

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC THÁI NGUYÊN

Võ Quang Vinh - Dương Quốc Hưng

GIÁO TRÌNH

HỆ VI ĐIỀU

UYÊN
LIỆU



NHÀ XUẤT BẢN KHOA HỌC

KHIỂN

C VÀ KỸ THUẬT

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC THÁI NGUYÊN**

VÕ QUANG VINH - DƯƠNG QUỐC HƯNG

GIÁO TRÌNH HỆ VI ĐIỀU KHIỂN



**NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
HÀ NỘI - 2010**

MỤC LỤC

LỜI NÓI ĐẦU	7
CHƯƠNG I: CÁC KHÁI NIỆM CƠ BẢN VÀ CẤU TRÚC CHUNG	9
Phần lý thuyết	9
1.1. GIỚI THIỆU CHUNG VỀ VI XỬ LÝ	9
1.1.1. Tổng quan	9
1.1.2. Đặc điểm chung của bộ vi xử lý	12
1.2. CẤU TRÚC CHUNG HỆ VI XỬ LÝ VÀ TỔ CHỨC BÊN TRONG CPU	14
1.2.1. Cấu trúc chung của một hệ vi xử lý	14
1.2.2. Cấu trúc chung của CPU	17
1.3. CẤU TRÚC DỮ LIỆU TRONG HỆ VI XỬ LÝ	21
1.3.1. Các hệ đếm dùng trong hệ vi xử lý	21
1.3.2. Biểu diễn các số và ký tự theo mã hệ hai	22
1.3.3. Các phép toán số học với mã hệ hai	29
Phần bài tập	33
CHƯƠNG II: BỘ VI XỬ LÝ INTEL 8086	35
Phần lý thuyết	35
2.1. CẤU TRÚC CỦA BỘ VI XỬ LÝ 8086	35
2.1.1. Tổng quan	35
2.1.2. Cấu trúc bên trong và hoạt động của bộ vi xử lý 8086	36
2.1.3. Mô tả chức năng các chân của vi xử lý 8086	44
2.1.4. Tổ chức bộ nhớ của vi xử lý 8086	50
2.2. CÁC CHẾ ĐỘ ĐỊA CHỈ CỦA BỘ VI XỬ LÝ 8086	51
2.2.1. Chế độ địa chỉ thanh ghi	52
2.2.2. Chế độ địa chỉ tức thì	52
2.2.3. Chế độ địa chỉ trực tiếp	53
2.2.4. Chế độ địa chỉ gián tiếp qua thanh ghi	53
2.2.5. Chế độ địa chỉ tương đối cơ sở	53
2.2.6. Chế độ địa chỉ tương đối chỉ số	54
2.2.7. Chế độ địa chỉ tương đối chỉ số cơ sở	54

2.3. TẬP LỆNH CỦA VI XỬ LÝ 8086	55
2.3.1. Giới thiệu chung	55
2.3.2. Tập lệnh của vi xử lý 8086	56
2.4. CÁC MẠCH PHỤ TRỢ	88
2.4.1. Mạch tạo xung nhịp	88
2.4.2. Mạch chốt và đệm Bus	90
2.4.3. Các bộ nhớ	93
2.4.4. Mạch giải mã địa chỉ	96
2.5. BIỂU ĐỒ THỜI GIAN ĐỌC/GHI CỦA VI XỬ LÝ	98
2.5.1. Xung nhịp và chu kỳ máy	98
2.5.2. Chu kỳ đọc/ghi của vi xử lý 8086	99
2.6. CÁC CHẾ ĐỘ LÀM VIỆC CỦA VI XỬ LÝ	102
2.6.1. Chế độ khởi động	102
2.6.2. Chế độ đợi	102
2.6.3. Chế độ treo	103
2.6.4. Chế độ ngắt	103
2.6.5. Chế độ dừng	104
2.7. MỘT SỐ VÍ DỤ	104
2.8. LẬP TRÌNH HỢP NGỮ CHO VI XỬ LÝ 8086/8088	108
2.8.1. Giới thiệu chung về hợp ngữ	108
2.8.2. Cú pháp của chương trình hợp ngữ	109
2.8.3. Khuôn dạng của chương trình hợp ngữ	113
2.8.4. Cấu trúc lập trình cơ bản bằng hợp ngữ	119
Phản bài tập	125
CHƯƠNG III: HỌ VI ĐIỀU KHIỂN 8 BIT 8051	129
Phần lý thuyết	129
3.1. CẤU TRÚC PHẦN CỨNG HỌ VI ĐIỀU KHIỂN 8051	129
3.1.1. Giới thiệu chung	129
3.1.2. Sơ đồ cấu trúc của họ vi điều khiển 8051	132
3.1.3. Mô tả chức năng các chân của 8051	133
3.1.4. Tổ chức bộ nhớ 8051	137
3.1.5. Hoạt động RESET	145

3.2. CÁC CHẾ ĐỘ ĐỊA CHỈ CỦA 8051	146
3.2.1. Kiểu định địa chỉ dùng thanh ghi (Register Addressing)	146
3.2.2. Kiểu định địa chỉ trực tiếp (Direct Addressing)	147
3.2.3. Kiểu định địa chỉ gián tiếp (Indirect Addressing)	147
3.2.4. Định địa chỉ tức thời (Immediate Addressing)	147
3.2.5. Định địa chỉ tương đối	147
3.2.6. Định địa chỉ tuyệt đối	148
3.2.7. Định địa chỉ dài (Long Addressing)	148
3.2.8. Định địa chỉ chỉ số (Index Addressing)	148
3.3. TẬP LỆNH CỦA 8051	149
3.3.1. Nhóm lệnh số học	150
3.3.2. Nhóm lệnh logic	153
3.3.3. Nhóm lệnh di chuyển dữ liệu	157
3.3.4. Nhóm lệnh xử lý bit	159
3.3.5. Nhóm lệnh rẽ nhánh	161
3.4. LẬP TRÌNH HỢP NGỮ CHO VI ĐIỀU KHIỂN 8051	166
3.4.1. Trình dịch hợp ngữ	166
3.4.2. Khuôn dạng của chương trình hợp ngữ	167
3.4.3. Kiểu dữ liệu và chỉ dẫn	167
3.4.4. Lập trình vào/ra các cổng	169
3.4.5. Bộ đếm/định thời của 8051 và lập trình	174
3.4.6. Lập trình truyền thông nối tiếp cho 8051	188
3.4.7. Lập trình ngắt 8051	203
Phần bài tập	221
CHƯƠNG IV: GHÉP NỐI VÀ TRAO ĐỔI DỮ LIỆU TRONG HỆ VI XỬ LÝ	225
Phần lý thuyết	225
4.1. GHÉP NỐI VI XỬ LÝ/VI ĐIỀU KHIỂN VỚI BỘ NHỚ NGOÀI	225
4.1.1. Một số chân của 8051 phục vụ cho ghép nối bộ nhớ ngoài	226
4.1.2. Ghép nối vi điều khiển 8051 với bộ nhớ chương trình ngoài	228

4.1.3. Ghép nối vi điều khiển 8051 với bộ nhớ dữ liệu ngoài	229
4.2. GHÉP NỐI SONG SONG 8255A VÀ LẬP TRÌNH	235
4.2.1. Tổng quan	235
4.2.2. Ghép nối 8051 với 8255A	235
4.3. PHỐI GHÉP VI ĐIỀU KHIỂN VỚI THẺ GIỚI THỰC 1: Phối ghép 8051 với màn hình tinh thể lỏng LCD, LED7, chuyển đổi tương tự - số ADC và cảm biến nhiệt	247
4.3.1. Phối ghép vi điều khiển với màn tinh thể lỏng LCD	247
4.3.2. Phối ghép vi điều khiển với LED 7 thanh	258
4.3.3. Ghép nối ADC với 8051 và lập trình	270
4.3.4. Nối ghép 8051 với cảm biến nhiệt	279
4.4. PHỐI GHÉP VI ĐIỀU KHIỂN VỚI THẺ GIỚI THỰC 2: Phối ghép 8051 với động cơ bước, bàn phím và bộ biến đổi số - tương tự DAC	283
4.4.1. Nối ghép 8051 với động cơ bước	283
4.4.2. Nối ghép 8051 với bàn phím	298
4.4.3. Phối ghép một DAC với 8051	306
Phần bài tập	309
PHỤ LỤC 1: DANH MỤC CÁC TỪ VIẾT TẮT	318
PHỤ LỤC 2: TẬP LỆNH CỦA VI ĐIỀU KHIỂN HỌ 8051	320
TÀI LIỆU THAM KHẢO	329

LỜI NÓI ĐẦU

Sự ra đời và phát triển nhanh chóng của kỹ thuật vi điện tử mà đặc trưng là kỹ thuật vi xử lý đã tạo ra một bước ngoặt quan trọng trong sự phát triển của khoa học tính toán, điều khiển và xử lý thông tin. Kỹ thuật vi xử lý đóng một vai trò rất quan trọng trong tất cả các lĩnh vực của cuộc sống và khoa học - kỹ thuật, đặc biệt là lĩnh vực tin học và tự động hóa.

Từ các bộ vi xử lý ban đầu chỉ là các bộ xử lý trung tâm trong một hệ thống, không thể hoạt động nếu thiếu các bộ phận như RAM, ROM, bo mạch chủ... các hãng đã phát triển các bộ vi xử lý này lên thành các bộ vi điều khiển để phục vụ các mục đích riêng biệt, khác nhau trong công nghiệp. Một bộ vi điều khiển là một hệ vi xử lý được tổ chức trong một chip bao gồm một bộ vi xử lý (microprocessor), bộ nhớ chương trình (ROM), bộ nhớ dữ liệu (RAM)... Hiện nay, các bộ vi điều khiển được sử dụng rất rộng rãi và ngày càng được chuẩn hóa để có thể sử dụng nhiều hơn nữa trong các ngành công nghiệp, có mặt trong nhiều máy móc, trong các hàng tiêu dùng.

*Được sự động viên, khích lệ của các thầy giáo, cô giáo trong Khoa Điện Trường Đại học Kỹ thuật Công nghiệp - Đại học Thái Nguyên, chúng tôi đã mạnh dạn viết cuốn giáo trình **Hệ Vi điều khiển**. Giáo trình nhằm phục vụ cho sinh viên các chuyên ngành Điện và Điện tử, các bạn đọc giả quan tâm đến lĩnh vực Vi xử lý và Vi điều khiển.*

Giáo trình không đi quá sâu vào diễn giải lý thuyết. Chủ yếu đưa ra những kiến thức cơ bản và những ví dụ ứng dụng về 2 Chip tiêu biểu đó là: Vi xử lý 8086 và vi điều khiển 8051. Nội dung được chia làm 4 chương:

***Chương I:** Bao gồm các kiến thức tổng quan về vi xử lý như: Cấu trúc chung của một bộ vi xử lý, một hệ vi xử lý. Các hệ đếm và*

biểu diễn thông tin trong các hệ vi xử lý - vi điều khiển, các phép toán thực hiện trên số nhị phân.

Chương II: Trình bày về cấu trúc của bộ vi xử lý 8086 - thuộc họ vi xử lý 80x86 của Intel và cơ bản về lập trình hợp ngữ trên họ vi xử lý này, các nội dung chính của chương bao gồm: Cấu trúc phần cứng, cách quản lý bộ nhớ, các chế độ địa chỉ, tập lệnh và các vi mạch phụ trợ như mạch tạo xung 8284, vi mạch chốt, giải mã, các bộ nhớ để tạo nên một hệ vi xử lý hoàn chỉnh; cách lập trình hợp ngữ dựa trên tập lệnh của bộ vi xử lý để tạo ra chương trình dạng *.EXE hoặc *.COM.

Chương III: Trình bày về hệ vi điều khiển OnChip họ 8051, bao gồm: Cấu trúc tổ chức phần cứng, bản đồ bộ nhớ, các chế độ địa chỉ và tập lệnh; lập trình hợp ngữ với những bài toán cơ bản: Nhập xuất dữ liệu với cổng, định thời, truyền thông nối tiếp, ngắt, ...

Chương IV: Trình bày về ghép nối vi điều khiển với bộ nhớ, thiết bị nhập xuất. Trong chương này, tác giả cũng giới thiệu về vi mạch cổng song song lập trình được PPI 8255A – một vi mạch cổng được sử dụng trong rất nhiều bài toán ghép nối khác nhau. Phần cuối của chương là các sơ đồ ghép nối tiêu biểu như: Ghép nối vi điều khiển với bộ nhớ, LCD, ADC, DAC, Keyboard, step motor, ...

Trong quá trình biên soạn chắc chắn không thể tránh khỏi những thiếu sót. Rất mong nhận được sự đóng góp ý kiến của đông đảo bạn đọc. Mọi sự góp ý xin gửi về: Bộ môn Tự động hoá - Khoa Điện - Trường Đại học Kỹ thuật Công nghiệp Thái Nguyên.

Xin chân thành cảm ơn!

CÁC TÁC GIẢ

CHƯƠNG I

CÁC KHÁI NIỆM CƠ BẢN VÀ CẤU TRÚC CHUNG

• Phần lý thuyết

** Mục đích của chương:*

Đưa ra những khái niệm cơ bản về vi xử lý, hệ vi xử lý, lịch sử phát triển của vi xử lý. Giúp sinh viên có những cái nhìn ban đầu về vi xử lý. Nắm được cấu trúc chung cũng như những đặc điểm chung về bộ vi xử lý, hệ vi xử lý.

Nhắc lại những kiến thức về hệ đếm và cách biểu diễn thông tin trong hệ vi xử lý, các phép toán với số hệ 2.

** Tóm tắt nội dung:*

Nội dung bao gồm các kiến thức tổng quan về vi xử lý như: Cấu trúc chung của một bộ vi xử lý, một hệ vi xử lý. Các hệ đếm và biểu diễn thông tin trong các hệ vi xử lý – vi điều khiển, các phép toán thực hiện trên số nhị phân.

1.1. GIỚI THIỆU CHUNG VỀ VI XỬ LÝ

1.1.1. TỔNG QUAN

1.1.1.1. Khái niệm

Vi xử lý là một vi mạch tích hợp cỡ lớn hoặc cực lớn với chức năng chính là xử lý dữ liệu. Để làm được điều này chúng phải có các mạch điện tử logic cho việc xử lý và điều khiển dữ liệu. Các mạch

logic sẽ di chuyển dữ liệu từ nơi này sang nơi khác và thực hiện các phép toán trên dữ liệu. Để vi xử lý điều khiển các mạch logic thì cần phải có chương trình. Chương trình là tập hợp các lệnh để xử lý dữ liệu, các lệnh đã được lưu trữ trong bộ nhớ. Công việc thực hiện lệnh bao gồm các bước như sau: Đón lệnh từ bộ nhớ, sau đó các mạch logic điều khiển sẽ giải mã lệnh và thực hiện lệnh sau khi giải mã.

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. Các mạch điện dùng để lưu trữ lệnh để vi xử lý xử lý được gọi là bộ nhớ (M), các mạch điện giao tiếp để di chuyển dữ liệu từ bên ngoài vào bên trong vi xử lý và xuất dữ liệu từ bên trong vi xử lý ra ngoài được gọi là các thiết bị ghép nối vào ra I/O.

1.1.1.2. Lịch sử phát triển của các bộ vi xử lý

Vi xử lý là sự kết hợp của hai kỹ thuật công nghệ quan trọng nhất: Đó là máy tính dùng kỹ thuật số (Digital Computer) và các mạch vi điện tử. Hai công nghệ này kết hợp lại với nhau vào năm 1970, sau đó các nhà nghiên cứu đã chế tạo ra vi xử lý (Microprocessor). Dựa vào lịch sử phát triển của các bộ vi xử lý ta có thể chia ra làm bốn thế hệ như sau:

➤ Thế hệ thứ nhất (1971 – 1973)

Năm 1971, Intel cho ra đời bộ vi xử lý đầu tiên là 4004 (4 bit số liệu, 12 bit địa chỉ), sau đó Intel và các nhà sản xuất khác cũng lần lượt cho ra đời các bộ vi xử lý khác: 4040 (4 bit) và 8008 (8 bit) của Intel. PPS – 4 (4 bit) của Rockwell International, IPM – 16 (bit) của National Semiconductor.

Đặc điểm chung của các bộ vi xử lý thế hệ này là:

- Thông thường là 4 bit dữ liệu (cũng có thể dài hơn).
- Công nghệ chế tạo PMOS với đặc điểm mật độ phần tử nhỏ, tốc độ chậm, giá rẻ và chỉ có khả năng đưa ra dòng tải nhỏ.

- Tốc độ thực hiện lệnh: $10 - 60 \mu\text{s}/\text{lệnh}$. Với tần số đồng hồ $f_{\text{clk}} = 0,1 - 0,8 \text{ MHz}$.
- Tập lệnh đơn giản và phải cần nhiều mạch phụ trợ mới tạo nên một vi xử lý hoàn chỉnh.

➤ **Thế hệ thứ hai (1974 – 1977)**

Các bộ vi xử lý đại diện cho thế hệ này là vi xử lý 8 bit 6502 của MOS Technology, 6800 và 6809 của Motorola, 8080 của Intel và đặc biệt là bộ vi xử lý Z80 của Zilog. Các bộ vi xử lý này có tập lệnh phong phú hơn và thường có khả năng phân biệt địa chỉ bộ nhớ với dung lượng đến 64 KB. Có một số bộ vi xử lý còn có khả năng phân biệt được 256 địa chỉ cho các thiết bị ngoại vi (họ Intel và Zilog). Chúng đã được sử dụng rộng rãi trong công nghiệp và nhất là để tạo ra các máy tính 8 bit nổi tiếng một thời như Apple II Commodore 64. Tất cả các vi xử lý thời kỳ này đều được chế tạo bằng công nghệ NMOS (mật độ phân tử trên một đơn vị diện tích cao hơn so với công nghệ PMOS) hoặc CMOS (tiết kiệm điện năng tiêu thụ) và cho phép đạt được tốc độ từ $1 - 8 \mu\text{s}/\text{lệnh}$ với tần số đồng hồ $1 - 5 \text{ MHz}$.

➤ **Thế hệ thứ ba (1978 – 1982)**

Đại diện là các bộ vi xử lý 16 bit 8086/80186/80286 của Intel hoặc 68000/68010 của Motorola. Các bộ vi xử lý này có độ dài 16 bit và tập lệnh phong phú với lệnh nhân, chia và các thao tác với chuỗi ký tự. Khả năng phân biệt địa chỉ cho bộ nhớ hoặc cho thiết bị ngoại vi của các vi xử lý thế hệ này cũng lớn hơn (từ 1 – 16 MB cho bộ nhớ và tới 64 KB cho địa chỉ thiết bị ngoại vi đối với họ Intel). Phần lớn các bộ vi xử lý thế hệ này đều đạt tốc độ từ $0,1 - 1 \mu\text{s}/\text{lệnh}$ với tần số đồng hồ $f_{\text{clk}} = 5 - 10 \text{ MHz}$.

➤ **Thế hệ thứ tư (1983 đến nay)**

Các bộ vi xử lý đại diện cho thế hệ này là các vi xử lý 32 bit 80386/80486 và 64 bit của Pentium của Intel, các vi xử lý 32 bit

68020/68040/68060 của Motorola. Đặc điểm của vi xử lý thế hệ này là bus địa chỉ đều là 32 bit (phân biệt 4GB bộ nhớ) và có khả năng làm việc với bộ nhớ ảo. Bảng 1.1 trình bày thông số của các bộ vi xử lý của Intel.

Bảng 1.1. Thông số của các bộ vi xử lý

Thông số	8086	8088	80286
Năm sản xuất	6/1978	1979	2/1982
f_{clkmax} (xung nhịp đồng hồ)	10 MHz	10 MHz	20 MHz
MIPS (triệu lệnh/s)	0,33	0,33	1,2
Số tranzitor	29.000	29.000	134.000
Bus số liệu	16 bit	8 bit	16 bit
Bus địa chỉ	20 bit	20 bit	24 bit
Khả năng địa chỉ	1 MB	1 MB	16 MB
Số chân	40	40	68
Chế độ bộ nhớ ảo	Không	Không	Có
Có bộ quản lý bộ nhớ bên trong	Không	Không	Có

1.1.2. ĐẶC ĐIỂM CHUNG CỦA BỘ VI XỬ LÝ

Mỗi bộ vi xử lý đều có những đặc điểm và chức năng riêng, tuy nhiên chúng đều có những đặc điểm chung sau:

Chiều dài bus dữ liệu (độ lớn của mạch vi xử lý): Được đánh giá bằng số bit lớn nhất mà vi xử lý có thể xử lý song song đồng thời trong một lần thao tác. Ví dụ: Mạch vi xử lý 16 bit thì có nghĩa vi xử lý có thể xử lý song song một lần 16 bit. Các vi xử lý có chiều dài bus 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 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. Để tăng tốc độ làm việc của vi xử lý là mục tiêu hàng đầu của các nhà chế tạo.

Chiều dài bus địa chỉ (khả năng truy xuất bộ nhớ) - Dung lượng bộ nhớ mà vi xử lý có thể truy xuất. Vi xử lý 4004 có 14 đường địa chỉ nên có thể truy xuất được $2^{14} = 16.384$ ô nhớ, vi xử lý 8 bit có 16 đường địa chỉ nên có thể truy xuất được $2^{16} = 65.536$ ô nhớ, vi xử lý 16 bit có 20 đường địa chỉ nên có thể truy xuất được $2^{20} = 1.048.576 = 1\text{MB}$ ô nhớ, vi xử lý 32 bit như 386 hay 68020 có thể truy xuất 4GB 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.

Số chân và các hình dạng ngoài của vi xử lý: Các bộ vi xử lý 8088/8086 được đóng trong vỏ hai hàng chân, đó là loại vỏ bằng Plastic có các chân hướng xuống dưới theo hai hàng song song hai bên. Các bộ vi xử lý lớn hơn có 64 chân hay nhiều hơn nữa có vỏ bọc với các chân ra tạo thành mảng sắp xếp trên cả bốn phía. Thông thường các vi xử lý có số chân càng nhiều thì càng hoàn hảo. Tuy vậy, trong thực tế người ta có thể dùng phương pháp dồn chân, nghĩa là một chân tín hiệu có thể làm hai chức năng và để phân biệt hai chức năng ấy người sử dụng phải lập chương trình.

Xung nhịp hệ thống (System clock): Dùng để đồng bộ các thao tác bên trong và bên ngoài của vi xử lý bằng cách phát ra các xung nhịp thời gian theo các khoảng cách cố định.

Tần số xung clock quyết định đến tốc độ làm việc của vi xử lý, vi xử lý có tốc độ làm việc càng lớn thì khả năng xử lý lệnh càng nhanh. Tuy nhiên, tần số xung nhịp không phải là yếu tố duy nhất. Tốc độ vi xử lý còn phụ thuộc vào cách thức xử lý thông tin trong cấu trúc của vi xử lý. Tần số xung clock làm việc của các vi xử lý được cho bởi các nhà chế tạo.

Điện áp hoạt động (nguồn nuôi): Với công nghệ PMOS các bộ vi xử lý cần nguồn nuôi với nhiều điện áp hoạt động khác nhau. Ví dụ: $\pm 5V$, $\pm 10V$. Còn với công nghệ NMOS thì chỉ cần nguồn nuôi duy nhất $+5V$. Đây là mức điện áp đủ cao để bù lại các sụt áp trong các mạch số, đồng thời cũng đủ nhỏ để tránh gây tạp âm. Các bộ vi xử lý ngày nay với công nghệ CMOS đã cho ra đời các loại vi xử lý 3, 3V, đòi hỏi dòng tiêu thụ chỉ bằng 60% so với chip 5V.

Các mạch phụ cận: Là các mạch làm chức năng giải mã chuyển đổi số liệu, mạch ghép nối logic, mạch đếm và định thời gian, mạch điều khiển bàn phím, hiển thị... Loại mạch này cũng được chế tạo theo hãng và thường tương ứng với mạch vi xử lý của hãng ấy.

Mạch phối ghép vào ra: Làm nhiệm vụ trao đổi số liệu vào và ra cho CPU. Mỗi hãng sẽ chế tạo mạch phối ghép đi kèm. Loại vi xử lý nào càng nhiều mạch phối ghép thì càng dễ sử dụng.

Phần mềm (chương trình): Mỗi một vi xử lý có một phần mềm được hãng trang bị hay còn gọi là tập lệnh. Người sử dụng căn cứ vào tập lệnh này để khai thác sử dụng và tạo thành các chương trình phục vụ cho các mục đích cụ thể.

1.2. CẤU TRÚC HỆ VI XỬ LÝ VÀ TỔ CHỨC BÊN TRONG CPU

1.2.1. CẤU TRÚC CHUNG CỦA MỘT HỆ VI XỬ LÝ

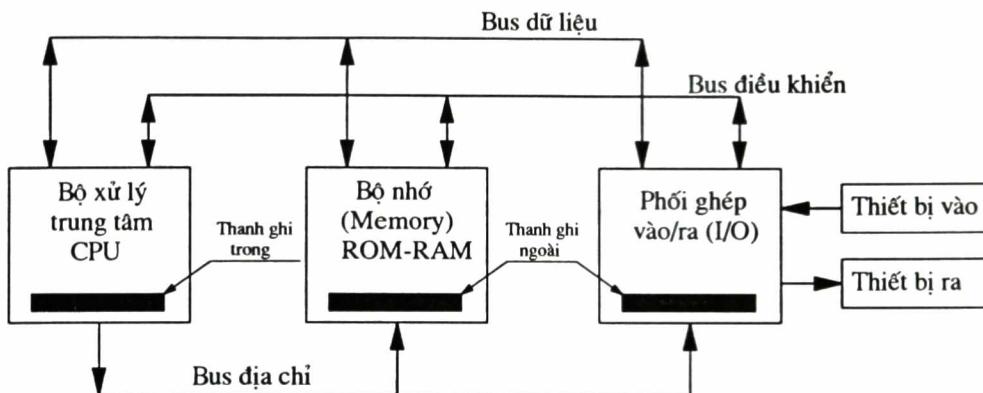
Trên đây ta đã thấy bộ vi xử lý là một thành phần rất cơ bản không thể thiếu được để tạo nên một máy vi tính. Trong thực tế, bộ vi xử lý còn phải kết hợp thêm các bộ phận điện tử khác như: bộ nhớ, các bộ phối ghép vào/ra để tạo nên một hệ vi xử lý hoàn chỉnh.

Hình 1.1 giới thiệu sơ đồ khối tổng quát của của một hệ vi xử lý.

Trong sơ đồ này ta thấy rõ các khối chức năng chính của một hệ vi xử lý bao gồm:

- Khối xử lý trung tâm (Central Processing Unit, CPU).

- Bộ nhớ bán dẫn (Memory, M).
- Khối phối ghép với các thiết bị ngoại vi (Input/Output, I/O).
- Các bus truyền thông tin.



Hình 1.1. Sơ đồ khối tổng quát hệ vi xử lý

1.2.1.1. Khối xử lý trung tâm, CPU

Khối xử lý trung tâm đóng vai trò chủ đạo trong hệ thống vi xử lý, đây là một mạch vi điện tử có độ tích hợp rất cao. Khi hoạt động nó đọc mã lệnh dưới dạng các bit 0 và 1 từ bộ nhớ. Sau đó, sẽ giải mã các lệnh này thành dãy các xung điều khiển ứng với các thao tác trong lệnh để điều khiển các khối khác thực hiện từng bước các thao tác đó. Để làm được điều này bên trong CPU có các thanh ghi cùng bộ tính toán số học và logic (ALU) để thao tác với dữ liệu. Ngoài ra, ở đây còn có hệ thống mạch điện tử rất phức tạp để giải mã lệnh và từ đó tạo ra các xung điều khiển cho toàn hệ.

1.2.1.2. Bộ nhớ bán dẫn (Memory)

Bộ nhớ bán dẫn hay còn gọi là bộ nhớ trong là một bộ phận rất quan trọng của hệ vi xử lý. Tại đây (trong ROM) chứa chương trình điều khiển hoạt động của toàn hệ, để khi bật điện lên thì CPU có thể lấy lệnh từ đây mà khởi động hệ thống. Một phần của chương trình điều khiển hệ thống, các chương trình ứng dụng, dữ liệu cùng các kết

quả của chương trình được để trong RAM. Các dữ liệu và chương trình muốn lưu trữ lâu dài sẽ được để ở bộ nhớ ngoài.

1.2.1.3. Khối phối ghép vào/ra (I/O)

Khối phối ghép vào/ra tạo ra khả năng giao tiếp giữa hệ vi xử lý với thế giới bên ngoài. Các thiết bị ngoại vi như bàn phím, chuột, màn hình, máy in, chuyển đổi số tương tự (D/A), chuyển đổi tương tự số (A/D) đều được liên hệ với vi xử lý thông qua bộ phận này. Bộ phận phối ghép cụ thể giữa bus hệ thống với thế giới bên ngoài thường được gọi là cổng. Như vậy, ta sẽ có cổng vào để lấy thông tin từ ngoài vào và cổng ra để đưa thông tin từ trong hệ ra ngoài. Tùy theo yêu cầu cụ thể mà các mạch cổng này có thể được xây dựng từ các mạch logic đơn giản hoặc từ các vi mạch chuyên dụng lập trình được.

1.2.1.4. Hệ thống Bus

Trên sơ đồ khối hệ vi xử lý ta thấy ba khối chức năng đầu liên hệ với nhau thông qua tập các đường dây để truyền tín hiệu gọi chung là bus hệ thống. Bus hệ thống bao gồm ba bus thành phần ứng với các tín hiệu địa chỉ, dữ liệu và điều khiển.

Bus địa chỉ: Thường có từ 16, 20, 24, 32 đường dây song song truyền tải thông tin của các bit địa chỉ. Khi đọc/ghi bộ nhớ CPU sẽ đưa ra trên bus này địa chỉ của ô nhớ liên quan. Khả năng phân biệt địa chỉ (số lượng địa chỉ cho ô nhớ mà CPU có khả năng phân biệt được) phụ thuộc vào số bit của bus địa chỉ. Ví dụ một CPU có số đường dây địa chỉ là $N = 16$ thì nó có khả năng địa chỉ hoá được $2^N = 2^{16} = 65536 = 64\text{KB}$ ô nhớ khác nhau ($1\text{K} = 2^{10} = 1024$). Trên sơ đồ khối ta dễ nhận ra tính một chiều của bus địa chỉ qua chiều của mũi tên. Chỉ có CPU mới có khả năng đưa ra địa chỉ trên bus địa chỉ.

Bus dữ liệu: Thường có từ 8, 16, 20, 24, 32 đến 64 đường dây tùy theo các bộ vi xử lý cụ thể. Số lượng đường dây này quyết định số bit dữ liệu mà CPU có khả năng xử lý một lúc. Chiều mũi tên trên bus chỉ ra rằng đây là bus 2 chiều.

Bus điều khiển: Thường gồm hàng chục đường dây tín hiệu khác nhau. Mỗi tín hiệu điều khiển có một chiều nhất định, vì khi hoạt động CPU đưa tín hiệu điều khiển tới các khối khác trong hệ. Đồng thời nó cũng nhận tín hiệu điều khiển từ các khối đó để phối hợp hoạt động toàn hệ, nên các tín hiệu này trên hình vẽ được thể hiện bằng mũi tên hai chiều.

Hoạt động của hệ vi xử lý trên cũng có thể được nhìn nhận theo một cách khác. Trong khi hoạt động và tại một thời điểm nhất định, về mặt chức năng mỗi khối trong hệ trên tương đương các thanh ghi trong (trong CPU) hoặc các thanh ghi ngoài (nằm trong bộ nhớ ROM, RAM và trong khối ghép nối I/O). Hoạt động của toàn hệ thực chất là sự phối hợp hoạt động của các thanh ghi trong và ngoài nói trên để thực hiện sự biến đổi dữ liệu hoặc sự trao đổi dữ liệu theo các yêu cầu đã định trước.

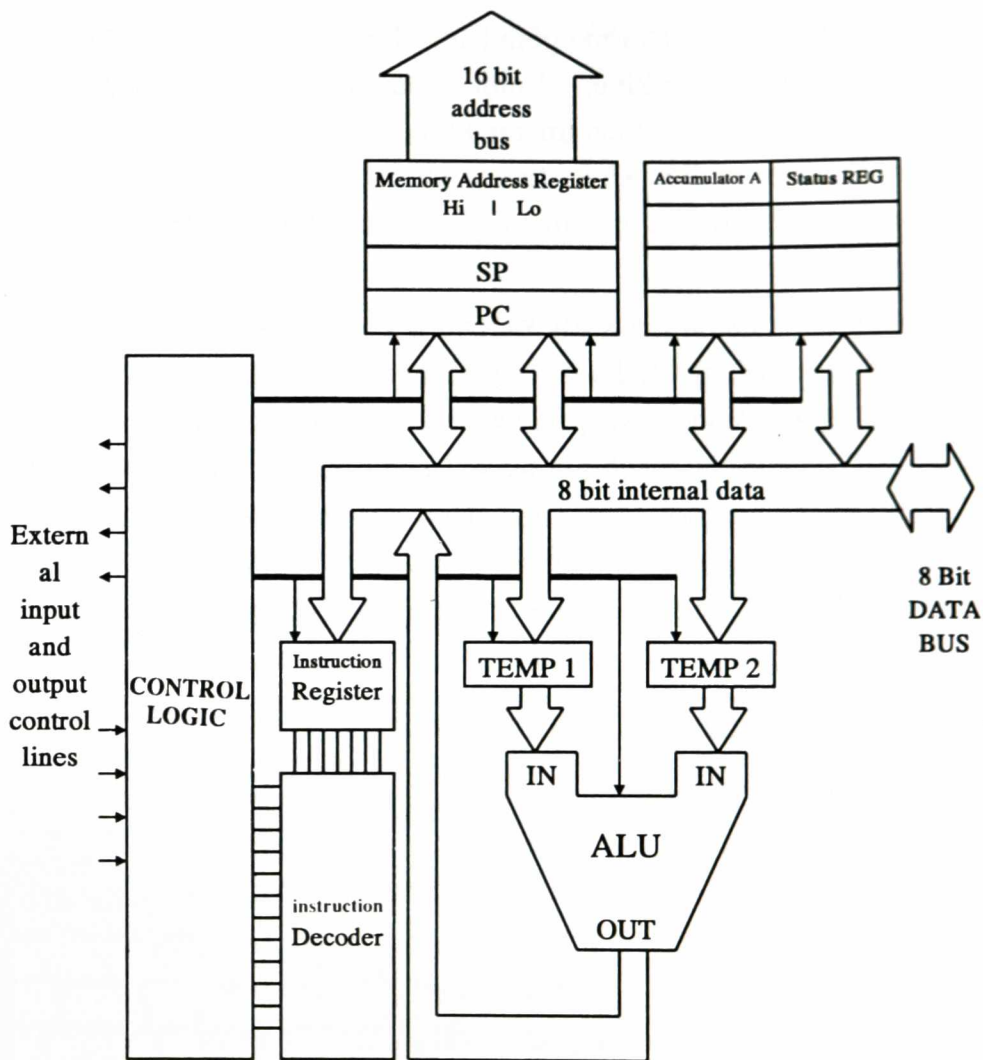
1.2.2. CẤU TRÚC CHUNG CỦA CPU

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.

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.

Hình 1.2 trình bày sơ đồ khối của vi xử lý 8 bit.

Nhìn vào sơ đồ khối trên ta có thể chia cấu trúc chung của một CPU ra làm ba phần chính: Khối ALU (Arithmetic Logic Unit), khối điều khiển (Control Logic), các thanh ghi (Register) và Bus dữ liệu.



Hình 1.2. Sơ đồ khối của vi xử lý 8 bit

1.2.2.1. Khối ALU

ALU là khối quan trọng nhất của vi xử lý, khối ALU chứa các mạch điện logic chuyên về xử lý dữ liệu. Khối ALU có hai ngõ vào có tên là “IN” chính là các ngõ vào dữ liệu cho ALU xử lý và một ngõ ra có tên là “OUT” chính 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 TEMP 1 và TEMP 2. 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. Sự kết nối này cho phép ALU có thể lấy bất kỳ dữ liệu nào trên bus dữ liệu bên trong vi xử lý. Ngõ ra OUT của ALU cho phép ALU có thể gửi kết quả 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.

Ví dụ: Khi ALU cộng hai dữ liệu thì một trong hai dữ liệu được chứa trong thanh ghi Accumulator, sau khi phép cộng được thực hiện bởi ALU thì kết quả sẽ gửi trở lại thanh ghi Accumulator 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 2 dữ liệu thì ALU sẽ xử lý 2 dữ liệu và dùng 2 ngõ vào “IN” để nhập dữ liệu, khi tăng một dữ liệu nào đó lên 1 đơn vị hay lấy bù một dữ liệu, khi đó ALU chỉ xử lý 1 dữ liệu và chỉ cần một ngõ vào “IN”.

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

Add (cộng)	Complement (lấy bù)	OR (Phép OR)
Subtract (trừ)	Shift right (dịch phải)	Increment (tăng)
AND(phép AND)	Shift left (dịch trái)	Decrement (giảm)
Exclusive OR (phép XOR – hoặc có loại trừ)		

Như vậy: Chức năng chính của khối ALU là làm thay đổi dữ liệu hay chuyển về xử lý dữ liệu nhưng không lưu trữ dữ liệu.

1.2.2.2. Các thanh ghi bên trong vi xử lý

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 khi

thực hiện các lệnh đặc biệt, các thanh ghi còn lại gọi là các thanh ghi thông dụng.

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 ta luôn sử dụng các thanh ghi này. Số lượng các thanh ghi thông dụng thay đổi tùy thuộc vào từng vi xử lý.

Các thanh ghi cơ bản luôn có trong một vi xử lý là thanh ghi tích lũy 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 SF (Status register – Flag register), các thanh ghi thông dụng, thanh ghi lệnh IR (Instruction register), thanh ghi địa chỉ AR (address register).

1.2.2.3. Khối điều khiển logic và khối giải mã lệnh

– Chức năng của khối giải mã lệnh (Instruction decoder) là nhận lệnh từ thanh ghi lệnh, sau đó giải mã để gửi 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. Khối điều khiển logic được xem là một vi xử lý nhỏ nằm trong một vi xử lý.

1.2.2.4. Bus dữ liệu bên trong vi xử lý (Internal data bus)

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 I/O.

1.3. CẤU TRÚC DỮ LIỆU TRONG HỆ VI XỬ LÝ

Một vấn đề quan trọng trong hệ vi xử lý là việc lưu trữ và xử lý các dữ liệu số. Thông thường chúng ta đã quen thuộc với các số trong hệ đếm cơ số 10, nhưng trong thực tế còn có nhiều hệ đếm với các cơ số khác nhau. Trong phần này chúng ta sẽ làm quen với cách biểu diễn số nhị phân, số Hex, các khái niệm bit, byte, word, sau đó là các phép toán cơ bản với mã hệ hai.

1.3.1. CÁC HỆ ĐẾM DÙNG TRONG HỆ VI XỬ LÝ

1.3.1.1. Hệ cơ số mười (Decimal – thập phân)

Trong cuộc sống hàng ngày ta dùng hệ cơ số mười, nói gọn là hệ mười (Decimal number system, viết tắt là hệ D) để biểu diễn các giá trị số. Chúng ta dùng tổ hợp các chữ số từ 0 đến 9 để biểu diễn các giá trị số. Đi kèm theo tập hợp đó có thể dùng thêm hậu tố D ở cuối để chỉ ra rằng đó là hệ mười.

1.3.1.2. Hệ cơ số hai (Binary – nhị phân)

Trong thế giới máy tính lại khác, để biểu diễn một giá trị số chúng ta dùng hệ cơ số hai hoặc nói gọn là hệ hai (Binary number system, viết tắt là hệ B). Sử dụng hai chữ số là 0 và 1 để biểu diễn (ứng với hai trạng thái có điện và không có điện của các mạch điện tử cấu tạo nên máy). Đặc điểm của hệ đếm cơ số hai là trong cùng một số có hai chữ số giống nhau thì chữ số bên trái có giá trị gấp hai lần chữ số bên phải.

Từ hệ đếm cơ số hai ta có các khái niệm sau:

- Bit: Mỗi chữ số trong hệ hai là một bit. Chữ số đầu tiên bên trái trong dãy các số hệ hai gọi là bit có nghĩa lớn nhất hay bit có trọng số lớn nhất (Most Significant Bit – MSB), còn bit cuối cùng bên phải trong dãy gọi là bit có nghĩa bé nhất hay bit có trọng số nhỏ nhất (Least Significant Bit – LSB).

- Byte: 1Byte = 8 Bit.
- Word: 1 Word = 2 Byte = 16 Bit.
- Double Word (từ kép): 1 Double Word = 32 Bit.

Ví dụ: 15D = 1111B

1.3.1.3. Hệ cơ số 16 (Hexa decimal – thập lục phân)

Nếu ta dùng hệ hai để biểu diễn các số có giá trị lớn ta sẽ gặp điều bất tiện là số hệ hai thu được quá dài. Ví dụ, để biểu diễn số 255 ta cần đến 8bit viết như sau: 255 = 11111111B

Trong thực tế, để viết kết quả biểu diễn của các số cho gọn người ta tìm cách nhóm bốn số hệ hai thành một số hệ 16 và sử dụng 16 chữ số cơ bản: 0 ÷ 9, A, B, C, D, E, F để biểu diễn.

Đặc điểm của hệ cơ số 16 là một chữ số ở cơ số 16 có hai chữ số giống nhau thì chữ số bên trái có giá trị gấp 16 lần chữ số bên phải.

Ví dụ: $(11001010)_2 = 2^7 + 2^6 + 2^3 + 2 = 128 + 64 + 8 + 2 = 202$

$$(11001010)_2 = (CA)_{16} = 12.16 + 10 = 202$$

Như vậy: Hệ cơ số 16 thực tế cũng là hệ hai nhưng được viết gọn hơn và điều quan trọng nhất là giúp cho người sử dụng ít bị nhầm.

1.3.1.4. Hệ mười mã hoá bằng hệ hai (Hệ BCD)

Thích hợp cho thiết bị đo có thêm phần hiển thị số ở đầu ra dùng các loại đèn hiện số khác nhau. Ở đây ta dùng bốn số hệ hai để mã hoá một số hệ mười có giá trị nằm trong khoảng từ 0, 1, ..., 9.

Ví dụ: Số 520 nếu biểu diễn theo kiểu số BCD thì được $(0101\ 0010\ 0000)_{BCD}$

1.3.2. BIỂU DIỄN CÁC SỐ VÀ KÝ TỰ THEO MÃ HỆ HAI

1.3.2.1. Biểu diễn số nguyên

Biểu diễn số nguyên không dấu

Nếu ta chỉ dùng số nguyên dương thì cách biểu diễn là đơn giản. Một số 8 bit có thể được sử dụng để biểu diễn các số từ 0 đến 255. Thật vậy:

$$\begin{aligned} 00000000 &= 0 \\ 10000000 &= 128 \\ 11111111 &= 255 \end{aligned}$$

Thông thường, nếu một chuỗi n bit của các số nhị phân $a_{n-1}a_{n-2} \dots a_1a_0$ sẽ được tính như một số nguyên A không dấu, mà giá trị của nó là:

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

Biểu diễn dấu – độ lớn (hay độ lớn có dấu)

Việc sử dụng các số nguyên không dấu là không đủ vì ta cần biểu diễn cả số âm cũng như các số nguyên dương. Vì vậy, để biểu diễn các số có dấu, bit MSB trong từ được dùng làm bit dấu. Nếu bit MSB là 0 thì số biểu diễn là số dương và nếu bằng 1 thì là số âm. Trong một từ n bit thì $n - 1$ bit cuối cùng giữ giá trị số nguyên.

Ví dụ: $(+20 = 00010100 \text{ hoặc } -20 = 10010100)$

Công thức tổng quát có thể được biểu diễn như sau:

$$A = \sum_{i=0}^{n-2} 2^i a_i \quad \text{Nếu } a_{n-1} = 0$$

$$A = -\sum_{i=0}^{n-2} 2^i a_i \quad \text{Nếu } a_{n-1} = 1$$

Biểu diễn số bù hai

Giống như cách biểu diễn dấu - độ lớn, cách biểu diễn số bù hai cũng sử dụng bit cao nhất (MSB) như là một bit dấu. Nhưng nó khác với cách biểu diễn dấu – độ lớn các bit còn lại sẽ có ý nghĩa khác đi.

Công thức tổng quát để biểu diễn một số bù hai như sau:

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Trong trường hợp các số nguyên dương thì $a_{n-1} = 0$ cho nên biểu thức $-2^{n-1} a_{n-1} = 0$. Do vậy, biểu thức trên được biểu diễn số bù hai cho cả số dương và số âm.

Bảng 1.2 so sánh cách biểu diễn số bù hai và biểu diễn dấu – độ lớn đối với các số nguyên 4 bit.

Bảng 1.2

Biểu diễn số thập phân	Biểu diễn dấu – độ lớn	Biểu diễn số bù hai
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
-0	1000	-
-1	1001	1111
-2	1010	1110
-3	1011	1101
-4	1100	1100
-5	1101	1011
-6	1110	1010
-7	1111	1001
-8	-	1000

Quan sát bảng trên ta thấy:

- Nếu ta dùng 4 bit để biểu diễn số thì ta thu được 16 tổ hợp có giá trị từ 0 đến 15 tức là chỉ biểu diễn được số dương.
- Với cách biểu diễn theo dấu độ lớn ta có thể biểu diễn được cả số dương và âm nằm trong khoảng $-7...-0 + 0...+7$.
- Với cách biểu diễn theo mã bù hai ta cũng biểu diễn được cả số dương và âm nằm trong khoảng $-8...0...+7$.

Biến đổi bit

Có trường hợp người ta lấy số nguyên n bit nhưng lại lưu nó vào trong m bit mà $m > n$. Trong trường hợp biểu diễn dấu độ lớn có thể thực hiện một cách dễ dàng bằng cách di chuyển bit dấu đến vị trí mới ở bên trái và điền thêm 0 vào đó.

Ví dụ:

+18 = 00010010	(dấu – độ lớn, 8-bit)
+18 = 00000000 00010010	(dấu – độ lớn, 16-bit)
-18 = 10010010	(dấu – độ lớn, 8-bit)
-18 = 10000000 00010010	(dấu – độ lớn, 16-bit)

Điều này không còn đúng nữa với số nguyên âm bù hai

Ví dụ:

+18 = 00010010	(số bù hai, 8-bit)
+18 = 00000000 00010010	(số bù hai, 16-bit)
-18 = 11101110	(số bù hai, 8-bit)
-32658 = 10000000 01101110	(số bù hai, 16-bit)

Quy tắc để biến đổi bit số bù hai là di chuyển bit dấu đến vị trí mới phía bên trái và điền thêm các bit có giá trị bằng bit dấu (cả vào vị trí cũ của bit dấu). Đối với các số dương thì điền thêm 0 với các số âm thì điền thêm 1. Như vậy, cách biểu diễn 18 theo mã bù hai như sau:

+18 = 00010010	(số bù hai, 8-bit)
+18 = 00000000 00010010	(số bù hai, 16-bit)
-18 = 11101110	(số bù hai, 8-bit)
-18 = 11111111 11101110	(số bù hai, 16-bit)

1.3.2.2. Biểu diễn số thực

Số thực A trong hệ nhị phân được biểu diễn như sau:

$$A = D_1D_2D_3...D_n, d_1d_2d_3...d_n = D, d$$

Trong đó:

$$D = D_1D_2D_3...D_n \quad \text{Là phần nguyên}$$

$$d = d_1d_2d_3...d_n \quad \text{Là phần thập phân}$$

Việc biểu diễn A dưới dạng nhị phân được thực hiện làm hai bước. Đầu tiên đổi phần nguyên ra dạng nhị phân theo kiểu thông thường, sau đó đổi tiếp phần thập phân.

Giả sử phần thập phân $0.d_1d_2d_3...d_n$ có dạng biểu diễn nhị phân là $b_1b_2b_3...b_m$.

Khi đó giá trị của nó trong hệ cơ số 10 được tính như sau:

$$\begin{aligned} d &= d_1d_2d_3...d_n = (b_1b_2b_3...b_m)_2 \\ &= (b_12^{-1} + b_22^{-2} + b_32^{-3} + ... + b_m2^{-m})_{10} \end{aligned}$$

1.3.2.3. Biểu diễn ký tự theo mã ASCII

Trong thực tế xử lý của máy tính ngoài các ký tự chữ cái còn có các ký tự đặc biệt và các ký tự số. Ký hiệu phổ biến nhất ngày nay là dùng mã ASCII (American Standard Code for Information Interchange) mã chuẩn của Mỹ dùng trong trao đổi thông tin. Việc dùng các ký tự để mã hoá thông tin theo bảng mã chuẩn sẽ cho chúng ta khả năng trao đổi thông tin với hầu hết các hệ vi xử lý khác, kể cả đối với các hệ vi xử lý tiên tiến (hình 1.3).

Khi tra các bảng này ta đọc mã của ký tự theo thứ tự cột – hàng.

Ví dụ: Mã ASCII của một số ký tự thường dùng trong khi lập trình.

<BEL> (Bell, chuông): 07H

<BS> (Back space, xoá lùi): 08H

<LF> (Line Feed, thêm dòng mới): 0AH

Hexa decimal	0	1	2	3	4	5	6	7
0	<NUL> 0	<DLE> 16	<SP> 32	0 48	@ 64	P 80	` 96	p 112
1	<SQH> 1	<DCL> 17	! 33	1 49	A 65	Q 81	a 97	q 113
2	<STX> 2	<DC2> 18	" 34	2 50	B 66	R 82	b 98	r 114
3	<ETX> 3	<DC3> 19	# 35	3 51	C 67	S 83	c 99	s 115
4	<EQT> 4	<DC4> 20	\$ 36	4 52	D 68	T 84	d 100	t 116
5	<ENQ> 5	<NAK> 21	% 37	5 53	E 69	U 85	e 101	u 117
6	<ACK> 6	<SYN> 22	& 38	6 54	F 70	V 86	f 102	v 118
7	<BEL> 7	<ETB> 23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	<CAN> 24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	 25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	<SUB> 26	* 42	: 58	J 74	Z 90	j 106	z 122
B	<VT> 11	<ESC> 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	<EF> 12	<FS> 28	, 44	< 60	L 76	\ 92	l 108	 124
D	<CR> 13	<GS> 29	- 45	= 61	M 77	] 93	m 109	} 125
E	<SO> 14	<RSI> 30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	<SI> 15	<US> 31	/, 47	? 63	O 79	- 95	o 111	 127

Hình 1.3. Bảng mã ASCII tiêu chuẩn

Khi xem xét bảng mã ta có thể rút ra những nhận xét sơ bộ sau:

- Hai cột đầu tiên của bảng mã được dùng cho các ký tự điều khiển.
 - Cột 2 dành riêng cho các ký tự ngăn cách.
 - Các ký tự số nằm ở cột 3.
 - Các chữ cái hoa nằm ở cột 4, 5. Các chữ cái thường ở cột 6, 7.
- Khoảng cách giữa chữ thường và chữ hoa cùng tên là 20H.

Quan hệ giữa mã ASCII với số BCD

Trong khi làm toán với các số BCD ta thường kết hợp hai số BCD thành một Byte, dạng số BCD viết theo kiểu này được gọi là dạng BCD chuẩn hay còn gọi là dạng BCD gói. Khi lưu trữ, khi hiển thị hoặc khi truyền giữa các thiết bị các giá trị số 0...9 thực chất ta làm việc với mã ASCII của các số đó tức là các số 30H...39H. Như vậy, ta thấy trong 1 byte mã ASCII biểu diễn các số 0...9 có 4 bit thấp ứng với mã BCD của chính các số đó, 4 bit cao bao giờ cũng ứng với mã BCD của số 3. Nếu trong byte mã ASCII này ta thay 4 bit cao bằng 0H thì ta thu được số BCD không gói. Nói cách khác, số BCD không gói là số dài 1 byte, trong đó có 4 bit cao bằng 0; 4 bit thấp là số BCD chuẩn mã hoá số cần biểu diễn.

Ví dụ:

Mã ASCII của	5	là 0011 0101 = 35H
Mã ASCII của	9	là 0011 1001 = 39H
Mã BCD không gói của	5	là 0000 0101 = 05H
Mã BCD không gói của	9	là 0000 1001 = 09H
Mã BCD gói chuẩn của	59	là 0101 1001 = 59H

1.3.3. CÁC PHÉP TOÁN SỐ HỌC VỚI MÃ HỆ HAI

1.3.3.1. Phép cộng

Phép cộng các số hệ hai được thực hiện giống như khi làm với các số hệ 10. Quy tắc phép cộng với số hệ hai được cho trong bảng 1.3.

Bảng 1.3

A	B	Y	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Y: kết quả

C: nhớ (Carry)

Ví dụ: $99 + 95 = 194$

Cộng hệ mười

	Nhớ
99	Số hạng 1
95	Số hạng 2
194	Tổng

Cộng hệ hai

1111 1110
0110 0011
0101 1111
1100 0010

Các bộ cộng trong các khối tính toán số học của máy tính sẽ thực hiện các phép cộng theo cách đã nói ở trên.

1.3.3.2. Phép trừ

Phép trừ các số hệ hai được thực hiện giống như khi làm với các số hệ 10. Quy tắc phép trừ với số hệ hai được cho trong bảng 1.4:

Bảng 1.4

a	b	Y	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Ví dụ: $194 - 95 = 99$

Trừ hệ mười	Mượn	Trừ hệ hai
		1111 1110
194	Số bị trừ	1100 0010
95	Số trừ	0101 1111
99	Hiệu	0110 0011

Trong khi làm phép trừ ta nhận thấy có thể thực hiện phép trừ bằng phép cộng. Cộng số bị trừ với số đối của số trừ (số bù hai).

Ví dụ: $15 - 13 = 2$

Ta đã biết -13 biểu diễn theo số bù hai như sau: 1111 0011

$$\begin{aligned}
 15 &= (0000\ 1111)_2 \\
 -13 &= (1111\ 0011)_{bù\ 2} \\
 \text{Tổng} &= (0000\ 0010)_2 = 2
 \end{aligned}$$

1.3.3.3. Phép nhân

Phép nhân các số hệ hai được thực hiện giống như khi làm với các số hệ 10. Quy tắc phép nhân với số hệ hai được cho trong bảng 1.5:

Bảng 1.5

a	b	Y
0	0	0
0	1	0
1	0	0
1	1	1

Trên cơ sở quy tắc vừa nêu và để cho đơn giản ta thực hiện một phép nhân 2 số hệ hai 4 bit để làm sáng tỏ thuật toán nhân:

1001	Số bị nhân (9)
0110	Số bị nhân (6)
0000	Thành phần 1 của tổng tích lũy
1001	Thành phần 2 của tổng tích lũy
1001	Thành phần 3 của tổng tích lũy
0000	Thành phần 4 của tổng tích lũy
0110110	Tổng tích lũy (54)

Tổng tích lũy = 54

Độ dài cực đại của kết quả trong trường hợp này là 8 bit. Nếu ta có các toán hạng 8 (hoặc 16) bit thì độ dài cực đại của phép toán là 16 (hoặc 32) bit. Phân tích kỹ ví dụ trên ta thấy phép nhân có thể thực hiện theo thuật toán cộng và dịch (trên cơ sở các bộ cộng và dịch như sau):

- Thành phần đầu tiên của tổng tích lũy thu được là tích của số LSB trong số nhân với số bị nhân. Nếu $LSB = 0$ thì thành phần này cũng bằng 0, còn nếu $LSB = 1$ thì thành phần này chính bằng số bị nhân.
- Mỗi thành phần thứ i tiếp theo của tổng tích lũy sẽ tính được bằng cách tương tự, nhưng phải dịch trái i bit (có thể bỏ qua các thành phần bằng 0)
- Tổng của các tổng thành phần là tích cần tìm. Lấy luôn ví dụ trên để minh họa.

1001	Số bị nhân (9)
0110	Số bị nhân (6)
1001	Số bị nhân dịch trái
1001	Số bị nhân dịch trái
0110110	Tổng tích lũy (54)

1.3.3.4. Phép chia

Vì vi xử lý gồm các phần tử để thực hiện phép cộng và dịch, vì vậy ta sẽ đưa ra quy tắc thực hiện phép chia bằng cách áp dụng phương pháp cộng và dịch.

Bước 1: Đổi số chia ra số bù hai của nó.

Bước 2: Lấy số bị chia cộng với số bù hai của số chia (trừ đi số chia).

Nếu kết quả này có bit dấu bằng 0 (nghĩa là phần này của số bị chia chia được cho số chia) thì bit tương ứng của thương bằng 1.

Nếu kết quả này có bit dấu bằng 1 (nghĩa là phần này của số bị chia không chia được cho số chia) thì bit tương ứng của thương bằng 0 và buộc phải khôi phục lại giá trị ban đầu của số bị chia bằng cách cộng kết quả với số chia ở mã hệ hai.

Bước 3: Dịch trái kết quả thu được ở trên và làm lại bước 2 cho đến khi nhận được kết quả là 0 (chia hết) hoặc nhỏ hơn số chia (chia còn dư).

Ví dụ: Thực hiện phép chia: 255/22

Số bị chia: 215 = 011010111

Số chia: 22 = 010110

Số bù hai của số chia = 101010

Các bước thực hiện được tiến hành như sau:

Số bị chia = 011010111

Số chia ở mã bù 2 để cộng = 101010

Tổng 1 = 000100111 bit dấu = 0 => thương = 1

Dịch trái tổng 1 = 00100111

Số chia ở mã bù 2 để cộng = 101010

Tổng 2 = 11001111 bit dấu = 1 => thương = 0

Số chia ở mã hệ 2 để cộng = 010110

Số bị chia = 00100111

Dịch trái số bị chia = 0100111

Số chia ở mã bù 2 để cộng = 101010

Tổng 3 = 1111011 bit dấu = 1 => thương = 0

Số chia ở mã hệ hai để cộng = 010110

Số bị chia = 0100111

Dịch trái số bị chia = 100111

Số chia ở mã bù 2 để cộng = 101010

Tổng 4 = 010001 bit dấu = 0 => thương = 1

Đến đây không thể thực hiện được nữa vì tổng 4 là 010001 = 17 < 22

Như vậy kết quả phép chia là: 1001 = 9 và dư 010001 = 17

• Phần bài tập

Bài 1. Thế nào là một hệ vi xử lý? Chức năng của từng khối trong hệ vi xử lý?

Nếu một hệ vi xử lý không có bộ nhớ trong ROM, RAM thì có hoạt động được không? Tại sao? Nếu một hệ vi xử lý không có khối ghép nối vào ra I/O thì có thể hoạt động được không? Tại sao?

Bài 2. Có mấy phương pháp biểu diễn số nguyên? Ưu, nhược điểm của từng phương pháp?

Thực hiện các phép tính sau:

$$01001101b + 00110010b = ?$$

$$10111100b - 01011111b = ?$$

Cho biết kết quả của phép tính ở hệ cơ số 10 nếu các số nói trên là:

a) Số nguyên không dấu;

b) Số nguyên có dấu biểu diễn theo kiểu dấu - độ lớn;

c) Số nguyên có dấu biểu diễn theo kiểu mã bù hai.

Bài 3. Để mã hóa một số -125 trong hệ vi xử lý có thể thực hiện theo những phương pháp nào? Mã của số trên cho từng phương pháp?

Thế nào là hiện tượng tràn trên trong phép cộng? Cho ví dụ minh họa.

CHƯƠNG II

HỌ VI XỬ LÝ INTEL 8086

• Phần lý thuyết

** Mục đích của chương:*

Giúp sinh viên khảo sát và nắm chắc cấu trúc một vi xử lý cụ thể của hãng Intel: Nắm được cấu trúc phần cứng (cấu trúc bên trong, sơ đồ chức năng các chân), cấu trúc phần mềm (các chế độ địa chỉ). Khảo sát các vi mạch phụ trợ (nắm được vai trò, chức năng các vi mạch phụ trợ, biết cách ghép nối chúng với vi xử lý 8086).

Nắm được cấu trúc của lập trình hợp ngữ 8086, từ đó lập trình những chương trình đơn giản sử dụng bộ vi xử lý 8086.

** Tóm tắt nội dung:*

Nội dung chương II tập chung trình bày kỹ về cấu trúc của bộ vi xử lý 8086 – thuộc họ vi xử lý 80x86 của Intel và cơ bản về lập trình hợp ngữ trên họ vi xử lý này, các nội dung chính của chương bao gồm: Cấu trúc phần cứng, cách quản lý bộ nhớ, các chế độ địa chỉ, tập lệnh và các vi mạch phụ trợ như mạch tạo xung 8284, vi mạch chốt, giải mã, các bộ nhớ để tạo nên một hệ vi xử lý hoàn chỉnh; cách lập trình hợp ngữ dựa trên tập lệnh của bộ vi xử lý để tạo ra chương trình dạng *.EXE hoặc *.COM.

2.1. CẤU TRÚC CỦA BỘ VI XỬ LÝ 8086

2.1.1. TỔNG QUAN

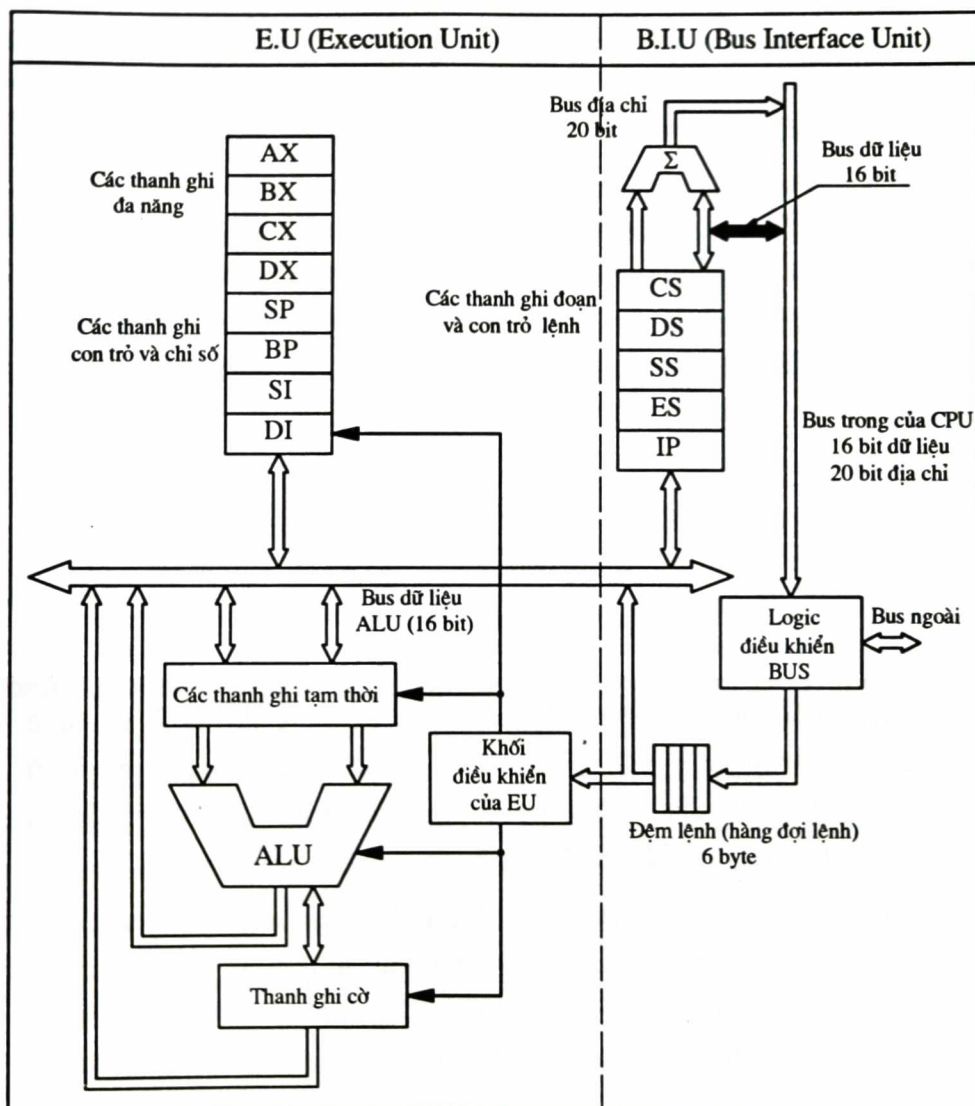
Sau khi đã tìm hiểu qua về cấu trúc của vi xử lý, tiếp theo ta sẽ đi sâu tìm hiểu một bộ vi xử lý cụ thể và rất điển hình: Bộ vi xử lý 80x86 của Intel. Các thông số của 8086 như sau:

- Năm sản xuất: 6/1978
- f_{clkmax} (đồng hồ nhịp): 10MHz
- MIPS (triệu lệnh/s): 0,33
- Số tranzitor: 29000
- Bus số liệu: 16 bit
- Bus địa chỉ: 20 bit
- Khả năng địa chỉ: 1MB
- Số chân: 40
- Độ dài bộ nhớ đệm lệnh (hàng đợi): 6 byte
- Có thể thao tác với bit, byte, từ, từ khối
- Có khả năng thực hiện phép tính với các số 8 và 16 bit có dấu hoặc không có dấu dạng nhị phân hoặc thập phân, bao gồm cả phép chia và nhân.

2.1.2. CẤU TRÚC BÊN TRONG VÀ HOẠT ĐỘNG CỦA VI XỬ LÝ 8086

Hình 2.1 là sơ đồ khối cấu trúc bên trong của vi xử lý 8086.

- EU: Execution Unit, khối thực hiện lệnh.
- BIU: Bus Interface Unit, khối phối ghép bus.
- ALU: Arithmetic and Logic Unit, khối số học và logic.



Hình 2.1. Sơ đồ khối cấu trúc bên trong của vi xử lý 8086

2.1.2.1. Các khối chức năng của CPU

Chức năng bên trong vi xử lý về mặt logic được chia làm hai khối xử lý. Khối thứ nhất là khối giao diện bus (BIU) và khối thứ hai là khối thực hiện lệnh (EU).

BIU: Cung cấp các chức năng liên quan đến việc nhận lệnh và xếp hàng đệm lệnh (hàng đợi), lưu trữ các toán hạng và định vị các địa chỉ. Khối này cũng cung cấp các chức năng điều khiển Bus cơ sở.

Hàng đợi là một RAM nội bộ tốc độ cao, độ dài của hàng đợi này với vi xử lý 8086 là 6byte. Kỹ thuật hàng đợi lệnh cho phép BIU sử dụng bộ nhớ rất hiệu quả. BIU sẽ lấy mã lệnh trong bộ nhớ rồi đưa vào hàng đợi. Như vậy, BIU có thể cung cấp các lệnh một cách liên tục mà không độc chiếm BIU. Điều này làm giảm đáng kể thời gian chết trên Bus. Hàng đợi lệnh làm việc như một bộ đệm lệnh FIFO (First In First Out, vào trước ra trước).

Nếu có sự vào/ra liên tục của dòng mã lệnh trong bộ đệm này thì có nghĩa là có sự phối hợp hoạt động hiệu quả giữa hai khối EU và BIU theo cơ chế xử lý xen kẽ liên tục dòng mã lệnh để làm tăng tốc độ xử lý tổng thể.

EU: Nhận các lệnh được lấy ra trước từ hàng đợi lệnh và cung cấp các toán hạng, các địa chỉ cho BIU để khối này đọc lệnh và dữ liệu. Trong khi đó, bản thân EU sẽ giải mã lệnh, thực hiện, rồi lại chuyển các kết quả tới BIU để lưu trữ.

Các lệnh chứa trong hàng đợi lệnh chính là những lệnh cất trong các ô nhớ liên tiếp nhau và kế tiếp lệnh đang được thực hiện. Nếu EU thực hiện một lệnh rồi chuyển điều khiển đến một nơi khác thì BIU sẽ xóa hàng đợi, lấy lệnh từ địa chỉ mới, chuyển ngay cho EU rồi lại bắt đầu lấy tiếp các lệnh để đưa vào hàng đợi.

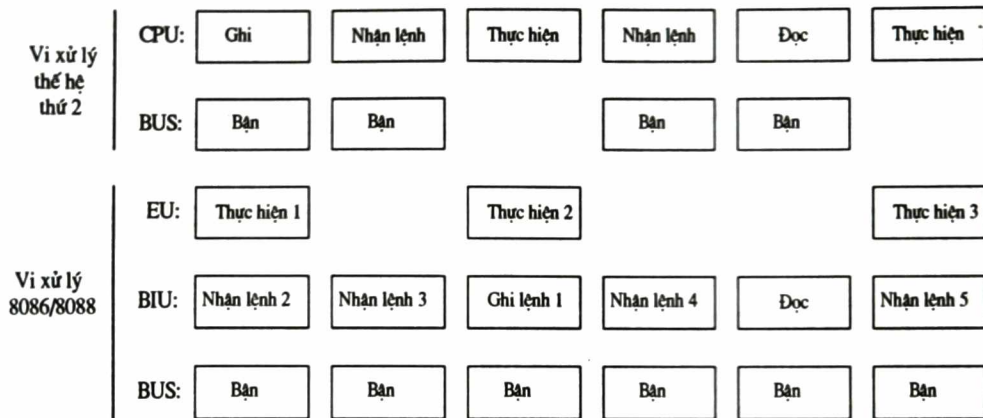
Cơ cấu nhận lệnh và thực hiện lệnh của vi xử lý 8086/8088

Chi dẫn lệnh:

Lệnh 1 (đã có sẵn): Thực hiện và ghi kết quả

Lệnh 2: Chỉ thực hiện lệnh

Lệnh 3: Đọc toán hạng và thực hiện



ALU: Đây chỉ là một tập con của EU, nhưng trong thực tế nó giống như một phần có cấu trúc độc lập, chịu trách nhiệm thực hiện các thao tác số học và các thao tác logic. Kết quả lại được định vị trong một thanh ghi hoặc trong bộ nhớ và 6 cờ trạng thái được cập nhật dựa trên kết quả của các thao tác này.

2.1.2.2. Các thanh ghi của CPU

Các thanh ghi có thể được chia làm 4 nhóm lần lượt có tên là:

- Các thanh ghi đoạn: CS, DS, SS, ES.
- Các thanh ghi đa năng: AX, BX, CX, DX.
- Các thanh ghi con trỏ và chỉ số: IP, BP, SP, SI, DI.
- Thanh ghi cờ. FR (Flag).

Thanh ghi đoạn

Khởi BIU đưa ra trên Bus địa chỉ 20 bit địa chỉ. Như vậy, 8086 có khả năng phân biệt được $2^{20} = 1048576 = 1\text{M}$ ô nhớ hay 1MB. Trong không gian 1MB này bộ nhớ cần được chia ra thành các vùng khác nhau dành riêng để:

- Chứa mã chương trình.
- Chứa dữ liệu và kết quả trung gian của chương trình.

– Tạo ra một vùng nhớ đặc biệt gọi là ngăn xếp (stack) dùng vào việc quản lý các thông số của bộ vi xử lý khi gọi chương trình con hoặc trở về từ chương trình con.

Trong thực tế vi xử lý 8086/8088 có các thanh ghi 16 bit liên quan đến địa chỉ đầu của các vùng (đoạn) kể trên và chúng được gọi là các thanh ghi đoạn (Segment register). Đó là các thanh ghi:

– CS (Code Segment): **Thanh ghi đoạn mã**, chứa địa chỉ bắt đầu của đoạn chương trình (đoạn mã) mang những lệnh thực hiện được và thông thường là một vùng nhớ chứa dữ liệu dạng hàng không thể thay đổi được hoặc là một vùng ROM/EPROM.

– DS (Data Segment): **Thanh ghi đoạn dữ liệu**, chứa địa chỉ bắt đầu của đoạn dữ liệu, bao gồm các tham số, các biến, các mảng số liệu...

– SS (Stack Segment): **Thanh ghi đoạn ngăn xếp**, chứa địa chỉ bắt đầu của đoạn ngăn xếp. Đây là một mảng của RAM, nơi mà dữ liệu tồn tại trong các thanh ghi được lưu trữ trong suốt quá trình ngắt hoặc thực hiện chương trình con.

– ES (Extra Segment): **Thanh ghi đoạn dữ liệu phụ**, chứa địa chỉ bắt đầu của vùng nhớ bổ sung.

Dung lượng lớn nhất của mỗi đoạn nhớ này là 64 Kbyte. Việc thay đổi giá trị các thanh ghi đoạn tương ứng có thể dịch chuyển linh hoạt trong phạm vi không gian 1 Mbyte. Vì vậy, các đoạn này có thể nằm cách nhau khi thông tin cần lưu trữ trong chúng đòi hỏi dung lượng đủ 64 Kbyte hoặc cũng có thể nằm trùm lên nhau do có những đoạn không cần dùng hết dung lượng 64 Kbyte.

Nội dung của thanh ghi đoạn cho phép ta xác định địa chỉ ô nhớ nằm ở đâu đoạn. Địa chỉ này gọi là địa chỉ cơ sở, địa chỉ của các ô nhớ khác nằm trong đoạn được tính bằng cách cộng thêm vào địa chỉ cơ sở một giá trị gọi là địa chỉ lệnh hay độ lệnh (offset). Độ lệnh này được

xác định bởi một thanh ghi 16 bit khác đóng vai trò thanh ghi lệnh (offset register).

Mọi sự trao đổi thông tin trong hệ thống vi xử lý đều dùng địa chỉ vật lý, còn địa chỉ được tạo bởi thanh ghi đoạn và thanh ghi lệnh như trên được gọi là địa chỉ logic và được ký hiệu như sau:

Địa chỉ logic = Thanh ghi đoạn: Thanh ghi lệnh

Địa chỉ logic tồn tại dưới dạng giá trị các thanh ghi cụ thể bên trong CPU và khi cần thiết truy nhập ô nhớ nào đó thì nó phải được đổi ra địa chỉ vật lý để đưa lên bus địa chỉ (phần tử Σ trên hình vẽ). Việc chuyển đổi này do một bộ tạo địa chỉ thực hiện.

Địa chỉ vật lý của ô nhớ được tính theo công thức sau:

20 bit địa chỉ vật lý = Thanh ghi đoạn $\times$ 16 + Thanh ghi lệnh

Ví dụ: Cặp CS:IP sẽ chỉ ra địa chỉ của lệnh sắp thực hiện trong đoạn mã. Nếu tại một thời điểm nào đó ta có CS = F000H và IP = FFF0H thì

$$\text{CS:IP} \sim \text{F000H} \times 16 + \text{FFF0H} = \text{F0000H} + \text{FFF0H} = \text{FFFF0H}$$

Ta cũng cần chú ý rằng một giá trị địa chỉ vật lý sẽ có nhiều cách tạo ra từ nhiều giá trị thanh ghi đoạn và thanh ghi lệnh.

Ví dụ: địa chỉ vật lý của 32412H có thể được tạo ra từ các giá trị.

Thanh ghi đoạn	Thanh ghi lệnh
3000H	2412H
3200H	0412H
3240H	0012H
...	

Các thanh ghi đa năng

Trong khối EU có 4 thanh ghi đa năng AX, BX, CX, DX. Điều đặc biệt là khi cần chứa dữ liệu 8 bit thì mỗi thanh ghi này có thể tách ra làm 2 thanh ghi 8 bit cao và thấp làm việc độc lập nhau, đó là các

thanh ghi AH và AL, BH và BL, CH và CL, DH và DL. Mỗi thanh ghi có thể được dùng một cách vạn năng để chứa các loại dữ liệu khác nhau, nhưng chúng cũng có những chức năng riêng:

- AX (Accumulator, Acc): **Thanh chứa**, các kết quả của các thao tác thường được chứa ở đây, nếu kết quả là 8 bit thì thanh ghi AL được gọi là Acc.
- BX (Base): **Thanh ghi cơ sở**, thường chứa địa chỉ cơ sở của một bảng trong bộ nhớ.
- CX (Count): **Thanh ghi đếm**, thường dùng để chứa số lần lặp của lệnh lặp, còn CL thường dùng chứa số lần dịch hoặc quay trong các lệnh dịch hoặc quay.
- DX (Data): **Thanh ghi dữ liệu**. DX và AX tham gia vào thao tác của các phép nhân hoặc chia 16 bit, DX còn dùng để chứa địa chỉ của các cổng trong các lệnh vào/ra dữ liệu (IN/OUT).

Các thanh ghi con trỏ và chỉ số

8086 có ba thanh ghi con trỏ và hai thanh ghi chỉ số 16 bit, các thanh ghi này (trừ IP) đều có thể được dùng như các thanh ghi đa năng, nhưng ứng dụng chính của mỗi thanh ghi là chúng được ngầm định như là thanh ghi lệnh cho các đoạn tương ứng.

- IP (Instruction Pointer): **Con trỏ lệnh**, IP luôn trỏ vào lệnh tiếp theo sẽ được thực hiện nằm trong đoạn mã CS. Địa chỉ đầy đủ của lệnh tiếp theo này ứng với CS:IP và được xác định theo cách đã nói ở trên.
- BP (Base Pointer): **Con trỏ cơ sở**, BP luôn trỏ vào một dữ liệu nằm trong đoạn ngăn xếp SS. Địa chỉ đầy đủ của một phần tử trong đoạn ngăn xếp ứng với SS:BP và được xác định theo cách đã nói ở trên.

