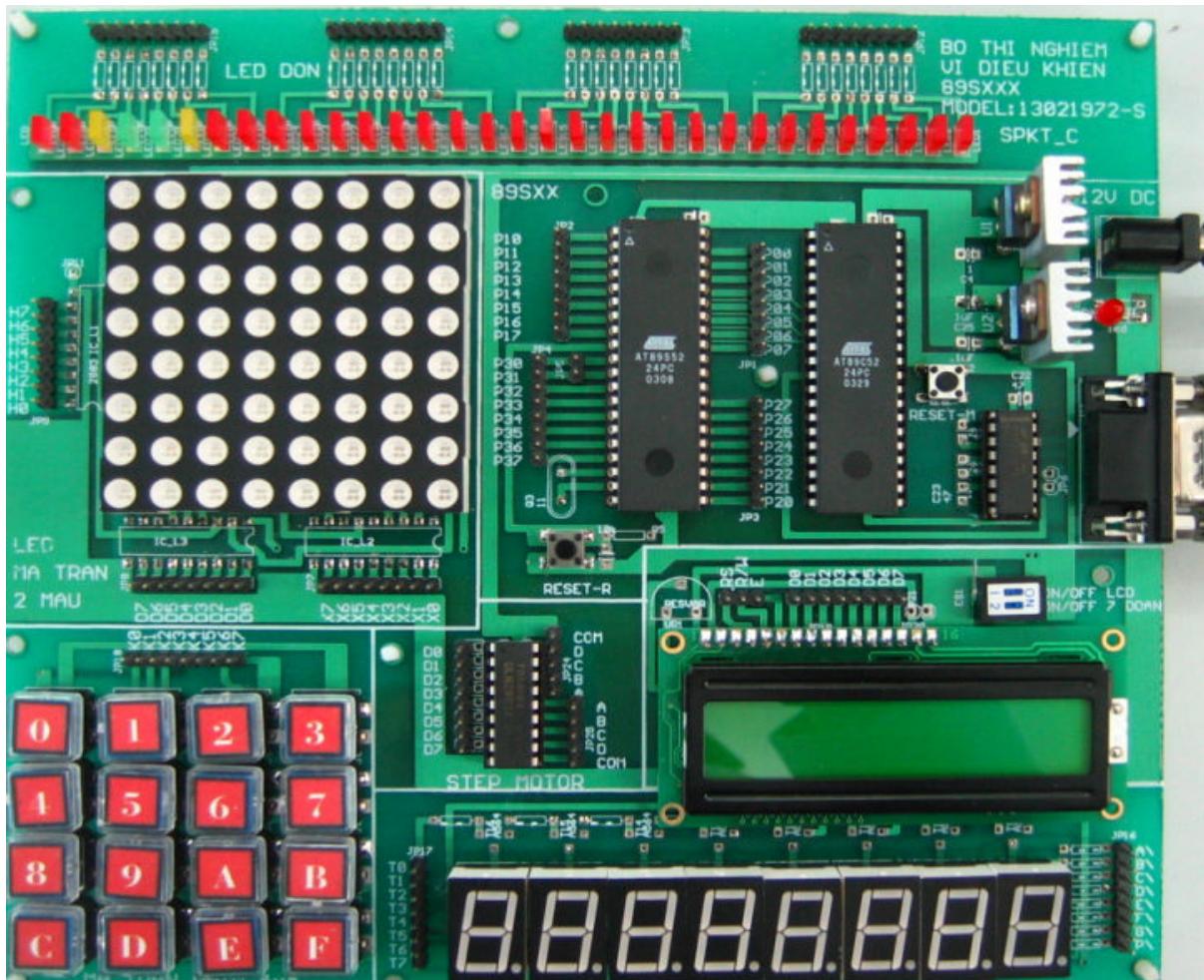


Trường Đại Học Sư Phạm Kỹ Thuật  
Khoa Điện Tử – Bộ môn Công Nghệ Viễn Thông

-----oo-----

# VI XỬ LÝ 1



NGUYỄN ĐÌNH PHÚ  
NĂM 2006

## *Chương 1*

# **GIỚI THIỆU LỊCH SỬ PHÁT TRIỂN VI XỬ LÝ**

### GIỚI THIỆU LỊCH SỬ PHÁT TRIỂN CỦA CÁC HỆ VI XỬ LÝ

1. GIỚI THIỆU LỊCH SỬ PHÁT TRIỂN CỦA CÁC VI XỬ LÝ
2. CHỨC NĂNG CỦA VI XỬ LÝ
3. MÁY VI TÍNH

### CÁC KHÁI NIÊM CƠ BẢN VỀ CẤU TRÚC CỦA VI XỬ LÝ

1. CHIỀU DÀI TỪ DỮ LIỆU.
2. KHẢ NĂNG TRUY XUẤT BỘ NHỚ.
3. TỐC ĐỘ LÀM VIỆC CỦA VI XỬ LÝ.
4. CÁC THANH GHI CỦA VI XỬ LÝ.
5. CÁC LỆNH CỦA VI XỬ LÝ.
6. CÁC KIỂU TRUY XUẤT BỘ NHỚ.
7. CÁC LOẠI BỘ NHỚ.
8. CÁC MẠCH ĐIỆN GIAO TIẾP NGOAI VI CỦA VI XỬ LÝ.

## *LỊỆT KÊ CÁC HÌNH*

**Hình 1-1. Cấu trúc của một máy vi tính.**

**Hình 1-2. Tổ chức mảng bộ nhớ 4x4.**

**Hình 1-3. Mạch hỗ trợ cho bộ nhớ.**

**Hình 1-4. Giản đồ thời gian đọc ghi bộ nhớ.**

## *LỊỆT KÊ CÁC BẢNG*

**Bảng 1-1. Tần số làm việc của một số vi xử lý.**

**Bảng 1-2. Các kiểu truy xuất bộ nhớ của các vi xử lý.**

**Bảng 1-3. Các IC ngoại vi giao tiếp với vi xử lý Z80.**

**Bảng 1-4. Các IC ngoại vi giao tiếp với vi xử lý 8088/80286.**

## I. GIỚI THIỆU LỊCH SỬ PHÁT TRIỂN CỦA CÁC HỆ VI XỬ LÝ:

### 1 GIỚI THIỆU LỊCH SỬ PHÁT TRIỂN CỦA CÁC HỆ VI XỬ LÝ:

Máy tính số là một hệ thống bao gồm các mạch điện tử xử lý tín hiệu dạng số được điều khiển bởi chương trình, có thể làm những công việc mà con người mong muốn. Chương trình sẽ điều khiển các mạch điện số cách di chuyển và xử lý dữ liệu (data) bằng cách điều khiển các mạch logic số học, các bộ nhớ (memory), các thiết bị xuất / nhập (Input/output). Cách thức các mạch điện logic của máy tính số kết hợp lại với nhau tạo thành các mạch logic số học, các vi mạch nhớ và các thiết bị xuất / nhập được gọi là cấu trúc.

Vi xử lý có cấu trúc giống như máy tính số và có thể xem nó là máy tính số vì cả hai đều tính toán dưới sự điều khiển của chương trình.

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

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

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

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

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

- Tháng 4 năm 1971, Intel sản xuất ra vi xử lý 4 bit có mã số 4004 có thể truy xuất 4096 ô nhớ 4 bit và có 45 lệnh.
- Tháng 4 năm 1972, Intel cải tiến và cho ra vi xử lý 8 bit có mã số 8008 có thể truy xuất 16K ô nhớ 8 bit và có 48 lệnh.
- Tháng 4 năm 1974, Intel cải tiến vi xử lý 8008 thành vi xử lý 8080 có thể truy xuất 64Kbyte bộ nhớ và có nhiều lệnh hơn, chạy nhanh gấp 10 lần so với 8008.
- Tháng 4 năm 1976, Intel cải tiến vi xử lý 8080 thành vi xử lý 8085 có thêm mạch tạo xung clock được tích hợp bên trong, có nhiều ngắt trên chip phục vụ cho nhiều ứng dụng và tích hợp mạch điều khiển hệ thống trên chip.
- Tháng 6 năm 1978, Intel sản xuất vi xử lý 16 bit có mã số là 8086: có 20 đường địa chỉ cho phép truy xuất 1MB bộ nhớ và bus dữ liệu bên trong và bên ngoài đều là 16bit.
- Tháng 6 năm 1979, Intel sản xuất vi xử lý 16 bit có mã số là 8088 chủ yếu dựa vào vi xử lý 8086 nhưng khác với vi xử lý 8086 là bus dữ liệu bên ngoài chỉ có 8

bit nhưng bus dữ liệu bên trong vi xử lý là 16 bit, mục đích cải tiến này nhằm hạ giá thành hệ thống và trở thành vi xử lý trong máy tính IBM-PC/XT.

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

## 2. **CHỨC NĂNG CỦA VI XỬ LÝ:**

Vi xử lý dùng các cổng logic giống như các cổng logic được sử dụng trong đơn vị xử lý trung tâm (central processing unit) của máy tính số. Do cấu trúc giống như CPU và được xây dựng từ các mạch vi điện tử nên có tên là vi xử lý: microprocessor. Giống như CPU, microprocessor có các mạch điện tử cho việc điều khiển dữ liệu (data) và tính toán dữ liệu dưới sự điều khiển của chương trình. Ngoài ra microprocessor là một đơn vị xử lý dữ liệu.

**Công việc xử lý dữ liệu là chức năng chính của vi xử lý.** Việc xử lý dữ liệu bao gồm tính toán và điều khiển dữ liệu. Việc tính toán được thực hiện bởi các mạch điện logic được gọi là đơn vị xử lý logic số học (Arithmetic Logic Unit: ALU ) có thể thực hiện các phép toán như Add, Subtract, And, Or, Compare, Increment, Decrement.

ALU không thể thực hiện một phép toán mà không có dữ liệu, ví dụ ALU cộng 2 dữ liệu với nhau thì 2 dữ liệu phải đặt đúng vị trí trước khi cộng. ALU không thể thực hiện việc chuyển dữ liệu từ nơi này đến nơi khác. Để ALU có dữ liệu cho việc xử lý thì ngoài mạch điện ALU, vi xử lý còn có các mạch điện logic khác để điều khiển dữ liệu. Các mạch điện logic điều khiển dữ liệu sẽ di chuyển dữ liệu vào đúng vị trí để khởi ALU xử lý dữ liệu. Sau khi thực hiện xong, khối điều khiển sẽ di chuyển dữ liệu đến bất cứ nơi nào mong muốn.

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

từng 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 sau cùng thì các mạch logic điều khiển sẽ thực hiện lệnh sau khi mã giải mã.

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

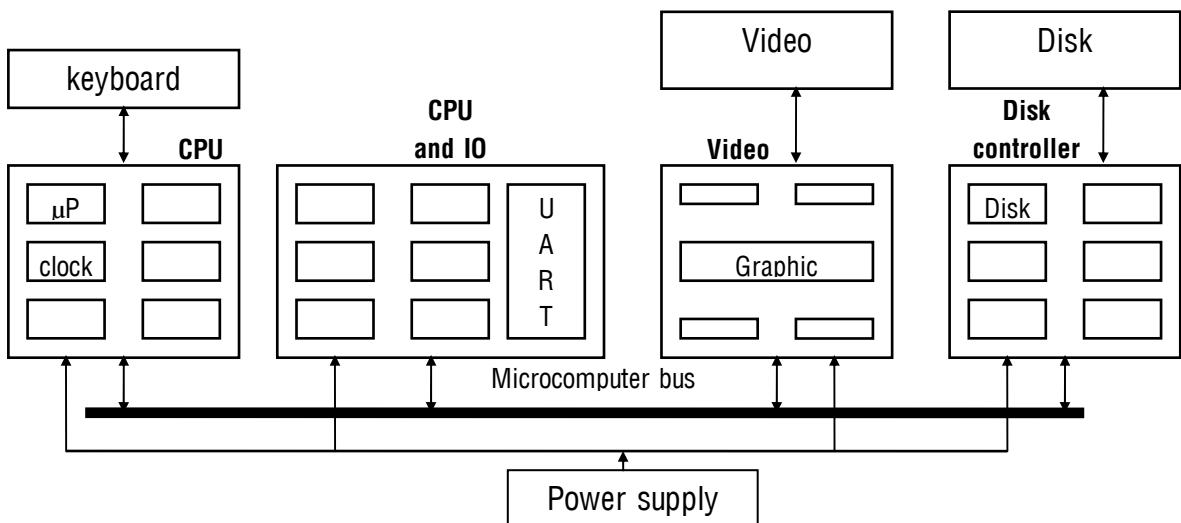
**Tóm tắt:** *Chức năng chính của vi xử lý là xử lý dữ liệu. Để làm được điều này vi xử lý phải có các mạch logic cho việc xử lý và điều khiển dữ liệu và các mạch logic điều khiển. Các mạch logic xử lý sẽ di chuyển dữ liệu từ nơi này sang nơi khác và thực hiện các phép toán trên dữ liệu, mạch logic điều khiển sẽ quyết định mạch điện nào cho việc xử lý dữ liệu. Vi xử lý thực hiện một lệnh với trình tự như sau: đón lệnh từ bộ nhớ, tiếp theo mạch logic điều khiển sẽ giải mã lệnh để xem lệnh đó yêu cầu vi xử lý thực hiện công việc gì, sau đó vi xử lý sẽ thực hiện đúng công việc của lệnh đã yêu cầu, quá trình này được gọi là chu kỳ đón - và - thực hiện lệnh (fetch / execute cycle).*

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

### 3. MÁY VI TÍNH (MICROCOMPUTER):

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

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



Hình 1-1. Cấu trúc của một máy vi tính.

Máy vi tính tổ chức theo card bao gồm: card CPU, card bộ nhớ RAM, card điều khiển đĩa, card điều khiển màn hình, ngoài ra máy vi tính còn có màn hình video, bàn phím...

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

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

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

Để giao tiếp với màn hình video cần phải có card video, bên cạnh các IC giao tiếp với bus của vi xử lý còn có các IC điều khiển màn hình Video. Màn hình Video dùng để hiển thị nội dung của một vùng nhớ đặc biệt trong bộ nhớ RAM do đó Card video có các IC RAM.

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

## II. CÁC KHÁI NIÊM CƠ BẢN VỀ CẤU TRÚC CỦA VI XỬ LÝ:

### 1 CHIỀU DÀI TỪ DỮ LIỆU:

Vi xử lý đầu tiên có chiều dài từ dữ liệu là 4 bit, tiếp theo là các vi xử lý 8 bit, 16 bit, 32 bit và 64 bit. Mỗi vi xử lý có chiều dài từ dữ liệu khác nhau sẽ có một khả năng ứng dụng khác nhau, các vi xử lý có chiều dài từ dữ liệu lớn, tốc độ làm việc nhanh, khả năng truy xuất bộ nhớ lớn được dùng trong các công việc xử lý dữ liệu, điều khiển phức tạp, các vi xử lý có chiều dài từ dữ liệu nhỏ hơn, khả năng truy xuất bộ nhớ nhỏ hơn, tốc độ làm việc thấp hơn được sử dụng trong các công việc điều khiển và xử lý đơn giản, chính vì thế các vi xử lý này vẫn tồn tại.

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

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

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

**Khái niệm Bus dữ liệu:** là đường truyền dữ liệu 2 chiều để chuyển dữ liệu giữa vi xử lý và các thành phần khác của hệ thống như bộ nhớ, IC ngoại vi. Vi xử lý 8 bit thì bus dữ liệu sẽ là 8 bit, vi xử lý 16 bit thì bus dữ liệu giao tiếp cũng là 16 bit ngoại trừ vi xử lý 8088.

## 2. KHẢ NĂNG TRUY XUẤT BỘ NHỚ:

Dung lượng bộ nhớ mà vi xử lý có thể truy xuất là một phần trong cấu trúc của vi xử lý. Các vi xử lý đầu tiên bị giới hạn về khả năng truy xuất bộ nhớ: vi xử lý 4004 có 14 đường địa chỉ nên có thể truy xuất được  $2^{14} = 16384$  ô nhớ, vi xử lý 8 bit có 16 đường địa chỉ nên có thể truy xuất được  $2^{16} = 65536$  ô nhớ, vi xử lý 16 bit có 20 đường địa chỉ nên có thể truy xuất  $2^{20} = 1024000$  ô nhớ, vi xử lý 32 bit như 386 hay 68020 có thể truy xuất 4 G ô nhớ. Vi xử lý có khả năng truy xuất bộ nhớ càng lớn nên có thể xử lý các chương trình lớn. Tùy theo ứng dụng cụ thể mà chọn một vi xử lý thích hợp.

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

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

## 3. TỐC ĐỘ LÀM VIỆC CỦA VI XỬ LÝ:

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

Vi xử lý	Tần số xung clock	Chiều dài từ dữ liệu
8051	12MHz	8-bit
Z80A	4MHz	8-bit
Z80B	6MHz	8-bit
286	16MHz	16-bit
486DX2-66	66Mhz	32-bit
Pentium	66MHz	32-bit

Bảng 1-1. Tần số làm việc của một số vi xử lý.

## 4. CÁC THANH GHI CỦA VI XỬ LÝ:

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

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

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

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

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

### 5. CÁC LỆNH CỦA VI XỬ LÝ:

Tập lệnh của vi xử lý là một trong những yếu tố cơ bản để đánh giá tốc độ làm việc của vi xử lý. Nếu vi xử lý có nhiều mạch điện logic bên trong để thực hiện thì số lệnh điều khiển của vi xử lý càng nhiều, khi đó vi xử lý càng lớn và độ phức tạp càng lớn. Ví dụ so sánh 2 tập lệnh của 2 vi xử lý 8 bit là 80C51 và Z80: vi xử lý 80C51 có 111 lệnh, vi xử lý Z80 có 178 lệnh. Tập lệnh của một vi xử lý càng nhiều càng rất có ích khi lập trình hay viết chương trình cho vi xử lý.

### 6. CÁC KIỂU TRUY XUẤT BỘ NHỚ:

Một yếu tố quyết định sự mềm dẻo trong lập trình là số lượng các kiểu truy xuất bộ nhớ khác nhau của vi xử lý, vi xử lý có nhiều kiểu truy xuất bộ nhớ sẽ có khả năng xử lý càng nhanh và cấu trúc các mạch điện bên trong càng phức tạp. Các kiểu truy xuất bộ nhớ của các vi xử lý 8 bit và 16 bit:

Kiểu truy xuất bộ nhớ ( Addressing mode )	Vi xử lý 6800	Vi xử lý Z80	Vi xử lý 8088
Implied - hiểu ngầm.	x	x	x
8-bit	x	x	x
16-direct	x	x	x
8-bit immediate	x	x	x
16-bit immediate	x	x	x
8-bit relative	x	x	x
8-bit index	x	x	x
16-bit index			x
Bit		x	x
8-bit indirect			x
16-bit indirect		x	x
16-bit computed			x
8-bit I/O		x	x
16-bit I/O			x

Bảng 1-2. Các kiểu truy xuất bộ nhớ của các vi xử lý.

Vi xử lý 16 bit và 32 bit có số lượng các kiểu truy xuất bộ nhớ rất lớn, tùy thuộc vào yêu cầu điều khiển mà chọn vi xử lý thích hợp.

### 7. BỘ NHỚ:

**Ô nhớ Bit** là một đơn vị nhớ nhỏ nhất, ô nhớ bit là ô nhớ chỉ có thể lưu được dữ liệu 1 bit là số nhị phân chỉ có 1 trong 2 giá trị ‘0’ hay ‘1’. Ô nhớ bit thường được tích hợp trong các vi xử lý điều khiển công nghiệp.

**Ô nhớ Byte** là một ô nhớ có thể chứa được dữ liệu số nhị phân 8 bit.

**Ô nhớ Word** là một ô nhớ có thể chứa được dữ liệu số nhị phân 16 bit – thường là sự kết hợp của 2 ô nhớ byte kết nối song song.

Mỗi một ô nhớ có 1 địa chỉ duy nhất, địa chỉ của bộ nhớ là số nhị phân và dữ liệu lưu trong ô nhớ cũng là số nhị phân.

#### Các loại bộ nhớ:

**Bộ nhớ RAM (Random Access Memory)** là loại bộ nhớ cho phép đọc ghi dữ liệu ở bất kỳ ô nhớ nào trong bộ nhớ. Điều này khác với bộ nhớ bộ nhớ băng từ là không thể đọc ghi tùy ý. Bộ nhớ bán dẫn được làm từ các flip flop, khi ta ghi dữ liệu vào flip flop thì dữ liệu sẽ được lưu lại và dữ liệu sẽ mất đi khi mất điện. Loại RAM này được gọi là SRAM.

**Bộ nhớ DRAM (Dynamic RAM = RAM động)** sử dụng tụ điện gate – source (cổng nguồn) của MOSFET để lưu trữ thông tin như điện tích trên tụ. Do có rò rỉ điện tích trong tụ điện nên khi sử dụng DRAM thì phải cần làm tươi sau mỗi một thời gian vài ms (thường khoảng 2 đến 10ms). Quá trình làm tươi bộ nhớ thường được thực hiện bằng IC điều khiển bộ nhớ động hoặc chế tạo sẵn những đường điều khiển làm tươi bộ nhớ trong CPU. Cả 2 loại bộ nhớ SRAM và DRAM đều là bộ nhớ bay hơi (mất điện thì mất dữ liệu).

**ROM (Read Only Memory)** là loại bộ nhớ chỉ đọc nội dung bộ nhớ đã được ghi trước, quá trình ghi dữ liệu được kết hợp với quá trình sản xuất. Dữ liệu do người dùng cung cấp cho nhà sản xuất. Dữ liệu trong bộ nhớ không bị mất khi mất điện. ROM thường được dùng để lưu các dữ liệu cố định – không thay đổi. Bộ nhớ này không cho phép xoá và ghi lại dữ liệu mới.

**PROM (Programmable ROM)** giống như bộ nhớ ROM nhưng việc ghi dữ liệu vào bộ nhớ độc lập với quá trình sản xuất, nhà sản xuất ra các PROM chưa có dữ liệu và người dùng có thể thực hiện việc ghi dữ liệu tùy ý vào bộ nhớ và chỉ được phép ghi 1 lần. Dữ liệu bên trong mặc nhiên là dữ liệu 1 và người lập trình thay đổi trạng thái từ 1 về 0 bằng cách đốt cầu chì bên trong PROM.

**EPROM (Erasable PROM)** giống như bộ nhớ PROM nhưng cho phép xoá và ghi lại dữ liệu nhiều lần, khi xoá phải dùng tia cực tím chiếu vào cửa sổ để khôi phục lại trạng thái dữ liệu 1. Sau khi ghi xong dữ liệu thì phải che cửa sổ lại để tránh tia cực tím làm mất dữ liệu.

**EEPROM (Electrically EPROM)** giống như bộ nhớ EPROM nhưng cho phép xoá bằng xung điện thay vì xoá bằng tia cực tím và cho phép ghi lại dữ liệu. Quá trình nạp xoá có thể thực hiện được từ 1000 đến 100000 lần.

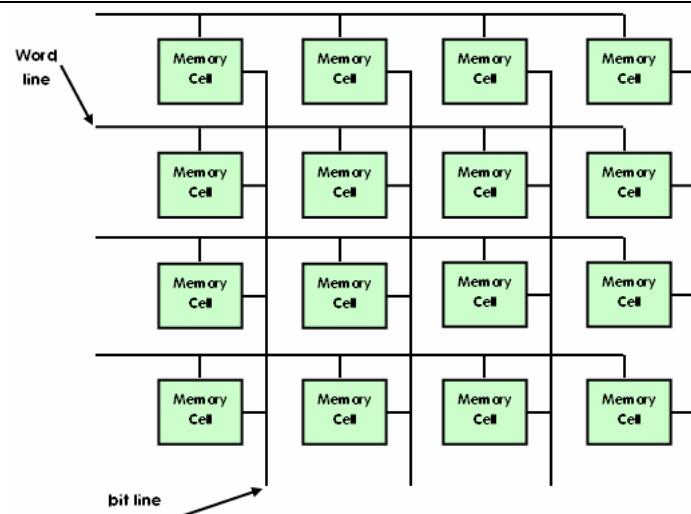
**Dung lượng bộ nhớ:** Một bộ nhớ có N đường địa chỉ và m đường dữ liệu thì sẽ có dung lượng là  $2^N \times m$ .

Dung lượng bộ nhớ được tính theo Kilobyte (KB), Megabyte (MB) và Gigabyte (GB).

#### Cấu trúc bộ nhớ:

**Mảng bộ nhớ (memory array):** phần chính của bộ nhớ là mảng bộ nhớ, mỗi hàng của các tế bào nhớ (memory cell) được điều khiển bởi đường từ (word line). Khi một hàng cụ thể được kích hoạt qua đường từ, các giá trị dữ liệu của những hàng tế bào của hàng đó có thể được đọc hay ghi vào qua các đường bit (bit line).

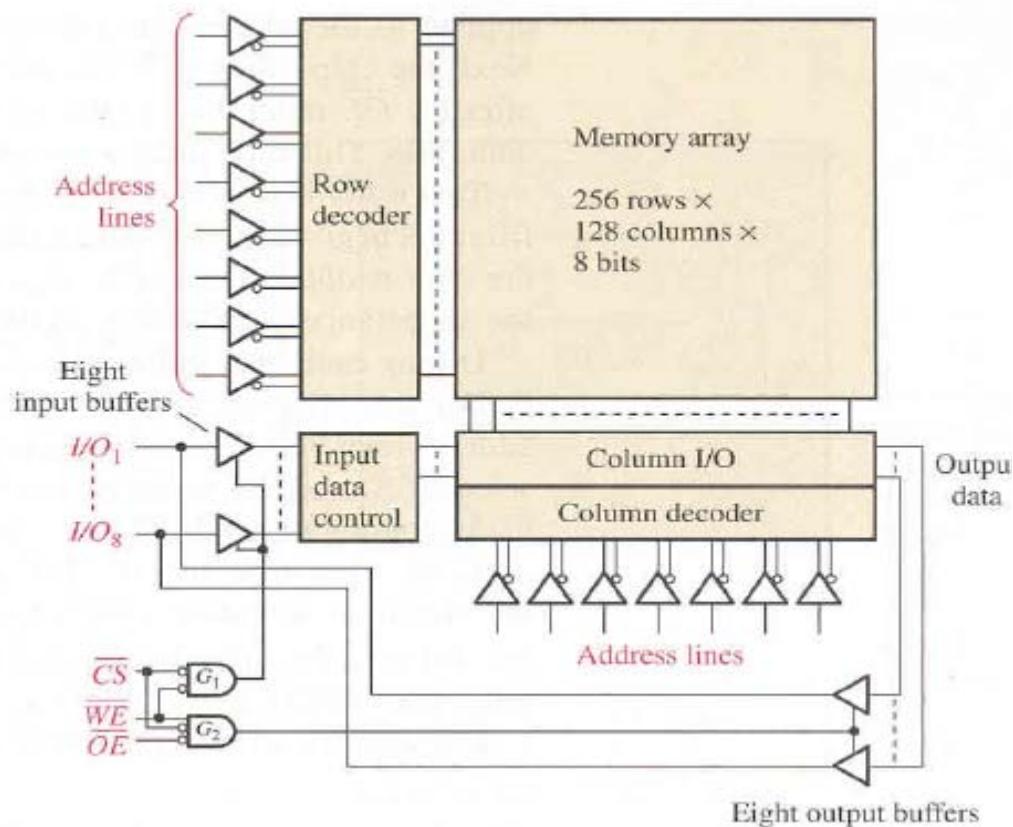
**Ví dụ:** Sơ đồ của bộ nhớ 16 bit được tổ chức theo mảng 4x4 như hình 1-2

Hình 1-2. Tổ chức mảng bộ nhớ  $4 \times 4$ .

Các kiểu bộ nhớ khác nhau được phân biệt chủ yếu dựa vào cách thức lưu trữ tế bào nhớ.

### Mạch hỗ trợ:

Ngoài mảng bộ nhớ còn có mạch hỗ trợ để giao tiếp với các thiết bị bên ngoài. Sơ đồ khái niệm một bộ nhớ như hình 1-3:



Hình 1-3. Mạch hỗ trợ cho bộ nhớ.

Mạch hỗ trợ có chức năng:

- Giải mã địa chỉ để xác định ô nhớ cần truy xuất.
- Cung cấp các tín hiệu điều khiển để đọc và ghi mảng bộ nhớ.

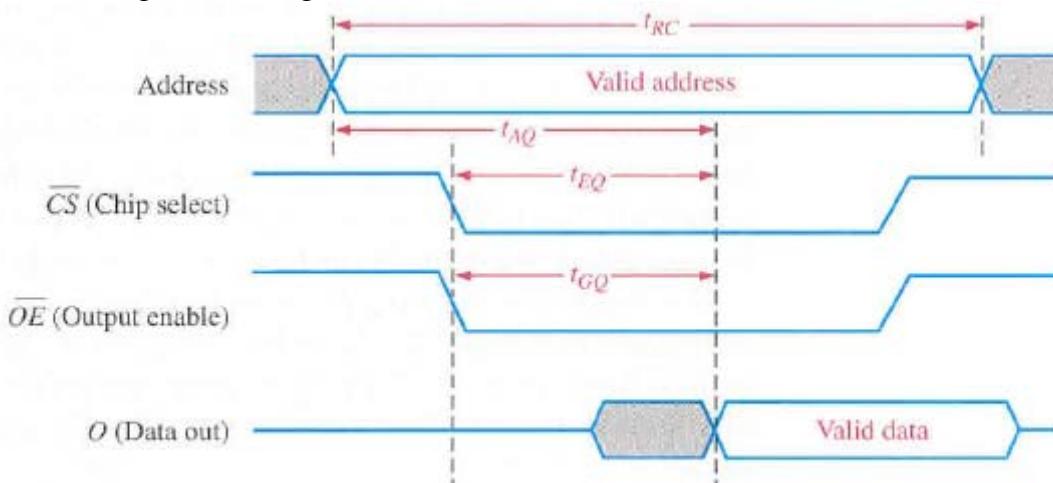
### Các tín hiệu điều khiển:

Tín hiệu  $\overline{CS}$  (chip select): là tín hiệu chọn chip: nếu  $\overline{CS} = 1$  thì bộ nhớ không được chọn và các đường dữ liệu sẽ ở trạng thái tổng trặc cao, nếu  $\overline{CS} = 0$  thì bộ nhớ được phép truy xuất ghi hay đọc.

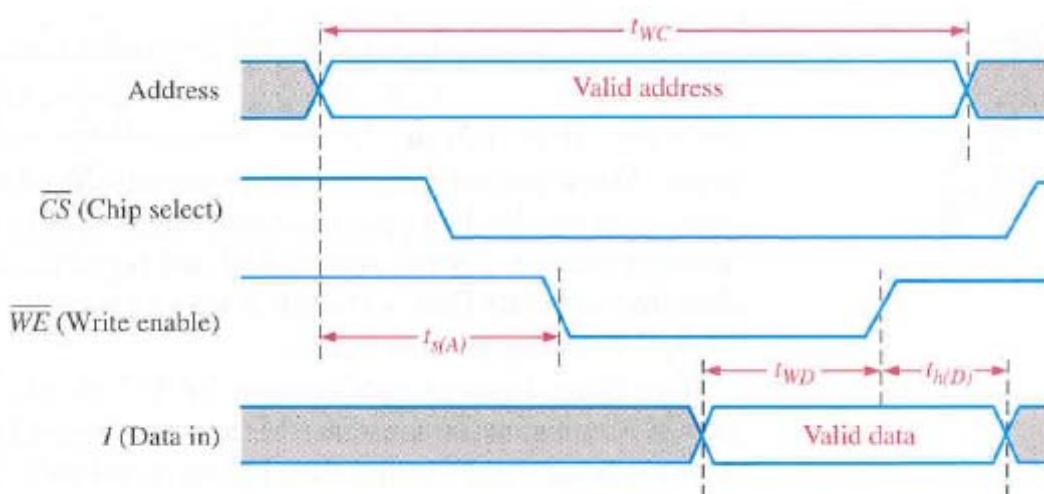
Tín hiệu  $\overline{OE}$  (Output Enable): cho phép xuất dữ liệu từ bộ nhớ đưa ra ngoài cho các thiết bị khác ví dụ như vi xử lý điều khiển chân  $\overline{OE}$  để yêu cầu bộ nhớ xuất dữ liệu.

Tín hiệu  $\overline{WR}$  (Write Enable): cho phép ghi dữ liệu từ bên ngoài và bộ nhớ ví dụ như vi xử lý điều khiển chân  $\overline{WR}$  để yêu cầu bộ nhớ nhận dữ liệu từ bus dữ liệu để cất vào ô nhớ.

Giản đồ thời gian đọc và ghi dữ liệu của bộ nhớ như hình 1-4:



(a) Read cycle ( $\overline{WE}$  HIGH)



(b) Write cycle ( $\overline{WE}$  LOW)

Hình 1-4. Giản đồ thời gian đọc ghi bộ nhớ.

### 8. CÁC MẠCH ĐIỆN GIAO TIẾP NGOẠI VI CỦA VI XỬ LÝ:

Ngoài giao tiếp với bộ nhớ, vi xử lý có các mạch điện giao tiếp với các mạch điện bên ngoài để điều khiển hay mở rộng khả năng điều khiển. Các mạch điện bên ngoài là các IC và được gọi là IC ngoại vi. Mỗi IC ngoại vi có một chức năng riêng, tùy thuộc vào yêu cầu điều khiển mà chọn các IC ngoại vi.

Bảng 1-3 liệt kê các IC ngoại vi có thể giao tiếp với Z80:

Mã số IC	Chức năng	dạng vỏ
8410	Direct memory access controller	40 pin -DIP
8420	Parallel input/output controller	40 pin -DIP
8430	Counter timer circuit	28 pin -DIP
8440	Serial input/output controller	40 pin -DIP
8470	Dual channel asynchronous receiver transmitter	40 pin -DIP
8530	Serial communications controller	40 pin -DIP

**Bảng 1-3. Các IC ngoại vi giao tiếp với vi xử lý Z80.**

Bảng 1-4 các IC ngoại vi có thể giao tiếp với 8088/80286:

Mã số IC	Chức năng	dạng vỏ
8087/80287	Arithmetic coprocessor	40 pin -DIP
8116	Dual baud rate clock generator (programmable)	18 pin -DIP
8202	Dynamic RAM controller	40 pin -DIP
8224	Clock generator/driver	16 pin -DIP
8250	Asynchronous communications element	40 pin -DIP
8253	Programmable interval timer	24 pin -DIP
8272	Floppy disk controller	40 pin -DIP

**Bảng 1-4. Các IC ngoại vi giao tiếp với vi xử lý 8088/80286.**

[return](#)

**the end**

## *Chương 2*

# CẤU TRÚC VI XỬ LÝ 8 bit VÀ TẬP LỆNH

### SƠ ĐỒ KHỐI CPU 8 BIT CƠ BẢN

- 1 SƠ ĐỒ KHỐI CỦA VI XỬ LÝ.
- 2 KHỐI ALU.

### TỔ CHỨC CÁC THANH GHI

- 1 CÁC THANH GHI BÊN TRONG VI XỬ LÝ.
- 2 CHỨC NĂNG CÁC THANH GHI.
  - a. Thanh ghi Accumulator
  - b. Thanh ghi bộ đếm chương trình – PC (program counter)
  - c. Thanh ghi trạng thái status register
  - d. Thanh ghi con trỏ ngăn xếp – SP (stack pointer)
  - e. Thanh ghi địa chỉ bộ nhớ
  - f. Thanh ghi lệnh – IR (instruction register)
  - g. Thanh ghi chứa dữ liệu tạm thời
  - h. Khối điều khiển logic và khối giải mã lệnh
  - i. Bus dữ liệu bên trong vi xử lý

### LỆNH CỦA VI XỬ LÝ

- 1 TẬP LỆNH CỦA VI XỬ LÝ.
- 2 TỪ GỌI NHỚ.
3. CÁC NHÓM LỆNH CƠ BẢN
4. CÁC KIỂU TRUY XUẤT ĐỊA CHỈ CỦA VI XỬ LÝ

### VI XỬ LÝ Z80

- 1 SƠ ĐỒ CẤU TRÚC BÊN TRONG CỦA VI XỬ LÝ Z80.
- 2 TỔ CHỨC THANH GHI BÊN TRONG VI XỬ LÝ Z80.
3. CHỨC NĂNG CÁC CHÂN CỦA VI XỬ LÝ Z80.
4. GẢI ĐỒ THỜI GIAN CỦA VI XỬ LÝ Z80.
5. ĐÁP ỨNG YÊU CẦU NGẮT CỦA VI XỬ LÝ Z80.
6. PHẦN CỨNG VÀ PHẦN MỀM CỦA VI XỬ LÝ Z80.

### VI XỬ LÝ 8085A

- 1 SƠ ĐỒ CẤU TRÚC BÊN TRONG CỦA VI XỬ LÝ 8085A.
- 2 TỔ CHỨC THANH GHI BÊN TRONG CỦA VI XỬ LÝ 8085A.
3. SƠ ĐỒ CHÂN CỦA VI XỬ LÝ 8085A.
4. GẢI ĐỒ THỜI GIAN CỦA VI XỬ LÝ 8085A.
5. GIẢI ĐẠ HỢP ĐỊA CHỈ VÀ DỮ LIỆU CHO VI XỬ LÝ 8085A

**6. TẬP LỆNH CỦA VI XỬ LÝ 8085A****TÓM TẮT – CÂU HỎI ÔN TẬP – BÀI TẬP**1 TÓM TẮT.2 CÂU HỎI ÔN TẬP – BÀI TẬP.***LIỆT KÊ CÁC HÌNH*****Hình 2-1. Sơ đồ cấu trúc bên trong của vi xử lý.****Hình 2-2. Sơ đồ minh họa các thanh ghi bên trong của Microprocessor được tô đậm.****Hình 2-3. Cấu trúc của một thanh ghi trạng thái.****Hình 2-4. Trước khi cộng dữ liệu.****Hình 2-5. Dữ liệu thanh ghi A được đưa đến thanh ghi Temp1.****Hình 2-6. Dữ liệu thanh ghi D được đưa đến thanh ghi Temp2.****Hình 2-7. Kết quả lưu trở lại thanh ghi A.****Hình 2-8. Các thanh ghi tạm trở lại trạng thái ban đầu.****Hình 2-9. Chu kỳ thực hiện lệnh của vi xử lý.****Hình 2-10. Cấu trúc lệnh của vi xử lý.****Hình 2-11. Sơ đồ cấu trúc bên trong của vi xử lý Z80.****Hình 2-12. Sơ đồ chân của vi xử lý Z80.****Hình 2-13. Sơ đồ chân của vi xử lý Z80.****Hình 2-14. Các chu kỳ máy của xử lý Z80.****Hình 2-15. Chu kỳ đón mã lệnh.****Hình 2-16. Chu kỳ đọc hoặc ghi bộ nhớ.****Hình 2-17. Chu kỳ đọc hoặc ghi thiết bị ngoại vi.****Hình 2-18. Chu kỳ yêu cầu bus/ trả lời bus.****Hình 2-19. Chu kỳ ngắn và đáp ứng yêu cầu ngắn.****Hình 2-20. Giản đồ thời gian của ngắn không ngắn được  $\overline{NMI}$ .****Hình 2-21. Giản đồ thời gian của lệnh HALT.****Hình 2-22. Hai bit flip flop IFF1 và IFF2 của ngắn  $\overline{INT}$ .****Hình 2-23. Đáp ứng ngắn ở mode 2.****Hình 2-24. Hệ thống tối thiểu dùng vi xử lý Z80.****Hình 2-25. Mở rộng thêm bộ nhớ.****Hình 2-26. Thêm một chu kỳ đợi vào chu kỳ M1.****Hình 2-27. Thêm một chu kỳ đợi để truy xuất bộ nhớ bất kỳ.****Hình 2-28. Thêm một chu kỳ đợi để truy xuất bộ nhớ bất kỳ.****Hình 2-29. Lệnh tổng quát "LD reg8D,reg8S".****Hình 2-30. Lệnh tổng quát "LD reg8,imm8".****Hình 2-31. Lệnh tổng quát "RL reg8".****Hình 2-32. Sơ đồ cấu trúc bên trong của vi xử lý 8085A.****Hình 2-33. Các thanh ghi bên trong VI xử lý 8085A.****Hình 2-34. Sơ đồ chân của vi xử lý 8085A.****Hình 2-35. Giản đồ thời gian hoạt động của vi xử lý 8085A.****Hình 2-36. Giải đa hợp địa chỉ và dữ liệu của vi xử lý 8085A.*****LIỆT KÊ CÁC BẢNG*****Bảng 2-1. Bảng liệt kê các trạng thái bit IFF1 và IFF2.****Bảng 2-2. Các trạng thái làm việc của vi xử lý 8085A.**

**Bảng 2-3. Thứ tự ưu tiên ngắn của vi xử lý 8085A.**

**Bảng 2-4. Mã các thanh ghi của vi xử lý 8085A.**

**Bảng 2-5. Mã các cặp thanh ghi 16 bit của vi xử lý 8085A.**

**Bảng 2-6. Các bit trong thanh ghi trạng thái của vi xử lý 8085A.**

**Bảng 2-7. Bảng vector địa chỉ ngắn của vi xử lý 8085A.**

**Bảng 2-8. Tóm tắt tập lệnh của vi xử lý 8085A.**

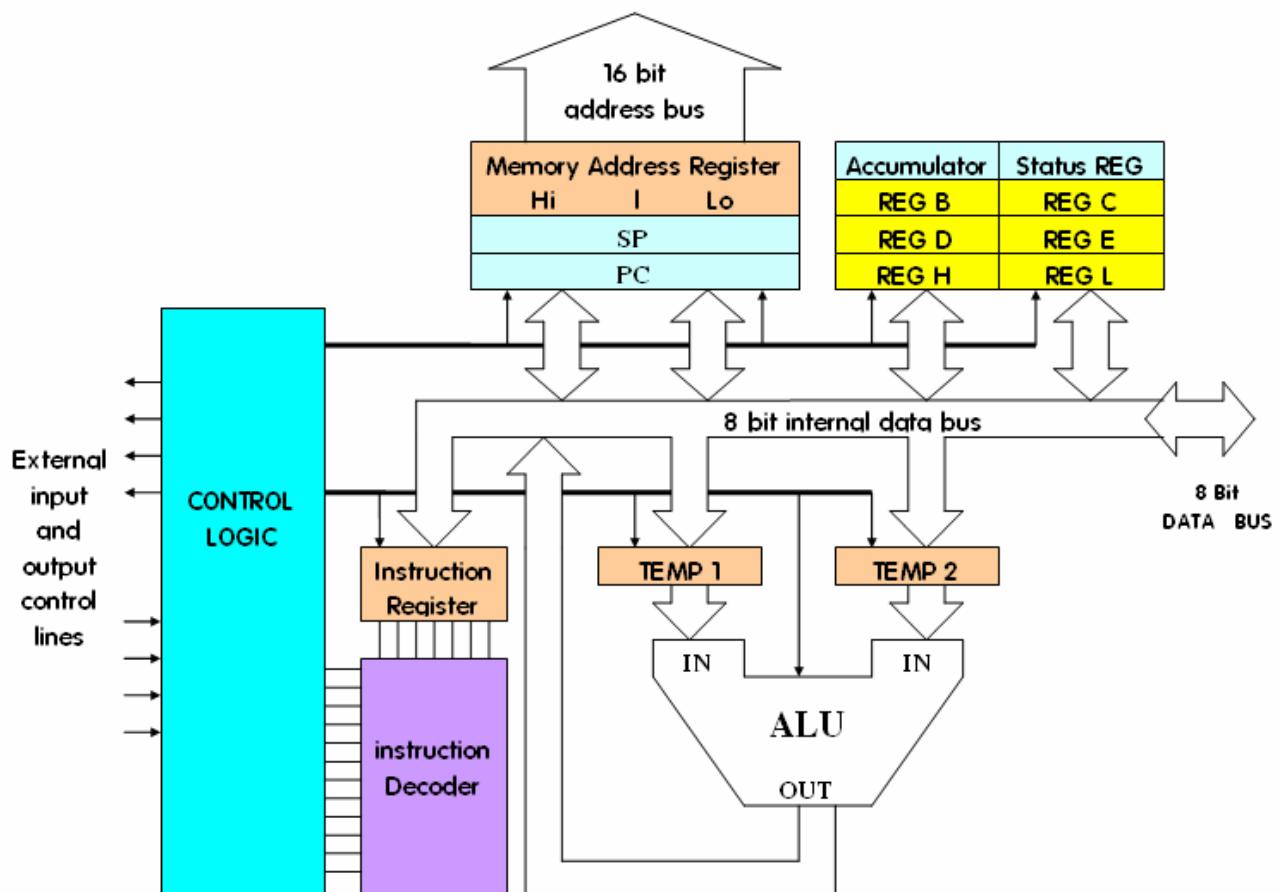
## I. SƠ ĐỒ KHỐI CPU 8 BIT CƠ BẢN:

### 1. SƠ ĐỒ KHỐI 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.

Sơ đồ khối của vi xử lý sẽ trình bày cấu trúc của một vi xử lý. Mỗi một vi xử lý khác nhau sẽ có cấu trúc khác nhau. Ví dụ vi xử lý 8 bit sẽ có cấu trúc khác với vi xử lý 16 bit...

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 khái rất tiện lợi và dễ trình bày nguyên lý hoạt động của vi xử lý. Hình 2-1 trình bày sơ đồ khái của vi xử lý 8 bit:



Hình 2-1. Sơ đồ cấu trúc bên trong của vi xử lý.

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

### 2. KHỐI ALU:

ALU là khối quan trọng nhất của vi xử lý, khối ALU chứa các mạch điện tử logic chuyên về xử lý dữ liệu. Khối ALU có 2 ngõ vào có tên là “IN” – là các ngõ vào dữ liệu cho ALU xử lý và 1 ngõ ra có tên là “OUT” – là ngõ ra kết quả dữ liệu sau khi ALU xử lý xong.

Dữ liệu trước khi vào ALU được chứa ở thanh ghi tạm thời (Temporarily Register) có tên là **TEMP1** và **TEMP2**. Bus dữ liệu bên trong vi xử lý được kết nối với 2 ngõ vào “IN” của ALU thông qua 2 thanh ghi tạm thời. Việc kết nối này cho phép ALU có thể lấy bất kỳ dữ liệu nào trên bus dữ liệu bên trong vi xử lý.

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

Ví dụ khi ALU cộng 2 dữ liệu thì một trong 2 dữ liệu được chứa trong thanh ghi Accumulator, sau khi phép cộng được thực hiện bởi ALU thì kết quả sẽ gởi trở lại thanh ghi Accumulator và lưu trữ ở thanh ghi này.

ALU xử lý một dữ liệu hay 2 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:

<b>Add</b>	<b>Complement</b>	<b>OR</b>	<b>Exclusive OR</b>
<b>Subtract</b>	<b>Shift right</b>	<b>Increment</b>	
<b>AND</b>	<b>Shift left</b>	<b>Decrement</b>	

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

## II. TỔ CHỨC CÁC THANH GHI:

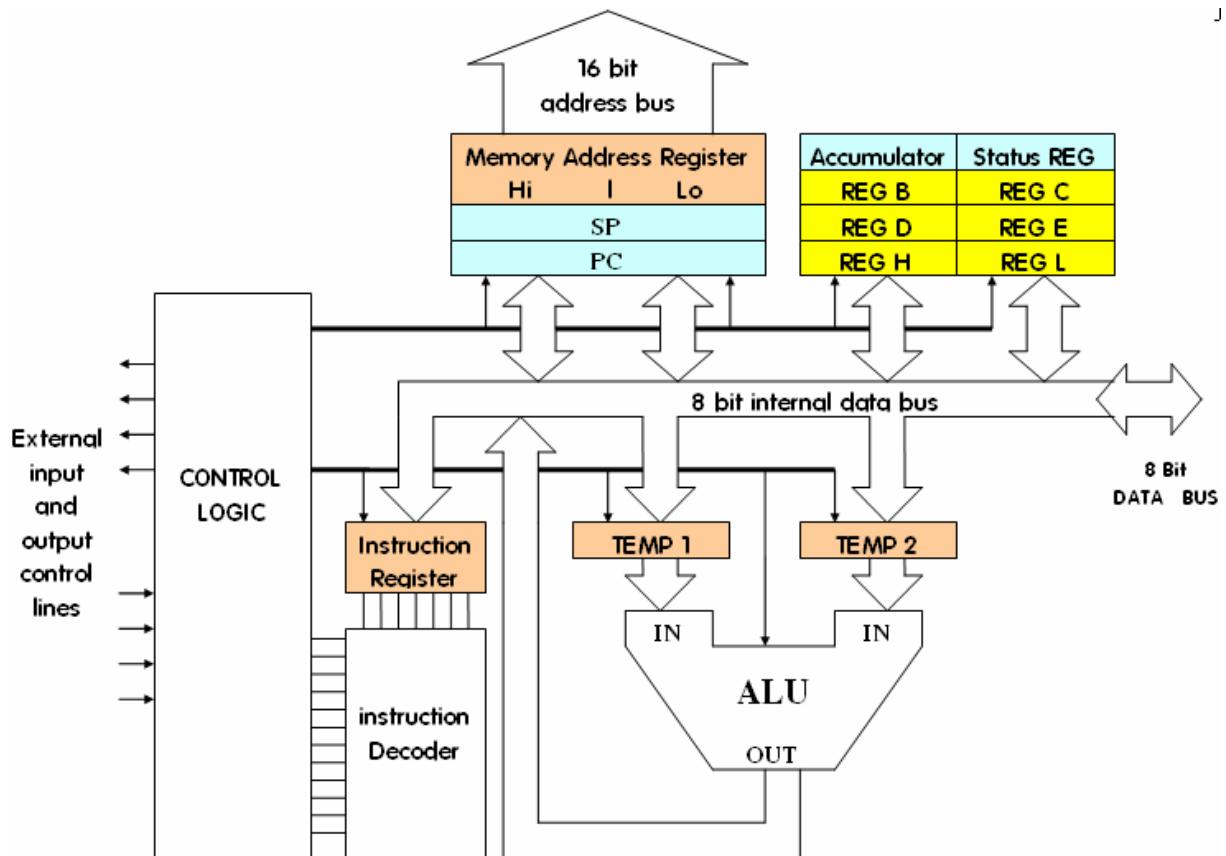
### 1 CÁC THANH GHI BÊN TRONG CỦA VI XỬ LÝ:

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** thực hiện các lệnh đặc biệt hay các chức năng đặc biệt, các thanh ghi còn lại gọi là các **thanh ghi thông dụng**. Với sơ đồ khái minh họa ở trên, các thanh ghi thông dụng có tên Reg B, Reg C, Reg D, Reg E.

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

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

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



Hình 2-2. Sơ đồ minh họa các thanh ghi bên trong của Microprocessor được tô đậm.

## 2. CHỨC NĂNG CỦA CÁC THANH GHI:

### a. Thanh ghi Accumulator:

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

Ví dụ khi thực hiện một lệnh cộng 1 dữ liệu A với một dữ liệu B, thì một dữ liệu phải chứa trong thanh ghi Accumulator giả sử là dữ liệu A, sau đó sẽ thực hiện lệnh cộng dữ liệu A (chứa trong Accumulator) với dữ liệu B (có thể chứa trong ô nhớ hoặc trong một thanh ghi thông dụng), kết quả của lệnh cộng là dữ liệu C sẽ được đặt trong thanh ghi A thay thế cho dữ liệu A trước đó.

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

Một chức năng quan trọng khác của thanh ghi Accumulator là để truyền dữ liệu từ bộ nhớ hoặc từ các thanh ghi bên trong của vi xử lý ra các thiết bị điều khiển bên ngoài thì dữ liệu đó phải chứa trong thanh ghi Accumulator.

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

### b. Thanh ghi bộ đếm chương trình PC (Program counter):

Thanh ghi PC là một thanh ghi có vai trò quan trọng nhất của vi xử lý. Chương trình là một chuỗi các lệnh nối tiếp nhau trong bộ nhớ của vi xử lý, các lệnh này sẽ yêu cầu vi xử lý thực hiện chính xác các công việc để giải quyết một vấn đề.

Từng lệnh phải đơn giản và chính xác và các lệnh phải theo đúng một trình tự để chương trình thực hiện đúng. Chức năng của thanh ghi PC là quản lý lệnh đang thực hiện và lệnh sẽ được thực hiện tiếp theo.

Thanh ghi PC trong vi xử lý có chiều dài từ dữ liệu lớn hơn chiều dài từ dữ liệu của vi xử lý. Ví dụ đối với các vi xử lý 8 bit có thể giao tiếp với 65536 ô nhớ thì thanh ghi PC phải có chiều dài là 16 bit để có thể truy xuất từng ô nhớ bắt đầu từ ô nhớ thứ 0 đến ô nhớ thứ 65535.

**Chú ý:** nội dung chứa trong thanh ghi PC chính là nội dung chứa trong thanh ghi địa chỉ.

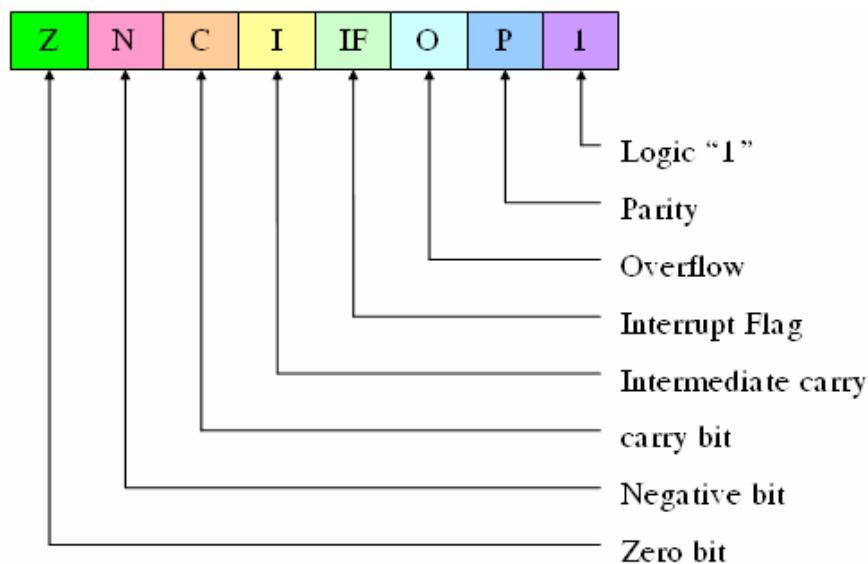
Trước khi vi xử lý thực hiện một chương trình thì thanh ghi PC phải được nạp một con số : **Đó chính là địa chỉ của ô nhớ chứa lệnh đầu tiên của chương trình**.

Địa chỉ của lệnh đầu tiên được gởi đến IC nhớ thông qua bus địa chỉ 16 bit. Sau đó bộ nhớ sẽ đặt nội dung của ô nhớ lên bus dữ liệu, nội dung này chính là mã lệnh, quá trình này gọi là đón lệnh từ bộ nhớ.

Tiếp theo vi xử lý tự động tăng nội dung của thanh ghi PC để chuẩn bị đón lệnh kế. PC chỉ được tăng khi vi xử lý bắt đầu thực hiện lệnh được đón trước đó. Lệnh đang thực hiện có chiều dài bao nhiêu byte thì thanh ghi PC tăng lên đúng bấy nhiêu byte.

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

### c. Thanh ghi trạng thái (Status Register):



Hình 2-3. Cấu trúc của một thanh ghi trạng thái.

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

Các bit thường có trong một thanh ghi trạng thái được trình bày ở hình 2-3.

Các lệnh xảy ra trong khối ALU thường ảnh hưởng đến thanh ghi trạng thái, ví dụ khi thực hiện một lệnh cộng 2 dữ liệu 8 bit, nếu kết quả lớn hơn  $1111111_2$  thì bit carry sẽ mang giá trị

là 1. Ngược lại nếu kết quả của phép cộng nhỏ hơn  $11111111_2$  thì bit carry bằng 0. Ví dụ lệnh tăng hay giảm giá trị của một thanh ghi, nếu kết quả trong thanh ghi khác 0 thì bit Z luôn bằng 0, ngược lại nếu kết quả bằng 0 thì bit Z bằng 1.

Ví dụ về rẽ nhánh khi kiểm tra bit trong thanh ghi trạng thái: hãy viết một chương trình giảm giá trị của một thanh ghi có giá trị là 10.

1. Nạp vào thanh ghi một số nhị phân có giá trị là 10.
2. Giảm nội dung của thanh ghi đi 1.
3. Kiểm tra bit Zero của thanh ghi trạng thái có bằng 1 hay không ?
4. Nếu không nhảy đến thực hiện tiếp lệnh ở bước 2
5. Nếu đúng kết thúc chương trình.

Ý nghĩa của các bit trong thanh ghi trạng thái:

**[a]. Carry/borrow (cờ tràn/mượn):** là bit carry khi thực hiện một phép cộng có giá trị tùy thuộc vào kết quả của phép cộng. Kết quả tràn thì bit carry =1, ngược lại bit carry = 0. Là bit borrow khi thực hiện một phép trừ: nếu số bị trừ lớn hơn số trừ thì bit borrow = 0, ngược lại bit borrow =1. Bit carry hay bit borrow là 1 bit chỉ được phân biệt khi thực hiện lệnh cụ thể.

**[b]. Zero:** bit Z bằng 1 khi kết quả của phép toán bằng 0, ngược lại bit Z=0.

**[c]. Negative (cờ số âm):** bit N = 1 khi bit MSB của thanh ghi có giá trị là 1, ngược lại N=0.

**[d]. Intermediate carry (cờ tràn phụ):** giống như bit Carry nhưng chỉ có tác dụng đối với phép cộng hay trừ 4 bit thấp.

**[e]. Interrupt Flag (cờ báo ngắt):** Bit IF có giá trị là 1 khi người lập trình muốn cho phép ngắt, ngược lại thì không cho phép ngắt.

**[f]. Overflow (cờ tràn số có dấu):** bit này bằng 1 khi bit tràn của phép toán cộng với bit dấu của dữ liệu.

**[g]. Parity (cờ chẵn lẻ):** bit này có giá trị là 1 khi kết quả của phép toán là số chẵn, ngược lại là số lẻ thì bit P = 0.

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

#### **d. Thanh ghi con trỏ ngăn xếp (Stack Pointer Register):**

Thanh ghi con trỏ ngăn xếp là một thanh ghi quan trọng của vi xử lý, độ dài từ dữ liệu của thanh ghi SP bằng thanh ghi PC, chức năng của thanh ghi SP gần giống như thanh ghi PC nhưng nó dùng để quản lý bộ nhớ ngăn xếp khi muốn lưu trữ tạm thời dữ liệu vào ngăn xếp.

Giống như thanh ghi PC, thanh ghi SP cũng tự động chỉ đến ô nhớ kế. Trong hầu hết các vi xử lý, thanh ghi SP giảm (để chỉ đến ô nhớ tiếp theo trong ngăn xếp) sau khi thực hiện lệnh cất dữ liệu vào ngăn xếp. Do đó khi thiết lập giá trị cho thanh ghi SP là địa chỉ cuối cùng của bộ nhớ.

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

#### e. Thanh ghi địa chỉ bộ nhớ (address Register):

Mỗi khi vi xử lý truy xuất bộ nhớ thì thanh ghi địa chỉ phải tạo ra đúng địa chỉ mà vi xử lý muốn. Ngõ ra của thanh ghi địa chỉ được đặt lên bus địa chỉ 16 bit. Bus địa chỉ dùng để lựa chọn một ô nhớ hay lựa chọn 1 port Input/Output.

Nội dung của thanh ghi địa chỉ ô nhớ và nội dung của thanh ghi PC là giống nhau khi vi xử lý truy xuất bộ nhớ để đón lệnh, khi lệnh đang được giải mã thì thanh ghi PC tăng lên để chuẩn bị đón lệnh tiếp theo, trong khi đó nội dung của thanh ghi địa chỉ bộ nhớ không tăng, trong suốt chu kỳ thực hiện lệnh, nội dung của thanh ghi địa chỉ phụ thuộc vào lệnh đang được thực hiện, nếu lệnh yêu cầu vi xử lý truy xuất bộ nhớ thì thanh ghi địa chỉ bộ nhớ được dùng lần thứ 2 trong khi thực hiện một lệnh.

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

#### f. Thanh ghi lệnh (instruction Register):

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

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

Đầu tiên là lệnh được đón từ bộ nhớ, sau đó PC chỉ đến lệnh kế trong bộ nhớ. Khi một lệnh được đón có nghĩa là dữ liệu trong ô nhớ đó được copy vào vi xử lý thông qua bus dữ liệu đến thanh ghi lệnh. Tiếp theo lệnh sẽ được thực hiện, trong khi thực hiện lệnh bộ giải mã lệnh đọc nội dung của thanh ghi lệnh. Bộ giải mã sẽ giải mã lệnh để báo cho vi xử lý thực hiện chính xác công việc mà lệnh yêu cầu.

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

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

#### g. Thanh ghi chứa dữ liệu tạm thời (Temporary data Register):

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

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

#### h. Khối điều khiển logic (control logic) và khối giải mã lệnh (instruction decoder):

Chức năng của khối giải mã lệnh 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.

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

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

### i. Bus dữ liệu bên trong vi xử lý:

Bus dữ liệu dùng để kết nối các thanh ghi bên trong và ALU với nhau, tất cả các dữ liệu di chuyển trong vi xử lý đều thông qua bus dữ liệu này. Các thanh ghi bên trong có thể nhận dữ liệu từ bus hay có thể đặt dữ liệu lên bus nên bus dữ liệu này là bus dữ liệu 2 chiều. Bus dữ liệu bên trong có thể kết nối ra bus bên ngoài khi vi xử lý cần truy xuất dữ liệu từ bộ nhớ bên ngoài hay các thiết bị IO. Bus dữ liệu bên ngoài cũng là bus dữ liệu 2 chiều vì vi xử lý có thể nhận dữ liệu từ bên ngoài hay gởi dữ liệu ra.

Để biết trình tự làm việc của bus dữ liệu bên trong vi xử lý hoạt động, hãy cho vi xử lý thực hiện một lệnh cộng 2 số nhị phân chứa trong thanh ghi 2 thanh ghi: thanh ghi Accumulator (gọi tắt là A) = 1101 1110<sub>2</sub> và thanh ghi D=1101 1010<sub>2</sub>.

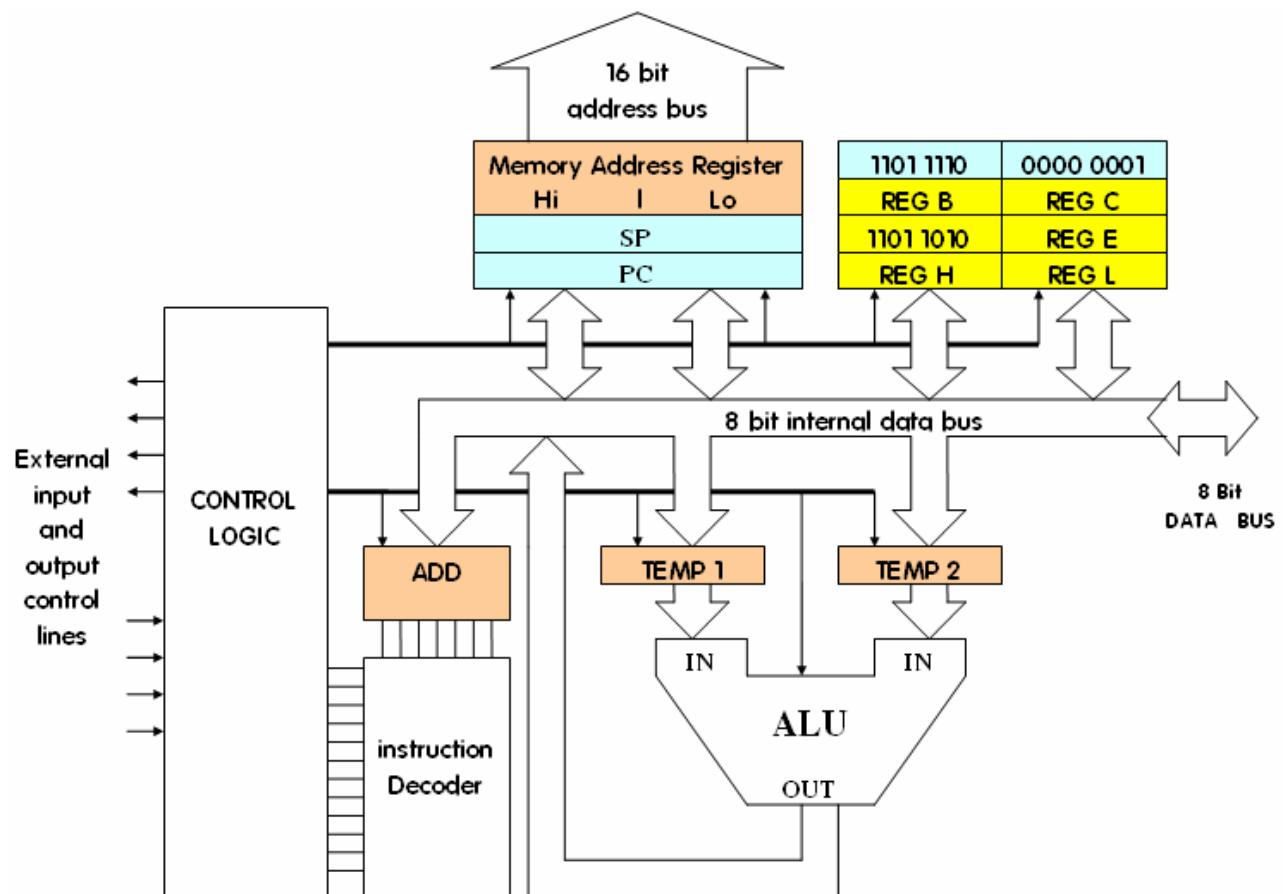
Trình tự cộng như sau:

- ◆ Trước khi thực hiện lệnh cộng, nội dung của 2 thanh ghi phải chứa 2 dữ liệu và 2 thanh ghi này có thể đang kết nối với các thiết bị khác. Để thực hiện lệnh cộng nội dung 2 thanh ghi A và D thì thanh ghi lệnh phải mang đúng mã lệnh của phép cộng này và giả sử mã lệnh đó là ADD. Được trình bày ở hình 2-4.
- ◆ Dữ liệu của thanh ghi A được đặt lên bus dữ liệu bên trong vi xử lý, một trong 2 thanh ghi lưu trữ dữ liệu tạm thời được kết nối với Bus dữ liệu. Thanh ghi tạm thời sẽ copy dữ liệu chứa trong thanh ghi A. Chỉ có thanh ghi A và thanh ghi tạm thời được kết nối với bus tại thời điểm này. Xem hình 2-5
- ◆ Dữ liệu của thanh ghi D được kết nối với bus dữ liệu và thanh ghi tạm thời còn lại cũng được phép kết nối với bus dữ liệu. Thanh ghi tạm thời sẽ copy nội dung của thanh ghi D. Chỉ có thanh ghi D và thanh ghi tạm thời được kết nối với bus tại thời điểm này. Xem hình 2-6.
- ◆ ALU sẽ cộng trực tiếp 2 dữ liệu tại 2 ngõ vào. Ngõ ra của ALU được kết nối với thanh ghi A, kết quả của phép cộng được nạp vào thanh ghi A. Xem hình 2-7.
- ◆ Sau khi đặt kết quả vào trong thanh ghi A và cắp nhật sự thay đổi các bit trong thanh ghi trạng thái thì sự kết nối giữa thanh A và khối ALU chấm dứt, các thanh ghi tạm thời trở lại trạng thái sẵn sàng cho lệnh tiếp theo. Xem hình 2-8

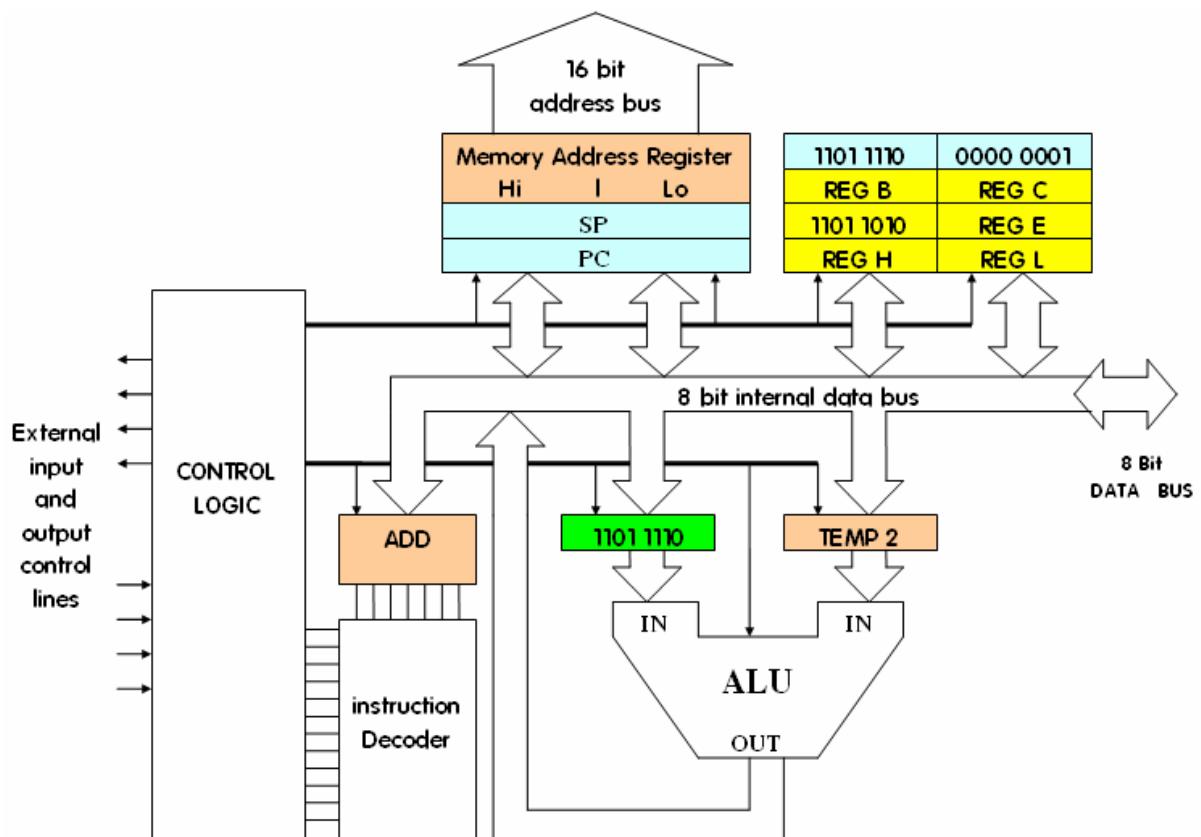
Phép cộng 2 số nhị phân:

$$\begin{array}{r}
 1101\ 1110 \\
 +\ 1101\ 1010 \\
 \hline
 1\ 1011\ 1000
 \end{array}$$

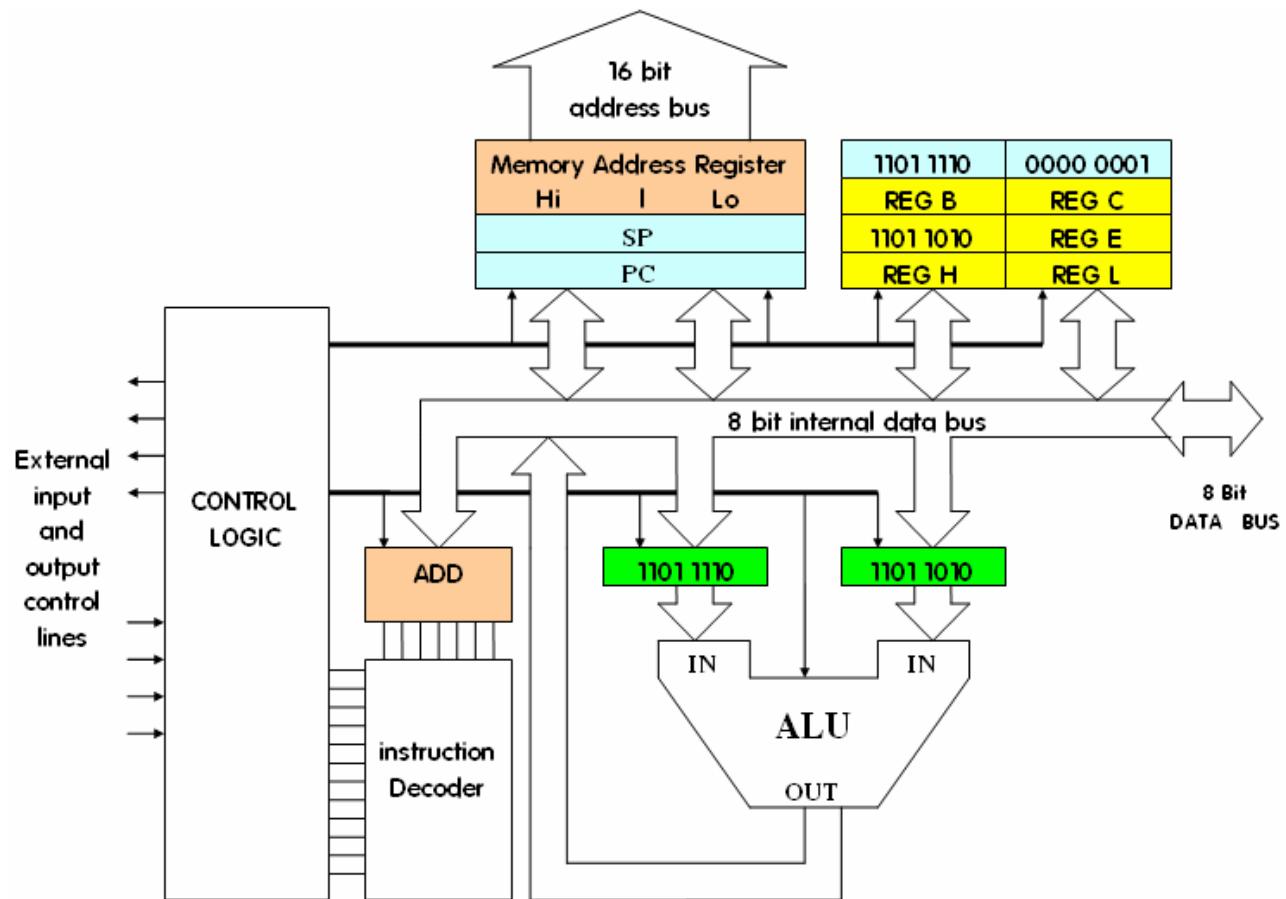
Carry negative



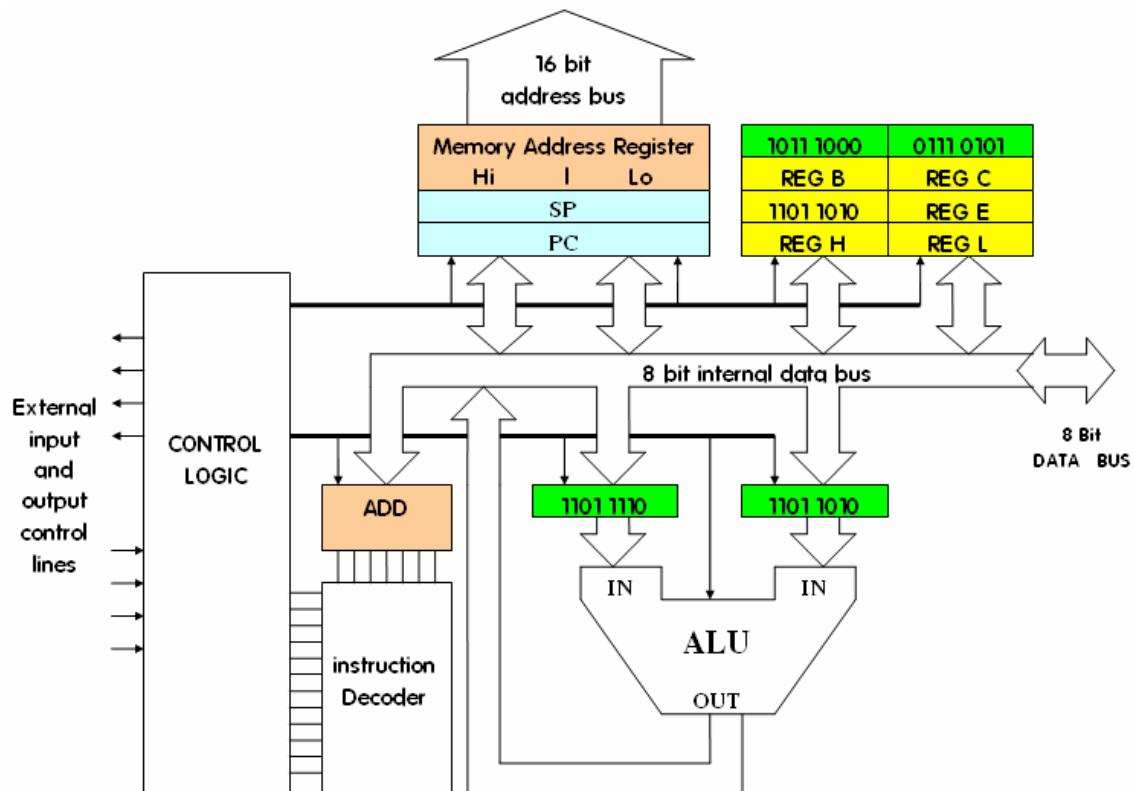
Hình 2-4. Trước khi cộng dữ liệu.



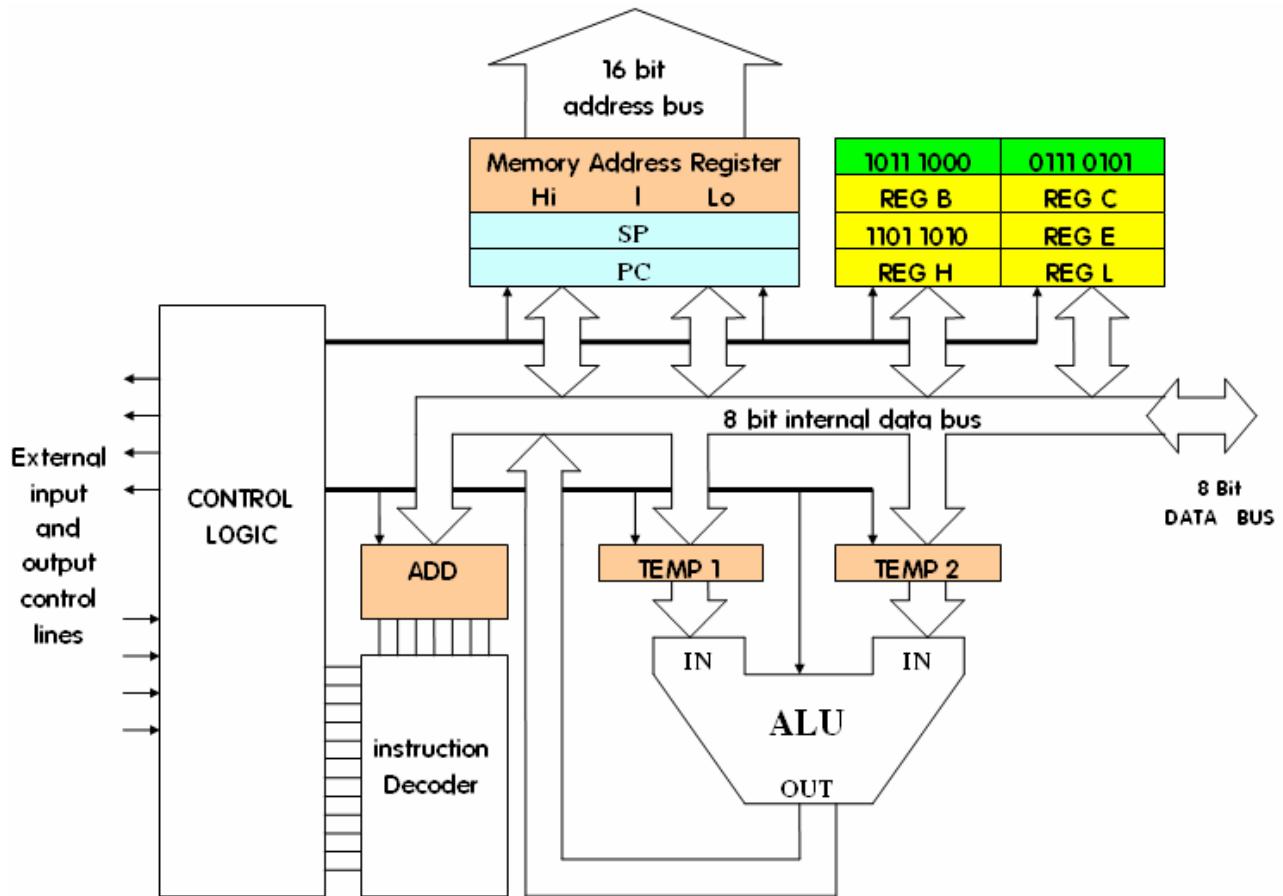
Hình 2-5. Dữ liệu thanh ghi A được đưa đến thanh ghi Temp1.



Hình 2-6. Dữ liệu thanh ghi D được đưa đến thanh ghi Temp2.



Hình 2-7. Kết quả lưu trữ lại thanh ghi A.



Hình 2-8. Các thanh ghi tạm trở lại trạng thái ban đầu.

### III. LỆNH CỦA VI XỬ LÝ:

#### 1 TẬP LỆNH CỦA VI XỬ LÝ:

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

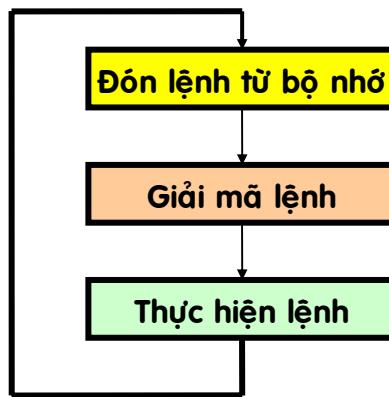
Khi nói đến tập lệnh của vi xử lý tức nói đến tất cả các lệnh mà vi xử lý có thể hiểu và thực hiện được.

Nếu tập lệnh của một vi xử lý giống với tập lệnh của một vi xử lý khác thì cấu trúc của 2 vi xử lý giống nhau.

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

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

Hình 2-9 minh họa chu kỳ thực hiện lệnh:

**Hình 2-9. Chu kỳ thực hiện lệnh của vi xử lý.**

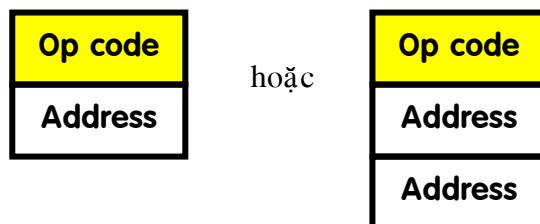
Một lệnh được thực hiện cần phải hội đủ 2 yếu tố:

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

**Yếu tố thứ hai** là lệnh phải cho vi xử lý biết các thông tin địa chỉ tức là vị trí của các dữ liệu mà vi xử lý phải thực hiện. Ví dụ khi thực hiện một lệnh cộng nội dung 2 thanh ghi A và B, hoặc nội dung thanh ghi A và dữ liệu chứa trong một ô nhớ. Yếu tố thứ 2 trong trường hợp này là các thanh ghi A và B, hoặc thanh ghi A và địa chỉ của ô nhớ.

Yếu tố thứ nhất gọi là mã lệnh : op code (operation code) và yếu tố thứ 2 gọi là địa chỉ. Mã lệnh sẽ báo cho vi xử lý làm gì và địa chỉ sẽ cho vi xử lý biết vị trí của dữ liệu.

Sơ đồ hình 2-10 minh họa cho cấu trúc 1 lệnh.

**Hình 2-10. Cấu trúc của một lệnh bao gồm mã lệnh và địa chỉ.**

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

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

## 2. TỪ GỐI NHỚ (MNEMONICS):

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

Để giải quyết vấn đề này lệnh được viết thành các từ gõi nhớ rất gần với chức năng và ý nghĩa của các lệnh. Trong hầu hết các từ gõi nhớ của lệnh, mã lệnh được rút gọn chỉ còn khoảng 3 ký tự. Ví dụ lệnh di chuyển dữ liệu có từ gõi nhớ là MOV, lệnh trừ là SUB, ... Khi lệnh có các địa chỉ đi sau thì các địa chỉ này vẫn là các con số. Ví dụ lệnh nhảy đến một ô nhớ viết như sau:

**JMP FA90<sub>H</sub>.**

Khi sử dụng các từ gõi nhớ này giúp người lập trình rất dễ nhớ tất cả các lệnh của vi xử lý, khi viết chương trình người lập trình dùng các từ gõi nhớ để viết chương trình, các từ gõi nhớ này tạo thành một ngôn ngữ gọi là Assembly – khi viết chương trình bằng ngôn ngữ Assembly thì vi xử lý sẽ không hiểu – muốn vi xử lý hiểu và thực hiện chương trình thì phải sử dụng chương trình biên dịch Assembler để chuyển các lệnh viết dưới dạng ngôn ngữ Assembly thành các lệnh dạng số nhị phân và các địa chỉ dạng nhị phân tương ứng rồi nạp vào bộ nhớ thì vi xử lý mới thực hiện được.

### 3. CÁC NHÓM LỆNH CƠ BẢN CỦA VI XỬ LÝ:

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

- ◆ Nhóm lệnh truyền dữ liệu: Data transfers.
- ◆ Nhóm lệnh trao đổi, truyền khối dữ liệu, lệnh tìm kiếm: Exchanges, Block transfers, Searches.
- ◆ Nhóm lệnh số học và logic: arithmetic and logic.
- ◆ Nhóm lệnh xoay và dịch: Rotates and shifts.
- ◆ Nhóm lệnh điều khiển CPU.
- ◆ Nhóm lệnh về bit: Bit set, bit reset, and bit test.
- ◆ Nhóm lệnh nhảy: Jumps.
- ◆ Nhóm lệnh gọi, trả về và nhóm lệnh bắt đầu: Calls, Return, and Restarts.
- ◆ Nhóm lệnh xuất nhập: Input and Output.

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

### 4. CÁC KIỂU TRUY XUẤT ĐỊA CHỈ CỦA MỘT VI XỬ LÝ:

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

**Chú Ý:** Danh từ truy xuất bộ nhớ có nghĩa là tạo ra địa chỉ để truy xuất dữ liệu, vi xử lý truy xuất dữ liệu có thể là lấy dữ liệu từ ô nhớ hoặc lưu trữ dữ liệu vào ô nhớ. Có thể gọi là các kiểu địa chỉ hóa bộ nhớ hay các kiểu tạo địa chỉ để truy xuất bộ nhớ.

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

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

- ◆ Kiểu định địa chỉ ngầm định (Implied Addressing Mode).

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

#### a. Kiểu định địa chỉ ngầm định:

Để hiểu các kiểu truy xuất phải dùng tập lệnh của một vi xử lý 8 bit.

Ví dụ lệnh cộng: ADD reg

Lệnh này được hiểu là nội dung của thanh ghi A được cộng với nội dung của thanh ghi Reg kết quả lưu trữ vào thanh ghi A.

#### b. Kiểu định địa chỉ tức thời:

Một lệnh được chia ra làm 2 phần thứ nhất là mã lệnh hay còn gọi là mã công tác, phần thứ 2 là địa chỉ. Đối với kiểu địa chỉ tức thời thì phần thứ 2 là dữ liệu không phải là địa chỉ.

Ví dụ lệnh nạp một dữ liệu tức thời vào thanh ghi A được viết như sau: MVI A, FE<sub>H</sub>. Trong đó MVI là mã gợi nhớ, FE là dữ liệu dạng số Hex. Vì thanh ghi A chỉ có 8 bit nên dữ liệu tức thời có độ dài là 8 bit.

#### c. Kiểu định địa chỉ trực tiếp:

Ví dụ lệnh di chuyển nội dung của một ô nhớ có địa chỉ 8000<sub>H</sub> vào thanh ghi A: “LDA 8000<sub>H</sub>”. LDA là mã gợi nhớ, địa chỉ 8000<sub>H</sub> được viết trực tiếp trong câu lệnh, với vi xử lý 8 bit có 16 đường địa chỉ nên phải dùng 4 số Hex để chỉ định một ô nhớ.

Đối với những lệnh dùng kiểu địa chỉ trực tiếp thì lệnh có độ dài là 3 byte: một byte là mã lệnh, 2 byte còn lại là địa chỉ của ô nhớ (đối với vi xử lý 8 bit).

#### d. Kiểu định địa chỉ gián tiếp dùng thanh ghi:

Để minh họa kiểu địa chỉ gián tiếp dùng thanh ghi ta dùng lệnh sau:

Ví dụ: “MOV A,M”. Lệnh này sẽ di chuyển nội dung của ô nhớ M có địa chỉ chứa trong một cặp thanh ghi. Đối với vi xử lý 8085 thì địa chỉ này thường chứa trong cặp thanh ghi HL, vì địa chỉ 16 bit nên phải dùng cặp thanh ghi mới chứa hết 16 bit địa chỉ.

**Chú ý khi dùng lệnh kiểu này người lập trình phải quản lý giá trị trong cặp thanh ghi.**

#### e. Kiểu định địa chỉ chỉ số:

Đối với một vài vi xử lý có các thanh ghi chỉ số (Index register) được dùng cho kiểu địa chỉ chỉ số.

Kiểu địa chỉ này được thực hiện bằng cách cộng byte thứ 2 của lệnh với nội dung của thanh ghi chỉ số ID. Ví dụ: lệnh cộng nội dung thanh ghi A với nội dung của ô nhớ có địa chỉ chứa trong thanh ghi chỉ số ID với byte dữ liệu thứ 2: “ADD A, (ID +n)” n là một số có chiều dài 8 bit.

### f. Kiểu định địa chỉ tương đối:

Kiểu địa chỉ này gần giống như kiểu địa chỉ số nhưng thanh ghi ID được thay thế bằng thanh ghi PC. Địa chỉ của ô nhớ cần truy xuất được tính bằng cách cộng nội dung hiện tại chứa trong thanh ghi PC cộng với byte dữ liệu thứ 2.

Ví dụ lệnh JP 05<sub>H</sub> : nhảy đến tới thực hiện lệnh có địa chỉ cách bộ đếm chương trình PC là 5 byte.

## IV. KHẢO SÁT VI XỬ LÝ 8 BIT Z80:

Sau khi khảo sát một vi xử lý tổng quát bây giờ chúng ta cần khảo sát một vài vi xử lý 8 bit cụ thể được sử dụng phổ biến trong các hệ thống điều khiển.

Mặc dù hiện tại các vi xử lý ngày càng mạnh về số bit, tốc độ làm việc cao, khả năng truy xuất bộ nhớ lớn nhưng các vi xử lý 8 bit vẫn tồn tại vì chúng được sử dụng trong các hệ thống điều khiển nhỏ với lượng dữ liệu xử lý không nhiều ví dụ như máy giặt tự động thì vi xử lý chỉ thực hiện các công việc như điều khiển động cơ quay thuận nghịch theo thời gian, hiển thị thời gian trên led 7 đoạn, kiểm tra các valve để đóng mở nước, kiểm tra mực nước,... lượng dữ liệu xử lý không nhiều nếu so với lượng dữ liệu mà các vi xử lý của máy tính xử lý.

Trong phần này chúng ta khảo sát 2 vi xử lý 8 bit: vi xử lý 8085 của hãng Intel và vi xử lý Z80 của hãng Zilog.

Vi xử lý Z80 của hãng Zilog được phát triển từ vi xử lý gốc 8080 của hãng Intel.

### 1 SƠ ĐỒ CẤU TRÚC BÊN TRONG CỦA VI XỬ LÝ Z80:

Cấu trúc của vi xử lý Z80 như hình 2-11 :

Trong sơ đồ cấu trúc của vi xử lý Z80 có:

- Khối ALU cùng với 2 thanh ghi có tên là Latch có chức năng giống như đã khảo sát.
- Khối các thanh ghi gồm có: thanh ghi A, B, C, D, E, H, L, F, A', B', C', D', E', H', L', F', IP, PC, SP, IX và IY.
- Bus dữ liệu 2 chiều D0 ÷ D7 dùng giao tiếp dữ liệu với các thiết bị bên ngoài – khối đệm bus dữ liệu có chức năng đệm để cho phép kết nối nhiều thiết bị hay tăng fan\_out.
- Bus dữ liệu bên trong cho phép liên lạc dữ liệu giữa các khối.
- Bus địa chỉ một chiều gồm có 16 đường có chức năng tải địa chỉ để truy xuất bộ nhớ và thiết bị ngoại vi, khối đệm bus địa chỉ dùng để tăng fan-out.
- Khối giải mã lệnh dùng để giải mã lệnh cho khối điều khiển thực hiện lệnh.
- Khối điều khiển ngắt.

Các khối còn bao gồm: khối điều khiển bus, khối điều khiển đọc ghi bộ nhớ, đọc ghi IO, khối điều khiển vi xử lý, khối reset và làm tươi.

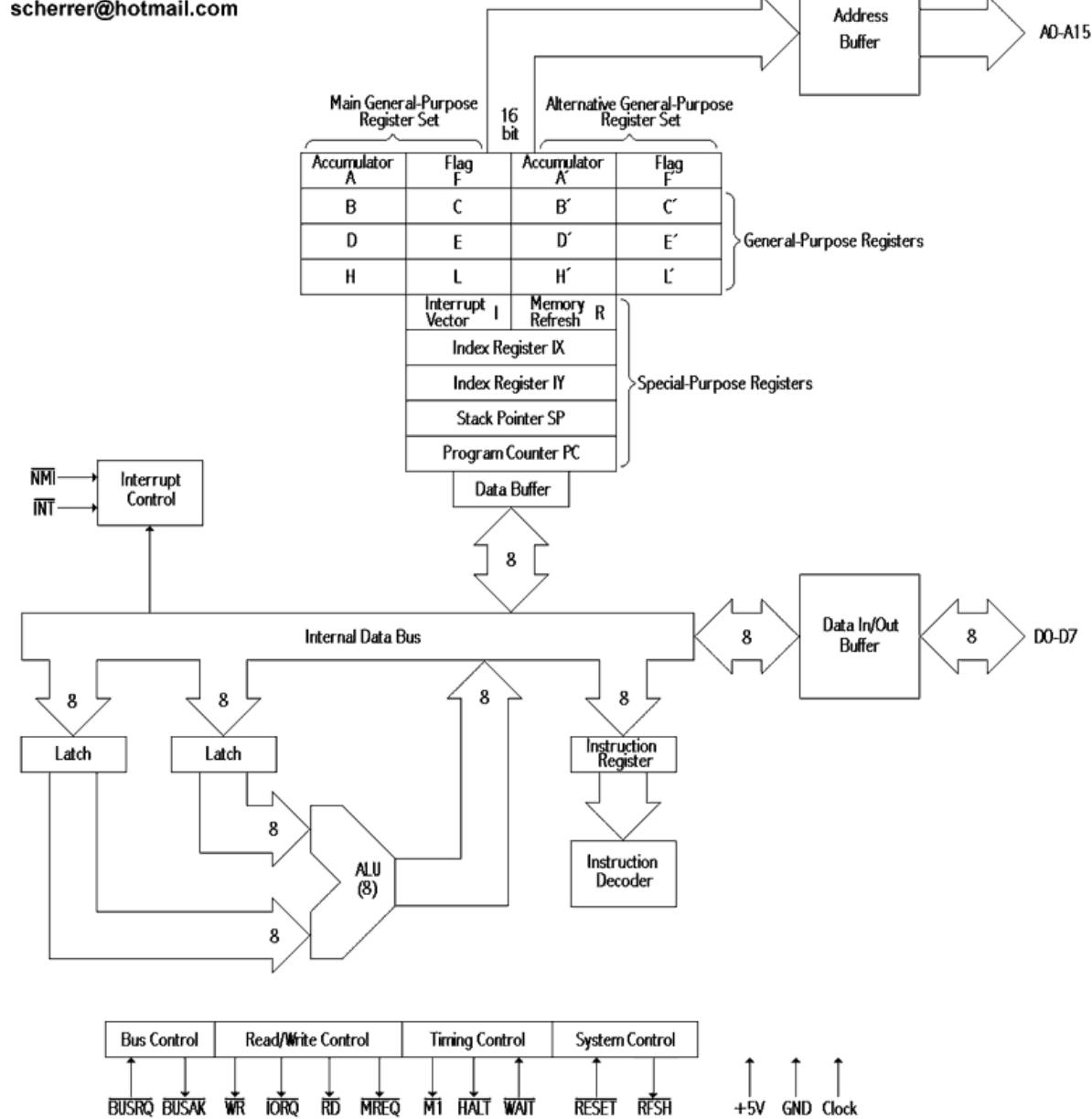
Nguồn sử dụng cho vi xử lý là +5V, xung clock được cung cấp từ bên ngoài.

Drawing by: Thomas Scherrer

<http://www.geocities.com/SiliconValley/Peaks/3938/>

scherrer@hotmail.com

## Z80 CPU Block Diagram

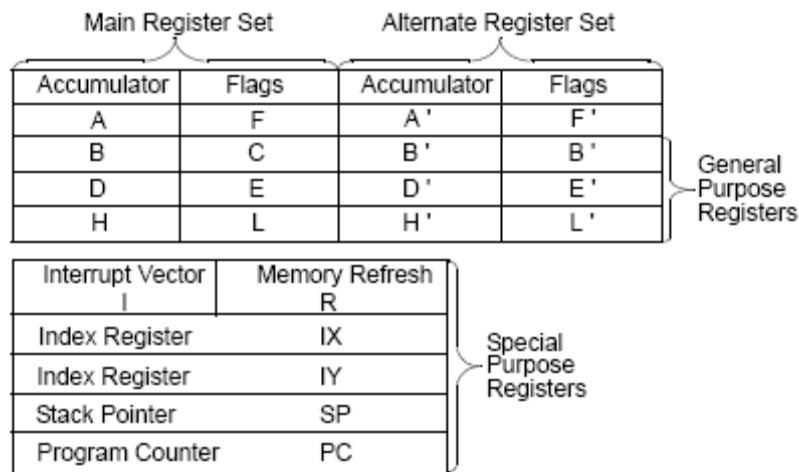


Hình 2-11. Sơ đồ cấu trúc bên trong của vi xử lý Z80.

### 2. TỔ CHỨC THANH GHI BÊN TRONG CỦA VI XỬ LÝ Z80:

Vi xử lý Z80 có 16 thanh ghi 8 bit và 4 thanh ghi 16 bit được trình bày như hình 2-12. Tất cả các thanh ghi của vi xử lý Z80 được sử dụng giống như các ô nhớ RAM tĩnh. Các thanh ghi bao gồm 2 nhóm thanh ghi thông dụng có thể sử dụng một cách độc lập như những thanh ghi 8 bit hoặc có thể kết hợp lại tạo thành những cặp thanh ghi 16 bit. Mỗi nhóm thanh ghi gồm 6 thanh ghi thông dụng, một thanh ghi tích lũy A và một thanh ghi trạng thái.

**Thanh ghi PC:** thanh ghi này giống như thanh ghi PC đã trình bày. Thanh ghi này chứa địa chỉ của lệnh hiện hành đang được đón từ bộ nhớ. Nội dung của thanh ghi PC được tăng lên để quản lý lệnh tiếp theo sau khi nội dung của nó được chuyển sang bus địa chỉ. Khi thực hiện lệnh Jump thì địa chỉ của lệnh nơi nhảy đến sẽ được đặt vào thanh ghi PC và sẽ ghi đè lên giá trị đã tăng.



Hình 2-12. Sơ đồ chẩn của vi xử lý Z80.

**Thanh ghi SP:** thanh ghi này giống như thanh ghi SP đã trình bày. Thanh ghi SP chứa địa chỉ hiện hành của bộ nhớ ngắn xếp nằm trong vùng bộ nhớ RAM. Vùng nhớ RAM bên ngoài được tổ chức theo kiểu vào sau ra trước (LIFO). Dữ liệu của các thanh ghi được cất vào ngắn xếp hoặc được lấy ra khỏi ngắn xếp bằng cách thực hiện các lệnh PUSH và POP. Bộ nhớ ngắn xếp còn được dùng để lưu trữ địa chỉ của thanh ghi PC khi thực hiện lệnh gọi chương trình con hay khi CPU bị ngắt.

**Thanh ghi chỉ số IX và IY:** 2 thanh ghi chỉ số lưu địa chỉ 16 bit được dùng cho kiểu định địa chỉ dùng chỉ số. Trong kiểu định địa chỉ này thì một thanh ghi sẽ quản lý địa chỉ của vùng dữ liệu được lưu trữ hoặc quản lý địa chỉ của vùng dữ liệu cần di chuyển.

Ngoài ra 2 thanh ghi này còn được sử dụng trong kiểu định địa chỉ dùng chỉ số có độ dời sẽ tiện ích hơn khi bảng dữ liệu được sử dụng.

**Thanh ghi lưu địa chỉ ngắn (I):** thường thì các ngắt của vi xử lý có 1 địa chỉ ngắn cố định ví dụ như ngắt không ngắn được NMI thì khi vi xử lý đáp ứng ngắt này nó sẽ thực hiện chương trình phục vụ ngắt tại địa chỉ 0066H do nhà thiết kế qui định. Với kiểu như vậy thì chỉ có một chương trình con phục vụ ngắt và phải bắt đầu tại địa chỉ qui định. Để đáp ứng được nhiều chương trình con phục vụ ngắt khác nhau thì nhà thiết kế vi xử lý Z80 sử dụng ngắt INT và việc xử lý địa chỉ như sau: địa chỉ của chương trình con phục vụ ngắt là 16 bit, trong đó 8 bit cao được lưu trong thanh ghi I, còn 8 bit địa chỉ thấp thì do thiết bị yêu cầu ngắt cung cấp.

Cấu trúc này cho phép các chương trình con phục vụ ngắt có thể định vị tại bất kỳ vị trí nào trong vùng nhớ với thời gian nhảy đến thực hiện chương trình con phục vụ ngắt là tối thiểu.

**Thanh ghi làm tươi bộ nhớ R (refresh register):** vi xử lý Z80 có một bộ đếm làm tươi bộ nhớ để cho phép sử dụng bộ nhớ động DRAM giống như SRAM. 7 bit của thanh ghi 8 bit này chính là bộ đếm sẽ tự động tăng giá trị sau mỗi chu kỳ đón lệnh từ bộ nhớ. Bit thứ 8 còn lại có thể lập trình được từ lệnh “LD R, A”. Dữ liệu đếm trong bộ đếm 7 bit được gởi đến bus địa chỉ thấp cùng với tín hiệu làm tươi bộ nhớ khi CPU đang giải mã lệnh và thực hiện lệnh đã đón về. Với kiểu làm tươi bộ nhớ như thế sẽ không làm chậm tốc độ làm việc của CPU.

Trong khoảng thời gian làm tươi bộ nhớ thì nội dung của thanh ghi I được đặt lên 8 bit cao của bus địa chỉ.

**Thanh ghi Accumulator và thanh ghi Flag:** vi xử lý Z80 có 2 thanh ghi A và 2 thanh ghi cỡ 8 bit. Hoạt động của các thanh ghi này giống như đã trình bày ở phần trước.

**Các thanh ghi thông dụng:** vi xử lý Z80 có 2 nhóm thanh ghi thông dụng – mỗi nhóm có 8 thanh ghi 8 bit có thể sử dụng như những thanh ghi 8 bit độc lập hay cũng có thể sử dụng như những cặp thanh ghi 16 bit. Một nhóm được gọi là BC, DE và HL trong khi nhóm còn lại là BC', DE' và HL'.

Tại mỗi một thời điểm, người lập trình có thể lựa chọn 1 trong 2 nhóm thanh ghi để làm việc thông qua lệnh. Trong những hệ thống đòi hỏi đáp ứng ngắt nhanh thì nhóm các thanh ghi thông dụng cùng với thanh ghi A, thanh ghi cỡ có thể được để dành sử dụng cho chương trình con phục vụ ngắt yêu cầu đáp ứng nhanh – chỉ cần thực hiện 1 lệnh trao đổi để chuyển sang chương trình con. Chức năng này làm thời gian đáp ứng yêu cầu ngắt nhanh hơn vì nếu không có chức năng này thì phải tiến hành cất dữ liệu của các thanh ghi vào bộ nhớ ngắn xép làm tổn nhiều thời gian trước khi thực hiện chương trình con phục vụ ngắt.

**Khối ALU:** khối ALU bên trong vi xử lý Z80 có chức năng thực hiện các lệnh số học và lệnh logic. Khối ALU kết nối với các thanh ghi bên trong và bus dữ liệu bên ngoài thông qua bus dữ liệu bên trong. Các lệnh được thực hiện bởi khối ALU bao gồm:

Lệnh cộng, lệnh trừ, lệnh AND, lệnh OR, lệnh EX-OR, lệnh so sánh, lệnh dịch trái, dịch phải, lệnh xoay, lệnh tăng, lệnh giảm, lệnh set bit, lệnh reset bit lệnh test bit.

**Thanh ghi lệnh IR và khối điều khiển CPU:** mỗi mã lệnh được đón về từ bộ nhớ sẽ lưu trong thanh ghi lệnh IR và sẽ được giải mã. Các thành phần của khối điều khiển sẽ thực hiện chức năng này, sau đó khối điều khiển tạo ra và cung cấp các tín hiệu điều khiển cần thiết để đọc dữ liệu từ các thanh ghi hoặc ghi dữ liệu lên các thanh ghi, điều khiển khối ALU thực hiện phép toán và tạo ra các tín hiệu điều khiển các đối tượng bên ngoài.

### 3. CHỨC NĂNG CÁC CHÂN CỦA VI XỬ LÝ Z80:

Vi xử lý Z80 có sơ đồ chân như hình 2-13:

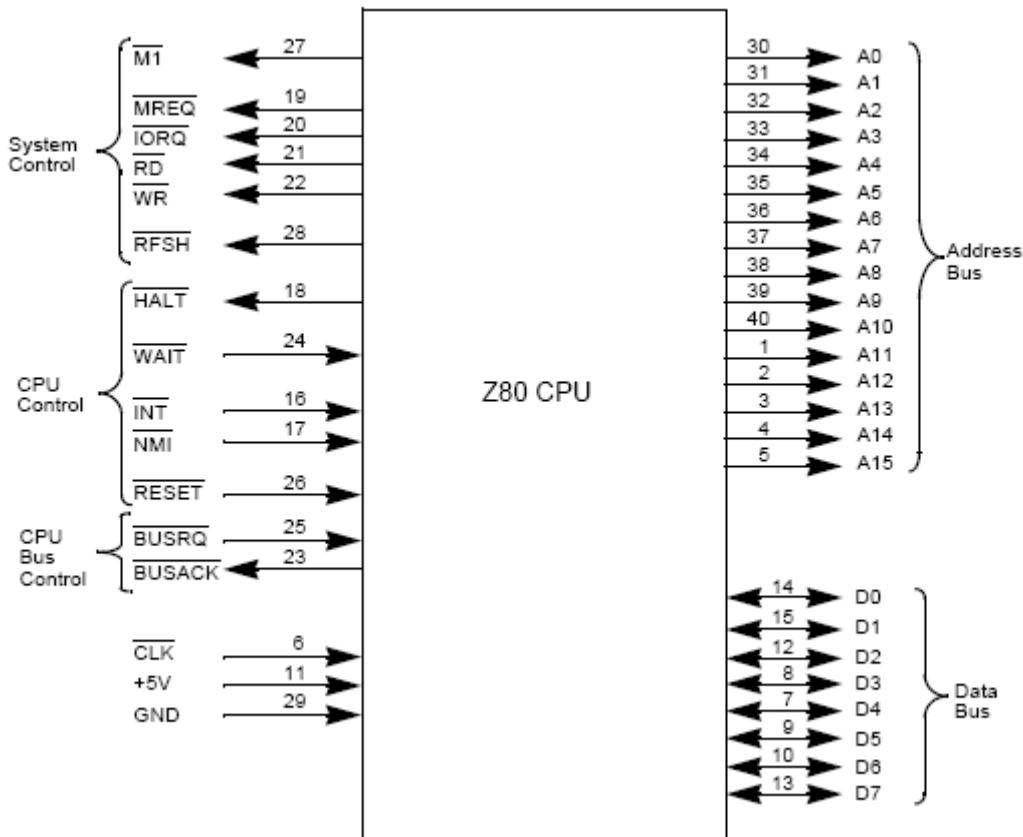
Vi xử lý Z80 có 40 chân, trong đó:

- Bus địa chỉ có 16 chân từ A0 ÷ A15.
- Bus dữ liệu có 8 chân từ D0 ÷ D7.
- Bus điều khiển hệ thống có 6 chân:  $\overline{M1}$ ,  $\overline{MREQ}$ ,  $\overline{IORQ}$ ,  $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{RFSH}$
- Điều khiển CPU có 5 chân:  $\overline{HALT}$ ,  $\overline{WAIT}$ ,  $\overline{INT}$ ,  $\overline{NMI}$ ,  $\overline{RESET}$
- Điều khiển bus có 2 chân:  $\overline{BUSRQ}$ ,  $\overline{BUSACK}$
- Nguồn cung cấp 2 chân: Vcc và GND.
- Nguồn nhận xung clock 1 chân CLK.

Giống như vi xử lý tổng quát thì chức năng của bus địa chỉ dùng để tải địa chỉ khi vi xử lý truy xuất bộ nhớ hoặc ngoại vi IO. Chức năng của bus dữ liệu là tải dữ liệu giữa vi xử lý với bộ nhớ hoặc thiết bị ngoại vi. Hai bus địa chỉ và dữ liệu dùng để kết nối với bộ nhớ và ngoại vi IO.

Tín hiệu  $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{MREQ}$ ,  $\overline{IORQ}$ : là các chân xuất tín hiệu, khi vi xử lý điều khiển các chân  $\overline{RD}$ ,  $\overline{WR}$  cùng với tín hiệu  $\overline{MREQ}$  để thực hiện quá trình đọc hoặc ghi dữ liệu giữa vi xử

lý với bộ nhớ. Khi vi xử lý điều khiển các chân  $\overline{RD}$ ,  $\overline{WR}$  cùng với tín hiệu  $\overline{IORQ}$  để thực hiện quá trình đọc hoặc ghi dữ liệu giữa vi xử lý với thiết bị ngoại vi IO. Trong hệ thống ta phải sử dụng các chân này để giao tiếp với bộ nhớ và ngoại vi IO.



Hình 2-13. Sơ đồ chân của vi xử lý Z80.

**Tín hiệu  $\overline{RFSH}$ :** dùng để điều khiển làm tươi bộ nhớ DRAM. Khi không cần làm tươi thì chân này để trống.

**Tín hiệu  $\overline{M1}$ :** là tín hiệu báo hiệu một chu kỳ máy. Tín hiệu  $\overline{M1}$  kết hợp với tín hiệu  $\overline{MREQ}$  để xác định chu kỳ máy hiện tại là chu kỳ đón lệnh trong quá trình thực hiện lệnh. Tín hiệu  $\overline{M1}$  kết hợp với tín hiệu  $\overline{IORQ}$  để xác định chu kỳ trả lời ngắn.

**Tín hiệu  $\overline{INT}$ :** là tín hiệu yêu cầu ngắn từ bên ngoài hay từ thiết bị ngoại vi tích cực mức thấp. CPU sẽ thực hiện yêu cầu ngắn sau khi thực hiện xong lệnh đang thực hiện và trừ khi ngắn này được cho phép bởi phần mềm. Khi sử dụng thì ngõ vào này thường nối với điện trở kéo lên để khi tác động thì xuống mức thấp.

**Tín hiệu  $\overline{NMI}$ :** là tín hiệu yêu cầu ngắn từ bên ngoài hay từ thiết bị ngoại vi tích cực cạnh xuống. Ngắt  $\overline{NMI}$  có mức ưu tiên cao hơn so với ngắt  $\overline{INT}$  và là tín hiệu ngắn không thể ngăn được. CPU sẽ thực hiện yêu cầu ngắn sau khi thực hiện xong lệnh đang thực hiện, địa chỉ của chương trình con phục vụ ngắn tại địa chỉ 0066H.

**Tín hiệu  $\overline{RESET}$ :** là tín hiệu ngõ vào tích cực mức thấp, khi ngõ vào reset bị tác động thì CPU sẽ reset tất cả các bit cho phép ngắn, xoá thanh ghi PC, các thanh ghi I và R thiết lập trạng thái ngắn ở mode 0. Trong khoảng thời gian ngắn thì các bus địa chỉ và bus dữ liệu ở trạng thái

tổng trễ cao, tất cả các đường tín hiệu điều khiển đều ở trạng thái không tích cực. Chú ý tín hiệu reset phải ở trạng thái tích cực trong khoảng thời gian ít nhất là 3 chu kỳ xung clock.

**Tín hiệu  $\overline{WAIT}$** : là tín hiệu ngõ vào tích cực mức thấp, tín hiệu này (khi ở mức cao) dùng để báo cho CPU biết ô nhớ đang truy xuất hay thiết bị ngoại vi đang truy xuất chưa sẵn sàng cho việc nhận hay gửi dữ liệu. CPU sẽ vào trạng thái chờ cho đến khi tín hiệu  $\overline{WAIT}$  trở lại trạng thái tích cực mức thấp.

**Tín hiệu  $\overline{HALT}$** : là tín hiệu ngõ ra tích cực mức thấp, tín hiệu này (khi ở mức cao) dùng để báo cho thiết bị bên ngoài biết: CPU đang thực hiện lệnh HALT và đang đợi một trong 2 yêu cầu ngắn xảy ra trước khi khôi phục lại hoạt động bình thường. Trong khoảng thời gian HALT thì CPU thực hiện lệnh NOP để tiếp tục giữ hoạt động làm tươi bộ nhớ.

**Tín hiệu  $\overline{BUSRQ}$** : là tín hiệu ngõ vào tích cực mức thấp, tín hiệu này có mức ưu tiên cao hơn ngắn không che được  $\overline{NMI}$  và nó sẽ được thực hiện ngay sau mỗi chu kỳ máy hay chu kỳ xung clock. Tín hiệu này sẽ yêu cầu vi xử lý điều khiển bus địa chỉ, bus dữ liệu và các đường tín hiệu điều khiển  $\overline{MREQ}$ ,  $\overline{IORQ}$ ,  $\overline{RD}$ ,  $\overline{WR}$  ở trạng thái tổng trễ cao và thiết bị khác có thể sử dụng các bus trong hệ thống. Ta có thể xem tín hiệu  $\overline{BUSACK}$  là tín hiệu trả lời của CPU khi CPU nhận tín hiệu yêu cầu những bus  $\overline{BUSRQ}$ .

#### 4. GẦN ĐỒ THỜI GIAN CỦA VI XỬ LÝ Z80:

Vi xử lý Z80 thực hiện các lệnh theo từng bước thông qua các hoạt động cơ bản bao gồm:

- Đón lệnh.
- Đọc hoặc ghi bộ nhớ.
- Đọc hoặc ghi IO.
- Yêu cầu bus/trả lời bus.
- Yêu cầu ngắn/trả lời ngắn.
- Đáp ứng ngắn không ngăn được.

Tất cả các lệnh là một chuỗi hoạt động cơ bản nối tiếp nhau. Mỗi một hoạt động có thể thực hiện từ 3 chu kỳ xung clock đến 6 chu kỳ xung clock để hoàn tất hoặc có thể bị kéo dài hơn để đồng bộ tốc độ làm việc của CPU với tốc độ làm việc của các thiết bị bên ngoài.

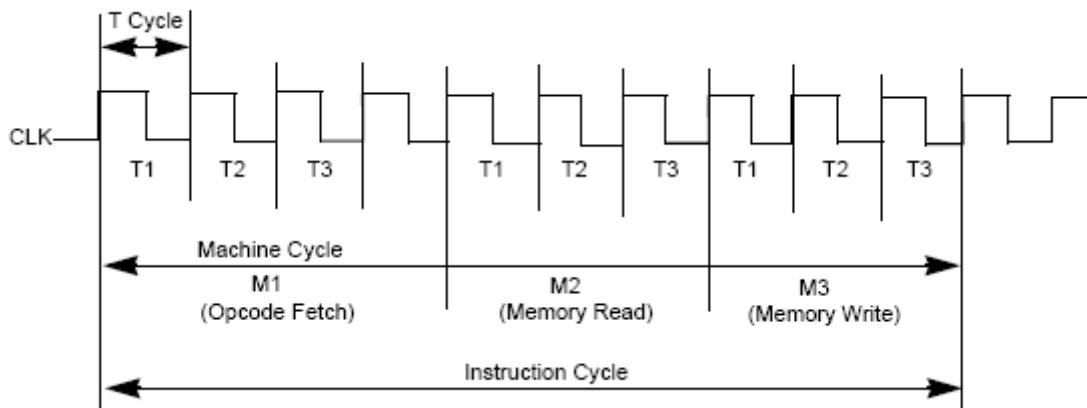
Các chu kỳ xung clock được xem là các chu kỳ T (time) và các hoạt động được xem là các chu kỳ máy M (machine). Hình 2-14 minh họa các chu kỳ xung clock T và chu kỳ máy M khi CPU thực hiện liên tục các lệnh:

Chú ý lệnh này bao gồm 3 chu kỳ máy M1, M2 và M3.

Ở chu kỳ máy thứ nhất của bất kỳ lệnh nào chính là chu kỳ đón lệnh từ bộ nhớ – chu kỳ này cần từ 4, 5 hoặc 6 chu kỳ xung clock T. Chu kỳ đón lệnh được sử dụng để đón mã lệnh của lệnh kế để thực hiện.

Tiếp theo là các chu kỳ di chuyển dữ liệu (đọc hoặc ghi) giữa CPU và bộ nhớ hoặc thiết bị ngoại vi IO xảy ra khoảng từ 3 đến 5 chu kỳ xung clock T.

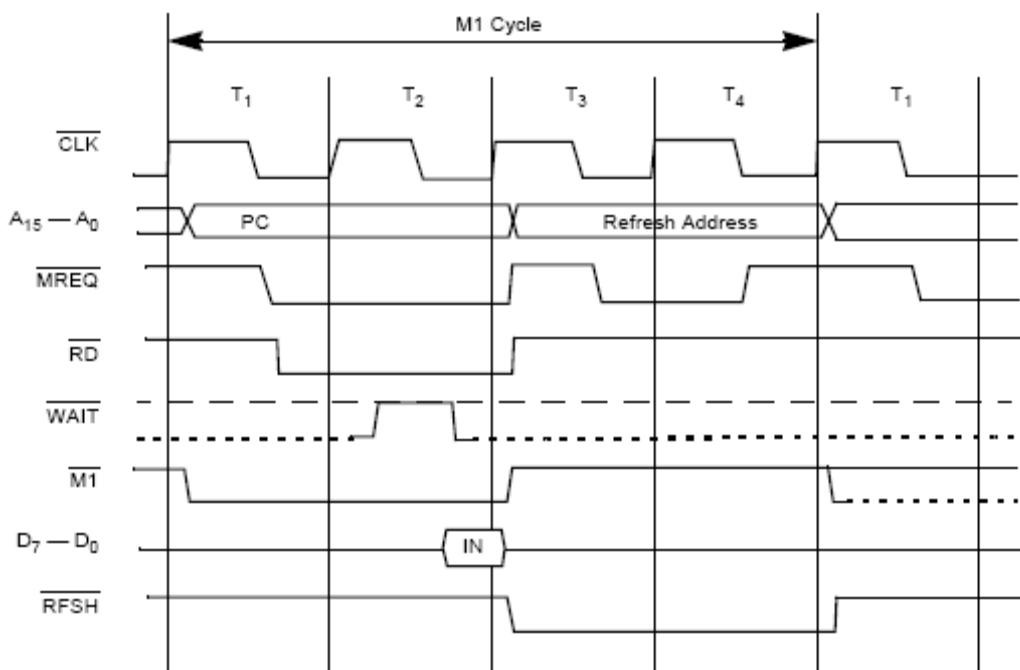
Trong khoảng thời gian T2 và mỗi chu kỳ đợi Tw sau thì CPU sẽ lấy mẫu tín hiệu ngõ vào WAIT khi có cạnh xuống của xung clock. Nếu tín hiệu WAIT ở trạng thái tích cực thì một chu kỳ đợi tiếp theo sẽ được thực hiện. Sử dụng kỹ thuật này để CPU có thể kéo dài thời gian đọc hoặc ghi để tương thích với bất kỳ loại bộ nhớ nào.



Hình 2-14. Các chu kỳ máy của vi xử lý Z80.

**Chu kỳ Đón lệnh:** hình 2-15 là giản đồ thời gian của chu kỳ đón lệnh từ bộ nhớ hay chu kỳ M1. Trong chu kỳ này CPU đặt địa chỉ lên bus địa chỉ tại thời điểm bắt đầu của chu kỳ M1. Sau nữa chu kỳ của xung clock: thì tín hiệu MREQ chuyển sang trạng thái tích cực. Tại thời điểm này địa chỉ trên bus đã có đủ thời gian để ổn định, do đó cạnh xuống của tín hiệu MREQ được dùng như xung clock cho phép chip đối với bộ nhớ động.

Tín hiệu RD cũng chuyển sang trạng thái tích cực để xác định rằng dữ liệu đọc từ bộ nhớ sẽ được phép vào bus dữ liệu của CPU.



Hình 2-15. Chu kỳ đón mã lệnh.

CPU sẽ lấy mẫu dữ liệu từ bộ nhớ trên bus dữ liệu khi có cạnh lên của xung clock ở chu kỳ T3 và đồng thời CPU sẽ chuyển các tín hiệu  $\overline{MREQ}$ ,  $\overline{RD}$  sang trạng thái không tích cực. Vậy CPU đã lấy dữ liệu trước khi chuyển tín hiệu  $\overline{RD}$  sang trạng thái không tích cực. Các chu kỳ T3 và T4 dùng để làm tươi các bộ nhớ RAM động. CPU sẽ dùng các khoảng thời gian này để giải mã lệnh và thực hiện lệnh đã được đón nên sẽ không có hoạt động nào được thực hiện trong khoảng thời gian của T3 và T4.

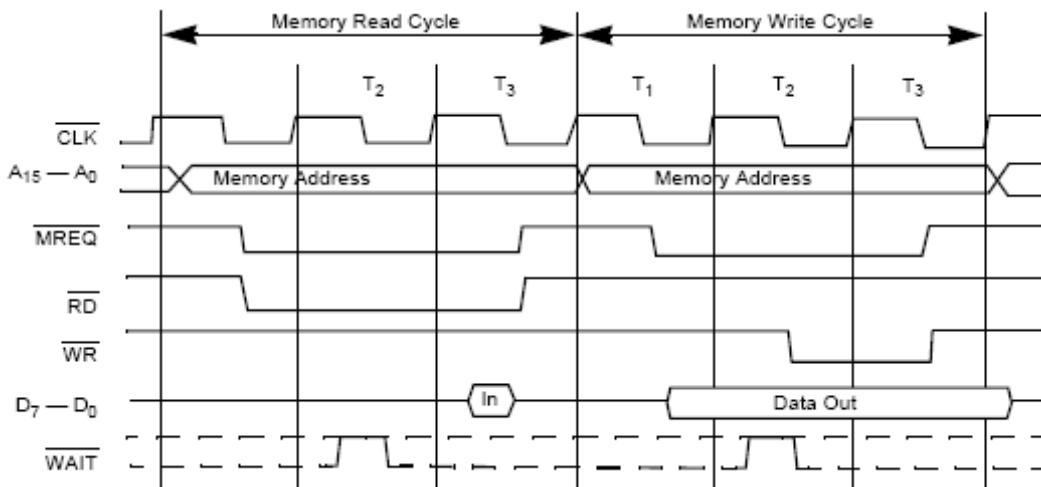
Chú ý dữ liệu đọc ở chu kỳ này là mã lệnh.

**Chu kỳ đọc hoặc ghi dữ liệu :** hình 2-16 là giản đồ thời gian của chu kỳ đọc hoặc ghi dữ liệu giữa CPU với bộ nhớ khác với chu kỳ đón lệnh. Tín hiệu  $\overline{MREQ}$  và tín hiệu  $\overline{RD}$  được điều khiển giống nhau trong chu kỳ đọc dữ liệu.

Trong chu kỳ ghi dữ liệu lên bộ nhớ, tín hiệu  $\overline{MREQ}$  cũng ở trạng thái tích cực khi địa chỉ trên bus đã ổn định nên nó được dùng như là tín hiệu cho phép đổi với bộ nhớ động.

Tín hiệu  $\overline{WR}$  sẽ chuyển sang trạng thái tích cực khi dữ liệu trên bus đã ổn định nên nó được dùng để điều khiển chân  $\overline{WR}$  của bộ nhớ để yêu cầu bộ nhớ cất dữ liệu trên bus dữ liệu.

Tín hiệu  $\overline{WR}$  sẽ chuyển sang trạng thái không tích cực trước khi địa chỉ và dữ liệu trên bus thay đổi hay trước khi chuyển sang chu kỳ máy kế tiếp.

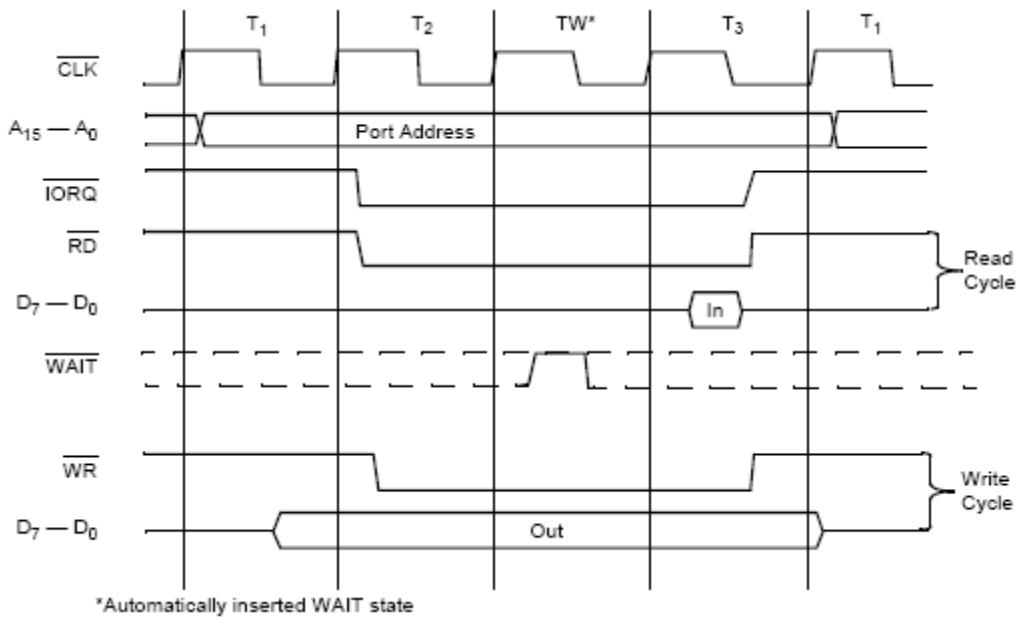


Hình 2-16. Chu kỳ đọc hoặc ghi bộ nhớ.

**Chu kỳ đọc hoặc ghi ngoại vi :** hình 2-17 là giản đồ thời gian của chu kỳ đọc hoặc ghi dữ liệu giữa CPU và thiết bị ngoại vi. Trong khoảng thời gian đọc/ghi IO thì CPU thường tự động xen vào 1 chu kỳ đợi. Lý do chèn thêm 1 chu kỳ đợi là do khoảng thời gian CPU điều khiển chân  $\overline{IORQ}$  sang trạng thái tích cực cho đến khi lấy mẫu rất ngắn. Nếu không chèn thêm chu kỳ đợi thì các thiết bị IO loại MOS sẽ không bắt kịp tốc độ làm việc của CPU.

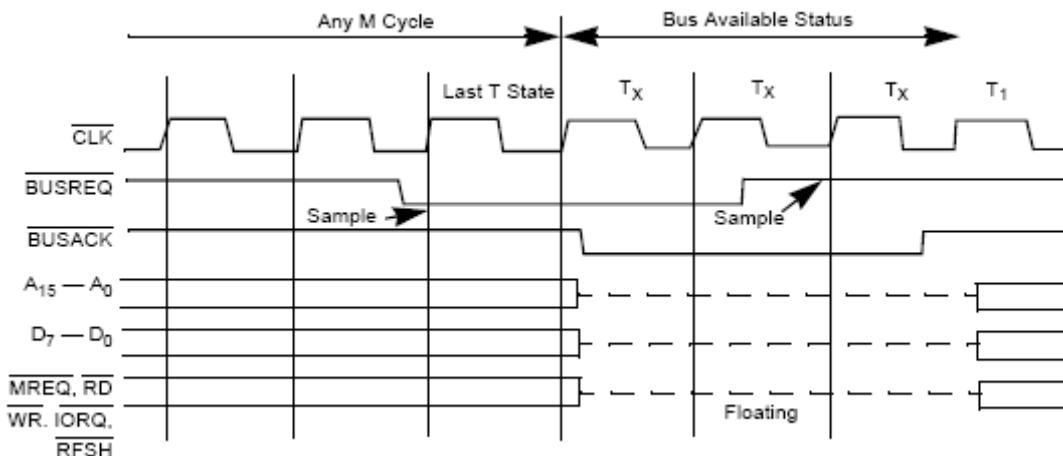
Trong khoảng thời gian đợi thì tín hiệu đợi  $\overline{WAIT}$  sẽ được lấy mẫu.

Trong hoạt động đọc dữ liệu từ IO thì tín hiệu  $\overline{RD}$  được dùng để yêu cầu thiết bị IO đã chỉ định xuất dữ liệu, trong hoạt động ghi dữ liệu lên IO thì tín hiệu  $\overline{WR}$  được dùng để yêu cầu thiết bị IO đã chỉ định nhận dữ liệu.



Hình 2-17. Chu kỳ đọc hoặc ghi thiết bị ngoại vi.

**Chu kỳ yêu cầu bus/trả lời yêu cầu bus :** hình 2-18 là giản đồ thời gian của chu kỳ yêu cầu bus / trả lời bus. Tín hiệu  $\overline{BUSRQ}$  được CPU lấy mẫu khi có cạnh lên của xung clock cuối cùng của mỗi chu kỳ máy. Nếu tín hiệu  $\overline{BUSRQ}$  ở trạng thái tích cực thì CPU thiết lập bus địa chỉ, bus dữ liệu và các tín hiệu điều khiển 3 trạng thái sang trạng thái tổng trở cao ngay khi có cạnh lên của xung clock kế. Tại thời điểm này bất kỳ thiết bị nào bên ngoài có thể điều khiển các bus để chuyển dữ liệu giữa bộ nhớ và các thiết bị ngoại vi IO – hoạt động này thường được gọi là truy xuất bộ nhớ trực tiếp DMA (direct memory access).

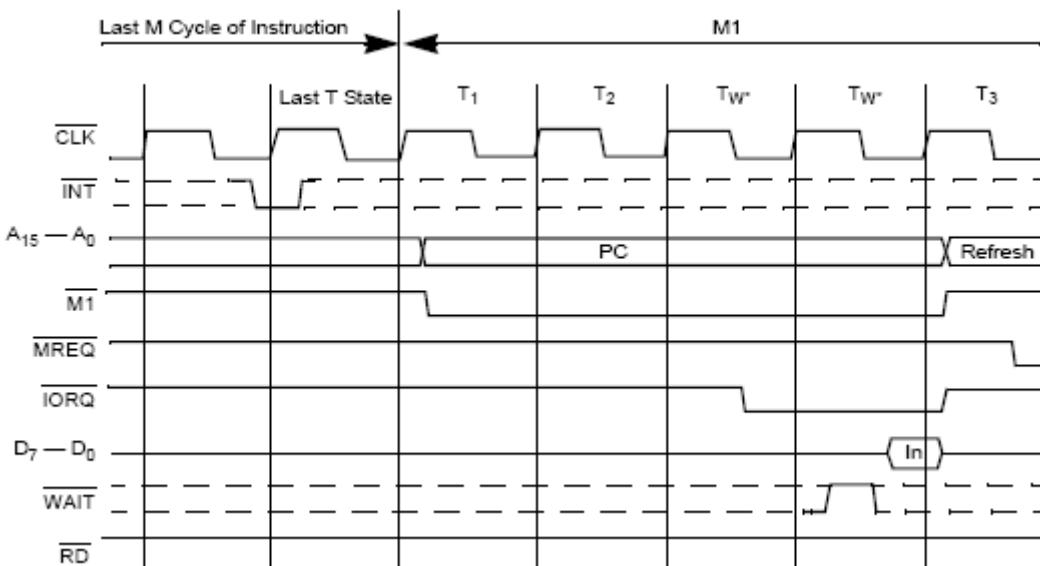


Hình 2-18. Chu kỳ yêu cầu bus/ trả lời bus.

Thời gian tối đa để CPU đáp ứng yêu cầu bus dài đúng bằng 1 chu kỳ máy và thiết bị yêu cầu bus phải giữ tín hiệu điều khiển  $\overline{BUSRQ}$  ở trạng thái tích cực trong khoảng đúng bằng 1 chu kỳ máy. Nếu thời gian yêu cầu bus quá dài và nếu hệ thống sử dụng bộ nhớ động thì thiết bị yêu cầu bus phải thực hiện công việc làm tươi bộ nhớ. Trường hợp này chỉ xảy ra khi khối lượng dữ liệu quá lớn cần phải di chuyển bởi DMA. Trong

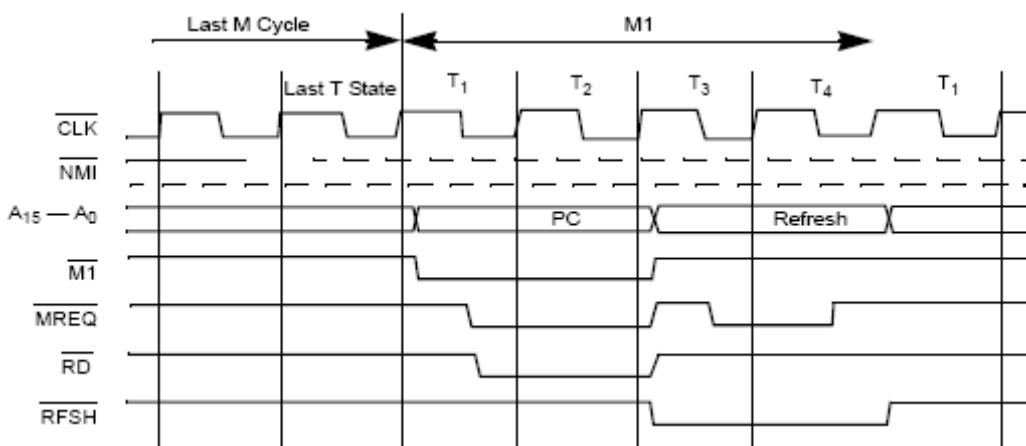
khoảng thời gian thực hiện yêu cầu nhường bus thì CPU không thể thực hiện bất kỳ yêu cầu ngắn nào cho dù là ngắn không che được.

**Chu kỳ yêu cầu ngắn/trả lời ngắn :** hình 2-19 là giản đồ thời gian thực hiện yêu cầu ngắn của CPU. CPU lấy mẫu tín hiệu  $\overline{INT}$  khi có cạnh lên của xung clock cuối cùng của mỗi chu kỳ đón mã lệnh. Tín hiệu ngắn  $\overline{INT}$  sẽ không được đáp ứng nếu như bị cấm bởi phần mềm hoặc tín hiệu yêu cầu nhường bus đang ở trạng thái tích cực. Trong khoảng thời gian của chu kỳ M1 kế tiếp, tín hiệu  $IORQ$  trở nên tích cực để xác định rằng thiết bị yêu cầu ngắn có thể đặt vector địa chỉ ngắn lên bus dữ liệu. Hai chu kỳ trạng thái đợi được tự động thêm vào trong chu kỳ đáp ứng yêu cầu ngắn. Hai trạng thái này được thêm vào để đủ thời gian cho các tín hiệu để ổn định và xác định thiết bị yêu cầu ngắn nào cần phải đưa vector địa chỉ ngắn.



Hình 2-19. Chu kỳ ngắn và đáp ứng yêu cầu ngắn.

**Đáp ứng ngắn không ngăn được :** hình 2-20 là giản đồ thời gian thực hiện yêu cầu ngắn không ngăn được  $\overline{NMI}$  của CPU. CPU lấy mẫu tín hiệu  $\overline{NMI}$  giống như ngắn  $\overline{INT}$  nhưng yêu cầu ngắn  $\overline{NMI}$  có mức độ ưu tiên cao nhất và không thể cấm bằng phần mềm. Ngắn này thường được sử dụng để CPU đáp ứng tức thời cho những yêu cầu quan trọng như sự cố hỏng của nguồn điện.



