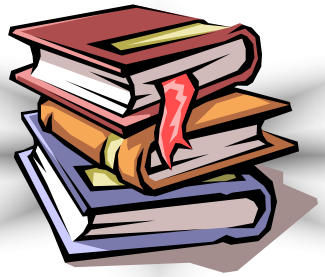


Kỹ Thuật Vi Điều Khiển

CHƯƠNG I : GIỚI THIỆU TỔNG QUÁT VỀ HỆ VI XỬ LÝ

1.1. Kiến trúc của hệ Vi xử lý

Kiến trúc hệ Vi xử lý là một thuật ngữ dùng để chỉ rõ những đặc trưng của hệ vi xử lý trong đó bao gồm có cấu trúc phần cứng và tổ chức phần mềm được cài đặt trong hệ. Một hệ thống vi xử lý hay gọi ngắn hơn là hệ vi xử lý thường bao gồm các thành phần cơ bản như:

- Bộ xử lý trung tâm CPU (Central Processing Unit) là trung tâm đầu não của hệ
- Bộ nhớ tâm bao gồm 2 thành phần là ROM và RAM
- Thiết bị vào/ra dữ liệu
- Các kênh thông tin hay Bus hệ thống…

Tất cả các thiết bị có các chức năng như vậy đều được gọi là một hệ vi xử lý.Trong thực tế có rất nhiều hãng chế tạo các bộ vi xử lý cho các máy vi tính như: IBM, Intel, Cyrix, AMD, Motorola.... nhưng thông dụng nhất vẫn là bộ vi xử lý của Intel. Các bộ vi xử lý của Intel được phát triển qua các thời kỳ như sau:

Năm 1971, Intel đưa ra bộ vi xử lý 4004 với 4 bit dữ liệu, 12 bit địa chỉ; 0,8MHz

Năm 1972, bộ vi xử lý Intel 8080 ra đời với 8bit dữ liệu, 12 bit địa chỉ;

Năm 1978, bộ vi xử lý Intel 8086 ra đời với 16bit dữ liệu, 20 bit địa chỉ; tốc độ 10MHz

Năm 1979, bộ vi xử lý Intel 8088 ra đời nhưng vẫn tương thích với hệ thống 8086

Năm 1982 bộ vi xử lý 80286 ra đời với 16bit dữ liệu, 20 bit địa chỉ, tốc độ 20MHz

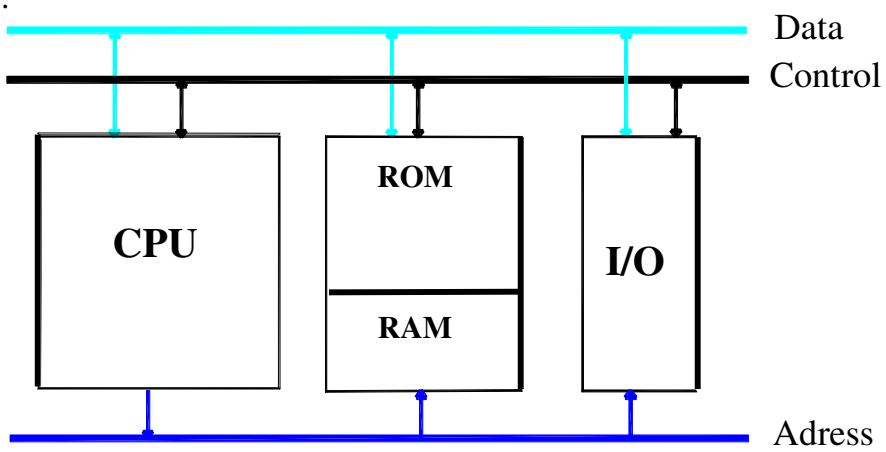
Năm 1985-1988, bộ vi xử lý 80386 ra đời với 32 bit dữ liệu và 32 bit địa …

Năm 1989, bộ vi xử lý 80486 ra đời với 32 bit dữ liệu và 32 bit địa chỉ tốc độ đến 60M

Năm 1993, bộ vi xử lý Pentium ra đời với 64 bit dữ liệu, tốc độ xử lý 100MHz sau đó là các bộ vi xử lý Pentium Pro,Pentium II, Pentium III, Celeron, Pentium 4…

1.1.1. Sơ đồ khối tổng quát của hệ Vi xử lý

❖ Sơ đồ khối :



***Hình 1.1** Cấu trúc các kênh thông tin trong hệ VXL*

❖ Chức năng các khối :

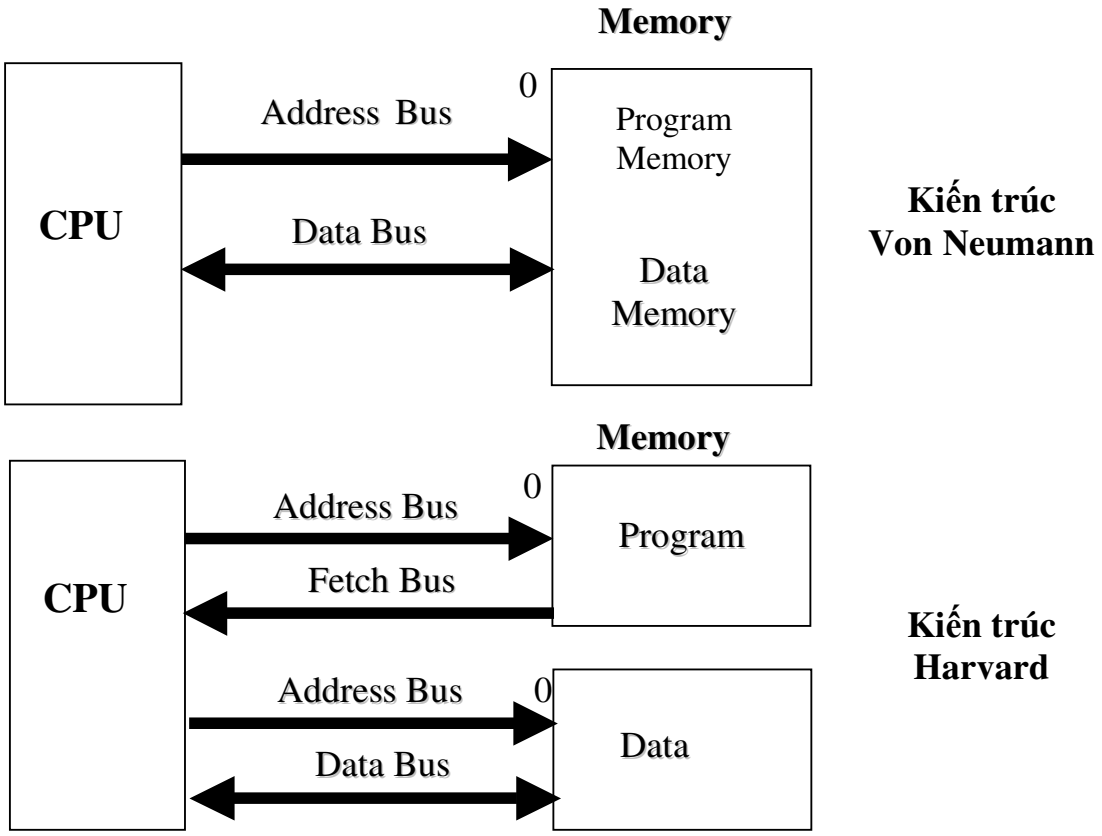
Bộ xử lý trung tâm CPU (Central Processing Unit) là trung tâm đầu não của hệ, nó có chức năng thu thập, xử lý thông tin và điều khiển mọi hoạt động của hệ vi xử lý.

Bộ nhớ trung tâm có nhiệm vụ lưu trữ thông tin dữ liệu trước khi CPU xử lý

Thiết bị I/O thực hiện việc nhận dữ liệu từ các kênh thông tin từ bên ngoài vào để cho CPU xử lý và xuất ra các tín hiệu điều khiển hệ thống

Các kênh thông tin hay Bus hệ thống là cầu nối liên kết giữa các khối trong đó thực hiện 3 việc chính là liên kết các Bus địa chỉ, Bus điều khiển và Bus dữ liệu.

❖ Tổ chức các kênh thông tin trong hệ Vi xử lý



***Hình 1.2*: Cấu trúc các kênh thông tin trong hệ VXL**

Trên đây là 2 cách tổ chức bộ nhớ theo kiểu Von Neumann và Harvard. Với kiểu tổ chức bộ nhớ chương trình và dữ liệu tách biệt cho phép tốc độ truy xuất thông tin nhanh hơn đáng kể. Các kênh dữ liệu đều là kênh song song và dùng chung cho tất cả các bộ nhớ, tuy nhiên nó phải được kiểm soát thông qua các cổng lôgic 3 trạng thái. Cổng này có nhiệm vụ tạo ra trạng thái đặc biệt khi có những thành phần không được kích hoạt làm việc, trạng thái đặc biệt này sẽ cách ly về mặt tín hiệu giữa kênh thông tin với từng thành phần trong hệ mặc dù chúng vẫn được kết nối với nhau về mặt vật lý.

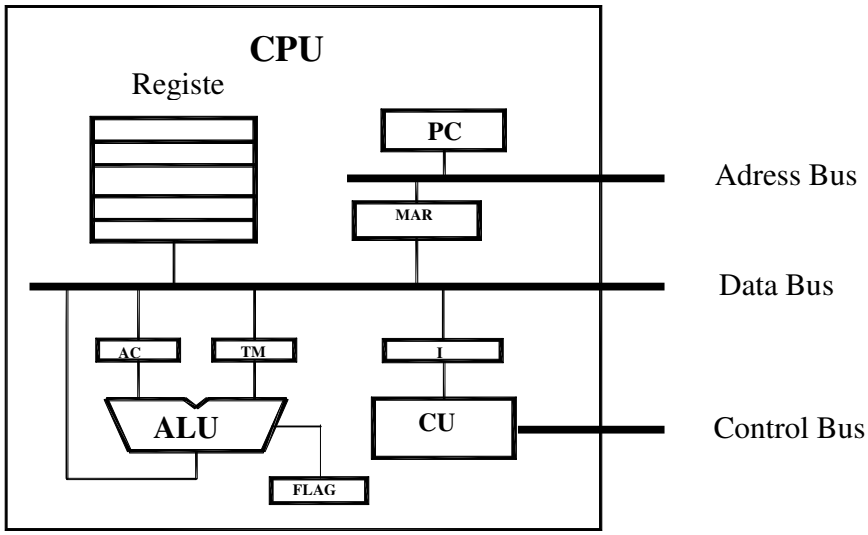
1.1.2. Khối xử lý trung tâm (Central Processing Unit)

Bộ vi xử lý là hạt nhân của hệ vi xử lý, nó là thành phần quan trọng nhất kiểm soát tất cả các hoạt động của hệ và thực hiện các thao tác trên dữ liệu. Hầu hết các CPU được hình thành từ các mạch lôgic nhằm thực hiện liên tục 2 thao tác là tìm nạp lệnh từ bộ nhớ để giải mã và thực thi lệnh. CPU có khả năng hiểu và thực thi các lệnh dựa trên một tập các mã nhị phân

gọi là mã máy trong đó mỗi mã nhằm thực hiện một thao tác nào đó.Tổ hợp các lệnh cho mỗi loại CPU gọi là tập lệnh và nó thường chia ra thành các nhóm lệnh như:

- *Nhóm lệnh chuyển dữ liệu*
- *Nhóm lệnh số học*
- *Nhóm lệnh Lôgic*
- *Nhóm lệnh rẽ nhánh chương trình*
- *Nhóm lệnh xử lý bít*

Cấu trúc đơn giản của một loại CPU được minh họa như sau:

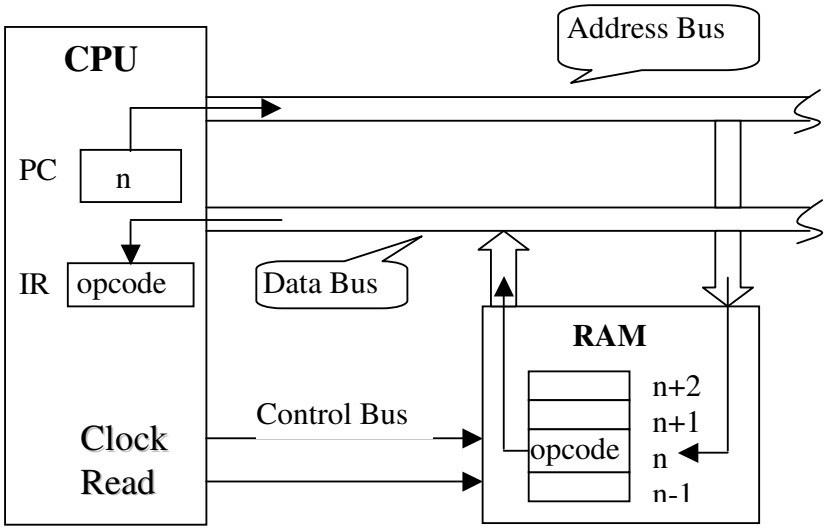


***Hình 1.3** Cấu trúc chung của một bộ CPU*

- PC (Program Couter): Bộ đếm trương trình có vai trò như một con trỏ, trỏ đến ô nhớ chứa lệnh mà CPU cần truy nhập
- IR (Instruction Register):Thanh ghi lệnh IR (Instruction Register) Thanh ghi lệnh thực hiện chức năng chứa lệnh mà CPU đang thực hiện.
- CU (Control Unit) Đơn vị điều khiển có chức năng giải mã lệnh.
- MAR (Memory Address Register) Thanh ghi chỉ bộ nhớ thực hiện chức năng chứa địa chỉ của ô nhớ hiện thời mà CPU đang truy nhập.
- ALU (Arithmetic Logic Unit) đơn vị số học logic, thực hiện các phép tính số học, logic và các phép xử lý dữ liệu khác.
- ACC (Accumulator) Thanh chứa , chứa toán hạng của một phép tính hoặc kết quả của phép tính.
- TMP (Temporary) Thanh ghi tạm, chứa toán hạng thứ hai của phép tính.
- FLAGS Thanh ghi cờ chứa thông tin về trạng thái kết quả phép tính sau khi thực hiện lệnh.
- Address Bus : Bus địa chỉ
- Data Bus : Bus dữ liệu
- Control Bus : Bus điều khiển

1.1.3. Quá trình truy xuất và xử lý thông tin

Việc tìm nạp một lệnh từ Ram hệ thống là một trong những thao tác cơ bản nhất mà CPU phải thực hiện. Quá trình tìm nạp lệnh được thực hiện theo trình tự như sau:



***Hình1.4** : Hoạt động của Bus cho chu kỳ tìm nạp lệnh*

Địa chỉ đang chứa trong PC sẽ được gửi lên trên bus địa chỉ.

Tín hiệu cho phép đọc lệnh từ bộ nhớ sẽ được kích hoạt sang trạng thái tích cực

Dữ liệu hay mã lệnh sẽ được đọc từ bộ nhớ và gửi lên kênh dữ liệu rồi chuyển về thanh ghi lệnh IR. Tiếp theo đó là nội dung của PC (tức địa chỉ) sẽ được tăng lên1để trỏ tới địa chỉ kế tiếp trong bộ nhớ.

Mã lệnh sẽ được chuyển xuống bộ giải mã và căn cứ theo mã lệnh CPU sẽ triển khai thực hiện lệnh.Trường hợp toán hạng nằm trong bộ nhớ chương trình ngay sau mã lệnh. CPU tiếp tục đưa PC nên bus địa chỉ để trỏ tới toán hạng rồi đưa ra thanh ghi để thực hiện lệnh.

VD: ADD A,#3EH

Nếu toán hạng nằm ngay trong thanh ghi của CPU, khi đó CPU thực hiện lệnh ngay

VD : ADD A,R1

Toán hạng nằm trong bộ nhớ mà địa chỉ của nó trong 1 thanh ghi của CPU .CPU đưa địa chỉ cho thanh ghi địa chỉ (MAR) để đọc dữ liệu và thực hiện lệnh. VD : ADD A,@R0

1.1.4. Bộ nhớ chỉ đọc *(Read Only Memory - ROM):*

❖ *ROM cơ bản*:

ROM dùng để lưu trữ chương trình điều hành (Monitor) của hệ VXL. Chương trình này sẽ quy định mọi hoạt động của hệ VXL. Bộ VXL sẽ căn cứ vào các lệnh chứa trong chương trình để điều khiển hệ VXL thực hiện các chức năng, nhiệm vụ được ấn định trong lệnh. Nói cách khác, hệ VXL sẽ thực hiện một cách trung thực thuật toán mà người thiết kế phần mềm đã xây dựng và cài đặt vào ROM của hệ.

Ngoài ra, ROM trong hệ VXL còn dùng để lưu trữ các bảng biểu, tham số của hệ thống mà trong quá trình hoạt động không được thay đổi như: bảng địa chỉ cổng giao tiếp, các bảng tra cứu số liệu, các bộ mã cần sử dụng trong hệ.

ROM cũng được quản lý theo phương thức ma trận điểm, nó có nhiều chủng loại khác nhau: ROM, PROM, EPROM, EEPROM,…

ROM là bộ nhớ cố định có cấu trúc đơn giản nhất. Nội dung của nó do nhà sản xuất chế tạo, người sử dụng không thể thay đổi nội dung này được nữa.

❖ *PROM (Programmable ROM - ROM có khả năng lập trình được):*

Đặc điểm chung: Nội dung của PROM do nhà sản xuất hoặc người thiết kế hệ VXL nạp vào nhưng chỉ đươc 1 lần. Sau khi nạp xong nội dung này không thể thay đổi được nữa.

❖ *EPROM (Eraseable PROM ROM nạp/xoá được nhiều lần):*

EPROM là bộ nhớ cố định có cấu trúc đặc biệt. Nội dung của nó do nhà sản xuất hay người thiết kế hệ VXL nạp vào và có thể nạp/xoá nhiều lần. Người ta tạo ra 1 bit thông tin trong EPROM dựa trên nguyên tắc làm việc của Transistor trường có cực cửa cách ly kênh cảm ứng (MOSFET kênh cảm ứng).

❖ *EEPROM (Electrical EPROM ROM có khả năng lập trình và xoá được bằng điện).*

1.1.5. Bộ nhớ W/R còn gọi là bộ nhớ truy cập ngẫu nhiên (RAM)

RAM là bộ nhớ có thể ghi và đọc được, thông tin trên RAM sẽ bị mất khi mất nguồn cung cấp. Theo phương thức lưu trữ thông tin, RAM được chia thành 2 loại cơ bản: RAM tĩnh và RAM động.

RAM tĩnh: Có thể lưu trữ thông tin lâu tuỳ ý miễn là được cung cấp điện năng - tất cả các loại phần tử nhớ bằng Trigơ đều thuộc loại này.

RAM động: Chỉ lưu được thông tin trong 1 khoảng thời gian nhất định. Muốn kéo dài thời gian này cần có phương thức làm tươi lại thông tin trong phần tử nhớ RAM. Phần tử nhớ của RAM động đơn giản nhất là một linh kiện điện dung - tụ diện. Sử dụng RAM động có phức tạp nhưng về cấu trúc nhớ lại đơn giản, tiêu tốn ít năng lượng, tăng mật độ bộ nhớ và đôi khi còn làm tăng cả tốc độ làm việc của bộ nhớ.

Cấu trúc mạch điện của các bộ nhớ RAM rất đa dạng cả về công nghệ chế tạo chúng (TTL, MOS,…) và các yêu cầu sử dụng chúng như các yêu cầu về ghép nối, tốc độ làm việc, mật độ linh kiện và dung lương cần thiết…

1.1.6. Các thiết bị xuất/nhập:

Các thiết bị xuất/nhập hay các thiết bị ngoại vi kết hợp với các mạch giao tiếp (Interface) sẽ tạo ra các đường truyền thông giữa hệ VXL với thế giới bên ngoài. Tuy nhiên để trao đổi thông tin giữa hệ VXL với các thiết bị ngoại vi, cần có các phương pháp điều khiển thích hợp như: Điều khiển vào/ra bằng chương trình, điều khiển vào/ra bằng ngắt, điều khiển vào/ra bằng phần cứng vv…

1.1.7. Cấu trúc kênh chung của hệ VXL:

Kênh (Bus) là tập hợp các đường thông tin có cùng mục đích. Để CPU có thể giao tiếp được với các bộ phận khác trong hệ VXL theo yêu cầu, mỗi hệ VXL cần sử dụng 3 kênh như :

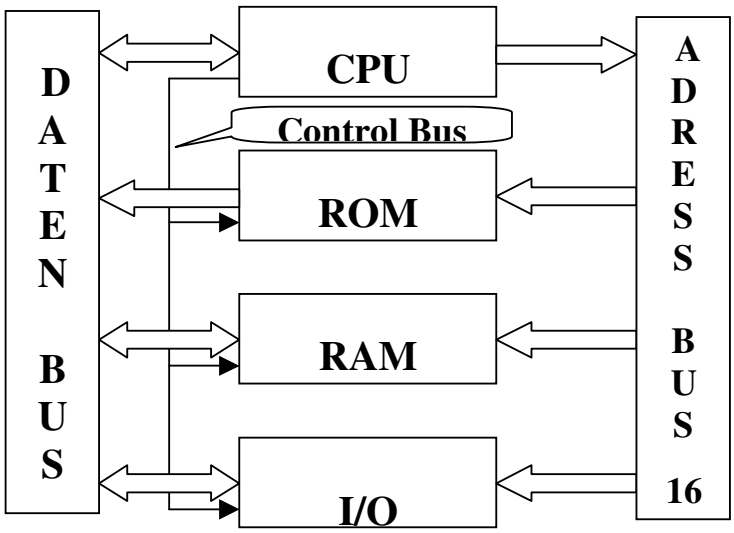
- Kênh địa chỉ (Adress Bus).
- Kênh dữ liệu (Daten Bus).
- Kênh điều khiển (Control Bus).

Để thực hiện thao tác đọc hoặc ghi, CPU xác định rõ vị trí (địa chỉ) của dữ liệu (hoặc lệnh) bằng cách đặt địa chỉ đó lên kênh địa chỉ, sau đó kích hoạt tín hiệu ***Read*** hoặc ***Write*** trên kênh điều khiển để chỉ ra thao tác là ***đọc*** hay ***ghi.***

Nếu kích hoạt tín hiệu điều khiển Read, thao tác đọc lấy 1 byte dữ liệu từ bộ nhớ ở vị trí đã xác định và đặt byte này lên kênh dữ liệu. CPU sẽ đọc dữ liệu và cất dữ liệu vào 1 trong các thanh ghi nội của CPU.

Nếu kích hoạt tín hiệu điều khiển Write, CPU sẽ thực hiện thao tác ghi bằng cách xuất dữ liệu lên kênh dữ liệu. Nhờ vào tín hiệu điều khiển, bộ nhớ nhận biết được đây là thao tác ghi và lưu dữ liệu vào vị trí đã được xác định.

Kênh dữ liệu cho phép trao đổi thông tin giữa CPU và bộ nhớ, cũng như giữa CPU với thiết bị ngoại vi. Thông thường các hệ VXL dành hầu hết thời gian cho việc di chuyển dữ liệu, đa số các thao tác di chuyển dữ liệu xảy ra giữa 1 thanh ghi của CPU với ROM và RAM ngoài. Do đó độ lớn của kênh dữ liệu ảnh hưởng rất lớn tới hiệu suất của hệ VXL. Nếu bộ nhớ của hệ thống rất lớn và CPU có khả năng tính toán cao, nhưng việc truy xuất dữ liệu – di chuyển dữ liệu giữa bộ nhớ và CPU thông qua kênh dữ liệu lại bị nghẽn thì hiện tượng "nghẽn cổ chai" này chính là hậu quả của độ rộng kênh dữ liệu không đủ lớn. Để khắc phục hiện tượng này, cần tăng đường tín hiệu cho kênh dữ liệu.



Hình 1.5 :Cấu trúc kênh chung của hệ thống VXL

Như ở hình 1.3, kênh dữ liệu là kênh 2 chiều, còn kênh địa chỉ là kênh 1 chiều. Các thông tin về địa chỉ luôn được cung cấp bởi CPU, trong khi các dữ liệu di chuyển theo cả 2 hướng tuỳ thuộc vào thao tác thực hiện là đọc hay ghi. Thuật ngữ "*dữ liệu*" được sử dụng theo nghĩa tổng quát: "*thông tin*" di chuyển trên kênh dữ liệu có thể là ***lệnh*** của chương trình, ***địa chỉ*** theo sau lệnh hoặc ***dữ liệu*** được sử dụng bởi chương trình.

Kênh điều khiển là tập hợp các tín hiệu, mỗi tín hiệu có một vai trò riêng trong việc điều khiển có trật tự hoạt động của hệ thống. Các tín hiệu điều khiển được cung cấp bởi CPU để đồng bộ việc di chuyển thông tin trên các kênh địa chỉ và dữ liệu. Các bộ VXL thường có 3 tín hiệu điều khiển: ***Read, Write, Clock***. Tuy nhiên tuỳ vào yêu cầu cụ thể cũng như cấu trúc phần cứng của từng hệ VXL mà số lượng tín hiệu điều khiển có thể khác nhau.

1.2. Các hệ thống số liên quan VĐK

❖ Hệ đếm thập phân (Decimal): Hệ đếm thập phân còn gọi là hệ đếm cơ số mười và nó được biểu diễn bởi 10 con số từ (0,1,2,3,4,5,6,7,8,9) những con số này được sử dụng rất nhiều trong khoa học kỹ thuật cũng như trong đời sống hàng ngày, khi biểu diễn số thập phân thì đứng sau dãy số thường có chữ D.

Ví dụ: Ba nghìn Chin trăm Bảy mươi Tám được biểu diễn như sau

$3978 = 3x10^3 + 9x10^2 + 7x10^1 + 8x10^0$

$= 3000 + 900 + 70 + 8$

❖ Hệ đếm thập lục phân (Hexadecimal): Hệ đếm thập phân còn gọi là hệ đếm cơ số mười sáu và nó được biểu diễn bởi 16 ký số (0,1,2,3,4,5,6,7,8,9,B,C,D,E,F) những con số này được sử dụng rất nhiều trong khoa học kỹ thuật đặc biệt là khoa học máy tính vì biểu diễn mã Hexa rất ngắn gọn, khi biểu diễn số thập lục phân thì sau dãy số phải có chữ H.

Ví dụ: 3978h , 12CCh, 1998h, ABCDh, 2008h …

❖ Hệ đếm nhị phân (Binary): Hệ đếm nhị phân còn gọi là hệ đếm cơ số hai và nó được biểu diễn bởi 2 con số là 0 và 1, trong kỹ thuật điện tử số thì số 0 gọi là mức logic thấp ứng với điện áp thấp, số 1gọi là mức logic cao tương ứng với điện áp cao nhất. Mỗi ký hiệu 0 hoặc 1 được gọi là 1 Bit (Binary Digit), khi biểu diễn số nhị phân thì đứng sau dãy số phải có chữ B.

Ví dụ:

1100b	; gọi là 1 nibble
10011001b	; gọi là 1 Byte
1010101111001101b	; gọi là 1 Word

Trong dãy số nhị phân được biểu diễn thì số nhị phân sát phải gọi là bít LSB còn số nhị phân sát trái gọi là bít MSB

Ví dụ: 1010101010101010

MSB LSB

Số nhị phân thường được biểu diễn ở 2 dạng là số nhị phân có dấu và số nhị phân không dấu, nếu số nhị phân không dấu sẽ chỉ biểu diễn các số không âm (≥0)
còn số nhị phân có dấu thì biểu diễn được cả giá trị âm

Ví dụ : $(1101) = 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 8 + 4 + 0 + 1 = 13$

Dải giá trị của các số có dấu 8 bít là [-128, +127]
Dải giá trị của các số có dấu 16 bít là [-32768, +32767]
Trong khi biểu diễn dãy số nhị phân có dấu thì người ta sử dụng bít MSB để quy ước cho bít dấu, với bít 0 cho dãy số nhị phân dương còn bít 1 cho dãy số nhị phân âm.

❖ Hệ đếm BCD (Binary Coded Decimal): Số đếm BCD được định nghĩa là số thập phân nhưng được biểu diễn dưới dạng nhị phân 4 bít, nhưng dãy số nhị phân 4 bít này khi quy sang hệ thập phân thì giá trị của nó phải ≤ 9 .Trong kỹ thuật điện tử nói chung thì mã BCD được sử dụng để giải mã hiển thị LED bảy thanh…

Ví dụ: (0011), (0100), (0101), (0110), …(1001) ; đây gọi là số BCD không nén

Ví dụ: (00110100), (01010110), … (01111000) ; đây gọi là số BCD nén

Dưới đây là bảng các mã BCD

Thập phân	BCD	Thập phân	BCD
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Ví dụ: Cho số thập phân là 15 , biểu diễn dưới dạng số BCD là 00010101

❖ Số bù 2: Trong kỹ thuật Vi xử lý để biểu diễn một con số nào đó dưới dạng dãy số nhị phân thì ngoài việc biểu diễn số không dấu, số có dấu thì người ta còn sử dụng cách biểu diễn số bù 2. Vậy số bù 2 sẽ biểu diễn như thế nào?

Ví dụ: Hãy biểu diễn dãy số A = 10011001 sang số bù 2 của nó:

- b1: Tìm số bù 1 của A(bằng cách lấy bù tất cả các bit của A): 01100110
- b2: Tìm số bù 2 của A (bằng cách lấy số bù 1 cộng cho 1) : 01100111

đến đây ta nhận thấy rằng số bù 2 của một số nào đó nó chính là số đối của nó và tổng = 0

❖ Phép cộng nhị phân không dấu :

Ví dụ : Cho 2 số nhị phân như sau A = 10010011 ; B = 00111001 hãy tìm tổng Y của 2 số nhị phân đã cho

```
A :      10010011b
B :      00111001b
-------------------
Y = A+B  = 11001100b
```

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 ; nhớ 1

❖ Phép trừ nhị phân:

Ví dụ : Cho 2 số nhị phân như sau A = 10010011 ; B = 00111001 hãy tìm hiệu Z của 2 số nhị phân đã cho *(Lưu ý rằng phép trừ có thể thực hiện bằng cách biến thành phép cộng)*

```
A :      10010011b
B :      00111001b
-------------------
Y = A - B = 01011010b
```

0 - 0 = 0
0 - 1 = 1 ; mượn 1
1 - 0 = 1
1 - 1 = 0

❖ Phép nhân nhị phân :

Ví dụ : Cho 2 số nhị phân như sau A = 00100101 ; B = 00000100 hãy tìm tích F

Khi nhân 2 dãy số nhị phân với nhau thì ta đặt phép toán nhân giống như nhân số thập phân, kết quả của phép nhân 2 dãy số nhị phân 8 bít sẽ thu được dãy số nhị phân là 16 bít, như vậy ta có F = A*B = 0000000010010100b

❖ Phép chia nhị phân :

Ví dụ : Cho 2 số nhị phân như sau A = 10010110 ; B = 00000100 hãy tìm thương

Khi chia 2 dãy số nhị phân với nhau thì ta đặt phép toán chia cũng giống như khi chia 2 số thập phân, kết quả của phép chia cũng như phần dư (nếu có) thu được tương tự như khi làm phép chia đối với số thập phân, như vậy ta có M = A/B = 00100101b dư 0010b

❖ Chuyển đổi số thập phân sang nhị phân :

Để chuyển đổi số thập phân sang số nhị phân người ta thường dùng phương pháp lấy số thập phân cần chuyển rồi chia 2 liên tiếp đến khi không thể chia được nữa thì dừng

Ví dụ : Chuyển số thập phân 25 sang số nhị phân không dấu

Chia 2		Thương số	Dư số	
25/2	=	12	1	LSB
12/2	=	6	0	
6/2	=	3	0	
3/2	=	1	1	
1/2	=	0	1	MSB

Kết quả thu được là: 11001 với số dư lần thứ nhất là bít có trọng số nhỏ nhất

❖ Chuyển đổi số nhi phân sang thập phân :

Để chuyển đổi số nhị phân sang số thập phân người ta thường dùng phương pháp lấy tổng của tích n các số nhị phân cần chuyển nhân với 2^0 đến 2^{N-1} hay theo biểu thức tổng quát như sau: $A = B_{(N-1)}* 2^{(N-1)} + B_{(N-2)}* 2^{(N-2)} + B_{(N-3)}* 2^{(N-3)} + \ldots + B_{(1)}* 2^{(1)} + B_{(0)}* 2^{(0)}$

Ví dụ : Chuyển số nhị phân không dấu 01011110b sang số thập phân

$A = 0*2^7 + 1*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = 94$

như vậy ta có 01011110b = 94

❖ Chuyển đổi số nhị phân sang Hexa :

Ví dụ : Chuyển số nhị phân 1100101011111110 sang số hexa

Trước hếtta chia số nhị phân đã cho thành các nhóm 4-bit tính từ bít có trọng số nhỏ nhất, sau đó thay thế mỗi nhóm 4-bit bằng ký hiệu hexa tương ứng với nó ta sẽ thu được kết quả như sau:

1100	1010	1111	1110	hay 1100101011111110b = CAFEh
C	A	F	E	

❖ Chuyển đổi số Hexa sang nhị phân:

Ví dụ : Chuyển số hexa 2F8h và ABBAh sang số nhị phân

Tương tự như trường hợp trên ta sẽ thay thế mỗi ký hiệu hexa bằng 4-bit nhị phân tương ứng với nó và ta sẽ thu được kết quả như sau:

2	F	8		
0010	1111	1000		hay 2F8h = 001011111000b
A	B	B	A	
1010	1011	1011	1010	hay ABBAh = 1010101110111010b

❖ Mã ASCII: (American Standard Code for Information Interchange)

Quá trình trao đổi thông tin trong máy tính nói chung cũng như quá trình xử lý thông tin của các bộ vi xử lý, tất cả các thông tin đều được biểu diễn dưới dạng các số 0 và 1. Mỗi tổ hợp số 0 hoặc 1 được gán một ký tự chữ cái, chữ số hoặc một ký tự đặc biệt nào đó. Khi thông tin được truyền đi, được lưu giữ trong bộ nhớ và hiển thị trên màn hình đều ở dưới dạng ký tự và tuân theo một loại mã chuẩn được sử dụng rất rộng rãi gọi là mã ASCII.

BẢNG MÃ ASSII (American Standard Code for Information Interchange)

Decimal	Hex	ASCII	Decimal	Hex	ASCII	Decimal	Hex	ASCII	Decimal	Hex	ASCII
0	0	NUL	32	20		64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	HT	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SOH	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	□

BẢNG MÃ ASSII mở rộng

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

CHƯƠNG II : BỘ VI ĐIỀU KHIỂN 80C51/89C51

2.1. Giới thiệu chung về các bộ Vi điều khiển.

Vi điều khiển (VĐK) là một "hệ" Vi xử lý (VXL) được tổ chức trong một chip. Nó bao gồm:

- Bộ VXL (CPU)
- Bộ nhớ chương trình (ROM/EPROM/EEPROM/FLASH).
- Bộ nhớ dữ liệu (RAM).
- Các thanh ghi chức năng, các cổng I/O, cơ chế điều khiển ngắt và truyền tin nối tiếp.
- Các bộ thời gian dùng trong lĩnh vực chia tần và tạo thời gian thực.
- ...

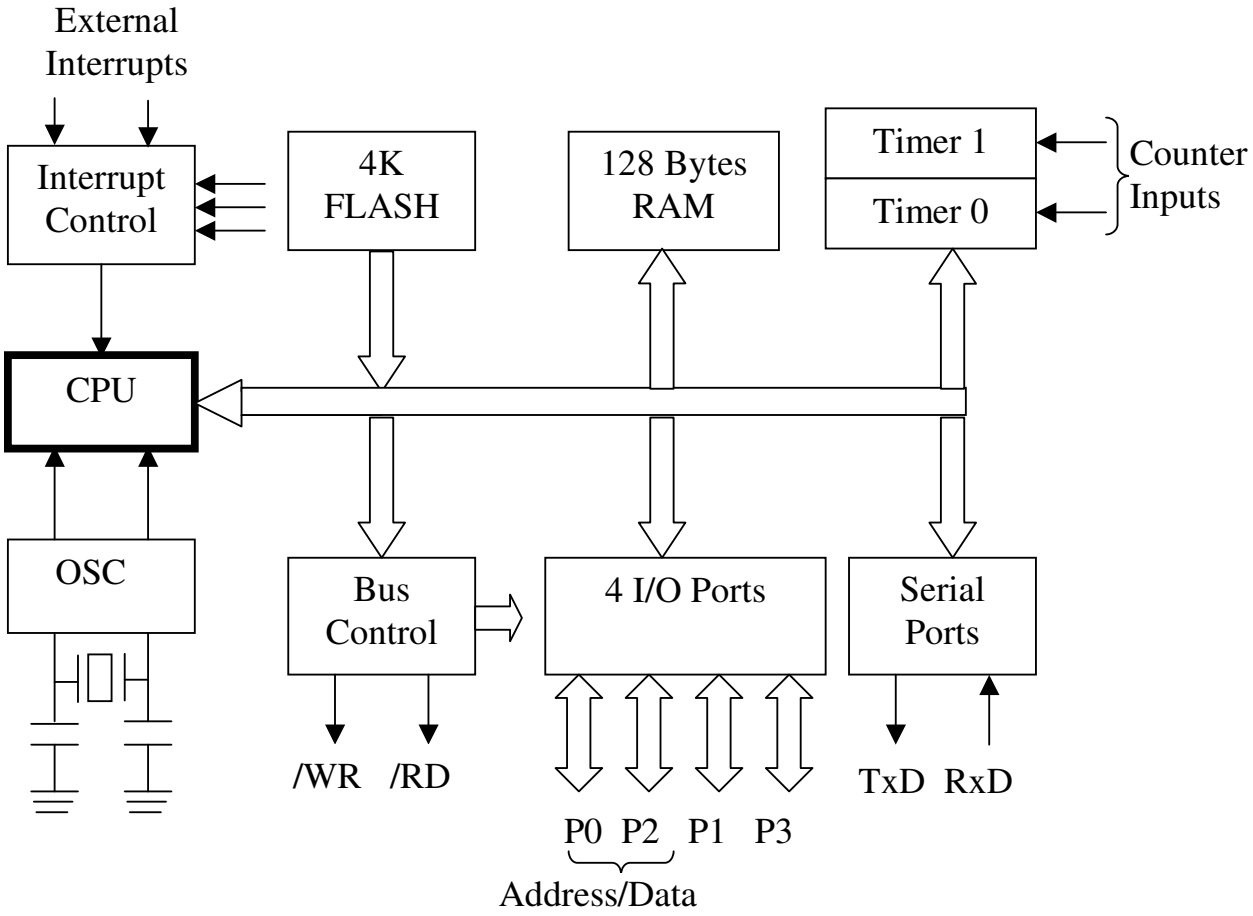
Bộ VĐK có thể được lập trình để điều khiển các thiết bị thông tin, viễn thông, thiết bị đo lường, thiết bị điều chỉnh cũng như các ứng dụng trong công nghệ thông tin và kỹ thuật điều khiển tự động. Có thể xem bộ VĐK như một hệ VXL On-chip, đối với AT89C51, nó có đầy đủ chức năng của một hệ VXL 8 bit, đươc điều khiển bởi một hệ lệnh, có số lệnh đủ mạnh, cho phép lập trình bằng hợp ngữ (Assembly).

2.2. Sự khác nhau giữa bộ Vi xử lý và bộ Vi điều khiển.

Tiêu chí SS	**VXL**	**VĐK**
Phần cứng	CPU đơn chíp.	CPU, RAM, ROM, Timers, SFR, mạch giao tiếp, hệ thống ngắt và cơ chế điều khiển ngắt…..
Tập lệnh	Sử dụng các tập lệnh bao quát, mạnh về kiểu định địa chỉ. Các lệnh này có thể truy xuất dữ liệu lớn, thực hiện ở dạng 1/2 Byte, Byte, Word, Double Word.	Sử dụng các lệnh điều khiển xuất nhập, có thể truy xuất dữ liệu ở dạng Bit hoặc Byte. Các nhóm lệnh chính: Chuyển dữ liệu, điều khiển biến logic, rẽ nhánh chương trình, tính toán số học và logic.
Ứng dụng	Trong các hệ máy vi tính.	Trong các hệ thống điều khiển, đo lường và điều chỉnh…

2.3. Cấu trúc chung của bộ Vi điều khiển 80C51.

2.3.1. Sơ đồ khối.



Hình 2.1 : Sơ đồ khối họ VĐK AT89C51

Bộ VĐK 8 bit AT89C51 hoạt động ở tần số 12 MHz, với bộ nhớ ROM 4Kbyte, bộ nhớ RAM 128 Byte cư trú bên trong và có thể mở rộng bộ nhớ ra ngoài. Ở bộ VĐK này còn có 4 cổng 8 bit (P0…P3) vào/ ra 2 chiều để giao tiếp với thiết bị ngoại vi. Ngoài ra, nó còn có:

- CPU
- 2 bộ đinh thời 16 bit (Timer 0 và Timer 1)
- Mạch giao tiếp nối tiếp.
- Bộ xử lý bit (thao tác trên các bit riêng rẽ).
- Hệ thống điều khiển và xử lý ngắt.
- Các kênh điều khiển/ dữ liệu/ địa chỉ.
- Các thanh ghi chức năng đặc biệt (SFR).

Tuy nhiên, tuỳ thuộc vào từng họ VĐK của từng hãng sản xuất khác nhau mà tính năng cũng như phạm vi ứng dụng của mỗi bộ VĐK là khác nhau, và chúng được thể hiện trong các bảng thống kê sau:

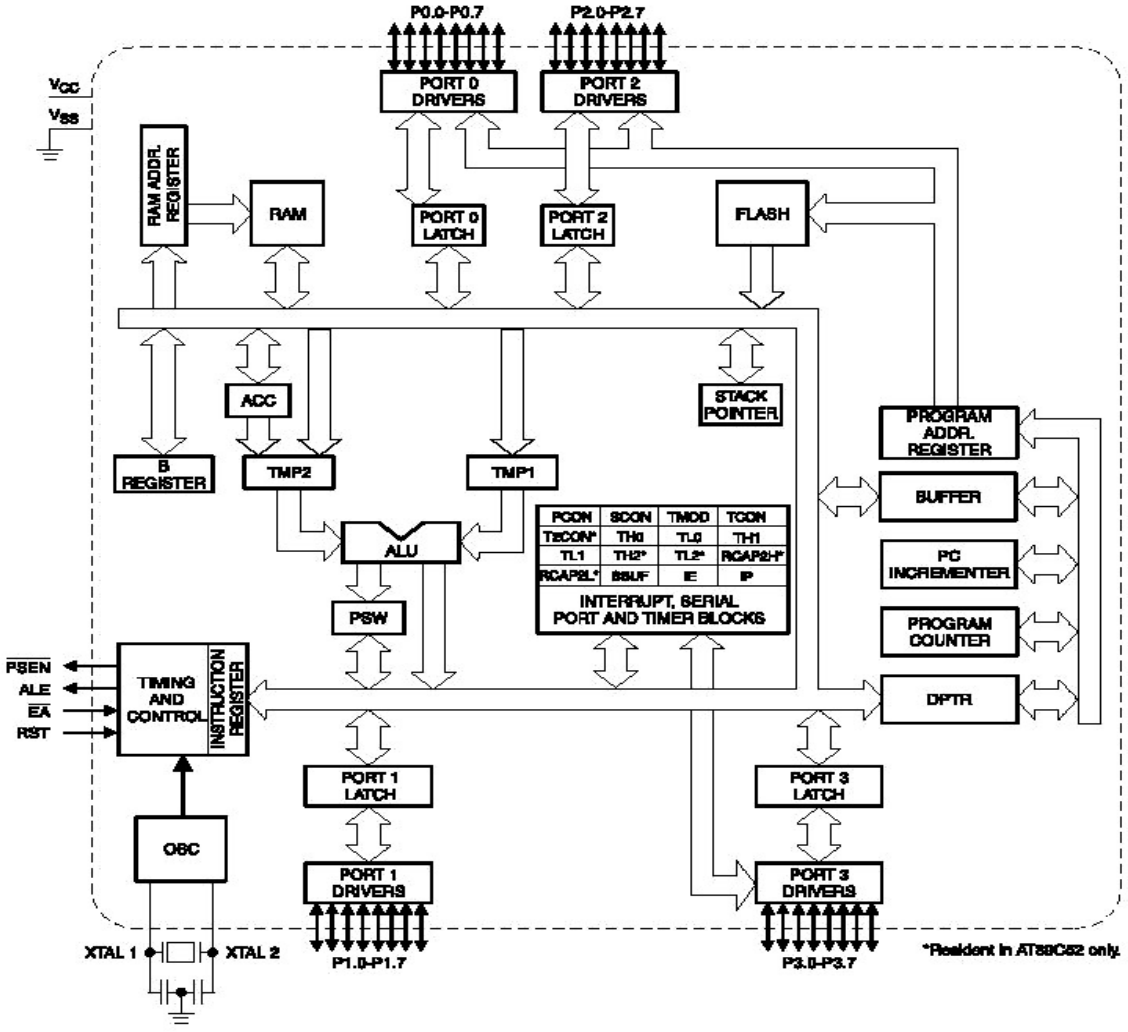
Họ VĐK	ROM (bytes)	RAM (bytes)	Tốc độ (MHz)	Chân I/O	Timer/ Counter	UART	Ngắt
8051							
8031AH	ROMLESS	128	12	32	2	1	5
8051AH	4K ROM	128	12	32	2	1	5
8051AHP	4K ROM	128	12	32	2	1	5
8751H	4K EPROM	128	12	32	2	1	5
8751BH	4K EPROM	128	12	32	2	1	5
8052							
8032AH	ROMLESS	256	12	32	3	1	6
8052AH	8K ROM	256	12	32	3	1	6
8752BH	8K EPROM	256	12	32	3	1	6
80C51				32			
80C31BH	ROMLESS	128	12,16	32	2	1	5
80C51BH	4K ROM	128	12,16	32	2	1	5
80C31BH	4K ROM	128	12,16	32	2	1	5
87C51	4K EPROM	128	12,16,20,24	32	2	1	5
8xC52/54/58							
80C32	ROMLESS	256	12,16,20,24	32	3	1	6
80C52	8K ROM	256	12,16,20,24	32	3	1	6
87C52	8K EPROM	256	12,16,20,24	32	3	1	6
80C54	16K ROM	256	12,16,20,24	32	3	1	6
87C54	16K EPROM	256	12,16,20,24	32	3	1	6
Họ VĐK	**ROM (bytes)**	**RAM (bytes)**	**Tốc độ (MHz)**	**Chân I/O**	**Timer/ Counter**	**UART**	**Ngắt**
80C58	32K ROM	256	12,16,20,24	32	3	1	6
87C58	32K EPROM	256	12,16,20,24	32	3	1	6
8xL52/54/58							
80L52	8K ROM	256	12,16,20	32	3	1	6
87L52	8K OTP ROM	256	12,16,20	32	3	1	6
80L54	16K ROM	256	12,16,20	32	3	1	6
87L54	16KOTP ROM	256	12,16,20	32	3	1	6
80L58	32K ROM	256	12,16,20	32	3	1	6
87L58	32KOTP ROM	256	12,16,20	32	3	1	6

Họ VĐK	*Bộnhớ ROM(Bytes)*	*Bộ nhớ dữ liệu (Bytes)*	*Timer 16 bit*	*Công nghệ*
AT89C1051	1K Flash	64 RAM	1	CMOS
AT89C2051	2K Flash	128 RAM	2	CMOS
AT89C51	4K Flash	128 RAM	2	CMOS
AT89C52	8K Flash	256 RAM	3	CMOS
AT89C55	20K Flash	256 RAM	3	CMOS
AT89S8252	8K Flash	256RAM+2K EPROM	3	CMOS
AT89S53	12K Flash	256 RAM	3	CMOS

Bảng 2.1. Các thông số của các họ VĐK thuộc hãng Intel _ Atmel (MSC 51)

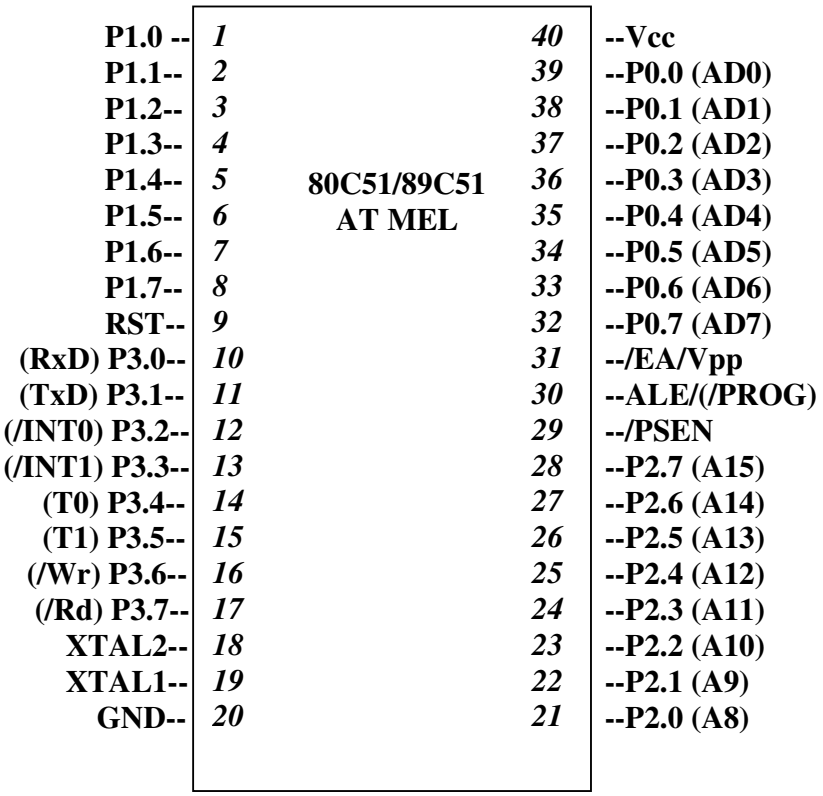
Trong thực tế có rất nhiều họ Vi điều khiển được sản xuất để sử dụng cho đa mục đích, tuy nhiên trong khuôn khổ tài liệu này chỉ giới hạn ở một số vi điều khiển của hãng ATMEL

Figure 1. AT89 Series Flash-Based Microcontroller Architectural Block Diagram



***Hình 2.2** : Cấu trúc các thanh ghi bên trong của VĐK 80C51/89C51*

2.3.2. Sơ đồ chân tín hiệu.



***Hình 2.3** : Sơ đồ chân tín hiệu của VXL 80C51/89C51*

Chức năng của các chân tín hiệu như sau:

- P0.0 đến P0.7 là các chân của cổng 0.
- P1.0 đến P1.7 là các chân của cổng 1.
- P2.0 đến P2.7 là các chân của cổng 2
- P3.0 đến P3.7 là các chân của cổng 3
- RxD: Nhận tín hiệu kiểu nối tiếp.
- TxD: Truyền tín hiệu kiểu nối tiếp.
- /INT0: Ngắt ngoài 0.
- /INT1: Ngắt ngoài 1.
- T0: Chân vào 0 của bộ Timer/Counter 0.
- T1: Chân vào 1 của bộ Timer/Counter 1.
- /Wr: Ghi dữ liệu vào bộ nhớ ngoài.
- /Rd: Đọc dữ liệu từ bộ nhớ ngoài.
- RST: Chân vào Reset, tích cực ở mức logic cao trong khoảng 2 chu kỳ máy.
- XTAL1: Chân vào mạch khuyếch đại dao động
- XTAL2: Chân ra từ mạch khuyếch đại dao động.
- /PSEN : Chân cho phép đọc bộ nhớ chương trình ngoài (ROM ngoài).

- ALE (/PROG): Chân tín hiệu cho phép chốt địa chỉ để truy cập bộ nhớ ngoài, khi On-chip xuất ra byte thấp của địa chỉ. Tín hiệu chốt được kích hoạt ở mức cao, tần số xung chốt = 1/6 tần số dao động của bộ VĐK. Nó có thể được dùng cho các bộ Timer ngoài hoặc cho mục đích tạo xung Clock. Đây cũng là chân nhận xung vào để nạp chương trình cho Flash (hoặc EEPROM) bên trong On-chip khi nó ở mức thấp.
- /EA/Vpp: Cho phép On-chip truy cập bộ nhớ chương trình ngoài khi /EA=0, nếu /EA=1 thì On-chip sẽ làm việc với bộ nhớ chương trình nội trú (trường hợp cần truy cập vùng nhớ lớn hơn dung lượng bộ nhớ chương trình nội trú, thì bộ nhớ chương trình ngoài cũng được sử dụng). Khi chân này được cấp nguồn điện áp 12V (Vpp) thì On-chip đảm nhận chức năng nạp chương trình cho Flash bên trong nó.
- Vcc: Cung cấp dương nguồn cho On-chip (+ 5V).
- GND: nối Mass.

2.4. Các thanh ghi chức năng đặc biệt.

Thanh ghi	***MSB***	**Nội dung**						***LSB***
IE	EA	-	ET2	ES	ET1	EX1	ET0	EX0
IP	-	-	PT2	PS	PT1	PX1	PT0	PX0
PSW	CY	AC	FO	RS1	RS0	OV	-	P
TMOD	GATE	C/(/T)	M1	M0	GATE	C/(/T)	M1	M0
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
PCON	SMOD	-	-	-	GF1	GF0	PD	IDL
P1	T2	T2EX			/SS	MOSI	MISO	SCK
P3	RXD	TXD	/INT0	/INT1	T0	T1	/WR	/RD

❖ ***Bảng địa chỉ trực tiếp của các thanh ghi đặc biệt được lưu trữ trong RAM***

Symbol	**Name**	**Address**	**Reset Values**
* ACC	Thanh ghi tích luỹ	0E0h	00000000b
* B	Thanh ghi B	0F0h	00000000b
* PSW	Từ trạng thái chương trình	0D0h	00000000b
SP	Con trỏ ngăn xếp	81h	00000111b
DP0L	Byte cao của con trỏ dữ liệu 0	82h	00000000b
DP0H	Byte thấp của con trỏ dữ liệu 0	83h	00000000b
* P0	Cổng 0	80h	11111111b
* P1	Cổng 1	90h	11111111b

Symbol	Name	Address	Reset Values
* P2	Cổng 2	0A0h	11111111b
* P3	Cổng 3	0B0h	11111111b
* IP	TG điều khiển ngắt ưu tiên	0B8h	xxx00000b
* IE	TG điều khiển cho phép ngắt	0A8h	0xx00000b
TMOD	Điều khiển kiểu Timer/Counter	89h	00000000b
* TCON	TG điều khiển Timer/Counter	88h	00000000b
TH0	Byte cao của Timer/Counter 0	8Ch	00000000b
TL0	Byte thấp của Timer/Counter 0	8Ah	00000000b
TH1	Byte cao của Timer/Counter 1	8Dh	00000000b
TL1	Byte thấp của Timer/Counter 1	8Bh	00000000b
* SCON	Serial Control	98h	00000000b
SBUF	Serial Data Buffer	99h	indeterminate
PCON	Power Control	87h	0xxx0000b
* : có thể định địa chỉ bit, x: không định nghĩa			

***Bảng 2.2* : Địa chỉ, ý nghĩa và giá trị của các SFR sau khi Reset**

2.4.1. Thanh ghi ACC:

ACC.7	ACC.6	ACC.5	ACC.4	ACC.3	ACC.2	ACC.1	ACC.0

Thanh ghi ACC là thanh ghi tích luỹ, nó có độ dài 8 bits và dùng để lưu trữ kết quả của phép tính. Trong các tập lệnh của On-chip, nó thường được quy ước đơn giản là A.

2.4.2. Thanh ghi B :

Thanh ghi B cũng có độ dài 8 bít . Nó thường được dùng chung với thanh ghi A trong các phép toán nhân hoặc chia. Khi nhân thì nó còn lưu trữ kết quả của byte cao còn khi chia thì dùng để lưu kết quả phần dư. Đối với các lệnh khác, nó có thể xem như là thanh ghi đệm tạm thời. Thanh ghi B dài 8 bits.

2.4.3. Thanh ghi SP:

Thanh ghi con trỏ ngăn xếp dài 8 bit. SP chứa địa chỉ của dữ liệu hiện đang hiện hành ở đỉnh của ngăn xếp hay nối khác là SP luôn trỏ tới ngăn nhớ sử dụng cuối cùng (gọi là đỉnh ngăn xếp). Giá trị của nó được tự động tăng lên khi thực hiện lệnh PUSH trước khi dữ liệu được lưu trữ trong ngăn xếp. SP sẽ tự động giảm xuống khi thực hiện lệnh POP. Ngăn xếp có thể đặt ở bất cứ nơi nào trong RAM on-chip, nhưng sau khi khởi động lại hệ thống thì con trỏ ngăn xếp mặc định sẽ trỏ tới địa chỉ khởi đầu là 07h, vì vậy ngăn xếp sẽ bắt đầu từ địa chỉ 08h. Ta cũng có thể định con trỏ ngăn xếp tại địa chỉ mong muốn bằng các lệnh di chuyển dữ liệu thông qua định địa chỉ tức thời.

Nói thêm về ngăn xếp: Ngăn xếp là một vùng của bộ nhớ RAM dùng để lưu trữ thông tin tạm thời (có thể là dữ liệu hoặc địa chỉ), lý do cần có không gian lưu trữ này là vì số lượng

thanh ghi có hạn. Ngăn xếp chiếm 1 vùng nhớ có địa chỉ từ 08h ÷1Fh tức là toàn bộ 3 bank thanh ghi1,2,3 (gồm24 Byte). Nếu trong 1 chương trình mà cần phải có ngăn xếp > 24 Byte thì phải gán địa chỉ cho ngăn xếp lên vùng nhớ có địa chỉ từ 30h trở lên. Nhớ rằng khi reset hệ thống thì giá trị của SP = 07h.

2.4.4. Thanh ghi DPTR:

Dph								**Dpl**							

Thanh ghi con trỏ dữ liệu (16 bit) bao gồm 1 thanh ghi byte cao (DPH-8bit) và 1 thanh ghi byte thấp (DPL-8bit). DPTR có thể được dùng như thanh ghi 16 bit hoặc 2 thanh ghi 8 bit độc lập. Thanh ghi này được dùng để truy cập RAM ngoài.

2.4.5. Ports 0 to 3:

P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0

P0, P1, P2, P3 là các chốt của các cổng 0, 1, 2, 3 tương ứng. Mỗi chốt gồm 8 bit. Khi ghi mức logic 1 vào một bit của chốt, thì chân ra tương ứng của cổng ở mức logic cao. Còn khi ghi mức logic 0 vào mỗi bit của chốt thì chân ra tương ứng của cổng ở mức logic thấp. Khi các cổng đảm nhiệm chức năng như các đầu vào thì trạng thái bên ngoài của các chân cổng sẽ được giữ ở bit chốt tương ứng. Tất cả 4 cổng của on-chip đều là cổng I/O hai chiều, mỗi cổng đều có 8 chân ra, bên trong mỗi chốt bit có bộ "Pullup-tăng cường" do đó nâng cao khả năng nối ghép của cổng với tải (có thể giao tiếp với 4 đến 8 tải loại TTL).

2.4.6. Thanh ghi SBUF:

SBUF							
SBUF							

Đệm dữ liệu nối tiếp gồm 2 thanh ghi riêng biệt, một thanh ghi đệm phát và một thanh ghi đệm thu. Khi dữ liệu được chuyển tới SBUF, nó sẽ đi vào bộ đệm phát, và được giữ ở đấy để chế biến thành dạng truyền tin nối tiếp. Khi dữ liệu được truyền đi từ SBUF, nó sẽ đi ra từ bộ đệm thu.

2.4.7. Các Thanh ghi Timer: Các đôi thanh ghi (TH0, TL0), (TH1, TL1) là các thanh ghi đếm 16 bit tương ứng với các bộ Timer/Counter 0 và 1.

TH_X								**TL_X**							

2.4.8. Các thanh ghi điều khiển: Các thanh ghi chức năng đặc biệt: IP, IE, TMOD, TCON, SCON, và PCON bao gồm các bit trạng thái và điều khiển đối với hệ thống ngắt, các bộ Timer/Counter và cổng nối tiếp. Chúng sẽ được mô tả ở phần sau.

2.4.9. Thanh ghi PSW:

CY	AC	FO	RS1	RS0	OV	-	P

Từ trạng thái chương trình dùng để chứa thông tin về trạng thái chương trình. PSW có độ dài 8 bit, mỗi bit đảm nhiệm một chức năng cụ thể. Thanh ghi này cho phép truy cập ở dạng mức bit.

* CY: Cờ nhớ. Trong các phép toán số học, nếu có nhớ từ phép cộng bit 7 hoặc có số mượn mang đến bit 7 thì CY được đặt bằng 1.

* AC: Cờ nhớ phụ (Đối với mã BCD). Khi cộng các giá trị BCD, nếu có một số nhớ được tạo ra từ bit 3 chuyển sang bit 4 thì AC được đặt bằng 1. Khi giá trị được cộng là BCD, lệnh cộng phải được thực hiện tiếp theo bởi lệnh ***DA A*** (hiệu chỉnh thập phân thanh chứa A) để đưa các kết quả lớn hơn 9 về giá trị đúng.

* F0: Cờ 0 (Có hiệu lực với các mục đích chung của người sử dụng)

* RS1: Bit 1 điều khiển chọn băng thanh ghi.

* RS0: Bit 0 điều khiển chọn băng thanh ghi.

Lưu ý: RS0, RS1 được đặt/xoá bằng phần mềm để xác định băng thanh ghi đang hoạt động (Chọn băng thanh ghi bằng cách đặt trạng thái cho 2 bit này)

	RS1 (PSW. 4)	**RS0 (PSW. 3)**
Bank 0	0	0
Bank 1	0	1
Bank 2	1	0
Bank 3	1	1

Bảng 2.3* : *Chọn băng thanh ghi

* OV: Cờ tràn. Khi thực hiện các phép toán cộng hoặc trừ mà xuất hiện một tràn số học, thì OV được đặt bằng 1. Khi các số có dấu được cộng hoặc được trừ, phần mềm có thể kiểm tra OV để xác định xem kết quả có nằm trong tầm hay không. Với phép cộng các số không dấu, OV được bỏ qua. Kết quả lớn hơn +128 hoặc nhỏ hơn -127 sẽ đặt OV=1.

* -: Bit dành cho người sử dụng tự định nghĩa(Nếu cần).

* P: Cờ chẵn lẻ. Được tự động đặt/ xoá bằng phần cứng trong mỗi chu trình lệnh để chỉ thị số chẵn hay lẻ của bit 1 trong thanh ghi tích luỹ. Số các bit 1 trong A cộng với bit P luôn luôn là số chẵn.

2.4.10. Thanh ghi PCON: Thanh ghi điều khiển nguồn.

SMOD	-	-	-	GF1	GF0	PD	IDL

* SMOD: Bit tạo tốc độ Baud gấp đôi. Nếu Timer 1 được sử dụng để tạo tốc độ baud và SMOD=1, thì tốc độ Baud được tăng lên gấp đôi khi cổng truyền tin nối tiếp được dùng bởi các kiểu 1, 2 hoặc 3.

* -: Không sử dụng, các bit này có thể được dùng ở các bộ VXL trong tương lai. Người sử dụng không được phép tự định nghĩa cho các bit này.

* GF0, GF1: Cờ dùng cho các mục đích chung (đa mục đích).

* PD: bit nguồn giảm. Đặt bit này ở mức tích cực để vận hành chế độ nguồn giảm trong AT89C51. Chỉ có thể ra khỏi chế độ bằng Reset.

* IDL: bit chọn chế độ nghỉ. Đặt bit này ở mức tích cực để vận hành kiểu Idle (Chế độ không làm việc) trong AT89C51.

Lưu ý: Nếu PD và IDL cùng được kích hoạt cùng 1 lúc ở mức tích cực, thì PD được ưu tiên thực hiện trước. Chỉ ra khỏi chế độ bằng 1 ngắt hoặc Reset lại hệ thống.

2.4.11. Thanh ghi IE: Thanh ghi cho phép ngắt

EA	-	ET2	ES	ET1	EX1	ET0	EX0

* EA: Nếu EA=0, không cho phép bất cứ ngắt nào hoạt động.

* Nếu EA=1, mỗi nguồn ngắt riêng biệt sẽ phụ thuộc và bít cho phép ngắt tương ứng

* -: Không dùng, người sử dụng không nên định nghĩa cho Bit này, bởi vì nó có thể được dùng ở các bộ AT89 trong tương lai.

* ET2: Bit cho phép hoặc không cho phép ngắt bộ Timer 2.

* ES: Bit cho phép hoặc không cho phép ngắt cổng nối tiếp (SPI và UART).

* ET1: Bit cho phép hoặc không cho phép ngắt tràn bộ Timer 1

* EX1: Bit cho phép hoặc không cho phép ngắt ngoài 1.

* ET0: Bit cho phép hoặc không cho phép ngắt tràn bộ Timer 0

* EX0: Bit cho phép hoặc không cho phép ngắt ngoài 0.

2.4.12. Thanh ghi IP: Thanh ghi ưu tiên ngắt**.**

-	-	PT2	PS	PT1	PX1	PT0	PX0

* - : Không dùng, người sử dụng không nên ghi "1" vào các Bit này.

* PT2: Xác định mức ưu tiên của ngắt Timer 2.

* PS: Định nghĩa mức ưu tiên của ngắt cổng nối tiếp.

* PT1: Định nghĩa mức ưu tiên của ngắt Timer 1.

* PX1: Định nghĩa mức ưu tiên của ngắt ngoàI 1.

* PT0: Định nghĩa mức ưu tiên của ngắt Timer 0.

* PX0: Định nghĩa mức ưu tiên của ngắt ngoàI 0.

2.4.13. Thanh ghi TCON : Thanh ghi điều khiển bộ Timer/Counter

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

* TF1: Cờ tràn Timer 1. Được đặt bởi phần cứng khi bộ Timer 1 tràn. Được xoá bởi phần cứng khi bộ vi xử lý hướng tới chương trình con phục vụ ngắt.

TR1: Bit điều khiển bộ Timer 1 hoạt động. Được đặt/xoá bởi phần mềm để điều khiển bộ Timer 1 ON/OFF

* TF0: Cờ tràn Timer 0. Được đặt bởi phần cứng khi bộ Timer 0 tràn. Được xoá bởi phần cứng khi bộ vi xử lý hướng tới chương trình con phục vụ ngắt.

* TR0: Bit điều khiển bộ Timer 0 hoạt động. Được đặt/xoá bởi phần mềm để điều khiển bộ Timer 0 ON/OFF.

* IE1: Cờ ngắt ngoài 1. Được đặt bởi phần cứng khi sườn xung của ngắt ngoài 1 được phát hiện. Được xoá bởi phần cứng khi ngắt được xử lý.

* IT1: Bit điều khiển ngắt 1 để tạo ra ngắt ngoài. Được đặt/xoá bởi phần mềm.

* IE0: Cờ ngắt ngoài 0. Được đặt bởi phần cứng khi sườn xung của ngắt ngoài 0 được phát hiện. Được xoá bởi phần cứng khi ngắt được xử lý.

* IT0: Bit điều khiển ngắt 0 để tạo ra ngắt ngoài. Được đặt/xoá bởi phần mềm.

2.4.14. Thanh ghi TMOD: Thanh ghi điều khiển kiểu Timer/Counter

GATE	C/(/T)	M1	M0	GATE	C/(/T)	M1	M0
Dành cho Timer 1				Dành cho Timer 0			

* GATE: Khi GATE=1 và TRx =1, bộ TIMER/COUTERx hoạt động chỉ khi chân INTx ở mức cao. Khi GATE=0, bộ TIMER/COUNTERx sẽ hoạt động chỉ khi TRx=1

* C/(/T): Bit này cho phép chọn chức năng là Timer hay Counter.

- Bit này =0 thì thực hiện chức năng Timer
- Bit này =1 thì thực hiện chức năng Counter

* M0, M1: Bit chọn Mode, để xác định trạng thái và kiểu Timer/Counter:

- M1=0, M0=0: Chọn kiểu bộ Timer 13 bit.Trong đó THx dài 8 bit, TLx dài 5 bit. -
- M1=0, M0=1: Chọn kiểu bộ Timer 16 bit. THx và TLx dài 16 bit được ghép tầng.
- M1=1, M0=0: 8 bit Auto reload. Các thanh ghi tự động nạp lại mỗi khi bị tràn. Khi bộ Timer bị tràn, THx dài 8 bit được giữ nguyên giá trị, còn giá trị nạp lại được đưa vào TLx.
- M1=1, M0=1: Kiểu phân chia bộ Timer. TL0 là 1 bộ Timer/Counter 8 bit, được điều khiển bằng các bit điều khiển bộ Timer 0, Còn TH0 chỉ là bộ Timer 8 bit, được điều khiển bằng các bit điều khiển Timer 1.
- M1=1, M0=1: Timer/Counter 1 Stopped

2.4.15. Thanh ghi SCON:

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SCON là thanh ghi trạng thái và điều khiển cổng nối tiếp. Nó không những chứa các bit chọn chế độ, mà còn chứa bit dữ liệu thứ 9 dành cho việc truyền và nhận tin (TB8 và RB8) và chứa các bit ngắt cổng nối tiếp.

* SM0, SM1: Là các bit cho phép chọn chế độ cho cổng truyền nối tiếp.

SM0	**SM1**	**Mode**	**Đặc điểm**	**Tốc độ Baud**
0	0	0	Thanh ghi dịch	$F_{osc}/12$
0	1	1	8 bit UART	Có thể thay đổi (được đặt bởi bộ Timer)
1	0	2	9 bit UART	$F_{osc}/64$ hoặc $F_{osc}/32$
1	1	3	9 bit UART	Có thể thay đổi (được đặt bởi bộ Timer)

***Bảng 2.4* : Chọn Mode trong SCON**

* SM2: Cho phép truyền tin đa xử lý, thể hiện ở Mode 2 và 3. ở chế độ 2 hoặc 3, nếu đặt SM2 = 1 thì RI sẽ không được kích hoạt nếu bit dữ liệu thứ 9 (RB8) nhận được giá trị bằng 0. ở Mode 1, nếu SM2=1 thì RI sẽ không được kích hoạt nếu bit dừng có hiệu lực đã không được nhận. ở chế độ 0, SM2 nên bằng 0

* REN: Cho phép nhận nối tiếp. Được đặt hoặc xoá bởi phần mềm để cho phép hoặc không cho phép nhận.

* TB8: Là bit dữ liệu thứ 9 mà sẽ được truyền ở Mode 2 và 3. Được đặt hoặc xoá bởi phần mềm.

* RB8: Là bit dữ liệu thứ 9 đã được nhận ở Mode 2 và 3. Ở Mode 1, nếu SM2=0 thì RB8 là bit dừng đã được nhận. Ở Mode 0, RB8 không được sử dụng.

* TI: Cờ ngắt truyền. Được đặt bởi phần cứng tại cuối thời điểm của bit thứ 8 trong Mode 0, hoặc đầu thời điểm của bit dừng trong các Mode khác. Ở bất kỳ quá trình truyền nối tiếp nào, nó cũng phải được xoá bằng phần mềm.

* RI: Cờ ngắt nhận. Được đặt bởi phần cứng tại cuối thời điểm của bit thứ 8 trong Mode 0, hoặc ở giữa thời điểm của bit dừng trong các Mode khác. Ở bất kỳ quá trình nhận nối tiếp nào (trừ trường hợp ngoại lệ, xem SM2), nó cũng phải được xoá bằng phần mềm.

2.5. Các cổng vào/ra của 80C51/89C51.

Vi điều khiển 8051/8951 có 4 cổng, mỗi cổng có 8 bit để thực hiện việc xuất /nhập dữ liệu. Bốn cổng này sẽ cho phép người lập trình truy xuất dữ liệu dưới dạng cả byte hoặc truy xuất từng bít riêng rẽ, khi truy xuất cả byte thì nó được ký hiệu là P_0 , P_1, P_2 và P_3. Một chú ý là khi khởi động lại bộ VĐK (Reset) thì giá trị của các cổng đều ở mức lôgic 1.

Cổng P_0 có 8 chân và số thứ tự chân từ 32 đến chân 39(chân 39~$P_{0.0}$ và chân 32~$P_{0.7}$)

Cổng P_1 có 8 chân và số thứ tự chân từ 1 đến chân 8 (chân 1~$P_{1.0}$ và chân 8 ~$P_{1.7}$)

Cổng P_2 có 8 chân và số thứ tự chân từ 21 đến chân 28(chân 21~$P_{2.0}$ và chân 28~$P_{2.7}$)

Cổng P_3 có 8 chân và số thứ tự chân từ 10 đến chân 17(chân 10~$P_{3.0}$ và chân 17~$P_{3.7}$)

Bình thường thì P_0 được dùng làm đầu ra, khi sử dụng P_0 vừa làm đầu ra vừa làm đầu vào thì cần phải sử dụng điện trở kéo lên vì riêng P_0 được thiết kế kiểu cực máng hở.

2.6. Khối tạo thời gian và bộ đếm.

2.6.1. Giới thiệu các bộ Timer/Counter trong 80C51/89C51:

Hệ vi xử lý on-chip AT89C51 có 2 thanh ghi Timer/Counter dài 16 bit, đó là: Timer 0 và Timer 1. Trong On-chip AT89C52, ngoài Timer 0 và Timer 1 nó còn có thêm bộ Timer 2. Cả 3 bộ Timer này đều có thể được điều khiển để thực hiện chức năng thời gian hay bộ đếm, thông qua thanh ghi TMOD.

Khi thanh ghi Timer/Counter làm việc ở kiểu Timer, thì sau mỗi chu kỳ máy nội dung trong thanh ghi được gia tăng thêm 1 đơn vị. Vì vậy thanh ghi này đếm số chu kỳ máy. Một chu kỳ máy có 12 chu kỳ dao động, do đó tốc độ đếm của thanh ghi là 1/12 tần số đđ.

Khi thanh ghi Timer/Counter làm việc ở kiểu Counter, xung nhịp bên ngoài được đưa vào để đếm ở T0 hoặc T1. Nội dung thanh ghi được tăng lên khi có sự chuyển trạng thái từ 1 về 0 tại chân đầu vào ngoài T0 hoặc T1. Xung nhịp ở các đầu vào ngoài được lấy mẫu tại thời điểm S5P2 của mỗi chu kỳ máy. Khi quá trình lấy mẫu phát hiện ra mức cao ở 1 chu kỳ và mức thấp ở chu kỳ tiếp theo, thì bộ đếm được tăng lên. Giá trị mới của bộ đếm xuất hiện trong thanh ghi tại thời điểm S3P1 của chu kỳ máy sau khi sự chuyển trạng thái đã được phát hiện.

Vì vậy để nội dung của thanh ghi tăng lên 1 đơn vị phải mất 2 chu kỳ máy, nên tốc độ đếm tối đa là 1/24 tần số bộ dao động. Không có sự giới hạn số vòng thực hiện của tín hiệu ở đầu vào ngoài, nhưng nó sẽ giữ ít nhất 1 chu kỳ máy đầy đủ để đảm bảo chắc chắn rằng một mức đã cho được lấy mẫu ít nhất 1 lần nữa trước khi nó thay đổi.

Do xung nhịp bên ngoài có tần số bất kỳ nên các bộ Timer (0 và 1) có 4 chế độ làm việc khác nhau để lựu chọn: (13 bit Timer, 16 bit Timer, 8 bit auto-reload, split Timer).

2.6.2. Chế độ hoạt động của các bộ Timer/Counter

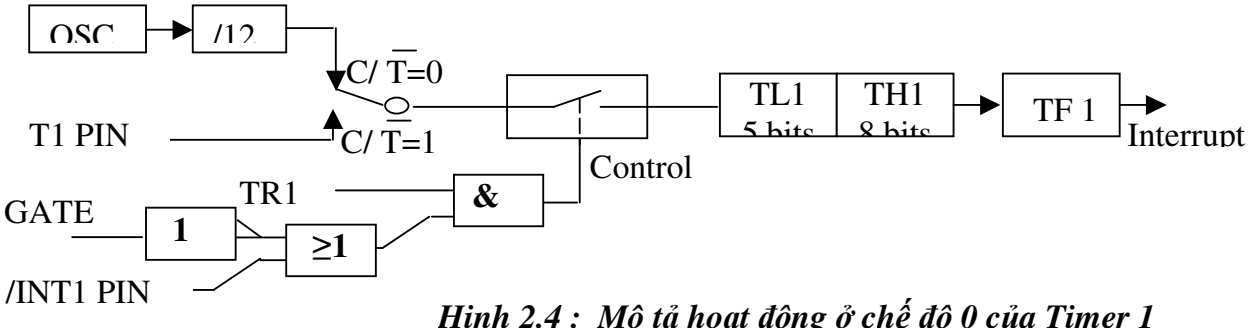
M1	M0	Chế độ đếm
0	0	0_đếm 13 bit
0	1	1_đếm 16 bit
1	0	2_đếm 8 bit
1	1	3_đếm 8 bit

Trong AT89C51 và AT89C52 đều có các bộ Timer 0 và 1, chức năng Timer hay Counter sẽ được lựa chọn bởi các bit điều khiển C/(/T) trong thanh ghi TMOD. Các bộ Timer/Counter này có 4 chế độ hoạt động và nó được lựa chọn bởi cặp bit (M0, M1) trong thanh ghi TMOD. Chế độ 0, chế độ 1 và chế độ 2 hoạt động giống nhau cho các chức năng Timer/Counter, nhưng chế độ 3 thì hơi khác so với 3 chế độ trên và còn gọi là chế độ đếm 8 bít chia sẻ. Bốn chế độ hoạt động của Timer/Counter được mô tả như sau:

❖ ***Chế độ 0***: Cả 2 bộ Timer 0 và 1 ở chế độ 0 có cấu hình như một thanh ghi 13 bit, bao gồm 8 bit của thanh ghi THx và 5 bit thấp của TLx. 3 bit cao của TLx không xác định chắc chắn, nên được làm ngơ. Khi thanh ghi được xoá về 0, thì cờ ngắt thời gian TFx được thiết lập. Bộ Timer/Counter hoạt động khi bit điều khiển TRx được thiết lập (TRx=1) và, hoặc Gate trong TMOD bằng 0, hoặc /INTx=1. Nếu đặt GATE=1 thì cho phép điều khiển Timer/ Counter bằng đường vào ngoài /INTx, để dễ dàng xác định độ rộng xung.

Khi hoạt động ở chức năng thời gian thì bit C/(/T)=0, do vậy xung nhịp từ bộ dao động nội, qua bộ chia tần cho ra tần số $f=f_{osc}/12$ được đưa vào để đếm trong

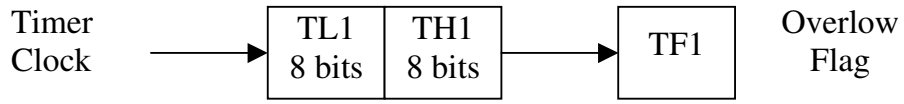
Ta có thể phân tích hoạt động của các Timer ở ***chế độ 0*** như hình vẽ sau :



Hình 2.4 : Mô tả hoạt động ở chế độ 0 của Timer 1

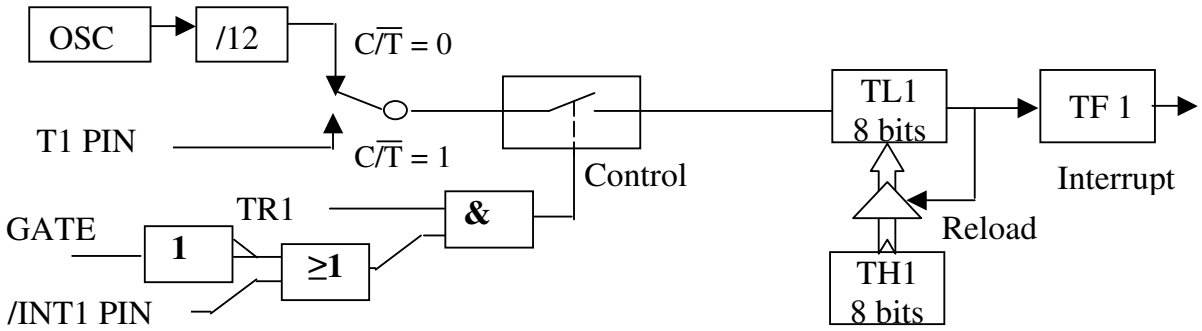
Khi các thanh ghi Timer/Counter hoạt động ở chức năng bộ đếm thì bit C/(/T)=1, lúc đó xung nhịp ngoài đưa vào sẽ được đếm.

❖ ***Chế độ 1***: hoạt động tương tự như chế độ 0, chỉ khác là thanh ghi Timer/Counter được sử dụng cả 16 bit. Xung nhịp được dùng kết hợp với các thanh ghi thời gian byte thấp và byte cao (TH1 và TL1). Khi xung Clock được nhận, bộ Timer sẽ đếm tăng lên: 0000h, 0001h, 0002, Khi hiện tượng tràn xẩy ra, cờ tràn sẽ chuyển FFFFh về 0000h, và bộ Timer tiếp tục đếm. Cờ tràn của Timer 1 là bit TF1 ở trong TCON, nó được đọc hoặc ghi bởi phần mềm, xem hình 2.5 (Timer/Counter 1 Mode 1: 16 bit Counter).



Hình 2.5: Mô tả hoạt động ở chế độ 1 của Timer 1

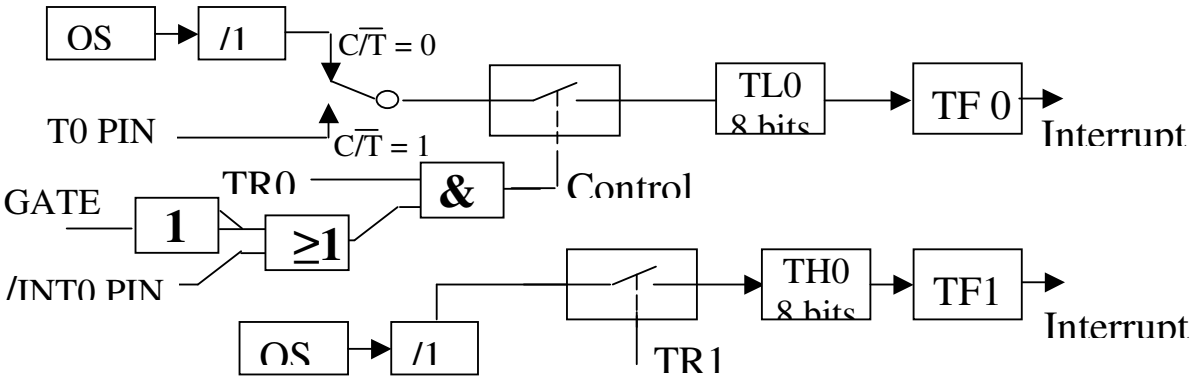
❖ ***Chế độ 2***: Chế độ 2 còn gọi là chế độ đếm 8 bít và tự động lạp lại giá trị đếm ban đầu. Ở chế độ này của thanh ghi Timer cũng hoạt động tương tự như 2 chế độ trên nhưng nó được tổ chức như bộ đếm 8 bit sử dụng (TL1). Khi xẩy ra hiện tượng tràn ở TL1, không chỉ thiết lập bit TF1 mà còn tự động nạp lại cho TL1 bằng nội dung của TH1, đã được thiết lập bởi phần mềm. Quá trình nạp lại cho phép nội dung của TH1 không bị thay đổi. Chế độ 2 của Timer/Counter 0 cũng tương tự như Timer/Counter 1.



Hình 2.6: Mô tả hoạt động ở chế hế đô 2 của Timer 1

❖ ***Chế độ 3***:Ở chế độ này, chức năng Timer/Counter 0 và chức năng Timer/Counter 1 khác nhau. Bộ Timer 1 ở chế độ 3 chỉ chứa chức năng đếm của nó, kết quả giống khi đặt TR1=0. Bộ Timer 0 ở chế độ 3 thiết lập TH0, TL0 như là 2 bộ đếm riêng biệt. Mạch Logic đối với chế độ 3 của Timer 0 thể hiện ở hình 2.7. Bộ đếm TL0 được điều khiển bởi các bit: C/(/T), GATE, TR0, /INT0 và khi đếm tràn nó thiết lập cờ ngắt TF0. Bộ đếm TH0 chỉ được điều khiển bởi bit TR1, và khi đếm tràn nó thiết lập cờ ngắt TF1. Vậy, TH0 điều khiển ngắt Timer/Counter 1.

Chế độ 3 thường được dùng khi yêu cầu cần có bộ thời gian hoặc bộ đếm ngoài 8 bit. Đối với Timer 0 ở chế độ 3, AT89C51 có thể có 3 bộ Timer/Counter, còn AT89C52 có thể có 4 bộ. Khi Timer 0 hoạt động ở chế độ 3, thì Timer 1 có thể được bật hoặc tắt bằng chuyển mạch ngoài. Ở chế độ này, Timer 1 có thể được sử dụng bởi cổng nối tiếp như một bộ tạo tốc độ Baud, hoặc trong bất kỳ ứng dụng nào mà không yêu cầu một ngắt.



Hình 2.7: Chế độ 3 của Timer 0

2.7 Tổ chức không gian bộ nhớ của 80C51.

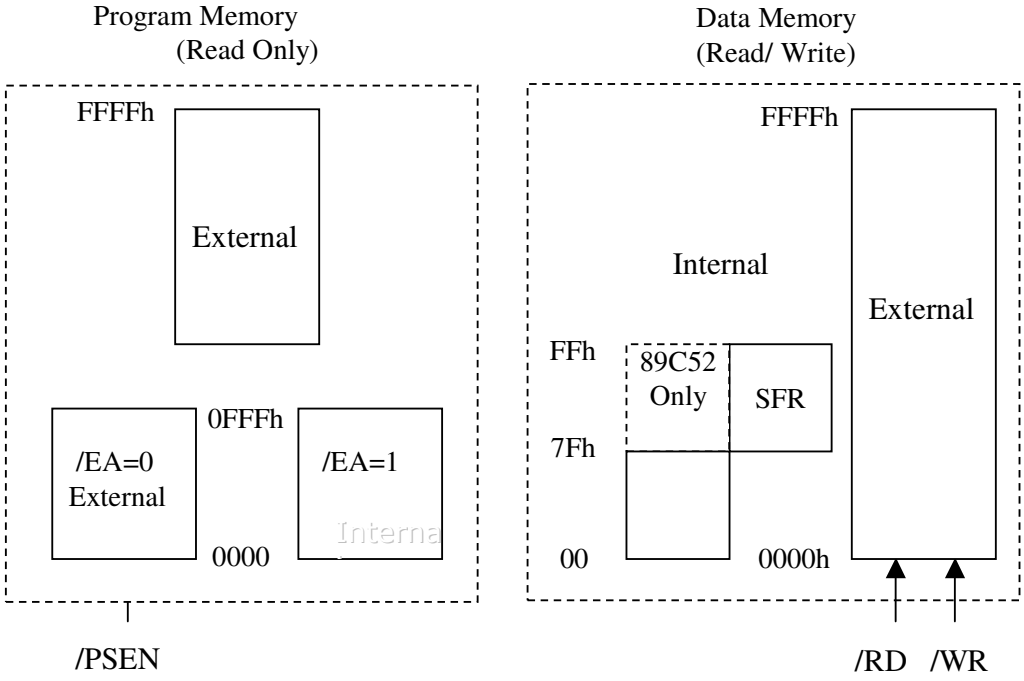
2.7.1 Sơ đồ khối.

Tất cả các bộ Flash Microcontrollers của hãng Atmel đều tổ chức các vùng địa chỉ tách biệt đối với bộ nhớ chương trình và bộ nhớ dữ liệu, được mô tả ở hình 7.8. Các vùng nhớ chương trình và dữ liệu tách biệt cho phép lưu trữ và truy xuất dữ liệu với tốc độ cao ở vùng địa chỉ 8 bít. Tuy nhiên, địa chỉ bộ nhớ dữ liệu 16 bit cũng có thể được tạo ra thông qua thanh ghi con trỏ dữ liệu (DPTR).

Bộ nhớ chương trình có thể chỉ được đọc. Chúng có thể là bộ nhớ chương trình 4 Kbyte ROM trong có khả năng định địa chỉ trực tiếp hoặc 64 Kbyte bộ nhớ chương trình ngoài. Khi truy xuất bộ nhớ ngoài thì cần xác định trạng thái phù hợp cho chân /PSEN.

Bộ nhớ dữ liệu trong có dung lượng là 128 Byte nằm ở vùng địa chỉ riêng biệt so với bộ nhớ chương trình, tuy nhiên 64 Kbyte bộ nhớ ngoài cũng có thể được kết nối khi cần thiết . Để đọc dữ liệu ở bộ nhớ RAM ngoài ngoài thì CPU cần tạo ra tín hiệu đọc và ghi (/RD, /WR) cho phù hợp để truy cập bộ nhớ dữ liệu ngoài.

Bộ nhớ chương trình ngoài và bộ nhớ dữ liệu ngoài có thể được kết hợp bởi các tín hiệu /RD và /PSEN để đưa vào 1 cổng AND và sử dụng đầu ra của cổng này để đọc nội dung từ bộ nhớ dữ liệu/chương trình ngoài.



Hình 2.8. Cấu trúc bộ nhớ của AT89C51

2.7.2. Bộ nhớ chương trình và bộ nhớ dữ liệu nội trú.

2.7.2.1. Bộ nhớ chương trình nội trú:

Bộ nhớ chương trình của AT89C51 được tổ chức như thể hiện ở hình trên. Không gian nhớ cực đại của bộ nhớ này chiếm 64 Kbyte, được định địa chỉ từ 0000h đến FFFFh, trong đó có 4 Kbyte Flash nội trú bên trong nó và được định địa chỉ từ 0000h đến 0FFFh. Do đó có thể mở rộng thêm 60 Kbyte bộ nhớ chương trình bên ngoài, được định địa chỉ từ 1000h đến

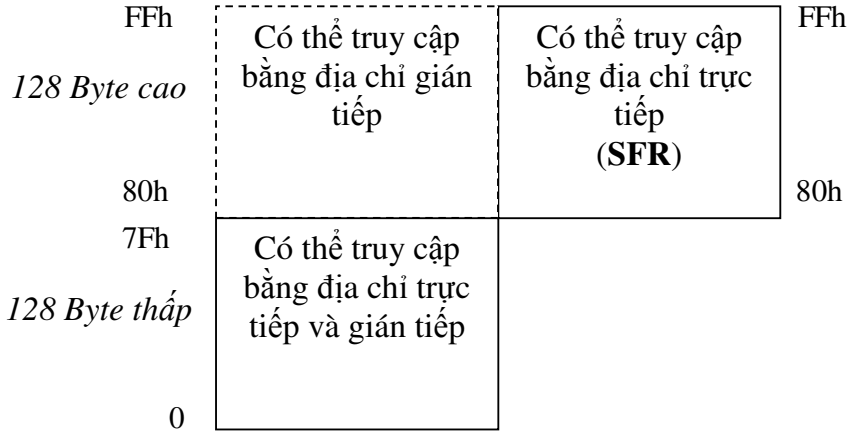
FFFFh. Tuy nhiên bộ VĐK này cũng có thể sử dung toàn bộ bộ nhớ chương trình ngoài bao gồm 64 Kbyte được định địa chỉ từ 0000h đến FFFFh.

Cũng từ hình trên ta thấy, thông qua việc chọn mức logic cho bit /EA có thể lựa chọn để truy cập bộ nhớ chương trình nội trú (4Kb), bộ nhớ chương trình mở rộng ngoại trú (60Kb), hoặc toàn bộ bộ nhớ chương trình ngoại trú bên ngoài On-chip (64Kb). Cụ thể, khi /EA = 1 thì bộ VĐK sử dụng cả bộ nhớ chương trình nội trú và ngoại trú. Ngược lại, khi /EA = 0 thì bộ VĐK chỉ sử dụng bộ nhớ chương trình ngoại trú.

Mỗi khi được Reset, bộ VĐK sẽ truy cập bộ nhớ chương trình tại địa chỉ khởi đầu là 0000h, sau đó nếu cơ chế ngắt được sử dụng thì nó sẽ truy cập tới địa chỉ quy định trong bảng vecter ngắt.

Khi truy cập bộ nhớ chương trình, bộ VĐK sử dụng xung chọn /PSEN để điều khiển. Nếu on-chip làm việc với bộ nhớ chương trình nội trú thì chân phát ra xung chọn /PSEN không sử dụng. Nếu bộ VĐK làm việc với bộ nhớ chương trình ngoại trú thì chân phát ra xung chọn /PSEN được sử dụng. Khi đó nếu /PSEN = 0 thì cho phép bộ VĐK đọc bộ nhớ chương trình ngoài, ngược lại nếu /PSEN = 1 thì bộ VĐK chỉ làm việc với bộ nhớ chương trình nội trú.

2.7.2.2. *Bộ nhớ dữ liệu nội trú:*



Hình 2.9. Bộ nhớ dữ liệu trong

AT89C51 có bộ nhớ dữ liệu chiếm một khoang không gian bộ nhớ độc lập với bộ nhớ chương trình. Dung lượng của RAM nội trú ở họ VĐK này là 128 Byte, được định địa chỉ từ 00h đến 7Fh. Phạm vi địa chỉ từ 80h đến FFh dành cho SFR. Tuy nhiên bộ VĐK cũng có thể làm việc với RAM ngoại trú có dung lượng cực đại là 64 Kbyte được định địa chỉ từ 0000h đến FFFFh.

❖ *Vùng nhớ 128 Byte thấp*

Vùng nhớ 128 Byte thấp được định địa chỉ từ 00h đến 7Fh, được chia thành 3 vùng con như thể hiện ở hình 7.10.

- Vùng thứ nhất có độ lớn 32 Byte được định địa chỉ từ 00h đến 1Fh bao gồm 4 băng thanh ghi (băng 0...băng 3), mỗi băng có 8 thanh ghi 8 bit. Các thanh ghi trong mỗi băng có tên gọi từ R0 đến R7. Vùng RAM này được truy cập bằng địa chỉ trực tiếp mức Byte, và quá

trình chọn để sử dung băng thanh ghi nào là tùy thuộc vào việc lựa chon giá trị cho RS1 và RS0 trong PSW.

- Vùng thứ 2 có độ lớn 16 Byte được định địa chỉ từ 20h đến 2Fh, cho phép truy cập trực tiếp bằng địa chỉ mức bit. Bộ VĐK cung cấp các lệnh có khả năng truy cập tới vùng nhớ 128 bit này (nếu truy cập ở dạng mức bit thì vùng này có địa chỉ được định từ 00h đến 7Fh) ở mức bit. Ở vùng nhớ này, địa chỉ được truy xuất dưới dạng Byte hay Bit tuỳ vào lệnh cụ thể. Chẳng hạn, để đặt bit tại địa chỉ 5Fh có mức logic 1, ta thực hiện lệnh: ***SETB 5Fh*** . Sau khi thực hiện lệnh này, mặc dầu 5Fh là địa chỉ bit cao nhất trong Byte có địa chỉ 2Bh, nhưng nó không làm ảnh hưởng tới các bit khác trong Byte này. Trong khi đó, ở các bộ VXL để thực hiện chức năng như trên cần dùng những lệnh sau:

```
MOV     A,2Bh
ORL     A,#10000000b
MOV     2Bh,A
```

Đây là ưu điểm rõ nét của các bộ VĐK khi thực hiện việc truy xuất các bit riêng rẽ thông qua phần mềm. Các bit có thể được đặt, xoá, hay thực hiện chức năng AND, OR...chỉ thông qua 1 lệnh. Ngoài ra các cổng xuất/nhập cũng có thể được định địa chỉ dạng bit, điều này làm đơn giản việc giao tiếp bằng phần mềm với các thiết bị xuất/nhập đơn bit.

- Vùng nhớ còn lại gồm 80 Byte có địa chỉ từ 30h đến 7Fh được dành riêng cho người sử dụng để lưu trữ dữ liệu. Đây có thể xem là vùng RAM đa mục đích. Có thể truy cập vùng nhớ này bằng địa chỉ trực tiếp hoặc gián tiếp thông qua các thanh ghi (R0 hoặc R1) ở dạng mức Byte.

❖ *Vùng nhớ 128 Byte cao*

Vùng này được định địa chỉ từ 80h đến FFh dùng để chứa địa chỉ của các thanh ghi có chức năng đặc biệt và được truy cập bằng địa chỉ trực tiếp.

❖ *Các lệnh truy cập bộ nhớ dữ liệu nội trú*

- MOV A, <src>: Chuyển dữ liệu từ toán hạng nguồn (các ô nhớ, thanh ghi có địa chỉ trực tiếp hoặc gián tiếp trong on chip, các giá trị trực hằng chứa trong câu lệnh) vào thanh ghi tích luỹ.

- MOV <dest>, <src>: Chuyển dữ liệu từ toán hạng nguồn vào toán hạng đích (các ô nhớ, thanh ghi có địa chỉ trực tiếp hoặc gián tiếp trong on chip).

- MOV <dest>, A : Chuyển dữ liệu từ A vào toán hạng đích.

- MOV DPTR, #data16: Chuyển giá trị hằng 16 bit vào thanh ghi con trỏ dữ liệu.

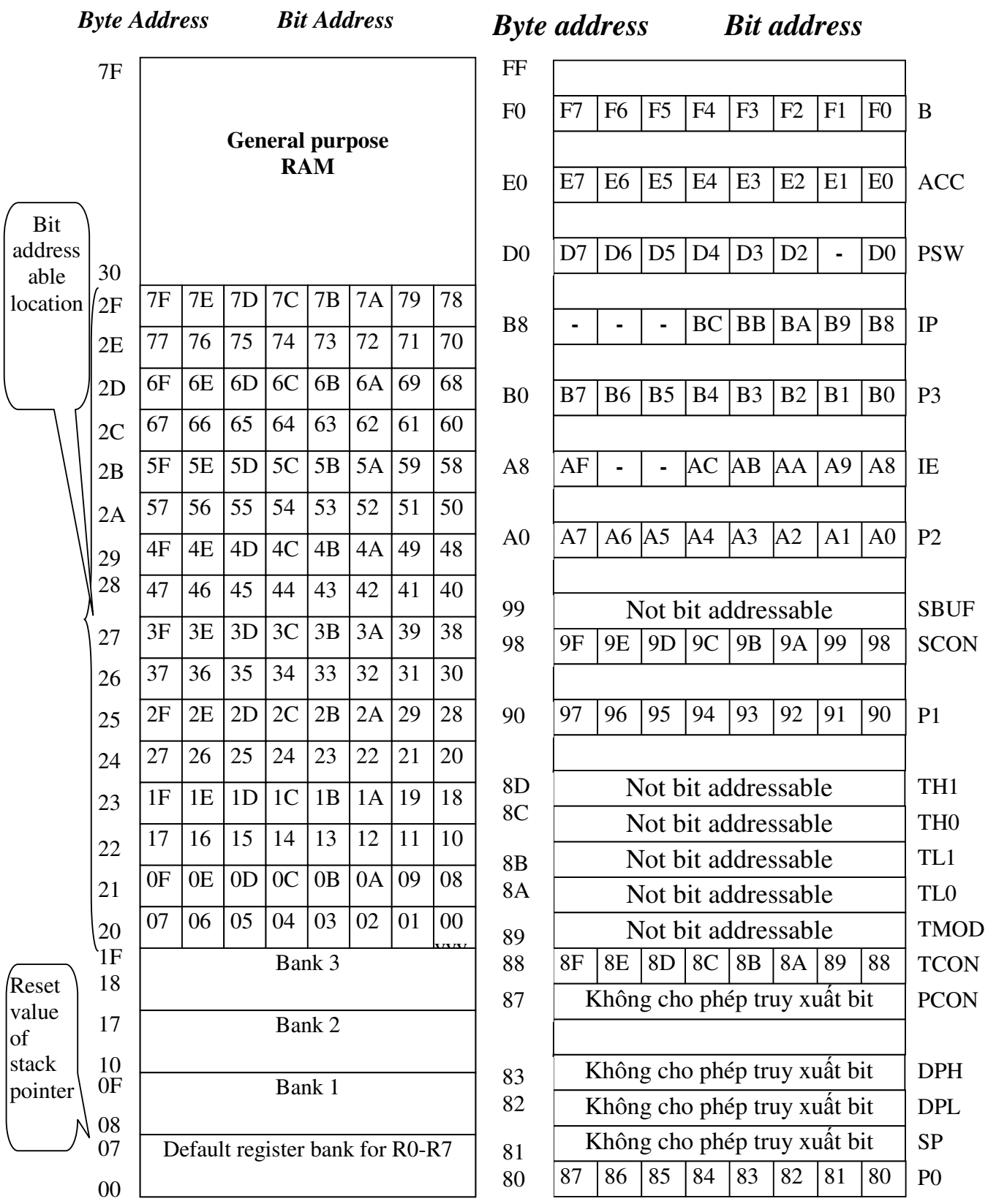
- PUSH <src>: Chuyển giá trị từ toán hạng nguồn vào ngăn xếp.

- POP <dest>: Chuyển giá trị từ ngăn xếp vào toán hạng đích.

- XCH A, <byte>: Chuyển đổi dữ liệu giữa toán hạng nguồn dạng byte vớiA.

- XCHD A, @Ri: Chuyển đổi nửa thấp của A với nội dung trong RAM tại địa chỉ là nội dung của Ri.

Hình 7. 10 sẽ minh họa bản đồ bộ nhớ RAM trong của 80C51/89C51 (Trang bên)



Hình 2.10. 128 Byte thấp của RAM nội và 128 Byte cao của RAM nội

2.7.3. Bộ nhớ chương trình và bộ nhớ dữ liệu ngoại trú.

Để tăng khả năng ứng dụng trong các lĩnh vực điều khiển, đo lường… Bộ VĐK cho phép mở rộng không gian nhớ RAM ngoài đến 64 Kbyte và ROM ngoài đến 64 Kbyte khi cần thiết. Các IC giao tiếp ngoại vi cũng có thể được thêm vào để mở rộng khả năng xuất/nhập và chúng trở thành 1 phần của không gian nhớ dữ liệu ngoài.

Khi bộ nhớ ngoài được sử dụng, cổng P0 không còn đảm nhận chức năng xuất/nhập nữa, mà nó trở thành kênh địa chỉ (A0…A7) và kênh dữ liệu đa hợp (D0…D7). Ngõ ra ALE chốt byte thấp của địa chỉ ở thời điểm bắt đầu của mỗi 1 chu kỳ bộ nhớ ngoài. Cổng P2 thường được dùng làm byte cao của kênh địa chỉ.

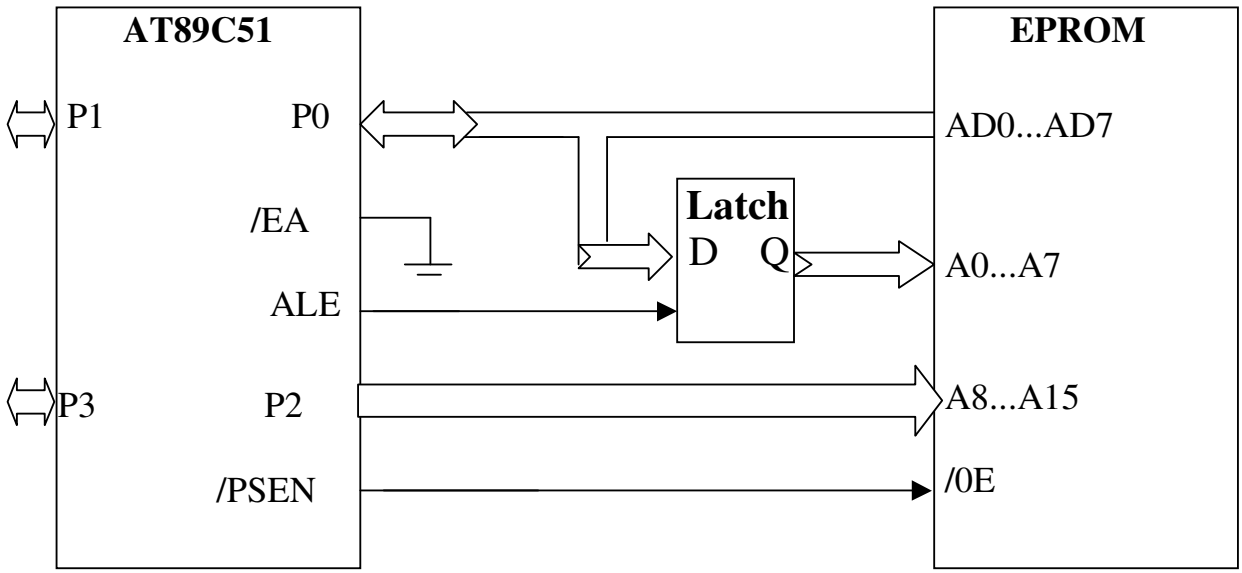
Hoạt động của các bộ nhớ ngoài thường được thực hiện theo kiểu sắp xếp đa hợp, nghĩa là: trong nửa chu kỳ đầu của chu kỳ bộ nhớ, byte thấp của địa chỉ được cung cấp bởi cổng P0 và được chốt nhờ tín hiệu ALE. Mạch chốt giữ cho byte thấp của địa chỉ ổn định trong cả chu kỳ bộ nhớ. Trong nửa chu kỳ sau của bộ nhớ, cổng P0 được sử dụng làm kênh dữ liệu, lúc này dữ liệu có thể được đọc hoặc ghi.

2.7.3.1. Bộ nhớ chương trình ngoại trú:

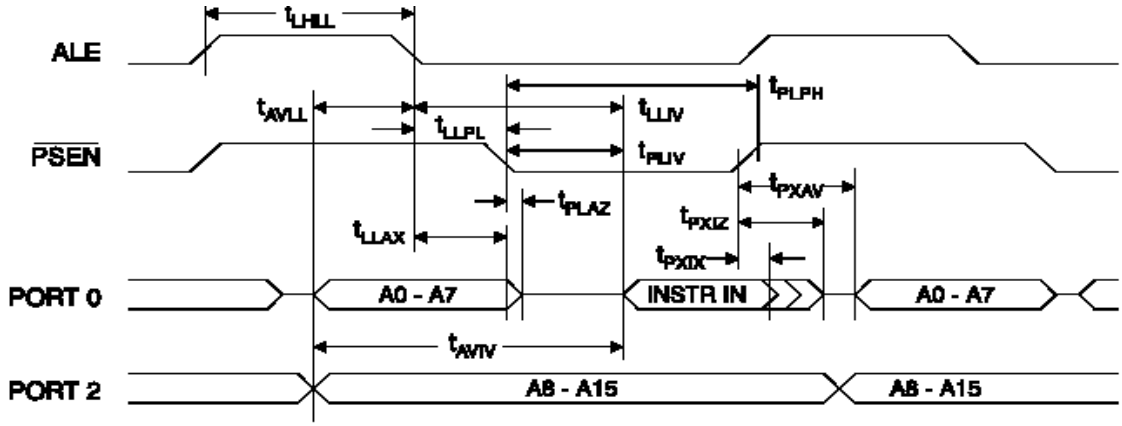
Để tăng khả năng ứng dụng trong các lĩnh vực điều khiển, đo lường… Bộ VĐK cho phép mở rộng không gian nhớ RAM ngoài đến 64 Kbyte và ROM ngoài đến 64 Kbyte khi cần thiết. Các IC giao tiếp ngoại vi cũng có thể được thêm vào để mở rộng khả năng xuất/nhập và chúng trở thành 1 phần của không gian nhớ dữ liệu ngoài.

Khi bộ nhớ ngoài được sử dụng, cổng P0 không còn đảm nhận chức năng xuất/nhập nữa, mà nó trở thành kênh địa chỉ (A0…A7) và kênh dữ liệu đa hợp (D0…D7). Ngõ ra ALE chốt byte thấp của địa chỉ ở thời điểm bắt đầu của mỗi 1 chu kỳ bộ nhớ ngoài. Cổng P2 thường được dùng làm byte cao của kênh địa chỉ.

Hoạt động của các bộ nhớ ngoài thường được thực hiện theo kiểu sắp xếp đa hợp, nghĩa là: trong nửa chu kỳ đầu của chu kỳ bộ nhớ, byte thấp của địa chỉ được cung cấp bởi cổng P0 và được chốt nhờ tín hiệu ALE. Mạch chốt giữ cho byte thấp của địa chỉ ổn định trong cả chu kỳ bộ nhớ. Trong nửa chu kỳ sau của bộ nhớ, cổng P0 được sử dụng làm kênh dữ liệu, lúc này dữ liệu có thể được đọc hoặc ghi. Quá trình truy xuất bộ nhớ ngoài được minh họa bởi hình vẽ 7.11 (Trang bên):



Hình 2.11. Truy cập bộ nhớ chương trình ngoài



Hình 2.12. Đồ thị thời gian quá trình nhận lệnh từ ROM ngoài

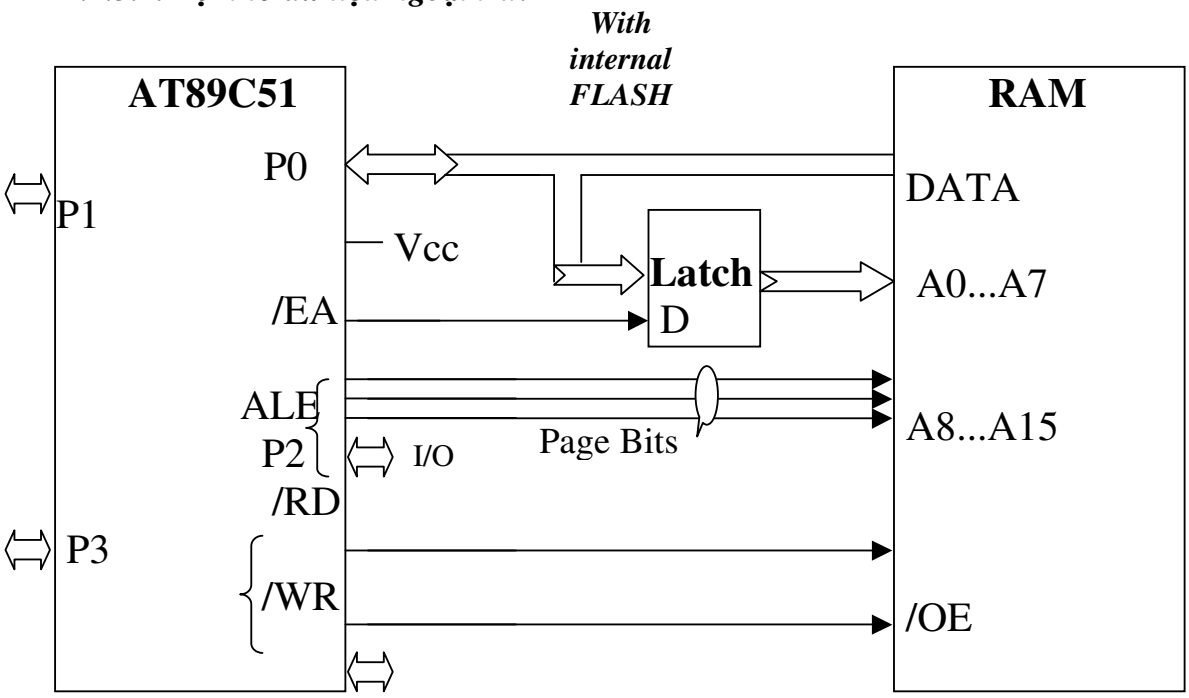
Bộ nhớ chương trình ngoài là bộ nhớ chỉ đọc, được cho phép bởi tín hiệu /PSEN. Khi có một EPROM ngoài được sử dụng, cả P0 và P2 đều không còn là các cổng I/O nữa. Khi bộ VĐK truy cập bộ nhớ chương trình ngoại trú, nó luôn sử dụng kênh địa chỉ 16 bit thông qua P0 và P2.

Một chu kỳ máy của bộ VĐK có 12 chu kỳ dao động. Nếu bộ dao động trên chip có tần số 12 MHz, thì 1 chu kỳ máy dài 1μs. Trong một chu kỳ máy điển hình, ALE có 2 xung và 2 Byte của lệnh được đọc từ bộ nhớ chương trình (nếu lệnh chỉ có 1 byte thì byte thứ 2 được loại bỏ). Khi truy cập bộ nhớ chương trình ngoại trú, bộ VĐK phát ra 2 xung chốt địa chỉ trong mỗi chu kỳ máy. Mỗi xung chốt tồn tại trong 2 chu kỳ dao động từ P2-S1 đến P1-S2, và từ P2-S4 đến P1-S5.

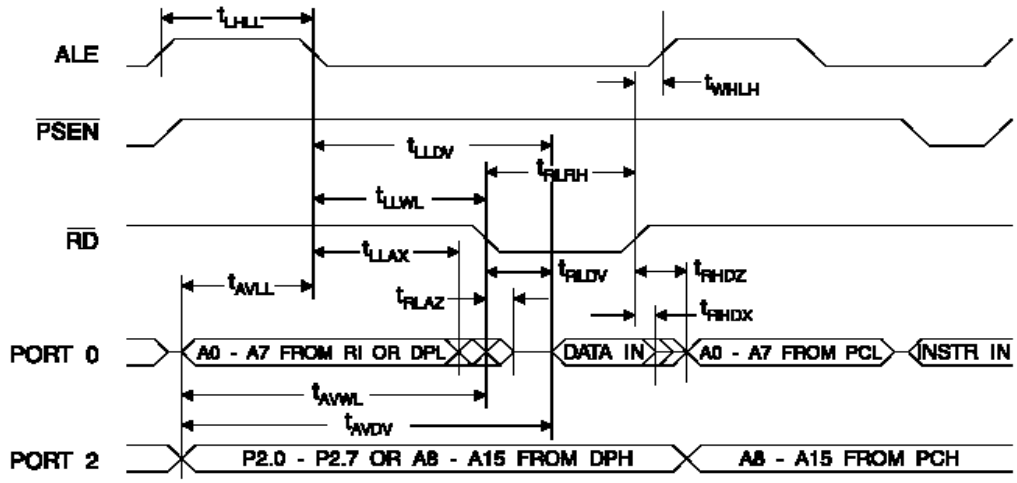
Để địa chỉ hoá bộ nhớ chương trình ngoại trú, byte thấp của địa chỉ (A0…A7) từ bộ đếm chương trình của bộ VĐK được xuất qua cổng P0 tại các trạng thái S2 và S5 của chu kỳ

máy, byte cao của địa chỉ (A8…A15) từ bộ đếm chương trình được xuất qua cổng P2 trong khoảng thời gian của cả chu kỳ máy. Tiếp theo xung chốt, bộ VĐK phát ra xung chọn /PSEN. Mỗi chu kỳ máy của chu kỳ lệnh gồm 2 xung chọn, mỗi xung chọn tồn tại trong 3 chu kỳ dao động từ P1-S3 đến hết P1-S4 và từ P1-S6 đến hết P1-S1 của chu kỳ máy tiếp theo. Trong khoảng thời gian phát xung chọn thì byte mã lệnh được đọc từ bộ nhớ chương trình để nhập vào On chip.

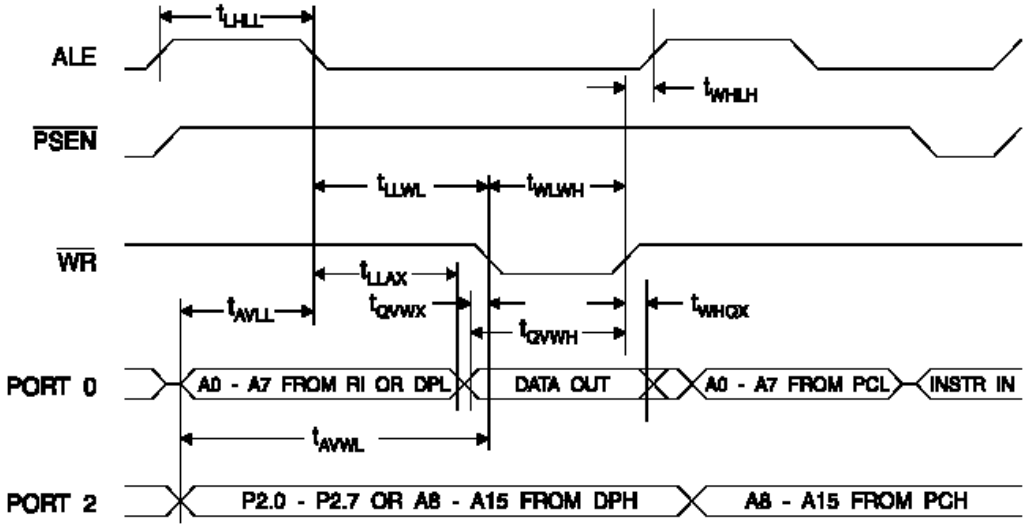
2.7.3.2. Bộ nhớ dữ liệu ngoại trú:



Hình 2. 13. Truy cập bộ nhớ dữ liệu ngoài



Hình 2.14. Đồ thị thời gian chu kỳ đọc dữ liệu từ RAM ngoài



Hình 2.15. Đồ thị thời gian chu kỳ ghi dữ liệu vào RAM ngoài

Bộ nhớ dữ liệu ngoại trú được cho phép bởi các tín hiệu /WR và /RD ở các chân P3.6 và P3.7. VĐK truy cập bộ nhớ dữ liệu ngoài bằng địa chỉ 2 byte (thông qua cổng P0 và P2) hoặc 1 byte (thông qua cổng P0). Lệnh dùng để truy xuất bộ nhớ dữ liệu ngoài là MOVX, sử dụng hoặc DPTR hoặc Ri (R0 và R1) làm thanh ghi chứa địa chỉ.

Trong hình 7.14 ta thấy:

- /EA được nối với +Vcc để cho phép VĐK làm việc với bộ nhớ chương trình nội trú.
- /RD nối với đường cho phép xuất dữ liệu (/OE-Output Data Enable) của RAM.
- /WR nối với đường cho phép ghi dữ liệu (/WE-Write Data Enable) của RAM.

Nguyên lý truy cập bộ nhớ dữ liệu ngoại trú được thể hiện bằng các đồ thị thời gian ở trên. Tuy nhiên, tuỳ thuộc vào nhiệm vụ đọc dữ liệu từ bộ nhớ hay ghi dữ liệu vào bộ nhớ mà nguyên lý truy cập bộ nhớ dữ liệu là khác nhau.

**** Quá trình đọc dữ liệu từ bộ nhớ ngoại trú:*** Khi truy cập bộ nhớ dữ liệu ngoại trú, bộ VĐK *phát ra 1 xung chốt địa chỉ* (ALE) cho bộ chốt bên ngoài (Latch) trong mỗi chu kỳ máy, tồn tại trong 2 chu kỳ dao động từ P2-S4 đến P1-S5. Để địa chỉ hoá bộ nhớ dữ liệu ngoài, byte thấp của địa chỉ từ thanh ghi con trỏ dữ liệu (DPL) hoặc từ Ri của VĐK được xuất qua cổng P0 trong khoảng các trạng thái S5 của chu kỳ máy trong chu kỳ lệnh. Tiếp theo byte thấp của địa chỉ từ bộ đếm chương trình (PCL) cũng được xuất ra qua cổng P0 đưa tới bộ đếm chương trình để thực hiện lệnh tiếp theo. Byte cao của địa chỉ từ DPTR (DPH) của VĐK được xuất qua cổng P2 trong khoảng thời gian từ S5 đến S4 của chu kỳ máy tiếp theo. Sau đó byte cao của địa chỉ từ PC (PCH) cũng được xuất qua cổng P2 để đưa đến bộ nhớ chương trình. Nếu địa chỉ có độ dài 1 byte thì nó được xuất qua cổng P0 từ DPL hoặc Ri. Tiếp theo xung chốt, VĐK xuất ra tín hiệu điều khiển /RD để cho phép đọc dữ liệu từ bộ nhớ ngoài. Xung /RD tồn tại trong 3 trạng thái của mỗi chu kỳ máy từ P1-S1 đến P2-S3, và trong khoảng thời gian này dữ liệu từ bộ nhớ ngoài được đọc vào VĐK .

**** Quá trình ghi dữ liệu vào bộ nhớ ngoại trú:*** Tương tự như quá trình đọc dữ liệu, nhưng ở đây dùng tín hiệu điều khiển ghi /WR.

**** Các lệnh truy cập bộ nhớ dữ liệu ngoại trú:***

- MOVX A, @Ri: Chuyển (đọc) dữ liệu 8 bit từ ô nhớ của RAM ngoài tại địa chỉ được xác đinh trong thanh ghi của băng thanh ghi hiện hành vào A.

- MOVX @Ri, A: Chuyển (ghi) dữ liệu 8 bit từ A vào ô nhớ của RAM ngoài tại địa chỉ được xác định trong thanh ghi của băng thanh ghi hiện hành.

- MOVX A,@DPTR: Chuyển (đọc) dữ liệu 16 bit từ ô nhớ của RAM ngoài tại địa chỉ được xác đinh trong thanh ghi con trỏ dữ liệu vào A.

- MOVX @DPTR, A: Chuyển (ghi) dữ liệu 16 bit từ A vào ô nhớ của RAM ngoài tại địa chỉ được xác định trong thanh ghi con trỏ dữ liệu.

Ví dụ: MOV R0, #4Fh
MOVX A,@R0

Sẽ chuyển nội dung ở RAM ngoài tại địa chỉ 4Fh vào A.

2.8. Cơ chế ngắt trong On-chip 80C51.

2.8.1. Khái niệm về ngắt (*GV :Lấy ví dụ minh họa*)

2.8.2. Phân loại ngắt trong 8051/8951 (bảng địa chỉ vetor)

Vi điều khiển AT89C51có tất cả 5 nguyên nhân gây ra hiện tượng ngắt quãng của chương trình. Trong 5 nguyên nhân gây ra ngắt gồm có : 2 ngắt ngoài (/INT0 và /INT1), 2 ngắt cờ tràn của khối thời gian (Timer 0, 1), và 1 ngắt cổng truyền tin nối tiếp.

Mỗi nguồn ngắt có thể được kích hoạt hoặc không kích hoạt bằng cách đặt hoặc xoá Bit ở trong IE. IE cũng chứa bit có thể không cho tất cả các ngắt hoạt động EA (Nếu EA=0). Các ngắt ngoài có thể được kích hoạt theo mức hoặc theo sườn xung, tuỳ thuộc vào giá trị của các bit IT0, IT1 trong TCON. Ngắt ngoài có 2 cờ ngắt tương ứng là IE0, IE1 cũng nằm trong thanh ghi TCON. Khi một ngắt được thực hiện thì cờ ngắt tương ứng của nó bị xoá bằng phần cứng. Chương trình con phục vụ ngắt ngoài hoạt động chỉ khi ngắt được kích hoạt theo sườn xung. Nếu ngắt đực kích hoạt theo mức thì nguồn yêu cầu ngắt từ bên ngoài điều khiển cờ ngắt. Khi mỗi ngắt được kích hoạt thì chương trình chính sẽ bị dừng ở địa chỉ đó và nhảy thẳng về địa chỉ vector của ngắt tương ứng, địa chỉ tương ứng của các ngắt được cho trong bảng vector ngắt như sau:

Ngắt	Nguồn ngắt	Địa chỉ Véc tơ
External 0	IE0	0003h
Timer 0	TF0	000Bh
External 1	IE1	0013h
Timer 1	TF1	001Bh
Serial Port	RI hoặc TI	0023h
Timer 2 (AT89C52)	TF2 hoặc EXF2	002Bh
System Reset	RST	0000h

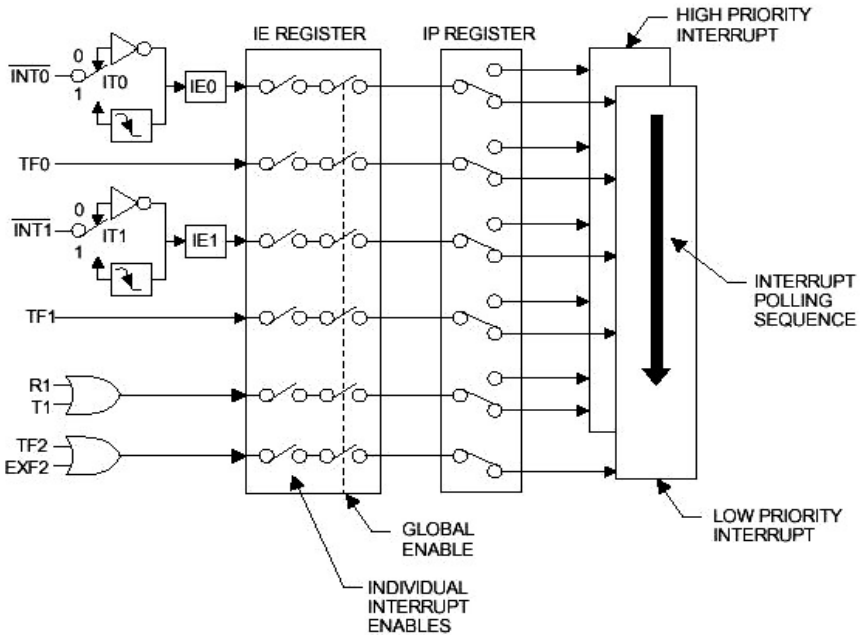
Bảng 2.5. Bảng địa chỉ véc tơ của các ngắt

2.8.3. Nguyên lý điều khiển ngắt

Các cờ ngắt được thiết lập tại thời điểm S5P2 của mỗi chu kỳ máy. Chu kỳ máy tiếp theo sau chu kỳ máy có cờ ngắt được thiết lập, thì chương trình con phục vụ ngắt được thực hiện khi có lệnh gọi LCALL. Lệnh LCALL phát sinh nhưng lại bị cấm hoạt động khi gặp các tình huống sau:

a- Đồng thời có ngắt với mức ưu tiên cao hơn hoặc bằng ngắt đang phục vụ.
(Một ngắt có mức ưu tiên bằng hoặc cao hơn đang sẵn sàng để được phục vụ)
b- Chu kỳ máy hiện hành không phải là chu kỳ máy cuối của lệnh đang thực hiện.
c- Lệnh đang thực hiện là RETI hoặc bất kỳ lệnh nào ghi vào thanh ghi IE hoặc IP.

Figure 20. AT89 Interrupt Control System



Hình 2.16. Hệ thống ngắt của AT89C51

Bất kỳ một trong 3 điều kiện này xuất hiện sẽ cản trở việc tạo ra LCALL đối với chương trình phục vụ ngắt. Điều kiện 2 đảm bảo rằng, lệnh đang thực hiện sẽ được hoàn thành trước khi trỏ tới bất kỳ chương trình phục vụ nào. Điều kiện 3 đảm bảo rằng, nếu lệnh đang thực hiện là RETI hoặc bất kỳ sự truy cập nào vào IE hoặc IP, thì ít nhất một lệnh nữa sẽ được thực hiện trước khi bất kỳ ngắt nào được trỏ tới. Chu trình kiểm tra vòng được lặp lại với mỗi chu trình máy, và các giá trị được kiểm tra là các giá trị mà đã xuất hiện ở thời điểm S5P2 của chu trình máy trước đó. Nếu một chỉ thị ngắt có hiệu lực nhưng không được đáp ứng vì các điều kiện trên và nếu chỉ thị này vẫn chưa có hiệu lực khi điều kiện cản trở được loại bỏ, thì ngắt bị từ chối này sẽ không được phục vụ nữa.

LCALL do phần cứng tạo ra sẽ chuyển nội dung của bộ đếm chương trình vào ngăn xếp (Nhưng không ghi vào PSW) và nạp lại cho PC một địa chỉ phụ thuộc vào nguồn gây ra ngắt đang được phục vụ, như bảng địa chỉ vec tơ ngắt (Bảng 7.5)

Lệnh RETI thông báo cho bộ VXL rằng thủ tục ngắt này đã kết thúc, sau đó lấy ra 2 Byte từ ngăn xếp và nạp lại cho PC để trả lại quyền điều khiển cho chương trình chính.

2.8.4. Trình tự các bước thực hiện chương trình ngắt

Theo đúng trình tự, để sử dụng các ngắt trong Flash Microcontroller, cần thực hiện các bước như sau:

- Đặt bit EA ở trong IE mức logic 1.
- Đặt bit cho phép ngắt tương ứng ở trong IE mức logic 1.
- Bắt đầu chương trình con phục vụ ngắt tại địa chỉ của ngắt tương ứng đó.

(Xem bảng địa chỉ Vector của các nguồn ngắt)

Ngoài ra, đối với các ngắt ngoài, các chân /INT0, /INT1 phải được đặt mức 1. Và tuỳ thuộc vào ngắt được kích hoạt bằng mức hay sườn xung, mà các bit IT0, IT1 ở trong TCON có thể cần phải đặt mức 1.

ITx=0: Kích hoạt bằng mức

ITx=1: Kích hoạt bằng sườn xung.

2.8.5. Mức ngắt ưu tiên trong chíp

Mỗi nguồn ngắt có thể được lập trình riêng cho 1 hoặc 2 mức ưu tiên bằng cách đặt hoặc xoá 1 bit trong IP của SFR. Mỗi ngắt ưu tiên ở mức thấp có thể được ngắt bằng ngắt ưu tiên ở mức cao hơn nhưng không thể ngắt bằng ngắt có mức ưu tiên ở mức thấp hơn được. Một ngắt ưu tiên ở mức cao có thể được ngắt bởi bất kỳ nguồn ngắt nào khác.

Nếu có yêu cầu ngắt của 2 mức ưu tiên cùng nhau (cùng 1 lúc), yêu cầu của mức ưu tiên cao hơn sẽ được phục vụ (Ngắt nào có mức ưu tiên cao hơn sẽ được phục vụ). Nếu các yêu cầu ngắt có cùng mức ưu tiên, thì thứ tự quay vòng bên trong sẽ quyết định ngắt nào được phục vụ.

Thứ tự ưu tiên ngắt từ cao xuống thấp của AT89C51 như sau:

IE0, TF0, IE1, TF1, RI hoặc TI.

2.8.6. Các ngắt ngoài

Vì các chốt ngắt ngoài được tạo mẫu mỗi lần trong mỗi chu trình máy, nên một giá trị cao hoặc thấp của đầu vào sẽ duy trì trong ít nhất là 12 chu kỳ xung nhịp của bộ dao động để đảm bảo tạo mẫu. Nếu ngắt ngoài được kích hoạt bằng sườn xung , thì nguồn ngắt ngoài phải duy trì ở chốt yêu cầu giá trị cao ít nhất 1 chu kỳ máy và sau đó duy trì giá trị thấp ít nhất 1 chu kỳ máy nữa. Việc này được thực hiện để đảm bảo rằng quá trình chuyển tiếp cho thấy chỉ thị yêu cầu ngắt IEx sẽ được xác lập. IEx sẽ tự động được xoá bởi CPU khi thủ tục ngắt đáp ứng được gọi.

Nếu ngắt ngoài được kích hoạt theo mức, thì nguồn ngắt bên ngoài phải duy trì cho yêu cầu này có hiệu lực cho đến khi ngắt đã được yêu cầu thực sự được tạo ra. Sau đó nguồn ngắt ngoài phải huỷ yêu cầu đó trước khi thủ tục phục vụ ngắt hoàn thành, nếu không ngắt khác sẽ được tạo ra.

2.8.7. Vận hành Step

Cấu trúc ngắt AT89C51 cho phép thực hiện các bước đơn với sự tham gia của rất ít phần mềm. Như đã lưu ý trước đây, một yêu cầu ngắt sẽ không được đáp ứng khi một ngắt khác có cùng mức ưu tiên vẫn đang hoạt động, nó cũng không được đáp ứng sau khi có lệnh RETI cho đến khi có ít nhất một lệnh khác đã được thực hiện. Do đó mỗi khi một thủ tục ngắt được đưa vào, thì nó không thể được đưa vào lần nữa cho đến khi ít nhất một lệnh của chương trình ngắt được thực hiện. Một cách để sử dụng đặc điểm này đối với hoạt động theo bước đơn lẻ là lập trình cho 1 trong những ngắt ngoài(chẳng hạn /INT0) được kích hoạt theo mức.

Nếu chân /INT0 được duy trì ở mức thấp, thì CPU sẽ chuyển ngay đến thủ tục ngắt ngoài 0 và dừng ở đó cho tới khi INT0 được nhận xung từ thấp lên cao rồi xuống thấp. Sau đó nó sẽ thực hiện lệnh RETI, trở lại nhiệm vụ chương trình, thực hiện một lệnh, và ngay sau đó nhập lại thủ tục ngfắt ngoài 0 để đợi xung nhịp tiếp theo của P3.2. Mỗi bước của nhiệm vụ chương trình được thực hiện vào mỗi thời điểm chân P3.2 được nhận xung.

2.9. Phương thức truyền tin nối tiếp.

Hệ VXL on-chip này truyền tin nối tiếp bằng cổng RxD và TxD, dữ liệu xuất nhập truyền qua cổng nối tiếp bằng tốc độ Baud và đều qua vùng đệm nối tiếp SBUF. Cổng truyền nối tiếp là cổng truyền tin 2 chiều, nghĩa là nó có thể đồng thời truyền và nhận thông tin cùng 1 lúc. Nó cũng có khả năng vừa thực hiện chức năng nhận vừa thực hiện chức năng đệm, tức là nó có thể nhận byte kế tiếp trước khi byte được nhận trước đó được đọc từ thanh ghi đệm. (Tuy nhiên, nếu byte đầu tiên vẫn chưa được đọc tại thời điểm nhận của byte thứ 2, thì một trong 2 byte này sẽ bị mất). Điều khiển cổng nối tiếp bằng thanh ghi SCON, trạng thái của 2 bit SM0 và SM1 trong thanh ghi này thiết lập nên 4 chế độ hoạt động giao tiếp nối tiếp chuẩn như sau:

❖ ***Chế độ 0***: Dữ liệu nối tiếp vào và ra sẽ thông qua chân RxD. Chân TxD đưa ra xung nhịp đồng hồ. 8 bit dữ liệu được truyền/nhận nối tiếp, với bit LSB được thực hiện đầu tiên. Tốc độ Baud được cố định bằng 1/12 tần số của bộ dao động.

❖ ***Chế độ 1***: 10 bit được truyền (thông qua TxD) hoặc nhận (thông qua RxD), trong đó gồm có: 1 bit khởi động (có giá trị 0), 8 bit dữ liệu (đầu tiên là LSB), và 1 bit dừng (có giá trị là 1). Khi nhận, bit dừng được chuyển vào RB8 của thanh ghi SCON. Tốc độ Baud có thể thay đổi được.

❖ ***Chế độ 2***: 11 bit được truyền (thông qua TxD) hoặc nhận (thông qua RxD) bao gồm: bit khởi động (có giá trị 0), 8 bit dữ liệu (đầu tiên là LSB), một bit dữ liệu thứ 9 có thể lập trình được, và một bit dừng (có giá trị 1). Khi truyền, bit dữ liệu thứ 9 (TB8 ở trong SCON) có thể được gán giá trị 0 hoặc 1. Chẳng hạn như bit chẵn lẻ (P ở trong PSW) có thể được chuyển vào TB8. Khi nhận, bit dữ liệu thứ 9 được chuyển vào RB8 ở thanh ghi SCON, trong khi bit dừng được lọc bỏ. Tốc độ Baud có thể lập trình được bằng 1/32 hoặc 1/64 tần số bộ dao động.

❖ ***Chế độ 3***: 11 bit được truyền (thông qua TxD) hoặc được nhận (thông qua RxD) bao gồm: 1 bit khởi động (có giá trị 0), 8 bit dữ liệu (đầu tiên là LSB), 1 bit dữ liệu thứ 9 có thể lập

trình được, và 1 bit dừng (có giá trị 1). Trên thực tế, chế độ 3 giống chế độ 2 ở mọi góc độ trừ tốc độ Baud. Tốc độ Baud ở chế độ 3 là khả biến và được xác định theo bộ Timer 1.

Trong cả 4 chế độ trên, việc truyền được bắt đầu bởi bất kỳ một lệnh nào mà sử dụng thanh ghi SBUF như là một thanh ghi đích. Việc nhận được bắt đầu ở chế độ 0 khi RI=0 và REN=1. Đối với các chế độ khác, việc nhận được bắt đầu khi bit REN=1.

***2.9.1. Liên lạc đa xử lý** (Multiprocessor Communications):*

Chế độ 2 và 3 có một dự trù (chuẩn bị) đặc biệt cho các liên lạc đa xử lý. Trong các chế độ này 9 bit dữ liệu được sử dụng. Bit thứ 9 sẽ chuyển vào RB8, sau đó là 1 bit dừng. Cổng nối tiếp có thể được lập trình để thoả mãn điều kiện: khi bit dừng đực nhận thì ngắt của cổng nối tiếp được kích hoạt chỉ khi RB8=1. Đặc điểm này có thể thực hiện được bằng cách đặt bit SM2 ở trong SCON.

Ví dụ dưới đây cho thấy, cách thức sử dụng ngắt cổng truyền nối tiếp để tạo liên lạc đa xử lý. Khi bộ xử lý chủ (Master) muốn truyền 1 khối dữ liệu tới một trong những bộ xử lý (Slave) khác, đầu tiên nó gửi đi 1 byte địa chỉ để xác định địa chỉ của bộ xử lý đích (Slave). Một byte địa chỉ khác với một byte dữ liệu ở chỗ: bit thứ 9 bằng 1 ở byte địa chỉ và bằng 0 ở byte dữ liệu. Với SM2=1, không có bộ xử lý (Slave) nào được ngắt bởi 1 byte dữ liệu. Tuy nhiên 1 byte địa chỉ sẽ ngắt tất cả các bộ xử lý (Slave) khác, để cho mỗi bộ xử lý (slave) khác có thể kiểm tra byte nhận được và để xem có phải nó đang được trỏ tới không. Bộ xử lý (slave) nào được trỏ tới sẽ xoá (clear) bit SM2 của nó và chuẩn bị nhận các byte dữ liệu sẽ đưa đến. Các bộ xử lý (Slave) khác nếu không được trỏ tới, thì sẽ thiết lập (set) bit SM2 của chúng và tiếp tục hoạt động của mình mà không cần quan tâm tới dữ liệu trên kênh.