- SP (Stack Pointer): **Con trỏ ngăn xếp**, luôn trỏ vào đỉnh hiện thời của ngăn xếp nằm trong đoạn ngăn xếp SS. Địa chỉ đầy đủ của đỉnh ngăn xếp ứng với SS:SP và được xác định theo cách đã nói ở trên.
- SI (Source Index): **Chỉ số nguồn**, SI chỉ vào dữ liệu cần lấy trong đoạn dữ liệu DS mà địa chỉ đầy đủ tương ứng với DS:SI và được xác định theo cách đã nói ở trên.
- DI (Destination Index): **Chỉ số đích**, DI chỉ vào dữ liệu cần cất trong đoạn dữ liệu DS mà địa chỉ đầy đủ tương ứng với DS:DI và được xác định theo cách đã nói ở trên.

Thanh ghi cờ FR (Flag Register)

Mỗi bit của thanh ghi cờ để phản ánh một trạng thái nhất định của kết quả phép toán do ALU thực hiện hoặc một hoạt động của EU. Dựa vào các cờ này mà người lập trình có thể đưa ra các lệnh thích hợp tiếp theo cho vi xử lý (các lệnh nhảy có điều kiện). Thanh ghi cờ có 16 bit nhưng chỉ sử dụng 9 bit làm bit cờ.

X	X	X	X	O	D	I	T	S	Z	X	A	X	P	X	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

x: Không được định nghĩa

Các cờ cụ thể:

Các cờ trạng thái

- C hoặc CF (Carry Flag): **Cờ nhớ** CF = 1 khi có nhớ hoặc mượn từ MSB.
- F hoặc PF (Parity Flag): **Cờ chẵn lẻ**, phản ánh tính chẵn lẻ của tổng số bit 1 có trong kết quả. CF = 1 khi tổng số bit 1 trong kết quả là chẵn.
- A hoặc AF (Auxiliary carry Flag): **Cờ nhớ phụ**, rất có ý nghĩa khi ta làm việc với các số BCD, AF = 1 khi có nhớ

hoặc mượn từ một số BCD thấp (4 bit thấp) sang một số BCD cao (4 bit cao).

- Z hoặc ZF (Zero Flag): **Cờ rỗng**, ZF = 1 khi kết quả bằng 0.
- S hoặc SF (Sign Flag): **Cờ dấu**, SF = 1 khi kết quả âm.
- O hoặc OF (Overflow Flag): **Cờ tràn**, OF = 1 khi kết quả là số bù hai vượt ra ngoài giá trị biểu diễn của nó.

Các cờ điều khiển (có thể lập hoặc xoá bằng các lệnh riêng)

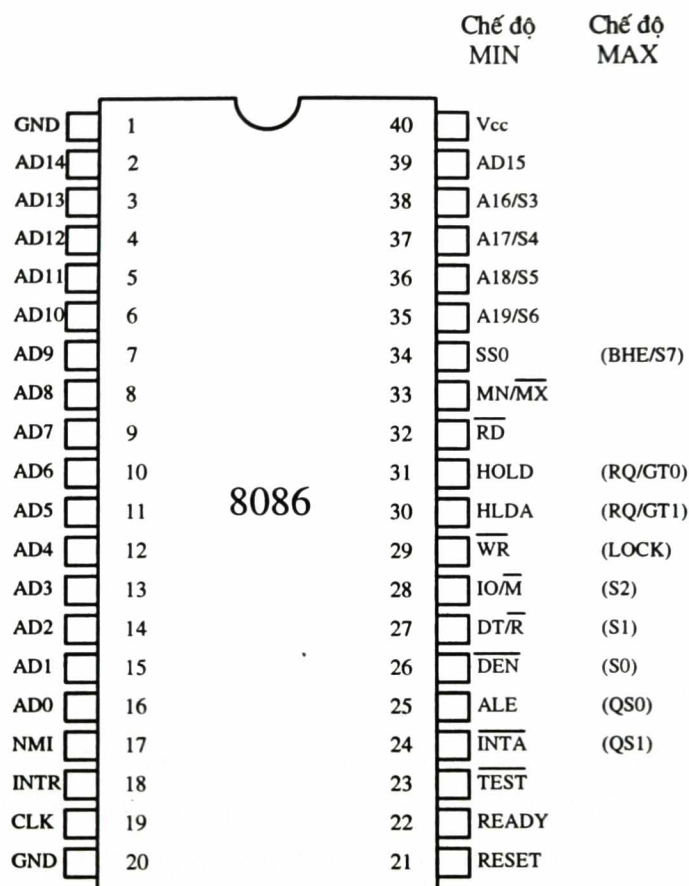
- T hoặc TF (Trap Flag): **Cờ bẫy**, TF = 1 thì CPU làm việc ở chế độ chạy từng lệnh (chế độ này dùng khi cần tìm lỗi chương trình).
- I hoặc IF (Interrupt enable Flag): **Cờ cho phép ngắt**, IF = 1 thì CPU cho phép các yêu cầu ngắt được tác động.
- D hoặc DF (Direction Flag): **Cờ hướng**, DF = 1 khi CPU làm việc với chuỗi ký tự theo ký tự từ phải sang trái (vì vậy D chính là cờ lùi).

2.1.3. MÔ TẢ CHỨC NĂNG CÁC CHÂN CỦA VI XỬ LÝ 8086

Hình 2.2 là sơ đồ bố trí chân của vi xử lý 8086.

Ta ký hiệu I/O tương ứng là tín hiệu đi vào và đi ra khỏi vi xử lý.

- AD0 ÷ AD15 [I, O]: Các chân dẫn kênh cho các tín hiệu của Bus dữ liệu và Bus địa chỉ. Xung ALE sẽ báo cho mạch ngoài biết khi nào trên các đường đó có tín hiệu dữ liệu (ALE = 0) hoặc địa chỉ (ALE = 1).
- A16/S3, A17/S4, A18/S5, A19/S6 [O]: Địa chỉ/trạng thái. Đây là 4 đường địa chỉ cao nhất. Địa chỉ A16 – A19 sẽ có mặt tại các chân đó khi ALE = 1 còn khi ALE = 0 thì trên các chân đó có tín hiệu trạng thái S3 – S6.



Hình 2.2. Sơ đồ chân của vi xử lý 8086

A17/S4	A16/S3	Truy nhập đến
0	0	Đoạn dữ liệu phụ (ES)
0	1	Đoạn ngăn xếp (SS)
1	0	Đoạn mã (CS) hoặc không đoạn nào
1 S6 luôn là 0	1	Đoạn dữ liệu (DS)

Bit S6 = 0 liên tục, bit S5 phản ánh giá trị bit IF của thanh ghi cờ, hai bit S3, S4 phối hợp với nhau để chỉ ra việc truy nhập các thanh ghi

đoạn. Tín hiệu này chuyển sang trạng thái trở kháng cao khi Bus nội bộ ghi nhận tín hiệu treo.

- RD [O]: Đọc. Tín hiệu đọc cho biết bộ vi xử lý đang thực hiện đọc bộ nhớ hay đọc I/O phụ thuộc vào trạng thái chân S2. RD = 0 thì Bus dữ liệu sẵn sàng nhận số liệu từ bộ nhớ hoặc thiết bị ngoại vi.
- READY [I]: Tín hiệu báo cho CPU biết tình trạng sẵn sàng của thiết bị ngoại vi hay bộ nhớ. Khi READY = 1 thì CPU thực hiện đọc/ghi mà không cần chờ thêm các chu kỳ đợi. Ngược lại, khi thiết bị ngoại vi hay bộ nhớ có tốc độ hoạt động chậm, chúng có thể đưa tín hiệu READY = 0 để báo cho CPU biết mà chờ chúng, lúc này CPU tự động kéo dài thời gian thực hiện lệnh đọc/ghi bằng cách chờ thêm các chu kỳ đợi.
- INTR [I]: Tín hiệu yêu cầu ngắt che được. Khi có yêu cầu ngắt mà cờ cho phép ngắt IF = 1 thì CPU kết thúc lệnh đang làm dở, sau đó đi vào chu kỳ chấp nhận ngắt và đưa ra bên ngoài tín hiệu INTA = 0.
- TEST [I]: Tín hiệu tại chân này được kiểm tra bởi lệnh WAIT (xem phần tập lệnh). Khi CPU thực hiện lệnh WAIT mà lúc đó tín hiệu TEST = 1 nó sẽ chờ cho đến khi TEST = 0 thì nó mới thực hiện lệnh tiếp theo.
- NMI [I]: Tín hiệu yêu cầu ngắt không che được. Tín hiệu này không chịu sự khống chế của cờ IF và nó sẽ được CPU nhận biết bằng tác động của sườn lên của xung yêu cầu ngắt. Nhận được yêu cầu này CPU kết thúc lệnh đang làm dở sau đó nó chuyển sang chương trình phục vụ ngắt kiểu INT 2.

- RESET [I]: Tín hiệu khởi động lại 8086. Khi RESET = 1 kéo dài ít nhất trong thời gian 4 chu kỳ đồng hồ thì 8086 buộc phải khởi động lại.
- CLK [I]: Tín hiệu đồng hồ (xung nhịp). Xung nhịp có độ rộng là 77% và cung cấp nhịp làm việc cho CPU.
- Vcc [I]: Chân nguồn nuôi, tại đây CPU được cung cấp nguồn +5V±10%, 340mA.
- GND [O]: Hai chân nguồn để nối với điểm 0V của nguồn nuôi.
- MN/MX [I]: Chân điều khiển hoạt động của CPU theo chế độ MIN/MAX.

Chế độ MIN: Chân MN/MX được nối thẳng vào +5V mà không qua điện trở. Trong chế độ MIN tất cả các tín hiệu điều khiển liên quan đến thiết bị ngoại vi truyền thông và bộ nhớ đã có sẵn trong 8086. Vì vậy, việc phối ghép với các thiết bị đó rất dễ dàng và chính vì tận dụng được các phối ghép ngoại vi có sẵn nên có thể giảm giá thành hệ thống.

- IO/M [O]: Tín hiệu này phân biệt trong thời điểm đã định phần tử nào trong các thiết bị vào/ra (IO) hoặc bộ nhớ (M) được chọn làm việc với CPU.
- WR [O]: Xung cho phép ghi. Khi CPU đưa ra WR = 0 thì trên bus dữ liệu các dữ liệu đã ổn định và chúng sẽ được ghi vào bộ nhớ hoặc thiết bị ngoại vi tại thời điểm đột biến WR = 1.
- ALE [O]: Xung cho phép chốt địa chỉ. Khi ALE = 1 có nghĩa là trên bus dữ liệu AD có các địa chỉ của thiết bị vào/ra hay của ô nhớ. ALE không bao giờ ở trạng thái trở kháng cao, khi CPU bị treo thì ALE = 0.
- DT/R [O]: Tín hiệu điều khiển các đệm hai chiều của Bus dữ liệu để chọn chiều chuyển trên Bus D.

- DEN [O]: Tín hiệu báo cho bên ngoài biết là lúc này trên Bus dồn kênh ADDRESS có dữ liệu ổn định.
- SS0 [O]: Tín hiệu trạng thái. Tín hiệu này giống như S0 ở chế độ MAX và được dùng kết hợp với IO/M, DT/R để giải mã các chu kỳ hoạt động của Bus.

Bảng 2.1.

IO/M	DT/R	SS0	Chu kỳ điều khiển bus
0	0	0	Đọc mã lệnh
0	0	1	Đọc bộ nhớ
0	1	0	Ghi bộ nhớ
0	1	1	Bus rỗi
1	0	0	Chấp nhận yêu cầu ngắt
1	0	1	Đọc thiết bị ngoại vi
1	1	0	Ghi thiết bị ngoại vi
1	1	1	Dừng

Bảng các chu kỳ của bus qua các tín hiệu SS0, IO/M, DT/R

- HOLD [I]: Tín hiệu yêu cầu treo CPU để mạch ngoài thực hiện việc trao đổi dữ liệu với bộ nhớ bằng cách thâm nhập trực tiếp (Direct Memory Access, DMA). Khi HOLD = 1, CPU sẽ tự tách ra khỏi hệ thống bằng cách treo bus (các bus ở trạng thái trở kháng cao) để bộ điều khiển DMA (DMA Controller, DMAC) có thể lấy được quyền điều khiển hệ thống để làm các công việc trao đổi dữ liệu.
- HLDA [O]: Báo tín hiệu cho bên ngoài biết yêu cầu treo CPU để dùng các bus đã được chấp nhận và CPU 8086 đã treo các bus.

Chế độ MAX: Chân MN/MX nổi đất. Trong chế độ này một số tín hiệu điều khiển cần thiết được tạo ra trên cơ sở các tín hiệu trạng thái nhờ dùng thêm ở bên ngoài một bộ mạch điều khiển bus 8288. Chế độ MAX được sử dụng khi trong hệ thống có mặt bộ đồng xử lý toán học 8087.

– S2, S1, S0 [O]: Các chân trạng thái dùng trong chế độ MAX để ghép nối với mạch điều khiển bus 8288 (xem phần điều khiển bus trong chế độ MAX). Các tín hiệu này được 8288 dùng để tạo ra các tín hiệu điều khiển trong các chu kỳ hoạt động của bus.

Bảng 2.2. Bảng các tín hiệu điều khiển của 8288

S2	S1	S0	Chu kỳ điều khiển bus	Tín hiệu
0	0	0	Chấp nhận yêu cầu ngắt	INTA
0	0	1	Đọc thiết bị ngoại vi	IORC
0	1	0	Ghi thiết bị ngoại vi	IOWC, AIOWC
0	1	1	Dùng	Không
1	0	0	Đọc mã lệnh	MRDC
1	0	1	Đọc bộ nhớ	MRDC
1	1	0	Ghi bộ nhớ	MWTC, AMWC
1	1	1	Bus rỗi	Không

– QS0 và QS1 [O]: Tín hiệu thông báo các trạng thái khác nhau của đệm lệnh (hàng đợi).

– RQ/GT0 và RQ/RT1 [I/O]: Các tín hiệu yêu cầu dùng Bus của các bộ vi xử lý khác hoặc thông báo chấp nhận treo của CPU để cho phép các bộ vi xử lý khác dùng Bus. RQ/GT0 có mức ưu tiên cao hơn RQ/RT1.

Bảng 2.3. Bảng các trạng thái của đệm lệnh

QS1	QS0	Trạng thái đệm lệnh
0	0	Không hoạt động
0	1	Đọc byte mã lệnh đầu tiên từ đệm lệnh
1	0	Đệm lệnh rỗng
1	1	Đọc byte tiếp theo từ đệm lệnh

– LOCK [O]: Tín hiệu do CPU đưa ra để cấm các bộ vi xử lý khác dùng Bus trong khi nó đang thi hành một lệnh nào đó đặt sau lệnh LOCK (xem phân tập lệnh vi xử lý 8086).

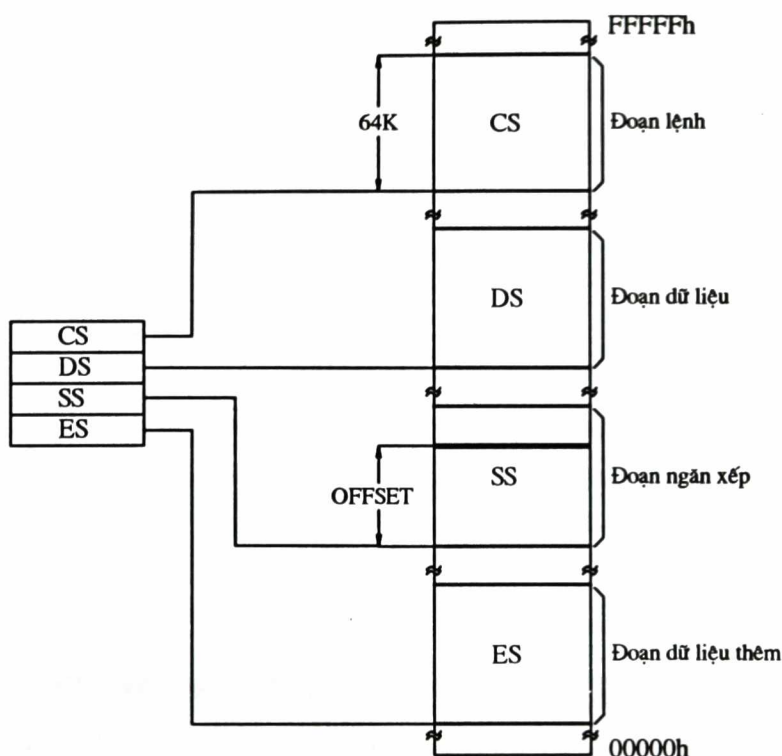
2.1.4. TỔ CHỨC BỘ NHỚ CỦA VI XỬ LÝ 8086

Vi xử lý 8086 có 20 đường địa chỉ và do đó có thể địa chỉ hoá được 2^{20} ô nhớ dữ liệu 8 bit kéo dài từ địa chỉ 00000h đến FFFFFh. Bộ nhớ logic được chia thành đoạn lệnh (CS), đoạn dữ liệu (DS), đoạn ngăn xếp (SS), đoạn thêm (ES) với 64 Kbyte cho mỗi đoạn. Tổ chức bộ nhớ của 8086 được biểu diễn như hình 2.3.

Bộ nhớ của 8086 được đánh địa chỉ theo đơn vị byte (8 bit), trong khi đó có nhiều thao tác sử dụng số 16 bit. Trong bộ nhớ một số 16 bit được lưu thành hai byte kề nhau. Byte thấp được lưu ở địa chỉ thấp hơn. Chẳng hạn, với số 3D7Fh sẽ được lưu trong bộ nhớ thành 7F3Dh. Đây là một chú ý quan trọng khi làm việc với các số 16 bit trong bộ nhớ.

CPU sẽ thực hiện lệnh đầu tiên ở địa chỉ FFFF0h mà ở đó thường chứa một lệnh nhảy.

Các vị trí từ 00000h – 003FFh được dự tính cho các thao tác xử lý ngắt. Trong phần này chứa CPU sẽ thực hiện lệnh đầu tiên ở địa chỉ FFFF0h mà ở đó thường chứa một lệnh nhảy.



Hình 2.3. Tổ chức bộ nhớ của vi xử lý 8086

Các vị trí từ 00000h – 003FFh được dự tính cho các thao tác xử lý ngắt. Trong phần này chứa địa chỉ chương trình con xử lý ngắt nên được gọi là bảng vector ngắt.

2.2. CÁC CHẾ ĐỘ ĐỊNH ĐỊA CHỈ CỦA BỘ VI XỬ LÝ 8086

Chế độ địa chỉ (addressing mode) là cách để CPU tìm thấy toán hạng cho các lệnh của nó khi hoạt động. Một bộ vi xử lý có thể có nhiều chế độ địa chỉ. Các chế độ địa chỉ này được xác định ngay từ khi chế tạo và không thể thay đổi được. Bộ vi xử lý 8086/8088 có 7 chế độ địa chỉ sau:

- Chế độ địa chỉ thanh ghi.
- Chế độ địa chỉ tức thì.

- Chế độ địa chỉ trực tiếp.
- Chế độ địa chỉ gián tiếp qua thanh ghi.
- Chế độ địa chỉ tương đối cơ sở.
- Chế độ địa chỉ tương đối chỉ số.
- Chế độ địa chỉ tương đối cơ sở chỉ số.

2.2.1. CHẾ ĐỘ ĐỊA CHỈ THANH GHI

Trong chế độ địa chỉ này người ta sử dụng các thanh ghi có sẵn trong CPU như là các toán hạng để chứa dữ liệu cần thao tác, vì vậy khi thực hiện có thể đạt tốc độ truy nhập cao hơn so với các lệnh truy nhập đến bộ nhớ.

Ví dụ:

MOV BX, DX ;copy nội dung DX vào BX

ADD AX, BX ;cong AX với BX rồi ghi kết quả vào AX

2.2.2. CHẾ ĐỘ ĐỊA CHỈ TỨC THÌ

Trong chế độ này toán hạng đích là một thanh ghi hay một ô nhớ, còn toán hạng nguồn là một hằng số. Ta có thể dùng chế độ địa chỉ này để nạp dữ liệu cần thao tác vào bất kỳ thanh ghi nào trừ (thanh ghi đoạn và thanh ghi cờ) và bất kỳ ô nhớ nào trong đoạn dữ liệu DS.

Ví dụ:

MOV CL, 100;chuyển 100 vào CL.

MOV AX, 0BC8h;chuyển 0BC8h vào AX để rồi

MOV DS, AX;copy nội dung AX vào DS (vì ;không được chuyển trực tiếp vào thanh ghi đoạn)

MOV [BX], 20;chuyển 20 vào ô nhớ tại
;địa chỉ DS:BX.

2.2.3. CHẾ ĐỘ ĐỊA CHỈ TRỰC TIẾP

Trong chế độ địa chỉ này một toán hạng chứa địa chỉ lệnh của ô nhớ dùng chứa dữ liệu, còn toán hạng kia có thể là thanh ghi mà không được là ô nhớ.

Ví dụ:

```
MOV AL, [0243H];chuyen noi dung o nho  
;DS:0243 vao AL
```

```
MOV [4320],CX;chuyen CX vao hai o nho lien  
;tiep DS:4320 va DS:4321
```

2.2.4. CHẾ ĐỘ ĐỊA CHỈ GIÁN TIẾP QUA THANH GHI

Trong chế độ địa chỉ này một toán hạng là một thanh ghi được sử dụng để chứa địa chỉ lệnh của ô nhớ dữ liệu, còn toán hạng kia là thanh ghi mà không được là ô nhớ.

Ví dụ:

```
MOV AL, [BX];copy noi dung o nho co dia  
;chi DS:BX
```

```
MOV [SI], CL;copy noi dung CL vao o nho  
;co dia chi DS:SI
```

```
MOV [DI], AX;copy noi dung AX vao hai o  
;nho lien tiep co dia chi DS:DI va DS:(DI+1)
```

2.2.5. CHẾ ĐỘ ĐỊA CHỈ TƯƠNG ĐỐI CƠ SỞ

Một toán hạng là thanh ghi, một toán hạng là ô nhớ chứa dữ liệu có địa chỉ lệnh được biểu diễn bởi một thanh ghi cơ sở BX (hoặc BP) cộng với một hằng số biểu diễn giá trị dịch chuyển. Thanh ghi đoạn là DS hoặc SS

Ví dụ:

```
MOV CX, [BX+10];copy noi dung hai o nho  
;lien tiep co dia chi DS:BX+10 va  
;DS:BX+11 vao CX
```

```
MOV AL, [BP+5];chuyen noi dung o nho co  
;dia chi SS:BP+5 vao AL
```

Quan sát trên ta thấy: 10 và 5 là các dịch chuyển của các toán hạng tương ứng.

BX+10, BP+5 gọi là địa chỉ hiệu dụng.

DS:BX+10, SS:BP+5 chính là địa chỉ logic ứng với địa chỉ vật lý.

2.2.6. CHẾ ĐỘ ĐỊA CHỈ TƯƠNG ĐỐI CHỈ SỐ

Một toán hạng là thanh ghi, còn toán hạng kia là ô nhớ chứa dữ liệu có địa chỉ lệnh được biểu diễn bởi một thanh ghi chỉ số SI (hoặc DI) cộng với một hằng số biểu diễn giá trị dịch chuyển. Thanh ghi đoạn là DS.

Ví dụ:

```
MOV CX, [SI+10];copy noi dung hai o nho lien  
;tiep co dia chi DS:SI+10 va DS:SI+11vao CX  
MOV AL, [DI+5];chuyen noi dung o nho co  
;dia chi DS:DI+5 vao AL
```

2.2.7. CHẾ ĐỘ ĐỊA CHỈ TƯƠNG ĐỐI CHỈ SỐ CƠ SỞ

Một toán hạng là thanh ghi, một toán hạng là ô nhớ chứa dữ liệu có địa chỉ lệnh được biểu diễn bởi một thanh ghi cơ sở BX (hoặc BP) cộng với một thanh ghi chỉ số SI (hoặc DI) cộng với một hằng số biểu diễn giá trị dịch chuyển. Thanh ghi đoạn là DS hoặc SS.

Ví dụ:

```
MOV BX, [BX+SI+10]
```

MOV AL, [BP+DI+5]

Các chế độ địa chỉ đã trình bày ở trên có thể tóm tắt lại trong bảng 2.4:

Bảng 2.4

Chế độ địa chỉ	Toán hạng	Thanh ghi đoạn ngầm định
Thanh ghi	Reg	
Tức thì	Data	
Trực tiếp	[offset]	DS
Gián tiếp qua thanh ghi	[BX]	DS
	[SI]	DS
	[DI]	DS
Tương đối cơ sở	[BX]+Disp	DS
	[BP]+Disp	SS
Tương đối chỉ số	[DI]+Disp	DS
	[SI]+Disp	DS
Tương đối chỉ số cơ sở	[BX]+[DI]+Disp	DS
	[BX]+[SI]+Disp	DS
	[BP]+[DI]+Disp	SS
	[BP]+[SI]+Disp	SS

Chú ý: Reg: Thanh ghi, Data: Dữ liệu tức thì, Disp: Dịch chuyển.

2.3. TẬP LỆNH CỦA VI XỬ LÝ 8086

2.3.1. GIỚI THIỆU CHUNG

Tập lệnh của họ vi xử lý 80x86 đảm bảo tương thích thế hệ sau với thế hệ trước. Điều đó có nghĩa là các chương trình viết cho 8086

vẫn chạy được trên các bộ vi xử lý mới hơn mà không phải thay đổi (không đảm bảo thứ tự ngược lại). Tập lệnh của một bộ vi xử lý thường có rất nhiều lệnh (hàng trăm lệnh), vì thế mà việc tiếp cận và làm chủ chúng là tương đối khó khăn. Có nhiều cách trình bày tập lệnh của bộ vi xử lý: Trình bày theo nhóm lệnh hoặc theo thứ tự abc. Để có thể nhanh chóng và dễ dàng sử dụng các lệnh cơ bản và lập trình được ngay, ta sẽ trình bày theo nhóm lệnh:

- Nhóm các lệnh vận chuyển (sao chép) dữ liệu.
- Nhóm các lệnh tính toán số học.
- Nhóm các lệnh tính toán logic.
- Nhóm các lệnh dịch, quay toán hạng.
- Nhóm các lệnh nhảy (rẽ nhánh).
- Nhóm các lệnh lặp.
- Nhóm các lệnh điều khiển, đặc biệt khác.

2.3.2. TẬP LỆNH CỦA VI XỬ LÝ 8086

2.3.2.1. Nhóm các lệnh vận chuyển (sao chép) dữ liệu

1. LDS – Load register and DS with words from memory (nạp một từ (từ bộ nhớ) vào thanh ghi cho trong lệnh và một từ tiếp theo vào DS).

Dạng lệnh: LDS Đích, Nguồn

Trong đó:

- Đích là một trong các thanh ghi: AX, BX, CX, DX, SP, BP, SI, DI.
- Nguồn là ô nhớ trong đoạn DS được chỉ ra trong lệnh.

Đây là lệnh nạp vào thanh ghi đã chọn và vào DS từ 4 ô nhớ liên tiếp. Một trong những ứng dụng của lệnh này là làm cho SI và DS chỉ

vào địa chỉ đầu của vùng nhớ chứa chuỗi Nguồn trước khi đến lệnh thao tác chuỗi.

Các cờ bị thay đổi: Không.

Ví dụ:

```
LDS      SI, STR_PTR
```

Nạp vào thanh ghi SI nội dung 2 ô nhớ STR_PTR và STR_PTR+1 và nạp vào DS nội dung 2 ô nhớ STR_PTR+3 và STR_PTR+4. Các ô nhớ này đều nằm trong đoạn dữ liệu DS và chứa địa chỉ của chuỗi Nguồn. Do vậy, sau đó DS:SI chỉ vào đầu chuỗi Nguồn cần thao tác.

2. LEA – Load Effective Address (nạp địa chỉ hiệu dụng vào thanh ghi).

Dạng lệnh: LEA Đích, Nguồn

Trong đó:

- Đích là một trong các thanh ghi: BX, CX, DX, BP, SI, DI.
- Nguồn là tên biến trong đoạn DS được chỉ rõ trong lệnh hoặc ô nhớ cụ thể.

Đích←Địa chỉ lệnh của Nguồn, hoặc

Đích←Địa chỉ hiệu dụng của Nguồn.

Đây là lệnh để tính địa chỉ lệnh của biến hoặc địa chỉ của ô nhớ chọn làm Nguồn rồi nạp vào thanh ghi đã chọn.

Các cờ bị thay đổi: Không.

Ví dụ:

```
LEA  DX, Label ; nạp địa chỉ lech của Label vào DX
```

```
LEA  CX, [BX][DI]; nạp vào CX địa chỉ hiệu dụng
```

```
      ;do BX và DI chỉ ra EA=BX+DI
```


3. LES – Load register and ES with words from memory (nạp một từ (từ bộ nhớ) vào thanh ghi cho trong lệnh và một từ tiếp theo vào ES).

Dạng lệnh: LES Đích, Nguồn

Trong đó:

- Đích là một trong các thanh ghi: AX, BX, CX, DX, SP, BP, SI, DI.
- Nguồn là ô nhớ trong đoạn ES được chỉ ra trong lệnh.

Đây là lệnh nạp vào thanh ghi đã chọn và vào ES từ 4 ô nhớ liên tiếp. Một trong những ứng dụng của lệnh này là làm cho DI và ES chỉ vào địa chỉ đầu của vùng nhớ chứa chuỗi Nguồn trước khi đến lệnh thao tác chuỗi.

Các cờ bị thay đổi: Không.

Ví dụ:

LDS DI, [BX]

Nạp vào thanh ghi DI nội dung 2 ô nhớ BX và BX+1 và nạp vào ES nội dung 2 ô nhớ BX+3 và BX+4. Các ô nhớ này đều nằm trong đoạn dữ liệu ES và chứa địa chỉ của chuỗi Nguồn. Do vậy, sau đó ES:SI chỉ vào đầu chuỗi Nguồn cần thao tác.

4. MOV – Mov a byte or word (chuyển một byte hay từ)

Dạng lệnh: MOV Đích, Nguồn

Mô tả: Đích←Nguồn

Trong đó, toán hạng Đích và Nguồn có thể tìm được theo các chế độ địa chỉ khác nhau, nhưng phải có cùng độ dài và không được phép đồng thời là hai ô nhớ hoặc hai thanh ghi đoạn.

Các cờ bị thay đổi: Không.

Ví dụ:

MOV AL, AH ;AL←AH

MOV CX, 50 ;CX←50

MOV DL, [SI] ;DL←{DS:SI}

5. MOVS/MOVSMB/MOVSX – Move String byte or String word
(chuyển một phần tử của một chuỗi sang một chuỗi khác).

Dạng lệnh:

MOVS Chuỗi_Đích, Chuỗi_Nguồn

MOVSMB

MOVSX

Mô tả:

Phần tử chuỗi_Đích←phần tử chuỗi_Nguồn

Lệnh này dùng để chuyển từng byte hay từng từ của chuỗi Nguồn sang chuỗi Đích.

Các cờ bị thay đổi: Không.

Ví dụ:

CLD; xoa co huong lam viec voi chuoi theo chieu →

MOV DI, OFFSETChuoi_dich;lay dia chi lech
;cua chuoi_dich tai ES vao DI

MOV SI, OFFSET Chuoi_goc;lay dia chi lech
;cua chuoi_goc tai DS vao SI

MOVSMB;chuyen 1byte, SI va DI tang them 1

6. OUT – Output a byte or a word to a port.

Dạng lệnh: OUT Port, Acc

Mô tả: Acc→{Port}

- Nếu Acc là AL thì dữ liệu 8 bit được đưa ra cổng Port.
- Nếu Acc là AX thì dữ liệu 16 bit được đưa ra cổng Port và Port + 1.

Có một cách khác để chứa địa chỉ cổng là thông qua thanh ghi DX. Khi dùng thanh ghi DX để chứa địa chỉ cổng ta có khả năng địa chỉ hoá cổng mềm dẻo hơn. Lúc này địa chỉ cổng nằm trong dải 0000H... FFFFH và viết lệnh theo dạng:

OUT DX, Acc

Các cờ bị thay đổi: không.

Ví dụ:

OUT 45H, AL ;đưa dữ liệu từ AL ra cổng 45H

MOV DX, 00FFH; nạp địa chỉ cổng vào DX

OUT DX, AX; đưa dữ liệu từ AX ra 00FFH

7. POP – Pop word from top of Stack (lấy lại 1 từ vào thanh ghi từ đỉnh ngăn xếp)

Dạng lệnh: POP Đích

Mô tả:

Đích ← {SP}

SP ← SP + 2

Toán hạng Đích có thể là các thanh ghi đa năng, thanh ghi đoạn (nhưng không được là thanh ghi đoạn mã CS) hoặc ô nhớ.

Các cờ bị thay đổi: không.

Ví dụ:

POP DX; lấy 2 byte từ đỉnh ngăn xếp đưa vào DX

8. POPF – Pop word from top of Stack to Flag register (lấy 1 từ vào thanh ghi cờ từ đỉnh ngăn xếp).

Dạng lệnh: POPF

Mô tả:

$$FR \leftarrow \{SP\}$$
$$SP \leftarrow SP + 2$$

Chuyển các bit xác định của từ ở đỉnh của Stack (được SP trỏ tới) tới các cờ và bằng cách đó nó thay thế tất cả các cờ hiện thời.

Các cờ bị thay đổi: tất cả các cờ.

9. PUSH – Push word on the Stack (cất 1 từ vào ngăn xếp)

Dạng lệnh: PUSH Nguồn

Mô tả:

$$SP \leftarrow SP - 2$$
$$\text{Nguồn} \rightarrow \{SP\}$$

Toán hạng Đích có thể là các thanh ghi đa năng, thanh ghi đoạn (kể cả CS) hoặc ô nhớ.

Các cờ bị thay đổi: không.

Ví dụ:

PUSH BX;BX -> ngăn xếp tại vị trí do SP chỉ ra

10. PUSHF – Push Flag register to the Stack (cất thanh ghi cờ vào ngăn xếp)

Dạng lệnh: PUSHF

Mô tả:

$$SP \leftarrow SP - 2$$
$$FR \rightarrow \{SP\}$$

PUSH giảm SP đi 2 byte và nhờ đó nó chuyển tất cả các cờ tới đỉnh của Stack.

Các cờ bị thay đổi: không.

Ví dụ:

PUSH BX;BX -> ngăn xếp tại vị trí do SP chỉ ra

11. XCHG – Exchange (hoán đổi nội dung hai toán hạng)

Dạng lệnh: XCHG Đích, Nguồn

Mô tả: Đích $\leftrightarrow$ Nguồn

Toán hạng Đích và Nguồn phải có cùng độ dài, không được đồng thời là 2 ô nhớ cũng không được là thanh ghi đoạn.

Các cờ bị thay đổi: không.

Ví dụ:

XCHG AH, AL ; trao đổi nội dung AH và AL

XCHG AX, BX ; trao đổi nội dung AX và BX

2.3.2.2. Nhóm các lệnh tính toán số học

12. ADC – Add with Carry (cộng có nhớ)

Dạng lệnh: ADC Đích, Nguồn

Mô tả: Đích $\leftarrow$ Đích + Nguồn + CF

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF.

Ví dụ:

ADC CL, BL ; CL $\leftarrow$ CL + BL + CF

13. ADD – Add (cộng hai toán hạng)

Dạng lệnh: ADD Đích, Nguồn

Mô tả: Đích $\leftarrow$ Đích + Nguồn

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF.

Ví dụ:

ADD DX, CX ; DX $\leftarrow$ DX + CX

14. DEC – Decrement (giảm byte hay word đi một giá trị)

Dạng lệnh: DEC Đích

DEC trừ toán hạng Đích đi 1.

Các cờ bị thay đổi: AF, OF, PF, SF, ZF.

Ví dụ:

```
MOV BX, 1200H           ;chuyen 1200H vao BX
```

```
DEC BX                   ;BX=11FFH
```

15. DIV – Division (chia không dấu)

Dạng lệnh: DIV Nguồn

Toán hạng Nguồn là số chia. Tuỳ theo độ dài toán hạng Nguồn ta có hai trường hợp bố trí phép chia.

– Nếu Nguồn là số 8 bit: AX/Nguồn, thương để vào AL, số dư để vào AH.

– Nếu Nguồn là số 16 bit: DXAX/Nguồn, thương để vào AX, số dư để vào DX.

Các cờ bị thay đổi: không.

Ví dụ:

```
MOV AX, 0033H           ;chuyen 0033H vao AX
```

```
MOV BL, 25
```

```
DIV BL                   ;AL=02H va AH=01H
```

16. IDIV – Integer Division (chia có dấu)

Dạng lệnh: IDIV Nguồn

Thực hiện một phép chia có dấu thanh ghi tổng (và phần mở rộng của nó) cho toán hạng Nguồn.

– Sau phép chia AL (AX) chứa thương số (số có dấu), AH (DX) chứa số dư (số có dấu).

– Dấu của số dư trùng với dấu của số bị chia.

Các cờ bị thay đổi: không.

Ví dụ:

IDIV CL

17. IMUL – Integer Multiplication (nhân có dấu)

Dạng lệnh: IMUL Nguồn

– Nếu Nguồn là số có dấu 8 bit: ALxNguồn. Sau khi nhân $AX \leftarrow \text{tích}$.

– Nếu Nguồn là số có dấu 16 bit: AXxNguồn. Sau khi nhân $DXAX \leftarrow \text{tích}$.

Các cờ bị thay đổi: CF, OF.

Ví dụ:

IMUL CL

18. IN – Input data from a port (đọc dữ liệu từ cổng vào thanh ghi Acc).

Dạng lệnh: IN Acc, địa_chi_cổng

Lệnh IN truyền một byte hoặc một từ từ một cổng vào lần lượt tới thanh ghi AL hoặc AX. Địa chỉ của cổng có thể được xác định là một hằng tức thì kiểu byte cho phép truy nhập các cổng từ 0...255 hoặc thông qua một số đã được đưa ra trước đó trong thanh ghi DX mà cho phép truy nhập các cổng từ 0...65535.

Các cờ bị thay đổi: không.

Ví dụ:

IN AL, 45H ;doc mot byte tu mot cong

IN AX, DX ;doc mot tu tu mot cong

19. INC – Increment (tăng toán hạng lên 1)

Dạng lệnh: INC Đích

Mô tả: $\text{Đích} \leftarrow \text{Đích} + 1$

Các cờ bị thay đổi: AF, OF, PF, SF, ZF.

Ví dụ:

INC AL

20. MUL – Multiply unsigned byte or word (nhân số không dấu)

Dạng lệnh: MUL Nguồn

Nếu Nguồn là số 8 bit: $AL \times \text{Nguồn}$. Số bị nhân phải là số 8 bit đặt trong AL, sau khi nhân tích lưu vào AX.

Nếu Nguồn là số 16 bit: $AX \times \text{Nguồn}$. Số bị nhân phải là số 16 bit đặt trong AX, sau khi nhân tích lưu vào DXAX.

Các cờ bị thay đổi: CF, OF.

Ví dụ:

MUL CX ; $AX \times CX \rightarrow DXAX$

MUL BL ; $AL \times BL \rightarrow AX$

21. NEG – Negation (lấy bù hai của một toán hạng, đảo dấu của một toán hạng).

Dạng lệnh: NEG Đích

Mô tả: $\text{Đích} \leftarrow 0 - \text{Đích}$

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF

Ví dụ:

NEG AL ; $AL \leftarrow 0 - (AL)$

22. SBB – Subtract with Borrow (trừ có mượn).

Dạng lệnh: SBB Đích, Nguồn

Mô tả: $\text{Đích} \leftarrow \text{Đích} - \text{Nguồn} - CF$

Toán hạng Đích vào Nguồn phải chứa cùng một loại dữ liệu và không được đồng thời là hai ô nhớ, cũng không được là thanh ghi đoạn.

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF.

Ví dụ:

SBB AL, 78H ; AL ← AL - 78H - CF

23. SUB – Subtract (trừ hai toán hạng)

Dạng lệnh: SUB Đích, Nguồn

Mô tả: Đích ← Đích - Nguồn

Toán hạng Đích vào Nguồn phải chứa cùng một loại dữ liệu và không được đồng thời là hai ô nhớ, cũng không được là thanh ghi đoạn.

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF.

Ví dụ:

SUB AL, 78H ; AL ← AL - 78H

2.3.2.3. Nhóm các lệnh tính toán logic

24. AND (phép và logic)

Dạng lệnh: AND Đích, Nguồn

Mô tả: Đích ← Đích ^ Nguồn

Các cờ bị thay đổi: CF, OF, PF, SF, ZF.

Ví dụ:

AND DX, CX ; DX ← DX AND CX theo từng bit

25. NOT – Logical Negation (phủ định logic)

Dạng lệnh: NOT Đích

NOT đảo các giá trị của các bit của toán hạng Đích.

Các cờ bị thay đổi: không.

Ví dụ:

MOV AL, 02H ; AL = (0000 0010) B

NOT AL ; AL = (1111 1101) B

26. OR – Logic OR (phép hoặc logic)

Dạng lệnh: OR Đích, Nguồn

Mô tả: Đích = Đích $\vee$ Nguồn

Các cờ bị thay đổi: CF, OF, PF, SF, ZF.

Ví dụ:

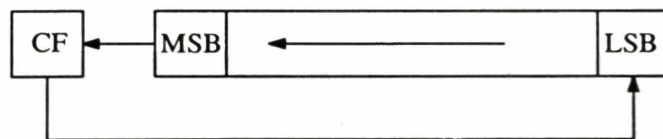
OR AX, BX ; AX $\leftarrow$ AX $\vee$ BX theo từng bit

2.3.2.4. Nhóm các lệnh dịch, quay toán hạng

27. RCL – Rotate though CF to the Left (quay trái thông qua cờ nhớ)

Dạng lệnh: RCL Đích, CL

Mô tả:



CL phải được chứa sẵn số lần quay. Trong trường hợp quay 1 lần có thể viết RCL Đích, 1

Các cờ bị thay đổi: CF, OF.

Ví dụ:

MOV CL, 3 ; số lần quay là 3

RCL AL, CL

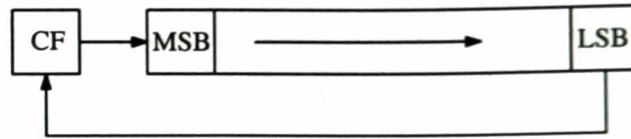
Trước khi thực hiện lệnh: AL = 01011110, CF = 0.

Sau khi thực hiện lệnh: AL = 11110001, CF = 0.

28. RCR – Rotate though CF to the Right (quay phải thông qua cờ nhớ)

Dạng lệnh: RCR Đích, CL

Mô tả:



CL phải được chứa sẵn số lần quay. Trong trường hợp quay 1 lần có thể viết RCR Đích, 1

Các cờ bị thay đổi: CF, OF.

Ví dụ:

```
MOV CL, 2 ;so lan quay la 2
```

```
RCR AL, CL
```

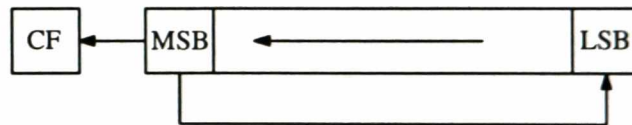
Trước khi thực hiện lệnh: AL = 11000010, CF = 1.

Sau khi thực hiện lệnh: AL = 01110000, CF = 1.

29. ROL – Rotate all bit to the Left (quay vòng sang trái).

Dạng lệnh: ROL Đích, CL.

Mô tả:



CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết ROL Đích, 1

Các cờ bị thay đổi: CF, OF.

Ví dụ:

```
MOV CL, 2 ;so lan quay la 2
```

```
ROL AL, CL
```

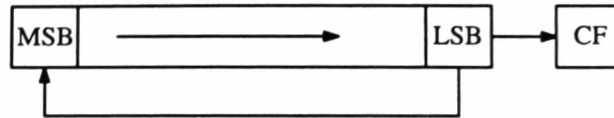
Trước khi thực hiện lệnh: AL = 11001100, CF = 1

Sau khi thực hiện lệnh: AL = 00110011, CF = 1

30. ROR – Rotate all bit to the Left (quay vòng sang phải).

Dạng lệnh: ROR Đích, CL

Mô tả:



CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết ROR Đích, 1

Các cờ bị thay đổi: CF, OF.

Ví dụ:

```
MOV CL, 2 ;so lan quay la 2
```

```
ROR AL, CL
```

Trước khi thực hiện lệnh: AL = 11001100, CF = 0

Sau khi thực hiện lệnh: AL = 00110011, CF = 0

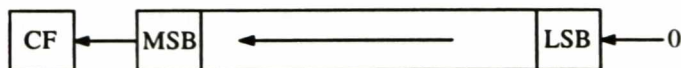
31. SAL/SHL - Shift Arithmetically Left (dịch trái số học)/Shift Logically Left (dịch trái logic).

Dạng lệnh:

SAL Đích, CL

SHL Đích, CL

Mô tả:



CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết SAL Đích, 1

Các cờ bị thay đổi: SF, ZF, CF, OF, PF.

Ví dụ:

```
MOV CL, 2 ;so lan quay la 2
```

```
SAL AL, CL
```

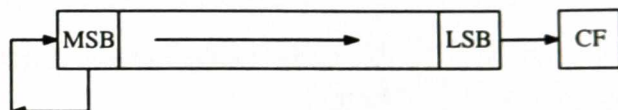

Trước khi thực hiện lệnh: $AL = 11001100, CF = 0$

Sau khi thực hiện lệnh: $AL = 00110000, CF = 1$

32. SAR - Shift Arithmetically Right (dịch phải số học).

Dạng lệnh: SAR Đích, CL

Mô tả:



CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần thì ta có thể viết SAR Đích, 1.

Các cờ bị thay đổi: SF, ZF, CF, OF, PF.

Ví dụ:

```
MOV CL, 2 ;so lan quay la 2
```

```
SAR AL, CL
```

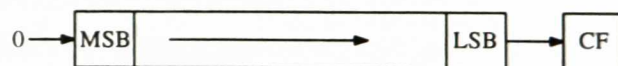
Trước khi thực hiện lệnh: $AL = 11001100, CF = 1$

Sau khi thực hiện lệnh: $AL = 11110011, CF = 0$

33. SHR – Shift logically Right (dịch phải logic)

Dạng lệnh: SHR Đích, CL

Mô tả:



CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết SHR Đích, 1

Các cờ bị thay đổi: SF, ZF, CF, OF, PF.

Ví dụ:

```
MOV CL, 2 ;so lan quay la 2
```

```
SHR AL, CL
```

Trước khi thực hiện lệnh: AL = 11001100, CF = 1

Sau khi thực hiện lệnh: AL = 00110011, CF = 0

34. TEST – Logic Comparison (lệnh so sánh logic)

Dạng lệnh: TEST Đích, Nguồn

Mô tả: Đích $\wedge$ Nguồn

TEST thực hiện phép so sánh hai toán hạng (dạng byte hoặc từ) và cập nhật các cờ nhưng không trả lại kết quả (tức là không có toán hạng nào thay đổi, không lưu kết quả). Nếu phía sau lệnh TEST là lệnh JNZ (nhảy nếu khác 0), thì một lệnh nhảy sẽ được thực hiện nếu có một cặp các bit tương ứng đều bằng 1.

Các cờ bị thay đổi: CF, OF, PF, SF, ZF.

Ví dụ:

TEST AL, 3FH

TEST AX, 0034H

35. XOR – Exclusive OR (lệnh logic XOR (hoặc đảo)).

Dạng lệnh: XOR Đích, Nguồn

Mô tả: Đích $\leftarrow$ Đích $\oplus$ Nguồn.

Các cờ bị thay đổi: CF, OF, PF, SF, ZF.

Ví dụ:

XOR AX, AX

XOR BX, BX

MOV AX, 5857H

MOV BX, 58A8H

XOR AX, BX

Trước khi thực hiện lệnh XOR

AX=5857H

BX=58A8H

Sau khi thực hiện lệnh XOR

AX=00FFH

BX=58A8H

2.3.2.5. Nhóm các lệnh so sánh

36. CMP – Compare (so sánh)

Dạng lệnh: CMP Đích, Nguồn

CMP trừ toán hạng Đích cho toán hạng Nguồn, nhưng không lưu trữ kết quả. Các toán hạng không bị thay đổi. Kết quả của lệnh này dùng để cập nhật các cờ và có thể được dùng để làm điều kiện cho các lệnh nhảy có điều kiện tiếp theo.

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF.

Các cờ chính theo quan hệ Đích và Nguồn khi so sánh hai số không dấu.

So sánh	CF	ZF
Đích = Nguồn	0	1
Đích > Nguồn	0	0
Đích < Nguồn	1	0

37. CMPS/CMPSB/CMPSW – Compare String Bytes or String Words (so sánh hai chuỗi byte hay hai chuỗi từ).

Dạng lệnh:

CMPS Chuỗi_Đích, Chuỗi_Nguồn

CMPSB

CMPSW

– Lệnh này so sánh từng phần tử (byte hay từ) của hai xâu có các phần tử cùng loại. Lệnh chỉ tạo các cờ, không lưu kết quả so sánh, sau khi so sánh các toán hạng không bị thay đổi.

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF.

Ví dụ:

```
StrByte1      DB      "chuoi byte1"
StrByte2      DB      "chuoi byte2"
StrWord1      DW      "chuoi word1"
StrWord2      DW      "chuoi word2"

LEA SI, StrByte1      ;nap gia tri hieu dung
                        ;StrByte1 vao SI

LEA DI, StrByte2      ;nap gia tri hieu dung
                        ;StrByte2 vaoDI

COMPS StrByte2, StrByte1      ;co the thay bang
                                ;COMPSB

LEA SI, StrWord1      ;nap gia tri hieu dung
                        ;StrWord1 vao SI

LEA DI, StrWord2      ;nap gia tri hieu dung
                        ;StrWord2 vaoDI

COMPS StrWord2,StrWord      ;co the thay bang COMPSW
```

2.3.2.6. Nhóm các lệnh nhảy (rẽ nhánh)

38. JA/JNBE – Jump if Above/Jump if Not Below or Equal (nhảy nếu cao hơn/nhảy nếu không thấp hơn hoặc bằng).

Dạng lệnh:

JA	NHAN
JNBE	NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu $CF + ZF = 0$. Quan hệ cao hơn/thấp hơn là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn hai số không dấu.

Các cờ bị thay đổi: không.

Ví dụ:

CMP AX, 12ABH; so sánh AX với 12ABH

JA THOI ;nhảy đến THOI nếu AX cao hơn 12ABH

39. JAE/JNB/JNC – Jump if Above or Equal/Jump if Not Below/Jump if No Carry (nhảy nếu lớn hơn hoặc bằng/nhảy nếu không thấp hơn/nhảy nếu không có nhớ).

Dạng lệnh:

JAE NHAN

JNB NHAN

JNC NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Ba lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu $CF = 0$. Quan hệ cao hơn/thấp hơn là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn hai số không dấu.

Các cờ bị thay đổi: không.

Ví dụ:

CMP AL, 10H ;so sánh AL với 10H

JAE THOI;Đến THOI nếu AL cao hơn hoặc bằng 10H

40. JB/JC/JNAE – Jump if Below/Jump if Carry/Jump if Not Above or Equal (nhảy nếu thấp hơn/nhảy nếu có nhớ/nhảy nếu không cao hơn hoặc bằng).

Dạng lệnh:

JB NHAN

JC NHAN

JNAE NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Ba lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu $CF = 1$. Quan hệ cao hơn/thấp hơn là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn hai số không dấu.

Các cờ bị thay đổi: không.

Ví dụ:

CMP AL, 10H ;so sanh AL voi 10H

JB THOI ;nhay den THOI neu AL thap hon 10H

41. JBE/JNA – Jump if Below or Equal/Jump if Not Above (nhảy nếu thấp hơn hoặc bằng/nhảy nếu không cao hơn).

Dạng lệnh:

JBE NHAN

JNA NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu $CF + ZF = 1$. Quan hệ cao hơn/thấp là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn hai số không dấu.

Các cờ bị thay đổi: không.

Ví dụ:

CMP AL, 10H ;so sanh AL voi 10H

JBE THOI;Den THOI neu AL thap hon hoac bang 10H

42. JCXZ – Jump if CX register is Zero (nhảy nếu nội dung thanh đếm CX rỗng).

Dạng lệnh: JCXZ NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Lệnh trên biểu diễn thao tác nhảy có điều kiện tới NHAN nếu $CX = 0$ và không liên quan đến ZF.

Các cờ bị thay đổi: không.

Ví dụ:

JCXZ THOI ;nhay den THOI neu $CX = 0$

43. JE/JZ – Jump if Equal/Jump if Zero (nhảy nếu bằng nhau/nhảy nếu kết quả bằng không)

Dạng lệnh:

JE NHAN

JZ NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Lệnh trên biểu diễn thao tác nhảy có điều kiện tới NHAN nếu $ZF = 1$.

Các cờ bị thay đổi: không.

Ví dụ:

SUB AL, 10H ;tru AL cho 10H

JE THOI ;nhay den THOI neu AL bang 10H

44. JG/JNLE – Jump if Greater than/Jump if Not Less than or Equal (nhảy nếu lớn hơn/nhảy nếu không bé hơn hoặc bằng)

Dạng lệnh:

JG NHAN

JNLE NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu $(SF \text{ xor } OF) + ZF = 0$. Quan hệ lớn hơn/bé hơn là quan hệ

dành cho việc so sánh (do lệnh CMP thực hiện) của hai số có dấu. Lớn hơn có nghĩa là dương hơn.

Các cờ bị thay đổi: không.

Ví dụ:

CMP AL, 10H ;so sanh AL voi 10H

JG THOI ;nhay den THOI neu AL lon hon 10H

45. JGE/JNL – Jump if Greater than or Equal/Jump if Not Less than (nhảy nếu lớn hơn hoặc bằng/nhảy nếu không nhỏ hơn)

Dạng lệnh:

JGE NHAN

JNL NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu $(SF \text{ xor } OF) = 0$. Quan hệ lớn hơn/bé hơn là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) của hai số có dấu. Lớn hơn có nghĩa là dương hơn.

Các cờ bị thay đổi: không.

Ví dụ:

CMP AL, 10H ;so sanh AL voi 10H

JGE THOI;Den THOI neu AL lon hon hoac bang 10H

46. JL/JNGE – Jump if Less than/Jump if Not Greater than or Equal (nhảy nếu bé hơn/nhảy nếu không lớn hơn hoặc bằng).

Dạng lệnh:

JL NHAN

JNGE NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu $(SF \text{ xor } OF) = 1$. Quan hệ lớn hơn/bé hơn là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) của hai số có dấu. Lớn hơn có nghĩa là dương hơn.

Các cờ bị thay đổi: không.

Ví dụ:

CMP AL, 10H ;so sanh AL voi 10H

JL THOI ;nhay den THOI neu AL nho hon 10H

47. JLE/JNG – Jump if Less than or Equal/Jump if Not Greater than (nhảy nếu nhỏ hơn hoặc bằng/nhảy nếu không lớn hơn)

Dạng lệnh:

JLE NHAN

JNG NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu $(SF \text{ xor } OF) + ZF = 1$. Quan hệ lớn hơn/bé hơn là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) của hai số có dấu. Lớn hơn có nghĩa là dương hơn.

Các cờ bị thay đổi: không.

Ví dụ:

CMP AL, 10H ;so sanh AL voi 10H

JLE THOI;Den THOI neu AL nho hon hoac bang 10H

48. JMP – Unconditional Jump (lệnh nhảy không điều kiện).

JMP trao quyền điều khiển cho vùng mục tiêu một cách không điều kiện. Lệnh này có các chế độ giống như lệnh CALL.

Dạng lệnh: Sau đây là những cách viết lệnh không điều kiện.

JMP NHAN

Các cờ bị thay đổi: không.

49. JNE/JNZ – Jump if Not Equal/Jump if Not Zero (nhảy nếu không bằng nhau/nhảy nếu kết quả không rỗng).

Dạng lệnh:

JNE NHAN

JNZ NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu $ZF = 0$.

Các cờ bị thay đổi: không.

Ví dụ:

CMP AL, 10H ;so sanh AL voi 10H

JNE THOI ;nhay den THOI neu AL khac 10H

50. JNO – Jump if Not Overflow (nhảy nếu không tràn)

Dạng lệnh: JNO NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Đây là lệnh nhảy có điều kiện tới NHAN nếu $OF = 0$.

Các cờ bị thay đổi: không.

Ví dụ:

MUL AL, BL ;nhan AL voi BL

JNO THOI ;nhay den THOI neu khong tran

51. JNP/JPO – Jump if Not Parity/Jump if Parity Odd (nhảy nếu parity lẻ).

Dạng lệnh:

JNP NHAN

JPO NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu $PF = 0$.

Các cờ bị thay đổi: không.

Ví dụ:

```
IN    AL, 98H ;doc ky tu tu cong
OR    AL, AL  ;tao co
JNP   THOI    ;nhay den THOI neu parity le
```

52. JNS - Jump Not Signed (nhảy nếu kết quả dương).

Dạng lệnh: JNS NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Đây là lệnh nhảy có điều kiện tới NHAN nếu $SF = 0$. Kết quả là dương sau khi thực hiện các phép toán có dấu.

Các cờ bị thay đổi: không.

Ví dụ:

```
SUB   AL, AH  ;tru hai so co dau AL cho AH
JNS   THOI    ;nhay den THOI neu ket qua duong
```

53. JO – Jump if Overflow (nhảy nếu tràn)

Dạng lệnh: JO NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Đây là lệnh nhảy có điều kiện tới NHAN nếu $OF = 1$. Tức là xảy ra tràn sau khi thực hiện các phép toán có dấu.

Các cờ bị thay đổi: không.

Ví dụ:

```
ADD   AL, AH  ;cong AL voi AH
JO    THOI    ;nhay den THOI neu co tran
```

54. JP/JPE – Jump if Parity/Jump if Parity Even (nhảy nếu parity chẵn)

Dạng lệnh:

JP NHAN

JPE NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu $PF = 1$.

Các cờ bị thay đổi: không.

Ví dụ:

IN AL, 99H ;doc ky tu tu cong

OR AL, AL ;tao co

JP THOI ;nhay den THOI neu parity chan

55. JS – Jump if Sign (nhảy nếu âm)

Dạng lệnh: JS NHAN

Mô tả: $IP \leftarrow IP + \text{dịch chuyển}$

Đây là lệnh nhảy có điều kiện tới NHAN nếu $SF = 1$. Kết quả là âm sau khi thực hiện các phép toán có dấu.

Các cờ bị thay đổi: không.

Ví dụ:

SUB AL, AH ;tru hai so co dau AL cho AH

JS THOI ;nhay den THOI neu ket qua duong

2.3.2.7. Nhóm các lệnh lặp

56. LOOP – Loop if CX is not 0 (lặp nếu $CX \neq 0$)

Dạng lệnh: LOOP NHAN

Mô tả: Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ NHAN đến hết lệnh LOOP NHAN) cho đến khi số lần lặp CX = 0. Điều này có nghĩa là trước khi vào vòng lặp ta phải đưa số lần lặp mong muốn vào CX, và sau mỗi lần lặp thì CX tự động giảm đi 1.

Các cờ bị thay đổi: không.

Ví dụ:

```
MOV AL, 0      ;xoa AL
MOV CX, 10     ;nap so lan lap vao CX
LAP:INC AL     ;tang AL len 1
LOOP LAP       ;lap lai 10 lan, AL=10
```

57. LOOPE/LOOPZ – Loop while CX = 0 or ZF = 0 (lặp lại đoạn chương trình cho đến khi CX = 0 hoặc ZF = 0).

Dạng lệnh:

LOOPE NHAN

LOOPZ NHAN

Mô tả: Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ NHAN đến hết lệnh LOOPE NHAN hoặc LOOPZ NHAN) cho đến khi số lần lặp CX = 0 hoặc cờ ZF = 0. Điều này có nghĩa là trước khi vào vòng lặp ta phải đưa số lần lặp mong muốn vào CX, và sau mỗi lần lặp thì CX tự động giảm đi 1.

Các cờ bị thay đổi: không.

Ví dụ:

```
MOV AL, AH     ;AL=AH
MOV CX, 50     ;nap so lan lap vao CX
LAP:INC AL     ;tang AL
CMP AL, 16     ;so sanh AL voi 16
```

LOOPE LAP;lap lai cho den khi $AL \neq 16$ hoac $CX = 0$

58. LOOPNE/LOOPNZ – Loop while $CX = 0$ or $ZF = 1$ (lặp lại đoạn chương trình cho đến khi $CX = 0$ hoặc $ZF = 1$).

Dạng lệnh:

LOOPNE NHAN

LOOPNZ NHAN

Mô tả: Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ NHAN đến hết lệnh LOOPNE NHAN hoặc LOOPNZ NHAN) cho đến khi số lần lặp $CX = 0$ hoặc cờ $ZF = 1$. Điều này có nghĩa là trước khi vào vòng lặp ta phải đưa số lần lặp mong muốn vào CX, và sau mỗi lần lặp thì CX tự động giảm đi 1.

Các cờ bị thay đổi: không.

Ví dụ:

MOV AL, AH; $AL=AH$

MOV CX, 50; nạp số lần lặp vào CX

LAP: INC AL ; tăng AL

COMP AL, 16; so sánh AL với 16

LOOPNE LAP ; lặp lại cho đến khi $AL=16$ hoặc $CX = 0$

59. REP – Repeat String Instruction until $CX = 0$ (lặp lại lệnh viết sau đó cho tới khi $CX = 0$).

Đây là tiếp đầu ngữ dùng để viết trước các lệnh thao tác với chuỗi dữ liệu mà ta muốn lặp lại một số lần. Số lần lặp phải để trước trong CX. Khi các lệnh này được thực hiện thì CX tự động giảm đi 1. Quá trình lặp kết thúc khi $CX = 0$.

Các cờ bị thay đổi: không.

Ví dụ:

```
REP MOVSB      ;lap lai lenh chuyen byte cua
                ;chuoi toi khi CX = 0
```

- 60. REPE/REPZ** – Repeat String Instruction until CX = 0 or ZF = 0
(lặp lại lệnh viết sau đó cho tới khi CX = 0 hoặc ZF = 0).

Đây là tiếp đầu ngữ dùng để viết trước các lệnh thao tác với chuỗi dữ liệu mà ta muốn lặp lại một số lần. Số lần lặp phải để trước trong CX. Khi các lệnh này được thực hiện thì CX tự động giảm đi 1. Quá trình lặp kết thúc khi CX = 0 hoặc khi hai phần tử so sánh khác nhau (ZF = 0).

Các cờ bị thay đổi: không.

Ví dụ:

```
REPE CMPSB;lập lại lệnh so sánh các byte của
                ;chuoi toi khi CX = 0 hoặc ZF = 0
```

- 61. REPNE/REPZ** – Repeat String Instruction until CX = 0 or ZF = 1 (lặp lại lệnh viết sau đó cho tới khi CX = 0 hoặc ZF = 1).

Đây là tiếp đầu ngữ dùng để viết trước các lệnh thao tác với chuỗi dữ liệu mà ta muốn lặp lại một số lần. Số lần lặp phải để trước trong CX. Khi các lệnh này được thực hiện thì CX tự động giảm đi 1. Quá trình lặp kết thúc khi CX = 0 hoặc khi Acc bằng phần tử của chuỗi (ZF = 1).

Các cờ bị thay đổi: không.

Ví dụ:

```
REPNE SCASB;lập lại lệnh quét các byte của
                ;chuoi toi khi het chuoi (CX = 0)
                ;hoac AL bang mot phan tu cua chuoi
```

2.3.2.8. Nhóm các lệnh điều khiển, đặc biệt khác

62. CALL – Call a procedure (gọi chương trình con)

Dạng lệnh: CALL Tên_Chương_Trình_con

Mô tả: Lệnh này dùng để chuyển hoạt động của vi xử lý từ chương trình chính (CTC) sang chương trình con (ctc). Vi xử lý sẽ ngừng công việc hiện tại và căn cứ vào vị trí tên chương trình con nhảy đến thực hiện.

Các cờ bị thay đổi: AF, CF, OF, PF, SF, ZF.

Ví dụ:

CALL NEAR

CALL FAR

63. CLC – Clear the Carry flag (xoá cờ nhớ)

Dạng lệnh: CLC

Xoá cờ nhớ CF và không làm ảnh hưởng đến các cờ khác.

Các cờ bị thay đổi: CF.

64. CLD – Clear the Direction flag (xoá cờ hướng)

Dạng lệnh: CLD

Xoá cờ hướng DF và không làm ảnh hưởng đến các cờ khác.

Các cờ bị thay đổi: DF.

65. CLI – Clear the Interrupt flag (xoá cờ ngắt)

Dạng lệnh: CLD

Xoá cờ ngắt IF và không làm ảnh hưởng đến các cờ khác. Các yêu cầu ngắt che được sẽ bị che.

Các cờ bị thay đổi: IF.

66. CMC – Complement the Carry flag (đảo cờ nhớ).

Dạng lệnh: CMC

Mô tả: $CF = \overline{CF}$. Đảo cờ nhớ CF

Các cờ bị thay đổi: CF

67. HLT – Halt processing (dừng)

Dạng lệnh: HLT

Khi gặp lệnh này, các hoạt động của vi xử lý 8086 bị tạm dừng và bước vào trạng thái dừng. Để thoát khỏi trạng thái dừng chỉ có cách tác động vào một trong các chân INTR, NMI, RESET của bộ vi xử lý.

Các cờ bị thay đổi: không.

68. INT – Interrupt (lệnh gọi ngắt)

Dạng lệnh: INT N (N=0...FFH)

Khi gặp lệnh này vi xử lý sẽ ngừng mọi công việc hiện tại và nhảy đến địa chỉ bảng vectơ ngắt để thực hiện chương trình con ngắt.

69. IRET – Interrupt Return (trở về CTC từ ctc phục vụ ngắt)

Dạng lệnh: IRET

Trở về chương trình chính từ chương trình con phục vụ ngắt.

70. NOP – No Operation (CPU không làm gì)

Dạng lệnh: NOP

Lệnh này không thực hiện một công việc gì ngoài việc làm tăng nội dung của IP và tiêu tốn ba chu kỳ đồng hồ. Nó thường được dùng để tính thời gian trễ trong các vòng trễ hoặc để chiếm chỗ các lệnh cần thêm vào chương trình sau này mà không làm ảnh hưởng đến độ dài chương trình.

Các cờ bị thay đổi: không.

71. RET – Return from Procedure to Calling Program (trở về chương trình chính từ chương trình con).

Dạng lệnh: RET hoặc RET N (N là số nguyên dương)

Mô tả: RET được đặt cuối ctc để vi xử lý lấy lại địa chỉ trở về mà nó đã được tự động cất tại ngăn xếp khi có lệnh gọi ctc.

Các cờ bị thay đổi: không.

72. STC – Set the Carry Flag (lập cờ nhớ)

Dạng lệnh: STC

Mô tả: $CF \leftarrow 1$

STC thiết lập cờ nhớ bằng 1 và không ảnh hưởng đến các cờ khác.

Các cờ bị thay đổi: $CF = 1$.

73. STD – Set the Direction Flag (lập cờ hướng).

Dạng lệnh: STD

Mô tả: $DF \leftarrow 1$

STD thiết lập cờ hướng bằng 1 và không ảnh hưởng đến các cờ khác. Các cờ bị thay đổi: $DF = 1$.

74. STI – Set the Interrupt Flag (lập cờ cho phép ngắt)

Dạng lệnh: STI

Mô tả: $IF \leftarrow 1$

Lệnh này lập cờ cho phép ngắt để cho phép các yêu cầu ngắt tác động vào chân INTR được CPU nhận biết. Khi $IF = 1$ nếu có tín hiệu $INTR = 1$ thì 8086 sẽ bị ngắt.

Các cờ bị thay đổi: $IF = 1$

75. WAIT – Wait for TEST or INTR Signal (chờ tín hiệu từ chân TEST hoặc INTR).

Dạng lệnh: WAIT

Mô tả: Lệnh này đưa bộ vi xử lý 8086 vào trạng thái nghỉ và nó sẽ ở trạng thái này cho đến khi có tín hiệu ở mức thấp tác động vào chân TEST hoặc khi có tín hiệu ở mức cao tác động vào chân INTR.

Nếu có yêu cầu ngắt và yêu cầu này được phép tác động vào chân INTR thì sau khi chương trình phục vụ ngắt được thực hiện nó sẽ lại trở về trạng thái nghỉ. Lệnh này dùng để đồng bộ hoá hoạt động của 8086 với các bộ đồng xử lý bên ngoài.

Các cờ bị thay đổi: không.

2.4. CÁC MẠCH PHỤ TRỢ

2.4.1. MẠCH TẠO XUNG NHỊP

Cho dù làm việc trong chế độ MIN hay MAX thì 8086 luôn cần xung nhịp (xung đồng hồ) từ mạch tạo xung nhịp 8284. Mạch tạo xung nhịp không những cung cấp xung nhịp với tần số thích hợp cho toàn hệ thống mà nó còn ảnh hưởng tới việc đồng bộ tín hiệu RESET và tín hiệu READY của CPU.

CSYNC	1	18	Vcc
PCLK	2	17	X1
AEN1	3	16	X2
RDY1	4	15	ASYNC
READY	5	14	EFI
RDY2	6	13	F/C
AEN2	7	12	OSC
CLK	8	11	RES
GND	9	10	RESET

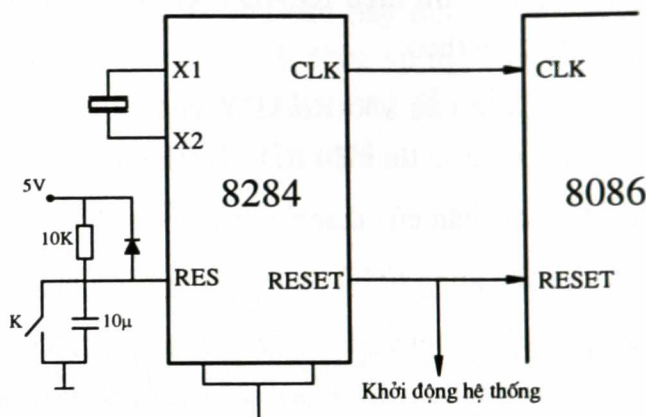
Hình 2.4. Sơ đồ bố trí chân của 8284

Ý nghĩa các chân tín hiệu:

- AEN1, AEN2: Tín hiệu cho phép chọn đầu vào tương ứng RDY1, RDY2 làm tín hiệu báo tình trạng sẵn sàng của bộ nhớ hoặc thiết bị ngoại vi.
- RDY1, RDY2: Cùng với AEN1, AEN2 dùng để gây ra các chu kỳ đợi.
- ASYNC: Chọn đồng bộ hai tầng hoặc đồng bộ một tầng cho tín hiệu RDY1, RDY2. Trong chế độ đồng bộ một tầng (ASYNC = 1) tín hiệu RDY có ảnh hưởng đến tín hiệu READY tới tận sườn xuống của xung đồng hồ tiếp theo, còn trong chế độ đồng bộ hai tầng (ASYNC = 0) tín hiệu RDY

có ảnh hưởng đến tín hiệu READY khi có sườn xuống của xung đồng hồ tiếp theo.

- READY: Nối đến đầu vào READY của CPU. Tín hiệu này được đồng bộ với các tín hiệu RDY1, RDY2.
- X1, X2: Nối hai chân của thạch anh với tần số f_x .
- EFI: Lối vào cho xung từ bộ dao động ngoài.
- F/C: Dùng để chọn nguồn tín hiệu chuẩn cho 8284. Khi chân này ở mức cao thì xung đồng hồ bên ngoài sẽ được làm xung nhịp cho 8284, ngược lại thì xung đồng hồ của mạch dao động bên trong dùng thạch anh sẽ được chọn để làm xung nhịp.
- CLK: Xung nhịp $f_{CLK} = f_x/3$ với độ rộng 77% nối đến chân CLK của 8086.
- PCLK: Xung nhịp $f_{CLK} = f_x/6$ với độ rộng 50% dành cho thiết bị ngoại vi.
- OSC: Xung nhịp đã được khuếch đại có tần số bằng f_x của bộ dao động.
- RES: Chân khởi động nối vào mạch R - C để 8284 có thể khởi động khi bật nguồn.
- RESET: Lối vào RESET của 8086 là tín hiệu khởi động lại cho toàn hệ.
- CSYNC: Lối vào cho xung đồng hồ chung khi trong hệ thống có các 8284 dùng dao động ngoài tại chân EFI. Khi dùng mạch dao động trong thì phải nối đất chân này.



Hình 2.5. Sơ đồ nối 8284 với 8086

2.4.2. MẠCH CHỐT VÀ ĐỆM BUS

2.4.2.1. Mạch chốt (Latch)

Các mạch chốt là các mạch mà khi tín hiệu cho phép chốt ALE (Address Latch Enable) ở mức L (Low) thì các đầu ra sẽ giữ nguyên trạng thái như trạng thái của các tín hiệu đầu vào trước khi chốt, lúc này cho dù các đầu vào thay đổi thì các đầu ra vẫn không thay đổi. Muốn thay



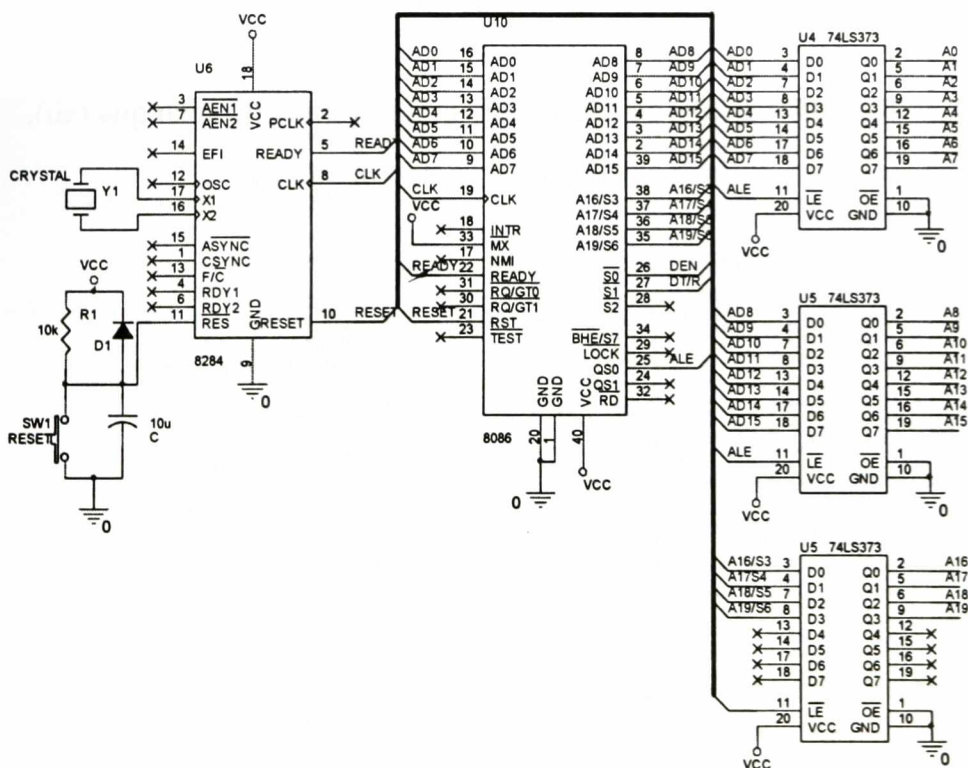
Hình 2.6.

đổi trạng thái các đầu ra thì ALE phải ở mức H (Hight). Vì mạch chốt sử dụng phổ biến trong các hệ vi xử lý hiện nay là 74LS373.

Bảng 2.5

Điều khiển ra (OE)	Cho phép chốt (LE)	Tín hiệu vào (D)	Tín hiệu ra (Q)
L	H	H	H
L	H	L	L
L	L	*	Không đổi
H	*	*	Trở kháng cao

Vi mạch 74LS373 thường được sử dụng để chốt và đệm các đường dây địa chỉ với các bus được dồn chân. Chú ý là thông tin địa chỉ chỉ tồn tại trên bus trong một thời gian ngắn. Mạch chốt được điều khiển bởi tín hiệu ALE để giữ lại thông tin địa chỉ đó. Hình vẽ 2.6 là sơ đồ nguyên lý của bộ chốt tín hiệu dùng 74LS373. Khi chân LE ở mức cao và chân OE ở mức thấp các tín hiệu vào được chuyển tới đầu ra. Lúc này 74LS373 làm việc như bộ đệm. Khi tín hiệu trên ALE chuyển từ cao xuống thấp, các tín hiệu đầu vào D sẽ được chốt và ở trạng thái này, các tín hiệu đầu ra được giữ nguyên cho dù tín hiệu đầu vào có thay đổi. Hình 2.7 trình bày sơ đồ ghép nối 8086 với mạch dao động 8284 và chốt 74LS373.