Hình 2-20. Gian do thoi gian cua ngan khong ngan deu  $\overline{NMI}$ .

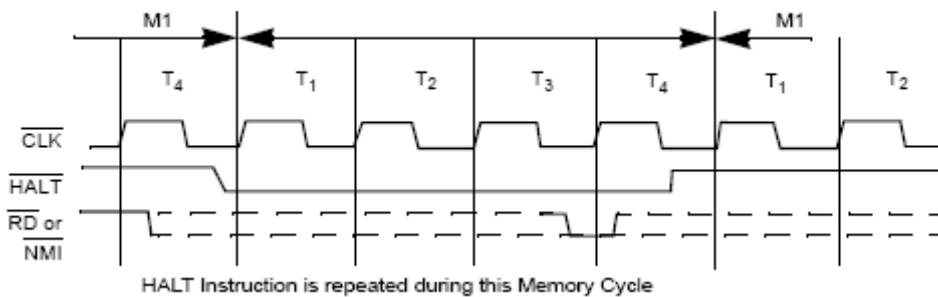
CPU đáp ứng yêu cầu ngắt không ngăn được  $\overline{NMI}$  tương tự như hoạt động đọc bộ nhớ bình thường. Sự khác biệt là nội dung của dữ liệu trên bus bị bỏ qua trong khi đó vi xử lý tự động lưu trữ nội dung của thanh ghi PC (là địa chỉ lệnh kế) vào bộ nhớ ngăn xếp bên ngoài và thực hiện chương trình tại địa chỉ 0066H. Chương trình con phục vụ ngắt phải được viết tại địa chỉ này.

**Lệnh Halt:** khi thực hiện lệnh Halt thì CPU thực hiện lệnh NOP cho đến khi CPU nhận được tín hiệu yêu cầu ngắt ( $\overline{INT}$  hoặc  $\overline{NMI}$ ). Hai tín hiệu ngắt này được lấy mẫu khi có cạnh lên của xung clock ở mỗi chu kỳ T4 như hình 2-21.

Khi có yêu cầu ngắt không ngăn được  $\overline{NMI}$  hoặc ngắt  $\overline{INT}$  được cho phép bởi phần mềm thì sau đó CPU sẽ thoát khỏi trạng thái Halt khi có cạnh lên của xung clock kế.

Chu kỳ tiếp theo là đáp ứng yêu cầu ngắt tùy thuộc vào ngắt nhận được. Nếu cả 2 ngắt cùng xảy ra thì ngắt có mức ưu tiên cao hơn sẽ được đáp ứng chính là ngắt  $\overline{NMI}$ .

Chức năng thực hiện lệnh NOP khi CPU ở trạng thái Halt nhằm để làm tươi bộ nhớ RAM động. Mỗi chu kỳ trong trạng thái Halt được xem là chu kỳ M1 (đón lệnh) chỉ ngoại trừ dữ liệu nhận từ bộ nhớ bị bỏ qua và lệnh NOP được thực hiện bên trong vi xử lý. Tín hiệu trả lời HALT sẽ ở trạng thái tích cực để báo cho biết vi xử lý đang ở trạng thái HALT.



Hình 2-21. Giản đồ thời gian của lệnh HALT.

## 5. ĐÁP ỨNG YÊU CẦU NGẮT CỦA VI XỬ LÝ Z80:

### Giới thiệu:

Một yêu cầu ngắt cho phép các thiết bị ngoại vi ngừng hoạt động của CPU và yêu cầu CPU phải thực hiện chương trình con phục vụ ngắt của thiết bị yêu cầu ngắt đó. Chương trình con phục vụ ngắt bao gồm các công việc trao đổi dữ liệu, trạng thái hoặc các thông tin điều khiển giữa CPU và thiết bị ngoại vi. Khi chương trình phục vụ ngắt kết thúc thì CPU trở lại thực hiện tiếp chương trình đang gián đoạn.

### Cho phép/cấm ngắt:

Vi xử lý Z80 có 2 ngõ vào ngắt: một là ngắt có thể che được  $\overline{INT}$  (cho phép/cấm bằng phần mềm) và một ngắt không thể che được  $\overline{NMI}$ . Ngắt không thể che được có nghĩa là không thể cấm bằng phần mềm, ngắt này thường được sử dụng cho các chức năng quan trọng được lựa chọn bởi người lập trình.

Đối với ngắt có thể che được  $\overline{INT}$  thì Z80 có một flip flop cho phép ngắt (Interrupt Flip Flop = IFF) và người lập trình có thể set hay reset bằng cách dùng các lệnh cho phép ngắt EI

(Enable Interrupt) và lệnh cấm ngắt DI (Disable Interrupt). Khi bit IFF bị reset thì CPU sẽ không thực hiện yêu cầu ngắt  $\overline{INT}$ .

Có 2 bit flip flop như hình 2-22:



**Hình 2-22. Hai bit flip flop IFF1 và IFF2 của ngắt  $\overline{INT}$ .**

Khi CPU bị reset thì CPU sẽ reset cả 2 bit IFF1 và IFF2 để cấm ngắt. Các yêu cầu có thể được phép tại bất kỳ thời điểm nào phụ thuộc vào lệnh EI của người lập trình. Khi lệnh EI được thực hiện thì bất kỳ yêu cầu ngắt nào đang đợi (vì chưa được phép) thì sẽ được đáp ứng ngay sau khi CPU thực hiện xong lệnh EI.

Một lệnh trì hoãn là rất cần thiết được thực hiện trước khi thực hiện lệnh kết thúc chương trình con phục vụ ngắt trở lại chương trình chính. Các yêu cầu ngắt không được phép cho đến khi thực hiện xong lệnh trở về. Khi CPU thực hiện yêu cầu ngắt có thể che được thì cả 2 bit IFF1 và IFF2 bị reset để cấm các yêu cầu ngắt khác cho đến khi thực hiện lại lệnh EI.

Vai trò của bit IFF2 là cất trạng thái của IFF1 khi xảy ra yêu cầu ngắt không ngăn được  $\overline{NMI}$ . Khi yêu cầu ngắt  $\overline{NMI}$  được chấp nhận thì bit IFF1 bị reset để cấm các yêu cầu ngắt khác xảy ra cho đến khi được phép trở lại bởi người lập trình. Vậy sau khi ngắt  $\overline{NMI}$  được chấp nhận, các yêu cầu ngắt có thể ngăn được bị cấm nhưng trạng thái trước đó của IFF1 được cất để tiếp tục cho phép ngắt sau khi CPU thực hiện xong yêu cầu ngắt  $\overline{NMI}$ . Khi lệnh “LD A,I” hoặc lệnh “LD A,R” được thực hiện thì trạng thái của IFF2 được copy vào cờ chẵn lẻ trong thanh ghi trạng thái để kiểm tra hay để lưu trữ.

Một phương pháp khác để khôi phục lại trạng thái của IFF1 bằng cách thực hiện lệnh trở về từ lệnh ngắt không ngăn được RETI (return from non – maskable interrupt). Lệnh RETI là lệnh kết thúc chương trình con phục vụ ngắt để trở về chương trình chính và nội dung của IFF2 được copy trở lại IFF1 để cho trạng thái của IFF1 trước khi chấp nhận yêu cầu ngắt không ngăn được tự động khôi phục.

Bảng liệt kê trạng thái các flip flop cho phép /cấm ngắt:

Hoạt động	IFF1	IFF2	Diễn giải
Khi CPU bị reset	0	0	Ngắt ngăn được bị cấm
Thực hiện lệnh DI	0	0	Ngắt ngăn được bị cấm
Thực hiện lệnh EI	1	1	Ngắt ngăn được được phép
Thực hiện lệnh LD A,I	*	*	IFF2 $\Rightarrow$ cờ chẵn lẻ
Thực hiện lệnh LD A,R	*	*	IFF2 $\Rightarrow$ cờ chẵn lẻ
Thực hiện ngắt $\overline{NMI}$	0	*	Ngắt ngăn được bị cấm
Thực hiện lệnh RETI	IFF2	*	IFF2 $\Rightarrow$ xác định kết thúc chương

			trình con phục vụ ngắn không thể ngăn được.
--	--	--	--

**Bảng 2-1. Bảng liệt kê các trạng thái bit IFF1 và IFF2.****Ngắn không ngăn được:**

CPU thường chấp nhận yêu cầu ngắn không ngăn được. Khi ngắn này xảy ra thì CPU không thực hiện lệnh kế mà tiến hành thực hiện lệnh bắt đầu tại địa chỉ 0066H. Hoạt động này của CPU giống như thực hiện lệnh khởi động lại nhưng khác địa chỉ. Lệnh khởi động đơn thuần chỉ là lệnh gọi chương trình tại một địa chỉ đặc biệt trong trang thứ 0 của bộ nhớ.

CPU có thể được lập trình để đáp ứng ngắn có thể ngăn được bằng một trong 3 cách.

**Mode 0:**

Trong mode này thiết bị yêu cầu ngắn có thể đặt bất kỳ lệnh nào lên bus dữ liệu và CPU sẽ thực hiện. Do đó thiết bị yêu cầu ngắn cung cấp lệnh kế để được thực hiện. Thường là lệnh khởi động lại bởi vì thiết bị yêu cầu ngắn chỉ cần cung cấp một byte mã lệnh duy nhất.

Tương tự đối với lệnh khác chẳng hạn như lệnh call (có 3 byte mã lệnh) thì chương trình có thể thực hiện tại bất kỳ vị trí nào trong bộ nhớ.

Số lượng chu kỳ xung clock cần thiết để thực hiện đáp ứng ngắn nhiều hơn thực hiện lệnh bình thường 2 chu kỳ xung clock. Lý do là CPU tự động thêm vào 2 chu kỳ đợi để chu kỳ đáp ứng yêu cầu ngắn có đủ thời gian thực hiện. Sau khi reset thì CPU sẽ tự động hoạt động ở mode 0.

**Mode 1:**

Khi mode 1 được lựa chọn bởi người lập trình thì CPU đáp ứng ngắn bằng cách thực hiện chương trình bắt đầu tại địa chỉ 0038H giống như ngắn không ngăn được thì thực hiện ở địa chỉ 0066H.

Số lượng chu kỳ yêu cầu để thực hiện nhiều hơn 2 chu kỳ do CPU cộng thêm 2 chu kỳ đợi.

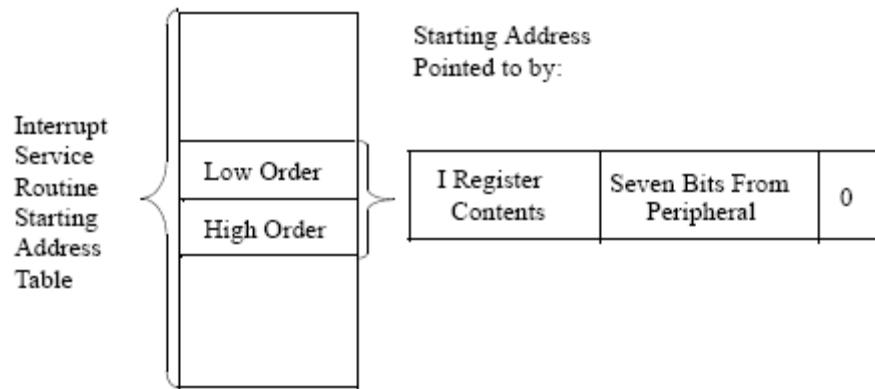
**Mode 2:**

Đây là mode mạnh nhất trong các mode đáp ứng yêu cầu ngắn. Với byte 8 bit duy nhất từ người dùng thì một lệnh gọi gián tiếp có thể được thực hiện ở bất kỳ ô nhớ nào.

Trong mode này người lập trình thiết lập một bảng các địa chỉ bắt đầu 16 bit cho mỗi chương trình con phục vụ ngắn như hình 2-23. Bảng này có thể được định vị ở bất kỳ vị trí nào trong bộ nhớ. Khi ngắn được chấp nhận thì con trỏ 16 bit phải được thiết lập để có được địa chỉ bắt đầu chương trình con phục vụ ngắn yêu cầu trong bảng địa chỉ.

8 bit địa chỉ cao của con trỏ được thiết lập từ nội dung của thanh ghi I, thanh ghi I phải được nạp giá trị địa chỉ của chương trình con bởi người lập trình bằng lệnh “LD I,A”. Khi CPU bị reset thì giá trị trong thanh ghi I bằng 00H.

8 bit địa chỉ thấp của con trỏ phải được cung cấp bởi thiết bị ngắn. Chỉ có 7 bit được yêu cầu từ thiết bị yêu cầu ngắn còn bit LSB phải bằng 0. Điều này là bắt buộc bởi vì con trỏ được sử dụng để lấy 2 byte liên tiếp để tạo địa chỉ bắt đầu của chương trình con phục vụ ngắn 16 bit và các địa chỉ phải luôn bắt đầu tại ô nhớ có địa chỉ chẵn.

**Hình 2-23. Đáp ứng ngắt ở mode 2.**

Byte thứ nhất trong bảng là byte địa chỉ thấp. Người lập trình phải thiết lập đầy đủ địa chỉ chính xác trước khi yêu cầu được chấp nhận.

Người lập trình có thể thay đổi bảng địa chỉ để một yêu cầu ngắt có thể thực hiện nhiều chương trình con phục vụ ngắt khác nhau.

Khi thiết bị yêu cầu ngắt cung cấp địa chỉ thấp đến con trỏ thì CPU tự động cất địa chỉ của thanh ghi PC vào bộ nhớ ngăn xếp, nhận địa chỉ bắt đầu của chương trình con phục vụ ngắt từ bảng và bắt đầu thực hiện chương trình tại địa chỉ đó.

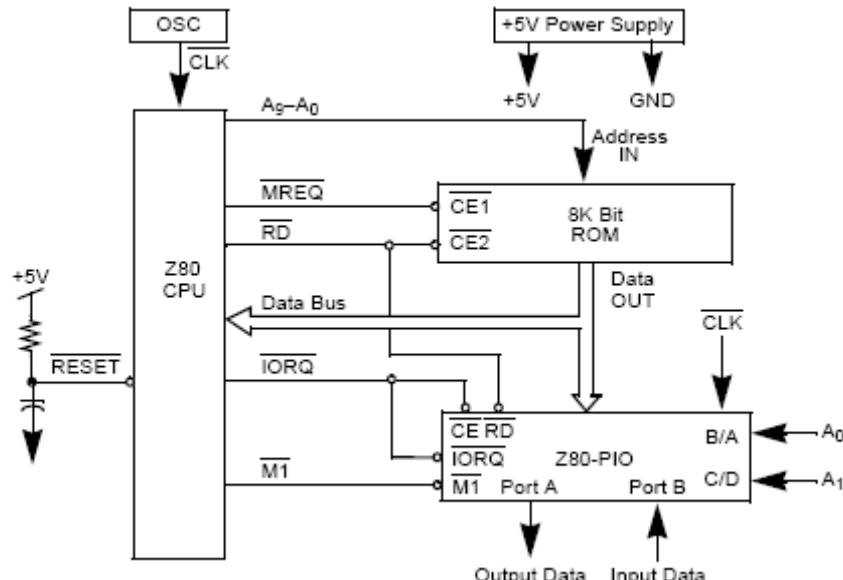
Mode này cần 19 chu kỳ xung clock để thực hiện (7 chu kỳ để đón 8 bit địa chỉ thấp từ thiết bị yêu cầu ngắt, 6 chu kỳ để cất nội dung thanh ghi PC vào bộ nhớ ngăn xếp và 6 chu kỳ để nhảy đến địa chỉ của chương trình con phục vụ ngắt).

## 6. PHẦN CỨNG VÀ PHẦN MỀM CỦA VI XỬ LÝ Z80:

### a. Phần cứng:

#### Hệ thống tối thiểu:

Trong phần này sẽ giới thiệu các hệ thống ứng dụng dùng vi xử lý Z80. Hình 2-24 trình bày một hệ thống đơn giản dùng Z80.

**Hình 2-24. Hệ thống tối thiểu dùng vi xử lý Z80.**

Bất kỳ hệ thống dùng vi xử lý Z80 đều phải có các khối sau:

- Khối nguồn cung cấp 5V.
- Khối dao động.
- Các IC nhớ.
- Các mạch điện kết nối với thiết bị ngoại vi.
- CPU Z80.

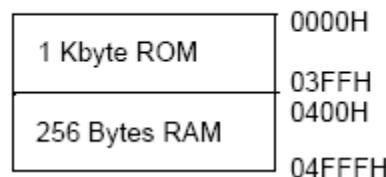
Bộ nhớ ngoại của Z80 có thể là tổng hợp 2 bộ nhớ chuẩn RAM, ROM hoặc PROM. Trong hình 2-24 có bộ nhớ ROM 8Kbit (1kbyte), các thanh ghi bên trong đủ để làm các ô nhớ có thể đọc ghi nên không cần thêm bộ nhớ Ram bên ngoài.

Các mạch IO cho phép hệ thống giao tiếp với các thiết bị bên ngoài. Trong hình 2-24 có 8 ngõ ra điều khiển và có 8 ngõ vào nhận tín hiệu từ bên ngoài. Các dữ liệu từ bên ngoài có thể đưa vào bên trong vi xử lý thông qua khối điều khiển 3 trạng thái, khối ngõ ra được chốt bởi mạch chốt dùng TTL chuẩn. IC IO Z80 PIO sẽ thực hiện chức năng này.

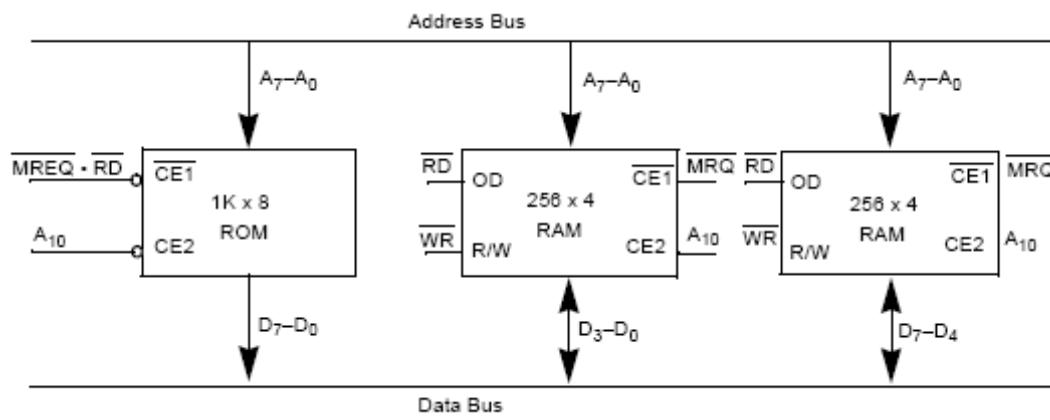
#### Mở rộng bộ nhớ RAM :

Hầu hết các hệ thống máy tính đều dùng bộ nhớ mở rộng Read/Write để lưu trữ dữ liệu và làm bộ nhớ ngắn xếp. Hình 2-25 trình bày bộ nhớ có 256 byte bộ nhớ RAM tĩnh được thêm vào hệ thống ở hình 2-24. Không gian bộ nhớ được thêm vào có địa chỉ như sau:

Address:



Địa chỉ của bộ nhớ được mô tả ở dạng số HEX. Khi phân tích ra thành số nhị phân thì bit địa chỉ A10 sẽ là đường phân chia bộ nhớ ROM và bộ nhớ RAM nên sẽ dùng đường địa chỉ này để điều khiển chọn chip. Khi lượng bộ nhớ mở rộng khá lớn có nhiều RAM, nhiều ROM thì phải dùng mạch giải mã TTL để thực hiện chọn chip.

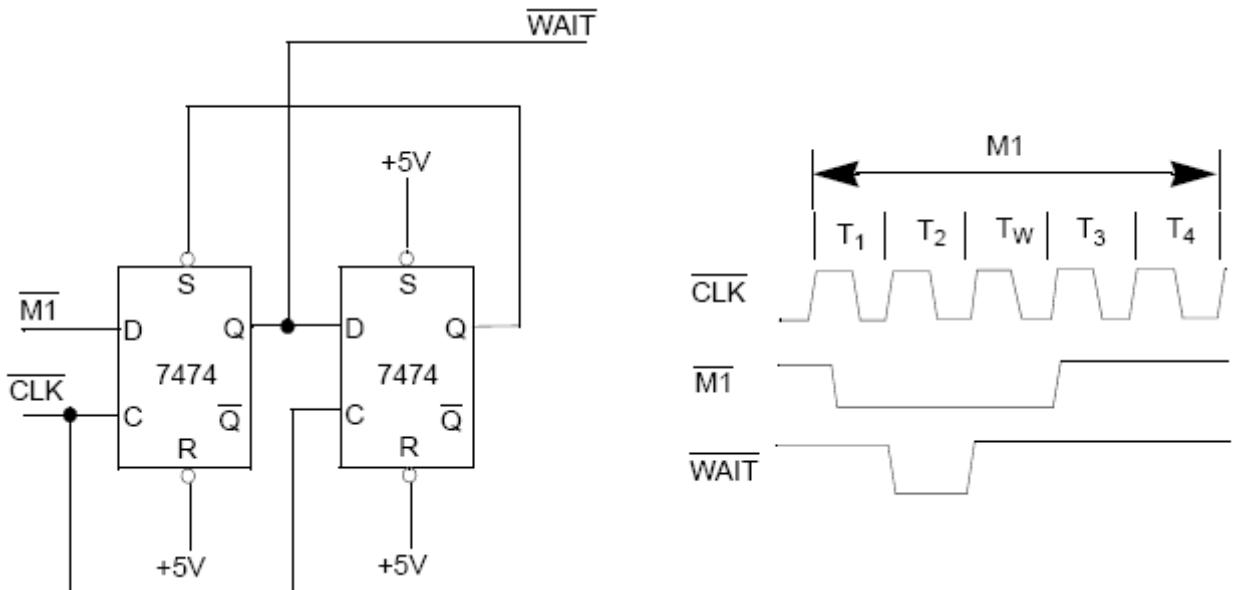


Hình 2-25. Mở rộng thêm bộ nhớ.

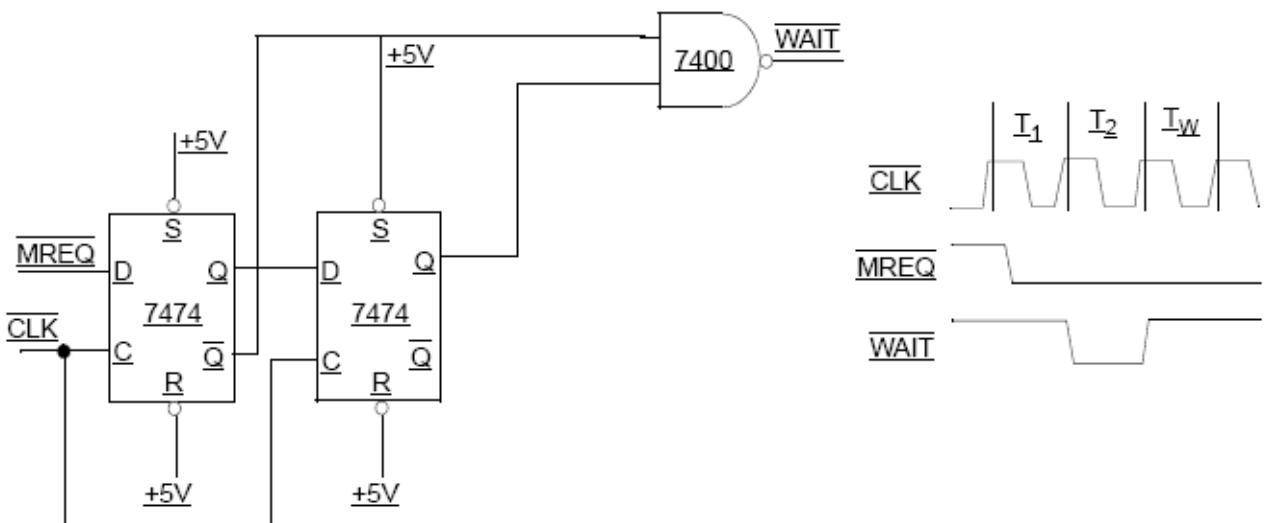
#### Điều khiển tốc độ bộ nhớ :

Bộ nhớ có tốc độ thấp thì giá thành cũng thấp dẫn đến giá thành của hệ thống cũng thấp. Tín hiệu WAIT của vi xử lý cho phép vi xử lý có thể làm việc với các bộ nhớ có tốc độ bất kỳ nhanh hay chậm. Như đã trình bày ở phần giản đồ thời gian truy xuất bộ nhớ, tất cả các chu kỳ truy xuất bộ nhớ sẽ hoàn tất trong khoảng thời gian nữa chu kỳ, tuy nhiên trong một số trường hợp cần phải thêm một chu kỳ đợi trong chu kỳ M1 để có thể sử dụng các bộ nhớ có tốc độ chậm.

Hình 2-26 là một mạch điện đơn giản để cộng thêm một chu kỳ đợi. Mạch điện hình 2-26 có thể thay đổi như hình 2-27 để có thể cộng thêm một chu kỳ đợi để truy xuất bất kỳ bộ nhớ nào.



Hình 2-26. Thêm một chu kỳ đợi vào chu kỳ M1.



Hình 2-27. Thêm một chu kỳ đợi để truy xuất bộ nhớ bất kỳ.

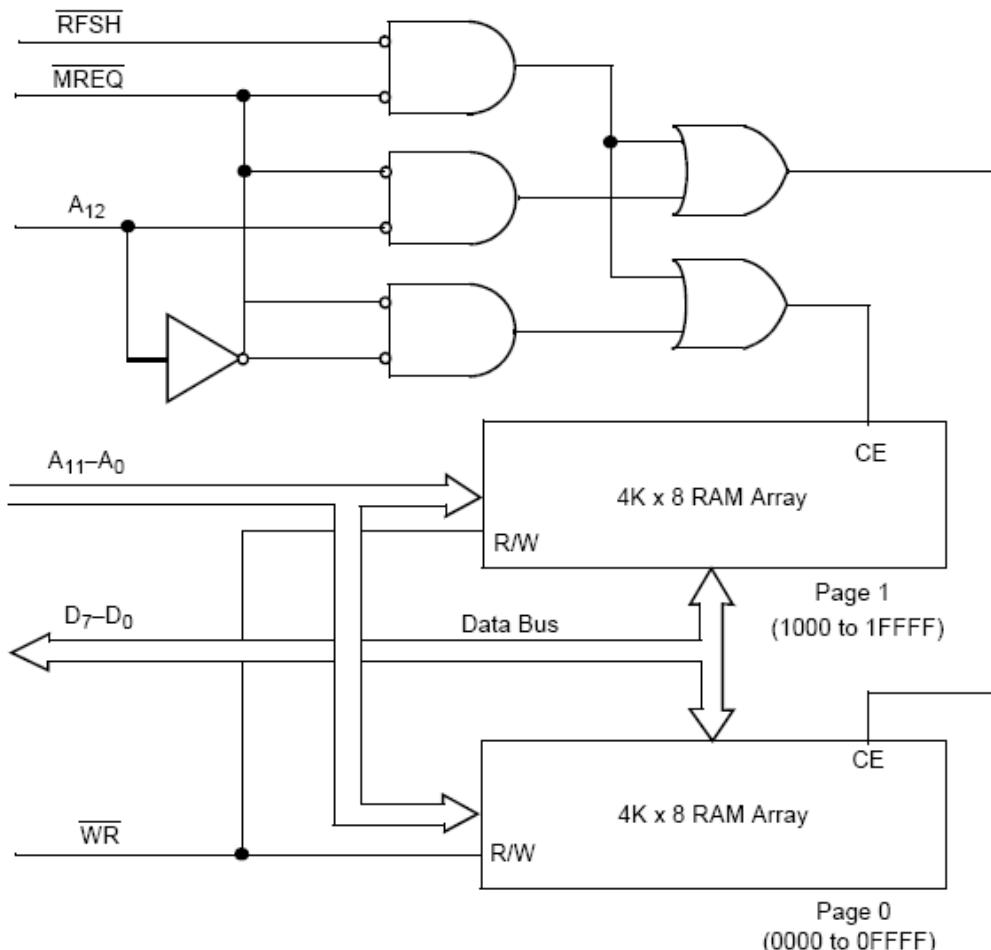
#### Giao tiếp với bộ nhớ động:

Mỗi bộ nhớ RAM động có các thông số làm việc. Hình 2-28 minh họa mạch điện dùng các cổng logic để giao tiếp bộ nhớ RAM động 8 kbyte sử dụng 2 bộ nhớ 4Kbyte.

Giả sử đường địa chỉ A12 được dùng làm đường điều khiển chọn chip select.

Trong khoảng thời gian làm tươi bộ nhớ, tất cả các bộ nhớ trong hệ thống phải được đọc. CPU cung cấp địa chỉ làm tươi bộ nhớ trên các đường địa chỉ từ A0 đến A6. Khi mở rộng bộ nhớ cho hệ thống này thì chỉ cần thay thế các cổng logic hoạt động với đường địa chỉ A12 bằng mạch giải mã hoạt động với các đường địa chỉ còn lại tùy dung lượng bộ nhớ mở rộng thêm.

Cần phải có các bộ đệm cho các đường địa chỉ khi hệ thống giao tiếp nhiều bộ nhớ.



Hình 2-28. Thêm một chu kỳ đợi để truy xuất bộ nhớ bất kỳ.

### b. Phần mềm và các chương trình ví dụ :

#### Tổng quan về cấu trúc phần mềm :

Tập lệnh của vi xử lý Z80 cung cấp cho người sử dụng tất cả các hoạt động để điều khiển CPU Z80. Các thanh ghi chính, thanh ghi luân phiên và các thanh ghi chỉ số có thể lưu trữ dữ liệu cho các lệnh logic và lệnh số học, lưu trữ địa chỉ của bộ nhớ hoặc lưu trữ dữ liệu.

Dữ liệu có thể chuyển từ thanh ghi sang thanh ghi, từ bộ nhớ sang bộ nhớ, từ bộ nhớ sang thanh ghi hoặc từ thanh ghi sang bộ nhớ. Bên cạnh đó còn có các lệnh trao đổi dữ liệu giữ các thanh ghi, giữa thanh ghi và bộ nhớ.

Đặc biệt là nội dung của các thanh ghi chính và thanh ghi luân phiên có thể trao đổi bằng hai lệnh duy nhất là EX và EXX. Lệnh trao đổi dữ liệu của các thanh ghi được dùng để chia tập các thanh ghi đang làm việc khác với tập thanh ghi trong chương trình con hay trong thủ tục.

Việc cất dữ liệu của các thanh ghi tạm thời vào bộ nhớ ngắn xếp và lấy dữ liệu ra khỏi bộ nhớ ngắn xếp có thể được thực hiện bằng lệnh Push và lệnh Pop. Địa chỉ của bộ nhớ ngắn xếp được quản lý bởi thanh ghi SP.

Khi thực hiện chương trình con thì địa chỉ của lệnh kế trong thanh ghi PC được CPU tự động cất vào bộ nhớ ngắn xếp và khi kết thúc chương trình con bằng lệnh trở về thì địa chỉ được trả lại cho thanh ghi PC và chương trình chính được thực hiện tiếp tục.

Kết quả thực hiện của các lệnh logic, lệnh số học, lệnh dịch, lệnh so sánh sẽ điều khiển 6 cờ trạng thái trong thanh ghi trạng thái.

Sau khi thực hiện một lệnh thì các thanh ghi cờ sẽ bị tác động tùy thuộc vào kết quả sau khi xử lý, các cờ được dùng để làm điều kiện cho lệnh nhảy hoặc lệnh trở về.

Một nhóm lệnh đầy đủ các lệnh logic bao gồm lệnh AND, OR, XOR, CPL, NEG có thể thực hiện với dữ liệu 8 bit giữa thanh ghi A và tất cả các thanh ghi còn lại, giữa thanh ghi A và bộ nhớ và giữa thanh ghi A và dữ liệu tức thời.

Ngoài ra còn có các lệnh số học và các lệnh dịch theo cả hai chiều, lệnh này thực hiện được đối với dữ liệu 8 bit chứa trong các thanh ghi 8 bit hoặc trong một ô nhớ có địa chỉ bất kỳ. Cờ carry có thể được sử dụng trong các lệnh xoay, lệch dịch để tạo dữ liệu đầu vào hay nhận dữ liệu dịch ở đầu ra để kiểm tra kết quả của lệnh dịch.

### Các chương trình ví dụ :

#### Ví dụ 1:

Chương trình thực hiện việc copy 737 byte dữ liệu trong vùng nhớ bắt đầu tại địa chỉ DATA sang vùng nhớ có địa chỉ bắt đầu BUFFER như sau:

LD	HL, DATA	; HL quản lý địa chỉ bắt đầu vùng dữ liệu DATA
LD	DE, BUFFER	; DE quản lý địa chỉ bắt đầu vùng dữ liệu BUFFER
LD	BC, 737	; BC quản lý chiều dài vùng dữ liệu.
LDIR		; copy chuỗi được chỉ định bởi HL sang vùng nhớ được chỉ định bởi DE, giảm HL, giảm DE và giảm BC tiếp tục thực hiện cho đến khi BC = 0.

Khi biên dịch thì đoạn chương trình trên chiếm 11 byte và mỗi một byte khi di chuyển mất 21 chu kỳ xung clock.

#### Ví dụ 2:

Chương trình thực hiện việc copy một chuỗi (có chiều dài tối đa là 132) bắt đầu tại địa chỉ bắt đầu DATA sang một vùng nhớ có địa chỉ bắt đầu BUFFER cho đến khi kí tự \$ được tìm thấy:

LD	HL, DATA	; HL quản lý địa chỉ bắt đầu vùng dữ liệu DATA
LD	DE, BUFFER	; DE quản lý địa chỉ bắt đầu vùng dữ liệu BUFFER
LD	BC, 132	; BC quản lý chiều dài vùng dữ liệu.

LD	A, '\$'	;Nạp kí tự “\$” vào A để so sánh.
LOOP:	CP (HL)	;So sánh nội dung ô nhớ với A.
JR	Z, END	;nhảy kết thúc nếu đúng là kí tự ‘\$’.
LDI		;không đúng thì tiến hành copy dữ liệu và tăng ....
JP	PE, LOOP	;nhảy lại làm tiếp cho đến khi hết 132 kí tự.
END:		

Khi biên dịch thì đoạn chương trình trên chiếm 19 byte.

### c. Mô tả lệnh của vi xử lý Z80 :

#### Tổng quan về tập lệnh:

Vi xử lý Z80 có 158 lệnh khác nhau bao gồm luôn cả 78 lệnh của vi xử lý 8080A. Các lệnh này được chia ra làm các nhóm như sau:

- Lệnh nạp và trao đổi dữ liệu.
- Lệnh chuyển khối dữ liệu và lệnh tìm kiếm.
- Lệnh số học và lệnh logic.
- Lệnh xoay và lệnh dịch.
- Lệnh điều khiển bit: set, reset và test.
- Lệnh nhảy, lệnh gọi chương trình con và lệnh kết thúc chương trình con.
- Lệnh xuất nhập IO
- Các lệnh điều khiển CPU.

#### Các kiểu định địa chỉ:

Hầu hết các lệnh của vi xử lý Z80 đều xử lý dữ liệu được lưu trữ ở các thanh ghi bên trong CPU, lưu trữ ở bộ nhớ ngoài hoặc lưu trữ ở IO. Kiểu định địa chỉ để chỉ rõ địa chỉ của dữ liệu mà lệnh sẽ xử lý.

Các kiểu định địa chỉ của vi xử lý Z80 giống như những kiểu đã trình bày ở phần trước, các kiểu bao gồm:

#### Kiểu định địa chỉ tức thời (Immediate):

**Kiểu định địa chỉ tức thời mở rộng (Immediate Extended):** giống như kiểu định địa chỉ tức thời như dữ liệu là 16 bit được nạp cho cặp thanh ghi 8 bit.

**Kiểu định địa chỉ hiệu chỉnh trang bộ nhớ zero :** Z80 có 1 lệnh gọi đặc biệt chỉ có 1 byte để xác định 1 trong 8 vị trí trong trang bộ nhớ zero. Mã lệnh của lệnh này chỉ là 1 byte nhưng cho phép thành lập địa chỉ 16 bit và lệnh này được dùng để gọi các chương trình thường sử dụng nằm trong trang bộ nhớ để tiết kiệm dung lượng bộ nhớ (vì lệnh 1 byte thay vì lệnh 3 byte).

Op Code	One Byte
B7	B0 Effective Address is (B5 B4 B3 000)2

**Kiểu định địa chỉ tương đối (relative addressing):**

**Kiểu định địa chỉ mở rộng(Extended ):**

Kiểu này cung cấp luôn 2 byte (16 bit) của địa chỉ trong lệnh. 2 byte địa chỉ có thể là địa chỉ của lệnh nhảy hay là địa chỉ của dữ liệu.

**Kiểu định địa chỉ chỉ số (Indexed addressing):**

**Kiểu định địa chỉ thanh ghi (register addressing):**

**Kiểu định địa chỉ hiểu ngầm (Implied addressing):**

**Kiểu định địa chỉ gián tiếp dùng thanh ghi (register indirect addressing):**

**Kiểu định địa chỉ bit (Bit addressing):**

**Mô tả vài lệnh của vi xử lý Z80:**

**Lệnh LD giữa các thanh ghi :**

Hình 2-29 tóm tắt lệnh chuyển dữ liệu từ thanh ghi nguồn 8 bit “reg8S” sang thanh ghi đích 8 bit “reg8D”.

Mã lệnh 1 byte 8 bit bao gồm 2 bit “01[reg8D][reg8S]”: trong đó [reg8D] là 3 bit của thanh ghi “reg8D” và [reg8S] là 3 bit của thanh ghi “reg8S”.

Mỗi thanh ghi có 3 bit được trình bày trong hình, thời gian thực hiện lệnh mất 4 chu kỳ T và mất 7 chu kỳ T đối với lệnh chuyển dữ liệu giữa ô nhớ (có địa chỉ lưu trong HL) và thanh ghi. Nếu ghi chi tiết cho từng thanh ghi thì lệnh tổng quát này có tổng cộng 64 lệnh: 8 lệnh cho thanh ghi A và các thanh ghi, 8 lệnh cho thanh ghi B và các thanh ghi, ...

Chú ý: (HL) là ô nhớ có địa chỉ lưu trong HL được xem như là 1 thanh ghi có mã là [110].

## LD

**LD reg8D,reg8S**

<b>Operation</b>	The contents of register <b>reg8S</b> are stored into <b>reg8D</b> .																		
<b>Op Code</b>	<b>01[reg8D][reg8S]</b>																		
	<table border="1"> <thead> <tr> <th>Register</th> <th>Bit Field</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>111</td> </tr> <tr> <td>B</td> <td>000</td> </tr> <tr> <td>C</td> <td>001</td> </tr> <tr> <td>D</td> <td>010</td> </tr> <tr> <td>E</td> <td>011</td> </tr> <tr> <td>H</td> <td>100</td> </tr> <tr> <td>L</td> <td>101</td> </tr> <tr> <td>(HL)</td> <td>110</td> </tr> </tbody> </table>	Register	Bit Field	A	111	B	000	C	001	D	010	E	011	H	100	L	101	(HL)	110
Register	Bit Field																		
A	111																		
B	000																		
C	001																		
D	010																		
E	011																		
H	100																		
L	101																		
(HL)	110																		
<b>T States</b>	<b>4 or 7 (HL)</b>																		

**Hình 2-29. Lệnh tổng quát “LD reg8D,reg8S”.**

Hình 2-30 tóm tắt lệnh nạp dữ liệu tức thời 8 bit vào thanh ghi 8 bit “reg8”.

Mã lệnh 1 byte 8 bit bao gồm 2 bit “01[reg8]110”: trong đó [reg8D] là 3 bit của thanh ghi “reg8”. Thời gian thực hiện lệnh mất 7 chu kỳ T và mất 10 chu kỳ T đối với lệnh nạp dữ liệu vào ô nhớ (có địa chỉ lưu trong HL).

**LD reg8,imm8**

<b>Operation</b>	Stores the immediate value into <b>reg8</b> .																		
<b>Op Code</b>	<b>00[reg8]110 : [imm8]</b>																		
	<table border="1"> <thead> <tr> <th>Register</th> <th>Bit Field</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>111</td> </tr> <tr> <td>B</td> <td>000</td> </tr> <tr> <td>C</td> <td>001</td> </tr> <tr> <td>D</td> <td>010</td> </tr> <tr> <td>E</td> <td>011</td> </tr> <tr> <td>H</td> <td>100</td> </tr> <tr> <td>L</td> <td>101</td> </tr> <tr> <td>(HL)</td> <td>110</td> </tr> </tbody> </table>	Register	Bit Field	A	111	B	000	C	001	D	010	E	011	H	100	L	101	(HL)	110
Register	Bit Field																		
A	111																		
B	000																		
C	001																		
D	010																		
E	011																		
H	100																		
L	101																		
(HL)	110																		
<b>T States</b>	<b>7 or 10 (HL)</b>																		

Hình 2-30. Lệnh tổng quát “LD reg8,imm8”.

Hình 2-31 tóm tắt lệnh xoay trái thanh ghi 8 bit “reg8”. Mã lệnh 2 byte 8 bit: “11001011 : 00010 [reg8]”. Thời gian thực hiện lệnh mất 8 chu kỳ T và mất 15 chu kỳ T đối với lệnh xoay dữ liệu trong ô nhớ (có địa chỉ lưu trong HL).

**RL****RL reg8**

<b>Operation</b>	The contents of <b>reg8</b> are rotated left one bit position. The contents of bit 7 are copied to the carry flag and the previous contents of the carry flag are copied to bit 0.																		
<b>Op Code</b>	<b>11001011 : 00010[reg8]</b>																		
	<table border="1"> <thead> <tr> <th>Register</th> <th>Bit Field</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>111</td> </tr> <tr> <td>B</td> <td>000</td> </tr> <tr> <td>C</td> <td>001</td> </tr> <tr> <td>D</td> <td>010</td> </tr> <tr> <td>E</td> <td>011</td> </tr> <tr> <td>H</td> <td>100</td> </tr> <tr> <td>L</td> <td>101</td> </tr> <tr> <td>(HL)</td> <td>110</td> </tr> </tbody> </table>	Register	Bit Field	A	111	B	000	C	001	D	010	E	011	H	100	L	101	(HL)	110
Register	Bit Field																		
A	111																		
B	000																		
C	001																		
D	010																		
E	011																		
H	100																		
L	101																		
(HL)	110																		
<b>Flags</b>	S Z are affected as defined H N are reset P/V is parity See instruction for C																		
<b>T States</b>	<b>8 or 15 (HL)</b>																		

Hình 2-31. Lệnh tổng quát “RL reg8”.

Do có rất nhiều lệnh nên ở phần này chỉ trình bày một số lệnh, có thể xem thêm ở phần phụ lục.

**d. Tóm tắt tập lệnh của vi xử lý Z80:**

Tập lệnh được tóm tắt ở các trang tiếp theo với nguồn thông tin được lấy từ mạng internet với địa chỉ “<http://www.ftp83plus.net/Tutorials/z80inset1A.htm>”

Mnemonic	Operation	CZPSNTH	Len	Tim	Comments
ADC A,r	A := A+r+CY	***V*0*	1	4	r - A,B,C,D,E,H,L
ADC A,(HL)	A := A+(HL)+CY		1	7	
ADC A,n	A := A+n+CY		2	7	n - byte (0..FF)
ADC A,(ii+n)	A := A+(ii+n)+CY		3	19	ii - IX,IY
ADC HL,rr	HL := HL+rr+CY	***V*0*	2	15	rr - BC,DE,HL,SP
ADC A,r	A := A+r	***V*0*	1	4	
ADC A,(HL)	A := A+(HL)		1	7	
ADC A,n	A := A+n		2	7	
ADC A,(ii+n)	A := A+(ii+n)		3	19	
ADC HL,rr	HL := HL+rr	*...0x	1	11	IY - BC,DE,SP,IY
ADD IX,ry	X := IX+px		2	15	rx - BC,DE,SP,IX
ADD IY,rx	IY := IY+py		2	15	
AND r	A := A and r	0*P*01	1	4	
AND (HL)	A := A and (HL)		1	7	
AND n	A := A and n		2	7	
AND (ii+n)	A := A and (ii+n)		3	19	
BIT b,r	Z := not rb	.***01	2	8	b - bit number (0..7)
BIT b,(HL)	Z := not (HL)b		2	12	zb - bit b of
BIT b,(ii+n)	Z := not (ii+n)b		4	20	location z
CALL nn	PUSH PC;PC:=nn	.....	3	17	nn - word (0..FFFF)
CALL cc,nn	If cc then CALL nn		3	17	cc - C,NC,Z,NZ,M,P,PE,PO
	else continue		10		
CCF	CY := not CY	*...0x	1	4	
CP r	A-r	***V*1*	1	4	
CP (HL)	A-(HL)		1	7	
CP n	A-n		2	7	
CP (ii+n)	A-(ii+n)		3	19	
CDP	A-(HL);dec HL;dec BC	***1*	2	16	PV=0 if BC=0, else PV=1
CPDR	Repeat CPD until Z=1 or BC=0	***1*	2	21	
			16		

CPI	A-(HL);inc HL;dec BC	***1*	2	16	PV=0 if BC=0,else PV=1
CPIR	Repeat CPI	***1*	2	21	
CPL	A:=A xor 255	....11	1	4	
DAA	Decimal adjust Acc.	**P*.**	1	4	
DEC r	r:=r-1	.**V**1*	1	4	
DEC (HL)	(HL):=(HL)-1		1	11	
DEC (ii+n)	(ii+n):=(ii+n)-1		3	23	
DEC rr	rr:=rr-1	.....	1	6	
DEC ii	ii:=ii-1		2	10	
DI	IFF:=0	.....	1	4	
DJNZ e	dec B;if B > 0 JR e if B=0 continue	.....	2	13	e - adresse relative
EI	IFF:=1	.....	1	4	
EX AF,AF'	AF<->AF'	.....	1	4	
EX DE,HL	DE<->HL		1	4	
EX (SP),HL	(SP)<->HL		1	4	
EX (SP),ii	(SP)<->ii		2	23	
EXX	BC<->BC';DE<->DE'; HL<->HL'	.....	1	4	
Halt	Halte le processeur	.....	1	4	
IM 1	Interrupt mode 1	.....	2	8	
IM 2	Interrupt mode 2				
IM 3	Interrupt mode 3				
IN A,(n)	A:=port(n)	.....	2	11	
IN r,(C)	r:=port(C)		2	12	
IN ?,(C)	only set flags as as IN r,(C) does		2	12	can't be entered as a command;code is ED 70

INC r	r:=r+1	*V*0*	1	4
INC (HL)	(HL):=(HL)+1		1	11
NC (ii+n)	(ii+n):=(ii+n)+1		3	23
INC rr	rr:=rr+1		1	6
INC ii	ii:=ii+1		2	10
IND	(HL):=port(C); dec HL;dec B	x***1x	2	16
INDR	Repeat IND until B=0	x1xx1x	2	21
INT	(HL):=port(C); inc HL;dec B	x***1x	2	16
INTR	Repeat INTI until B=0	x1xx1x	2	21
JP nn	PC:=nn	.....	3	10
JP cc,nn	If cc then JP nn	.....	3	10
JP (HL)	PC:=HL		1	4
JP (ii)	PC:=ii		2	8
JR e	PC:=PC+e	.....	2	12
JR cond,e	If cond then JR e else NOP	.....	2	12
LD r,r	r:=r	.....	1	4
LD r,(HL)	r:=(HL)	.....	1	7
LD r,n	r:=n	.....	2	7
LD r,(ii+n)	r:=(ii+n)	.....	3	19
LD (HL),r	(HL):=r	.....	1	7
LD (ii+n),r	(ii+n):=r	.....	3	19
LD (HL),n	(HL):=n	.....	2	10
LD (ii+n),n	(ii+n):=n	.....	4	19
LD A,(BC)	A:=(BC)	.....	1	7
LD A,(DE)	A:=(DE)	.....	1	7
LD A,(nn)	A:=(nn)	.....	3	13

LD (BC),A	(BC) := A	1	7	
LD (DE),A	(DE) := A	1	13	PV=IFF
LD (nn),A	(nn) := A	3	9	PV=IFF
LD A,I	A := I	2	9	
LD A,R	A := R	2	9	
LD I,A	I := A	2	9	
LD R,A	R := A	2	9	
LD rr,nn	rr := nn	3	10	
LD ii,nn	ii := nn	4	14	
LD HL,(nn)	HL := (nn)	3	16	
LD rr,(nn)	rr := (nn)	4	20	
LD ii,(nn)	ii := (nn)	4	20	
LD (nn),HL	(nn) := HL	3	16	
LD (nn),rr	(nn) := rr	4	20	
LD (nn),ii	(nn) := ii	4	20	
LD SP,HL	SP := HL	1	6	
LD SP,ii	SP := ii	1	10	
LDD	(DE) := (HL); dec DE,HL,BC	...*.00	2	PV=0 if BC=0, else PV=1
LDLR	Repeat LDD	..0.00	2	
LDI	(DE) := (HL); inc DE,HL;dec BC	...*.00	2	PV=0 if BC=0, else PV=1
LDIR	Repeat LDI	..0.00	2	
NEG	A := 0-A	**V*1*	2	
NOP	No operation	.....	1	
OR r	A := A or r	.....	1	
OR (HL)	A := A or (HL)	0*P*00	1	
OR n	A := A or n		1	
OR (ii+n)	A := A or (ii+n)		2	
			3	19

OTDR	Repeat OUTD until B=0	x1xx1x	2	21
OTIR	Repeat OUTI until B=0	x1xx1x	2	16
OUT (n),A OUT (C),R	port(n):=A port(C):=R	.....	2	21
OUTD	port(C):=(HL); dec HL;dec B	xxxx1x	2	16
OUTI	port(C):=(HL); inc HL;dec B	xxxx1x	2	16
POP qq POP ii	qq:=(SP);SP:=SP+2 ii:=(SP);SP:=SP+2	.....	1	10
PUSH qq PUSH ii	SP:=SP-2;(SP):=qq SP:=SP-2;(SP):=ii	.....	2	14
RES b,r RES b,(HL) RES b,(ii+n)	rb:=0 (HL)b:=0 (ii+n)b:=0	.....	1	11
RET RET cc	rb:=0 (HL)b:=0 (ii+n)b:=0	.....	2	15
RET RET cc	POP PC If cc then RET else NOP	.....	1	10
RETI RETN	Return from interr. Return from NMI	.....	2	14
RL r RL (HL) RL (ii+n)	+---->-----+   +---+ +-----+     +CY+<+7 <-- 0++   +--+ +-----+	**P*00 2 2 4	8 2 4	15 23
RLA	+-----+ *....00	1	4	
RLC r RLC (HL) RLC (ii+n)	+---->-----+   +-----+     CY+<+7 <-- 0++   +--+ +-----+	**P*00 2 2 4	8 2 4	15 23

RLCA	A +--->----+(HL)	*... .00	1	4
RLD	+---+---+---+---+---+   7 4   3 0   7 4   3 0   +---+---+---+---+---+---+ +--<-++-<-+	.**P*00	2	18
RR r	+---<-----+  +---+ +---+  ++7 --> 0++>CY++ + + + + + + + +	**P*00	2	8
RR (HL)			2	15
RR (ii+n)			4	23
RRA		*... .00	1	4
RRC r	+---<-----+  +---+ +---+  +---+  ++7 --> 0++>CY  + + + + + + + +	**P*00	2	8
RRC (HL)			2	15
RRC (ii+n)			4	23
RRCA		*... .00	1	4
RRD	A +---<----+(HL)	.**P*00	2	18
	+---+---+---+---+---+   7 4   3 0   7 4   3 0   +---+---+---+---+---+ +-->-++->-+			
RST adr	CALL adr	.....	1	11
SBC A,r	A:=A-r-CY	**V*1*	1	4
SBC A,(HL)	A:=A-(HL)-CY		1	7
SBC A,n	A:=A-n-CY		2	7
SBC A,(ii+n)	A:=A-(ii+n)-CY		3	19
SBC HL,rr	HL:=HL-rr-CY	**V*1x	2	15
SCF	CY:=1	1... .00	1	4
SET b,r	rb:=1	.....	2	8
SET b,(HL)	(HL)b:=1	.....	2	15
SET b,(ii+n)	(ii+n)b:=1	.....	4	23

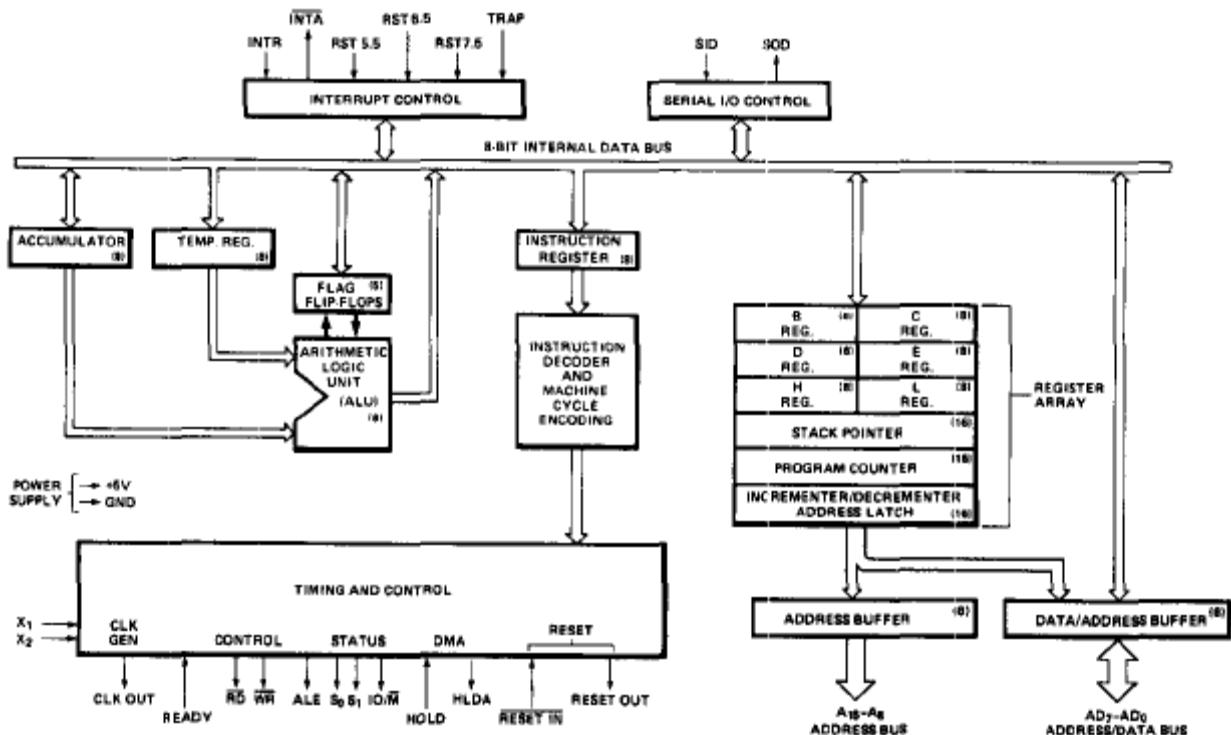
SLA r	+---+ +----+ +----+   CY<+7 <-- 0+<0	**P*00	2	8
SLA (HL)	+---+ +----+ +----+		2	15
SLA (ii+n)	+---+ +----+ +----+		4	23
SRA r	+----+ +----+ +----+   CY>+7 --> 0+>+CY	**P*00	2	8
SRA (HL)	+----+ +----+ +----+		2	15
SRA (ii+n)	+----+ +----+ +----+		4	23
SRL r	+----+ +----+ +----+   0>+7 --> 0+>+CY	**P*00	2	8
SRL (HL)	+----+ +----+ +----+		2	15
SRL (ii+n)	+----+ +----+ +----+		4	23
SUB r	A:=A-R	**V*0*	1	4
SUB (HL)	A:=A-(HL)		1	7
SUB n	A:=A-n		2	7
SUB (ii+n)	A:=A-(ii+n)		3	19
XOR r	A:=A xor R	0*P*00	1	4
XOR (HL)	A:=A xor (HL)		1	7
XOR n	A:=A xor n		2	7
XOR (ii+n)	A:=A xor (ii+n)		3	19

## V. KHẢO SÁT VI XỬ LÝ 8 BIT 8085A:

Chúng ta mới vừa khảo sát xong vi xử lý Z80 thì ở phần này sẽ tiếp tục khảo sát vi xử lý 8085A của Intel. Tác giả muốn trình bày thêm vi xử lý 8085A để chúng ta có thể làm quen với một số cấu hình khác mà vi xử lý Z80 không có.

### 1. SƠ ĐỒ CẤU TRÚC BÊN TRONG CỦA VI XỬ LÝ 8085A:

Hình 2-32 là sơ đồ cấu trúc của vi xử lý 8085A:



**Hình 2-32. Sơ đồ cấu trúc bên trong của vi xử lý 8085A.**

Trong sơ đồ cấu trúc của vi xử lý 8085A có:

- Khối ALU cùng với 2 thanh ghi có tên là thanh ghi A và thanh ghi TEMP có chức năng giống như đã khảo sát nhằm lưu trữ dữ liệu cho khối ALU thực hiện các phép toán. Kết quả sau khi thực hiện sẽ lưu trữ lại thanh ghi A và tuỳ thuộc vào kết quả mà khối ALU sẽ điều khiển các bit cờ trong thanh ghi trạng thái.
- Khối các thanh ghi gồm có: thanh ghi A, B, C, D, H, L, PC, SP, IDAL, IP.
- Bus dữ liệu bên trong D0 ÷ D7 cho phép trao đổi dữ liệu giữa các khối với nhau và được đưa tới khối đệm và đa hợp với 8 đường địa chỉ thấp A0 ÷ A7 để tạo thành bus địa chỉ / dữ liệu AD0 ÷ AD7.

**Ưu điểm:** của việc đa hợp 8 đường địa chỉ và 8 đường dữ liệu là thuận tiện cho việc đóng gói IC 40 chân, nếu không tích hợp thì IC sẽ nhiều hơn 40 chân. Có 2 lý do cho việc đa hợp: thứ nhất là tại thời điểm đó công nghệ mới bắt đầu nên chưa có thể tăng số chân cho IC và phải thay đổi công nghệ nếu số chân thay đổi, thứ 2 là việc đa hợp vẫn không làm thay đổi chức năng hay tốc độ của vi xử lý.

**Khuyết điểm:** mạch điện phức tạp vì phải đa hợp và giải đa hợp khi sử dụng để tách địa chỉ và dữ liệu độc lập với nhau, phải có thêm tín hiệu điều khiển giải đa hợp.

- Thanh ghi lệnh IR sẽ lưu trữ mã lệnh và khối giải mã lệnh dùng để giải mã lệnh cho khối điều khiển thực hiện lệnh.
- Khối điều khiển ngắn, khối truyền dữ liệu nối tiếp.

Các khối còn lại bao gồm: khối điều khiển bus, khối điều khiển đọc ghi bộ nhớ, đọc ghi IO, khối điều khiển vi xử lý, khối reset .

Nguồn sử dụng cho vi xử lý là +5V, mạch tạo xung clock được tích hợp bên trong và chỉ cần gắn thêm tụ thạch anh bên ngoài.

## 2. TỔ CHỨC THANH GHI CỦA VI XỬ LÝ 8085A:

Vi xử lý 8085A có 8 thanh ghi 8 bit và 3 thanh ghi 16 bit được trình bày như hình 2-33.

Vai trò của các thanh ghi giống như đã trình bày ở vi xử lý Z80 chỉ có khác là số thanh ghi trong vi xử lý 8085A ít hơn so với Z80. Trong nhóm thanh ghi 16 bit thì có thanh ghi IDAL có chức năng như sau: khi CPU truy xuất mã lệnh thì nội dung của thanh ghi PC được trao cho thanh ghi IDAL và nó sẽ điều khiển bus địa chỉ tạo ra đúng địa chỉ cần truy xuất mã lệnh.

Sau khi nhận mã lệnh về để thực hiện thì nội dung của PC đã tăng lên để chuẩn bit thực hiện lệnh tiếp theo, nếu lệnh đang thực hiện là truy xuất byte dữ liệu ở bộ nhớ ngoài thì CPU sẽ điều khiển thanh ghi IDAL tăng lên 1 hoặc 2 giá trị để truy xuất đúng ô nhớ cần lấy dữ liệu. Nếu địa chỉ của ô nhớ chứa trong cặp thanh ghi HL thì nội dung trong HL sẽ trao cho IDAL để truy xuất bộ nhớ.

<b>A reg (8)</b>	<b>F reg (8)</b>
<b>B reg (8)</b>	<b>C reg (8)</b>
<b>D reg (8)</b>	<b>E reg (8)</b>
<b>H reg (8)</b>	<b>L reg (8)</b>
<b>STACK POINTER</b>	<b>(16)</b>
<b>PROGRAM COUNTER</b>	<b>(16)</b>
<b>INCREMENTER/DECREMENTER</b>	
<b>ADDRESS LATCH</b>	<b>(16)</b>

Hình 2-33. Các thanh ghi bên trong VI xử lý 8085A.

## 3. SƠ ĐỒ CHÂN CỦA VI XỬ LÝ 8085A:

Hình 2-34 là sơ đồ chân của vi xử lý 8085A:

Vi xử lý 8085A có 40 chân, trong đó:

**A8 ÷ A15 (Address)** là bus địa chỉ cao, tín hiệu ra dạng 3 trạng thái, sẽ ở trạng thái tổng trở cao khi ở mode Hold và Halt và khi bị reset.

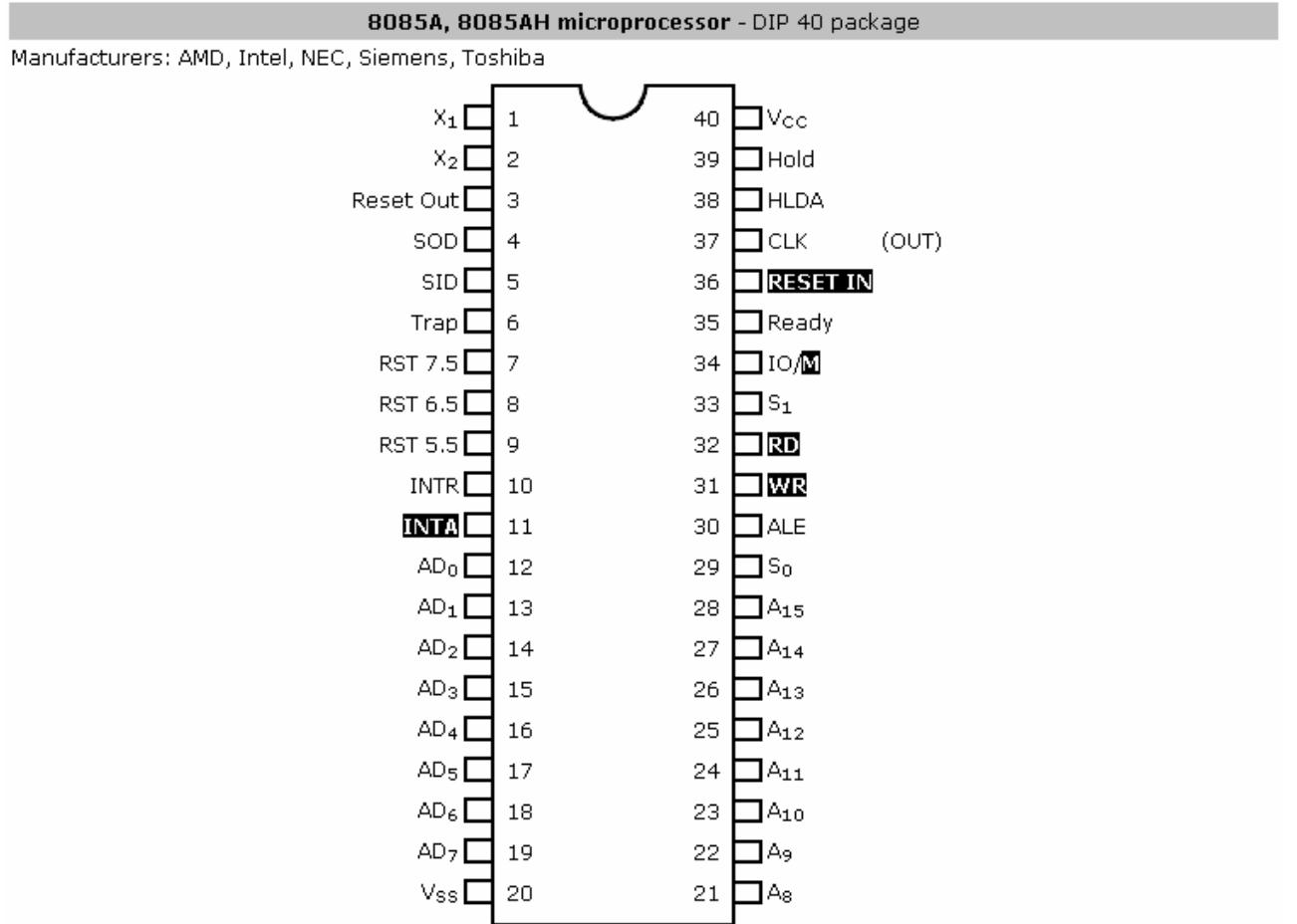
**AD0 ÷ AD7 (Address/Data)** là bus đa hợp địa chỉ và dữ liệu, tín hiệu vào/ra 3 trạng thái, bus sẽ tải địa chỉ trong khoảng thời gian xung clock thứ nhất của chu kỳ máy và sẽ tải dữ liệu ở các chu kỳ xung clock còn lại.

**ALE** (Address Latch Enable) là tín hiệu ra dùng để tạo xung chốt địa chỉ khi bus AD0 ÷ AD7 đóng vai trò là tải địa chỉ và chỉ chốt vào thời điểm cạnh xuống.

S0, S1 và **IO/M** là các tín hiệu ngõ ra cho biết các trạng thái làm việc của vi xử lý:

<b>IO/M</b>	S1	S0	Trạng thái
0	0	1	Ghi dữ liệu lên bộ nhớ
0	1	0	Đọc dữ liệu từ bộ nhớ
1	0	1	Ghi dữ liệu lên IO
1	1	0	Đọc dữ liệu từ IO
0	1	1	Đón mã lệnh từ bộ nhớ
1	1	1	Đón mã lệnh từ bộ nhớ
1	1	1	Trả lời yêu cầu ngắn
HZ	0	0	Halt
HZ	x	x	Hold
HZ	x	x	Reset

**Bảng 2-2. Các trạng thái làm việc của vi xử lý 8085A.**

**Hình 2-34. Sơ đồ chân của vi xử lý 8085A.**

S1 có thể được dùng như là trạng thái đọc ghi  $R/\overline{W}$ . Các tín hiệu  $IO/\overline{M}$ , S1, S0 sẽ bắt đầu có giá trị (hiệu lực) khi bắt đầu một chu kỳ máy và vẫn ổn định ở trạng thái này ở các chu kỳ clock còn lại. Cạnh xuống của xung ALE có thể chốt các tín hiệu này.

**$\overline{RD}$**  là tín hiệu ngõ ra, khi  $\overline{RD}$  ở mức thấp sẽ điều khiển đọc dữ liệu từ bộ nhớ hoặc IO và sẽ ở trạng thái tổng trở cao khi vi xử lý ở trạng thái HOLD, HALT và RESET.

**$\overline{WR}$**  là tín hiệu ngõ ra, khi  $\overline{WR}$  ở mức thấp sẽ điều khiển ghi dữ liệu lên bộ nhớ hoặc IO và sẽ ở trạng thái tổng trở cao khi vi xử lý ở trạng thái HOLD, HALT và RESET.

**READY** là tín hiệu ngõ vào, nếu READY ở mức cao trong khoảng thời gian đọc hoặc ghi dữ liệu giữa vi xử lý và bộ nhớ để cho biết bộ nhớ hoặc thiết bị ngoại vi IO đã sẵn sàng cho việc đọc hoặc ghi dữ liệu. Nếu tín hiệu READY ở mức thấp thì CPU phải thực hiện thêm các chu kỳ đợi chờ cho đến khi READY trở lại mức cao thì hoàn tất chu kỳ đọc hoặc ghi bộ nhớ.

**HOLD** là tín hiệu ngõ vào tích cực mức cao, thiết bị ngoại vi điều khiển chân này lên mức cao để yêu cầu vi xử lý nhường quyền sử dụng bus địa chỉ và bus dữ liệu. Khi CPU nhận được tín hiệu Hold thì sẽ ngừng sử dụng bus ngay lập tức và chuyển quyền sử dụng bus. Quá trình xử lý lệnh bên trong sẽ tiếp tục được thực hiện khi yêu cầu dùng bus của thiết bị ngoại vi không còn.

**HLDA** là tín hiệu ngõ ra dùng để báo cho thiết bị yêu cầu HOLD biết vi xử lý sẽ ngừng sử dụng bus ở chu kỳ xung clock kế. HLDA sẽ trở lại mức thấp khi thiết bị yêu cầu ngắt không còn yêu cầu nhường bus.

**INTR** là tín hiệu ngõ vào yêu cầu ngắt tổng quát. Tín hiệu này được lấy mẫu trong khoảng thời gian của chu kỳ xung clock kế cuối và trong khoảng thời gian HOLD và HALT.

Nếu INTR ở trạng thái tích cực thì vi xử lý sẽ ngừng thực hiện lệnh tiếp theo và điều khiển trả lời ngắt ở chân **INTA**. Trong khoảng chu kỳ này thì lệnh RESTART hoặc lệnh CALL có thể được chèn vào để nhảy đến chương trình con phục vụ ngắt.

INTR là tín hiệu ngắt có thể cho phép hoặc cấm bằng phần mềm.

**INTA** là tín hiệu ngõ ra trả lời đáp ứng yêu cầu ngắt, tín hiệu này được dùng thay thế cho chân **RD** trong khoảng thời gian chu kỳ lệnh sau khi tín hiệu yêu cầu ngắt INTR được chấp nhận. Tín hiệu này được dùng để tác động đến chip điều khiển ngắt 8259A hoặc các thiết bị ngắt khác.

**RST 5.5**, **RST 6.5**, **RST 7.5** là tín hiệu ngõ vào yêu cầu ngắt. Ba tín hiệu ngắt này giống như tín hiệu ngắt INTR và chỉ khác là chúng được bắt đầu tại địa chỉ hoàn toàn tự động.

**TRAP** là tín hiệu ngõ vào yêu cầu ngắt không ngăn được, tín hiệu ngắt này được lấy mẫu giống như các tín hiệu yêu cầu ngắt INTR và các ngắt RST. Yêu cầu ngắt này không thể ngăn được bằng phần mềm nên nó có mức độ ưu tiên cao nhất.

**RESET \_IN** là tín hiệu ngõ vào reset tích cực mức thấp sẽ xoá thanh ghi PC về 0000H và sẽ reset các bit cho phép ngắt và flip flop HLDA. Bus dữ liệu, bus địa chỉ và bus điều khiển đều ở trạng thái tổng trở cao khi bị reset. Các thanh ghi bên trong và thanh ghi cờ có thể bị thay đổi với những giá trị mới không xác định khi CPU bị reset. Ngõ vào reset này được nối với R và C để tạo mạch auto reset khi mở máy. Khi PC = 0000H thì chương trình được bắt đầu thực hiện tại địa chỉ 0000H tương ứng.

**RESET OUT** là tín hiệu ngõ ra để báo hiệu CPU đang bị reset. Tín hiệu này có thể dùng để reset các thiết bị khác trong hệ thống ví dụ như IC ngoại vi 8255A, 8279.

**X1, X2** là tín hiệu ngõ vào, hai ngõ vào này được nối với tụ thạch anh hoặc mạch LC hoặc mạch RC để tạo dao động cho vi xử lý hoạt động.

Mạch dao động đã tích hợp bên trong và chỉ cần nối thêm tụ thạch anh nhưng nếu có nguồn tạo xung bên ngoài thì ta có thể cung cấp xung đó đến 1 ngõ vào X1.

Tần số xung clock ngõ vào hay tần số của tụ thạch anh được chia cho 2 rồi mới cung cấp cho các hoạt động bên trong của vi xử lý.

**CLK** là tín hiệu ngõ ra xung clock dùng để cung cấp cho các thiết bị khác cần xung clock để làm việc ví dụ như IC 8279. Tần số của xung clock này bằng tần số của tụ thạch anh đang sử dụng chia cho 2.

**SID** là tín hiệu ngõ vào dữ liệu nối tiếp, dữ liệu ở ngõ vào này được nạp vào bit thứ 7 của thanh ghi A khi thực hiện RIM.

**SOD** là tín hiệu ngõ ra dữ liệu nối tiếp, dữ liệu ở ngõ ra này ở mức logic 0 hay mức logic 1 được thực hiện bởi lệnh SIM.

**V<sub>CC</sub>** là chân nguồn cung cấp +5V cho vi xử lý.

**V<sub>SS</sub>** là chân nguồn cung cấp GND cho vi xử lý.

Bảng thứ tự ưu tiên ngắt và các địa chỉ ngắt:

Tên	Thứ tự ưu tiên	Địa chỉ của các chương trình con phục vụ ngắt	Mức tác động
TRAP	1	0024H	Cạnh lén và với mức cao chờ cho đến khi được lấy mẫu
RST 7.5	2	003CH	Cạnh lén (được chốt lại)
RST 6.5	3	0034H	Mức cao cho đến khi được lấy mẫu
RST 5.5	4	002CH	Mức cao cho đến khi được lấy mẫu
INTR	5	Xem chú ý	Mức cao cho đến khi được lấy mẫu

**Bảng 2-3. Thứ tự ưu tiên ngắt của vi xử lý 8085A.**

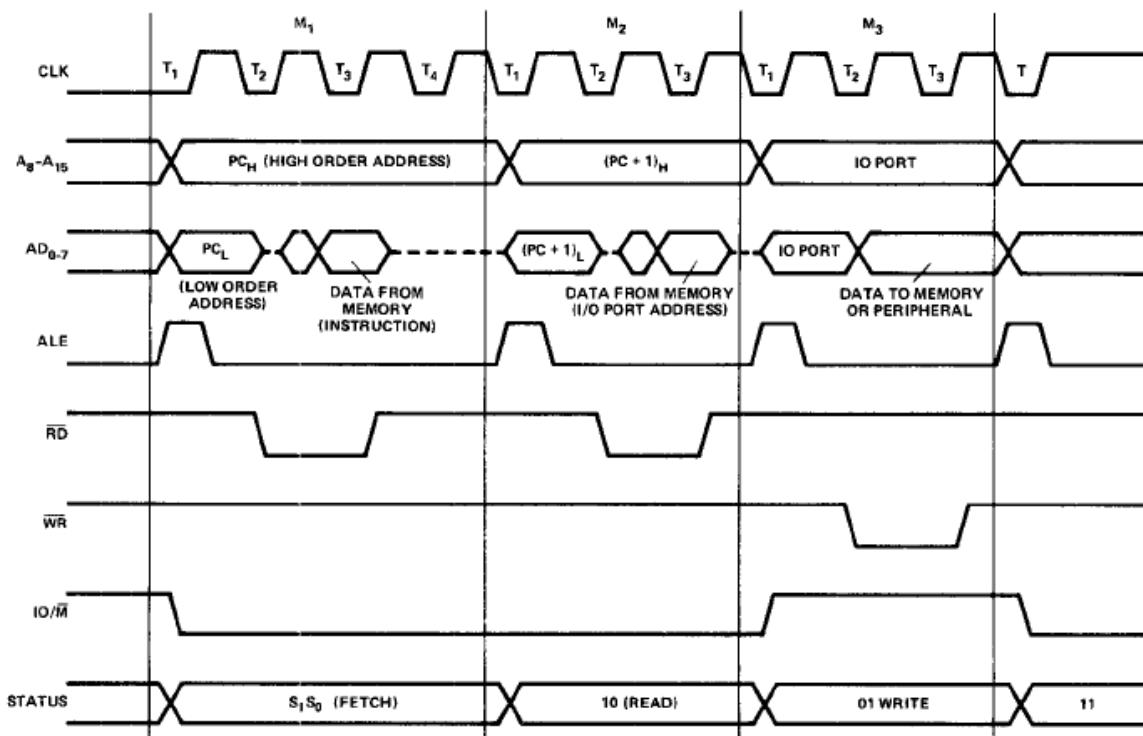
**Chú ý:** địa chỉ của chương trình con phục vụ ngắt tuỳ thuộc vào lệnh cung cấp cho CPU khi ngắt được chấp nhận.

#### 4. **GẢI ĐỒ THỜI GIAN CỦA VI XỬ LÝ 8085A:**

Vi xử lý 8085A có đa hợp bus dữ liệu và bus địa chỉ, ALE được dùng để chốt 8 bit địa chỉ thấp trên bus AD0 ÷ AD7, hình 2-35 trình bày giản đồ thời gian đón lệnh, chu kỳ đọc bộ nhớ và chu kỳ ghi dữ liệu lên bộ nhớ.

Có 7 loại chu kỳ máy khác nhau tuỳ thuộc vào lệnh đang thực hiện và 7 trạng thái khác nhau này được thể hiện thông qua trạng thái của các tín hiệu  $IO/M$ , S1, S0 và các chân điều khiển  $RD$ ,  $WR$ ,  $INTA$ .

Một chu kỳ máy thông thường gồm 3 trạng thái T ngoại trừ chu kỳ đón lệnh từ bộ nhớ thì thực hiện cần 4 hoặc 6 chu kỳ trạng thái T (chưa tính đến các tình huống đợi WAIT hoặc HOLD).



Hình 2-35. Giản đồ thời gian hoạt động của vi xử lý 8085A.

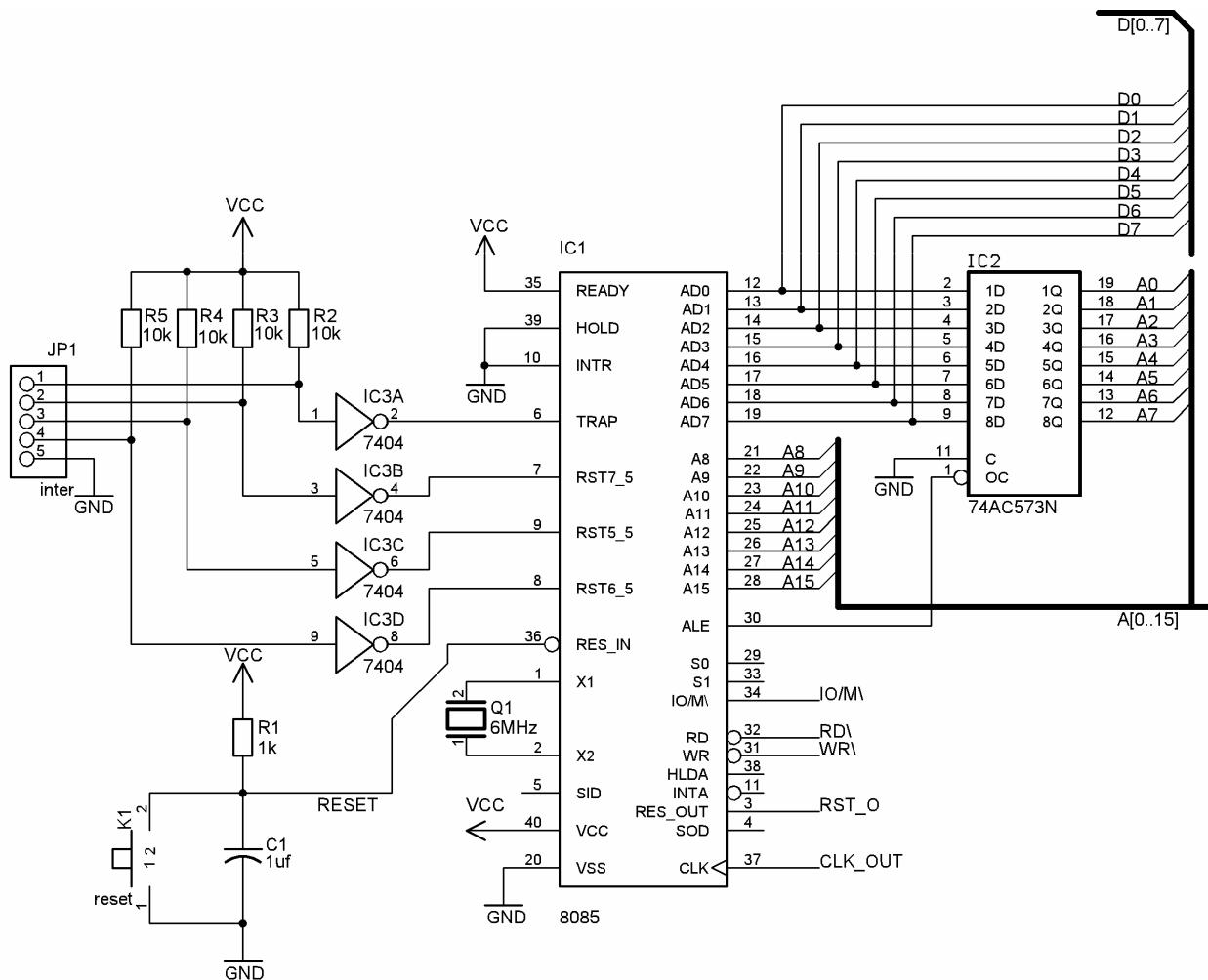
**Ở chu kỳ đón lệnh:** trong khoảng thời gian của trạng thái T1: thì CPU sẽ tạo ra địa chỉ 8 bit cao ở bus A<sub>8</sub> ÷ A<sub>15</sub>, 8 bit thấp ở bus AD<sub>0</sub> ÷ AD<sub>7</sub>, các trạng thái điều khiển S<sub>1</sub>, S<sub>0</sub> và đường điều khiển IO/M xuống mức 0. Đồng thời tạo xung chốt ALE để chốt địa chỉ và các trạng thái điều khiển. Khi sang chu kỳ trạng thái T2 thì CPU điều khiển chân RD xuống mức thấp để tác động đến bộ nhớ thực hiện việc đọc dữ liệu. Bộ nhớ ngay lập tức xuất dữ liệu tương ứng với ô nhớ đã nhận địa chỉ và dữ liệu đó được đưa lên bus AD<sub>0</sub> ÷ AD<sub>7</sub> – bây giờ bus này đóng vai trò tải dữ liệu. CPU điều khiển chân RD trở lại trạng thái không tích cực ở trạng thái T3 và tiến hành nhận dữ liệu từ bus – đó chính là mã lệnh và cất vào thanh ghi lệnh IR – chấm dứt quá trình đón lệnh từ bộ nhớ ở trạng thái T4.

Tương tự cho các chu kỳ đọc dữ liệu và ghi dữ liệu thì chỉ có 3 trạng thái T.

##### 5. GIẢI ĐA HỢP ĐỊA CHỈ VÀ DỮ LIỆU CHO VI XỬ LÝ 8085A:

Khi sử dụng vi xử lý 8085A có đa hợp bus dữ liệu và bus địa chỉ thì phải sử dụng IC chốt để tách địa chỉ và dữ liệu như hình 2-36.

Chân ALE sẽ điều khiển IC chốt 74573 để chốt 8 bit địa chỉ, ngõ ra của IC chốt là 8 bit địa chỉ thấp A<sub>0</sub> ÷ A<sub>7</sub> kết hợp với 8 bit địa chỉ cao A<sub>8</sub> ÷ A<sub>15</sub> tạo thành bus địa chỉ 16 bit đọc lặp với 8 đường dữ liệu D<sub>0</sub> ÷ D<sub>7</sub> giống như vi xử lý Z80.



**Hình 2-36. Giải đa hợp địa chỉ và dữ liệu của vi xử lý 8085A.**

Trong hệ thống không sử dụng trạng thái HOLD và ngắt INTR nên chúng được nối với GND, các ngắt TRAP, RST được sử dụng, tín hiệu ngõ vào reset được kết nối với tụ C1, điện trở R1 và nút nhấn reset, tụ thạch anh sử dụng là 6Mhz được nối với X1 và X2.

Tín hiệu READY cũng được nối với mức cao để vi xử lý không chèn thêm các chu kỳ đợi nên phải sử dụng các loại bộ nhớ có thời gian truy xuất nhanh hơn vi xử lý.

Các đường điều khiển còn lại dùng để kết nối với bộ nhớ và thiết bị ngoại vi IO.

## 6. TẬP LỆNH CỦA VI XỬ LÝ 8085A:

Tập lệnh của vi xử lý 8085 giống như tập lệnh của vi xử lý 8080 ngoại trừ 2 lệnh mới thêm vào là lệnh SIM và RIM.

Phần này sẽ giới thiệu thành phần mã lệnh, hoạt động của các bit trong thanh ghi cờ trạng thái, các kiểu định địa chỉ và cách thức sử dụng cặp thanh ghi H, L cho kiểu định địa chỉ gián tiếp.

Các kí hiệu dùng trong tập lệnh được dùng để diễn tả chức năng và mối quan hệ giữa các thành phần của lệnh. Các kí hiệu chữ thường như “r” dùng để chỉ định một thanh ghi, “rp” để xác định cặp thanh ghi, “addr” dùng để xác định địa chỉ, “data” dùng để xác định byte dữ liệu 8 bit, “data 16” dùng để xác định, “rh” và “rl” dùng để xác định các thanh ghi byte cao và thanh ghi byte thấp của một cặp thanh ghi theo thứ tự.

### Tổ chức lệnh:

Các lệnh của vi xử lý 8085 thường định dạng theo byte bao gồm các lệnh một byte, lệnh 2 byte và lệnh 3 byte. Byte đầu tiên là byte chứa các thông tin cơ bản của lệnh chính là mã lệnh, các byte thứ 2 và thứ 3 là các byte thông tin cho lệnh như dữ liệu hay địa chỉ của dữ liệu.

Trong byte mã lệnh xử lý dữ liệu trên các thanh ghi thì bit thứ “2,1,0” thường là mã của các thanh ghi nguồn và các bit thứ “5,4,3” là mã của thanh ghi đích, mã của các thanh ghi như bảng sau :

DDD or SSS	Register Name
111	A
000	B
001	C
010	D
011	E
100	H
101	L

**Bảng 2-4. Mã các thanh ghi 8 bit của vi xử lý 8085A.**

Tương tự cho các lệnh có liên quan đến cặp thanh ghi thì mã của các cặp thanh ghi như bảng sau:

RP Bits	Register Pair
00	B-C
01	D-E
10	H-L
11	SP

**Bảng 2-5. Mã các cặp thanh ghi 16 bit của vi xử lý 8085A.**

### Các kiểu định địa chỉ của vi xử lý 8085:

Vi xử lý 8085 có 4 kiểu định địa chỉ khác nhau để truy xuất dữ liệu nằm trong các thanh ghi hay dữ liệu nằm trong bộ nhớ.

**Kiểu định địa chỉ trực tiếp:** kiểu này có mã lệnh dài 3 byte. Byte thứ nhất là byte mã lệnh, byte thứ 2 và byte thứ 3 là địa chỉ của ô nhớ cần lấy dữ liệu. Byte thứ 2 là byte địa chỉ thấp và byte thứ 3 là byte địa chỉ cao.

**Kiểu định địa chỉ dùng thanh ghi:** kiểu này có mã lệnh dài 1 byte. Trong byte mã lệnh có các bit chỉ định thanh ghi chứa dữ liệu hay cặp thanh ghi chứa dữ liệu.

**Kiểu định địa chỉ gián tiếp dùng thanh ghi:** kiểu này có mã lệnh dài 1 byte. Trong byte mã lệnh có các bit chỉ định cặp thanh ghi chứa dữ liệu.

**Kiểu định địa chỉ tức thời:** kiểu này có mã lệnh dài 2 byte hoặc 3 byte. Byte thứ nhất là mã lệnh, byte thứ 2 và byte thứ 3 là dữ liệu tức thời 8 bit hay 16 bit. Nếu dữ liệu 16 bit thì byte thứ 2 là byte thấp và byte thứ 3 là byte cao.

### Các kiểu lệnh rẽ nhánh:

Việc thực hiện các lệnh bình thường Theo thứ tự từ trên xuống. Những lệnh rẽ nhánh có tác dụng thay thế tiến trình đang thực hiện. Các lệnh này chỉ định địa chỉ của lệnh kế mà vi xử lý phải thực hiện, có 2 dạng lệnh nhảy:

**Lệnh nhảy trực tiếp:** trong kiểu lệnh nhảy này chứa địa chỉ của nơi mà vi xử lý sẽ nhảy đến để thực hiện, địa chỉ của lệnh nhảy này là 2 byte, byte thứ 2 trong mã lệnh là byte địa chỉ thấp và byte thứ 3 là byte địa chỉ cao.

**Lệnh nhảy gián tiếp dùng thanh ghi:** trong kiểu lệnh nhảy này địa chỉ mà vi xử lý nhảy đến được chứa trong cặp thanh ghi. Địa chỉ phải lưu trong cặp thanh ghi trước khi thực hiện lệnh nhảy.

### Các nhóm lệnh:

Hãng Intel chia các lệnh của vi xử lý 8085 thành 5 nhóm bao gồm: nhóm lệnh truyền dữ liệu, nhóm lệnh số học, nhóm lệnh logic, nhóm lệnh nhảy và nhóm lệnh điều khiển IO.

**Nhóm lệnh truyền dữ liệu:** trong nhóm lệnh này có chức năng copy dữ liệu giữa các thanh ghi bên trong vi xử lý, giữa bộ nhớ và thanh ghi.

**Nhóm lệnh MOV:** bao gồm 3 lệnh (MOV r1,r2; MOV r,M và MOV M,r) có chức năng copy dữ liệu giữa thanh ghi và thanh ghi giữa thanh ghi và ô nhớ. Lệnh chỉ thực hiện copy 1 byte dữ liệu. Trong 3 lệnh trên thì lệnh thứ 2 và lệnh thứ 3 sử dụng kiểu định địa chỉ gián tiếp dùng thanh ghi, địa chỉ của ô nhớ lưu trong cặp thanh ghi HL.

**Nhóm lệnh MVI:** bao gồm 2 lệnh (MVI r,data và MVI M,data) có chức năng nạp một byte dữ liệu tức thời vào 1 thanh ghi hay một ô nhớ.

**Nhóm lệnh LXI:** bao gồm 1 lệnh (LXI rp,data 16) có chức năng nạp 2 byte dữ liệu tức thời vào 1 cặp thanh ghi, byte thứ 2 trong mã lệnh là byte dữ liệu thấp và byte thứ 3 trong mã lệnh là byte cao của dữ liệu.

**Nhóm lệnh LDA và STA:** là lệnh Load Accumulator Direct (LDA) và Store Accumulator Direct (STA) có chức năng copy dữ liệu giữa thanh ghi A và ô nhớ trong RAM, địa chỉ của ô nhớ được lưu trong thanh ghi HL.

**Nhóm lệnh LHLD và SHLD:** là lệnh Load H&L Direct (LHLD) và Store H&L Direct (SHLD) có chức năng copy dữ liệu giữa thanh ghi H&L với dữ liệu trong bộ nhớ RAM. Lệnh LHLD sẽ copy dữ liệu của 2 ô nhớ có địa chỉ lưu trong HL và HL+1 nạp vào thanh ghi L và thanh ghi H theo thứ tự. Lệnh SHLD thì lưu nội dung của cặp thanh ghi HL vào 2 ô nhớ có địa chỉ lưu trong HL và HL+1.

**Nhóm lệnh LDAX và STAX:** là lệnh Load Accumulator Indirect (LDAX) và Store Accumulator Indirect (STAX) có chức năng copy dữ liệu giữa thanh ghi A và ô nhớ RAM có địa chỉ gián tiếp, địa chỉ này có thể lưu trong cặp thanh ghi BC hoặc DE.

**Nhóm lệnh XCHG:** là lệnh Exchange (XCHG) cho phép trao đổi nội dung cặp thanh ghi HL với DE.

**Nhóm lệnh số học :** trong nhóm lệnh này có chức năng thực hiện các phép toán số học với dữ liệu 8 bit.

**Nhóm lệnh ADD:** là lệnh ADD Register (ADD r) và ADD memory (ADD M) có chức năng cộng nội dung thanh ghi A với thanh ghi hoặc nội dung thanh ghi A với nội dung của ô nhớ có địa chỉ lưu trong cặp thanh ghi HL. Nếu kết quả phép toán không tràn thì cờ carry bằng 0 còn nếu có tràn thì cờ carry bằng 1. Ngoài ra lệnh còn ảnh hưởng đến các bit cờ khác.

**Nhóm lệnh ADD với cờ carry:** là lệnh ADD Register with carry (ADC r) và ADD memory with carry (ADC M) có chức năng cộng nội dung thanh ghi A với thanh ghi và cờ carry hoặc nội dung thanh ghi A với nội dung của ô nhớ có địa chỉ lưu trong cặp thanh ghi HL và cờ carry. Nếu kết quả phép toán không tràn thì cờ carry bằng 0 còn nếu có tràn thì cờ carry bằng 1. Ngoài ra lệnh còn ảnh hưởng đến các bit cờ khác.

**Nhóm lệnh ADD tức thời:** là lệnh ADD Immediate (ADI data) và ADD Immediate memory with carry (ACI data) có chức năng cộng nội dung thanh ghi A với dữ liệu tức thời 8 bit và cộng nội dung thanh ghi A với dữ liệu tức thời 8 bit và cờ carry. Nếu kết quả phép toán không tràn thì cờ carry bằng 0 còn nếu có tràn thì cờ carry bằng 1. Ngoài ra lệnh còn ảnh hưởng đến các bit cờ khác.

**Nhóm lệnh SUBTRACT:** là lệnh Subtract Register (SUB r) và Subtract memory (SUB M) có chức năng trừ nội dung thanh ghi A với thanh ghi hoặc nội dung thanh ghi A với nội dung của ô nhớ có địa chỉ lưu trong cặp thanh ghi HL. Nếu kết quả phép toán không mượn thì cờ carry bằng 0 còn nếu có mượn thì cờ carry bằng 1. Ngoài ra lệnh còn ảnh hưởng đến các bit cờ khác.

**Nhóm lệnh Subtract với cờ carry:** là lệnh Subtract Register with borrow (SBB r) và Subtract memory with borrow (SBB M) có chức năng trừ nội dung thanh ghi A với thanh ghi và cờ carry hoặc nội dung thanh ghi A với nội dung của ô nhớ có địa chỉ lưu trong cặp thanh ghi HL và cờ carry. Nếu kết quả phép toán không mượn thì cờ carry bằng 0 còn nếu có mượn thì cờ carry bằng 1. Ngoài ra lệnh còn ảnh hưởng đến các bit cờ khác.

**Nhóm lệnh Subtract tức thời:** là lệnh Subtract immediate (SUI data) và Subtract immediate with borrow (SBI data) có chức năng trừ nội dung thanh ghi A với dữ liệu tức thời 8 bit hoặc nội dung thanh ghi A với dữ liệu tức thời 8 bit và cờ carry. Nếu kết quả phép toán không mượn thì cờ carry bằng 0 còn nếu có mượn thì cờ carry bằng 1. Ngoài ra lệnh còn ảnh hưởng đến các bit cờ khác.

**Nhóm lệnh tăng Increment :** là lệnh increment register (INR r) và Increment memory (INR M) có chức năng tăng nội dung thanh ghi lên 1 hoặc tăng nội dung ô nhớ có địa chỉ lưu trong cặp thanh ghi HL lên 1. Lệnh không ảnh hưởng đến thanh ghi trạng thái.

**Nhóm lệnh giảm Decrement :** là lệnh Decrement register (DCR r) và Decrement memory (DCR M) có chức năng giảm nội dung thanh ghi xuống 1 hoặc giảm nội dung ô nhớ có địa chỉ lưu trong cặp thanh ghi HL xuống 1. Lệnh không ảnh hưởng đến thanh ghi trạng thái.

**Nhóm lệnh xử lý cặp thanh ghi :** là lệnh Increment Register Pair (INR rp) và Decrement Register Pair (DCR rp) có chức năng tăng nội dung cặp thanh ghi lên 1 hoặc giảm nội dung cặp thanh ghi xuống 1.

**Nhóm lệnh cộng cặp thanh ghi**: là lệnh Add Register Pair (DAD rp) có chức năng cộng nội dung cặp thanh ghi BC hoặc DE với cặp thanh ghi HL, kết quả lưu vào cặp thanh ghi HL. Lệnh sẽ ảnh hưởng đến thanh ghi trạng thái.

**Lệnh hiệu chỉnh thập phân**: là lệnh Decimal Adjust Accumulator (DAA) có chức năng hiệu chỉnh giá trị 8 bit trong thanh ghi A thành số BCD sau khi thực hiện phép toán cộng 2 số BCD.

**Nhóm lệnh logic**: trong nhóm lệnh này có chức năng thực hiện các phép toán logic với dữ liệu 8 bit.

**Nhóm lệnh AND**: là lệnh AND Register (ANA r) và AND memory (ANA M) có chức năng AND nội dung thanh ghi A với thanh ghi hoặc nội dung thanh ghi A với nội dung của ô nhớ có địa chỉ lưu trong cặp thanh ghi HL. Lệnh And sẽ thực hiện phép toán and từng cặp bit với nhau theo thứ tự. Kết quả sau khi thực hiện được lưu vào thanh ghi A và thanh ghi trạng thái bị ảnh hưởng và cờ carry thì luôn bằng 0.

**Nhóm lệnh AND Immediate**: là lệnh AND Immediate (ANI data) có chức năng AND nội dung thanh ghi A với dữ liệu tức thời 8 bit. Kết quả sau khi thực hiện được lưu vào thanh ghi A và thanh ghi trạng thái bị ảnh hưởng và cờ carry thì luôn bằng 0.

**Nhóm lệnh OR**: là lệnh Or Register (ORA r) và Or memory (ORA M) có chức năng OR nội dung thanh ghi A với thanh ghi hoặc nội dung thanh ghi A với nội dung của ô nhớ có địa chỉ lưu trong cặp thanh ghi HL. Kết quả sau khi thực hiện được lưu vào thanh ghi A và thanh ghi trạng thái bị ảnh hưởng và cờ carry thì luôn bằng 0.

**Nhóm lệnh OR Immediate**: là lệnh OR Immediate (ORI data) có chức năng OR nội dung thanh ghi A với dữ liệu tức thời 8 bit. Kết quả sau khi thực hiện được lưu vào thanh ghi A và thanh ghi trạng thái bị ảnh hưởng và cờ carry thì luôn bằng 0.

**Nhóm lệnh Exclusive Or**: là lệnh Exclusive Or Register (XRA r) và Exclusive Or memory (XRA M) có chức năng Exclusive nội dung thanh ghi A với thanh ghi hoặc nội dung thanh ghi A với nội dung của ô nhớ có địa chỉ lưu trong cặp thanh ghi HL. Kết quả sau khi thực hiện được lưu vào thanh ghi A và thanh ghi trạng thái bị ảnh hưởng và cờ carry thì luôn bằng 0.

**Nhóm lệnh Exclusive Immediate**: là lệnh Exclusive Or Immediate (XRI data) có chức năng Exclusive Or nội dung thanh ghi A với dữ liệu tức thời 8 bit. Kết quả sau khi thực hiện được lưu vào thanh ghi A và thanh ghi trạng thái bị ảnh hưởng và cờ carry thì luôn bằng 0.

**Nhóm lệnh so sánh**: là lệnh Compare Register (CMP r) và Compare memory (CMP M) có chức năng so sánh nội dung thanh ghi A với thanh ghi hoặc nội dung thanh ghi A với nội dung của ô nhớ có địa chỉ lưu trong cặp thanh ghi HL. Kết quả sau khi thực hiện ảnh hưởng đến thanh ghi trạng thái.

**Nhóm lệnh so sánh tức thời**: là lệnh Compare Immediate (CPI data) có chức năng so sánh nội dung thanh ghi A với dữ liệu tức thời 8 bit. Kết quả sau khi thực hiện ảnh hưởng đến thanh ghi trạng thái.

**Nhóm lệnh xoay**: là lệnh Rotate Left (RLC) và Rotate Right (RRC) có chức năng xoay nội dung thanh ghi A sang trái hoặc sang phải 1 bit. Chỉ có cờ carry bị ảnh hưởng.

**Nhóm lệnh xoay thông qua cờ carry**: là lệnh Rotate Left through carry (RAL) và Rotate Right through carry (RAR) có chức năng xoay nội dung thanh ghi A và cờ carry sang trái hoặc sang phải 1 bit. Chỉ có cờ carry bị ảnh hưởng.

**Nhóm lệnh bù nội dung thanh ghi A**: là lệnh Complement Accumulator (CMA) có chức năng thực hiện phép toán bù 1 nội dung thanh ghi A.

**Nhóm lệnh bù và clear cờ carry**: là lệnh Complement carry (CMC) và Set Carry (STC) có chức năng nghịch đảo cờ Carry và làm cờ carry bằng 1.

**Nhóm lệnh rẽ nhánh**: trong nhóm lệnh này gồm có lệnh nhảy không điều kiện và lệnh nhảy có điều kiện, lệnh gọi chương trình con không điều kiện và lệnh gọi chương trình con có điều kiện.

**Nhóm lệnh nhảy**: là lệnh Jump (JMP addr) và Jump Conditional (Jxx Addr) có chức năng thay đổi nội dung của thanh ghi PC để nhảy đến địa chỉ Addr để tiếp tục thực hiện chương trình tại đó. Đối với lệnh nhảy có điều khiển thì sẽ nhảy khi thoả điều kiện và sẽ không nhảy khi không thoả điều kiện. Các lệnh nhảy có điều kiện là căn cứ vào các bit trạng thái được liệt kê ở bảng sau:

Mnemonic	Condition	CCC Bits
NZ	Not Zero ( $Z=0$ )	000
Z	Zero ( $Z=1$ )	001
NC	Not Carry ( $C=0$ )	010
C	Carry ( $C=1$ )	011
PO	Parity Odd ( $P=0$ )	100
PE	Parity Even ( $P=1$ )	101
P	Plus ( $S=0$ )	110
M	Minus ( $S=1$ )	111

Bảng 2-6. Các bit trong thanh ghi trạng thái của vi xử lý 8085A.

**Nhóm lệnh gọi chương trình con**: là lệnh Call (Call addr) và Call Conditional (Cxx Addr) có chức năng thực hiện chương trình con tại địa chỉ Addr và lệnh gọi có điều kiện thì chương trình con chỉ được thực hiện khi thoả điều kiện, nếu không thoả thì không thực hiện. Các lệnh gọi chương trình con có điều kiện sẽ căn cứ vào các bit cờ giống như lệnh nhảy có điều kiện. Khi thực hiện các lệnh gọi chương trình con thì nội dung của thanh ghi PC được cất vào ngăn xếp rồi mới nạp địa chỉ của chương trình con vào thanh ghi PC, sau khi thực hiện xong chương trình con thì nội dung cất trong ngăn xếp được trả lại cho PC để tiếp tục thực hiện lệnh tiếp theo trong chương trình chính, việc cất địa chỉ trong PC và lấy lại tương ứng với 2 lệnh gọi Call và lệnh trả về RET.

**Nhóm lệnh kết thúc chương trình con**: là lệnh Return (RET) và Return Conditional (Rxx) có chức năng kết thúc chương trình con để trở về chương trình chính không điều kiện và có điều khiển. Trong chương trình con phải luôn có lệnh kết thúc RET. Không được nhảy từ chương trình con về chương trình chính và ngược lại.

**Nhóm lệnh bắt đầu lại**: là lệnh Restart (RST n) có chức năng gọi một trong 8 chương trình con tại địa chỉ do nhà chế tạo vi xử lý qui định giống như các chương trình con phục vụ ngắn được liệt kê ở bảng sau:

NAME:	ADDRESS:
RST 0	00H
RST 1	08H
RST 2	10H
RST 3	18H
RST 4	20H
<b>TRAP</b>	<b>24H</b>
RST 5	28H
<b>REST 5.5</b>	<b>2CH</b>
RST 6	30H
<b>REST 6.5</b>	<b>34H</b>
RST 7	38H
<b>REST 7.5</b>	<b>3CH</b>

Bảng 2-7. Bảng vector địa chỉ ngắn của vi xử lý 8085A.

Cách thức thực hiện cũng giống như lệnh gọi Call.

**Nhóm lệnh nhảy gián tiếp**: là lệnh Jump H&L Indirect (PCHL) có chức năng thay đổi nội dung của thanh ghi PC bằng giá trị trong cặp thanh ghi HL và bắt đầu thực hiện chương trình tại địa chỉ mới vừa thay đổi.

Nhóm lệnh điều khiển máy :

**Nhóm lệnh cất và lấy dữ liệu**: là lệnh Push Register Pair (PUSH rp) và Pop Register Pair (POP rp) có chức năng cất tạm nội dung cặp thanh ghi vào bộ nhớ ngăn xếp và lấy lại nội dung đã cất tạm trong ngăn xếp.

**Nhóm lệnh cất và lấy nội dung thanh ghi trạng thái**: là lệnh Push Processor Status Word (PUSH PSW) và Pop Processor Status Word (POP PSW) có chức năng cất tạm nội dung cặp thanh ghi A và thanh ghi trạng thái vào bộ nhớ ngăn xếp và lấy lại từ ngăn xếp giống như các cặp thanh ghi khác.

**Nhóm lệnh trao đổi nội dung thanh ghi với bộ nhớ ngắn xép** : là lệnh Exchange Stack Top (XTHL) sẽ thực hiện trao đổi nội dung của cặp thanh ghi HL với nội dung của 2 ô nhớ ngắn xép.

**Nhóm lệnh copy nội dung cặp thanh ghi** : là lệnh Move HL Register to SP (SPHL) sẽ thực hiện trao nội dung của cặp thanh ghi HL cho thi SP.

**Nhóm lệnh IO :**

**Nhóm lệnh nhập và xuất dữ liệu ra port** : là lệnh Input (IN port) và Output (OUT port) có chức năng nhập và xuất dữ liệu giữa thanh ghi A trong vi xử lý với các thiết bị ngoại vi IO. Nội dung cần gửi ra port phải lưu trong thanh ghi A và dữ liệu đọc từ port về cũng lưu trong thanh ghi A.

**Nhóm lệnh cho phép ngắt và cấm ngắt** : là lệnh Enable Interrupt (EI) và Disenable Interrupt (DI) có chức năng cho phép và cấm ngắt đối với các ngắt có thể ngăn được.

**Nhóm lệnh ngừng và lệnh Nop** : là lệnh Halt (HLT) và No - operation (NOP) có chức năng ngừng thực hiện chương trình và lệnh Nop thì không làm gì cả để dành vùng nhớ sử dụng về sau hoặc delay một vài  $\mu$ s.

**Tóm tắt tập lệnh vi xử lý 8085 :**

Mnemonic	Op	SZAPC	~s	Description	Notes
ACI n	CE *****	7		Add with Carry Immediate	A=A+n+CY
ADC r	8F *****	4		Add with Carry	A=A+r+CY(21X)
ADC M	8E *****	7		Add with Carry to Memory	A=A+[ HL ]+CY
ADD r	87 *****	4		Add	A=A+r (20X)
ADD M	86 *****	7		Add to Memory	A=A+[ HL ]
ADI n	C6 *****	7		Add Immediate	A=A+n
ANA r	A7 ****0	4		AND Accumulator	A=A&r (24X)
ANA M	A6 ****0	7		AND Accumulator and Memory	A=A&[ HL ]
ANI n	E6 **0*0	7		AND Immediate	A=A&n
CALL a	CD -----	18		Call unconditional	- [ SP ]=PC, PC=a
CC a	DC -----	9		Call on Carry	If CY=1(18~s)
CM a	FC -----	9		Call on Minus	If S=1 (18~s)
CMA	2F -----	4		Complement Accumulator	A=~A
CMC	3F -----*	4		Complement Carry	CY=~CY
CMP r	BF *****	4		Compare	A-r (27X)
CMP M	BF *****	7		Compare with Memory	A-[ HL ]
CNC a	D4 -----	9		Call on No Carry	If CY=0(18~s)
CNZ a	C4 -----	9		Call on No Zero	If Z=0 (18~s)
CP a	F4 -----	9		Call on Plus	If S=0 (18~s)
CPE a	EC -----	9		Call on Parity Even	If P=1 (18~s)
CPI n	FE *****	7		Compare Immediate	A-n
CPO a	E4 -----	9		Call on Parity Odd	If P=0 (18~s)
CZ a	CC -----	9		Call on Zero	If Z=1 (18~s)
DAA	27 *****	4		Decimal Adjust Accumulator	A=BCD format
DAD B	09 -----*	10		Double Add BC to HL	HL=HL+BC
DAD D	19 -----*	10		Double Add DE to HL	HL=HL+DE
DAD H	29 -----*	10		Double Add HL to HL	HL=HL+HL
DAD SP	39 -----*	10		Double Add SP to HL	HL=HL+SP
DCR r	3D *****	4		Decrement	r=r-1 (0X5)
DCR M	35 *****	10		Decrement Memory	[ HL ]=[ HL ]-1
DCX B	0B -----	6		Decrement BC	BC=BC-1
DCX D	1B -----	6		Decrement DE	DE=DE-1
DCX H	2B -----	6		Decrement HL	HL=HL-1
DCX SP	3B -----	6		Decrement Stack Pointer	SP=SP-1

DI	F3	----	4	Disable Interrupts		
EI	FB	----	4	Enable Interrupts		
HLT	76	----	5	Halt		
IN p	DB	-----	10	Input	A=[p]	
INR r	3C	****	4	Increment	r=r+1 (0X4)	
INR M	3C	****	10	Increment Memory	[HL]=[HL]+1	
INX B	03	-----	6	Increment BC	BC=BC+1	
INX D	13	-----	6	Increment DE	DE=DE+1	
INX H	23	-----	6	Increment HL	HL=HL+1	
INX SP	33	-----	6	Increment Stack Pointer	SP=SP+1	
JMP a	C3	-----	7	Jump unconditional	PC=a	
JC a	DA	-----	7	Jump on Carry	If CY=1(10~s)	
JM a	FA	-----	7	Jump on Minus	If S=1 (10~s)	
JNC a	D2	-----	7	Jump on No Carry	If CY=0(10~s)	
JNZ a	C2	-----	7	Jump on No Zero	If Z=0 (10~s)	
JP a	F2	-----	7	Jump on Plus	If S=0 (10~s)	
JPE a	EA	-----	7	Jump on Parity Even	If P=1 (10~s)	
JPO a	E2	-----	7	Jump on Parity Odd	If P=0 (10~s)	
JZ a	CA	-----	7	Jump on Zero	If Z=1 (10~s)	
LDA a	3A	-----	13	Load Accumulator direct	A=[a]	
LDAX B	0A	-----	7	Load Accumulator indirect	A=[BC]	
LDAX D	1A	-----	7	Load Accumulator indirect	A=[DE]	
LHLD a	2A	-----	16	Load HL Direct	HL=[a]	
LXI B,nn	01	-----	10	Load Immediate BC	BC=nn	
LXI D,nn	11	-----	10	Load Immediate DE	DE=nn	
LXI H,nn	21	-----	10	Load Immediate HL	HL=nn	
LXI SP,nn	31	-----	10	Load Immediate Stack Ptr	SP=nn	
MOV r1,r2	7F	-----	4	Move register to register	r1=r2 (1XX)	
MOV M,r	77	-----	7	Move register to Memory	[HL]=r (16X)	
MOV r,M	7E	-----	7	Move Memory to register	r=[HL] (1X6)	
MVI r,n	3E	-----	7	Move Immediate	r=n (0X6)	
MVI M,n	36	-----	10	Move Immediate to Memory	[HL]=n	
NOP	00	-----	4	No Operation		
ORA r	B7	**0*0	4	Inclusive OR Accumulator	A=Avr (26X)	
ORA M	B6	**0*0	7	Inclusive OR Accumulator	A=Av[HL]	
ORI n	F6	**0*0	7	Inclusive OR Immediate	A=Avn	
OUT p	D3	-----	10	Output	[p]=A	
PCHL	E9	-----	6	Jump HL indirect	PC=[HL]	
POP B	C1	-----	10	Pop BC	BC=[SP]+	
POP D	D1	-----	10	Pop DE	DE=[SP]+	
POP H	E1	-----	10	Pop HL	HL=[SP]+	
POP PSW	F1	-----	10	Pop Processor Status Word	{PSW,A}=[SP]+	

Mnemonic	Op	SZAPC	~s	Description	Notes
PUSH B	C5	----	12	Push BC	-[SP]=BC
PUSH D	D5	----	12	Push DE	-[SP]=DE
PUSH H	E5	----	12	Push HL	-[SP]=HL
PUSH PSW	F5	-----	12	Push Processor Status Word	-[SP]={PSW,A}
RAL	17	----	*	Rotate Accumulator Left	A={CY,A}<-
RAR	1F	----	*	Rotate Accumulator Right	A=->{CY,A}
RET	C9	-----	10	Return	PC=[SP]+
RC	D8	-----	6	Return on Carry	If CY=1(12~s)
RIM	20	-----	4	Read Interrupt Mask	A=mask
RM	F8	-----	6	Return on Minus	If S=1 (12~s)
RNC	D0	-----	6	Return on No Carry	If CY=0(12~s)
RNZ	C0	-----	6	Return on No Zero	If Z=0 (12~s)
RP	F0	-----	6	Return on Plus	If S=0 (12~s)
RPE	E8	-----	6	Return on Parity Even	If P=1 (12~s)
RPO	E0	-----	6	Return on Parity Odd	If P=0 (12~s)
RZ	C8	-----	6	Return on Zero	If Z=1 (12~s)

RLC	07	----	*	4	Rotate Left Circular	A=A<-
RRC	0F	----	*	4	Rotate Right Circular	A=>A
RST z	C7	-----		12	Restart (3X7)	-[ SP ]=PC, PC=z
SBB r	9F	*****		4	Subtract with Borrow	A=A-r-CY
SBB M	9E	*****		7	Subtract with Borrow	A=A-[ HL ]-CY
SBI n	DE	*****		7	Subtract with Borrow Immed	A=A-n-CY
SHLD a	22	-----		16	Store HL Direct	[ a ]=HL
SIM	30	-----		4	Set Interrupt Mask	mask=A
SPHL	F9	-----		6	Move HL to SP	SP=HL
STA a	32	-----		13	Store Accumulator	[ a ]=A
STAX B	02	-----		7	Store Accumulator indirect	[ BC ]=A
STAX D	12	-----		7	Store Accumulator indirect	[ DE ]=A
STC	37	----1		4	Set Carry	CY=1
SUB r	97	*****		4	Subtract	A=A-r (22X)
SUB M	96	*****		7	Subtract Memory	A=A-[ HL ]
SUI n	D6	*****		7	Subtract Immediate	A=A-n
XCHG	EB	-----		4	Exchange HL with DE	HL<->DE
XRA r	AF	**0*0		4	Exclusive OR Accumulator	A=Axr (25X)
XRA M	AE	**0*0		7	Exclusive OR Accumulator	A=Ax[ HL ]
XRI n	EE	**0*0		7	Exclusive OR Immediate	A=Axn
XTHL	E3	-----		16	Exchange stack Top with HL	[ SP ]<->HL
<hr/>						
PSW	-*01				Flag unaffected/affected/reset/set	
S	S				Sign (Bit 7)	
Z	Z				Zero (Bit 6)	
AC	A				Auxiliary Carry (Bit 4)	
P	P				Parity (Bit 2)	
CY	C				Carry (Bit 0)	
<hr/>						
a p					Direct addressing	
M z					Register indirect addressing	
n nn					Immediate addressing	
r					Register addressing	
<hr/>						
DB n(,n)					Define Byte(s)	
DB 'string'					Define Byte ASCII character string	
DS nn					Define Storage Block	
DW nn(,nn)					Define Word(s)	
<hr/>						
A B C D E H L					Registers (8-bit)	
BC DE HL					Register pairs (16-bit)	
PC					Program Counter register (16-bit)	
PSW					Processor Status Word (8-bit)	
SP					Stack Pointer register (16-bit)	
<hr/>						
a nn					16-bit address/data (0 to 65535)	
n p					8-bit data/port (0 to 255)	
r					Register (X=B,C,D,E,H,L,M,A)	
z					Vector (X=0H,8H,10H,18H,20H,28H,30H,38H)	
<hr/>						
+ -					Arithmetic addition/subtraction	
& ~					Logical AND/NOT	
v x					Logical inclusive/exclusive OR	
<- ->					Rotate left/right	
<->					Exchange	
[ ]					Indirect addressing	
[ ]+ - [ ]					Indirect address auto-inc/decrement	
{ }					Combination operands	
( X )					Octal op code where X is a 3-bit code	
If (~s)					Number of cycles if condition true	

**Bảng 2-8. Tóm tắt tập lệnh của vi xử lý 8085A.****VI. TÓM TẮT – CÂU HỎI ÔN TẬP – BÀI TẬP:****1 TÓM TẮT:**

1. Một vi xử lý có 3 phần chính: khối ALU, các thanh ghi, và khối control logic.
2. Khối ALU có 2 ngõ vào và một ngõ ra. Một trong 2 ngõ vào nhận dữ liệu từ data bus, ngõ vào còn lại sẽ nhận dữ liệu từ thanh ghi A. Khối ALU có thể xử lý 1 hoặc 2 dữ liệu, chức năng chính của khối ALU là thực hiện các phép toán số học, các phép toán logic và kiểm tra dữ liệu.
3. Thanh ghi trong vi xử lý có thể lưu trữ tạm thời các dữ liệu, các thanh ghi thông dụng, các thanh ghi có chức năng đặc biệt.
4. Tất cả các vi xử lý đều có các thanh ghi cơ bản:
  - ◆ Accumulator.
  - ◆ Program counter.
  - ◆ Stack pointer.
  - ◆ General purpose registers.
  - ◆ Memory address register and logic.
  - ◆ Instruction register.
  - ◆ Temporary register.
5. Các thanh ghi rất cần thiết cho vi xử lý làm việc. Tuy nhiên người lập trình không thể sử dụng hết tất cả các thanh ghi này.
6. Thanh ghi Accumulator luôn làm việc với khối ALU, Accumulator là một thanh ghi quan trọng của vi xử lý trong xử lý dữ liệu. Chiều dài từ dữ liệu của thanh ghi Accumulator bằng với chiều dài từ dữ liệu của vi xử lý.
7. Thanh ghi Program counter có chức năng tạo địa chỉ để đón lệnh khi vi xử lý thực hiện chương trình, khi vi xử lý đón lệnh xong và thực hiện lệnh thì nội dung của PC tăng lên để chuẩn bị đón lệnh tiếp theo. Thanh ghi PC phải có chiều dài từ dữ liệu hay số bit có khả năng truy xuất hết bộ nhớ.
8. Một chương trình có thể bắt đầu tại bất kỳ vị trí nào trong bộ nhớ và có thể kết thúc tại bất kỳ vị trí nào trong bộ nhớ. Tuy nhiên các lệnh trong chương trình phải theo đúng một trình tự hợp lý.
9. Khi vi xử lý bắt đầu thực hiện chương trình:
  - ◆ Từng lệnh trong chương trình sẽ được thực hiện theo một trình tự nối tiếp trừ khi có lệnh đặc biệt làm thay đổi trình tự này.
  - ◆ Khi PC chỉ đến một ô nhớ thì khối control logic sẽ đón lệnh từ ô nhớ này.
  - ◆ Mỗi khi lệnh được đón, vi xử lý sẽ tăng nội dung của PC để chuẩn bị cho lệnh kế và bắt đầu thực hiện lệnh vừa đón.

- ◆ Thanh ghi địa chỉ bộ nhớ chỉ đến mỗi ô nhớ để truy xuất dữ liệu khi vi xử lý yêu cầu.
  - ◆ Dữ liệu trong thanh ghi địa chỉ bộ nhớ sẽ được gởi ra bus địa chỉ bên ngoài để kết nối với bộ nhớ. Thanh ghi địa chỉ bộ nhớ phải có đủ số bit để có thể truy xuất hết tất cả các ô nhớ mà vi xử lý có thể.
10. Thanh ghi trạng thái sẽ lưu trữ lại các kết quả của một số lệnh, một thanh ghi trạng thái thường có các bit sau: bit zero, bit negative, bit carry, bit half carry, bit parity, bit overflow, bit interrupt ... các bit trạng thái dùng để xác định trạng thái của một số lựa chọn trong chương trình.
11. Thanh ghi SP chỉ đến một ô nhớ dùng để lưu trữ dữ liệu tạm thời. Mỗi khi ngăn xếp dùng để lưu trữ dữ liệu thì giá trị trong SP sẽ giảm để chuẩn bị cho việc lưu trữ dữ liệu tiếp theo.
12. Thanh ghi lệnh lưu trữ lệnh dạng số nhị phân để ra lệnh cho khối control logic thực hiện các công việc mà lệnh yêu cầu.
13. Khi lệnh được đón từ bộ nhớ có nghĩa là thực hiện một quá trình copy dữ liệu trong ô nhớ của chương trình vào thanh ghi lệnh.
14. Trong quá trình thực hiện lệnh khối control logic và khối giải mã lệnh sẽ đọc lệnh trong thanh ghi lệnh.
15. Thanh ghi tạm thời dùng để lưu trữ dữ liệu cho ALU xử lý.
16. Khối giải mã lệnh thực hiện công việc giải mã lệnh để xem lệnh yêu cầu thực hiện công việc gì, sau đó khối control logic sẽ thực hiện đúng công việc đó.
17. Các khối trong vi xử lý được kết nối với nhau thông qua bus dữ liệu bên trong. Quá trình kết nối để trao đổi dữ liệu được khối control logic quyết định, sự quyết định này tùy thuộc vào lệnh. Bus dữ liệu luôn là bus 2 chiều.
18. Một lệnh của vi xử lý là một từ dạng số nhị phân, dùng để khối giải mã và khối control logic thực hiện một công việc nhất định.
19. Tập lệnh của vi xử lý là tất cả các lệnh mà vi xử lý có thể hiểu và thực hiện được.
20. Chiều dài của 1 lệnh (số lượng các bit của 1 lệnh) bằng với chiều dài từ dữ liệu
21. Lệnh của vi xử lý được giải mã và thực hiện khi lệnh được nạp thanh ghi lệnh bên trong vi xử lý ở chu kỳ đón lệnh. Ở chu kỳ thực hiện lệnh khối giải mã và khối control logic sẽ thực hiện các yêu cầu của lệnh.
22. Một lệnh của vi xử lý bao gồm 2 phần hay 2 thông tin. Thông tin thứ nhất để báo cho vi xử lý biết làm công việc gì. Thông tin thứ 2 báo cho vi xử lý địa chỉ của dữ liệu. Thông tin thứ nhất thường gọi là mã lệnh hay mã công tác. Thông tin thứ 2 gọi là địa chỉ hay địa chỉ công tác (có nghĩa là lệnh xảy ra đối với dữ liệu tại địa chỉ đó).
23. Có nhóm lệnh cơ bản và có rất nhiều mã lệnh cho 1 nhóm lệnh.
24. Lệnh của vi xử lý là một số nhị phân gồm cả thông tin và địa chỉ được gọi là mã máy. Để dễ nhớ lệnh mã máy đã được chuyển sang mã gọi nhớ, từ gọi nhớ có ý nghĩa gần với chức năng của lệnh. Tập hợp các từ gọi nhớ gọi là ngôn ngữ

Assembly. Khi viết chương trình bằng Assembly, để máy thực hiện chương trình này thì phải có chương trình dịch các lệnh viết bằng Assembly sang mã máy để vi xử lý thực hiện, chương trình dịch được gọi là Assembler.

## 2. CÂU HỎI ÔN TẬP VÀ BÀI TẬP TRẮC NGHIỆM:

### 1. Sơ đồ khối của vi xử lý dùng để

- A. Diễn tả chi tiết các cổng logic và các Flip Flop được dùng để thiết kế nên vi xử lý.
- B. Diễn tả các mạch logic của vi xử lý kết nối với các thiết bị IO và bộ nhớ bên ngoài.
- C. Dùng để trình bày các khối logic có chức năng xử lý dữ liệu để giải quyết một vấn đề.
- D. Cả 3 câu trên đều đúng.

### 2. Trong các câu sau câu nào không phải là chức năng của ALU:

- A. Add
- B. Shift
- C. Complement
- D. Lưu trữ dữ liệu.

### 3. ALU có 2 ngõ vào, 2 ngõ vào này được kết nối với:

- A. Program Counter.
- B. Bus dữ liệu bên trong.
- C. Control logic.
- D. Thanh ghi địa chỉ bộ nhớ.

### 4. Chức năng chính của khối ALU:

- A. Thực hiện phép cộng.
- B. Đóng vai trò xuất dữ liệu giống như thanh ghi Accumulator.
- C. Thực hiện các phép toán logic và số học để xử lý dữ liệu.
- D. Tất cả 3 câu trên đều đúng.

### 5. Hầu hết các phép toán logic và số học trong vi xử lý thực hiện giữa nội dung của một ô nhớ hoặc nội dung của một thanh ghi với:

- A. Nội dung của thanh ghi Accumulator.
- B. Nội dung thanh ghi Program Counter.
- C. Nội dung thanh ghi địa chỉ.
- D. Thanh ghi lệnh.

### 6. Một vi xử lý 16 bit có thể truy xuất $2^{20} = 1.048.567$ ô nhớ có thể cho biết thanh ghi PC của Microprocessor này có chiều dài từ dữ liệu bao nhiêu bit:

- A. 4
- B. 8
- C. 16
- D. 20
- E. 32

### 7. Thanh ghi Program counter của vi xử lý là một trong những thanh ghi:

- A. Đặc biệt.
- B. Thông dụng
- C. Memory.
- D. Tất cả 3 câu trên.

### 8. Khi vi xử lý đang thực hiện lệnh, thanh ghi PC đang chỉ đến:

- A. Lệnh vừa thực hiện.
- B. Lệnh đang thực hiện.
- C. Lệnh tiếp theo.
- D. Cả 3 câu trên đều sai.

9. Hãy thực hiện các phép cộng các số nhị phân 8 bit. Sau khi cộng xong các con số này, hãy xác định ảnh hưởng của phép cộng đến các bit Zero (Z), bit Negative (N), bit Carry (C).

$$\begin{array}{r} 0000\ 1111 \\ + \underline{1111\ 0000} \\ \hline \end{array} \quad \begin{array}{r} 0011\ 1011 \\ + \underline{1100\ 0101} \\ \hline \end{array}$$

$$\begin{array}{r} 1110\ 1111 \\ + \underline{1111\ 1111} \\ \hline \end{array} \quad \begin{array}{r} 0000\ 0001 \\ + \underline{1111\ 0001} \\ \hline \end{array}$$

$$\begin{array}{r} 0111\ 0100 \\ + \underline{1100\ 1100} \\ \hline \end{array} \quad \begin{array}{r} 0000\ 0001 \\ + \underline{0111\ 1111} \\ \hline \end{array}$$

10. Khi tăng nội dung của thanh ghi lên 3 lần (mỗi lần tăng 1) làm cho bit Zero của thanh trạng thái được Set ở mức logic 1 ở lần tăng thứ 3. Vậy giá trị ban đầu chứa trong thanh ghi là bao nhiêu?

11. Thanh ghi địa chỉ bộ nhớ dùng để chỉ đến:

- A. Nội dung của bộ nhớ.
- B. Vị trí của ô nhớ.
- C. Vị trí của CPU.
- D. Vị trí của các thanh ghi.

12. Thanh ghi địa chỉ bộ nhớ kết nối với bus dữ liệu bên trong vi xử lý để nó có thể nạp giá trị từ:

- A. Thanh ghi Program counter.
- B. Các thanh ghi thông dụng.
- C. Memory.
- D. Cả 3 câu trên.

13. Các ngõ ra thanh ghi địa chỉ dùng để kết nối với

- A. Thanh ghi Accumulator của vi xử lý.
- B. Bus dữ liệu bên trong của Microprocessor.
- C. Bus địa chỉ bộ nhớ bên ngoài của vi xử lý.
- D. Với ngõ vào của bộ giải mã lệnh.

*return  
the end*

# CẤU TRÚC VI XỬ LÝ 16 BIT 8086/88

## CẤU TRÚC VI XỬ LÝ 8086/88

1. CẤU TRÚC BÊN TRONG VI XỬ LÝ 8086/88.
2. CÁC THANH GHI BÊN TRONG VI XỬ LÝ.
3. BỘ NHÓ PHÂN ĐOẠN CỦA VI XỬ LÝ.
4. RESET VI XỬ LÝ.
5. TÓM TẮT PHẦN CỨNG VI XỬ LÝ 8086.
  - a. Khảo sát đặc tính điện
  - b. Mô tả chân của vi xử lý
  - c. Tạo xung clock và mạch reset 8086/88

## CÁC PHƯƠNG PHÁP ĐỊNH ĐỊA CHỈ CỦA VI XỬ LÝ 8086

1. Cách định địa chỉ dùng thanh ghi
2. Các định địa chỉ tức thời
3. Các định địa chỉ trực tiếp
4. Các cách định địa chỉ gián tiếp
  - a. Định địa chỉ gián tiếp thanh ghi
  - b. Định địa chỉ nền
  - c. Định địa chỉ chỉ số
  - d. Định địa chỉ nền chỉ số với độ dài
5. Các định địa chỉ chuỗi
6. Các định địa chỉ cổng
7. Các định địa chỉ ngăn xếp
8. Các quy tắc kết thanh ghi đoạn và thanh ghi offset

## TẬP LỆNH CỦA VI XỬ LÝ 8086

1. Mã lệnh của vi xử lý 8086/88.
2. Tổng quan về tập lệnh của vi xử lý 8086.
3. Khảo sát tập lệnh của vi xử lý 8086.

## CÂU HỎI VÀ BÀI TẬP

## ***LỊỆT KÊ CÁC HÌNH***

- Hình 3-1. Cấu trúc bên trong của vi xử lý 8086/88.  
 Hình 3-2. Quá trình nhận lệnh và thực thi lệnh.  
 Hình 3-3. 14 thanh ghi bên trong vi xử lý.  
 Hình 3-4. Các thanh ghi đoạn quản lý các vùng nhớ.  
 Hình 3-5. Các bit trong thanh ghi cờ.  
 Hình 3-6. Tổ chức bộ nhớ của vi xử lý 8086.  
 Hình 3-7. Sơ đồ chân vi xử lý 8086 và vi xử lý 8088.  
 Hình 3-8. Giản đồ thời gian truy xuất bộ nhớ của vi xử lý 8086.  
 Hình 3-9. Tạo các tín hiệu ghi đọc bộ nhớ và IO.  
 Hình 3-10. Vi xử lý hoạt động ở chế độ tối thiểu.  
 Hình 3-11. Vi xử lý hoạt động ở chế độ tối đa.  
 Hình 3-12. Mạch tạo tín hiệu reset.  
 Hình 3-13a. Trước khi thực hiện lệnh MOV AX, BX.  
 Hình 3-13b. Sau khi thực hiện lệnh MOV AX, BX.  
 Hình 3-14a. Trước khi thực hiện lệnh MOV AL,15H.  
 Hình 3-14b. Sau khi thực hiện lệnh MOV AL,15H.  
 Hình 3-15a. Trước khi thực hiện lệnh MOV CX,BETA.  
 Hình 3-15b. Sau khi thực hiện lệnh MOV CX,BETA.  
 Hình 3-16a. Trước khi thực hiện lệnh MOV AX,[SI].  
 Hình 3-16b. Sau khi thực hiện lệnh MOV AX,[SI].  
 Hình 3-17a. Trước khi thực hiện lệnh MOV [BX] +BETA,AL.  
 Hình 3-17b. Sau khi thực hiện lệnh MOV [BX] +BETA,AL.  
 Hình 3-18a. Trước khi thực hiện lệnh MOV AL,[SI] +ARRAY.  
 Hình 3-18b. Sau khi thực hiện lệnh MOV AL,[SI] +ARRAY.  
 Hình 3-19a. Trước khi thực hiện lệnh MOV AX,[BX] [SI] +BETA.  
 Hình 3-19b. Sau khi thực hiện lệnh MOV AX,[BX] [SI] +BETA.  
 Hình 3-20. Hoạt động của ngăn xếp với lệnh PUSH.  
 Hình 3-21. Dạng mã lệnh.

## ***LỊỆT KÊ CÁC BẢNG***

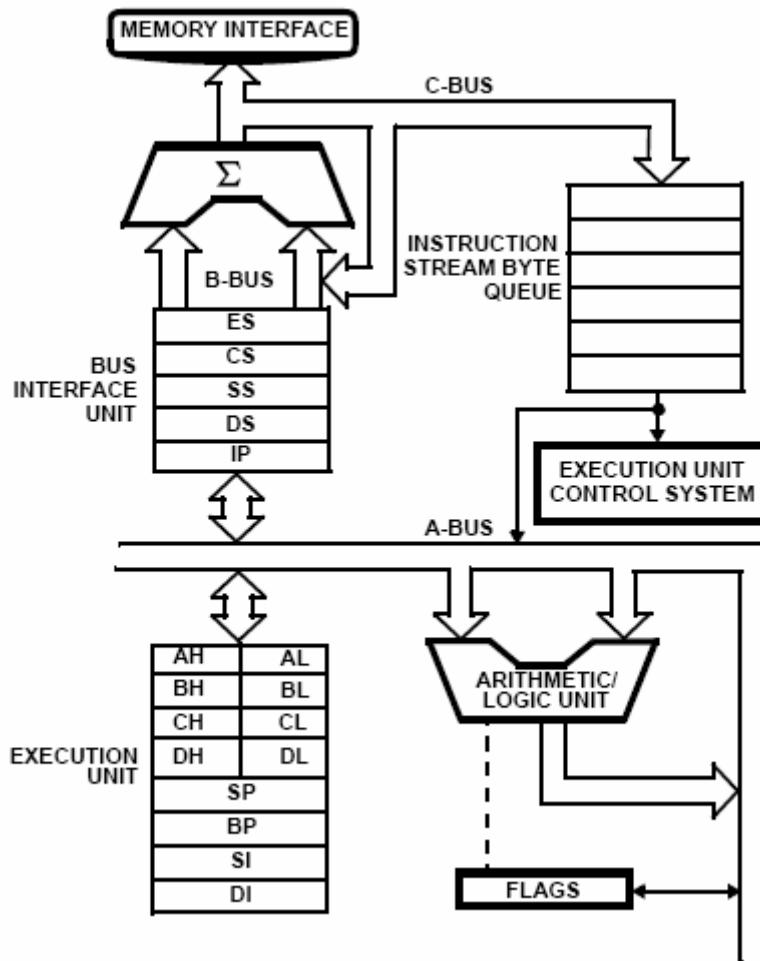
- Bảng 3-1. Các thanh ghi khi sử dụng được hiểu ngầm.  
 Bảng 3-2. Bảng địa chỉ 20 bit.  
 Bảng 3-3. Các đoạn bộ nhớ độc lập.  
 Bảng 3-4. Trạng thái của các thanh ghi khi vi xử lý bị reset.  
 Bảng 3-5. Đặc tính điện.  
 Bảng 3-6. Đặt tính tốc độ và công suất tiêu tán.  
 Bảng 3-7. Mã hóa truy xuất bộ nhớ của vi xử lý 8086.  
 Bảng 3-8. Tính toán địa chỉ hiệu dụng cho các cách định địa chỉ gián tiếp.  
 Bảng 3-9. Qui tắc kết hợp các thanh ghi đoạn và thanh ghi offset.  
 Bảng 3-10. Bảng liệt kê giá trị của REG.  
 Bảng 3-11. Bảng liệt kê giá trị của MOD.  
 Bảng 3-12. Bảng liệt kê giá trị của R/M.  
 Bảng 3-13. Bảng liệt kê chu kỳ thực hiện của các lệnh.

## I. CẤU TRÚC CỦA VI XỬ LÝ 8086/88:

### 1. CẤU TRÚC BÊN TRONG VI XỬ LÝ 8086/88:

Như đã trình bày ở phần giới thiệu thì chương này sẽ khảo sát vi xử lý 16 bit và vi xử lý tiêu biểu là 8086/88.

Cấu trúc bên trong của vi xử lý gồm hai khối chính như hình 3-1, nó được tổ chức thành 2 bộ xử lý riêng là BIU (Bus Interface Unit) và EU (Execution Unit).



Hình 3-1. Cấu trúc bên trong của vi xử lý 8086/88.

**Chức năng của khối BIU:** đón mã lệnh từ bộ nhớ và đặt chúng vào hàng chờ lệnh. Khối EU giải mã và thực thi những mã lệnh trong hàng chờ lệnh.

Vì các đơn vị làm việc độc lập với nhau nên khối BIU có thể đang nhận 1 lệnh mới trong khi EU đang thực thi lệnh trước đó. Khi khối EU sẵn sàng thực thi lệnh mới thì nó sẽ tìm thấy ngay mã lệnh mới đang đợi trong hàng chờ lệnh.

