KQADC/10;
            GIAIMA_ASCII();   HIENTHI_LCD_K7 ();
        }
    }
```

III. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP

1. Câu hỏi ôn tập

Câu số 8-1: Hãy cho biết chức năng các tín hiệu Start, ALE, EOC của ADC 0809.

Câu số 8-2: Hãy cho biết chức năng của các tín hiệu V_{REF+} và V_{REF-}.

Câu số 8-3: Hãy vẽ mạch giao tiếp vi điều khiển AT89S52 với ADC 0809 và ba led 7 đoạn.

Câu số 8-4: Hãy nêu trình tự chuyển đổi của ADC 0809.

2. Câu hỏi mở rộng

Câu số 8-5: Hãy khảo sát ADC giao tiếp chuẩn I2C PCF 8591.

3. Câu hỏi trắc nghiệm

Câu 8-1: IC 0809 là ADC:

(a) 10 bit (b) 12 bit (c) 16bit (d) 8 bit

Câu 8-2: ADC 0809 có:

(a) 1 kênh (b) 2 kênh (c) 4 kênh (d) 8 kênh

Câu 8-3: ALE của ADC 0809 có chức năng:

(a) Bắt đầu chuyển đổi (b) Chốt dữ liệu

(c) Chốt địa chỉ kênh (d) Báo kết thúc

Câu 8-4: EOC của ADC 0809 có chức năng:

(a) Bắt đầu chuyển đổi (b) Chốt dữ liệu

(c) Chốt địa chỉ kênh (d) Báo kết thúc

Câu 8-5: Tín hiệu nào của ADC 0809 cho phép xuất dữ liệu:

(a) Start (b) OE (c) EOC (d) ALE

Câu 8-6: Thời gian chuyển đổi của ADC 0809 phụ thuộc vào:

(a) Start (b) OE (c) Tần số xung clock (d) ALE

Câu 8-7: Các ngõ vào V_{REF+} và V_{REF-} của ADC 0809 có chức năng:

(a) Thiết lập tốc độ chuyển đổi

(b) Tạo hệ số chuyển đổi step size

(c) Cấp nguồn cho ADC hoạt động

(d) Nhận tín hiệu tương tự cần chuyển đổi

Câu 8-8: Tín hiệu nào của ADC 0809 thực hiện quá trình chuyển đổi:

(a) Start (b) OE (c) EOC (d) ALE

Câu 8-9: ADC 0809 có bao nhiêu tín hiệu chọn kênh:

(a) 1 (b) 2 (c) 3 (d) 8

Câu 8-10: Công thức tính độ phân giải của ADC 0809 là:

(a) $SS = \frac{V_{REF-} - V_{REF+}}{2^8 - 1}$ (b) $SS = \frac{V_{REF-} - V_{REF+}}{2^8}$

(c) $SS = \frac{V_{REF+} - V_{REF-}}{2^8}$ (d) $SS = \frac{V_{REF+} - V_{REF-}}{2^8 - 1}$

Câu 8-11: Công thức tính giá trị nhị phân sau khi chuyển đổi của ADC 0809 là:

(a) $N = \frac{V_{REF+} - V_{REF-}}{2^8 - 1}$ (b) $N = \frac{V_I - V_{REF+}}{V_I - V_{REF-}}$

(c) $N = \frac{V_I - V_{REF-}}{V_{REF+} - V_{REF-}}(2^8 - 1)$ (d) $N = \frac{V_I - V_{REF-}}{V_{REF+} - V_{REF-}}(2^8)$

4. Bài tập

Bài tập 8-1: Mạch đo nhiệt độ một kênh dùng cảm biến LM35, ADC 0809, vi điều khiển AT89S52, ba led 7 đoạn và một relay dùng nguồn 5V.

Hãy vẽ mạch, viết lưu đồ và chương trình đo nhiệt độ hiển thị trên ba led đồng thời điều khiển relay với yêu cầu như sau: khi nhiệt độ nhỏ hơn 40 độ thì relay đóng, lớn hơn hay bằng 40 thì relay ngắt.

Bài tập 8-2: Mạch đo nhiệt độ hai kênh (kênh 0 và kênh 1) dùng hai cảm biến LM35, ADC 0809, vi điều khiển AT89S52, sáu led 7 đoạn và hai relay dùng nguồn 5V.

Hãy vẽ mạch, viết lưu đồ và chương trình đo nhiệt độ hiển thị trên 6 led (mỗi kênh ba led) đồng thời điều khiển relay với yêu cầu như sau: khi nhiệt độ kênh 0 nhỏ hơn 40 độ thì relay0 đóng, lớn hơn hay bằng 40 thì relay0 ngắt, khi nhiệt độ kênh 1 nhỏ hơn 45 độ thì relay1 đóng, lớn hơn hay bằng 45 thì relay1 ngắt.

Bài tập 8-3: Mạch đo nhiệt độ hai kênh dùng cảm biến LM35, ADC 0809, vi điều khiển AT89S52 và ba led 7 đoạn.

Hãy vẽ mạch, viết lưu đồ và chương trình đo nhiệt độ hai kênh lần lượt hiển thị trên ba led, mỗi kênh đo trong khoảng thời gian 1s, định thời 1 giây chuyển kênh dùng Timer T0.

Bài tập 8-4: Mạch đo nhiệt độ 8 kênh dùng 8 cảm biến LM35, ADC 0809, vi điều khiển AT89S52 và ba led 7 đoạn.

Hãy vẽ mạch, viết lưu đồ và chương trình đo nhiệt độ 8 kênh lần lượt hiển thị trên ba led, mỗi kênh đo trong khoảng thời gian 1s, định thời 1 giây chuyển kênh dùng Timer T0.

Bài tập 8-5: Mạch đo nhiệt độ 8 kênh dùng 8 cảm biến LM35, ADC 0809, vi điều khiển AT89S52 và ba led 7 đoạn, hai nút nhấn UP, DN dùng để chuyển kênh cần chuyển đổi.

Hãy vẽ mạch, viết lưu đồ và chương trình đo nhiệt độ 8 kênh hiển thị trên ba led, mặc nhiên là đo kênh thứ 0, khi nhấn UP thì tăng lên kênh kế, khi đến kênh 7 nếu nhấn UP nữa thì về kênh thứ 0, khi nhấn DN thì giảm, khi giảm về kênh 0 mà nhấn nữa thì chuyển sang kênh 7.

Chương 9
VI ĐIỀU KHIỂN AT89S52: NGẮT

- **GIỚI THIỆU**
- **TỔNG QUAN VỀ NGẮT**
- **NGẮT CỦA VI ĐIỀU KHIỂN ATMEL AT89S52**
  - **CÁC NGUỒN NGẮT CỦA AT89S52**
  - **CÁC THANH GHI NGẮT CỦA AT89S5**
    - ✓ *Thanh ghi IE (Interrupt Enable)*
    - ✓ *Thanh ghi IP (Interrupt Priority)*
    - ✓ *Xử lý ngắt*
    - ✓ *Các vector ngắt (Interrupt Vectors)*
  - KHAI BÁO NGẮT CỦA AT89S5hai TRONG LẬP TRÌNH KEIL-C
  - ỨNG DỤNG NGẮT CỦA AT89S52
- **CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP**
  - CÂU HỎI ÔN TẬP
  - CÂU HỎI MỞ RỘNG
  - CÂU HỎI TRẮC NGHIỆM
  - BÀI TẬP

I. GIỚI THIỆU

Ngắt có nhiều tiện ích trong điều khiển nên hầu hết các vi điều khiển đều tích hợp, ở chương này chúng ta sẽ khảo sát nguyên lý hoạt động của ngắt, các nguồn ngắt, vector địa chỉ ngắt, viết chương trình con phục vụ ngắt cho các ứng dụng.

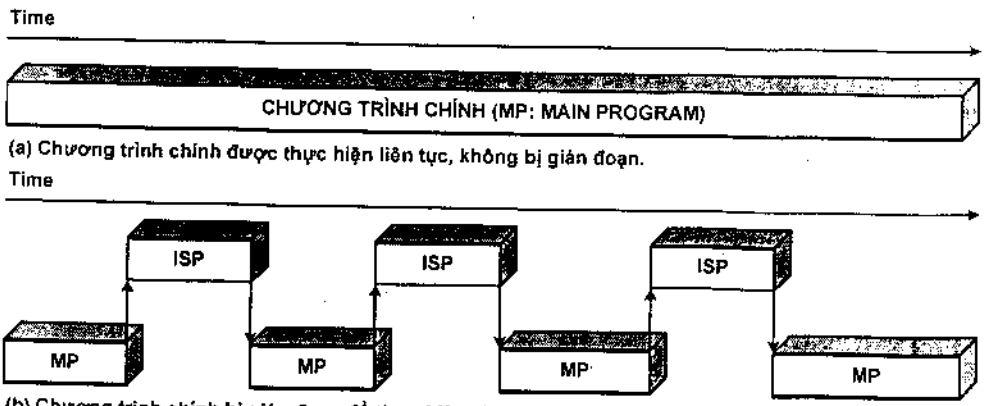
Sau khi kết thúc chương này, bạn có thể sử dụng được ngắt của các vi điều khiển.

II. TỔNG QUAN VỀ NGẮT

Ngắt sử dụng trong vi xử lý hay vi điều khiển hoạt động như sau: vi xử lý hay vi điều khiển luôn thực hiện một chương trình thường gọi là chương trình chính, khi có tác động từ bên ngoài bằng phần cứng hay tác động bên trong làm cho vi xử lý ngừng thực hiện chương trình chính để thực hiện một chương trình khác (còn gọi là chương trình phục vụ ngắt ISR) và sau khi thực hiện xong vi xử lý trở lại thực hiện tiếp chương trình chính. Quá trình làm gián đoạn vi xử lý thực hiện chương trình chính xem như là ngắt.

Có nhiều tác động làm ngừng chương trình chính gọi là các nguồn ngắt, ví dụ khi timer/counter đếm tràn sẽ phát sinh yêu cầu ngắt.

Ngắt đóng một vai trò quan trọng trong lập trình điều khiển, vi xử lý hay vi điều khiển sử dụng ngắt để đáp ứng nhiều sự kiện quan trọng khác trong mà vẫn đảm bảo thực hiện được chương trình chính.



Hình 9-1: Vi điều khiển thực hiện chương trình chính trong hai trường hợp không và có ngắt.

Ví dụ trong khi vi điều khiển đang thực hiện chương trình chính thì có dữ liệu từ hệ thống khác gởi đến, khi đó vi điều khiển ngừng chương trình

chính để thực hiện chương trình phục vụ ngắt nhận dữ liệu xong rồi trở lại tiếp tục thực hiện chương trình chính, hoặc có một tín hiệu báo ngắt từ bên ngoài thì vi điều khiển sẽ ngừng thực hiện chương trình chính để thực hiện chương trình ngắt rồi tiếp tục thực hiện chương trình chính.

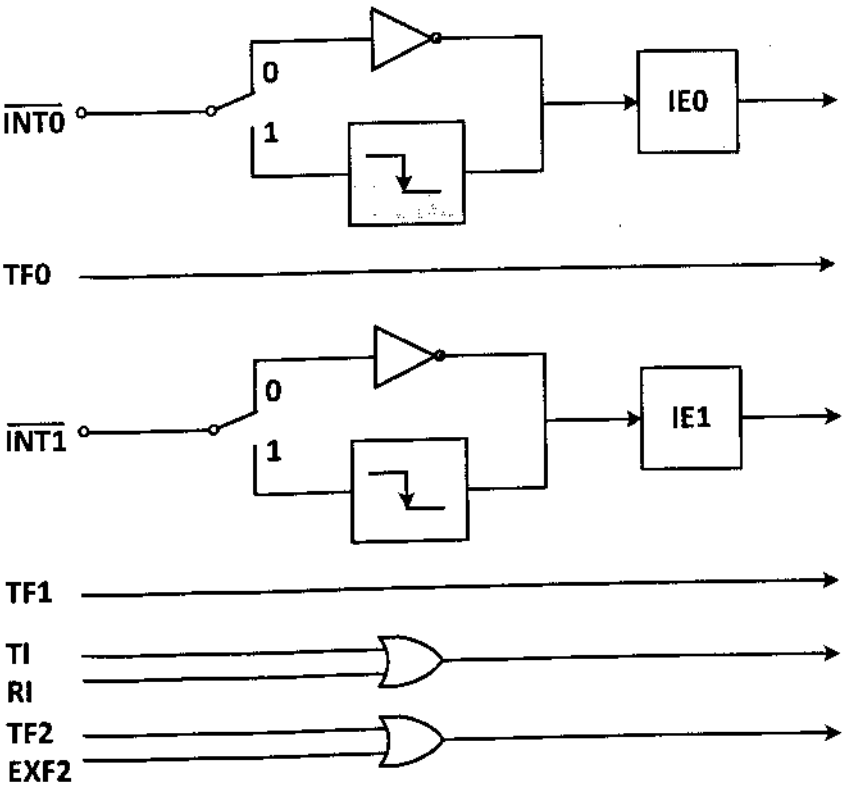
Có thể sử dụng ngắt để yêu cầu vi điều khiển thực hiện nhiều chương trình cùng một lúc có nghĩa là các chương trình được thực hiện xoay vòng.

CPU thực hiện chương trình trong trường hợp có ngắt và không có ngắt như hình 9-1.

III. NGẮT CỦA VI ĐIỀU KHIỂN ATMEL AT89S52

1. Các nguồn ngắt của AT89S52

Vi điều khiển AT89S52 có 6 nguồn ngắt: hai ngắt ngoài, ba ngắt Timer và một ngắt truyền dữ liệu nối tiếp, các nguồn ngắt như hình 9-2. Mặc nhiên khi vi điều khiển bị reset thì tất cả các ngắt đều bị cấm và được cho phép bởi phần mềm.



Hình 9-2: Các nguồn ngắt của vi điều khiển AT89S52.

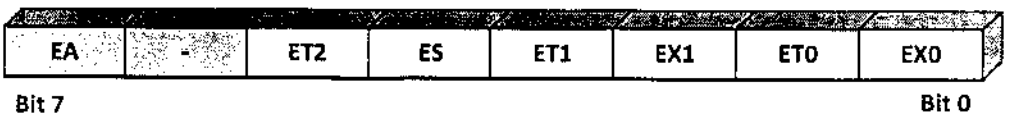
Trong trường hợp có hai hoặc nhiều nguồn ngắt tác động đồng thời hoặc vi điều khiển đang phục vụ ngắt thì xuất hiện một ngắt khác, khi đó sẽ có hai cách giải quyết là: kiểm tra liên tiếp và sử dụng chế độ ưu tiên.

2. Các thanh ghi ngắt của AT89S52

Trong vi điều khiển AT89S52 có hai thanh ghi phục vụ cho ngắt là IE và IP.

➢ ***Thanh ghi IE (Interrupt Enable)***

Có chức năng cho phép hay không cho phép đối với từng nguồn ngắt và cho toàn bộ các nguồn ngắt. Tổ chức của thanh ghi như hình sau:



Hình 9-3: *Thanh ghi IE.*

Bảng 9-1: *Tóm tắt chức năng các bit trong thanh ghi cho phép ngắt IE:*

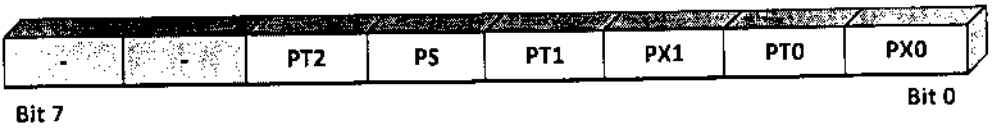
Bit	Kí hiệu	Chức năng (Enable = 1; Disable = 0)
IE.7	EA	Cho phép hoặc cấm toàn bộ các nguồn ngắt.
IE.6	-	Chưa dùng đến
IE.5	ET2	Cho phép ngắt Timer 2 (8052).
IE.4	ES	Cho phép ngắt Port nối tiếp.
IE.3	ET1	Cho phép ngắt Timer 1.
IE.2	EX1	Cho phép ngắt ngoài External 1 (INT1).
IE.1	ET0	Cho phép ngắt Timer 0.
IE.0	EX0	Cho phép ngắt ngoài External 0 (INT0).

Trong thanh ghi IE có bit IE.6 chưa dùng đến, bit IE.7 là bit cho phép/cấm ngắt toàn bộ các nguồn ngắt. Khi bit IE.7= 0 thì cấm hết tất cả các nguồn ngắt, khi bit IE.7=1 thì cho phép tất cả các nguồn ngắt nhưng còn phụ thuộc vào từng bit điều khiển ngắt của từng nguồn ngắt.

Khi có nhiều nguồn ngắt tác động cùng lúc, ngắt nào quan trọng cần thực hiện trước và ngắt nào không quan trọng thì thực hiện sau giống như các công việc mà ta giải quyết hằng ngày. Ngắt cũng được thiết kế có sự sắp xếp thứ tự ưu tiên từ thấp đến cao để người lập trình sắp xếp các nguồn ngắt theo yêu cầu công việc mà mình xử lý.

➢ ***Thanh ghi IP (Interrupt Priority)***

Có chức năng thiết lập chế độ ưu tiên cho các nguồn ngắt. Tổ chức của thanh ghi như sau:



Hình 9-4: Thanh ghi IP.

Bảng 9-2: Tóm tắt chức năng các bit trong thanh ghi ưu tiên ngắt IP:

Bit	Kí hiệu	Chức năng
IP.7	-	Chưa sử dụng
IP.6	-	Chưa sử dụng
IP.5	PT2	Ưu tiên cho ngắt Timer 2.
IP.4	PS	Ưu tiên cho ngắt Port nối tiếp.
IP.3	PT1	Ưu tiên cho ngắt Timer 1.
IP.2	PX1	Ưu tiên cho ngắt ngoài External 1.
IP.1	PT0	Ưu tiên cho ngắt Timer 0.
IP.0	PX0	Ưu tiên cho ngắt ngoài External 0.

Khi reset vi điều khiển thì thanh ghi IP bị xóa và tất cả các ngắt ở mức ưu tiên thấp nhất.

Trong AT89S52 có hai mức ưu tiên thấp và hai mức ưu tiên cao.

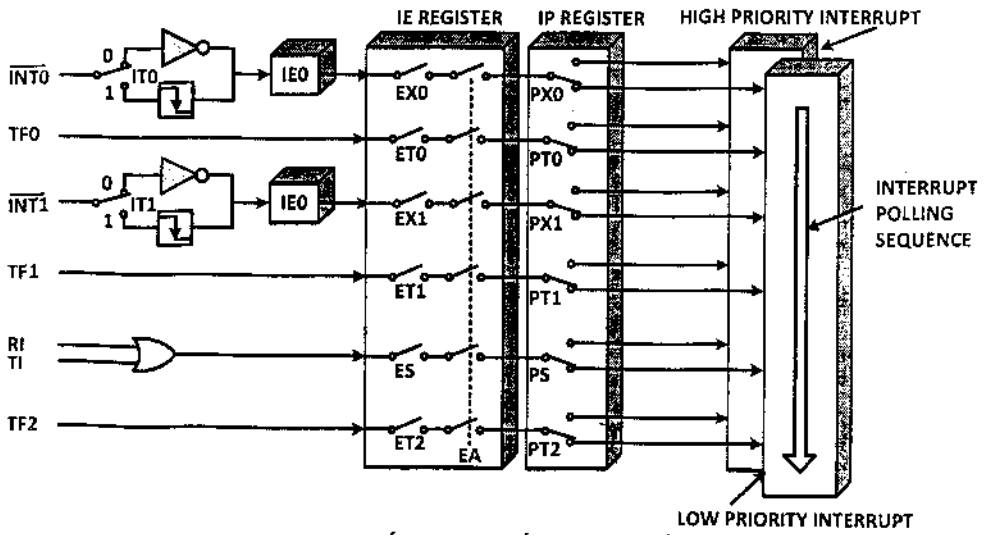
Nếu vi điều khiển đang thực hiện ngắt có mức ưu tiên thấp và nếu có một yêu cầu ngắt mức ưu tiên cao hơn xuất hiện thì vi điều khiển ngừng thực hiện ngắt có mức ưu tiên thấp để thực hiện ngắt có ưu tiên cao hơn.

Ngược lại nếu vi điều khiển đang thực hiện ngắt có mức ưu tiên cao và có yêu cầu ngắt ưu tiên thấp xuất hiện thì vi điều khiển vẫn tiếp tục thực hiện ngắt ưu tiên cao cho đến khi thực hiện xong rồi mới thực hiện ngắt có ưu tiên thấp đang yêu cầu.

Chương trình chính mà vi điều khiển luôn thực hiện thì ở mức thấp nhất, luôn luôn bị ngắt bất chấp ngắt ưu tiên cao hay thấp. Nếu có hai yêu cầu ngắt với các ưu tiên khác nhau xuất hiện đồng thời thì yêu cầu ngắt có mức ưu tiên cao hơn sẽ được phục vụ trước.

Nếu hai yêu cầu ngắt có cùng mức ưu tiên xuất hiện đồng thời thì vòng quét kiểm tra : sẽ xác định yêu cầu ngắt nào sẽ được phục vụ trước.

Vòng quét kiểm tra theo thứ tự ưu tiên từ trên xuống: ngắt ngoài thứ 0 (INT0), ngắt timer T0, ngắt ngoài thứ nhất (INT1), ngắt Timer 1, ngắt truyền dữ liệu nối tiếp (Serial Port), ngắt timer 2. Hình 9-5 sẽ minh họa cho trình tự trên.



***Hình 9-5:** Cấu trúc ngắt của vi điều khiển.*

Hình 9-5 có bảy nguồn ngắt và các bit của thanh ghi IE hoạt động như một contact (On/Off) còn thanh ghi IP hoạt động như một contact chuyển mạch giữa hai vị trí để lựa chọn một trong hai mức độ ưu tiên.

Bit cho phép ngắt toàn cục (Global Enable) nếu cho phép sẽ đóng toàn bộ các contact và tùy thuộc vào bit cho phép của từng nguồn ngắt: nếu cho phép thì đóng mạch và tín hiệu yêu cầu ngắt sẽ đưa vào bên trong để xử lý, nếu cấm thì contact hở mạch nên tín hiệu yêu cầu ngắt sẽ không được xử lý.

Tín hiệu sau khi ra khỏi thanh ghi IE thì đưa đến thanh ghi IP để sắp xếp ưu tiên cho các nguồn ngắt.

Có hai mức độ ưu tiên: mức ưu tiên cao và mức ưu tiên thấp.

Nếu ngắt có ưu tiên cao thì contact chuyển mạch sẽ đưa yêu cầu ngắt đó đến vòng kiểm tra có ưu tiên cao, nếu ngắt có ưu tiên thấp thì contact chuyển mạch sẽ đưa yêu cầu ngắt đó đến vòng kiểm tra có ưu tiên thấp.

Vòng kiểm tra ngắt ưu tiên cao được thực hiện trước và sẽ kiểm tra theo thứ tự từ trên xuống và khi gặp yêu cầu ngắt nào thì yêu cầu ngắt đó sẽ được thực hiện. Sau đó tiếp tục thực hiện cho vòng kiểm tra có mức ưu tiên thấp.

Ngắt truyền dữ liệu nối tiếp tạo ra từ tổ hợp OR của hai cờ báo nhận RI và cờ báo phát TI. Khi ngắt truyền dữ liệu xảy ra và muốn biết là do cờ nhận hay cờ phát tạo ra ngắt để thực hiện hai công việc khác nhau thì phải kiểm tra cờ RI và TI để biết thực hiện công việc nào tương ứng.

Ví dụ trong truyền dữ liệu: khi có báo ngắt truyền dữ liệu thì ta phải kiểm tra xem cờ RI = 1 hay không? Nếu đúng thì hệ thống khác đang gởi dữ liệu đến và phải chuyển hướng chương trình phục vụ ngắt sang hướng nhận dữ liệu, nếu không phải thì chắc chắn là cờ TI=1 báo cho chúng ta biết rằng dữ liệu đã truyền đi xong và sẵn sàng truyền kí tự tiếp theo và khi đó ta phải chuyển hướng chương trình phục vụ ngắt sang phát dữ liệu tiếp theo.

Tương tự, các yêu cầu ngắt của Timer2 tạo ra từ tổ hợp OR của cờ tràn TF2 và cờ nhận biết tín hiệu ngoài EXF2.

Bảng 9-3: *Các cờ của các nguồn ngắt:*

Interrupt	Flag	SFR Register and Bit Position
External 0	IE0	TCON 1
External 1	IE1	TCON 3
Timer 1	TF1	TCON 7
Timer 0	TF0	TCON 5
Serial Port	TI	SCON 1
Serial Port	RI	SCON 0
Timer 2	TF2	T2CON 7
Timer 2	EXF2	T2CON 6

➢ ***Xử lý ngắt***

Khi tín hiệu yêu cầu ngắt xuất hiện và được chấp nhận thì CPU thực hiện các công việc sau:

- Nếu CPU đang thực hiện lệnh thì phải chờ thực hiện xong lệnh đang thực hiện.
- Giá trị của bộ đếm chương trình PC được cất vào bộ nhớ ngăn xếp.
- Trạng thái ngắt hiện hành được lưu vào bên trong.
- Các yêu cầu ngắt khác sẽ bị ngăn lại.

- Địa chỉ chương trình phục vụ ngắt được nạp vào bộ đếm chương trình PC.
- Bắt đầu thực hiện chương trình phục vụ ngắt.

Trong chương trình phục vụ ngắt luôn kết thúc bằng lệnh ***RETI.*** Khi gặp lệnh RETI thì CPU sẽ lấy lại địa chỉ của lệnh tiếp theo trong ngăn xếp trả lại cho thanh ghi PC để tiếp tục thực hiện các công việc tiếp theo của chương trình chính.

➢ ***Các vector ngắt (Interrupt Vectors)***

Khi một yêu cầu ngắt xảy ra, sau khi cất địa chỉ trong PC vào ngăn xếp, địa chỉ của chương trình con phục vụ ngắt còn gọi là vector địa chỉ ngắt sẽ được nạp vào thanh ghi PC, địa chỉ này là cố định và do nhà chế tạo vi điều khiển thiết kế. Các chương trình ngắt phải viết đúng tại vector địa chỉ ngắt.

Bảng 9-4: *Các vector địa chỉ của các nguồn ngắt:*

Interrupt	Flag	Vectors Address
System Reset	RST	0000H
External0	IE0	0003H
Timer0	TF0	000BH
External1	IE1	0013H
Timer1	TF1	001BH
Serial Port	RI or TI	0023H
Timer2	TF2 or EXF2	002BH

Vector reset hệ thống bắt đầu tại địa chỉ 0000H: khi reset vi điều khiển thì thanh ghi PC = 0000H và chương trình chính luôn bắt đầu tại địa chỉ này.

Khi sử dụng yêu cầu ngắt nào thì chương trình con phục vụ ngắt phải viết đúng tại địa chỉ tương ứng. Ví dụ sử dụng ngắt timer T0 thì chương trình ngắt bạn phải viết tại địa chỉ 000BH.

Do khoảng vùng nhớ giữa các vector địa chỉ của các nguồn ngắt chỉ có vài ô nhớ ví dụ như vector địa chỉ ngắt INT0 tại 0003H và vector địa chỉ ngắt T0 tại 000BH chỉ cách nhau có 9 ô nhớ. Nếu chương trình phục vụ ngắt INT0 có kích thước lớn hơn 9 byte thì sẽ đụng đến vùng nhớ của ngắt T0. Cách giải quyết là ngay tại địa chỉ 0003H nên viết lệnh nhảy đến một vùng nhớ khác rộng hơn. Còn nếu ngắt T0 và các ngắt khác không sử dụng thì có thể viết chương trình tại đó cũng được.

3. Khai báo ngắt của AT89S52 trong lập trình Keil-C

Khai báo chương trình con phục vụ ngắt trong Keil giống như khai báo chương trình con nhưng thêm phần thứ tự ngắt và khai báo sử dụng thanh ghi cho chương trình con phục vụ ngắt sử dụng, cú pháp như sau:

Void tên_chương_trình_ngắt() **interrupt n using m**

Trong đó "tên_chương_trình_ngắt()" do chúng ta tự đặt và nên đặt đúng với ngắt đang dùng, "**interrupt n**" với n là số thứ tự các nguồn ngắt như bảng 9-5.

***Bảng 9-5:** Thứ tự các nguồn ngắt:*

Thứ tự **Interrupt n**	Mô tả	Địa chỉ
0	External INT 0	0003H
1	Timer/counter 0	000BH
2	External INT 1	0013H
3	Timer/counter 1	001BH
4	SERIAL PORT	0023H
5	Timer/counter 2	002BH

Giá trị n của ngắt dùng để tính toán ra địa chỉ của vector ngắt theo công thức: $8*n+3$.

Khai báo "using m" dùng để sử dụng bank thanh ghi thứ m từ 0 đến 3.

Ví dụ 9-1: Khai báo ngắt timer0 như sau: "void timer0_interrupt() interrupt 1 using 0", hằng số n bằng 1 thì vector địa chỉ là: $8*n+3=8*1+3=11=000BH$, sử dụng bank thanh ghi thứ 0.

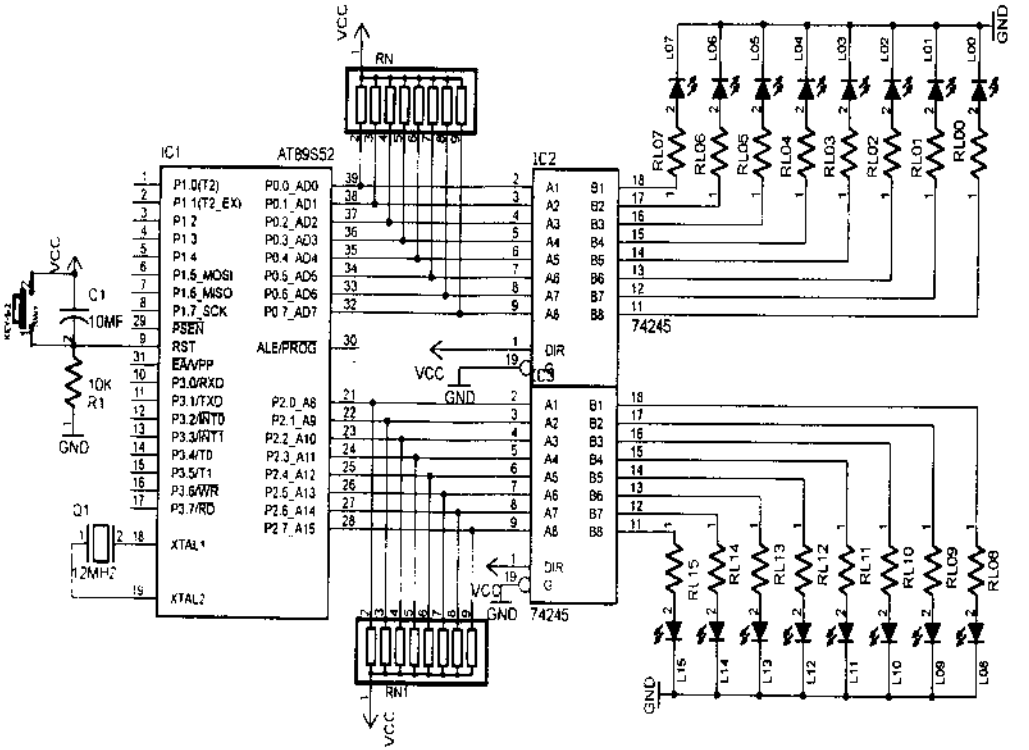
Ví dụ 9-2: Khai báo ngắt timer2 như sau: "void timer2_interrupt() interrupt 5 using 0", hằng số n bằng 5 thì vector địa chỉ là: $8*n+3=8*5+3=43=002BH$, sử dụng bank thanh ghi thứ 0.

4. Ứng dụng ngắt của AT89S52

Phần này trình bày các ứng dụng ngắt đơn giản của AT89S52, qua các ứng dụng này giúp bạn biết viết lưu đồ cho những ứng dụng có dùng ngắt, biết viết chương trình có sử dụng ngắt và đặc biệt là biết thêm cách tính toán thời gian delay cho các timer. Từ các kiến thức cơ bản này sẽ giúp bạn hiểu và viết được các ứng dụng khác.

Bài 9-1: Dùng vi điều khiển AT89S52 thực hiện hai yêu cầu: điều khiển port 2 đếm nhị phân hiển thị bằng led đơn với thời gian trễ tùy ý, dùng timer T0 đếm 65536µs tạo ngắt để điều khiển 8 led đơn nối với port 0 chớp tắt.

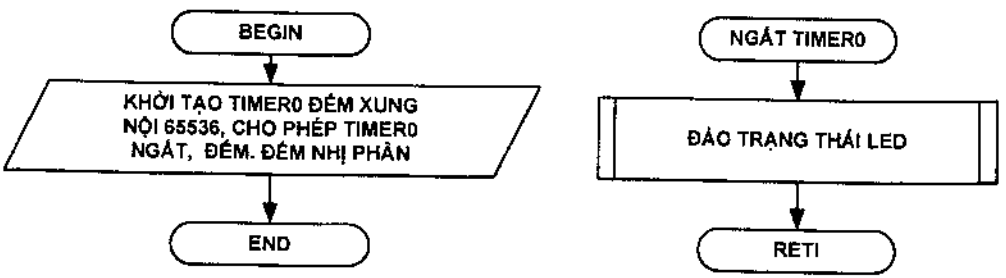
➢ Sơ đồ mạch:



Hình 9-6: *Sơ đồ kết nối điều khiển led đơn qua IC đệm.*

Mạch điện này giống như đã trình bày ở chương sử dụng port.

Lưu đồ: chương trình chính có chức khởi tạo bộ định thời timer đếm xung nội, cho phép ngắt khi bị tràn. Chương trình con phục vụ ngắt khi tràn thì đảo trạng thái của Led.



Hình 9-7. *Lưu đồ điều khiển led đơn dùng ngắt của timer.*

➢ ***Chương trình Keil-C:***