Hình 2.7. Sơ đồ ghép nối 8086 với vi mạch chốt 74LS373

2.4.2.2. Bộ đệm Bus

Mục đích của việc sử dụng đệm là để tăng công suất truyền tải cho Bus

Bộ đệm một chiều 74LS244 (hình 2.8):

$\overline{1OE} = 0$: Cho phép các tín hiệu đầu vào nhóm 1 (A1 - A4) được chuyển đến đầu ra nhóm 1 (Y1 - Y4).

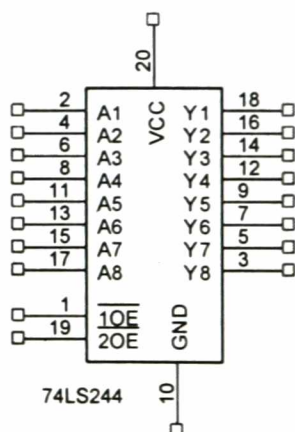
$\overline{1OE} = 1$: Đầu ra nhóm 1 ở trạng thái trở kháng cao Hi-Z.

$\overline{2OE} = 0$: Cho phép các tín hiệu đầu vào nhóm 2 (A5 - A8) được chuyển đến đầu ra nhóm 2 (Y5 - Y8).

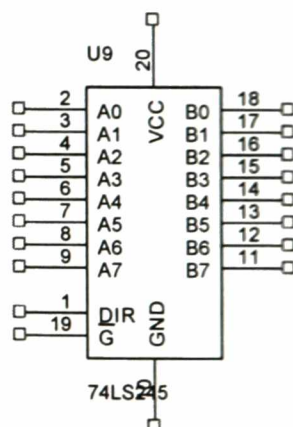
$\overline{2OE} = 1$: Đầu ra nhóm 2 ở trạng thái trở kháng cao Hi - Z.

Bộ đệm số liệu hai chiều 74LS245 (hình 2.9)

- DIR = 1: Dữ liệu được truyền từ A sang B (từ trái qua phải).
- DIR = 0: Dữ liệu được truyền từ B sang A (từ phải qua trái).
- A₀ - A₇ và B₀ - B₇: Các chân trao đổi dữ liệu.

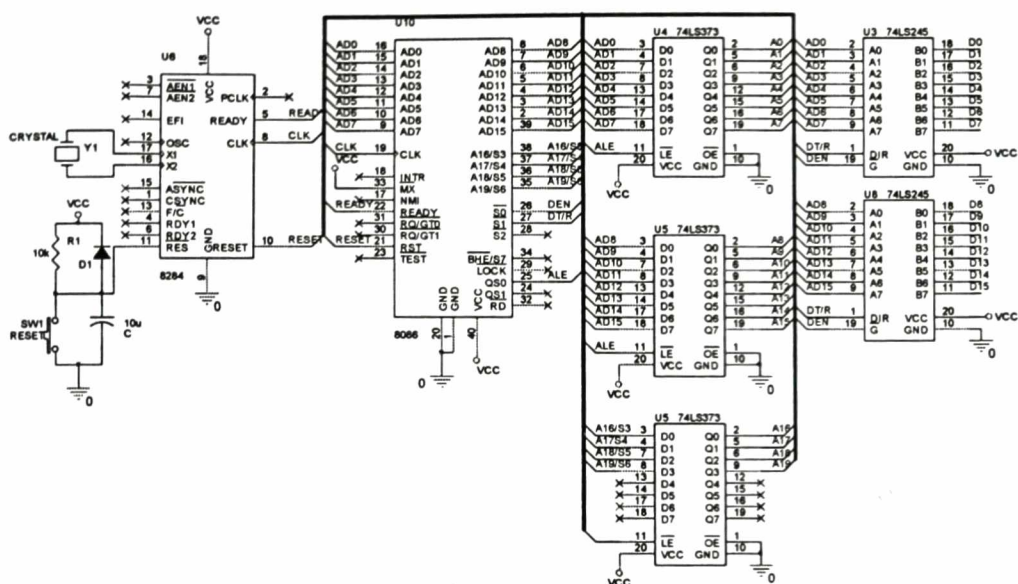


Hình 2.8. Đệm một chiều 74LS244



Hình 2.9. Đệm hai chiều 74LS245

Hình 2.10. Trình bày sơ đồ ghép nối 8086 với mạch chốt và đệm (trong đó có sử dụng 3 vi mạch chốt 74LS373 để tách 20 đường bus dồn kênh).



Hình 2.10. Sơ đồ ghép nối 8086 với mạch Reset 8284, chốt 74LS373, đệm 74LS245

2.4.3. CÁC BỘ NHỚ

Bộ nhớ dùng để lưu trữ mã lệnh và dữ liệu. Bộ nhớ được ghép nối trực tiếp với CPU và là nơi đầu tiên được CPU lấy thông tin. Các bộ nhớ bán dẫn có tốc độ truy nhập cao, có dung lượng lớn, kích thước nhỏ và tiêu thụ năng lượng thấp. Chúng được chế tạo thành những chip riêng biệt hoặc được tích hợp ngay bên trong CPU. Sau đây là các bộ nhớ thường được sử dụng trong bộ vi xử lý.

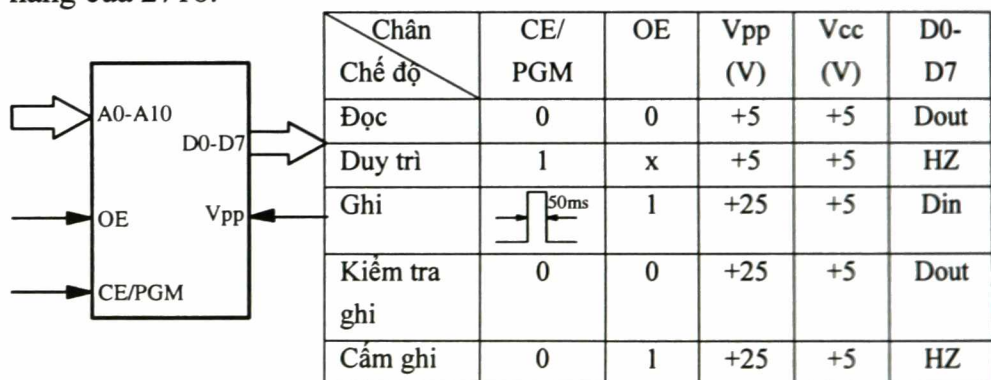
2.4.3.1. Bộ nhớ cố định ROM (Read Only Memory)

– **ROM:** Được gọi là vi mạch nhớ cố định vì số liệu do hãng chế tạo nạp một lần và thông tin sẽ được lưu trữ vĩnh viễn, sau đó chỉ có thể đọc ra mà thôi. Khi nguồn điện mất thì số liệu vẫn được giữ nguyên. Vì vậy, các bộ nhớ cố định thường được sử dụng để lưu trữ các chương trình kiểm tra và khởi động một hệ vi xử lý khi bật nguồn hoặc các thông số của những lần chạy trước.

– **PROM** (Programable ROM – ROM có khả năng lập trình): Người thiết kế có khả năng nạp chương trình vào ROM nhưng chỉ được một lần duy nhất và không thể sửa đổi được.

2.4.3.2. Bộ nhớ bán cố định EPROM (Erasable Programable ROM)

Người sử dụng có thể nạp và xóa chương trình một số lần. Có thể nạp bằng xung điện và xóa bằng tia cực tím. Hiện tại EPROM đang được dùng khá phổ biến. Họ 27xxx có các loại sau: 2708 (1Kx8), 2716 (2Kx8), 2732 (4Kx8), 2764 (8Kx8), 27128 (16Kx8), 27256(32Kx8), 27512 (64x8) với thời gian thâm nhập $t_{ac}=250-450$ ns tùy theo loại cụ thể. Trên hình 2.11 là sơ đồ các tín hiệu và bảng chức năng của 2716.



Hình 2.11. Sơ đồ các tín hiệu và bảng chức năng của 2716

- A0-A10: Địa chỉ
- D0-D7: Dữ liệu
- OE: Cho phép đưa dữ liệu ra
- CE/PGM: Chọn vò/điều khiển ghi
- Vpp: Điện áp ghi.

Ngoài ra tùy theo các nạp xóa số liệu người ta còn chia bộ nhớ bán cố định ra những loại như EEPROM, EAPROM.

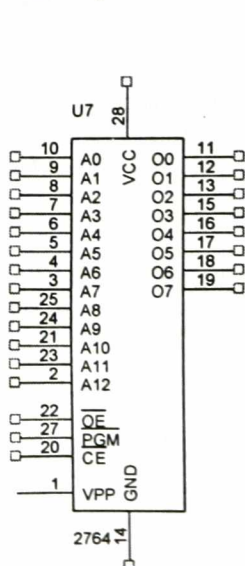
- EEPROM: Có thể nạp xoá số liệu bằng phương pháp điện.
- EAROM: Có thể nạp xoá chương trình một số lần, việc nạp xoá đơn giản bằng tín hiệu điện nhưng nhược điểm của loại này là thường đòi hỏi nhiều loại điện áp khác nhau.

2.4.3.3. RAM (Random Access Memory – bộ nhớ đọc/ghi)

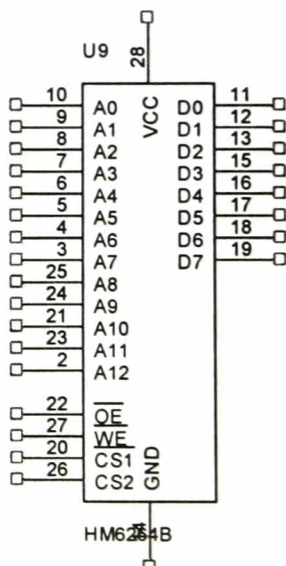
Bộ nhớ đọc/ghi còn gọi là bộ nhớ truy nhập ngẫu nhiên, người sử dụng có thể ghi số liệu vào và đọc số liệu ra. Khi mất điện thì số liệu cũng mất theo. RAM có hai loại.

– **RAM tĩnh** (Static RAM): Loại RAM này lấy cấu trúc Flip-Flop làm đơn vị nhớ cơ sở. Vì vậy, khi số liệu đã được ghi vào mà nguồn nuôi vẫn có thì số liệu trong RAM vẫn còn .

– **RAM động** (Dynamic RAM): RAM động lợi dụng điện dung ký sinh của cực cổng (Gate) transistor trường MOS để chứa dữ liệu. Vì vậy, khi số liệu đã được đưa vào mặc dù còn nguồn nuôi nhưng số liệu cứ mất dần do điện tích của tụ giảm theo thời gian. Để giữ lại số liệu ấy phải làm thêm một mạch gọi là mạch làm tươi và có tín hiệu làm tươi (Refresh).



Hình 2.12. EPROM 2764



Hình 2.13. SRAM HM6264B

2.4.4. MẠCH GIẢI MÃ ĐỊA CHỈ

Mỗi mạch nhớ nối ghép với CPU cần phải được CPU quy chiếu tới một cách chính xác khi thực hiện các thao tác đọc ghi. Điều đó có nghĩa là mỗi mạch nhớ phải được gán cho một vùng riêng biệt có địa chỉ xác định nằm trong không gian địa chỉ tổng thể của bộ nhớ. Việc gán địa chỉ cụ thể cho mạch nhớ được thực hiện nhờ một xung chọn và được lấy từ mạch giải mã địa chỉ. Khi CPU muốn thực hiện trao đổi thông tin với bộ nhớ hay thiết bị ngoại vi, nó đưa ra địa chỉ của thiết bị cần trao đổi trên Bus địa chỉ, sau đó qua một bộ giải mã địa chỉ sẽ xuất tín hiệu chọn chip CS (Chip Select) hoặc tín hiệu cho phép CE (Chip Enable) gửi tới bộ nhớ hay thiết bị vào/ra cần thiết. Bộ nhớ hay thiết bị vào/ra nào nhận được tín hiệu chọn sẽ được phép trao đổi dữ liệu với CPU còn các mạch giao tiếp khác sẽ bị cấm (đầu ra ở trạng thái Hi-Z).

Một trong các mạch giải mã được sử dụng rộng rãi nhất là 74LS138. Sơ đồ hình dáng ngoài và bảng chức năng của 74LS138 (hình 2.14).

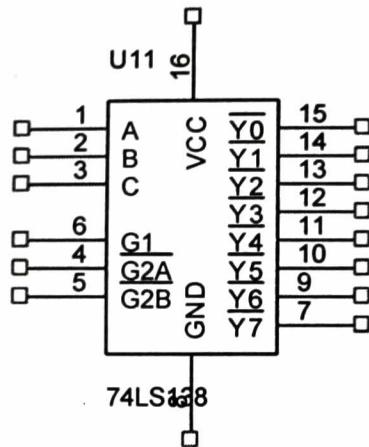
Ví dụ: Sử dụng mạch nhớ EPROM 2764 (8Kx8) có địa chỉ từ F0000H – FFFFFH. Ta sẽ dùng mạch giải mã địa chỉ 74LS138 thực hiện giải mã địa chỉ.

Từ bảng chức năng ta thấy $Y_0 = 0$ khi mà các đầu vào chọn có $C = 0, B = 0, A = 0$. Các đầu vào cho phép $G_2A = 0, G_2B = 0, G_1 = 1$. Vì vậy, sơ đồ ghép nối 74LS138 với EPROM 2764 như hình vẽ 2.14.

Các bit địa chỉ A16 – A19 được đặt bằng 1, trong đó A16, A17, A18 đưa qua mạch logic NAND và đưa vào G2A.

Tín hiệu IO/M đưa vào chân G2B

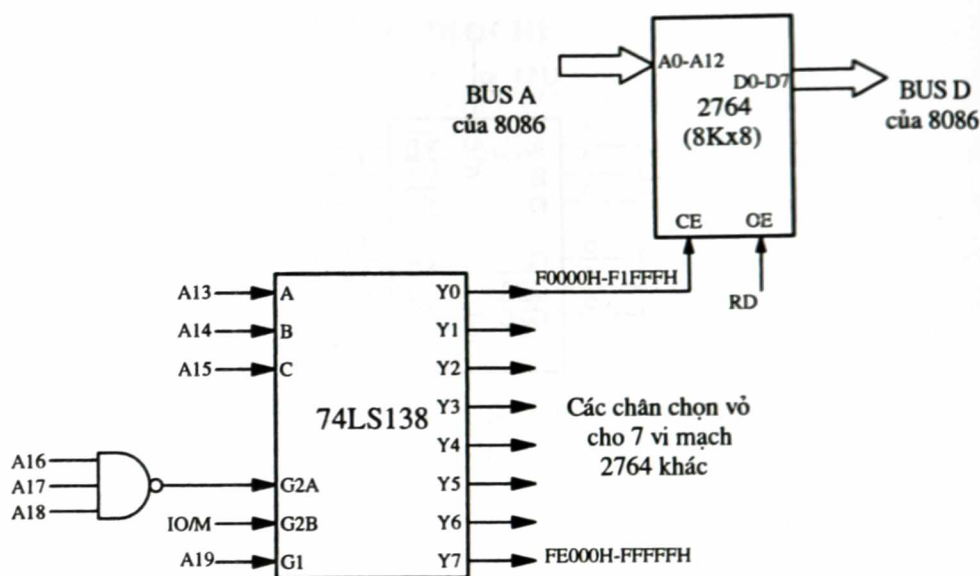
A13, A14, A15 được đặt bằng 0 và lần lượt đưa vào chân A, B, C.



Các đầu vào						Các đầu ra							
Chọn			Cho phép										
C	B	A	G2B	G2A	G1	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
x	x	x	1	x	x	1	1	1	1	1	1	1	1
x	x	x	x	1	x	1	1	1	1	1	1	1	1
x	x	x	x	x	0	1	1	1	1	1	1	1	1
0	0	0	0	0	1	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	1	0	0	0	1	1	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1	1	0	1	1	1	1
1	0	0	0	0	1	1	1	1	1	0	1	1	1
1	0	1	0	0	1	1	1	1	1	1	0	1	1
1	1	0	0	0	1	1	1	1	1	1	1	0	1
1	1	1	0	0	1	1	1	1	1	1	1	1	0

x: không quan tâm.

Hình 2.14. Hình dáng ngoài và bảng chức năng của 74LS138

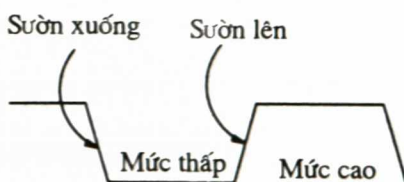


Hình 2.15. Mạch giải mã dùng 74LS138

2.5. BIỂU ĐỒ THỜI GIAN ĐỌC/GHI CỦA VI XỬ LÝ

2.5.1. XUNG NHỊP VÀ CHU KỲ MÁY

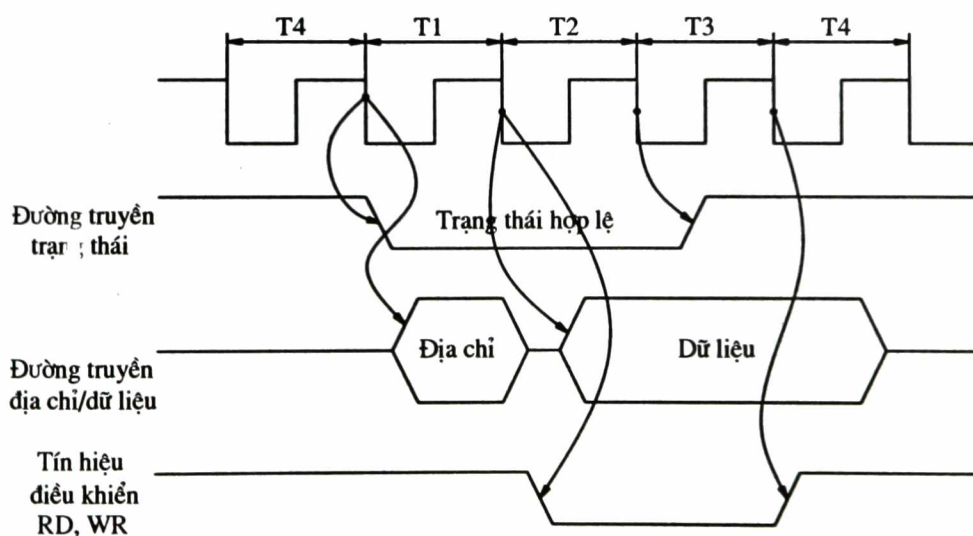
Trên hình 2.16 mô tả một chu kỳ xung nhịp (chu kỳ đồng hồ), một chu kỳ đồng hồ bao gồm 2 pha đối xứng gọi là pha 1 (tín hiệu đồng hồ ở mức thấp) và pha 2 (tín hiệu đồng hồ ở mức cao). Các chu kỳ đồng hồ này được đưa đến lối vào xung nhịp của vi xử lý. Một chu kỳ xung nhịp còn được gọi là một nhịp. Thời gian cần thiết và số xung nhịp cơ sở cho một thao tác của vi xử lý gọi là một chu kỳ máy. Mỗi một chu kỳ máy có 4 nhịp. Hình 2.17 là giản đồ thời gian thực hiện chu kỳ bus của vi xử lý 8086.



Hình 2.16. Xung nhịp

Nhìn vào chu kỳ bus ta có thể phân ra làm 4 pha như sau:

- Pha T1: Các đường trạng thái hoạt động để xác định kiểu thao tác nào được CPU thực hiện, đồng thời các địa chỉ bộ nhớ I/O cũng được truyền.
- Pha T2: Các tín hiệu địa chỉ được thay thế bằng các tín hiệu dữ liệu, các tín hiệu đọc/ghi trở nên tích cực.
- Pha T3: Trước tiên các tín hiệu trạng thái được truyền đi để báo hiệu sắp hết một chu kỳ Bus, sau đó tín hiệu điều khiển cũng được truyền đi.
- Pha T4: Nói chung đây là pha rảnh rỗi, trong pha này CPU và các khối bên ngoài có thời gian để vô hiệu hoá Bus dữ liệu.

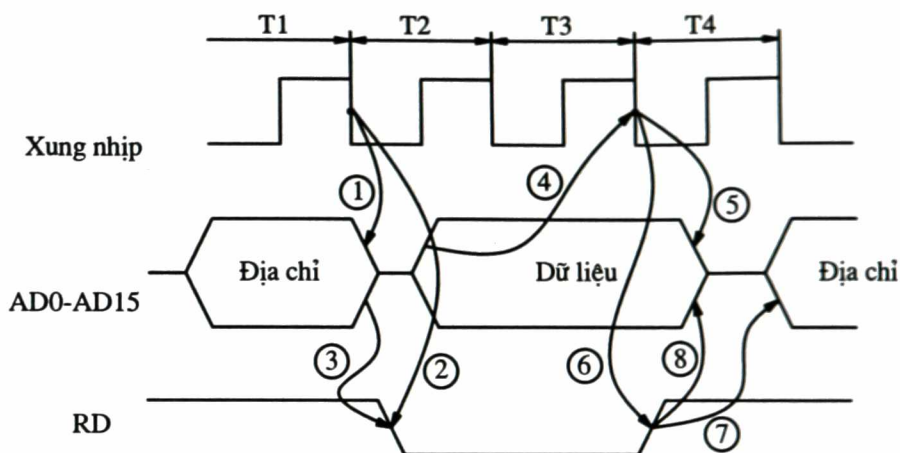


Hình 2.17. Thời gian thực hiện chu kỳ Bus của vi xử lý 8086

2.5.2. CHU KỲ ĐỌC/GHI CỦA VI XỬ LÝ 8086

Hình 2.18 chỉ ra một chu kỳ đọc của vi xử lý 8086, ngoài pha 1 được mô tả như ở trên ta cần chú ý đến các pha còn lại. Cũng cần chú ý rằng theo đặc điểm kỹ thuật thì dữ liệu phải tồn tại ít nhất 20 ns

trước khi kết thúc T3 và vẫn phải tồn tại ít nhất 10 ns sau khi kết thúc T4. Trong khi đó ta có thể huỷ bỏ đồng thời tín hiệu báo đọc (0 ns).



Hình 2.18. Chu kỳ đọc của vi xử lý

Trong đó các đường:

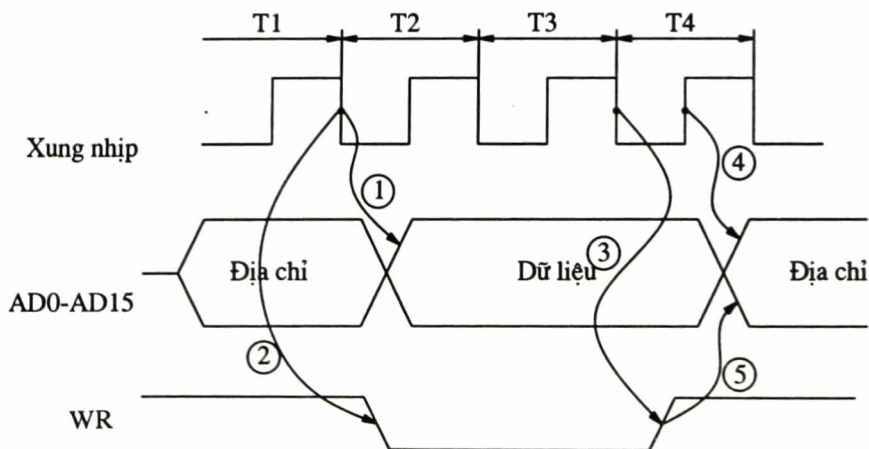
1. t_{CRAZ} : Đồng hồ ở mức thấp cho đến khi bus địa chỉ ở trạng thái Hi - Z = 35ns Max.
2. t_{CLRL} : Đồng hồ ở mức thấp cho đến khi RD hoạt động = 70ns Max.
3. t_{AZLN} : Bus địa chỉ được thả nổi cho đến khi RD hoạt động = 0ns Min.
4. t_{OVCL} : Dữ liệu hợp lệ cho đến khi đồng hồ ở mức thấp = 20ns Min.
5. t_{CLDX} : Đồng hồ ở mức thấp cho đến khi dữ liệu không hợp lệ = 10ns Min.
6. t_{CLRH} : Đồng hồ mức thấp cho đến khi RD ở mức cao = 10ns Min.
7. t_{RMAV} : RD ở mức cao cho đến khi các địa chỉ hợp lệ = 85ns Min.
8. t_{RHDX} : Đọc dữ liệu ở mức cao cho đến khi dữ liệu không hợp lệ = 0 Min.

Việc truy nhập bộ nhớ kéo dài từ T1 – T3 (gần 3 chu kỳ đồng hồ $3 \cdot T = 3 \cdot 200 = 600\text{ns}$). Trong tổng số thời gian này phải tính đến thời gian trễ khi truyền địa chỉ $t_{\text{trễ địa chỉ}} = 110\text{ns}$, thời gian giữ của dữ liệu khi đọc $t_{\text{giữ}} = 30\text{ns}$ và thời gian trễ do truyền tín hiệu qua các mạch đệm nhiều nhất là $t_{\text{trễ đệm}} = 40\text{ns}$. Như vậy, các bộ nhớ nối với 8086 – 5MHz cần phải có thời gian thâm nhập nhỏ hơn:

$$3 \cdot T - t_{\text{trễ địa chỉ}} - t_{\text{giữ}} - t_{\text{trễ đệm}} = 600 - 110 - 30 - 40 = 420\text{ns}$$

Hình 2.19 chỉ ra một chu kỳ ghi của vi xử lý 8086, ngoài pha T1 được mô tả như ở trên ta cần chú ý đến các pha sau:

- Pha T2: Trong pha này CPU xuất ra dữ liệu cần được ghi và tín hiệu báo ghi tới bộ nhớ hoặc I/O.
- Pha T3: Trong giai đoạn này dữ liệu ghi là ổn định và tín hiệu báo ghi đã được tạo ra.
- Pha T4: Tín hiệu báo ghi bị vô hiệu hoá và sau đó dữ liệu cần ghi cũng bị huỷ bỏ để dành chỗ cho các địa chỉ của pha T1 của chu kỳ tiếp theo.



Hình 2.19. Chu kỳ ghi của vi xử lý 8086

1. t_{CLDV} : Đồng hồ ở mức thấp cho đến khi dữ liệu hợp lệ = 44ns Max.
2. t_{CVTCV} : Đồng hồ ở mức thấp cho đến khi WR hoạt động = 70ns Max.
3. t_{CVCTX} : Đồng hồ ở mức thấp cho đến khi WR không hoạt động = 55ns Max.
4. t_{CHDX} : Đồng hồ ở mức cao cho đến khi dữ liệu không hợp lệ = 10ns Min.
5. WR không hoạt động cho đến khi dữ liệu không hoạt động = 10ns.

2.6. CÁC CHẾ ĐỘ LÀM VIỆC CỦA VI XỬ LÝ

2.6.1. CHẾ ĐỘ KHỞI ĐỘNG

Vi xử lý có hai chế độ khởi động: khởi động cứng và khởi động mềm.

– Khởi động cứng: Là đưa tín hiệu tác dụng vào chân Reset của mạch vi xử lý (thường ở mức thấp). Đối với bộ vi xử lý 8086, sau tín hiệu Reset chương trình được thực hiện từ ô nhớ F000H – FFF0H. Trong đa số trường hợp ô nhớ này thường chứa một lệnh nhảy tới một thủ tục thực hiện việc khởi tạo hệ thống.

– Khởi động mềm: Là quá trình khởi động lại (Restart) của mạch vi xử lý sau khi đã thực hiện các lệnh đặc biệt. Mỗi lệnh khởi động có một địa chỉ riêng giữ ở một ô nhớ nhất định trong ROM. Thông thường các hãng sản xuất tổ chức một vùng nhớ đặc biệt để làm nơi chứa các lệnh khởi động này gọi là vùng con chứa vector khởi động.

2.6.2. CHẾ ĐỘ ĐỢI

Mục đích của việc vi xử lý làm việc ở chế độ đợi là để phối hợp tốc độ làm việc của vi xử lý với thiết bị ngoại vi. Ví dụ: một CPU thực hiện lệnh mất 500ns, trong khi đó một thiết bị ngoại vi để thâm nhập

được số liệu mất $50\mu s$. Do vậy, để tìm được số liệu bên trong ngoại vi cần kéo dài xung đọc bằng cách thêm các chu kỳ đợi (TWAIT) để làm giảm tốc độ hoạt động của Bus trong khi vẫn giữ cho dữ liệu và các tín hiệu điều khiển hoạt động trong toàn bộ thời gian cần thiết cho các thiết bị tốc độ chậm nhận hoặc xuất tín hiệu.

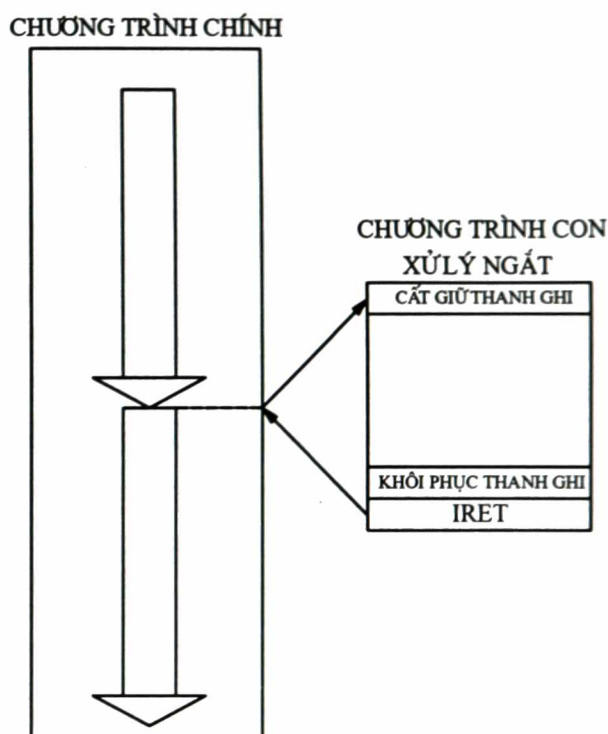
2.6.3. CHẾ ĐỘ TREO

Chế độ treo được sử dụng trong trường hợp cần nâng cao tốc độ truyền số liệu khi các bộ vi xử lý khác hoặc thiết bị ngoại vi muốn thâm nhập trực tiếp bộ nhớ mà không phải thông qua mạch vi xử lý. Muốn vậy, ngoại vi hoặc các bộ vi xử lý khác chỉ cần thông báo cho vi xử lý biết để xin phép sử dụng các đường Bus. Khi điều này có thể thực hiện được thì vi xử lý trả lời tín hiệu đồng ý và chuyển sang chế độ treo. Chế độ ngoại vi thâm nhập trực tiếp bộ nhớ được gọi là chế độ DMA (Direct Memory Access). Trong chế độ treo các đường dây được nối với chân ra của mạch vi xử lý, các đường địa chỉ và dữ liệu đều ở chế độ Hi-Z.

2.6.4. CHẾ ĐỘ NGẮT

Trong trường hợp vi xử lý làm việc với ngoại vi chậm mà phải xử lý với khối lượng tin tức lớn, thì hoạt động trong chế độ đợi sẽ mất quá nhiều thời gian và ảnh hưởng tới hoạt động chung của toàn hệ. Lúc đó, người ta cho vi xử lý làm việc ở chế độ ngắt, ở chế độ này ngoại vi sẽ gửi tín hiệu xin ngắt đến bộ vi xử lý, nếu là ngắt hợp lý thì vi xử lý sẽ thực hiện nốt lệnh hiện hành và chuyển sang chế độ xử lý ngắt. Vi xử lý gửi ra tín hiệu đồng ý ngắt, ngoại vi nhận được tín hiệu đó sẽ gửi vào CPU các địa chỉ tương ứng với chương trình con cần xử lý ngắt mà thực chất đây là quá trình nhận dữ liệu từ ngoại vi đã xử lý xong. Sau khi nhận được địa chỉ ngắt vi xử lý tạm thời dừng công việc chính, lưu giữ địa chỉ của chương trình hiện hành và nạp vào địa chỉ của ngắt. Sau khi thực hiện xong để ra khỏi chế độ của ngắt cần có

lệnh quay trở về IRET ở cuối chương trình ngắt. Khi nhận được tín hiệu trở về vi xử lý sẽ quay trở về chương trình chính với đúng chế độ trước khi ngắt.



Hình 2.20

2.6.5. CHẾ ĐỘ DỪNG

Sau khi thực hiện lệnh dừng (HLT) mạch vi xử lý sẽ thực hiện một chu kỳ máy nào đó (thông qua lệnh NOP). Sau lệnh (NOP), vi xử lý sẵn sàng làm việc nhưng phải có khoảng trống để nghỉ ngơi. Muốn thoát khỏi chế độ này cần đưa vào lệnh RESET thì vi xử lý sẽ bắt đầu làm việc trở lại.

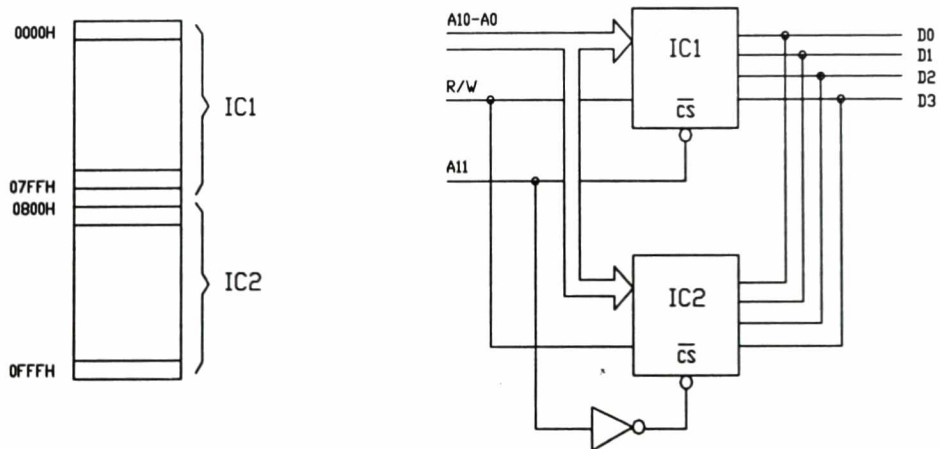
2.7. MỘT SỐ VÍ DỤ

Ví dụ 1: Thiết kế bộ nhớ có dung lượng 4K x 4bit dùng các IC nhớ 2K x 4bit.

Giải:

Các bộ nhớ 2K cần 11 bit địa chỉ A0 ÷ A10, trong khi đó bộ nhớ 4K lại cần 12 bit địa chỉ A0 ÷ A11. Vì vậy, để thực hiện bộ nhớ 4K x 4bit ta dùng hai IC này đấu song song các đầu dữ liệu và các đầu địa chỉ từ A10 ÷ A0. Bit địa chỉ thứ 12 là A11 được sử dụng để chọn các chip nhớ. Khi A11 = 0 chân CS\ của IC1 có mức thấp, còn chân CS\ của IC2 có mức cao vì tín hiệu A11 được đưa qua bộ đảo. Khi đó IC1 được chọn còn IC2 sẽ bị cấm. Khi A11 = 1 thì ngược lại, IC1 bị cấm và IC2 được chọn.

Sơ đồ thực hiện như sau:



Hình 2.21. Bộ nhớ 4K x 4bit từ các IC nhớ 2K x 4 bit

Ví dụ 2: Thiết kế bộ nhớ ROM 8K x 8 bit từ các IC ROM 1K x 8bit.

Giải:

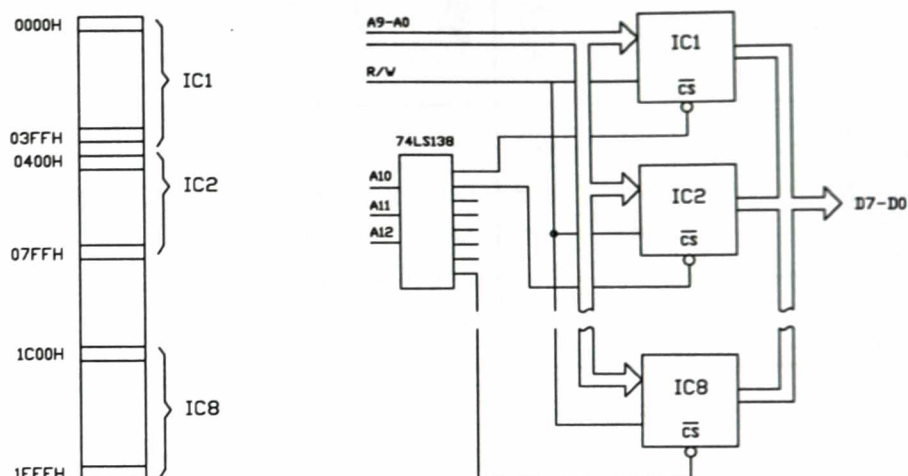
Các IC ROM 1K = 2¹⁰ có 10 bit địa chỉ A0 ÷ A9, còn bộ nhớ 8K = 2³.2¹⁰ = 2¹³, cần 13 bit địa chỉ. Như vậy, còn thiếu tổ hợp các bit A10, A11, A12. Khi đó ta dùng bộ giải mã địa chỉ 74LS138 (3 đầu vào, tám đầu ra). Địa chỉ của các IC cụ thể như bảng 2.6:

Bảng 2.6. Bảng phân vùng địa chỉ cho các IC nhớ

[illegible]

IC5	1	0	0	0	0	0	0	0	0	0	0	0	0	1000H
	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	1	0	0	1	1	1	1	1	1	1	1	1	1	13FFH
IC6	1	0	1	0	0	0	0	0	0	0	0	0	0	1400H
	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	1	0	1	1	1	1	1	1	1	1	1	1	1	17FFH
IC7	1	1	0	0	0	0	0	0	0	0	0	0	0	1800H
	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	1	1	0	1	1	1	1	1	1	1	1	1	1	1BFFH
IC8	1	1	1	0	0	0	0	0	0	0	0	0	0	1C00H
	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH

Và thực hiện bộ nhớ như 2.22:



Hình 2.22. Bộ nhớ 8K x 8bit với các IC nhớ 1K x 8bit

2.8. LẬP TRÌNH HỢP NGỮ CHO VI XỬ LÝ 8086/8088

2.8.1. GIỚI THIỆU CHUNG VỀ HỢP NGỮ

Thời kỳ đầu của máy tính, các lập trình viên phải viết chương trình dưới dạng ngôn ngữ máy. Mặc dù số Hexa đã biểu diễn khá hiệu quả số nhị phân, song làm việc trên mã máy vẫn còn là công việc nặng nhọc đối với con người. Cuối cùng hợp ngữ đã được xây dựng, trong đó có sử dụng các từ gọi nhớ và cùng với những đặc tính khác nữa, hợp ngữ đã giúp cho công việc lập trình trở nên dễ dàng và ít lỗi hơn.

Các chương trình hợp ngữ cần được dịch ra dưới dạng mã máy nhờ một chương trình được gọi là chương trình dịch hợp ngữ hay trình hợp dịch. Hợp ngữ được coi là một ngôn ngữ bậc thấp vì nó có quan hệ trực tiếp với cấu trúc bên trong CPU. Để lập trình hợp ngữ, lập trình viên phải nắm vững cấu trúc bên trong CPU và tập lệnh của vi xử lý.

Một trình hợp dịch dùng để dịch chương trình hợp ngữ ra dạng mã máy (đôi khi còn được gọi là mã đối tượng (Object Code) hay mã 108

lệnh (Opcode)). Với vi xử lý 8086 ta có thể sử dụng chương trình dịch hợp ngữ PWB (Microsoft Programmer's WorkBench Version 1.1). Chương trình PWB cho phép soạn thảo, dịch, liên kết và chạy chương trình đã dịch thành công. Ngoài ra, còn được sử dụng cho việc tìm lỗi, chạy chương trình từng bước, hiển thị nội dung của từng thanh ghi và trạng thái các cờ.

2.8.2. CÚ PHÁP CỦA CHƯƠNG TRÌNH HỢP NGỮ

2.8.2.1. Các trường của hợp ngữ

Một dòng lệnh của một chương trình hợp ngữ có thể có những trường sau (không nhất thiết phải có đủ các trường).

Tên (nhãn): Mã lệnh Các toán hạng ;Chú thích

– **Nhãn:** Có thể là nơi đến của một lệnh nhảy, hay một lời gọi đến một chương trình con. Các nhãn này sẽ được chương trình dịch gán bằng các địa chỉ cụ thể của ô nhớ. Nhãn có thể dài đến 31 ký tự nhưng thường ngắn hơn nhiều, các ký tự có thể là chữ, số, ký tự đặc biệt nhưng lưu ý rằng ký tự đầu tiên không được là số. Một nhãn thường kết thúc bằng dấu “:”.

– **Mã lệnh:** Trong trường mã lệnh nói chung sẽ có các lệnh thật và lệnh giả. Đối với các lệnh thật thì trường này sẽ chứa các mã lệnh gọi nhớ. Mã lệnh này sẽ được chương trình dịch dịch ra dạng mã máy. Đối với các hướng dẫn chương trình dịch thì trường này chứa các thông tin khác nhau liên quan đến các lệnh giả của hướng dẫn.

– **Toán hạng:** Đối với lệnh thật thì trường này chứa các toán hạng của lệnh. Tùy theo từng loại lệnh mà ta có thể có 0, 1 hoặc 2 toán hạng trong một lệnh. Đối với các hướng dẫn chương trình dịch thì trường này chứa các thông tin khác nhau liên quan đến các lệnh giả của hướng dẫn.

– **Chú thích:** Ghi vào các lời giải thích cho chương trình và được viết sau dấu “;”. Trình hợp dịch sẽ bỏ qua tất cả các chú thích này khi dịch chương trình.

Ví dụ 1: Một dòng lệnh gọi nhớ:

```
BACK:    MOV  AH, [BX][SI]    ; nạp vào AH nội dung ở
                                nhớ có địa chỉ DS:(BX+SI)
```

Trường nhãn có tên là BACK, trường mã lệnh có lệnh MOV, tại trường toán hạng có thanh ghi AH, BX, SI và phần chú thích là dòng chữ “nạp vào AH nội dung ở nhớ có địa chỉ DS: (BX+SI)”.

Ví dụ 2: Dòng lệnh với các hướng dẫn cho chương trình dịch.

```
MAIN      PROC
MAIN      ENDP
```

Trường tên có tên thủ tục là MAIN, ở trường mã lệnh ta có các lệnh giả PROC và ENDP, đây là các lệnh giả dùng để bắt đầu và kết thúc thủ tục MAIN.

Tiếp theo ta sẽ nghiên cứu cách thức khai báo biến, khai báo hằng và khai báo chương trình con.

2.8.2.2. Khai báo biến, hằng, chương trình con

• Khai báo biến

Biến byte

Biến byte chiếm 1 byte trong bộ nhớ và việc định nghĩa được thực hiện như sau:

Tên	DB	Giá trị khởi đầu
-----	----	------------------

Ví dụ:

BIEN1	DB	5
BIEN2	DB	?
BIEN3	DB	'\$'

Trong các ví dụ trên BIEN1 có giá trị khởi đầu là 5, BIEN2 không có giá trị khởi đầu, BIEN3 là một biến ký tự.

Biến Word

Biến byte chiếm 2 byte trong bộ nhớ và việc định nghĩa được thực hiện như sau:

Tên	DW	Giá trị khởi đầu
-----	----	------------------

Ví dụ:

BIEN1	DW	50
BIEN2	DW	?

Trong các ví dụ trên BIEN1 có giá trị khởi đầu là 50, BIEN2 không có giá trị khởi đầu.

Biến mảng

Biến mảng là biến có một dãy liên tiếp các phần tử cùng loại byte hay từ. Khi định nghĩa biến mảng ta gán tên cho một dãy liên tiếp các byte hay từ trong bộ nhớ cùng với các giá trị ban đầu tương ứng.

Ví dụ 1:

M1	DB	2, 3, 4, 5, 6
----	----	---------------

Ví dụ trên định nghĩa biến mảng có tên là M1 gồm 6 byte. Phần tử đầu là 2 có địa chỉ trùng với địa chỉ của M1, phần tử thứ hai là 3 có địa chỉ là M1+1...

Khi ta muốn khởi đầu các phần tử của mảng với cùng một giá trị chúng ta có thể thêm toán tử DUP trong lệnh

Ví dụ 2:

M2	DB	100 DUP(0)
M3	DB	100 DUP(?)

Ví dụ trên định nghĩa biến mảng có tên là M2 gồm 100 byte, dành chỗ trong bộ nhớ cho nó để chứa 100 giá trị khởi đầu bằng 0. M3

gồm 100 byte, dành chỗ trong bộ nhớ cho nó để chứa 100 giá trị nhưng chưa được khởi đầu.

Biến kiểu xâu ký tự

Là một trường hợp đặc biệt của biến mảng, trong đó các phần tử của mảng là các ký tự. Một xâu ký tự có thể được định nghĩa bằng các ký tự hoặc bằng mã ASCII của ký tự đó.

Ví dụ: Các lệnh sau đều định nghĩa cùng một xâu ký tự.

```
STR1 DB      'STRING'
```

```
STR2 DB      73H, 74H, 72H, 69H, 6EH, 67H
```

```
STR3 DB      73H, 74H, 'R', 'I', 6EH, 67H
```

• Khai báo hằng

Các hằng số trong chương trình hợp ngữ thường được định nghĩa và gán bằng một tên để nhớ để làm cho chương trình dễ đọc hơn, thuận tiện cho việc kiểm tra và hiệu chỉnh. Việc chỉ dẫn được thực hiện bằng toán tử EQU

Ví dụ 1:

```
HANG1      EQU      100
```

```
HANG2      EQU      'XAU KY TU'
```

Với hằng là một xâu ký tự như HANG2 thì có thể sử dụng để định nghĩa cho một biến mảng khác.

Ví dụ 2:

```
BIENMANG   DB      HANG2, ' '
```

• Khai báo chương trình con

Khi một chương trình cần thiết phải thực hiện một nhóm lệnh nhất định (chúng có thể bao gồm các tham số khác nhau) nhiều lần, chúng ta nên nhóm các lệnh đó thành một chương trình con. Sau đó, nó được gọi và thực hiện bởi lệnh CALL, kết thúc chương trình con

phải có lệnh RET. Sau lệnh RET chúng sẽ trao quyền điều khiển cho chương trình chính tiếp theo tại vị trí mà nó đã đạt tới trước khi thực hiện lệnh CALL.

Tên_ctc

; các lệnh của thân chương trình con

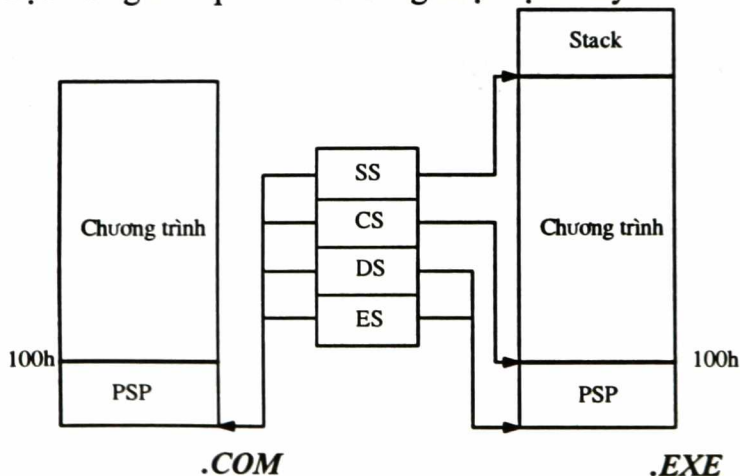
RET

2.8.3. KHUÔN DẠNG CỦA CHƯƠNG TRÌNH HỢP NGỮ

Như chúng ta đã biết có hai loại tập tin thực hiện được là .EXE và .COM.

– Chương trình .EXE là một thủ tục xa (Far Procedure), có thể được gọi từ một đoạn khác. Với chương trình .EXE có thể được định nghĩa 4 đoạn khác nhau: Đoạn lệnh (code), đoạn dữ liệu (data), đoạn ngăn xếp (stack) và đoạn thêm (extra).

– Khác với chương trình .EXE, chương trình .COM là một thủ tục gần (Near Procedure), chỉ gọi được từ bên trong đoạn lệnh và chỉ có thể được định nghĩa với duy nhất một đoạn lệnh, do đó phải đặt đoạn dữ liệu và ngăn xếp vào bên trong đoạn lệnh này.



Hình 2.23

2.8.3.1. Chương trình .EXE

Khi một chương trình .EXE được nạp vào bộ nhớ, DOS sẽ tạo ra một vùng gồm 256 byte gọi là đoạn mở đầu của chương trình (PSP), dùng để chứa các thông tin liên quan đến việc thực hiện chương trình. Vùng PSP nằm ngay trước phần mã lệnh. Các thông số liên quan đến chương trình cũng được đưa vào DS, ES. Vì vậy, DS và ES không chứa địa chỉ đoạn dữ liệu, do đó phải đặt thanh ghi DS để nó chứa địa chỉ đoạn dữ liệu bằng các lệnh sau đây:

```
MOV      AX, data
```

```
MOV      DS, AX
```

Khung cho một chương trình .EXE có dạng sau:

```
TITLE      Tên_chương_trình
COMMENT    *
           Bắt đầu của một khối chú thích
           *

;Khai báo đoạn stack
stack      SEGMENT STACK
           ;Khai báo kích thước Stack
stack      ENDS

;Khai báo đoạn dữ liệu
data SEGMENT
           ;Định nghĩa các dữ liệu
data ENDS

;Khai báo đoạn mã
code SEMENT
           ASSUME  CS:code, DS:data
```

Start:

 ;Các lệnh

code ENDS

END Start

Nhãn Start là nơi chứa địa chỉ đầu tiên mà CPU sẽ thực hiện, chỉ thị END sẽ kết thúc chương trình.

Ví dụ: Viết chương trình .EXE in ra màn hình dòng chữ “DHKTCN!”.

Trước hết ta sẽ đưa ra một số hàm ngắt của INT 21H và INT 20H

Hàm 1 của ngắt INT 21H: Đọc một ký tự từ bàn phím.

Vào: AH = 1

Ra: AL = mã ASCII củ ký tự

AL = 0 khi ký tự gõ vào là các phím chức năng

Hàm 2 của ngắt INT 21H: Hiện một ký tự lên màn hình.

Vào: AH = 2

DL = mã ASCII của ký tự cần hiển thị

Hàm 9 của ngắt INT 21H: Hiện chuỗi ký tự với \$ ở cuối lên màn hình.

Vào: AH = 9

DX = địa chỉ lệnh của chuỗi ký tự

Hàm 4CH, ngắt INT 21H Kết thúc chương trình loại .EXE hoặc .COM

Vào: AH = 4CH

Ngắt INT 20H dành riêng để kết thúc chương trình dạng .COM

Chương trình được viết như sau:

```

TITLE          VIDU_EXE
COMMENT        *
in ra man hinh dong chu DHKTCN
Chuong trinh thuoc dang .EXE
                *

stack          SEGMENT  STACK
                DB      20 DUP('STACK')
stack          ENDS

data SEGMENT
STRING         DB      'DHKTCN!$'
data ENDS

code SEGMENT
                ASSUME  CS:code, DS:data

Start:
;Dat thanh ghi DS
                MOV     AX, data
                MOV     DS, AX

;In ra man hinh
                MOV     AH, 9
                LEA     DX, STRING          ;chuyen dia chi
                                                ;lech cua xau vao DX

                INT     21h
                MOV     AH, 4Ch
                INT     21h
code ENDS

```

```
END Start
```

Trong chương trình trên phần chỉ thị:

```
stack          SEGMENT STACK
                DB          20 DUP ( 'STACK' )
stack          ENDS
```

Là phần khai báo đoạn ngăn xếp có tên là STACK. Ngăn xếp có kích thước là $20 \times 5 = 100$ byte và lúc khởi đầu được lấp đầy bằng từ STACK.

```
data SEGMENT
STRING          DB          'DHKTCN!$'
data ENDS
```

Là phần khai báo đoạn dữ liệu có tên là data, đang chứa xâu STRING là dòng chữ “DHKTCN!” cần in ra màn hình. Xâu này kết thúc bởi ký tự \$ để báo cho DOS biết là đã đến cuối xâu.

Đoạn chương trình được bao quanh bởi các chỉ thị sau:

```
code SEGMENT
        ASSUME  CS:code, DS:data
        ...
code ENDS
```

Trong đó chỉ dẫn ASSUME báo cho Assembler biết rằng thanh ghi CS trỏ đến đoạn lệnh, thanh ghi DS trỏ đến đoạn dữ liệu.

Trong chương trình trên có sử dụng ngắt 21h (xem phần phụ lục A trang 320 – Kỹ thuật vi xử lý của Văn Thế Minh). Trong đó hàm 9 của INT 21h được dùng để in xâu có địa chỉ đặt trong thanh ghi DS:DX ra màn hình, hàm 4Ch để kết thúc chương trình và trở về DOS.

2.8.3.2. Chương trình .COM

Chương trình .COM chỉ có một đoạn duy nhất. Khi chương trình .COM bắt đầu chạy thì cả 4 thanh ghi đoạn đều trở về đầu của vùng PSP. Các chỉ thị cho CPU bắt đầu tạo Offset 100h ngay sau vùng PSP. Vì vậy, trong chương trình .COM, chỉ dẫn ASSUME phải báo cho CPU biết các thanh ghi CS, DS đều trở về đầu đoạn lệnh, không cần tạo ra một ngăn xếp riêng biệt và không cần đặt lại một thanh ghi đoạn nào. Với chương trình .COM chỉ cần cho thanh ghi IP (con trỏ lệnh) chỉ đến địa chỉ đầu tiên trong đoạn lệnh bằng lệnh chỉ dẫn ORG 100h ngay đầu đoạn lệnh.

Khung cho một chương trình .COM có dạng sau:

```
TITLE          Tên_chương_trình
COMMENT        *
               Bắt đầu của một khối chú thích
               *

;Khai báo đoạn mã
code SEGMENT
    ASSUME     CS:code, DS:code
    ORG        100h

Start:
    ;Các lệnh

code ENDS
END Start
```

Trở lại ví dụ trên. Bây giờ chúng ta sẽ viết chương trình dạng .COM

```
TITLE          VIDU_COM
COMMENT        *
```



```

in ra màn hình dòng chữ DHKTCN
Chương trình thuộc dạng .COM

*

code SEGMENT
    ASSUME    CS:code, DS:code
    ORG      100h
Start:      ;In ra màn hình
    MOV      AH, 9          ;hàm của INT21h
    LEA      DX, STRING     ;lay địa chỉ offset
                                ;của xâu
    INT      21h
    MOV      AH, 4Ch
    INT      21h
STRING      DB      'DHKTCN!$'
code ENDS
    END      Start

```

2.8.4. CẤU TRÚC LẬP TRÌNH CƠ BẢN BẰNG HỢP NGỮ

2.8.4.1. Cấu trúc tuần tự

Kiểu cấu trúc này các lệnh được viết tuần tự, lệnh nọ tiếp lệnh kia. Đây cũng là cấu trúc thông dụng và đơn giản nhất.

Ví dụ: Các thanh ghi BX và CX chứa các giá trị của biến b và c. Hãy tính giá trị của biểu thức $a = 2(b + c)$ và lưu kết quả vào AX.

```

XOR  AX, AX    ;Xóa thanh ghi AX
ADD  AX, BX    ;cong thanh ghi AX với b
ADD  AX, CX    ;cong tiếp với c

```

SAL AX, 1 ;nhan doi ket qua $a=2(b+c) \rightarrow AX$

2.8.4.2. Cấu trúc rẽ nhánh

Câu lệnh rẽ nhánh giúp quyết định nhánh nào của chương trình được thực hiện căn cứ vào điều kiện được đặt ra.

Cấu trúc rẽ nhánh một khối lệnh (hình 2.24)

Nếu điều kiện được thoả mãn thì khối lệnh được thực hiện, nếu không khối lệnh sẽ được bỏ qua và thực hiện lệnh tiếp theo sau khối này (hình 2.24). Điều này tương đương với việc dùng lệnh nhảy có điều kiện để bỏ qua một số thao tác nào đó trong chương trình hợp ngữ.

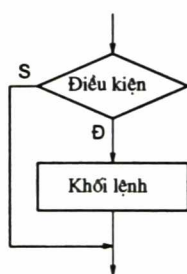
Ví dụ: Gán cho AX giá trị tuyệt đối của BX

CMP BX, 0 ;so sanh BX voi 0

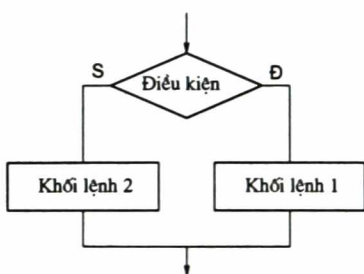
JNL GAN ;nhay den GAN neu $BX \geq 0$

NEG BX ;neu $BX < 0$ thi dao dau cua BX roi

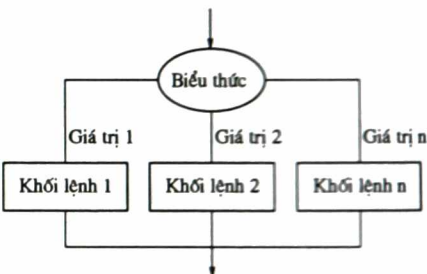
GAN:MOV AX, BX ;GAN



Hình 2.24
Cấu trúc rẽ nhánh
một khối lệnh



Hình 2.25
Cấu trúc rẽ nhánh
hai khối lệnh



Hình 2.26
Cấu trúc rẽ nhánh
nhiều khối lệnh

Cấu trúc rẽ nhánh hai khối lệnh (hình 2.25)

Nếu thoả mãn điều kiện 1 thì thực hiện khối lệnh 1, nếu không thì thực hiện khối lệnh 2 (hình 2.25). Điều này tương đương với việc

dùng lệnh nhảy có điều kiện và không điều kiện để nhảy đến các nhánh nào đó trong chương trình hợp ngữ.

Ví dụ 1: Giả sử có hai số 8 bit nằm trong hai ô nhớ 1234h và 1235h. Viết chương trình cho vi xử lý 8086 tìm số lớn hơn và gửi số lớn đến ô nhớ 1236h.

```
MOV     AL, [1234H]    ;AL←[1234H]
MOV     BL, [1235H]    ;BL←[1235H]
CMP     AL, BL         ;so sanh AL va BL
JG      MAX1           ;den MAX1 neu AL>BL
JL MAX2                ;den MAX2 neu AL<BL
MAX1: MOV [1236H], AL   ;[1236H]←AL
        JMP     EXIT    ;thoat
MAX2: MOV [1236H], BL   ;[1236H]←BL
        JMP     EXIT    ;thoat
EXIT:
```

Ví dụ 2: Gán cho BL bit dấu của AX

```
OR      AX, AX         ;or AX va AX
JS      GAN1           ;nhay den GAN1 neu AX<0
MOV     BL, 0          ;neu AX>0 thi BL←0
JMP     THOAT          ;roi thoat
GAN1: MOV     BL, 1     ;gan BL←1
THOAT:                ;thoat
```

Cấu trúc rẽ nhánh nhiều khối lệnh (hình 2.26)

Nếu biểu thức có giá trị bằng 1 thì công việc 1 được thực hiện, nếu biểu thức có giá trị 2 thì công việc 2 được thực hiện, nếu biểu thức có giá trị n thì công việc n được thực hiện.

Ví dụ: Các thanh ghi AX và BX đang lần lượt chứa hai số a và b.
Hãy so sánh hai số đó và ghi kết quả vào CX như sau:

Nếu $a > b$ thì $CX = 2$

Nếu $a = b$ thì $CX = 0$

Nếu $a < b$ thì $CX = 1$

Chương trình như sau:

```
CMP      AX, BX    ;so sanh AX va BX
JG       GAN2      ;den GAN2 neu AX>BX
JE       GAN0      ;den GAN0 neu AX=BX
JL       GAN1      ;den GAN1 neu AX<BX
GAN0:    MOV       CX, 0    ;CX←0
        JMP       THOAT    ;roi thoat
GAN1 :   MOV       CX, 1    ;CX←1
        JMP       THOAT    ;roi thoat
GAN2:    MOV       CX, 2    ;CX←2
        JMP       THOAT    ;roi thoat
THOAT:
```

2.8.4.3. Cấu trúc lặp

Cấu trúc lặp là cấu trúc mà trong đó một hoặc nhiều lệnh được thực hiện lặp đi lặp lại.

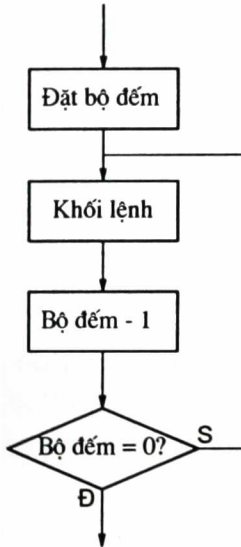
Lặp với số lần biết trước (hình 2.27)

Lặp lại các công việc trong khối lệnh với một số lần biết trước. Khi đó ta dùng lệnh LOOP và gán số lần lặp trong thanh ghi đếm CX.

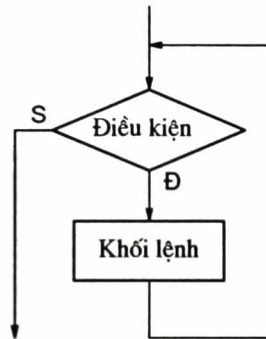
Ví dụ: Thực hiện cộng 5 lần liên tiếp số 10.

Giải:

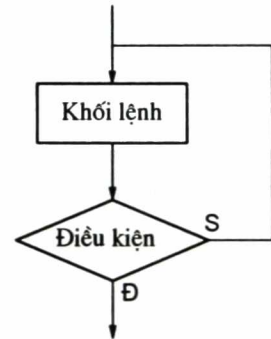
```
MOV     AX, 10
MOV     CX, 5
LAP:    ADD     AX, 10
        LOOP   LAP
        NOP
```



Hình 2.27



Hình 2.28



Hình 2.29

Lặp với số lần không biết trước

Lặp với số lần không biết trước được chia làm 2 dạng: Lặp kiểm tra điều kiện trước và lặp kiểm tra điều kiện sau.

Lặp kiểm tra điều kiện trước (hình 2.28).

Trước hết kiểm tra điều kiện, nếu đúng sẽ thực hiện khối lệnh, sau đó lại quay trở lại kiểm tra điều kiện. Nếu điều kiện sai khối lệnh sẽ không được thực hiện.

Ví dụ: Giả sử thanh ghi AX chứa một số a 16 bit. Viết chương trình kiểm tra nếu a lớn hơn 10 thì giảm a đi 1, và lặp lại cho đến khi a = 10. Ghi số lần lặp vào thanh ghi CX.

Giải:

```
XOR  CX, CX           ;xoa thanh ghi CX
XOR  AX, AX           ;xoa thanh ghi AX
MOV  AX, a             ;chuyen a vao AX
LAP: CMP      AX, 10    ;so sanh a voi 10
      JE      THOAT     ;neu a=10 thi thoat
      INC     CX        ;neu khac thi tang CX
      SUB     AX, 1      ;a=a-1
      JMP     LAP       ;quay tro lai LAP
THOAT:                ;thoat
```

Lặp kiểm tra điều kiện sau (hình 2.29)

Các công việc trong khối lệnh sẽ được thực hiện trước, sau đó mới kiểm tra điều kiện. Công việc sẽ được thực hiện lặp đi lặp lại cho đến khi điều kiện được thỏa mãn. Như vậy công việc sẽ được thực hiện ít nhất một lần.

Ví dụ 1: Giả sử địa chỉ hexa của cổng vào là 1180H và cổng ra là 1280H. Hãy lập chương trình để vi xử lý 8086 thực hiện công việc sau: Trước hết đặt cổng ra bằng FFh, giữ trạng thái này cho đến khi cổng vào bằng 0 thì đặt cổng ra bằng 0 rồi thoát khỏi chương trình.

Giải:

```
MOV  AL, FFh          ;AL←FFh
MOV  DX, 1280h         ;dat dia chi cong ra
OUT  DX, AL           ;dua so lieu trong AL ra cong
LAP: MOV      DX, 1180h; dat dia chi cong vao
IN   AL, DX          ;doc dia chi cong vao
CMP  AL, 0            ;AL=0?
```

```

JE    GAN                ;dung thi nay den GAN
JMP   LAP                ;sai thi doc lai
GAN:MOV AL, 0            ;AL←0
MOV   DX, 1280h          ;dat dia chi cong ra
OUT   DX, AL             ;dua so lieu trong AL ra cong
NOP

```

• Phần bài tập

Bài 1. So sánh sự giống và khác nhau cơ bản giữa VXL 8086 và 8088.

Tại sao nói chip 8088 ra đời sau VXL 8086 là một bước lùi về kỹ thuật nhưng đem lại nhiều lợi ích kinh tế?

Bài 2. Khi VXL 8086 thực hiện ghi dữ liệu ra một thiết bị ngoại vi thì các tín hiệu điều khiển $\overline{RD}$, $\overline{WR}$, $IO/\overline{M}$, ALE, $\overline{DEN}$ có mức logic gì? Khi đọc dữ liệu từ bộ nhớ thì các tín hiệu đó có mức logic gì?

Bài 3. Bus là gì? Anh (chị) hãy trình bày ý nghĩa của các loại Bus trong hệ thống Bus? Cho ví dụ.

Anh (chị) hãy phân tích để thấy được ý nghĩa của việc sử dụng cơ chế xử lý xen kẽ liên tục dòng mã lệnh trong 8088/8086?

Bài 4. Chế độ địa chỉ là gì? VXL 8088/8086 có mấy chế độ địa chỉ? Kể tên và lấy ví dụ cho mỗi chế độ địa chỉ. Sự phong phú của các chế độ địa chỉ có ý nghĩa gì? Hãy minh họa bằng ví dụ.

Bài 5. Mạch giải mã địa chỉ là gì? có mấy loại mạch giải mã địa chỉ. Ưu, nhược điểm của từng loại.

Sử dụng mạch giải mã 74LS138 thực hiện ghép nối VXL 8088 với bộ nhớ 32Kx8 từ các IC2764 (8Kx8). Biết địa chỉ của ô nhớ đầu tiên của vùng nhớ là 02000h.

+ Hãy cho biết IC2764 thuộc loại IC nào? Cho biết số chân địa chỉ và dữ liệu của IC2764?

+ Vẽ sơ đồ ghép nối và phân vùng địa chỉ cho các IC.

Bài 6.

+ Thiết kế bộ nhớ có dung lượng $2K \times 8\text{bit}$ dùng các IC nhớ $2K \times 4\text{bit}$.

+ Thiết kế bộ nhớ ROM $8K \times 8\text{ bit}$ từ các IC ROM $1K \times 8\text{bit}$.

+ Thiết kế bộ nhớ $16K \times 8\text{bit}$ dùng các IC nhớ $16K \times 1\text{ bit}$.

+ Thiết kế bộ nhớ có dung lượng $4K \times 8\text{bit}$ dùng các IC nhớ $2K \times 4\text{bit}$.

+ Thiết kế bộ nhớ có dung lượng $4K \times 4\text{bit}$ dùng các IC nhớ $2K \times 4\text{bit}$.

Bài 7. Giả sử địa chỉ hexa của cổng vào là 1180H và cổng ra là 1280H. Hãy lập chương trình để vi xử lý 8086 thực hiện công việc sau: Trước hết đặt cổng ra bằng FFh, giữ trạng thái này cho đến khi cổng vào bằng 0 thì đặt cổng ra bằng 0 rồi thoát khỏi chương trình.

Bài 8. Cho một hệ thống vi xử lý điều khiển lò điện trở nhiệt trong công nghiệp như hình 2.30:

+ Cổng vào có địa chỉ 301H (Dữ liệu 8 bit)

+ Cổng ra có địa chỉ 303H (Dữ liệu 8 bit)

+ Giả sử cảm biến nhiệt đo nhiệt độ từ 0 đến 255°C . Đầu ra của cảm biến 0 đến 5V

+ Chuyển đổi A/D từ 00H đến FFH (tương ứng 0 đến 5V)

+ $U_2 > U_1$

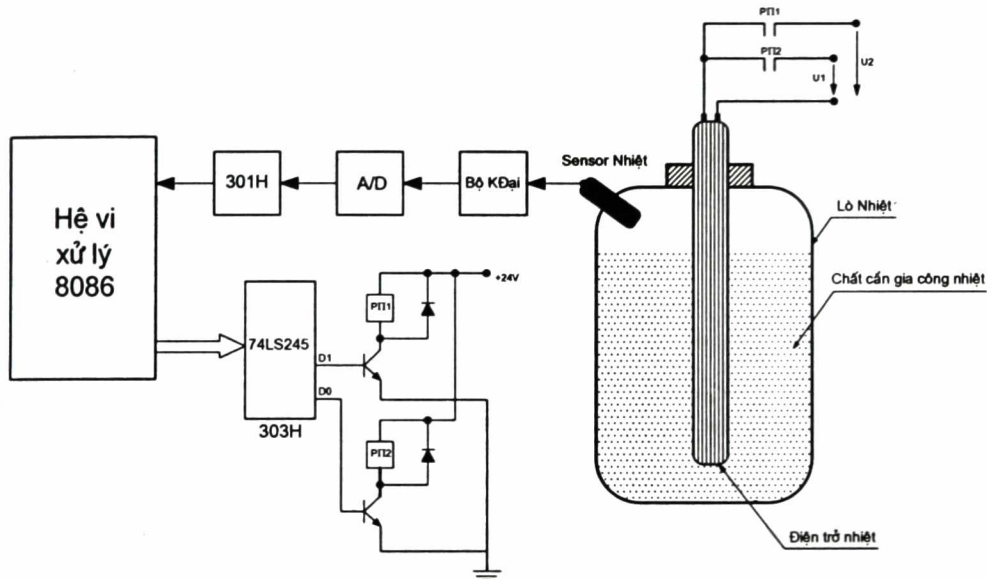
Hãy viết chương trình để 8086 điều khiển hệ thống lò nhiệt như sau:

+ Khi $t^0 < 100^{\circ}\text{C}$ thì cấp nguồn 1

+ Khi $100^{\circ}\text{C} \leq t^0 \leq 200^{\circ}\text{C}$ thì cấp nguồn 2

+ Khi $t^0 > 200^0\text{C}$ thì ngừng cấp điện để ổn định nhiệt quanh 200^0C

+ Khi $t^0 \geq 210^0\text{C}$ thì ngừng toàn bộ hệ thống.



Hình 2.30

Bài 9. Sử dụng mạch giải mã 74LS138 thực hiện ghép nối VXL 8086 với bộ nhớ 32Kx8 từ các IC2764 (8Kx8). Biết địa chỉ của ô nhớ đầu tiên của vùng nhớ là D000h.

a. Hãy cho biết IC2764 thuộc loại IC nào? Cho biết số chân địa chỉ và dữ liệu của IC2764?

b. Vẽ sơ đồ ghép nối và phân vùng địa chỉ cho các IC.

Bài 10. Sử dụng mạch giải mã 74LS138 thực hiện ghép nối VXL 8088 với bộ nhớ 32Kx16 từ các IC2764 (8Kx8). Biết địa chỉ của ô nhớ đầu tiên của vùng nhớ là B000h.

Hãy cho biết IC2764 thuộc loại IC nào? Cho biết số chân địa chỉ và dữ liệu của IC2764?

Vẽ sơ đồ ghép nối và phân vùng địa chỉ cho các IC.

Bài 11. Có 3 số 8 bit lấy ở 3 cổng vào có địa chỉ 1180h, 1181h, 1182h. Anh (chị) hãy:

- a. Sử dụng 74LS138 vẽ mạch giải mã cho các cổng có địa chỉ 1180h, 1181h, 1182h, 1183h.
- b. Viết chương trình hợp ngữ dạng .COM để VXL 8088/8086 thực hiện tìm max 3 số lấy từ 3 cổng vào và đưa kết quả ra cổng ra có địa chỉ 1183h.

CHƯƠNG III

HỌ VI ĐIỀU KHIỂN 8 BIT 8051

• Phần lý thuyết

**** Mục đích của chương:***

Giúp sinh viên so sánh được ưu, nhược điểm giữa vi xử lý và vi điều khiển. Giới thiệu một họ vi điều khiển nổi tiếng của Intel 8051. Nội dung của chương giúp sinh viên nắm vững cấu trúc vi điều khiển 8051 (cả phần cứng và phần mềm). Biết cách lập trình chúng (lập trình vào/ra các cổng, lập trình bộ Timer/Counter, lập trình truyền thông nối tiếp, lập trình ngắt).

**** Tóm tắt nội dung:***

Nội dung chương này tập trung trình bày về hệ vi điều khiển OnC. ip họ 8051, bao gồm: Cấu trúc tổ chức phần cứng, bản đồ bộ nhớ, các chế độ địa chỉ và tập lệnh; Lập trình hợp ngữ với những bài toán cơ bản: nhập xuất dữ liệu với cổng, định thời, truyền thông nối tiếp, ngắt, ...

3.1. CẤU TRÚC PHẦN CỨNG Họ VI ĐIỀU KHIỂN 8051

3.1.1. GIỚI THIỆU CHUNG

Khi sử dụng vi xử lý cần phải thiết kế một hệ thống gồm có:

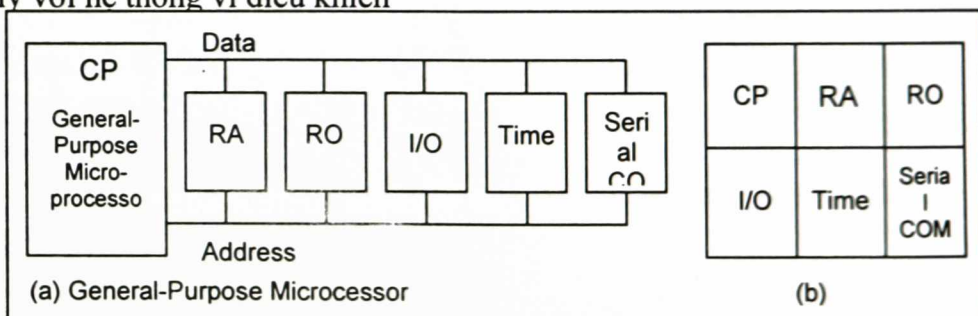
- Bộ vi xử lý.
- Bộ nhớ.
- Các IC ngoại vi (các mạch cổng giao tiếp).

Bộ nhớ dùng để chứa chương trình cho vi xử lý thực hiện và chứa dữ liệu xử lý, các IC 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 có đơn giản, ví dụ chỉ cần đóng mở 1 đèn led theo một thời gian yêu cầu nào đó 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, người sử dụng hiểu thật rõ về hệ thống. Một lý do chính 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 một ít 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). Hình 3.1 so sánh hệ thống vi xử lý với hệ thống vi điều khiển



Hình 3.1

Ứng dụng của vi điều khiển rất nhiều trong các hệ thống điều khiển công nghiệp, các dây chuyền sản xuất, các bộ điều khiển lập trình, máy giặt, máy điều hòa nhiệt độ, máy bơm xăng tự động,...

Có rất nhiều hãng chế tạo được vi điều khiển, hãng sản xuất nổi tiếng là ATMEL. Hãng Intel là nhà thiết kế. Có nhiều họ vi điều khiển mang các mã số khác nhau, một trong họ nổi tiếng là họ MCS-51.

Vi mạch tổng quát của họ MCS-51 là chip 8051 được sản xuất vào năm 1981 với các thông số kỹ thuật như sau:

- 4 KB ROM.
- 128 byte RAM.
- 4 port xuất nhập (I/O port) 8 bit.
- 2 bộ định thời 16 bit.
- Một cổng nối tiếp.
- Không gian nhớ chương trình ngoài 64 K.
- Bộ xử lý bit (thao tác trên các bit riêng rẽ).
- 210 vị trí nhớ được định địa chỉ, mỗi vị trí một bit.
- Nhân/chia trong 4 μ s.