**Chức năng của khối EU:** nhận những mã lệnh của chương trình và dữ liệu từ BIU, thực thi lệnh và gởi kết quả trở lại cho khối BIU để lưu vào thanh ghi hay bộ nhớ hoặc có thể xuất ra ngoài thiết bị giao tiếp. Chú ý: khối EU không có kết nối với bus hệ thống nên chỉ nhận và xuất dữ liệu đều thông qua khối BIU.

Vi xử lý 8088 cũng là một vi xử lý 16 bit, sự khác nhau giữa vi xử lý 8086 và 8088 là khối BIU: ở vi xử lý 8086 thì bus dữ liệu giao tiếp bên trong và bên ngoài đều là 16 bit trong khi đó vi xử lý 8088 thì bus bên trong là 16 bit còn bus bên ngoài chỉ có 8 bit. Sự khác nhau này dẫn đến bộ nhớ giao tiếp với vi xử lý 8086 và 8088 cũng khác nhau. Vi xử lý 8088 thì bộ nhớ tổ chức theo byte, còn vi xử lý 8086 thì bộ nhớ tổ chức theo word (2 byte).

Một điểm khác biệt thứ 2 đó là hàng đợi lệnh của vi xử lý 8086 có 6 byte còn vi xử lý 8088 chỉ có 4 byte.

Còn tất cả các khối còn lại đều giống nhau nên điều này cho phép một chương trình cho 8086 thì có thể chạy trên vi xử lý 8088.

### **Quá trình nhận lệnh và thực thi:**

**Bước 1:** Khối BIU xuất nội dung của thanh ghi con trỏ lệnh IP (**Instruction Pointer**) ra bus địa chỉ để truy xuất bộ nhớ tiến hành đọc mã lệnh rồi lưu vào khối BIU.

**Bước 2:** Tăng thanh ghi con trỏ lệnh lên để trỏ đến lệnh kế.

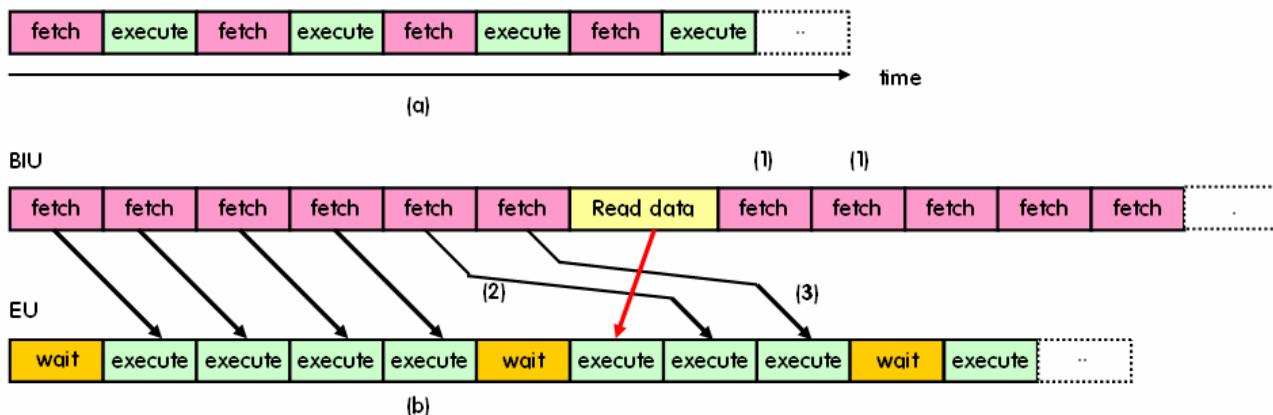
**Bước 3:** Khi mã lệnh đưa vào trong BIU thì nó được chuyển vào hàng chờ lệnh (**queue**). Hàng chờ lệnh là các thanh ghi tổ chức theo kiểu vào trước ra trước (FIFO = First In First Out).

**Bước 4:** Giả sử ban đầu hàng đợi lệnh trống thì khối EU sẽ lập tức lấy mã lệnh và thực hiện ngay mã lệnh đó.

**Bước 5:** Trong khi khối EU đang thực thi lệnh thì khối BIU tiến hành nhận mã lệnh mới. Tuỳ thuộc vào thời gian thực thi lệnh mà khối BIU có thể lấy nhiều mã lệnh trước khi khối EU lấy mã lệnh kế.

Khối BIU được lập trình để nhận một lệnh mới bất kỳ khi nào hàng đợi còn chờ trống. Sự kết hợp này có ưu điểm là khối EU thực hiện lệnh liên tục không phải chờ nhận mã lệnh so với vi xử lý trước nhờ khối BIU dẫn đến làm tăng tốc độ xử lý chương trình.

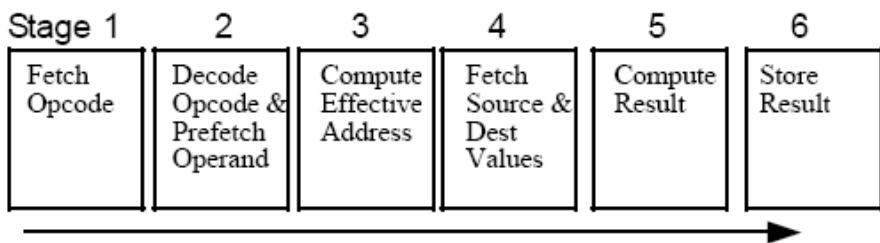
Toàn bộ quá trình thực hiện được minh họa như hình 3-2:



Hình (a). Kiểu thực hiện của các vi xử lý theo cách thông thường.

Hình (b). Kiểu thực hiện của vi xử lý theo cấu trúc đường ống.

Hình 3-2. Quá trình nhận lệnh và thực thi lệnh.



T1	T2	T3	T4	T5	T6	T7	T8	T9...
Opcode	Decode	Address	Values	Compute	Store	Instruction #1		
	Opcode	Decode	Address	Values	Compute	Store	Instruction #2	
	Opcode	Decode	Address	Values	Compute	Store	Instruction #3	
		Opcode	Decode	Address	Values	Compute	Store	

**Ghi chú:** (1) Các mã lệnh này bị bỏ qua.

(2) Lệnh này cần dữ liệu không nằm trong hàng đợi.

(3) Khi thực hiện lệnh Jump.

(Fetch: đón lệnh từ bộ nhớ, Execute: thực hiện lệnh, Read data: đọc dữ liệu, Wait: đợi).

Trong quá trình khối BIU nhận mã lệnh đưa vào hàng đợi và khối EU thực thi lệnh thì vẫn xảy ra 3 trường hợp khối EU phải rời vào trạng thái chờ và khối BIU cũng rời vào trạng thái chờ.

**Trường hợp 1:** xảy ra khi lệnh cần truy xuất đến một ô nhớ để lấy dữ liệu không có trong hàng đợi. Trong trường hợp này khối BIU phải tạm ngừng việc nhận mã lệnh mà tiến hành xuất địa chỉ để nhận dữ liệu từ bộ nhớ gởi cho khối EU. Sau khi thực thi xong khối EU tiếp tục nhận mã lệnh từ hàng và BIU tiếp tục đón mã lệnh gởi vào hàng.

**Trường hợp 2:** xảy ra khi lệnh đang thực thi là lệnh nhảy “jump”. Trong trường hợp này khối EU sẽ nhảy tới thực hiện lệnh tại địa chỉ mới, trong khi đó khối BIU đã nhận mã của lệnh kế, chính vì thế EU phải đợi khối BIU nhận địa chỉ của nơi nhảy đến, sau khi nhận địa chỉ từ khối BIU thì khối EU sẽ nhảy và khối BIU tiếp tục nhận mã lệnh từ bộ nhớ đưa vào hàng đợi.

**Trường hợp 3:** trường hợp làm BIU ngừng nhận mã lệnh. Trường hợp này xảy ra khi khối EU thực thi lệnh mất nhiều thời gian. Ví dụ như lệnh AAM (lệnh hiệu chỉnh thập phân cho lệnh nhân) cần 83 chu kỳ xung nhịp mới thực hiện xong, trong khi đó khối BIU cần 4 xung thì nhận xong một mã lệnh nên BIU nhận mã lệnh làm đầy hàng đợi nhưng EU vẫn chưa thực hiện xong lệnh nên BIU phải chờ.

Quá trình nhận lệnh trong khi EU thực thi lệnh có một tiện ích là có thể sử dụng các bộ nhớ có tốc độ truy xuất chậm mà vẫn không làm ảnh hưởng đến năng suất hệ thống nhờ có kiến trúc đường ống.

**Chuẩn PC:** Với bus dữ liệu 16 bit thì cần 2 dãy bộ nhớ nhưng tại thời điểm đó bộ nhớ rất đắt tiền. Do đó Intel cho ra đời vi xử lý 8088 hoàn toàn giống 8086 ngoại trừ bus dữ liệu bên ngoài là 8 bit.

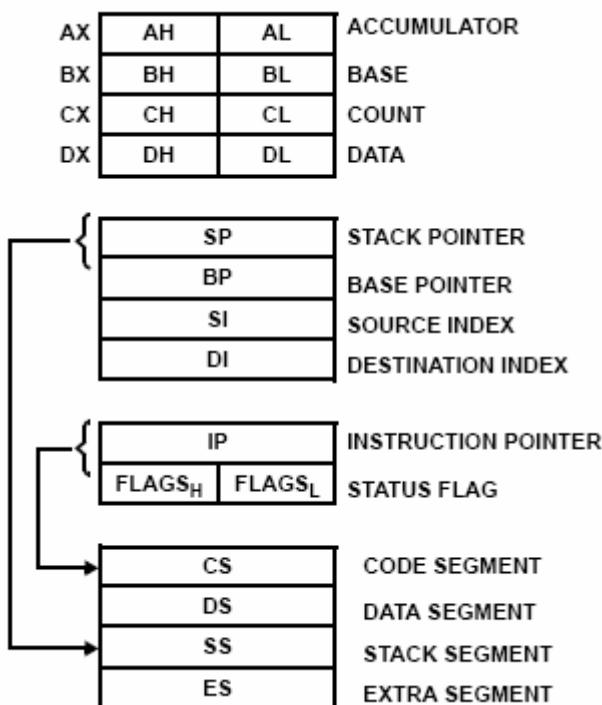
Sau đó IBM cho ra đời máy tính IBM-PC sử dụng vi xử lý 8088 và bộ nhớ 16Kbyte (có thể mở rộng lên đến 64Kbyte) và cho ra máy tính PC chuẩn với tốc độ xung clock là 4.77 MHz.

## 2. CÁC THANH GHI TRONG VI XỬ LÝ 8086/88:

Vi xử lý 8086/88 có 14 thanh ghi và được phân loại như sau:

- Các thanh ghi dữ liệu.
- Các thanh ghi chỉ số và con trỏ.
- Các thanh ghi đoạn hay còn gọi là các thanh ghi phân đoạn.
- Các thanh ghi trạng thái và điều khiển.

Hình 3-3 trình bày các thanh ghi bên trong vi xử lý 8086/88:



**Hình 3-3. 14 thanh ghi bên trong vi xử lý.**

### a. Khảo sát các thanh ghi dữ liệu:

Các thanh ghi dữ liệu bao gồm 4 thanh ghi có tên là thanh ghi AX, BX, CX và DX, chúng đều là thanh ghi 16 bit nên cho phép lưu trữ được dữ liệu 16 bit. Mỗi thanh ghi có thể chia ra làm 2 thanh ghi: thanh ghi lưu trữ byte thấp (Low) và thanh ghi lưu trữ byte cao (High) có thể truy xuất độc lập và cụ thể là:

- AH, AL – được chia từ thanh ghi AX.
- BH, BL – được chia từ thanh ghi BX.
- CH, CL – được chia từ thanh ghi CX.
- DH, DL – được chia từ thanh ghi DX.

Chức năng của các thanh ghi: dùng để thực hiện các phép toán số học, phép toán logic và chuyển dữ liệu.

Có một số thanh ghi khi sử dụng được hiểu ngầm trong một số lệnh như cho ở bảng 3-1.

Thanh ghi	Được hiểu ngầm trong một số lệnh:
AX	MUL, IMUL, DIV, IDIV IN, OUT, CWD và các phép toán chuỗi.
AL	MUL, IMUL, DIV, IDIV, IN, OUT AAA, AAD, AAM, AAX, CBW DAA, DAS và các phép toán chuỗi.
AH	MUL, IMUL, DIV, IDIV, CBW
DX	XLAT
CX	LOOP, LOOPE, LOOPNE Các phép toán string với tiếp đầu ngữ REP
CL	RCR, RCL, ROR, ROL (quay với số lần lưu trong CL) SHR, SAR, SAL (dịch với số lần lưu trong CL)
DX	MUL, IMUL, DIV, IDIV

Bảng 3-1. Các thanh ghi khi sử dụng được hiểu ngầm.

- **Thanh ghi tích lũy AX (Accumulator):**

Là thanh ghi được sử dụng nhiều nhất trong các lệnh số học, lệnh logic và truyền dữ liệu với lý do là khi sử dụng thanh ghi này tạo ra mã máy ngắn nhất.

Trong các phép toán nhân hoặc chia thì một trong các số hạn tham gia phải chứa trong AH hoặc AL, các thao tác vào ra cũng sử dụng thanh ghi AH hoặc AL.

- **Thanh ghi cơ sở BX (Base):**

Thanh ghi BX được dùng cho tính toán địa chỉ trong phương pháp định địa chỉ gián tiếp.

- **Thanh ghi đếm CX (Count):**

Việc thực hiện các vòng lặp được thực hiện dễ dàng nhờ thanh ghi CX, trong đó CX đóng vai trò là bộ đếm vòng lặp. Một lệnh thường sử dụng thanh ghi CX đó là lệnh REP (Repeat). CL cũng được sử dụng là một biến đếm trong các lệnh dịch hay xoay các bit.

- **Thanh ghi dữ liệu DX (Data):**

DX dùng để định địa chỉ gián tiếp trong các thao tác xuất nhập (In/Out), nó cũng còn được sử dụng chứa toán hạn, kết quả trong phép nhân và chia.

**b. Khảo sát các thanh ghi chỉ số và thanh ghi con trỏ:**

Các thanh ghi SP, BP, SI và DI là các thanh ghi 16 bit dùng để chứa các giá trị offset (còn gọi là độ lệch) khi định địa chỉ trong một bộ nhớ đoạn (*segment*). Các thanh ghi này còn được sử dụng trong các phép toán số học và logic.

Hai thanh ghi con trỏ SP và BP dùng để quản lý bộ nhớ ngăn xếp hiện hành.

Hai thanh ghi chỉ số SI và DI dùng để truy xuất hay quản lý vùng nhớ dữ liệu và vùng nhớ dữ liệu mở rộng (*extra segment*).

- **Thanh ghi con trỏ ngăn xếp SP (Stack Pointer):**

Dùng để kết hợp với thanh ghi đoạn SS (Stack Segment) để quản lý địa chỉ của bộ nhớ ngắn xép.

- **Thanh ghi con trả cơ sở BP (Base Pointer):**

Thanh ghi này được dùng để truy cập dữ liệu trong ngắn xép mà không làm thay đổi SP. Tuy nhiên, khác với SP thanh ghi BP cũng còn được sử dụng để truy cập dữ liệu ở các vùng nhớ khác.

- **Thanh ghi chỉ số nguồn SI (Source Index):**

Thanh ghi SI được sử dụng để trỏ tới các ô nhớ trong đoạn dữ liệu được chỉ định bởi thanh ghi đoạn dữ liệu DS (Data Segment), có thể truy cập dễ dàng các ô nhớ liên tiếp bằng cách tăng giá trị của thanh ghi SI.

- **Thanh ghi chỉ số đích DI (Destination Index):**

Thanh ghi DI có chức năng tương tự như thanh ghi SI và được dùng kết hợp với thanh ghi đoạn mở rộng ES (Extra Segment). Cả hai DI và SI thích hợp trong các thao tác sao chép, di chuyển hoặc so sánh các khối dữ liệu có dung lượng đến 64kB.

**c. Khảo sát các thanh ghi đoạn:**

Có 4 thanh ghi đoạn CS, DS, SS và ES đều là các thanh ghi 16 bit.

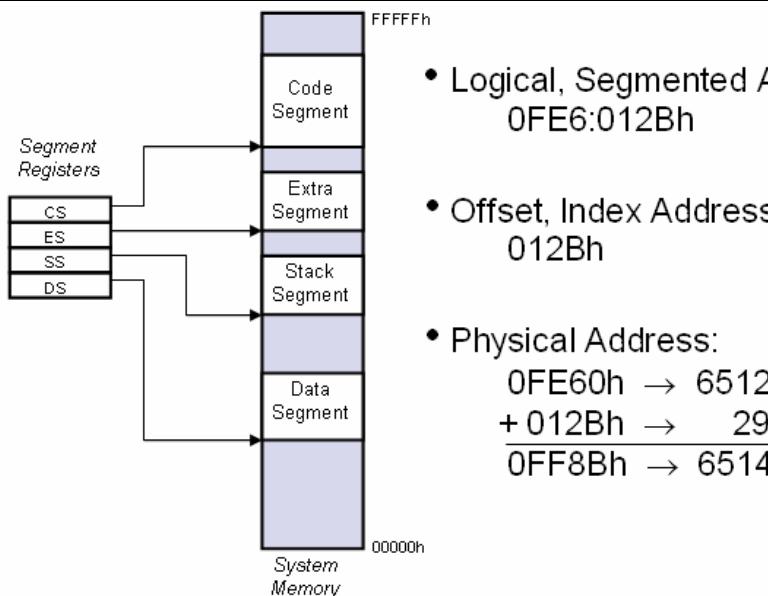
Trước khi trình bày chức năng của các thanh ghi này chúng ta cần phân tích bộ nhớ giao tiếp với vi xử lý 8086/88.

Bộ nhớ là tập hợp các byte ô nhớ trong đó mỗi byte có một địa chỉ xác định. Vi xử lý 8086/88 có 20 đường địa chỉ để giao tiếp với bộ nhớ có dung lượng 1 Mbyte, mỗi ô nhớ có một địa chỉ 20 bit. Các byte đầu tiên của bộ nhớ có địa chỉ như sau:

A <sub>19</sub>	A <sub>18</sub>	A <sub>17</sub>	A <sub>16</sub>	A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	HEX	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00000
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	00001	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	00002	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFFF

**Bảng 3-2. Bảng địa chỉ 20 bit.**

Do các địa chỉ của các ô nhớ là 20 bit không thể chứa trong một thanh ghi 16 bit của vi xử lý 8086/88 nên 8086/88 chia bộ nhớ 1Mbyte thành 16 đoạn bộ nhớ (Memory Segment). 16 đoạn bộ nhớ do các thanh ghi đoạn quản lý gọi là thanh ghi **segment** được trình bày như hình 3-4:

**Hình 3-4. Các thanh ghi đoạn quản lý các vùng nhớ.**

Mỗi đoạn bộ nhớ đều có dung lượng là 64Kbyte và có địa chỉ từ 0000H cho đến FFFFH. Để quản lý địa chỉ vùng nhớ của một đoạn phải sử dụng thanh ghi 16 bit gọi là thanh ghi **offset** – **địa chỉ trong 1 đoạn còn gọi là địa chỉ offset**.

Khi đó địa chỉ vật lý (còn gọi là địa chỉ logic) của một ô nhớ được xác định bằng cách kết hợp “thanh ghi quản lý địa chỉ đoạn” và “thanh ghi quản lý địa chỉ lệnh” **segment:offset**.

Trong hình 3-4 có cho ví dụ địa chỉ **segment:offset = 0FE6:012Bh** thì địa chỉ segment là 0FE6H và địa chỉ offset là 012BH. Để tìm địa chỉ vật lý (Physical Address = PA) của ô nhớ thì vi xử lý phải dịch địa chỉ segment về bên trái 4 bit (bit thêm vào là bit 0) hay có thể xem như thêm vào một con số 0H bên trái (địa chỉ 0FE6H sau khi dịch thì trở thành 0FE60H) và sau đó cộng với địa chỉ offset (012BH), kết quả địa chỉ vật lý của ô nhớ có địa chỉ **0FE6:012Bh** sẽ là **0FF8Bh**.

Bốn thanh ghi đoạn có chức năng quản lý 4 vùng nhớ đoạn tương ứng:

- Thanh ghi đoạn CS cùng với thanh ghi offset có chức năng quản lý đoạn bộ nhớ chứa mã lệnh (Code Segment).
- Thanh ghi đoạn ES cùng với thanh ghi offset có chức năng quản lý đoạn bộ nhớ mở rộng (Extra Segment).
- Thanh ghi đoạn SS cùng với thanh ghi offset có chức năng quản lý đoạn bộ nhớ ngăn xếp chứa các dữ liệu tạm thời (Stack Segment).
- Thanh ghi đoạn DS cùng với thanh ghi offset có chức năng quản lý đoạn bộ nhớ chứa dữ liệu (Data Segment).

Vùng nhớ 1Mbyte được chia ra làm 16 đoạn độc lập như bảng 3-3:

Vùng nhớ đoạn	Segment: offset (bắt đầu)	Segment: offset (kết thúc)	Địa chỉ vật lý
0	0000:0000	0000:FFFF	00000 ÷ 0FFFF

<b>1</b>	<b>1000:0000</b>	<b>1000:FFFF</b>	<b>10000 ÷ 1FFFF</b>
<b>2</b>	<b>2000:0000</b>	<b>2000:FFFF</b>	<b>20000 ÷ 2FFFF</b>
<b>3</b>	<b>3000:0000</b>	<b>3000:FFFF</b>	<b>30000 ÷ 3FFFF</b>
<b>4</b>	<b>4000:0000</b>	<b>4000:FFFF</b>	<b>40000 ÷ 4FFFF</b>
<b>5</b>	<b>5000:0000</b>	<b>5000:FFFF</b>	<b>50000 ÷ 5FFFF</b>
<b>6</b>	<b>6000:0000</b>	<b>6000:FFFF</b>	<b>60000 ÷ 6FFFF</b>
<b>7</b>	<b>7000:0000</b>	<b>7000:FFFF</b>	<b>70000 ÷ 7FFFF</b>
<b>8</b>	<b>8000:0000</b>	<b>8000:FFFF</b>	<b>80000 ÷ 8FFFF</b>
<b>9</b>	<b>9000:0000</b>	<b>9000:FFFF</b>	<b>90000 ÷ 9FFFF</b>
<b>10</b>	<b>A000:0000</b>	<b>A000:FFFF</b>	<b>A0000 ÷ AFFFF</b>
<b>11</b>	<b>B000:0000</b>	<b>B000:FFFF</b>	<b>B0000 ÷ BFFFF</b>
<b>12</b>	<b>C000:0000</b>	<b>C000:FFFF</b>	<b>C0000 ÷ CFFFF</b>
<b>13</b>	<b>D000:0000</b>	<b>D000:FFFF</b>	<b>D0000 ÷ DFFFF</b>
<b>14</b>	<b>E000:0000</b>	<b>E000:FFFF</b>	<b>E0000 ÷ EFFFF</b>
<b>15</b>	<b>F000:0000</b>	<b>F000:FFFF</b>	<b>F0000 ÷ FFFFF</b>

**Bảng 3-3. Các đoạn bộ nhớ độc lập.**

Với việc phân chia như bảng trên thì các đoạn là độc lập không bị chồng lên nhau thật ra còn có thể cách xa nhau, nhưng trong thực tế do phần cứng thiết kế không đầy đủ dung lượng 1Mbyte, ví dụ như hệ thống chỉ có 128Kbyte thì bắt buộc các đoạn phải chồng lên nhau. Người lập trình phải phân chia các vùng nhớ theo hệ thống phần cứng đã thiết kế.

Một chương trình không phải lúc nào cũng chiếm hết một đoạn 64KByte, do đặc điểm chồng nhau giữa các đoạn nên cho phép các đoạn của một chương trình nhỏ hơn 64KB có thể đặt gần nhau. Tại một thời điểm, chỉ có các ô nhớ được định địa chỉ bởi 4 thanh ghi đoạn mới có thể truy cập, nghĩa là chỉ có 4 đoạn bộ nhớ là tác động. Tuy nhiên nội dung của các thanh ghi đoạn có thể thay đổi bởi chương trình để truy cập đến các đoạn khác nhau.

#### **d. Khảo sát thanh ghi con trả lệnh và thanh ghi trạng thái :**

**Thanh ghi con trả lệnh IP (Instruction Pointer):** có chức năng quản lý lệnh giống như thanh ghi PC (Program Counter) của các vi xử lý 8 bit.

Khối BIU quản lý thanh ghi IP để lưu trữ địa chỉ offset và kết hợp với thanh ghi CS để tạo ra địa chỉ vật lý để nhận mã lệnh từ bộ nhớ chứa mã lệnh Code Segment. Sau khi đón mã lệnh thì nội dung của thanh ghi IP tăng lên để đón mã của lệnh kế.

Người lập trình không thể sử dụng trực tiếp thanh ghi này.

**Thanh ghi cờ (Flag Register):** của 8086 có độ dài 16bit, trong đó có 3 bit điều khiển và 6 bit trạng thái còn được gọi là cờ. Các bit còn lại chưa được thiết lập nên người dùng không thể truy xuất. Hình 3-5 là cấu tạo của thanh ghi cờ trong 8086/88:

	15		0
x	x	x	x
O	D	IF	T
S	Z	x	A
x	P	x	C
CF	<i>Carry Flag</i>	Arithmetic Carry/Borrow	
OF	<i>Overflow Flag</i>	Arithmetic Overflow	
ZF	<i>Zero Flag</i>	Zero Result; Equal Compare	
SF	<i>Sign Flag</i>	Negative Result; Non-Equal Compare	
PF	<i>Parity Flag</i>	Even Number of "1" bits	
AF	<i>Auxiliary Carry</i>	Used with BCD Arithmetic	
DF	<i>Direction Flag</i>	Auto-Increment/Decrement used for "string operations"	
IF	<i>Interrupt Flag</i>	Enables Interrupts allows "fetch-execute" to be interrupted	
TF	<i>Trap Flag</i>	Allows Single-Step for debugging; causes interrupt after each op	

Hình 3-5. Các bit trong thanh ghi cờ.

Khi vi xử lý thực hiện các phép toán thì căn cứ vào kết quả sau khi xử lý sẽ tác động đến một số cờ trong thanh ghi trạng thái như đã trình bày ở phần trước. Chức năng của các thanh ghi cờ được đề cập lại như sau:

#### + Cờ tràn CF:

Cờ nhỡ CF bằng 1 khi thực hiện phép toán cộng có tràn hoặc phép toán trừ có mượn, ngược lại nếu không tràn, không mượn thì cờ CF bằng 0. Cờ CF cũng bị ảnh hưởng bởi các lệnh xoay và lệnh dịch.

#### + Cờ chẵn lẻ PF:

Sau khi thực hiện các lệnh số học hoặc lệnh logic đổi với dữ liệu dạng byte: nếu byte kết quả là số chẵn thì cờ PF bằng '1' ngược lại thì cờ PF bằng '0' nếu là số lẻ, đối với dữ liệu xử lý là word chỉ xét các byte thấp.

#### + Cờ nhỡ phụ AF:

Khi thực hiện lệnh cộng hoặc lệnh trừ số BCD: nếu dữ liệu dạng byte thì cờ AF bằng 1 khi cộng hoặc trừ 4 bit thấp bị tràn hoặc có mượn, nếu dữ liệu dạng word thì cờ AF bằng 1 khi cộng hoặc trừ byte thấp bị tràn hoặc có mượn. Cờ AF được sử dụng trong các lệnh xử lý với dữ liệu dạng số BCD.

#### + Cờ zero ZF:

Cờ zero bằng 1 khi kết quả xử lý bằng 0.

#### + Cờ dấu SF:

Cờ dấu bằng '1' khi bit MSB của kết quả bằng '1' tức là số âm, đối với lệnh byte MSB là bit 7 và trong lệnh word là bit 15. Ngược lại cờ SF bằng 0.

#### + Cờ bẫy TF:

Có chức năng thực hiện chương trình theo từng bước, khi TF bằng ‘1’ thì vi xử lý 8086/88 phát sinh ngắt cứng loại 1. Chương trình DEBUG sử dụng khi thi hành lệnh T (trace) để chạy từng lệnh một. Đầu tiên DEBUG thiết lập cờ TF rồi mới chuyển điều khiển cho lệnh đó. Sau khi lệnh được thi hành vi xử lý sẽ phát sinh ngắt do TF được lập chương trình DEBUG sử dụng chính phục vụ ngắt này để lấy quyền điều khiển từ vi xử lý.

#### + Cờ ngắt IF:

Cờ ngắt được sử dụng để điều khiển các ngắt phần cứng bên ngoài, nếu cờ IF bằng 1 thì các ngắt được phép. Khi cờ IF bằng 0 thì cấm các ngắt ngoài hay còn gọi là các ngắt đã bị che. Thực ra vẫn còn một ngắt cứng được phép ngắt đó là ngắt NMI (Non Maskable Interrupt).

Trước khi vi xử lý thực hiện chương trình phục vụ ngắt nó sẽ tự động xóa cả cờ IF và cờ TF, việc này có chức năng cấm các ngắt khác làm gián đoạn vi xử lý đang thực hiện chương trình hiện tại. Tuy nhiên nếu chúng ta cho phép ngắt trong chương trình phục vụ ngắt và nếu ngắt xảy ra thì chương trình này sẽ bị ngắt để phục vụ chương trình ngắt mới.

#### + Cờ tràn OF:

Cờ tràn OF bằng 1 khi kết quả là số nhị phân có dấu vượt quá giới hạn qui định và ngược lại thì cờ OF bằng ‘0’.

Như đã trình bày ở trên thì cờ CF là cờ tràn đối với dữ liệu xử lý và kết quả sau khi xử lý đều là số nhị phân không dấu. Còn cờ OF là cờ tràn đối với số nhị phân có dấu.

Với số nhị phân không dấu 8 bit thì vùng dữ liệu có giá trị bắt đầu từ 0 đến 255, số nhị phân 16 bit thì từ 0 đến 65535.

Với số nhị phân có dấu 8 bit thì vùng dữ liệu có giá trị từ -128 đến +127, số nhị phân 16 bit có giá trị từ -32768 đến 32767.

#### + Cờ điều khiển DF:

Là một trong ba cờ điều khiển dùng điều khiển các lệnh xử lý của vi xử lý, công dụng của DF là dịch hướng cho các thao tác chuỗi, các thao tác này được thực hiện bởi hai thanh ghi chỉ số SI & DI, nội dung của hai thanh ghi này sẽ tự động tăng lên khi DF = 0 và giảm xuống khi DF = 1.

### 3. BỘ NHỚ PHÂN ĐOẠN

Vi xử lý 8086 là vi xử lý 16 bit nhưng bộ nhớ vẫn tổ chức theo byte (8bit). Có 2 nguyên nhân của việc sử dụng tổ chức bộ nhớ như thế là:

**Nguyên nhân thứ nhất:** Cho phép vi xử lý truy xuất byte và truy xuất word đều được. Tổ chức rất quan trọng với các thiết bị IO như máy in, thiết bị đầu cuối, modem đều xử lý dữ liệu được mã hoá dạng ASCII 7 bit hay 8 bit.

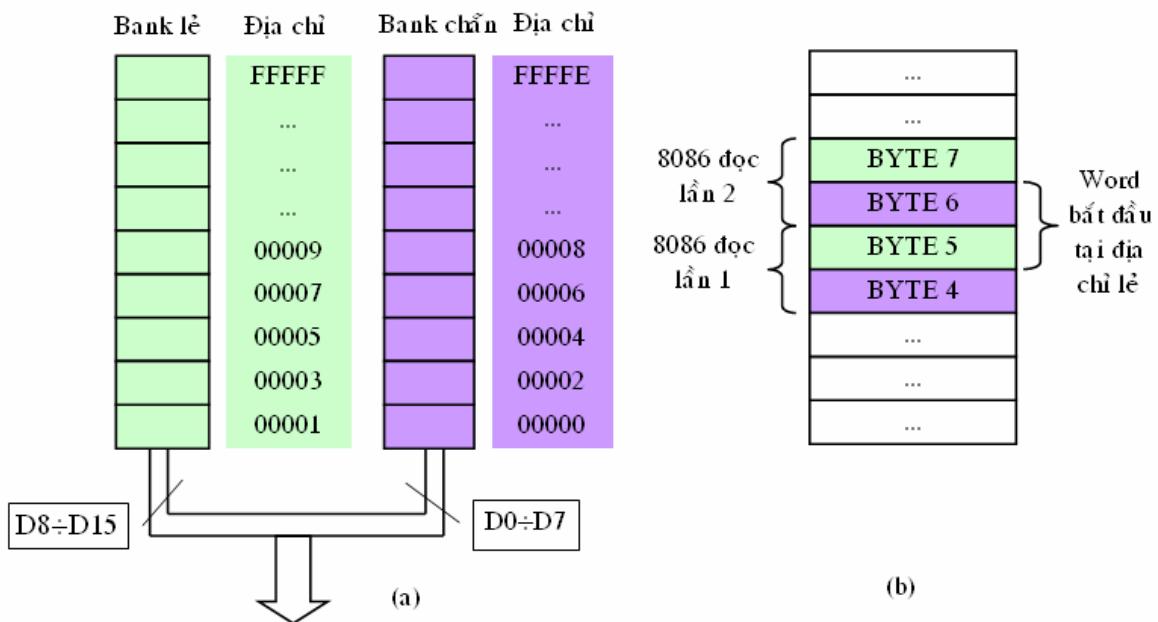
**Nguyên nhân thứ hai:** Nhiều mã lệnh của vi xử lý 8086/88 chỉ có độ dài 1 byte, các lệnh khác có độ dài từ 2 đến 8 byte. Với cách tổ chức theo byte sẽ cho phép truy xuất các byte một cách độc lập để xử lý các lệnh có số byte mã lệnh lẻ.

Vi xử lý 8086 khi truy xuất dữ liệu dài 16 bit thì nó sẽ truy xuất đồng thời byte có địa chỉ lẻ và byte có địa chỉ chẵn nên tổ chức bộ nhớ của vi xử lý 8086 được chia làm 2 dãy bộ nhớ, một dãy có địa chỉ lẻ và một dãy có địa chỉ chẵn như hình 3-6.

Với cách tổ chức bộ nhớ thành 2 dãy ta có thể xem bộ nhớ 1024Kbyte mỗi ô nhớ 1 byte chỉ còn 512Kword – mỗi ô nhớ chứa 2 byte. Khi đó dữ liệu của mỗi ô đều có địa chỉ chẵn.

Khi truy xuất dữ liệu 16 bit có địa chỉ bắt đầu là địa chỉ chẵn thì chỉ cần 1 lần truy xuất thì có thể lấy được 16 bit dữ liệu, nhưng khi truy xuất dữ liệu 16 bit có địa chỉ lẻ: trong đó 8 bit thấp nằm ở ô nhớ có địa chỉ lẻ và 8 bit cao nằm ở ô nhớ có địa chỉ chẵn thì vi xử lý cũng thực hiện được nhưng phải thực hiện 2 lần truy xuất và sẽ làm chậm quá trình xử lý chương trình.

Với vi xử lý 8088 thì do bus dữ liệu bên ngoài chỉ có 8 bit nên chỉ có 1 dãy bộ nhớ và chỉ truy xuất byte – điều này cho phép vi xử lý chỉ thực hiện 2 lần đọc (mỗi lần 1 byte) cho dù địa chỉ của dữ liệu 16bit có địa chỉ chẵn hay lẻ.



Hình 3-6. Tổ chức bộ nhớ của vi xử lý 8086.

#### Các ưu điểm của bộ nhớ phân đoạn:

Với kiểu tổ chức bộ nhớ theo đoạn bao gồm đoạn bộ nhớ chứa mã lệnh, đoạn bộ nhớ chứa dữ liệu, đoạn bộ nhớ chứa dữ liệu mở rộng, đoạn bộ nhớ làm ngăn xếp để chứa các dữ liệu tạm thời, các bộ nhớ đoạn này hoàn toàn có thể độc lập với nhau.

Với kiểu tách độc lập này cho phép chương trình có thể xử lý nhiều đoạn dữ liệu khác nhau bằng cách chỉ cần thay đổi giá trị của thanh ghi DS để trỏ đến vùng dữ liệu mới.

Một ưu điểm lớn nhất với kiểu bộ nhớ phân đoạn là các chương trình sử dụng kiểu định địa chỉ tham chiếu có thể nạp và chạy ở bất kỳ vị trí nào trong bộ nhớ. Điều này thực hiện được là do các địa chỉ logic luôn trải từ địa chỉ 00000H đến FFFFFH không phụ thuộc địa chỉ của đoạn.

Một chương trình chưa thực thi được cất tạm trên đĩa từ và khi muốn thực thi thì được nạp vào bộ nhớ mà không cần phải quan tâm đến địa chỉ vật lý của chương trình trong bộ nhớ hệ thống. Các chương trình như thế được xem là các chương trình tái định vị được có nghĩa là chúng có thể chạy ở bất kỳ vị trí nào trong bộ nhớ. Các yêu cầu để có một chương trình tái định vị được là không có sử dụng các tham chiếu địa chỉ vật lý và không làm thay đổi các thanh ghi đoạn cho phép.

#### Các khuyết điểm của bộ nhớ phân đoạn:

Phức tạp phần cứng: do địa chỉ cần 2 thanh ghi: 1 thanh ghi segment và 1 thanh offset.

Phức tạp phần mềm: các chương trình bị giới hạn kích thước chỉ nằm trong khoảng 64Kbyte – từ vi xử lý 80386 trở đi các đoạn có dung lượng 4Gbyte bao gồm toàn bộ dung lượng bộ nhớ – không cho phép phân đoạn bộ nhớ.

#### 4. RESET VI XỬ LÝ:

Khi mở máy hoặc khi reset vi xử lý 8086/88 bắt đầu thực thi lệnh tại địa chỉ FFFF:0000H. Các thanh ghi được thiết lập các giá trị như ở bảng 3-4:

Các thanh ghi	Giá trị
Thanh ghi cờ	Tất cả đều bị xoá
IP	0000H
CS	FFFFH
DS	0000H
SS	0000H
ES	0000H

Bảng 3-4. Trạng thái của các thanh ghi khi vi xử lý bị reset.

Trong các máy vi tính, địa chỉ logic FFFF:0000H chứa mã của lệnh nhảy JUMP để nhảy đến lệnh thứ nhất trong chương trình BIOS còn gọi là POST – Power On Self Test – tự kiểm tra khi bắt đầu mở máy để kiểm tra và khởi động phần cứng.

#### 5. TÓM TẮT PHẦN CỨNG VI XỬ LÝ 8086/88:

##### a. Khảo sát đặc tính điện:

Sơ đồ chân của vi xử lý 8086 như hình 3-7.

Do vi xử lý 8086 có bus dữ liệu bên ngoài 16 bit còn vi xử lý 8088 thì bus dữ liệu bên ngoài chỉ có 8 bit nên sơ đồ chân của chúng khác nhau nên không thể thay thế chúng trong mạch điện.

##### **Các yêu cầu về nguồn điện:**

Vi xử lý 8086 sử dụng nguồn +5V cho phép sai số  $\pm 10\%$ .

Dòng điện tiêu thụ với phiên bản NMOS thì vi xử lý 8086 dòng làm việc khoảng 360mA, vi xử lý 8088 dòng làm việc khoảng 340mA.

Với phiên bản CMOS thì dòng tiêu thụ khoảng 10mA.

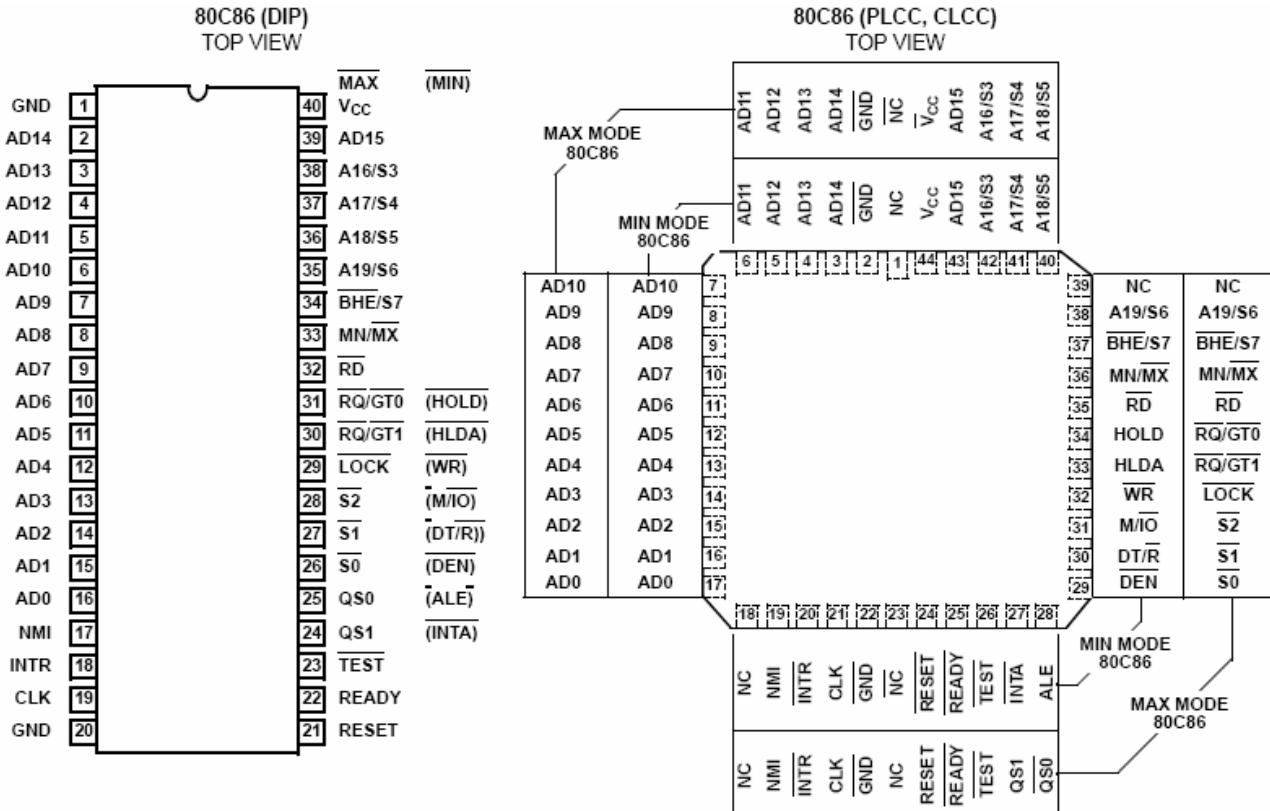
Tâm nhiệt độ làm việc với phiên bản thương mại từ 0 đến 70°C.

##### **Đặc tính của các chân:**

Đặc tính mức logic ngõ vào DC:

	Mức logic	Điện áp	Dòng điện
Đặc tính ngõ vào	0	-0.5V (max) ÷ -0.8V (max)	10µA (max)
	1	2V(min) +0.5V (max)	10µA (max)

<b>Đặc tính ngõ ra</b>	<b>0</b>	<b>0.45V (max)</b>	<b><math>I_{OL} = 2.5\text{mA}</math> (ở 0.45V)</b>
	<b>1</b>	<b>2.4V(max)</b>	<b><math>I_{OH} = -400\mu\text{A}</math> (ở 2.4V)</b>
	<b>High Z</b>	<b><math>0.45V \leq V_{out} \leq V_{cc}</math></b>	<b><math>I_{IO} = \pm 10\mu\text{A}</math> (max)</b>

**Bảng 3-5. Đặc tính điện.****Hình 3-7. Sơ đồ chân vi xử lý 8086 và vi xử lý 8088.****Những khác biệt với TTL:**

- Mức logic 0 của vi xử lý 8086 là 0.45V
- Mức logic 0 của TTL là 0.4V.
- Miễn nhiễu của TTL là  $0.8V - 0.4V = 0.4V$
- Miễn nhiễu của vi xử lý 8086 là  $0.8V - 0.45V = 0.35V$  nên miễn nhiễu bị giảm.

**Tương thích với các họ logic:**

Vi xử lý 8086 tương thích hầu hết với các họ logic.

**b. Mô tả chân của vi xử lý 8086 :**

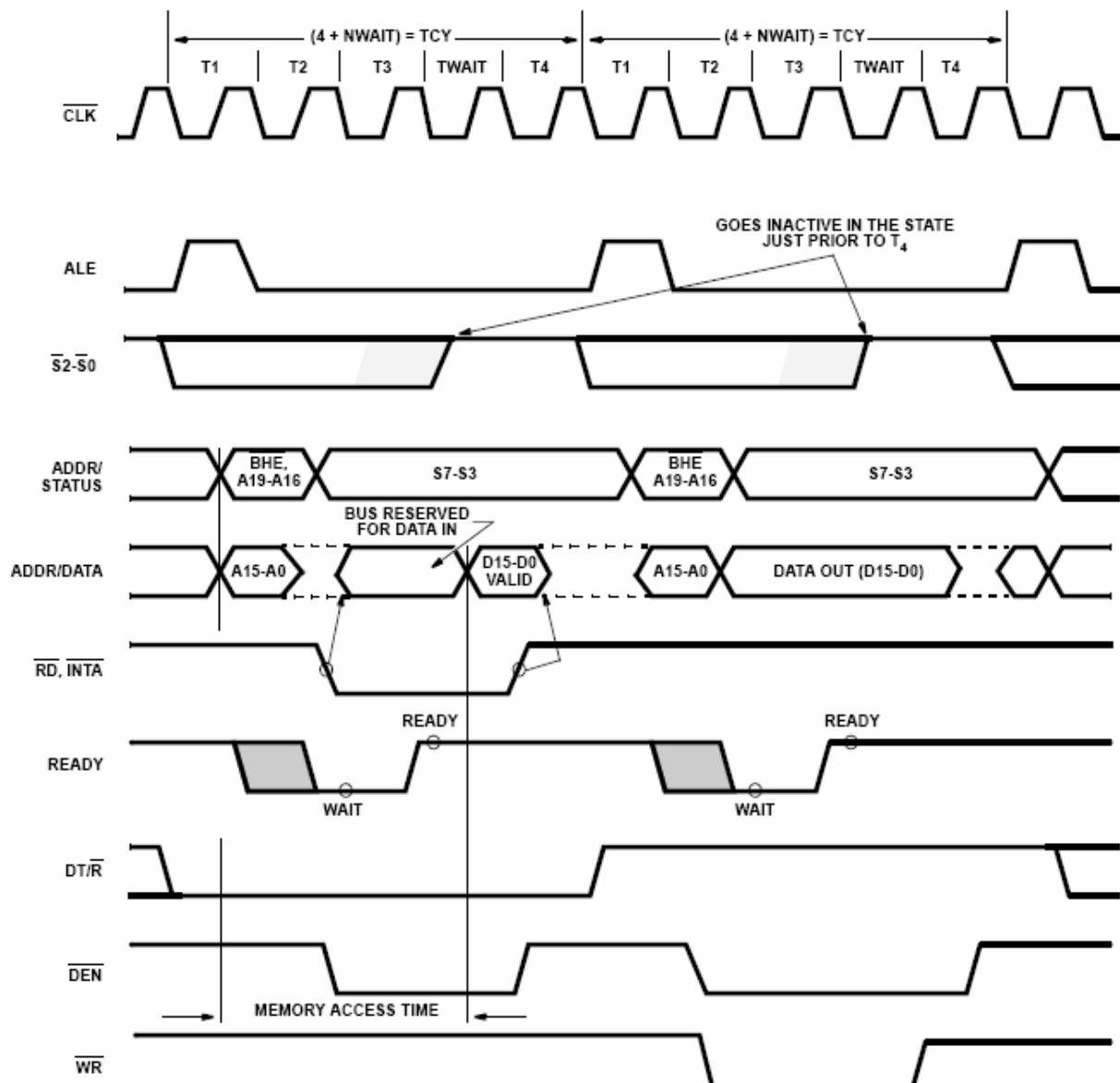
Vi xử lý 8086 có một bus địa chỉ 20 bit, bus dữ liệu 16 bit, 3 chân nguồn và 17 chân còn lại dùng cho các chức năng điều khiển và định thời. Tất cả các chân này được chia ra làm 3 nhóm bus: bus địa chỉ, bus dữ liệu và bus điều khiển.

Bus dữ liệu 16 đường được đa hợp với 16 đường địa chỉ thấp của bus địa chỉ để làm giảm bớt số chân cho IC. Hình 3-7 trình bày sơ đồ chân của vi xử lý 8086 với 2 chế độ hoạt động tối thiểu (Minimum Mode) và chế độ hoạt động tối đa (Maximum Mode). Ở chế độ hoạt động tối

đa, vi xử lý sẽ kết nối với bộ điều khiển bus 8288 để tạo các tín hiệu điều khiển và có thể làm việc trong hệ thống có nhiều vi xử lý.

Các nhà thiết kế vi xử lý thường sử dụng kỹ thuật đa hợp hay dồn kênh theo thời gian để cho phép 1 chân có thể thực hiện nhiều chức năng như 16 đường AD0 ÷ AD15. Khi truy xuất bộ nhớ thì 16 đường AD0 ÷ AD15 được phân chia theo thời gian như sau: trong khoảng thời gian T1 chúng đóng vai trò là 16 đường địa chỉ hay có chức năng tải địa chỉ, trong khoảng thời gian còn lại từ T2 đến T4, chúng đóng vai trò là 16 đường dữ liệu hay có chức năng tải dữ liệu. Khi thực hiện chức năng tải dữ liệu thì địa chỉ xuất ra ở thời gian T1 không còn nữa nên phải dùng mạch phân kênh để tách và lưu lại địa chỉ.

Hình 3-8 trình bày giàn đồ thời gian làm truy xuất bộ nhớ của vi xử lý và giải thích hoạt động của từng chu kỳ.



Hình 3-8. Giản đồ thời gian truy xuất bộ nhớ của vi xử lý 8086.

+ Ở chu kỳ T1:

Trong chu kỳ này, địa chỉ của bộ nhớ hay thiết bị ngoại vi đưa ra trên các đường địa chỉ. Các tín hiệu điều khiển ALE,  $DT/R$ ,  $M/IO$  cũng được đưa ra để hỗ trợ các vi mạch hoàn tất việc gởi thông tin địa chỉ này.

#### + Ở chu kỳ T2:

Trong chu kỳ này, vi xử lý đưa ra các tín hiệu điều khiển  $RD$  hoặc  $WR$ ,  $DEN$  và tín hiệu dữ liệu trên D0 ÷ D15 nếu là lệnh ghi.  $DEN$  thường dùng để mở các bộ đệm của bus dữ liệu nếu chúng được dùng trong hệ thống. Tại cuối chu kỳ T2 thì vi xử lý lấy mẫu tín hiệu Ready để xử lý trong chu kỳ tiếp theo khi nó phải đợi khi làm việc với bộ nhớ hay thiết bị ngoại vi có tốc độ chậm hay không.

#### + Ở chu kỳ T3:

Trong chu kỳ này vi xử lý dành thời gian cho bộ nhớ hay thiết bị ngoại vi truy cập dữ liệu. Nếu là chu kỳ đọc dữ liệu thì tại cuối T3 vi xử lý sẽ lấy mẫu tín hiệu của bus dữ liệu. Nếu tại cuối chu kỳ T2 mà vi xử lý phát hiện ra tín hiệu Ready = 0 thì vi xử lý tự xen vào T3 một chu kỳ đợi Twait để tạo chu kỳ đợi  $T_w = n * T$  nhằm kéo dài thời gian thực hiện lệnh, tạo điều kiện cho bộ nhớ và thiết bị ngoại vi tốc độ chậm có đủ thời gian hoàn tất công việc đọc-ghi dữ liệu.

#### + Ở chu kỳ T4:

Trong chu kỳ này, các tín hiệu trên bus được đưa về trạng thái không tích cực để chuẩn bị cho chu kỳ đọc/ghi mới. Tín hiệu  $WR$  trong khi chuyển trạng thái từ ‘0’ lên ‘1’ sẽ kích hoạt trạng thái ghi vào bộ nhớ hay thiết bị ngoại vi.

### **Chức năng các chân của vi xử lý 8086:**

- **Bus dữ liệu AD0 ÷ AD15:** 16 chân này là bus dữ liệu 2 chiều trong khoảng thời gian ở các chu kỳ từ T2 đến T4. Trong khoảng thời gian T1, 16 chân này có chức năng tải địa chỉ 16 bit của bộ nhớ hoặc của IO.

- **Bus địa chỉ AD0 ÷ AD15 và A16/S3 ÷ A19/S6:** 20 chân này tương ứng với bus địa chỉ 20 bit cho phép truy xuất 1048576 ô nhớ. Các đường địa chỉ là các ngõ ra chỉ xuất hiện trong khoảng thời gian T1.

- **Chân cho phép chốt địa chỉ ALE (Address Latch Enable):** tín hiệu ra ở chân ALE được dùng để tách địa chỉ và dữ liệu ở các đường AD0 ÷ AD15, tách địa chỉ và trạng thái ở các đường A16/S3 ÷ A19/S6 và tách  $BHE/S7$ . Hình 3-8 có trình bày dạng sóng của tín hiệu xung ALE cho các chu kỳ đọc và ghi bộ nhớ. Mọi chu kỳ đọc hay ghi luôn bắt đầu bằng 1 xung nhịp ALE xuất hiện trong khoảng thời của chu kỳ T1. Địa chỉ 20 bit phải ổn định khi có xung ALE chuyển trạng thái từ mức 1 xuống mức 0 gần cuối chu kỳ T1 – xung ALE tích cực cạnh xuống. Xung ALE được dùng để điều khiển mạch chốt để chốt lại địa chỉ.

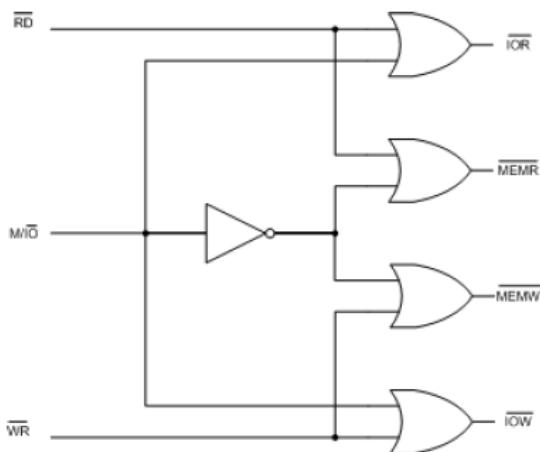
- **Tín hiệu điều khiển bộ nhớ/ ngoại vi M/IO (memory/IO):** tín hiệu ra này có chức năng báo cho biết vi xử lý đang truy xuất bộ nhớ thì chân này  $M/IO = 1$  và khi vi xử lý truy xuất IO thì  $M/IO = 0$ .

- **Tín hiệu điều khiển đọc bộ nhớ/ ngoại vi RD (Read):** bình thường chân này ở mức logic 1, khi vi xử lý đọc dữ liệu từ bộ nhớ hoặc từ thiết bị ngoại vi IO thì vi xử lý sẽ điều khiển chân  $RD$  xuống mức logic 0 để điều khiển bộ nhớ hoặc IO xuất dữ liệu, sau đó vi xử lý cho nó

trở lại mức 1 để kết thúc quá trình đọc dữ liệu. Trong giản đồ thời gian ta thấy chân  $\overline{RD}$  xuống mức 0 ở thời điểm cuối chu kỳ T2 và trở lại mức 1 ở chu kỳ T4 .

- **Tín hiệu điều khiển ghi bộ nhớ/ ngoại vi WR (Write):** bình thường chân này ở mức logic 1, khi vi xử lý muốn ghi dữ liệu vào bộ nhớ hoặc vào thiết bị ngoại vi IO thì vi xử lý sẽ điều khiển chân  $\overline{WR}$  xuống mức logic 0 để điều khiển bộ nhớ hoặc IO nhận dữ liệu, sau đó vi xử lý cho nó trở lại mức 1 để kết thúc quá trình ghi dữ liệu. Trong giản đồ thời gian ta thấy chân  $\overline{WR}$  xuống mức 0 ở thời điểm cuối chu kỳ T2 và trở lại mức 1 ở chu kỳ T4.

Các chân điều khiển  $\overline{RD}$  và  $\overline{WR}$  thường kết hợp với chân  $M/\overline{IO}$  qua các cổng logic để tạo ra các đường điều khiển ghi đọc bộ nhớ và ghi đọc IO một cách độc lập như hình 3-9.



Hình 3-9. Tạo các tín hiệu ghi đọc bộ nhớ và IO.

- **Tín hiệu xung nhịp clock:** tất cả các hoạt động trong vi xử lý được đồng bộ với xung clock được đưa đến chân CLK. Bảng sau cho biết tần số hoạt động tối đa là 10MHz đối với vi xử lý 8086-1.

Vị trí xử lý	Tần số (MHz)	Dòng điện $I_{CC}$ (max)(mA)	Công suất tiêu tán (W)
8086	5	340	1,7
8086 - 1	8	350	1,75
8086 - 2	10	360	1,8
8088	5	340	1,7
8088 - 2	8	350	1,75
P8088	5	250	1,25

Bảng 3-6. Đặt tính tốc độ và công suất tiêu tán.

- **Tín hiệu BHE** (Bus High Enable): tín hiệu này được vi xử lý xuất ra trong khoảng thời gian của chu kỳ T1. Khi BHE ở mức thấp sẽ xác định bus AD0 ÷ AD15 liên quan đến chuyển dữ liệu có thể xảy ra đổi với việc truy xuất bộ nhớ hoặc IO hoặc truy xuất một byte dữ liệu ở địa chỉ lẻ. Tín hiệu BHE và đường địa chỉ A0 thường được dùng để chọn bank bộ nhớ chẵn hay lẻ hoặc các IO chẵn hay lẻ được liệt kê ở bảng sau:

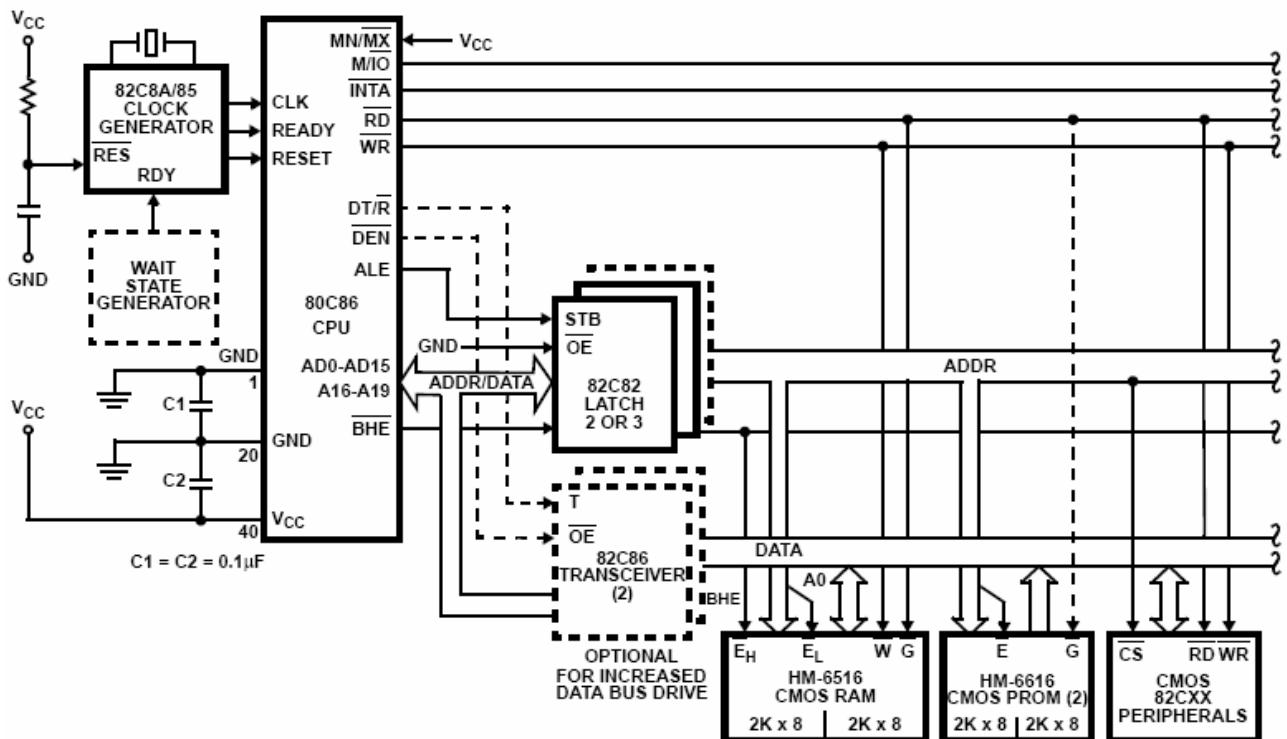
$\overline{BHE}$	A0	Trạng thái
0	0	Truy xuất từ dữ liệu 16 bit
0	1	Truy xuất byte dữ liệu lẻ ( $D_8 \div D_{15}$ )
1	0	Truy xuất byte dữ liệu chẵn ( $D_0 \div D_7$ )
1	1	Không tác động

Bảng 3-7. Mã hoá truy xuất bộ nhớ của vi xử lý 8086.

• **Tín hiệu truyền/nhận dữ liệu DT /  $\overline{R}$**  (Data Transmit/Receive): tín hiệu này dùng để điều khiển hướng vào/ra của dữ liệu qua các bộ đệm vào bus dữ liệu hệ thống của vi xử lý. Khi tín hiệu này ở mức thấp thì vi xử lý đang tiến hành đọc dữ liệu – khi đó tín hiệu này sẽ điều khiển các mạch đệm mở cổng đệm dữ liệu từ bộ nhớ hay IO qua bộ đệm rồi đưa đến vi xử lý, khi tín hiệu này ở mức cao thì vi xử lý đang tiến hành ghi dữ liệu – khi đó tín hiệu này sẽ điều khiển các mạch đệm mở cổng đệm - dữ liệu từ vi xử lý qua bộ đệm rồi đưa đến bộ nhớ hay IO.

• **Tín hiệu cho phép ghi dữ liệu DEN** (Data Enable): tín hiệu này cùng với tín hiệu DT /  $\overline{R}$  để cho phép các bộ đệm 2 chiều được nối đến bus dữ liệu của hệ thống để ngăn chặn việc tranh chấp bus (hai mạch cùng điều khiển một đường bus) bằng cách cấm các bộ đệm dữ liệu cho đến chu kỳ T2 – khi đó bus địa chỉ/dữ liệu không còn tải địa chỉ bắt đầu tải dữ liệu.

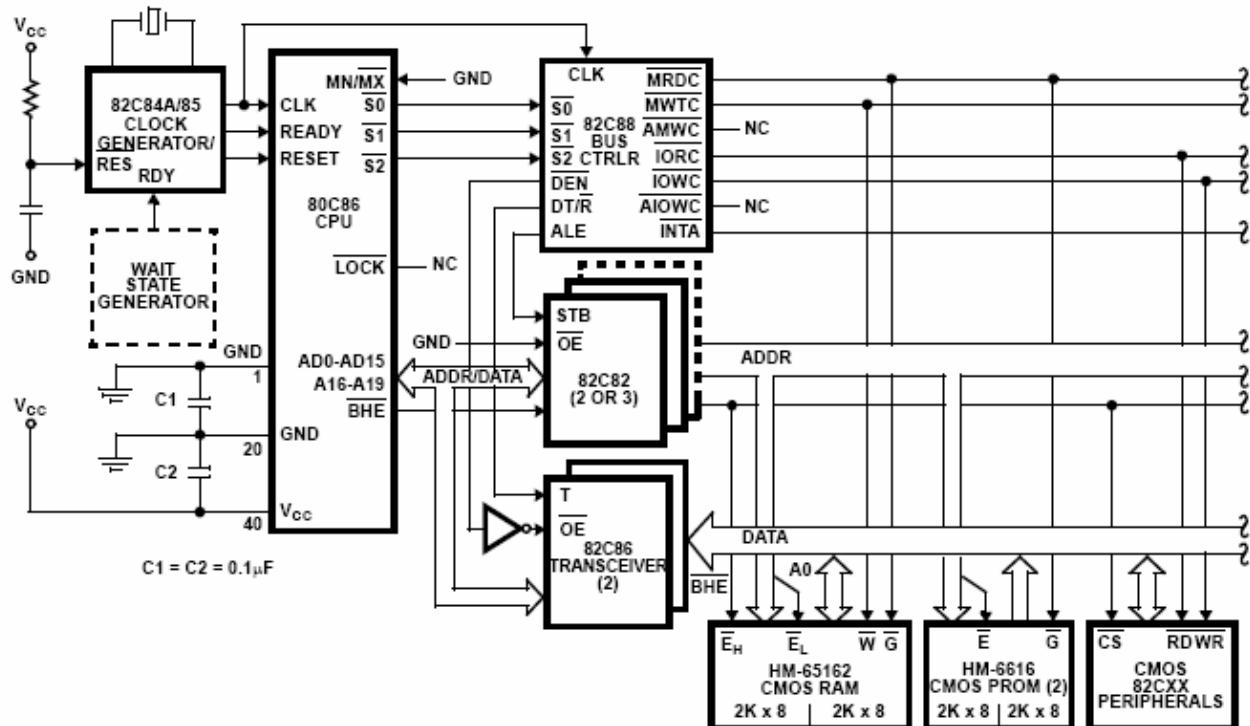
• **Tín hiệu chọn chế độ tối thiểu/tối đa MN /  $\overline{MX}$**  (Minimum/Maximum Mode): Chức năng của các chân từ 24 đến 32 thay đổi phụ thuộc vào mức logic đưa đến chân này. Nếu  $MN / \overline{MX} = 0$  thì vi xử lý 8086 hoạt động ở mode tối đa, ngược lại thì hoạt động ở mode tối thiểu. Hình 3-9 minh họa vi xử lý hoạt động ở mode tối thiểu và hình 3-10 minh họa vi xử lý hoạt động ở mode tối đa:



Hình 3-10. Vi xử lý hoạt động ở chế độ tối thiểu.

• **Tín hiệu reset:** khi ngõ vào này có xung ở mức cao sẽ reset vi xử lý dẫn đến kết thúc hoạt động hiện tại và bắt đầu thực hiện chương trình tại địa chỉ bắt đầu FFFF:0000H như đã trình bày. Trạng thái reset được tác động khi mới bắt đầu cấp điện cho hệ thống hoặc khi có sự cố hoạt động của hệ thống.

• **Tín hiệu test:** ngõ vào này được dùng cùng với lệnh wait: nếu ngõ vào test ở mức cao khi gặp lệnh wait trong chương trình đang xử lý thì vi xử lý sẽ ngừng chương trình đang thực hiện và chuyển sang chế độ nghỉ (*idle mode*) và khi ngõ vào test trở lại mức thấp thì vi xử lý sẽ tiếp tục thực hiện lệnh tiếp theo sau lệnh wait. Thường thì ngõ vào này được điều khiển bởi IC toán học 8087.



Hình 3-11. Vi xử lý hoạt động ở chế độ tối đa.

• **Tín hiệu Ready:** ngõ vào này được vi xử lý lấy mẫu ở cạnh lên của xung nhịp T2: nếu tín hiệu này được phát hiện đang ở mức cao thì vi xử lý sẽ kết thúc quá trình đọc hoặc ghi đúng 4 chu kỳ xung nhịp từ T1 ÷ T4, nếu tín hiệu này được phát hiện đang ở mức thấp thì vi xử lý sẽ thêm vào các chu kỳ đợi (wait) ngay sau chu kỳ T3 (hãy xem giản đồ tín hiệu) cho đến khi vi xử lý phát hiện tín hiệu ready trở lại mức cao thì thực hiện tiếp chu kỳ còn lại T4 và chấm dứt quá trình xử lý.

• **Các tín hiệu ngắt interrupt (INTR, NMI và INTA):** các tín hiệu ngõ vào ngắt INTR và NMI là các yêu cầu ngắt được tác động bằng phần cứng (tín hiệu điện). Tín hiệu ngắt NMI tích cực khi có cạnh lên của tín hiệu, INTR tích cực mức cao. Ngắt INTR có thể điều khiển cho phép hoặc không cho phép bằng phần mềm, ngắt NMI là ngắt không thể che được nên luôn luôn được phục vụ khi tác động. Ngắt NMI được dùng xử lý các sự kiện như hỏng nguồn hoặc các lỗi bộ nhớ.

Khi NMI tác động thì vi xử lý tự động chuyển điều khiển đến địa chỉ được lưu sẵn ở các ô nhớ có địa chỉ 00008 ÷ 0000BH.

Khi INTR tác động thì chu kỳ ghi nhận ngắt được thực hiện. Nếu được phép thì vi xử lý sẽ trả lời bằng cách tác động đến ngõ ra INTA báo cho thiết bị báo ngắt biết vi xử lý đã sẵn sàng đáp ứng, thiết bị yêu cầu ngắt sẽ gởi địa chỉ 8 bit vào bus dữ liệu thấp rồi chuyển đến địa chỉ được chứa ở các vị trí (địa chỉ  $\times 4$ ) đến (địa chỉ  $\times 4 + 3$ ).

- **Các tín hiệu HOLD và HLDA:** hold là tín hiệu ngõ vào tác động mức cao làm cho vi xử lý treo tất cả các đường bus ở trạng thái tổng trở cao hay có thể xem như tất cả hở mạch để cho phép vi xử lý khác truy cập đến bộ nhớ hay các thiết bị ngoại vi IO. Quá trình này được xem như là “truy xuất bộ nhớ trực tiếp” DMA, tín hiệu HLDA báo cho thiết bị yêu cầu vi xử lý ngừng biết vi xử lý đã vào trạng thái ngừng để thiết bị bắt đầu được phép truy xuất bộ nhớ hoặc IO. Sau khi truy xuất xong thiết bị đó phải làm chân Hold xuống mức 0 để vi xử lý thực hiện tiếp.

- **Nguồn Vcc và mass GND:** vi xử lý 8086 sử dụng nguồn +5V và có 2 chân GND.

#### c. Tạo xung clock và mạch reset vi xử lý 8086:

Như ta đã biết vi xử lý cũng là 1 IC số nên cần phải có xung clock để điều khiển tất cả các hệ thống làm việc.

Vi xử lý 8086 cần xung clock có thời gian lên nhanh và thời gian xuống nhanh nhỏ hơn 10ns, các mức logic 0 và 1 nằm trong khoảng từ  $-0.5V \div 0.6V$  và từ  $3.9V \div 5.0V$  và có hệ số công tác là 33%.

Tín hiệu reset của vi xử lý phải được đồng bộ với xung nhịp hệ thống và tồn tại ít nhất trong 4 chu kỳ trạng thái T.

Do các yêu cầu nghiêm ngặt của xung clock nên Intel đã chế tạo luôn mạch tạo xung nhịp dùng chip 8284A. Ngoài chức năng thực hiện các yêu cầu của xung clock thì vi mạch này còn có chức năng đồng bộ các yêu cầu đợi từ các bộ nhớ có tốc độ chậm.

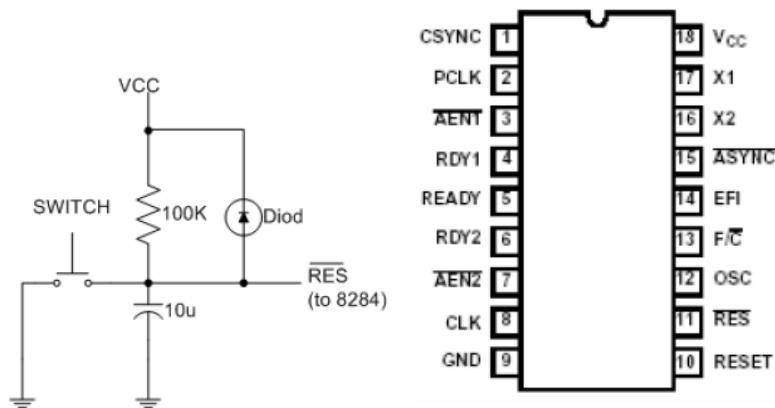
Tất cả các hoạt động của vi xử lý 8086 đều tuân tự và đồng bộ với tín hiệu xung nhịp của hệ thống. Trong khoảng thời gian T1 của clock thì vi xử lý xuất địa chỉ của bộ nhớ hoặc của IO, trong khoảng thời gian T2 thì vi xử lý xuất các tín hiệu điều khiển để tác động đến bộ nhớ hoặc IO, trong khoảng thời gian T3 thì vi xử lý chờ bộ nhớ hoặc IO đáp ứng xuất hoặc nhận dữ liệu và ở khoảng thời gian T4 thì vi xử lý kết thúc hoạt động.

Nếu không có tín hiệu xung clock để đồng bộ các hoạt động thì hệ thống sẽ chạy sai. Các thiết bị bộ nhớ và IO cần có thời gian thiết lập và thời gian giữ được cung cấp bởi chu kỳ bus cơ bản gồm 4 xung clock của vi xử lý.

(Thời gian thiết lập và thời gian giữ của bộ nhớ chính là khoảng thời gian bộ nhớ từ khi nhận địa chỉ cho đến khi giải mã tìm đúng ô nhớ và thực hiện quá trình xuất hoặc ghi dữ liệu).

Xung clock kết hợp với các cổng logic để làm tươi các bộ nhớ DRAM khỏi bị mất dữ liệu.

Mạch tạo tín hiệu reset như hình 3-12:

**Hình 3-12. Mạch tạo tín hiệu reset.**

Mạch tạo tín hiệu reset cho phép reset bằng 2 cách: reset khi bắt đầu cấp điện cho hệ thống và reset khi nhấn nút reset.

Khi cấp điện thì tụ điện được nạp điện từ 0v đến 5v. Với thời gian thích hợp ở ngõ vào RES được duy trì ở mức thấp đủ thời gian cần thiết để đảm bảo reset vi xử lý.

Khi ta nhấn nút reset làm cho ngõ vào RES được duy trì ở mức thấp sau một khoảng thời gian làm tụ xả hết điện và khi ta buông tay ra thì tụ được nạp điện trở lại và mạch reset hoạt động giống như khi tự động reset khi cấp điện.

Thời gian RC nên chọn để được thời gian tích cực reset tối thiểu là 50μs. Diode có trong mạch có chức năng xả điện cho tụ điện trở lại nguồn cung cấp khi tắt nguồn cung cấp cho hệ thống.

## II. CÁC PHƯƠNG PHÁP ĐỊNH ĐỊA CHỈ:

Các lệnh của vi xử lý có chức năng xử lý dữ liệu và dữ liệu thì được cất ở trong bộ nhớ hoặc ở thiết bị IO hoặc ở trong các thanh ghi của vi xử lý. Cách định địa chỉ là cung cấp địa chỉ của dữ liệu cho vi xử lý biết để tiến hành lấy dữ liệu để thực thi lệnh.

Vi xử lý 8086 có 9 cách định địa chỉ gồm:

- Định địa chỉ tức thời.
- Định địa chỉ thanh ghi.
- Định địa chỉ trực tiếp.
- Định địa chỉ gián tiếp.
- Định địa chỉ chỉ số.
- Định địa chỉ có nền.
- Định địa chỉ có nền và có chỉ số.
- Định địa chỉ có nền và có chỉ số kết hợp với độ dời.
- Định địa chỉ kiểu chuỗi.

Trình biên dịch assembler sẽ xác định địa chỉ nào đang được sử dụng trong lệnh thông qua cú pháp của vùng toán hạng.

### 1. Cách định địa chỉ dùng thanh ghi:

Với cách định địa chỉ dùng thanh ghi thì tác tổ cần truy xuất thường là dữ liệu lưu trong thanh ghi của vi xử lý.

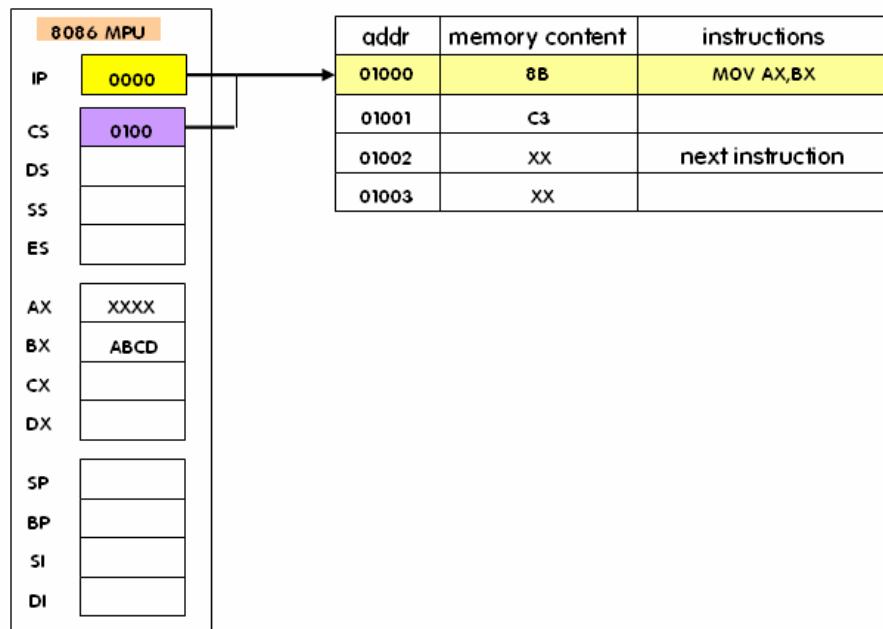
**Ví dụ 1:** lệnh **MOV AX,BX**

Lệnh này sẽ thực hiện việc copy dữ liệu lưu trong thanh ghi BX (dữ liệu nguồn – source operand) sang thanh ghi DX (dữ liệu đích– destination operand).

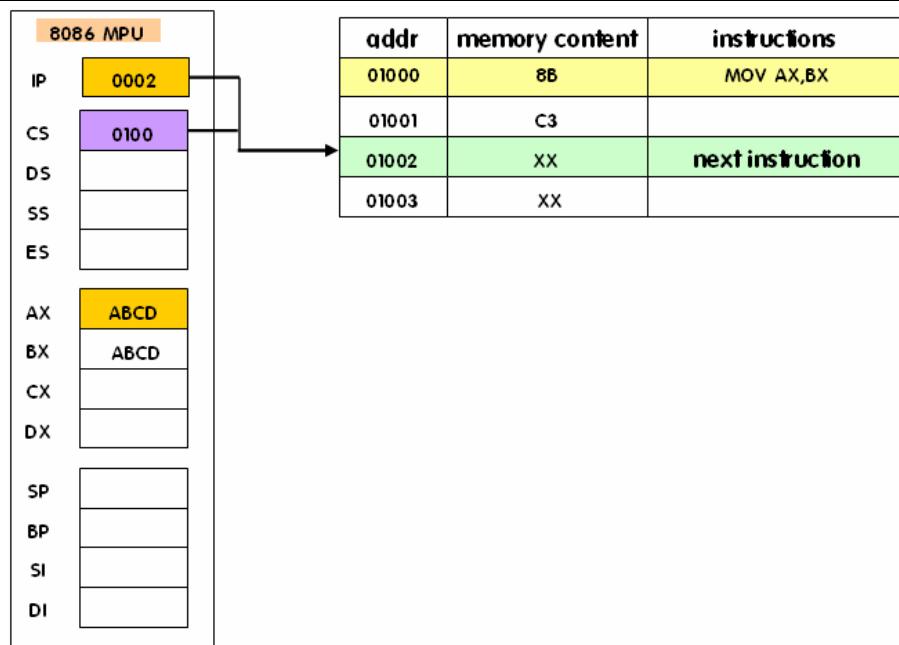
Hình 3-13a và 3-13b sẽ minh họa quá trình vi xử lý trước và sau khi thực hiện lệnh.

Nội dung của thanh ghi AX trước khi thực hiện là xxxx, nội dung thanh ghi BX bằng ABCD và sau khi thực hiện thì nội dung thanh ghi AX bằng ABCD.

Lệnh 2 byte nên sau khi thực hiện xong con trỏ lệnh IP tăng lên 2 để trỏ đến lệnh kế.



Hình 3-13a. Trước khi thực hiện lệnh **MOV AX, BX**.

**Hình 3-13b. Sau khi thực hiện lệnh **MOV AX, BX**.**

## 2. Cách định địa chỉ tức thời:

Các lệnh dùng địa chỉ tức thời xem dữ liệu là một phần của lệnh. Trong cách định địa chỉ tức thời thì dữ liệu được đặt trong lệnh.

### Ví dụ2: Lệnh **MOV AL,15H**

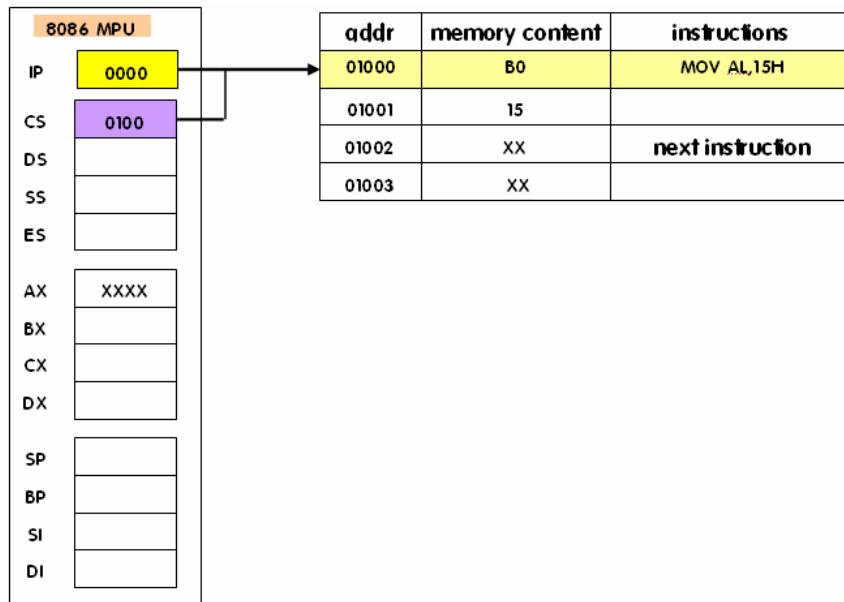
Lệnh có chức năng nạp dữ liệu 15H vào thanh ghi AL và quá trình được thực hiện như hình 3-14a và 3-14b.

Cách định địa chỉ tức thời dùng để nạp 1 dữ liệu vào 1 thanh ghi hoặc nạp dữ liệu vào ô nhớ. Các lệnh này không thể áp dụng cho các thanh ghi đoạn.

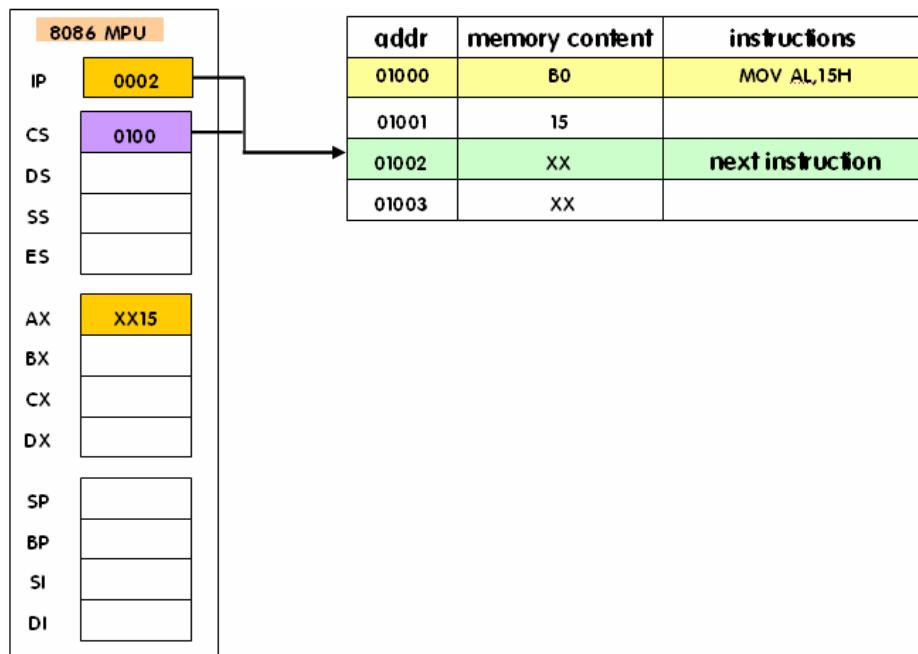
Các hằng số toán hạng có chiều dài là 1 byte hoặc 2 byte.

Trước khi thực hiện lệnh thì nội dung thanh ghi AX là xxxx, sau khi thực hiện lệnh thì nội dung thanh ghi AX là xx15.

Lệnh 2 byte nên sau khi thực hiện thì nội dung thanh ghi IP tăng lên 2 để chuẩn bị thực hiện lệnh kế.



Hình 3-14a. Trước khi thực hiện lệnh MOV AL,15H.



Hình 3-14b. Sau khi thực hiện lệnh MOV AL,15H.

### 3. Cách định địa chỉ trực tiếp:

Cách định địa chỉ gián tiếp khác với cách định địa chỉ trực tiếp là các ô nhớ đi sau mã lệnh là địa chỉ hiệu dụng *effective memory address* (EA) thay vì dữ liệu. EA là địa chỉ offset 16 bit sẽ kết hợp với thanh ghi DS để tạo ra địa chỉ vật lý của ô nhớ cần truy xuất dữ liệu.

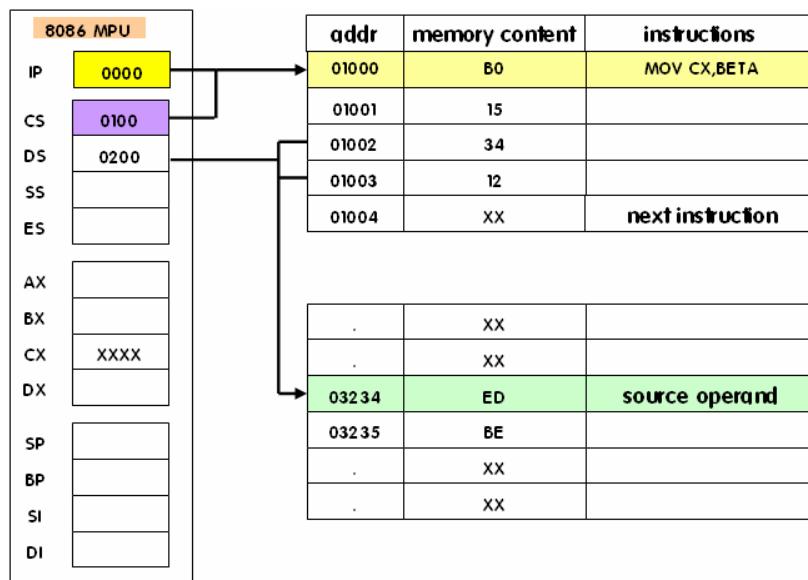
#### Ví dụ 3: Lệnh **MOV CX, BETA**

Lệnh có chức năng “copy dữ liệu trong ô nhớ có địa chỉ offset được kí hiệu là BETA trong đoạn bộ nhớ dữ liệu vào thanh ghi CX” và quá trình được thực hiện như hình 3-15a và 3-15b.

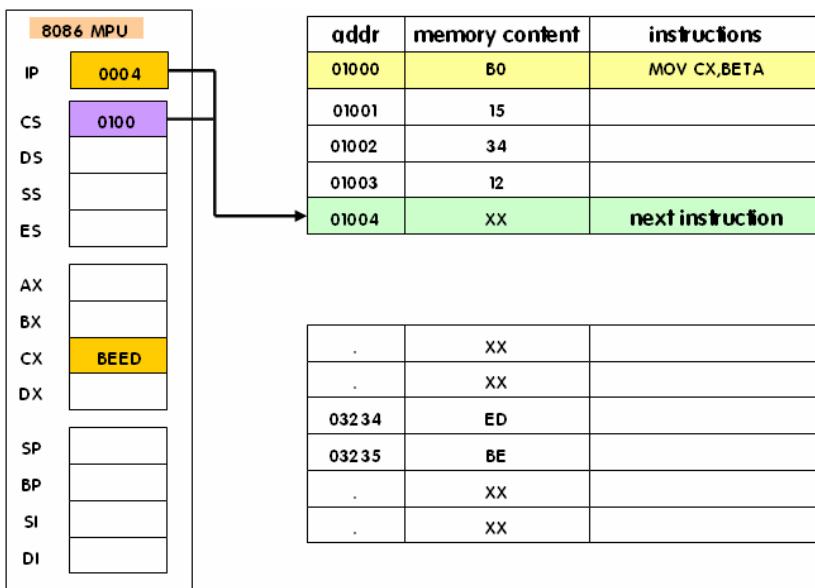
Trong lệnh này thì địa chỉ offset bằng 1234H nằm sau 2 byte có mã lệnh B0 và 15H. Địa chỉ offset này được cộng thêm địa chỉ của thanh ghi DS hiện hành sau khi nhân với 16 như sau:

$$PA = 02000H + 1234H = 03234H$$

Chính là địa chỉ của ô nhớ cần truy xuất dữ liệu.



Hình 3-15a. Trước khi thực hiện lệnh `MOV CX,BETA`.



Hình 3-15b. Sau khi thực hiện lệnh `MOV CX,BETA`.

Sau khi thực hiện lệnh thì thanh ghi IP tăng lên 4 đơn vị vì mã của lệnh này bằng 4 byte, dữ liệu trong 2 ô nhớ liên tiếp có địa chỉ 03234H và 03235H được copy vào thanh ghi CX theo thứ tự byte thấp trước byte cao sau.

Kết quả thanh ghi CX = BEEDH.

#### 4. Các cách định địa chỉ gián tiếp:

Cách định địa chỉ trực tiếp thì sử dụng rất tiện lợi cho các trường hợp truy cập bộ nhớ không thường xuyên vùng dữ liệu truy cập chỉ có 1 hoặc 2 byte hoặc word. Trong trường hợp vùng dữ liệu nhiều và liên tục thì cách dùng địa chỉ trực tiếp là không hiệu quả ví dụ copy dữ liệu từ vùng nhớ này sang vùng nhớ khác với dung lượng dữ liệu là vài trăm byte.

Cách thực hiện tốt nhất là kiểu định địa chỉ gián tiếp dùng thanh ghi, trong đó địa chỉ của nguồn dữ liệu do 1 thanh ghi quản lý và địa chỉ đích của nguồn dữ liệu do 1 thanh ghi khác quản lý và điều này là phải kết hợp với thanh ghi đoạn. Ngoài ra độ dời bù 2 có thể được cộng vào thanh ghi con trả và thanh ghi chỉ số để dời đi so với vị trí được chỉ đến. Bảng sau trình bày nhiều kết hợp có thể sử dụng:

Cách định địa chỉ	Địa chỉ hiệu dụng		
	Độ dời	Thanh ghi nền	Thanh ghi chỉ số
Gián tiếp thanh ghi	Không có + Không có +	BX hoặc BP + Không có +	Không có SI hoặc DI
Có chỉ số	-128 đến 127 +	Không có +	SI hoặc DI
Có nền	-128 đến 127 +	BX hoặc BP +	Không có
Có nền và có chỉ số	Không có +	BX hoặc BP +	SI hoặc DI
Có nền, có chỉ số với độ dời	-128 đến 127 +	BX hoặc BP +	SI hoặc DI

**Bảng 3-8. Tính toán địa chỉ hiệu dụng cho các cách định địa chỉ gián tiếp.**

Tổng quát thì một độ dời có thể được cộng vào thanh ghi nền và cộng với thanh ghi chỉ số để tạo ra địa chỉ vật lý.

Trong lập trình ta có thể sử dụng cách định địa chỉ gián tiếp dưới dạng như sau:

#### **MOV AX, TABLE[SI]**

Trong đó SI là nhãn cho trước của vùng dữ liệu lưu trong bộ nhớ.

##### **a. Định địa chỉ gián tiếp thanh ghi:**

Cách định địa chỉ dùng thanh ghi giống như cách định địa chỉ trực tiếp tuy nhiên địa chỉ offset được lưu trong thanh ghi.

Địa chỉ offset có thể lưu trữ ở các thanh ghi BP, BX, DI hoặc SI. Các dấu ngoặc vuông [ ] dùng để chỉ cách định địa chỉ gián tiếp.

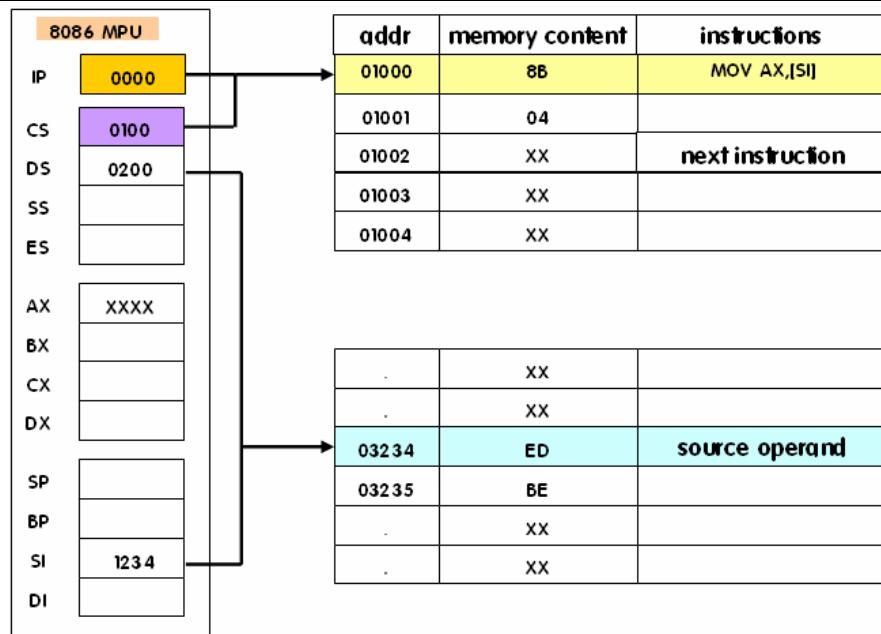
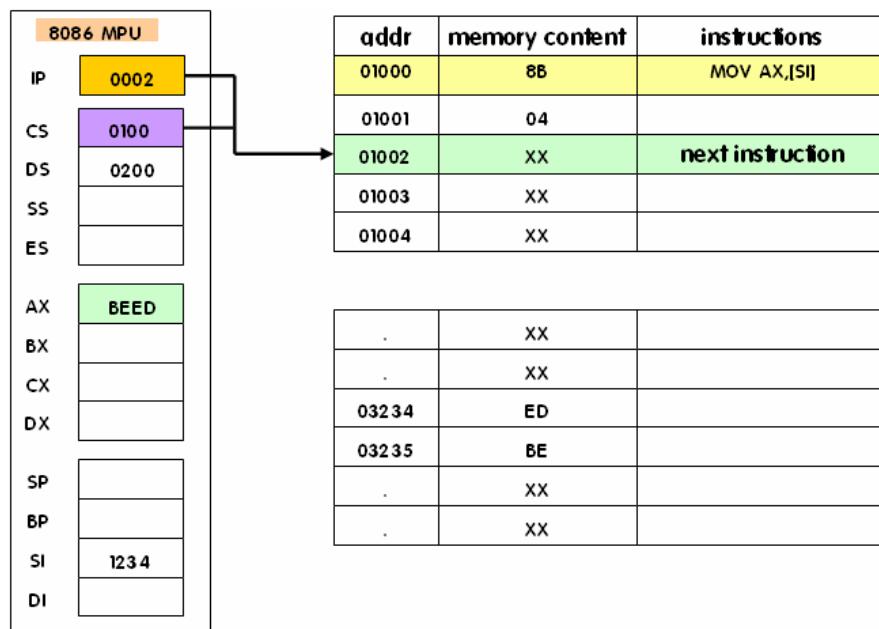
##### **Ví dụ 4: Lệnh MOV AX, [SI]**

Lệnh có chức năng “copy dữ liệu trong ô nhớ có địa chỉ offset lưu trong thanh ghi SI trong đoạn bộ nhớ dữ liệu vào thanh ghi AX” và quá trình được thực hiện như hình 3-16a và 3-16b.

Trong lệnh này thì địa chỉ offset lưu trong SI bằng 1234H nằm trong vi xử lý. Địa chỉ offset này được cộng thêm địa chỉ của thanh ghi DS hiện hành sau khi nhân với 16 như sau:

$$PA = 02000H + 1234H = 03234H$$

Chính là địa chỉ của ô nhớ cần truy xuất dữ liệu.

Hình 3-16a. Trước khi thực hiện lệnh `MOV AX,[SI]`.Hình 3-16b. Sau khi thực hiện lệnh `MOV AX,[SI]`.

Sau khi thực hiện lệnh thì thanh ghi IP tăng lên 2 đơn vị vì mã của lệnh này bằng 2 byte, dữ liệu trong 2 ô nhớ liên tiếp có địa chỉ 03234H và 03235H được copy vào thanh ghi AX theo thứ tự byte thấp trước byte cao sau.

Thanh ghi DS là thanh ghi địa chỉ đoạn dữ liệu mặc nhiên. Trong một số trường hợp không rõ ràng thì trình biên dịch cần thêm chỉ dẫn BYTE PTR hoặc WORD PTR để chỉ kích thước của địa chỉ dữ liệu bằng con trỏ bộ nhớ.

**Ví dụ:** lệnh `MOV [DI],10H` là lệnh không rõ ràng và có thể hiểu là cất dữ liệu 10H vào ô nhớ byte hay word. Để tránh hiểu nhầm ta phải thêm chỉ dẫn BYTE PTR hay WORD PTR như sau:

**MOV BYTE PTR [DI], 10H****MOV WORD PTR [DI], 10H**

Cách định địa chỉ gián tiếp thanh ghi thường dùng để truy xuất bảng dữ liệu trong bộ nhớ.

**b. Định địa chỉ nền (base addressing):**

Trong cách định địa chỉ này thì địa chỉ vật lý của ô nhớ chứa toán hạng được tính toán bằng cách cộng độ dời gián tiếp hoặc trực tiếp với nội dung của thanh ghi BX hoặc thanh ghi BP và cộng với nội dung thanh ghi đoạn DS hoặc SS theo thứ tự.

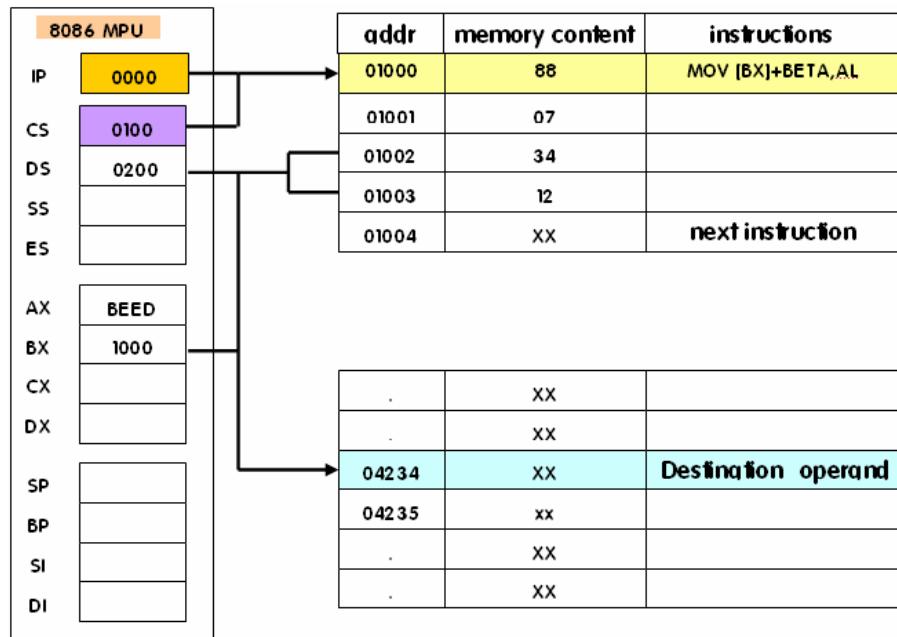
**Ví dụ 5:** Lệnh **MOV [BX] + BETA, AL**

Lệnh có chức năng “copy dữ liệu trong thanh ghi AL sang ô nhớ có địa chỉ offset bằng giá trị lưu trong thanh ghi BX cộng với hằng số BETA trong vùng nhớ đoạn bộ nhớ dữ liệu” và quá trình được thực hiện như hình 3-17a và 3-17b.

Trong lệnh này thì giá trị lưu trong thanh ghi BX bằng 1000H nằm trong vi xử lý, giá trị BETA bằng 1234H nên địa chỉ offset bằng 2234H. Địa chỉ offset này được cộng thêm địa chỉ của thanh ghi DS hiện hành sau khi nhân 16 như sau:

$$PA = 02000H + 2234H = 04234H$$

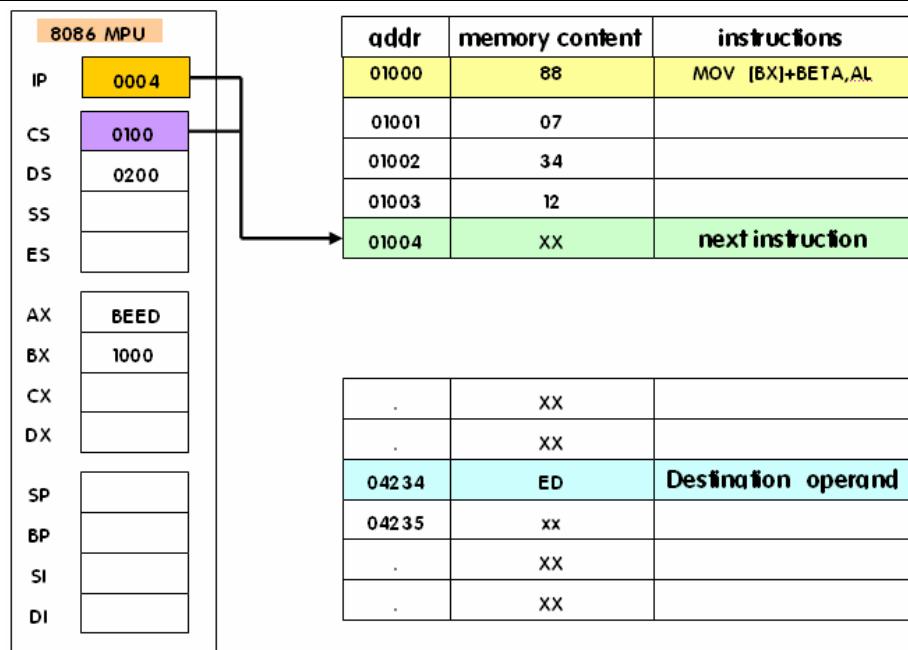
Chính là địa chỉ của ô nhớ cần truy xuất dữ liệu.



**Hình 3-17a. Trước khi thực hiện lệnh MOV [BX] +BETA,AL.**

Sau khi thực hiện lệnh thì thanh ghi IP tăng lên 4 đơn vị vì mã của lệnh này bằng 4 byte, dữ liệu trong ô nhớ có địa chỉ 04234H mang giá trị bằng EDH.

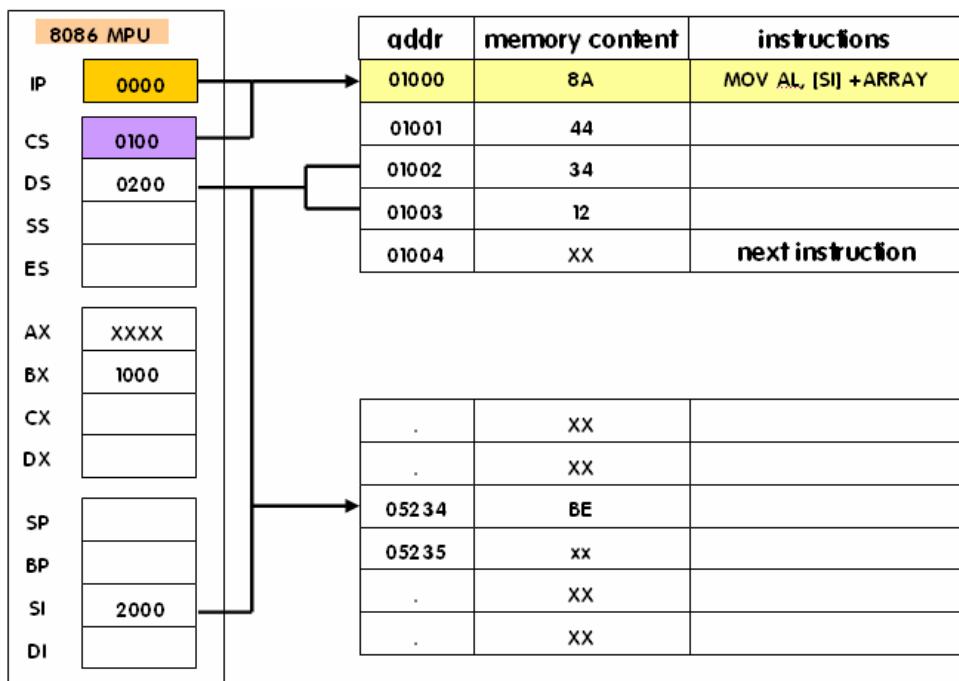
Độ dời 8 bit hay 16 bit phải xuất hiện trong vùng toán hạng và được xem là số nhị phân có dấu. Với số nhị phân 8 bit thì độ dời phải nằm trong tầm từ -128 ÷ +127, với số nhị phân 16 bit thì độ dời phải nằm trong tầm từ -32768 ÷ +32767.

Hình 3-17b. Sau khi thực hiện lệnh `MOV [BX] +BETA, AL`.

### c. Định địa chỉ chỉ số (index addressing):

Giống như kiểu định địa chỉ nền nhưng thanh ghi sử dụng là thanh ghi SI hoặc DI. Toán hạng có địa chỉ là tổng của độ dời của số nhị phân có dấu 8 bit hay 16 bit với thanh ghi SI hoặc DI và kết hợp với thanh ghi đoạn DS.

#### Ví dụ 6: Lệnh `MOV AL, [SI] + ARRAY`

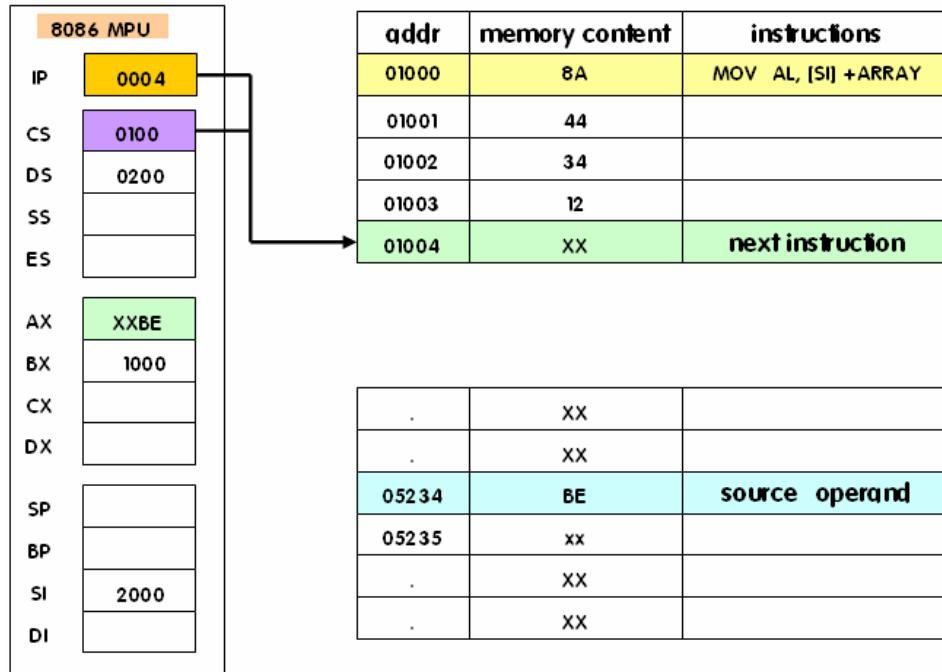
Hình 3-18a. Trước khi thực hiện lệnh `MOV AL,[SI] +ARRAY`.

Lệnh có chức năng “copy dữ liệu trong ô nhớ có địa chỉ offset bằng giá trị lưu trong thanh ghi SI với hằng số ARRAY trong đoạn bộ nhớ dữ liệu sang thanh ghi AL” và quá trình được thực hiện như hình 3-18a và 3-18b.

Trong lệnh này thì giá trị lưu trong thanh ghi SI bằng 2000H nằm trong vi xử lý, giá trị ARRAY bằng 1234H nên địa chỉ offset bằng 3234H. Địa chỉ offset này được cộng thêm địa chỉ của thanh ghi DS hiện hành sau khi dịch sang phải 4 bit như sau:

$$PA = 02000H + 3234H = 05234H$$

Chính là địa chỉ của ô nhớ cần truy xuất dữ liệu.



**Hình 3-18b. Sau khi thực hiện lệnh **MOV AL, [SI] + ARRAY**.**

Sau khi thực hiện lệnh thì thanh ghi IP tăng lên 4 đơn vị vì mã của lệnh này bằng 4 byte, dữ liệu trong ô nhớ có địa chỉ 04234H mang giá trị bằng EDH được copy sang thanh ghi AL.

Cách định địa chỉ nền và cách định địa chỉ chỉ số còn được gọi là cách định địa chỉ tương đối dùng thanh ghi.

#### d. Định địa chỉ nền – chỉ số với độ dời :

Các thanh ghi nền và thanh ghi chỉ số được cộng vào để tạo địa chỉ offset. Thanh ghi nền BX hay BP được cộng với thanh ghi chỉ số SI hoặc DI.

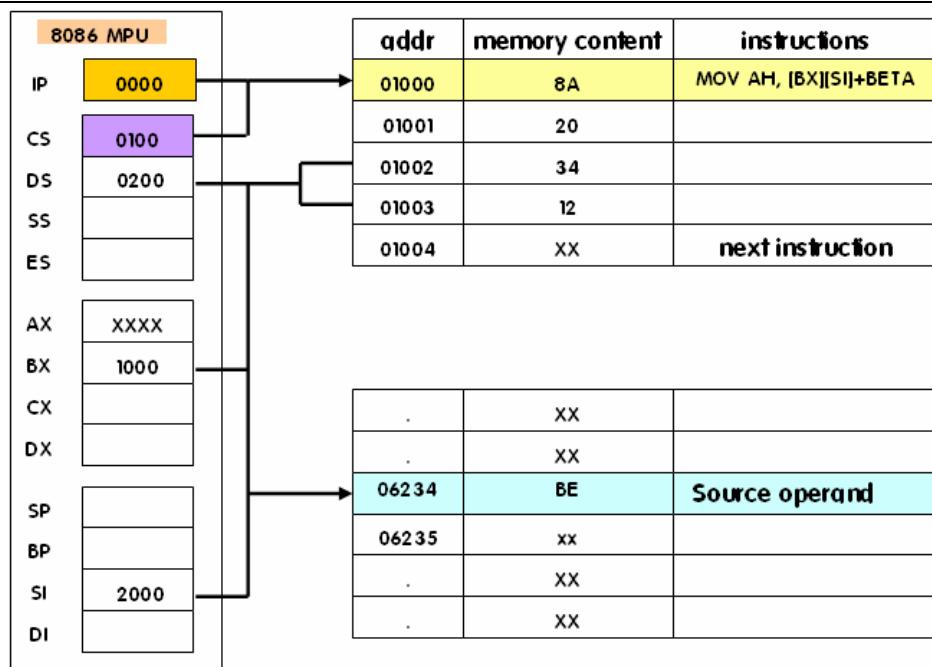
#### Ví dụ 7: Lệnh **MOV AH, [BX][SI] + BETA**

Lệnh có chức năng “copy dữ liệu trong ô nhớ có địa chỉ offset bằng giá trị lưu trong thanh ghi BX cộng với giá trị lưu trong thanh ghi BX với hằng số BETA trong đoạn bộ nhớ dữ liệu sang thanh ghi AH” và quá trình được thực hiện như hình 3-19a và 3-19b.

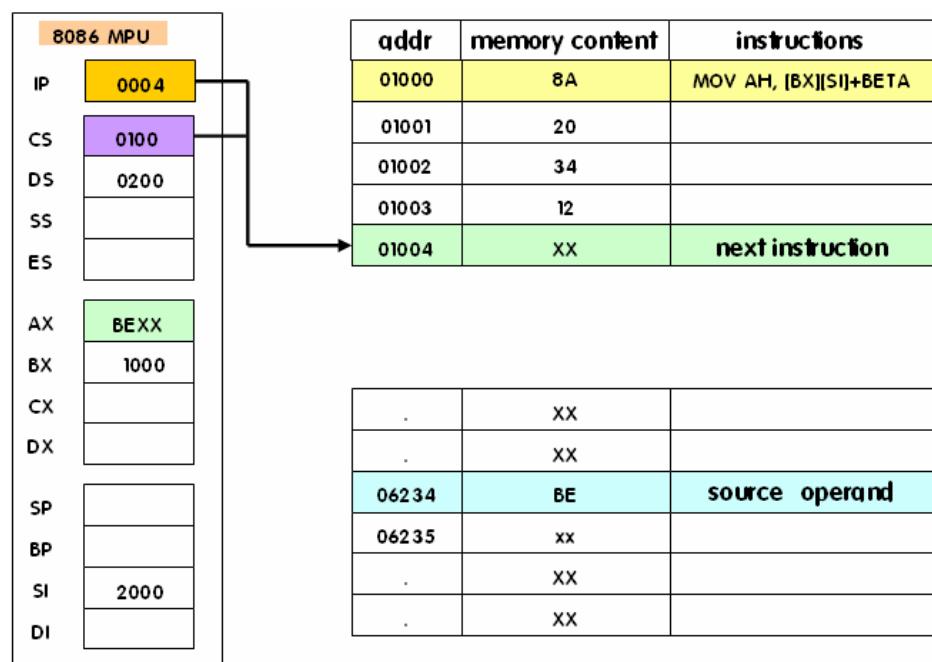
Trong lệnh này thì giá trị lưu trong thanh ghi BX bằng 1000H, cộng với giá trị lưu trong thanh ghi SI bằng 2000H nằm trong vi xử lý, giá trị BETA bằng 1234H nên địa chỉ offset bằng 4234H. Địa chỉ offset này được cộng thêm địa chỉ của thanh ghi DS sau khi nhân 16 như sau:

$$PA = 02000H + 4234H = 06234H$$

Chính là địa chỉ của ô nhớ cần truy xuất dữ liệu.



Hình 3-19a. Trước khi thực hiện lệnh MOV AX,[BX] [SI] +BETA.



Hình 3-19b. Sau khi thực hiện lệnh MOV AX,[BX] [SI] +BETA.

Sau khi thực hiện lệnh thì thanh ghi IP tăng lên 4 đơn vị vì mã của lệnh này bằng 4 byte, dữ liệu trong ô nhớ có địa chỉ 04234H mang giá trị bằng BEH được copy sang thanh ghi AH.

##### 5. Cách định địa chỉ chuỗi (string):

Chuỗi là 1 dãy liên tục các byte hay các word được cất trong bộ nhớ và được lưu trữ ở dạng ASCII. Vì xử lý 8086/88 có nhiều lệnh được thiết kế riêng để xử lý các chuỗi ký tự.

Các lệnh này có cách định địa chỉ đặc biệt và sử dụng DS:SI để quản lý địa chỉ của chuỗi nguồn và ES:DI để quản lý địa chỉ của chuỗi đích. Lệnh MOVSBL dùng cách định địa chỉ chuỗi

có chức năng chuyển dữ liệu nguồn đến vị trí đích. Giá trị của SI và DI sẽ tự động tăng lên hoặc giảm phụ thuộc vào giá trị của cờ DF trong thanh ghi trạng thái.

Các lệnh với chuỗi không được sử dụng bất kỳ cách định địa chỉ nào khác, các chuỗi có thể dài đến 64kbyte.

**Ví dụ 8:** cho DS = 1000H ES = 2000H, SI = 10H, DI = 20H

Lệnh: MOVS B ;copy nội dung ô nhớ [10010H] đến [20020H]

#### 6. Cách định địa chỉ cổng (port addressing mode):

Vi xử lý 8086/88 có vùng nhớ và vùng IO độc lập với nhau, vi xử lý 8086/88 có thể giao tiếp lên đến 65536 IO. Các cổng IO có thể được truy xuất với địa chỉ trực tiếp ở dạng byte thì chỉ truy xuất được 256 cổng IO:

**Ví dụ 9:**

Lệnh: IN AL,40H ;copy nội dung cổng IO có địa chỉ 40H vào AL  
OUT 40H,AL ;copy nội dung của AL sang cổng IO có địa chỉ 40H

Cổng IO có thể truy cập bằng thanh ghi – khi đó có thể truy cập đến 65536 cổng IO:

**Ví dụ 10:**

Lệnh: IN AL,DX ;copy nội dung cổng IO có địa chỉ lưu trong DX vào AL  
OUT DX,AL ;copy nội dung của AL sang cổng IO có địa chỉ trong DX

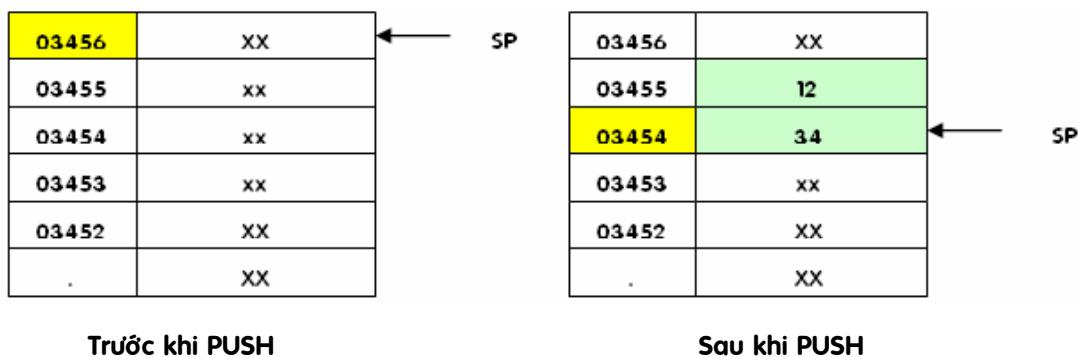
#### 7. Cách định địa chỉ ngăn xếp:

Cách này sử dụng cho các lệnh gọi chương trình con và lệnh kết thúc chương trình con CALL/RET và lệnh PUSH /POP. Ngăn xếp của vi xử lý 8086/88 được tổ chức theo cấu trúc vào sau ra trước LIFO.

**Ví dụ 11:**

Lệnh: PUSH AX ;cất nội dung AX vào ngăn xếp tại [SS:SP-1] và [SS:SP-2]

Giả sử cho AX = 1234H thì nó cất vào ngăn xếp được minh họa như hình 3-20:



Hình 3-20. Hoạt động của ngăn xếp với lệnh PUSH.

Đối với lệnh gọi chương trình con CALL thì địa chỉ của lệnh quay về trong thanh ghi IP được tự động cất vào ngăn xếp. Khi kết thúc chương trình con bằng lệnh RET thì địa chỉ đã cất trong ngăn xếp được trả lại cho thanh ghi IP. Nếu là lệnh gọi xa (far call) thì 2 word được cất

vào ngăn xếp (offset được cất trước) và khi lấy lại cũng lấy 2 word và mã kết thúc lệnh bằng lệnh RET khác.

Chú ý: lệnh PUSH và POP luôn làm việc với toán hạng là word.

### 8. Các qui tắc kết hợp các thanh ghi đoạn và offset:

Vi xử lý có các qui tắc kết hợp thanh ghi đoạn và thanh ghi offset được sử dụng cho 1 số cách định địa chỉ nhất định. Tuy nhiên ta có thể viết chồng lên bằng cách sử dụng tiếp đầu ngữ viết chồng đoạn, ví dụ MOV CL,DS:[BP]

Thanh ghi Offset	Thanh ghi đoạn mặc nhiên	Tiếp đầu ngữ viết chồng
IP	CS	Không có
SP	SS	Không có
BP	SS	DS, ES hoặc CS
SI, DI	DS	ES, SS hoặc CS
DI cho các lệnh chuỗi kí tự	ES	ES, SS hoặc CS

Bảng 3-9. Qui tắc kết hợp các thanh ghi đoạn và thanh ghi offset.

#### Ví dụ 12:

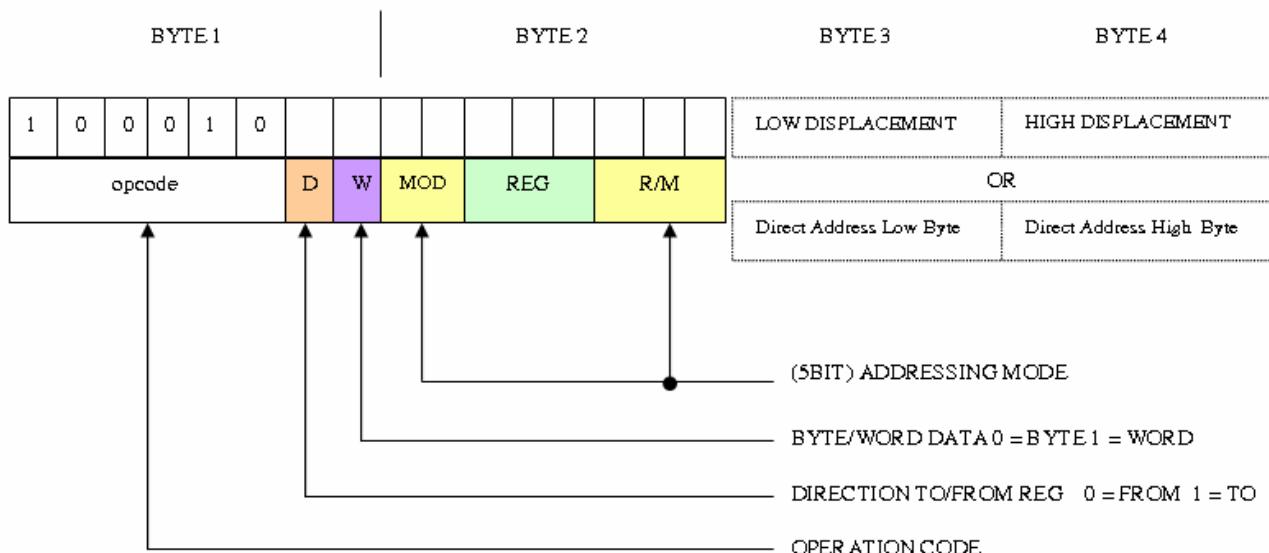
Lệnh:      MOV AX,[DI]                                ;AX  $\leftrightarrow$  DS: [DI]  
                 MOV AX,ES:[DI]                               ;AX  $\leftrightarrow$  ES: [DI]

### III. TẬP LỆNH:

#### 1 Mã lệnh vi xử lý 8086/88:

Vi xử lý 8086/88 có tập lệnh khoảng 300 lệnh, mỗi lệnh có thể dài 1 byte đến 6 byte.

Hình 3-21 là dạng mã lệnh của vi xử lý 8086/88:



Hình 3-21. Dạng mã lệnh.

Giải thích mã lệnh:

**Vùng thanh ghi REG (register) :**

REG	W=1	W=0
000	AX	AL
001	BX	CL
010	CX	DL
011	DX	BL
100	SP	AH
101	BP	BH
110	SI	DH
111	DI	BH

**Bảng 3-10. Bảng liệt kê giá trị của REG.****Vùng MOD :**

MOD	Chức năng
00	Toán hạng lờ bộ nhớ trực tiếp nếu R/M =110, ngược lại toán hạng trực tiếp.
01	Toán hạng gián tiếp, độ dời 8 bit
10	Toán hạng gián tiếp, độ dời 16 bit
11	Hai thanh ghi được sử dụng, vùng R/M là vùng REG

**Bảng 3-11. Bảng liệt kê giá trị của MOD.****Vùng R/M (register/memory) :**

Nếu vùng MOD = 11 thì vùng R/M là thanh ghi nguồn dùng các mã của vùng REG, ngược lại thì:

R/M	Kiểu
000	DS:[BX+SI+d]
001	DS:[BX+DI+d]
010	SS:[BP+SI+d]
011	SS:[BP+DI+d]
100	DS:[SI+d]
101	DS:[DI+d]
110	DS:[BP+d]
111	DS:[BX+d]

**Bảng 3-12. Bảng liệt kê giá trị của R/M.****2. Tổng quan về tập lệnh vi xử lý 8086/88:**

Trong phần này chúng ta sẽ khảo sát các lệnh của vi xử lý và sử dụng các toán hạng hay chỉ dẫn assembler dựa trên trình biên dịch hợp ngữ của Microsoft (MASM 6.0).

Vi xử lý 8086 có 90 lệnh cơ bản (không kể các lệnh biến đổi về cách định địa chỉ), được chia làm 6 nhóm lệnh như sau: nhóm lệnh chuyển dữ liệu, nhóm lệnh số học, nhóm lệnh xử lý chuỗi, nhóm lệnh chuyển quyền điều khiển và nhóm lệnh điều khiển vi xử lý.

**Nhóm lệnh chuyển dữ liệu:** gồm có 14 lệnh như sau:

MOV, PUSH, POP, XCHG, IN, OUT, XLAT, LEA,  
LDS, LES, LAHF, SAHF, PUSHF, POPF.

**Nhóm lệnh số học:** gồm có 20 lệnh như sau:

ADD, ADC, INC AAA, BAA, SUB, SSB, DEC, NEG, CMP,

AAS, DAS, MUL, IMUL, AAM, DIV, IDIV, AAD, CBW, CWD.

**Nhóm lệnh LOGIC:** gồm có 12 lệnh như sau:

NOT, SHL, SAL, SHR, SAR, ROL, ROR, RCL, RCR, AND, TEST, OR, XOR.

**Nhóm lệnh xử lý chuỗi:** gồm có 6 lệnh như sau:

REP, MOVS, XMPs, SCAS, LODS, STOS.

**Nhóm lệnh chuyển điều khiển:** gồm có 26 lệnh như sau:

CALL, JMP, RET, JE/JZ, JL/JNGE, JLE/JNG, JB/JNAE,  
JBE/JNA, JP/JPE, JO, JS, JNE/JNZ, JNL/JGE, JNLE/JG, JNB/JAE,  
JNBE/JA, JNP/JPO, JNO, JNS, LOOP, LOOPZ/LOOPE, CJXZ, INT, INTO, IRETR.

**Nhóm lệnh điều khiển VI XỬ LÝ :** gồm có 12 lệnh như sau:

CLR, CMC, STC, CLD, STD, CLI, STI, HLT, WAIT, ESC, LOCK, NOP.

Mã lệnh bao gồm những thông tin như mã lệnh, địa chỉ, dữ liệu.

Số lượng chu kỳ xung clock cần để thực hiện lệnh phụ thuộc vào cách định địa chỉ. Với hệ thống 8086/88 thì quá trình tính toán địa chỉ thật (20bit) cần nhiều xung clock – với những vi xử lý từ 80286 trở về sau thì không còn tính toán địa chỉ nên không phải mất nhiều chu kỳ xung clock để tính địa chỉ.

#### Ví dụ 13: Một số lệnh và số chu kỳ thực hiện:

Cách định địa chỉ	Lệnh minh họa	Số chu kỳ
Gián tiếp thanh ghi	MOV CL,[DI]	5
Trực tiếp	MOV CL,DATA	3
Chỉ số nền	MOV BL,[BP+DI]	7
Chỉ số nền (có độ dời)	MOV CX,[BP+DI+DATA]	11
Tiếp đầu ngũ viết chồng đoạn	MOV AL,ES:DATA	EA+2

Bảng 3-13. Bảng liệt kê chu kỳ thực hiện của các lệnh.

Tóm tắt tập lệnh của vi xử lý 8086/88:

Mnemonic and Description	Instruction Code																														
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0																											
NOT: AL = 8-bit accumulator AX = 16-bit accumulator CX = Count register DS= Data segment ES = Extra segment Above/below refers to unsigned value. Greater = more positive; Less = less positive (more negative) signed values if d = 1 then "to" reg; if d = 0 then "from" reg if w = 1 then word instruction; if w = 0 then byte instruction if mod = 11 then r/m is treated as a REG field if mod = 00 then DISP = 0†, disp-low and disp-high are absent if mod = 01 then DISP = disp-low sign-extended 16-bits, disp-high is absent if mod = 10 then DISP = disp-high:disp-low if r/m = 000 then EA = (BX) + (SI) + DISP if r/m = 001 then EA = (BX) + (DI) + DISP if r/m = 010 then EA = (BP) + (SI) + DISP if r/m = 011 then EA = (BP) + (DI) + DISP if r/m = 100 then EA = (SI) + DISP if r/m = 101 then EA = (DI) + DISP if r/m = 110 then EA = (BP) + DISP † if r/m = 111 then EA = (BX) + DISP DISP follows 2nd byte of instruction (before data if required) † except if mod = 00 and r/m = 110 then EA = disp-high: disp-low. †† MOV CS, REG/MEMORY not allowed.	if s:w = 01 then 16-bits of immediate data form the operand. if s:w. = 11 then an immediate data byte is sign extended to form the 16-bit operand. if v = 0 then "count" = 1; if v = 1 then "count" in (CL) x= don't care z is used for string primitives for comparison with ZF FLAG.	SEGMENT OVERRIDE PREFIX 001 reg 11 0	REG is assigned according to the following table:  <table border="1"><thead><tr><th>16-BIT (w = 1)</th><th>8-BIT (w = 0)</th><th>SEGMENT</th></tr></thead><tbody><tr><td>000 AX</td><td>000 AL</td><td>00 ES</td></tr><tr><td>001 CX</td><td>001 CL</td><td>01 CS</td></tr><tr><td>010 DX</td><td>010 DL</td><td>10 SS</td></tr><tr><td>011 BX</td><td>011 BL</td><td>11 DS</td></tr><tr><td>100 SP</td><td>100 AH</td><td>00 ES</td></tr><tr><td>101 BP</td><td>101 CH</td><td>00 ES</td></tr><tr><td>110 SI</td><td>110 DH</td><td>00 ES</td></tr><tr><td>111 DI</td><td>111 BH</td><td>00 ES</td></tr></tbody></table> Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file: FLAGS = X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF) Mnemonics © Intel, 1978	16-BIT (w = 1)	8-BIT (w = 0)	SEGMENT	000 AX	000 AL	00 ES	001 CX	001 CL	01 CS	010 DX	010 DL	10 SS	011 BX	011 BL	11 DS	100 SP	100 AH	00 ES	101 BP	101 CH	00 ES	110 SI	110 DH	00 ES	111 DI	111 BH	00 ES	
16-BIT (w = 1)	8-BIT (w = 0)	SEGMENT																													
000 AX	000 AL	00 ES																													
001 CX	001 CL	01 CS																													
010 DX	010 DL	10 SS																													
011 BX	011 BL	11 DS																													
100 SP	100 AH	00 ES																													
101 BP	101 CH	00 ES																													
110 SI	110 DH	00 ES																													
111 DI	111 BH	00 ES																													

MNEMONIC AND DESCRIPTION	INSTRUCTION CODE			
	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
<b>DATA TRANSFER</b>				
<b>MOV = MOVE:</b>				
Register/Memory to/from Register	1 0 0 0 1 0 d w	mod reg r/m		
Immediate to Register/Memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1
Immediate to Register	1 0 1 1 w reg	data	data if w = 1	
Memory to Accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high	
Accumulator to Memory	1 0 1 0 0 0 1 w	addr-low	addr-high	
Register/Memory to Segment Register ↑↑	1 0 0 0 1 1 1 0	mod 0 reg r/m		
Segment Register to Register/Memory	1 0 0 0 1 1 0 0	mod 0 reg r/m		
<b>PUSH = Push:</b>				
Register/Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m		
Register	0 1 0 1 0 reg			
Segment Register	0 0 0 reg 1 1 0			
<b>POP = Pop:</b>				
Register/Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m		
Register	0 1 0 1 1 reg			
Segment Register	0 0 0 reg 1 1 1			
<b>XCHG = Exchange:</b>				
Register/Memory with Register	1 0 0 0 0 1 1 w	mod reg r/m		
Register with Accumulator	1 0 0 1 0 reg			
<b>IN = Input from:</b>				
Fixed Port	1 1 1 0 0 1 0 w	port		
Variable Port	1 1 1 0 1 1 0 w			
<b>OUT = Output to:</b>				
Fixed Port	1 1 1 0 0 1 1 w	port		
Variable Port	1 1 1 0 1 1 1 w			
<b>XLAT = Translate Byte to AL</b>				
LEA = Load EA to Register2	1 1 0 1 0 1 1 1			
LDS = Load Pointer to DS	1 0 0 0 1 1 0 1	mod reg r/m		
LES = Load Pointer to ES	1 1 0 0 0 1 0 1	mod reg r/m		
LAHF = Load AH with Flags	1 1 0 0 0 1 0 0	mod reg r/m		
SAHF = Store AH into Flags	1 0 0 1 1 1 1 1			
PUSHF = Push Flags	1 0 0 1 1 1 1 0			
POPF = Pop Flags	1 0 0 1 1 1 0 1			
<b>ARITHMETIC</b>				
<b>ADD = Add:</b>				
Register/Memory with Register to Either	0 0 0 0 0 0 d w	mod reg r/m		
Immediate to Register/Memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s:w = 01
Immediate to Accumulator	0 0 0 0 0 1 0 w	data	data if w = 1	
<b>ADC = Add with Carry:</b>				
Register/Memory with Register to Either	0 0 0 1 0 0 d w	mod reg r/m		

Immediate to Register/Memory	100000 s w	mod 0 1 0 r/m	data	data if s:w = 01
Immediate to Accumulator	0001010 w	data	data if w = 1	
<b>INC = Increment:</b>				
Register/Memory	1111111 w	mod 0 0 0 r/m		
Register	01000 reg			
AAA = ASCII Adjust for Add	00110111			
DAA = Decimal Adjust for Add	00100111			
<b>SUB = Subtract:</b>				
Register/Memory and Register to Either	001010 d w	mod reg r/m		
Immediate from Register/Memory	100000 s w	mod 1 0 1 r/m	data	data if s:w = 01
Immediate from Accumulator	0010110 w	data	data if w = 1	
<b>SBB = Subtract with Borrow:</b>				
Register/Memory and Register to Either	0001110 d w	mod reg r/m		
Immediate from Register/Memory	100000 s w	mod 0 1 1 r/m	data	data if s:w = 01
Immediate from Accumulator	0001110 w	data	data if w = 1	
<b>DEC = Decrement:</b>				
Register/Memory	1111111 w	mod 0 0 1 r/m		
Register	01001 reg			
NEG = Change Sign	1111011 w	mod 0 1 1 r/m		
<b>CMP = Compare:</b>				
Register/Memory and Register	0011110 d w	mod reg r/m		
Immediate with Register/Memory	100000 s w	mod 1 1 1 r/m	data	data if s:w = 01
Immediate with Accumulator	0011110 w	data	data if w = 1	
AAS = ASCII Adjust for Subtract	00111111			
DAS = Decimal Adjust for Subtract	00101111			
MUL = Multiply (Unsigned)	1111011 w	mod 1 0 0 r/m		
IMUL = Integer Multiply (Signed)	1111011 w	mod 1 0 1 r/m		
AAM = ASCII Adjust for Multiply	11010100	00001010		
DIV = Divide (Unsigned)	1111011 w	mod 1 1 0 r/m		
IDIV = Integer Divide (Signed)	1111011 w	mod 1 1 1 r/m		
AAD = ASCII Adjust for Divide	11010101	00001010		
CBW = Convert Byte to Word	10011000			
CWD = Convert Word to Double Word	10011001			
<b>LOGIC</b>				
NOT = Invert	1111011 w	mod 0 1 0 r/m		
SHL/SAL = Shift Logical/Arithmetic Left	110100 v w	mod 1 0 0 r/m		
SHR = Shift Logical Right	110100 v w	mod 1 0 1 r/m		
SAR = Shift Arithmetic Right	110100 v w	mod 1 1 1 r/m		
ROL = Rotate Left	110100 v w	mod 0 0 0 r/m		
ROR = Rotate Right	110100 v w	mod 0 0 1 r/m		
RCL = Rotate Through Carry Flag Left	110100 v w	mod 0 1 0 r/m		

RCR = Rotate Through Carry Right	110100vw	mod 011 r/m	
<b>AND = And:</b>			
Reg./Memory and Register to Either			
Immediate to Register/Memory	0010000dw	mod reg r/m	
Immediate to Accumulator	10000000w	mod 100 r/m	data
TEST = And Function to Flags, No Result:			
Register/Memory and Register	00100010w	mod reg r/m	
Immediate Data and Register/Memory	10000000w	mod 000 r/m	data
Immediate Data and Accumulator	1010100w	data	data if w = 1
<b>OR = Or:</b>			
Register/Memory and Register to Either	000010d w	mod reg r/m	
Immediate to Register/Memory	10000000w	mod 101 r/m	data
Immediate to Accumulator	0000110w	data	data if w = 1
<b>XOR = Exclusive or:</b>			
Register/Memory and Register to Either	001100d w	mod reg r/m	
Immediate to Register/Memory	10000000w	mod 110 r/m	data
Immediate to Accumulator	0011010w	data	data if w = 1
<b>STRING MANIPULATION</b>			
REP = Repeat	1111001z		
MOV\$ = Move Byte/Word	1010010w		
CMPS = Compare Byte/Word	1010011w		
SCAS = Scan Byte/Word	1010111w		
LODS = Load Byte/Word to AL/AX	1010110w		
STOS = Stor Byte/Word from AL/A	1010101w		
<b>CONTROL TRANSFER</b>			
<b>CALL = Call:</b>			
Direct Within Segment	11101000	disp-low	disp-high
Indirect Within Segment	11111111	mod 010 r/m	
Direct Intersegment	10011010	offset-low	offset-high
		seg-low	seg-high
Indirect Intersegment	11111111	mod 011 r/m	
<b>JMP = Unconditional Jump:</b>			
Direct Within Segment	11101001	disp-low	disp-high
Direct Within Segment-Short	11101011	disp	
Indirect Within Segment	11111111	mod 100 r/m	
Direct Intersegment	11101010	offset-low	offset-high
		seg-low	seg-high
Indirect Intersegment	11111111	mod 101 r/m	
<b>RET = Return from CALL:</b>			
Within Segment	11000011		
Within Seg Adding Immed to SP	11000010	data-low	data-high

Intersegment	11001011		
Intersegment Adding Immediate to SP	11001010	data-low	data-high
JE/JZ = Jump on Equal/Zero	01110100	disp	
JL/JNGE = Jump on Less/Not Greater or Equal	01111100	disp	
JLE/JNG = Jump on Less or Equal/ Not Greater	01111110	disp	
JB/JNAE = Jump on Below/Not Above or Equal	01110010	disp	
JBE/JNA = Jump on Below or Equal/Not Above	01110110	disp	
JP/JPE = Jump on Parity/Parity Even	01111010	disp	
JO = Jump on Overflow	01110000	disp	
JS = Jump on Sign	01111000	disp	
JNE/JNZ = Jump on Not Equal/Not Zero	01110101	disp	
JNL/JGE = Jump on Not Less/Greater or Equal	01111101	disp	
JNLE/JG = Jump on Not Less or Equal/Greater	01111111	disp	
JNB/JAE = Jump on Not Below/Above or Equal	01110011	disp	
JNBE/JA = Jump on Not Below or Equal/Above	01110111	disp	
JNP/JPO = Jump on Not Par/Par Odd	01111011	disp	
JNO = Jump on Not Overflow	01110001	disp	
JNS = Jump on Not Sign	01111001	disp	
LOOP = Loop CX Times	11100010	disp	
LOOPZ/LOOPE = Loop While Zero/Equal	11100001	disp	
LOOPNZ/LOOPNE = Loop While Not Zero/Equal	11100000	disp	
JCXZ = Jump on CX Zero	11100011	disp	
INT = Interrupt			
Type Specified	11001101	type	
Type 3	11001100		
INTO = Interrupt on Overflow	11001110		
IRET = Interrupt Return	11001111		
<b>PROCESSOR CONTROL</b>			
CLC = Clear Carry	11111000		
CMC = Complement Carry	11110101		
STC = Set Carry	11111001		
CLD = Clear Direction	11111100		
STD = Set Direction	11111101		
CLI = Clear Interrupt	11111010		
ST = Set Interrupt	11111011		
HLT = Halt	11110100		
WAIT = Wait	10011011		
ESC = Escape (to External Device)	11011xxx	mod xxx r/m	
LOCK = Bus Lock Prefix	11110000		

### 3. Khảo sát tập lệnh vi xử lý 8086/88:

Sau khi đã khảo sát lệnh tóm tắt đã trình bày ở trên thì việc tính toán ra mã lệnh do trình biên dịch assembler thực hiện, người lập trình chỉ cần biết có bao nhiêu lệnh và chức năng xử lý dữ liệu của từng lệnh để viết trình.