```
#include <AT89X52.h>
void timer0_interrupt() interrupt 1 using 0
{     P0 = ~ P0;  }
void delay (unsigned    int x)
{ unsigned int y;
            for (y=0; y<x; y++)     { }    }
void MAIN ()
{     TMOD = T0_M0_;     TR0 = 1;    EA=1;       ET0=1;
      while(1)    { P2++;     delay(10000);  }
}
```

❖ **Giải thích chương trình:**

Chương trình chính: Lệnh TMOD = T0_M0_ lựa chọn timer T0 hoạt động chế độ 1 đếm xung nội 16 bit. Lệnh TR0 = 1 có chức năng cho timer0 bắt đầu đếm. Lệnh EA = 1 cho phép ngắt toàn cục, lệnh ET0 =1 cho phép timer0 ngắt. Vòng lặp while: điều khiển P2 đếm nhị phân rồi delay.

Khi timer T0 tràn thì phát sinh yêu cầu ngắt, chương trình chính sẽ ngừng để đi thực hiện chương trình con phục vụ ngắt của timer T0.

Chương trình phục vụ ngắt timer0: có chức năng đảo dữ liệu port0 và tự động xóa cờ tràn của timer T0 để báo tràn cho lần sau, khi thực hiện xong thì trở lại chương trình chính.

Khi nào thì ngắt xảy ra để ngừng chương trình chính: Nếu thạch anh sử dụng 12MHz thì sau khi qua bộ chia cho 12 bên trong mạch dao động thì tần số còn lại là 1MHz, chu kỳ cho mỗi xung là 1μs, timer T0 hoạt động đếm 16 bit từ 0 đến 65536 xung tương đương với thời gian là 65536μs.

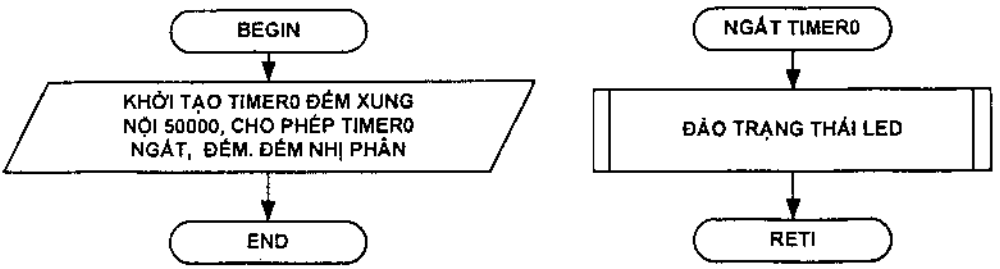
Chu kỳ ngắt là 65536μs.

Thời gian thực hiện chương trình con phục vụ ngắt: rất ngắn vì chỉ thực hiện một lệnh đảo dữ liệu port0 – khoảng vài micro giây rồi trở lại chương trình chính.

Bài 9-2: Dùng vi điều khiển AT89S52 thực hiện hai yêu cầu: điều khiển port 2 đếm nhị phân hiển thị bằng led đơn với thời gian trễ tùy ý, dùng timer T0 đếm 50000μs tạo ngắt để điều khiển 8 led đơn nối với port 0 chớp tắt.

➢ ***Sơ đồ mạch:*** giống bài 9-1.

➢ ***Lưu đồ***: Trong lưu đồ, khởi tạo timer đếm 50000 xung và khi ngắt xảy ra thì phải khởi tạo lại để đếm 50000 xung tiếp theo.



Hình 9-8: *Lưu đồ dùng ngắt timer delay 50ms.*

➢ ***Chương trình Keil-C***:

```
#include <AT89X52.h>
unsigned int DEM;
void timer0_interrupt() interrupt 1 using 0
{     P0 = ~ P0;
      TL0 =     DEM;     TH0 =     DEM >> 8;
}
void delay (unsigned    int x)
{ unsigned int y;
            for (y=0; y<x; y++)     { }    }
void MAIN ()
{     TMOD = T0_M0_:     TR0 = 1;   EA=1;     ET0=1;
      DEM = 15536;          TL0 = DEM;     TH0 = DEM >> 8;
  while(1)
            {     P2++;          delay(10000); }
}
```

❖ **Giải thích chương trình:**

Chương trình chính: ngoài các khai báo giống bài trước thì thêm phần gán giá cho biến DEM bằng 15536. Con số này ở đâu ra? Do yêu cầu của bài này là delay 50000µs, nên timer sẽ đếm 50000 xung và sau khi đếm

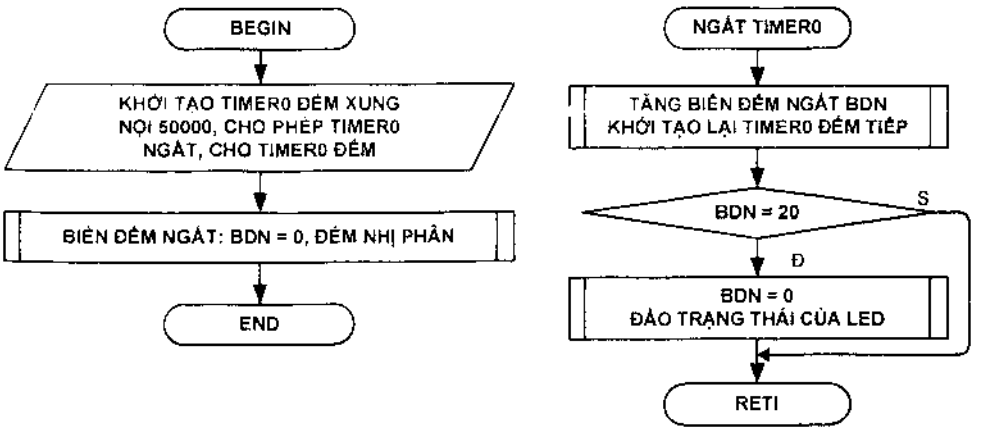
xong phải báo ngắt, nên giá trị xuất phát của timer phải là: 65536 – 50000 = 15536 chính là giá trị khởi tạo của biến DEM và được gán cho hai thanh ghi là TH0 và TL0.

Chương trình phục vụ ngắt timer0: có chức năng khi ngắt xảy ra thì đảo dữ liệu port0, đồng thời khởi tạo lại giá trị 15536 cho hai thanh ghi để bắt đầu chu kỳ đếm 50ms tiếp theo.

Bài 9-3: Dùng vi điều khiển AT89S52 thực hiện hai yêu cầu: điều khiển port 2 đếm nhị phân hiển thị bằng led đơn với thời gian trễ tùy ý, dùng timer T0 đếm 1s tạo ngắt để điều khiển 8 led đơn nối với port 0 chớp tắt.

➢ ***Sơ đồ mạch: giống bài 9-1.***

➢ ***Lưu đồ:***



Hình 9-9: *Lưu đồ dùng ngắt – đếm ngắt delay 1s.*

➢ **Chương trình Keil-C:**

```
#include <AT89X52.h>
unsigned int DEM;
int BDN;
void timer0_interrupt() interrupt 1 using 0
{     BDN++;    TL0      DEM;      TH0 =      DEM >> 8;
      if (BDN == 20)
            {     BDN =0;   P0 = ~ P0;}
}
void delay (unsigned    int x)
```

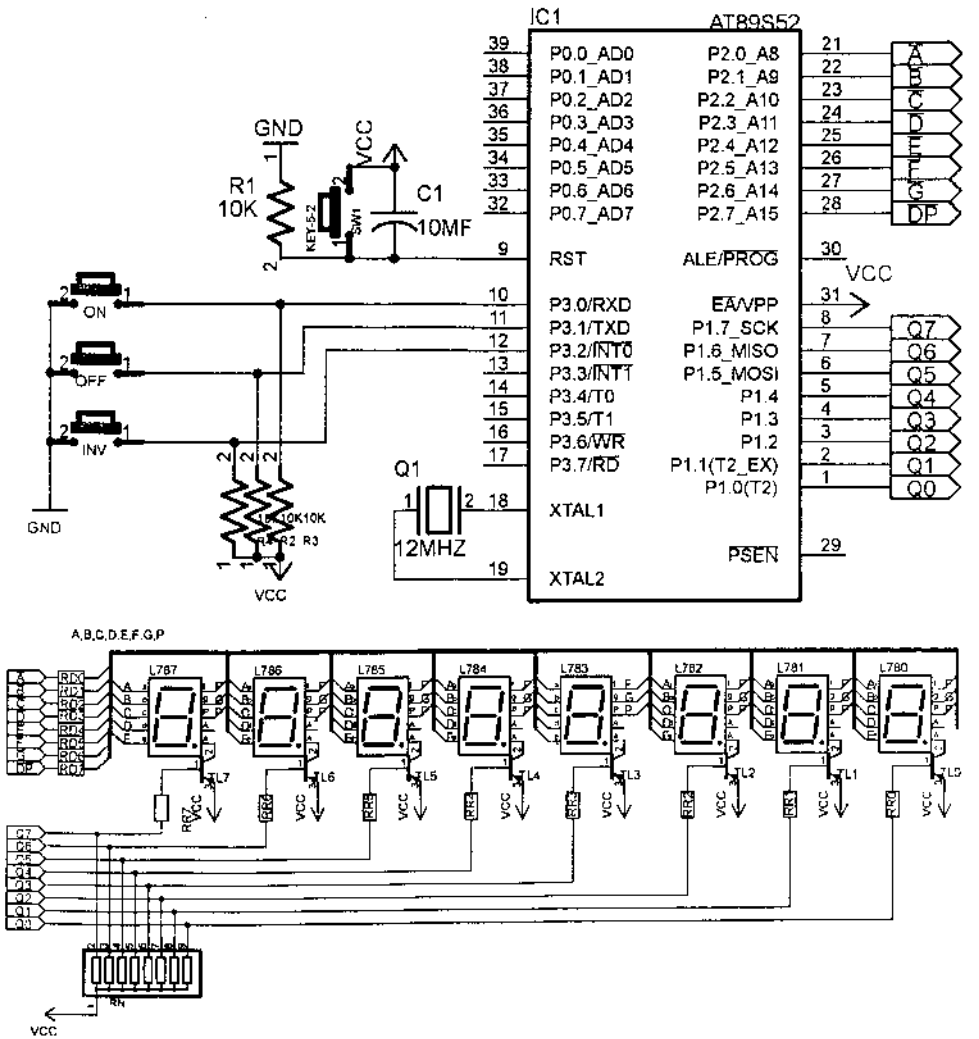
```
{ unsigned int y;
        for (y=0; y<x; y++)     { }    }
void MAIN ()
{·    TMOD = T0_M0_;     TR0 = 1;   EA=1;      ET0=1;
      DEM= 15536;   TL0 =DEM;      TH0 =      DEM >> 8;
      BDN = 0;
  while(1)
        { P2++;     delay(10000); }
}
```

❖ **Giải thích chương trình:**

Chương trình phục vụ ngắt timer0: khi ngắt xảy ra sau mỗi thời gian 50ms thì tăng BDN, khởi tạo lại giá trị 15536 cho hai thanh ghi để bắt đầu chu kỳ đếm 50ms tiếp theo, tiếp theo là kiểm tra BDN xem có bằng 20 hay chưa, nếu chưa thì thoát, nếu đúng thì đảo dữ liệu port0. Thời gian sáng bằng thời gian tắt bằng 20 lần của 50ms tương đương 1 giây.

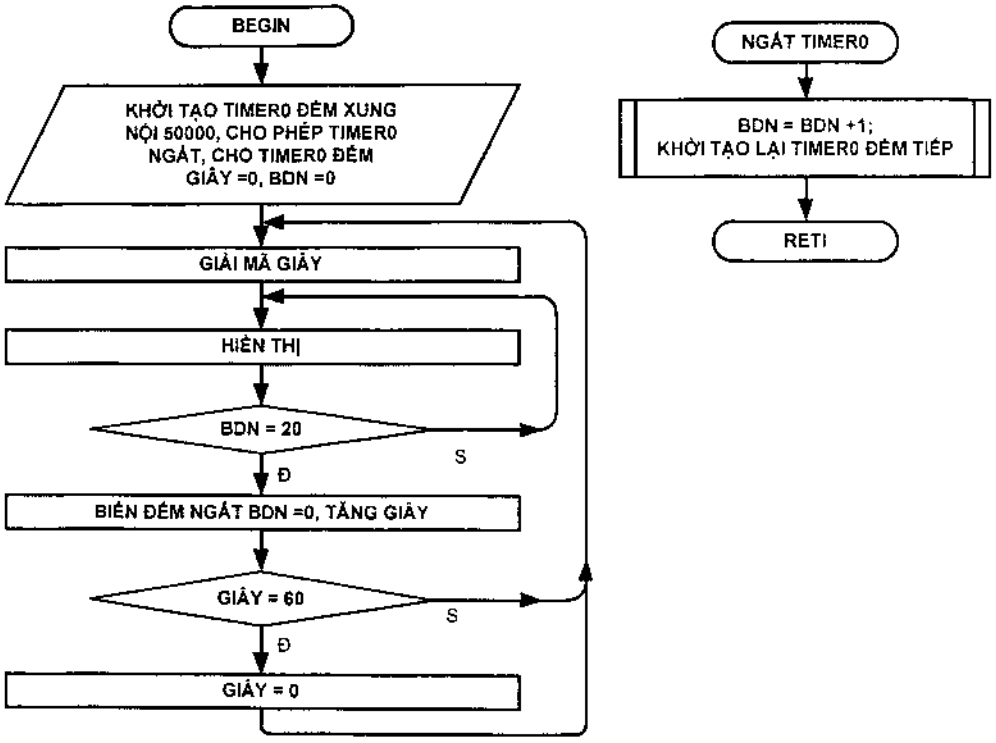
Bài 9-4: Dùng vi điều khiển AT89S52 giao tiếp hai led 7 đoạn quét để đếm giây chính xác dùng định thời timer0 báo ngắt.

Sơ đồ mạch: do yêu cầu mở rộng nên sơ đồ mạch được vẽ cho 8 led 7 đoạn.



Hình 9-10: *Sơ đồ mạch giao tiếp AT89S52 với 8 led 7 đoạn quét.*

➢ ***Lưu đồ:***



Hình 9-11: *Lưu đồ đếm giây – dùng ngắt delay 1s.*

➢ ***Chương trình Keil-C:***