Bit SM2 không có tác dụng ở chế độ 0, nhưng nó có thể được sử dụng để kiểm tra bit dừng trong chế độ 1. Trong quá trình nhận tin ở chế độ 1, nếu SM2=1 thì ngắt nhận tin sẽ không được kích hoạt trừ khi bit dừng được nhận vào.

2.9.2. Các tốc độ Baud:

+ Tốc độ Baud ở chế độ 0 được cố định, và bằng ***Tần số bộ dao động/12***

+ Tốc độ Baud ở chế độ 2 phụ thuộc vào giá trị của bit SMOD trong thanh ghi PCON. Nếu SMOD=0 (giá trị sau khi reset), thì tốc độ Baud =1/64 tần số của bộ dao động. Nếu SMOD=1 thì tốc độ Baud =1/32 tần số của bộ dao động.

$$\textit{Tốc độ Baud chế độ 2} = (2^{SMOD} * \textit{Tần số bộ dao động})/64$$

Trong AT89C51, các tốc độ Baud ở chế độ 1 và 3 do Timer 1 quyết định, Trong AT89C52 tốc độ Baud của các chế độ này có thể được quyết định bởi Timer 1 hoặc Timer 2, hoặc cả hai (một bộ timer xác định tốc độ truyền, bộ kia xác định tốc độ nhận).

2.9.3. Sử dụng Timer 1 để tạo ra các tốc độ Baud :

Khi bộ Timer 1 được dùng để tạo tốc độ Baud, thì các tốc độ Baud ở các chế độ 1 và 3 do tốc độ tràn của timer 1 và giá trị của SMOD quyết định:

$$\textit{Tốc độ Baud ở chế độ 1 và 3} = (2^{SMOD} * (\textit{Tốc độ tràn của timer 1}))/32$$

Ngắt của Timer 1 sẽ mất tác dụng trong ứng dụng này.

Bản thân bộ Timer có thể được thiết lập để thực hiện chức năng thời gian hay bộ đếm ở bất kỳ một trong 3 chế độ hoạt động. Trong hầu hết các kiểu ứng dụng, nó thường được thiết lập để thực hiện chức năng thời gian, hoạt động ở chế độ Auto-reload (nửa byte cao của TMOD = 0010b). Trong trường hợp này, tốc độ baud được tính bằng công thức:

Tốc độ Baud chế độ 1 và 3 = (2^{SMOD}*Tần số bộ dao động)/(32*(12*[256-(TH1)])

Ta có thể nhận được các tốc độ Baud rất thấp với bộ Timer 1 bằng cách làm cho ngắt của timer 1 có tác dụng, và thiết lập Timer 1 để hoạt động như một bộ đếm thời gian 16 bit (Nửa byte cao của TMOD=0001b). Bảng 2.8 liệt kê các tốc độ Baud khác nhau thường được sử dụng và cách chúng có thể nhận được từ Timer 1.

Tốc độ Baud (Hz)	**Tần số d.động (MHz)**	SMODE	**Timer** 1		
			C/(/T)	***Mode***	***Giá trị nạp lại***
Mode 0 Max: 1M	12	x	X	X	**X**
Mode 2 Max: 375K	12	1	X	X	X
Mode 1,3 Max:62,5K	12	1	0	2	FFh
19,2K	11,059	1	0	2	FDh
9,6K	11,059	0	0	2	FDh
4,8K	11,059	0	0	2	FAh
2,4K	11,059	0	0	2	F4h
1,2K	11,059	0	0	2	E8h
137,5	11,966	0	0	2	1Dh
110	6	0	0	2	72h
110	12	0	0	1	FEEBh

Bảng 2.6. Các tốc độ Baud được tạo ra khi sử dụng Timer 1

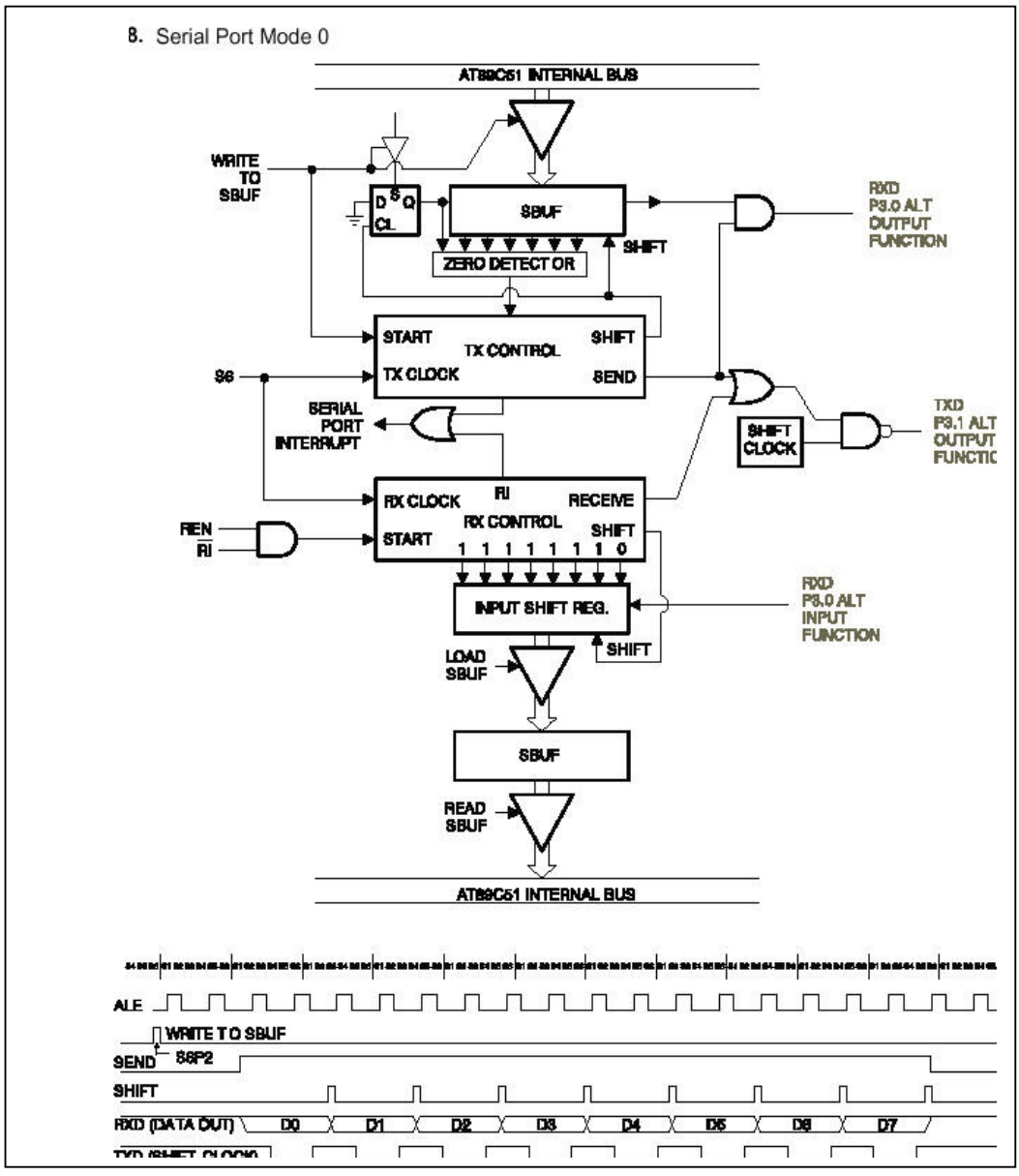
2.9.4. Hoạt động của chế độ 0:

Dữ liệu nối tiếp vào và ra thông qua RxD. TxD cho ra đồng hồ xung nhịp. 8 bit dữ liệu được truyền/nhận (với LSB đầu tiên) được thực hiện ở chế độ này. Tốc độ Baud được cố định bằng 1/12 tần số bộ dao động.

Hình 2.19 (Seriel Port Mode 0) mô tả sơ đồ chức năng của cổng nối tiếp ở chế độ 0 và các mốc thời gian có liên quan. Quá trình truyền được bắt đầu bằng bất kỳ lệnh nào mà sử dung SBUF như là một thanh ghi đích. Tín hiệu "ghi vào SBUF" tại thời điểm S6P2 cũng nạp giá trị 1 vào vị trí thứ 9 của thanh ghi dịch trong quá trình truyền và bật cờ báo cho khối

điều khiển phát (Tx Control) về yêu cầu truyền tin. Thời gian được xác lập bên trong cho 1 chu trình máy đầy đủ sẽ bắt đầu từ thời điểm "ghi vào SBUF" cho tới khi SEND được kích hoạt.

SEND cho phép nội dung của thanh ghi dịch đưa tới đầu ra P3.0 và cho phép tín hiệu SHIFT CLOCK đến đầu ra P3.1. SHIFT CLOCK có giá trị thấp trong các trạng thái S3, S4 và S5 của mỗi chu trình máy, và có giá trị cao trong các trạng thái S6, S1 và S2. Tại thời điểm S6P2 của mỗi chu trình máy khi SEND có mức tích cực, thì nội dung của thanh ghi dịch phát được dịch sang bên phải một bit.



Hình 2.17. Truyền nối tiếp ở chế độ 0

Khi các bit dữ liệu dịch sang bên phải để đi ra ngoài thì các giá trị 0 được gán vào bên trái. Khi bit có trọng số lớn nhất MSB của Byte dữ liệu ở vị trí đầu của thanh ghi dịch, thì giá trị 1 (đã được nạp từ đầu vào vị trí thứ 9) được đặt vào bên trái của MSB, và tất cả các vị trí ở bên trái còn lại của MSB đều chứa giá trị 0. Điều kiện này sẽ chỉ thị cho khối điều khiển phát thực hiện một phép dịch cuối cùng và sau đó huỷ tác dụng của SEND và thiết lập cờ ngắt truyền TI. Cả 2 tác động này xảy ra tại thời điểm S1P1 của chu trình máy thứ 10 kể từ thời điểm "ghi vào SBUF".

Quá trình nhận tin được khởi đầu bằng điều kiện REN=1 và RI=0. Tại thời điểm S6P2 của chu trình máy tiếp theo, khối điều khiển nhận (Rx Control) sẽ ghi các bit 11111110 (Xóa RI) vào thanh ghi dịch nhận, và sẽ kích hoạt RECEIVE trong pha xung nhịp tiếp theo.

RECEIVE cho phép SHIFT CLOCK (đồng hồ xung nhịp) đưa đến đầu ra P3.1. SHIFT CLOCK sẽ tạo ra việc phát tin tại thời điểm S3P1 và S6P1 của mỗi chu trình máy. Tại giai đoạn S6P2 của mỗi chu trình máy khi RECEIVE có mức tích cực thì nội dung của thanh ghi dịch nhận tin được dịch sang trái một vị trí. Giá trị đưa vào từ bên phải là giá trị đã được tạo mẫu ở chân P3.0 tại thời điểm S5P2 của cùng chu trình máy.

Khi các bit dữ liệu được đưa vào từ bên phải, thì các giá trị 1 sẽ đi ra bên trái. Khi giá trị 0 (đã được nạp ban đầu vào vị trí tận cùng bên phải) dịch đến vị trí tận cùng bên trái trong thanh ghi dịch, thì nó chỉ thị cho khối điều khiển nhận thực hiện phép dịch cuối cùng và nạp vào SBUF. Tại thời điểm S1P1 của chu trình máy thứ 10 sau thời điểm ghi vào SCON (đã xoá RI), thì RECEIVE được xoá và RI được thiết lập.

2.9.5. Hoạt động của chế độ 1:

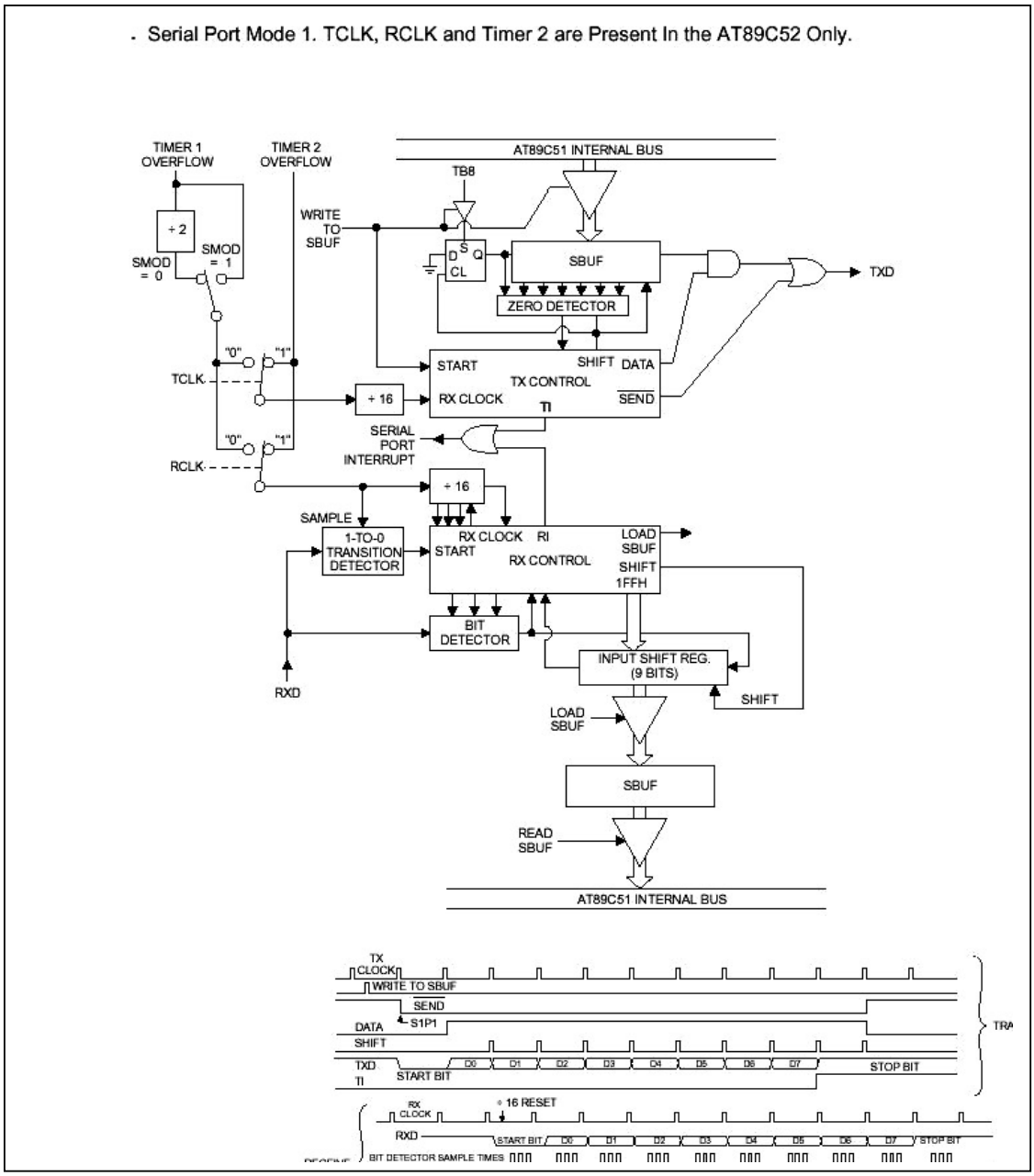
Ở chế độ này 10 bit được truyền (thông qua TxD) hoặc nhận (thông qua RxD) bao gồm: 1 bit khởi đầu(có giá trị 0), 8 bit dữ liệu (LSB đầu tiên) và 1 bit dừng (Có giá trị 1). Khi nhận tin, bit dừng chuyển vào RB8 trong SCON. Trong AT89C51, tốc độ Baud được xác định bằng tốc độ tràn của Timer 1. Trong AT89C52, tốc độ Baud được xác định bằng tốc độ tràn của Timer 1 hoặc tốc độ tràn của Timer 2 hoặc bằng tốc độ tràn của cả 2 bộ Timer này. Trong trường hợp này, khi cả 2 bộ Timer được sử dụng thì một bộ Timer sẽ xác định tốc độ truyền tin, còn bộ Timer kia xác định tốc độ nhận tin.

Hình 2.20 (Seriel Port Mode 1) là sơ đồ chức năng của cổng nối tiếp chế độ 1 và đồ thị thời gian liên quan tới quá trình truyền và nhận tin của chế độ này.

Quá trình truyền tin được khởi đầu bởi bất kỳ lệnh nào có sử dụng SBUF như 1 thanh ghi đích. Tín hiệu "ghi vào SBUF" sẽ nạp giá trị 1 vào bit thứ 9 của thanh ghi dịch truyền và bật cờ báo cho khối điều khiển phát (Tx Control) về yêu cầu cần truyền tin. Quá trình truyền thực tế bắt đầu tại thời điểm S1P1 của chu kỳ máy theo sau quá trình quay vòng (rollover) kế tiếp ở trong bộ đếm /16 (-:-16). Do đó, các thời điểm của bit truyền được đồng bộ với nhịp bộ đếm chia16, chứ không phải với tín hiệu "ghi vào SBUF".

Quá trình truyền tin bắt đầu khi /SEND được kích hoạt để mở cổng OR và bit khởi đầu được đặt tại TxD. Sau đó tín hiệu DATA được kích hoạt để mở tiếp cổng AND. Điều này

cho phép mở thông đường truyền từ thanh ghi dịch truyền đến đầu ra TxD. Xung nhịp đầu tiên để dịch các bit trong thanh ghi dịch truyền sẽ xuất hiện ngay sau đó.



Hình 2.18. Truyền nối tiếp ở chế độ 1

Khi các bit dữ liệu dịch sang phải, thì các giá trị 0 được đưa vào từ bên trái. Khi MSB của Byte dữ liệu ở vị trí đầu ra của thanh ghi dịch thì giá trị 1(ban đầu đã được nạp vào bit thứ 9) sẽ được điền vào ngay bên trái của bit MSB, còn các bit kể từ nó sang trái đều có giá trị 0. Điều kiện này sẽ chỉ thị cho khối điều khiển phát thực hiện lần dịch cuối cùng và sau đó đưa trả /SEND về mức thụ động, đồng thời thiết lập cờ ngắt TI. Thời điểm này rơi vào chu kỳ thứ 10 của bộ đếm chia 16, sau thời điểm "ghi vào SBUF".

Quá trình nhận tin được khởi đầu bằng việc phát hiện có sự chuyển trạng thái từ 1 về 0 ở đường thu nối tiếp RxD. Để phát hiện chính xác, tín hiệu trên RxD được lấy mẫu ở tốc độ gấp 16 lần tốc độ Baud của đường truyền. Khi sự chuyển trạng thái (từ 1 về 0) được phát hiện thì bộ đếm chia 16 được tái xác lập ngay và giá trị 1FFh được ghi vào thanh ghi dịch đầu vào (Input shift register). Việc tái thiết lập bộ đếm chia 16 sẽ đồng nhất thời điểm tràn của nó với các biên (ranh giới) thời gian của bit đang đi tới đầu thu.

Mỗi bit được chia thành 16 phần (States) thời gian bằng nhau (16 phần của bộ đếm). Tại các phần thời gian thứ 7, 8, 9 của mỗi bit, bộ phát hiện bit (Bit Detector) sẽ trích mẫu giá trị của RxD. Giá trị được chấp nhận là giá trị đã có ở ít nhất 2 trong 3 mẫu. Phương pháp này được thực hiện để chống nhiễu đường truyền. Nếu giá trị được chấp nhận đối với bit đầu tiên không phảI là 0 (không phảI bit START), thì các mạch thu được tái xác lập để quay lại chờ một đột biến từ 1 về 0 khác. Nếu bit khởi đầu (START) có giá trị hợp lệ thì dữ liệu được dịch vào thanh ghi dịch đầu vào, và quá trình nhận tin được tiếp tục.

Khi các bit dữ liệu đi vào từ bên phải của thanh ghi dịch, thì các giá trị 1 được dịch ra bên trái của nó. Khi bit khởi đầu đến vị trí tận cùng bên trái của thanh ghi dịch (ở chế độ 1, nó là thanh ghi 9 bit), nó sẽ chỉ thị cho khối điều khiển nhận (Rx Control) thực hiện phép dịch chuyển cuối cùng, rồi nạp SBUF và RB8, và thiết lập RI. Tín hiệu để nạp SBUF và RB8, và để thiết lập RI sẽ được tạo ra khi và chỉ khi các điều kiện sau đây được thoả mãn ở thời điểm xung nhịp cuối cùng được tạo ra:

1. RI=0, và
2. Hoặc SM2=0, hoặc bit STOP nhận được =1.

Nếu một trong hai điều kiện này không được thảo mãn, thì Byte tin nhận được sẽ bị mất. Nếu cả 2 điều kiện được thoả mãn, thì bit dừng chuyển vào RB8, 8 bit dữ liệu chuyển vào SBUF, và RI được kích hoạt. Tại thời điểm này, bất kể các điều kiện trên được thoả mãn hay không, thì khối điều khiển vẫn quay trở lại để tiếp tục chức năng phát hiện đột biến mới từ 1 về 0 trên đường thu tin RxD.

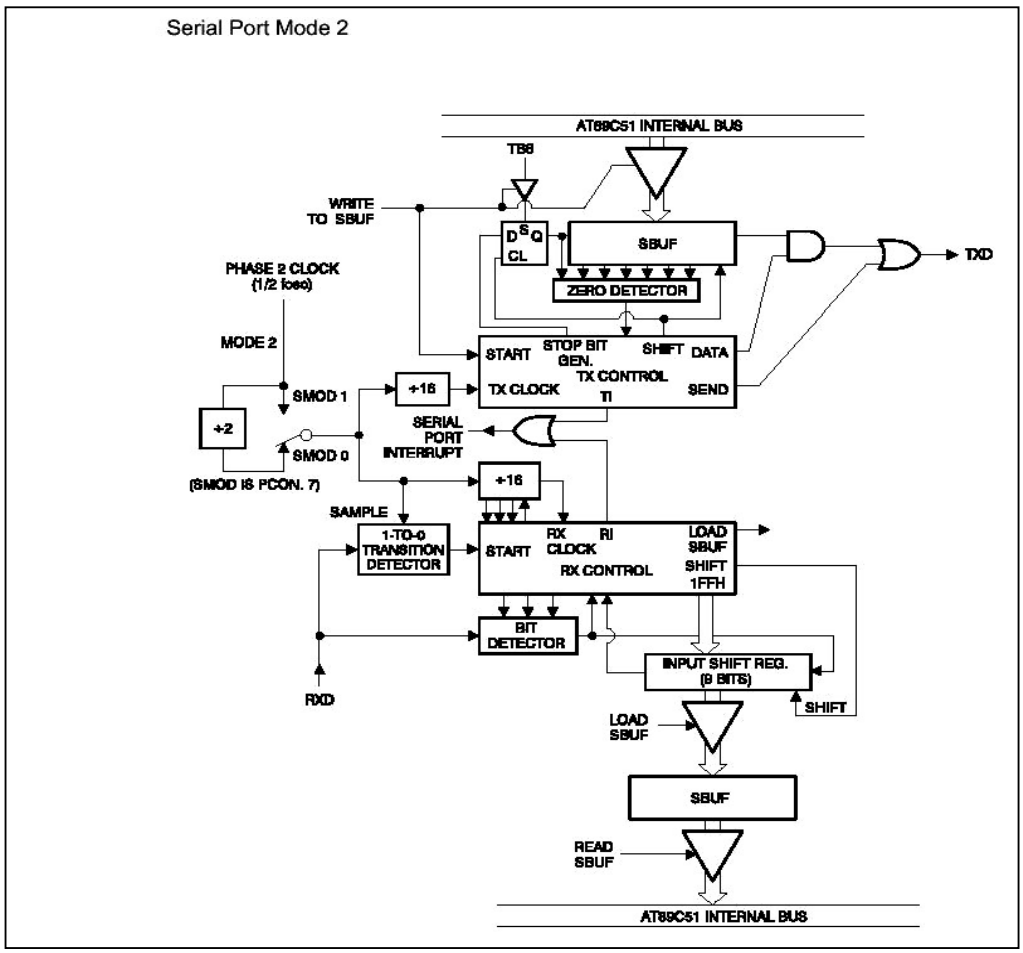
2.9.7 Hoạt động của chế độ 2 và 3:

Ở chế độ này 11 bit được truyền đi (thông qua TxD) hoặc nhận vào (thông qua RxD), bao gồm: 1 bit khởi đầu (0), 8 bit dữ liệu (LSB đầu tiên), 1 bit dữ liệu thứ 9 có thể lập trình được và 1 bit dừng (1). Khi truyền tin đi, bit dữ liệu thứ 9 (TB8) có thể được gán giá trị 0 hoặc 1. Khi nhận tin, bit dữ liệu thứ 9 chuyển vào RB8 trong SCON. Tốc độ Baud có thể lập trình được bằng 1/32 hoặc 1/64 tần số của bộ dao động ở chế độ 2. Chế độ 3 có thể có tốc độ Baud khả biến do Timer 1 hoặc Timer 2 tạo ra, tuỳ thuộc vào trạng thái của các bit TCLK và RCLK.

Hình 2.21 (Seriel Port Mode 2) và Hình 2.22 (Seriel Port Mode 3) là các sơ đồ chức năng và đồ thị thời gian của các chế độ 2 và 3. Phần nhận tin được tổ chức giống như chế độ 1. Phần truyền tin khác với chế độ 1 chỉ ở bit thứ 9 của thanh ghi dịch truyền.

Quá trình truyền tin được khởi đầu bằng bất kỳ lệnh nào mà có sử dụng SBUF như một thanh ghi đích. Tín hiệu "ghi vào SBUF" cũng nạp bit TB8 vào vị trí bit thứ 9 của thanh ghi dịch truyền và chỉ thị cho khối điều khiển truyền (Tx Control) rằng có yêu cầu phải truyền tin. Quá trình truyền được bắt đầu tại S1P1 của chu kỳ máy ngay sau thời điểm tràn của bộ đếm chia 16. Do đó, các bit được đồng bộ đối với chu kỳ bộ đếm chia 16, chứ không phải với tín hiệu "ghi vào SBUF".

Quá trình truyền bắt đầu khi tín hiệu /SEND được kích hoạt, và bit khởi đầu được đặt tại TxD. Sau đó đến tín hiệu DATA được kích hoạt. Điều này cho phép mở thông đường truyền từ thanh ghi dịch truyền đến đầu ra TxD. Xung nhịp đầu tiên chuyển giá trị 1 (Bit dừng) vào vị trí bit thứ 9 của thanh ghi dịch. Còn sau đó chỉ các giá trị 0 được đưa vào. Vì vậy, khi các bit dữ liệu dịch ra sang phải, thì các giá trị 0 được đưa vào từ bên trái. Khi TB8 ở vị trí đầu ra của thanh ghi dịch, thì bit dừng chuyển đến bên trái của TB8, và tất cả các giá trị ở bên trái của nó đều là giá trị 0. Điều kiện này chỉ thị cho khối điều khiển phát (Tx Control) thực hiện phép dịch cuối cùng và sau đó trả SEND về trạng thái thụ động, đồng thời thiết lập TI. Thời điểm này ứng với chu kỳ lần thứ 11 (sự quay vòng lần thứ 11) của bộ đếm chia 16 sau khi có tín hiệu "ghi vào SBUF".



Hình : 2.19 .Hoạt động ở chế độ 2

Quá trình nhận tin được khởi đầu bằng sự phát hiện đột biến (chuyển trạng thái) từ 1 về 0 ở RxD. Với mục đích đảm bảo độ tin cậy khi trích mẫu trên RxD, thì tốc độ trích mẫu được lấy gấp 16 lần tốc độ Baud của đường truyền (Tín hiệu trên RxD được lấy mẫu với tốc độ gấp 16 lần tốc độ Baud của đường truyền). Khi sự chuyển trạng thái từ 1 về 0 được phát hiện, bộ đếm chia 16 được tái xác lập (reset) lại ngay, và giá trị 1FFh được ghi vào thanh ghi dịch đầu vào.

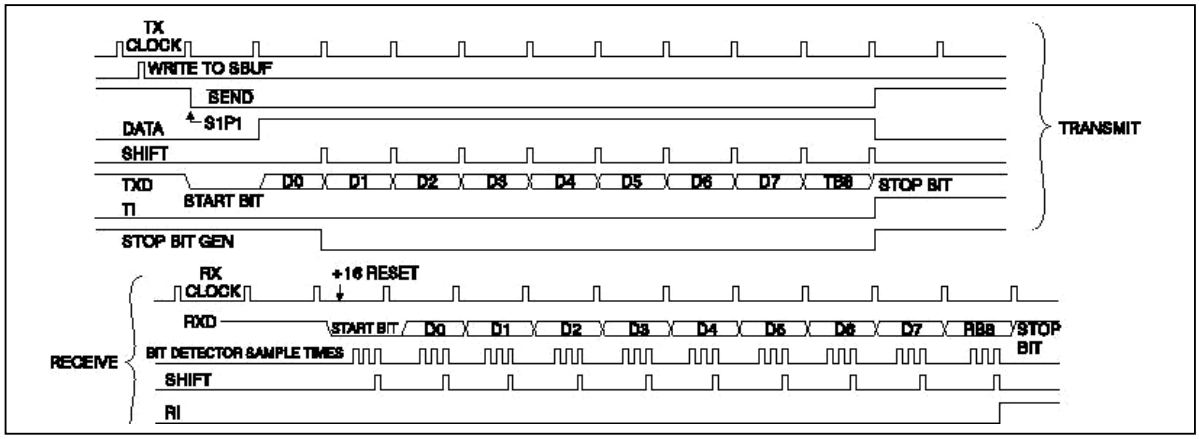
Ở các phần thời gian thứ 7, 8 và 9 của mỗi bit, bộ phát hiện bit (Bit Detector) sẽ trích mẫu giá trị của RxD. Giá trị được chấp nhận là giá trị đã thấy ở ít nhất 2 trong 3 mẫu. Phương pháp này được thực hiện để chống nhiễu đường truyền. Nếu giá trị được chấp nhận đối với bit đầu tiên không phải là 0 (không phải bit START), thì các mạch thu được tái xác lập để quay lại chờ một đột biến từ 1 về 0

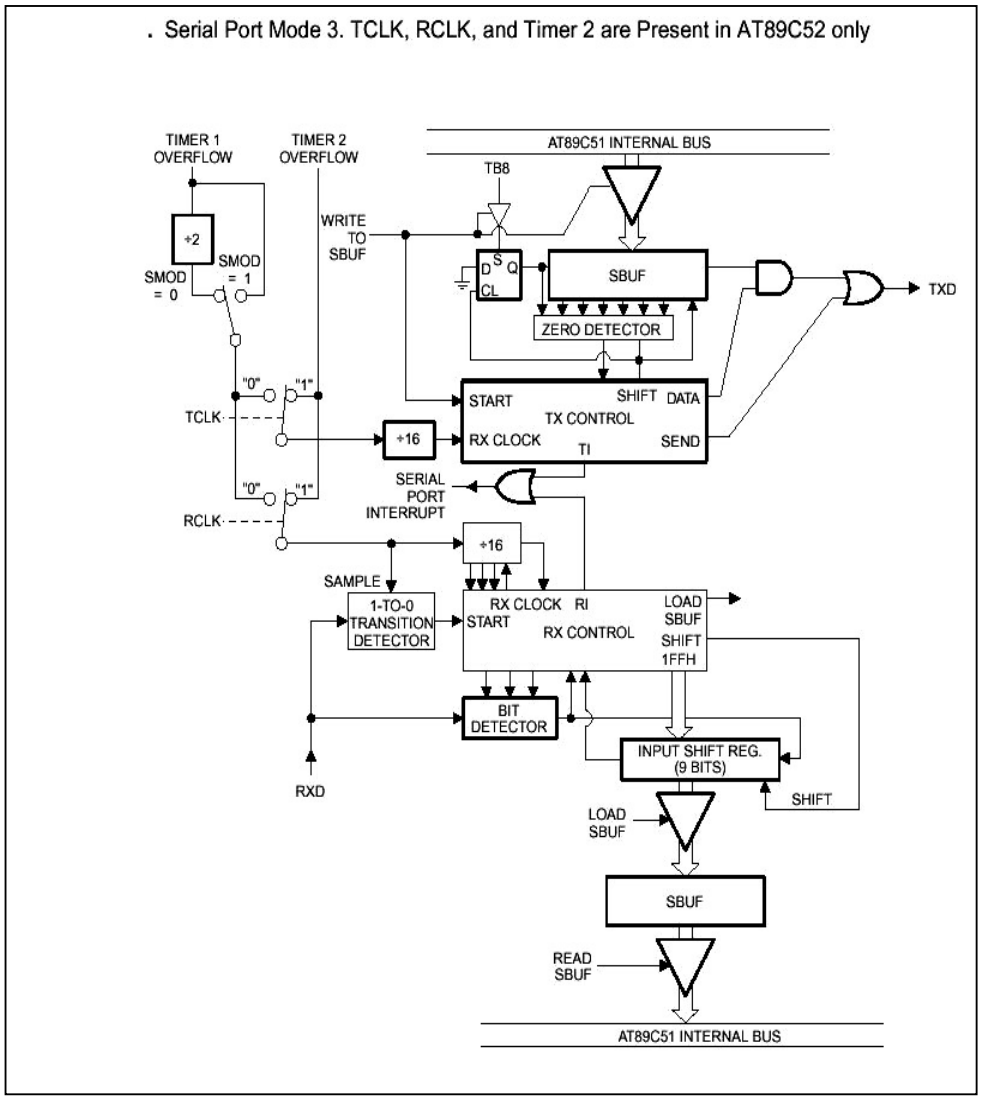
khác. Nếu bit khởi đầu (START) có giá trị hợp lệ thì dữ liệu được dịch vào thanh ghi dịch đầu vào, và quá trình nhận tin được tiếp tục.

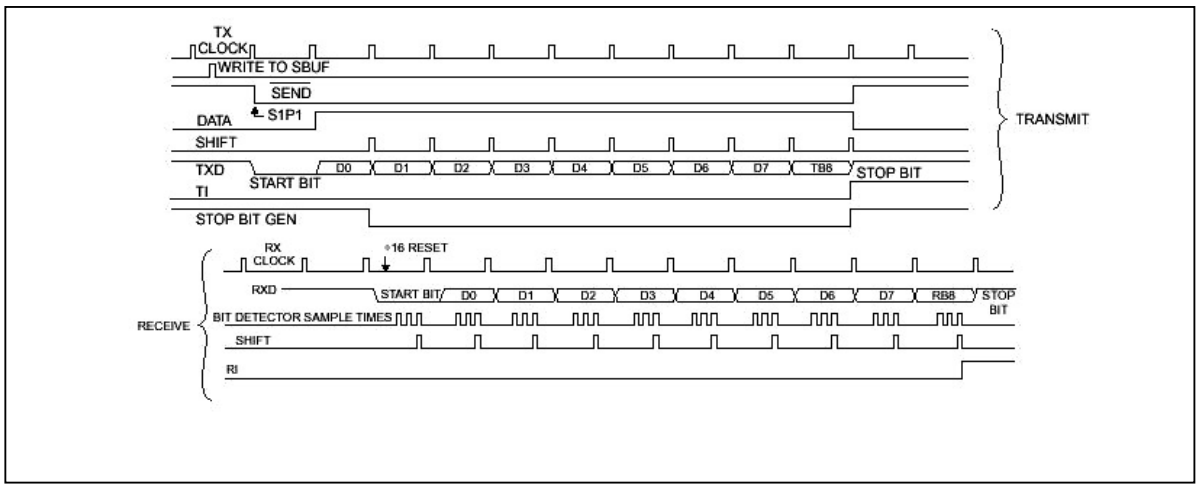
Khi các bit dữ liệu thu được đi vào từ bên phải của thanh ghi dịch, thì các giá trị 1 được dịch ra bên trái của nó. Khi bit khởi đầu đến vị trí tận cùng bên trái của
thanh ghi dịch (ở chế độ 2 và 3, nó là thanh ghi 9 bit), nó sẽ chỉ thị cho khối điều khiển nhận (Rx Control) thực hiện phép dịch chuyển cuối cùng, rồi nạp SBUF và RB8, và thiết lập RI. Tín hiệu để nạp SBUF và RB8, và để thiết lập RI sẽ được tạo
ra khi và chỉ khi các điều kiện sau đây được thoả mãn ở thời điểm xung nhịp cuối cùng được tạo ra:

1. RI=0, và
2. Hoặc SM2=0, hoặc bit STOP nhận được =1

Nếu một trong hai điều kiện này không được thảo mãn, thì Byte tin nhận được sẽ bị mất và RI cũng không được xác lập. Nếu cả 2 điều kiện được thoả mãn, thì bit dừng chuyển vào RB8, 8 bit dữ liệu chuyển vào SBUF, và RI được kích hoạt. Tại thời điểm này, bất kể các điều kiện trên được thoả mãn hay không, thì khối điều khiển vẫn quay trở lại để tiếp tục chức năng phát hiện đột biến mới từ 1 về 0 trên đường thu tin RxD.







Hình : 2.20 .Hoạt động ở chế độ 3

2.10. Nguyên lý khởi động của On-chip AT89C51:

RST là chân Reset, chính là đầu vào của Trigger Schmitt. Việc Reset được thực hiện bằng cách giữ chân RST ở mức cao ít nhất 2 chu kỳ máy(24 chu kỳ dao động).

Tín hiệu khởi động lại bên ngoài đưa vào chân RST không đồng bộ với xung Clock bên trong. Chân RST được lấy mẫu tại thời điểm P2S5 của mỗi chu kỳ máy. Các chân của cổng sẽ giữ hoạt động hiện hành của chúng cho 19 chu kỳ dao động sau khi giá trị logic 1 đã được lấy mẫu ở chân RST.

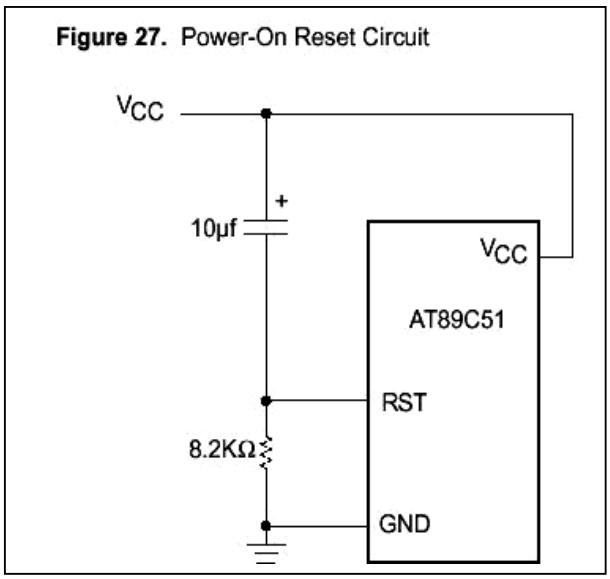
Trong khi chân RST ở mức cao, ALE và /PSEN được đẩy dần lên mức cao. Sau đó RST được đẩy xuống, nó sẽ giữ 1 đến 2 chu kỳ máy đối với ALE và /PSEN để khởi động xung Clock. Vì lý do này mà các dịch vụ khác(đưa từ ngoài vào) không thể đồng bộ được với bộ thời gian bên trong của AT89C51.

Khi Reset có hiệu lực thì giá trị của các SFR, được mô tả ở (Table -Reset Values of SFRs). Trong đó thanh ghi SBUF, và một số bit của PCON, T2MOD, IE, IP có giá trị bất định. Các chốt cổng từ P0,P3 có giá trị FFh, SP có giá trị 07h.Các thanh ghi còn lại có giá trị 00h

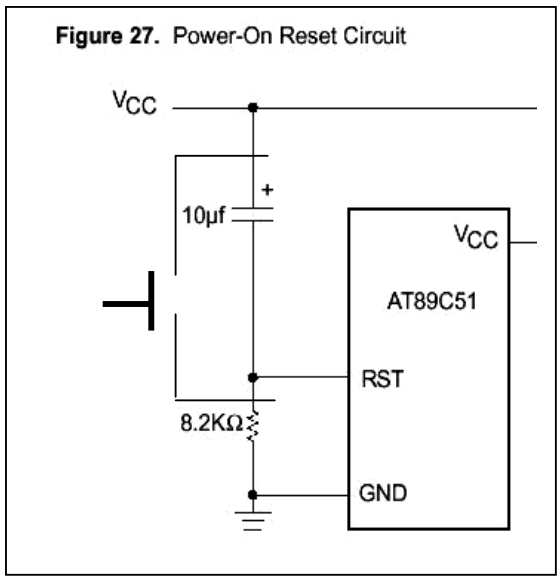
Riêng đối với RAM bên trong On-chip AT89S8252 khi cấp nguồn hay Reset lại không bị tác động, mà nội dung trong RAM có giá trị ngẫu nhiên.

Khởi động lại cho On-chip có thể hoạt động ở trạng thái tự động hoặc bán tự động . Khởi động tự động có thể được tạo ra khi cấp nguồn điện +Vcc cho on-chip bằng mạch điện RC. Sau khi cấp nguồn, mạch RC giữ cho chân RST ở trạng thái cao trong thời gian tuỳ thuộc vào hằng số thời gian của mạch RC. Để đảm bảo khởi động được on-chip, thời gian chân RST ờ trạng thái cao phải đủ lớn (Khoảng lớn hơn 2 chu kỳ máy) để mạch dao động chuyển sang trạng thái dao động ổn định

Nếu khởi động bán tự động, sau khi cấp nguồn mạch sẽ tự động Reset như khởi động tự động. Khi cần khởi động lại phải nhấn công tắc thường ngắt K, thời gian nhấn công tắc chân RST phải ở trạng thái cao.

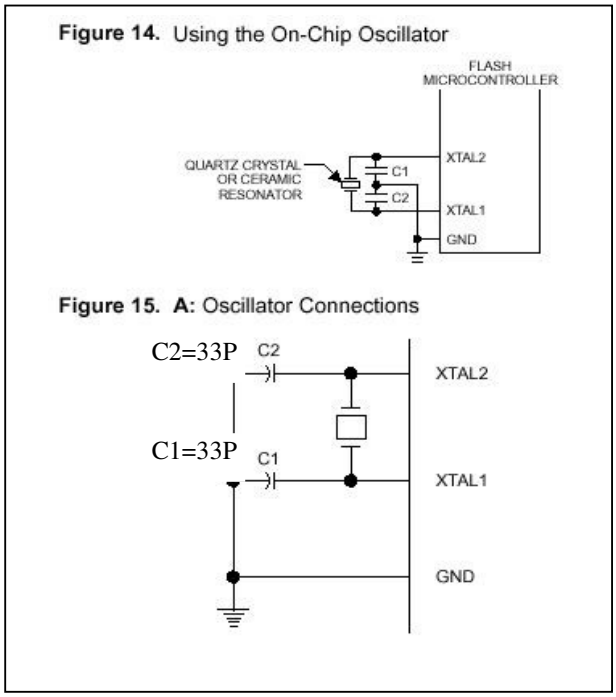


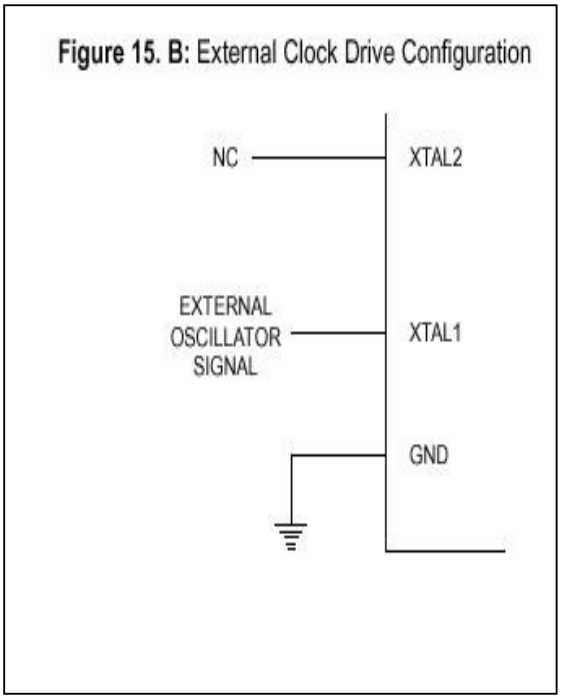
Hình 2.21. Khởi động tự động



Hình 2.22. Khởi động bán tự

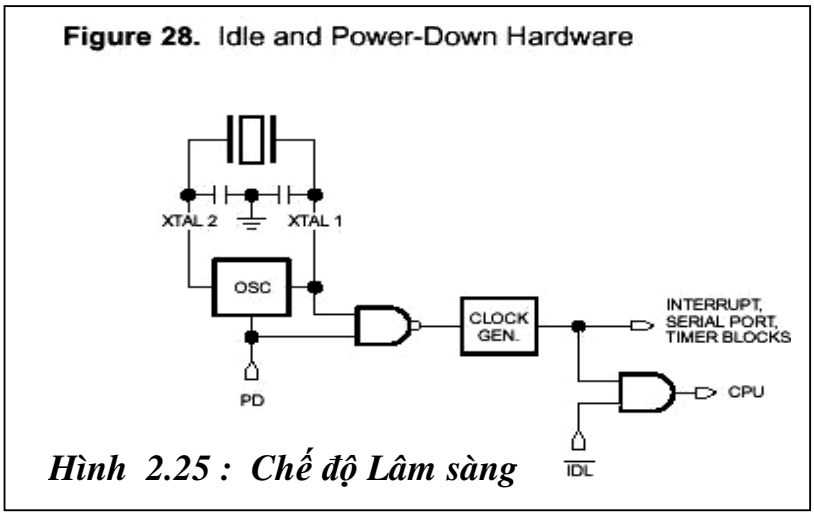
2.11. Mạch dao động.





Hình 2.23. Kết nối mạch dao động ***Hình 2.24. Dùng dao động ngoài***

2.12. Chế độ nguồn giảm và nghỉ



Hình 2.25 : Chế độ Lâm sàng

2.13. Kết nối hệ thống (Kết nối phần cứng).

(Kết nối hệ thống phần cứng, SV xem tài liệu VĐK 8051 của Thầy Tống Văn On)

2.13.1. Thiết kế mạch điều khiển LED đơn
2.13.2. Thiết kế mạch điều khiển LED 7 thanh
2.13.3. Thiết kế mạch điều khiển động cơ DC, động cơ bước...
2.13.4. Thiết kế mạch điều khiển đèn giao thông

CHƯƠNG III : Tập lệnh của bộ Vi điều khiển 80C51/89C51

3.1. Khái quát về tập lệnh của VĐK

Bộ VĐK có tập lệnh được tối ưu hoá để ứng dụng trong các hệ thống điều khiển, đo lường 8 bit. Để tăng khả năng truy xuất RAM nội trên các dữ liệu nhỏ, các kiểu định địa chỉ đặc biệt đã được áp dụng. Ngoài ra tập lệnh của VĐK còn hỗ trợ các biến 1 bit, cho phép quản lý bit trực tiếp trong các hệ logic và điều khiển bit có yêu cầu xử lý bit. Do họ VĐK AT89/80C51 có các mã lệnh 8 bit, nên số lệnh có thể lên đến 256 lệnh (thực tế có 255 lệnh, còn 1 lệnh chưa được định nghĩa).

Trong 255 byte thì bao gồm:

- 139 lệnh 1 byte
- 92 lệnh 2 byte
- 24 lệnh 3 byte

Mỗi lệnh đều được đặc trưng bởi mã lệnh hay còn gọi là mã máy, mã gợi nhớ, số byte của lệnh và số chu kỳ máy cần để thực thi lệnh.

Các lệnh của AT89/80C51 chia thành 5 nhóm lệnh:

- Nhóm lệnh di chuyển dữ liệu.
- Nhóm lệnh số học.
- Nhóm lệnh logic.
- Nhóm lệnh rẽ nhánh chương trình.
- Nhóm lệnh điều khiển biến logic.

3.2. Một số quy ước trong tập lệnh

❖ ***Các quy ước trong tập lệnh***

+ Rn: Thanh ghi R0-R7 của băng thanh ghi hiện hành đang được chọn để định địa chỉ thanh ghi.

+ Direct: Địa chỉ 8 bit của ô nhớ dữ liệu nội trú, nó có thể là ô nhớ trong RAM nội hoặc SFR. (00h-FFh)

+ @Ri: Ô nhớ 8 bit của RAM nội được định địa chỉ gián tiếp thông qua thanh ghi R0 hoặc R1.

+ Source (Src): toán hạng nguồn, có thể là Rn hoặc direct hoặc @Ri.

+ Dest: Toán hạng đích, có thể là Rn hoặc direct hoặc @Ri.

+ #Data: Hằng số 8 bit chứa trong lệnh.

+ #Data16: Hằng số 16 bit chứa trong lệnh.

+ Bit: Bit được định địa chỉ trực tiếp trong RAM nội trú hoặc SFR.

+ Rel: Offset 8 bit có dấu (từ -128 đến +127). Nó được lệnh SJMP và các lệnh nhảy có điều kiện sử dụng.

+ Addr11: địa chỉ 11 bit của bộ nhớ chương trình , thường dùng lệnh ACALL

+ Addr16: địa chỉ 16 bit của 64Kb bộ nhớ chương trình, thường dùng lệnh LCALL

❖ ***Một số chỉ dẫn thường gặp:***

ORG (Origin): Chỉ dẫn Org để báo địa chỉ bắt đầu của chương trình. Số sau của Org có thể viết dạng mã Hexa, nhị phân hay thập phân...

EQU (Equate): Nó được dùng để định nghĩa một hằng số, người ta gán cho nhãn bằng một giá trị hằng số nào đố và khi nhãn xuất hiện trong chương trình thì nhãn sẽ thay cho giá trị hằng số đó. Vậy ưu điểm của việc sử dụng chỉ dẫn EQU là khi trong chương trình mà giá trị của hằng số được sử dụng ở nhiều chỗ khác nhau, nay muốn thay đổi giá trị này thì ta chỉ việc thay một lần giá trị cho nhãn là xong.

END: Chỉ dẫn END sẽ báo cho trình dịch biết kết thúc một tập nguồn ASM, tất cả những lệnh sau chỉ dẫn END đều vô nghĩa và bị trình dịch bỏ qua.

Quy định tên nhãn: Tên của nhãn thường dùng các từ gợi nhớ và phải là duy nhất không được trùng. Có thể dùng các chữ cái viết hoa hay viết thường, các dấu ghạch chân, dấu chấm, các số 0, 1 , 2 , 3 vv... nhưng không được dùng các từ khoá trùng với các câu lệnh, phía sau của nhãn bao giờ cũng phải có dấu 2 chấm.

Các lời chú giải: Trong chương trình mà muốn có lời chú giải thì phía trước lời chú giải phải có dấu chấm phẩy. Tương tự như trong một chương trình mà muốn bỏ qua một câu lệnh nào đó phía trước của câu lệnh cũng phải có dấu chấm phẩy.

❖ ***Các ký hiệu dùng trong mô tả lệnh:***

Ký hiệu	***ý nghĩa***
<-	Được thay thế bởi…
( )	Nội dung của…
(( ))	Dữ liệu được trỏ bởi…
rrr	1 trong 8 thanh ghi (R0-R7) của các băng thanh ghi
dddddddd	Các bit dữ liệu
aaaaaaaa	Các bit địa chỉ
bbbbbbbb	địa chỉ của 1 bit
i	Định địa chỉ gián tiếp thông qua R0 hoặc R1
eeeeeeee	Địa chỉ tương đối 8 bit

❖ ***Các phương pháp định địa chỉ***

Tập lệnh của Vi điều khiển 80C51/89C51 cho phép sử dụng 7 phương pháp định địa chỉ , tuy nhiên ta thường xuyên sử dụng 5 cách :

- Phương pháp định địa chỉ tức thời Ví dụ: Mov A, # 15h
- Phương pháp định địa chỉ thanh ghi Ví dụ: Mov A, R2
- Phương pháp định địa chỉ trực tiếp Ví dụ: Mov R2, 03h
- Phương pháp định địa chỉ gián tiếp Ví dụ: Mov A, @ R0
- Phương pháp định địa chỉ cơ sở Ví dụ: Movc A, @ a + Dptr

Còn 2 cách định địa chỉ còn lại ta sẽ giới thiệu trong quá trình phân tích tập lệnh

3.3. Nhóm lệnh di chuyển dữ liệu

3.3.1. Lệnh di chuyển dạng Byte

Cú pháp câu lệnh: **MOV <dest-byte>, <src-byte>**

Chức năng: Sao chép nội dung của toán hạng nguồn vào toán hạng đích, nội dung của toán hạng nguồn không thay đổi. Lệnh này không làm ảnh hưởng tới các cờ và các thanh ghi khác.

Câu lệnh	*Số byte*	*Sốchukỳ*	*Mã lệnh*	*Hoạt động*
MOV A, Rn	1	1	11101rrr	(A)<-(Rn)
MOV A, direct	2	1	11100101 aaaaaaaa	(A)<-(direct)
MOV A, @Ri	1	1	1110111i	(A)<-((Ri))
MOV A, #data	2	1	01110100 dddddddd	(A)<-#data
MOV Rn, A	1	1	11111rrr	(Rn)<-(A))
MOV Rn, direct	2	2	10101rrr aaaaaaaa	(Rn)<-(direct)
MOV Rn, #data	2	1	01111rrr dddddddd	(Rn)<-#data
MOV direct, A	2	1	11110101 aaaaaaaa	(direct)<-(A)
MOV direct, Rn	2	2	10001rrr aaaaaaaa	(direct)<-(Rn)
MOV direct, direct	3	2	10000101 aaaaaaaa aaaaaaaa	(direct)<-(direct)
MOV direct, @Ri	2	2	1000011i aaaaaaaa	(direct)<-((Ri))
MOV direct, #data	3	2	01110101 aaaaaaaa dddddddd	(direct)<-#data
MOV @Ri, A	1	1	1111011i	((Ri))<-(A)
MOV @Ri, direct	2	2	1010011i	((Ri))<-(direct)
MOV @Ri, #data	2	1	0111011i dddddddd	((Ri))<-#data

Ví dụ 1: Chuyển nội dung của R1 vào RAM trong tại địa chỉ 30h.

- Cách 1: Định địa chỉ kiểu thanh ghi và trực tiếp

```
MOV     30h, R1     ; (30h) = (R1)
```

- Cách 2: Định địa chỉ kiểu trực tiếp

```
MOV     30h, 01h    ; (30h) = (01h)
```

- Cách 3: Định địa chỉ kiểu gián tiếp

```
MOV     R0, #30h    ; (R0) = 30h
MOV     @R0, 01h    ; ((R0)) = (01h)
```

Ví dụ 2*:* Chuyển nội dung 35h vào RAM trong tại địa chỉ 20h.

- Cách 1: Định địa chỉ kiểu dữ liệu tức thời và trực tiếp

```
MOV     20h, #35h   ; (20h) = 35
```

- Cách 2: Định địa chỉ kiểu dữ liệu tức thời và gián tiếp

```
MOV     R1,#20h     ; (R1) = 20h
MOV     @R1,#35h    ; ((R1)) = 35h
```

3.3.2. Lệnh MOV *di chuyển dạng Bít* :

Cú pháp câu lệnh: **MOV <dest-bit>, <scr-bit>**

Chức năng: Chuyển bit dữ liệu ở dạng sao chép toán hạng nguồn vào toán hạng đích. Một trong 2 toán hạng phải là cờ nhớ (C), toán hạng còn lại sẽ là bit bất kỳ được định địa chỉ trực tiếp. Lệnh không làm ảnh hưởng tới các thanh ghi khác hoặc các cờ khác.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
MOV C, bit	2	1	10100010 bbbbbbbb	(C)<-(bit)
MOV bit, C	2	2	10010010 bbbbbbbb	(bit)<-(C)

Ví dụ: Thiết lập cờ nhớ bằng lệnh MOV

```
MOV     A, #80h     ; (A) = 80h
MOV     C, Acc.7    ; (C) =(Acc.7)  = 1
```

3.3.3. Lệnh di chuyển dạng Word

Cú pháp câu lệnh: **MOV DPTR, #data16**

Chức năng: Giá trị 16 bit ở toán hạng thứ 2 trực tiếp trong câu lệnh được nạp vào thanh ghi DPTR. Hằng số 16 bit này được đặt ở byte 2 và byte 3 của lệnh. Byte 2 là byte cao được nạp cho thanh ghi DPH, byte 3 là byte thấp được nạp vào thanh ghi DPL. Lệnh này không ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Sốchu kỳ*	*Mã lệnh*	*Hoạt động*
MOV DPTR,#data16	3	2	10010000 dddddddd	(C)<-(bit)

Ví dụ: Trỏ DPTR tới vùng dữ liệu có địa chỉ bắt đầu là 1020h

```
- Cách 1:   MOV     DPTR,#1020h     ; (DPTR) = 1020h
- Cách 2:   MOV     DPH,#10h        ; (DPH) = 10h
            MOV     DPL, #20h       ; (DPL) = 20h
```

3.3.4. Lệnh chuyển Byte mã lệnh

***Cú pháp câu lệnh*: MOVC A, @A + <thanh ghi cơ sở>**

Chức năng: Nạp cho thanh ghi tích luỹ byte mã lệnh từ bộ nhớ chương trình. Địa chỉ của byte được tìm nạp trong bộ nhớ là tổng nội dung của thanh ghi A 8 bit với nội dung của thanh ghi cơ sở 16 bit (có thể là DPTR hoặc PC - thanh ghi đếm chương trình). Trong trường hợp sau, PC được tăng để trỏ đến địa chỉ của lệnh tiếp theo ((PC)<-(PC+1)) trước khi được công với nội dung của thanh ghi A, còn thanh ghi DPTR không bị thay đổi. Lệnh không ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
MOVC A,@A+DPTR	1	2	10010011	(A)<-((A)+(DPTR))
MOVC A,@A+PC	1	2	10000011	(A)<-((A)+(PC))

Ví dụ: Chuyển nội dung tại địa chỉ 1234h ở trong bộ nhớ chương trình ra cổng P1

```
MOV     A, #0           ; (A) = 0
MOV     DPTR, #1234h    ; (DPTR) = 1234h
MOVC    A, @A+DPTR      ; (A) = ((DPTR))
MOV     P1, A           ; (P1) = (A)
```

3.3.5. Lệnh truy xuất dữ liệu ngoài

Cú pháp câu lệnh: **MOVX <dest-byte>, <src-byte>**

Chức năng: Chuyển dữ liệu giữa thanh ghi tích luỹ với bộ nhớ ngoài. Các lệnh này được chia làm 2 loại, một loại cung cấp địa chỉ 8 bit và 1 loại cung cấp địa chỉ 16 bit.

Nếu dữ liệu được chuyển là 8 bit, nội dung của R0 hoặc R1 trong băng thanh ghi hiện hành sẽ cung cấp địa chỉ 8 bit đa hợp với dữ liệu trên P0. 8 bit địa chỉ này đủ để mã hoá cho các cổng I/O mở rộng bên ngoài chip hoặc cho 1 dãy RAM kích thước tương đối nhỏ. Với các dãy RAM có kích thước lớn hơn một chút, một vài chân của cổng bất kỳ nào đó có thể được sử dụng để tạo ra các bit địa chỉ cao. Các chân này nên được điều khiển bởi 1 lệnh xuất đặt trước lệnh MOVX.

Nếu dữ liệu được chuyển là 16 bit, thì DPTR tạo ra địa chỉ 16 bit. P2 xuất ra 8 bit địa chỉ cao (nội dung của DPH), còn P0 xuất ra 8 bit địa chỉ thấp đa hợp với dữ liệu.Thanh ghi chức năng đặc biệt P2 duy trì nội dung trước đó trong khi các bộ đệm xuất của P2 đang phát các nội dung của DPH. Dạng này nhanh hơn và hiệu quả hơn khi truy xuất nhiều dãy dữ liệu rất lớn (lên đến 64 Kb) do ta không cần thêm lệnh để thiết lập các cổng khác.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
MOVX A, @Ri	1	2	11100011	(A)<-((Ri))
MOVX @Ri, A	1	2	11110011	((Ri))<(A)
MOVX A, @DPTR	1	2	11100000	(A)<-((DPTR))
MOVX @DPTR, A	1	2	11110000	((DPTR))<-(A)

Ví dụ: Chuyển nội dung 34h vào địa chỉ 1234h ở RAM ngoài

```
MOV     A, #34h         ; (A) = 34h
MOV     DPTR, #1234h    ; (DPTR) = 1234h
MOVX    @DPTR, A        ; ((DPTR)) = (A)
```

3.3.6. Lệnh chuyển dữ liệu vào ngăn xếp

Cú pháp câu lệnh: **PUSH direct**

Chức năng: Chuyển số liệu có trong câu lệnh vào ngăn xếp. Trước tiên, con trỏ ngăn xếp (SP) được tăng lên 1, sau đó số liệu sẽ được chuyển vào đỉnh của ngăn xếp mà địa chỉ đỉnh này được trỏ bởi SP. Ngăn xếp nằm ở RAM nội trú.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
PUSH direct	2	2	11000000 aaaaaaaa	(SP)<-(SP+1) ((SP))<-(direct)

Ví dụ: Ghi giá trị 55h vào địa chỉ 35h của ngăn xếp

```
MOV     SP,#34h     ; (SP) = 34h
MOV     A, #55h     ; (A) = 55h
PUSH    0E0h        ; (SP) = (SP) + 1 = 35h
                    ; ((SP)) = (A)
```

Ta thấy rằng, khi lệnh PUSH được thực hiện thì giá trị mà SP đang trỏ sẽ tự động tăng lên 1 đơn vị.

3.3.7. Lệnh chuyển dữ liệu ra khỏi ngăn xếp

***Cú pháp câu lệnh*:** **POP direct**

Chức năng: Chuyển nội dung của ngăn xếp ở RAM trong, có địa chỉ được SP trỏ tới đến nơi có địa chỉ trực tiếp trong câu lệnh. Sau đó, con trỏ ngăn xếp (SP) được giảm đi 1. Lệnh không ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
POP direct	2	2	11010000 aaaaaaaa	(direct)<-((SP)) (SP)<-(SP-1)

Ví dụ: Chuyển nội dung tại đỉnh ngăn xếp (có địa chỉ 20h) ra A

```
MOV     SP,#20h     ; (SP) = 20h
POP     0E0h        ; (A) = ((SP))
                    ; (SP) = (SP) -1= 1Fh
```

Sau khi thực hiện lệnh POP thì giá trị của SP sẽ giảm đi 1 đơn vị.

3.3.8. Lệnh hoán chuyển dữ liệu

***Cú pháp câu lệnh*:** **XCH A, <byte>**

Chức năng: Hoán chuyển nội dung giữa thanh ghi A với thanh ghi hoặc bộ nhớ có địa chỉ chứa trong toán hạng thứ 2 của câu lệnh. Toán hạng thứ 2 có thể được định địa chỉ kiểu thanh ghi, thanh ghi trực tiếp hoặc thanh ghi gián tiếp.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
XCH A, Rn	1	1	11001rrr	(A)<-->(Rn)
XCH A, direct	2	1	11000101 aaaaaaaa	(A) <-->(direct)
XCH A, @Ri	1	1	1100011i	(A) <-->((Ri))

Ví dụ: Hoán chuyển nội dung của A với nội dung của R1

```
XCH     A, R1       ; (A)<-->(R1)
```

Sau khi thực hiện lệnh XCH thì nội dung của A trước đây sẽ nằm trong thanh ghi R1 còn nội dung của R1 sẽ được chuyển vào A.

3.3.9. Lệnh hoán chuyển nội dung 4 bít thấp

Cú pháp câu lệnh: **XCHD A, @Ri**

Chức năng: Hoán chuyển 4 bit thấp nội dung trong thanh ghi A với ô nhớ của RAM bên trong, có địa chỉ được định gián tiếp qua thanh ghi được chỉ ra trong lệnh. Lệnh này không ảnh hưởng tới trạng thái các cờ và nửa cao của các thanh ghi trong lệnh.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
XCHD A, @Ri	1	1	1101011i	(A3-A0)<-->((Ri3-Ri0))

Ví dụ: Hoán chuyển 4 bit thấp của A với 4 bit thấp của R5

```
MOV     30h, R5     ; (30h) = (R5)
MOV     R0, #30h    ; (R0) = 30h
XCHD    A, @R0      ; (A3-A0)<-->((R0.3-R0.0))
```

3.4. Nhóm lệnh số học

3.4.1. Lệnh thực hiện phép cộng

Cú pháp của câu lệnh: **ADD A, <scr-byte>**

Chức năng: Cộng giá trị 1 byte ở địa chỉ được chỉ ra ở câu lệnh với nội dung trong thanh ghi tích luỹ, kết quả được lưu vào thanh ghi tích luỹ. Nếu có nhớ từ bit số 7 hoặc bit số 3 thì cờ nhớ hoặc cờ nhớ phụ được thiết lập, ngược lại các cờ nêu trên được xoá. Khi cộng 2 số nguyên không dấu mà bị tràn thì cờ nhớ cũng được thiết lập để cho ta biết phép toán bị tràn. Trường hợp thực hiện lệnh ADD mà có nhớ từ bit số 6 nhưng không có nhớ từ bit số 7, hoặc có nhớ từ bit số 7 nhưng không có nhớ từ bit số 6 thì cờ tràn sẽ được thiết lập, ngược lại thì OV bị xoá. Khi cộng 2 số nguyên có dấu mà tổng là 1 số âm thì OV được thiết lập.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
ADD A, Rn	1	1	00101rrr	(A)<- (A) + (Rn)
ADD A, direct	2	1	00100101 aaaaaaaa	(A)<- (A) + (direct)
ADD A, @Ri	1	1	0010011i	(A)<- (A) + ((Ri))
ADD A, #data	2	1	00100100 dddddddd	(A)<- (A) + #data

Ví dụ 1: Thực hiện phép tính F = 25h + 34h

```
MOV     A, #25h     ; (A) = 25h
ADD     A, #34h     ; (A) = (A) + 34h = 59h
                    ; (PSW) = 00h
```

```
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 ; Nhớ 1

0 - 0 = 0
0 - 1 = 1; mượn1
1 - 0 = 1
1 - 1 = 0
```

Ví dụ 2:

3.4.2. Lệnh cộng có nhớ

Cú pháp của câu lệnh: **ADDC A, <scr-byte>**

Chức năng: Cộng đồng thời nội dung của 1 byte ở địa chỉ được chỉ ra trong câu lệnh với nội dung chứa trong thanh ghi tích luỹ và cờ nhớ. Nếu có nhớ từ bit số 7 hoặc số 3 thì cờ nhớ hoặc cờ nhớ phụ được thiết lập bằng 1, ngược lại các cờ nêu trên bị xoá. Khi cộng các số nguyên không dấu mà bị tràn thì cờ nhớ cũng được thiết lập. Trường hợp thực hiện lệnh ADDC mà có nhớ từ bit số 6 nhưng không nhớ từ bit số 7, hoặc có nhớ từ bit só 7 nhưng không nhớ từ bit số 6 thì cờ tràn sẽ được thiết lập, ngược lại cờ này bị xoá. Khi cộng các số nguyên có dấu mà tổng là 1 số âm thì OV được thiết lập.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
ADDC A, Rn	1	1	00110rrr	(A)<- (A) + (C) + (Rn)
ADDC A, direct	2	1	00110101 aaaaaaaa	(A)<- (A) + (C) + (direct)
ADDC A, @Ri	1	1	0011011i	(A)<- (A) + (C) + ((Ri))
ADDC A, #data	2	1	00110100 dddddddd	(A)<- (A) + (C) + #data

Ví dụ: Thực hiện phép tính F = 25h + 34h + (C)

```
MOV     A, #25h     ; (A) = 25h
SETB    C           ; (C) = 1
ADDC    A, #34h     ; (A) = (A) + 34h + (C) = 5Ah
                    ; (PSW) = 00h
```

3.4.3. Lệnh thực hiện phép trừ

Cú pháp của câu lệnh: **SUBB A, <scr-byte>**

Chức năng: Trừ thanh ghi tích luỹ cho toán hạng thứ 2 và cờ nhớ, kết quả được lưu vào thanh ghi tích luỹ. Cờ nhớ được đặt bằng 1 nếu có số mượn được cần đến cho bit số 7, ngược lại thì cờ nhớ bị xoá. Cờ nhớ phụ được thiết lập nếu có nhớ cho bit 3. Trường hợp thực hiện lệnh SUBB mà có số mượn được cần đến cho bit 7(không phải cho bit 6), hoặc cho bit 6 (không phải cho bit 7) thì cờ tràn sẽ được thiết lập, ngược lại thì OV bị xoá. Khi trừ các số nguyên có dấu mà kết quả là 1 số âm thì OV được thiết lập.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
SUBB A, Rn	1	1	10011rrr	(A)<- (A) - (C) - (Rn)
SUBB A, direct	2	1	10010101 aaaaaaaa	(A)<- (A) - (C) - (direct)
SUBB A, @Ri	1	1	1001011i	(A)<- (A) - (C) - ((Ri))
SUBB A, #data	2	1	10010100 dddddddd	(A)<- (A) - (C) - #data

Ví dụ: Thực hiện phép tính F = 95h - 34h

```
MOV     A, #95h     ; (A) = 95h
SUBB    A, #34h     ; (A) = (A) - (C) - 34h = 61h
                    ; (PSW) = 05h
```

3.4.4. Lệnh thực hiện phép nhân

Cú pháp của câu lệnh: **MUL AB**

Chức năng: Nhân các số nguyên không dấu 8 bit trong thanh ghi tích luỹ với thanh ghi B. Byte thấp của kết quả 16 bit được lưu trong thanh ghi tích luỹ, còn byte cao được lưu trong thanh ghi B. Nếu kết quả lớn hơn 0FFh thì cờ tràn được thiết lập, cờ nhớ luôn bị xoá.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
MUL AB	1	4	10100100	(B)<- byte cao của (A)x(B) (A)<- byte thấp của (A)x(B)

Ví dụ: Thực hiện phép tính F = 80h x 90h

```
MOV     A, #80h     ; (A) = 80h = 128
MOV     B,#90h      ; (B) = 90h = 144
MUL     AB          ; (A) = 00h và (B) = 48h
                    ; F = 4800h = 18432
                    ; (PSW) = 04h
```

3.4.5. Lệnh thực hiện phép chia

Cú pháp của câu lệnh: **DIV AB**

Chức năng: Chia số nguyên không dấu 8 bit trong thanh ghi tích luỹ cho số nguyên không dấu 8 bit trong thanh ghi B. Thương số được lưu trong thanh ghi tích luỹ, còn số dư được lưu trong thanh ghi B. Cờ tràn và cờ nhớ bị xoá.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
DIV AB	1	4	10000100	(A)<- thương của (A)/(B) (B)<- số dư của (A)/(B)

Ví dụ1: Thực hiện phép tính F = 65 : 15

```
MOV     A, #65; (A) = 65 = 41h
MOV     B,#15       ; (B) = 15 = 0Fh
DIV     AB          ; (A) = 04h và (B) = 05h
                    ; F = 4 dư 5
                    ; (PSW) = 01h
```

Ví dụ2:…….

3.4.6. Lệnh tăng lên 1 đơn vị

Cú pháp của câu lệnh: **INC <byte>**

Chức năng: Tăng giá trị của byte trong câu lệnh lên 1 đơn vị. Nếu giá trị ban đầu của byte là 0FFh, thì sau khi thực hiện lệnh INC nội dung của byte sẽ là 00h. Lệnh này không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
INC A	1	1	00000100	(A)<- (A) + 1
INC Rn	1	1	00001rrr	(Rn)<- (Rn) + 1
INC direct	2	1	00000101 aaaaaaaa	(direct)<- (direct) + 1
INC @Ri	1	1	0000011i	((Ri))<- ((Ri)) + 1

Ví dụ: Tăng nội dung tại địa chỉ FEh của RAM nội lên 2 đơn vị

```
MOV     R1,#0FEh        ; (R1) = 0FEh
INC     @R1             ; (0FEh) = (0FEh) +1
INC     @R1             ; (0FEh) = (0FEh) +1
```

3.4.7. Lệnh giảm đi 1 đơn vị

Cú pháp của câu lệnh: **DEC <byte>**

Chức năng: Giảm giá trị của byte trong câu lệnh xuống 1 đơn vị. Nếu giá trị ban đầu của byte là 00h, thì sau khi thực hiện lệnh DEC nội dung của byte sẽ là 0FFh. Lệnh này không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
DEC A	1	1	00010100	(A)<- (A) - 1
DEC Rn	1	1	00011rrr	(Rn)<- (Rn) - 1
DEC direct	2	1	00010101 aaaaaaaa	(direct)<- (direct) - 1
DEC @Ri	1	1	0001011i	((Ri))<- ((Ri)) - 1

Ví dụ 1: Giảm nội dung tại địa chỉ 7Eh của RAM nội đi 2 đơn vị

```
DEC     7Eh             ; (7Eh) = (7Eh) -1
DEC     7Eh             ; (0FEh) = (7Eh) -1
```

Ví dụ 2 : ……

3.4.8. Lệnh tăng con trỏ dữ liệu(Dptr)

Cú pháp của câu lệnh: **INC DPTR**

Chức năng: Tăng con trỏ dữ liệu lên 1 đơn vị. Khi byte thấp của con trỏ dữ liệu bị tràn, thì byte cao của con trỏ dữ liệu tăng lên 1 đơn vị. Lệnh này không ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
INC DPTR	1	2	10100011	(DPTR)<- (DPTR) + 1

Ví dụ: Tăng nội dung của DPTR lên 2 đơn vị

```
INC     DPTR ; (DPTR) = (DPTR) +1
INC     DPTR ; (DPTR) = (DPTR) +1
```

3.4.9. Lệnh hiệu chỉnh số thập phân

Cú pháp của câu lệnh: **DA A**

Chức năng: Hiệu chỉnh thập phân nội dung 8 bit trong thanh ghi A sau khi thực hiện phép cộng.

Nếu 4 bit thấp trong thanh ghi A có giá trị lớn hơn 9 hoặc cờ nhớ phụ được thiết lập thì phải cộng thêm 6 vào thanh ghi A để cho chữ số thập phân được chính xác. Phép cộng này sẽ đặt cờ nhớ nếu số nhớ từ 4 bit thấp chuyển đến tất cả 4 bit cao, ngược lại phép toán không xoá cờ nhớ.

Nếu 4 bit cao trong thanh ghi A có giá trị lớn hơn 9 hoặc cờ nhớ (CF) được thiết lập, thì cũng phải cộng thêm 6 vào thanh ghi A.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
DA A	1	1	11010100	A<- Kq Byte thấp B<- Kq Byte cao

Mô tả hoạt động:

- Nếu [[(A3-A0)>9]hoặc[(AC)=1]] thì (A3-A0)<-(A3-A0) + 6
- Nếu [[(A7-A4)>9]hoặc[(C)=1]] thì (A7-A4)<-(A7-A4) + 6

Ví dụ: Hiệu chỉnh số thập phân sau khi thực hiện phép cộng có nhớ.

```
MOV     A,#56h      ; (A) = 56h
SETB    C           ; (C) = 1
ADDC    A,#67h      ; (A) = 56h + (C) + 67h = 0BEh
DA      A           ; Eh+6= 4 nhớ 1
                    ; Bh+6+1 (gọi nhớ) = 2 nhớ 1
                    ; (C) = 1
                    ; (A) = 24h và chỉ ra các digit dạng
                    ; BCD của số thập phân 24
```

Ví dụ 2 : …….

3.5. Nhóm lệnh lôgic học

3.5.1. Lệnh AND Byte

Cú pháp câu lệnh: **ANL <dest-byte>, <src-byte>**

Chức năng: Thực hiện phép toán logic AND theo mức bit giữa các biến dài 1 byte đã cho, kết quả được lưu vào toán hạng đích. Toán hạng nguồn cho phép 6 chế độ địa chỉ hoá. Khi toán hạng đích là thanh ghi tích luỹ thì toán hạng nguồn có thể là thanh ghi, trực tiếp, thanh ghi-gián tiếp hoặc tức thời. Khi toán hạng đích là địa chỉ trực tiếp thì toán hạng nguồn có thể là thanh ghi tích luỹ hoặc dữ liệu tức thời. Lệnh này không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
ANL A, Rn	1	1	01011rrr	(A)<-(A) AND (Rn)
ANL A, direct	2	1	01010101 aaaaaaaa	(A)<-(A) AND (dir.)
ANL A, @Ri	1	1	0101011i	(A)<- (A)AND ((Ri))
ANL A, #data	2	1	01010100 dddddddd	(A)<- (A) AND #data
ANL direct, A	2	1	01010010 aaaaaaaa	(dir.)<-(dir.)AND (A)
ANL direct, #data	3	2	01010011 aaaaaaaa dddddddd	(dir)<(dir)AND#data

Ví dụ 1: Thực hiện phép AND logic giữa 2 Byte 54h và 67h

```
MOV     A, #54h     ; (A) = 54h
ANL     A, #67h     ; (A) = 54h AND 67h =  44h
```

Ví dụ 2:

3.5.2 Lệnh OR Byte

Cú pháp câu lệnh: **ORL <dest-byte>, <src-byte>**

Chức năng: Thực hiện phép toán logic OR theo mức bit giữa các biến dài 1 byte đã cho, kết quả được lưu vào toán hạng đích. Toán hạng nguồn cho phép 6 chế độ địa chỉ hoá. Khi toán hạng đích là thanh ghi tích luỹ thì toán hạng nguồn có thể là thanh ghi, trực tiếp, thanh ghi-gián tiếp hoặc tức thời. Khi toán hạng đích là địa chỉ trực tiếp thì toán hạng nguồn có thể là thanh ghi tích luỹ hoặc dữ liệu tức thời. Lệnh này không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
ORL A, Rn	1	1	01001rrr	(A)<-(A) OR (Rn)
ORL A, direct	2	1	01000101 aaaaaaaa	(A)<-(A) OR (dir.)
ORL A, @Ri	1	1	0100011i	(A)<- (A) OR ((Ri))
ORL A, #data	2	1	01000100 dddddddd	(A)<- (A) OR #data
ORL direct, A	2	1	01000010 aaaaaaaa	(dir.)<-(dir.) OR (A)
ORL direct, #data	3	2	01000011 aaaaaaaa dddddddd	(dir.)<(dir.) OR #data

Ví dụ: Thực hiện phép OR logic giữa 2 Byte 55h và 0A5h

```
MOV     A, #55h     ; (A) = 55h
ORL     A, #0A5h    ; (A) = 55h OR 0A5h = 0F5h
```

3.5.3 Lệnh X-OR Byte

Cú pháp câu lệnh: **XRL <dest-byte>, <src-byte>**

Chức năng: Thực hiện phép toán logic X-OR theo mức bit giữa các biến dài 1 byte đã cho, kết quả được lưu vào toán hạng đích. Toán hạng nguồn cho phép 6 chế độ địa chỉ hoá. Khi toán hạng đích là thanh ghi tích luỹ thì toán hạng nguồn có thể là thanh ghi, trực tiếp, thanh ghi-gián tiếp hoặc tức thời. Khi toán hạng đích là địa chỉ trực tiếp thì toán hạng nguồn có thể là thanh ghi tích luỹ hoặc dữ liệu tức thời. Lệnh này không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
XRL A, Rn	1	1	01101rrr	(A)<-(A) XOR (Rn)
XRL A, direct	2	1	01100101 aaaaaaaa	(A)<-(A) XOR (dir.)
XRL A, @Ri	1	1	0110011i	(A)<- (A) XOR((Ri))
XRL A, #data	2	1	01100100 dddddddd	(A)<- (A) XOR #data
XRL direct, A	2	1	01100010 aaaaaaaa	(dir.)<-(dir.)XOR(A)
XRL direct, #data	3	2	01100011 aaaaaaaa dddddddd	(dir.)<(dir.)XOR#data

Ví dụ 1 : Thực hiện hàm khác dấu giữa 2 Byte 67h và 78h

```
MOV     A, #67h     ; (A) = 67h
XRL     A, #78h     ; (A) = 67h XOR 78h = 1Fh
```

Ví dụ 2 :

3.5.4. Lệnh AND Bit

Cú pháp câu lệnh: **ANL C, <src-bit>**

Chức năng: Thực hiện phép tính logic AND cho các biến mức bit. Nếu giá trị logic của toán hạng nguồn bằng 0, thì cờ nhớ bị xoá. Dấu "/" đứng trước 1 toán hạng cho biết bit nguồn được lấy bù trước khi thực hiện AND với cờ nhớ nhưng ***giá trị của bit nguồn không bị thay đổi bởi thao tác lấy bù***. Lệnh này không làm ảnh hưởng tới trạng thái các cờ khác. Toán hạng nguồn chỉ được sử dụng kiểu định địa chỉ trực tiếp.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
ANL C, bit	2	2	10000010 bbbbbbbbb	(C)<-(C) AND (bit)
ANL C, /bit	2	2	10110000 bbbbbbbbb	(C)<-(C) AND NOT (bit)

Ví dụ 1: Thực hiện cổng AND Logic với 2 đầu vào (P1.0 và P1.1) và 1 đầu ra (P1.2)

```
MOV     C, P1.0     ; (C) = (P1.0)
ANL     C, P1.1     ; (C) = (P1.0) AND (P1.1)
MOV     P1.2, C     ; (P1.2) = (C)
```

Ví dụ 2 : Dùng VĐK chế tạo thành 2 con IC lôgic cổng OR 2 đầu vào

3.5.5 Lệnh OR Bít

Cú pháp câu lệnh: **ORL C, <src-bit>**

Chức năng: Thực hiện phép tính logic OR cho các biến mức bit. Nếu giá trị logic của toán hạng nguồn bằng 1, thì cờ nhớ được thiết lập. Dấu "/" đứng trước 1 toán hạng cho biết bit nguồn được lấy bù trước khi thực hiện OR với cờ nhớ nhưng giá trị của bit nguồn không bị thay đổi bởi thao tác lấy bù. Lệnh này không làm ảnh hưởng tới trạng thái các cờ khác.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
ORL C, bit	2	2	01110010 bbbbbbbbb	(C)<-(C) OR (bit)
ORL C, /bit	2	2	10100000 bbbbbbbbb	(C)<-(C) OR NOT (bit)

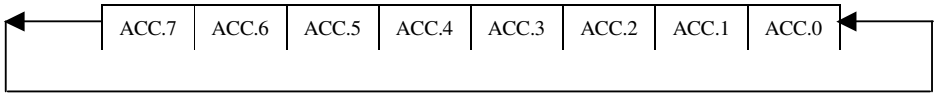
Ví dụ: Thực hiện cổng OR Logic với 2 đầu vào (P1.0 và P1.1) và 1 đầu ra (P1.2)

```
MOV     C, P1.0     ; (C) = (P1.0)
ORL     C, P1.1     ; (C) = (P1.0) OR (P1.1)
MOV     P1.2, C     ; (P1.2) = (C)
```

3.5.6 Lệnh dịch trái thanh ghi A

Cú pháp câu lệnh: **RL A**

Chức năng: Nội dung trong thanh ghi A được dịch trái 1 bit. Bit 7 được quay đến vị trí của bit 0. Các cờ không bị ảnh hưởng.



Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
RL A	1	1	00100011	(An+1) <- (An), với n = 0…6 (A0) <- (A7)

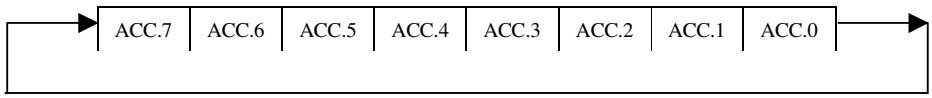
Ví dụ:

```
MOV     A, #11111110b     ; (A) = 11111110b
RL      A                 ; (A) = 11111101b
```

3.5.7 Lệnh dịch phải thanh ghi A

Cú pháp câu lệnh: **RR A**

Chức năng: Nội dung trong thanh ghi A được dịch sang phải 1 bit. Bit 0 được quay đến vị trí của bit 7. Các cờ không bị ảnh hưởng.



Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
RR A	1	1	00000011	(An) <- (An+1), với n = 0…6 (A7) <- (A0)

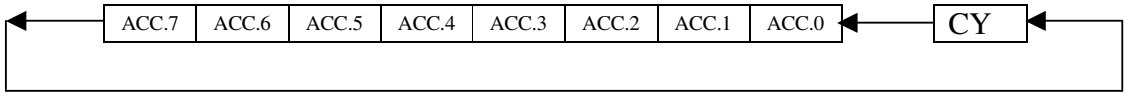
Ví dụ:

```
MOV     A, #01111111b     ; (A) = 01111111b
RR      A                 ; (A) = 10111111b
```

3.5.8 Lệnh dịch trái thanh ghi A qua cờ CY

Cú pháp câu lệnh: **RLC A**

Chức năng: Nội dung trong thanh ghi A và cờ nhớ cùng được dịch trái 1 bit. Bit 7 được chuyển đến cờ nhớ và trạng thái ban đầu của cờ nhớ được đưa về bit 0. Các cờ khác không bị ảnh hưởng.



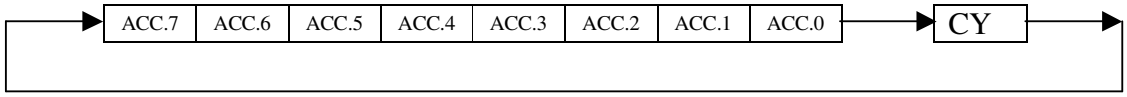
Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
RLC A	1	1	00110011	(An+1) <- (An), với n = 0…6 (A0) <- (C) còn (C) <- (A7)

Ví dụ:
```
MOV   A, #11111110b   ; (A) = 11111110b
MOV   C, ACC.0        ; (C) = 0
RLC   A               ; (A) = 11111100b  ;  và (C) = 1
```

3.5.9 Lệnh dịch phải thanh ghi A qua cờ CY

Cú pháp câu lệnh: **RRC A**

Chức năng: Nội dung trong thanh ghi A và cờ nhớ cùng được dịch phải 1 bit. Bit 0 được chuyển đến cờ nhớ và trạng thái ban đầu của cờ nhớ được đưa về bit 7. Các cờ khác không bị ảnh hưởng.



Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
RRC A	1	1	00010011	(An) <- (An+1), với n = 0…6 (A7) <- (C) còn (C) <- (A0)

Ví dụ:
```
MOV   A, #01111111b   ; (A) = 01111111b
MOV   C, ACC.7        ; (C) = 0
RRC   A               ; (A) = 00111111b ; và (C) = 1
```

3.5.10 Lệnh hoán chuyển nội dung 2 nửa byte của A

Cú pháp câu lệnh: **SWAP A**

Chức năng: hoán chuyển nội dung 2 nửa thấp và cao (mỗi nửa 4 bit) của thanh ghi A (các bit từ 0 đến 3 và các bit từ 4 đến 7). Thao tác này còn được hiểu là quay thanh ghi A 4 bit. Các cờ không bị ảnh hưởng.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
SWAP A	1	1	11000100	(A3-A0) <- (A7-A4)

Ví dụ: Tráo đổi nội dung 2 nửa thấp và cao của Byte 56h
```
MOV   A, #56h     ; (A) = 56h = 01010110b
SWAP  A           ; (A) = 65h = 01100101b
```

3.6. Nhóm lệnh rẽ nhánh

3.6.1. Lệnh gọi ngắn

Cú pháp câu lệnh: **ACALL addr11**

Chức năng: Gọi không điều kiện một chương trình con đặt tại địa chỉ được chỉ ra trong câu lệnh. Lệnh này tăng bộ đếm chương trình thêm 2 đơn vị để PC chứa địa chỉ của lệnh kế lệnh ACALL, sau đó cất nội dung 16 bit của PC vào ngăn xếp (byte thấp cất trước) và tăng con trỏ ngăn xếp lên 2 đơn vị. Địa chỉ đích sẽ được hình thành bằng cách ghép 5 bit cao của thanh ghi PC (sau khi được tăng), 3 bit cao của byte mã lệnh và byte thứ 2 của lệnh. Do đó chương trình con được gọi phải nằm trong đoạn 2 Kbyte của bộ nhớ chương trình chí it phải chứa lệnh đầu tiên của chương trình con này. Lệnh không làm ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
ACALL addr11	2	2	aaa10001 aaaaaaaa	(PC) <- (PC) + 2 (SP) <- (SP) + 1 ((SP)) <- (PC7-PC0) (SP) <- (SP) + 1 ((SP)) <- (PC15-PC8) (PC10-PC0) <- (page address)

Ví dụ:

```
Đ.Chỉ  Mã lệnh          Chương trình
                $INCLUDE(REG51.INC)
0000            ORG     0000H     ; Bắt đầu ch.trình tại đ/c 0000h
0000   758107   MOV     SP,#07H   ; (SP) = 07h
0003   7901     MOV     R1,#01H   ; (R1) = 01h
0005   1130     ACALL   GIAM_R1   ; Gọi ch.trình con GIAM_R1
                                  ; (PC) = (PC) +2 = 0007h
                                  ; (SP) = (SP) + 1 = 08h
                                  ; (08h) = (PCL) = 07h
                                  ; (SP) = (SP) + 1 = 09h
                                  ; (09h) = (PCH) = 00h
                                  ; (PC) = 0030h
0007   80FE   STOP:  JMP     STOP      ; Nhảy tại chỗ
0030           ORG     0030H
0030   19     GIAM_R1:DEC     R1        ; Giảm R1 đi 1 đơn vị
0031   22             RET               ; Quay trở lại ch.trình chính
                                  ; (PCH) = ((SP)) = (09h) = 00h
                                  ; (SP) = (SP) -  1 = 08h
                                  ; (PCL) = ((SP)) = (08h) = 07h
                                  ; (SP) = (SP) -  1 = 07h
                END
```

3.6.2. Lệnh gọi dài

Cú pháp câu lệnh: **LCALL addr16**

Chức năng: Gọi một chương trình con đặt tại địa chỉ được chỉ ra trong câu lệnh. Lệnh này tăng bộ đếm chương trình thêm 3 đơn vị để PC chứa địa chỉ của lệnh kế lệnh LCALL, sau đó cất nội dung 16 bit của PC vào ngăn xếp (byte thấp cất trước) và tăng con trỏ ngăn xếp lên 2 đơn vị. Tiếp theo nó sẽ chuyển byte thứ 2 và byte thứ 3 trong câu lệnh LCALL vào byte cao và byte thấp của PC. Việc thực thi chương trình tiếp tục với lệnh ở địa chỉ này. Như vậy chương trình con có thể bắt đầu bằng bất cứ nơi nào trong không gian bộ nhớ chương trình 64 Kbyte. Lệnh không làm ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
LCALL addr16	3	2	00010010 aaaaaaaa aaaaaaaa	(PC) <- (PC) + 3 (SP) <- (SP) + 1 ((SP)) <- (PC7-PC0) (SP) <- (SP) + 1 ((SP)) <- (PC15-PC8) (PC) <- addr15-addr0

Ví dụ:

```
      Đ.Chỉ  Mã lệnh                Chương trình
                                    $INCLUDE(REG51.INC)
      0000                          ORG  0000H      ; Bắt đầu ch.trình 0000h
0000  758107          MOV           SP,#07H         ; (SP) = 07h
0003  7901            MOV           R1,#01H         ; (R1) = 01h
0005  121130          LCALL         GIAM_R1         ; Gọi ch.trình con GIAM_R1
                                                    ; (PC) = (PC) +3 = 0008h
                                                    ; (SP) = (SP) + 1 = 08h
                                                    ; (08h) = (PCL) = 08h
                                                    ; (SP) = (SP) + 1 = 09h
                                                    ; (09h) = (PCH) = 00h
                                                    ; (PC) = 1130h
0008  80FE            STOP:         JMP     STOP
1130                                ORG     1130H
1130  19              GIAM_R1:      DEC     R1      ; Giảm R1 đi 1 đơn vị
1131  22                            RET             ; Quay trở lại ch.trình chính
                                                    ; (PCH) = ((SP)) = (09h) = 00h
                                                    ; (SP) = (SP) -  1 = 08h
                                                    ; (PCL) = ((SP)) = (08h) = 08h
                                                    ; (SP) = (SP) -  1 = 07h
                                    END             ; Kết thúc chương trình
```

3.6.3. Lệnh nhảy vô điều kiện

Cú pháp câu lệnh: **JMP rel hoặc LJMP Addr16**

Chức năng: Nhảy vô điều kiện đến một dịa chỉ nào đó

3.6.4. Lệnh nhảy gián tiếp

Cú pháp câu lệnh: **JMP @ A+DPTR**

Chức năng: Cộng giá trị không dấu 8 bit của thanh ghi A với con trỏ dữ liệu 16 bit và nạp kết quả vào bộ đếm chương trình, kết quả này chính là địa chỉ để nạp lệnh kế tiếp. Việc cộng 16 bit được thực hiện: Số nhớ từ 8 bit thấp được truyền đến tất cả các bit cao. Cả 2, thanh ghi A và DPTR đều không bị thay đổi. Lệnh này không ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
JMP @A+DPTR	1	2	01110011	(PC)<-(A)+(DPTR)

Ví dụ: Nhảy tới câu lệnh có địa chỉ 1002h trong chương trình

```
            ORG     0000h
            MOV     DPTR,#1000h         ; (DPTR) = 1000h
            MOV     A, #2               ; (A) = 2
            JMP     @A+DPTR             ; (PC) = 1002h
            ORG     1000h
1000h:      MOV     R1,#55h             ; (R1) = 55h
                                        ; Dòng lệnh này không được thực thi
1002h:      MOV     R1,#45h             ; (R1) = 45h
            END
```

3.6.5. Lệnh nhảy nếu cờ nhớ bằng 1

Cú pháp câu lệnh: **JC rel**

Chức năng: Nếu cờ CY có giá trị bằng 1 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh, ngược lại thì lệnh này được bỏ qua và nó sẽ tiếp tục thực hiện lệnh kế tiếp. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 2 của lệnh với nội dung trong PC (sau khi được tăng bởi 2). Lệnh không ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
JC rel	2	2	01000000 eeeeeeee	(PC)<-(PC)+2 Nếu (C)=1 thì: (PC)<- (PC) + rel

Ví dụ 1 :

```
X0      :   MOV   A ,# 00h
            MOV   PSW ,# 80h
            RLC   A
            JC    X0
            JMP   $
```

Ví dụ 2 : ……

3.6.6. Lệnh nhảy nếu cờ nhớ bằng 0

Cú pháp câu lệnh: **JNC rel**

Chức năng: Nếu cờ CY có giá trị bằng 0 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh, ngược lại thì lệnh này được bỏ qua và nó sẽ tiếp tục thực hiện lệnh kế tiếp. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 2 của lệnh với nội dung trong PC (sau khi được tăng bởi 2). Lệnh không ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
JNC rel	2	2	01010000 eeeeeeee	(PC)<-(PC)+2 Nếu (C)=0 thì: (PC)<- (PC) + rel

Ví dụ: tương tự như lệnh JC

3.6.7. Lệnh nhảy nếu một bít bằng 1

Cú pháp câu lệnh: **JB bit, rel**

Chức năng: Nếu bit đã cho có giá trị bằng 1 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh, ngược lại thì lệnh này được bỏ qua và nó sẽ tiếp tục thực hiện lệnh kế tiếp. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 3 của lệnh với nội dung trong PC (sau khi được tăng đến địa chỉ của byte đầu tiên của lệnh kế tiếp). Bit được kiểm tra không bị thay đổi, lệnh không ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
JB bit, rel	3	2	00100000 bbbbbbbbb eeeeeeee	(PC)<-(PC)+3 Nếu (bit)=1 thì: (PC)<-(PC) + rel

Ví dụ 1: Thay đổi nội dung của cổng P1 theo trạng thái của bit P0.0, như sau:

- Nếu P0.0 = 0 thì P1 = 00h
- Nếu P0.0 = 1 thì P1 = 0FFh

```
        ORG     0000h
Lap:    JB      P0.0, nhan1
        MOV     P1,#00h
        JMP     lap
Nhan1:  MOV     P1,#0FFh
        JMP     lap
        END
```

Ví dụ 2: …….

3.6.8. Lệnh nhảy nếu một bít bằng 0

Cú pháp câu lệnh: **JNB bit, rel**

Chức năng: Nếu bit đã cho có giá trị bằng 0 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh, ngược lại thì lệnh này được bỏ qua và nó sẽ tiếp tục thực hiện lệnh kế tiếp. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 3 của lệnh với nội dung trong PC (sau khi được tăng đến địa chỉ của byte đầu tiên của lệnh kế tiếp). Bit được kiểm tra không bị thay đổi, lệnh không ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
JNB bit, rel	3	2	00110000 bbbbbbbbb eeeeeeee	(PC)<-(PC)+3 Nếu (bit)=0 thì: (PC)<-(PC) + rel

Ví dụ: 0RG 0000h

X0: JNB P3.2 , Hiển thị số 0

Hiển thị số 1: ……..

JMP X0 .

3.6.9. Lệnh nhảy nếu nội dung A khác 0

Cú pháp câu lệnh: **JNZ rel**

Chức năng: Nếu có 1 hoặc nhiều bit của thanh ghi A có giá trị khác 0 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh, ngược lại nó sẽ tiếp tục thực hiện lệnh tiếp theo. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 2 của lệnh với nội dung trong PC (sau khi được tăng bởi 2). Lệnh không ảnh hưởng tới các cờ. Nội dung thanh ghi A không bị thay đổi.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
JNZ rel	2	2	01110000 eeeeeeee	(PC)<-(PC)+2 Nếu (A) < > 0 thì: (PC)<- (PC) + rel

Ví dụ:

```
        ORG     0000h   ; Bắt đầu chương trình tại đ/c 0000h
        MOV     A, #0   ; (A) = 00h
        MOV     R1,#5   ; (R1) = 5
        JNZ     nhan1   ; Nhảy tới 'nhan1' nếu (A) khác 0
        DEC     R1      ; Giảm R1 đi 1 đơn vị
        JMP     $       ; Nhảy tại chỗ
Nhan1:  INC     R1      ; Tăng nội dung R1 lên 1 đơn vị
        JMP     $       ; Nhảy tại chỗ
        END             ; Kết thúc chương trình
                        ; Kết quả: Do (A) khác 0 nên (R1) = 4.
```

3.6.10. Lệnh nhảy nếu nội dung A bằng 0

Cú pháp câu lệnh: JZ rel

Chức năng: Nếu tất cả các bit của thanh ghi A có giá trị bằng 0 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh, ngược lại nó sẽ tiếp tục thực hiện lệnh tiếp theo. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 2 của lệnh với nội dung trong PC (sau khi được tăng bởi 2). Lệnh không ảnh hưởng tới các cờ. Nội dung thanh ghi A không bị thay đổi.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
JZ rel	2	2	01100000 eeeeeeee	(PC)<-(PC)+2 Nếu (A)=0 thì: (PC)<- (PC) + rel

```
Ví dụ:      ORG     0000h     ; Bắt đầu chương trình tại đ/c 0000h
            MOV     A, #0     ; (A) = 00h
            MOV     R1,#5     ; (R1) = 5
            JZ      nhan1     ; Nhảy tới 'nhan1' nếu (A) = 0
            DEC     R1        ; Giảm R1 đi 1 đơn vị
            JMP     $         ; Nhảy tại chỗ
Nhan1:      INC     R1        ; Tăng nội dung R1 lên 1 đơn vị
            JMP     $         ; Nhảy tại chỗ
            END               ; Kết thúc chương trình
                              ; Kết quả: Do (A) = 0 nên (R1) = 6.
```

3.6.11. Lệnh nhảy nếu một bít bằng 1 và xóa ngay bít đó

Cú pháp câu lệnh: **JBC bit, rel**

Chức năng: Nếu bit đã cho có giá trị bằng 1 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh và xoá bit này, ngược lại nó sẽ tiếp tục thực hiện lệnh tiếp theo. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 3 của lệnh với nội dung trong PC (sau khi được tăng đến địa chỉ của byte đầu tiên của lệnh kế tiếp). Lệnh không ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
JBC bit, rel	3	2	00010000 bbbbbbbbb eeeeeeee	(PC)<-(PC)+3 Nếu (bit)=1 thì:bit = 0 (PC)<- (PC) + rel

```
Ví dụ:      ORG     0000h
            MOV     A,#10101010b      ; (A) = 0AAh
            JBC     ACC.1, nhan1      ; So sánh và nhảy
            MOV     P1,#02h           ; (P1) = 02h
            JMP     $                 ; Nhảy tại chỗ
Nhan1:      MOV     P1,#01h           ;  (P1) = 01h
            JMP     $                 ; Nhảy tại chỗ
            END                       ; Kết thúc chương trình
```

3.6.12. Lệnh giảm và nhảy nếu chưa bằng 0

Cú pháp câu lệnh: **DJNZ <byte>, <rel-address>**

Chức năng: Giảm ô nhớ đi 1 và nhảy tới địa chỉ cho bởi toán hạng thứ 2 nếu như kết quả khác 0. Nếu kết quả ban đầu là 00h thì nó chuyển qua 0FFh. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu trong byte lệnh cuối cùng với nội dung của PC (sau khi tăng PC tới byte đầu tiên của lệnh tiếp theo). Ngăn nhớ được giảm giá trị có thể là 1 thanh ghi hoặc 1 byte địa chỉ trực tiếp. Lệnh này không ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
DJNZ Rn, rel	2	2	11011rrr eeeeeeee	(PC)<-(PC)+2 (Rn)<- (Rn) - 1 Nếu (Rn) < > 0 thì: (PC) <- (PC) + rel
DJNZ Direct, rel	3	2	11010101 aaaaaaaa eeeeeeee	(PC)<-(PC)+2 (dir.)<- (dir.) - 1 Nếu (dir.) < > 0 thì: (PC) <- (PC) + rel

Ví dụ 1: Dịch trái thanh ghi A 7 lần.

```
        MOV     R1,#7       ; Đặt giá trị ban đầu cho biến đếm
        MOV     A,#01h      ; Đặt giá trị ban đầu cho A
Lap:    RL      A           ; Dịch trái A
        DJNZ    R1,lap ; Kiểm tra số lần dịch trái A
        END                 ; Kết thúc chương trình
```

Ví dụ 2: Viết chương trình tạo thời gian trễ 1s sử dụng lệnh DJNZ

3.6.13. Lệnh tạm ngưng hoạt động

Cú pháp câu lệnh: **NOP**

Chức năng: Tạm ngừng hoạt động khi có lệnh này và chương trình sẽ tiếp tục được thực hiện ở lệnh tiếp theo. Lệnh này không ảnh hưởng tới trạng thái các thanh ghi và các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
NOP	1	1	00000000	(PC)<-(PC)+2

Ví dụ: Tạo ra một xung có mức thấp tại bit P1.0 chính xác 4 chu kỳ máy.

```
MOV    P1,# 0FEh        ; (P1.0) = 0
NOP                     ; tạm ngừng hoạt động
NOP
NOP
MOV    P1,# 01h         ; (P1.0) = 1
```

3.6.14. Lệnh nhảy nếu 2 toán hạng không bằng nhau

Cú pháp câu lệnh: **CJNE <dest-byte>, <src-byte>, rel**

Chức năng: So sánh giá trị của 2 toán hạng đầu tiên, nếu 2 toán hạng không bằng nhau thì chương trình được rẽ nhánh. Địa chỉ đích rẽ nhánh được tính bằng cách cộng độ lệch tương đối (có dấu) trong byte sau cùng của lệnh với nội dung của PC (sau khi nội dung của PC được tăng đến địa chỉ bắt đầu của lệnh kế tiếp CJNZ). Cờ nhớ (CF) sẽ được thiết lập nếu như giá trị nguyên không dấu của toán hạng đích nhỏ hơn giá trị nguyên không dấu của toán hạng nguồn, ngược lại thì cờ này bị xoá. Lệnh này không làm thay đổi giá trị của các toán hạng

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
CJNE A, direct, rel	3	2	10110101 aaaaaaaa eeeeeeee	(PC)<-(PC)+3 Nếu (A) < > (dir.) thì: (PC)<- (PC) + offset Nếu (A) < (dir.) thì C =1 Ngược lại thì C =1
CJNE A, #data, rel	3	2	10110100 dddddddd eeeeeeee	(PC)<-(PC)+3 Nếu (A) < > #data thì: (PC)<- (PC) + offset Nếu (A) < #data thì C=1 Ngược lại: C =0
CJNE Rn, #data, rel	3	2	10111rrr dddddddd eeeeeeee	(PC)<-(PC)+3 Nếu (Rn)< >#data thì:(PC)<- (PC) + offset Nếu (Rn) < #data thì: C =1 Ngược lại: C =0

CJNE @Ri, #data, rel	3	2	1011011i dddddddd eeeeeeee	(PC)<-(PC)+3
				Nếu ((Ri))< >#data thì:
				(PC)<- (PC) + offset
				Nếu ((Ri)) < #data thì:
				C = 1
				Ngược lại: C =0

Ví dụ 1: Dịch trái thanh ghi A 7 lần.

```
        MOV     R1,#0       ; Đặt giá trị ban đầu cho biến đếm
        MOV     A,#01h      ; Đặt giá trị ban đầu cho A
Lap:    RL      A           ; Dịch trái A
        INC     R1          ; Tăng biến đếm lên 1 đơn vị
        CJNE    R1,#7,lap   ; Kiểm tra số lần dịch trái A
        END                 ; Kết thúc chương trình
```

Ví dụ 2: ……..

3.6.15. Lệnh kết thúc chương trình con

Cú pháp câu lệnh: **RET**

Chức năng: Trở về từ chương trình con. Lệnh này được thực hiện sau khi thực hiện xong lệnh ACALL hoặc LCALL. RET lấy lại byte cao và byte thấp của PC từ ngăn xếp, giảm SP đI 2 đơn vị. Chương trình tiếp tục được thực hiện với lệnh có địa chỉ ở trong PC. Các cờ không bị ảnh hưởng.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
RET	1	2	00100010	(PC15-PC8) <- ((SP)) (SP) <- (SP) - 1 (PC7-PC0) <- ((SP)) (SP) <- (SP) - 1

Ví dụ 1 : Xem ở các phần 8.6.1 và 8.6.2

3.6.16. Lệnh kết thúc chương trình ngắt

Cú pháp câu lệnh: **RETI**

Chức năng: Trở về từ chương trình con. RETI lấy lại byte cao và byte thấp của PC từ ngăn xếp, phục hồi logic ngắt để có thể nhận các ngắt khác có cùng mức ưu tiên ngắt với ngắt được xử lý, sau đó giảm SP đi 2 đơn vị. Chương trình tiếp tục được thực hiện với lệnh trước khi xử lý ngắt với địa chỉ ở trong PC. Các cờ không bị ảnh hưởng.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
RETI	1	2	00110010	(PC15-PC8) <- ((SP)) (SP) <- (SP) - 1 (PC7-PC0) <- ((SP)) (SP) <- (SP) - 1

Ví dụ: Tương tự như lệnh RET

3.7. Nhóm lệnh xử lý bít

3.7.1. Lệnh thiết lập bít

Cú pháp câu lệnh: **SETB bit**

Chức năng: Thiết lập bit được chỉ ra trong câu lệnh lên mức logic 1. Lệnh này có thể thao tác trên cờ nhớ, hoặc trên 1 bit bất kỳ được định địa chỉ trực tiếp. Lệnh không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
SETB C	1	1	11010011	(C) <- 1
SETB bit	2	1	11010010 bbbbbbbb	(bit) <- 1

Ví dụ:
```
Setb  C          ; thiết lập cờ nhớ
Setb  P0.0       ; Thiết lập chân P0.0
Setb  EA         ; Thiết lập bít cho phép ngắt toàn cục
```

3.7.2. Lệnh xóa bít

Cú pháp câu lệnh: **CLR bit**

Chức năng: Xoá bit được chỉ ra trong câu lệnh về 0. Lệnh này có thể thao tác trên cờ nhớ, hoặc trên 1 bit bất kỳ được định địa chỉ trực tiếp. Lệnh không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
CLR C	1	1	11000011	(C) <- 0
CLR bit	2	1	11000010 bbbbbbbb	(bit) <- 0

3.7.3. Lệnh lấy bù bít

Cú pháp câu lệnh: **CPL <bit>**

Chức năng: Lấy bù bit được chỉ ra trong câu lệnh. Một bit có giá trị 1 được đổi thành 0 và ngược lại. Lệnh này có thể thao tác trên cờ nhớ, hoặc trên 1 bit bất kỳ được định địa chỉ trực tiếp. Lệnh không làm ảnh hưởng tới trạng thaI các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
CPL C	1	1	10110011	(C) <- NOT (C)
CPL bit	2	1	10110010 bbbbbbbb	(bit) <- NOT (bit)

Ví dụ: Tạo ra 4 xung xuất hiện ở bit 7 của cổng P1. Mỗi một xung 3 chu kỳ máy.

```
        MOV   R1, #8
Lap:    CPL   P1.7
        DJNZ  R1, lap
```

3.7.4. Lệnh xóa thanh ghi A

Cú pháp câu lệnh: **CLR A**

Chức năng: Xoá nội dung thanh ghi tích luỹ về 0. Các cờ không bị ảnh hưởng.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
CLR A	1	1	11100100	(A) <- 0

Ví dụ:

```
MOV   A,#12h    ; (A) = 12h
CLR   A         ; (A) = 00h
```

3.7.5. Lệnh lấy bù thanh ghi A

Cú pháp câu lệnh: **CPL A**

Chức năng: Lấy bù tất cả các bit của thanh ghi A. Lệnh không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
CPL A	1	1	11110100	(A) <- NOT (A)

Ví dụ:

```
MOV   A,#01010101b    ; (A) = 01010101b
CPL   A               ; (A) = 10101010b
```

CHƯƠNG IV : Lập trình ứng dụng

4.1. Khái quát về lưu đồ thuật toán

4.1.1. Khái niệm

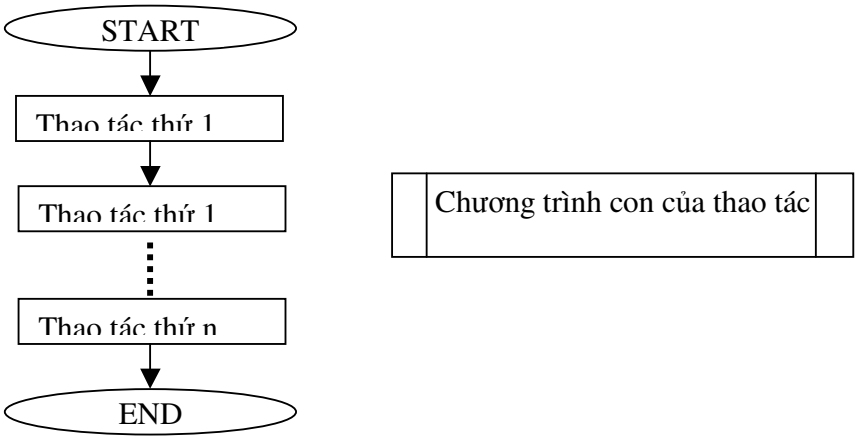
Trong kỹ thuật lập trình, người ta có nhiều các cách để giải quyết vấn đề bằng những thao tác cụ thể và được sắp xếp theo một trình tự nhất định gọi là thuật toán. Như vậy thuật toán là cái cốt lõi mang tính sáng tạo của việc lập trình, người lập trình viên phải biết sắp xếp cách tổ chức dữ liệu và xử lý theo một hình thức nào đó cho ngắn nhất và hiệu quả nhất.Trong thức tế, các lập trình viên sẽ có nhiều dạng lập trình khác nhau tùy thuộc vào:

- Công cụ sử dụng : Ví dụ lập trình trên Lô gô hay trên máy tính PC ...
- Mức độ chuyên sâu : Lập trình hệ thống hay lập trình ứng dụng ...
- Ngôn ngữ sử dụng : Ngôn ngữ cấp cao hay ngôn ngữ máy ....
- Lĩnh vực ứng dụng: Khoa học kỹ thuật, quản lý hay điều khiển vv...

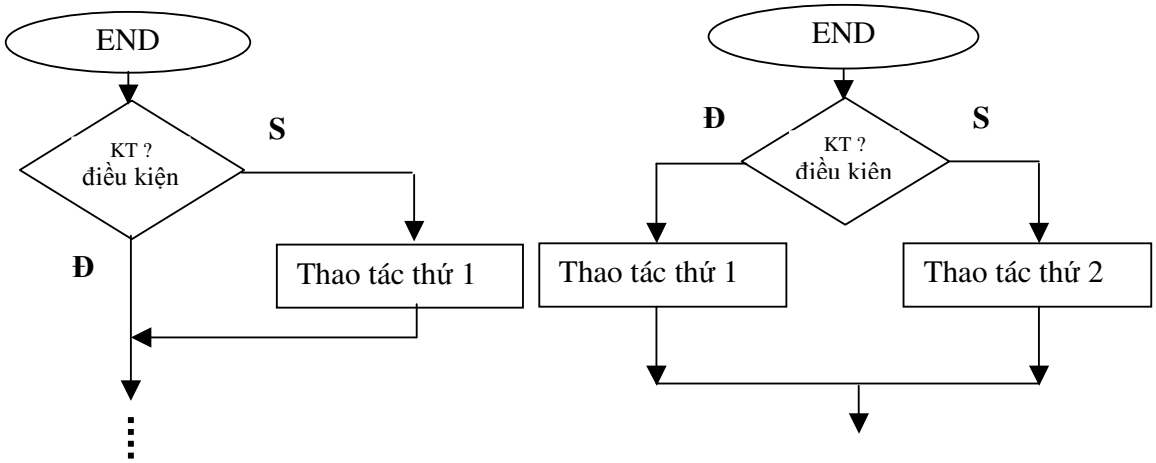
Như đã phát biểu ở trên thì trong thuật toán sẽ bao gồm hàng loạt các thao tác được sắp xếp theo trình tự, đương nhiên với những trình tự khác nhau nó sẽ cho kết quả là khác nhau. Cơ cấu thể hiện trình tự thực hiện của các thao tác gọi là cấu trúc điều khiển.

4.1.2. Các dạng cấu trúc thường sử dụng

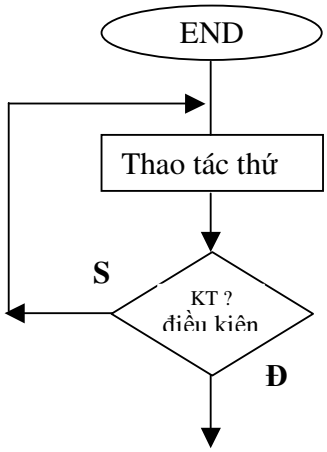
Cấu trúc tuần tự: Thể hiện quá trình thực hiện các thao tác một cách có sắp xếp theo 1 trình tự



Cấu trúc chọn lựa: Thể hiện quá trình thực hiện các thao tác theo một điều kiển nào đó cho trước. Nếu thỏa mãn điều kiện cho trước thì thực hiện công việc này và nếu không thỏa mãn thì không thực hiện hoặc thực hiện công việc khác



Cấu trúc lặp: Dạng cấu trúc này thể hiện sự lặp lại của chương trình



4.2. Trình tự các bước thiết kế một bài toán

Bước 1 : Phân tích tất cả các yêu cầu của bài toán
Bước 2 : Thiết kế phần cứng (mạch điều khiển)
Bước 3 : Lập lưu đồ thuật toán
Bước 4 : Viết chương trình điều khiển , hiệu chỉnh khi cần thiết

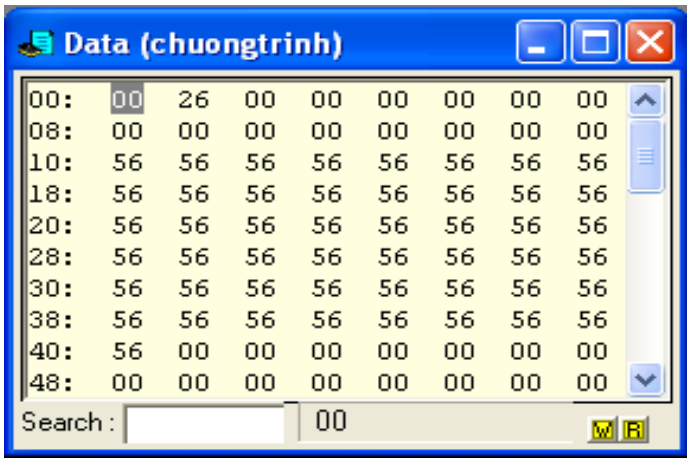
4.3. Lập trình với bộ nhớ

4.3.1. Di chuyển dữ liệu giữa bộ nhớ trong

Ví dụ : Chuyển 56h vào vùng nhớ có địa chỉ từ 10h đến 40h của RAM nội trú.

```
        ORG     0000H
        MOV     R1,#10h       ; Xác định địa chỉ ban đầu của vùng nhớ
Lap:    MOV     @R1,#56h      ; Chuyển dữ liệu vào từng địa chỉ vùng nhớ
        INC     R1            ; Tăng đại chỉ
        CJNE    R1,#41h,Lap   ; Kiểm tra điều kiện
        END                   ; Kết thúc chương trình
```

Kết quả:



Bài tập 1: Thực hiện Ví dụ 1 bằng phương pháp khác**.** (sử dụng DEC R1)

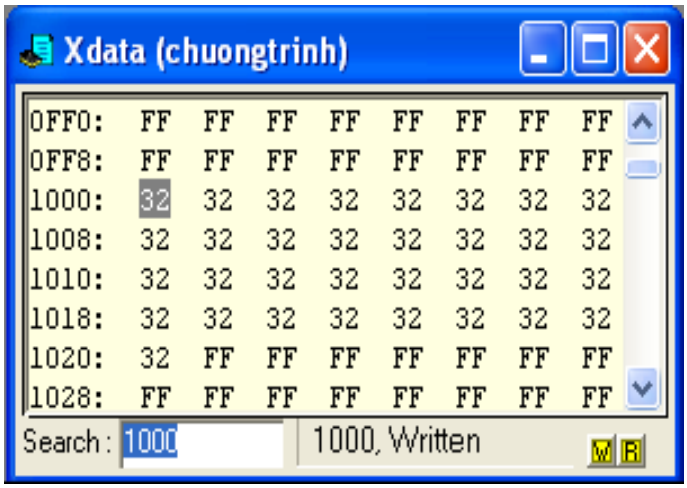
Bài tập 2: Viết chương trình chuyển nội dung của vùng nhớ có địa chỉ từ 10h đến 30h vào vùng nhớ có địa chỉ bắt đầu từ 40h của RAM nội trú.

4.3.2. Di chuyển dữ liệu giữa bộ nhớ trong với bộ nhớ ngoài và ngược lại

Ví dụ : Chuyển 32h vào vùng nhớ có địa chỉ từ 1000h đến 1020h

```
        ORG     0000H
        MOV     A,#32h
        MOV     DPTR,#1000h
Lap:    MOVX    @DPTR,A
        INC     DPL
        MOV     R1,DPL
        CJNE    R1,#21h,Lap
        END
```

Kết quả:



Bài tập 1: Viết chương trình chuyển nội dung tại địa chỉ 1010h của RAM ngoại trú vào vùng nhớ có địa chỉ từ 20h đến 60h của RAM nội trú.

Bài tập 2: Viết chương trình chuyển nội dung của vùng nhớ có địa chỉ từ 20h đến 50h của RAM nội trú vào vùng nhớ có địa chỉ bắt đầu là 1000h của RAM ngoại trú. (Lưu ý: có thể thực hiện bằng nhiều phương pháp khác nhau)

4.3.3. Di chuyển dữ liệu giữa bộ nhớ ngoài

Bài tập: Viết chương trình chuyển nội dung của vùng nhớ có địa chỉ từ 1000h đến 1020h vào vùng nhớ có địa chỉ từ 1040h đến 1060h của RAM ngoại trú. Cho biết có bao nhiêu phương pháp để thực hiện bài tập này ?

4.4. Lập trình điều khiển dữ liệu trên các cổng

4.4.1. Lập trình điều khiển LED đơn trên các cổng_ (10 bài toán cụ thể)

- Điều khiển 8 LED đơn hiển thị các trạng thái khác nhau
- Điều khiển 16 LED đơn hiển thị các trạng thái khác nhau

4.4.2. Lập trình điều khiển LED 7 thanh trên các cổng _(hiển thị số đếm, ký tự...)

- Điều khiển dãy LED 7 thanh hiển thị dứng yên
- Điều khiển dãy LED 7 thanh hiển thị dịch chuyển

4.4.3. Lập trình điều khiển Ma trận LED_ (các bài toán quang báo)

- Điều khiển dãy LED 7 thanh hiển thị dứng yên
- Điều khiển dãy LED 7 thanh hiển thị dịch chuyển

4.5. Lập trình với hàm thời gian

4.5.1. Lập trình với hàm trễ dùng vòng lập DJNZ

Ví dụ: Viết hàm chờ 1 ms

```
Cho_1ms:  Mov   r2 ,#249      ; 2µs
Lap:      Nop                 ; 1µs
          Nop                 ; 1µs
          Djnz  r2, lap       ; 2µs   ; 249*(1µs +1µs +2µs) = 996 µs
          Ret                 ; 2µs
                              ; Tổng thời gian là: (2+996+2) µs = 1ms
```

Bài tập: Hãy xác định thời gian chờ của chương trình sau.

```
wait:  Mov   r2,#10
w3:    Mov   r1,#200
w2:    Mov   r0,#228
w1:    Djnz  r0,w1
       Djnz  r1,w2
       Djnz  r2,w3
       Ret
```

4.5.2. Lập trình với hàm trễ dùng các Timer

Ví dụ1: Viết chương trình tạo thời gian trễ 200 chu kỳ máy dùng Timer1

Trước hết chúng ta phải phân tích xem bài toán yêu cầu những gì để xác định các chế độ hoạt động cho phù hợp. Từ đó xác định được thông số cho thanh ghi TMOD và các thông số cần nạp cho thanh ghi Timer (TH_X và TL_X)

```
Delay: Mov Tmod , # 20h       ; chế độ 2 Timer1
       Mov Th1, # 56          ; lấy 256 – 200 = 56
       Setb Tr1               ; cho phép Timer hoạt động
       Jnb Tf1 , $            ; chờ đợi cờ tràn thiết lập
       Clr  Tf1,
       Clr  Tr1
       Ret                    ;Kêt thúc chương trình con
```

Ví dụ2: Viết hàm chờ khoảng 1 s sử dụng Timer 0 chế độ 1

```
       ; 50000  = C350h
       ; FFFFh-C350h= 3CAFh
       ; Giá trị ban đầu là 3CAFh+1=3CB0h
wait:  Mov   r1,#20
       Mov   TMOD,#01h       ; Chọn Timer 0, chế độ 1
lap:   Clr   TF0             ; Xóa cờ ngắt Timer 0
       Mov   TL0, #0B0h      ; Đặt giá trị ban đầu cho TL0
       Mov   TH0, #03Ch      ; Đặt giá trị ban đầu cho TH0
       Setb  TR0             ; Khởi động Timer 0
       Jnb   TF0,$           ; Kiểm tra cờ tràn Timer 0
       Djnz  r1,lap          ; Đếm đủ rồi thì thoát
       Ret
```

Bài tập 1: Viết chương trình tạo thời gian trễ 500 chu kỳ máy dùng Timer0

Bài tập 2: Viết chương trình tạo thời gian trễ 65536 chu kỳ máy dùng Timer1

4.5.3. Lập trình tạo tín hiệu xung, điều chỉnh độ rộng xung, tạo tín hiệu âm thanh...

Ví dụ : Viết chương trình tạo sóng vuông có tần số 1 KHz tại chân P1.7

```
                ;65535(0FFFFh) – 500 (01F4h) = 65035 (0FE0Bh)
        Mov   TMOD,#10h     ; Chọn Timer 1, chế độ 1
Lap:    Mov   TH1,#0FEh     ; TH1 chứa giá trị 0FEh
        Mov   TL1,#0Bh      ; TL1 chứa giá trị 0Bh
        Setb  TR1           ; Bộ Timer 1 hoạt động
Cho:    Jnb   TF1,cho       ; Chờ tràn
        Clr   TR1           ; Bộ Timer 1 ngừng hoạt động
        Clr   TF1           ; Xoá cờ tràn
        Cpl   P1.7          ; Đổi trạng thái bit P1.7
        Sjmp  Lap           ; Lặp lại
```

Bài tập : Hãy viết chương trình tạo sóng vuông có tần số 10 KHz tại chân P1.0.

4.6. Lập trình với các ngắt

4.6.1. Lập trình với ngắt Timer

Ví dụ 7: Tạo tín hiệu dưới dạng xung vuông có tần số 10 KHz tại chân P1.0 của VĐK. Sử dụng ngắt Timer 0.

```
        Org     0000h
        Ljmp    main
        Org     000Bh
        Ljmp    Timer0
Main:
        Mov     TMOD,#02h       ; Lựa chọn Timer 0 , chế độ 2.
        Mov     TL0,#-50        ; Xác định giá trị ban đầu cho TL0.
        Mov     TH0,#-50        ; Xác định giá trị ban đầu cho TH0.
        Setb    TR0             ; Khởi tạo Timer 0.
        Mov     IE,#82h         ; Khởi tạo ngắt Timer 0.
        Jmp     $
Timer0:                         ; Chương trình con phục vụ ngắt T0.
        Cpl     P1.0            ; Đảo trạng thái để tạo xung.
        Reti                    ; Thoát khỏi ch/trình con phục vụ ngắt.
```

Bài tập: Tạo tín hiệu dưới dạng xung vuông có tần số 5 KHz tại chân P1.0 của VĐK. Sử dụng ngắt Timer 1.

4.6.2. Lập trình với ngắt ngoài (bài toán lập trình điều khiển Rô bôt)

Ví dụ 6: Chương trình đếm sản phẩm với số lượng tối đa là 255. Lưu kết quả vào thanh ghi R0. Cảm biến được kết nối với chân P3.3 của VĐK.

```
        Org     0000h           ; Bắt đầu chương trình chính
        Ljmp    Main
        Org     0013h           ; Sử dụng ngắt ngoài 1
        Ljmp    Ngat1
```

```
        Org     0030h
Main:
        Setb    EX1             ; Thiết lập các ĐK để ngắt hoạt động
        Setb    IT1
        Setb    EA
        Mov     R0,#0           ; Đặt giá trị ban đầu cho biến đếm
        Jmp     $
Ngat1:
        Inc     R0              ; Tăng biến đếm số lượng sản phẩm
        Cjne    R0,#255,Tiep    ; So sánh biến đếm với giá trị tối đa
        Clr     EX1             ; Ngắt ngừng hoạt động
Tiep:   Reti                    ; Thoát khỏi ch / trình con phục vụ ngắt
        End
```

Bài tập: Viết chương trình đếm sản phẩm với số lượng tối đa là 250, dùng ngắt INT0 . Hiển thị kết quả trên 3 LED 7 thanh. Trong đó:

- LED 7 là loại chung Anốt
- 3 LED 7 thanh được kết nối với các cổng P0, P1, P2.

Bài toán: Lựa chọn phần cứng, lập lưu đồ thuật toán và viết chương trình điều khiển 2 động cơ một chiều (M1, M2) theo yêu cầu: (K1, K2 là các công tắc thường mở)

K1	**K2**	**Chức năng**		**Chế độ**
		M1	M2	
Mở	Mở	Dừng	Dừng	1
Đóng	Mở	Tiến	Dừng	2
Mở	Đóng	Dừng	Tiến	3
Đóng	Đóng	Tiến	Tiến	4

4.6.3. Lập trình với ngắt nối tiếp, truyền thông , (giao tiếp VĐK với PC)

4.7. Hướng dẫn Bài tập lớn

Bài tập 1: Thiết kế và lập trình hệ thống điều khiển động cơ DC/động cơ bước.

Bài tập 2: Thiết kế và lập trình hệ thống quang báo dùng Led 7 thanh/ Led.Matrận

Bài tập 3: Thiết kế và lập trình hệ thống điều khiển tín hiệu giao thông.

Bài tập 4: Thiết kế và lập trình hệ thống đếm sản phẩm sử dụng ngắt ngoài.

Bài tập 5: Thiết kế và lập trình hệ thống điều khiển đếm và phân loại SP.

Bài tập 6: Thiết kế và lập trình hệ thống tự động điều khiển nhiệt độ.

4.8. Hệ thống bài thực hành Vi điều khiển :

TÀI LIỆU THAM KHẢO

[1]. Đỗ Xuân Tiến

Kỹ thuật Vi xử lý và lập trình Assembly cho hệ VXL_ NXBKH&KT 2001

[2]. Tống Văn On

Vi điều khiển 8051 _ NXB Lao động & xã hội 2001

[3]. Văn Thế Minh

Kỹ thuật Vi xử lý _ NXBGD 1997

[4]. Ngô Diên Tập

Vi xử lý trong đo lường và điều khiển – NXB KHKT – 2000

[5]. Hệ thống bài thí nghiệm VXL_VĐK

Bộ môn KTĐT _ Đại học bách khoa TPHCM

[6]. Bạch Hưng Trường

Giáo trình kỹ thuật Vi điều khiển_ Đại học SPKT Hưng Yên

[7]. Nguyễn Mạnh Giang

Cấu trúc, lập trình ghép nối và ứng dụng của VĐK – NXB LĐ&XH - 2005

[8]. Nguyễn Tăng Cường

Cấu trúc và lập trình họ vi điều khiển 8051 – NXB KH&KT - 2004

Dangvankhanh35@yahoo.com

N99KDD@gmail.com

DD: 0904346773

Giáo trình

Kỹ thuật Vi Điều Khiển

MỤC LỤC

LỜI GIỚI THIỆU

Khoa học kỹ thuật đang ngày càng phát triển rất mạnh mẽ, các công nghệ mới thuộc các lĩnh vực khác nhau cũng nhờ đó đã ra đời nhằm đáp ứng nhu cầu của xã hội và kỹ thuật ***Vi điều khiển*** cũng nằm

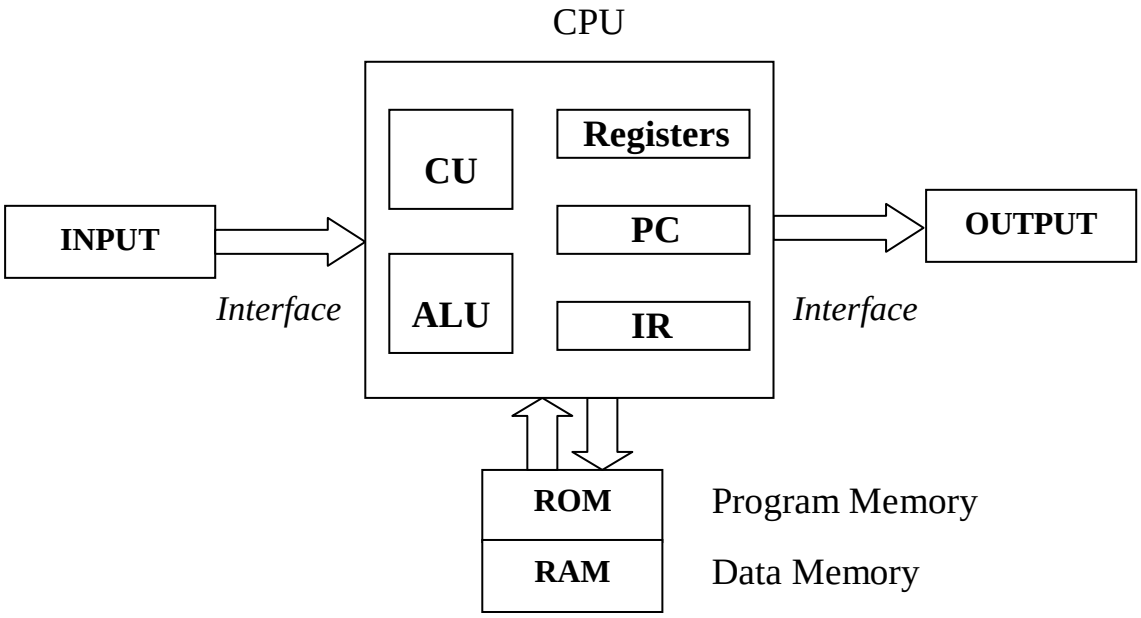
trong số đó. Hiện nay kỹ thuật Vi xử lý đã được giảng dạy rộng rãi ở các trường Đại học và Cao đẳng trong cả nước, tuy nhiên lĩnh vực mới ***Vi điều khiển*** vẫn đang còn rất mới mẻ, và những ứng dụng của nó vẫn chưa được khai thác triệt để trong các hệ thống điều khiển, đo lường và điều chỉnh của các dây chuyền công nghiệp. Qua quá trình tham gia giảng dạy tại trường Đại học SPKT Hưng yên và thời gian học tập nâng cao ở CHLB Đức, tác giả đã tập trung nghiên cứu và biên soạn ***giáo trình kỹ thuật Vi điều khiển*** nhằm phục vụ công việc giảng dạy lĩnh vực này tại trường. Toàn bộ nội dung giáo trình được chia làm 2 phần. Phần 1 bao gồm các kiến thức cơ bản về phần cứng và các tập lệnh của họ Vi điều khiển 80C51/ AT89C51. Ở phần 2 tác giả tập trung trình bày phần cứng họ Vi điều khiển 80C52/ AT89S8252 và kỹ thuật lập trình bằng hợp ngữ. Đối tượng của quyển giáo trình này là các sinh viên ngành Điện, Điện tử, Cơ điện tử, Công nghệ thông tin. Tuy nhiên để tiếp thu tốt nội dung từ quyển giáo trình này, người học cần có kiến thức về kỹ thuật số, kỹ thuật mạch điện tử và đã biết qua một ngôn ngữ lập trình cấp cao như Pascal, C…

Mặc dù đã rất cố gắng trong quá trình biên soạn, nhưng do trình độ và thời gian còn bị hạn chế nên chắc chắn quyển giáo trình này không tránh khỏi những thiếu sót, rất mong nhận được những ý kiến đóng góp, phê bình của bạn đọc.

Hưng yên, tháng 10 năm 2003

Tác giả

***Chương 1*. KIẾN TRÚC HỆ VXL**



Hình1.1. Khái quát chung về hệ VXL

1.1. CPU(Central Processing Unit):

Bộ vi xử lý (VXL) là thuật ngữ được bắt nguồn từ tên gọi tiếng Anh là *MICROPROCESSOR* (MP) hoặc *CENTRAL PROCESING UNIT* (CPU). Trong mỗi hệ VXL, CPU luôn là thành phần quan trọng nhất, nó quản lý tất cả các hoạt động của hệ VXL và thực hiện các thao tác trên dữ liệu. Hầu hết các CPU chỉ bao gồm một tập các ***mạch Logic*** thực hiện liên tục 2 thao tác: ***tìm nạp lệnh và thực thi lệnh***. CPU có khả năng hiểu và thực thi các lệnh dựa trên một tập các ***mã nhị phân***, trong đó mỗi một mã thực hiện một thao tác nào đó. Các lệnh này bao gồm:

- Nhóm lệnh ***di chuyển dữ liệu*** (Mov,...).
- Nhóm lệnh ***số học*** (Mul, Div, Add, Subb,...).
- Nhóm lệnh ***Logic*** (ANL, ORL, CPL, XRL,...).
- Nhóm lệnh ***rẽ nhánh chương trình*** (Jmp, Call, ...).
- Nhóm lệnh ***điều khiển biến Logic*** (Setb, Clr,...)....

Các nhóm lệnh trên được biểu thị bởi 1 tập các mã nhị phân và được gọi là tập lệnh.

Mỗi bộ VXL (CPU) thường bao gồm:

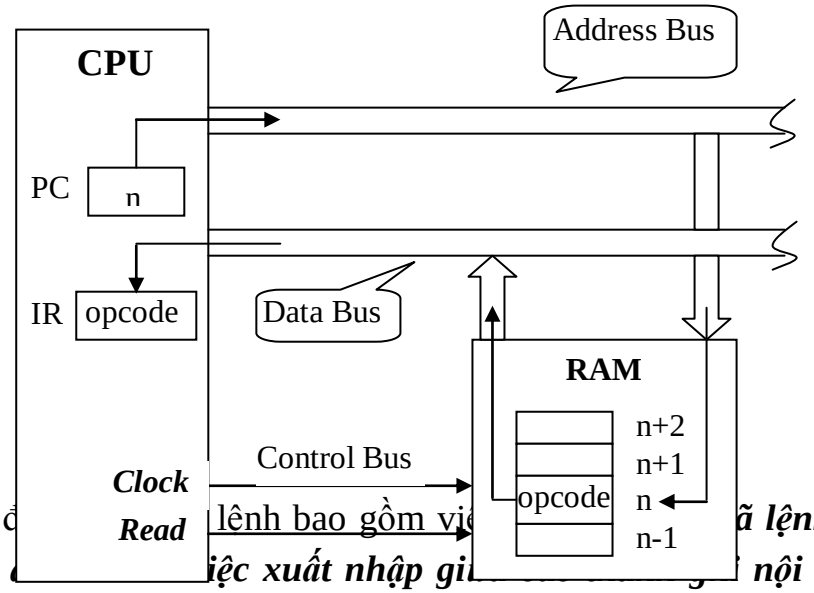
- Các thanh ghi nội (Registers): có nhiệm vụ ***lưu giữ tạm thời*** các thông tin, dữ liệu.

- Đơn vị số học logic (Arithmetic Logic Unit - ALU): ***Thực hiện các thao tác*** trên các thông tin hay dữ liệu đã được lưu giữ tạm thời trong thanh ghi nội.
- Đơn vị điều khiển (Control Unit - CU): Có nhiệm vụ giải mã lệnh và ***điều khiển*** việc thực hiện các thao tác, đồng thời thiết lập các hoạt động cần thiết để thực hiện các thao tác đó.
- Thanh ghi lệnh (Instruction Register - IR): ***Lưu giữ mã nhị phân*** của lệnh để được thực thi.
- Bộ đếm chương trình (Program Counter - PC): ***Lưu giữ địa chỉ của lệnh kế tiếp*** trong bộ nhớ cần được thực thi.

1.2. Quá trình tìm nạp lệnh và thực thi lệnh của CPU:

+ Việc tìm nạp một lệnh từ RAM hệ thống là một trong những thao tác cơ bản nhất mà CPU thực hiện. Quá trình tìm nạp được thực hiện theo các bước sau:

- Nội dung của PC được gửi lên *kênh địa chỉ*.
- Tín hiệu điều khiển READ được xác lập (chuyển sang trạng thái tích cực).
- Dữ liệu (mã lệnh) được đọc từ RAM và gửi đi trên *kênh dữ liệu.*
- Mã lệnh được chốt vào thanh ghi lệnh bên trong CPU.
- Nội dung của PC được tăng lên để chuẩn bị tìm nạp lệnh kế tiếp từ bộ nhớ.