Do có rất nhiều lệnh và mỗi lệnh đều có ví dụ minh họa nên phần này được trình bày ở phần phụ lục.

## IV. CÂU HỎI VÀ BÀI TẬP:

**Câu 1:** Người lập trình vi xử lý 8086/88 bằng ngôn ngữ Assembly cần biết những thanh ghi nào bên trong của vi xử lý.

**Câu 2:** Có bao nhiêu thanh ghi bên trong của vi xử lý 8086/88.

**Câu 3:** Địa chỉ cao nhất của vi xử lý 8086 có giá trị bằng bao nhiêu? Thấp nhất bằng bao nhiêu?

**Câu 4:** Địa chỉ cao nhất của vi xử lý 8088 có giá trị bằng bao nhiêu? Thấp nhất bằng bao nhiêu?

**Câu 5:** Tổ chức bộ nhớ của vi xử lý 8086 theo bit, byte, word hay double word.

**Câu 6:** Tổ chức bộ nhớ của vi xử lý 8088 theo bit, byte, word hay double word.

**Câu 7:** Các nội dung của các ô nhớ có địa chỉ B0000H là FFH và B0001H là 00H vậy thì nội dung của word có địa chỉ chẵn là B0000H là bao nhiêu?

**Câu 8:** Hãy trình bày cách lưu trữ dữ liệu double word 12345678H trong ô nhớ có địa chỉ bắt đầu A001H như thế nào?

**Câu 9:** Các thanh ghi nào bên trong vi xử lý làm thanh ghi đoạn bộ nhớ.

**Câu 10:** Thành phần nào trong không gian địa chỉ bộ nhớ của vi xử lý 8086 được dùng để lưu trữ lệnh của chương trình.

**Câu 11:** Cái gì sẽ được lưu tại địa chỉ FFFF0H ?

**Câu 12:** Chức năng của thanh ghi con trỏ lệnh IP ?

**Câu 13:** Sau khi thực hiện lệnh thì thanh ghi IP thay đổi như thế nào ?

**Câu 14:** Hãy liệt kê các thanh ghi đa năng của vi xử lý 8086/88 ?

**Câu 15:** Hãy cho biết vai trò chức năng lưu trữ của các thanh ghi dữ liệu đa năng ?

**Câu 16:** Hãy cho biết loại dữ liệu nào được lưu trữ trong thanh ghi con trỏ (pointer) và thanh ghi chỉ số (index) ?

**Câu 17:** Hãy cho biết tên của 2 thanh ghi con trỏ?

**Câu 18:** Hãy cho biết thanh ghi con trỏ kết hợp với thanh ghi đoạn nào ?

**Câu 19:** Hãy cho biết thanh ghi chỉ số kết hợp với thanh ghi đoạn nào ?

**Câu 20:** Hãy cho sự khác nhau của thanh ghi con trỏ và thanh ghi chỉ số ?

**Câu 21:** Hãy liệt kê các bit trong thanh ghi trạng thái và cho biết chức năng của chúng ?

**Câu 22:** Hãy cho biết bit nào trong thanh ghi trạng thái chỉ hướng địa chỉ tăng hoặc giảm?

**Câu 23:** Hãy cho biết hai thành phần nào kết hợp lại tạo thành địa chỉ thật (PA) ?

**Câu 24:** Nếu giá trị hiện tại của thanh ghi CS và thanh ghi IP bằng 0200H và 01ACH theo thứ tự thì địa chỉ thật của lệnh kế bằng bao nhiêu ?

**Câu 25:** Nếu đoạn dữ liệu được định vị tại địa chỉ A0000H đến AFFFFH thì giá trị phải nạp vào thanh ghi DS bằng bao nhiêu ?

**Câu 26:** Từ kết quả ở trên nếu muốn ô nhớ có địa chỉ A1234H lưu dữ liệu thì giá trị cần phải nạp vào thanh ghi DI là bao nhiêu ?

**Câu 27:** Hãy cho biết chức năng của ngăn xếp ?

**Câu 28:** Nếu giá trị hiện tại của SS và SP là 0C00H và FF00H theo thứ tự thì địa chỉ đỉnh của ngăn xếp bằng bao nhiêu ?

**Câu 29:** Hãy trình bày quá trình thực hiện lưu giá trị trong thanh ghi AX = EE11H vào ngăn xếp nếu giá trị hiện tại của SS và SP là 0C00H và FF00H ?

**Câu 30:** Đối với vi xử lý 8086 thì vùng nhớ và vùng IO là chung hay độc lập ?

**Câu 31:** Hãy cho biết vùng IO lớn nhất là bao nhiêu ?

**Câu 32:** Hãy phân biệt các lệnh trên thuộc kiểu định địa chỉ nào:

- (a) MOV AL,BL
- (b) MOV AX,OFFH
- (c) MOV [DI],AX
- (d) MOV DI,[SI]
- (e) MOV [BX] +XYZ,CX
- (f) MOV [DI] + XZY,AH
- (g) MOV [BX][DI]+XYZ,AL

**Câu 33:** Nếu cho các giá trị của các thanh ghi: CS = 0A00H, DS = 0B00H, SI = 0100H, DI = 0200H, BX = 0300H và XYZ = 0400H. Hãy tính:

- (a) Địa chỉ của dữ liệu đến trong lệnh (c) của bài 32.
- (b) Địa chỉ của dữ liệu nguồn trong lệnh (d) của bài 32.
- (c) Địa chỉ của dữ liệu đến trong lệnh (e) của bài 32.
- (d) Địa chỉ của dữ liệu đến trong lệnh (f) của bài 32.
- (e) Địa chỉ của dữ liệu đến trong lệnh (g) của bài 32.

[return](#)  
**the end**

# HỆ VI ĐIỀU KHIỂN 8 BIT MCS51

## GIỚI THIỆU VI ĐIỀU KHIỂN MCS51

### KHẢO SÁT PHẦN CỨNG VI ĐIỀU KHIỂN MCS51

1. SƠ ĐỒ CẤU TRÚC BÊN TRONG VI ĐIỀU KHIỂN MCS51
2. KHẢO SÁT SƠ ĐỒ CHÂN 89C51
3. SƠ ĐỒ MẠCH KẾT NỐI MỘT SỐ ỨNG DỤNG ĐƠN GIẢN DÙNG BỘ NHỚ NỘI

### CẤU TRÚC BỘ NHỚ CỦA VI ĐIỀU KHIỂN

1. TỔ CHỨC BỘ NHỚ
2. CÁC THANH GHỊ CÓ CHỨC NĂNG ĐẶC BIỆT

### KHẢO SÁT TẬP LỆNH CỦA VI ĐIỀU KHIỂN

1. GIỚI THIỆU
2. CÁC KIỂU ĐỊNH ĐỊA CHỈ
3. KHẢO SÁT TẬP LỆNH VI ĐIỀU KHIỂN
  - a. Nhóm lệnh di chuyển dữ liệu
  - b. Nhóm lệnh số học
  - c. Nhóm lệnh logic
  - d. Nhóm lệnh chuyển quyền điều khiển
  - e. Nhóm lệnh xử lý bit
4. KHẢO SÁT TẬP LỆNH VI ĐIỀU KHIỂN

### KHẢO SÁT HOẠT ĐỘNG TIMER/COUNTER CỦA VI ĐIỀU KHIỂN

1. GIỚI THIỆU
2. THANH GHỊ CHỌN KIỂU LÀM VIỆC CHO TIMER – MODE REGISTER
3. THANH GHỊ ĐIỀU KHIỂN TIMER – CONTROL REGISTER
4. CÁC KIỂU HOẠT ĐỘNG CỦA TIMER VÀ CỜ TRÀN
5. CÁC NGUỒN XUNG ĐẾM
6. ĐIỀU KHIỂN CÁC TIMER HOẠT ĐỘNG
7. KHỞI TẠO VÀ TRUY XUẤT CÁC CỦA TIMER/COUNTER
8. KHẢO SÁT TIMER T2 CỦA HỘ VI ĐIỀU KHIỂN MCS52

### HOẠT ĐỘNG TRUYỀN DỮ LIỆU CỦA VI ĐIỀU KHIỂN

1. GIỚI THIỆU
2. THANH GHỊ ĐIỀU KHIỂN TRUYỀN DỮ LIỆU NỐI TIẾP
3. CÁC KIỂU TRUYỀN DỮ LIỆU NỐI TIẾP
  - a. Truyền dữ liệu kiểu thứ 0 – thanh ghi dịch 8 bit
  - b. Truyền dữ liệu kiểu 1 – thu phát bất đồng bộ 8 bit, tốc độ thay đổi
  - c. Truyền dữ liệu kiểu 2 – thu phát bất đồng bộ 9 bit, tốc độ cố định
  - d. Truyền dữ liệu kiểu 3 – thu phát bất đồng bộ 9 bit, tốc độ thay đổi
4. KHỞI TẠO VÀ TRUY XUẤT CÁC THANH GHỊ TRUYỀN DỮ LIỆU NỐI TIẾP
5. TRUYỀN DỮ LIỆU NỐI TIẾP TRONG HỆ THỐNG NHIỀU VI ĐIỀU KHIỂN
6. THIẾT LẬP TỐC ĐỘ TRUYỀN DỮ LIỆU NỐI TIẾP

## HOẠT ĐỘNG NGẮT CỦA VI ĐIỀU KHIỂN

1. GIỚI THIỆU
2. TỔ CHỨC NGẮT
3. CHO PHÉP NGẮT VÀ CẤM NGẮT
4. ĐIỀU KHIỂN ƯU TIÊN NGẮT
5. KIỂM TRA NGẮT THEO VÒNG QUÉT LIÊN TỤC
6. XỬ LÝ NGẮT

## BÀI TẬP

### *LIỆT KÊ CÁC HÌNH*

Hình 4-1. Cấu trúc bên trong của vi điều khiển.

Hình 4-2. Sơ đồ chân của 89C51

Hình 4-3. Kết nối vi điều khiển với IC chốt, bộ nhớ EPROM ngoại, mạch reset, tụ thạch anh.

Hình 4-4. Mạch đồng hồ số dùng led 7 đoạn.

Hình 4-5. Mạch định thời điều khiển 1 relay và hiển thị thời gian trên 2 led.

Hình 4-6. Mạch đồng hồ số hiển thị dùng LCD.

Hình 4-7. Mạch đồng hồ số hiển thị dùng LCD có thêm báo chuông giờ học.

Hình 4-8. Bảng tóm tắt các vùng nhớ 89C51.

Hình 4-9: Cấu trúc bộ nhớ RAM bên trong vi điều khiển.

Hình 4-10. Minh họa cách gán bank thanh ghi cho nhóm thanh ghi R.

Hình 4-11. Timer 1 hoạt động ở mode 0.

Hình 4-12. Timer 1 hoạt động ở mode 2.

Hình 4-13. Timer 0 hoạt động ở mode 3.

Hình 4-14. Các nguồn xung đưa đến timer / counter.

Hình 4-15. Đo độ rộng xung từ bên ngoài.

Hình 4-16. Sơ đồ chân của 89C52 với ngõ vào T2 là P1.0 và T2EX là P1.1 .

Hình 4-17. Hoạt động của timer T2 ở chế độ tự động nạp lại.

Hình 4-18. Hoạt động của timer T2 ở chế độ Thu nhận dữ liệu.

Hình 4-19. Sơ đồ khôi của truyền dữ liệu nối tiếp.

Hình 4-20. Giản đồ thời gian.

Hình 4-21. Giản đồ thời gian truyền dữ liệu mod 0.

Hình 4-22. Một ứng dụng kiểu 0 để tăng thêm ngõ ra bằng thanh ghi dịch.

Hình 4-23. Cung cấp xung cho truyền dữ liệu nối tiếp.

Hình 4-24. Cờ báo phát xong dữ liệu TI.

Hình 4-25. Kết nối nhiều vi xử lý.

Hình 4-26. Thiết lập tốc độ Baud.

Hình 4-27. Vi điều khiển thực hiện chương trình chính trong 2 trường hợp không và có ngắt.

Hình 4-28. Vi điều khiển 89C52 có 6 nguồn ngắt.

Hình 4-29. Cấu trúc ngắt của vi điều khiển.

### *LIỆT KÊ CÁC BẢNG*

Bảng 4-1. Chức năng các chân của port 3.

Bảng 4-2. Các thanh ghi sau khi vi điều khiển bị reset.

Bảng 4-3. Các bit trong thanh ghi trạng thái.

**Bảng 4-4. Các bit lựa chọn bank thanh ghi.**

**Bảng 4-5. Tóm tắt tập lệnh vi điều khiển MCS51.**

**Bảng 4-6. Các thanh ghi timer vi điều khiển MCS51.**

**Bảng 4-7. Các bit trong thanh ghi TMOD.**

**Bảng 4-8. Các bit chọn mode trong thanh ghi TMOD.**

**Bảng 4-9. Các bit trong thanh ghi TCON.**

**Bảng 4-10. Các bit trong thanh ghi T2CON.**

**Bảng 4-11. Các bit trong thanh ghi điều khiển truyền dữ liệu.**

**Bảng 4-12. Các kiểu truyền dữ liệu.**

**Bảng 4-13. Tóm tắt tốc độ baud.**

**Bảng 4-14. Tóm tắt chức năng các bit của thanh ghi IE.**

**Bảng 4-15. Tóm tắt chức năng các bit của thanh ghi IP.**

**Bảng 4-16. Tóm tắt các bit cờ của các nguồn ngắn.**

**Bảng 4-17. Tóm tắt vector địa chỉ ngắn.**

## PCON – Power Control Register

D7	D6	D5	D4	D3	D2	D1	D0
SMOD	x	x	x	GF1	GF0	PD	IDL

Address: 87H (not bit addressable)

SMOD – Serial mode bit used to determine the baud rate with Timer 1.

$$\text{Baud rate} = \frac{\text{Oscillator frequency in Hz}}{N[256 - (TH1)]}$$

If SMOD = 0 then N = 384. If SMOD = 1 then N = 192. TH1 is the high byte of timer 1 when it is in 8-bit autoreload mode.

GF1 and GF0 are General purpose flags not implemented on the standard device

PD is the power down bit. Not implemented on the standard device

IDL activate the idle mode to save power. Not implemented on the standard device

## I. GIỚI THIỆU VI ĐIỀU KHIỂN 8051:

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

Vi xử lý có rất nhiều loại bắt đầu từ 4 bit cho đến 32 bit, vi xử lý 4 bit hiện nay không còn nhưng vi xử lý 8 bit vẫn còn mặc dù đã có vi xử lý 32 bit.

Lý do sự tồn tại của vi xử lý 8 bit là phù hợp với 1 số yêu cầu điều khiển của các thiết bị điều khiển trong công nghiệp. Các vi xử lý 32 bit thường sử dụng cho các máy tính vì khối lượng dữ liệu của máy tính rất lớn nên cần các vi xử lý càng mạnh càng tốt.

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

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

- Vi xử lý.
- Có bộ nhớ.
- Các IC ngoại vi.

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.

Khi vi điều khiển ra đời đã mang lại sự tiện lợi là dễ dàng sử dụng trong điều khiển công nghiệp, việc sử dụng vi điều khiển không đòi hỏi người sử dụng phải hiểu biết một lượng kiến thức quá nhiều như người sử dụng vi xử lý – dĩ nhiên người sử dụng hiểu biết càng nhiều thì càng tốt nhưng đối với người bắt đầu thì việc sử dụng vi xử lý là điều rất phức tạp trong khi đó mong muốn là sử dụng được ngay.

Các phần tiếp theo chúng ta sẽ khảo sát vi điều khiển để thấy rõ sự tiện lợi trong vấn đề điều khiển trong công nghiệp.

Có rất nhiều hãng chế tạo được vi điều khiển, hãng sản xuất nổi tiếng là ATMEL. Hãng Intel là nhà thiết kế. Có thể truy xuất để lấy tài liệu của hãng bằng địa chỉ "<http://www.atmel.com/>"

Có nhiều họ vi điều khiển mang các mã số khác nhau, một trong họ nổi tiếng là họ **MCS-51**.

- ✚ Trong họ MCS-51 thì vi điều khiển đầu tiên là 80C31 không có bộ nhớ bên trong là do chưa tích hợp được.
- ✚ Vi điều khiển 80C51 tích hợp được 4 kbyte bộ nhớ Prom. Chỉ lập trình 1 lần không thể xóa để lập trình lại được.
- ✚ Vi điều khiển 87C51 tích hợp được 4 kbyte bộ nhớ EPROM. Cho phép lập trình nhiều lần và xóa bằng tia cực tím.
- ✚ Vi điều khiển 89C51 tích hợp được 4 kbyte bộ nhớ flash rom nạp và xóa bằng điện một cách tiện lợi và nhanh chóng. Có thể cho phép nạp xóa hàng ngàn lần.

Song song với họ MCS-51 là họ MCS-52 có 3 timer nhiều hơn họ MCS-51 một timer và dung lượng bộ nhớ nội lớn gấp đôi tức là 8kbyte.

Hiện nay có rất nhiều vi điều khiển thế hệ mới có nhiều đặc tính hay hơn, nhiều thanh ghi hơn, dung lượng bộ nhớ lớn hơn.

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 thể nói vi xử lý và vi điều khiển được ứng dụng trong hầu hết mọi lĩnh vực tự động.

## II. KHẢO SÁT PHẦN CỨNG VI ĐIỀU KHIỂN HỘ MCS-51:

Đến thời điểm hiện nay có rất nhiều loại Vi điều khiển thuộc họ MCS-51, trong tài liệu sẽ giới thiệu về vi điều khiển 89C51 hoặc 89C52. Các vi điều khiển thế hệ sau sẽ được đề cập ở phần sau.

Các vi điều khiển họ MCS-51 có các đặc điểm chung như sau:

- ◆ Có 4 Kbyte bộ nhớ FLASH ROM bên trong dùng để lưu chương trình điều khiển.
- ◆ Có 128 Byte RAM nội.
- ◆ 4 Port xuất/nhập (Input/Output) 8 bit.
- ◆ Có khả năng giao tiếp truyền dữ liệu nối tiếp.
- ◆ Có thể giao tiếp với 64 Kbyte bộ nhớ bên ngoài dùng để lưu **chương trình** điều khiển.
- ◆ Có thể giao tiếp với 64 Kbyte bộ nhớ bên ngoài dùng để lưu **dữ liệu**.
- ◆ Có 210 bit có thể truy xuất từng bit. Có các lệnh xử lý bit.

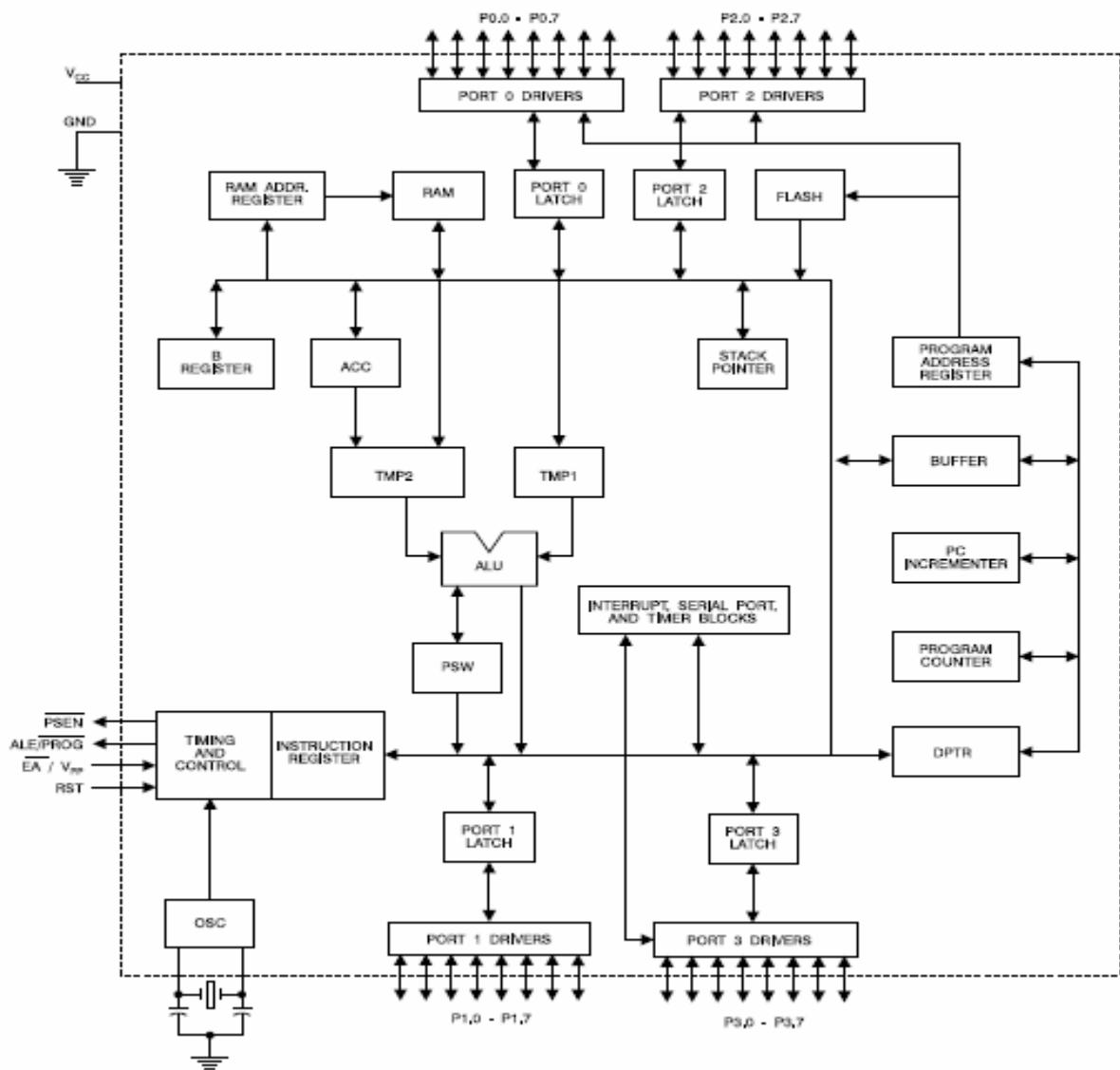
Tất cả các vi điều khiển cùng họ MCS-51 hoặc MCS-52 đều có các đặc tính cơ bản giống nhau như phần mềm, còn phần cứng thì khác nhau, các vi điều khiển sau này sẽ có nhiều tính năng hay hơn các vi điều khiển thế hệ trước. Ví dụ vi điều khiển 89C51 sẽ tiện cho việc sử dụng hơn vi điều khiển 80C51 hay 87C51. Vi điều khiển 89S51 sẽ hay hơn 89C51 vì có nhiều thanh ghi hơn, có thêm chế độ nạp nối tiếp rất tiện lợi. Những thế hệ đi sau sẽ kế thừa tất cả những gì của thế hệ đi trước. Trong phần này chỉ đề cập đến vi điều khiển 89C51/89C52.

### 1. SƠ ĐỒ CẤU TRÚC BÊN TRONG CỦA VI ĐIỀU KHIỂN

Sơ đồ cấu trúc của vi điều khiển được trình bày ở hình 4-1. Các thanh ghi có trong vi điều khiển bao gồm:

- 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 ngắn, truyền dữ liệu, khối timer/counter.
- Thanh ghi A, B, dptr và 4 port0, port1, port2, port3 có chốt và đệm.
- Thanh ghi bộ đếm chương trình PC (program counter).
- 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.



Hình 4-1. Cấu trúc bên trong của vi điều khiển.

Các khối bên trong của vi điều khiển có các thành phần giống như đã trình bày ở phần chương 2 như khối ALU, thanh ghi temp1, thanh ghi temp2, thanh ghi bộ đếm chương trình PC,

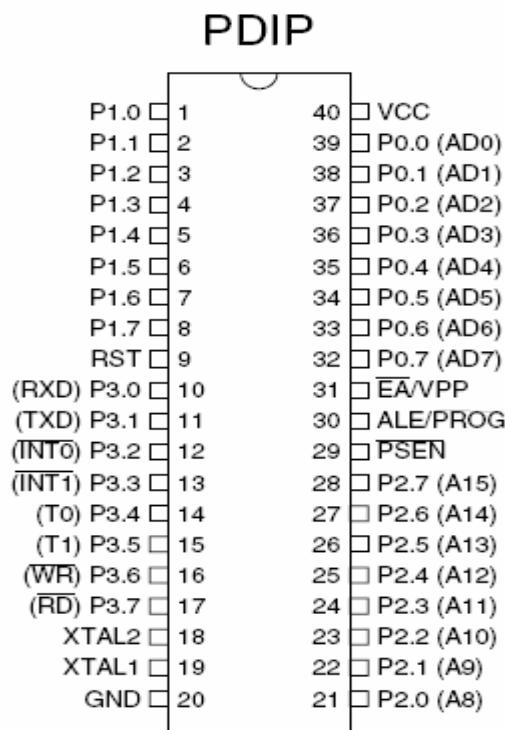
thanh con trỏ ngăn xếp, thanh ghi trạng thái PSW, thanh ghi lệnh IR, khối giải mã lệnh, khối điều khiển logic.

Tập lệnh cho người lập trình là kết quả của sự liên kết các khối bên trong của vi điều khiển – những gì tập lệnh cung cấp là đều do phần cứng xây dựng nên.

## 2. KHẢO SÁT SƠ ĐỒ CHÂN 89C51

Sơ đồ chân của vi điều khiển 89C51 được trình bày ở hình 4-2.

Vi điều khiển 89C51 có tất cả 40 chân. Trong đó có 24 chân có tác dụng kép (có nghĩa là 1 chân có 2 chức năng), mỗi đường có thể hoạt động như đường xuất nhập điều khiển IO [input output] hoặc là thành phần của các bus dữ liệu và bus địa chỉ để tải địa chỉ và dữ liệu khi giao tiếp với bộ nhớ ngoài.



Hình 4-2. Sơ đồ chân của 89C51

### Chức năng các chân của 89C51:

#### a. Các Port:

##### 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 lớn sử dụng bộ nhớ mở rộng bên ngoài thì port 0 có chức năng là bus địa chỉ và bus dữ liệu AD7 ÷ AD0. (Address: địa chỉ, data: dữ liệu)

##### 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 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.

Các chân của port này có nhiều chức năng, các công dụng chuyển đổi có liên hệ với các đặc tính đặc biệt của 89C51 như ở bảng 4-1:

Bit	Tên	Chức năng chuyển đổi
P3.0	RxD	Ngõ vào nhận dữ liệu nối tiếp.
P3.1	TxD	Ngõ xuất dữ liệu nối tiếp.
P3.2	$\overline{INT_0}$	Ngõ vào ngắt cứng thứ 0.
P3.3	$\overline{INT_1}$	Ngõ vào ngắt cứng thứ 1.
P3.4	T0	Ngõ vào nhận xung đếm của timer/counter thứ 0.
P3.5	T1	Ngõ vào nhận xung đếm của timer/counter thứ 1.
P3.6	$\overline{WR}$	Tín hiệu điều khiển ghi dữ liệu lên bộ nhớ ngoài.
P3.7	$\overline{RD}$	Tín hiệu điều khiển đọc dữ liệu từ bộ nhớ ngoài.

Bảng 4-1. Chức năng các chân của port 3.

#### b. Các tín hiệu điều khiển:

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

$\overline{PSEN}$  là tín hiệu ngõ ra ở chân 29 có tác dụng cho phép đọc bộ nhớ chương trình mở rộng thường nối đến chân  $\overline{OE}$  (output enable hoặc  $\overline{RD}$ ) của EPROM cho phép đọc các byte mã lệnh.

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

$\overline{PSEN}$  ở mức thấp trong thời gian vi điều khiển 89C51 lấy lệnh. Các mã lệnh của chương trình đọc từ EPROM qua bus dữ liệu và được chốt vào thanh ghi lệnh bên trong 89C51 để giải mã lệnh. Khi 89C51 thi hành chương trình trong EPROM nội thì  $\overline{PSEN}$  ở mức logic 1.

##### Tín hiệu điều khiển ALE (Address Latch Enable) :

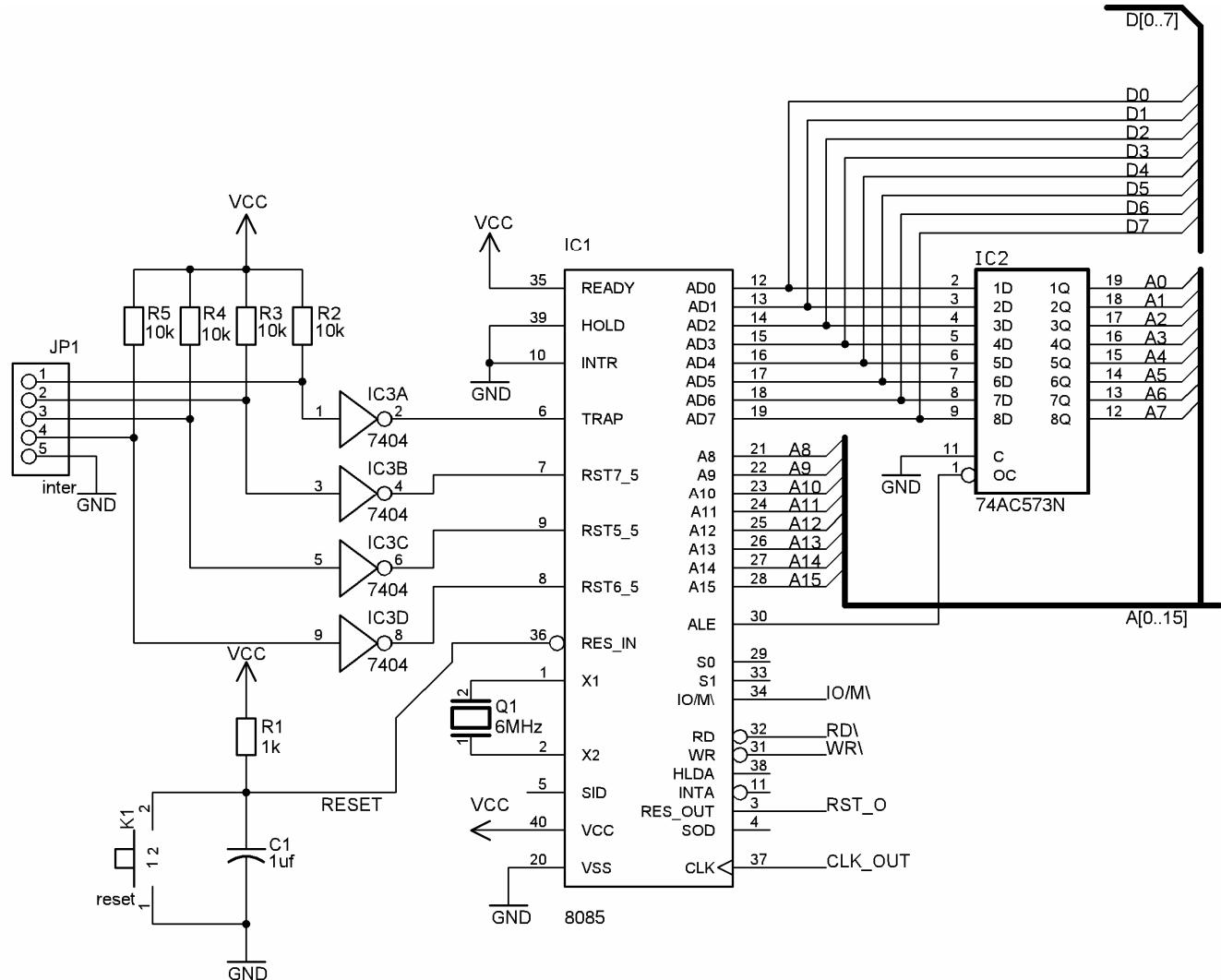
Khi vi điều khiển 89C51 truy xuất bộ nhớ bên ngoài, port 0 có chức năng là bus tải địa chỉ và bus dữ liệu [AD7 ÷ AD0] do đó phải tách các đường dữ liệu và địa chỉ. Tín hiệu ra ALE ở chân thứ 30 dùng làm tín hiệu điều khiển để giải đa hợp các đường địa chỉ và dữ liệu khi kết nối chúng với IC chốt. Xem hình 4-3.

Tín hiệu ra ở chân ALE là một xung trong khoảng thời gian port 0 đóng vai trò là địa chỉ thấp nên việc chốt địa chỉ được thực hiện 1 cách hoàn toàn tự động.

Các xung tín hiệu ALE có tần số bằng 1/6 tần số dao động thạch anh gắn vào vi điều khiển và có thể dùng tín hiệu xung ngõ ra ALE làm xung clock cung cấp cho các phần khác của hệ thống.

Trong chế độ lập trình cho bộ nhớ nội của vi điều khiển thì chân ALE được dùng làm ngõ vào nhận xung lập trình từ bên ngoài để lập trình cho bộ nhớ Flash Rom trong 89C51.

Ở hình 4-3 chỉ là minh họa kết nối vi điều khiển 89C52 với bộ nhớ EPROM ngoại để thấy vai trò của tín hiệu ALE, các đường còn lại của vi điều khiển có thể dùng để kết nối điều khiển các đối tượng khác.



**Hình 4-3. Kết nối vi điều khiển với IC chốt, bộ nhớ EPROM ngoại, mạch reset, tụ thạch anh.**

#### ☐ Tín hiệu $\overline{EA}$ (External Access):

Tín hiệu vào  $\overline{EA}$  ở chân 31 thường nối lên mức 1 hoặc mức 0.

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

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

#### ☐ Tín hiệu RST (Reset):

Ngõ vào RST ở chân 9 là ngõ vào Reset của 89C51. Sơ đồ kết nối mạch reset như hình 4-3. 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.

Trạng thái của tất cả các thanh ghi sau khi reset hệ thống được tóm tắt như bảng 4-2.

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

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

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 (1111 1111)
IP	XXX0 0000 B
IE	0X0X 0000 B
Các thanh ghi định thời	00H
SCON SBUF	00H
PCON (HMOS)	00H
PCON (CMOS)	0XXX XXXXXH 0XXX 0000 B

**Bảng 4-2. Các thanh ghi sau khi vi điều khiển bị reset.**

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

Bộ dao động được tích hợp bên trong 89C51, khi sử dụng 89C51 người thiết kế chỉ cần kết nối thêm tụ thạch anh và các tụ như trong hình 4-3. Tần số tụ thạch anh thường sử dụng cho 89C51 là 12Mhz ÷ 24Mhz.

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

**3. SƠ ĐỒ MẠCH KẾT NỐI MỘT SỐ ỨNG DỤNG ĐƠN GIẢN DÙNG BỘ NHỚ NỘI**

**a. Mạch đồng hồ số hiển thị giờ phút giây trên led 7 đoạn:**

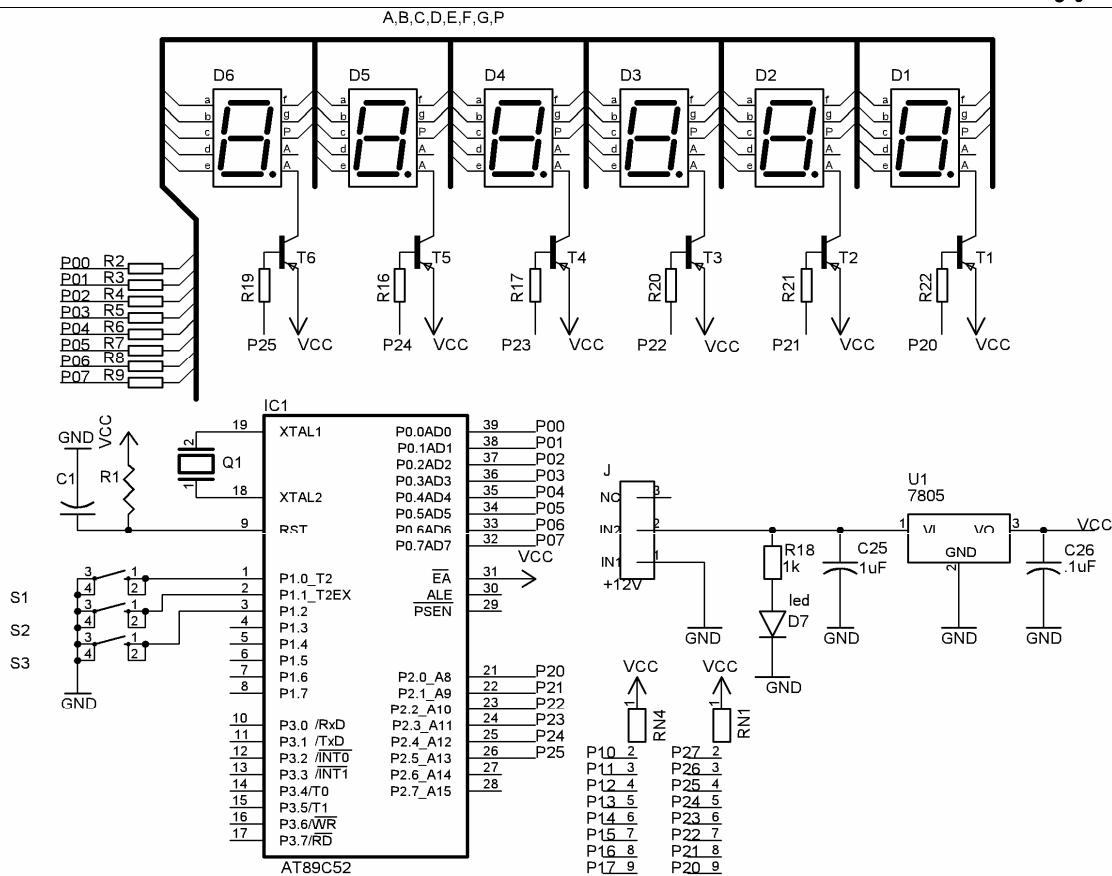
Trong sơ đồ hình 4-4 sử dụng 6 led 7 đoạn loại anode chung để hiển thị giờ, phút và giây sử dụng phương pháp quét, 6 transistor sử dụng là loại pnp (trong sơ đồ dùng A564) và điện trở cực B có giá trị khoảng  $10k\Omega$ , điện trở hạn dòng cho các đoạn có giá trị  $220\Omega$ . Để hiểu hoạt động điều khiển quét led 7 đoạn hãy đọc chương 8.

Hai điện trở màng 9 chân có giá trị là  $4,7k\Omega$  hoặc  $10k\Omega$ , tụ reset có giá trị  $10\mu F$ , điện trở reset có giá trị  $10k\Omega$ , thạch anh có giá trị thường là 12MHz.

3 nút nhấn S1, S2 và S3 dùng để chỉnh các thông số giờ phút giây.

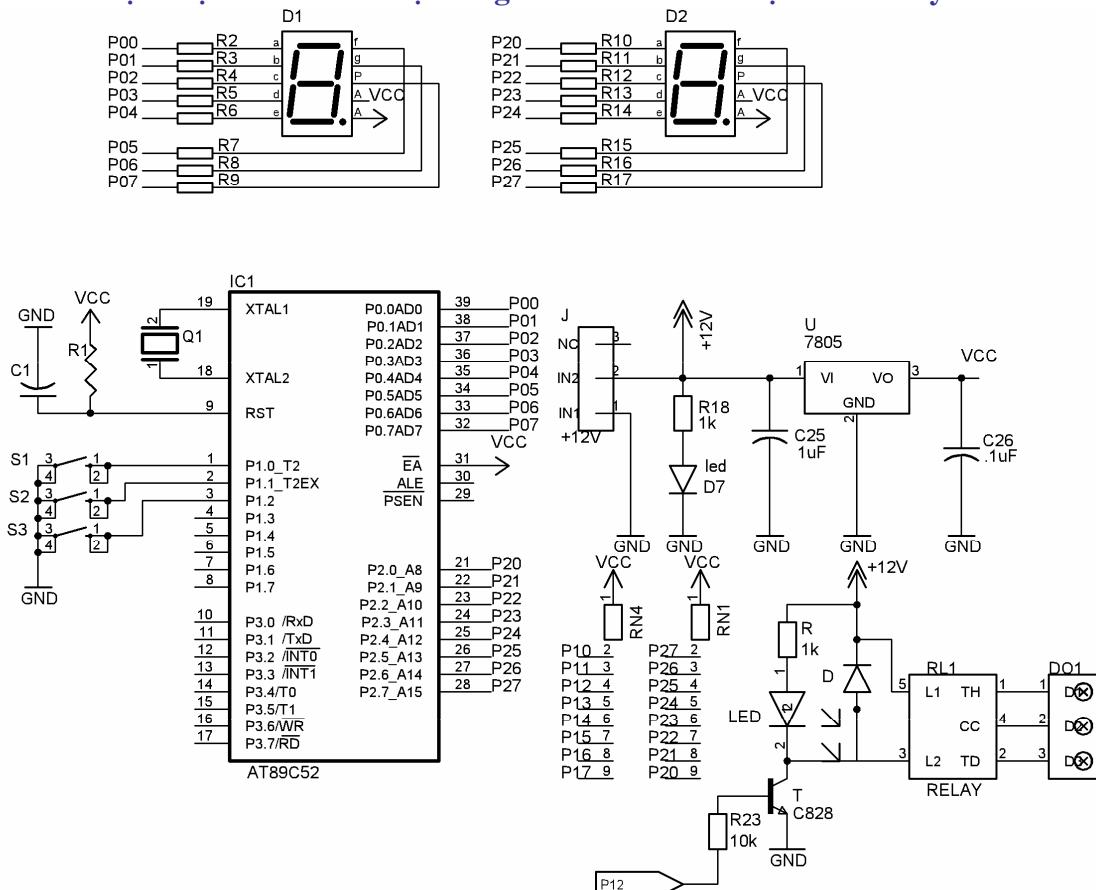
Mạch nguồn sử dụng IC ổn áp 5V.

Để hệ thống hoạt động thì phải có chương trình.



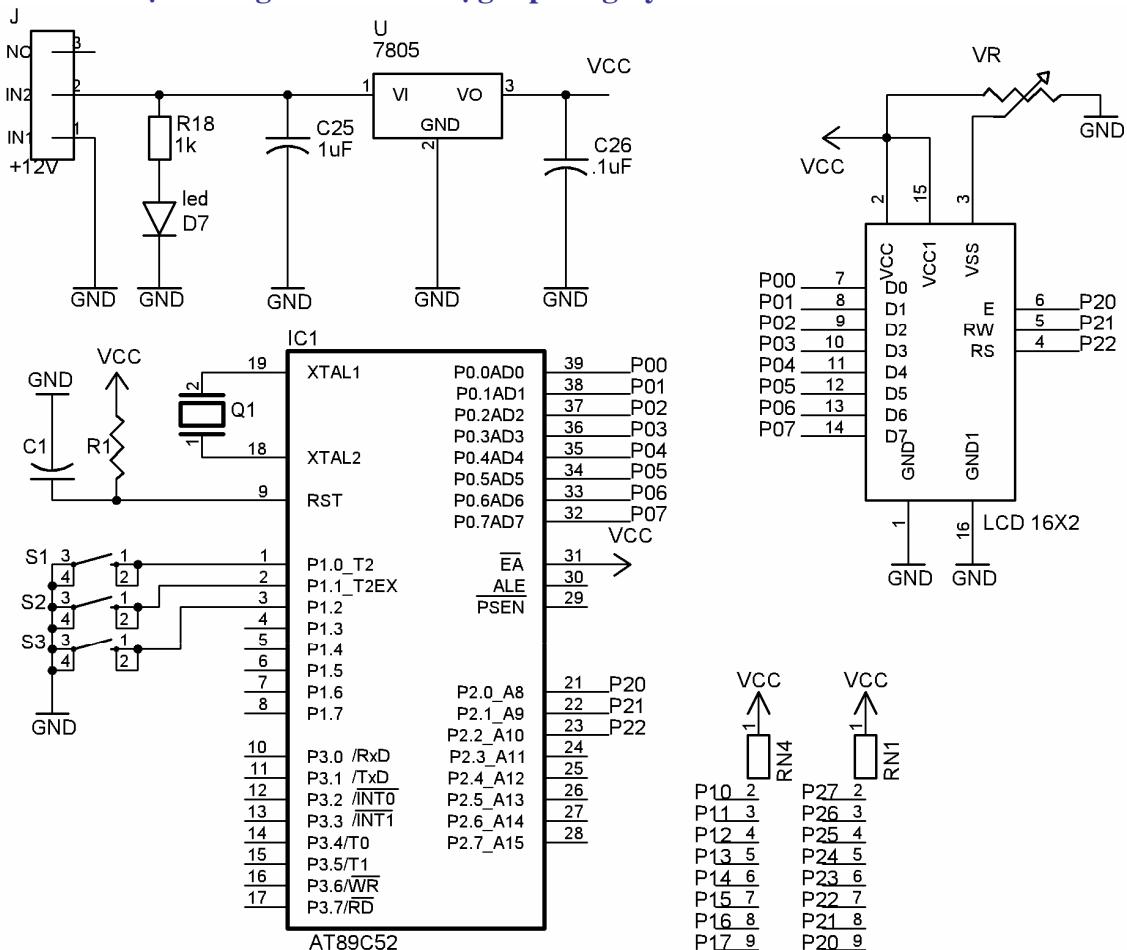
Hình 4-4. Mạch đồng hồ số dùng led 7 đoạn.

### b. Mạch định thời hiển thị thời gian trên 2 led 7 đoạn có 1 relay điều khiển:



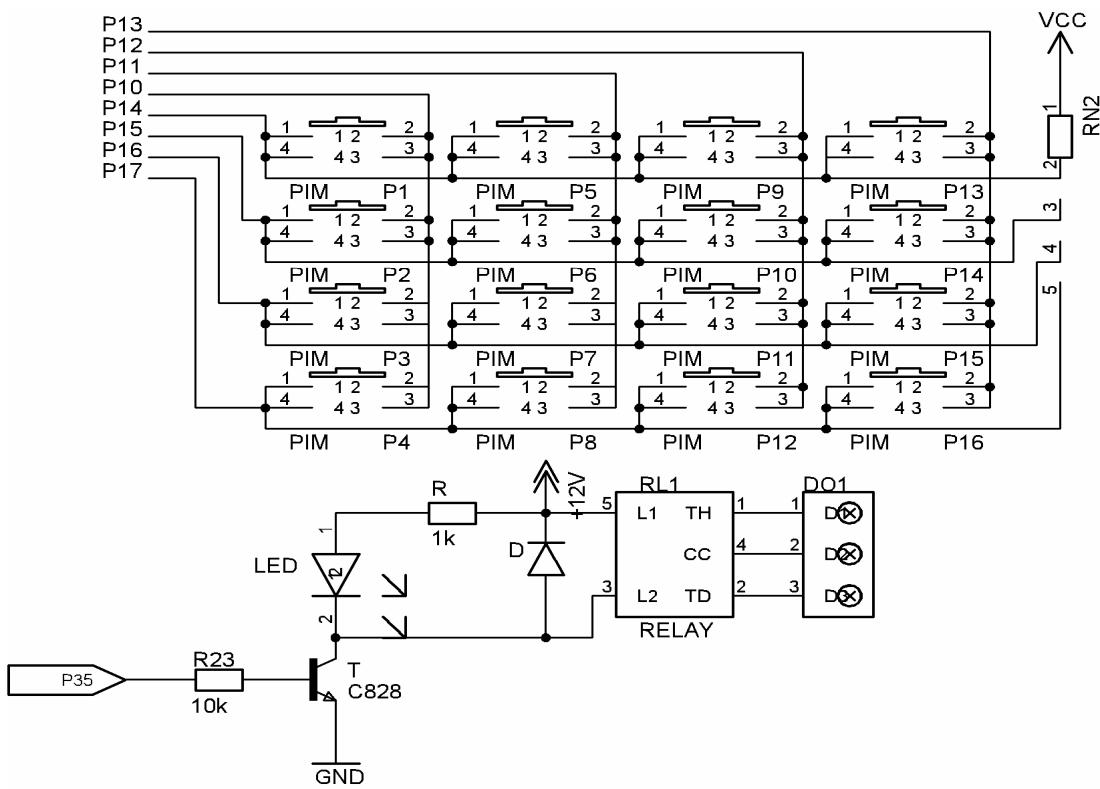
Hình 4-5. Mạch định thời điều khiển 1 relay và hiển thị thời gian trên 2 led.

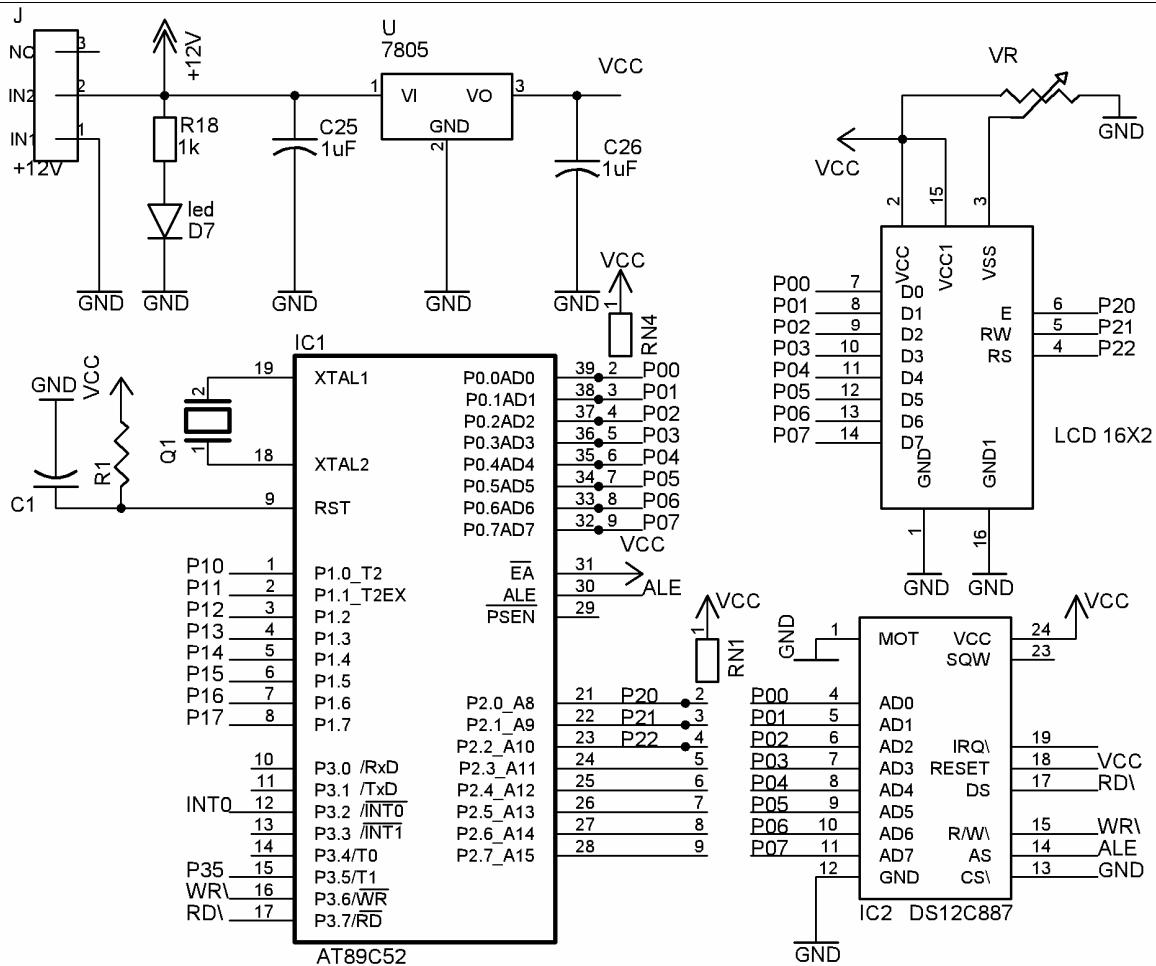
### c. Mạch đồng hồ số hiển thị giờ phút giây trên LCD:



Hình 4-6. Mạch đồng hồ số hiển thị giờ phút giây trên LCD.

### d. Mạch đồng hồ có thể báo chuông tiết học sử dụng real-time:





Hình 4-7. Mạch đồng hồ số hiển thị dùng LCD có thêm báo chuông giờ học.

### III. CẤU TRÚC BỘ NHỚ CỦA VI ĐIỀU KHIỂN:

#### 1 TỔ CHỨC BỘ NHỚ:

Vi điều khiển 89C51 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. [89C52 có 8 kbyte, 89W55 có 16kbyte].

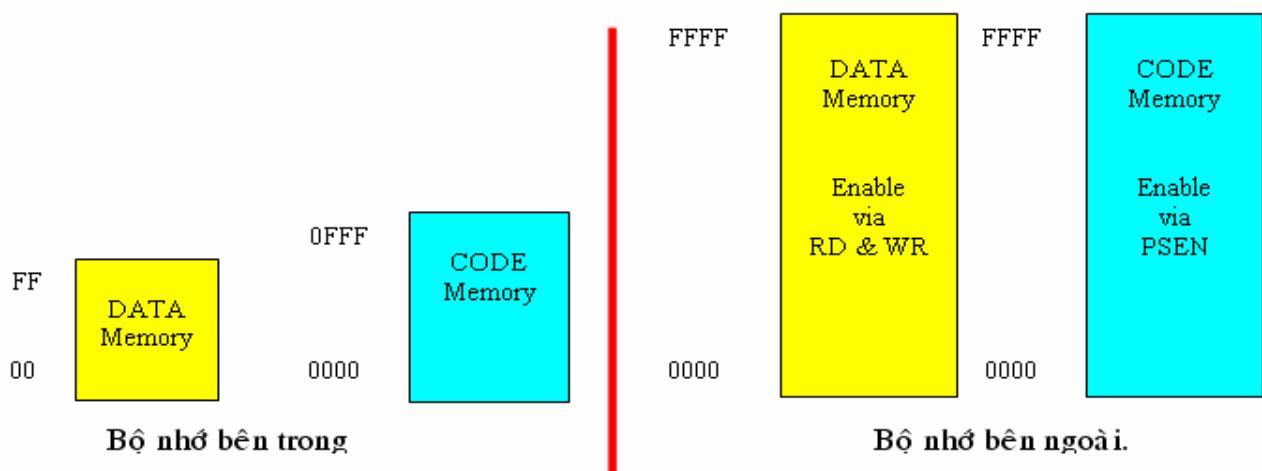
Bộ nhớ mở rộng bên ngoài cũng được chia ra làm 2 loại bộ nhớ: bộ nhớ dữ liệu và bộ nhớ chương trình. Khả năng giao tiếp là 64kbyte cho mỗi loại. Hình 4-8 minh họa khả năng giao tiếp bộ nhớ của vi điều khiển 89C51.

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

**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.**

Sơ đồ cấu trúc bên trong của bộ nhớ này được trình bày như hình 4-9.



Hình 4-8. Bảng tóm tắt các vùng nhớ 89C51.

**RAM bên trong 89C51 được phân chia như sau:**

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

#### □ Các bank thanh ghi :

32 byte thấp của bộ nhớ nội được dành cho 4 bank thanh ghi.

Bộ lệnh 89C51 hỗ trợ thêm 8 thanh ghi có tên là R0 đến R7 và theo mặc định sau khi reset hệ thống thì các thanh ghi R0 đến R7 được gán cho 8 ô nhớ có địa chỉ từ 00H đến 07H được minh họa bởi hình 4-10, khi đó bank 0 có 2 cách truy xuất bằng địa chỉ trực tiếp và bằng thanh ghi R.

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

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

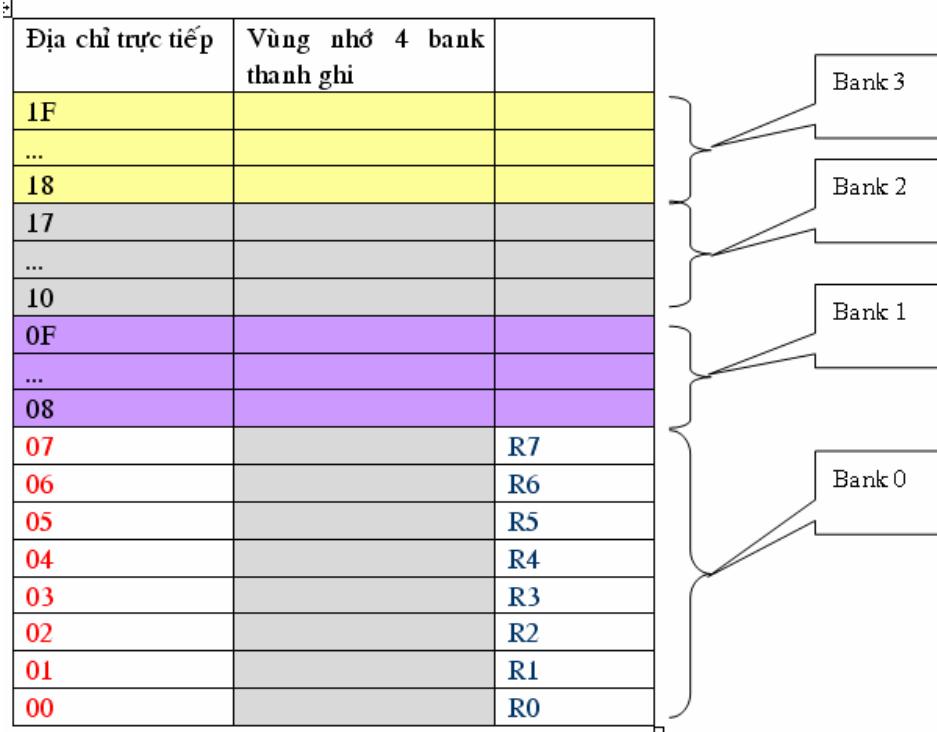
Do có 4 bank thanh ghi nên tại một thời điểm chỉ có một bank thanh ghi được truy xuất bởi các thanh ghi R0 đến R7, để chuyển đổi việc truy xuất các bank thanh ghi ta phải thay đổi các bit chọn bank trong thanh ghi trạng thái.

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

Chức năng chính của 4 bank thanh ghi này là nếu trong hệ thống có sử dụng nhiều chương trình thì chương trình thứ nhất bạn có thể sử dụng hết các thanh ghi R0 đến R7 của bank0, khi chuyển sang chương trình thứ 2 để xử lý một công việc gì đó và vẫn sử dụng các thanh ghi R0 đến R7 để lưu trữ cho việc sử lý dữ liệu mà không làm ảnh hưởng đến các dữ liệu R0 đến R7 trước đây và không cần phải thực hiện công việc cất dữ liệu thì cách nhanh nhất là gán nhóm thanh ghi R0 đến R7 cho bank1 là xong. Tương tự có thể mở thêm hai chương trình nữa và gán cho các bank 3 và 4.

Địa chỉ byte	Địa chỉ bit	Địa chỉ byte	Địa chỉ bit
7F	RAM đa dụng	FF	
30		F0	F7 F6 F5 F4 F3 F2 F1 F0
2F	7F 7E 7D 7C 7B 7A 79 78	E0	E7 E6 E5 E4 E3 E2 E1 E0
2E	77 76 75 74 73 72 71 70	D0	D7 D6 D5 D4 D3 D2 D1 D0
2D	6F 6E 6D 6C 6B 6A 69 68	B8	- - - BC BB BA B9 B8
2C	67 66 65 64 63 62 61 60	B0	B7 B6 B5 B4 B3 B2 B1 B0
2B	5F 5E 5D 5C 5B 5A 59 58	A8	AF      AC AB AA A9 A8
2A	57 56 55 54 53 52 51 50	A0	A7 A6 A5 A4 A3 A2 A1 A0
29	4F 4E 4D 4C 4B 4A 49 48	99	
28	47 46 45 44 43 42 41 40	98	9F 9E 9D 9C 9B 9A 99 98
27	3F 3E 3D 3C 3B 3A 39 38	90	97 96 95 94 93 92 91 90
26	37 36 35 34 33 32 31 30	8D	
25	2F 2E 2D 2C 2B 2A 29 28	8C	
24	27 26 25 24 23 22 21 20	8B	
23	1F 1E 1D 1C 1B 1A 19 18	8A	
22	17 16 15 14 13 12 11 10	89	
21	0F 0E 0D 0C 0B 0A 09 08	88	8F 8E 8D 8C 8B 8A 89 88
20	07 06 05 04 03 02 01 00	87	
1F	Bank 3	83	
18		82	
17	Bank 2	81	
10		80	87 86 85 84 83 82 81 80
0F	Bank 1		
08			
07	Bank thanh ghi 0 (mặc định cho gán cho R0 -R7)		
00			
	RAM		Các Thanh Ghi có Chức Năng Đặc Biệt

Hình 4-9: Cấu trúc bộ nhớ RAM bên trong vi điều khiển.



Hình 4-10. Minh họa cách gán bank thanh ghi cho nhóm thanh ghi R.

#### □ RAM có thể truy xuất từng bit:

Vi điều khiển 89C51 có 210 ô nhớ bit có thể truy xuất từng bit, trong đó có 128 bit nằm ở các ô nhớ byte có địa chỉ từ 20H đến 2FH và các bit còn lại chứa trong nhóm thanh ghi có chức năng đặc biệt.

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 duy nhất, trong khi đó để xử lý các bit thì *vi xử lý* vẫn có thể xử lý được nhưng phải sử dụng rất nhiều lệnh để đạt được cùng một kết quả vì *vi xử lý* thường xử lý byte.

Các port cũng có thể truy xuất được từng bit.

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

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

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

#### □ RAM đa dụng :

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

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

Bộ nhớ ngăn xếp của vi điều khiển dùng Ram nội nên dung lượng bộ nhớ ngăn xếp nhỏ, trong khi đó các vi xử lý dùng bộ nhớ bên ngoài làm bộ nhớ ngăn xếp nên dung lượng tùy ý mở rộng.

## 2 CÁC THANH GHI CÓ CHỨC NĂNG ĐẶC BIỆT:

Các thanh ghi nội của 89C51 được truy xuất ngầm định bởi bộ lệnh.

Các thanh ghi trong 89C51 được định dạng như một phần của RAM trên chip vì vậy mỗi thanh ghi sẽ có một địa chỉ (ngoại trừ thanh ghi bộ đếm chương trình và thanh ghi lưu trữ mã lệnh vì các thanh ghi này đã có chức năng cố định). Cũng như các thanh ghi R0 đến R7, vi điều khiển 89C51 có 21 thanh ghi có chức năng đặc biệt nằm ở vùng trên của RAM nội có địa chỉ từ 80H đến FFH.

**Chú ý:** 128 ô nhớ có địa chỉ từ 80H đến FFH thì chỉ có 21 thanh ghi có chức năng đặc biệt được xác định – còn các ô nhớ còn lại thì chưa thiết lập và trong tương lai sẽ được các nhà thiết kế vi điều khiển thiết lập thêm, khi đó sẽ có các vi điều khiển thế hệ mới hơn.

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

- **Các ô nhớ có địa chỉ 80H, 90H, A0H, B0H:**

Là các Port của 89C51 bao gồm Port0 có địa chỉ 80H, Port1 có địa chỉ 90H, Port2 có địa chỉ A0H và Port3 có địa chỉ B0H. Tất cả các Port này đều có thể truy xuất từng bit nên rất thuận tiện trong điều khiển. Địa chỉ của các bit được đặt tên với ô bắt đầu chính là địa chỉ của port tương ứng ví dụ như bit đầu tiên của port 0 là 80h cũng chính là địa chỉ bắt đầu của port 0. Người lập trình không cần nhớ địa chỉ các bit trong các port vì phần mềm lập trình cho phép truy xuất bằng tên từng bit dễ nhớ như sau: P0.0 chính là bit có địa chỉ 80h của port0.

Ngoài trừ thanh ghi A có thể được truy xuất ngầm, đa số các thanh ghi có chức năng đặc biệt SFR có thể địa chỉ hóa từng bit hoặc byte.

- **Ô nhớ có địa chỉ 81H:**

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

Các lệnh liên quan đến ngăn xếp bao gồm các lệnh cất dữ liệu vào ngăn xếp (lệnh Push) và lấy dữ liệu ra khỏi ngăn xếp (lệnh Pop).

Lệnh cất dữ liệu vào ngăn xếp sẽ làm tăng SP trước khi ghi dữ liệu vào.

Sau lệnh lấy dữ liệu ra khỏi ngăn xếp sẽ làm giảm SP.

Bộ nhớ ngăn xếp của 89C51 nằm trong RAM nội và bị giới hạn về cách truy xuất địa chỉ - chỉ cho phép truy xuất địa chỉ gián tiếp. Dung lượng bộ nhớ ngăn xếp lớn nhất là 128 byte ram nội của 89C51.

Khi Reset 89C51 thì thanh ghi SP sẽ mang giá trị mặc định là 07H và dữ liệu đầu tiên sẽ được cất vào ô nhớ ngăn xếp có địa chỉ 08H.

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

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 trả 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.

- **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 dữ liệu bên ngoài. Các vi điều khiển sau này có thêm thanh ghi dptr1.

- **Ô nhớ có địa chỉ 87H:**

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

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

Là các thanh ghi phục vụ cho 2 timer/counter T1, T0.

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

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

Thanh ghi TH0 và TL0 kết hợp lại tạo thành 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.

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

- **Các ô nhớ có địa chỉ 98H và 99H :**

Là 2 thanh ghi SCON và SBUF. SCON (series control): thanh ghi điều khiển truyền dữ liệu nối tiếp. Sbuf (series buffer): thanh ghi bộ 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.

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

- **Các ô nhớ có địa chỉ A8H và B8H :**

Là 2 thanh ghi IE và IP. Thanh ghi IE (Interrupt Enable): thanh ghi điều khiển cho phép / không cho phép ngắt. Thanh ghi IP (Interrupt Priority): thanh ghi điều khiển ưu tiên ngắt. Khi có sử dụng đến ngắt thì phải dùng đến 2 thanh ghi này. Mặc nhiên các thanh ghi này được khởi tạo ở chế độ cấm ngắt.

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

- **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 4-3:

BIT	SYMBOL	ADDRESS	DESCRIPTION
PSW.7	C	D7H	Carry Flag
PSW.6	AC	D6H	Auxiliary Carry Flag
PSW.5	F0	D5H	Flag 0 còn gọi là cờ Zero kí hiệu là Z
PSW4	RS1	D4H	Register Bank Select 1: bit lựa chọn bank thanh ghi.
PSW.3	RS0	D3H	Register Bank Select 0: bit lựa chọn bank thanh ghi.
			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.

Bảng 4-3. Các bit trong thanh ghi trạng thái.

Chức năng từng bit trạng thái:

- **Cờ Carry C (Carry Flag):**

Cờ nhớ có tác dụng kép. Cờ C được sử dụng cho các lệnh toán học:

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

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

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

Khi cộng những giá trị BCD (Binary Code Decimal), cờ nhớ phụ AC được set [AC=1] nếu kết quả 4 bit lớn hơn 09H, ngược lại AC= 0. Cờ AC được dùng để chỉnh số BCD khi thực hiện lệnh cộng 2 số BCD.

- **Cờ 0 (Flag 0):**

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

- **Các bit chọn bank thanh ghi truy xuất:**

Hai bit RS1 và RS0 dùng để đổi cách gán 8 thanh ghi R7 – R0 cho 1 trong 4 bank thanh ghi. Hai bit này sẽ bị xóa sau khi reset vi điều khiển và được thay đổi bởi chương trình của người lập trình.

RS1	RS0	Bank thanh ghi được lựa chọn
0	0	Bank 0

0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

Bảng 4-4. Các bit lựa chọn bank thanh ghi.

- **Cờ tràn OV (Over Flag) :**

Khi các số có dấu được cộng hoặc trừ với nhau, phần mềm có thể kiểm tra bit này để xác định xem kết quả có nằm trong vùng giá trị xác định hay không. Với số nhị phân 8 bit có dấu thì số dương từ 0 đến +127, số âm từ -128 đến -1. Nếu kết quả cộng 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) :**

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

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

- **Thanh ghi B :**

Thanh ghi B ở địa chỉ F0H được dùng cùng với thanh ghi A để thực hiện các phép toán nhân chia. Lệnh MUL AB: sẽ nhân những giá trị không dấu 8 bit với 8 bit trong hai thanh ghi A và B, rồi trả về kết quả 16 bit trong A (byte cao) và B (byte thấp). Lệnh DIV AB: lấy giá trị trong thanh ghi A chia cho giá trị trong thanh ghi B, kết quả nguyên lưu trong A, số dư lưu trong B.

Thanh ghi B có thể được dùng như một thanh ghi đệm trung gian nhiều chức năng.

### Tóm tắt

- ✚ Vi điều khiển có tất cả các thành phần cấu trúc bên trong giống như vi xử lý như khối ALU, các thanh ghi: thanh ghi A, thanh ghi PC, thanh ghi SP, thanh ghi trạng thái, ...
- ✚ Vi điều khiển có tích hợp bộ nhớ nội bộ bên trong bao gồm bộ nhớ chương trình và dữ liệu.
- ✚ Bộ nhớ chương trình dùng để chứa chương trình điều khiển.
- ✚ Bộ nhớ dữ liệu là Ram dùng để lưu trữ dữ liệu phục vụ cho việc xử lý chương trình.
- ✚ Kích thước của bộ nhớ chương trình bên trong tùy thuộc vào từng loại vi điều khiển cụ thể.
- ✚ Ngoài các bộ nhớ bên trong vi điều khiển còn có thể giao tiếp với bộ nhớ mở rộng bên ngoài. Khi giao tiếp với bộ nhớ bên ngoài thì port 0 và port 2 có chức năng giao tiếp địa chỉ, dữ liệu và các đường điều khiển của port 3, số lượng đường điều khiển IO còn lại rất ít. Muốn có thêm nhiều đường điều khiển IO thì phải giao tiếp thêm IC ngoại vi.

- ✚ Vi điều khiển có 32 đường IO, có timer/counter, có truyền dữ liệu nối tiếp, có ngắt cho phép giao tiếp dễ dàng với đối tượng điều khiển.
- ✚ Một trong những thế mạnh của vi điều khiển là khả năng xử lý dữ liệu bit – rất phù hợp trong lãnh vực điều khiển.

## IV. TẬP LỆNH VI ĐIỀU KHIỂN MCS51

### 1. GIỚI THIỆU

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

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

Các khái niệm về chương trình, lệnh, tập lệnh và ngôn ngữ gợi nhớ đã trình bày ở chương 1 và 2, ở đây chỉ tóm tắt lại.

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

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

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

Họ vi điều khiển MCS-51 đều có chung 1 tập lệnh, các vi điều khiển thế hệ sau chỉ phát triển nhiều về phần cứng còn lệnh thì ít mở rộng.

Tập lệnh họ MCS-51 có mã lệnh 8 bit nên có khả năng cung cấp  $2^8 = 256$  lệnh.

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

Trong toàn bộ tập lệnh của vi điều khiển có 139 lệnh 1 byte, 92 lệnh 2 byte và 24 lệnh 3 byte.

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

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

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

## 2. CÁC KIỂU ĐỊNH ĐỊA CHỈ CỦA VI ĐIỀU KHIỂN MCS51:

Phần này đã trình bày một cách tổng quát ở chương 2, ở đây sẽ trình bày một cách chi tiết hơn. Các kiểu định địa chỉ là một qui ước thống nhất của tập lệnh.

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

Vi điều khiển họ MCS-51 có 8 kiểu định địa chỉ như sau:

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

### a. Kiểu định địa chỉ dùng thanh ghi (Register Addressing) :

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

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

### b. 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 RAM nội của vi điều khiển 89C51.

Các lệnh thuộc kiểu này thường có mã lệnh 2 byte: byte thứ nhất là mã lệnh, byte thứ 2 là **địa chỉ của ô nhớ**:

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

### c. 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 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. Còn dptr dùng để truy xuất ô nhớ ngoại. Các lệnh thuộc dạng này chỉ có 1 byte.

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

### d. 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 ở byte thứ 2 (hoặc byte thứ 3) vào thanh ghi hoặc ô nhớ.

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

#### e. Định địa chỉ tương đối :

Kiểu định địa chỉ tương đối chỉ sử dụng với những lệnh nhảy. Nơi nhảy đến có địa chỉ bằng địa chỉ đang lưu trong thanh ghi PC cộng với 1 giá trị 8 bit [còn gọi là giá trị lệch tương đối: relative offset] có giá trị từ -128 đến +127 nên vi điều khiển có thể nhảy lùi [nếu số cộng với số âm] và nhảy tới [nếu số cộng với số dương]. Lệnh này có mã lệnh 2 byte, byte thứ 2 chính là giá trị lệch tương đối.

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

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

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

#### f. Định địa chỉ tuyệt đối (Absolute Addressing) :

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

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

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

#### g. 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 – trong đó có 2 byte (16bit) là địa chỉ của nơi đến. Cấu trúc mã lệnh là 3 byte như sau:

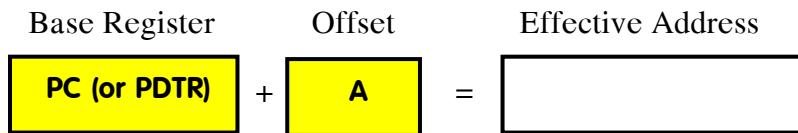
Ưu điểm của định địa chỉ dài là có thể gọi 1 chương trình con hoặc có thể nhảy đến bất kỳ vùng nhớ nào vùng nhớ 64K, nhược điểm là các lệnh kiểu này dài 3 byte và phụ thuộc vào vị trí đến – điều này sẽ bất tiện bởi không thể dời toàn bộ mã lệnh của chương trình từ vùng nhớ này sang các vùng nhớ khác – có nghĩa là khi chương trình đã viết nơi đến tại địa chỉ 1000h thì sau khi dịch ra mã lệnh dạng số nhị phân thì sau đó nạp vào bộ nhớ thì địa chỉ bắt đầu phải đúng với địa chỉ đã viết là 1000h; nếu nạp ở vùng địa chỉ khác địa chỉ 1000h thì chương trình sẽ thực hiện sai.

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

#### h. 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 lệch (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 dụ8: MOVX A, @A + D PTR ;lấy dữ liệu trong ô nhớ có địa chỉ bằng DPTR + A

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

### 3. KHẢO SÁT TẬP LỆNH VI ĐIỀU KHIỂN MCS51:

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 tương trưng cho ô nhớ nội có địa chỉ **bất kỳ từ có địa chỉ từ 00H đến FFH**.
- **Rn** tương trưng cho các thanh ghi từ thanh ghi R0 đến thanh ghi R7.
- **@Ri** tương trưng cho ô nhớ có địa chỉ lưu trong thanh ghi Ri và chỉ có 2 thanh ghi là R0 và R1.
- Các lệnh thường xảy ra giữa các đối tượng sau:
  - + Thanh ghi A.
  - + Thanh ghi Rn.
  - + Ô nhớ có địa chỉ direct.
  - + Ô nhớ có địa chỉ lưu trong thanh ghi @Ri.
  - + Dữ liệu 8 bit **#data**.
  - + addr11 là địa chỉ 11 bit từ A10÷A0: địa chỉ này phục vụ cho lệnh nhảy hoặc lệnh gọi chương trình con trong phạm vi 2 kbyte.
  - + Addr16 là địa chỉ 16 bit từ A15÷A0: địa chỉ này phục vụ cho lệnh nhảy và lệnh gọi chương trình con ở xa trong phạm vi 64 kbyte – đó chính là địa chỉ nhảy đến, hoặc địa chỉ của chương trình con.

Khi viết chương trình người lập trình có thể thay thế địa chỉ bằng nhãn (label) để khỏi phải tính toán các địa chỉ cụ thể. Nhãn (label) sẽ được đặt tại vị trí addr thay cho addr và phải có một nhãn đặt tại nơi muốn nhảy đến - gọi là 1 cặp nhãn cùng tên.

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

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

#### a. Nhóm lệnh di chuyển dữ liệu (8 bit) :

##### 1. Lệnh chuyển dữ liệu từ một thanh ghi vào thanh ghi A:

- Cú pháp : **Mov A,Rn**

- Mã lệnh :

1	1	1	0	1	n2	n1	n0
---	---	---	---	---	----	----	----

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng:** Chuyển nội dung của thanh ghi Rn vào thanh ghi A, nội dung thanh ghi Rn vẫn giữ nguyên.

**Ví dụ 1:** Giả sử thanh ghi R0 có nội dung là 32h , lệnh:

MOV A,R0 ;kết quả như sau: (A) = 32h, (R0) = 32h.

Giá trị ban đầu chứa trong A thì không cần quan tâm.

## 2. Lệnh chuyển dữ liệu từ ô nhớ trực tiếp vào thanh ghi A :

- Cú pháp : **Mov A, direct**

- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng:** Chuyển nội dung của ô nhớ trong Ram nội có địa chỉ direct ở byte thứ hai vào thanh ghi A. Trực tiếp có nghĩa là địa chỉ của ô nhớ được ghi ở trong lệnh.

**Ví dụ 2:** Giả sử ô nhớ có địa chỉ 30h lưu nội dung 32h. Lệnh:

MOV A,**30H** ;chuyển nội dung của ô nhớ có địa chỉ là 30h sang thanh ghi A.

;Kết quả như sau: (A)= 32h. Chú ý địa chỉ **30h** ghi trong lệnh.

;a7..a0 = 00110000b là địa chỉ của ô nhớ 30h

## 3. Lệnh chuyển dữ liệu từ ô nhớ gián tiếp vào thanh ghi A :

- Cú pháp : **MOV A,@Ri**

- Mã lệnh :

1	1	1	0	0	1	1	i
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng:** Chuyển nội dung ô nhớ trong Ram nội, có địa chỉ chứa trong thanh ghi Ri, vào thanh ghi A.

**Ví dụ 3:** thay vì thực hiện “mov A,30h” của ví dụ trên thì ta có thể thay bằng lệnh “mov A,@R0” sẽ có cùng 1 kết quả nếu địa chỉ 30h lưu vào R0. Địa chỉ 30h không còn ghi trong lệnh mà được thay bằng @R0 – nên kiểu lệnh này gọi là lệnh gián tiếp vì địa chỉ 30h không còn xuất hiện trong lệnh. Chú ý trước khi sử dụng lệnh này ta phải làm cho R0 mang giá trị là 30h.

**Ví dụ 4:** Giả sử R0 có nội dung là 70h, ô nhớ có địa chỉ 70h chứa nội dung là 0B8h. Lệnh:

MOV A,@R0 ;kết quả như sau: (A) = 0B8h.

## 4. Lệnh nạp dữ liệu 8 bit vào thanh ghi A :

- Cú pháp : **MOV A, #data**

- Mã lệnh :

0	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---

d7	d6	d5	d4	d3	d2	d1	d0
----	----	----	----	----	----	----	----

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nạp dữ liệu 8 bit data (d0 đến d7) vào thanh ghi A.

**Ví dụ 5:** Giả sử A có nội dung 47h, dữ liệu trực tiếp là 32h, lệnh:

MOV A,#32H ;kết quả như sau: (A) = 32h.  
;d7..d0 = 00110010b

### 5. Lệnh chuyển dữ liệu từ thanh ghi A vào thanh ghi :

- Cú pháp : **Mov Rn, A**
- Mã lệnh :

1	1	1	1	1	n2	n1	n0
---	---	---	---	---	----	----	----

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Chuyển nội dung của thanh ghi A vào thanh ghi Rn.

**Ví dụ 6:** Giả sử A có nội dung 47h , lệnh:

MOV R0, A ;kết quả như sau: (R0) = 47h, (A) = 47h.

### 6. Lệnh chuyển dữ liệu từ ô nhớ trực tiếp vào thanh ghi Rn :

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

1	0	1	0	1	n2	n1	n0
a7	a6	a5	a4	a3	a2	a1	a0

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Chuyển nội dung của ô nhớ trong Ram nội bộ có địa chỉ direct vào thanh ghi Rn.

**Ví dụ 7:** Giả sử R1 có nội dung 47h, ô nhớ có địa chỉ 30h chứa nội dung 0AFh. Lệnh:

MOV R1,30H

Lệnh chuyển nội dung ô nhớ có địa chỉ 30h sang thanh ghi R1.

Kết quả như sau: (R1) = 0AFh, dữ liệu trong ô nhớ có địa chỉ 30h không đổi.

### 7. Lệnh chuyển tức thời dữ liệu 8 bit vào thanh ghi Rn :

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

0	1	1	1	1	n2	n1	n0
d7	d6	d5	d4	d3	d2	d1	d0

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.
- Chức năng: Nạp dữ liệu 8 bit data (d0 đến d7) vào thanh ghi Rn.

**Ví dụ 8:** Giả sử muốn chuyển dữ liệu 47h vào thanh ghi R1:

MOV R1,#47H ;kết quả như sau: (R1)= 47h.

### 8. Lệnh chuyển dữ liệu từ thanh ghi A vào ô nhớ trực tiếp :

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

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.
- Chức năng: Chuyển nội dung của thanh ghi A vào ô nhớ trong Ram nội có địa chỉ direct.

**Ví dụ 9:** Cho nội dung thanh ghi (A) = 35H, nội dung ô nhớ có địa chỉ 10H bằng 50H.

MOV 10H,A

Sau khi thực hiện xong thì nội dung ô nhớ có địa chỉ 10H bằng 35H.

#### 9. Lệnh chuyển dữ liệu từ thanh ghi Rn vào ô nhớ trực tiếp :

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

1	0	0	0	1	n2	n1	n0
a7	a6	a5	a4	a3	a2	a1	a0

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: Chuyển nội dung của thanh ghi Rn vào ô nhớ trong Ram nội có địa chỉ direct.

**Ví dụ 10:** Cho nội dung thanh ghi (R0) = 35H, nội dung ô nhớ 10H bằng 50H.

MOV 10H,R0

Sau khi thực hiện xong thì nội dung ô nhớ có địa chỉ 10H bằng 35H.

#### 10. Lệnh chuyển dữ liệu từ ô nhớ trực tiếp vào ô nhớ trực tiếp :

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

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

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: Chuyển nội dung của ô nhớ trong Ram nội có địa chỉ direct vào ô nhớ có địa chỉ trực tiếp.

**Ví dụ 11:** Cho nội dung ô nhớ có địa chỉ 20H bằng 35H và nội dung ô nhớ có địa chỉ 10H bằng 50H.

MOV 10H,20H

Sau khi thực hiện xong thì nội dung ô nhớ có địa chỉ 10H bằng 35H.

#### 11. Lệnh chuyển dữ liệu từ ô nhớ gián tiếp vào ô nhớ trực tiếp :

- Cú pháp : **MOV direct, @Ri**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: Chuyển nội dung ô nhớ có địa chỉ chứa trong thanh ghi Ri vào ô nhớ có địa chỉ direct.

**Ví dụ 12:** Cho nội dung thanh ghi (R0) = 05H, nội dung ô nhớ có địa chỉ 05h bằng FFH và nội dung ô nhớ có địa chỉ 10H bằng 50H.

MOV 10H,@R0

Sau khi thực hiện xong thì nội dung ô nhớ có địa chỉ 10h bằng FFH.

#### 12. Lệnh chuyển dữ liệu vào ô nhớ trực tiếp :

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

0	1	1	1	0	1	0	1
a7	a6	a5	a4	a3	a2	a1	a0
d7	d6	d5	d4	d3	d2	d1	d0

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: Nạp dữ liệu data 8 bit (d0 đến d7) vào ô nhớ có địa chỉ direct.

**Ví dụ 13:** Cho nội dung ô nhớ có địa chỉ 05h bằng FFH.

MOV 05H,#25H

Sau khi thực hiện xong thì nội dung ô nhớ có địa chỉ 05h bằng 25H.

#### 13. Lệnh chuyển dữ liệu từ thanh ghi A vào ô nhớ gián tiếp :

- Cú pháp : **MOV @Ri, A**
- Mã lệnh :

1	1	1	1	0	1	1	i
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng : Chuyển nội dung của thanh ghi A vào ô nhớ trong Ram nội có địa chỉ chứa trong thanh ghi Ri.

#### 14. Lệnh chuyển dữ liệu từ ô nhớ trực tiếp vào ô nhớ gián tiếp :

- Cú pháp : **MOV @Ri, direct**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: Chuyển nội dung ô nhớ có địa chỉ direct vào ô nhớ có địa chỉ chứa trong thanh ghi Ri.

#### 15. Lệnh chuyển dữ liệu tức thời vào ô nhớ gián tiếp :

- Cú pháp : **MOV @Ri, #data**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nạp dữ liệu data 8 bit (d0 đến d7) vào ô nhớ có địa chỉ chứa trong thanh ghi Ri.

**16. Lệnh chuyển dữ liệu tức thời 16 bit vào thanh ghi con trả dữ liệu :**

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

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

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: Nạp dữ liệu data 16 bit vào thanh ghi con trả dữ liệu dptr.

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

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

1	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng : chuyển nội dung của ô nhớ trong bộ nhớ chương trình (Code Memory), có địa chỉ chứa bằng dptr cộng với giá trị chứa trong A, chuyển vào thanh ghi A.

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

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

1	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng : chuyển nội dung của ô nhớ trong bộ nhớ chương trình (Code Memory) có địa chỉ chứa bằng PC cộng với giá trị chứa trong A được chuyển vào thanh ghi A.

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

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

1	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng : chuyển nội dung ô nhớ ngoài có địa chỉ chứa trong thanh ghi Ri vào thanh ghi A.

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

- Cú pháp : **MOVX A,@DPTR**

- Mã lệnh :

1	1	1	0	0	0	0	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng : chuyển nội dung của ô nhớ ngoài có địa chỉ chứa trong thanh ghi dptr vào thanh ghi A.

#### 21. Lệnh chuyển dữ liệu từ thanh ghi A vào ô nhớ ngoài gián tiếp (8 bit địa chỉ) :

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

1	1	1	1	0	0	1	i
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng : chuyển nội dung của thanh ghi A ra ô nhớ ngoài có địa chỉ chứa trong thanh ghi Ri.

#### 22. Lệnh chuyển dữ liệu từ thanh ghi A vào ô nhớ ngoài gián tiếp (16 bit địa chỉ) :

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

1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng : Chuyển nội dung của thanh ghi A ra ô nhớ ngoài có địa chỉ chứa trong thanh ghi dptr.

#### 23. Lệnh cất nội dung ô nhớ trực tiếp vào ngăn xếp :

- Cú pháp : **PUSH direct**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: cất nội dung của ô nhớ có địa chỉ direct vào ô nhớ ngăn xếp. Con trỏ ngăn xếp SP tăng lên 1 trước khi lưu nội dung.

#### 24. Lệnh lấy dữ liệu từ ngăn xếp trả về ô nhớ trực tiếp :

- Cú pháp : **POP direct**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: lấy nội dung của ô nhớ ngăn xếp trả cho ô nhớ có địa chỉ direct. Con trỏ ngăn xếp SP giảm 1 sau khi lấy dữ liệu ra.

#### 25. Lệnh trao đổi dữ liệu giữa thanh ghi với thanh ghi A :

- Cú pháp : **XCH A,Rn**
- Mã lệnh :

1	1	0	0	1	n2	n1	n0
---	---	---	---	---	----	----	----

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng : Trao đổi nội dung của thanh ghi Rn với thanh ghi A.

**Ví dụ 14:** cho nội dung của thanh ghi (A) = 35H và (R0) = 70H

XCH A,R0

Kết quả sau khi thực hiện (A) = 70H và (R0) = 35H

26. Lệnh trao đổi dữ liệu giữa ô nhớ trực tiếp với thanh ghi A :

- Cú pháp : **XCH A,Direct**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng : Trao đổi nội dung của thanh ghi A với nội dung ô nhớ có địa chỉ direct.

27. Lệnh trao đổi dữ liệu giữa ô nhớ gián tiếp với thanh ghi A :

- Cú pháp : **XCH A,@Ri**
- Mã lệnh :

1	1	0	0	0	1	1	i
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng : Trao đổi nội dung của ô nhớ có địa chỉ chứa trong thanh ghi Ri với thanh ghi A.

28. Lệnh trao đổi 4 bit dữ liệu giữa ô nhớ gián tiếp với thanh ghi A :

- Cú pháp : **XCHD A,@Ri**
- Mã lệnh :

1	1	0	1	0	1	1	i
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng : Trao đổi dữ liệu 4 bit thấp của ô nhớ có địa chỉ chứa trong thanh ghi Ri với dữ liệu 4 bit thấp trong thanh ghi A.

## b. Nhóm lệnh số học :

1. Lệnh cộng thanh ghi A với thanh ghi :

- Cú pháp : **ADD A,Rn**
- Mã lệnh :

0	0	1	0	1	n2	n1	n0
---	---	---	---	---	----	----	----

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng : cộng nội dung thanh ghi A với nội dung thanh ghi Rn, kết quả lưu trong thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

**Ví dụ 15:** Giả sử A có nội dung 47h và R0 có nội dung là 32h, lệnh:

ADD A,R0 ;kết quả như sau: (A) = 79h, (C) = 0.

**Ví dụ 16:** Giả sử A có nội dung 0D9h và R0 có nội dung là 0B8h, lệnh:  
ADD A,R0 ;kết quả như sau: (A) = 91h, (C) =1.

## 2. Lệnh cộng nội dung ô nhớ trực tiếp vào thanh ghi A :

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

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- **Chức năng:** Cộng nội dung của ô nhớ có địa chỉ direct với nội dung thanh ghi A, kết quả chứa ở thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

**Ví dụ 17:** Giả sử A có nội dung 0D9h và ô nhớ có địa chỉ 30h lưu nội dung 0B8h, lệnh:  
ADD A,30h ;kết quả như sau: (A) = 81h, (C) =1.

**Ví dụ 18:** Giả sử A có nội dung 47h và ô nhớ có địa chỉ 30h lưu nội dung 32h, lệnh:  
ADD A,30h ;kết quả như sau: (A) = 79h, (C) =0.

## 3. Lệnh cộng nội dung ô nhớ gián tiếp vào thanh ghi A :

- Cú pháp: **ADD A,@Ri**
- Mã lệnh:

0	0	1	0	0	1	1	i
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- **Chức năng:** 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. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

**Ví dụ 19:** Giả sử A có nội dung 0D9h, ô nhớ có địa chỉ 30h có nội dung là 0B8h, R0 có nội dung là 30h, lệnh:

ADD A,@R0 ;kết quả như sau: (A) = 91h, (C) =1.

## 4. Lệnh cộng dữ liệu tức thời 8 bit vào thanh ghi A :

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

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- **Chức năng:** Cộng dữ liệu data 8 bit (d0 đến d7) với nội dung thanh ghi A, kết quả lưu trữ trong A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

**Ví dụ 20:** Giả sử A có nội dung 47h, dữ liệu trực tiếp là 32h, lệnh:  
ADD A,#32h ;kết quả như sau: (A) = 79h, (C) = 0.

**Ví dụ 21:** Giả sử A có nội dung D9h, dữ liệu trực tiếp là B8h, lệnh:  
ADD A,#0B8h ;kết quả như sau: (A) = 91h, (C) = 1.

## 5. Lệnh cộng thanh ghi A với thanh ghi có bit carry :

- Cú pháp : **ADDC A,Rn**

- Mã lệnh :

0	0	1	1	1	n2	n1	n0
---	---	---	---	---	----	----	----

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng : cộng nội dung thanh ghi A với nội dung thanh ghi Rn với bit C, kết quả lưu trong thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

**Ví dụ 22:** Giả sử A có nội dung 47h, R1 có nội dung 32h và cờ (C) = 1, lệnh:

ADDC A,R1 ;kết quả như sau: (A) = 7ah, (C) = 0.

**Ví dụ 23:** Giả sử A có nội dung 0D9h, R0 có nội dung là 0B8h, (C) =1, lệnh:

ADDC A,R0 ;kết quả như sau: (A) = 92h, (C)=1.

#### 6. Lệnh cộng nội dung ô nhớ trực tiếp vào thanh ghi A có bit carry :

- Cú pháp : **ADDC A, direct**

- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: 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. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

**Ví dụ 24:** Giả sử A có nội dung 47h, ô nhớ 30h có nội dung 32h và cờ (C) = 0, lệnh:

ADDC A,30h ;kết quả như sau: (A) = 79h, (C) = 0.

**Ví dụ 25:** Giả sử A có nội dung 0D9h, ô nhớ 30h có nội dung là 0B8h, C:=1, lệnh:

ADDC A,30h ;kết quả như sau: (A) = 92h, (C) = 1.

#### 7. Lệnh cộng nội dung ô nhớ gián tiếp vào thanh ghi A có bit carry :

- Cú pháp : **ADDC A,@Ri**

- Mã lệnh :

0	0	1	1	0	1	1	i
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng : 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. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

**Ví dụ 26:** Giả sử A có nội dung 47h, ô nhớ 30h có nội dung 32h, R0 có nội dung là 30h và cờ (C) = 0, lệnh:

ADDC A,@R0 ;kết quả như sau: (A) = 79h, (C) = 0.

**Ví dụ 27:** Giả sử A có nội dung 0D9h, ô nhớ 30h có nội dung là 0B8h, R0 có nội dung 30h và (C) =1, lệnh:

ADDC A,@R0 ;kết quả như sau: (A) = 92h, (C)=1.

## 8. Lệnh cộng dữ liệu 8 bit vào thanh ghi A có bit carry :

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

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.
- Chức năng: Cộng dữ liệu data 8 bit (d0 đến d7) với nội dung thanh ghi A và bit C, kết quả lưu trữ trong A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

## 9. Lệnh trừ thanh ghi A với thanh ghi :

- Cú pháp : **SUBB A,Rn**
- Mã lệnh :

1	0	0	1	1	n2	n1	n0
---	---	---	---	---	----	----	----

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng : Trừ nội dung thanh ghi A với nội dung thanh ghi Rn và trừ cho cờ Carry, kết quả lưu trong thanh ghi A. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

**Ví dụ 28:** Giả sử A có nội dung 47h, nội dung thanh ghi R0 là 32h và cờ (C)=0, lệnh:

SUBB A,R0 ;kết quả như sau: (A) = 15h (C)=0.

**Ví dụ 29:** Giả sử A có nội dung 0B9h, thanh ghi R0 có nội dung là 5Ah và (C)=1, lệnh:

SUBB A,R0 ;kết quả như sau: (A) = 5Eh, (C) =0.

## 10. Lệnh trừ nội dung thanh ghi A cho nội dung ô nhớ trực tiếp :

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

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: 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. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

## 11. Lệnh trừ nội dung thanh ghi A cho nội dung ô nhớ gián tiếp :

- Cú pháp : **SUBB A,@Ri**
- Mã lệnh :

1	0	0	1	0	1	1	i
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: 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. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

## 12. Lệnh trừ nội dung thanh ghi A cho dữ liệu tức thời 8 bit :

- Cú pháp : **SUBB A, #data** (subtract: trừ)
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: 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. Lệnh có ảnh hưởng đến thanh ghi trạng thái.

#### 13. Lệnh tăng nội dung thanh ghi A :

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

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Tăng nội dung thanh ghi A lên 1.

**Ví dụ 30:** Giả sử A có nội dung 35h, lệnh:

INC A ;kết quả như sau: (A) = 36h.

**Ví dụ 31:** Giả sử A có nội dung FFh, lệnh:

INC A ;kết quả như sau: (A) = 00h.

#### 14. Lệnh tăng nội dung của thanh ghi :

- Cú pháp : **INC Rn**
- Mã lệnh :

0	0	0	0	1	n2	n1	n0
---	---	---	---	---	----	----	----

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Tăng nội dung thanh ghi Rn lên 1.

#### 15. Lệnh tăng nội dung ô nhớ trực tiếp :

- Cú pháp : **INC direct**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Tăng nội dung của ô nhớ có địa chỉ trực tiếp ở byte thứ 2 lên 1.

#### 16. Lệnh tăng nội dung ô nhớ gián tiếp :

- Cú pháp : **INC @Ri**
- Mã lệnh :

0	0	0	0	0	1	1	i
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Tăng nội dung của ô nhớ có địa chỉ chứa trong thanh ghi Ri lên 1.

**Ví dụ 32:** Giả sử nội dung ô nhớ 30h là 35h, thanh ghi R0 có nội dung là 30h, lệnh:

INC R0 ;kết quả như sau: ô nhớ (30h) = 36h

### 17. Lệnh tăng nội dung con trỏ dữ liệu Dptr :

- Cú pháp : **INC dptr**
- Mã lệnh :

1	0	1	0	0	0	1	1
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: Tăng nội dung của thanh ghi con trỏ dữ liệu dptr lên 1.

### 18. Lệnh giảm nội dung thanh ghi A :

- Cú pháp : **DEC A**
- Mã lệnh :

0	0	0	1	0	1	0	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Giảm nội dung thanh ghi A xuống 1.

**Ví dụ 33:** Giả sử A có nội dung 35h, lệnh:

DEC A ;kết quả như sau: (A) = 34h.

**Ví dụ 34:** Giả sử A có nội dung 00h, lệnh:

DEC A ;kết quả như sau: (A) = FFh.

### 19. Lệnh giảm nội dung của thanh ghi :

- Cú pháp : **DEC Rn**
- Mã lệnh :

0	0	0	1	1	n2	n1	n0
---	---	---	---	---	----	----	----

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy.
- Chức năng: Giảm nội dung thanh ghi Rn xuống 1.

**Ví dụ 351:** Giả sử R0 là 35h , lệnh:

DEC R0 ;kết quả như sau: (R0) =34h.

### 20. Lệnh giảm nội dung ô nhớ trực tiếp :

- Cú pháp : **DEC direct**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Giảm nội dung của ô nhớ có địa chỉ direct ở byte thứ 2 xuống 1.

**Ví dụ 36:** Giả sử ô nhớ 30h có nội dung là 35h , lệnh:

DEC 30h ;kết quả như sau: ô nhớ có địa chỉ là 30h lưu 34h.

## 21. Lệnh giảm nội dung ô nhớ gián tiếp :

- Cú pháp : **DEC @Ri**
- Mã lệnh :

0	0	0	1	0	1	1	i
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Giảm nội dung của ô nhớ có địa chỉ chứa trong thanh ghi Ri xuống 1.

## 22. Lệnh nhân thanh ghi A với thanh ghi B :

- Cú pháp : **MUL AB**
- Mã lệnh :

1	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 4 chu kỳ máy
- Chức năng: Nội dung của thanh ghi A nhân với nội dung của thanh ghi B, kết quả là một dữ liệu 16 bit, 8 bit thấp lưu trữ trong thanh ghi A, 8 bit cao lưu trữ trong thanh ghi B.

**Ví dụ 37:** Giả sử thanh ghi A có nội dung là 50h, thanh ghi B có nội dung 0A0h, lệnh:

MUL AB ;Kết quả như sau: 50h \* A0h = 3200h thì (A) = 00 và (B) = 32h.

## 23. Lệnh chia thanh ghi A cho thanh ghi B :

- Cú pháp : **DIV AB**
- Mã lệnh :

1	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 4 chu kỳ máy
- Chức năng: 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. Lệnh ảnh hưởng đến thanh ghi trạng thái: Bit C và bit OV bị xóa về 0, nếu phép chia này mà dữ liệu trong thanh ghi B = 00h thì nội dung thanh ghi A không thay đổi, nội dung chứa trong thanh ghi B không xác định và bit OV = 1, bit Cy = 0.

## 24. Lệnh điều chỉnh thập phân nội dung thanh ghi A :

- Cú pháp : **DA A**
- Mã lệnh :

1	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 4 chu kỳ máy
- Chức năng: 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 thanh ghi A. Kết quả sau cùng trong thanh ghi A là số BCD.

**c. Nhóm lệnh logic :**

## 1. Lệnh AND thanh ghi A với thanh ghi :

- Cú pháp : **ANL A,Rn (and logic)**
  - Mã lệnh :
- |   |   |   |   |   |    |    |    |
|---|---|---|---|---|----|----|----|
| 0 | 1 | 0 | 1 | 1 | n2 | n1 | n0 |
|---|---|---|---|---|----|----|----|
- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
  - Chức năng: Nội dung thanh ghi A and với nội dung thanh ghi Rn, kết quả lưu trữ trong thanh ghi A.

**Ví dụ 38:**

MOV A ,#10110011b

MOV R0,#11001011b

ANL A,R0 ;kết quả (A) = 10000011b

**2. Lệnh and thanh ghi A với nội dung ô nhớ trực tiếp :**

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

0	1	0	1	0	1	0	1
a7	A6	a5	a4	a3	A2	a1	a0

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung thanh ghi A and với nội dung của ô nhớ có địa chỉ direct, kết quả chứa ở thanh ghi A.

**3. Lệnh and thanh ghi A với nội dung ô nhớ gián tiếp :**

- Cú pháp : **ANL A, @Ri**
- Mã lệnh :

0	1	0	1	0	1	1	i
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: 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.

**4. Lệnh and thanh ghi A với dữ liệu tức thời 8 bit :**

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

0	1	0	1	0	1	0	0
d7	d6	d5	d4	d3	D2	d1	d0

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung của thanh ghi A and với dữ liệu d0 đến d7 , kết quả lưu trữ trong thanh ghi A.

**Ví dụ 39:**

MOV A ,#10110011b

ANL A,#00001111b ;kết quả (A) = 00000011b

5. Lệnh and nội dung ô nhớ trực tiếp với nội dung thanh ghi A :

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

0	1	0	1	0	0	1	0
a7	a6	A5	a4	a3	A2	a1	a0

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung ô nhớ có địa chỉ direct and với nội dung của thanh ghi A, kết quả lưu trữ vào ô nhớ.

**Ví dụ 40:**

```
MOV A ,#10110011b
MOV 10h,#11110000b
ANL 10H,A ;kết quả ô nhớ có địa chỉ 10h lưu 10110000b.
```

6. Lệnh and nội dung ô nhớ trực tiếp với dữ liệu tức thời 8 bit :

- Cú pháp : **ANL direct, #data**
- Mã lệnh :

0	1	0	1	0	0	1	1
a7	a6	a5	a4	a3	A2	a1	a0
d7	d6	d5	d4	d3	d2	d1	d0

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: Nội dung của ô nhớ có địa chỉ direct and với 8 bit dữ liệu 8 bit, kết quả lưu trữ vào ô nhớ.

7. Lệnh or thanh ghi A với thanh ghi :

- Cú pháp : **ORL A, Rn**
- Mã lệnh :

0	1	0	0	1	n2	n1	n0
---	---	---	---	---	----	----	----

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung thanh ghi A or với nội dung thanh ghi Rn, kết quả lưu trữ trong thanh ghi A.

**Ví dụ 41:**

```
MOV A ,#10110011b
MOV R0,#11001011b
ORL A,R0 ;kết quả (A) = 11111011b.
```

8. Lệnh or thanh ghi A với nội dung ô nhớ trực tiếp :

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

0	1	0	0	0	1	0	1
---	---	---	---	---	---	---	---

a7	a6	a5	a4	a3	a2	a1	a0
----	----	----	----	----	----	----	----

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung thanh ghi A or với nội dung của ô nhớ có địa chỉ direct, kết quả chứa ở thanh ghi A.

#### 9. Lệnh or thanh ghi A với nội dung ô nhớ gián tiếp :

- Cú pháp : **ORL A, @Ri**
- Mã lệnh :

0	1	0	0	0	1	1	i
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: 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.

#### 10. Lệnh or thanh ghi A với dữ liệu tức thời 8 bit :

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

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: 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.

#### 11. Lệnh or nội dung ô nhớ trực tiếp với nội dung thanh ghi A :

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

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: 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.

#### 12. Lệnh or nội dung ô nhớ trực tiếp với dữ liệu tức thời 8 bit :

- Cú pháp : **ORL direct, #data**
- Mã lệnh :

0	1	0	0	0	0	1	1
a7	a6	a5	a4	a3	a2	a1	a0
d7	d6	d5	d4	d3	d2	d1	d0

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: 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.

#### 13. Lệnh ex-or thanh ghi A với thanh ghi :

- Cú pháp : **XRL A, Rn**

- Mã lệnh :

0	1	1	0	1	n2	n1	n0
---	---	---	---	---	----	----	----

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung thanh ghi A ex-or với nội dung thanh ghi Rn, kết quả lưu trữ trong thanh ghi A.

#### 14. Lệnh ex-or thanh ghi A với nội dung ô nhớ trực tiếp :

- Cú pháp : **XRL A, direct**

- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung thanh ghi A ex-or với nội dung của ô nhớ có địa chỉ direct, kết quả chứa ở thanh ghi A.

#### 15. Lệnh ex-or thanh ghi A với nội dung ô nhớ gián tiếp :

- Cú pháp : **XRL A, @Ri**

- Mã lệnh :

0	1	1	0	0	1	1	i
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: 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.

#### 16. Lệnh ex-or thanh ghi A với dữ liệu tức thời 8 bit :

- Cú pháp : **XRL A, #data**

- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung của thanh ghi A ex-or với dữ liệu data, kết quả lưu trữ trong thanh ghi A.

#### 17. Lệnh ex-or nội dung ô nhớ trực tiếp với nội dung thanh ghi A :

- Cú pháp : **XRL direct, A**

- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung ô nhớ có địa chỉ direct ex-or với nội dung của thanh ghi A, kết quả lưu trữ vào ô nhớ.

#### 18. Lệnh ex-or nội dung ô nhớ trực tiếp với dữ liệu tức thời 8 bit :

- Cú pháp : **XRL direct, #data**

- Mã lệnh :

0	1	1	0	0	0	1	1
a7	a6	a5	a4	a3	a2	a1	a0
d7	d6	d5	d4	d3	d2	d1	d0

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: 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.

#### 19. Lệnh xóa nội dung thanh ghi A :

- Cú pháp :

**CLR A**

(clear a)

- Mã lệnh :

1	1	1	0	0	1	0	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung thanh ghi A bằng zero.

#### 20. Lệnh bù nội dung thanh ghi A :

- Cú pháp :

**CPL A**

(complement A)

- Mã lệnh :

1	1	1	1	0	1	0	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung thanh ghi A được lấy bù 1, kết quả chứa trong A.

#### Ví dụ 42:

MOV A, #10110011b

CPL A ;kết quả (A) = 01001100b.

#### 21. Lệnh xoay trái nội dung thanh ghi A :

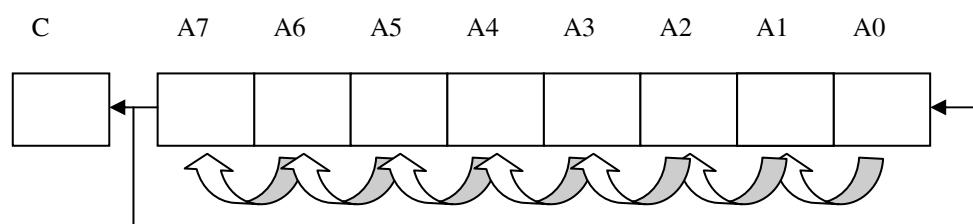
- Cú pháp :

**RL A** (rotate left)

- Mã lệnh :

0	0	1	0	0	0	1	1
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung thanh ghi A được xoay trái 1 bit minh họa như hình vẽ.



#### Ví dụ 43:

MOV A, #1011 0011b ;

RL A ;lệnh thứ nhất

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 A7 bằng 1 chuyển sang.

RL A ;lệnh thứ 2

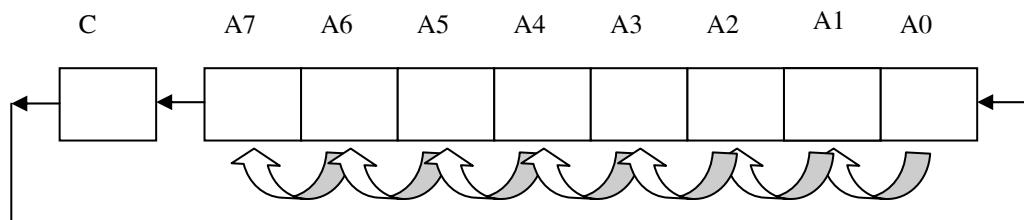
Kết quả sau khi xoay thì (A) = 11001110b và cờ (C) = 0 là do bit A7 bằng 0 chuyển sang.

## 22. Lệnh xoay trái nội dung thanh ghi A và bit carry :

- Cú pháp : **RLC A**
- Mã lệnh :

0	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung thanh ghi A và bit C được xoay trái 1 bit.



**Ví dụ 44:** Giả sử cho cờ C = 0 trước khi thực hiện lệnh

MOV A ,#10110011b

RLC A ;kết quả (A) = 01100110b và cờ (C) = 1

**Ví dụ 45:**

SETB C ;làm cờ C bằng 1

MOV A,#00000000b

RLC A ;kết quả (A) = 0000 0001b và cờ (C) = 0

SETB C ;làm cờ C bằng 1

RLC A ;kết quả (A) = 0000 0011b và cờ (C) = 0

....

SETB C ;làm cờ C bằng 1

RLC A ;kết quả (A) = 0111 1111b và cờ (C) = 0

SETB C ;làm cờ C bằng 1 (lần thứ 8)

RLC A ;kết quả (A) = 1111 1111b và cờ (C) = 0

SETB C ;làm cờ C bằng 1 (lần thứ 9)

RLC A ;kết quả (A) = 1111 1111b và cờ (C) = 1

;xxxx

CLR C ;làm cờ C bằng 0

RLC A ;kết quả (A) = 1111 1110b và cờ (C) = 1

## 23. Lệnh xoay phải nội dung thanh ghi A :

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

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung thanh ghi A được xoay phải 1 bit ngược với lệnh RL A.

## 24. Lệnh xoay phải nội dung thanh ghi A và bit carry :

- Cú pháp : **RRC A**
- Mã lệnh :

1	1	1	0	0	1	0	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung thanh ghi A và bit C được xoay phải 1 bit ngược với lệnh RLC A.

## 25. Lệnh xoay thanh ghi A 4 bit :

- Cú pháp : **SWAP A**
- Mã lệnh :

1	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: hoán chuyển 4 bit thấp và 4 bit cao trong thanh ghi A.

**Ví dụ 4:**

```
MOV A,#3EH
SWAP A ;kết quả (A) = E3H
```

**d. Nhóm lệnh chuyển quyền điều khiển :**

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

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

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

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

## 1. 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.
- **Chức năng:** 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).
- **Chú ý:** Trước khi nạp địa chỉ mới vào thanh ghi PC thì địa chỉ của lệnh kế trong chương trình chính được cất vào bộ nhớ ngắn xếp.

## 2. 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
- **Chức năng:** 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).
- 16 bit địa chỉ A15 – A0 được nạp vào PC, vi điều khiển sẽ thực hiện chương trình con tại địa chỉ vừa nạp vào PC. **Chú ý:** Trước khi nạp địa chỉ vào thanh ghi PC thì địa chỉ của lệnh kế trong chương trình chính được cất vào bộ nhớ ngắn xếp.

## 3. Lệnh trả về từ chương trình con :

- Cú pháp : **RET**
- Mã lệnh :

0	0	1	0	0	0	1	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- **Chức năng:** Lệnh này sẽ kết thúc chương trình con, vi điều khiển sẽ trở lại chương trình chính để tiếp tục thực hiện chương trình.
- **Chú ý:** lệnh này sẽ lấy địa chỉ của lệnh kế đã lưu trong bộ nhớ ngắn xếp (khi thực hiện lệnh gọi) trả lại cho thanh ghi PC để tiếp tục thực hiện chương trình chính. **Khi viết chương trình con thì phải luôn luôn kết thúc bằng lệnh ret.**

## 4. Lệnh trả về từ chương trình con phục vụ ngắn :

- Cú pháp : **RETI**
- Mã lệnh :

0	0	1	1	0	0	1	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- **Chức năng:** Lệnh này sẽ kết thúc chương trình phục vụ ngắn, vi điều khiển sẽ trở lại chương trình chính để tiếp tục thực hiện chương trình.

## 5. Lệnh nhảy dùng địa chỉ tuyệt đối :

- Cú pháp : **AJMP addr11**

- Mã lệnh :

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

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

## 6. Lệnh nhảy dùng địa chỉ 16 bit :

- Cú pháp : **LJMP addr16**
- Mã lệnh :

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

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: vi điều khiển sẽ nhảy đến địa chỉ addr16 để thực hiện chương trình tại đó. Nơi nhảy đến tùy ý nằm trong vùng 64 kbyte.

## 7. Lệnh nhảy tương đối :

- Cú pháp : **SJMP rel**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: vi điều khiển sẽ nhảy đến lệnh có địa chỉ tương đối (rel) để thực hiện tiếp. Có thể thay thế rel bằng nhãn.
- Lệnh này chỉ nhảy trong tầm vực 256 byte: có thể nhảy tới 128 byte và có thể nhảy lùi 128 byte. Khi tầm vực nhảy xa hơn ta nên dùng lệnh AJMP hay LJMP.
- Chú ý: rel [relative: tương đối]: các lệnh có xuất hiện “rel” đều liên quan đến lệnh nhảy: nơi nhảy đến được tính bằng cách lấy nội dung của PC cộng với số lượng byte của các lệnh nằm giữa lệnh nhảy và nơi nhảy đến. Chúng ta không cần quan tâm đến điều này vì chương trình biên dịch của máy tính sẽ tính giúp chúng ta.

## 8. Lệnh nhảy gián tiếp :

- Cú pháp : **JMP @A + D PTR**
- Mã lệnh :

0	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 2 chu kỳ máy

- Chức năng: 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.

#### 9. Lệnh nhảy nếu cờ Z = 1 (nội dung thanh ghi A bằng 0) :

- Cú pháp :
- Mã lệnh :

**JZ rel**

**(jump zero)**

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: nếu bit Z = 1 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel (thỏa điều kiện), nếu Z = 0 thì vi điều khiển sẽ tiếp tục thực hiện lệnh kế (không thỏa điều kiện).

#### 10. Lệnh nhảy nếu cờ Z = 0 (nội dung thanh ghi A khác 0):

- Cú pháp :
- Mã lệnh :

**JNZ rel**

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: nếu Z = 0 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

#### 11. Lệnh nhảy nếu bit carry = 1 :

- Cú pháp :
- Mã lệnh :

**JC rel**

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: nếu bit carry C = 1 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

#### 12. Lệnh nhảy nếu bit carry = 0 :

- Cú pháp :
- Mã lệnh :

**JNC rel**

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: nếu bit carry C = 0 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

#### 13. Lệnh nhảy nếu bit = 1 :

- Cú pháp :
- Mã lệnh :

**JB bit, rel**

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

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: nếu nội dung của bit có địa chỉ bit [được xác định bởi byte thứ 2] bằng 1 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

#### 14. Lệnh nhảy nếu bit = 0 :

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

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

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: nếu nội dung của bit có địa chỉ bit [được xác định bởi byte thứ 2] bằng 0 thì vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ bằng rel.

#### 15. Lệnh nhảy nếu bit = 1 và xóa bit :

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

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

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: nếu bit được xác định bởi byte thứ 2 bằng 1 thì bit này được xóa về 0 và vi điều khiển sẽ nhảy đến thực hiện chương trình tại địa chỉ rel.

#### 16. Lệnh so sánh ô nhớ trực tiếp với nội dung thanh ghi A :

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

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

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: lệnh này ảnh hưởng đến cờ C và thực hiện việc nhảy như sau:
  - ✓ Nếu nội dung của A ≥ nội dung của ô nhớ có địa chỉ direct thì bit C = 0.
  - ✓ Nếu nội dung của A < nội dung của ô nhớ có địa chỉ direct thì bit C = 1.
  - ✓ Nếu nội dung của A khác nội dung của ô nhớ có địa chỉ direct thì lệnh sẽ nhảy đến và thực hiện lệnh tại địa chỉ rel.
  - ✓ Nếu nội dung của A bằng nội dung của ô nhớ có địa chỉ direct thì không nhảy và làm lệnh kế.

#### 17. Lệnh so sánh dữ liệu tức thời với nội dung thanh ghi A :

- Cú pháp : **CJNE A, #data, rel**
- Mã lệnh :

1	0	1	1	0	1	0	0
d7	d6	d5	d4	d3	d2	d1	d0
r7	r6	r5	r4	r3	r2	r1	r0

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: lệnh này ảnh hưởng đến cờ C và thực hiện việc nhảy như sau:
  - ✓ Nếu nội dung của A  $\geq$  data 8 bit thì bit C = 0.
  - ✓ Nếu nội dung của A  $<$  data 8 bit thì bit C = 1.
  - ✓ Nếu nội dung của A khác data 8 bit thì lệnh sẽ nhảy đến thực hiện lệnh tại địa chỉ rel.
  - ✓ Nếu nội dung của A bằng data 8 bit thì không nhảy và làm lệnh kế.

#### 18. Lệnh so sánh dữ liệu tức thời với nội dung thanh ghi :

- Cú pháp : **CJNE Rn, #data, rel**
- Mã lệnh :

1	0	1	1	1	n2	n1	n0
d7	d6	d5	d4	d3	d2	d1	d0
r7	r6	r5	r4	r3	r2	r1	r0

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: lệnh này ảnh hưởng đến cờ C và thực hiện việc nhảy như sau:
  - ✓ Nếu nội dung của thanh ghi Rn  $\geq$  data 8 bit thì bit C = 0.
  - ✓ Nếu nội dung của thanh ghi Rn  $<$  data 8 bit thì bit C = 1.
  - ✓ Nếu nội dung của thanh ghi Rn khác data 8 bit thì lệnh sẽ nhảy đến thực hiện lệnh tại địa chỉ rel.
  - ✓ Nếu nội dung của thanh ghi Rn bằng data 8 bit thì không nhảy và làm lệnh kế.

#### 19. Lệnh so sánh dữ liệu tức thời với dữ liệu gián tiếp :

- Cú pháp : **CJNE @Ri, #data, rel**
- Mã lệnh :

1	0	1	1	0	1	1	i
d7	d6	d5	d4	d3	d2	d1	d0
r7	r6	r5	r4	r3	r2	r1	r0

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: lệnh này ảnh hưởng đến cờ C và thực hiện việc nhảy như sau:
  - ✓ Nếu nội dung của ô nhớ có địa chỉ lưu trong thanh ghi Ri  $\geq$  data 8 bit thì bit C = 0.
  - ✓ Nếu nội dung của ô nhớ có địa chỉ lưu trong thanh ghi Ri  $<$  data 8 bit thì bit C = 1.
  - ✓ 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.
  - ✓ Nếu nội dung của ô nhớ có địa chỉ lưu trong thanh ghi Ri bằng data 8 bit thì không nhảy và làm lệnh kế.

## 20. Lệnh giảm thanh ghi và nhảy :

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

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: 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.

## 21. Lệnh giảm ô nhớ trực tiếp và nhảy :

- Cú pháp : **DJNZ direct, rel**
- Mã lệnh :

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

- Lệnh này chiếm 3 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: 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.

## 22. Lệnh Nop :

- Cú pháp : **NOP**
- Mã lệnh :

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Nội dung của PC tăng lên 1 và tiếp tục thực hiện lệnh tiếp theo.

## e. Nhóm lệnh xử lý bit :

## 1. Lệnh xóa bit carry :

- Cú pháp : **CLR C**
- Mã lệnh :

1	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Xóa bit C về 0.

## 2. Lệnh xóa bit :

- Cú pháp : **CLR bit**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Xóa bit có địa chỉ được xác định bởi byte thứ 2 về 0.

### 3. Lệnh đặt bit carry :

- Cú pháp : **SETB C**
- Mã lệnh :

1	1	0	1	0	0	1	1
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Đặt bit C = 1.

### 4. Lệnh đặt bit :

- Cú pháp : **SETB bit**
- Mã lệnh :

1	1	0	1	0	0	1	0
b7	b6	b5	b4	B3	b2	b1	b0

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Đặt bit có địa chỉ được xác định bởi byte thứ 2 lên 1.

### 5. Lệnh bù bit carry :

- Cú pháp : **CPL C**
- Mã lệnh :

1	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

- Lệnh này chiếm 1 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Bù bit carry, nếu trước đó C = 1 thì C = 0, ngược lại C = 0 thì C = 1.

### 6. Lệnh bù bit :

- Cú pháp : **CPL bit**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Bù bit có địa chỉ xác định bởi byte thứ 2, 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.

### 7. Lệnh and bit carry với bit :

- Cú pháp : **ANL C, bit**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Bit C and với bit có địa chỉ được xác định bởi byte thứ 2, kết quả chứa ở bit C.

## 8. Lệnh and bit carry với bù bit :

- Cú pháp : **ANL C, /bit**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: Bit C and với bù bit có địa chỉ được xác định bởi byte thứ 2, kết quả chứa ở bit C.

## 9. Lệnh or bit carry với bit :

- Cú pháp : **ORL C, bit**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: Bit C or với bit có địa chỉ được xác định bởi byte thứ 2, kết quả chứa ở bit C.

## 10. Lệnh or bit carry với bù bit :

- Cú pháp : **ORL C, /bit**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: Bit C or với bù bit có địa chỉ được xác định bởi byte thứ 2, kết quả chứa ở bit C.

## 11. Lệnh di chuyển bit vào bit carry :

- Cú pháp : **MOV C, bit**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 1 chu kỳ máy
- Chức năng: Bit có địa chỉ được xác định bởi byte thứ 2 được chuyển vào bit C.

## 12. Lệnh di chuyển bit carry vào bit :

- Cú pháp : **MOV bit, C**
- Mã lệnh :

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

- Lệnh này chiếm 2 byte và thời gian thực hiện lệnh là 2 chu kỳ máy
- Chức năng: Bit C được chuyển vào bit có địa chỉ được xác định bởi byte thứ 2.

**4. TÓM TẮT TẬP LỆNH VI ĐIỀU KHIỂN MCS51****Data Transfer Instructions.**

Mnemonic	Instruction code								Hexa decimal	Explanation
	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>		
MOV A,Rn	1	1	1	0	1	n <sub>2</sub>	n <sub>1</sub>	n <sub>0</sub>	E8 ÷ EF	(A) ← (Rn)
MOV A,direct	1	1	1	0	0	1	0	1	E5 Byte 2	(A) ← (direct)
MOV A,@Ri	1	1	1	0	0	1	1	I	E6 ÷ E7	(A) ← ((Ri))
MOV A, #data	0	1	1	1	0	1	0	0	74 Byte 2	(A) ← #data
MOV Rn, A	1	1	1	1	1	n <sub>2</sub>	n <sub>1</sub>	n <sub>0</sub>	F8 ÷ FF	(Rn) ← (A)
MOV Rn,direct	1	0	1	0	1	n <sub>2</sub>	n <sub>1</sub>	n <sub>0</sub>	A8 ÷ AF Byte 2	(Rn) ← (direct)
MOV Rn, #data	0	1	1	1	1	n <sub>2</sub>	n <sub>1</sub>	n <sub>0</sub>	78 ÷ 7F Byte 2	(Rn) ← #data
MOV direct,A	1	1	1	1	0	1	0	1	F5 Byte 2	(direct) ← (A)
MOV direct,Rn	1	0	0	0	1	n <sub>2</sub>	n <sub>1</sub>	n <sub>0</sub>	88 ÷ 8F Byte 2	(direct) ← (Rn)
MOV direct,direct	1	0	0	0	0	1	0	1	85 Byte 2 Byte 3	(direct) ← (direct) (source) (destination)
MOV direct,@Ri	1	0	0	0	0	1	1	I	86 ÷ 87 Byte 2	(direct) ← ((Ri))
MOV direct,#data	0	1	1	1	0	1	0	1	75 Byte 2 Byte 3	(direct) ← #data
MOV @Ri, A	1	1	1	1	0	1	1	I	F6 ÷ F7	((Ri)) ← (A)
MOV @Ri, direct	1	0	1	0	0	1	1	I	A6 ÷ A7 Byte 2	((Ri)) ← (direct)
MOV @Ri, #data	0	1	1	1	0	1	1	i	76 ÷ 77 Byte 2	((Ri)) ← (data)
MOV dptr,#data16	1	0	0	1	0	0	0	0	90 Byte 2 Byte 3	(dptr) ← #data <sub>15-0</sub> (dpH) ← #data <sub>15-8</sub> (dpL) ← #data <sub>7-0</sub>
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 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	C0 Byte 2	(SP) ← (SP) + 1 ((SP)) ← (direct)
POP direct	1 1 0 1 0 0 0 0 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	D0 Byte 2	(direct) ← ((SP)) (SP) ← (SP) - 1
XCH A, Rn	1 1 0 0 1 n <sub>2</sub> n <sub>1</sub> n <sub>0</sub>	C8 ÷ CF	(A) ↔ (Rn)
XCH A, direct	1 1 0 0 0 1 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	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 <sub>3-0</sub> ) ↔ ((Ri <sub>3-0</sub> ))

**Mathematical (Arithmetic) Instructions.**

Mnemonic	Instruction code	Hexa Decimal	Explanation
	D <sub>7</sub> D <sub>6</sub> D <sub>5</sub> D <sub>4</sub> D <sub>3</sub> D <sub>2</sub> D <sub>1</sub> D <sub>0</sub>		
ADD A,Rn	0 0 1 0 1 n <sub>2</sub> n <sub>1</sub> n <sub>0</sub>	28 ÷ 2F	(A) ← (A) + (Rn)
ADD A,direct	0 0 1 0 0 1 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	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 <sub>7</sub> d <sub>6</sub> d <sub>5</sub> d <sub>4</sub> d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>	24 Byte 2	(A) ← (A) + #data
ADDC A, Rn	0 0 1 1 1 n <sub>2</sub> n <sub>1</sub> n <sub>0</sub>	38 ÷ 3F	(A) ← (A) + (Rn) + (C)
ADDC A, direct	0 0 1 1 0 1 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	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 <sub>7</sub> d <sub>6</sub> d <sub>5</sub> d <sub>4</sub> d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>	34 Byte 2	(A) ← (A) + #data +(C)
SUBB A,Rn	1 0 0 1 1 n <sub>2</sub> n <sub>1</sub> n <sub>0</sub>	98 ÷ 9F	(A) ← (A) - (Rn) - (C)
SUBB A,direct	1 0 0 1 0 1 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	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 <sub>7</sub> d <sub>6</sub> d <sub>5</sub> d <sub>4</sub> d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>	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 <sub>2</sub> n <sub>1</sub> n <sub>0</sub>	08 ÷ 0F	(Rn) ← (Rn) + 1
INC direct	0 0 0 0 0 1 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	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 <sub>2</sub> n <sub>1</sub> n <sub>0</sub>	18 ÷ 1F	(Rn) ← (Rn) - 1
DEC direct	0 0 0 1 0 1 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	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 <sub>15-8</sub> ),(A <sub>7-0</sub> ) ← (A) × (B)
DIV AB	1 0 0 0 0 1 0 0	84	(A <sub>15-8</sub> ),(B <sub>7-0</sub> ) ← (A) / (B)
DA A	1 1 0 1 0 1 0 0	D4	Content of A là BCD

**Logic Instructions.**

Mnemonic	Instruction code								Hexa Decimal	Explanation
	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>		
ANL A,Rn	0	1	0	1	1	n <sub>2</sub>	n <sub>1</sub>	n <sub>0</sub>	58 ÷ 5F	(A) ← (A) AND (Rn)
ANL A,direct	0	1	0	1	0	1	0	1	55	(A) ← (A) AND (direct)
	a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	Byte 2	
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	54	(A) ← (A) AND #data
	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	Byte 2	
ANL direct, A	0	1	0	1	0	0	1	0	52	(direct)←(direct) and (A)
	a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	Byte 2	
ANL direct, #data	0	1	0	1	0	0	1	1	53	(direct)←(direct) and #data
	a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	Byte 2	
	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	Byte 3	
ORL A, Rn	0	1	0	0	1	n <sub>2</sub>	n <sub>1</sub>	n <sub>0</sub>	48 ÷ 4F	(A) ← (A) OR (Rn)
ORL A, direct	0	1	0	0	0	1	0	1	45	(A) ← (A) OR (direct)
	a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	Byte 2	
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	44	(A) ← (A) OR #data
	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	Byte 2	
ORL direct, A	0	1	0	0	0	0	1	0	42	(direct)←(direct) OR (A)
	a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	Byte 2	
ORL direct, #data	0	1	0	0	0	0	1	1	43	(direct)←(direct) OR #data
	a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	Byte 2	
	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	Byte 3	
XRL A, Rn	0	1	1	0	1	n <sub>2</sub>	n <sub>1</sub>	n <sub>0</sub>	68 ÷ 6F	(A) ← (A) XOR (Rn)
XRL A, direct	0	1	1	0	0	1	0	1	65	(A) ← (A) XOR (direct)
	a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	Byte 2	
XRL A, @Ri	0	1	1	0	0	1	1	I	66 ÷ 67	(A) ← (A) XOR ((Ri))
XRL A, #data	0	1	1	0	0	1	0	0	64	(A) ← (A) XOR #data
	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	Byte 2	
XRL direct, A	0	1	1	0	0	0	1	0	62	(direct)←(direct) XOR (A)
	a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	Byte 2	
XRL direct, #data	0	1	1	0	0	0	1	1	63	(direct)←(direct) XOR #data
	a <sub>7</sub>	a <sub>6</sub>	a <sub>5</sub>	a <sub>4</sub>	a <sub>3</sub>	a <sub>2</sub>	a <sub>1</sub>	a <sub>0</sub>	Byte 2	
	d <sub>7</sub>	d <sub>6</sub>	d <sub>5</sub>	d <sub>4</sub>	d <sub>3</sub>	d <sub>2</sub>	d <sub>1</sub>	d <sub>0</sub>	Byte 3	
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) ← (Ā) lệnh not hay bù 1
RL A	0	0	1	0	0	0	1	1	23	The contents of the accumulator are rotated left by one bit.
RLC A	0	0	1	1	0	0	1	1	33	The contents of the accumulator and carry are rotated left by one bit.
RR A	0	0	0	0	0	0	1	1	03	The contents of the accumulator are rotated

			right by one bit.
RRC A	0 0 0 1 0 0 1 1	13	The contents of the accumulator and carry are rotated right by one bit.
SWAP A	1 1 0 0 0 1 0 0	C4	(A <sub>3-0</sub> ) ↔ (A <sub>7-4</sub> )

## Control Transfer Instructions.

Mnemonic	Instruction code	Hexa decimal	Explanation
	D <sub>7</sub> D <sub>6</sub> D <sub>5</sub> D <sub>4</sub> D <sub>3</sub> D <sub>2</sub> D <sub>1</sub> D <sub>0</sub>		
ACALL addr 11	A <sub>10</sub> a <sub>9</sub> a <sub>8</sub> 1 0 0 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	Byte 1 Byte 2	(PC) ← (PC) + 2 (SP) ← (SP) + 1 ((SP)) ← (PC <sub>7-0</sub> ) (SP) ← (SP) + 1 ((SP)) ← (PC <sub>15-8</sub> ) (PC) ← page address
LCALL addr 16	0 0 0 1 0 0 0 1 0 a <sub>15</sub> a <sub>14</sub> a <sub>13</sub> a <sub>12</sub> a <sub>11</sub> a <sub>10</sub> a <sub>9</sub> a <sub>8</sub> a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	12 Byte 2 Byte 3	(PC) ← (PC) + 3 (SP) ← (SP) + 1 ((SP)) ← (PC <sub>7-0</sub> ) (SP) ← (SP) + 1 ((SP)) ← (PC <sub>15-8</sub> ) (PC) ← addr <sub>15-0</sub>
RET	0 0 1 0 0 0 0 1 0	22	(PC <sub>15-8</sub> ) ← ((SP)) (SP) ← (SP) - 1 (PC <sub>7-0</sub> ) ← ((SP)) (SP) ← (SP) - 1
RETI	0 0 1 1 0 0 0 1 0	32	(PC <sub>15-8</sub> ) ← ((SP)) (SP) ← (SP) - 1 (PC <sub>7-0</sub> ) ← ((SP)) (SP) ← (SP) - 1
AJMP addr 11	a <sub>10</sub> a <sub>9</sub> a <sub>8</sub> 0 0 0 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	Byte 1 Byte 2	(PC) ← (PC) + 2 (PC) ← page address
LJMP addr 16	0 0 0 0 0 0 0 1 0 a <sub>15</sub> a <sub>14</sub> a <sub>13</sub> a <sub>12</sub> a <sub>11</sub> a <sub>10</sub> a <sub>9</sub> a <sub>8</sub> a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	02 Byte 2 Byte 3	(PC) ← addr <sub>15-0</sub>
SJMP rel	1 0 0 0 0 0 0 0 0 r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	80 Byte 2	(PC) ← (PC) + 2 (PC) ← (PC) + rel
JMP @A + dptr	0 1 1 1 0 0 0 1 1	73	(PC) ← (A) + (dptr)
JZ rel	0 1 1 0 0 0 0 0 0 r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	60 Byte 2	(PC) ← (PC) + 2 IF (A) = 0 then (PC) ← (PC) + rel
JNZ rel	0 1 1 1 0 0 0 0 0 r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	70 Byte 2	(PC) ← (PC) + 2 IF (A) ≠ 0 then (PC) ← (PC) + rel
JC rel	0 1 0 0 0 0 0 0 0 r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	40 Byte 2	(PC) ← (PC) + 2 IF (C) = 1 then (PC) ← (PC) + rel

JNC rel	0 1 0 1 0 0 0 0 r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	50 Byte 2	(PC) $\leftarrow$ (PC) + 2 IF (C) = 0 then (PC) $\leftarrow$ (PC) + rel
JB bit, rel	0 0 1 0 0 0 0 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub> r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	20 Byte 2 Byte 3	(PC) $\leftarrow$ (PC) + 3 IF (bit) = 1 then (PC) $\leftarrow$ (PC) + rel
JNB bit, rel	0 0 1 1 0 0 0 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub> r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	30 Byte 2 Byte 3	(PC) $\leftarrow$ (PC) + 3 IF (bit) = 0 then (PC) $\leftarrow$ (PC) + rel
JBC bit, rel	0 0 0 1 0 0 0 0 b <sub>7</sub> b <sub>6</sub> b <sub>5</sub> b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub> b <sub>0</sub> r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	10 Byte 2 Byte 3	(PC) $\leftarrow$ (PC) + 3 IF (bit) = 1 then (bit) $\leftarrow$ 0 (PC) $\leftarrow$ (PC) + rel
CJNE A, direct, rel	1 0 1 1 0 1 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub> r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	B5 Byte 2 Byte 3	(PC) $\leftarrow$ (PC) + 3 IF (direct) < (A) then (C) $\leftarrow$ 0 and (PC) $\leftarrow$ (PC) + rel IF (direct) > (A) then (C) $\leftarrow$ 1 and (PC) $\leftarrow$ (PC) + rel
CJNE A, #data, rel	1 0 1 1 0 1 0 0 d <sub>7</sub> d <sub>6</sub> d <sub>5</sub> d <sub>4</sub> d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub> r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	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	1 0 1 1 0 n <sub>2</sub> n <sub>1</sub> n <sub>0</sub> d <sub>7</sub> d <sub>6</sub> d <sub>5</sub> d <sub>4</sub> d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub> r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	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	1 0 1 1 0 1 1 0 d <sub>7</sub> d <sub>6</sub> d <sub>5</sub> d <sub>4</sub> d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub> r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	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	1 1 0 1 1 n <sub>2</sub> n <sub>1</sub> n <sub>0</sub> r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	D8 ÷ DF Byte 2	(PC) $\leftarrow$ (PC) + 2 (Rn) $\leftarrow$ (Rn) - 1 IF ((Rn)) $\neq$ 0 then (PC) $\leftarrow$ (PC) + rel
DJNZ direct, rel	1 1 0 1 0 1 0 1 a <sub>7</sub> a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub> r <sub>7</sub> r <sub>6</sub> r <sub>5</sub> r <sub>4</sub> r <sub>3</sub> r <sub>2</sub> r <sub>1</sub> r <sub>0</sub>	D5 Byte 2 Byte 3	(PC) $\leftarrow$ (PC) + 3 (direct) $\leftarrow$ (direct) - 1 IF (direct) $\neq$ 0 then (PC) $\leftarrow$ (PC) + rel

NOP	0 0 0 0 0 0 0 0	00	(PC) $\leftarrow$ (PC) + 1
-----	-----------------	----	----------------------------

### Bit Oriented Instructions.

Mnemonic	Instruction code								Hexa decimal	Explanation
	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>		
CLR C	1	1	0	0	0	0	1	1	C3	(C) $\leftarrow$ 0
CLR bit	1	1	0	0	0	0	1	0	C2	(bit) $\leftarrow$ 0
	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	Byte 2	
SETB C	1	1	0	1	0	0	1	1	D3	(C) $\leftarrow$ 1
SETB bit	1	1	0	1	0	0	1	0	D2	(bit) $\leftarrow$ 1
	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	Byte 2	
CPL C	1	0	1	1	0	0	1	1	B3	(C) $\leftarrow$ ( $\bar{C}$ )
CPL bit	1	0	1	1	0	0	1	0	B2	
	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	Byte 2	(bit) $\leftarrow$ ( $\bar{bit}$ )
ANL C,bit	1	0	0	0	0	0	1	0	82	(C) $\leftarrow$ (C) AND (bit)
	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	Byte 2	
ANL C,/bit	1	0	1	1	0	0	0	0	B0	
	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	Byte 2	(C) $\leftarrow$ (C) AND ( $\bar{bit}$ )
ORL C,bit	0	1	1	1	0	0	1	0	72	(C) $\leftarrow$ (C) OR (bit)
	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	Byte 2	
ORL C,/bit	1	0	1	0	0	0	0	0	A0	
	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	Byte 2	(C) $\leftarrow$ (C) OR ( $\bar{bit}$ )
MOV C,bit	1	0	1	0	0	0	1	0	A2	(C) $\leftarrow$ (bit)
	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	Byte 2	
MOV bit,C	1	0	0	1	0	0	1	0	92	(bit) $\leftarrow$ (C)
	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>	b <sub>0</sub>	Byte 2	

Bảng 4-5. Tóm tắt tập lệnh vi điều khiển MCS51.