```
#include<AT89X52.H>
unsigned char BDN,GIAY,X,Y;
unsigned char MADONVIS,MACHUCS;
unsigned int DEM;
const unsigned char
MA7D[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
void timer0_interrupt() interrupt 1
{      BDN++;    TL0 = DEM;      TH0 = DEM >> 8;      }

void delay(unsigned int x)
{unsigned int y;
      for(y=0;y<x;y++)            { }
}
```

```
void hienthi ( )
{     P2= MADONVIS;     P1_0 =0;   delay(200);P1_0=1;
      P2= MACHUCS;      P1_1=0;    delay(200);P1_1=1;      }
void giaima()
{     X = GIAY%10;                 Y = GIAY/10;
      MADONVIS=MA7D[X];            MACHUCS=MA7D[Y];
}
void main( )
{     TMOD = T0_M0_;    TR0 = 1;   EA=1;          ET0=1;
      DEM= 15536;       TL0 =DEM;  TH0 =     DEM >> 8;
      BDN = 0;          GIAY = 0;
      while(1)
            {     giaima();
                  if (BDN <20)     hienthi( );
                  else { BDN= BDN-20;
                        GIAY++;
                        if (GIAY==60) GIAY=0;}
            }
}
```

❖ **Giải thích chương trình:**

Chương trình phục vụ ngắt timer0: khi ngắt xảy ra sau mỗi thời gian 50ms thì tăng BDN, khởi tạo lại giá trị 15536 cho hai thanh ghi để bắt đầu chu kỳ đếm 50ms tiếp theo.

Chương trình chính sau khi tiến hành các lệnh khởi tạo thì giải mã và kiểm tra biến đếm ngắt BDN nếu còn nhỏ hơn 20 thì cho hiển thị, nếu bằng hoặc lớn hơn 20 thì trừ BDN cho 20, tiến hành tăng giây và xử lý giây.

Bài 9-5: Dùng vi điều khiển AT89S52 giao tiếp bốn led 7 đoạn quét để đếm phút giây chính xác dùng định thời timer0 báo ngắt.

- ***Sơ đồ mạch***: giống bài 9-4.
- ***Lưu đồ:*** từ lưu đồ đếm giây ta có thể suy ra lưu đồ đếm phút giây.
- **Chương trình Keil-C:**

```
#include<AT89X52.H>
unsigned char BDN,PHUT,GIAY,X,Y;
unsigned char MADONVIS,MACHUCS,MADONVIM,MACHUCM;
unsigned int DEM;
const unsigned char
MA7D[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
void timer0_interrupt() interrupt 1
{     BDN++;    TL0 = DEM;      TH0 = DEM >> 8;      }
void delay(unsigned int x)
{unsigned int y;
      for(y=0;y<x;y++)            {}       }
void hienthi ( )
{     P2= MADONVIS;      P1_0 =0;   delay(200);P1_0=1;
      P2= MACHUCS;       P1_1=0;    delay(200);P1_1=1;
      P2= MADONVIM;      P1_3 =0;   delay(200);P1_3=1;
      P2= MACHUCM;       P1_4=0;    delay(200);P1_4=1;    }
void giaima()
{     X = GIAY%10;                  Y = GIAY/10;
      MADONVIS=MA7D[X];             MACHUCS=MA7D[Y];
      X = PHUT%10;                  Y = PHUT/10;
      MADONVIM=MA7D[X];             MACHUCM=MA7D[Y];}
void main( )
{     TMOD = T0_M0_;     TR0 = 1;   EA=1;      ET0=1;
      DEM= 15536;        TL0 =DEM;      TH0 =      DEM >> 8;
      BDN = 0;           PHUT = 0; GIAY = 0;
      while(1)
       {    giaima();
            if (BDN <20)     hienthi( );
            else { BDN= BDN-20;
                        GIAY++;
                        if (GIAY==60)
```

```
                                {    GIAY=0;   PHUT++;
                                     if (PHUT==60) PHUT=0;
}    }          }                    }
```

Bài 9-6: Dùng vi điều khiển AT89S52 giao tiếp 8 led 7 đoạn quét để đếm giờ phút giây chính xác dùng định thời timer0 báo ngắt.

- ***Sơ đồ mạch***: giống bài 9-4.
- ***Lưu đồ***: tương tự ta có thể suy ra lưu đồ cho giờ phút giây.
- **Chương trình Keil-C:**

```
#include<AT89X52.H>
unsigned char BDN,GIO,PHUT,GIAY,X,Y;
unsigned char
MADONVIS,MACHUCS,MADONVIM,MACHUCM,MADONVIH,MAC
HUCH;
unsigned int DEM;
const unsigned char
MA7D[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};

void timer0_interrupt() interrupt 1
{    BDN++;   TL0 = DEM;    TH0 = DEM >> 8;    }
void delay(unsigned int x)
{unsigned int y;
   for(y=0;y<x;y++)      {}      }
void hienthi ( )
{    P2= MADONVIS;     P1_0 =0;   delay(100);P1_0=1;
     P2= MACHUCS;      P1_1=0;    delay(100);P1_1=1;
     P2= MADONVIM;     P1_3 =0;   delay(100);P1_3=1;
     P2= MACHUCM;      P1_4=0;    delay(100);P1_4=1;
     P2= MADONVIH;     P1_6 =0;   delay(100);P1_6=1;
     P2= MACHUCH;      P1_7=0;    delay(100);P1_7=1;    }

void giaima()
```

```
{     X = GIAY%10;                    Y = GIAY/10;
      MADONVIS=MA7D[X];               MACHUCS=MA7D[Y];
      X = PHUT%10;                    Y = PHUT/10;
      MADONVIM=MA7D[X];               MACHUCM=MA7D[Y];
      X = GIO%10;                     Y = GIO/10;
      MADONVIH=MA7D[X];               MACHUCH=MA7D[Y];}
void main( )
{     TMOD = T0_M0_;    TR0 = 1;  EA=1;     ET0=1;
      DEM= 15536;       TL0 =DEM;     TH0 =     DEM >> 8;
      BDN = 0;  GIO=0;  PHUT = 0; GIAY = 0;
      while(1)
       {    giaima();
            if (BDN <20)     hienthi( );
            else { BDN= BDN-20;
                        GIAY++;
                        if (GIAY==60)
                              {     GIAY=0;   PHUT++;
                                    if (PHUT==60)
                                          {     PHUT=0;  GIO++;
                                                if (GIO==24)     GIO=0;
      }
}     }           }                 }
```

Bài 9-7: Dùng vi điều khiển AT89S52 giao tiếp 8 led 7 đoạn quét để đếm giờ phút giây chính xác dùng định thời timer0 báo ngắt, có ba nút nhấn MOD, UP, DOWN dùng để chỉnh thời gian giờ phút giây.

Nhấn MOD lần thứ nhất thì cho phép chỉnh giây: nhấn UP thì tăng giây, nhấn DOWN thì giảm giây.

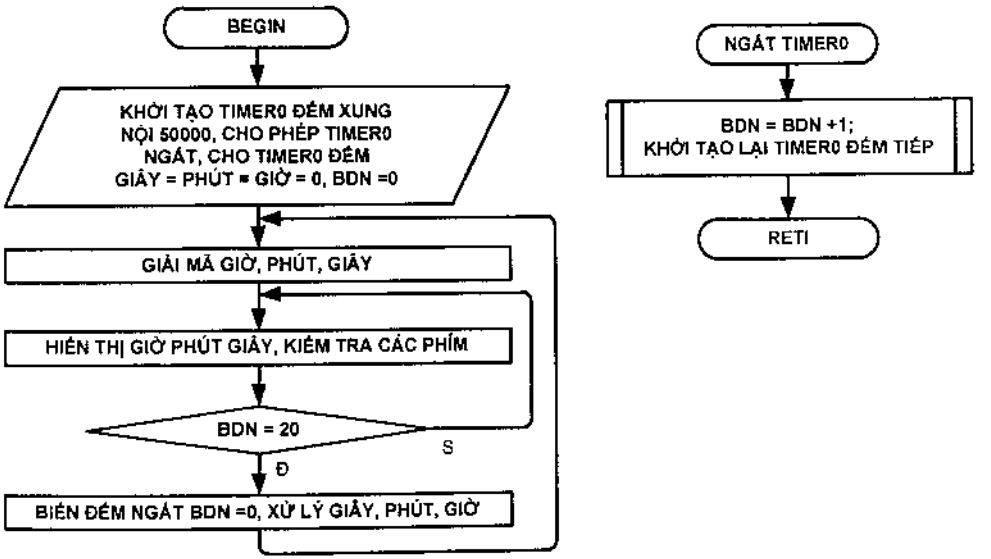
Nhấn MOD thêm lần nữa (lần 2) thì chuyển sang chỉnh phút.

Nhấn MOD thêm lần nữa (lần 3) thì chuyển sang chỉnh giờ.

Nhấn MOD thêm lần nữa (lần 4) thì chuyển về trạng thái không cho chỉnh thời gian.

➢ ***Sơ đồ mạch: giống bài 9-4.***

➢ *Lưu đồ:*



Hình 9-14: *Lưu đồ đếm giờ phút giây – dùng ngắt delay 1s, có phím chỉnh thời gian.*

➢ ***Chương trình Keil-C:***

```
#include<AT89X52.H>
#define MOD             P3_0
#define UP        P3_1
#define DOWN   P3_2
signed char GIO,PHUT,GIAY;
unsigned char
BDN,X,Y,BMOD,DAUCHAMS,DAUCHAMM,DAUCHAMH;
unsigned char
MADONVIS,MACHUCS,MADONVIM,MACHUCM,MADONVIH,MAC
HUCH;
unsigned int DEM;
const unsigned char
MA7D[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
void timer0_interrupt() interrupt 1
{      BDN++;     TL0 = DEM;       TH0 = DEM >> 8;       }
```

```
void delay(unsigned int x)
{unsigned int y;
   for(y=0;y<x;y++)                {      }}
void hienthi ( )
{     P2= MADONVIS;          P1_0 =0;   delay(100); P1_0=1;
      P2= MACHUCS;           P1_1=0;    delay(100); P1_1=1;
      P2= MADONVIM;          P1_3 =0;   delay(100); P1_3=1;
      P2= MACHUCM;           P1_4=0;    delay(100); P1_4=1;
      P2= MADONVIH;          P1_6 =0;   delay(100); P1_6=1;
      P2= MACHUCH;           P1_7=0;    delay(100); P1_7=1;    }
void delay_HIENTHI(unsigned int x)
{unsigned int y;
   for(y=0;y<x;y++)          {hienthi ( );}}

void giaima()
{     X = GIAY%10;                    Y = GIAY/10;
      MADONVIS=MA7D[X];               MACHUCS=MA7D[Y];
      X = PHUT%10;                    Y = PHUT/10;
      MADONVIM=MA7D[X];               MACHUCM=MA7D[Y];
      X = GIO%10;                     Y = GIO/10;
      MADONVIH=MA7D[X];               MACHUCH=MA7D[Y];
      MADONVIS   =   MADONVIS   &   DAUCHAMS;
      MADONVIM   =   MADONVIM   &   DAUCHAMM;
      MADONVIH   =   MADONVIH   &   DAUCHAMH;
      }
void CHONGDOI_MOD ()
{     if (MOD == 0)
            {     delay_HIENTHI(20);
                  if (MOD == 0)
                        {     BMOD++;
                              if (BMOD==4)        BMOD =0;
```

```
if(BMOD==0)   {     DAUCHAMS = 0XFF;  DAUCHAMM = 0XFF;
                    DAUCHAMH = 0XFF;}
if(BMOD==1)   {     DAUCHAMS = 0X7F;  DAUCHAMM = 0XFF;
                    DAUCHAMH = 0XFF;}

if(BMOD==2)   {     DAUCHAMS = 0XFF;  DAUCHAMM = 0X7F;
                    DAUCHAMH = 0XFF;}
if(BMOD==3)   {     DAUCHAMS = 0XFF;  DAUCHAMM = 0XFF;
                    DAUCHAMH = 0X7F;}
                        giaima();
                        do   {     hienthi( );   }
                        while (MOD == 0);
}          }          }

void CHONGDOI_UP ()
{    if (UP == 0)
          {    delay_HIENTHI (20);
               if (UP == 0)
{    switch (BMOD)
     {    case 1:    GIAY++;
                     if (GIAY==60)  GIAY=0;  break;
          case 2:    PHUT++;
                     if (PHUT==60)  PHUT=0;  break;
          case 3:    GIO++;
               if (GIO==24)    GIO=0;    break;
          default:   break;
     }
                     giaima();
                     do   {     hienthi( );   }
                     while (UP == 0);
}          }          }
```

```
void CHONGDOI_DOWN ()
{     if (DOWN == 0)
            {     delay_HIENTHI (20);
                  if (DOWN == 0)
      {     switch (BMOD)
            {     case 1:     GIAY--;    if (GIAY==-1)
                              GIAY=59; break;
                  case 2:     PHUT--;    if (PHUT==-1)
                              PHUT=59; break;
                  case 3:     GIO--;     if (GIO==-1)
                              GIO=23;    break;
                  default:    break;
            }
                        giaima();
                        do    {     hienthi( );   }
                        while (DOWN == 0);
}}}

void main( )
{     TMOD = T0_M0_;     TR0 = 1;   EA=1;      ET0=1;
      DEM= 15536;        TL0 =DEM;             TH0 =      DEM >> 8;

      BDN = 0;  GIO=0;      PHUT = 0; GIAY = 0; BMOD=0;
      DAUCHAMS = 0XFF;  DAUCHAMM = 0XFF;
      DAUCHAMH = 0XFF;
      while(1)
       {    giaima();
            if (BDN <20)
                  {     hienthi( );  CHONGDOI_DOWN ();
                        CHONGDOI_UP ();  CHONGDOI_MOD ();  }
            else { BDN= BDN-20;
```

```
                    GIAY++;
                    if (GIAY==60)
                        {   GIAY=0;   PHUT++;
                            if (PHUT==60)
                                {   PHUT=0;  GIO++;
                                    if (GIO==24)    GIO=0;
    }
}   }       }               }
```

❖ **Giải thích chương trình:**

Chương trình chính sau khi tiến hành các lệnh khởi tạo thì giải mã và kiểm tra biến đếm ngắt BDN nếu còn nhỏ hơn 20 thì cho hiển thị và kiểm tra xem có nhấn các phím hay không, nếu không nhấn thì quay lại làm tiếp.

Khi biến đếm ngắt bằng 20 thì tiến hành xử lý giây, phút và giờ bình thường.

Nếu có nhấn phím MOD: thì tiến hành chống dội và tăng giá trị biến mod BMOD lên một đơn vị, đồng thời xử lý cho hiển thị dấu chấm ở led hàng đơn vị giây để cho biết đang cho phép chỉnh giây, hiển thị dấu chấm ở led hàng đơn vị phút để cho biết đang cho phép chỉnh phút, hiển thị dấu chấm ở led hàng đơn vị giờ để cho biết đang cho phép chỉnh giờ, nếu không cho phép chỉnh thì dấu chấm tắt.

Nếu có nhấn phím UP: thì tiến hành chống dội và căn cứ vào giá trị của biến mod BMOD để xử lý tăng giây hoặc phút hoặc giờ. Trong khi chờ nhả phím, tiến hành hiển thị để led không bị tắt.

Nếu có nhấn phím DOWN: thì tiến hành chống dội và căn cứ vào giá trị của biến mod BMOD để xử lý giảm giây hoặc phút hoặc giờ. Trong khi chờ nhả phím, tiến hành hiển thị để led không bị tắt.

Khi giá trị biến BMOD bằng 0 thì không cho phép chỉnh thời gian và tắt dấu chấm.

Khi biến BMOD bằng 1 thì cho dấu chấm đơn vị giây sáng, tắt đơn vị phút và đơn vị giờ.

Khi biến BMOD bằng 2 thì cho dấu chấm đơn vị phút sáng, tắt đơn vị giây và đơn vị giờ.

Khi biến BMOD bằng 3 thì cho dấu chấm đơn vị giờ sáng, tắt đơn vị giây và đơn vị phút.

Bài 9-10: Dùng vi điều khiển AT89S52 giao tiếp LCD để đếm giờ phút giây chính xác dùng định thời timer0 báo ngắt, có ba nút nhấn MOD, UP, DOWN dùng để chỉnh thời gian giờ phút giây.

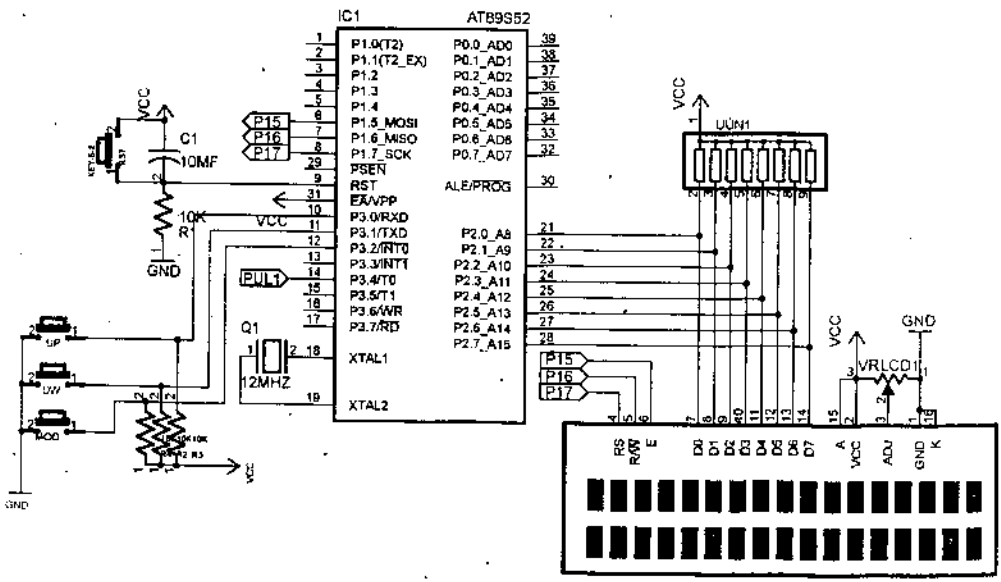
Nhấn MOD lần thứ nhất thì cho phép chỉnh giây: nhấn UP thì tăng giây, nhấn DOWN thì giảm giây.

Nhấn MOD thêm lần nữa (lần 2) thì chuyển sang chỉnh phút.

Nhấn MOD thêm lần nữa (lần 3) thì chuyển sang chỉnh giờ.

Nhấn MOD thêm lần nữa (lần 4) thì chuyển về trạng thái không cho chỉnh thời gian.

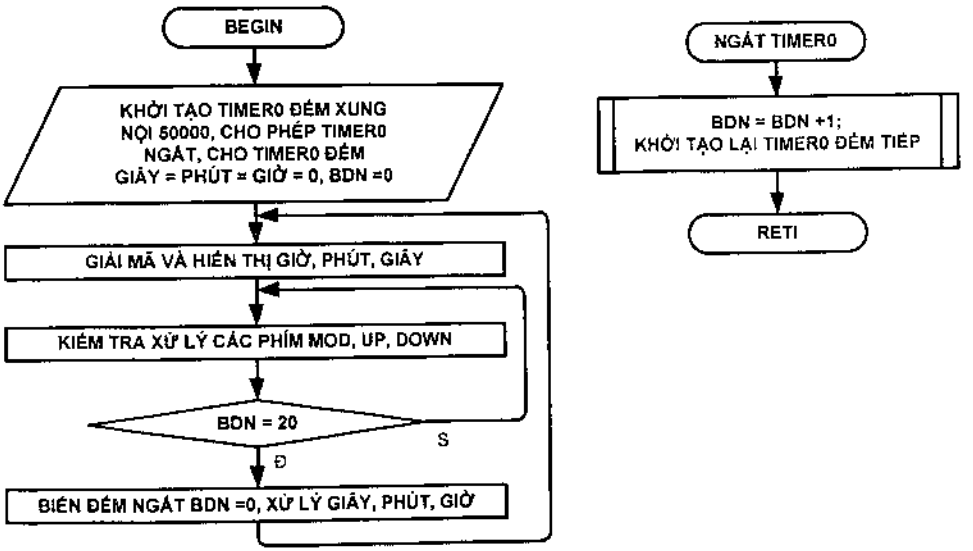
➢ ***Sơ đồ mạch:***



Hình 9-15: *Sơ đồ mạch đồng hồ hiển thị LCD có ba phím để chỉnh thời gian.*

➢ ***Lưu đồ:***

Lưu đồ của đồng hồ hiển thị trên LCD khác với hiển thị trên led 7 đoạn dùng phương pháp quét là không cần phải hiển thị quét liên tục, ở led 7 đoạn thì phải quét liên tục để led sáng, nếu ngừng quét thì led sẽ tắt. Do không cần quét nên chương trình viết cho LCD sẽ đơn giản hơn nhiều so với Led.



Hình 9-16: *Lưu đồ đếm giờ phút giây – dùng ngắt delay 1s, có phím chỉnh thời gian.*

➢ ***Chương trình Keil-C:***