+ Giai đ lệnh bao gồm vi ***ã lệnh*** và ***tạo ra các tín hiệu để ệc xuất nhập gi nội với ALU***, đồng thời thông

Hình 1.2. Hoạt động của Bus cho chu kỳ tìm nạp lệnh

1.3. Bộ nhớ trung tâm của hệ Vi xử lý:

Bộ nhớ trung tâm là bộ phận rất quan trọng đối với mỗi hệ VXL, nó là tập hợp các thanh ghi thông tin với số lượng lớn. Chức năng cơ bản của bộ nhớ là để trao đổi và lưu trữ thông tin.

1.3.1. Bộ nhớ chỉ đọc (Read Only Memory - ROM):

1.3.1.1. ROM cơ bản:

ROM dùng để lưu trữ chương trình điều hành (Monitor) của hệ VXL. Chương trình này sẽ quy định mọi hoạt động của hệ VXL. Bộ VXL sẽ căn cứ vào các lệnh chứa trong chương trình để điều khiển hệ VXL thực hiện các chức năng, nhiệm vụ được ấn định trong lệnh. Nói cách khác, hệ VXL sẽ thực hiện một cách trung thực thuật toán mà người thiết kế phần mềm đã xây dựng và cài đặt vào ROM của hệ.

Ngoài ra, ROM trong hệ VXL còn dùng để lưu trữ các bảng biểu, tham số của hệ thống mà trong quá trình hoạt động không được thay đổi như: bảng địa chỉ cổng giao tiếp, các bảng tra cứu số liệu, các bộ mã cần sử dụng trong hệ.

ROM cũng được quản lý theo phương thức ma trận điểm, nó có nhiều chủng loại khác nhau: ROM, PROM, EPROM, EEPROM,…

ROM là bộ nhớ cố định có cấu trúc đơn giản nhất. Nội dung của nó do nhà sản xuất chế tạo, người sử dụng không thể thay đổi nội dung này được nữa.

1.3.1.2. PROM (Programmable ROM - ROM có khả năng lập trình được):

Đặc điểm chung: Nội dung của PROM do nhà sản xuất hoặc người thiết kế hệ VXL nạp vào nhưng chỉ đươc 1 lần. Sau khi nạp xong nội dung này không thể thay đổi được nữa.

1.3.1.3. EPROM (Eraseable PROM – ROM nạp/xoá được nhiều lần):

EPROM là bộ nhớ cố định có cấu trúc đặc biệt. Nội dung của nó do nhà sản xuất hay người thiết kế hệ VXL nạp vào và có thể nạp/xoá nhiều lần. Người ta tạo ra 1 bit thông tin trong EPROM dựa trên nguyên tắc làm việc của Transistor trường có cực cửa cách ly kênh cảm ứng (MOSFET kênh cảm ứng).

1.3.1.4. EEPROM (Electrical EPROM – ROM có khả năng lập trình và xoá được bằng điện).

1.3.2. Bộ nhớ truy cập ngẫu nhiên (Random Acess Memory - RAM):

RAM là bộ nhớ có thể ghi và đọc được, thông tin trên RAM sẽ bị mất khi mất nguồn cung cấp. Theo phương thức lưu trữ thông tin, RAM được chia thành 2 loại cơ bản: RAM tĩnh và RAM động.

RAM tĩnh: Có thể lưu trữ thông tin lâu tuỳ ý miễn là được cung cấp điện năng - tất cả các loại phần tử nhớ bằng Trigơ đều thuộc loại này.

RAM động: Chỉ lưu được thông tin trong 1 khoảng thời gian nhất định. Muốn kéo dài thời gian này cần có phương thức làm tươi lại thông tin trong phần tử nhớ RAM. Phần tử nhớ của RAM động đơn giản nhất là một linh kiện điện dung - tụ diện. Sử dụng RAM động có phức tạp nhưng về cấu trúc nhớ lại đơn giản, tiêu tốn ít năng lượng, tăng mật độ bộ nhớ và đôi khi còn làm tăng cả tốc độ làm việc của bộ nhớ.

Cấu trúc mạch điện của các bộ nhớ RAM rất đa dạng cả về công nghệ chế tạo chúng (TTL, MOS,…) và các yêu cầu sử dụng chúng như các yêu cầu về ghép nối, tốc độ làm việc, mật độ linh kiện và dung lương cần thiết…

1.4. Các thiết bị xuất/nhập:

Các thiết bị xuất/nhập hay các thiết bị ngoại vi kết hợp với các mạch giao tiếp (Interface) sẽ tạo ra các đường truyền thông giữa hệ VXL với thế giới bên ngoài. Tuy nhiên để trao đổi thông tin giữa hệ VXL với các thiết bị ngoại vi, cần có các phương pháp điều khiển thích hợp như:

- Điều khiển vào/ra bằng chương trình.
- Điều khiển vào/ra bằng ngắt.
- Điều khiển vào/ra bằng phần cứng.

Nội dung này sẽ được xét kỹ ở các chương sau.

1.5. Cấu trúc kênh chung của hệ VXL:

Kênh (Bus) là tập hợp các đường thông tin có cùng mục đích. Để CPU có thể giao tiếp được với các bộ phận khác trong hệ VXL theo yêu cầu, mỗi hệ VXL cần sử dụng 3 kênh như sau:

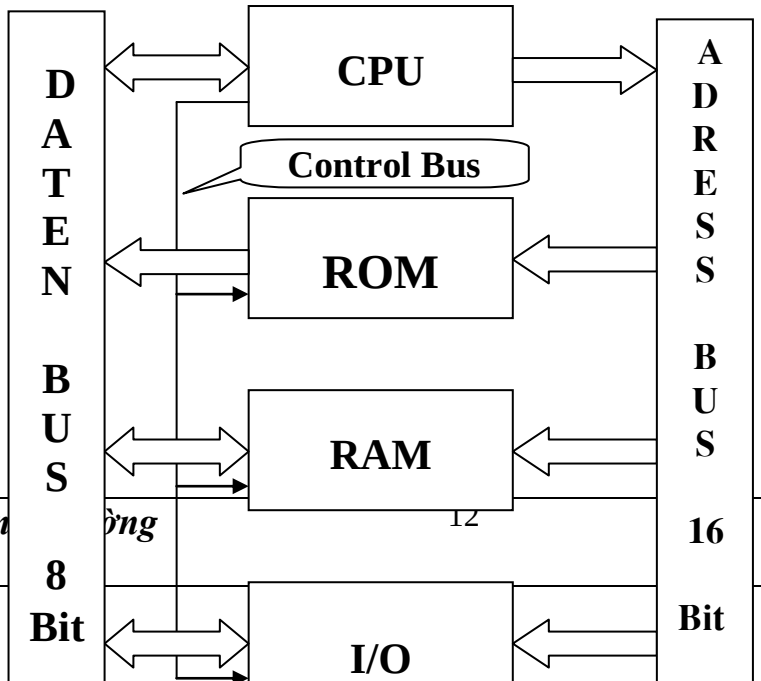
- Kênh địa chỉ (Adress Bus).
- Kênh dữ liệu (Daten Bus).
- Kênh điều khiển (Control Bus).

Để thực hiện thao tác đọc hoặc ghi, CPU xác định rõ vị trí (địa chỉ) của dữ liệu (hoặc lệnh) bằng cách đặt địa chỉ đó lên kênh địa chỉ, sau đó kích hoạt tín hiệu ***Read*** hoặc ***Write*** trên kênh điều khiển để chỉ ra thao tác là ***đọc*** hay ***ghi.***

Nếu kích hoạt tín hiệu điều khiển Read, thao tác đọc lấy 1 byte dữ liệu từ bộ nhớ ở vị trí đã xác định và đặt byte này lên kênh dữ liệu. CPU sẽ đọc dữ liệu và cất dữ liệu vào 1 trong các thanh ghi nội của CPU.

Nếu kích hoạt tín hiệu điều khiển Write, CPU sẽ thực hiện thao tác ghi bằng cách xuất dữ liệu lên kênh dữ liệu. Nhờ vào tín hiệu điều khiển, bộ nhớ nhận biết được đây là thao tác ghi và lưu dữ liệu vào vị trí đã được xác định.

Kênh dữ liệu cho phép trao đổi thông tin giữa CPU và bộ nhớ, cũng như giữa CPU với thiết bị ngoại vi. Thông thường các hệ VXL dành hầu hết thời gian cho việc di chuyển dữ liệu, đa số các thao tác di chuyển dữ liệu xảy ra giữa 1 thanh ghi của CPU với ROM và RAM ngoài. Do đó độ lớn của kênh dữ liệu ảnh hưởng rất lớn tới hiệu suất của hệ VXL. Nếu bộ nhớ của hệ thống rất lớn và CPU có khả năng tính toán cao, nhưng việc truy xuất dữ liệu – di chuyển dữ liệu giữa bộ nhớ và CPU thông qua kênh dữ liệu lại bị nghẽn thì hiện tượng "nghẽn cổ chai" này chính là hậu quả của độ rộng kênh dữ liệu không đủ lớn. Để khắc phục hiện tượng này, cần tăng đường tín hiệu cho kênh dữ liệu.



Như ở hình 1.3, kênh dữ liệu là kênh 2 chiều, còn kênh địa chỉ là kênh 1 chiều. Các thông tin về địa chỉ luôn được cung cấp bởi CPU, trong khi các dữ liệu di chuyển theo cả 2 hướng tuỳ thuộc vào thao tác thực hiện là đọc hay ghi. Thuật ngữ "*dữ liệu*" được sử dụng theo nghĩa tổng quát: "*thông tin*" di chuyển trên kênh dữ liệu có thể là ***lệnh*** của chương trình, ***địa chỉ*** theo sau lệnh hoặc ***dữ liệu*** được sử dụng bởi chương trình.

Kênh điều khiển là tập hợp các tín hiệu, mỗi tín hiệu có một vai trò riêng trong việc điều khiển có trật tự hoạt động của hệ thống. Các tín hiệu điều khiển được cung cấp bởi CPU để đồng bộ việc di chuyển thông tin trên các kênh địa chỉ và dữ liệu. Các bộ VXL thường có 3 tín hiệu điều khiển: ***Read, Write, Clock***. Tuy nhiên tuỳ vào yêu cầu cụ thể cũng như cấu trúc phần cứng của từng hệ VXL mà số lượng tín hiệu điều khiển có thể khác nhau.

Chương 2:
BỘ VI ĐIỀU KHIỂN AT89C51 (80C51)

2.1. Giới thiệu chung:

Vi điều khiển (VĐK) là một "hệ" Vi xử lý (VXL) được tổ chức trong một chip. Nó bao gồm:

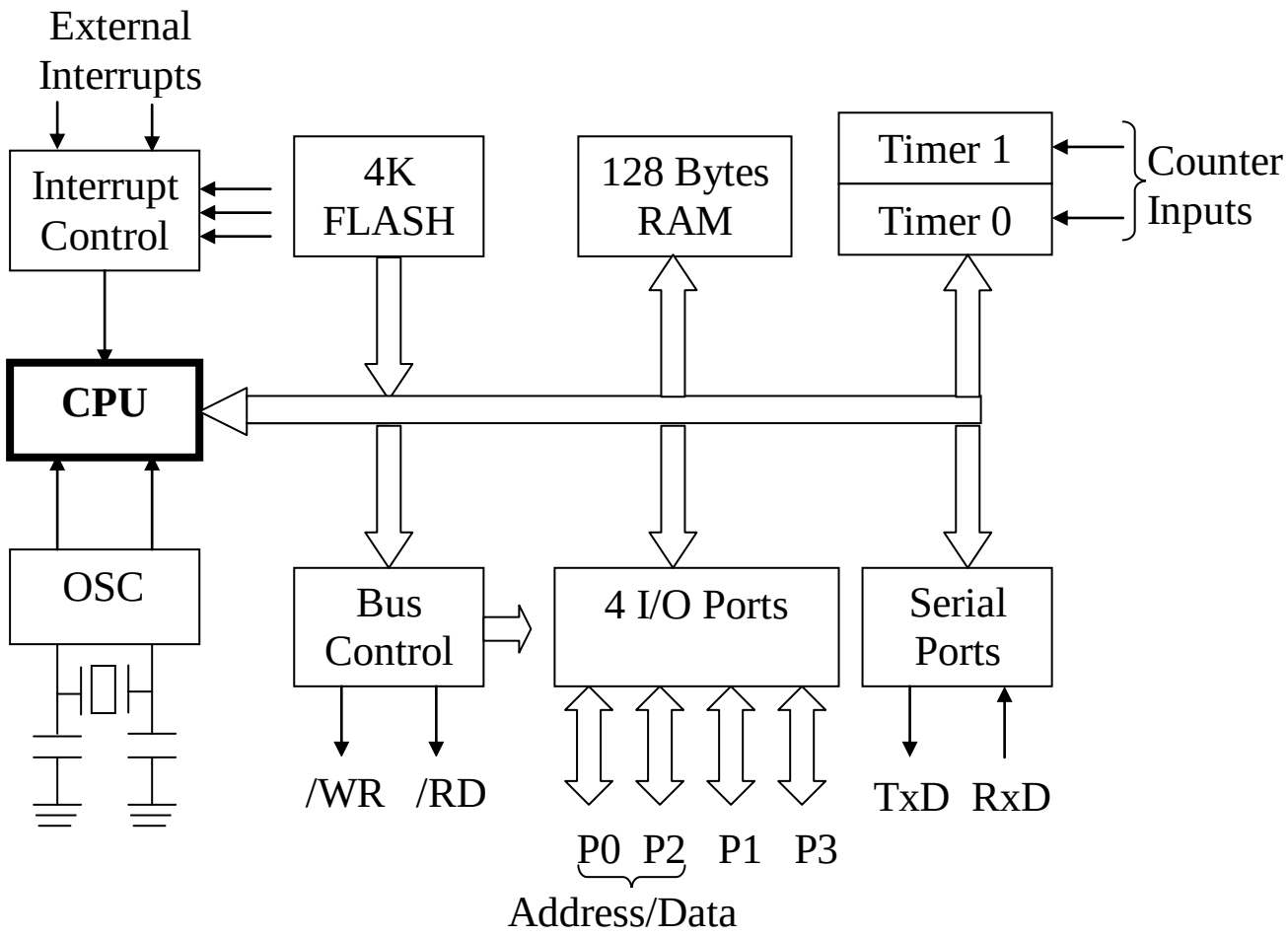
- Bộ VXL
- Bộ nhớ chương trình (ROM/EPROM/EEPROM/FLASH).
- Bộ nhớ dữ liệu (RAM).
- Các thanh ghi chức năng, các cổng I/O, cơ chế điều khiển ngắt và truyền tin nối tiếp.
- Các bộ thời gian dùng trong lĩnh vực chia tần và tạo thời gian thực.
- ……

Bộ VĐK có thể được lập trình để điều khiển các thiết bị thông tin, viễn thông, thiết bị đo lường, thiết bị điều chỉnh cũng như các ứng dụng trong công nghệ thông tin và kỹ thuật điều khiển tự động. Có thể xem bộ VĐK như một hệ VXL On-chip, đối với họ AT89C51, nó có đầy đủ chức năng của một hệ VXL 8 bit, đươc điều khiển bởi một hệ lệnh, có số lệnh đủ mạnh, cho phép lập trình bằng hợp ngữ (Assembly).

2.2. Sự khác nhau giữa bộ VXL và bộ VĐK.

	VXL	**VĐK**
Phần cứng	CPU đơn chíp.	CPU, RAM, ROM, Timers, SFR, mạch giao tiếp, hệ thống ngắt và cơ chế điều khiển ngắt…..
Tập lệnh	Sử dụng các tập lệnh bao quát, mạnh về kiểu định địa chỉ. Các lệnh này có thể truy xuất dữ liệu lớn, thực hiện ở dạng 1/2 Byte, Byte, Word, Double Word.	Sử dụng các lệnh điều khiển xuất nhập, có thể truy xuất dữ liệu ở dạng Bit hoặc Byte. Các nhóm lệnh chính: Chuyển dữ liệu, điều khiển biến logic, rẽ nhánh chương trình, tính toán số học và logic.
	VXL	**VĐK**
ứng dụng	Trong các hệ máy vi tính.	Trong các hệ thống điều khiển, đo lường và điều chỉnh…

2.3. Sơ đồ khối.



Hình 2.1. Sơ đồ khối họ VĐK AT89C51

Bộ VĐK 8 bit AT89C51 hoạt động ở tần số 12 MHz, với bộ nhớ ROM 4Kbyte, bộ nhớ RAM 128 Byte cư trú bên trong và có thể mở rộng bộ nhớ ra ngoài. Ở bộ VĐK này còn có 4 cổng 8 bit (P0…P3) vào/ ra 2 chiều để giao tiếp với thiết bị ngoại vi. Ngoài ra, nó còn có:

- 2 bộ đinh thời 16 bit (Timer 0 và Timer 1)
- Mạch giao tiếp nối tiếp.
- Bộ xử lý bit (thao tác trên các bit riêng rẽ).
- Hệ thống điều khiển và xử lý ngắt.
- Các kênh điều khiển/ dữ liệu/ địa chỉ.
- CPU
- Các thanh ghi chức năng đặc biệt (SFR).

…

Tuy nhiên, tuỳ thuộc vào từng họ VĐK của từng hãng sản xuất khác nhau mà tính năng cũng như phạm vi ứng dụng của mỗi bộ VĐK là khác nhau, và chúng được thể hiện trong các bảng thống kê sau:

Họ VĐK	ROM (bytes)	RAM (bytes)	Tốc độ (MHz)	Các chân I/O	Timer/ Counter	UART	Nguồn ngắt
8051							
8031AH	ROMLESS	128	12	32	2	1	5
8051AH	4K ROM	128	12	32	2	1	5
8051AHP	4K ROM	128	12	32	2	1	5
8751H	4K EPROM	128	12	32	2	1	5
8751BH	4K EPROM	128	12	32	2	1	5
8052							
8032AH	ROMLESS	256	12	32	3	1	6
8052AH	8K ROM	256	12	32	3	1	6
8752BH	8K EPROM	256	12	32	3	1	6
80C51				32			
80C31BH	ROMLESS	128	12,16	32	2	1	5
80C51BH	4K ROM	128	12,16	32	2	1	5
80C31BHP	4K ROM	128	12,16	32	2	1	5
87C51	4K EPROM	128	12,16,20 ,24	32	2	1	5
8xC52/54/58							
80C32	ROMLESS	256	12,16,20 ,24	32	3	1	6
80C52	8K ROM	256	12,16,20 ,24	32	3	1	6
87C52	8K EPROM	256	12,16,20 ,24	32	3	1	6
80C54	16K ROM	256	12,16,20 ,24	32	3	1	6
87C54	16K EPROM	256	12,16,20 ,24	32	3	1	6
Họ VĐK	**ROM (bytes)**	**RAM (bytes)**	**Tốc độ (MHz)**	**Các chân I/O**	**Timer/ Counter**	**UART**	**Nguồn ngắt**
80C58	32K ROM	256	12,16,20 ,24	32	3	1	6
87C58	32K EPROM	256	12,16,20	32	3	1	6

			,24				
8xL52/54/58							
80L52	8K ROM	256	12,16,20	32	3	1	6
87L52	8K OTP ROM	256	12,16,20	32	3	1	6
80L54	16K ROM	256	12,16,20	32	3	1	6
87L54	16K OTP ROM	256	12,16,20	32	3	1	6
80L58	32K ROM	256	12,16,20	32	3	1	6
87L58	32K OTP ROM	256	12,16,20	32	3	1	6
…							

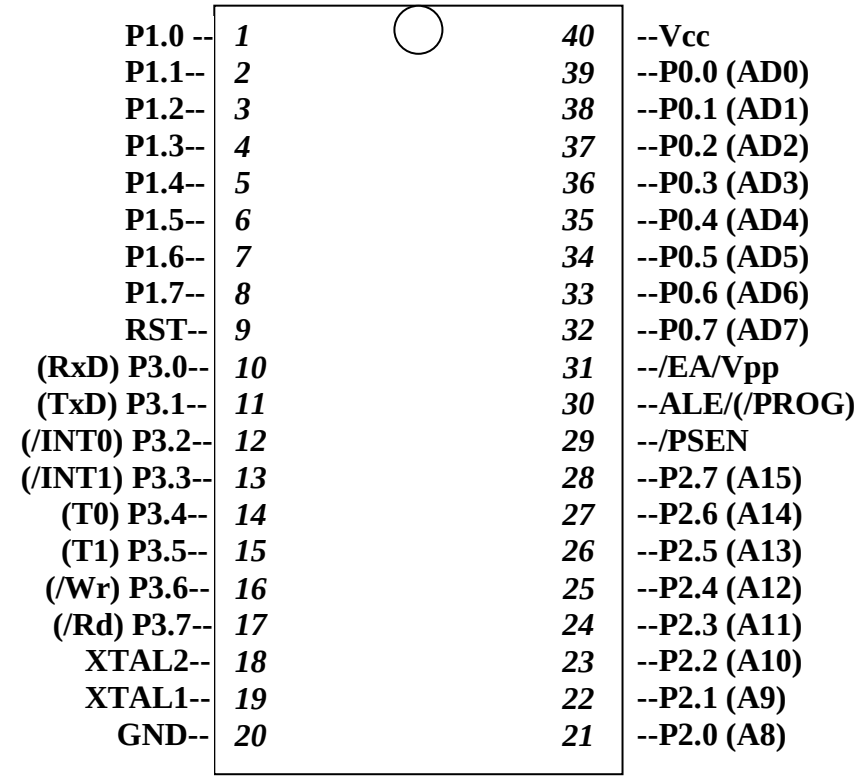
***Bảng 2.1*.** Các thông số của các họ VĐK thuộc hãng Intel (MSC 51)

Họ VĐK	***Bộ nhớ chương trình(Bytes)***	***Bộ nhớ dữ liệu (Bytes)***	***Timer 16 bit***	***Công nghệ***
AT89C1051	1K Flash	64 RAM	1	CMOS
AT89C2051	2K Flash	128 RAM	2	CMOS
AT89C51	4K Flash	128 RAM	2	CMOS
AT89C52	8K Flash	256 RAM	3	CMOS
AT89C55	20K Flash	256 RAM	3	CMOS
AT89S8252	8K Flash	256 RAM + 2K EEPROM	3	CMOS
AT89S53	12K Flash	256 RAM	3	CMOS

***Bảng 2.2*.** Các thông số của các họ VĐK thuộc hãng Atmel

.Trong khuôn khổ tài liệu này, tác giả sẽ tập trung trình bày cấu trúc phần cứng của họ VĐK AT89C51 thuộc hãng Atmel.

2.4. Sơ đồ chân tín hiệu của 80C51/AT89C51.



Hình 2.2. IC 80C51/AT89C51

Chức năng của các chân tín hiệu như sau:

- P0.0 đến P0.7 là các chân của cổng 0.
- P1.0 đến P1.7 là các chân của cổng 1.
- P2.0 đến P2.7 là các chân của cổng 2
- P3.0 đến P3.7 là các chân của cổng 3
- RxD: Nhận tín hiệu kiểu nối tiếp.
- TxD: Truyền tín hiệu kiểu nối tiếp.
- /INT0: Ngắt ngoài 0.
- /INT1: Ngắt ngoài 1.
- T0: Chân vào 0 của bộ Timer/Counter 0.
- T1: Chân vào 1 của bộ Timer/Counter 1.

- /Wr: Ghi dữ liệu vào bộ nhớ ngoài.
- /Rd: Đọc dữ liệu từ bộ nhớ ngoài.
- RST: Chân vào Reset, tích cực ở mức logic cao trong khoảng 2 chu kỳ máy.
- XTAL1: Chân vào mạch khuyếch đại dao động
- XTAL2: Chân ra từ mạch khuyếch đại dao động.
- /PSEN : Chân cho phép đọc bộ nhớ chương trình ngoài (ROM ngoài).
- ALE (/PROG): Chân tín hiệu cho phép chốt địa chỉ để truy cập bộ nhớ ngoài, khi On-chip xuất ra byte thấp của địa chỉ. Tín hiệu chốt được kích hoạt ở mức cao, tần số xung chốt = 1/6 tần số dao động của bộ VĐK. Nó có thể được dùng cho các bộ Timer ngoài hoặc cho mục đích tạo xung Clock. Đây cũng là chân nhận xung vào để nạp chương trình cho Flash (hoặc EEPROM) bên trong On-chip khi nó ở mức thấp.
- /EA/Vpp: Cho phép On-chip truy cập bộ nhớ chương trình ngoài khi /EA=0, nếu /EA=1 thì On-chip sẽ làm việc với bộ nhớ chương trình nội trú. Khi chân này được cấp nguồn điện áp 12V (Vpp) thì On-chip đảm nhận chức năng nạp chương trình cho Flash bên trong nó.
- Vcc: Cung cấp dương nguồn cho On-chip (+ 5V).
- GND: nối mát.

2.5. Chức năng các thành phần của AT89C51:

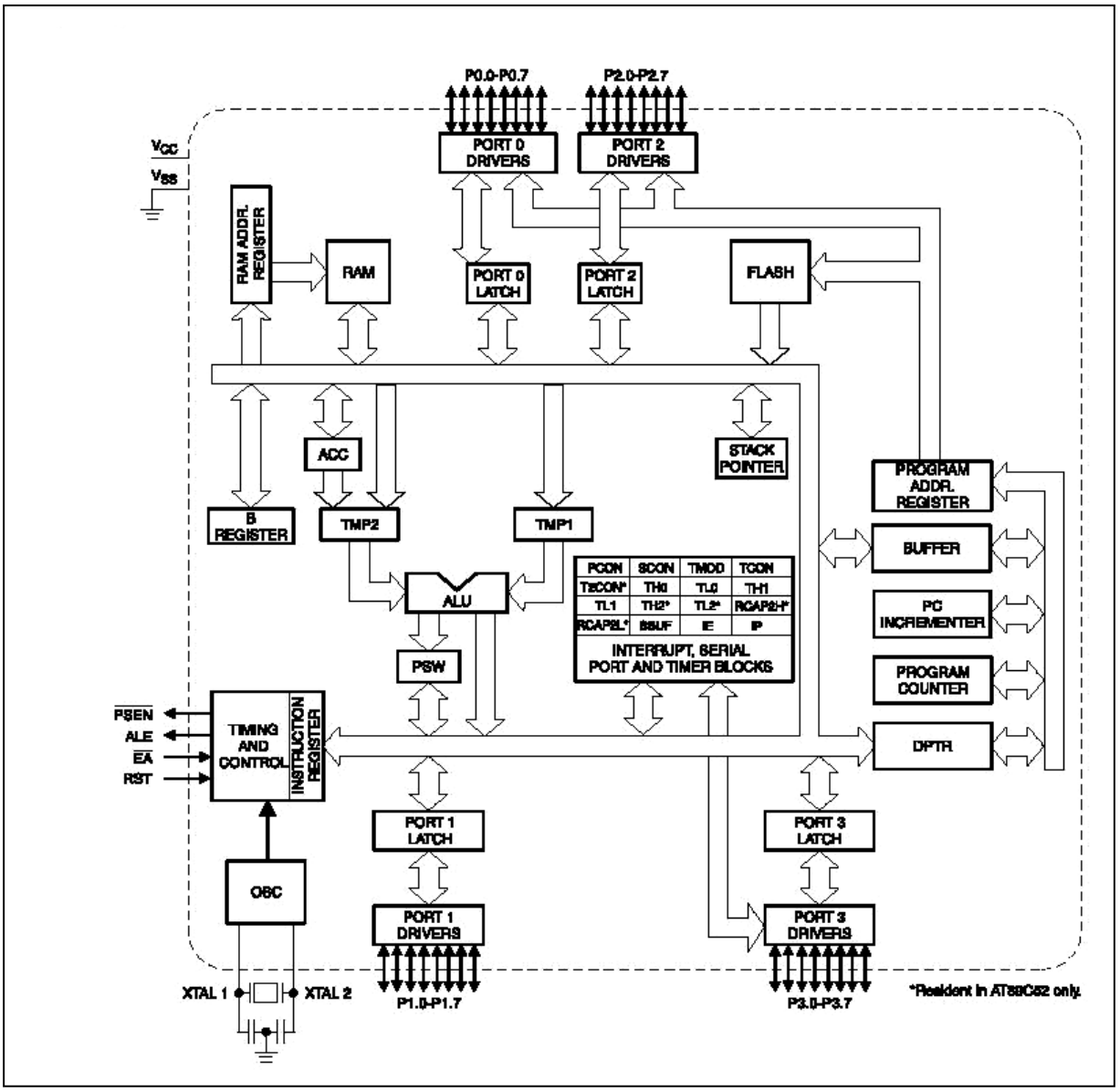
2.5.1. Các thanh ghi chức năng đặc biệt.

SFR đảm nhiệm các chức năng khác nhau trong On-chip. Chúng nằm ở RAM bên trong On-chip, chiếm vùng không gian nhớ 128 Byte được định địa chỉ từ 80h đến FFh. Cấu trúc của SFR bao gồm các chức năng thể hiện ở bảng 2.3 và bảng 2.4.

Thanh ghi	**Nội dung** *MSB*							*LSB*
IE	EA	-	ET2	ES	ET1	EX1	ET0	EX0
IP	-	-	PT2	PS	PT1	PX1	PT0	PX0
PSW	CY	AC	FO	RS1	RS0	OV	-	P
TMOD	GATE	C/(/T)	M1	M0	GATE	C/(/T)	M1	M0
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
PCON	SMOD	-	-	-	GF1	GF0	PD	IDL
P1	T2	T2EX			/SS	MOSI	MISO	SCK

P3	RXD	TXD	/INT0	/INT1	T0	T1	/WR	/RD

***Bảng 2.3*.** Chức năng riêng của từng thanh ghi trong SFR



Hình 2.3. Sơ đồ khối của AT89

Symbol	Name	Address	Reset Values

* ACC	Thanh ghi tích luỹ	0E0h	00000000b
* B	Thanh ghi B	0F0h	00000000b
* PSW	Từ trạng thái chương trình	0D0h	00000000b
SP	Con trỏ ngăn xếp	81h	00000111b
DP0L	Byte cao của con trỏ dữ liệu 0	82h	00000000b
DP0H	Byte thấp của con trỏ dữ liệu 0	83h	00000000b
* P0	Cổng 0	80h	11111111b
* P1	Cổng 1	90h	11111111b
Symbol	**Name**	**Address**	**Reset Values**
* P2	Cổng 2	0A0h	11111111b
* P3	Cổng 3	0B0h	11111111b
* IP	TG điều khiển ngắt ưu tiên	0B8h	xxx00000b
* IE	TG điều khiển cho phép ngắt	0A8h	0xx00000b
TMOD	Điều khiển kiểu Timer/Counter	89h	00000000b
* TCON	TG điều khiển Timer/Counter	88h	00000000b
TH0	Byte cao của Timer/Counter 0	8Ch	00000000b
TL0	Byte thấp của Timer/Counter 0	8Ah	00000000b
TH1	Byte cao của Timer/Counter 1	8Dh	00000000b
TL1	Byte thấp của Timer/Counter 1	8Bh	00000000b
* SCON	Serial Control	98h	00000000b
SBUF	Serial Data Buffer	99h	Indeterminate
PCON	Power Control	87h	0xxx0000b

* : có thể định địa chỉ bit, x: không định nghĩa

Bảng 2.4. Địa chỉ, ý nghĩa và giá trị của các SFR sau khi Reset

2.5.1.1. Thanh ghi ACC: là thanh ghi tích luỹ, dùng để lưu trữ các toán hạng và kết quả của phép tính. Thanh ghi ACC dài 8 bits. Trong các tập lệnh của On-chip, nó thường được quy ước đơn giản là A.

2.5.1.2. Thanh ghi B : Thanh ghi này được dùng khi thực hiện các phép toán nhân và chia. Đối với các lệnh khác, nó có thể xem như là thanh ghi đệm tạm thời. Thanh ghi B dài 8 bits. Nó thường được dùng chung với thanh ghi A trong các phép toán nhân hoặc chia.

2.5.1.3. Thanh ghi SP: Thanh ghi con trỏ ngăn xếp dài 8 bit. SP chứa địa chỉ của dữ liệu hiện đang ở đỉnh của ngăn xếp. Giá trị của nó được tự động tăng lên khi thực hiện lệnh PUSH trước khi dữ liệu được lưu trữ trong ngăn xếp. SP sẽ tự động giảm xuống khi thực hiện lệnh POP. Ngăn xếp có thể đặt ở bất cứ nơi nào trong RAM on-chip, nhưng sau khi khởi động lại hệ thống thì con trỏ ngăn xếp mặc định sẽ trỏ tới địa chỉ khởi đầu là 07h, vì vậy ngăn xếp sẽ bắt đầu từ địa chỉ 08h. Ta cũng có thể định con trỏ ngăn xếp tại địa chỉ mong muốn bằng các lệnh di chuyển dữ liệu thông qua định địa chỉ tức thời.

2.5.1.4. Thanh ghi DPTR: Thanh ghi con trỏ dữ liệu (16 bit) bao gồm 1 thanh ghi byte cao (DPH-8bit) và 1 thanh ghi byte thấp (DPL-8bit). DPTR có thể được dùng như thanh ghi 16 bit hoặc 2 thanh ghi 8 bit độc lập. Thanh ghi này được dùng để truy cập RAM ngoài.

2.5.1.5. Ports 0 to 3: P0, P1, P2, P3 là các chốt của các cổng 0, 1, 2, 3 tương ứng. Mỗi chốt gồm 8 bit. Khi ghi mức logic 1 vào một bit của chốt, thì chân ra tương ứng của cổng ở mức logic cao. Còn khi ghi mức logic 0 vào mỗi bit của chốt thì chân ra tương ứng của cổng ở mức logic thấp. Khi các cổng đảm nhiệm chức năng như các đầu vào thì trạng thái bên ngoài của các chân cổng sẽ được giữ ở bit chốt tương ứng. Tất cả 4 cổng của on-chip đều là cổng I/O hai chiều, mỗi cổng đều có 8 chân ra, bên trong mỗi chốt bit có bộ "Pullup-tăng cường" do đó nâng cao khả năng nối ghép của cổng với tải (có thể giao tiếp với 4 đến 8 tải loại TTL).

2.5.1.6. Thanh ghi SBUF: Đệm dữ liệu nối tiếp gồm 2 thanh ghi riêng biệt, một thanh ghi đệm phát và một thanh ghi đệm thu. Khi dữ liệu được chuyển tới SBUF, nó sẽ đi vào bộ đệm phát, và được giữ ở đấy để chế biến thành dạng truyền tin nối tiếp. Khi dữ liệu được truyền đi từ SBUF, nó sẽ đi ra từ bộ đệm thu.

2.5.1.7. Các Thanh ghi Timer: Các đôi thanh ghi (TH0, TL0), (TH1, TL1) là các thanh ghi đếm 16 bit tương ứng với các bộ Timer/Counter 0 và 1.

2.5.1.8. Các thanh ghi điều khiển: Các thanh ghi chức năng đặc biệt: IP, IE, TMOD, TCON, SCON, và PCON bao gồm các bit trạng thái và điều khiển đối với hệ thống ngắt, các bộ Timer/Counter và cổng nối tiếp. Chúng sẽ được mô tả ở phần sau.

2.5.1.9. Thanh ghi PSW: Từ trạng thái chương trình dùng để chứa thông tin về trạng thái chương trình. PSW có độ dài 8 bit, mỗi bit đảm nhiệm một chức năng cụ thể. Thanh ghi này cho phép truy cập ở dạng mức bit.

* CY: Cờ nhớ. Trong các phép toán số học, nếu có nhớ từ phép cộng bit 7 hoặc có số mượn mang đến bit 7 thì CY được đặt bằng 1.

* AC: Cờ nhớ phụ (Đối với mã BCD). Khi cộng các giá trị BCD, nếu có một số nhớ được tạo ra từ bit 3 chuyển sang bit 4 thì AC được đặt bằng 1. Khi giá trị được cộng là BCD, lệnh cộng phải được thực hiện tiếp theo bởi lệnh ***DA A*** (hiệu chỉnh thập phân thanh chứa A) để đưa các kết quả lớn hơn 9 về giá trị đúng.

* F0: Cờ 0 (Có hiệu lực với các mục đích chung của người sử dụng)

* RS1: Bit 1 điều khiển chọn băng thanh ghi.

* RS0: Bit 0 điều khiển chọn băng thanh ghi.

Lưu ý: RS0, RS1 được đặt/xoá bằng phần mềm để xác định băng thanh ghi đang hoạt động (Chọn băng thanh ghi bằng cách đặt trạng thái cho 2 bit này)

	RS1	**RS0**
Bank 0	0	0
Bank 1	0	1
Bank 2	1	0
Bank 3	1	1

Bảng 2.5. Chọn băng thanh ghi

* OV: Cờ tràn. Khi thực hiện các phép toán cộng hoặc trừ mà xuất hiện một tràn số học, thì OV được đặt bằng 1. Khi các số có dấu được cộng hoặc được trừ, phần mềm có thể kiểm tra OV để xác định xem kết quả có nằm trong tầm hay không. Với phép cộng các số không dấu, OV được bỏ qua. Kết quả lớn hơn +128 hoặc nhỏ hơn -127 sẽ đặt OV=1.

* -: Bit dành cho người sử dụng tự định nghĩa(Nếu cần).

* P: Cờ chẵn lẻ. Được tự động đặt/ xoá bằng phần cứng trong mỗi chu trình lệnh để chỉ thị số chẵn hay lẻ của bit 1 trong thanh ghi tích luỹ. Số các bit 1 trong A cộng với bit P luôn luôn là số chẵn.

2.5.1.10. Thanh ghi PCON: Thanh ghi điều khiển nguồn.

* SMOD: Bit tạo tốc độ Baud gấp đôi. Nếu Timer 1 được sử dụng để tạo tốc độ baud và SMOD=1, thì tốc độ Baud được tăng lên gấp đôi khi cổng truyền tin nối tiếp được dùng bởi các kiểu 1, 2 hoặc 3.

* -: Không sử dụng, các bit này có thể được dùng ở các bộ VXL trong tương lai. Người sử dụng không được phép tự định nghĩa cho các bit này.

* GF0, GF1: Cờ dùng cho các mục đích chung (đa mục đích).

* PD: bit nguồn giảm. Đặt bit này ở mức tích cực để vận hành chế độ nguồn giảm trong AT89C51. Chỉ có thể ra khỏi chế độ bằng Reset.

* IDL: bit chọn chế độ nghỉ. Đặt bit này ở mức tích cực để vận hành kiểu Idle (Chế độ không làm việc) trong AT89C51.

Lưu ý: Nếu PD và IDL cùng được kích hoạt cùng 1 lúc ở mức tích cực, thì PD được ưu tiên thực hiện trước. Chỉ ra khỏi chế độ bằng 1 ngắt hoặc Reset lại hệ thống.

2.5.1.11. Thanh ghi IE: Thanh ghi cho phép ngắt

* EA: Nếu EA=0, không cho phép bất cứ ngắt nào hoạt động. Nếu EA=1, mỗi nguồn ngắt riêng biệt được phép hoặc không được phép hoạt động bằng cách đặt hoặc xoá bit Enable của nó.
* -: Không dùng, người sử dụng không nên định nghĩa cho Bit này, bởi vì nó có thể được dùng ở các bộ AT89 trong tương lai.
* ET2: Bit cho phép hoặc không cho phép ngắt bộ Timer 2.
* ES: Bit cho phép hoặc không cho phép ngắt cổng nối tiếp (SPI và UART).
* ET1: Bit cho phép hoặc không cho phép ngắt tràn bộ Timer 1

* EX1: Bit cho phép hoặc không cho phép ngắt ngoài 1.
* ET0: Bit cho phép hoặc không cho phép ngắt tràn bộ Timer 0
* EX0: Bit cho phép hoặc không cho phép ngắt ngoài 0.

2.5.1.12. Thanh ghi IP: Thanh ghi ưu tiên ngắt**.**

* - : Không dùng, người sử dụng không nên ghi "1" vào các Bit này.
* PT2: Xác định mức ưu tiên của ngắt Timer 2.
* PS: Định nghĩa mức ưu tiên của ngắt cổng nối tiếp.
* PT1: Định nghĩa mức ưu tiên của ngắt Timer 1.
* PX1: Định nghĩa mức ưu tiên của ngắt ngoàI 1.
* PT0: Định nghĩa mức ưu tiên của ngắt Timer 0.
* PX0: Định nghĩa mức ưu tiên của ngắt ngoàI 0.

2.5.1.13. Thanh ghi TCON : Thanh ghi điều khiển bộ Timer/Counter

* TF1: Cờ tràn Timer 1. Được đặt bởi phần cứng khi bộ Timer 1 tràn. Được xoá bởi phần cứng khi bộ vi xử lý hướng tới chương trình con phục vụ ngắt. * TR1: Bit điều khiển bộ Timer 1 hoạt động. Được đặt/xoá bởi phần mềm để điều khiển bộ Timer 1 ON/OFF
* TF0: Cờ tràn Timer 0. Được đặt bởi phần cứng khi bộ Timer 0 tràn. Được xoá bởi phần cứng khi bộ vi xử lý hướng tới chương trình con phục vụ ngắt.
* TR0: Bit điều khiển bộ Timer 0 hoạt động. Được đặt/xoá bởi phần mềm để điều khiển bộ Timer 0 ON/OFF.
* IE1: Cờ ngắt ngoài 1. Được đặt bởi phần cứng khi sườn xung của ngắt ngoài 1 được phát hiện. Được xoá bởi phần cứng khi ngắt được xử lý.
* IT1: Bit điều khiển ngắt 1 để tạo ra ngắt ngoài. Được đặt/xoá bởi phần mềm.
* IE0: Cờ ngắt ngoài 0. Được đặt bởi phần cứng khi sườn xung của ngắt ngoài 0 được phát hiện. Được xoá bởi phần cứng khi ngắt được xử lý.
* IT0: Bit điều khiển ngắt 0 để tạo ra ngắt ngoài. Được đặt/xoá bởi phần mềm.

2.5.1.14. Thanh ghi TMOD: Thanh ghi điều khiển kiểu Timer/Counter

* GATE: Khi TRx được thiết lập và GATE=1, bộ TIMER/COUTERx hoạt động chỉ khi chân INTx ở mức cao. Khi GATE=0, TIMER/COUNTERx sẽ hoạt động chỉ khi TRx=1.
* C/(/T): Bit này cho phép chọn chức năng là Timer hay Counter.

- Bit này được xoá để thực hiện chức năng Timer
- Bit này được đặt để thực hiện chức năng Counter

* M0, M1: Bit chọn Mode, để xác định trạng thái và kiểu Timer/Counter:

- M1=0, M0=0: Chọn kiểu bộ Timer 13 bit. Trong đó THx dài 8 bit, còn TLx dài 5 bit.

- M1=0, M0=1: Chọn kiểu bộ Timer 16 bit. THx và TLx dài 16 bit được ghép tầng.

- M1=1, M0=0: 8 bit Auto reload. Các thanh ghi tự động nạp lại mỗi khi bị tràn. Khi bộ Timer bị tràn, THx dài 8 bit được giữ nguyên giá trị, còn giá trị nạp lại được đưa vào TLx.

- M1=1, M0=1: Kiểu phân chia bộ Timer. TL0 là 1 bộ Timer/Counter 8 bit, được điều khiển bằng các bit điều khiển bộ Timer 0, Còn TH0 chỉ là bộ Timer 8 bit, được điều khiển bằng các bit điều khiển Timer 1.

- M1=1, M0=1: Timer/Counter 1 Stopped

2.5.1.15. Thanh ghi SCON:

SCON là thanh ghi trạng thái và điều khiển cổng nối tiếp. Nó không những chứa các bit chọn chế độ, mà còn chứa bit dữ liệu thứ 9 dành cho việc truyền và nhận tin (TB8 và RB8) và chứa các bit ngắt cổng nối tiếp.

* SM0, SM1: Là các bit cho phép chọn chế độ cho cổng truyền nối tiếp.

SM0	SM1	Mode	Đặc điểm	Tốc độ Baud
0	0	0	Thanh ghi dịch	F_{osc} /12
0	1	1	8 bit UART	Có thể thay đổi (được đặt bởi bộ Timer)
1	0	2	9 bit UART	F_{osc} /64 hoặc F_{osc} /32
1	1	3	9 bit UART	Có thể thay đổi (được đặt bởi bộ Timer)

Bảng 2.6. Chọn Mode trong SCON

* SM2: Cho phép truyền tin đa xử lý, thể hiện ở Mode 2 và 3. ở chế độ 2 hoặc 3, nếu đặt SM2 = 1 thì RI sẽ không được kích hoạt nếu bit dữ liệu thứ 9

(RB8) nhận được giá trị bằng 0. ở Mode 1, nếu SM2=1 thì RI sẽ không được kích hoạt nếu bit dừng có hiệu lực đã không được nhận. ở chế độ 0, SM2 nên bằng 0

* REN: Cho phép nhận nối tiếp. Được đặt hoặc xoá bởi phần mềm để cho phép hoặc không cho phép nhận.

* TB8: Là bit dữ liệu thứ 9 mà sẽ được truyền ở Mode 2 và 3. Được đặt hoặc xoá bởi phần mềm.

* RB8: Là bit dữ liệu thứ 9 đã được nhận ở Mode 2 và 3. Ở Mode 1, nếu SM2=0 thì RB8 là bit dừng đã được nhận. Ở Mode 0, RB8 không được sử dụng.

* TI: Cờ ngắt truyền. Được đặt bởi phần cứng tại cuối thời điểm của bit thứ 8 trong Mode 0, hoặc đầu thời điểm của bit dừng trong các Mode khác. Ở bất kỳ quá trình truyền nối tiếp nào, nó cũng phải được xoá bằng phần mềm.

* RI: Cờ ngắt nhận. Được đặt bởi phần cứng tại cuối thời điểm của bit thứ 8 trong Mode 0, hoặc ở giữa thời điểm của bit dừng trong các Mode khác. Ở bất kỳ quá trình nhận nối tiếp nào (trừ trường hợp ngoại lệ, xem SM2), nó cũng phải được xoá bằng phần mềm.

2.5.2. Khối tạo thời gian và bộ đếm (Timer/Counter).

On-chip AT89C51 có 2 thanh ghi Timer/Counter dài 16 bit, đó là: Timer 0 và Timer 1. Trong On-chip AT89C52, ngoài Timer 0 và Timer 1 nó còn có thêm bộ Timer 2. Cả 3 bộ Timer này đều có thể được điều khiển để thực hiện chức năng thời gian hay bộ đếm, thông qua thanh ghi TMOD.

Khi thanh ghi Timer/Counter làm việc ở kiểu Timer, thì sau mỗi chu kỳ máy nội dung trong thanh ghi được gia tăng thêm 1 đơn vị. Vì vậy thanh ghi này đếm số chu kỳ máy. Một chu kỳ máy có 12 chu kỳ dao động, do đó tốc độ đếm của thanh ghi là 1/12 tần số dao động.

Khi thanh ghi Timer/Counter làm việc ở kiểu Counter, xung nhịp bên ngoài được đưa vào để đếm ở T0 hoặc T1. Nội dung thanh ghi được tăng lên khi có sự chuyển trạng thái từ 1 về 0 tại chân đầu vào ngoài T0 hoặc T1. Xung nhịp ở các đầu vào ngoài được lấy mẫu tại thời điểm S5P2 của mỗi chu kỳ máy. Khi quá trình lấy mẫu phát hiện ra mức cao ở 1 chu kỳ và mức thấp ở chu kỳ tiếp theo, thì bộ đếm được tăng lên. Giá trị mới của bộ đếm xuất hiện trong thanh ghi tại thời điểm S3P1 của chu kỳ máy sau khi sự chuyển trạng thái đã được phát hiện. Vì vậy để nội dung của thanh ghi tăng lên 1 đơn vị phải mất 2 chu kỳ máy, nên tốc độ đếm tối đa là 1/24 tần số bộ dao động. Không có sự giới hạn số vòng thực hiện của tín hiệu ở đầu vào ngoài, nhưng nó sẽ giữ ít nhất 1 chu kỳ máy đầy đủ để

đảm bảo chắc chắn rằng một mức đã cho được lấy mẫu ít nhất 1 lần nữa trước khi nó thay đổi.

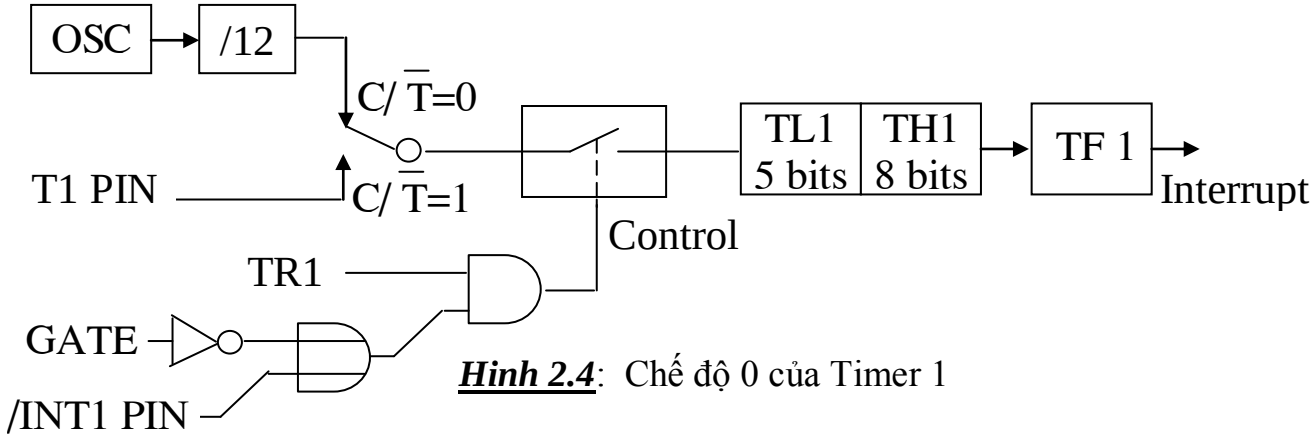
Do xung nhịp bên ngoài có tần số bất kỳ nên các bộ Timer (0 và 1) có 4 chế độ làm việc khác nhau để lựu chọn: (13 bit Timer, 16 bit Timer, 8 bit auto-reload, split Timer).

Timer 0 và Timer 1:

Trong AT89C51 và AT89C52 đều có các bộ Timer 0 và 1. Chức năng Timer hay Counter được chọn lựa bởi các bit điều khiển C/(/T) trong thanh ghi TMOD. Hai bộ Timer/Counter này có 4 chế độ hoạt động, được lựa chọn bởi cặp bit (M0, M1) trong TMOD. Chế độ 0, 1 và 2 giống nhau cho các chức năng Timer/Counter, nhưng chế độ 3 thì khác. Bốn chế độ hoạt động được mô tả như sau:

+ ***Chế độ 0***: Cả 2 bộ Timer 0 và 1 ở chế độ 0 có cấu hình như một thanh ghi 13 bit, bao gồm 8 bit của thanh ghi THx và 5 bit thấp của TLx. 3 bit cao của TLx không xác định chắc chắn, nên được làm ngơ. Khi thanh ghi được xoá về 0, thì cờ ngắt thời gian TFx được thiết lập. Bộ Timer/Counter hoạt động khi bit điều khiển TRx được thiết lập (TRx=1) và, hoặc Gate trong TMOD bằng 0, hoặc /INTx=1. Nếu đặt GATE=1 thì cho phép điều khiển Timer/ Counter bằng đường vào ngoài /INTx, để dễ dàng xác định độ rộng xung.

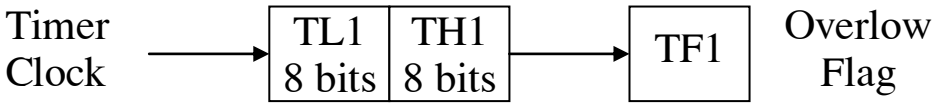
Khi hoạt động ở chức năng thời gian thì bit C/(/T)=0, do vậy xung nhịp từ bộ dao động nội, qua bộ chia tần cho ra tần số $f=f_{osc}/12$ được đưa vào để đếm trong



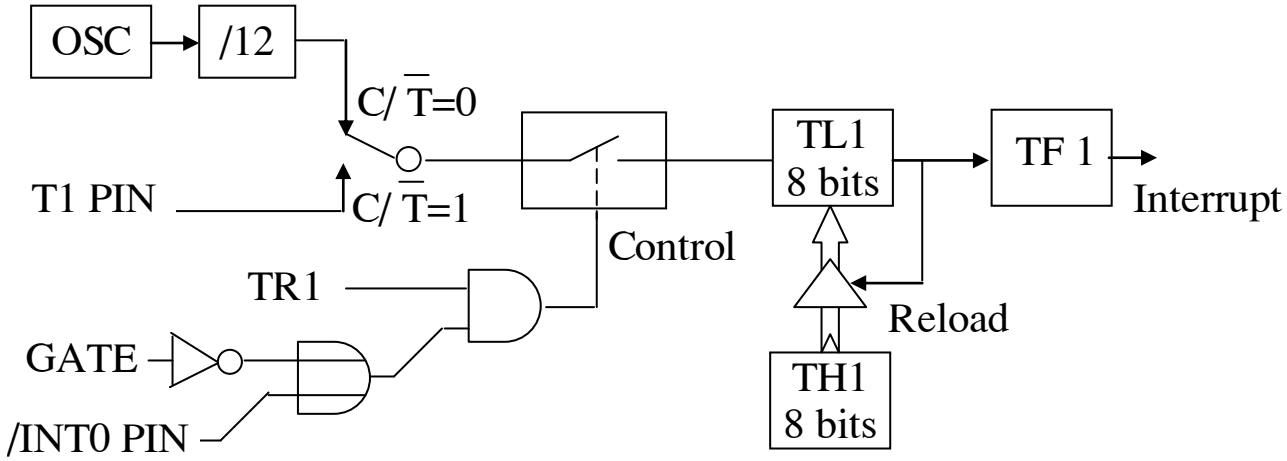
Hình 2.4: Chế độ 0 của Timer 1

thanh ghi Timer/Counter. Khi hoạt động ở chức năng bộ đếm thì bit C/(/T)=1, lúc đó xung nhịp ngoài đưa vào sẽ được đếm.

+ ***Chế độ 1***: hoạt động tương tự như chế độ 0, chỉ khác là thanh ghi Timer/Counter được sử dụng cả 16 bit. Xung nhịp được dùng kết hợp với các thanh ghi thời gian byte thấp và byte cao (TH1 và TL1). Khi xung Clock được nhận, bộ Timer sẽ đếm tăng lên: 0000h, 0001h, 0002,…Khi hiện tượng tràn xẩy ra, cờ tràn sẽ chuyển FFFFh về 0000h, và bộ Timer tiếp tục đếm. Cờ tràn của Timer 1 là bit TF1 ở trong TCON, nó được đọc hoặc ghi bởi phần mềm, xem hình 2.5 (Timer/Counter 1 Mode 1: 16 bit Counter).



Hình 2.5: Chế độ 1 của Timer 1



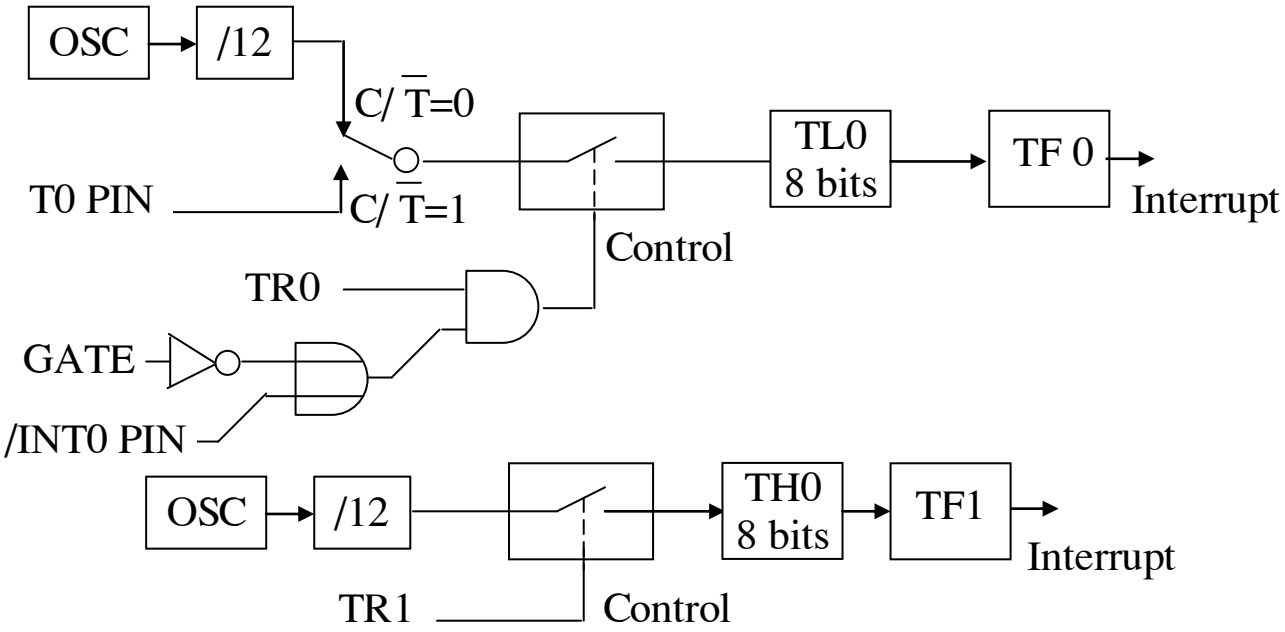
Hình 2.6: Chế độ 2 của Timer 1

+ ***Chế độ 2***: Chế độ này của thanh ghi Timer cũng hoạt động tương tự như 2 chế độ trên, nhưng nó được tổ chức như bộ đếm 8 bit (TL1) với chế độ tự động nạp lại, như hình 2.6. Khi xẩy ra hiện tượng tràn ở TL1, không chỉ thiết lập bit TF1 mà còn tự động nạp lại cho TL1 bằng nội dung của TH1, đã được thiết lập

bởi phần mềm. Quá trình nạp lại cho phép nội dung của TH1 không bị thay đổi. Chế độ 2 của Timer/Counter 0 cũng tương tự như Timer/Counter 1.

+ ***Chế độ 3***: Ở chế độ này, chức năng Timer/Counter 0 và chức năng Timer/Counter 1 khác nhau. Bộ Timer 1 ở chế độ 3 chỉ chứa chức năng đếm của nó, kết quả giống khi đặt TR1=0. Bộ Timer 0 ở chế độ 3 thiết lập TH0, TL0 như là 2 bộ đếm riêng biệt. Mạch Logic đối với chế độ 3 của Timer 0 thể hiện ở hình 2.7. Bộ đếm TL0 được điều khiển bởi các bit: C/(/T), GATE, TR0, /INT0 và khi đếm tràn nó thiết lập cờ ngắt TF0. Bộ đếm TH0 chỉ được điều khiển bởi bit TR1, và khi đếm tràn nó thiết lập cờ ngắt TF1. Vậy, TH0 điều khiển ngắt Timer/Counter 1.

Chế độ 3 thường được dùng khi yêu cầu cần có bộ thời gian hoặc bộ đếm ngoài 8 bit. Đối với Timer 0 ở chế độ 3, AT89C51 có thể có 3 bộ Timer/Counter, còn AT89C52 có thể có 4 bộ. Khi Timer 0 hoạt động ở chế độ 3, thì Timer 1 có thể được bật hoặc tắt bằng chuyển mạch ngoài. Ở chế độ này, Timer 1 có thể được sử dụng bởi cổng nối tiếp như một bộ tạo tốc độ Baud, hoặc trong bất kỳ ứng dụng nào mà không yêu cầu một ngắt.



Hình 2.7: Chế độ 3 của Timer 0

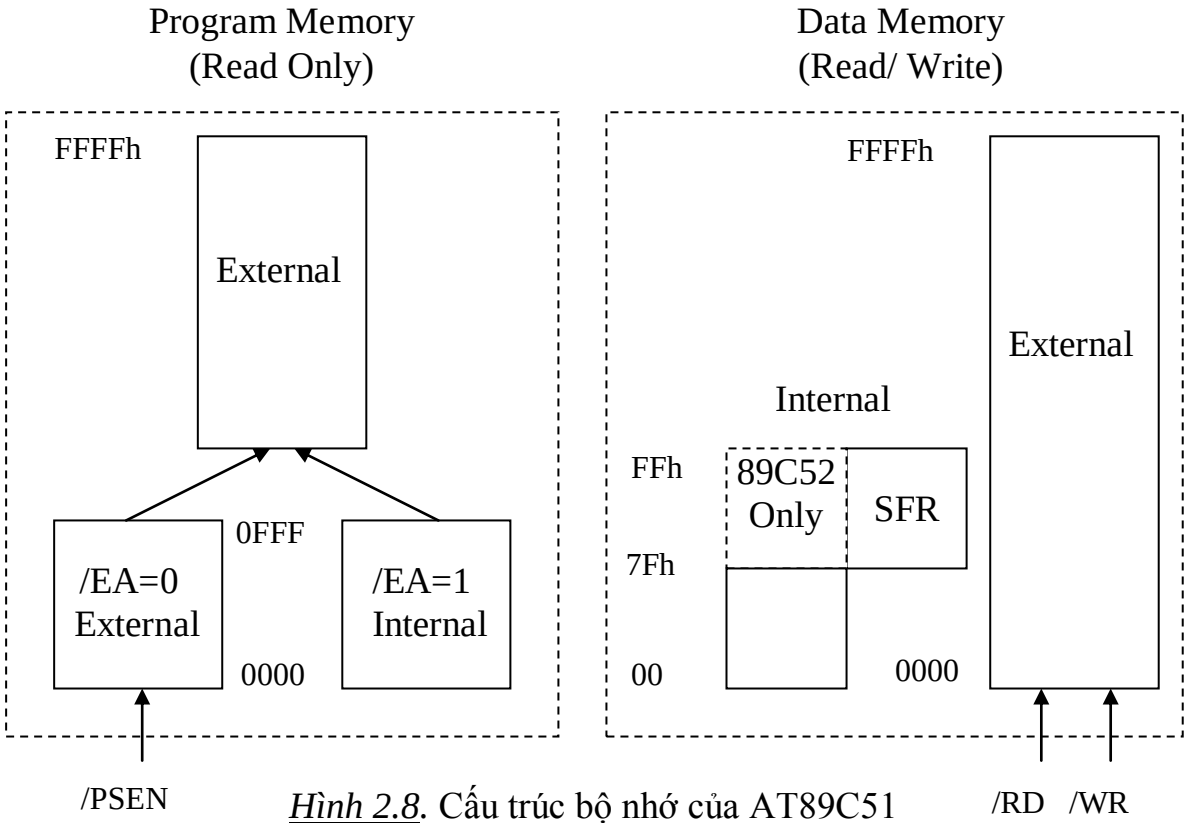
2.5.3. Bộ nhớ chương trình và bộ nhớ dữ liệu nội trú.

Tất cả các bộ Flash Microcontrollers của Atmel đều tổ chức các vùng địa chỉ tách biệt đối với bộ nhớ chương trình và bộ nhớ dữ liệu, được mô tả ở hình dưới đây. Các vùng nhớ chương trình và dữ liệu tách biệt cho phép bộ nhớ dữ liệu được truy cập bởi địa chỉ 8 bit, có thể được lưu trữ với tốc độ cao và được vận hành bởi một bộ CPU 8 bit. Tuy nhiên, địa chỉ bộ nhớ dữ liệu 16 bit cũng có thể được tạo ra thông qua thanh ghi con trỏ dữ liệu (DPTR).

Bộ nhớ chương trình có thể chỉ được đọc. Chúng có thể là bộ nhớ chương trình 64 Kbyte có khả năng định địa chỉ trực tiếp. Để đọc nội dung từ bộ nhớ chương trình ngoài, cần xác định trạng thái phù hợp cho chân /PSEN.

Bộ nhớ dữ liệu chiếm 1 vùng địa chỉ riêng biệt so với bộ nhớ chương trình. 64 Kbyte bộ nhớ ngoài có thể được định địa chỉ trực tiếp trong vùng bộ nhớ dữ liệu ngoài. CPU tạo ra tín hiệu đọc và ghi (/RD, /WR) để truy cập bộ nhớ dữ liệu ngoài.

Bộ nhớ chương trình ngoài và bộ nhớ dữ liệu ngoài có thể được kết hợp bởi các tín hiệu /RD và /PSEN để đưa vào 1 cổng AND và sử dụng đầu ra của cổng này để đọc nội dung từ bộ nhớ dữ liệu/chương trình ngoài.



Hình 2.8. Cấu trúc bộ nhớ của AT89C51

2.5.3.1. Bộ nhớ chương trình nội trú.

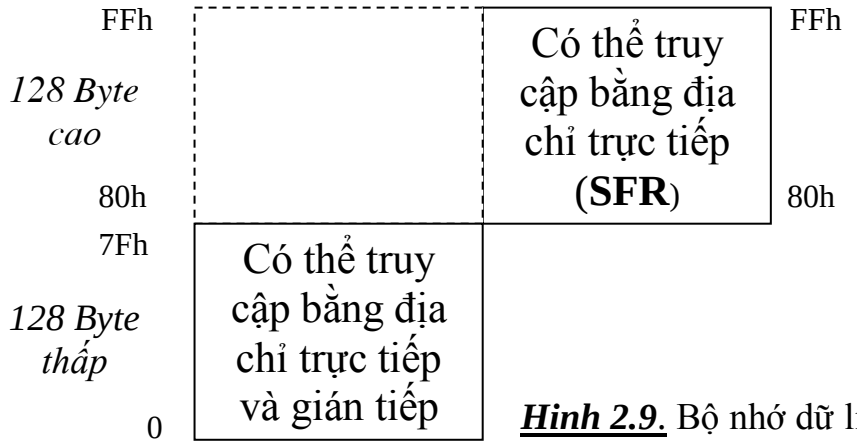
Bộ nhớ chương trình của AT89C51 được tổ chức như thể hiện ở hình trên. Không gian nhớ cực đại của bộ nhớ này chiếm 64 Kbyte, được định địa chỉ từ 0000h đến FFFFh, trong đó có 4 Kbyte Flash nội trú bên trong nó và được định địa chỉ từ 0000h đến 0FFFh. Do đó có thể mở rộng thêm 60 Kbyte bộ nhớ chương trình bên ngoài, được định địa chỉ từ 1000h đến FFFFh. Tuy nhiên bộ VĐK này cũng có thể sử dung toàn bộ bộ nhớ chương trình ngoài bao gồm 64 Kbyte được định địa chỉ từ 0000h đến FFFFh.

Cũng từ hình trên ta thấy, thông qua việc chọn mức logic cho bit /EA có thể lựa chọn để truy cập bộ nhớ chương trình nội trú (4Kb), bộ nhớ chương trình mở rộng ngoại trú (60Kb), hoặc toàn bộ bộ nhớ chương trình ngoại trú bên ngoài On-chip (64Kb). Cụ thể, khi /EA = 1 thì bộ VĐK sử dụng cả bộ nhớ chương trình nội trú và ngoại trú. Ngược lại, khi /EA = 0 thì bộ VĐK chỉ sử dụng bộ nhớ chương trình ngoại trú.

Mỗi khi được Reset, bộ VĐK sẽ truy cập bộ nhớ chương trình tại địa chỉ khởi đầu là 0000h, sau đó nếu cơ chế ngắt được sử dụng thì nó sẽ truy cập tới địa chỉ quy định trong bảng vecter ngắt.

Khi truy cập bộ nhớ chương trình, bộ VĐK sử dụng xung chọn /PSEN để điều khiển. Nếu on-chip làm việc với bộ nhớ chương trình nội trú thì chân phát ra xung chọn /PSEN không sử dụng. Nếu bộ VĐK làm việc với bộ nhớ chương trình ngoại trú thì chân phát ra xung chọn /PSEN được sử dụng. Khi đó nếu /PSEN = 0 thì cho phép bộ VĐK đọc bộ nhớ chương trình ngoài, ngược lại nếu /PSEN = 1 thì bộ VĐK chỉ làm việc với bộ nhớ chương trình nội trú.

2.5.3.2. Bộ nhớ dữ liệu nội trú.



Hình 2.9. Bộ nhớ dữ liệu trong

AT89C51 có bộ nhớ dữ liệu chiếm một khoảng không gian bộ nhớ độc lập với bộ nhớ chương trình. Dung lượng của RAM nội trú ở họ VĐK này là 128 Byte, được định địa chỉ từ 00h đến 7Fh. Phạm vi địa chỉ từ 80h đến FFh dành cho SFR. Tuy nhiên bộ VĐK cũng có thể làm việc với RAM ngoại trú có dung lượng cực đại là 64 Kbyte được định địa chỉ từ 0000h đến FFFFh.

2.5.3.2.1. Vùng nhớ 128 Byte thấp

Vùng nhớ 128 Byte thấp được định địa chỉ từ 00h đến 7Fh, được chia thành 3 vùng con như thể hiện ở hình 2.10.

- Vùng thứ nhất có độ lớn 32 Byte được định địa chỉ từ 00h đến 1Fh bao gồm 4 băng thanh ghi (băng 0...băng 3), mỗi băng có 8 thanh ghi 8 bit. Các thanh ghi trong mỗi băng có tên gọi từ R0 đến R7. Vùng RAM này được truy cập bằng địa chỉ trực tiếp mức Byte, và quá trình chọn để sử dung băng thanh ghi nào là tùy thuộc vào việc lựa chon giá trị cho RS1 và RS0 trong PSW.

- Vùng thứ 2 có độ lớn 16 Byte được định địa chỉ từ 20h đến 2Fh, cho phép truy cập trực tiếp bằng địa chỉ mức bit. Bộ VĐK cung cấp các lệnh có khả năng truy cập tới vùng nhớ 128 bit này (nếu truy cập ở dạng mức bit thì vùng này có địa chỉ được định từ 00h đến 7Fh) ở mức bit. Ở vùng nhớ này, địa chỉ được truy xuất dưới dạng Byte hay Bit tuỳ vào lệnh cụ thể. Chẳng hạn, để đặt bit tại địa chỉ 5Fh có mức logic 1, ta thực hiện lệnh: ***SETB 5Fh*** . Sau khi thực hiện lệnh này, mặc dầu 5Fh là địa chỉ bit cao nhất trong Byte có địa chỉ 2Bh, nhưng nó không làm ảnh hưởng tới các bit khác trong Byte này. Trong khi đó, ở các bộ VXL để thực hiện chức năng như trên cần dùng những lệnh sau:

MOV A,2Bh

ORL A,#10000000b

MOV 2Bh,A

Đây là ưu điểm rõ nét của các bộ VĐK khi thực hiện việc truy xuất các bit riêng rẽ thông qua phần mềm. Các bit có thể đươc đặt, xoá, hay thực hiện chức năng AND, OR...chỉ thông qua 1 lệnh. Ngoài ra các cổng xuất/nhập cũng có thể được định địa chỉ dạng bit, điều này làm đơn giản việc giao tiếp bằng phần mềm với các thiết bị xuất/nhập đơn bit.

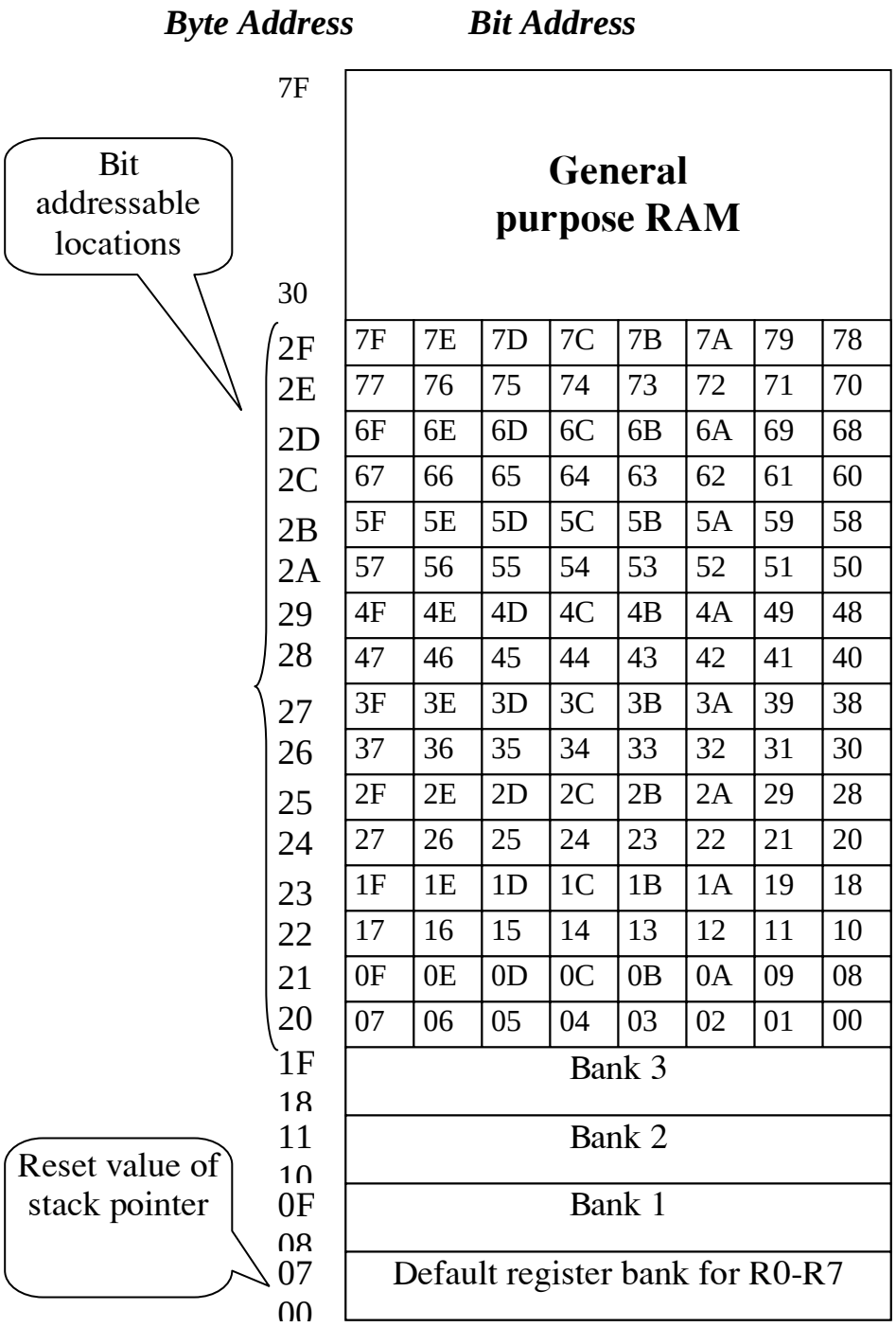
- Vùng nhớ còn lại gồm 80 Byte có địa chỉ từ 30h đến 7Fh được dành riêng cho người sử dụng để lưu trữ dữ liệu. Đây có thể xem là vùng RAM đa mục đích. Có thể truy cập vùng nhớ này bằng địa chỉ trực tiếp hoặc gián tiếp thông qua các thanh ghi (R0 hoặc R1) ở dạng mức Byte.

2.5.3.2.2. Vùng nhớ dành cho SFR

Vùng nhớ này được định địa chỉ từ 80h đến FFh, và được truy cập bằng địa chỉ trực tiếp.

2.5.3.2.3. Các lệnh truy cập bộ nhớ dữ liệu nội trú.

- MOV A, <src>: Chuyển dữ liệu từ toán hạng nguồn (các ô nhớ, thanh ghi có địa chỉ trực tiếp hoặc gián tiếp trong on chip, các giá trị trực hằng chứa trong câu lệnh) vào thanh ghi tích luỹ.
- MOV <dest>, <src>: Chuyển dữ liệu từ toán hạng nguồn vào toán hạng đích (các ô nhớ, thanh ghi có địa chỉ trực tiếp hoặc gián tiếp trong on chip).
- MOV <dest>, A : Chuyển dữ liệu từ A vào toán hạng đích.
- MOV DPTR, #data16: Chuyển giá trị hằng 16 bit vào thanh ghi con trỏ dữ liệu.
- PUSH <src>: Chuyển giá trị từ toán hạng nguồn vào ngăn xếp.
- POP <dest>: Chuyển giá trị từ ngăn xếp vào toán hạng đích.
- XCH A, <byte>: Chuyển đổi dữ liệu giữa toán hạng nguồn dạng byte với A.
- XCHD A, @Ri: Chuyển đổi nửa thấp của A với nội dung trong RAM tại địa chỉ là nội dung của Ri.



Hình 2.10. 128 Byte thấp của RAM trong

Byte address	***Bit address***								
FF									
F0	F7	F6	F5	F4	F3	F2	F1	F0	B
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
D0	D7	D6	D5	D4	D3	D2	-	D0	PSW
B8	-	-	-	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
A8	AF	-	-	AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	Not bit addressable								SBUF
98	9F	9E	9D	9C	9B	9A	99	98	SCON
90	97	96	95	94	93	92	91	90	P1
8D	Not bit addressable								TH1
8C	Not bit addressable								TH0
8B	Not bit addressable								TL1
8A	Not bit addressable								TL0
89	Not bit addressable								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	Not bit addressable								PCON
83	Not bit addressable								DPH
82	Not bit addressable								DPL
81	Not bit addressable								SP
80	87	86	85	84	83	82	81	80	P0

Hình 2.11. Các thanh ghi chức năng đặc biệt (SFR)

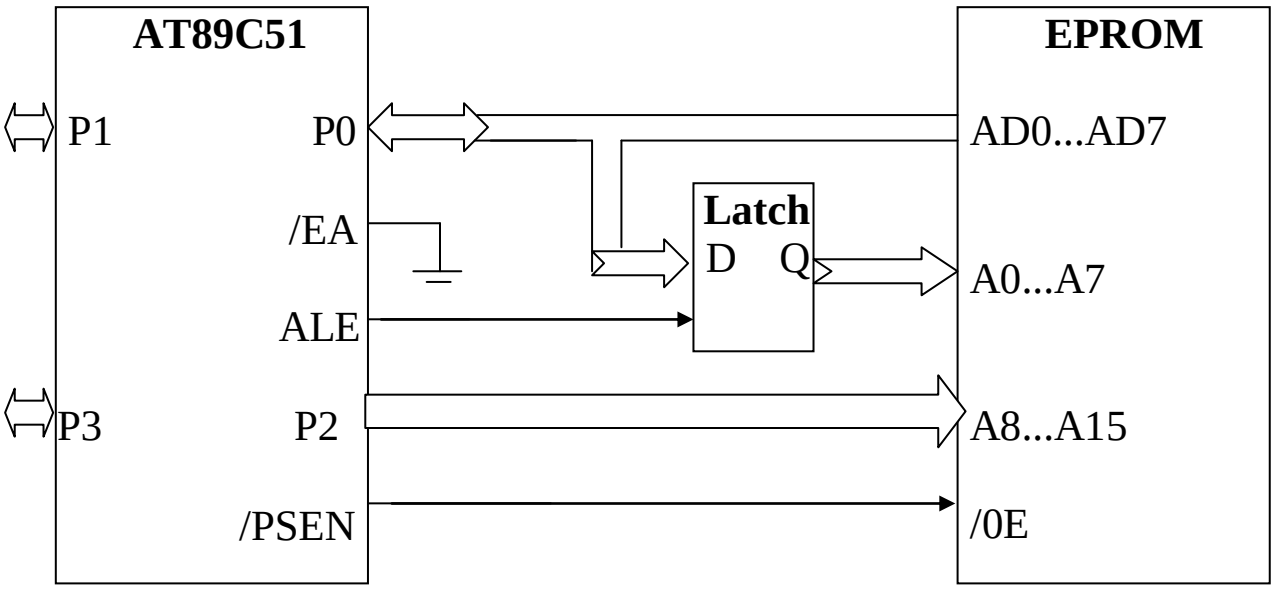
2.5.4. Bộ nhớ chương trình và bộ nhớ dữ liệu ngoại trú.

Để tăng khả năng ứng dụng trong các lĩnh vực điều khiển, đo lường… Bộ VĐK cho phép mở rộng không gian nhớ RAM ngoài đến 64 Kbyte và ROM ngoài đến 64 Kbyte khi cần thiết. Các IC giao tiếp ngoại vi cũng có thể được thêm vào để mở rộng khả năng xuất/nhập và chúng trở thành 1 phần của không gian nhớ dữ liệu ngoài.

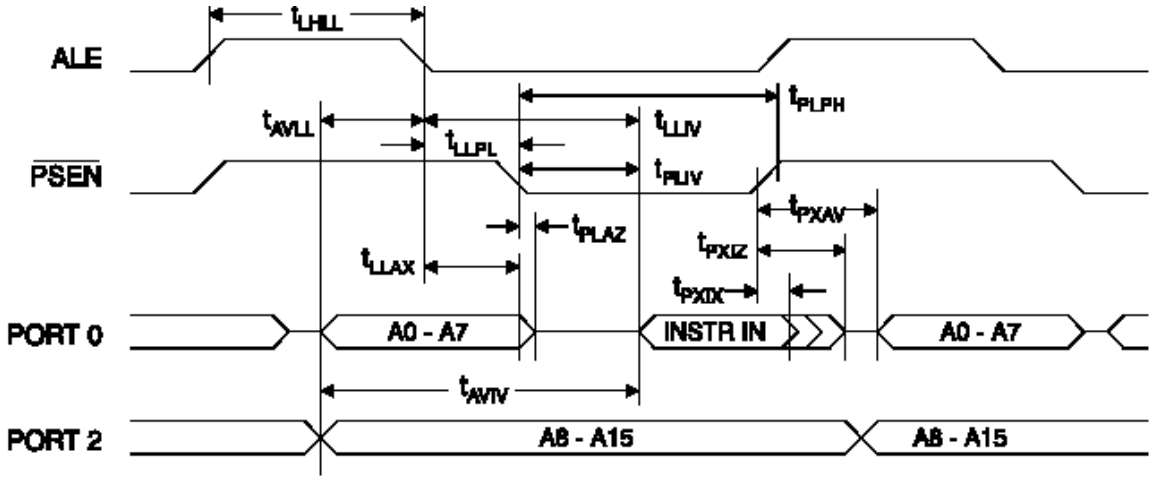
Khi bộ nhớ ngoài được sử dụng, cổng P0 không còn đảm nhận chức năng xuất/nhập nữa, mà nó trở thành kênh địa chỉ (A0…A7) và kênh dữ liệu đa hợp (D0…D7). Ngõ ra ALE chốt byte thấp của địa chỉ ở thời điểm bắt đầu của mỗi 1 chu kỳ bộ nhớ ngoài. Cổng P2 thường được dùng làm byte cao của kênh địa chỉ.

Hoạt động của các bộ nhớ ngoài thường được thực hiện theo kiểu sắp xếp đa hợp, nghĩa là: trong nửa chu kỳ đầu của chu kỳ bộ nhớ, byte thấp của địa chỉ được cung cấp bởi cổng P0 và được chốt nhờ tín hiệu ALE. Mạch chốt giữ cho byte thấp của địa chỉ ổn định trong cả chu kỳ bộ nhớ. Trong nửa chu kỳ sau của bộ nhớ, cổng P0 được sử dụng làm kênh dữ liệu, lúc này dữ liệu có thể được đọc hoặc ghi.

2.5.4.1. Bộ nhớ chương trình ngoại trú.



Hình 2.12. Truy cập bộ nhớ chương trình ngoài



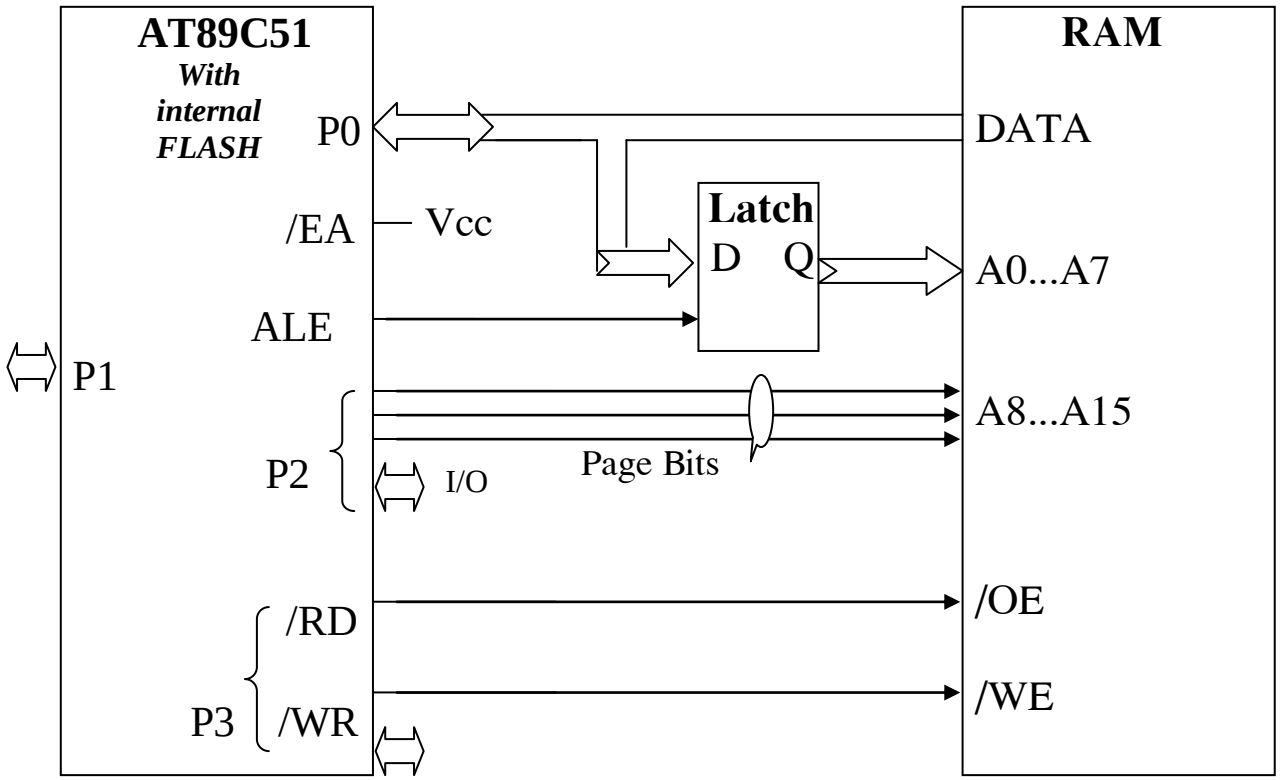
Hình 2.13. Đồ thị thời gian quá trình nhận lệnh từ ROM ngoài

Bộ nhớ chương trình ngoài là bộ nhớ chỉ đọc, được cho phép bởi tín hiệu /PSEN. Khi có một EPROM ngoài được sử dụng, cả P0 và P2 đều không còn là các cổng I/O nữa. Khi bộ VĐK truy cập bộ nhớ chương trình ngoại trú, nó luôn sử dụng kênh địa chỉ 16 bit thông qua P0 và P2.

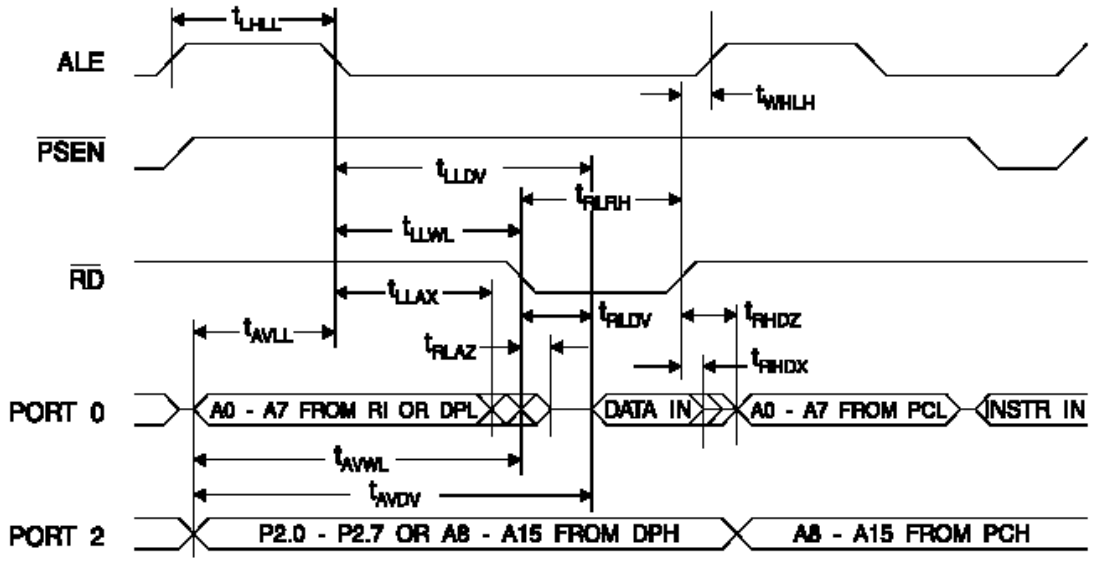
Một chu kỳ máy của bộ VĐK có 12 chu kỳ dao động. Nếu bộ dao động trên chip có tần số 12 MHz, thì 1 chu kỳ máy dài 1μs. Trong một chu kỳ máy điển hình, ALE có 2 xung và 2 Byte của lệnh được đọc từ bộ nhớ chương trình (nếu lệnh chỉ có 1 byte thì byte thứ 2 được loại bỏ). Khi truy cập bộ nhớ chương trình ngoại trú, bộ VĐK phát ra 2 xung chốt địa chỉ trong mỗi chu kỳ máy. Mỗi xung chốt tồn tại trong 2 chu kỳ dao động từ P2-S1 đến P1-S2, và từ P2-S4 đến P1-S5.

Để địa chỉ hoá bộ nhớ chương trình ngoại trú, byte thấp của địa chỉ (A0…A7) từ bộ đếm chương trình của bộ VĐK được xuất qua cổng P0 tại các trạng thái S2 và S5 của chu kỳ máy, byte cao của địa chỉ (A8…A15) từ bộ đếm chương trình được xuất qua cổng P2 trong khoảng thời gian của cả chu kỳ máy. Tiếp theo xung chốt, bộ VĐK phát ra xung chọn /PSEN. Mỗi chu kỳ máy của chu kỳ lệnh gồm 2 xung chọn, mỗi xung chọn tồn tại trong 3 chu kỳ dao động từ P1-S3 đến hết P1-S4 và từ P1-S6 đến hết P1-S1 của chu kỳ máy tiếp theo. Trong khoảng thời gian phát xung chọn thì byte mã lệnh được đọc từ bộ nhớ chương trình để nhập vào On chip.

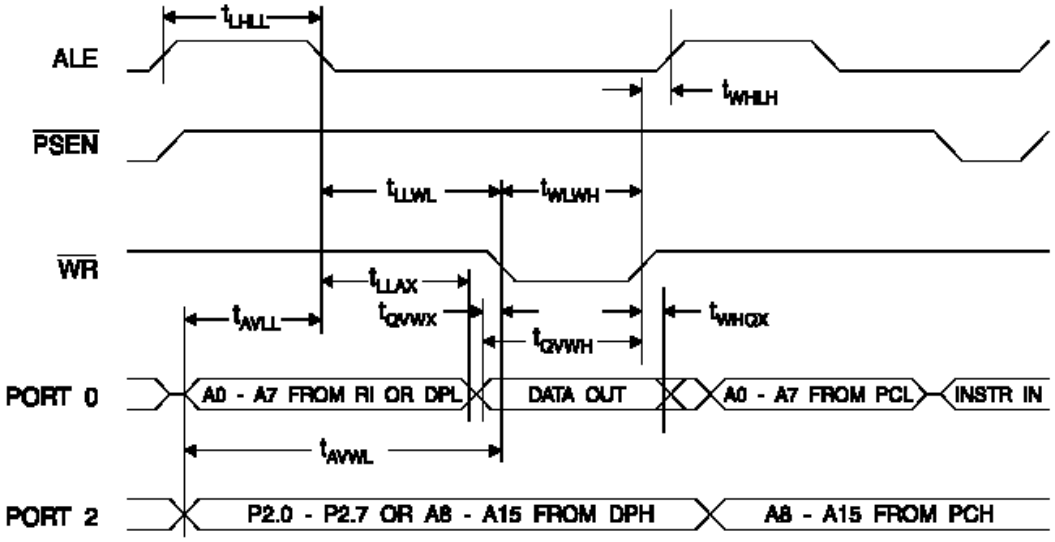
2.5.4.2. Bộ nhớ dữ liệu ngoại trú.



Hình 2. 14. Truy cập bộ nhớ dữ liệu ngoài



Hình 2.15. Đồ thị thời gian chu kỳ đọc dữ liệu từ RAM ngoài



Hình 2.16. Đồ thị thời gian chu kỳ ghi dữ liệu vào RAM ngoài

Bộ nhớ dữ liệu ngoại trú được cho phép bởi các tín hiệu /WR và /RD ở các chân P3.6 và P3.7. VĐK truy cập bộ nhớ dữ liệu ngoài bằng địa chỉ 2 byte (thông qua cổng P0 và P2) hoặc 1 byte (thông qua cổng P0). Lệnh dùng để truy xuất bộ nhớ dữ liệu ngoài là MOVX, sử dụng hoặc DPTR hoặc Ri (R0 và R1) làm thanh ghi chứa địa chỉ.

Ở hình 2.14 ta thấy:

- /EA được nối với +Vcc để cho phép VĐK làm việc với bộ nhớ chương trình nội trú.
- /RD nối với đường cho phép xuất dữ liệu (/OE-Output Data Enable) của RAM.
- /WR nối với đường cho phép ghi dữ liệu (/WE-Write Data Enable) của RAM.

Nguyên lý truy cập bộ nhớ dữ liệu ngoại trú được thể hiện bằng các đồ thị thời gian ở trên. Tuy nhiên, tuỳ thuộc vào nhiệm vụ đọc dữ liệu từ bộ nhớ hay ghi dữ liệu vào bộ nhớ mà nguyên lý truy cập bộ nhớ dữ liệu là khác nhau.

**** Quá trình đọc dữ liệu từ bộ nhớ ngoại trú:*** Khi truy cập bộ nhớ dữ liệu ngoại trú, bộ VĐK *phát ra 1 xung chốt địa chỉ* (ALE) cho bộ chốt bên ngoài (Latch) trong mỗi chu kỳ máy, tồn tại trong 2 chu kỳ dao động từ P2-S4 đến P1-

S5. Để địa chỉ hoá bộ nhớ dữ liệu ngoài, byte thấp của địa chỉ từ thanh ghi con trỏ dữ liệu (DPL) hoặc từ Ri của VĐK được xuất qua cổng P0 trong khoảng các trạng thái S5 của chu kỳ máy trong chu kỳ lệnh. Tiếp theo byte thấp của địa chỉ từ bộ đếm chương trình (PCL) cũng được xuất ra qua cổng P0 đưa tới bộ đếm chương trình để thực hiện lệnh tiếp theo. Byte cao của địa chỉ từ DPTR (DPH) của VĐK được xuất qua cổng P2 trong khoảng thời gian từ S5 đến S4 của chu kỳ máy tiếp theo. Sau đó byte cao của địa chỉ từ PC (PCH) cũng được xuất qua cổng P2 để đưa đến bộ nhớ chương trình. Nếu địa chỉ có độ dài 1 byte thì nó được xuất qua cổng P0 từ DPL hoặc Ri. Tiếp theo xung chốt, VĐK xuất ra tín hiệu điều khiển /RD để cho phép đọc dữ liệu từ bộ nhớ ngoài. Xung /RD tồn tại trong 3 trạng thái của mỗi chu kỳ máy từ P1-S1 đến P2-S3, và trong khoảng thời gian này dữ liệu từ bộ nhớ ngoài được đọc vào VĐK .

**** Quá trình ghi dữ liệu vào bộ nhớ ngoại trú:*** Tương tự như quá trình đọc dữ liệu, nhưng ở đây dùng tín hiệu điều khiển ghi /WR.

**** Các lệnh truy cập bộ nhớ dữ liệu ngoại trú:***

- MOVX A, @Ri: Chuyển (đọc) dữ liệu 8 bit từ ô nhớ của RAM ngoài tại địa chỉ được xác đinh trong thanh ghi của băng thanh ghi hiện hành vào A.

- MOVX @Ri, A: Chuyển (ghi) dữ liệu 8 bit từ A vào ô nhớ của RAM ngoài tại địa chỉ được xác định trong thanh ghi của băng thanh ghi hiện hành.

- MOVX A,@DPTR: Chuyển (đọc) dữ liệu 16 bit từ ô nhớ của RAM ngoài tại địa chỉ được xác đinh trong thanh ghi con trỏ dữ liệu vào A.

- MOVX @DPTR, A: Chuyển (ghi) dữ liệu 16 bit từ A vào ô nhớ của RAM ngoài tại địa chỉ được xác định trong thanh ghi con trỏ dữ liệu.

Ví dụ: MOV R0, #4Fh
MOVX A,@R0

Sẽ chuyển nội dung ở RAM ngoài tại địa chỉ 4Fh vào A.

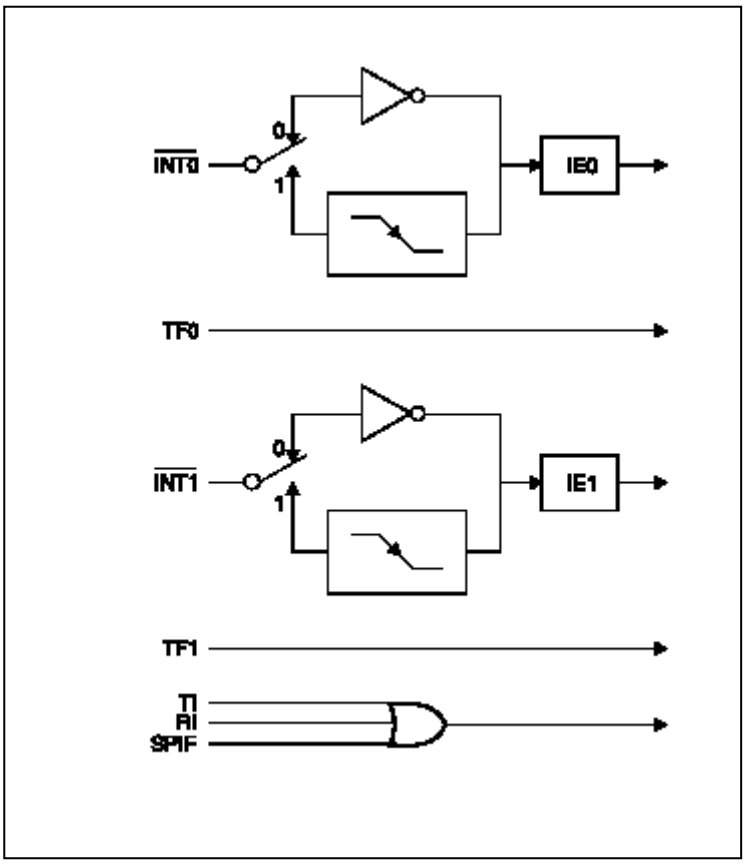
2.5.5. Cơ chế ngắt trong On-chip AT89C51:

2.5.5.1- Phân loại ngắt trong On-chip:

Bộ AT89C51 có tất cả 5 Vectors ngắt bao gồm: 2 ngắt ngoài (/INT0 và /INT1), 2 ngắt của khối thời gian (Timer 0, 1), và ngắt cổng truyền tin nối tiếp.

Mỗi nguồn ngắt có thể được kích hoạt hoặc không kích hoạt bằng cách đặt hoặc xoá Bit ở trong IE. IE cũng chứa bit có thể không cho tất cả các ngắt hoạt động EA (Nếu EA=0). Các ngắt ngoài có thể được kích hoạt theo mức hoặc theo sườn xung, tuỳ thuộc vào giá trị của các bit IT0, IT1 trong TCON. Ngắt ngoài có 2 cờ ngắt tương ứng là IE0, IE1 cũng nằm trong TCON. Khi một ngắt được thực

hiện thì cờ ngắt tương ứng của nó bị xoá bằng phần cứng. Chương trình con phục vụ ngắt hoạt động chỉ khi ngắt được kích hoạt theo sườn xung. Nếu ngắt được kích hoạt theo mức thì nguồn yêu cầu ngắt từ bên ngoài điều khiển cờ ngắt.



Hình 2.17. Các nguồn ngắt của AT89C51

Các ngắt trong, với ngắt Timer/Counter 0, 1 được phát sinh bởi cờ ngắt TF0, TF1. Hai cờ ngắt này được thiết lập khi thanh ghi Timer/Counter thực hiện quay vòng, tại thời điểm S5P2 của chu trình máy. Khi một ngắt được thực hiện thì cờ ngắt tương ứng phát sinh ra ngắt sẽ bị xoá bằng phần cứng trong On-chip.

Ngắt cổng nối tiếp được phát sinh bởi các ngắt RI, TI, SPIF thông qua phần tử Logic OR, khi chương trình con phục vụ ngắt được kích hoạt thì các cờ ngắt phát sinh tương ứng được xoá bằng phần mềm. Các ngắt trong có thể được phép hoặc không đuợc phép kích hoạt bằng cách đặt hoặc xoá một bit trong IE.

2.5.5.2.Các bước thực hiện ngắt.

Theo đúng trình tự, để sử dụng các ngắt trong Flash Microcontroller, cần thực hiện các bước như sau:

- Đặt bit EA ở trong IE mức logic 1.
- Đặt bit cho phép ngắt tương ứng ở trong IE mức logic 1.
- Bắt đầu chương trình con phục vụ ngắt tại địa chỉ của ngắt tương ứng đó.

(Xem bảng địa chỉ Vector của các nguồn ngắt)

Ngoài ra, đối với các ngắt ngoài, các chân /INT0, /INT1 phải được đặt mức 1. Và tuỳ thuộc vào ngắt được kích hoạt bằng mức hay sườn xung, mà các bit IT0, IT1 ở trong TCON có thể cần phải đặt mức 1.

ITx=0: Kích hoạt bằng mức

ITx=1: Kích hoạt bằng sườn xung.

2. 5.5.3. Mức ngắt ưu tiên trong on-chip:

Mỗi nguồn ngắt có thể được lập trình riêng cho 1 hoặc 2 mức ưu tiên bằng cách đặt hoặc xoá 1 bit trong IP của SFR. Mỗi ngắt ưu tiên ở mức thấp có thể được ngắt bằng ngắt ưu tiên ở mức cao hơn nhưng không thể ngắt bằng ngắt có mức ưu tiên ở mức thấp hơn được. Một ngắt ưu tiên ở mức cao có thể được ngắt bởi bất kỳ nguồn ngắt nào khác.

Nếu có yêu cầu ngắt của 2 mức ưu tiên cùng nhau (cùng 1 lúc), yêu cầu của mức ưu tiên cao hơn sẽ được phục vụ (Ngắt nào có mức ưu tiên cao hơn sẽ được phục vụ). Nếu các yêu cầu ngắt có cùng mức ưu tiên, thì thứ tự quay vòng bên trong sẽ quyết định ngắt nào được phục vụ.

Thứ tự ưu tiên ngắt từ cao xuống thấp của AT89C51 như sau:

IE0, TF0, IE1, TF1, RI hoặc TI.

2.5.5.4. Nguyên lý điều khiển ngắt của AT89:

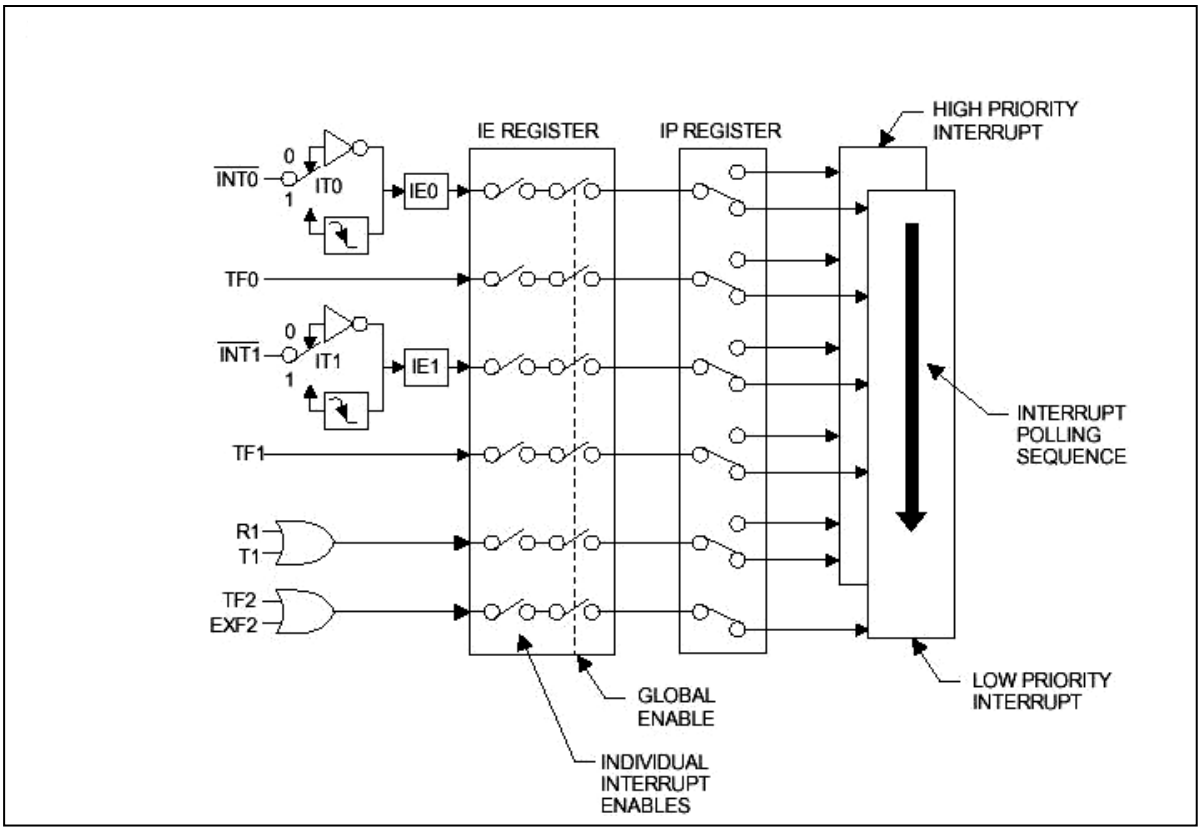
Các cờ ngắt được thiết lập tại thời điểm S5P2 của mỗi chu kỳ máy. Chu kỳ máy tiếp theo sau chu kỳ máy có cờ ngắt được thiết lập, thì chương trình con được thiết lập khi có lệnh gọi LCALL. Lệnh LCALL phát sinh nhưng lại bị cấm hoạt động khi gặp các tình huống sau:

a- Đồng thời có ngắt với mức ưu tiên cao hơn hoặc bằng ngắt đang phục vụ.

(Một ngắt có mức ưu tiên bằng hoặc cao hơn đang sẵn sàng để được phục vụ)

b- Chu kỳ máy hiện hành không phải là chu kỳ máy cuối cùng của lệnh đang thực hiện.

c- Lệnh đang thực hiện là RETI hoặc bất kỳ lệnh nào ghi vào thanh ghi IE hoặc IP.



Hình 2.18. Hệ thống ngắt của AT89

Bất kỳ một trong 3 điều kiện này xuất hiện sẽ cản trở việc tạo ra LCALL đối với chương trình phục vụ ngắt. Điều kiện 2 đảm bảo rằng, lệnh đang thực hiện sẽ được hoàn thành trước khi trỏ tới bất kỳ chương trình phục vụ nào. Điều kiện 3 đảm bảo rằng, nếu lệnh đang thực hiện là RETI hoặc bất kỳ sự truy cập nào vào IE hoặc IP, thì ít nhất một lệnh nữa sẽ được thực hiện trước khi bất kỳ ngắt nào được trỏ tới. Chu trình kiểm tra vòng được lặp lại với mỗi chu trình máy, và các giá trị được kiểm tra là các giá trị mà đã xuất hiện ở thời điểm S5P2 của chu trình máy trước đó. Nếu một chỉ thị ngắt có hiệu lực nhưng không được đáp ứng vì các điều kiện trên và nếu chỉ thị này vẫn chưa có hiệu lực khi điều kiện cản trở được loại bỏ, thì ngắt bị từ chối này sẽ không được phục vụ nữa.

LCALL do phần cứng tạo ra sẽ chuyển nội dung của bộ đếm chương trình vào ngăn xếp (Nhưng không ghi vào PSW) và nạp lại cho PC một địa chỉ phụ thuộc vào nguồn gây ngắt đang được phục vụ, như bảng dưới đây:

Ngắt	**Nguồn ngắt**	**Địa chỉ Véc tơ**
External 0	IE0	0003h
Timer 0	TF0	000Bh
External 1	IE1	0013h
Timer 1	TF1	001Bh
Serial Port	RI hoặc TI	0023h
Timer 2 (AT89C52)	TF2 hoặc EXF2	002Bh
System Reset	RST	0000h

Bảng 2.7. Địa chỉ véc tơ ngắt

Lệnh RETI thông báo cho bộ VXL rằng thủ tục ngắt này đã kết thúc, sau đó lấy ra 2 Byte từ ngăn xếp và nạp lại cho PC để trả lại quyền điều khiển cho chương trình chính.

2.5.5.4.1.Các ngắt ngoài:

Vì các chốt ngắt ngoài được tạo mẫu mỗi lần trong mỗi chu trình máy, nên một giá trị cao hoặc thấp của đầu vào sẽ duy trì trong ít nhất là 12 chu kỳ xung nhịp của bộ dao động để đảm bảo tạo mẫu. Nếu ngắt ngoài được kích hoạt bằng sườn xung , thì nguồn ngắt ngoài phải duy trì ở chốt yêu cầu giá trị cao ít nhất 1 chu kỳ máy và sau đó duy trì giá trị thấp ít nhất 1 chu kỳ máy nữa. Việc này được thực hiện để đảm bảo rằng quá trình chuyển tiếp cho thấy chỉ thị yêu cầu ngắt IEx sẽ được xác lập. IEx sẽ tự động được xoá bởi CPU khi thủ tục ngắt đáp ứng được gọi.

Nếu ngắt ngoài được kích hoạt theo mức, thì nguồn ngắt bên ngoài phải duy trì cho yêu cầu này có hiệu lực cho đến khi ngắt đã được yêu cầu thực sự được tạo ra. Sau đó nguồn ngắt ngoài phải huỷ yêu cầu đó trước khi thủ tục phục vụ ngắt hoàn thành, nếu không ngắt khác sẽ được tạo ra.

2.5.5.4.2. Vận hành Single-Step:

Cấu trúc ngắt AT89C51 cho phép thực hiện các bước đơn với sự tham gia của rất ít phần mềm. Như đã lưu ý trước đây, một yêu cầu ngắt sẽ không được đáp ứng khi một ngắt khác có cùng mức ưu tiên vẫn đang hoạt động, nó cũng không được đáp ứng sau khi có lệnh RETI cho đến khi có ít nhất một lệnh khác đã được thực hiện. Do đó mỗi khi một thủ tục ngắt được đưa vào, thì nó không thể được đưa vào lần nữa cho đến khi ít nhất một lệnh của chương trình ngắt được thực hiện. Một cách để sử dụng đặc điểm này đối với hoạt động theo bước đơn lẻ là lập trình cho 1 trong những ngắt ngoài(chẳng hạn /INT0) được kích hoạt theo mức.

Nếu chân /INT0 được duy trì ở mức thấp, thì CPU sẽ chuyển ngay đến thủ tục ngắt ngoài 0 và dừng ở đó cho tới khi INT0 được nhận xung từ thấp lên cao rồi xuống thấp. Sau đó nó sẽ thực hiện lệnh RETI, trở lại nhiệm vụ chương trình, thực hiện một lệnh, và ngay sau đó nhập lại thủ tục ngfắt ngoài 0 để đợi xung nhịp tiếp theo của P3.2. Mỗi bước của nhiệm vụ chương trình được thực hiện vào mỗi thời điểm chân P3.2 được nhận xung.

2.5.6. Nguyên lý truyền tin nối tiếp của AT89C51:

2.5.6.1. Phương thức truyền tin nối tiếp(Serial Interface):

Hệ VXL on-chip này truyền tin nối tiếp bằng cổng RxD và TxD, dữ liệu xuất nhập truyền qua cổng nối tiếp bằng tốc độ Baud và đều qua vùng đệm nối tiếp SBUF. Cổng truyền nối tiếp là cổng truyền tin 2 chiều, nghĩa là nó có thể đồng thời truyền và nhận thông tin cùng 1 lúc. Nó cũng có khả năng vừa thực hiện chức năng nhận vừa thực hiện chức năng đệm, tức là nó có thể nhận byte kế tiếp trước khi byte được nhận trước đó được đọc từ thanh ghi đệm. (Tuy nhiên, nếu byte đầu tiên vẫn chưa được đọc tại thời điểm nhận của byte thứ 2, thì một trong 2 byte này sẽ bị mất). Điều khiển cổng nối tiếp bằng thanh ghi SCON, trạng thái của 2 bit SM0 và SM1 trong thanh ghi này thiết lập nên 4 chế độ hoạt động giao tiếp nối tiếp chuẩn như sau:

+ ***Chế độ 0***: Dữ liệu nối tiếp vào và ra sẽ thông qua chân RxD. Chân TxD đưa ra xung nhịp đồng hồ. 8 bit dữ liệu được truyền/nhận nối tiếp, với bit LSB được thực hiện đầu tiên. Tốc độ Baud được cố định bằng 1/12 tần số của bộ dao động.

+ ***Chế độ 1***: 10 bit được truyền (thông qua TxD) hoặc nhận (thông qua RxD), trong đó gồm có: 1 bit khởi động (có giá trị 0), 8 bit dữ liệu (đầu tiên là

LSB), và 1 bit dừng (có giá trị là 1). Khi nhận, bit dừng được chuyển vào RB8 của thanh ghi SCON. Tốc độ Baud có thể thay đổi được.

+ ***Chế độ 2***: 11 bit được truyền (thông qua TxD) hoặc nhận (thông qua RxD) bao gồm: bit khởi động (có giá trị 0), 8 bit dữ liệu (đầu tiên là LSB), một bit dữ liệu thứ 9 có thể lập trình được, và một bit dừng (có giá trị 1). Khi truyền, bit dữ liệu thứ 9 (TB8 ở trong SCON) có thể được gán giá trị 0 hoặc 1. Chẳng hạn như bit chẵn lẻ (P ở trong PSW) có thể được chuyển vào TB8. Khi nhận, bit dữ liệu thứ 9 được chuyển vào RB8 ở thanh ghi SCON, trong khi bit dừng được lọc bỏ. Tốc độ Baud có thể lập trình được bằng 1/32 hoặc 1/64 tần số bộ dao động.

+ ***Chế độ 3***: 11 bit được truyền (thông qua TxD) hoặc được nhận (thông qua RxD) bao gồm: 1 bit khởi động (có giá trị 0), 8 bit dữ liệu (đầu tiên là LSB), 1 bit dữ liệu thứ 9 có thể lập trình được, và 1 bit dừng (có giá trị 1). Trên thực tế, chế độ 3 giống chế độ 2 ở mọi góc độ trừ tốc độ Baud. Tốc độ Baud ở chế độ 3 là khả biến và được xác định theo bộ Timer 1.

Trong cả 4 chế độ trên, việc truyền được bắt đầu bởi bất kỳ một lệnh nào mà sử dụng thanh ghi SBUF như là một thanh ghi đích. Việc nhận được bắt đầu ở chế độ 0 khi RI=0 và REN=1. Đối với các chế độ khác, việc nhận được bắt đầu khi bit REN=1.

2.5.6.2. Liên lạc đa xử lý (Multiprocessor Communications):

Chế độ 2 và 3 có một dự trù (chuẩn bị) đặc biệt cho các liên lạc đa xử lý. Trong các chế độ này 9 bit dữ liệu được sử dụng. Bit thứ 9 sẽ chuyển vào RB8, sau đó là 1 bit dừng. Cổng nối tiếp có thể được lập trình để thoả mãn điều kiện: khi bit dừng đực nhận thì ngắt của cổng nối tiếp được kích hoạt chỉ khi RB8=1. Đặc điểm này có thể thực hiện được bằng cách đặt bit SM2 ở trong SCON.

Ví dụ dưới đây cho thấy, cách thức sử dụng ngắt cổng truyền nối tiếp để tạo liên lạc đa xử lý. Khi bộ xử lý chủ (Master) muốn truyền 1 khối dữ liệu tới một trong những bộ xử lý (Slave) khác, đầu tiên nó gửi đi 1 byte địa chỉ để xác định địa chỉ của bộ xử lý đích (Slave). Một byte địa chỉ khác với một byte dữ liệu ở chỗ: bit thứ 9 bằng 1 ở byte địa chỉ và bằng 0 ở byte dữ liệu. Với SM2=1, không có bộ xử lý (Slave) nào được ngắt bởi 1 byte dữ liệu. Tuy nhiên 1 byte địa chỉ sẽ ngắt tất cả các bộ xử lý (Slave) khác, để cho mỗi bộ xử lý (slave) khác có thể kiểm tra byte nhận được và để xem có phải nó đang được trỏ tới không. Bộ xử lý

(slave) nào được trỏ tới sẽ xoá (clear) bit SM2 của nó và chuẩn bị nhận các byte dữ liệu sẽ đưa đến. Các bộ xử lý (Slave) khác nếu không được trỏ tới, thì sẽ thiết lập (set) bit SM2 của chúng và tiếp tục hoạt động của mình mà không cần quan tâm tới dữ liệu trên kênh.

Bit SM2 không có tác dụng ở chế độ 0, nhưng nó có thể được sử dụng để kiểm tra bit dừng trong chế độ 1. Trong quá trình nhận tin ở chế độ 1, nếu SM2=1 thì ngắt nhận tin sẽ không được kích hoạt trừ khi bit dừng được nhận vào.

2.5.6.3. Các tốc độ Baud:

+ Tốc độ Baud ở chế độ 0 được cố định, và bằng ***Tần số bộ dao động/12***

+ Tốc độ Baud ở chế độ 2 phụ thuộc vào giá trị của bit SMOD trong thanh ghi PCON. Nếu SMOD=0 (giá trị sau khi reset), thì tốc độ Baud =1/64 tần số của bộ dao động. Nếu SMOD=1 thì tốc độ Baud =1/32 tần số của bộ dao động.

$$\textit{Tốc độ Baud chế độ 2} = (2^{SMOD} * \textit{Tần số bộ dao động})/64$$

Trong AT89C51, các tốc độ Baud ở chế độ 1 và 3 do Timer 1 quyết định, Trong AT89C52 tốc độ Baud của các chế độ này có thể được quyết định bởi Timer 1 hoặc Timer 2, hoặc cả hai (một bộ timer xác định tốc độ truyền, bộ kia xác định tốc độ nhận).

2.5.6.4. Sử dụng Timer 1 để tạo ra các tốc độ Baud :

Khi bộ Timer 1 được dùng để tạo tốc độ Baud, thì các tốc độ Baud ở các chế độ 1 và 3 do tốc độ tràn của timer 1 và giá trị của SMOD quyết định:

$$\textit{Tốc độ Baud ở chế độ 1 và 3} = (2^{SMOD} * \textit{(Tốc độ tràn của timer 1)})/32$$

Ngắt của Timer 1 sẽ mất tác dụng trong ứng dụng này.

Bản thân bộ Timer có thể được thiết lập để thực hiện chức năng thời gian hay bộ đếm ở bất kỳ một trong 3 chế độ hoạt động. Trong hầu hết các kiểu ứng dụng, nó thường được thiết lập để thực hiện chức năng thời gian, hoạt động ở chế độ Auto-reload (nửa byte cao của TMOD = 0010b). Trong trường hợp này, tốc độ baud được tính bằng công thức:

$$\textit{Tốc độ Baud chế độ 1 và 3} = (2^{SMOD} * \textit{Tần số bộ dao động})/(32*(12*[256-(TH1)])$$

Ta có thể nhận được các tốc độ Baud rất thấp với bộ Timer 1 bằng cách làm cho ngắt của timer 1 có tác dụng, và thiết lập Timer 1 để hoạt động như một bộ đếm thời gian 16 bit (Nửa byte cao của TMOD=0001b). Bảng 2.8 liệt kê các tốc độ Baud khác nhau thường được sử dụng và cách chúng có thể nhận được từ Timer 1.

Tốc độ Baud (Hz)	**Tần số d.động (MHz)**	**SMODE**	**Timer** 1		
			C/(/T)	***Mode***	***Giá trị nạp lại***
Mode 0 Max: 1M	12	x	X	X	X
Mode 2 Max: 375K	12	1	X	X	X
Mode 1,3 Max:62,5K	12	1	0	2	FFh
19,2K	11,059	1	0	2	FDh
9,6K	11,059	0	0	2	FDh
4,8K	11,059	0	0	2	FAh
2,4K	11,059	0	0	2	F4h
1,2K	11,059	0	0	2	E8h
137,5	11,966	0	0	2	1Dh
110	6	0	0	2	72h
110	12	0	0	1	FEEBh

Bảng 2.8. Các tốc độ Baud được tạo ra khi sử dụng Timer 1

2.5.6.5. Hoạt động của chế độ 0:

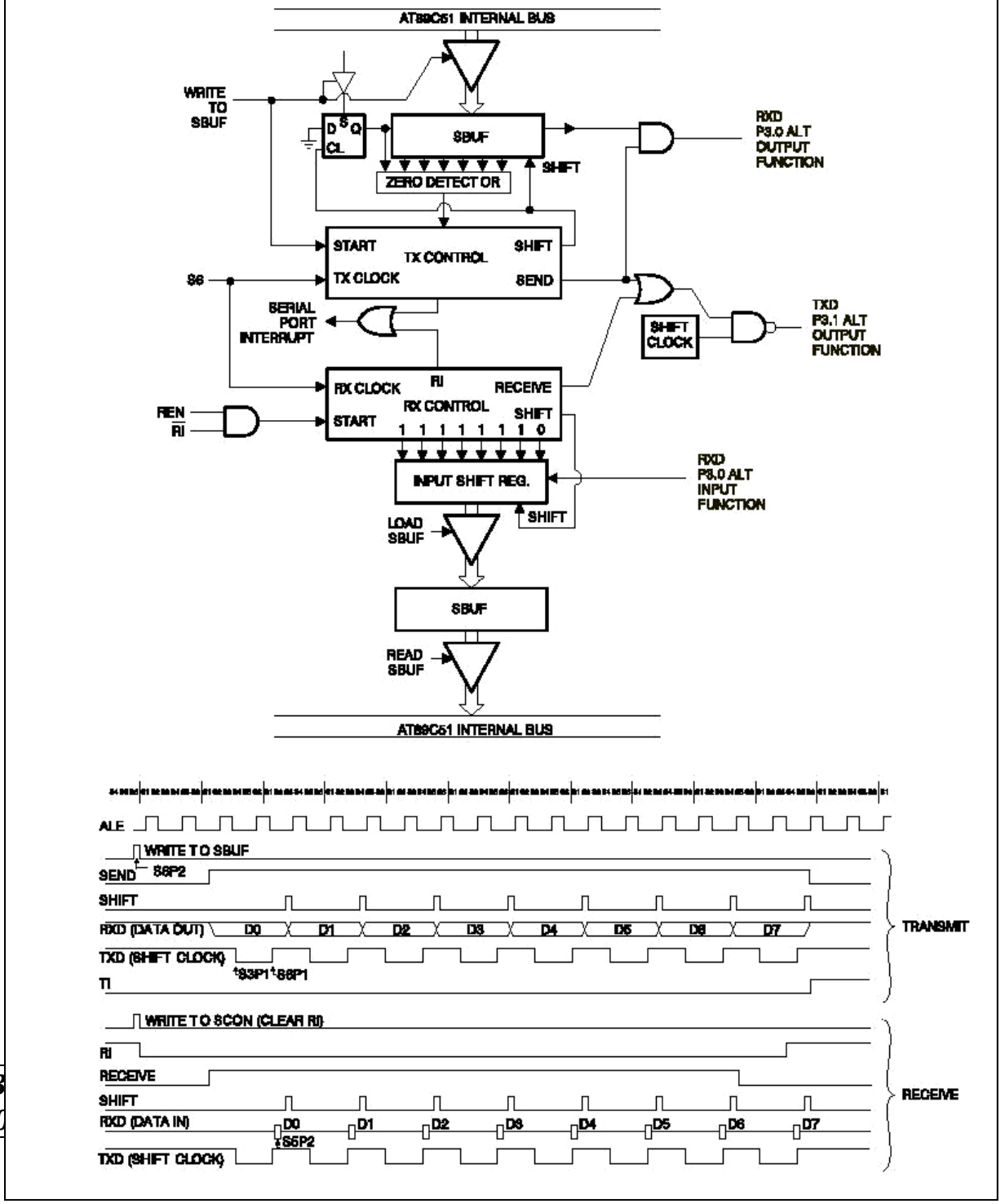
Dữ liệu nối tiếp vào và ra thông qua RxD. TxD cho ra đồng hồ xung nhịp. 8 bit dữ liệu được truyền/nhận (với LSB đầu tiên) được thực hiện ở chế độ này. Tốc độ Baud được cố định bằng 1/12 tần số bộ dao động.

Hình 2.19 (Seriel Port Mode 0) mô tả sơ đồ chức năng của cổng nối tiếp ở chế độ 0 và các mốc thời gian có liên quan. Quá trình truyền được bắt đầu bằng bất kỳ lệnh nào mà sử dung SBUF như là một thanh ghi đích. Tín hiệu "ghi vào SBUF" tại thời điểm S6P2 cũng nạp giá trị 1 vào vị trí thứ 9 của thanh ghi dịch

trong quá trình truyền và bật cờ báo cho khối điều khiển phát (Tx Control) về yêu cầu truyền tin. Thời gian được xác lập bên trong cho 1 chu trình máy đầy đủ sẽ bắt đầu từ thời điểm "ghi vào SBUF" cho tới khi SEND được kích hoạt.

SEND cho phép nội dung của thanh ghi dịch đưa tới đầu ra P3.0 và cho phép tín hiệu SHIFT CLOCK đến đầu ra P3.1. SHIFT CLOCK có giá trị thấp trong các trạng thái S3, S4 và S5 của mỗi chu trình máy, và có giá trị cao trong các trạng thái S6, S1 và S2. Tại thời điểm S6P2 của mỗi chu trình máy khi SEND có mức tích cực, thì nội dung của thanh ghi dịch phát được dịch sang bên phải một bit.

Hình 2.19. Serial Port Mode 0



Khi các bit dữ liệu dịch sang bên phải để đi ra ngoài thì các giá trị 0 được gán vào bên trái. Khi bit có trọng số lớn nhất MSB của Byte dữ liệu ở vị trí đầu của thanh ghi dịch, thì giá trị 1 (đã được nạp từ đầu vào vị trí thứ 9) được đặt vào bên trái của MSB, và tất cả các vị trí ở bên trái còn lại của MSB đều chứa giá trị 0. Điều kiện này sẽ chỉ thị cho khối điều khiển phát thực hiện một phép dịch cuối cùng và sau đó huỷ tác dụng của SEND và thiết lập cờ ngắt truyền TI. Cả 2 tác động này xảy ra tại thời điểm S1P1 của chu trình máy thứ 10 kể từ thời điểm "ghi vào SBUF".

Quá trình nhận tin được khởi đầu bằng điều kiện REN=1 và RI=0. Tại thời điểm S6P2 của chu trình máy tiếp theo, khối điều khiển nhận (Rx Control) sẽ ghi các bit 11111110 (Xóa RI) vào thanh ghi dịch nhận, và sẽ kích hoạt RECEIVE trong pha xung nhịp tiếp theo.

RECEIVE cho phép SHIFT CLOCK (đồng hồ xung nhịp) đưa đến đầu ra P3.1. SHIFT CLOCK sẽ tạo ra việc phát tin tại thời điểm S3P1 và S6P1 của mỗi chu trình máy. Tại giai đoạn S6P2 của mỗi chu trình máy khi RECEIVE có mức tích cực thì nội dung của thanh ghi dịch nhận tin được dịch sang trái một vị trí. Giá trị đưa vào từ bên phải là giá trị đã được tạo mẫu ở chân P3.0 tại thời điểm S5P2 của cùng chu trình máy.

Khi các bit dữ liệu được đưa vào từ bên phải, thì các giá trị 1 sẽ đi ra bên trái. Khi giá trị 0 (đã đươc nạp ban đầu vào vị trí tận cùng bên phải) dịch đến vị trí tận cùng bên trái trong thanh ghi dịch, thì nó chỉ thị cho khối điều khiển nhận thực hiện phép dịch cuối cùng và nạp vào SBUF. Tại thời điểm S1P1 của chu trình máy thứ 10 sau thời điểm ghi vào SCON (đã xoá RI), thì RECEIVE được xoá và RI được thiết lập.

2.5.6.6. Hoạt động của chế độ 1:

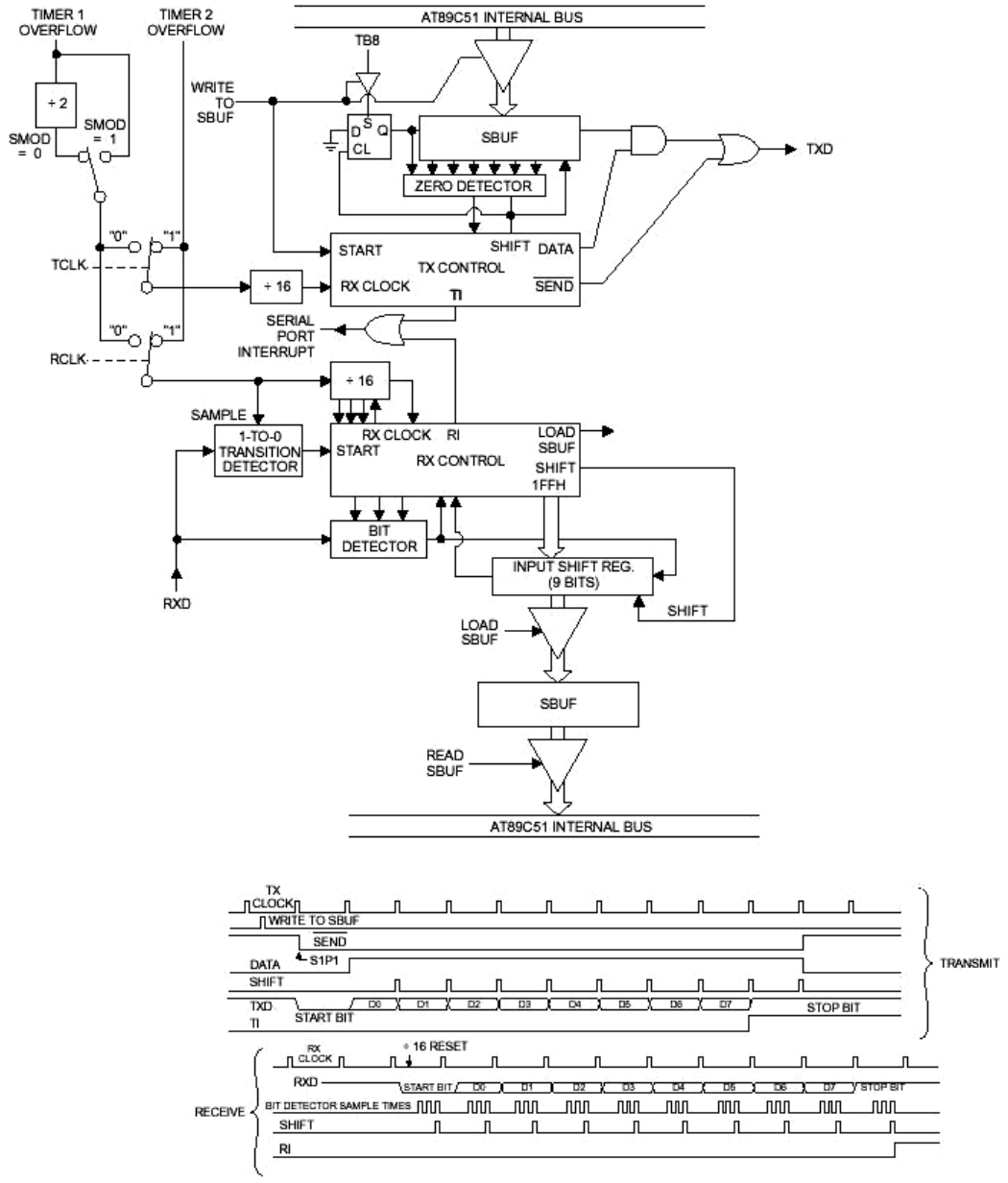
Ở chế độ này 10 bit được truyền (thông qua TxD) hoặc nhận (thông qua RxD) bao gồm: 1 bit khởi đầu(có giá trị 0), 8 bit dữ liệu (LSB đầu tiên) và 1 bit dừng (Có giá trị 1). Khi nhận tin, bit dừng chuyển vào RB8 trong SCON. Trong AT89C51, tốc độ Baud được xác định bằng tốc độ tràn của Timer 1. Trong AT89C52, tốc độ Baud được xác định bằng tốc độ tràn của Timer 1 hoặc tốc độ tràn của Timer 2 hoặc bằng tốc độ tràn của cả 2 bộ Timer này. Trong trường hợp này, khi cả 2 bộ Timer được sử dụng thì một bộ Timer sẽ xác định tốc độ truyền tin, còn bộ Timer kia xác định tốc độ nhận tin.

Hình 2.20 (Seriel Port Mode 1) là sơ đồ chức năng của cổng nối tiếp chế độ 1 và đồ thị thời gian liên quan tới quá trình truyền và nhận tin của chế độ này.

Quá trình truyền tin được khởi đầu bởi bất kỳ lệnh nào có sử dụng SBUF như 1 thanh ghi đích. Tín hiệu "ghi vào SBUF" sẽ nạp giá trị 1 vào bit thứ 9 của thanh ghi dịch truyền và bật cờ báo cho khối điều khiển phát (Tx Control) về yêu cầu cần truyền tin. Quá trình truyền thực tế bắt đầu tại thời điểm S1P1 của chu kỳ máy theo sau quá trình quay vòng (rollover) kế tiếp ở trong bộ đếm chia 16 (-:-16). Do đó, các thời điểm của bit truyền được đồng bộ với nhịp bộ đếm chia16, chứ không phải với tín hiệu "ghi vào SBUF".

Quá trình truyền tin bắt đầu khi /SEND được kích hoạt để mở cổng OR và bit khởi đầu được đặt tại TxD. Sau đó tín hiệu DATA được kích hoạt để mở tiếp cổng AND. Điều này cho phép mở thông đường truyền từ thanh ghi dịch truyền đến đầu ra TxD. Xung nhịp đầu tiên để dịch các bit trong thanh ghi dịch truyền sẽ xuất hiện ngay sau đó.

Hình 2.20 Serial Port Mode 1. TCLK, RCLK and Timer 2 are Present In the AT89C52 Only.



Khi các bit dữ liệu dịch sang phải, thì các giá trị 0 được đưa vào từ bên trái. Khi MSB của Byte dữ liệu ở vị trí đầu ra của thanh ghi dịch thì giá trị 1(ban đầu đã được nạp vào bit thứ 9) sẽ được điền vào ngay bên trái của bit MSB, còn các bit kể từ nó sang trái đều có giá trị 0. Điều kiện này sẽ chỉ thị cho khối điều khiển phát thực hiện lần dịch cuối cùng và sau đó đưa trả /SEND về mức thụ động, đồng thời thiết lập cờ ngắt TI. Thời điểm này rơi vào chu kỳ thứ 10 của bộ đếm chia 16, sau thời điểm "ghi vào SBUF".

Quá trình nhận tin được khởi đầu bằng việc phát hiện có sự chuyển trạng thái từ 1 về 0 ở đường thu nối tiếp RxD. Để phát hiện chính xác, tín hiệu trên RxD được lấy mẫu ở tốc độ gấp 16 lần tốc độ Baud của đường truyền. Khi sự chuyển trạng thái (từ 1 về 0) được phát hiện thì bộ đếm chia 16 được tái xác lập ngay và giá trị 1FFh được ghi vào thanh ghi dịch đầu vào (Input shift register). Việc tái thiết lập bộ đếm chia 16 sẽ đồng nhất thời điểm tràn của nó với các biên (ranh giới) thời gian của bit đang đi tới đầu thu.

Mỗi bit được chia thành 16 phần (States) thời gian bằng nhau (16 phần của bộ đếm). Tại các phần thời gian thứ 7, 8, 9 của mỗi bit, bộ phát hiện bit (Bit Detector) sẽ trích mẫu giá trị của RxD. Giá trị được chấp nhận là giá trị đã có ở ít nhất 2 trong 3 mẫu. Phương pháp này được thực hiện để chống nhiễu đường truyền. Nếu giá trị được chấp nhận đối với bit đầu tiên không phảI là 0 (không phảI bit START), thì các mạch thu được tái xác lập để quay lại chờ một đột biến từ 1 về 0 khác. Nếu bit khởi đầu (START) có giá trị hợp lệ thì dữ liệu được dịch vào thanh ghi dịch đầu vào, và quá trình nhận tin được tiếp tục.

Khi các bit dữ liệu đi vào từ bên phải của thanh ghi dịch, thì các giá trị 1 được dịch ra bên trái của nó. Khi bit khởi đầu đến vị trí tận cùng bên trái của thanh ghi dịch (ở chế độ 1, nó là thanh ghi 9 bit), nó sẽ chỉ thị cho khối điều khiển nhận (Rx Control) thực hiện phép dịch chuyển cuối cùng, rồi nạp SBUF và RB8, và thiết lập RI. Tín hiệu để nạp SBUF và RB8, và để thiết lập RI sẽ được tạo ra khi và chỉ khi các điều kiện sau đây được thoả mãn ở thời điểm xung nhịp cuối cùng được tạo ra:

1. RI=0, và
2. Hoặc SM2=0, hoặc bit STOP nhận được =1.

Nếu một trong hai điều kiện này không được thảo mãn, thì Byte tin nhận được sẽ bị mất. Nếu cả 2 điều kiện được thoả mãn, thì bit dừng chuyển vào RB8, 8 bit dữ liệu chuyển vào SBUF, và RI được kích hoạt. Tại thời điểm này, bất kể các điều kiện trên được thoả mãn hay không, thì khối điều khiển vẫn quay trở lại để tiếp tục chức năng phát hiện đột biến mới từ 1 về 0 trên đường thu tin RxD.