## V. BỘ ĐỊNH THỜI TIMER TRONG VI ĐIỀU KHIỂN MCS51

### 1 GIỚI THIỆU

Trong vi điều khiển MCS51 có 2 timer/counter T0 và T1, còn MCS52 thì có 3 timer/counter. 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 tạo ra lượng thời gian cần thiết – ở chế độ timer vi điều khiển thường đếm xung nhận từ bên ngoài đưa đến ngõ vào T0 đối với timer/counter thứ 0 và ngõ vào T1 đối với timer/counter thứ 1. Đếm xung từ bên ngoài còn gọi là đếm sự kiện. Một ứng dụng cho chế độ counter là có thể sử dụng vi điều khiển làm các mạch đếm sản phẩm.

Nếu chúng ta 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. Chế độ counter thường thì đếm xung nhận từ bên ngoài đưa đến ngõ vào T0 đối với timer/counter thứ 0 và ngõ vào T1 đối với timer/counter thứ 1. Đếm xung từ bên ngoài còn gọi là đếm sự kiện. 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.

Các timer/counter của vi điều khiển sử dụng 16 flip flop nên ta gọi là timer/counter 16 bit và số lượng xung mà timer/ counter có thể đếm được tính theo số nhị phân bắt đầu từ 0000 0000 0000 0000<sub>2</sub> đến 1111 1111 1111 1111<sub>2</sub>, nếu viết theo số thập lục phân thì bắt đầu từ 0000H đến FFFFH và nếu tính theo giá trị thập phân thì bắt đầu từ 0 đến 65535.

Khi đạt đến giá trị cực đại và nếu có thêm 1 xung nữa thì bộ đếm sẽ bị tràn, khi bị tràn thì giá trị đếm sẽ tự động về 0 (giống như mạch đếm nhị phân 4 bit khi đếm lên 1111 và nếu có 1 xung nữa thì giá trị đếm về 0000) và cờ tràn của timer/counter lên 1 để báo hiệu timer/counter đã bị tràn (trước khi đếm thì phải xoá cờ tràn).

Người lập trình sử dụng trạng thái cờ tràn lên 1 để rẽ nhánh hoạt chấm dứt thời gian cần thiết đã định để chuyển sang làm một công việc khác, khi cờ tràn lên 1 sẽ tạo ra ngắt cũng để rẽ nhánh chương trình để thực hiện một chương trình khác – bạn sẽ nắm rõ ở phần ứng dụng.

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.

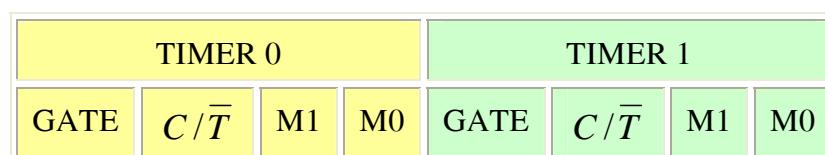
Ngoài các thanh ghi lưu trữ số xung đếm vừa giới thiệu thì còn có 2 thanh ghi hỗ trợ kèm theo: thanh ghi TMOD và thanh ghi TCON dùng để thiết lập nhiều chế độ hoạt động khác nhau cho timer để đáp ứng được sự đa dạng các yêu cầu ứng dụng trong thực tế.

Bảng 4-6 sẽ liệt kê tên, chức năng, địa chỉ của các thanh ghi liên quan đến các timer/counter của vi điều khiển 89C51.

Tên	Chức năng	Địa chỉ	Cho phép truy xuất bit
TCON	Control	88H	YES
TMOD	Mode	89H	NO
TL0	Timer 0 low-byte	8AH	NO
TL1	Timer 1 low-byte	8BH	NO
TH0	Timer 0 high-byte	8CH	NO
TH1	Timer 1 high-byte	8DH	NO

**Bảng 4-6. Các thanh ghi timer vi điều khiển MCS51.**

## 2. THANH GHI CHỌN KIỂU LÀM VIỆC CHO TIMER – MODE REGISTER:



Thanh ghi TMOD gồm hai nhóm 4 bit: 4 bit thấp dùng để thiết lập các chế độ hoạt động cho T0 và 4 bit cao thiết lập các chế độ hoạt động cho T1.

Các bit của thanh ghi TMOD được tóm tắt như bảng 4-7 :

Bit	Tên	Timer	Chức năng
7	GATE	1	Nếu GATE = 1 thì Timer 1 chỉ làm việc khi INT1= 1.
6	C/T	1	Bit lựa chọn counter hay timer: C/T = 1 : đếm xung từ bên ngoài đưa đến ngõ vào T1. C/T = 0 : định thời đếm xung nội bộ bên trong.
5	M1	1	Bit chọn mode của Timer 1.
4	M0	1	Bit chọn mode của Timer 1.
3	GATE	0	Nếu GATE = 1 thì Timer 0 chỉ làm việc khi INT0= 1.
2	C/T	0	Bit lựa chọn counter hay timer: giống như trên.
1	M1	0	Bit chọn mode của Timer 0.
0	M0	0	Bit chọn mode của Timer 0.

Bảng 4-7. Các bit trong thanh ghi TMOD.

Hai bit M0 và M1 tạo ra 4 trạng thái tương ứng với 4 kiểu làm việc khác nhau của T0 hoặc của T1 như bảng 4-8.

M1	M0	Kiểu	Chức năng
0	0	0	Mode Timer 13 bit (mode 8048)
0	1	1	Mode Timer 16 bit
1	0	2	Mode tự động nạp 8 bit
1	1	3	Mode tách Timer ra : <i>Timer0</i> : được tách ra làm 2 timer 8 bit gồm có: Timer 8 bit TL0 được điều khiển bởi các bit của T0. Timer 8 bit TH0 được điều khiển bởi các bit của T1. <b>Timer1</b> : không được hoạt động ở mode 3.

Bảng 4-8. Các bit chọn mode trong thanh ghi TMOD.

### 3. THANH GHI ĐIỀU KHIỂN TIMER – CONTROL REGISTER:

Thanh ghi điều khiển tcon chứa các bit trạng thái và các bit điều khiển cho T0 và T1.



Hoạt động của từng bit của thanh ghi tcon được tóm tắt như bảng 4-9:

Bit	Kí hiệu	Địa chỉ	Chức năng
7	TF1	8FH	Cờ tràn Timer 1: TF1 = 1 khi timer 1 bị tràn và xóa bằng phần mềm hoặc khi vi điều khiển thực hiện chương trình con phục vụ ngắt timer1 thì tự động xóa luôn cờ tràn TF1.
6	TR1	8EH	Bit điều khiển Timer 1 đếm / ngừng đếm: TR1 = 1 thì timer 1 được phép đếm xung. TR1 = 0 thì timer 1 không được phép đếm xung (ngừng).
5	TF0	8DH	Cờ tràn Timer 0 (hoạt động tương tự TF1)

<b>4</b>	<b>TRO</b>	<b>8CH</b>	<b>Bit điều khiển Timer 0 (giống TR1)</b>
<b>3</b>	<b>IE1</b>	<b>8BH</b>	Cờ báo ngắt INT1. Khi có ngắt xảy ra ở ngõ vào INT1 (cạnh xuống) thì cờ IE1 tác động lên mức 1. Khi vi điều khiển thực hiện chương trình con phục vụ ngắt INT1 thì tự động xóa luôn cờ báo ngắt IE1.
<b>2</b>	<b>IT1</b>	<b>8AH</b>	Bit điều khiển cho phép ngắt INT1 tác động bằng mức hay bằng cạnh. IT1 = 0 thì ngắt INT1 tác động bằng mức. IT1 = 1 thì ngắt INT1 tác động bằng cạnh xuống.
<b>1</b>	<b>IE0</b>	<b>89H</b>	Giống như IE1 nhưng phục vụ cho ngắt INTO
<b>0</b>	<b>IT0</b>	<b>88H</b>	Giống như IT1 nhưng phục vụ cho ngắt INTO

Bảng 4-9. Các bit trong thanh ghi TCON.

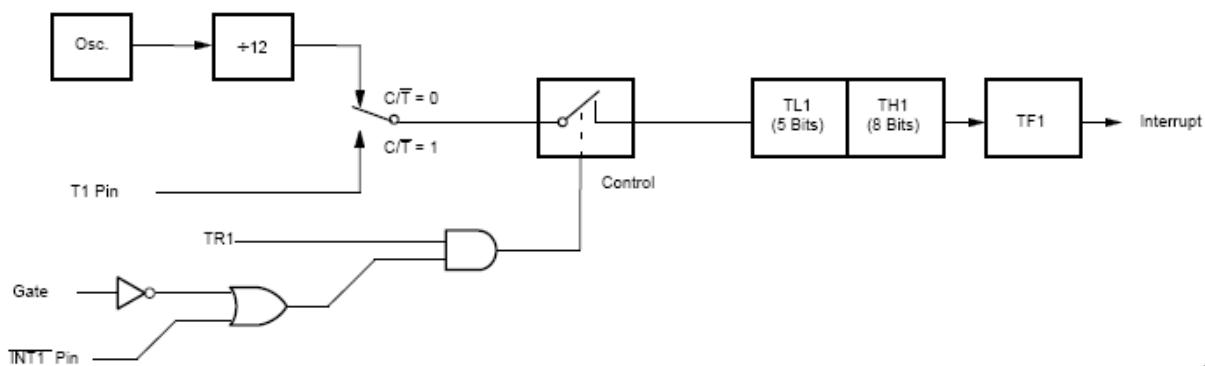
#### 4. CÁC KIỂU HOẠT ĐỘNG CỦA TIMER VÀ CỜ TRÀN

MCS51 có 2 timer là T0 và T1. Ta dùng ký hiệu TLx và THx để chỉ 2 thanh ghi byte thấp và byte cao của Timer0 hoặc Timer1. Như đã trình bày ở trên các timer có 4 kiểu hoạt động, phần này ta sẽ khảo sát chi tiết các kiểu hoạt động của timer.

##### 1. MODE 0 (Mode Timer 13 bit) :

Mode 0 là mode đếm 13 bit: trong đó 8 bit cao sử dụng hết 8 bit của thanh ghi THx, 5 bit còn lại chỉ sử dụng 5 bit trọng số thấp của thanh ghi TLx, còn 3 bit cao của TLx không dùng như hình 4-11.

Ở mode này giá trị đếm lớn nhất là  $2^{13} = 8192$  tức đếm từ 0 0000 0000 0000<sub>2</sub> đến 1 1111 1111 1111<sub>2</sub> và nếu có thêm một xung nữa thì bộ đếm sẽ tràn và làm cho cờ tràn lên 1. Nếu trong chương trình có cho phép timer ngắt thì ngắt sẽ tác động.



Hình 4-11. Timer 1 hoạt động ở mode 0.

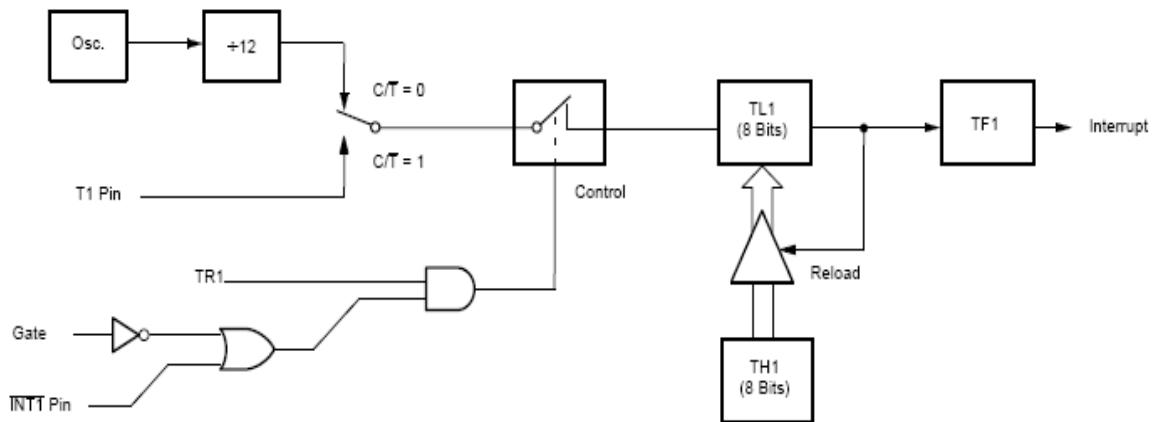
##### 2. MODE 1 (Mode Timer 16 bit) :

Mode 1 là mode đếm 16 bit. Ở mode này giá trị đếm là lớn nhất là  $2^{16}$ . Timer hoạt động ở mode 1 giống như mode 0 chỉ khác là mode 1 đếm 16bit.

##### 3. MODE 2 (Mode Timer tự động nạp 8 bit) :

Mode 2 là mode tự động nạp 8 bit, byte thấp TLx của Timer hoạt động như một Timer 8 bit trong khi byte cao THx của Timer dùng để lưu trữ giá trị để nạp lại cho thanh ghi TLx.

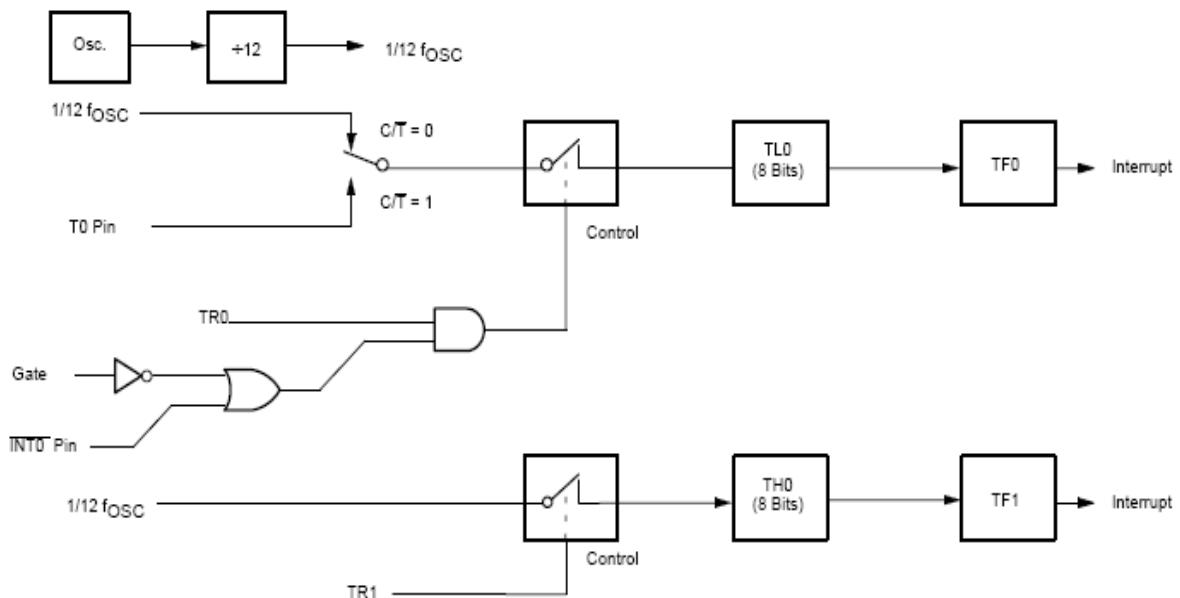
Khi bộ đếm TLx chuyển trạng thái từ FFH sang 00H: thì cờ tràn được set và giá trị lưu trong THx được nạp vào TLx. Bộ đếm TLx tiếp tục đếm từ giá trị vừa nạp từ THx lên và cho đến khi có chuyển trạng thái từ FFH sang 00H kế tiếp và cứ thế tiếp tục. Sơ đồ minh họa cho timer hoạt động ở mode 2 như hình 4-12.



Hình 4-12. Timer 1 hoạt động ở mode 2.

#### 4. MODE 3 (Mode Timer tách ra):

Mode 3 là mode Timer 0 tách ra làm 2 timer cùng với timer 1 tạo thành 3 timer.



Hình 4-13. Timer 0 hoạt động ở mode 3.

Khi Timer0 định ở cấu hình mode 3 thì timer0 được chia là 2 timer 8 bit TL0 và TH0 hoạt động như những Timer riêng lẻ và sử dụng các bit TF0 và TF1 làm các bit cờ tràn tương ứng như hình 4-13.

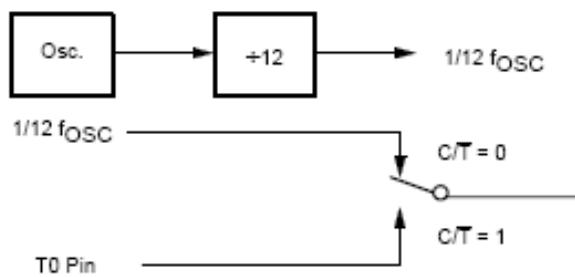
Timer 1 không thể sử dụng ở mode 3, nhưng có thể được khởi động trong các mode khác và không thể báo tràn vì cờ tràn TF1 đã dùng để báo tràn cho timer TH0.

Khi timer 0 hoạt động ở Mode 3 sẽ cung cấp thêm 1 Timer 8 bit thứ ba. Khi Timer 0 ở mode 3 thì Timer 1 có thể hoạt động như là một bộ dao động thiết lập tốc độ Baud phục vụ cho Port nối

tiếp để truyền và nhận dữ liệu, hoặc nó có thể dùng trong các ứng dụng mà không sử dụng chế độ bão tràn và bão ngắn. Sơ đồ minh họa cho timer hoạt động ở mode 3 như hình 4-13.

### 5. CÁC NGUỒN XUNG ĐẾM:

Timer/counter có thể đếm xung từ hai nguồn: nếu timer/counter sử dụng ở chế độ định thời timer thì sẽ đếm xung bên trong (xung nội) đã biết tần số, nếu timer/counter sử dụng ở chế độ counter thì sẽ đếm xung từ bên ngoài như hình 4-14. Bit  $C/T$  trong TMOD cho phép chọn chế độ timer hay counter khi khởi tạo ở thanh ghi tmod.



Hình 4-14. Các nguồn xung đưa đến timer / counter.

#### 1. Đếm thời gian:

Nếu bit  $C/T = 0$  thì Timer hoạt động đếm xung nội liên tục lấy từ dao động trên Chip. Tần số ngõ vào tụ thạch anh được đưa qua một mạch chia 12 để giảm tần số phù hợp với các ứng dụng. Nếu dùng thạch anh 12MHz thì sau khi qua bộ chia 12, tần số đưa đến bộ đếm timer là 1MHz.

Timer sẽ sinh ra tràn khi nó đã đếm đủ số xung tương ứng thời gian qui định, phụ thuộc vào giá trị khởi tạo được nạp vào các thanh ghi THx và TLx.

#### 2. Đếm các sự kiện bên ngoài (Event Counting):

Nếu bit  $C/T = 1$  thì Timer hoạt động đếm xung đến từ bên ngoài và chu kỳ của mỗi xung do nguồn tạo tín hiệu bên ngoài quyết định. Hoạt động này thường dùng để đếm các sự kiện. Số lượng các sự kiện được lưu trữ trong thanh ghi của các Timer.

Nguồn xung clock bên ngoài đưa vào chân P3.4 là ngõ nhập xung clock của Timer0 (T0) và P3.5 là ngõ nhập xung clock của Timer1 (T1).

Trong các ứng dụng đếm xung từ bên ngoài: các thanh ghi Timer sẽ tăng giá trị đếm khi xung ngõ vào Tx chuyển trạng thái từ 1 sang 0 (tác động xung clock cạnh xuống). Ngõ vào nhận xung bên ngoài được lấy mẫu trong suốt khoảng thời gian S5P2 của mỗi chu kỳ máy, do đó khi xung ở mức 1 trong một chu kỳ này và chuyển sang mức 0 trong một chu kỳ kế thì bộ đếm tăng lên một. Để nhận ra sự chuyển đổi từ 1 sang 0 phải mất 2 chu kỳ máy, nên tần số xung bên ngoài lớn nhất là 500KHz nếu hệ thống vi điều khiển sử dụng dao động thạch anh 12 MHz.

### 6. ĐIỀU KHIỂN CÁC TIMER HOẠT ĐỘNG :

Bit TRx trong thanh ghi TCON được điều khiển bởi phần mềm để cho phép các Timer bắt đầu quá trình đếm hoặc ngừng.

Để bắt đầu cho các Timer đếm thì phải set bit TRx bằng lệnh:

SETB TR0 ; cho phép timer T0 bắt đầu đếm

SETB TR1 ; cho phép timer T1 bắt đầu đếm

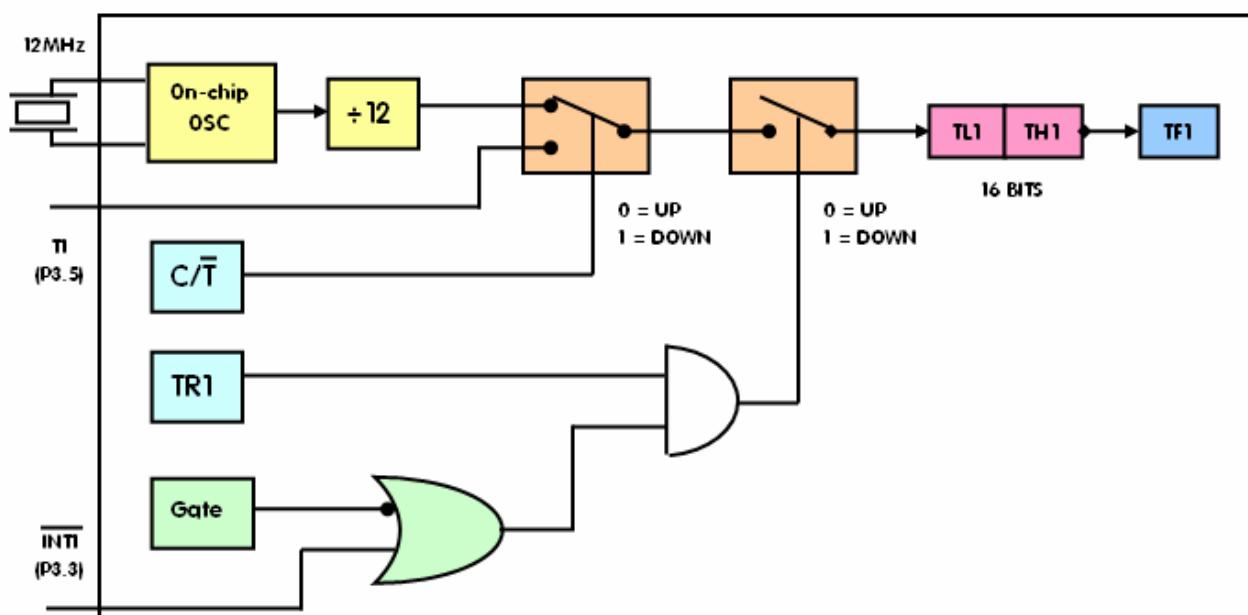
Để các Timer ngừng đếm ta dùng lệnh Clear bit TRx.

Ví dụ Timer 0 bắt đầu bởi lệnh SETB TR0 và ngừng đếm bởi lệnh CLR TR0.

Bit TRx bị xóa khi reset hệ thống, do đó ở chế độ mặc định khi mở máy các Timer bị cấm.

Một phương pháp khác để điều khiển các Timer là dùng bit GATE trong thanh ghi TMOD và ngõ nhập bên ngoài INTx như hình 4-15. Phương pháp này được dùng để đo các độ rộng xung.

Giả sử xung cần đo độ rộng đưa vào chân INT0, ta phải khởi tạo Timer 0 hoạt động ở mode 1 là mode Timer 16 bit với giá trị khởi tạo ban đầu là TL0/TH0 = 0000H, bit GATE = 1, bit TR0 = 1. Khi xung đưa đến ngõ vào INT0 = 1 thì “cổng được mở” để cho xung nội có tần số 1MHz vào timer 0. Quá trình timer 0 đếm xung nội sẽ dừng lại cho đến khi xung đưa đến ngõ vào INT0 xuống mức 0. Thời gian đếm được của timer 0 chính là độ rộng xung cần đo.



Hình 4-15. Đo độ rộng xung từ bên ngoài.

#### 7. KHỞI TẠO VÀ TRUY XUẤT CÁC THANH GHI CỦA TIMER/COUNTER:

Các Timer thường được khởi tạo 1 lần ở đầu chương trình để thiết lập mode hoạt động phục vụ cho các ứng dụng điều khiển liên quan đến định thời hay đếm xung ngoại. Tùy thuộc vào yêu cầu điều khiển cụ thể mà ta điều khiển các timer bắt đầu đếm, ngừng hay khởi động đếm lại từ đầu ...

Thanh ghi TMOD là thanh ghi đầu tiên cần phải khởi tạo để thiết lập mode hoạt động cho các Timer. Ví dụ khởi động cho Timer0 hoạt động ở mode 1 (mode Timer 16 bit) và hoạt động định thời đếm xung nội bên trong thì ta khởi tạo bằng lệnh: MOV TMOD, #00000001B. Trong lệnh này M1 = 0, M0 = 1 để vào mode 1 và C/T = 0, GATE = 0 để cho phép đếm xung nội bên trong đồng thời xóa các bit mode của Timer 1. Sau lệnh trên Timer 0 vẫn chưa đếm và timer 0 chỉ đếm khi set bit điều khiển chạy TR0.

Nếu ta không thiết lập các giá trị bắt đầu đếm cho các thanh ghi TLx/THx thì Timer sẽ bắt đầu đếm từ 0000H lên và khi chuyển trạng thái từ FFFFH sang 0000H sẽ sinh ra tràn làm cho bit TFx = 1 rồi tiếp tục đếm từ 0000H lên tiếp . . .

Nếu ta thiết lập giá trị bắt đầu đếm cho TLx/THx khác 0000H, thì Timer sẽ bắt đầu đếm từ giá trị thiết lập đó lên nhưng khi chuyển trạng thái từ FFFFH sang 0000H thì timer lại đếm từ 0000H lên.

Để timer luôn bắt đầu đếm từ giá trị ta gán thì ta có thể lập trình chờ sau mỗi lần tràn ta sẽ xóa cờ TFx và gán lại giá trị cho TLx/THx để Timer luôn luôn bắt đầu đếm từ giá trị khởi gán lên.

Đặc biệt nếu bộ định thời hoạt động trong phạm vi nhỏ hơn 256 µs thì ta nên dùng Timer ở mode 2 (tự động nạp 8 bit). Sau khi khởi tạo giá trị đầu cho thanh ghi THx, và TLx, khi set bit TRx thì Timer sẽ bắt đầu đếm từ giá trị đã gán trong TLx và khi tràn từ FFH sang 00H trong TLx, thì cờ tràn TFx tự động được set, đồng thời giá trị trong Thx tự động nạp sang cho TLx và Timer bắt đầu đếm từ giá trị khởi gán này lên. Nói cách khác, sau mỗi lần tràn ta không cần khởi gán lại cho các thanh ghi Timer mà chúng vẫn đếm được lại từ giá trị đã gán.

#### Ví dụ 47: Chương trình tạo xung vuông tần số 1kHz sử dụng timer1 mode1:

Tần số 1KHz = 1000Hz thì chu kỳ T = 1ms = 1000µs. Thời gian ở mức 0 bằng thời gian ở mức 1 bằng 500µs. Sử dụng thạch anh 12 MHz nên mỗi chu kỳ là 1µs, để đếm 500 xung thì ta có thể chọn định cấu hình timer ở mode 0 hoặc mode 1 đều được. Ngõ ra xung là bit P1.0.

```

MOV TMOD,#01H      ;chọn mode 1 timer 0 đếm 16 bit
LOOP1: MOV TH0,#HIGH(-500) ;độ rộng xung 500µs
       MOV TL0,#LOW(-500) ;
       SETB TR0          ;cho timer bắt đầu đếm
       JNB TF0,$         ;chờ báo ngắt
       CLR TF0           ;xóa cờ ngắt
       CPL P1.0          ;nghịch đảo bit p1.0
       SJMP LOOP1        ;quay trở lại thực hiện tiếp
    
```

#### Ví dụ 48: Chương trình tạo xung vuông tần số 10 kHz sử dụng timer mode2:

Tần số 10KHz = 10000Hz thì chu kỳ T = 100µs. Thời gian ở mức 0 bằng thời gian ở mức 1 bằng 50µs. Sử dụng thạch anh 12 MHz nên mỗi chu kỳ là 1µs, để đếm 50 xung thì ta có thể chọn định cấu hình timer ở mode 2.

```

MOV TMOD,#02H      ;chọn mode 2 chế độ tự động nạp lại 8 bit
MOV TH0,#-50        ;tạo độ rộng xung 50µs
SETB TR0            ;cho timer bắt đầu đếm
LOOP: JNB TF0,$     ;chờ báo ngắt
      CLR TF0         ;xóa cờ ngắt
      CPL P1.0          ;nghịch đảo bit p1.0
      SJMP LOOP        ;trở lại loop
    
```

#### 8. TIMER/COUNTER T2 CỦA VI ĐIỀU KHIỂN HỘ MCS52:

Hộ vi điều khiển MCS52 có 3 timer T0, T1, T2. Các timer T0 và T1 có các thanh ghi và hoạt động giống như họ 51. Ở đây chỉ trình bày thêm phần hoạt động của timer T2.

Các thanh ghi của timer/counter T2 bao gồm: thanh ghi TL2, TH2, thanh ghi điều khiển T2CON, thanh ghi RCAP2L và RCAP2H.

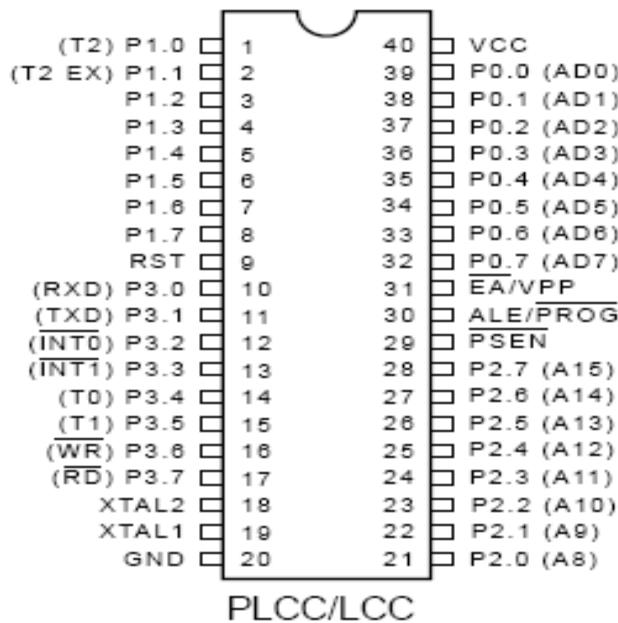
Timer/counter T2 có thể dùng để định thời timer hoặc dùng như bộ đếm counter để đếm xung ngoài đưa đến ngõ vào T2 chính là chân P1.0 của port 1 như hình 4-16.

Timer/counter T2 có 3 kiểu hoạt động: tự động nạp lại, thu nhận và thiết lập tốc độ baud để phục vụ cho truyền dữ liệu.

Chức năng của thanh ghi điều khiển T2CON như bảng 4-10:

Bit	Kí hiệu	Địa chỉ	Chức năng
7	TF2	CFH	Cờ tràn Timer 2: hoạt động giống như các timer trên (TF2 sẽ không được thiết lập lên mức 1 nếu bit TCLK hoặc RCLK ở mức 1).
6	EXF2	CEH	Cờ ngoài của timer T2: chỉ được set khi xảy ra sự thu nhận hoặc nạp lại dữ liệu bởi sự chuyển trạng thái từ 1 sang 0 ở ngõ vào T2EX và EXEN2 = 1; khi cho phép timer T2 ngắn, EXF2=1 thì CPU sẽ thực hiện chương trình con phục vụ ngắn Timer T2, bit EXF2 có thể bằng phần mềm.
5	RCLK	CDH	Xung clock thu của timer 2. Khi RCLK=1 thì timer T2 cung cấp tốc độ baud cho port nối tiếp để nhận dữ liệu về và timer T1 sẽ cung cấp tốc độ baud cho port nối tiếp để phát dữ liệu đi.
4	TCLK	CCH	Xung clock phát của timer 2. Khi TCLK=1 thì timer T2 cung cấp tốc độ baud cho port nối tiếp để phát dữ liệu đi và timer T1 sẽ cung cấp tốc độ baud cho port nối tiếp để nhận dữ liệu về.
3	EXEN2	CBH	Bit điều khiển cho phép tác động từ bên ngoài. Khi EXEN2 = 1 thì hoạt động thu nhận và nạp lại của timer T2 chỉ xảy ra khi ngõ vào T2EX có sự chuyển trạng thái từ 1 sang 0.
2	TR2	CAH	Bit điều khiển Timer 1 đếm / ngừng đếm: TR2 = 1 thì timer 1 được phép đếm xung. TR2 = 0 thì timer 1 không được phép đếm xung (ngừng). Dùng lệnh điều khiển bit TR2 để cho phép timer1 đếm hay ngừng đếm.
1	C/T2	C9H	Bit lựa chọn counter hay timer: C/T2 = 1 : đếm xung từ bên ngoài đưa đến ngõ vào T2. C/T2 = 0 : định thời đếm xung nội bộ trong.
0	CP/RL2	C8H	Cờ thu nhận/nạp lại dữ liệu của timer T2. Khi bit này = 1 thì thu nhận chỉ xảy ra khi có sự chuyển trạng thái từ 1 sang 0 ở ngõ vào T2EX và EXEN2=1; khi bit này = 0 thì quá trình tự động nạp lại khi timer T2 tràn hoặc khi có sự chuyển trạng thái ở ngõ vào T2EX và bit EXEN2 = 1; nếu bit RCLK hoặc TCLK = 1 thì bit này xem như bỏ.

Bảng 4-10. Các bit trong thanh ghi T2CON.



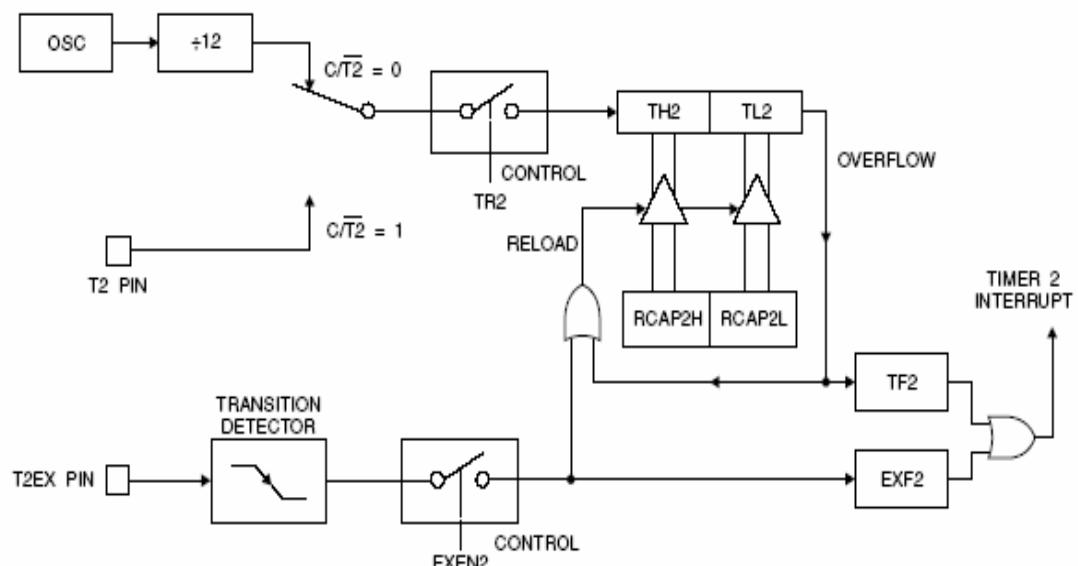
Hình 4-16. Sơ đồ chân của 89C52 với ngõ vào T2 là P1.0 và T2EX là P1.1.

### a. Chế độ tự động nạp lại:

Bit thu nhận/nạp lại  $CP/RL2$  lựa chọn một trong hai chế độ: tự động nạp lại và thu nhận. Khi  $CP/RL2 = 0$  thì timer hoạt động ở chế độ tự động nạp lại: các thanh ghi TL2, TH2 sẽ lưu trữ số xung đếm còn 2 thanh ghi RCAP2L và RCAP2H lưu trữ giá trị để nạp lại cho TL2, TH2. Giá trị lưu và nạp lại là 16 bit.

Khi timer đếm tràn thì làm cho cờ báo tràn TF2 bằng 1 đồng thời tự động thực hiện nạp lại dữ liệu.

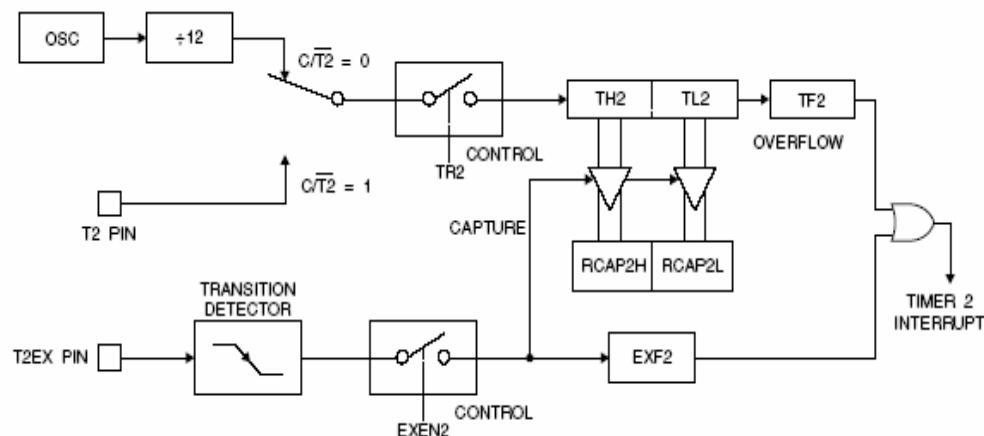
Tương tự nếu bit EXEN2 = 1 thì chế độ tự động nạp lại xảy ra khi có sự chuyển trạng thái từ 1 sang 0 ở ngõ vào T2EX đồng thời làm cho bit EXF2 = 1. Tương tự như cờ TF2 thì cờ EXF2 cũng có thể được kiểm tra bằng phần mềm hoặc tạo ngắn. Bit EXF2 phải xóa bằng phần mềm. Hoạt động tự nạp của timer T2 được trình bày như hình 4-17.



Hình 4-17. Hoạt động của timer T2 ở chế độ tự động nạp lại.

### b. Chế độ thu nhận:

Khi  $CP/\overline{RL2} = 1$  thì timer hoạt động ở chế độ thu nhận. Khi đó timer T2 hoạt động bình thường như một timer/counter 16 bit, thanh ghi TL2, TH2 sẽ lưu trữ xung đếm và nếu có sự chuyển trạng thái từ FFFFH sang 0000H thì sẽ sinh ra tràn và làm cho cờ tràn TF2=1. Bit cờ tràn có thể kiểm tra bằng phần mềm hay có thể tạo ra ngắt.



**Hình 4-18. Hoạt động của timer T2 ở chế độ Thu nhận dữ liệu.**

Để cho phép chế độ thu nhận hoạt động thì làm cho bit EXEN2 = 1. Nếu bit EXEN2 = 1 và khi có sự chuyển trạng thái trái từ 1 sang 0 ở ngõ vào T2EX thì chế độ thu nhận sẽ xảy ra: giá trị đếm được trong thanh ghi TL2, TH2 sẽ được chuyển sang 2 thanh ghi RCAP2L và RCAP2H. Cờ EXF2 cũng được chuyển lên mức 1 để báo hiệu quá trình thu nhận đã xảy ra, cờ EXF2 có thể kiểm tra bằng phần mềm hoặc tạo ngắt.

Hoạt động thu nhận dữ liệu của timer T2 được trình bày ở hình 4-18.

## VI. HOẠT ĐỘNG TRUYỀN DỮ LIỆU CỦA VI ĐIỀU KHIỂN MCS51

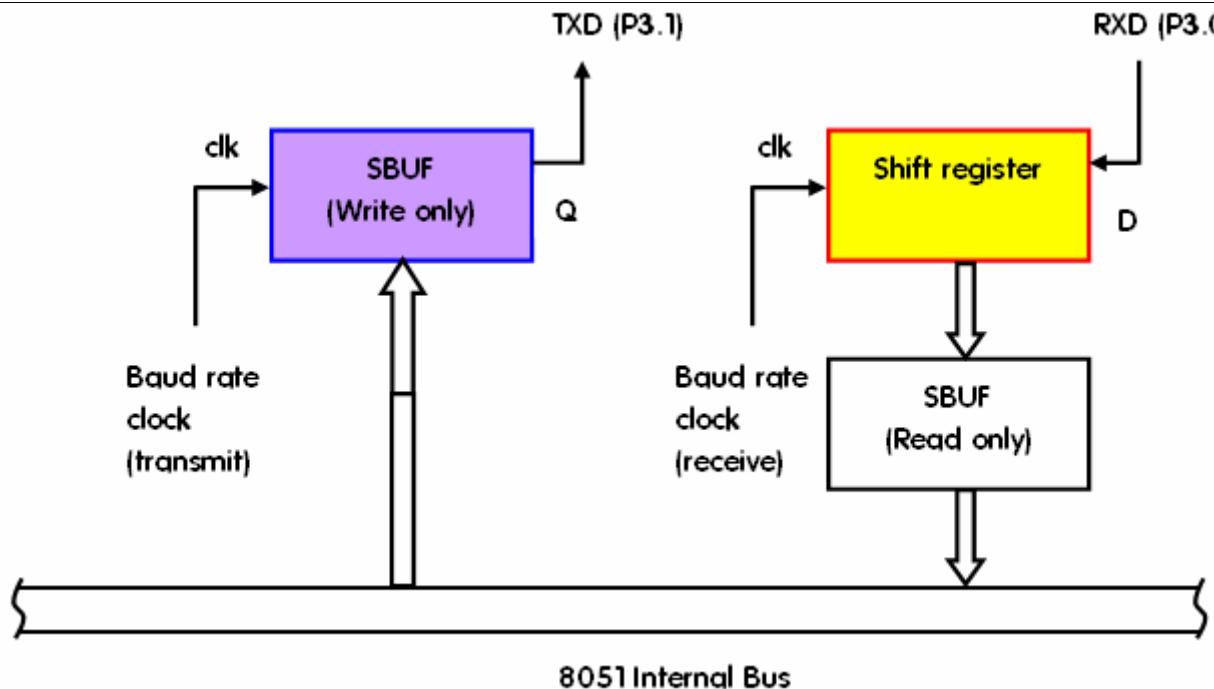
### 1 GIỚI THIỆU

Truyền dữ liệu nối tiếp của MCS51 có thể hoạt động ở nhiều kiểu riêng biệt trong phạm vi cho phép của tần số. Dữ liệu dạng song song được chuyển thành nối tiếp để truyền đi và dữ liệu nhận về dạng nối tiếp được chuyển thành song song.

Chân TxD (P3.1) là ngõ xuất dữ liệu đi và chân RxD (P3.0) là ngõ nhận dữ liệu về.

Đặc trưng của truyền dữ liệu nối tiếp là hoạt động song công có nghĩa là có thể thực hiện truyền và nhận dữ liệu cùng một lúc.

Hai thanh ghi chức năng đặc biệt phục vụ cho truyền dữ liệu là thanh ghi đệm SBUF và thanh ghi điều khiển SCON. Thanh ghi đệm SBUF nằm ở địa chỉ 99H có 2 chức năng: nếu vi điều khiển ghi dữ liệu lên thanh ghi sbuf thì dữ liệu đó sẽ được truyền đi, nếu hệ thống khác gởi dữ liệu đến thì sẽ được lưu vào thanh ghi đệm sbuf. Sơ đồ khối của hệ thống truyền dữ liệu như hình 4-19.



Hình 4-19. Sơ đồ khái niệm của truyền dữ liệu nối tiếp.

Thanh ghi điều khiển truyền dữ liệu SCON nằm ở địa chỉ 98H là thanh ghi cho phép truy suất bit bao gồm các bit trạng thái và các bit điều khiển. Các bit điều khiển dùng để thiết lập nhiều kiểu hoạt động truyền dữ liệu khác nhau, còn các bit trạng thái cho biết thời điểm kết thúc khi truyền xong một kí tự hoặc nhận xong một kí tự. Các bit trạng thái có thể được kiểm tra trong chương trình hoặc có thể lập trình để sinh ra ngắt.

Tần số hoạt động của truyền dữ liệu nối tiếp còn gọi tốc độ BAUD (số lượng bit dữ liệu được truyền đi trong một giây) có thể hoạt động cố định (sử dụng dao động trên chip) hoặc có thể thay đổi. Khi cần tốc độ Baud thay đổi thì phải sử dụng Timer 1 để tạo tốc độ baud.

## 2. THANH GHI ĐIỀU KHIỂN TRUYỀN DỮ LIỆU NỐI TIẾP:

Thanh ghi scon sẽ thiết lập các kiểu hoạt động truyền dữ liệu khác nhau cho MCS51. Cấu trúc của thanh ghi Scon như sau:



Bảng 4-11 tóm tắt thanh ghi điều khiển Port nối tiếp scon như sau :

Bit	Ký hiệu	Địa chỉ	Mô tả hoạt động
7	SM0	9FH	Bit chọn kiểu truyền nối tiếp: bit thứ 0.
6	SM1	9EH	Bit chọn kiểu truyền nối tiếp: bit thứ 1.
5	SM2	9DH	Bit cho phép truyền kết nối nhiều vi xử lý ở mode 2 và 3; RI sẽ không tích cực nếu bit thứ 9 đã thu vào là 0.
4	REN	9CH	Bit cho phép nhận kí tự. REN = 1 sẽ cho phép nhận kí tự.
3	TB8	9BH	Dùng để lưu bit 9 để truyền đi khi hoạt động ở mode 2 và 3. TB8 bằng 0 hay 1 là do người lập trình thiết lập.
2	RB8	9AH	Dùng để lưu bit 9 nhận về khi hoạt động ở mode 2 và 3.
1	TI	99H	Cờ báo hiệu này lên mức 1 khi truyền xong 1 kí tự và xóa bởi

			người lập trình để sẵn sàng truyền kí tự tiếp theo.
0	RI	98H	Cờ báo hiệu này lên mức 1 khi nhận xong 1 kí tự và xóa bởi người lập trình để sẵn sàng nhận kí tự dữ liệu tiếp theo.

**Bảng 4-11. Các bit trong thanh ghi điều khiển truyền dữ liệu.****3. CÁC KIỂU TRUYỀN DỮ LIỆU NỐI TIẾP:**

Trước khi truyền dữ liệu thì thanh ghi SCON phải được khởi tạo đúng kiểu. Ví dụ để khởi tạo truyền dữ liệu kiểu 1 thì 2 bit: SM0 SM1 = 01, bit cho phép thu: REN =1, và cờ ngắt truyền TI = 1 để sẵn sàng truyền, ta dùng lệnh sau : MOV SCON, #01010010b.

Truyền dữ liệu nối tiếp của MCS51 có 4 kiểu hoạt động tùy thuộc theo 4 trạng thái của 2 bit SM0 SM1 được liệt kê ở bảng 4-12.

Ba trong bốn kiểu cho phép truyền đồng bộ với mỗi kí tự thu hoặc phát sẽ được kết hợp với bit Start hoặc bit Stop.

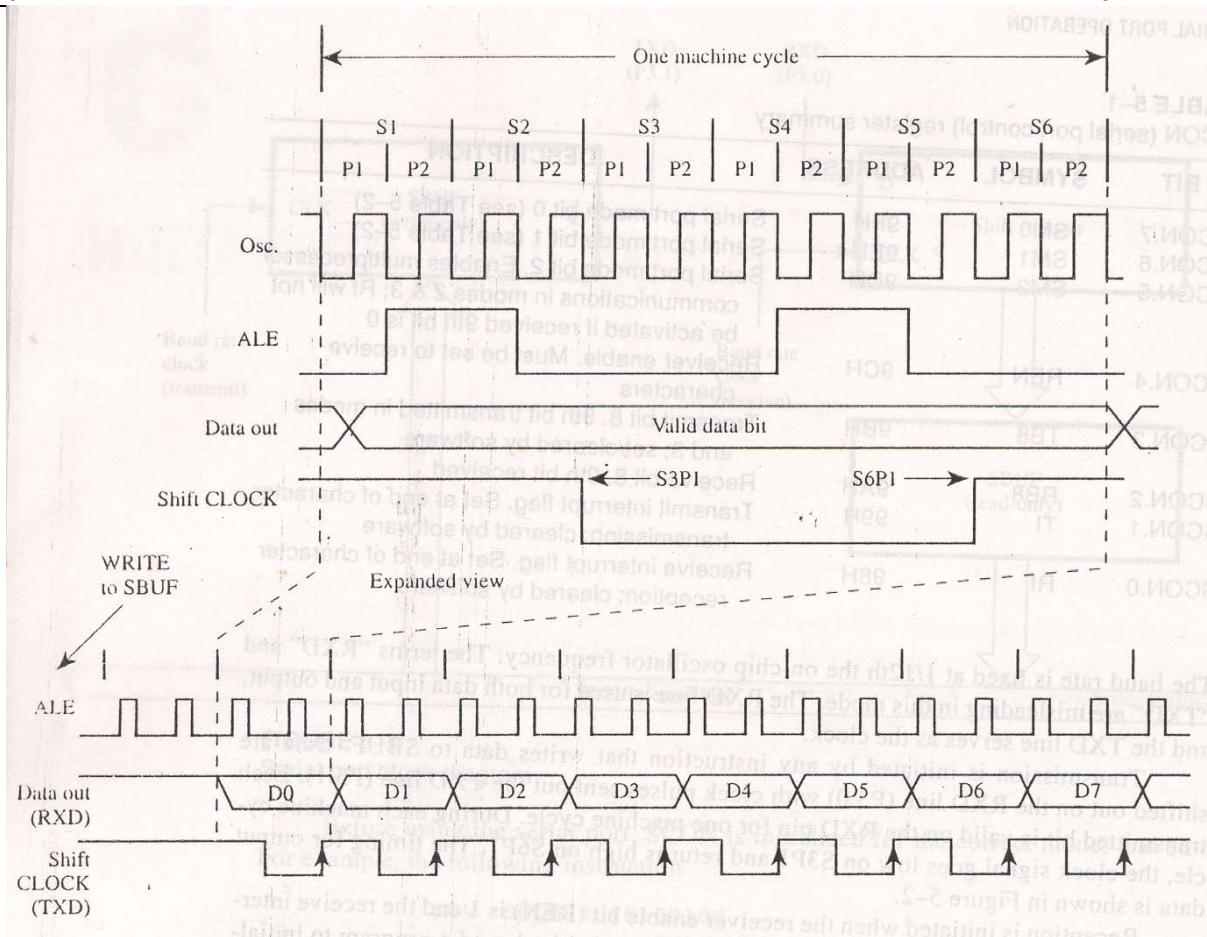
SM0	SM1	Kiểu	Mô tả	Tốc độ baud
0	0	0	Thanh ghi dịch	Cố định (tần số dao động f/12).
0	1	1	UART 8 bit	Thay đổi (được đặt bởi Timer).
1	0	2	UART 9 bit	Cố định (tần số dao động f/12 or f/64)
1	1	3	UART 9 bit	Thay đổi (được đặt bởi Timer).

**Bảng 4-12. Các kiểu truyền dữ liệu.****a. Truyền dữ liệu kiểu 0 – kiểu thanh ghi dịch 8 bit :**

Để định cấu hình cho truyền dữ liệu nối tiếp ở kiểu 0 thì 2 bit SM1 SM0 = 00. Dữ liệu nối tiếp nhận vào và dữ liệu truyền đi đều thông qua chân RxD, còn chân TxD thì dùng để dịch chuyển xung clock. 8 bit dữ liệu để truyền đi hoặc nhận về thì luôn bắt đầu với bit có trọng số nhỏ nhất LSB. Tốc độ Baud được thiết lập cố định ở tần số bằng  $\frac{1}{12}$  tần số dao động thạch anh trên Chip.

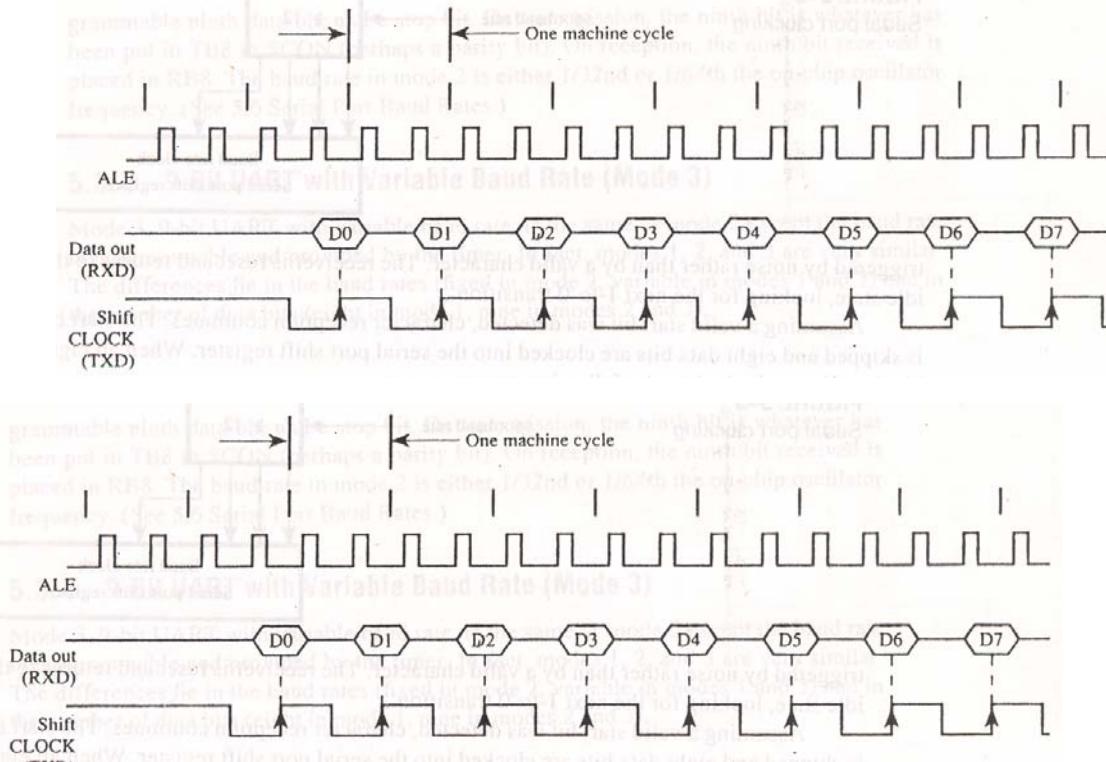
Khi thực hiện lệnh ghi dữ liệu lên thanh ghi sbuf thì quá trình truyền dữ liệu bắt đầu. Dữ liệu được dịch ra ngoài thông qua chân RxD cùng với các xung nhịp cũng được gửi ra ngoài thông qua chân TxD. Mỗi bit truyền đi chỉ có xuất hiện trên chân RxD trong khoảng thời gian một chu kỳ máy. Trong khoảng thời gian của mỗi chu kỳ máy, tín hiệu xung clock xuống mức thấp tại thời điểm S3P1 và lên mức cao tại thời điểm S6P1 trong giản đồ thời gian hình 4-20.

Quá trình nhận được khởi động khi bit cho phép nhận REN = 1 và cờ nhận RI = 0. Nguyên tắc chung là khởi tạo bit REN = 1 ở đầu chương trình để khởi động truyền dữ liệu, và xóa bit RI để sẵn sàng nhận dữ liệu vào. Khi bit RI bị xóa, các xung clock sẽ xuất ra bên ngoài thông qua chân TxD, bắt đầu chu kỳ máy kế tiếp thì dữ liệu từ bên ngoài sẽ được dịch vào bên trong thông qua chân RxD.



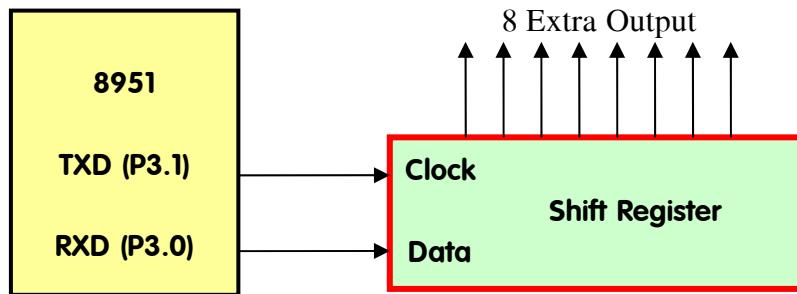
Hình 4-20. Giản đồ thời gian.

Biểu đồ thời gian của dữ liệu nối tiếp truyền vào vi điều khiển ở kiểu 0 như sau :



Hình 4-21. Giản đồ thời gian truyền dữ liệu mod 0.

Một ứng dụng cụ thể sử dụng mode 0 là dùng để mở rộng thêm số lượng ngõ ra cho MCS51 với cách thức thực hiện như sau: một thanh ghi dịch từ nối tiếp thành song song được nối đến các đường TxD và RxD của MCS51 để mở rộng thêm 8 đường ra như hình 4-22. Nếu dùng thêm nhiều thanh ghi dịch mắc nối tiếp vào thanh ghi dịch đầu tiên sẽ mở rộng được nhiều ngõ ra.



Hình 4-22. Một ứng dụng kiểu 0 để tăng thêm ngõ ra bằng thanh ghi dịch.

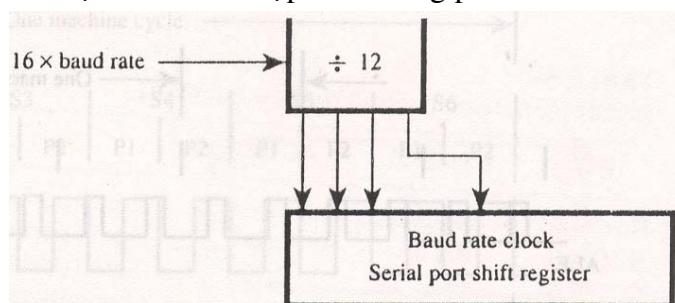
#### b. Truyền dữ liệu kiểu 1 – Thu phát bất đồng bộ 8 bit với tốc độ Baud thay đổi :

Trong mode này, truyền dữ liệu nối tiếp hoạt động bất đồng bộ UART 8 bit có tốc độ Baud thay đổi được. UART là bộ thu và phát dữ liệu nối tiếp với mỗi ký tự dữ liệu luôn bắt đầu bằng 1 bit Start (ở mức 0) và kết thúc bằng 1 bit Stop (ở mức 1), bit parity đôi khi được ghép vào giữa bit dữ liệu sau cùng và bit Stop.

Trong kiểu này, 10 bit dữ liệu sẽ phát đi ở chân TxD và nếu nhận thì sẽ nhận ở chân RxD. 10 bit đó bao gồm: 1 bit start, 8 bit data (LSB là bit đầu tiên), và 1 bit stop. Đối với hoạt động nhận dữ liệu thì bit Stop được đưa vào bit RB8 trong thanh ghi SCON.

Trong MCS51, tốc độ Baud được thiết lập bởi tốc độ tràn của Timer T1. Đối với họ 52 có 3 timer thì tốc độ baud có thể thiết lập bởi tốc độ tràn của timer T1 hoặc timer T2 hoặc cả 2 timer T1 và T2: một timer cho máy phát và 1 timer cho máy thu.

Nguồn cung cấp xung clock để đồng bộ các thanh ghi truyền dữ liệu nối tiếp hoạt động ở kiểu 1, 2, 3 được thiết lập bởi bộ đếm 16 như hình 4-23, ngõ ra của bộ đếm là xung clock tạo tốc độ baud. Xung ngõ vào của bộ đếm có thể lập trình bằng phần mềm.

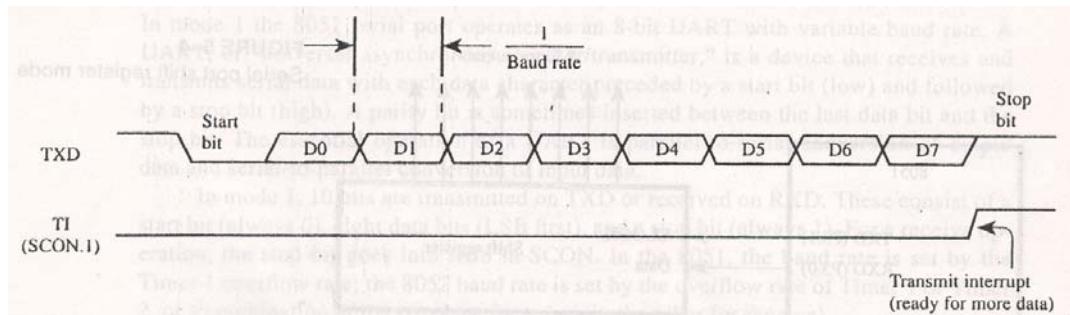


Hình 4-23. Cung cấp xung cho truyền dữ liệu nối tiếp.

Khi có một lệnh ghi dữ liệu lên thanh ghi sbuf thì quá trình truyền dữ liệu bắt đầu nhưng nó chưa truyền mà chờ cho đến khi bộ chia 16 (cung cấp tốc độ Baud cho truyền dữ liệu nối tiếp) bị tràn. Dữ liệu được xuất ra trên chân TxD bắt đầu với bit start sau là 8 bit data và sau cùng là bit stop. Các cờ phát TI được nâng lên mức 1 cùng lúc với thời điểm xuất hiện bit Stop trên chân TxD như hình 4-24.

Quá trình nhận dữ liệu được khởi động khi có sự chuyển đổi từ mức 1 sang mức 0 ở ngõ vào RxD. Bộ đếm 4 bit được reset ngay lập tức để sắp xếp bit dữ liệu đang đến từ ngõ vào RxD. Mỗi bit dữ liệu đến được lấy mẫu ở trạng thái đếm thứ 8 trong một chu kỳ 16 trạng thái của bộ đếm 4 bit.

Khi có sự chuyển trạng thái từ 1 xuống 0 ở ngõ vào RxD của bộ thu thì trạng thái 0 này phải tồn tại trong 8 trạng thái liên tục của bộ đếm 4 bit. Nếu trường hợp này không đúng thì bộ thu xem như bị tác động bởi tín hiệu nhiễu. Bộ thu sẽ reset và trở về trạng thái nghỉ và chờ sự chuyển trạng thái tiếp theo.



**Hình 4-24. Cờ báo phát xong dữ liệu TI.**

Giả sử việc kiểm tra bit Start là hợp lệ thì bit Start sẽ được bỏ qua và 8 bit data được nhận vào thanh ghi dịch nối tiếp.

Khi tất cả 8 bit được ghi vào thanh ghi dịch thì 3 công việc sau sẽ được thực hiện tiếp theo:

- Bit thứ 9 (bit Stop) được dịch vào bit RB8 trong SCON.
- 8 bit data được nạp vào thanh ghi SBUF.
- Cờ ngắt nhận RI =1.

Tuy nhiên, 3 công việc trên chỉ xảy ra nếu hai điều kiện sau tồn tại :

- RI = 0 và
- SM2 = 1 và bit Stop nhận được = 1 hoặc SM2 = 0.

#### c. Truyền dữ liệu kiểu 2 – Thu phát bất đồng bộ 9 bit với tốc độ Baud cố định :

Khi SM1 SM0 = 10 thì truyền dữ liệu hoạt động ở kiểu 2 có tốc độ Baud cố định. Có 11 bit được phát hoặc thu : 1 bit Start, 8 bit data, 1 bit data thứ 9 được lập trình và 1 bit Stop. Khi phát thì bit thứ 9 được đặt vào TB8 của SCON (có thể bit parity). Khi thu thì bit thứ 9 được đặt vào bit RB8 của thanh ghi SCON. Tốc độ Baud trong mode 2 bằng 1/12 hoặc 1/64 tần số dao động trên Chip.

#### d. Truyền dữ liệu kiểu 3 – Thu phát bất đồng bộ 9 bit với tốc độ Baud thay đổi :

Khi SM1 SM0 = 11 thì truyền dữ liệu hoạt động ở kiểu 3 là kiểu UART 9 bit có tốc độ Baud thay đổi. Kiểu 3 tương tự kiểu 2 ngoại trừ tốc độ Baud được lập trình và được cung cấp bởi Timer. Các kiểu 1, kiểu 2 và kiểu 3 rất giống nhau, những điểm khác nhau là ở tốc độ Baud (kiểu 2 cố định, kiểu 1 và kiểu 3 thay đổi) và số bit dữ liệu (kiểu 1 có 8 bit, kiểu 2 và kiểu 3 có 9 bit data).

### 4. KHỞI TẠO VÀ TRUY XUẤT CÁC THANH GHI TRUYỀN DỮ LIỆU NỐI TIẾP:

#### a. Bit điều khiển cho phép nhận dữ liệu (Receive Enable) :

Để cho phép thu dữ liệu thì chương trình phải làm cho bit REN = 1 và điều này được thực hiện ở đầu chương trình.

Ta có thể khởi tạo cho phép truyền dữ liệu bằng lệnh :

**Setb ren** hoặc **MOV SCON, # Xxx1xxxxb**

#### b. Bit dữ liệu thứ 9:

Bit dữ liệu thứ 9 được phát trong kiểu 2 và kiểu 3 phải được nạp vào bit TB8 bằng phần mềm có nghĩa là người lập trình phải thực hiện công việc này trước khi truyền dữ liệu đi, còn bit dữ liệu thứ 9 của dữ liệu thu được thì tự động đặt vào trong bit RB8.

Phần mềm có thể hoặc không đòi hỏi bit dữ liệu thứ 9 tham gia vào quá trình truyền dữ liệu tùy thuộc vào đặc tính của các thiết bị nối tiếp kết nối với nhau thiết lập ra qui định. Bit dữ liệu thứ 9 đóng 1 vai trò quan trọng trong truyền thông nhiều vi xử lý.

#### c. Bit kiểm tra chẵn lẻ Parity :

Bit thứ 9 thường được dùng là bit kiểm tra chẵn lẻ. Ở mỗi chu kỳ máy, bit P trong thanh ghi trạng thái PSW bằng 1 hay bằng 0 tùy thuộc vào quá trình kiểm tra chẵn 8 bit dữ liệu chứa trong thanh ghi A.

**Ví dụ 49:** Nếu hệ thống truyền dữ liệu yêu cầu 8 bit data cộng thêm 1 bit kiểm tra chẵn, thì các lệnh sau đây sẽ phát 8 bit trong thanh ghi A cộng với bit kiểm tra chẵn được cộng vào bit thứ 9.

MOV C,P	;chuyển cờ chẵn lẻ P sang cờ C
MOV TB8,C	;chuyển cờ C sang bit TB8 để chuẩn bị truyền đi
MOV SBUF,A	;truyền dữ liệu 8 bit trong A và bit thứ 9 trong TB8 đi.

Nếu kiểm tra lẻ được yêu cầu thì các lệnh trên được sửa lại là :

MOV C,P	;chuyển cờ chẵn lẻ P sang cờ C
CPL C	;nghịch đảo chẵn thành lẻ
MOV TB8,C	;chuyển cờ C sang bit TB8 để chuẩn bị truyền đi
MOV SBUF,A	;truyền dữ liệu 8 bit trong A và bit thứ 9 trong TB8 đi.

Trong kiểu 1 ta vẫn có thể sử dụng bit kiểm tra chẵn lẻ như sau: 8 bit data được phát trong kiểu 1 có thể bao gồm 7 bit dữ liệu, và 1 bit kiểm tra chẵn lẻ. Để phát một mã ASCII 7 bit với 1 bit kiểm tra chẵn vào 8 bit, các lệnh sau đây được dùng:

MOV C, P	; Đưa Parity chẵn vào C
MOV ACC.7, C	; Đưa Parity chẵn vào bit MSB của A
MOV SBUF, A	; Gởi bit data cùng bit Parity chẵn

#### d. Cờ ngắt :

Cờ ngắt nhận RI và phát TI trong thanh ghi SCON đóng một vai trò quan trọng trong truyền dữ liệu của MCS51. Cả hai bit đều được set bởi phần cứng nhưng phải xóa bởi phần mềm.

Điển hình là cờ RI được set ở mức 1 khi kết thúc quá trình nhận đầy đủ 1 kí tự và cho biết thanh ghi đệm thu đã đầy. Trạng thái của cờ RI có thể kiểm tra bằng phần mềm hoặc có thể lập trình để sinh ra ngắn. Nếu muốn nhận một ký tự từ một thiết bị đã được kết nối đến Port nối tiếp, thì chương trình phải chờ cho đến khi cờ RI = 1, sau đó xóa cờ RI và đọc ký tự từ thanh ghi SBUF. Quá trình này được lập trình như sau :

WAIT : JNB RI, WAIT	:Kiểm tra RI xem có bằng 1 hay không. Chờ nếu = 0
CLR RI	:khi cờ RI = 1 thì đã nhận xong dữ liệu và xóa cờ RI
MOV A, SBUF	:đọc ký tự nhận được từ thanh ghi Sbuf

Cờ TI lên mức 1 cho biết đã phát xong ký tự và cho biết thanh ghi đệm sbuf đã rỗng. Nếu muốn gửi 1 ký tự đến một thiết bị đã được kết nối đến Port nối tiếp thì trước tiên phải kiểm tra xem Port nối tiếp đã sẵn sàng chưa. Nếu ký tự trước đang được gửi đi, thì phải chờ cho đến khi kết thúc quá trình gửi. Các lệnh sau đây dùng để phát một ký tự trong thanh ghi A ra :

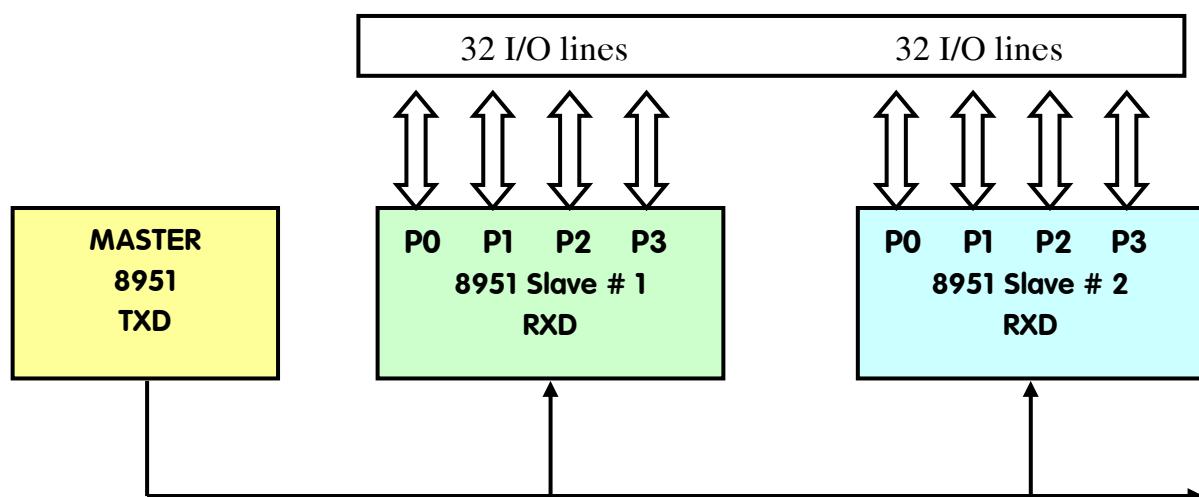
WAIT : JNB TI, WAIT	:Kiểm tra TI có bằng 1 hay không và chờ nếu bằng 0.
CLR TI	:Xóa cờ ngắn thu TI
MOV SBUF,A	:gửi nội dung trong thanh ghi A đi

Hoặc

MOV SBUF,A	:gửi nội dung trong thanh ghi A đi
WAIT : JNB TI, WAIT	:Kiểm tra TI có bằng 1 hay không và chờ nếu bằng 0.
CLR TI	:Xóa cờ ngắn thu TI

#### 5. TRUYỀN DỮ LIỆU NỐI TIẾP TRONG HỆ THỐNG NHIỀU VI ĐIỀU KHIỂN:

Kiểu 2 và kiểu 3 có một chức năng đặc biệt cho việc truyền thông đa xử lý. Ở các mode 2 và 3, 9 bit dữ liệu được thu và bit thứ 9 được lưu vào bit RB8. Truyền dữ liệu có thể lập trình sao cho khi thu được bit Stop thì ngắn của truyền dữ liệu nối tiếp tác động chỉ khi bit RB8 = 1. Cấu trúc này được phép bằng cách set bit SM2 = 1 trong thanh ghi SCON. Kiểu này được ứng dụng trong mạng sử dụng nhiều MCS51 được tổ chức theo cấu hình máy chủ và máy tớ như hình 4-25.



Hình 4-25. Kết nối nhiều vi xử lý.

Trong cấu hình kết nối ở trên thì mỗi một vi xử lý tớ sẽ có một địa chỉ duy nhất do chúng ta qui định.

Khi bộ xử lý chủ muốn phát một khối dữ liệu đến một trong các bộ xử lý tớ thì trước tiên vi xử lý chủ phải gởi ra 1 byte địa chỉ để nhận diện bộ xử lý tớ muốn kết nối.

Byte địa chỉ được phân biệt với byte dữ liệu bởi bit thứ 9: trong byte địa chỉ thì bit thứ 9 bằng 1 và trong byte dữ liệu thì bit thứ 9 bằng 0.

Các vi xử lý tớ sau khi nhận được byte địa chỉ sẽ biết được vi xử lý chủ muốn giao tiếp tớ nào. Khi có vi xử lý tớ được phép thì nó sẽ xóa bit SM2 để bắt đầu nhận các byte dữ liệu tiếp theo. Còn các vi xử lý không được phép thì vẫn giữ nguyên bit SM2=1 để không nhận các byte dữ liệu truyền giữa vi xử lý chủ và vi xử lý tớ đang được phép. Vi xử lý tớ sau khi kết nối với vi xử lý chủ xong thì phải làm cho bit SM2=1 để sẵn sàng kết nối cho những lần tiếp theo.

Sau khi thực hiện xong việc trao đổi dữ liệu thì vi xử lý muốn truy xuất một vi xử lý khác thì phải tạo ra một địa chỉ mới và vi xử lý tớ tương ứng với địa chỉ đó được phép và hoạt động giống như vừa trình bày.

#### **6. THẾT LẬP TỐC ĐỘ TRUYỀN DỮ LIỆU NỐI TIẾP:**

Truyền dữ liệu nối tiếp nếu hoạt động ở kiểu 0 và kiểu 2 thì có tốc độ truyền cố định. Trong kiểu 0 thì tốc độ truyền bằng  $\frac{1}{12}$  tần số dao động trên Chip. Nếu sử dụng thạch anh 12 MHz thì tốc độ truyền của kiểu 0 là 1MHz như hình 4-26a.

Trong thanh ghi PCON có một bit SMOD có chức năng làm tăng gấp đôi tốc độ baud, mặc nhiên sau khi reset hệ thống thì bit SMOD = 0 thì các kiểu truyền dữ liệu hoạt động với tốc độ qui định, khi bit SMOD = 1 thì tốc độ tăng gấp đôi.

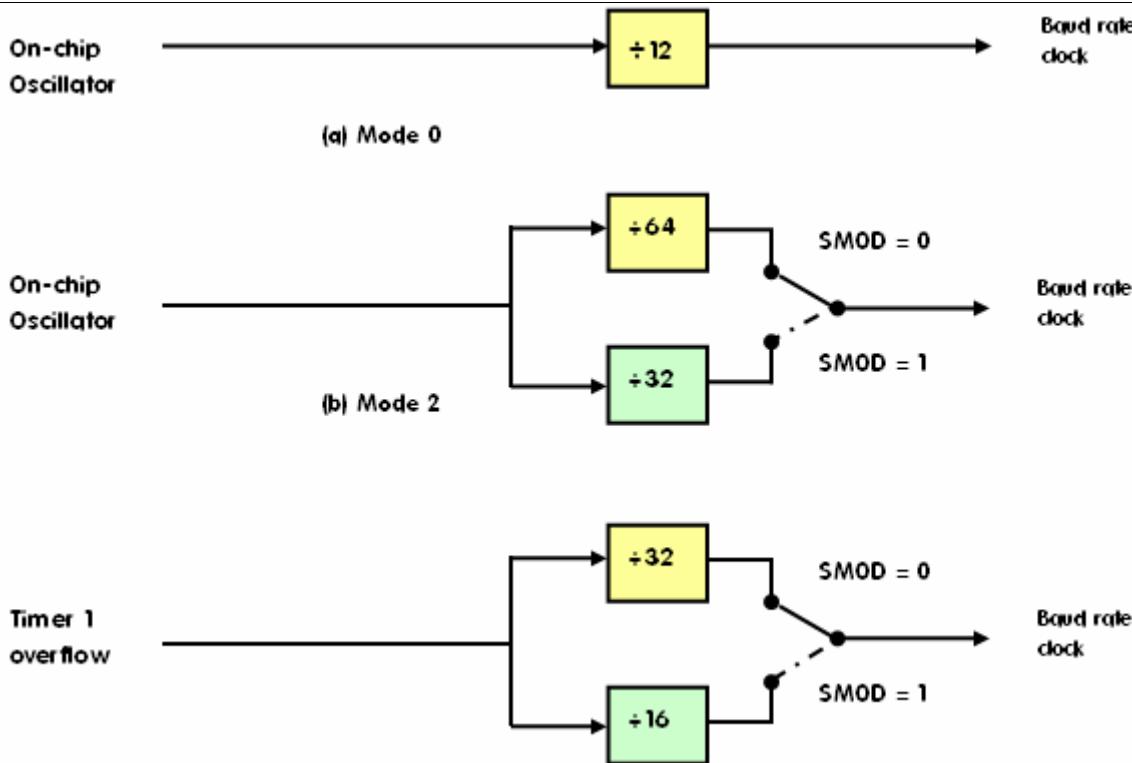
Ví dụ trong kiểu 2, tốc độ truyền có thể tăng gấp đôi từ giá trị mặc định 1/64 tần số dao động trên Chip (SMOD = 0) lên đến 1/32 tần số dao động trên Chip (ứng với SMOD =1) như hình 4-26b.

Do thanh ghi PCON không cho phép truy suất bit nên để set bit SMOD mà không thay đổi các bit khác của thanh ghi PCON thì phải thực hiện lệnh sau.

Lệnh sau đây set bit SMOD để tăng gấp đôi tốc độ truyền:

**OR PCON, #1000 0000b ;bit Smod ở vị trí thứ 7**

Các tốc độ Baud trong kiểu 1 và kiểu 3 của MCS51 được xác định bởi tốc độ tràn của Timer 1. Bởi vì Timer hoạt động ở tần số tương đối cao nên phải chia cho 32 khi bit smod = 0 và chia cho 16 nếu SMOD = 1 trước khi cung cấp xung clock để thiết lập tốc độ Baud cho Port nối tiếp. Tốc độ Baud ở kiểu 1 và 3 của MCS51 được xác định bởi tốc độ tràn của Timer 1 hoặc Timer 2, hoặc cả 2 như hình 4-26c.



Hình 4-26. Thiết lập tốc độ Baud.

**Thiết lập tốc độ Baud dùng timer 1:**

Muốn có tốc độ Baud thì ta khởi tạo thanh ghi TMOD ở kiểu tự động nạp 8 bit (kiểu 2) và đặt giá trị nạp lại vào thanh ghi TH1 của Timer 1 để tạo ra tốc độ tràn chính xác để thiết lập tốc độ Baud. Thanh ghi tmod được khởi tạo để thiết lập tốc độ baud như sau:

**MOV TMOD,#0010xxxxB** ;chỉ quan tâm đến timer 1

Một cách khác để tạo tốc độ baud là nhận tín hiệu xung clock từ bên ngoài đưa đến ngõ vào T1. Công thức chung để xác định tốc độ Baud trong mode 1 và mode 3 là :

$$\text{BAUD RATE} = \text{TIMER 1 OVERFLOW RATE} \div 32$$

**Ví dụ 50:** Truyền dữ liệu cần tốc độ baud là 1200 thì ta tính toán như sau:

Tốc độ tràn của timer 1 bằng  $1200 \times 32 = 38,4\text{KHz}$ . Nếu hệ thống sử dụng thạch anh 12 MHz thì xung cung cấp cho Timer 1 đếm có tần số là 1 MHz hay 1000KHz. Vậy để đạt tốc độ tràn 38,4 KHz thì ta tính được số lượng xung đếm cho mỗi chu kỳ tràn là  $1000\text{ KHz}/38,4\text{ KHz} = 26,4$  xung (làm tròn bằng 26).

Do các Timer đếm lên và thời điểm tràn xảy ra khi chuyển trạng thái đếm từ FFH → 00H nên ta phải nạp giá trị bắt đầu từ  $(256 - 26 = 230)$  để từ giá trị này timer 1 đếm lên 26 xung nữa thì sinh ra tràn. Giá trị 230 được nạp vào thanh ghi TH1 để tự động nạp lại cho thanh ghi TL1 khi tràn bằng lệnh: “**MOV TH1,#230**”. Bạn có thể không cần phải tính toán ra giá trị 230 mà có thể thay bằng lệnh : “**MOV TH1,#-26**” thì trình biên dịch sẽ tính cho bạn.

Bảng tóm tắt tốc độ Baud ứng với 2 loại thạch anh 12 MHz và 11,059 MHz :

Tốc độ baud	Tần số thạch anh	SMOD	Giá trị nạp cho TH1	Tốc độ thực	Sai số
9600	12MHz	1	- 7 (F9H)	8923	7%
2400	12MHz	0	-13 (F3H)	2404	0,16%
1200	12MHz	0	-26 (E6H)	1202	~0%
19200	11,059MHz	1	-3 (FDH)	19200	0%
9600	11,059MHz	0	-3 (FDH)	9600	0%
2400	11,059MHz	0	-12 (F4H)	2400	0%
1200	11,059MHz	0	-24 (E8H)	1200	0%

Bảng 4-13. Tóm tắt tốc độ baud.

**Ví dụ 51:** Hãy khởi tạo truyền dữ liệu nối tiếp hoạt động như UART 8 bit ở tốc độ Baud 2400, dùng Timer 1 để tạo tốc độ Baud.

Chương trình sau sẽ thiết lập đúng theo yêu cầu đề ra:

```

MOV SCON, #01010010B : Port nối tiếp mode 1.
MOV TMOD, #20H       : Timer 1 mode 2
MOV TH1, # -13        : Nạp vào bộ đếm tốc độ 2400 Baud.
SETB TR1              : Start Timer 1.

```

Trong thanh ghi SCON có: hai bit SM0 SM1 = 01 thiết lập mode UART 8 bit, bit REN = 1 cho phép sẵn sàng nhận dữ liệu, bit TI = 1 báo cho biết thanh ghi đếm rỗng sẵn sàng cho phép phát dữ liệu.

Thanh ghi TMOD có hai bit M1M0 = 10 để thiết lập Timer 1 ở mode 2 tự động nạp 8 bit. Lệnh setb TR1 cho phép Timer làm việc tạo tốc độ baud.

Từ tốc độ Baud 2400 ta tính được tốc độ tràn cho Timer 1 là  $2400 \times 32 = 76,8$  KHz và giả sử Timer 1 đếm xung nội ở tần số 1000 KHz (ứng với thạch anh 12 MHz).

Vậy để đạt tốc độ tràn 76,8 KHz thì ta tính được số lượng xung đếm cho mỗi chu kỳ tràn là  $1000\text{KHz}/76,8\text{KHz} = 13,02$  xung (làm tròn bằng 13). Nên lệnh thứ 3 sẽ nạp giá trị -13 vào thanh ghi TH1 để tạo tốc độ baud là 2400.

Việc sử dụng truyền dữ liệu ở tốc độ baud nào tùy thuộc vào yêu cầu thực tế. Tốc độ càng cao thì dữ liệu truyền càng nhanh. Khi truyền nhiều dữ liệu thì ngoài tốc độ qui định thống nhất giữa 2 hệ thống kết nối với nhau còn phải quan tâm đến tốc độ xử lý dữ liệu nhận về và lấy dữ liệu gởi đi để không bị mất dữ liệu trong quá trình truyền và nhận. Một trong những giải pháp để kiểm tra xem dữ liệu có bị mất hay không thì phải sử dụng thủ tục bắt tay.

## VII. HOẠT ĐỘNG NGẮT CỦA VI ĐIỀU KHIỂN MCS51

### 1 GIỚI THIỆU

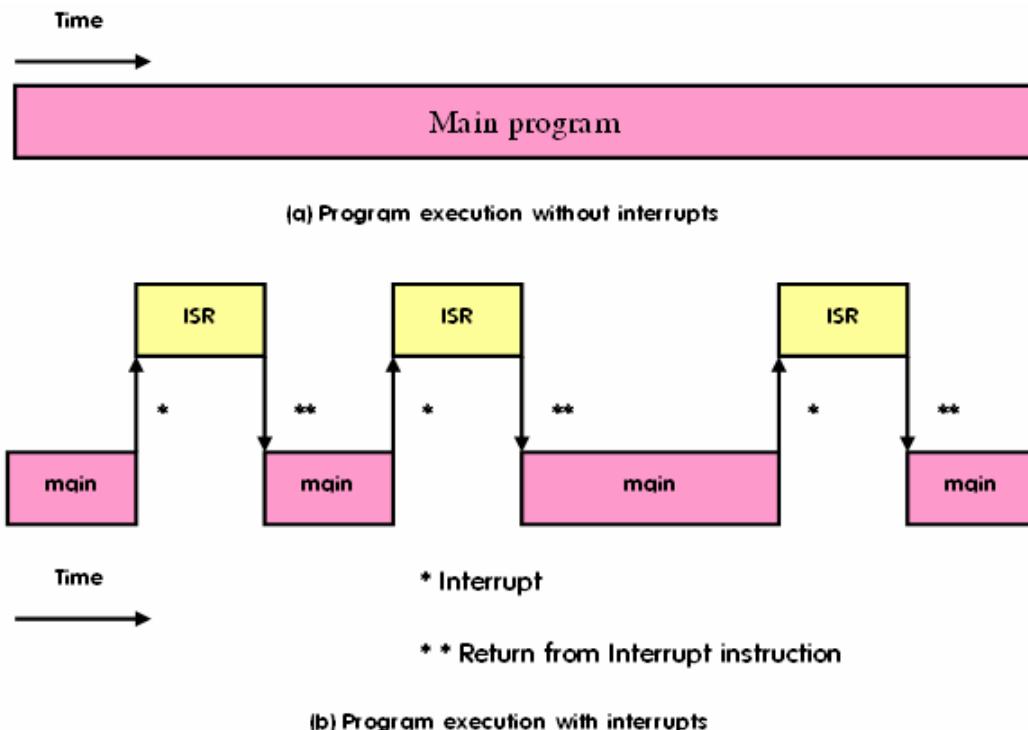
Ngắt sử dụng trong vi xử lý hay vi điều khiển hoạt động như sau: vi xử lý hay vi điều khiển luôn thực hiện một chương trình mà ta thường gọi là chương trình chính, khi có một sự tác động từ bên ngoài bằng phần cứng hay sự tác động bên trong làm cho vi xử lý ngừng thực hiện chương trình chính để thực hiện một chương trình khác (còn gọi là chương trình phục vụ ngắt ISR) và sau khi thực hiện xong vi xử lý trở lại thực hiện tiếp chương trình chính. Quá trình làm gián đoạn vi xử lý thực hiện chương trình xem như là ngắt.

Có nhiều sự tác động làm ngừng chương trình chính gọi là các nguồn ngắt, trong vi điều khiển khi timer/counter đếm tràn sẽ tạo ra ngắt. Ngắt đóng một vai trò quan trọng trong lập trình điều khiển.

Khi sử dụng ngắt sẽ cho phép vi xử lý hay vi điều khiển đáp ứng nhiều sự kiện quan trọng và giải quyết sự kiện đó trong khi chương trình khác đang thực thi. Ví dụ: trong vi điều khiển đang thực hiện chương trình chính thì có dữ liệu từ hệ thống khác gửi đến, khi đó vi điều khiển ngừng chương trình chính để thực hiện chương trình phục vụ ngắt nhận dữ liệu xong rồi trở lại tiếp tục thực hiện chương trình chính, hoặc có một tín hiệu báo ngắt từ bên ngoài thì vi điều khiển sẽ ngừng thực hiện chương trình chính để thực hiện chương trình ngắt rồi tiếp tục thực hiện chương trình chính.

Ta có thể sử dụng ngắt để yêu cầu vi điều khiển thực hiện nhiều chương trình cùng một lúc có nghĩa là các chương trình được thực hiện xoay vòng.

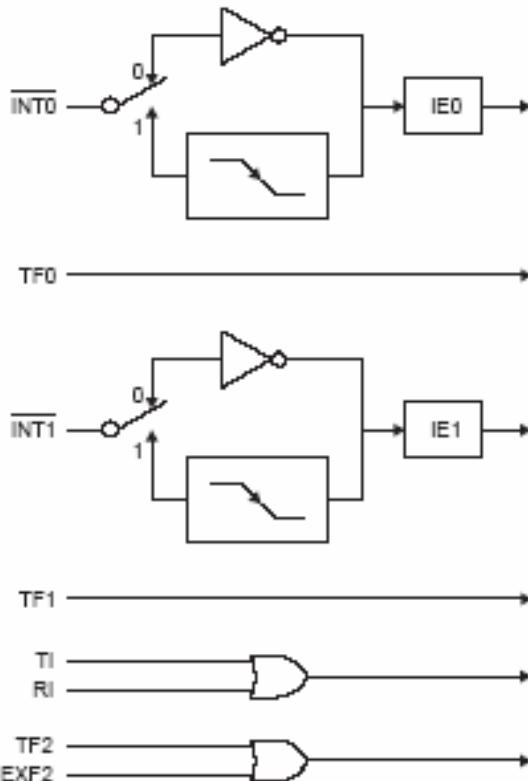
Ta có thể minh họa quá trình thực hiện 1 chương trình trong trường hợp có ngắt và không có ngắt như hình 4-27.



**Hình 4-27. Vi điều khiển thực hiện chương trình chính trong 2 trường hợp không và có ngắt.**

Trong đó : Ký hiệu \* cho biết vi điều khiển ngừng chương trình chính để thực thi chương trình con phục vụ ngắt ISR. Còn ký hiệu \*\* cho biết vi điều khiển quay trở lại thực hiện tiếp chương trình chính sau khi thực hiện xong chương trình con phục vụ ngắt ISR.

## 2. TỔ CHỨC NGẮT:



Hình 4-28. Vi điều khiển 89C52 có 6 nguồn ngắt.

Vi điều khiển 89C51 có 5 nguồn ngắt: 2 ngắt ngoài, 2 ngắt Timer và một ngắt Port nối tiếp. Vi điều khiển 89C52 có thêm một nguồn ngắt là cửa timer T2 như hình 4-28. Mặc nhiên khi vi điều khiển bị reset thì tất cả các ngắt sẽ mất tác dụng và được cho phép bởi phần mềm.

Trong trường hợp có hai hoặc nhiều nguồn ngắt tác động đồng thời hoặc vi điều khiển đang phục vụ ngắt thì xuất hiện một ngắt khác, khi đó sẽ có hai cách giải quyết: là kiểm tra liên tiếp và sử dụng chế độ ưu tiên.

## 3. CHO PHÉP NGẮT/CẤM NGẮT:

Trước tiên chúng ta phải hiểu cho phép và không cho phép ngắt là như thế nào? Khi ta cho phép ngắt và khi ngắt tác động thì vi điều khiển sẽ ngừng chương trình chính để thực hiện chương trình con phục vụ ngắt, còn khi không cho phép thì dù có sự tác động đến ngắt vi điều khiển vẫn tiếp tục thực hiện chương trình chính – không thực hiện chương trình phục vụ ngắt.

Trong vi điều khiển có 1 thanh ghi IE (Interrupt Enable) ở tại địa chỉ 0A8H có chức năng cho phép/cấm ngắt. Ta sử dụng thanh ghi này để cho phép hay không cho phép đối với từng nguồn ngắt và cho toàn bộ các nguồn ngắt. Tổ chức của thanh ghi như sau:



Hoạt động của từng bit trong thanh ghi cho phép ngắt IE được tóm tắt trong bảng 4-14:

Bit	Kí hiệu	Địa chỉ bit	Chức năng (Enable = 1; Disable = 0)
IE.7	EA	AFH	Cho phép hoặc cấm toàn bộ các nguồn ngắt.
IE.6	-	AEH	Chưa dùng đến

<b>IE.5</b>	<b>ET2</b>	<b>ADH</b>	<b>Cho phép ngắt Timer 2 (8052).</b>
<b>IE.4</b>	<b>ES</b>	<b>ACH</b>	<b>Cho phép ngắt Port nối tiếp.</b>
<b>IE.3</b>	<b>ET1</b>	<b>ABH</b>	<b>Cho phép ngắt Timer 1.</b>
<b>IE.2</b>	<b>EX1</b>	<b>AAH</b>	<b>Cho phép ngắt ngoài External 1 (INT1).</b>
<b>IE.1</b>	<b>ET0</b>	<b>A9H</b>	<b>Cho phép ngắt Timer 0.</b>
<b>IE.0</b>	<b>EX0</b>	<b>A8H</b>	<b>Cho phép ngắt ngoài External 0 (INT0).</b>

**Bảng 4-14. Tóm tắt chức năng các bit của thanh ghi IE.**

Trong thanh ghi IE có bit IE.6 chưa dùng đến, bit IE.7 là bit cho phép/cấm ngắt toàn bộ các nguồn ngắt. Khi bit IE.7=0 thì cấm hết tất cả các nguồn ngắt, khi bit IE.7=1 thì cho phép tất cả các nguồn ngắt nhưng còn phụ thuộc vào từng bit điều khiển ngắt của từng nguồn ngắt.

**Ví dụ 52:** để cho phép Timer 1 ngắt ta có thể thực hiện trên bit:

SETB EA ;cho phép ngắt toàn bộ

SETB ET1 ;cho phép timer 1 ngắt

Hoặc có thể dùng lệnh sau:

MOV IE, #10001000B

Đối với yêu cầu của ví dụ trên thì 2 cách thực hiện trên là xong nhưng ta hãy so sánh 2 cách thực hiện và chú ý một vài điều trong lập trình:

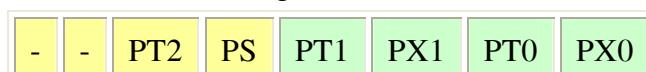
- Các lệnh của cách 1 không ảnh hưởng các bit còn lại trong thanh ghi IE.
- Cách thứ hai sẽ xóa các bit còn lại trong thanh ghi IE.

Ở đầu chương trình ta nên khởi gán IE với lệnh MOV BYTE, nhưng khi điều khiển cho phép hay cấm trong chương trình thì ta sẽ dùng các lệnh SET BIT và CLR BIT để tránh làm ảnh hưởng đến các bit khác trong thanh ghi IE.

#### 4. ĐIỀU KHIỂN ƯU TIÊN NGẮT:

Khi có nhiều nguồn ngắt tác động cùng lúc thì ngắt nào quan trọng cần thực hiện trước và ngắt nào không quan trọng thì thực hiện sau giống như các công việc mà ta giải quyết hằng ngày. Ngắt cũng được thiết kế có sự sắp xếp thứ tự ưu tiên từ thấp đến cao để người lập trình sắp xếp các nguồn ngắt theo yêu cầu công việc mà mình xử lý.

Thanh ghi có chức năng thiết lập chế độ ưu tiên trong vi điều khiển là thanh ghi IP (Interrupt Priority) tại địa chỉ 0B8H. Tổ chức của thanh ghi như sau:



Hoạt động của từng bit trong thanh ghi IP được tóm tắt trong bảng 4-15.

Bit	Kí hiệu	Địa chỉ bit	Chức năng
IP.7	-	-	Chưa sử dụng
IP.6	-	-	Chưa sử dụng
IP.5	PT2	BDH	Ưu tiên cho ngắt Timer 2 (8052).
IP.4	PS	BCH	Ưu tiên cho ngắt Port nối tiếp.

IP.3	PT1	BBH	Ưu tiên cho ngắt Timer 1.
IP.2	PX1	BAH	Ưu tiên cho ngắt ngoài External 1.
IP.1	PT0	B9H	Ưu tiên cho ngắt Timer 0.
IP.0	PX0	B8H	Ưu tiên cho ngắt ngoài External 0.

**Bảng 4-15. Tóm tắt chức năng các bit của thanh ghi IP.**

Khi reset hệ thống thì thanh ghi IP bị xóa và tất cả các ngắt ở mức ưu tiên thấp nhất.

Trong 89C51 có 2 mức ưu tiên thấp và 2 mức ưu tiên cao. Nếu vi điều khiển đang thực hiện chương trình con phục vụ ngắt có mức ưu tiên thấp và có một yêu cầu ngắt với mức ưu tiên cao hơn xuất hiện thì vi điều khiển phải ngừng thực hiện chương trình con phục vụ ngắt có mức ưu tiên thấp để thực hiện chương trình con phục vụ ngắt mới có ưu tiên cao hơn.

Ngược lại nếu vi điều khiển đang thực hiện chương trình con phục vụ ngắt có mức ưu tiên cao hơn và có yêu cầu ngắt với mức ưu tiên thấp hơn xuất hiện thì vi điều khiển vẫn tiếp tục thực hiện cho đến khi thực hiện xong chương trình phục vụ ngắt có ưu tiên cao hơn rồi mới thực hiện chương trình phục vụ ngắt có ưu tiên thấp đang yêu cầu.

Chương trình chính mà vi điều khiển luôn thực hiện trong một hệ thống thì ở mức thấp nhất, không có liên kết với yêu cầu ngắt nào, luôn luôn bị ngắt bất chấp ngắt ở mức ưu tiên cao hay thấp. Nếu có 2 yêu cầu ngắt với các ưu tiên khác nhau xuất hiện đồng thời thì yêu cầu ngắt có mức ưu tiên cao hơn sẽ được phục vụ trước.

### 5. KIỂM TRA NGẮT THEO VÒNG QUÉT LIÊN TIẾP:

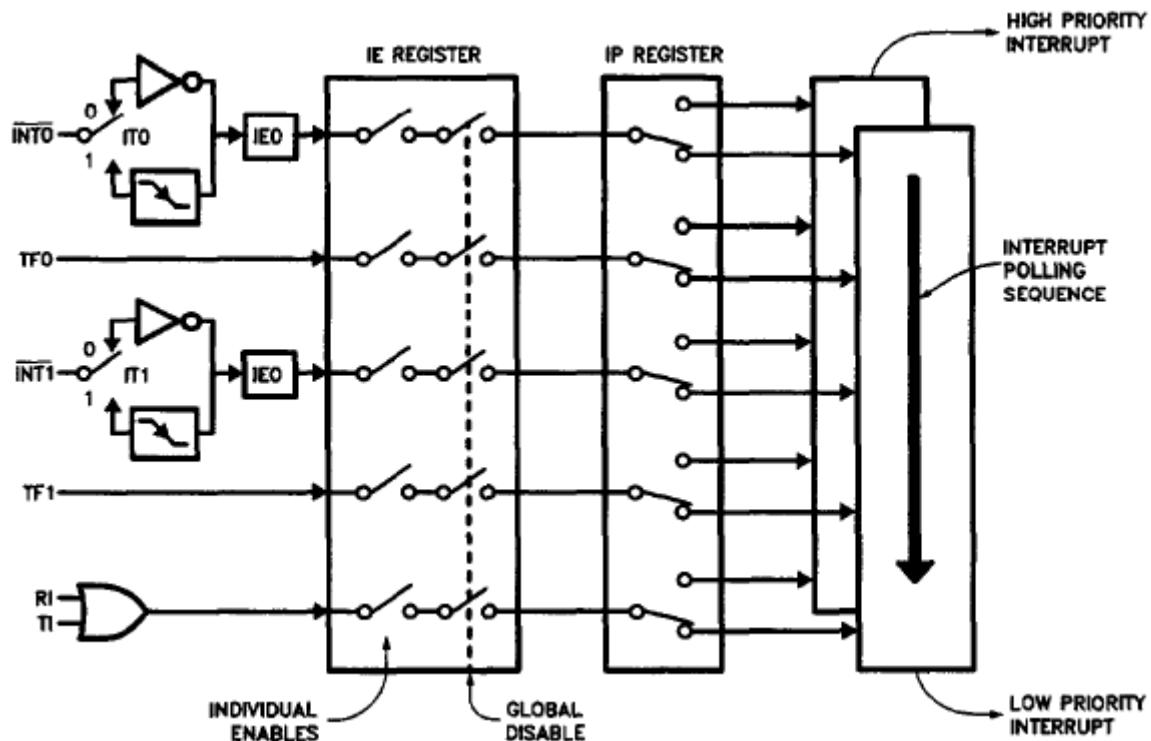
Nếu 2 yêu cầu ngắt có cùng mức ưu tiên xuất hiện đồng thời thì vòng quét kiểm tra liên tiếp sẽ xác định yêu cầu ngắt nào sẽ được phục vụ trước tiên. Vòng quét kiểm tra liên tiếp theo thứ tự ưu tiên từ trên xuống là: ngắt ngoài thứ 0 (INT0), ngắt timer T0, ngắt ngoài thứ 1 (INT1), ngắt Timer 1, ngắt truyền dữ liệu nối tiếp (Serial Port), ngắt timer 2. Hình 4-29 sẽ minh họa cho trình tự trên.

Quan sát trong hình 4-29 chúng ta thấy có 6 nguồn ngắt của 89C52 và tác dụng của các thanh ghi IE hoạt động như một contact On/Off còn thanh ghi IP hoạt động như một contact chuyển mạch giữa 2 vị trí để lựa chọn 1 trong 2.

Ta hãy bắt đầu từ thanh ghi IE trước: bit cho phép ngắt toàn cục (Global Enable) nếu được phép sẽ đóng toàn bộ các contact và tùy thuộc vào bit cho phép của từng nguồn ngắt có được phép hay không và chúng hoạt động cũng giống như một contact: nếu được phép thì đóng mạch và tín hiệu yêu cầu ngắt sẽ đưa vào bên trong để xử lý, nếu không được phép thì contact mở mạch nên tín hiệu yêu cầu ngắt sẽ không đưa vào bên trong và không được xử lý.

Tiếp theo là thanh ghi IP: tín hiệu sau khi ra khỏi thanh ghi IE thì đưa đến thanh ghi IP để sắp xếp ưu tiên cho các nguồn ngắt. Có 2 mức độ ưu tiên: mức ưu tiên cao và mức ưu tiên thấp. Nếu các nguồn nào có ưu tiên cao thì contact chuyển mạch sẽ đưa tín hiệu yêu cầu ngắt đó đến vòng kiểm tra có ưu tiên cao, nếu các nguồn nào có ưu tiên thấp thì contact chuyển mạch sẽ đưa tín hiệu yêu cầu ngắt đó đến vòng kiểm tra có ưu tiên thấp.

Vòng kiểm tra ngắt ưu tiên cao sẽ được thực hiện trước và sẽ kiểm tra theo thứ tự từ trên xuống và khi gặp yêu cầu ngắt nào thì yêu cầu ngắt đó sẽ được thực hiện. Sau đó tiếp tục thực hiện cho vòng kiểm tra ưu tiên ngắt có mức ưu tiên thấp hơn.



Hình 4-29. Cấu trúc ngắt của vi điều khiển.

Trong hình còn cho chúng ta thấy yêu cầu ngắt truyền dữ liệu nối tiếp tạo ra từ tổ hợp OR của 2 cờ báo nhận RI và cờ báo phát TI. Khi ngắt truyền dữ liệu xảy ra và ta muốn biết là do cờ nhận hay cờ phát tạo ra ngắt để thực hiện 2 công việc khác nhau thì ta phải kiểm tra cờ RI và TI để biết thực hiện công việc nào tương ứng.

Ví dụ trong truyền dữ liệu: khi có báo ngắt truyền dữ liệu thì ta phải kiểm tra xem cờ RI = 1 hay không? Nếu đúng thì hệ thống khác đang gửi dữ liệu đến và ta phải chuyển hướng chương trình phục vụ ngắt sang hướng nhận dữ liệu, nếu không phải thì chắc chắn là cờ TI=1 báo cho chúng ta biết rằng dữ liệu đã truyền đi xong và sẵn sàng truyền kí tự tiếp theo và khi đó ta phải chuyển hướng chương trình phục vụ ngắt sang phát dữ liệu tiếp theo.

Tương tự, các yêu cầu ngắt của Timer2 tạo ra từ tổ hợp OR của cờ tràn TF2 và cờ nhập ngoài EXF2.

Các bit cờ của các nguồn ngắt được tóm tắt ở bảng 4-16:

Interrupt	Flag	SFR Register and Bit Position
External 0	IE0	TCON 1
External 1	IE1	TCON 3
Timer 1	TF1	TCON 7
Timer 0	TF0	TCON 5
Serial Port	TI	SCON 1
Serial Port	RI	SCON 0
Timer 2	TF2	T2CON 7 (8052)
Timer 2	EXF2	T2CON 6 (8052)

Bảng 4-16. Tóm tắt các bit cờ của các nguồn ngắt.

## 6. XỬ LÝ NGẮT:

Khi tín hiệu yêu cầu ngắt xuất hiện và được chấp nhận bởi CPU thì CPU thực hiện các công việc sau:

- ❖ Nếu CPU đang thực hiện lệnh thì phải chờ thực hiện xong lệnh đang thực hiện.
- ❖ Giá trị của bộ đếm chương trình PC được cất vào bộ nhớ ngăn xếp (chính là địa chỉ của lệnh tiếp theo trong chương trình chính).
- ❖ Trạng thái ngắt hiện hành được lưu vào bên trong.
- ❖ Các yêu cầu ngắt khác sẽ bị ngăn lại.
- ❖ Địa chỉ của chương trình phục vụ ngắt tương ứng sẽ được nạp vào bộ đếm chương trình PC.
- ❖ Bắt đầu thực hiện chương trình phục vụ ngắt ISR.

Trong chương trình phục vụ ngắt luôn kết thúc bằng lệnh **RETI**. Khi gặp lệnh RETI thì CPU sẽ lấy lại địa chỉ của lệnh tiếp theo trong ngăn xếp trả lại cho thanh ghi PC để tiếp tục thực hiện các công việc tiếp theo của chương trình chính.

Chú ý: chương trình con phục vụ ngắt **không được làm mất hoặc làm sai địa chỉ của PC đã lưu trong ngăn xếp** nếu điều này xảy ra thì khi trở lại chương trình chính CPU sẽ không thực hiện tiếp công việc của chương trình chính và chúng ta cũng không xác định CPU đang làm gì và ở đâu. Khi đó chúng ta mất quyền kiểm soát vi xử lý.

Giống như ta đang đọc một cuốn sách vì một công việc khác ta phải ngừng lại và ta có làm dấu tại trang đang đọc, sau khi làm xong công việc thì ta tiếp tục quay lại để đọc tiếp cuốn sách tại trang đang đọc. Tất cả đều xảy ra như vậy thì rất là bình thường nhưng trong khi ta thực hiện công việc thì có một người khác xem cuốn sách của ta và vô tình làm mất dấu, khi ta quay lại ta sẽ đọc không đúng trang đang dừng lại. Nguyên tắc làm việc của vi xử lý hoàn toàn giống như vậy.

Trong “vi điều khiển” thì bộ nhớ ngăn xếp là bộ nhớ RAM nội bộ nên chúng sẵn sàng hoạt động cho việc lưu trữ tạm, còn đối với “vi xử lý” thì bộ nhớ ngăn xếp sử dụng bộ nhớ ngoài nên bạn phải khởi tạo bộ nhớ ngăn xếp phải là vùng nhớ RAM để có thể ghi và đọc lại được, nếu bạn khởi tạo tại vùng nhớ EPROM hoặc khởi tạo tại nơi mà bộ nhớ không ghi vào được thì sẽ làm mất địa chỉ – dữ liệu lưu vào bộ nhớ ngăn xếp dẫn đến chương trình sẽ thực hiện sai.

Một điều cần phải chú ý nữa là trong lập trình chúng ta không được nhảy từ chương trình con sang chương trình chính để thực hiện tiếp chương trình vì làm như vậy sau nhiều lần thực hiện thì bộ nhớ ngăn xếp sẽ bị tràn và ghi đè lên các dữ liệu khác làm sai chương trình. Trong trường hợp này chúng ta sẽ thấy rằng chương trình chúng ta thực hiện đúng một vài lần và sau đó thì sai.

#### Các vector ngắt (Interrupt Vectors) :

Như đã trình bày ở trên, khi có một yêu cầu ngắt xảy ra thì sau khi cất giá trị địa chỉ trong PC vào ngăn xếp thì địa chỉ của chương trình con phục vụ ngắt tương ứng còn gọi là vector địa chỉ ngắt sẽ được nạp vào thanh ghi PC, địa chỉ này là cố định và do nhà chế tạo vi điều khiển qui định. Các chương trình ngắt phải bắt đầu viết đúng tại địa chỉ quy định đó. Các vector địa chỉ ngắt được cho trong bảng 4-17:

Interrupt	Flag	Vectors Address
System Reset	RST	0000H
External0	IE0	0003H
Timer0	TF0	000BH
External1	IE1	0013H

Timer1	TF1	001BH
Serial Port	RI or TI	0023H
Timer2	TF2 or EXF2	002BH

**Bảng 4-17. Tóm tắt vector địa chỉ ngắn.**

Vector reset hệ thống bắt đầu tại địa chỉ 0000H: khi reset vi điều khiển thì thanh ghi PC = 0000H và chương trình chính luôn bắt đầu tại địa chỉ này.

Khi bạn sử dụng yêu cầu ngắn nào thì chương trình con phục vụ ngắn phải viết đúng tại địa chỉ tương ứng. Ví dụ bạn sử dụng ngắn timer T0 thì chương trình ngắn bạn phải viết tại địa chỉ 000BH.

Do khoảng vùng nhớ giữa các vector địa chỉ của các nguồn ngắn chỉ có vài ô nhớ ví dụ như vector địa chỉ ngắn của ngắn INT0 tại 0003H và vector địa chỉ ngắn của ngắn T0 tại 000BH chỉ cách nhau có 9 ô nhớ. Nếu chương trình phục vụ ngắn của ngắn INT0 có kích thước lớn hơn 9 byte thì nó sẽ đụng đến vùng nhớ của ngắn T0. Cách giải quyết tốt nhất là ngay tại địa chỉ 0003H ta viết lệnh nhảy đến một vùng nhớ rộng hơn. Còn nếu các ngắn T0 và các ngắn khác không sử dụng thì ta có thể viết chương trình tại đó cũng được.

Chương trình chính luôn bắt đầu tại địa chỉ 0000H sau khi reset hệ thống, nếu trong chương trình có sử dụng ngắn thì ta phải dùng lệnh nhảy tại địa chỉ 0000H để nhảy đến một vùng nhớ khác rộng hơn không bị giới hạn để viết tiếp.

## VIII. BÀI TẬP:

**Câu 1:** So sánh sự khác nhau của vi xử lý và vi điều khiển 8 bit.

**Câu 2:** Sự khác nhau của vi điều khiển 89C51 và 89C52.

**Câu 3:** Cho biết ứng dụng của vi xử lý và vi điều khiển.

**Câu 4:** Cho biết sự khác nhau giữa các vi điều khiển 80C31, 80C51, 87C51 và 89C51.

**Câu 5:** Hãy tóm tắt phần cứng của vi điều khiển 80C31.

**Câu 8:** Hãy tóm tắt phần cứng của vi điều khiển 89C51, 89C52, 89S52, 89S8252

**Câu 12:** So sánh sự khác nhau của vi điều khiển 89C52 và 89S52.

**Câu 13:** Hãy nêu trình tự nạp bộ nhớ Flash của vi điều khiển 89C51.

**Câu 15:** Hãy nêu các thành phần bên trong sơ đồ khối của vi điều khiển 89C51.

**Câu 16:** Hãy cho biết chức năng các port của vi điều khiển 89C51.

**Câu 17:** Hãy cho biết chức năng của các tín hiệu điều khiển ALE, EA, PSEN, reset, Xtal1, Xtal2 của vi điều khiển 89C51.

**Câu 18:** Bộ nhớ RAM nội của vi điều khiển 89C51 được chia ra làm mấy loại bộ nhớ, hãy liệt kê tên các loại và địa chỉ của từng loại.

**Câu 19:** Vùng nhớ bank thanh ghi có bao nhiêu bank, địa chỉ của từng bank, chức năng của vùng nhớ này, mặc nhiên bank nào được gán cho nhóm thanh ghi R, làm thế nào để thay đổi cách gán nhóm thanh ghi R cho các bank còn lại.

**Câu 20:** Vùng nhớ RAM truy xuất từng bit có bao nhiêu BYTE, địa chỉ tính theo byte, có bao nhiêu bit, địa chỉ của tất cả các bit, chức năng của vùng nhớ này, có mấy cách truy xuất vùng nhớ này.

**Câu 21:** Hãy cho biết địa chỉ của các port trong bộ nhớ RAM và cách truy xuất địa chỉ bit của các port.

**Câu 22:** Trong vi điều khiển 89C51, hãy cho biết địa chỉ của thanh ghi SP, giá trị khởi tạo sau khi reset và địa chỉ của vùng nhớ ngăn xếp, dung lượng bộ nhớ ngăn xếp tối đa bằng bao nhiêu, chức năng của bộ nhớ ngăn xếp.

**Câu 23:** Hãy cho biết đặc điểm và chức năng của thanh ghi DPTR trong vi điều khiển 89C51.

**Câu 24:** Hãy cho biết chức năng của thanh ghi PSW trong vi điều khiển 89C51.

**Câu 25:** Hãy thiết kế một hệ thống đếm sản phẩm hiển thị kết quả đếm trên 3 led 7 đoạn dùng vi điều khiển 89C51 hoặc 89C52 hoặc 89S52.

**Câu 26:** Hãy thiết kế một hệ thống đếm đồng hồ số hiển thị giờ phút giây trên 6 led 7 đoạn dùng vi điều khiển 89C51 hoặc 89C52 hoặc 89S52.

**Câu 27:** Hãy thiết kế một hệ thống đo nhiệt độ dùng cảm biến LM35 và ADC 0808 hiển thị kết quả trên 3 led 7 đoạn dùng vi điều khiển 89C51 hoặc 89C52 hoặc 89S52.

**Câu 28:** Hãy thiết kế một hệ thống đo nhiệt độ dùng cảm biến LM35 và ADC 0808 hiển thị kết quả trên LCD dùng vi điều khiển 89C51 hoặc 89C52 hoặc 89S52.

[return](#)

**the end**

## *Chương 5*

# LẬP TRÌNH HỢP NGỮ

## CHO VI ĐIỀU KHIỂN MCS51

### GIỚI THIỆU

### HOẠT ĐỘNG CỦA TRÌNH BIÊN DỊCH ASSEMBLER

### CẤU TRÚC CỦA CHƯƠNG TRÌNH HỢP NGỮ

### TÍNH TOÁN BIỂU THỨC TRONG QUÁ TRÌNH DỊCH

1. CÁC BIỂU THỨC VÀ TOÁN TỬ
2. CHUỖI KÝ TỰ (CHARACTER STRING)
3. BỘ ĐẾM VỊ TRÍ (LOCATION COUNTER)
4. CÁC TOÁN TỬ SỐ HỌC (ARITHMETIC OPERATION)
5. CÁC TOÁN TỬ LOGIC
6. CÁC TOÁN TỬ QUAN HỆ (RELATION OPERATORS)
7. CÁC TOÁN TỬ KHÁC
8. THỨ TỰ UU TIÊN CỦA CÁC TOÁN TỬ

### CÁC CHỈ DẪN CHO ASSEMBLER

1. CÁC CHỈ DẪN ĐIỀU KHIỂN TRẠNG THÁI ASSEMBLER
2. CHỈ DẪN ĐỊNH NGHĨA KÍ HIỆU
3. CHỈ DẪN KHỞI TẠO TRỊ TRONG BỘ NHỚ
4. CHỈ DẪN DÀNH CHỖ TRONG BỘ NHỚ
5. CÁC CHỈ DẪN LIÊN KẾT CHƯƠNG TRÌNH
6. CÁC CHỈ DẪN CHỌN SEGMENT

### CÁC ĐIỀU KHIỂN CỦA TRÌNH BIÊN DỊCH ASSEMBLER

### HOẠT ĐỘNG LIÊN KẾT - LINKER

### SỬ DỤNG MACRO

1. TRUYỀN THAM SỐ CHO MACRO
2. MACRO VỚI NHÃN CỤC BỘ
3. TÁC VỤ REPEAT (LẶP LẠI)
4. CÁC TÁC VỤ ĐIỀU KHIỂN

### CÁCH VIẾT CHƯƠNG TRÌNH HỢP NGỮ

1. GIỚI THIỆU
2. PHÂN TÍCH
  - a. Phát biểu statement
  - b. Cấu trúc lặp while/do
  - c. Cấu trúc vòng lặp repeat/until
  - d. Cấu trúc lựa chọn If/then/else

## *LIỆT KÊ CÁC HÌNH HÌNH*

Hình 5-1. Biên dịch một chương trình nguồn.

Hình 5-2. Hoạt động của chương trình linker có tên là RL51.

## *LIỆT KÊ CÁC BẢNG*

Bảng 5-1. Các kiểu toán hạng

Bảng 5-2. Thứ tự ưu tiên các toán tử

Bảng 5-3. Các từ khoá điều khiển khi biên dịch.

## I. GIỚI THIỆU:

Hợp ngữ (assembly language) thay thế những mã nhị phân bằng các từ gợi nhớ để lập trình dễ dàng hơn. Máy tính không hiểu hợp ngữ do đó trình biên dịch hợp ngữ Assembler và trình liên kết Linker có chức năng dịch những chương trình viết bằng hợp ngữ thành ngôn ngữ máy.

Một số khái niệm:

### **Chương trình hợp ngữ (Assembly Language Program)**

Là chương trình được viết bằng cách dùng các nhãn, các từ gợi nhớ,..., trong đó mỗi phát biểu tương ứng với một lệnh của ngôn ngữ máy. Chương trình viết bằng hợp ngữ gọi là mã nguồn và chương trình này không thể thực thi mà nhằm giúp người lập trình đọc hiểu những gì vi xử lý thực hiện và gỡ rối một cách dễ dàng.

### **Chương trình ngôn ngữ máy (Machine Language Program)**

Là chương trình gồm các mã nhị phân tương ứng với 1 lệnh của vi xử lý. Các chương trình viết bằng ngôn ngữ máy thường được gọi là mã đối tượng (object code) và thực thi được.

### **Chương trình Assembler:**

Là chương trình dịch một chương trình viết bằng hợp ngữ sang chương trình ngôn ngữ máy. Chương trình ngôn ngữ máy có thể ở dạng tuyệt đối hoặc ở dạng tái định vị.

### **Chương trình Linker:**

Là chương trình kết hợp các chương trình đối tượng tái định vị được để tạo ra chương trình đối tượng tuyệt đối để thực thi được.

### **Segment:**

Là một đơn vị bộ nhớ chứa mã lệnh hoặc chứa dữ liệu. Một segment có thể ở dạng tuyệt đối hoặc tái định vị được.

Segment tái định vị được sẽ có tên, kiểu và các thuộc tính cho phép chương trình linker kết hợp nó với các phần của các đoạn khác nếu cần để định vị đúng đoạn. Segment ở dạng tuyệt đối không có tên và không thể kết hợp được với các đoạn khác.

### **Module:**

Chứa 1 hay nhiều segment hoặc một phần segment. Một module có tên do người sử dụng đặt. Những định nghĩa module xác định tầm của các ký hiệu cục bộ. Một tập tin đối tượng chứa 1 hay nhiều module. Một module được xem như là một tập tin trong nhiều tình huống.

### **Chương trình:**

Gồm nhiều module tuyệt đối, trộn tất cả các đoạn tuyệt đối và tái định vị được từ tất cả các module nhập. Một chương trình chỉ chứa các mã nhị phân cho các chỉ thị mà máy tính hiểu.

## II. HOẠT ĐỘNG CỦA TRÌNH BIÊN DỊCH ASSEMBLER:

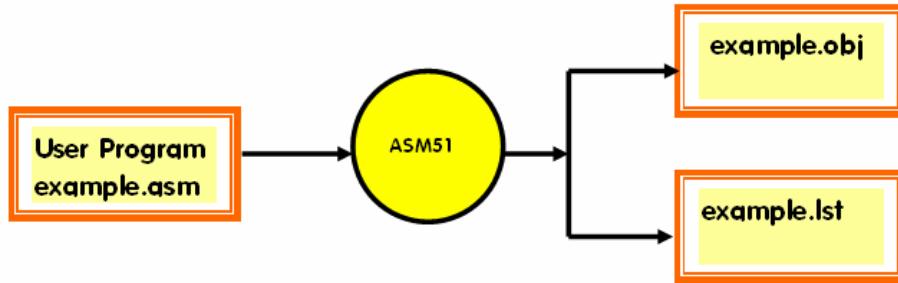
ASM51 là assembler chạy trên máy tính do Intel cung cấp để biên dịch cho họ MCS51.

Cách dùng ASM51 để biên dịch chương trình viết bằng hợp ngữ như sau: từ dấu nháy ở DOS hay ở Win commander ta thực hiện theo cú pháp:

**ASM51 source\_file[assembler\_control]**

Trong đó source\_file là tên tập tin nguồn viết bằng hợp ngữ, assembler\_control là những điều khiển assembler.

Assembler nhận tập tin nguồn (ví dụ “example.asm”) sẽ tạo ra tập tin đối tượng (“example.Obj”) và tập tin kiểu liệt kê (“example.lst”) như hình 5-1:



**Hình 5-1. Biên dịch một chương trình nguồn.**

Tất cả các chương trình biên dịch đều quét chương trình nguồn 2 lần để thực hiện dịch ra ngôn ngữ máy nên được gọi là Assembler hai bước. Assembler sử dụng bộ đếm vị trí làm địa chỉ của các lệnh và các giá trị cho các nhãn. Hoạt động của từng bước được mô tả như sau:

**Bước 1:** nhận diện các nhãn và các kí hiệu trong chương trình nguồn, tính toán các địa chỉ tương đối của chúng và cất vào bảng kí hiệu.

Bảng kí hiệu chứa những vị trí nhãn, những kí hiệu và những giá trị tương ứng của chúng.

Bộ đếm vị trí: lưu giữ địa chỉ của các lệnh và giá trị của nhãn chương trình.

**Bước 2:** tạo ra tập tin đối tượng và tập tin liệt kê:

- Các từ gợi nhớ
- Các toán hạng được định vị và đặt sau các mã lệnh.
- Các giá trị kí hiệu được truy cập để tính đúng dữ liệu hoặc địa chỉ.
- Cho phép tham chiếu tới.

Tập tin tái định vị chứa thông tin cần cho linker và định vị.

Tập tin liệt kê chứa chương trình nguồn và mã lệnh.

### III. CẤU TRÚC CỦA CHƯƠNG TRÌNH HỢP NGỮ:

- Những lệnh của vi xử lý.
- Những chỉ dẫn assembler (Assembler Directive).
- Những điều khiển Assembler.
- Các chú thích.

Cú pháp lệnh của vi xử lý như sau:

**[label:] mnemonic [operand] [,operand] [...] [/comment]**

Trong đó label là nhãn – theo sau bởi dấu hai chấm “:”, mnemonic là từ gợi nhớ của lệnh, operand là toán hạng tuỳ thuộc vào lệnh có một hoặc nhiều toán hạng hoặc không có toán hạng, cuối cùng là chú thích cho lệnh đó – đi sau dấu chấm phẩy “;”.

**Kí hiệu** là tên được định nghĩa để biểu diễn một giá trị, khối văn bản, địa chỉ hoặc tên thanh ghi và cũng có thể biểu diễn các hằng số và các biểu thức.

Các tên của các kí hiệu cho phép tối đa 31 kí tự với kí tự đầu phải là chữ hoặc dấu “?” hoặc “-”, và sau đó là các chữ, số, “?” hoặc “-”. Các kí hiệu có thể sử dụng các kí tự in hoa hay thường không phân biệt. Chú ý các từ kí hiệu là các từ đã sử dụng nên người lập trình không được dùng chúng làm kí hiệu cho các mục đích khác.

#### Ví dụ1:

Bdn EQU R2

**Nhãn** là một loại kí hiệu dùng để định nghĩa vị trí trong chương trình:

- Tên nhãn tương ứng với một địa chỉ.
- Vùng văn bản thứ nhất trong dòng hợp ngữ
- Theo sau nhãn là dấu hai chấm “::”
- Trên một hàng chỉ có thể định nghĩa một nhãn.
- Không được đặt tên các nhãn trùng nhau.

#### Ví dụ2:

Label1: MOV R2,#35h

**Mnemonic** là tất cả các từ gợi nhớ cho tất cả các lệnh và các chỉ dẫn assembler:

- Mnemonic cho lệnh: ADD, SUB, MUL, DIV, MOV,...
- Mnemonic cho chỉ dẫn assembler: org, equ, db, bit,...

**Toán hạng operand** là đối số hoặc biểu thức được đặt tách riêng với lệnh hoặc chỉ dẫn assembler, toán hạng có thể là địa chỉ hoặc dữ liệu.

Các chỉ dẫn assembler luôn cần các toán hạng là hằng số hoặc kí hiệu.

#### Ví dụ3:

Biendemngat EQU R2

Số toán hạng trong lệnh tùy thuộc vào lệnh có toán hạng hay không có:

#### Ví dụ4:

MOV R0,#75h

NOP

RET

Trong hợp ngữ ASM51 có các kiểu toán hạng bảng 5-1:

Kiểu toán hạng	Mô tả
Dữ liệu tức thời	Kí hiệu hoặc hằng số được dùng làm giá trị số
Địa chỉ bit trực tiếp	Kí hiệu hoặc hằng số tham chiếu địa chỉ bit
Địa chỉ chương trình	Kí hiệu hoặc hằng số tham chiếu địa chỉ mã
Địa chỉ dữ liệu trực tiếp	Kí hiệu hoặc hằng số tham chiếu địa chỉ dữ liệu
Địa chỉ gián tiếp	Tham chiếu gián tiếp đến bộ nhớ, có thể là offset
Kí hiệu assembler đặc biệt	Tên thanh ghi.

**Bảng 5-1. Các kiểu toán hạng****Dữ liệu tức thời (immediate data):**

Là biểu thức số được mã hoá như một phần trong lệnh ngôn ngữ máy. Toán hạng này phải có kí hiệu “#” đi trước.

**Ví dụ 5:**

```
MOV R0,#75h
```

Trong ví dụ này 75h là dữ liệu tức thời.

**Địa chỉ bit trực tiếp: (direct bit address)::**

Kiểu này dùng để truy cập các bit của các ô nhớ cho phép truy xuất bit.

Có 3 cách để định địa chỉ bit:

- Truy xuất trực tiếp địa chỉ bit.
- Truy xuất toán tử chấm (byte.bit).
- Kí hiệu assembler được định nghĩa trước.

**Ví dụ 6:**

```
SETB 00h      ;bit có địa chỉ 00h
CLR  ACC.7    ;xoá bit thứ 7 của thanh ghi A
CLR  EA       ;xoá bit ngắn toàn cục
```

**Địa chỉ chương trình: (program address)::**

Là toán hạng của lệnh nhảy.

- Lệnh nhảy tương đối: trong kiểu lệnh này toán hạng này có độ dài 8 bit được xem là offset sử dụng cho lệnh nhảy không điều kiện sjmp và lệnh nhảy có điều kiện.
- Lệnh nhảy và lệnh gọi tuyệt đối: trong kiểu lệnh này toán hạng này có độ dài 11 bit dùng để quản lý trang bộ nhớ cho lệnh AJMP và ACALL.
- Lệnh nhảy và lệnh gọi có địa chỉ dài: trong kiểu lệnh này toán hạng này có độ dài 16 bit dùng để quản lý toàn bộ bộ nhớ cho lệnh LJMP và LCALL.

**Nhảy và gọi generic:**

Lệnh JMP có thể được dịch hợp thành lệnh SJMP, AJMP hoặc LJMP.

Lệnh Call có thể được dịch hợp thành lệnh ACALL hoặc LCALL

Người lập trình không cần quan tâm đến địa chỉ thật khi nhảy hay gọi. Quy tắc chuyển thành tuỳ thuộc vào assembler:

Lệnh SJMP: không có tham chiếu tới và địa chỉ đích trong vùng -128 byte so với địa chỉ của lệnh kế.

Lệnh AJMP/ACALL: không có tham chiếu tới và địa chỉ đích trong vùng nhớ cùng khối 2 KByte so với lệnh kế.

Lệnh AJMP/ACALL: có tham chiếu tới địa chỉ đích trong vùng nhớ 64Kbyte.

**Ví dụ 7:**

1234	1		org	1234h
	2			
1234	04	3	start:	INC A
1235	80FD	4		JMP start
		5		
12FC		6	org	start+200
12FC	4134	7	JMP	start
		8		
12FE	021304	9		JMP finish
1301	121307	10		CALL delay
		11		
1304	14	12	finish:	DEC A
1305	4134	13		JMP start
		14		
1307	7F00	15	delay:	MOV R7,#0
1309	22	16		RET
		17		end

Hàng lệnh thứ 4 jmp được biên dịch thành sjmp, hàng lệnh jmp thứ 7 được biên dịch thành Ajmp và hàng lệnh jmp thứ 9 được biên dịch thành ljmp.

#### Địa chỉ dữ liệu trực tiếp (direct data address):

Địa chỉ này dùng để truy xuất bộ nhớ dữ liệu nội từ có địa chỉ 00H đến 7FH và các vùng nhớ chứa các thanh ghi đặc biệt từ 80H đến FFH. Các kí hiệu được định nghĩa đều có thể sử dụng được cho các thanh ghi chức năng.

**Ví dụ8:** hai lệnh sau là tương đương:

```
MOV A,90H
MOV A,P1
```

#### Địa chỉ dữ liệu gián tiếp (indirect data address):

Kiểu này dùng các thanh ghi để chứa địa chỉ của các ô nhớ cần truy xuất dữ liệu. Các thanh ghi sử dụng cho kiểu này là thanh ghi R0, R1, DPTR và PC.

#### Các kí hiệu đặc biệt của assembler:

Các kí hiệu này dùng cho cách định địa chỉ dùng thanh ghi như A, DPTR, R0 đến R7, PC, cờ C và cặp thanh ghi AB.

Kí hiệu dấu “\$” dùng để tham chiếu đến giá trị hiện hành của bộ đếm vị trí.

**Ví dụ9:** hai lệnh sau là tương đương:

```
WAIT: JNB RI,WAIT
      JNB RI,$
```

Kí hiệu “;” đi sau nó là các chú thích

## IV. TÍNH TOÁN BIỂU THỨC TRONG QUÁ TRÌNH DỊCH:

### 1 CÁC BIỂU THỨC VÀ TOÁN TỬ:

Toán tử được dùng để kết hợp và so sánh các toán hạng trong chương trình hợp ngữ.

Biểu thức dùng để kết hợp các số, các chuỗi ký tự, các ký hiệu và các toán tử để tính toán ra số nhị phân 16 bit. Dùng biểu thức trong lập trình sẽ giúp cho chương trình dễ đọc hơn và uyển chuyển hơn.

Các toán hạng gồm có: số, ký tự, chuỗi ký tự và bộ đếm vị trí.

Các toán tử gồm có: toán tử số học, toán tử nhị phân, toán tử quan hệ và các toán tử khác.

**Số:** có thể được sử dụng là:

- Số thập lục phân (hexadecimal = hex, có cơ số 16): H, h.
- Số thập phân (decimal, có cơ số 10): D, d hoặc không cần ghi.
- Số bát phân (octal, có cơ số 8): O, o, Q, q.
- Số nhị phân (binary, có cơ số 2): B, b.

**Chú ý:** với số hex nếu kí tự số hex đầu tiên bên trái là chữ (từ A đến F) thì phải có thêm kí tự số 0 ở trước.

**Ví dụ10:** lệnh nạp dữ liệu F4H vào thanh ghi R0

```
MOV R0,#0F4H
```

**Ký tự:** cho phép tối đa 2 ký tự nằm giữa 2 dấu nháy (`) có thể được dùng làm toán hạng trong biểu thức.

**Ví dụ11:** ‘A’ có giá trị tương đương 0041H (bảng mã ASCII)

‘AB’ có giá trị tương đương 4142H

‘a’ có giá trị tương đương là 0061H

‘ab’ có giá trị tương đương 6162H

Chúng ta cũng có thể sử dụng ký tự làm toán hạng cho dữ liệu tức thời.