```
#include<AT89X52.H>
#include<THUVIEN_LCD16X2.C>
#define MOD          P1_0
#define UP       P1_1
#define DOWN  P1_2
const unsigned char HANG1[16]={"CLOCK:        "};
const unsigned char HANG2[16]={"* DHSPKT TPHCM *"};

signed char GIO,PHUT,GIAY,j;
unsigned char BDN,X,Y,BMOD;
unsigned char
MADONVIS,MACHUCS,MADONVIM,MACHUCM,MADONVIH,MACHUCH;
unsigned int DEM;

void timer0_interrupt() interrupt 1
{    BDN++;    TL0 = DEM;      TH0 = DEM >> 8;      }
```

```
void hienthi ( )
{     COMMAND_WRITE (0x88);
      DATA_WRITE (MACHUCH);          DATA_WRITE
(MADONVIH);
      DATA_WRITE (' ');
      DATA_WRITE (MACHUCM);   DATA_WRITE (MADONVIM);
      DATA_WRITE (' ');
      DATA_WRITE (MACHUCS);          DATA_WRITE
(MADONVIS);
}
void CHOPTAT ( )
{     if (BMOD ==1)          {     COMMAND_WRITE (0x8E);
      DATA_WRITE (' ');
      DATA_WRITE (' ');              COMMAND_WRITE (0x90);
                                     }
      else if (BMOD ==2)     {     COMMAND_WRITE (0x8B);
      DATA_WRITE (' ');
      DATA_WRITE (' ');              COMMAND_WRITE (0x90);
                             }
      else if (BMOD ==3)     {     COMMAND_WRITE (0x88);
DATA_WRITE (' ');
      DATA_WRITE (' ');              COMMAND_WRITE (0x90);
                                     }
}
void giaima()
{     X = GIAY%10;                   Y = GIAY/10;
      MADONVIS=X +0X30;              MACHUCS=Y +0X30;
      X = PHUT%10;                   Y = PHUT/10;
      MADONVIM=X +0X30;              MACHUCM=Y +0X30;
      X = GIO%10;                    Y = GIO/10;
      MADONVIH=X +0X30;              MACHUCH=Y +0X30;
```

```
    }
void CHONGDOI_MOD ()
{   if (MOD == 0)
        {   delay(10000);
            if (MOD == 0)
                {   BMOD++; COMMAND_WRITE (0x0F);
                    if (BMOD==4) BMOD =0;
                    do  {   }
                    while (MOD == 0);
}       }       }

void CHONGDOI_UP ()
{   if (UP == 0)
        {  delay(10000);
           if (UP == 0)
{  switch (BMOD)
    {   case 1:   GIAY++;  if (GIAY==60) GIAY=0;   break;
        case 2:   PHUT++;  if (PHUT==60) PHUT=0;   break;
        case 3:   GIO++;   if (GIO==24)   GIO=0;   break;
        default:  break;
    }
                  giaima();  hienthi( );
                  do  {   }
                  while (UP == 0);
}}}

void CHONGDOI_DOWN ()
{   if (DOWN == 0)
        {  delay(10000);
           if (DOWN == 0)
               {   switch (BMOD)
```

```
{     case 1:     GIAY--;   if (GIAY==-1)   GIAY=59; break;
      case 2:     PHUT--;   if (PHUT==-1)   PHUT=59; break;
      case 3:     GIO--;    if (GIO==-1)    GIO=23;   break;
      default:    break;
      }
                        giaima();          hienthi( );
                        do   {      }
                        while (DOWN == 0);
}}}

void main( )
{     TMOD = T0_M0_;     TR0 = 1;   EA=1;      ET0=1;
      DEM= 15536;        TL0 =DEM;      TH0 =      DEM >> 8;
      BDN = 0;  GIO=0;   PHUT = 0; GIAY = 0; BMOD=0;
      SETUP_LCD( );
            COMMAND_WRITE(addr_line1);       delay(10);
            for( j=0;j<12;j++)       {DATA_WRITE(HANG1[j]);}
            COMMAND_WRITE(addr_line2);       delay(10);
            for( j=0;j<16;j++)       {DATA_WRITE(HANG2[j]);}
      while(1)
       {
            if (BDN <20)
                  {     CHONGDOI_DOWN (); CHONGDOI_UP ();
                        CHONGDOI_MOD ();
                        if (BDN==10)    {CHOPTAT();}
                  }
            else { BDN= BDN-20;
                  GIAY++;
                  if (GIAY==60)
                        {     GIAY=0;          PHUT++;
                              if (PHUT==60)
```

```
                                {   PHUT=0;  GIO++;
                                    if (GIO==24)    GIO=0;    }}
                            giaima();  hienthi( );
    }           }               }
```

❖ **Giải thích chương trình:**

Chương trình chính sau khi tiến hành các lệnh khởi tạo thì giải mã và kiểm tra biến đếm ngắt BDN. Nếu còn nhỏ hơn 20 thì cho kiểm tra xem có nhấn các phím hay không và kiểm tra xem biến đếm ngắt BDN. Nếu bằng 10 (tương đương 0,5 giây) thì đi kiểm tra biến BMOD nếu đang ở trạng thái không cho chỉnh thời gian thì thoát, nếu có thì tiến hành kiểm tra đang chỉnh giây hoặc phút hoặc giờ thì tiến hành hiển thị khoảng trắng để tắt và dời con trỏ để tạo hiệu ứng nhấp nháy cho biết đối tượng đang hiệu chỉnh.

Khi biến đếm ngắt bằng 20 thì tiến hành xử lý giây, phút và giờ bình thường.

Nếu có nhấn phím MOD: thì tiến hành chống dội và tăng giá trị biến mod BMOD lên 1 đơn vị.

Nếu có nhấn phím UP: thì tiến hành chống dội và căn cứ vào giá trị của biến mod BMOD để xử lý tăng giây hoặc phút hoặc giờ.

Nếu có nhấn phím DOWN: thì tiến hành chống dội và căn cứ vào giá trị của biến mod BMOD để xử lý giảm giây hoặc phút hoặc giờ.

IV. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP

1. Câu hỏi ôn tập

Câu số 9-1: Hãy cho biết tên các thanh ghi liên quan đến ngắt của vi điều khiển AT89S52.

Câu số 9-2: Hãy cho biết chức năng của các bit trong thanh ghi IE và IP của vi điều khiển AT89S52.

Câu số 9-3: Hãy cho biết các vector địa chỉ ngắt của vi điều khiển AT89S52.

2. Câu hỏi mở rộng

Câu 9-4: Hãy so sánh ngắt của vi điều khiển AT89S52 với PIC16F887.

3. Câu hỏi trắc nghiệm

Câu 9-1: Trong AT89S52 thì các thanh ghi có liên quan đến ngắt:

(a) TH0, TL0 (b) TCON, SCON

(c) IE, IP (d) TH1, TL1

Câu 9-2: Trong AT89S52 thì có bao nhiêu nguồn ngắt:

(a) 4 (b) 5

(c) 6 (d) 7

Câu 9-3: Bit nào của AT89S52 thì cho phép ngắt toàn cục:

(a) ET0 (b) ET1

(c) EA (d) EX0

Câu 9-4: Bit nào của AT89S52 thì cho phép ngắt ngoài thứ 0 và thứ 1:

(a) ET0, EX1 (b) ET1, EX0

(c) ET0, ET1 (d) EX0, EX1

Câu 9-5: Bit nào của AT89S52 thì cho phép ngắt timer0 và timer1:

(a) ET0, EX1 (b) ET1, EX0

(c) ET0, ET1 (d) EX0, EX1

Câu 9-6: Khi nào thì ngắt timer0 của AT89S52 xảy ra:

(a) Khi ngõ vào INT1 xuống mức 0

(b) Khi có dữ liệu đến

(c) Khi timer đếm tràn bị tràn

(d) Khi phát xong dữ liệu

Câu 9-7: Khi nào thì ngắt truyền dữ liệu của AT89S52 xảy ra:

(a) Khi ngõ vào INT1 xuống mức 0

(b) Khi mất nguồn

(c) Khi đếm tràn

(d) Khi phát xong dữ liệu

Câu 9-8: Vector địa chỉ ngắt truyền dữ liệu của AT89S52 là:

(a) 0013H (b) 0003H

(c) 000BH (d) 0023H

Câu 9-9: Vector địa chỉ ngắt timer2 của AT89S52 là:

(a) 0013H (b) 0003H

(c) 002BH (d) 0023H

Câu 9-10: Vector địa chỉ ngắt ngoài thứ nhất của AT89S52 là:

(a) 0013H (b) 0003H

(c) 002BH (d) 0023H

Câu 9-11: Bit nào của AT89S52 thì cho phép ngắt truyền dữ liệu:

(a) ET0 (b) ET1

(c) ES (d) EX0

Câu 9-12: Điều kiện để chương trình con ngắt timer0 của AT89S52 được thực hiện là:

(a) ET0 = 0 và ET1 = 1 (b) ET0 = 1 và EA= 0

(c) ET0 = 1 và EA= 1 (d) ET0 = 0 và EA= 1

Câu 9-13: Điều kiện để chương trình con ngắt INT0, INT1 của AT89S52 được thực hiện là:

(a) EX0 = 0 và EX1 = 1 (b) EX0 = 1, EX1 = 1 và EA= 0

(c) EX0 = 1 và EA= 1 (d) EX0 = 1, EX1 = 1 và EA= 1

4. Bài tập

Bài tập 9-1: Một vi điều khiển AT89S52 giao tiếp với năm led 7 đoạn loại anode chung. Hãy viết lưu đồ và chương trình đếm giây chính xác hiển thị trên hai led sử dụng timer1 định thời báo ngắt và đếm sản phẩm hiển thị trên các led còn lại giới hạn đếm từ 000 đến 999, dùng T0.

Bài tập 9-2: Một vi điều khiển AT89S52 giao tiếp với 8 led 7 đoạn loại anode chung dùng phương pháp quét. Hãy viết lưu đồ và chương trình đếm phút giây chính xác sử dụng timer1 định thời báo ngắt và đếm sản phẩm hiển thị trên các led còn lại giới hạn đếm từ 000 đến 999, dùng T0.

Bài tập 9-3: Một vi điều khiển AT89S52 giao tiếp với 9 led 7 đoạn loại anode chung dùng phương pháp quét. Hãy viết lưu đồ và chương trình đếm giờ phút giây chính xác sử dụng timer1 định thời báo ngắt và đếm sản phẩm hiển thị trên các led còn lại giới hạn đếm từ 000 đến 999, dùng T0.

Bài tập 9-4: Một vi điều khiển AT89S52 giao tiếp với năm led 7 đoạn loại anode chung. Hãy viết lưu đồ và chương trình đếm giây chính xác hiển thị trên hai led sử dụng timer1 định thời báo ngắt và đếm sản phẩm hiển thị trên các led còn lại giới hạn đếm từ 000 đến 999, dùng T0. Khi mạch đếm giây về 00 thì đồng thời xóa luôn giá trị đếm của counter.

Bài tập 9-5: Một vi điều khiển AT89S52 giao tiếp với LCD. Hãy viết lưu đồ và chương trình đếm giờ phút giây chính xác sử dụng timer1 định thời báo ngắt, có thể hẹn giờ để đóng mở thiết bị hiển thị trên LCD ở hàng 2, có ba nút để chỉnh thời gian và cài đặt thời gian hẹn giờ.

Chương 10
VI ĐIỀU KHIỂN AT89S52: PORT XUẤT NHẬP

- ❖ **GIỚI THIỆU**
- ❖ **TỔNG QUAN VỀ CÁC KIỂU TRUYỀN DỮ LIỆU**
- ❖ **TRUYỀN DỮ LIỆU NỐI TIẾP ĐỒNG BỘ VÀ KHÔNG ĐỒNG BỘ**
- ❖ **TRUYỀN DỮ LIỆU NỐI TIẾP CỦA AT89S52**
  - TRUYỀN DỮ LIỆU KHÔNG ĐỒNG BỘ CỦA AT89S52
  - CHỨC NĂNG CÁC THANH GHI TRUYỀN DỮ LIỆU CỦA AT89S5
    - ✓ *Thanh ghi SBUF (serial buffer)*
    - ✓ *Thanh ghi SCON (Serial Control)*
  - **CÁC KIỂU TRUYỀN DỮ LIỆU CỦA AT89S52**
    - ✓ *Truyền dữ liệu kiểu 0 – kiểu thanh ghi dịch*
    - ✓ *Truyền dữ liệu kiểu 1 – thu phát bất đồng bộ tốc độ thay đổi*
    - ✓ *Truyền dữ liệu kiểu 2 – thu phát bất đồng bộ 9 bit tốc độ cố định*
    - ✓ *Truyền dữ liệu kiểu 3 – thu phát bất đồng bộ 9 bit tốc độ thay đổi*
  - **ỨNG DỤNG TRUYỀN DỮ LIỆU UART CỦA AT89S52**
- ❖ **TRUYỀN DỮ LIỆU NỐI TIẾP SPI CỦA AT89S8252** 188
  - TRUYỀN DỮ LIỆU SPI CỦA AT89S8252
  - CHỨC NĂNG CÁC THANH GHI TRUYỀN DỮ LIỆU SPI CỦA AT89S8252
  - DẠNG SÓNG TRUYỀN DỮ LIỆU SPI 190
  - ỨNG DỤNG TRUYỀN DỮ LIỆU SPI CỦA AT89S8252
- ❖ **TRUYỀN DỮ LIỆU NỐI TIẾP I2C**
  - GIỚI THIỆU

- TỔNG QUAN VỀ TRUYỀN DỮ LIỆU I2C
- QUY TRÌNH TRUYỀN DỮ LIỆU CHUẨN I2C
- DẠNG SÓNG TRUYỀN DỮ LIỆU CHUẨN I2C
- KHẢO SÁT REALTIME DS13B07
- DẠNG SÓNG TRUYỀN DỮ LIỆU SPI
- ỨNG DỤNG REALTIME

❖ **CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP**

- CÂU HỎI ÔN TẬP
- CÂU HỎI MỞ RỘNG
- CÂU HỎI TRẮC NGHIỆM
- BÀI TẬP

I. GIỚI THIỆU

Truyền dữ liệu nối tiếp có nhiều tiện ích trong điều khiển nên hầu hết các vi điều khiển đều tích hợp, ở chương này chúng ta sẽ khảo sát nguyên lý hoạt động truyền dữ liệu đồng bộ và không đồng bộ, các viết chương trình truyền dữ liệu cho các ứng dụng.

Sau khi kết thúc chương này, bạn có thể sử dụng được truyền dữ liệu nối tiếp của các vi điều khiển.

II. TỔNG QUAN VỀ CÁC KIỂU TRUYỀN DỮ LIỆU

Có nhiều kiểu truyền dữ liệu phổ biến tích hợp trong các họ vi điều khiển bao gồm:

- Truyền dữ liệu nối tiếp đồng bộ và bất đồng bộ (UART synchronous asynchronous receiver and transmitter).
- Truyền dữ liệu giữa các vi điều khiển với các thiết bị ngoại vi (SPI serial peripheral interface).
- Truyền dữ liệu hai dây (I2C: inter-integrated circuit).

Ở kiểu truyền nối tiếp đồng bộ thì có một đường dữ liệu và một đường xung clock, thiết bị nào cấp xung clock thì thiết bị đóng vai trò là chủ, thiết bị nhận xung clock đóng vai trò là tớ, tốc độ truyền dữ liệu phụ thuộc vào tần số xung clock.

Ở kiểu truyền nối tiếp bất đồng bộ thì có một đường phát dữ liệu và một đường nhận dữ liệu, không còn tín hiệu xung clock nên gọi là bất đồng bộ. Để truyền được dữ liệu thì cả bên phát và bên nhận phải tự tạo xung clock có cùng tần số và thường gọi là tốc độ truyền dữ liệu (baud), ví dụ 2400baud, 4800baud, ..., 2400baud có nghĩa là truyền 2400 bit trên 1 giây.

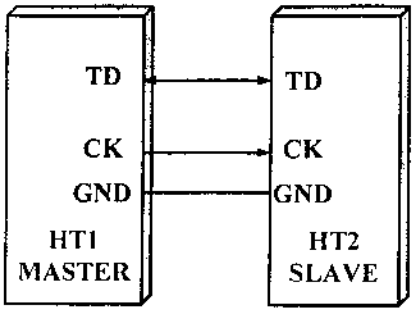
III. TRUYỀN DỮ LIỆU NỐI TIẾP ĐỒNG BỘ VÀ KHÔNG ĐỒNG BỘ

Truyền dữ liệu đồng bộ gồm các đường truyền dữ liệu (DT) và tín hiệu xung clock (CK) – chức năng của CK dùng để dịch chuyển dữ liệu, ***mỗi 1 xung ck là 1 bit dữ liệu được truyền đi.***

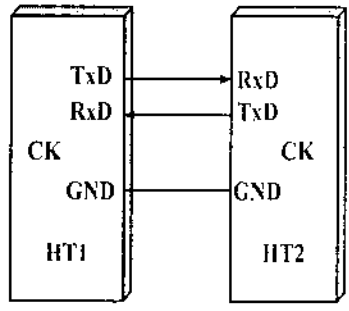
Trong hệ thống truyền dữ liệu đồng bộ, hệ thống nào cung cấp xung CK thì đóng vai trò là master (chủ) – những hệ thống còn lại nhận xung ck đóng vai trò là slave (tớ).

Tốc độ truyền dữ liệu chính là tốc độ của xung ck – chính là tần số xung ck.

Ví dụ tần số xung ck là 1MHz thì tốc độ truyền dữ liệu là 1MBPS – 1M baud.



Hình 10-1: Hệ thống truyền đồng bộ.



Hình 10-2: Hệ thống truyền bất đồng bộ.

Truyền dữ liệu không đồng bộ giống như hệ thống truyền dữ liệu đồng bộ nhưng không có xung CK. Không còn phân biệt chủ và tớ – các hệ thống là ngang cấp.

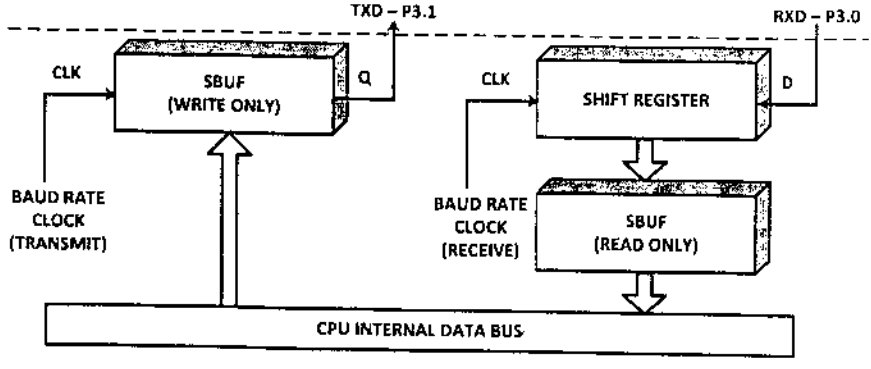
Mỗi 1 xung ck là 1 bit dữ liệu được truyền đi – bây giờ không còn xung Ck thì làm sao để truyền dữ liệu?

Để truyền dữ liệu thì mỗi hệ thống phải có một mạch dao động tạo xung CK – hai hệ thống sẽ có hai mạch dao động độc lập nhưng phải cùng tần số hay cùng tốc độ.

IV. TRUYỀN DỮ LIỆU NỐI TIẾP CỦA AT89S52

1. Truyền dữ liệu không đồng bộ của AT89S52

Trong vi điều khiển AT89S52 có tích hợp truyền dữ liệu không đồng bộ có sơ đồ khối như sau:



Hình 10-3: Sơ đồ khối hệ thống truyền bất đồng bộ.

Dữ liệu cần truyền đi sẽ được lưu vào thanh ghi SBUF, dữ liệu trong thanh ghi này sẽ được xung clk dịch ra ngoài chân TxD (ở chân P3.1). Dữ liệu nhận về dạng nối tiếp ở ngõ vào RxD (ở chân P3.0) sẽ được xung clk dịch vào và chuyển thành song song lưu vào thanh ghi SBUF.

Khi dữ liệu truyền đi thì sẽ làm cờ TI lên 1, khi có dữ liệu nhận về thì cờ RI lên 1.

Có hai thanh ghi phục vụ cho truyền dữ liệu nối tiếp là SCON và SBUF.

2. Chức năng các thanh ghi truyền dữ liệu của AT89S52

Có hai thanh ghi phục vụ cho truyền dữ liệu nối tiếp là SCON và SBUF.

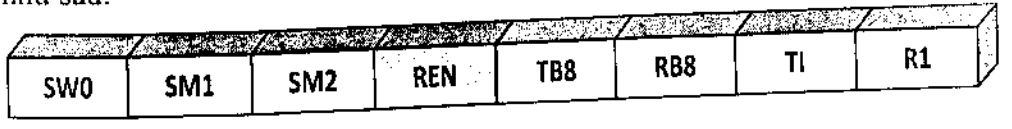
➢ ***Thanh ghi SBUF (serial buffer)***

Có hai thanh ghi sbuf: một cho phát và một cho nhận, thanh ghi phát thì chuyển dữ liệu song song thành nối tiếp để truyền đi. Thanh ghi nhận thì chuyển dữ liệu nhận nối tiếp về và chuyển thành song song.

Trong tập lệnh, khi thực hiện lệnh ghi, vi điều khiển tự động ghi dữ liệu vào thanh ghi phát để phát đi, khi đọc dữ liệu thì vi điều khiển sẽ đọc dữ liệu từ thanh ghi nhận.

➢ ***Thanh ghi SCON (Serial Control)***

Có chức năng thiết lập chế độ truyền dữ liệu. Tổ chức của thanh ghi như sau:



Hình 10-4: *Thanh ghi SCON.*

Bảng 10-1: Tóm tắt chức năng các bit trong thanh ghi SCON:

Bit	Kí hiệu	Chức năng
7	SM0	Bit chọn kiểu truyền nối tiếp: bit thứ 0.
6	SM1	Bit chọn kiểu truyền nối tiếp: bit thứ 1.
5	SM2	Bit cho phép truyền nhiều vi xử lý ở mode 2 và 3; RI sẽ không tích cực nếu bit thứ 9 đã thu vào là 0.

4	REN	REN = 1 sẽ cho phép nhận kí tự.
3	TB8	Dùng để lưu bit 8 để truyền đi khi hoạt động ở mode 2 và 3.
2	RB8	Dùng để lưu bit 8 nhận về khi hoạt động ở mode 2 và 3.
1	TI	Cờ báo hiệu này lên mức 1 khi truyền xong 1 kí tự.
0	RI	Cờ báo hiệu này lên mức 1 khi nhận xong 1 kí tự.

3. Các kiểu truyền dữ liệu của AT89S52

Hai bit SM1, SM0 dùng để thiết lập các kiểu truyền dữ liệu.

Bảng 10-2: Tóm tắt các chế độ truyền dữ liệu:

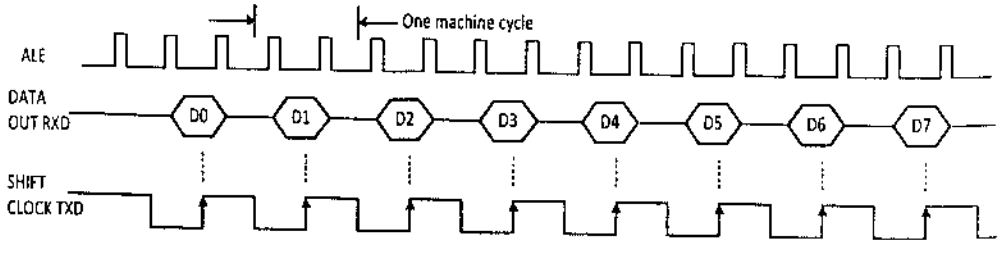
SM0	SM1	Kiểu	Mô tả	Tốc độ baud
0	0	0	Thanh ghi dịch	Cố định bằng tần số dao động f/12.
0	1	1	UART 8 bit	Tốc độ truyền thay đổi bởi Timer.
1	0	2	UART 9 bit	Cố định bằng tần số dao động f/32 or f/644
1	1	3	UART 9 bit	Tốc độ truyền thay đổi bởi Timer.

Khảo sát chi tiết các kiểu truyền dữ liệu.

➢ ***Truyền dữ liệu kiểu 0 – kiểu thanh ghi dịch***

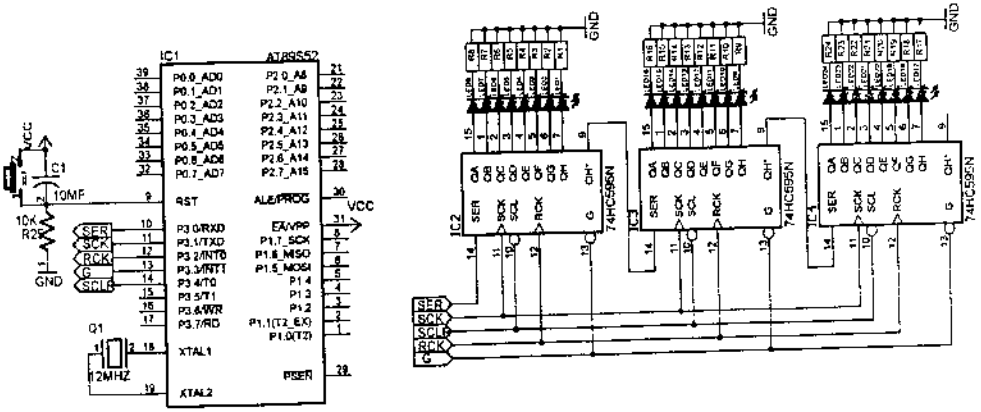
Dữ liệu nối tiếp nhận vào và dữ liệu truyền đi đều thông qua chân RxD. Chân TxD thì dùng để tạo xung clock. 8 bit dữ liệu để truyền đi hoặc nhận về thì luôn bắt đầu với bit có trọng số nhỏ nhất LSB, xem hình 10-5. Mỗi một xung clock thì 1 bit dữ liệu được truyền đi.

Tốc độ Baud được thiết lập cố định ở tần số bằng $1/12$ tần số dao động thạch anh trên chip.



Hình 10-5: *Trình tự truyền dữ liệu ở mod 0.*

Ứng dụng của kiểu truyền này dùng để mở rộng port như hình sau:



***Hình 10-6:** Ứng dụng kiểu truyền mod 0 để mở rộng port.*

Trong sơ đồ mạch như hình 10-6 sử dụng thêm các IC thanh ghi dịch 74595 dịch vào nối tiếp ra song song và nối tiếp. Ra song song để điều khiển, ra nối tiếp để ghép nhiều IC với nhau.

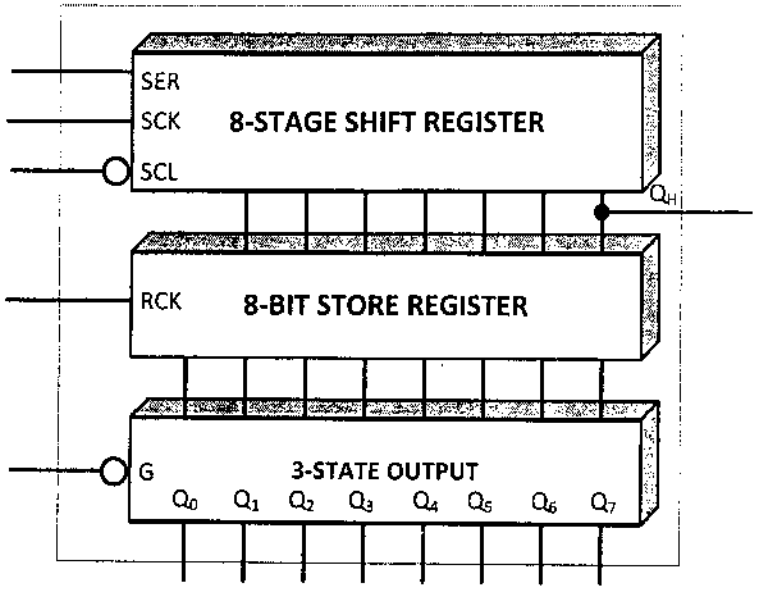
Với ba IC trong mạch ta có thêm 24 đường xuất dữ liệu ra để điều khiển, muốn nhiều hơn ta kết nối thêm IC.

Chân tín hiệu SER của 74595 là ngõ vào nhận dữ liệu nối tiếp sẽ nối với chân xuất nhập dữ liệu của vi điều khiển.

Chân tín hiệu SCK dùng để nhận xung clock để nhịp đẩy dữ liệu vào thanh ghi dịch sẽ nối với chân xuất tín hiệu xung clock của vi điều khiển.

Chân tín hiệu SCLR dùng để xóa dữ liệu trong thanh ghi dịch nối tiếp.

Thanh ghi dịch 74595 có hai bộ thanh ghi 8 bit: một bộ thanh ghi dùng để nhận dữ liệu dịch vào (shift register) và một bộ thanh ghi dịch dùng để lưu dữ liệu xuất ra ngoài (storage register), tín hiệu RCK dùng để nạp dữ liệu từ thanh ghi dịch bên trong sang thanh ghi dịch xuất ra ngoài, xem hình sau:



***Hình 10-7:** Cấu trúc của thanh ghi dịch 74595.*

Chân tín hiệu G dùng để mở bộ đệm ba trạng thái xuất tín hiệu ra ngoài.

Chức năng hai tầng thanh ghi là khi dịch chuyển dữ liệu sẽ không làm ảnh hưởng đến ngõ ra, chờ cho dịch chuyển hết dữ liệu mới xuất dữ liệu ra ngoài.

Chương trình truyền dữ liệu 3 byte ra ba thanh ghi như sau:

```
MOV   SCON,#00010000B    ;SM0SM1=00 – MOD 0; REN = 1
MOV   SBUF,30H           ;GOI BYTE THU 1
JNB   TI,$               ;CHỜ TRUYỀN XONG
CLR   TI                 ;XÓA CỜ
MOV   SBUF,31H           ;GOI BYTE THU 2
JNB   TI,$
CLR   TI
MOV   SBUF,32H           ;GOI BYTE THU 3
JNB   TI,$
  CLR    TI
  JMP    $
  END
```

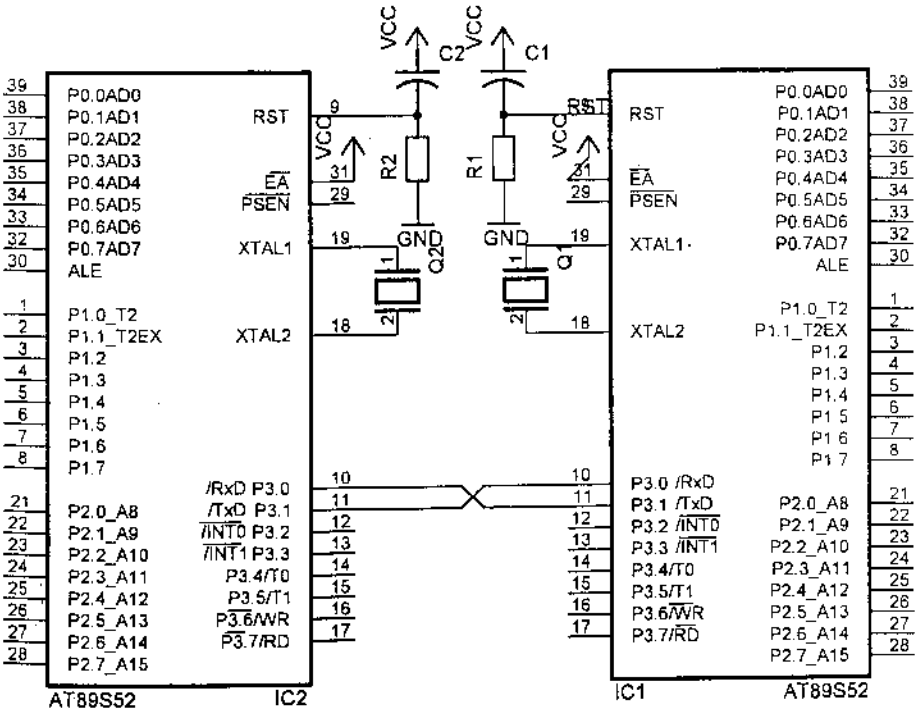
Lệnh thứ nhất khởi tạo chế độ truyền dữ liệu mod 0, ở kiểu này thì tốc độ truyền là cố định nên không cần thiết lập tốc độ.

Lệnh "MOV SBUF,30H" có chức năng gởi nội dung ô nhớ có địa chỉ 30H ra thanh ghi, kiểm tra cờ TI xem bằng 1 hay chưa, nếu chưa thì chờ lên 1, có nghĩa là đã gởi xong, xóa cờ TI và tiếp tục gởi.

➢ ***Truyền dữ liệu kiểu 1 – thu phát bất đồng bộ tốc độ thay đổi***

Truyền dữ liệu nối tiếp bất đồng bộ UART 8 bit có tốc độ Baud thay đổi.

Trong kiểu này, 10 bit dữ liệu sẽ phát đi ở chân TxD và dữ liệu nhận về ở chân RxD, sơ đồ giao tiếp hai vi điều khiển như hình 10-8:



Hình 10-8: *Giao tiếp truyền dữ liệu nối tiếp hai vi điều khiển.*

10 bit gồm có: 1 bit start, 8 bit data (LSB là bit đầu tiên) và 1 bit stop.

Tốc độ Baud được thiết lập bởi tốc độ tràn của ***Timer T1 hoặc timer T2*** hoặc cả hai timer T1 và T2: một timer cho máy phát và một timer cho máy thu.

Thiết lập tốc độ truyền dữ liệu: "tốc độ truyền dữ liệu bằng tốc độ tràn của timer chia cho 32 khi bit SMOD = 0 và chia cho 16 khi bit SMOD= 1".

$$TDTDL = \frac{TDTRAN}{32}\Big|_{SMOD=0} = \frac{TDTRAN}{16}\Big|_{SMOD=1}$$

Ví dụ 10-1: Hãy tính giá trị của timer tràn nếu chọn tốc độ truyền là 9600BPS = 9600BAUD, cho SMOD = 0, sử dụng thạch anh có tần số 12MHz.

$$TDTDL = \frac{TDTRAN}{32}\Big|_{SMOD=0}$$

Tốc độ tràn của timer là:

$$TDTRAN = TDTDL \times 32 = 9600BAUD \times 32 = 307200Hz$$

Tần số thạch anh sử dụng là 12MHz qua bộ chia 12 còn 1MHz, mỗi xung có chu kỳ là 1μs. Để tràn 307,200 lần trong 1 giây thì timer sẽ đếm số xung bằng:

$$SOXUNGDEM = \frac{1000000}{307200} = 3{,}2552$$

Số lượng xung đếm và tràn là 3,2552 xung, giá trị này lẻ sẽ gây sai số, để giảm sai số thì ta tính ngược lại như sau:

$$SOLUONGXUNG = 3 \times 307200 = 921600Hz$$

Tính tần số của tụ thạch anh bằng:

$$TSTA = 921600Hz \times 12 = 11059200Hz = 11{,}0592MHz$$

Vậy nếu sử dụng thạch anh có tần số bằng 11,059MHz thì sai số là 0%. Các nhà chế tạo ra tụ thạch anh này để phục vụ cho truyền dữ liệu không bị sai số.

Bảng 10-3: *Các thông số tốc độ truyền dữ liệu:*

Tốc độ baud	Tần số thạch anh	SMOD	Giá trị nạp cho TH1	Tốc độ thực	Sai số
9600	12MHz	1	- 7 (F9H)	8923	7%
2400	12MHz	0	-13 (F3H)	2404	0,16%
1200	12MHz	0	-26 (E6H)	1202	~0%
19200	11,0592MHz	1	-3 (FDH)	19200	0%
9600	11,0592MHz	0	-3 (FDH)	9600	0%

4800	11,0592MHz	0	-6 (FDH)	4800	0%
2400	11,0592MHz	0	-12 (F4H)	2400	0%
1200	11,0592MHz	0	-24 (E8H)	1200	0%

Trong bảng trên, khi sử dụng thạch anh 12MHz luôn có sai số, khi sử dụng thạch anh 11,059MHz thì sai số luôn là 0 %. Các giá trị nạp vào số âm là sử dụng số nhị phân có dấu.

➢ ***Truyền dữ liệu kiểu 2 – thu phát bất đồng bộ 9 bit tốc độ cố định***

Khi SM1 SM0 = 10 thì truyền dữ liệu hoạt động ở kiểu 2 có tốc độ Baud cố định. Có 11 bit được phát hoặc thu: 1 bit Start, 8 bit data, 1 bit data thứ 9 được lập trình và 1 bit Stop. Khi phát thì bit thứ 8 được đặt vào TB8 của SCON (có thể bit parity). Khi thu thì bit thứ 8 được đặt vào bit RB8 của thanh ghi SCON. Tốc độ Baud trong mode 2 bằng 1/12 hoặc 1/64 tần số dao động trên Chip.

➢ ***Truyền dữ liệu kiểu 3 – thu phát bất đồng bộ 9 bit tốc độ thay đổi***

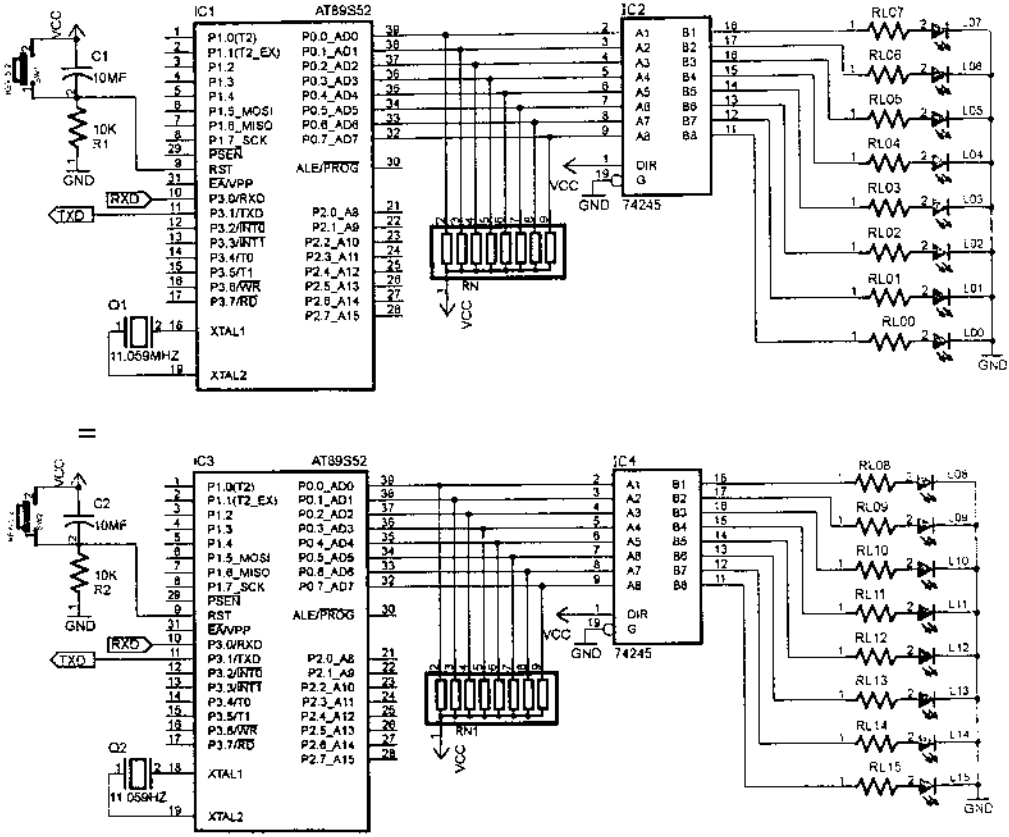
Khi SM1 SM0 = 11 thì truyền dữ liệu hoạt động ở kiểu 3 là kiểu UART 9 bit có tốc độ Baud thay đổi. Kiểu 3 tương tự kiểu 2 ngoại trừ tốc độ Baud được thiết lập bởi Timer. Các kiểu 1, kiểu 2 và kiểu 3 rất giống nhau, những điểm khác nhau là ở tốc độ Baud (kiểu 2 cố định, kiểu 1 và kiểu 3 thay đổi) và số bit dữ liệu (kiểu 1 có 8 bit, kiểu 2 và kiểu 3 có 9 bit data).

4. Ứng dụng truyền dữ liệu của AT89S52

Phần này trình bày các ứng dụng truyền dữ liệu giản của AT89S52, qua các ứng dụng này giúp bạn biết mạch giao tiếp truyền dữ liệu, biết viết chương trình truyền dữ liệu không sử dụng ngắt và có sử dụng ngắt. Từ các kiến thức cơ bản này sẽ giúp bạn hiểu và viết được các ứng dụng khác.

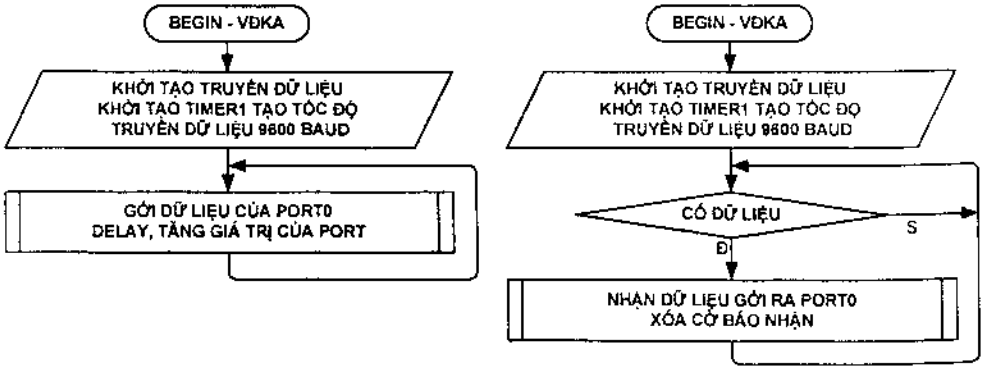
Bài 10-1: Một hệ thống dùng hai vi điều khiển AT89S52 truyền dữ liệu nối tiếp UART. Vi điều khiển A dùng port0 điều khiển 8 led đơn đếm nhị phân đồng thời gởi dữ liệu của port0 sang vi điều khiển B để gởi ra port0. Tốc độ truyền dữ liệu là 9600baud, dùng thạch anh 11,059MHz.

➢ **Sơ đồ mạch:**



Hình 10-9: Sơ đồ giao tiếp hai vi điều khiển để truyền dữ liệu.

➢ ***Lưu đồ:***



Hình 10-10: Lưu đồ điều khiển truyền dữ liệu giữa hai vi điều khiển.

➢ **Chương trình Assembly:**

;chương trình vi điều khiển A