Và sau đây là bảng các đặc tính kỹ thuật của họ MCS - 51 và MCS - 52:

Bảng 3.1

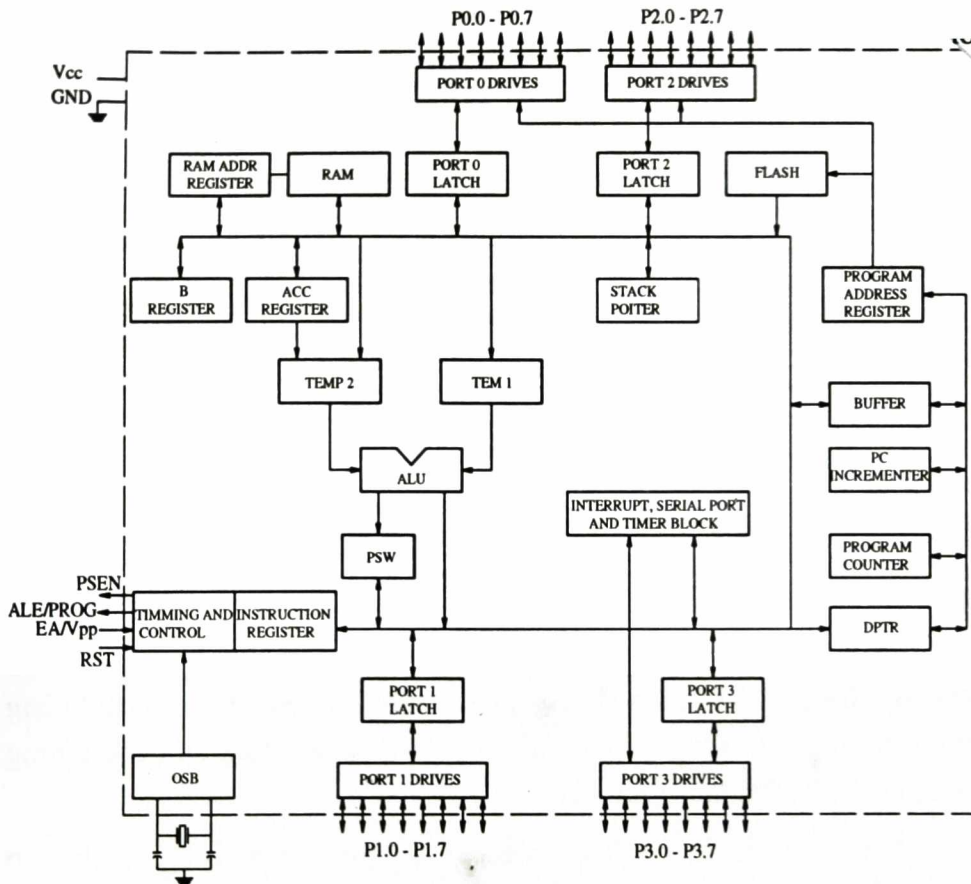
Ký hiệu	ROM	RAM	Chân I/O	Timer	Ngắt	Vcc	Đóng vỏ
AT89C51	4K	128	32	2	6	5V	40
AT89LV51	4K	128	32	2	6	3V	40
AT89C1051	1K	64	15	1	3	3V	20
AT89C2051	2K	128	15	2	6	3V	20
AT89C52	8K	128	32	3	8	5V	40
AT89LV52	8K	128	32	3	8	3V	40

3.1.2. SƠ ĐỒ CẤU TRÚC CỦA HỘ VI ĐIỀU KHIỂN 8051

Hình 3.2 mô tả sơ đồ cấu trúc bên trong vi điều khiển 8051

- Khối ALU đi kèm với các thanh ghi temp1, temp2 và thanh ghi trạng thái PSW.
- Bộ điều khiển logic (timing and control).
- Vùng nhớ RAM nội và vùng nhớ Flash Rom lưu trữ chương trình.
- 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/counmter.
- Thanh ghi A, B, dptr và 4 port0, port1, port2, port3 có chốt và đệm.
- Thanh ghi bộ đếm chương trình PC (program counter).
- Con trỏ dữ liệu dptr (data pointer).
- Thanh ghi con trỏ ngăn xếp SP (stack pointer).
- Thanh ghi lệnh IR (instruction register).
- Ngoài ra, còn có 1 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.

Các khối bên trong của vi điều khiển có các thành phần giống như đã trình bày ở chương 1 như khối ALU, thanh ghi temp1, thanh ghi temp2, thanh ghi bộ đếm chương trình PC, thanh con trỏ ngăn xếp, thanh ghi trạng thái PSW, thanh ghi lệnh IR, khối giải mã lệnh, khối điều khiển logic.



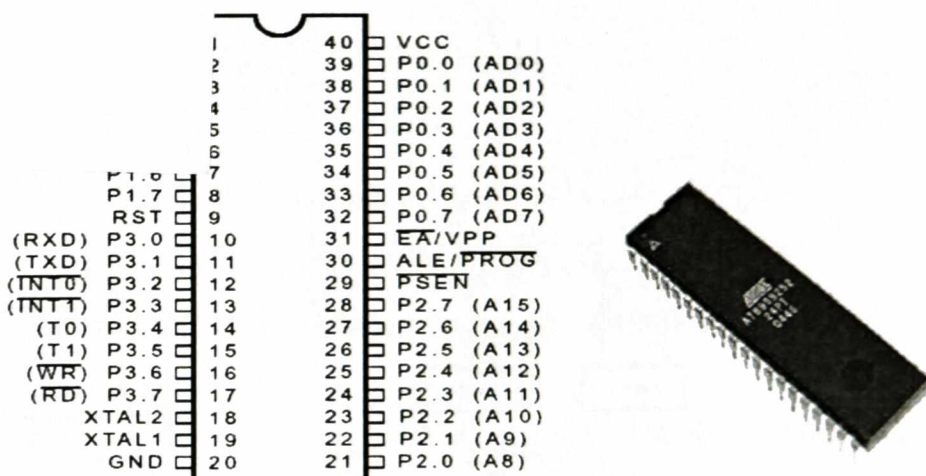
Hình 3.2. Sơ đồ cấu trúc của họ vi điều khiển 8051

3.1.3. MÔ TẢ CHỨC NĂNG CÁC CHÂN CỦA 8051

Sơ đồ chân của vi điều khiển 8051 được trình bày ở hình 3.3.

3.1.3.1. Các Port

Vi điều khiển 8051 có tất cả 40 chân. Trong đó có 32 chân dành cho 4 cổng (Port) là P0, P1, P2, P3. Mỗi cổng có 8 bit (chân – pin). Khi RESET mặc định chúng là các cổng ra và đều mang giá trị 1. Muốn chúng là cổng vào thì ghi 1 tới tất cả các bit của cổng.



Hình 3.3. Sơ đồ chân của vi điều khiển 8051

➤ Port 0

Port 0 là port có 2 chức năng với số thứ tự chân 32 – 39. Trong các hệ thống điều khiển đơn giản sử dụng bộ nhớ bên trong không dùng bộ nhớ mở rộng bên ngoài thì port 0 được dùng làm các đường điều khiển IO (Input - Output).

Trong các hệ thống điều khiển có quy mô lớn sử dụng bộ nhớ mở rộng bên ngoài thì port 0 có chức năng dồn kênh bus địa chỉ và bus dữ liệu AD7 - AD0.

➤ Port 1

Port 1 với số thứ tự chân 1- 8. Port1 chỉ có 1 chức năng dùng làm các đường điều khiển xuất nhập IO, port 1 không có chức năng khác.

➤ Port 2

Port 2 là port có 2 chức năng với số thứ tự chân 21 – 28.

Trong các hệ thống điều khiển đơn giản sử dụng bộ nhớ bên trong không dùng bộ nhớ mở rộng bên ngoài thì port 2 được dùng làm các đường điều khiển IO (Input - Output).

Trong các hệ thống điều khiển có quy mô lớn sử dụng bộ nhớ mở rộng bên ngoài thì port 2 có chức năng là bus địa chỉ cao A8 - A15.

➤ **Port 3**

Port 3 là port có 2 chức năng với số thứ tự chân 10 - 17.

Bình thường P3 cũng có chức năng là các cổng xuất/nhập.

Ngoài ra chúng còn có chức năng riêng (bảng 3.2)

Bảng 3.2

Bit	Tên	Địa chỉ	Chức năng
P3.0	RxD	B0H	Ngõ vào nhận dữ liệu nối tiếp.
P3.1	TxD	B1H	Ngõ xuất dữ liệu nối tiếp.
P3.2	INT0\	B2H	Ngõ vào ngắt cứng thứ 0.
P3.3	INT1\	B3H	Ngõ vào ngắt cứng thứ 1.
P3.4	T0	B4H	Ngõ vào của timer/counter thứ 0.
P3.5	T1	B5H	Ngõ vào của timer/counter thứ 1.
P3.6	WR\	B6H	Tín hiệu điều khiển ghi dữ liệu lên bộ nhớ ngoài.
P3.7	RD\	B7H	Tín hiệu điều khiển đọc dữ liệu từ bộ nhớ ngoài.

3.1.3.2. Các ngõ tín hiệu điều khiển

Ngõ tín hiệu PSEN (Program Store ENable)

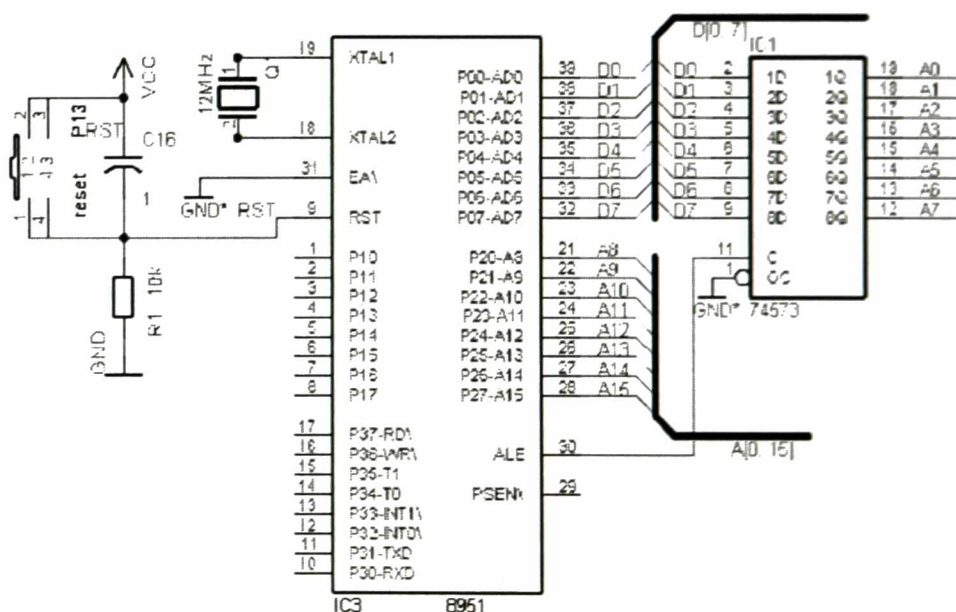
PSEN (chân 29): Cho phép đọc bộ nhớ chương trình mở rộng thường nối đến chân OE (output enable hoặc RD) của Eprom cho phép đọc các byte mã lệnh.

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

PSEN ở mức thấp trong thời gian vi điều khiển 8051 lấy lệnh. Các mã lệnh của chương trình đọc từ Eprom qua bus dữ liệu và được chốt vào thanh ghi lệnh IR bên trong 8051 để giải mã lệnh.

Ngõ tín hiệu điều khiển ALE (Address Latch Enable)

Khi vi điều khiển 8051 truy xuất bộ nhớ bên ngoài, port 0 có chức năng là bus tải địa chỉ và bus dữ liệu [AD7 – AD0]. Do đó, phải tách các đường dữ liệu và địa chỉ. Tín hiệu ra ALE ở chân thứ 30 dùng làm tín hiệu điều khiển để giải đa hợp các đường địa chỉ và dữ liệu khi kết nối chúng với IC chốt. Xem hình 3.4.



Hình 3.4. Ghép nối vi điều khiển 8951 với IC chốt, mạch Reset, tự thạch anh

Ngõ tín hiệu EA (External Access)

Tín hiệu vào EA ở chân 31 thường nối lên mức 1 hoặc mức 0. Nếu nối EA lên mức logic 1 (+5v) thì vi điều khiển sẽ thi hành chương trình từ bộ nhớ nội. Nếu nối EA với mức logic 0 (0V) thì vi điều khiển sẽ thi hành chương trình từ bộ nhớ ngoại.

Ngõ tín hiệu RST (Reset)

Ngõ vào RST ở chân 9 là ngõ vào Reset của 8051. Sơ đồ kết nối mạch reset như hình vẽ 3.4. Khi cấp điện cho hệ thống hoặc khi nhấn

nút reset thì mạch sẽ reset vi điều khiển. Khi reset thì tín hiệu reset phải ở mức cao ít nhất là 2 chu kỳ máy, khi đó các thanh ghi bên trong được nạp những giá trị thích hợp để khởi động hệ thống.

3.1.3.3. Các chân nguồn và đồng hồ

Các ngõ vào bộ dao động XTAL1, XTAL2

Bộ dao động được tích hợp bên trong 8051, khi sử dụng 8051 người thiết kế chỉ cần kết nối thêm tụ thạch anh và các tụ như sơ đồ hình 3.4. Tần số tụ thạch anh thường sử dụng cho 8051 là 12Mhz đến 24Mhz.

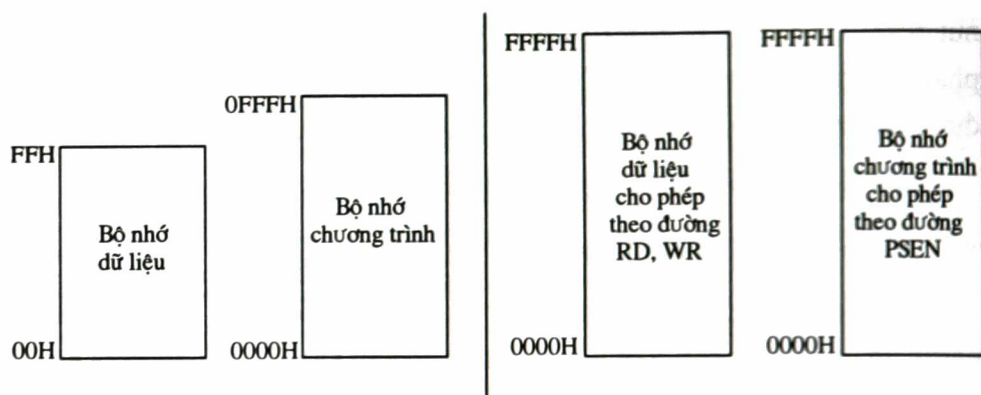
Chân 40 (Vcc) được nối lên nguồn 5V

Chân 20 GND nối mass

3.1.4. TỔ CHỨC BỘ NHỚ CỦA 8051

Vi điều khiển 8051 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 2 loại bộ nhớ: bộ nhớ dữ liệu và bộ chương trình. Bộ nhớ dữ liệu có 256 byte, bộ nhớ chương trình có dung lượng 4kbyte. Bộ nhớ mở rộng bên ngoài cũng được chia ra làm 2 loại bộ nhớ: bộ nhớ dữ liệu và bộ nhớ chương trình. Khả năng giao tiếp là 64kbyte cho mỗi loại. Hình 3.5 minh họa khả năng giao tiếp bộ nhớ của vi điều khiển 8051. 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 mã chương trình nên không cần phải khảo sát. Bộ nhớ ram nội bên trong là một bộ nhớ đặc biệt người sử dụng vi điều khiển cần phải nắm rõ các tổ chức và các chức năng đặc biệt của bộ nhớ này.



Hình 3.5. Bảng tóm tắt các vùng nhớ 8051

Sơ đồ cấu trúc bên trong của bộ nhớ này được trình bày như bảng 3.3.

RAM bên trong 8051 được phân chia như sau:

- Các thanh ghi có địa chỉ từ 00H đến 1FH.
- RAM địa chỉ hóa từng bit có địa chỉ từ 20H đến 2FH.
- RAM đa dụng từ 30H đến 7FH.

Các thanh ghi chức năng đặc biệt từ 80H đến FFH.

Bảng 3.3. Cấu trúc bộ nhớ dữ liệu bên trong vi điều khiển 8051

Địa chỉ byte	Địa chỉ bit									Địa chỉ byte	Địa chỉ bit									
7F	RAM đa dụng									FF										
										F0	F7	F6	F5	F4	F3	F2	F1	F0	B	
										E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC	
										D0	D7	D6	D5	D4	D3	D2	D1	D0	PSW	
30										B8	-	-	-	BC	BB	BA	B9	B8	IP	
2F	7F	7E	7D	7C	7B	7A	79	78												
2E	77	76	75	74	73	72	71	70		B0	B7	B6	B5	B4	B3	B2	B1	B0	P3	

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.

3.1.4.2. RAM địa chỉ hóa từng bit có địa chỉ từ 20H đến 2FH

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 1 lệnh đơn trong khi đó để xử lý các bit thì vi xử lý vẫn xử lý được, nhưng phải sử dụng rất nhiều lệnh để đạt được cùng một kết quả. 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.

Ví dụ: Để đặt bit 67H lên 1 ta có thể sử dụng một trong 2 lệnh sau:

```
MOV 2Ch, #10000000b ; hoặc  
SETB 67h
```

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.

3.1.4.3. RAM đa dụng có địa chỉ từ 30H – 7FH

Vùng nhớ RAM đa dụng gồm có 80 byte có địa chỉ từ 30H đến 7FH – vùng nhớ này không có gì đặc biệt so với 2 vùng nhớ trên chúng dùng để lưu trữ các loại dữ liệu khác nhau.

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

3.1.4.4. Các thanh ghi có chức năng đặc biệt

Các Port (tương ứng các ô nhớ có địa chỉ 80H, 90H, A0h, B0h)

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 IO.

Thanh ghi con trỏ ngăn xếp SP (ô nhớ có địa chỉ 81h)

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.

Ngăn xếp được truy xuất trực tiếp bằng các lệnh PUSH và POP để lưu trữ tạm thời và lấy lại dữ liệu, hoặc truy xuất ngầm bằng lệnh gọi chương trình con (ACALL, LCALL) và các lệnh trở về (RET, RETI) để lưu trữ địa chỉ của bộ đếm chương trình khi bắt đầu thực hiện chương trình con và lấy lại địa chỉ khi kết thúc chương trình con.

Thanh ghi con trỏ dữ liệu DPTR (ô nhớ có địa chỉ 82h và 83h)

Là 2 thanh ghi DPL (byte thấp) có địa chỉ là 82H và DPH (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 1 thanh ghi 16 bit có tên là DPTR và gọi là con trỏ dữ liệu - được dùng để lưu địa chỉ 16 bit khi truy xuất dữ liệu của bộ nhớ.

Thanh ghi PCON (ô nhớ có địa chỉ 87h)

Thanh ghi PCON (power control) có chức năng điều khiển công suất khi vi điều khiển làm việc hay ở chế độ chờ để giảm bớt công suất tiêu thụ nhất là khi nguồn cung cấp cho vi điều khiển là pin.

Các thanh ghi phục vụ cho Timer/Counter (các ô nhớ có địa chỉ từ 88h đến 8dh)

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

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

Thanh ghi TH0 và TL0 kết hợp lại tạo thành 1 thanh ghi 16 bit có chức năng lưu trữ xung đếm cho timer/counter T0. Tương tự cho 2 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.

Các thanh ghi phục vụ truyền thông nối tiếp (các ô nhớ có địa chỉ từ 98h đến 99h)

Là 2 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 2 thanh ghi này.

Các thanh ghi phục vụ ngắt (các ô nhớ có địa chỉ từ A8h đến B8h)

Là 2 thanh ghi IE và IP – thanh ghi IE (interrupt enable): thanh ghi điều khiển cho phép/không cho phép ngắt. IP (interrupt priority): thanh ghi điều khiển ưu tiên ngắt. Khi có sử dụng đến ngắt thì phải dùng đến 2 thanh ghi này. Mặc nhiên các thanh ghi này được khởi tạo ở chế độ cấm ngắt.

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 3.4:

Bảng 3.4

BIT	KÝ HIỆU	ĐỊA CHỈ	MÔ TẢ
PSW.7	C hoặc CY	D7H	Carry Flag: Cờ nhớ
PSW.6	AC	D6H	Auxiliary Carry Flag: Cờ nhớ phụ
PSW.5	F0	D5H	Flag 0 còn gọi là cờ Zero kí hiệu là Z
PSW.4	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.
			00 = Bank 0; ô nhớ có address 00H÷07H gán cho R0-R7
			01 = Bank 1; ô nhớ có address 08H÷0FH gán cho R0-R7
			10 = Bank 2; ô nhớ có address 10H÷17H gán cho R0-R7
			11 = Bank 3; ô nhớ có address 18H÷1FH gán cho R0-R7
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: cờ chẵn lẻ.

Chức năng từng bit trạng thái:

- Cờ Carry CY (Carry Flag): Cờ nhớ. C = 1 nếu phép toán cộng có tràn hoặc phép trừ có mượn.
- Cờ Carry phụ AC (Auxiliary Carry Flag): AC = 1 nếu có nhớ hoặc mượn từ 4 bit thấp lên 4 bit cao

- Cờ 0 (Flag 0): Cờ 0 (F0) còn gọi là cờ zero, cờ zero = 1 khi kết quả xử lý bằng 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 1 trong 4 bank thanh ghi như sau:

Bảng 3.5

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. Nếu kết quả cộng 2 số dương lớn hơn +127 hoặc cộng 2 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): P = 1 nếu tổng số bit 1 trong kết quả là một số lẻ.

Thanh ghi tổng A (ô nhớ có địa chỉ E0h)

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 thanh ghi A. Một chức năng quan trọng khác của thanh ghi A là để truyền dữ liệu từ bộ nhớ hoặc từ bên trong vi xử lý ra các thiết bị điều khiển bên ngoài thì dữ liệu đó phải chứa trong thanh ghi A.

Thanh ghi B (ô nhớ có địa chỉ F0h)

Thanh ghi B được dùng cùng với thanh ghi A để thực hiện các phép toán nhân chia.

3.1.5. HOẠT ĐỘNG RESET

Khi reset thì tín hiệu reset phải ở mức cao ít nhất là 2 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 trong 89C51 sau khi reset hệ thống được tóm tắt như sau:

Bảng 3.6

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	xxx0 0000 B
IE	0xx0 0000 B
Các thanh ghi định thời	00H
SCON	00H
SBUF	00H
PCON (HMOS)	0xxx xxxxB
PCON (CMOS)	0xxx 0000B

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

Nội dung của RAM trên chip 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).

3.2. CÁC CHẾ ĐỘ ĐỊA CHỈ CỦA 8051

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 8051 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 chỉ số.

3.2.1. KIỂU ĐỊNH ĐỊA CHỈ DÙNG THANH GHI (Register Addressing)

Sử dụng các thanh ghi chứa dữ liệu cần thao tác. Nhưng lưu ý không được dùng 2 thanh ghi R để thao tác với nhau.

Ví dụ:

```
MOV A, R1    ;copy nội dung thanh ghi R1 vào  
thanh ghi A
```

3.2.2. 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 256 byte bộ nhớ dữ liệu nội của vi điều khiển 8051.

Ví dụ:

```
MOV A, 05H      ;copy noi dung o nho co dia chi  
                ;05H vao thanh ghi A
```

3.2.3. ĐỊNH ĐỊA CHỈ GIÁN TIẾP (Indirect Addressing)

Kiểu định địa chỉ gián tiếp được tượng trưng bởi ký hiệu @ và được đặt trước các thanh ghi R0, R1 (không sử dụng các thanh ghi R2 – R7 trong chế độ địa chỉ này) hay DPTR. R0 và R1 có thể hoạt động như một thanh ghi con trỏ, nội dung của nó cho biết địa chỉ của một ô nhớ trong RAM nội mà dữ liệu sẽ ghi hoặc sẽ đọc.

Ví dụ:

```
MOV A, @R1      ;copy noi dung o nho co dia chi  
                ;dat trong thanh ghi R1 vao thanh ghi A
```

3.2.4. ĐỊNH ĐỊA CHỈ TỨC THỜI (Immediate Addressing)

Kiểu định địa chỉ tức thời được tượng trưng bởi ký hiệu # và được đặt trước một hằng số. Lệnh này thường dùng để nạp 1 giá trị là 1 hằng số vào thanh ghi hoặc ô nhớ.

Ví dụ:

```
MOV A, #30H     ;nap du lieu 30H vao thanh ghi A
```

3.2.5. ĐỊNH ĐỊA CHỈ TƯƠNG ĐỐI

Kiểu định địa chỉ tương đối chỉ sử dụng với những lệnh nhảy. Nơi nhảy đến có giá trị từ -128 đến +127. Lệnh này có mã lệnh 2 byte.

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

Ví dụ:

```
SJMP X1      ;nhay den nhan co ten X1 nam trong  
              ;tam vuc 256 byte
```

3.2.6. ĐỊNH ĐỊA CHỈ TUYỆT ĐỐI

Kiểu định địa chỉ tuyệt đối được dùng với các lệnh ACALL và AJMP. Các lệnh này có mã lệnh 2 byte.

Ví dụ:

```
AJMP X1      ;nhay den nhan co ten X1 nam trong  
              ;tam vuc 2Kbyte
```

3.2.7. ĐỊNH ĐỊA CHỈ DÀI (Long Addressing)

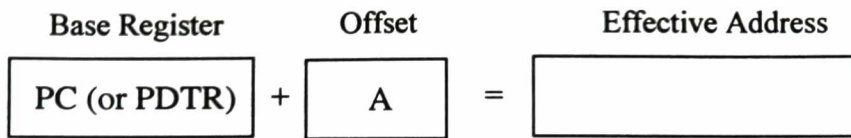
Kiểu định địa chỉ dài được dùng với lệnh LCALL và LJMP. Các lệnh này có mã lệnh 3 byte. Định địa chỉ dài là có thể gọi 1 chương trình con hoặc có thể nhảy đến bất kỳ vùng nhớ nào vùng nhớ 64K.

Ví dụ:

```
LJMP X1      ;nhay den nhan co ten X1 nam trong  
              ;tam vuc 64Kbyte
```

3.2.8. 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 bộ đếm dữ liệu DPTR” kết hợp với “một giá trị lệnh (offset) còn gọi là giá trị tương đối [thường lưu trong thanh ghi]” để tạo ra 1 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:

MOVC A, @A + DPTR ;lay du lieu trong o nho
;DPTR+A de nap vao thanh ghi A

3.3. TẬP LỆNH CỦA 8051

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 trực tiếp.
- 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 Ri chỉ có 2 thanh ghi là R0 và R1.
- #data tượng trưng dữ liệu tức thì.
- 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 A11 – 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.

3.3.1. NHÓM LỆNH SỐ HỌC

1. Lệnh cộng không nhớ

ADD A, Rn; Cộng nội dung thanh ghi A với nội dung thanh ghi Rn, kết quả lưu trong thanh ghi A.

ADD A, direct; Cộng nội dung của ô nhớ có địa chỉ direct với nội dung thanh ghi A, kết quả chứa ở thanh ghi A.

Ví dụ: Giả sử A có nội dung 0D9h và ô nhớ có địa chỉ 30h lưu nội dung 0B8h, lệnh:

ADD A, 30h ; A=91h, (C)=1.

ADD A, @Ri; Cộng nội dung của ô nhớ có địa chỉ chứa trong thanh ghi Ri với thanh ghi A, kết quả lưu trữ trong thanh ghi A.

ADD A, #data; Cộng dữ liệu data 8 bit (d0 đến d7) với nội dung thanh ghi A, kết quả lưu trữ trong A.

2. Lệnh cộng có nhớ

ADDC A, Rn; Cộng nội dung thanh ghi A với nội dung thanh ghi Rn với bit C, kết quả lưu trong thanh ghi A.

Ví dụ: Giả sử A có nội dung 0D9h, R0 có nội dung là 0B8h, (C) = 1, lệnh:

ADDC A, R0 ; A=92h, (C)=1.

ADDC A, direct; Cộng nội dung của ô nhớ có địa chỉ direct nội dung thanh ghi A và bit C, kết quả chứa ở thanh ghi A.

ADDC **A, @Ri;** Cộng nội dung của ô nhớ có địa chỉ chứa trong thanh ghi Ri với thanh ghi A với bit C, kết quả lưu trữ trong thanh ghi A.

ADDC **A, #data;** Cộng dữ liệu data 8 bit (d0 đến d7) với nội dung thanh ghi A và bit C, kết quả lưu trữ trong A.

3. Lệnh trừ có nhớ

SUBB **A, Rn;** Trừ nội dung thanh ghi A cho nội dung thanh ghi Rn và trừ cho cờ Carry, kết quả lưu trong thanh ghi A.

Ví dụ: Giả sử A có nội dung 0B9h, thanh ghi R0 có nội dung là 5Ah và (C) = 1, lệnh:

SUBB A, R0 ; A=5Eh, (C)=0.

SUBB **A, direct;** Trừ nội dung thanh ghi A cho nội dung của ô nhớ có địa chỉ direct và trừ cho cờ Carry, kết quả chứa ở thanh ghi A.

SUBB **A, @Ri;** Trừ nội dung của thanh ghi A cho dữ liệu của ô nhớ có địa chỉ chứa trong thanh ghi Ri và trừ cho cờ carry, kết quả lưu; trữ trong thanh ghi A.

SUBB **A, #data;** Trừ nội dung thanh ghi A cho dữ liệu 8 bit d0 đến d7 và trừ cho cờ carry, kết quả lưu trữ trong A.

4. Lệnh tăng: (increment: tăng lên 1 đơn vị)

INC **A** ; Tăng nội dung thanh ghi A lên 1.

Ví dụ: Giả sử A có nội dung FFh, lệnh:

```
INC A ;A=00h.
```

INC Rn ; Tăng nội dung thanh ghi Rn lên 1.

- INC direct** ; Tăng nội dung của ô nhớ có địa chỉ trực tiếp lên 1.
- INC @Ri** ; Tăng nội dung của ô nhớ có địa chỉ chứa trong thanh ghi Ri lên 1.
- INC dptr** ; Tăng nội dung của thanh ghi con trỏ dữ liệu dptr lên 1.

5. Lệnh giảm: (Decrement: giảm xuống 1 đơn vị)

- DEC A** ; Giảm nội dung thanh ghi A xuống 1.
- DEC Rn** ; Giảm nội dung thanh ghi Rn xuống 1.
- DEC direct** ; Giảm nội dung của ô nhớ có địa chỉ direct ở byte thứ 2 xuống 1.
- DEC @Ri** ; Giảm nội dung của ô nhớ có địa chỉ chứa trong thanh ghi Ri xuống 1.

6. Lệnh nhân thanh ghi A với thanh ghi B

MUL AB; 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 trữ trong thanh ghi A, 8 bit cao lưu trữ trong thanh ghi B.

Ví dụ: Giả sử thanh ghi A có nội dung là 50h, thanh ghi B có nội dung 0A0h, lệnh:

MUL AB ; $50h * A0h = 3200h$, (A) = 00, (B) = 32h.

7. Lệnh chia thanh ghi A cho thanh ghi B

DIV AB; Nội dung của thanh ghi A chia cho nội dung của thanh ghi B, kết quả của phép chia lưu trữ trong thanh ghi A, số dư lưu trữ trong thanh ghi B.

8. Lệnh điều chỉnh thập phân nội dung thanh ghi A

DA A

Ý nghĩa: Nếu 4 bit thấp $A3A2A1A0 > 9$ hoặc bit $AC = 1$ thì $A3A2A1A0 + 6$, kết quả lưu trữ lại trong A. Nếu 4 bit cao $A7A6A5A4 > 9$ hoặc bit $Cy = 1$ thì $A7A6A5A4 + 6$, kết quả lưu trữ lại thành ghi A. Kết quả sau cùng trong thanh ghi A là số BCD. Lệnh DA A chỉ dùng sau lệnh ADD mà không bao giờ dùng sau lệnh INC.

3.3.2. NHÓM LỆNH LOGIC

9. Lệnh nhân logic

ANL A, Rn; Nội dung thanh ghi A and với nội dung thanh ghi Rn, kết quả lưu trữ trong thanh ghi A.

Ví dụ:

MOV A, #10110011b ;A= 10110011b

MOV R0, #11001011b ;R0=11001011b

ANL A, R0 ;A= 10000011b

ANL A, direct; Nội dung thanh ghi A and với nội dung của ô nhớ có địa chỉ direct, kết quả chứa ở thanh ghi A.

ANL A, @Ri; Nội dung thanh ghi A and với ô nhớ có địa chỉ chứa trong thanh ghi Ri, kết quả lưu trữ trong thanh ghi A.

ANL A, #data; Nội dung của thanh ghi A and với dữ liệu d0 đến d7, kết quả lưu trữ trong thanh ghi A.

ANL direct, A; Nội dung ô nhớ có địa chỉ direct and với nội dung của thanh ghi A, kết quả lưu trữ vào ô nhớ.

ANL direct, #data; Nội dung của ô nhớ có địa chỉ direct and với 8 bit dữ liệu 8 bit, kết quả lưu trữ vào ô nhớ.

10. Lệnh cộng logic

ORL A, Rn; Nội dung thanh ghi A or với nội dung thanh ghi Rn, kết quả lưu trữ trong thanh ghi A.

Ví dụ:

```
MOV A , #10110011b
```

```
MOV R0, #11001011b
```

```
ORL A, R0 ;A=11111011b.
```

ORL A, direct; Nội dung thanh ghi A or với nội dung của ô nhớ có địa chỉ direct, kết quả chứa ở thanh ghi A.

ORL A, @Ri; Nội dung thanh ghi A or với ô nhớ có địa chỉ chứa trong thanh ghi Ri, kết quả lưu trữ trong thanh ghi A.

ORL A, #data; Nội dung của thanh ghi A or với dữ liệu 8 bit data (từ d0 đến d7), kết quả lưu trữ trong thanh ghi A.

ORL direct, A; Nội dung ô nhớ có địa chỉ direct or với nội dung của thanh ghi A, kết quả lưu trữ trong ô nhớ có địa chỉ direct.

ORL direct, #data; Nội dung của ô nhớ có địa chỉ direct or với dữ liệu 8 bit (từ d0 đến d7) ở byte thứ 3, kết quả lưu trữ trong ô nhớ.

11. Lệnh cộng đảo logic

XRL A, Rn; Nội dung thanh ghi A ex-or với nội dung thanh ghi Rn, kết quả lưu trữ trong thanh ghi A.

Ví dụ:

```
MOV A , #10110011b
```

```
MOV R0, #11001011b
```

```
XRL A, R0 ;A=01111000b.
```

XRL A, direct; Nội dung thanh ghi A ex-or với nội dung của ô nhớ có địa chỉ direct, kết quả chứa ở thanh ghi A.

XRL A, @Ri; Nội dung thanh ghi A ex-or với ô nhớ có địa chỉ chứa trong thanh ghi Ri, kết quả lưu trữ trong thanh ghi A.

XRL A, #data; Nội dung của thanh ghi A ex-or với dữ liệu data, kết quả lưu trữ trong thanh ghi A.

XRL direct, A; Nội dung ô nhớ có địa chỉ direct ex-or với nội dung của thanh ghi A, kết quả lưu trữ vào ô nhớ.

XRL direct, #data; Nội dung của ô nhớ có địa chỉ direct ex-or với 8 bit dữ liệu data 8 bit, kết quả lưu trữ vào ô nhớ.

12. Lệnh xóa nội dung thanh ghi A

CLR A; Nội dung thanh ghi A bằng zero.

13. Lệnh bù nội dung thanh ghi A

CPL A; Nội dung thanh ghi A được lấy bù 1, kết quả chứa ở A.

Ví dụ:

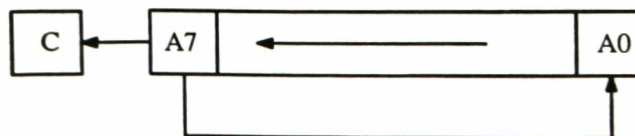
MOV A, #10110011b

CPL A ; A=01001100b.

14. Lệnh xoay trái nội dung thanh ghi A

RL A ;(rotate to the left)

Ý nghĩa: Nội dung thanh ghi A được xoay trái 1 bit minh họa như hình vẽ.



Ví dụ:

```
MOV A, #1011 0011b
```

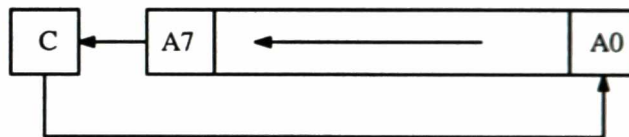
```
RL A
```

Giá trị ban đầu của C ta không cần quan tâm đến. Kết quả sau khi xoay thì (A) = 0110 0111b và cờ (C) = 1 là do bit MSB của thanh ghi chuyển sang.

15. Lệnh xoay trái nội dung thanh ghi A và bit carry

```
RLC A
```

Ý nghĩa: Nội dung thanh ghi A và bit C được xoay trái 1 bit.



16. Lệnh xoay phải nội dung thanh ghi A

```
RR A ;(rotate to the right)
```

Ý nghĩa: Nội dung thanh ghi A được xoay phải 1 bit ngược với lệnh RL A.

17. Lệnh xoay phải nội dung thanh ghi A và bit carry

```
RRC A
```

Ý nghĩa: Nội dung của A và bit C được xoay phải 1 bit ngược với lệnh RLC A.

18. Lệnh xoay 4 bit thanh ghi A

```
SWAP A; Hoán chuyển 4 bit thấp và 4 bit cao  
;trong thanh ghi A.
```

Ví dụ:

```
MOV A, #3EH
```

```
SWAP A ; A=E3H
```

3.3.3. NHÓM LỆNH DI CHUYỂN DỮ LIỆU

19. Lệnh MOV

MOV A, Rn; Chuyển nội dung của thanh ghi Rn vào thanh ghi A, nội dung thanh ghi Rn vẫn giữ nguyên.

Ví dụ: Giả sử thanh ghi R0 có nội dung là 32h, lệnh:

MOV A, R0 ; A=32h, R0=32h.

Giá trị ban đầu chứa trong A thì không cần quan tâm.

MOV A, direct; Chuyển nội dung của ô nhớ trong Ram nội có địa chỉ direct vào thanh ghi A.

MOV A, @Ri; Chuyển nội dung ô nhớ trong Ram nội, có địa chỉ chứa trong thanh ghi Ri, vào thanh ghi A.

MOV A, #data; Nạp dữ liệu 8 bit data (d0 đến d7) vào thanh ghi A.

MOV Rn, A; Chuyển nội dung của thanh ghi A vào thanh ghi Rn.

MOV Rn, direct; Chuyển nội dung của ô nhớ trong Ram nội có địa chỉ direct vào thanh ghi Rn.

MOV Rn, #data; Nạp dữ liệu 8 bit data (d0 đến d7) vào thanh ghi Rn.

MOV direct, A; Chuyển nội dung của thanh ghi A vào ô nhớ trong Ram nội có địa chỉ direct.

MOV direct, Rn; Chuyển nội dung của thanh ghi Rn vào ô nhớ trong Ram nội có địa chỉ direct.

MOV direct1, direct2; Chuyển nội dung của ô nhớ trong Ram nội có địa chỉ direct2 vào ô nhớ có địa chỉ direct1.

MOV direct, @Ri; Chuyển nội dung ô nhớ có địa chỉ chứa trong thanh ghi Ri vào ô nhớ có địa chỉ direct.

MOV direct, #data; Nạp dữ liệu data 8 bit (d0 đến d7) vào ô nhớ có địa chỉ direct.

MOV @Ri, A; Chuyển nội dung của thanh ghi A vào ô nhớ trong Ram nội có địa chỉ chứa trong thanh ghi Ri.

MOV @Ri, direct; Chuyển nội dung ô nhớ có địa chỉ direct vào ô nhớ có địa chỉ chứa trong thanh ghi Ri.

MOV @Ri, #data; Nạp dữ liệu data 8 bit (d0 đến d7) vào ô nhớ có địa chỉ chứa trong thanh ghi Ri.

MOV dptr, #data1; Nạp dữ liệu data 16 bit vào thanh ghi dptr.

20. Lệnh MOVC

MOVC A, @A+DPTR; Chuyển nội dung của ô nhớ chương trình, có địa chỉ chứa bằng dptr cộng với giá trị chứa trong A, chuyển vào thanh ghi A.

MOVC A, @A+PC; Chuyển nội dung của ô nhớ chương trình, có địa chỉ chứa bằng PC cộng với giá trị chứa trong A được chuyển vào thanh ghi A.

21. Lệnh MOVX

MOVX A, @Ri; Chuyển nội dung ô nhớ dữ liệu ngoài có địa chỉ chứa trong thanh ghi Ri vào thanh ghi A.

MOVX A, @DPTR; Chuyển nội dung của ô nhớ ngoài có địa chỉ chứa trong thanh ghi dptr vào thanh ghi A.

MOVX @ Ri, A; Chuyển nội dung của thanh ghi A ra ô nhớ dữ liệu ngoài có địa chỉ chứa trong thanh ghi Ri.

MOVX @DPTR, A; Chuyển nội dung của thanh ghi A ra ô nhớ dữ liệu ngoài có địa chỉ chứa trong thanh ghi dptr.

22. Lệnh cất nội dung ô nhớ trực tiếp vào ngăn xếp
PUSH direct

Ý nghĩa: 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.

23. Lệnh lấy dữ liệu từ ngăn xếp trả về ô nhớ trực tiếp
POP direct

Ý nghĩa: 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.

24. Lệnh trao đổi dữ liệu

XCH A, Rn ; Trao đổi nội dung của thanh ghi Rn với thanh ghi A.

XCH A, Direct; Trao đổi nội dung của thanh ghi A với nội dung ô nhớ có địa chỉ direct.

XCH A, @Ri; Trao đổi nội dung của ô nhớ có địa chỉ chứa trong thanh ghi Ri với thanh ghi A.

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

XCHD A, @Ri; 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 A.

3.3.4. NHÓM LỆNH XỬ LÝ BIT

26. Lệnh xóa bit carry

Cú pháp: **CLR C**

Ý nghĩa: Xóa bit C về 0.

27. Lệnh xóa bit

Cú pháp: **CLR bit**

Ý nghĩa: Xóa bit về 0.

Ví dụ:

CLR P1.0 ;xoa P1.0=0

28. Lệnh đặt bit carry

Cú pháp: SETB C

Ý nghĩa: Đặt bit C = 1.

29. Lệnh đặt bit

Cú pháp: SETB bit

Ý nghĩa: Đặt bit lên 1.

Ví dụ:

SETB P1.0 ;dat P1.0=1

30. Lệnh bù bit carry

Cú pháp: CPL C

Ý nghĩa: Bù bit carry, nếu trước đó C = 1 thì C = 0, ngược lại C = 0 thì C = 1.

31. Lệnh bù bit

Cú pháp: CPL bit

Ý nghĩa: Bù bit. Nếu trước đó bit này = 0 thì kết quả bit này bằng 1 và ngược lại nếu trước đó bằng 1 thì nó sẽ bằng 0.

32. Lệnh and bit carry với bit

Cú pháp: ANL C, bit

Ý nghĩa: Bit C and với bit có địa chỉ được xác định, kết quả chứa ở bit C.

33. Lệnh and bit carry với bù bit

Cú pháp: ANL C, /bit

Ý nghĩa: Bit C and với bù bit có địa chỉ được xác định, kết quả chứa ở bit C.

34. Lệnh or bit carry với bit

Cú pháp: ORL C, bit

Thời gian thực hiện là hai chu kỳ máy.

Ý nghĩa: Bit C or với bit có địa chỉ được xác định, kết quả chứa ở bit C.

35. Lệnh or bit carry với bù bit

Cú pháp: ORL C, /bit

Ý nghĩa: Bit C or với bù bit có địa chỉ được xác định, kết quả chứa ở bit C.

36. Lệnh di chuyển bit vào bit carry

Cú pháp: MOV C, bit

Ý nghĩa: Bit có địa chỉ được xác định được chuyển vào bit C.

37. Lệnh di chuyển bit carry vào bit

Cú pháp: MOV bit, C

Ý nghĩa: Bit C được chuyển vào bit có địa chỉ được xác định.

3.3.5. NHÓM LỆNH RẼ NHÁNH

38. Lệnh gọi chương trình con

❖ 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

Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Ý nghĩa: Khi lệnh này được thực hiện thì 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).

❖ **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

– Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy.

Ý nghĩa: Khi lệnh này được thực hiện thì vi điều khiển sẽ thực hiện chương trình con tại địa chỉ addr16. Lệnh này có thể gọi chương trình con ở đâu cũng được trong vùng 64kbyte. Addr16 của chương trình con có thể thay bằng nhãn (tên của chương trình con).

39. Lệnh trở về từ chương trình con

Cú pháp: RET

Ý nghĩa: Lệnh này sẽ kết thúc chương trình con, vi điều khiển sẽ trở lại chương trình chính để tiếp tục thực hiện chương trình.

40. Lệnh trở về từ chương trình con phục vụ ngắt

Cú pháp: RETI

Ý nghĩa: 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.

41. Lệnh nhảy không điều kiện

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

Cú pháp: AJMP addr11

Ý nghĩa: 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. 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 dài 16 bit**

Cú pháp: LJMP addr16

Ý nghĩa: 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; rel [relative: tương đối]

Ý nghĩa: 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

Lệnh sẽ nhảy đến nơi có địa chỉ bằng nội dung của A cộng với dptr để tiếp tục thực hiện chương trình tại đó.

42. Lệnh nhảy có điều kiện

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

Cú pháp: JZ rel (jump zero)

Ý nghĩa: Nếu bit Z = 1 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

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

Cú pháp: JNZ rel

Ý nghĩa: Nếu Z = 0 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

❖ **Lệnh nhảy nếu bit carry = 1**

Cú pháp: JC rel

Ý nghĩa: Nếu bit carry C = 1 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

❖ **Lệnh nhảy nếu bit carry = 0**

Cú pháp: JNC rel

Ý nghĩa: Nếu bit carry C = 0 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

❖ **Lệnh nhảy nếu bit = 1**

Cú pháp: JB bit, rel

Ý nghĩa: Nếu nội dung của bit có địa chỉ bit được xác định bằng 1 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

❖ **Lệnh nhảy nếu bit = 0**

Cú pháp: JNB bit, rel

Ý nghĩa: Nếu nội dung của bit có địa chỉ bit được xác định bằng 0 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ bằng rel.

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

Cú pháp: JBC bit, rel

Ý nghĩa: Nếu bit được xác định bằng 1 thì bit này được xóa về 0 và vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

43. Lệnh so sánh và nhảy nếu không bằng nhau

- ❖ **So sánh thanh ghi A với ô nhớ và nhảy nếu không bằng nhau**

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

Ý nghĩa: Nếu nội dung của A khác nội dung của ô nhớ có địa chỉ direct thì lệnh sẽ nhảy đến và thực hiện lệnh tại địa chỉ rel.

- ❖ **So sánh thanh ghi A với dữ liệu tức thì và nhảy nếu không bằng nhau**

Cú pháp: **CJNE** A, #data, rel

Ý nghĩa: Nếu nội dung của A khác nội dung của data 8 bit thì lệnh sẽ nhảy đến và thực hiện lệnh tại địa chỉ rel.

- ❖ **So sánh thanh ghi Rn với dữ liệu tức thì và nhảy nếu không bằng nhau**

Cú pháp: **CJNE** Rn, #data, rel

Ý nghĩa: Nếu nội dung của Rn khác nội dung của data 8 bit thì lệnh sẽ nhảy đến và thực hiện lệnh tại địa chỉ rel.

- ❖ **So sánh ô nhớ có địa chỉ trong Ri với dữ liệu tức thì và nhảy nếu không bằng nhau**

Cú pháp: **CJNE** @Ri, #data, rel

Ý nghĩa: Nếu nội dung của ô nhớ có địa chỉ lưu trong thanh ghi Ri khác data 8 bit thì lệnh sẽ nhảy đến thực hiện lệnh tại địa chỉ rel.

44. Lệnh giảm thanh ghi và nhảy

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

Ý nghĩa: Nội dung của thanh ghi Rn giảm đi 1 và nếu kết quả trong thanh ghi Rn sau khi giảm khác 0 thì vi điều khiển sẽ thực hiện

chương trình tại địa chỉ rel, nếu kết quả bằng 0 thì vi điều khiển sẽ tiếp tục thực hiện lệnh kế.

45. Lệnh giảm ô nhớ trực tiếp và nhảy

Cú pháp: DJNZ direct, rel

Ý nghĩa: Nếu nội dung của ô nhớ có địa chỉ direct giảm đi 1 và nếu kết quả sau khi giảm khác 0 thì vi điều khiển sẽ thực hiện chương trình tại địa chỉ rel, ngược lại nếu kết quả bằng 0 thì vi điều khiển sẽ tiếp tục thực hiện lệnh kế.

46. Lệnh Nop

Cú pháp: NOP

Ý nghĩa: Khi gặp lệnh này vi điều khiển sẽ không làm gì, mục đích của lệnh này là làm tăng khoảng thời gian trễ.

3.4. LẬP TRÌNH HỢP NGỮ CHO VI ĐIỀU KHIỂN 8051

3.4.1. TRÌNH DỊCH HỢP NGỮ

Trong phần lập trình hợp ngữ cho vi xử lý 8086 ta đã trình bày rất rõ khái niệm hợp ngữ và trình dịch hợp ngữ. Trình hợp dịch được dùng để dịch chương trình hợp ngữ (.ASM) ra dạng mã máy hay mã lệnh (.HEX), còn các ngôn ngữ bậc cao được dịch thành mã máy bằng một chương trình gọi là trình biên dịch. Ví dụ khi viết một chương trình bằng C, ta cần phải sử dụng trình biên dịch C để dịch chương trình ra dạng mã máy.

Chương trình 8051IDE hoặc Keil C là những chương trình bao gồm 3 chức năng: Soạn thảo chương trình nguồn, dịch chương trình nguồn, mô phỏng được ứng dụng để lập trình cho họ vi điều khiển MCS – 51. Trình soạn thảo được ứng dụng để viết và sửa chương trình nguồn, sau đó dịch chương trình nguồn bằng cách chọn lệnh Assembler từ menu Assembler. Trong quá trình dịch, chương trình sẽ kiểm tra và phát hiện lỗi nếu có. Sau khi dịch xong sử dụng chương

166

trình mô phỏng để quan sát từng bước thực hiện chương trình. Có thể quan sát sự thay đổi nội dung của các thanh ghi, các cờ, các cổng và bộ nhớ khi chương trình thực hiện.

3.4.2. KHUÔN DẠNG CỦA CHƯƠNG TRÌNH HỢP NGỮ

Một chương trình hợp ngữ bao gồm một chuỗi các dòng lệnh hợp ngữ. Một lệnh hợp ngữ có một từ gọi nhớ (mnemonic), và tùy theo từng lệnh mà sau đó có một hoặc hai toán hạng. Từ gọi nhớ là lệnh yêu cầu CPU thực hiện lệnh còn các toán hạng là các dữ liệu cần thao tác.

Nói chung lệnh hợp ngữ của 8051 có 4 trường như sau (tương tự chương II):

[Nhãn:]	[Từ gọi nhớ]	[Các toán hạng]	[;Chú thích]
---------	--------------	-----------------	--------------

Ví dụ:

ORG	0H		;Bat dau (origin) tai ngan nho 0
MOV	A, #0		;nap 0 vao thanh ghi A
MOV	R4, #12H		;nap 12H vao R4
MOV	R5, #13H		;nap 13H vao R5
ADD	A, R4		;cong noi dung R4 vao A (A=A+R4)
ADD	A, R5		;cong noi dung R5 vao A (A=A+R5)
HERE:	SJMP	HERE	;o lai trong vong lap
END			;ket thuc chuong trinh

3.4.3. KIỂU DỮ LIỆU VÀ CHỈ DẪN

3.4.3.1. Kiểu dữ liệu

Bộ vi điều khiển 8051 chỉ có một kiểu dữ liệu, đó là kiểu 8 bit và độ dài thanh ghi cũng là 8 bit. Nhiệm vụ của lập trình viên đối với dữ liệu trên 8 bit là phân chia dữ liệu thành các phần 8 bit (từ 00H đến FFH) để CPU xử lý. Dữ liệu của 8051 có thể là số âm hoặc số dương.

Ví dụ: Hãy viết chương trình cộng hai số 16 bit. Các số đó là 3CE7H và 3B8DH, cất tổng vào 2 thanh ghi R6 và R7 trong đó R7 chứa byte thấp.

```
CLR                ;xoa co CY=0
MOV  A, #0E7H ;nap byte thap vao A → A=E7H
ADD  A, #8DH  ;cong byte thap vao A → A=74H va
CY=1
MOV  R7, A ;luu byte thap cua tong vao R7
MOV  A, #3CH ;nap byte cao vao A → A=3CH
ADDC A, #3BH ;cong byte cao co nho vao A →
A=78H
MOV  R6, A ;luu byte cao vao R6
```

3.4.3.2. Chỉ dẫn

❖ Chỉ dẫn địa chỉ byte DB

DB là định nghĩa dữ liệu 8 bit và được sử dụng rộng rãi khi lập trình hợp ngữ. 8 bit dữ liệu được định nghĩa có thể ở dạng thập phân, nhị phân, hexa hay ASCII và hợp ngữ luôn chuyển về dạng hexa.

Ví dụ:

```
DATA1: DB 12 ;so thap phan
DATA2: DB 00010100B ;so nhi phan
DATA3: DB 30H ;so hexa
DATA4: DB '2341' ;cac so ASCII
DATA5: DB 'DHKTCN' ;ky tu ASCII
```

❖ Chỉ dẫn ORG (origin)

Dùng để báo địa chỉ bắt đầu của chương trình. Số sau ORG có thể là số ở dạng hexa hoặc thập phân.

❖ **Chỉ dẫn EQU**

Được dùng để định nghĩa một hằng số, chỉ dẫn EQU không dùng ô nhớ để cất dữ liệu, mà thực hiện gán một hằng số cho nhãn sao cho khi nhãn xuất hiện trong chương trình thì giá trị hằng số sẽ được thay thế cho nhãn. Mục đích của việc sử dụng EQU là khi muốn thay đổi giá trị của một hằng (cố định) được sử dụng nhiều trong chương trình, lập trình viên chỉ cần sử dụng EQU và thay đổi một lần, khi đó toàn bộ những vị trí mà hằng xuất hiện sẽ đều bị thay đổi.

Ví dụ:

```
CONST      EQU      30  
  
...  
MOV  A, #CONST      ;A=30
```

❖ **Chỉ dẫn END**

Chỉ dẫn này báo cho trình hợp dịch biết kết thúc tệp nguồn (asm). Chỉ dẫn END là dòng cuối cùng của chương trình 8051.

3.4.4. LẬP TRÌNH VÀO/RA CÁC CỔNG

Như ta đã biết cả 4 cổng P0, P1, P2, P3 đều có 8 chân và tạo thành cổng 8 bit. Tất cả các cổng khi Reset đều được cấu hình làm cổng ra. Để làm đầu vào thì cần ghi 1 tới các bit của cổng.

3.4.4.1. Truy cập tất cả các bit của cổng

Để thay đổi giá trị bit tất cả bit cổng ta có thể thực hiện theo ba cách.

– *Thông qua thanh ghi tích lũy*

Quay trở lại ví dụ cổng P0: Viết chương trình để 8051 gửi ra cổng P0 giá trị 55H rồi lại gửi ra P0 giá trị AAH.

```
                MOV      A, #55H  
BACK:          MOV      P0, A  
                ACALL    DELAY
```

```

CPL      A
SJMP     BACK
DELAY:   DELAY:  MOV R6, #0FFH
HERE2:   MOV R7, #0FFH
HERE1:   DJNZ R7, HERE1
          DJNZ R6, HERE2
          RET

```

- Truy cập trực tiếp các cổng

```

BACK:    MOV      P0, #55H
          ACALL   DELAY5S
          MOV     P0, #0AAH
          ACALL   DELAY
          SJMP    BACK
DELAY:   DELAY:  MOV R6, #0FFH
HERE2:   MOV R7, #0FFH
HERE1:   DJNZ R7, HERE1
          DJNZ R6, HERE2
          RET

```

- Truy cập các cổng bằng kỹ thuật: Đọc – sửa đổi – ghi

Vẫn ví dụ trên ta sẽ sử dụng kỹ thuật đọc – sửa đổi – ghi.

```

BACK:    MOV      P0, #55H; P1 = 01010101
          ACALL   DELAY
          XLR     P1, #0FFH; P1 XOR 1111
          1111 = AAH
          ACALL   DELAY
          SJMP    BACK
DELAY5S: DELAY:  MOV R6, #0FFH
HERE2:   MOV R7, #0FFH

```

RET

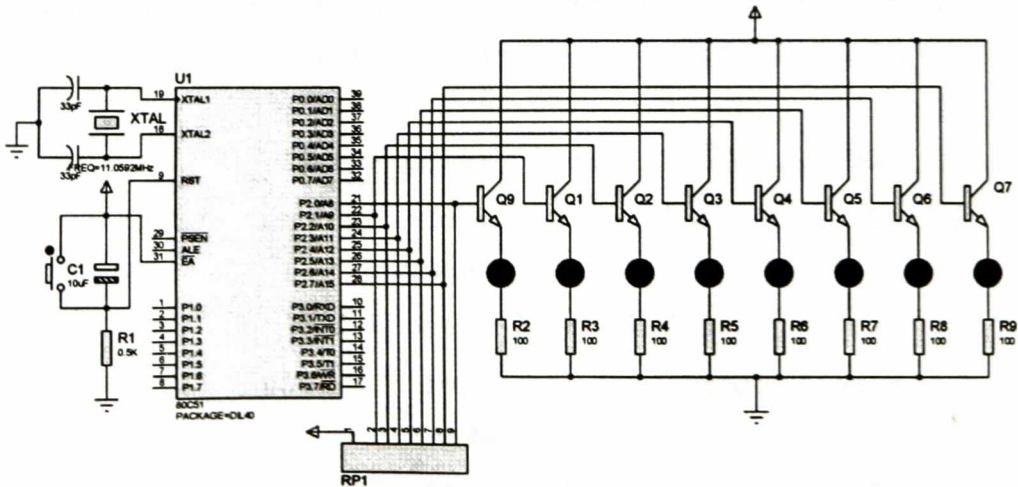
3.4.4.2. Truy cập từng bit của cổng

Nhiều ứng dụng ta chỉ cần truy cập 1 đến hai bit của cổng, thay vì truy cập cả 8 bit. Một điểm mạnh của 8051 là có khả năng truy cập từng bit mà không làm thay đổi các bit còn lại của cổng.

Ví dụ: Viết chương trình đề 8051 thực hiện các công việc sau:

- + Duy trì kiểm tra P0.0 cho đến khi bit này lên cao.
- + Khi P0.0 lên cao hãy ghi 12H vào cổng P1.
- + Gửi một xung “cao” xuống “thấp” đến P0.2 (H – to – L).

```
AGAIN: JNB     P0.0, AGAIN      ;thoat khi P0.0=1
      MOV     P1, #12H        ;xuat 12H den P1
      SETB    P0.2            ;dua P0.2 len cao
      NOP                      ;tao do rong
      CLR     P0.2            ;dua P0.2xuong thap
```



Hình 3.6

Ví dụ tổng quát: Cho mạch điện ghép nối vi điều khiển 8051 với các LED như hình 3.6. Hãy viết chương trình để 8051 điều khiển các LED sáng lần lượt từ phải qua trái. Khi cả 8 LED cùng sáng thì cùng tắt và lặp lại chu kỳ như ban đầu.

Giải:

```
ORG 0000H
```

```
PORT2:
```

```
MOV P2, #00000000B ;tat port 2
```

```
ACALL DELAY ;goi chuong trinh con  
delay
```

```
MOV P2, #00000001B ;sang 1 led
```

```
ACALL DELAY ;goi chuong trinh con  
delay
```

```
MOV P2, #00000011B ;sang 2 led
```

```
ACALL DELAY ;goi chuong trinh con  
delay
```

```
MOV P2, #00000111B ;sang 3 led
```

```
ACALL DELAY ;goi chuong trinh con  
delay
```

```
MOV P2, #00001111B ;sang 4 led
```

```
ACALL DELAY ;goi chuong trinh con  
delay
```

```
MOV P2, #00011111B ;sang 5 led
```

```
ACALL DELAY ;goi chuong trinh con  
delay
```

```
MOV P2, #00111111B ;sang 6 led
```

```

ACALL DELAY          ;goi  chuong  trinh  con
                      delay
MOV P2, #01111111B   ;sang 7 led
ACALL DELAY          ;goi  chuong  trinh  con
                      delay
MOV P2, #11111111B   ;sang 8 led
ACALL DELAY          ;goi  chuong  trinh  con
                      delay

SJMP PORT2

DELAY:                MOV R6, #0FFH
HERE2:                MOV R7, #0FFH
HERE1:                DJNZ R7, HERE1
                      DJNZ R6, HERE2
                      RET

END

```

Vẫn ví dụ trên nhưng ta có thể viết theo cách 2 ngắn gọn hơn nhiều sử dụng lệnh xoay thanh ghi thông qua cờ nhớ CY.

```
org 0000h
```

```
LAPLAI:
```

```

MOV P2, #00h         ;tat port 2

SANGLED:
ACALL DELAY          ;goi  chuong  trinh  con
                      delay

SETB C               ;lam cho bit C = 1

MOV A, P2             ;chuyen port2  vao  thanh
                      ghi A

```



```

RLC A          ;xoay thanh ghi A sang
                trai
MOV P2, A      ;tra lai cho port2
JNC SANGLED    ;nhay ve de thuc hien tiep
SJMP LAPLAI    ;sau khi 8 led sang het
                thi ;quay lai tu dau

DELAY:         MOV R6, #0FFH
HERE2:         MOV R7, #0FFH
HERE1:         DJNZ R7, HERE1
                DJNZ R6, HERE2
                RET

END

```

3.4.5. BỘ ĐẾM/ĐỊNH THỜI CỦA 8051 VÀ LẬP TRÌNH

Trong những ứng dụng kỹ thuật nhiều khi ta phải tạo ra khoảng thời gian trễ để điều khiển các thiết bị theo thời gian hoặc đếm sự kiện từ bên ngoài, khi đó ta phải sử dụng các Timer và Counter để thực hiện nhiệm vụ trên.

Trong vi điều khiển 8051 có 2 timer/counter T0 và T1. Các timer hay counter chỉ là một và chính là bộ đếm có chức năng đếm xung.

Nếu ta sử dụng ở chế độ timer thì thời gian định thời nhân với chu kỳ của mỗi xung sẽ có được lượng thời gian cần thiết – ở chế độ timer vì điều khiển đếm xung nội lấy từ mạch dao động bên trong vì điều khiển có chu kỳ ổn định. Chế độ timer dùng để định thời gian chính xác để điều khiển các thiết bị theo thời gian.

Nếu sử dụng ở chế độ counter thì ta chỉ cần quan tâm đến số lượng xung đếm được – không cần quan tâm đến chu kỳ của xung

đếm. Một ứng dụng cho chế độ counter là có thể sử dụng vi điều khiển làm các mạch đếm sản phẩm.

Đến đây ta có thể xem timer hay counter là 1 và chú ý rằng tại mỗi một thời điểm ta chỉ sử dụng một trong 2 hoặc là timer hoặc là counter.

3.4.5.1. Các thanh ghi phục vụ cho Timer/counter

❖ Thanh ghi của bộ Timer 0 và Timer 1 (hình 3.7 và 3.8)

Các giá trị đếm được của timer/counter T0 thì lưu trong 2 thanh ghi TH0 và TL0 – mỗi thanh ghi 8 bit kết hợp lại thành 16 bit.

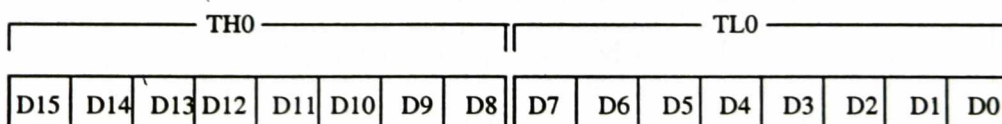
Tương tự, các giá trị đếm được của timer/counter T1 thì lưu trong 2 thanh ghi TH1 và TL1 – mỗi thanh ghi 8 bit kết hợp lại thành 16 bit.

Các thanh ghi này cũng có thể được truy cập và được đọc như các thanh ghi khác như thanh ghi A, B, R0, R1 ...

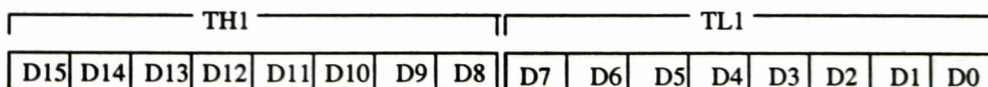
Ví dụ:

```
MOV TL0, #55H
```

```
MOV R0, TH0
```



Hình 3.7. Các thanh ghi của Timer 0



Hình 3.8. Các thanh ghi của Timer 1

❖ Thanh ghi TMOD (hình 3.9)

Thanh ghi TMOD là thanh ghi 8 bit gồm 4 bit thấp cho Timer 0 và 4 bit cao cho Timer 1.

– Các bit M0, M1: Là các bit chế độ dùng để chọn chế độ 0, 1, 2 và 3 của các bộ Timer 0 và Timer 1 (bảng 3.7). Ta chỉ tập trung vào các chế độ được sử dụng rộng rãi là chế độ 1 và 2.

(MSB)				(LSB)			
GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer1				Timer0			

Hình 3.9. Thanh ghi TMOD

– Bit C/T (đồng hồ/bộ định thời): Là bit dùng để xác định bộ định thời được sử dụng làm bộ tạo trễ hay bộ đếm sự kiện. Nếu C/T = 0 thì nó là bộ tạo trễ. C/T = 1 thì bộ định thời được chọn là bộ đếm.

Bảng 3.7. Các chế độ làm việc

M1	M0	Mode	Chế độ hoạt động
0	0	0	Chế độ định thời 13 bit. Bộ định thời/bộ đếm 8 bit, định tỉ lệ trước 5 bit
0	1	1	Chế độ định thời 16 bit, không định tỉ lệ trước
1	0	2	Chế độ 8 bit tự nạp lại
1	1	3	Chế độ bộ định thời chia tách

Ví dụ: Xác định chế độ và bộ định thời của các trường hợp sau:

a) MOV TMOD, #20H

b) MOV TMOD, #12H

Giải:

Trường hợp a: TMOD = 20H = 0010 0000B. Đối chiếu với thanh ghi TMOD ta thấy đây là chế độ 2 bộ định thời Timer 1 được chọn.

Trường hợp b: TMOD = 12H = 0001 0010B. Đối chiếu với thanh ghi TMOD ta thấy đây là chế độ 1 bộ định thời Timer 1 và chế độ 2 bộ định thời Timer 0 được chọn.

Nếu C/T = 0 thì tần số thạch anh trên chip 8051 làm nguồn cho đồng hồ bộ định thời. Điều đó có nghĩa là giá trị của tần số thạch anh của 8051 quyết định tốc độ đồng hồ của bộ định thời 8051. Tần số của bộ định thời luôn bằng 1/12 tần số của thạch anh trên 8051.

Ví dụ: Tìm tần số đồng hồ và chu kỳ của bộ định thời của các hệ thống xây dựng trên 8051 với tần số thạch anh như sau:

a) $f_{XTAL} = 12\text{MHz}$

b) $f_{XTAL} = 16\text{MHz}$

c) $f_{XTAL} = 11,0592\text{MHz}$

Giải:

a) $\frac{1}{12} \times 12\text{MHz} = 1\text{MHz}$ và $T = \frac{1}{1\text{MHz}} = 1\mu\text{s}$

b) $\frac{1}{12} \times 16\text{MHz} = 1,333\text{Mz}$ và $T = \frac{1}{1,333\text{MHz}} = 0,75\mu\text{s}$

c) $\frac{1}{12} \times 11,0592\text{MHz} = 0,9216\text{MHz}$ và $T = \frac{1}{0,9216\text{MHz}} = 1,085\mu\text{s}$

– Bit cổng GATE: Xác định cách khởi động bộ định thời.

GATE = 0: Khởi động và dừng bộ định thời được thực hiện bằng phần mềm nhờ các bit khởi động bộ định thời TR (Timer Start) là TR0 và TR1 (hai bit này nằm trên thanh ghi TCON (hình 3.10). Lệnh thực hiện khởi động và dừng Timer 0 tương ứng là “SETB TR0” và “CLR TR0” còn đối với Timer 1 là “SETB TR1” và “CLR TR1”.

Nếu dùng phần cứng từ bên ngoài để khởi động và dừng bộ định thời thì ta cần đặt bit GATE = 1.

D7				D0			
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Hình 3.10. Thanh ghi TCON

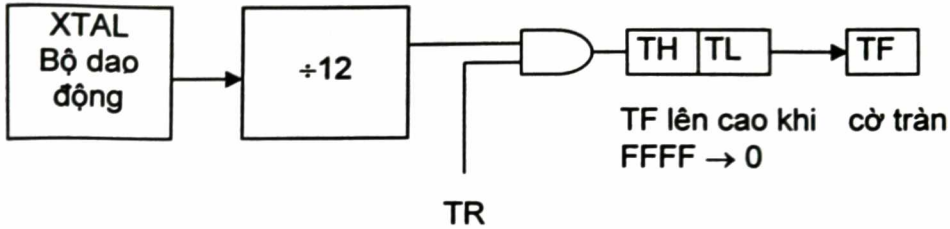
- + TF1: Cờ tràn của Timer 1
- + TR1: Điều khiển hoạt động của Timer 1. Nó được thiết lập bằng phần mềm để bật/tắt Timer 1.
- + TF0: Cờ tràn của Timer 0
- + TR0: Điều khiển hoạt động của Timer 1. Nó được thiết lập bằng phần mềm để bật/tắt Timer 0.
- + IE1, IT1, IE0, IT0: Các bit phục vụ cho ngắt.

3.4.5.2. Lập trình các bộ định thời gian của 8051

❖ Lập trình chế độ 1 (Model 1)

Dưới đây là những đặc trưng và những phép toán của Model 1.

- Đó là bộ định thời 16 bit, do vậy các giá trị từ 0000 – FFFFH được nạp vào thanh ghi TL và TH của bộ định thời.
- Sau đó bộ định thời được khởi động nhờ lệnh “SETB TR0” cho Timer 0 và “SETB TR1” cho Timer 1.
- Sau khi được khởi động thì bộ định thời bắt đầu đếm tăng. Bộ định thời đếm lên cho đến khi đạt giới hạn FFFFH. Khi đó nó sẽ quay vòng từ FFFFH về 0 và bật cờ định thời TF (Timer Flag) lên mức cao.
- Bộ định thời sau khi đạt giá trị giới hạn thì thực hiện quay vòng về 0. Để lặp lại quá trình đếm, các thanh ghi TH và TL phải được nạp lại giá trị ban đầu và cờ TF cần được xóa về 0.



Hình 3.11

Các bước lập trình chế độ 1:

Để tạo ra bộ trễ thời gian khi dùng chế độ 1 của bộ định thời cần phải thực hiện các bước dưới đây:

- Bước 1: Nạp giá trị cho thanh ghi TMOD xác định bộ định thời nào (Timer 0 hay Timer 1) và chế độ nào được chọn.
- Bước 2: Nạp giá trị đếm ban đầu cho các thanh ghi TL và TH.
- Bước 3: Khởi động bộ định thời.
- Bước 4: Kiểm tra trạng thái bật của cờ bộ định thời. Thoát khỏi vòng lặp khi TF được bật lên.
- Bước 5: Dừng bộ định thời.
- Bước 6: Xoá cờ TF cho vòng kế tiếp.
- Bước 7: Quay trở lại bước 2 để nạp lại giá trị cho TL và TH.

Để tính toán thời gian trễ chính xác (với XTAL cụ thể) ta có thể sử dụng một trong hai cách tính.

Tính theo số hexa	Tính theo số thập phân
$(FFFFH - YYXXH + 1) \times T$ Trong đó: YYXXH: Giá trị khởi tạo của TH, TL tương ứng T: Chu kỳ bộ định thời	Đổi YYXXH ra số thập phân NNNNN. Sau đó tính $(65536 - NNNNN) \times T$

Ví dụ 1: Viết chương trình tạo ra xung vuông độ dày 50% trên chân P0.1 sử dụng bộ định thời Timer 0 để tạo trễ.

```

CLR  TF0                      ;xoa co Timer 0
MOV  TMOD,#01H;su dung Timer 0 che do 1
HERE: MOV  TL0,#0F2H;TL0 = F2H, byte thap
      MOV  TH0,#0FFH;TH0 = FFH, byte cao
      CPL  P0.1                ;su dung chan P0.1
      ACALL      DELAY        ;tao tre
      SJMP HERE                ;nap lai TH, TL
;***** Tao tre dung Timer 0.
DELAY:  SETB    TR0;khoi dong bo dinh thoi Timer 0
AGAIN:  JNB     TF0,AGAIN;kiem tra co bo dinh thoi
      CLR      TR0            ;tat Timer 0
      CLR      TF0            ;xoa co Timer 0
      RET

```

Với giả thiết tần số XTAL = 11, 0592MHz ta sẽ tính được thời gian tạo trễ như sau. Bộ định thời làm việc ở tần số đồng hồ bằng 1/12 tần số XTAL. Do vậy ta có:

$$f = 11,0592/12 = 0,9216\text{MHz và } T = 1/f = 1/0,9216 = 1,085\mu\text{s.}$$

Thời gian trong chương trình con DELAY là $(FFFFH - FFF2H + 1) \times 1,085 = 14 \times 1,085 = 15,19\mu\text{s.}$

Ví dụ 2: Giả sử tần số XTAL = 11,0592 MHz. Hãy viết chương trình tạo xung có tần số 2KHz trên chân P1.5.

Giải:

- Chu kỳ sóng vuông là: $T = 1/f = 1/2\text{KHz} = 0,5\text{ms} = 500\mu\text{s.}$
- Khoảng thời gian cao (hoặc thấp) là: $0,5T = 0,5.500 = 250\mu\text{s.}$
- Số nhịp đếm cần trong khoảng thời gian đó là: $250/1,085 = 230 \text{ nhịp}$

- Giá trị cần nạp vào thanh ghi TH và TL là: $65536 - 230 = 65306 = \text{FF1AH}$.

Chương trình được viết như sau:

```

CLR    TF0            ;xoa co Timer 0
MOV    TMOD,#01H;su dung Timer 0 che do 1
AGAIN: MOV    TL0,#1AH    ;TL0 = 1AH, byte thap
MOV    TH0,#0FFH    ;TH0 = FFH, byte cao
SETB   TR0            ;khởi động bộ định
                        ;thời Timer 0
BACK:   JNB    TF0,BACK    ;kiểm tra có bộ định thời
CLR    TR0            ;tắt Timer 0
CPL    P1.5            ;đảo bit P1.5
CLR    TF0            ;xoa co Timer 0
SJMP   AGAIN          ;nhập lại TH, TL

```

Ví dụ 3: Viết chương trình tạo trễ 8s.

DELAY8S:

```

CLR    TF1            ;xoa co Timer 1
MOV    TMOD,#10H;su dung Timer 1 che do 1
MOV    R3,200        ;tạo trễ thời gian lớn
AGAIN: MOV    TL1,#0FEH;TL1 = FEH, byte thap
MOV    TH1,#6FH;TH1 = 6FH, byte cao
SETB   TR1            ;khởi động Timer 1
BACK:   JNB    TF1,BACK;kiểm tra có bộ định thời
CLR    TR1            ;tắt Timer 1
CLR    TF1            ;xoa co Timer 1
DJNZ   R3,AGAIN ;nếu R3 ≠ 0 thì nhập lại

```

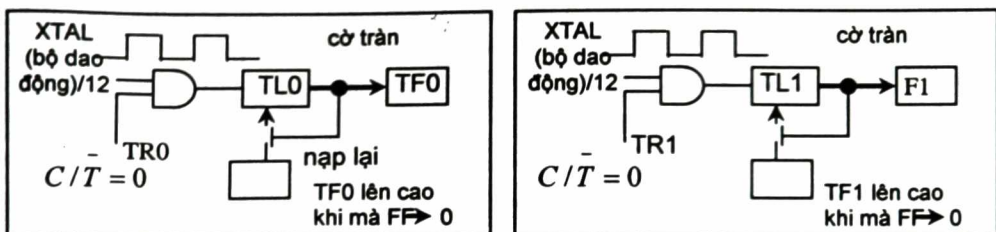
Từ chương trình trên ta thấy:

- Số nhíp đếm trong khoảng thời gian là: $65536 - 28670$
(6FFE_H) = 36866
- Thời gian trễ trong AGAIN: $36866 \times 1,085 = 39999610 \mu s \approx 0,04s$
- Thời gian trong DELAY: $200 \times 0,04 = 8s$

❖ Lập trình chế độ 2 (Model 2)

Những đặc trưng và những phép toán của Model 2.

- Đó là bộ định thời 8 bit, do vậy các giá trị từ 00 – FF_H được nạp vào thanh ghi TH của bộ định thời.
- Sau khi TH được nạp giá trị 8 bit thì 8051 sao nội dung đó vào TL và bộ định thời được khởi động nhờ lệnh “SETB TR0” cho Timer 0 và “SETB TR1” cho Timer 1.
- Sau khi được khởi động thì bộ định thời bắt đầu đếm tăng bằng cách tăng thanh ghi TL. Bộ định thời đếm lên cho đến khi đạt giới hạn FF_H. Khi đó nó sẽ quay vòng từ FF_H về 0 và bật cờ định thời TF (Timer Flag) lên mức cao.
- Khi thanh ghi TL quay từ FF_H về 00, cờ TF được bật lên 1 thì thanh ghi TL tự động được nạp lại với giá trị ban đầu được đặt tại TH. Để lặp lại quá trình đơn giản chỉ cần xóa cờ TF và để bộ định thời tự làm việc mà không cần lập trình viên can thiệp hay nạp lại giá trị ban đầu. Vì vậy, chế độ 2 được gọi là chế độ tự nạp lại.