**Ví dụ12:**

```
MOV R0,#'0'
```

## 2. CHUỖI KÝ TỰ (CHARACTER STRING):

Có thể kết hợp với chỉ dẫn DB để định nghĩa các thông báo trong chương trình hợp ngữ.

**Ví dụ13:**

Keymsg	DB	‘press any key’
--------	----	-----------------

Chỉ dẫn trên sẽ tạo ra một vùng nhớ dữ liệu chứa các mã ASCII tương ứng là 50H (chữ P), 72H (chữ r), 65H (chữ e), ... , lưu vào vùng nhớ bắt đầu từ địa chỉ keymsg.

## 3. BỘ ĐẾM VỊ TRÍ (LOCATION COUNTER):

Dùng để xác định địa chỉ của từng mã lệnh trong chương trình biên dịch tùy thuộc vào chỉ dẫn ORG.

Ký tự ‘\$’ sẽ trả về giá trị hiện hành của bộ đếm vị trí.

**Ví dụ14:**

LOC	OBJ	LINE	SOURCE
-----	-----	------	--------

(vị trí	mã đối tượng	hàng		mã nguồn)	
1234		1		org	1234h
		2			
1234	04	3	start:	INC	A
1235	80FD	4		JMP	start
		5			
12FC		6		org	start+200
12FC	4134	7		JMP	start
		8			
12FE	021304	9		JMP	finish
1301	121307	10		CALL	delay
		11			
1304	14	12	finish:	DEC	A
1305	4134	13		JMP	start
		14			
1307	7F00	15	delay:	MOV	r7,#0
1309	22	16		RET	
		17			end

#### 4. CÁC TOÁN TỬ SỐ HỌC (ARITHMETIC OPERATION):

- Toán tử cộng “+” expr + expr
- Toán tử trừ “-” expr - expr
- Toán tử nhân “×” expr \* expr
- Toán tử chia “/” expr / expr
- Toán tử chia lấy phần dư “mod” expr MOD expr

Trong đó expr là biểu thức.

#### Ví dụ 15:

MOV A,#10 + 10H ;hai lệnh này tương đương

MOV A,#1AH

MOV A,#25 MOD 7 ;hai lệnh này tương đương

MOV A,#4

#### 5. CÁC TOÁN TỬ LOGIC:

- Toán tử NOT: NOT expr lấy bù đảo từng bit
- Toán tử SHR expr SHR n dịch sang phải n bit
- Toán tử SHL expr SHL n dịch sang trái n bit
- Toán tử AND expr AND expr and từng cặp bit tương ứng
- Toán tử OR expr OR expr or từng cặp bit tương ứng
- Toán tử XOR expr XOR expr xor từng cặp bit tương ứng

Trong đó expr là biểu thức và x là số vị trí cần dịch.

#### Ví dụ 16: 3 lệnh sau là tương đương

```

THREE    EQU      3
MINUS3   EQU     -3

MOV      A,#(NOT THREE) +1
MOV      A,#MINUS3
MOV      A,#11111101B

```

### 6. CÁC TOÁN TỬ QUAN HỆ (RELATION OPERATORS):

- EQ = bằng nhau (equal)
- NE <> không bằng nhau (not equal)
- LT < nhỏ hơn (less than)
- LE <= nhỏ hơn hoặc bằng (less than or equal)
- GT > lớn hơn (greater than)
- GE >= lớn hơn hay bằng (greater than or equal)

Kết quả luôn trả về đúng (FFFFH) hoặc sai (0000H).

**Ví dụ 17:** các lệnh sau là tương đương

```

MOV      A,#5=5
MOV      A,#5 EQ 5
MOV      A,#5 NE 4
MOV      A,#5 <> 4
MOV      A,#0FFH

```

### 7. CÁC TOÁN TỬ KHÁC:

- Toán tử LOW expr có chức năng lấy kết quả byte thấp của expr.
- Toán tử HIGH expr có chức năng lấy kết quả byte cao của expr.

**Ví dụ 18:**

```

MOV      DPH,#HIGH(1234H)      ;hai lệnh này tương đương
MOV      DPH,#12H

MOV      DPL,#LOW(1234H)      ;hai lệnh này tương đương
MOV      DPL,#34H

```

### 8. THỨ TỰ UYU TIÊN CỦA CÁC TOÁN TỬ:

Danh sách quyền ưu tiên của các toán tử được sắp theo thứ tự từ cao nhất đến thấp nhất như bảng 5-2:

Thứ tự	Toán tử
1	()
2	HIGH LOW
3	* / MOD SHL SHR

<b>4</b>	<b>+ -</b>
<b>5</b>	<b>EQ NE LT LE GT GE = &lt;&gt; &lt; &lt;= &gt; &gt;=</b>
<b>6</b>	<b>NOT</b>
<b>7</b>	<b>AND</b>
<b>8</b>	<b>OR</b>
<b>9</b>	<b>XOR</b>

**Bảng 5-2. Thứ tự ưu tiên các toán tử**

Khi các toán tử được sử dụng có quyền ưu tiên ngang nhau thì việc tính toán ra giá trị sẽ bắt đầu tính từ trái sang phải.

## V. CÁC CHỈ DẪN CHO ASSEMBLER:

Là những chỉ thị lệnh cho assembler và được chia ra làm các nhóm như sau:

- Điều khiển trạng thái Assembler: ORG, END, USING.
- Định nghĩa ký hiệu: segment, equ, set, data, Idata, Xdata, bit, code.
- Khởi tạo hay định nghĩa trong bộ nhớ: DS, DBIT, DB, DW.
- Liên kết chương trình: public, extrn, name.
- Chọn đoạn: Rseg, Cseg, Dseg, Iseg, Bseg, Xseg.

### 1 CÁC CHỈ DẪN ĐIỀU KHIỂN TRANG THÁI ASSEMBLER:

**ORG:** có chức năng thay đổi bộ đếm vị trí của segment hiện thời để đặt gốc chương trình mới cho các phát biểu theo sau org.

Cách sử dụng: *Org expr*

#### Ví dụ 19:

2200	1	ORG	2200h	;khai báo địa chỉ bắt đầu 2200h
2200	7435	2	MOV A,#35H	
2202	7435	3	MOV A,#35H	
2204	7435	4	MOV A,#35H	
		5		
		6		
3000		7	ORG (\$+1000H) and 0F000h	;khai báo địa chỉ bằng địa chỉ hiện tại cộng
3000	7435	8	MOV A,#35H	;thêm 1000H và and với F000H để chuyển
		9	END	;sang 4 kbyte kế

Nếu bỏ lệnh AND với F000H thì địa chỉ mới sẽ là 3206H = 2206H + 1000H

Ta có thể sử dụng khai báo ORG trong bất kỳ loại segment nào. Nếu segment hiện thời là tuyệt đối thì giá trị sẽ là địa chỉ tuyệt đối trong segment hiện thời. Nếu segment hiện thời là tái định vị được thì giá trị của biểu thức ORG được xử lý như offset của địa chỉ nền của segment hiện thời.

**USING:** có chức năng báo cho assembler biết bank thanh ghi tích cực hiện thời, nhưng nó không chuyển băng thanh ghi, do đó để có thể sử dụng đúng thì ta phải sử dụng AR0 đến AR7 sau khai báo USING thay vì dùng R0 đến R7. Khi đó assembler sẽ tự động sử dụng đúng thanh ghi trong băng thanh ghi mong muốn đó và khi dịch assembler sẽ đổi ARN sang địa chỉ trực tiếp.

Cách sử dụng: **USING expr**

### Ví dụ 20:

**USING 2**

```
MOV AR3,#70H
MOV R0,#22H
```

Trong lệnh thứ nhất, AR3 chính là thanh ghi R3 của bank thanh ghi 2 và sẽ được thay thế bằng địa chỉ trực tiếp là 13H. Trong lệnh thứ hai, R0 vẫn truy cập trong bank thanh ghi hiện tại là bank 0.

**END:** là phát biểu cuối cùng trong tập tin nguồn, những gì sau chỉ dẫn end sẽ không được xử lý.

### 2 CHỈ DẪN ĐỊNH NGHĨA KÝ HIỆU

Những chỉ dẫn này tạo các ký hiệu để biểu diễn các segment, các thanh ghi, số và địa chỉ. Không được sử dụng nhãn cho chỉ dẫn. Những ký hiệu được định nghĩa bởi các chỉ dẫn này là duy nhất, ngoại trừ chỉ dẫn SET cho phép định nghĩa lại.

**EQU hay SET:** có chức năng gán 1 ký số hay ký hiệu thanh ghi cho tên ký hiệu được đặt tâ.

Cách sử dụng:	<b>symbol equ expr</b>
	<b>symbol set expr</b>

Trong đó **symbol** là ký hiệu do người dùng định nghĩa và **expr** là biểu thức.

### Ví dụ 21:

```
BDN EQU R2
GIAY SET 40
```

**Segment:** có chức năng khai báo segment tái định vị được.

Cách sử dụng: **symbol segment segment\_type**

Trong đó **symbol** là ký hiệu do người dùng định nghĩa và **segment\_type** là kiểu segment. Có các kiểu segment như sau:

- Code: segment mã chương trình.
- Xdata: segment vùng dữ liệu chứa ở bộ nhớ bên ngoài.
- Data: segment vùng dữ liệu nội có địa chỉ trực tiếp từ 00H÷7FH
- Idata: segment vùng dữ liệu nội có địa chỉ gián tiếp từ 00H÷7FH đối với 8051 và 00H÷FFH đối với 8052.
- Bit: segment vùng nhớ bit nằm trong vùng nhớ cho phép truy xuất bit từ 20H÷2FH

**Ví dụ22:****EPROM SEGMENT CODE**

Khai báo ký hiệu eprom là segment kiểu code. Chú ý phát biểu này chỉ khai báo EPROM là kiểu code, để sử dụng segment này thì phải sử dụng chỉ dẫn RSEG

**CODE/DATA/IDATA/XDATA/BIT**

Dùng để gán địa chỉ của kiểu tương ứng với ký hiệu, tuy nhiên nếu có sử dụng thì assembler kiểm tra kiểu.

**Ví dụ23:**

LOC	OBJ	LINE	SOURCE
0005		1	flag1 equ 05h
0005		2	flag2 bit 05h
0000	D205	3	SETB flag1
0002	D205	4	SETB flag2
0004	750500	5	MOV flag1,#0
0007	750500	6	MOV flag2,#0
*** ERROR #37, LINE #6 (0), DATA SEGMENT ADDRESS EXPECTED			
		7	end

**SYMBOL TABLE LISTING**

NAME	TYPE	VALUE	ATTRIBUTES
FLAG1...	NUMB	0005H	A
FLAG2...	B ADDR	0020H.5	A

REGISTER BANK(S) USED: 0  
ASSEMBLY COMPLETE, 1 ERROR FOUND (6)

Trong chương trình ví dụ trên ta đã khai báo flag1 là ô nhớ có địa chỉ 05H, flag2 là bit có địa chỉ 05H. Hai lệnh setb khi biên dịch không có lỗi vì assembler xem chúng là các bit có địa chỉ 05H.

Lệnh thứ 5 khi biên dịch sẽ xem flag1 là ô nhớ có địa chỉ 05H, nhưng lệnh thứ 5 thì khi biên dịch sẽ báo lỗi vì lệnh MOV không thể thực hiện đối với ô nhớ bit.

**3. CHỈ DẪN KHỞI TẠO TRONG BỘ NHỚ:**

**DB (define byte):** định nghĩa byte, có chức năng khởi tạo vùng nhớ mã với các giá trị kiểu byte.

Cách sử dụng: **[label:] db expr [,expr][...]**

Trong đó **label** là nhãn do người dùng định nghĩa và **expr** là biểu thức.

**Ví dụ24:**

LOC	OBJ	LINE	SOURCE
---		1	cseg at 0100h
0100	C0	2	ma7d: db 0c0h,0a4h
0101	A4	3	
0102	48656C6C	4	msg: db 'Hello'
0106	6F	5	
		6	end
SYMBOL TABLE LISTING			
-----			

**DW (define word):** định nghĩa từ, có chức năng khởi tạo vùng nhớ mã với các giá trị kiểu word.

Cách sử dụng: **[label:] dw expr [,expr][...]**

Trong đó **label** là nhãn do người dùng định nghĩa và **expr** là biểu thức.

#### 4. CHỈ DẪN DÀNH CHỖ TRONG BỘ NHỚ:

**DS (define storage):** định nghĩa vùng lưu trữ, có chức năng dành vùng nhớ theo byte. Chỉ dẫn này có thể được sử dụng trong bất kỳ loại segment nào ngoại trừ DBIT.

Cách sử dụng: **[label:] ds expr**

Trong đó **label** là nhãn do người dùng định nghĩa và **expr** là biểu thức không có tham chiếu tới.

Khi gặp chỉ dẫn DS trong chương trình thì bộ đếm vị trí của segment hiện tại được tăng thêm số byte là giá trị của expr.

**Ví dụ 25:** tạo vùng nhớ Ram nội 40 byte để lưu dữ liệu:

DSEG	AT	10H	;vùng nhớ dữ liệu nội
LEN	EQU	40	
BUF:	DS	LEN	;dành 40 byte bắt đầu từ địa chỉ 10H

**Ví dụ 26:** tạo vùng nhớ Ram ngoại 1000 byte để lưu dữ liệu:

XSEG	AT	2000H	;vùng nhớ dữ liệu ngoại
XLEN	EQU	1000	
XBUF:	DS	LEN	;dành 1000 byte bắt đầu từ địa chỉ 2000H

;bắt đầu mã chương trình

CSEG	AT	0000H	
	MOV	DPTR,#XBUF	;nạp địa chỉ của vùng nhớ ngoại vào dptr
LOOP:	CLR	A	
	MOVX	@DPTR,A	
	INC	DPTR	
	MOV	A,DPL	
	CJNE	A,#LOW(XBUF+XLEN+1),LOOP	;so sánh địa chỉ byte thấp

```

MOV    A,DPH
CJNE   A,#HIGH(XBUF+XLEN),LOOP      ;so sánh địa chỉ byte cao để kết thúc
SJMP   $
END

```

**.Dbit (define bit):** định nghĩa vùng lưu trữ dữ liệu bit, có chức năng dành vùng nhớ theo bit trong segment bit.

Cách sử dụng: **[label:] dbit expr**

Trong đó **label** là nhãn do người dùng định nghĩa và **expr** là biểu thức không có tham chiếu tới. Khi gặp chỉ dẫn DBIT trong chương trình thì bộ đếm vị trí của segment BIT hiện tại được tăng thêm với số bit là giá trị của **expr**.

### Ví dụ 27:

LOC	OBJ	LINE	SOURCE	
---	1	bseg		;chỉ dẫn vùng nhớ bit mặc nhiên bắt đầu từ 00H
0000	2	kbflag:	dbit 1	
0001	3	pbflag:	dbit 1	
0002	4	dbflag:	dbit 1	
	5			
---	6	bseg	at 30h	;chỉ dẫn vùng nhớ bit bắt đầu từ 30H
	7			
0030	8	F1:	DBIT 1	
0031	9	F2:	DBIT 2	
0033	10	F3:	DBIT 1	
	11			
---	12	cseg		
0000 D231	13	setb F2		
	14			
	15	end		

## SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
DBFLAG ..	B ADDR	0020H.2	A ;bit có địa chỉ bit là 02H của ô nhớ có địa chỉ 20H
F1 .....	B ADDR	0026H.0	A ;bit có địa chỉ bit là 30H của ô nhớ có địa chỉ 26H
F2 .....	B ADDR	0026H.1	A ;bit có địa chỉ bit là 31H của ô nhớ có địa chỉ 26H
F3 .....	B ADDR	0026H.3	A ;bit có địa chỉ bit là 32H của ô nhớ có địa chỉ 26H
KBFLAG ..	B ADDR	0020H.0	A ;bit có địa chỉ bit là 00H của ô nhớ có địa chỉ 20H
PBFLAG ..	B ADDR	0020H.1	A ;bit có địa chỉ bit là 01H của ô nhớ có địa chỉ 20H

## 5. CÁC CHỈ DẪN LIÊN KẾT CHƯƠNG TRÌNH

Cho phép các module (các tập tin) được hợp dịch riêng có thể liên lạc với nhau bằng cách cho phép tham chiếu giữa các module và đặt tên các module.

**Public:** Liệt kê các ký hiệu có thể được sử dụng trong các module đối tượng khác.

Cách sử dụng: **public symbol [,symbol][,..]**

Trong đó **ký hiệu symbol** được khai báo trong Public phải được định nghĩa trong module hiện hành.

**Ví dụ28:**

Public inchar, outchar, inline, outstr, extern

**Extrn:** Liệt kê các ký hiệu được tham chiếu trong module nguồn hiện hành nhưng chúng được khai báo trong các module khác.

Cách sử dụng: **extrn segment\_type(symbol [,symbol][,..])**

Các segment\_type là CODE, XDATA, DATA, IDATA, BIT và NUMBER (NUMBER là ký hiệu không có kiểu được định nghĩa bằng EQU).

**Ví dụ29:** Có 2 tập tin MAIN.SRC và message.SRC

```
;main.src
Extrn code (HELLO, GOOD_BYE)
...
CALL HELLO
...
CALL GOOD_BYE
...
END

;MESSAGE.SRC
PUBLIC HELLO, GOOD_BYE
...
HELLO;
...
RET
GOOD_BYE;
...
RET
```

Hai module trên không phải là chương trình đầy đủ: chúng được biên dịch riêng và liên kết với nhau để tạo chương trình khả thi. Trong khi liên kết, các tham chiếu ngoài được thay thế với địa chỉ đúng cho các lệnh CALL.

**Name:** dùng để đặt tên của module đối tượng được sinh ra trong chương trình hiện hành.

Cách sử dụng: **Name module\_name**

Trong đó *module\_name* là tên module.

#### 6. CÁC CHỈ DẪN CHỌN SEGMENT:

Segment là khối chứa mã lệnh hay vùng nhớ chứa dữ liệu mà assembler tạo ra từ mã hay dữ liệu trong tập tin nguồn hợp ngữ 8051. Khi Assembler gấp chỉ dẫn chọn segment thì nó sẽ chuyển hướng mã hoặc dữ liệu theo sau vào segment được chọn cho đến khi gấp chỉ dẫn chọn segment khác.

**.RSEG (relocatable segment – segment tái định vị được):** cho phép chọn segment tái định vị được mà đã định nghĩa trước bằng segment.

Cách sử dụng: **Rseg segment\_name**

Trong đó *segment\_name* là tên segment đã định nghĩa trước đó.

Chỉ dẫn này chuyển hướng mã và dữ liệu theo sau vào đoạn *segment\_name* cho đến khi gấp chỉ dẫn chọn segment khác.

Các kiểu chỉ dẫn chọn segment **CSEG/ DSEG/ ISEG/ XSEG** Cho phép chọn 1 segment tuyệt đối.

Cách sử dụng: **aSEG [at address]**

Trong đó *a* có thể là C, D, I, B hoặc X và *address* là địa chỉ.

#### Ví dụ 30:

LOC	OBJ	LINE	SOURCE	
		1	onchip	SEGMENT
		2	EPROM	SEGMENT
		3		
----		4	bseg	at 70h ;segment bit tuyệt đối
		5		
0070		6	flag1:	dbit 1
0071		7	flag2:	dbit 2
		8		
----		9	Rseg	onchip ;bắt đầu segment dữ liệu tái định vị được
0000		10	total:	ds 1
0001		11	count:	ds 1
0002		12	sum16:	ds 2
		13		
----		14	rseg	eprom ;bắt đầu segment mã tái định vị được
0000 750000 F		15	begin:	mov total,#0
		16		
		17	end	

#### VI. CÁC ĐIỀU KHIỂN CỦA TRÌNH BIÊN DỊCH ASSEMBLER:

Ta có thể đặt các dòng điều khiển trong chương trình nguồn để điều khiển những công việc như kết xuất (listing). Mỗi dòng điều khiển bắt đầu bằng ký tự chữ “\$” và sau nó là danh sách các từ khoá điều khiển cách nhau bởi khoảng trống.

Phần lớn các từ khoá điều khiển này có dạng tích cực và không tích cực và thường từ khoá điều khiển viết tắt bằng 2 ký tự.

Một số từ khoá điều khiển thông dụng được liệt kê ở bảng sau:

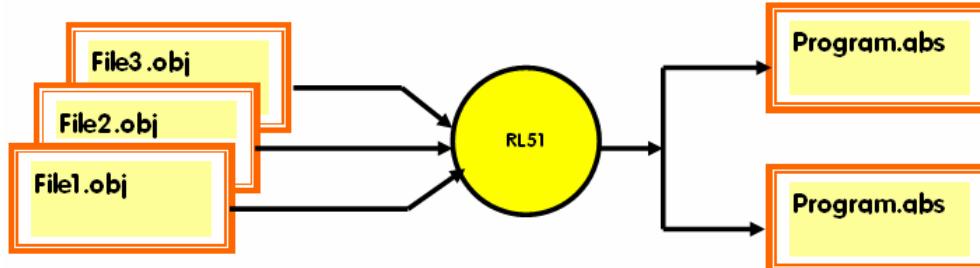
Tên điều khiển	Viết tắt	Mặc nhiên	Ý nghĩa
PAGELENGTH(n)	PL	PL(60)	Đặt số dòng tối đa cho mỗi trang của tập tin kết xuất (tầm vị trí từ 10 đến 65536)
PAGEWIDTH(n)	PW	PW(120)	Đặt số ký tự tối đa cho mỗi dòng (tầm trị từ 72 đến 132)
XREF/NOXREF	XR/NOXR	NOXR	Tạo bảng liệt kê tham chiếu chéo của tất cả các ký hiệu sử dụng trong chương trình.
EJECT	EJ		Tiếp tục kết xuất ở trang kế.
GEN/NOGEN	GE/NOGE	NOGE	Tạo ra đầy đủ các khai triển macro.
INCLUDE (filename)	IC		Chỉ tập tin có tên <i>filename</i> xem như là một phần của chương trình.
LIST/NOLIST	LI/NOLI	LI	In những dòng mã nguồn liên tục trong tập tin kết xuất.
SAVE/RESTORE	SA/RS		Lưu trữ những cài đặt điều khiển cho LIST và GEN, và lấy lại từ ngăn xếp.
TITLE(string)	TT		Đặt chuỗi ký tự string ở đầu trang tối đa 60 ký tự.

Bảng 5-3. Các từ khoá điều khiển khi biên dịch.

## VII. HOẠT ĐỘNG LIÊN KẾT – LINKER:

Với những ứng dụng lớn người lập trình thường chia chương trình thành nhiều chương trình con hay các module và có thể tái định vị được.

Ta cần chương trình liên kết và định vị để kết hợp các module thành một module đối tượng tuyệt đối mà có thể thực thi được. Tất cả các ký hiệu ngoài được thay thế bằng các giá trị đúng và được đặt vào trong các tập tin xuất được minh họa như hình 5-2:



Hình 5-2. Hoạt động của chương trình linker có tên là RL51.

### VIII. SỬ DỤNG MACRO:

Phương tiện xử lý macro của ASM51 là phương tiện thay thế chuỗi ký tự. Macro cho phép các phần mã sử dụng thường xuyên sẽ được định nghĩa một lần bằng cách dùng từ gọi nhớ đơn giản và có thể sử dụng bất kỳ chỗ nào trong chương trình bằng cách chèn vào từ gọi nhớ đó.

Ta có thể định nghĩa macro ở bất kỳ chỗ nào trong chương trình nguồn và sau đó sử dụng như các lệnh khác. Cú pháp của định nghĩa macro như sau:

```
%*define (call_pattern) (macro_body)
```

Trong đó *call\_pattern* là từ gọi nhớ do người dùng định nghĩa và *macro\_body* là thân macro chính là đoạn chương trình thường lặp lại.

Để phân biệt với các lệnh thật thì người ta đặt thêm ký hiệu “%” trước tên macro và khi hợp dịch thì tất cả các lệnh trong thân macro được thay thế vào nơi gọi chương trình macro.

**Ví dụ 31:** Nếu định nghĩa macro sau ở đầu tập tin nguồn

```
%*define (push_dptr)
(
    PUSH  DPH
    PUSH  DPL
)
```

Thì khi gặp phát biểu %push\_dptr trong chương trình nguồn thì trình biên dịch sẽ thay thế bằng 2 lệnh trên

```
PUSH  DPH
PUSH  DPL
```

trong tập tin .lst

Các tiện lợi khi sử dụng macro:

- Chương trình nguồn có sử dụng macro thì dễ đọc hơn vì từ gọi nhớ của macro cho biết ý nghĩa của công việc phải thực hiện.
- Chương trình ngắn gọn hơn nên ít đánh máy hơn.
- Sử dụng macro sẽ làm giảm bớt lỗi.
- Sử dụng macro giúp cho người lập trình không phải bận rộn với những chi tiết cấp thấp.

#### 1 TRUYỀN THAM SỐ CHO MACRO:

Macro với các tham số được truyền từ chương trình chính có dạng như sau:

```
%*define (macro_name (parameter_list)) (macro_body)
```

Trong đó *macro\_name* là tên macro, *parameter\_list* là danh sách các tham số và *macro\_body* là thân macro.

**Ví dụ 32:** Định nghĩa macro có truyền tham số như sau

```
%*define (cmpa#(value))
(
    CJNE A,#%value,$+3
)
```

Thì khi gọi phải truyền tham số như sau:

```
%Cmpa#(20H)
```

Khi biên dịch sẽ trở thành

```
CJNE A,#20H,$+3
```

**Chú ý:** lệnh cjne là lệnh 3 byte, do đó \$+3 chính là địa chỉ của lệnh kế nằm sau lệnh cjne.

**Ví dụ33:** Để thực hiện lệnh nhảy nếu lớn hơn thì ta có thể tạo macro như sau:

```
%*define (JGT(value,label))
(
    CJNE a,%value+1,$+3      ;so sánh với giá trị (value+1) rồi nhảy tới $+3
    JNC %label               ;nhảy tới label nếu cờ C = 0 có nghĩa là lớn hơn
)
```

## 2 MACRO VỚI NHÃN CỤC BỘ

Ta cũng có thể sử dụng các nhãn cục bộ trong macro có dạng như sau:

```
%*define (macro_name [(parameter_list)])
[local_list_of_labels](macro_body)
```

Trong đó *macro\_name* là tên macro, *parameter\_list* là danh sách các tham số, *list\_of\_labels* là danh sách các nhãn cục bộ và *macro\_body* là thân macro.

**Ví dụ34:** Định nghĩa macro có nhãn cục bộ như sau

```
%*define (dec_dptr) local skip
(
    DEC DPL
    MOV A,DPL
    CJNE A,#0FFh,%skip
    DEC DPH
    %Skip:)
```

Khi macro được gọi

```
%dec_dptr
```

Thì trình biên dịch sẽ thay thế lệnh gọi trên bằng các lệnh đã định nghĩa trong macro ở file .lst như sau:

```
DEC DPL
MOV A,DPL
CJNE A,#0FFh,skip00
DEC DPH
```

Skip00:

Nhãn cục bộ không quan hệ với nhãn có cùng tên trong chương trình chính vì trình biên dịch ASM51 đã tự động thêm vào mã số đi theo sau nhãn cục bộ khi biên dịch.

Nhưng nếu chúng ta định nghĩa macro như sau thì khi biên dịch ASM51 sẽ không đổi tên nhãn cục bộ:

```
%*define (dec_dptr) local skip
(
DEC DPL
MOV A,DPL
CJNE A,#0FFH,SKIP
DEC DPH
Skip:)
```

### 3. TÁC VỤ REPEAT (LẮP LẠI):

Là một trong các macro được xây dựng sẵn trong Assembler. Cú pháp:

```
%repeat (expression) (text)
```

Trong đó *expression* là biểu thức và *text* là văn bản cần lặp lại.

**Ví dụ 35:** Để thực hiện 100 lệnh NOP thì ta có thể sử dụng macro repeat như sau:

```
%repeat(100)
(
Nop
)
```

Khi biên dịch thì trong file .lst sẽ thay hàng lệnh trên bằng 100 lệnh NOP.

### 4. CÁC TÁC VỤ ĐIỀU KHIỂN:

ASM51 cung cấp các định nghĩa macro luồng điều khiển để cho phép hợp dịch có điều kiện các phần mã. Dạng lệnh macro như sau:

Cú pháp:

```
%IF (expression) THEN (balanced_text)
[ELSE (balanced_text)] FI
```

Trong đó *expression* là biểu thức và *balanced\_text* là văn bản cần thay đổi theo điều kiện.

**Ví dụ 36:**

```
internal    equ   1
; 1= 8051 serial I/O drivers
; 0= 8052 serial I/O drivers

%IF (internal) THEN
(inchar:           ;8051 drivers
...               ;các lệnh nhận ký tự hay dữ liệu
...
Outchar:          ... ;các lệnh truyền ký tự hay dữ liệu
...
)ELSE
(inchar:           ;8052 drivers
...               ;các lệnh nhận ký tự hay dữ liệu
```

Outchar: ... ;các lệnh truyền ký tự hay dữ liệu  
...  
)  
Nếu biến INTERNAL = 1 thì thực hiện việc truyền dữ liệu theo kiểu của 8051 nhưng nếu INTERNAL = 0 thì truyền dữ liệu theo kiểu của 8052.

## IX. CÁCH VIẾT CHƯƠNG TRÌNH HỢP NGỮ:

### 1 GIỚI THIỆU:

- Đặc điểm của chương trình là: giảm bớt mức độ phức tạp, công việc hay tác vụ xử lý rõ ràng tiện lợi cho việc gỡ rối và chỉnh sửa.
- Các kỹ thuật để phát triển chương trình: lập trình có cấu trúc, một chương trình có cấu trúc chứa một hệ phân cấp các chương trình con.
- Chỉ có 3 cấu trúc trong chương trình: các phát biểu, các vòng lặp và các lựa chọn.
- Đặc tính của cấu trúc: mỗi cấu trúc có một điểm vào duy nhất và một điểm ra duy nhất.
- iASM51 cung cấp các định nghĩa macro luồng điều khiển để cho phép hợp dịch có điều

### 2 PHÂN TÍCH:

#### a. Phát biểu (statement):

**Ví dụ 37:** Gán giá trị cho 1 biến bằng *pseudo-code* (mã giả) là  
[count=0]

**Ví dụ 38:** Gọi 1 chương trình con bằng *pseudo-code*  
Print\_string("select option")

**Ví dụ 39:** Khối phát biểu bằng *pseudo-code*  
Begin  
    [statement1]  
    [statement2]  
    [statement3]  
end

#### b. Cấu trúc lặp while/do:

**Ví dụ 40:** Minh họa một cấu trúc while/do khi cờ C = 1 thì thực thi phát biểu:

*pseudo-code:*

while [c==1] do [statement]

mã 8051:

enter:	JNC skip
	(phát biểu) [statement]
skip:	(tiếp tục)

**Ví dụ 41:** Yêu cầu bài toán: Chương trình con tính tổng SUM của một chuỗi số: tổng số dữ liệu dạng byte lưu trong thanh ghi R7 và địa chỉ bắt đầu của vùng dữ liệu lưu trong thanh ghi R0, kết quả tổng lưu trong thanh ghi A.

Với *pseudo-code*:

```
[summ =0]
while [chieu_dai>0] do
begin
    [sum= sum+@pointer]
    [pointer=pointer+1]      ;tăng pointer
    [chieu_dai=chieu_dai - 1] ;giảm chiều dài
end
```

Với mã lệnh 8051 (viết theo *pseudo-code*)

```
Sum:     CLR   A
Loop:    CJNE  R7,#0, continues
          JMP   exit
continues: ADD   A,@R0
            INC   R0
            DEC   R7
            JMP   Loop
exit:    RET
```

Với mã lệnh 8051 (*viết gọn hơn*)

```
Sum:     CLR   A
continues: ADD   A,@R0
            INC   R0
            DJNZ  R7,continues
            RET
```

Trong kiểu thứ 3 này chương trình được viết gọn hơn như phải thỏa điều kiện là  $R7 > 0$ .

**Ví dụ 42:** Yêu cầu bài toán: sử dụng cấu trúc while/do để thực thi “phát biểu” khi thanh ghi A khác 0Dh và thanh ghi R7 khác 0.

*Pseudo-code:*

While [acc != 0DH and R7!= 0] do [statements]

Với mã lệnh 8051 (viết theo *pseudo-code*)

```
enter:   CJNE  A,#0DH, skip
          JMP   exit
skip:    CJNE  R7,#0,statement
          JMP   exit
statement:
...
...
```

```
JMP    enter
Exit:
```

### c. Cấu trúc vòng lặp repeat/until:

**Ví dụ 43:** Yêu cầu bài toán: viết chương trình con tìm kiếm mã ký tự Z trong một chuỗi ký tự (kết thúc bằng ký tự rỗng) có địa chỉ bắt đầu lưu thanh ghi R0. Thanh ghi A được trả về với ký tự ASCII “Z” nếu nó có trong chuỗi ký tự, ngược lại thì thanh ghi A chứa giá trị 0.

*pseudo-code:*

```
repeat
    [Acc=@pointer]
    [increment pointer]
Until [acc =='Z' or Acc=0]
```

Với mã lệnh 8051 (viết theo *pseudo-code*)

```
statement: MOV A,@R0
          INC R0
          JZ exit
          CJNE A,#'Z',statement
exit:     RET
```

### d. Cấu trúc lựa chọn if/then/else:

**Ví dụ 44:** Yêu cầu bài toán: viết chương trình kiểm tra thanh ghi A: nếu MSB của thanh ghi A bằng 1 thì nạp giá trị FFH và ngược lại nạp 00H vào thanh ghi A.

*pseudo-code:*

```
if [MSB ACC ==1]
then [ACC = FFH]
else [ACC=0]
```

Với mã lệnh 8051

```
begin:      RLC A
            JNC statement2
statement1: MOV A,#0FFH
            JMP exit
statement2: CLR A
exit:
```

### Cấu trúc lựa chọn (case):

**Ví dụ 45:** Yêu cầu bài toán: Giả sử có một chương trình con có tên là INCH khi được gọi để đọc các phím nhấn ['0', '1', '2', '3'] trả về mã ASCII của phím được lưu trong thanh ghi A. Hãy viết chuỗi lệnh đọc phím nhấn và thực thi đoạn lệnh tương ứng.

*pseudo-code:*

```
case [kytu] of
    '0': [statement 0]
```

'1': [statement 1]  
 '2': [statement 2]  
 '3': [statement 3]

End\_case

Với mã lệnh 8051

```

      CALL inch
      CJNE A,#'0',skip1
Xulyphim0: ... ;xử lý công việc ứng với phím 0
      JMP exit ;kết thúc
Skip1:    CJNE A,#'1',skip2
Xulyphim1: ... ;xử lý công việc ứng với phím 1
      JMP exit
Skip2:    CJNE A,#'2',skip3
Xulyphim2: ... ;xử lý công việc ứng với phím 2
      JMP exit
Skip3:    CJNE A,#'3',exit
Xulyphim3: ... ;xử lý công việc ứng với phím 3
      JMP exit
  
```

### **Phong cách lập trình:**

Chương trình phải rõ ràng và nhất quán vì điều này rất quan trọng nếu chúng ta lập trình theo nhóm gồm nhiều người.

Cách đặt tên nhãn trong chương trình: các nhãn chỉ có thể được gán một lần trong mỗi chương trình. Sử dụng các nhãn phải làm cho việc đọc chương trình rõ ràng và dễ dàng hơn.

Với các lệnh được lặp lại thì các nhãn nên đặt tên là: *loop, back, more, ...*

Với lệnh rẽ nhánh chương trình và nhảy qua nhiều lệnh thì nên đặt tên nhãn là: *skip, ahead, ...*

Với lệnh rẽ nhánh chương trình khi đọc bit trạng thái thì nên đặt tên nhãn là: *wait, again, ...*

Có thể sử dụng các tên nhãn được sắp xếp tuần tự như: skip1, skip2, skip3, ...

Cách chú thích lệnh và chú thích khối: không nên chú thích mọi dòng mã lệnh mà chỉ chú thích những lệnh có nghĩa ẩn.

**Ví dụ 46:** Các dòng lệnh sau thì cần chú thích:

```

PUSH 00H ;cất R0 vào ngăn xếp
MOV R0,#60h ;R0 quản lý vùng nhớ có địa chỉ bắt đầu từ 60h
MOV R7,#31 ;R7 dùng làm bộ đếm
  
```

**Ví dụ 47:** Các dòng lệnh sau thì không cần phải chú thích vì lệnh đã rõ ràng:

```

PUSH ACC ;cất A vào ngăn xếp
INC R0 ;tăng R0
DEC r7 ;giảm bộ đếm1
  
```

Các chú thích khối có ở bắt đầu mỗi chương trình con và các chú thích bao gồm:

- Tên của chương trình con.
- Chức năng và giới hạn của chương trình con.
- Các điều kiện nhập xuất hay còn gọi là dữ liệu đầu vào và dữ liệu sau khi xử lý.
- Các chương trình con khác sử dụng trong chương trình con này.
- Các thanh ghi được sử dụng.

Ví dụ : Một chương trình minh họa:

;-----

;tên chương trình:

;chức năng:

;dữ liệu vào:

;dữ liệu ra:

;các chương trình con sử dụng:

;các thanh ghi có sử dụng:

;-----

#### **Lưu trữ nội dung thanh ghi và khôi phục:**

Khi gọi một chương trình con thì chương trình con đó sẽ sử dụng một số thanh ghi để phục vụ cho việc xử lý nên chúng ta phải tiến hành việc lưu dữ liệu của các thanh ghi đó vào ngăn xếp và tiến hành lấy lại trước khi kết thúc chương trình con.

#### **Tổ chức chương trình:**

Các phần chương trình được sắp xếp theo thứ tự như sau:

- Cho bảng
- Các lệnh khởi tạo trị.
- Chương trình chính.
- Các chương trình con.
- Khai báo vùng dữ liệu hằng số.
- Các ô nhớ dữ liệu RAM (sử dụng chỉ dẫn DS).

[return](#)

**the end**

# **KHẢO SÁT CÁC IC NGOẠI VI THIẾT KẾ HỆ THỐNG VI XỬ LÝ**

## CÁC LINH KIÊN SỐ

1. [IC CHỐT](#)
2. [IC GÂM MÃ](#)
3. [IC ĐÈM](#)
4. [IC NHỚ EPROM VÀ RAM](#)

## CÁC LINH KIÊN IC NGOẠI VI

1. [IC NGOẠI VI LẬP TRÌNH 8255A](#)
  - a. [Sơ đồ chân và sơ đồ khối của 8255A](#)
  - b. [Cấu trúc từ điều khiển của 8255A](#)
2. [IC NGOẠI VI LẬP TRÌNH 8279](#)
  - a. [Sơ đồ chân và sơ đồ khối của 8279](#)
  - b. [Cấu trúc từ điều khiển của 8279](#)
  - c. [Ứng dụng 8279](#)

## THIẾT KẾ HỆ THỐNG VI XỬ LÝ

1. [BÀI THIẾT KẾ SỐ 1](#)
2. [BÀI THIẾT KẾ SỐ 2](#)

## BÀI TẬP

## ***LỊỆT KÊ CÁC HÌNH H***

**Hình 6-1. Sơ đồ chân và sơ đồ logic IC chốt 74573.**

**Hình 6-2. Sơ đồ chân và sơ đồ logic IC giải mã 74138.**

**Hình 6-3. Kết nối 4 IC giải mã 74138 để giải mã 5 đường sang 32 đường.**

**Hình 6-4. Sơ đồ chân và sơ đồ logic IC giải mã 74139.**

**Hình 6-5. Sơ đồ chân và sơ đồ logic IC đệm 74245.**

**Hình 6-6. Sơ đồ chân và sơ đồ logic IC đệm 74244.**

**Hình 6-7. Sơ logic IC đệm 74244 chia ra làm 2 khối.**

**Hình 6-8. Sơ đồ chân IC nhớ họ 27xxx.**

**Hình 6-9. Sơ đồ chân IC2716, 2732, 2764.**

**Hình 6-10. Dạng sóng truy xuất bộ nhớ 2716.**

**Hình 6-11. Sơ đồ chân và sơ đồ logic IC nhớ 6264.**

**Hình 6-12. Dạng sóng đọc bộ nhớ RAM 6264.**

**Hình 6-13. Dạng sóng ghi dữ liệu vào bộ nhớ RAM 6264.**

**Hình 6-14. Sơ đồ chân và sơ đồ logic IC nhớ 62256.**

**Hình 6-15. Sơ đồ chân và sơ đồ logic của IC 8255A.**

**Hình 6-16. Sơ đồ khối của IC 8255A.**

**Hình 6-17. Cấu trúc từ điều khiển thứ nhất của IC 8255A.**

**Hình 6-18. Từ điều khiển khi 2 nhóm A, B làm việc ở mode 0.**

**Hình 6-19. Từ điều khiển khi 2 nhóm A, B làm việc ở mode 1.**

**Hình 6-20. Port A của IC 8255A hoạt động ở mode 1 – nhận dữ liệu.**

**Hình 6-21. Dạng sóng của các đường tín hiệu điều khiển – port A input.**

**Hình 6-22. Port A của IC 8255A hoạt động ở mode 1 – xuất dữ liệu.**

**Hình 6-23. Dạng sóng của các đường tín hiệu điều khiển – port A output.**

**Hình 6-24. Port B của IC 8255A hoạt động ở mode 1 – nhận dữ liệu.**

**Hình 6-25. Port B của IC 8255A hoạt động ở mode 1 – xuất dữ liệu.**

**Hình 6-26. Từ điều khiển hoạt động ở mode 2 của nhóm A.**

**Hình 6-27. Nhóm A của IC 8255A hoạt động ở mode 2.**

**Hình 6-28. Dạng sóng của các đường tín hiệu điều khiển nhóm A.**

**Hình 6-29. Từ điều khiển thứ 2 của IC 8255A.**

**Hình 6-30. Sơ đồ chân và sơ đồ logic của IC 8279.**

**Hình 6-31. Từ điều khiển keyboard/Display mode set của IC 8279.**

**Hình 6-32. Từ điều khiển Program clock của IC 8279.**

**Hình 6-33. Từ điều khiển Red FIFO/sensor RAM của IC 8279.**

**Hình 6-34. Từ điều khiển Display RAM của IC 8279.**

**Hình 6-35. Từ điều khiển End Interrupt của IC 8279.**

**Hình 6-36. Từ điều khiển Write Display RAM của IC 8279.**

**Hình 6-37. Từ điều khiển Clear của IC 8279.**

**Hình 6-38. Ba bit  $C_D$  của từ điều khiển Clear.**

**Hình 6-39. Cấu trúc mã phím của IC 8279.**

**Hình 6-40a. Sơ đồ nguyên lý của khối vi xử lý và IC chốt.**

**Hình 6-40b. Sơ đồ nguyên lý của khối bộ nhớ với vi xử lý.**

**Hình 6-40c. Sơ đồ nguyên lý của khối 2 IC ngoại vi 8255.**

**Hình 6-40d. Sơ đồ nguyên lý của khối ngoại vi 8279 với bàn phím và led 7 đoạn.**

**Hình 6-40e Sơ đồ nguyên lý của 8 led hiển thị quét.**

**Hình 6-40f. Sơ đồ nguyên lý của khối ma trận phím 8x3.**

**Hình 6-41a. Sơ đồ nguyên lý của khối bộ nhớ và IC giải mã.**

**Hình 6-41b. Sơ đồ nguyên lý của khối ngoại vi 8255.**

**Hình 6-41c. Sơ đồ nguyên lý của khối 8279.**

**Hình 6-42a. Sơ đồ nguyên lý của khối vi xử lý và IC chốt.**

**Hình 6-42b. Sơ đồ nguyên lý của khối ngoại vi.**

**Hình 6-42c. Sơ đồ nguyên lý của khối bộ nhớ.**

## *LIỆT KÊ CÁC BẢNG*

**Bảng 6-1. Bảng trạng thái hoạt động của IC chốt 74573.**

**Bảng 6-2. Bảng trạng thái hoạt động của IC giải mã 74138.**

**Bảng 6-3. Bảng trạng thái hoạt động của IC giải mã 74139.**

**Bảng 6-4. Bảng trạng thái hoạt động của IC đệm 74245.**

**Bảng 6-5. Bảng trạng thái hoạt động của IC đệm 74244.**

**Bảng 6-6. Tên các chân của IC nhớ họ 27.**

**Bảng 6-7. Bảng trạng thái hoạt động của IC2716.**

**Bảng 6-8. Bảng trạng thái hoạt động của IC2732.**

**Bảng 6-9. Bảng trạng thái hoạt động của IC2764.**

**Bảng 6-10. Bảng thông số thời gian hoạt động của IC2716.**

**Bảng 6-11. Tên các chân của IC nhớ 6264.**

**Bảng 6-12. Bảng trạng thái hoạt động của IC6264.**

**Bảng 6-13. Bảng thông số thời gian hoạt động đọc của IC 6264.**

**Bảng 6-14. Bảng thông số thời gian hoạt động ghi của RAM 6264.**

**Bảng 6-15. Tên các chân và bảng trạng thái của IC nhớ 62256.**

**Bảng 6-16. Tên các chân của IC 8255A.**

**Bảng 6-17. Tên các chân của IC 8279.**

**Bảng 6-18. Các mod hiển thị khác nhau của 8279.**

**Bảng 6-19. Các mod quét ma trận phím khác nhau của 8279.**

**Bảng 6-20. Bảng địa chỉ bộ nhớ.**

**Bảng 6-21. Bảng địa chỉ IO.**

**Bảng 6-22. Bảng địa chỉ bộ nhớ gồm bộ nhớ và IO.**

**Bảng 6-23. Bảng địa chỉ bộ nhớ cho 4 eprom và 1 ram.**

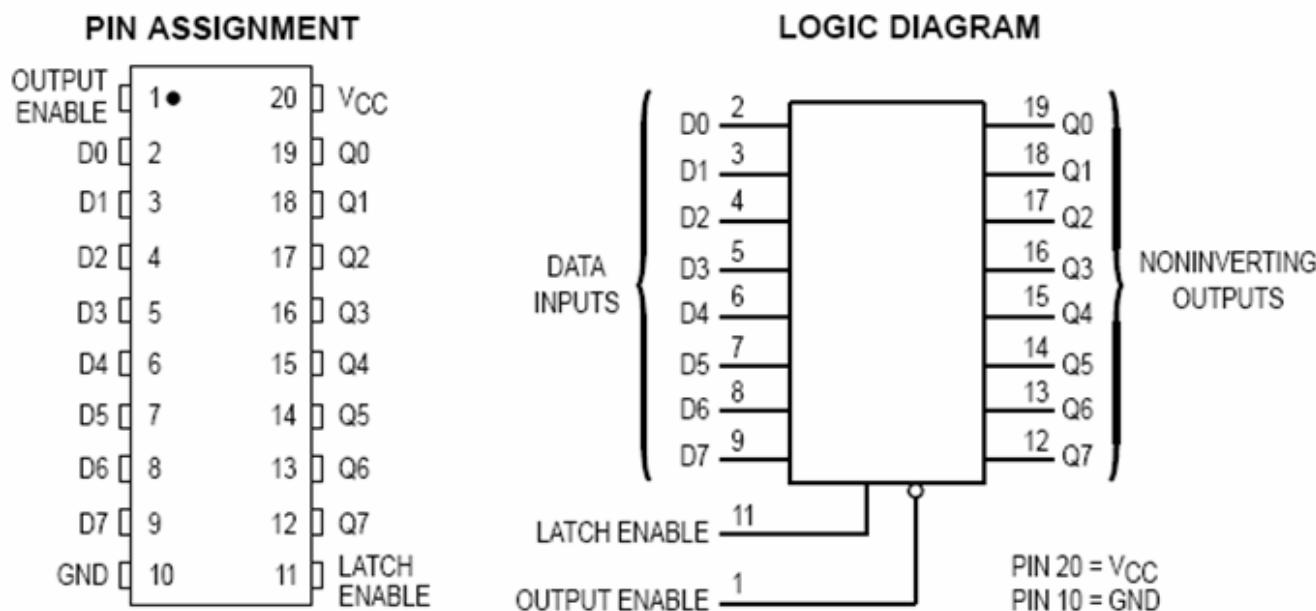
**Bảng 6-24. Bảng địa chỉ bộ nhớ cho 4 rom và 4 ram.**

## II. CÁC LINH KIỆN IC SỐ:

### 1. IC CHỐT:

Trong hệ thống vi xử lý có đa hợp địa chỉ và dữ liệu thì khi kết nối vi xử lý với hệ thống phải tiến hành tách riêng từng đường địa chỉ và dữ liệu. Các IC thường được dùng để tách địa chỉ và dữ liệu là IC chốt 8 bit 74573.

Sơ đồ chân và sơ đồ kí hiệu logic như hình 6-1:



Hình 6-1. Sơ đồ chân và sơ đồ logic IC chốt 74573.

Bảng trạng thái hoạt động của IC chốt 74573.

FUNCTION TABLE

Inputs		Output	
Output Enable	Latch Enable	D	Q
L	H	H	H
L	H	L	L
L	L	X	No Change
H	X	X	Z

X = Don't Care  
Z = High Impedance

Bảng 6-1. Bảng trạng thái hoạt động của IC chốt 74573.

Trong bảng trạng thái thì khi ngõ cho phép chốt (LE) ở mức cao (H) thì dữ liệu ngõ ra Q phụ thuộc vào dữ liệu ngõ vào D ( $Q = D$ ), khi LE ở mức thấp dữ liệu tại các ngõ ra được chốt lại không phụ còn phụ thuộc dữ liệu của ngõ vào (không thay đổi) bất chấp dữ liệu ngõ vào. Khi đó ta xem dữ liệu đã được chốt lại.

Chân điều khiển cho phép OE tích cực mức thấp. Khi OE ở mức cao thì 8 ngõ ra sẽ ở trạng thái tổng trống.

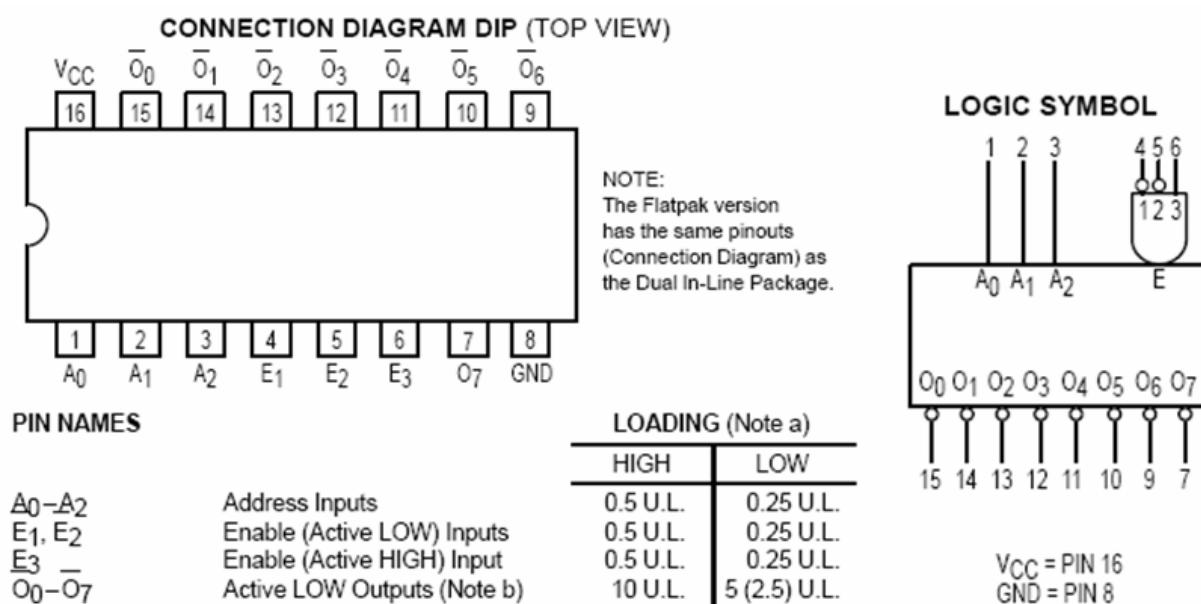
## 2 IC GIẢI MÃ

Trong hệ thống vi xử lý thường kết nối với nhiều bộ nhớ và nhiều thiết bị ngoại vi khác nhau. Vì các bộ nhớ và thiết bị ngoại vi sử dụng chung bus địa chỉ và bus dữ liệu nên phải tiến hành giải mã để phân chia các vùng nhớ khác nhau cho các bộ nhớ và các thiết bị ngoại vi sao cho một địa chỉ tương ứng với 1 ô nhớ duy nhất.

Các IC số thường được dùng để giải mã bộ nhớ trong các hệ thống vi xử lý là các cổng logic, các IC giải mã và các IC PROM hay PAL, ...

Trong phần này giới thiệu các IC giải mã thường sử dụng như IC 74139 giải mã 2 đường sang 4 đường và IC 74138 giải mã 3 đường sang 8 đường.

Sơ đồ chân và sơ đồ kí hiệu logic như hình 6-2:



Hình 6-2. Sơ đồ chân và sơ đồ logic IC giải mã 74138.

Bảng trạng thái hoạt động của IC giải mã 74138.

INPUTS						OUTPUTS							
E <sub>1</sub>	E <sub>2</sub>	E <sub>3</sub>	A <sub>0</sub>	A <sub>1</sub>	A <sub>2</sub>	O <sub>0</sub>	O <sub>1</sub>	O <sub>2</sub>	O <sub>3</sub>	O <sub>4</sub>	O <sub>5</sub>	O <sub>6</sub>	O <sub>7</sub>
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

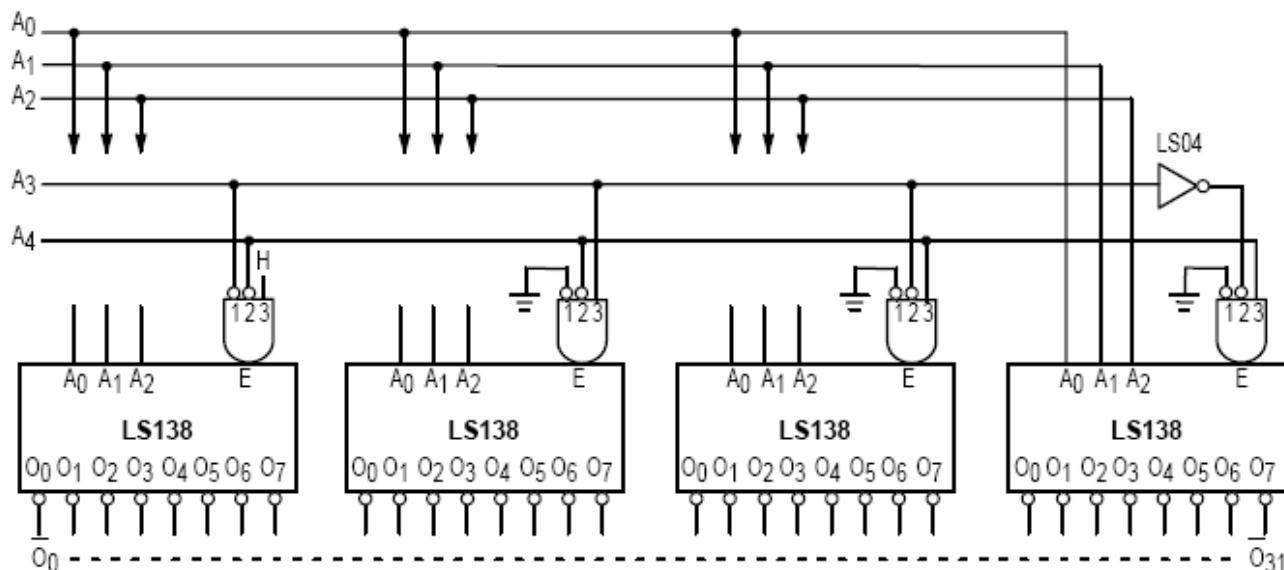
H = HIGH Voltage Level  
L = LOW Voltage Level  
X = Don't Care

Bảng 6-2. Bảng trạng thái hoạt động của IC giải mã 74138.

Trong bảng trạng thái, 3 trạng thái đầu thì IC không được phép giải mã – tất cả các ngõ ra đều ở mức H.

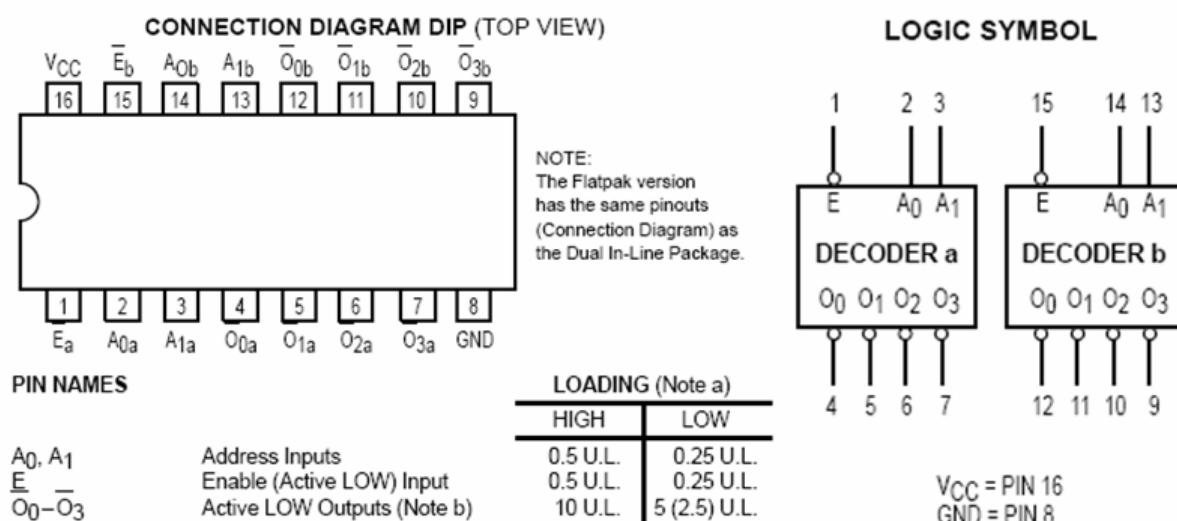
8 trạng thái còn lại cho phép giải mã 8 ngõ ra theo 3 ngõ vào, mỗi một trạng thái có 1 ngõ ra ở mức L.

Khi cần nhiều ngõ ra thì kết nối nhiều IC giải mã 74138 ví dụ như cần giải mã 5 đường sang 32 đường thì sẽ dùng 4 IC giải mã kết nối như hình 6-3:



**Hình 6-3. Kết nối 4 IC giải mã 74138 để giải mã 5 đường sang 32 đường.**

Cùng họ với IC giải mã 74138 thì còn IC giải mã 74139 giải mã 2 đường sang 4 đường có sơ đồ chân và sơ đồ logic như hình 6-4:



**Hình 6-4. Sơ đồ chân và sơ đồ logic IC giải mã 74139.**

Trong IC giải mã 74139 có 2 mạch giải mã 2 đường sang 4 đường.

Bảng trạng thái hoạt động của IC giải mã 74139.

TRUTH TABLE					
INPUTS			OUTPUTS		
E	A <sub>0</sub>	A <sub>1</sub>	O <sub>0</sub>	O <sub>1</sub>	O <sub>2</sub>
H	X	X	H	H	H
L	L	L	L	H	H
L	H	L	H	L	H
L	L	H	H	H	L
L	H	H	H	H	L

H = HIGH Voltage Level

L = LOW Voltage Level

X = Don't Care

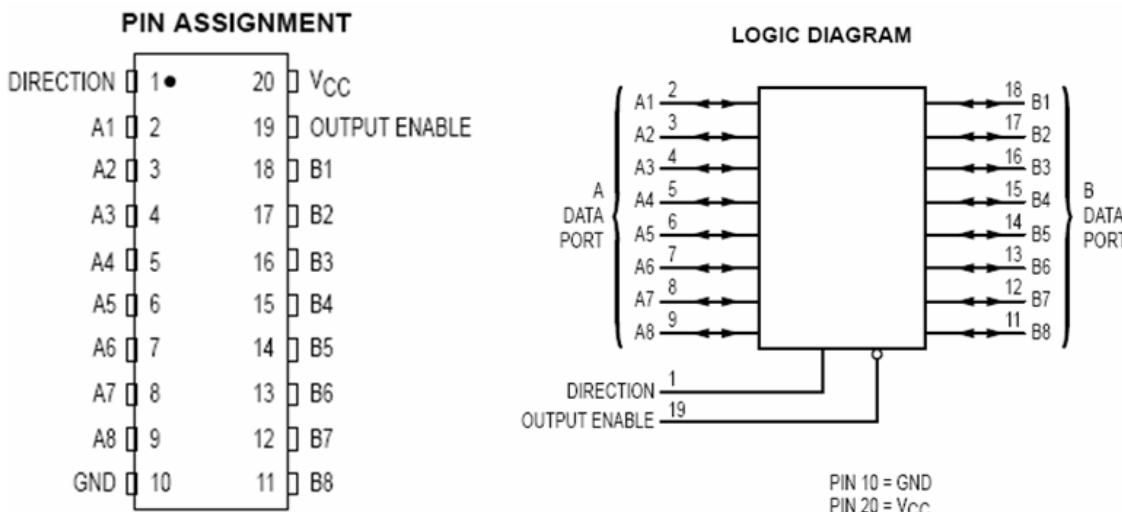
**Bảng 6-3. Bảng trạng thái hoạt động của IC giải mã 74139.**

Ở trạng thái đầu tiên thì chân cho phép E ở mức H sẽ không cho phép giải mã, các ngõ ra ở mức H. 4 trạng thái còn lại thì cho phép giải mã, ngõ ra tích cực mức thấp.

### 3. IC ĐỆM

Các IC đếm có dòng vào, dòng ra, áp vào, áp ra lớn để đếm các đường địa chỉ và các đường dữ liệu khi vi xử lý kết nối với nhiều IC nhỏ hay IC ngoại vi. Có 2 dạng IC đếm: đếm 1 chiều và đếm 2 chiều.

Đếm 2 chiều thường sử dụng là IC74245 có sơ đồ chân và sơ đồ logic như hình 6-5:

**Hình 6-5. Sơ đồ chân và sơ đồ logic IC đếm 74245.**

Bảng trạng thái hoạt động của IC đếm 74245.

**FUNCTION TABLE**

Control Inputs		Operation
Output Enable	Direction	
L	L	Data Transmitted from Bus B to Bus A
L	H	Data Transmitted from Bus A to Bus B
H	X	Buses Isolated (High-Impedance State)

X = don't care

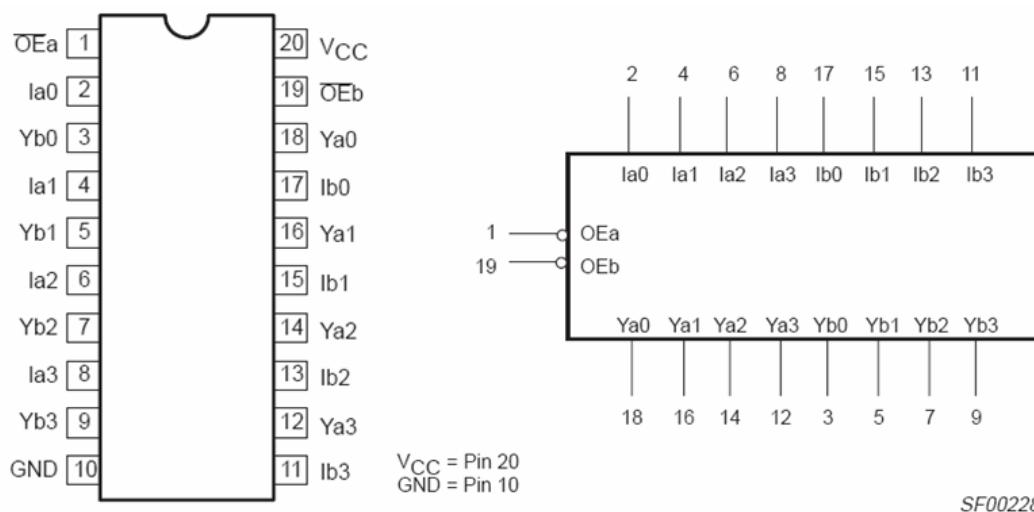
**Bảng 6-4. Bảng trạng thái hoạt động của IC đếm 74245.**

Trong bảng trạng thái thì 2 trạng thái đầu OE = L cho phép IC hoạt động và nếu Direction = L thì dữ liệu truyền từ bus ngõ vào B sang bus ngõ ra A, Direction = H thì dữ liệu truyền từ bus ngõ vào A sang bus ngõ ra B. Khi OE = H thì các bus ở trạng thái tổng trở cao.

IC 74245 thường được dùng để đệm bus dữ liệu hai chiều của vi xử lý.

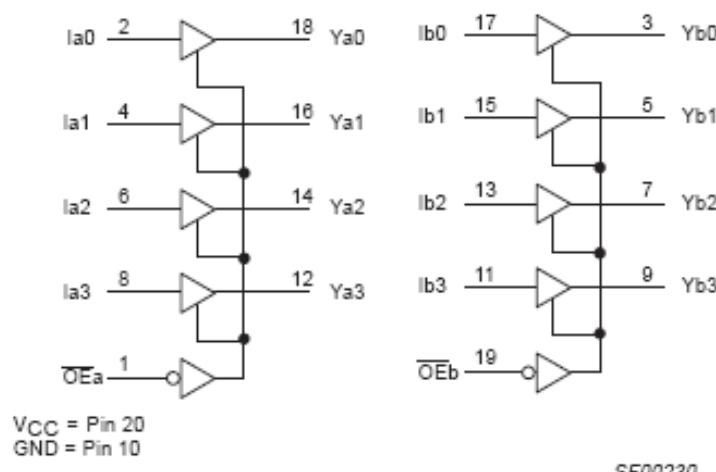
Ngoài IC đệm 2 chiều còn có các IC đệm một chiều 74244 và IC đệm đảo 74240:

Đệm 1 chiều thường sử dụng là IC74244 có sơ đồ chân và sơ đồ logic như hình 6-6:



**Hình 6-6. Sơ đồ chân và sơ đồ logic IC đệm 74244.**

Trong sơ đồ logic hình 6-7 sẽ thấy rõ ràng hơn về IC đệm 74244: trong IC 74244 chia ra làm 2 khối đếm 4 bit hoạt động độc lập.



**Hình 6-7. Sơ logic IC đệm 74244 chia ra làm 2 khối.**

Bảng trạng thái hoạt động của IC đệm 74244.

**FUNCTION TABLE**

INPUTS				OUTPUTS	
$\overline{OE}_a$	$I_a$	$\overline{OE}_b$	$I_b$	$Y_a$	$Y_b$
L	L	L	L	L	L
L	H	L	H	H	H
H	X	H	X	Z	Z

**NOTES:**

H = High voltage level

L = Low voltage level

X = Don't care

Z = High impedance "off" state

**Bảng 6-5. Bảng trạng thái hoạt động của IC đệm 74244.**

Trong bảng trạng thái thì 2 trạng thái đầu  $OE = L$  cho phép IC đệm dữ liệu. Khi  $OE = H$  thì các ngõ ra ở trạng thái tổng trở cao.

IC 74240 là IC đệm hoạt động giống như IC đệm 74244 nhưng ngõ ra bị đảo.

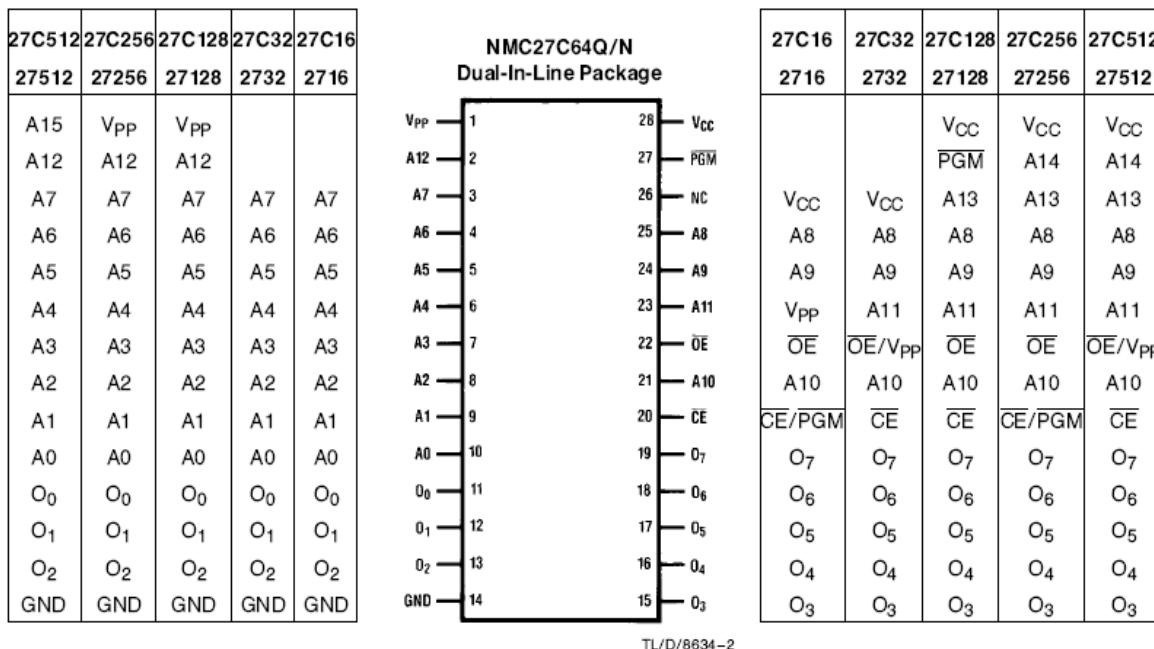
**4. IC NHỚ EPROM, SRAM**

Trong các hệ thống vi xử lý luôn sử dụng bộ nhớ ROM và bộ nhớ RAM để lưu chương trình và dữ liệu xử lý. Tuỳ thuộc vào yêu cầu mà hệ thống bộ nhớ được sử dụng nhiều hay ít.

Các bộ nhớ ROM thường sử dụng là loại EPROM họ 27xxx, loại EEPROM họ 28xxx, họ 29xxx.

Bộ nhớ RAM sử dụng nhiều loại và phổ biến là họ 62xx.

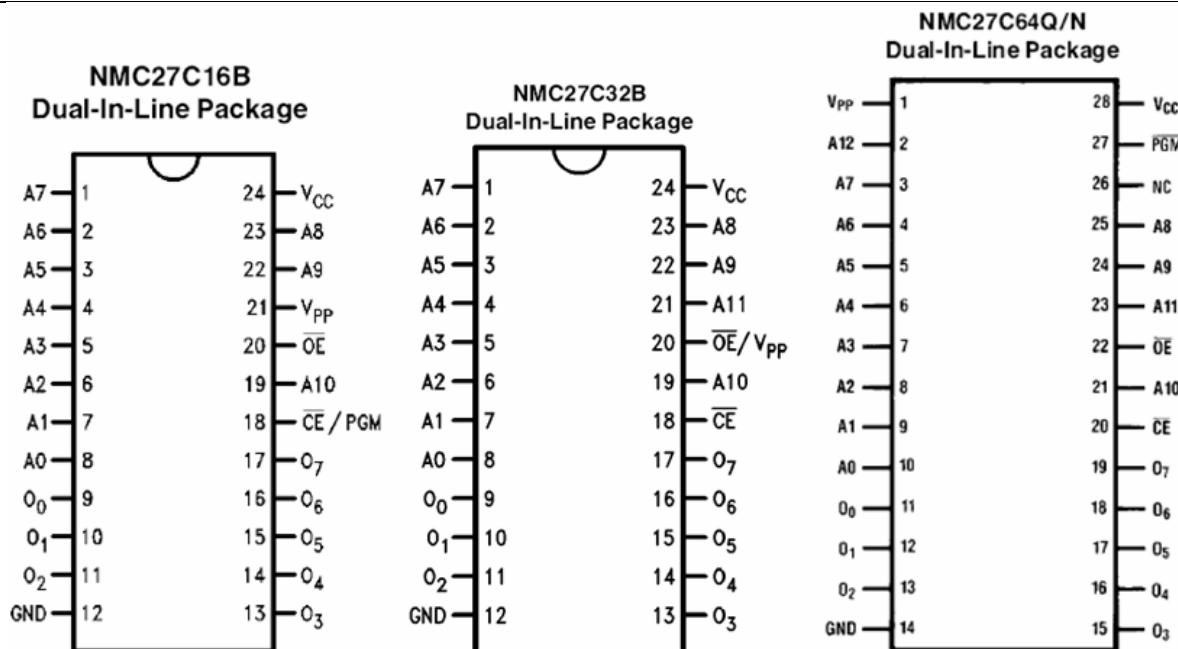
**Khảo sát EPROM họ 27xxx:** bao gồm 2716 đến 27512 như hình 6-8:

**Connection Diagram**

TL/D/8634-2

**Hình 6-8. Sơ đồ chân IC nhớ họ 27xxx.**

Trong sơ đồ tổng quát hình 6-8 thì các IC 2716 và 2732 chỉ có 24 chân, các IC từ 2764 đến 27512 thì có 28 chân, để quan sát rõ hơn hãy xem hình 6-9. EPROM 2716 có dung lượng 2 Kbyte, 2732 có dung lượng 4Kbyte, 2764 có dung lượng 8Kbyte.

**Hình 6-9. Sơ đồ chân IC2716, 2732, 2764.**

Tên của các chân như bảng 6-6:

Pin Names	
A0–A12	Addresses
CE	Chip Enable
OE	Output Enable
O <sub>0</sub> –O <sub>7</sub>	Outputs
PGM	Program
NC	No Connect
V <sub>PP</sub>	Programming Voltage
V <sub>CC</sub>	Power Supply
GND	Ground

**Bảng 6-6. Tên các chân của IC nhớ họ 27.**

Bảng trạng thái hoạt động của các IC nhớ 2716, 2732 và 2764: như các bảng 6-7, 6-8, 6-9.

Mode	Pins	CE/PGM (18)	OE (20)	V <sub>PP</sub> (21)	V <sub>CC</sub> (24)	Outputs (9-11), (13-17)
Read	V <sub>IL</sub>	V <sub>IL</sub>	V <sub>CC</sub>	5	D <sub>OUT</sub>	
Standby	V <sub>IH</sub>	Don't Care	V <sub>CC</sub>	5	Hi-Z	
Output Disable	Don't Care	V <sub>IH</sub>	V <sub>CC</sub>	5	Hi-Z	
Program	V <sub>IH</sub>	V <sub>IH</sub>	12.75V	6.25	D <sub>IN</sub>	
Program Verify	V <sub>IL</sub>	V <sub>IL</sub>	12.75V	6.25	D <sub>OUT</sub>	
Program Inhibit	V <sub>IL</sub>	V <sub>IH</sub>	12.75V	6.25	Hi-Z	

**Bảng 6-7. Bảng trạng thái hoạt động của IC2716.**

Pins	$\overline{CE}$ (18)	$\overline{OE}/V_{PP}$ (20)	$V_{CC}$ (24)	Outputs (9-11, 13-17)
Mode				
Read	$V_{IL}$	$V_{IL}$	5V	$D_{OUT}$
Standby	$V_{IH}$	Don't Care	5V	Hi-Z
Program	$V_{IL}$	12.75V	6.25V	$D_{IN}$
Program Verify	$V_{IL}$	$V_{IL}$	6.25V	$D_{OUT}$
Program Inhibit	$V_{IH}$	12.75V	6.25V	Hi-Z
Output Disable	Don't Care	$V_{IH}$	5V	Hi-Z

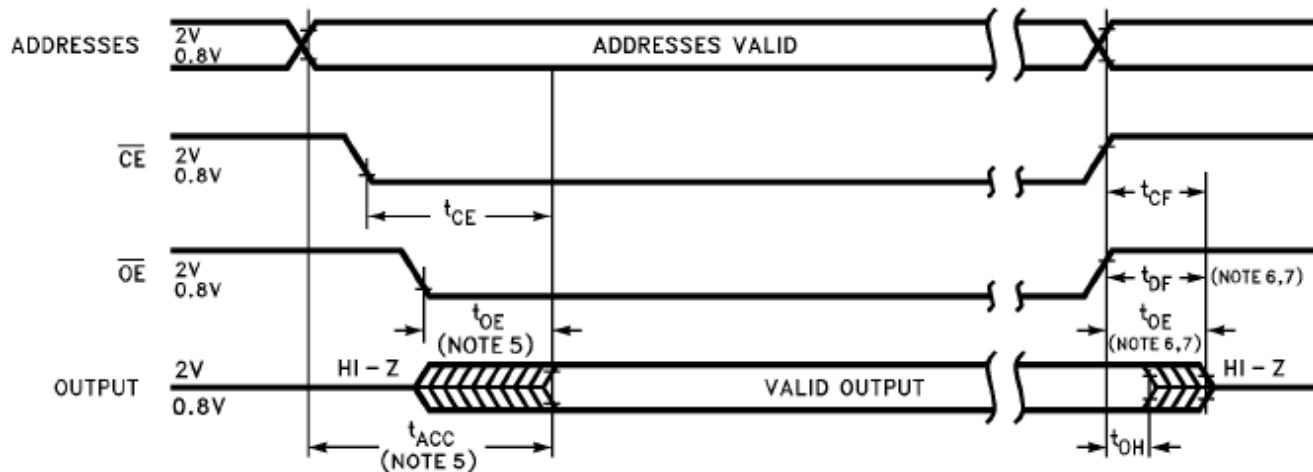
Bảng 6-8. Bảng trạng thái hoạt động của IC2732.

Mode	Pins	$\overline{CE}$ (20)	$\overline{OE}$ (22)	$PGM$ (27)	$V_{PP}$ (1)	$V_{CC}$ (28)	Outputs (11-13, 15-19)
Read	$V_{IL}$	$V_{IL}$	$V_{IH}$	5V	5V	5V	$D_{OUT}$
Standby	$V_{IH}$	Don't Care	Don't Care	5V	5V	5V	Hi-Z
Output Disable	Don't Care	$V_{IH}$	$V_{IH}$	5V	5V	5V	Hi-Z
Program	$V_{IL}$	$V_{IH}$		13V	6V	6V	$D_{IN}$
Program Verify	$V_{IL}$	$V_{IL}$	$V_{IH}$	13V	6V	6V	$D_{OUT}$
Program Inhibit	$V_{IH}$	Don't Care	Don't Care	13V	6V	6V	Hi-Z

Bảng 6-9. Bảng trạng thái hoạt động của IC2764.

Trong bảng trạng thái liệt kê đầy đủ các trạng thái nhưng nếu sử dụng trong hệ thống giao tiếp với vi xử lý thì chỉ cần quan tâm đến các trạng thái sau: “Read”, “Standby” và “Output Disable”. Các trạng thái còn lại dành cho các nhà nghiên cứu thiết kế bộ nạp Eprom.

Khi sử dụng bộ nhớ kết nối với vi xử lý thì cần phải quan tâm đến các thông số của IC nhớ cho đầy đủ trong datasheet của IC. Có rất nhiều thông số nhưng người sử dụng cần biết các thông số cơ bản đó là thời gian truy xuất bộ nhớ. Hình 6-10 trình bày dạng sóng truy xuất bộ nhớ của IC nhớ 2716:



Hình 6-10. Dạng sóng truy xuất bộ nhớ 2716.

Phân tích dạng sóng: sau khi bộ nhớ 2716 nhận được địa chỉ đã xác lập, tiếp theo chân điều khiển  $\overline{CE}$  chuyển trạng thái từ H xuống mức L để cho phép bộ nhớ, tiếp theo chân  $\overline{OE}$  chuyển trạng thái từ H xuống mức L để cho phép xuất dữ liệu.

Nhìn vào dạng sóng thì ta thấy có 3 thông số:

- Thời số  $t_{OE}$  được tính từ khi chân  $\overline{OE}$  xuống mức L cho đến khi dữ liệu xuất ra bus.
- Thời số  $t_{CE}$  được tính từ khi chân  $\overline{CE}$  xuống mức L cho đến khi dữ liệu xuất ra bus.
- Thời số  $t_{ACC}$  được tính từ khi địa chỉ xác lập cho đến khi dữ liệu xuất ra bus (thông số này thường được gọi là thời gian truy xuất).

Cả 3 thông số được cho ở bảng 6-10. Trong bảng này liệt kê các thông số cho 2 loại bộ nhớ.

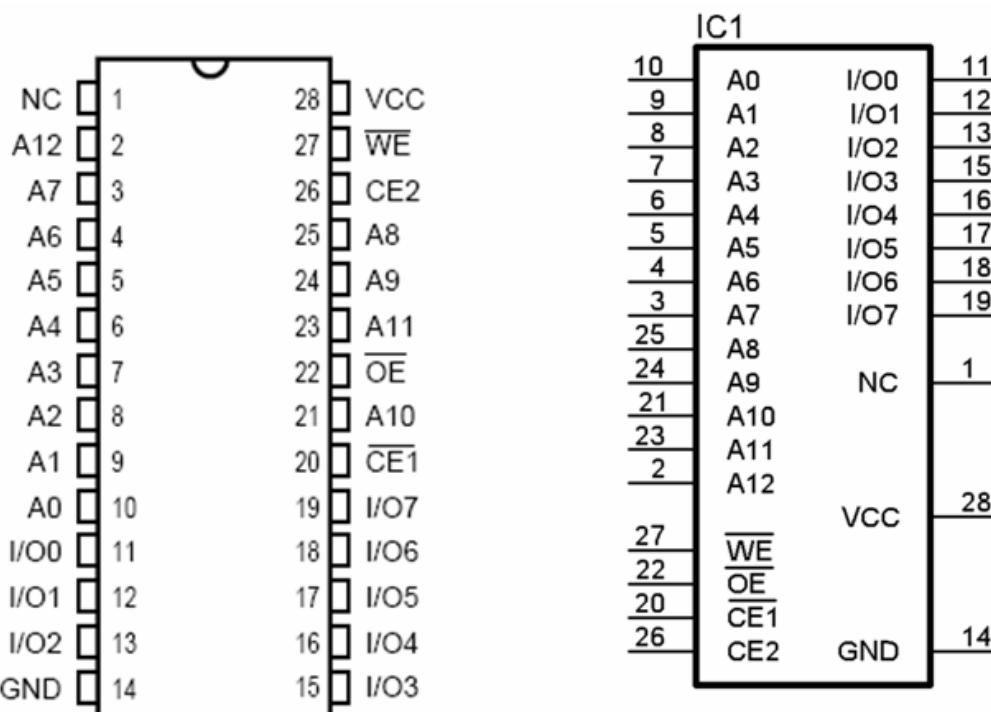
### AC Electrical Characteristics

Symbol	Parameter	Conditions	NMC27C16BQ				Units	
			Q150, QE150		Q200, QE200			
			Min	Max	Min	Max		
$t_{ACC}$	Address to Output Delay	$\overline{CE} = \overline{OE} = V_{IL}$		150		200	ns	
$t_{CE}$	$\overline{CE}$ to Output Delay	$\overline{OE} = V_{IL}$		150		200	ns	
$t_{OE}$	$\overline{OE}$ to Output Delay	$\overline{CE} = V_{IL}$		60		60	ns	
$t_{DF}$	$\overline{OE}$ High to Output Float	$\overline{CE} = V_{IL}$	0	50	0	60	ns	
$t_{OF}$	$\overline{CE}$ High to Output Float	$\overline{OE} = V_{IL}$	0	50	0	60	ns	
$t_{OH}$	Output Hold from Addresses, $\overline{CE}$ or $\overline{OE}$ , Whichever Occurred First	$\overline{OE} = \overline{CE} = V_{IL}$	0		0		ns	

Bảng 6-10. Bảng thông số thời gian hoạt động của IC2716.

Khảo sát SRAM họ 62xxx: bao gồm 6264 và 62256, 62512:

**SRAM 6264:** bộ nhớ này có dung lượng 8 kbyte có sơ đồ chân và sơ đồ logic như hình 6-11:



Hình 6-11. Sơ đồ chân và sơ đồ logic IC nhớ 6264.

Tên của các chân như bảng 6-11:

**PIN DESCRIPTIONS**

A0-A12	Address Inputs
$\overline{CE1}$	Chip Enable 1 Input
CE2	Chip Enable 2 Input
$\overline{OE}$	Output Enable Input
$\overline{WE}$	Write Enable Input
I/O0-I/O7	Input/Output
Vcc	Power
GND	Ground

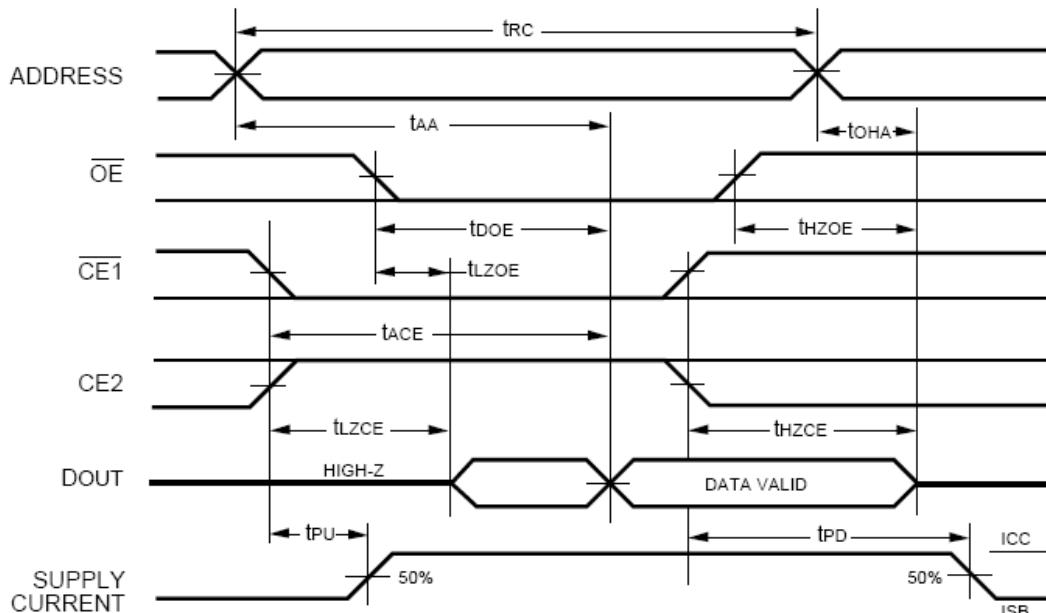
**Bảng 6-11. Tên các chân của IC nhớ 6264.**

Bảng trạng thái hoạt động của các IC nhớ 6264 như bảng 6-12:

Mode	WE	$\overline{CE1}$	CE2	$\overline{OE}$	I/O Operation	Vcc Current
Not Selected (Power-down)	X	H	X	X	High-Z	$I_{SB1}, I_{SB2}$
	X	X	L	X	High-Z	$I_{SB1}, I_{SB2}$
Output Disabled	H	L	H	H	High-Z	$I_{CC1}, I_{CC2}$
Read	H	L	H	L	DOUT	$I_{CC1}, I_{CC2}$
Write	L	L	H	X	DIN	$I_{CC1}, I_{CC2}$

**Bảng 6-12. Bảng trạng thái hoạt động của IC6264.**

**Dạng sóng đọc dữ liệu từ bộ nhớ RAM 6264 và các thông số:** Tương tự như bộ nhớ EPROM khi sử dụng phải quan tâm đến các thông số truy xuất bộ nhớ đọc và ghi dữ liệu. Hình 6-12 trình bày dạng sóng đọc dữ liệu bộ nhớ của RAM 6264:

**Hình 6-12. Dạng sóng đọc bộ nhớ RAM 6264.**

Phân tích dạng sóng: sau khi bộ nhớ 6264 nhận được địa chỉ đã xác lập, tiếp theo các chân điều khiển  $\overline{CE1}$  và CE2 chuyển sang trạng thái cho phép bộ nhớ, tiếp theo chân  $\overline{OE}$  chuyển trạng thái từ H xuống mức L để cho phép xuất dữ liệu.

Nhìn vào dạng sóng thì ta thấy có 3 thông số:

- Thông số  $t_{DOE}$  được tính từ khi chân  $\overline{OE}$  xuống mức L cho đến khi dữ liệu xuất ra bus.

- Thời gian truy xuất  $t_{ACE1}$  được tính từ khi chân  $\overline{CE1}$  xuống mức L cho đến khi dữ liệu xuất ra bus.
- Thời gian truy xuất  $t_{ACE2}$  được tính từ khi chân  $CE2$  xuống mức L cho đến khi dữ liệu xuất ra bus.
- Thời gian truy xuất  $t_{AA}$  được tính từ khi địa chỉ xác lập cho đến khi dữ liệu xuất ra bus (thời gian này thường được gọi là thời gian truy xuất).

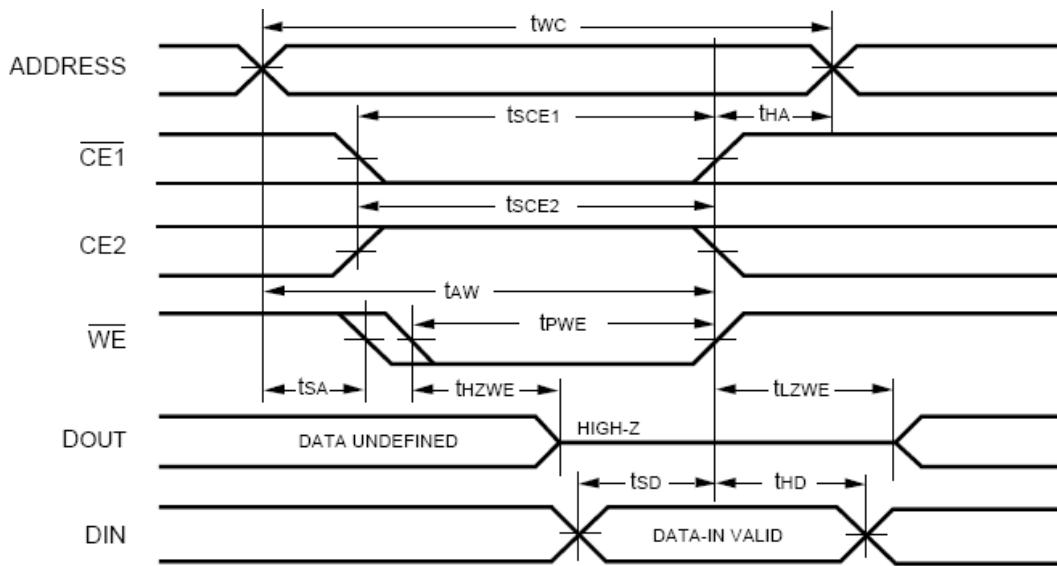
Cả 3 thông số được cho ở bảng 6-13.

#### READ CYCLE SWITCHING CHARACTERISTICS<sup>(1)</sup> (Over Commercial Operating Range)

Symbol	Parameter	-45 ns		-70 ns		-100 ns		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
$t_{RC}$	Read Cycle Time	45	—	70	—	100	—	ns
$t_{AA}$	Address Access Time	—	45	—	70	—	100	ns
$t_{OHA}$	Output Hold Time	3	—	3	—	3	—	ns
$t_{ACE1}$	$\overline{CE1}$ Access Time	—	45	—	70	—	100	ns
$t_{ACE2}$	$CE2$ Access Time	—	45	—	70	—	100	ns
$t_{DOE}$	$\overline{OE}$ Access Time	—	25	—	35	—	50	ns
$t_{LZOE}^{(2)}$	$\overline{OE}$ to Low-Z Output	0	—	0	—	0	—	ns
$t_{HZOE}^{(2)}$	$\overline{OE}$ to High-Z Output	—	20	—	25	—	25	ns
$t_{LZCE1}^{(2)}$	$\overline{CE1}$ to Low-Z Output	3	—	3	—	3	—	ns
$t_{LZCE2}^{(2)}$	$CE2$ to Low-Z Output	3	—	3	—	3	—	ns
$t_{HZCE}^{(2)}$	$\overline{CE1}$ or $CE2$ to High-Z Output	—	20	—	25	—	25	ns
$t_{PU}^{(3)}$	$\overline{CE1}$ or $CE2$ to Power-Up	0	—	0	—	0	—	ns
$t_{PD}^{(3)}$	$\overline{CE1}$ or $CE2$ to Power-Down	—	30	—	50	—	50	ns

Bảng 6-13. Bảng thông số thời gian hoạt động đọc của IC 6264.

**Dạng sóng ghi dữ liệu vào bộ nhớ RAM 6264 và các thông số:** dạng sóng ghi dữ liệu vào ô nhớ của RAM 6264 như hình 6-13:



Hình 6-13. Dạng sóng ghi dữ liệu vào bộ nhớ RAM 6264.

Phân tích dạng sóng: sau khi bộ nhớ 6264 nhận được địa chỉ đã xác lập, tiếp theo các chân điều khiển  $CE1$  và  $CE2$  chuyển sang trạng thái cho phép bộ nhớ, tiếp theo chân  $WE$  chuyển trạng thái từ H xuống mức L để cho phép ghi dữ liệu.

Nhìn vào dạng sóng thì ta thấy có 3 thông số:

- Thông số  $t_{PWE}$  được tính từ khi chân WE xuống mức L cho đến khi dữ liệu được ghi.
- Thông số  $t_{SCE1}$  được tính từ khi chân CE1 xuống mức L cho đến khi dữ liệu được ghi.
- Thông số  $t_{SCE2}$  được tính từ khi chân CE2 xuống mức L cho đến khi dữ liệu được ghi.
- Thông số  $t_{WC}$  được tính từ khi địa chỉ xác lập cho đến khi dữ liệu được ghi (thông số này thường được gọi là chu kỳ ghi).

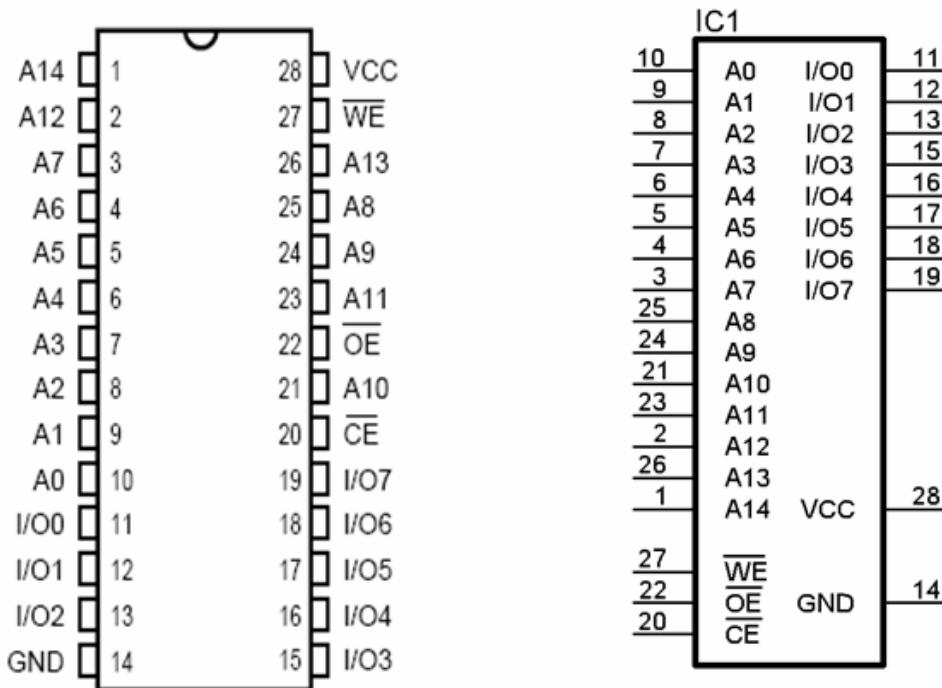
Cả 3 thông số được cho ở bảng 6-14.

#### WRITE CYCLE SWITCHING CHARACTERISTICS<sup>(1,3)</sup> (Over Commercial Operating Range)

Symbol	Parameter	-45 ns		-70 ns		-100 ns		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
$t_{WC}$	Write Cycle Time	45	—	70	—	100	—	ns
$t_{SCE1}$	CE1 to Write End	35	—	60	—	80	—	ns
$t_{SCE2}$	CE2 to Write End	35	—	60	—	80	—	ns
$t_{AW}$	Address Setup Time to Write End	35	—	60	—	80	—	ns
$t_{HA}$	Address Hold from Write End	0	—	0	—	0	—	ns
$t_{SA}$	Address Setup Time	0	—	0	—	0	—	ns
$t_{PWE}^{(4)}$	WE Pulse Width	35	—	55	—	60	—	ns
$t_{SD}$	Data Setup to Write End	25	—	30	—	35	—	ns
$t_{HD}$	Data Hold from Write End	0	—	0	—	0	—	ns
$t_{LZWE}^{(2)}$	WE LOW to High-Z Output	—	20	—	25	—	25	ns
$t_{LZWE}^{(2)}$	WE HIGH to Low-Z Output	0	—	0	—	0	—	ns

Bảng 6-14. Bảng thông số thời gian hoạt động ghi của RAM 6264.

SRAM 62256: bộ nhớ này có dung lượng 32 kbyte có sơ đồ chân và sơ đồ logic như hình 6-14:



Hình 6-14. Sơ đồ chân và sơ đồ logic IC nhớ 62256.

Tên của các chân và bảng trạng thái hoạt động như bảng 6-15:

<b>PIN DESCRIPTIONS</b>		<b>TRUTH TABLE</b>					
		<b>Mode</b>	<b>WE</b>	<b>CE</b>	<b>OE</b>	<b>I/O Operation</b>	<b>Vcc Current</b>
A0-A14	Address Inputs	Not Selected (Power-down)	X	H	X	High-Z	I <sub>SB1</sub> , I <sub>SB2</sub>
<b>CE</b>	Chip Enable Input	Output Disabled	H	L	H	High-Z	I <sub>CC1</sub> , I <sub>CC2</sub>
<b>OE</b>	Output Enable Input	Read	H	L	L	D <sub>OUT</sub>	I <sub>CC1</sub> , I <sub>CC2</sub>
<b>WE</b>	Write Enable Input	Write	L	L	X	D <sub>IN</sub>	I <sub>CC1</sub> , I <sub>CC2</sub>
I/O0-I/O7	Input/Output						
Vcc	Power						
GND	Ground						

Bảng 6-15. Tên các chân và bảng trạng thái của IC nhớ 62256.

### III. CÁC LINH KIỆN IC NGOẠI VI :

#### 1. IC ngoại vi lập trình 8255A:

Vi xử lý không thể trực tiếp điều khiển các thiết bị dù chỉ điều khiển 1 bóng đèn Led, mọi thiết bị ngoại vi muốn giao tiếp với vi xử lý đều thông qua các IC ngoại vi.

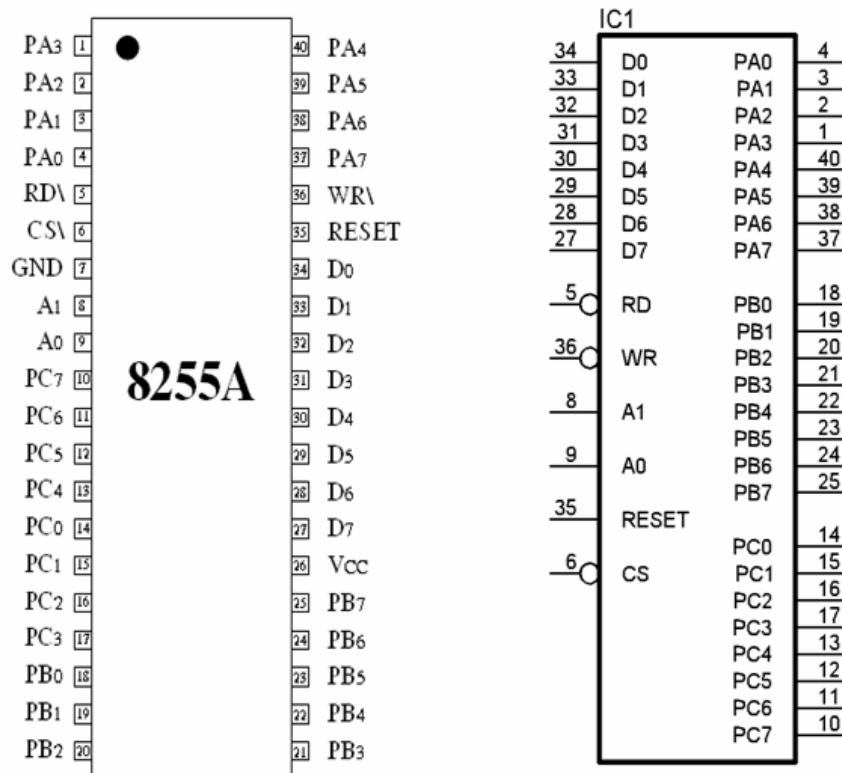
Cùng với vi xử lý thì có rất nhiều IC ngoại vi có các chức năng khác nhau phục vụ cho vi xử lý như đã trình bày ở phần trước. Một trong những IC được sử dụng phổ biến nhất là IC ngoại vi 8255A.

IC ngoại vi 8255A được chế tạo theo công nghệ LSI dùng để giao tiếp giữa vi xử lý và thiết bị bên ngoài.

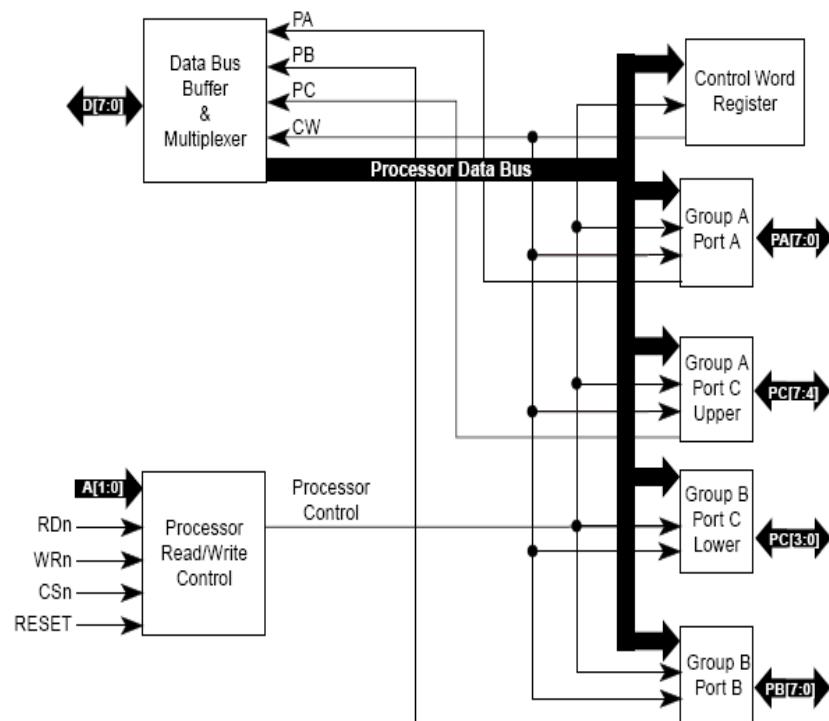
Vi mạch 8255A thường được gọi là mạch giao tiếp ngoại vi lập trình được (Programmable Peripheral Interface – PPI). Do khả năng đa năng trong các ứng dụng thực tế nên 8255A là vi mạch giao tiếp được dùng rất phổ biến cho các hệ vi xử lý 8 bit – 16 bit.

#### (a) Sơ đồ chân và sơ đồ khối của 8255A :

IC 8255A có 40 chân như hình 6-15 và sơ đồ cấu trúc bên trong như hình 6-16:



Hình 6-15. Sơ đồ chân và sơ đồ logic của IC 8255A.



Hình 6-16. Sơ đồ khối của IC 8255A.

Tên các chân của IC 8255A như bảng sau:

Signal	Signal Direction	Description
D[7:0]	In/Out	Data Bus
A[1:0]	Input	Address
RD <sub>n</sub>	Input	Read Control
WR <sub>n</sub>	Input	Write Control
CS <sub>n</sub>	Input	Chip Select
RESET	Input	Reset
PA[7:0]	In/Out	Port A
PC[7:0]	In/Out	Port B
PB[7:0]	In/Out	Port C

**Bảng 6-16. Tên các chân của IC 8255A.**

IC 8255A giao tiếp với vi xử lý thông qua 3 bus: bus dữ liệu 8 bit D<sub>7</sub>-D<sub>0</sub>, bus địa chỉ A<sub>1</sub>A<sub>0</sub>, bus điều khiển RD, WR, CS, RESET.

- Mã lệnh và dữ liệu đều được truyền trên 8 đường dữ liệu D7-D0. Vi xử lý gởi dữ liệu đến 8255A hoặc vi xử lý đọc dữ liệu từ 8255A tùy thuộc vào lệnh điều khiển. Các đường tín hiệu RD, WR của 8255A được kết nối với các đường RD, WR của vi xử lý.

- Tín hiệu RESET dùng để khởi động 8255A khi cấp điện, khi bị reset các thanh ghi bên trong 8255A đều bị xóa và 8255A ở trạng thái sẵn sàng làm việc. Khi giao tiếp với vi xử lý ngõ vào tín hiệu RESET này được kết nối với tín hiệu RESET ngõ ra của vi xử lý.

- Tín hiệu Chip Select (CS) dùng để chọn 8255A khi vi xử lý giao tiếp với nhiều 8255A.

8255A có 3 port xuất nhập (I/O) có tên port A, port B, port C, mỗi port 8 bit. Port A gồm các bit PA<sub>0</sub>-PA<sub>7</sub>, port B gồm các bit PB<sub>0</sub>-PB<sub>7</sub> và port C gồm PC<sub>0</sub>-PC<sub>7</sub>. Các port này có thể là các port input hoặc output tùy thuộc vào lệnh điều khiển, lệnh điều khiển do vi xử lý gởi tới lưu trong thanh ghi điều khiển để định cấu hình làm việc cho 8255A.

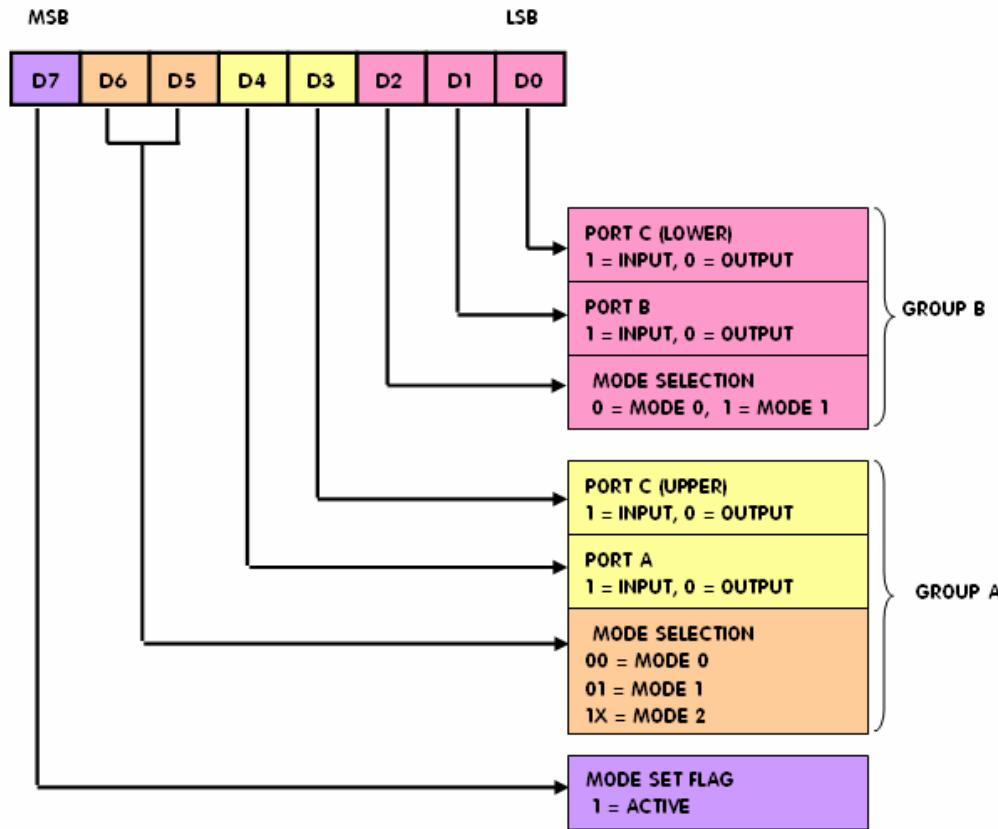
Các đường địa chỉ A<sub>1</sub>A<sub>0</sub> của 8255A dùng để lựa chọn các port và thanh ghi, A<sub>1</sub>A<sub>0</sub>=00<sub>2</sub> dùng để chọn Port A, A<sub>1</sub>A<sub>0</sub> = 01<sub>2</sub> dùng để chọn Port B, A<sub>1</sub>A<sub>0</sub> = 10<sub>2</sub> dùng để chọn Port C, A<sub>1</sub>A<sub>0</sub> = 11<sub>2</sub> dùng để chọn thanh ghi điều khiển.

Trong sơ đồ khối 8255A, các port I/O chia làm hai nhóm: nhóm A gồm port A và 4 bit cao của port C, nhóm B gồm port B và 4 bit thấp của port C. Để sử dụng các port của 8255A người lập trình phải gởi từ điều khiển ra thanh ghi điều khiển để 8255A định cấu hình cho các port đúng theo yêu cầu mà người lập trình mong muốn.

#### (b) Cấu trúc từ điều khiển của 8255A:

Trong 8255 có 2 từ điều khiển: Control Word và Bit Set/Reset.

Từ điều khiển Control Word có cấu trúc như hình 6-17:

**Hình 6-17. Cấu trúc từ điều khiển thứ nhất của IC 8255A.**

Do các port của 8255A được chia làm hai nhóm: nhóm A và nhóm B tách rời nhau trên từ điều khiển của 8255A cũng được chia làm hai nhóm.

**☐ Các bit  $D_2D_0$  dùng để định cấu hình cho nhóm B:**

- ◆ Bit  $D_0$  dùng để thiết lập 4 bit thấp của port C:

$D_0 = 0$  port C thấp là port xuất dữ liệu (output)

$D_0 = 1$  port C thấp là port nhập dữ liệu (input).

- ◆ Bit  $D_1$  dùng thiết lập port B:

$D_1 = 0$  port B là port xuất dữ liệu (output)

$D_1 = 1$  port B là port nhập dữ liệu (input).

- ◆ Bit  $D_2$  dùng thiết lập Mode điều khiển của nhóm B:

- $D_2 = 0$ : nhóm B hoạt động ở Mode 0.

- $D_2 = 1$ : nhóm B hoạt động ở Mode 1.

**☐ Các bit  $D_6D_5D_4D_3$  dùng để định cấu hình cho nhóm A:**

- ◆ Bit  $D_3$  dùng để thiết lập 4 bit cao của port C:

$D_3 = 0$  port C là port xuất dữ liệu (output)

$D_3 = 1$  port C là port nhập dữ liệu (input).

- ◆ Bit  $D_4$  dùng để thiết lập port A:

$D_4 = 0$  port A là port xuất dữ liệu (output)

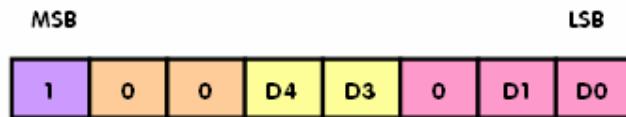
D4 = 1 port A là port nhập dữ liệu (input).

- ◆ Bit D<sub>6</sub>D<sub>5</sub> dùng thiết lập Mode điều khiển của nhóm A:

- D<sub>6</sub>D<sub>5</sub> = 00: nhóm A hoạt động ở Mode 0.
- D<sub>6</sub>D<sub>5</sub> = 01: nhóm A hoạt động ở Mode 1.
- D<sub>6</sub>D<sub>5</sub> = 1x: nhóm A hoạt động ở Mode 2.

#### **Các nhóm A, B làm việc ở cấu hình ở Mode 0:**

Từ điều khiển thiết lập nhóm A & B hoạt động ở Mode 0 như hình 6-18:

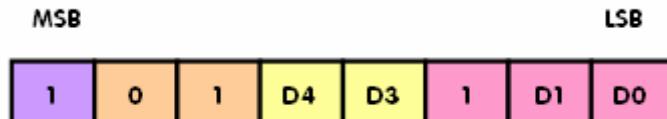


**Hình 6-18. Từ điều khiển khi 2 nhóm A, B làm việc ở mode 0.**

Ở Mode 0 các port A, port B, port C thấp và port C cao là các port xuất hoặc nhập dữ liệu độc lập. Do có 4 bit để lựa chọn nên có 16 từ điều khiển khác nhau cho 16 trạng thái xuất nhập của 4 port.

#### **Các nhóm A & B làm việc ở cấu hình ở Mode 1:**

Từ điều khiển nhóm A, B hoạt động ở Mode 1 như hình 6-19:



**Hình 6-19. Từ điều khiển khi 2 nhóm A, B làm việc ở mode 1.**

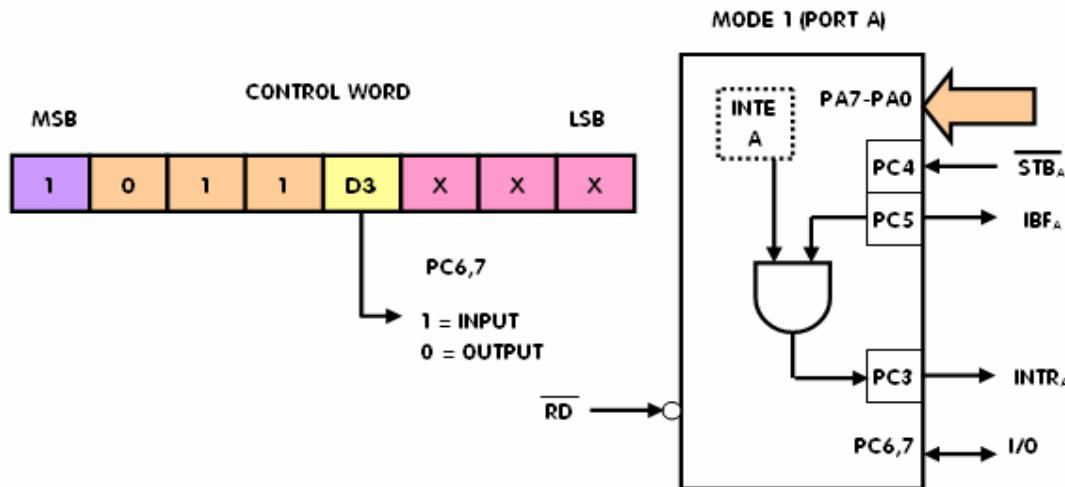
Ở Mode 1 các port A & B làm việc xuất nhập có chốt (Strobe I/O). Ở Mode này hai port A & B hoạt động độc lập với nhau và mỗi port có 1 port 4 bit điều khiển. Các port 4 bit điều khiển được hình thành từ 4 bit thấp và 4 bit cao của port C.

Khi 8255A được cấu hình ở Mode 1, thiết bị giao tiếp muốn 8255A nhận dữ liệu, thiết bị đó phải tạo ra tín hiệu yêu cầu 8255A nhận dữ liệu, ngược lại 8255A muốn gửi dữ liệu đến thiết bị khác, 8255A phải tạo ra tín hiệu yêu cầu thiết bị đó nhận dữ liệu, tín hiệu yêu cầu đó gọi là tín hiệu Strobe.

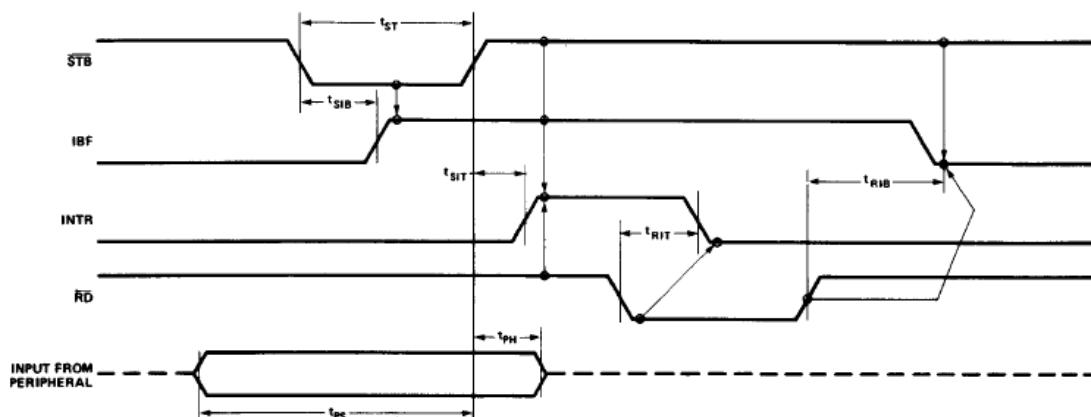
#### **□ Phân tích nhóm A làm việc ở cấu hình Mode 1**

- ◆ Port A được cấu hình là port nhập dữ liệu:

Chức năng của các đường tín hiệu được trình bày ở hình 6-20 và dạng sóng làm việc của các đường tín hiệu điều khiển như hình 6-21:



Hình 6-20. Port A của IC 8255A hoạt động ở mode 1 – nhận dữ liệu.



Hình 6-21. Dạng sóng của các đường tín hiệu điều khiển – port A input.

Các đường tín hiệu của port C trở thành các đường điều khiển của port A.

Bit PC<sub>4</sub> trở thành bit  $\overline{STB_A}$  (Strobe Input, tích cực mức thấp) dùng để nhận tín hiệu chốt từ “thiết bị gửi” để báo cho 8255A biết dữ liệu đã được gửi đến ở các ngõ vào PA<sub>7</sub> – PA<sub>0</sub>.

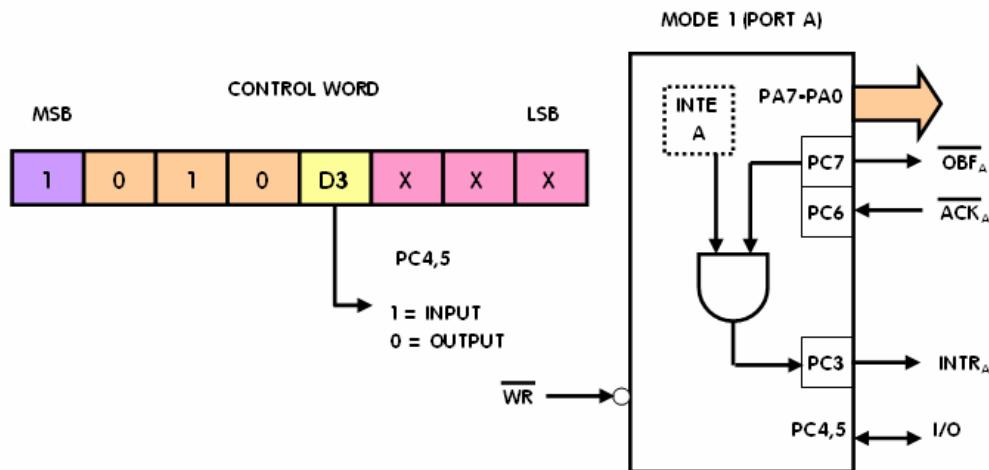
Bit PC<sub>5</sub> trở thành bit  $IBF_A$  (Input Buffer Full, tích cực mức cao), dùng để báo cho “thiết bị gửi” biết bộ đệm đã đầy dừng gởi byte tiếp theo.

Bit PC<sub>3</sub> trở thành bit  $INTR_A$  (Interrupt Request, tích cực ở mức cao), bit này có mức Logic 1 khi hai bit  $\overline{STB_A} = 1$ ,  $IBF_A = 1$  và bit  $INTE_A$  (Interrupt Enable) ở bên trong 8255A bằng 1. Bit  $INTE_A$  được thiết lập mức logic 1 hay 0 do từ điều khiển Set/Reset của 8255A. Ở hình trên, bit  $INTE_A = 1$  dùng để cho phép tín hiệu  $IBF_A$  xuất hiện tại ngõ ra  $INTE_A$  của cổng AND. Tín hiệu  $INTR_A$  được kết nối với ngõ vào ngắn của vi xử lý để báo cho vi xử lý biết: dữ liệu mới đã xuất hiện ở port A yêu cầu vi xử lý thực hiện chương trình phục vụ ngắn để nhận dữ liệu tại port A và xóa yêu cầu ngắn để điều khiển tín hiệu  $IBF_A$  về mức 0 sẵn sàng nhận byte tiếp theo.

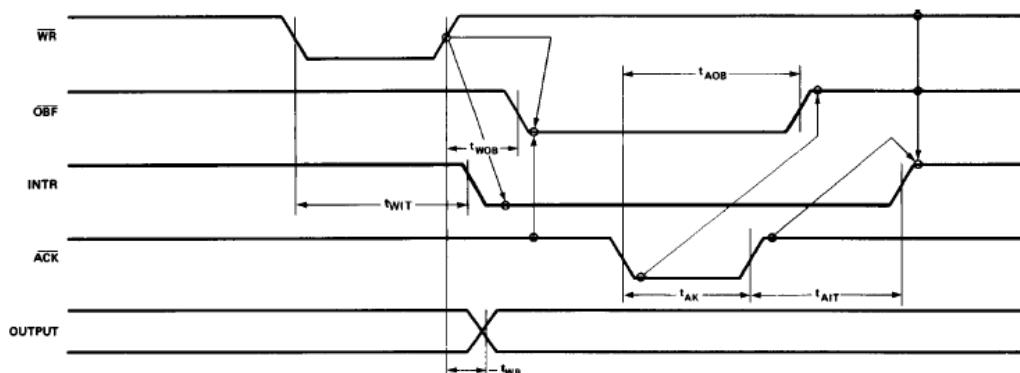
Các bit còn lại của port C: PC<sub>6</sub>, PC<sub>7</sub> là các bit xuất/nhập bình thường tùy thuộc vào bit D<sub>3</sub> trong từ điều khiển. Các bit **xxx** được dùng để thiết lập cho nhóm B.

◆ Port A được cấu hình là port xuất dữ liệu:

Chức năng của các đường tín hiệu được trình bày ở hình 6-22 và dạng sóng làm việc của các đường tín hiệu điều khiển như hình 6-23:



Hình 6-22. Port A của IC 8255A hoạt động ở mode 1 – xuất dữ liệu.



Hình 6-23. Dạng sóng của các đường tín hiệu điều khiển – port A output.

Bit PC<sub>7</sub> trở thành bit  $\overline{OBF}_A$  (Output Buffer Full, tích cực mức thấp), khi có dữ liệu từ vi xử lý gởi ra port A, tín hiệu  $\overline{OBF}_A$  sẽ yêu cầu thiết bị bên ngoài nhận dữ liệu.

Bit PC<sub>6</sub> trở thành bit  $\overline{ACK}_A$  (Acknowledge Input, tích cực mức thấp) thiết bị nhận dữ liệu dùng tín hiệu này để báo cho 8255A biết tín hiệu đã được nhận và sẵn sàng nhận dữ liệu tiếp theo.

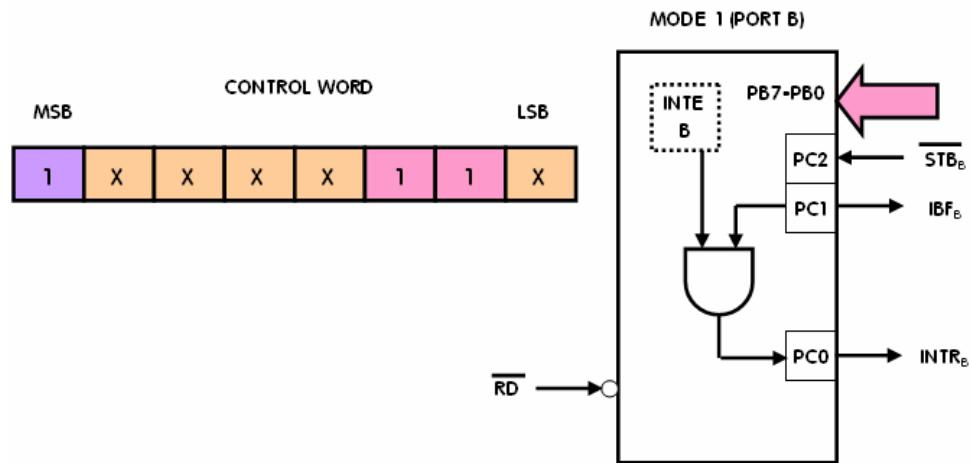
Bit PC<sub>3</sub> trở thành bit  $INTR_A$  (Interrupt Request, tích cực mức cao), bit này có mức logic khi hai bit  $\overline{OBF}_A = 1$ ,  $\overline{ACK}_A = 1$  và bit INTE<sub>A</sub> (Interrupt Enable) ở bên trong 8255A ở mức 1. Tín hiệu  $INTR_A$  tác động đến ngõ vào ngắn của vi xử lý để báo cho vi xử lý biết: thiết bị bên ngoài đã nhận xong dữ liệu ở port A và thực hiện gởi byte dữ liệu tiếp theo.

Các bit còn lại của port C: PC<sub>4</sub>, PC<sub>5</sub> là các bit xuất/nhập bình thường tùy thuộc vào bit D<sub>3</sub> trong từ điều khiển. Các bit xxxx dùng để thiết lập cho nhóm B.

#### □ Phân tích nhóm B làm việc ở cấu hình Mode 1:

- ◆ Port B được cấu hình là port nhập dữ liệu :

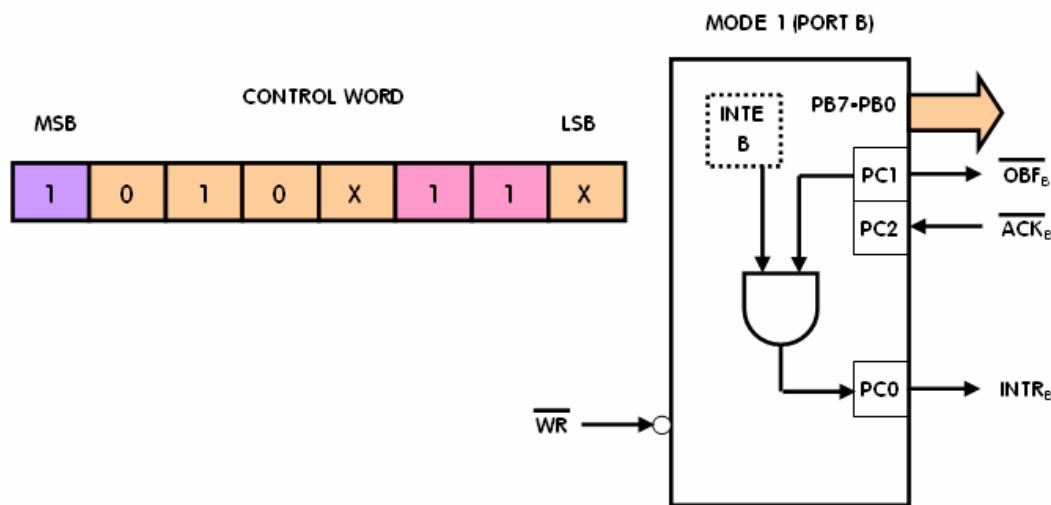
Chức năng của các đường tín hiệu được trình bày ở hình 6-24:

**Hình 6-24. Port B của IC 8255A hoạt động ở mode 1 – nhận dữ liệu.**

Chức năng của các bit điều khiển giống như nhóm A hoạt động ở Mode 1.

- ♦ Port B được cấu hình là port xuất dữ liệu:

Chức năng của các đường tín hiệu được trình bày ở hình 6-25:

**Hình 6-25. Port B của IC 8255A hoạt động ở mode 1 – xuất dữ liệu.**

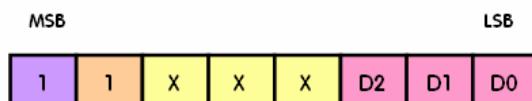
Chức năng của các bit điều khiển giống như nhóm A hoạt động ở Mode 1.

Các bit  $\times \times \times$  được dùng thiết lập cho nhóm A, bit  $D_0$  không có tác dụng trong trường hợp cả hai nhóm cùng làm việc ở Mode 1.

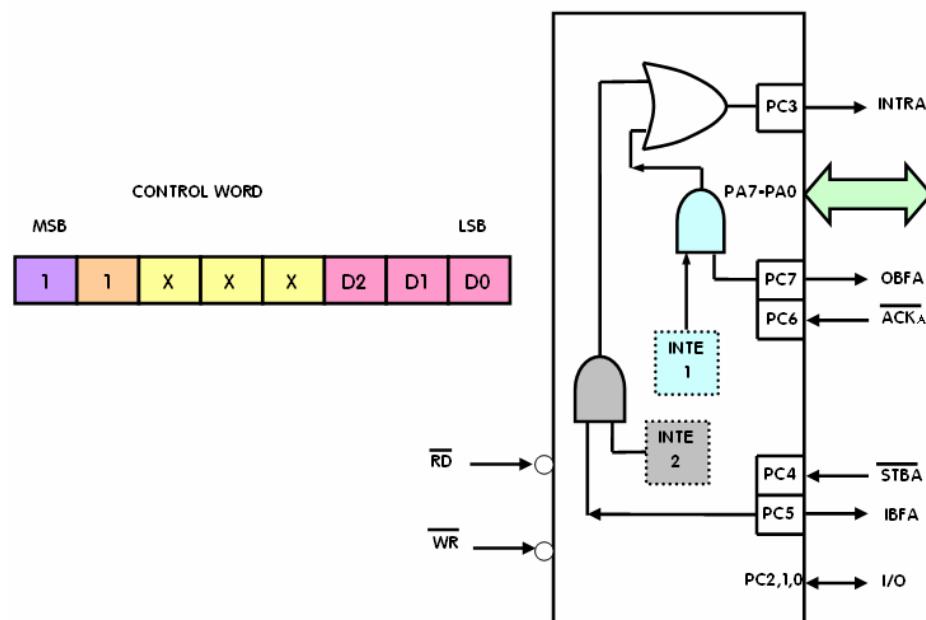
#### Các nhóm A làm việc ở cấu hình Mode 2:

Mode 2 là kiểu hoạt động *Stable Bi-directional IO*, khác với Mode 1 là port có thể xuất – nhập dữ liệu.

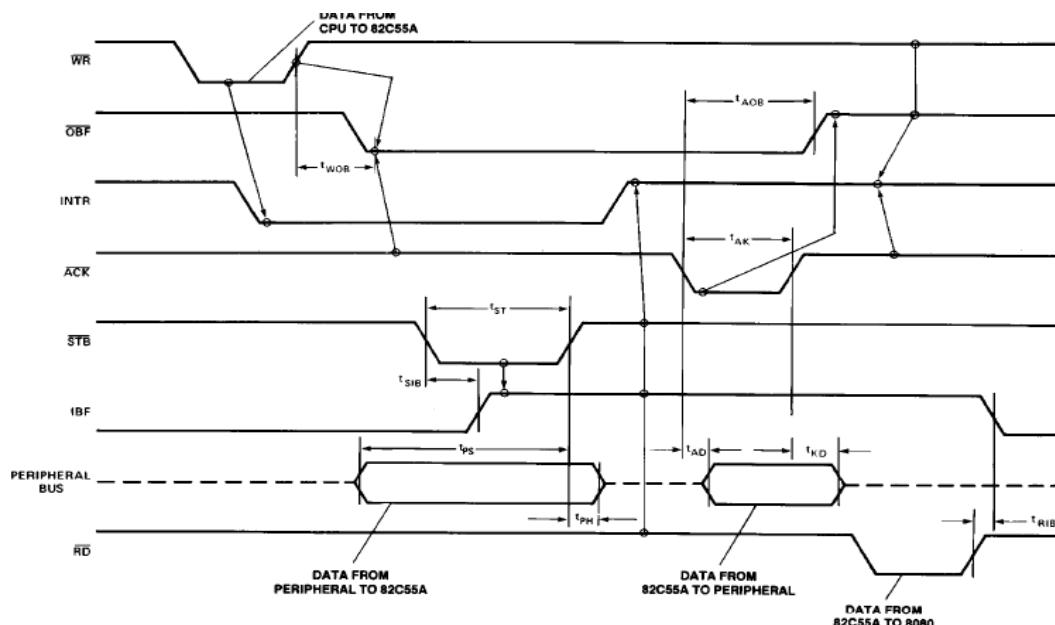
Tùy điều khiển khi hai nhóm A hoạt động ở Mode 2 như hình 6-26:

**Hình 6-26. Tùy điều khiển hoạt động ở mode 2 của nhóm A.**

Chức năng của các đường tín hiệu được trình bày ở hình 6-27 và dạng sóng làm việc của các đường tín hiệu điều khiển như hình 6-28:



Hình 6-27. Nhóm A của IC 8255A hoạt động ở mode 2.



Hình 6-28. Dạng sóng của các đường tín hiệu điều khiển nhóm A.

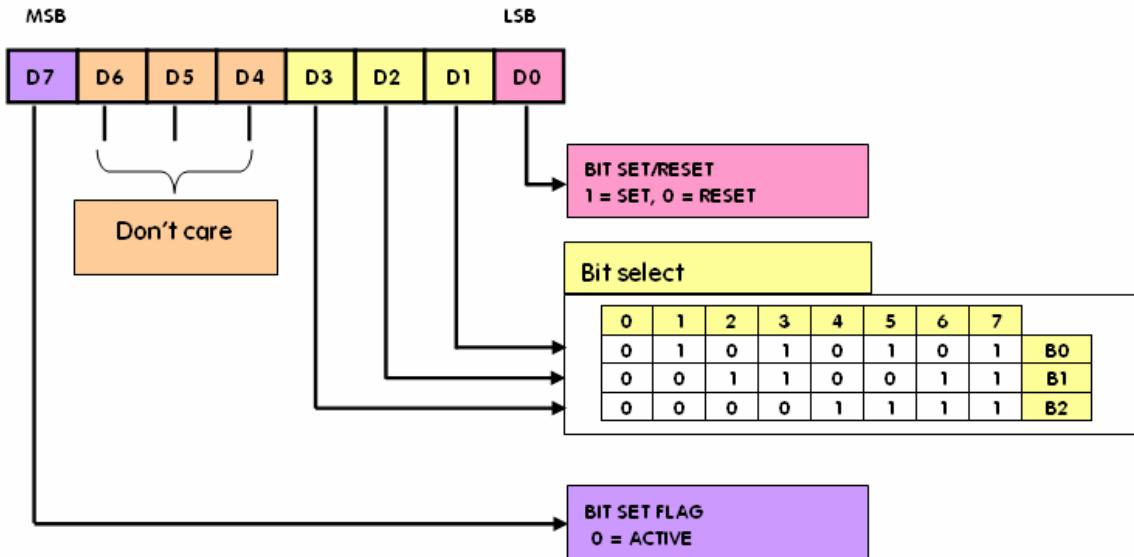
Các đường tín hiệu của port C trở thành các đường điều khiển của port A.

Bit  $PC_7$  trở thành bit  $OBF_A$ ,  $PC_6$  trở thành bit  $\overline{ACK}_A$ ,  $PC_4$  trở thành bit  $\overline{STB}_A$ ,  $PC_5$  trở thành  $IBF_A$  và bit  $PC_3$  trở thành bit  $INTR_A$ . Chức năng của các đường tín hiệu giống như Mode 1, chỉ khác là tín hiệu ngõ ra  $INTR_A = 1$ ,  $INTE1 = 1$  hoặc  $IBF_A = 1$ ,  $INTE2 = 1$ .

Các bit  $PC_{2,1,0}$  còn lại có thể là các bit I/O tùy thuộc vào bit điều khiển của nhóm B.

**Chú ý** khi nhóm A làm việc ở Mode 2, nhóm B chỉ được phép hoạt động ở Mode 0.

Cấu hình của từ điều khiển Set/Reset bit INTE khi 8255A hoạt động ở Mode 1 hoặc Mode 2 được trình bày ở hình 6-29:

**Hình 6-29. Từ điều khiển thứ 2 của IC 8255A.**

Cấu hình này còn cho phép Set/Reset từng bit của port C. Từ điều khiển này khác với từ điều khiển cấu hình là bit D7 = 0.

Bit D0 dùng để Set/Reset bit INTE, khi D0 = 1 thì INTE = 1 (cho phép ngắn), khi D0 = 0 thì INTE = 0 (không cho phép ngắn). Ba bit D1, D2, D3 dùng để chọn một bit của port C, gán mức logic của bit D0 cho 1 bit của port C đã chọn.

Trong thực tế port A và port B thường được cấu hình với nhiều Mode khác nhau. Ví dụ nhóm A hoạt động ở Mode 2 nhóm B làm việc ở Mode 0.