```
            ORG   0000H
            MOV   SCON,#01010000B; MOD1, REN=1, RB8=TB8=0,
TI=RI=0
            MOV   TMOD,#20H          ;20H=0010 0000 =
T1:MODE2
            MOV   TH1,#-3            ;BAUD=9600, 11.0592MHZ
            SETB  TR1                ;T1:RUN
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            MOV   P0,#00             ;P0
MAINA:      MOV   SBUF,P0            ;GOI DL DI
            JNB   TI,$               ;CHO GOI XONG THI TI=1
            CLR   TI
            CALL  DELAY1S
            INC   P0
            JMP   MAINA              ;NHAY VE TRUYEN TIEP
$INCLUDE(TV_DELAY.ASM)
            END
```

Chương trình chính có chức năng khởi truyền dữ liệu mod1, cho phép nhận dữ liệu, xóa các cờ phát và nhận. Thiết lập timer1 hoạt động ở mod 2 để tạo tốc độ tràn phục vụ cho truyền dữ liệu, lệnh khởi tạo giá trị đếm 3 xung là tràn để thiết lập tốc độ 9600baud, lệnh thứ bốn cho phép timer1 hoạt động.

Lệnh "MOV P0,#0" làm port0 bằng 0, lệnh "MOV SBUF,P0" tiến hành gởi dữ liệu đi, lệnh "JNB TI,$" kiểm tra cờ phát TI lên 1 hay chưa, nếu chưa thì chờ, nếu lên 1 thì dữ liệu đã phát xong, tiến hành xóa cờ. Lệnh gọi chương trình con delay để làm trễ rồi tăng port0 lên 1 đơn vị, lệnh nhảy về nhãn "MAINA" để tiếp tục gởi, ...

;chương trình vi điều khiển B

```
            ORG   0000H
            MOV   SCON,#01010000B; MOD1, REN=1, RB8=TB8=0,
TI=RI=0
            MOV   TMOD,#20H          ;20H=0010 0000 = T1:MODE2
            MOV   TH1,#-3            ;BAUD=9600, 11.0592MHZ
```

```
            SETB   TR1              ;T1:RUN
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
MAINB:      JNB    RI,$; CHO NHAN DL, CHO DEN KHI CO DL
            CLR    RI;   XOA DE BAO LAN SAU
            MOV    P0,SBUF; LAY DL TU TG SBUF GOI RA PORT0
            JMP    MAINB; NHAY VE NHAN TIEP
            END
```

Chương trình chính của vi điều khiển B với phần khởi tạo giống như của vi điều khiển A. Sau khi khởi tạo xong thì tiến hành kiểm tra cờ nhận dữ liệu RI có bằng 1 hay không, nếu bằng 0 thì không có dữ liệu gởi đến, nếu bằng 1 thì đã có dữ liệu, tiến hành xóa cờ nhận để báo hiệu cho lần sau, tiến hành nhận dữ liệu rồi gởi ra port0 rồi quay trở lại kiểm tra cờ để nhận byte tiếp theo.

➢ **Chương trình Keil-C: cho vi điều khiển A**

```
#include <AT89X52.h>
void DELAY(unsigned int x)
{ unsigned int x;
      for (y = 0; y<x; y++)    {;}
}
void MAIN ()
{     SCON = 0X50;
      TMOD = T1_M1_;      TH1 = -3;          TR1 = 1;
      P0=0X00;
 while(1)
            {     SBUF = P0;
                  do{ }
                  while (TI ==0);
                  TI =0;              DELAY(10000); P0++;

      }
}
```

Chương trình chính: thực hiện khởi tạo truyền dữ liệu bằng cách gán giá trị 50H = 01010000B cho thanh ghi SCON giống như ở chương trình viết bằng ngôn ngữ Assembly. Tương tự cho các lệnh gán còn lại. Lệnh

kiểm tra cờ TI bằng vòng lặp do while, nếu TI lên 1 thì thoát, tiến hành xóa TI, delay và tăng giá trị port0 rồi lặp lại.

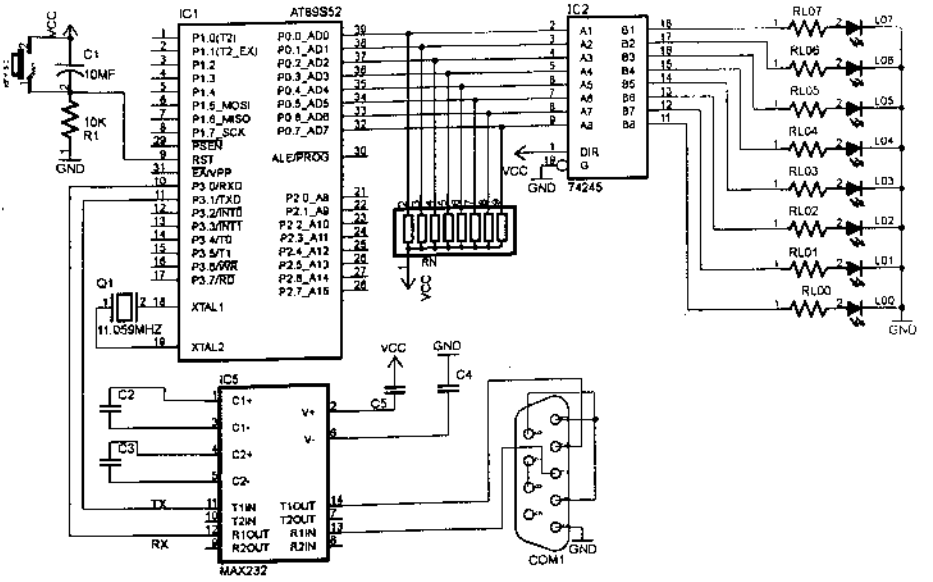
➢ **Chương trình cho vi điều khiển B**

```
#include <AT89X52.h>
void MAIN ()
{     SCON = 0X50;
      TMOD = T1_M1_;      TH1 = -3;          TR1 = 1;
 while(1)
            {
                  do{ }
                  while (RI ==0);
                  P0 = SBUF;              RI = 0;
      }
}
```

Bài 10-2: Một hệ thống dùng vi điều khiển AT89S52 giao tiếp nối tiếp với máy tính, vi điều khiển chờ nhận dữ liệu từ máy tính gởi xuống và nhận dữ liệu gởi ra port0 để điều khiển 8 led đơn.

Sử dụng tốc độ truyền là 9600 baud.

➢ ***Sơ đồ mạch:***



Hình 10-11: *Sơ đồ giao tiếp vi điều khiển với máy tính.*

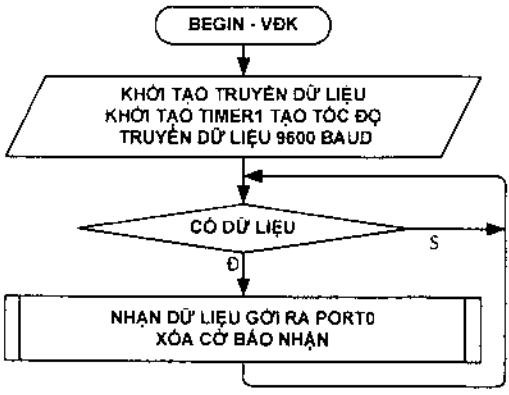
Máy tính có chuẩn truyền dữ liệu nối tiếp qua cổng COM theo chuẩn RS232 có thể giao tiếp với các vi điều khiển.

Chuẩn RS232 có mức logic '1' với điện áp từ - 4V đến -15V, mức logic '0' với điện áp từ 4V đến 15V, có thể truyền đi xa với khoảng cách 1000m.

Các vi điều khiển chỉ tương thích với chuẩn TTL với logic '1' là 5V và logic '0' là 0V.

Chuẩn RS232 không tương thích với chuẩn TTL nên sử dụng thêm IC RS232 hay MAX232 để chuyển đổi chuẩn RS232 sang chuẩn TTL (transistor – transistor logic).

➢ ***Lưu đồ:***



Hình 10-12: *Lưu đồ chờ nhận dữ liệu từ máy tính.*

➢ ***Chương trình Keil-C:***

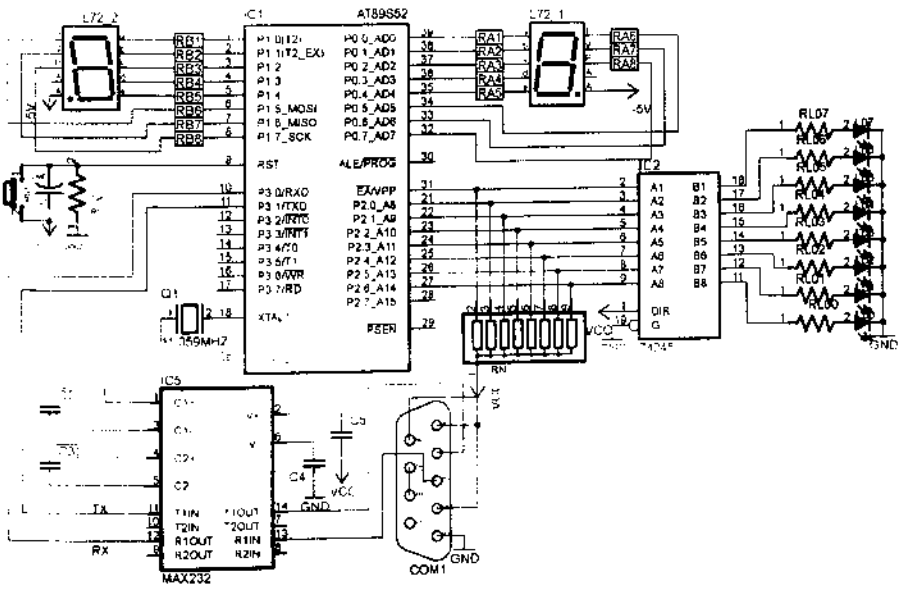
```
#include <AT89X52.h>
void MAIN ()
{     SCON = 0X50;
      TMOD = T1_M1  :      TH1 = -3:          TR1 = 1;
while(1)
            {
                  do{}
                  while (RI ==0);
                  P0 = SBUF:                 RI = 0;
      }
}
```

❖ Giải thích chương trình:

Ở tài liệu này chỉ trình bày chương trình cho vi điều khiển chờ nhận dữ liệu rồi gởi ra port, bên máy tính muốn giao tiếp gởi dữ liệu xuống vi điều khiển thì phải sử dụng một ngôn ngữ lập trình như Visual basic hay C++ hay C#, phần này không được trình bày ở tài liệu này, bạn có thể tham khảo thêm ở phần tài liệu thực hành.

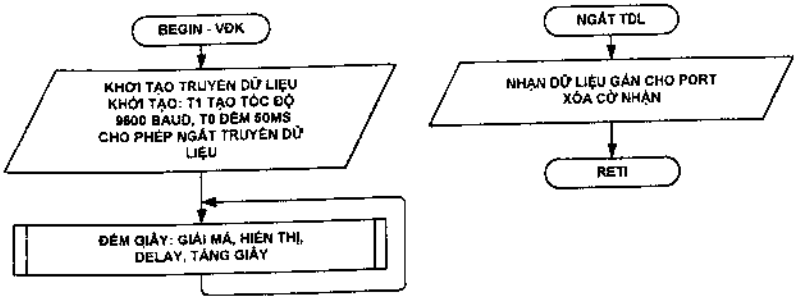
Bài 10-3: Một hệ thống gồm vi điều khiển AT89S52 điều khiển đếm giây hiển thị trên hai led 7 đoạn dùng port0 và port1, port2 điều khiển 8 led đơn, có giao tiếp với máy tính để nhận dữ liệu gởi ra port 2, sử dụng ngắt để nhận dữ liệu, mạch đếm giây dùng timer0 để định thời.

➢ Sơ đồ mạch:



Hình 10-13: Sơ đồ mạch đếm giây và giao tiếp máy tính nhận dữ liệu.

➢ ***Lưu đồ:***



Hình 10-14: Lưu đồ điều khiển và nhận dữ liệu từ máy tính dùng ngắt.

➢ **Chương trình Keil-C:**

```
#include <AT89X52.h>
unsigned char MA7D[10] =
{0XC0,0XF9,0xA4,0XB0,0X99,0X92,0X82,0XF8,0X80,0X90};
int DEM,CHUC,DONVI;

void SERIAL_interrupt() interrupt 4 using 0
{           P2 = SBUF;          RI = 0;     }

void DELAY_TIMER0()
{     int BDN;   TR0 = 1;
      for (BDN = 0; BDN<20;BDN++)
      {     do    {}
            while(TF0==0);
            TF0 = 0;   TH0 = 0X3C;    TL0 = 0XB0;
      }
      TR0 = 0;
}
MAIN ()
{     SCON = 0X50;          TMOD = T1_M1_+ T0_M0_;
      TH1 = -3;       TR1 = 1;          TH0 = 0X3C;    TL0 = 0XB0;
      EA=1;                 ES=1;
 while(1)
            {
            for (DEM = 0; DEM <60; DEM++)
                  {     CHUC = DEM /10;          DONVI = DEM %10;
                        P0 = MA7D[DONVI];        P1 = MA7D[CHUC];
                        DELAY_TIMER0();
                  }
            }
}
```

❖ **Giải thích chương trình:**

Chương trình chính có chức năng khởi tạo truyền dữ liệu với tốc độ là 9600 baud, timer1 hoạt động mod 2, timer0 hoạt động mod 1. Nạp hằng số đếm bằng – 3 vào thanh ghi TH1 để tạo tốc độ 9600 baud, nạp các hằng số vào các thanh ghi của timer0 để đếm 50ms, cho phép ngắt truyền dữ liệu và ngắt toàn cục.

Sau khi khởi tạo xong thì cho phép đếm giây từ 00 đến 59 bằng vòng lặp for, tiến hành tách hàng chục và đơn vị, giải mã hiển thị, delay 1s dùng timer0.

Khi có dữ liệu từ máy tính gởi xuống thì sẽ phát sinh yêu cầu ngắt, chương trình chính sẽ bị ngừng để nhảy đến chương trình con phục vụ ngắt để nhận dữ liệu gán cho port2, xóa cờ báo ngắt để phục vụ cho lần sau, trở lại thực hiện tiếp chương trình chính.

V. TRUYỀN DỮ LIỆU SPI CỦA AT89S8252

1. Truyền dữ liệu SPI của AT89S8252

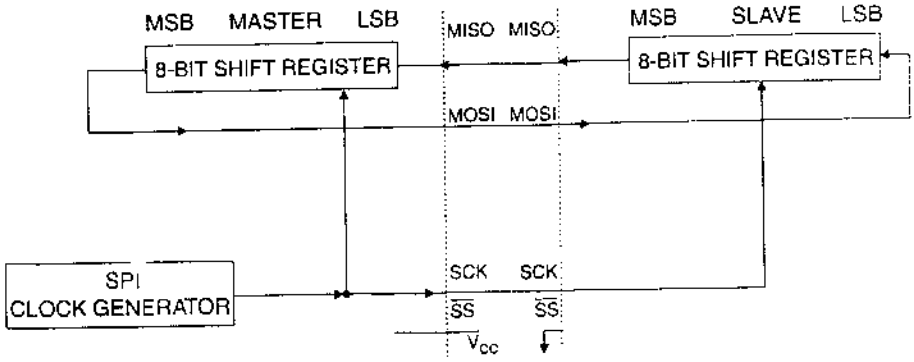
Vi điều khiển AT89S52 không tích hợp chuẩn truyền dữ liệu SPI và I2C, vi điều khiển có tích hợp chuẩn SPI là AT89S8252.

Truyền dữ liệu SPI cho phép truyền dữ liệu đồng bộ với tốc độ cao giữa vi điều khiển AT89S8252 và các thiết bị ngoại vi hoặc giữa vi điều khiển AT89S8252 với nhau.

Cấu trúc truyền dữ liệu SPI của AT89S8252 như sau:

- Truyền dữ liệu đồng bộ ba dây song công.
- Có thể hoạt động ở chế độ chủ hoặc tớ.
- Tốc độ truyền lớn nhất là 1,5MHz.
- Có thể lập trình truyền dữ liệu bắt đầu bằng bit dữ liệu LSB hay MSB.
- Có hai bit để lập trình tốc độ truyền dữ liệu.
- Có cờ báo ngắt khi truyền xong.
- Có cờ phát hiện ghi dữ liệu chồng.
- Có thể đánh thức CPU AT89S8252 khỏi chế độ ngủ (chỉ dùng cho tớ).

Sơ đồ cấu trúc truyền dữ liệu SPI giữa chủ và tớ như hình 10-15:



Hình 10-15: Hệ thống truyền dữ liệu SPI.

Tín hiệu xung SCK do vi điều khiển chủ cung cấp cho tớ. Khi tiến hành ghi dữ liệu lên thanh ghi SPI của vi điều khiển chủ sẽ làm bộ phát xung clock của khối SPI hoạt động tạo xung clock dịch dữ liệu ra ngoài chân MOSI, đồng thời dịch dữ liệu từ ngõ vào của chân MISO từ tớ.

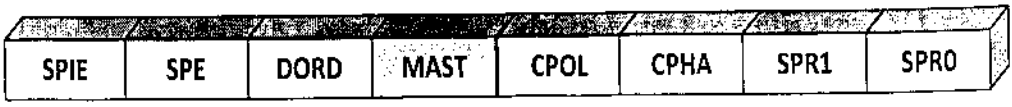
Sau mỗi lần truyền xong một byte thì bộ phát xung clock sẽ ngừng, cờ báo truyền dữ liệu SPI là SPIF sẽ lên 1, nếu bit cho phép ngắt SPIE bằng 1 thì sẽ phát sinh yêu cầu ngắt. Yêu cầu ngắt của SPI sẽ or với ngắt của port truyền dữ liệu nối tiếp UART.

2. Chức năng các thanh ghi truyền dữ liệu SPI của AT89S8252

Các bit điều khiển và các bit trạng thái của truyền dữ liệu SPI chứa trong thanh ghi SPCR. Các bit dữ liệu chứa trong thanh ghi SPSR.

➢ ***Thanh ghi SPCR (SPI control register)***

Thanh ghi SPCR chứa các bit để điều khiển truyền dữ liệu SPI, cấu trúc của thanh ghi như hình sau:



Hình 10-16: Thanh ghi SPCR.

Bảng 10-4: Tóm tắt chức năng các bit trong thanh ghi SPCR:

Bit	Kí hiệu	Chức năng
7	SPIE	SPI Interrupt Enable: bit cho phép/cấm ngắt SPIE = 1: cho phép SPI ngắt. SPIE = 0: cấm SPI ngắt.

6	SPE	SPE Enable: bit cho phép thiết lập chức năng port SPE = 1: cho phép các port: P1.4, P1.5, P1.6, P1.7 là các chân SPI SPE = 0: không cho phép port hoạt động chức năng SPI
5	DORD	Data Order: lựa chọn trình tự xuất dữ liệu: DORD = 1: truyền theo trình tự từ bit LSB đến MSB. DORD = 0: truyền theo trình tự từ bit MSB đến LSB.
4	MSTR	Master/Slave Select: bit lựa chọn chủ tớ MSTR = 1: chọn chức năng chủ. MSTR = 0: chọn chức năng tớ.
3	CPOL	Clock Polariry: bit lựa chọn mức tích cực của xung clock CPOL = 1: SCK sẽ ở mức 1 khi ngừng. CPOL = 0: SCK sẽ ở mức 0 khi ngừng
2	CPHA	Clock Phase: bit lựa chọn pha của xung clock. Bit này cùng với bit CPOL sẽ điều khiển mối quan hệ giữa xung clock và dữ liệu của chủ và tớ. Trình bày rõ ở dạng sóng.
1	SPR1	SPI Clock Rate Select. Bit lựa chọn tần số xung clock – bit thứ 1.
0	SPR0	SPI Clock Rate Select. Bit lựa chọn hệ số chia xung clock – bit thứ 0. Tần số truyền bằng tần số dao động thạch anh chia cho hệ số n. Hai bit SPR1SPR0 có bốn trạng thái 00, 01, 10, 11 tương đương với n bằng 4, 16, 64, 128.

➢ ***Thanh ghi SPSR (SPI Staus Register)***

Thanh ghi SPSR chứa các bit trạng thái, cấu trúc của thanh ghi như hình sau:

Hình 10-17: *Thanh ghi SPSR.*

Bảng 10-5: Tóm tắt chức năng các bit trong thanh ghi SPSR:

Bit	Kí hiệu	Chức năng
7	SPIF	SPI Interrupt Flag: cờ báo ngắt của SPI Khi truyền xong dữ liệu thì cờ SPIE lên 1. Nếu cờ SPIF và WCOL bằng 1 thì sẽ bị xóa khi đọc thanh ghi SPSR, sau đó tiến hành đọc thanh ghi chứa dữ liệu.
5	WCOL	Write Collision Flag: Cờ báo hiệu xung đột dữ liệu: cờ này sẽ lên 1 nếu nhận dữ liệu mới trong khi dữ liệu cũ truyền chưa xong. Khi đọc thanh ghi SPSR sẽ xóa cờ này.

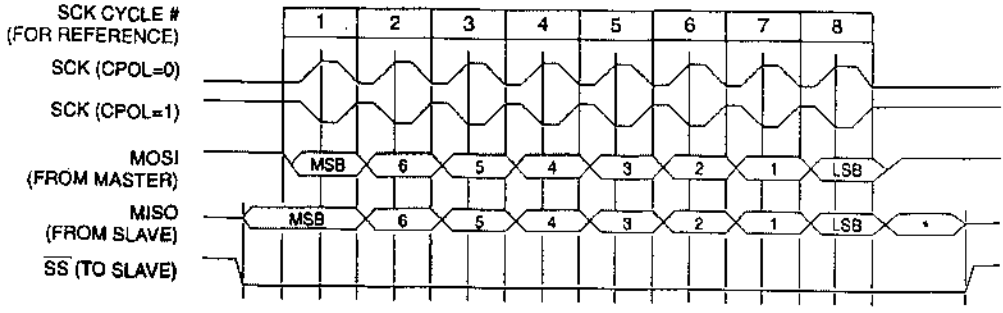
➢ ***Thanh ghi dữ liệu SPDR (SPI Data Register)***

Thanh ghi SPDR chứa dữ liệu để phát đi, cấu trúc của thanh ghi như hình sau:

Hình 10-18: Thanh ghi SPDR.

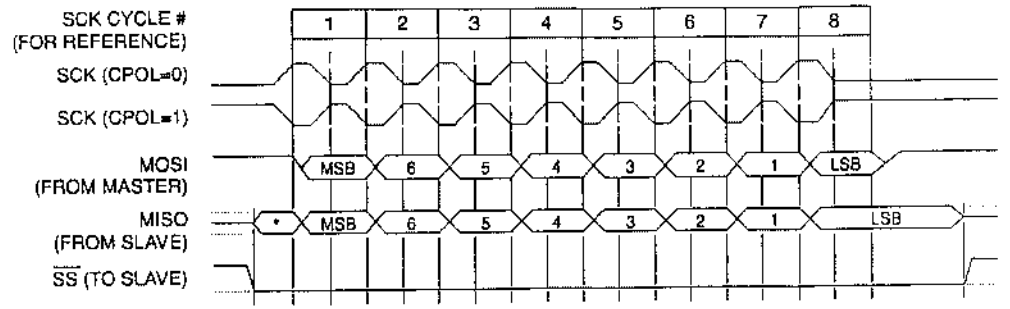
3. Dạng sóng truyền dữ liệu SPI của AT89S8252

Dạng sóng truyền dữ liệu SPI khi bit CPHA bằng 0 như hình 10-19:



Hình 10-19: Dạng sóng truyền SPI khi bit CPHA bằng 0.

Dạng sóng truyền dữ liệu SPI khi bit CPHA bằng 1 như hình 10-20:



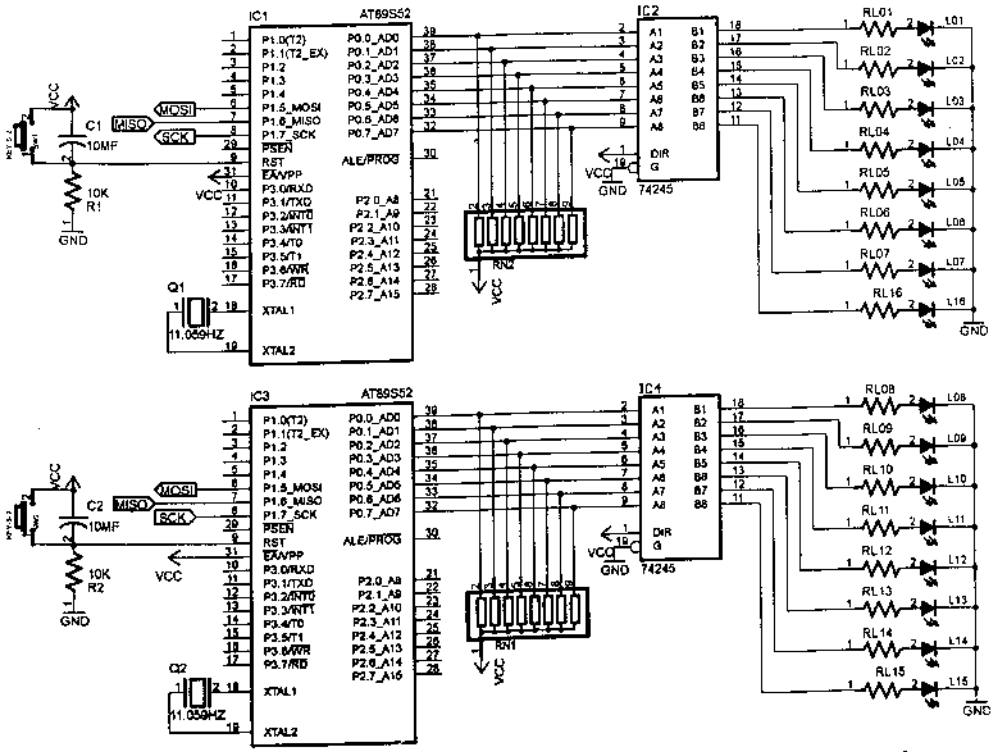
Hình 10-20: Dạng sóng truyền SPI khi bit CPHA bằng 1.

4. Ứng dụng truyền dữ liệu của AT89S52

Phần này trình bày các ứng dụng truyền dữ liệu SPI của AT89S8252.

Bài 10-4: Một hệ thống dùng hai vi điều khiển AT89S8252 truyền dữ liệu nối tiếp SPI. Vi điều khiển A (chủ) dùng port0 điều khiển 8 led đơn đếm nhị phân đồng thời gởi dữ liệu của port0 sang vi điều khiển B (tớ) để gởi ra port0.

➢ ***Sơ đồ mạch:***



Hình 10-21: Sơ đồ giao tiếp hai vi điều khiển để truyền dữ liệu chuẩn SPI.

➢ ***Chương trình Assembly:*** cho vi điều khiển A (chủ) truyền dữ liệu:

```
        MOSI    BIT  P1.5            ;SPI
        MISO    BIT  P1.6            ;SPI
        SCK BIT  P1.7            ;SPI

        SPCR DATA 0D5H           ; SPI CONTROL REGISTER
        SPSRDATA 0AAH            ; SPI STATUS REGISTER
        SPIF EQU      10000000B; INTERRUPT FLAG
        SPDR DATA 086H           ; SPI DATA REGISTER

        ORG 0000
MAIN:       MOV     SPCR, #01010101B        ; cho phép ngắt,
nhận dữ liệu, master, hệ số chia 16
        MOV     P0, #0
MAIN1:  CALL    GOI_DLSPI
        CALL    DELAY1S
        INC  R0
        JMP  MAIN1
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
;CHUONG TRINH CON TRUYEN DU LIEU SANG SLAVE
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
GOI_DLSPI:  MOV     SPDR, P0
GOI_DLSPI1:  MOVA, SPSR
        ANL A, #SPIF
        JZ   GOI_DLSPI1
        MOV     A, SPDR
        RET
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

$INCLUDE(TV_DELAY.ASM)
        END
```

➢ **Chương trình Assembly:** cho vi điều khiển B (tớ) nhận dữ liệu không dùng ngắt:

```
MOSI   BIT          P1.5          ;SPI
MISO   BIT          P1.6          ;SPI
SCK    BIT          P1.7          ;SPI

SPCR DATA 0D5H      ; SPI CONTROL REGISTER
SPSR DATA 0AAH      ; SPI STATUS REGISTER
SPIF EQU 10000000B  ; INTERRUPT FLAG
SPDR DATA 086H      ; SPI DATA REGISTER
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
       ORG   0000
       MOV   SPCR,#01000101b; cho phép ngắt, nhận dữ liệu, slave, hệ số chia 16
MAIN1: MOV   A,SPSR  ;đọc thanh ghi trạng thái
       ANL   A,#SPIF ;chỉ giữ lại bit SPIF
       JZ    MAIN1   ;quay lại nếu bằng 1 hay chưa có dữ liệu

       MOV   P0,SPDR       ;READ INPUT DATA
       SJMP  MAIN1
       END
```

➢ ***Chương trình Assembly:*** cho vi điều khiển B (tớ) dùng ngắt để nhận dữ liệu:

```
MOSI   BIT          P1.5          ;SPI
MISO   BIT          P1.6          ;SPI
SCK    BIT          P1.7          ;SPI

SPCR DATA 0D5H      ; SPI CONTROL REGISTER
SPSR DATA 0AAH      ; SPI STATUS REGISTER
SPIF EQU 10000000B  ; INTERRUPT FLAG
SPDR DATA 086H      ; SPI DATA REGISTER
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
            ORG     0000
            JMP     MAIN
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
            ORG     0023H
            MOV     P0,SPDR                 ;READ INPUT DATA
            CLR     RI
            RETI
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
MAIN:       MOV     SPCR,#11000101b     ;cho phép ngắt, nhận dữ liệu,
hệ số chia 16
            SETB    EA
            JMP     $
            END
```

➢ ***Chương trình Keil-C:*** cho vi điều khiển A

```
#include <AT898252.H>
void DELAY_TIMER0()
{     int BDN;
      TR0 = 1;
      for (BDN = 0; BDN<20; BDN++)
      {     do    {}   while(TF0==0);
            TF0 = 0;   TH0 = 0X3C;     TL0 = 0XB0;
      }
      TR0 = 0;
}
MAIN ()
{     SPCR = 0X55;
      TMOD = T0_M0_; TH0 = 0X3C; TL0 = 0XB0; P0=0X00;
      while(1)
            {     SPDR = P0;
                  do {} while(SPIF_ ==0);
                  DELAY_TIMER0();   P0 = P0+1;
```

```
        }
}
```

Chương trình chính: thực hiện khởi tạo truyền dữ liệu SPI, khởi tạo timer0 đếm 50ms, cho port0 bằng 0, vòng lặp while tiến hành gởi dữ liệu ra chân MOSI, vòng lặp do while tiến hành kiểm tra cờ báo ngắt xem nếu còn bằng 0 thì chờ, nếu lên 1 thì thoát. Tiến hành delay và tăng giá trị của port0 lên 1 rồi quay trở lại làm tiếp.

➢ ***Chương trình cho vi điều khiển B*** – tớ chỉ chờ nhận dữ liệu

```
#include <AT898252.H>
MAIN ()
{     SPCR = 0X45;    P0 = 0X00;
      while(1)
            {     do { } while(SPIF_ ==0);
                  P0 = SPDR;
            }
}
```

Chương trình tớ sau khi khởi tạo thì kiểm tra cờ báo ngắt của SPI và chờ nếu chưa có dữ liệu, nếu có thì đọc dữ liệu và quay lại làm tiếp cho dữ liệu tiếp theo.

VI. TRUYỀN DỮ LIỆU I2C

1. Giới thiệu

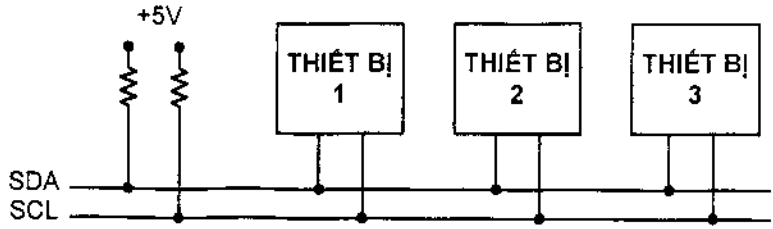
Vi điều khiển AT89S52 không tích hợp chuẩn truyền dữ liệu I2C nhưng có thể lập trình giao tiếp với các thiết bị ngoại vi theo chuẩn I2C.

Có rất nhiều ngoại vi tích chuẩn giao tiếp I2C như bộ nhớ Eeprom nối tiếp, ADC-DAC, Realtime, IC mở rộng ngoại vi, …

Trong phần này trình bày, chuẩn giao tiếp I2C và lập trình cho vi điều khiển AT89S52 thực hiện giao tiếp với đồng hồ thời gian thực theo chuẩn I2C.

2. Tổng quan về truyền dữ liệu chuẩn I2C

I2C viết tắc của từ Inter-Integrated Circuit là một chuẩn truyền thông do hãng điện tử Philips Semiconductor sáng lập cho phép giao tiếp một thiết bị chủ với nhiều thiết bị tớ với nhau như hình 10-23.



Hình 10-23: *Hệ thống các thiết bị giao tiếp theo chuẩn I2C.*

Chuẩn giao tiếp I2C có hai đường tín hiệu tên là SDA (serial data) có chức năng truyền tải dữ liệu và tín hiệu SCL (serial clock) truyền tải xung clock để dịch chuyển dữ liệu.

Trong hệ thống truyền dữ liệu I2C thì thiết bị nào cung cấp xung clock thì được gọi là chủ (master), thiết bị nhận xung clock được gọi là tớ (slave).

Thiết bị chủ chỉ có 1, thiết bị tớ thì có nhiều, mỗi thiết bị tớ sẽ có một địa chỉ độc lập, chuẩn truyền ban đầu dùng địa chỉ 7 bit nên có thể một chủ giao tiếp với 128 thiết bị tớ. Các thiết bị sau này tăng thêm số bit địa chỉ nên có thể giao tiếp nhiều hơn.

Địa chỉ của thiết bị tớ thường thì do nhà chế tạo thiết bị thiết lập sẵn.

3. Quy trình truyền dữ liệu chuẩn I2C

Quy trình thiết bị chủ giao tiếp với thiết bị tớ để thực hiện việc ghi đọc dữ liệu như sau:

➢ ***Quá trình thiết bị chủ ghi dữ liệu vào thiết bị tớ***

Bước 1: Thiết bị chủ tạo trạng thái START để bắt đầu quá trình truyền dữ liệu – các thiết bị tớ sẽ ở trạng thái sẵn sàng nhận địa chỉ từ thiết bị chủ.

Bước 2: Thiết bị chủ gởi địa chỉ của thiết bị tớ cần giao tiếp – khi đó tất cả các thiết bị tớ đều nhận địa chỉ và so sánh với địa chỉ của mình, các thiết bị tớ sau khi phát hiện không phải địa chỉ của mình thì chờ cho đến khi nào nhận trạng thái START mới.

Trong dữ liệu 8 bit thì có 7 bit địa chỉ và 1 bit điều khiển đọc/ghi (R/W): thì bit này bằng 0 để báo cho thiết bị tớ sẽ nhận byte tiếp theo.

Bước 3: Thiết bị chủ chờ nhận tín hiệu bắt tay từ thiết bị tớ. Thiết bị tớ nào đúng địa chỉ thì phát một tín hiệu trả lời cho chủ biết.

Bước 4: Thiết bị chủ tiến hành gởi địa chỉ của ô nhớ bắt đầu cần ghi dữ liệu, bit R/W ở trạng thái ghi.

Bước 5: Thiết bị chủ chờ nhận tín hiệu trả lời từ thiết bị tớ.

Bước 6: Thiết bị chủ gởi tiến hành gởi dữ liệu để ghi vào thiết bị tớ, mỗi lần ghi 1 byte, sau khi gởi xong thì tiến hành chờ nhận tín hiệu trả lời từ thiết bị tớ, quá trình thực hiện cho đến byte cuối cùng xong rồi thì thiết bị chủ chuyển sang trạng thái STOP để chấm dứt quá trình giao tiếp với thiết bị tớ.

➢ ***Quá trình thiết bị chủ đọc dữ liệu vào thiết bị tớ***

Bước 1: Thiết bị chủ tạo trạng thái START để bắt đầu quá trình truyền dữ liệu – các thiết bị tớ sẽ ở trạng thái sẵn sàng nhận địa chỉ từ thiết bị chủ.

Bước 2: Thiết bị chủ gởi địa chỉ của thiết bị tớ cần giao tiếp – khi đó tất cả các thiết bị tớ đều nhận địa chỉ và so sánh với địa chỉ của mình, các thiết bị tớ sau khi phát hiện không phải địa chỉ của mình thì chờ cho đến khi nào nhận trạng thái START mới.

Trong dữ liệu 8 bit thì có 7 bit địa chỉ và 1 bit điều khiển đọc/ghi (R/W): thì bit này bằng 0 để báo cho thiết bị tớ sẽ nhận byte tiếp theo.

Bước 3: Thiết bị chủ chờ nhận tín hiệu bắt tay từ thiết bị tớ. Thiết bị tớ nào đúng địa chỉ thì phát một tín hiệu trả lời cho chủ biết.

Bước 4: Thiết bị chủ tiến hành gởi địa chỉ của ô nhớ bắt đầu cần đọc dữ liệu, bit R/W ở trạng thái đọc.

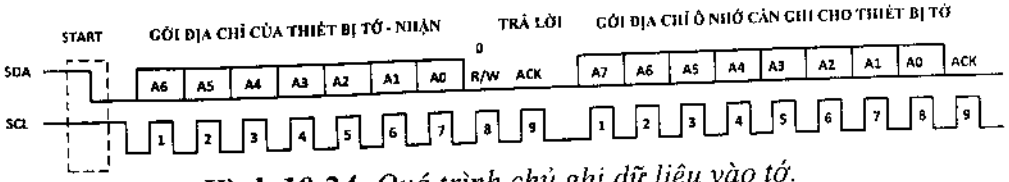
Bước 5: Thiết bị chủ chờ nhận tín hiệu trả lời từ thiết bị tớ.

Bước 6: Thiết bị chủ chuyển sang trạng thái STOP, bắt đầu lại trạng thái START, tiến hành gởi địa chỉ của thiết bị và bit R/W bằng 1 để yêu cầu tớ gởi dữ liệu nội dung ô nhớ của địa chỉ đã nhận.

Bước 7: Thiết bị chủ sau khi nhận sẽ báo tín hiệu trả lời, quá trình này thực hiện cho đến khi nhận hết dữ liệu mong muốn thì thiết bị chủ tạo tín hiệu STOP để chấm dứt.

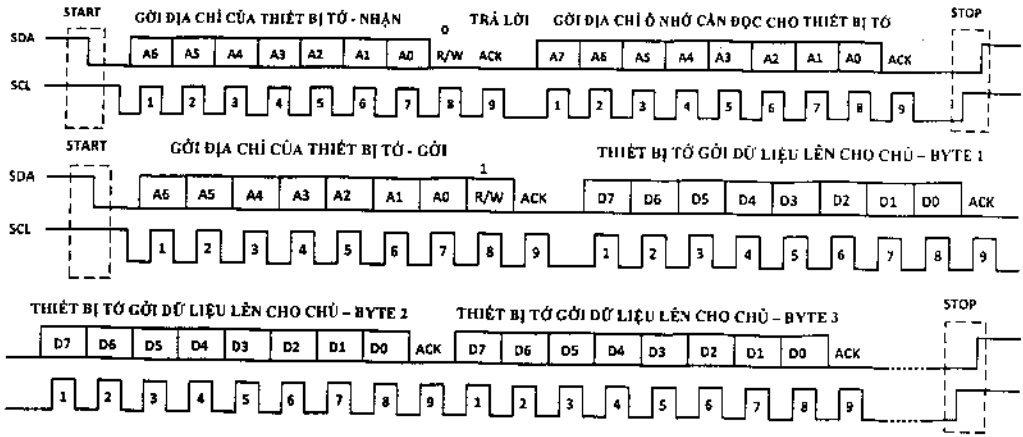
4. Dạng sóng truyền dữ liệu chuẩn I2C

Dạng sóng truyền dữ liệu I2C của quá trình ghi dữ liệu từ chủ lên tớ như hình 10-24.



Hình 10-24: *Quá trình chủ ghi dữ liệu vào tớ.*

Dạng sóng truyền dữ liệu I2C của quá trình đọc dữ liệu từ tớ như hình 10-25.

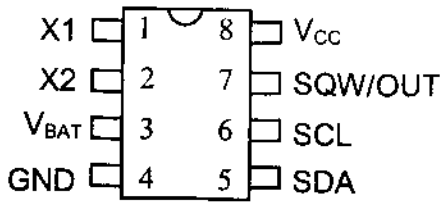


Hình 10-26: Quá trình chủ ghi dữ liệu vào tớ.

5. Khảo sát IC Realtime DS13B07

DS13B07 là chip đồng hồ thời gian thực của Dallas Semiconductor. Chip 64 ô nhớ, trong đó có 8 ô nhớ 8-bit chứa thời gian: giây, phút, giờ, thứ (trong tuần), ngày, tháng, năm và thanh ghi điều khiển ngõ ra, vùng nhớ còn lại là 56 ô dùng có thể dùng để lưu dữ liệu.

DS1307 giao tiếp theo chuẩn nối tiếp I2C nên sơ đồ chân bên ngoài chỉ có 8 chân.



Hình 10-27: Sơ đồ chân DS1307.

Các chân của DS1307:

Hai chân **X1** và **X2**: dùng để kết nối với tụ thạch anh có tần số 32768Hz để tạo dao động.

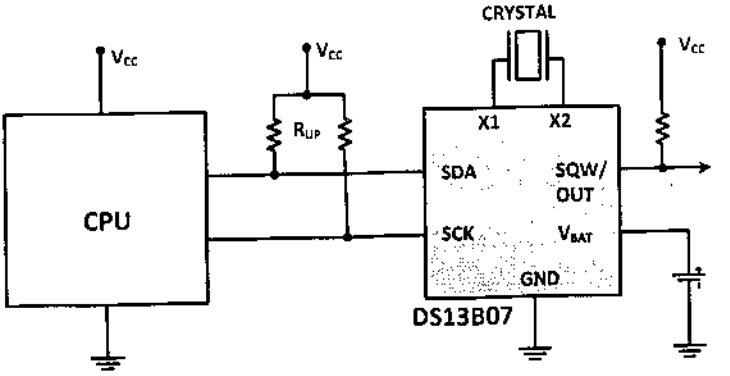
Chân $\mathbf{V_{BAT}}$ nối với nguồn dương của pin 3V3 để chip vẫn hoạt động đếm thời gian khi mất nguồn.

Chân **GND** là chân nối 0V.

Chân **Vcc** nối với nguồn Vcc có thể là 5V hoặc 3V3 tùy thuộc vào từng loại chíp.

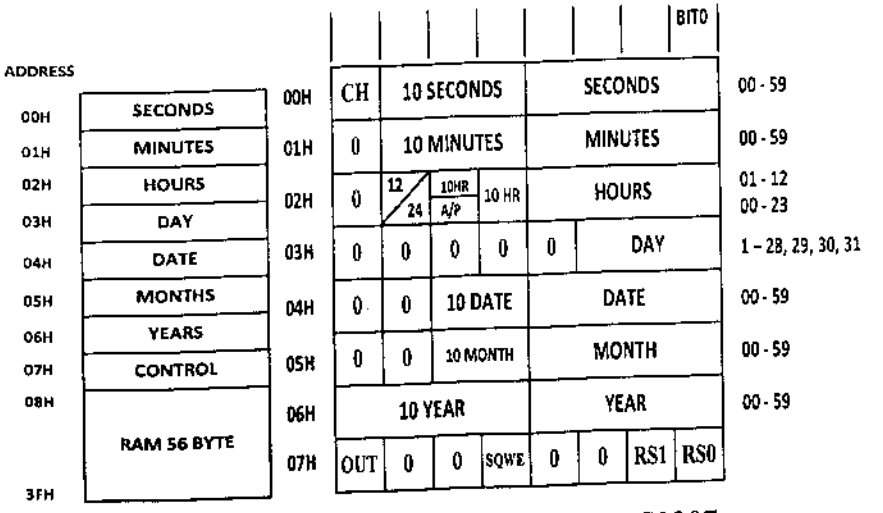
Chân **SQW/OUT** là ngõ ra tạo xung vuông (Square Wave / Output Driver), tần số có thể lập trình.

Hai chân **SCL, SDA** là hai đường tín hiệu giao tiếp chuẩn I2C.



***Hình 10-27:** Sơ đồ kết nối DS1307*

Tổ chức bộ nhớ bên trong của RT DS13B07 như hình



***Hình 10-28:** Tổ chức bộ nhớ của DS1307.*

***Hình 10-29:** Tổ chức các thanh ghi thời gian.*

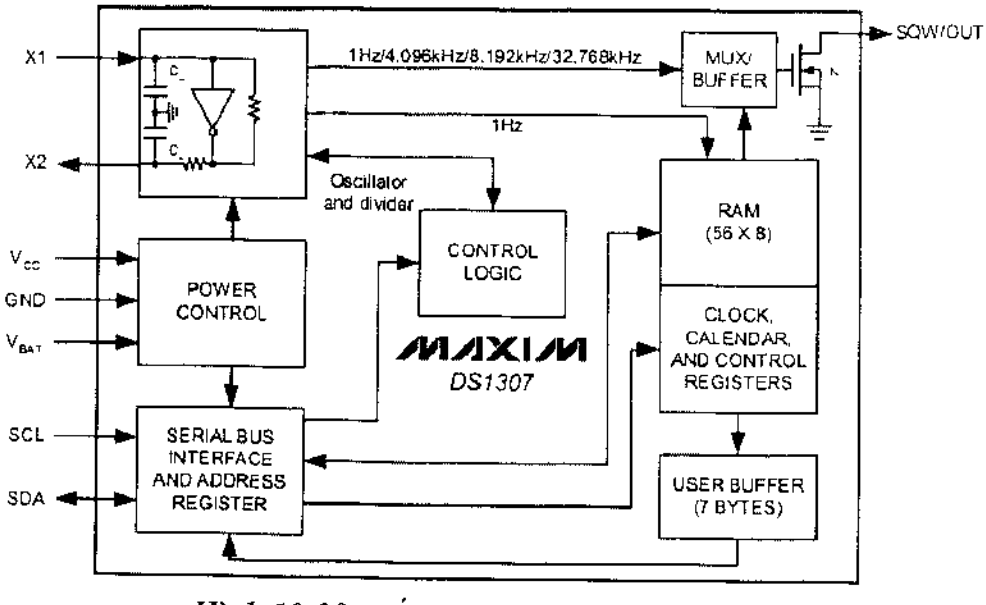
Vì 7 thanh ghi đầu tiên là quan trọng nhất trong hoạt động của DS1307 nên ta quan sát tổ chức theo từng bit của các thanh ghi này như trong hình dưới:

Ô nhớ lưu giây (SECONDS): có địa chỉ là 0x00 có chức năng lưu hàng chục giây và hàng đơn vị giây. Bit thứ 7 có tên CH (Clock halt – ngừng

đồng hồ), nếu bit này bằng 1 thì bộ dao động trong chip ngừng làm đồng hồ ngừng hoạt động. Nếu muốn đồng hồ hoạt động thì bit này phải bằng 0.

Ô nhớ lưu phút (MINUTES): có địa chỉ 0x01, có chức năng lưu phút hàng đơn vị và hàng chục, bit 7 luôn bằng 0.

Ô nhớ lưu giờ (HOURS): có địa chỉ 0x02 có chức năng lưu hàng chục giờ và hàng đơn vị giờ ở 2 chế độ 12 giờ và 24 giờ được lựa chọn bởi bit thứ 6 có tên là 12/24.



***Hình 10-30:** Cấu trúc bên trong DS13B07.*

Nếu bit 12/24 chọn chế độ 24 giờ thì phần hàng chục giờ sử dụng 2 bit thứ tư và thứ năm có ký hiệu là 10HR.

Nếu bit 12/24 chọn chế độ 12 giờ thì phần hàng chục giờ sử dụng bit thứ tư, còn bit thứ năm sẽ có ký hiệu là A/P tương ứng với hai chế độ giờ AM và PM.

Bit 7 luôn bằng 0.

Ô nhớ lưu thứ (DAY – ngày trong tuần): có địa chỉ 0x03, có chức năng lưu thứ trong tuần có giá trị từ 1 đến 7 tương ứng từ Chủ nhật đến thứ bảy trong tuần, chỉ sử dụng 3 bit thấp.

Các ô còn lại là ngày tháng năm.

Cấu trúc bên trong của **DS13B07** như hình 10-30:

6. Ứng dụng realtime DS13B07

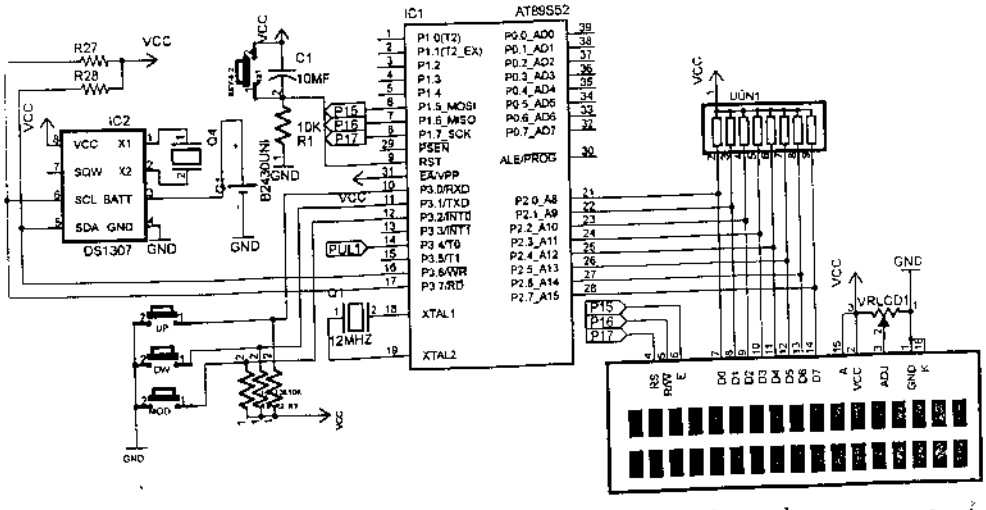
Phần này trình bày các ứng dụng truyền dữ liệu theo chuẩn I2C.

Bài 10-5: Một hệ thống đồng hồ thời gian thực dùng vi điều khiển AT89S8252 giao tiếp với DS13B07 theo chuẩn I2C, giao tiếp với LCD và ba nút nhấn để hiệu chỉnh thời gian.

Hãy viết chương trình đồng hồ hiển thị giờ phút giây, ngày tháng năm trên LCD.

Sau đó hãy thêm phần hiệu chỉnh thời gian bằng ba phím: MOD, UP, DOWN

➢ ***Sơ đồ mạch:***



Hình 10-31: *Sơ đồ giao tiếp AT89S52 với DS13B07 để truyền dữ liệu chuẩn I2C.*

➢ ***Chương trình Keil-C:***