Hình 3.12

Các bước lập trình chế độ 2:

Để tạo ra bộ trễ thời gian khi dùng chế độ 2 của bộ định thời thì cần phải thực hiện các bước dưới đây:

- Bước 1: Nạp giá trị cho thanh ghi TMOD xác định bộ định thời nào (Timer 0 hay Timer 1) và chế độ nào được chọn.
- Bước 2: Nạp giá trị đếm ban đầu cho thanh ghi TH.
- Bước 3: Khởi động bộ định thời.
- Bước 4: Kiểm tra trạng thái bật của cờ bộ định thời. Thoát khỏi vòng lặp khi TF được bật lên.
- Bước 5: Xoá cờ TF.
- Bước 6: Quay trở lại bước 4 vì chế độ 2 là chế độ tự động nạp lại.

Ví dụ 1: Giả sử tần số thạch anh là XTAL = 11, 0592 MHz. Hãy tìm:

a) Tần số sóng vuông được tạo ra trên chân P1.0 trong đoạn chương trình sau.

b) Tần số nhỏ nhất có thể có được bằng chương trình này và giá trị TH cần có là bao nhiêu?

```

MOV      TMOD, #20H;chon Timer 1 che do 2
MOV      TH1, #5  ;TH1 = 5
SETB     TR1      ;khoi dong Timer1
BACK:    JNB     TF1, BACK;duy tri kiem tra TF1
CPL      P1.0     ;tao xung tren P1.0

```



```
CLR      TF1      ;xóa cờ TF1
SJMP BACK      ;chờ đợi 2 chu kỳ nạp lại
```

Giải:

a) Phần cao của xung có thời gian tồn tại là: $(256 - 5) \times 1,085 = 272,33\mu s$

– Cả chu kỳ của xung là: $T = 2 \times 272,33 = 544,66\mu s$

– Tần số là: $f = 1/T = 1/544,66 = 0,001836\text{MHz} = 1,836\text{KHz}$

b) Để có được tần số nhỏ nhất tức là T phải lớn nhất, thì TH = 00H. Khi đó $T = 2 \times 256 \times 1,084 = 555,52\mu s$ và tần số nhỏ nhất sẽ là: $f = 1/T = 1,8\text{KHz}$.

3.4.5.3. Lập trình cho bộ đếm

Nếu $C/T = 1$ thì ta sử dụng bộ đếm khi đó các xung được đưa đến các chân 14 và 15. Các chân này có tên là T0 và T1 và đều thuộc về cổng P3.

Bảng 3.8

Chân	Chân cổng	Chức năng	Mô tả
14	P3.4	T0	Đầu vào ngoài của bộ đếm 0
15	P3.5	T1	Đầu vào ngoài của bộ đếm 1

Ví dụ: Giả sử có một nguồn xung đồng hồ được cấp tới chân T0. Hãy viết chương trình cho bộ đếm 0 ở chế độ 2 đếm các xung và hiển thị trạng thái của số đếm TL0 trên cổng P1.

Giải:

```
MOV      TMOD, #00000110B;chọn bộ đếm 0 chế
                        ;độ 2, bit C/T = 1 xung ngoài
MOV      TH0, #0      ;xóa TH0
SETB     P3.4          ;lay dấu vào T0
AGAIN: SETB     TR0     ;khởi động bộ đếm
BACK: MOV      A, TL0   ;lay ban sao kq đếm
```

```

MOV    P1,A           ;dua ra cong P1
JNB    TF0,Back       ;duy tri kiem tra TF0
CLR    TR0            ;dung bo dem
CLR    TF0            ;xoa co TF0
SJMP   AGAIN          ;tiếp tục đếm

```

Trong chương trình trên ta phải sử dụng lệnh “SETB P3.4” vì các cổng được thiết lập dành cho đầu vào, muốn nó là đầu ra thì phải bật nó lên cao.

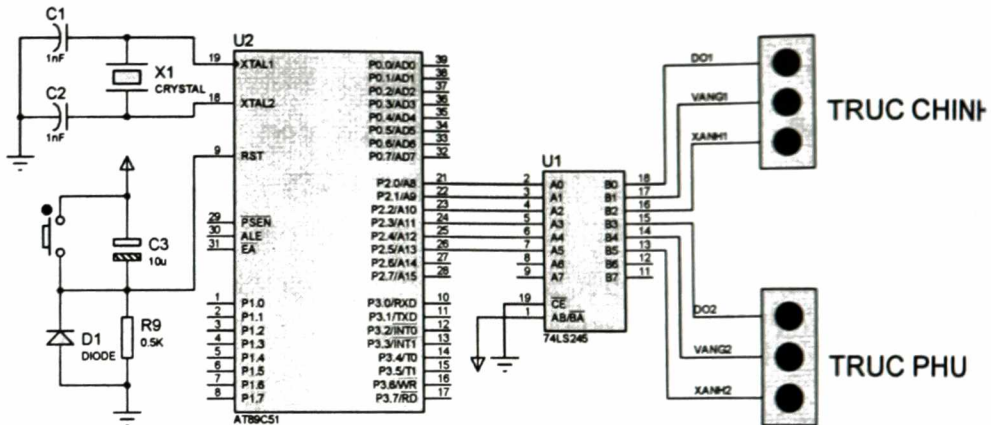
❖ Trường hợp bit GATE = 1

Như ta đã biết, khi bit GATE = 0 các bộ định thời/bộ đếm được khởi động bằng phần mềm. Nhưng nếu GATE = 1 thì các bộ định thời/bộ đếm lại được khởi động bằng phần cứng và được thực hiện từ bên ngoài thông qua các chân P3.2 và P3.3. Tuy nhiên, nếu TRx được bật lên bằng lệnh “SETB TRx” thì cũng cho phép khởi động và dừng bộ định thời từ bên ngoài tại bất kỳ thời điểm nào thông qua công tắc chuyển mạch đơn giản. Ví dụ: ứng dụng 8051 để thiết kế bộ báo động bằng cách sử dụng Timer 0. Bộ Timer 0 được bật lên bằng lệnh “SETB TR0” và nằm ngoài sự kiểm soát của người dùng. Tuy nhiên, nếu nối một công tắc chuyển mạch tới chân P3.2 thì ta có thể dừng và khởi động bộ định thời và như vậy có thể tắt báo động.

Ví dụ tổng quát: Cho hệ thống ghép nối vi điều khiển 8051 với đèn giao thông như hình vẽ 3.13. Hãy sử dụng các Timer để viết chương trình điều khiển các đèn sáng tắt theo quy luật như sau:

Bảng 3.9

Đèn	Bit điều khiển	Thời gian tồn tại
Xanh 1	P2.2	9s
Vàng 1	P2.1	1s
Đỏ 1	P2.0	
Xanh 2	P2.5	6s
Vàng 2	P2.4	1s
Đỏ 2	P2.3	



Hình 3.13. Sơ đồ mô phỏng đèn giao thông

Giải:

```
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX;
CHUONG TRINH DIEU KHIEN HE THONG DEN GIAO THONG
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ORG 0000H
PORT2:

    MOV P2,#00001100B    ;XANH CHINH, DO PHU
    LCALL DELAY9S        ;GOI CTC DELAY9S
    MOV P2,#00001010B    ;VANG CHINH, DO PHU
    LCALL DELAY1S        ;GOI CTC DELAY1S
    MOV P2,#00100001B    ;XANH PHU, DO CHINH
    LCALL DELAY6S        ;GOI CTC DELAY6S
    MOV P2,#00010001B    ;VANG PHU, DO CHINH
    LCALL DELAY1S        ;GOI CTC DELAY1S
    SJMP PORT2           ;LAP LAI

;XXXXXXX CHUONG TRINH CON DELAY XXXXXXXXXX
;XXXXXXX CTC DELAY9S
DELAY9S:

    CLR TF1              ;XOA CO TIMER 1
```

```

        MOV TMOD,#10H      ;SU DUNG TIMER 1 CHE DO 1
        MOV R3,#225        ;TAO TRE 9S
AGAIN1: MOV TL1,#0FEH      ;TL1 = FEH, BYTE THAP
        MOV TH1,#6FH       ;TH1 = 6FH, BYTE CAO
        SETB TR1           ;KHOI DONG TIMER 1
BACK1:  JNB TF1,BACK1      ;KIEM TRA CO BO DINH THOI
        CLR TR1           ;TAT TIMER 1
        CLR TF1           ;XOA CO TIMER 1
        DJNZ R3,AGAIN1     ;NEU R3 KHAC 0 THI NAP

```

LAI

RET

;XXXXXXXXXX CTC DELAY6S

DELAY6S:

```

        CLR TF1           ;XOA CO TIMER 1
        MOV TMOD,#10H     ;SU DUNG TIMER 1 CHE DO 1
        MOV R3,#150       ;TAO TRE 6S
AGAIN2: MOV TL1,#0FEH     ;TL1 = FEH, BYTE THAP
        MOV TH1,#6FH      ;TH1 = 6FH, BYTE CAO
        SETB TR1          ;KHOI DONG TIMER 1
BACK2:  JNB TF1,BACK2     ;KIEM TRA CO BO DINH THOI
        CLR TR1           ;TAT TIMER 1
        CLR TF1           ;XOA CO TIMER 1
        DJNZ R3,AGAIN2    ;NEU R3 KHAC 0 THI NAP

```

LAI

RET

;XXXXXXXXXX CTC DELAY1S

DELAY1S:

```

        CLR TF1           ;XOA CO TIMER 1
        MOV TMOD,#10H     ;SU DUNG TIMER 1 CHE DO 1
        MOV R3,#25        ;TAO TRE 1S

```



```

AGAIN3: MOV TL1,#0FEH      ;TL1 = FEH, BYTE THAP
        MOV TH1,#6FH       ;TH1 = 6FH, BYTE CAO
        SETB TR1           ;KHOI DONG TIMER 1
BACK3:  JNB TF1,BACK3       ;KIEM TRA CO BO DINH THOI
        CLR TR1            ;TAT TIMER 1
        CLR TF1            ;XOA CO TIMER 1
        DJNZ R3,AGAIN3      ;NEU R3 KHAC 0 THI NAP
LAI
RET
END

```

Từ chương trình trên ta thấy:

- Số nhịp đếm trong khoảng thời gian là: $65536 - 28670$
(6FFE) = 36866
- Thời gian trễ trong AGAIN1, AGAIN2, AGAIN3: $36866 \times 1,085 = 39999,610 \mu s \approx 0,04s$
- Thời gian trong DELAY9s: $225 \times 0,04 = 9s$
- Thời gian trong DELAY6s: $150 \times 0,04 = 6s$
- Thời gian trong DELAY1s: $25 \times 0,04 = 1s$

Cũng vẫn chương trình trên ta có thể viết chương trình con DELAY bằng các bộ timer khác nhau ở các chế độ 1 hoặc 2 khác nhau. Thời gian trễ hoàn toàn có thể đặt theo ý muốn.

3.4.6. LẬP TRÌNH TRUYỀN THÔNG NÓI TIẾP CHO 8051

3.4.6.1. Cơ sở truyền thông nối tiếp

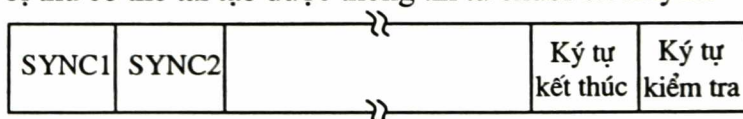
Khi các thiết bị nằm gần nhau trong hệ thống vi xử lý, việc truyền thông có thể thực hiện thông qua Bus song song mở rộng hoặc qua các mạch phối ghép song song, trong đó các byte hoặc từ được chuyển từ bộ phận này sang bộ phận khác trên một tập đường dẫn

bằng mạch in hoặc dây cáp song song. Ưu điểm của phương pháp truyền tin song song là có thể truyền được một khối lượng lớn tin tức trong thời gian ngắn, nên tốc độ truyền rất nhanh, tuy nhiên cũng có nhược điểm là tổn dây dẫn, hệ thống cồng kềnh, và nếu truyền xa sẽ gây méo tín hiệu.

Trong trường hợp phải truyền thông tin giữa các thiết bị ở xa nhau, ta không thể dùng cách truyền như trên mà phải dùng cách khác để tiết kiệm được dây, vừa không làm méo tín hiệu khi truyền. Từ yêu cầu đó ra đời phương pháp truyền thông tin nối tiếp: ở đầu phát dữ liệu dưới dạng song song đầu tiên được chuyển thành dạng nối tiếp, tín hiệu nối tiếp sau đó được truyền đi liên tiếp từng bit trên một đường dây (ví dụ như đường dây điện thoại). Ở đầu thu tín hiệu nối tiếp sẽ được biến đổi ngược lại để tái tạo tín hiệu dạng song song thích hợp cho việc xử lý tiếp theo.

Trong thực tế có hai phương pháp truyền thông tin kiểu nối tiếp: Truyền đồng bộ và truyền không đồng bộ (dị bộ).

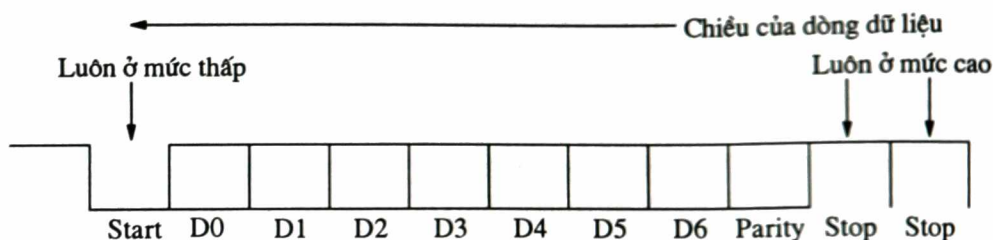
– Truyền đồng bộ: Dữ liệu được truyền theo từng mảng (các ký tự) với một tốc độ xác định. Mảng ký tự trước khi được truyền đi sẽ được gắn thêm ở đầu mảng và cuối mảng các byte (hoặc một nhóm bit) đánh dấu đặc biệt gọi là ký tự đồng bộ (SYNC). Nhờ những ký tự SYNC này mà thiết bị thu có thể tái tạo được thông tin từ chuỗi bit truyền.



Hình 3.14. Khuôn dạng truyền tin nối tiếp đồng bộ

– Truyền tin không đồng bộ (truyền tin dị bộ): Dữ liệu được truyền đi theo từng ký tự. Ký tự được truyền đi bao giờ cũng bắt đầu bằng bit Start (luôn ở mức thấp) và một hoặc hai bit Stop ở cuối để báo kết thúc. Tùy theo hệ thống truyền tin, bên cạnh các bit mã dữ liệu còn có thể tùy chọn có hay không một bit Parity để kiểm tra lỗi khi

truyền. Phương pháp truyền này tuy đơn giản nhưng có hiệu suất không cao.



Hình 3.15. Khuôn dạng truyền tin nối tiếp dị bộ

Tốc độ truyền tin nối tiếp được đo bằng bit/s. Ngoài ra người ta cũng hay dùng đơn vị Baud. Các giá trị tốc độ truyền thường gặp trong thực tế là 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200 baud.

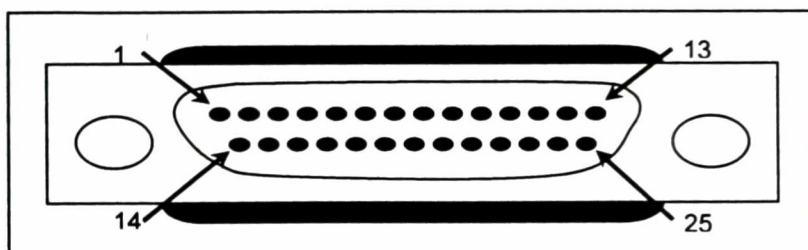
Để tạo điều kiện dễ dàng cho việc phối ghép đường truyền nối tiếp người ta thường sử dụng các vi mạch chuyên dụng phục vụ cho việc truyền dữ liệu nối tiếp, những IC này dùng làm các bộ thu phát dị bộ tổng hợp UART và các bộ thu phát đồng – dị bộ tổng hợp USART. Bộ vi điều khiển 8051 được xây dựng sẵn một bộ UART và thực hiện truyền thông nối tiếp thông qua 2 chân trên cổng P3 đó là RxD (P3.0), TxD (P3.1).

3.4.6.2. Chuẩn RS232

Để cho phép tương thích giữa các thiết bị truyền thông dữ liệu được sản xuất bởi các hãng khác nhau thì một chuẩn giao diện được gọi là RS232 đã được thiết lập. Chuẩn này được sử dụng trong máy tính PC và hàng loạt các thiết bị khác nhau. Tuy nhiên, vì nó được thiết lập trước họ logic TTL rất lâu, do vậy điện áp đầu vào và đầu ra của nó không tương thích với mức TTL. Trong RS232 thì mức 1 được biểu diễn bởi -3V đến -25V trong khi đó mức 0 thì ứng với điện áp $+3\text{V}$ đến $+25\text{V}$ làm cho điện áp -3V đến $+3\text{V}$ là không xác định. Vì lý do này để kết nối một RS232 bất kỳ đến một hệ vi điều khiển ta phải sử dụng các bộ biến đổi điện áp như MAX232 để chuyển đổi các mức logic TTL về mức điện áp RS232 và ngược lại. Các chip IC MAX232

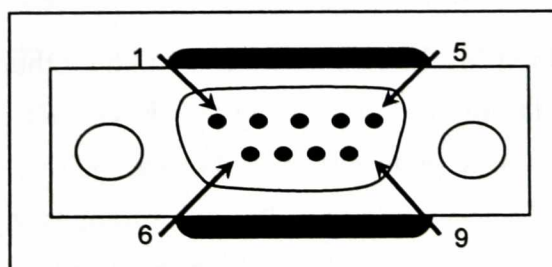
nhìn chung được coi như các bộ điều khiển đường truyền. Kết nối RS232 đến MAX232 được trình bày ngay ở phần sau.

Sơ đồ chân của cáp RS232 và các tên gọi của chúng thường được gọi là đầu nối DB - 25. Trong ký hiệu thì đầu nối cắm vào (đầu đực) gọi là DB - 25p và đầu nối cái được gọi là DB - 25s (hình 3.16).

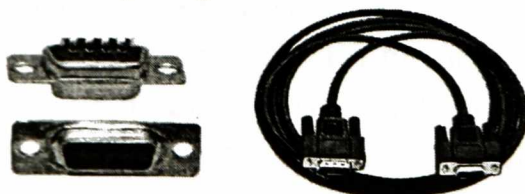
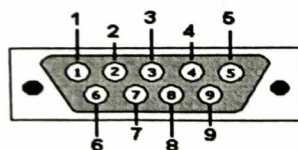


Hình 3.16. Đầu nối DB - 25 của RS232

Vì không phải tất cả mọi chân đều được sử dụng trong cáp của máy tính PC, nên IBM đưa ra phiên bản của chuẩn vào/ra nối tiếp chỉ sử dụng 9 chân gọi là DB - 9 như sau:



RS232 DB9 (EIA/TIA 574)



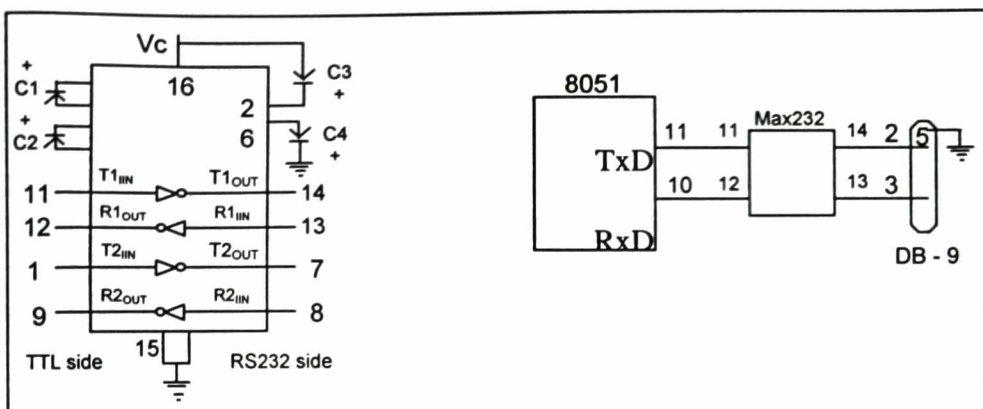
Hình 3.17. Đầu nối DB - 9 và cáp RS232

Bảng 3.10

Mô tả	Mô tả chân DB – 9	
1	Data carrier detect (DCD)	Tách tín hiệu mang dữ liệu
2	Received data (RxD)	Dữ liệu được nhận
3	Transmitted data (TxD)	Dữ liệu được gửi
4	Data terminal ready (DTR)	Đầu dữ liệu sẵn sàng
5	Signal ground (GND)	Đất của tín hiệu
6	Data set ready (DSR)	Dữ liệu sẵn sàng
7	Request to send (RTS)	Yêu cầu gửi
8	Clear to send (CTS)	Xoá để gửi
9	Ring indicator (RL)	Báo chuông

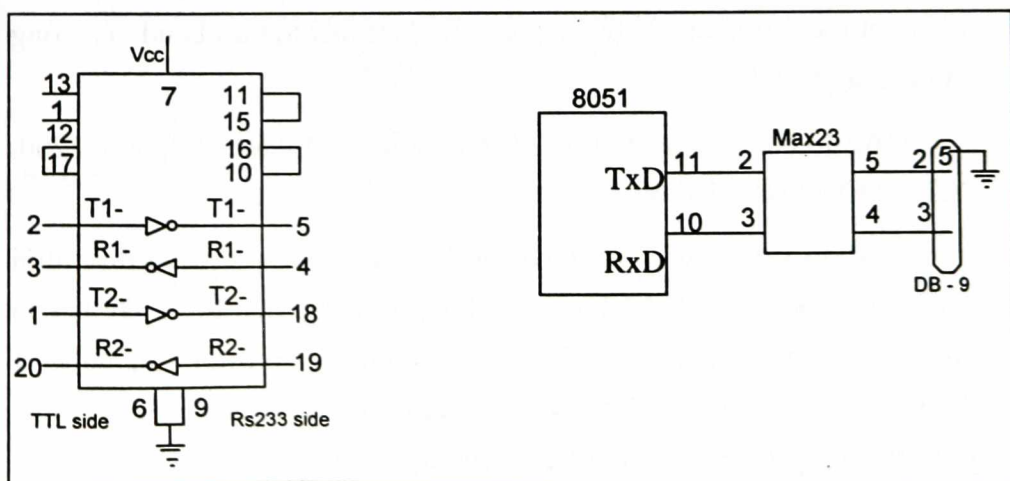
3.4.6.3. Ghép nối 8051 với RS232

Bộ điều khiển MAX232 có hai bộ điều khiển thường để nhận và truyền dữ liệu. Trong nhiều ứng dụng thì chỉ có một cặp được dùng. Ví dụ: T1 và R1 được dùng với nhau đối với TxD và RxD của 8051, còn cặp R2 và T2 thì chưa dùng đến. Để ý rằng trong MAX232 bộ điều khiển T1 có gán T1in và T1out trên các chân số 11 và 14 tương ứng. Chân T1in là ở phía TTL và được nối tới chân TxD của bộ vi điều khiển, còn T1out là ở phía RS232 được nối tới chân RxD của đầu nối DB của RS232. Bộ điều khiển đường R1 cũng có gán R1in và R1out trên các chân số 13 và 12 tương ứng. Chân R1in (chân số 13) là ở phía RS232 được nối tới chân TxD của đầu nối DB của RS232 và chân R1out (chân số 12) là ở phía TTL mà nó được nối tới chân RxD của bộ vi điều khiển.

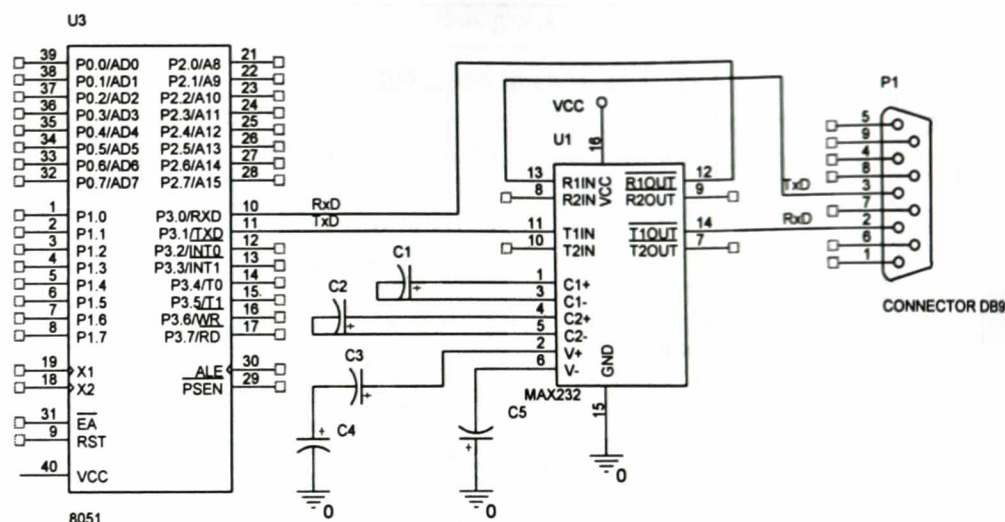


Hình 3.18

Để tiết kiệm không gian trên bảng mạch, nhiều nhà thiết kế sử dụng chip MAX233 từ hãng Maxim. Bộ điều khiển MAX233 thực hiện cùng những công việc như MAX232 lại không cần đến các tụ điện. Tuy nhiên, chip MAX233 lại đắt hơn rất nhiều so với MAX232, không có sơ đồ chân giống nhau (không tương thích). Chúng ta không thể lấy một chip MAX232 ra khỏi một bảng mạch và thay vào đó RS233.



Hình 3.19. Sơ đồ ghép nối vi điều khiển 8051 chip MAX 232 và ổ cắm DB-9



Hình 3.20. Sơ đồ ghép nối vi điều khiển 8051 chip MAX 232 và ổ cắm DB-9

3.4.6.4. Lập trình truyền thông nối tiếp 8051

❖ Đặt tốc độ truyền

Để thực hiện truyền dữ liệu không có lỗi giữa máy tính PC và 8051 thì tốc độ baud của 8051 phải phù hợp với tốc độ baud của cổng COM máy tính PC.

Tốc độ baud của máy tính PC có thể là: 100, 150, 300, 600, 1200, 2400, 4800, 9600, 19200.

Tốc độ truyền của 8051 có thể lập trình được nhờ bộ định thời Timer 1. Mạch điện UART truyền thông nối tiếp của 8051 lấy tần số đồng hồ của 8051 chia cho 32; sau đó mới dùng bộ định thời Timer 1 để tạo ra tốc độ baud. Muốn Timer 1 làm việc đặt tốc độ baud thì nó phải được lập trình về chế độ 2 là chế độ 8 bit tự động nạp lại. Để có được tốc độ baud tương thích với PC ta phải nạp TH1 theo các giá trị cho trong bảng 3.11:

Bảng 3.11

Tốc độ baud	TH1 (thập phân)	TH1 (số Hex)
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	E8

Ví dụ: Xác định giá trị cần nạp vào TH1 để có tốc độ baud là 9600, 2400, 1200. Nếu biết tần số của XTAL là 11,0592MHz.

Với tần số XTAL = 11,0592MHz = 11059,2KHz thì tần số đồng hồ của 8051 là $11059,2/12 = 921,6\text{KHz}$, lấy $921,6\text{KHz}/32 = 28,8\text{KHz} = 28800\text{Hz}$ là tần số UART cấp tới bộ định thời Timer 1 để thiết lập tốc độ.

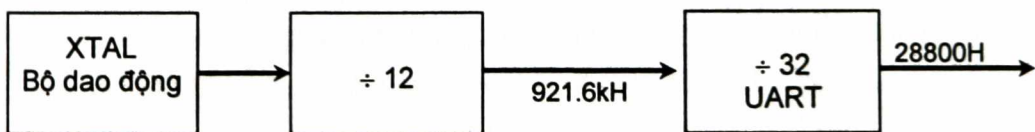
Việc chia 1/12 tần số thạch anh cho 32 là giá trị mặc định khi kích hoạt chân RESET của 8051.

Như vậy:

$28000/3 = 9600$ Trong đó -3 = FD được nạp vào TH1

$28000/12 = 2400$ Trong đó -12 = F4 được nạp vào TH1

$28000/24 = 1200$ Trong đó -24 = E8 được nạp vào TH1



Các thanh ghi phục vụ cho truyền thông nối tiếp

Thanh ghi SBUF

SBUF là thanh ghi 8 bit được dùng cho truyền thông nối tiếp của 8051. Để byte dữ liệu được truyền qua đường TxD thì cần đặt dữ liệu trong thanh ghi SBUF và byte đó sẽ được định khung với bit start và stop. Ngược lại, SBUF sẽ lưu một byte dữ liệu khi nó được truyền qua

đường RxD (trước đó 8051 mở khung, tức là loại trừ các bit start và stop để lấy ra một byte dữ liệu nhận được và đặt vào thanh ghi SBUF).

Thanh ghi SCON: Đây là thanh ghi được dùng cho một số công việc trong đó có lập trình bit khởi động start, bit dừng stop và các bit dữ liệu khi định khung dữ liệu.

SCON.7	SCON.6	SCON.5	SCON.4	SCON.3	SCON.2	SCON.1	SCON.0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

– SM0, SM1: Xác định chế độ khung dữ liệu bằng cách xác định số bit của một ký tự và các bit start, stop.

Bảng 3.12

SM0	SM1	Chế độ
0	0	Chế độ nối tiếp 0
0	1	Chế độ nối tiếp 1, 8 bit dữ liệu, start
1	0	Chế độ nối tiếp 2
1	1	Chế độ nối tiếp 3

Trong các chế độ trên ta chỉ quan tâm đến chế độ nối tiếp 1. Ở chế độ này mỗi ký tự gồm 10 bit được truyền, trong đó bit đầu tiên là start, sau đó là 8 bit dữ liệu và cuối cùng là bit stop.

- SM2: Dùng trong hệ đa xử lý của 8051. Nếu không sử dụng 8051 trong các hệ đa xử lý thì đặt SM2 = 0.
- REN: Cho phép thu (Receiver Enable). Khi ở mức cao nó cho phép 8051 thu dữ liệu trên chân RxD. Như vậy, muốn 8051 vừa phát vừa thu dữ liệu thì REN cần đặt lên 1. Nếu REN = 0 thì bộ thu bị khoá. Có thể dùng lệnh “SETB SCON.4” và “CLR SCON.4” để đặt REN = 1 hoặc bằng 0.
- TB8 (Transfer Bit 8): Được dùng cho chế độ nối tiếp 2 và 3. Nếu ở chế độ 1 thì đặt TB8 = 0.

- RB8 (Receiver Bit 8): Ở chế độ 1 bit này nhận một bản sao của bit stop khi một dữ liệu 8 bit được nhận. Bit này cũng hiếm khi được sử dụng, vì vậy đặt nó bằng 0 vì không sử dụng chế độ nối tiếp 2 và 3.
- TI (Transmit Interrupt) và RI (Receiver Interrupt): Khi kết thúc phát một ký tự 8 bit thì cờ TI bật lên để báo kết thúc truyền và sẵn sàng phát byte tiếp theo. Bit TI được bật khi bắt đầu bit Stop.

Khi 8051 nhận được dữ liệu nối tiếp qua RxD thì tiến hành tách bit start và stop để lấy ra 8 bit dữ liệu đặt vào thanh ghi SBUF. Sau khi quá trình này kết thúc cờ RI bật lên để báo bộ vi điều khiển đã nhận xong một byte và cần được cất đi nếu không sẽ bị mất. Cờ RI được bật khi đang tách bit stop.

❖ Lập trình truyền dữ liệu nối tiếp

Trình tự các bước khi lập trình 8051 truyền các byte ký tự nối tiếp như sau:

- B1: Nạp giá trị 20H vào thanh ghi TMOD báo sử dụng Timer 1 ở chế độ 2 để thiết lập chế độ baud.
- B2: Nạp giá trị cho trong bảng 3.11 vào thanh ghi TH1 để thiết lập chế độ baud truyền dữ liệu nối tiếp (nếu XTAL = 11,0592MHz).
- B3: Nạp giá trị 50H vào thanh ghi SCON để báo chế độ nối tiếp 1 để định khung 8 bit dữ liệu, 1 bit start, 1 bit stop.
- B4: Bật TR1 = 1 khởi động Timer 1.
- B5: Xóa bit TI bằng lệnh "CLR TI".
- B6: Ghi byte ký tự cần truyền vào SBUF.
- B7: Kiểm tra cờ TI bằng lệnh "JNB TI, xx" để báo hoàn tất việc truyền ký tự.

– B8: Trở về bước 5 để truyền ký tự tiếp theo.

Ví dụ: Viết chương trình truyền nối tiếp liên tục chữ “HELLO” với tốc độ truyền 9600 baud.

```
MOV      TMOD, #20H      ;Timer 1 che do 2
MOV      TH1, #-3        ;toc do 9600 baud
MOV      SCON, #50H      ;truyen 8 bit du
                        ;lieu, 1 bit stop va cho phep thu
SETB     TR1              ;khoi dong Timer 1
AGAIN:   MOV      A, #"H"      ;truyen ky tu "H"
ACALL    TRUYEN
MOV      A, #"E"          ;truyen ky tu "E"
ACALL    TRUYEN
MOV      A, #"L"          ;truyen ky tu "L"
ACALL    TRUYEN
MOV      A, #"L"          ;truyen ky tu "L"
ACALL    TRUYEN
MOV      A, #"O"          ;truyen ky tu "O"
ACALL    TRUYEN
SJMP     AGAIN            ;lap lai

;-----chuong trinh con truyen du lieu noi tiep
TRUYEN:  MOV      SBUF, A      ;nap SBUF
HERE:    JNB      TI, HERE     ;kiem tra co TI
CLR      TI                ;xoa co
RET                               ;tro ve chuong
                                trinh chinh
```


❖ Lập trình nhận dữ liệu nối tiếp

Trình tự các bước khi lập trình 8051 nhận các byte ký tự nối tiếp như sau:

- B1: Nạp giá trị 20H vào thanh ghi TMOD báo sử dụng Timer 1 ở chế độ 2 để thiết lập chế độ baud.
- B2: Nạp giá trị cho trong bảng 3.11 vào thanh ghi TH1 để thiết lập chế độ baud truyền dữ liệu nối tiếp (nếu XTAL = 11,0592MHz).
- B3: Nạp giá trị 50H vào thanh ghi SCON để báo chế độ nối tiếp 1 để định khung 8 bit dữ liệu, 1 bit start, 1 bit stop.
- B4: Bật TR1 = 1 khởi động Timer 1.
- B5: Xoá bit RI bằng lệnh “CLR RI”.
- B6: Bit cờ RI được hiển thị bằng lệnh “JNB RI, xx” để xem toàn bộ ký tự đã được nhận chưa.
- B7: Khi RI được thiết lập thì trong SBUF đã có một 1 byte. Các nội dung của nó được cất giữ vào một nơi an toàn.
- B8: Trở về bước 5 để nhận ký tự tiếp theo.

Ví dụ: Hãy lập trình cho 8051 nhận các byte dữ liệu nối tiếp và đặt chúng vào cổng P0. Tốc độ baud là 9600, 8 bit dữ liệu và 1 bit stop.

Giải:

```
MOV    TMOD, #20H    ;chon Timer 1 che do 2
MOV    TH1, #-3      ;chon toc do 9600 baud
MOV    SCON, #50H    ;khung du lieu 8
                        ;bit, 1 stop
SETB    TR1          ;khoi dong Timer
HERE:  JNB RI, HERE   ;kiem tra RI
```

```

MOV    A, SBUF    ;lưu ký tự vào thanh ghi A
MOV    P0, A      ;gửi ra cổng P0
CLR    RI         ;xóa cờ RI
SJMP   HERE       ;tiếp tục nhận dữ liệu

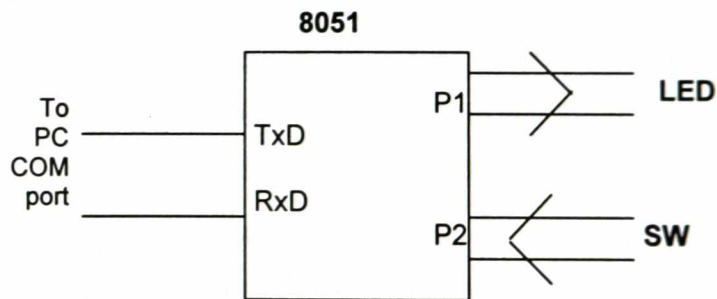
```

Ví dụ tổng quát:

Giả sử cổng nối tiếp của 8051 được nối vào cổng COM của máy tính PC. 8051 có cổng P1 được nối với các LED, P2 nối với các công tắc hành trình. Hãy viết chương trình để 8051.

- Nhận bất kỳ dữ liệu gì được PC gửi đến và chuyển đến đèn LED đang được nối tới các chân của P1 (chạy liên tục với tốc độ 4800 baud).
- Nhận dữ liệu trên P2 và gửi tới máy tính PC (chạy liên tục với tốc độ 4800 baud).

Giải:



Hình 3.21

```

ORG    0H
MOV    P2, #0FFH    ;lấy cổng P2 làm cổng vào
MOV    TMOD, #20H    ;chọn Timer 1 chế độ 2
MOV    TH1, #0FAH    ;chọn tốc độ 4800 baud
MOV    SCON, #50H    ;tạo khung 8 bit dữ liệu, 1 stop
SETB   TR1           ;khởi động Timer 1

```

```

LAP:MOV  A, P2          ;doc du lieu tu cong P2
      ACALL  GUI         ;truyen noi tiep
      ACALL  NHAN        ;nhan du lieu noi tiep
      MOV   P1, A        ;hien thi ra den LED
      SJMP  LAP          ;quay lai vong lap
;-----chuong trinh con truyen du lieu noi tiep
GUI: MOV  SBUF, A        ;truyen du lieu
HERE:JNB TI, HERE ;kiem tra TI
      CLR  TI            ;xoa TI
      RET              ;ve chuong trinh chinh
;-----chuong trinh con nhan du lieu noi tiep
NHAN:JNB RI, NHAN      ;kiem tra RI
      MOV  A, SBUF       ;luu ket qua vao thanh ghi A
      CLR  RI            ;xoa co RI
      RET              ;ve chuong trinh chinh
END

```

3.4.6.5. Nhân đôi tốc độ baud trong 8051

Có hai cách để tăng tốc độ baud truyền dữ liệu trong 8051.

- Sử dụng tần số thạch anh cao hơn.
- Thay đổi một bit trong thanh ghi điều khiển công suất PCON (Power Control) như chỉ ra dưới đây.

D7							D0
SM0D	-	-	-	GF0	GF1	PD	IDL

Phương án một là không thực thi trong nhiều trường hợp vì tần số thạch anh của hệ thống là cố định. Quan trọng hơn là nó không khả

thì, vì tần số thạch anh mới không tương thích với tốc độ baud của các cổng COM nối tiếp của IBM PC. Do vậy, ta sẽ tập trung vào phương án hai.

Khi 8051 được bật nguồn thì bit SMOD của thanh ghi PCON ở mức thấp 0. Chúng ta có thể đặt nó lên 1 bằng phần mềm và do vậy nhân đôi được tốc độ baud. Thứ tự các lệnh được sử dụng để thiết lập bit D7 của PCON lên cao như sau (thanh ghi PCON là thẻ đánh địa chỉ theo bit).

```
MOV A, PCON ;Đặt bản sao của PCON vào ACC
SETB ACC.7 ;Đặt D7 của ACC lên 1.
MOV PCON, A ;SMOD = 1
```

Bảng 3.13. So sánh tốc độ baud khi SMOD thay đổi

TH1 (thập phân)	TH1 (Hex)	Tốc độ baud	
		SMOD = 0	SMOD = 1
-3	FD	9600	19200
-6	FA	4800	9600
-12	F4	2400	4800
-24	E8	1200	2400

Ví dụ: giả sử tần số XTAL = 11.0592MHz cho chương trình dưới đây, hãy phát biểu:

- Chương trình này làm gì?
- Hãy tính toán tần số được Timer1 sử dụng để đặt tốc độ baud?
- Hãy tìm tốc độ baud truyền dữ liệu.

```
MOV A, PCON ;sao noi dung thanh ghi PCON vao A
SETB ACC.7 ;dat D7 = 0
MOV PCON, A ;dat SMOD = 1
```

```

MOV  TMOD, #20H          ;chon Timer 1 che do 2
MOV  TH1, #-3             ;chon toc do 19200
                               ; (57600/3=19200) vi SMOD=1
MOV  SCON, #50H          ;che do truyen 1
SETB TR1                  ;khởi động Timer1
MOV  A, #"B"              ;truyen ky tu "B"
AGAIN:
    CLR    TI              ;khang dinh TI = 0
    MOV    SBUF, A         ;truyen di
HERE: JNB    TI, HERE      ;doi den khi TI bat len
    SJMP   AGAIN           ;tiếp tục gửi "B"

```

Lời giải:

- a) Chương trình này truyền liên tục mã ASCII của chữ B (ở dạng nhị phân là 0100 0010).
- b) Với tần số XTAL = 11.0592MHz và SMOD = 1 trong chương trình trên ta có:
 $11.0592\text{MHz}/12 = 921.6\text{kHz}$ là tần số chu trình máy
 $921.6\text{kHz}/16 = 57.6\text{kHz}$ là tần số được Timer1 sử dụng để đặt tốc độ baud
- c) $57.6\text{kHz}/3 = 19.200$ là tốc độ cần tìm.

3.4.7. LẬP TRÌNH NGẮT 8051

3.4.7.1. Tổng quan

Một bộ vi điều khiển có thể phục vụ một số thiết bị, mỗi khi có một thiết bị cần được phục vụ thì thiết bị báo cho bộ vi điều khiển bằng cách gửi một tín hiệu ngắt. Khi nhận được tín hiệu này thì bộ vi

điều khiển sẽ ngừng mọi công việc đang thực hiện để chuyển sang phục vụ thiết bị. Chương trình đi cùng với ngắt được gọi là trình phục vụ ngắt ISR (Interrupt Service Routine) hay còn gọi là bộ quản ngắt (Interrupt handler). Trình phục vụ ngắt của mỗi ngắt có một vị trí cố định trong bộ nhớ để giữ địa chỉ ISR. Tập hợp các ô nhớ lưu giữ địa chỉ của tất cả các ISR được gọi là bảng vector ngắt (bảng 3.14).

Từ bảng 3.14 ta thấy: Chỉ có 3 byte không gian bộ nhớ ROM được giành cho RESET. Đó là địa chỉ 0, 1 và 2. Địa chỉ 3 thuộc về ngắt phần cứng ngoài - ngắt 0. Vì lý do đó, trong chương trình cần đặt lệnh LJMP làm lệnh đầu tiên để hướng bộ vi xử lý ra ngoài bảng vector ngắt.

```
ORG      0           ;địa chỉ bắt đầu khi khởi động
        LJMP MAIN    ;bỏ qua bảng vector ngắt

;--- Bắt đầu chương trình chính
        ORG      30H

MAIN:
        ...

END
```

Bảng 3.14. Bảng vector ngắt của 8051

Ngắt	Địa chỉ ROM (Hexa)	Chân
RESET	0000	9
Ngắt phần cứng ngoài (INT0)	0003	12 (P3.2)
Ngắt bộ Timer 0 (TF0)	000B	
Ngắt phần cứng ngoài 1 (INT1)	0013	13 (P3.3)
Ngắt bộ Timer 1 (TF1)	001B	
Ngắt COM nối tiếp (RI và TI)	0023	

3.4.7.2. Trình tự thực hiện ngắt

Khi một ngắt được kích hoạt trình tự thực hiện của bộ vi điều khiển như sau:

1. Kết thúc lệnh hiện tại và lưu địa chỉ của lệnh kế tiếp (PC) vào ngăn xếp.
2. Lưu lại trạng thái hiện hành của tất cả các ngắt vào bên trong (nghĩa là không lưu vào ngăn xếp).
3. Nhảy đến một vị trí cố định trong bộ nhớ được gọi là bảng vector ngắt, nơi lưu giữ địa chỉ của trình phục vụ ngắt.
4. Nhận địa chỉ ISR từ bảng vector ngắt rồi nhảy tới địa chỉ đó và bắt đầu thực hiện trình phục vụ ngắt cho đến khi gặp lệnh RETI (trở về từ ngắt).
5. Kết thúc trình phục vụ ngắt, bộ vi điều khiển gặp lệnh RETI và trở về nơi nó đã bị ngắt. Trước hết, hai byte của đỉnh ngăn xếp được nạp vào bộ đếm PC, tiếp theo, bộ vi điều khiển bắt đầu thực hiện lệnh tại địa chỉ đó.

3.4.7.3. Sáu ngắt của 8051 (bảng 3.14)

Thực tế 8051 chỉ có 5 ngắt dành cho người dùng, song nhiều tài liệu kỹ thuật của các nhà sản xuất vẫn nói rằng có 6 ngắt vì họ tính cả lệnh RESET. Bố trí 6 ngắt của 8051 như sau:

- RESET: Khi chân RESET được kích hoạt thì 8051 nhảy về địa chỉ 0000. Đây là địa chỉ bật lại nguồn.
- Hai ngắt dành cho bộ định thời Timer 0 và Timer 1. Địa chỉ ở bảng vector ngắt của hai ngắt này tương ứng với Timer 0 và Timer 1 là 000BH và 001BH.
- Hai ngắt phần cứng dành cho các thiết bị bên ngoài nối tới chân 12 (P3.2) và 13 (P3.3) của cổng P3 là INT0 và INT1

tương ứng. Vị trí nhớ trong bảng vector ngắt này là 0003H và 0013H.

- Truyền thông nối tiếp có một ngắt cho cả thu lẫn phát. Địa chỉ của ngắt này trong bảng vector ngắt là 0023H.

3.4.7.4. Cho phép ngắt và cấm ngắt

Khi RESET tất cả mọi ngắt đều bị cấm (bị che), có nghĩa là không có ngắt nào được bộ vi điều khiển đáp ứng nếu chúng được kích hoạt. Các ngắt phải được cho phép bằng phần mềm để bộ vi điều khiển có thể đáp ứng được. Có một thanh ghi được gọi là thanh ghi cho phép ngắt IE (Interrupt Enable) chịu trách nhiệm về việc cho phép (không che) hoặc cấm (che) các ngắt. IE là một thanh ghi có thể định địa chỉ bit.

IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0
EA	---	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	Nếu EA = 0 thì không ngắt nào được báo nhận. Nếu EA = 1 thì từng nguồn ngắt sẽ được mở hoặc cấm bằng cách bật hoặc xóa bit tương ứng.
---	IE.6	Dự phòng cho tương lai.
ET2	IE.5	Cho phép hoặc cấm ngắt tràn hoặc thu của Timer 2 (AT89C52).
ES	IE.4	Cho phép hoặc cấm ngắt tràn công nối tiếp.
ET1	IE.3	Cho phép hoặc cấm ngắt tràn của Timer 1.
EX1	IE.2	Cho phép hoặc cấm ngắt ngoài 1.
ET0	IE.1	Cho phép hoặc cấm ngắt tràn của Timer 0.
EX0	IE.0	Cho phép hoặc cấm ngắt ngoài 0.

Ví dụ: Hãy viết lệnh thực hiện nhiệm vụ sau

– Cho phép ngắt nối tiếp, ngắt Timer 1 và ngắt phần cứng ngoài 0 (EX0).

– Cấm ngắt Timer 0

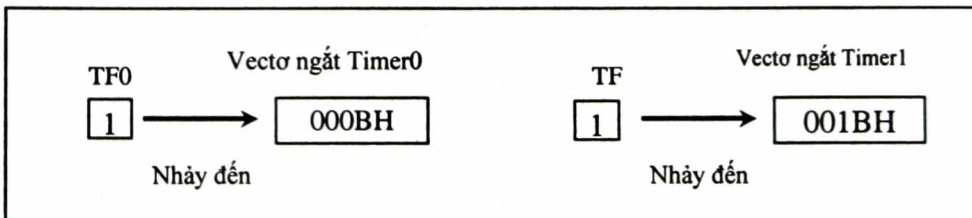
– Cấm tất cả mọi ngắt chỉ bằng một lệnh duy nhất.

Giải:

```
MOV IE, #10010011B ;cho phép ngắt nối tiếp,  
;Timer 0 và cho phép ngắt phần cứng ngoài  
CLR IE.1;xóa (che) ngắt Timer 0  
CLR IE.7;Cấm tất cả mọi ngắt.
```

3.4.7.5. Lập trình ngắt bộ định thời 8051

Nếu bit ngắt bộ định thời ở thanh ghi IE được phép thì mỗi khi bộ định thời này quay về 0, cờ TF được bật lên và bộ vi điều khiển ngừng mọi việc đang thực hiện và nhảy tới bảng vectơ ngắt để phục vụ ISR. Bằng cách đó bộ vi điều khiển có thể làm những công việc khác nhau cho đến khi nào được thông báo rằng bộ định thời đã quay về 0 (hình 3.22).



Hình 3.22. Các ngắt bộ định thời Timer 0 và Timer 1

Ví dụ: Viết chương trình nhận liên tục dữ liệu 8 bit ở cổng P0 và gửi đến cổng P1. Trong thời gian này cần tạo ra trên chân P2.1 một sóng vuông tần số 50Hz. Sử dụng bộ Timer 0 để tạo sóng vuông, tần số thạch anh của 8051 XTAL = 11, 0592MHz.

Giải:

Giả sử ta sử dụng Timer 0 chế độ 1, trước hết cần tính giá trị cần nạp cho thanh ghi TH0 và TL0.

- Chu kỳ sóng vuông: $T = 1/f = 1/50 = 0,02s = 20ms$
- Thời gian nửa chu kỳ phần cao: $0.5T = 0,01s = 10ms = 10000\mu s$
- Số nhịp đồng hồ: $n = 10000/1,085 = 9216$
- Giá trị cần nạp vào TH0 và TL0 là: $65536 - 9216 = 56320 = DC00H$.

Chương trình được viết như sau:

```
ORG 0000H

LJMP MAIN      ;bo qua bang vec to ngat

;--- Trình ISR cho Timer0 de tao xung vuong
ORG 000BH      ;dia chi ngat Timer 0
LJMP ISR_T0     ;nhay den ISR

;--- Chuong trinh chinh
ORG 0030H      ;dia chi chuong trinh chinh
MAIN:  MOV      TMOD, #01H;chon Timer 0 che do 1
        MOV      P0, #0FFH;chon P0 lam cong vao
        MOV      TL0, #00H          ;dat TL0=00H
        MOV      TH0, #0DCH         ;dat TH0=DCH
        MOV      IE, #82H;cho phep ngat Timer 0
        SETB     TR0                ;khoi dong Timer 0
BACK:  MOV      A, P0                ;doc cong P0
        MOV      P1, A              ;chuyen ra cong P1
        SJMP     BACK               ;tiếp tục đọc
```



```

;--Trình ISR của Timer 0 để do 1 phải được nạp lại
ISR_T0: CPL      P2.1

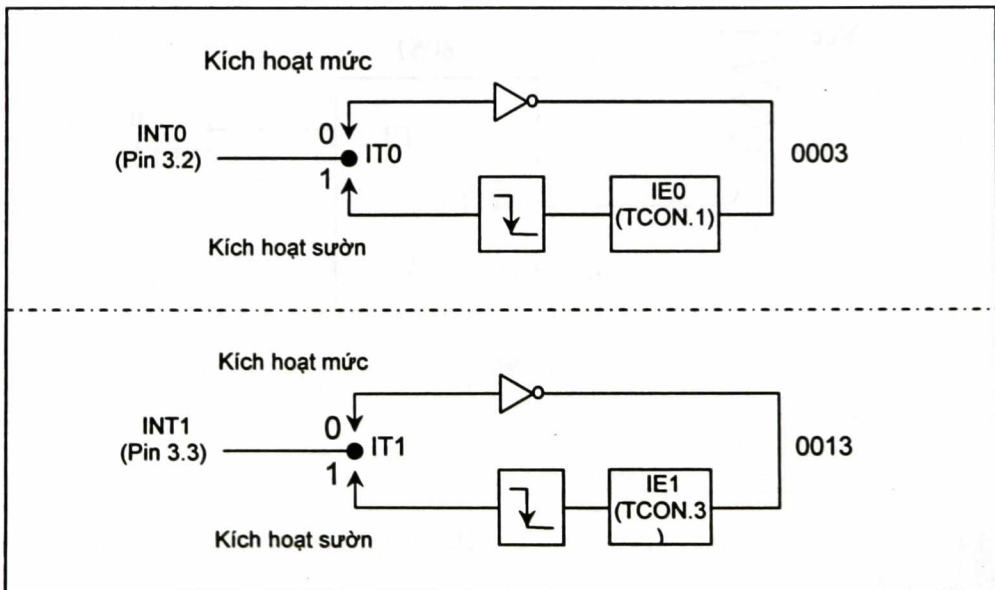
        MOV      TL0, #00H          ; nạp lại TL0
        MOV      TH0, #0DCH        ; nạp lại TH0

        RETI                      ; về chương trình chính
END

```

3.4.7.6. Lập trình ngắt phần cứng ngoài

Bộ vi điều khiển 8051 có hai ngắt phần cứng bên ngoài là chân 12 (P3.2) và chân 13 (P3.3) dùng cho ngắt INT0 và INT1 địa chỉ trong bảng vector ngắt là 0003H và 0013H. Khi kích hoạt những chân này thì vi điều khiển 8051 ngừng ngay mọi công việc đang thực hiện và nhảy đến bảng vector ngắt để chạy trình phục vụ ngắt (hình 3.23).



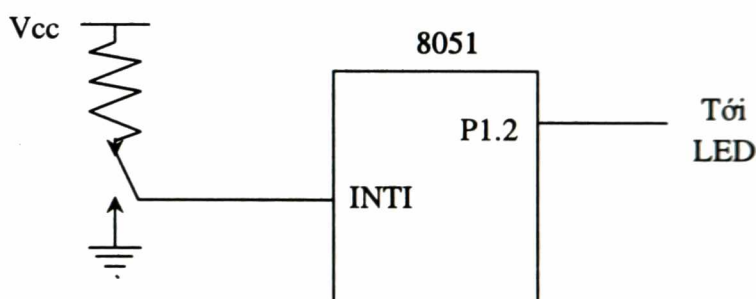
Hình 3.23. Kích hoạt INT0 và INT1

Có hai cách kích hoạt ngắt phần cứng ngoài, đó là theo mức và theo sườn.

❖ Ngắt kích phát mức

Ở chế độ ngắt theo mức, các chân INT0 và INT1 bình thường ở mức cao. Nếu có tín hiệu mức thấp (khoảng 4 chu kỳ máy) cấp tới thì tín hiệu này kích hoạt ngắt. Trước khi thực hiện lệnh cuối cùng của trình phục vụ ngắt RETI, thì mức thấp tại chân INT phải chuyển sang cao, nếu không sẽ tạo ra một ngắt mới và nhảy đến bảng vector ngắt để thực hiện ISR.

Ví dụ: Giả sử INT1 được nối tới một công tắc hành trình, bình thường ở mức cao (nối với tiếp điểm thường đóng). Mỗi khi chân này xuống mức thấp thì bật đèn LED. Đèn LED được nối tới chân P1.2 và bình thường ở chế độ tắt. Nếu đèn LED được bật sáng thì phải sáng trong vài giây. Khi công tắc được chuyển xuống thấp, đèn LED phải sáng liên tục.



Hình 3.24

Giải:

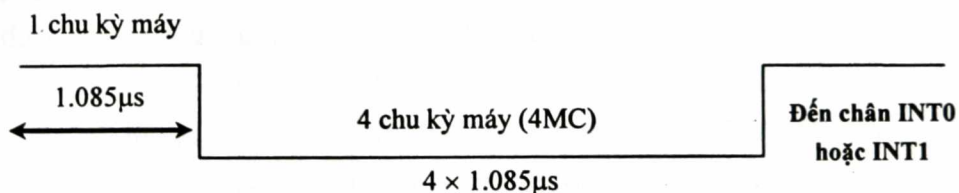
```
ORG      0000H
LJMP     MAIN      ;bo qua bang vector ngat
;--- chương trình con xử lý ngắt
ORG      0013H      ;trình ISR cho INT1
LJMP     ISR_INT1    ;den trình phục vụ ngắt
;--- chương trình chính
```

```

                ORG      30H
MAIN:          MOV      IE, #84H;cho phép ngắt ngoài INT1
HERE:          SJMP     HERE;doi cho den khi duoc ngắt
ISR_INT1:
                SETB     P1.2      ;bat den LED
                MOV      R3, #0FFH
BACK:          DJNZ     R3, BACK;giu den LED sang mot luc
                CLR      P1.2      ;tat den LED
                RETI          ;ve chuong trinh chinh
                END

```

Chúng ta cũng cần chú ý rằng khi bật lại nguồn (RESET) thì cả hai chân INT0 và INT1 đều ở mức cao.



**Hình 3.25. Thời gian tối thiểu của ngắt kích phát mức với
XTAL = 11, 0592MHz**

❖ Ngắt kích phát sườn

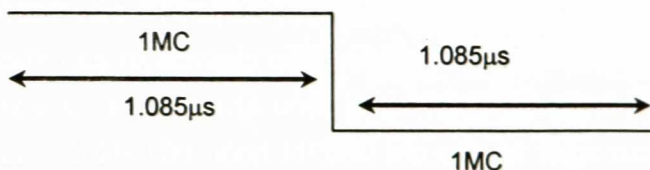
Như đã nói ở trên khi RESET thì 8051 đặt ngắt INT0 và INT1 ở chế độ kích phát mức thấp. Để đổi các ngắt thành kích phát sườn thì cần phải viết chương trình ghi 1 tới IT1 hoặc IT0 của thanh ghi TCON (hình 3.26). Khi đó nếu có một tín hiệu chuyển từ cao xuống thấp cấp cho chân P3.2 hoặc P3.3 thì bộ vi điều khiển sẽ bị ngắt và buộc nhảy đến bảng vector ngắt tại địa chỉ 0003H hoặc 0013H để thực hiện trình phục vụ ngắt. Tuy nhiên, phải giả thiết rằng bit ngắt tương ứng trên thanh ghi IE đã được đặt.

D7							D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Hình 3.26. Thanh ghi TCON (Timer/Counter)

- TF1 TCON.7 Cờ tràn của Timer 1.
- TR1 TCON.6 Bit điều khiển hoạt động của Timer 1.
- TF0 TCON.5 Tương tự TF1 nhưng là cho Timer 0.
- TR0 TCON.4 Tương tự TR1 nhưng là cho Timer 0.
- IE1 TCON.3 Cờ ngắt ngoài 1 kích phát sườn, được CPU thiết lập khi phát hiện có sườn xuống ngắt ngoài và được CPU xoá khi ngắt được xử lý. Cờ này chỉ chốt ngắt kích phát sườn nhưng không chốt ngắt kích phát mức thấp.
- IT1 TCON.2 Bit điều khiển ngắt 1 được thiết lập và xoá bởi phần mềm để xác định kiểu ngắt ngoài kích phát sườn xuống hay mức thấp.
- IE0 TCON.1 Tương tự IE1 nhưng là cho ngắt ngoài 0.
- IT0 TCON.0 Tương tự IT1 nhưng là cho ngắt ngoài 0.

Chúng ta phải chú ý một điểm nữa rằng: đối với ngắt kích phát sườn, nguồn ngắt ngoài phải được giữ ở mức cao tối thiểu một chu kỳ máy, và sau đó duy trì ở mức thấp cũng tối thiểu một chu kỳ máy để đảm bảo bộ vi điều khiển nhận biết được quá trình chuyển sườn xung từ cao xuống thấp. Hình 3.27 biểu diễn thời gian xung tối thiểu để phát hiện ra ngắt phát sườn với tần số XTAL = 11,0592MHz.



Hình 3.27. Thời gian tối thiểu phát hiện ngắt kích phát sườn

Ví dụ: Giả thiết chân P3.3 (INT1) được nối với một máy tạo xung. Hãy viết chương trình trong đó sườn xuống của xung sẽ chuyển chân P1.3 lên cao, chân này được nối tới LED báo.

Giải:

```

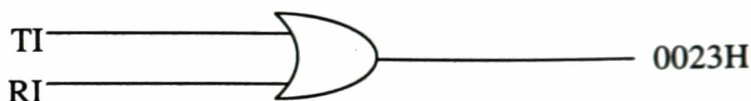
ORG      0000H

LJMP     MAIN      ;bo qua bang vector ngat
;--- chương trình ngat
ORG      0013H      ;trinh ISR cho INT1
SETB     P1.3       ;bat den LED
MOV       R3, #0FFH
BACK:    DJNZ      R3, BACK;giu den LED sang mot luc
CLR       P1.3       ;tat den LED
RETI                      ;ve chương trình chính
;--- chương trình chính
ORG      30H
MAIN:    SETB      TCON.2 ;ngat kích phát suon
MOV       IE, #84H;cho phép ngat ngoài INT1
HERE:    SJMP      HERE   ;doi cho den khi duoc ngat
END

```

3.4.7.7. Lập trình ngắt truyền thông nối tiếp cho 8051

Ở 8051 chỉ có một ngắt dành riêng cho truyền thông nối tiếp. Ngắt này được dùng cho cả phát và thu dữ liệu. Nếu bit ngắt trong thanh ghi IE (IE.4) được thiết lập, khi RI (TI) bật lên thì 8051 nhận được ngắt và nhảy đến địa chỉ trình phục vụ ngắt dành cho truyền thông nối tiếp 0023H trong bảng vector ngắt và thực hiện nó. Lúc đó ta cần kiểm tra xem cờ nào gây ngắt để đáp ứng cho phù hợp.



Hình 3.28. Ngắt truyền thông nối tiếp được kích hoạt bởi các cờ TI và RI

Chúng ta cũng cần lưu ý rằng trước lệnh trở về chương trình chính RETI của ISR là các lệnh xoá cờ RI và TI. Việc xoá cờ là không thể thiếu, vì ngắt dùng để 8051 thu và phát không biết được nguồn gây ra ngắt. Vì vậy, trình phục vụ ngắt phải xoá các cờ này để cho phép ngắt tiếp theo được đáp ứng sau khi kết thúc ngắt.

Ví dụ: Hãy viết chương trình sử dụng các ngắt thực hiện các công việc sau:

- Đọc dữ liệu từ cổng P1 và truyền nối tiếp với tốc độ truyền 4800 baud và sao dữ liệu cổng P1 vào cổng P2.
- Thu dữ liệu nối tiếp và gửi đến cổng P0.
- Sử dụng Timer 0 tạo sóng vuông tần số 50KHz trên P3.1.

Giả thiết tần số XTAL = 11,0592MHz

Giải:

```

ORG      0
LJMP     MAIN                ;bo qua bang vectơ ngắt
ORG      000BH               ;ISR tao xung vuong
        CPL      P0.1
        RETI
        ORG      23H        ;dia chi cho ngắt truyền nt
        SJMP     SERIAL
;--- chương trình chính
        ORG      30H
MAIN:    MOV P1, #0FFH;chon cong P1 lam cong vào
  
```

```

MOV      TMOD, #22H   ;chon Timer 0 va
                        ;Timer 1 che do 2

MOV      TH1, #0FAH   ;chon toc do baud 4800
MOV      SCON, #50H

MOV      TH0, #92

MOV      IE, #10010010B

SETB     TR1

SETB     TR0

BACK:    MOV      A, P1

MOV      SBUF, A

MOV      P2, A

SJMP     BACK

ORG      100H

SERIAL:  JB       TI, TRANS

MOV      A, SBUF

MOV      P0, A

CLR      RI

.RETI

TRANS:   CLR      TI

RETI

END

```

Ví dụ tổng quát:

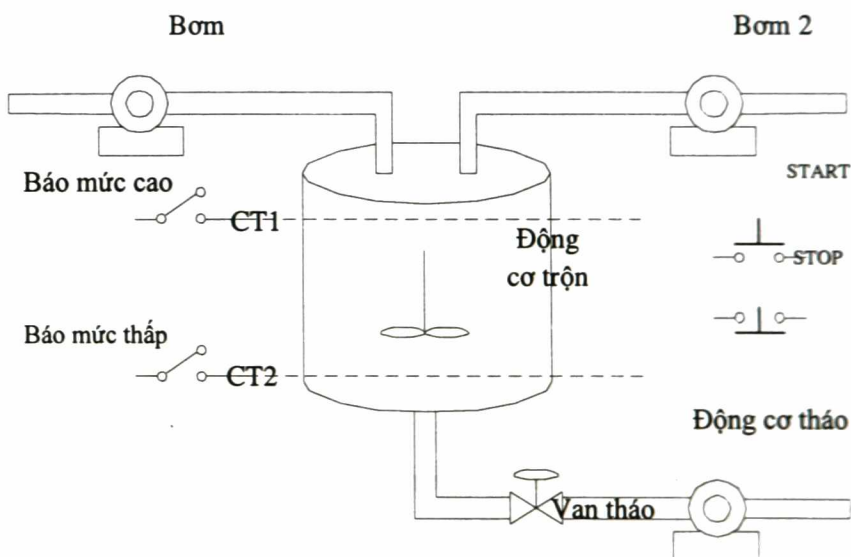
Cho hệ thống máy trộn sơn như hình 3.29:

- Bơm 1, bơm 2 là hai động cơ bơm hai loại sơn khác màu cần phải trộn.

- CT1, CT2 là hai sensor báo mức cao và mức thấp. Hai sensor này tác động ngược nhau. Khi CT1 đóng thì CT2 mở và ngược lại.
- START, STOP là nút khởi động và nút dừng hệ thống.
- Động cơ trộn: Trộn hai loại sơn khác màu.
- Van tháo và động cơ tháo dùng để tháo hỗn hợp sơn sau khi trộn xong.

❖ Yêu cầu công nghệ

- Khi bật nguồn hệ thống tự động kiểm tra lượng sơn trong bình, nếu còn sơn (CT2 tác động) thì tự động xả sơn cũ ra ngoài cho đến hết.
- Nếu sơn hết thì tự động bơm hai loại sơn nhờ Bơm 1 và Bơm 2, khi sơn đầy (CT1 tác động) thì ngừng Bơm 1 và Bơm 2.



Hình 3.29. Sơ đồ máy trộn sơn

- Khởi động động cơ trộn để trộn sơn trong 10s.
- Sau khi trộn 10s thì cắt động cơ trộn, khởi động van tháo và động cơ tháo để bơm sơn vào các hộp bên ngoài.
- Khi sơn được bơm hết ra ngoài (CT2 lại tác động) thì đóng van tháo và động cơ tháo rồi lại khởi động Bơm 1 và Bơm 2.
- Hệ thống sẽ tự dừng lại sau 10 lần trộn hoặc khi ấn nút STOP.

❖ **Yêu cầu:**

- Vẽ sơ đồ ghép nối giữa vi điều khiển 8051 với hệ thống, gán các địa chỉ đầu vào, địa chỉ đầu ra.

Ứng dụng các ngắt, các bộ Timer/Counter để lập trình điều khiển hệ thống trên. Từ yêu cầu công nghệ như trên ta xây dựng được lưu đồ thuật toán như sau:

```
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
;CHUONG TRINH DIEU KHIEN MAY TRON SON
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
;KET NOI CAC TIN HIEU RA: BOM 1, 2 (P1.0),
;DONG CO TRON (P1.2), VAN+DC THAO (P1.3)
;KET NOI CAC TIN HIEU VAO: START (P2.0), CT1
(P2.1), CT2 (P2.2), STOP (P3.2)
;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ORG      0000H

LJMP     MAIN

ORG      0003H      ;dia chi bang vecto ngat
                  ;int0 (stop)
```

```

NGUNG:  MOV P1,#00H      ;cat tat ca
        ;cac dong co dau ra

SJMP    NGUNG

RETI

ORG     0030H ;dia chi chuong trinh chinh

MAIN:   MOV     R0, #10 ;dat so lan tron=10.

MOV     IE, #10000001B; cho phep ngat int0

MOV     P2, #0FFH;lay p2 lam cong vao

MOV     P1, #00H ;cat tat ca cac
        ;dong co dau ra

SETB    P3.2      ;dat p3.2 la dau vao
        ;ngat int0 (stop)

HERE:   JB      P2.0, HERE;k.tra bam
        ;start chua?

JNB     P2.2, XA ;kiem tra ct2=0?
        ;neu dung thi xa, sai thi bom

SJMP    BOM              ;nhay den bom

XA:     SETB    P1.2      ;mo van xa

JB      P2.2, KTRA      ;kiem tra ct2=1?
        ;neu dung thi k.tra bo dem=10?,
        ;neu sai thi tiep tuc xa

SJMP    XA

KTRA:   DEC     R0

CJNE    R0, #0,BOM;da tron du 10 lan chua?

```

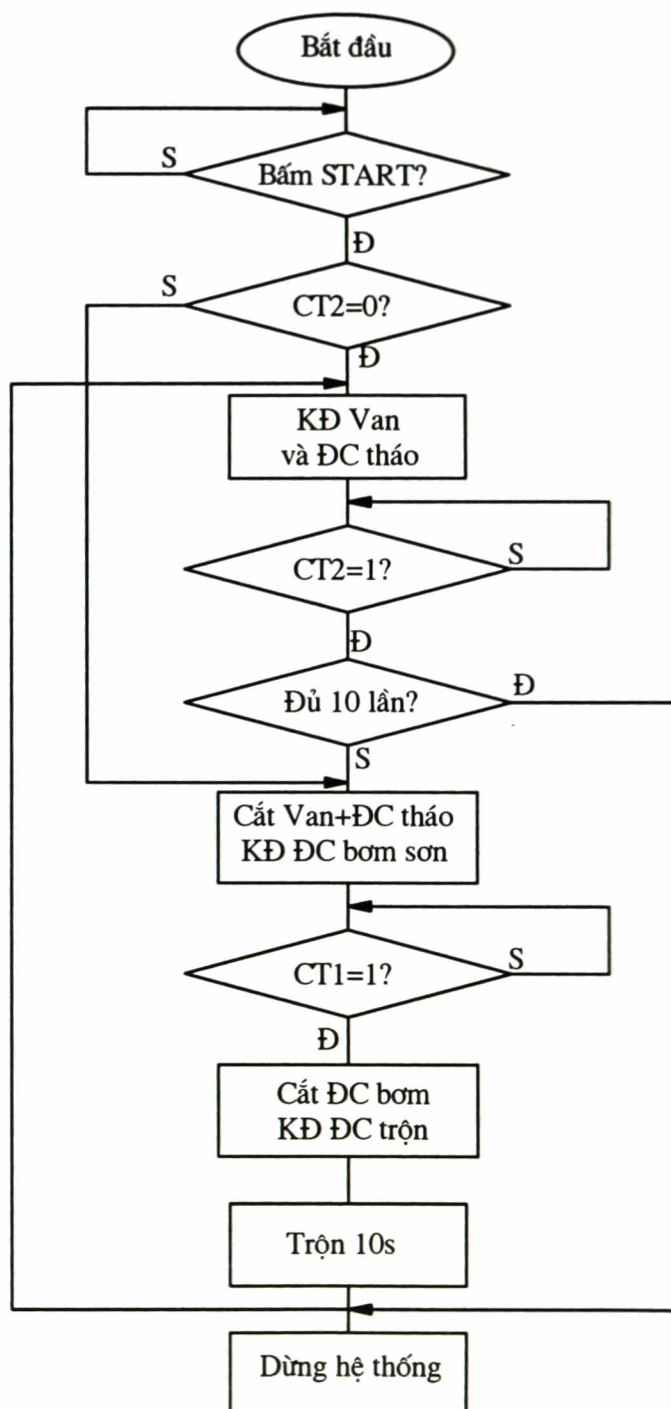


```

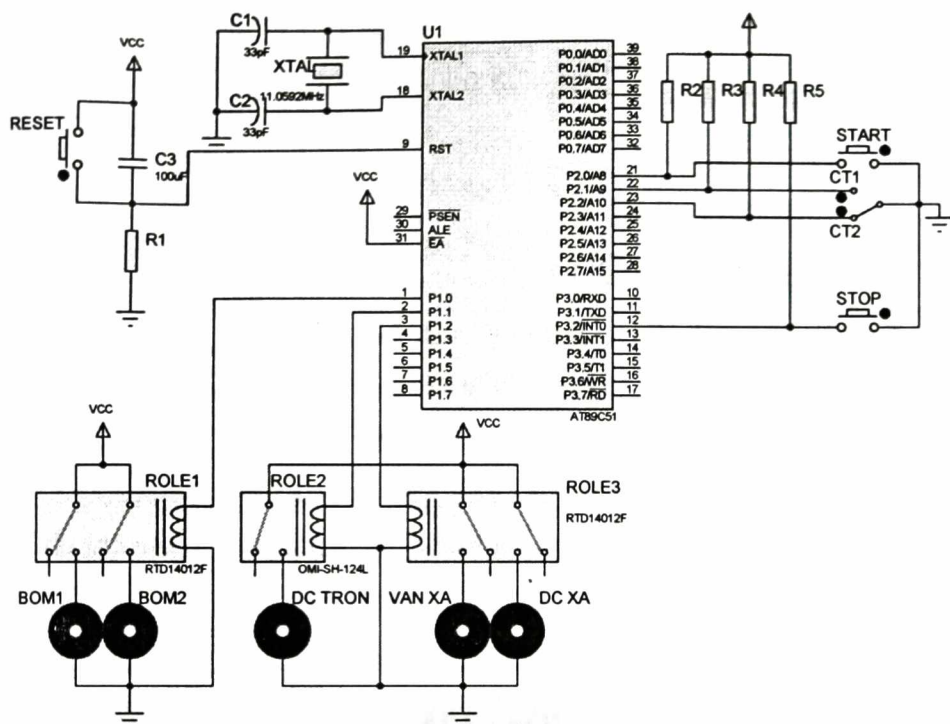
        SJMP      NGUNG      ;neu du thi dung h.thong
BOM:    CLR       P1.2       ;cat van xa
        SETB      P1.0       ;khoidongdongco bom
        JB        P2.1, TRON;kiemtra xem ct1=1?
        ;neudungthi tron,sai thi bom tiep
        SJMP      BOM        ;tiectuc bom
TRON:   CLR       P1.0       ;ngung bom
        SETB      P1.1       ;khoidongdongco tron
        ACALL     DELAY      ;taothoigiantre
        CLR       P1.1       ;ngung tron
        SJMP      XA         ;xa son
DELAY:  CLR       TF1        ;xoa co time 1
        MOV       TMOD, #10H;su dung timer 1, mode 1
        MOV       R3, #250 ;taotre 10s
AGAIN:  MOV       TL1, #0FEH
        MOV       TH1, #6FH
        SETB      TR1        ;khoidong timer 1
BACK:   JNB       TF1, BACK;k.tra cobodinhthoi
        CLR       TR1        ;tat timer 1
        CLR       TF1        ;xoa co timer 1
        DJNZ      R3, AGAIN;neur3 ≠ 0 thilap lai
        RET

```

END



Hình 3.30. Lưu đồ thuật toán cho hệ thống máy trộn sơn



Hình 3.31

• Phần bài tập

Bài 1. Thế nào là vi điều khiển? Giữa vi xử lý và vi điều khiển có điểm giống và khác nhau cơ bản nào? Sự ra đời của vi điều khiển nhằm sử dụng cho những ứng dụng như thế nào? Lấy ví dụ.

Bài 2. Hãy giải thích tại sao bộ vi xử lý 8086/8088 quản lý được bộ nhớ có dung lượng tối đa là 1MB, còn bộ vi điều khiển 8051 chỉ quản lý được bộ nhớ có dung lượng tối đa là 64 KB

Bài 3. Hãy cho biết trong các dòng lệnh sau, dòng lệnh nào là lệnh sai. Hãy giải thích và sửa lại cho đúng:

- MOV A, #30h
- MOV R0,R1
- PUSH A

- d. MOV A, 30h
- e. ADD R0, 30h
- f. MOV @R0,A
- g. MOV A, @ R2
- h. MOV A, #0FF30h

Bài 4. Hãy viết đoạn chương trình kiểm tra xem thanh ghi tích lũy có chứa một số chẵn không? Nếu có thì chia cho 2, nếu không thì tăng lên 1 để chẵn hoá số đó rồi chia nó cho 2?

Bài 5. Viết chương trình để xoá Acc sau đó cộng 5 vào Acc 15 lần liên tiếp. Cuối cùng gửi kết quả ra ô nhớ bên ngoài có địa chỉ chứa trong thanh ghi R5?

Bài 6. Xem đoạn chương trình trong bảng 3.15:

Bảng 3.15

Dòng	M lệnh
1	ADD A,R2
2	MOV DPTR,#0AC10h
3	MOV R0,12h
4	MOVSX A,@R0
5	PUSH R3

- a. Hãy xác định chế độ địa chỉ của lệnh trên dòng 1
- b. Hãy xác định chế độ địa chỉ của lệnh trên dòng 2.
- c. Hãy xác định chế độ địa chỉ của lệnh trên dòng 3.
- d. Hãy xác định chế độ địa chỉ của lệnh trên dòng 4.
- e. Hãy giải thích ngắn gọn vì sao lệnh trên dòng 5 lại không hợp lệ?