2.5.6.7. Hoạt động của chế độ 2 và 3:

Ở chế độ này 11 bit được truyền đi (thông qua TxD) hoặc nhận vào (thông qua RxD), bao gồm: 1 bit khởi đầu (0), 8 bit dữ liệu (LSB đầu tiên), 1 bit dữ liệu thứ 9 có thể lập trình được và 1 bit dừng (1). Khi truyền tin đi, bit dữ liệu thứ 9 (TB8) có thể được gán giá trị 0 hoặc 1. Khi nhận tin, bit dữ liệu thứ 9 chuyển vào RB8 trong SCON. Tốc độ Baud có thể lập trình được bằng 1/32 hoặc 1/64 tần số

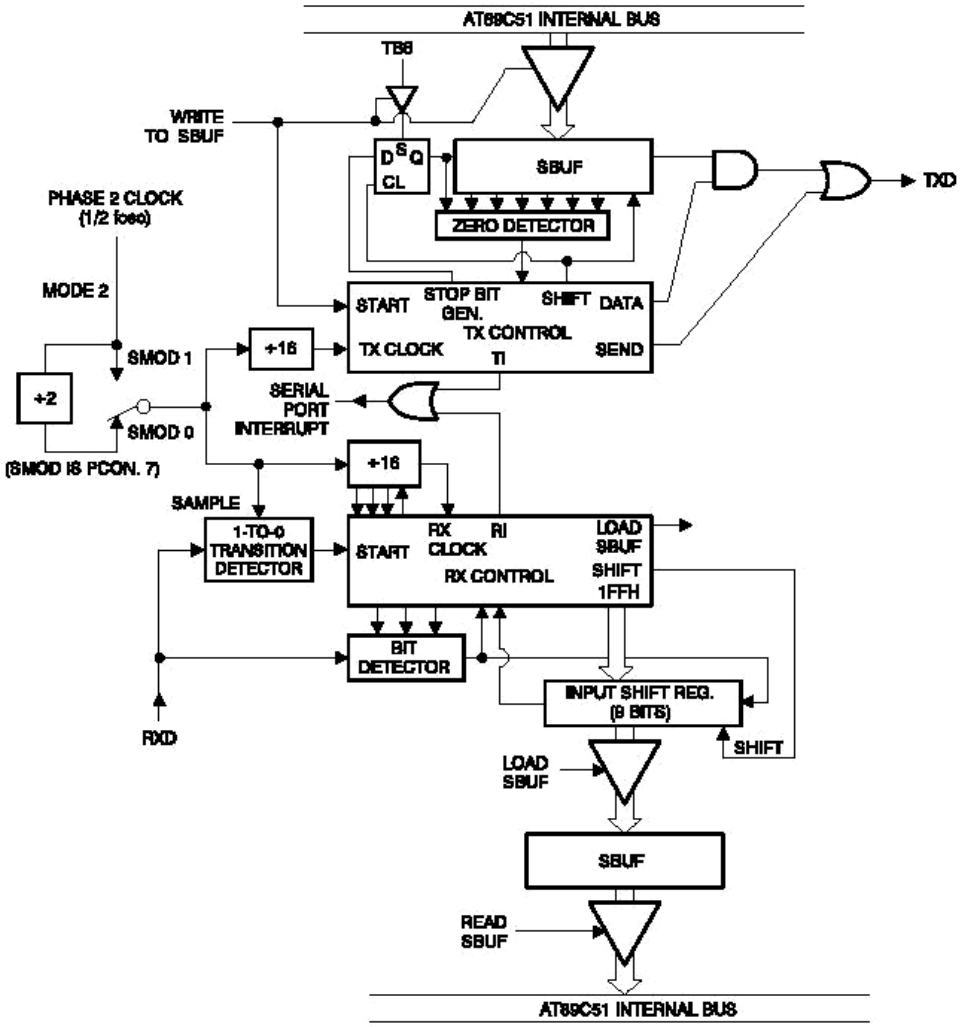
của bộ dao động ở chế độ 2. Chế độ 3 có thể có tốc độ Baud khả biến do Timer 1 hoặc Timer 2 tạo ra, tuỳ thuộc vào trạng thái của các bit TCLK và RCLK.

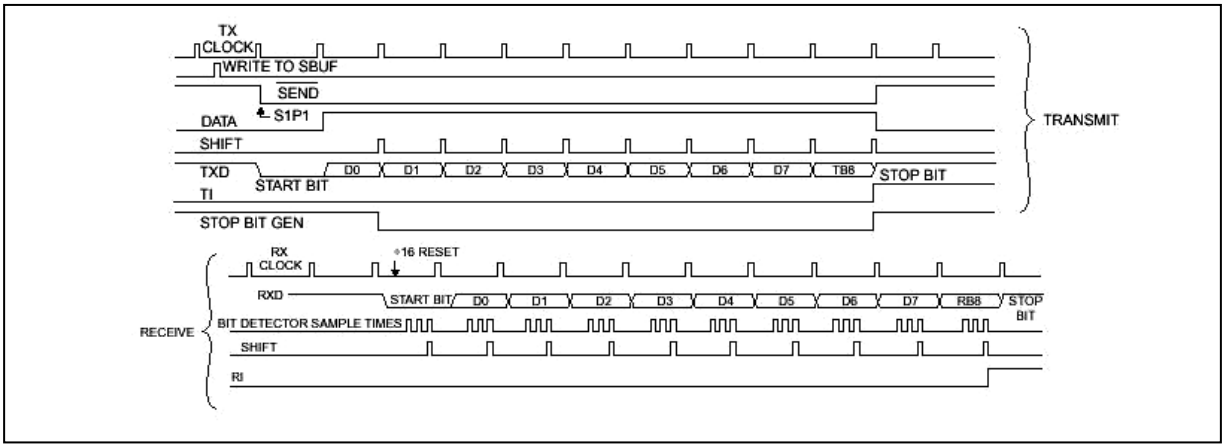
Hình 2.21 (Seriel Port Mode 2) và Hình 2.22 (Seriel Port Mode 3) là các sơ đồ chức năng và đồ thị thời gian của các chế độ 2 và 3. Phần nhận tin được tổ chức giống như chế độ 1. Phần truyền tin khác với chế độ 1 chỉ ở bit thứ 9 của thanh ghi dịch truyền.

Quá trình truyền tin được khởi đầu bằng bất kỳ lệnh nào mà có sử dụng SBUF như một thanh ghi đích. Tín hiệu "ghi vào SBUF" cũng nạp bit TB8 vào vị trí bit thứ 9 của thanh ghi dịch truyền và chỉ thị cho khối điều khiển truyền (Tx Control) rằng có yêu cầu phải truyền tin. Quá trình truyền được bắt đầu tại S1P1 của chu kỳ máy ngay sau thời điểm tràn của bộ đếm chia 16. Do đó, các bit được đồng bộ đối với chu kỳ bộ đếm chia 16, chứ không phải với tín hiệu "ghi vào SBUF".

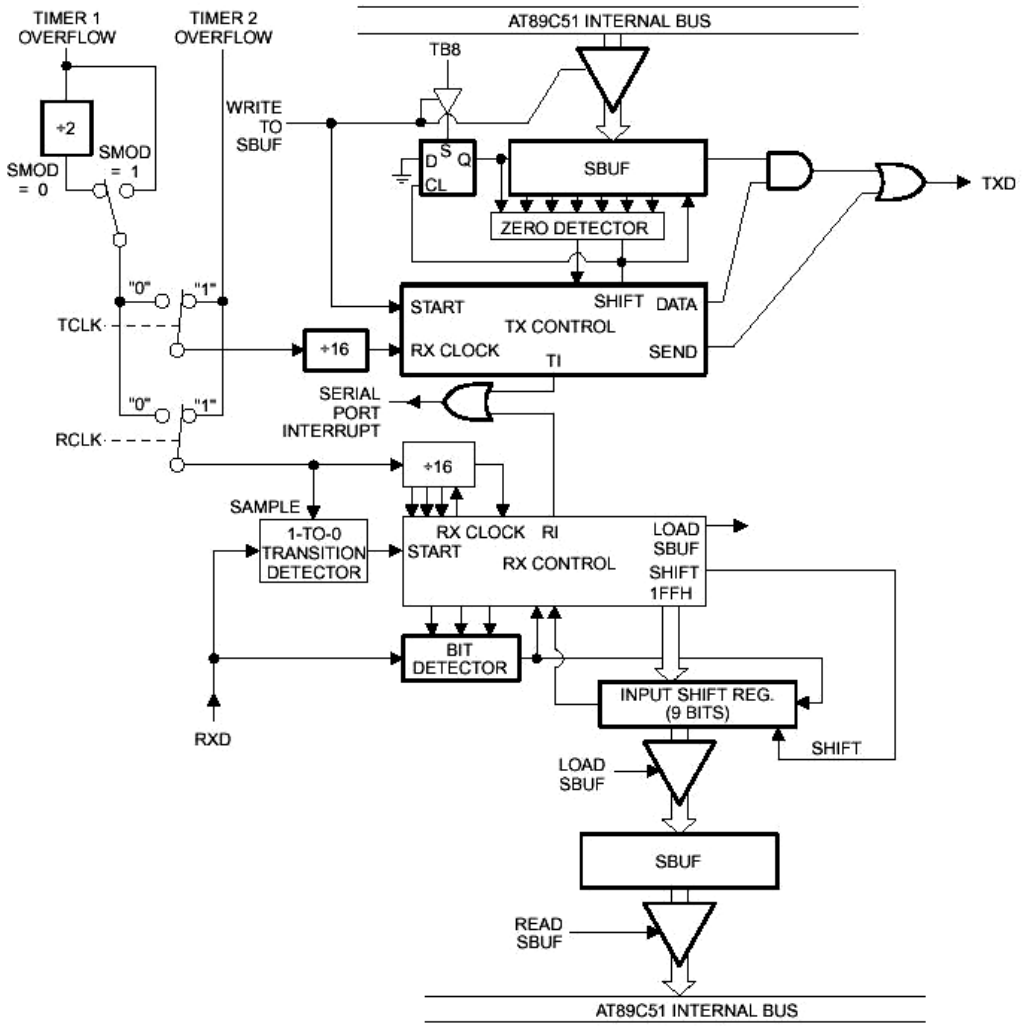
Quá trình truyền bắt đầu khi tín hiệu /SEND được kích hoạt, và bit khởi đầu được đặt tại TxD. Sau đó đến tín hiệu DATA được kích hoạt. Điều này cho phép mở thông đường truyền từ thanh ghi dịch truyền đến đầu ra TxD. Xung nhịp đầu tiên chuyển giá trị 1 (Bit dừng) vào vị trí bit thứ 9 của thanh ghi dịch. Còn sau đó chỉ các giá trị 0 được đưa vào. Vì vậy, khi các bit dữ liệu dịch ra sang phải, thì các giá trị 0 được đưa vào từ bên trái. Khi TB8 ở vị trí đầu ra của thanh ghi dịch, thì bit dừng chuyển đến bên trái của TB8, và tất cả các giá trị ở bên trái của nó

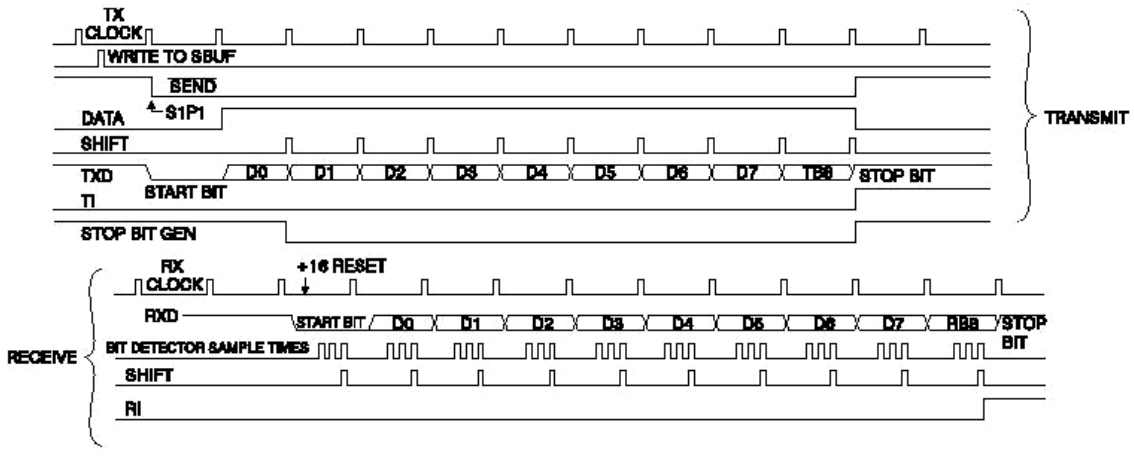
Hình 2.21 Serial Port Mode 2





Hình 2.22 Serial Port Mode 3. TCLK, RCLK, and Timer 2 are Present in AT89C52 only





đều là giá trị 0. Điều kiện này chỉ thị cho khối điều khiển phát (Tx Control) thực hiện phép dịch cuối cùng và sau đó trả SEND về trạng thái thụ động, đồng thời thiết lập TI. Thời điểm này ứng với chu kỳ lần thứ 11 (sự quay vòng lần thứ 11) của bộ đếm chia 16 sau khi có tín hiệu "ghi vào SBUF".

Quá trình nhận tin được khởi đầu bằng sự phát hiện đột biến (chuyển trạng thái) từ 1 về 0 ở RxD. Với mục đích đảm bảo độ tin cậy khi trích mẫu trên RxD, thì tốc độ trích mẫu được lấy gấp 16 lần tốc độ Baud của đường truyền (Tín hiệu trên RxD được lấy mẫu với tốc độ gấp 16 lần tốc độ Baud của đường truyền). Khi sự chuyển trạng thái từ 1 về 0 được phát hiện, bộ đếm chia 16 được tái xác lập (reset) lại ngay, và giá trị 1FFh được ghi vào thanh ghi dịch đầu vào.

Ở các phần thời gian thứ 7, 8 và 9 của mỗi bit, bộ phát hiện bit (Bit Detector) sẽ trích mẫu giá trị của RxD. Giá trị được chấp nhận là giá trị đã thấy ở ít nhất 2 trong 3 mẫu. Phương pháp này được thực hiện để chống nhiễu đường truyền. Nếu giá trị được chấp nhận đối với bit đầu tiên không phải là 0 (không phải bit START), thì các mạch thu được tái xác lập để quay lại chờ một đột biến từ 1 về 0 khác. Nếu bit khởi đầu (START) có giá trị hợp lệ thì dữ liệu được dịch vào thanh ghi dịch đầu vào, và quá trình nhận tin được tiếp tục.

Khi các bit dữ liệu thu được đi vào từ bên phải của thanh ghi dịch, thì các giá trị 1 được dịch ra bên trái của nó. Khi bit khởi đầu đến vị trí tận cùng bên trái của thanh ghi dịch (ở chế độ 2 và 3, nó là thanh ghi 9 bit), nó sẽ chỉ thị cho khối điều khiển nhận (Rx Control) thực hiện phép dịch chuyển cuối cùng, rồi nạp SBUF và RB8, và thiết lập RI. Tín hiệu để nạp SBUF và RB8, và để thiết lập RI sẽ được tạo

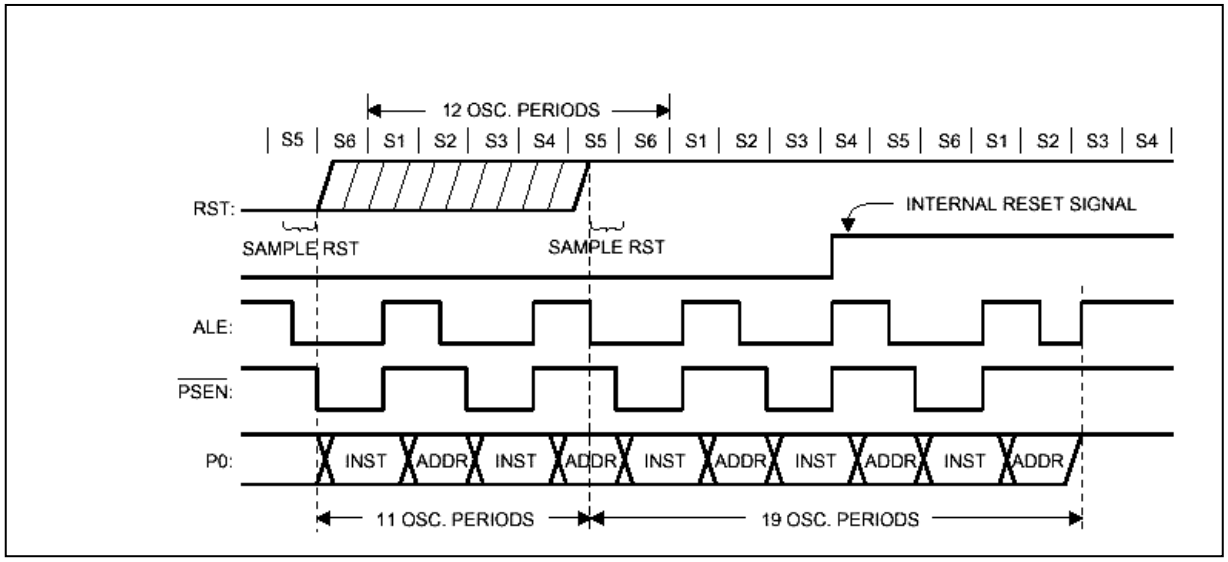
ra khi và chỉ khi các điều kiện sau đây được thoả mãn ở thời điểm xung nhịp cuối cùng được tạo ra:

1. RI=0, và
2. Hoặc SM2=0, hoặc bit STOP nhận được =1

Nếu một trong hai điều kiện này không được thảo mãn, thì Byte tin nhận được sẽ bị mất và RI cũng không được xác lập. Nếu cả 2 điều kiện được thoả mãn, thì bit dừng chuyển vào RB8, 8 bit dữ liệu chuyển vào SBUF, và RI được kích hoạt. Tại thời điểm này, bất kể các điều kiện trên được thoả mãn hay không, thì khối điều khiển vẫn quay trở lại để tiếp tục chức năng phát hiện đột biến mới từ 1 về 0 trên đường thu tin RxD.

2.5.7. Nguyên lý khởi động của On-chip AT89C51:

RST là chân Reset, chính là đầu vào của Trigger Schmitt. Việc Reset được thực hiện bằng cách giữ chân RST ở mức cao ít nhất 2 chu kỳ máy(24 chu kỳ dao động), trong khi bộ dao động đang làm việc. Khi đó on-chip sẽ được khởi động lại (nhờ mạch reset bên trong nó) theo đồ thị thời gian (Hình 2.23. Reset Timing).



Hình 2.23. Đặt lại thời gian cho AT89C51

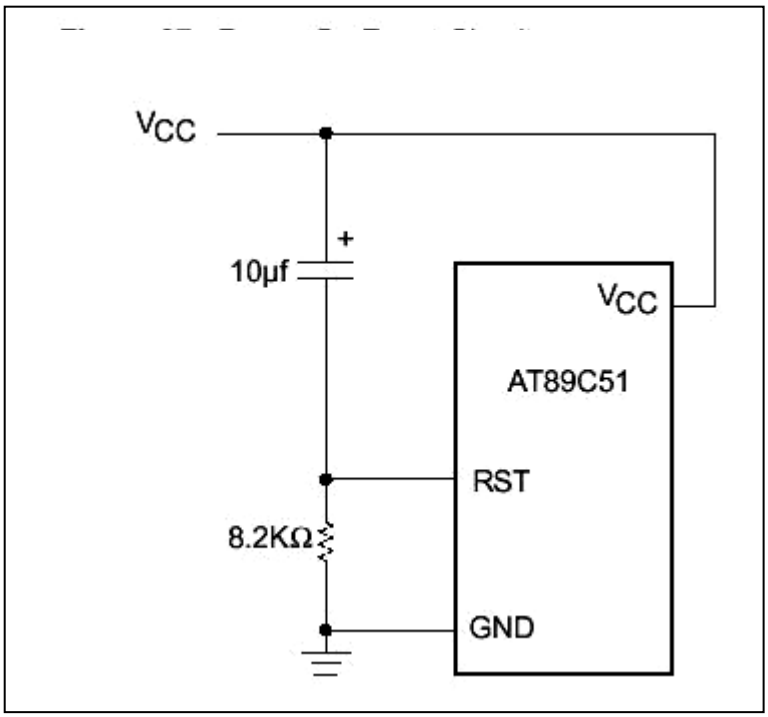
Tín hiệu khởi động lại bên ngoài đưa vào chân RST không đồng bộ với xung Clock bên trong. Chân RST được lấy mẫu tại thời điểm P2S5 của mỗi chu kỳ máy. Các chân của cổng sẽ giữ hoạt động hiện hành của chúng cho 19 chu kỳ dao động sau khi giá trị logic 1 đã được lấy mẫu ở chân RST.

Trong khi chân RST ở mức cao, ALE và /PSEN được đẩy dần lên mức cao. Sau đó RST được đẩy xuống, nó sẽ giữ 1 đến 2 chu kỳ máy đối với ALE và /PSEN để khởi động xung Clock. Vì lý do này mà các dịch vụ khác(đưa từ ngoài vào) không thể đồng bộ được với bộ thời gian bên trong của AT89C51.

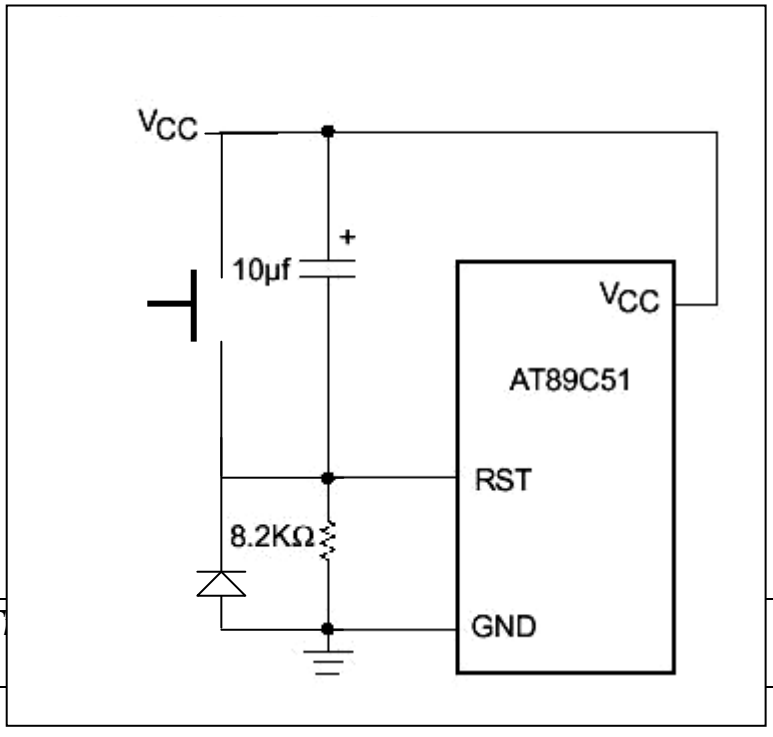
Khi Reset có hiệu lực thì giá trị của các SFR, được mô tả ở (Table -Reset Values of SFRs). Trong đó thanh ghi SBUF, và một số bit của PCON, T2MOD, IE, IP có giá trị bất định. Các chốt cổng từ P0…P3 có giá trị FFh, SP có giá trị 07h. Các thanh ghi còn lại có giá trị 00h.

Riêng đối với RAM bên trong On-chip AT89S8252 khi cấp nguồn hay Reset lại không bị tác động, mà nội dung trong RAM có giá trị ngẫu nhiên.

Khởi động lại cho On-chip có thể hoạt động ở trạng thái tự động hoặc bán tự động .



Hình 2.24. Khởi động tự động cho AT89C51.



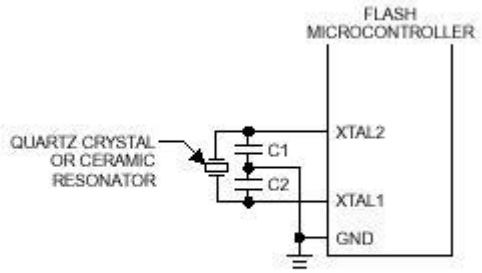
Hình 2.25. Khởi động bán tự động cho AT89C51.

Khởi động tự động có thể được tạo ra khi cấp nguồn điện +Vcc cho on-chip bằng mạch điện RC. Sau khi cấp nguồn, mạch RC giữ cho chân RST ở trạng thái cao trong thời gian tuỳ thuộc vào hằng số thời gian của mạch RC. Để đảm bảo khởi động được on-chip, thời gian chân RST ờ trạng thái cao phải đủ lớn (Khoảng lớn hơn 2 chu kỳ máy) để mạch dao động chuyển sang trạng thái dao động ổn định

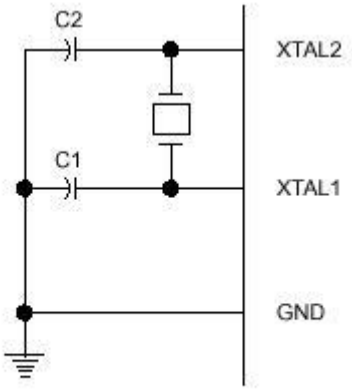
Nếu khởi động bán tự động, sau khi cấp nguồn mạch sẽ tự động Reset như khởi động tự động. Khi cần khởi động lại phải nhấn công tắc thường ngắt K, thời gian nhấn công tắc chân RST phải ở trạng thái cao.

2.5.8. Mạch dao động.

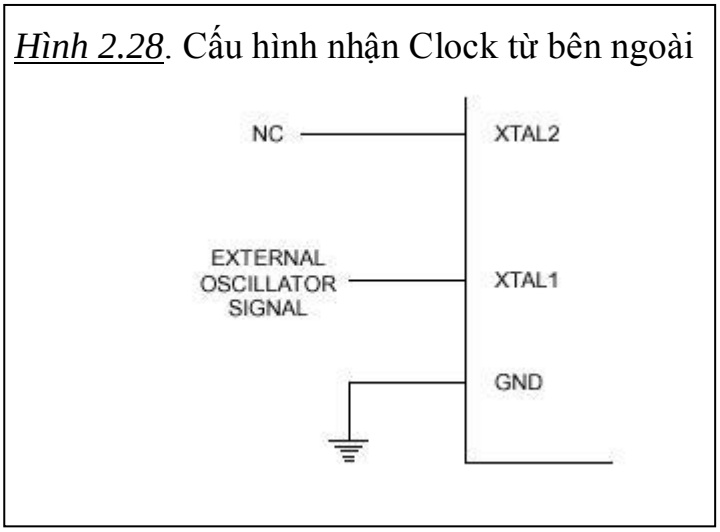
Hình 2.26. Sử dụng mạch dao động trên On chip



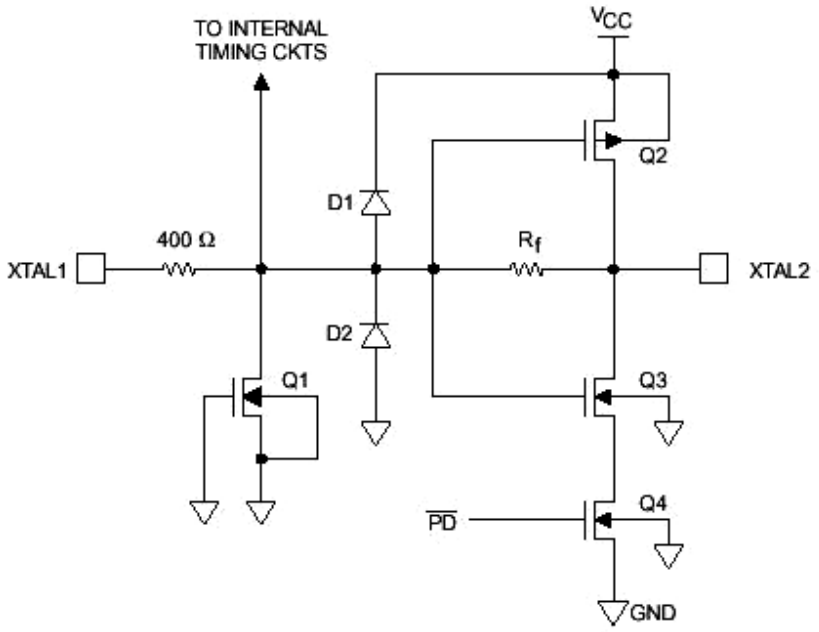
Hình 2.27. Kết nối mạch dao động



Hình 2.28. Cấu hình nhận Clock từ bên ngoài



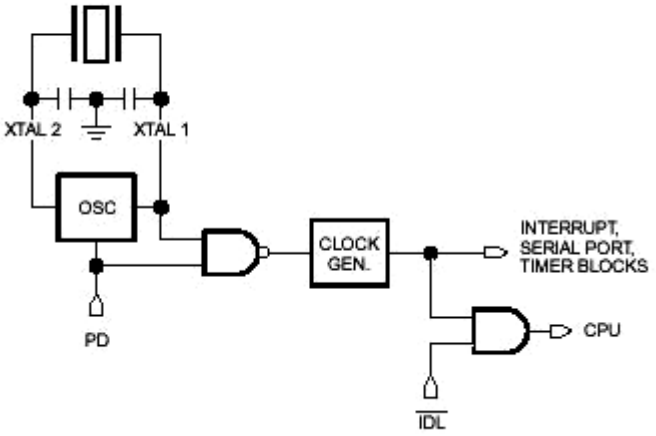
Hình 2.29. Mạch dao động bên trong On chip AT89C51



Note: In Atmel's CMOS microcontrollers the Oscillator Specification differs from that in NMOS versions.

2.5.9. Chế độ nguồn giảm và chế độ nghỉ

Hình 2.30. Phần cứng phục vụ chế độ nguồn giảm và chế độ nghỉ



2.5.11. Bảo vệ chương trình.

Họ VĐK	**Các bit khoá**
AT89C51	LB1, LB2, LB3
AT89C52	LB1, LB2, LB3
AT89C2051	LB1, LB2
AT89C1051	LB1, LB2

Khoá bộ nhớ chương trình cho họ VĐK AT89C51:

Chế độ	***LB1***	***LB2***	***LB3***	**Loại bảo vệ**
1	U	U	U	Không có đặc trưng khoá chương trình.
2	P	U	U	Các lệnh MOVC được thực thi từ bộ nhớ chương trình ngoài, không được phép tìm nạp lệnh từ bộ nhớ nội. EA được lấy mẫu và chốt khi reset. Việc lập trình trên Flash bị cấm.
3	P	P	U	Như chế độ 2, ngoài ra còn cấm việc kiểm tra chương trình.
4	P	P	P	Như chế độ 3, ngoài ra còn cấm việc thực thi chương trình ngoài.

Lưu ý: P=Programmed, U = Unprogrammed

Chương 3:
TẬP LỆNH CỦA HỌ VĐK AT89/80C51

Bộ VĐK có tập lệnh được tối ưu hoá để ứng dụng trong các hệ thống điều khiển, đo lường 8 bit. Để tăng khả năng truy xuất RAM nội trên các dữ liệu nhỏ, các kiểu định địa chỉ đặc biệt đã được áp dụng. Ngoài ra tập lệnh của VĐK còn hỗ trợ các biến 1 bit, cho phép quản lý bit trực tiếp trong các hệ logic và điều khiển bit có yêu cầu xử lý bit. Do họ VĐK AT89/80C51 có các mã lệnh 8 bit, nên số lệnh có thể lên đến 256 lệnh (thực tế có 255 lệnh, còn 1 lệnh chưa được định nghĩa). Trong đó có 139 lệnh 1 byte, 92 lệnh 2 byte và 24 lệnh 3 byte. Mỗi lệnh đều được đặc trưng bởi mã lệnh (mã máy), mã gợi nhớ, số byte của lệnh và số chu kỳ máy cần để thực thi lệnh. Các lệnh của AT89/80C51 được chia thành 5 nhóm lệnh:

- Nhóm lệnh di chuyển dữ liệu.
- Nhóm lệnh số học.

- Nhóm lệnh logic.
- Nhóm lệnh rẽ nhánh chương trình.
- Nhóm lệnh điều khiển biến logic.

Các quy ước trong câu lệnh và địa chỉ:

+ Rn: Thanh ghi R0-R7 của băng thanh ghi hiện hành đang được chọn để định địa chỉ thanh ghi.

+ Direct: Địa chỉ 8 bit của ô nhớ dữ liệu nội trú, nó có thể là ô nhớ trong RAM nội hoặc SFR. (00h-FFh)

+ @Ri: Ô nhớ 8 bit của RAM nội được định địa chỉ gián tiếp thông qua thanh ghi R0 hoặc R1.

+ Source (Src): toán hạng nguồn, có thể là Rn hoặc direct hoặc @Ri.

+ Dest: Toán hạng đích, có thể là Rn hoặc direct hoặc @Ri.

+ #Data: Hằng số 8 bit chứa trong lệnh.

+ #Data16: Hằng số 16 bit chứa trong lệnh.

+ Bit: Bit được định địa chỉ trực tiếp trong RAM nội trú hoặc SFR.

+ Rel: Offset 8 bit có dấu (từ -128 đến +127). Nó được lệnh SJMP và các lệnh nhảy có điều kiện sử dụng.

+ Addr11: địa chỉ 11 bit của bộ nhớ chương trình , được lệnh ACALL và AJMP sử dụng.

+ Addr16: địa chỉ 16 bit của 64Kb bộ nhớ chương trình, được lệnh LCALL và LJMP sử dụng.

Các ký hiệu dùng trong mô tả lệnh:

Ký hiệu	***ý nghĩa***
<-	Được thay thế bởi…
( )	Nội dung của…
(( ))	Dữ liệu được trỏ bởi…
rrr	1 trong 8 thanh ghi (R0-R7) của các băng thanh ghi
dddddddd	Các bit dữ liệu
aaaaaaaa	Các bit địa chỉ
bbbbbbbb	địa chỉ của 1 bit
i	Định địa chỉ gián tiếp thông qua R0 hoặc R1

eeeeeeee	Địa chỉ tương đối 8 bit

3.1. Nhóm lệnh di chuyển dữ liệu

3.1.1. Lệnh MOV dạng Byte:

Cú pháp câu lệnh: **MOV <dest-byte>, <src-byte>**

Chức năng: Sao chép nội dung của toán hạng nguồn vào toán hạng đích, nội dung của toán hạng nguồn không thay đổi. Lệnh này không làm ảnh hưởng tới các cờ và các thanh ghi khác.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
MOV A, Rn	1	1	11101rrr	(A)<-(Rn)
MOV A, direct	2	1	11100101 aaaaaaaa	(A)<-(direct)
MOV A, @Ri	1	1	1110111i	(A)<-((Ri))
MOV A, #data	2	1	01110100 dddddddd	(A)<-#data
MOV Rn, A	1	1	11111rrr	(Rn)<-(A))
MOV Rn, direct	2	2	10101rrr aaaaaaaa	(Rn)<-(direct)
MOV Rn, #data	2	1	01111rrr dddddddd	(Rn)<-#data
MOV direct, A	2	1	11110101 aaaaaaaa	(direct)<-(A)
Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
MOV direct, Rn	2	2	10001rrr aaaaaaaa	(direct)<-(Rn)
MOV direct, direct	3	2	10000101 aaaaaaaa aaaaaaaa	(direct)<-(direct)
MOV direct, @Ri	2	2	1000011i aaaaaaaa	(direct)<-((Ri))
MOV direct, #data	3	2	01110101 aaaaaaaa dddddddd	(direct)<-#data
MOV @Ri, A	1	1	1111011i	((Ri))<-(A)
MOV @Ri, direct	2	2	1010011i	((Ri))<-(direct)
MOV @Ri, #data	2	1	0111011i dddddddd	((Ri))<-#data

3.1.2. Lệnh MOV dạng Bit:

Cú pháp câu lệnh: **MOV <dest-bit>, <scr-bit>**

Chức năng: Chuyển bit dữ liệu ở dạng sao chép toán hạng nguồn vào toán hạng đích. Một trong 2 toán hạng phải là cờ nhớ (C), toán hạng còn lại sẽ là bit bất kỳ được định địa chỉ trực tiếp. Lệnh không làm ảnh hưởng tới các thanh ghi khác hoặc các cờ khác.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
MOV C, bit	2	1	10100010 bbbbbbbb	(C)<-(bit)
MOV bit, C	2	2	10010010 bbbbbbbb	(bit)<-(C)

3.1.3. Lệnh MOV dạng Word:

Cú pháp câu lệnh: **MOV DPTR, #data16**

Chức năng: Giá trị 16 bit ở toán hạng thứ 2 trực tiếp trong câu lệnh được nạp vào thanh ghi DPTR. Hằng số 16 bit này được đặt ở byte 2 và byte 3 của lệnh. Byte 2 là byte cao được nạp cho thanh ghi DPH, byte 3 là byte thấp được nạp vào thanh ghi DPL. Lệnh này không ảnh hưởng tới các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
MOV DPTR,#data16	3	2	10010000 dddddddd dddddddd	(C)<-(bit)

3.1.4. Lệnh chuyển byte mã lệnh:

Cú pháp câu lệnh: **MOVC A, @A + <thanh ghi cơ sở>**

Chức năng: Nạp cho thanh ghi tích luỹ byte mã lệnh từ bộ nhớ chương trình. Địa chỉ của byte được tìm nạp trong bộ nhớ là tổng nội dung của thanh ghi A 8 bit với nội dung của thanh ghi cơ sở 16 bit (có thể là DPTR hoặc PC - thanh ghi đếm chương trình). Trong trường hợp sau, PC được tăng để trỏ đến địa chỉ của lệnh tiếp theo ((PC)<-(PC+1)) trước khi được công với nội dung của thanh ghi A, còn thanh ghi DPTR không bị thay đổi. Lệnh không ảnh hưởng tới các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***

MOVC A,@A+DPTR	1	2	10010011	(A)<-((A)+(DPTR))
MOVC A,@A+PC	1	2	10000011	(A)<-((A)+(PC))

3.1.5. Lệnh chuyển dữ liệu ra ngoài:

Cú pháp câu lệnh: **MOVX <dest-byte>, <src-byte>**

Chức năng: Chuyển dữ liệu giữa thanh ghi tích luỹ với bộ nhớ ngoài. Các lệnh này được chia làm 2 loại, một loại cung cấp địa chỉ 8 bit và 1 loại cung cấp địa chỉ 16 bit.

Nếu dữ liệu được chuyển là 8 bit, nội dung của R0 hoặc R1 trong băng thanh ghi hiện hành sẽ cung cấp địa chỉ 8 bit đa hợp với dữ liệu trên P0. 8 bit địa chỉ này đủ để mã hoá cho các cổng I/O mở rộng bên ngoài chip hoặc cho 1 dãy RAM kích thước tương đối nhỏ. Với các dãy RAM có kích thước lớn hơn một chút, một vài chân của cổng bất kỳ nào đó có thể được sử dụng để tạo ra các bit địa chỉ cao. Các chân này nên được điều khiển bởi 1 lệnh xuất đặt trước lệnh MOVX.

Nếu dữ liệu được chuyển là 16 bit, thì DPTR tạo ra địa chỉ 16 bit. P2 xuất ra 8 bit địa chỉ cao (nội dung của DPH), còn P0 xuất ra 8 bit địa chỉ thấp đa hợp với dữ liệu.Thanh ghi chức năng đặc biệt P2 duy trì nội dung trước đó trong khi các bộ đệm xuất của P2 đang phát các nội dung của DPH. Dạng này nhanh hơn và hiệu quả hơn khi truy xuất nhiều dãy dữ liệu rất lớn (lên đến 64 Kb) do ta không cần thêm lệnh để thiết lập các cổng khác.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
MOVX A, @Ri	1	2	11100011	(A)<-((Ri))
MOVX @Ri, A	1	2	11110011	((Ri))<(A)
MOVX A, @DPTR	1	2	11100000	(A)<-((DPTR))
MOVX @DPTR, A	1	2	11110000	((DPTR))<-(A)

3.1.6. Lệnh chuyển số liệu vào ngăn xếp:

Cú pháp câu lệnh: **PUSH direct**

Chức năng: Chuyển số liệu có trong câu lệnh vào ngăn xếp. Trước tiên, con trỏ ngăn xếp (SP) được tăng lên 1, sau đó số liệu sẽ được chuyển vào đỉnh của ngăn xếp mà địa chỉ đỉnh này được trỏ bởi SP. Ngăn xếp nằm ở RAM nội trú.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
PUSH direct	2	2	11000000 aaaaaaaa	(SP)<-(SP+1)

				((SP))<-(direct)

3.1.7. Lệnh chuyển số liệu ra khỏi ngăn xếp:

***Cú pháp câu lệnh**:* **POP direct**

Chức năng: Chuyển nội dung của ngăn xếp ở RAM trong, có địa chỉ được SP trỏ tới đến nơi có địa chỉ trực tiếp trong câu lệnh. Sau đó, con trỏ ngăn xếp (SP) được giảm đi 1. Lệnh không ảnh hưởng tới các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
POP direct	2	2	11010000 aaaaaaaa	(direct)<-((SP)) (SP)<-(SP-1)

3.1.8.Hoán chuyển dữ liệu:

Cú pháp câu lệnh: **XCH A, <byte>**

Chức năng: Hoán chuyển nội dung giữa thanh ghi A với thanh ghi hoặc bộ nhớ có địa chỉ chứa trong toán hạng thứ 2 của câu lệnh. Toán hạng thứ 2 có thể được định địa chỉ kiểu thanh ghi, thanh ghi trực tiếp hoặc thanh ghi gián tiếp.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
XCH A, Rn	1	1	11001rrr	(A)<-->(Rn)
XCH A, direct	2	1	11000101 aaaaaaaa	(A) <-->(direct)
XCH A, @Ri	1	1	1100011i	(A) <-->((Ri))

3.1.9.Hoán chuyển 4 bit thấp:

Cú pháp câu lệnh: **XCHD A,@Ri**

Chức năng: Hoán chuyển 4 bit thấp nội dung trong thanh ghi A với ô nhớ của RAM bên trong, có địa chỉ được định gián tiếp qua thanh ghi được chỉ ra trong lệnh. Lệnh này không ảnh hưởng tới trạng thái các cờ và nửa cao của các thanh ghi trong lệnh.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
XCHD A, @Ri	1	1	1101011i	(A3-A0)<-->((Ri3-Ri0))

3.2. Nhóm lệnh tính toán số học.

3.2.1. Lệnh thực hiện phép cộng.

Cú pháp của câu lệnh: **ADD A, <scr-byte>**

Chức năng: Cộng giá trị 1 byte ở địa chỉ được chỉ ra ở câu lệnh với nội dung trong thanh ghi tích luỹ, kết quả được lưu vào thanh ghi tích luỹ. Nếu có nhớ từ bit số 7 hoặc bit số 3 thì cờ nhớ hoặc cờ nhớ phụ được thiết lập, ngược lại các cờ nêu trên được xoá. Khi cộng 2 số nguyên không dấu mà bị tràn thì cờ nhớ cũng được thiết lập để cho ta biết phép toán bị tràn. Trường hợp thực hiện lệnh ADD mà có nhớ từ bit số 6 nhưng không có nhớ từ bit số 7, hoặc có nhớ từ bit số 7 nhưng không có nhớ từ bit số 6 thì cờ tràn sẽ được thiết lập, ngược lại thì OV bị xoá. Khi cộng 2 số nguyên có dấu mà tổng là 1 số âm thì OV được thiết lập.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
ADD A, Rn	1	1	00101rrr	(A)<- (A) + (Rn)
ADD A, direct	2	1	00100101 aaaaaaaa	(A)<- (A) + (direct)
ADD A, @Ri	1	1	0010011i	(A)<- (A) + ((Ri))
ADD A, #data	2	1	00100100 dddddddd	(A)<- (A) + #data

3.2.2. Lệnh cộng có nhớ.

Cú pháp của câu lệnh: **ADDC A, <scr-byte>**

Chức năng: Cộng đồng thời nội dung của 1 byte ở địa chỉ được chỉ ra trong câu lệnh với nội dung chứa trong thanh ghi tích luỹ và cờ nhớ. Nếu có nhớ từ bit số 7 hoặc số 3 thì cờ nhớ hoặc cờ nhớ phụ được thiết lập bằng 1, ngược lại các cờ nêu trên bị xoá. Khi cộng các số nguyên không dấu mà bị tràn thì cờ nhớ cũng được thiết lập. Trường hợp thực hiện lệnh ADDC mà có nhớ từ bit số 6 nhưng không nhớ từ bit số 7, hoặc có nhớ từ bit só 7 nhưng không nhớ từ bit số 6 thì cờ tràn sẽ được thiết lập, ngược lại cờ này bị xoá. Khi cộng các số nguyên có dấu mà tổng là 1 số âm thì OV được thiết lập.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
ADDC A, Rn	1	1	00110rrr	(A)<- (A) + (C) + (Rn)

ADDC A, direct	2	1	00110101 aaaaaaaa	(A)<- (A) + (C) + (direct)
ADDC A, @Ri	1	1	0011011i	(A)<- (A) + (C) + ((Ri))
ADDC A, #data	2	1	00110100 dddddddd	(A)<- (A) + (C) + #data

3.2.3. Lệnh trừ có mượn.

Cú pháp của câu lệnh: **SUBB A, <scr-byte>**

Chức năng: Trừ thanh ghi tích luỹ cho toán hạng thứ 2 và cờ nhớ, kết quả được lưu vào thanh ghi tích luỹ. Cờ nhớ được đặt bằng 1 nếu có số mượn được cần đến cho bit số 7, ngược lại thì cờ nhớ bị xoá. Cờ nhớ phụ được thiết lập nếu có nhớ cho bit 3. Trường hợp thực hiện lệnh SUBB mà có số mượn được cần đến cho bit 7(không phải cho bit 6), hoặc cho bit 6 (không phải cho bit 7) thì cờ tràn sẽ được thiết lập, ngược lại thì OV bị xoá. Khi trừ các số nguyên có dấu mà kết quả là 1 số âm thì OV được thiết lập.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
SUBB A, Rn	1	1	10011rrr	(A)<- (A) - (C) - (Rn)
SUBB A, direct	2	1	10010101 aaaaaaaa	(A)<- (A) - (C) - (direct)
SUBB A, @Ri	1	1	1001011i	(A)<- (A) - (C) - ((Ri))
SUBB A, #data	2	1	10010100 dddddddd	(A)<- (A) - (C) - #data

3.2.4. Lệnh tăng lên 1 đơn vị.

Cú pháp của câu lệnh: **INC <byte>**

Chức năng: Tăng giá trị của byte trong câu lệnh lên 1 đơn vị. Nếu giá trị ban đầu của byte là 0FFh, thì sau khi thực hiện lệnh INC nội dung của byte sẽ là 00h. Lệnh này không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
INC A	1	1	00000100	(A)<- (A) + 1
INC Rn	1	1	00001rrr	(Rn)<- (Rn) + 1

INC direct	2	1	00000101 aaaaaaaa	(direct)<- (direct) + 1
INC @Ri	1	1	0000011i	((Ri))<- ((Ri)) + 1

3.2.5. Lệnh giảm 1 đơn vị.

Cú pháp của câu lệnh: **DEC <byte>**

Chức năng: Giảm giá trị của byte trong câu lệnh xuống 1 đơn vị. Nếu giá trị ban đầu của byte là 00h, thì sau khi thực hiện lệnh DEC nội dung của byte sẽ là 0FFh. Lệnh này không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
DEC A	1	1	00010100	(A)<- (A) – 1
DEC Rn	1	1	00011rrr	(Rn)<- (Rn) – 1
DEC direct	2	1	00010101 aaaaaaaa	(direct)<- (direct) – 1
DEC @Ri	1	1	0001011i	((Ri))<- ((Ri)) – 1

3.2.6. Lệnh tăng con trỏ dữ liệu.

Cú pháp của câu lệnh: **INC DPTR**

Chức năng: Tăng con trỏ dữ liệu lên 1 đơn vị. Khi byte thấp của con trỏ dữ liệu bị tràn, thì byte cao của con trỏ dữ liệu tăng lên 1 đơn vị. Lệnh này không ảnh hưởng tới trạng thái các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
INC DPTR	1	2	10100011	(DPTR)<- (DPTR) + 1

3.2.7. Lệnh thực hiện phép nhân.

Cú pháp của câu lệnh: **MUL AB**

Chức năng: Nhân các số nguyên không dấu 8 bit trong thanh ghi tích luỹ với thanh ghi B. Byte thấp của kết quả 16 bit được lưu trong thanh ghi tích luỹ, còn byte cao được lưu trong thanh ghi B. Nếu kết quả lớn hơn 0FFh thì cờ tràn được thiết lập, cờ nhớ luôn bị xoá.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
MUL AB	1	4	10100100	(B)<- byte cao của (A)x(B) (A)<- byte thấp của (A)x(B)

3.2.8. Lệnh thực hiện phép chia.

Cú pháp của câu lệnh: **DIV AB**

Chức năng: Chia số nguyên không dấu 8 bit trong thanh ghi tích luỹ cho số nguyên không dấu 8 bit trong thanh ghi B. Thương số được lưu trong thanh ghi tích luỹ, còn số dư được lưu trong thanh ghi B. Cờ tràn và cờ nhớ bị xoá.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
DIV AB	1	4	10000100	(A)<- thương của (A)/(B) (B)<- số dư của (A)/(B)

3.2.9. Hiệu chỉnh số thập phân.

Cú pháp của câu lệnh: **DA A**

Chức năng: Hiệu chỉnh thập phân nội dung 8 bit trong thanh ghi A sau khi thực hiện phép cộng.

Nếu 4 bit thấp trong thanh ghi A có giá trị lớn hơn 9 hoặc cờ nhớ phụ được thiết lập thì phải cộng thêm 6 vào thanh ghi A để cho chữ số thập phân được chính xác. Phép cộng này sẽ đặt cờ nhớ nếu số nhớ từ 4 bit thấp chuyển đến tất cả 4 bit cao, ngược lại phép toán không xoá cờ nhớ.

Nếu 4 bit cao trong thanh ghi A có giá trị lớn hơn 9 hoặc cờ nhớ (CF) được thiết lập, thì cũng phải cộng thêm 6 vào thanh ghi A.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*
DA A	1	1	11010100

Hoạt động:

- Nếu [[(A3-A0)>9]hoặc[(AC)=1]] thì (A3-A0)<-(A3-A0) + 6
- Nếu [[(A7-A4)>9]hoặc[(C)=1]] thì (A7-A4)<-(A7-A4) + 6

3.3. Nhóm lệnh tính toán logic.

3.3.1. Lệnh AND cho các biến 1 byte.

Cú pháp câu lệnh: **ANL <dest-byte>, <src-byte>**

Chức năng: Thực hiện phép toán logic AND theo mức bit giữa các biến dài 1 byte đã cho, kết quả được lưu vào toán hạng đích. Toán hạng nguồn cho phép 6 chế độ địa chỉ hoá. Khi toán hạng đích là thanh ghi tích luỹ thì toán hạng nguồn có thể là thanh ghi, trực tiếp, thanh ghi-gián tiếp hoặc tức thời. Khi toán hạng đích là địa chỉ trực tiếp thì toán hạng nguồn có thể là thanh ghi tích luỹ hoặc dữ liệu tức thời. Lệnh này không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
ANL A, Rn	1	1	01011rrr	(A)<-(A) AND (Rn)
ANL A, direct	2	1	01010101 aaaaaaaa	(A)<-(A) AND (dir.)
ANL A, @Ri	1	1	0101011i	(A)<- (A) AND ((Ri))
ANL A, #data	2	1	01010100 dddddddd	(A)<- (A) AND #data
ANL direct, A	2	1	01010010 aaaaaaaa	(dir.)<-(dir.)AND (A)
ANL direct, #data	3	2	01010011 aaaaaaaa dddddddd	(dir.)<(dir.)AND#data

3.3.2. Lệnh AND cho các biến 1 bit

Cú pháp câu lệnh: **ANL C, <src-bit>**

Chức năng: Thực hiện phép tính logic AND cho các biến mức bit. Nếu giá trị logic của toán hạng nguồn bằng 0, thì cờ nhớ bị xoá. Dấu "/" đứng trước 1 toán hạng cho biết bit nguồn được lấy bù trước khi thực hiện AND với cờ nhớ nhưng ***giá trị của bit nguồn không bị thay đổi bởi thao tác lấy bù***. Lệnh này không làm ảnh hưởng tới trạng thái các cờ khác. Toán hạng nguồn chỉ được sử dụng kiểu định địa chỉ trực tiếp.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***

ANL C, bit	2	2	10000010 bbbbbbbbb	(C)<-(C) AND (bit)
ANL C, /bit	2	2	10110000 bbbbbbbbb	(C)<-(C) AND NOT (bit)

3.3.3. Lệnh OR cho các biến 1 byte

***Cú pháp câu lệnh**:* **ORL <dest-byte>, <src-byte>**

Chức năng: Thực hiện phép toán logic OR theo mức bit giữa các biến dài 1 byte đã cho, kết quả được lưu vào toán hạng đích. Toán hạng nguồn cho phép 6 chế độ địa chỉ hoá. Khi toán hạng đích là thanh ghi tích luỹ thì toán hạng nguồn có thể là thanh ghi, trực tiếp, thanh ghi-gián tiếp hoặc tức thời. Khi toán hạng đích là địa chỉ trực tiếp thì toán hạng nguồn có thể là thanh ghi tích luỹ hoặc dữ liệu tức thời. Lệnh này không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
ORL A, Rn	1	1	01001rrr	(A)<-(A) OR (Rn)
ORL A, direct	2	1	01000101 aaaaaaaa	(A)<-(A) OR (dir.)
ORL A, @Ri	1	1	0100011i	(A)<- (A) OR ((Ri))
ORL A, #data	2	1	01000100 dddddddd	(A)<- (A) OR #data
ORL direct, A	2	1	01000010 aaaaaaaa	(dir.)<-(dir.) OR (A)
ORL direct, #data	3	2	01000011 aaaaaaaa dddddddd	(dir.)<(dir.) OR #data

3.3.4. Lệnh OR cho các biến 1 bit

***Cú pháp câu lệnh**:* **ORL C, <src-bit>**

Chức năng: Thực hiện phép tính logic OR cho các biến mức bit. Nếu giá trị logic của toán hạng nguồn bằng 1, thì cờ nhớ được thiết lập. Dấu "/" đứng trước 1 toán hạng cho biết bit nguồn được lấy bù trước khi thực hiện OR với cờ nhớ nhưng ***giá trị của bit nguồn không bị thay đổi bởi thao tác lấy bù***. Lệnh này không làm ảnh hưởng tới trạng thái các cờ khác.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
ORL C, bit	2	2	01110010 bbbbbbbbb	(C)<-(C) OR (bit)
ORL C, /bit	2	2	10100000 bbbbbbbbb	(C)<-(C) OR NOT (bit)

3.3.5. Lệnh X-OR cho các biến 1 byte

Cú pháp câu lệnh: **XRL <dest-byte>, <src-byte>**

Chức năng: Thực hiện phép toán logic X-OR theo mức bit giữa các biến dài 1 byte đã cho, kết quả được lưu vào toán hạng đích. Toán hạng nguồn cho phép 6 chế độ địa chỉ hoá. Khi toán hạng đích là thanh ghi tích luỹ thì toán hạng nguồn có thể là thanh ghi, trực tiếp, thanh ghi-gián tiếp hoặc tức thời. Khi toán hạng đích là địa chỉ trực tiếp thì toán hạng nguồn có thể là thanh ghi tích luỹ hoặc dữ liệu tức thời. Lệnh này không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
XRL A, Rn	1	1	01101rrr	(A)<-(A) XOR (Rn)
XRL A, direct	2	1	01100101 aaaaaaaa	(A)<-(A) XOR (dir.)
XRL A, @Ri	1	1	0110011i	(A)<- (A) XOR ((Ri))
XRL A, #data	2	1	01100100 dddddddd	(A)<- (A) XOR #data
XRL direct, A	2	1	01100010 aaaaaaaa	(dir.)<-(dir.)XOR (A)
XRL direct, #data	3	2	01100011 aaaaaaaa dddddddd	(dir.)<(dir.) XOR #data

3.3.6. Lệnh dịch trái thanh ghi A

Cú pháp câu lệnh: **RL A**

Chức năng: 8 bit trong thanh ghi A được dịch trái 1 bit. Bit 7 được quay đến vị trí của bit 0. Các cờ không bị ảnh hưởng.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
RL A	1	1	00100011	(An+1) <- (An), với n = 0…6 (A0) <- (A7)

3.3.7. Lệnh dịch trái thanh ghi A cùng với cờ nhớ

Cú pháp câu lệnh: **RLC A**

Chức năng: 8 bit trong thanh ghi A và cờ nhớ cùng được dịch trái 1 bit. Bit 7 được chuyển đến cờ nhớ và trạng thái ban đầu của cờ nhớ được đưa về bit 0. Các cờ khác không bị ảnh hưởng.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
RLC A	1	1	00110011	(An+1) <- (An), với n = 0…6 (A0) <- (C) (C) <- (A7)

3.3.8. Lệnh dịch phải thanh ghi A.

***Cú pháp câu lệnh**:* **RR A**

Chức năng: 8 bit trong thanh ghi A được dịch sang phải 1 bit. Bit 0 được quay đến vị trí của bit 7. Các cờ không bị ảnh hưởng.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
RR A	1	1	00000011	(An) <- (An+1), với n = 0…6 (A7) <- (A0)

3.3.9. Lệnh dịch phải thanh ghi A cùng với cờ nhớ

***Cú pháp câu lệnh**:* **RRC A**

Chức năng: 8 bit trong thanh ghi A và cờ nhớ cùng được dịch phải 1 bit. Bit 0 được chuyển đến cờ nhớ và trạng thái ban đầu của cờ nhớ được đưa về bit 7. Các cờ khác không bị ảnh hưởng.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
RRC A	1	1	00010011	(An) <- (An+1), với n = 0…6 (A7) <- (C) (C) <- (A0)

3.3.10. Lệnh tráo đổi nội dung hai nửa byte của A

***Cú pháp câu lệnh**:* **SWAP A**

Chức năng: Tráo đổi nội dung 2 nửa thấp và cao (mỗi nửa 4 bit) của thanh ghi A (các bit từ 0 đến 3 và các bit từ 4 đến 7). Thao tác này còn được hiểu là quay thanh ghi A 4 bit. Các cờ không bị ảnh hưởng.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
SWAP A	1	1	11000100	(A3-A0) <- (A7-A4)

3.4. Nhóm lệnh rẽ nhánh chương trình.

3.4.1. Lệnh gọi tuyệt đối.

Cú pháp câu lệnh: **ACALL addr11**

Chức năng: Gọi không điều kiện một chương trình con đặt tại địa chỉ được chỉ ra trong câu lệnh. Lệnh này tăng bộ đếm chương trình thêm 2 đơn vị để PC chứa địa chỉ của lệnh kế lệnh ACALL, sau đó cất nội dung 16 bit của PC vào ngăn xếp (byte thấp cất trước) và tăng con trỏ ngăn xếp lên 2 đơn vị. Địa chỉ đích sẽ được hình thành bằng cách ghép 5 bit cao của thanh ghi PC (sau khi được tăng), 3 bit cao của byte mã lệnh và byte thứ 2 của lệnh. Do đó chương trình con được gọi phải nằm trong đoạn 2 Kbyte của bộ nhớ chương trình chí it phải chứa lệnh đầu tiên của chương trình con này. Lệnh không làm ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
ACALL addr11	2	2	aaa10001 aaaaaaaa	(PC) <- (PC) + 2 (SP) <- (SP) + 1 ((SP)) <- (PC7-PC0) (SP) <- (SP) + 1 ((SP)) <- (PC15-PC8) (PC10-PC0) <- (page address)

3.4.2. Lệnh gọi dài.

Cú pháp câu lệnh: **LCALL addr16**

Chức năng: Gọi một chương trình con đặt tại địa chỉ được chỉ ra trong câu lệnh. Lệnh này tăng bộ đếm chương trình thêm 3 đơn vị để PC chứa địa chỉ của lệnh kế lệnh LCALL, sau đó cất nội dung 16 bit của PC vào ngăn xếp (byte thấp cất trước) và tăng con trỏ ngăn xếp lên 2 đơn vị. Tiếp theo nó sẽ chuyển byte thứ 2 và byte thứ 3 trong câu lệnh LCALL vào byte cao và byte thấp của PC. Việc thực thi chương trình tiếp tục với lệnh ở địa chỉ này. Như vậy chương trình con có thể

bắt đầu bằng bất cứ nơi nào trong không gian bộ nhớ chương trình 64 Kbyte. Lệnh không làm ảnh hưởng tới các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
LCALL addr16	3	2	00010010 aaaaaaaa aaaaaaaa	(PC) <- (PC) + 3 (SP) <- (SP) + 1 ((SP)) <- (PC7-PC0) (SP) <- (SP) + 1 ((SP)) <- (PC15-PC8) (PC) <- addr15-addr0

3.4.3. Lệnh quay trở lại từ chương trình con.

Cú pháp câu lệnh: **RET**

Chức năng: Trở về từ chương trình con. Lệnh này được thực hiện sau khi thực hiện xong lệnh ACALL hoặc LCALL. RET lấy lại byte cao và byte thấp của PC từ ngăn xếp, giảm SP đI 2 đơn vị. Chương trình tiếp tục được thực hiện với lệnh có địa chỉ ở trong PC. Các cờ không bị ảnh hưởng.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
RET	1	2	00100010	(PC15-PC8) <- ((SP)) (SP) <- (SP) - 1 (PC7-PC0) <- ((SP)) (SP) <- (SP) – 1

3.4.4. Lệnh quay trở lại từ ngắt.

Cú pháp câu lệnh: **RETI**

Chức năng: Trở về từ chương trình con. RETI lấy lại byte cao và byte thấp của PC từ ngăn xếp, phục hồi logic ngắt để có thể nhận các ngắt khác có cùng mức ưu tiên ngắt với ngắt được xử lý, sau đó giảm SP đi 2 đơn vị. Chương trình tiếp tục được thực hiện với lệnh trước khi xử lý ngắt với địa chỉ ở trong PC. Các cờ không bị ảnh hưởng.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
RETI	1	2	00110010	(PC15-PC8) <- ((SP)) (SP) <- (SP) - 1 (PC7-PC0) <- ((SP)) (SP) <- (SP) – 1

3.4.5. Lệnh nhảy gián tiếp.

Cú pháp câu lệnh: **JMP @A+DPTR**

Chức năng: Cộng giá trị không dấu 8 bit của thanh ghi A với con trỏ dữ liệu 16 bit và nạp kết quả vào bộ đếm chương trình, kết quả này chính là địa chỉ để nạp lệnh kế tiếp. Việc cộng 16 bit được thực hiện: Số nhớ từ 8 bit thấp được truyền đến tất cả các bit cao. Cả 2, thanh ghi A và DPTR đều không bị thay đổi. Lệnh này không ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
JMP @A+DPTR	1	2	01110011	(PC)<-(A)+(DPTR)

3.4.6. Lệnh nhảy nếu 1 bit được thiết lập.

Cú pháp câu lệnh: **JB bit, rel**

Chức năng: Nếu bit đã cho có giá trị bằng 1 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh, ngược lại nó sẽ tiếp tục thực hiện lệnh tiếp theo. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 3 của lệnh với nội dung trong PC (sau khi được tăng đến địa chỉ của byte đầu tiên của lệnh kế tiếp). Bit được kiểm tra không bị thay đổi, lệnh không ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
JB bit, rel	3	2	00100000 bbbbbbbbb eeeeeeee	(PC)<-(PC)+3 Nếu (bit)=1 thì: (PC)<-(PC) + rel

3.4.7. Lệnh nhảy nếu 1 bit không được thiết lập.

Cú pháp câu lệnh: **JNB bit, rel**

Chức năng: Nếu bit đã cho có giá trị bằng 0 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh, ngược lại nó sẽ tiếp tục thực hiện lệnh tiếp theo. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 3 của lệnh với nội dung trong PC (sau khi được tăng đến địa chỉ của byte đầu tiên của lệnh kế tiếp). Bit được kiểm tra không bị thay đổi, lệnh không ảnh hưởng tới các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
JNB bit, rel	3	2	00110000 bbbbbbbbb eeeeeeee	(PC)<-(PC)+3 Nếu (bit)=0 thì: (PC)<-(PC) + rel

3.4.8. Lệnh nhảy nếu 1 bit được thiết lập và xoá bit đó.

Cú pháp câu lệnh: **JBC bit, rel**

Chức năng: Nếu bit đã cho có giá trị bằng 1 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh và xoá bit này, ngược lại nó sẽ tiếp tục thực hiện lệnh tiếp theo. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 3 của lệnh với nội dung trong PC (sau khi được tăng đến địa chỉ của byte đầu tiên của lệnh kế tiếp). Lệnh không ảnh hưởng tới các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
JBC bit, rel	3	2	00010000 bbbbbbbbb eeeeeeee	(PC)<-(PC)+3 Nếu (bit)=1 thì: (bit)<- 0 (PC)<- (PC) + rel

3.4.9. Lệnh nhảy nếu cờ nhớ được thiết lập.

Cú pháp câu lệnh: **JC rel**

Chức năng: Nếu cờ CF có giá trị bằng 1 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh, ngược lại nó sẽ tiếp tục thực hiện lệnh tiếp theo. Địa chỉ đích được

tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 2 của lệnh với nội dung trong PC (sau khi được tăng bởi 2). Lệnh không ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
JC rel	2	2	01000000 eeeeeeee	(PC)<-(PC)+2 Nếu (C)=1 thì: (PC)<- (PC) + rel

3.4.10. Lệnh nhảy nếu cờ nhớ không được thiết lập.

Cú pháp câu lệnh: **JNC rel**

Chức năng: Nếu cờ CF có giá trị bằng 0 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh, ngược lại nó sẽ tiếp tục thực hiện lệnh tiếp theo. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 2 của lệnh với nội dung trong PC (sau khi được tăng bởi 2). Lệnh không ảnh hưởng tới các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
JNC rel	2	2	01010000 eeeeeeee	(PC)<-(PC)+2 Nếu (C)=0 thì: (PC)<- (PC) + rel

3.4.11. Lệnh nhảy nếu thanh ghi A bằng 0.

Cú pháp câu lệnh: JZ rel

Chức năng: Nếu tất cả các bit của thanh ghi A có giá trị bằng 0 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh, ngược lại nó sẽ tiếp tục thực hiện lệnh tiếp theo. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 2 của lệnh với nội dung trong PC (sau khi được tăng bởi 2). Lệnh không ảnh hưởng tới các cờ. Nội dung thanh ghi A không bị thay đổi.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
JZ rel	2	2	01100000 eeeeeeee	(PC)<-(PC)+2

				Nếu (A)=0 thì: (PC)<- (PC) + rel

3.4.12. Lệnh nhảy nếu thanh ghi A khác 0.

Cú pháp câu lệnh: **JNZ rel**

Chức năng: Nếu có 1 hoặc nhiều bit của thanh ghi A có giá trị bằng 1 thì nó nhảy tới địa chỉ đã xác định trong câu lệnh, ngược lại nó sẽ tiếp tục thực hiện lệnh tiếp theo. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu (tương đối) trong byte thứ 2 của lệnh với nội dung trong PC (sau khi được tăng bởi 2). Lệnh không ảnh hưởng tới các cờ. Nội dung thanh ghi A không bị thay đổi.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
JNZ rel	2	2	01110000 eeeeeeee	(PC)<-(PC)+2 Nếu (A) < > 0 thì: (PC)<- (PC) + rel

3.4.13. Lệnh nhảy khi so sánh 2 toán hạng.

Cú pháp câu lệnh: **CJNE <dest-byte>, <src-byte>, rel**

Chức năng: So sánh giá trị của 2 toán hạng đầu tiên, nếu 2 toán hạng không bằng nhau thì chương trình được rẽ nhánh. Địa chỉ đích rẽ nhánh được tính bằng cách cộng độ lệch tương đối (có dấu) trong byte sau cùng của lệnh với nội dung của PC (sau khi nội dung của PC được tăng đến địa chỉ bắt đầu của lệnh kế tiếp CJNZ). Cờ nhớ (CF) sẽ được thiết lập nếu như giá trị nguyên không dấu của toán hạng đích nhỏ hơn giá trị nguyên không dấu của toán hạng nguồn, ngược lại thì cờ này bị xoá. Lệnh này không làm thay đổi giá trị của các toán hạng

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
CJNE A, direct, rel	3	2	10110101 aaaaaaaa eeeeeeee	(PC)<-(PC)+3 Nếu (A) < > (dir.) thì: (PC)<- (PC) + offset Nếu (A) < (dir.) thì: (C) <- 1, ngược lại: (C) <- 0

CJNE A, #data, rel	3	2	10110100 dddddddd eeeeeeee	(PC)<-(PC)+3 Nếu (A) < > #data thì: (PC)<- (PC) + offset Nếu (A) < #data thì: (C) <- 1, ngược lại: (C) <- 0
CJNE Rn, #data, rel	3	2	10111rrr dddddddd eeeeeeee	(PC)<-(PC)+3 Nếu (Rn)< >#data thì: (PC)<- (PC) + offset Nếu (Rn) < #data thì: (C) <- 1, ngược lại: (C) <- 0
Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
CJNE @Ri, #data, rel	3	2	1011011i dddddddd eeeeeeee	(PC)<-(PC)+3 Nếu ((Ri))< >#data thì: (PC)<- (PC) + offset Nếu ((Ri)) < #data thì: (C) <- 1, ngược lại: (C) <- 0

3.4.14. Lệnh giảm và nhảy.

Cú pháp câu lệnh: **DJNZ <byte>, <rel-address>**

Chức năng: Giảm ô nhớ đi 1 và nhảy tới địa chỉ cho bởi toán hạng thứ 2 nếu như kết quả khác 0. Nếu kết quả ban đầu là 00h thì nó chuyển qua 0FFh. Địa chỉ đích được tính bằng cách cộng thêm độ lệch có dấu trong byte lệnh cuối cùng với nội dung của PC (sau khi tăng PC tới byte đầu tiên của lệnh tiếp theo). Ngăn nhớ được giảm giá trị có thể là 1 thanh ghi hoặc 1 byte địa chỉ trực tiếp. Lệnh này không ảnh hưởng tới trạng thái các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
DJNZ Rn, rel	2	2	11011rrr	(PC)<-(PC)+2

			eeeeeeee	(Rn)<- (Rn) - 1 Nếu (Rn) < > 0 thì: (PC) <- (PC) + rel
DJNZ Direct, rel	3	2	11010101 aaaaaaaa eeeeeeee	(PC)<-(PC)+2 (dir.)<- (dir.) - 1 Nếu (dir.) < > 0 thì: (PC) <- (PC) + rel

3.4.15. Lệnh tạm ngừng hoạt động.

Cú pháp câu lệnh: **NOP**

Chức năng: Tạm ngừng hoạt động khi có lệnh này và chương trình sẽ tiếp tục được thực hiện ở lệnh tiếp theo. Lệnh này không ảnh hưởng tới trạng thái các thanh ghi và các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
NOP	1	1	00000000	(PC)<-(PC)+2

3.5. Nhóm lệnh điều khiển biến logic.

3.5.1. Lệnh xoá bit

Cú pháp câu lệnh: **CLR bit**

Chức năng: Xoá bit được chỉ ra trong câu lệnh về 0. Lệnh này có thể thao tác trên cờ nhớ, hoặc trên 1 bit bất kỳ được định địa chỉ trực tiếp. Lệnh không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	***Số byte***	***Số chu kỳ***	***Mã lệnh***	***Hoạt động***
CLR C	1	1	11000011	(C) <- 0
CLR bit	2	1	11000010 bbbbbbbb	(bit) <- 0

3.5.2. Lệnh xoá thanh ghi tích luỹ

Cú pháp câu lệnh: **CLR A**

Chức năng: Xoá tất cả các bit của thanh ghi tích luỹ về 0. Các cờ không bị ảnh hưởng.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
CLR A	1	1	11100100	(A) <- 0

3.5.3. Lệnh thiết lập bit

Cú pháp câu lệnh: **SETB bit**

Chức năng: Thiết lập bit được chỉ ra trong câu lệnh lên mức logic 1. Lệnh này có thể thao tác trên cờ nhớ, hoặc trên 1 bit bất kỳ được định địa chỉ trực tiếp. Lệnh không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
SETB C	1	1	11010011	(C) <- 1
SETB bit	2	1	11010010 bbbbbbbb	(bit) <- 1

3.5.4. Lệnh lấy bù của bit

Cú pháp câu lệnh: **CPL <bit>**

Chức năng: Lấy bù bit được chỉ ra trong câu lệnh. Một bit có giá trị 1 được đổi thành 0 và ngược lại. Lệnh này có thể thao tác trên cờ nhớ, hoặc trên 1 bit bất kỳ được định địa chỉ trực tiếp. Lệnh không làm ảnh hưởng tới trạng tháI các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
CPL C	1	1	10110011	(C) <- NOT (C)
CPL bit	2	1	10110010 bbbbbbbb	(bit) <- NOT (bit)

3.5.5. Lệnh lấy bù của thanh ghi tích luỹ

Cú pháp câu lệnh: **CPL A**

Chức năng: Lấy bù tất cả các bit của thanh ghi A. Lệnh không làm ảnh hưởng tới trạng thái các cờ.

Câu lệnh	*Số byte*	*Số chu kỳ*	*Mã lệnh*	*Hoạt động*
CPL A	1	1	11110100	(A) <- NOT (A)

Phụ lục A **: TRA CỨU NHANH TẬP LỆNH**

Câu lệnh	**Chức năng**	**Các kiểu định đ/chỉ**				**Thời gian Thực hiện(us)**
		Trực tiếp	*Gián tiếp*	*Thanh ghi*	*Tức thời*	
ADD A,<byte>	A = A + <byte>	x	x	x	x	1
ADDC A,<byte>	A = A + <byte> + C	x	x	x	x	1
SUBB A,<byte>	A = A - <byte>	x	x	x	x	1
INC A	A = A + 1	Chỉ với A				1
INC <byte>	<byte> = <byte>+1	x	x	x		1
INC DPTR	DPTR = DPTR + 1	Chỉ với DPTR				2
DEC A	A = A - 1	Chỉ với A				1
DEC <byte>	<byte> = <byte>+1	x	x	x		1
MUL AB	BA = A*B	Chỉ với A&B				4
DIV AB	A=Int(A/B); B=Mode(A/B)	Chỉ với A&B				4
DA A	Hiệu chỉnh số thập phân	Chỉ với A				1

Bảng 1 : *Các lệnh toán học của bộ VĐK họ ATMEN*:

Bảng 2. *Các lệnh chuyển đổi dữ liệu để truy cập vùng nhớ dữ liệu trong*:

Câu lệnh	Chức năng	Các kiểu định địa chỉ				Thời gian Thực hiện(us)
		Trực tiếp	*Gián tiếp*	*Thanh ghi*	*Tức thời*	
MOV A,<scr>	A = <scr>	x	x	x	x	1
MOV <dest>,A	<dest> = A	x	x	x		1
MOV <dest>,<scr>	<dest> = <scr>	x	x	x	x	2
MOV <DPTR>, #data16	DPTR = h/số tức thời 16 bit				x	2
PUSH <scr>	INC SP; Mov "@SP", <scr>	x				2
POP <dest>	Mov <dest>,"@SP" ;DEC SP	x				2
XCH a,<byte>	Đổi dữ liệu giữa A&byte	x	x	x		1
XCHD A,@Ri	Đổi nửa bit thấp giữa A&@Ri		x			1

Bảng 3. *Các lệnh số học*:

Câu lệnh	Chức năng	Các kiểu định địa chỉ				Thời gian Thực hiện(us)
		Trực tiếp	*Gián tiếp*	*Thanh ghi*	*Tức thời*	
ANL A,<byte>	A=A AND <byte>	x	x	x	x	1
ANL <byte>,A	<byte>= <byte> AND A	x				1
ANL <byte>,#data	<byte>= <byte> AND #data	x				2
ORL A, <byte>	A=A OR <byte>	x	x	x	x	1
ORL <byte>,A	<byte>= <byte> OR A	x				1
ORL <byte>,#data	<byte>= <byte> OR #data	x				2

XRL A, <byte>	A=A XOR <byte>	x	x	x	x	1
XRL <byte>,A	<byte>= <byte> XOR A	x				1
XRL <byte>,#data	<byte>= <byte> XOR #data	x				2
CLR A	A = 00h	Chỉ với A				1
CPL A	A = NOT A	Chỉ với A				1
RL A	Dịch A sang trái 1 bit	Chỉ với A				1
RLC A	Dịch A sang trái thông qua C	Chỉ với A				1
RR A	Dịch A sang phải 1 bit	Chỉ với A				1
RRC A	Dịch A sang phải thông qua C	Chỉ với A				1
SWAP A	Đổi nửa bit trong A	Chỉ với A				1

Bảng 4. *Các lệnh đại số*:

Câu lệnh	**Chức năng**	**Thời gian Thực hiện(us)**
ANL C,bit	C = C AND bit	2
ANL C,/bit	C = C AND NOT bit	2
ORL C,bit	C = C ORL bit	2
ORL C,/bit	C = C ORL NOT bit	2
MOV C,bit	C = bit	1
MOV bit,C	Bit = C	2
CLR C	C = 0	1
CLR bit	Bit = 0	1
SETB C	C = 1	1
SETB bit	Bit = 1	1

CPL C	C = NOT C	1
CPL bit	Bit = NOT bit	1
JC rel	Nhảy nếu C = 1	2
JNC rel	Nhảy nếu C = 0	2
JB bit,rel	Nhảy nếu bit = 1	2
JNB bit,rel	Nhảy nếu bit = 0	2
JBC bit,rel	Nhảy nếu bit=1, sau đó xoá bit	2

***Bảng 5**. Các lệnh chuyển đổi dữ liệu để truy cập RAM ngoài*:

Độ lớn địa chỉ	**Câu lệnh**	**Chức năng**	**Th.gian thực hiện (us)**
8 bits	MOVX A,@Ri	Đọc RAM ngoài tại @Ri	2
8 bits	MOVX @Ri,A	Ghi vào RAM ngoài tại @Ri	2
16 bits	MOVX A,@DPTR	Đọc RAM ngoài tại @DPTR	2
16 bits	MOVX @DPTR,A	Ghi vào RAM ngoài tại @DPTR	2

***Bảng 6**. Các lệnh chuyển Byte mã lệnh:*

Câu lệnh	**Chức năng**	**Thời gian thực hiện**
MOVC A,@A+DPTR	Đọc ROM tại (A+DPTR)	2 us
MOVC A,@A+PC	Đọc ROM tại (A+PC)	2 us

***Bảng 7.** Các lệnh nhảy không điều kiện trong Flash Microcontrollers:*

Câu lệnh	**Chức năng**	**Thời gian thực hiện(us)**

JMP addr	Nhảy tới addr.	2
JMP @A+DPTR	Nhảy tới A+DPTR.	2
CALL addr	Gọi C.trình con tại addr.	2
RET	Quay trở về từ C.trình con.	2
RETI	Quay trở về từ ngắt.	2

***Bảng 8**. Các lệnh nhảy có điều kiện:*

Câu lệnh	**Chức năng**	**Các kiểu định địa chỉ**				**Thời gian Thực hiện(us)**
		Trực tiếp	*Gián tiếp*	*Th/ ghi*	*Tức thời*	
JZ rel	Nhảy nếu A=0	Chỉ với A				2
JNZ rel	Nhảy nếu A≠0	Chỉ với A				2
DJNZ <byte>,rel	Giảm & nhảy nếu ≠ 0	x		x		2
CJNE A,<byte>,rel	Nhảy nếu A ≠ <byte>	x			x	2
CJNE <byte>,#data,rel	Nhảy nếu <byte> ≠ #data		x	x		2

Phụ lục B **: CÁC HỆ THỐNG SỐ**

1. BẢNG CHUYỂN ĐỔI HỆ THẬP PHÂN / NHỊ PHÂN.

Hệ thập phân	**Hệ nhị phân**		**Hệ thập phân**	**Hệ nhị phân**
0	00000000		32	00100000
1	00000001		33	00100001
2	00000010		.	
3	00000011		.	
4	00000100		.	
5	00000101		.	
6	00000110		.	
7	00000111		.	

8	00001000		.	
9	00001001		63	00111111
10	00001010		64	01000000
11	00001011		65	01000001
12	00001100		.	
13	00001101		.	
14	00001110		.	
15	00001111		.	
16	00010000		.	
17	00010001		.	
.			.	
.			127	01111111
.			128	10000000
.			129	10000001
.			.	
.			.	
.			.	
.			.	
.			.	
.			.	
.			.	
.			.	
.			254	11111110
31	00011111		255	11111111

2. BẢNG MÃ THẬP LỤC PHÂN:

Hệ thập phân	**Hệ nhị phân**	**Hệ thập lục phân**
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6

7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

3. HỆ THỐNG SỐ CÓ DẤU:

Hệ nhị phân	**Hệ thập lục phân**	**Hệ thập phân**
0111 1111	7F	+ 127
0111 1110	7E	+ 126
0111 1101	7D	+ 125
.	.	.
.	.	.
.	.	.
0001 0000	10	+ 16
0000 1111	0F	+ 15

.	.	.
.	.	.
.	.	.
.	.	.
0000 0010	02	+ 2
0000 0001	01	+ 1
0000 0000	00	0
1111 1111	FF	- 1
1111 1110	FE	- 2
1111 1101	FD	- 3
.	.	.
.	.	.
.	.	.
1001 0000	90	- 112
1000 1111	8F	- 113
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
1000 0010	82	- 126
1000 0001	81	- 127
1000 0000	80	- 128

TÀI LIỆU THAM KHẢO:

1. The 8051 Microcontroller - I. Scott Mackenzie.
2. The MCS*51 Microcontroller Family User's Manuel - INTEL - 1994.
3. The AT89 Family of Microcontrollers - ATMEL - 2003.
4. Microcomputer Components – SAB80C515 8 bit Single-chip Microcontroller Family - SIEMENS - 1995.

5. Mikrocomputertechnik – Prof.Dr.Ing. G.Schnell – Fachhochschule Frankfurt am Main - 2001.
6. Kỹ thuật Vi xử lý - Văn Thế Minh - NXB GD - 1997.
7. Kỹ thuật VXL & lập trình ASSEMBLY cho hệ VXL - Đỗ Xuân Tiến – NXB KH&KT - 2001.
8. Họ VĐK 8051 - Tống Văn On & Hoàng Đức Hải - NXB LĐXH - 2001

BỘ GIAO DỤC & ĐAO TẠO
TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHỆ
THÀNH PHỐ HỒ CHÍ MINH
KHOA ĐIỆN – ĐIỆN TỬ
--- oOo ---

GIÁO TRÌNH

VI XỬ LÝ

Tác giả: ThS. PHẠM HÙNG KIM KHÁNH

08/2006

LỜI NÓI ĐẦU

Giáo trình Vi xử lý được biên soạn nhằm cung cấp cho sinh viên kiến thức cơ bản về vi xử lý, cấu trúc của một hệ vi xử lý cũng như cách thức lập trình điều khiển thiết bị dựa cơ sở trên Vi xử lý 8086/8088.

Giáo trình được sử dụng cho khóa học 60 tiết dành cho sinh viên hệ đại học Khoa Điện Điện tử trường Đại học Dân lập Kỹ thuật Công nghệ TPHCM.

Bố cục giáo trình gồm 4 chương dựa theo đề cương môn học Kỹ thuật Vi xử lý dành cho sinh viên ngành Điện Tử Viễn Thông:

Chương 1. Tổ chức hệ thống Vi xử lý

Chương 2. Lập trình hợp ngữ

Chương 3. Tổ chức nhập / xuất

Chương 4. Giao tiếp với các thiết bị đơn giản

Phụ lục 1: 8255

Phụ lục 2: Tập lệnh của họ 8086

PHẠM HÙNG KIM KHÁNH

MỤC LỤC

CHƯƠNG 1: TỔ CHỨC HỆ THỐNG VI XỬ LÝ

1. Các hệ thống số dùng trong máy tính và các loại mã

1.1. Hệ thập phân (Decimal Number System)

Trong thực tế, ta thường dùng hệ thập phân để biểu diễn các giá trị số. Ở hệ thống này, ta dùng các tổ hợp của các chữ số 0..9 để biểu diễn các giá trị. Một số trong hệ thập phân được biểu diễn theo các số mũ của 10.

VD: Số 5346.72 biểu diễn như sau:

$5346.72 = 5x10^3 + 3x10^2 + 4x10 + 6 + 7x10^{-1} + 2x10^{-2}$

Tuy nhiên, trong các mạch điện tử, việc lưu trữ và phân biệt 10 mức điện áp khác nhau rất khó khăn nhưng việc phân biệt hai mức điện áp thì lại dễ dàng. Do đó, người ta sử dụng hệ nhị phân để biểu diễn các giá trị trong hệ thống số.

1.2. Hệ nhị phân (Binary Number System)

Hệ nhị phân chỉ dùng các chữ số 0 và 1 để biểu diễn các giá trị số. Một số nhị phân (binary digit) thường được gọi là ***bit***. Một chuỗi gồm 4 bit nhị phân gọi là ***nibble***, chuỗi 8 bit gọi là ***byte***, chuỗi 16 bit gọi là ***word*** và chuỗi 32 bit gọi là ***double word***. Chữ số nhị phân bên phải nhất của chuỗi bit gọi là ***bit có ý nghĩa nhỏ nhất*** (***least significant bit – LSB***) và chữ số nhị phân bên trái nhất của chuỗi bit gọi là ***bit có ý nghĩa lớn nhất*** (***most significant bit – MSB***). Một số trong hệ nhị phân được biểu diễn theo số mũ của 2. Ta thường dùng chữ ***b*** cuối chuỗi bit để xác định đó là số nhị phân.

VD: Số 101110.01b biểu diễn giá trị số:

$101110.01b \rightarrow 1x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 1x2^1 + 0 + 0x2^{-1} + 1x2^{-2}$

❖ **Chuyển số nhị phân thành số thập phân:**

Để chuyển một số nhị phân thành một số thập phân, ta chỉ cần nhân các chữ số của số nhị phân với giá trị thập phân của nó và cộng tất cả các giá trị lại.

VD: $1011.11B \rightarrow 1x2^3 + 0x2^2 + 1x2^1 + 1 + 1x2^{-1} + 1x2^{-2} = 11.75$

❖ **Chuyển số thập phân thành số nhị phân:**

Để chuyển một số thập phân thành số nhị phân, ta dùng 2 phương pháp sau:

- **Phương pháp 1**: Ta lấy số thập phân cần chuyển trừ đi 2^i trong đó 2^i là số lớn nhất nhỏ hơn hay bằng số thập phân cần chuyển. Sau đó, ta lại lấy kết quả này và thực hiện tương tự cho đến 2^0 thì dừng. Trong quá trình thực hiện, ta sẽ ghi lại các giá trị 0 hay 1 cho các bit tuỳ theo trường hợp số thập phân nhỏ hơn 2^i (0) hay lớn hơn 2^i (1).

VD: Xét số 21 thì số 2^i lớn nhất là 2^4

	2^4	2^3	2^2	2^1	2^0
	16	8	4	2	1
21 =	1	0	1	0	1
	5	5	1	1	0

(21 → 10101B)

- **Phương pháp 2**: Lấy số cần chuyển chia cho 2, ta nhớ lại số dư và lấy tiếp thương của kết quả trên chia cho 2 và thực hiện tương tự cho đến khi thương cuối cùng bằng 0. Kết quả chuyển đổi sẽ là chuỗi các bit là các số dư lấy theo thứ tự ngược lại.

VD: Chuyển 227 ra số nhị phân

Số bị chia	Thương	Số dư
227	113	1 (LSB)
113	56	1
56	28	0
28	14	0
14	7	0
7	3	1
3	1	1
1	0	1 (MSB)

(227 → 11100011b)

- Để thực hiện chuyển các số thập phân nhỏ hơn 1 sang các số nhị phân, ta làm như sau: lấy số cần chuyển nhân với 2, giữ lại phần nguyên và lại lấy phần lẻ nhân với 2. Quá trình tiếp tục cho đến khi phần lẻ bằng 0 thì dừng. Kết quả chuyển đổi là chuỗi các bit là giá trị các phần nguyên.

VD: Chuyển 0.625 thành số nhị phân:

0.625 × 2 = 1.25
0.25 × 2 = 0.5
0.5 × 2 = 1.0
(0.625 = 0.101b)

- Để thực hiện chuyển đổi số nhị phân bất kỳ, ta thực hiện chuyển đổi tương ứng với số nhị phân lớn hơn 1 và nhỏ hơn 1 như trên.

VD: Chuyển 227.625 thành số nhị phân:

227 → 11100011b
0.625 → 0.101b
227.625 → 11100011.101b

1.3. Hệ thập lục phân (Hexadecimal Number System)

Như đã biết ở trên, nếu dùng hệ nhị phân thì sẽ cần một số lượng lớn các bit để biểu diễn. Giả sử như số $1024 = 2^{10}$ sẽ cần 10 bit để biểu diễn. Để rút ngắn kết quả

biểu diễn, ta dùng hệ thập lục phân dựa cơ sở trên số mũ của 16. Khi đó, 4 bit trong hệ nhị phân (1 nibble) sẽ biểu diễn bằng 1 chữ số trong hệ thập lục phân (gọi là số hex).

Trong hệ thống này, ta dùng các số 0..9 và các kí tự A..F để biểu diễn cho một giá trị số. Thông thường, ta dùng chữ ***h*** ở cuối để xác định đó là số thập lục phân.

1.4. Mã BCD (Binary Coded Decimal)

Trong thực tế, đối với một số ứng dụng như đếm tần, đo điện áp, … ngõ ra ở dạng số thập phân, ta dùng mã BCD. Mã BCD dùng 4 bit nhị phân để mã hoá cho một số thập phân 0..9. Như vậy, các số hex A..F không tồn tại trong mã BCD.

Mã BCD gồm có 2 loại:

- Mã BCD không nén (unpacked): biểu diễn một số BCD bằng 8 bit nhị phân
- Mã BCD nén (packed): biểu diễn một số BCD bằng 4 bit nhị phân

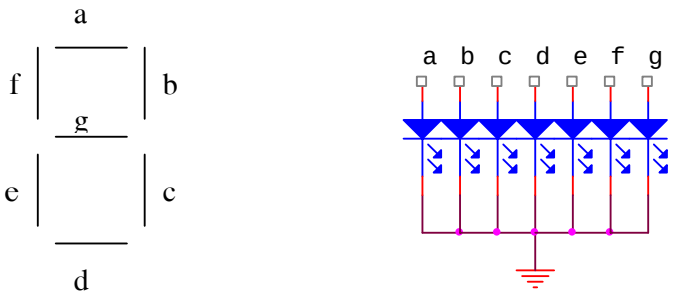
VD: Số thập phân 5 2 9

Số BCD không nén 0000 0101b 0000 0010b 0000 1001b

Số BCD nén 0101b 0010b 1001b

1.5. Mã hiển thị Led 7 đoạn (7-segment display)

Đối với các ứng dụng dùng hiển thị số liệu ra Led 7 đoạn, ta dùng mã hiển thị Led 7 đoạn. Ứng với mỗi loại Led 7 đoạn (anode hay cathode chung) và tuỳ theo sơ đồ kết nối sẽ có một bảng mã riêng. Một ví dụ của mã Led 7 đoạn cho trong bảng 1.1.



Hình 1.1 – Led 7 đoạn dạng cathode chung

Bảng 1.1:

Số thập phân	Số thập lục phân	Số nhị phân	Mã Led 7 đoạn	
			a b c d e f g	Hiển thị
0	0	0000	1 1 1 1 1 1 0	0
1	1	0001	0 1 1 0 0 0 0	1
2	2	0010	1 1 0 1 1 0 1	2
3	3	0011	1 1 1 1 0 1 1	3
4	4	0100	0 1 1 0 0 1 1	4
5	5	0101	1 0 1 1 0 1 1	5
6	6	0110	1 0 1 1 1 1 1	6
7	7	0111	1 1 1 0 0 0 0	7

8	8	1000	1 1 1 1 1 1 1	8
9	9	1001	1 1 1 0 0 1 1	9
10	A	1010	1 1 1 1 1 0 1	A
11	B	1011	0 0 1 1 1 1 1	B
12	C	1100	0 0 0 1 1 0 1	C
13	D	1101	0 1 1 1 1 0 1	D
14	E	1110	1 1 0 1 1 1 1	E
15	F	1111	1 0 0 0 1 1 1	F

2. Các phép toán số học

2.1. Hệ nhị phân

2.1.1. Phép cộng

Phép cộng trong hệ nhị phân cũng thực hiện giống như trong hệ thập phân. Bảng sự thật của phép cộng 2 bit với 1 bit nhớ (carry) như sau:

Bảng 1.2:

Vào			Ra	
A	**B**	**C_{IN}**	**S**	**C_{OUT}**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$S = A \oplus B \oplus C_{IN}$

$C_{OUT} = AB + C_{IN}(A \oplus B)$

VD: 1001 1010b

1 +1100 1100b

Nhớ 0111 0110b

2.1.2. Số bù 2 (2's component)

Trong hệ thống số thông thường, để biểu diễn số âm ta chỉ cần thêm dấu – vào các chữ số. Tuy nhiên, trong hệ thống máy tính, ta không thể biểu diễn được như trên. Phương pháp thông dụng là dùng bit có ý nghĩa lớn nhất (MSB) làm bit dấu (sign bit): nếu MSB = 1 sẽ là số âm còn MSB = 0 là số dương. Khi đó, các bit còn lại sẽ biểu diễn độ lớn (magnitude) của số. Như vậy, nếu ta dùng 8 bit để biểu diễn thì sẽ thu được 256 tổ hợp ứng với các giá trị 0..255 (số không dấu) hay –127.. –0 +0 … +127 (số có dấu).

Để thuận tiện hơn trong việc tính toán số có dấu, ta dùng một dạng biểu diễn đặc biệt là số bù 2. Số bù 2 của một số nhị phân xác định bằng cách lấy đảo các bit rồi cộng thêm 1.

VD: Số 7 biểu diễn là : 0000 0111b có MSB = 0 (biểu diễn số dương)

Số bù 2 là : 111 1000b + 1b = 111 1001b. Số đại diện cho số – 7 là: 1111 1001b có MSB = 1 (biểu diễn số âm)

Ta thấy, để thực hiện việc xác định số bù 2 của một số A, cần phải:

- Biểu diễn số A theo mã bù 2 của nó.
- Đảo các bit (tìm số bù 1 của A).
- Cộng thêm 1 vào để nhận được số bù 2.

Khi biểu diễn theo số bù 2, nếu sử dụng 8 bit ta sẽ có các giá trị số thay đổi từ -128..127.

2.1.3. Phép trừ

Phép trừ các số nhị phân cũng được thực hiện tương tự như trong hệ thập phân. Bảng sự thật của phép trừ 2 bit với 1 bit mượn (borrow) như sau:

Bảng 1.3:

Vào			Ra	
A	**B**	**BIN**	**D**	**B_{OUT}**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$S = A \oplus B \oplus BIN$

$BOUT = \overline{A}B + \overline{(A \oplus B)}B_{IN}$

VD: 0110 1101b → 149

- 0011 0001b → 49

0011 1100b → 100

Ngoài cách trừ như trên, ta cũng có thể thực hiện phép trừ thông qua số bù 2 của số trừ.

VD: 0110 1101b

- 0011 0001b → + 1100 1111b

1 0011 1100b

Số bù 1 Nhớ

100 1110b + 1b = 100 1111b (Số bù 2)

Trong phép cộng với số bù 2, ta bỏ qua bit nhớ cuối cùng → kết quả phép cộng số bù 2 là 0011 1100. Đây cũng chính là kết quả phép trừ, bit MSB = 0 cho biết kết quả là số dương.

VD:

77	0100 1101b		0100 1101b
- 88	- 0101 1000b	→	+ 1010 1000b
- 11			1111 0101b

Số 88 → 0101 1000b → số bù 1 là 010 0111 → số bù 2: 010 1000 và bit dấu = 1 → -88 trở thành 1010 1000b

Kết quả phép cộng số bù 2 là 1111 0101b có MSB = 1 nên là số âm. Số bù 1 là 000 1010b → số bù 2: 000 1011b. Kết quả này chính là 11 nên phép trừ sẽ cho kết quả là –11.

Ta thấy, để thực hiện chuyển số bù 2 thành số có dấu thì cần thực hiện:

- Lấy bù các bit để tìm số bù 1.
- Cộng với 1.
- Thêm dấu trừ để xác định là số âm.

2.1.4. Phép nhân

Phép nhân các số nhị phân cũng tương tự như đối với các số thập phân. Chú ý rằng đối với phép nhân nếu nhân 2 số 4 bit sẽ có kết quả là số 8 bit, 2 số 8 bit sẽ có kết quả là số 16 bit, …

VD:

11	1011b
X 9	1001b
99	1011
	0000
	0000
	1011
	1100011b

Đối với máy tính, phép nhân được thực hiện bằng phương pháp cộng và dịch phải (add-and-right-shift):

- Thành phần dầu tiên của tổng sẽ chính là số bị nhân nếu như LSB của số nhân là 1. Ngược lại, nếu LSB của số nhân bằng 0 thì thành phần này bằng 0.
- Mỗi thành phần thứ i kế tiếp sẽ được tính tương tự với điều kiện là phải dịch trái số bị nhân i bit.
- Kết quả cần tìm chính là tổng các thành phần nói trên.

2.1.5. Phép chia

Phép chia các số nhị phân cũng tương tự như đối với các số thập phân.

VD: 30/5 = 6

```
11110 b  | 110b
110      | 101b
 011     |
 000     |
  110    |
  110    |
    0    |
```

Tương tự như đối với phép nhân, ta có thể dùng phép trừ và phép dịch trái cho đến khi không thể thực hiện phép trừ được nữa. Tuy nhiên, để thuận tiện cho tính toán, thay vì dùng phép trừ đối với số chia, ta sẽ thực hiện phép cộng đối với số bù 2 của số chia.

- Đổi số chia ra số bù 2 của nó.
- Lấy số bị chia cộng với số bù 2 của số chia.
  + Nếu kết quả này có bit dấu = 0 thì bit tương ứng của thương = 1.
  + Nếu kết quả này có bit dấu = 1 thì bit tương ứng của thương = 0 và ta phải khôi phục lại giá trị của số bị chia bằng cách cộng kết quả này với số chia.
- Dịch trái kết quả thu được và thực hiện tiếp tục như trên cho đến khi kết quả là 0 hay nhỏ hơn số chia.

2.2. Hệ thập lục phân

2.2.1. Phép cộng

Thực hiện chuyển các số hex cần cộng thành các số nhị phân, tính kết quả trên số nhị phân và sau đó chuyển lại thành số hex.

VD: 7Ah → 0111 1010b

3Fh → 0011 1111b

B9h ← 1011 1001b

Thực hiện cộng trực tiếp trên số hex, nếu kết quả cộng lớn hơn 15 thì sẽ nhớ và trừ cho 16.

VD: 7 Ah

3 Fh

10_{10} 25_{10} → B9h

Ah + Fh = $10_{10} + 15_{10} = 25_{10}$ → nhớ 1 và $25_{10} - 16_{10} = 9_{10} = 9h$

7h + 3h = $7_{10} + 3_{10} = 10_{10}$ → cộng số nhớ: $10_{10} + 1_{10} = 11_{10} = Bh$

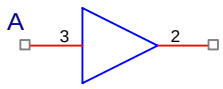
2.2.2. Phép trừ

Thực hiện tương tự như phép cộng.

3. Các thiết bị số cơ bản

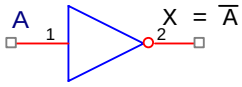
3.1. Cổng đệm (buffer) và các cổng logic (logic gate)

❖ Cổng đệm:



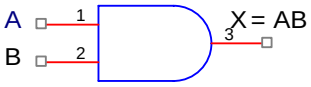
A	X
0	0
1	1

❖ Cổng NOT:



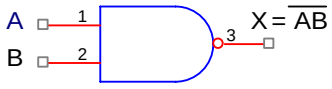
A	X
0	1
1	0

❖ Cổng AND:



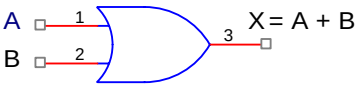
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

❖ Cổng NAND:



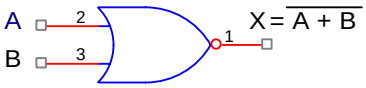
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

❖ Cổng OR:



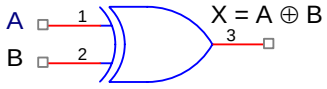
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

❖ Cổng NOR:



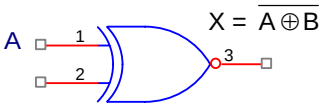
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

❖ **Cổng EX-OR:**



A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

❖ **Cổng EX-NOR:**



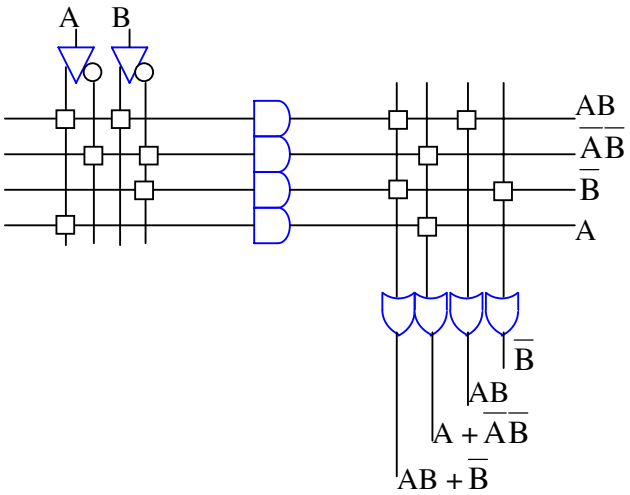
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

3.2. Thiết bị logic lập trình được

Thay vì sử dụng các cổng logic rời rạc, ta có thể dùng các thiết bị logic lập trình được (programmable logic device) như PLA (Programmable Logic Array), PAL (Programmable Array of Logic) để liên kết các thiết bị LSI (Large Scale Intergration).

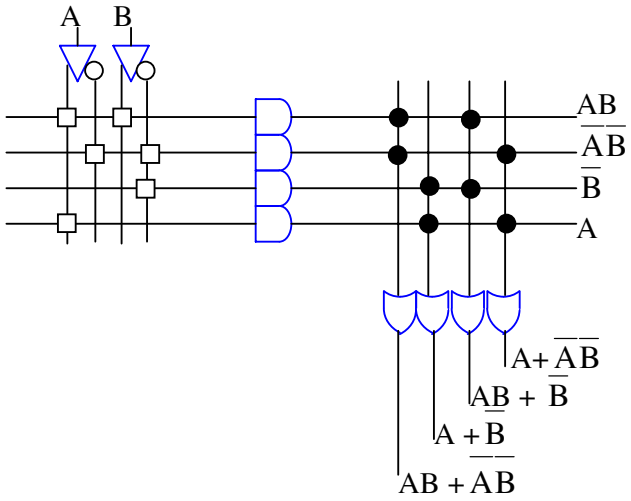
❖ **PLA (hay FPLA – Field PLA):**

Dùng ma trận cổng AND và OR để lập trình bằng cácc phá huỷ các cầu chì. FPLA rất linh động nhưng lại khó lập trình.



Hình 1.2 – Sơ đồ PLA

❖ **PAL**: ma trận OR đã cố định sẵn và ta chỉ lập trình trên ma trận AND.

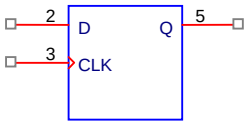


Hình 1.3 – Sơ đồ PAL

3.3. Chốt, flipflop và thanh ghi

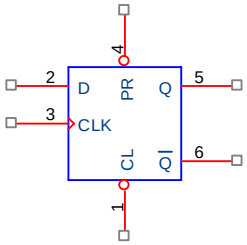
❖ **Chốt (latch):**

Chốt là thiết bị số lưu trữ lại giá trị số tại ngõ ra của nó.



D	CLK	Q
X	0	QN
0	1	0
1	1	1

❖ **Flipflop:**



PR	CL	D	CLK	Q	$\overline{Q}$
1	1	1	↑	1	0
1	1	0	↑	0	1
1	1	X	0	Q_N	$\overline{Q_N}$
1	1	X	1	Q_N	$\overline{Q_N}$
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	.	.

CL: clear PR: Preset CLK: Clock

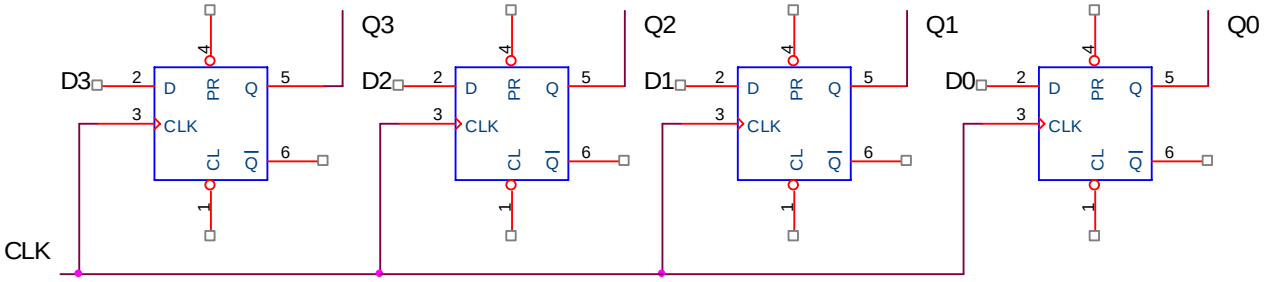
- Nếu xuất hiện cạnh lên của tín hiệu CLK thì ngõ ra Q sẽ có giá trị theo dữ liệu tại D.

- Nếu PR = 0 thì Q = 1. Nếu CL = 0 thì Q = 0.

- Trạng thái PR = CL = 0 là trạng thái cấm, ngõ ra sẽ không ổn định.

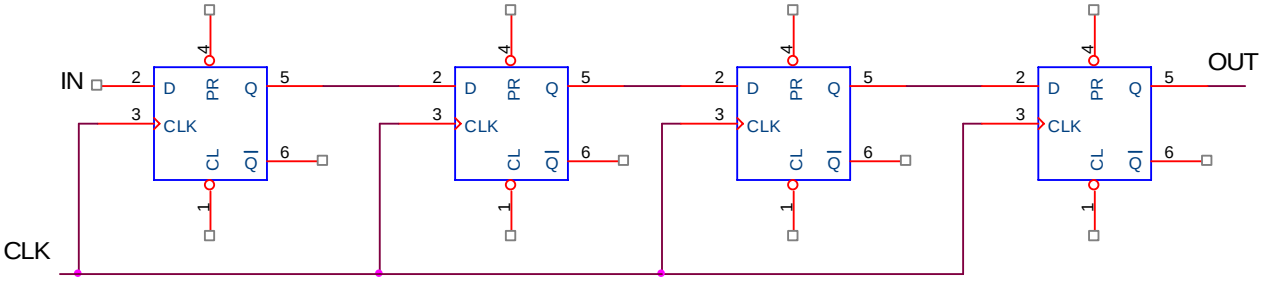
❖ **Thanh ghi (register):**

Thanh ghi là một nhóm các flipflop được kết nối song song để lưu trữ các số nhị phân. Giá trị nhị phân sẽ được đưa vào ngõ vào của các flipflop. Khi có tác động cạnh lên của tín hiệu CLK thì ngõ ra các flipflop sẽ lưu trữ giá trị nhị phân cho đến khi một số nhị phân mới được đưa vào và tác động một cạnh len cho tín hiệu CLK.



Hình 1.4 – Thanh ghi dạng đơn giản

Trong trường hợp các flipflop được kết nối nối tiếp với nhau, ta sẽ có thanh ghi dịch (shift register).



Hình 1.5 – Thanh ghi dịch

3.4. Bộ nhớ

3.4.1. Các kiểu bộ nhớ

❖ **ROM (Read Only Memory):**

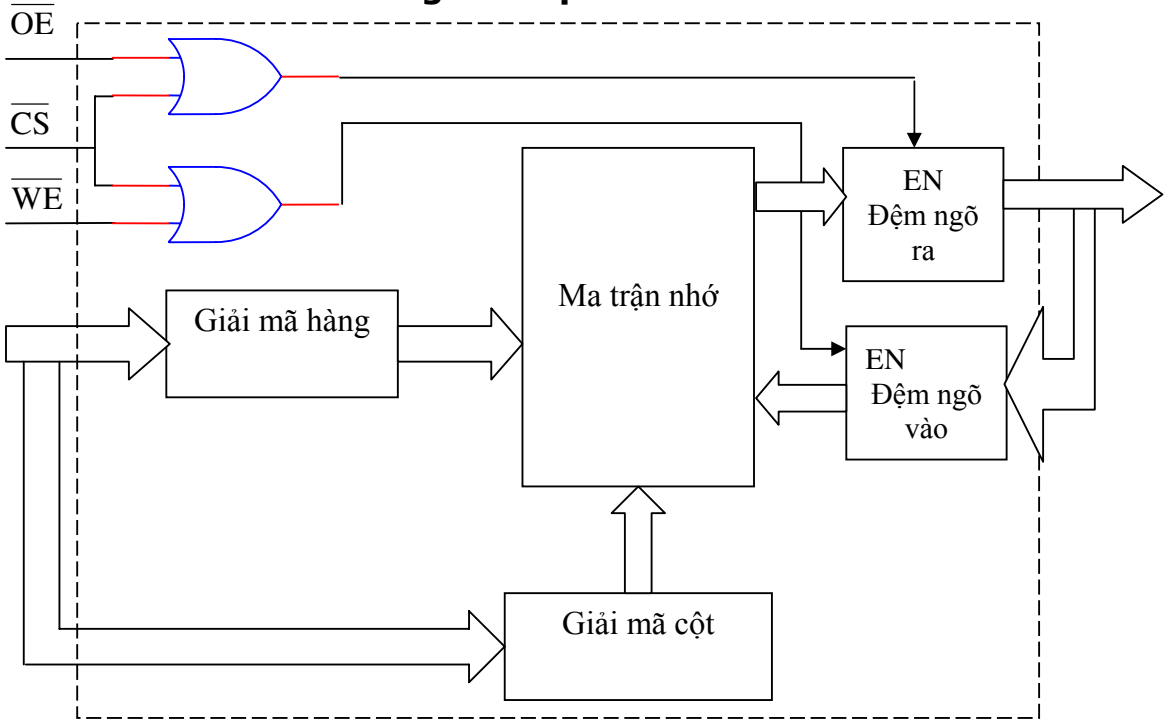
Đặc tính chung của ROM là dữ liệu lưu trữ sẽ không bị mất đi dù cho không còn nguồn cung cấp cho ROM (tính nonvolatile – ổn định). Ta chỉ có thể thực hiện tác vụ đọc đối với ROM. ROM có thể được chia thành: ROM che mặt nạ (Masked ROM), PROM (ROM lập trình được), EPROM (ROM có thể xoá bằng tia cực tím) và EEPROM (ROM có thể xoá bằng điện).

❖ **RAM (Random Access Memory):**

RAM có đặc tính là tất cả nội dung chứa trong RAM sẽ bị mất đi khi không còn nguồn cung cấp cho RAM (tính volatile – không ổn định). Có 2 loại RAM: tĩnh và động.

- **SRAM (Static RAM):** dùng các ma trận flipflop để lưu trữ dữ liệu nên ta có thể ghi các giá trị nhị phân vào RAM bằng cách đưa dữ liệu vào các ngõ vào các flipflop và cấp xung clock cho các flipflop này.
- **DRAM (Dynamic RAM):** tạo ra bằng các cổng transistor và lưu trữ bằng điện tích. Tuy nhiên, do hiện tượng rò rỉ điện tích theo thời gian, ta phải thực hiện nạp điện lại. Quá trình này gọi là làm tươi (refreshing) bộ nhớ. Thuận lợi của DRAM là một số lượng lớn transistor có thể được đặt trên một chip nhớ nên nó có dung lượng cao hơn và nhanh hơn SRAM.

3.4.2. Cấu trúc bên trong của bộ nhớ



Hình 1.6 – Cấu trúc nội một bộ nhớ tiêu biểu

$\overline{CS}$ (Chip Select):cho phép bộ nhớ hoạt động

$\overline{OE}$ (Output Enable): cho phép đọc dữ liệu từ bộ nhớ ra ngoài

$\overline{WE}$ (Write Enable): cho phép ghi dữ liệu vào trong bộ nhớ

4. Giới thiệu vi xử lý

4.1. Các thế hệ vi xử lý

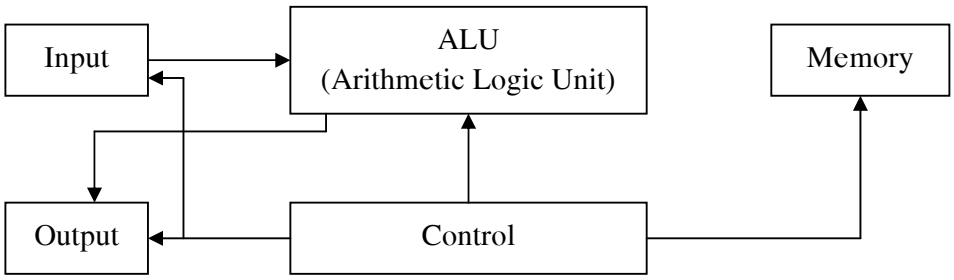
- ***Thế hệ 1 (1971 – 1973):*** vi xử lý 4 bit, đại diện là 4004, 4040, 8080 (Intel) hay IPM-16 (National Semiconductor).
  + Độ dài word thường là 4 bit (có thể lớn hơn).
  + Chế tạo bằng công nghệ PMOS với mật độ phần tử nhỏ, tốc độ thấp, dòng tải thấp nhưng giá thành rẻ.
  + Tốc độ 10 ÷ 60 μs / lệnh với tần số xung nhịp 0.1 ÷ 0.8 MHz.
  + Tập lệnh đơn giản và phải cần nhiều vi mạch phụ trợ.
- ***Thế hệ 2 (1974 – 1977):*** vi xử lý 8 bit, đại diện là 8080, 8085 (Intel) hay Z80 (Zilog).
  + Tập lệnh phong phú hơn.
  + Địa chỉ có thể đến 64 KB. Một số bộ vi xử lý có thể phân biệt 256 địa chỉ cho thiết bị ngoại vi.
  + Sử dụng công nghệ NMOS hay CMOS.
  + Tốc độ 1 ÷ 8 μs / lệnh với tần số xung nhịp 1 ÷ 5 MHz
- ***Thế hệ 3 (1978 – 1982):*** vi xử lý 16 bit, đại diện là 68000/68010 (Motorola) hay 8086/80286/80386 (Intel)
  + Tập lệnh đa dạng với các lệnh nhân, chia và xử lý chuỗi.
  + Địa chỉ bộ nhớ có thể từ 1 ÷ 16 MB và có thể phân biệt tới 64KB địa chỉ cho ngoại vi
  + Sử dụng công nghệ HMOS.
  + Tốc độ 0.1 ÷ 1 μs / lệnh với tần số xung nhịp 5 ÷ 10 MHz.
- ***Thế hệ 4:*** vi xử lý 32 bit 68020/68030/68040/68060 (Motorola) hay 80386/80486 (Intel) và vi xử lý 32 bit Pentium (Intel)
  + Bus địa chỉ 32 bit, phân biệt 4 GB bộ nhớ.
  + Có thể dùng thêm các bộ đồng xử lý (coprocessor).
  + Có khả năng làm việc với bộ nhớ ảo.
  + Có các cơ chế pipeline, bộ nhớ cache.
  + Sử dụng công nghệ HCMOS.
- ***Thế hệ 5:*** vi xử lý 64 bit

4.2. Vi xử lý (μP – microproccessor)

4.2.1. Phân loại vi xử lý

- ***Multi chip***: dùng 2 hay nhiều chip LSI (Large Scale Intergration: tích hợp từ 1000 ÷ 10000 transistor) cho ALU và control.
- ***Microprocessor***: dùng 1 chip LSI/VLSI (Very Large Scale Intergration: tích hợp ÷ 10000 transistor) cho ALU và control.
- ***Single chip microprocessor*** (còn gọi là microcomputer / microcontroller): là 1 chip LSI/VLSI chứa toàn bộ các khối như hình 1.7.

4.2.2. Sơ đồ khối một máy tính cổ điển



Hình 1.7 – Sơ đồ khối một máy tính cổ điển

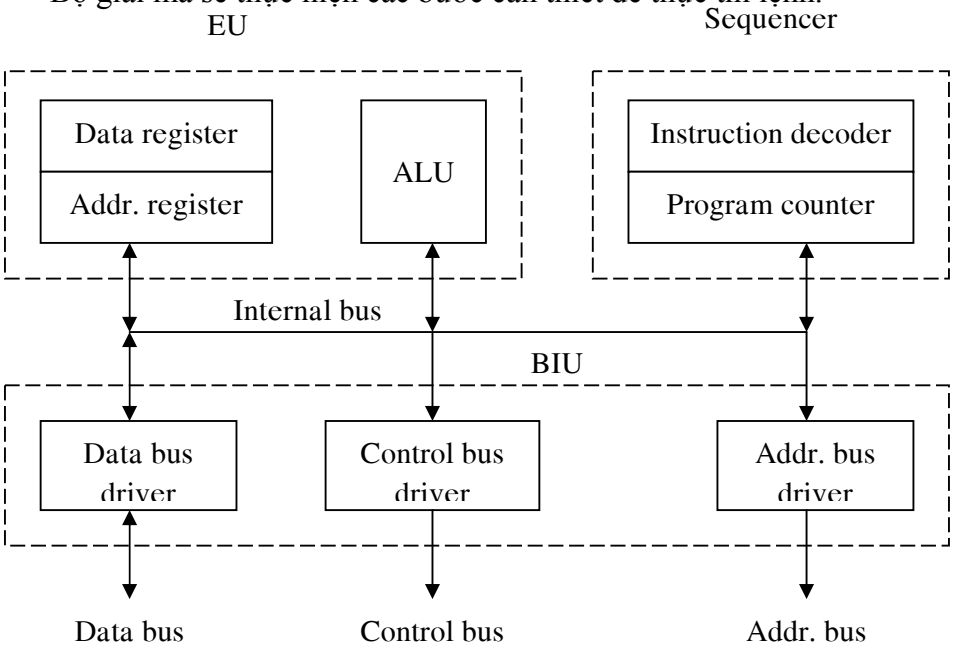
- *ALU (đơn vị logic số học)*: thực hiện các bài toán cho máy tính bao gồm: +, -, *, /, phép toán logic, …
- *Control (điều khiển)*: điều khiển, kiểm soát các đường dữ liệu giữa các thành phần của máy tính.
- *Memory (bộ nhớ)*: lưu trữ chương trình hay các kết quả trung gian.
- *Input (nhập), Output (Xuất)*: xuất nhập dữ liệu (còn gọi là thiết bị ngoại vi).

4.2.3. Sơ đồ khối của μP

Có 3 khối chức năng: đơn vị thực thi (EU - Execution unit), bộ tuần tự (Sequencer) và đơn vị giao tiếp bus (BIU – Bus interface unit).

- EU: thực hiện các lệnh số học và logic. Các toán hạng được chứa trong các thanh ghi dữ liệu (data register) hay thanh ghi địa chỉ (address register), hay từ bus nội (internal bus).
- Bộ tuần tự: gồm bộ giải mã lệnh (instruction decoder) và bộ đếm chương trình (program counter)

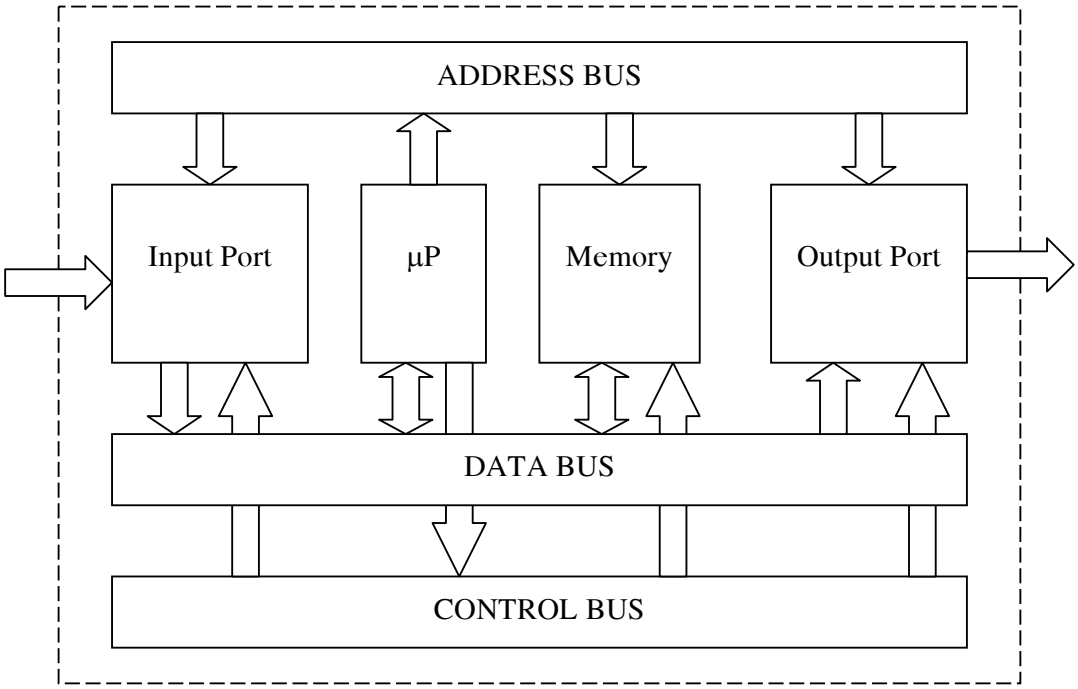
+ Bộ đếm chương trình chứa các lệnh kế tiếp sẽ thực hiện
+ Bộ giải mã sẽ thực hiện các bước cần thiết để thực thi lệnh.



Hình 1.8 – Sơ đồ khối của vi xử lý

Khi chương trình bắt đầu, bộ đếm chương trình (PC) sẽ ở địa chỉ bắt đầu. Địa chỉ này được chuyển qua bộ nhớ thông qua address bus. Khi tín hiệu Read đưa vào control bus, nội dung bộ nhớ liên quan sẽ đưa vào bộ giải mã lệnh. Bộ giải mã lệnh sẽ khởi động các phép toán cần thiết để thực thi lệnh. Quá trình này đòi hỏi một số chu kỳ máy (machine cycle) tuỳ theo lệnh. Sau khi lệnh đã thực thi, bộ giải mã lệnh sẽ đặt PC đến địa chỉ của lệnh kế.

4.2.4. Sơ đồ khối của hệ vi xử lý cơ bản



Hình 1.9 – Sơ đồ khối hệ vi xử lý

Mọi hoạt động cơ bản của một hệ vi xử lý đều giống nhau, không phụ thuộc loại vi xử lý hay quá trình thực hiện. µP sẽ đọc một lệnh từ bộ nhớ (memory), thực thi lệnh và sau đó đọc lệnh kế. Quá trình đọc lệnh gọi là instruction fetch còn quá trình thực hiện tuần tự như trên gọi là fetch – execute sequence. Tuy nhiên có một số µP sẽ nhận một số lệnh rồi mới bắt đầu thực thi.

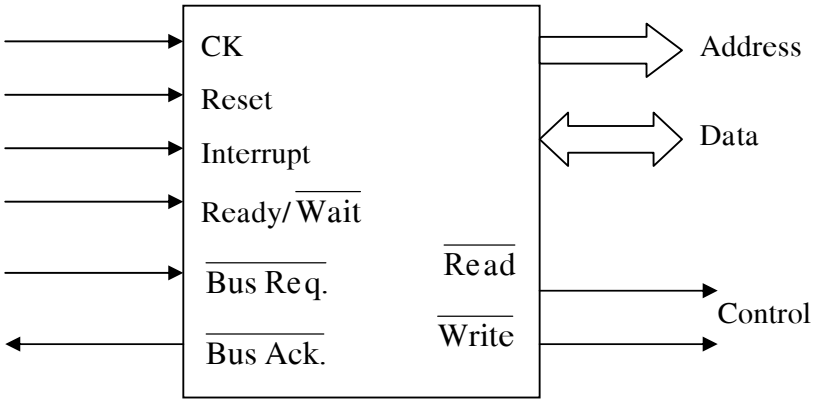
❖ **Các port I/O:**

Các port nhập (input) và xuất (output) dùng để giao tiếp giữa µP và thiết bị ngoại vi (không thể nối trực tiếp với các bus).

Port xuất là một thanh ghi. Khi µP ghi dữ liệu ra địa chỉ của Port thì Port sẽ chứa dữ liệu hiện tại trên data bus. Dữ liệu này sẽ được chốt tại Port cho đến khi µP ghi dữ liệu mới ra Port.

Port nhập là một driver 3 trạng thái. Khi µP đọc vào từ địa chỉ của Port, driver 3 trạng thái lái dữ liệu từ bên ngoài vào data bus. Sau đó, µP đọc dữ liệu từ bus.

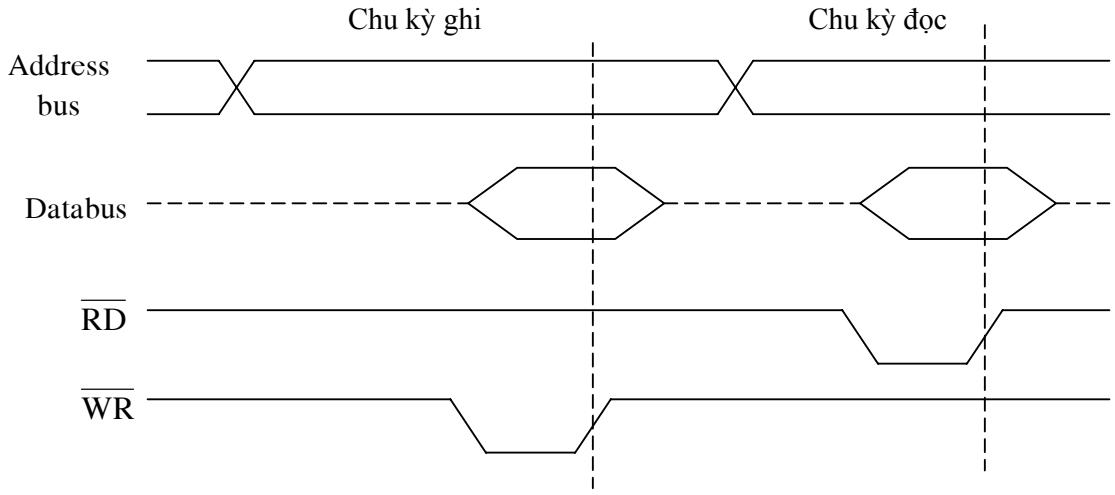
❖ **Các tín hiệu tiêu biểu của một µP:**



Hình 1.10 – Các tín hiệu cơ bản trong µP

Các bus dùng để liên kết các thành phần của hệ thống với µP. µP sẽ chọn một thiết bị cần sử dụng thông qua address bus và đọc hay ghi dữ liệu thông qua data bus. Data bus là bus 2 chiều, dùng chung cho tất cả các quá trình trao đổi dữ liệu. Mỗi chu kỳ bus (bus cycle) là việc thực hiện trao đổi một từ dữ liệu giữa µP và ô nhớ hay thiết bị I/O.

Mỗi chu kỳ bus bắt đầu khi µP xuất một địa chỉ nhằm chọn thiết bị I/O hay chọn một ô nhớ nào đó.



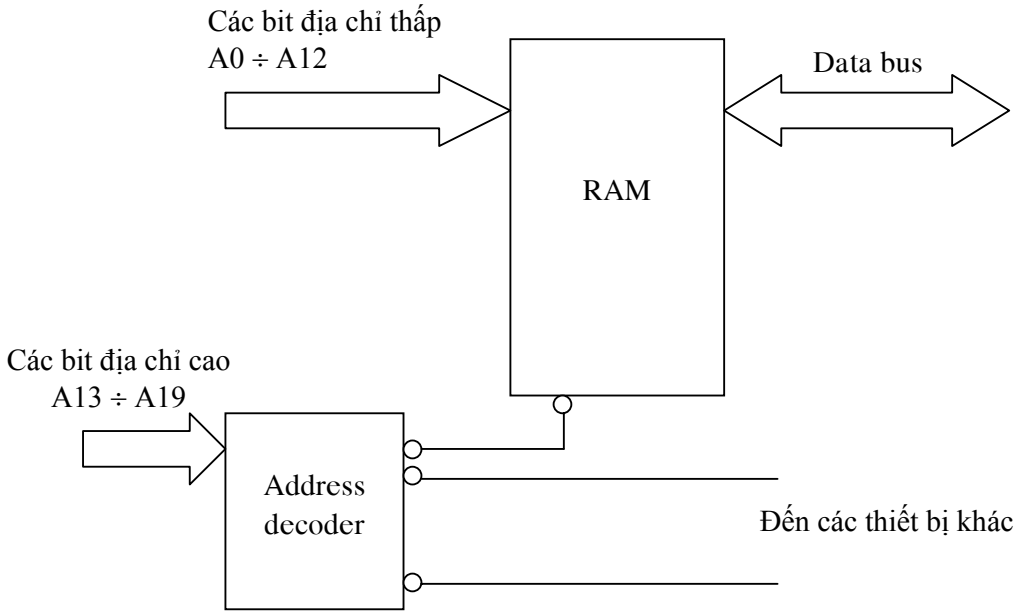
Hình 1.11 – Định thì bus cơ bản

4.3. Giao tiếp với bộ nhớ

4.3.1. Giao tiếp bus cơ bản

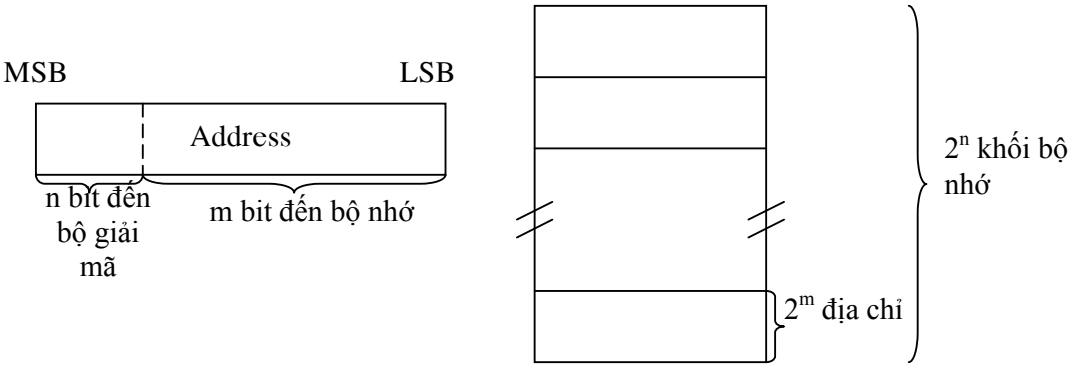
- Các bit địa chỉ thấp (giả sử 13 đường A0 ÷ A12) nối trực tiếp đến chip bộ nhớ (giả sử RAM có dung lượng 8K × 8)

- Các bit địa chỉ cao (giả sử A13 ÷ A19) nối với bộ giải mã địa chỉ (address decoder) tạo tín hiệu cho phép chip bộ nhớ. Do đó, khi thiết kế ta phải xác định mỗi chip bộ nhớ thuộc vùng địa chỉ nào. Tập hợp các vùng này theo bảng gọi là bảng bộ nhớ (memory map).



Hình 1.12 – Giao tiếp bus cơ bản

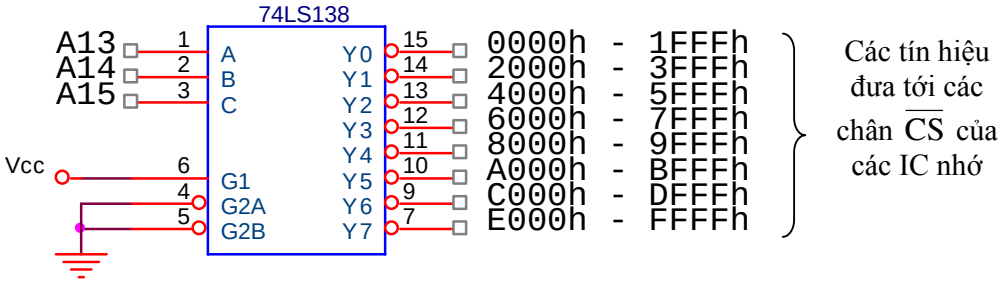
Quan hệ giữa giải mã địa chỉ và bảng bộ nhớ:



Hình 1.13 – Bảng bộ nhớ

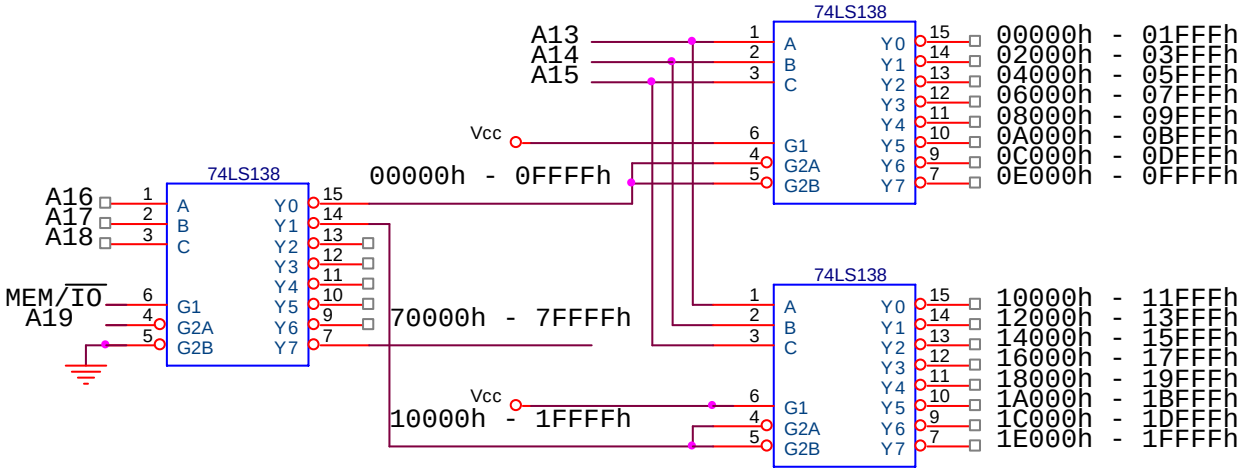
4.3.2. Giải mã địa chỉ

4.3.2.1. Dùng 74LS138



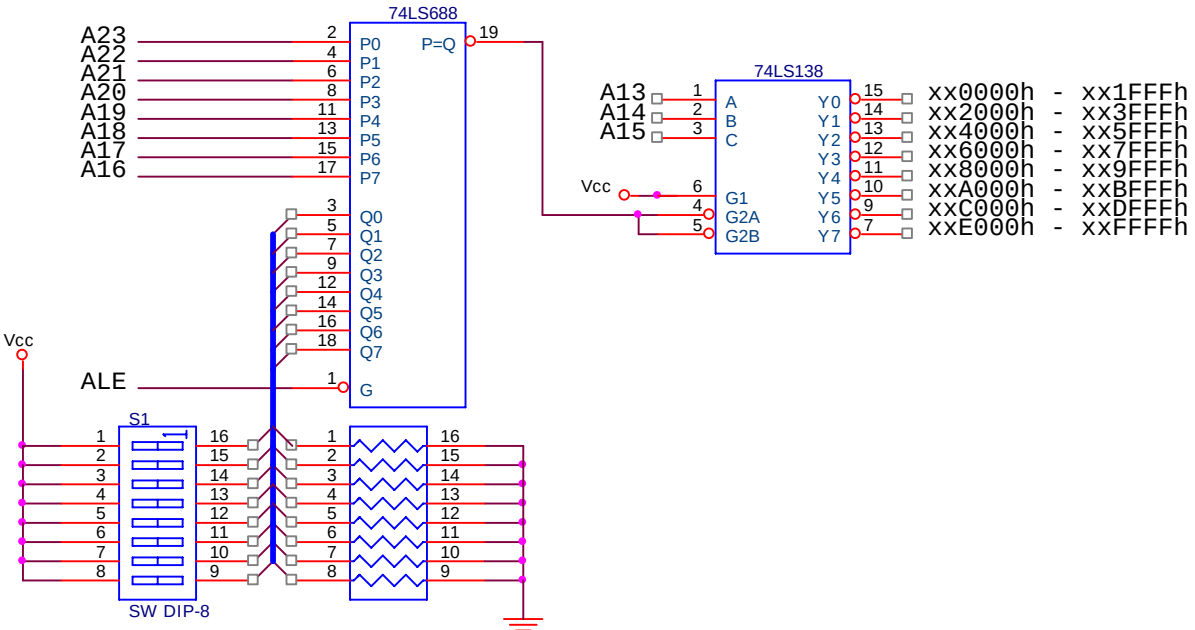
Hình 1.14 – Giải mã địa chỉ dùng 74LS138

4.3.2.2. Dùng nhiều 74LS138



Hình 1.15 – 74LS138 mắc cascaded (liên tầng)

4.3.2.3. Dùng bộ so sánh



Hình 1.16 – Giải mã dùng bộ so sánh

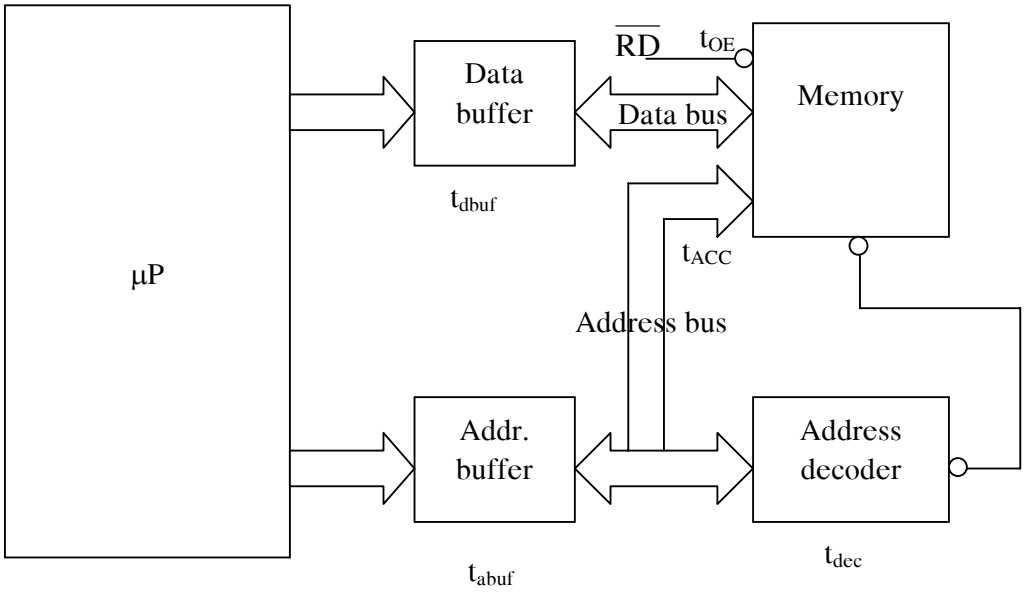
4.3.3. Định thì bộ nhớ

❖ **Thời gian truy xuất (access time):**

- Với chu kỳ đọc: thời gian truy xuất là thời gian tính từ lúc địa chỉ mới xuất hiện ở bộ nhớ cho đến khi có dữ liệu đúng ở ngõ ra của bộ nhớ.
- Với chu kỳ ghi: thời gian truy xuất là thời gian tính từ lúc địa chỉ mới xuất hiện ở bộ nhớ cho đến khi dữ liệu đã đưa vào bộ nhớ.