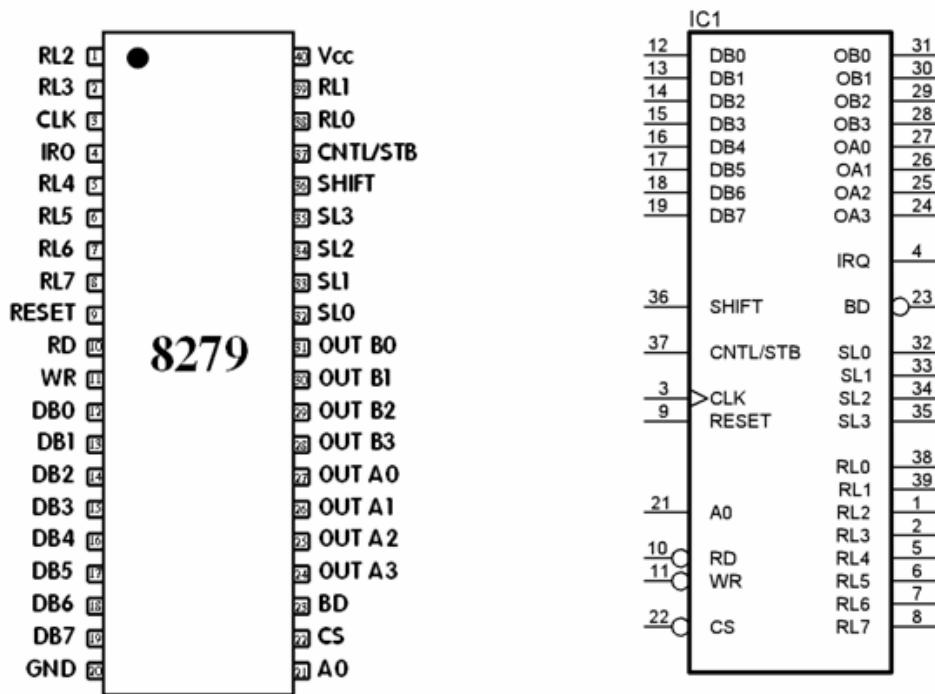
## 2. IC NGOẠI VI LẬP TRÌNH 8279:

IC ngoại vi 8279 là IC chuyên dùng để quét led 7 đoạn và quét bàn phím có cấu trúc như sau:

- IC 8279 có thể hoạt động ở mode quét bàn phím.
- IC 8279 có thể hoạt động ở mode các cảm biến.
- Hoạt động ở kiểu ngõ vào có chốt.
- Bộ nhớ FIFO có thể lưu 8 byte mã phím nhấn.
- Có chức năng khóa 2 phím hoặc quét vòng N phím có chống dội phím.
- Có thể hiển thị quét 2 bộ quét 4 đường, 8 đường hoặc 16 đường.
- Quét hiển thị 8 led 7 đoạn hoặc 16 led 7 đoạn.
- Có các lệnh lập trình.
- Có thể lập trình bộ nhớ Ram chứa dữ liệu hiển thị 16 byte với lối vào trái hoặc phải.
- Có thể lập trình thời gian quét.
- Có thể tạo tín hiệu ngắn khi có nhấn phím.

### (a) Sơ đồ chân và sơ đồ khối của 8279 :

IC 8279 có 40 chân như hình 6-30:



Hình 6-30. Sơ đồ chân và sơ đồ logic của IC 8279.

8279 có một bộ nhớ RAM 16 byte dùng để hiển thị và một bộ nhớ RAM dùng để chứa mã của phím nhấn. Trong 8279 có một thanh ghi đệm dữ liệu, khi dữ liệu được vi xử lý gửi tới thì nó được chuyển tới thanh ghi này. Ngoài ra 8279 còn có thanh ghi trạng thái để chỉ trạng thái của 8279 tại một thời điểm.

Tên các chân của IC 8279 như bảng 6-17:

Tên tín hiệu	Hướng tín hiệu	Chức năng
A0	I	Address
DB7 – DB0	I/O	Data Bus (Bi-direction)
CLK	I	Clock Input
RESET	I	Reset Input
CS\	I	Chip Select
RD\	I	Read Input
WR\	I	Write Input
IRQ	O	Interrupt Request Input
SLO-SL3	O	Scan Lines
RLO-RL7	I	Return Lines
SHIFT	I	Shift Input
CNTL/STB	I	Control/Strobe Input
OUT A3-0	O	Display (A) Output
OUT B3-0	O	Display (B) Output
BD\	O	Blank Display Output

Bảng 6-17. Tên các chân của IC 8279.

IC 8279 kết nối với vi xử lý thông qua 3 bus gồm: bus dữ liệu D7-D0, bus địa chỉ có 1 đường A0, bus điều khiển  $\overline{RD}$ ,  $\overline{WR}$ ,  $\overline{CS}$ , RESET, IRQ, CLK giống như 8255A.

Tín hiệu chọn  $\overline{CS}$  được kết nối với một ngõ ra nào đó của IC giải mã địa chỉ IO hoặc IC giải mã địa chỉ bộ nhớ. Nếu xem IC 8279 là một bộ nhớ thì bộ nhớ này có hai ô nhớ.

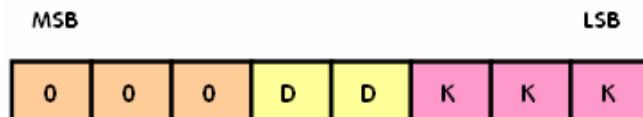
**(b) Cấu trúc từ điều khiển của 8279:**

IC 8279 có 1 đường địa chỉ A0 có chức năng lựa chọn như sau:

- A0 = 0: IC 8279 xem dữ liệu từ vi xử lý gởi đến để chọn dữ liệu hiển thị.
- A0 = 1: IC 8279 xem dữ liệu từ vi xử lý gởi đến là dữ liệu của lệnh điều khiển 8279.

**◆ Lệnh Keyboard/Display Mode Set:**

Từ điều khiển như hình 6-31:



**Hình 6-31. Từ điều khiển keyboard/Display mode set của IC 8279.**

Trong đó 2 bit DD dùng để thiết lập Mode hiển thị, 3 bit KKK dùng để thiết lập Mode quét bàn phím.

Chức năng của 2 bit DD:

Hai bit DD	Chức năng
00	hiển thị 8 ký tự – lối vào trái.
01	hiển thị 16 ký tự – lối vào trái.
10	hiển thị 8 ký tự – lối vào phải.
11	hiển thị 16 ký tự – lối vào phải.

**Bảng 6-18. Các mod hiển thị khác nhau của 8279.**

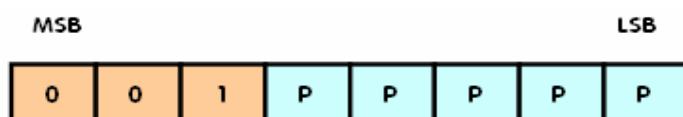
Chức năng của 3 bit KKK:

Ba bit KKK	Chức năng
000	encode scan keyboard – 2 key lockout
001	decode scan keyboard – 2 key lockout
010	encode scan keyboard – N key rollover
011	decode scan keyboard – N key rollover
100	encode scan sensor matrix
101	decode scan sensor matrix
110	strobe input, encode display scan
111	strobe input, decode display scan

**Bảng 6-19. Các mod quét ma trận phím khác nhau của 8279.**

**◆ Lệnh Program clock:**

Từ điều khiển như hình 6-32:

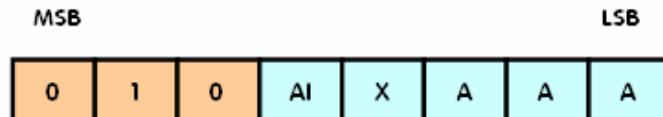


**Hình 6-32. Từ điều khiển Program clock của IC 8279.**

Lệnh này có chức năng chia tần số xung clock ở ngõ vào CLK ở chân số 3, các bit PPPPP dùng để xác định số chia nằm trong khoảng từ 1 đến 31, tần số hoạt động của mạch quét hiển thị và chống dội của 8279 thường là 100kHz, nếu tần số ở ngõ vào là 2MHz thì chia cho 20 ta được 100kHz, khi đó các bit PPPPP có giá trị là 10100.

**◆ Lệnh Read FIFO/sensor RAM:**

Từ điều khiển như hình 6-33:

**Hình 6-33. Từ điều khiển Red FIFO/sensor RAM của IC 8279.**

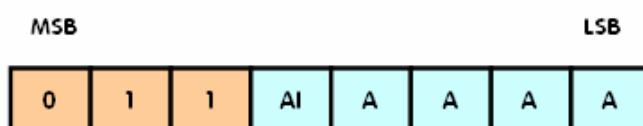
IC 8279 có 8 byte RAM bên trong để chứa mã của phím ấn hay mã của các sensor được phân biệt bởi 3 bit địa chỉ AAA. Trước khi đọc dữ liệu của ô nhớ nào thì ta phải thiết lập từ điều khiển này cho IC 8279 rồi mới tiến hành đọc dữ liệu.

Ta có thể thay đổi con trỏ quản lý 8 byte này bằng cách thay đổi giá trị của các bit AAA trong từ điều khiển. Bộ nhớ thuộc kiểu FIFO.

Bit AI (Automatically increment): nếu ở mức 1 thì sau khi đọc 1 byte thì con trỏ tăng lên 1 để có thể đọc byte kế tiếp. Nếu AI = 0 con trỏ sẽ không thay đổi, muốn đọc byte kế tiếp thì phải thay đổi địa chỉ bằng từ điều khiển này.

◆ Lệnh Read Display RAM:

Từ điều khiển như hình 6-34:

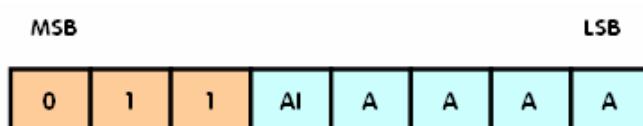
**Hình 6-34. Từ điều khiển Display RAM của IC 8279.**

IC 8279 có 16 byte RAM bên trong do con trỏ 4 bit AAAA quản lý, 16 byte RAM này dùng để chứa dữ liệu cần hiển thị do vi xử lý gởi tới, vi xử lý có thể đọc lại dữ liệu của 1 ô nhớ bất kỳ trong vùng nhớ RAM này bằng cách điều chỉnh các bit AAAA tương ứng trước khi đọc dữ liệu.

Bit AI (Automatically Increment): ở mức 1 làm con trỏ tự động tăng lên ô nhớ kế để sẵn sàng để đọc byte dữ liệu của ô nhớ kế. Nếu AI = 0 con trỏ sẽ không thay đổi.

◆ Lệnh End Interrupt:

Từ điều khiển như hình 6-35:

**Hình 6-35. Từ điều khiển End Interrupt của IC 8279.**

Bit E = 1 sẽ xóa ngắt IRQ về mức Logic 0.

◆ Lệnh Write Display RAM:

Từ điều khiển như hình 6-36:

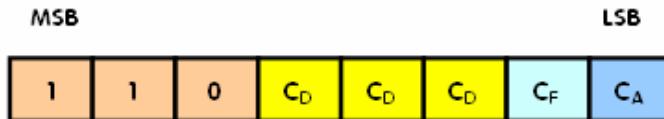
**Hình 6-36. Từ điều khiển Write Display RAM của IC 8279.**

IC 8279 có 16 byte RAM bên trong do con trỏ 4 bit AAAA quản lý, 16 byte RAM này dùng để chứa dữ liệu cần hiển thị do vi xử lý gởi tới, để ghi dữ liệu vào ô nhớ nào trong vùng nhớ RAM này ta có thể điều chỉnh các bit AAAA tương ứng.

Bit AI (Automatically Increment): ở mức 1 làm con trỏ tự động tăng lên ô nhớ kế để sẵn sàng lưu byte dữ liệu kế. Nếu AI = 0 con trỏ sẽ không thay đổi, do đó, byte dữ liệu sau sẽ ghi đè lên byte dữ liệu trước đó.

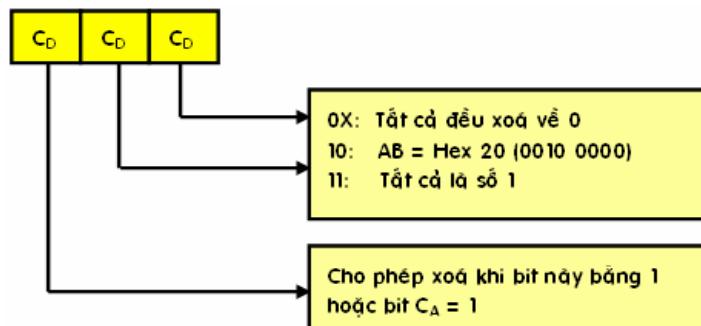
#### ◆ Lệnh Clear:

Từ điều khiển như hình 6-37:



**Hình 6-37. Từ điều khiển Clear của IC 8279.**

Những Bit  $C_D$  trong lệnh này dùng để xóa tất cả các hàng của bộ nhớ RAM hiển thị sang mã nhấp nháy cho phép lựa chọn như hình 6-38:



**Hình 6-38. Ba bit  $C_D$  của từ điều khiển Clear.**

Trong suốt thời gian đang xoá bộ nhớ RAM hiển thị thì ta không được ghi dữ liệu vào (thời gian xoá khoảng 160μs). Bit MSB của từ trạng thái FIFO ở mức 1 trong khoảng thời gian xoá và sẽ trở về mức 0 sau khi xoá xong. Người lập trình có thể kiểm tra bit này để biết khi nào xoá xong để gởi dữ liệu ra hiển thị.

Nếu như Bit  $C_F$  tích cực ( $C_F = 1$ ), từ trạng thái FIFO sẽ bị xóa và ngõ ra Interrupt bị Reset.

Bit  $C_A$  có chức năng xóa tất cả các bit, bit này còn kết hợp với bit  $C_D$  và  $C_F$ . Bit  $C_D$  để xóa mã trên RAM hiển thị và nó cũng xóa luôn trạng thái FIFO.

#### (c) Ứng dụng IC 8279:

IC 8279 là chuyên về giải mã hiển thị Led 7 đoạn và quét phím theo nhiều phương pháp khác nhau.

Dữ liệu cần hiển thị từ vi xử lý gởi đến sẽ được lưu trong 16 byte RAM bên trong được gọi là bộ nhớ hiển thị. Các tín hiệu này lần lượt gởi ra trên 8 đường tín hiệu từ A[3÷0] đến B[3÷0].

Các đường tín hiệu SL[3÷0] dùng để quét, dữ liệu trên đường này có thể được thiết lập theo kiểu mã hoá (Encode) hoặc giải mã (Decode) tùy thuộc vào kiểu thiết kế phần cứng. Các đường này có hai chức năng vừa hiển thị vừa quét giải mã bàn phím.

Các đường tín hiệu RL[7÷0] là các đường tín hiệu input kết hợp với các đường tín hiệu quét SL[3÷0] tạo thành ma trận phím, phím được nhấn sẽ làm cho một ngõ vào RL xuống mức 0, kết hợp với các đường tín hiệu quét sẽ cho biết mã của phím nhấn. **Chú ý** các đường SL[3÷0] phải ở chế độ Decode.

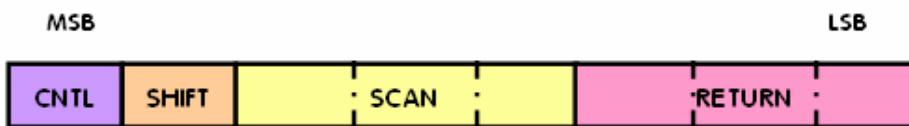
Các ngõ vào SHIFT và CNTL được dùng để mở rộng các phím tổ hợp.

Số lượng các phím kết nối có thể lên tới 64 phím rời.

IC 8279 gởi dữ liệu trong vùng nhớ RAM hiển thị ra các Led và tự động quét bàn phím để tìm phím bị tác động và tự chống dội, sau đó thiết lập mã của phím bị nhấn, trước khi sử dụng IC 8279 người sử dụng phải thiết lập các từ điều khiển gởi ra IC 8279 khi bit địa chỉ A0 = 1. Dữ liệu đọc vào hoặc gởi ra IC 8279 khi bit địa chỉ A0 = 0.

Khi có phím bị nhấn, IC 8279 sẽ tự động chống dội sau 10,3 ms và kiểm tra lại một lần nữa để xem phím đó có bị nhấn hay không, nếu còn thì IC 8279 sẽ thiết lập mã cho phím này và lưu trữ mã của phím vào bộ nhớ RAM bên trong. Sau đó sẽ báo cho vi xử lý biết đã có một phím tác động và yêu cầu vi xử lý nhận mã của phím này bằng cách tác động đến tín hiệu ngắt IRQ. Tín hiệu IRQ được kết nối đến một ngõ vào ngắt của vi xử lý và chương trình phục vụ cho ngắt này là chương trình ngắt phím. Nhiệm vụ của vi xử lý là đọc mã của phím bị nhấn vào để xử lý và Reset ngắt của IC 8279 để chuẩn bị cho phím tiếp theo.

Khung mã dữ liệu của phím bị nhấn như hình 6-39:



**Hình 6-39. Cấu trúc mã phím của IC 8279.**

Giải thích khung mã phím: IC 8279 tạo ra một ma trận phím  $8 \times 8 = 64$  phím và 2 phím hoạt động độc lập là CNTL và SHIFT kết nối với 2 ngõ vào của IC 8279.

Tổ hợp  $8 \times 8$  bao gồm 8 đường RL[7÷0] và 3 đường SL[2÷0] (đường SL3 chỉ phục vụ quét hiển thị) sau khi qua mạch giải mã 3 sang 8 sẽ được 8 đường.

Nếu không sử dụng phím tổ hợp thì 2 ngõ vào CNTL và SHIFT của IC 8279 phải nối mass (0V) – khi đó 2 bit CNTL và SHIFT trong khung mã phím sẽ bằng 0.

Khi nhấn 1 phím thì IC 8279 sẽ biết được phím nhấn tương ứng với trạng thái quét nào của 3 bit SL[2÷0] và chính là 3 bit “**SCAN**”, phím nhấn trên sẽ làm thay đổi trạng thái 1 trong 8 đường RL qua mã hóa 8 sang 3 sẽ tạo ra mã nhị phân 3 bit chính là 3 bit “**RETURN**” trong khung mã dữ liệu.

Nếu không dùng phím CNTL và SHIFT thì tổ hợp 6 bit sẽ tương ứng với 64 phím khác nhau tương ứng với số hex [00H÷3FH], khi nhấn thêm phím tổ hợp gồm phím SHIFT và 1 trong 64 phím thì sẽ có 64 mã khác tương ứng với số hex [40H÷7FH], khi nhấn thêm phím tổ hợp gồm phím CNTL và 1 trong 64 phím thì sẽ có 64 mã khác tương ứng với số hex [80H÷BFH], cuối cùng khi nhấn thêm phím tổ hợp gồm phím CNTL + SHIFT và 1 trong 64 phím thì sẽ có 64 mã khác tương ứng với số hex [C0H÷FFH]. Vậy tổng cộng có tất cả 256 tổ hợp phím.

#### IV. THIẾT KẾ HỆ THỐNG VI XỬ LÝ:

Một mình vi xử lý không thể hoạt động được mà phải kết hợp với bộ nhớ ROM, RAM để lưu trữ chương trình và dữ liệu, phải kết hợp với IC ngoại vi để giao tiếp xuất nhập tín hiệu điều khiển – gọi là hệ thống vi xử lý hay kit vi xử lý hay hệ thống điều khiển vi xử lý.

Trong phần này trình bày cách thiết kế một hệ thống điều khiển dùng vi xử lý thông qua các yêu cầu điều khiển.

##### 1 BÀI THIẾT KẾ SỐ 1

**Yêu cầu:** “ Hãy thiết kế 1 hệ thống dùng vi xử lý 8 bit 8085A giao tiếp với 2 loại bộ nhớ có dung lượng 8Kbyte EPROM và 8 Kbyte RAM. Giao tiếp với 2 IC ngoại vi 8255A và 1 IC 8279 để điều khiển hiển thị 8 led 7 đoạn và ma trận bàn phím gồm 24 phím”.

#### Phân tích yêu cầu

Hệ thống sử dụng vi xử lý 8085A có 16 đường địa chỉ nên có thể giao tiếp với 64Kbyte bộ nhớ. Trong hệ thống này chỉ sử dụng 16Kbyte là đáp ứng được.

Do vi xử lý 8085A khi reset thì địa chỉ thanh ghi PC = 0000H nên bộ nhớ EPROM luôn được thiết kế bắt đầu tại địa chỉ 0000H.

Vi xử lý 8085A cho phép giao tiếp với IC ngoại vi thông qua 8 bit địa chỉ thấp và đường điều khiển IO/M nên có thể giao tiếp với 256 ô nhớ ngoại vi IO.

#### Chọn linh kiện

Sau khi phân tích xong ta chọn loại bộ nhớ EPROM loại 2764 có dung lượng 8Kbyte và Ram 6264 có dung lượng 8Kbyte.

#### Thiết kế giải mã địa chỉ :

Tổng IC giao tiếp với vi xử lý là 5 IC gồm 2 IC nhớ và 3 IC ngoại vi. Do điều kiện bài thiết kế không bắt buộc nên ta có thể thiết kế giải mã như sau:

Lập bảng địa chỉ bộ nhớ của vi xử lý như sau:

TB	NHỊ PHÂN																HEX
	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
EPROM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
2764	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
RAM	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
6264	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF

Bảng 6-20. Bảng địa chỉ bộ nhớ.

Trong bảng địa chỉ nhớ bắt đầu là IC nhớ 2764 có 13 đường địa chỉ A[12÷0] nên trong bảng địa chỉ của vi xử lý thì ô nhớ có địa chỉ bắt đầu tương ứng với 13 đường địa chỉ A[12÷0] đều bằng 0. Ô nhớ cuối cùng có địa chỉ tương ứng với 13 đường địa chỉ A[12÷0] đều bằng 1. Địa chỉ tính theo số hex của 8Kbyte EPROM là [0000H÷1FFFH].

Tiếp theo là bộ nhớ RAM 6264 cũng có 13 đường địa chỉ A[12÷0] nên trong bảng địa chỉ của vi xử lý thì ô nhớ có địa chỉ bắt đầu tương ứng với 13 đường địa chỉ A[12÷0] đều bằng 0. Ô nhớ cuối cùng có địa chỉ tương ứng với 13 đường địa chỉ A[12÷0] đều bằng 1. Để phân biệt 2 bộ nhớ này độc lập với nhau thì phải dùng các đường địa chỉ còn lại từ A[13÷15]. Trong bài này tác giả sử dụng đường địa chỉ A13 bằng cách cho A13 = 1 đối với bộ nhớ RAM. Địa chỉ tính theo số hex của 8Kbyte RAM là [2000H÷3FFFH].

Với yêu cầu này thì ta dùng đường địa chỉ A13 để điều khiển chip select phân biệt 2 IC nhớ.

Lập bảng địa chỉ IO của vi xử lý như sau:

TB	NHỊ PHÂN								HEX
	A7	A6	A5	A4	A3	A2	A1	A0	

<b>8255A</b>	0	0	0	0	0	0	0	0	00
<b>1</b>	0	0	0	0	0	0	1	1	03
<b>8255A</b>	0	0	0	0	0	1	0	0	04
<b>2</b>	0	0	0	0	0	1	1	1	07
<b>8279</b>	0	0	0	0	1	0	0	0	08
<b>3</b>	0	0	0	0	1	0	0	1	09

**Bảng 6-21. Bảng địa chỉ IO.**

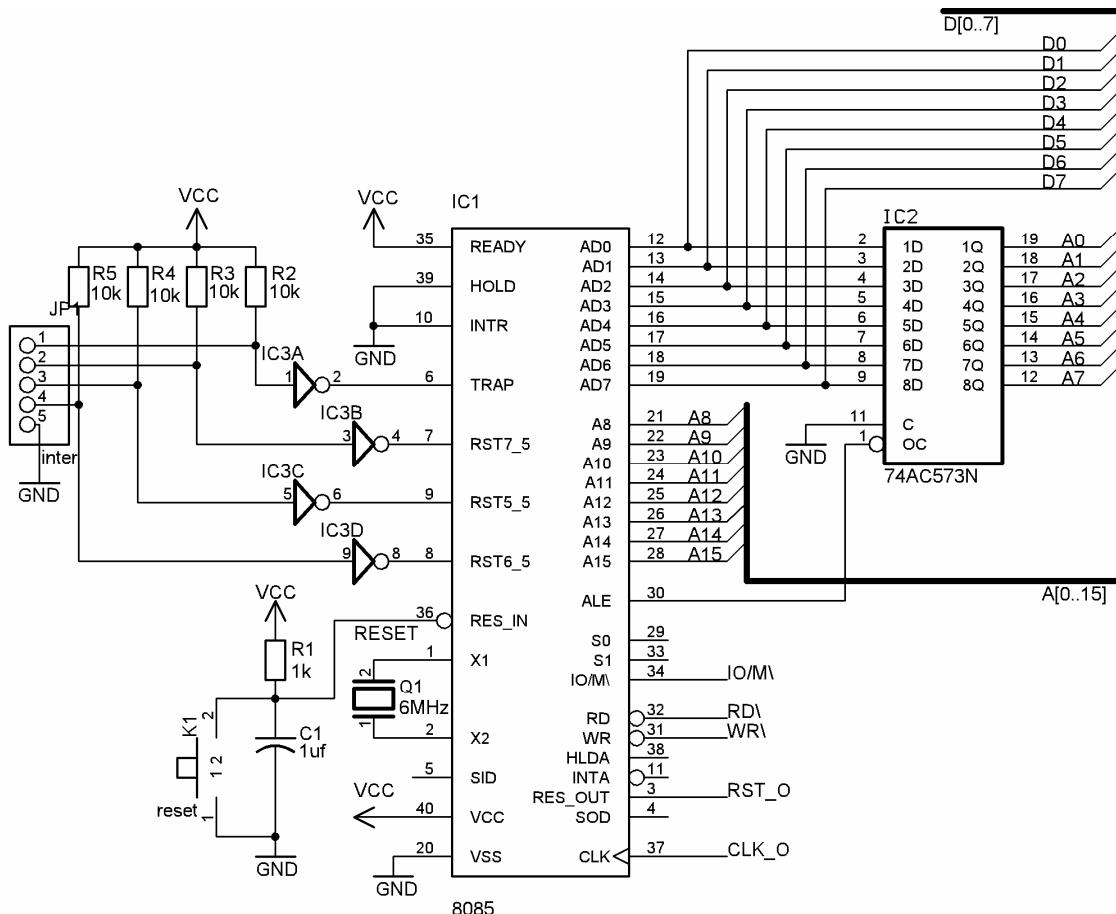
Trong bảng đồ nhớ bắt đầu là IC nhớ 8255A thứ 1, mỗi IC 8255 có 2 đường địa chỉ nên ô nhớ có địa chỉ bắt đầu tương ứng với 2 đường địa chỉ A[1÷0] đều bằng 0. Ô nhớ cuối cùng có địa chỉ tương ứng với 2 đường địa chỉ A[1÷0] đều bằng 1. Địa chỉ tính theo số hex của 8255A-1 là [00÷03].

Tiếp theo là IC 8255A thứ 2, ô nhớ có địa chỉ bắt đầu tương ứng với 2 đường địa chỉ A[1÷0] đều bằng 0. Ô nhớ cuối cùng có địa chỉ tương ứng với 2 đường địa chỉ A[1÷0] đều bằng 1. Địa chỉ tính theo số hex của 8255A-2 là [04÷07].

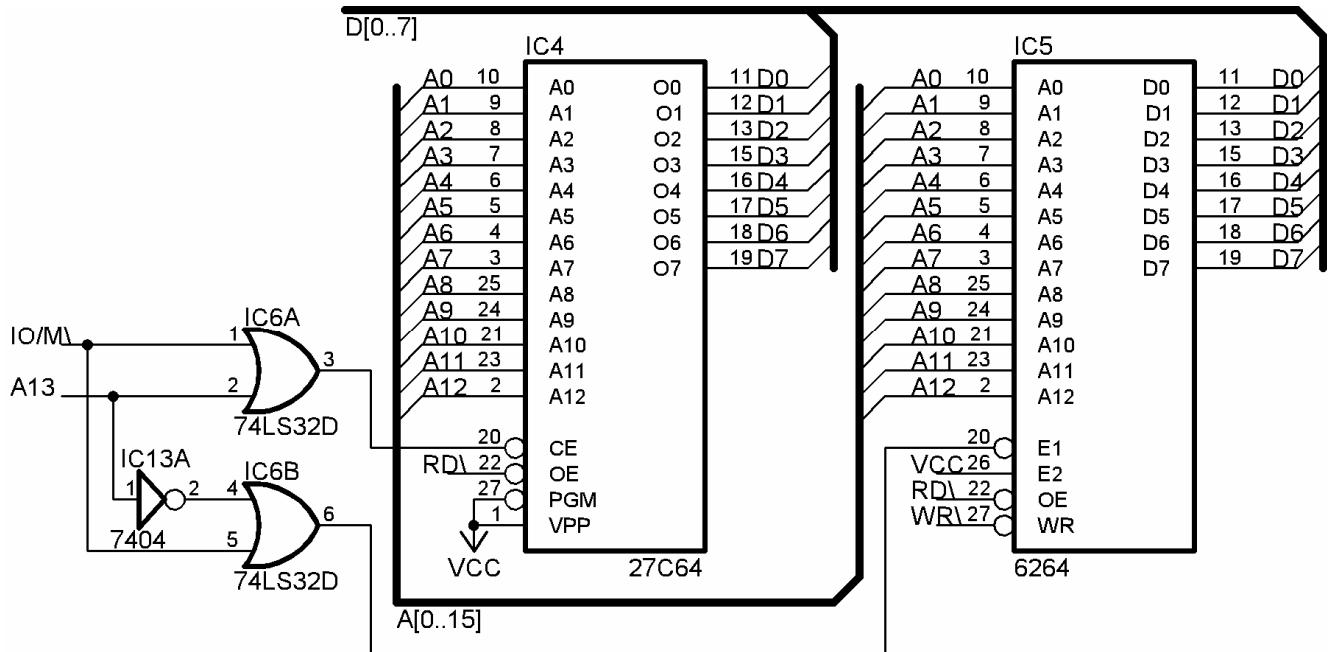
Tiếp theo là IC 8279 thứ 3, ô nhớ có địa chỉ bắt đầu tương ứng với 1 đường địa chỉ A[0] bằng 0. Ô nhớ cuối cùng có địa chỉ tương ứng với 1 đường địa chỉ A[0] bằng 1. Địa chỉ tính theo số hex của 8279-3 là [08÷09].

Trong bảng đồ IO thì phải dùng 2 đường địa chỉ A[3÷:2] để phân biệt 3 IC ngoại vi.

Sơ đồ nguyên lý của hệ thống như hình 6-40.

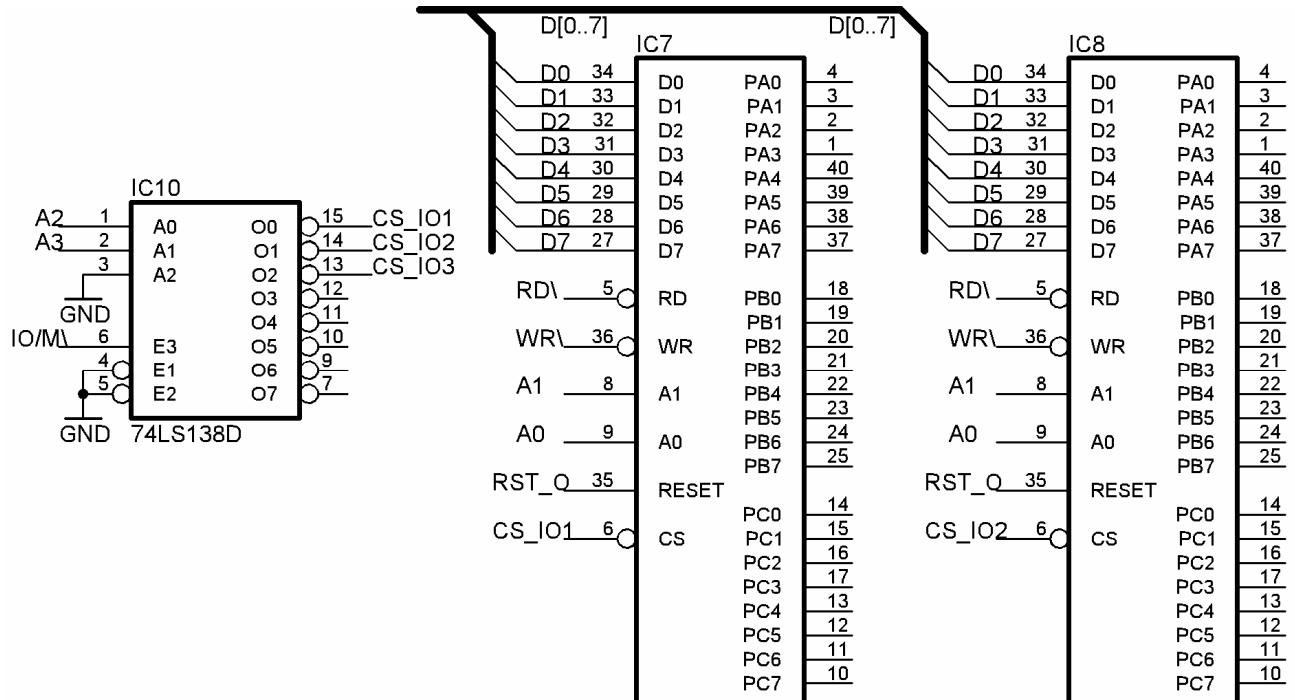
**Hình 6-40a. Sơ đồ nguyên lý của khối vi xử lý và IC chốt.**

Trong hệ thống sử dụng vi xử lý 8085A và IC chốt 74573 để tách địa chỉ và dữ liệu, kết quả được 8 đường dữ liệu D[7:0] và 16 đường địa chỉ A[15:0], các đường điều khiển  $IO/\bar{M}$ ,  $\bar{RD}$ ,  $\bar{WR}$ , RST\_O, CLK\_O.



Hình 6-40b. Sơ đồ nguyên lý của khối bộ nhớ với vi xử lý.

IC4 là Eprom 2764 và IC5 là SRAM 6264: 13 đường địa chỉ A[12:0] của các IC nhớ được kết nối với 13 đường địa chỉ A[12÷0] của vi xử lý, 8 đường dữ liệu D[7:0] của bộ nhớ được kết nối với 8 đường dữ liệu của vi xử lý, chân điều khiển đọc OE được kết nối với chân RD của vi xử lý, chân điều khiển ghi WR của SRAM 6264 được kết nối với WR của vi xử lý.



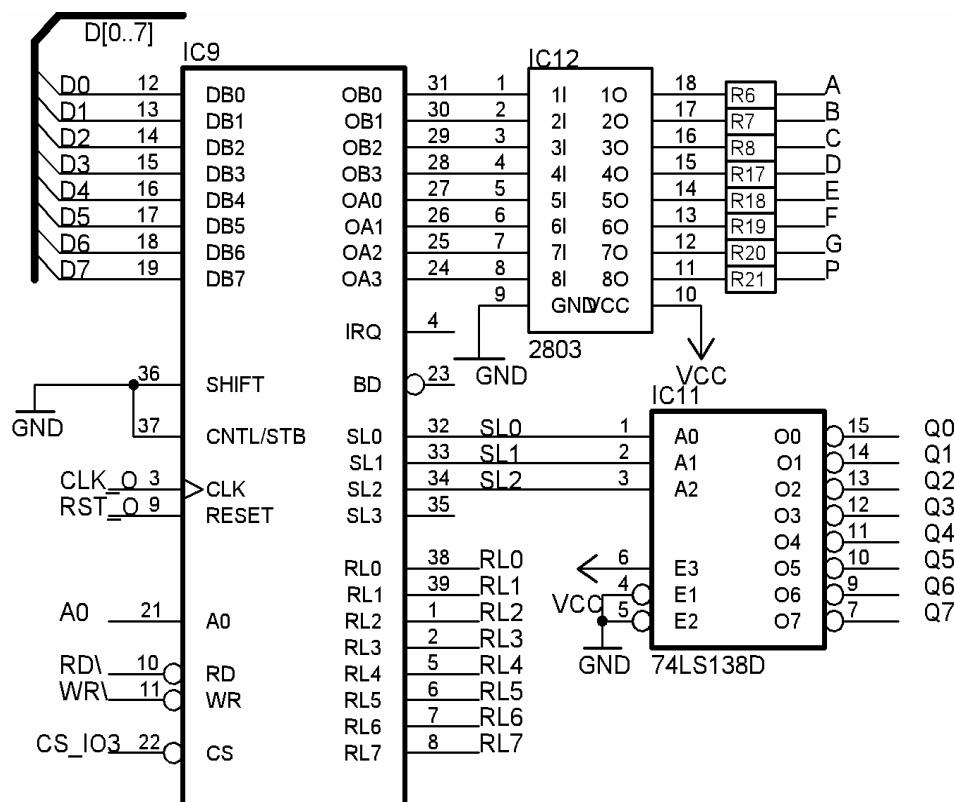
Hình 6-40c. Sơ đồ nguyên lý của khối 2 IC ngoại vi 8255.

Khi vi xử lý truy xuất bộ nhớ thì chân  $IO/M = 0$ , nếu truy xuất Eprom 2764 thì địa chỉ A13 = 0 nên qua cổng OR IC13A làm cho chân CS = 0: cho phép bộ nhớ Eprom 2764. Trong khi đó A13 = 0 qua cổng đảo IC6A và cổng OR IC13B làm chân CS1 = 1 sẽ không cho phép SRAM hoạt động: bus địa chỉ ở trạng thái tổng trở cao.

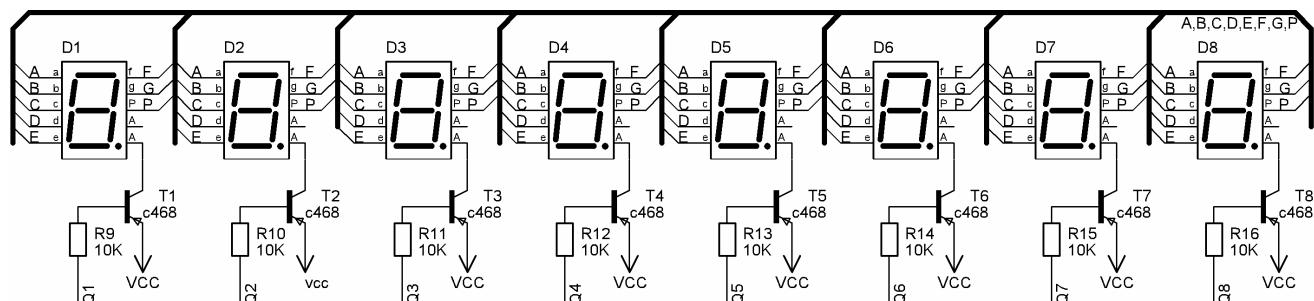
Tương tự khi truy xuất bộ nhớ SRAM 6264 thì bộ nhớ EPROM sẽ ở trạng thái không cho phép.

Khi vi xử lý truy xuất bộ nhớ thì đường điều khiển  $IO/M$  ở mức L, khi truy xuất IO thì ở mức H nên đường điều khiển này dùng để cho phép truy xuất bộ nhớ hoặc IO.

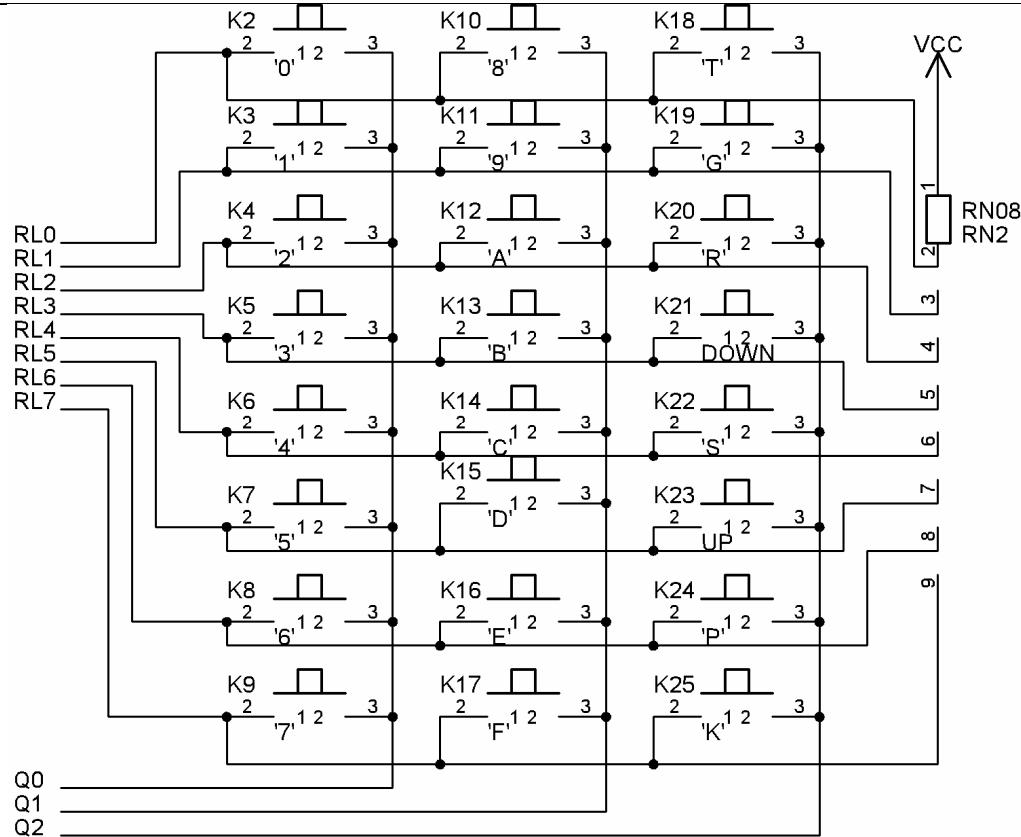
Trong sơ đồ nguyên lý hình 6-40c đã sử dụng IC 74238 giải mã địa chỉ cho 2 IC 8255A và 1 IC 8279. Các bus địa chỉ, bus dữ liệu và điều khiển kết nối với IC ngoại vi giống như kết nối với bộ nhớ. Đường điều khiển  $IO/M$  sẽ điều khiển chân G1 của IC 74138 để cho phép IC này giải mã khi vi xử lý truy xuất IO, còn khi truy xuất bộ nhớ thì chân  $IO/M = 0$  sẽ không cho phép IC giải mã nên các IC ngoại vi cũng không được phép, bus dữ liệu sẽ ở trạng thái tổng trở cao.



Hình 6-40d. Sơ đồ nguyên lý của khối ngoại vi 8279 với bàn phím và led 7 đoạn.



Hình 6-40e Sơ đồ nguyên lý của 8 led hiển thị quét.



Hình 6-40f. Sơ đồ nguyên lý của khối ma trận phím 8x3.

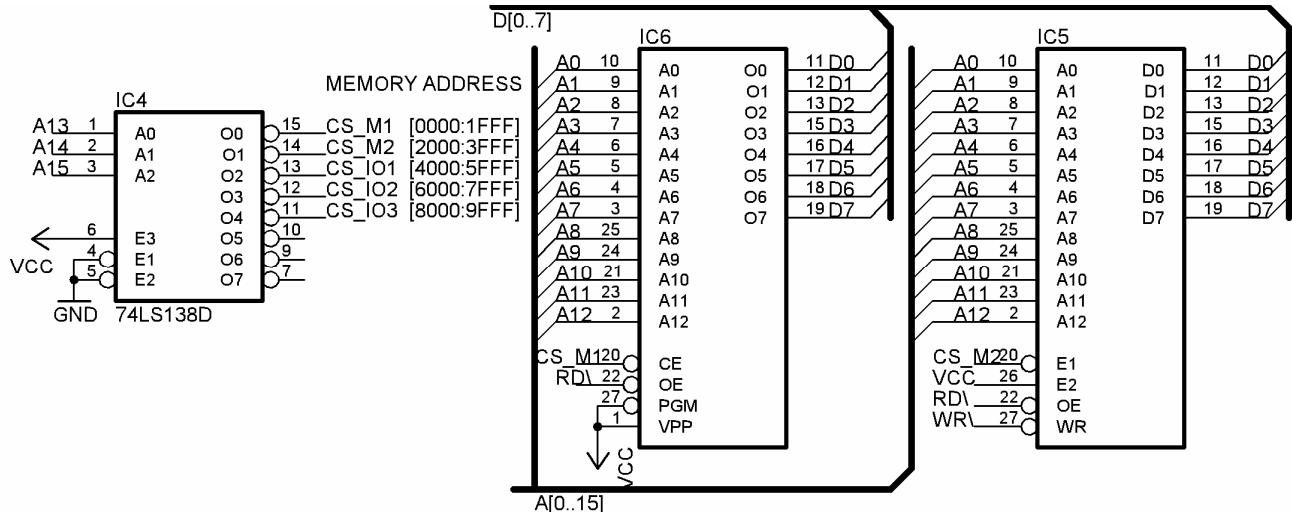
Với bài toán thiết kế trên thì các IC giải mã đã sử dụng gồm 3 loại IC: cổng OR - 7432, cổng Not – 7404 và giải mã 74138. Với kiểu giải mã này thì sử dụng quá nhiều IC và ta có thể giải mã đơn giản tiết kiệm IC hơn bằng cách xem các IC ngoại vi 8255A và 8279 như là các IC nhớ có cùng dung lượng 8Kbyte như 2 bộ nhớ trong hệ thống, bảng địa chỉ bộ nhớ của hệ thống được thiết lập như sau:

Lập bảng địa chỉ bộ nhớ của vi xử lý như sau:

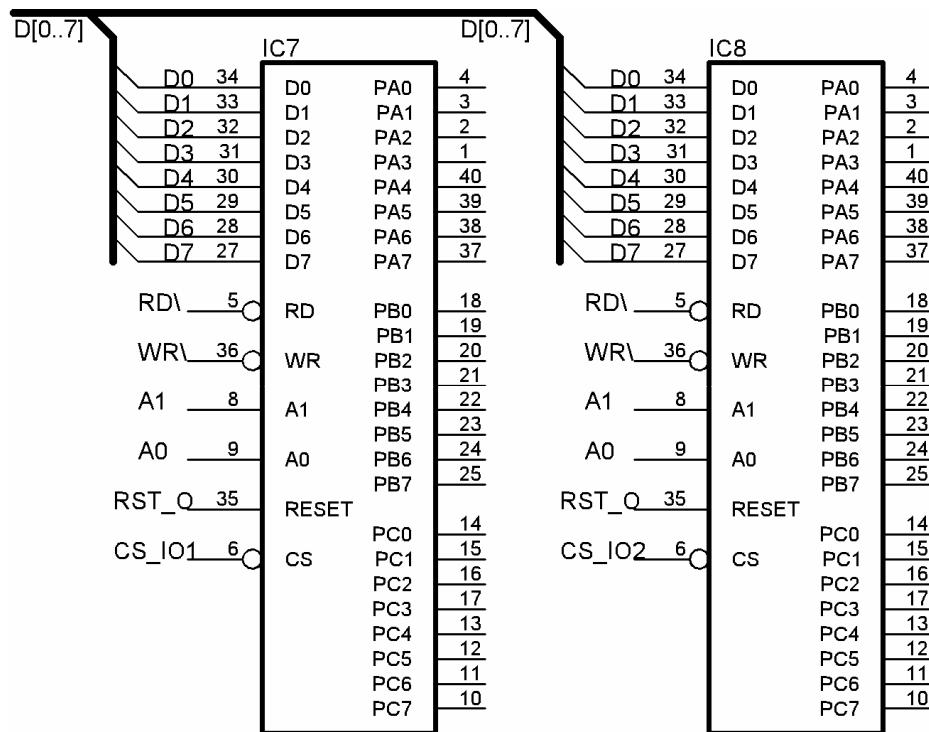
TB	NHỊ PHÂN																HEX
	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
EPROM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
2764	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
RAM	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
6264	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF
8255	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000
1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFF
8255	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
2	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFF
8279	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000
3	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	9FFF

Bảng 6-22. Bảng địa chỉ bộ nhớ gồm bộ nhớ và IO.

Trong bảng địa chỉ bộ nhớ 6-22 thì để phân biệt các IC nhớ và ngoại vi ta phải dùng 3 đường địa chỉ A[15÷13] và khi đó ta chỉ cần dùng 1 IC giải mã 74138 và toàn bộ khối bộ nhớ và ngoại vi được thiết kế lại như hình 6-41:



Hình 6-41a. Sơ đồ nguyên lý của khối bộ nhớ và IC giải mã.



Hình 6-41b. Sơ đồ nguyên lý của khối ngoại vi 8255.

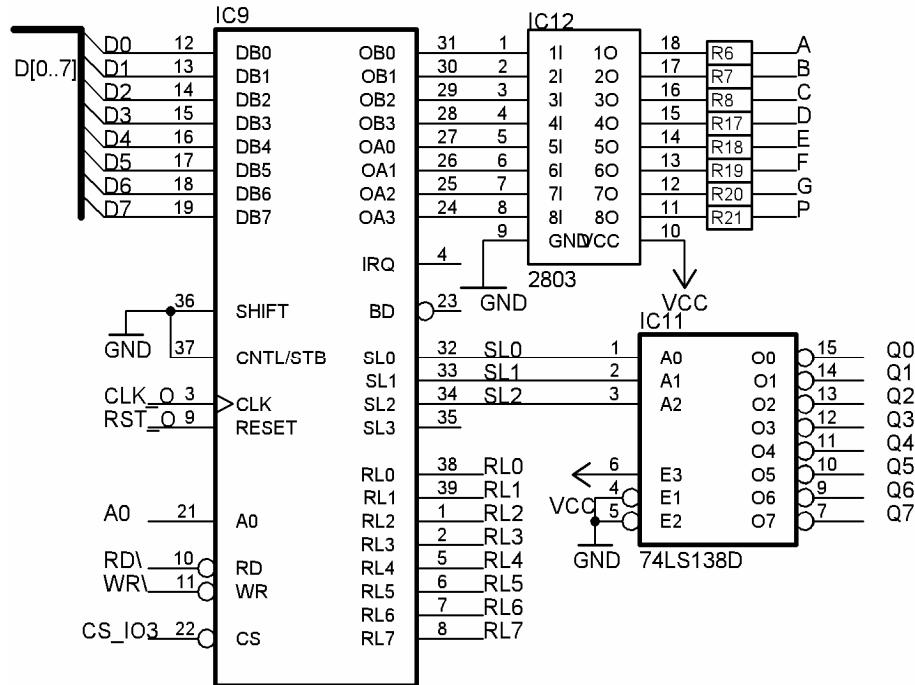
Khi thiết kế khối ngoại vi 8255A theo kiểu IO thì lệnh dùng để truy xuất các IO là lệnh “IN và OUT”, nhưng khi kết nối theo kiểu bộ nhớ thì lệnh dùng để truy xuất IO giống như lệnh dùng cho các ô nhớ.

Trong kiểu giải mã này thì địa chỉ của 8255A – 1 (IC7) có địa chỉ là 4000H đến 5FFFH nhưng do 8255A chỉ có 2 đường địa chỉ A[1÷0] nên ta chỉ cần quan tâm đến 4 địa chỉ như sau:

- Địa chỉ 4000H là địa chỉ của port A.
  - Địa chỉ 4001H là địa chỉ của port B.
  - Địa chỉ 4002H là địa chỉ của port C.

- Địa chỉ 4003H là địa chỉ của từ điều khiển.

Tương tự địa chỉ của 8255A – 2 (IC8) có địa chỉ là 6000H đến 7FFFH nhưng ta chỉ cần quan tâm đến 4 địa chỉ như sau:



**Hình 6-41c. Sơ đồ nguyên lý của khối 8279.**

- Địa chỉ 6000H là địa chỉ của port A.
- Địa chỉ 6001H là địa chỉ của port B.
- Địa chỉ 6002H là địa chỉ của port C.
- Địa chỉ 6003H là địa chỉ của từ điều khiển.

Địa chỉ của 8279 – 3 (IC9) có địa chỉ là 8000H đến 9FFFH nhưng do 8279 chỉ có 1 đường địa chỉ A0 nên ta chỉ cần quan tâm đến 2 địa chỉ như sau:

- Địa chỉ 8000H là địa chỉ truy xuất dữ liệu.
- Địa chỉ 8001H là địa chỉ truy xuất từ điều khiển của 8279.

## 2 BÀI THIẾT KẾ SỐ 2

**Yêu cầu:** “ Hãy thiết kế 1 hệ thống dùng vi xử lý 8 bit 8085A giao tiếp với 32Kbyte bộ nhớ EPROM sử dụng 4 IC 2764 có dung lượng 8Kbyte và 32 Kbyte RAM sử dụng 1 IC 62256. Giao tiếp với 2 IC ngoại vi 8255A và 1 IC 8279 để điều khiển hiển thị 8 led 7 đoạn và ma trận bàn phím gồm 24 phím”.

### Phân tích yêu cầu

Hệ thống sử dụng vi xử lý 8085A có 16 đường địa chỉ nên có thể giao tiếp với 64Kbyte bộ nhớ. Trong hệ thống này bộ nhớ của vi xử lý đã sử dụng hết 64 kbyte bao gồm 32Kbyte Eprom và 32 Kbyte Ram. Phần giao tiếp với 2 IC 8255 và IC 8279 thì phải giải mã theo kiểu IO.

### Thiết kế giải mã địa chỉ :

Tổng IC giao tiếp với vi xử lý là 8 IC gồm 5 IC nhớ và 3 IC ngoại vi.

Lập bảng địa chỉ bộ nhớ của vi xử lý như sau:

TB	NHỊ PHÂN																HEX
	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
EPROM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
2764-1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
EPROM	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
2764-2	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF
EPROM	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000
2764-3	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFF
EPROM	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
2764-4	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFF
RAM	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000
62256	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFF

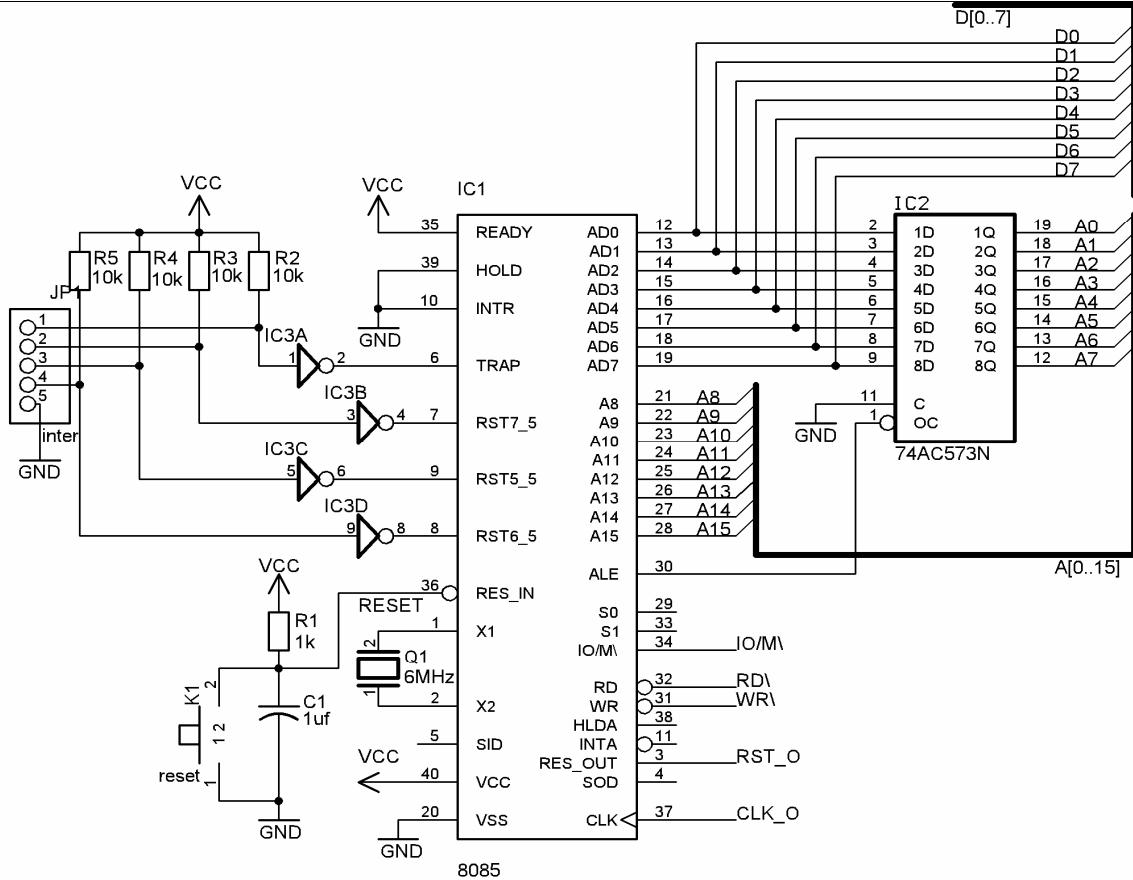
Bảng 6-23. Bảng địa chỉ bộ nhớ cho 4 eprom và 1 ram.

Trong bảng đồ nhớ thì 4 IC nhớ 2764 được thiết lập địa chỉ giống như bài 1, bộ nhớ SRAM có dung lượng 32Kbyte sẽ có 15 đường địa chỉ nên địa chỉ bắt đầu tương ứng với 15 đường bằng 0 và địa chỉ cuối tương ứng với 15 đường bằng 1. Nhìn vào bảng ta sẽ thấy 32kbyte EPROM phân biệt với 32Kbyte Ram chính là đường địa chỉ A15.

Ta có thể xem bộ nhớ Ram 62256 như là 4 IC nhớ 6264 khi đó bảng địa chỉ bộ nhớ được thiết lập lại như sau:

TB	NHỊ PHÂN																HEX
	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
EPROM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
2764-1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
EPROM	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
2764-2	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF
EPROM	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000
2764-3	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFF
EPROM	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
2764-4	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFF
RAM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000
6264-1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	9FFF
RAM	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	A000
6264-2	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	BFFF
RAM	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C000
6264-3	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	DFFF
RAM	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	E000
6264-4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFF

Bảng 6-24. Bảng địa chỉ bộ nhớ cho 4 rom và 4 ram.

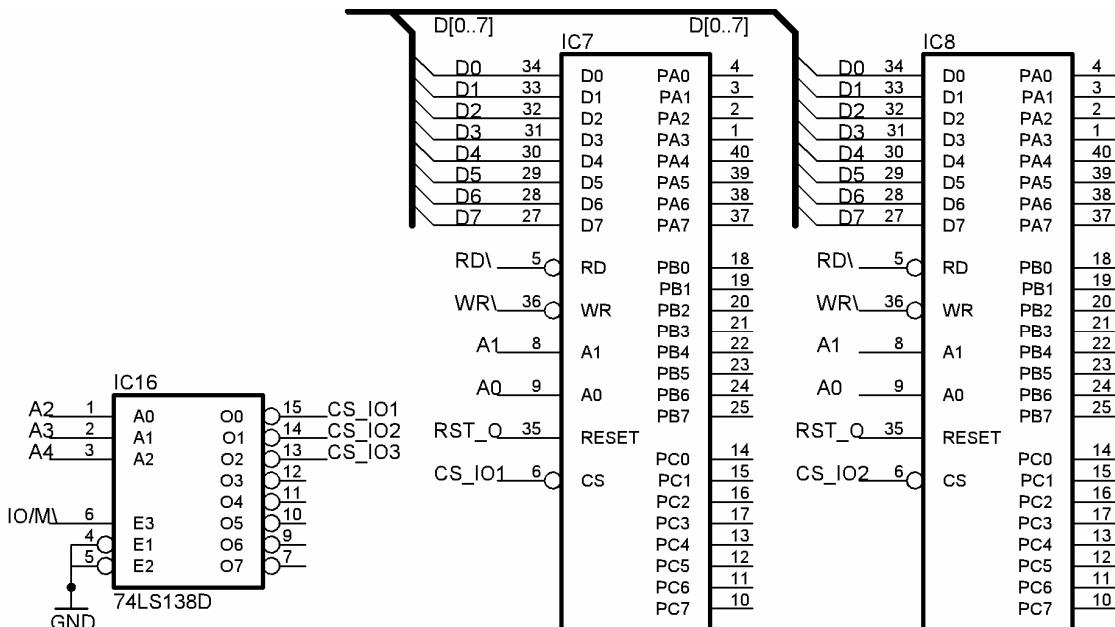


Hình 6-42a. Sơ đồ nguyên lý của khối vi xử lý và IC chốt.

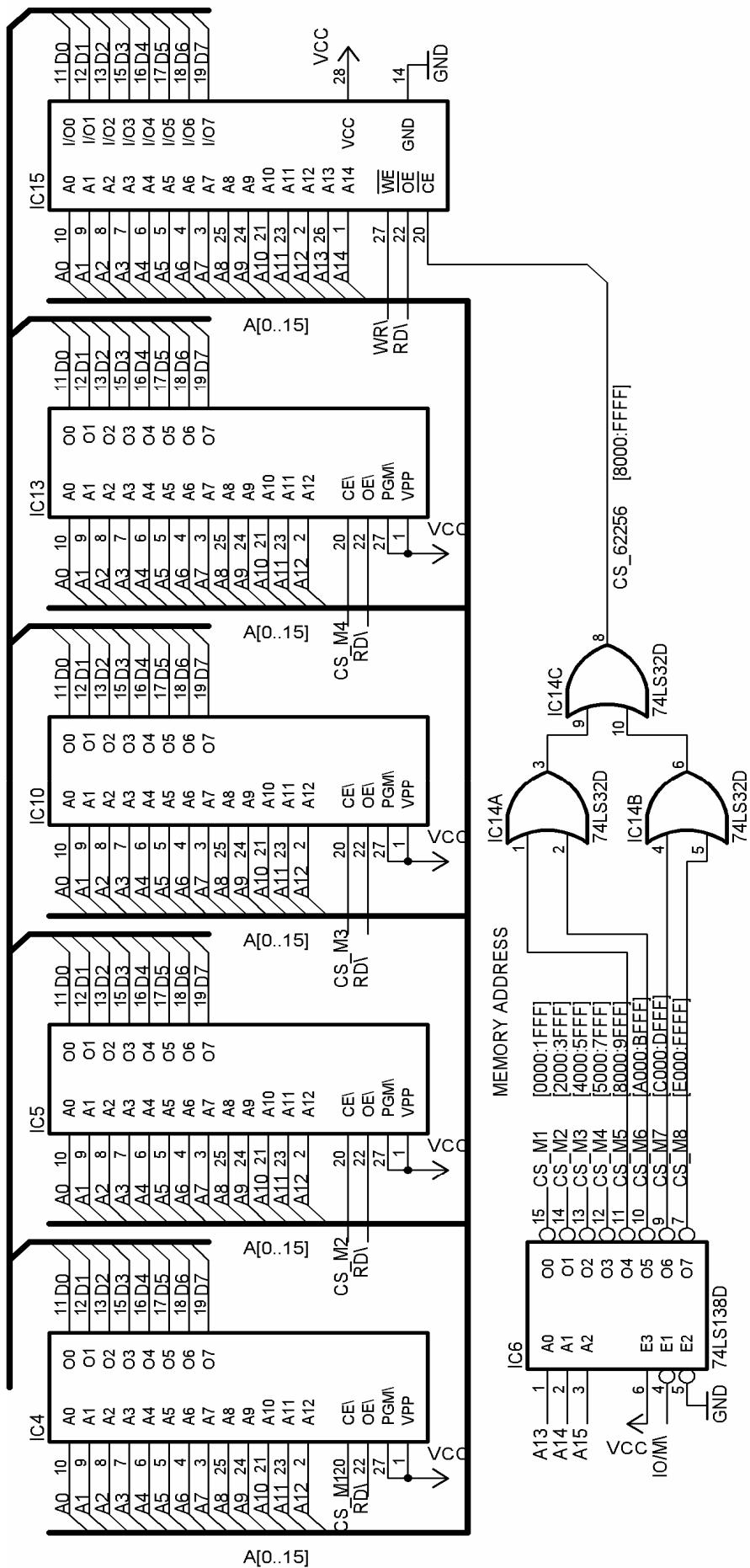
Với bảng địa chỉ bộ nhớ này thì ta sẽ dùng 3 đường địa chỉ A[15:13] giải mã thành 8 đường điều khiển để chọn 8 chip select cho 8 IC nhớ: trong đó 4 đường đầu tiên dùng để chọn CS cho 4 Eprom, 4 đường còn lại ta phải OR lại để điều khiển chọn CS của Ram 62256.

Phần giải mã IC ngoại vi giống như bài thiết kế số 1.

Sơ đồ nguyên lý của hệ thống như hình 6-42.



Hình 6-42b. Sơ đồ nguyên lý của khối ngoại vi.



Hình 6-42c. Sơ đồ nguyên lý của khồi bộ nhỏ.

**V. BÀI TẬP:**

**Bài tập số 1:** Hãy thiết kế một hệ thống gồm vi xử lý 8085A giao tiếp với 32 Kbyte bộ nhớ Eprom sử dụng 8 IC nhớ 2732 có dung lượng 4 Kbyte cho mỗi IC và 32Kbyte bộ nhớ RAM dùng 4 IC 6264. Các IC ngoại vi bao gồm 7 IC ngoại vi 8255 và 1 IC 8279.

**Bài tập số 2:** Hãy thiết kế một hệ thống gồm vi xử lý 8085A giao tiếp với 32 Kbyte bộ nhớ Eprom sử dụng 4 IC nhớ 2764 có dung lượng 8 Kbyte cho mỗi IC và 16Kbyte bộ nhớ RAM dùng 2 IC 6264. Các IC ngoại vi bao gồm 1 IC ngoại vi 8255 và 1 IC 8279. Thiết kế sao cho tối ưu nhất.

**Bài tập số 3:** Hãy thiết kế một hệ thống gồm vi điều khiển 89S52 giao tiếp với 32 Kbyte bộ nhớ Eprom sử dụng 4 IC nhớ 2764 có dung lượng 8 Kbyte cho mỗi IC và 16Kbyte bộ nhớ RAM dùng 2 IC 6264. Các IC ngoại vi bao gồm 1 IC ngoại vi 8255 và 1 IC 8279. Thiết kế sao cho tối ưu nhất.

**Bài tập số 4:** Hãy thiết kế một hệ thống gồm vi điều khiển 89S52 sử dụng 8Kbyte bộ nhớ Flash ROM nội và giao tiếp với 24Kbyte bộ nhớ Eprom sử dụng IC nhớ 2764 có dung lượng 8 Kbyte cho mỗi IC và 32Kbyte bộ nhớ RAM dùng 4 IC 6264. Các IC ngoại vi bao gồm 3 IC ngoại vi 8255 và 1 IC 8279. Thiết kế sao cho tối ưu nhất.

[return](#)  
**the end**



# GIAO TIẾP ỨNG DỤNG

## VI ĐIỀU KHIỂN

### VI ĐIỀU KHIỂN GIAO TIẾP VỚI LED

1. GIAO TIẾP VỚI LED ĐƠN
  - a. Giao tiếp phần cứng
  - b. Các chương trình ví dụ
2. GIAO TIẾP VỚI LED 7 ĐOAN
  - a. Giao tiếp với 1 led 7 đoạn
  - b. Các chương trình giao tiếp với 1 led 7 đoạn
  - c. Giao tiếp với nhiều led 7 đoạn
  - d. Các chương trình ví dụ giao tiếp với 8 led 7 đoạn

### VI ĐIỀU KHIỂN GIAO TIẾP VỚI PHÍM NHẤN

1. GIAO TIẾP VỚI 1 HOẶC 2 PHÍM NHẤN
2. GIAO TIẾP VỚI MA TRÂN PHÍM

### VI ĐIỀU KHIỂN GIAO TIẾP VỚI ADC, DAC

1. GIAO TIẾP VỚI ADC 0809
2. GIAO TIẾP VỚI ADC 7109
3. GIAO TIẾP VỚI DAC 0808

### VI ĐIỀU KHIỂN GIAO TIẾP VỚI RELAY VÀ ĐỘNG CƠ BƯỚC

1. GIAO TIẾP VỚI RELAY
2. GIAO TIẾP VỚI ĐỘNG CƠ BƯỚC

### VI ĐIỀU KHIỂN GIAO TIẾP VỚI LCD

1. GIỚI THIỆU LCD
2. SƠ ĐỒ CHÂN CỦA LCD
3. SƠ ĐỒ MẠCH GIAO TIẾP VI ĐIỀU KHIỂN VỚI LCD
4. CÁC LỆNH ĐIỀU KHIỂN LCD
5. LƯU ĐỒ ĐIỀU KHIỂN LCD
6. CHƯƠNG TRÌNH ĐIỀU KHIỂN LCD DÙNG VI ĐIỀU KHIỂN

### VI ĐIỀU KHIỂN GIAO TIẾP VỚI LED MA TRÂN

1. GIỚI THIỆU LED MA TRÂN
2. CẤU TẠO VÀ PHƯƠNG PHÁP ĐIỀU KHIỂN LED MA TRÂN

### 3. CHƯƠNG TRÌNH ĐIỀU KHIỂN LED MA TRẬN

## VI ĐIỀU KHIỂN GIAO TIẾP VỚI REALTIME

1. GIỚI THIỆU REALTIME
2. CÁC THÔNG SỐ CỦA REALTIME DS12887
3. SƠ ĐỒ CHÂN VÀ CHỨC NĂNG CÁC CHÂN CỦA REALTIME DS12887
4. HOẠT ĐỘNG CỦA REALTIME KHI MẤT NGUỒN VÀ KHI CẤP ĐIỆN
5. HOẠT ĐỘNG CỦA CÁC THANH GH ĐIỀU KHIỂN
6. MẠCH GIAO TIẾP VI ĐIỀU KHIỂN VỚI REAL-TIME
7. CHƯƠNG TRÌNH KHỞI TẠO REALTIME

## BÀI TẬP

## *LIỆT KÊ CÁC HÌNH*

Hình 7-1. Giao tiếp vi điều khiển với led đơn.

Hình 7-2. Giao tiếp vi điều khiển với 32 led đơn.

Hình 7-3. Giao tiếp trực tiếp vi điều khiển với 1 led đoạn.

Hình 7-4. Giao tiếp gián tiếp vi điều khiển với led đơn.

Hình 7-5. Giao tiếp vi điều khiển với 8 led 7 đoạn.

Hình 7-6. Giao tiếp vi điều khiển với 8 led 7 đoạn chỉ dùng 1 port 8 đường.

Hình 7-7. Giao tiếp vi điều khiển với 2 nút nhấn.

Hình 7-8. Lưu đồ điều khiển.

Hình 7-9. Giao tiếp vi điều khiển với ma trận phím 4x4.

Hình 7-10. Lưu đồ quét ma trận phím 4x4.

Hình 7-11. Lưu đồ chống dội sau khi quét phím.

Hình 7-12. Sơ đồ chân IC ADC 0809.

Hình 7-13. Sơ đồ khồi bên trong IC ADC 0809.

Hình 7-14. Giao tiếp vi điều khiển với ADC 0809.

Hình 7-15. Giản đồ thời gian của ADC 0809.

Hình 7-16. Lưu đồ điều khiển ADC 0809.

Hình 7-17. Sơ đồ chân IC ADC ICL 7109.

Hình 7-18. Giao tiếp vi điều khiển AT89S52 với IC ADC ICL 7109.

Hình 7-19. Lưu đồ điều khiển ADC ICL 7109.

Hình 7-20. Sơ đồ chân IC DAC 0808.

Hình 7-21. Sơ đồ giao tiếp vi điều khiển với DAC 0808.

Hình 7-22. Sơ đồ giao tiếp vi điều khiển với relay.

Hình 7-23. Sơ đồ giao tiếp vi điều khiển với 4 relay qua IC giao tiếp ULN2803.

Hình 7-24. Hình động cơ bước loại nhỏ.

Hình 7-25. Các cuộn dây bên trong động cơ bước.

Hình 7-26. Các cuộn dây bên trong động cơ bước.

Hình 7-27. Điều khiển kích 1 cuộn dây.

Hình 7-28. Điều khiển kích 2 cuộn dây.

Hình 7-29. Điều khiển phôi hợp cả hai.

Hình 7-30. Sơ đồ giao tiếp vi điều khiển với động cơ bước qua IC ULN2803.

Hình 7-31. Hình của LCD

Hình 7-32. Giao tiếp vi điều khiển 87C751 với LCD.

Hình 7-33. Giao tiếp vi điều khiển AT89S52 với LCD.

Hình 7-34. Dạng sóng điều khiển của LCD.

Hình 7-35. Lưu đồ điều khiển LCD.

Hình 7-36. Lưu đồ xuất lệnh hoặc dữ liệu ra LCD.

Hình 7-37. Led ma trận 5x7.

Hình 7-38. Sáng chữ A.

Hình 7-39. Sơ đồ mạch điều khiển.

Hình 7-40. Sơ đồ chân của real-time DS12C887.

Hình 7-41. Tổ chức bộ nhớ bên trong của Real-time DS12C887.

Hình 7-42. Giao tiếp vi điều khiển với Real-time.

## LIỆT KÊ CÁC BẢNG

Bảng 7-1. Mã quét điều khiển led.

Bảng 7-2. Mã 7 đoạn của các số.

Bảng 7-3. Mã quét xuất ra cột và mã hàng được đọc về.

Bảng 7-4. Bảng trạng thái chọn kênh ADC.

Bảng 7-6. Các chân của LCD

Bảng 7-7. Các lệnh của LCD

Bảng 7-8. Mã chữ A.

Bảng 7-9. Quét theo cách 2.

Bảng 7-10. Các thông số tín hiệu ngo ra SQW.

Bảng 7-11. Các định dạng của các thông số thời gian.

## I. VI ĐIỀU KHIỂN GIAO TIẾP VỚI LED:

### 1 GIAO TIẾP VỚI LED ĐƠN

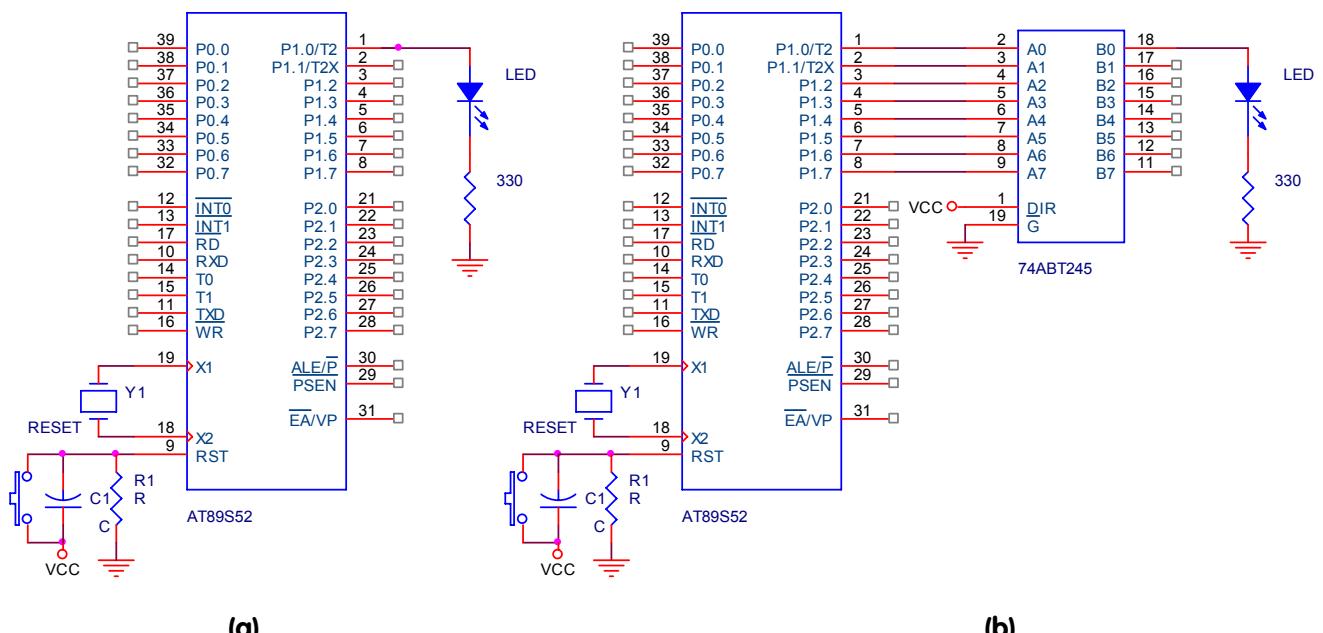
#### a Giao tiếp phần cứng

Trong các ứng dụng có sử dụng led đơn để chỉ thị nên phần này sẽ trình bày phần giao tiếp với led đơn.

Các thông số của led đơn thường sử dụng là điện áp làm việc của led khoảng 2V, dòng qua led khoảng từ 10 đến 20 mA.

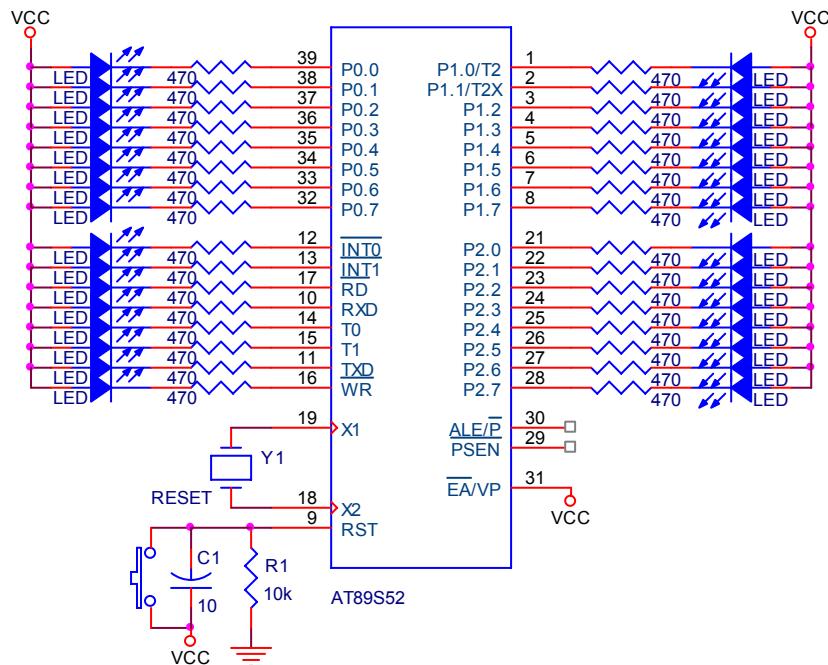
Tra các thông số làm việc của mỗi ngõ ra vi điều khiển thì khi ngõ ra ở mức H dòng chạy ra (source) có giá trị rất nhỏ khoảng từ 10 $\mu$ A đến 60 $\mu$ A đối với port1, 2, 3 và dòng khoảng từ 80 $\mu$ A đến 800 $\mu$ A đối với port0. Khi ngõ ra ở mức L dòng chạy vào (sink) khoảng 15mA đối với các port1, 2, 3 (15mA/1 port) và dòng khoảng 20mA đối với port0 (20mA/port0).

Khi giao tiếp với led đơn thì sẽ có 2 kiểu giao tiếp như hình 7-1.



Hình 7-1. Giao tiếp vi điều khiển với led đơn.

- Hình 7-1a kết nối trực tiếp ngõ ra của port với led thì mức 1 led sáng nhưng không đủ dòng cung cấp cho led sáng nên led sáng mờ, nếu muốn sáng rõ thì phải dùng thêm IC đệm hoặc transistor để khuếch đại. Hình 7-1b dùng IC đệm 74245 để điều khiển led.
- Hình 7-2 kết nối trực tiếp ngõ ra của port với led thì mức 0 led sáng và đủ dòng cung cấp cho led sáng nên led sáng rõ không cần IC đệm.



Hình 7-2. Giao tiếp vi điều khiển với 32 led đơn.

Điện trở hạn dòng cho led được tính như sau:

$$R = \frac{V_{CC} - V_{LED} - V_{OL}}{I_{LED}} = \frac{5V - 1.8V - 0.45V}{5.6mA} = 491\Omega$$

Nên chọn loại LED sáng rõ, dòng làm việc khá nhỏ nên điện trở hạn dòng khoảng  $470\Omega$ .

**Chú ý:** khi sử dụng vi điều khiển tuỳ thuộc vào kết nối ta có thể dùng hoặc không dùng điện trở kéo lên nhưng theo tác giả thì tất cả các port nên dùng điện trở kéo lên, nếu trong sơ đồ không trình bày thì bạn đọc hiểu ngầm là luôn có kết nối.

### b. Các chương trình ví dụ:

#### Ví dụ 1: Với sơ đồ nguyên lý hình 7-2, hãy viết chương trình điều khiển 32 led đơn chớp tắt:

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh dieu khien 32 led chup tat
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    org    0000h          ;khai bao dia chi bat dau cua chuong trinh
lb:    mov    p0,#00h      ;sang 8 led cua p0
        mov    p1,#00h      ;sang 8 led cua p1
        mov    p2,#00h      ;sang 8 led cua p2
        mov    p3,#00h      ;sang 8 led cua p3
        lcall   delay       ;goi chuong trinh con delay

        mov    p0,#0ffh      ;tat 8 led cua p0
        mov    p1,#0ffh      ;tat 8 led cua p1
        mov    p2,#0ffh      ;tat 8 led cua p2
        mov    p3,#0ffh      ;tat 8 led cua p3
        lcall   delay       ;goi chuong trinh con delay
        sjmp   lb            ;lam lai tu dau
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
delay:   mov    r6,#0ffh      ;nap hang so delay FFH vao thanh ghi r6
de :    mov    r7,#0ffh      ;nap hang so delay FFH vao thanh ghi r7
        djnz   r7,$         ;giam thanh ghi r7 di 1 va nhay khi r7 khac 00
```

	djnz r6,de ret end	;giam thanh ghi r6 di 1 va nhay khi r6 khac 00 ;thoat khoi chuong trinh con
--	--------------------------	--

**Ví dụ 2:** Với sơ đồ hình 7-2, hãy viết chương trình điều khiển 32 led sáng dần và tắt dần:

```

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh dieu khien 32 led sang dan len va tat dan
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        org    0000h
        mov    p0,#0ffh      ;tat port 0
        mov    p1,#0ffh      ;tat port 1
        mov    p2,#0ffh      ;tat port 2
        mov    p3,#0ffh      ;tat port 3
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;dieu khien sang dan len
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
lb:     lcall   delay      ;goi chuong trinh con delay
        clr    c          ;lam cho bit C = 0

        mov    a,p0      ;chuyen noi dung port0 vao thanh ghi A
        rlc    a          ;xoay noi dung thanh ghi A sang trai
        mov    p0,a      ;chuyen noi dung port0 vao thanh ghi A
        mov    a,p1      ;xoay noi dung p1
        rlc    a
        mov    p1,a
        mov    a,p2      ;xoay noi dung p2
        rlc    a
        mov    p2,a
        mov    a,p3      ;xoay noi dung p3
        rlc    a
        mov    p3,a
        jc     lb          ;thuc hien tiep khi C=1
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;dieu khien tat dan
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
lb1:   lcall   delay      ;goi chuong trinh con delay
        setb   c          ;lam cho bit C = 1
        mov    a,p0      ;chuyen noi dung port0 vao thanh ghi A
        rlc    a          ;xoay noi dung thanh ghi A sang trai
        mov    p0,a      ;chuyen noi dung port0 vao thanh ghi A
        mov    a,p1      ;xoay noi dung p1
        rlc    a
        mov    p1,a
        mov    a,p2      ;xoay noi dung p2
        rlc    a
        mov    p2,a
        mov    a,p3      ;xoay noi dung p3
        rlc    a
        mov    p3,a
        jnc   lb1         ;thuc hien tiep khi C=0
        sjmp  lb          ;sau khi 8 led sang het thi quay lai tu dau
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con delay
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
delay:  mov    r6,#0ffh
de :    mov    r7,#0ffh
        djnz  r7,$
        djnz  r6,de
        ret
end

```

**Bài tập 1:** Hãy viết chương trình điều khiển 32 sáng tắt dần từ trái sang phải, từ phải sang trái, từ ngoài vào và từ giữa ra.

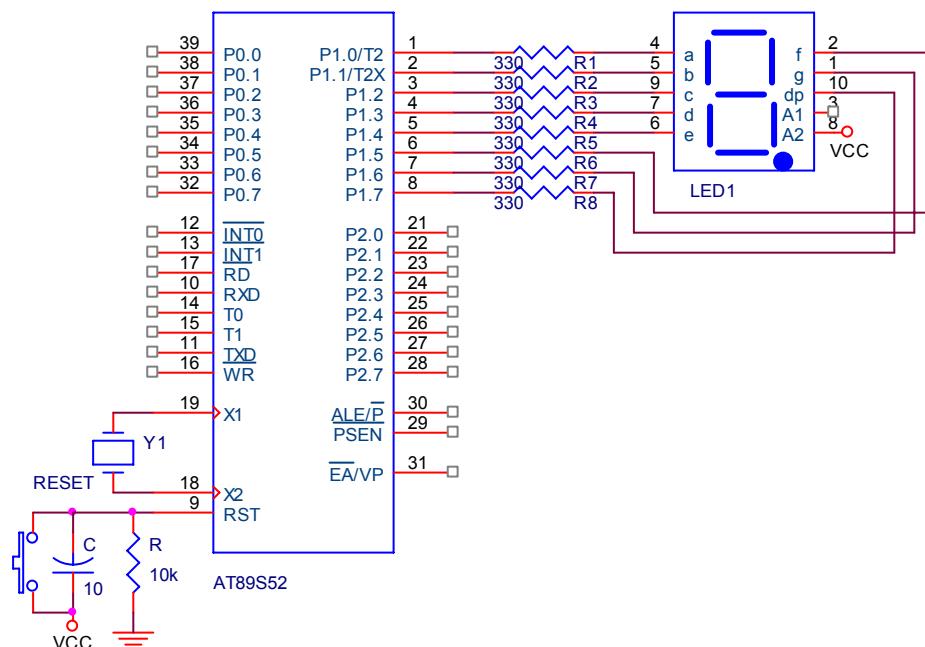
**Bài tập 2:** Hãy viết chương trình giống như bài 1 nhưng mỗi kiểu sáng được thực hiện 5 lần.

## 2. GIAO TIẾP VỚI LED 7 ĐOẠN

### a. Giao tiếp với 1 led 7 đoạn

Trong ứng dụng chỉ có 1 led 7 đoạn thì có nhiều cách giao tiếp vi điều khiển với led:

- Giao tiếp trực tiếp: kiểu này sẽ dùng 1 port của vi điều khiển kết nối với led 7 đoạn loại Anode chung như hình 7-3. Kiểu này dùng 8 đường IO.
- Giao tiếp gián tiếp: kiểu này chỉ dùng 4 đường kết nối với IC giải mã led 7 đoạn như hình 7-4. Kiểu này dùng 4 đường IO.



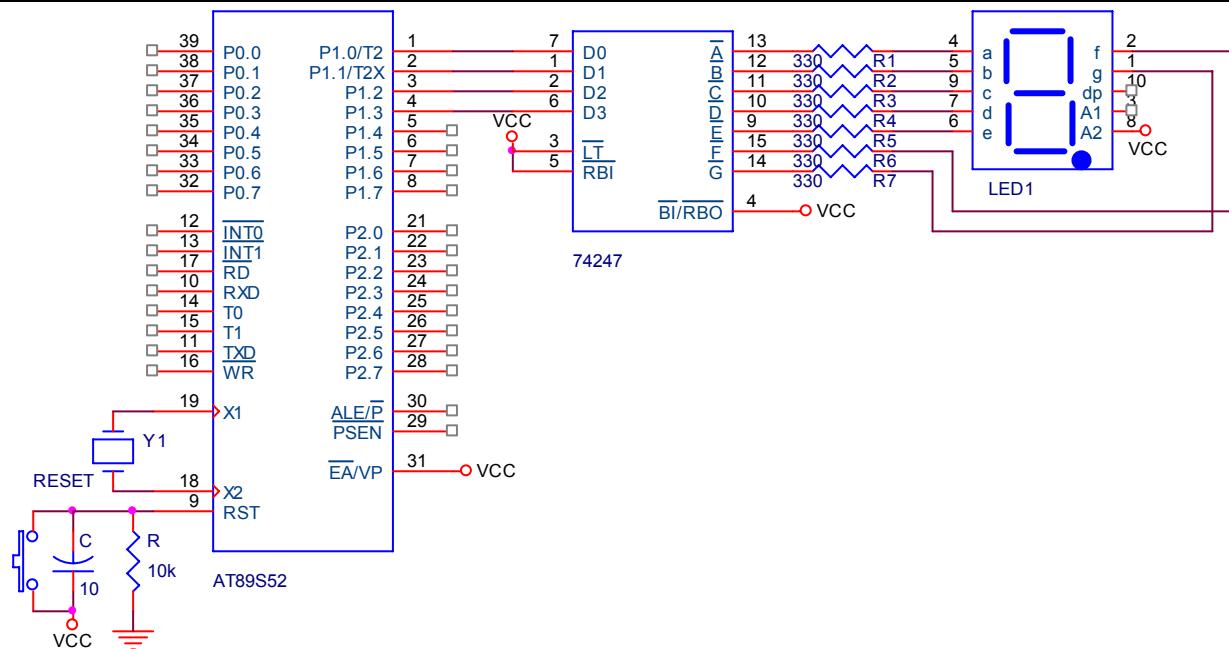
Hình 7-3. Giao tiếp trực tiếp vi điều khiển với 1 led 7 đoạn.

Điện trở hạn dòng cho led được tính như sau:

$$R = \frac{V_{CC} - V_{LED} - V_{OL}}{I_{LED}} = \frac{5V - 1.8V - 0.45V}{5.6mA} = 491\Omega$$

Nên chọn điện trở khoảng  $470\Omega$ .

**Câu hỏi 1:** Hãy cho biết ưu khuyết điểm của 2 cách giao tiếp vi điều khiển với 1 led 7 đoạn ở 2 hình trên.



Hình 7-4. Giao tiếp gián tiếp vi điều khiển với led đơn.

## b. Các chương trình giao tiếp với 1 led 7 đoạn:

Ví dụ 3: Hãy dùng sơ đồ hình 7-3, viết chương trình điều khiển 1 led 7 đoạn đếm lên từ 0 đến 9.

;xx

;chuong trinh dem len tu 0 den 9 hien thi tren 1 led

;xx

```

        dem    equ    r2
        led     equ    p1

        org    0000h           ;bat dau chuong trinh
        mov    dptr,#ma7doan   ;
        mov    dem,#00h         ;dem=00
        main0:   mov    a,dem
        main1:   movc   a,@a+dptr ;lay ma 7 doan
                  mov    led,a      ;goi 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

```

;xx

;chuong trinh con delay

;xx

```

delay:   mov    r5,##0ffh
loop1:   mov    r6,#0ffh
          djnz  r6,$
          djnz  r5,loop1
          ret

```

;xx

;khai bao ma 7 doan tu so '0' den so '9'

;xx

ma7doan: db 0C0h,0F9h,0A4h,0B0h,99h,92h,82h,0F8h,80h,90h

end

Trong ví dụ 3 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 gởi 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 4: Hãy dùng sơ đồ hình 7-4, viết chương trình điều khiển 1 led 7 đoạn đếm lên từ 0 đến 9.

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
;chuong trinh dem len tu 0 den 9 hien thi tren 1 led
```

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
    dem    equ     r2
    led     equ     p1

    org      0000h           ;bat dau chuong trinh

main0:   mov      dem,#00h        ;dem=00
main1:   mov      a,dem
         mov      led,a          ;goi 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

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con delay
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

delay:    mov      r5,##0ffh
loop1:   mov      r6,#0ffh
         djnz   r6,$
         djnz   r5,loop1
         ret

end
```

Trong ví dụ 4 thì yêu cầu giống ví dụ 3 nhưng trong mạch điện đã dùng IC giải mã led 7 đoạn nên trong chương trình không cần phải tiến hành giải mã, chỉ cần gởi mã BCD ra port.

#### c. Giao tiếp với nhiều led 7 đoạn:

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 7-5.

Tại mỗi một thời điểm ta chỉ cho 1 transistor dẫn và 7 transistor còn lại tắt, dữ liệu gởi ra sẽ sáng trên led tương ứng với transistor dẫn. Sau đó cho 1 transistor khác dẫn và gởi 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 gởi dữ liệu nhanh và do mắt ta có lưu ảnh nên ta nhìn thấy 8 led sáng cùng 1 lúc.

Mã quét: mức logic 0 thì transistor dẫn, mức logic 1 thì transistor ngắn được trình bày ở bảng 7-1.

MÃ HEX	Mã quét điều khiển các transistor							
	1	1	1	1	1	1	0	Transistor 1 ON
FE	1	1	1	1	1	1	0	Transistor 2 ON
FD	1	1	1	1	1	0	1	Transistor 3 ON
FB	1	1	1	1	0	1	1	Transistor 4 ON
F7	1	1	1	1	0	1	1	Transistor 5 ON

<b>EF</b>	1	1	1	0	1	1	1	1	<b>Transistor 5 ON</b>
<b>DF</b>	1	1	0	1	1	1	1	1	<b>Transistor 6 ON</b>
<b>BF</b>	1	0	1	1	1	1	1	1	<b>Transistor 7 ON</b>
<b>7F</b>	0	1	1	1	1	1	1	1	<b>Transistor 8 ON</b>

**Bảng 7-1. Mã quét điều khiển led.**

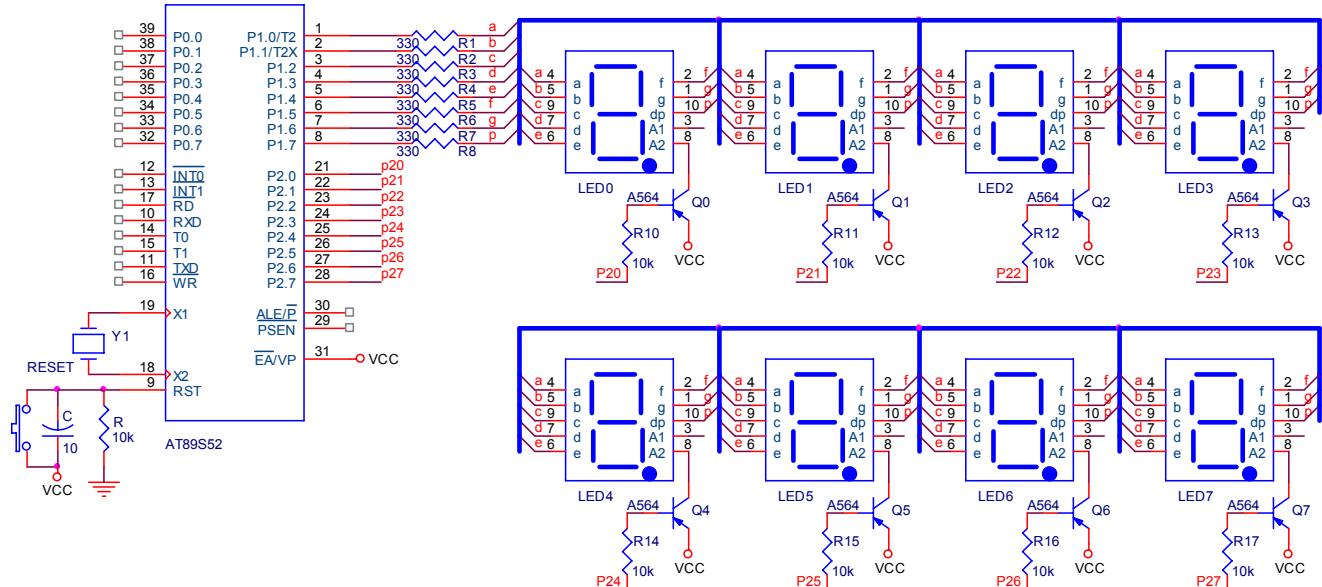
Mã 7 đoạn: trong hệ thống sử dụng led 7 đoạn loại Anode chung nên mức logic 0 thì led sáng và mức logic 1 thì led tắt. Các mã 7 đoạn của các số thập phân từ 0 đến 9 và các số hex từ A đến F được trình bày ở bảng 7-2:



Số hex	dp	g	f	e	d	c	b	a	Mã số hex
0	1	1	0	0	0	0	0	0	C0
1	1	1	1	1	1	0	0	1	F9
2	1	0	1	0	0	1	0	0	A4
3	1	0	1	1	0	0	0	0	B0
4	1	0	0	1	1	0	0	1	99
5	1	0	0	1	0	0	1	0	92
6	1	0	0	0	0	0	1	0	82
7	1	1	1	1	1	0	0	0	F1
8	1	0	0	0	0	0	0	0	80
9	1	0	0	1	0	0	0	0	90
A	1	0	0	0	1	0	0	0	88
B	1	0	0	0	0	0	1	1	83
C	1	1	0	0	0	0	1	0	C2
D	1	0	1	0	0	0	0	1	A1
E	1	0	0	0	0	1	1	0	86
F	1	0	0	0	1	1	1	0	8E

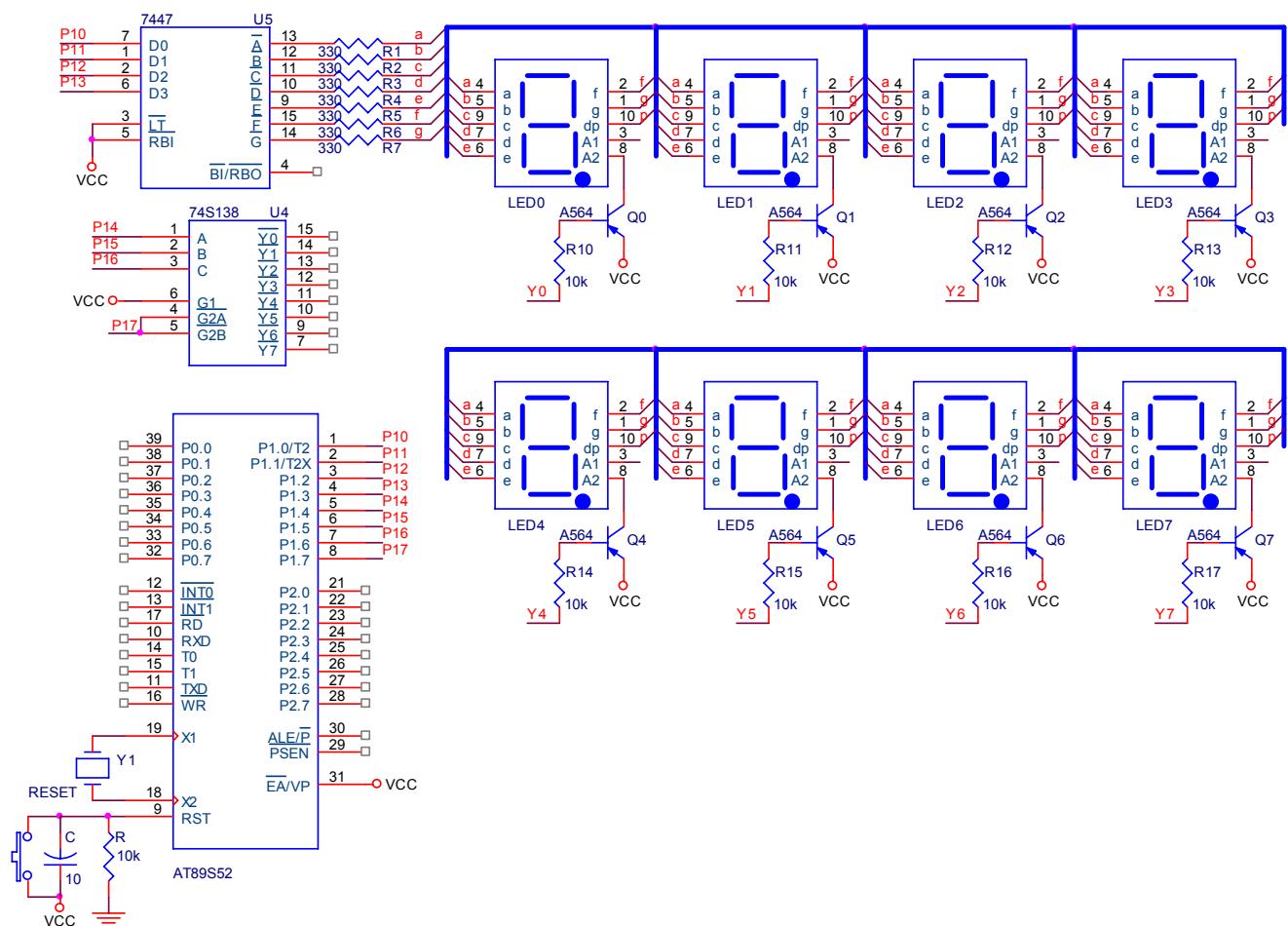
**Bảng 7-2. Mã 7 đoạn của các số.**

Các mã khác có thể tự thiết lập.



Hình 7-5. Giao tiếp vi điều khiển với 8 led 7 đoạn.

Trong sơ đồ hình 7-5 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 7-6.



Hình 7-6. Giao tiếp vi điều khiển với 8 led 7 đoạn chỉ dùng 1 port 8 đường.

Trong sơ đồ hình 7-6 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 gởi 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 ½ chu kỳ và tắt ½ 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 > 40Hz.

Với 8 led 7 đoạn dùng phương pháp quét thì tần số quét đo được trong thực tế phải lớn hơn >60Hz.

**Cách tính toán như sau:** với dòng làm việc bình thường (không quét) chọn là  $I_{LED} = 5mA$ , áp làm việc định mức  $V_{LED} = 1.8V$ . Khi đó điện trở của mỗi led là:

$$R_{BT} = \frac{V_{CC} - V_{LED} - V_{OL}}{I_{LED}} = \frac{5V - 1.8V - 0.45V}{5mA} = 550\Omega$$

**Khi dùng phương pháp quét:** thì dòng tức thời phải bằng 40mA – gấp 8 lần. Để tăng dòng thì có 2 cách: tăng áp hoặc giảm điện trở. Để phù hợp với điện áp làm việc nên ta thực hiện cách giảm điện trở và điện trở được tính lại như sau:

$$R_{Quet} = \frac{V_{CC} - V_{LED} - V_{OL}}{I_{LED\_QUET}} = \frac{5V - 1.8V - 0.45V}{40mA} = 68,75\Omega$$

Vậy điện trở hạn dòng giảm và thường được chọn nằm trong phạm vi từ 68Ω đến 220Ω nhằm tăng thêm cường độ sáng và bảo vệ quá dòng.

**Câu hỏi 2: Hãy cho biết ưu khuyết điểm của 2 cách giao tiếp vi điều khiển với 8 led 7 đoạn ở 2 hình trên.**

#### d. Các chương trình ví dụ giao tiếp với 8 led 7 đoạn:

**Ví dụ 5: Chương trình đếm giây hiển thị trên led 7 đoạn sử dụng phương pháp quét.**

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh dem len tu 00 den 59 hien thi tren 2 led
;su dung ngat timer t0 de dem chinh xac ve thoi gian
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        giay    equ     r2          :gan bien dem giay la R2
        bdn     equ     r1          :gan bien dem ngat

        quet    equ     p2
        led7    equ     p0
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; chuong trinh chinh
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        org     0000h          ;bat dau chuong trinh
        ljmp   main           ;nhay den chtr chinh
```

```

        org 000bh
        ljmp int_t0 ;nhay den chtr con ngat timer0

main:    mov tmod,#01h ;timer0: mod 1 - dem 16 bit
        mov dptr,#ma7doan ;dptr quan ly vung ma 7 doan

        clr tf0 ;xoa co tran
        mov IE,#10000010B ;cho phep timer0 ngat
        mov TH0,#high(-50000) ;khoi tao timer delay 50ms
        mov TL0,#low(-50000)
        setb tr0 ;cho phep timer bat dau dem

main0:   mov giay,#00h ;giay=00
main1:   mov bdn,#00 ;nap bien den so lan ngat
        lcall gma
main2:   lcall hthi ;goi chtr con hien thi

        cjne bdn,#20,main2 ;chua dung 20 lan [tuc 1 giay]
        mov a,giay ;chuyen giay sang A
        add a,#1 ;tang giay len 1
        da a ;hieu chinh so BCD trong A
        mov giay,a ;tra lai cho giay
        cjne giay,#60h,main1 ;ss giay voi 60h
        ljmp main0 ;lam lai tu dau

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; chuong trinh con giao ma
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

gma:    mov a,giay ;xoa 4 bit cao hang chuc giay
        anl a,#0fh ;lay ma 7 doan
        movc a,@a+dptr ;cat ma vao o nho 20h

        mov a,giay ;xoa 4 bit thap hang dvi
        anl a,#0f0h ;chuyen 4 bit cao xuong vi tri thap
        swap a ;lay ma 7 doan hang chuc
        movc a,@a+dptr ;lay ma 7 doan hang chuc
        mov 27h,a
        ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;CHUONG TRINH CON NGAT TIMER0 SAU KHOANG THOI GIAN 50MS
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

int_t0:  inc bdn ;tang bien dem giay len 1
        mov TH0,#high(-50000) ;khoi tao timer delay 50ms
        mov TL0,#low(-50000)
        clr TF0
        reti

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con hien thi
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

hthi:   mov a,#01111111b ;ma quet
        mov r0,#27h

ht1:    mov led7,@r0
        mov quet,a
        lcall delay1
        mov quet,#0ffh
        dec r0
        rr a ;chuyen sang led ke
        cjne r0,#25h,ht1
        ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con delay1

```

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx................................................................
```

**Bài tập 3:** Hãy viết chương trình đếm giây cho sơ đồ hình 7-6.

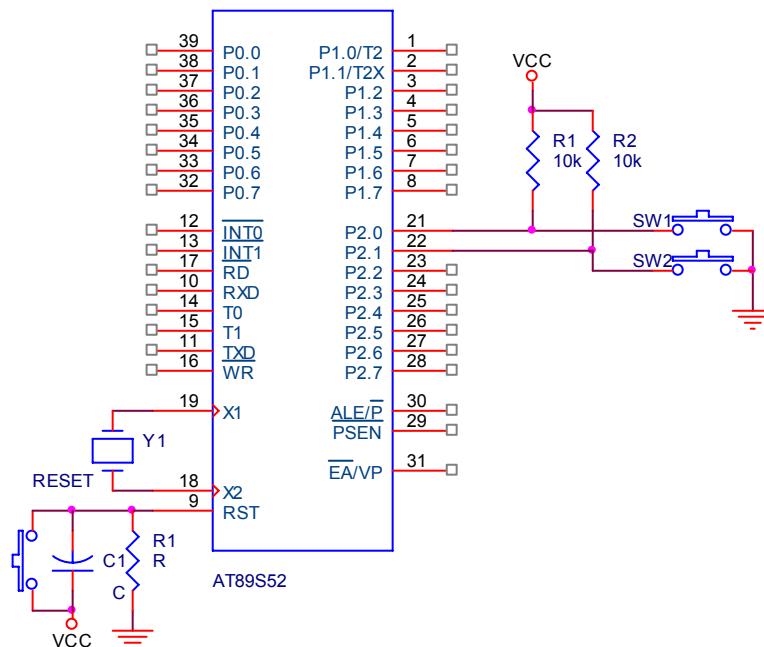
**Bài tập 4:** Hãy viết chương trình đếm phút giây cho cả 2 mạch.

**Bài tập 5:** Hãy viết chương trình đếm giờ phút giây cho cả 2 mạch.

## II. VI ĐIỀU KHIỂN GIAO TIẾP VỚI PHÍM NHẤN:

### 1. GIAO TIẾP VỚI 1 HOẶC 2 NÚT NHẤN:

Trong một số hệ thống điều khiển luôn có nút nhấn hay phím nhấn hay switch dùng để giao tiếp giữa con người và thiết bị điều khiển ví dụ như máy vi tính, máy tính phải có bàn phím để nhập dữ liệu mã, thông tin,... nếu số lượng nút nhấn ít – khoảng 1 đến vài nút thì nên kết nối 1 nút nhấn trực tiếp với 1 ngõ vào như hình 7-7:



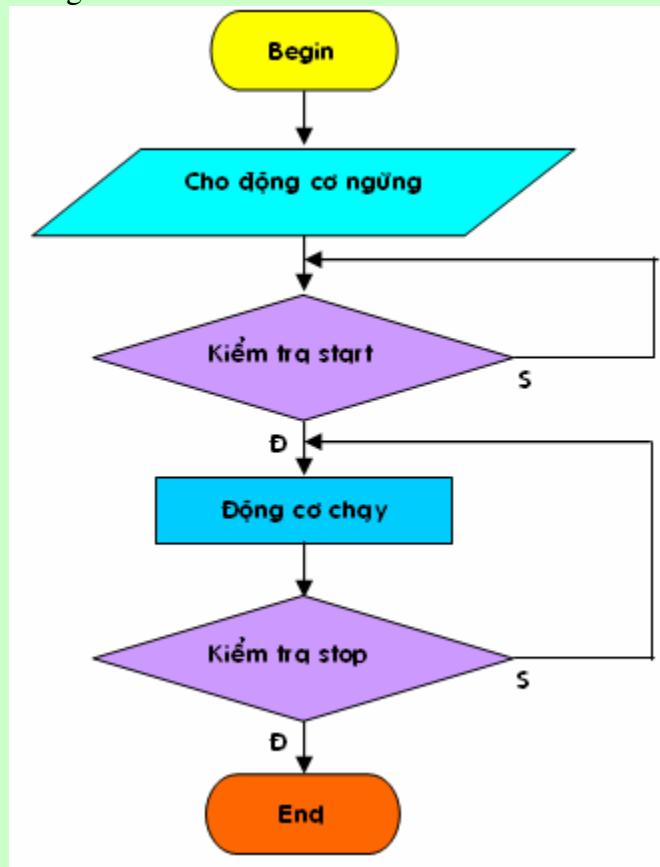
Hình 7-7. Giao tiếp vi điều khiển với 2 nút nhấn.

Khi chưa nhấn thì các ngõ vào của nút nhấn ở mức logic H, khi nhấn thì sẽ làm ngõ ra chuyển sang mức logic L. Trong hình 7-7, 2 nút nhấn được nối đến ngõ vào P2.0 và P2.1 của vi điều khiển. Chương trình sẽ kiểm tra sự thay đổi trạng thái của mức logic và thực thi các công việc tương ứng với từng nút nhấn.

Với vi điều khiển thì các port thường ở mức logic 1 nên 2 điện trở kéo lên là không cần thiết nhưng ở các vi mạch khác không có tính năng đó nếu không có 2 điện trở kéo lên thì không hoạt động được.

**Ví dụ 6:** Chương trình điều khiển động cơ DC có 2 nút nhấn Start và Stop.

Lưu đồ điều khiển cho chương trình như hình 7-8:



Hình 7-8. Lưu đồ điều khiển.

Giải thích lưu đồ:

Mặc nhiên khi cấp điện thì động cơ ngừng, chờ kiểm tra phím Start có được nhấn hay không: nếu không nhấn thì tiếp tục chờ, nếu nhấn thì cho động cơ chạy. Kiểm tra xem có nhấn phím Stop hay không: nếu không nhấn thì động cơ tiếp tục chạy, nếu có nhấn stop thì kết thúc chương trình quay lại từ đầu.

Chương trình được viết như sau:

```

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr dieu khien dong co: khi nhan nut start thi dong co quay
;khi nhan nut stop thi dong co ngung
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

```

start bit p2.0 ;phim sw1
stop bit p2.1 ;phim sw2
dk_dc equ p1.0 ;ngo ra dieu khien dong co

main1: org 0000h
       clr dk_dc ;cho dong co ngung
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con dieu khien dong co quay thuan
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
       jb start,$ ;cho nhan start
       jnb start,$ ;cho buong nut nhan

```

```

setb dk_dc ;cho dong co chay
jb stop,$ ;cho nhan stop
jnb stop,$ ;cho buong nut nhan
sjmp main1

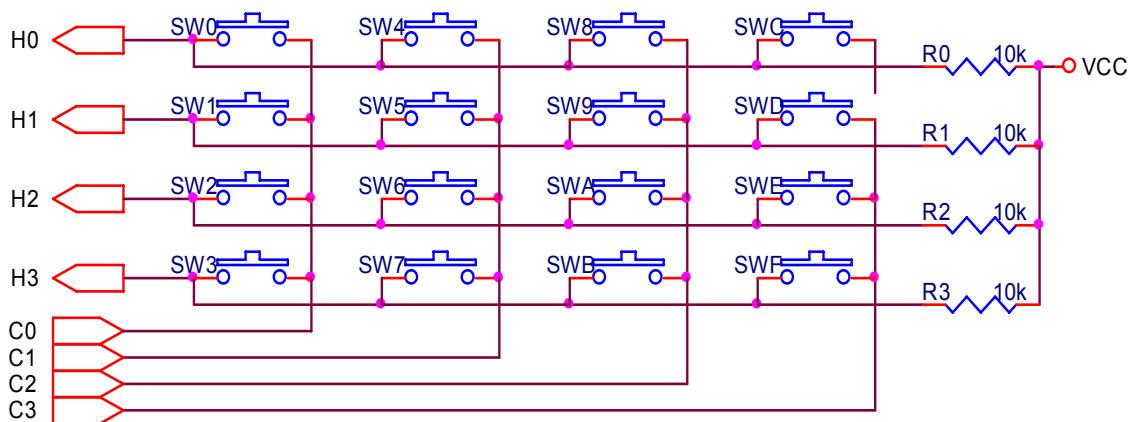
end

```

Hai nút nhấn được kết nối với 2 ngõ vào của port2. Bit P1.0 của Port1 kết nối điều khiển động cơ.

## 2 GIAO TIẾP VỚI MA TRẬN NHIỀU PHÍM

Khi số lượng nút nhấn nhiều ví dụ như 16 phím, 20 phím, hoặc nhiều hơn như bàn phím máy tính thì phải dùng cách kết nối kiểu ma trận để giảm bớt số lượng đường kết nối. Sơ đồ kết nối vi điều khiển với 16 nút nhấn như hình 7-9.



**Hình 7-9. Giao tiếp vi điều khiển với ma trận phím 4x4.**

Trong bàn phím ma trận  $4 \times 4$  sẽ có 4 hàng từ  $H[0:3]$  đóng vai trò là các ngõ vào bình thường ở mức H, và có 4 cột  $C[0:3]$  dùng để xuất mã quét.

4 hàng và 4 cột được kết nối với 1 port của vi điều khiển, vi điều khiển sẽ xuất mã quét ra các cột và sau đó đọc dữ liệu ở các hàng kết quả của quá trình thực hiện như bảng 7-3:

Mã quét xuất ra 4 cột				Nhập dữ liệu của 4 hàng				Nhấn Phím
C3	C2	C1	C0	H3	H2	H1	H0	
1	1	1	0	1	1	1	1	Không nhấn phím
1	1	1	0	1	1	1	0	Nhấn phím SW0
1	1	1	0	1	1	0	1	Nhấn phím SW1
1	1	1	0	1	0	1	1	Nhấn phím SW2
1	1	1	0	0	1	1	1	Nhấn phím SW3
<hr/>								
1	1	0	1	1	1	1	1	Không nhấn phím
1	1	0	1	1	1	1	0	Nhấn phím SW4
1	1	0	1	1	1	0	1	Nhấn phím SW5

1	1	0	1	1	0	1	1	Nhấn phím SW6
1	1	0	1	0	1	1	1	Nhấn phím SW7
<hr/>								
1	0	1	1	1	1	1	1	Không nhấn phím
1	0	1	1	1	1	1	0	Nhấn phím SW8
1	0	1	1	1	1	0	1	Nhấn phím SW9
1	0	1	1	1	0	1	1	Nhấn phím SWA
1	0	1	1	0	1	1	1	Nhấn phím SWB
<hr/>								
0	1	1	1	1	1	1	0	Không nhấn phím
0	1	1	1	1	1	1	0	Nhấn phím SWC
0	1	1	1	1	1	0	1	Nhấn phím SWD
0	1	1	1	1	0	1	1	Nhấn phím SWE
0	1	1	1	0	1	1	1	Nhấn phím SWF

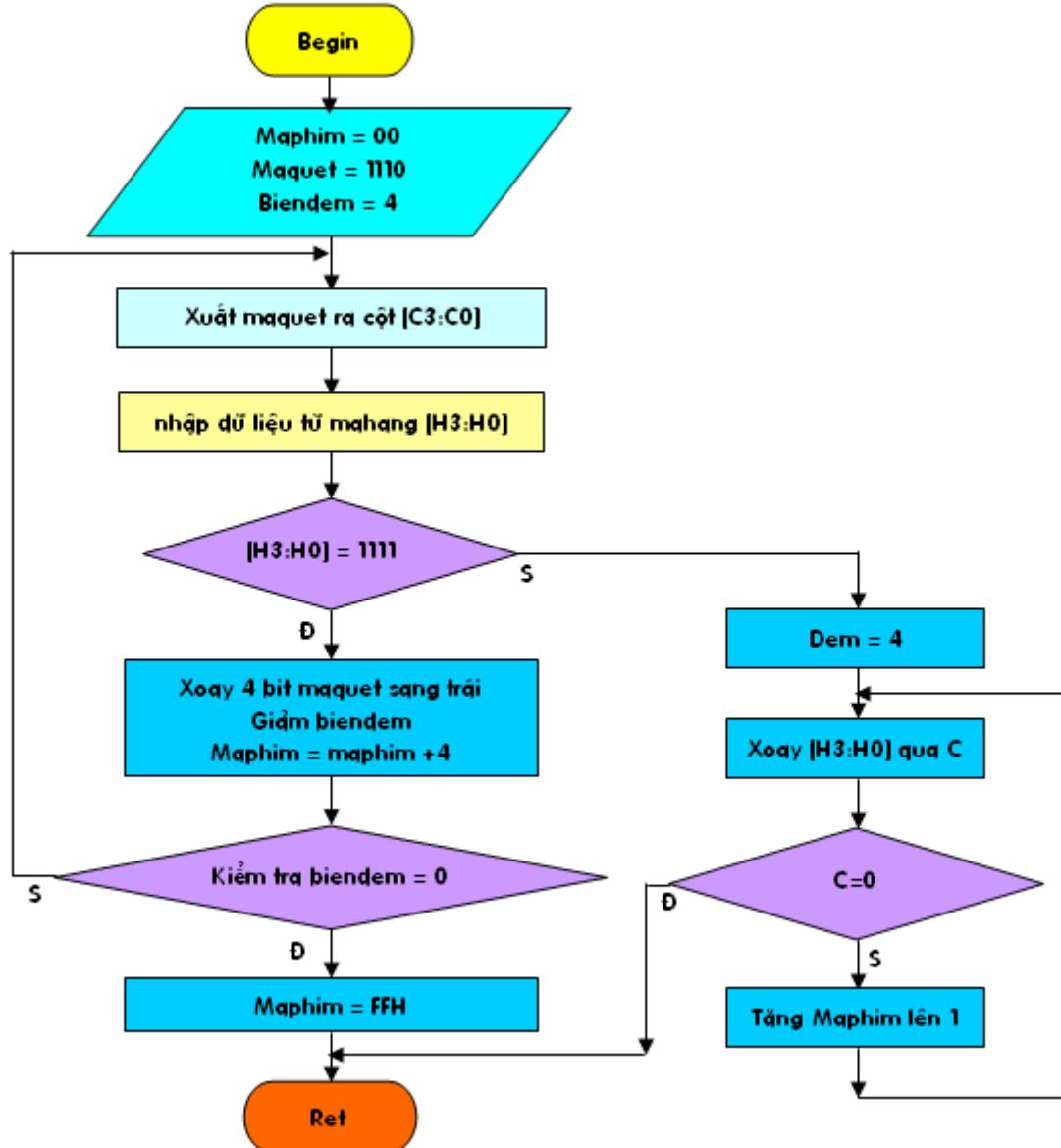
**Bảng 7-3. Mã quét xuất ra cột và mã hàng được đọc về.**

Trong các ứng dụng lớn nên dùng IC chuyên dùng quét bàn phím ma trận và quét led 7 đoạn hiển thị 8279 như đã trình bày ở chương trước.

Chương trình quét phím thường được viết dạng chương trình con và các chương trình khác sẽ gọi chương trình con quét phím để kiểm tra xem có sự tác động từ bên ngoài hay không và nếu có thì xử lý theo yêu cầu.

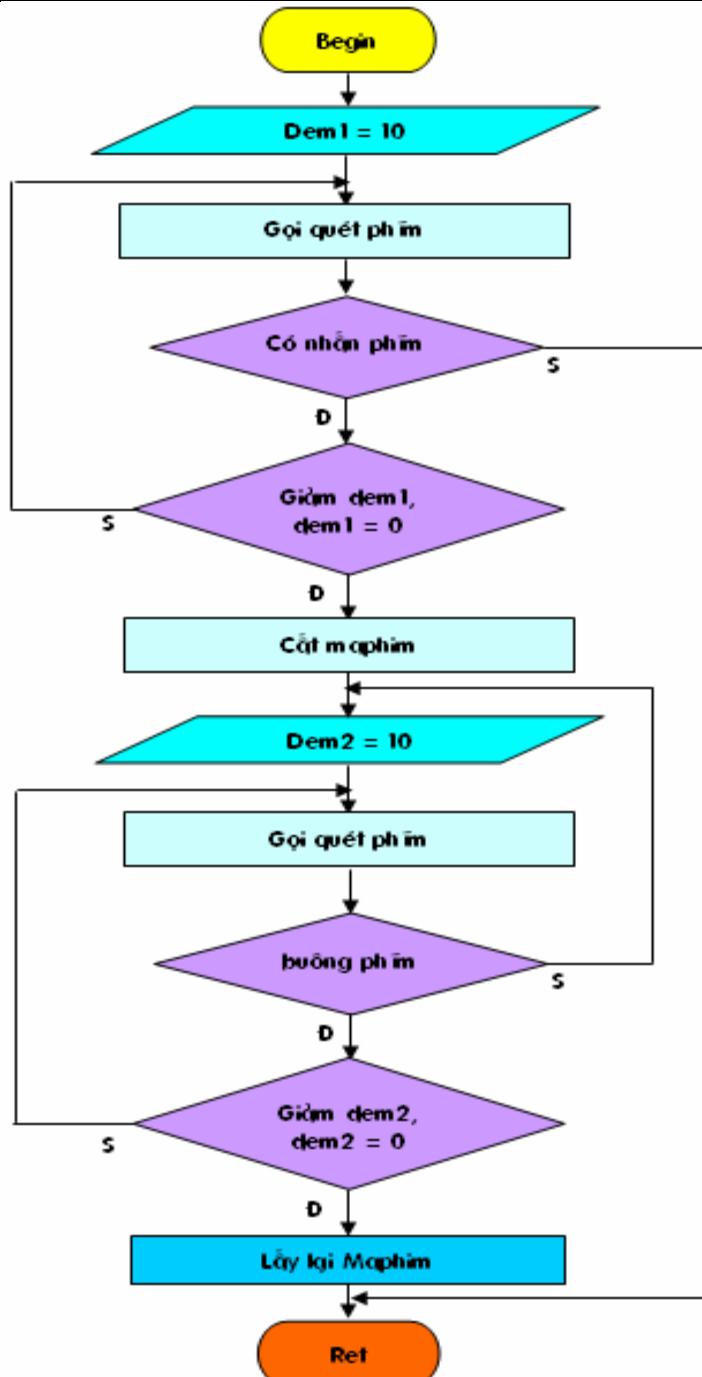
Khi ta nhấn phím thì thời gian nhấn phím khá dài từ vài chục ms đến hàng giây, trong khi đó tốc độ của vi xử lý rất cao nên khi ta nhấn phím thì vi xử lý sẽ thực hiện ngay lập tức và có thể thực hiện nhiều lần nếu thời gian nhấn phím dài. Điều này dẫn đến khi ta nhấn một phím ví dụ như chữ A thì sẽ có nhiều chữ A hiển thị trên màn hình – nguyên nhân do dội phím nhấn và do tốc độ thực hiện của vi xử lý nhanh, sau khi thực hiện xong công việc của phím nhấn đó rồi quay lại phát hiện ra phím vẫn còn nhấn và tiếp tục thực hiện tiếp cho đến khi nào buông phím thì ngừng.

Lưu đồ xuất mã quét, kiểm tra phím nhấn và tạo mã phím như hình 7-10:



Hình 7-10. Lưu đồ quét ma trận phím 4x4.

Lưu đồ chống dội phím như hình 7-11:



Hình 7-11. Lưu đồ chống dội sau khi quét phím.

Để chống dội phím ta thực hiện như sau: gọi chương trình con quét phím nếu không có nhấn phím thì thoát, nếu có thì tiến hành gọi lại chương trình con 10 lần rồi cắt mã phím nhấn, sau đó thực hiện chương trình quét chờ cho buông phím và phải kiểm tra 10 lần sau khi đã buông phím nhấn, lấy lại mã phím và thoát.

**Ví dụ 7:** Chương trình giao tiếp vi điều khiển với bàn phím ma trận 4x4 và 8 led 7 đoạn.

```

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;dung port3 ket noi voi ban fím ma tran 4x4
;dung port0 va port 2 ket noi dieu khien 8 led 7 doan quet
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    quet      equ      p2
    led7     equ      p0
  
```

```

        mtphim      equ    p3          ;ket noi voi ma tran ban phim
        maphim      equ    r5          ;luu tamma phim
        maquet     equ    11111110B   ;FEH

        org 0000h
        mov dptr,#ma7doan
        mov quet,#07fh           ;xuat ma quet chi cho 1 led sang
        mov led7,#7fh

main:    lcall keypres       ;goi chtr con quet phim
        cjne a,#0ffh,main1
        sjmp main

main1:   lcall gma_hthi    ;goi chtr con giao ma hien thi
        sjmp main

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;Chuong trinh con giao ma fim nhan va hien thi ra 1 led 7 doan
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
gma_hthi:  movc a,@a+dptr
            mov  led7,a
            ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;Chuong trinh con quet phim va chong doi phim
;su dung cac thanh ghi: R4, R5, R6, R7, A
;neu khong nhan thi (A) = FF, neu nhan thi (A) chua ma phim nhan
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
keypres:  mov  r4,#10          ;nhap so dem 10 lan
keypres1 : lcall KEY          ;Neu co phim an thi co c=1
          jc   pn1           ;kiem tra tiep neu c = 1
          ret             ;Neu khong co phim nhan thi co c=0

pn1:     djnz r4,keypres1   ;Quay ve lap lai chong nay
          push acc          ;Cat noi dung ma phim trong A

keypres2: mov  r4,#10          ;Nhaph so dem 10 lan cho nha phim
keypres3: lcall key          ;Co phim nhan hay khong
          jc   keypres2      ;Co thi kiem tra lai
          djnz r4,keypres3   ;Khong thi lap lai 50 lan va dam bao
          pop  acc          ;Khoi phuc lai gia tri cho A
          ret              ;ket thuc mot chuong trinh con
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;Chuong trinh con quet phim
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
key:     mov  r7,#maquet      ;bat dau voi cot so 0(feh)
        mov  r6,#4           ;Su dung r6 lam bo dem
        mov  maphim,#00

key1:    mov  mtphim,r7      ;xuat ma quet ra cot
        mov  a,mtphim
        anl  a,#0f0h
        cjne a,#0f0h,key2   ;xoai 4 bit thap la hang
                           ;co nhan fim thi nhay

        mov  a,r7
        rl   a
        mov  r7,a           ;xoay de chuyen den cot ke tiep

        mov  a,maphim
        add  a,#4
        mov  maphim,a

        djnz r6,key1        ;Neu nhu sau moi lan 1 cot ma khong

```

```

        clr    c           ;clr c neu nhu khong co phim duoc an
        mov    a,#0ffh      ;thoat voi ma trong a = FFh
        ret

key2:    swap   a
key4:    rrc    a           ;xoay sang phai tim bit 0
        jnc    key3      ;nhay neu (c)=0
        inc    maphim     ;tang ma fim len cot ke
        sjmp   key4      ;tiep tuc cho den khi duoc (C)=0

key3:    mov    a,maphim
        setb   c
        ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;khai bao du lieu ma phim
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ma7doan: db      0c0h,0f9h,0a4h,0b0h,099h,092h,082h,0f8h
          db      080h,090h,088h,083h,0c6h,0a1h,086h,08eh

end

```

Giải thích: chương trình này thực hiện quét bàn phím để kiểm tra phím nhấn, chống dội, tạo mã phím, giải mã phím và hiển thị trên 1 led 7 đoạn trong hệ thống 8 led kết nối theo phương pháp quét.

Bàn phím ma trận kết nối với port3, quét điều khiển 8 transistor dùng port2, điều khiển các đoạn dùng port0.

Chỉ dùng 1 led để hiển thị nên xuất mã quét ra là 7FH = 011111B để cho 1 transistor dẫn.

Mặc nhiên cho dấu chấm thập phân sáng trên led 7 đoạn.

Gọi chương trình kiểm tra xem có nhấn phím hay không?

Nếu không nhấn phím thì mã phím cuối cùng lưu trong thanh ghi A =FFH, nếu có nhấn phím thì tiến hành chống dội bằng cách thực hiện delay và kiểm tra lại xem phím nhấn có còn tác động hay không? Nếu không còn thì xem như lần nhấn vừa rồi không có tác dụng, nếu còn nhấn thì tiến hành kiểm tra xem nhấn phím nào và thiết lập mã phím tương ứng.

Mã phím sẽ được giải mã sang mã 7 đoạn và được hiển thị trên led.

Nếu nhấn phím mới thì mã của phím mới sẽ hiển thị còn mã phím trước mất.

**Bài tập 6:** Hãy mở rộng chương trình trên với yêu cầu như sau: mặc nhiên hiển thị số 0, khi nhấn phím mới thì phím cũ sẽ được dịch sang led kế bên trái như cách hiển thị trên máy tính (calculator).

**Bài tập 7:** Hãy dùng bàn phím ma trận điều khiển động cơ bước với các yêu cầu sau:

Một nút Start: khi nhấn nút này thì động cơ chạy.

Một nút Stop: khi nhấn nút này thì động cơ ngừng.

Một nút đảo chiều: khi nhấn nút này thì động cơ đảo chiều.

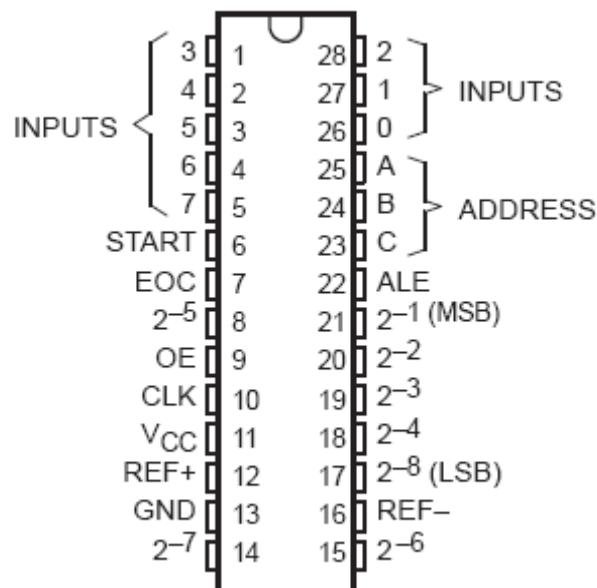
### III. VI ĐIỀU KHIỂN GIAO TIẾP VỚI VI MẠCH CHUYỂN ĐỔI ADC:

#### 1. GIAO TIẾP VI ĐIỀU KHIỂN VỚI ADC 0809:

Vi mạch chuyển đổi tín hiệu tương tự sang tín hiệu số có rất nhiều loại phân biệt theo số kênh và số bit. Một vi mạch chuyển đổi thường được sử dụng nhiều là ADC 0808 hoặc ADC 0809. Các thông số làm việc của vi mạch như sau:

- ADC 8 bit.
- Thời gian chuyển đổi 100μs.
- Độ giao tiếp với vi xử lý hoặc vi điều khiển.
- Các ngõ ra 3 trạng thái có chốt.
- Các ngõ vào địa chỉ có chốt.
- Dùng nguồn 5V.

Sơ đồ chân của IC ADC 0809 như hình 7-12:

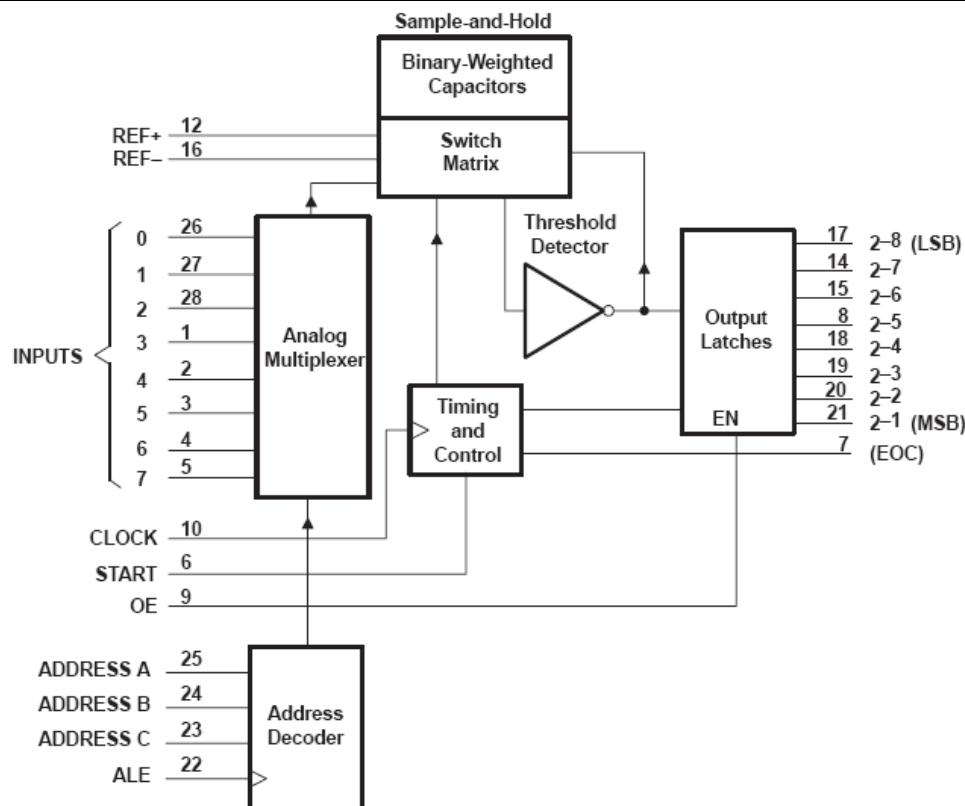


Hình 7-12. Sơ đồ chân IC ADC 0809.

Bảng trạng thái IC ADC 0809 như bảng 7-4:

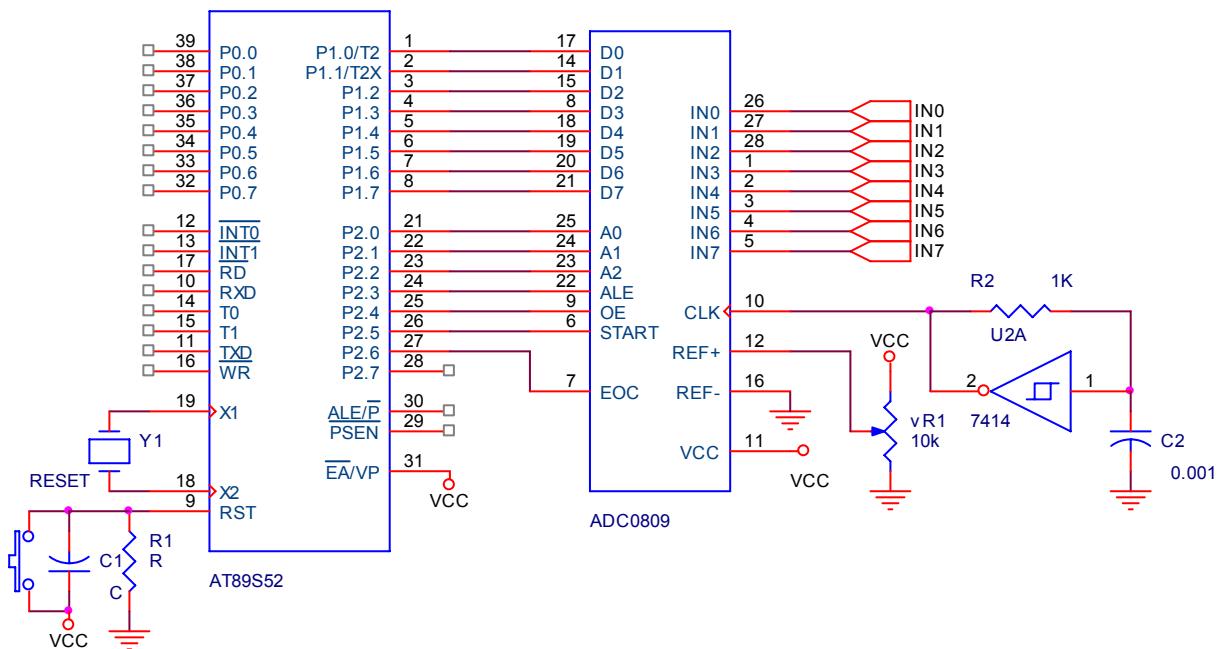
INPUTS			ALE	SELECTED ANALOG CHANNEL		
ADDRESS						
C	B	A				
L	L	L	↑	0		
L	L	H	↑	1		
L	H	L	↑	2		
L	H	H	↑	3		
H	L	L	↑	4		
H	L	H	↑	5		
H	H	L	↑	6		
H	H	H	↑	7		

Bảng 7-4. Bảng trạng thái chọn kênh ADC.