Bài 7. Sử dụng 1 vi mạch 74138 và các cổng cần thiết để thiết kế mạch giải mã địa chỉ tạo ra các tín hiệu chọn chip tương ứng các vùng địa chỉ sau:

Bảng 3.16

Tín hiệu chọn chip	Vùng địa chỉ	Đặc tính truy xuất
$\overline{CS0}$	0000H - 3FFFH	$\overline{PSEN}$
$\overline{CS1}$	4000H - 7FFFH	$\overline{PSEN}$
$\overline{CS2}$	6000H - 7FFFH	$\overline{RD}, \overline{WR}$
$\overline{CS3}$	8000H - 87FFFH	$\overline{RD}$
$\overline{CS4}$	8800H - 8FFFFH	$\overline{WR}$

Bài 8. Sử dụng 1 vi mạch 74138 và các cổng cần thiết để thiết kế mạch giải mã địa chỉ tạo ra các tín hiệu chọn chip tương ứng các vùng địa chỉ sau:

Bảng 3.17

Tín hiệu chọn chip	Vùng địa chỉ	Đặc tính truy xuất
$\overline{CS0}$	9800H - 9BFFFH	$\overline{PSEN}$
$\overline{CS1}$	9800H - 9BFFFH	$\overline{RD}, \overline{WR}$
$\overline{CS2}$	9C00H - 9DFFFH	$\overline{RD}, \overline{WR}$
$\overline{CS3}$	9E00H - 9EFFFH	$\overline{RD}, \overline{WR}$

Bài 9. Chỉ dùng một vi mạch 74138 (không dùng thêm cổng), thiết kế mạch giải mã địa chỉ tạo ra một tín hiệu chọn chip /CS tương ứng tầm địa chỉ F000H - F3FFFH.

Bài 10. Giả sử Port 1 được nối đến một thiết bị thu dữ liệu (ví dụ như 8 Sensor). Hãy viết chương trình nhận liên tiếp 10 byte dữ liệu từ thiết bị thu này và ghi vào 10 ô nhớ (RAM nội) liên tiếp bắt đầu từ ô nhớ 50H. Chuyển một chuỗi dữ liệu gồm 10 byte trong RAM nội có địa chỉ đầu là 30H đến vùng RAM ngoài có địa chỉ đầu là 4000H.

Bài 11. Viết chương trình tạo chuỗi xung vuông có $f = 100 \text{ KHz}$ tại chân P1.1 với độ dày 50% (Xtal 12 MHz).

Bài 12. Cho một chuỗi số 8 bit không dấu trong RAM nội gồm 10 số bắt đầu từ ô nhớ 30H. Hãy viết chương trình con cộng chuỗi số này và ghi kết quả vào ô nhớ 2FH (giả sử kết quả nhỏ hơn hoặc bằng 255).

Bài 13. Cho một chuỗi ký tự dưới dạng mã ASCII trong RAM nội, dài 20 byte, bắt đầu từ địa chỉ 50H. Viết chương trình xuất các ký tự in hoa có trong chuỗi này ra Port 1. Biết rằng mã ASCII của ký tự in hoa là từ 65 (chữ A) đến 90 (chữ Z).

Bài 14. Viết chương trình chờ nhận data từ 1 thiết bị ngoài gửi đến 8051 qua port nối tiếp (chế độ UART 8 bit, 19200 baud). Nếu nhận được ký tự STX (02H) thì bật sáng LED, nếu nhận được ký tự ETX (03H) thì tắt LED, biết rằng LED được điều khiển bằng ngõ P1.3 (LED sáng khi bit điều khiển bằng 1). Sử dụng ngắt serial. Xtal 11,059MHz.

Bài 15. Viết chương trình chờ nhận 1 xung cạnh xuống đưa vào chân /INT0 (P3.2), khi có xung thì nhập data từ Port1 và phát ra port nối tiếp ở chế độ UART 9 bit 4800 baud, bit thứ 9 là bit parity lẻ. Xtal 6MHz.

Bài 16. Viết chương trình đếm số xung đưa vào chân /INT1 (P3.3) và điều khiển relay thông qua chân P3.0 (relay đóng khi P3.0 bằng 1), cất số đếm vào ô nhớ 40H của Ram nội, nếu số đếm chưa đến 100 thì đóng relay, nếu số đếm đạt 100 thì ngắt relay.

CHƯƠNG IV

GHÉP NỐI VÀ TRAO ĐỔI DỮ LIỆU TRONG HỆ VI XỬ LÝ

• Phần lý thuyết

**** Mục đích của chương:***

Giúp sinh viên làm quen với việc ghép nối vi điều khiển với bộ nhớ, với vi mạch lập trình song song chuyên dụng 8255A, biết cách ghép nối, và lập trình vi điều khiển với các thiết bị ngoại vi đang được ứng dụng nhiều trong thực tế.

Từ những ví dụ và những kiến thức cơ bản này sinh viên có thể tự tạo ra những mạch điện rất hữu dụng trong thực tế và hoàn toàn có thể tự thiết kế được.

**** Tóm tắt nội dung:***

Nội dung chương IV tập trung trình bày về ghép nối vi xử lý – vi điều khiển với bộ nhớ, thiết bị nhập xuất. Trong chương này cũng giới thiệu về vi mạch cổng song song lập trình được PPI 8255 – một vi mạch cổng được sử dụng trong rất nhiều bài toán ghép nối khác nhau. Phần cuối của chương là các sơ đồ ghép nối tiêu biểu như: ghép nối vi điều khiển với bộ nhớ, màn tinh thể lỏng LCD, chuyển đổi tương tự sang số (ADC), số sang tương tự (DAC), bàn phím (Keyboard), LED 7 thanh, động cơ bước (stepper motor), ...

4.1. GHÉP NỐI VI XỬ LÝ/VI ĐIỀU KHIỂN VỚI BỘ NHỚ NGOÀI

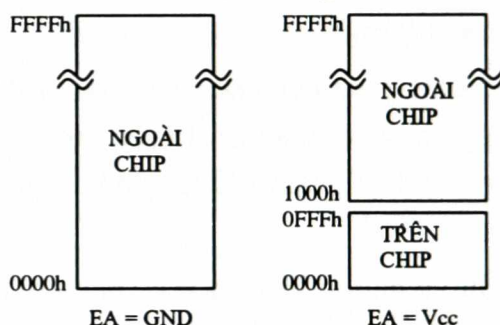
Trong chương II chúng ta đã nói đến chức năng cũng như một số bộ nhớ bán dẫn thông dụng. Trong phần này chúng ta sẽ tiếp tục

nghiên cứu cách thức ghép nối vi điều khiển 8051 với bộ nhớ ngoài khi muốn mở rộng thêm không gian nhớ.

4.1.1. MỘT SỐ CHÂN CỦA 8051 PHỤC VỤ CHO GHÉP NỐI VỚI BỘ NHỚ NGOÀI

4.1.1.1. Chân EA

Ở các ví dụ từ trước đến nay, chúng ta sử dụng ROM trên chip hoặc ROM ngoài để lưu mã chương trình. Nếu muốn sử dụng đồng thời cả hai bộ nhớ ROM thì có được không? Câu trả lời là có (hình 4.1).



Hình 4.1. Bộ nhớ ROM trên chip và ngoài chip

– Bằng cách nối EA lên Vcc, khi RESET thì 8051 thực hiện chương trình ở trên chip trước, khi chạy xong thì mới chuyển sang chạy chương trình trên ROM ngoài. 8051 sẽ nạp các mã lệnh bắt đầu từ địa chỉ 0000h đến 0FFFh (địa chỉ cuối cùng của ROM trên chip). Sau đó, bộ đếm chương trình (con trỏ lệnh PC) tạo ra địa chỉ 1000h và tự động chuyển hướng ra ROM ngoài có chứa mã chương trình.

– Nếu nối EA xuống Mass thì vi điều khiển sẽ chỉ thực hiện lệnh có trên ROM ngoài từ địa chỉ 0000h đến FFFFh

4.1.1.2. Cổng P0 và P2

Do thanh ghi PC và thanh ghi DPTR là thanh ghi 16 bit nên vi điều khiển 8051 có thể mở rộng không gian nhớ tới 64 KB cho mỗi loại. Cổng P0 và P2 dùng để cung cấp địa chỉ. Cổng P0 cấp 8 bit địa

The diagram shows an 8051 microcontroller (U65) interfaced with an 8051 EPROM (U65) and an 8051 SRAM (U66). The 8051 microcontroller (U65) is connected to the 8051 EPROM (U65) via its P0, P2, P3, and P4 ports. The 8051 microcontroller (U65) is also connected to the 8051 SRAM (U66) via its P0, P2, P3, and P4 ports. The 8051 microcontroller (U65) is connected to the 8051 SRAM (U66) via its P0, P2, P3, and P4 ports. The 8051 microcontroller (U65) is connected to the 8051 SRAM (U66) via its P0, P2, P3, and P4 ports.

4.1.1.3. Chân PSEN

227

chương trình ngoài. Nếu không ghép nối với bộ nhớ chương trình ngoài thì chân PSEN để trống.

4.1.1.4. Chân $\overline{RD}$ và $\overline{WR}$

Là chân cho phép đọc và ghi dữ liệu khi ghép nối 8051 với bộ nhớ dữ liệu ngoài. Chân $\overline{RD}$ sẽ nối với chân OE của bộ nhớ dữ liệu ngoài. Chân $\overline{WR}$ sẽ nối với chân $\overline{WE}$ của bộ nhớ dữ liệu ngoài. Khi đọc dữ liệu từ bộ nhớ vào vi điều khiển thì chân $\overline{RD}$ sẽ tích cực ở mức thấp, ngược lại khi ghi dữ liệu từ vi điều khiển ra bộ nhớ thì chân $\overline{WR}$ sẽ tích cực ở mức thấp.

4.1.2. GHEP NỐI VI ĐIỀU KHIỂN 8051 VỚI BỘ NHỚ CHƯƠNG TRÌNH NGOÀI

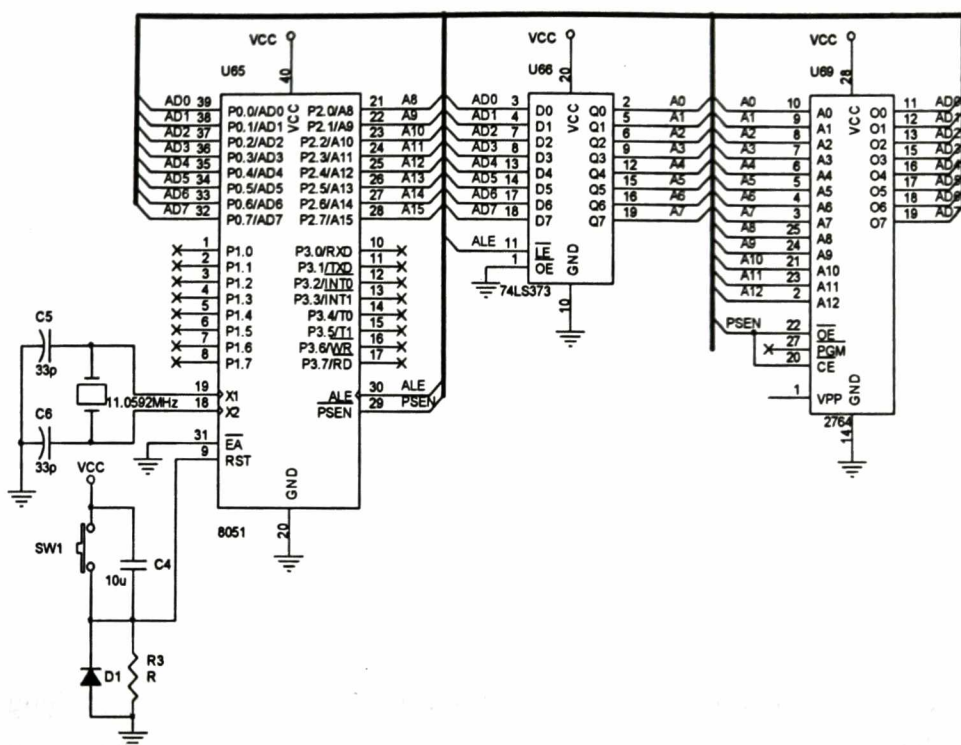
Việc ghép nối vi điều khiển 8051 với bộ nhớ chương trình ngoài không có gì là đặc biệt, chỉ lưu ý rằng chân PSEN luôn được nối với chân OE của bộ nhớ bán dẫn.

Ví dụ: Thực hiện ghép nối vi điều khiển 8051 với bộ nhớ ROM chương trình ngoài 2764 (8Kx8).

Giải:

IC nhớ 2764 (8Kx8) có 13 chân địa chỉ từ A0 – A12 (Vì $1K = 2^{10}$ byte $\Rightarrow 8KB = 2^3 \cdot 2^{10} = 2^{13}$ byte nên nó có 13 chân địa chỉ). Số ô nhớ mà IC này tạo ra là $2^{13} = 8192$ ô nhớ = 2000h ô nhớ. Nếu nối chân EA xuống Mass thì vùng nhớ cho phép nạp chương trình sẽ là 0000h – 1FFFh. Sơ đồ kết nối như sau:

Khi vi điều khiển lấy lệnh từ bộ nhớ thì chân PSEN được kích hoạt xuống mức thấp, do vậy chân PSEN được nối với chân OE và CE của ROM chương trình 2764.



Hình 4.3. Ghép nối vi điều khiển 8051 với bộ nhớ chương trình ngoài

4.1.3. GHÉP NỐI VI ĐIỀU KHIỂN 8051 VỚI BỘ NHỚ DỮ LIỆU NGOÀI

Ta biết rằng thanh ghi bộ đếm chương trình PC của 8051 là 16 bit, do vậy có thể truy cập được $2^{16} = 64\text{Kb}$ mã chương trình. Ở nhiều ví dụ dữ liệu được đặt trong không gian mã chương trình và dùng lệnh **MOVC A, @A+DPTR** để lấy dữ liệu. Chữ C trong lệnh MOVC là từ chữ mã lệnh (Code) để báo rằng dữ liệu được đặt trong không gian mã lệnh của 8051. Tuy nhiên, họ 8051 còn có không gian dữ liệu riêng biệt.

Vì thanh ghi con trỏ dữ liệu DPTR có độ dài 16 bit nên nó cũng có thể quản lý được tối đa 64Kb ô nhớ dữ liệu. Như vậy, 8051 có tổng cộng 128Kb không gian địa chỉ, trong đó 64Kb dành cho mã chương

trình và 64Kb dành cho dữ liệu. Không gian chương trình được truy cập nhờ thanh ghi PC, còn không gian dữ liệu được truy cập nhờ thanh ghi DPTR và một lệnh có tên MOVX (chữ X là từ External để chỉ không gian bộ nhớ dữ liệu được thực hiện từ bộ nhớ ngoài). Để chuyển dữ liệu lưu trữ ở bộ nhớ dữ liệu ngoài vào CPU lệnh **MOVB A, @DPTR** được thực hiện và lệnh sẽ đọc byte dữ liệu do thanh ghi DPTR trỏ đến. Mặc dù cả hai lệnh **MOVB A, @A+DPTR** và **MOVB A, @DPTR** khá giống nhau về hình thức, nhưng điểm khác nhau cơ bản một lệnh thực hiện nhận dữ liệu từ vùng nhớ chương trình, còn lệnh kia thì nhận dữ liệu từ vùng dữ liệu.

Để nối ghép 8051 với bộ nhớ dữ liệu ngoài thì cần sử dụng chân RD để điều khiển đọc dữ liệu và chân WR để điều khiển ghi dữ liệu. Bộ nhớ dữ liệu có thể là ROM dữ liệu hoặc RAM dữ liệu. Khi kết nối với ROM dữ liệu thì chỉ cần sử dụng chân RD (vì bộ nhớ ROM là bộ nhớ chỉ đọc), còn khi kết nối với bộ nhớ RAM dữ liệu thì cần sử dụng cả hai chân RD và WR (bộ nhớ RAM vừa cho phép đọc vừa cho phép ghi dữ liệu).

Ví dụ 1: Hãy vẽ sơ đồ kết nối vi điều khiển 8051 với bộ nhớ ROM dữ liệu dung lượng 8Kx8 và viết chương trình đọc 30 byte dữ liệu ở bộ nhớ dữ liệu ngoài từ địa chỉ 1000h đưa ra cổng P1.

Giải:

Sơ đồ kết nối 8051 với bộ nhớ ROM dữ liệu như hình 4.4.

Lưu ý rằng ta sử dụng cổng logic để làm bộ giải mã cho ROM dữ liệu tạo ra địa chỉ vùng dữ liệu từ 0000h – 1FFFh và chân RD của 8051 được nối với chân OE của ROM dữ liệu. Chương trình được viết như sau:

```

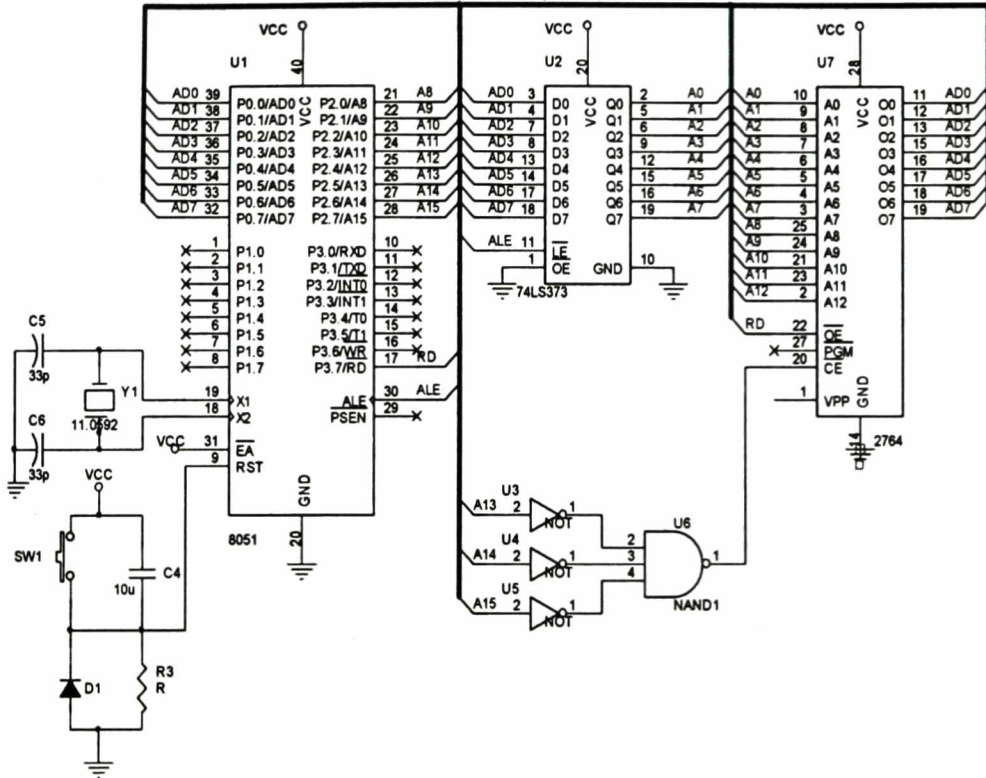
MOV      DPTR, #1000H; địa chỉ đầu ROM ngoài
MOV      R2, #30      ; số byte cần chuyển vào
AGAIN:
MOVXB    A, @DPTR

```

```

MOV     P1, A
INC     DPTR
DJNZ    R2, AGAIN

```



Hình 4.4. Sơ đồ kết nối 8051 với ROM dữ liệu 8Kx8

Ví dụ 2: Vẽ sơ đồ ghép nối 8051 với RAM dữ liệu ngoài 6264 dung lượng 8Kx8 và viết chương trình để 8051 lấy 10 byte dữ liệu được cất từ địa chỉ 1000h trên RAM dữ liệu ngoài đưa vào RAM nội có địa chỉ từ 30h.

Giải: Sơ đồ kết nối như hình 4.5.

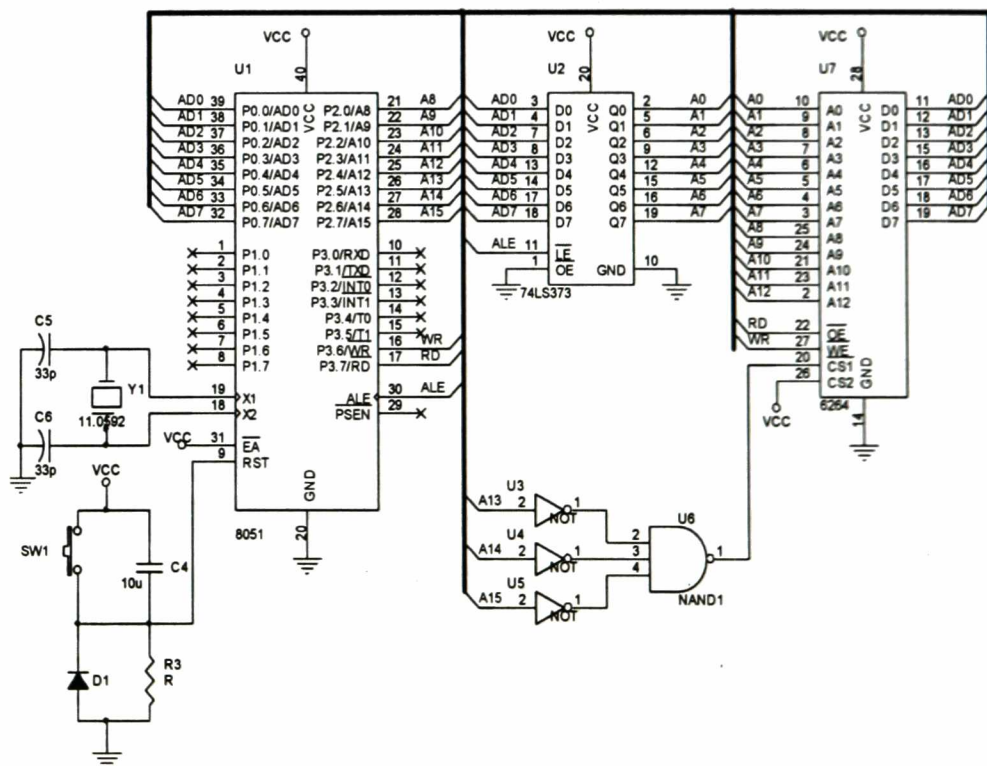
Lưu ý rằng ta sử dụng cổng logic để làm bộ giải mã cho RAM dữ liệu tạo ra địa chỉ vùng dữ liệu từ 0000h – 1FFFh và chân RD, WR của 8051 được nối lần lượt với chân OE, WE của RAM dữ liệu.

Chương trình được viết như sau:

```
MOV     DPTR, #1000H; địa chỉ đầu RAM ngoài
MOV     R2, #10      ; số byte cần chuyển
MOV     R0, #30H     ; địa chỉ đầu RAM nội
```

AGAIN:

```
MOVX    A, @DPTR
MOV      @R0, A
INC      DPTR
INC      R0
DJNZ     R2, AGAIN
```



Hình 4.5. Sơ đồ kết nối vi điều khiển 8051 với bộ nhớ RAM dữ liệu 6264

Trường hợp muốn ghép nối nhiều ROM, RAM thì ta phải sử dụng vi mạch giải mã 74LS138 để giải mã cho từng IC như ví dụ dưới đây.

Ví dụ 3: Sử dụng một vi mạch giải mã 74LS138 để thiết kế mạch giải mã địa chỉ tạo ra các tín hiệu chọn chip và thực hiện vẽ sơ đồ ghép nối vi điều khiển 8051 với bộ nhớ tương ứng.

Bảng 4.1

Tín hiệu chọn chip	Vùng địa chỉ	Đặc tính truy xuất	Loại bộ nhớ Dung lượng
$\overline{CS0}$	0000H - 3FFFH	$\overline{PSEN}$	EPROM 27128 16Kx8
$\overline{CS1}$	4000H - 7FFFH	$\overline{PSEN}$	EPROM 27128 16Kx8
$\overline{CS2}$	8000H - 9FFFH	$\overline{RD}, \overline{WR}$	HM6264B 8Kx8

Giải:

Trước hết ta hãy phân tích về vùng nhớ của từng IC trên.

- EPROM 27128 (16Kx8) có 14 chân địa chỉ từ A0 – A13 vì $16K = 2^4 \cdot 2^{10} = 2^{14}$ byte.
- IC này có thể địa chỉ được 2^{14} byte = 2^{14} ô nhớ = 4000h ô nhớ. Nên nếu địa chỉ của ô nhớ đầu tiên là 0000h thì địa chỉ của ô nhớ cuối cùng phải là 3FFFh.
- HM6264B (8Kx8) là RAM có 13 chân địa chỉ từ A0 – A12 vì $1K = 2^{10}$ byte $\Rightarrow 8KB = 2^3 \cdot 2^{10} = 2^{13}$ byte.

Bảng phân chia vùng nhớ được xác định như sau:

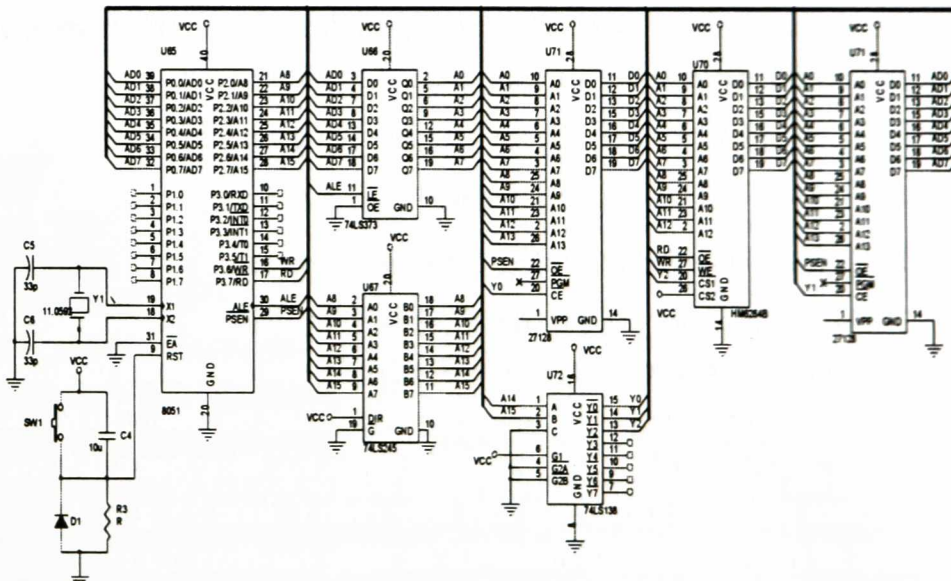
Những vùng có địa chỉ thay đổi đã được đánh dấu như trên bảng 4.2. Những vùng này sẽ được đưa vào những chân địa chỉ tương ứng của từng IC. Ở đây ta sử dụng 3 IC nhớ, do vậy sẽ dùng ba chân đầu ra của IC giải mã 74LS138 là Y0, Y1, Y2. Đối chiếu với bảng trạng thái của IC này (xem phần các mạch phụ trợ) thấy chân C luôn bằng 0, do vậy chân này sẽ được nối xuống Mass. Ta sử dụng hai chân địa chỉ cao là A14 và A15 để phối hợp với A và B tạo ra các tín hiệu chọn chip.

Bảng 4.2

IC	A 1 5	A 1 4	A 1 3	A 1 2	A 1 1	A 1 0	A 9	A 8	A 7	A 6	A 5	A 4	A 3	A 2	A 1	A 0	Địa chỉ
IC1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
IC2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
IC3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Sơ đồ kết nối như hình vẽ 4.6.

Trong sơ đồ ta sử dụng một IC đệm bus là 74LS245 mục đích để tăng công suất cho bus (xem phần các mạch phụ trợ).



Hình 4.6. Sơ đồ kết nối 8051 với bộ nhớ chương trình và dữ liệu ngoài

4.2. GHÉP NỐI SONG SONG 8255A VÀ LẬP TRÌNH

4.2.1. TỔNG QUAN

Với vi điều khiển đã có sẵn các cổng vào/ra. Nhưng trong trường hợp yêu cầu kỹ thuật đòi hỏi nhiều cổng vào/ra hơn hoặc đã sử dụng một số cổng để ghép nối với bộ nhớ. Lúc đó ta cần phải ghép thêm các thiết bị để tăng số lượng cổng vào/ra.

Một trong những vi mạch ghép nối song song được sử dụng rộng rãi nhất hiện nay là IC 8255A. 8255A là một vi mạch ghép nối được dùng rất phổ biến cho các hệ vi xử lý 8 - 16 bit. Nó cho phép giao tiếp mềm dẻo trong nhiều ứng dụng thực tế với nhiều cửa vào ra. Đối với các hệ vi xử lý nhỏ, 8255A có thể nối trực tiếp với các mạch vi xử lý. Trong các hệ thống lớn hơn, việc ghép nối chỉ cần thêm một số mạch tối thiểu. Đó là một trong những tính chất quan trọng của bộ ghép nối 8255A.

Tính linh hoạt của vi mạch này thể hiện ở khả năng lập trình. Qua một thanh ghi điều khiển, người sử dụng có thể đặt chế độ hoạt động và cấu hình của các cửa vào ra. Các chân số liệu tạo nên Bus dữ liệu theo hai hướng, rộng 8 bit. Tất cả các dữ liệu khi truy nhập đọc hoặc ghi được dẫn qua các đường dẫn này.

4.2.2. GHÉP NỐI 8051 VỚI 8255A

4.2.2.1. Chức năng các chân của 8255A

8255A là một chip DIP 40 chân (sơ đồ chân hình 4.7 và sơ đồ khối). Có ba cổng truy cập riêng biệt. Cổng A, cổng B, cổng C và đều là các cổng 8 bit. Các cổng này có thể là các cổng vào hoặc ra và có thể thay đổi một cách linh hoạt.

– Các chân từ PA0 – PA7: Có thể lập trình thành 8 bit đầu vào hoặc 8 bit đầu ra hoặc cả 8 bit hai chiều vào/ra.

– Các chân từ PB0 – PB7: Có thể lập trình thành 8 bit đầu vào hoặc 8 bit đầu ra hoặc cả 8 bit hai chiều vào/ra.

– Các chân từ PC0 – PC7: Có thể lập trình thành 8 bit đầu vào hoặc 8 bit đầu ra. 8 bit này cũng có thể được chia thành hai phần: Các bit cao (PC4 – PC7) là PCH và các bit thấp (PC0 – PC3) là PCL. Mỗi phần có thể được dùng làm đầu vào hoặc đầu ra.

– Các chân $\overline{RD}$ và $\overline{WR}$: Đây là hai chân điều khiển tích cực ở mức thấp tới 8255A được nối từ $\overline{RD}$, $\overline{WR}$ của vi xử lý. Đó là các tín hiệu điều khiển đọc, ghi. Khi tín hiệu chọn chip ($\overline{CS}$) tích cực ở mức thấp ($\overline{CS} = 0$) và $\overline{RD} = 0$ thì các tín hiệu của cổng được chọn dẫn đến Bus dữ liệu và có thể được gọi bởi các vi mạch khác. Khi $\overline{WR} = 0$ thì mọi việc xảy ra ngược lại, dữ liệu từ Bus dữ liệu được đưa đến cổng đã chọn.

– Các chân D0 – D7: Là các chân dữ liệu của 8255A. Các chân này được nối tới các chân dữ liệu của vi xử lý để cho phép nó trao đổi dữ liệu giữa vi xử lý và chip 8255A.

– Chân RESET: Đây là đầu vào tín hiệu tích cực ở mức cao tới 8255A, được dùng để xoá thanh ghi điều khiển. Khi RESET được kích hoạt thì tất cả các cổng được kích hoạt lại như cổng đầu vào. Trong nhiều thiết kế thì chân này được nối đất để không kích hoạt nó hoặc nó cũng có thể để hở.

– Các chân A0, A1, $\overline{CS}$: Chân $\overline{CS}$ tích cực ở mức thấp. Khi $\overline{CS}$ được chọn thì nó cùng với A1, A0 chọn các cổng riêng biệt. Các chân này dùng để truy cập các cổng A, B, C hoặc các thanh ghi điều khiển theo bảng dưới đây (bảng 4.3).

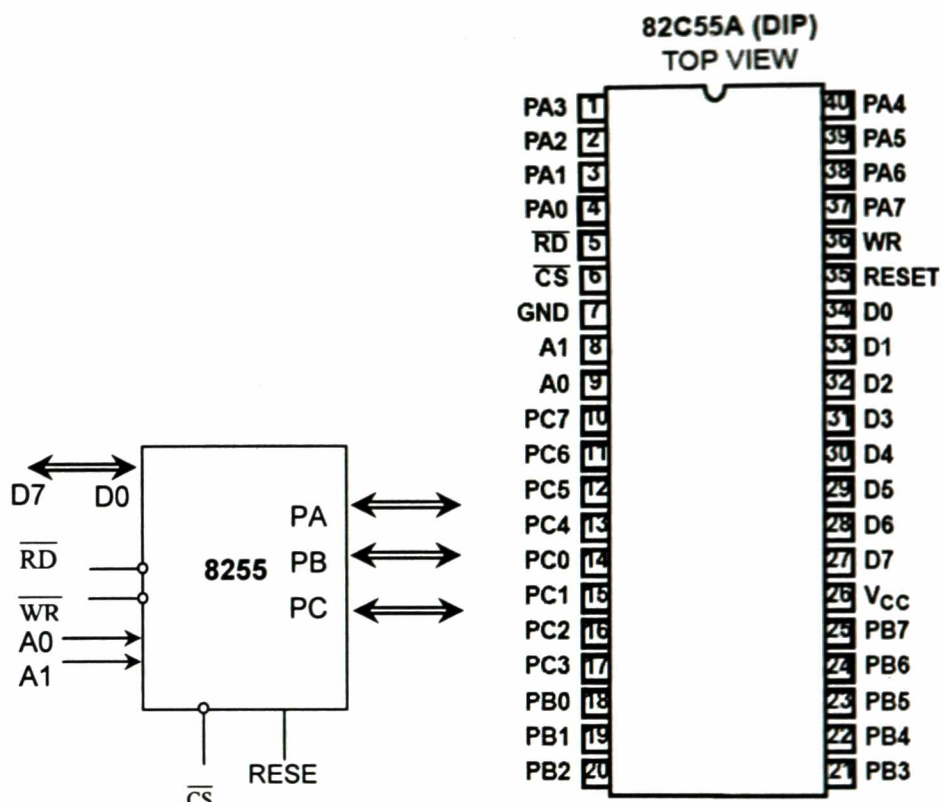
Sau đây là bảng phối hợp các chân $\overline{CS}$, A1, A2, $\overline{RD}$ và $\overline{WR}$ để tạo ra chức năng các cổng và thanh ghi điều khiển của 8255A.

Bảng 4.3. Phối hợp $\overline{CS}$ và $A_1 A_2$ chọn cổng

$\overline{CS}$	A_1	A_0	Chọn cổng
0	0	0	Cổng A
0	0	1	Cổng B
0	1	0	Cổng C
0	1	1	Thanh ghi điều khiển
1	x	x	8255 không được chọn

Bảng 4.4. Các lệnh chọn cổng và thanh ghi của 8255

$\overline{CS}$	A_1	A_0	$\overline{RD}$	$\overline{WR}$	Chức năng
0	0	0	0	1	Cổng A $\rightarrow$ Bus dữ liệu
0	0	1	0	1	Cổng B $\rightarrow$ Bus dữ liệu
0	1	0	0	1	Cổng C $\rightarrow$ Bus dữ liệu
0	0	0	1	0	Bus dữ liệu $\rightarrow$ Cổng A
0	0	1	1	0	Bus dữ liệu $\rightarrow$ Cổng B
0	1	0	1	0	Bus dữ liệu $\rightarrow$ Cổng C
0	1	1	1	0	Ghi vào từ điều khiển
1	x	x	x	x	Đường dẫn dữ liệu ở trạng thái điện trở cao
0	x	x	1	1	Đường dẫn dữ liệu ở trạng thái điện trở cao



Hình 4.7. Sơ đồ chân của 8255A

4.2.2.2. Chọn chế độ của 8255A

Trong khi các cổng A, B, C được dùng để xuất/nhập dữ liệu thì thanh ghi điều khiển phải được lập trình để chọn chế độ làm việc cho các cổng này. Các cổng của 8255A có thể được lập trình theo một trong các chế độ dưới đây.

❖ Chế độ 0: Vào/ra thông thường

- Các cửa A, B, C được làm việc độc lập nhau.
- Các cửa A, B, C có thể là cửa vào hoặc ra tùy theo chế độ trong thanh ghi điều khiển (ra số liệu được chốt, vào không chốt).

- Không có sự đối thoại giữa vi xử lý với thiết bị ngoại vi. Nếu muốn có tín hiệu đối thoại phải dùng các bit của một cửa nào đó (thường là cửa C) bằng cách xác lập từng bit PCi. Khi đó, cửa C được xem như là hai cửa 4 bit với khả năng lập/xóa từng bit.

❖ **Chế độ 1: Chốt vào/ra**

Chia làm hai nhóm.

- Nhóm A: Gồm cửa A để trao đổi số liệu và nửa C cao (PC4 - PC7) để đối thoại với vi xử lý và thiết bị ngoại vi của cửa A.
- Nhóm B: Gồm cửa B để trao đổi số liệu và nửa C thấp (PC0 - PC3) để đối thoại với vi xử lý và thiết bị ngoại vi của cửa B.

❖ **Chế độ 2: Bus hai chiều**

Chỉ dùng cho cửa A với số liệu vào/ra 2 chiều tạo thành một Bus chứa:

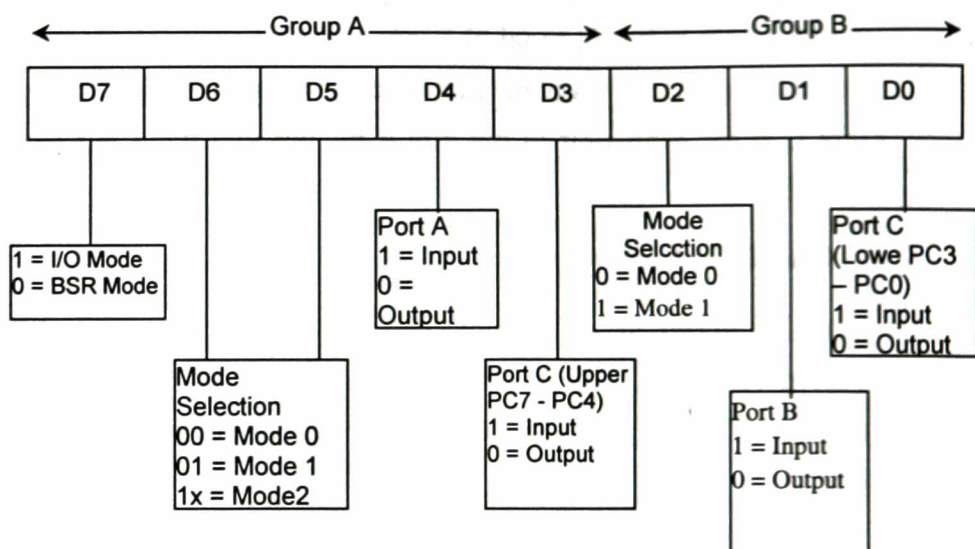
- Cửa A (Bus hai chiều).
- 5 đường điều khiển (PC3 – PC7).
- Logic điều khiển.
- Vào và ra đều chốt.

Việc chọn chế độ làm việc cho 8255A được thực hiện nhờ thanh ghi từ điều khiển. Đối với 8255A có hai loại từ điều khiển: Từ điều khiển cấu hình của các cổng (hình 4.8) và từ điều khiển việc lập/xóa các bit ở đầu ra PC (hình 4.9).

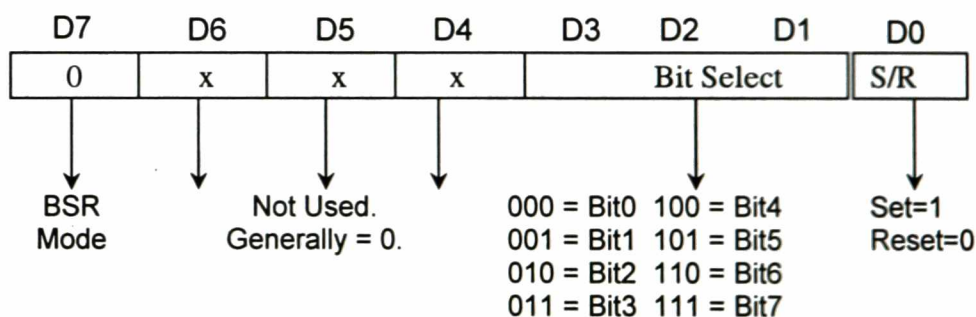
Ví dụ 1: Hãy tìm từ điều khiển cho 8255A cho các cấu hình sau:

Tất cả các cổng A, B, C đều là các cổng ra (chế độ 0).

PA là đầu vào, PB là đầu ra, PCL bằng đầu vào và PCH là đầu ra.



Hình 4.8. Từ điều khiển cấu hình của 8255A



Hình 4.9. Từ điều khiển lập/xoá các bit PCi của 8255A

Giải:

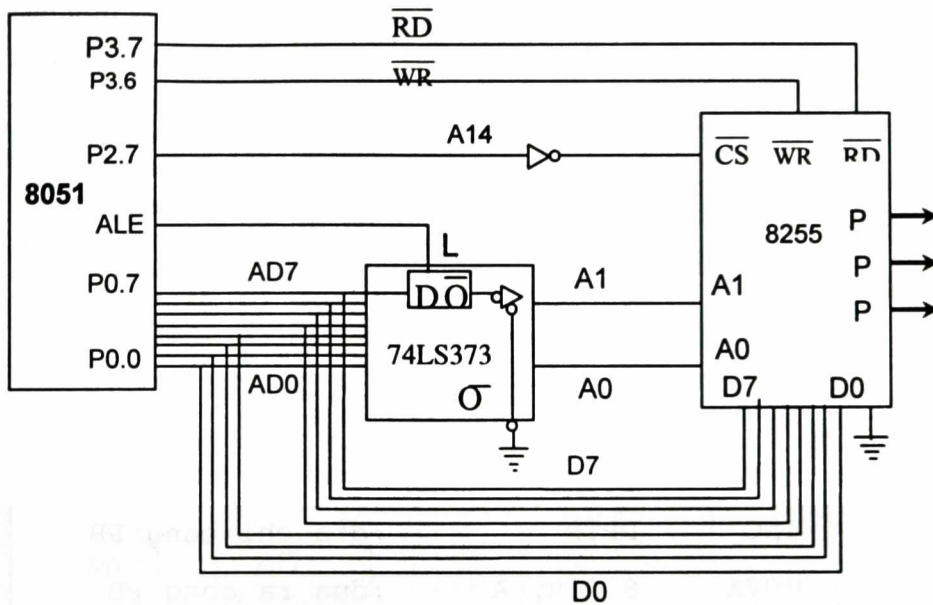
Từ hình 4.8 ta có:

- Trường hợp 1 có từ điều khiển là: 1000 0000 = 80H
- Trường hợp 2 có từ điều khiển là: 1001 0001 = 91H

Ví dụ 2: Cho sơ đồ ghép nối 8081 với 8255A (hình 4.10) hãy:

- Tìm các địa chỉ vào/ra được gán cho các cổng A, B, C và thanh ghi điều khiển.

- b) Hãy lập trình 8255A cho các cổng A, B, C thành các cổng ra.
- c) viết một chương trình để gửi 55H và AAH đến cổng ra một cách liên tục.



Hình 4.10. Nối ghép 8051 với 8255A cho ví dụ 2

Giải:

- a) Giả sử trong sơ đồ trên ta sử dụng 16 bit địa chỉ (A0 – A15), ta sẽ có địa chỉ cơ sở dành cho 8255A như sau:

Bảng 4.5

A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	Địa chỉ
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=4000H (PA)
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	=4001H (PB)
0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	=4002H (PC)
0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	=4003H (CR)

- b) Từ điều khiển cho 8255A cho cấu hình: Tất cả các cổng A, B, C đều là các cổng ra (chế độ 0) là 80H (đã trình bày ở ví dụ 1).
- c) Viết chương trình để gửi 55H và AAH đến cổng ra một cách liên tục.

```

MOV     A, #80H           ;tu dieu khien
MOV     DPTR, #4003H      ;nap d/c cong cua
                           ;thanh ghi dieu khien

MOVX    @DPTR, A          ;xuat tu dieu khien
MOV     A, #55H           ;gan A = 55
AGAIN:  MOV     DPTR, #4000H ;dia chi cong PA
MOVX    @DPTR, A          ;dua ra cong PA
INC     DPTR              ;dia chi cong PB
MOVX    @DPTR, A          ;dua ra cong PB
INC     DPTR              ;dia chi cong PC
MOVX    @DPTR, A          ;dua ra cong PC
CPL     A                 ;dao gtri cua A (A=AAH)
ACALL   DELAY             ;tao tre
SJMP    AGAIN             ;tiếp tục

```

Ví dụ 3: Cho sơ đồ ghép nối 8051 với 8255A như hình 4.11, hãy:

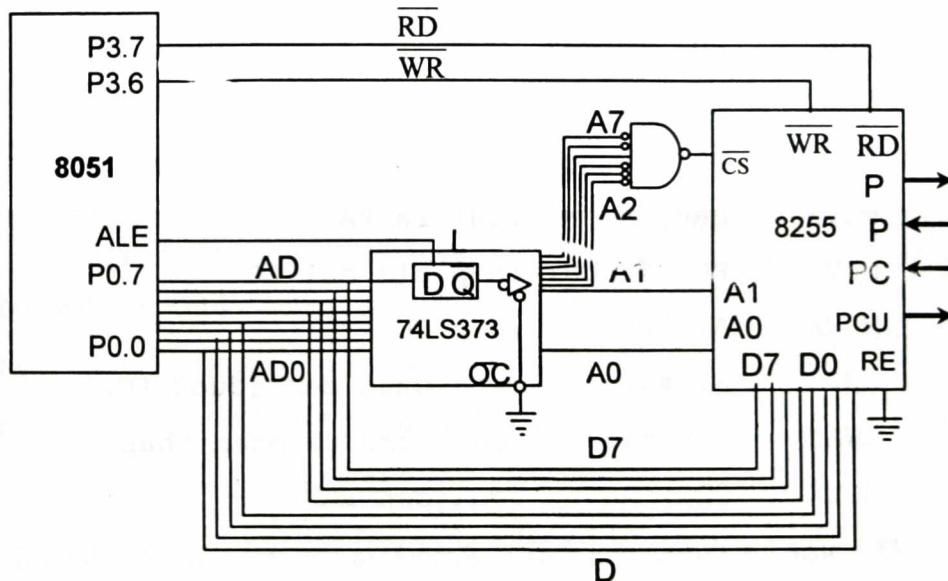
- a) Tìm địa chỉ cổng vào/ra được gán cho các cổng A, B, C và thanh ghi điều khiển.
- b) Tìm từ điều khiển cho trường hợp PA là đầu ra, PB là đầu vào, PC0 – PC3 là đầu vào, PC4 – PC7 là đầu ra.
- c) Viết chương trình để nhận dữ liệu từ PB gửi ra PA, nhận dữ liệu từ PCL gửi ra PCH.

Giải:

a) Từ sơ đồ ghép nối ta được bảng 4.6:

Bảng 4.6

A7	A6	A5	A4	A3	A2	A1	A0	Đ/c cổng, CR
0	0	1	0	0	0	0	0	= 20H (PA)
0	0	1	0	0	0	0	1	= 21H (PB)
0	0	1	0	0	0	1	0	= 22H (PC)
0	0	1	0	0	0	1	1	= 23H (CR)



Hình 4.11. Nối ghép 8051 với 8255A cho ví dụ 3

- b) Từ điều khiển cho trường hợp PA là đầu ra, PB là đầu vào, PC0 – PC3 là đầu vào, PC4 – PC7 là đầu ra là: 1000 0011B = 83H.
- c) Viết chương trình để nhận dữ liệu từ PB gửi ra PA, nhận dữ liệu từ PCL gửi ra PCH.

```
CR      EQU    83H    ; thanh ghi tu dieu khien
APORT   EQU    20H    ; dia chi cong A
```



```

BPORT    EQU        21H    ;dia chi cong B
CPORT    EQU        22H    ;dia chi cong C
CRPORT    EQU        23H    ;dia chi t.ghi d.khien
...

MOV       A, #CR          ;PA, PCH la cong vao, PB,
                           ;PCL la cong ra

MOV       R0, #CRPORT;nap dia chi cong thanh ghi
                           ;dieu khien

MOVX      @R0, A          ;xuat tu dieu khien

MOV       R0, #BPORT ;nap dia chi PB

MOVX      A, @R0          ;doc PB

DEC       R0              ;chi den PA (R0 = 20H)

MOVX      @R0, A          ;gui ra PA

MOV       R0, #CPORT ;nap dia chi PC

MOVX      A, @R0          ;doc PC

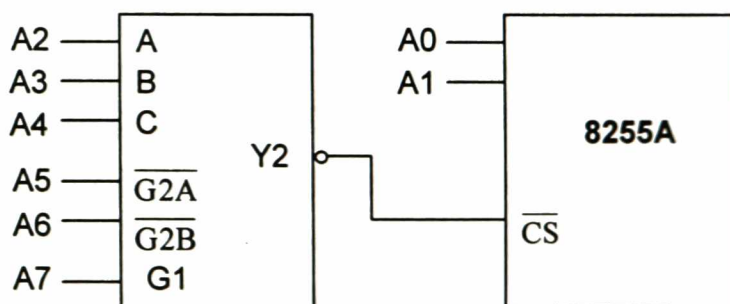
ANL       A, #0FH         ;che phan cao (duoc PCL)

SWAP      A ;trao doi phan cao va phan thap

MOVX      @R0, A          ;gui den PCH

```

Đối với hệ thống cần nhiều 8255A ta sử dụng mạch giải mã 74LS138 để giải mã như sơ đồ hình 4.12:

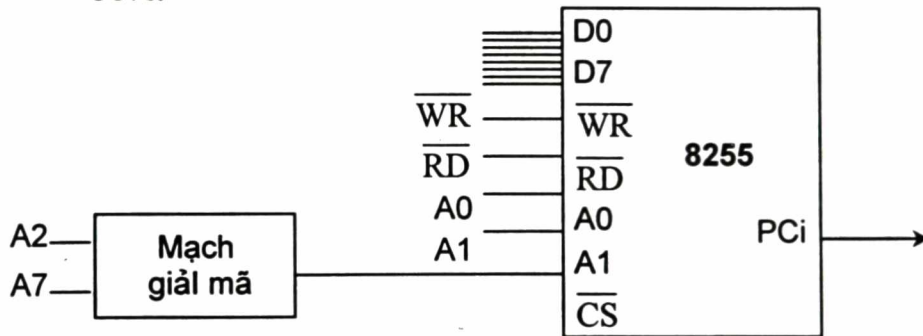


Hình 4.12. Sử dụng 74LS138 để ghép nối nhiều vi mạch 8255A

Ví dụ 4: Cho sơ đồ hình 4.13, hãy lập trình cho 8255A để:

a) Đặt PC1 lên cao.

b) Sử dụng PC2 để tạo xung vuông liên tục với độ dày xung là 50%.



Hình 4.13. Hình vẽ mô tả cho ví dụ 4

Giải:

a) Đặt PC1 lên cao: Từ hình vẽ 4.13 ta tìm ra được từ điều khiển ở chế độ lập bit PC1 như sau: 0000 0011B

MOV R0, #CRPORT ; nạp địa chỉ thanh ghi điều khiển

MOV A, #00000011; chọn PC1 = 1

MOVX @R0, A ; đặt PC1 lên cao (PC1 = 1)

b) Sử dụng PC2 để tạo xung vuông liên tục với độ dày xung là 50%

AGAIN: MOV R0, #CRPORT ; nạp địa chỉ cổng
; thanh ghi điều khiển

MOV A, #00000101 ; chọn PC2 = 1

MOVX @R0, A ; đặt PC2 lên cao

ACALL DELAY ; thời gian giữ chân

MOV A, #00000100 ; chọn PC2 = 0

MOVX @R0, A ; đặt PC2 xuống thấp

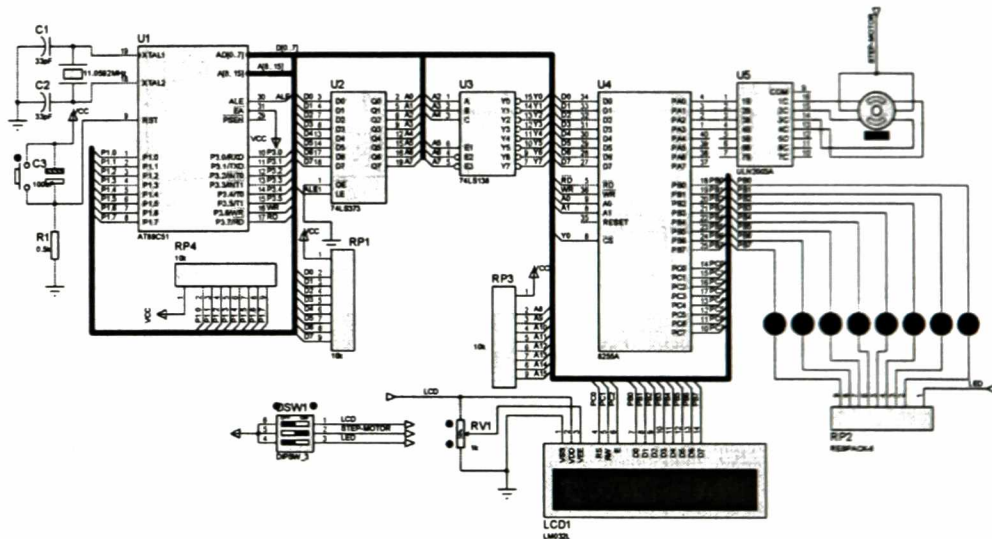
```

ACALL    DELAY    ;thoi gian giu cham
SJMP     AGAIN    ;lap lai

```

Ví dụ 5: Hãy sử dụng mạch giải mã 74LS138 để giải mã chọn chip 8255A với địa chỉ cơ sở của cổng là 20h. Viết chương trình để điều khiển các 8 LED nối trên cổng PB (từ PB0 – PB7) sáng theo quy luật từ phải qua trái.

Giải:



Hình 4.14. Sơ đồ kết nối 8051 với 8255A thông qua mạch chốt 74LS373 và giải mã 74LS138

```

ORG 0000H

CR      EQU 80H;PA,PB,PC LA CONG RA
BPORT   EQU 21H;DIA CHI CONG B
CRPORT  EQU 23H;DIA CHI THANH GHI DIEU KHIEN

MOV R0,#CRPORT
MOV A,#CR
MOVX @R0,A

SANGLED:

```

```

MOV R0, #BPORT
MOV A, #0FFH
LAP: CLR C
MOVX @R0, A
ACALL DELAY
RLC A
JC LAP
SJMP SANGLED

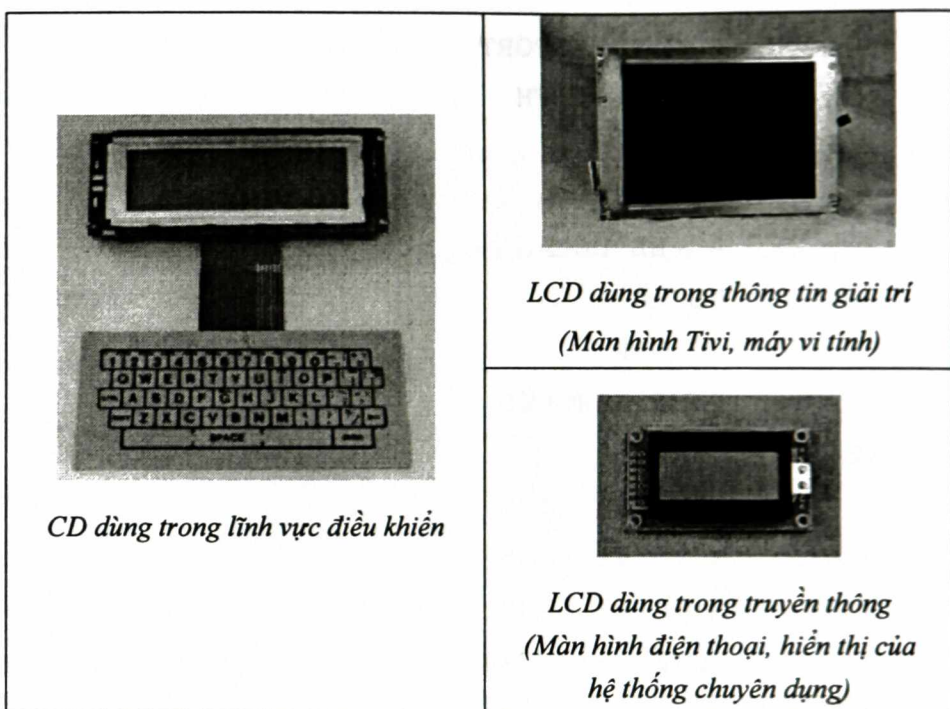
DELAY:
MOV R3, #100
HERE2: MOV R4, #255
HERE: DJNZ R4, HERE
      DJNZ R3, HERE2
RET
END.

```

4.3. PHỐI GHÉP VI ĐIỀU KHIỂN VỚI THỂ GIỚI THỰC 1: PHỐI GHÉP 8051 VỚI MÀN HÌNH TINH THỂ LỎNG LCD, LED 7 THANH, CHUYỂN ĐỔI TƯƠNG TỰ – SỐ ADC VÀ CẢM BIẾN NHIỆT

4.3.1. PHỐI GHÉP VI ĐIỀU KHIỂN VỚI MÀN HÌNH TINH THỂ LỎNG LCD

Ngày nay trong lĩnh vực thông tin và giải trí, việc dùng màn hình ống tia ca tốt truyền thống (CRT) đang dần được thay thế bằng việc sử dụng màn hình tinh thể lỏng (LCD - Liquid Crystal Display). Vì LCD có nhiều ưu điểm vượt trội như: Độ dày màn hình nhỏ hơn rất nhiều, kích thước đa dạng từ loại màn hình nhỏ tới màn hình cực lớn, tiêu thụ ít năng lượng và không nguy hiểm bằng CRT.



Hình 4.15

Ở phần này chúng ta chỉ xét tới LCD loại nhỏ và việc dùng LCD để hiển thị của hệ thống chuyên dụng và đi nghiên cứu cách ghép nối LCD với 8051 và lập trình.

4.3.1.1. Mô tả chân của LCD

LCD có tổng số 14 chân chia làm 3 nhóm (hình 4.16):

Nhóm 1: (3 chân)

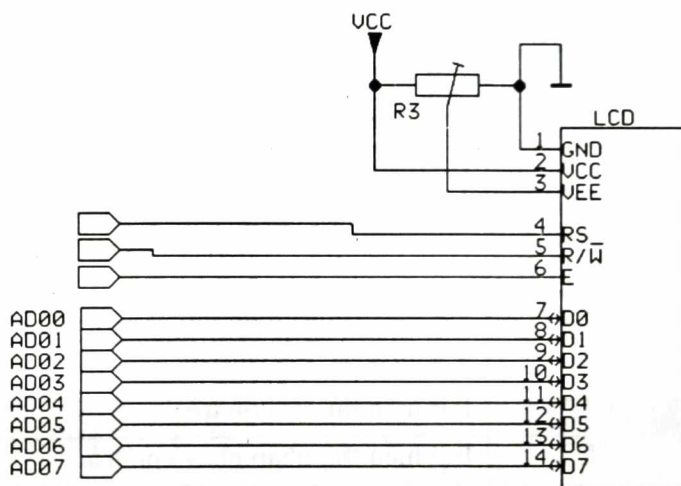
– VDD, VSS: cấp 5V, 0V

– VEE: thay đổi điện áp để thay đổi độ tương phản.

Nhóm 2: (8 chân) Vào ra thông tin với vi điều khiển: Từ chân D0-D7

Nhóm 3: (3 chân) Điều khiển việc vào ra thông tin: E, RS, R/W

- E (Enable) – Chân cho phép: được sử dụng để chốt thông tin hiển thị có trên chân dữ liệu. Khi dữ liệu được cấp đến chân dữ liệu thì một xung từ mức cao xuống mức thấp được áp đến chân E để LCD chốt dữ liệu trên chân dữ liệu. Xung này phải rộng tối thiểu 450ns.



Hình 4.16. Mô tả chân của LCD

- RS (loại thông tin trao đổi): Thông tin trao đổi là lệnh điều khiển hay là dữ liệu để hiển thị. Nếu RS = 0 thì thanh ghi mã lệnh được chọn, cho phép người dùng gửi một lệnh chẳng hạn như xoá màn hình, đưa con trỏ về đầu dòng,... Nếu RS = 1 thì thanh ghi dữ liệu được chọn và cho phép người dùng gửi dữ liệu cần hiển thị lên LCD.
- R/W (Read/Write) – Chân đọc/ghi: cho phép người dùng đọc/ghi thông tin từ/lên LCD. Nếu R/W = 1 thì đọc dữ liệu, nếu R/W = 0 thì ghi dữ liệu.

4.3.1.2. Bảng lệnh của LCD

Cũng có các mã lệnh được gửi đến LCD để xoá màn hình hoặc đưa con trỏ về đầu dòng hoặc nhấp nháy con trỏ,... Bảng 4.7.

Bảng 4.7. Mã lệnh LCD

Mã (Hex)	Lệnh đến thanh ghi của LCD
1	Xoá màn hình hiển thị
2	Trở về đầu dòng
4	Dịch con trỏ sang trái
6	Dịch con trỏ sang phải
5	Dịch hiển thị sang phải
7	Dịch hiển thị sang trái
8	Tắt con trỏ, tắt hiển thị
A	Tắt hiển thị, bật con trỏ
C	Bật hiển thị, tắt con trỏ
E	Bật hiển thị, nhấp nháy con trỏ
F	Tắt con trỏ, nhấp nháy con trỏ
10	Dịch vị trí con trỏ sang trái
14	Dịch vị trí con trỏ sang phải
18	Dịch toàn bộ hiển thị sang trái
1C	Dịch toàn bộ hiển thị sang phải
80	Đưa con trỏ về đầu dòng thứ nhất
C0	Đưa con trỏ về đầu dòng thứ hai
38	Hai dòng ma trận 5x7

4.3.1.3. Lập trình LCD

Gửi có trễ lệnh và gửi dữ liệu đến LCD

Để gửi bất kỳ một lệnh nào ở bảng 4.7 đến LCD, cần đưa chân RS = 0, còn để gửi dữ liệu thì đặt RS = 1. Sau đó, gửi một sườn xung cao xuống thấp đến chân E để cho phép chốt dữ liệu trong LCD. Với

phương pháp này ta cần lưu ý rằng luôn phải đặt một độ trễ lớn trong quá trình xuất dữ liệu hoặc lệnh ra LCD.

Ví dụ 1: Ghép nối 8051 với LCD như hình 4.17, hãy lập trình thực hiện công việc sau:

- Ghi lệnh chọn LCD 2 dòng, ma trận 5x7.
- Hiển thị màn hình và con trỏ.
- Xoá LCD.
- Dịch con trỏ sang phải.
- Hiển thị liên tục dòng chữ “BM KY THUAT VDK...!”

Giải:

; Chân P1.0-P1.7 được nối tới chân dữ liệu D0-D7 của ;LCD.

; Chân P2.0 được nối tới chân RS của LCD.

; Chân P2.1 được nối tới chân R/W của LCD.

; Chân P2.2 được nối đến chân E của LCD.

ORG 0000H

```
LAP:      MOV      A, #38H ;LCD 2 dong, ma tran 5x7
          ACALL    COMMAND ;truyen lenh den LCD
          ACALL    DELAY   ;tao tre cho LCD
          MOV      A, #0EH ;hien thi man hinh, con tro
          ACALL    COMMAND ;truyen lenh den LCD
          ACALL    DELAY   ;tao tre cho LCD
          MOV      A, #01H ;xoa LCD
          ACALL    COMMAND ;truyen lenh den LCD
          ACALL    DELAY   ;tao tre cho LCD
```

```

MOV      A, #06H ;dich con tro sang phai
ACALL    COMMAND ;truyen lenh den LCD
MOV      DPTR, #DATA
MOV      R2, #19
HIEN:    CLR A
MOV      A, @A+DPTR
ACALL    DISPLAY
ACALL    DELAY
INC      DPTR
DJNZ     R2, HIEN ;kiem tra xem da het chuoi
           ;ky tu?
SJMP     LAP
COMMAND:                                ;ctc truyen lenh den LCD
MOV      P1, A ;sao thanh ghi A den cong P1
CLR      P2.0 ;RS = 0 de gui lenh
CLR      P2.1 ;R/W = 0 de ghi
SETB     P2.2 ;dat E = 1 cho xung cao
CLR      p2.2
RET                                             ;ve chuong trinh chinh
DISPLAY:                                ;ctc hien thi
MOV      P1, A ;sao thanh ghi A den cong P1
SETB     P2.0 ;RS = 1 de gui du lieu
CLR      P2.1 ;R/W = 0 de ghi
SETB     P2.2 ;dat E = 1 cho xung cao
CLR      p2.2
RET                                             ;ve chuong trinh chinh

```

```

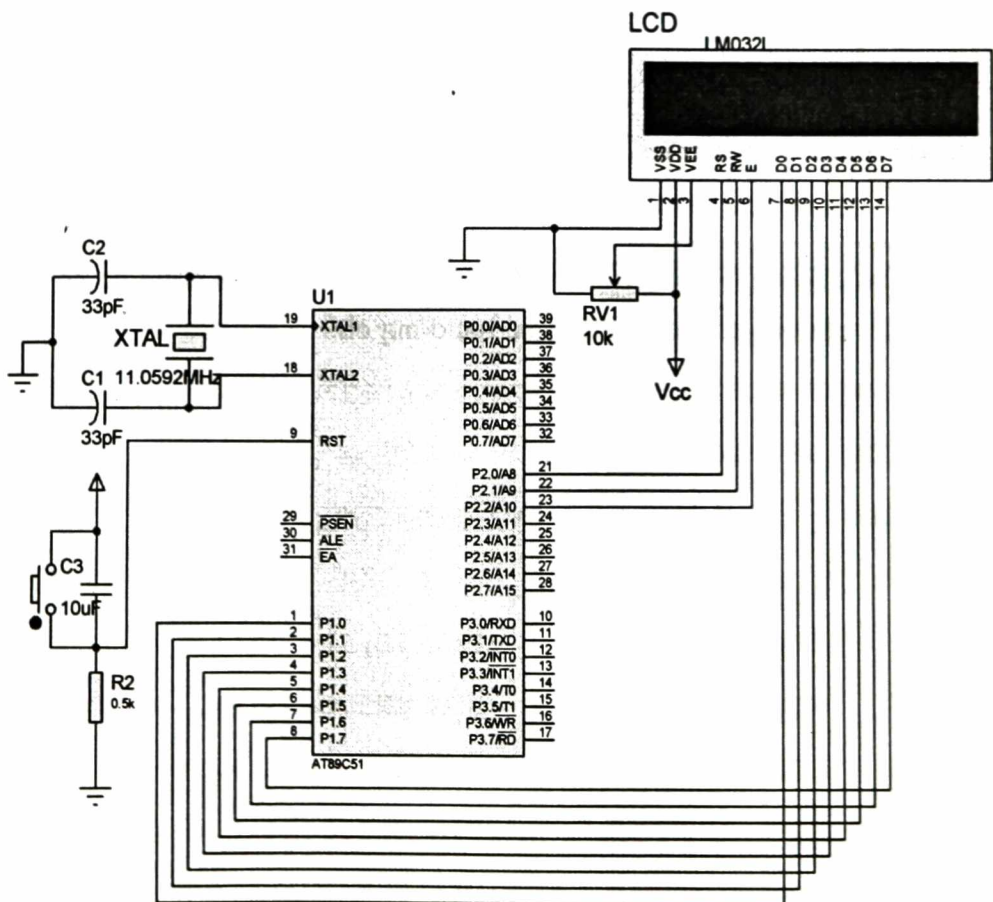
DELAY:  MOV     R3, #100 ;tao thoi gian tre
HERE2:  MOV     R4, #255 ;dat R4 = 255
HERE:   DJNZ    R4, HERE ;R4 = 0
        DJNZ    R3, HERE2
        RET

DATA:

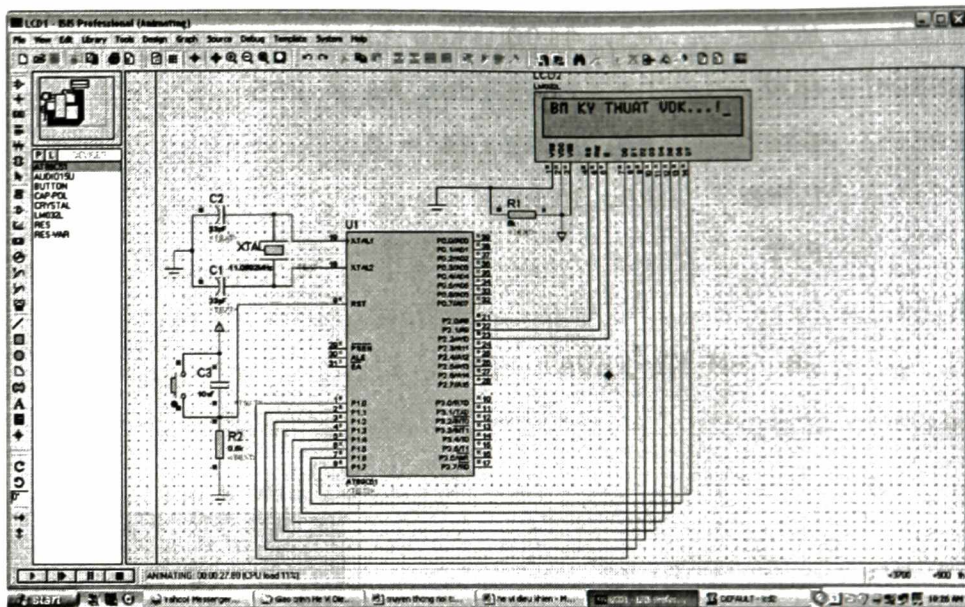
        DB 'BM KY THUAT VDK...!'

END

```



Hình 4.17. Hình vẽ ví dụ 1



Hình 4.18. Kết quả mô phỏng trên phần mềm Proteus

Ví dụ 2: Hãy ghép nối 8255A với 8051 với địa chỉ cổng cơ sở là 20h. Viết chương trình hiển thị liên tục dòng chữ .

“ GHEP NOI 8255A...!

HIEN THI LCD LM032L.”

Giải:

ORG 0000H

CR EQU 80H ;PA, PB, PC LA CONG RA

APORT EQU 20H ;DIA CHI CONG A

BPORT EQU 21H ;DIA CHI CONG B

CPORT EQU 22H ;DIA CHI CONG C

RPORT EQU 23H ;DIA CHI THANH GHI DIEU

KHIEN

MOV R0, #CRPORT

```

        MOV      A, #CR
        MOVX     @R0, A

START:  MOV A, #38H      ;LCD 2 dong, ma tran 5x7
        ACALL    COMMAND ;truyen lenh den LCD
        ACALL    DELAY   ;tao tre cho LCD
        MOV      A, #0EH ;hien thi man hinh, con tro
        ACALL    COMMAND;truyen lenh den LCD
        ACALL    DELAY   ;tao tre cho LCD
        MOV      A, #01H ;xoa LCD
        ACALL    COMMAND;truyen lenh den LCD
        ACALL    DELAY   ;tao tre cho LCD
        MOV      A, #06H ;dich con tro sang phai
        ACALL    COMMAND;truyen lenh den LCD
        MOV      DPTR, #DATA1

HIEN:   CLR      A
        MOVC     A, @A+DPTR
        MOV      R5, A
        ACALL    DISPLAY
        ACALL    DELAY
        MOV      A, R5
        INC      DPTR
        CJNE     A, #'!', HIEN ;kiem tra xem da
                                het chuoi;ky tu?

```

```

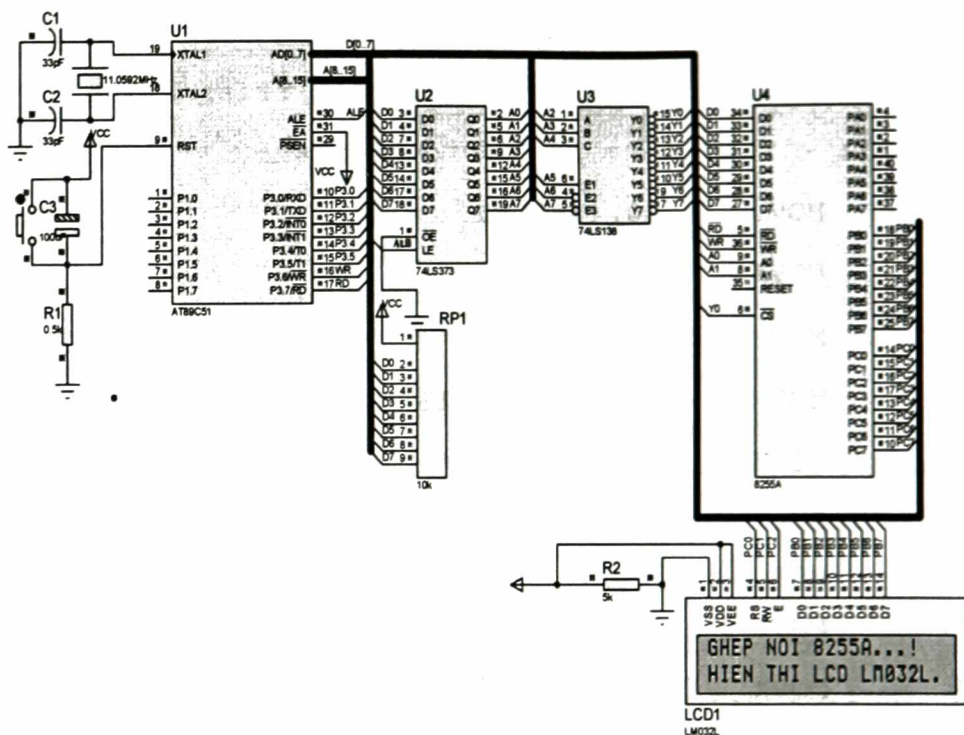
        MOV     A, #0C0H      ;xuống đầu dòng
        ACALL   COMMAND
        ACALL   DELAY
        MOV     DPTR, #DATA2
HIEN1:   CLR     A
        MOVC    A, @A+DPTR
        MOV     R5, A
        ACALL   DISPLAY
        ACALL   DELAY
        MOV     A, R5
        INC     DPTR
        CJNE    A, #'.', HIEN1 ;kiểm tra xem đã
                                   hết chuỗi;ky tu?
        SJMP    START
COMMAND: ;ctc truyền lệnh đến LCD
        MOV     R0, #BPORT
        MOVX    @R0, A
        MOV     R0, #CPORT
        MOV     A, #00000100B
        MOVX    @R0, A
        NOP
        NOP
        MOV     A, #00000000B

```

```

MOVX    @R0, A
RET                                ;ve chuong trinh chinh
DISPLAY:                                ;ctc hien thi
MOV      R0, #BPORT
MOVX     @R0, A
MOV      R0, #CPORT
MOV      A, #00000101B
MOVX     @R0, A
NOP
NOP
MOV      A, #00000001B
MOVX     @R0, A;ve chuong trinh chinh
DELAY:   MOV      R3, #100 ;tao thoi gian tre
HERE2:   MOV      R4, #255 ;dat R4 = 255
HERE:    DJNZ     R4, HERE ;R4 = 0
DJNZ     R3, HERE2
RET
-----;Khai bao vung du lieu can hien thi len LCD
DATA1:
DB       'GHEP NOI 8255A...!'
DATA2:
DB       'HIEN THI LCD LM032L.'
END

```



Hình 4.19. Ghép nối 8051 với 8255A và LCD LM032L, mô phỏng kết quả trên Proteus

4.3.2. PHỐI GHÉP VI ĐIỀU KHIỂN VỚI LED 7 THANH

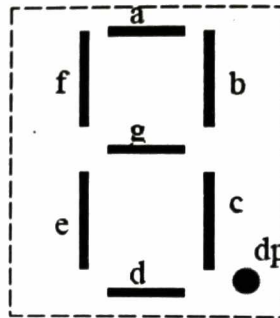
4.3.2.1. Cơ sở lý thuyết

LED 7 thanh được chia làm 2 loại là: Anot chung và Katot chung:

Đối với dạng Led Anot chung, chân COM phải có mức logic 1 và muốn sáng Led thì tương ứng các chân a - f, dp sẽ ở mức logic 0.