❖ **Thời gian chu kỳ (cycle time):** là thời gian từ lúc bắt đầu chu kỳ bộ nhớ đến khi bắt đầu chu kỳ kế tiếp.

Ngoài ra, µP có thể sử dụng thêm một số trạng thái chờ khi đọc bộ nhớ.



Hình 1.17 – Các đường trì hoãn trong giao tiếp µP với bộ nhớ

t_{dbuf}: thời gian trì hoãn ở bộ đệm dữ liệu (data buffer)
t_{abuf}: thời gian trì hoãn ở bộ đệm địa chỉ (address buffer)
t_{OE}: thời gian đáp ứng của bộ nhớ với tín hiệu cho phép ngõ ra (ouput enable)
t_{CS}: thời gian bộ nhớ truy xuất từ Chip Select
t_{ACC}: thời gian bộ nhớ truy xuất từ địa chỉ, thông thường tACC = tcs
t_{dec}: thời gian trì hoãn ở bộ giải mã (decoder)

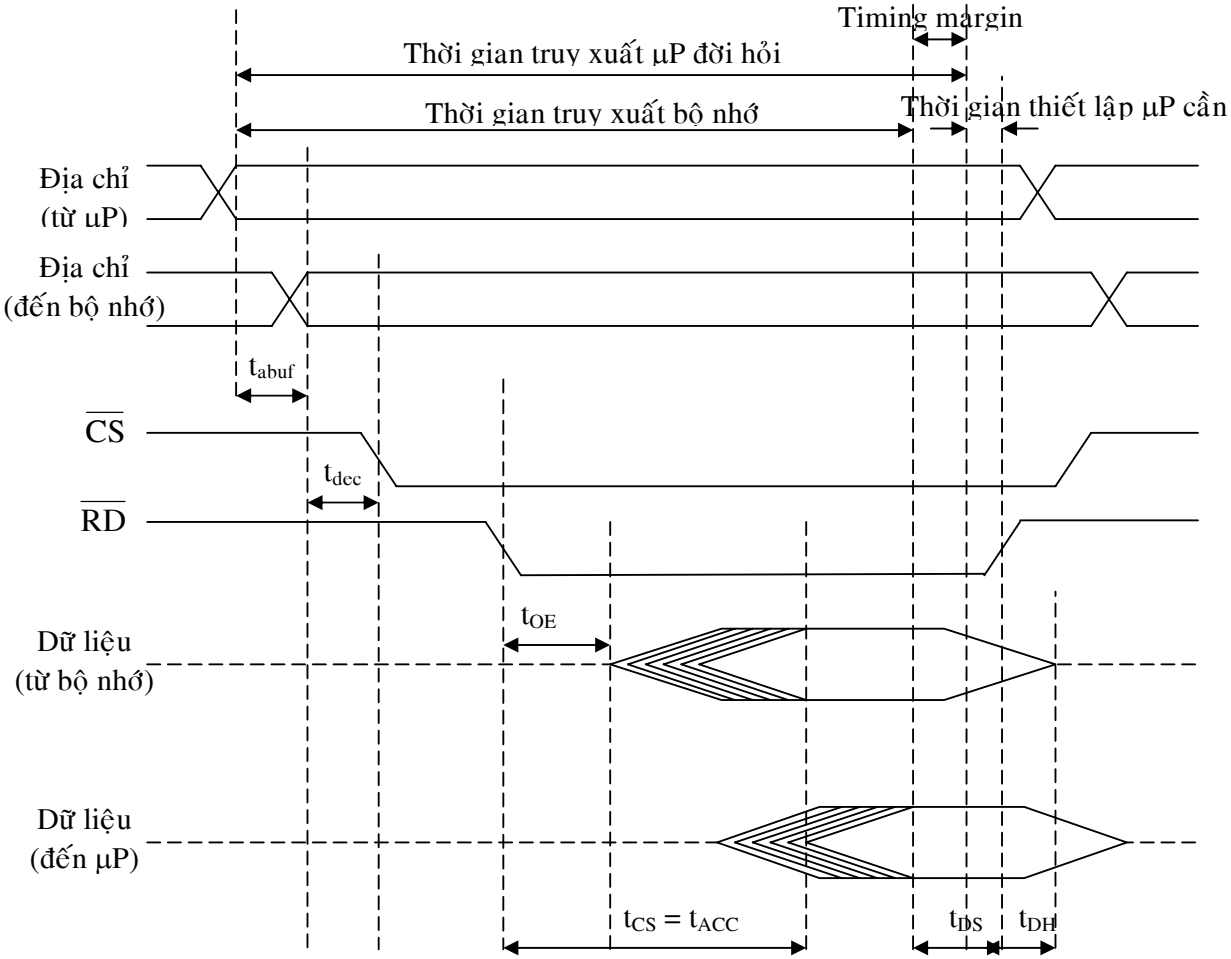
❖ **Định thì đọc bộ nhớ:**

Thời gian truy xuất tổng cộng của hệ thống bộ nhớ chính là tổng thời gian trì hoãn trong các bộ đệm và thời gian truy xuất (access time) bộ nhớ.

Hiệu giữa thời gian truy xuất cần thiết bởi µP với thời gian truy xuất thật sự của bộ nhớ gọi là biên định thì (timing margin).

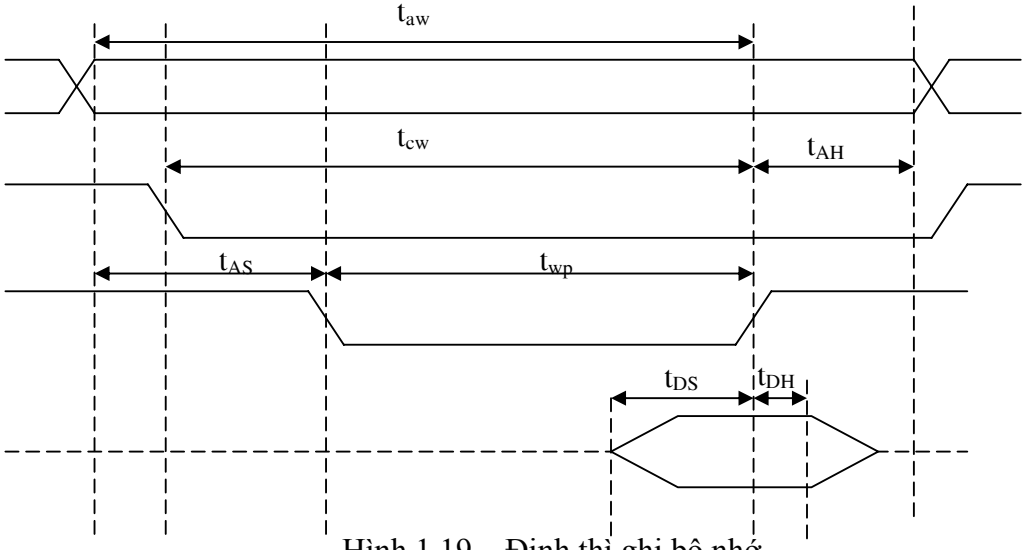
t_{DS} (Data Setup): thời gian thiết lập dữ liệu cung cấp bởi hệ thống bộ nhớ

t_{DH} (Data Hold): thời gian giữ dữ liệu cung cấp bởi hệ thống bộ nhớ



Hình 1.18 – Định thì đọc bộ nhớ

❖ **Định thì ghi bộ nhớ:**



Hình 1.19 – Định thì ghi bộ nhớ

t_{aw}: thời gian truy xuất ghi (access write)
t_{wp}: độ rộng xung ghi tối thiểu (write pulse)
t_{AS}: thời gian địa chỉ hợp lệ trước khi $\overline{WR} = 0$

Thông thường, ta không quan tâm đến địa chỉ cho đến khi xác nhận $\overline{CS}$ nên thường $t_{cw} = t_{aw}$.

5. µP 8086/8088

5.1. Giới thiệu

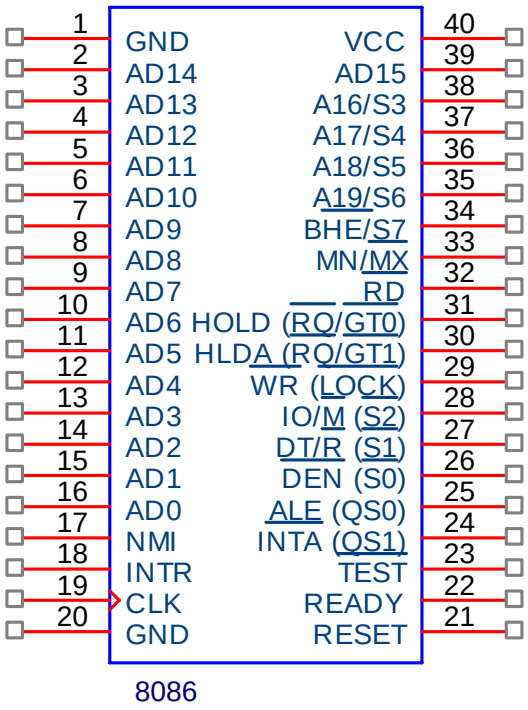
Tất cả các máy vi tính IBM họ PC hoặc các máy vi tính tương thích IBM đều sử dụng µP Intel họ iAPX. Bảng 2.1 liệt kê các đặc tính cơ bản của một số µP của Intel trong đó 80486 chứa một bộ điều khiển cache tích hợp và 8 KB RAM tĩnh, Pentium chứa cache 16 KB RAM tĩnh.

Bảng 1.4: Kiến trúc các µP của Intel 8 bit, 16 bit và 32 bit

	Tốc độ	Bus	Số transistor	Dung lượng bộ nhớ tối đa	Bộ nhớ ảo
4004	108 KHz	4 bits	2,300 (10 microns)	640 bytes	
8008	108 KHz	8 bits	3,500	16 KBytes	
8080	2 MHz	8 bits	6,000 (6 microns)	64 KBytes	
8086	5 MHz 8 MHz 10 MHz	16 bits	29,000 (3 microns)	1 Megabyte	
8088	5 MHz 8 MHz	8 bits	29,000 (3 microns)		
80286	8 MHz 10 MHz 12 MHz	16 bits	134,000 (1.5 microns)	16 Megabytes	1 gigabyte
Intel386(TM)DX Microprocessor	16 MHz 20 MHz 25 MHz 33 MHz	32 bits	275,000 (1 micron)	4 gigabytes	64 terabytes
Intel386(TM)SX Microprocessor	16 MHz 20 MHz	16 bits	275,000 (1 micron)	4 gigabytes	64 terabytes
Intel486(TM)DX Microprocessor	25 MHz 33 MHz 50 MHz	32 bits	1,200,000 (1 micron, .8 micron with 50 MHz)	4 gigabytes	64 terabytes
Intel486(TM)SX Microprocessor	16 MHz 20 MHz 25 MHz 33 MHz	32 bits	1,185,000 (.8 micron)	4 gigabytes	64 terabytes
Pentium® Processor	60MHz	32	3.1 million	4 gigabytes	64

	66MHz 75MHz 90MHz 100MHz 120MHz 133MHz 150MHz 166MHz	bits	(.8 micron)		terabytes
Pentium® Pro Processor	150MHz 180MHz 200MHz	32 bits	5.5 million (.32 micron)	4 gigabytes	64 terabytes

5.2. Mô tả chân



Hình 1.20 – Sơ đồ chân của 8086

8086 có bus địa chỉ 20 bit, bus dữ liệu 16 bit, 3 chân nguồn và 17 chân dùng cho các chức năng điều khiển. Tuy nhiên, ta có thể dùng kỹ thuật ghép kênh thời gian (time multiplexing) để cho phép một chân có nhiều chức năng nên các chân sẽ được phân ra:

- 16 chân dữ liệu và địa chỉ (AD0 ÷ AD15): các chân này sẽ là các đường địa chỉ trong trạng thái T1 và dữ liệu trong các trạng thái T2 – T4.
- 4 chân địa chỉ và trạng thái
- 3 chân nguồn
- 17 chân định thì và điều khiển

8086 có thể hoạt động ở chế độ tối thiểu (minimum mode) hay chế độ tối đa (maximum mode). Chế độ tối thiểu chỉ dùng cho các hệ thống μP đơn giản còn chế độ tối đa dùng cho các hệ thống phúc tạp hơn giao tiếp với các bộ nhớ và I/O riêng.

❖ **Các tín hiệu chung cho cả hai chế độ tối đa và tối thiểu:**

Bảng 1.5:

Chân	Chức năng	Loại
AD15 ÷ AD0	Bus dữ liệu / địa chỉ	2 chiều, 3 trạng thái
A19/S6 ÷ A16/S3	Địa chỉ / trạng thái	Ngõ ra 3 trạng thái
$\overline{MX}$	Điều khiển chế độ	Ngõ vào
$\overline{RD}$	Điều khiển đọc	Ngõ ra 3 trạng thái
$\overline{TEST}$	Chờ kiểm tra điều khiển	Ngõ vào
READY	Chờ trạng thái điều khiển	Ngõ vào
RESET	Reset hệ thống	Ngõ vào
NMI	Yêu cầu ngắt không thể che	Ngõ vào
INTR	Yêu cầu ngắt	Ngõ vào
CLK	Xung nhịp hệ thống	Ngõ vào
VCC	+5V	Ngõ vào
GND	GND	Ngõ vào

❖ **Các tín hiệu chỉ dùng trong chế độ tối thiểu:**

Bảng 1.6:

Chân	Chức năng	Loại
HOLD	Yêu cầu giữ	Ngõ vào
HLDA	Ghi nhận giữ	Ngõ vào
$\overline{WR}$	Điều khiển ghi	Ngõ ra 3 trạng thái
IO/$\overline{M}$	Điều khiển I/O và bộ nhớ	Ngõ ra 3 trạng thái
DT/$\overline{R}$	Truyền / nhận dữ liệu	Ngõ ra 3 trạng thái
$\overline{DEN}$	Cho phép dữ liệu	Ngõ ra 3 trạng thái
$\overline{BHE}$/S7	Đường trạng thái	Ngõ ra 3 trạng thái
ALE	Cho phép chốt địa chỉ	Ngõ ra
$\overline{INTA}$	Ghi nhận ngắt	Ngõ ra

❖ **Các tín hiệu chỉ dùng trong chế độ tối đa:**

Bảng 1.7:

Chân	Chức năng	Loại
$\overline{RQ}$ / $\overline{GT1,0}$	Yêu cầu / cấp bus	2 chiều
$\overline{LOCK}$	Điều khiển khóa ưu tiên bus	Ngõ ra 3 trạng thái
$\overline{S2} \div \overline{S0}$	Trạng thái chu kỳ bus	Ngõ ra 3 trạng thái
QS1, QS2	Trạng thái hàng lệnh	Ngõ ra

❖ **Trạng thái bus:**

Bảng 1.8:

Ngõ vào trạng thái			Chu kỳ CPU
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	
0	0	0	Ghi nhận ngắt
0	0	1	Đọc I/O port
0	1	0	Ghi I/O port
0	1	1	Ngừng
1	0	0	Nhận lệnh
1	0	1	Đọc bộ nhớ
1	1	0	Ghi bộ nhớ
1	1	1	Thụ động

❖ **Trạng thái hàng lệnh:**

Bảng 1.9:

QS1	QS0	Trạng thái hàng lệnh
0	0	Không hoạt động
0	1	Lấy byte đầu tiên của lệnh
1	0	Hàng rỗng
1	1	Lấy byte kế tiếp

❖ **Nguồn cung cấp và xung nhịp (VCC, GND và CLK):**

- 8086 sử dụng nguồn cấp điện +5V và có 2 chân đất.
- Dòng điện cực đại là 340 mA (10 mA cho loại CMOS).
- Xung nhịp dùng dạng xung chữ nhật có chu kỳ với thời gian cạnh lên và xuống nhỏ hơn 10 ns.
- Tiêu hao công suất và tần số xung nhịp cực đại:

❖ **Các chân trạng thái trong chế độ tối đa (S0, S1 và S2 - status):**

Các chân này sử dụng bởi bộ điều khiển bus 8288 để tạo các tín hiệu điều khiển như bảng 2.5.

❖ **Các chân điều khiển bus (HOLD, HLDA, $\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$, $\overline{LOCK}$):**

Chế độ tối thiểu:

- HOLD (giữ): ngõ vào tác động mức cao làm cho μP hở mạch tất cả các bus của nó, tách μP khỏi bộ nhớ của nó và I/O để cho phép thiết bị khác xử lý

bus hệ thống. Quá trình này gọi là truy xuất bộ nhớ trực tiếp (DMA – Direct Memory Access).

- HLDA (Hold acknowledge): ghi nhận yêu cầu DMA đối với bộ điều khiển DMA.

Chế độ tối đa:

- $\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$ (Request / Grant): các chân này dùng cả hai chức năng vào (nhận yêu cầu) và ra (chấp nhận yêu cầu). Khi một thiết bị muốn lấy điều khiển của bus cục bộ, nó sẽ phát yêu cầu bằng cách đưa tín hiệu mức thấp vào chân yêu cầu. Sau khi nhận yêu cầu, 8086 sẽ ở trạng thái HOLD và gởi tín hiệu chấp nhận ra chân này. Ở đây, chân $\overline{RQ}/\overline{GT0}$ có độ ưu tiên cao hơn chân $\overline{RQ}/\overline{GT1}$.
- $\overline{LOCK}$: báo cho các thiết bị khác biết không thể lấy điều khiển của bus cục bộ.

❖ **Các chân ngắt (NMI, INTR và $\overline{INTA}$):**

INTR và NMI là các yêu cầu ngắt khởi động bằng phần cứng, làm việc chính xác như các ngắt mềm. NMI (Non-Maskable Interrupt) là ngõ vào tác động cạnh lên. NMI là ngắt không thể che được và luôn được phục vụ, thường dùng cho các sự kiện như hư nguồn hay các lỗi bộ nhớ. INTR tác động mức cao và có thể bị che bằng cách xoá cờ IF trong thanh ghi cờ (xem 2.3.4) bằng lệnh CLI.

Khi NMI tích cực, điều khiển sẽ được chuyển đến địa chỉ chứa trong các vị trí 00008h ÷ 0000Bh. Khi INTR tích cực, chu kỳ ghi nhận ngắt (interrupt acknowledge cycle) được thực hiện. Quá trình này giống như chu kỳ đọc bộ nhớ ngoại trừ $\overline{INTA}$ tích cực thay vì $\overline{RD}$. Thiết bị tạo ngắt sẽ đặt một giá trị 8 bit vào data bus và chuyển điều khiển đến vị trí giá trị × 4 đến giá trị × 4 + 3.

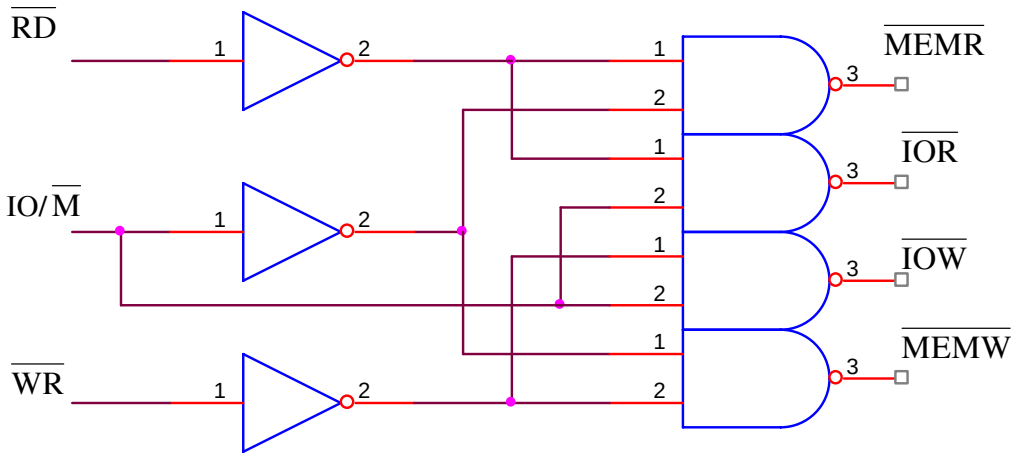
❖ **Chân RESET**: hoạt động khi có xung tác động mức cao, dùng để khởi động lại (P. Sau khi khởi động, (P sẽ đọc lệnh tại địa chỉ FFFF0h. RESET được sử dụng khi hệ thống có sự cố.

❖ **Các chân điều khiển bus (READY, $\overline{RD}$, ALE, $\overline{DEN}$, DT/$\overline{R}$, $\overline{WR}$ và IO/$\overline{M}$):**

Trong các chân điều khiển này, chỉ có hai chân READY và $\overline{RD}$ làm việc ở chế độ tối đa.

- Chân READY: ngõ vào READY được lấy mẫu ở cạnh lên của xung nhịp T2. Nếu chân này ở mức thấp (không sẵn sàng) thì sẽ thêm vào một chu kỳ T3 nữa. Chu trình này sẽ tiếp tục cho đến khi nào chân READY lên mức cao. Ngõ vào này thường được điều khiển bởi thiết bị bộ nhớ chậm, không thể cung cấp dữ liệu kịp thời cho μP.

- Chân IO/$\overline{M}$ (IO/Memory – Xuất nhập /Bộ nhớ): xác định chu kỳ bus hiện hành làm việc với bộ nhớ (mức thấp) hay I/O (mức cao).
- Chân $\overline{RD}$ (Read): tín hiệu tác động mức thấp chỉ chiều truyền dữ liệu từ bộ nhớ hay I/O đến µP. Ta có thể kết hợp với tín hiệu này với IO/$\overline{M}$ để tạo các tín hiệu $\overline{MEMR}$ và $\overline{IOR}$. Nó được xuất ra trong trạng thái T2 và lấy đi trong trạng thái T4. Thiết bị bộ nhớ hay I/O giả sử là đã đặt byte hay word vào các đường dữ liệu khi $\overline{RD}$ trở về mức cao.
- Chân $\overline{WR}$ (Write): tín hiệu này ngược với $\overline{RD}$, nó xác định chiều truyền dữ liệu từ µP đến I/O hay bộ nhớ.

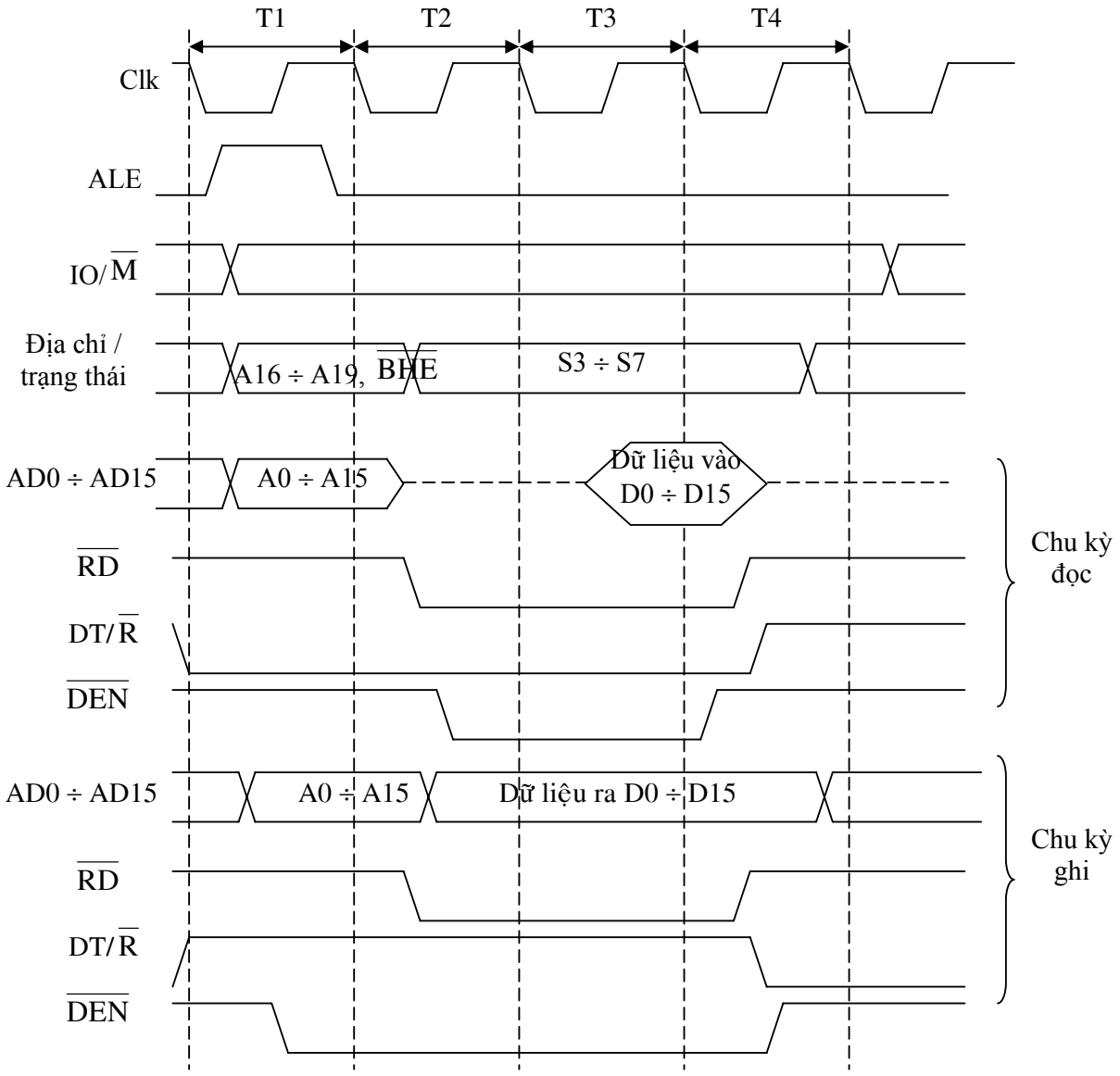


Hình 1.21 – Tạo tín hiệu điều khiển bộ nhớ và I/O

- Chân ALE (Address Latch Enable - cho phép chốt địa chỉ): tín hiệu ra trên chân này có thể dùng để phân kênh các đường địa chỉ, dữ liệu và trạng thái trên AD0 ÷ AD15, A16/S3 ÷ A19/S6 và $\overline{BHE}$/S7. Mọi chu kỳ bắt đầu với xung ALE trong trạng thái T1. Địa chỉ 20 bit được bảo đảm sẽ hợp lệ khi ALE chuyển từ mức cao xuống mức thấp.

- Chân $\overline{DEN}$ (Data Enable – cho phép dữ liệu): tín hiệu này được dùng với DT/$\overline{R}$ để cho phép nối các bộ đệm hai chiều vào data bus. Nó ngăn ngừa sự tranh chấp bus bằng cách cấm các bộ đệm dữ liệu cho đến trạng thái T2 khi các đường dữ liệu / địa chỉ không còn lưu trữ địa chỉ của bộ nhớ hay I/O.

- Chân DT/$\overline{R}$ (Data transmit/receive – truyền/nhận dữ liệu): dùng để điều khiển chiều của luồng dữ liệu qua các bộ đệm (nếu có) vào bus dữ liệu của hệ thống. Khi ở mức thấp, nó chỉ thực hiện tác vụ đọc và khi ở mức cao nó chỉ thực hiện tác vụ ghi.



Hình 1.22 – Các chu kỳ đọc và ghi của 8086

❖ **Các chân trạng thái (AD16/S3 ÷ AD19/S6 và $\overline{\text{BHE}}$/S7):**

5 tín hiệu trạng thái này được xuất ra trong các trạng thái T2 ÷ T4, dùng cho các mục đích kiểm tra. Bit S7 là bit trạng thái dư (không dùng), bit S6 luôn bằng 0, S5 mô tả trạng thái của cờ ngắt IF còn S3, S4 dùng để xác định đoạn đang sử dụng:

Bảng 1.10:

S4	S3	Đoạn
0	0	Thêm
0	1	Stack
1	0	Mã (hay không)
1	1	Dữ liệu

Tín hiệu $\overline{BHE}$/S7 (Bus High Enable) chỉ được xuất trong trạng thái T1. Khi chân này ở mức thấp, nó sẽ chỉ AD8 ÷ AD15 liên quan đến việc truyền dữ liệu. Quá trình này có thể xảy ra đối với các truy xuất bộ nhớ, I/O hay truy xuất 1 byte dữ liệu từ địa chỉ lẻ.

❖ **Bus dữ liệu (AD0 ÷ AD15):**

16 chân này tạo thành bus dữ liệu hai chiều. Các đường này chỉ hợp lệ trong các trạng thái T2 ÷ T4. Trong trạng thái T1, chúng giữ 16 bit thấp của địa chỉ bộ nhớ hoặc I/O.

❖ **Bus địa chỉ (AD0 ÷ AD15 và AD16/S3 ÷ AD19/S6):**

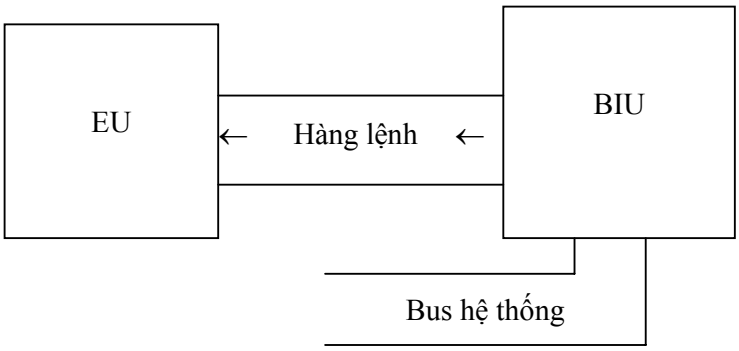
20 chân này tương ứng với bus địa chỉ 20 bit và cho phép μP truy xuất 1 MB vị trí bộ nhớ. Các đường ra này chỉ hợp lệ trong trạng thái T1, chuyển thành các đường dữ liệu và trạng thái trong trạng thái T2 ÷ T4.

❖ **Chọn chế độ $\overline{MX}$:**

Chân này dùng để chọn chế độ hoạt động cho 8086, nếu ở mức cao thì sẽ hoạt động ở chế độ tối thiểu còn ở mức thấp thì sẽ hoạt động ở chế độ tối đa.

5.3. Kiến trúc nội

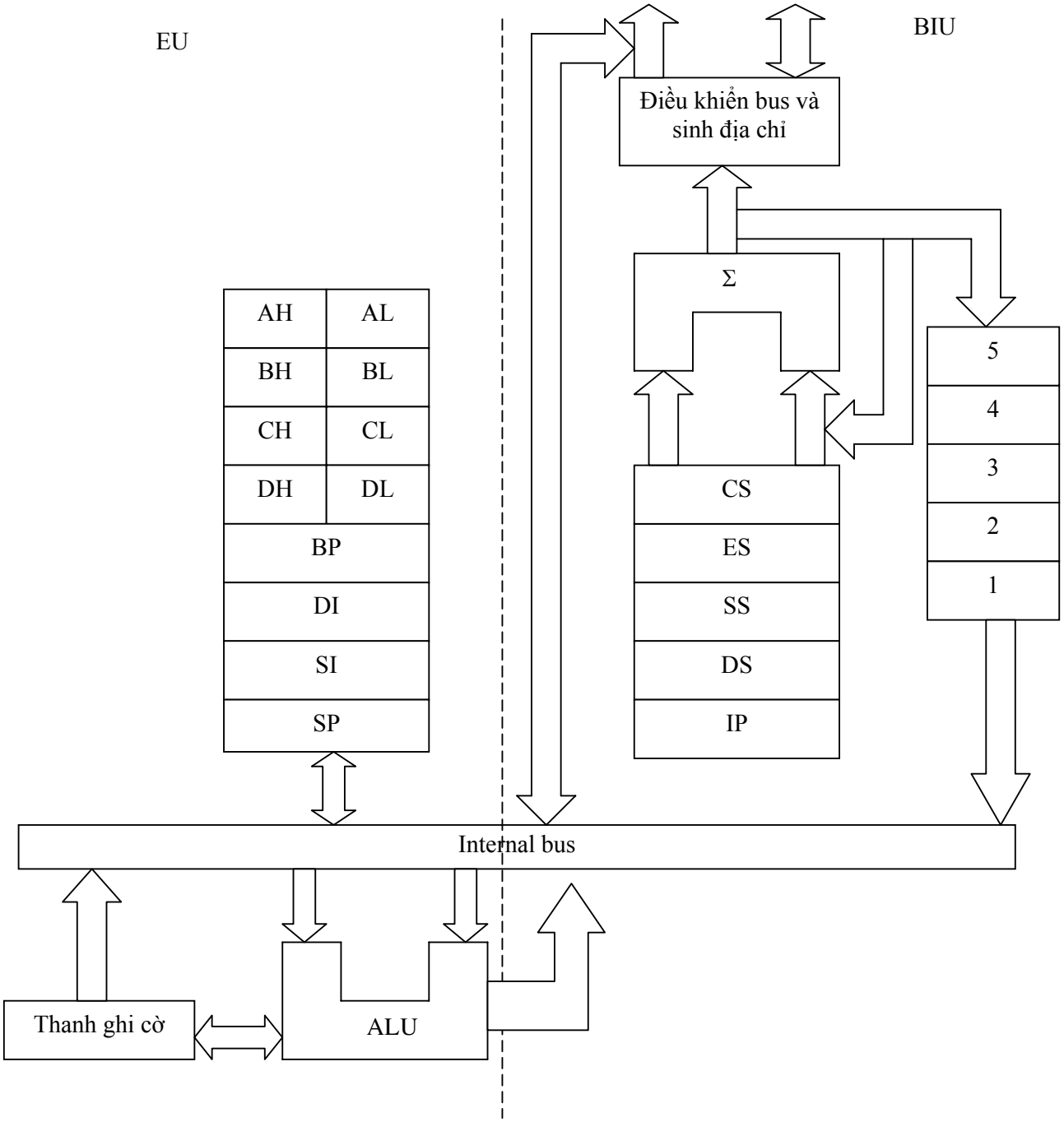
μP có khả năng thực hiện các tác vụ dữ liệu theo tập lệnh bên trong. Một lệnh được ghi nhận bằng mã đã được định nghĩa trước, gọi là mã lệnh (opcode). Trước khi thực thi một lệnh, μP phải nhận được mã lệnh từ bộ nhớ chương trình của nó. Quá trình xử lý này gọi là chu kỳ nhận lệnh (fetch cycle). Một khi các mã được nhận và được giải mã thì mạch bên trong μP có thể tiến hành thực thi (execute) mã lệnh.



Hình 1.23 – Kiến trúc nội của μP 8086

BIU (Bus Interface Unit – đơn vị giao tiếp bus) nhận các mã lệnh từ bộ nhớ và đặt chúng vào hàng chờ lệnh. EU (Execute Unit – đơn vị thực thi) sẽ giải mã và thực hiện các lệnh trong hàng. Chú ý rằng các đơn vị EU và BIU làm việc độc lập với nhau nên BIU có khả năng đang nhận một lệnh mới trong khi EU đang thực thi lệnh trước đó. Khi EU đã thực hiện xong lệnh, nó sẽ lấy mã lệnh kế tiếp trong hàng đợi lệnh (instruction queue).

Kiến trúc nội của μP 8086 ở hình 1.24. Nó có 2 bộ xử lý riêng: BIU và EU. BIU cung cấp các chức năng phần cứng, bao gồm tạo các địa chỉ bộ nhớ và I/O để chuyển dữ liệu giữa EU và bên ngoài μP.



Hình 1.24 – Kiến trúc nội của 8086

EU nhận các mã lệnh chương trình và dữ liệu từ BIU, thực thi các lệnh này và chứa các kết quả trong các thanh ghi. Ngoài ra, dữ liệu cũng có thể chứa trong một vị trí bộ nhớ hay được ghi vào thiết bị xuất. Chú ý rằng EU không có bus hệ thống nên phải thực hiện nhận và xuất tất cả các dữ liệu của nó thông qua BIU.

Sự khác biệt giữa µP 8086 và 8088 là BIU. Trong 8088, đường bus dữ liệu là 8 bit trong khi của 8086 là 16 bit. Ngoài ra hàng lệnh của 8088 dài 4 byte trong khi của 8086 là 6 byte.

Tuy nhiên do EU giữa hai loại µP này giống nhau nên ***các chương trình viết cho 8086 có thể chạy được trên 8088 mà không cần thay đổi gì cả***.

5.4. Các thanh ghi

µP 8086/8088 có tất cả 14 thanh ghi nội. Các thanh ghi này có thể phân loại như sau:

- Thanh ghi dữ liệu (data register)
- Thanh ghi chỉ số và con trỏ (index & pointer register)
- Thanh ghi đoạn (segment register)
- Thanh ghi trạng thái và điều khiển (status & control register)

5.4.1. Các thanh ghi dữ liệu

Các thanh ghi dữ liệu gồm có các thanh ghi 16 bit AX, BX, CX và DX trong đó nửa cao và nửa thấp của mỗi thanh ghi có thể định địa chỉ một cách độc lập. Các nửa thanh ghi này (8 bit) có tên là AH và AL, BH và BL, CH và CL, DH và DL.

Các thanh ghi này được sử dụng trong các phép toán số học và logic hay trong quá trình chuyển dữ liệu.

Bảng 1.11:

Thanh ghi	Sử dụng trong
AX	MUL, IMUL (toán hạng nguồn kích thước word) DIV, IDIV (toán hạng nguồn kích thước word) IN (nhập word) OUT (xuất word) CWD Các phép toán xử lý chuỗi (string)
AL	MUL, IMUL (toán hạng nguồn kích thước byte) DIV, IDIV (toán hạng nguồn kích thước byte) IN (nhập byte) OUT (xuất byte) XLAT AAA, AAD, AAM, AAS (các phép toán ASCII) CBW (đổi sang word) DAA, DAS (số thập phân) Các phép toán xử lý chuỗi (string)
AH	MUL, IMUL (toán hạng nguồn kích thước byte) DIV, IDIV (toán hạng nguồn kích thước byte) CBW (đổi sang word)
BX	XLAT
CX	LOOP, LOOPE, LOOPNE Các phép toán string với tiếp dầu ngữ REP

CL	RCR, RCL, ROR, ROL (quay với số đếm byte) SHR, SAR, SAL (dịch với số đếm byte)
DX	MUL, IMUL (toán hạng nguồn kích thước word) DIV, IDIV (toán hạng nguồn kích thước word)

AX (ACC – Accumulator): thanh ghi tích luỹ
BX (Base): thanh ghi cơ sở
CX (Count): đếm
DX (Data): thanh ghi dữ liệu

5.4.2. Các thanh ghi chỉ số và con trỏ

Bao gồm các thanh ghi 16 bit SP, BP, SI và DI, thường chứa các giá trị offset (độ lệch) cho các phần tử định địa chỉ trong một phân đoạn (segment). Chúng có thể được sử dụng trong các phép toán số học và logic. Hai thanh ghi con trỏ (SP – Stack Pointer và BP – Base Pointer) cho phép truy xuất dễ dàng đến các phần tử đang ở trong ngăn xếp (stack) hiện hành. Các thanh ghi chỉ số (SI – Source Index và DI – Destination Index) được dùng để truy xuất các phần tử trong các đoạn dữ liệu và doạn thêm (extra segment). Thông thường, các thanh ghi con trỏ liên hệ đến đoạn stack hiện hành và các thanh ghi chỉ số liên hệ đến doạn dữ liệu hiện hành. SI và DI dùng trong các phép toán chuỗi.

5.4.3. Các thanh ghi đoạn

Bao gồm các thanh ghi 16 bit CS (Code segment), DS (Data segment), SS (stack segment) và ES (extra segment), dùng để định địa chỉ vùng nhớ 1 MB bằng cách chia thành 16 đoạn 64 KB.

Tất cả các lệnh phải ở trong đoạn mã hiện hành, được định địa chỉ thông qua thanh ghi CS. Offset (độ lệch) của mã được xác định bằng thanh ghi IP. Dữ liệu chương trình thường được đặt ở đoạn dữ liệu, định vị thông qua thanh ghi DS. Stack định vị thông qua thanh ghi SS. Thanh ghi đoạn thêm có thể sử dụng để định địa chỉ các toán hạng, dữ liệu, bộ nhớ và các phần tử khác ngoài đoạn dữ liệu và stack hiện hành.

5.4.4. Các thanh ghi điều khiển và trạng thái

Thanh ghi con trỏ lệnh IP (Instruction Pointer) giống như bộ đếm chương trình (Program Counter). Thanh ghi điều khiển này do BIU quản lý nhằm lưu trữ offset từ bắt đầu đoạn mã đến lệnh thực thi kế tiếp. Ta không thể xử lý trực tiếp trên thanh ghi IP.

Thanh ghi cờ (Flag register) hay từ trạng thái 16 bit chứa 3 bit điều khiển (TF, IF và DF) và 6 bit trạng thái (OF, SF, ZF, AF, PF và CF) còn các bit còn lại mà 8086/8088 không sử dụng thì không thể truy xuất được.

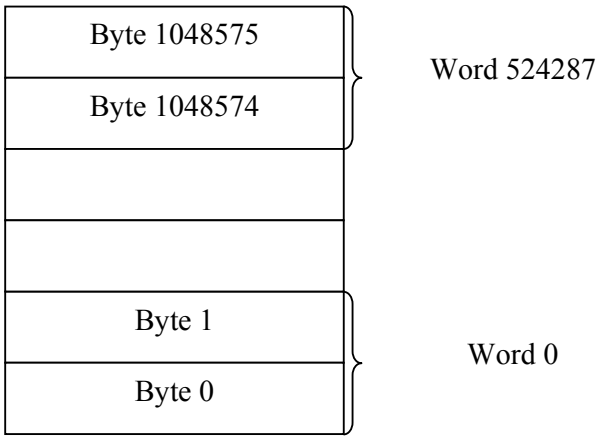
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

- OF (Overflow - tràn): OF = 1 xác định tràn số học, xảy ra khi kết quả vượt ra ngoài phạm vi biểu diễn
- DF (Direction- hướng): xác định hướng chuyển string, DF = 1 khi μP làm việc với string theo thứ tự từ phải sang trái.
- IF (Interrupt - ngắt): cho phép hay cấm các interrupt có mặt nạ
- TF (Trap - bẫy): đặt μP vào chế độ từng bước, dùng cho các chương trình gỡ rối (debugger).
- SF (Sign - dấu): dùng để chỉ các kết quả số học là số dương (SF = 0) hay âm (SF = 1).
- ZF (Zero): = 1 nếu kết quả của phép toán trước là 0.
- AF (Auxiliary – nhớ phụ): dùng trong các số thập phân để chỉ nhớ từ nửa byte thấp hay mượn từ nửa byte cao.
- PF (Parity): PF = 1 nếu kết quả của phép toán là có tổng số bit 1 là chẵn (dùng để kiểm tra lỗi truyền dữ liệu)
- CF (Carry): CF = 1 nếu có nhớ hay mượn từ bit cao nhất của kết quả. Cờ này cũng dùng cho các lệnh quay.

6. Phân đoạn bộ nhớ

Ta biết rằng dù 8086 là μP 16 bit (có bus dữ liệu 16 bit) nhưng vẫn dùng bộ nhớ theo các byte. Điều này cho phép μP làm việc với byte cũng như word, nó rất quan trọng trong giao tiếp với các thiết bị I/O như máy in, thiết bị đầu cuối và modem (chúng được thiết kế để chuyển dữ liệu mã hoá ASCII 7 hay 8 bit). Ngoài ra, nhiều mã lệnh của 8086/8088 có chiều dài 1 byte nên cần phải truy xuất được các byte riêng biệt để có thể xử lý các lệnh này.

8086/8088 có bus địa chỉ 20 bit nên có thể cho phép truy xuất 2^{20} = 1048576 địa chỉ bộ nhớ khác nhau.

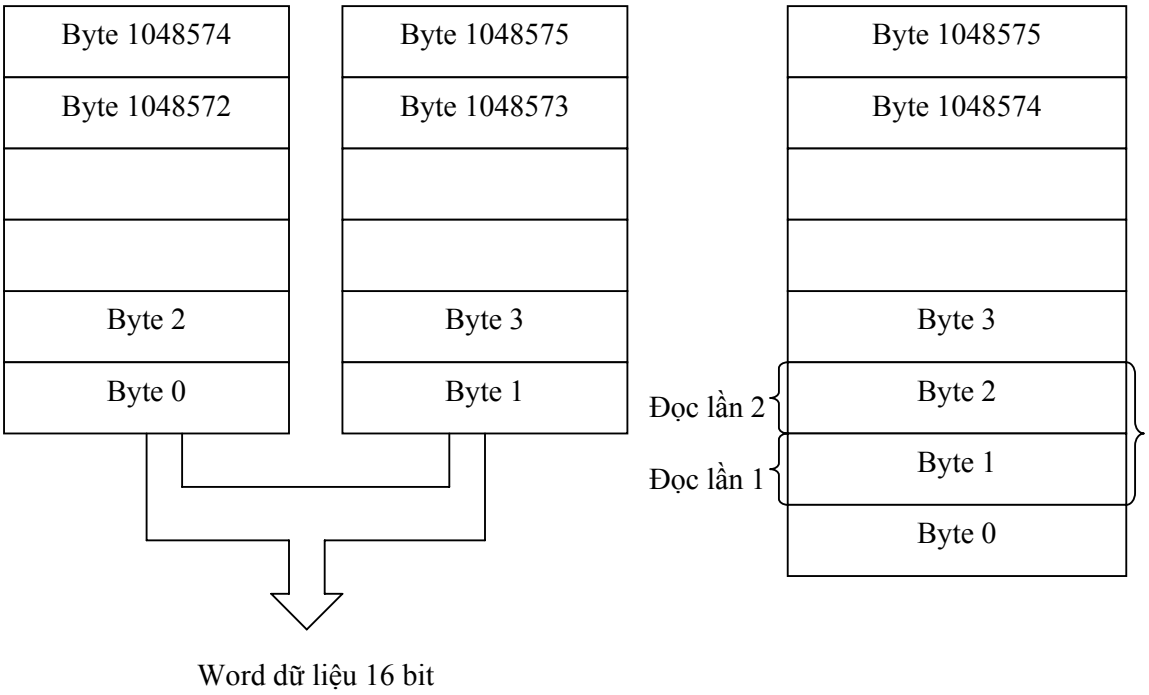


Hình 1.25 – Vùng nhớ của 8086/8088 có 1048576 byte hay 524288 word

Để thực hiện đọc 16 bit từ bộ nhớ, 8086 sẽ thực hiện đọc đồng thời byte có địa chỉ lẻ và byte có địa chỉ chẵn. Do đó, 8086 tổ chức bộ nhớ thành các bank chẵn và lẻ. Theo hình 1.25, ta có thể thấy rằng các word luôn bắt đầu tại địa chỉ chẵn nhưng ta vẫn

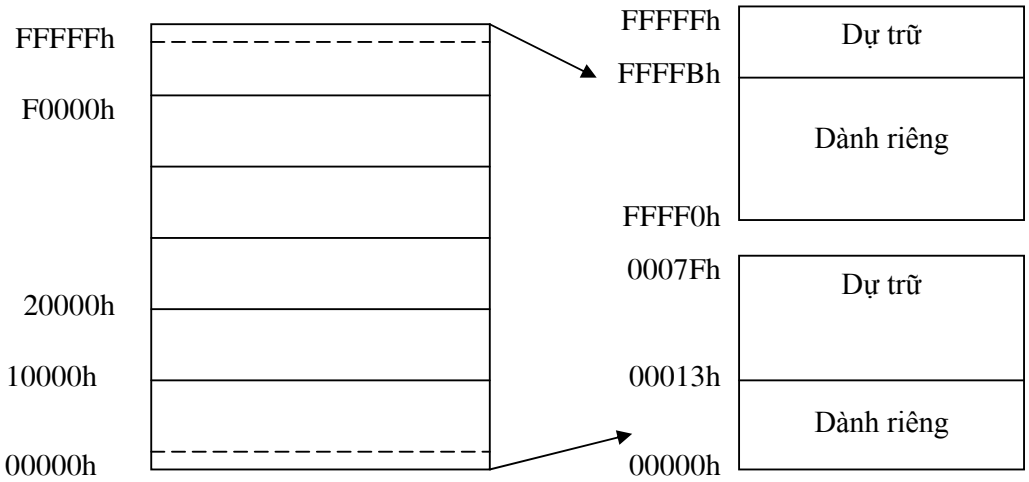
có thể đọc word có địa chỉ lẻ bằng cách thực hiện 2 chu kỳ đọc bộ nhớ: một chu kỳ đọc byte thấp và một chu kỳ đọc byte cao. Điều này sẽ làm chậm tốc độ xử lý.

Đối với 8088 thì do bus dữ liệu 8 bit nên dù word có địa chỉ chẵn hay lẻ, nó cũng cần phải thực hiện 2 chu kỳ đọc hay ghi bộ nhớ và giao tiếp với bộ nhớ như một bank.



Hình 1.26 – Đọc word địa chỉ chẵn và địa chỉ lẻ

Ngoài ra bộ nhớ cũng chia thành 16 khối, mỗi khối có kích thước 64 KB, bắt đầu ở địa chỉ 00000h và kết thúc ở FFFFFh. Địa chỉ bắt đầu mỗi khối sẽ tăng lên 1 ở số hex có ý nghĩa nhiều nhất khi thay đổi từ khối này sang khối kia. Ví dụ như khối 00000h → 10000h → 20000h …



Hình 1.27 – Bảng bộ nhớ cho 8086/8088

❖ Các thanh ghi phân đoạn:

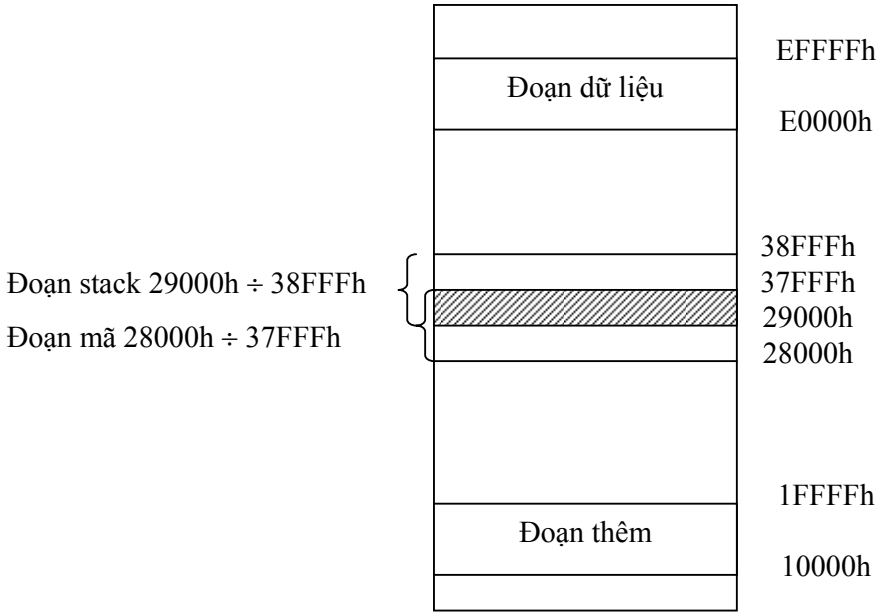
8086/8088 định nghĩa 4 khối bộ nhớ 64KB: đoạn mã (code segment) giữ các mã lệnh chương trình, đoạn ngăn xếp (stack segment) lưu các địa chỉ sẽ trả về từ các chương trình con (subroutine) hay trình phục vụ ngắt (interrupt subroutine), đoạn dữ liệu (data segment) lưu trữ dữ liệu cho chương trình và đoạn thêm (extra segment) thường dùng cho các dữ liệu dùng chung.

Các thanh ghi đoạn (CS, DS, SS và ES) dùng để chỉ vị trí nền của mỗi đoạn. Các thanh ghi này có 16 bit trong khi địa chỉ bộ nhớ là 20 bit nên để xác dịnh vị trí bộ nhớ, ta sẽ thêm 4 bit 0 vào các bit thấp của thanh ghi đoạn. Giả sử như thanh ghi CS chứa giá trị 1111h thì nó sẽ chỉ tới địa chỉ nền là 11110h. Chú ý rằng địa chỉ bắt đầu một đoạn không thể tuỳ ý mà phải bắt đầu tại một địa chỉ chia hết cho 16. Nghĩa là 4 bit thấp phải là 0. Ta cũng chú ý rằng 4 đoạn có thể không tách rời nhau mà chồng lấp lên nhau và ta cũng có thể cho 4 giá trị của các thanh ghi đoạn bằng nhau nghĩa là 4 đoạn này trùng nhau.

VD: Thanh ghi DS có giá trị là 1000h thì địa chỉ nền là 10000h. Địa chỉ kết thúc tìm được bằng cách cộng địa chỉ nền với giá trị FFFFh (64K) → địa chỉ kết thúc là 10000h + FFFFh = 1FFFFh. Như vậy đoạn dữ liệu có địa chỉ từ 10000h = 1FFFFh.

Các vị trí bộ nhớ không được định nghĩa trong các đoạn hiện hành không thể truy xuất được. Muốn truy xuất đến các vị trí đó, ta phải định nghĩa lại một trong các thanh ghi đoạn sau cho đoạn phải chứa vị trí đó. Như vậy, tại một thời điểm bất kỳ ta chỉ có thể truy xuất tối đa 4 × 64 KB = 256 KB bộ nhớ. Nội dung của các thanh ghi đoạn chỉ có thể xác định thông qua phần mềm.

VD: Giả sử các thanh ghi đoạn có các giá trị CS = 2800h, DS = E000h, SS = 2900h và ES = 1000h. Ta có vị trí các đoạn trong bảng bộ nhớ như sau:



Hình 1.28 – Vị trí các phân đoạn theo giá trị các thanh ghi đoạn

❖ **Địa chỉ logic và địa chỉ vật lý:**

Các địa chỉ trong một đoạn thay đổi từ 0000h ÷ FFFFh, tương ứng với chiều dài đoạn là 64 KB. Một địa chỉ trong một đoạn được gọi là ***địa chỉ logic hay offset***. Ví dụ như địa chỉ logic 0010h của đoạn mã trong hình 1.28 sẽ có địa chỉ thật sự là 28000h + 0010h = 28010h. Địa chỉ này gọi là ***địa chỉ vật lý***.

Như vậy, địa chỉ vật lý chính là địa chỉ thật sự xuất hiện ở bus địa chỉ, nó có chiều dài 20 bit còn địa chỉ logic là độ lệch (offset) từ vị trí 0 của một đoạn cho trước.

VD: Giả sử xét các đoạn như hình 1.28. Địa chỉ vật lý tương ứng với địa chỉ logic 1000h trong đoạn stack là:

29000h + 1000h = 2A000h

Địa chỉ vật lý tương ứng với địa chỉ logic 2000h trong đoạn mã là:

28000h + 2000h = 2A000h

Ta thấy rằng có thể địa chỉ vật lý trùng nhau khi địa chỉ logic khác nhau nghĩa là một địa chỉ vật lý có thể có nhiều địa chỉ logic khác nhau.

Để chỉ địa chỉ logic 1000h trong đoạn mã, ta dùng ký hiệu CS:1000h. Tương tự như vậy cho các đoạn khác, nghĩa là địa chỉ logic 1111h trong đoạn dữ liệu sẽ là DS:1111h.

Mọi lệnh tham chiếu bộ nhớ sẽ có một thanh ghi đoạn mặc nhiên. Thanh ghi IP cung cấp địa chỉ offset khi truy xuất đến đoạn mã và BP cho đoạn stack. Ví dụ như IP = 1000h và CS = 2000h thì BIU sẽ truy xuất đến địa chỉ 20000h + 1000h = 21000h và nhận byte tại vị trí này.

Bảng 1.12:

Tham chiếu bộ nhớ	Đoạn mặc nhiên	Đoạn khác	Offset
Nhận lệnh	CS	Không	IP
Tác vụ stack	SS	Không	SP
Dữ liệu tổng quát	DS	CS,ES,SS	Địa chỉ hiệu dụng
Nguồn của string	DS	CS,ES,SS	SI
Đích của string	ES	Không	DI
BX dùng làm con trỏ	DS	CS,ES,SS	Địa chỉ hiệu dụng
BP dùng làm con trỏ	SS	CS,ES,SS	Địa chỉ hiệu dụng

VD: Ta sử dụng lệnh MOV [BP],AL với BP = 2C00h. Ở đây BP dùng làm con trỏ nên dùng đoạn stack. Giả sử các phân đoạn như hình 2.11 thì địa chỉ vật lý sẽ là 29000h + 2C00h = 2BC00h

❖ **Định nghĩa các vị trí bộ nhớ:**

Thông thường ít khi nào ta cần biết đến địa chỉ vật lý của một vị trí bộ nhớ mà ta chỉ quan tâm đến địa chỉ logic của nó mà thôi. Lý do là vì địa chỉ vật lý còn phải phụ thuộc vào nội dung của các thanh ghi đoạn ngay cả khi địa chỉ logic giữ không đổi như đã xét ở trên.

7. Các cách định địa chỉ

Bảng 1.13:

Cách định địa chỉ	Mã đối tượng	Ví dụ			
		Từ gợi nhớ	Đoạn truy xuất	Hoạt động	Mô tả
Tức thời	B80010	MOV AX,1000h	Mã	AH ← 10h AL ← 00h	(1)
Thanh ghi	8BD1	MOV DX,CX	Trong µP	DX ← CX	(2)
Trực tiếp	8A260010	MOV AH,[1000h]	Döõ lieäu	AH ← [1000h]	(3)
Gián tiếp thanh ghi	8B04 FF25 FE4600 FF0F	MOV AX,[SI] JMP [DI] INC BYTE PTR [BP] DEC WORD PTR [BX]	Dữ liệu Dữ liệu Stack Dữ liệu	AL ← [SI]; AH ←[SI+1] IP←[DI+1:DI] [BP]←[BP]+1 [BX+1:BX]← [BX+1:BX]-1	(4)
Có chỉ số	8B4406 FF6506	MOV AX,[SI+6] JMP [DI+6]	Dữ liệu Dữ liệu	AL ← [SI+6]; AH ←[SI+7] IP←[DI+7:DI+6]	(5)
Có nền	8B4602 FF6702	MOV AX,[BP+2] JMP [BP+2]	Stack Dữ liệu	AL←[BP+2]; AH ←[BP+3] IP←[BX+3:BX+6]	(6)
Có nền và có chỉ số	8B00 FF21 FE02 FF0B	MOV AX,[BX+SI] JMP [BX+DI] INC BYTE PTR [BP+SI] DEC WORD PTR [BP+DI]	Dữ liệu Dữ liệu Stack Stack	AL←[BX+SI];AH←[BX+SI+1] IP←[BX+DI+1:BX+DI] [BP+SI]←[BP+SI]+1 [BP+DI+1:BP+DI]← [BP+DI+1:BP+DI]-1	(7)
Có nền và có chỉ số với độ dời	8B4005 FF6105 FE4205 FF4B05	MOV AX,[BX+SI+5] JMP [BX+DI+5] INC BYTE PTR [BP+SI+5] DEC WORD PTR [BP+DI+5]	Dữ liệu Dữ liệu Stack Stack	AL←[BX+SI+5] AH←[BX+SI+1] IP←[BX+DI+6:BX+DI+5] [BP+SI+5]←[BP+SI+5]+1 [BP+DI+6:BP+DI+5]← [BP+DI+6:BP+DI+5]-1	(8)
String	A4	MOVSB	Thêm, dữ liệu	[ES:DI] ← [DS:DI] Nếu DF = 0 thì SI ← SI + 1; DI ← DI + 1 Nếu DF = 1 thì SI ← SI - 1; DI ← DI - 1	(9)

- BYTE PTR và WORD PTR tránh lầm giữa truy xuất byte và word.
- Độ dời được cộng vào thanh ghi con trỏ hay nền là số nhị phân dạng bù 2.
- (1): nguồn dữ liệu trong lệnh
- (2): đích và nguồn là các thanh ghi của µP
- (3): địa chỉ bộ nhớ cung cấp trong lệnh
- (4): địa chỉ bộ nhớ cung cấp trong thanh ghi con trỏ hay chỉ số
- (5): địa chỉ bộ nhớ là tổng của thanh ghi chỉ số cộng với độ dời trong lệnh
- (6): địa chỉ bộ nhớ là tổng của thanh ghi BX hay BP cộng với độ dời trong lệnh
- (7): địa chỉ bộ nhớ là tổng của thanh ghi chỉ số và thanh ghi nền

- (8): địa chỉ bộ nhớ là tổng của thanh ghi chỉ số, thanh ghi nền và độ dời trong lệnh
- (9): địa chỉ nguồn bộ nhớ là thanh ghi SI trong đoạn dữ liệu và địa chỉ đích bộ nhớ là thanh ghi DI trong đoạn thêm

7.1. Định địa chỉ tức thời

Các lệnh dùng cách định địa chỉ tức thời lấy dữ liệu trong lệnh làm một phần của lệnh. Trong cách này, dữ liệu sẽ được chứa trong đoạn mã thay vì trong đoạn dữ liệu. Dữ liệu cho lệnh MOV AX,1000h được cung cấp tức thời sau mã lệnh B8. Chú ý rằng trong mã đối tượng byte dữ liệu cao đi sau byte dữ liệu thấp.

Cách định địa chỉ tức thời thường dùng để nạp một thanh ghi hay vị trí bộ nhớ với các dữ liệu ban đầu. Sau đó, các lệnh kế tiếp sẽ làm việc với các dữ liệu này. Tuy nhiên, cách định địa chỉ này không sử dụng được cho các thanh ghi đoạn.

7.2. Định địa chỉ thanh ghi

Một số lệnh chỉ làm công việc chuyển dữ liệu giữa các thanh ghi của μP. Ví dụ như MOV DX,CX sẽ chuyển dữ liệu từ thanh ghi CX vào thanh ghi DX. Ở đây ta không cần thực hiện tham chiếu bộ nhớ.

Ta có thể kết hợp cách định địa chỉ tức thời và định địa chỉ thanh ghi để nạp dữ liệu cho các thanh ghi đoạn.

VD:

```
MOV  AX, 1000h
MOV CS,AX
```

Sau khi thực hiện 2 lệnh này, giá trị của thanh ghi CS sẽ là 1000h.

7.3. Định địa chỉ trực tiếp

Ngoài 2 cách định địa chỉ trên, tất cả các cách định địa chỉ còn lại cho trong bảng 2.6 đều cần phải truy xuất đến bộ nhớ với ít nhất một toán hạng. Trong cách định địa chỉ trực tiếp, địa chỉ bộ nhớ được cung cấp trực tiếp như là một phần của lệnh. Ví dụ như lệnh MOV AH,[1000h] sẽ đưa nội dung chứa trong ô nhớ DS:1000h vào thanh ghi AH hay lệnh MOV [2000h],AX sẽ đưa nội dung chứa trong AX vào 2 ô nhớ liên tiếp DS:2000h và DS:2001h

7.4. Định địa chỉ truy xuất bộ nhớ gián tiếp

Các cách định địa chỉ trực tiếp sẽ thuận lợi cho các truy xuất bộ nhớ không thường xuyên. Tuy nhiên, nếu một ô nhớ cần phải truy xuất nhiều lần trong một chương trình thì quá trình nhận địa chỉ (2 byte) sẽ phải thực hiện nhiều lần. Điều này sẽ không hiệu quả.

Để giải quyết vấn đề này, ta thực hiện lưu trữ địa chỉ của ô nhớ cần truy xuất trong một thanh ghi con trỏ, chỉ số hay thanh ghi cơ sở (BX, BP, SI hay DI). Ngoài ra,

ta có thể sử dụng độ dời bù 2 bằng cách cộng vào các thanh ghi để dời đi so với vị trí được các thanh ghi chỉ đến.

Bảng 2.13:

Cách định địa chỉ	Địa chỉ hiệu dụng (EA – Effective Address)		
	Độ dời	**Thanh ghi nền**	**Thanh ghi chỉ số**
Gián tiếp thanh ghi	Không	BX hay BP	Không
	Không	Không	SI hay DI
Có chỉ số	-128 ÷ 127	Không	SI hay DI
Có nền	-128 ÷ 127	BX hay BP	Không
Có nền và chỉ số	Không	BX hay BP	SI hay DI
Có nền và chỉ số với độ dời	-128 ÷ 127	BX hay BP	SI hay DI

Như vậy, một độ dời có thể được cộng vào thanh ghi nền và kết quả này được cộng tiếp vào thanh ghi chỉ số. Địa chỉ thu được gọi là địa chỉ hiệu dụng EA.

Ngoài ra ta cũng có thể viết cách định địa chỉ gián tiếp như sau:

```
MOV AX,table[SI]
```

Trong đó ***table*** là nhãn gán cho một vị trí ô nhớ nào đó. Lệnh này sẽ truy xuất phần tử thứ SI trong dãy ***table*** (giả sử SI = 2 thì sẽ truy xuất phần tử thứ 2). Ta cũng có thể viết lệnh trên như sau:

```
MOV AX,[table + SI]
```

Chú ý rằng các đoạn mặc định cho các cách định địa chỉ gián tiếp là đoạn stack khi dùng BP, là đoạn dữ liệu khi dùng BX, SI hay DI.

VD: Lệnh:

```
MOV  AH,10h          thực hiện định địa chỉ tức thời
MOV  AX,[BP + 10]    thực hiện định địa chỉ có nền
MOV  AH,[BP + SI]    thực hiện định địa chỉ có nền và có chỉ số
```

7.5. Định địa chỉ chuỗi

Chuỗi là một dãy liên tục các byte hay word lưu trữ trong bộ nhớ dưới dạng các ký tự ASCII. 8086/8088 có các lệnh dùng để xử lý chuỗi, các lệnh này sử dụng cặp thanh ghi DS:SI để chỉ nguồn chuỗi ký tự và ES:DI để chỉ đích chuỗi. Lệnh MOVSB sẽ chuyển byte dữ liệu nguồn đến vị trí đích trong đó SI và DI sẽ tăng hay giảm tuỳ theo giá trị của DF (xem 2.3.4 và bảng 2.13)

7.6. Thay đổi thanh ghi đoạn mặc định

Như đã nói ở phần trên, khi sử dụng các lệnh định địa chỉ thanh ghi, ta chỉ cần dùng các thanh ghi để xác định độ lệch còn các thanh ghi đoạn thì được hiểu mặc định. Ví dụ như ta dùng lệnh MOV AH,[BP] thì sẽ đưa dữ liệu tại ô nhớ SS:BP vào thanh ghi AH. Trong trường hợp không muốn dùng thanh ghi đoạn mặc định, ta có thể thay đổi bằng cách thêm tên thanh ghi đoạn vào để loại bỏ thanh ghi đoạn mặc định. Ví dụ lệnh MOV AH,CS:[BP] sẽ đưa dữ liệu tại CS:[BP] vào AH.

BÀI TẬP CHƯƠNG 1

1. Cần bao nhiêu byte để tạo thành một word 32 bit?

2. Giả sử µP có tất cả 16 đường địa chỉ, hỏi nó có thể xử lý tất cả bao nhiêu địa chỉ?

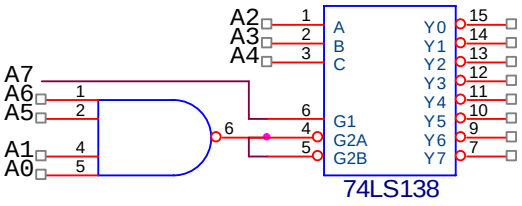
3. Nếu một IC nhớ có dung lượng 1024 × 4 bits thì để tạo 2KB bộ nhớ phải cần bao nhiêu IC?

   Nếu một IC nhớ có dung lượng 256 × 1 bits thì để tạo 1KB bộ nhớ phải cần bao nhiêu IC?

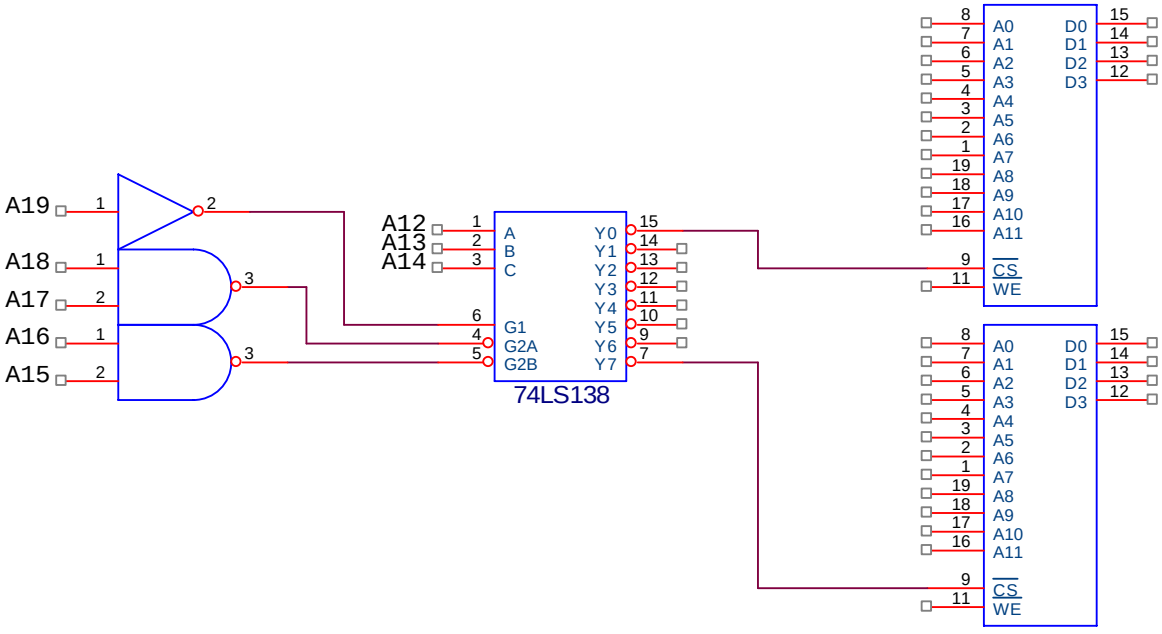
   Nếu một IC nhớ có dung lượng 8K × 8 bits thì cần phải có bao nhiêu đường địa chỉ?

4. Giả sử một IC nhớ có dung lượng 8 KB bắt đầu tại địa chỉ 1000h trong bảng bộ nhớ. Xác định vùng địa chỉ của IC.

5. Ngõ nào của bộ giải mã 74LS138 sẽ tích cực (mức thấp) nếu dữ liệu ngõ vào từ A7 ÷ A0 = 11110111? Nếu muốn ngõ ra Y4 tích cực thì dữ liệu ngõ vào phải là bao nhiêu?



6. Xác định bảng bộ nhớ:



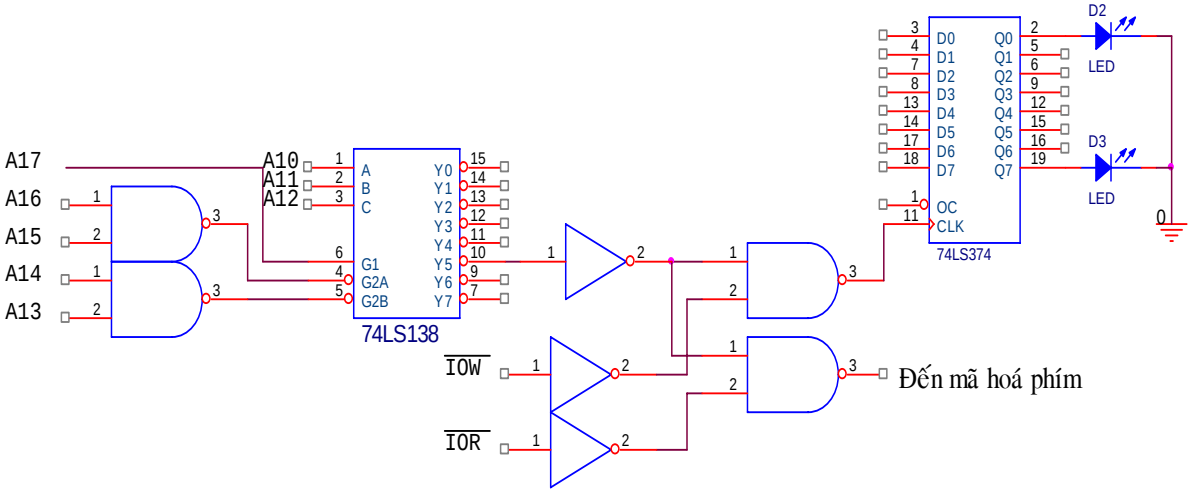
7. Thiết kế mạch giải mã địa chỉ dùng 74LS138 và các cổng logic cho 4 ROM kích thước 8K × 8 bits, 2 RAM 4K × 8 bits và 1 ROM 16K x 8 bits dùng bus địa chỉ 20 bits (A0 ÷ A19).

8. Thiết kế bộ nhớ có dung lượng 16KB từ các bộ nhớ có dung lượng 2K x 4 bits.

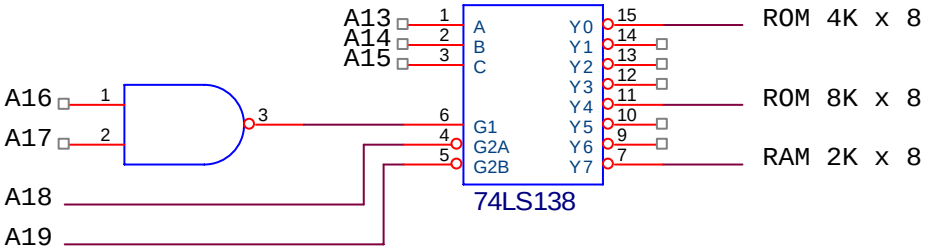
9. Thiết kế mạch giải mã địa chỉ chọn chip theo vùng địa chỉ sau:

   CS0: 8000h – 9FFFh CS2: C000h - DFFFh
   CS1: A000h – BFFFh CS3: E000h – FFFFh

10. Tìm địa chỉ của các I/O port và các giá trị của các chân $\overline{IOR}$, $\overline{IOW}$ khi thực hiện xuất dữ liệu ra Led và đọc bàn phím.



11. Xác định dãy địa chỉ của các thiết bị:



12. Xác định giá trị các cờ SF, ZF, AF và CF sau khi thực hiện các phép toán sau:

a. 1000h - 1234h
b. 1234h + 13FFh
c. ABCDh + 1234h
d. 2345h – 2345h

13. Xác định phương pháp định địa chỉ trong các lệnh sau:

a. MOV AH,DS:[SI]

b. MOV AL,AH

c. MOV DS:[BX+1],AX

d. MOV AL,DS:[BX+SI]

e. MOV CX,DS:[BX+SI+10]

f. MOV DX,20h

g. MOV DS:[10],CL

CHƯƠNG 2: LẬP TRÌNH HỢP NGỮ

1. Các tập tin .EXE và .COM

DOS chỉ có thể thi hành được các tập tin dạng .COM và .EXE. Tập tin .COM thường dùng để xây dựng cho các chương trình nhỏ còn .EXE dùng cho các chương trình lớn.

1.1. Tập tin .COM

- Tập tin .COM chỉ có một đoạn nên kích thước tối đa của một tập tin loại này là 64 KB.
- Tập tin .COM được nạp vào bộ nhớ và thực thi nhanh hơn tập tin .EXE nhưng chỉ áp dụng được cho các chương trình nhỏ.
- Chỉ có thể gọi các chương trình con dạng near.

Khi thực hiện tập tin .COM, DOS định vị bộ nhớ và tạo vùng nhớ dài 256 byte ở vị trí 0000h, vùng này gọi là PSP (Program Segment Prefix), nó sẽ chứa các thông tin cần thiết cho DOS. Sau đó, các mã lệnh trong tập tin sẽ được nạp vào sau PSP ở vị trí 100h và đưa giá trị 0 vào stack. Như vậy, kích thước tối đa thực sự của tập tin .COM là 64 KB – 256 byte PSP – 2 byte stack.

Tất cả các thanh ghi đoạn đều chỉ đến PSP và thanh ghi con trỏ lệnh IP chỉ đến 100h, thanh ghi SP có giá trị 0FFFEh.

1.2. Tập tin .EXE

- Nằm trong nhiều đoạn khác nhau, kích thước thông thường lớn hơn 64 KB.
- Có thể gọi được các chương trình con dạng near hay far.
- Tập tin .EXE chứa một header ở đầu tập tin để chứa các thông tin điều khiển cho tập tin.

2. Khung của một chương trình hợp ngữ

Khung của một chương trình hợp ngữ có dạng như sau:

```
TITLE     Chương trình hợp ngữ
.MODEL    Kiểu kích thước bộ nhớ    ; Khai báo quy mô sử
                                    ; dụng bộ nhớ
.STACK    Kích thước                ; Khai báo dung lượng
                                    ; đoạn stack
.DATA                               ; Khai báo đoạn dữ  liệu
msg  DB   'Hello$'
.CODE                               ; Khai báo đoạn mã
main  PROC
…
CALL      Subname                   ; Gọi chương trình con
…
main  ENDP
```

```
Subname     PROC                ; Định nghĩa chương
                                ; trình con
…
RET
Subname     ENDP
END  main
```

❖ **Quy mô sử dụng bộ nhớ:**

Bảng 2.1:

Loại	Mô tả
Tiny	Mã lệnh và dữ liệu nằm trong một đoạn
Small	Mã lệnh trong một đoạn, dữ liệu trong một đoạn
Medium	Mã lệnh không nằm trong một đoạn, dữ liệu trong một đoạn
Compact	Mã lệnh trong một đoạn, dữ liệu không nằm trong một đoạn
Large	Mã lệnh không nằm trong một đoạn, dữ liệu không nằm trong một đoạn và không có mảng nào lớn hơn 64KB
Huge	Mã lệnh không nằm trong một đoạn, dữ liệu không nằm trong một đoạn và các mảng có thể lớn hơn 64KB

Thông thường, các ứng dụng đơn giản chỉ đòi hỏi mã chương trình không quá 64 KB và dữ liệu cũng không lớn hơn 64 KB nên ta sử dụng ở dạng Small:

```
.MODEL      SMALL
```

❖ **Khai báo kích thước stack:**

Khai báo stack dùng để dành ra một vùng nhớ dùng làm stack (chủ yếu phục vụ cho chương trình con), thông thường ta chọn khoảng 256 byte là đủ để sử dụng (nếu không khai báo thì chương trình dịch tự động cho kích thước stack là 1 KB):

```
.STACK      256
```

❖ **Khai báo đoạn dữ liệu:**

Đoạn dữ liệu dùng để chứa các biến và hằng sử dụng trong chương trình.

❖ **Khai báo đoạn mã:**

Đoạn mã dùng chứa các mã lệnh của chương trình. Đoạn mã bắt đầu bằng một chương trình chính và có thể có các lệnh gọi chương trình con (CALL).

Một chương trình chính hay chương trình con bắt đầu bằng lệnh PROC và kết thúc bằng lệnh ENDP (đây là các lệnh giả của chương trình dịch). Trong chương trình con, ta sử dụng thêm lệnh RET để trả về địa chỉ lệnh trước khi gọi chương trình con.

Chương trình được kết thúc bằng lệnh END trong đó tên chương trình phía sau lệnh END sẽ xác định đó là chương trình chính. Nếu sau lệnh END không chỉ ra

chương trình nào cả thì sẽ lấy chương trình con ở đàu đoạn mã làm chương trình chính.

3. Cú pháp của các lệnh trong chương trình hợp ngữ

Một dòng lệnh trong chương trình hợp ngữ gồm có các trường (field) sau (không nhất thiết phải đầy đủ tất cả các trường):

```
Tên   Lệnh    Toán hạng   Chú thích
A:    MOV     AH,10h      ; Đưa giá trị 10h vào thanh ghi AH
Main  PROC
```

Trường tên chứa nhãn, tên biến hay tên thủ tục. Các tên nhãn có thể chứa tối đa 31 ký tự, không chứa ký tự trắng (space) và không được bắt đầu bằng số (A: hay Main:). Các nhãn được kết thúc bằng dấu '**:**'.

Trường lệnh chứa các lệnh sẽ thực hiện. Các lệnh này có thể là các lệnh thật (MOV) hay các lệnh giả (PROC). Các lệnh thật sẽ được dịch ra mã máy.

Trường toán hạng chứa các toán hạng cần thiết cho lệnh (AH,10h).

Trường chú thích phải được bắt đầu bằng dấu '**;**'. Trường này chỉ dùng cho người lập trình để ghi các lời giải thích cho chương trình. Chương trình dịch sẽ bỏ qua các lệnh nằm phía sau dấu **;**.

3.1. Khai báo dữ liệu

Khi khai báo dữ liệu trong chương trình, nếu sử dụng số nhị phân, ta phải dùng thêm chữ **B** ở cuối, nếu sử dụng số thập lục phân thì phải dùng chữ **H** ở cuối. ***Chú ý rằng đối với số thập lục phân, nếu bắt đầu bằng chữ A..F thì phải thêm vào số 0 ở phía trước.***

Ví dụ:

```
1011b     ; Số nhị phân
1011      ; Số thập phân
1011d     ; Số thập phân
1011h     ; Số thập lục phân
```

3.2. Khai báo biến

Khai báo biến nhằm để chương trình dịch cung cấp một địa chỉ xác định trong bộ nhớ. Ta dùng các lệnh giả sau để định nghĩa các biến ứng với các kiểu dữ liệu khác nhau: DB (define byte), DW (define word) và DD (define double word).

VD:

```
A1    DB    1     ; Định nghĩa biến A1 dài 1 byte (chương
                  ; trình dịch sẽ dùng 1 byte trong bộ nhớ để
                  ; lưu trữ A1), trị ban đầu A1 = 1
A2    DB    ?     ; Biến A2 kiểu byte, không có giá trị gán
```

```
                        ; ban đầu
A3      DB      'A'     ; Biến kiểu ký tự
A4      DW      1       ; Định nghĩa biến A4 dài 2 byte, giá trị ban
                        ; đầu A4 = 1, ta cũng có thể dùng dấu ? để
                        ; xác định biến không cần khởi tạo giá trị ban đầu
A5      DD      1       ; Biến A5 dài 4 byte
A6      DB      1,2,3   ; Định nghĩa biến mảng (array) gồm có 3
                        ; phần tử, mỗi phần tử dài 1 byte (nghĩa là
                        ; sẽ dùng 3 byte lưu trữ) với các giá trị ban
                        ; đầu của các phần tử lần lượt là 1,2,3
A7      DB      10      DUP(0)
                        ; Khai báo biến mảng gồm 10 phần tử, mỗi
                        ; phần tử có chiều dài 1 byte với giá trị gán
                        ; ban đầu là 0
A8      DB      10      DUP(?)
                        ; Khai báo biến mảng gồm 10 phần tử, mỗi
                        ; phần tử có chiều dài 1 byte, không cần
                        ; gán giá trị ban đầu
```

Ngoài ra ta có thể dùng các toán tử DUP lồng vào nhau khi khai báo biến mảng. Giả sử ta cần khai báo mảng A9 có các giá trị gán ban đầu 1,2,3,1,1,3,2,2,1,1,3,2,2. Ta có thể thực hiện như sau:

```
        A9      DB      1,2,3,1,1,3,2,2,1,1,3,2,2
Hay:    A9      DB      1,2,3,2 DUP(1,1,3,2,2)
Hay:    A9      DB      1,2,3,2 DUP(2 DUP(1),3,2 DUP(2))
```

Đối với các biến có nhiều hơn 1 byte, byte thấp sẽ chứa ở ô nhớ có địa chỉ thấp và byte cao sẽ chứa ở ô nhớ có địa chỉ cao.

VD:

```
        A10     DW      1234h
```

Biến A10 giả sử bắt đầu lưu tại địa chỉ 1000h thì ô nhớ 1000h chứa giá trị 34h còn ô nhớ 1001h chứa giá trị 12h.

Đối với biến kiểu chuỗi (string), thực chất là một mảng các ký tự, ta có thể khai báo như sau:

```
        A11     DB      'ABCD'
Hay     A11     DB      65h,66h,67h,68h
```

Sau lệnh khai báo này thì ô nhớ 1000h (giả sử biến A11 lưu trữ tại địa chỉ 1000h) chứa 'A', 1001h chứa 'B', 1002h chứa 'C' và 1003h chứa 'D'.

3.3. Khai báo hằng

Các hằng khai báo trong chương trình hợp ngữ bằng lệnh giả EQU để chương trình dễ hiểu hơn. Hằng có thể ở dạng số, ký tự hay chuỗi.

VD:

```
A12   EQU  10
A13   EQU  'AAA'
```

Sau khi sử dụng khai báo này, nếu ta dùng lệnh:

```
MOV  AH,A12
```

thì AH = 10h

```
A14   DB    'B',A13
```

thì khai báo chuỗi A14 với giá trị gán ban đầu là 'BAAA'.

4. Các toán tử trong hợp ngữ

❖ **Toán tử số học:**

Bảng 2.2:

Toán tử	Cú pháp	Mô tả
+	+bt	Số dương
-	-bt	Số âm
*	bt1*bt2	Nhân
/	bt1/bt2	Chia
mod	bt1 mod bt2	Lấy phần dư
+	bt1 + bt2	Cộng
-	bt1 – bt2	Trừ
shl	bt shl n	Dịch trái n bit
shr	bt shr n	Dịch phải n bit

Trong đó bt, bt1, bt2 là các biểu thức hằng, n là số nguyên.

```
VD:  MOV  AH,(8+1)*7/3          ; AH ← 21
     MOV  AH, 00010001b shr 2   ; AH ← 0000 0100b
     MOV  AH,00010001b shl 2    ; AH ← 0100 0100b
     MOV  AH,100 mod 3          ; AH ← 1
```

❖ **Toán tử logic:**

Bao gồm các toán tử AND, OR, NOT, XOR

```
VD:  MOV  AH,10 OR 4 AND 2      ; AH = 10
     MOV  AH, 0F0h AND 7Fh      ; AH = 70h
```

❖ **Toán tử quan hệ:**

Các toán tử quan hệ so sánh 2 biểu thức, cho giá trị true (-1) nếu điều kiện thoả và false (0) nếu không thoả.

Bảng 2.3:

Toán tử	Cú pháp	Mô tả
EQ	bt1 EQ bt2	Bằng
NE	bt1 NE bt2	Không bằng
LT	bt1 LT bt2	Nhỏ hơn
LE	bt1 LE bt2	Nhỏ hơn hay bằng
GT	bt1 GT bt2	Lớn hơn
GE	bt1 GE bt2	Lớn hơn hay bằng

❖ **Các toán tử cung cấp thông tin:**

➢ **Toán tử SEG:**

SEG bt

Toán tử SEG xác định địa chỉ đoạn của biểu thức ***bt***. ***bt*** có thể là biến, nhãn, hay các toán hạng bộ nhớ.

➢ **Toán tử OFFSET:**

OFFSET bt

Toán tử OFFSET xác định địa chỉ offset của biểu thức ***bt***. ***bt*** có thể là biến, nhãn, hay các toán hạng bộ nhớ.

```
VD:  MOV  AX,SEG A        ; Nạp địa chỉ đoạn và địa chỉ offset
     MOV  DS,AX           ; của biến A vào cặp thanh ghi
     MOV  AX,OFFSET A     ; DS:AX
```

➢ **Toán tử chỉ số [ ]: (index operator)**

Toán tử chỉ số thường dùng với toán hạng trực tiếp và gián tiếp.

➢ **Toán tử (:) (segment override operator)**

Segment:bt

Toán tử **:** quy định cách tính địa chỉ đối với segment được chỉ. ***Segment*** là các thanh ghi đoạn CS, DS, ES, SS.

Chú ý rằng khi sử dụng toán tử : kết hợp với toán tử [ ] thì ***segment:*** phải đặt ngoài toán tử [ ].

VD: Cách viết [CS:BX] là sai, ta phải viết CS:[BX]

➢ **Toán tử TYPE:**

TYPE bt

Trả về giá trị biểu thị dạng của biểu thức ***bt***.

- Nếu ***bt*** là biến thì sẽ trả về 1 nếu biến có kiểu byte, 2 nếu biến có kiểu word, 4 nếu biến có kiểu double word.
- Nếu ***bt*** là nhãn thì trả về 0FFFFh nếu bt là near và 0FFFEh nếu bt là far.
- Nếu ***bt*** là hằng thì trả về 0.

➢ **Toán tử LENGTH:**

LENGTH bt

Trả về số đơn vị bộ nhớ cấp cho biến ***bt***

➢ **Toán tử SIZE:**

SIZE bt

Trả về tổng số các byte cung cấp cho biến ***bt***

```
VD: A    DD   100   DUP(?)
    MOV  AX,LENGTH A     ; AX = 100
    MOV  AX,SIZE A       ; AX = 400
```

❖ **Các toán tử thuộc tính:**

➢ **Toán tử PTR:**

Loai PTR bt

Toán tử này cho phép thay đổi dạng của biểu thức ***bt***.

- Nếu ***bt*** là biến hay toán hạng bộ nhớ thì ***Loai*** là byte, word hay dword.
- Nếu ***bt*** là nhãn thì ***Loai*** là near hay far.

```
VD: A    DW   100   DUP(?)
    B    DD   ?
    MOV  AH,BYTE PTR A      ; Đưa byte đầu tiên trong mảng A
                            ; vào thanh ghi AH
    MOV  AX,WORD PTR B      ; Đưa 2 byte thấp trong biến B
                            ; vào thanh ghi AX
```

➢ **Toán tử HIGH, LOW:**

HIGH bt

LOW bt

Cho giá trị của byte cao và thấp của biểu thức ***bt***, ***bt*** phải là một hằng.

```
VD:  A      EQU  1234h
     MOV  AH,HIGH A          ; AH ← 12h
     MOV  AH,LOW A           ; AH ← 34h
```

5. Các cách định địa chỉ trong hợp ngữ

❖ **Toán hạng trực tiếp:**

Toán hạng trực tiếp là một biểu thức hằng xác định. Các hằng số có thể ở dạng thập phân (có dấu và không dấu), nhị phân, thập lục phân, các hằng số định nghĩa bằng lệnh EQU, …

```
VD:  MOV  AH,10
     MOV  AH,1010b
     MOV  AH,0Ah
     MOV  AH,A12
     MOV  AX,OFFSET msg
     MOV  AX,SEG msg
```

❖ **Toán hạng thanh ghi:**

Các thanh ghi có thể sử dụng trong phép định địa chỉ thanh ghi là AH, BH, CH, DH, AL, BL, CL, DL, AX, BX, CX, DX, SP, BP, SI, DI, CS, DS, ES, SS.

❖ **Toán hạng bộ nhớ:**

➢ **Trực tiếp:**

Toán hạng này xác định dữ liệu lưu trong bộ nhớ tại một địa chỉ xác định khi dịch, địa chỉ này là một biểu thức hằng (có thể kết hợp với toán tử chỉ số [ ] hay toán tử +, -, :). Thanh ghi đoạn mặc định là thanh ghi DS nhưng ta có thể dùng toán tử : để chỉ thanh ghi đoạn khác.

```
VD:  A      DW   1000h
     B      DB   100   DUP(0)
     MOV  AX,A              ; Chuyển nội dung của biến A vào
     MOV  AX,[A]            ; thanh ghi AX
     MOV  AH,B              ; Truy xuất phần tử đầu tiên của
     MOV  AH,B[0]           ; mảng B
     MOV  AH,B + 1          ; Truy xuất phần tử thứ hai của
     MOV  AH,B[1]           ; mảng B
     MOV  AH,B  + 5         ; Truy xuất phần tử thứ 6 của
     MOV  AH,B[5]           ; mảng B
```

Chú ý rằng lệnh MOV AX,[1000h] sẽ chuyển giá trị 1000h vào thanh ghi AX. Nếu muốn chuyển nội dung tại ô nhớ 1000h vào thanh ghi AX thì phải dùng lệnh MOV AX,DS:[1000h] hay MOV AX,DS:1000h

➢ **Gián tiếp:**

Toán hạng bộ nhớ gián tiếp cho phép dùng các thanh ghi BX, BP, SI, DI để chỉ các giá trị trong bộ nhớ.

VD:
```
MOV  BX,2
MOV  SI,3
MOV  AH,B[BX]          ; Chuyển phần tử thứ 3 của mảng B
                       ; vào thanh ghi AH
MOV  AH,B[BX+1]        ; Chuyển phần tử thứ 4 của mảng B
MOV  AH,B[BX]+1        ; vào thanh ghi AH (BX + 1 = 3)
MOV  AH,B[BX+SI]       ; Chuyển phần tử thứ 6 của mảng B
MOV  AH,B[BX][SI]      ; vào thanh ghi AH
MOV  AH,[B+BX+SI]      ; BX + SI = 5
MOV  AH,[B][BX][SI]
MOV  AH,B[BX+SI+5]     ; Chuyển phần tử thứ 11 của mảng B
MOV  AH,B[BX][SI]+5    ; vào thanh ghi AH
MOV  AH,[B+BX+SI+5]    ; BX + SI + 5 = 10
```

6. Tạo và thực thi chương trình hợp ngữ

Ta có thể tạo và thực thi một chương trình hợp ngữ trên một máy PC theo các bước sau:

- Dùng một chương trình soạn thảo văn bản ***không định dạng*** (như NC, Notepad, …) tạo một tập tin chứa chương trình hợp ngữ (gán phần mở rộng của tập tin này là .ASM, giả sử là TEMP.ASM).
- Dùng chương trình TASM.EXE (Turbo Assembler) để dịch ra mã máy dạng .OBJ: **TASM TEMP**
- Sau khi dịch xong, ta sẽ được file TEMP.OBJ chứa các mã máy của chương trình. Để chuyển thành file thực thi, ta dùng chương trình TLINK.EXE để chuyển thành tập tin .EXE: **TLINK TEMP**
- Nếu tập tin thực thi ở dạng .COM thì ta dùng thêm chương trình EXE2BIN.EXE: **EXE2BIN TEMP TEMP.COM**

7. Tập lệnh hợp ngữ

7.1. Nhóm lệnh chuyển dữ liệu

7.1.1. Nhóm lệnh chuyển dữ liệu đa dụng

❖ Lệnh **MOV dst,src**: chuyển nội dung toán hạng src vào toán hạng dst.

Toán hạng nguồn src có thể là thanh ghi (reg), bộ nhớ (mem) hay giá trị tức thời (immed); toán hạng đích dst có thể là reg hay mem.

Lệnh MOV có thể có các trường hợp sau:

```
Reg8 ← reg8          MOV  AL,AH
Reg16 ← reg16        MOV  AX,BX
Mem8 ← reg8          MOV  [BX],AL
Reg8 ← mem8          MOV  AL,[BX]
Mem16 ← reg16        MOV  [BX],AX
Reg16 ← mem16        MOV  AX,[BX]
Reg8 ← immed8        MOV  AL,04h
Mem8 ← immed8        MOV  mem[BX],01h
Reg16 ← immed16      MOV  AL,0F104h
```

Mem16 ← immed16	MOV mem[BX],0101h
SegReg ← reg16	MOV DS,AX
SegReg ← mem16	MOV DS,mem
Reg16 ← segreg	MOV AX,DS
Mem16 ← segreg	MOV [BX],DS

- Lệnh MOV không ảnh hưởng đến các cờ.
- Không thể chuyển trực tiếp dữ liệu giữa hai ô nhớ mà phải thông qua một thanh ghi

```
MOV  AX,mem1

MOV  mem2,AX
```

- Không thể chuyển giá trị trực tiếp vào thanh ghi đoạn

```
MOV  AX,1010h

MOV  DS,AX
```

- Không thể chuyển trực tiếp giữa 2 thanh ghi đoạn
- Không thể dùng thanh ghi CS làm toán hạng đích

❖ Lệnh **XCHG dst,src**: (Exchange) hoán chuyển nội dung 2 toán hạng.

Toán hạng chỉ có thể là reg hay mem.

- Lệnh XCHG không ảnh hưởng đến các cờ
- Không thể dùng cho các thanh ghi đoạn

❖ Lệnh **PUSH src**: cất nội dung một thanh ghi vào stack.

Toán hạng chỉ có thể là reg16

❖ Lệnh **POP dst**: lấy dữ liệu 16 bit từ stack đưa vào toán hạng dst.

Ta có thể dùng nhiều lệnh PUSH để cất dữ liệu vào stack nhưng khi dùng lệnh POP để lấy dữ liệu ra thì phải dùng theo thứ tự ngược lại.

```
PUSH    AX
PUSH    BX
PUSH    CX
…
POP     CX
POP     BX
POP     AX
```

❖ Lệnh **XLAT [src]**: chuyển nội dung của ô nhớ 8 bit vào thanh ghi AL.

Địa chỉ ô nhớ xác định bằng cặp thanh ghi DS:BX (nếu không chỉ ra src) hay src, địa chỉ offset chứa trong thanh ghi AL.

Lệnh XLAT tương đương với các lệnh:

```
MOV  AH,0
MOV  SI,AX
```

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

7.1.2. Nhóm lệnh chuyển địa chỉ

- Lệnh **LEA reg16,mem16**: (Load Effective Address) chuyển địa chỉ offset của toán hạng bộ nhớ vào thanh ghi reg16.

Lệnh này sẽ tương đương với **MOV reg16, OFFSET mem16**

- Lệnh **LDS reg16,mem32**: (Load pointer using DS) chuyển nội dung bộ nhớ toán hạng mem32 vào cặp thanh ghi DS:reg16.

Lệnh LDS AX,mem tương đương với:

```
MOV  AX,mem
MOV  BX,mem+2
MOV  DS,BX
```

- Lệnh **LES reg16,mem32**: (Load pointer using ES) giống như lệnh LDS nhưng dùng cho thanh ghi ES

7.1.3. Nhóm lệnh chuyển cờ hiệu

- Lệnh **LAHF**: (Load AH from flag) chuyển các cờ SF, ZF, AF, PF và CF vào các bit 7,6,4,2 và 0 của thanh ghi AH (3 bit còn lại không đổi)

- Lệnh **SAHF**: (Store AH into flag) chuyển các bit 7,6,4,2 và 0 của thanh ghi AH vào các cờ SF, ZF, AF, PF và CF.

- Lệnh **PUSHF**: chuyển thanh ghi cờ vào stack

- Lệnh **POPF**: lấy dữ liệu từ stack chuyển vào thanh ghi cờ

7.1.4. Nhóm lệnh chuyển dữ liệu qua cổng

Mỗi I/O port giao tiếp với CPU sẽ có một địa chỉ 16 bit cho nó. CPU gởi hay nhận dữ liệu từ cổng bằng cách chỉ đến địa chỉ cổng đó. Tuỳ theo chức năng mà cổng có thể: chỉ đọc dữ liệu (input port), chỉ ghi dữ liệu (output port) hay có thể đọc và ghi dữ liệu (input/output port).

- Lệnh **IN**: đọc dữ liệu từ cổng và đưa vào thanh ghi AL

```
IN    AL,port8
IN    AL,DX
```

Nếu địa chỉ port chỉ có 8 bit thì có thể đưa giá trị trực tiếp vào, nếu là 16 bit thì phải thông qua thanh ghi AX.

- Lệnh **OUT**: ghi dữ liệu trong thanh ghi AL ra cổng

```
OUT  port8,AL
OUT  DX,AL
```

VD:

```
MOV  AL,3
OUT  61h,AL          ; Gởi giá trị 03h ra cổng 61h
MOV  AL,1
```

```
MOV  DX,03F8h          ; Xuất ra cổng máy in
OUT  DX,AL
MOV  DX,03F8h
IN   AL,DX             ; Đọc dữ liệu từ cổng máy in
```

7.2. Nhóm lệnh chuyển điều khiển

7.2.1. Lệnh nhảy không điều kiện JMP

JMP label

JMP reg/mem

Lệnh JMP dùng để chuyển điều khiển chương trình từ vị trí này sang vị trí khác (thay đổi nội dung cặp thanh ghi CS:IP).

7.2.2. Lệnh nhảy có điều kiện

Lệnh nhảy có điều kiện chỉ sử dụng cho các nhãn nằm trong khoảng từ –127 đến 128 byte so với vị trí của lệnh.

❖ Lệnh **JA label**: (Jump if Above)

Nếu CF = 0 và ZF = 0 thì JMP label

❖ Lệnh **JAE label**: (Jump if Above or Equal)

Nếu CF = 0 thì JMP label

❖ Lệnh **JB label**: (Jump if Below)

Nếu CF = 1 thì JMP label

❖ Lệnh **JBE label**: (Jump if Below or Equal)

Nếu CF = 1 hoặc ZF = 1 thì JMP label

❖ Lệnh **JNA label**: (Jump if Not Above)

Giống lệnh JBE

❖ Lệnh **JNAE label**: (Jump if Not Above or Equal)

Giống lệnh JB

❖ Lệnh **JNB label**: (Jump if Not Below)

Giống lệnh JAE

❖ Lệnh **JNBE label**: (Jump if Not Below or Equal)

Giống lệnh JA

❖ Lệnh **JG label**: (Jump if Greater)

Nếu SF = OF và ZF = 0 thì JMP label

❖ Lệnh **JGE label**: (Jump if Greater or Equal)

Nếu SF = OF thì JMP label

❖ Lệnh **JL label**: (Jump if Less)

Nếu SF <> OF thì JMP label

❖ Lệnh **JLE label**: (Jump if Less or Equal)

Nếu CF <> OF hoặc ZF = 1 thì JMP label

❖ Lệnh **JNG label**: (Jump if Not Greater)

Giống lệnh JLE

❖ Lệnh **JNGE label**: (Jump if Not Greater or Equal)

Giống lệnh JL

❖ Lệnh **JNL label**: (Jump if Not Less)

Giống lệnh JGE

❖ Lệnh **JNLE label**: (Jump if Not Less or Equal)

Giống lệnh JG

❖ Lệnh **JC label**: (Jump if Carry)

Giống lệnh JB

❖ Lệnh **JNC label**: (Jump if Not Carry)

Giống lệnh JNB

❖ Lệnh **JZ label**: (Jump if Zero)

Nếu ZF = 1 thì JMP label

❖ Lệnh **JE label**: (Jump if Equal)

Giống lệnh JZ

❖ Lệnh **JNZ label**: (Jump if Not Zero)

Nếu ZF = 0 thì JMP label

❖ Lệnh **JNE label**: (Jump if Equal)

Giống lệnh JNZ

❖ Lệnh **JS label**: (Jump on Sign)

Nếu SF = 1 thì JMP label

❖ Lệnh **JNS label**: (Jump if No Sign)

Nếu SF = 0 thì JMP label

❖ Lệnh **JO label**: (Jump on Overflow)

Nếu OF = 1 thì JMP label

❖ Lệnh **JNO label**: (Jump if No Overflow)

Nếu OF = 0 thì JMP label

❖ Lệnh **JP label**: (Jump on Parity)

Nếu PF = 1 thì JMP label

❖ Lệnh **JNP label**: (Jump if No Parity)

Nếu PF = 0 thì JMP label

❖ Lệnh **JCXZ label**: (Jump if CX Zero)

Nếu CX = 1 thì JMP label

7.2.3. Lệnh so sánh

CMP left(reg/mem), right(reg/mem/immed)

Lệnh CMP dùng để so sánh nội dung 2 toán hạng, kết quả chứa vào thanh ghi cờ và không làm thay đổi nội dung các toán hạng.

VD: Đoạn chương trình so sánh 2 số A và B: A >B thì nhảy đến label1, A = B thì nhảy đến label2, A < B thì nhảy đến label3.

```
MOV  AX,A
CMP  AX,B
JG   label1
JL   label2
JMP  label3
```

7.2.4. Các lệnh vòng lặp

❖ Lệnh **LOOP**:

LOOP label

Mô tả:

CX = CX – 1
Nếu CX <> 0 thì JMP label

❖ Lệnh **LOOPE**:

LOOPE label

Mô tả:
CX = CX – 1
Nếu (ZF = 1) và (CX <> 0) thì JMP label

❖ Lệnh **LOOPZ**:

Giống lệnh LOOPE

❖ Lệnh **LOOPNE**:

LOOPNE label

Mô tả:

CX = CX – 1
Nếu (ZF = 0) và (CX <> 0) thì JMP label

❖ Lệnh **LOOPNZ**:

Giống lệnh LOOPNE

7.2.5. Lệnh liên quan đến chương trình con

❖ Lệnh **CALL**:

Lệnh CALL dùng để gọi một chương trình con, có thể là near hay far.

```
CALL    label     ; Gọi chương trình con tại vị trí xác định
                  ; bởi nhãn label
CALL    reg/mem   ; Gọi chương trình con tại vị trí xác định
                  ; trong reg/mem
```

❖ Lệnh **RET**: (return)

RET [n]

RETN [n]

RETF [n]

Lệnh RET dùng để kết thúc chương trình con, điều khiển sẽ được đưa về địa chỉ trước khi gọi chương trình con. RETN để kết thúc chương trình con dạng near và RETF dùng để kết thúc chương trình con dạng far.

Trong trường hợp lệnh RET có hằng số n theo sau thì sẽ cộng với thanh ghi SP giá trị n (n phải là số chẵn). Lệnh này dùng để loại bỏ một số tham số chương trình con sử dụng ra khỏi stack.

7.3. Nhóm lệnh xử lý số học

7.3.1. Xử lý phép cộng

❖ Lệnh **ADD dst,src**:

dst ← dst + src

Toán hạng src có thể là reg, mem hay immed còn toán hạng dst là reg hay mem.

- Không thể cộng trực tiếp 2 thanh ghi đoạn
- Lệnh ADD ảnh hưởng đến các cờ sau:
  + Cờ CF: = 1 khi kết quả phép cộng có nhớ hay có mượn
  + Cờ AF: = 1 khi kết quả phép cộng có nhớ hay có mượn đối với 4 bit thấp
  + Cờ PF: = 1 khi kết quả phép cộng có tổng 8 bit thấp là một số chẵn.
  + Cờ ZF: = 1 khi kết quả phép cộng là 0.
  + Cờ SF: = 1 nếu kết quả phép cộng là một số âm
  + Cờ OF: = 1 nếu kết quả phép cộng bị sai dấu, nghĩa là vượt ra ngoài phạm vi lớn nhất hay nhỏ nhất mà số có dấu có thể chứa trong toán hạng dst.

❖ Lệnh **ADC dst, src**: (Add with Carry)

dst ← dst + src + CF

Lệnh ADC thường dùng để cộng các số lớn hơn 16 bit.

❖ Lệnh **INC dst**: (Increment)

dst ← dst + 1

Dst có thể là reg hay mem.

❖ Lệnh **AAA**: (ASCII Adjust for Addition)

Hiệu chỉnh kết quả phép cộng 2 số BCD dạng không nén (mỗi chữ số BCD lưu bằng 1 byte).

```
VD:  MOV  AX,9
     MOV  BX,3
     ADD  AL,BL        ; Kết quả là AX = 0Ch
     AAA               ; AX = 0102h (AH = 1, AL = 2)
```

Lệnh AAA chỉ ảnh hưởng đến các cờ AF và CF, không ảnh hưởng đến các cờ còn lại.

❖ Lệnh **DAA**: (Decimal Adjust for Addition)

Hiệu chỉnh kết quả phép cộng 2 số BCD dạng nén (mỗi chữ số BCD lưu bằng 4 bit, nghĩa là 1 byte biểu diễn được các số nguyên từ 0 đến 99).

```
VD:  MOV  AX,4338h
     ADD  AL,AH        ; AX ← 437Bh
     DAA               ; AX ← 4381h (43 + 38 = 81)
```

Lệnh DAA chỉ ảnh hưởng đến các cờ AF, CF, PF, SF, ZF và không ảnh hưởng đến thanh ghi AH.

7.3.2. Xử lý phép trừ

❖ Lệnh **SUB dst,src**:

dst ← dst - src

Toán hạng src có thể là reg, mem hay immed còn toán hạng dst chỉ có thể là reg hay mem.

- Không thể trừ trực tiếp thanh ghi đoạn
- Ảnh hưởng đến các cờ AF, CF, OF, PF, SF và ZF.

❖ Lệnh **SBB dst,src**:

dst ← dst – src – CF

Lệnh ADC thường dùng để trừ các số lớn hơn 16 bit.

❖ Lệnh **DEC dst**: (decrement)

dst ← dst – 1

dst là reg hay mem. Lệnh DEC ảnh hưởng đến các cờ AF, OF, PF, SF, ZF.

❖ Lệnh **NEG dst**:

dst ← - dst

dst là reg hay mem.

Lệnh NEG ảnh hưởng đến các cờ:

CF = 1 nếu nội dung kết quả là số khác 0.
SF = 1 nếu nội dung kết quả là số âm khác 0.
PF = 1 nếu tổng 8 bit thấp là một số chẵn.
ZF = 1 nếu nội dung kết quả là 0.
OF = 1 nếu nội dung toán hạng dst là 80h (dạng byte) hay 8000h (dạng word).

VD: Nếu muốn thực hiện phép toán 100 – AH, ta không thể cùng lệnh:

```
SUB   100,AH
```

mà phải dùng lệnh:

```
SUB   AH,100
NEG   AH
```

❖ Lệnh **AAS**: (Ascii Adjust for Substract)

Hiệu chỉnh kết quả phép trừ 2 số BCD dạng không nén (mỗi chữ số BCD lưu bằng 1 byte). Lệnh AAS chỉ ảnh hưởng cờ AF và CF.

❖ Lệnh **DAS**: (Decimal Adjust for Substract)

Hiệu chỉnh kết quả phép trừ 2 số BCD dạng nén (mỗi chữ số BCD lưu bằng 4 bit). Lệnh AAS chỉ ảnh hưởng cờ AF và CF.

7.3.3. Xử lý phép nhân

❖ Lệnh **MUL src**:

Nếu src là reg hay mem 8 bit: AX ← AL*src

Nếu src là reg hay mem 16 bit: DX:AX ← AX*src

Lệnh MUL chỉ ảnh hưởng đến cờ CF và OF.

❖ Lệnh **IMUL src**:

Giống như lệnh MUL nhưng kết quả là số có dấu.

❖ Lệnh **AAM**: (Ascii Adjust for Multiple)

Hiệu chỉnh kết quả phép nhân 2 số BCD dạng không nén, lệnh AAM thực hiện chia AL cho 10, lưu phần thương vào AL và phần dư vào AH. Lệnh AAM ảnh hưởng đến các cờ PF, SF và ZF.

7.3.4. Xử lý phép chia

❖ Lệnh **DIV src**:

Nếu src là reg/mem 8 bit: AL ← AX DIV src và AH ← AX MOD src

Nếu src là reg/mem 16 bit: AX ← DX:AX DIV src và DX ← DX:AX MOD src

Lệnh DIV không ảnh hưởng đến các cờ nhưng xảy ra tràn trong các trường hợp sau:

- Chia cho 0
- Thương lớn hơn 256 đối với dạng 8 bit.
- Thương lớn hơn 65536 đối với dạng 16 bit.

❖ Lệnh **IDIV src**:

Giống như lệnh DIV nhưng kết quả là số có dấu. Các trường hợp tràn:

- Chia cho 0
- Thương nằm ngoài khoảng (-128,127) đối với dạng 8 bit.
- Thương nằm ngoài khoảng (-32767,32768) đối với dạng 16 bit.

❖ Lệnh **AAD**: (Ascii Adjust for Division)

Hiệu chỉnh kết quả phép chia 2 số BCD dạng không nén. Lưu ý rằng lệnh AAD phải được thực hiện trước lệnh chia. Sau khi thực hiện chia thì phải hiệu chỉnh lại dạng BCD bằng cách dùng lệnh AAM.

❖ Lệnh **CBW**: (Convert Byte to Word)

Nếu AL < 80h thì AH = 0, ngược lại AH = 0FFh

Lệnh CBW dùng để chuyển số nhị phân có dấu 8 bit thành số nhị phân có dấu 16 bit.

❖ Lệnh **CWD**: (Convert Word to Double word)

Nếu AX < 8000h thì DX = 0, ngược lại DX = 0FFFFh

Lệnh CWD dùng để chuyển số nhị phân có dấu 16 bit thành số nhị phân có dấu 32 bit chứa trong DX:AX.

7.3.5. Dịch chuyển và quay

❖ Lệnh **SHL**: (Shift Logical Left)

SHL dst,1

SHL dst,CL

Dịch trái 1 bit hay CL bit.

CF ← dst7 ← dst6 … ← dst0 ← 0

❖ Lệnh **SHR**: (Shift Logical Right)

SHR dst,1

SHR dst,CL

Dịch phải 1 bit hay CL bit.

0 → dst7 → dst6 … → dst0 → CF

❖ Lệnh **SAL**: giống SHL

❖ Lệnh **SAR**:

Giống như lệnh SHR nhưng giá trị bit dst7 không thay đổi, nghĩa là

dst7 → dst7 → dst6 … → dst0 → CF

❖ Lệnh **ROL**: (Rotate Left)

ROL dst,1

ROL dst,CL

Quay trái 1 bit hay CL bit.

CF ← dst7 ← dst6 … ← dst0 ← dst7

❖ Lệnh **ROR**: (Rotate Right)

ROR dst,1

ROR dst,CL

Quay phải 1 bit hay CL bit.

dst0 → dst7 → dst6 … → dst0 → CF

❖ Lệnh **RCL**: (Rotate though Carry Left)

RCL dst,1

RCL dst,CL

Quay trái 1 bit hay CL bit.

CF ← dst7 ← dst6 … ← dst0 ← CF

❖ Lệnh **RCR**: (Rotate though Carry Right)

RCR dst,1

RCR dst,CL

Quay phải 1 bit hay CL bit.

CF → dst7 → dst6 … → dst0 → CF

7.3.6. Các lệnh logic

❖ Lệnh **AND**:

AND dst,src

dst ← dst AND src

CF ← 0, OF ← 0

Src là reg, mem hay immed còn dst là reg, mem.

❖ Lệnh **OR**:

OR dst,src

dst ← dst OR src

CF ← 0, OF ← 0

❖ Lệnh **XOR**:

XOR dst,src

dst ← dst XOR src

CF ← 0, OF ← 0

❖ Lệnh **NOT**:

NOT dst

dst ← NOT dst

Lệnh NOT không ảnh hưởng đến các cờ.

❖ Lệnh **TEST**:

TEST dst,src

Lệnh TEST thực hiện phép toán AND 2 toán hạng nhưng chỉ ảnh hưởng đến các cờ và không ảnh hưởng đến toán tử.

7.4. Nhóm lệnh xử lý chuỗi

Bao gồm các lệnh sau:

- Lệnh MOVS: chuyển dữ liệu từ vùng nhớ này sang vùng nhớ khác.

+ **MOVSB**: chuyển 1 byte từ vị trí chỉ đến bởi SI đến vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 1, DI ← DI + 1 còn nếu DF = 1 thì SI ← SI - 1, DI ← DI - 1.

+ **MOVSW**: chuyển 1 word từ vị trí chỉ đến bởi SI đến vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 2, DI ← DI + 2 còn nếu DF ← 1 thì SI ← SI - 2, DI ← DI - 2.

- Lệnh CMPS: so sánh nội dung 2 vùng nhớ
  + **CMPSB**: so sánh 1 byte tại vị trí chỉ đến bởi SI và tại vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 1, DI ← DI + 1 còn nếu DF ← 1 thì SI ← SI - 1, DI ← DI - 1.
  + **CMPSW**: so sánh 1 word tại vị trí chỉ đến bởi SI và tại vị trí chỉ bởi DI. Nếu DF = 0 thì SI ← SI + 2, DI ← DI + 2 còn nếu DF = 1 thì SI ← SI - 2, DI ← DI - 2.

- Lệnh SCAS: tìm một phần tử trong vùng nhớ, địa chỉ vùng nhớ xác định bằng cặp thanh ghi ES:DI, giá trị cần tìm đặt trong thanh ghi AL, nếu tìm thấy thì ZF = 1. Giá trị của DI và SI thay đổi giống như trên.

- Lệnh LODS: đưa một byte hay word có địa chỉ xác định bởi cặp thanh ghi DS:SI vào thanh ghi AL hay AX. Giá trị của DI và SI thay đổi giống như trên.

- Lệnh STOS: chuyển nội dung của AL hay AX vào vùng nhớ xác định bởi cặp thanh ghi ES:DI. Giá trị của DI và SI thay đổi giống như trên.

8. Các cấu trúc cơ bản trong lập trình hợp ngữ

8.1. Cấu trúc tuần tự

Cấu trúc tuần tự là cấu trúc đơn giản nhất. Trong cấu trúc tuần tự, các lệnh được sắp xếp tuần tự, lệnh này tiếp theo lệnh kia.

Lệnh 1
Lệnh 2
…
Lệnh n

VD: Cộng 2 giá trị của thanh ghi BX và CX, rồi nhân đôi kết quả, kết quả cuối cùng chứa trong AX

```
MOV  AX,BX
ADD  AX,CX      ; Cộng BX với CX
SHL  AX,1       ; Nhân đôi
```

8.2. Cấu trúc IF – THEN, IF – THEN – ELSE

IF Điều kiện THEN Công việc
IF Điều kiện THEN Công việc1 ELSE Công việc2
VD: Gán BX = |AX|

```
CMP  AX,0       ; AX > 0?
```

```
            JNL   DUONG     ; AX dương
            NEG   AX        ; Nếu AX < 0 thì đảo dấu
DUONG:      MOV   BX,AX
NEXT:
```

VD: Gán CL giá trị bit dấu của AX

```
            CMP   AX,0      ; AX > 0?
            JNS   AM        ; AX âm
            MOV   CL,1      ; CL = 1 (AX dương)
            JMP   NEXT
      AM:   MOV   CL,0      ; CL = 0 (AX âm)
      NEXT:
```

8.3. Cấu trúc CASE

CASE Biểu thức
Giá trị 1: Công việc 1
Giá trị 2: Công việc 2
…
Giá trị n: Công việc n
END

VD: Nếu AX > 0 thì BH = 0, nếu AX < 0 thì BH = 1. Ngược lại BH = 2

```
      CMP   AX,0
            JL    AM
      JE    KHONG
            JG    DUONG
DUONG:      MOV   BH,0
            JMP   NEXT
AM:         MOV   BH,1
            JMP   NEXT
KHONG:      MOV   BH,2
NEXT:
```

8.4. Cấu trúc FOR

FOR Số lần lặp DO Công việc

VD: Cho vùng nhớ M dài 200 bytes trong đoạn dữ liệu, chương trình đếm số chữ A trong vùng nhớ M như sau:

```
            MOV   CX,200            ; Đếm 200 bytes
            MOV   BX,OFFSET M       ; Lấy địa chỉ vùng nhớ
            XOR   AX,AX             ; AX = 0
NEXT:       CMP   BYTE PTR [BX],'A'; So sánh với chữ A
            JNZ   ChuA              ; Nếu không phải là chữ A thì tiếp
            INC   AX                ; tục, ngược lại thì tăng AX
ChuA:       INC   BX
            LOOP        NEXT
```

8.5. Cấu trúc lặp WHILE

WHILE Điều kiện DO Công việc

VD: Chương trình đọc vùng nhớ bắt đầu tại địa chỉ 1000h vào thanh ghi AH, đến khi gặp ký tự '$' thì thoát:

```
        MOV  BX,1000h
CONT:   CMP  AH,'$'
        JZ   NEXT
        MOV  AH,DS:[BX]
        JMP  CONT
NEXT:
```

8.6. Cấu trúc lặp REPEAT

REPEAT Công việc UNTIL Điều kiện

VD: Chương trình đọc vùng nhớ bắt đầu tại địa chỉ 1000h vào thanh ghi AH, đến khi gặp ký tự '$' thì thoát:

```
        MOV  BX,1000h
CONT:   MOV  AH,DS:[BX]
        CMP  AH,'$'
        JZ   NEXT
        JMP  CONT
NEXT:
```

9. Các ngắt của 8086

Bảng 2.4:

Vector ngắt	Công dụng
00h	CPU: tác động khi chia cho 0
01h	CPU: chương trình thực thi từng bước
02h	CPU: ngắt không che được
03h	CPU: tạo điểm dừng chochương trình
04h	CPU: tác động khi kết quả số học tràn
05h	Tác động khi nhấn Print Screen
06h - 07h	Dành riêng
08h	Tác động bởi nhịp đồng hồ (18.2 lần/s)
09h	Tác động khi có phím nhấn
0Ah	Dành riêng
0Bh - 0Ch	Tác động phần cứng liên lạc nối tiếp
0Dh	Đĩa cứng
0Eh	Đĩa mềm
0Fh	Máy in
10h	BIOS: màn hình
11h	BIOS: xác định cấu hình máy tính
12h	BIOS: thông báo kích thước RAM
13h	BIOS: gọi các phục vụ đĩa cứng/mềm

14h	BIOS: giao tiếp nối tiếp
15h	BIOS: truy xuất cassette hay mở rộng ngắt
16h	BIOS: xuất / nhập bàn phím
17h	BIOS: máy in
18h	Xâm nhập ROM basic
19h	BIOS: khởi động máy tính
1Ah	BIOS: ngày / giờ hệ thống
1Bh	Lấy điều khiển từ ngắt bàn phím
1Ch	Lấy điều khiển từ ngắt đồng hồ (sau int 08h)
1Dh	Địa chỉ bảng tham số màn hình
1Eh	Địa chỉ bảng tham số đĩa
1Fh	Địa chỉ bộ mã ký tự
20h	DOS: kết thúc chương trình
21h	DOS: các chức năng DOS
22h	Địa chỉ cần chuyển khi kết thúc chương trình
23h	Địa chỉ cần chuyển khi gặp Ctrl – Break
24h	Địa chỉ cần chuyển khi gặp lỗi
25h	DOS: đọc đĩa cứng / mềm
26h	DOS: ghi đĩa cứng / mềm
27h	DOS: chấm dứt chương trình và thường trú
28h – 3Fh	Dành riêng cho DOS
40h	BIOS: các chức năng đĩa mềm
41h	Bảng thông số đĩa cứng thứ nhất
42h – 45h	Dành riêng
46h	Bảng thông số đĩa cứng thứ hai
47h – 49h	Định nghĩa do người sử dụng
4Ah	Giờ báo hiệu (chỉ trong AT)
4Bh – 67h	Định nghĩa do người sử dụng
68h – 6Fh	Không sử dụng
70h	Đồng hồ thời gian thực (chỉ trong AT)
71h – 7Fh	Dành riêng
80h – 85h	Dành riêng
86h – F0h	Sử dụng bởi chương trình thông dịch BASIC
F1h – FFh	Không sử dụng

9.1. Ngắt 21h

❖ **Hàm 01h**: nhập một ký tự từ bàn phím và hiện ký tự nhập ra màn hình. Nếu không có ký tự nhập, hàm 01h sẽ đợi cho đến khi nhập.

- Gọi: AH = 01h
- Trả về: AL chứa mã ASCII của ký tự nhập

```
MOV  AH,01h
INT   21h          ; AL chứa mã ASCII của ký tự nhập
```

❖ **Hàm 02h**: xuất một ký tự trong thanh ghi DL ra màn hình tại vị trí con trỏ hiện hành

- Gọi AH = 02h, DL = mã ASCII của ký tự
- Trả về: không có

```
MOV  AH,02h
MOV  DL,'A'
INT  21h
```

❖ **Hàm 08h**: giống hàm 01h nhưng không hiển thị ký tự ra màn hình

❖ **Hàm 09h**: xuất một chuỗi ký tự ra màn hình tại vị trí con trỏ hiện hành, địa chỉ chuỗi được chứa trong DS:DX và phải được kết thúc bằng ký tự $

- Gọi AH = 09h, DS:DX = địa chỉ chuỗi
- Trả về: không có

```
.DATA
Msg  DB   'Hello$'
…
MOV  AH,09h
LEA  DX,Msg
INT  21h
```

❖ **Hàm 0Ah**: nhập một chuỗi ký tự từ bàn phím (tối đa 255 ký tự), dùng phím ENTER kết thúc chuỗi

- Gọi AH = 0Ah, DS:DX = địa chỉ lưu chuỗi
- Trả về: không có

Chuỗi phải có dạng sau:

- Byte 0: Số byte tối đa cần đọc (kể cả ký tự Enter)
- Byte 1: số byte đã đọc
- Byte 2: lưu các ký tự đọc

```
.DATA
Msg  DB   101          ; Đọc tối đa 100 ký tự
     DB   ?
     DB   101  DUP(?)
     …
     MOV  AH,0Ah
     LEA  DX,Msg
     INT  21h
```

❖ **Hàm 0Bh**: kiểm tra phím nhấn trên bàn phím

- Gọi: AH = 0Bh
- Trả về: AL = 0FFh nếu có nhấn phím, AL = 0 nếu không nhấn phím

❖ **Hàm 4Ch**: kết thúc chương trình

```
MOV   AH,4Ch
INT   21h
```

9.2. Ngắt 10h

❖ **Xoá màn hình:**

- Gọi AX = 02h

- Trả về: không có

```
MOV  AX,02h
INT  10h
```

❖ **Chuyển toạ độ con trỏ:**

- Gọi AH = 02h, DH = dòng, DL = cột

```
MOV  AH,02h
MOV  DX,0F15h
INT  10h
```

10. Truyền tham số giữa các chương trình

Trong lập trình, một vấn đề ta cần quan tâm là truyền tham số giữa chương trình chính và chương trình con. Để thực hiện truyền tham số, ta có thể dùng các cách sau đây:

- Truyền tham số qua thanh ghi
- Truyền tham số qua ô nhớ (biến)
- Truyền tham số qua ô nhớ do thanh ghi chỉ đến
- Truyền tham số qua stack

10.1. Truyền tham số qua thanh ghi

Ta thực hiện truyền tham số qua thanh ghi bằng cách: một chương trình con sẽ đưa giá trị vào thanh ghi và chương trình con khác sẽ xử lý giá trị trên thanh ghi đó.

VD: Cộng giá trị tại 2 ô nhớ 1000h và 1001h, kết quả chứa trong 1002h (bye cao) và 1003h (byte thấp).

```
.MODEL    SMALL
.STACK    100h
.CODE
main PROC
     MOV       AX,@DATA
     MOV       DS,AX
; ••a giá tr• vào các ô nh•
     MOV       BYTE PTR DS:[1000h],10h
     MOV       BYTE PTR DS:[1001h],0FFh
     CALL      Read
     CALL      Sum
     Mov       AH,4Ch
     INT       21h
main ENDP
Read PROC      ; ••c d• li•u vào thanh ghi AX
     MOV       AH,DS:[1000h]
     MOV       AL,DS:[1001h]
     RET
Read ENDP

Sum  PROC
     ADD       AH,AL
     JZ        next
     MOV       DS:[1003h],1
next:     MOV       DS:[1002h],AH
```

```
RET
Sum  ENDP
END  main
```

10.2. Truyền tham số qua ô nhớ (biến)

Quá trình truyền tham số cũng giống như trên nhưng thay vì thực hiện thông qua thanh ghi, ta sẽ thực hiện thông qua các ô nhớ.

VD: Cộng giá trị tại 2 ô nhớ m1 và m2, kết quả chứa trong m3 (byte cao) và m4 (byte thấp).

```
.MODEL    SMALL
.STACK    100h
.DATA
     m1   db   ?
     m2   db   ?
     m3   db   ?
     m4   db   ?
.CODE
main PROC
     MOV       AX,@data
     MOV       DS,AX
     MOV       m1,10h    ; ••a giá tr• vào
     MOV       m2,0FFh   ; các ô nh•
     CALL      Sum
     MOV       AH,4Ch
     INT       21h
main ENDP
Sum  PROC
     MOV       m4,0
     MOV       AH,m1
     ADD       AH,m2
     JNC       next
     MOV       m4,1
next:     MOV       m3,AH
RET
Sum  ENDP
END  main
```

10.3. Truyền tham số qua ô nhớ do thanh ghi chỉ đến

Trong cách truyền tham số này, ta dùng các thanh ghi SI, DI, BX để chỉ địa chỉ offset của các tham số còn thanh ghi đoạn mặc định là DS.

VD: Cộng giá trị tại 2 ô nhớ m1 và m2, kết quả chứa trong m3 (byte cao) và m4 (byte thấp).

```
.MODEL    SMALL
.STACK    100h
.DATA
     m1   db   ?
     m2   db   ?
     m3   db   ?
     m4   db   ?
.CODE
main PROC
```

```
        MOV     AX,@data
        MOV     DS,AX
        LEA     SI,m1
        LEA     DI,m2
        LEA     BX,m3
        MOV     [SI],10h ; ••a giá tr• vào
        MOV     [DI],0FFh; các ô nh•
        CALL    Sum
        MOV     AH,4Ch
        INT     21h
main ENDP
Sum  PROC
        MOV     AL,[SI]
        ADD     AL,[DI]
        JZ      next
        MOV     [BX+1],1
next:   MOV     [BX],AL
RET
Sum  ENDP
END  main
```

10.4. Truyền tham số qua stack

Trong phương pháp truyền tham số này, ta dùng stack làm nơi chứa các tham số cần truyền thông qua các tác vụ PUSH và POP.

VD: Cộng giá trị tại 2 ô nhớ m1 và m2, kết quả chứa trong m3 (byte cao) và m4 (byte thấp).

```
.MODEL  SMALL
.STACK  100h
.DATA
        m1  dw  ?
        m2  dw  ?
        m3  dw  ?
        m4  dw  ?
.CODE
main PROC
        MOV     AX,@data
        MOV     DS,AX
        LEA     SI,m1
        LEA     DI,m2
        MOV     [SI],1234h    ; ••a giá tr• vào
        MOV     [DI],0FEDCh   ; các ô nh•
        PUSH    m1            ; ••a vào stack
        PUSH    m2
        CALL    Sum
        POP     m3   ; L•y k•t qu• ••a vào stack
        POP     m4
        MOV     AH,4Ch
        INT     21h
main ENDP
Sum  PROC
        POP     DX   ; L•u ••a ch• tr• v• c•a l•nh CALL
        POP     AX   ; L•y d• li•u t• stack
        POP     BX
```

```
            ADD     AX,BX
            JNC     next
            PUSH    1
        next:
            PUSH    AX
            PUSH    DX   ; L•y ••a ch• tr• v• c•a l•nh CALL
        RET
        Sum ENDP
        END main
```

11. Các ví dụ minh hoạ

11.1. In chuỗi ký tự ra màn hình

```
    .MODEL   SMALL
    .STACK   100h
    .DATA
        msg DB   'Hello$'
    .CODE
    main PROC
        MOV AX,@DATA      ; Kh•i ••ng thanh ghi DS
        MOV DS,AX
        MOV AX,02h        ; Xoá màn hình
        INT 10h
        MOV AH,02h        ; Chuy•n to• •• con tr•
        MOV DX,0C15h      ; ••n dòng 12 (0Ch) và c•t 21
(15h)
        INT 10h
        LEA DX,msg        ; ••a ch• thông •i•p
        MOV AH,09h        ; In thông •i•p ra màn hình
        INT 21h
        MOV AH,4Ch        ; K•t thúc ch••ng trình
        INT 21h
    main ENDP
    END main
```

11.2. In chuỗi ký tự ra màn hình tại toạ độ nhập vào

```
    .MODEL   SMALL
    .STACK   100h
    .DATA
        msg  DB   'Hello$'
        msg1 DB   'Nhap vao toa do:$'
        Crlf DB   0Dh,0Ah,'$'
        Td   DB   3
             DB   ?
             DB   3    DUP(?)
    .CODE
    main PROC
        MOV AX,@DATA
        MOV DS,AX         ; Kh•i ••ng thanh ghi DS
        MOV AX,02h
        INT 10h           ; Xóa màn hình
        LEA DX,msg1
        MOV AH,09h        ; In thông •i•p
        INT 21h
        CALL Nhap         ; Nh•p dòng
```

```
          MOV  CL,AL
          LEA  DX,Crlf        ; Xu•ng dòng
          MOV  AH,09h
          INT  21h
          CALL Nhap           ; Nh•p c•t
          MOV  CH,AL
          MOV  AH,02h         ; Chuy•n t•a •• con tr•
          MOV  DX,CX
          INT  10h
          LEA  DX,msg
          MOV  AH,09h         ; In ra màn hình
          INT  21h
          MOV  AH,4Ch         ; K•t thúc ch••ng trình
          INT  21h
     main ENDP
     Nhap PROC
          MOV  AH,0Ah         ; Nh•p vào
          LEA  DX,Td
          INT  21h
          LEA  BX,Td              ; L•y ch• s• hàng ch•c
          MOV  AL,DS:[BX+2]
          SUB  AL,'0'             ; Chuy•n t• d•ng ký t•
sang d•ng s•
          MOV  BL,10
          MUL  BL             ; Nhân s• hàng ch•c v•i 10
          PUSH AX
          LEA  BX,Td              ; L•y ch• s• hàng d•n v•
          MOV  AL,DS:[BX+3]
          SUB  AL,'0'
          POP  BX
          ADD  AL,BL
          RET
     Nhap ENDP
     END  main
```

11.3. Cộng 2 số nhị phân dài 5 byte

```
     .MODEL    SMALL
     .STACK    100h
     .DATA
          m1   DB   00h,08h,10h,13h,24h,00h
          m2   DB   0FFh,0FCh,0FAh,0F0h,0F1h,00h;
          m3   DB   6    DUP(0)
     .CODE
     main PROC
          MOV  AX,@DATA
          MOV  DS,AX          ; Kh•i ••ng thanh ghi DS
          LEA  SI,m1
          LEA  DI,m2
          LEA  BX,m3
          MOV  CX,6
          XOR  AL,AL
     next:     MOV  AL,[SI]
          ADC  AL,[DI]
          MOV  [BX],AL
          INC  BX
```

```
        INC  SI
        INC  DI
        LOOP next
        MOV  AH,4Ch
        INT  21h
    main ENDP
    END  main
```

11.4. Nhập một chuỗi ký tự và chuyển chữ thường thành chữ hoa

```
    .MODEL   SMALL
    .STACK   100h
    .DATA
        m1   DB   81
             DB   ?
             DB   81   DUP(?)
        m2   DB   'Chuoi da doi:$'
    .CODE
    main PROC
        MOV  AX,@DATA
        MOV  DS,AX          ; Kh•i ••ng thanh ghi DS
        MOV  ES,AX
        LEA  DX,m1
        MOV  AH,0Ah         ; Nh•p chu•i
        INT  21h
        LEA  SI,m1          ; L•y ••a ch• chu•i
        ADD  SI,2
        MOV  DI,SI          ; Chu•i ngu•n và •ích
                            ; trùng nhau
    Next:    LODSB          ; L•y ký t•
        CMP  AL,0Dh         ; N•u là ký t• Enter
                            ; thì k•t thúc
        JE   quit
        CMP  AL,'a'         ; N•u ký t• nh•p không ph•i
                            ; là ký t• th••ng t• 'a'
        JB   cont           ; ••n 'z' thì b• qua
        CMP  AL,'z'
        JA   cont
        SUB  AL,20h         ; Chuy•n ký t• th••ng
                            ; thành ký t• hoa
        STOSB               ; L•u ký t• v•a chuy•n
        DEC  DI             ; N•u là ký t• th••ng
                            ; thì dùng l•nh STOSB
                            ; nên DI t•ng lên 1,
                            ; ta ph•i gi•m DI
    cont:    INC  DI        ; T•ng lên ký t• k•
        JMP  next
    quit:    MOV  AL,'$'
        STOSB
        MOV  AX,02h         ; Xóa màn hình
        INT  10h
        LEA  DX,m2
        MOV  AH,09h
        INT  21h
        LEA  DX,m1+2
```

```
        MOV  AH,09h
        INT  21h
        MOV  AH,4Ch
        INT  21h
main ENDP
END  main
```

BÀI TẬP CHƯƠNG 2

1. Xác định địa chỉ offset của các ô nhớ có địa chỉ vật lý sau, biết rằng địa chỉ segment là 1ACDh:

a. 1FFFFh b. 20000h c. 2ABCDh d. 1ECDFh

2. Xác định địa chỉ vật lý của các địa chỉ sau:

a. 1234h:1234h b. 1111h:ABCDh c. AAAAh:BBBBh

3. Xác định nội dung của các thanh ghi AX, BX và các ô nhớ 1000h, 1001h, 1002h, 1003h sau khi thực thi các đoạn chương trình sau:

a.
```
MOV  AX,1000h
MOV  BL,7
MOV  BH,0F0h
AND  BH,AH
MOV  WORD PTR DS:[1000h],0F0h
MOV  BYTE PTR DS:[1002h],0Fh
MOV AX,DS:[1001h]
MOV AL,DS:[1002h]
```
b.
```
MOV  AX,1234h
MOV  BX,0AAAAh
MOV  CH,AL
MOV  CL,AH
MOV  DS:[1000h],AX
MOV  DS:[1002h],BX
```
c.
```
MOV  AX,12h
MOV  DS:[1001h],AX
MOV  BX,AX
ADD  BH,10h
MOV  DS:[1002h],BH
```

4. Viết chương trình xuất 10 chuỗi "Hello" ra màn hình tại dòng thứ 10, cột 10.

5. Viết chương trình nhập chuỗi từ bàn phím cho đến khi nhập ký tự 'T' thì xuất chuỗi nhập ra màn hình và kết thúc chương trình.