```
#include<AT89X52.H>
#define    GIAY_HTAI      0X55
#define    PHUT_HTAI      0X20
#define    GIO_HTAI       0X15
#define    THU_HTAI       2
#define    NGAY_HTAI      0X17
#define    THANG_HTAI     0X10
#define    NAM_HTAI       0X12
```

```
#define     MA_DS                0X98

#include<THUVIEN_LCD1.C>
#include< THUVIEN_DS13B07.C>
unsigned char j;
const unsigned char HANG1[16]={"CLOCK:        "};
const unsigned char HANG2[16]={"*DHSPKT*      "};

void HIENTHI_TIME_DS13B07 ( )
{     COMMAND_WRITE (0xC8);
      DATA_WRITE (MACHUCH);    DATA_WRITE (MADONVIH);
      DATA_WRITE (' ');
      DATA_WRITE (MACHUCM);    DATA_WRITE (MADONVIM);
      DATA_WRITE (' ');
      DATA_WRITE (MACHUCS);    DATA_WRITE (MADONVIS);

      COMMAND_WRITE (0x88);
      DATA_WRITE (MACHUCD);    DATA_WRITE (MADONVID);
      DATA_WRITE (' ');
      DATA_WRITE (MACHUCMT);   DATA_WRITE (MADONVIMT);
      DATA_WRITE (' ');
      DATA_WRITE (MACHUCY);    DATA_WRITE (MADONVIY);

}

void GIAI_MA_TIME_DS13B07()
{     MADONVIS=(GIAY_DS13%16) +0X30;
      MACHUCS=(GIAY_DS13/16) +0X30;
      MADONVIM=(PHUT_DS13%16) +0X30;
      MACHUCM=(PHUT_DS13/16) +0X30;
      MADONVIH=(GIO_DS13%16)  +0X30;
      MACHUCH=(GIO_DS13/16)  +0X30;
```

```
    MADONVID=(NGAY_DS13%16) +0X30;
    MACHUCD=(NGAY_DS13/16) +0X30;
    MADONVIMT=(THANG_DS13%16) +0X30;
    MACHUCMT=(THANG_DS13/16) +0X30;
    MADONVIY=(NAM_DS13%16) +0X30;
    MACHUCY=(NAM_DS13/16) +0X30;
    }
void main( )
{   READ_CODE_DS13B07();
    if(MA_DS13!=0X98)
    {   LOAD_CURRENT_TIME();
    SETUP_NEW_TIME_DS13B07();}
    SETUP_LCD ();
    COMMAND_WRITE(addr_line1);      delay(20);
    for (j=0;j<6;j++)               {   DATA_WRITE (HANG1[j]);
    }
    COMMAND_WRITE(addr_line2);      delay(20);
    for (j=0;j<6;j++)               {   DATA_WRITE (HANG2[j]);
    }
    READ_TIME_DS13B07(1);
    while(1)
     {  GIAI_MA_TIME_DS13B07();
    HIENTHI_TIME_DS13B07();
        do {  READ_SECOND_I2C();}
        while (GIAYTAM_DS13==GIAY_DS13);
        GIAYTAM_DS13=GIAY_DS13;
    READ_TIME_DS13B07(0);
        }
     }
```

Thư viện cho real-time DS13B07:

```
#ifndef SDA
    #define SDA          P3_6
```

```
#endif
#ifndef SCL
  #define SCL                P3_7
#endif
#define ADDR_WR_13B07        0xD0
#define ADDR_RD_13B07        0xD1
#define ADDR_MEM             0x00
unsigned char bdata    RWI2C;
sbit   LSB_RWI2C = RWI2C^0 ;
sbit   MSB_RWI2C = RWI2C^7 ;
signed char
      NAM_DS13,THANG_DS13,NGAY_DS13,THU_DS13,GIO_DS13,
DSTAM,

      PHUT_DS13,GIAY_DS13,MA_DS13,CONTROL_DS13,GIAYTAM
_DS13;
unsigned char
      MADONVIS,MACHUCS,MADONVIM,MACHUCM,MADONVIH,
MACHUCH;
unsigned char
      MADONVID,MACHUCD,MADONVIMT,MACHUCMT,MADON
VIY,MACHUCY;


void START_I2C()
{     SDA = 1;   SCL = 1;    SDA = 0;   SCL = 0;    SDA = 1;}
void STOP_I2C()
{     SDA = 0;   SCL = 1;          SDA = 1; }
void SEND_BYTE_I2C(unsigned char SBYTE)
{     unsigned char I;
      RWI2C = SBYTE;
      for (I=0;I<8;I++)
            {     SDA = MSB_RWI2C; SCL = 1;    SCL = 0;
                  RWI2C = (RWI2C <<1);}
```

```
                SDA = 1;        SCL = 1;
                LSB_RWI2C = SDA;  SCL = 0;
}
void LOAD_CURRENT_TIME()
{     GIAY_DS13     =     GIAY_HTAI;
      PHUT_DS13     =     PHUT_HTAI;
      GIO_DS13      =     GIO_HTAI;
      THU_DS13      =     THU_HTAI;
      NGAY_DS13     =     NGAY_HTAI;
      THANG_DS13    =     THANG_HTAI;
      NAM_DS13      =     NAM_HTAI;
      CONTROL_DS13 = 0X90;
      MA_DS13       =     MA_DS;
}
void SETUP_NEW_TIME_DS13B07()
{     START_I2C();
      SEND_BYTE_I2C(ADDR_WR_13B07);
      SEND_BYTE_I2C(0X00);
      SEND_BYTE_I2C(GIAY_DS13);
      SEND_BYTE_I2C(PHUT_DS13);
      SEND_BYTE_I2C(GIO_DS13);
      SEND_BYTE_I2C(THU_DS13);
      SEND_BYTE_I2C(NGAY_DS13);
      SEND_BYTE_I2C(THANG_DS13);
      SEND_BYTE_I2C(NAM_DS13);
      SEND_BYTE_I2C(CONTROL_DS13);
      SEND_BYTE_I2C(MA_DS13);
      STOP_I2C();
}

char READ_LAST_BYTE_I2C()
{     unsigned char I;
```

```
    for (I=0;I<8;I++)
        {SCL = 1; RWI2C = (RWI2C <<1);    LSB_RWI2C = SDA;
    SCL = 0;}
    return(RWI2C);
}
char READ_1BYTE_I2C()
{   unsigned char RT_RB;
    RT_RB = READ_LAST_BYTE_I2C();//READ_1BYTE_I2C();
    SDA = 0;  SCL = 1;  SCL = 0;
    return(RT_RB);
}
char READ_BYTE_I2C()
{   unsigned char RT_RBT;
    RT_RBT = READ_1BYTE_I2C();
    SDA = 1;
    return (RT_RBT);}
char READ_MEM_I2C(unsigned char ADDR)
{   unsigned char CODE_DS13;
    START_I2C();        SEND_BYTE_I2C(ADDR_WR_13B07);
                        SEND_BYTE_I2C(ADDR);
    STOP_I2C();
    START_I2C();        SEND_BYTE_I2C(ADDR_RD_13B07);
                        CODE_DS13 = READ_LAST_BYTE_I2C();
    STOP_I2C();         return (CODE_DS13);
}
void READ_SECOND_I2C()
{   GIAY_DS13=READ_MEM_I2C(0X00);}
void READ_CODE_DS13B07()
{   MA_DS13 = READ_MEM_I2C(0X08);}
void HTHI_CODE_DS()
{ unsigned char X,Y;
```

```
    X=(MA_DS13%16)+0X30;  Y=(MA_DS13/16)+0X30;
    COMMAND_WRITE(0xD6);
    DATA_WRITE (Y);   DATA_WRITE (X);}
void READ_TIME_DS13B07(bit TT_READ_DST)
{   START_I2C();
    SEND_BYTE_I2C(ADDR_WR_13B07);
    SEND_BYTE_I2C(0X00);
    STOP_I2C();
    START_I2C();
    SEND_BYTE_I2C(ADDR_RD_13B07);

    if(TT_READ_DST==1)
        {   GIAY_DS13 = READ_BYTE_I2C();
            PHUT_DS13 = READ_BYTE_I2C();
            GIO_DS13  = READ_BYTE_I2C();
            THU_DS13  = READ_BYTE_I2C();
            NGAY_DS13 = READ_BYTE_I2C();
            THANG_DS13= READ_BYTE_I2C();
            NAM_DS13  = READ_BYTE_I2C();
            CONTROL_DS13= READ_BYTE_I2C();
            MA_DS13  =  READ_BYTE_I2C();
        }
    else{       GIAY_DS13 = READ_BYTE_I2C();
        if (GIAY_DS13==0x00)
        {   PHUT_DS13 = READ_BYTE_I2C();
            if (PHUT_DS13==0)
            {   GIO_DS13  = READ_BYTE_I2C();
                if (GIO_DS13==0)
                {   THU_DS13  = READ_BYTE_I2C();
                    NGAY_DS13 = READ_BYTE_I2C();
                    THANG_DS13= READ_BYTE_I2C();
                    NAM_DS13  = READ_BYTE_I2C();
```

```
                    CONTROL_DS13= READ_BYTE_I2C();

                    MA_DS13 = READ_BYTE_I2C();
                }}
            }}
                DSTAM = READ_LAST_BYTE_I2C();
                STOP_I2C();
}
```

VII. CÂU HỎI ÔN TẬP – TRẮC NGHIỆM - BÀI TẬP

1. Câu hỏi ôn tập

Câu số 10-1: Hãy cho biết tên các thanh ghi liên quan đến truyền dữ liệu UART của AT89S52.

Câu số 10-2: Hãy cho biết chức năng của các bit trong thanh ghi SCON của vi điều khiển AT89S52.

Câu số 10-3: Hãy cho biết hoạt động truyền dữ liệu mod 0 của vi điều khiển AT89S52.

Câu số 10-4: Hãy cho biết hoạt động truyền dữ liệu mod 1 của vi điều khiển AT89S52.

Câu số 10-5: Hãy cho biết tên các thanh ghi liên quan đến truyền dữ liệu SPI của AT89S8252.

2. Câu hỏi mở rộng

Câu số 10-5: Hãy tìm hiểu so sánh hai vi điều khiển AT89S52 và AT89S8252.

3. Câu hỏi trắc nghiệm

Câu 10-1: ***Trong*** AT89S52 thì các thanh ghi có liên quan đến truyền dữ liệu:

(a) TH0, TL0, SBUF, SCON

(b) SCON, SBUF

(c) TH0, TL0, TCON, TMOD, TH1, TL1

(d) TH1, TL1, TCON, TMOD, SCON, SBUF

Câu 10-2: Trong AT89S52 thì có bao nhiêu mod truyền dữ liệu:

(a) 2 (b) 3

(c) 4 (d) 5

Câu 10-3: Bit lựa cho mod truyền dữ liệu trong AT89S52 là:

(a) M1, M0 (b) SM1, SM0

(c) SM2, SM1 (d) SM2, SM1, SM0

Câu 10-4: Trong AT89S52 thì timer nào không thể sử dụng để tạo tốc độ baud:

(a) T1 và T0 (b) T1 và T2

(c) T2 và T0 (d) T0

Câu 10-5: Trong AT89S52 thì bit nào lên 1 khi phát xong một kí tự:

(a) RB8 (b) TB8

(c) TI (d) RI

Câu 10-6: Trong AT89S52 thì bit nào lên 1 khi nhận xong một kí tự:

(a) RB8 (b) TB8

(c) TI (d) RI

Câu 10-7: Trong AT89S52 thì ngắt truyền dữ liệu có số thứ tự là:

(a) 4 (b) 5

(c) 6 (d) 2

Câu 10-10: Trong AT89S52 thì ngắt truyền dữ liệu có vector địa chỉ là:

(a) 0003H (b) 0023H

(c) 000BH (d) 002BH

Câu 10-11: Trong AT89S52 thì giá trị nào sẽ thiết lập tốc độ 4800 baud:

(a) TH1 = -3 (b) TH1 = -6

(c) TL1 = -3 (d) TL1 = -6

Câu 10-12: Trong AT89S52 thì timer tạo tốc độ baud hoạt động ở mod:

(a) 2 (b) 3

(c) 1 (d) 0

Câu 10-13: Khi truyền dữ liệu thì bit nào sẽ truyền đi đầu tiên:

(a) Bit dữ liệu D0 (b) Bit Stop

(c) D7 (d) Bit Start

Câu 10-14: Khi truyền dữ liệu thì bit nào sẽ truyền sau cùng:

(a) Bit dữ liệu D0 (b) Bit Stop

(c) D7 (d) Bit Start

Câu 10-15: Khi truyền dữ liệu thì trạng thái của bit start và stop là:

(a) Cả 2 đều ở mức 0 (b) Cả 2 đều ở mức 1

(c) Start ở mức 0, stop ở mức 1 (d) Start ở mức 1, stop ở mức 0

4. Bài tập

Bài tập 10-1: Một hệ thống hai vi điều khiển AT89S52: vi điều khiển A giao tiếp với 24 led đơn, vi điều khiển B giao tiếp với 24 led đơn dùng port0, 1, 2. Hai vi điều khiển giao tiếp truyền dữ liệu nối tiếp. Hãy viết chương trình điều khiển 48 led sáng dần, tắt dần do vi điều khiển A thực hiện, gởi sang vi điều khiển B để hiển thị.

Bài tập 10-2: Một hệ thống hai vi điều khiển AT89S52: vi điều khiển A giao tiếp với bàn phím ma trận 8x8, vi điều khiển B giao tiếp với hai led 7 đoạn anode chung port0, port1. Hai vi điều khiển giao tiếp truyền dữ liệu nối tiếp. Hãy viết chương trình điều khiển khi nhấn phím nào thì hai led sáng đúng mã phím đó.

TÀI LIỆU THAM KHẢO

[1]. AVTAR SINGH – WALTER A TRIEBEL, "The 8088 Microprocessor – Programming, interfacing, software, hardware, and Applications", Prentice Hall International Editions.

[2]. DOUGLAS V. HALL, "Microprocessor and Interfacing Programming, and hardware", McGraw – Hill International Editions.

[3]. John Uffenbeck, "The 8088/8086 family : Designing, programming and interfacing", Prentice Hall, 1987

[4]. James L. Antonakos, "The 68000 Microprocessor: hardware and software principles and applications", Prentice Hall fifth edition 2004.

[5]. Jack L. Davies, "The Innovative 80x86 – Volume I: the 80286 Microprocessor, architecture", Prentice Hall.

[6]. Jack L. Davies, "Z80 Family CPU user manual", www.zilog.com.

[7]. MetaLink Corporation Chandler – Arizona, "8051 Cross Assembler User's Manual", 1996

[8]. "MCS51 Microcontroller Family User's Manual", 1994

Giáo trình
VI XỬ LÝ

Nguyễn Đình Phú, Trương Ngọc Anh

NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH

Khu phố 6, Phường Linh Trung, Quận Thủ Đức, TPHCM

Số 3 Công trường Quốc tế, Quận 3, TP. HCM

ĐT: 38 239 172 - 38 239 170

Fax: 38 239 172 – E-mail: vnuhp@vnuhcm.edu.vn

***

Chịu trách nhiệm xuất bản

TS HUỲNH BÁ LÂN

Tổ chức bản thảo và chịu trách nhiệm về tác quyền

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TPHCM

Biên tập

NGUYỄN ĐỨC MAI LÂM

Sửa bản in

THUỲ DƯƠNG

Thiết kế bìa

HƯNG PHÚ

GT.01.KTh(V) 155-2012/CXB/543-08
ĐHQG.HCM-13

KTh.GT90. -13(T)

In 300 cuốn khổ 16 x 24cm, tại Công ty TNHH In và Bao bì Hưng Phú. Số đăng ký kế hoạch xuất bản: 155-2012/CXB/543-08/ĐHQGTPHCM. Quyết định xuất bản số: 23/QĐ-ĐHQGTPHCM cấp ngày 31/1/2013 của Nhà xuất bản ĐHQGTPHCM. In xong và nộp lưu chiểu Quí I, 2013.