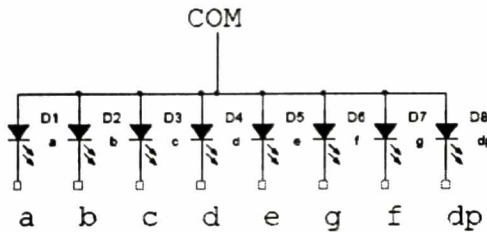
Đối với dạng Led Katot chung, chân COM phải có mức logic 0 và muốn sáng Led thì tương ứng các chân a - f, dp sẽ ở mức logic 1

Sơ đồ nguyên lý của LED 7 thanh được trình bày như hình 4.20a:

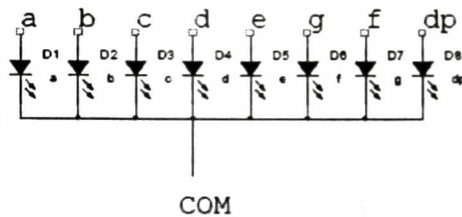


Hình 4.20a. Sơ đồ nguyên lý của LED 7 thanh

Sơ đồ triển khai dạng như sau:



Hình 4.20b. Loại Anot chung



Hình 4.20c. Loại Katot chung

Bảng mã như sau:

**Bảng 4.8. Bảng mã cho
Led Anot chung**

Số	dp	g	f	e	d	c	b	a	Mã hex
0	1	1	0	0	0	0	0	0	0C0h
1	1	1	1	1	1	0	0	1	0F9h
2	1	0	1	0	0	1	0	0	0A4h
3	1	0	1	1	0	0	0	0	0B0h
4	1	0	0	1	1	0	0	1	99h
5	1	0	0	1	0	0	1	0	92h
6	1	0	0	0	0	0	1	0	82h
7	1	1	1	1	1	0	0	0	0F8h
8	1	0	0	0	0	0	0	0	80h
9	1	0	0	1	0	0	0	0	90h

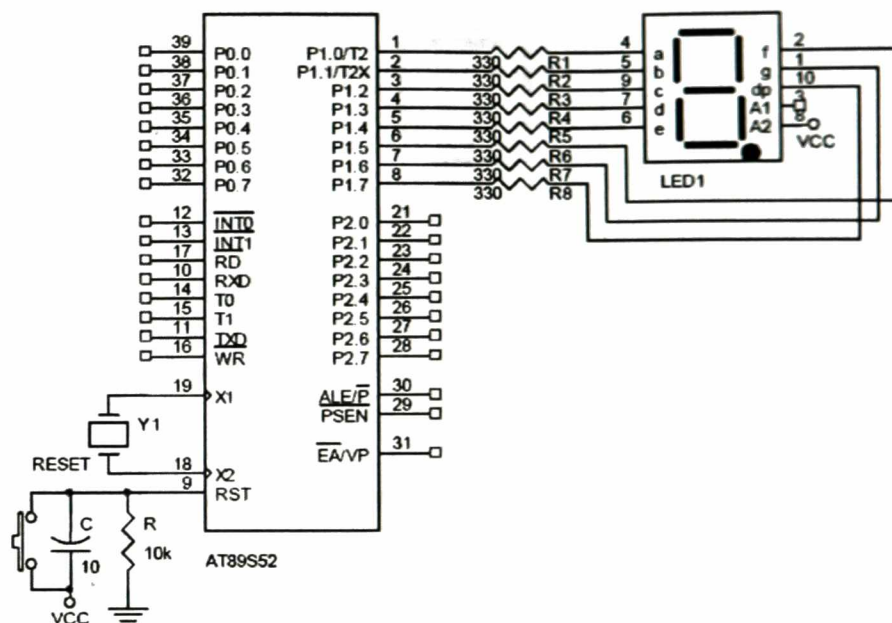
**Bảng 4.9. Bảng mã cho
Led Katot chung**

Số	dp	g	f	e	d	c	b	a	Mã hex
0	0	0	1	1	1	1	1	1	3Fh
1	0	0	0	0	0	1	1	0	06h
2	0	1	0	1	1	0	1	1	5Bh
3	0	1	0	0	1	1	1	1	4Fh
4	0	1	1	0	0	1	1	0	66h
5	0	1	1	0	1	1	0	1	6Dh
6	0	1	1	1	1	1	0	1	7Dh
7	0	0	0	0	0	1	1	1	07h
8	0	1	1	1	1	1	1	1	7Fh
9	0	1	1	0	1	1	1	1	6Fh

4.3.2.2. Giao tiếp

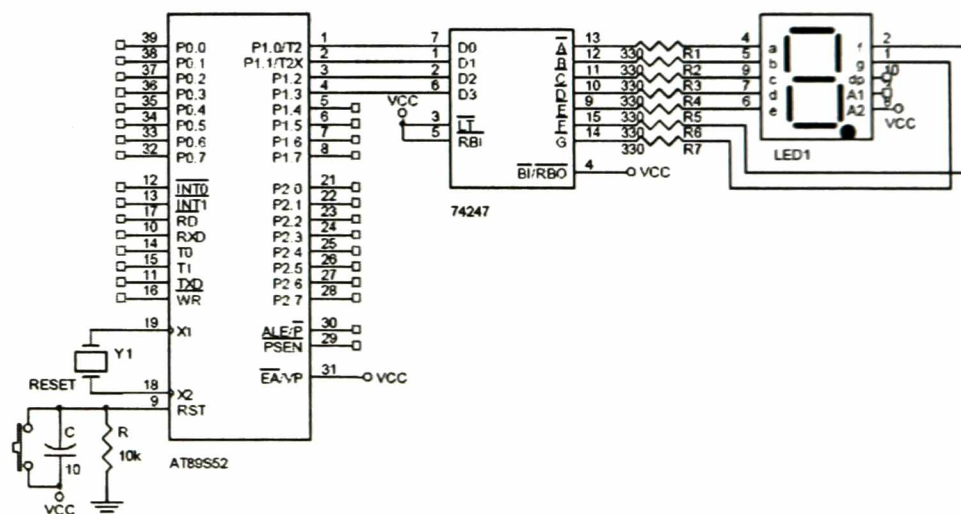
a) Giao tiếp 1 led 7 đoạn

– Giao tiếp trực tiếp: kiểu này sẽ dùng 1 port (hình 4.21).



Hình 4.21. Giao tiếp trực tiếp vi điều khiển với 1 led đoạn

– Giao tiếp gián tiếp: kiểu này chỉ dùng 4 đường kết nối với IC giải mã led 7 đoạn như hình 4.22.



Hình 4.22. Giao tiếp gián tiếp vi điều khiển với LED 7 thanh

Ví dụ 1: Hãy dùng sơ đồ hình 4.21, viết chương trình điều khiển 1 led 7 đoạn đếm lên từ 0 đến 9.

```

;-----
;-----CHUONG TRINH DEM LEN TU 0
DEN 9 HIEN THI TREN 1 LED
;-----
                DEM EQU R2
                LED EQU P1
ORG 0000H                                ;BAT DAU CHUONG TRINH
                MOV DPTR,#MA7DOAN ;DC BANG MA LED 7 DOAN
MAIN0: MOV DEM,#00H                        ;DEM=00
MAIN1  MOV A,DEM
                MOVC A,@A+DPTR    ;LAY MA 7 DOAN
                MOV LED,A          ;RA LED HIEN THI
                LCALL DELAY
                INC DEM            ;TANG GIA TRI DEM
                CJNE DEM,#10,MAIN1 ;SS DEM VOI 10
                LJMP MAIN0         ;LAM LAI TU DAU
;-----
;-----CHUONG TRINH CON DELAY
;-----
DELAY:  MOV R5,##0FFH
LOOP1:  MOV R6,#0FFH
                DJNZ R6,$
                DJNZ R5,LOOP1
RET
;-----
;-----KHAi BAO MA 7 DOAN TU SO '0' DEN SO '9'
;-----
MA7DOAN:
                DB 0C0H,0F9H,0A4H,0B0H,99H,92H,82H,0F8H,80H,90H
END

```

Trong ví dụ trên chúng ta cần quan tâm đến biến có tên “dem” với giá trị ban đầu bằng 00h, được giải mã sang mã 7 đoạn và gửi ra led để hiển thị, tiếp theo là thực hiện chương trình con delay 1 giây, sau đó tăng giá trị đếm lên 1 đơn vị và kiểm tra xem nếu chưa bằng 10 thì quay về giải mã và hiển thị số mới, nếu bằng 10 thì xoá về 00 và bắt đầu lại chu kỳ mới.

Ví dụ 2: Hãy dùng sơ đồ hình 4.23, viết chương trình điều khiển 2 led 7 đoạn đếm lên từ 00 đến 99.

ORG 0H

START:

MOV R1, #00H

LAP: MOV A, R1

ACALL CHUYENBCD

MOV P2, A

ACALL DELAY

INC R1

CJNE R1, #100, LAP

CHUYENBCD:

MOV B, #10

DIV AB

SWAP A

ADD A, B

RET

DELAY:

MOV TMOD, #10H ;SU DUNG TIMER 1 CHE DO 1

MOV R3, #25 ;TAO TRE 10s

AGAIN: MOV TL1, #0FEH ;TL1 = FEH, BYTE THAP

MOV TH1, #6FH ;TH1 = 6FH, BYTE CAO

SETB TR1 ;KHOI DONG TIMER 1

BACK: JNB TF1, BACK ;KIEM TRA CO BO DINH THOI

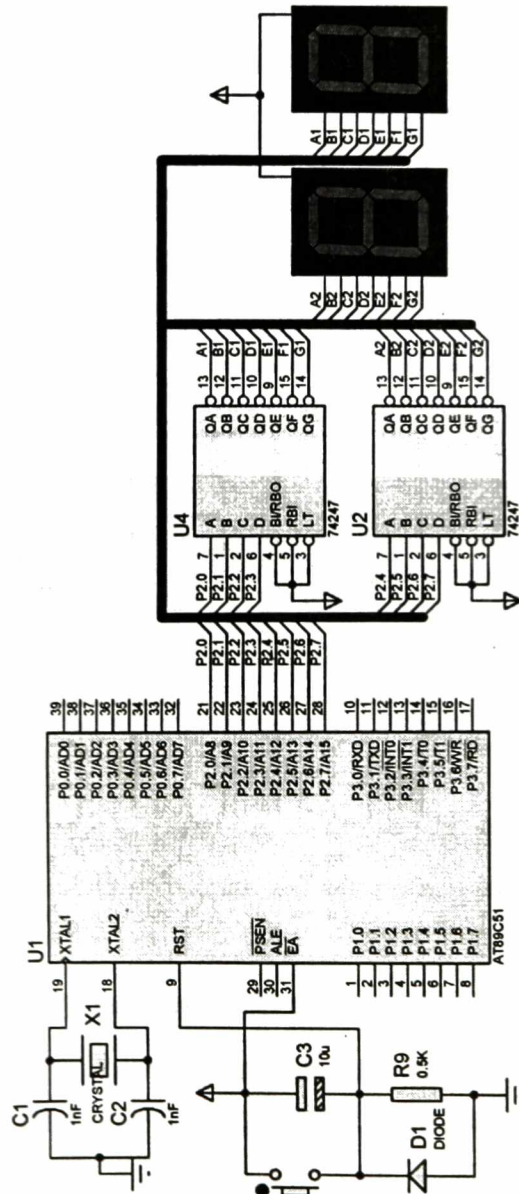
CLR TR1 ;TAT TIMER 1

```

CLR TF1          ;XOA CO TIMER 1
DJNZ R3,AGAIN    ;NEU R3 KHAC 0 THI LAP LAI
RET

```

END.



Hình 4.23

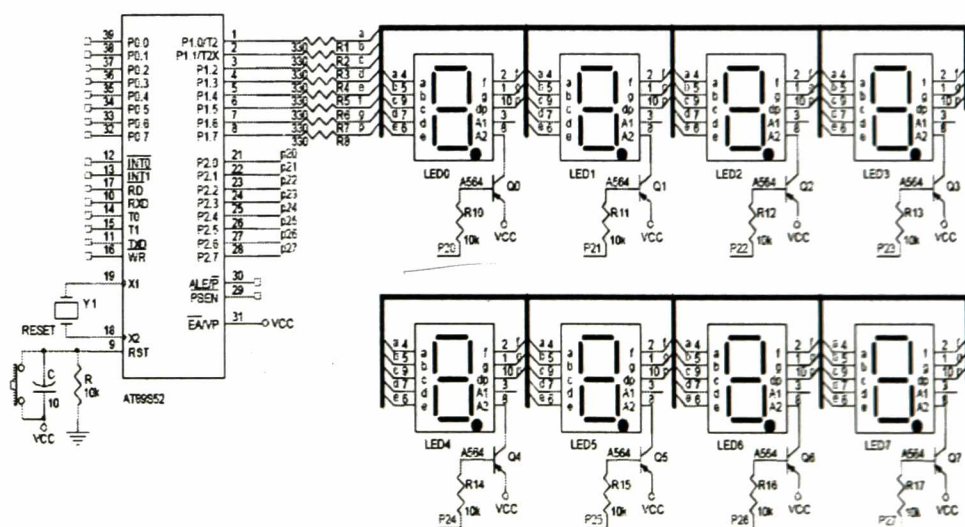
b) Giao tiếp nhiều led 7 đoạn

* Sử dụng phương pháp quét

Để làm quen với cách thức giao tiếp điều khiển nhiều led 7 đoạn thì nên kết nối theo phương pháp quét. Sơ đồ nguyên lý của led 7 đoạn hình 4.24.

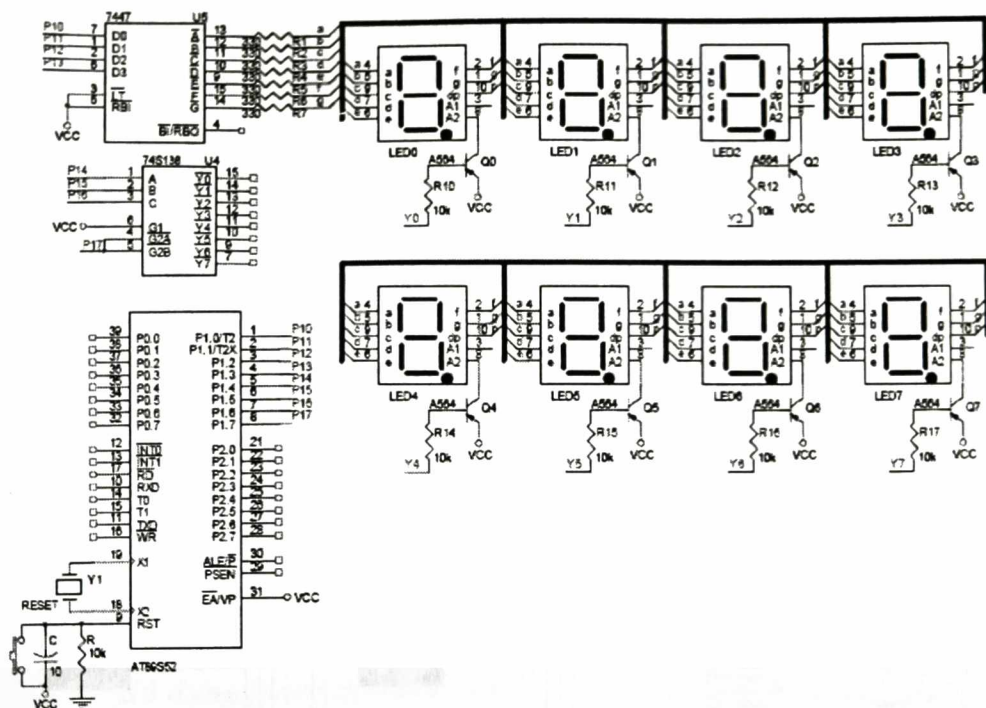
Tại mỗi một thời điểm ta chỉ cho 1 transistor dẫn và 7 transistor còn lại tắt, dữ liệu gửi ra sẽ sáng trên led tương ứng với transistor dẫn. Sau đó, cho 1 transistor khác dẫn và gửi dữ liệu hiển thị cho led đó, quá trình điều khiển này diễn ra lần lượt cho đến khi hết 8 led.

Với tốc độ gửi dữ liệu nhanh và do mắt ta có lưu ảnh nên ta nhìn thấy 8 led sáng cùng 1 lúc.



Hình 4.24. Giao tiếp vi điều khiển với 8 led 7 đoạn

Trong sơ đồ hình 4.24 ta có thể giảm bớt số lượng đường điều khiển bằng cách dùng thêm IC số đóng vai trò quét và giải mã như hình 4.25.



Hình 4.25. Giao tiếp vi điều khiển với 8 led 7 đoạn chỉ dùng 1 port 8 đường

Trong sơ đồ hình 4.25 ta dùng IC 7447 đóng vai trò giải mã số BCD sang mã 7 đoạn, IC 74138 có chức năng giải mã 3 đường sang 8 đường điều khiển 8 transistor. Số lượng đường điều khiển cần dùng là 7 đường.

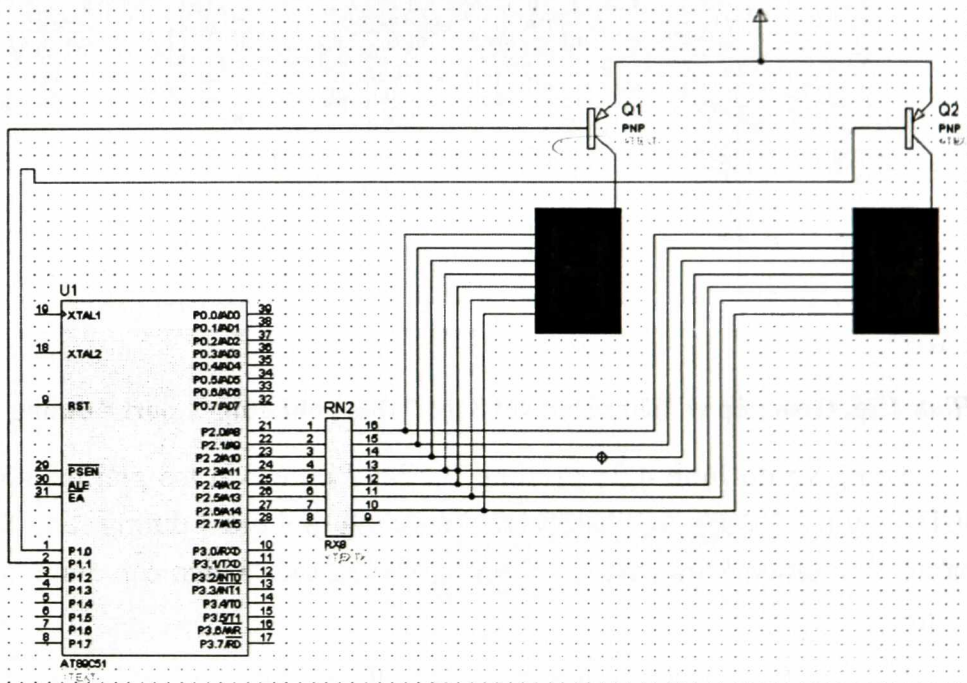
Trong 7 đường điều khiển của port1 thì vi điều khiển sẽ dùng 4 bit thấp để gửi mã BCD ra IC giải mã điều khiển led 7 đoạn, 3 bit tiếp theo dùng để điều khiển IC giải mã cho phép 1 transistor dẫn. Đường tín hiệu P1.7 điều khiển chân cho phép của IC 74138: khi muốn cho phép hiển thị thì P1.7 phải ở mức 0, khi cấm thì P1.7 phải ở mức H.

Trong phương pháp quét sử dụng 8 led thì thời gian được phép sáng của 1 led bằng $1/8$ chu kỳ quét, thời gian tắt bằng $7/8$ chu kỳ quét. Do thời gian led tắt khá dài so với thời gian led sáng nên phải quét nhanh thì ta mới nhìn thấy tất cả các led đều sáng.

Với led đơn sáng 1/2 chu kỳ và tắt 1/2 chu kỳ thì tần số để mắt ta nhìn thấy led sáng liên tục (sáng luôn do mắt lưu ảnh) đo được trong thực tế phải $> 40\text{Hz}$.

Với 8 led 7 đoạn dùng phương pháp quét thì tần số quét đo được trong thực tế phải lớn hơn $> 60\text{Hz}$.

Ví dụ: Thực hiện đoạn chương trình sau để hiển thị số 26 (hình 4.26) ra 2 Led 7 đoạn:



Hình 4.26. Sử dụng phương pháp quét LED

org 0h

main:

```
MOV     P2, #82h           ; Mã của số 6
CLR     P1.0               ; Hiện số
CALL    Delay
SETB    P1.0
MOV     P2, #0A4H          ; Mã của số 2
```

```

CLR      P1.1
CALL     Delay
SETB     P1.1
SJMP     main

```

Delay:

```

PUSH     07H
MOV       R7, #100
DJNZ     R7, $
POP       07H
RET

```

END

**** Sử dụng phương pháp chốt***

Khi thực hiện tách riêng các đường dữ liệu của Led, ta có thể cho phép các Led sáng đồng thời mà sẽ không có hiện tượng ảnh hưởng giữa các Led. IC chốt cho phép lưu trữ dữ liệu cho các Led có thể sử dụng là 74LS373, 74LS374.

Ví dụ: Sử dụng sơ đồ hình 4.27. Viết chương trình để hiển thị số tăng dần từ 00 đến 99 ra 2 Led 7 đoạn .

ORG 0H

main:

```
MOV R0, #0 ;R0 chứa giá trị xuất ra LED
```

lap:

```
MOV A, R0
```

```
MOV B, #10; A chứa hàng chục, B chứa hàng đơn vị
```

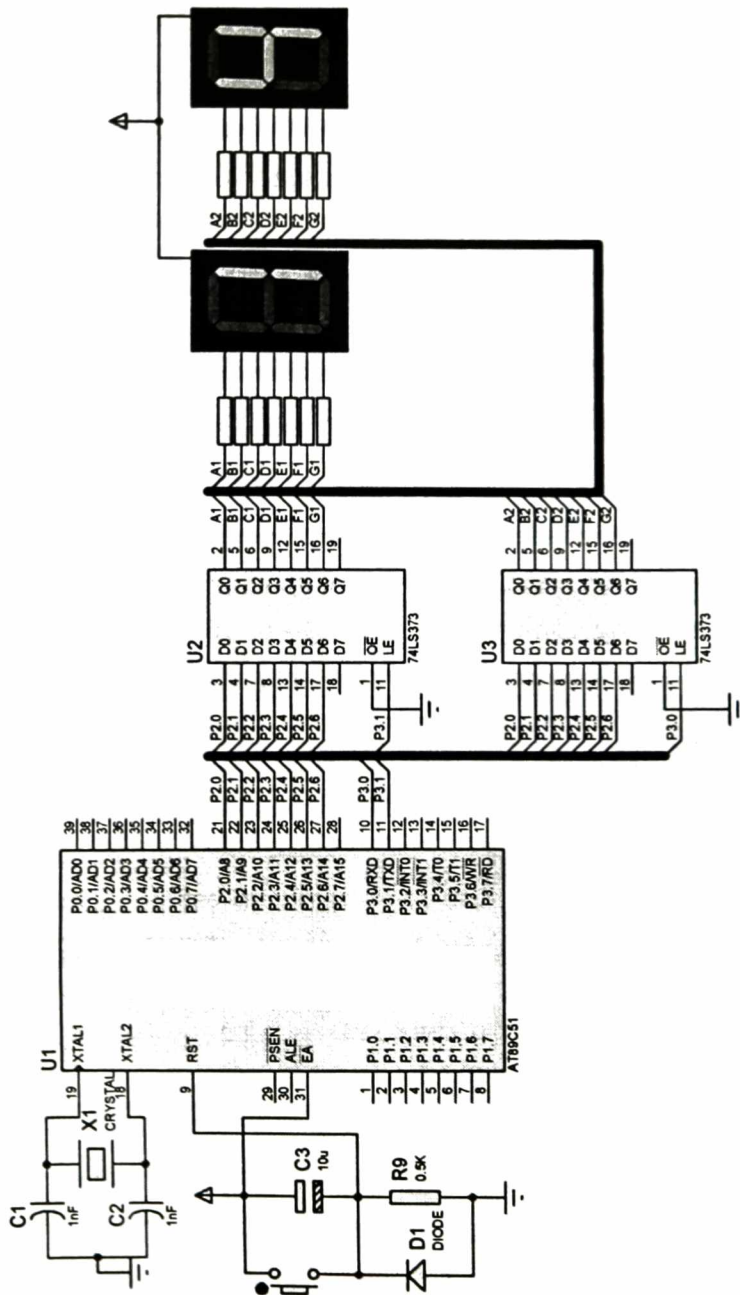
```
DIV AB
```

```

MOV DPTR,#Maled7
MOVC A,@A+DPTR    ;Lấy mã LED 7 đoạn
MOV P2,A
SETB P3.1          ;Xuất số hàng chục
CLR P3.1
MOV A,B
MOVC A,@A+DPTR
MOV P2,A
SETB P3.0          ;Xuất số hàng đơn vị
CLR P3.0
CALL Delay
INC R0              ;Tăng R0
CJNE R0,#100,lap   ;So sánh với 100, nếu #0
                    thì tiếp tục
SJMP main          ;bằng 100 thì giảm về 0
;-----
Maled7:
    DB 0C0h,0F9h,0A4h,0B0h,99h,92h,82h,0F8h,80h,90h
;-----
Delay:
    PUSH 07
    PUSH 06
    MOV R6,#255
Delay1:
    MOV R7,#255
    DJNZ R7,$
    DJNZ R6,Delay1
    POP 06
    POP 07
    RET

```

END.



Hình 4.27. Sử dụng phương pháp chốt

4.3.3. GHÉP NỐI ADC VỚI 8051 VÀ LẬP TRÌNH

Các bộ chuyển đổi ADC được sử dụng rất rộng rãi. Lý do là máy tính làm việc trên các giá trị nhị phân. Tuy nhiên trong thực tế, các đại lượng vật lý đều ở dạng tương tự (liên tục). Ví dụ: nhiệt độ, áp suất, lưu lượng,... là các đại lượng vật lý được chuyển về dòng điện hoặc điện áp (tương tự) qua một thiết bị gọi là bộ biến đổi. Do vậy, cần một bộ chuyển đổi tương tự số để bộ vi điều khiển có thể đọc được chúng. Chip ADC được sử dụng rộng rãi hiện nay đó là ADC0804.

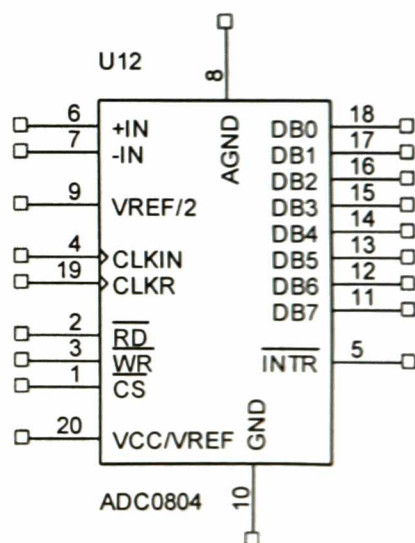
4.3.3.1. Giới thiệu về ADC0804

ADC0804 là bộ chuyển đổi tương tự – số thuộc họ ADC0800 của hãng National Semiconductor. Chip có điện áp nuôi +5V và độ phân giải 8 bit. Ngoài độ phân giải thì thời gian chuyển đổi cũng là một thông số quan trọng đánh giá bộ ADC, đó là thời gian cần thiết để bộ ADC chuyển đầu vào tương tự thành số nhị phân. Đối với ADC0804 thời gian chuyển đổi phụ thuộc vào tần số đồng hồ cấp tới chân CLK và CLK IN và không bé hơn $110\mu s$. Hình 4.29 mô tả ADC ở chế độ chạy tự do.

Các chân của ADC801 có chức năng như sau:

– CS (Chip Select) - Chọn chip: Đầu vào tích cực mức thấp được sử dụng để kích hoạt chip ADC0804.

– RD (Read) – Đọc: Tín hiệu vào tích cực mức thấp. Khi CS = 0 và nếu có một xung từ cao xuống thấp áp đến chân RD thì dữ liệu ra dạng số 8 bit được đưa tới các chân dữ liệu D0 – D7. Chân RD còn được gọi là chân cho phép đầu ra.



Hình 4.28

– WR (Write) – Ghi: Chân này dùng để báo cho ADC0804 bắt đầu quá trình chuyển đổi. Nếu CS = 0 khi WR tạo ra xung thấp lên cao thì bộ ADC0804 bắt đầu tiến hành chuyển đổi giá trị đầu vào tương tự Vin về số nhị phân 8 bit. Lượng thời gian cần thiết để chuyển đổi thay đổi phụ thuộc vào tần số đưa đến chân CLK IN và CLK R. Khi việc chuyển đổi dữ liệu được hoàn tất thì chân INTR được ADC0804 hạ xuống thấp.

– CLK IN và CLK R: CLK IN là chân vào nối tới đồng hồ ngoài khi đồng hồ ngoài được sử dụng để tạo thời gian. Tuy nhiên, ADC0804 cũng có một bộ tạo xung đồng hồ trên chip. Để dùng đồng hồ trong thì các chân CLK IN và CLK R được nối tới một tụ điện và một điện trở (hình 4.29). Tần số đồng hồ được xác định: $f = \frac{1}{1,1RC}$.

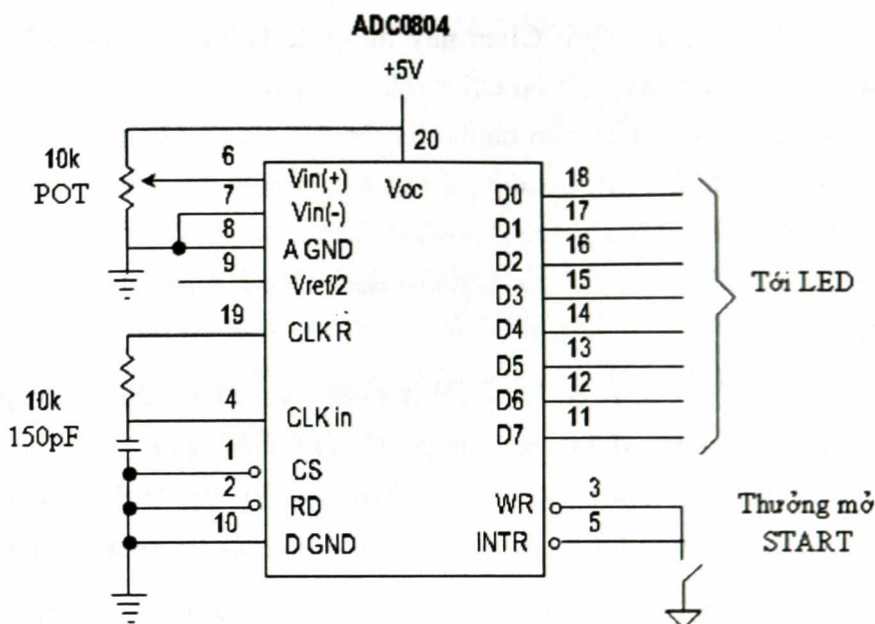
Giá trị thông thường của đại lượng trên là R = 10K, C = 150pF và tần số nhận được là f = 606 KHz, thời gian chuyển đổi sẽ là 110μs.

– INTR (Interrupt) – Ngắt: Còn gọi là kết thúc việc chuyển đổi. Đây là chân ra tích cực mức thấp. Bình thường chân này ở mức cao khi chuyển đổi hoàn tất thì nó chuyển xuống mức thấp để báo cho CPU biết dữ liệu được chuyển đổi đã sẵn sàng để lấy đi. Sau khi INTR xuống thấp cần đặt CS = 0 và gửi một xung cao xuống thấp tới chân RD để đưa dữ liệu ra.

– Vin (+) và Vin (-): Đây là hai đầu vào tương tự vi sai, trong đó

– Vin = Vin (+) – Vin (-). Thông thường Vin (-) được nối xuống đất và Vin (+) được dùng làm đầu vào tương tự và tín hiệu này sẽ được chuyển về dạng số.

– VCC: Là chân nguồn nuôi +5V. Chân này còn được dùng làm điện áp tham chiếu khi đầu vào Vref/2+ (chân 9) để hở.



Hình 4.29. Kiểm tra ADC ở chế độ chạy tự do

Kích thước bước là sự thay đổi nhỏ nhất mà ADC có thể phân biệt được.

– D0 – D7: Là các chân dữ liệu số. Các chân này được đệm 3 trạng thái và dữ liệu đã được chuyển đổi chỉ được truy cập khi chân CS = 0 và chân RD đưa xuống mức thấp. Để tính điện áp đầu ra có thể sử dụng công thức sau:

$$Dout = \frac{Vin}{\text{kích thước bước}}$$

– Dout: Là đầu ra dữ liệu số

– Vin: Điện áp đầu vào tương tự

– Vref/2: Là chân điện áp đầu vào được dùng làm điện áp tham chiếu. Nếu chân này hở (không được nối đất) thì điện áp đầu vào tương tự cho ADC0804 nằm trong dải 0 đến 5V (giống như Vcc). Tuy nhiên, có nhiều ứng dụng mà đầu vào tương tự áp đến Vin khác với

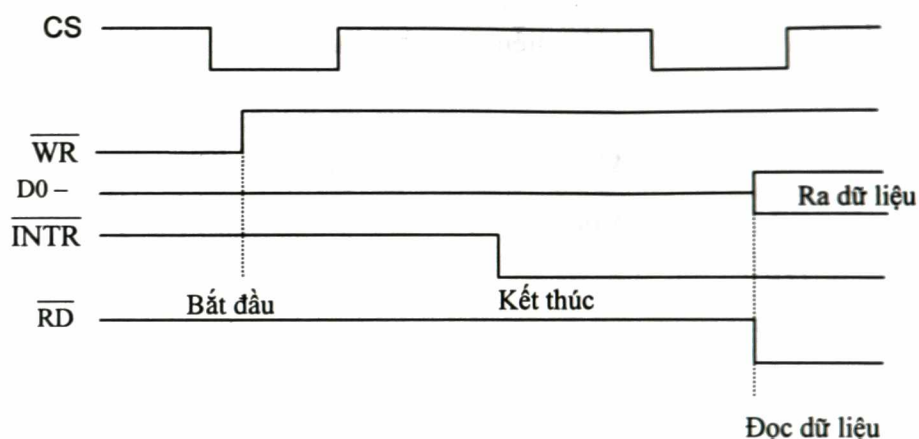
dải 0 đến 5V. Bảng 4.10 biểu diễn quan hệ dải điện áp V_{in} đối với các đầu vào $V_{ref}/2$ khác nhau:

Bảng 4.10. Quan hệ điện áp $V_{ref}/2$ với V_{in}

$V_{ref}/2$ (V)	V_{in} (V)	Kích thước bước (mV)
Hở	0 đến 5	$5/256 = 19.53$
2.0	0 đến 4	$4/256 = 15.62$
1.5	0 đến 3	$3/256 = 11.71$
1.28	0 đến 2.56	$2.56/256 = 10$
1.0	0 đến 2	$2/256 = 7.81$
0.5	0 đến 1	$1/256 = 3.90$

Các bước khi ADC chuyển đổi dữ liệu là:

1. Bật $WR = 1$ để gửi một xung thấp lên cao để bắt đầu chuyển đổi.
2. Duy trì kiểm tra chân INTR. Nếu INTR xuống thấp thì việc chuyển đổi được hoàn tất và có thể chuyển sang bước tiếp theo.
3. Sau khi chân INTR xuống thấp, bật $CS = 0$ và gửi một xung cao xuống thấp đến chân RD để nhận dữ liệu từ chip ADC0804. Phân chia thời gian cho quá trình này được trình bày như sau (hình 4.30).

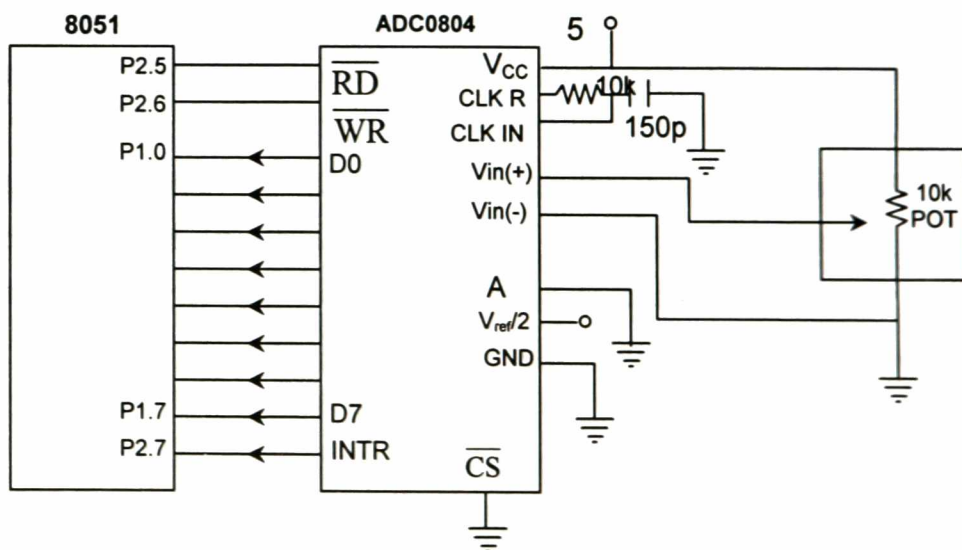


Hình 4.30. Phân chia thời gian đọc và ghi của ADC0804

4.3.3.2. Nối ghép ADC0804 với 8051 và lập trình

Sơ đồ ghép nối ADC0804 với 8051 như hình 4.31 sử dụng nguồn đồng hồ sẵn có trong 8051.

Từ sơ đồ ghép nối hãy: Viết chương trình kiểm tra chân $\overline{INTR}$ và chuyển tín hiệu đầu vào tương tự vào thanh ghi A.



Hình 4.31. Nối ghép ADC0804 với 8051 sử dụng nguồn đồng hồ sẵn có

Giải:

; Đặt P2.6=WR (bắt đầu chuyển đổi cần 1 xung thấp lên cao)

; Đặt chân P2.7=0 khi kết thúc chuyển đổi

; Đặt P2.5=RD (xung cao - xuống - thấp sẽ đọc dữ liệu từ ADC)

; P1.0 - P1.7 nối tới chân của ADC0804

ORG 0h

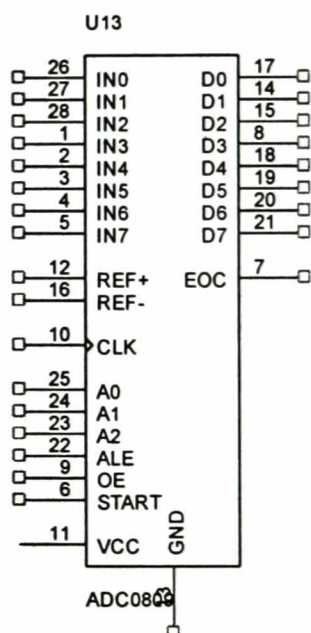
```
                SETB    p2.7      ;P2.7 làm công vào
                MOV     P1, #0FFH;chọn P1 làm công vào
BACK:           CLR     P2.6      ;đặt WR = 0
                SETB    P2.6      ;đặt WR = 1 để chuyển đổi
HERE:           JB      P2.7, HERE;kiểm tra INTR
                CLR     P2.5      ;kết thúc chuyển đổi cho
                                phép đọc
                MOV     A, P1      ;chuyển dữ liệu đọc được
                                ;vào thanh ghi A
                SETB    P2.5      ;đặt RD = 1 để đọc lần sau
                SJMP    BACK
```

END

Một chip rất hữu ích khác của National Semiconductor là ADC808/809 (hình 4.32). Chip này có đến 8 đầu vào và chúng cũng có đầu ra dữ liệu 8 bit như ADC0804. 8 kênh đầu vào tương tự được dồn kênh và được chọn như bảng 4.11 nhờ 3 chân địa chỉ A, B, C.

Đối với ADC0808/0809, các điện áp $V_{ref}(+)$ và $V_{ref}(-)$ thiết lập điện áp tham chiếu. Nếu $V_{ref}(-) = 0$ và $V_{ref}(+) = +5V$ thì độ phân giải là $5/256 = 19,53mV$. Muốn có độ phân giải $= 10mV$ ta đặt

$V_{ref}(+) = 2,56$ và $V_{ref}(-) = 0$. Các chân địa chỉ A, B, C được dùng để chọn kênh đầu vào IN0 – IN7 và kích hoạt chân ALE chốt địa chỉ. Chân SC dùng để bắt đầu chuyển đổi. Chân EOC dùng để kết thúc chuyển đổi và chân OE cho phép đọc đầu ra.



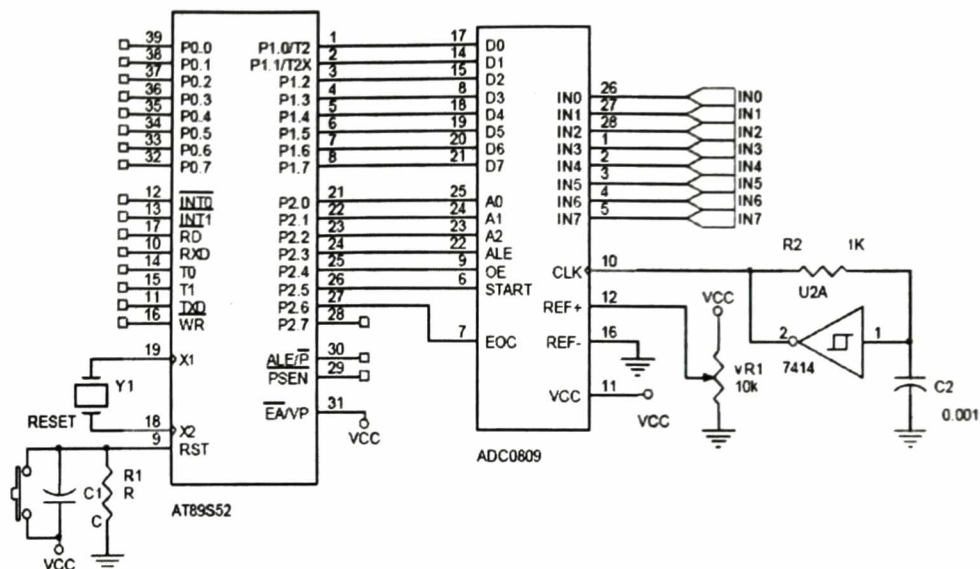
Bảng 4.11. Chọn kênh tương tự của ADC809

Chọn kênh tương tự	C	B	A
IN0	0	0	0
IN1	0	0	1
IN2	0	1	0
IN3	0	1	1
IN4	1	0	0
IN5	1	0	1
IN6	1	1	0
IN7	1	1	1

Hình 4.32. Bộ biến đổi ADC808/809

ADC0808/0809 không có xung đồng hồ bên trong, do đó phải được cấp xung đồng hồ từ bên ngoài đến chân Clock.

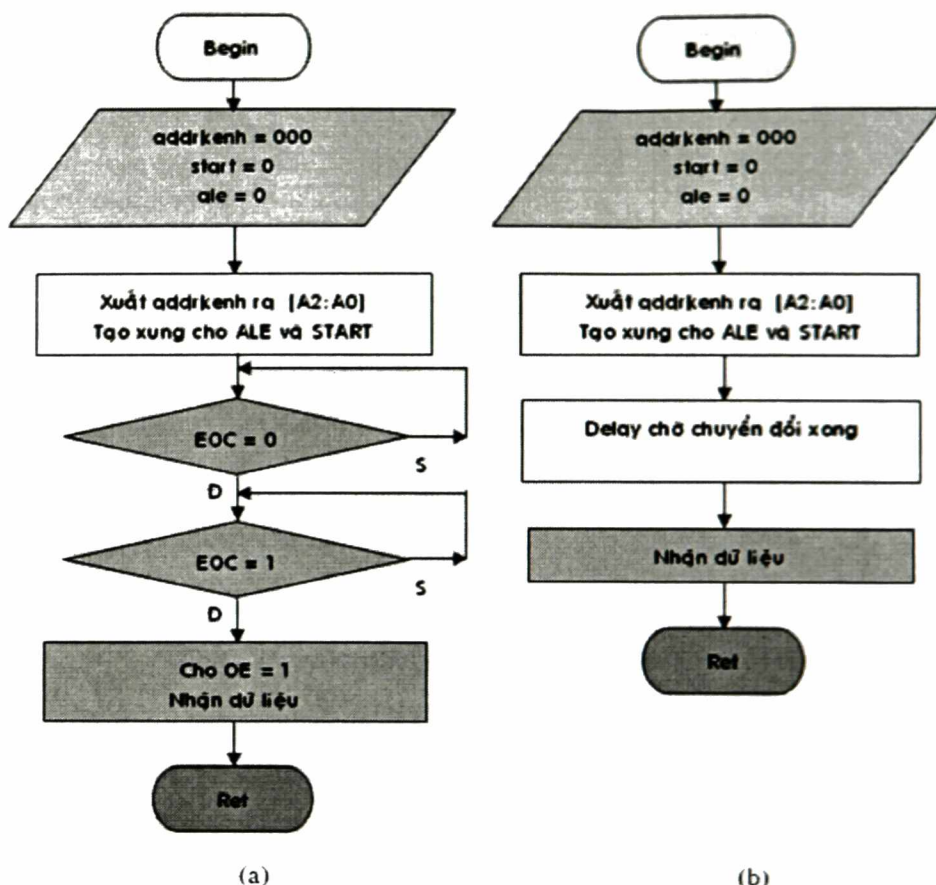
Trong hệ thống hình 4.33, vi điều khiển sử dụng 15 đường điều khiển để giao tiếp với ADC0809, trong đó có 8 đường dùng để nhận dữ liệu số sau khi chuyển đổi (D7:D0), 3 đường dùng để xuất địa chỉ chọn 1 trong 8 kênh (A2A1A0), 1 đường dùng để xuất tín hiệu chốt địa chỉ (ALE), 1 đường dùng để xuất tín hiệu điều khiển ADC0809 bắt đầu chuyển đổi (START), 1 đường dùng để xuất tín hiệu điều khiển cho phép xuất dữ liệu (OE), 1 đường dùng để nhận tín hiệu báo quá trình chuyển đổi kết thúc để tiến hành nhận dữ liệu.



Trong ứng dụng ta có thể dùng 1 đường điều khiển cả 2 tín hiệu ALE và START. Sau khi ra lệnh ADC thực hiện quá trình chuyển đổi thì vi điều khiển sẽ kiểm tra tín hiệu EOC xem chúng xuống mức thấp là báo hiệu quá trình chuyển đổi đang xảy ra, và chờ cho đến khi tín hiệu này lên mức cao trở lại thì quá trình chuyển đổi kết thúc mới tiến hành nhân dữ liệu.

Ta có thể không thực hiện theo cách kiểm tra vừa trình bày bằng cách chờ ADC chuyển đổi xong sau 1 khoảng thời gian tCONV lớn nhất là 116μs theo sổ tay tra cứu (datasheet) thì bắt đầu nhận dữ liệu. Khi đó, ta không cần phải dùng 1 đường tín hiệu kiểm tra chân EOC và chân này bỏ trống.

Lưu đồ điều khiển ADC 0809 chuyển đổi 1 kênh theo 2 cách vừa trình bày như hình 4.34:



Hình 4.34. Lưu đồ điều khiển ADC 0809

; -----Chương trình chuyển Analog sang Digital, kết quả cất vào thanh ghi A

```

ORG 0H
    INADC EQU P1      ;NHAP DU LIEU TU ADC
    CONTROL EQU P2    ;DIEU KHIEN ADC
    START BIT P2.3    ;START VÀ ALE BIT P2.3
START: MOV CONTROL, #00000000B ;CHUYEN RA PORT 2
    SETB START        ;START = 1
    NOP
    NOP
    CLR START         ;START = 0
  
```



```

        LCALL DELAY
        MOV A, INADC                ;DOC DU LIEU SAU
CHUYEN DOI
        SJMP START
DELAY:
        MOV R3, #52
HERE:   DJNZ R3, HERE
        RET
END

```

4.3.4. NỐI GHÉP 8051 VỚI CẢM BIẾN NHIỆT

4.3.4.1. Giới thiệu chung về cảm biến nhiệt

Các bộ chuyển đổi các đại lượng vật lý như nhiệt độ, áp suất, lưu lượng, cường độ ánh sáng,... thành tín hiệu điện được gọi là cảm biến. Ví dụ, nhiệt độ được biến đổi thành các tín hiệu điện sử dụng các bộ biến đổi, bộ biến đổi đó được gọi là cảm biến nhiệt. Loại cảm biến đơn giản và hay dùng bao gồm các loại họ LM34 hoặc LM35 của hãng National Semiconductor Corp như các bảng dưới đây.

Bảng 4.12. Cảm biến nhiệt họ LM34

Mã ký hiệu	Dải nhiệt độ	Độ chính xác	Đầu ra
LM34A	-55 F to + 300 C	+ 2.0 F	10mV/F
LM34	-55 F to + 300 C	+ 3.0 F	10mV/F
LM34CA	-40 F to + 230 C	+ 2.0 F	10mV/F
LM34C	-40 F to + 230 C	+ 3.0 F	10mV/F
LM34D	-32 F to + 212 C	+ 4.0 F	10mV/F

Bảng 4.13. Cảm biến nhiệt họ LM35

Mã ký hiệu	Dải nhiệt độ	Độ chính xác	Đầu ra
LM35A	-55 C to + 150 C	+ 1.0 C	10 mV/F
LM35	-55 C to + 150 C	+ 1.5 C	10 mV/F
LM35CA	-40 C to + 110 C	+ 1.0 C	10 mV/F
LM35C	-40 C to + 110 C	+ 1.5 C	10 mV/F
LM35D	0 C to + 100 C	+ 2.0 C	10 mV/F

4.3.4.2. Phối hợp tín hiệu và phối ghép LM35 với 8051

Hầu hết các bộ cảm biến đều đưa ra tín hiệu dưới dạng điện áp, dòng điện, dung kháng hoặc trở kháng. Tuy nhiên, chúng ta cần chuyển đổi các tín hiệu này về điện áp để đưa đến đầu vào của bộ chuyển đổi ADC. Sự chuyển đổi này được gọi là phối hợp tín hiệu.

Ví dụ: Xét trường hợp nối LM35 hoặc LM34 tới ADC0804. Vì ADC0804 có độ phân giải 8 bit với tối đa 256 mức (2^8), và LM35 (hoặc LM34) tạo ra điện áp 10mV ứng với sự thay đổi 10C, nên ta có thể phối hợp V_{in} của ADC0804 để tạo ra $V_{out} = 2560\text{mV}$ (2,6V) ở đầu ra đầy thang đo. Để tạo ra V_{out} đầy thang 2,56V của ADC0804 ta cần đặt điện áp $V_{ref}/2 = 1,28\text{V}$ (sự phụ thuộc giữa V_{out} và $V_{ref}/2$ cho trong bảng 4.13). Bảng 4.14 biểu diễn sự phụ thuộc nhiệt độ và V_{out} của ADC0804.

Bảng 4.14. Nhiệt độ và V_{out} của ADC0804

Nhiệt độ ($^{\circ}\text{C}$)	V_{in} (mV)	V_{out} (D7 – D0)
0	0	0000 0000
1	10	0000 0001
2	20	0000 0010
3	30	0000 0011
10	100	0000 1010
30	300	0001 1110

Ví dụ: Lập trình cho hệ thống đo nhiệt độ dùng cảm biến nhiệt LM35, ADC0804 và hiển thị lên LED 7 thanh (hình 4.35).

ORG 000H

MOV P1, #0FFH

SETB P3.2

LAP: CLR P3.1

SETB P3.1

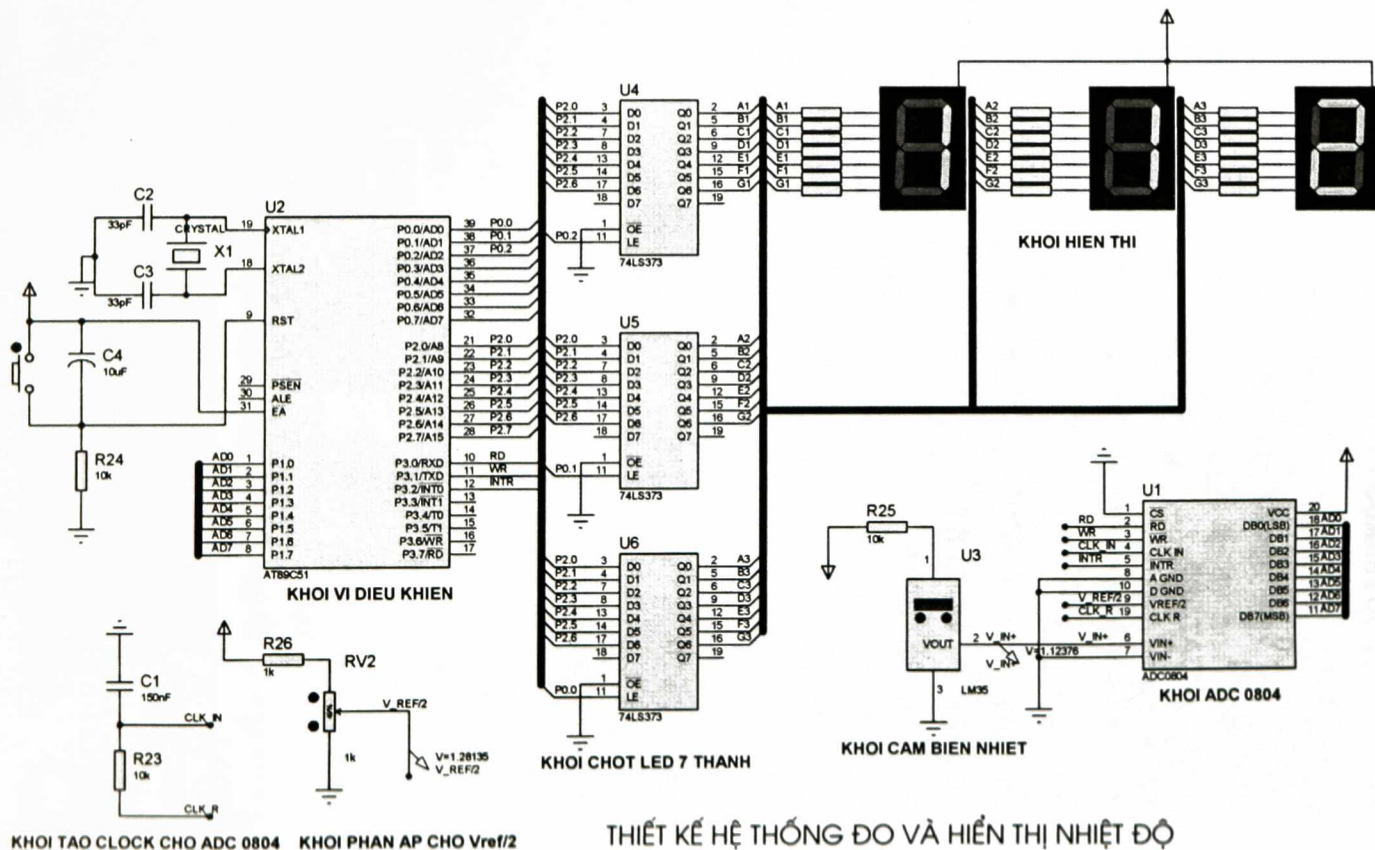
CHUYENADC:

```

        JB P3.2,CHUYENADC
        CLR P3.0
        SETB P3.0
        MOV A,P1
        ACALL XUATLED
        SJMP LAP
XUATLED:
        MOV B,#100
        DIV AB
        MOV DPTR,#Maled7
        MOVC A,@A+DPTR
        MOV P2,A
        SETB P0.2
        CLR P0.2
        MOV A,B
        MOV B,#10
        DIV AB
        MOVC A,@A+DPTR
        MOV P2,A
        SETB P0.1
        CLR P0.1
        MOV A,B
        MOVC A,@A+DPTR
        MOV P2,A
        SETB P0.0
        CLR P0.0
        RET
Maled7:
        db
0C0h,0F9h,0A4h,0B0h,99h,92h,82h,0F8h,80h,90h
END.

```

Hình 4.35. Gép nối 8051 với ADC0804 và bộ cảm biến nhiệt



THIẾT KẾ HỆ THỐNG ĐO VÀ HIỂN THỊ NHIỆT ĐỘ
DÀI HIỂN THỊ 0 - 150 ĐỘ C

Nếu có nhiều cảm biến nhiệt cần phải chuyển đổi rồi mới ghép nối với 8051 thì có thể sử dụng ADC808/809. Mỗi chip loại này có thể kiểm soát được tối đa 8 cảm biến nhiệt.

4.4. PHỐI GHÉP VI ĐIỀU KHIỂN VỚI THẾ GIỚI THỰC 2: PHỐI GHÉP 8051 VỚI ĐỘNG CƠ BƯỚC, BÀN PHÍM VÀ BỘ BIẾN ĐỔI SỐ - TƯƠNG TỰ DAC

4.4.1. NỐI GHÉP 8051 VỚI ĐỘNG CƠ BƯỚC

Phần này bắt đầu với việc giới thiệu tổng quan về hoạt động của các động cơ bước. Sau đó, mô tả cách phối ghép một động cơ bước với bộ vi điều khiển 8051. Cuối cùng sử dụng các chương trình hợp ngữ để trình diễn điều khiển góc và hướng quay của động cơ bước.

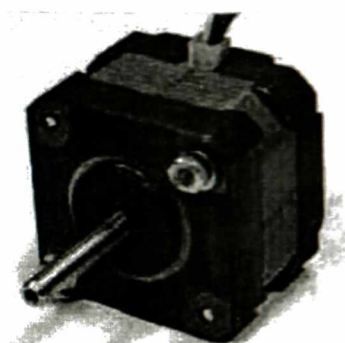
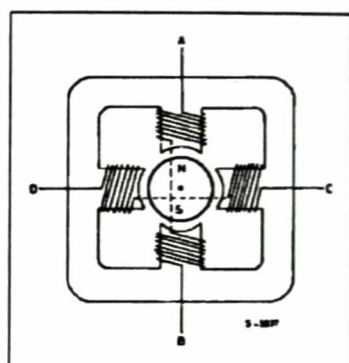
4.4.1.1. Phân loại động cơ bước

Động cơ bước là một thiết bị sử dụng rộng rãi để chuyển các xung điện thành chuyển động cơ học. Trong các ứng dụng chẳng hạn như các bộ điều khiển đĩa, các máy in kim ma trận và các máy rôbot thì động cơ bước được dùng để điều khiển chuyển động.

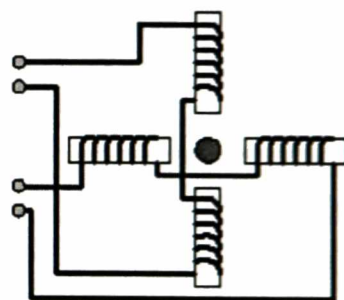
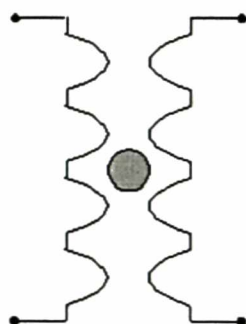
Mỗi động cơ bước đều có phần quay rôto là nam châm vĩnh cửu (cũng còn được gọi là trục dẫn - shaft) được bao bọc xung quanh là một bộ phận đứng yên gọi stato (xem hình 4.36).

Các động cơ bước thường được phân loại theo số pha:

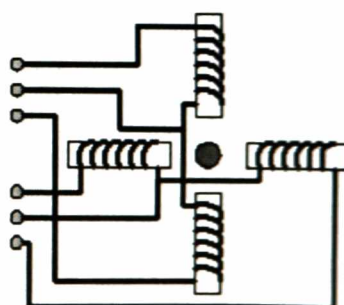
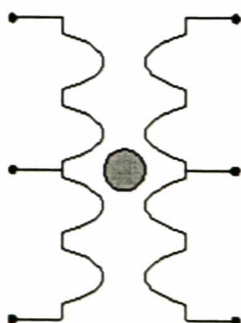
- Các động cơ bước lưỡng cực là các motor không chổi than hai pha và có 4 đầu dây (hình 4.37).
- Động cơ đơn cực là động cơ không chổi than, có bốn pha và có 5 hoặc 6 đầu dây (hình 4.38).



Hình 4.36. Mặt cắt Stator, Rotor động cơ bước



Hình 4.37. Động cơ bước lưỡng cực 2 pha không chổi than 4 đầu dây



Hình 4.38. Động cơ bước đơn cực 4 pha không chổi than 6 đầu dây

Trong trường hợp động cơ đơn cực 5 dây, cặp dây đất đã được nối ngầm ở bên trong. Một động cơ đơn cực có thể được sử dụng như một động cơ lưỡng cực bằng cách để hở các dây nối đất. Điều này sẽ làm tăng góc chuyển động, tăng mômen xoắn và do đó làm tăng lực kéo.

Ngoài những động cơ bước đơn cực và lưỡng cực, còn có một loại động cơ bước có 8 đầu dây. Với loại động cơ phổ dụng này, bằng cách nối các đầu dây với nhau theo một cách thức nhất định, người sử dụng có thể tạo ra động cơ bước đơn cực hay lưỡng cực.

4.4.1.2. Nguyên lý hoạt động của động cơ bước

Nguyên lý hoạt động của động cơ bước được mô tả như sau: Có 4 đường tín hiệu đưa vào 4 cuộn dây. Ở mỗi bước, nếu các cuộn dây này được nạp năng lượng theo đúng trình tự sẽ tạo ra một từ trường cố định và khiến cho động cơ dịch chuyển theo một chiều nhất định. Nếu ta đảo trật tự của các bước sẽ làm đảo chiều quay của động cơ.

Để dễ dàng hiểu được hoạt động của động cơ bước, dưới đây là mô phỏng điều khiển bước quay của một động cơ bước đơn cực 4 pha. Có ba cách kích thích các cuộn dây tương ứng với ba cách hoạt động khác nhau của động cơ bước.

❖ Single Coil Excitation (Full Step)

Cách thức điều khiển này được gọi là chuỗi xung 4 bước điều khiển dạng sóng (hình 4.39).

Mỗi cuộn dây trong 4 cuộn dây lần lượt được kích hoạt. Trật tự kích thích này sẽ tạo ra sự di chuyển trơn tru và tổn ít năng lượng nhất.

Step	Coil 4	Coil 3	Coil 2	Coil 1	
b.1	On	Off	Off	Off	
b.2	Off	On	Off	Off	
b.3	Off	Off	On	Off	
b.4	Off	Off	Off	On	

Chiều kim đồng hồ ↓	Bước	Cuộn dây 4	Cuộn dây 3	Cuộn dây 2	Cuộn dây 1	Chiều quay bộ đếm ↑
		1	0	0	0	
	2	0	1	0	0	
	3	0	0	1	0	
	4	0	0	0	1	

Hình 4.39. Chuỗi xung 4 bước điều khiển dạng sóng

❖ Two Coil Excitation (FullStep)

Cách thức điều khiển này được gọi là chuỗi xung 4 bước thông thường (hình 4.40). Ở mỗi bước, có hai cuộn dây liên tiếp được nạp năng lượng.

Step	Coil 4	Coil 3	Coil 2	Coil 1	
b.1	On	On	Off	Off	
b.2	Off	On	On	Off	
b.3	Off	Off	On	On	
b.4	On	Off	Off	On	

Chiều kim đồng hồ ↓	Bước	Cuộn dây 4	Cuộn dây 3	Cuộn dây 2	Cuộn dây 1	↑ Chiều quay bộ đếm
	1	1	1	0	0	
	2	0	1	1	0	
	3	0	0	1	1	
	4	1	0	0	1	

Hình 4.40. Chuỗi xung 4 bước thông thường

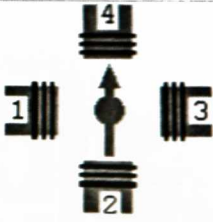
Cách kích thích này gây chuyển động không trơn tru và tốn nhiều năng lượng nhưng lại tạo ra được mômen quay lớn hơn. Cần chú ý rằng:

- Cuộn dây 4 luôn ngược chiều với cuộn dây 2.
- Cuộn dây 1 luôn ngược chiều với cuộn dây 3.

❖ Half Step

Cách thức điều khiển này được gọi là chuỗi xung 8 bước (hình 4.41)

	Bước	Cuộn dây 4	Cuộn dây 3	Cuộn dây 2	Cuộn dây 1
Chiều kim đồng hồ ↓	1	1	0	0	0
	2	1	1	0	0
	3	0	1	0	0
	4	0	1	1	0
	5	0	0	1	0
	6	0	0	1	1
	7	0	0	0	1
	8	1	0	0	1
					↑ Chiều quay bộ đếm

Step	Coil 4	Coil 3	Coil 2	Coil 1	
b.1	On	Off	Off	Off	
b.2	On	On	Off	Off	

b.3	Off	On	Off	Off	
b.4	Off	On	On	Off	
b.5	Off	Off	On	Off	
b.6	Off	Off	On	On	
b.7	Off	Off	Off	On	
b.8	On	Off	Off	On	

Hình 4.41. Xung 8 bước

4.4.1.3. Một số thông số và khái niệm

❖ Góc bước (Step Angle)

Vậy mỗi bước có độ dịch chuyển là bao nhiêu? Điều này phụ thuộc vào cấu trúc bên trong của động cơ, đặc biệt là số răng của stato và rôto. Góc bước là độ quay nhỏ nhất của một bước, các động cơ khác nhau có các góc bước khác nhau. Bảng 4.15 trình bày một số góc bước đối với các động cơ khác nhau. Trong bảng có sử dụng thuật ngữ số bước trong một vòng (Steps per revolution). Đây là tổng số bước cần để quay hết một vòng 360^0 (chẳng hạn $180 \text{ bước} \times 2^0 = 360^0$).

Bảng 4.15

Góc bước	Số bước/vòng
0.72	500
1.8	200
2.0	180
2.5	144
5.0	72
7.5	48
15	24

❖ Quan hệ số bước/giây và số vòng quay/phút RPM

Quan hệ giữa số vòng quay trong phút RPM (revolutions per minute), số bước trong vòng quay và số bước trong vòng giây là quan hệ thuộc về trực giác và nó được biểu diễn như sau:

$$\text{Số bước trong giây} = \frac{\text{RPM} \times \text{Số bước trong vòng quay}}{60}$$

❖ Chuỗi xung bốn bước và số răng trên rôto

Chuỗi xung chuyển mạch được trình bày ở bảng trong hình 4.40 được gọi là chuỗi chuyển mạch 4 bước. Vậy độ dịch chuyển của 4 bước này sẽ là bao nhiêu? Sau mỗi khi thực hiện 4 bước này thì rôto chỉ dịch được một bước răng. Do vậy, trong động cơ bước với 200 bước/vòng thì rôto của nó có 50 răng vì $50 \times 4 = 200$ bước cần để quay hết một vòng. Điều này dẫn đến một kết luận là góc bước tối thiểu luôn là hàm của số răng trên rôto. Hay nói cách khác, góc bước càng nhỏ thì rôto quay được càng nhiều răng. Hãy xét ví dụ sau:

Ví dụ 1:

Hãy tính số lần của chuỗi 4 bước ở bảng trong hình 4.40 phải cấp cho một động cơ bước để tạo ra một dịch chuyển 80° nếu động cơ góc bước là 2° .

Lời giải:

Một động cơ có góc bước là 2° thì phải có những đặc tính sau: góc bước 2° , số bước/vòng là 180, số răng của rôto là 45, độ dịch chuyển sau mỗi chuỗi 4 bước là 8° . Vậy để dịch chuyển 80° thì cần 10 chuỗi 4 bước.

Nhìn vào ví dụ trên thì có người sẽ hỏi vậy muốn dịch chuyển đi 45° thì làm thế nào khi góc bước là 2° . Muốn có độ phân giải nhỏ hơn thì tất cả mọi động cơ bước đều cho phép chuỗi chuyển mạch 8 bước, chuỗi 8 bước cũng còn được gọi chuỗi nửa bước (half - stepping), vì trong chuỗi 8 bước thì mỗi bước là một nửa của góc bước bình thường. Ví dụ, một động cơ có góc bước là 2° có thể sử dụng góc bước 1° nếu áp dụng chuỗi ở bảng trong hình 4.41.

❖ Tốc độ động cơ

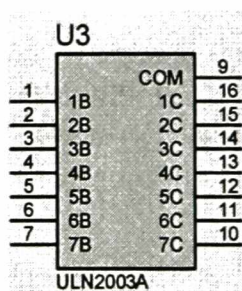
Tốc độ động cơ được đo bằng số bước trong một giây (bước/giây) là một hàm của tốc độ chuyển mạch.

❖ Mômen giữ

Mômen giữ là lượng mômen ngoài cần thiết để làm quay trục động cơ từ vị trí giữ của nó với điều kiện trục động cơ đang đứng yên hoặc đang quay với tốc độ $RPM = 0$. Đại lượng này được đo bằng tỷ lệ điện áp và dòng cấp đến động cơ. Đơn vị của mômen giữ là kilôgam – centimet.

❖ Dòng kích từ của cuộn dây

Chúng ta cũng phải lưu ý một điều rằng dòng đầu ra của 8051 không thể đủ lớn để điều khiển các cuộn dây động cơ. Vì vậy, ta cần phải sử dụng thêm các IC chuyên dụng chẳng hạn như ULN2003A để cấp dòng cho cuộn dây, hoặc sử dụng các bóng bán dẫn Tranzitor công suất, nhưng lưu ý phải mắc thêm các Diot để ngăn dòng cảm ứng ngược. Vì vậy, lý do chính mà ULN2003A được ưa chuộng đó là có sẵn các Diot bên trong.



INPUT (1B – 7B)	OUTPUT (1C – 7C)
H	L
L	H

Hình 4.42. Sơ đồ chân và bảng chân lý: H = Hight logic, L = Low logic

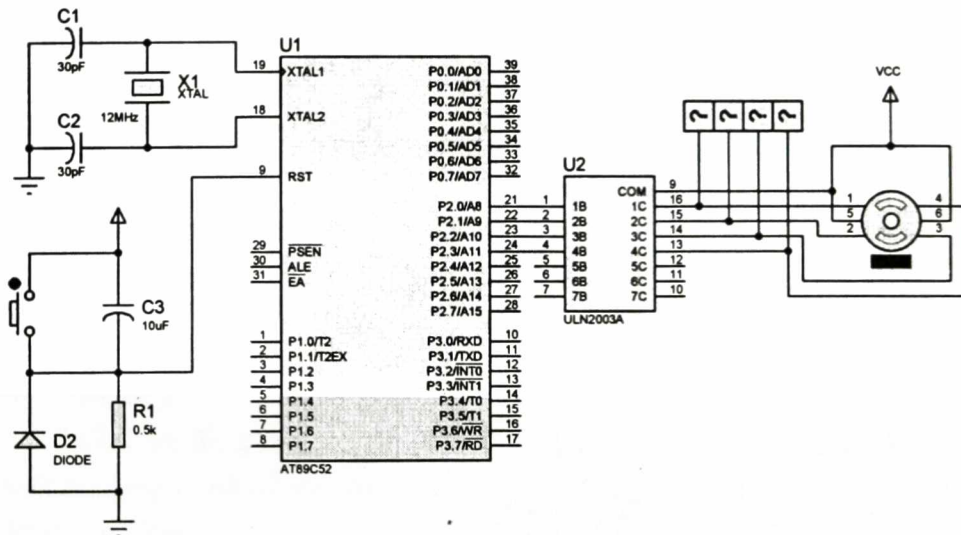
Ví dụ 2:

Hãy mô tả kết nối 8051 với động cơ bước của hình 4.43 và viết một chương trình quay nó liên tục bằng cách sử dụng chuỗi xung 4 bước thông thường.

Lời giải:

Các bước dưới đây trình bày việc kết nối 8051 với động cơ bước và lập trình của nó.

1. Sử dụng một ôm kế để đo trở kháng của các đầu dây. Điều này xác định đầu chung (COM) nào được nối tới cuộn dây nào?
2. Các dây chung được nối tới đầu dương của nguồn cấp cho động cơ. Trong nhiều động cơ thì + 5V là đủ.
3. Chương trình được viết như sau:



Hình 4.43. Ghép nối 8051 với động cơ bước

```
ORG 0000H
```

```
MOV A, #66H; nạp chuỗi xung bước
```

```
BACK: MOV P2, A ; xuất chuỗi xung đến động cơ
```

```
RR A ; quay theo chiều kim đồng hồ
```

```
ACALL DELAY ; tạo trễ
```

```
SJMP BACK ; tiếp tục chạy
```

```
DELAY:
```

```
MOV R2, #200
```

```
H1: MOV R3, #255
```

```
H2: DJNZ R3, H2
```



```

        DJNZ      R2, H1
        RET
END

```

Thay đổi giá trị của DELAY để đặt tốc độ quay. Ta cũng có thể sử dụng lệnh đơn bit SETB và CLR hoặc đưa tới cổng P2 các giá trị tương ứng với bit nối tới cuộn dây thay cho lệnh RR A để tạo ra chuỗi xung.

```

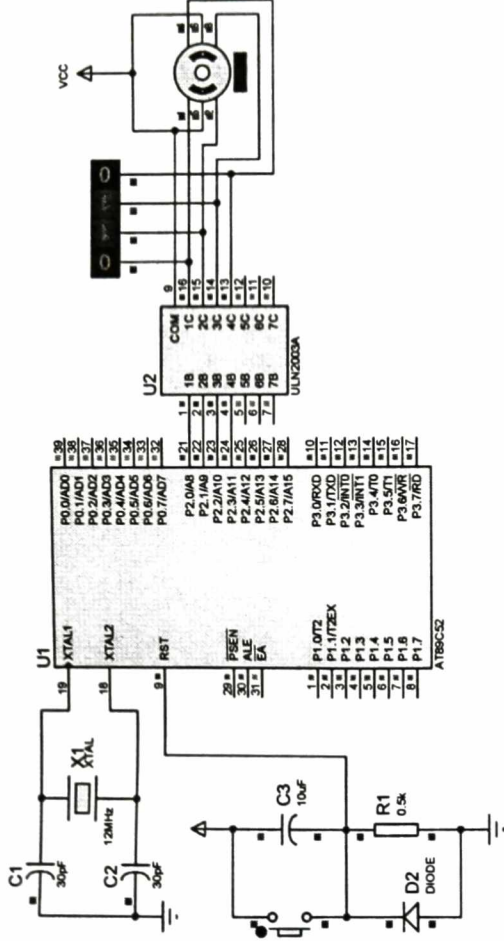
ORG 0H

LAP: MOV      P2, #06H
ACALL        DELAY
MOV          P2, #03H
ACALL        DELAY
MOV          P2, #09H
ACALL        DELAY
MOV          P2, #0CH
ACALL        DELAY
SJMP         LAP

DELAY: MOV     R2, #200
H1:  MOV R3, #255
H2:  DJNZ      R3, H2
      DJNZ R2, H1
      RET
END

```

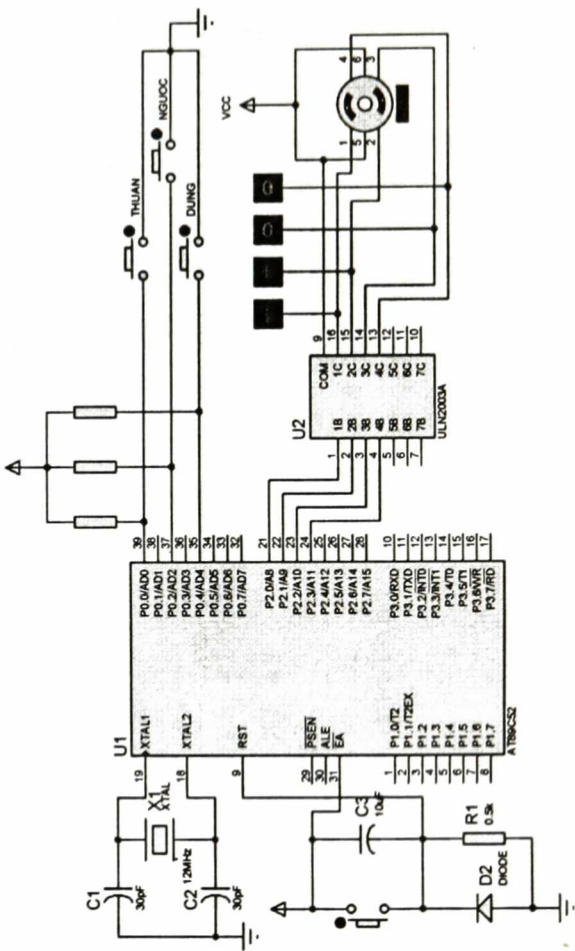
Sơ đồ ghép nối và mô phỏng trên phần mềm Proteus như sau:



Hình 4.44. Mô phỏng ghép nối 8051 với động cơ bước trên phần mềm Proteus

Ví dụ 3:

Cho sơ đồ ghép nối vi điều khiển 8051 với động cơ bước. Hãy viết chương trình để khi ấn nút THUAN động cơ sẽ quay thuận, ấn nút NGUOC động cơ quay ngược, ấn nút DUNG động cơ dừng.