6. Viết chương trình thực hiện chuyển đổi một chuỗi ký tự trong bộ nhớ từ chữ thường thành chữ hoa và in chuỗi đã chuyển đổi lên màn hình.

7. Viết chương trình thực hiện in các ký tự chứa tại ô nhớ 1000h – 3000h theo thứ tự ngược lại.

8. Viết chương trình thực hiện chuyển đổi số nhị phân chứa trong thanh ghi DX thành số BCD chứa trong thanh ghi AX. Nếu kết quả chuyển đổi lớn hơn 16 bit thì giá trị trong thanh ghi AX là FFFFh.

9. Viết chương trình so sánh 2 arrray 8 bit A và B, mỗi array có 100 phần tử chứa từ địa chỉ 1000h (array A) và 2000h (array B). Nếu 2 array này giống

nhau thì lưu vào ô nhớ 3000h giá trị FFFFh. Ngược lại thì lưu vào ô nhớ 3000h địa chỉ đầu tiên của phần tử trong array A khác với phần tử trong array B.

10. Viết chương trình thực hiện đọc dữ liệu 16 bit liên tục từ thiết bị ngoại vi có địa chỉ 300h.

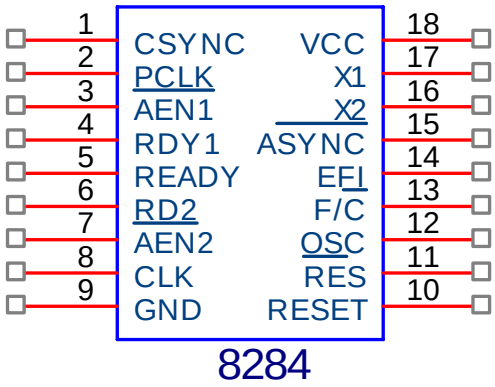
- Nếu dữ liệu đọc là 0 thì kết thúc chương trình
- Nếu dữ liệu đọc là FFFFh thì in lên màn hình chuỗi: “Error”
- Ngược lại thì xuất giá trị vừa nhập ra thiết bị ngoại vi có địa chỉ 301h

CHƯƠNG 3: TỔ CHỨC NHẬP / XUẤT

1. Các mạch phụ trợ 8284 và 8288

1.1. Mạch tạo xung nhịp 8284

Mạch tạo xung nhịp dùng để cung cấp xung nhịp cho µP.

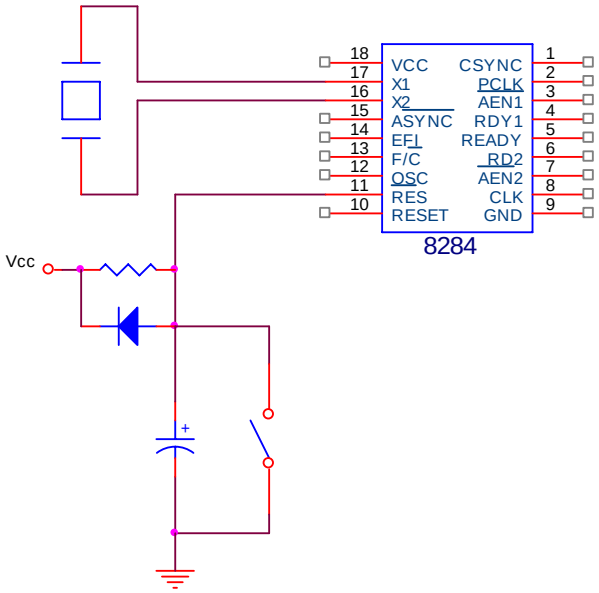


Hình 3.1 – Mạch tạo xung nhịp 8284

CSYNC (Clock Synchronisation): ngõ vào xung đồng bộ chung khi hệ thống có các 8284 dùng dao động ngoài tại chân EFI. Khi dùng mạch dao động trong thì phải nối đất.

PCLK (Peripheral Clock): xung nhịp f = f_X/6 (f_X là tần số thạch anh)

$\overline{AEN1}$, $\overline{AEN2}$ (Address Enable): cho phép chọn các chân RDY1, RDY2 báo hiệu trạng thái sẵn sàng của bộ nhớ hay thiết bị ngoại vi



Hình 3.2 – Mạch khởi động cho 8284

RDY1, RDY2 (Bus ready): tạo các chu kỳ đợi ở CPU

READY: nối đến chân READY của µP.

CLK (Clock): xung nhịp f = f_X/3, nối với chân CLK của µP.

RESET: nối với chân RESET của µP, là tín hiệu khởi động lại toàn hệ thống

$\overline{RES}$ (Reset Input): chân khởi động cho 8284

OSC: ngõ ra xung nhịp có tần số f_X

F/$\overline{C}$ (Frequency / Crystal): chọn nguồn tín hiệu chuẩn cho 8284, nếu ở mức cao thì chọn tần số xung nhịp bên ngoài, ngược lại thì dùng xung nhịp từ thạch anh

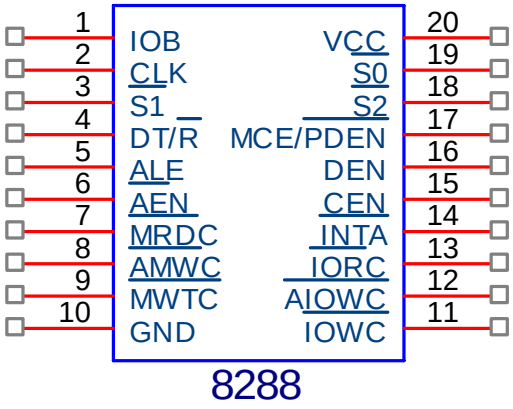
EFI (External Frequency Input): xung nhịp từ bộ dao động ngoài

$\overline{ASYNC}$: chọn chế độ làm việc cho tín hiệu RDY.

X1,X2: ngõ vào của thạch anh

1.2. Mạch điều khiển bus 8288

Mạch điều khiển bus 8288 lấy một số tín hiệu điều khiển của µP và cung cấp các tín hiệu điều khiển cần thiết cho hệ vi xử lý.



Hình 3.3 – Mạch điều khiển bus 8288

IOB (Input / Output Bus Mode): điều khiển để 8288 làm việc ở các chế độ bus khác nhau.

CLK (Clock): ngõ vào lấy từ xung nhịp hệ thống.

$\overline{S2}$, $\overline{S1}$, $\overline{S0}$: các tín hiệu trạng thái lấy trực tiếp từ µP. Tuỳ theo các giá trị nhận được mà 8288 sẽ đưa các tín hiệu theo bảng 3.1.

Bảng 3.1:

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	**Tạo tín hiệu**
0	0	0	$\overline{INTA}$
0	0	1	$\overline{IORC}$
0	1	0	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	Không
1	0	0	$\overline{MRDC}$
1	0	1	$\overline{MRDC}$
1	1	0	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	Không

DT/$\overline{R}$ (Data Transmit/Receive): μP truyền (1) hay nhận (0) dữ liệu.

ALE (Address Latch Enable): tín hiệu cho phép chốt địa chỉ

$\overline{AEN}$ (Address Enable): chờ thời gian trễ khoảng 150 ns sẽ tạo các tín hiệu điều khiển ở đầu ra của 8288 để đảm bảo rằng địa chỉ sử dụng đã hợp lệ.

$\overline{MRDC}$ (Memory Read Command): điều khiển đọc bộ nhớ

$\overline{MWTC}$ (Memory Write Command): điều khiển ghi bộ nhớ

$\overline{AMWC}$ (Advanced MWTC),: giống như $\overline{MWTC}$ nhưng hoạt động sớm hơn một chút dùng cho các bộ nhớ chậm đáp ứng kịp tốc độ μP.

$\overline{IOWC}$ (I/O Write Command): điều khiển ghi ngoại vi

$\overline{AIOWC}$ (Advanced IOWC),: giống như $\overline{IOWC}$ nhưng hoạt động sớm hơn một chút dùng cho các ngoại vi chậm đáp ứng kịp tốc độ μP.

$\overline{IORC}$ (I/O Read Command): điều khiển đọc ngoại vi

$\overline{INTA}$ (Interrupt Acknowledge): ngõ ra thông báo μP chấp nhận yêu cầu ngắt của thiết bị ngoại vi

CEN (Command Enable): cho phép đưa ra các tín hiệu của 8288.

DEN (Data Enable): tín hiệu điều khiển bus dữ liệu thành bus cục bộ hay bus hệ thống.

MCE / $\overline{PDEN}$ (Master Cascade Enable / Peripheral Data Enable): định chế độ làm việc cho mạch điều khiển ngắt PIC 8259.

2. Giao tiếp với thiết bị ngoại vi

2.1. Các kiểu giao tiếp nhập / xuất

2.1.1. Thiết bị ngoại vi có địa chỉ tách rời với bộ nhớ

Trong cách giao tiếp này, bộ nhớ dùng toàn bộ không gian 1 MB. Các thiết bị ngoại vi sẽ có một không gian 64 KB cho mỗi loại cổng. Trong kiểu giao tiếp này, ta phải dùng tín hiệu IO/$\overline{M}$ và các lệnh trao đổi dữ liệu thích hợp.

Bộ nhớ: IO/$\overline{M}$ = 0, dùng lệnh MOV

Ngoại vi: IO/$\overline{M}$ = 1, dùng lệnh IN (nhập) hay OUT (xuất)

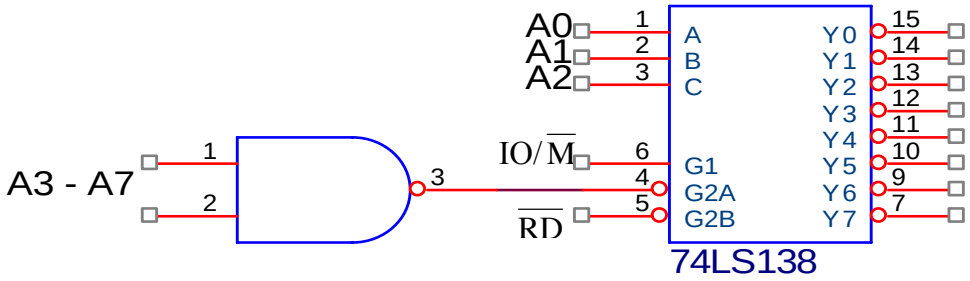
2.1.2. Thiết bị ngoại vi và bộ nhớ có chung không gian địa chỉ

Trong kiểu giao tiếp này, thiết bị ngoại vi sẽ chiếm một vùng nào đó trong không gian địa chỉ 1 MB và ta chỉ dùng lệnh MOV để thực hiện trao đổi dữ liệu.

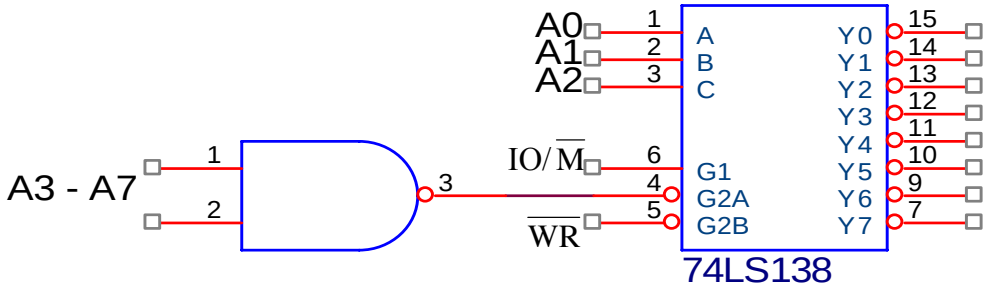
2.2. Giải mã địa chỉ cho thiết bị nhập / xuất

Việc giải mã địa chỉ cho thiết bị ngoại vi cũng tương tự với việc giải mã địa chỉ cho bộ nhớ. Thông thường, các cổng có địa chỉ 8 bit A0 – A7. Tuy nhiên, trong một số hệ vi xử lý, các cổng sẽ có địa chỉ 16 bit.

Ta có thể dùng mạch NAND để tạo tín hiệu chọn cổng nhưng mạch này chỉ có thể giải mã cho 1 cổng. Trong trường hợp cần nhiều tín hiệu chọn cổng, ta có thể dùng bộ giải mã 74LS138 để giải mã cho 8 cổng khác nhau.



(a) Giải mã cho cổng vào



(b) Giải mã cho cổng ra

Hình 3.4 – Giải mã cho các cổng

2.3. Các mạch cổng đơn giản

Các mạch cổng có thể được xây dựng từ các mạch chốt 8 bit (74LS373: kích theo mức, 74LS374: kích theo cạnh), các mạch đệm 8 bit (74LS245). Chúng được dùng trong các giao tiếp đơn giản để µP và ngoại vi hoạt động tương thích với nhau.

2.4. Giao tiếp nhập / xuất song song lập trình được 8255A PPI (Programmable Peripheral Interface)

2.4.1. Giới thiệu

8255A là thiết bị xuất nhập song song lập trình được. Nó là một thiết bị I/O đa dụng có thể sử dụng với bất cứ µP nào, có thể lập trình để truyền dữ liệu, từ I/O thông thường đến I/O interrupt.

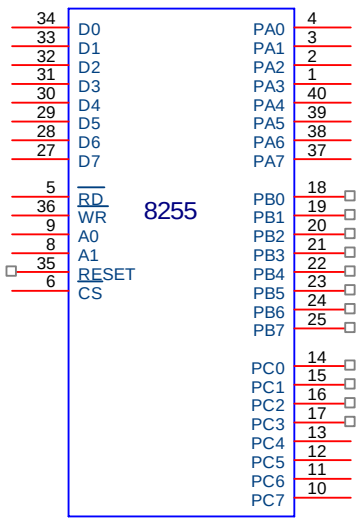
8255A có thể chia thành 3 Port: A, B và C; mỗi port 8 bit trong đó Port C có thể sử dụng như 8 bit riêng hay chia thành 2 nhóm, mỗi nhóm 4 bit: PCH (PC7 ÷ PC4) và PCL (PC3 ÷ PC0).

8255A có thể hoạt động ở 2 chế độ (mode): BSR (Bit Set/Reset) và I/O.

- **Chế độ BSR**: dùng để đặt hay xóa các bit của Port C.

- **Chế độ I/O**: gồm có 3 chế độ:

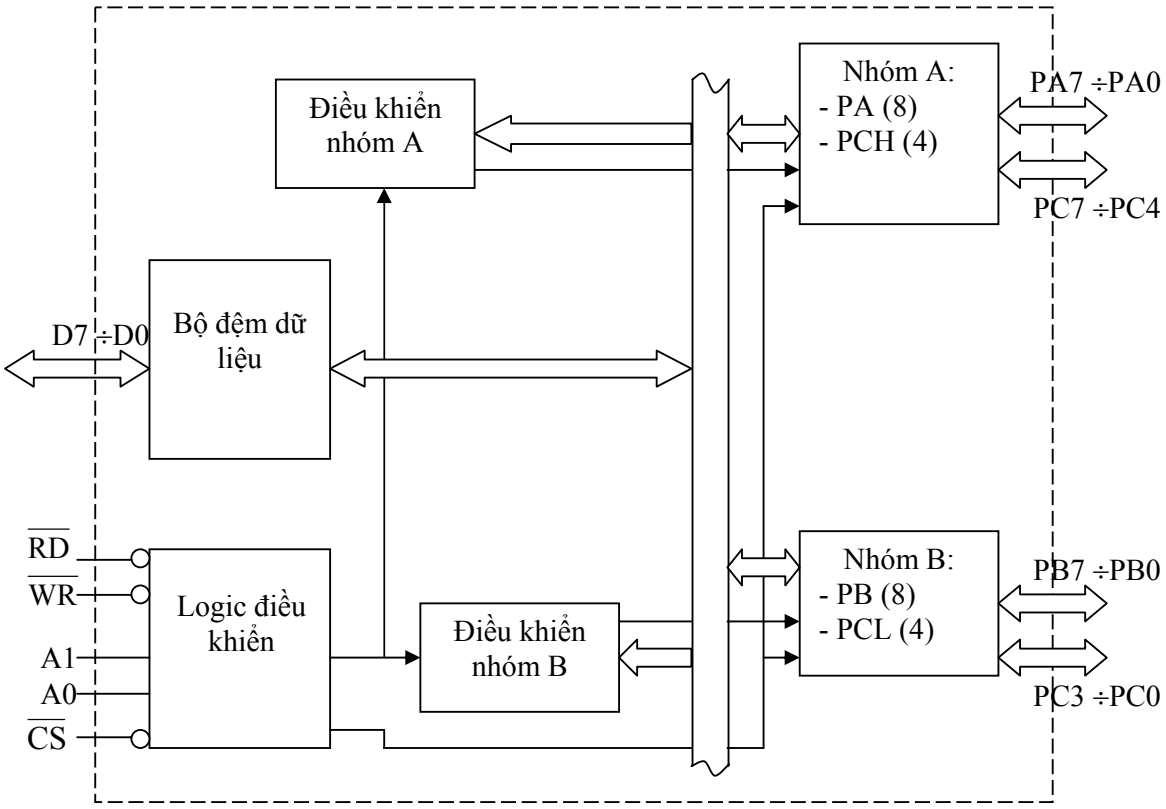
  - Chế độ 0: tất cả các Port làm việc như các Port I/O đơn giản.
  - Chế độ 1 (chế độ bắt tay: handshake): các Port A và B dùng các bit của Port C làm tín hiệu bắt tay. Trong chế độ này, các kiểu truyền dữ liệu I/O có thể được cài đặt, kiểm tra trạng thái và ngắt.
  - Chế độ 2: Port A có thể dùng để truyền dữ liệu song hướng dùng các tín hiệu bắt tay từ Port C còn Port B được thiết lập ở chế độ 0 hay 1.



D7 – D0: bus dữ liệu
PA7 – PA0: Port A
PB7 – PB0: Port B
PC7 – PC0: Port C
A1, A0: giải mã
RESET: ngõ vào Reset
$\overline{CS}$: Chip Select
$\overline{RD}$: Read
$\overline{WR}$: Write
VCC: +5V
GND: 0V

Hình 3.5 – Sơ đồ chân của 8255A

2.4.2. Sơ đồ khối



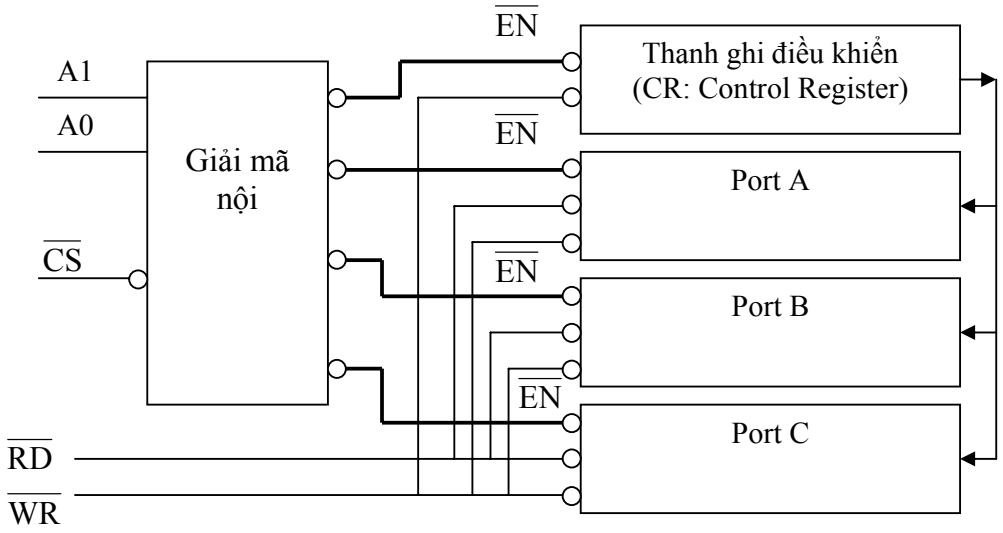
Hình 3.6 – Sơ đồ khối của 8255A

Logic điều khiển của 8255A gồm có 6 đường:

- $\overline{RD}$ (Read): cho phép ĐỌC. Khi chân này ở mức THẤP thì cho phép đọc dữ liệu từ Port I/O đã chọn.
- $\overline{WR}$ (Write): cho phép GHI. Khi chân này ở mức THẤP thì cho phép ghi dữ liệu ra Port I/O đã chọn.
- RESET: khi chân này ở mức cao thì sẽ xoá thanh ghi điều khiển và đặt các Port ở chế độ nhập.
- $\overline{CS}$ (Chip Select): chân chọn chip, thông thường $\overline{CS}$ được nối vào địa chỉ giải mã.
- A1, A0: giải mã xác định Port

Bảng 3.2:

$\overline{CS}$	A1	A0	Chọn
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Thanh ghi điều khiển
1	x	x	8255A không hoạt động

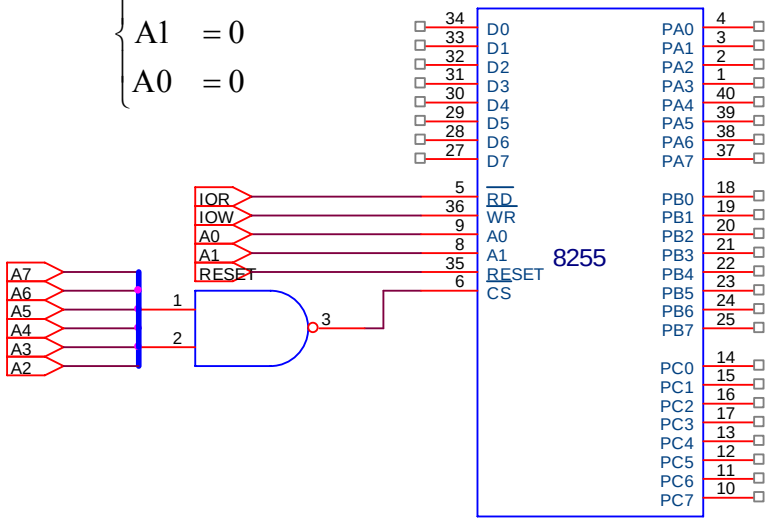


Hình 3.7 – Giải mã chọn các Port

Ví dụ: Xét sơ đồ kết nối 8255A như hình vẽ trang bên:

Theo bảng 3.2, để chọn Port A, ta phải có:

$$\begin{cases} \overline{CS} = 0 \\ A1 = 0 \\ A0 = 0 \end{cases}$$



Hình 3.8 – Logic chọn chip 8255A

Mà $\overline{CS}$ = 0 khi A7 = A6 = A5 = A4 = A3 = A2 = 1. Từ đó ta được địa chỉ Port I/O như sau:

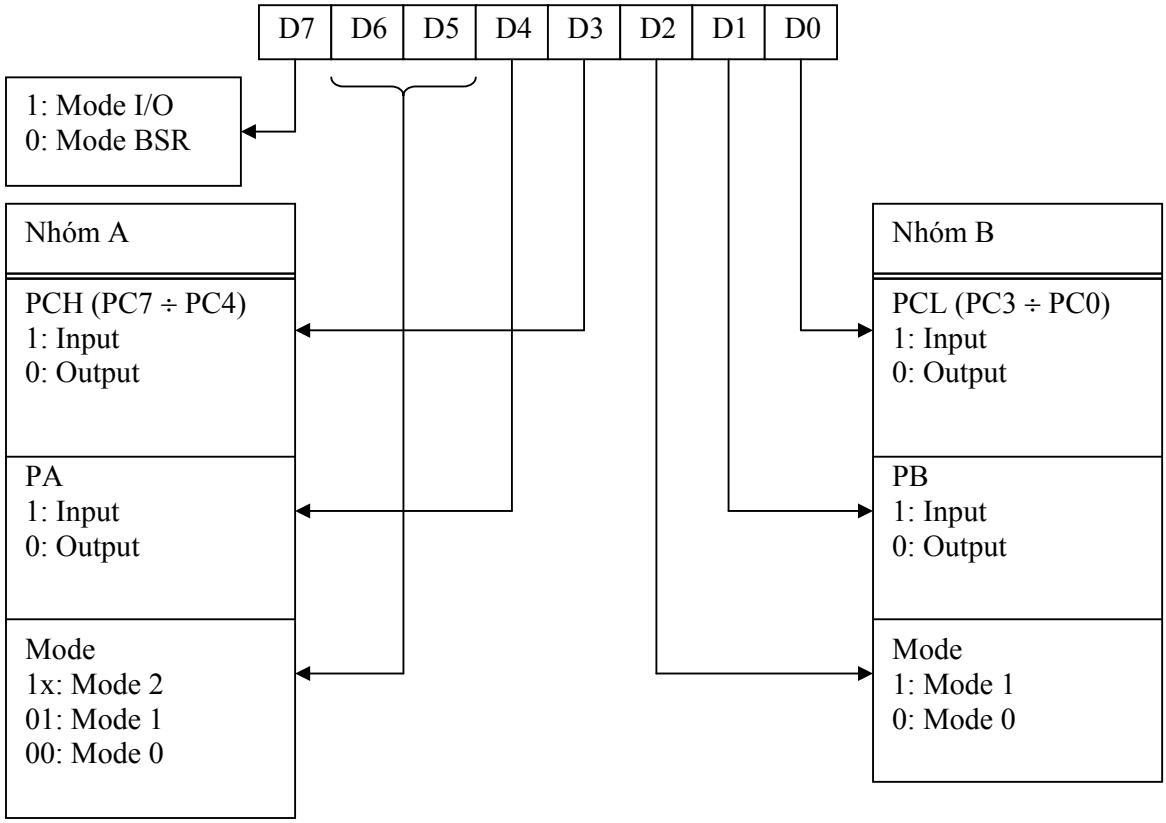
Bảng 3.3:

$\overline{CS}$						A1	A0	Port	Địa chỉ hex
A7	A6	A5	A4	A3	A2	A1	A0		
1	1	1	1	1	1	0	0	A	FCh
						0	1	B	FDh
						1	0	C	FEh
						1	1	CR	FFh

❖ **Thanh ghi điều khiển:**

Như đã biết, 8255A có 2 chế độ hoạt động và các Port của nó có thể có các chức năng I/O khác nhau. Để xác định chức năng của các Port, 8255A có một thanh ghi điều khiển (CR: Control Register). Nội dung của thanh ghi này gọi là từ điều khiển (CW: Control Word). Thanh ghi điều khiển sẽ được truy xuất khi A1 = A0 = 1. Chú ý rằng ta không thể thực hiện tác vụ Đọc đối với thanh ghi này.

Nếu bit D7 = 0, Port C làm việc ở chế độ BSR nhưng từ điều khiển BSR không ảnh hưởng đến chức năng các Port A, B.



Hình 3.9 – Dạng từ điều khiển cho 8255A ở chế độ I/O

2.4.3. Mode 0: Nhập / xuất đơn giản

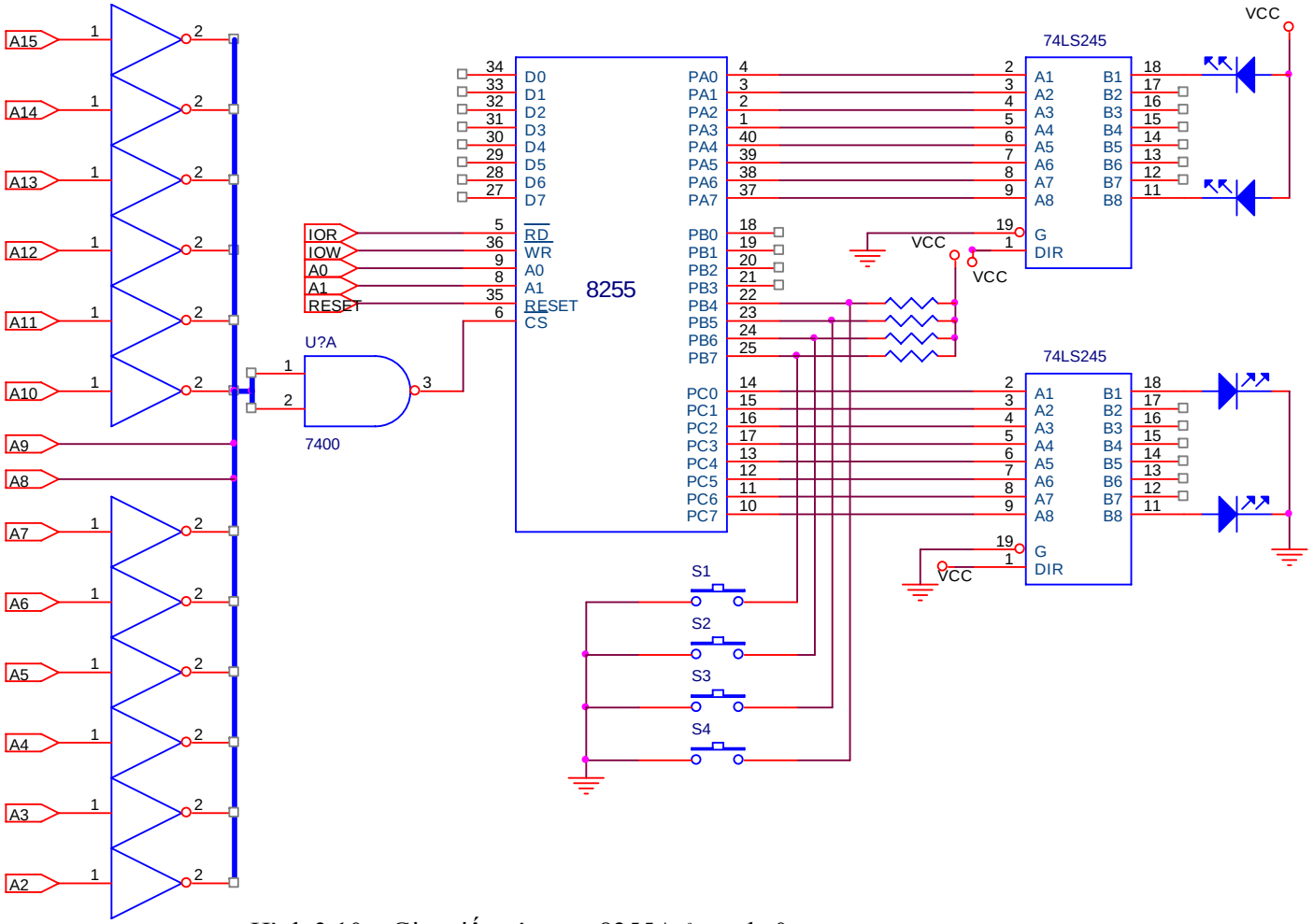
Trong chế độ này, mỗi port (hay nửa port của Port C) làm việc như các port nhập hay xuất với các tính chất sau:

- Các ngõ ra được chốt.
- Các ngõ vào không được chốt.
- Các port không có khả năng bắt tay và ngắt.

Để giao tiếp với ngoại vi thông qua 8255A cần phải:

- ***Xác định địa chỉ của các port A, B, C và CR thông qua các chân chọn chip $\overline{\text{CS}}$ và giải mã A1, A0.***
- ***Ghi từ điều khiển vào thanh ghi điều khiển.***
- ***Ghi các lệnh I/O để giao tiếp với ngoại vi qua các port A, B, C.***

Ví dụ: Xét sơ đồ kết nối 8255A như sau:



Hình 3.10 – Giao tiếp các port 8255A ở mode 0

- Xác định địa chỉ port:

Bảng 3.4:

$\overline{CS}$														A1	A0	Port	Địa chỉ hex
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0		
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	A	300h
														0	1	B	301h
														1	0	C	302h
														1	1	CR	303h

- Từ điều khiển:

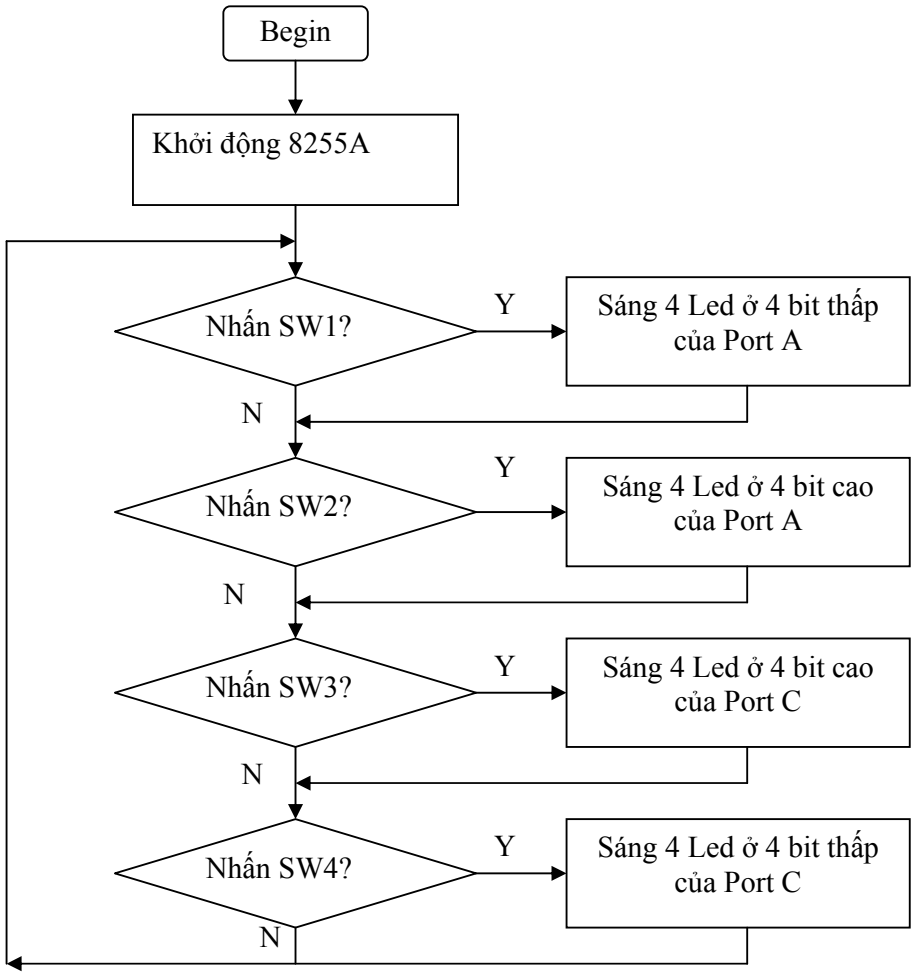
Bảng 3.5:

D7	D6	D5	D4	D3	D2	D1	D0	
1	0	0	0	0	0	1	0	= 82h
I/O mode	Nhóm A ở mode 0		PA: Output	PCH: Output	Nhóm B ở mode 0	PB: Input	PCL: Output	

Các Port của 8255A được khởi động bằng cách đặt từ điều khiển 82h vào thanh ghi điều khiển.

Trong sơ đồ kết nối này, 4 bit cao của Port B dùng làm Port nhập còn Port A và Port C làm Port xuất. Các tác vụ Đọc và Ghi được phân biệt bằng các tín hiệu điều khiển $\overline{IOR}$ và $\overline{IOW}$.

- Lưu đồ giải thuật:



- Chương trình:

```
        .MODEL    SMALL
        .STACK    100h
        .CODE
main    PROC
        ; ••nh c•u hình cho 8255
        MOV  AL,82h          ; T• •i•u khi•n (CW) là 82h
        MOV  DX,303h         ; ••a ch• thanh ghi
                             ; •i•u khi•n (CR)
        OUT  DX,AL           ; Ghi CW vào CR
cont:        MOV  DX,301h    ; ••a ch• Port B
        IN   AL,DX           ; ••c d• li•u t• Port B
                             ; (công t•c)
        AND  AL,0F0h         ; Che 4 bit th•p
        MOV  AH,AL
        CMP  AH,01110000b    ; Ki•m tra công t•c 1
        JNE  notSW1          ; N•u không nh•n
        MOV  AL,0Fh          ; N•u nh•n công t•c 1 thì
        MOV  DX,300h         ; xu•t ra Port A
        OUT  DX,AL           ; •• sáng 4 Led •
                             ; 4 bit th•p (Port A)

notSW1:  CMP  AH,10110000b   ; Ki•m tra công t•c 2
        JNE  notSW2          ; N•u không nh•n
```

```
          MOV  AL,0F0h          ; N•u nh•n công t•c 2 thì
          MOV  DX,300h          ; xu•t ra Port A •• sáng
          OUT  DX,AL            ; 4 Led • 4 bit cao (Port A)

notSW2:   CMP  AH,11010000b     ; Ki•m tra công t•c 3
          JNE  notSW3           ; N•u không nh•n
          MOV  AL,0Fh           ; N•u nh•n công t•c 3 thì
          MOV  DX,302h          ; xu•t ra Port C •• sáng 4
          OUT  DX,AL            ; Led • 4 bit cao (Port C)

notSW3:   CMP  AH,11100000b     ; Ki•m tra công t•c 4
          JNE  notSW4           ; N•u không nh•n
          MOV  AL,F0h           ; N•u nh•n công t•c 4 thì
          MOV  DX,302h          ; xu•t ra Port C •• sáng 4
          OUT  DX,AL            ; Led • 4 bit th•p (Port C)
notSW4:   JMP  cont
     main ENDP
     END  main
```

2.4.4. Mode BSR

Mode BSR chỉ liên quan đến 8 bit của Port C, có thể đặt hay xoá các bit bằng cách ghi một từ điều khiển thích hợp vào thanh ghi điều khiển. Một từ điều khiển với D7 = 0 gọi là từ điều khiển BSR, từ điều khiển này không làm thay đổi bất cứ từ điều khiển nào được truyền trước đó với D7 = 1, nghĩa là các hoạt động I/O của Port A và B không bị ảnh hưởng bởi từ điều khiển BSR.

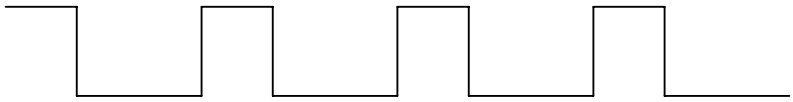
❖ **Từ điều khiển BSR:**

Từ điều khiển BSR khi được ghi vào thanh ghi điều khiển sẽ đặt hay xoá mỗi lần 1 bit.

D7	D6	D5	D4	D3	D2	D1	D0
0	x	x	X				S/R
Mode BSR	Không sử dụng			Chọn bit 000: PC0 001: PC1 010: PC2 011: PC3 100: PC4 101: PC5 110: PC6 111: PC7			0: Xoá (Reset) 1: Đặt (Set)

Ví dụ: Xét sơ đồ kết nối 8255A như hình 3.10. Giả sử ta cần tạo một sóng chữ nhật tại bit PC0.

Để tạo một sóng chữ nhật tại PC0, ta cần 2 mức logic là 0 và 1 tại PC0.



Bảng 3.6:

	D7	D6	D5	D4	D3	D2	D1	D0	
Đặt bit PC0 = 1	0	0	0	0	0	0	0	1	= 01h
Xoá bit PC0 = 0	0	0	0	0	0	0	0	0	= 00h

- Địa chỉ thanh ghi điều khiển (bảng 3.4): 303h

- Chương trình con:

```
bsr: MOV       AL,01h     ; T• •i•u khi•n BSR
     MOV       DX,303h    ; ••a ch• thanh ghi •i•u khi•n (CR)
     OUT       DX,AL      ; ••t PC0 = 1
     CALL      DELAY1     ; Ch•
     MOV       AL,00h     ; T• •i•u khi•n BSR
     OUT       DX,AL      ; Xóa PC0 = 0
     CALL      DELAY2     ; Ch•
     JMP       bsr
```

Khi sử dụng ở mode BSR, cần chú ý các điều sau:

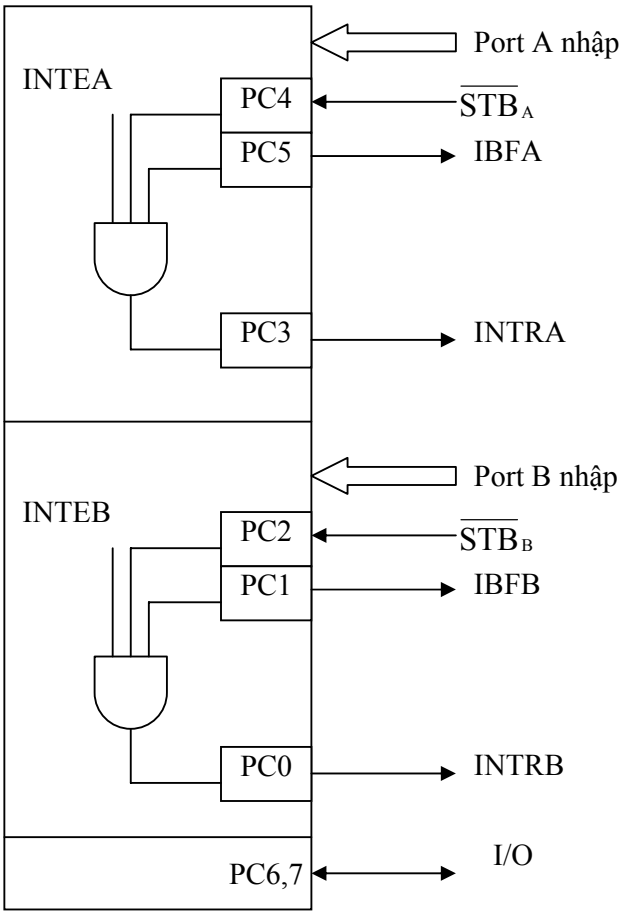
- ***Để đặt hay xoá các bit ở Port C, từ điều khiển được ghi vào thanh ghi điều khiển chứ không ghi vào Port C.***
- ***Một từ điều khiển BSR chỉ ảnh hưởng đến một bit của Port C.***
- ***Từ điều khiển BSR không ảnh hưởng đến I/O mode.***

2.4.5. Mode 1: Nhập / xuất với bắt tay (handshake)

Trong mode 1, các tín hiệu bắt tay được trao đổi giữa µP và thiết bị ngoại vi trước khi truyền dữ liệu. Các đặc tính ở chế độ này là:

- Hai Port A, B làm việc như các Port I/O 8 bit.
- Mỗi Port sử dụng 3 đường từ Port C làm các tín hiệu bắt tay. Hai đường còn lại có thể dùng cho các chức năng I/O đơn giản.
- Dữ liệu nhập / xuất được chốt.
- Hỗ trợ ngắt.

2.4.5.1. Các tín hiệu điều khiển nhập



Hình 3.11 – Cấu hình nhập của 8255A ở mode 1

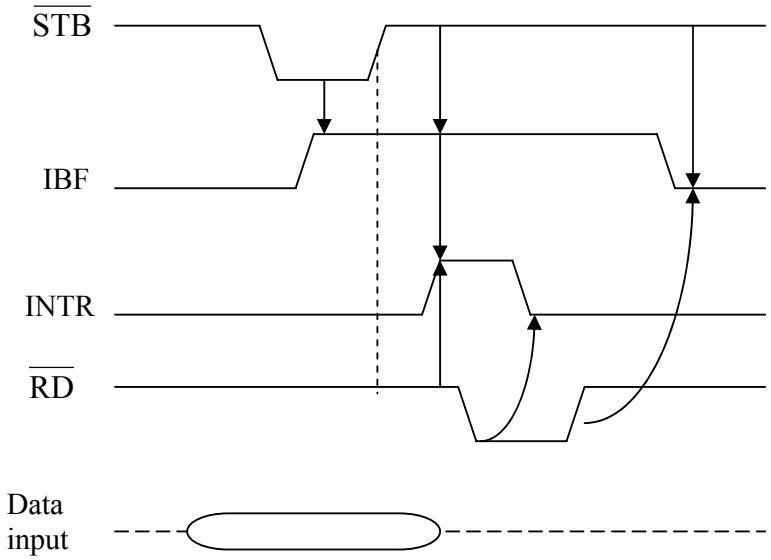
Theo hình vẽ, ta thấy Port A dùng 3 đường tín hiệu trên PC3, PC4 và PC5; Port B dùng 3 đường tín hiệu trên PC0, PC1 và PC2 làm các tín hiệu bắt tay. Các tín hiệu này có các chức năng sau khi các port A và B được đặt cấu hình là nhập:

- $\overline{STB}$ (Strobe Input): tích cực mức thấp, tín hiệu này được tạo bởi thiết bị ngoại vi để xác định rằng ngoại vi đã truyền 1 byte dữ liệu. Khi 8255A đáp ứng $\overline{STB}$, nó sẽ tạo ra IBF và INTR (hình 3.12).

- IBF (Input Buffer Full): tín hiệu này dùng để xác nhận 8255A đã nhận byte dữ liệu. Nó sẽ bị xoá khi μP đọc dữ liệu.

- INTR (Interrupt Request): Đây là tín hiệu xuất dùng để ngắt μP. Nó được tạo ra nếu $\overline{STB}$, IBF và INTE (flipflop bên trong) đều ở mức logic 1 và bị xoá bởi cạnh xuống của tín hiệu $\overline{RD}$ (Hình 3.12).

- INTE (Interrupt Enable): là một flipflop dùng để cho phép hay cấm quá trình tạo ra tín hiệu INTR. Hai flipflop INTEA và INTEB được đặt / xoá dùng BSR mode thông qua PC4 và PC2.



Hình 3.12 – Dạng sóng định thì cho ngõ vào có strobe

❖ **<u>Các từ điều khiển và trạng thái</u>:**

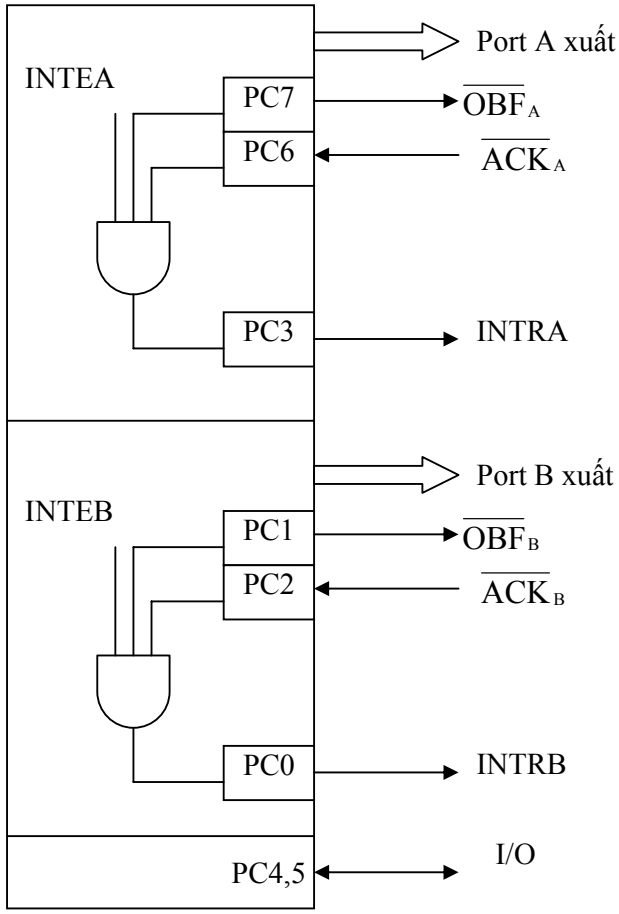
- Từ điều khiển: để xác định từ điều khiển, ta sử dụng hình 3.9

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	1/0	1	1	X
I/O mode	PA: Mode 1		PA: nhập	PC6,7 1: nhập 0: xuất	PB: Mode 1	PB: nhập	

- Từ trạng thái: sẽ được đặt trong thanh ghi tích luỹ nếu đọc Port C.

D7	D6	D5	D4	D3	D2	D1	D0
I/O	I/O	IBFA	INTEA	INTRA	INTEB	IBFB	INTRB

2.4.5.2. Các tín hiệu điều khiển xuất



Hình 3.13 – Cấu hình xuất của 8255A ở mode 1

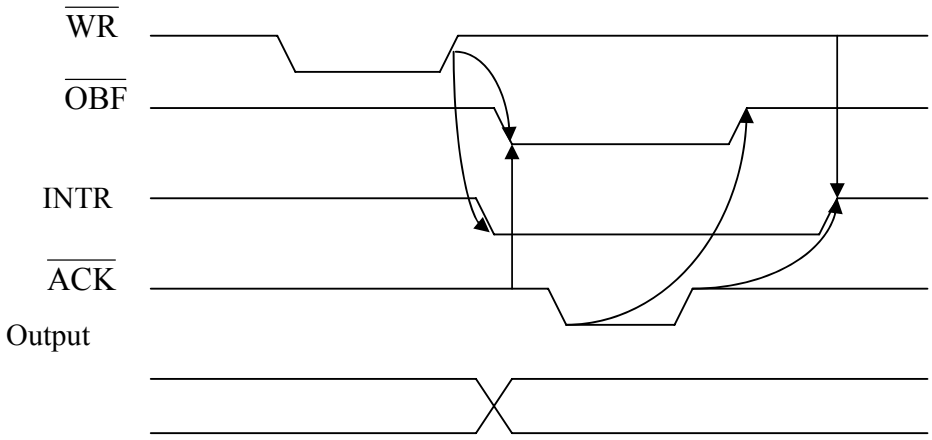
Chức năng các đường tín hiệu :

- $\overline{OBF}$ (Output Buffer Full): tín hiệu này sẽ xuống mức thấp khi μP ghi dữ liệu vào Port xuất của 8225A. Tín hiệu này đưa đến thiết bị ngoại vi để xác định dữ liệu sẵn sàng đưa vào ngoại vi (Hình 3.14). Nó sẽ lên mức cao khi 8255A nhận $\overline{ACK}$ từ ngoại vi.

- $\overline{ACK}$ (Acknowledge): đây là tín hiệu nhập từ ngoại vi (tích cực mức thấp) xác nhận dữ liệu đã nhập vào ngoại vi.

- INTR (Interrupt Request): đây là tín hiệu xuất, đặt bằng cạnh lên của tín hiệu $\overline{ACK}$. Tín hiệu này có thể dùng để ngắt μP yêu cầu byte dữ liệu kế tiếp để xuất. INTR được đặt khi $\overline{OBF}$, $\overline{ACK}$ và INTE ở mức logic 1 (Hình 4.14) và được xoá bởi cạnh xuống của tín hiệu $\overline{WR}$

- INTE (Interrupt Enable): đây là flipflop nội dùng để tạo tín hiệu INTR. Hai flipflop INTEA và INTEB điều khiển bằng các bit PC6 và PC2 thông qua BSR mode.



Hình 3.14 – Dạng sóng cho xuất strobe (có lấy mẫu) (với bắt tay)

❖ **Từ điều khiển và trạng thái:**

- Từ điều khiển:

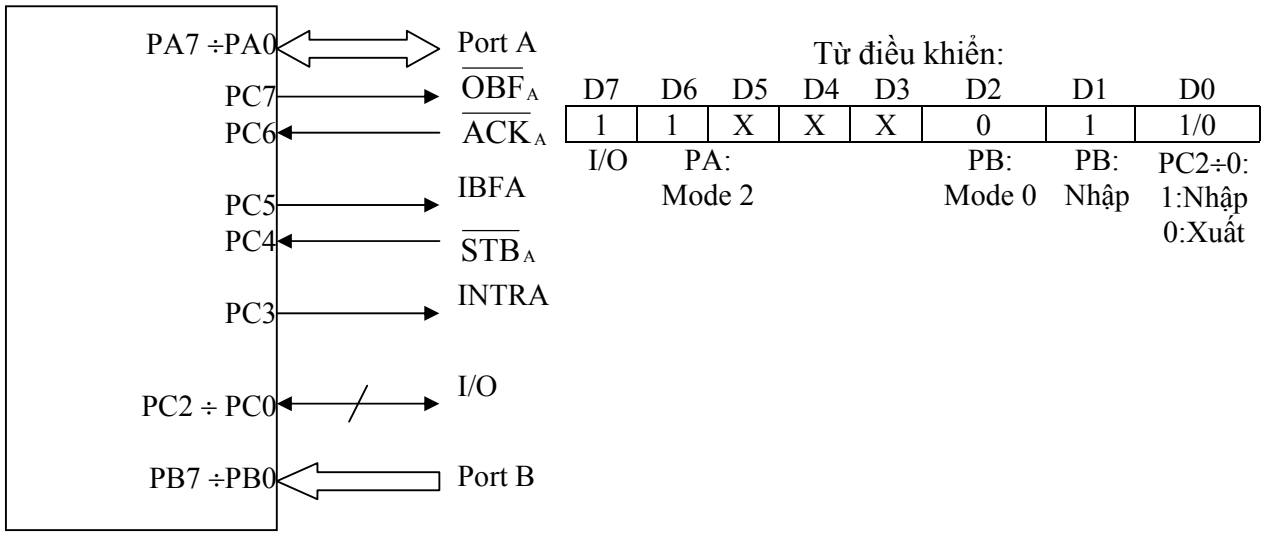
D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	0	1/0	1	0	X
I/O mode	PA: Mode 1		PA: xuất	PC4,5 1: nhập 0: xuất	PB: mode 1	PB: xuất	

- Từ trạng thái:

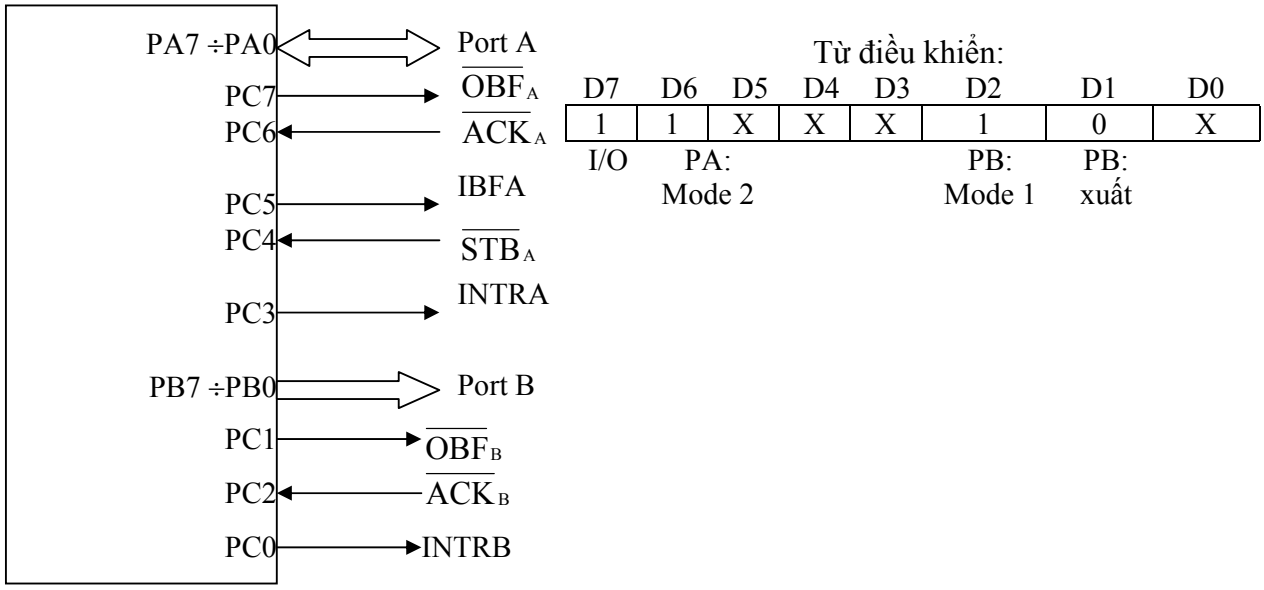
D7	D6	D5	D4	D3	D2	D1	D0
$\overline{OBF}_A$	INTEA	I/O	I/O	INTRA	INTEB	$\overline{OBF}_B$	INTRB

2.4.6. Mode 2: Truyền dữ liệu song hướng

Mode nay dùng chủ yếu trong các ứng dụng như truyền dữ liệu giữa hai máy tính hay giao tiếp bộ điều khiển đĩa mềm. Trong mode này, Port A dùng làm Port song hướng và Port B làm việc ở Mode 0 hay 1. Port A sử dụng 5 tín hiệu tại Port C làm các tín hiệu điều khiển để truyền dữ liệu. Ba tín hiệu còn lại của Port C được dùng làm I/O đơn giản hay bắt tay cho Port B.



(a) 8255A ở mode 2 và mode 0 (nhập)



(a) 8255A ở mode 2 và mode 1 (xuất)

Hình 3.15 – 8255A dùng ở Mode 2

2.4.7. Các ví dụ minh họa

2.4.7.1. Giao tiếp với bộ chuyển đổi A/D ADC0804 dùng 8255A ở Mode 0 và Mode BSR

Ta thiết lập 8255A hoạt động như sau:

- Dùng Port A để đọc dữ liệu.

- Dùng PC0, PC3 điều khiển các chân $\overline{RD}$, $\overline{WR}$ của ADC0804.

Xét sơ đồ mạch có logic chọn chip giống như hình 4.10. Tầm địa chỉ Port từ 300h ÷ 303h.

- **Từ điều khiển mode 0:**

Port A: nhập

Pot B: không sử dụng

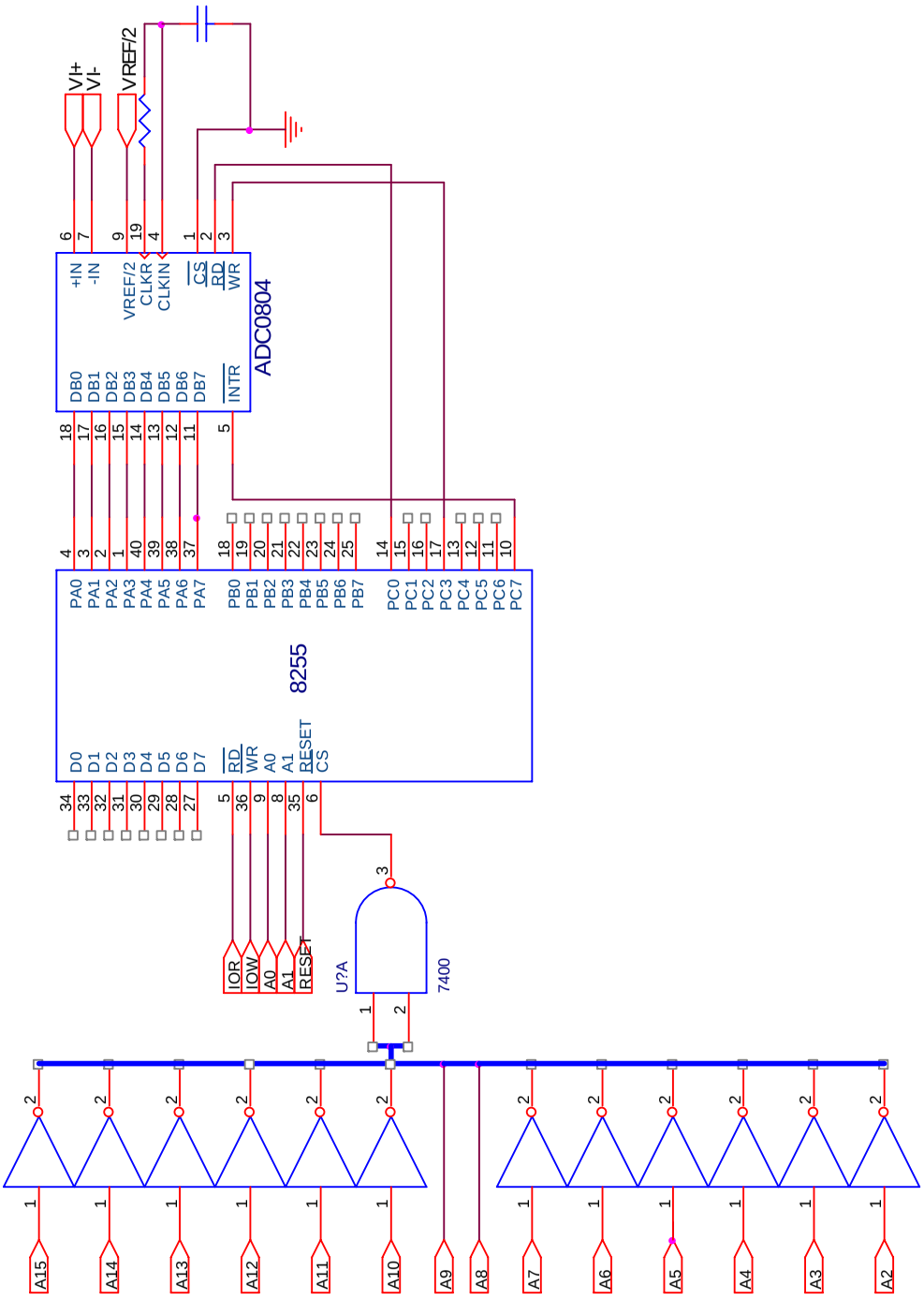
Port Clow: port xuất dùng để điều khiển 2 ngõ $\overline{RD}$, $\overline{WR}$ của ADC0804

Port Chigh: port nhập dùng để đọc trạng thái ở chân $\overline{INTR}$ của ADC0804

D7	D6	D5	D4	D3	D2	D1	D0	
1	0	0	1	0	0	0	0	= 90h
I/O	PA: mode 0		PA: nhập	PCH: xuất	PB: không sử dụng		PCL: xuất	

- **Từ điều khiển BSR:**

	D7	D6	D5	D4	D3	D2	D1	D0	
Đặt PC0	0	0	0	0	0	0	0	1	= 01h
Xoá PC0	0	0	0	0	0	0	0	0	= 00h
Đặt PC3	0	0	0	0	0	1	1	1	= 07h
Xoá PC3	0	0	0	0	0	1	1	0	= 06h



Hình 3.16 – Giao tiếp bộ chuyển đổi A/D ADC0804 dùng 8255A

- Mô tả chương trình:

  - Khởi động 8255A bằng cách đặt từ điều khiển mode 0 vào thanh ghi điều khiển.
  - Cấp một xung vào chân $\overline{RD}$ của 8255A.
  - Đọc trạng thái của ADC0804 từ chân $\overline{INTR}$.
  - Nếu $\overline{INTR}$ = 0 thì cấp một xung vào chân $\overline{WR}$ của ADC0804 để xuất dữ liệu.
  - Đọc dữ liệu từ ADC0804 vào thông qua Port A.

- Đoạn chương trình thực hiện:

```
adc: MOV       DX,303h   ; ••a ch• thanh ghi •i•u khi•n (CR)
     MOV       AL,90h    ; T• •i•u khi•n (CW)
     OUT       DX,AL     ;Ghi CW vào CR
     MOV       AL,01h    ; T• •i•u khi•n BSR •• PC0 = 1 (RD =
1)
     OUT       DX,AL     ; Xu•t ra CR
     MOV       AL,07h    ; T• •i•u khi•n BSR •• PC3 = 1
     OUT       DX,AL     ; Xu•t ra CR
     MOV       AL,06h    ; T• •i•u khi•n BSR •• PC3 = 0, t•o
                         ; xung WR
     OUT       DX,AL     ; Xu•t ra CR
     CALL      DELAY     ; Ch• quá trình chuy•n ••i th•c hi•n
xong
     MOV       AL,07h    ; T• •i•u khi•n BSR •• PC3 = 1
     OUT       DX,AL     ; Xu•t ra CR
     MOV       DX,300h   ; ••a ch• Port A
     IN        AL,DX     ; ••c d• li•u •ã chuy•n ••i t• ADC0804
     MOV       AL,01h    ; T• •i•u khi•n BSR •• PC0 = 1
                         ; (RD = 1)
     OUT       DX,AL     ; Xu•t ra CR
     RET                 ; vào t• Port A c•a 8255A
```

2.4.7.2. Giao tiếp với máy in trong chế độ bắt tay (Mode 1)

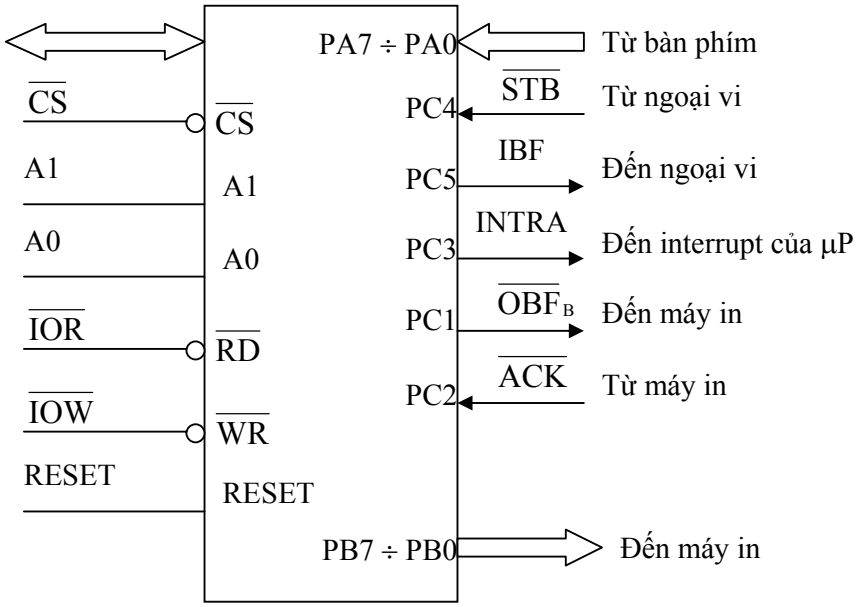
Xét mạch giao tiếp 8255A ở mode 1 với Port A được dùng làm Port nhập từ bàn phím với I/O interrupt và Port B được thiết kế làm Port xuất tới máy in với I/O kiểm tra trạng thái. Ta cần thực hiện các công việc sau:

- Xác định địa chỉ Port.
- Xác định từ điều khiển để Port A nhập và Port B xuất ở Mode 1.
- Xác định từ điều khiển BSR cho phép ngắt (INTEA).
- Xác định các byte mặt nạ để kiểm tra các đường $\overline{OBF_B}$ trong I/O kiểm tra trạng thái.
- Viết các lệnh khởi động và chương trình con in các ký tự chứa trong bộ nhớ.

Giả sử logic chọn chip như hình 3.10, địa chỉ Port cho trong bảng 3.4:

PA: FCh

PB: FDh
PC: FEh
CR: FFh



Hình 3.17 – Giao tiếp 8255A ở Mode 1

- **Từ điều khiển:** Port A nhập, Port B xuất ở Mode 1

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	0	1	0	0
I/O	PA: Mode 1		PA: nhập	Không sử dụng	PB: Mode 1	PB: xuất	Không sử dụng

= B4h

- **Từ điều khiển BSR**: dùng để đặt flipflop cho phép ngắt của Port A (INTEA), bit PC4 = 1

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	0	0	1
BSR mode	Không sử dụng			Bit PC4		Đặt bit (Set)	

= 09h

- **Từ trạng thái kiểm tra $\overline{OBF}_B$:**

D7	D6	D5	D4	D3	D2	D1	D0
X	x	x	x	x	x	$\overline{OBF}_B$	X

Byte mặt nạ: 0000 0010b

❖ **Khởi động:**

```
MOV     DX, 0FFh ; Kh•i ••ng 8255A
MOV     AL, 0B4h ; • Mode 1, Port A nh•p
OUT     DX, AL   ; Port B xu•t
```

```
     MOV       AL, 09h   ; ••t INTEA
     OUT       DX, AL    ; cho phép INTRA
     CALL      print
```

❖ **Chương trình con PRINT:**

```
print:
          LEA       DX,msg    ; Ch• ••n v• trí
                              ; ch•a các ký t•
          MOV       SI, DX
          ADD       SI,2
next:
          LODSB               ; L•y ký t• t• b• nh•
          CMP       AL,0      ; N•u không còn ký t• nào
          JNE       cont      ; thì k•t thúc
          JMP       exit
cont:
          MOV       AH,AL     ; L•u ký t• v•a ••c
          MOV       DX,0FEh
status:
          IN        AL,DX     ; ••c vào t• Port C
          AND       AL,02h    ; Ch• nh•n PC1
          JE        status    ; N•u máy in không
                              ; s•n sàng thì ch•
          MOV       AL,AH
          MOV       DX,0FDh   ; Xu•t ký t• •ã nh•n ra
          OUT       DX,AL     ; máy in (Port B)
          JMP       next      ; X• lý ký t• k• ti•p
exit:
          RET
```

❖ **Mô tả chương trình:**

- Ta sử dụng 8255A trong phần thiết kế này cho phép 2 hoạt động: xuất ra máy in và lấy dữ liệu vào từ bàn phím. Giao tiếp với máy in dùng kiểm tra trạng thái và giao tiếp bàn phím dùng ngắt.

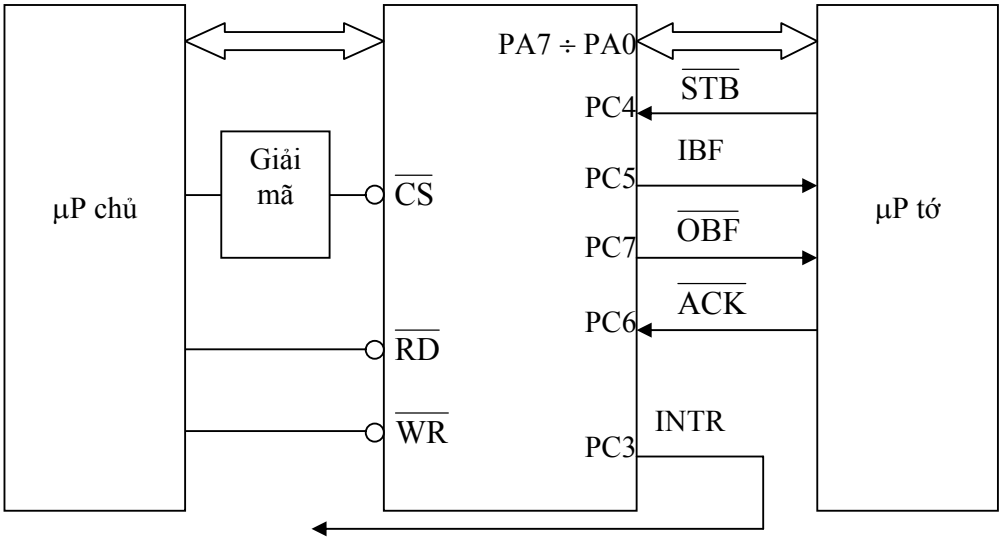
- Trong chương trình con PRINT, ký tự được đặt trong thanh ghi tích luỹ A và trạng thái đọc từ Port C. Ban đầu Port B trống, bit PC1 ($\overline{OBF_B}$) ở mức cao. Ta thực hiện lệnh OUT gởi dữ liệu ra Port B. Tín hiệu $\overline{OBF_B}$ sẽ xuống mức thấp do tác động cạnh lên của tín hiệu $\overline{WR}$, xác định rằng dữ liệu đã gởi ra máy in. Sau khi nhận byte dữ liệu, máy in gởi trở lại tín hiệu $\overline{ACK}$ xác định đã nhận. Tín hiệu $\overline{ACK}$ làm cho $\overline{OBF_B}$ ở mức cao xác định máy in sẵn sàng nhận ký tự kế tiếp và chương trình con PRINT tiếp tục thực hiện cho đến khi không còn ký tự nào trong vùng nhớ.

- Nếu một phím được nhấn khi chương trình con PRINT đang thực thi, byte dữ liệu truyền tới Port A và $\overline{STB_A}$ xuống mức thấp, đặt IBFA lên mức cao. Khi $\overline{STB_A}$ trở lại mức cao thì sẽ tạo ra INTRA. Tín hiệu này tạo ngắt đến μP và điều khiển được chuyển đến chương trình phục vụ

ngắt. Chương trình này sẽ đọc nội dung Port A, cho phép ngắt và quay về chương trình con PRINT.

2.4.7.3. Truyền dữ liệu giữa hai microprocessor trong xử lý phân bố dùng 8255A ở Mode 2

Ta thiết kế mạch giao tiếp để truyền dữ liệu hai chiều dạng chủ – tớ (master – slave) giữa hai µP.



Hình 3.18 – Thông tin 2 chiều giữa 2 µP dùng 8255A

Hình 3.18 chỉ sơ đồ khối thiết lập thông tin hay chiều giữa chủ và tớ . Sơ đồ khối chỉ hai data bas hai chiều – chủ và tớ – được nối với nhau thông qua 8225A, trong đó 8225A làm việc như thiết bị giao tiếp của µP chủ. Port A của 8225A được dùng để truyền dữ liệu hai chiều và 4 tín hiệu từ port C được dùng để bắt tay. Quá trình truyền dữ liệu tương tự như Mode 1 của 8225A. Khi µP chủ ghi 1 byte dữ liệu vào 8225A tín hiệu $\overline{OBF}$ xuống mức thấp để báo cho µP tớ biết là đã gởi dữ liệu vào, µP tớ sẽ báo nhận được khi nó đọc byte dữ liệu này. Tương tự, hai tín hiệu bắt tay khác được dùng khi µP tớ truyền 1 byte dữ liệu đến µP chủ.

µP chủ đòi hỏi các port I/O dùng để đọc và ghi dữ liệu và kiểm tra trạng thái của các tín hiệu bắt tay. Tương tự, µP tớ cần các port I/O để thực hiện Đọc và Ghi. Truyền dữ liệu có thể được thực hiện bằng cách kiểm tra trạng thái hay dùng ngắt. Tốc độ xử lý dữ liệu đối với µP chủ quan trọng hơn nên thường dùng µP chủ ở chế độ ngắt và µP tớ ở chế độ kiểm tra trạng thái. Ở ví dụ này, ta sẽ dùng cả 2 µP ở chế độ kiểm trạng thái.

Các hoạt động truyền dữ liệu giữa 2 I/O kiểm tra trạng thái có thể liệt kê như sau:

❖ **Truyền dữ liệu từ µP chủ đến µP tớ:**

1. µP chủ đọc trạng thái của $\overline{OBF}$ để kiểm tra xem µP tớ đã đọc dữ liệu chưa. Đây là chức năng nhập cho µP chủ.

2. µp chủ ghi dữ liệu vào Port A và 8225A báo cho µP tớ biết bằng cách đưa tín hiệu $\overline{OBF}$ xuống mức thấp. Đây là chức năng xuất của µP chủ.

3. µP tớ kiểm tra tín hiệu $\overline{OBF}$ (từ µP chủ) để xác định tính sẵn sàng của dữ liệu. Đây là chức năng nhập đối với µP tớ.

4. µP tớ đọc dữ liệu từ Port A và báo cho biết đã nhận được bằng cách đưa tín hiệu $\overline{ACK}$ xuống mức thấp . Đây là chức năng nhập đối với µP tớ.

❖ **Truyền dữ liệu từ µP tớ dến µp chủ:**

1. µP tớ kiểm tra tín hiệu bắt tay IBF để xem port A có sẵn sàng truyền dữ liệu hay không để truyền 1 byte . Đây là chức năng nhập đối với µP tớ .

2. µP đặt byte dữ liệu lên data bus và báo cho 8225A biết rằng sẵn sàng gởi dữ liệu bằng cách dùng tín hiệu $\overline{STB}$. Đây là chức năng xuất đối vói µP tớ.

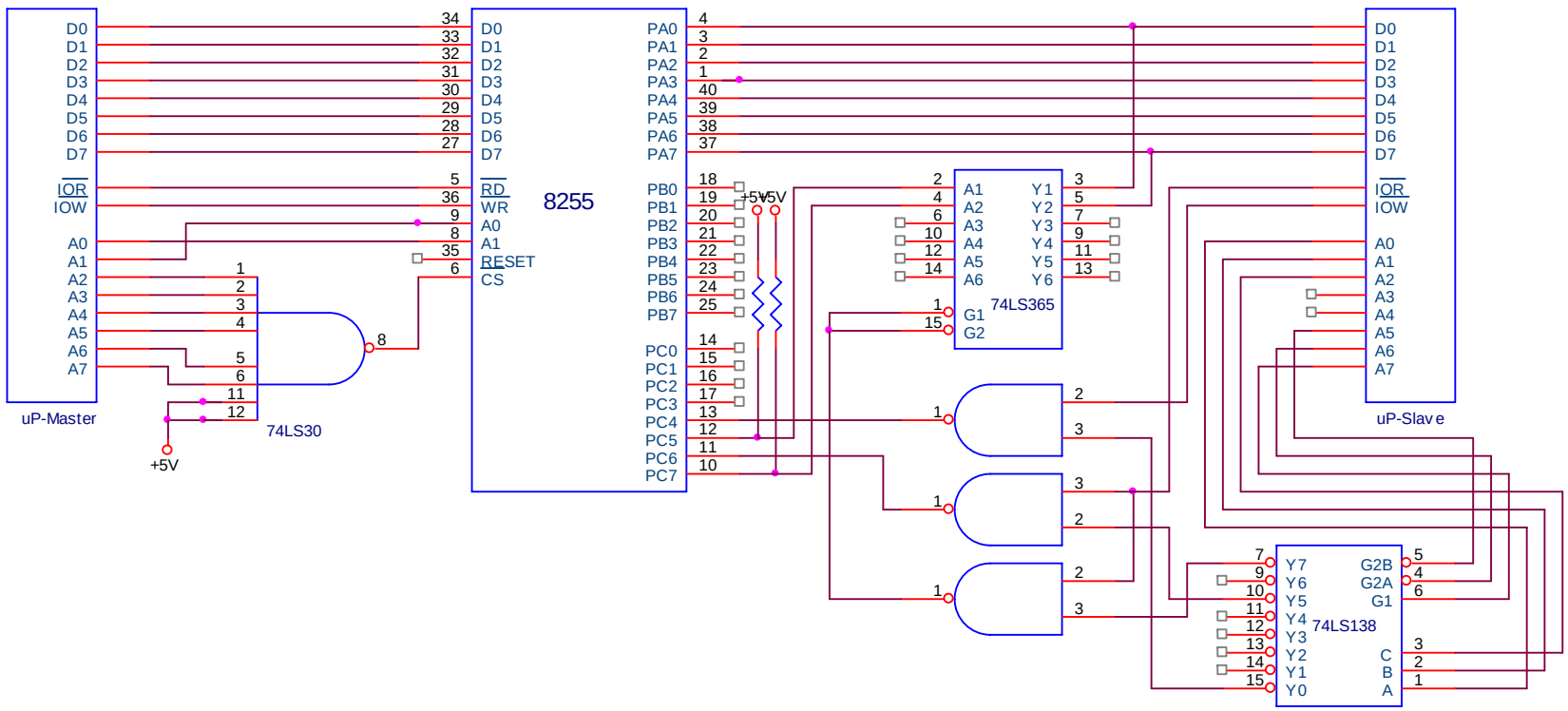
3. 8225A đưa IBF lên mức cao, µP chủ đọc tín hiệu này để xác định dữ liệu sẵn sàng chưa . Đây là chức năng nhập đối với µP chủ .

4. µP chủ đọc byte dữ liệu . Đây là chức năng nhập đối với µP chủ.

❖ **Kết nối phần cứng:**

Hình 3.19 cho thấy sơ đồ kết nối các port cần thiết và logic chọn chip cho 8255A. µP chủ thực hiện giải mã chọn 8255A dùng cổng NAND 8 ngõ vào nên 8255A được chọn khi tất cả các ngõ vào của cổng NAND đều ở mức 1. Từ đó, ta có các địa chỉ Port của 8255A đối với µP chủ là:

PA: FCh
PB: FDh
PC: FEh
CR: FFh



Hình 3.19 – Thông tin hai chiều giữa µP chủ và µP tớ

Port A được sử dụng ở Mode 2 dùng 4 tín hiệu từ Port C. µP chủ kiểm tra các tín hiệu $\overline{ACK}$ và $\overline{STB}$ bằng cách đọc các bit trạng thái $\overline{OBF}$ và IBF ở Port C.

Hai tín hiệu bắt tay khác - $\overline{OBF}$ và IBF – được nối tương ứng với các bit D7 và D0 của data bus của µP tớ thông qua bộ đệm 3 trạng thái 74LS365. Logic giải mã cho các đường tín hiệu tại Port C chính là bộ giải mã 3 sang 8 74LS138. Giả sử các đường logic không sử dụng (A3 và A4) ở mức 0, 8 đường ra của bộ giải mã sẽ cho phép vùng địa chỉ 80h ÷ 87h (Bảng 3.6). Hai đường ra của bộ giải mã được kết hợp với tín hiệu điều khiển $\overline{IOR}$ để tạo ra 2 xung chọn thiết bị nhận (85h và 87h). Xung chọn thiết bị nhập 87h được dùng để đọc trạng thái ở các đường dữ liệu D7 và D0. Đường giải mã có địa chỉ 80h được kết hợp với $\overline{IOW}$ để tạo tín hiệu $\overline{STB}$.

Bảng 3.7:

A7	A6	A5	A4	A3	A2	A1	A0	Chân giải mã	Địa chỉ hex
1	0	0	0	0	0	0	0	Y0	80h
					0	0	1	Y1	81h
					0	1	0	Y2	82h
					0	1	1	Y3	83h
					1	0	0	Y4	84h
					1	0	1	Y5	85h
					1	1	0	Y6	86h
					1	1	1	Y7	87h

❖ **Từ điều khiển mode 2:**

D7	D6	D5	D4	D3	D2	D1	D0	
1	1	0	0	0	0	0	0	= C0h
I/O	Mode 2			Không sử dụng				

❖ **Từ trạng thái mode 2:**

Trạng thái của hoạt động I/O ở Mode 2 có thể kiểm rta bằng cách đọc nội dung Port C.

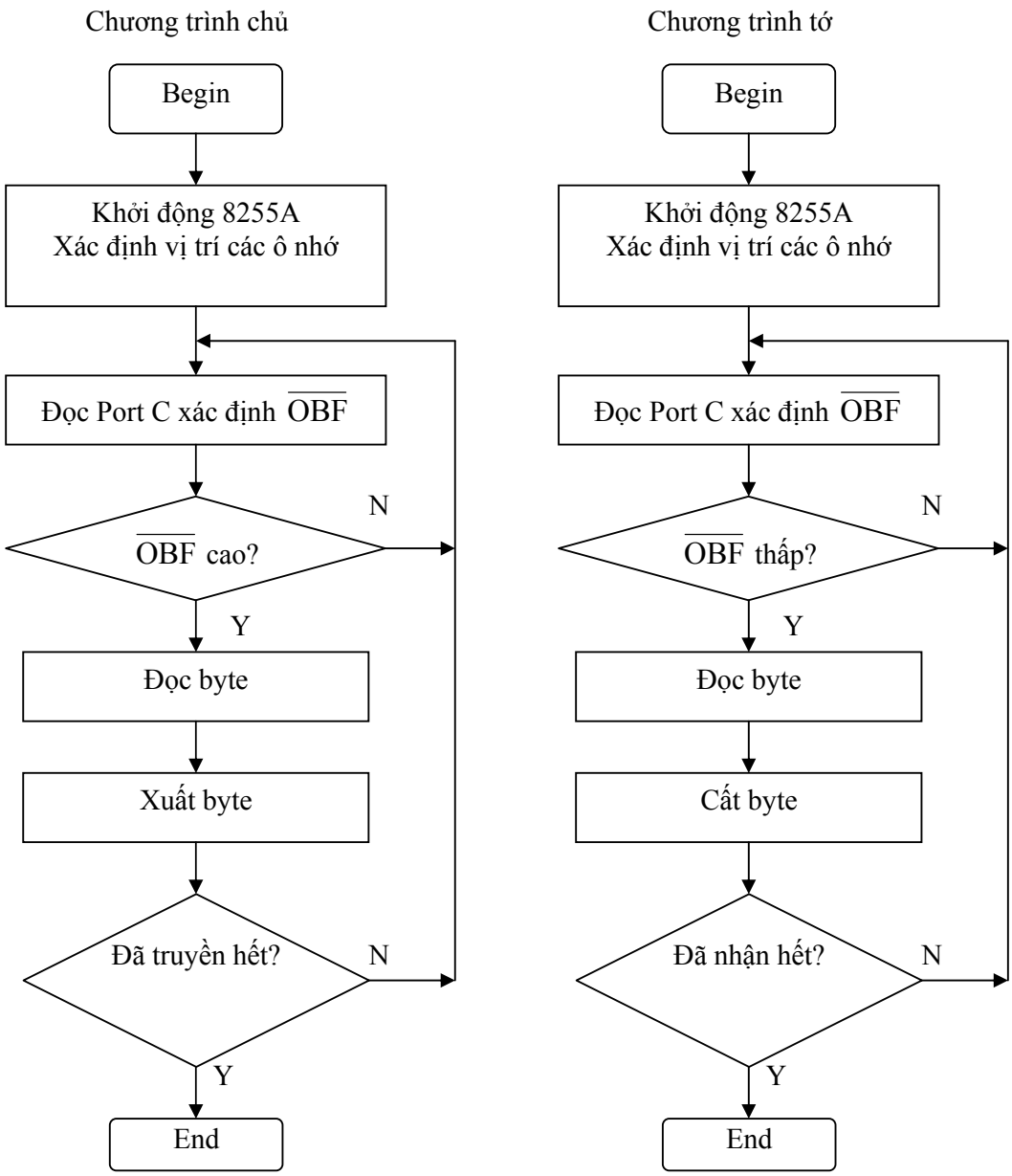
D7	D6	D5	D4	D3	D2	D1	D0
$\overline{OBF}_A$	$INTE_1$	IBFA	$INTE_2$	INTRA	X	X	X

Trạng thái của tín hiệu $\overline{OBF}$ được kiểm tra bằng cách đọc bit D7 và trang thái của IBF kiểm tra bằng bit D0.

❖ **Các tác vụ Đọc và Ghi của µP tớ:**

Một byte dữ liệu có thể được đọc bởi µP tớ từ Port A bằng cách gởi một xung chọn thiết bị tác động mức thấp đến tín hiệu $\overline{ACK}$, không cần xây dựng Port nhập. Tương tự, một byte dữ liệu có thể được ghi vào µP bằng cách đưa tín hiệu $\overline{STB}$ xuống thấp.

❖ **Lưu đồ giải thuật:**



❖ **Chương trình:**

➢ **Đoạn chương trình chủ: (Master program)**

```
        MOV     SP,stack1
        MOV     SI,master       ; ••a ch• các byte
                                ; c•n xu•t
        MOV     CX,byte_no      ; S• byte c•n xu•t
        MOV     AL,0C0h         ; T• •i•u khi•n
        MOV     DX,0FFh         ; ••a ch• thanh ghi
                                ; •i•u khi•n
        OUT     DX,AL
next:
        MOV     DX,0FEh         ; ••a ch• Port C
wait:
        IN      AL,DX           ; ••c vào t• Port C
        AND     AL,80h          ; Ki•m tra OBF
        JNE     wait            ; Ch• ••n khi OBF = 0
        LODSB                   ; ••c byte
        MOV     DX,0FCh         ; Xu•t byte v•a ••c
        OUT     DX,AL           ; ra Port A
        LOOP    next            ; N•u còn byte truy•n
                                ; thì ti•p t•c
```

➢ **Đoạn chương trình tớ: (Slave program)**

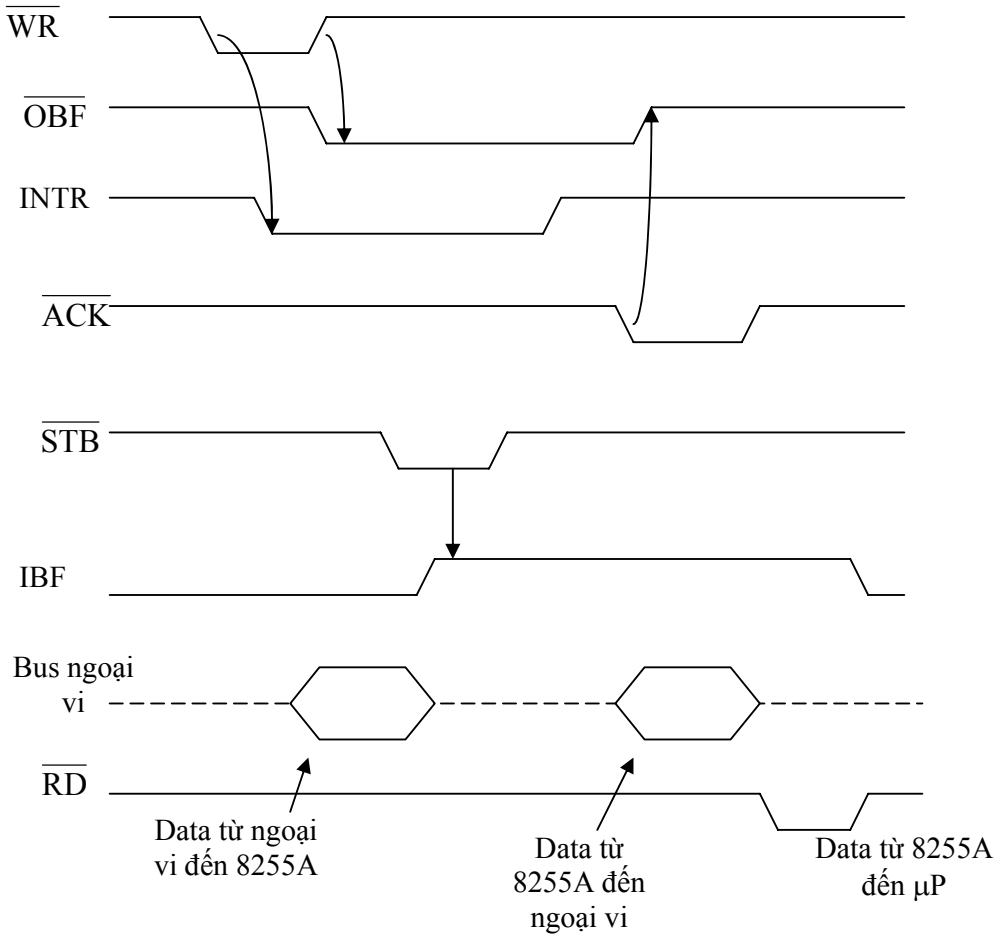
```
        MOV     ES,stack2
        MOV     DI,slave        ; ••a ch• các byte s• l•u
        MOV     CX,byte_no      ; S• byte c•n nh•n
next:
        MOV     DX,87h
wait:
        IN      AL,DX           ; ••c OBF
        AND     AL,80h          ; Ki•m tra OBF
        JE      wait            ; Ch• ••n khi OBF = 1
        MOV     DX,85h
        IN      AL,DX           ; ••c d• li•u
        STOSB                   ; C•t vào ô nh•
        LOOP    next            ; N•u còn byte truy•n
                                ; thì ti•p t•c
```

- Ta thấy rằng cả hai chương trình sẽ kiểm tra trạng thái $\overline{OBF}$. Chương trình chủ đợi cho đến khi $\overline{OBF}$ lên mức cao sẽ ghi một byte vào Port A. Ngược lại, chương trình tớ đợi cho đến khi $\overline{OBF}$ xuống mức thấp thì sẽ đọc dữ liệu.

- Khi μP chủ ghi một byte dữ liệu, nó sẽ chốt tại Port A và byte dữ liệu được đặt trên data bus của μP tớ khi $\overline{ACK}$ xuống mức thấp.

- Hai chương trình trên chỉ cho phép truyền một khối dữ liệu từ µP chủ đến µP tớ nhưng không thể truyền ngược lại. Để chuyển một khối dữ liệu từ µP tớ đến µP chủ, cần phải đọc tín hiệu IBF. µP chủ đợi cho đến khi IBF = 1 thì sẽ đọc một byte dữ liệu còn µP tớ đợi cho đến khi IBF = 0 thì ghi một byte dữ liệu.

- Giản đồ thời gian ở hình 3.20 cho thấy tín hiệu INTR dùng để truyền dữ liệu bằng ngắt. Trong ví dụ này, ta không sử dụng ngắt.



Hình 3.20 – Giản đồ thời gian ở Mode 2

BÀI TẬP CHƯƠNG 3

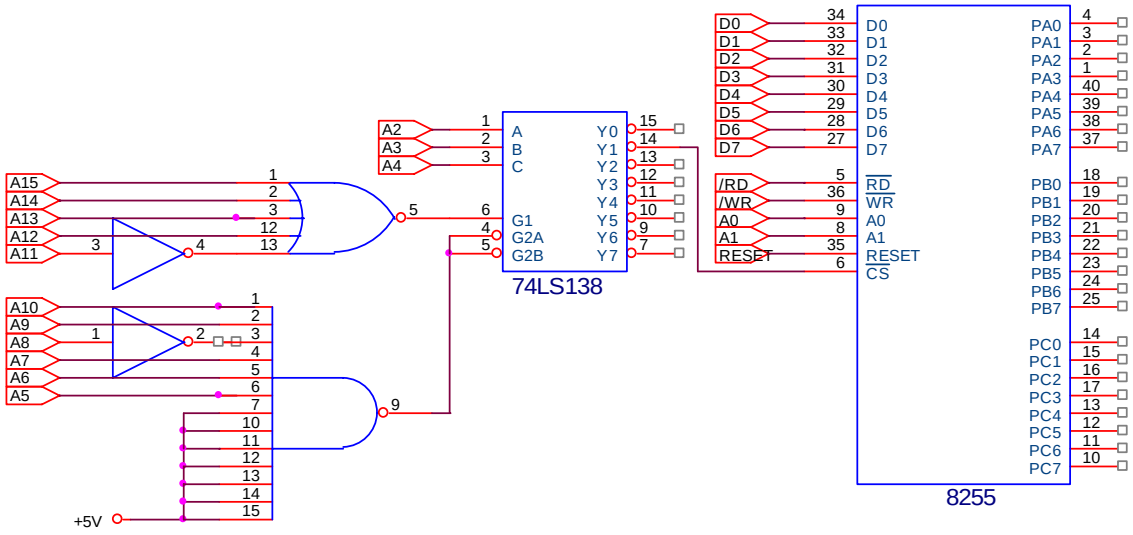
1. Xác định nội dung từ điều khiển của 8255 để:

a. Port A: nhập, Port B: xuất, PCH: nhập, PCL: xuất và hoạt động ở chế độ 0

b. Port A: xuất, Port B: nhập và hoạt động ở chế độ 1.

c. Nhóm A: chế độ 2, nhóm B: chế độ 1, Port B: nhập

2. Xác định địa chỉ của Port A, Port B, Port C và thanh ghi điều khiển.



3. Viết chương trình hợp ngữ thực hiện đọc nội dung tại Port B của 8255 và xuất nối tiếp giá trị vừa đọc ra PC2 theo thứ tự từ LSB → MSB.

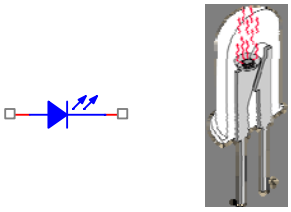
4. Viết chương trình hợp ngữ thực hiện đọc 10 giá trị từ Port A của 8255, sau đó xuất giá trị lớn nhất ra Port B.

CHƯƠNG 4: GIAO TIẾP VỚI CÁC THIẾT BỊ ĐƠN GIẢN

1. Giao tiếp LED (Light Emitting Diode)

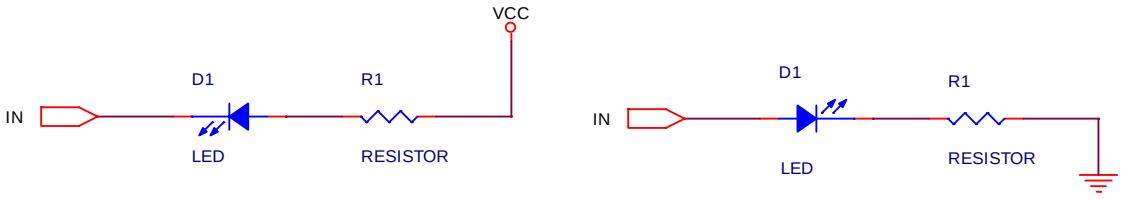
Để giúp cho người sử dụng có thể giao tiếp với máy móc thiết bị điều khiển, ta cần phải có một màn hình hiển thị số và chữ cái.Trong các hệ thống cần hiển thị một lượng lớn thông tin dữ liệu thường dùng CRT (màn hình) để hiển thị còn khi chỉ cần hiển thị một lượng nhỏ thông tin thì sẽ dùng các thiết bị hiển thị số đơn giản do giá rẻ và dễ điều khiển. Có nhiều loại màn hình hiển thị như CRT, LCD, LED, … Ta chỉ xét thiết bị hiển thị đơn giản là LED. Hiển thị số và chữ dùng LED có 3 loại chính.Với các ứng dụng hiển thị dùng để chỉ thị thì dùng LED đơn, để hiện số và chữ số thì dùng Led 7 đoạn hay Led 18 đoạn, để hiển thị ký tự bất kỳ thì dùng ma trận Led.

1.1. Giao tiếp LED đơn



Hình 4.1 - Mô tả LED và biểu diễn trong mạch

Khi LED sáng, dòng qua LED khoảng 10 – 40 mA và điện áp rơi trên LED vào khoảng 1.8V – 2V. Khi đó, ta có mạch điện điều khiển LED như sau:



Hình a Hình b

Hình 4.2 – Sơ đồ kết nối LED đơn

Giả sử mạch kết nối với 8255A có điện áp ứng với mức logic 0 từ 0 – 0.4V và mức logic 1 từ 4.6V – 5V. Chọn dòng qua LED là 20 mA và điện áp rơi trên LED là 2V.

Xét hình a: LED sáng khi mức logic tại chân IN là mức 0, ứng với điện áp 0.4V nên giá trị điện trở R1 là:

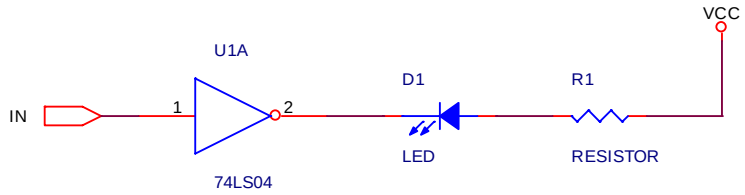
$$R1 = \frac{Vcc - V_{LED} - 0.4}{I_{LED}} = \frac{5 - 2 - 0.4}{20x10^{-3}} = 130\Omega \rightarrow \text{chọn } R1 = 150\ \Omega$$

Xét hình b: LED sáng khi mức logic tại chân IN là mức 1, ứng với điện áp 4.6V nên giá trị điện trở R1 là:

$$R1 = \frac{V_{IN} - V_{LED}}{I_{LED}} = \frac{4.6 - 2}{20x10^{-3}} = 130\Omega \rightarrow \text{chọn } R1 = 150\ \Omega$$

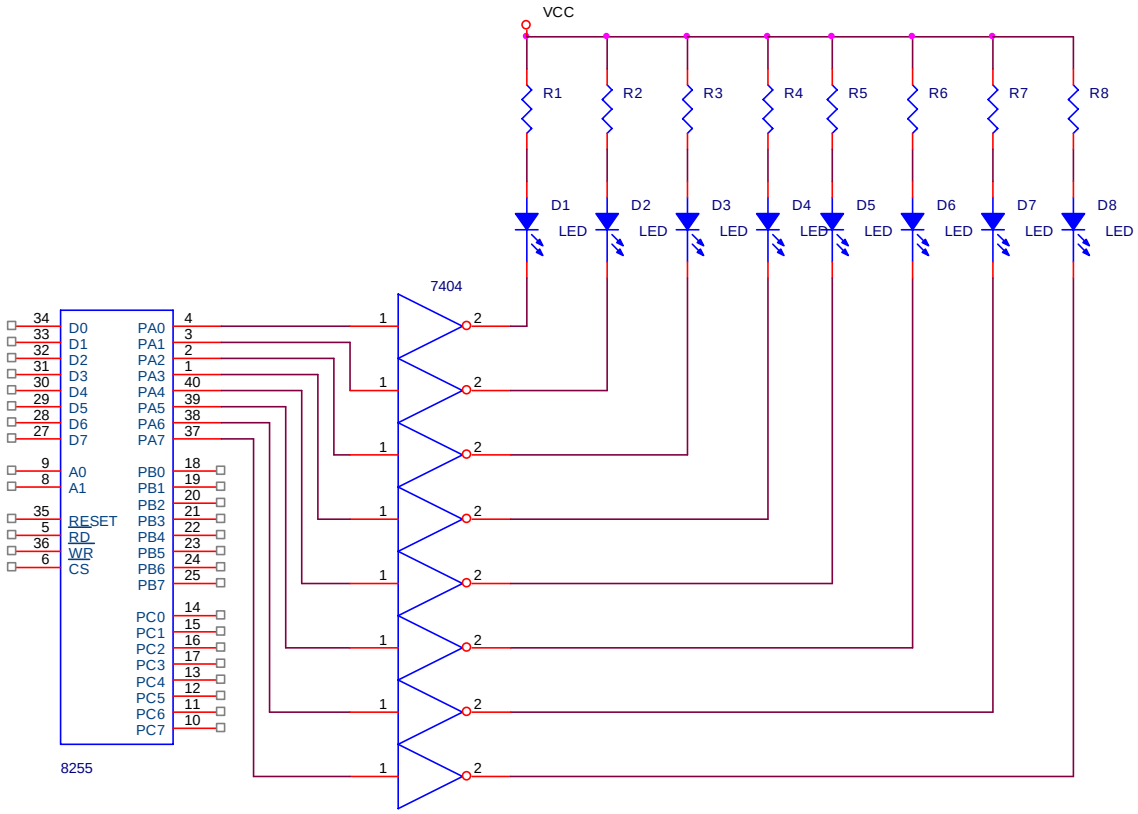
Tuy nhiên khi thiết kế mạch như hình b thì lưu ý rằng dòng tại chân IN phải đáp ứng được giá trị 20 mA. Đối với 8255A, dòng ngõ ra vào khoảng 2.5 mA nên không đáp ứng đủ dòng để sáng LED nên phải dùng thêm mạch khuếch đại.

Thông thường khi thiết kế giao tiếp với 8255A, ta sử dụng mạch hình a nhưng lưu ý là đối với cách thiết kế như trên thì dòng sẽ đi trực tiếp vào cổng vào/ra của 8255A nên ta có thể dùng thêm các cổng đệm hay đảo để tránh làm hư cổng.



Hình 4.3 – Sơ đồ kết nối LED đơn dùng cổng đảo

Xét sơ đồ kết nối giữa LED đơn và 8255 như sau:



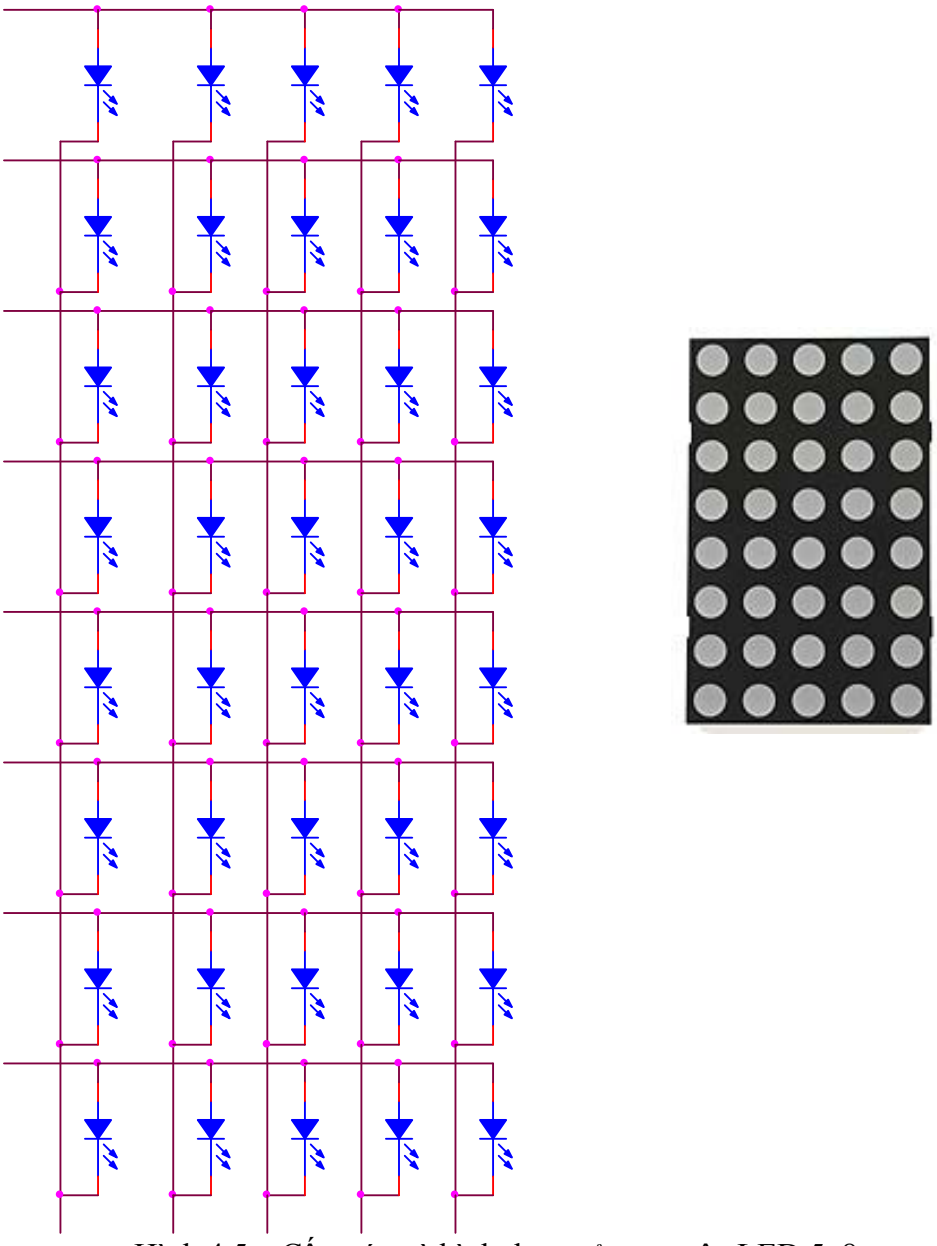
Hình 4.4 - Kết nối giữa LED đơn và 8255

Chương trình hợp ngữ quét LED từ trái sang phải như sau (giả sử địa chỉ Port A, Port B, Port C và CR của 8255 lần lượt là 300h, 301h, 302h và 303h):

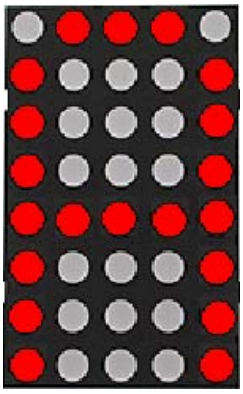
```
.MODEL SMALL
.STACK 100h
.DATA
     Led_data DB 01h,02h,04h,08h,10h,20h,40h,80h
.CODE
Main PROC
     MOV AX,@DATA
     MOV DS, AX          ; Gán địa chỉ cho Data segment
     MOV AL,80h          ; Định cấu hình cho 8255
     MOV DX,303h         ; Port A: xuất, Port B: xuất
     OUT DX,AL           ; Port C: xuất
     MOV BX,0
Lap:
     MOV AL,Led_data[BX]
     MOV DX,300h         ; Địa chỉ LED
     OUT DX,AL

     MOV CX,0FFh
Delay:                   ; Tạo thời gian trễ
     PUSH CX
     MOV CX,0FFFFh
     LOOP $
     POP CX
     LOOP delay

     INC BX
     CMP BX,8            ; LED có 8 trạng thái
     JNE lap

     MOV AH,4Ch          ; Kết thúc chương trình
     INT 21h
Main ENDP
END Main
```

1.2. Giao tiếp ma trận LED

Ma trận LED bao gồm nhiều LED cùng nằm trong một vỏ chia thành nhiều cột và hàng, mỗi giao điểm giữa hàng và cột có thể có 1 LED (ma trận LED một màu) hay nhiều LED (2 LED tại một vị trí tạo thành ma trận LED 3 màu). Để LED tại một vị trí nào đó sáng thì phải cấp điện áp dương tại Anode và điện áp âm tại Cathode (nghĩa là cấp mức logic 1 tại hàng và logic 0 tại cột ứng với ma trận LED có kết nối như hình 4.5). Trên cơ sở cấu trúc như hình vẽ 4.5, ta có thể mở rộng hàng và cột của ma trận LED để tạo thành các bảng quang báo.

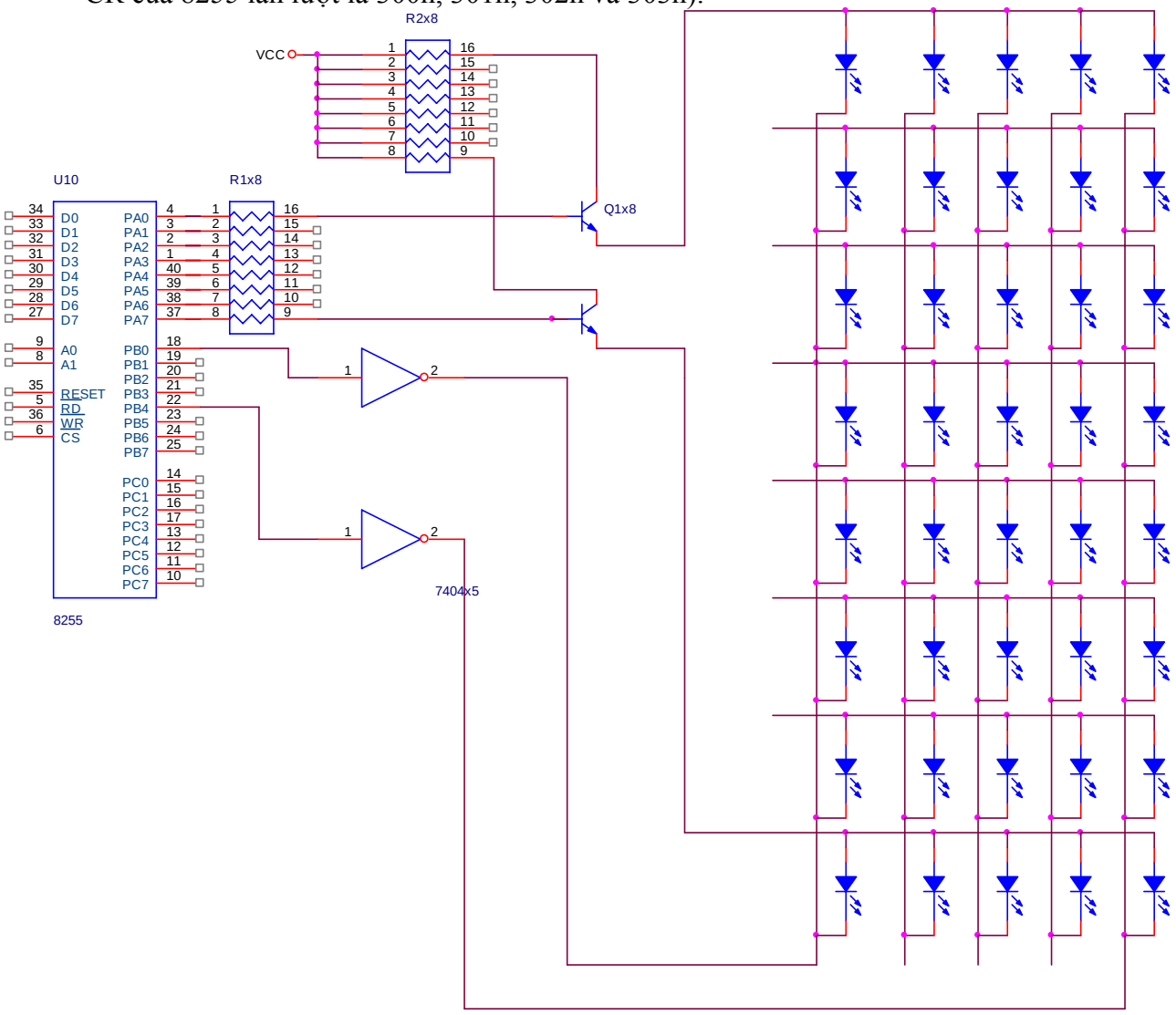


Hình 4.5 – Cấu trúc và hình dạng của ma trận LED 5x8



Hình 4.6 – Sáng chữ ‘A’ trên ma trận LED 5x8

Xét sơ đồ kết nối ma trận LED với 8255 như hình 4.7 trong đó Port A điều khiển hàng thông qua các transistor và Port B điều khiển cột (địa chỉ Port A, B, C và CR của 8255 lần lượt là 300h, 301h, 302h và 303h).



Hình 4.7 – Kết nối ma trận LED với 8255

Tính toán cho mạch:

Transistor Q1 hoạt động ở chế độ bão hoà với dòng I_c là dòng qua LED nên chọn Q1 có $\beta = 50$, $V_{CE} = 0{,}2V$, $V_{BE} = 0{,}7V$

$$R2 = \frac{Vcc - V_{LED} - V_{OL-7404} - V_{CE}}{I_{LED}} = \frac{5 - 2 - 0{,}4 - 0{,}2}{0{,}01} = 240\ \Omega, \text{ chọn } R2 = 220\ \Omega$$

$$R1 = \frac{V_{OH-8255} - V_{BE} - V_{LED} - V_{OL-7404}}{I_B} = \frac{4{,}6 - 0{,}7 - 2 - 0{,}4}{0{,}01/50} = 7.5\ K\Omega, \text{ chọn } R1 = 8.2\ K\Omega$$

Để hiển thị một ký tự trên ma trận LED, ta sẽ cho sáng các LED tương ứng. Ví dụ như để sáng chữ 'A' trên ma trận LED, ta cho tương ứng LED sáng như hình 4.6. Tuy nhiên, ta không thể cho sáng đồng thời 2 LED tại 2 vị trí hàng và cột khác nhau vì sẽ ảnh hưởng đến các LED còn lại. Ví dụ như khi sáng LED tại hàng 1, cột 2 (PA0 = 1, PB1 = 1) và hàng 2, cột 1 (PA1 = 1, PB0 = 1) thì do PA0 = 1, PB0 = 1; PA1 = 1, PB1 = 1 nên LED tại hàng 1, cột 1 và hàng 2, cột 2 cũng sáng.

Như vậy để sáng ký tự 'A' trên ma trận LED thì phải dùng phương pháp quét khi hiển thị dữ liệu trên ma trận LED. Quá trình quét có thể thực hiện quét dòng hay cột. Khi thực hiện quét cột, tại mỗi thời điểm chỉ có một cột sáng. Dữ liệu cho Port B và Port B của 8255A như sau:

- Cột 1: sáng 7 LED

```
PB0 = 1, PB1 – 4 = 0: PB = xxx0 0001b (01h)
PA0 = 0, PA1 – 7 = 1: PA = 1111 1110b (0FEh)
```

- Cột 2: sáng 2 LED

```
PB0 = 0, PB2 – 4 = 0, PB1 = 1: PB = xxx0 0010b (02h)
PA0 = 1, PA4 = 1, PA1 – 3 = 0, PA5 – 7 = 0:
     PA = 0001 0001b (11h)
```

- Cột 3: giống cột 2

```
PB0 - 1 = 0, PB3 – 4 = 0, PB2 = 1:
     PB = xxx0 0100b (04h)
PA0 = 1, PA4 = 1, PA1 – 3 = 0, PA5 – 7 = 0:
     PA = 0001 0001b (11h)
```

- Cột 4: giống cột 2

```
PB0 - 2 = 0, PB4 = 0, PB3 = 1: PB = xxx0 1000b (08h)
PA0 = 1, PA4 = 1, PA1 – 3 = 0, PA5 – 7 = 0:
     PA = 0001 0001b (11h)
```

- Cột 5: giống cột 1

```
PB0 - 3 = 0, PB4 = 1: PB = xxx1 0000b (10h)
PA0 = 0, PA1 – 7 = 1: PA = 1111 1110b (0FEh)
```

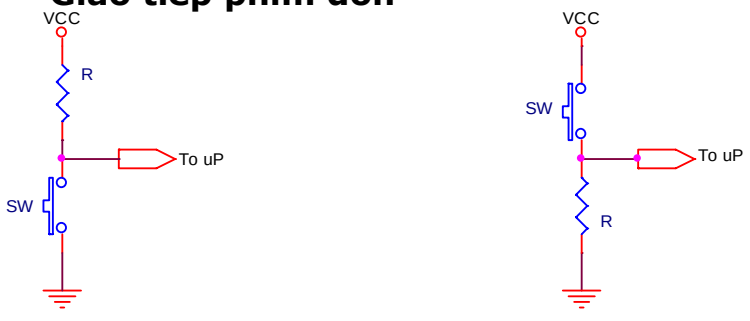
Chương trình hiển thị cho ma trận LED như sau:

```
.MODEL SMALL
.STACK 100h
.DATA
     pa DB 0FEh,11h,11h,11h,0FEh
     pb DB 01h,02h,04h,08h,10h
.CODE
Main PROC
     MOV AX,@DATA
     MOV DS, AX          ; Gán địa chỉ cho Data segment

     MOV AL,80h          ; Định cấu hình cho 8255
```

```
        MOV DX,303h         ; Port A: xuất, Port B: xuất
        OUT DX,AL           ; Port C: xuất
    Start:
        MOV AH,0Bh          ; Kiểm tra phím nhấn
        INT 21h
        CMP AL,0            ; Nếu có phím nhấn
        JNE Exit            ; thì kết thúc chương trình

        MOV BX,0
    Lap:
        MOV AL,pa[BX]
        MOV DX,300h         ; Xuất ra hàng
        OUT DX,AL

        MOV AL,pb[BX]
        MOV DX,301h         ; Xuất ra cột
        OUT DX,AL

        PUSH CX             ; Tạo thời gian trễ
        MOV CX,0FFFFh
        LOOP $
        POP CX

        INC BX
        CMP BX,5            ; Quét 5 cột
        JNE lap
        JMP Start

    Exit:
        MOV AH,4Ch          ; Kết thúc chương trình
        INT 21h
    Main ENDP

    END Main
```

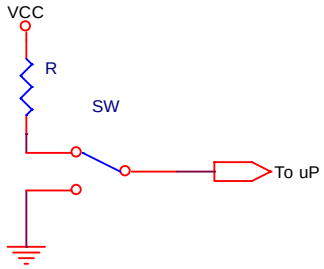
2. Giao tiếp bàn phím

2.1. Giao tiếp phím đơn



a/ b/

Hình 4.8 – Sơ đồ kết nối phím nhấn (2 chân) với vi xử lý



Hình 4.9 – Sơ đồ kết nối phím nhấn (3 chân) với vi xử lý

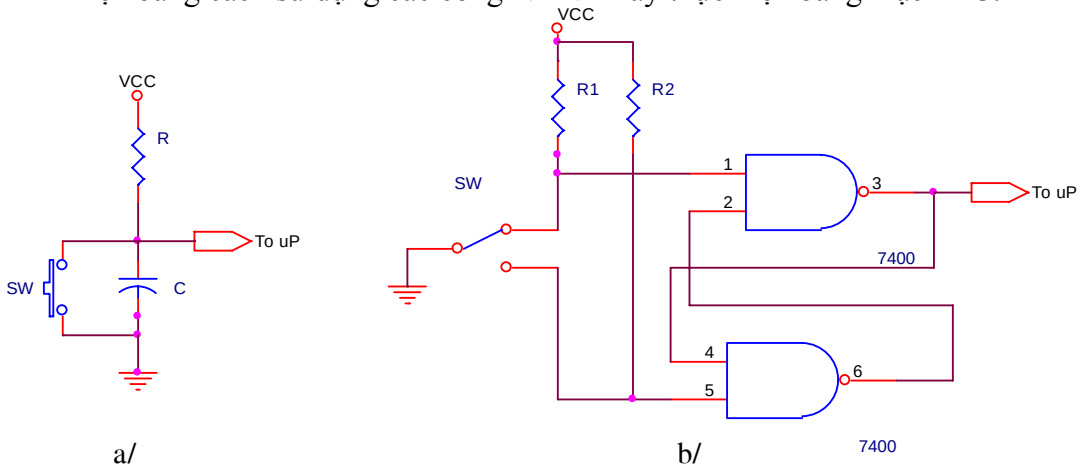
Các phím đơn dùng để điều khiển khi hệ thống vi xử lý không đòi hỏi nhiều giá trị nhập (chẳng như chỉ cần các điều khiển đóng mở thiết bị). Sơ đồ kết nối phím đơn mô tả như hình 4.8 hay 4.9.

Khi thực hiện kiểm tra phím nhấn, vấn đề cần thiết là phải thực hiện chống dội. Hiện tượng dội khi nhấn phím có thể mô tả như hình 4.10 (giả sử khi nhấn phím thì ngõ ra ở mức logic 1 và nhả phím thì ngõ ra ở mức logic 0 – hình 4.8b). Quá trình chống dội có thể thực hiện bằng phần mềm hay phần cứng.

Hình 4.10 – Hiện tượng dội khi nhấn phím

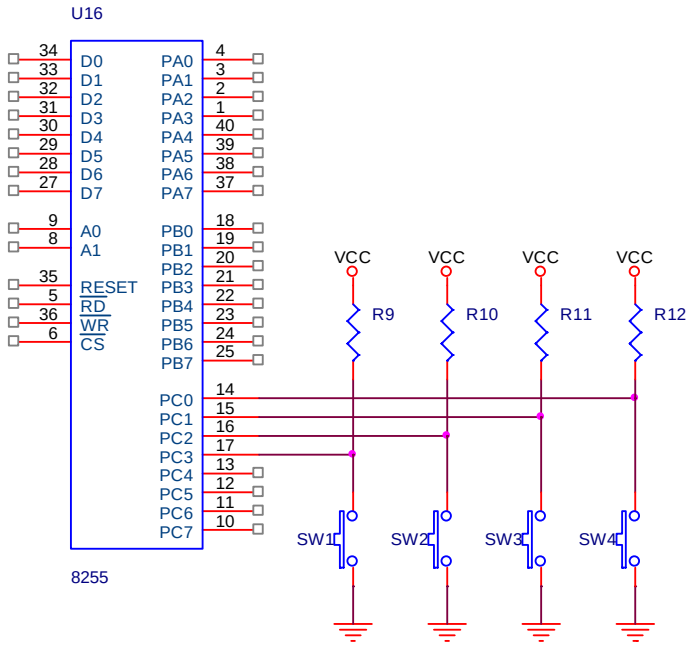
- **Phần mềm:** Do thời gian dội của phím vào khoảng 20ms nên quá trình chống dội bằng phần mềm đơn giản là tạo một thời gian trễ đủ lớn để chương trình bỏ qua ảnh hưởng khi dội.

- **Phần cứng:** Khi thực hiện chống dội bằng phần cứng, ta có thể thực hiện bằng cách sử dụng các cổng NAND hay thực hiện bằng mạch RC.



Hình 4.11 – Chống dội phím nhấn bằng phần cứng

Tuy nhiên khi thực hiện thiết kế phần cứng, thông thường ta sẽ chống dội bằng phần mềm để đơn giản mạch phần cứng. Xét sơ đồ phần cứng thiết kế như hình 4.12 (giả sử địa chỉ Port A, B, C và CR của 8255 lần lượt là 300h, 301h, 302h và 303h).



Hình 4.12 – Kết nối phím nhấn với 8255

Chương trình hợp ngữ kiểm tra các phím SW1, SW2, SW3, SW4 và hiển thị trạng thái phím nhấn lên màn hình:

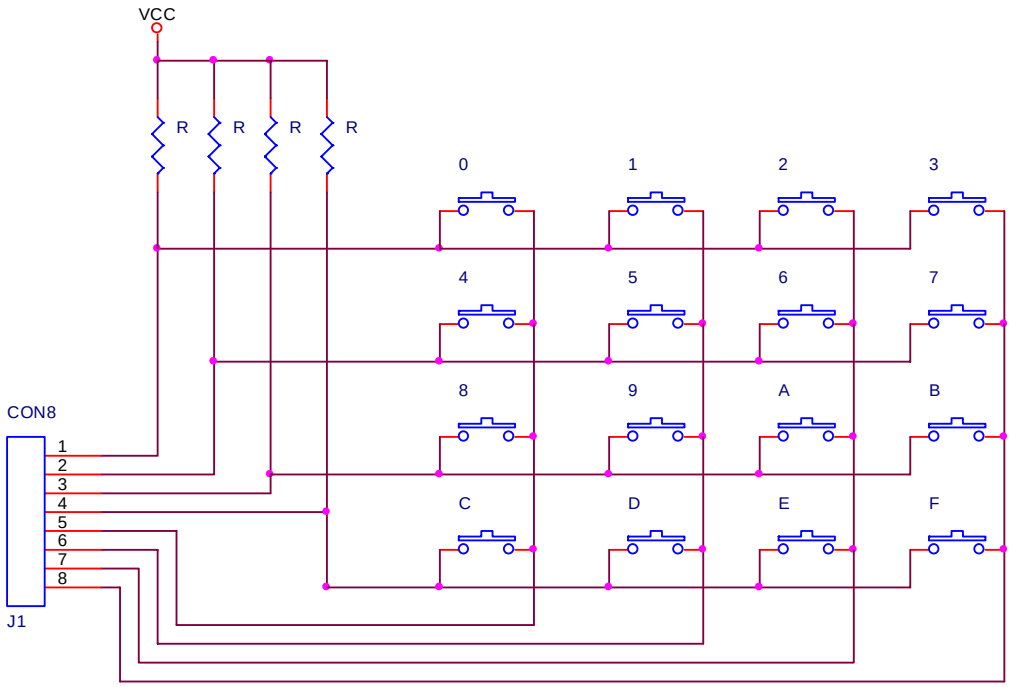
```
.MODEL SMALL
.STACK 100h
.DATA
     Msg DB 'Da nhan phim: SW'
     sw   DB   ?,13,10,'$'
.CODE
Main PROC
     MOV AX,@DATA
     MOV DS,AX

     MOV AL,81h          ; Định cấu hình cho 8255
     MOV DX,303h         ; PA, PB, PC (high): xuất
     OUT DX,AL           ; PC (low): nhập


Start:
     MOV AH,0Bh          ; Kiểm tra phím nhấn
     INT 21h
     CMP AL,0            ; Nếu có phím nhấn
     JNE Exit            ; thì kết thúc chương trình

     MOV DX,302h         ; Đọc từ Port C để kiểm tra
     IN AL,DX            ; phím nhấn
```

```
        AND AL,0Fh          ; Xoá 4 bit cao
        CMP  AL,00001110b   ; Nếu nhấn SW1 thì PC0 = 0
        JE Sw1
        CMP  AL,00001101b   ; Nếu nhấn SW2 thì PC1 = 0
        JE Sw2
        CMP  AL,00001011b   ; Nếu nhấn SW3 thì PC2 = 0
        JE Sw3
        CMP  AL,00000111b   ; Nếu nhấn SW4 thì PC3 = 0
        JE Sw4
        JMP Start
    Sw1:
        CALL Delay
        MOV sw,'1'
        MOV AH,09h
        LEA DX,Msg
        INT 21h
        JMP Start
    Sw2:
        CALL Delay
        MOV sw,'2'
        MOV AH,09h
        LEA DX,Msg
        INT 21h
        JMP Start
    Sw3:
        CALL Delay
        MOV sw,'3'
        MOV AH,09h
        LEA DX,Msg
        INT 21h
        JMP Start
    Sw4:
        CALL Delay
        MOV sw,'4'
        MOV AH,09h
        LEA DX,Msg
        INT 21h
        JMP Start

    Exit:
        MOV AH,4Ch          ; Kết thúc chương trình
        INT 21h

    Main ENDP
    ;-------------
    Delay PROC
        PUSH CX
        MOV CX,0FFFFh
        LOOP $
```

```
        POP CX
        RET
Delay ENDP
;-------------
END Main
```

2.2. Giao tiếp bàn phím Hex

Bàn phím Hex là bàn phím xây dựng theo cấu trúc ma trận gồm 16 phím chia thành 4 hàng và 4 cột (hình 4.13).



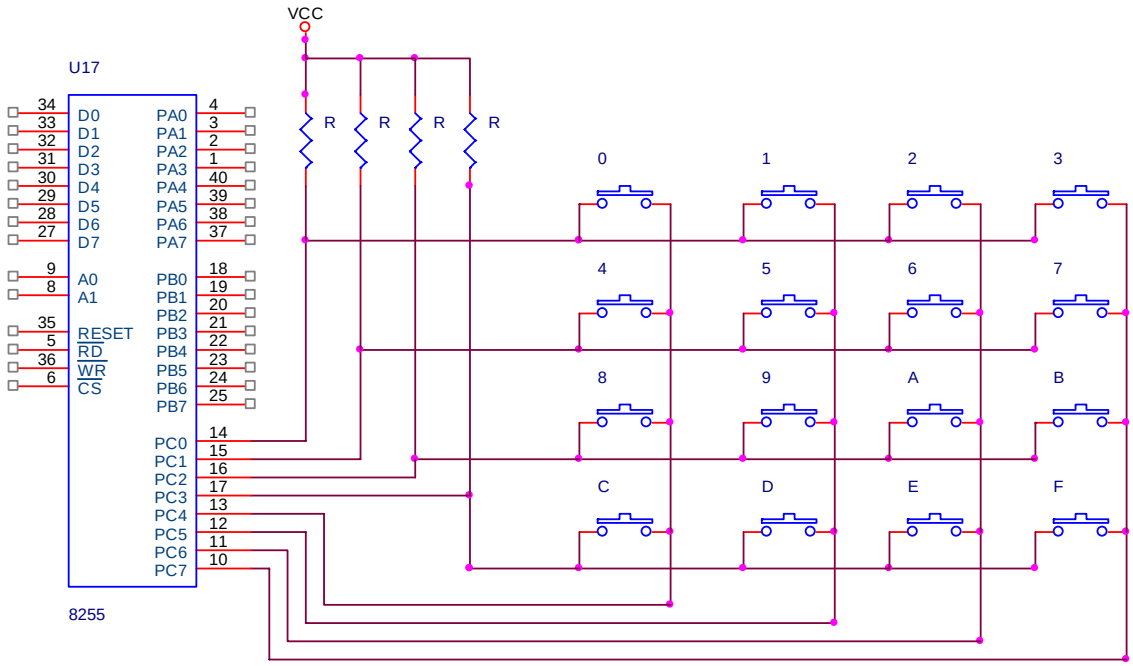
Hình 4.13 – Cấu trúc bàn phím Hex

Lưu ý rằng khi không nhấn phím thì hàng của bàn phím Hex nối với Vcc thông qua điện trở R nên có mức logic 1. Để phân biệt được trạng thái của phím nhấn thì mức logic khi nhấn phím phải là mức logic 0. Mà khi nhấn một phím nào đó thì tương ứng hàng và cột của bàn phím Hex sẽ kết nối với nhau. Do đó, để thực hiện kiểm tra một phím thì ta phải cho trước cột chứa phím tương ứng ở mức logic 0, sau đó kiểm tra hàng của phím, nếu hàng = 0 thì có nhấn phím còn hàng = 1 thì không nhấn phím.

Ví dụ như muốn kiểm tra phím 4 thì ta cho cột chứa phím 4 ở mức logic 0 (chân 5 của J1, các cột khác = 1, nghĩa là dữ liệu tại J1 là 1000xxxxb), sau đó thực hiện kiểm tra chân 2 của J1 (hàng của phím 4), nếu chân này = 0 thì phím 4 được nhấn.

Xét sơ đồ kết nối của bàn phím Hex với 8255 như hình 4.14 (giả sử địa chỉ Port A, B, C và CR của 8255 lần lượt là 300h, 301h, 302h và 303h). Ở đây ta thực hiện kết nối với Port C của 8255 do Port C có thể chia thành 2 phần: 4 bit cao và 4 bit thấp có thể xuất nhập độc lập trong khi đó Port A và Port B chỉ có thể đồng thời xuất hay nhập

còn để xác định có phím nhấn trên bàn phím Hex hay không thì phải xuất dữ liệu điều khiển cột và đọc dữ liệu từ hàng của bàn phím.



Hình 4.14 – Sơ đồ kết nối bàn phím Hex với 8255

Chương trình hợp ngữ kiểm tra bàn phím như sau:

```
.MODEL SMALL
.STACK 100h
.DATA
.CODE
Main PROC
     MOV AX,@DATA
     MOV DS,AX

     MOV AL,81h          ; Định cấu hình cho 8255
     MOV DX,303h         ; PA, PB, PC (high): xuất
     OUT DX,AL           ; PC (low): nhập


Start:
     MOV AH,0Bh          ; Kiểm tra phím nhấn
     INT 21h
     CMP AL,0            ; Nếu có phím nhấn
     JE Next
     JMP Exit            ; thì kết thúc chương trình

Next:
     MOV DX,302h         ; Địa chỉ Port C
     MOV AL,11100000b    ; Cho PC4 = 0 ứng với các
```

```
        OUT DX,AL           ; phím 0,4,8,C
        IN AL,DX            ; Đọc về kiểm tra hàng
        AND AL,0Fh          ; Xoá 4 bit cao
        CMP  AL,00001110b   ; Nếu nhấn phím 0 thì PC0 = 0
        JNE Not0
        CALL Sw0

    Not0:
        CMP  AL,00001101b   ; Nếu nhấn phím 4 thì PC1 = 0
        JNE Not4
        CALL Sw4

    Not4:
        CMP  AL,00001011b   ; Nếu nhấn phím 8 thì PC2 = 0
        JNE Not8
        CALL Sw8

    Not8:
        CMP  AL,00000111b   ; Nếu nhấn phím C thì PC3 = 0
        JNE NotC
        CALL SwC

    NotC:
        MOV DX,302h         ; Địa chỉ Port C
        MOV AL,11010000b    ; Cho PC5 = 0 ứng với các
        OUT DX,AL           ; phím 1,5,9,D
        IN AL,DX            ; Đọc về kiểm tra hàng
        AND AL,0Fh          ; Xoá 4 bit cao
        CMP  AL,00001110b   ; Nếu nhấn phím 1 thì PC0 = 0
        JNE Not1
        CALL Sw1

    Not1:
        CMP  AL,00001101b   ; Nếu nhấn phím 5 thì PC1 = 0
        JNE Not5
        CALL Sw5

    Not5:
        CMP  AL,00001011b   ; Nếu nhấn phím 9 thì PC2 = 0
        JNE Not9
        CALL Sw9

    Not9:
        CMP  AL,00000111b   ; Nếu nhấn phím D thì PC3 = 0
        JNE NotD
        CALL SwD

    NotD:
        MOV DX,302h         ; Địa chỉ Port C
```

```
        MOV AL,10110000b   ; Cho PC6 = 0 ứng với các
        OUT DX,AL          ; phím 2,6,A,E
        IN AL,DX           ; Đọc về kiểm tra hàng
        AND AL,0Fh         ; Xoá 4 bit cao
        CMP  AL,00001110b  ; Nếu nhấn phím 2 thì PC0 = 0
        JNE Not2
        CALL Sw2

    Not2:
        CMP  AL,00001101b  ; Nếu nhấn phím 6 thì PC1 = 0
        JNE Not6
        CALL Sw6

    Not6:
        CMP  AL,00001011b  ; Nếu nhấn phím A thì PC2 = 0
        JNE NotA
        CALL SwA

    NotA:
        CMP  AL,00000111b  ; Nếu nhấn phím E thì PC3 = 0
        JNE NotE
        CALL SwE

    NotE:
        MOV DX,302h        ; Địa chỉ Port C
        MOV AL,01110000b   ; Cho PC7 = 0 ứng với các
        OUT DX,AL          ; phím 3,7,B,F
        IN AL,DX           ; Đọc về kiểm tra hàng
        AND AL,0Fh         ; Xoá 4 bit cao
        CMP  AL,00001110b  ; Nếu nhấn phím 3 thì PC0 = 0
        JNE Not3
        CALL Sw3

    Not3:
        CMP  AL,00001101b  ; Nếu nhấn phím 7 thì PC1 = 0
        JNE Not7
        CALL Sw7

    Not7:
        CMP  AL,00001011b  ; Nếu nhấn phím B thì PC2 = 0
        JNE NotB
        CALL SwB

    NotB:
        CMP  AL,00000111b  ; Nếu nhấn phím F thì PC3 = 0
        JNE NotF
        CALL SwF

    NotF:
```

```
     JMP Start

Exit:
     MOV AH,4Ch         ; Kết thúc chương trình
     INT 21h
Main ENDP
;-----------

Sw0 PROC
     ; Chương trình cho phím 0
     RET
Sw0 ENDP
;-----------

Sw1 PROC
     ; Chương trình cho phím 1
     RET
Sw1 ENDP
;-----------

Sw2 PROC
     ; Chương trình cho phím 2
     RET
Sw2 ENDP
;-----------

Sw3 PROC
     ; Chương trình cho phím 3
     RET
Sw3 ENDP
;-----------

Sw4 PROC
     ; Chương trình cho phím 4
     RET
Sw4 ENDP
;-----------

Sw5 PROC
     ; Chương trình cho phím 5
     RET
Sw5 ENDP
;-----------

Sw6 PROC
     ; Chương trình cho phím 6
     RET
Sw6 ENDP
;-----------
```

```
Sw7 PROC
     ; Chương trình cho phím 7
     RET
Sw7 ENDP
;-----------

Sw8 PROC
     ; Chương trình cho phím 8
     RET
Sw8 ENDP
;-----------
Sw9 PROC
     ; Chương trình cho phím 9
     RET
Sw9 ENDP
;-----------

SwA PROC
     ; Chương trình cho phím A
     RET
SwA ENDP
;-----------

SwB PROC
     ; Chương trình cho phím B
     RET
SwB ENDP
;-----------

SwC PROC
     ; Chương trình cho phím C
     RET
SwC ENDP
;-----------

SwD PROC
     ; Chương trình cho phím D
     RET
SwD ENDP
;-----------

SwE PROC
     ; Chương trình cho phím E
     RET
SwE ENDP
;-----------

SwF PROC
```

```
        ; Chương trình cho phím F
        RET
SwF ENDP
;-----------

Delay PROC
        PUSH CX
        MOV CX,0FFFFh
        LOOP $
        POP CX
        RET
Delay ENDP
END Main
```

BÀI TẬP CHƯƠNG 4

1. Giả sử Port A của 8255 kết nối như hình 4.4, Port C kết nối với 2 công tắc SW1, SW2 tương ứng tại PC7, PC6 tương tự như hình 4.12. Viết chương trình hợp ngữ điều khiển công tắc sao cho:

- Nhấn SW1: LED sáng tuần tự từ trong ra ngoài, mỗi lần sáng tương ứng 2 LED.