Hình 7-13. Sơ đồ khái niệm bên trong IC ADC 0809.

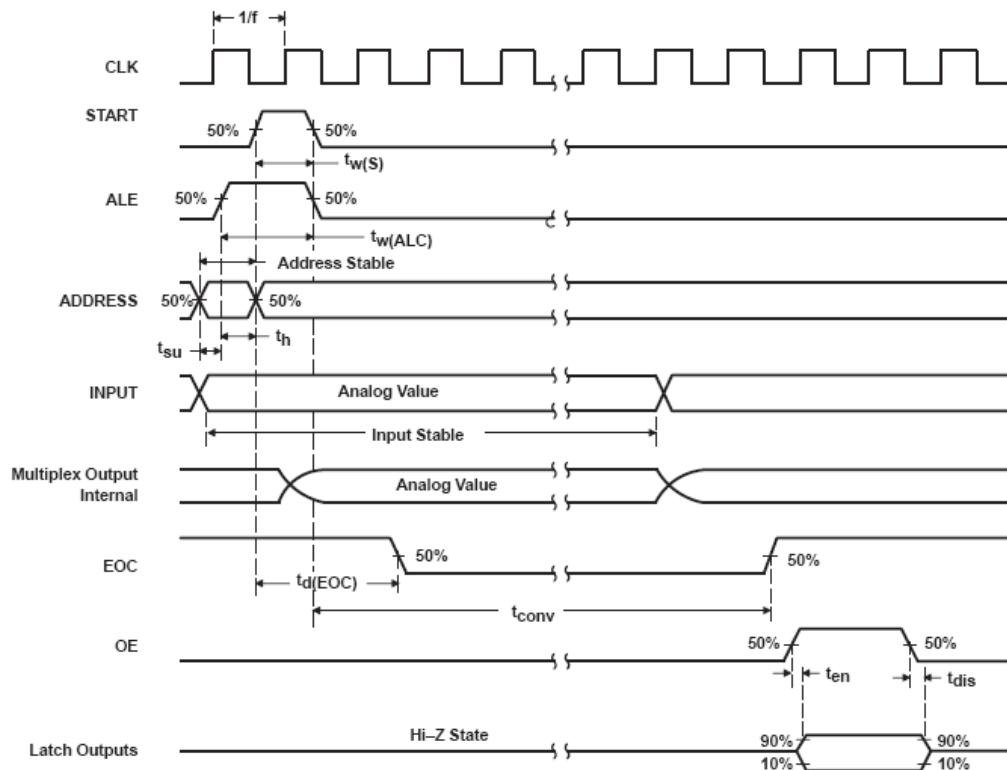
Sơ đồ kết nối vi điều khiển với IC ADC 0809 như hình 7-14:



Hình 7-14. Giao tiếp vi điều khiển với ADC 0809.

Trong hệ thống hình 7-14, 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 (D<sub>7</sub>:D<sub>0</sub>), 3 đường dùng để xuất địa chỉ chọn 1 trong 8 kênh (A<sub>2</sub>A<sub>1</sub>A<sub>0</sub>), 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.

Giản đồ thời gian của IC ADC0809 như hình 7-15:

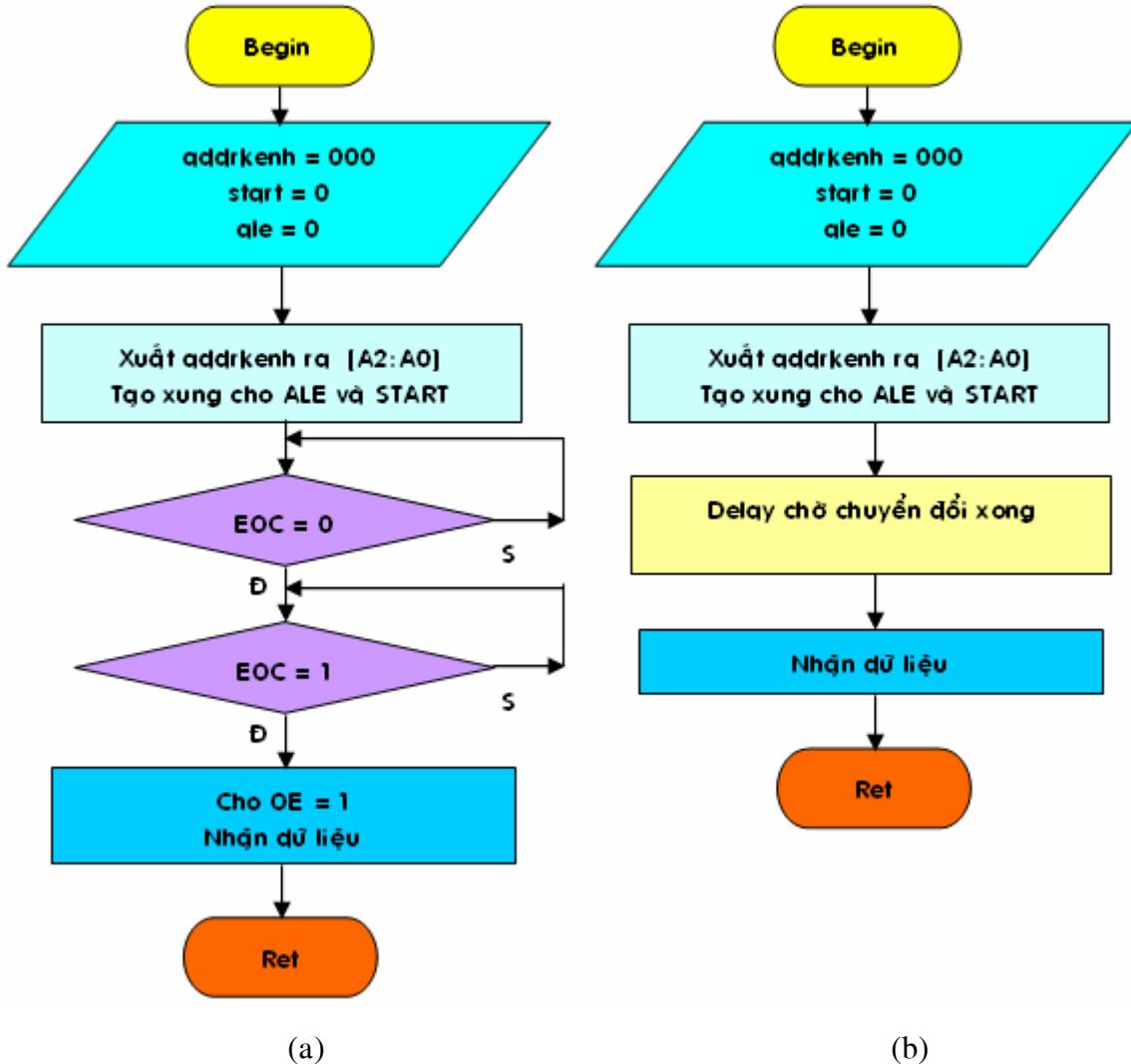


**Hình 7-15. Giản đồ thời gian của ADC 0809.**

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 nhận 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  $t_{CONV}$  lớn nhất là 116 $\mu$ s 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 7-16:



Hình 7-16. Lưu đồ điều khiển ADC 0809.

Lưu đồ (a) điều khiển chuyển đổi bằng cách chờ tín hiệu trả lời EOC, lưu đồ (b) thì dùng delay. Thời gian chờ phải lớn hơn thời gian chuyển đổi cho trong datasheet.

Ví dụ 7: Chương trình giao tiếp vi điều khiển với ADC 0809 và 8 led 7 đoạn.

Yêu cầu: dùng 1 kênh thứ 0 và dữ liệu sau khi chuyển đổi dạng số hex hiển thị trên 2 led.

Kết quả hiển thị nằm trong khoảng từ 00 đến FF.

```

quiet      equ p0      ;dieu khien quiet transistor
led7       equ p3      ;dieu khien cac doan a,b,c, ...
inadc     equ p1      ;nhap du lieu tu adc
control   equ p2      ;dieu khien adc
;ALE       bit  p0.3
start     bit  p2.3
main:
        org  0000h
        mov  dptr,#ma7doan
        lcall ctcd_adc          ;goi chtr con chuyen doi du lieu
        lcall gma_hex_bcd        ;goi chtrinh con giai ma so hex sang led 7 doan
        lcall delayhthi
        ljmp main                ;nhay ve chuyen doi tro lai
;  
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtrinh con chuyen so hex thanh so BCD va sau do thi chuyen

```

```

;ma BCD thanh ma 7 doan de hien thi so thap phan
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
gma_hex_bcd:
    mov    b,#10          ;chuyen so hex sang ma BCD
    div    ab
    mov    10h,b          ;luu hang don vi BCD

    mov    b,#10          ;(a) chua so hang tram, (b) chua hang chuc
    div    ab

    movc   a,@a+dptra
    mov    25h,a          ;cat so hang tram

    mov    a,b
    movc   a,@a+dptra
    mov    26h,a

    mov    a,10h
    movc   a,@a+dptra
    mov    27h,a
    ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con chuyen doi du lieu analog sang so ket qua luu trong A
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ctcd_adc:    mov    control,#0000000B      ;goi ra port
              setb   start           ;start = 1
              nop
              nop
              clr    start
              lcall  delayhthi        ;goi chtr con delay co hien thi
              mov    a,inadc         ;doc du lieu sau khi chuyen doi
              ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con co ghep chtr con hien thi
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
delayhthi:   mov    7eh,#20
pqn:         mov    7fh,#30
delpqn:     lcall  hthi           ;goi chuong trinh con hien thi
              djnz   7fh,delpqn
              djnz   7eh,pqn
              ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con hien thi ket qua dang so hex sau khi giai ma ra led
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hthi:        mov    led7,27h        ;lay du lieu tung kenh
              mov    quet,#11111110b   ;goi ma quet cho 1 led sang
              lcall  delay10
              mov    quet,#0ffh        ; tat het de chong lem

              mov    led7,26h        ;lay du lieu tung kenh
              mov    quet,#11111101b   ;goi ma quet cho 1 led sang
              lcall  delay10
              mov    quet,#0ffh        ; tat het de chong lem
              ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
delay10:    mov    7ch,#50h
              djnz   7ch,$
              ret

```

```
;khai báo ma 7 doan tu so '0' den so '9'
ma7doan:    db      0C0h,0F9h,0A4h,0B0h,099h,092h,082h,0F8h
              db      080h,090h,088h,083h,0c6h,0a1h,086h,08eh
end
```

Trong chương trình sử dụng các khai báo: “quet” và “led7” là 2 port giao tiếp điều khiển 8 led 7 đoạn, “inadc” là port kết nối với 8 đường dữ liệu số ngõ ra của adc 0809, “control” là port kết nối điều khiển chọn kênh và các tín hiệu điều khiển Start và ALE nối chung.

Trong chương trình sử dụng delay để chờ ADC chuyển đổi xong. Trong thời gian chờ thì vi điều khiển sẽ thực hiện quét 8 led 7 đoạn để hiển thị dữ liệu.

**Bài tập 8:** Hãy viết chương trình chuyển đổi kênh thứ 0 và kết quả hiển thị dạng số BCD trên 3 led.

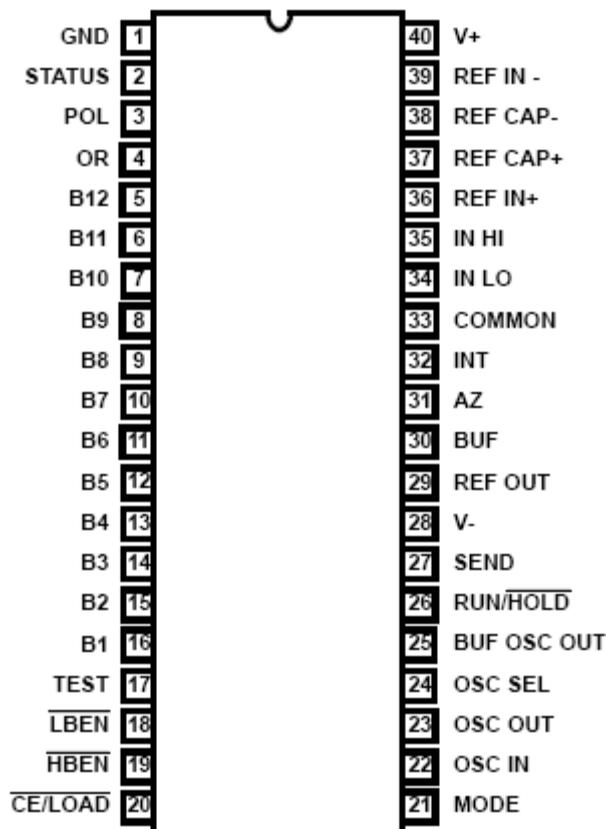
**Bài tập 9:** Hãy viết chương trình chuyển đổi ADC kênh thứ 0 và kênh thứ 1 hiển thị trên 3 led 7 đoạn.

## 2. GIAO TIẾP VI ĐIỀU KHIỂN VỚI ADC ICL7109.

Với vi mạch ADC 0809 chỉ chuyển đổi 8 bit, khi có những yêu cầu với số bit nhiều hơn thì vi mạch hiện đang được sử dụng phổ biến là ICL7109. Các thông số làm việc của vi mạch như sau:

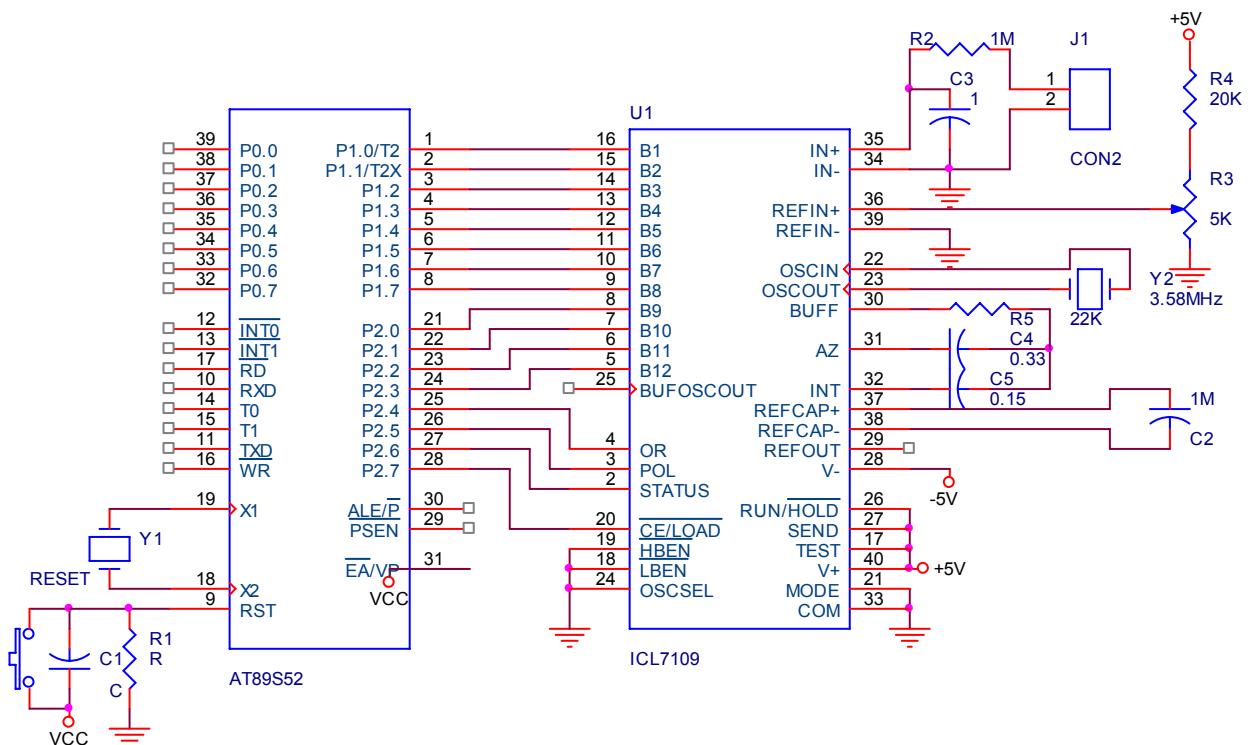
- ADC 2 độ dốc 12 bit.
- Các ngõ ra 3 trạng thái tương thích với TTL tổ chức theo byte.
- Có các ngõ vào RUN/HOLD và ngõ ra STATUS dùng để giám sát và điều khiển quá trình chuyển đổi.
- Nhiều có giá trị thấp  $15\mu\text{V}_{\text{P-P}}$ .
- Dòng ngõ vào rất nhỏ khoảng  $1\text{pA}$ .
- Tốc độ chuyển đổi 30 lần / 1giây.
- Mạch dao động tích hợp bên trong sử dụng thạch anh  $3.58\text{MHz}$ .
- Dùng nguồn  $5\text{V}$ .

Sơ đồ chân của IC ADC ICL 7109 như hình 7-17



Hình 7-17. Sơ đồ chân IC ADC ICL 7109.

Sơ đồ giao tiếp vi điều khiển với ADC ICL 7109 như hình 7-18:

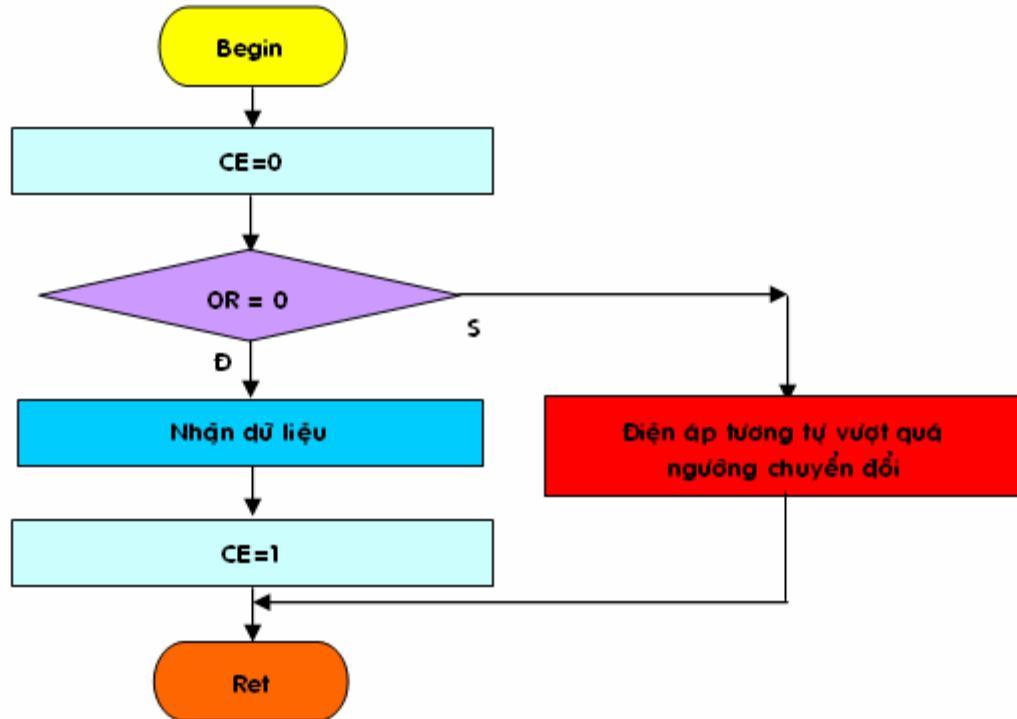


Hình 7-18. Giao tiếp vi điều khiển AT89S52 với IC ADC ICL 7109.

Trong sơ đồ hình 7-18, vi điều khiển dùng 12 đường kết nối với 12 đường dữ liệu số ngõ ra của ADC 7109 từ B12-B1. 4 đường điều khiển bao gồm: đường điều khiển chọn chip CE, đường

kiểm tra cực tính dương khi POL ở mức H, đường kiểm tra trạng thái STATUS ở mức H khi dữ liệu đã được chốt và đường kiểm tra tín hiệu OR cho biết tín hiệu tương tự ngõ vào có vượt quá ngưỡng hay không.

Đối với ADC 7109 thì quá trình điều khiển chuyển đổi đơn giản hơn vì nó chỉ có 1 kênh và chuyển đổi liên tục. Lưu đồ chuyển đổi như hình 7-19



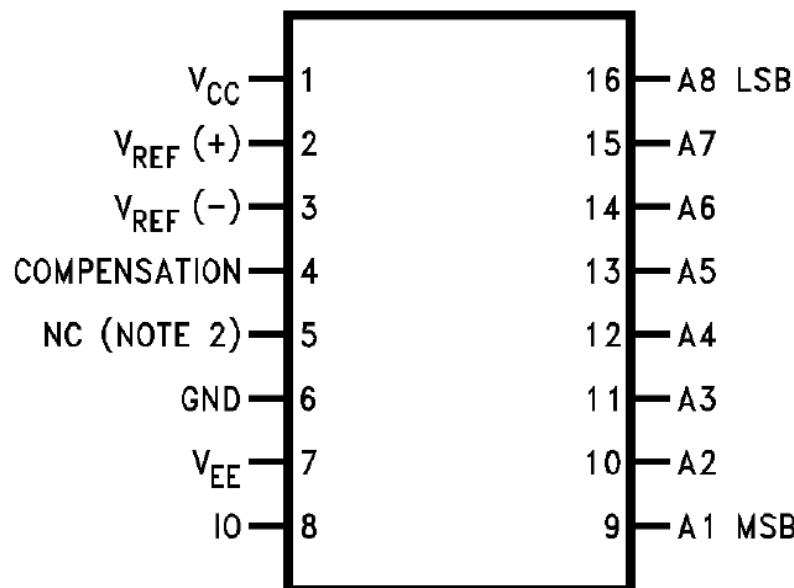
Hình 7-19. Lưu đồ điều khiển ADC ICL 7109.

Khi muốn đọc dữ liệu từ ADC thì điều khiển cho CE xuống mức 0, tiến hành kiểm tra xem tín hiệu báo tràn để biết có bị tràn hay không: nếu bị tràn thì dữ liệu số nhận về không có ý nghĩa vì chúng không đúng, nếu không bị tràn thì tiến hành nhận dữ liệu, cho CE lên mức 1 trở lại. Kết thúc 1 chu kỳ và tiến hành xử lý kết quả.

### 3. GIAO TIẾP VI ĐIỀU KHIỂN VỚI DAC 0808:

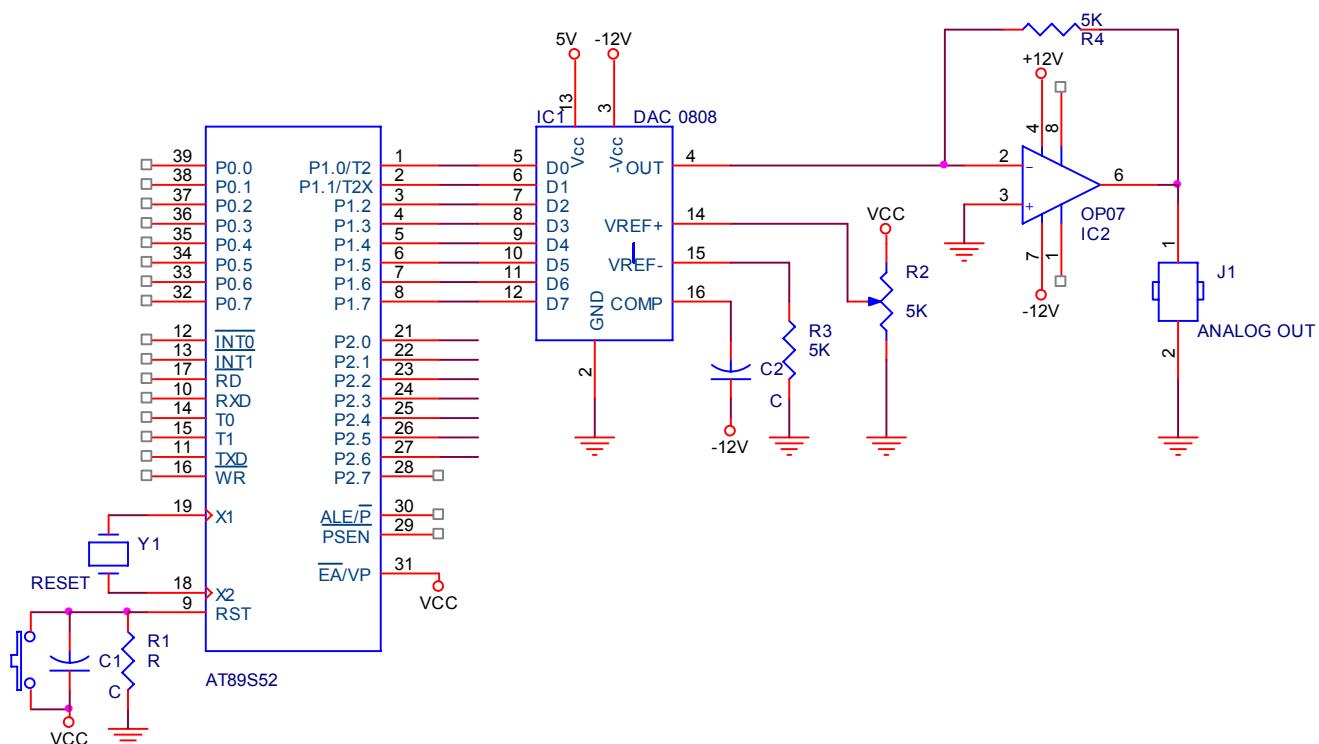
DAC là vi mạch chuyển đổi tín hiệu số sang tín hiệu tương tự, vi mạch thường được sử dụng phổ biến là DAC 0808 - 8 bit:

Sơ đồ chân DAC 0808 như hình 7-20:



Hình 7-20. Sơ đồ chân IC DAC 0808.

Sơ đồ giao tiếp vi điều khiển với DAC 0808 như hình 7-21:



Hình 7-21. Sơ đồ giao tiếp vi điều khiển với DAC 0808.

Trong sơ đồ giao tiếp vi điều khiển dùng port 1 để xuất dữ liệu số đến DAC 0808, tín hiệu ngõ ra của DAC được đưa qua mạch khuếch đại dùng op-amp OP07 hoặc op-amp 741, biến trở R2 dùng để chỉnh độ phân giải cho DAC.

#### IV. VI ĐIỀU KHIỂN GIAO TIẾP VỚI RELAY VÀ ĐỘNG CƠ BƯỚC:

##### 1 GIAO TIẾP VI ĐIỀU KHIỂN VỚI RELAY:

Trong các ứng dụng giao tiếp điều khiển các thiết bị sử dụng nguồn 220VAC thì phải dùng các linh kiện giao tiếp trung gian. Các linh giao tiếp bao gồm relay, SCR, TRIAC, ...

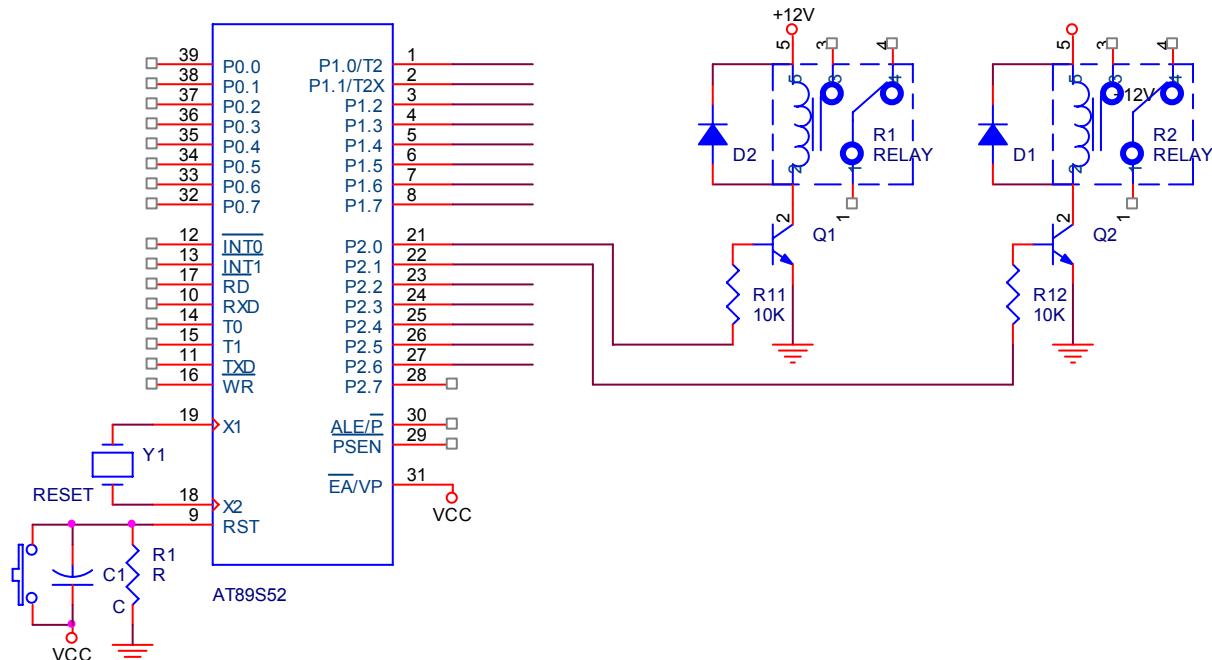
Trong phần này sẽ trình bày phần giao tiếp với relay.

Khi sử dụng relay cần phải biết các thông số: điện áp làm việc của cuộn dây, dòng làm việc của cuộn dây và dòng điện làm việc của tiếp điểm để điều khiển tải.

Sơ đồ mạch giao tiếp vi điều khiển với 2 relay như hình 7-22:

Trong sơ đồ sử dụng relay dùng nguồn 12V, transistor chọn có dòng làm việc  $I_C$  phải lớn hơn dòng làm việc của cuộn dây relay.

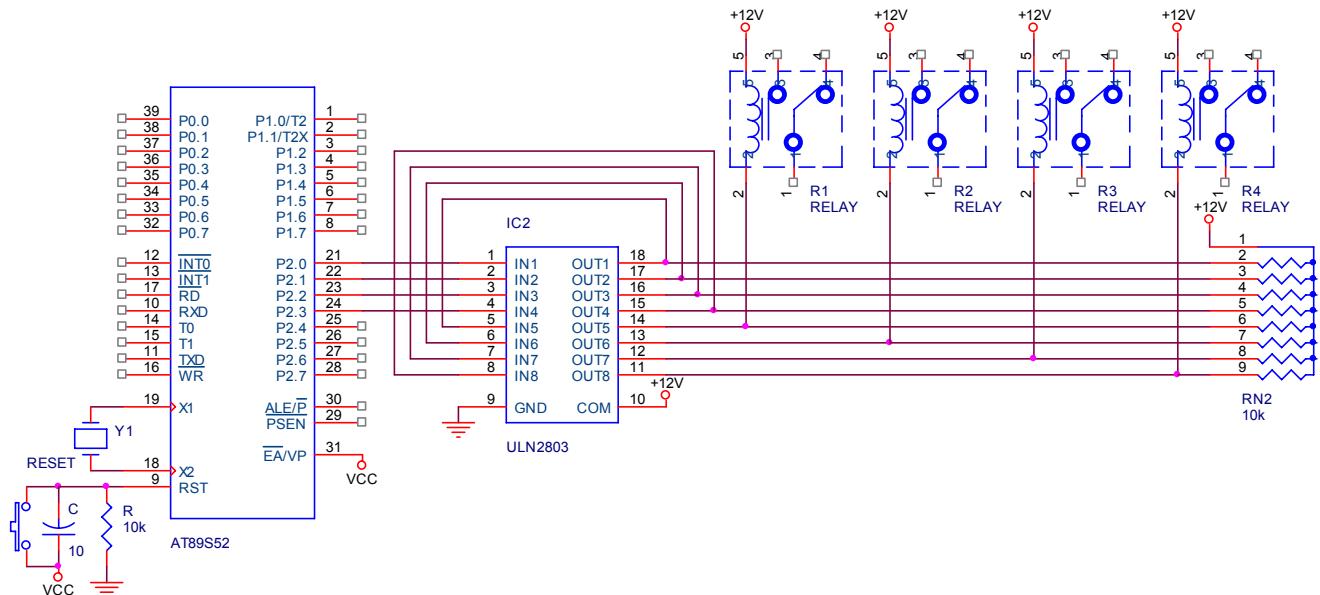
Mỗi relay thường có 1 tiếp điểm thường hở và 1 tiếp điểm thường đóng.



Hình 7-22. Sơ đồ giao tiếp vi điều khiển với relay.

Trong sơ đồ hình 7-22 có 1 khuyết điểm: nếu ta sử dụng tiếp điểm thường hở của relay để điều khiển tải là động cơ thì khi vi điều khiển mới được cấp điện thì mạch auto reset sẽ thực hiện làm các ngõ ra của các port ở mức H  $\Rightarrow$  làm các transistor dẫn  $\Rightarrow$  relay đóng  $\Rightarrow$  động cơ chạy. Hết thời gian reset thì vi điều khiển mới thực hiện chương trình tắt động cơ, sự kiện này sẽ xảy ra tương tự khi ta nhấn nút reset hệ thống. Điều này nguy hiểm cho các đối tượng được điều khiển bằng động cơ.

Để khắc phục khuyết điểm trên bằng cách thêm 1 cổng đảo ở ngõ ra trước khi điều khiển transistor hoặc dùng IC giao tiếp ULN2803 như hình 7-23:



Hình 7-23. Sơ đồ giao tiếp vi điều khiển với 4 relay qua IC giao tiếp ULN2803.

Trong sơ đồ trên ULN 2803 là IC giao tiếp đệm đảo: với mỗi ngõ ra chịu dòng chạy vào lên đến 500mA, điện áp làm việc lên đến 50V hoặc 95V tùy theo loại. Ngõ vào tương thích với họ TTL. Có diode bảo vệ ở ngõ ra. Trong mạch sử dụng 4 ngõ ra của vi điều khiển được đưa qua 2 cổng NOT nên khi vi điều khiển bị reset thì trạng thái ngõ ra của các port bằng 1, ngõ ra của IC ULN 2803 cũng bằng 1 tương ứng với nguồn 12V nên relay không dẫn. Ngược lại nếu ngõ ra của vi điều khiển bằng 0 thì relay dẫn.

## 2. GIAO TIẾP VI ĐIỀU KHIỂN VỚI ĐỘNG CƠ BƯỚC

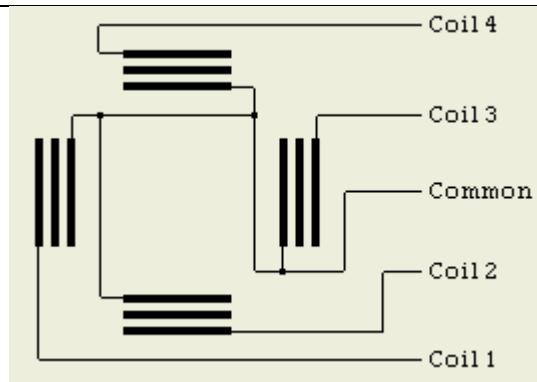
Động cơ bước có chức năng điều khiển các thiết bị di chuyển với khoảng cách ngắn ví dụ như điều khiển di chuyển đầu đọc đĩa. Tuỳ thuộc vào công suất của động cơ mà sử dụng các mạch giao tiếp cho phù hợp. Thường thì các động cơ bước luôn đi kèm với mạch điều khiển và chỉ nhận lệnh từ vi xử lý hay vi điều khiển.

Hình động cơ bước loại nhỏ dùng trong ổ đĩa mềm 1.2 MB như hình 7-24:



Hình 7-24. Hình động cơ bước loại nhỏ.

Sơ đồ dây bên trong động cơ bước như hình 7-25:

**Hình 7-25. Các cuộn dây bên trong động cơ bước.**

Động cơ bước đơn cực có 5 ngõ ra: trong đó có 4 đầu dây coil1÷coil4 dùng để điều khiển còn đầu dây common dùng để nối nguồn cung cấp. Kí hiệu các màu dây theo qui định như hình 7-26:

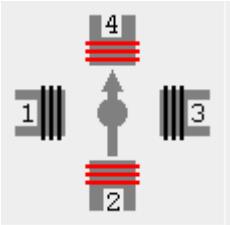
**Hình 7-26. Các cuộn dây bên trong động cơ bước.**

Động cơ bước đơn cực có 6 ngõ ra: trong đó có 4 đầu dây coil1÷coil4 dùng để điều khiển, 2 đầu dây còn lại chính là dây common được tách ra làm 2, khi dùng phải nối cả 2 với nguồn cung cấp. Hai dây common này có cùng màu.

Cách xác định các cuộn dây của động cơ bước theo trình tự như sau:

<b>Nối cuộn dây common với nguồn cung cấp</b>	
<b>Chọn một đầu dây và nối đầu dây đó xuống mass và giả sử đầu dây đó là dây thứ 4 – hãy xem hình bên phải tương ứng.</b>	
<b>Giữ nguyên bước trên</b>	
<b>Tiếp theo chọn 1 đầu dây trong 3 đầu còn lại và nối với mass – quan sát thấy rotor quay nhẹ 1 bước theo chiều kim đồng hồ thì đó là cuộn dây thứ 3.</b>	
<b>Tháo cuộn dây thứ 3 ra khỏi mass</b>	
<b>Tiếp theo chọn 1 đầu dây trong 2 đầu còn lại và nối với mass – quan sát thấy rotor quay nhẹ 1 bước ngược chiều kim đồng hồ thì đó là cuộn dây thứ 1.</b>	

Cuộn dây còn lại là cuộn dây thứ 2 nhưng ta cũng có thể kiểm tra bằng cách nối nó với mass – quan sát thay rotor không quay.



Cách xác định cuộn dây common bằng cách chọn 1 đầu dây tùy ý làm dây common, đo điện trở với 4 đầu dây còn lại: nếu 4 giá trị điện trở đều bằng nhau thì đầu dây đã chọn là đầu common, nếu không bằng thì chọn đầu dây khác và thực hiện lại sẽ tìm ra được đầu dây common.

Trình tự điều khiển động cơ bước gồm có 3 kiểu: kiểu kích 1 cuộn dây, kiểu kích 2 cuộn dây và cách điều khiển nữa bước.

- a. Kiểu điều khiển kích 1 cuộn dây như hình 7-27:

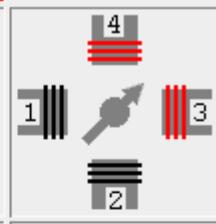
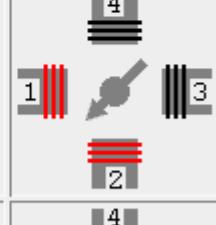
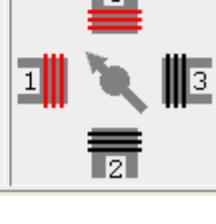
Step	Coil 4	Coil 3	Coil 2	Coil 1	
a.1	on	off	off	off	
a.2	off	on	off	off	
a.3	off	off	on	off	
a.4	off	off	off	on	

Hình 7-27. Điều khiển kích 1 cuộn dây.

Trong hình trên thì mỗi 1 thời điểm chỉ có 1 cuộn dây ở trạng thái “on”. Động cơ sẽ quay mỗi bước khi thay đổi trạng thái điều khiển.

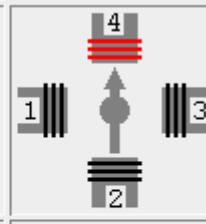
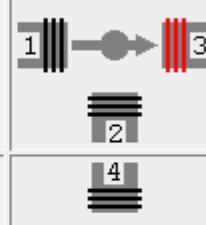
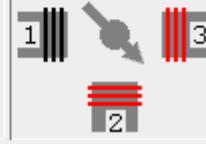
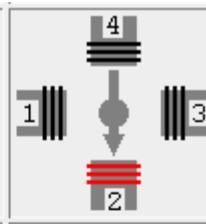
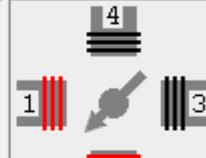
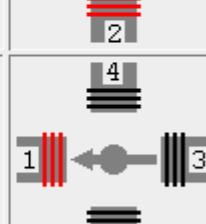
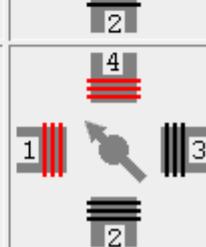
Muốn quay đảo chiều thì trình tự điều khiển tiến hành theo chiều ngược lại.

- b. Kiểu điều khiển kích 2 cuộn dây như hình 7-28:

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	

**Hình 7-28. Điều khiển kích 2 cuộn dây.**

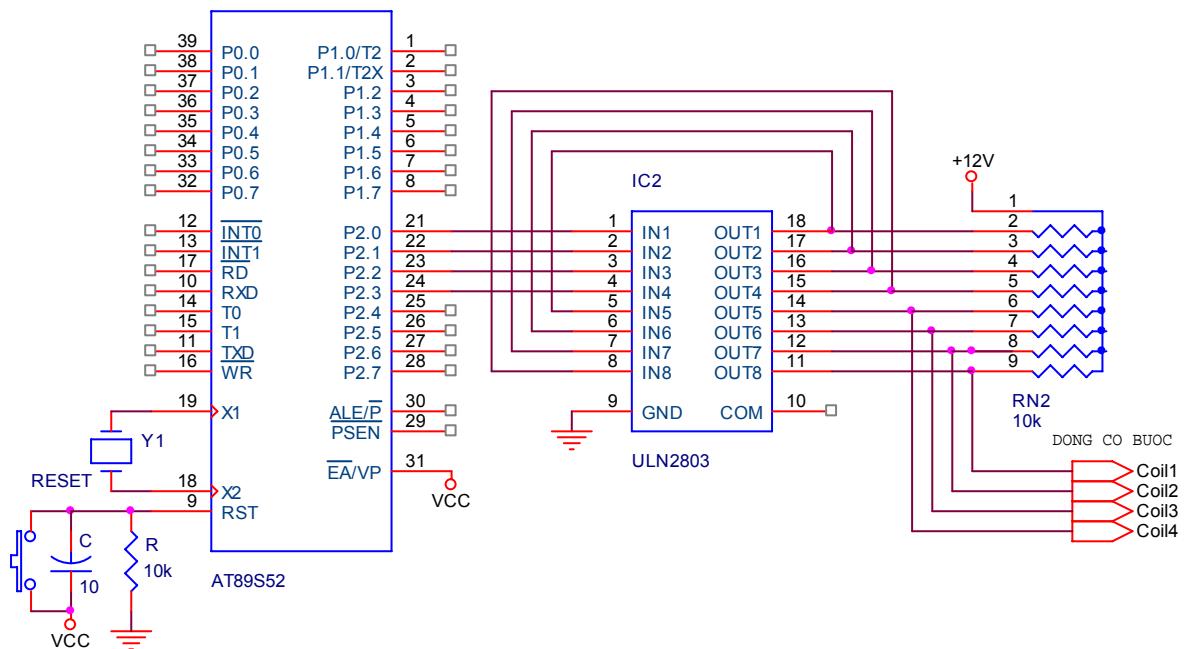
- c. Phối hợp 2 cách điều khiển để điều khiển nữa bước như hình 7-29:

Step	Coil 4	Coil 3	Coil 2	Coil 1	
a.1	on	off	off	off	
b.1	on	on	off	off	
a.2	off	on	off	off	
b.2	off	on	on	off	
a.3	off	off	on	off	
b.3	off	off	on	on	
a.4	off	off	off	on	
b.4	on	off	off	on	

Hình 7-29. Điều khiển phối hợp cả hai.

Trong cách điều khiển nǔa bước có tất cả 8 trạng thái khác nhau.

Sơ đồ mạch giao tiếp vi điều khiển với động cơ bước như hình 7-30:



**Hình 7-30. Sơ đồ giao tiếp vi điều khiển với động cơ bước qua IC ULN2803.**

Chân common của động cơ bước phải nối với nguồn mà động cơ làm việc. Nếu động cơ công suất lớn với dòng làm việc lớn hơn dòng của IC ULN2803 thì nên dùng transistor.

**Ví dụ 8:** Chương trình điều khiển động cơ bước quay liên tục 1 chiều: dùng kiểu điều khiển kích 1 cuộn dây:

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr dieu khien step motor quay lien tuc theo 1 chieu, thoi gian delay dai thi dong co quay cham
;thoi gain nho thi dong co quay nhanh, thoi gian qua nho thi dong co khong dap ung duoc se dung yen
;dung 1 port de dieu khien motor qua ic giao tiep 2803 - dung 4 bit thap hoac 4 bit cao
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
        outstep    equ      p1
        org      0000h

main:      mov      outstep,#10001000b
            lcall    delay
            mov      outstep,#01000100b
            lcall    delay
            mov      outstep,#00100010b
            lcall    delay
            mov      outstep,#00010001b
            lcall    delay
            sjmp   main

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con delay
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
delay:     mov      r6,#0
del:       mov      r7,#0
            djnz   r7,$
            djnz   r6,del
            ret

end
```

**Ví dụ 9:** Chương trình điều khiển động cơ bước quay 1 vòng rồi ngừng:

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```

;chtr dieu khien step motor quay 1 vong roi ngung luon
;dung 1 port de dieu khien motor qua ic giao tiep 2803
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

        outstep    equ     p1
        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 8 buoc

main1:   mov      a,r0
        movc   a,@a+dptra
        mov      outstep,a
        lcall   delay

        inc      r0
        anl      00h,#03h          ;anl r0 voi 00000011b
        djnz   r1,main1

        djnz   r2,main2
        sjmp   $                  ;dung lai
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con delay
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
delay:   mov      r6,#10h
del:     mov      r7,#0
        djnz   r7,$
        djnz   r6,del
        ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;khai du lieu dieu khien dong co buoc
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
datastep: db      10001000b
           db      01000100b
           db      00100010b
           db      00010001b
end

```

Động cơ sử dụng trong bài ví dụ này có thông số là 1 vòng 200 bước.

**Bài tập 10:** Hãy thiết kế mạch điều khiển 1 động cơ bước với các nút nhấn là Start, Stop, Inv và giao tiếp với 4 led 7 đoạn. Viết chương trình điều khiển động cơ bước với yêu cầu như sau: khi nhấn nút Start thì động cơ chạy và hiển thị số bước trên led 7 đoạn, khi nhấn nút stop thì động cơ ngừng, khi nhấn nút đảo chiều thì động cơ đảo chiều.

## V. VI ĐIỀU KHIỂN GIAO TIẾP VỚI LCD:

### 1. GIỚI THIỆU LCD:

Ở các phần giao tiếp với led 7 đoạn có hạn chế vì chỉ hiển thị được các số từ 0 đến 9 hoặc số hex từ 0 đến F – không thể nào hiển thị được các thông tin kí tự khác, nhưng chúng sẽ được hiển thị đầy đủ trên LCD.

LCD có rất nhiều dạng phân biệt theo kích thước từ vài kí tự đến hàng chục kí tự, từ 1 hàng đến vài hàng. Ví dụ LCD 16x2 có nghĩa là có 2 hàng, mỗi hàng có 16 kí tự. LCD 20x4 có nghĩa là có 4 hàng, mỗi hàng có 20 kí tự.

LCD 16x2 như hình 7-31:



**Hình 7-31. Hình của LCD**

### 2. SƠ ĐỒ CHÂN CỦA LCD:

LCD có nhiều loại và số chân của chúng cũng khác nhau nhưng có 2 loại phổ biến là loại 14 chân và loại 16 chân, sự khác nhau là các chân nguồn cung cấp, còn các chân điều khiển thì không thay đổi, khi sử dụng loại LCD nào thì phải tra datasheet của chúng để biết rõ các chân. Sơ đồ chân của LCD như bảng sau:

Pin No	Name	I/O	Description
1	Vss	Power	GND
2	Vdd	Power	+5V
3	Vo	Analog	Contrast Control
4	RS	Input	Register Select
5	R/W	Input	Read/Write
6	E	Input	Enable ( <i>Strobe</i> )
7	D0	I/O	Data <i>LSB</i>
8	D1	I/O	Data
9	D2	I/O	Data
10	D3	I/O	Data
11	D4	I/O	Data
12	D5	I/O	Data
13	D6	I/O	Data
14	D7	I/O	Data <i>MSB</i>

**Bảng 7-6. Các chân của LCD**

Trong 14 chân của LCD được chia ra làm 3 dạng tín hiệu như sau:

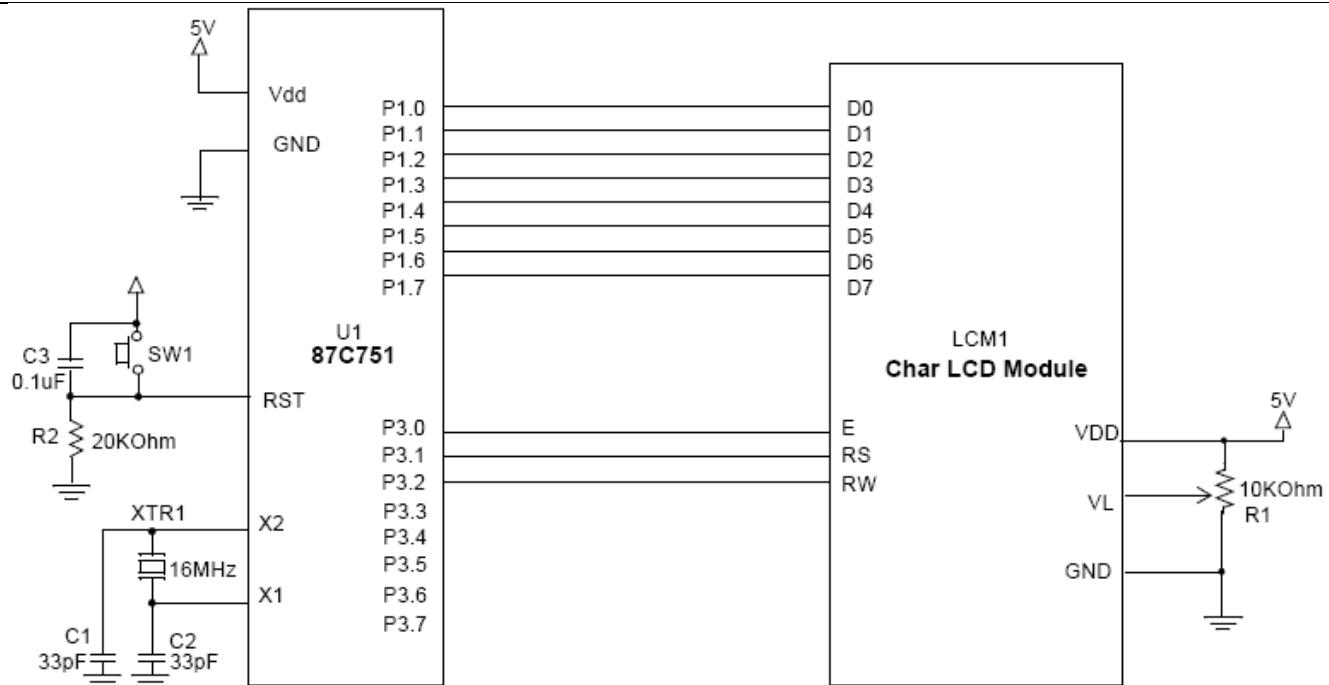
**Các chân cấp nguồn:** Chân số 1 là chân nối mass (0V), chân thứ 2 là Vdd nối với nguồn +5V. Chân thứ 3 dùng để chỉnh contrast thường nối với biến trở.

**Các chân điều khiển:** Chân số 4 là chân RS dùng để điều khiển lựa chọn thanh ghi. Chân R/W dùng để điều khiển quá trình đọc và ghi. Chân E là chân cho phép dạng xung chốt.

**Các chân dữ liệu D7-D0:** Chân số 7 đến chân số 14 là 8 chân dùng để trao đổi dữ liệu giữa thiết bị điều khiển và LCD.

### 3. SƠ ĐỒ MẠCH GIAO TIẾP VI ĐIỀU KHIỂN VỚI LCD:

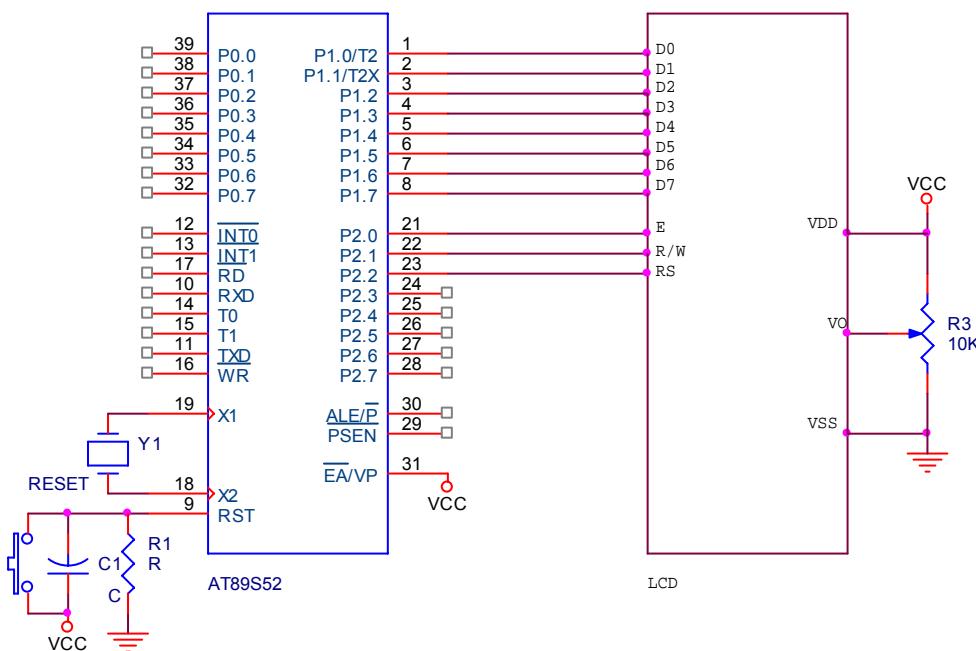
Trong phần này sẽ trình bày phần giao tiếp vi điều khiển với LCD. Hình 7-32 sẽ trình bày kết nối dùng vi điều khiển 87C751 của công ty “HANTRONIX” từ tài liệu trên Internet.



**HANTRONIX**  
INC.

Hình 7-32. Giao tiếp vi điều khiển 87C751 với LCD.

Có thể dùng vi điều khiển AT89S52 để điều khiển LCD với sơ đồ kết nối như hình 7-33:



Hình 7-33. Giao tiếp vi điều khiển AT89S52 với LCD.

**Chú ý:** khi lần đầu sử dụng LCD thì phải chỉnh biến trở sau cho các ký tự hiển thị trên LCD thì dừng lại.

#### 4. CÁC LỆNH ĐIỀU KHIỂN LCD

Để điều khiển LCD thì có các IC chuyên dùng được tích hợp bên dưới LCD có mã số 447801 đến các IC 447809. Trong IC này có bộ nhớ RAM dùng để lưu trữ dữ liệu cần hiển thị và thực hiện việc điều khiển LCD hiển thị.

Các điều khiển bao gồm các lệnh được liệt kê ở bảng sau:

Instruction	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	Description	Clocks
NOP	0	0	0	0	0	0	0	0	0	0	No Operation	0
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears display & sets address counter to zero.	165
Cursor Home	0	0	0	0	0	0	0	0	1	0	Sets address counter to zero, returns shifted display to original position. DDRAM contents remains unchanged.	3
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction, and specifies automatic shift.	3
Display Control	0	0	0	0	0	0	1	D	C	B	Turns display (D), cursor on/off (C) or cursor blinking(B).	3
Cursor/display shift	0	0	0	0	0	1	S/C	R/L	0	0	Moves cursor and shift display. DDRAM contents remains unchanged.	3
Function Set	0	0	0	0	1	DL	N	M	G	0	Sets interface data width(DL), number of display lines (N,M) and voltage generator control (G).	3
Set CGRAM Addr	0	0	0	1	Character Generator RAM				Sets CGRAM Address			
Set DDRAM Addr	0	0	1	Display Data RAM Address				Sets DDRAM Address				3
Busy Flag & Addr	0	1	BF	Address Counter				Reads Busy Flag & Address Counter				0
Read Data	1	0	Read Data				Reads data from CGRAM or DDRAM					3
Write Data	1	1	Write Data				Writes data from CGRAM or DDRAM					3

**Bảng 7-7. Các lệnh của LCD**

**Lệnh xoá màn hình “Clear Display”:** khi thực hiện lệnh này thì LCD sẽ bị xoá và bộ đếm địa chỉ được xoá về 0.

**Lệnh di chuyển con trỏ về đầu màn hình “Cursor Home”:** khi thực hiện lệnh này thì bộ đếm địa chỉ được xoá về 0, phần hiển thị trở về vị trí gốc đã bị dịch trước đó. Nội dung bộ nhớ RAM hiển thị DDRAM không bị thay đổi.

**Lệnh thiết lập lối vào “Entry mode set”:** lệnh này dùng để thiết lập lối vào cho các ký tự hiển thị, bit ID = 1 thì con trỏ tự động tăng lên 1 mỗi khi có 1 byte dữ liệu ghi vào bộ hiển thị, khi ID = 0 thì con trỏ sẽ không tăng: dữ liệu mới sẽ ghi đè lên dữ liệu cũ. Bit S = 1 thì cho phép dịch chuyển dữ liệu mỗi khi nhận 1 byte hiển thị.

**Lệnh điều khiển con trỏ hiển thị “Display Control”:** lệnh này dùng để điều khiển con trỏ (cho hiển thị thì bit D = 1, tắt hiển thị thì bit D = 0), tắt mở con trỏ (mở con trỏ thì bit C = 1, tắt con trỏ thì bit C = 0), và nhấp nháy con trỏ (cho nhấp nháy thì bit B = 1, tắt thì bit B = 0).

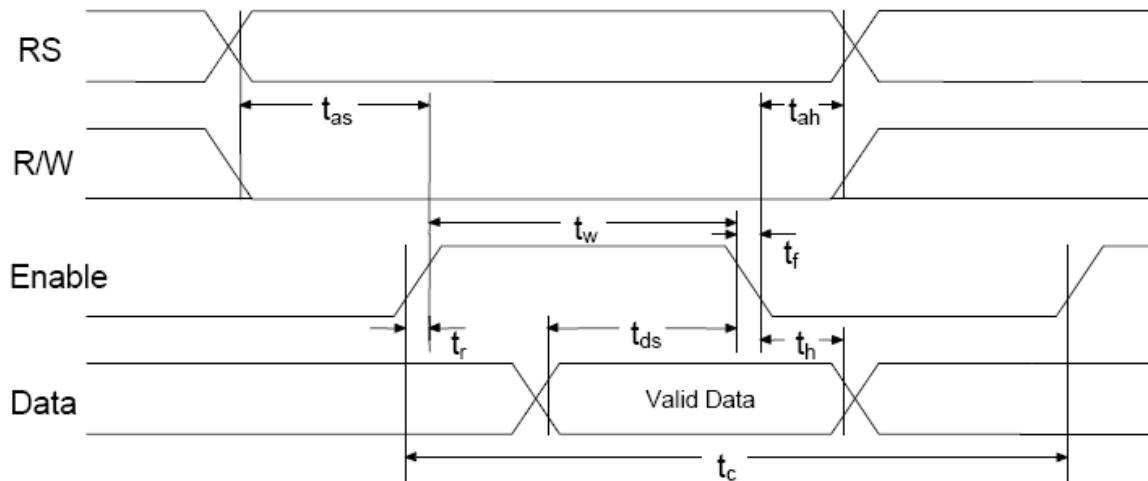
**Lệnh di chuyển con trỏ “Cursor /Display Shift”:** lệnh này dùng để điều khiển di chuyển con trỏ hiển thị dịch chuyển (SC = 1 cho phép dịch chuyển, SC = 0 thì không cho phép), hướng dịch chuyển (RL = 1 thì dịch phải, RL = 0 thì dịch trái). Nội dung bộ nhớ DDRAM vẫn không đổi.

**Lệnh thiết lập địa chỉ cho bộ nhớ RAM phát ký tự “Set CGRAM Addr”:** lệnh này dùng để thiết lập địa chỉ cho bộ nhớ RAM phát ký tự.

**Lệnh thiết lập địa chỉ cho bộ nhớ RAM hiển thị “Set DDRAM Addr”:** lệnh này dùng để thiết lập địa chỉ cho bộ nhớ RAM lưu trữ các dữ liệu hiển thị.

Hai lệnh cuối cùng là lệnh đọc và lệnh ghi dữ liệu LCD.

Dạng sóng các tín hiệu khi thực hiện ghi dữ liệu vào LCD như hình 7-34:

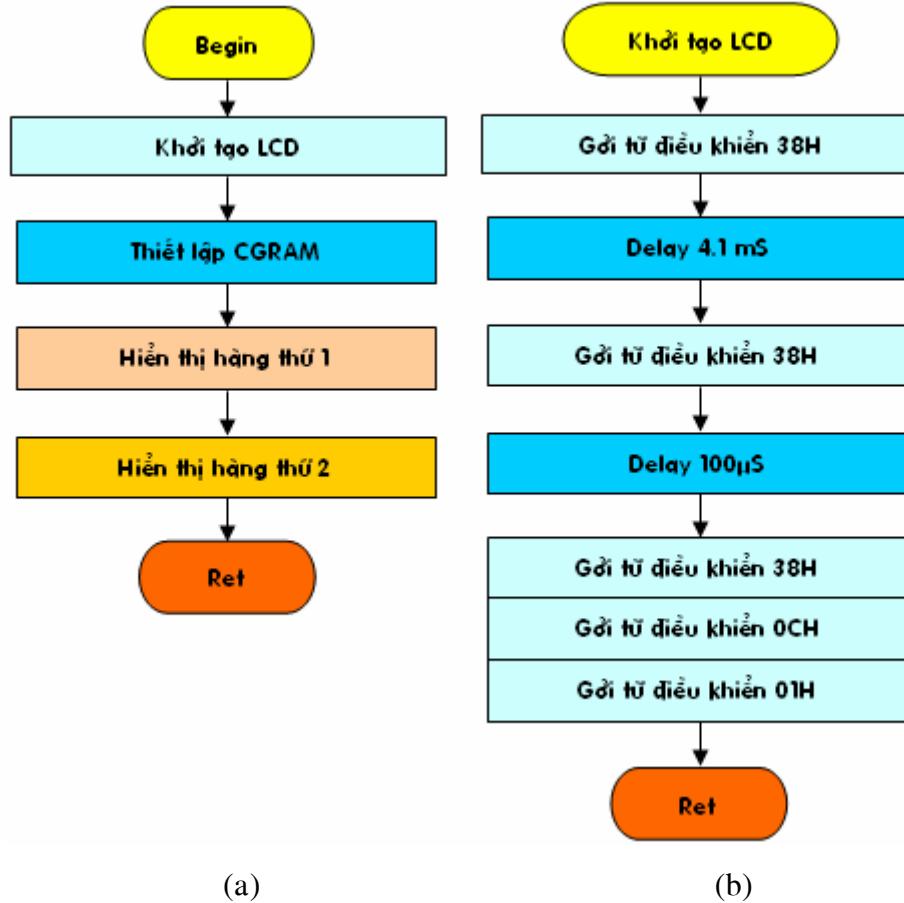
**Hình 7-34. Dạng sóng điều khiển của LCD.**

Nhìn vào dạng sóng ta có thể thấy được trình tự điều khiển như sau:

- Điều khiển tín hiệu RS.
- Điều khiển tín hiệu R/W xuống mức thấp.
- Điều khiển tín hiệu E lên mức cao để cho phép.
- Xuất dữ liệu D7÷D0.
- Điều khiển tín hiệu E về mức thấp.
- Điều khiển tín hiệu R/W lên mức cao trở lại.

##### **5. LƯU ĐỒ ĐIỀU KHIỂN LCD:**

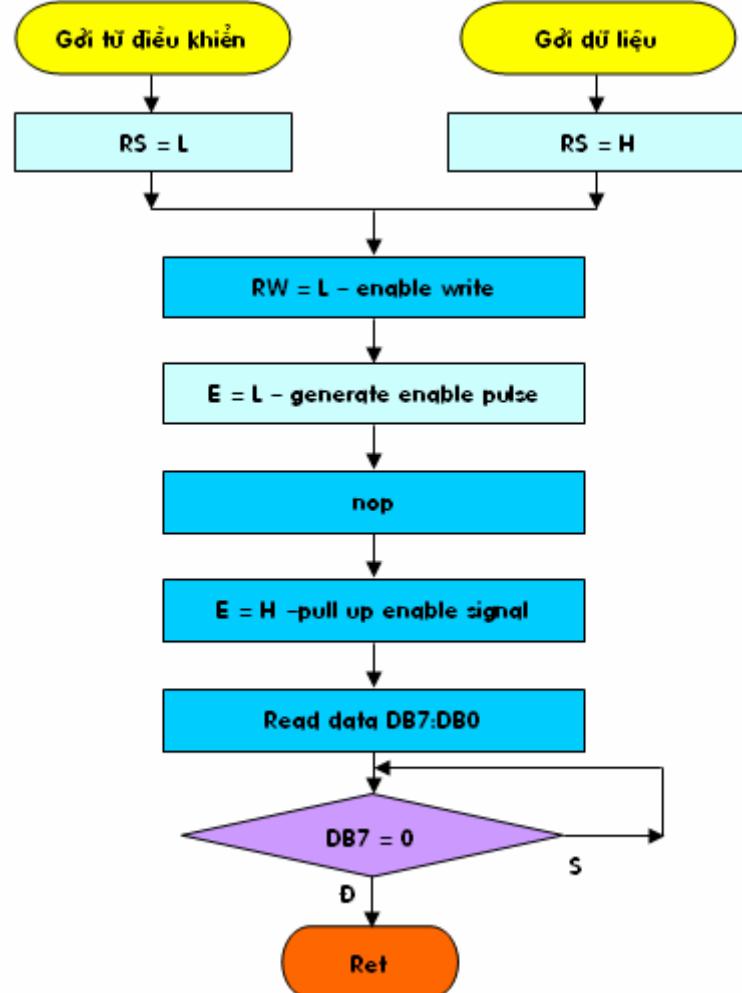
Để điều khiển LCD thì phải biết trình tự điều khiển được xây dựng theo các lưu đồ như hình 7-35:

**Hình 7-35. Lưu đồ điều khiển LCD.**

Trong lưu đồ hình (a) đó là lưu đồ chính bao gồm quá trình khởi tạo LCD, thiết lập địa chỉ của vùng nhớ RAM và thực hiện quá trình gởi dữ liệu ra LCD hiển thị ở hàng thứ 1 và tiếp theo là hàng thứ 2.

Lưu đồ hình (b) là lưu đồ cho biết trình tự thực hiện quá trình khởi tạo LCD bao gồm các bước: gởi mã điều khiển 38H ra LCD và thực hiện delay ít nhất là 4.1 ms. Tiếp tục thực hiện lần thứ 2 với thời gian delay 100 $\mu$ s. Cuối cùng gởi liên tục các mã điều khiển 38H, 0CH, và 01H để thiết lập cấu hình hoạt động cho LCD.

Lưu đồ thực hiện gởi từ điều khiển hay gởi dữ liệu ra LCD để hiển thị như hình sau:



Hình 7-36. Lưu đồ xuất lệnh hoặc dữ liệu ra LCD.

Trong lưu đồ có 2 đầu vào: đầu vào thứ nhất nếu gởi từ điều khiển đến LCD, đầu vào thứ 2 là gởi dữ liệu cần hiển thị đến LCD – sự khác nhau này thể hiện qua tín hiệu điều khiển RS.

Các bước còn lại thì giống nhau và sau khi tiến hành gởi xong từ điều khiển/dữ liệu thì đọc lại từ điều khiển/dữ liệu để kiểm tra bit DB7 để xem từ điều khiển/dữ liệu đã được nhận xong chưa nếu chưa thì phải chờ, nếu đã nhận xong thì thoát và sẵn sàng để nhận từ điều khiển/dữ liệu tiếp theo.

#### 6. CHƯƠNG TRÌNH ĐIỀU KHIỂN LCD DÙNG VI ĐIỀU KHIỂN 89S52:

Chương trình sau sẽ điều khiển LCD hiển thị 2 hàng với nội dung được cho ở cuối chương trình:

##### Ví dụ 10:

```

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh dieu khien LCD 16X2 tren kit vi dieu khien
;DUNG PORT 0 KET DOI VOI CAC DUONG DU LIEU CUA LCD P0-7 ->D0-7
;DUNG 3 BIT CUA PORT2: P20,P21,P22 DIEU KHIEN E,R/W,RS
;tren man hinh LCD se hien thi noi dung moi
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      E          BIT      P2.0
      rw         BIT      P2.1
      rs         BIT      P2.2
      byteout   equ      p0
  
```

```

ORG 0000H
mov 0a2h,#0
LCALL khtaolcd      ;khoi tao lcd
LCALL first_line     ;goi chtr con hien thi hang thu nhat
LCALL scond_line     ;goi chtr con hien thi hang thu hai
SJMP $


;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con khoi tao LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
khtaolcd:    setb e          ;Enable
              clr rs         ;RS low
              clr rw         ;RW low

              MOV a,#38h      ;tu dieu khien LCD
              LCALL KTAO
              LCALL ddelay41    ;delay 4.1 mSec

              MOV A,#38h      ;function set
              LCALL KTAO
              LCALL ddelay100   ;delay

              MOV A,#38h      ;function
              LCALL KTAO

              MOV A,#0ch       ;tu dieu khien display on
              LCALL KTAO
              MOV A,#01h       ;tu dieu khien Clear display
              LCALL KTAO

              MOV A,#06h       ;tu dieu khien entry mode set
              LCALL KTAO

              MOV A,#80h       ;thiet lap dia chi LCD (set DD RAM)
              LCALL KTAO

              MOV A,#0fh       ;enable display cursor
              LCALL KTAO

              RET

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con delay 4.1 ms
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ddelay41:    mov r6,#90h
del412:    mov r7,#200
              djnz r7,$
              djnz r6,del412
              ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con delay 255 microgiay
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ddelay100:   mov r7,#00
              djnz r7,$
              ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;Feed command/data to the LCD module
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
command_byte:
              clr rs           ;RS low for a command byte
              ljmp bdelay

```

```

data_byte:    setb   rs           ;RS high for a data byte
bdelay:      clr    rw           ;R/W low for a write mode
            clr    e
            nop
            setb   e           ;Enable pulse
            nop
            nop
            mov    byteout,#0ffh ;configure port1 to input mode
            setb   rw           ;set RW to read
            clr    rs           ;set RS to command
            clr    e            ;generate enable pulse
            nop
            nop
            setb   e
            lcall  ddelay100
            ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;Chuong trinh con hien thi noi dung hang thu 1 tren LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
first_line:   MOV    A,#080h       ;set DDRAM
              LCALL KTAO
              MOV    DPTR,#FLINE_DATA
              lcall  Write
              ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;Chuong trinh con hien thi noi dung hang thu 2 tren LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
scond_line:   mov    a,#0c0h       ;set DDRAM
              LCALL KTAO
              mov    dptr,#sline_data
              lcall  write
              ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con khai tao LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
KTAO:        mov    byteout,a
              lcall  command_byte
              RET
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con goi data hien thi ra LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
write:       MOV    A,#0
              MOVC  A,@a+dptr
              CJNE  A,#99h,Writea
              RET
Writea:      mov    byteout,a
              acall  data_byte
              inc    dptr
              SJMP  Write
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;    Data bytes
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
FLINE_DATA:  DB     'NGUYEN DINH PHU ',099h
SLINE_DATA:  DB     'DAI HOC SPKT HCM',099h
END

```

**Bài tập 11:** Hãy thiết kế mạch điều khiển 1 LCD giống như trên và 1 bàn phím ma trận  $4 \times 4$  và viết chương trình sau cho khi nhấn phím số nào thì trên màn hình hiển thị số đó tại vị trí tận cùng bên trái, các số trước dịch sang trái.

**Bài tập 12:** Hãy thiết kế mạch điều khiển 1 LCD giống như trên và 1 bàn phím ma trận  $4 \times 4$  và viết chương trình đồng hồ hiển thị giờ phút giây, ngày tháng năm.

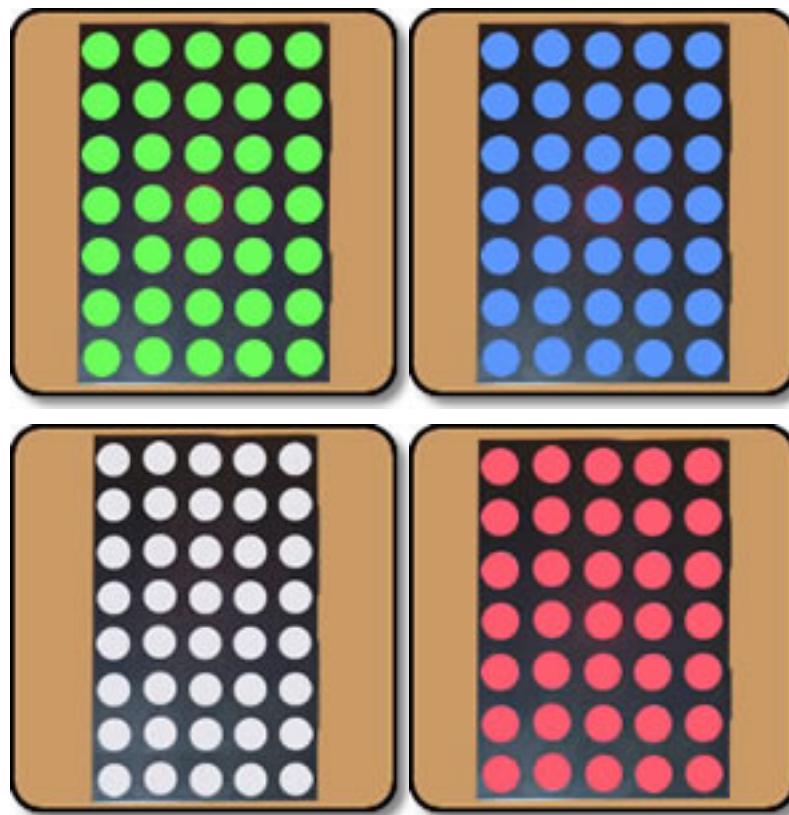
## VI. VI ĐIỀU KHIỂN GIAO TIẾP VỚI LED MA TRẬN:

### 1 GIỚI THIỆU LED MA TRẬN

Trong quảng cáo ở nơi công cộng thường sử dụng led ma trận, trong phần này sẽ giới thiệu và trình bày cách giao tiếp vi điều khiển với led ma trận và phần mềm điều khiển led ma trận.

Led ma trận là 1 tập các led đơn được bố trí theo dạng ma trận cho phép hiển thị được các ký tự bất kỳ. Các nhà chế tạo đã tích hợp theo các dạng ma trận tính theo led  $5 \times 7$  (5 cột và 7 hàng) hoặc  $8 \times 8$  (8 hàng và 8 cột) và tính theo kích thước và phân biệt theo màu của led, cuối cùng là phân biệt led sáng trong nhà (indoor) hay led sáng ngoài trời (outdoor). Led sáng trong nhà thì không thể dùng được ngoài trời vì ánh sáng mặt trời chiếu vào làm chúng ta không nhìn thấy rõ, chỉ có led outdoor mới thấy rõ, và dĩ nhiên led outdoor sẽ có giá thành cao hơn.

Hình 7-37 giới thiệu led ma trận  $5 \times 7$  với 4 màu khác nhau:



Hình 7-37. Led ma trận  $5 \times 7$ .

### 2 CẤU TẠO VÀ PHƯƠNG PHÁP ĐIỀU KHIỂN LED MA TRẬN

Led ma trận là một tập hợp các led đơn với cách kết nối như sau: theo hàng thì các anode nối chung, theo cột thì các cathode nối chung, với led ma trận  $5 \times 7$  thì có 7 hàng và 5 cột.

Anode nối với mức H và cathode nối mức L thì led sáng, các trường hợp còn lại thì led tắt.

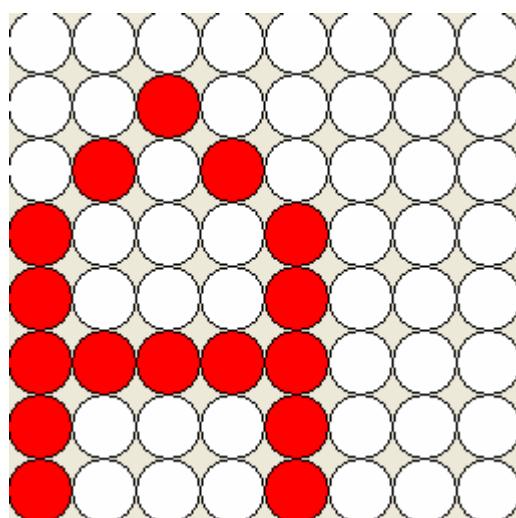
Hãy quan sát hình 7-38 led ma trận  $8 \times 8$  đang hiển thị chữ A, có 2 cách hiển thị chữ A:

Cách 1: dữ liệu gởi ra hàng, mã quét gởi ra cột;

Mỗi kí tự có 5 byte mã:

- Byte mã thứ nhất gởi ra 8 hàng (led nào sáng thì bit tương ứng bằng 1, led nào tắt thì bit tương ứng bằng 0) và cho cột thứ nhất xuống mức L, các cột còn lại ở mức H.
- Byte mã thứ hai gởi ra 8 hàng, cột thứ hai xuống mức L.
- Byte mã thứ ba gởi ra 8 hàng, cột thứ ba xuống mức L.
- Byte mã thứ tư gởi ra 8 hàng, cột thứ tư xuống mức L.
- Byte mã thứ năm gởi ra 8 hàng, cột thứ năm xuống mức L.

Do tốc độ thực hiện nhanh nên chúng ta sẽ nhìn thấy cả chữ A sáng.



Hình 7-38. Sáng chữ A.

Nếu hàng trên cùng là H0 và hàng dưới cùng là H7 thì mã của chữ A và mã quét được tóm tắt như bảng sau:

MÃ ĐIỀU KHIỂN HÀNG											MÃ QUÉT ĐIỀU KHIỂN CỘT				
H7	H6	H5	H4	H3	H2	H1	H0	Số hex	C0	C1	C2	C3	C4		
1	1	1	1	1	0	0	0	F8	0	1	1	1	1	Cột thứ 1 sáng	
0	0	1	0	0	1	0	0	24	1	0	1	1	1	Cột thứ 2 sáng	
0	0	1	0	0	0	1	0	22	1	1	0	1	1	Cột thứ 3 sáng	
0	0	1	0	0	1	0	0	24	1	1	1	0	1	Cột thứ 4 sáng	
1	1	1	1	1	0	0	0	F8	1	1	1	1	0	Cột thứ 5 sáng	

Bảng 7-8. Mã chữ A

Để thực hiện quét cột và dễ dàng cho việc mở rộng cột khi kết nối nhiều led ma trận với nhau thì thanh ghi dịch thường được sử dụng.

Với ma trận  $8 \times 8$  thì thời gian sáng của 1 cột là  $1/8$  chu kỳ quét, thời gian tắt là  $7/8$ , chính vì thế led sẽ sáng không rõ nếu làm việc với dòng và áp định mức. Để tăng cường độ sáng thì phải tăng áp và dòng. Bình thường led làm việc với dòng từ  $10mA$  đến  $20mA$  và áp là  $2V$ , nhưng với phương pháp quét cho 1 led  $8 \times 8$  thì dòng phải tăng lên khoảng 8 lần từ  $80mA$  đến  $160mA$  tương ứng.

**Cách tính toán như sau:** với dòng làm việc bình thường (không quét) chọn là  $20mA$ , áp làm việc định mức  $2V$ . Khi đó điện trở của mỗi led là:

$$R_{LED} = \frac{U}{I} = \frac{2V}{20mA} = 100\Omega$$

Khi dùng phương pháp quét thì dòng tức thời phải bằng  $160mA$ , khi đó dòng trung bình:

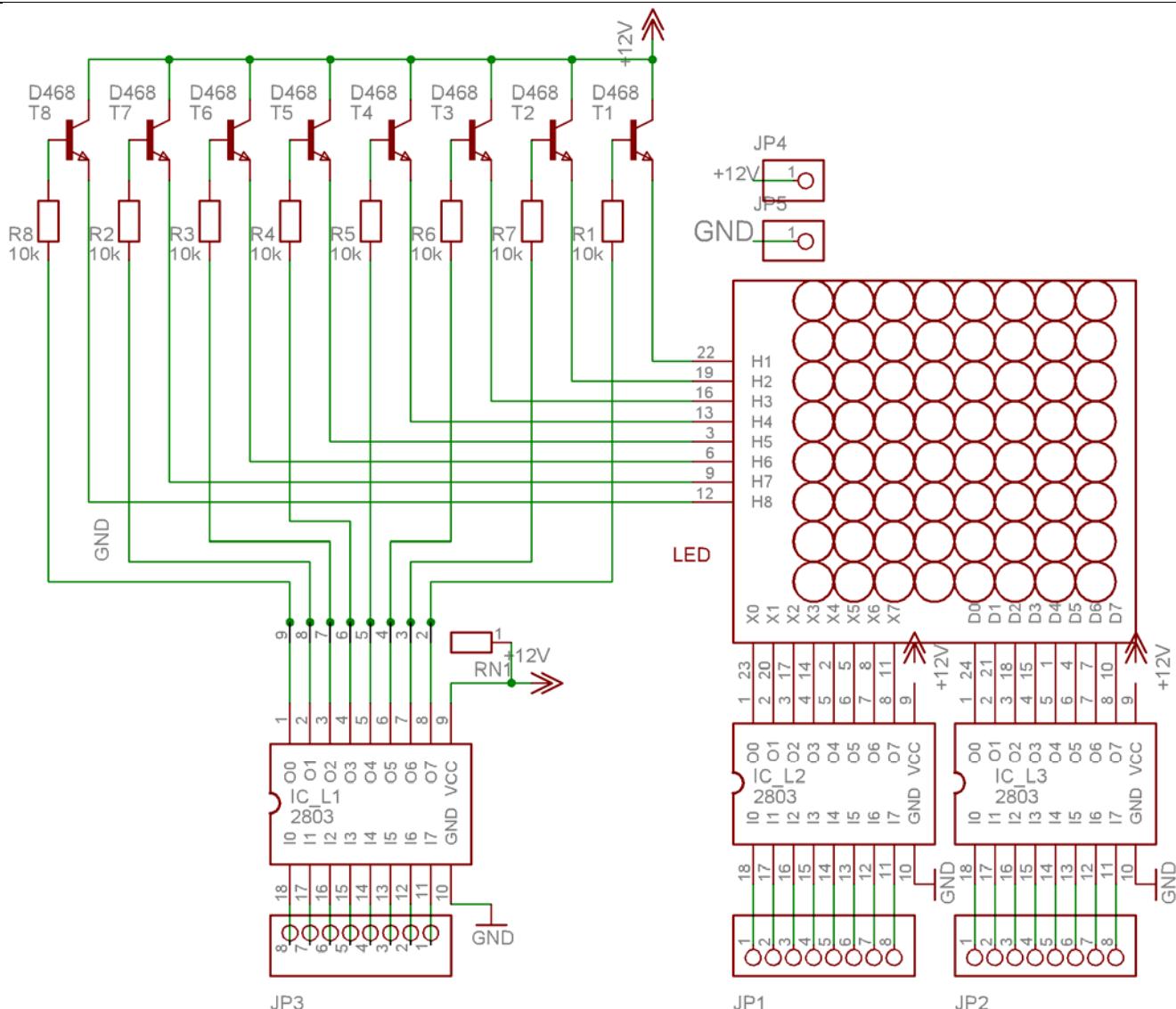
$$I_{LED\_TB} = \frac{160mA}{8} = 20mA$$

Suy ra điện áp cung cấp cho led:

$$V_{LED\_LED} = 160mA \times 100\Omega = 16V$$

Thường thì chọn dòng thấp hơn nên áp cung cấp sẽ thấp – khoảng từ  $9V$  đến  $12V$ .

Sơ đồ mạch điều khiển led ma trận như hình 7-39:



Hình 7-39. Sơ đồ mạch điều khiển.

**Ưu điểm:**

- Khi mở rộng thêm led thì chỉ cần kết nối song song 8 hàng và cột dùng thanh ghi dịch rất đơn giản do số lượng đường điều khiển ít.
- Mỗi một thời điểm chỉ có 1 cột sáng nên dòng tiêu thụ thấp.
- Mạch điện đơn giản.

**Khuyết điểm:**

- Bị giới hạn số cột vì khi mở rộng càng nhiều cột thì thời gian tắt của led tăng thêm  $\Rightarrow$  tăng dòng quá lớn nguy hiểm cho led và led sáng không rõ sinh ra hiện tượng chập chờn. Giả sử có 100 cột thì cho dù quét nhanh hay quét chậm thì thời gian sáng của mỗi cột vẫn không đổi bằng 1/100, thời gian tắt bằng 99/100.

**Cách 2: dữ liệu gửi ra cột, mã quét gửi ra hàng**

Mỗi ký tự có 5 byte mã giống như cách 1, nhưng trình tự thực hiện như sau:

MÃ ĐIỀU KHIỂN HÀNG									MÃ QUÉT ĐIỀU KHIỂN CỘT					
H7	H6	H5	H4	H3	H2	H1	H0	Số hex	C0	C1	C2	C3	C4	
0	0	0	0	0	0	0	1		1	1	1	1	1	Hàng thứ 0 sáng
0	0	0	0	0	0	1	0		1	1	0	1	1	Hàng thứ 1 sáng
0	0	0	0	0	1	0	0		1	0	1	0	1	Hàng thứ 2 sáng
0	0	0	0	1	0	0	0		0	1	1	1	0	Hàng thứ 3 sáng
0	0	0	1	0	0	0	0		0	1	1	1	0	Hàng thứ 4 sáng
0	0	1	0	0	0	0	0		0	0	0	0	0	Hàng thứ 5 sáng
0	1	0	0	0	0	0	0		0	1	1	1	0	Hàng thứ 6 sáng
1	0	0	0	0	0	0	0		0	1	1	1	0	Hàng thứ 7 sáng
								HEX	07	DB	DD	DB	07	

Bảng 7-9. Quét theo cách 2.

- Cho 5 bit thứ 0 của 5 byte mã chữ A được dịch ra các cột, cho hàng H0 dẫn – khi đó hàng H0 sáng tuỳ thuộc vào dữ liệu.
- Cho 5 bit thứ 1 của 5 byte mã chữ A được dịch ra các cột, cho hàng H1 dẫn – khi đó hàng H1 sáng tuỳ thuộc vào dữ liệu.
- Cho 5 bit thứ 2 của 5 byte mã chữ A được dịch ra các cột, cho hàng H2 dẫn – khi đó hàng H2 sáng tuỳ thuộc vào dữ liệu.
- ...
- Cho 5 bit thứ 7 của 5 byte mã chữ A được dịch ra các cột, cho hàng H7 dẫn – khi đó hàng H7 sáng tuỳ thuộc vào dữ liệu.

Với cách quét theo hàng thì thời gian sáng của led là 1/8 thời gian tắt là 7/8.

Khi tăng thêm led  $\Rightarrow$  số cột tăng lên thì thời gian sáng và tắt của mỗi led vẫn không đổi. Nhưng chu kỳ quét sẽ tăng vì phải tốn thêm thời gian gởi dữ liệu ra các cột, càng nhiều cột thì càng tốn thời gian gởi dữ liệu.

Để đáp ứng được tốc độ quét khi bảng quang báo có nhiều led thì phải sử dụng các IC có tốc độ làm việc cao khoảng vài chục MHz như CPLD để phụ trách công việc này.

### 3. CHƯƠNG TRÌNH ĐIỀU KHIỂN LED MA TRẬN

Các chương trình ví dụ sử dụng sơ đồ nguyên lý hình 7-39 và điều khiển theo cách 1.

**Ví dụ 11:** Chương trình điều khiển hiển thị chữ A:

```
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh hien thi ki tu chu A tren ma tran led dung yen
;ma chu A = 007H,0DBH,0DDH,0DBH,007H
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
        hang    equ          p3      ;dieu khien hang
        cotx   equ          p0      ;dieu khien cot mau xanh
        ctd    equ          p2      ;dieu khien cot mau do

        org    0000h
        mov    cotx,#0          ;tat quet neu co ket noi
        mov    ctd,#0           ;tat quet neu co ket noi
```

```

main:      mov   hang,#007h      ;goi du lieu ra hang 1
           mov   cotx,#00000001b    ;goi ma quet cho 1 transistor dan
           lcall delay
           mov   cotx,#00h        ;chong lem

           mov   hang,#0DBh       ;goi du lieu ra hang 2
           mov   cotx,#00000010b    ;cot thu 2 dan
           lcall delay
           mov   cotx,#00h        ;chong lem

           mov   hang,#0ddh       ;goi du lieu re hang 3
           mov   cotx,#00000100b
           lcall delay
           mov   cotx,#00h

           mov   hang,#0DBh       ;goi du lieu ra hang 4
           mov   cotx,#00001000b
           lcall delay
           mov   cotx,#00h

           mov   hang,#007h       ;goi du lieu ra hang 5
           mov   cotx,#00010000b
           lcall delay
           mov   cotx,#00h

           sjmp  main
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con delay nho
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
delay:     mov   r5,#10
de:        mov   r6,#20
           djnz r6,$
           djnz r5,de
           ret
end

```

**Bài tập 13:** Hãy thiết kế mạch điều khiển 4 led ma trận 8×8 dùng thanh ghi dịch 6B595 để quét cột. Viết chương trình hiển thị 1 chuỗi kí tự tùy ý.

## VII. VI ĐIỀU KHIỂN GIAO TIẾP VỚI REAL TIME 12C887:

### 1. GIỚI THIỆU REALTIME:

Real-time là bộ đếm thời gian thực thường được sử dụng trong các ứng dụng điều khiển theo thời gian, ví dụ máy tính luôn có 1 bộ đếm thời gian thực và căn cứ vào thời gian thực để biết các thông số về thời gian của 1 tập tin hay các thông tin được tạo ra, trong hệ thống vận hành của ngân hàng các giao dịch xảy ra đều phải xác định thời gian, trong hệ thống tổng đài luôn có đồng hồ thời gian thực để quản lý thuê bao về thời gian gọi, thời điểm gọi để sau này tính tiền các cuộc gọi, mọi hoạt động của chúng ta đều đồng bộ theo thời gian thực, qua những gì trình bày ta thấy được tầm quan trọng của bộ đếm thời gian thực nên trong phần này trình bày phần giao tiếp với IC thời gian thực là DS12C887 do hãng Dallas sản xuất.

### 2. CÁC THÔNG SỐ CỦA REALTIME DS12C887:

Cấu trúc của Real-time bao gồm:

- Chân IC tương thích với MC146818B và DS1287.
- Dữ liệu không bị mất sau khoảng 10 năm không cần nguồn cung cấp cho IC.
- Tích hợp bên trong nguồn pin và nguồn dao động.
- Bộ đếm giây, phút, giờ, thứ trong ngày, ngày, tháng, năm có giá trị đến 2100.
- Các thông số thời gian được định hình ở 2 dạng BCD hoặc nhị phân.
- Có 2 mode hoạt động: mode 12 giờ và mode 24 giờ.
- Cho phép lựa chọn kết nối theo kiểu Motorola và Intel.
- Đa hợp bus địa chỉ và bus dữ liệu để tiết kiệm chân IC.
- Có 128 byte ô nhớ RAM: trong đó 14 byte đầu dùng đồng hồ và thanh ghi điều khiển, 114 byte còn lại dùng làm bộ nhớ RAM đa dụng. Các dữ liệu lưu vào các ô nhớ này cũng không bị mất khi mất nguồn cung cấp.
- Có chân tín hiệu ngõ ra tạo xung vuông có thể lập trình.
- Có tín hiệu báo ngắn.
- Có 3 nguồn báo ngắn có thể che được và có thể kiểm tra được.

### 3. SƠ ĐỒ CHÂN VÀ CHỨC NĂNG CÁC CHÂN REALTIME DS12C887:

Hình 7-40 là sơ đồ chân của IC real-time DS12C887:

MOT	1	24	V <sub>CC</sub>
NC	2	23	SQW
NC	3	22	NC
AD0	4	21	NC
AD1	5	20	NC
AD2	6	19	IRQ
AD3	7	18	RESET
AD4	8	17	DS
AD5	9	16	NC
AD6	10	15	R/W
AD7	11	14	AS
GND	12	13	CS

**Hình 7-40. Sơ đồ chân của real-time DS12C887.**

Tên các chân của IC như sau:

AD0–AD7	– Multiplexed Address/Data Bus
N.C.	– No Connection
MOT	– Bus Type Selection
<u>CS</u>	– Chip Select
AS	– Address Strobe
<u>R/W</u>	– Read/Write Input
DS	– Data Strobe
<u>RESET</u>	– Reset Input
<u>IRQ</u>	– Interrupt Request Output
SQW	– Square-Wave Output
V <sub>CC</sub>	– +5V Supply
GND	– Ground

### Chức năng các chân:

**AD0–AD7 (Multiplexed Bidirectional Address/Data Bus):** Bus đa hợp tiết kiệm chân bởi vì thông tin địa chỉ và thông tin dữ liệu được dùng chung đường tín hiệu. Địa chỉ được xuất ra trong khoảng thời gian thứ nhất của chu kỳ đọc hoặc ghi và sau đó dữ liệu được xuất ra trong khoảng thời gian còn lại của chu kỳ đọc/ghi. Đa hợp địa chỉ/dữ liệu không làm chậm thời gian truy cập.

**GND, V<sub>CC</sub>:** Nguồn cung cấp cho Real-time. V<sub>CC</sub> là điện áp ngõ vào +5 volt .

Khi điện áp đúng bằng 5V thì real-time cho phép truy cập đầy đủ, có thể đọc và ghi dữ liệu.

Khi V<sub>CC</sub> thấp hơn 4.25 volts thì quá trình đọc và ghi sẽ bị cấm. Tuy nhiên, hoạt động đếm thời gian bên trong vẫn được tiếp tục không bị ảnh hưởng bởi điện áp bị sụt giảm bên ngoài.

Khi V<sub>CC</sub> giảm xuống thấp hơn 3V thì bộ nhớ RAM và hoạt động đếm thời gian được chuyển sang sử dụng nguồn năng lượng bên trong.

Chức năng hoạt động đếm thời gian với độ chính xác vào khoảng ±1 phút/tháng ở nhiệt độ 25<sup>0</sup> C bất chấp điện áp ở ngõ vào chân V<sub>CC</sub>.

**MOT (Mode Select):** Chân MOT là chân có chức năng lựa chọn giữa hai loại bus.

- Khi được nối lên V<sub>CC</sub>, bus định thời Motorola được lựa chọn.
- Khi được nối xuống GND hoặc bỏ trống thì bus định thời Intel được lựa chọn.

Chân có điện trở kéo xuống bên trong có giá trị vào khoảng 20K.

**SQW (Square Wave Output):** Chân SQW có thể xuất ra 1 tín hiệu trong 13 loại tín hiệu được cung cấp bởi các bộ chia 15 cấp bên trong của Real Time Clock.

Tần số của chân SQW có thể thay đổi bằng cách lập trình cho thanh ghi A như đã trình bày ở bảng 7-10.

Tín hiệu SQW có thể mở hoặc tắt sử dụng bit SQWE trong Register B. Tín hiệu SQW không xuất hiện khi V<sub>CC</sub> thấp hơn 4.25 volts.

SELECT BITS REGISTER A				$t_{PI}$ PERIODIC INTERRUPT RATE	SQW OUTPUT FREQUENCY
RS3	RS2	RS1	RS0		
0	0	0	0	None	None
0	0	0	1	3.90625ms	256Hz
0	0	1	0	7.8125ms	128Hz
0	0	1	1	122.070μs	8.192kHz
0	1	0	0	244.141μs	4.096kHz
0	1	0	1	488.281μs	2.048kHz
0	1	1	0	976.5625μs	1.024kHz
0	1	1	1	1.953125ms	512Hz
1	0	0	0	3.90625ms	256Hz
1	0	0	1	7.8125ms	128Hz
1	0	1	0	15.625ms	64Hz
1	0	1	1	31.25ms	32Hz
1	1	0	0	62.5ms	16Hz
1	1	0	1	125ms	8Hz
1	1	1	0	250ms	4Hz
1	1	1	1	500ms	2Hz

**Bảng 7-10. Các thông số tín hiệu ngõ ra SQW.**

**AS (Address Strobe Input):** ngõ vào nhận xung chốt địa chỉ. Sườn xuống của AS/ALE làm cho địa chỉ bị chốt lại bên trong của DS12C887. Sườn lên tiếp theo khi xuất hiện trên bus AS sẽ xoá địa chỉ bất chấp chân CS có được chọn hay không.

**DS (Data Strobe or Read Input):** Chân DS/RD có 2 kiểu sử dụng tùy thuộc vào mức của chân MOT.

Khi chân MOT được kết nối lên Vcc, bus định thời Motorola được lựa chọn. Trong kiểu này DS là xung dương trong khoảng thời gian còn lại của chu kỳ bus và được gọi là Data Strobe. Trong các chu kỳ đọc, tín hiệu DS xác định thời gian DS12C887 xuất dữ liệu. Trong các chu kỳ ghi, xung DS sẽ điều khiển DS12C887 chốt dữ liệu để ghi vào bên trong.

Khi chân MOT được nối xuống GND, bus định thời Intel được lựa chọn. Trong kiểu này, chân DS được gọi là Read( $\overline{RD}$ ).  $\overline{RD}$  xác định chu kỳ khi DS12C887 điều khiển bus đọc dữ liệu. Tín hiệu  $\overline{RD}$  cũng có thể xem là tín hiệu Output Enable ( $\overline{OE}$ ) thường thấy trong các bộ nhớ.

**R/ $\overline{W}$  (Read/Write Input):** Chân R/ $\overline{W}$  cũng có 2 cách hoạt động.

Khi chân MOT được kết nối lên Vcc cho chế độ định thời Motorola, thì mức điện áp của chân R/ $\overline{W}$  dùng để xác định chu kỳ hiện tại là chu kỳ đọc hoặc ghi.

Chu kỳ đọc đòi hỏi chân R/ $\overline{W}$  phải ở mức cao khi chân DS ở mức cao.

Chu kỳ ghi đòi hỏi chân R/ $\overline{W}$  phải ở mức thấp trong suốt quá trình chốt tín hiệu của DS.

Khi chân MOT được nối GND cho chế độ định thời Intel, tín hiệu R/ $\overline{W}$  là tín hiệu hoạt động tích cực mức thấp còn được gọi là  $\overline{WR}$ . Trong chế độ này, chân R/ $\overline{W}$  được định nghĩa như tín hiệu Write Enable ( $\overline{WR}$ ) nhưng trong các bộ nhớ RAM.

**CS (Chip Select Input):** Tín hiệu chọn lựa tích cực mức thấp để cho phép truy xuất DS12C887. CS phải ở trạng thái tích cực trong khoảng thời gian DS và AS của chế độ định thời Motorola và trong suốt  $\overline{RD}$  và  $\overline{WR}$  của chế độ định thời Intel. Các chu kỳ Bus xảy ra mà CS không ở trạng thái tích cực thì không được phép truy xuất.

Khi điện áp nguồn cung cấp Vcc thấp hơn 4.25 volts thì DS12C887 sẽ không cho phép truy xuất dữ liệu bằng cách không cho phép điều khiển chân CS nhằm bảo vệ dữ liệu của đồng hồ thời gian thực bên trong cũng như dữ liệu RAM trong suốt quá trình nguồn không đúng hay mất nguồn.

**IRQ (Interrupt Request Output):** Chân  $\overline{IRQ}$  là ngõ ra tích cực mức thấp của DS12C887 dùng để báo ngắt tới bộ xử lý. Ngõ ra  $\overline{IRQ}$  ở mức thấp khi bit trạng thái sinh ra ngắt xuất hiện cùng với bit cho phép ngắt tương ứng ở mức 1. Để xoá chân  $\overline{IRQ}$  thì vi xử lý phải tiến hành đọc thanh ghi C hoặc tác động đến  $\overline{RESET}$ .

Khi không sinh ra ngắt thì  $\overline{IRQ}$  ở trạng thái tổng trở cao – xem như hở mạch. Nhiều thiết bị ngắt có thể nối tới một  $\overline{IRQ}$  bus. Tín hiệu  $\overline{IRQ}$  với ngõ ra cực thu để hở nên phải dùng 1 điện trở kéo lên bên ngoài.

**RESET (Reset Input):** Chân  $\overline{RESET}$  không ảnh hưởng đến đồng hồ, lịch, hoặc RAM. Ở chế độ cấp nguồn, chân  $\overline{RESET}$  có thể bị kéo xuống trong thời gian cho phép để ổn định nguồn cung cấp. Thời gian mà chân  $\overline{RESET}$  bị kéo xuống mức thấp phụ thuộc vào ứng dụng. Tuy nhiên nếu chân  $\overline{RESET}$  được sử dụng ở chế độ cấp nguồn, thời gian  $\overline{RESET}$  ở mức thấp có thể vượt quá 200ms để chắc chắn bộ định thời bên trong DS12C887 ở chế độ power-up đã hết. Khi  $\overline{RESET}$  ở mức thấp và nguồn cung cấp  $V_{CC}$  ở trên 4.25 volts, những điều sau diễn ra:

- Bit cho phép ngắt định kỳ ((Periodic Interrupt Enable (PEI)) được đặt ở mức 0.
- Bit cho phép ngắt chuông (Alarm Interrupt Enable (AIE)) được đặt ở mức 0.
- Bit cờ cho phép ngắt kết thúc cập nhật ((Update Ended Interrupt Flag (UF)) được xoá về 0 zero.
- Bit cờ trạng thái yêu cầu ngắt (Interrupt Request Status Flag (IRQF)) được đặt ở mức 0.
- Bit cờ cho phép ngắt định kỳ (Periodic Interrupt Flag (PF)) được đặt ở mức 0.
- Thiết bị không sử dụng được cho tới khi chân  $\overline{RESET}$  trở lại mức logic 1.
- Bit cờ cho phép ngắt chuông (Alarm Interrupt Flag (AF)) được đặt ở mức 0.
- Chân  $\overline{IRQ}$  ở trong trạng thái tổng trở cao.
- Bit cho phép xuất sóng vuông (Square Wave Output Enable (SQWE)) được đặt ở mức 0.
- Bit cho phép ngắt kết thúc cập nhật (Update Ended Interrupt Enable (UIE)) bị xoá về mức 0.

Trong các ứng dụng thông thường chân  $\overline{RESET}$  có thể được nối lên  $V_{CC}$ . Kết nối như vậy sẽ cho phép DS12C887 hoạt động và khi mất nguồn sẽ không làm ảnh hưởng đến bất kỳ thanh ghi điều khiển nào.

#### 4. HOẠT ĐỘNG CỦA REAL-TIME CLOCK KHI CẤP NGUỒN HOẶC KHI MẤT ĐIỆN

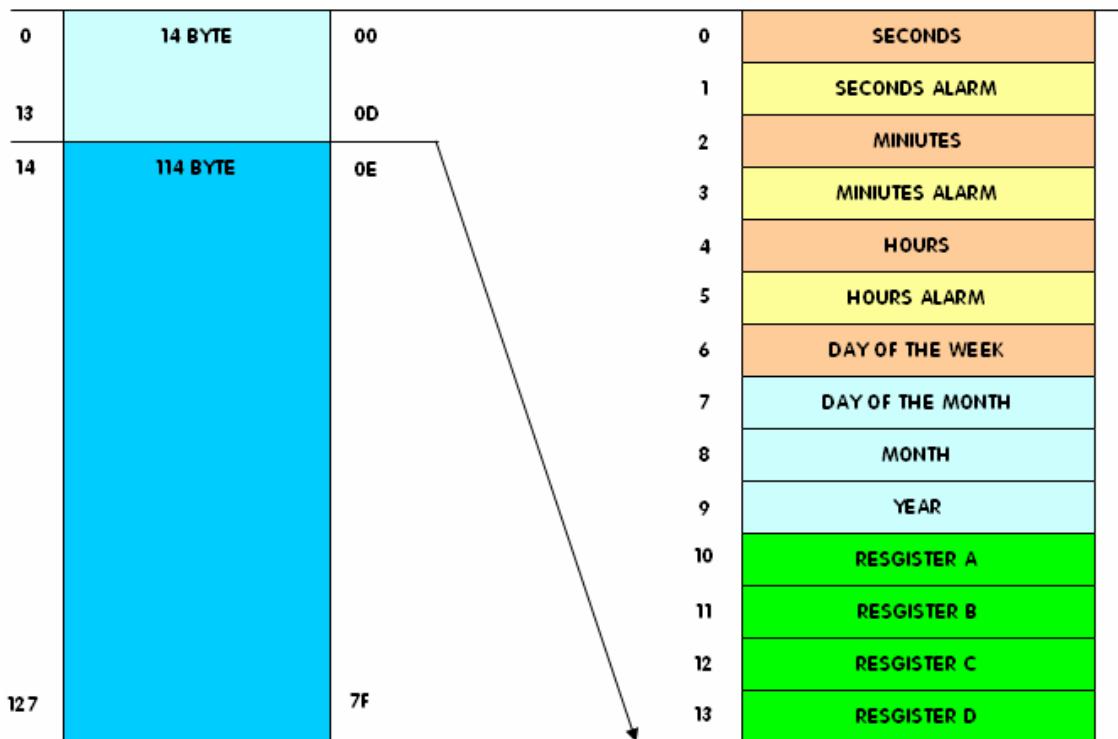
Chức năng của đồng hồ thời gian thực sẽ tiếp tục hoạt động và tất cả RAM, thời gian, lịch và thời gian báo giờ và những vùng nhớ sẽ không mất dữ liệu bất chấp điện áp ngõ vào  $V_{CC}$ .

Khi điện áp  $V_{CC}$  được cung cấp cho DS12C887 và đạt tới điện áp lớn hơn 4.25 volts, thiết bị có thể sử dụng được sau 200 ms, dao động được cung cấp, nó cho phép bộ dao động hoạt động và quá trình dao động không ảnh hưởng bởi chân  $\overline{RESET}$ . Tiếp theo hệ thống đã đi vào ổn định sau khi nguồn được cung cấp.

Khi  $V_{CC}$  rớt xuống dưới 4.25 volts, ngõ vào lựa chọn  $\overline{CS}$  bị bắt buộc chuyển sang trạng thái không hoạt động bất chấp giá trị tại ngõ vào chân  $\overline{CS}$  - khi đó DS12C887 được hoạt động ở chế độ chống ghi. Khi DS12C887 đang ở trạng thái chống ghi, mọi ngõ vào đều bị bỏ qua còn các ngõ ra đều ở trạng thái tổng trở cao.

Khi V<sub>CC</sub> rớt xuống điện áp khoảng 3 volts, điện áp V<sub>CC</sub> cung cấp bên ngoài được cắt đi và nguồn pin lithium ở bên trong DS12C887 sẽ cung cấp nguồn cho Real Time Clock và bộ nhớ RAM.

Bản đồ địa chỉ của DS12C887 được trình bày ở hình 7-41.



Hình 7-41. Tổ chức bộ nhớ bên trong của Real-time DS12C887.

Bản đồ địa chỉ bao gồm 114 bytes RAM thông dụng, 14 bytes RAM bao gồm đồng hồ thời gian thực, lịch, dữ liệu báo giờ và 4 thanh ghi được sử dụng cho việc điều khiển và thông báo tình trạng. Tất cả 128 bytes có thể được ghi hoặc đọc trực tiếp trừ những trường hợp sau:

- Thanh ghi C and D là hai thanh ghi chỉ đọc.
- Bit thứ 7 của thanh ghi A là bit chỉ đọc.
- Bit cao của byte thứ 2 là bit chỉ đọc.

Các thông số về thời gian và lịch có được bằng cách đọc các bytes tương ứng. Thời gian, lịch và báo giờ được cài đặt hoặc khởi tạo bằng cách ghi giá trị bytes RAM thích hợp.

Nội dung của 10 byte chứa thời gian, lịch và báo giờ đều có thể định dạng 1 trong 2 dạng nhị phân (Binary) hoặc BCD (Binary Coded Decimal).

Trước khi ghi các dữ liệu vào các ô nhớ thời gian, lịch, và các thanh ghi báo giờ bên trong thì bit SET ở thanh ghi B phải được thiết lập ở mức logic 1 để ngăn chặn việc cập nhật có thể xảy ra trong quá trình truy xuất.

Ngoài ra để ghi lên 10 thanh ghi chỉ thời gian, lịch, và thanh ghi báo giờ ở một định dạng đã lựa chọn (BCD hay nhị phân) thì bit chọn kiểu dữ liệu (Data mode – DM) của thanh ghi B phải được đặt ở mức logic thích hợp.

Tất cả 10 bytes thời gian, lịch và báo giờ phải sử dụng cùng kiểu dữ liệu. Phải xoá bit SET trong thanh ghi B sau khi thực hiện xong việc cập nhật thời gian cho đồng hồ, lịch và hẹn giờ.

Mỗi lần khởi động, đồng hồ thời gian thực cập nhật tất cả các thay đổi ở một kiểu đã lựa chọn. Dữ liệu sẽ không thể thay đổi nếu không khởi động lại 10 bytes dữ liệu.

Bảng 7-11 trình bày định dạng nhị phân và BCD của cả thời gian, lịch, và báo giờ.

Địa chỉ	Chức năng	Giá trị thập phân	Dãy lựa chọn	
			Nhị phân	BCD
0	Giây	0-59	00-3B	00-59
1	Giây hẹn	0-59	00-3B	00-59
2	Phút	0-59	00-3B	00-59
3	Phút hẹn	0-59	00-3B	00-59
4	Giờ: mode 12 giờ	1-12	01-0C AM, 81-8C PM	01-12 AM, 81-92 PM
	Giờ: mode 24 giờ	0-23	00-17	00-23
5	Giờ hẹn: 12 giờ	1-12	01-0C AM, 81-8C PM	01-12 AM, 81-92 PM
	Giờ hẹn: 24 giờ	0-23	00-17	00-23
6	Ngày của tuần, chủ nhật = 1	1-7	01-07	01-07
7	Ngày của tháng	1-31	01-1F	01-31
8	Tháng	1-12	01-0C	01-12
9	Năm	0-99	00-63	00-99

Bảng 7-11. Các định dạng của các thông số thời gian.

Bit lựa chọn kiểu 24/12 không thể thay đổi nếu không khởi động lại thanh ghi giờ. Khi định dạng 12 giờ được lựa chọn, bit cao của byte giờ xác định PM khi nó được đặt ở mức logic 1. Mỗi giây một lần, 11 byte được cập nhật và kiểm tra tình trạng báo giờ.

Nếu lệnh đọc dữ liệu thời gian và lịch diễn ra trong quá trình cập nhật, một vấn đề phát sinh là giờ, phút, giây, ... có thể không chính xác. Xác xuất đọc không chính xác dữ liệu thời gian và lịch là rất thấp. Có vài phương pháp tránh sai số có thể xảy ra khi đọc thời gian và lịch được đề cập sau.

3 byte báo giờ có thể sử dụng bằng 2 cách:

**Cách thứ nhất:** khi thời gian báo giờ được ghi vào các thanh ghi giờ, phút, giây, tác động báo giờ được bắt đầu tại thời gian chính xác trong ngày khi bit cho phép báo chuông được đặt ở mức cao.

**Cách thứ hai:** sử dụng để đặt trạng thái bất chấp vào một hoặc nhiều byte báo chuông. Mã bất chấp là bất kỳ mã số hex nào nằm trong giá trị từ C0 đến FF. Hai bit MSB của những byte trên đặt vào trạng thái bất chấp khi ở mức logic 1. Báo giờ sẽ được sinh ra mỗi giờ khi bit bất chấp được đặt vào byte giờ. Tương tự, báo giờ sẽ sinh ra mỗi phút nếu mã bất chấp có ở các byte giờ và các byte phút. Nếu mã bất chấp có ở trong cả 3 byte báo giờ thì nó sẽ tạo ra tín hiệu ngắn mỗi giây.

## 5. HOẠT ĐỘNG CỦA CÁC THANH GHI ĐIỀU KHIỂN

DS12C887 có 4 thanh ghi điều khiển được sử dụng mọi lúc kể cả trong quá trình cập nhật.

Thanh ghi A

MSB

LSB

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
UIP	DV2	DV1	DV0	RS3	RS2	RS1	RS0

**UIP - Update In Progress (UIP)** là bit cờ trạng thái có thể theo dõi được.

Khi bit UIP ở mức 1 thì cho phép quá trình cập nhật xảy ra ngay sau đó.

Khi bit UIP ở mức 0, quá trình cập nhật sẽ không xảy ra sau khoảng thời gian ít nhất là 244 µs. Những thông tin về thời gian, lịch, và báo giờ ở trong RAM có hiệu lực cho việc truy cập khi bit UIP ở mức 0.

Bit UIP là bit chỉ đọc và không bị ảnh hưởng của chân RESET. Khi ghi bit SET ở thanh ghi B lên 1 sẽ ngăn chặn mọi quá trình cập nhật và xoá của bit trạng thái UIP.

**DV2, DV1, DV0** – 3 bit này được sử dụng để mở hoặc tắt bộ dao động và cài đặt lại quá trình đếm xuống. Khi [DV2DV1 DV0] = [010] có chức năng duy nhất là bật bộ dao động lên và cho phép RTC đếm thời gian. Khi [DV2DV1 DV0] = [11x] sẽ cho phép dao động nhưng giữ quá trình đếm xuống ở mức reset. Quá trình cập nhật tiếp theo sẽ diễn ra sau 500ms khi kiểu 010 được ghi vào DV0, DV1 và DV2.

**RS3, RS2, RS1, RS0** - 4 bit lựa chọn tốc độ dùng để chọn một trong 13 loại của bộ chia 15 trạng thái hoặc không cho phép xuất tín hiệu chia ra ngoài. Loại được lựa chọn có thể dùng để phát ra sóng vuông (chân SQW) và/hoặc ngắn theo chu kỳ. Người sử dụng có thể sử dụng 1 trong những cách sau :

- Cho phép ngắn với bit PIE
- Cho phép xuất ngắn ra chân SQW với bit SQWE
- Cho phép cả hai hoạt động cùng một lúc và cùng một loại.
- Không kích hoạt cả 2

Bảng 7-10 liệt kê chu kỳ ngắn và tần số sóng vuông mà có thể chọn lựa với bit RS. Cả 4 bit đọc/ghi đều không bị ảnh hưởng bởi chân RESET.

#### Thanh ghi B

MSB								LSB							
BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	SET	PIE	AIE	UIE	SQWE	DM	24/12	DSE

**SET** – Khi bit SET ở mức 0, thông thường quá trình cập nhật bằng cách tăng giá trị đếm 1 lần 1 giây. Khi bit SET được ghi vào mức 1, mọi quá trình cập nhật đều bị cấm. Các chu kỳ đọc có thể thực thi ở cùng một kiểu. Bit SET là bit đọc/ghi và không chịu ảnh hưởng của reset hoặc các chức năng bên trong của DS12C887.

**PIE – Periodic Interrupt Enable:** Bit cho phép ngắn theo chu kỳ là bit đọc/ghi, bit này cho phép bit cờ ngắn theo chu kỳ (PF) trong thanh ghi C để điều khiển chân IRQ xuống mức thấp.

Khi bit PIE được đặt lên mức 1, chu kỳ ngắn được tạo ra bằng cách điều khiển chân IRQ xuống mức thấp tùy thuộc vào tỉ lệ phân bố của bit RS3 ÷ RS0 ở thanh ghi A.

Khi bit PIE = 0 sẽ ngăn chặn tín hiệu ngắn ra IRQ khỏi việc điều khiển bằng ngắn theo chu kỳ nhưng bit cờ chu kỳ PF vẫn thiết lập tốc độ cho chu kỳ.

Bit PIE không chịu ảnh hưởng các hoạt động bên trong của DS12C887 nhưng bị xoá về 0 khi real-time bị reset.

**AIE – Alarm Interrupt Enable:** Bit cho phép ngắt báo giờ là bit cho phép đọc/ghi, khi bit AIE bằng 1 sẽ cho phép bit cờ báo giờ (AF) ở thanh ghi C tạo tín hiệu ngắt  $\overline{IRQ}$ .

Tín hiệu ngắt báo giờ xảy ra ở tất cả các giây khi cả 3 bytes báo giờ chứa mã báo giờ “don’t care” được thể hiện ở byte nhị phân như sau 11XXXXXX.

Các chức năng bên trong của DS12C887 không bị ảnh hưởng bởi bit AIE.

**UIE – Update Ended Interrupt Enable:** Bit cho phép kết thúc quá trình ngắt cập nhật là bit đọc/ghi, bit này cho phép cờ kết thúc cập nhật UF ở thanh ghi C kích ngắt  $\overline{IRQ}$ . Khi real-time bị reset hoặc chân SET ở mức 1 sẽ xóa bit UIE.

**SQWE – Square Wave Enable:** Khi bit SQWE được đặt lên mức 1, một tín hiệu sóng vuông có tần số được lựa chọn bởi các bit RS3 đến RS0 sẽ xuất ra chân SQW. Khi bit SQWE được đặt ở mức thấp, chân SQW sẽ được giữ ở mức thấp. SQWE là bit đọc/ghi và được xóa khi real-time bị RESET.

**DM – Data Mode:** Bit kiểu dữ liệu DM quy định khi nào thì thông tin lịch và thời gian định dạng nhị phân hoặc BCD. Bit DM được đặt bởi chương trình để có định dạng thích hợp và có thể đọc khi được yêu cầu. Bit này không bị thay đổi bởi các hoạt động bên trong của real-time hoặc khi bị reset. Mức 1 của DM sẽ định dạng dữ liệu nhị phân còn mức 0 sẽ định dạng dữ liệu BCD.

**24/12** – Bit điều khiển 24/12 xác định kiểu cho các byte giờ. Khi ở mức 1 thì nó chỉ thị chế độ hiển thị 24 giờ, còn ở mức 0 thì chỉ thị chế độ hiển thị 12 giờ. Bit này là bit đọc/ghi và không bị ảnh hưởng bởi hoạt động bên trong của real-time cũng như khi real-time bị reset.

**DSE – Daylight Savings Enable:** Bit cho phép nhở công khai DSE là bit đọc/ghi.

Bit này cho phép 2 cập nhật đặc biệt khi DSE được đặt lên 1.

Vào Chủ Nhật đầu tiên của tháng 4, khi thời gian tăng đến giá trị 1:59:59 AM thì sẽ đổi thành 3:00:00 AM.

Vào Chủ Nhật cuối cùng của tháng 10, khi thời gian lần đầu tiên đạt đến 1:59:59 AM thì nó sẽ đổi thành 1:00:00 AM.

Chức năng đặc biệt này sẽ không được thực thi nếu bit DSE ở mức 0. Bit này không bị ảnh hưởng bởi các hoạt động bên trong của real-time cũng như khi bị reset.

### Thanh ghi C

Thanh ghi C							
MSB				LSB			
BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
IRQF	PF	AF	UF	0	0	0	0

**IRQF – Interrupt Request Flag:** Bit cờ yêu cầu ngắt được đặt lên 1 khi thoả một trong những điều dưới đây:

- PF = PIE = 1
- AF = AIE = 1
- UF = UIE = 1

Điều đó có nghĩa là IRQF = (PF . PIE) + (AF . AIE) + (UF . UIE)

Bất cứ lúc nào bit IRQF được đặt lên 1, chân  $\overline{IRQ}$  được kéo về mức thấp. Tất cả các cờ bị xoá sau khi thanh ghi C được đọc bởi chương trình hoặc khi real-time bị reset.

**PF – Periodic Interrupt Flag:** Bit cờ ngắn theo chu kỳ là bit chỉ đọc, nếu bit này được đặt lên mức 1 khi có 1 cạnh xung được phát hiện ở tín hiệu lựa chọn của bộ chia. Các bit RS3 đến RS0 thiết lập tốc độ cho chu kỳ. Cờ PF được đặt lên 1 không phụ thuộc vào trạng thái của bit PIE.

Khi cả PF và PIE đều ở mức 1, tín hiệu  $\overline{IRQ}$  sẽ chuyển sang trạng thái tích cực và sẽ đặt bit IRQF lên mức 1. Bit PF sẽ bị xoá bởi reset real-time hoặc bởi lệnh đọc thanh ghi C.

**AF – Alarm Interrupt Flag:** nếu  $AF = 1$  sẽ xác định thời gian hiện tại trùng với thời gian hẹn giờ. Nếu bit AIE cũng ở mức 1, chân  $\overline{IRQ}$  sẽ xuống mức thấp và bit IRQF sẽ bằng 1. Bit AF sẽ bị xoá bởi reset real-time hoặc bởi lệnh đọc thanh ghi C.

**UF – Update Ended Interrupt Flag:** Bit cờ ngắn kết thúc cập nhật được set sau mỗi chu kỳ cập nhật. Khi bit UIE được set lên 1, mức 1 ở UF sẽ làm cho bit IRQF lên mức 1 sẽ kích  $\overline{IRQ}$  báo ngắn. Bit UF sẽ bị xoá bởi reset real-time hoặc bởi lệnh đọc thanh ghi C.

**Từ bit 3 đến bit 0-** Đây là những bit không sử dụng của thanh ghi trạng thái C. Những bit này luôn luôn ở mức 0 và không thể ghi.

#### Thanh ghi D

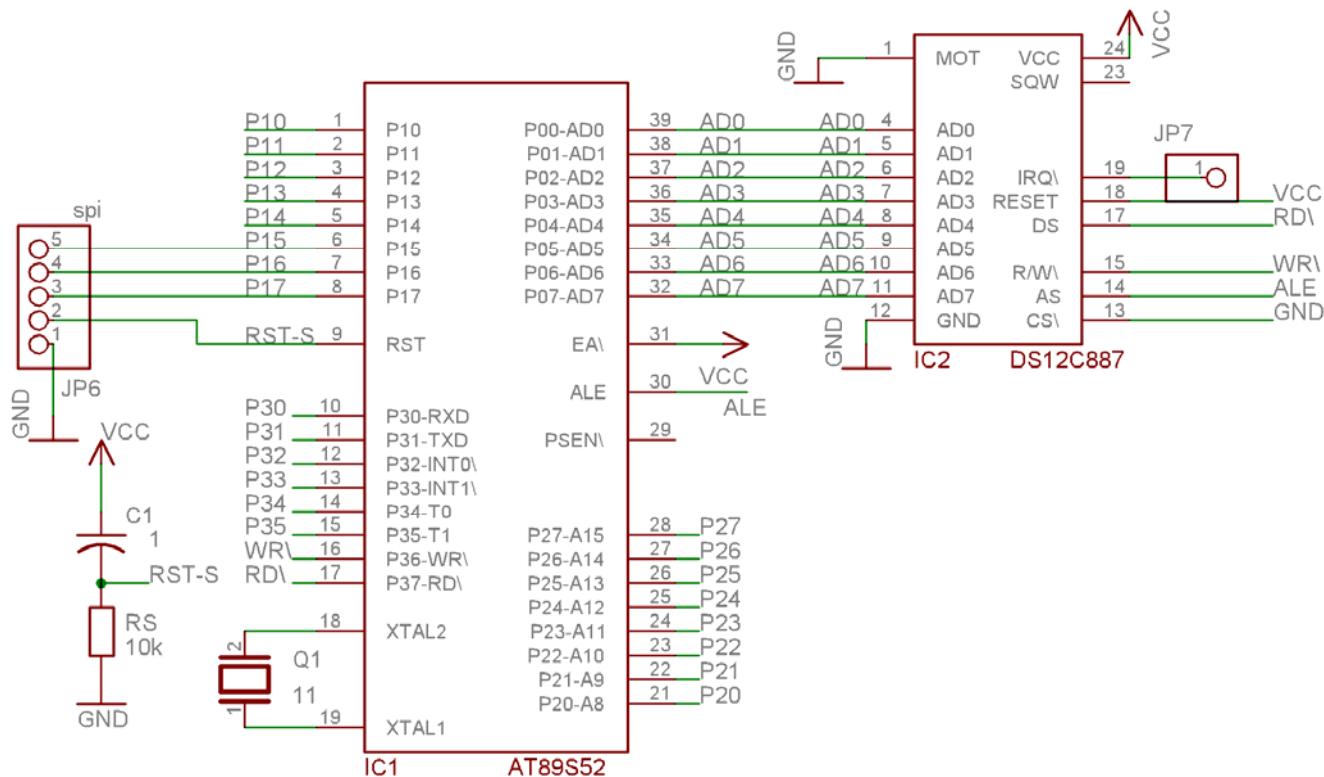
								MSB		LSB
BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0			
VRT	0	0	0	0	0	0	0			

**VRT – Valid RAM and Time:** Bit thời gian và RAM có hiệu lực cho biết tình trạng của pin được kết nối chân VBAT. Bit này không cho phép ghi được và khi đọc luôn có giá trị bằng 1. Nếu khi đọc bit này ở mức 0 thì sẽ cho biết nguồn năng lượng lithium bên trong đã cạn và cả hai thông số thời gian thực và dữ liệu RAM đều có thể sai. Bit này không chịu ảnh hưởng bởi chân RESET.

**BIT 6 ĐẾN BIT 0 –** Những bit được đề cập ở trên của thanh ghi D không được sử dụng. Chúng không ghi được và khi đọc thì luôn có giá trị bằng 0.

#### 6. MẠCH GIAO TIẾP VI ĐIỀU KHIỂN VỚI REAL-TIME

Mạch giao tiếp vi điều khiển với real-time DS12C887 như hình 7-42. Trong sơ đồ này vi điều khiển chỉ giao tiếp với 1 real-time với mục đích trình bày cách giao tiếp chứ trong thực tế thì vi điều khiển phải giao tiếp với nhiều thành phần khác thì hệ thống mới hoạt động ví dụ giao tiếp thêm led 7 đoạn hoặc LCD mới hiển thị được các thông số thời gian.



Hình 7-42. Giao tiếp vi điều khiển với Real-time.

Trong mạch điện hình 7-42, vi điều khiển giao tiếp với real-time theo kiểu bộ nhớ dữ liệu, vi điều khiển có 4 lệnh giao tiếp với bộ nhớ dữ liệu mở rộng bên ngoài:

- Lệnh đọc dữ liệu: “movx a,@dptr” (1)
- Lệnh xuất dữ liệu: “movx @dptr,a” (2)
- Lệnh đọc dữ liệu: “movx a,@Ri” (3)
- Lệnh xuất dữ liệu: “movx @Ri,a” (4)

Hai lệnh (1) và (2) thường được sử dụng khi dung lượng bộ nhớ mở rộng lớn hơn 256 byte, port 2 dùng để tải địa chỉ byte cao. Hai lệnh (3) và (4) được sử dụng khi dung lượng bộ nhớ nhỏ hơn hay bằng 256 byte, port 2 không dùng có thể điều khiển các thiết bị khác. Vậy trong trường hợp này nên dùng 2 lệnh (3) và (4).

### 7. CHƯƠNG TRÌNH KHỞI TẠO REAL-TIME:

Mạch giao tiếp vi điều khiển với real-time DS12C887 như hình 7-42. Nếu real-time chưa sử dụng lần nào thì phải chạy chương trình khởi tạo để thiết lập ngày giờ hiện tại cho real-time và chỉ chạy 1 lần. Trong chương trình bên dưới các thông số “giayhientai”, “phuthientai” và “giohientai” là các thông số thời gian hiện tại khi khởi động.

Chương trình khởi tạo cho real-time:

```

org      0000h
mov     r0,#0bh      ;dia chi cua thanh ghi B
mov     a,#0          ;du lieu nap vao thanh ghi B
movx   @r0,a          ;cho phep chinh cai dat gio

mov     a,#giayhientai
mov     r0,#0          ;dia chi cua o nho giay
movx   @r0,a          ;nap giay hien tai

```

```

        mov      a,#phuthientai
        mov      r0,#02      ;dia chi cua o nho phut
        movx    @r0,a       ;nap phut hien tai

        mov      a,#giohientai
        mov      r0,#04      ;dia chi cua o nho gio
        movx    @r0,a       ;nap gio hien tai

        mov      r0,#0ah     ;dia chi cua thanh ghi A
        mov      a,#020h     ;tu dieu khien cua thanh ghi A
        movx    @r0,a       ;cho phep bo dao dong chay
        lcall   delay50
        sjmp   $

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

delay50:  mov      7eh,#0fh
del12:   mov      7dh,#0ffh
del11:   djnz    7dh,del11
          djnz    7eh,del12
          ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

end

```

Chương trình ví dụ là chương trình đồng hồ dùng vi điều khiển, realtime và LCD: trong chương trình này vi điều khiển kết nối với realtime sử dụng port2 và 3 bit điều khiển của port1.

```

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh dieu khien LCD hien thi chuoi thong tin
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;dinh nghia cac bien

        E      BIT    P1.0
        rw     BIT    P1.1
        rs     BIT    P1.2
        byteout equ    p2
        dklcd  equ    70h

        bdn    equ    r6
        giay   equ    r5
        phut   equ    r4
        gio    equ    r3

        org    0000h
        mov    sp,#68h

;

        lcall  khoitao_rt      ;goi chtr con khai tao real time
        lcall  khoitao_lcd     ;goi chtr khai tao LCD
        lcall  xoavnhanh_thi  ;xoai vung nho hien thi

main1:   lcall  doc_gpg       ;doc thong so gio-phut-giay tu real time
        lcall  bcd_maascii    ;chuyen so bcd sang ma ASCII
        lcall  hienthichung   ;goi ra LCD de hien thi
        sjmp  main1          ;tiep tuc

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con doc gio phut giay
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
doc_gpg:  mov    r0,#00
          movx  a,@r0
          mov    giay,a

```

```

        mov    r0,#02
        movx   a,@r0
        mov    phut,a
        mov    r0,#04
        movx   a,@r0
        mov    gio,a
        ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con chuyen so bcd sang ma ASCII
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
bcd_maascii:  mov    a,giay
               anl    a,#0fh
               add    a,#30h
               mov    4fh,a

               mov    a,giay
               anl    a,#0f0h
               swap   a
               add    a,#30h
               mov    4eh,a

               mov    a,phut
               anl    a,#0fh
               add    a,#30h
               mov    4ch,a

               mov    a,phut
               anl    a,#0f0h
               swap   a
               add    a,#30h
               mov    4bh,a

               mov    a,gio
               anl    a,#0fh
               add    a,#30h
               mov    49h,a

               mov    a,gio
               anl    a,#0f0h
               swap   a
               add    a,#30h
               mov    48h,a
               ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;Chuong trinh con hien thi noi dung tren LCD cua2 vung nho
;40H->4Fh hang 1
;50H-> 5Fh hang 2;
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
hienthichung:  mov    A,#080h           ;set DDRAM
                lcall   ktao

                mov    r1,#16
                mov    r0,#40h           ;bat dau tu 0
                lcall   Write
                djnz   r1,fline

                mov    a,#0c0h           ;set DDRAM
                lcall   ktao
                mov    r1,#16
                lcall   Write

```

```

        djnz    r1,sline
        ret
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con goi data hien thi ra LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
write:      mov     a,@r0
write1:     mov     byteout,a
            lcall   data_byte
            inc    r0
            ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con khoi tao LCD
;chuong trinh dieu khien LCD 16X2 tren kit vi dieu khien LOAI NHO
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
khoitao_lcd:  mov    dklcd,#0
                lcall  khtaolcd           ;khởi tạo lcd
                ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chtr con khoi tao LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
khtaolcd:   setb   e                  ;Enable
            clr    rs                 ;RS low
            clr    rw                 ;RW low

            mov    a,#38h             ;tu dieu khien LCD
            lcall  ktao
            lcall  ddelay41          ;delay 4.1 mSec

            mov    A,#38h             ;function set
            lcall  ktao
            lcall  ddelay100          ;delay

            mov    A,#38h             ;function
            lcall  ktao

            mov    A,#0ch              ;tu dieu khien display on
            lcall  ktao
            mov    A,#01h              ;tu dieu khien Clear display
            lcall  ktao

            mov    A,#06h              ;tu dieu khien entry mode set
            lcall  ktao

            mov    A,#80h              ;thiet lap dia chi LCD (set DD RAM)
            lcall  ktao
            ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con khoi tao LCD
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ktao:       mov    byteout,a
            lcall  command_byte
            ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;Feed command/data to the LCD module
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
command_byte:
            clr    rs                 ;RS low for a command byte
            ljmp  bdelay

```

```

data_byte:    setb   rs           ;RS high for a data byte
bdelay:      clr    rw           ;R/W low for a write mode
             clr    e
             nop
             setb   e           ;Enable pulse
             nop
             nop
             mov    byteout,#0ffh ;configure port1 to input mode
             setb   rw           ;set RW to read
             clr    rs           ;set RS to command
             clr    e            ;generate enable pulse
             nop
             nop
             setb   e
             lcall  ddelay100
             ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con delay 4.1 ms
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ddelay41:    mov    7eh,#90h
de412:       mov    7fh,#200
             djnz  7fh,$
             djnz  7eh,del412
             ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con delay 255 microgiay
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ddelay100:   mov    7fh,#00
             djnz  7fh,$
             ret

delay1giay:  mov    7ah,#20
xx2:         mov    7bh,#0
xx1:         mov    7ch,#0
             djnz  7ch,$
             djnz  7bh,xx1
             djnz  7ah,xx2
             ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con khai tao real time
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

khoitao_rt:  mov    r0,#0bh
             mov    a,#0
             movx  @r0,a        ;cho phep chinh cai dat gio
             mov    r0,#0
             movx  @r0,a        ;giay = 0
             mov    r0,#02
             movx  @r0,a        ;phut = 0
             mov    r0,#04
             movx  @r0,a        ;gio = 0

```

```

        mov    r0,#0ah
        mov    a,#020h
        movx   @r0,a      ;cho phep bo dao dong chay
        lcall  delay1giay
        ret

;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;chuong trinh con xoa vung nho hien thi
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xoavn_hthi:  mov    r0,#40h
              mov    a,' '
xoac:        mov    @r0,a
              inc    r0
              cjne  r0,#60h,xoac
              ret

end

```

**Bài tập 14:** Hãy hiệu chỉnh lại chương trình để có đầy đủ các thông số thứ, ngày, tháng, năm hiển thị trên LCD.

### VIII. TÓM TẮT:

Trong chương này trình bày giao tiếp vi điều khiển với các thiết bị ngoại vi cơ bản thường được dùng trong các hệ thống điều khiển thực tế.

Một hệ thống điều khiển tuỳ thuộc vào mức độ phức tạp mà mạch giao tiếp với nhiều thiết bị khác nhau, ở chương này trình bày từng phần rời rạc, tác giả hy vọng rằng sau khi hiểu được từng phần người học hay người đọc có thể thiết kế các hệ thống giao tiếp theo yêu cầu.

return  
the end

## TÀI LIỆU THAM KHẢO

- [1]. AVTAR SINGH - WALTER A TRIEBEL, "The 8088 Microprocessor – Programming, interfacing, software, hardware, and Applications" , Prentice Hall International Editions.
- [2]. DOUGLAS V. HALL, " Microprocessor and Interfacing Programming, and hardware", McGraw – Hill International Editions.
- [3]. John Uffenbeck, "The 8088/8086 family : Designing, programming and interfacing", Prentice Hall, 1987
- [4]. James L. Antonakos, "The 68000 Microprocessor: hardware and software principles and applications", Prentice Hall fifth edition 2004.
- [5]. Jack L. Davies, "The Innovative 80x86 – Volume I: the 80286 Microprocessor, architecture", Prentice Hall.
- [6]. Jack L. Davies, "Z80 Family CPU user manual", [www.zilog.com](http://www.zilog.com).
- [7]. MetaLink Corporation Chandler – Arizona, "8051 Cross Assembler User's Manual", 1996
- [8]. "MCS51 Microcontroller Family User's Manual", 1994
- [9]. "M68332 User's Manual"
- [10]. Hồ Trung Mỹ, Vi Xử Lý, Nhà xuất bản Đại Học Quốc Gia Tp HCM
- [11]. Ngô Diên Tập, Kỹ Thuật AVR, Nhà xuất bản KH & KT – 2003.
- [12]. Văn Thể Minh, Kỹ Thuật vi xử lý, Nhà xuất bản giáo dục – 1997.