Hình 4.45. Sơ đồ ghép nối vi điều khiển 8051 với động cơ bước

Giải:

```
org 00h
MOV     P0,#0FFH
MOV     A,#66H
BAMNUT: JNB     P0.0,THUAN
        JNB     P0.2,NGUOC
        SJMP    BAMNUT
THUAN:  MOV     P2,A
        RR      A
        ACALL   DELAY
        JNB     P0.2,NGUOC
        JNB     P0.4,DUNG
        SJMP    THUAN
NGUOC:  MOV     P2,A
        RL      A
        ACALL   DELAY
        JNB     P0.0,THUAN
        JNB     P0.4,DUNG
        SJMP    NGUOC
DUNG:   MOV     P2,#0FFH
        SJMP    BAMNUT
;*****CHUONG TRINH CON TAO TRE
DELAY:
        MOV     R2,#200
H1:     MOV     R3,#255
H2:     DJNZ    R3,H2
        DJNZ    R2,H1
        RET
end
```

Ví dụ 4: Viết chương trình điều khiển động cơ bước quay một vòng rồi ngừng

```

        OUTSTEP EQU P2
        SOBUOC EQU 50      ;LOAI DCO:50X4=200 BUOC
ORG 0000H
        MOV DPTR,#DATASTEP;NAP DIA CHI QUAN LY MA
        MOV R0,#0
        MOV R2,#SOBUOC    ;50 CHU KY LA VONG
MAIN2:
        MOV R1,#4          ;1 CHU KY 4 BUOC
MAIN1:
        MOV A,R0
        MOVC A,@A+DPTR
        MOV OUTSTEP,A
        LCALL DELAY
        INC R0
        ANL R0H,#03H      ;ANL R0 VOI 00000011B
        DJNZ R1,MAIN1
        DJNZ R2,MAIN2
HERE:   SJMP HERE          ;DUNG LAI
;-----CHTR CON DELAY
DELAY:  MOV R6,#10H
DEL:    MOV R7,#0
DEL1:   DJNZ R7,DEL1
        DJNZ R6,DEL
        RET
;-----KHAi DU LIEU DIEU KHIEN
DONG CO BUOC
        DATASTEP:

```


DB 10001000B

DB 01000100B

DB 00100010B

DB 00010001B

END.

4.4.2. NỐI GHÉP 8051 VỚI BÀN PHÍM

Các bàn phím và LCD là những thiết bị vào/ra được sử dụng rộng rãi nhất của 8051 và cần phải hiểu một cách cơ bản về chúng. Ở phần này trước hết ta giới thiệu các kiến thức cơ bản về bàn phím với cơ cấu ấn phím và tách phím, sau đó giới thiệu về giao tiếp 8051 với bàn phím.

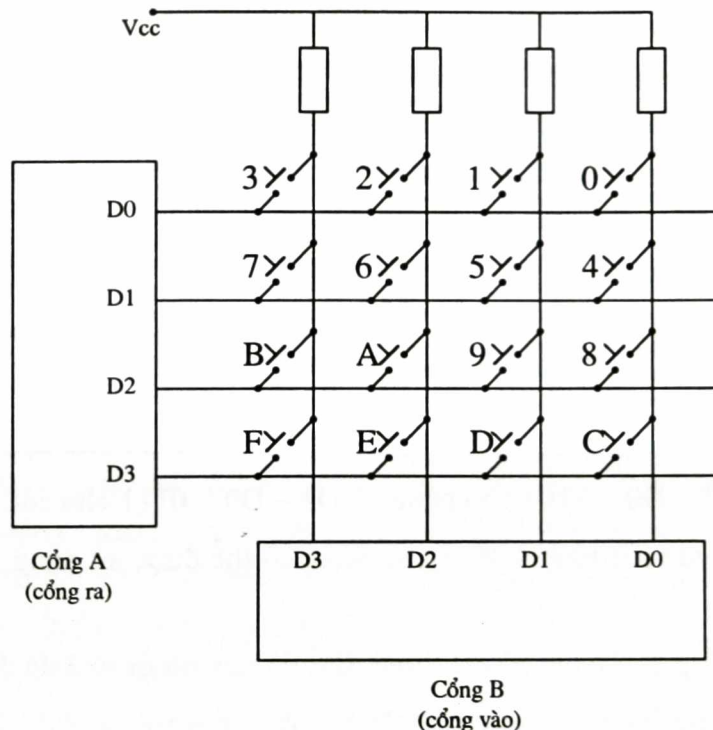
4.4.2.1. Phối ghép bàn phím với 8051

Ở mức thấp nhất các bàn phím được tổ chức dưới dạng một ma trận các hàng và các cột. CPU truy cập cả hàng lẫn cột thông qua các cổng. Do vậy, với hai cổng 8 bit thì có thể nối tới một bàn phím 8×8 tới bộ vi xử lý. Khi một phím được ấn thì một hàng và một cột được tiếp xúc, ngoài ra không có sự tiếp xúc nào giữa các hàng và các cột. Trong các bàn phím máy tính IBM PC có một bộ vi điều khiển (bao gồm một bộ vi xử lý, bộ nhớ RAM và EPROM và một số cổng tất cả được bố trí trên một chip) chịu trách nhiệm phối ghép phần cứng và phần mềm của bàn phím. Trong những hệ thống như vậy, nó là chức năng của các chương trình được lưu trong EPROM của bộ vi điều khiển để quét liên tục các phím, xác định xem phím nào đã được kích hoạt và gửi nó đến bo mạch chính. Trong phần này nghiên cứu về cơ cấu 8051 quét và xác định phím.

4.4.2.2. Quét và xác định phím

Hình 4.46 trình bày một ma trận 4×4 được nối tới hai cổng. Các hàng được nối tới một đầu ra và các cột được nối tới một cổng vào.

Nếu không có phím nào được ấn thì việc đóng cổng vào sẽ hoàn toàn là 1 cho tất cả các cột vì tất cả được nối tới dương nguồn Vcc. Nếu tất cả các hàng được nối đất và một phím được ấn thì một trong các cột sẽ có giá trị 0 vì phím được ấn tạo đường xuống đất. Chức năng của bộ vi điều khiển là quét liên tục để phát hiện và xác định phím được ấn.



Hình 4.46. Nối ghép bàn phím ma trận tới các cổng

4.4.2.3. Nối đất các hàng và đọc các cột

Để phát hiện một phím được ấn thì bộ vi điều khiển nối đất tất cả các hàng bằng cách cấp 0 tới chốt đầu ra, sau đó nó đọc các hàng. Nếu dữ liệu được đọc từ các cột là $D3 - D0 = 1111$ thì không có phím nào được ấn và quá trình tiếp tục cho đến khi phát hiện một phím được ấn. Tuy nhiên, nếu một trong các bit cột có số 0 thì điều đó có nghĩa là việc ấn phím đã xảy ra. Ví dụ, nếu $D3 - D0 = 1101$ có nghĩa là một phím ở cột 1 được ấn. Sau khi một ấn phím được phát hiện, bộ vi điều

khởi sẽ chạy quá trình xác định phím. Bắt đầu với hàng trên cùng, bộ vi điều khiển nổi bật bằng cách chỉ cấp mức thấp tới chân D0, sau đó nó đọc các cột. Nếu dữ liệu đọc được là toàn số 1 thì không có phím nào của hàng này được ấn và quá trình này chuyển sang hàng kế tiếp. Nó nổi bật hàng kế tiếp, đọc các cột và kiểm tra xem có số 0 nào không? Quá trình này tiếp tục cho đến khi xác định được hàng nào có phím ấn. Sau khi xác định được hàng có phím được ấn thì công việc tiếp theo là tìm ra phím ấn thuộc cột nào. Điều này thật dễ dàng vì bộ vi điều khiển biết tại thời điểm bất kỳ hàng nào và cột nào được truy cập. Hãy xét ví dụ sau:

Ví dụ 1:

Từ hình 4.46 hãy xác định hàng và cột của phím được ấn cho các trường hợp sau đây:

a. $D3 - D0 = 1110$ cho hàng và $D3 - D0 = 1011$ cho cột.

b. $D3 - D0 = 1101$ cho hàng và $D3 - D0 = 0111$ cho cột.

Lời giải: Từ hình 4.46 cột và hàng có thể được sử dụng xác định phím.

a. Hàng thuộc D0 và cột thuộc D2, do vậy phím số 2 đã được ấn.

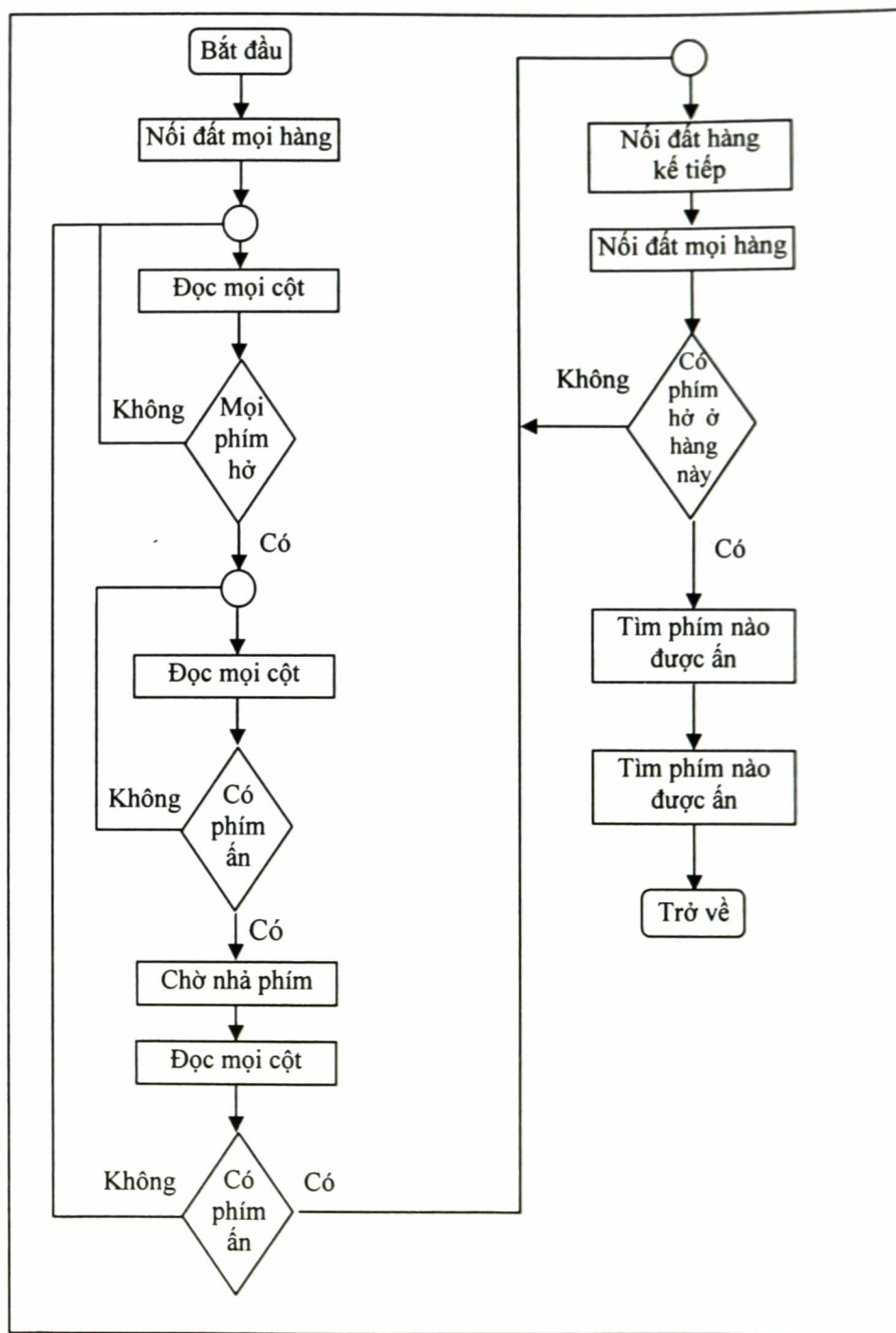
b. Hàng thuộc D1 và cột thuộc D3, do vậy phím số 7 đã được ấn.

Chương trình ví dụ 2 là chương trình hợp ngữ của 8051 để phát hiện và xác định sự kích hoạt phím. Trong chương trình này P1 và P2 được giả thiết là cổng ra và cổng vào tương ứng. Chương trình đi qua 4 giai đoạn chính sau đây.

1. Khẳng định phím trước đó đã được nhả, các số không là đầu ra tới tất cả các hàng cùng một lúc và các cột được đọc và kiểm tra chừng nào tất cả mọi cột đều cao. Khi tất cả các cột được phát hiện là đều cao thì chương trình chờ một thời gian ngắn trước khi nó chuyển sang giai đoạn kế tiếp để chờ một phím được ấn.

2. Để biết có một phím nào được ấn các cột được quét đi quét lại trong vòng vô tận cho đến khi có một cột có số 0. Hãy nhớ rằng các chốt đầu ra được nối tới các hàng vẫn có các số 0 ban đầu (được cấp ở giai đoạn 1) làm cho chúng được nối đất. Sau khi phát hiện ấn phím, nó đợi 20ms chờ cho phím nhả ra và sau đó quét lại các cột. Điều này phục vụ hai chức năng: a) nó đảm bảo rằng việc phát hiện ấn phím đầu tiên không bị sai do nhiễu và b) thời gian giữ chậm là 20ms ngăn ngừa việc ấn cùng một phím như là nhiều lần ấn. Nếu sau 20ms giữ chậm mà phím vẫn được ấn nó chuyển sang giai đoạn kế tiếp để phát hiện phím ấn thuộc hàng nào, nếu không nó quay chờ vòng lặp để phát hiện có một phím ấn thật.
3. Để phát hiện ấn phím thuộc hàng, nó nối đất mỗi hàng tại một thời điểm, đọc các cột mỗi lần. Nếu nó phát hiện tất cả mọi cột đều cao, điều này có nghĩa là ấn phím không thuộc hàng đó, do vậy nó nối đất hàng kế tiếp và tiếp tục cho đến khi phát hiện ra hàng có phím ấn. Khi tìm hàng có phím ấn, nó thiết lập địa chỉ bắt đầu cho bảng trình bày giữ các mã quét (hoặc giá trị ASCII) cho hàng đó và chuyển sang giai đoạn kế tiếp để xác định phím.
4. Để xác định phím ấn, nó quay các bit cột, mỗi lần một bit vào cờ nhớ và kiểm tra xem nó có giá trị thấp không? Khi tìm ra số 0, nó kéo mã ASCII dành cho phím đó ra từ bảng trình bày. Nếu không tìm được số 0 thì nó tăng con trỏ để chỉ đến phần tử kế tiếp của bảng trình bày. Hình 4.47 trình bày lưu đồ quá trình tìm phím ấn này.

Trong khi việc phát hiện ấn là chuẩn cho tất cả mọi bàn phím thì quá trình xác định phím nào được ấn lại không giống nhau. Phương pháp sử dụng bảng trình bày được đưa ra trong chương trình ví dụ 2 có thể được sửa đổi để làm việc với bất kỳ ma trận kích thước 8×8 nào. Hình 4.47 là lưu đồ thuật toán của chương trình ví dụ 2 để quét và xác định phím ấn.



Hình 4.47. Lưu đồ tìm phím ấn của chương trình ví dụ 2

Chương trình ví dụ 2:

; Các chân P1.0 - P1.3 được nối tới các hàng
; còn P2.0 - P2.3 nối tới các cột.

MOV P2,#0FFH ;chon P2 lam cong vao

K1:

MOV P1,#0 ;tiap dat cac hang
MOV A,P2 ;doc cac cot
ANL A,#00001111B ;che cac bit khong dung
CJNE A,#00001111B,K1 ;kiem tra nha phim

K2:

ACALL DELAY ;tao tre 20ms
MOV A,P2 ;kiem tra phim nhan
ANL A,#00001111B ;che cac bit khong dung
CJNE A,#00001111B,OVER;neu phim nhan, cho xong
SJMP K2 ;neu khong kiem tra lai

OVER:

ACALL DELAY ;tao tre 20ms
MOV A,P2 ;kiem tra nhan xong
ANL A,#00001111B ;che cac bit khong dung
CJNE A,#00001111B,OVER1;neu co phim nhan,
;tim hang
SJMP K2 ;neu khong thi giam sat

OVER1:

MOV P1,#11111110B ;tiap dat hang 0
MOV A,P2 ;doc tat ca cac cot

```

ANL  A,#00001111B      ;che cac bit khong dung
CJNE A,#00001111B,ROW_0;hang 0, tim cot
MOV  P1,#1111101B      ;tiep dat hang 1
MOV  A,P2               ;doc tat ca cac cot
ANL  A,#00001111B      ;che cac bit khong dung
CJNE A,#00001111B,ROW_1;hang 1, tim cot
MOV  P1,#11111011B     ;tiep dat hang 2
MOV  A,P2               ;doc tat ca cac cot
ANL  A,#00001111B      ;che cac bit khong dung
CJNE A,#00001111B,ROW_2;hang 2, tim cot
MOV  P1,#11110111B     ;tiep dat hang 3
MOV  A,P2               ;doc tat ca cac cot
ANL  A,#00001111B      ;che cac bit khong dung
CJNE A,#00001111B,ROW_3;hang 3, tim cot
LJMP K2

ROW_0:
MOV  DPTR,#KCODE0;dat DPTR khoi dong tu hang 0
SJMP FIND               ;tim cot co phim nhan

ROW_1:
MOV  DPTR,#KCODE1;dat DPTR khoi dong tu hang 1
SJMP FIND               ;tim cot co phim nhan

ROW_2:
MOV  DPTR,#KCODE2;dat DPTR khoi dong tu hang 2
SJMP FIND               ;tim cot co phim nhan

ROW_3:

```

```

MOV DPTR,#KCODE3;dat DPTR khoi dong tu hang 3
FIND:

```

```

RRC A          ;kiem tra CY co o muc thap
JNC MATCH      ;neu bang 0 lay ma ASCII
INC DPTR       ;neu khac 0 tro den cot tiep
SJMP FIND      ;tiep tuc tim

```

```

MATCH:

```

```

CLR A
MOVC A,@A+DPTR;xac dinh ma ASCII tu bang
MOV P0,A      ;hien thi phim nhan
LJMP K1

```

```

DELAY: MOV R2,#180H;

```

```

H1: MOV R3,#255

```

```

H2: DJNZ R3,H2
    DJNZ R2,H1

```

```

RET

```

```

;Bang tham chieu ma ASCII theo cac hang

```

```

ORG          300H

```

```

KCODE0:DB'0','1','2','3'          ;hang 0
KCODE1:DB'4','5','6','7'          ;hang 1
KCODE2:DB'8','9','A','B'          ;hang 2
KCODE3:DB'C','D','E','F'          ;hang 3

```

```

END

```

4.4.3. PHỐI GHÉP MỘT DAC VỚI 8051

4.4.3.1. Bộ biến đổi số - tương tự DAC

Bộ biến đổi - tương tự DAC là một thiết bị được sử dụng rộng rãi để chuyển đổi các xung số hoá về các tín hiệu tương tự. Trong phần này ta giới thiệu cơ sở phối ghép một bộ DAC với 8051.

Xem lại các kiến thức điện tử số ta thấy có hai cách tạo ra bộ DAC: Phương pháp trọng số nhị phân và phương trình thang R/2R. Nhiều bộ DAC dựa trên các mạch tổ hợp, bao gồm MC1408 (DAC0808) được sử dụng trong phần này đều sử dụng phương pháp hình thang R/2R vì nó có thể đạt độ chính xác cao hơn. Tiêu chuẩn đánh giá một bộ DAC đầu tiên là độ phân giải hàm của số đầu vào nhị phân. Các độ phân giải chúng là 8, 10 và 12 bit. Số các đầu vào bit dữ liệu quyết định độ phân giải của bộ DAC, vì số mức đầu ra tương tự bằng 2^n với n là đầu vào bit dữ liệu. Do vậy, một bộ DAC 8 bit như DAC0808 chẳng hạn có 256 mức đầu ra điện áp (dòng điện) rời rạc. Tương tự như vậy, một bộ DAC 12 bit cho 4096 mức điện áp rời rạc. Cũng có các bộ DAC 16 bit nhưng chúng rất đắt.

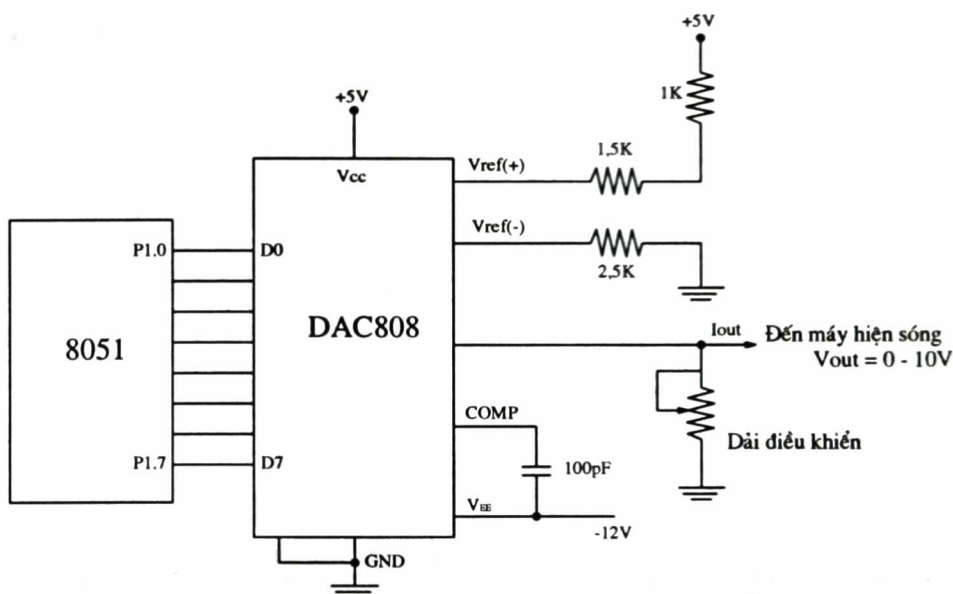
4.4.3.2. Bộ biến đổi DAC MC1408 (hay DAC0808)

Trong bộ DAC0808 các đầu vào số được chuyển đổi thành dòng (I_{out}) và việc nối một điện trở tới chân I_{out} ta chuyển kết quả thành điện áp. Dòng tổng được cấp bởi chân I_{out} là một hàm số nhị phân ở các đầu vào D0 – D7 của DAC0808 và tham chiếu I_{ref} như sau:

$$I_{out} = I_{ref} \left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256} \right)$$

Trong đó: D0 là bit thấp nhất LSB và D7 là bit cao nhất MSB đối với các đầu vào. Dòng đầu vào I_{ref} phải được áp vào chân 14. Dòng I_{ref} thường đặt giá trị 2mA. Hình 4.48 giới thiệu mạch tạo tham chiếu dòng điện (thiết lập $I_{ref} = 2mA$) bằng cách sử dụng điện áp nuôi 5V và các điện trở 1KΩ, 1.5KΩ. Nếu $I_{ref} = 2mA$, còn tất cả đầu vào nối tới

DAC ở mức cao thì dòng cực đại ở đầu ra là 1,99mA (tính theo công thức trên).



Hình 4.48. Ghép nối 8051 với DAC0808

4.4.3.3. Chuyển I_{out} sang điện áp ở DAC0808

Nếu nối điện trở tới chân I_{out} thì dòng được chuyển thành điện áp và có thể kiểm tra đầu ra bằng máy hiện sóng. Tuy nhiên, như vậy sẽ làm giảm độ chính xác do bị thay đổi trở kháng vào của tải. Vì vậy, dòng ra I_{out} cần được cách ly bằng cách dùng khuếch đại thuật toán với điện trở hồi tiếp là 5K chẳng hạn.

Ví dụ 1: Tính V_{out} khi đầu vào là mã nhị phân cho dưới đây và cho hệ số khuếch đại của khuếch đại thuật toán là 5, $I_{ref} = 2mA$.

a. 1001 1001 (99H)

b. 1100 1000 (C8H)

Giải:

a. $I_{out} = 2 \times (153/256) = 1,195mA$ và $V_{out} = 1,195 \times 5 = 5,975V$

b. $I_{out} = 2 \times (200/256) = 1,562\text{mA}$ và $V_{out} = 1,562 \times 5 = 7,813\text{V}$

Ví dụ 2:

Đề tạo ra điện áp dạng bậc thang hãy nối mạch theo hình 4.48 và nối máy hiện sóng tới đầu ra. Hãy viết đoạn chương trình gửi dữ liệu tới bộ DAC để tạo ra sóng dạng bậc thang.

Giải:

```

ORG      0H

                CLR      A

GAIN:        MOV      P2, A

                INC      A

                ACALL    DELAY

                SJMP     AGAIN

DELAY:        MOV      R3, #0FFH

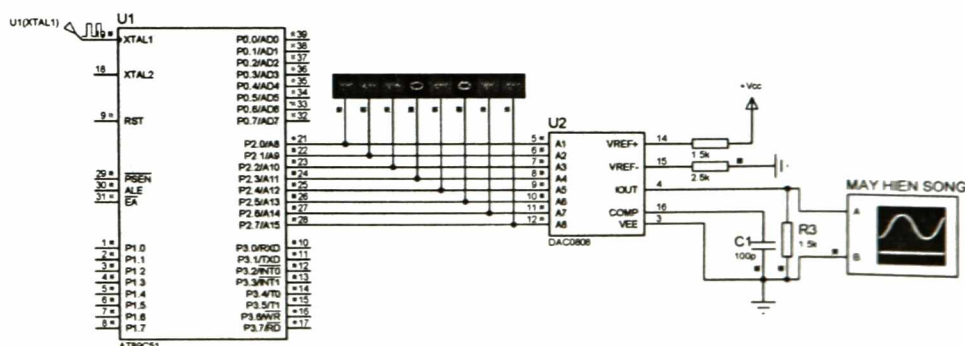
BACK:        DJNZ     R3, BACK

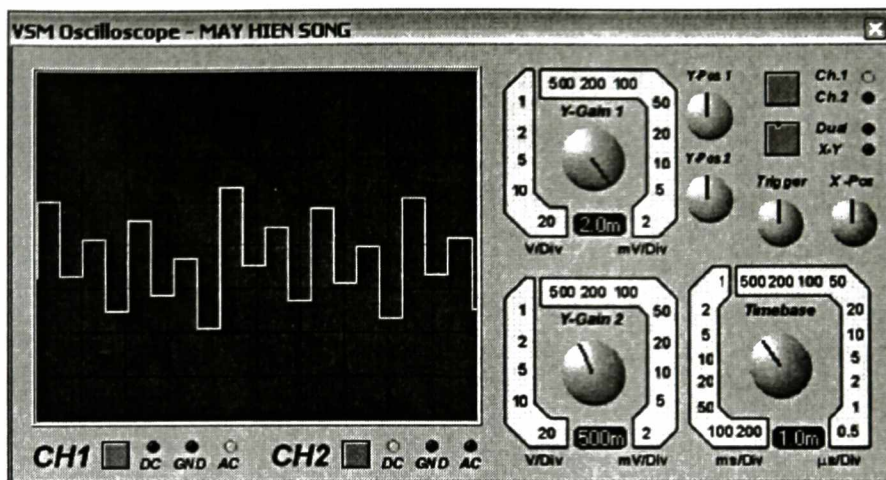
                RET

END.

```

Sơ đồ ghép nối và mô phỏng trên phần mềm Proteus giữa vi điều khiển 8051 với bộ chuyển đổi số – tương tự DAC0808 như hình 4.49.

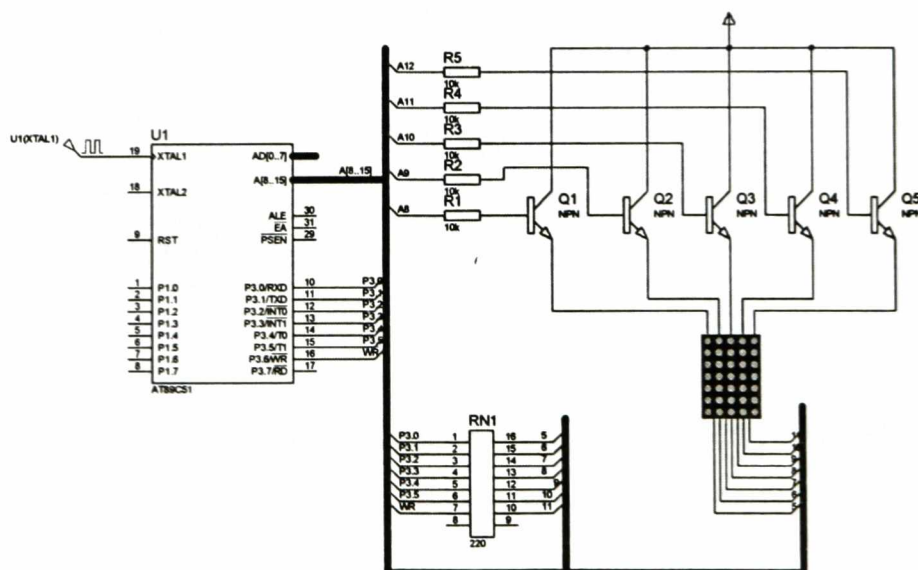




Hình 4.49. Sơ đồ ghép nối mô phỏng trên Proteus và kết quả đo được trên máy hiện sóng

• Phần bài tập

Bài 1. Cho sơ đồ ghép nối 8051 với ma trận LED như hình 4.50. Hãy viết chương trình liên tục chạy dòng chữ “BM KTVĐK” lên ma trận LED.

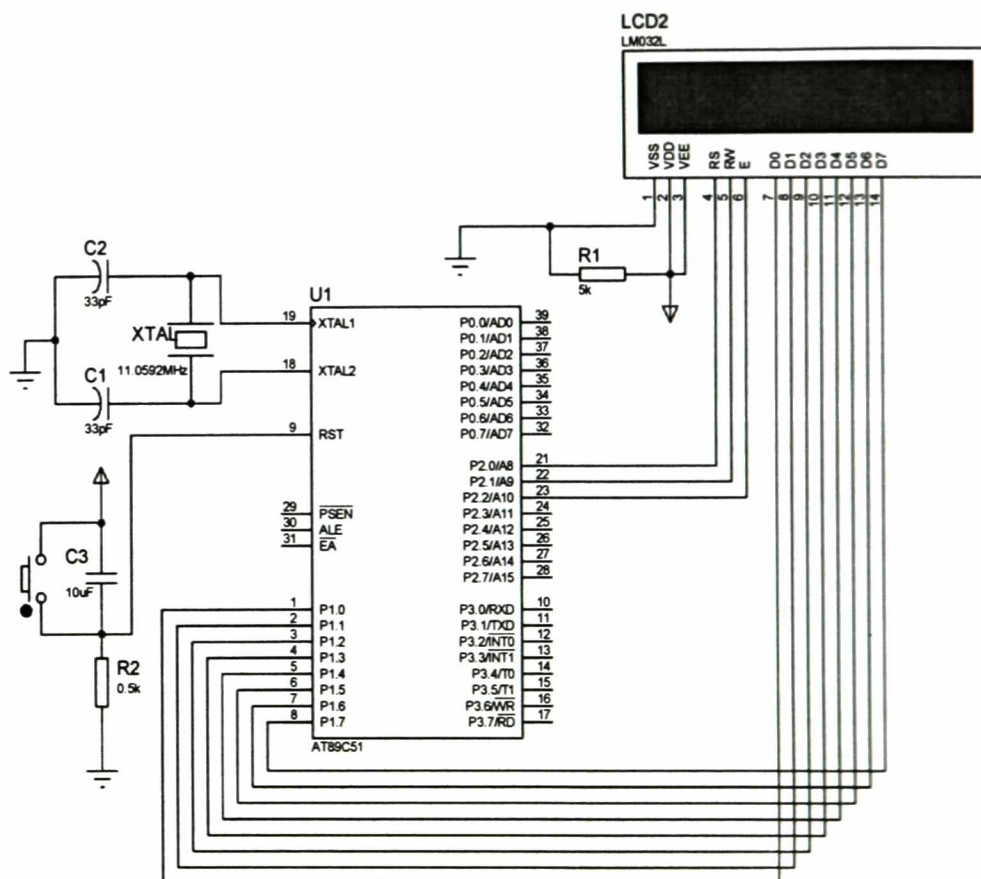


Hình 4.50

Bài 2. Cho sơ đồ ghép nối giữa 8051 với LCD LM032L. Viết chương trình liên tục hiển thị dòng chữ:

**“BO MÔN KTVĐK
KHOA ĐIỆN - ĐHKTCN”**

Lên LCD.

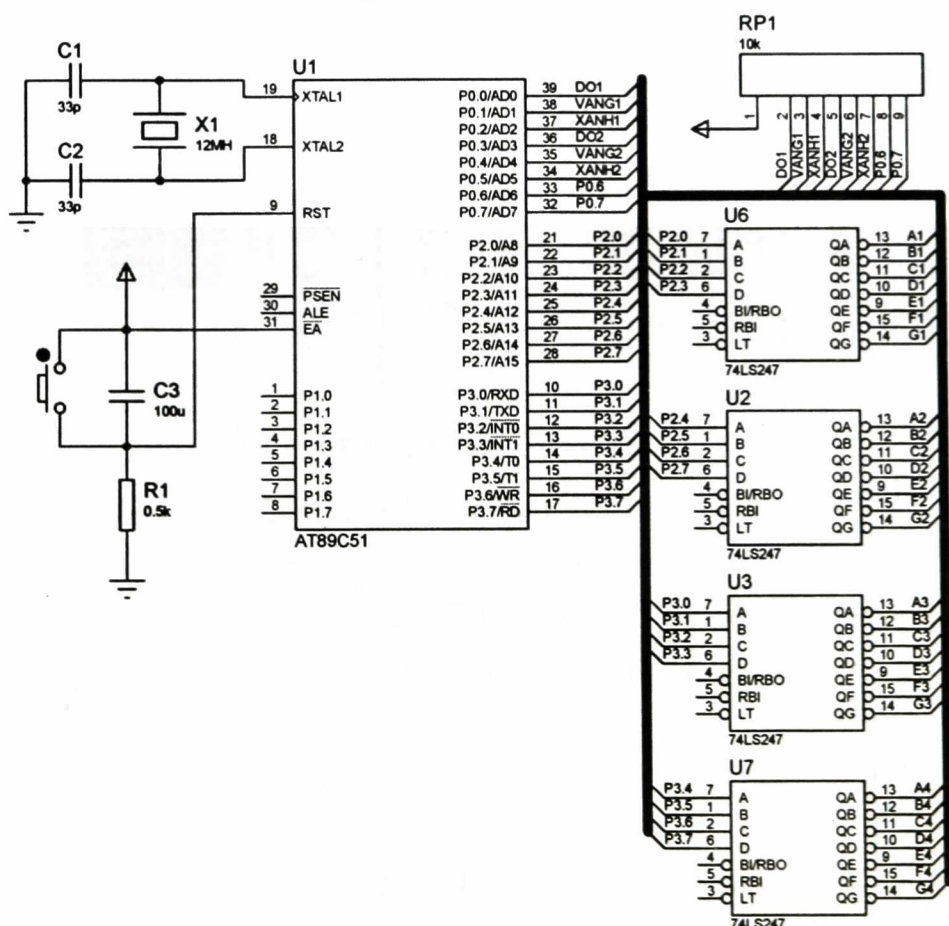


Hình 4.51

Bài 3. Cho hệ thống đèn giao thông như hình 4.52, 4.53:

Viết chương trình dùng Timer điều khiển đèn giao thông tại một giao lộ. Cho biết rằng:

Đèn	Bit điều khiển	Thời gian
Xanh chính		33s
Vàng chính		2s
Đỏ chính		
Xanh phụ		25s
Vàng phụ		2s
Đỏ phụ		

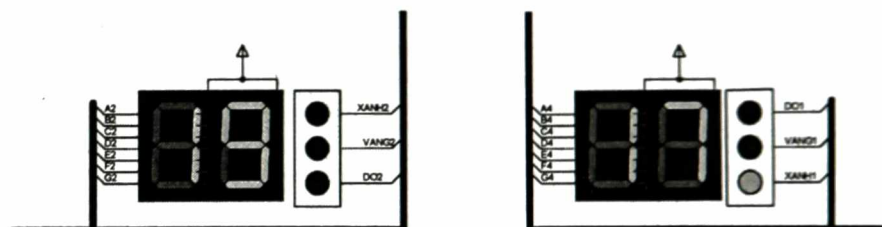


Hình 4.52. Mạch điều khiển

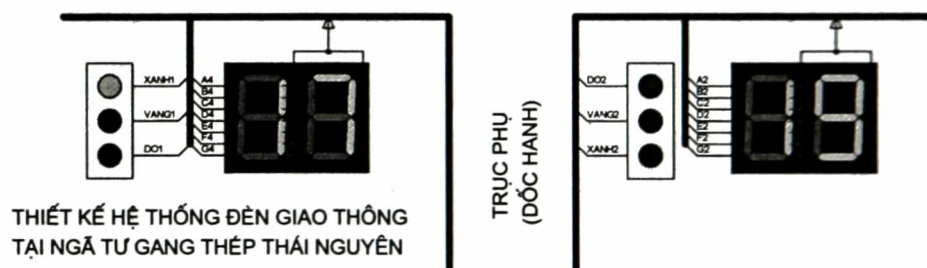
Đèn sáng khi bit điều khiển bằng 1.

+ Không yêu cầu hiển thị thời gian lên LED 7 thanh

+ Yêu cầu hiển thị thời gian lên LED 7 thanh



TRỰC CHÍNH (ĐƯỜNG CÁCH MẠNG THẮNG 8)



THIẾT KẾ HỆ THỐNG ĐÈN GIAO THÔNG
TẠI NGÃ TƯ GANG THÉP THÁI NGUYÊN

TRỰC PHỤ
(ĐỐC HẠNH)

Hình 4.53. Mạch động lực

Bài 4: Cho sơ đồ ghép nối 8051 với các thiết bị ngoại vi như hình 4.54 hãy:

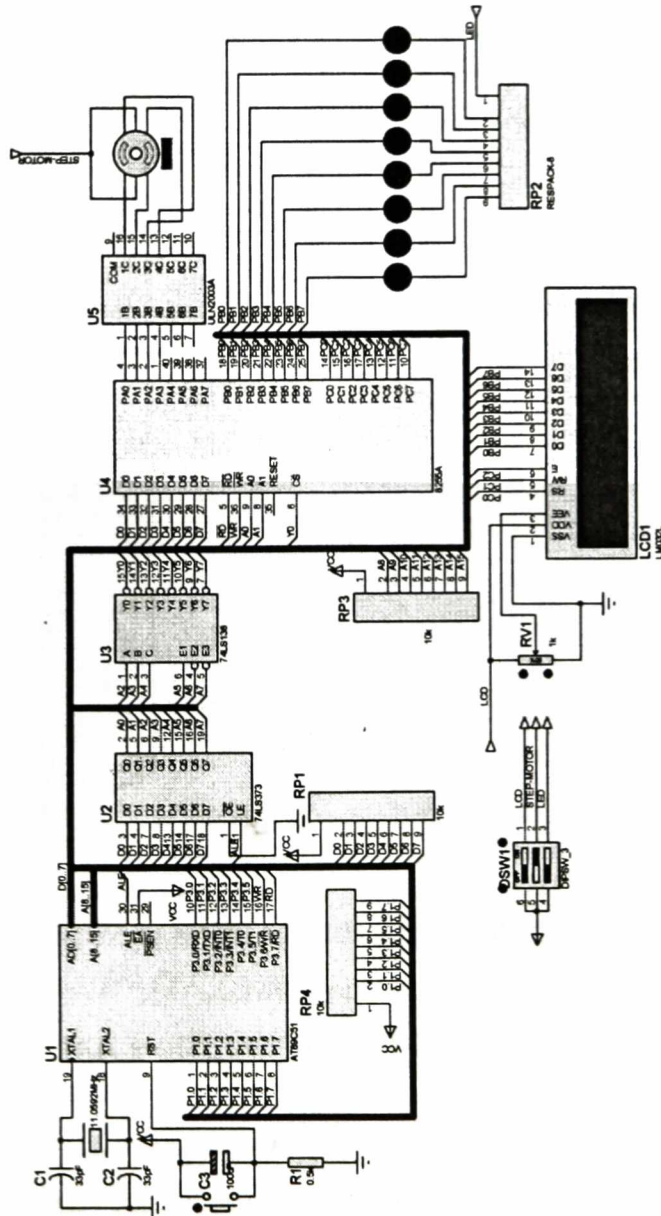
- Cho biết địa chỉ các thanh ghi 8255A.
- Viết chương trình điều khiển các LED liên tục sáng dần từ trái qua phải.

Bài 5: Vẫn cho sơ đồ ghép nối 8051 với các thiết bị ngoại vi như trên, hãy:

- Cho biết địa chỉ các thanh ghi 8255A.
- Viết chương trình điều khiển động cơ bước quay liên tục với chuỗi xung 8 bước.

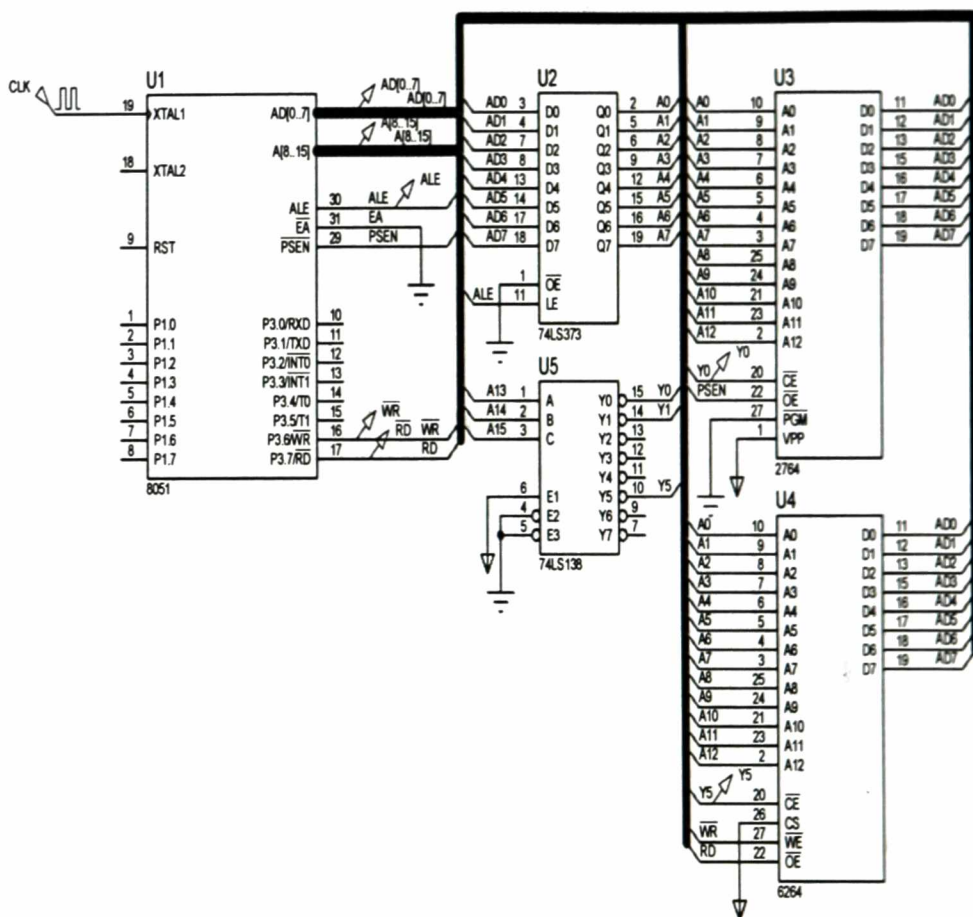
Bài 6: Vẽ sơ đồ ghép nối 8051 với các thiết bị ngoại vi như trên, hãy:

- Cho biết địa chỉ các thanh ghi 8255A.
- Viết chương trình hiển thị chữ “8051” lên LCD.



Hình 4.54. Hình vẽ bài 4, 5, 6

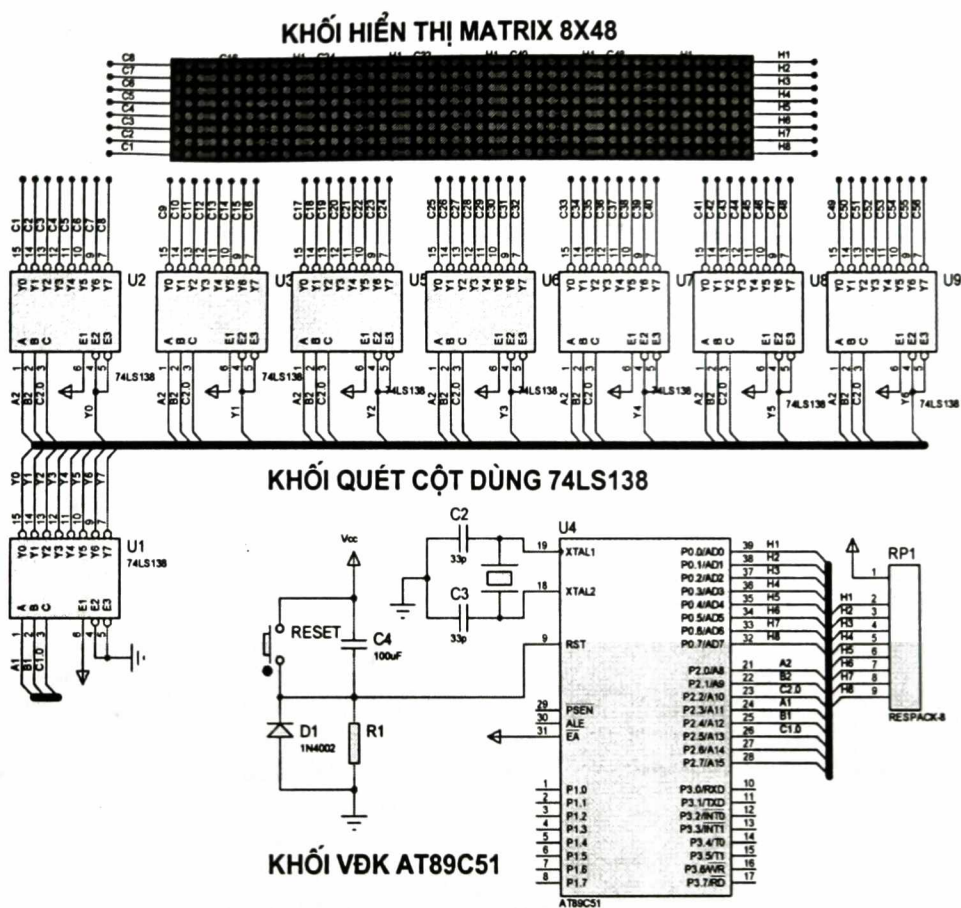
Bài 7: Cho sơ đồ ghép nối vi điều khiển 8051 với ROM, RAM như hình 4.55:



Hình 4.55. Hình vẽ bài 7

- Hãy cho biết vùng địa chỉ của ROM, RAM.
- Viết chương trình gửi liên tục dữ liệu 55h và AAh tới lần lượt ô nhớ đầu và cuối của RAM.

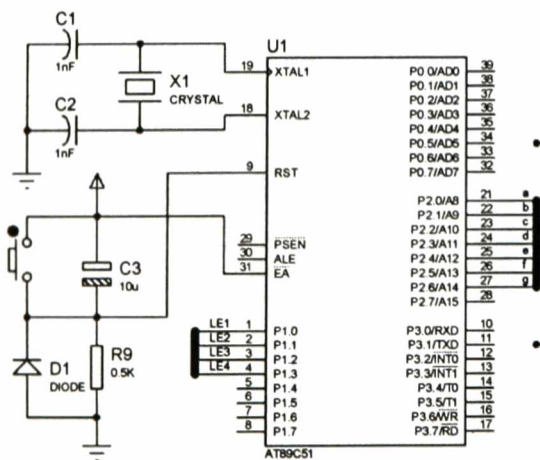
Bài 8: Cho sơ đồ ghép nối vi điều khiển 8051 ma trận LED như hình 4.56. Hãy viết chương trình hiển thị dòng chữ: “KHOA DIEN – DHKT CÔNG NGHIỆP”

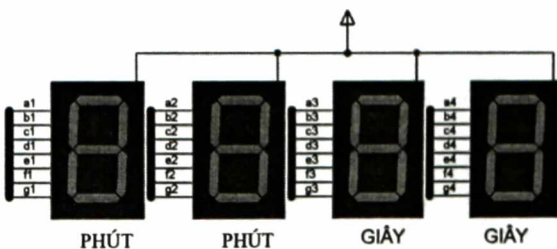
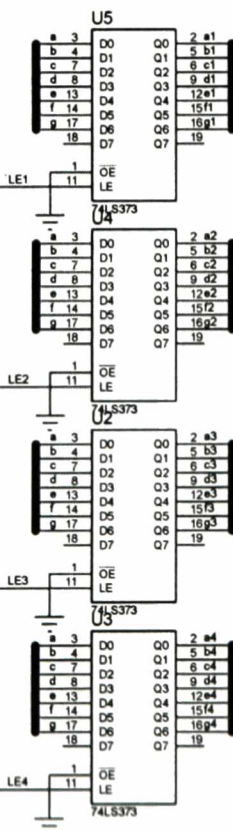


Hình 4.56. Hình vẽ bài 8

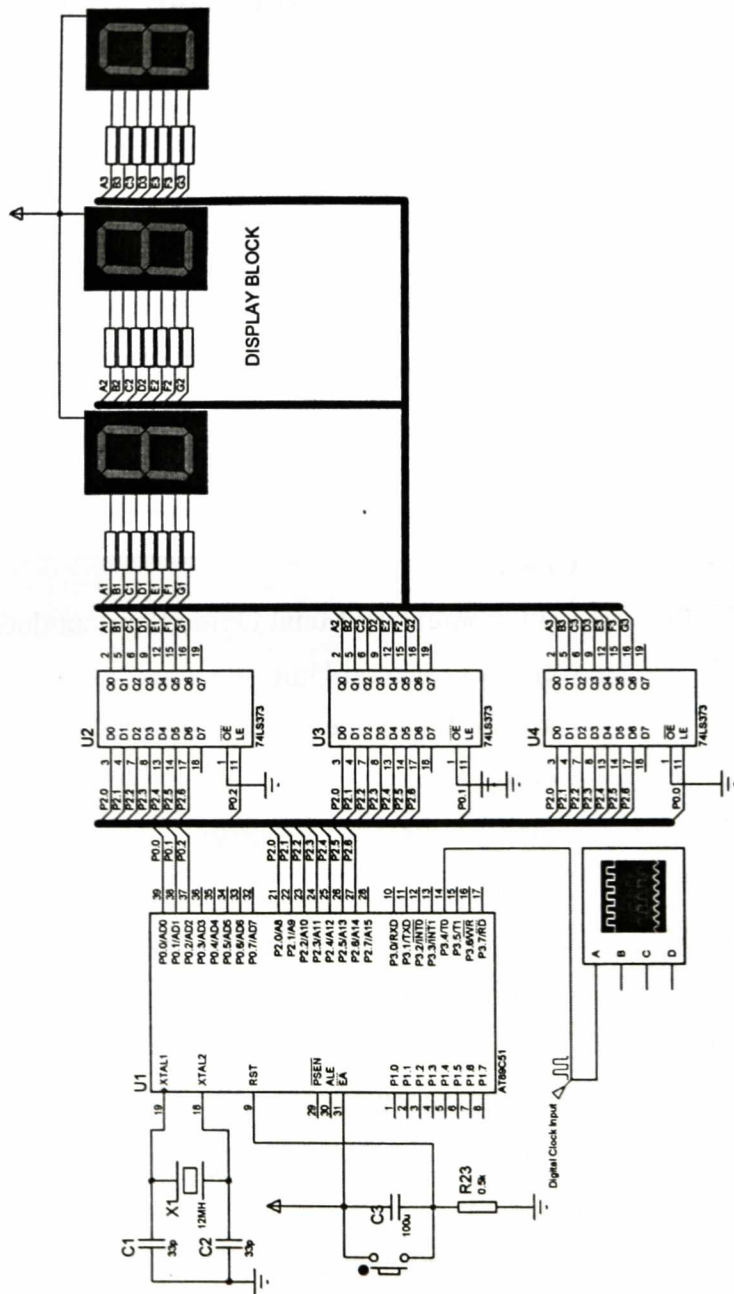
Bài 9: Vẽ và mô phỏng mạch điều khiển 4 LED 7 thanh bằng phương pháp chót. Điều khiển LED 7 thanh đếm PHÚT, GIÂY (dùng Timer tạo trễ) từ 00:00 đến 59:59

Hình 4.57. Hình vẽ bài 9





Bài 10: Lập trình cho bộ đếm 0 chế độ 1 đếm số xung từ cổng P3.4 hiển thị lên LED 7 đoạn



Hình 4.58. Hình vẽ bài 10

Phụ lục 1

Danh mục các từ viết tắt sử dụng trong giáo trình

TT	Từ viết tắt	Ý nghĩa
1	ADC	Analog Digital Converter
2	AEN	Address Enable
3	ALE	Address Last Enable
4	ALU	Arithnetic and Logic Unit
5	ASCII	American Standard Code for Information Interchange
6	BCD	Binary Code for Decimal
7	BIU	Bus Interface Unit
8	CLK	Clock
9	CMOS	Comprementary Metal Oxide Semiconductor
10	CPU	Central Processal Unit
11	CRT	Cathode Ray Tube
12	CU	Control Unit
13	DAC	Digital to Analog Converter
14	DEN	Data Enable
15	DRAM	Dinamic Random Access Memory
16	EU	Excutive Unit
17	GND	Ground
18	HLDA	Hold Acknowlege
19	I/O	In/Out
20	INTR	Interrupt
21	LCD	Liquid Crytal Display
22	LED	Light Emiting Diode

23	MCU	Micro Control Unit
24	MP	Micro Processor
25	NMI	None Markable Interrupt
26	NMOS	Negative Metal Oxide Semiconductor
27	PMOS	Positive Metal Oxide Semiconductor
28	PROM	Programmable Read Only Memory
29	RAM	Random Access Memory
30	ROM	Read only memory
31	SRAM	Static Random Access Memory

Phụ lục 2

Tập lệnh của vi điều khiển họ 8051

Nhóm lệnh số học (Mathematical (Arithmetic) Instructions)

Mnemonic	Instruction code	Hexa Decimal	Explanation
	D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀		
ADD A, Rn	0 0 1 0 1 n ₂ n ₁ n ₀	28 ÷ 2F	(A) ← (A) + (Rn)
ADD A, direct	0 0 1 0 0 1 0 1 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	25 Byte 2	(A) ← (A) + (direct)
ADD A, @Ri	0 0 1 0 0 1 1 I	26 ÷ 27	(A) ← (A) + ((Ri))
ADD A, #data	0 0 1 0 0 1 0 0 d ₇ d ₆ d ₅ d ₄ d ₃ d ₂ d ₁ d ₀	24 Byte 2	(A) ← (A) + #data
ADDC A, Rn	0 0 1 1 1 n ₂ n ₁ n ₀	38 ÷ 3F	(A) ← (A) + (Rn) + (C)
ADDC A, direct	0 0 1 1 0 1 0 1 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	35 Byte 2	(A) ← (A) + (direct) + (C)
ADDC A, @Ri	0 0 1 1 0 1 1 I	36 ÷ 37	(A) ← (A) + ((Ri)) + (C)
ADDC A, #data	0 0 1 1 0 1 0 0 d ₇ d ₆ d ₅ d ₄ d ₃ d ₂ d ₁ d ₀	34 Byte 2	(A) ← (A) + #data + (C)
SUBB A, Rn	1 0 0 1 1 n ₂ n ₁ n ₀	98 ÷ 9F	(A) ← (A) - (Rn) - (C)
SUBB A, direct	1 0 0 1 0 1 0 1 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	95 Byte 2	(A) ← (A) - (direct) - (C)
SUBB A, @Ri	1 0 0 1 0 1 1 I	96 ÷ 97	(A) ← (A) - ((Ri)) - (C)
SUBB A, #data	1 0 0 1 0 1 0 0 d ₇ d ₆ d ₅ d ₄ d ₃ d ₂ d ₁ d ₀	94 Byte 2	(A) ← (A) - #data - (C)
INC A	0 0 0 0 0 1 0 0	04	(A) ← (A) + 1
INC Rn	0 0 0 0 1 n ₂ n ₁ n ₀	08 ÷ 0F	(Rn) ← (Rn) + 1
INC direct	0 0 0 0 0 1 0 1 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	05 Byte 2	(direct) ← (direct) + 1

INC @Ri	0 0 0 0 0 1 1 I	06 ÷ 07	((Ri)) ← ((Ri)) + 1
INC dptr	1 0 1 0 0 0 1 1	A3	(dptr) ← (dptr) + 1
DEC A	0 0 0 1 0 1 0 0	14	(A) ← (A) - 1
DEC Rn	0 0 0 1 1 n ₂ n ₁ n ₀	18 ÷ 1F	(Rn) ← (Rn) - 1
DEC direct	0 0 0 1 0 1 0 1 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	15 Byte 2	(direct) ← (direct) - 1
DEC @Ri	0 0 0 1 0 1 1 I	16 ÷ 17	((Ri)) ← ((Ri)) - 1
MUL AB	1 0 1 0 0 1 0 0	A4	(B ₁₅₋₈), (A ₇₋₀) ← (A)×(B)
DIV AB	1 0 0 0 0 1 0 0	84	(A ₁₅₋₈), (B ₇₋₀) ← (A)/(B)
DA A	1 1 0 1 0 1 0 0	D4	Content of A là BCD

Nhóm lệnh Logic (Logic Instructions)

Mnemonic	Instruction code	Hexa	Explanation
	D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀	Decimal	
ANL A, Rn	0 1 0 1 1 n ₂ n ₁ n ₀	58 ÷ 5F	(A) ← (A) AND (Rn)
ANL A, direct	0 1 0 1 0 1 0 1 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	55 Byte 2	(A) ← (A) AND (direct)
ANL A, @Ri	0 1 0 1 0 1 1 I	56 ÷ 57	(A) ← (A) AND ((Ri))
ANL A, #data	0 1 0 1 0 1 0 0 d ₇ d ₆ d ₅ d ₄ d ₃ d ₂ d ₁ d ₀	54 Byte 2	(A) ← (A) AND #data
ANL direct, A	0 1 0 1 0 0 1 0 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	52 Byte 2	(direct) ← (direct) and (A)
ANL direct, #data	0 1 0 1 0 0 1 1 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀ d ₇ d ₆ d ₅ d ₄ d ₃ d ₂ d ₁ d ₀	53 Byte 2 Byte 3	(direct) ← (direct) and #data
ORL A, Rn	0 1 0 0 1 n ₂ n ₁ n ₀	48 ÷ 4F	(A) ← (A) OR (Rn)
ORL A, direct	0 1 0 0 0 1 0 1 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	45 Byte 2	(A) ← (A) OR (direct)
ORL A, @Ri	0 1 0 0 0 1 1 I	46 ÷ 47	(A) ← (A) OR ((Ri))

ORL A, #data	0 1 0 0 0 1 0 0 d7 d6 d5 d4 d3 d2 d1 d0	44 Byte 2	(A) ← (A) OR #data
ORL direct, A	0 1 0 0 0 0 1 0 a7 a6 a5 a4 a3 a2 a1 a0	42 Byte 2	(direct) ← (direct) OR (A)
ORL direct, #data	0 1 0 0 0 0 1 1 a7 a6 a5 a4 a3 a2 a1 a0 d7 d6 d5 d4 d3 d2 d1 d0	43 Byte 2 Byte 3	(direct) ← (direct) OR #data
XRL A, Rn	0 1 1 0 1 n2 n1 n0	68 ÷ 6F	(A) ← (A) XOR (Rn)
XRL A, direct	0 1 1 0 0 1 0 1 a7 a6 a5 a4 a3 a2 a1 a0	65 Byte 2	(A) ← (A) XOR (direct)
XRL A, @Ri	0 1 1 0 0 1 1 1	66 ÷ 67	(A) ← (A) XOR ((Ri))
XRL A, #data	0 1 1 0 0 1 0 0 d7 d6 d5 d4 d3 d2 d1 d0	64 Byte 2	(A) ← (A) XOR #data
XRL direct, A	0 1 1 0 0 0 1 0 a7 a6 a5 a4 a3 a2 a1 a0	62 Byte 2	(direct) ← (direct) XOR (A)
XRL direct, #data	0 1 1 0 0 0 1 1 a7 a6 a5 a4 a3 a2 a1 a0 d7 d6 d5 d4 d3 d2 d1 d0	63 Byte 2 Byte 3	(direct) ← (direct) XOR #data
CLR A	1 1 1 0 0 1 0 0	E4	(A) ← 0
CPL A	1 1 1 1 0 1 0 0	F4	(A) ← ($\bar{A}$)
RL A	0 0 1 0 0 0 1 1	23	Quay trái nội dung thanh ghi A
RLC A	0 0 1 1 0 0 1 1	33	Quay trái nội dung thanh ghi A với bit C
RR A	0 0 0 0 0 0 1 1	03	Quay phải nội dung thanh ghi A
RRC A	0 0 0 1 0 0 1 1	13	Quay phải nội dung thanh ghi A với bit C
SWAP A	1 1 0 0 0 1 0 0	C4	(A ₃₋₀) ↔ (A ₇₋₄)

Nhóm lệnh di chuyển dữ liệu (Data Transfer Instructions)

Mnemonic	Instruction code	Hexa decimal	Explanation
	D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀		
MOV A, Rn	1 1 1 0 1 n ₂ n ₁ n ₀	E8 ÷ EF	(A) ← (Rn)
MOV A, direct	1 1 1 0 0 1 0 1 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	E5 Byte 2	(A) ← (direct)
MOV A, @Ri	1 1 1 0 0 1 1 1	E6 ÷ E7	(A) ← ((Ri))
MOV A, #data	0 1 1 1 0 1 0 0 d ₇ d ₆ d ₅ d ₄ d ₃ d ₂ d ₁ d ₀	74 Byte 2	(A) ← #data
MOV Rn, A	1 1 1 1 1 n ₂ n ₁ n ₀	F8 ÷ FF	(Rn) ← (A)
MOV Rn, direct	1 0 1 0 1 n ₂ n ₁ n ₀ a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	A8 ÷ AF Byte 2	(Rn) ← (direct)
MOV Rn #data	0 1 1 1 1 n ₂ n ₁ n ₀ d ₇ d ₆ d ₅ d ₄ d ₃ d ₂ d ₁ d ₀	78 ÷ 7F Byte 2	(Rn) ← #data
MOV direct, A	1 1 1 1 0 1 0 1 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	F5 Byte 2	(direct) ← (A)
MOV direct, Rn	1 0 0 0 1 n ₂ n ₁ n ₀ a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	88 ÷ 8F Byte 2	(direct) ← (Rn)
MOV direct, direct	1 0 0 0 0 1 0 1 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀ a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	85 Byte 2 Byte 3	(direct) ← (direct) (source) (destination)
MOV direct, @Ri	1 0 0 0 0 1 1 1 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	86 ÷ 87 Byte 2	(direct) ← ((Ri))
MOV direct, #data	0 1 1 1 0 1 0 1 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀ d ₇ d ₆ d ₅ d ₄ d ₃ d ₂ d ₁ d ₀	75 Byte 2 Byte 3	(direct) ← #data
MOV @Ri, A	1 1 1 1 0 1 1 1	F6 ÷ F7	((Ri)) ← (A)

MOV @Ri, direct	1 0 1 0 0 1 1 1 a7 a6 a5 a4 a3 a2 a1 a0	A6 ÷ A7 Byte 2	((Ri)) ← (direct)
MOV @Ri, #data	0 1 1 1 0 1 1 i d7 d6 d5 d4 d3 d2 d1 d0	76 ÷ 77 Byte 2	((Ri)) ← (data)
MOV dptr, #data16	1 0 0 1 0 0 0 0 d15 d14 d13 d12 d11 d10 d9 d8 d7 d6 d5 d4 d3 d2 d1 d0	90 Byte 2 Byte 3	(dptr) ← #data ₁₅₋₀ (dpH) ← #data ₁₅₋₈ (dpL) ← #data ₇₋₀
MOVC A, @A + dptr	1 0 0 1 0 0 1 1	93	(A) ← ((A) + (dptr)) External Ram
MOVC A, @A + PC	1 0 0 0 0 0 1 1	83	(A) ← ((A) + (PC)) External Ram
MOVX A, @Ri	1 1 1 0 0 0 1 i	E2 ÷ E3	(A) ← ((Ri)) External Ram
MOVX A, @dptr	1 1 1 0 0 0 0 0	E0	(A) ← ((dptr)) External Ram
MOVX @Ri, A	1 1 1 1 0 0 1 i	F2 ÷ F3	((Ri)) ← (A)
MOVX @ dptr, A	1 1 1 1 0 0 0 0	F0	((dptr)) ← (A)
PUSH direct	1 1 0 0 0 0 0 0 a7 a6 a5 a4 a3 a2 a1 a0	C0 Byte 2	(SP) ← (SP) + 1 ((SP)) ← (direct)
POP direct	1 1 0 1 0 0 0 0 a7 a6 a5 a4 a3 a2 a1 a0	D0 Byte 2	(direct) ← ((SP)) (SP) ← (SP) - 1
XCH A, Rn	1 1 0 0 1 n2 n1 n0	C8 ÷ CF	(A) ↔ (Rn)
XCH A, direct	1 1 0 0 0 1 0 1 a7 a6 a5 a4 a3 a2 a1 a0	C5 Byte 2	(A) ↔ (direct)
XCH A, @Ri	1 1 0 0 0 1 1 i	C6 ÷ C7	(A) ↔ ((Ri))
XCHD A, @Ri	1 1 0 1 0 1 1 i	D6 ÷ D7	(A ₃₋₀) ↔ ((Ri ₃₋₀))

Nhóm lệnh xử lý bit (Bit Oriented Instructions)

Mnemonic	Instruction code	Hexa decimal	Explanation
	D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀		
CLR C	1 1 0 0 0 0 1 1	C3	(C) ← 0
CLR bit	1 1 0 0 0 0 1 0 b ₇ b ₆ b ₅ b ₄ b ₃ b ₂ b ₁ b ₀	C2 Byte 2	(bit) ← 0
SETB C	1 1 0 1 0 0 1 1	D3	(C) ← 1
SETB bit	1 1 0 1 0 0 1 0 b ₇ b ₆ b ₅ b ₄ b ₃ b ₂ b ₁ b ₀	D2 Byte 2	(bit) ← 1
CPL C	1 0 1 1 0 0 1 1	B3	(C) ← ($\bar{C}$)
CPL bit	1 0 1 1 0 0 1 0 b ₇ b ₆ b ₅ b ₄ b ₃ b ₂ b ₁ b ₀	B2 Byte 2	(bit) ← (/bit)
ANL C, bit	1 0 0 0 0 0 1 0 b ₇ b ₆ b ₅ b ₄ b ₃ b ₂ b ₁ b ₀	82 Byte 2	(C) ← (C) AND (bit)
ANL C, /bit	1 0 1 1 0 0 0 0 b ₇ b ₆ b ₅ b ₄ b ₃ b ₂ b ₁ b ₀	B0 Byte 2	(C) ← (C) AND (/bit)
ORL C, bit	0 1 1 1 0 0 1 0 b ₇ b ₆ b ₅ b ₄ b ₃ b ₂ b ₁ b ₀	72 Byte 2	(C) ← (C) OR (bit)
ORL C, /bit	1 0 1 0 0 0 0 0 b ₇ b ₆ b ₅ b ₄ b ₃ b ₂ b ₁ b ₀	A0 Byte 2	(C) ← (C) OR (/bit)
MOV C, bit	1 0 1 0 0 0 1 0 b ₇ b ₆ b ₅ b ₄ b ₃ b ₂ b ₁ b ₀	A2 Byte 2	(C) ← (bit)
MOV bit, C	1 0 0 1 0 0 1 0 b ₇ b ₆ b ₅ b ₄ b ₃ b ₂ b ₁ b ₀	92 Byte 2	(bit) ← (C)

Nhóm lệnh chuyển quyền điều khiển (rẽ nhánh) (Control Transfer Instructions)

Mnemonic	Instruction code	Hexa decimal	Explanation
	D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀		
ACALL addr 11	$A_{10} a_9 a_8 1 0 0 0 1$ $a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$	Byte 1 Byte 2	$(PC) \leftarrow (PC) + 2$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{7-0})$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{15-8})$ $(PC) \leftarrow \text{page address}$
LCALL addr 16	$0 0 0 1 0 0 1 0$ $a_{15} a_{14} a_{13} a_{12} a_{11} a_{10} a_9 a_8$ $a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$	12 Byte 2 Byte 3	$(PC) \leftarrow (PC) + 3$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{7-0})$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{15-8})$ $(PC) \leftarrow \text{addr}_{15-0}$
RET	$0 0 1 0 0 0 1 0$	22	$(PC_{15-8}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$ $(PC_{7-0}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$
RETI	$0 0 1 1 0 0 1 0$	32	$(PC_{15-8}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$ $(PC_{7-0}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$
AJMP addr 11	$a_{10} a_9 a_8 0 0 0 0 1$ $a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$	Byte 1 Byte 2	$(PC) \leftarrow (PC) + 2$ $(PC) \leftarrow \text{page address}$
LJMP addr 16	$0 0 0 0 0 0 1 0$ $a_{15} a_{14} a_{13} a_{12} a_{11} a_{10} a_9 a_8$ $a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$	02 Byte 2 Byte 3	$(PC) \leftarrow \text{addr}_{15-0}$

SJMP rel	1 0 0 0 0 0 0 0	r ₇ r ₆ r ₅ r ₄ r ₃ r ₂ r ₁ r ₀	Byte 2	(PC) ← (PC) + 2	80
JMP @A + dptr	0 1 1 1 0 0 1 1	0 1 1 1 0 0 1 1	73	(PC) ← (A) + (dptr)	73
JZ rel	0 1 1 0 0 0 0 0	r ₇ r ₆ r ₅ r ₄ r ₃ r ₂ r ₁ r ₀	Byte 2	(PC) ← (PC) + 2 IF (A) = 0 then	60
JNZ rel	0 1 1 1 0 0 0 0	r ₇ r ₆ r ₅ r ₄ r ₃ r ₂ r ₁ r ₀	Byte 2	(PC) ← (PC) + 2 IF (A) ≠ 0 then	70
JC rel	0 1 0 0 0 0 0 0	r ₇ r ₆ r ₅ r ₄ r ₃ r ₂ r ₁ r ₀	Byte 2	(PC) ← (PC) + 2 IF (C) = 1 then	40
JNC rel	0 1 0 1 0 0 0 0	r ₇ r ₆ r ₅ r ₄ r ₃ r ₂ r ₁ r ₀	Byte 2	(PC) ← (PC) + 2 IF (C) = 0 then	50
JB bit, rel	0 0 1 0 0 0 0 0	b ₇ b ₆ b ₅ b ₄ b ₃ b ₂ b ₁ b ₀	Byte 2	(PC) ← (PC) + 3 IF (bit) = 1 then	20
JNB bit, rel	0 0 1 1 0 0 0 0	b ₇ b ₆ b ₅ b ₄ b ₃ b ₂ b ₁ b ₀	Byte 2	(PC) ← (PC) + 3 IF (bit) = 0 then	30
JBC bit, rel	0 0 0 1 0 0 0 0	b ₇ b ₆ b ₅ b ₄ b ₃ b ₂ b ₁ b ₀	Byte 2	(PC) ← (PC) + 3 IF (bit) = 1 then (bit) ← 0	10
CJNE A, direct, rel	1 0 1 1 0 1 0 1	a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	Byte 2	(PC) ← 1 and IF (direct) > (A) then	Byte 3
		r ₇ r ₆ r ₅ r ₄ r ₃ r ₂ r ₁ r ₀	Byte 3	(PC) ← 0 and IF (direct) > (A) then	Byte 2
				(PC) ← (PC) + 3	Byte 3

CJNE A, #data, rel	$ \begin{array}{cccccccc} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ d_7 & d_6 & d_5 & d_4 & d_3 & d_2 & d_1 & d_0 \\ r_7 & r_6 & r_5 & r_4 & r_3 & r_2 & r_1 & r_0 \end{array} $	B4 Byte 2 Byte 3	$(PC) \leftarrow (PC) + 3$ IF #data < (A) then $(C) \leftarrow 0$ and $(PC) \leftarrow (PC) + rel$ IF #data > (A) then $(C) \leftarrow 1$ and $(PC) \leftarrow (PC) + rel$
CJNE Rn, #data, rel	$ \begin{array}{cccccccc} 1 & 0 & 1 & 1 & 0 & n_2 & n_1 & n_0 \\ d_7 & d_6 & d_5 & d_4 & d_3 & d_2 & d_1 & d_0 \\ r_7 & r_6 & r_5 & r_4 & r_3 & r_2 & r_1 & r_0 \end{array} $	B8 ÷ BF Byte 2 Byte 3	$(PC) \leftarrow (PC) + 3$ IF #data < (Rn) then $(C) \leftarrow 0$ and $(PC) \leftarrow (PC) + rel$ IF #data > (Rn) then $(C) \leftarrow 1$ and $(PC) \leftarrow (PC) + rel$
CJNE @Ri, #data, rel	$ \begin{array}{cccccccc} 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ d_7 & d_6 & d_5 & d_4 & d_3 & d_2 & d_1 & d_0 \\ r_7 & r_6 & r_5 & r_4 & r_3 & r_2 & r_1 & r_0 \end{array} $	B6 ÷ B7 Byte 2 Byte 3	$(PC) \leftarrow (PC) + 3$ IF #data < ((Ri)) then $(C) \leftarrow 0$ and $(PC) \leftarrow (PC) + rel$ IF #data > ((Ri)) then $(C) \leftarrow 1$ and $(PC) \leftarrow (PC) + rel$
DJNZ Rn, rel	$ \begin{array}{cccccccc} 1 & 1 & 0 & 1 & 1 & n_2 & n_1 & n_0 \\ r_7 & r_6 & r_5 & r_4 & r_3 & r_2 & r_1 & r_0 \end{array} $	D8 ÷ DF Byte 2	$(PC) \leftarrow (PC) + 2$ $(Rn) \leftarrow (Rn) - 1$ IF ((Rn)) ≠ 0 then $(PC) \leftarrow (PC) + rel$
DJNZ direct, rel	$ \begin{array}{cccccccc} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 \\ r_7 & r_6 & r_5 & r_4 & r_3 & r_2 & r_1 & r_0 \end{array} $	D5 Byte 2 Byte 3	$(PC) \leftarrow (PC) + 3$ $(direct) \leftarrow (direct) - 1$ IF (direct) ≠ 0 then $(PC) \leftarrow (PC) + rel$
NOP	$ \begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} $	00	$(PC) \leftarrow (PC) + 1$

TÀI LIỆU THAM KHẢO

- [1]; Nguyễn Tăng Cường, Phan Quốc Thắng, *Cấu trúc và lập trình họ vi điều khiển 8051*; Nhà xuất bản Khoa học và Kỹ thuật, 2004.
- [2]; Văn Thế Minh, *Kỹ thuật Vi xử lý*, Nhà xuất bản Giáo dục, 1997.
- [3]; Tống Văn On, *Họ Vi điều khiển 8051*, Nhà Xuất bản Lao động xã hội, 2001.
- [4]; Cao Xuân Tuyên, Vũ Quốc Đông, *Giáo trình Vi xử lý*, Đại học Kỹ thuật Công nghiệp, 2005.
- [5]; Nguyễn Đình Phú, *Kỹ thuật vi điều khiển*, Đại học Sư phạm kỹ thuật thành phố Hồ Chí Minh.
- [6]. John Uffenbeck, *The 8088/8086 family: Designing, programming and interfacing*, Pentice Hall, 1987.
- [7] Lâm Tăng Đức, Lê Tiến Dũng, *Giáo trình kỹ thuật Vi điều khiển*, Đại học Bách khoa.
- [8]; Một số tài liệu trên mạng Internet.
www.alldatasheet.com
www.8051.com
www.8052.com
www.dientuvietnam.net
www.diendandientu.com
www.ebook.edu.vn
www.8051projects.info

topprojects.blogspot.com/2008/02/8051-microcontroller-projects.html

nguyenvanbientbd47.110mb.com/home.html

giaoducvn.net/cdt/content/view/20/66/

.....

VÕ QUANG VINH - DƯƠNG QUỐC HƯNG

GIÁO TRÌNH HỆ VI ĐIỀU KHIỂN

Chịu trách nhiệm xuất bản: **TS. Phạm Văn Diễn**

Biên tập: **Lê Thị Hồng Thuỷ**

Trình bày bìa: **Thuỳ Dương**

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

70 Trần Hưng Đạo, Hà Nội

In 220 cuốn, khổ 15,5 x 22,5cm, tại Công ty Thanh bình.

Số đăng ký KHXB: 215-2010/CXB/368-17/KHKT, do CXB cấp ngày 05/3/2010.

Quyết định xuất bản: 113/QĐXB-NXBKHKT, ngày 25/06/2010.

In xong và nộp lưu chiểu quý III năm 2010.