- Nhấn SW2: tắt các LED và kết thúc chương trình

2. Giả sử Port A và Port B của 8255 kết nối với ma trận LED 5x8 như hình 4.7 còn Port C kết nối với bàn phím Hex như hình 4.14. Viết chương trình hợp ngữ điều khiển bàn phím Hex sao cho:

- Nhấn phím 2: sáng số ‘2’ trên ma trận LED

- Nhấn phím D: sáng chữ ‘D’ trên ma trận LED

82C55A

CMOS Programmable Peripheral Interface

June 1998

Features

- **Pin Compatible with NMOS 8255A**
- **24 Programmable I/O Pins**
- **Fully TTL Compatible**
- **High Speed, No "Wait State" Operation with 5MHz and 8MHz 80C86 and 80C88**
- **Direct Bit Set/Reset Capability**
- **Enhanced Control Word Read Capability**
- **L7 Process**
- **2.5mA Drive Capability on All I/O Ports**
- **Low Standby Power (ICCSB) 10μA**

Ordering Information

PART NUMBERS		PACKAGE	TEMPERATURE RANGE	PKG. NO.
5MHz	8MHz			
CP82C55A-5	CP82C55A	40 Ld PDIP	0^oC to 70^oC	E40.6
IP82C55A-5	IP82C55A		-40^oC to 85^oC	E40.6
CS82C55A-5	CS82C55A	44 Ld PLCC	0^oC to 70^oC	N44.65
IS82C55A-5	IS82C55A		-40^oC to 85^oC	N44.65
CD82C55A-5	CD82C55A	40 Ld CERDIP	0^oC to 70^oC	F40.6
ID82C55A-5	ID82C55A		-40^oC to 85^oC	F40.6
MD82C55A-5/B	MD82C55A/B		-55^oC to 125^oC	F40.6
8406601QA	8406602QA	SMD#		F40.6
MR82C55A-5/B	MR82C55A/B	44 Pad CLCC	-55^oC to 125^oC	J44.A
8406601XA	8406602XA	SMD#		J44.A

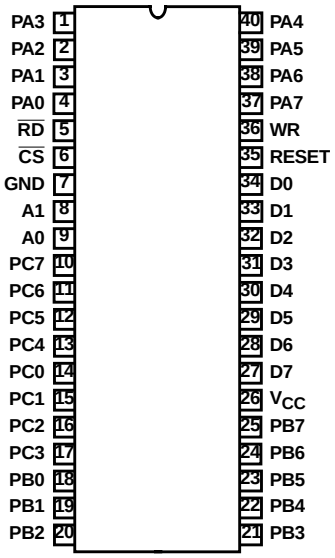
Description

The Intersil 82C55A is a high performance CMOS version of the industry standard 8255A and is manufactured using a self-aligned silicon gate CMOS process (Scaled SAJI IV). It is a general purpose programmable I/O device which may be used with many different microprocessors. There are 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. The high performance and industry standard configuration of the 82C55A make it compatible with the 80C86, 80C88 and other microprocessors.

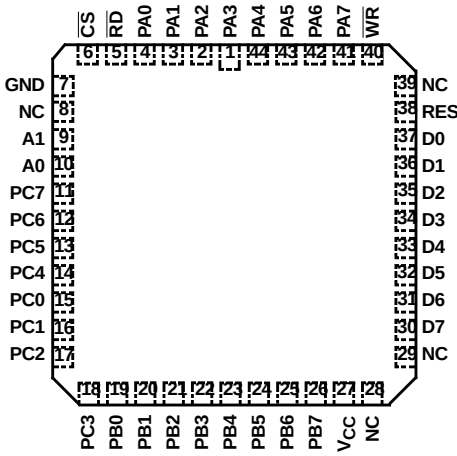
Static CMOS circuit design insures low operating power. TTL compatibility over the full military temperature range and bus hold circuitry eliminate the need for pull-up resistors. The Intersil advanced SAJI process results in performance equal to or greater than existing functionally equivalent products at a fraction of the power.

Pinouts

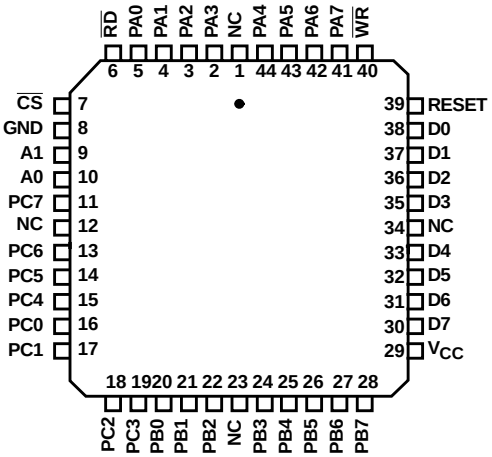
82C55A (DIP)
TOP VIEW



82C55A (CLCC)
TOP VIEW



82C55A (PLCC)
TOP VIEW

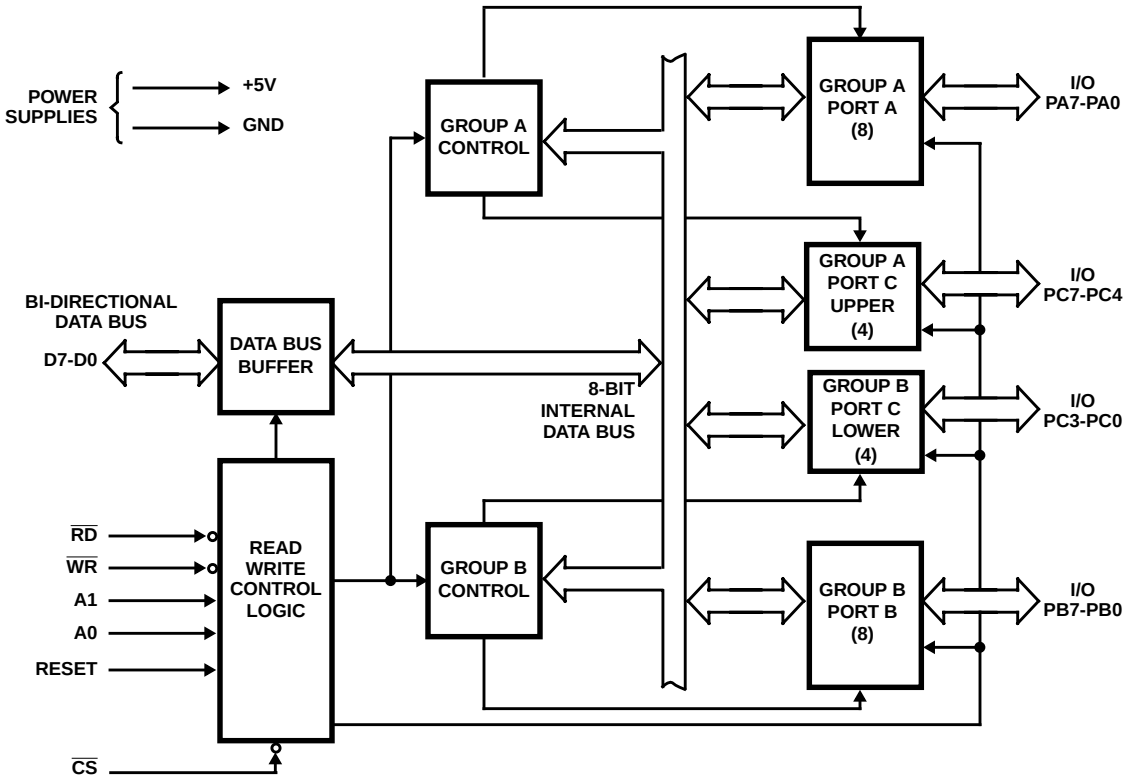


File Number **2969.2**

Pin Description

SYMBOL	PIN NUMBER	TYPE	DESCRIPTION
V_{CC}	26		V_{CC}: The +5V power supply pin. A 0.1µF capacitor between pins 26 and 7 is recommended for decoupling.
GND	7		GROUND
D0-D7	27-34	I/O	DATA BUS: The Data Bus lines are bidirectional three-state pins connected to the system data bus.
RESET	35	I	RESET: A high on this input clears the control register and all ports (A, B, C) are set to the input mode with the “Bus Hold” circuitry turned on.
$\overline{CS}$	6	I	CHIP SELECT: Chip select is an active low input used to enable the 82C55A onto the Data Bus for CPU communications.
$\overline{RD}$	5	I	READ: Read is an active low input control signal used by the CPU to read status information or data via the data bus.
$\overline{WR}$	36	I	WRITE: Write is an active low input control signal used by the CPU to load control words and data into the 82C55A.
A0-A1	8, 9	I	ADDRESS: These input signals, in conjunction with the $\overline{RD}$ and $\overline{WR}$ inputs, control the selection of one of the three ports or the control word register. A0 and A1 are normally connected to the least significant bits of the Address Bus A0, A1.
PA0-PA7	1-4, 37-40	I/O	PORT A: 8-bit input and output port. Both bus hold high and bus hold low circuitry are present on this port.
PB0-PB7	18-25	I/O	PORT B: 8-bit input and output port. Bus hold high circuitry is present on this port.
PC0-PC7	10-17	I/O	PORT C: 8-bit input and output port. Bus hold circuitry is present on this port.

Functional Diagram



Functional Description

Data Bus Buffer

This three-state bi-directional 8-bit buffer is used to interface the 82C55A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

(CS) Chip Select. A “low” on this input pin enables the communcation between the 82C55A and the CPU.

(RD) Read. A “low” on this input pin enables 82C55A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to “read from” the 82C55A.

(WR) Write. A “low” on this input pin enables the CPU to write data or control words into the 82C55A.

(A0 and A1) Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word register. They are normally connected to the least significant bits of the address bus (A0 and A1).

82C55A BASIC OPERATION

A1	A0	RD	WR	CS	INPUT OPERATION (READ)
0	0	0	1	0	Port A → Data Bus
0	1	0	1	0	Port B → Data Bus
1	0	0	1	0	Port C → Data Bus
1	1	0	1	0	Control Word → Data Bus
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	Data Bus → Port A
0	1	1	0	0	Data Bus → Port B
1	0	1	0	0	Data Bus → Port C
1	1	1	0	0	Data Bus → Control
					DISABLE FUNCTION
X	X	X	X	1	Data Bus → Three-State
X	X	1	1	0	Data Bus → Three-State

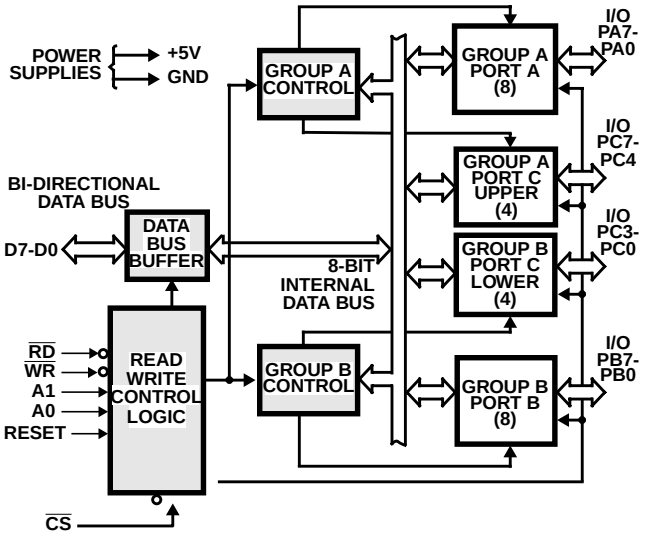


FIGURE 1. 82C55A BLOCK DIAGRAM. DATA BUS BUFFER, READ/WRITE, GROUP A & B CONTROL LOGIC FUNCTIONS

(RESET) Reset. A “high” on this input initializes the control register to 9Bh and all ports (A, B, C) are set to the input mode. “Bus hold” devices internal to the 82C55A will hold the I/O port inputs to a logic “1” state with a maximum hold current of 400µA.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU “outputs” a control word to the 82C55A. The control word contains information such as “mode”, “bit set”, “bit reset”, etc., that initializes the functional configuration of the 82C55A.

Each of the Control blocks (Group A and Group B) accepts “commands” from the Read/Write Control logic, receives “control words” from the internal data bus and issues the proper commands to its associated ports.

Control Group A - Port A and Port C upper (C7 - C4)

Control Group B - Port B and Port C lower (C3 - C0)

The control word register can be both written and read as shown in the “Basic Operation” table. Figure 4 shows the control word format for both Read and Write operations. When the control word is read, bit D7 will always be a logic “1”, as this implies control word mode information.

13Ports A, B, and C

The 82C55A contains three 8-bit ports (A, B, and C). All can be configured to a wide variety of functional characteristics by the system software but each has its own special features or “personality” to further enhance the power and flexibility of the 82C55A.

Port A One 8-bit data output latch/buffer and one 8-bit data input latch. Both “pull-up” and “pull-down” bus-hold devices are present on Port A. See Figure 2A.

Port B One 8-bit data input/output latch/buffer and one 8-bit data input buffer. See Figure 2B.

Port C One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal output and status signal inputs in conjunction with ports A and B. See Figure 2B.

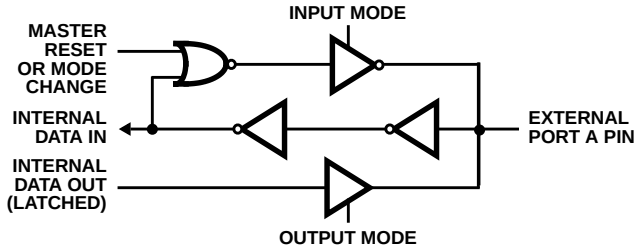


FIGURE 2A. PORT A BUS-HOLD CONFIGURATION

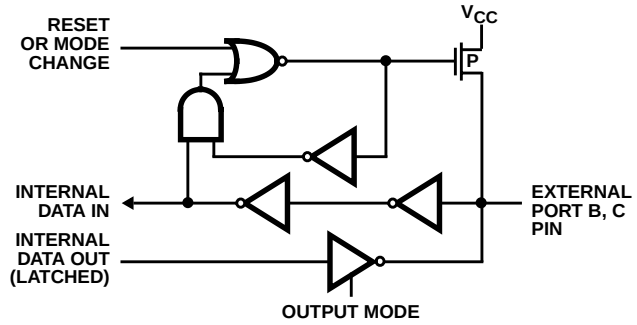


FIGURE 2B. PORT B AND C BUS-HOLD CONFIGURATION

FIGURE 2. BUS-HOLD CONFIGURATION

Operational Description

Mode Selection

There are three basic modes of operation than can be selected by the system software:

Mode 0 - Basic Input/Output
Mode 1 - Strobed Input/Output
Mode 2 - Bi-directional Bus

When the reset input goes “high”, all ports will be set to the input mode with all 24 port lines held at a logic “one” level by internal bus hold devices. After the reset is removed, the 82C55A can remain in the input mode with no additional initialization required. This eliminates the need to pullup or pull-down resistors in all-CMOS designs. The control word register will contain 9Bh. During the execution of the system program, any of the other modes may be selected using a single output instruction. This allows a single 82C55A to service a variety of peripheral devices with a simple software maintenance routine. Any port programmed as an output port is initialized to all zeros when the control word is written.

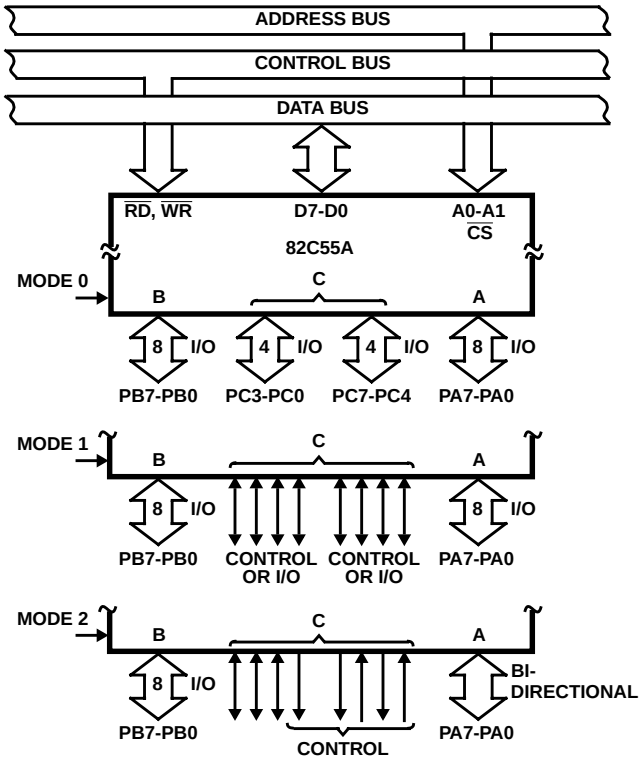


FIGURE 3. BASIC MODE DEFINITIONS AND BUS INTERFACE

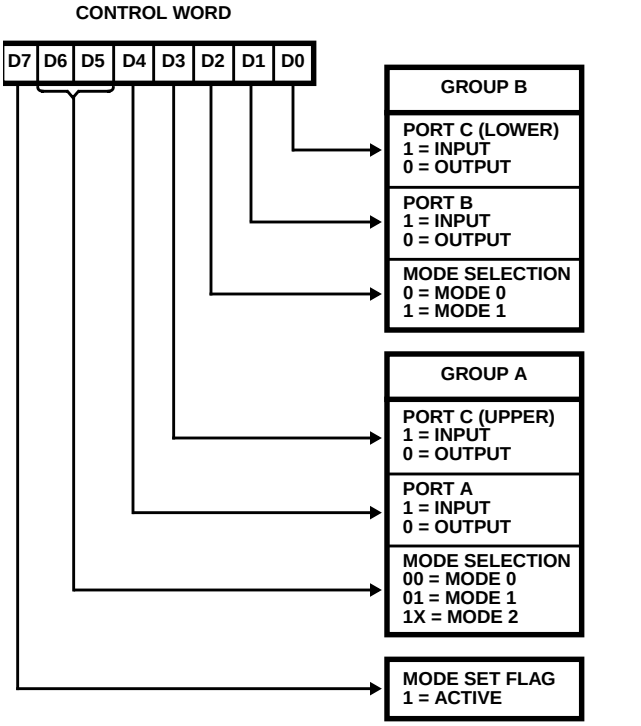


FIGURE 4. MODE DEFINITION FORMAT

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be “tailored” to almost any I/O structure. For instance: Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.

The mode definitions and possible mode combinations may seem confusing at first, but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 82C55A has taken into account things such as efficient PC board layout, control signal definition vs. PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature (Figure 5)

Any of the eight bits of Port C can be Set or Reset using a single Output instruction. This feature reduces software requirements in control-based applications.

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were output ports.

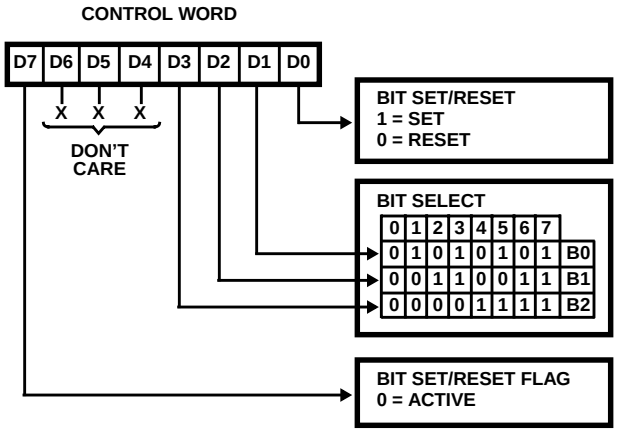


FIGURE 5. BIT SET/RESET FORMAT

Interrupt Control Functions

When the 82C55A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the bit set/reset function of port C.

This function allows the programmer to enable or disable a CPU interrupt by a specific I/O device without affecting any other device in the interrupt structure.

INTE Flip-Flop Definition

(BIT-SET)-INTE is SET - Interrupt Enable

(BIT-RESET)-INTE is Reset - Interrupt Disable

NOTE: All Mask flip-flops are automatically reset during mode selection and device Reset.

Operating Modes

Mode 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No handshaking is required, data is simply written to or read from a specific port.

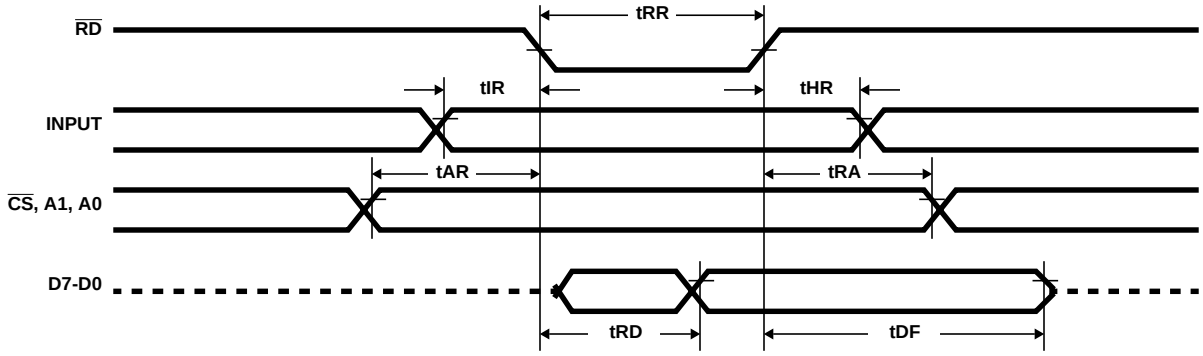
Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports
- Any Port can be input or output
- Outputs are latched
- Input are not latched
- 16 different Input/Output configurations possible

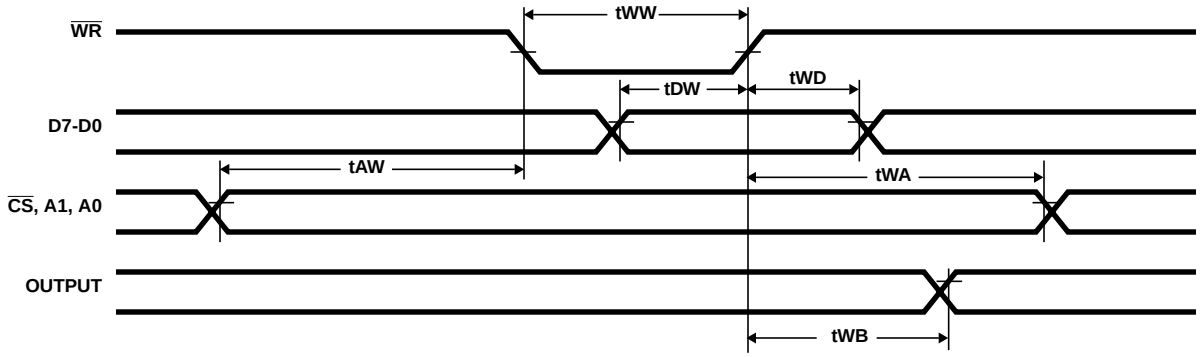
MODE 0 PORT DEFINITION

A		B		GROUP A			GROUP B	
D4	D3	D1	D0	PORT A	PORT C (Upper)	#	PORT B	PORT C (Lower)
0	0	0	0	Output	Output	0	Output	Output
0	0	0	1	Output	Output	1	Output	Input
0	0	1	0	Output	Output	2	Input	Output
0	0	1	1	Output	Output	3	Input	Input
0	1	0	0	Output	Input	4	Output	Output
0	1	0	1	Output	Input	5	Output	Input
0	1	1	0	Output	Input	6	Input	Output
0	1	1	1	Output	Input	7	Input	Input
1	0	0	0	Input	Output	8	Output	Output
1	0	0	1	Input	Output	9	Output	Input
1	0	1	0	Input	Output	10	Input	Output
1	0	1	1	Input	Output	11	Input	Input
1	1	0	0	Input	Input	12	Output	Output
1	1	0	1	Input	Input	13	Output	Input
1	1	1	0	Input	Input	14	Input	Output
1	1	1	1	Input	Input	15	Input	Input

Mode 0 (Basic Input)



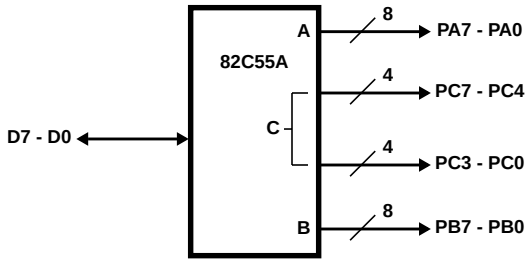
Mode 0 (Basic Output)



Mode 0 Configurations

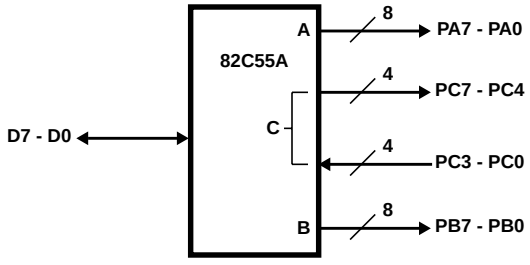
CONTROL WORD #0

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0



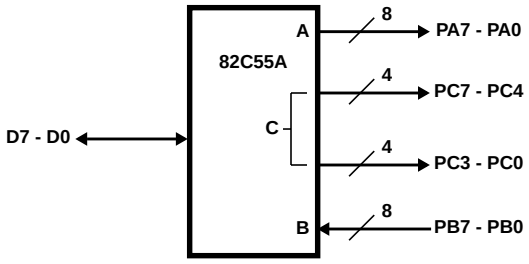
CONTROL WORD #1

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	1



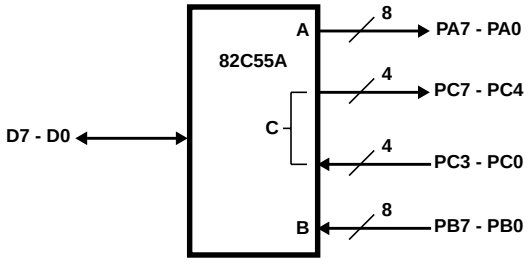
CONTROL WORD #2

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	1	0



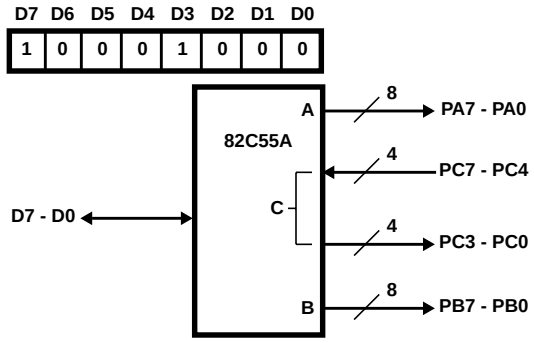
CONTROL WORD #3

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	1	1

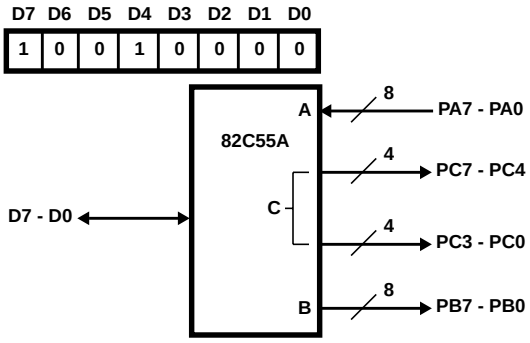


Mode 0 Configurations (Continued)

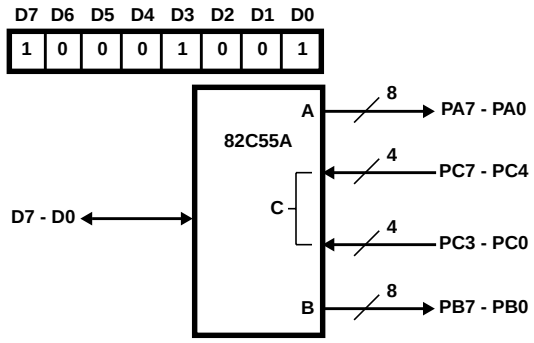
CONTROL WORD #4



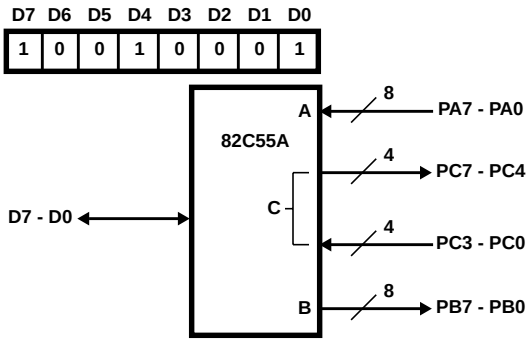
CONTROL WORD #8



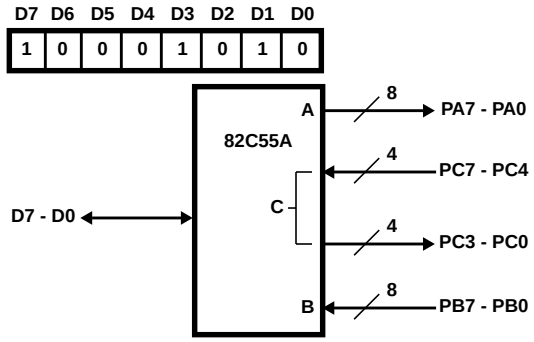
CONTROL WORD #5



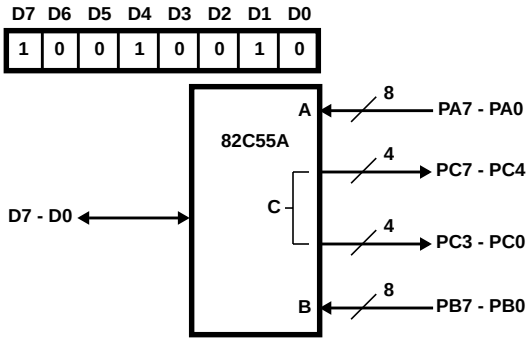
CONTROL WORD #9



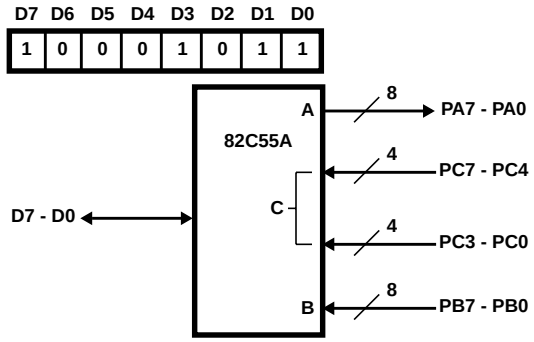
CONTROL WORD #6



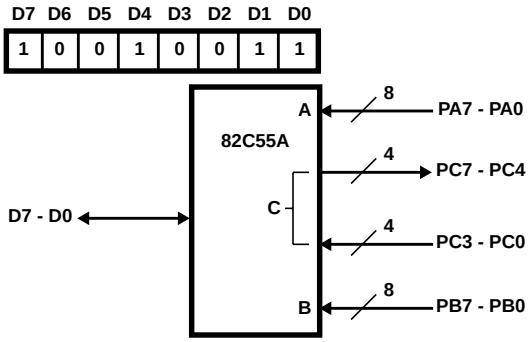
CONTROL WORD #10



CONTROL WORD #7

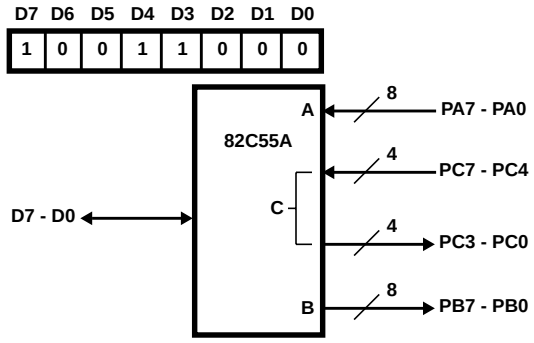


CONTROL WORD #11

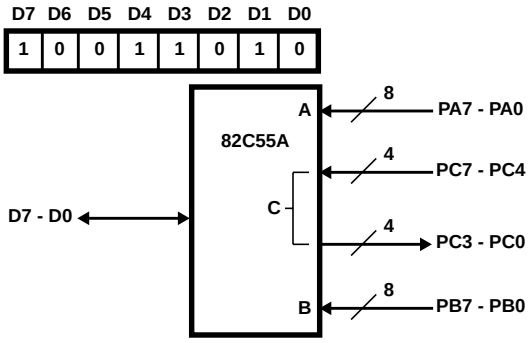


Mode 0 Configurations (Continued)

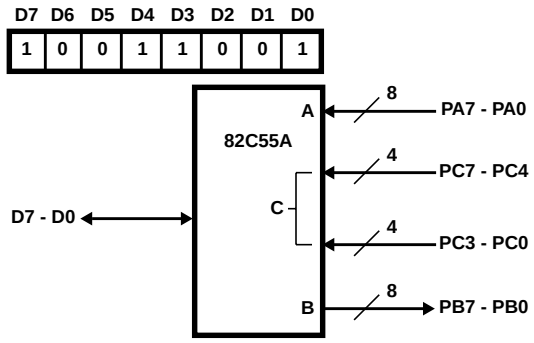
CONTROL WORD #12



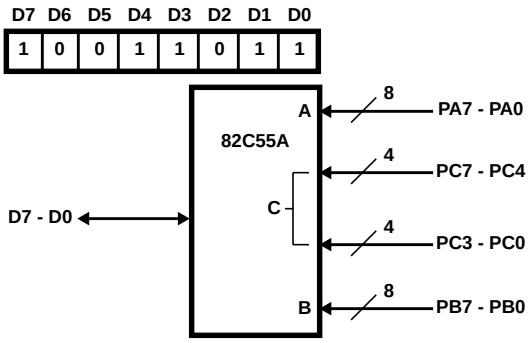
CONTROL WORD #14



CONTROL WORD #13



CONTROL WORD #15



Operating Modes

Mode 1 - (Strobed Input/Output). This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or “hand shaking” signals. In mode 1, port A and port B use the lines on port C to generate or accept these “hand shaking” signals.

Mode 1 Basic Function Definitions:

- Two Groups (Group A and Group B)
- Each group contains one 8-bit port and one 4-bit control/data port
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit port.

Input Control Signal Definition

(Figures 6 and 7)

STB (Strobe Input)

A “low” on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A “high” on this output indicates that the data has been loaded into the input latch: in essence, and acknowledgment. IBF is set by $\overline{STB}$ input being low and is reset by the rising edge of the $\overline{RD}$ input.

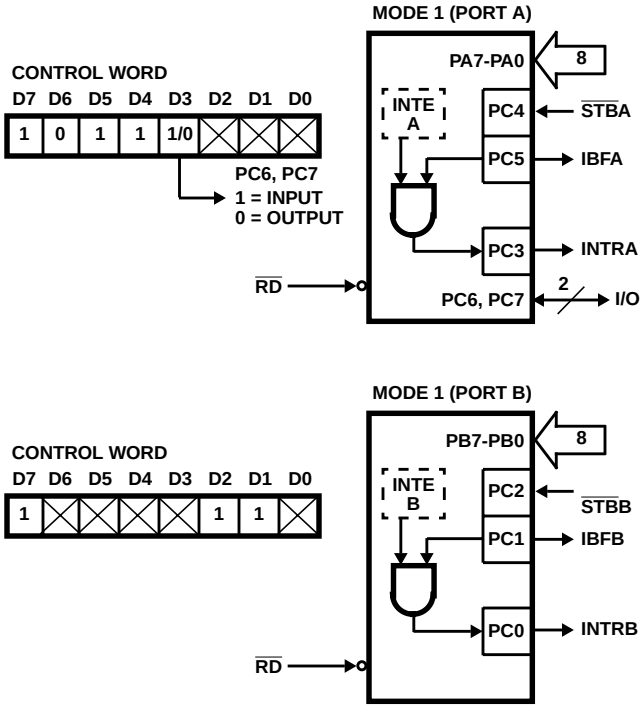


FIGURE 6. MODE 1 INPUT

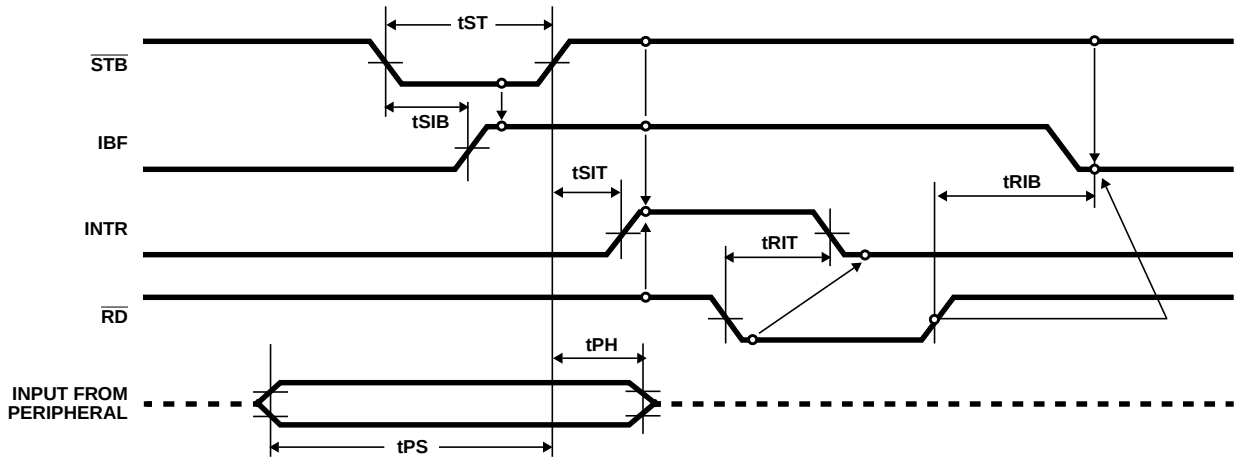


FIGURE 7. MODE 1 (STROBED INPUT)

INTR (Interrupt Request)

A “high” on this output can be used to interrupt the CPU when and input device is requesting service. INTR is set by the condition: $\overline{STB}$ is a “one”, IBF is a “one” and INTE is a “one”. It is reset by the falling edge of $\overline{RD}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A

Controlled by bit set/reset of PC4.

INTE B

Controlled by bit set/reset of PC2.

Output Control Signal Definition

(Figure 8 and 9)

$\overline{OBF}$ - Output Buffer Full F/F). The $\overline{OBF}$ output will go “low” to indicate that the CPU has written data out to be specified port. This does not mean valid data is sent out of the part at this time since $\overline{OBF}$ can go true before data is available. Data is guaranteed valid at the rising edge of $\overline{OBF}$, (See Note 1). The $\overline{OBF}$ F/F will be set by the rising edge of the $\overline{WR}$ input and reset by $\overline{ACK}$ input being low.

$\overline{ACK}$ - Acknowledge Input). A “low” on this input informs the 82C55A that the data from Port A or Port B is ready to be accepted. In essence, a response from the peripheral device indicating that it is ready to accept data, (See Note 1).

INTR - (Interrupt Request). A “high” on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when $\overline{ACK}$ is a “one”, OBF is a “one” and INTE is a “one”. It is reset by the falling edge of $\overline{WR}$.

INTE A

Controlled by Bit Set/Reset of PC6.

INTE B

Controlled by Bit Set/Reset of PC2.

NOTE:

1. To strobe data into the peripheral device, the user must operate the strobe line in a hand shaking mode. The user needs to send $\overline{OBF}$ to the peripheral device, generates an $\overline{ACK}$ from the peripheral device and then latch data into the peripheral device on the rising edge of $\overline{OBF}$.

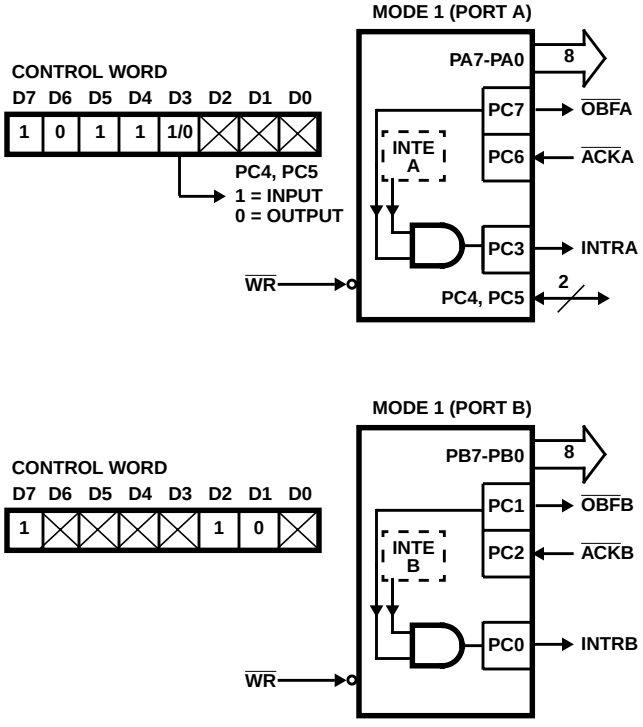


FIGURE 8. MODE 1 OUTPUT

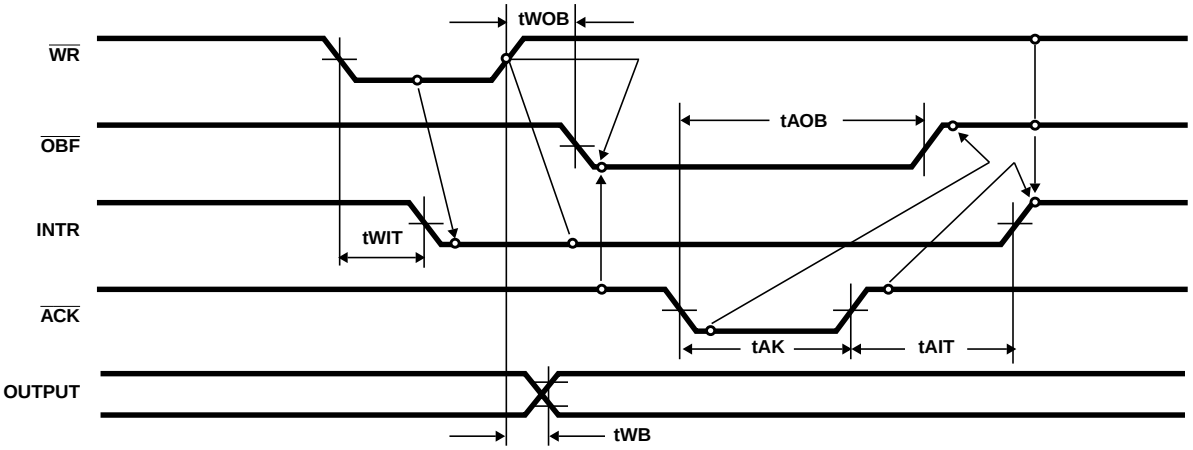
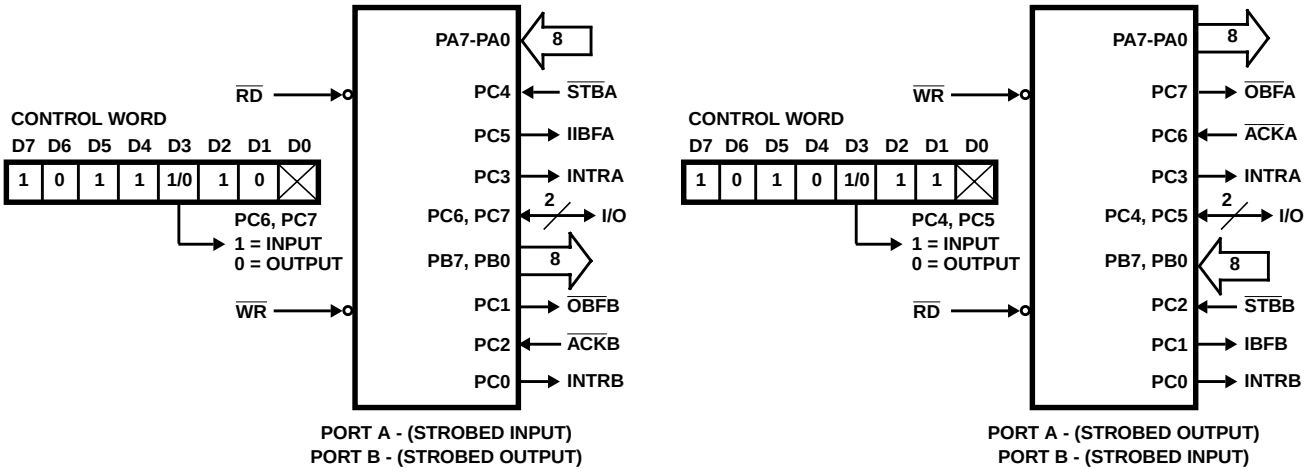


FIGURE 9. MODE 1 (STROBED OUTPUT)



Combinations of Mode 1: Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.

FIGURE 10. COMBINATIONS OF MODE 1

Operating Modes

Mode 2 (Strobed Bi-Directional Bus I/O)

The functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bi-directional bus I/O). “Hand shaking” signals are provided to maintain proper bus flow discipline similar to Mode 1. Interrupt generation and enable/disable functions are also available.

Mode 2 Basic Functional Definitions:

- Used in Group A only
- One 8-bit, bi-directional bus Port (Port A) and a 5-bit control Port (Port C)
- Both inputs and outputs are latched
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A)

Bi-Directional Bus I/O Control Signal Definition

(Figures 11, 12, 13, 14)

INTR - (Interrupt Request). A high on this output can be used to interrupt the CPU for both input or output operations.

Output Operations

$\overline{\text{OBF}}$ - (Output Buffer Full). The $\overline{\text{OBF}}$ output will go “low” to indicate that the CPU has written data out to port A.

$\overline{\text{ACK}}$ - (Acknowledge). A “low” on this input enables the three-state output buffer of port A to send out the data. Otherwise, the output buffer will be in the high impedance state.

INTE 1 - (The INTE flip-flop associated with $\overline{\text{OBF}}$). Controlled by bit set/reset of PC4.

Input Operations

$\overline{\text{STB}}$ - (Strobe Input). A “low” on this input loads data into the input latch.

IBF - (Input Buffer Full F/F). A “high” on this output indicates that data has been loaded into the input latch.

INTE 2 - (The INTE flip-flop associated with IBF). Controlled by bit set/reset of PC4.

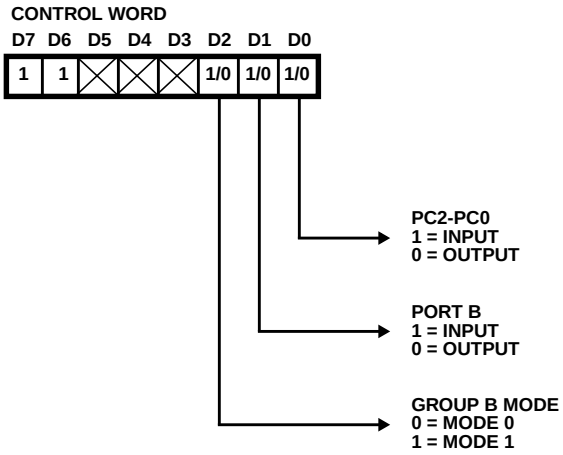


FIGURE 11. MODE CONTROL WORD

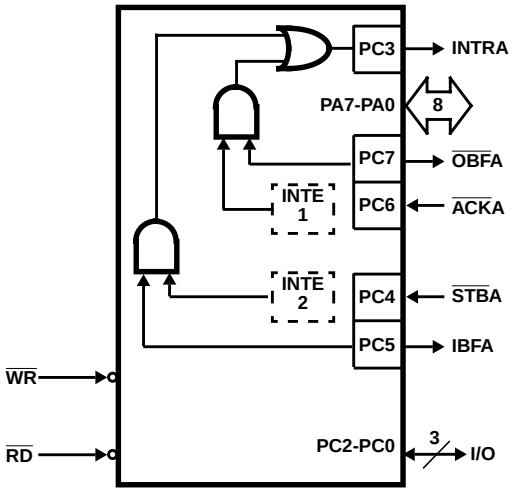
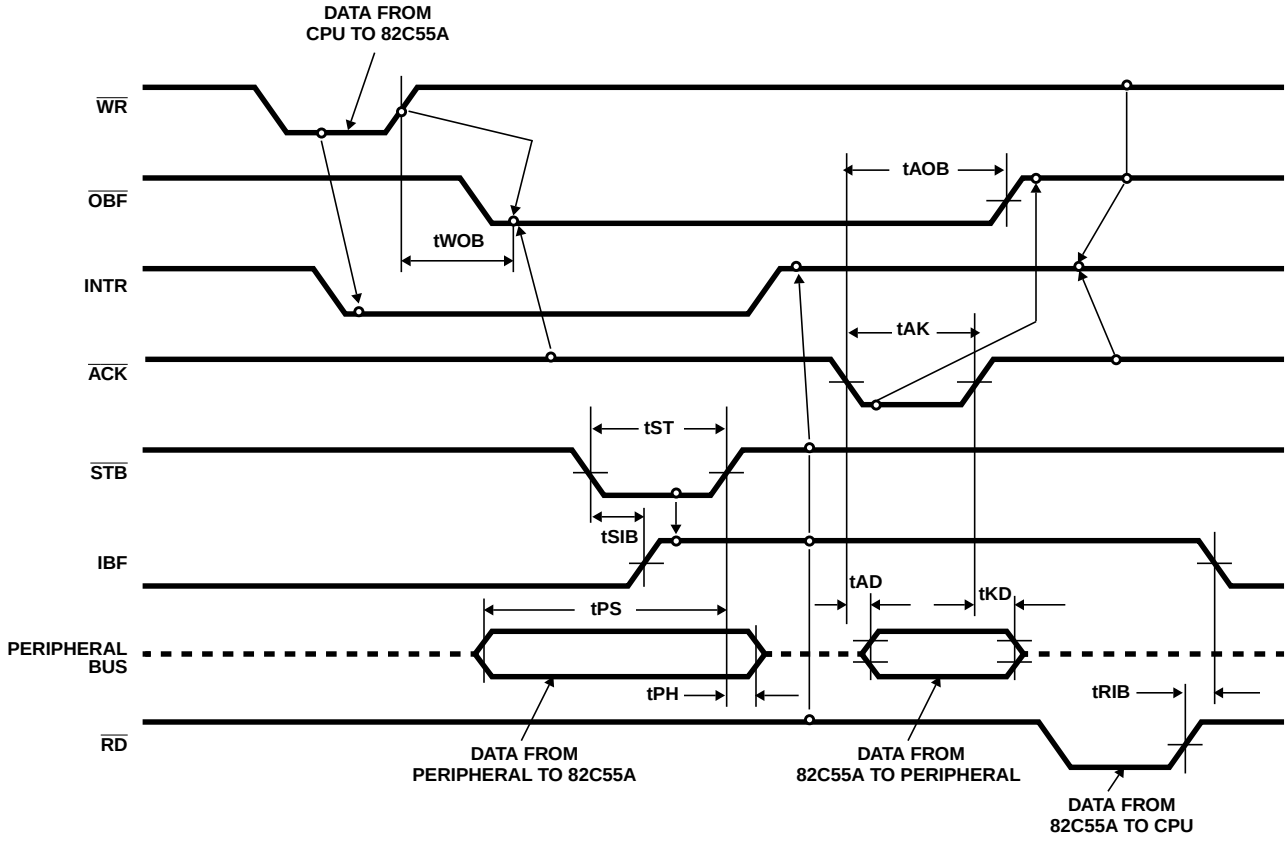


FIGURE 12. MODE 2



NOTE: Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before RD is permissible. (INTR = IBF • MASK • $\overline{STB}$ • $\overline{RD}$ ÷ $\overline{OBF}$ • MASK • $\overline{ACK}$ • $\overline{WR}$)

FIGURE 13. MODE 2 (BI-DIRECTIONAL)

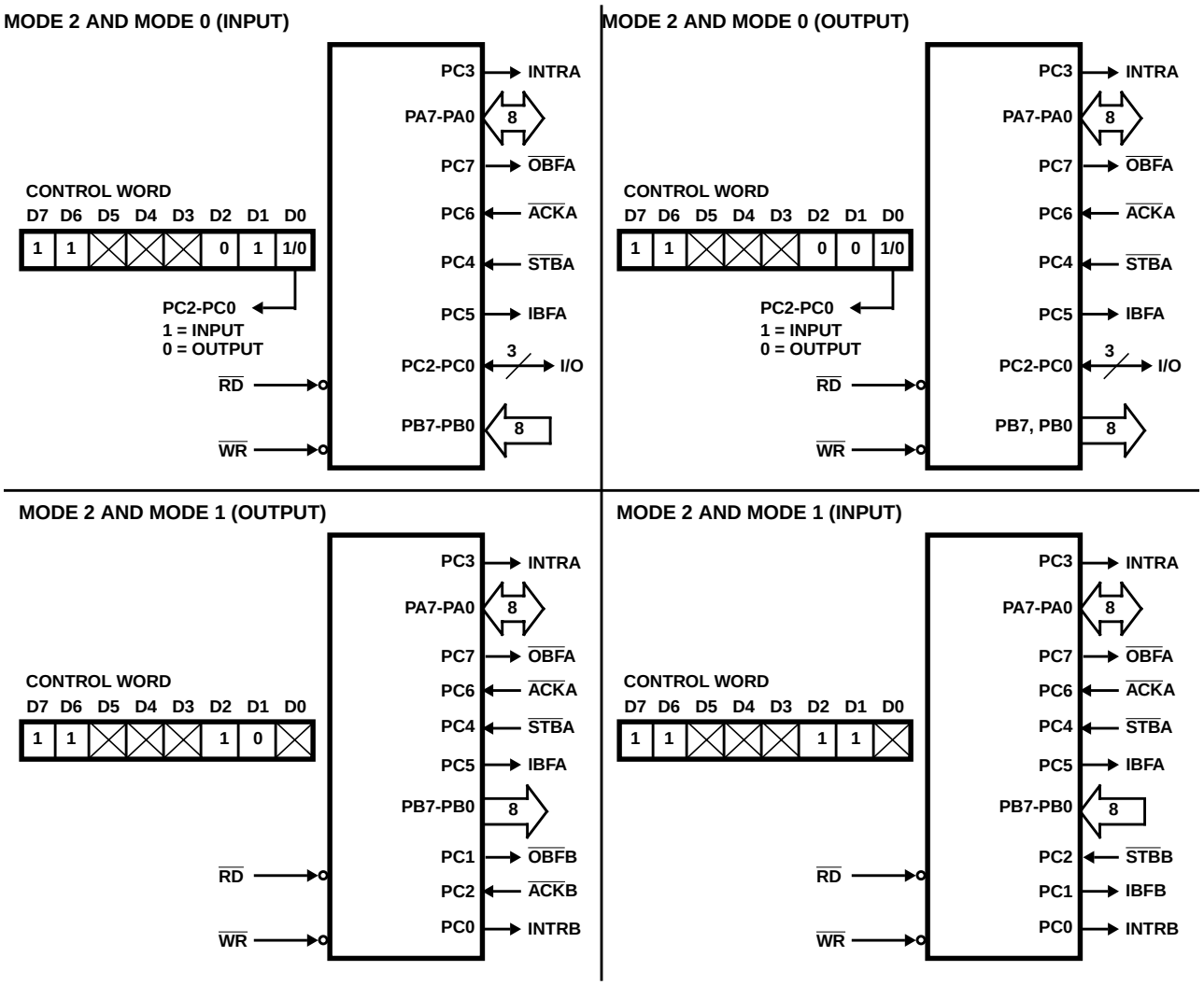


FIGURE 14. MODE 2 COMBINATIONS

MODE DEFINITION SUMMARY

	MODE 0		MODE 1		MODE 2	
	IN	OUT	IN	OUT	GROUP A ONLY	
PA0	In	Out	In	Out	← →	
PA1	In	Out	In	Out	← →	
PA2	In	Out	In	Out	← →	
PA3	In	Out	In	Out	← →	
PA4	In	Out	In	Out	← →	
PA5	In	Out	In	Out	← →	
PA6	In	Out	In	Out	← →	
PA7	In	Out	In	Out	← →	
PB0	In	Out	In	Out		Mode 0 or Mode 1 Only
PB1	In	Out	In	Out		
PB2	In	Out	In	Out		
PB3	In	Out	In	Out		
PB4	In	Out	In	Out		
PB5	In	Out	In	Out		
PB6	In	Out	In	Out		
PB7	In	Out	In	Out		
PC0	In	Out	INTRB	INTRB	I/O	
PC1	In	Out	IBFB	$\overline{OBF}$B	I/O	
PC2	In	Out	$\overline{STB}$B	$\overline{ACK}$B	I/O	
PC3	In	Out	INTRA	INTRA	INTRA	
PC4	In	Out	$\overline{STB}$A	I/O	$\overline{STB}$A	
PC5	In	Out	IBFA	I/O	IBFA	
PC6	In	Out	I/O	$\overline{ACK}$A	$\overline{ACK}$A	
PC7	In	Out	I/O	$\overline{OBF}$A	$\overline{OBF}$A	

Special Mode Combination Considerations

There are several combinations of modes possible. For any combination, some or all of Port C lines are used for control or status. The remaining bits are either inputs or outputs as defined by a “Set Mode” command.

During a read of Port C, the state of all the Port C lines, except the $\overline{ACK}$ and $\overline{STB}$ lines, will be placed on the data bus. In place of the $\overline{ACK}$ and $\overline{STB}$ line states, flag status will appear on the data bus in the PC2, PC4, and PC6 bit positions as illustrated by Figure 17.

Through a “Write Port C” command, only the Port C pins programmed as outputs in a Mode 0 group can be written. No other pins can be affected by a “Write Port C” command, nor can the interrupt enable flags be accessed. To write to any Port C output programmed as an output in Mode 1 group or to change an interrupt enable flag, the “Set/Reset Port C Bit” command must be used.

With a “Set/Reset Port Cea Bit” command, any Port C line programmed as an output (including IBF and $\overline{OBF}$) can be written, or an interrupt enable flag can be either set or reset. Port C lines programmed as inputs, including $\overline{ACK}$ and $\overline{STB}$ lines, associated with Port C fare not affected by a “Set/Reset Port C Bit” command. Writing to the corresponding Port C bit positions of the $\overline{ACK}$ and $\overline{STB}$ lines with the “Set Reset Port C Bit” command will affect the Group A and Group B interrupt enable flags, as illustrated in Figure 17.

INPUT CONFIGURATION

D7	D6	D5	D4	D3	D2	D1	D0
I/O	I/O	IBFA	INTEA	INTRA	INTEB	IBFB	INTRB
GROUP A					GROUP B		

OUTPUT CONFIGURATION

D7	D6	D5	D4	D3	D2	D1	D0
OBFA	INTEA	I/O	I/O	INTRA	INTEB	OBFB	INTRB
GROUP A					GROUP B		

FIGURE 15. MODE 1 STATUS WORD FORMAT

D7	D6	D5	D4	D3	D2	D1	D0
OBFA	INTE1	IBFA	INTE2	INTRA	X	X	X
GROUP A					GROUP B		

(Defined by Mode 0 or Mode 1 Selection)

FIGURE 16. MODE 2 STATUS WORD FORMAT

Current Drive Capability

Any output on Port A, B or C can sink or source 2.5mA. This feature allows the 82C55A to directly drive Darlington type drivers and high-voltage displays that require such sink or source current.

Reading Port C Status (Figures 15 and 16)

In Mode 0, Port C transfers data to or from the peripheral device. When the 82C55A is programmed to function in Modes 1 or 2, Port C generates or accepts “hand shaking” signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the “status” of each peripheral device and change the program flow accordingly.

There is not special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.

INTERRUPT ENABLE FLAG	POSITION	ALTERNATE PORT C PIN SIGNAL (MODE)
INTE B	PC2	$\overline{ACK}$B (Output Mode 1) or $\overline{STB}$B (Input Mode 1)
INTE A2	PC4	$\overline{STB}$A (Input Mode 1 or Mode 2)
INTE A1	PC6	$\overline{ACK}$A (Output Mode 1 or Mode 2)

FIGURE 17. INTERRUPT ENABLE FLAGS IN MODES 1 AND 2

Applications of the 82C55A

The 82C55A is a very powerful tool for interfacing peripheral equipment to the microcomputer system. It represents the optimum use of available pins and flexible enough to interface almost any I/O device without the need for additional external logic.

Each peripheral device in a microcomputer system usually has a “service routine” associated with it. The routine manages the software interface between the device and the CPU. The functional definition of the 82C55A is programmed by the I/O service routine and becomes an extension of the system software. By examining the I/O devices interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the detailed operational description, a control word can easily be developed to initialize the 82C55A to exactly “fit” the application. Figures 18 through 24 present a few examples of typical applications of the 82C55A.

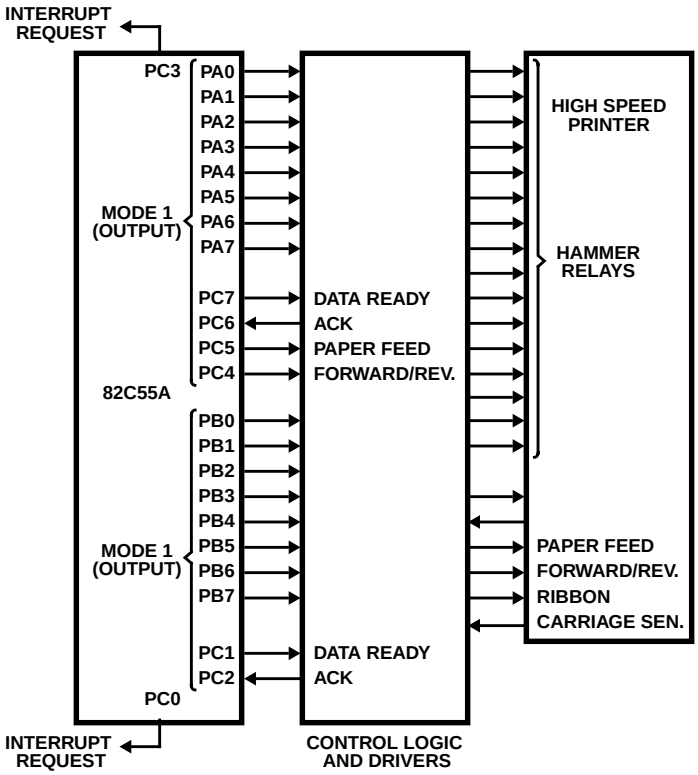


FIGURE 18. PRINTER INTERFACE

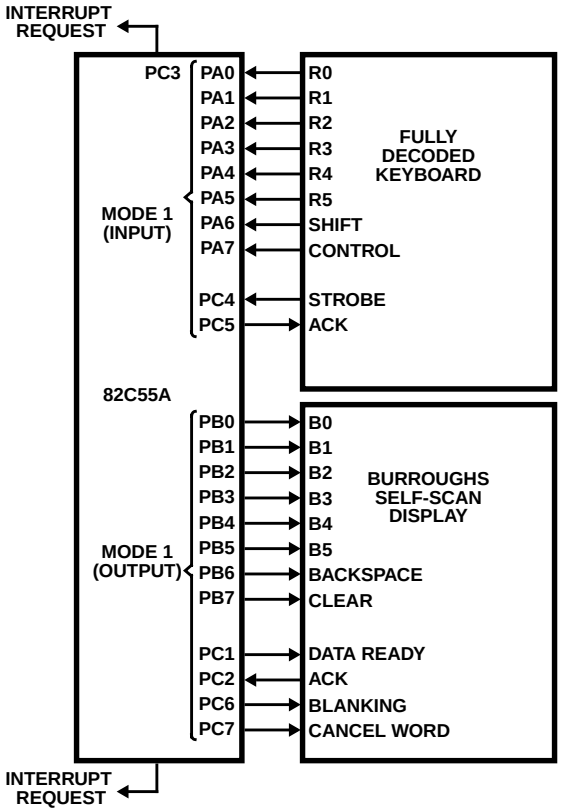


FIGURE 19. KEYBOARD AND DISPLAY INTERFACE

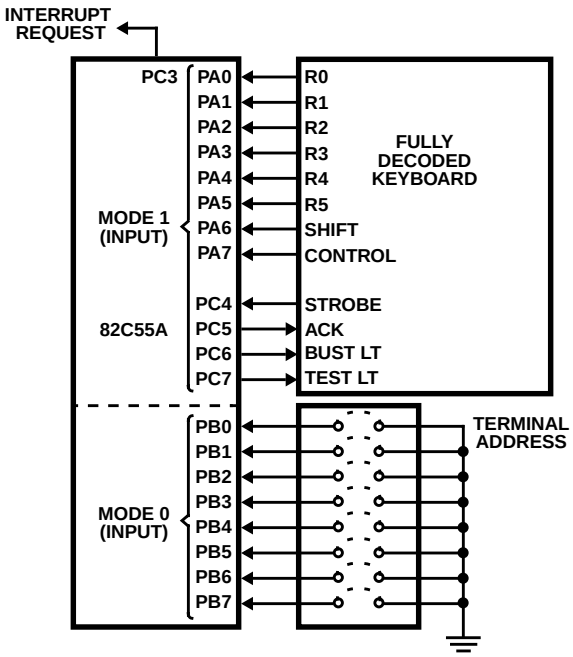


FIGURE 20. KEYBOARD AND TERMINAL ADDRESS INTERFACE

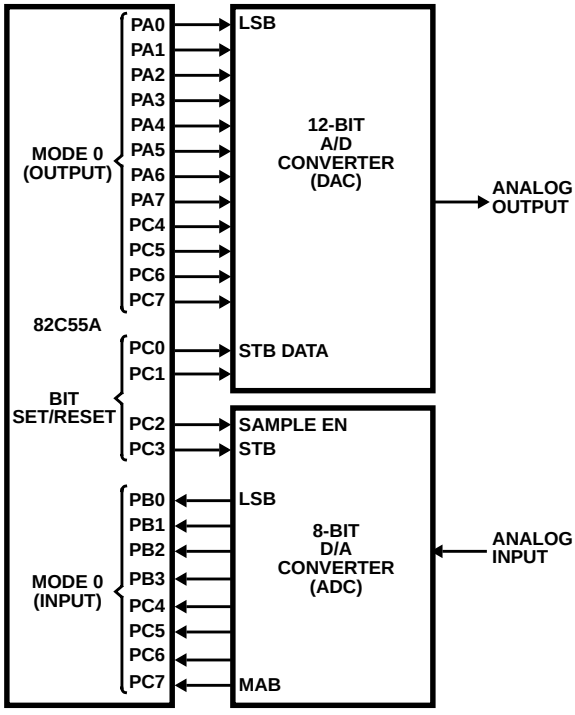


FIGURE 21. DIGITAL TO ANALOG, ANALOG TO DIGITAL

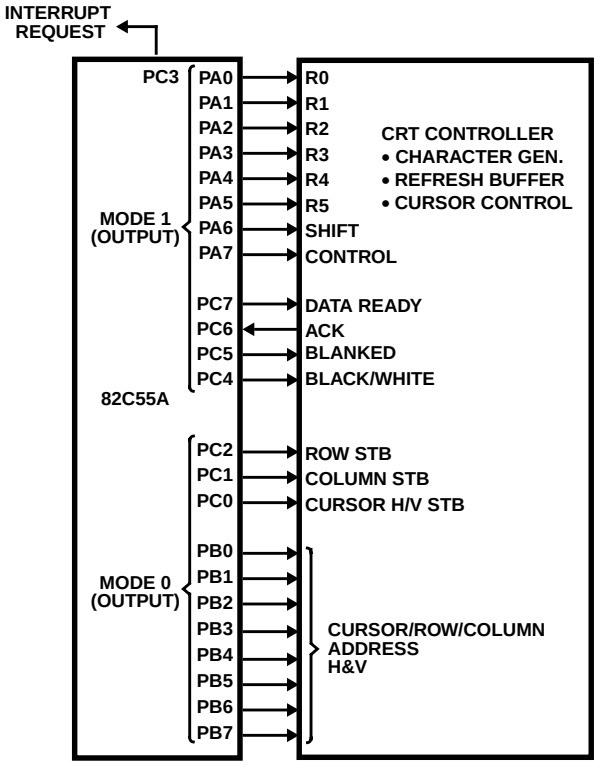


FIGURE 22. BASIC CRT CONTROLLER INTERFACE

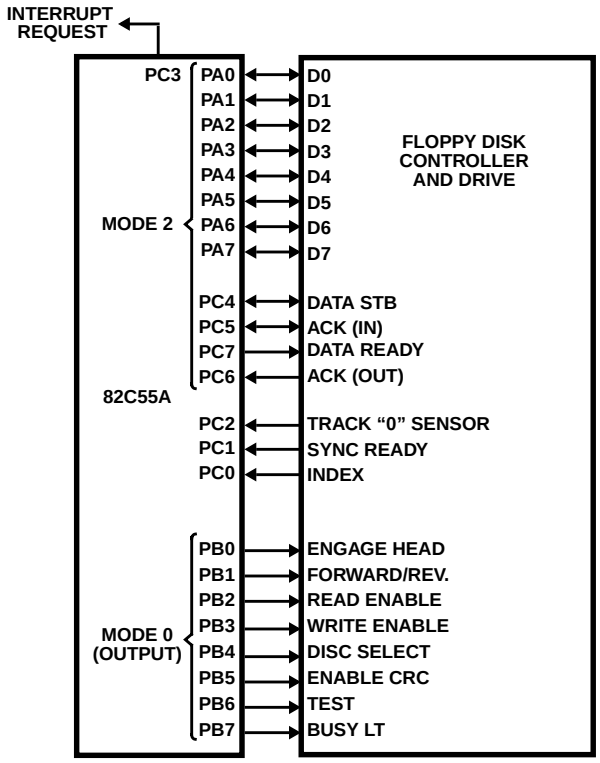


FIGURE 23. BASIC FLOPPY DISC INTERFACE

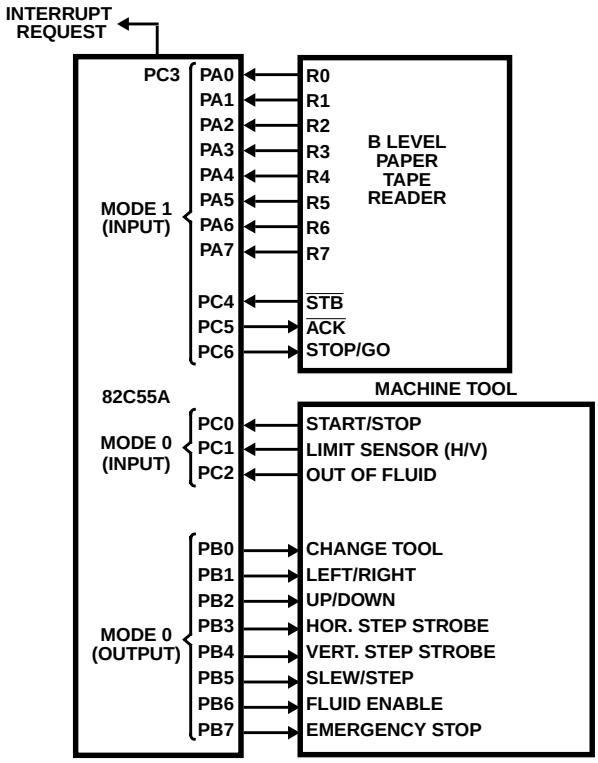


FIGURE 24. MACHINE TOOL CONTROLLER INTERFACE

Absolute Maximum Ratings $T_A = 25^oC$

Supply Voltage . . . . . +8.0V
Input, Output or I/O Voltage . . . . . GND-0.5V to V_{CC}+0.5V
ESD Classification . . . . . Class 1

Operating Conditions

Voltage Range . . . . . +4.5V to 5.5V
Operating Temperature Range
C82C55A . . . . . 0^oC to 70^oC
I82C55A . . . . . -40^oC to 85^oC
M82C55A . . . . . -55^oC to 125^oC

Thermal Information

Thermal Resistance (Typical, Note 1)	θ_{JA}	θ_{JC}
CERDIP Package	$50^oC/W$	$10^oC/W$
CLCC Package	$65^oC/W$	$14^oC/W$
PDIP Package	$50^oC/W$	N/A
PLCC Package	$46^oC/W$	N/A

Maximum Storage Temperature Range . . . . . -65^oC to 150^oC
Maximum Junction Temperature
CDIP Package . . . . . 175^oC
PDIP Package . . . . . 150^oC
Maximum Lead Temperature (Soldering 10s) . . . . . 300^oC
(PLCC Lead Tips Only)

Die Characteristics

Gate Count . . . . . 1000 Gates

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTE:

1. θ_{JA} is measured with the component mounted on an evaluation PC board in free air.

Electrical Specifications

V_{CC} = 5.0V ±10%; $T_A = 0^oC$ to $+70^oC$ (C82C55A);
$T_A = -40^oC$ to $+85^oC$ (I82C55A);
$T_A = -55^oC$ to $+125^oC$ (M82C55A)

SYMBOL	PARAMETER	LIMITS		UNITS	TEST CONDITIONS
		MIN	MAX		
V_{IH}	Logical One Input Voltage	2.0 2.2	-	V	I82C55A, C82C55A, M82C55A
V_{IL}	Logical Zero Input Voltage	-	0.8	V	
V_{OH}	Logical One Output Voltage	3.0 V_{CC} -0.4	-	V	I_{OH} = -2.5mA, I_{OH} = -100µA
V_{OL}	Logical Zero Output Voltage	-	0.4	V	I_{OL} +2.5mA
I_I	Input Leakage Current	-1.0	+1.0	µA	$V_{IN} = V_{CC}$ or GND, DIP Pins: 5, 6, 8, 9, 35, 36
IO	I/O Pin Leakage Current	-10	+10	µA	VO = V_{CC} or GND DIP Pins: 27 - 34
IBHH	Bus Hold High Current	-50	-400	µA	VO = 3.0V. Ports A, B, C
IBHL	Bus Hold Low Current	50	400	µA	VO = 1.0V. Port A ONLY
IDAR	Darlington Drive Current	-2.5	Note 2, 4	mA	Ports A, B, C. Test Condition 3
ICCSB	Standby Power Supply Current	-	10	µA	V_{CC} = 5.5V, $V_{IN} = V_{CC}$ or GND. Output Open
ICCOP	Operating Power Supply Current	-	1	mA/MHz	$T_A = +25^oC$, V_{CC} = 5.0V, Typical (See Note 3)

NOTES:

2. No internal current limiting exists on Port Outputs. A resistor must be added externally to limit the current.
3. ICCOP = 1mA/MHz of Peripheral Read/Write cycle time. (Example: 1.0µs I/O Read/Write cycle time = 1mA).
4. Tested as V_{OH} at -2.5mA.

Capacitance $T_A = 25^oC$

SYMBOL	PARAMETER	TYPICAL	UNITS	TEST CONDITIONS
CIN	Input Capacitance	10	pF	FREQ = 1MHz, All Measurements are referenced to device GND
CI/O	I/O Capacitance	20	pF	

AC Electrical Specifications V_{CC} = +5V± 10%, GND = 0V; T_A = -55°C to +125°C (M82C55A) (M82C55A-5);
T_A = -40°C to +85°C (I82C55A) (I82C55A-5);
T_A = 0°C to +70°C (C82C55A) (C82C55A-5)

SYMBOL	PARAMETER	82C55A-5		82C55A		UNITS	TEST CONDITIONS
		MIN	MAX	MIN	MAX		
READ TIMING							
(1) tAR	Address Stable Before $\overline{RD}$	0	-	0	-	ns	
(2) tRA	Address Stable After $\overline{RD}$	0	-	0	-	ns	
(3) tRR	$\overline{RD}$ Pulse Width	250	-	150	-	ns	
(4) tRD	Data Valid From $\overline{RD}$	-	200	-	120	ns	1
(5) tDF	Data Float After $\overline{RD}$	10	75	10	75	ns	2
(6) tRV	Time Between $\overline{RD}$s and/or $\overline{WR}$s	300	-	300	-	ns	
WRITE TIMING							
(7) tAW	Address Stable Before $\overline{WR}$	0	-	0	-	ns	
(8) tWA	Address Stable After $\overline{WR}$	20	-	20	-	ns	
(9) tWW	$\overline{WR}$ Pulse Width	100	-	100	-	ns	
(10) tDW	Data Valid to $\overline{WR}$ High	100	-	100	-	ns	
(11) tWD	Data Valid After $\overline{WR}$ High	30	-	30	-	ns	
OTHER TIMING							
(12) tWB	$\overline{WR}$ = 1 to Output	-	350	-	350	ns	1
(13) tIR	Peripheral Data Before $\overline{RD}$	0	-	0	-	ns	
(14) tHR	Peripheral Data After $\overline{RD}$	0	-	0	-	ns	
(15) tAK	ACK Pulse Width	200	-	200	-	ns	
(16) tST	STB Pulse Width	100	-	100	-	ns	
(17) tPS	Peripheral Data Before STB High	20	-	20	-	ns	
(18) tPH	Peripheral Data After STB High	50	-	50	-	ns	
(19) tAD	ACK = 0 to Output	-	175	-	175	ns	1
(20) tKD	ACK = 1 to Output Float	20	250	20	250	ns	2
(21) tWOB	$\overline{WR}$ = 1 to OBF = 0	-	150	-	150	ns	1
(22) tAOB	ACK = 0 to OBF = 1	-	150	-	150	ns	1
(23) tSIB	STB = 0 to IBF = 1	-	150	-	150	ns	1
(24) tRIB	$\overline{RD}$ = 1 to IBF = 0	-	150	-	150	ns	1
(25) tRIT	$\overline{RD}$ = 0 to INTR = 0	-	200	-	200	ns	1
(26) tSIT	STB = 1 to INTR = 1	-	150	-	150	ns	1
(27) tAIT	ACK = 1 to INTR = 1	-	150	-	150	ns	1
(28) tWIT	$\overline{WR}$ = 0 to INTR = 0	-	200	-	200	ns	1
(29) tRES	Reset Pulse Width	500	-	500	-	ns	1, (Note)

NOTE: Period of initial Reset pulse after power-on must be at least 50µsec. Subsequent Reset pulses may be 500ns minimum.

Timing Waveforms

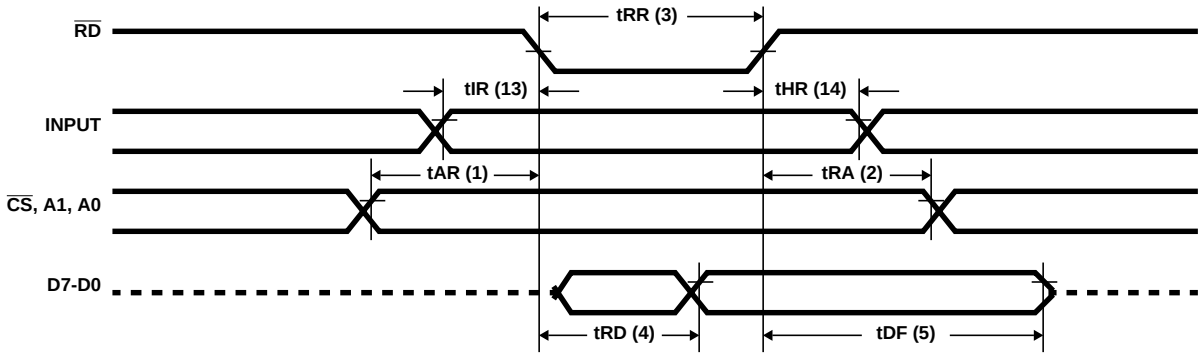


FIGURE 25. MODE 0 (BASIC INPUT)

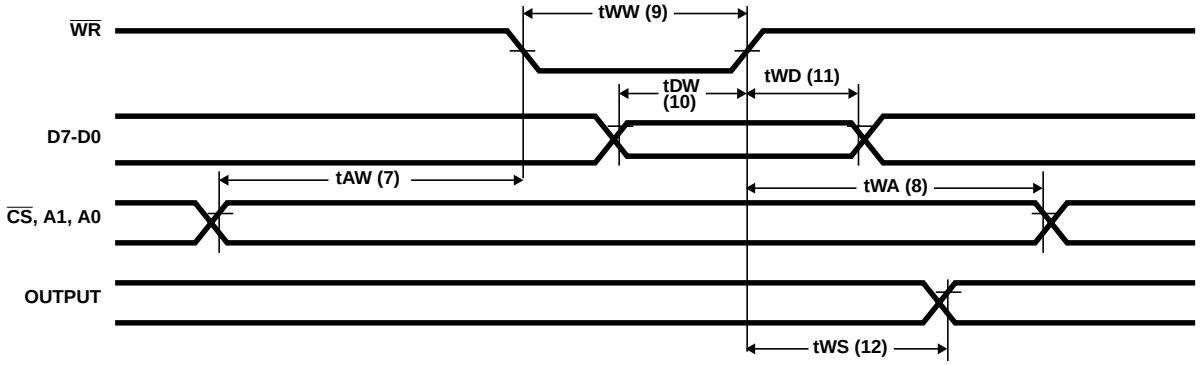


FIGURE 26. MODE 0 (BASIC OUTPUT)

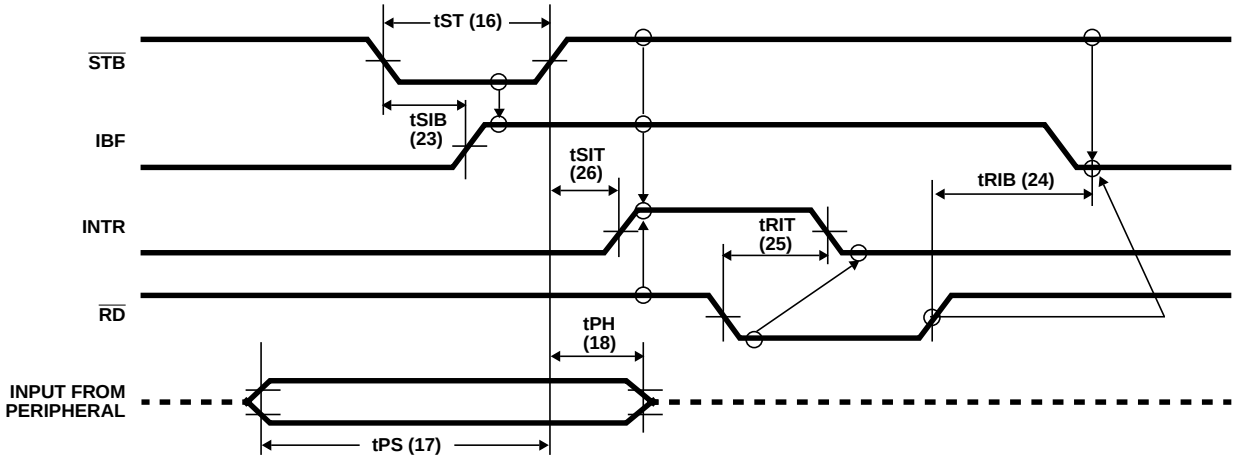


FIGURE 27. MODE 1 (STROBED INPUT)

Timing Waveforms (Continued)

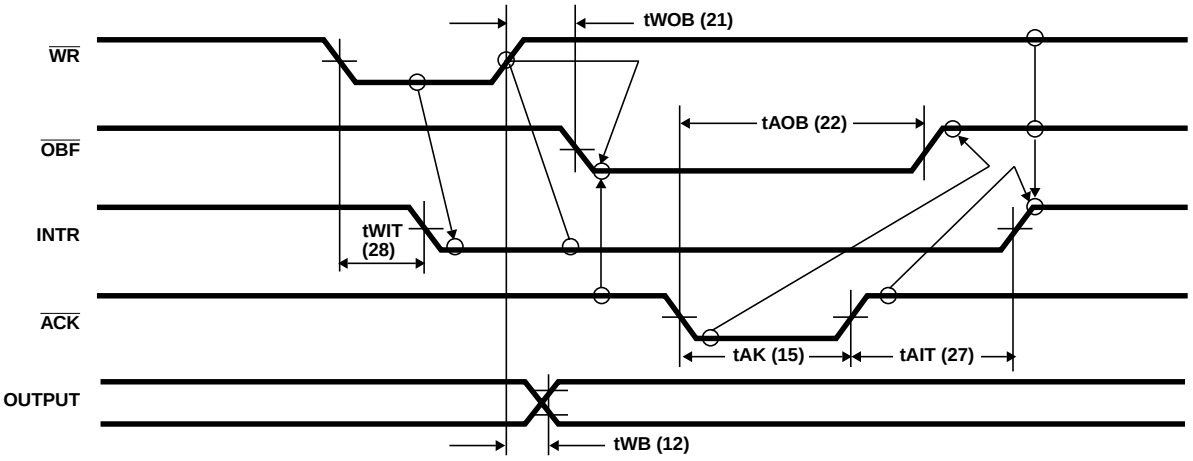


FIGURE 28. MODE 1 (STROBED OUTPUT)

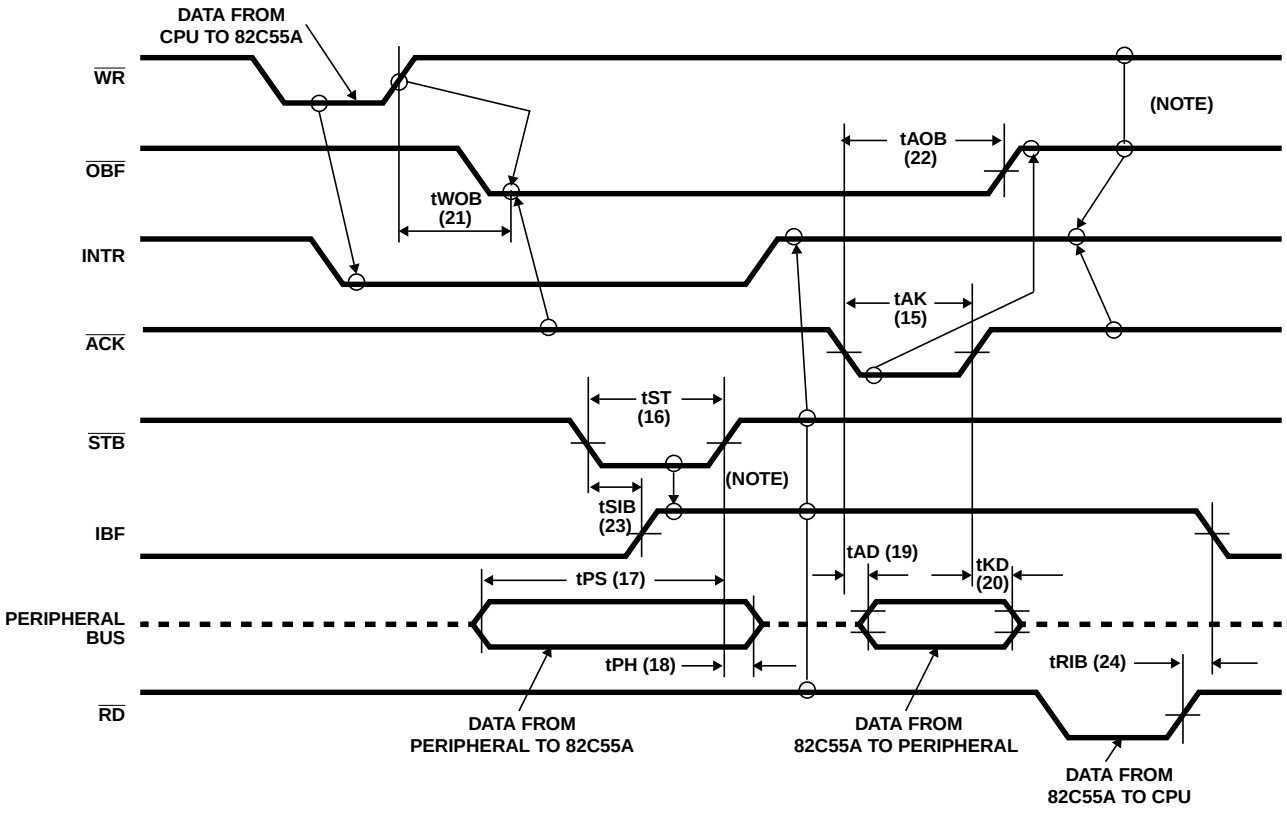


FIGURE 29. MODE 2 (BI-DIRECTIONAL)

NOTE: Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before $\overline{RD}$ is permissible. (INTR = IBF • $\overline{MASK}$ • $\overline{STB}$ • $\overline{RD}$ • $\overline{OBF}$ • $\overline{MASK}$ • $\overline{ACK}$ • $\overline{WR}$)

Timing Waveforms (Continued)

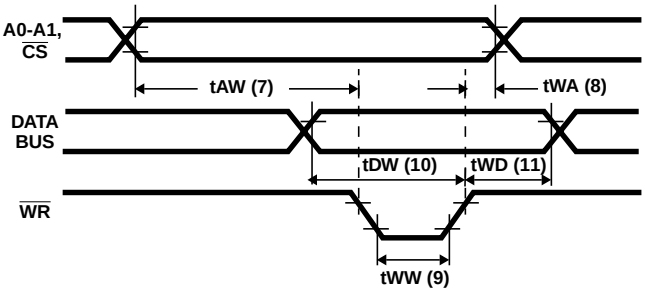


FIGURE 30. WRITE TIMING

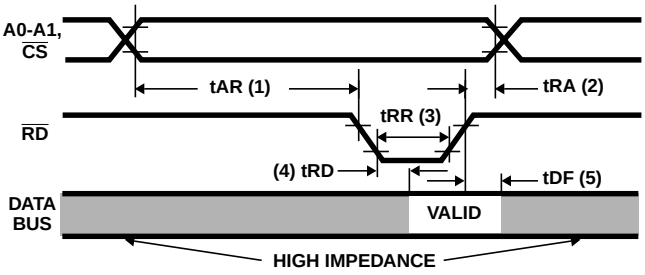
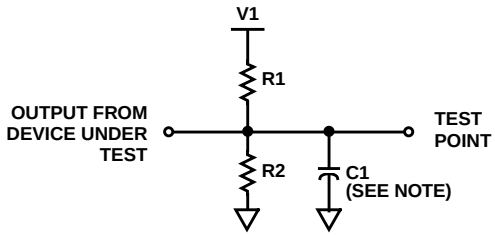


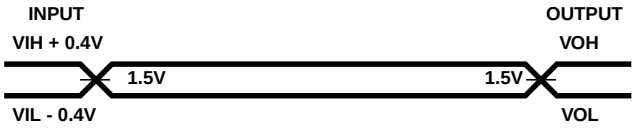
FIGURE 31. READ TIMING

AC Test Circuit



NOTE: Includes STRAY and JIG Capacitance

AC Testing Input, Output Waveforms



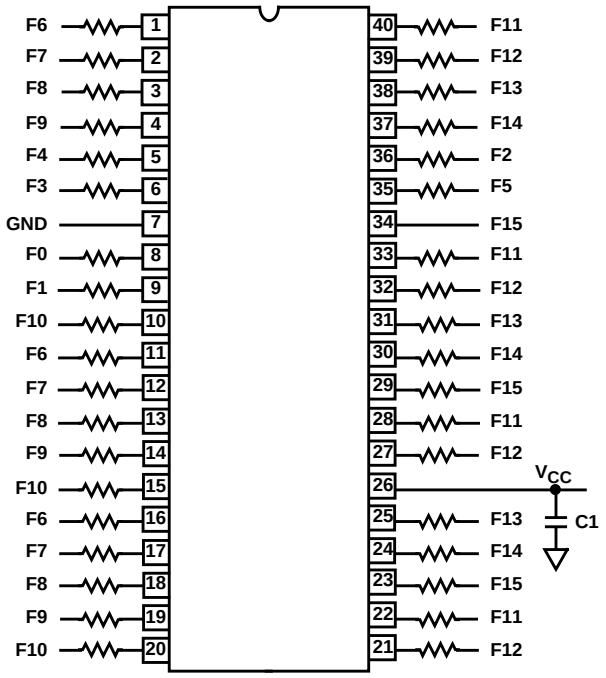
AC Testing: All AC Parameters tested as per test circuits. Input RISE and FALL times are driven at 1ns/V.

TEST CONDITION DEFINITION TABLE

TEST CONDITION	V1	R1	R2	C1
1	1.7V	523Ω	Open	150pF
2	V_{CC}	2kΩ	1.7kΩ	50pF
3	1.5V	750Ω	Open	50pF

Burn-In Circuits

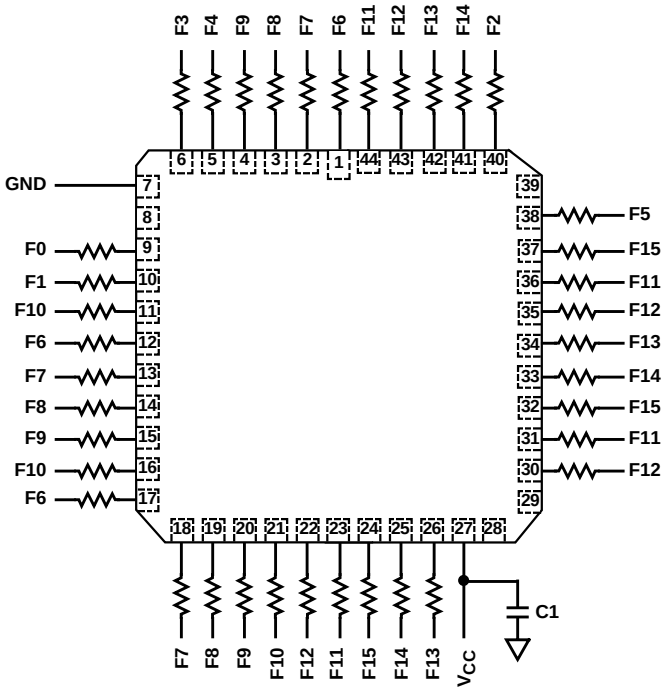
MD82C55A CERDIP



NOTES:

1. V_{CC} = 5.5V ± 0.5V
2. VIH = 4.5V ± 10%
3. VIL = -0.2V to 0.4V
4. GND = 0V

MR82C55A CLCC



NOTES:

1. C1 = 0.01µF minimum
2. All resistors are 47kΩ ± 5%
3. f0 = 100kHz ± 10%
4. f1 = f0 ÷ 2; f2 = f1 ÷ 2; . . . ; f15 = f14 ÷ 2

Die Characteristics

DIE DIMENSIONS:
95 x 100 x 19 ±1mils

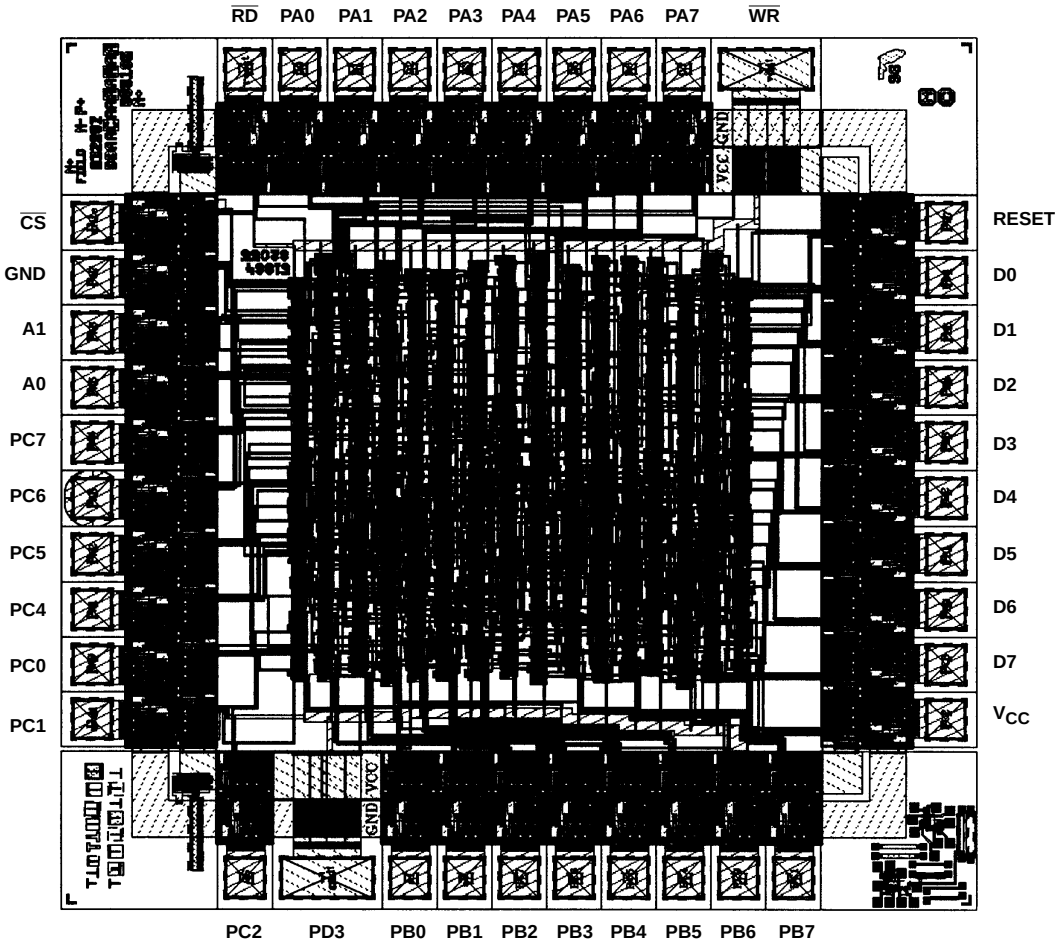
METALLIZATION:
Type: Silicon - Aluminum
Thickness: 11kÅ ±1kÅ

GLASSIVATION:
Type: SiO_2
Thickness: 8kÅ ±1kÅ

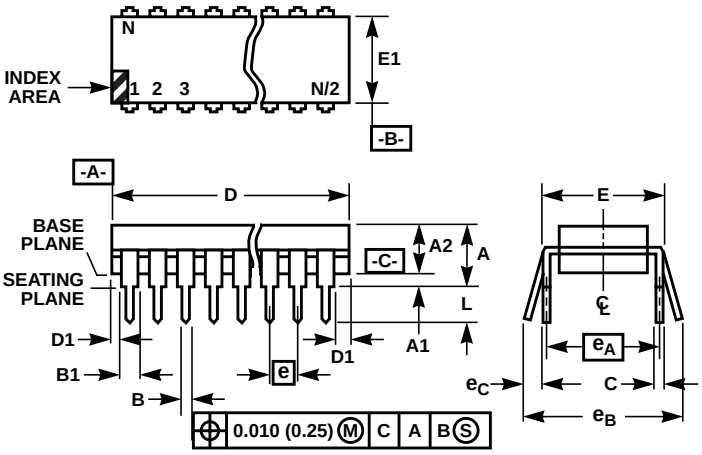
WORST CASE CURRENT DENSITY:
0.78×10^5 A/cm^2

Metallization Mask Layout

82C55A



Dual-In-Line Plastic Packages (PDIP)



NOTES:

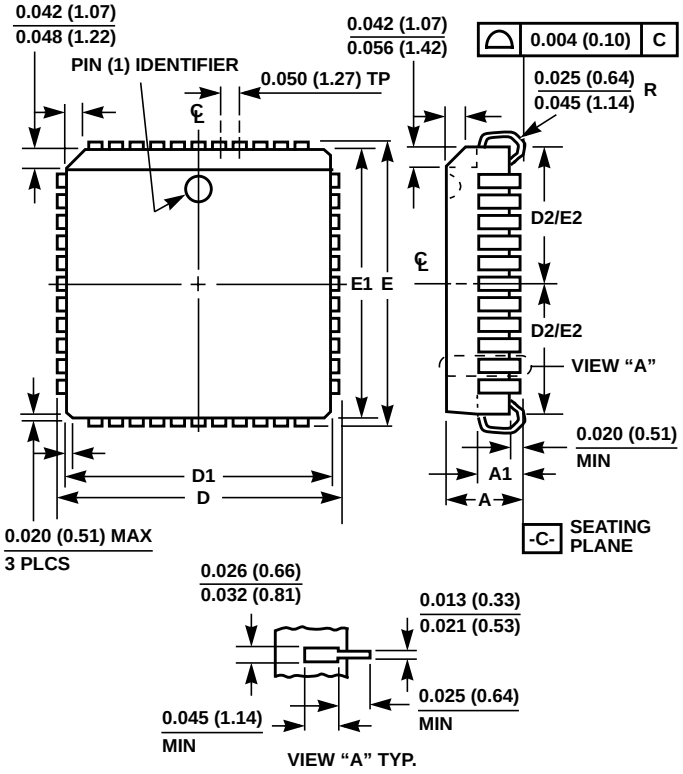
1. Controlling Dimensions: INCH. In case of conflict between English and Metric dimensions, the inch dimensions control.
2. Dimensioning and tolerancing per ANSI Y14.5M-1982.
3. Symbols are defined in the “MO Series Symbol List” in Section 2.2 of Publication No. 95.
4. Dimensions A, A1 and L are measured with the package seated in JEDEC seating plane gauge GS-3.
5. D, D1, and E1 dimensions do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.010 inch (0.25mm).
6. E and e_A are measured with the leads constrained to be perpendicular to datum -C-.
7. e_B and e_C are measured at the lead tips with the leads unconstrained. e_C must be zero or greater.
8. B1 maximum dimensions do not include dambar protrusions. Dambar protrusions shall not exceed 0.010 inch (0.25mm).
9. N is the maximum number of terminal positions.
10. Corner leads (1, N, N/2 and N/2 + 1) for E8.3, E16.3, E18.3, E28.3, E42.6 will have a B1 dimension of 0.030 - 0.045 inch (0.76 - 1.14mm).

E40.6 (JEDEC MS-011-AC ISSUE B)
40 LEAD DUAL-IN-LINE PLASTIC PACKAGE

SYMBOL	INCHES		MILLIMETERS		NOTES
	MIN	MAX	MIN	MAX	
A	-	0.250	-	6.35	4
A1	0.015	-	0.39	-	4
A2	0.125	0.195	3.18	4.95	-
B	0.014	0.022	0.356	0.558	-
B1	0.030	0.070	0.77	1.77	8
C	0.008	0.015	0.204	0.381	-
D	1.980	2.095	50.3	53.2	5
D1	0.005	-	0.13	-	5
E	0.600	0.625	15.24	15.87	6
E1	0.485	0.580	12.32	14.73	5
e	0.100 BSC		2.54 BSC		-
e_A	0.600 BSC		15.24 BSC		6
e_B	-	0.700	-	17.78	7
L	0.115	0.200	2.93	5.08	4
N	40		40		9

Rev. 0 12/93

Plastic Leaded Chip Carrier Packages (PLCC)



N44.65 **(JEDEC MS-018AC ISSUE A)**
44 LEAD PLASTIC LEADED CHIP CARRIER PACKAGE

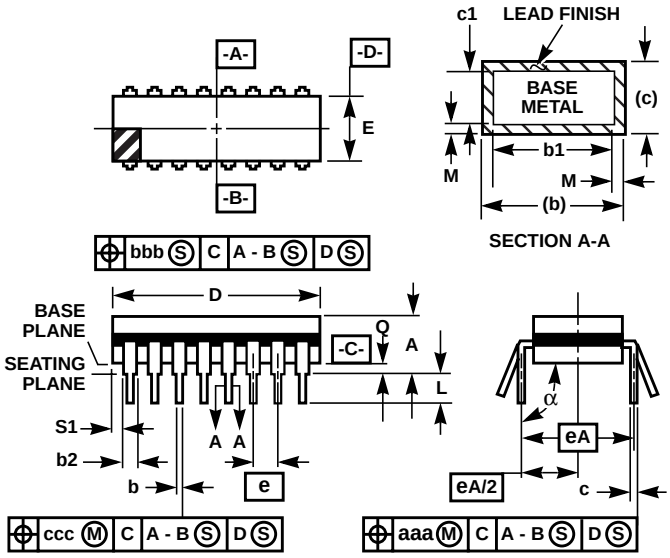
SYMBOL	INCHES		MILLIMETERS		NOTES
	MIN	MAX	MIN	MAX	
A	0.165	0.180	4.20	4.57	-
A1	0.090	0.120	2.29	3.04	-
D	0.685	0.695	17.40	17.65	-
D1	0.650	0.656	16.51	16.66	3
D2	0.291	0.319	7.40	8.10	4, 5
E	0.685	0.695	17.40	17.65	-
E1	0.650	0.656	16.51	16.66	3
E2	0.291	0.319	7.40	8.10	4, 5
N	44		44		6

Rev. 2 11/97

NOTES:

1. Controlling dimension: INCH. Converted millimeter dimensions are not necessarily exact.
2. Dimensions and tolerancing per ANSI Y14.5M-1982.
3. Dimensions D1 and E1 do not include mold protrusions. Allowable mold protrusion is 0.010 inch (0.25mm) per side. Dimensions D1 and E1 include mold mismatch and are measured at the extreme material condition at the body parting line.
4. To be measured at seating plane -C- contact point.
5. Centerline to be determined where center leads exit plastic body.
6. "N" is the number of terminal positions.

Ceramic Dual-In-Line Frit Seal Packages (CERDIP)



F40.6 MIL-STD-1835 GDIP1-T40 (D-5, CONFIGURATION A)
40 LEAD CERAMIC DUAL-IN-LINE FRIT SEAL PACKAGE

SYMBOL	INCHES		MILLIMETERS		NOTES
	MIN	MAX	MIN	MAX	
A	-	0.225	-	5.72	-
b	0.014	0.026	0.36	0.66	2
b1	0.014	0.023	0.36	0.58	3
b2	0.045	0.065	1.14	1.65	-
b3	0.023	0.045	0.58	1.14	4
c	0.008	0.018	0.20	0.46	2
c1	0.008	0.015	0.20	0.38	3
D	-	2.096	-	53.24	5
E	0.510	0.620	12.95	15.75	5
e	0.100 BSC		2.54 BSC		-
eA	0.600 BSC		15.24 BSC		-
eA/2	0.300 BSC		7.62 BSC		-
L	0.125	0.200	3.18	5.08	-
Q	0.015	0.070	0.38	1.78	6
S1	0.005	-	0.13	-	7
α	90^o	105^o	90^o	105^o	-
aaa	-	0.015	-	0.38	-
bbb	-	0.030	-	0.76	-
ccc	-	0.010	-	0.25	-
M	-	0.0015	-	0.038	2, 3
N	40		40		8

Rev. 0 4/94

NOTES:

1. Index area: A notch or a pin one identification mark shall be located adjacent to pin one and shall be located within the shaded area shown. The manufacturer's identification shall not be used as a pin one identification mark.
2. The maximum limits of lead dimensions b and c or M shall be measured at the centroid of the finished lead surfaces, when solder dip or tin plate lead finish is applied.
3. Dimensions b1 and c1 apply to lead base metal only. Dimension M applies to lead plating and finish thickness.
4. Corner leads (1, N, N/2, and N/2+1) may be configured with a partial lead paddle. For this configuration dimension b3 replaces dimension b2.
5. This dimension allows for off-center lid, meniscus, and glass overrun.
6. Dimension Q shall be measured from the seating plane to the base plane.
7. Measure dimension S1 at all four corners.
8. N is the maximum number of terminal positions.
9. Dimensioning and tolerancing per ANSI Y14.5M - 1982.
10. Controlling dimension: INCH.

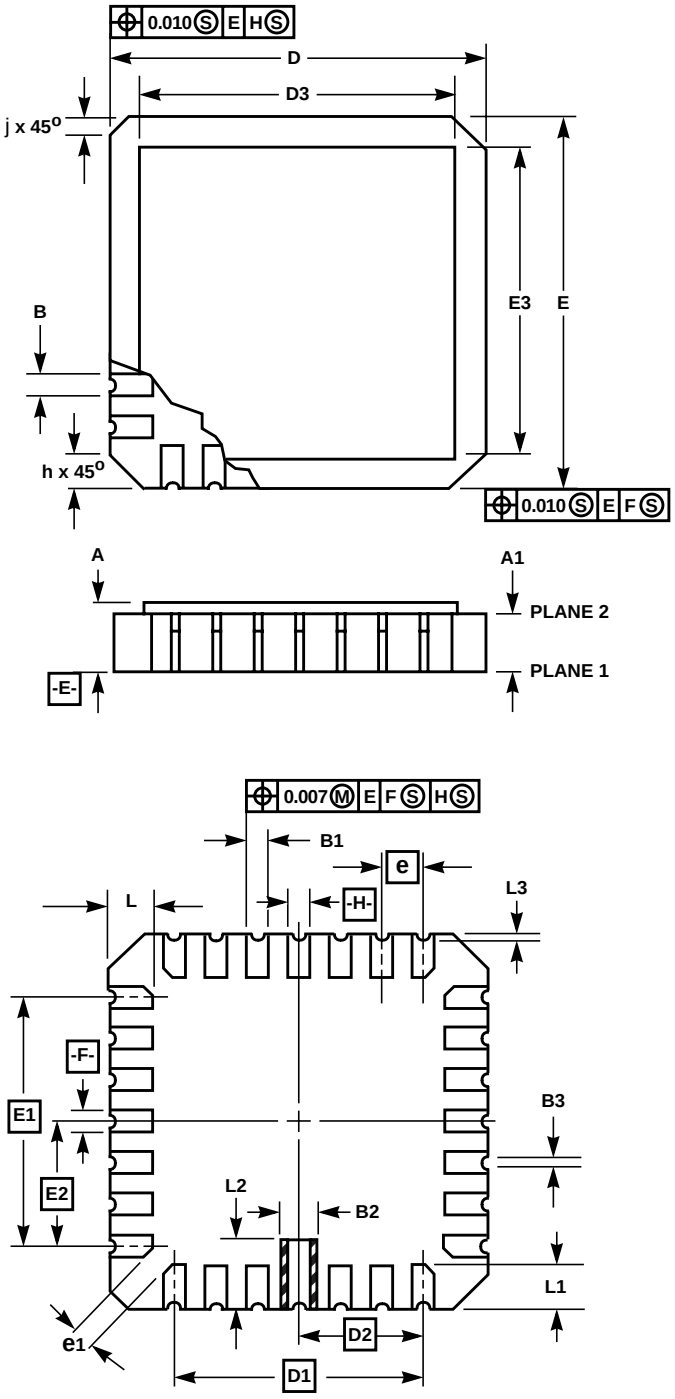
Sales Office Headquarters

NORTH AMERICA
Intersil Corporation
P. O. Box 883, Mail Stop 53-204
Melbourne, FL 32902
TEL: (407) 724-7000
FAX: (407) 724-7240

EUROPE
Intersil SA
Mercure Center
100, Rue de la Fusee
1130 Brussels, Belgium
TEL: (32) 2.724.2111
FAX: (32) 2.724.22.05

ASIA
Intersil (Taiwan) Ltd.
Taiwan Limited
7F-6, No. 101 Fu Hsing North Road
Taipei, Taiwan
Republic of China
TEL: (886) 2 2716 9310
FAX: (886) 2 2715 3029

Ceramic Leadless Chip Carrier Packages (CLCC)



J44.A **MIL-STD-1835 CQCC1-N44 (C-5)**
44 PAD CERAMIC LEADLESS CHIP CARRIER PACKAGE

SYMBOL	INCHES		MILLIMETERS		NOTES
	MIN	MAX	MIN	MAX	
A	0.064	0.120	1.63	3.05	6, 7
A1	0.054	0.088	1.37	2.24	-
B	0.033	0.039	0.84	0.99	4
B1	0.022	0.028	0.56	0.71	2, 4
B2	0.072 REF		1.83 REF		-
B3	0.006	0.022	0.15	0.56	-
D	0.640	0.662	16.26	16.81	-
D1	0.500 BSC		12.70 BSC		-
D2	0.250 BSC		6.35 BSC		-
D3	-	0.662	-	16.81	2
E	0.640	0.662	16.26	16.81	-
E1	0.500 BSC		12.70 BSC		-
E2	0.250 BSC		6.35 BSC		-
E3	-	0.662	-	16.81	2
e	0.050 BSC		1.27 BSC		-
e1	0.015	-	0.38	-	2
h	0.040 REF		1.02 REF		5
j	0.020 REF		0.51 REF		5
L	0.045	0.055	1.14	1.40	-
L1	0.045	0.055	1.14	1.40	-
L2	0.075	0.095	1.90	2.41	-
L3	0.003	0.015	0.08	0.38	-
ND	11		11		3
NE	11		11		3
N	44		44		3

Rev. 0 5/18/94

NOTES:

1. Metallized castellations shall be connected to plane 1 terminals and extend toward plane 2 across at least two layers of ceramic or completely across all of the ceramic layers to make electrical connection with the optional plane 2 terminals.
2. Unless otherwise specified, a minimum clearance of 0.015 inch (0.38mm) shall be maintained between all metallized features (e.g., lid, castellations, terminals, thermal pads, etc.)
3. Symbol “N” is the maximum number of terminals. Symbols “ND” and “NE” are the number of terminals along the sides of length “D” and “E”, respectively.
4. The required plane 1 terminals and optional plane 2 terminals (if used) shall be electrically connected.
5. The corner shape (square, notch, radius, etc.) may vary at the manufacturer’s option, from that shown on the drawing.
6. Chip carriers shall be constructed of a minimum of two ceramic layers.
7. Dimension “A” controls the overall package thickness. The maximum “A” dimension is package height before being solder dipped.
8. Dimensioning and tolerancing per ANSI Y14.5M-1982.
9. Controlling dimension: INCH.

Intel 8086 Family Architecture

General Purpose Registers

AH/AL AX (EAX) Accumulator
BH/BL BX (EBX) Base
CH/CL CX (ECX) Counter
DH/DL DX (EDX) Data

(Exx) indicates 386+ 32 bit register

Segment Registers

CS Code Segment
DS Data Segment
SS Stack Segment
ES Extra Segment
(FS) 386 and newer
(GS) 386 and newer

Pointer Registers

SI (ESI) Source Index
DI (EDI) Destination Index
IP Instruction Pointer

Stack Registers

SP (ESP) Stack Pointer
BP (EBP) Base Pointer

Status Registers

FLAGS Status Flags (see FLAGS)

Special Registers (386+ only)

CR0	Control Register 0	DR0	Debug Register 0
CR2	Control Register 2	DR1	Debug Register 1
CR3	Control Register 3	DR2	Debug Register 2
		DR3	Debug Register 3
TR4	Test Register 4	DR6	Debug Register 6
TR5	Test Register 5	DR7	Debug Register 7
TR6	Test Register 6		
TR7	Test Register 7		

Register	Default Segment	Valid Overrides
BP	SS	DS, ES, CS
SI or DI	DS	ES, SS, CS
DI strings	ES	None
SI strings	DS	ES, SS, CS

- see CPU DETECTING Instruction Timing

Instruction Clock Cycle Calculation

Some instructions require additional clock cycles due to a "Next Instruction Component" identified by a "+m" in the instruction clock cycle listings. This is due to the prefetch queue being purge on a control transfers. Below is the general rule for calculating "m":

88/86 not applicable
286 "m" is the number of bytes in the next instruction
386 "m" is the number of components in the next instruction
(the instruction coding (each byte), plus the data and the displacement are all considered components)

8088/8086 Effective Address (EA) Calculation

Description	Clock Cycles

Displacement	6
Base or Index (BX,BP,SI,DI)	5
Displacement+(Base or Index)	9
Base+Index (BP+DI,BX+SI)	7
Base+Index (BP+SI,BX+DI)	8
Base+Index+Displacement (BP+DI,BX+SI)	11
Base+Index+Displacement (BP+SI+disp,BX+DI+disp)	12

- add 4 cycles for word operands at odd addresses
- add 2 cycles for segment override
- 80188/80186 timings differ from those of the 8088/8086/80286

Task State Calculation

"TS" is defined as switching from VM/486 or 80286 TSS to one of the following:

	New Task				
Old Task	486 TSS (VM=0)	486 TSS (VM=1)	386 TSS (VM=0)	386 TSS (VM=1)	286 TSS
386 TSS (VM=0)			309	226	282
386 TSS (VM=1)			314	231	287
386 CPU/286 TSS			307	224	280
486 CPU/286 TSS	199	177			180

Miscellaneous

- all timings are for best case and do not take into account wait states, instruction alignment, the state of the prefetch queue, DMA refresh cycles, cache hits/misses or exception processing.
- to convert clocks to nanoseconds divide one microsecond by the processor speed in MegaHertz:

  (1000MHz/(n MHz)) = X nanoseconds

- see 8086 Architecture

FLAGS - Intel 8086 Family Flags Register

```
³11³10³F³E³D³C³B³A³9³8³7³6³5³4³3³2³1³0³
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  CF Carry Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  1
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  PF Parity Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  0
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  AF Auxiliary Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  0
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  ZF Zero Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  SF Sign Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  TF Trap Flag  (Single Step)
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  IF Interrupt Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  DF Direction Flag
 ³  ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ³ ÀÄÄÄ  OF Overflow flag
```

```
³  ³ ³ ³ ÀÄÁÄÄÄ  IOPL I/O Privilege Level  (286+ only)
³  ³ ³ ÀÄÄÄÄÄ  NT Nested Task Flag  (286+ only)
³  ³ ÀÄÄÄÄÄ  0
³  ÀÄÄÄÄÄ  RF Resume Flag (386+ only)
ÀÄÄÄÄÄÄ  VM  Virtual Mode Flag (386+ only)
```

- see PUSHF POPF STI CLI STD CLD

MSW - Machine Status Word (286+ only)

```
³31³30-5³4³3³2³1³0³  Machine Status Word
  ³   ³  ³ ³ ³ ³ ÀÄÄÄÄ Protection Enable (PE)
  ³   ³  ³ ³ ³ ÀÄÄÄÄÄ Math Present (MP)
  ³   ³  ³ ³ ÀÄÄÄÄÄÄ Emulation (EM)
  ³   ³  ³ ÀÄÄÄÄÄÄÄ Task Switched (TS)
  ³   ³  ÀÄÄÄÄÄÄÄÄ Extension Type (ET)
  ³   ÀÄÄÄÄÄÄÄÄÄÄ Reserved
  ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄ Paging (PG)
```

Bit 0	PE	Protection Enable, switches processor between protected and real mode
Bit 1	MP	Math Present, controls function of the WAIT instruction
Bit 2	EM	Emulation, indicates whether coprocessor functions are to be emulated
Bit 3	TS	Task Switched, set and interrogated by coprocessor on task switches and when interpretting coprocessor instructions
Bit 4	ET	Extension Type, indicates type of coprocessor in system
Bits 5-30		Reserved
bit 31	PG	Paging, indicates whether the processor uses page tables to translate linear addresses to physical addresses

- see SMSW LMSW

8086/80186/80286/80386/80486 Instruction Set

AAA - Ascii Adjust for Addition

Usage: AAA
Modifies flags: AF CF (OF,PF,SF,ZF undefined)

Changes contents of AL to valid unpacked decimal. The high order nibble is zeroed.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	8	3	4	3	1

AAD - Ascii Adjust for Division

Usage: AAD
Modifies flags: SF ZF PF (AF,CF,OF undefined)

Used before dividing unpacked decimal numbers. Multiplies AH by 10 and the adds result into AL. Sets AH to zero. This instruction is also known to have an undocumented behavior.

```
        AL := 10*AH+AL
        AH := 0

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        none              60    14    19    14            2
```

AAM - Ascii Adjust for Multiplication

```
        Usage:  AAM
        Modifies flags: PF SF ZF (AF,CF,OF undefined)

        AH := AL / 10
        AL := AL mod 10

        Used after multiplication of two unpacked decimal numbers, this
        instruction adjusts an unpacked decimal number.  The high order
        nibble of each byte must be zeroed before using this instruction.
        This instruction is also known to have an undocumented behavior.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        none              83    16    17    15            2
```

AAS - Ascii Adjust for Subtraction

```
        Usage:  AAS
        Modifies flags: AF CF (OF,PF,SF,ZF undefined)

        Corrects result of a previous unpacked decimal subtraction in AL.
        High order nibble is zeroed.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        none               8     3     4     3            1
```

ADC - Add With Carry

```
        Usage:  ADC     dest,src
        Modifies flags: AF CF OF SF PF ZF

        Sums two binary operands placing the result in the destination.
        If CF is set, a 1 is added to the destination.

                               Clocks                   Size
        Operands         808x  286   386   486          Bytes

        reg,reg            3     2     2     1            2
        mem,reg          16+EA   7     7     3           2-4  (W88=24+EA)
        reg,mem           9+EA   7     6     2           2-4  (W88=13+EA)
        reg,immed          4     3     2     1           3-4
        mem,immed        17+EA   7     7     3           3-6  (W88=23+EA)
        accum,immed        4     3     2     1           2-3
```

ADD - Arithmetic Addition

Usage: ADD dest,src
Modifies flags: AF CF OF PF SF ZF

Adds "src" to "dest" and replacing the original contents of "dest". Both operands are binary.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	16+EA	7	7	3	2-4	(W88=24+EA)
reg,mem	9+EA	7	6	2	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem,immed	17+EA	7	7	3	3-6	(W88=23+EA)
accum,immed	4	3	2	1	2-3	

AND - Logical And

Usage: AND dest,src
Modifies flags: CF OF PF SF ZF (AF undefined)

Performs a logical AND of the two operands replacing the destination with the result.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	16+EA	7	7	3	2-4	(W88=24+EA)
reg,mem	9+EA	7	6	1	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem,immed	17+EA	7	7	3	3-6	(W88=23+EA)
accum,immed	4	3	2	1	2-3	

ARPL - Adjusted Requested Privilege Level of Selector (286+ PM)

Usage: ARPL dest,src
(286+ protected mode)
Modifies flags: ZF

Compares the RPL bits of "dest" against "src". If the RPL bits of "dest" are less than "src", the destination RPL bits are set equal to the source RPL bits and the Zero Flag is set. Otherwise the Zero Flag is cleared.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,reg	-	10	20	9	2
mem,reg	-	11	21	9	4

BOUND - Array Index Bound Check (80188+)

Usage: BOUND src,limit
Modifies flags: None

Array index in source register is checked against upper and lower bounds in memory source. The first word located at "limit" is

the lower boundary and the word at "limit+2" is the upper array bound.

Interrupt 5 occurs if the source value is less than or higher than the source.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,mem32	-	nj=13	nj=10	7	2
reg32,mem64	-	nj=13	nj=10	7	2

- nj = no jump taken

BSF - Bit Scan Forward (386+)

Usage: BSF dest,src
Modifies flags: ZF

Scans source operand for first bit set. Sets ZF if a bit is found set and loads the destination with an index to first set bit. Clears ZF is no bits are found set. BSF scans forward across bit pattern (0-n) while BSR scans in reverse (n-0).

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,reg	-	-	10+3n	6-42	3
reg,mem	-	-	10+3n	7-43	3-7
reg32,reg32	-	-	10+3n	6-42	3-7
reg32,mem32	-	-	10+3n	7-43	3-7

BSR - Bit Scan Reverse (386+)

Usage: BSR dest,src
Modifies flags: ZF

Scans source operand for first bit set. Sets ZF if a bit is found set and loads the destination with an index to first set bit. Clears ZF is no bits are found set. BSF scans forward across bit pattern (0-n) while BSR scans in reverse (n-0).

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,reg	-	-	10+3n	6-103	3
reg,mem	-	-	10+3n	7-104	3-7
reg32,reg32	-	-	10+3n	6-103	3-7
reg32,mem32	-	-	10+3n	7-104	3-7

BSWAP - Byte Swap (486+)

Usage: BSWAP reg32
Modifies flags: none

Changes the byte order of a 32 bit register from big endian to little endian or vice versa. Result left in destination register is undefined if the operand is a 16 bit register.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg32	-	-	-	1	2

BT - Bit Test (386+)

Usage: BT dest,src
Modifies flags: CF

The destination bit indexed by the source value is copied into the Carry Flag.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,immed8	-	-	3	3	4-8
mem16,immed8	-	-	6	6	4-8
reg16,reg16	-	-	3	3	3-7
mem16,reg16	-	-	12	12	3-7

BTC - Bit Test with Compliment (386+)

Usage: BTC dest,src
Modifies flags: CF

The destination bit indexed by the source value is copied into the Carry Flag after being complimented (inverted).

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,immed8	-	-	6	6	4-8
mem16,immed8	-	-	8	8	4-8
reg16,reg16	-	-	6	6	3-7
mem16,reg16	-	-	13	13	3-7

BTR - Bit Test with Reset (386+)

Usage: BTR dest,src
Modifies flags: CF

The destination bit indexed by the source value is copied into the Carry Flag and then cleared in the destination.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,immed8	-	-	6	6	4-8
mem16,immed8	-	-	8	8	4-8
reg16,reg16	-	-	6	6	3-7
mem16,reg16	-	-	13	13	3-7

BTS - Bit Test and Set (386+)

Usage: BTS dest,src
Modifies flags: CF

The destination bit indexed by the source value is copied into the Carry Flag and then set in the destination.

Operands	Clocks 808x	286	386	486	Size Bytes
reg16,immed8	-	-	6	6	4-8
mem16,immed8	-	-	8	8	4-8
reg16,reg16	-	-	6	6	3-7
mem16,reg16	-	-	13	13	3-7

CALL - Procedure Call

Usage: CALL destination
Modifies flags: None

Pushes Instruction Pointer (and Code Segment for far calls) onto stack and loads Instruction Pointer with the address of proc-name. Code continues with execution at CS:IP.

Operands	Clocks 808x	286	386	486
rel16 (near, IP relative)	19	7	7+m	3
rel32 (near, IP relative)	-	-	7+m	3
reg16 (near, register indirect)	16	7	7+m	5
reg32 (near, register indirect)	-	-	7+m	5
mem16 (near, memory indirect)	-	21+EA	11	10+m
5				
mem32 (near, memory indirect)	-	-	10+m	5
ptr16:16 (far, full ptr supplied)	28	13	17+m	18
ptr16:32 (far, full ptr supplied)	-	-	17+m	18
ptr16:16 (far, ptr supplied, prot. mode)	-	26	34+m	20
ptr16:32 (far, ptr supplied, prot. mode)	-	-	34+m	20
m16:16 (far, indirect)	37+EA	16	22+m	17
m16:32 (far, indirect)	-	-	22+m	17
m16:16 (far, indirect, prot. mode)	-	29	38+m	20
m16:32 (far, indirect, prot. mode)	-	-	38+m	20
ptr16:16 (task, via TSS or task gate)	-	177	TS	37+TS
m16:16 (task, via TSS or task gate)	-	180/185	5+TS	37+TS
m16:32 (task)	-	-	TS	37+TS
m16:32 (task)	-	-	5+TS	37+TS
ptr16:16 (gate, same privilege)	-	41	52+m	35
ptr16:32 (gate, same privilege)	-	-	52+m	35
m16:16 (gate, same privilege)	-	44	56+m	35
m16:32 (gate, same privilege)	-	-	56+m	35
ptr16:16 (gate, more priv, no parm)	-	82	86+m	69
ptr16:32 (gate, more priv, no parm)	-	-	86+m	69
m16:16 (gate, more priv, no parm)	-	83	90+m	69
m16:32 (gate, more priv, no parm)	-	-	90+m	69
ptr16:16 (gate, more priv, x parms)	-	86+4x	94+4x+m	77+4x
ptr16:32 (gate, more priv, x parms)	-	-	94+4x+m	77+4x
m16:16 (gate, more priv, x parms)	-	90+4x	98+4x+m	77+4x
m16:32 (gate, more priv, x parms)	-	-	98+4x+m	77+4x

CBW - Convert Byte to Word

Usage: CBW
Modifies flags: None

Converts byte in AL to word Value in AX by extending sign of AL throughout register AH.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	3	3	1

CDQ - Convert Double to Quad (386+)

Usage: CDQ
Modifies flags: None

Converts signed DWORD in EAX to a signed quad word in EDX:EAX by extending the high order bit of EAX throughout EDX

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	-	2	3	1

CLC - Clear Carry

Usage: CLC
Modifies flags: CF

Clears the Carry Flag.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2	2	1

CLD - Clear Direction Flag

Usage: CLD
Modifies flags: DF

Clears the Direction Flag causing string instructions to increment the SI and DI index registers.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2	2	1

CLI - Clear Interrupt Flag (disable)

Usage: CLI
Modifies flags: IF

Disables the maskable hardware interrupts by clearing the Interrupt flag. NMI's and software interrupts are not inhibited.

	Clocks				Size
Operands	808x	286	386	486	Bytes

none	2	2	3	5	1	

CLTS - Clear Task Switched Flag (286+ privileged)

Usage: CLTS
Modifies flags: None

Clears the Task Switched Flag in the Machine Status Register. This is a privileged operation and is generally used only by operating system code.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
none	-	2	5	7	2	

CMC - Complement Carry Flag

Usage: CMC
Modifies flags: CF

Toggles (inverts) the Carry Flag

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
none	2	2	2	2	1	

CMP - Compare

Usage: CMP dest,src
Modifies flags: AF CF OF PF SF ZF

Subtracts source from destination and updates the flags but does not save result. Flags can subsequently be checked for conditions.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	9+EA	7	5	2	2-4	(W88=13+EA)
reg,mem	9+EA	6	6	2	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem,immed	10+EA	6	5	2	3-6	(W88=14+EA)
accum,immed	4	3	2	1	2-3	

CMPS - Compare String (Byte, Word or Doubleword)

Usage: CMPS dest,src
CMPSB
CMPSW
CMPSD (386+)
Modifies flags: AF CF OF PF SF ZF

Subtracts destination value from source without saving results. Updates flags based on the subtraction and the index registers (E)SI and (E)DI are incremented or decremented depending on the state of the Direction Flag. CMPSB inc/decrements the index registers by 1, CMPSW inc/decrements by 2, while CMPSD increments or decrements by 4. The REP prefixes can be used to process

entire data items.

	Clocks				Size
Operands	808x	286	386	486	Bytes
dest,src	22	8	10	8	1 (W88=30)

CMPXCHG - Compare and Exchange

Usage: CMPXCHG dest,src (486+)
Modifies flags: AF CF OF PF SF ZF

Compares the accumulator (8-32 bits) with "dest". If equal the "dest" is loaded with "src", otherwise the accumulator is loaded with "dest".

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,reg	-	-	-	6	2
mem,reg	-	-	-	7	2

- add 3 clocks if the "mem,reg" comparison fails

CWD - Convert Word to Doubleword

Usage: CWD
Modifies flags: None

Extends sign of word in register AX throughout register DX forming a doubleword quantity in DX:AX.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	5	2	2	3	1

CWDE - Convert Word to Extended Doubleword (386+)

Usage: CWDE
Modifies flags: None

Converts a signed word in AX to a signed doubleword in EAX by extending the sign bit of AX throughout EAX.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	-	3	3	1

DAA - Decimal Adjust for Addition

Usage: DAA
Modifies flags: AF CF PF SF ZF (OF undefined)

Corrects result (in AL) of a previous BCD addition operation. Contents of AL are changed to a pair of packed decimal digits.

	Clocks				Size
Operands	808x	286	386	486	Bytes

```
        none              4     3     4     2            1


DAS - Decimal Adjust for Subtraction

        Usage:  DAS
        Modifies flags: AF CF PF SF ZF (OF undefined)

        Corrects result (in AL) of a previous BCD subtraction operation.
        Contents of AL are changed to a pair of packed decimal digits.

                                 Clocks                 Size
        Operands         808x  286   386   486          Bytes

        none              4     3     4     2            1


DEC - Decrement

        Usage:  DEC     dest
        Modifies flags: AF OF PF SF ZF

        Unsigned binary subtraction of one from the destination.

                                 Clocks                 Size
        Operands         808x  286   386   486          Bytes

        reg8              3     2     2     1            2
        mem             15+EA   7     6     3           2-4
        reg16/32          3     2     2     1            1


DIV - Divide

        Usage:  DIV     src
        Modifies flags: (AF,CF,OF,PF,SF,ZF undefined)

        Unsigned binary division of accumulator by source.  If the source
        divisor is a byte value then AX is divided by "src" and the
quotient
        is placed in AL and the remainder in AH.  If source operand is a
word
        value, then DX:AX is divided by "src" and the quotient is stored in
AX
        and the remainder in DX.

                                 Clocks                 Size
        Operands         808x  286   386   486          Bytes

        reg8             80-90  14    14    16           2
        reg16           144-162 22    22    24           2
        reg32              -     -     38    40           2
        mem8        (86-96)+EA  17    17    16          2-4
        mem16     (150-168)+EA  25    25    24          2-4  (W88=158-
176+EA)
        mem32              -     -     41    40          2-4

ENTER - Make Stack Frame  (80188+)

        Usage:  ENTER   locals,level
        Modifies flags: None
```

Modifies stack for entry to procedure for high level language. Operand "locals" specifies the amount of storage to be allocated on the stack. "Level" specifies the nesting level of the routine. Paired with the LEAVE instruction, this is an efficient method of entry and exit to procedures.

	Clocks				Size
Operands	808x	286	386	486	Bytes
immed16,0	-	11	10	14	4
immed16,1	-	15	12	17	4
immed16,immed8	-	12+4(n-1)	15+4(n-1)	17+3n	4

ESC - Escape

Usage: ESC immed,src
Modifies flags: None

Provides access to the data bus for other resident processors. The CPU treats it as a NOP but places memory operand on bus.

	Clocks				Size
Operands	808x	286	386	486	Bytes
immed,reg	2	9-20	?		2
immed,mem	2	9-20	?		2-4

HLT - Halt CPU

Usage: HLT
Modifies flags: None

Halts CPU until RESET line is activated, NMI or maskable interrupt received. The CPU becomes dormant but retains the current CS:IP for later restart.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	5	4	1

IDIV - Signed Integer Division

Usage: IDIV src
Modifies flags: (AF,CF,OF,PF,SF,ZF undefined)

Signed binary division of accumulator by source. If source is a byte value, AX is divided by "src" and the quotient is stored in AL and the remainder in AH. If source is a word value, DX:AX is divided by "src", and the quotient is stored in AL and the remainder in DX.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg8	101-112	17	19	19	2	
reg16	165-184	25	27	27	2	
reg32	-	-	43	43	2	
mem8	(107-118)+EA	20	22	20	2-4	
mem16	(171-190)+EA	38	30	28	2-4	(W88=175-194)
mem32	-	-	46	44	2-4	

IMUL - Signed Multiply

```
        Usage:  IMUL    src
                IMUL    src,immed        (286+)
                IMUL    dest,src,immed8  (286+)
                IMUL    dest,src         (386+)
        Modifies flags: CF OF (AF,PF,SF,ZF undefined)
```

Signed multiplication of accumulator by "src" with result placed in the accumulator. If the source operand is a byte value, it is multiplied by AL and the result stored in AX. If the source operand is a word value it is multiplied by AX and the result is stored in DX:AX. Other variations of this instruction allow specification of source and destination registers as well as a third immediate factor.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	80-98	13	9-14	13-18	2
reg16	128-154	21	9-22	13-26	2
reg32	-	-	9-38	12-42	2
mem8	86-104	16	12-17	13-18	2-4
mem16	134-160	24	12-25	13-26	2-4
mem32	-	-	12-41	13-42	2-4
reg16,reg16	-	-	9-22	13-26	3-5
reg32,reg32	-	-	9-38	13-42	3-5
reg16,mem16	-	-	12-25	13-26	3-5
reg32,mem32	-	-	12-41	13-42	3-5
reg16,immed	-	21	9-22	13-26	3
reg32,immed	-	21	9-38	13-42	3-6
reg16,reg16,immed	-	2	9-22	13-26	3-6
reg32,reg32,immed	-	21	9-38	13-42	3-6
reg16,mem16,immed	-	24	12-25	13-26	3-6
reg32,mem32,immed	-	24	12-41	13-42	3-6

IN - Input Byte or Word From Port

```
        Usage:  IN      accum,port
        Modifies flags: None
```

A byte, word or dword is read from "port" and placed in AL, AX or EAX respectively. If the port number is in the range of 0-255 it can be specified as an immediate, otherwise the port number must be specified in DX. Valid port ranges on the PC are 0-1024, though values through 65535 may be specified and recognized by third party vendors and PS/2's.

	Clocks				Size
Operands	808x	286	386	486	Bytes
accum,immed8	10/14	5	12	14	2
accum,immed8 (PM)			6/26	8/28/27	2
accum,DX	8/12	5	13	14	1
accum,DX (PM)			7/27	8/28/27	1

- 386+ protected mode timings depend on privilege levels.

  first number is the timing if: CPL ó IOPL
  second number is the timing if: CPL > IOPL or in VM 86 mode (386)
  CPL ò IOPL (486)

```
          third number is the timing when:    virtual mode on 486 processor
        - 486 virtual mode always requires 27 cycles

INC - Increment

        Usage:  INC     dest
        Modifies flags: AF OF PF SF ZF

        Adds one to destination unsigned binary operand.

                                  Clocks                  Size
        Operands         808x  286   386   486           Bytes

        reg8              3     2     2     1             2
        reg16             3     2     2     1             1
        reg32             3     2     2     1             1
        mem             15+EA   7     6     3             2-4  (W88=23+EA)


INS - Input String from Port  (80188+)

        Usage:  INS     dest,port
                INSB
                INSW
                INSD    (386+)
        Modifies flags: None

        Loads data from port to the destination ES:(E)DI  (even if a
        destination operand is supplied).  (E)DI is adjusted by the size
        of the operand and increased if the Direction Flag is cleared and
        decreased if the Direction Flag is set.  For INSB, INSW, INSD no
        operands are allowed and the size is determined by the mnemonic.

                                  Clocks                  Size
        Operands         808x  286   386    486          Bytes

        dest,port         -     5     15     17            1
        dest,port (PM)    -     5    9/29 10/32/30         1
        none              -     5     15     17            1
        none (PM)         -     5    9/29 10/32/30         1

        - 386+ protected mode timings depend on privilege levels.

          first number is the timing if:    CPL ó IOPL
          second number is the timing if:   CPL > IOPL
          third number is the timing if:    virtual mode on 486 processor


INT - Interrupt

        Usage:  INT     num
        Modifies flags: TF IF

        Initiates a software interrupt by pushing the flags, clearing the
        Trap and Interrupt Flags, pushing CS followed by IP and loading
        CS:IP with the value found in the interrupt vector table.
Execution
        then begins at the location addressed by the new CS:IP

                                              Clocks           Size
            Operands                    808x  286   386   486  Bytes

 3 (constant)                          52/72  23+m   33    26    2
```

3 (prot. mode, same priv.)	-	40+m	59	44	2
3 (prot. mode, more priv.)	-	78+m	99	71	2
3 (from VM86 to PL 0)	-	-	119	82	2
3 (prot. mode via task gate)	-	167+m	TS	37+TS	2
immed8	51/71	23+m	37	30	1
immed8 (prot. mode, same priv.)	-	40+m	59	44	1
immed8 (prot. mode, more priv.)	-	78+m	99	71	1
immed8 (from VM86 to PL 0)	-	-	119	86	1
immed8 (prot. mode, via task gate)	-	167+m	TS	37+TS	1

INTO - Interrupt on Overflow

Usage: INTO
Modifies flags: IF TF

If the Overflow Flag is set this instruction generates an INT 4 which causes the code addressed by 0000:0010 to be executed.

		Clocks			Size
Operands	808x	286	386	486	Bytes
none: jump	53/73	24+m	35	28	1
no jump	4	3	3	3	
(prot. mode, same priv.)	-	-	59	46	1
(prot. mode, more priv.)	-	-	99	73	1
(from VM86 to PL 0)	-	-	119	84	1
(prot. mode, via task gate)	-	-	TS	39+TS	1

INVD - Invalidate Cache (486+)

Usage: INVD
Modifies flags: none

Flushes CPU internal cache. Issues special function bus cycle which indicates to flush external caches. Data in write-back external caches is lost.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	-	-	4	2

INVLPG - Invalidate Translation Look-Aside Buffer Entry (486+)

Usage: INVLPG
Modifies flags: none

Invalidates a single page table entry in the Translation Look-Aside Buffer. Intel warns that this instruction may be implemented differently on future processors.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	-	-	12	2

- timing is for TLB entry hit only.

IRET/IRETD - Interrupt Return

```
        Usage:  IRET
                IRETD  (386+)
        Modifies flags: AF CF DF IF PF SF TF ZF

        Returns control to point of interruption by popping IP, CS
        and then the Flags from the stack and continues execution at
        this location.  CPU exception interrupts will return to the
        instruction that cause the exception because the CS:IP placed
        on the stack during the interrupt is the address of the offending
        instruction.
```

Operands	808x	286	386	486	Size Bytes
iret	32/44	17+m	22	15	1
iret (prot. mode)	-	31+m	38	15	1
iret (to less privilege)	-	55+m	82	36	1
iret (different task, NT=1)	-	169+m	TS	TS+32	1
iretd	-	-	22/38	15	1
iretd (to less privilege)	-	-	82	36	1
iretd (to VM86 mode)	-	-	60	15	1
iretd (different task, NT=1)	-	-	TS	TS+32	1

- 386 timings are listed as real-mode/protected-mode

Jxx - Jump Instructions Table

Mnemonic	Meaning	Jump Condition
JA	Jump if Above	CF=0 and ZF=0
JAE	Jump if Above or Equal	CF=0
JB	Jump if Below	CF=1
JBE	Jump if Below or Equal	CF=1 or ZF=1
JC	Jump if Carry	CF=1
JCXZ	Jump if CX Zero	CX=0
JE	Jump if Equal	ZF=1
JG	Jump if Greater (signed)	ZF=0 and SF=OF
JGE	Jump if Greater or Equal (signed)	SF=OF
JL	Jump if Less (signed)	SF != OF
JLE	Jump if Less or Equal (signed)	ZF=1 or SF != OF
JMP	Unconditional Jump	unconditional
JNA	Jump if Not Above	CF=1 or ZF=1
JNAE	Jump if Not Above or Equal	CF=1
JNB	Jump if Not Below	CF=0
JNBE	Jump if Not Below or Equal	CF=0 and ZF=0
JNC	Jump if Not Carry	CF=0
JNE	Jump if Not Equal	ZF=0
JNG	Jump if Not Greater (signed)	ZF=1 or SF != OF
JNGE	Jump if Not Greater or Equal (signed)	SF != OF
JNL	Jump if Not Less (signed)	SF=OF
JNLE	Jump if Not Less or Equal (signed)	ZF=0 and SF=OF
JNO	Jump if Not Overflow (signed)	OF=0
JNP	Jump if No Parity	PF=0
JNS	Jump if Not Signed (signed)	SF=0
JNZ	Jump if Not Zero	ZF=0
JO	Jump if Overflow (signed)	OF=1
JP	Jump if Parity	PF=1
JPE	Jump if Parity Even	PF=1
JPO	Jump if Parity Odd	PF=0
JS	Jump if Signed (signed)	SF=1
JZ	Jump if Zero	ZF=1

	Clocks	Size

Operands	808x	286	386	486	Bytes
Jx: jump	16	7+m	7+m	3	2
no jump	4	3	3	1	
Jx near-label	-	-	7+m	3	4
no jump	-	-	3	1	

- It's a good programming practice to organize code so the expected case is executed without a jump since the actual jump takes longer to execute than falling through the test.
- see JCXZ and JMP for their respective timings

JCXZ/JECXZ - Jump if Register (E)CX is Zero

Usage: JCXZ label
JECXZ label (386+)
Modifies flags: None

Causes execution to branch to "label" if register CX is zero. Uses unsigned comparision.

		Clocks			Size
Operands	808x	286	386	486	Bytes
label: jump	18	8+m	9+m	8	2
no jump	6	4	5	5	

JMP - Unconditional Jump

Usage: JMP target
Modifies flags: None

Unconditionally transfers control to "label". Jumps by default are within -32768 to 32767 bytes from the instruction following the jump. NEAR and SHORT jumps cause the IP to be updated while FAR jumps cause CS and IP to be updated.

	Clocks			
Operands	808x	286	386	486
rel8 (relative)	15	7+m	7+m	3
rel16 (relative)	15	7+m	7+m	3
rel32 (relative)	-	-	7+m	3
reg16 (near, register indirect)	11	7+m	7+m	5
reg32 (near, register indirect)	-	-	7+m	5
mem16 (near, mem indirect)	18+EA	11+m	10+m	5
mem32 (near, mem indirect)	24+EA	15+m	10+m	5
ptr16:16 (far, dword immed)	-	-	12+m	17
ptr16:16 (far, PM dword immed)	-	-	27+m	19
ptr16:16 (call gate, same priv.)	-	38+m	45+m	32
ptr16:16 (via TSS)	-	175+m	TS	42+TS
ptr16:16 (via task gate)	-	180+m	TS	43+TS
mem16:16 (far, indirect)	-	-	43+m	13
mem16:16 (far, PM indirect)	-	-	31+m	18
mem16:16 (call gate, same priv.)	-	41+m	49+m	31
mem16:16 (via TSS)	-	178+m	5+TS	41+TS
mem16:16 (via task gate)	-	183+m	5+TS	42+TS
ptr16:32 (far, 6 byte immed)	-	-	12+m	13
ptr16:32 (far, PM 6 byte immed)	-	-	27+m	18
ptr16:32 (call gate, same priv.)	-	-	45+m	31
ptr16:32 (via TSS)	-	-	TS	42+TS

ptr16:32 (via task state)				-	-	TS 43+TS
m16:32 (far, address at dword)				-	-	43+m 13
m16:32 (far, address at dword)				-	-	31+m 18
m16:32 (call gate, same priv.)				-	-	49+m 31
m16:32 (via TSS)				-	-	5+TS 41+TS
m16:32 (via task state)				-	-	5+TS 42+TS

LAHF - Load Register AH From Flags

Usage: LAHF
Modifies flags: None

Copies bits 0-7 of the flags register into AH. This includes flags AF, CF, PF, SF and ZF other bits are undefined.

AH := SF ZF xx AF xx PF xx CF

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	4	2	2	3	1

LAR - Load Access Rights (286+ protected)

Usage: LAR dest,src
Modifies flags: ZF

The high byte of the of the destination register is overwritten by the value of the access rights byte and the low order byte is zeroed depending on the selection in the source operand. The Zero Flag is set if the load operation is successful.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,reg16	-	14	15	11	3
reg32,reg32	-	-	15	11	3
reg16,mem16	-	16	16	11	3-7
reg32,mem32	-	-	16	11	3-7

LDS - Load Pointer Using DS

Usage: LDS dest,src
Modifies flags: None

Loads 32-bit pointer from memory source to destination register and DS. The offset is placed in the destination register and the segment is placed in DS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,mem32	16+EA	7	7	6	2-4
reg,mem (PM)	-	-	22	12	5-7

LEA - Load Effective Address

Usage: LEA dest,src
Modifies flags: None

Transfers offset address of "src" to the destination register.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,mem	2+EA	3	2	1	2-4

- the MOV instruction can often save clock cycles when used in place of LEA on 8088 processors

LEAVE - Restore Stack for Procedure Exit (80188+)

Usage: LEAVE
Modifies flags: None

Releases the local variables created by the previous ENTER instruction by restoring SP and BP to their condition before the procedure stack frame was initialized.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	5	4	5	1

LES - Load Pointer Using ES

Usage: LES dest,src
Modifies flags: None

Loads 32-bit pointer from memory source to destination register and ES. The offset is placed in the destination register and the segment is placed in ES. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,mem	16+EA	7	7	6	2-4	(W88=24+EA)
reg,mem (PM)	-	-	22	12	5-7	

LFS - Load Pointer Using FS (386+)

Usage: LFS dest,src
Modifies flags: None

Loads 32-bit pointer from memory source to destination register and FS. The offset is placed in the destination register and the segment is placed in FS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,mem	-	-	7	6	5-7
reg,mem (PM)	-	-	22	12	5-7

LGDT - Load Global Descriptor Table (286+ privileged)

Usage: LGDT src
Modifies flags: None

Loads a value from an operand into the Global Descriptor Table (GDT) register.

	Clocks				Size
Operands	808x	286	386	486	Bytes
mem64	-	11	11	11	5

LIDT - Load Interrupt Descriptor Table (286+ privileged)

Usage: LIDT src
Modifies flags: None

Loads a value from an operand into the Interrupt Descriptor Table (IDT) register.

	Clocks				Size
Operands	808x	286	386	486	Bytes
mem64	-	12	11	11	5

LGS - Load Pointer Using GS (386+)

Usage: LGS dest,src
Modifies flags: None

Loads 32-bit pointer from memory source to destination register and GS. The offset is placed in the destination register and the segment is placed in GS. To use this instruction the word at the lower memory address must contain the offset and the word at the higher address must contain the segment. This simplifies the loading of far pointers from the stack and the interrupt vector table.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,mem	-	-	7	6	5-7
reg,mem (PM)	-	-	22	12	5-7

LLDT - Load Local Descriptor Table (286+ privileged)

Usage: LLDT src
Modifies flags: None

Loads a value from an operand into the Local Descriptor Table Register (LDTR).

	Clocks	Size

Operands	808x	286	386	486	Bytes
reg16	-	17	20	11	3
mem16	-	19	24	11	5

LMSW - Load Machine Status Word (286+ privileged)

Usage: LMSW src
Modifies flags: None

Loads the Machine Status Word (MSW) from data found at "src"

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16	-	3	10	13	3
mem16	-	6	13	13	5

LOCK - Lock Bus

Usage: LOCK
LOCK: (386+ prefix)
Modifies flags: None

This instruction is a prefix that causes the CPU assert bus lock signal during the execution of the next instruction. Used to avoid two processors from updating the same data location. The 286 always asserts lock during an XCHG with memory operands. This should only be used to lock the bus prior to XCHG, MOV, IN and OUT instructions.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	0	0	1	1

LODS - Load String (Byte, Word or Double)

Usage: LODS src
LODSB
LODSW
LODSD (386+)
Modifies flags: None

Transfers string element addressed by DS:SI (even if an operand is supplied) to the accumulator. SI is incremented based on the size of the operand or based on the instruction used. If the Direction Flag is set SI is decremented, if the Direction Flag is clear SI is incremented. Use with REP prefixes.

	Clocks				Size
Operands	808x	286	386	486	Bytes
src	12/16	5	5	5	1

LOOP - Decrement CX and Loop if CX Not Zero

Usage: LOOP label
Modifies flags: None

Decrements CX by 1 and transfers control to "label" if CX is not Zero. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction

		Clocks				Size
Operands		808x	286	386	486	Bytes
label:	jump	18	8+m	11+m	6	2
	no jump	5	4	?	2	

LOOPE/LOOPZ - Loop While Equal / Loop While Zero

Usage: LOOPE label
LOOPZ label
Modifies flags: None

Decrements CX by 1 (without modifying the flags) and transfers control to "label" if CX != 0 and the Zero Flag is set. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction.

		Clocks				Size
Operands		808x	286	386	486	Bytes
label:	jump	18	8+m	11+m	9	2
	no jump	5	4	?	6	

LOOPNZ/LOOPNE - Loop While Not Zero / Loop While Not Equal

Usage: LOOPNZ label
LOOPNE label
Modifies flags: None

Decrements CX by 1 (without modifying the flags) and transfers control to "label" if CX != 0 and the Zero Flag is clear. The "label" operand must be within -128 or 127 bytes of the instruction following the loop instruction.

		Clocks				Size
Operands		808x	286	386	486	Bytes
label:	jump	19	8+m	11+m	9	2
	no jump	5	4	?	6	

LSL - Load Segment Limit (286+ protected)

Usage: LSL dest,src
Modifies flags: ZF

Loads the segment limit of a selector into the destination register if the selector is valid and visible at the current privilege level.
If loading is successful the Zero Flag is set, otherwise it is cleared.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16,reg16	-	14	20/25	10	3
reg32,reg32	-	-	20/25	10	3

```
        reg16,mem16       -     16  21/26   10           5
        reg32,mem32       -     -   21/26   10           5

        - 386 times are listed "byte granular" / "page granular"

LSS - Load Pointer Using SS (386+)

        Usage:  LSS     dest,src
        Modifies flags: None

        Loads 32-bit pointer from memory source to destination register
        and SS.  The offset is placed in the destination register and the
        segment is placed in SS.  To use this instruction the word at the
        lower memory address must contain the offset and the word at the
        higher address must contain the segment.  This simplifies the
loading
        of far pointers from the stack and the interrupt vector table.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg,mem           -     -     7     6           5-7
        reg,mem (PM)      -     -     22    12          5-7


LTR - Load Task Register (286+ privileged)

        Usage:  LTR     src
        Modifies flags: None

        Loads the current task register with the value specified in "src".

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg16             -     17    23    20           3
        mem16             -     19    27    20           5


MOV - Move Byte or Word

        Usage:  MOV     dest,src
        Modifies flags: None

        Copies byte or word from the source operand to the destination
        operand.  If the destination is SS interrupts are disabled except
        on early buggy 808x CPUs.  Some CPUs disable interrupts if the
        destination is any of the segment registers

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg,reg           2     2     2     1           2
        mem,reg          9+EA   3     2     1           2-4  (W88=13+EA)
        reg,mem          8+EA   5     4     1           2-4  (W88=12+EA)
        mem,immed       10+EA   3     2     1           3-6  (W88=14+EA)
        reg,immed         4     2     2     1           2-3
        mem,accum         10    3     2     1           3    (W88=14)
        accum,mem         10    5     4     1           3    (W88=14)
        segreg,reg16      2     2     2     3           2
        segreg,mem16     8+EA   5     5     9           2-4  (W88=12+EA)
        reg16,segreg      2     2     2     3           2
        mem16,segreg     9+EA   3     2     3           2-4  (W88=13+EA)
```

reg32,CR0/CR2/CR3	-	-	6	4	
CR0,reg32	-	-	10	16	
CR2,reg32	-	-	4	4	3
CR3,reg32	-	-	5	4	3
reg32,DR0/DR1/DR2/DR3	-		22	10	3
reg32,DR6/DR7	-	-	22	10	3
DR0/DR1/DR2/DR3,reg32	-	-	22	11	3
DR6/DR7,reg32	-	-	16	11	3
reg32,TR6/TR7	-	-	12	4	3
TR6/TR7,reg32	-	-	12	4	3
reg32,TR3				3	
TR3,reg32				6	

- when the 386 special registers are used all operands are 32 bits

MOVS - Move String (Byte or Word)

Usage: MOVS dest,src
MOVSB
MOVSW
MOVSD (386+)
Modifies flags: None

Copies data from addressed by DS:SI (even if operands are given) to the location ES:DI destination and updates SI and DI based on the size of the operand or instruction used. SI and DI are incremented when the Direction Flag is cleared and decremented when the Direction
Flag is Set. Use with REP prefixes.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
dest,src	18	5	7	7	1	(W88=26)

MOVSX - Move with Sign Extend (386+)

Usage: MOVSX dest,src
Modifies flags: None

Copies the value of the source operand to the destination register with the sign extended.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,reg	-	-	3	3	3
reg,mem	-	-	6	3	3-7

MOVZX - Move with Zero Extend (386+)

Usage: MOVZX dest,src
Modifies flags: None

Copies the value of the source operand to the destination register with the zeroes extended.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg,reg	-	-	3	3	3

reg,mem	-	-	6	3	3-7

MUL - Unsigned Multiply

Usage: MUL src
Modifies flags: CF OF (AF,PF,SF,ZF undefined)

Unsigned multiply of the accumulator by the source. If "src" is a byte value, then AL is used as the other multiplicand and the result is placed in AX. If "src" is a word value, then AX is multiplied by "src" and DX:AX receives the result. If "src" is a double word value, then EAX is multiplied by "src" and EDX:EAX receives the result. The 386+ uses an early out algorithm which makes multiplying any size value in EAX as fast as in the 8 or 16 bit registers.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	70-77	13	9-14	13-18	2
reg16	118-113	21	9-22	13-26	2
reg32	-	-	9-38	13-42	2-4
mem8	(76-83)+EA	16	12-17	13-18	2-4
mem16	(124-139)+EA	24	12-25	13-26	2-4
mem32	-	-	12-21	13-42	2-4

NEG - Two's Complement Negation

Usage: NEG dest
Modifies flags: AF CF OF PF SF ZF

Subtracts the destination from 0 and saves the 2s complement of "dest" back into "dest".

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg	3	2	2	1	2	
mem	16+EA	7	6	3	2-4	(W88=24+EA)

NOP - No Operation (90h)

Usage: NOP
Modifies flags: None

This is a do nothing instruction. It results in occupation of both space and time and is most useful for patching code segments. (This is the original XCHG AL,AL instruction)

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	3	3	3	1	1

NOT - One's Compliment Negation (Logical NOT)

Usage: NOT dest
Modifies flags: None

Inverts the bits of the "dest" operand forming the 1s complement.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg	3	2	2	1	2	
mem	16+EA	7	6	3	2-4	(W88=24+EA)

OR - Inclusive Logical OR

Usage: OR dest,src
Modifies flags: CF OF PF SF ZF (AF undefined)

Logical inclusive OR of the two operands returning the result in the destination. Any bit set in either operand will be set in the destination.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	16+EA	7	7	3	2-4	(W88=24+EA)
reg,mem	9+EA	7	6	2	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem8,immed8	17+EA	7	7	3	3-6	
mem16,immed16	25+EA	7	7	3	3-6	
accum,immed	4	3	2	1	2-3	

OUT - Output Data to Port

Usage: OUT port,accum
Modifies flags: None

Transfers byte in AL,word in AX or dword in EAX to the specified hardware port address. If the port number is in the range of 0-255 it can be specified as an immediate. If greater than 255 then the port number must be specified in DX. Since the PC only decodes 10 bits of the port address, values over 1023 can only be decoded by third party vendor equipment and also map to the port range 0-1023.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
immed8,accum	10/14	3	10	16	2	
immed8,accum (PM)	-	-	4/24	11/31/29	2	
DX,accum	8/12	3	11	16	1	
DX,accum (PM)	-	-	5/25	10/30/29	1	

- 386+ protected mode timings depend on privilege levels.

  first number is the timing when: CPL ó IOPL
  second number is the timing when: CPL > IOPL
  third number is the timing when: virtual mode on 486 processor

OUTS - Output String to Port (80188+)

Usage: OUTS port,src
OUTSB
OUTSW
OUTSD (386+)
Modifies flags: None

Transfers a byte, word or doubleword from "src" to the hardware port specified in DX. For instructions with no operands the "src" is located at DS:SI and SI is incremented or decremented by the size of the operand or the size dictated by the instruction format. When the Direction Flag is set SI is decremented, when clear, SI is incremented. If the port number is in the range of 0-255 it can be specified as an immediate. If greater than 255 then the port number must be specified in DX. Since the PC only decodes 10 bits of the port address, values over 1023 can only be decoded by third party vendor equipment and also map to the port range 0-1023.

	Clocks				Size
Operands	808x	286	386	486	Bytes
port,src	-	5	14	17	1
port,src (PM)	-	-	8/28	10/32/30	1

- 386+ protected mode timings depend on privilege levels.

  first number is the timing when: CPL ó IOPL
  second number is the timing when: CPL > IOPL
  third number is the timing when: virtual mode on 486 processor

POP - Pop Word off Stack

Usage: POP dest
Modifies flags: None

Transfers word at the current stack top (SS:SP) to the destination then increments SP by two to point to the new stack top. CS is not a valid destination.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16	8	5	4	4	1
reg32	4	-	-	4	1
segreg	8	5	7	3	1
mem16	17+EA	5	5	6	2-4
mem32	5	-	-	6	2-4

POPA/POPAD - Pop All Registers onto Stack (80188+)

Usage: POPA
POPAD (386+)
Modifies flags: None

Pops the top 8 words off the stack into the 8 general purpose 16/32 bit registers. Registers are popped in the following order: (E)DI, (E)SI, (E)BP, (E)SP, (E)DX, (E)CX and (E)AX. The (E)SP value popped from the stack is actually discarded.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	19	24	9	1

POPF/POPFD - Pop Flags off Stack

```
        Usage:  POPF
                POPFD  (386+)
        Modifies flags: all flags

        Pops word/doubleword from stack into the Flags Register and then
        increments SP by 2 (for POPF) or 4 (for POPFD).

                                  Clocks                  Size
        Operands         808x  286   386   486           Bytes

        none             8/12   5     5     9             1  (W88=12)
        none  (PM)        -     -     5     6             1


PUSH - Push Word onto Stack

        Usage:  PUSH    src
                PUSH    immed   (80188+ only)
        Modifies flags: None

        Decrements SP by the size of the operand (two or four, byte values
        are sign extended) and transfers one word from source to the stack
        top (SS:SP).

                                  Clocks                  Size
        Operands         808x  286   386   486           Bytes

        reg16           11/15   3     2     1             1
        reg32             -     -     2     1             1
        mem16           16+EA   5     5     4            2-4  (W88=24+EA)
        mem32             -     -     5     4            2-4
        segreg          10/14   3     2     3             1
        immed             -     3     2     1            2-3


PUSHA/PUSHAD - Push All Registers onto Stack  (80188+)

        Usage:  PUSHA
                PUSHAD  (386+)
        Modifies flags: None

        Pushes all general purpose registers onto the stack in the
following
        order: (E)AX, (E)CX, (E)DX, (E)BX, (E)SP, (E)BP, (E)SI, (E)DI.  The
        value of SP is the value before the actual push of SP.

                                  Clocks                  Size
        Operands         808x  286   386   486           Bytes

        none              -     19    24    11            1

PUSHF/PUSHFD - Push Flags onto Stack

        Usage:  PUSHF
                PUSHFD  (386+)
        Modifies flags: None

        Transfers the Flags Register onto the stack.  PUSHF saves a 16 bit
        value while PUSHFD saves a 32 bit value.

                                  Clocks                  Size
        Operands         808x  286   386   486           Bytes
```

```
        none            10/14   3     4     4            1
        none  (PM)        -     -     4     3            1
```

RCL - Rotate Through Carry Left

```
        Usage:  RCL     dest,count
        Modifies flags: CF OF
```

```
        Rotates the bits in the destination to the left "count" times with
        all data pushed out the left side re-entering on the right.  The
        Carry Flag holds the last bit rotated out.

                                 Clocks                 Size
        Operands         808x  286   386   486          Bytes

        reg,1              2     2     9     3           2
        mem,1            15+EA   7     10    4           2-4   (W88=23+EA)
        reg,CL            8+4n  5+n    9    8-30         2
        mem,CL         20+EA+4n 8+n    10   9-31         2-4   (W88=28+EA+4n)
        reg,immed8         -    5+n     9   8-30         3
        mem,immed8         -    8+n    10   9-31         3-5
```

RCR - Rotate Through Carry Right

```
        Usage:  RCR     dest,count
        Modifies flags: CF OF
```

```
        Rotates the bits in the destination to the right "count" times with
        all data pushed out the right side re-entering on the left.  The
        Carry Flag holds the last bit rotated out.

                                 Clocks                 Size
        Operands         808x  286   386   486          Bytes

        reg,1              2     2     9     3           2
        mem,1            15+EA   7     10    4           2-4   (W88=23+EA)
        reg,CL            8+4n  5+n    9    8-30         2
        mem,CL         20+EA+4n 8+n    10   9-31         2-4
(W88=28+EA+4n)
        reg,immed8         -    5+n    9    8-30         3
        mem,immed8         -    8+n    10   9-31         3-5
```

REP - Repeat String Operation

```
        Usage:  REP
        Modifies flags: None

        Repeats execution of string instructions while CX != 0.  After
        each string operation, CX is decremented and the Zero Flag is
        tested.  The combination of a repeat prefix and a segment override
        on CPU's before the 386 may result in errors if an interrupt occurs
```

before CX=0. The following code shows code that is susceptible to this and how to avoid it:

```
 again:  rep movs  byte ptr ES:[DI],ES:[SI]   ; vulnerable instr.
             jcxz  next              ; continue if REP successful
             loop  again             ; interrupt goofed count
 next:
```

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2		1

REPE/REPZ - Repeat Equal / Repeat Zero

Usage: REPE
REPZ
Modifies flags: None

Repeats execution of string instructions while CX != 0 and the Zero Flag is set. CX is decremented and the Zero Flag tested after each string operation. The combination of a repeat prefix and a segment override on processors other than the 386 may result in errors if an interrupt occurs before CX=0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2		1

REPNE/REPNZ - Repeat Not Equal / Repeat Not Zero

Usage: REPNE
REPNZ
Modifies flags: None

Repeats execution of string instructions while CX != 0 and the Zero Flag is clear. CX is decremented and the Zero Flag tested after each string operation. The combination of a repeat prefix and a segment override on processors other than the 386 may result in errors if an interrupt occurs before CX=0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2		1

RET/RETF - Return From Procedure

Usage: RET nBytes
RETF nBytes
RETN nBytes
Modifies flags: None

Transfers control from a procedure back to the instruction address saved on the stack. "n bytes" is an optional number of bytes to release. Far returns pop the IP followed by the CS, while near returns pop only the IP register.

	Clocks				Size
Operands	808x	286	386	486	Bytes

```
        retn            16/20  11+m  10+m    5          1
        retn immed      20/24  11+m  10+m    5          3
        retf            26/34  15+m  18+m    13         1
        retf (PM, same priv.)   -    32+m    18         1
        retf (PM, lesser priv.) -      68    33         1
        retf immed      25/33  15+m  18+m    14         3
        retf immed (PM, same priv.)  32+m    17         1
        retf immed (PM, lesser priv.)  68    33         1
```

ROL - Rotate Left

```
        Usage:  ROL     dest,count
        Modifies flags: CF OF

        ÚÄ¿     ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
        ³C³<þÂÄþ³7 <ÄÄÄÄÄÄÄÄÄÄ 0³<Ä¿
        ÀÄÙ  ³  ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ  ³
             ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Rotates the bits in the destination to the left "count" times with all data pushed out the left side re-entering on the right. The Carry Flag will contain the value of the last bit rotated out.

```
                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg,1              2     2     3     3            2
        mem,1            15+EA   7     7     4           2-4  (W88=23+EA)
        reg,CL             8+4n  5+n    3     3           2
        mem,CL         20+EA+4n 8+n    7     4           2-4
(W88=28+EA+4n)
        reg,immed8         -     5+n    3     2           3
        mem,immed8         -     8+n    7     4          3-5
```

ROR - Rotate Right

```
        Usage:  ROR     dest,count
        Modifies flags: CF OF

           ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿     ÚÄ¿
        ÚÄ>³7 þÄÄÄÄÄÄÄÄÄ> 0³þÄÂÄ>³C³
        ³  ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ  ³  ÀÄÙ
        ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
```

Rotates the bits in the destination to the right "count" times with all data pushed out the right side re-entering on the left. The Carry Flag will contain the value of the last bit rotated out.

```
                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg,1              2     2     3     3            2
        mem,1            15+EA   7     7     4           2-4  (W88=23+EA)
        reg,CL             8+4n  5+n    3     3           2
        mem,CL         20+EA+4n 8+n    7     4           2-4
(W88=28+EA+4n)
        reg,immed8         -     5+n    3     2           3
        mem,immed8         -     8+n    7     4          3-5
```

SAHF - Store AH Register into FLAGS

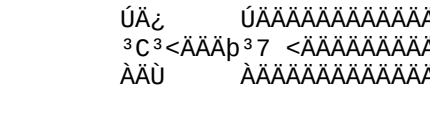
Usage: SAHF
Modifies flags: AF CF PF SF ZF

Transfers bits 0-7 of AH into the Flags Register. This includes AF, CF, PF, SF and ZF.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	4	2	3	2	1

SAL/SHL - Shift Arithmetic Left / Shift Logical Left

Usage: SAL dest,count
SHL dest,count
Modifies flags: CF OF PF SF ZF (AF undefined)

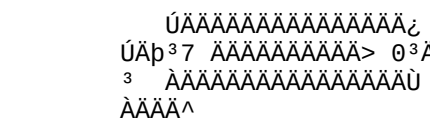


Shifts the destination left by "count" bits with zeroes shifted in on right. The Carry Flag contains the last bit shifted out.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,1	2	2	3	3	2	
mem,1	15+EA	7	7	4	2-4	(W88=23+EA)
reg,CL	8+4n	5+n	3	3	2	
mem,CL	20+EA+4n	8+n	7	4	2-4	(W88=28+EA+4n)
reg,immed8	-	5+n	3	2	3	
mem,immed8	-	8+n	7	4	3-5	

SAR - Shift Arithmetic Right

Usage: SAR dest,count
Modifies flags: CF OF PF SF ZF (AF undefined)



Shifts the destination right by "count" bits with the current sign bit replicated in the leftmost bit. The Carry Flag contains the last bit shifted out.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,1	2	2	3	3	2	
mem,1	15+EA	7	7	4	2-4	(W88=23+EA)
reg,CL	8+4n	5+n	3	3	2	
mem,CL	20+EA+4n	8+n	7	4	2-4	(W88=28+EA+4n)
reg,immed8	-	5+n	3	2	3	
mem,immed8	-	8+n	7	4	3-5	

SBB - Subtract with Borrow/Carry

Usage: SBB dest,src
Modifies flags: AF CF OF PF SF ZF

Subtracts the source from the destination, and subtracts 1 extra if the Carry Flag is set. Results are returned in "dest".

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	16+EA	7	6	3	2-4	(W88=24+EA)
reg,mem	9+EA	7	7	2	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem,immed	17+EA	7	7	3	3-6	(W88=25+EA)
accum,immed	4	3	2	1	2-3	

SCAS - Scan String (Byte, Word or Doubleword)

Usage: SCAS string
SCASB
SCASW
SCASD (386+)
Modifies flags: AF CF OF PF SF ZF

Compares value at ES:DI (even if operand is specified) from the accumulator and sets the flags similar to a subtraction. DI is incremented/decremented based on the instruction format (or operand size) and the state of the Direction Flag. Use with REP prefixes.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
string	15	7	7	6	1	(W88=19)

SETAE/SETNB - Set if Above or Equal / Set if Not Below (386+)

Usage: SETAE dest
SETNB dest
(unsigned, 386+)
Modifies flags: none

Sets the byte in the operand to 1 if the Carry Flag is clear otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETB/SETNAE - Set if Below / Set if Not Above or Equal (386+)

Usage: SETB dest
SETNAE dest
(unsigned, 386+)
Modifies flags: none

Sets the byte in the operand to 1 if the Carry Flag is set
otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETBE/SETNA - Set if Below or Equal / Set if Not Above (386+)

Usage: SETBE dest
SETNA dest
(unsigned, 386+)
Modifies flags: none

Sets the byte in the operand to 1 if the Carry Flag or the Zero
Flag is set, otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETE/SETZ - Set if Equal / Set if Zero (386+)

Usage: SETE dest
SETZ dest
Modifies flags: none

Sets the byte in the operand to 1 if the Zero Flag is set,
otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETNE/SETNZ - Set if Not Equal / Set if Not Zero (386+)

Usage: SETNE dest
SETNZ dest
Modifies flags: none

Sets the byte in the operand to 1 if the Zero Flag is clear,
otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETL/SETNGE - Set if Less / Set if Not Greater or Equal (386+)

Usage: SETL dest
SETNGE dest
(signed, 386+)

Modifies flags: none

Sets the byte in the operand to 1 if the Sign Flag is not equal to the Overflow Flag, otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETGE/SETNL - Set if Greater or Equal / Set if Not Less (386+)

Usage: SETGE dest
SETNL dest
(signed, 386+)
Modifies flags: none

Sets the byte in the operand to 1 if the Sign Flag equals the Overflow Flag, otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETLE/SETNG - Set if Less or Equal / Set if Not greater or Equal (386+)

Usage: SETLE dest
SETNG dest
(signed, 386+)
Modifies flags: none

Sets the byte in the operand to 1 if the Zero Flag is set or the Sign Flag is not equal to the Overflow Flag, otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETG/SETNLE - Set if Greater / Set if Not Less or Equal (386+)

Usage: SETG dest
SETNLE dest
(signed, 386+)
Modifies flags: none

Sets the byte in the operand to 1 if the Zero Flag is clear or the Sign Flag equals to the Overflow Flag, otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETS - Set if Signed (386+)

Usage: SETS dest
Modifies flags: none

Sets the byte in the operand to 1 if the Sign Flag is set, otherwise
sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETNS - Set if Not Signed (386+)

Usage: SETNS dest
Modifies flags: none

Sets the byte in the operand to 1 if the Sign Flag is clear,
otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETC - Set if Carry (386+)

Usage: SETC dest
Modifies flags: none

Sets the byte in the operand to 1 if the Carry Flag is set,
otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETNC - Set if Not Carry (386+)

Usage: SETNC dest
Modifies flags: none

Sets the byte in the operand to 1 if the Carry Flag is clear,
otherwise sets the operand to 0.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg8	-	-	4	3	3
mem8	-	-	5	4	3

SETO - Set if Overflow (386+)

```
        Usage:  SETO    dest
        Modifies flags: none

        Sets the byte in the operand to 1 if the Overflow Flag is set,
        otherwise sets the operand to 0.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg8              -     -     4     3            3
        mem8              -     -     5     4            3


SETNO - Set if Not Overflow (386+)

        Usage:  SETNO   dest
        Modifies flags: none

        Sets the byte in the operand to 1 if the Overflow Flag is clear,
        otherwise sets the operand to 0.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg8              -     -     4     3            3
        mem8              -     -     5     4            3

SETP/SETPE - Set if Parity / Set if Parity Even  (386+)

        Usage:  SETP    dest
                SETPE   dest
        Modifies flags: none

        Sets the byte in the operand to 1 if the Parity Flag is set,
        otherwise sets the operand to 0.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg8              -     -     4     3            3
        mem8              -     -     5     4            3


SETNP/SETPO - Set if No Parity / Set if Parity Odd (386+)

        Usage:  SETNP   dest
                SETPO   dest
        Modifies flags: none

        Sets the byte in the operand to 1 if the Parity Flag is clear,
        otherwise sets the operand to 0.

                                Clocks                  Size
        Operands         808x  286   386   486          Bytes

        reg8              -     -     4     3            3
        mem8              -     -     5     4            3


SGDT - Store Global Descriptor Table (286+ privileged)

        Usage:  SGDT    dest
        Modifies flags: none
```

Stores the Global Descriptor Table (GDT) Register into the specified operand.

	Clocks				Size
Operands	808x	286	386	486	Bytes
mem64	-	11	9	10	5

SIDT - Store Interrupt Descriptor Table (286+ privileged)

Usage: SIDT dest
Modifies flags: none

Stores the Interrupt Descriptor Table (IDT) Register into the specified operand.

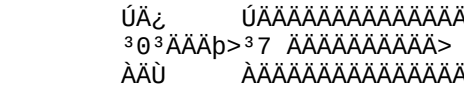
	Clocks				Size
Operands	808x	286	386	486	Bytes
mem64	-	12	9	10	5

SHL - Shift Logical Left

See: SAL

SHR - Shift Logical Right

Usage: SHR dest,count
Modifies flags: CF OF PF SF ZF (AF undefined)



Shifts the destination right by "count" bits with zeroes shifted in on the left. The Carry Flag contains the last bit shifted out.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,1	2	2	3		2	
mem,1	15+EA	7	7		2-4	(W88=23+EA)
reg,CL	8+4n	5+n	3		2	
mem,CL	20+EA+4n	8+n	7		2-4	(W88=28+EA+4n)
reg,immed8	-	5+n	3		3	
mem,immed8	-	8+n	7		3-5	

SHLD/SHRD - Double Precision Shift (386+)

Usage: SHLD dest,src,count
SHRD dest,src,count
Modifies flags: CF PF SF ZF (OF,AF undefined)

SHLD shifts "dest" to the left "count" times and the bit positions opened are filled with the most significant bits of "src". SHRD shifts "dest" to the right "count" times and the bit positions opened are filled with the least significant bits of the second operand. Only the 5 lower bits of "count" are used.

Operands	Clocks 808x	286	386	486	Size Bytes
reg16,reg16,immed8	-	-	3	2	4
reg32,reg32,immed8	-	-	3	2	4
mem16,reg16,immed8	-	-	7	3	6
mem32,reg32,immed8	-	-	7	3	6
reg16,reg16,CL	-	-	3	3	3
reg32,reg32,CL	-	-	3	3	3
mem16,reg16,CL	-	-	7	4	5
mem32,reg32,CL	-	-	7	4	5

SLDT - Store Local Descriptor Table (286+ privileged)

Usage: SLDT dest
Modifies flags: none

Stores the Local Descriptor Table (LDT) Register into the specified operand.

Operands	Clocks 808x	286	386	486	Size Bytes
reg16	-	2	2	2	3
mem16	-	2	2	3	5

SMSW - Store Machine Status Word (286+ privileged)

Usage: SMSW dest
Modifies flags: none

Store Machine Status Word (MSW) into "dest".

Operands	Clocks 808x	286	386	486	Size Bytes
reg16	-	2	10	2	3
mem16	-	3	3	3	5

STC - Set Carry

Usage: STC
Modifies flags: CF

Sets the Carry Flag to 1.

Operands	Clocks 808x	286	386	486	Size Bytes
none	2	2	2	2	1

STD - Set Direction Flag

Usage: STD
Modifies flags: DF

Sets the Direction Flag to 1 causing string instructions to auto-decrement SI and DI instead of auto-increment.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2	2	1

STI - Set Interrupt Flag (Enable Interrupts)

Usage: STI
Modifies flags: IF

Sets the Interrupt Flag to 1, which enables recognition of all hardware interrupts. If an interrupt is generated by a hardware device, an End of Interrupt (EOI) must also be issued to enable other hardware interrupts of the same or lower priority.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	2	2	2	5	1

STOS - Store String (Byte, Word or Doubleword)

Usage: STOS dest
STOSB
STOSW
STOSD
Modifies flags: None

Stores value in accumulator to location at ES:(E)DI (even if operand is given). (E)DI is incremented/decremented based on the size of the operand (or instruction format) and the state of the Direction Flag. Use with REP prefixes.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
dest	11	3	4	5	1	(W88=15)

STR - Store Task Register (286+ privileged)

Usage: STR dest
Modifies flags: None

Stores the current Task Register to the specified operand.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16	-	2	2	2	3
mem16	-	3	2	3	5

SUB - Subtract

Usage: SUB dest,src
Modifies flags: AF CF OF PF SF ZF

The source is subtracted from the destination and the result is stored in the destination.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	16+EA	7	6	3	2-4	(W88=24+EA)
reg,mem	9+EA	7	7	2	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem,immed	17+EA	7	7	3	3-6	(W88=25+EA)
accum,immed	4	3	2	1	2-3	

TEST - Test For Bit Pattern

Usage: TEST dest,src
Modifies flags: CF OF PF SF ZF (AF undefined)

Performs a logical AND of the two operands updating the flags register without saving the result.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	1	1	2	
reg,mem	9+EA	6	5	1	2-4	(W88=13+EA)
mem,reg	9+EA	6	5	2	2-4	(W88=13+EA)
reg,immed	5	3	2	1	3-4	
mem,immed	11+EA	6	5	2	3-6	
accum,immed	4	3	2	1	2-3	

VERR - Verify Read (286+ protected)

Usage: VERR src
Modifies flags: ZF

Verifies the specified segment selector is valid and is readable at the current privilege level. If the segment is readable, the Zero Flag is set, otherwise it is cleared.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16	-	14	10	11	3
mem16	-	16	11	11	5

VERW - Verify Write (286+ protected)

Usage: VERW src
Modifies flags: ZF

Verifies the specified segment selector is valid and is ratable at the current privilege level. If the segment is writable, the Zero Flag is set, otherwise it is cleared.

	Clocks				Size
Operands	808x	286	386	486	Bytes
reg16	-	14	15	11	3
mem16	-	16	16	11	5

WAIT/FWAIT - Event Wait

Usage: WAIT
FWAIT
Modifies flags: None

CPU enters wait state until the coprocessor signals it has finished its operation. This instruction is used to prevent the CPU from accessing memory that may be temporarily in use by the coprocessor. WAIT and FWAIT are identical.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	4	3	6+	1-3	1

WBINVD - Write-Back and Invalidate Cache (486+)

Usage: WBINVD
Modifies flags: None

Flushes internal cache, then signals the external cache to write back current data followed by a signal to flush the external cache.

	Clocks				Size
Operands	808x	286	386	486	Bytes
none	-	-	-	5	2

XCHG - Exchange

Usage: XCHG dest,src
Modifies flags: None

Exchanges contents of source and destination.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	4	3	3	3	2	
mem,reg	17+EA	5	5	5	2-4	(W88=25+EA)
reg,mem	17+EA	5	5	3	2-4	(W88=25+EA)
accum,reg	3	3	3	3	1	
reg,accum	3	3	3	3	1	

XLAT/XLATB - Translate

Usage: XLAT translation-table
XLATB (masm 5.x)
Modifies flags: None

Replaces the byte in AL with byte from a user table addressed by BX. The original value of AL is the index into the translate table.
The best way to discripe this is MOV AL,[BX+AL]

	Clocks				Size
Operands	808x	286	386	486	Bytes
table offset	11	5	5	4	1

XOR - Exclusive OR

Usage: XOR dest,src
Modifies flags: CF OF PF SF ZF (AF undefined)

Performs a bitwise exclusive OR of the operands and returns the result in the destination.

	Clocks				Size	
Operands	808x	286	386	486	Bytes	
reg,reg	3	2	2	1	2	
mem,reg	16+EA	7	6	3	2-4	(W88=24+EA)
reg,mem	9+EA	7	7	2	2-4	(W88=13+EA)
reg,immed	4	3	2	1	3-4	
mem,immed	17+EA	7	7	3	3-6	(W88=25+EA)
accum,immed	4	3	2	1	2-3	

LỜI NÓI ĐẦU

Công nghệ thông tin đang được ứng dụng rộng rãi trong nhiều lĩnh vực khoa học công nghệ và cuộc sống thường nhật. Bên cạnh khối lượng phần mềm hệ thống và ứng dụng đồ sộ, công nghệ phần cứng cũng phát triển vô cùng nhanh chóng. Có thể nói các hệ thống máy tính được cải thiện trong những khoảng thời gian rất ngắn, càng ngày càng nhanh hơn, mạnh hơn và hiện đại hơn.

Những kiến thức cơ bản về về phần cứng của các hệ thống máy tính luôn luôn là đòi hỏi cấp thiết của những người chọn công nghệ thông tin làm định hướng cho nghề nghiệp và sự nghiệp khoa học trong tương lai.

Giáo trình Kỹ thuật Vi xử lý này được viết trên cơ sở những bài giảng theo sát đề cương môn học đã được thực hiện tại Khoa Công nghệ thông tin trực thuộc Trường đại học Thái Nguyên từ khi thành lập đến nay, và luôn luôn được sửa chữa, bổ sung để đáp ứng nhu cầu kiến thức của sinh viên học tập tại Khoa.

Giáo trình được chia thành 5 chương:

Chương I giới thiệu những kiến thức tổng quan được sử dụng trong kỹ thuật Vi xử lý các hệ đếm, cách thức biểu diễn thông tin trong các hệ Vi xử lý và máy tính, cũng như nhìn nhận qua về lịch sử phát triển của các trung tâm Vi xử lý.

Chương II giới thiệu cấu trúc và hoạt động của các đơn vị xử lý trung tâm từ µP8085 đến các cấu trúc của Vi xử lý họ 80x86, các cấu trúc RISC và CISC. Do những ứng dụng thực tế rộng lớn trong đời sống, trong chương II có giới thiệu thêm cấu trúc và chức năng của chip Vi xử lý chuyên dụng µC8051.

Chương III cung cấp những kiến thức về tổ chức bộ nhớ cho một hệ Vi xử lý, kỹ thuật và các bước xây dựng vỉ nhớ ROM, RAM cho hệ Vi xử lý.

Chương IV đi sâu khảo sát một số mạch chức năng khả lập trình như mạch điều khiển vào/ra dữ liệu song song, mạch điều khiển vào/ra dữ liệu nối tiếp, mạch định thời và mạch điều khiển ngắt.

Chương V giới thiệu các cấu trúc và cách xây dựng, phối ghép một số thiết bị vào/ra cơ bản cho một hệ Vi xử lý như bàn phím Hexa, hệ thống chỉ thị 7 thanh, bàn phím máy tính và màn hình.

Cuốn giáo trình chắc chắn có nhiều thiếu sót, rất mong đợc sự góp ý của các độc giả. Mọi ý kiến đóng góp xin gửi theo địa chỉ:

Bộ môn Kỹ thuật máy tính Khoa Công nghệ Thông tin

Đại học Thái Nguyên

Thái Nguyên

*Hoặc theo địa chỉ Email **dongnt@hn.vnn.vn***

Nhóm biên soạn

MỤC LỤC

CHƯƠNG I. TỔNG QUAN VỀ CÁC HỆ VI XỬ LÝ

I.1 Các hệ đếm

Hệ đếm thông dụng nhất trong đời sống là hệ đếm cơ số 10 (thập phân – Decimal), sử dụng 10 ký tự số từ 0 đến 9. Ngoài ra, trong sản xuất, kinh doanh còn có khi sử dụng hệ đếm cơ số 12 (tá – dozen).

Trong các hệ thống máy tính, để xử lý, tính toán, ta sử dụng hệ đếm cơ số 2 (nhị phân – Binary), hệ cơ số 8 (bát phân – Octal), hệ cơ số 16 (Hexa). Tuy nhiên, việc nhập dữ liệu hay đưa kết quả xử lý, ta lại dùng hệ đếm cơ số 10.

Một số *N* trong một hệ đếm bất kỳ có *n+l* chữ số , trong đó gồm *n* chữ số thuộc phần nguyên và *l* chữ số thuộc phần thập phân, được triển khai theo công thức tổng quát:

$$N = \sum_{k=-l}^{n} a_k R^k$$ trong đó:

R là cơ số của hệ đếm

a_k là trọng của chữ số ở vị trí thứ k ($0 \le a_k < R$)

$\{ a_k \}_R = \{0, 1, 2, 3, \ldots, R - 1\}$

l, *n* là số nguyên

$N = a_n a_{n-1} \ldots a_1 a_0, a_{-1} a_{-2} \ldots a_{-l}$

Theo công thức trên, các số được biểu diễn trong các hệ đếm khác nhau sẽ như sau:

I.1.1 Hệ đếm thập phân (R = 10 - Decimal)

$\{ a_k \}_D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$123,45_D = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2}$

I.1.2 Hệ đếm nhị phân (R = 2 - Binary)

$\{ a_k \}_B = \{0, 1\}$

$11011.01_B = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} =$

$= 16 + 8 + 0 + 2 + 1 + 0 + 0,25 = 27,25_D$

I.1.3 Hệ đếm bát phân (R = 8 - Octal)

$\{ a_k \}_O = \{0, 1, 2, 3, 4, 5, 6, 7\}$

$653,12_O = 6 \times 8^2 + 5 \times 8^1 + 3 \times 8^0 + 1 \times 8^{-1} + 2 \times 8^{-2} =$

$= 384 + 40 + 3 + 0,125 + 0,03125 = 427,1562_D$

Lưu ý: Các chữ số trong hệ này có thể biểu diễn nhờ 3 ký tự số ("0" và "1") trong hệ đếm nhị phân theo bảng sau:

Octal	Binary	Octal	Binary	Octal	Binary	Octal	Binary
0_O	000_B	2_O	010_B	4_O	100_B	6_O	110_B
1_O	001_B	3_O	011_B	5_O	101_B	7_O	111_B

I.1.4 Hệ đếm 16 (R = 16 - Hexa)

$\{ a_k \}_H = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

$3A7,C_H = 3 \times 16^2 + 10 \times 16^1 + 7 \times 16^0 + 12 \times 16^{-1} =$

$= 768 + 160 + 7 + 0,75 = 935,75_D$

Lưu ý: Một giá trị ký tự số Hexa có thể biểu diễn thông qua 4 ký tự số ở hệ nhị phân theo bảng sau:

Hexa	Binary	Hexa	Binary	Hexa	Binary	Hexa	Binary
0_H	0000_B	4_H	0100_B	8_H	1000_B	C_H	1100_B
1_H	0001_B	5_H	0101_B	9_H	1001_B	D_H	1101_B
2_H	0010_B	6_H	0110_B	A_H	1010_B	E_H	1110_B
3_H	0011_B	7_H	0111_B	B_H	1011_B	F_H	1111_B

Nhận xét:

1. Trong các hệ đếm vừa được nêu, hệ đếm cơ số 2 có rất nhiều ưu điểm khi xử lý trong máy tính. Thứ nhất, việc mô phỏng giá trị của một ký tự số là rất đơn giản: chỉ cần một phần tử có hai trạng thái khác biệt. Sử dụng bản chất vật lý của vật mang thông tin để biểu diễn hai trạng thái này rất dễ thực hiện. Trên dây dẫn điện là các trường hợp có dòng điện (tương ứng với trọng số là 1) hoặc không có dòng điện (tương ứng với trọng số là 0).
2. Việc chuyển đổi giữa hai giá trị 0 hoặc 1 có thể thực hiện thông qua một công tắc, trong thực tế là các phần tử logic điện tử thực hiện các chức năng của khoá điện tử: đóng (dòng điện đi qua được) hoặc mở (dòng điện không đi qua).

I.2 Chuyển đổi lẫn nhau giữa các hệ đếm

I.2.1 Hệ nhị phân và hệ thập phân

a) Từ nhị phân sang thập phân: Sử dụng biểu thức triển khai tổng quát đã nêu, cộng tất cả các số hạng theo giá trị số thập phân, tổng số là dạng thập phân của số nhị phân đã cho.

Ví dụ: $11011.11_B = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$

$$= 16 + 8 + 0 + 2 + 1 + 0.5 + 0.25 = 27.75_D$$

b) Từ thập phân sang nhị phân:

Phần nguyên: Ta có đẳng thức sau (vế trái là số thập phân, vế phải là biểu diễn nhị phân của số đó):

$$S_D = k_n 2^n + k_{n-1}2^{n-1} + k_{n-2}2^{n-2} + \dots k_1 2^1 + k_0 2^0 + =$$

$$= 2(k_n 2^{n-1} + k_{n-1}2^{n-2} + k_{n-2}2^{n-3} + \dots + k_1) + k_0$$

Vì $k_i = \{0, 1\}$, đồng phân với số 0, 1 trong số thập phân, nên ta có thể viết:

$$\frac{S_D - k_0}{2} = k_n 2^{n-1} + k_{n-1}2^{n-2} + k_{n-2}2^{n-3} + \dots + k_1 = 2(k_n 2^{n-2} + k_{n-1}2^{n-3} + \dots + k_2) + k_1$$

Thấy rằng: Ký tự đầu tiên của số nhị phân là k_0, đúng với số dư khi chia S_D cho 2, ký tự tiếp theo, k_1 chính là số dư khi chia thương cho 2, v. v… nên ta có thể tìm tất cả các ký tự khác như sau:

Ví dụ: Đổi số 173_D ra số nhị phân

173	2	dư 1	k_0
86	2	dư 0	k_1
43	2	dư 1	k_2
21	2	dư 1	k_3
10	2	dư 0	k_4
5	2	dư 1	k_5
2	2	dư 0	k_6
1	2	dư 1	k_7
0			

Vậy $173_D = 10101101_B$

Phần phân số : Đẳng thức quan hệ giữa số thập phân và số nhị phân (phần phân số) (vế trái là số thập phân, vế phải là số nhị phân) như sau:

$$S_D = k_{-1}2^{-1} + k_{-2}2^{-2} + k_{-3}2^{-3} + \dots k_{-m+1}2^{-m+1} + k_{-m}2^{-m}$$

$$2S_D = k_{-1} + (k_{-2}2^{-1} + k_{-3}2^{-2} + \dots k_{-m+1}2^{-m+2} + k_{-m}2^{-m+1})$$

Thấy rằng k_{-1} trở thành phần nguyên của vế phải, vậy:

$$2S_D - k_{-1} = (k_{-2}2^{-1} + k_{-3}2^{-2} + \dots k_{-m+1}2^{-m+2} + k_{-m}2^{-m+1})$$

$$2(2S_D - k_{-1}) = k_{-2} + (k_{-3}2^{-1} + \dots k_{-m+1}2^{-m+3} + k_{-m}2^{-m+2})$$

k_{-2} là phần nguyên tiếp theo của vế phải có thể bằng "0" hoặc bằng "1". Tiếp tục tương tự, thu được các ký tự số của các phần tử còn lại.

Ví dụ: Chuyển đổi số 0.8128 thành số nhị phân

Thực hiện phép nhân liên tiếp với 2, phần nguyên của tích bao giờ cũng là các giá trị hoặc bằng "0" hoặc bằng "1", thu được kết quả sau:

0.8128	x 2	= 1.6256	= 1	+	0.6256
0.6256	x 2	= 1.2512	= 1	+	0.2512
0.2512	x 2	= 0.5024	= 0	+	0.5024
0.5024	x 2	= 1.0048	= 1	+	0.0048
0.0048	x 2	Quá nhỏ có thể bỏ qua			

***Lưu ý*:** Quá trình biến đổi này kết thúc khi phần phân số của tích số bằng 0, tuy nhiên, nếu quá kéo dài, tuỳ theo yêu cầu của độ chính xác dữ liệu khi tính toán và xử lý, có thể bỏ qua.

I.2.2 Hệ nhị phân và hệ Hexa

Chuyển đổi một dữ liệu nhị phân sang hệ Hexa rất đơn giản, nếu chú ý rằng ta có $2^4 = 16$, có nghĩa là một số Hexa tương ứng với một nhóm 4 số của số nhị phân (từ 0 đến F). Vì vậy, khi chuyển đổi, chỉ cần thay nhóm 4 chữ số của số nhị phân bằng một chữ số tương ứng của hệ Hexa như sau:

Tổ hợp nhị phân	*Ký tự số Hexa*	*Tổ hợp nhị phân*	*Ký tự số Hexa*	*Tổ hợp nhị phân*	*Ký tự số Hexa*	*Tổ hợp nhị phân*	*Ký tự số Hexa*
0 0 0 0	*0*	0 1 0 0	*4*	1 0 0 0	*8*	1 1 0 0	*C*
0 0 0 1	*1*	0 1 0 1	*5*	1 0 0 1	*9*	1 1 0 1	*D*
0 0 1 0	*2*	0 1 1 0	*6*	1 0 1 0	*A*	1 1 1 0	*E*
0 0 1 1	*3*	0 1 1 1	*7*	1 0 1 1	*B*	1 1 1 1	*F*

Ví dụ:

110	1101	0011	1001.	0110	0101_B	=	$6D39.65_H$
6	D	3	9.	6	5		

Lưu ý: Phần nguyên được nhóm tính từ vị trí của chữ số có trọng nhỏ nhất, phần phân số được nhóm tính từ vị trí của chữ số có trọng lớn nhất.

Từ cách chuyển đổi trên, dễ dàng nhận ra phép chuyển đổi ngược từ một số hệ Hexa sang số hệ nhị phân bằng cách thay một chữ số trong hệ Hexa bằng một nhóm 4 chữ số trong hệ nhị phân.

Ví dụ: $F5E7.8C_H = 1111\ 0101\ 1110\ 0111.1000\ 1100_B$

F	5	E	7.	8	C_H	= 1111 0101 1110 0111.1000 1100_B
1111	0101	1110	0111	1000	1100	

I.3 Biểu diễn thông tin trong các hệ Vi xử lý

Các hệ Vi xử lý xử lý các thông tin số và chữ. Các thông tin được biểu diễn dưới dạng mã nhất định. Bản chất vật lý của việc biểu diễn thông tin là điện áp ("0" ứng với không có điện áp, "1" ứng với điện áp ở mức quy chuẩn trong mạch điện tử) và việc mã hoá các thông tin số và chữ được tuân theo chuẩn quốc tế. Một biến logic với chỉ hai giá trị duy nhất là "0" hoặc "1" được gọi là một bit. Hai trạng thái này của bit được sử dụng để mã hoá cho tất cả các ký tự (gồm số, chữ và các ký tự đặc biệt khác). Các bit được ghép lại

thành các đơn vị mang thông tin đầy đủ cho các ký tự biểu diễn các số, các ký tự chữ và các ký tự đặc biệt khác.

Bit (BInary digiT) là đơn vị cơ bản của thông tin theo hệ đếm nhị phân. Các mạch điện tử trong máy tính phát hiện sự khác nhau giữa hai trạng thái (điện áp mức "1" và điện áp mức "0") và biểu diễn hai trạng thái đó dưới dạng một trong hai số nhị phân "1" hoặc "0".

Nhóm 8 bit ghép kề liền nhau, tạo thành đơn vị dữ liệu cơ sở của hệ Vi xử lý được gọi là 1 Byte. Do được lưu giữ tương đương với một ký tự (số, chữ hoặc ký tự đặc biệt) nên Byte cũng là đơn vị cơ sở để đo các khả năng lưu giữ, xử lý của hệ Vi xử lý. Các thuật ngữ như KiloByte, MegaByte hay GigaByte thường được dùng làm bội số trong việc đếm Byte, dĩ nhiên theo hệ đếm nhị phân, nghĩa là:

1KiloByte = 1024 Bytes,
1MegaByte = 1024 KiloBytes,
1GigaByte = 1024 MegaBytes.

Các đơn vị này được viết tắt tương ứng là KB, MB và GB.

I.3.1 Mã hoá các thông tin không số

Có hai loại mã phổ cập nhất được sử dụng là mã ASCII và EBCDIC.

- Mã ASCII (American Standard Code for Information Interchange) dùng 7 bits để mã hoá các ký tự
- Mã ABCDIC (Extended Binary Coded Decimal Interchange Code) dùng cả 8 bits (1 Byte) để mã hoá thông tin
- Loại mã được dùng trong ngành bưu điện, trong các máy teletype là mã BAUDOT, chỉ sử dụng 5 bits để mã hoá thông tin.

I.3.2 Mã hoá các thông tin số

Các số được mã hoá theo các loại mã sau:

- Mã nhị phân sử dụng các số được biểu diễn theo hệ đếm nhị phân như đã nêu ở trên
- Mã nhị thập phân (BCD Code – Binary Coded Decimal Code) sử dụng cách nhóm 4 bits nhị phân để biểu diễn một giá trị thập phân từ 0 đến 9. Các giá trị vượt quá giới hạn này (> 9) không được sử dụng.

I.3.3 Biểu diễn dữ liệu số trong máy tính

- Biểu diễn dữ liệu là số nguyên có dấu: Giả sử dùng 2 bytes (16 bits) để biểu diễn một số nguyên có dấu, bit cao nhất (MSB – Most Significant Bit) được dùng để đánh dấu. Số dương có bit dấu S = "0", số âm có bit dấu S = "1".

D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

– Biểu diễn dữ liệu là số thực có dấu: Về nguyên tắc, dấu của số vẫn là giá trị của MSB như đã quy ước ở trên. Có hai dạng số có dấu phẩy được sử dụng trong máy tính: Số dấu phẩy tĩnh (fixed point) và số dấu phẩy động (floatting point).

  - Dấu phẩy tĩnh sẽ phân chia chuỗi chữ số thành phần nguyên và phần phân số. Ví dụ ta có thể viết:

    ±001 1101.0110 1101

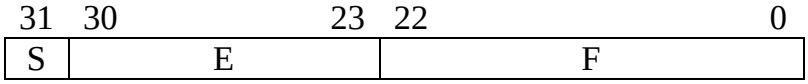
Nhưng nói chung, trong các máy chuyên dụng, thường phải tìm một phương pháp thích hợp để có thể biểu diễn số có dấu phẩy cố định mà dấu phẩy được đặt ngay sau ô dấu, nghĩa là số dấu phẩy tĩnh có dạng:

$$\pm 0.k_n k_{n-1} k_{n-2} \ldots k_1 k_0$$

  - Dấu phẩy động được dùng rất phổ biến, dạng chuẩn tắc như sau:

    $N = \pm F \times 2^{\pm E}$ trong đó F là phần định trị (Mantissa) và E là phần đặc tính (Exponent - số mũ)

Theo nguyên tắc này, một số thực được biểu diễn trong các máy 32 bit như sau:



Số được biểu diễn có giá trị thực tính theo biểu thức:

$$N = (-1)^s \times 2^{E-127} \times F$$

Với cách biểu diễn này, có thể thấy độ lớn của các số như sau:

Số dương: $+3.4 \times 10^{38} < N < +3.4 \times 10^{-38}$

Số âm: $-3.4 \times 10^{38} < N < -3.4 \times 10^{-38}$

Lưu ý: Khi kết quả phép tính vượt quá các giới hạn trên, nếu số mũ (exponent) là dương, sẽ được coi là $-\infty$ hoặc $+\infty$. Trong trường hợp số mũ là âm và vượt qua số mũ cực đại cho phép, kết quả được coi là bằng 0.

Dạng số chính xác gấp đôi (Double precision) được biểu diễn như sau (64 bits):

S	E	F

Và giá trị thực được tính theo biểu thức: $N = (-1)^S \times 2^{E-1023} \times F$.

Cũng cần lưu ý rằng, đối với các dữ liệu số có dấu, để thuận tiện cho xử lý và tính toán, trong máy thường được biểu diễn dưới các dạng mã thuận, mã ngược (complement) hoặc mã bù 2 (two-complement). Giả sử ta có số A=+0.10010, các mã trên đều biểu diễn như nhau, nhưng với số B = -0.10010 thì sẽ được biểu diễn như sau:

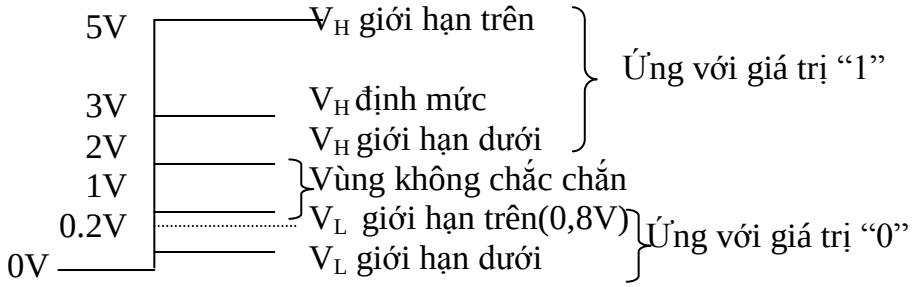
Bình thường A = -0.10010

Mã ngược A = 1.00110 (bù 1, tức là đảo các chữ số trong số đó)

Mã bù 2 A = 1.00111 (tương ứng với bù 1 cộng thêm 1)

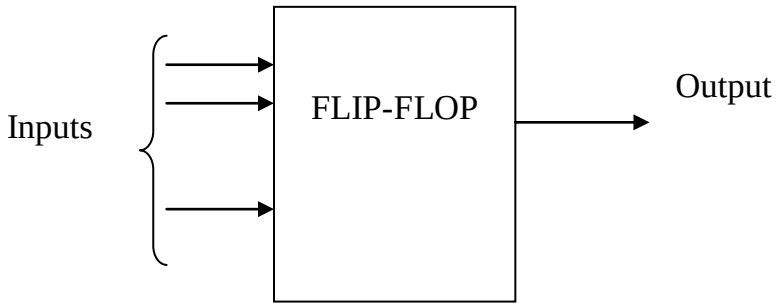
I.3.4 Bản chất vật lý của thông tin trong các hệ Vi xử lý

Trong các hệ Vi xử lý, thông tin về các giá trị "0" hay "1" được biểu diễn thông qua một mức điện áp so với mức chuẩn chung, thường là đất (GND - Ground). Độ lớn của điện áp biểu diễn các giá trị này phụ thuộc vào công nghệ được sử dụng để tạo nên phần tử mang thông tin. Đối với các mạch tổ hợp TTL (Transistor-Transistor-Logic), các mức điện áp được mô tả trong hình I.1



Hình I.1 Phạm vi mức cao "1", thấp"0" của mạch TTL

Ta thường dùng ký hiệu V_H để chỉ mức cao, V_L để chỉ mức thấp. Trong mạch TTL, ta dùng mức cao, mức thấp để chỉ điện áp cao, điện áp thấp so với điện áp chuẩn chung. Các mức cao, thấp không phải là một giá trị cố định, mà là một vùng giới hạn cho phép. Ngoài phạm vi đã nêu, vùng không thuộc hai mức trên là vùng không chắc chắn, không xác định.



Hình I.2 Một phần tử mang thông tin

Vật mang thông tin về các giá trị "0" hoặc "1" là một mạch điện tử đặc biệt mà đầu ra của nó sẽ tương ứng với một trong hai mức trên, được gọi chung là Flip-Flop. Tuỳ theo yêu cầu sử dụng, các Flip-Flop có các khả năng thu nhận các tín hiệu vào và đưa tín hiệu ra theo những quy luật nhất định (Hình I.2)

I.4 Vài nét về thực hiện các phép tính trong hệ đếm nhị phân

Phép cộng và phép trừ hai số nhị phân 1 bit được thực hiện theo quy tắc nêu trong bảng sau:

A		B		∑	Carry (Nhớ)
0	+	0	=	0	0
0	+	1	=	1	0
1	+	0	=	1	0
1	+	1	=	0	1

A		B		Hiệu	Borrow (Mượn)
0	-	0	=	0	0
0	-	1	=	1	1
1	-	0	=	1	0
1	-	1	=	0	0

I.4.1 Phép cộng và phép trừ

a) *Phép cộng đại số các số hạng dấu phẩy cố định*

Đối với phép cộng đại số: Thực hiện bình thường. Trong trường hợp có một toán hạng là một số âm, ta sử dụng mã ngược hoặc mã bù 2 của nó, hiệu chỉnh kết quả theo các quy tắc thông qua các ví dụ minh hoạ sau:

A = 0.10010	A = 0.10010	A = 0.10010
B = -0.11001	$(B)_{ng}$= 1.00110	$(B)_b$= 1.00111
Σ = -0.00111	Σ = 1.11000	Σ = 1.11001
	$(\Sigma)_{ng}$ = -0.00111	$(\Sigma)_b$ = -0.00111

Thấy rằng:

- Số biểu thị kết quả sẽ là mã thuận nếu là một số dương
- Số biểu thị kết quả là mã ngược nếu ta dùng mã ngược đối với số hạng âm và cho kết quả là một số âm
- Số biểu thị kết quả là một số bù 2 nếu dùng mã bù 2 đối với số hạng âm và kết quả là một số âm.

b) *Phép cộng đại số các số hạng dấu phẩy động:*

Đối với phép cộng đại số các số hạng dấu phẩy động, cần tiến hành các bước sau:

- Cân bằng phần đặc tính (số mũ) bằng cách dịch chuyển phần định trị
- Đặc tính của tổng bằng đặc tính chung
- Định trị của tổng bằng tổng các định trị
- Chuẩn hoá kết quả nếu cần.

I.4.2 Phép nhân và phép chia

a) Phép nhân:

Đối với phép nhân các toán hạng dấu phẩy tĩnh, việc quan trọng là phải xác định dấu của kết quả, theo đó dấu của kết quả bằng tổng modulo 2 của các bit dấu. Trị số của tích là kết quả của phép tĩnh tiến (dịch phải) và phép cộng.

Với các toán hạng có dấu phẩy động, dấu của tích được xác định như ở phép nhân với dấu phẩy tĩnh, sau đó tiến hành tìm tích số như sau:

- Cộng phần đặc tính (số mũ), kết quả là đặc tính của tích
- Nhân phần định trị, không để ý đến dấu của các toán hạng
- Chuẩn hoá kết quả nếu cần.

b) Phép chia:

Đối với phép chia các toán hạng dấu phẩy tĩnh, việc quan trọng là phải xác định dấu của kết quả, theo đó dấu của kết quả bằng tổng modulo 2 của các bit dấu. Trị số của thương số là kết quả của phép dịch trái và phép trừ.

Với các toán hạng có dấu phẩy động, dấu của thương số được xác định như ở phép chia với dấu phẩy tĩnh, sau đó tiến hành tìm thương số như sau:

- Trừ phần đặc tính (số mũ), kết quả là đặc tính của thương số
- Chia phần định trị, không để ý đến dấu của các toán hạng
- Chuẩn hoá kết quả nếu cần.

Nhận xét: Dễ dàng nhận thấy rằng các phép tính số học nêu trên chung quy lại vẫn chủ yếu là thực hiện phép cộng và phép dịch (shift).

I.5 Cấu trúc của hệ Vi xử lý và máy vi tính

I.5.1 Vài nét về lịch sử phát triển các trung tâm Vi xử lý

Sự xuất hiện của máy tính điện tử (MTĐT) vào khoảng năm 1948 đã mở ra một trang mới trong nghiên cứu khoa học nói chung và khoa học tính toán nói riêng. Nhưng phải mãi đến năm 1971, các hệ Vi xử lý mới bắt đầu xuất hiện. Sự ra đời của Single chip 4-bit Microprocessor Int$_e$l®4004 (μP4004) vào năm đó thực sự là một cuộc cách mạng trong ngành công nghiệp máy tính . Có thể nói μP4004, với độ dài từ xử lý 4 bits, đã làm đổi thay toàn bộ cách nhìn nhận về các thiết bị đầu cuối của MTĐT, hay các cơ cấu chấp hành trong điều khiển quá trình. μP4004 có thể quản lý trực tiếp 4K từ lệnh 8bit của bộ nhớ chương trình và 5120 bits bộ nhớ dữ liệu RAM. CPU còn có 16 thanh ghi chỉ số được sử dụng làm bộ nhớ tạm cho dữ liệu. Với tập lệnh gồm 46 lệnh, μP4004 đã chiếm được nhiều ưu thế trong các ứng dụng thực tế lúc bấy giờ. Tiếp tục của dòng μP 4bit này là μP4040, có nhiều cải tiến mạnh mẽ so với μP4004 và một loạt các chip chức năng, chip nhớ ra đời.

Trong giai đoạn tiếp theo từ năm 1974 đến 1977, Int$_e$l® đã đi đầu trong việc chế tạo các CPU 8bit, μP8008, μP8080 và đặc biệt là μP8085, những CPU có BUS dữ liệu 8 bits và BUS địa chỉ 16 bits. Các loại CPU này đã có khả năng quản lý dược 64K từ nhớ của bộ nhớ và 256 thiết bị ngoại vi. Điều đáng chú ý ở μP8085 là công nghệ dồn kênh và chia sẻ thời gian hợp lý trên BUS đã cho phép đưa ra thêm những tín hiệu điều khiển rất mạnh, cho phép xây dựng những máy vi tính đầu tiên.

Khoảng thời gian năm 1978 đến năm 1982 là giai đoạn ra đời và phát triển mạnh mẽ của các trung tâm Vi xử lý 16 bits. Đặc biệt ở cuối giai đoạn này là sự xuất hiện các trung tâm Vi xử lý μP8088, μP8086, với khả năng xử

lý dữ liệu 16 bits và BUS địa chỉ 20 bits, được sử dụng để tạo ra các máy vi tính XT, có ổ đĩa mềm để lưu giữ chương trình ứng dụng và dữ liệu.

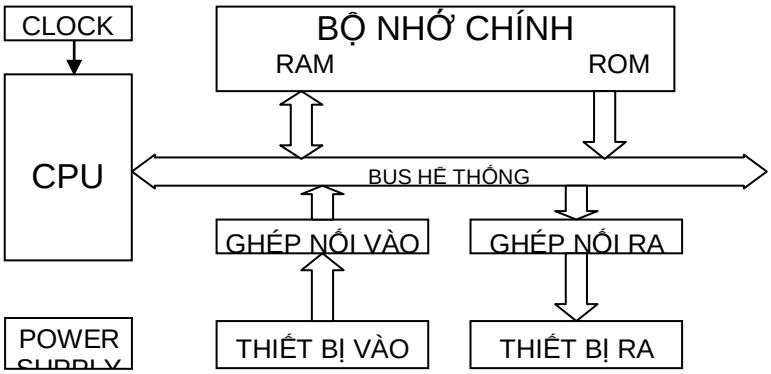
Tiếp theo của giai đoạn này là sự phát triển vũ bão của các loại µP80186, µP80286, 80386SX, 80486-SX và 80486-DX, với nhịp đồng hồ lên đến 100MHz. Máy vi tính AT và các máy tính PC ra đời trong giai đoạn này dù giá thành còn rất cao, nhưng đã trở thành rất thông dụng trong đời sống con người.

Từ khoảng giữa những năm 1993 trở lại đây, các trung tâm vi xử lý Pentium ra đời, tốc độ ngày càng cao, với nhịp đồng hồ lên đến hàng GHz, và sự xuất hiện của các trung tâm xử lý đa phân luồng như các chip Pentium IV hiện nay.

I.5.2 Cấu trúc cơ bản của hệ Vi xử lý

Các khối chức năng cơ bản của một hệ Vi xử lý (hình I.3) gồm:

- Đơn vị xử lý trung tâm (CPU)
- Bộ nhớ ROM, RAM
- Thiết bị vào (nhập dữ liệu - Input device)
- Thiết bị ra (đưa dữ liệu ra - Output device)
- Ngoài ra còn phải kể đến khối tạo xung nhịp (Clock Generator) và khối nguồn (Power Supply).



Hình I.3 Sơ đồ khối cấu trúc cơ bản hệ Vi xử lý

Các khối chức năng cơ bản được nối với nhau qua một tập đường dây truyền dẫn tín hiệu điện gọi là *BUS hệ thống*. BUS hệ thống bao gồm 3 BUS thành phần: *BUS địa chỉ*, *BUS dữ liệu* và *BUS điều khiển*. Thiết bị vào/ra thường được ghép nối với BUS hệ thống thông qua giao diện ghép nối (I/O Interface).

Đơn vị xử lý trung tâm (Central Processing Unit – CPU) là khối chức năng cơ bản nhất để tạo nên một hệ Vi xử lý hay máy tính cá nhân (*Personal Computer – PC*). Máy vi tính là một trong những ứng dụng cụ thể của một hệ thống gọi là *Hệ Vi xử lý*.

a) CPU thực hiện chức năng xử lý dữ liệu thông qua các hoạt động chính sau:
   - Đọc mã lệnh – đọc tập các bit thông tin "0" và "1" từ bộ nhớ chính
   - Giải mã lệnh – tạo các xung điều khiển tương ứng với mã lệnh để điều khiển hoạt động của các khối chức năng khác
   - Thực hiện từng bước các thao tác xử lý dữ liệu theo yêu cầu của lệnh.

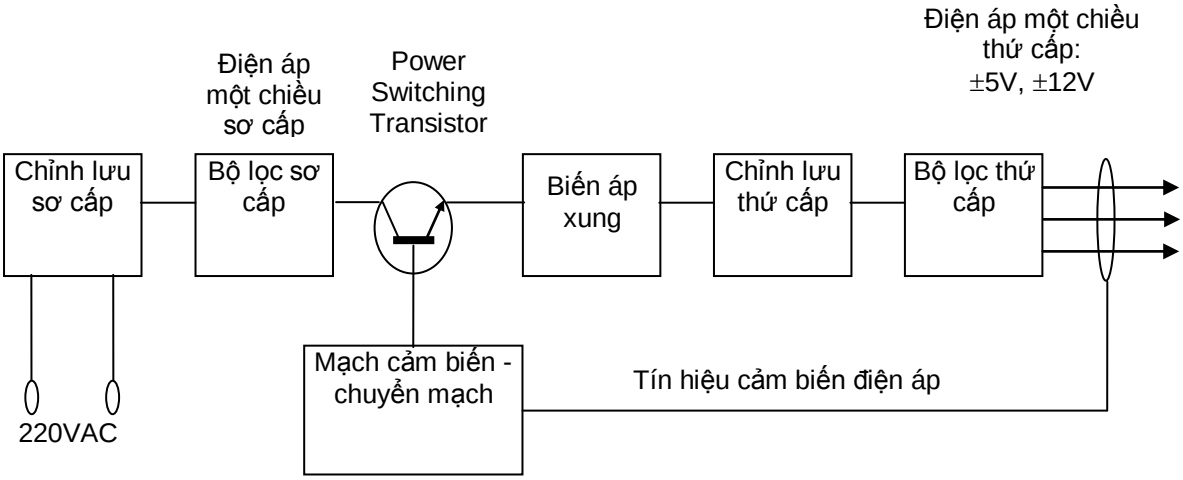
Bên trong CPU có các thanh ghi (*Registers*):
   - Thanh ghi con trỏ lệnh IP (*Instruction Pointer*), trong các trung tâm vi xử lý trước đây còn gọi là thanh đếm chương trình PC (*Program Counter*) chứa địa chỉ của lệnh kế tiếp cần được thực hiện trong tuần tự thực hiện chương trình
   - Các thanh ghi đa dụng khác GPRs (*General Purpose Registers*) để lưu trữ tạm thời dữ liệu, kết quả trung gian hay trạng thái của hệ thống cùng với đơn vị số học và logic ALU (*Arithmetic and Logic Unit*) thực hiện các thao tác xử lý dữ liệu
   - Đơn vị điều khiển CU (*Control Unit*) là thành phần phức tạp nhất, có chức năng giải mã lệnh và tạo các tín hiệu điều khiển hoạt động của toàn hệ thống.

b) Bộ nhớ chính được tổ chức từ các từ nhớ, trong IBM/PC *từ nhớ có độ dài 1 byte* (8 bits). Bộ nhớ này gồm các chip nhớ chỉ đọc ROM (*Read Only Memory*) và các chip nhớ truy xuất ngẫu nhiên RAM (*Random Access Memory*) có tốc độ truy cập nhanh. Bộ nhớ được sử dụng để chứa các chương trình và các dữ liệu điều khiển hoạt động của hệ thống. các chương trình ứng dụng và dữ liệu có thể được chứa ở ROM hoặc RAM, các kết quả trung gian hay kết quả cuối cùng của các thao tác xử lý có thể được chứa trong các thanh ghi đa dụng hoặc trong khối nhớ RAM

c) Các mạch ghép nối vào/ra là các mạch điện tử cho phép CPU trao đổi dữ liệu với các thiết bị ngoại vi như bàn phím, màn hình, máy in…làm giao diện với người dùng hoặc các bộ chuyển đổi số-tương tự DAC (*Digital/Analog Converter*), chuyển đổi tương tự-số ADC (*Analog/Digital Converter*), các mạch vào/ra số DO (*Digital Outputs*), DI (*Digital Inputs*)…

d) Hệ Vi xử lý còn có một mạch tạo xung nhịp gọi là đồng hồ hệ thống (*Clock Generator*) điều khiển và duy trì hoạt động đồng bộ của tất cả các khối chức năng. Bộ tạo xung này được điều khiển bằng một mạch thạch anh có tần số thích hợp và đảm bảo tần số làm việc ổn định cho toàn bộ hệ thống.
e) Một khối nguồn nuôi (*Power Supply*) cung cấp năng lượng cho hệ thống từ mạng điện lưới.

Bộ nguồn của các hệ Vi xử lý thông thường là bộ nguồn xung với kỹ thuật đóng-ngắt dùng bán dẫn công suất (*Switching Power Supply*), vừa gọn nhẹ, công suất lớn lại vừa đảm bảo độ gợn sóng nhỏ nhất và khả năng chống nhiễu cao. Hình I.4 là sơ đồ khối của bộ nguồn đóng-ngắt. Điện áp lưới (220VAC) được chỉnh lưu trực tiếp, lọc bằng tụ hoá để cung cấp cho một bộ dao động tần số cao (từ 20KHz đến 40KHz). Các xung điện áp tần số cao được chuyển sang biến áp xung công suất hạ áp. Điện áp ở lối ra của biến áp xung được chỉnh lưu và lọc thành điện áp nguồn một chiều cung cấp cho hệ thống. Nguyên lý ổn áp ở đây là thay đổi độ rộng của các xung có tần số ổn định, do vậy sự dao động của điện áp đầu ra khi có tải được chuyển qua bộ cảm biến để điều chỉnh độ rộng này, đảm bảo sự ổn định của điện áp ra.



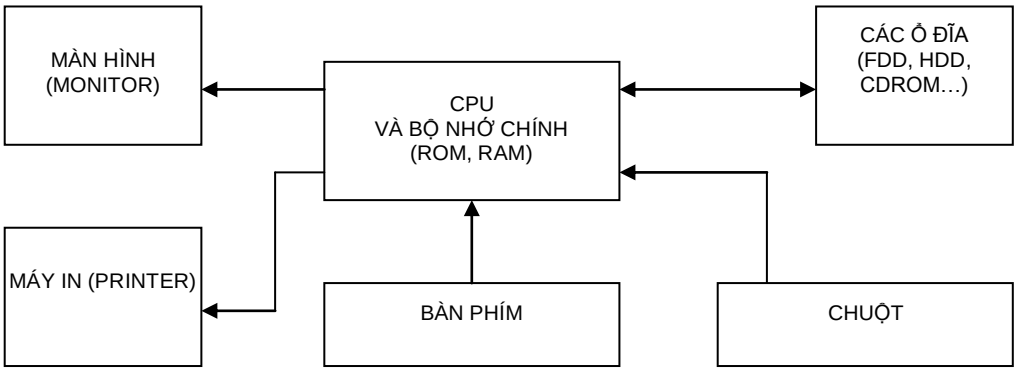
Hình I.4 Sơ đồ khối bộ nguồn nuôi máy tính

I.5.3 Từ hệ Vi xử lý đến máy vi tính PC

Trong thực tế, các hệ Vi xử lý hiện đại được trang bị thêm nhiều thiết bị ngoại vi tiện dụng tuỳ theo yêu cầu, mục đích sử dụng và có giao diện thân thiện với con người. Đó là các máy vi tính PC. Cũng có thể là những hệ Vi xử lý chuyên dụng cho những mục đích tính toán hay điều khiển.

a) ***Máy tính xử lý dữ liệu:*** Là các máy tính được dùng để tính toán, xử lý các dữ liệu như quản lý nhân viên trong cơ quan, tính toán tiền lương,

tính toán kết cấu công trình, phân tích dữ liệu trong kinh doanh, v.v… Quan điểm đúng cho rằng máy tính chỉ gồm CPU và bộ nhớ chính, còn các thiết bị phụ trợ khác như bàn phím, máy in, các ổ đĩa cứng, đĩa mềm, ổ CD, chuột, màn hình, máy in…, là những thiết bị ngoại vi. Các chương trình để xử lý dữ liệu được lưu giữ trong bộ nhớ chính hoặc trong các ổ đĩa, có nhiệm vụ xử lý những dữ liệu được người dùng nhập vào, và đưa kết quả xử lý ra màn hình, in ra giấy hoặc lưu giữ trong các ổ đĩa. Để đánh giá tính năng và chất lượng của các máy này, ta thường căn cứ vào tốc độ xử lý dữ liệu, dung lượng bộ nhớ, ổ đĩa, chất lượng màn hình, máy in v.v…



Hình I.4 Máy Vi tính PC

b) ***Máy tính là bộ xử lý số:*** Đối với các máy tính này, thời gian dành cho xử lý dữ liệu rất nhỏ, còn thời gian để tính toán, xử lý các số liệu lại vô cùng lớn. Các máy tính loại này được sử dụng chủ yếu trong các cơ quan dự báo, như dự báo khí tượng, thuỷ văn, trong tính toán quỹ đạo bay của tên lửa, máy bay, tàu thuỷ, v.v… hay trong các phòng nghiên cứu khoa học. Những máy tính loại này thông thường thực hiện những chương trình tính toán khổng lồ, nên chúng được trang bị các CPU rất mạnh và các thiết bị ngoại vi, bộ nhớ ngoài rất lớn. Đó là những siêu máy tính (*Supercomputer*).

c) ***Máy tính đo lường và điều khiển:*** Sự phát triển nhanh chóng của các hệ thống máy tính đã tạo ra những ứng dụng lớn lao trong các hệ thống đo lường và điều khiển tự động. Đối với các ứng dụng thông thường như trong các dụng cụ gia dụng, từ Tivi, điều hoà nhiệt độ, máy giặt v.v… Đó là những máy tính nhỏ được chế tạo dưới dạng một vi mạch (*Single-Chip Microcomputer*). Tuy nhiên, cũng cần phải tính đến những máy tính này trong các thiết bị hiện đại và phức tạp như trong các hệ thống tự động lái máy bay (*Autopilot*), tàu thuỷ, tên lửa…

d) ***Căn cứ vào tính năng kỹ thuật và các chỉ tiêu về kích thước:*** Các máy tính còn được chia ra thành *máy tính lớn* đẻ giải các bài toán cực lớn với tốc độ rất nhanh, *máy tính nhỏ* sử dụng trong gia đình, trong trường học hay các tính toán thông dụng, điều khiển các quá trình công nghệ vừa và nhỏ.

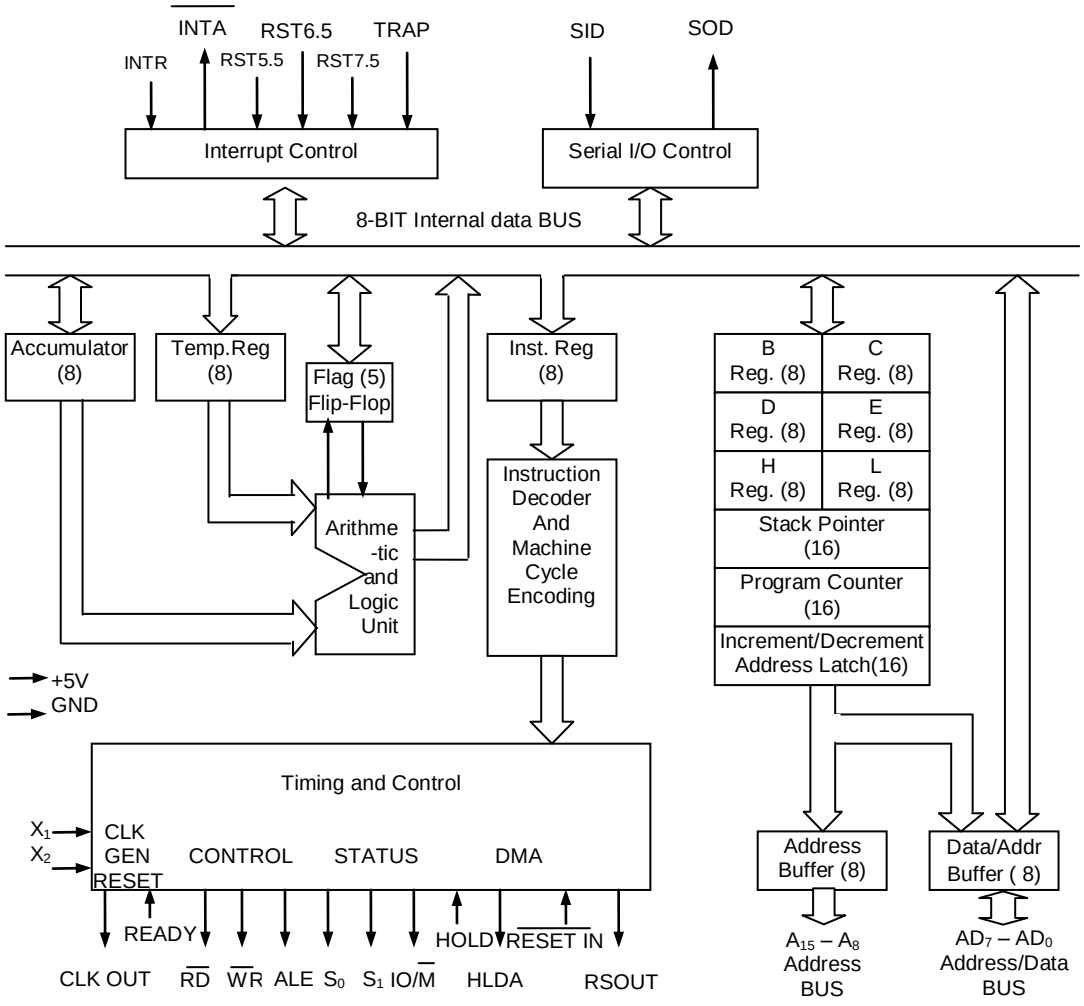
Cũng cần nhắc đến ở đây một sự khác biệt giữa hai khái niệm hệ Vi xử lý và máy vi tính: các máy vi tính luôn luôn được trang bị một phần mềm cơ bản là Hệ điều hành, ví dụ: MS-DOS hay các phiên bản điều hành đa nhiệm (MS-WINDOWS của hãng phần mềm Microsoft, hoặc các hệ điều hành của các hãng khác...) và các chương trình hay phần mềm ứng dụng, trong khi các hệ Vi xử lý chỉ cần trang bị một chương trình Monitor (chương trình giám sát) đơn giản được ghi trong bộ nhớ ROM.

CHƯƠNG II. CÁC ĐƠN VỊ VI XỬ LÝ TRUNG TÂM (CPU – Central Processing Unit)

Vì hầu hết các máy vi tính đang được sử dụng ở Việt nam đều được xây dựng trên cơ sở của các chip xử lý của hãng Intel®, nên tài liệu này cũng giới hạn sự trình bày trong khuôn khổ các trung tâm vi xử lý của hãng này. Các độc giả có thể tìm hiểu thêm về các trung tâm vi xử lý của các hãng khác như Motorola, AMD,…ở một số tài liệu tham khảo liệt kê ở phần cuối giáo trình.

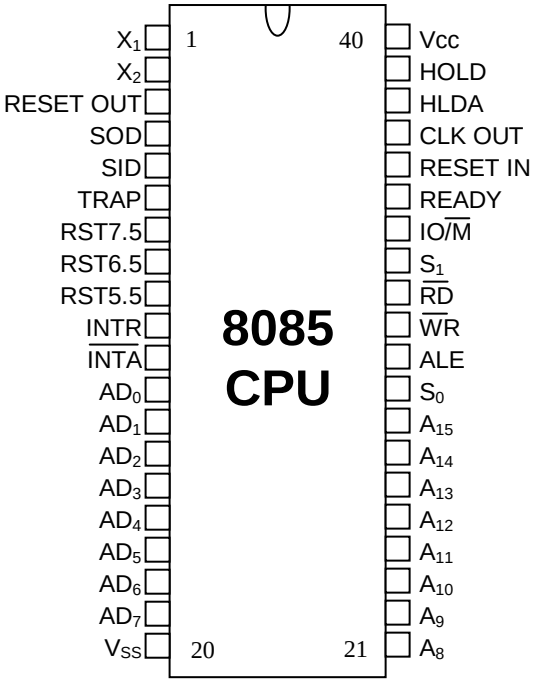
Khi CPU được chế tạo từ một mạch vi điện tử có độ tích hợp rất cao thì nó được gọi là bộ Vi xử lý (*μP – Microprocessor*). Trong quá trình phát triển, hãng Intel® đã cho ra đời nhiều thế hệ μP từ đơn giản đến phức tạp, từ thông dụng đến chuyên dụng. Tính phát huy và kế thừa luôn được coi trọng trong quá trình này, vì vậy, các chương trình ứng dụng chuẩn phần lớn có thể thực hiện được trên bất kỳ máy vi tính được xây dựng từ thế hệ μP nào.

II.1 Trung tâm Vi xử lý μP8085



Hình II.1a) Sơ đồ khối cấu trúc μP8085

Hình II.1a) là sơ đồ khối cấu trúc của µP8085, hình II.1b) là sơ đồ nối chân của µP8085. Khác với các loại µP xuất hiện trước đó như µP8008 hay µP8080, µP8085 có những bước phát triển có tính đột phá như sau:



Hình II.1b) Sơ đồ nối chân của µP8085

1. Cơ cấu ngắt theo nhiều mức khác nhau được hình thành qua một khối điều khiển ngắt, tạo ra một vector ngắt tránh được sự chèn nhau do lệnh RET N trên BUS dữ liệu. Tín hiệu nhận biết yêu cầu ngắt $\overline{INTA}$ được tạo bởi khối điều khiển ngắt, chứ không phải từ mạch phụ 8228 như ở µP8080.
2. Các tín hiệu điều khiển ghi/đọc $\overline{WR}$ và $\overline{RD}$ được tạo ra từ khối định thời và điều khiển chức năng. Các tín hiệu $\overline{INTA}$, $\overline{WR}$ và $\overline{RD}$ được tạo ngay trong CPU, chứ không do mạch phụ trợ bên ngoài.
3. µP8085 có mạch tạo xung đồng hồ được tích hợp ngay trong CPU.
4. Khối chức năng điều khiển vào/ra nối tiếp được tích hợp cũng cho phép µP8085 thực hiện các lệnh vào/ra dữ liệu nối tiếp mà nhiều khi không cần đến sự hỗ trợ của vi mạch chuyên dụng.
5. Đặc biệt hơn, µP8085 có hai thanh ghi đệm địa chỉ, đó là thanh ghi đệm $A_{15} - A_8$ và thanh ghi đệm $AD_7 - AD_0$ cho cả dữ liệu và địa chỉ. Việc dồn kênh như trên tạo điều kiện cho những chân chức năng khác được tạo thêm, làm tăng thêm sức mạnh cho CPU.

II.1.1 Các nhóm tín hiệu trong μP8085

$A_8 – A_{15}$. Nhóm tín hiệu ra: 8 bit cao của địa chỉ, các chân này là các chân được nối với bên ngoài qua mạch 3 trạng thái. Các phần tử 3 trạng thái sẽ được đặt ở trạng thái high-Z trong các trường hợp một trong các tín hiệu HOLD hay HALT là tích cực.

$AD_0 – AD_7$. Nhóm tín hiệu dồn kênh 3 trạng thái. Ở giai đoạn đầu của chu kỳ máy, T_1 của M_1, sẽ là byte thấp của 16 bit địa chỉ.

ALE (Address Latch Enable). Tín hiệu ra qua mạch 3 trạng thái. Được sử dụng để chốt byte thấp của tín hiệu địa chỉ ($A_0 – A_7$). Tín hiệu này được tạo ra trong giai đoạn đầu tiên của chu kỳ máy, T_1 của M_1, và cũng được dùng để chốt các tín hiệu trạng thái S_0 và S_1 khi cần thiết.

S_0, S_1 (Data BUS Status). Là các tín hiệu chỉ trạng thái của các chân thuộc BUS dữ liệu trong mỗi chu kỳ máy. Tổ hợp của hai tín hiệu này cũng cho biết trạng thái của CPU

S_1	S_0	*Hoạt động của BUS dữ liệu*
0	0	Trạng thái HALT
0	1	CPU đang thực hiện thao tác WRITE
1	0	CPU đang thực hiện thao tác READ
1	1	CPU đang thực hiện thao tác nhận lệnh Instruction Fetch

$\overline{RD}$ (Read). Chân ra 3 trạng thái. Nằm trong nhóm tín hiệu điều khiển. Tín hiệu tích cực khi CPU tiến hành đọc dữ liệu từ bộ nhớ hoặc từ thiết bị ngoại vi. Trong chế độ HALT hoặc DMA, chân ra này ở trạng thái high-Z.

$\overline{WR}$ (Write). Chân ra 3 trạng thái. Nằm trong nhóm tín hiệu điều khiển. Tín hiệu tích cực khi CPU tiến hành ghi dữ liệu vào bộ nhớ hoặc đưa dữ liệu ra thiết bị ngoại vi. Trong các chế độ HALT hoặc DMA, chân ra này ở trạng thái high-Z.

IO/$\overline{M}$. Trạng thái logic của đầu ra này cho biết CPU đang làm việc với thiết bị ngoại vi hay với bộ nhớ. Nếu là logic "1", CPU đang truy cập thiết bị vào/ra, còn nếu là "0", CPU đang truy cập bộ nhớ. Kết hợp với hai đầu ra $\overline{RD}$ và $\overline{WR}$ để tạo ra các tín hiệu $\overline{I/OR}$, $\overline{I/OW}$, $\overline{MEMR}$, và $\overline{MEMW}$ trong trường hợp sử dụng địa chỉ tách biệt đối với thiết bị vào/ra. Nằm trong nhóm tín hiệu điều khiển, nên IO/$\overline{M}$ cũng là đầu ra 3 trạng thái.

Interrupts. μP8085 có ngắt đa mức. Có 5 chân ngắt tất cả: (INTR, RST5.5, RST6.5, RST7.5 và TRAP). Ngoài chân ngắt không che được là TRAP, các chân khác đều có thể che hoặc không che nhờ lập trình phần mềm.

- ***INTR:*** Chân nhận yêu cầu ngắt từ bên ngoài, được đáp ứng theo nguyên tắc polling hoặc vectoring thông qua lệnh RST

- ***Các yêu cầu ngắt RST:*** Có 3 đầu vào yêu cầu ngắt với các mức ưu tiên khác nhau là RST7.5, RST6.5 và RST5.5. Khi yêu cầu ngắt xuất hiện tại các chân này, CPU tự động chuyển đến các vector ngắt tương ứng. Cụ thể như sau:
  - ***RST5.5*** là mức ưu tiên thấp nhất, phản ứng theo mức điện áp trên chân yêu cầu ngắt, địa chỉ vector ngắt này nằm ở ô nhớ có địa chỉ $2C_H$.
  - ***RST6.5:*** Ngắt ưu tiên thấp thứ 2, phản ứng theo mức điện áp trên chân yêu cầu ngắt, địa chỉ vector ngắt này nằm ở ô nhớ 34_H
  - ***RST7.5****:* Mức ưu tiên cao nhất. Phản ứng theo sườn lên của xung yêu cầu ngắt. Sườn lên của xung này tác động lên một flip-flop, mạch này giữ lại yêu cầu ngắt cho đến khi được xoá nhờ tín hiệu nhận biết yêu cầu ngắt Acknowledge. Địa chỉ của vector ngắt này nằm ở ô nhớ $3C_H$
- ***TRAP:*** Là chân nhận yêu cầu ngắt không che được (dĩ nhiên là nó có mức ưu tiên cao nhất). Địa chỉ của vector ngắt này ở ô nhớ 24_H.

$\overline{\text{INTA}}$. Tín hiệu ra nhận biết yêu cầu ngắt tại chân INTR. Các yêu cầu ngắt RST5.5, RST6.5, RST7.5 và TRAP không tác động đến $\overline{\text{INTA}}$.

HOLD. Trạng thái logic "1" ở chân này là yêu cầu của thao tác DMA. Các đầu ra $\overline{\text{RD}}$, $\overline{\text{WR}}$, IO/$\overline{\text{M}}$ và ALE sẽ được đưa về trạng thái high-Z.

HLDA. Tín hiệu nhận biết yêu cầu HOLD.

$\overline{\text{RESET IN}}$. Logic thấp "0" ở đầu vào của chân này yêu cầu tái khởi động hệ Vi xử lý. Do tác động của tín hiệu RESET IN tích cực, giá trị của thanh đếm chương trình PC sẽ được nạp lại là 0000_H. Các mặt nạ ngắt và tín hiệu HLDA cũng được tái thiết lập về giá trị mặc định.

RESET OUT. Đầu ra nhận biết hệ Vi xử lý được tái khởi động. Dùng tín hiệu này để tái khởi động toàn bộ hệ thống.

READY. Logic "1" ở đầu vào này thông báo trạng thái sẵn sàng cung cấp dữ liệu cho CPU hoặc nhận dữ liệu từ CPU của các thiết bị ngoại vi.

SID (Serial Input Data). Là cổng vào của dữ liệu nối tiếp của hệ Vi xử lý. Bit hiện diện tại cổng này được đọc vào CPU nhờ lệnh RIM, bit sẽ được đưa vào bit cao của Acc (MSB).

SOD (Serial Output Data). Bit cao (MSB) của Acc được truyền ra ngoài chân này khi sử dụng lệnh SIM.

X_1, X_2. Lối nối thạch anh hoặc một mạch dao động để tạo xung nhịp cho CPU. Có thể sử dụng thạch anh có tần số dao động trong khoảng từ 0.5 đến 3MHz.

CLK. Đầu ra của xung nhịp, có thể làm xung nhịp cho các thành phần chức năng khác trong hệ Vi xử lý.

Vcc, Vss. Lối nối nguồn +5V và GND cho μP8085. Cũng cần nhắc lại rằng, μP8085 chỉ cần một nguồn nuôi duy nhất là +5V, khả năng cung cấp dòng của nguồn cần được thiết kế tuỳ theo nhu cầu của toàn hệ Vi xử lý.

II.1.2 Khái niệm và bản chất vật lý của các BUS trong hệ Vi xử lý

Hoạt động của một hệ Vi xử lý thực chất là việc trao đổi và xử lý các giá trị nhị phân giữa các thành phần, các khối và các mạch vi điện tử trong toàn bộ hệ thống. Như đã biết, các giá trị nhị phân (hoặc "0" hoặc "1") được thể hiện qua mức điện áp so với một chuẩn nhất định. Giá trị "0" tương ứng với mức điện áp thấp (từ 0V đến +0,8V) và giá trị "1" tương ứng với mức điện áp từ khoảng +3V đến +5V. Để biểu diễn một số liệu nhị phân, các phần tử mang thông tin được liên kết kề nhau theo nhóm (ví dụ 1byte là 8 bits). Để đảm nhận công việc di chuyển các dữ liệu này trong toàn bộ hệ thống, có các đường dây truyền dẫn điện chuyên dụng được ghép song song thành hệ thống, mỗi dây truyền dẫn dành riêng cho 1 bit. Tập các đường truyền dẫn dành riêng cho các tín hiệu có cùng chức danh (dữ liệu, địa chỉ, điều khiển) được gọi là BUS. Như vậy, trong một hệ Vi xử lý, có ba loại BUS: BUS dữ liệu, BUS địa chỉ và BUS điều khiển. Các BUS này hợp lại thành BUS hệ thống.

Từ khái niệm trên, dễ dàng suy ra bản chất vật lý của các BUS trong một hệ Vi xử lý: Đó là các *đường truyền dẫn điện, có thể dưới các dạng cáp nhiều sợi, đường dẫn trong các bảng mạch in v.v…* Khả năng và chất lượng dẫn điện của các đường truyền dẫn này đóng vai trò quan trọng và quyết định đối với hoạt động của một hệ Vi xử lý. Đường truyền dẫn kém, trở kháng cao có thể gây ra sự suy giảm của tín hiệu điện dẫn đến các hiện tượng mất mát hoặc sai dữ liệu, rất nguy hiểm.

BUS là đường dẫn điện nội bộ mà theo đó các tín hiệu được truyền từ bộ phận này đến các bộ phận khác trong hệ Vi xử lý. Có 3 loại BUS trong một hệ Vi xử lý cũng như trong máy tính PC:

- BUS dữ liệu truyền dữ liệu theo hai chiều giữa bộ nhớ và trung tâm Vi xử lý, giữa các thiết bị ngoại vi và Trung tâm Vi xử lý
- BUS địa chỉ xác định các vị trí nhớ trong bộ nhớ, các thiết bị ngoại vi

- BUS điều khiển truyền các tín hiệu điều khiển đến các bộ phận cần được điều khiển

Các BUS được xây dựng bằng cách sử dụng các khe cắm, theo một quy ước chặt chẽ đối với từng tiếp điểm. Đối với các khe cắm, các tiếp điểm tương ứng sẽ được nối với nhau bằng các dây dẫn hoặc đường dẫn song song trên mạch in. Nhờ vậy, khi dữ liệu được truyền đi, tất cả các bit (8, 16, 32, hay 64) đều được truyền đi đồng thời, cùng một hướng (truyền dẫn song song).

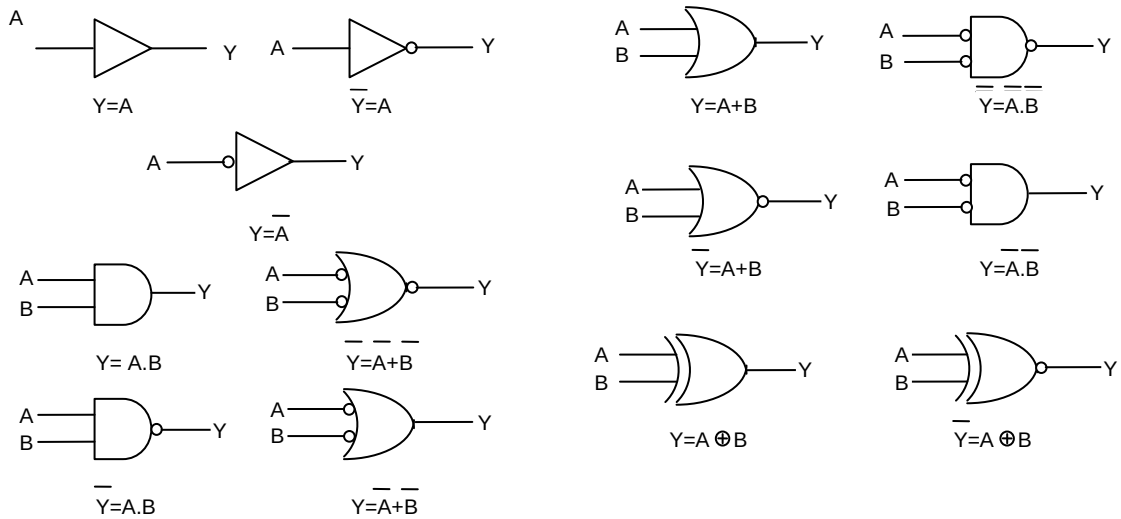
Cũng cần nói thêm rằng, trong máy PC, có 3 loại cấu trúc BUS thường gặp là ISA (Industrial Standard Architecture) EISA (Enhanced ISA) và PCI (Peripheral Component Interconnect).

II.1.3 Các mạch 3 trạng thái, mạch chốt và mạch khuyếch đại BUS 2 chiều

Trước tiên, cũng cần nhắc lại một số linh kiện điện tử số cơ bản sử dụng trong máy vi tính. Nhờ công nghệ cao, các linh kiện có độ tích hợp lớn và rất lớn đã ra đời, nhưng không thể không nhìn lại một số mạch tổ hợp thực hiện những hàm logic cơ bản nhất.

a) Các cổng logic

Ký hiệu các mạch được chỉ ra trên Hình II.2, cùng biểu thức hàm logic gồm: *mạch đệm (bufer), mạch đảo (NOT), mạch và (AND), mạch NAND, mạch hoặc (OR), mạch NOR và mạch XOR.*



Hình II.2 Một số cổng Logic thông dụng

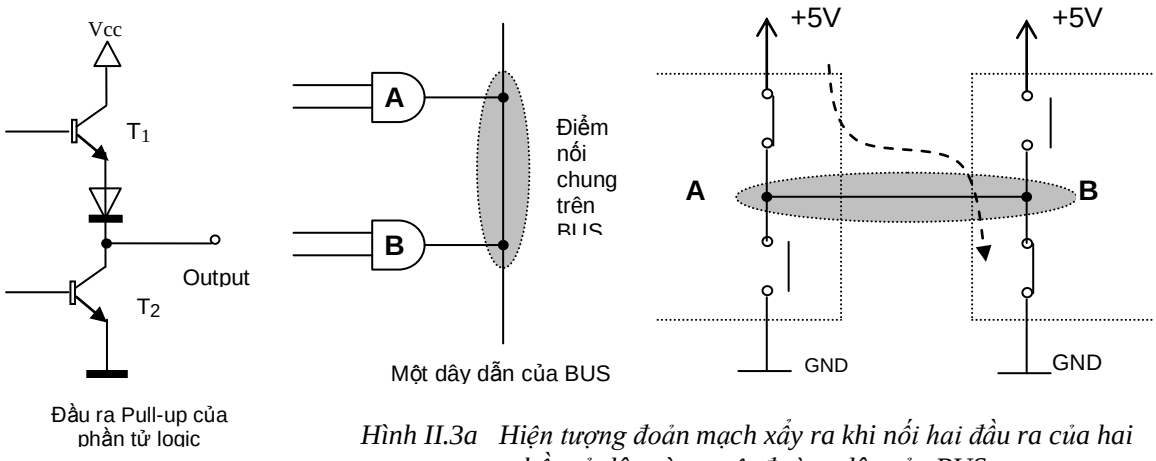
Các loại mạch này thường được sử dụng để tạo nên các mạch tổ hợp logic thực hiện các chức năng lập mã, giải mã, dồn kênh và phân kênh. Cũng

cần lưu ý rằng, một số mạch chức năng như giải mãm dồn kênh và phân kênh đã được các hãng tích hợp dưới dạng các mạch MSI. Một số mạch có thể kể ra như mạch giải mã 3/8 SN74138, mạch dồn kênh 74151, mạch cộng, và mạch nhân v.v…

b) Mạch 3 trạng thái (Tristate Component)

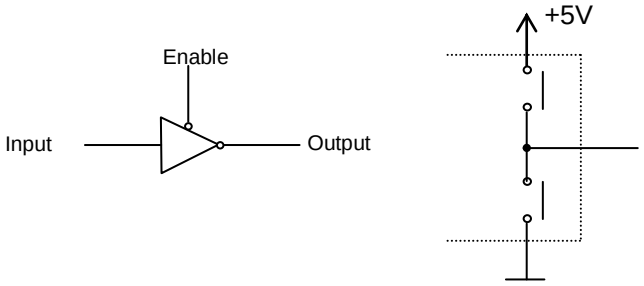
Trong hệ Vi xử lý, có nhiều khối chức năng cần thông tin, nhưng tại một thời điểm, bao giờ cũng chỉ có một khối đưa tín hiệu ra (dữ liệu) và một số hạn chế các khối thu nhận tín hiệu. Thay vì nối dây dẫn liên kết các khối qua từng đôi phần tử một, các tín hiệu này được đưa lên BUS. Với các cổng logic thông thường, không thể nối trực tiếp chúng lên cùng một đường dây vì sẽ xẩy ra tranh chấp BUS vì đoản mạch. Ví dụ đầu ra của phần tử A là "1" trong lúc đầu ra của phần tử B là "0". (Hình II.3). Các đầu ra của loại mạch này đều theo cấu trúc *pull-up*, nghĩa là có hai transistor được nối nối tiếp với nhau (xem hình vẽ), emitter của transistor này qua một diode rồi đến đầu ra, đến collector của transistor kia. Với hai trạng thái logic "1" và "0", tương ứng sẽ lả T_1 mở, T_2 đóng và ngược lại, T_2 mở và T_1 đóng. Trên hình vẽ II.2 hiện tượng nguy hiểm xẩy ra khi lối ra của phần tử logic A là "1", các khoá mở hay đóng tương đương việc transistor thông bão hoà hay ngắt, lối ra của phần tử logic B là "0" và hiện tượng đoản mạch xẩy ra.

Để tránh hiện tượng này, một loại cổng logic gọi là cổng 3 trạng thái (tri-state gate) được sử dụng cho lối ra của các khối nối chung vào BUS. Hình II.3a là một phần tử đảo đầu ra 3 trạng thái. Hình II.3b là sơ đồ tương đương của trạng thái high-Z, tương ứng với trường hợp đầu ra bị tách khỏi BUS.



Hình II.3a Hiện tượng đoản mạch xẩy ra khi nối hai đầu ra của hai phần tử lên cùng một đường dây của BUS

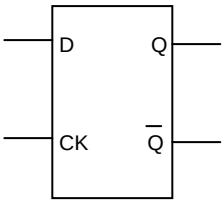
Như vậy, để tránh xung đột trên BUS, các phần tử có dầu ra nối với BUS cần phải đưa qua cổng 3 trạng thái.



Hình II.3b Phần tử đảo 3 trạng thái và sơ đồ tương đương đầu ra của phần tử ở trạng thái high-Z

c) Mạch chốt, thanh ghi:

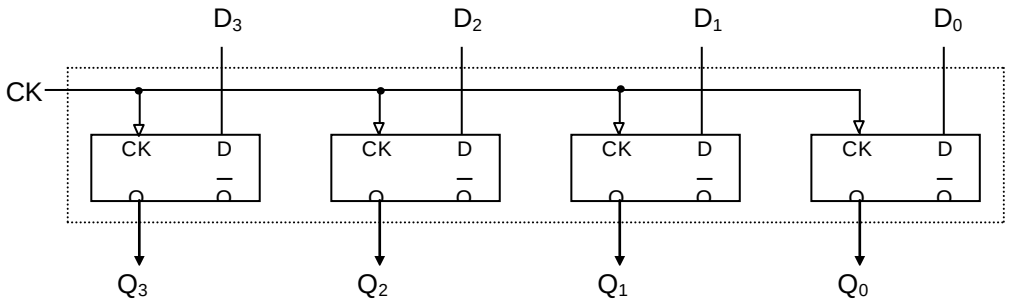
Mạch chốt là một mạch gồm các phần tử có khả năng lưu giữ các giá trị "0" hoặc "1" ở lối ra. Có thể dùng D flip-flop làm một mạch chốt với tín hiệu để chốt dữ liệu tại đầu ra Q theo bảng giá trị chân lý sau:



D	CK	Q	$\bar{Q}$
X	0	Q	Q_n
0	1	0	1
1	1	1	0

Hình II.4 Mạch chốt (hay phần tử nhớ) D Flip-Flop

Biết rằng Q^{n+1} = D với tín hiệu điều khiển là sự xuất hiện sườn dương của xung nhịp CK. Như vậy, giá trị logic (0 hoặc 1) tại D đã được chuyển sang đầu ra Q (chốt). Nếu CK giữ nguyên trạng thái bằng "1", thì trạng thái đầu ra Q được giữ nguyên. Như vậy, giá trị logic của D đã được lưu giữ ở Q (nhớ).



Hình II.5 Thanh ghi 4 bits

Thanh ghi (*Register*) là tập hợp các flip-flop được nối song song với nhau, có thể lưu giữ được các số liệu nhị phân. Hình II.5 là sơ đồ một thanh ghi lưu giữ số liệu nhị phân 4 bits được tạo từ 4 phần tử D flip-flop.

Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866

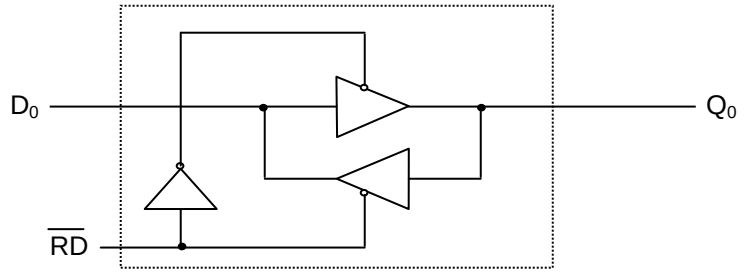
Một số liệu nhị phân bất kỳ từ D_3 đến D_0 sẽ được chốt sang các lối ra từ Q_3 đến Q_0 mỗi khi có một sườn lên của xung nhịp CK được đưa tới lối vào xung nhịp. Từ nhị phân này được lưu giữ ở lối ra cho đến khi có dữ liệu mới được đưa vào lối D và có xuất hiện sườn lên của xung nhịp CK.

d) Mạch khuyếch đại BUS 2 chiều

Trên cơ sở của các mạch 3 trạng thái, các mạch khuyếch đại BUS hai chiều được xây dựng theo nguyên lý sau:

Hai phần tử 3 trạng thái sẽ được ghép ngược với nhau (Hình II.6), chân điều khiển sẽ dùng tín hiệu đảo của tín hiệu đọc $\overline{RD}$. Khi xuất hiện tín hiệu $\overline{RD}$, dữ liệu được phép đi từ Q_0 sang D_0, ngược lại, tín hiệu chỉ được phép đi từ D_0 sang Q_0 và cho phép CPU đưa tín hiệu ghi dữ liệu ra ngoài.

Ghép nối đủ số phần tử cho tất cả các dây dữ liệu, ta có mạch khuyếch đại BUS hai chiều. Trong thực tế, mạch có chức năng trên đã được tích hợp theo chuẩn của TTL, được ký hiệu là 8228 hoặc 8288 (Octal BUS Transceiver).

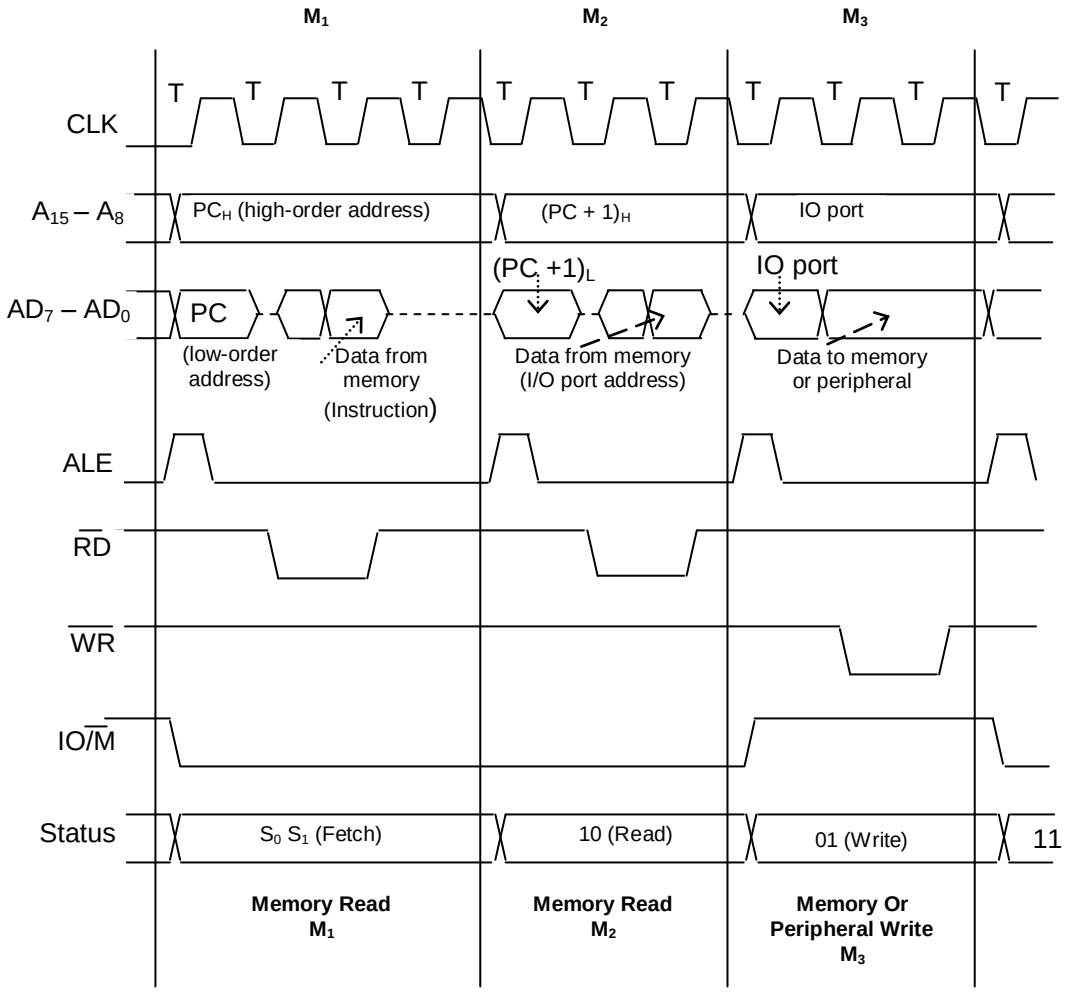


Hình II.6 Phần tử khuyếch đại BUS hai chiều

II.1.4 Biểu đồ Timing thực hiện lệnh của CPU μP8085

Việc thực hiện một lệnh trong μP8085 thực tế là một chuỗi các thao tác READ và WRITE. Mỗi thao tác READ hay WRITE tương ứng với một chu kỳ máy M). Mỗi lệnh được thực hiện qua 1 đến 5 chu kỳ máy. Mỗi chu kỳ máy cần từ 3 đến 5 nhịp đồng hồ (hay còn gọi là trạng thái T)

Ở chu kỳ máy thứ nhất, CPU thực hiện việc nhận mã lệnh (Instruction Code Fetch), Còn gọi là chu kỳ Opcode Fetch. Theo biểu đồ thời gian trên hình II.8, thấy rằng việc thực hiện chu kỳ máy M_1 (chu kỳ nhận lệnh Opcode Fetch), CPU gửi ra các tín hiệu IO/$\overline{M}$, S_1 và S_0 (tương ứng 0, 1, 1 trên biểu đồ thời gian) xác định thao tác của chu kỳ.

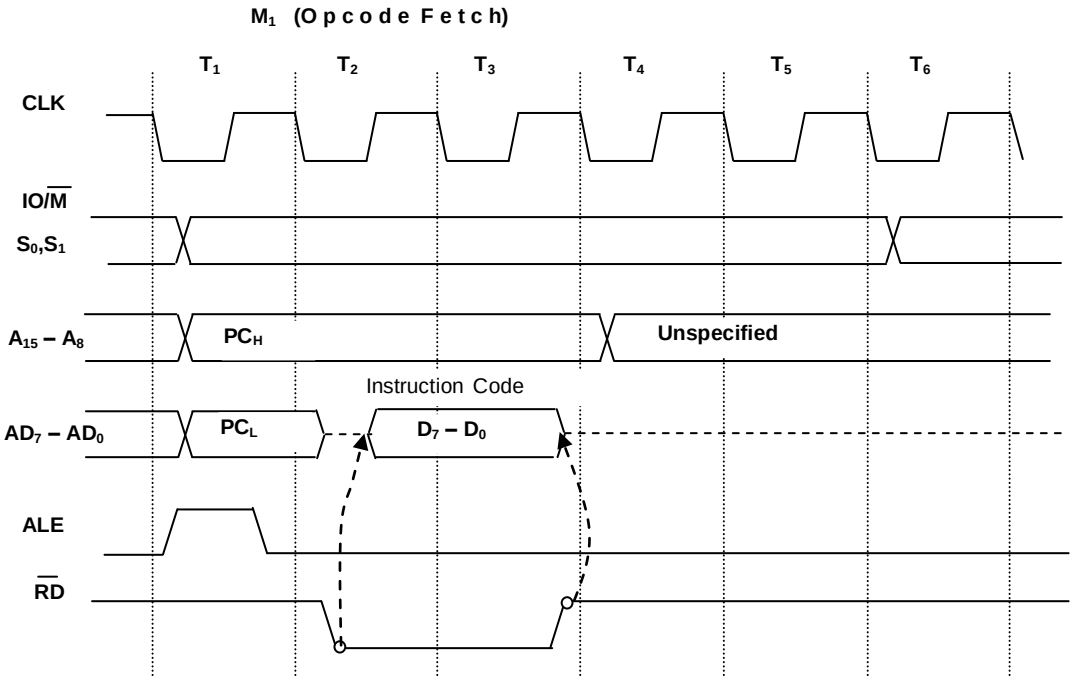


Hình II.7 Định thời cơ sở của µP8085 (Theo tài liệu của hãng Intel)

CPU cũng đồng thời gửi 16 bit địa chỉ ra ở chu kỳ máy đầu tiên, ngay từ nhịp đầu tiên (T_1) để xác định ô nhớ hay thiết bị I/O. Nội dung PC_L chỉ tồn tại trong thời gian 1 nhịp nên cần phải được chốt lại nhờ tín hiệu ALE ở mức cao.

Khi $D_7 – D_0$ đã ổn định trên các dây dữ liệu, CPU gửi tín hiệu $\overline{RD}$. Khi đã nhận được dữ liệu, $\overline{RD}$ chuyển lên mức cao để cấm vị trí ô nhớ hay thiết bị I/O.

Số lượng chu kỳ máy và trạng thái cần cho thực hiện một lệnh là cố định, song số lượng này khác nhau đối với các lệnh khác nhau, tuỳ theo độ dài của từ lệnh (1 byte, 2 bytes, 3 bytes). Số lượng chu kỳ máy phụ thuộc vào số lần CPU phải liên lạc với các phần tử khác trong hệ thống, chủ yếu là với các chip khác.



Hình II.8 Biểu đồ thời gian của các tín hiệu trong chu kỳ máy nhận lệnh (Opcode Fetch)

II.1.5 Khái niệm chu kỳ BUS

Khoảng thời gian (tính theo số lượng chu kỳ xung nhịp) để CPU (hoặc đơn vị làm chủ BUS) thực hiện hoàn thiện một thao tác di chuyển dữ liệu từ CPU đến bộ nhớ, đến thiết bị ngoại vi hoặc theo chiều ngược lại được gọi là *chu kỳ BUS.*

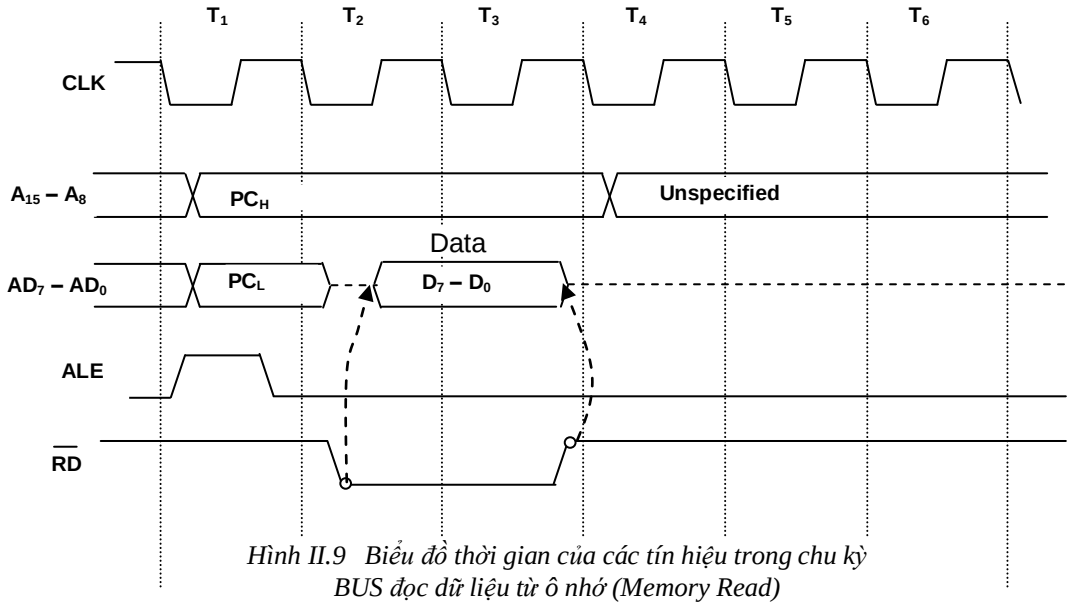
Một chu kỳ BUS được CPU hoặc đơn vị làm chủ BUS thực hiện trong hai giai đoạn:

- Giai đoạn một: CPU gửi địa chỉ vị trí cần truy xuất (ô nhớ hoặc thiết bị ngoại vi) lên BUS địa chỉ, khoảng thời gian này được gọi là *thời gian địa chỉ* (address time). Địa chỉ đích (destination - địa chỉ của một ô nhớ hay địa chỉ thanh ghi dữ liệu của thiết bị ngoại vi cần truy xuất) được CPU (hoặc đơn vị làm chủ BUS) gửi lên BUS địa chỉ cùng các tín hiệu xác định loại chu kỳ BUS
- Giai đoạn hai: CPU kiểm tra tín hiệu sẵn sàng (READY) của đơn vị cần truy xuất (bộ nhớ hoặc thiết bị ngoại vi) để thực hiện việc di chuyển và chốt dữ liệu. Khoảng thời gian này được gọi là *thời gian dữ liệu.*

Tồn tại 4 loại chu kỳ BUS cơ bản:

a. Chu kỳ BUS đọc dữ liệu từ bộ nhớ (Memory Read)
b. Chu kỳ BUS ghi dữ liệu vào bộ nhớ (Memory Write)

c. Chu kỳ BUS đọc dữ liệu từ thiết bị ngoại vi (I/O Read)
d. Chu kỳ BUS ghi liệu vào thiết bị ngoại vi (I/O Write)



Hình II.9 Biểu đồ thời gian của các tín hiệu trong chu kỳ BUS đọc dữ liệu từ ô nhớ (Memory Read)

Ngoài ra, do sự khác nhau về vận tốc, khả năng xử lý và chuẩn bị, hoàn thiện dữ liệu, tín hiệu READY chưa ở mức tích cực, các thao tác di chuyển dữ liệu của CPU phải tạo thêm các trạng thái đợi (Wait State), do vậy các loại chu kỳ BUS có độ dài khác nhau.

II.1.6 Ngắt (Interrupt)

Trong thực tế, tốc độ xử lý dữ liệu của CPU cao hơn rất nhiều so với "sự chế biến dữ liệu" của các thiết bị I/O. Vì vậy cần tạo ra một cơ chế vào/ra hợp lý để tăng hiệu suất làm việc của CPU. Ngắt trong hệ thống Vi xử lý nhằm mục đích giải quyết sự bất hợp lý do CPU phải chờ đợi thiết bị ngoại vi. Thiết bị ngoại vi chỉ yêu cầu CPU phục vụ việc nhận hay chuyển giao dữ liệu khi bản thân nó đã sẵn sàng. Để thực hiện tốt yêu cầu này, cơ chế phục vụ ngắt là hợp lý nhất.

Ngắt nghĩa là yêu cầu CPU tạm thời dừng công việc hiện tại để trao đổi hay xử lý dữ liệu không thuộc tuần tự của chương trình đang được thực hiện. Ngắt là một hiện tượng xuất hiện ngẫu nhiên về phương diện thời điểm nhưng được dự đoán trước.

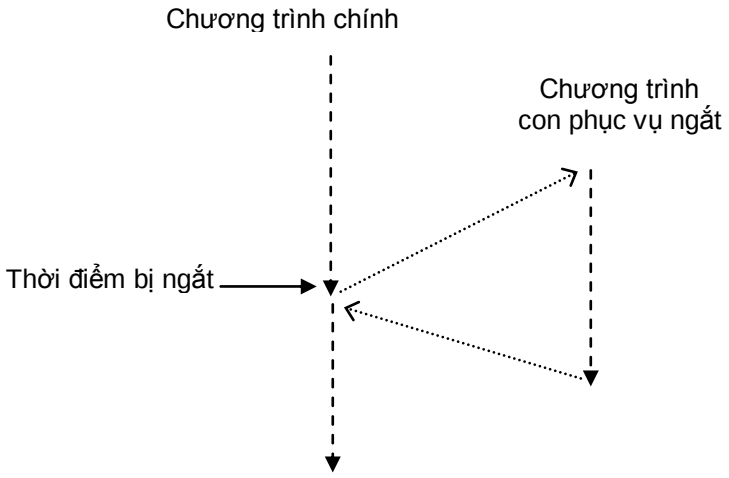
> Ngắt là hiện tượng một tín hiệu xuất hiện báo với CPU rằng có một sự kiện đã xẩy ra yêu cầu CPU phải xử lý. Quá trình xử lý của CPU sẽ bị tạm thời dừng lại để thực hiện một thao tác khác phục vụ sự kiện có yêu cầu. Khi thao tác này kết thúc, quá trình xử lý vừa bị tạm dừng sẽ được tiếp tục. Bản

thân sự kiện thông thường là yêu cầu phục vụ của thiết bị ngoại vi đối với CPU.

Trong thực tế, ngắt được sử dụng chủ yếu khi các thiết bị ngoại vi (thường rất chậm so với tốc độ xử lý của CPU) cần trao đổi thông tin với CPU.

Khi cần trao đổi thông tin, thiết bị ngoại vi gửi tín hiệu *yêu cầu ngắt* (Interrupt Request) tới CPU. CPU sẽ thực hiện nốt lệnh hiện tại và trả lời bằng tín hiệu *nhận biết yêu cầu ngắt* ($\overline{\text{INTA}}$). Chương trình chính lúc này bị tạm dừng (ngắt) và CPU chuyển sang thực hiện *chương trình con phục vụ ngắt*, túc là chương trình con trao đổi thông tin với thiết bị ngoại vi yêu cầu ngắt. Sau khi xong công việc phục vụ ngắt, CPU quay về thực hiện tiếp chương trình chính kể từ lệnh tiếp theo sau khi bị ngắt

Các tín hiệu yêu cầu phục vụ ngắt từ một thiết bị ngoại vi bất kỳ được gửi tới chân nhận yêu cầu ngắt của CPU có thể thông qua một khối điều khiển ngắt. Tuỳ theo người lập trình mà yêu cầu ngắt đó có được chuyển tới CPU hay không. Trong trường hợp yêu cầu ngắt được gửi tới CPU, xử lý của CPU gồm các bước sau:

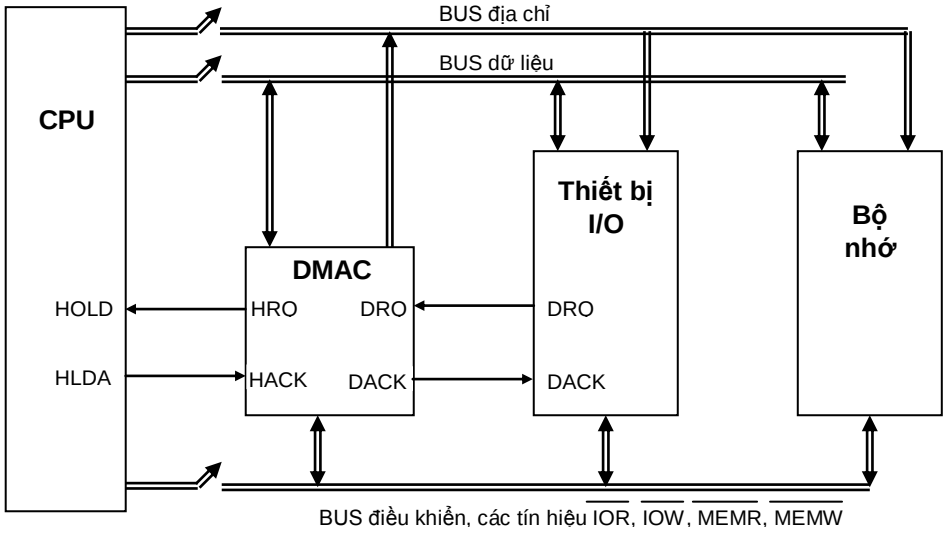


Hình II.9 Quá trình phục vụ ngắt

1. Thực hiện nốt lệnh đang được xử lý
2. Phát tín hiệu nhận biết yêu cầu ngắt gửi cho thiết bị yêu cầu phục vụ ngắt qua chân $\overline{\text{INTA}}$
3. Cất các cờ trạng thái hiện tại vào ngăn xếp
4. Xoá các cờ IF (*Interrupt Flag*) và cờ TF (*Trap Flag*)
5. Cất địa chỉ lệnh tiếp theo trong tuần tự chương trình đang thực hiện vào ngăn xếp
6. Lấy địa chỉ của chương trình con phục vụ ngắt trong bảng vector ngắt
7. Thực hiện chương trình con phục vụ ngắt.

II.1.7 Truy nhập trực tiếp bộ nhớ (Direct Memory Access – DMA)

Trong nhiều trường hợp, xẩy ra hiện tượng phải chuyển một khối dữ liệu từ thiết bị ngoại vi vào một vùng nhớ hoặc ngược lại. Với phương pháp vào/ra bằng chương trình, dữ liệu nào cũng phải đi qua CPU, do vậy làm chậm tốc độ trao đổi dữ liệu. Để khắc phục tình trạng này, ta dùng phương pháp trao đổi dữ liệu giữa một vùng nhớ với thiết bị ngoại vi một cách trực tiếp không thông qua CPU, đó là phương pháp *truy nhập trực tiếp bộ nhớ* (*DMA*). Trong phương pháp này, CPU giao quyền điều khiển BUS dữ liệu cho một chip điện tử chuyên dụng gọi là chip DMAC (DMA Controller). Chip DMAC tự tạo ra địa chỉ, tạo các tín hiệu điều khiển việc ghi/đọc bộ nhớ, đếm số từ dữ liệu đã được ghi vào hoặc đọc từ bộ nhớ và sẽ thông báo cho CPU khi đã thực hiện xong việc trao đổi dữ liệu với bộ nhớ. Quá trình được thực hiện hoàn toàn bằng phần cứng, trực tiếp giữa thiết bị vào/ra và bộ nhớ nên tốc độ trao đổi thông tin tương đối nhanh. CPU không cần nhận lệnh, giải mã lệnh và thực hiện các lệnh di chuyển dữ liệu.



Hình II. 10 Mô tả các tín hiệu điều khiển trong quá trình DMA

Khi có yêu cầu trao đổi dữ liệu theo DMA, thiết bị ngoại vi gửi tín hiệu yêu cầu DRQ tới chip DMAC, chip này gửi tín hiệu yêu cầu treo HRQ tới chân HOLD của CPU. Nếu yêu cầu được chấp nhận, CPU sẽ gửi xung ghi nhận HLDA tới chân HACK của chip DMAC và tự treo các BUS, cho phép DMAC sử dụng BUS. DMAC gửi tín hiệu DACK tới thiết bị ngoại vi cho phép thiết bị này thực hiện việc trao đổi dữ liệu. Kết thúc quá trình trao đổi dữ liệu, chip DMAC chuyển trạng thái của tín hiệu HRQ về mức thấp để thông báo cho CPU.

II.1.8 Vi chương trình (MicroProgram) và tập lệnh của μP8085

a) Đơn vị điều khiển CU – Control Unit

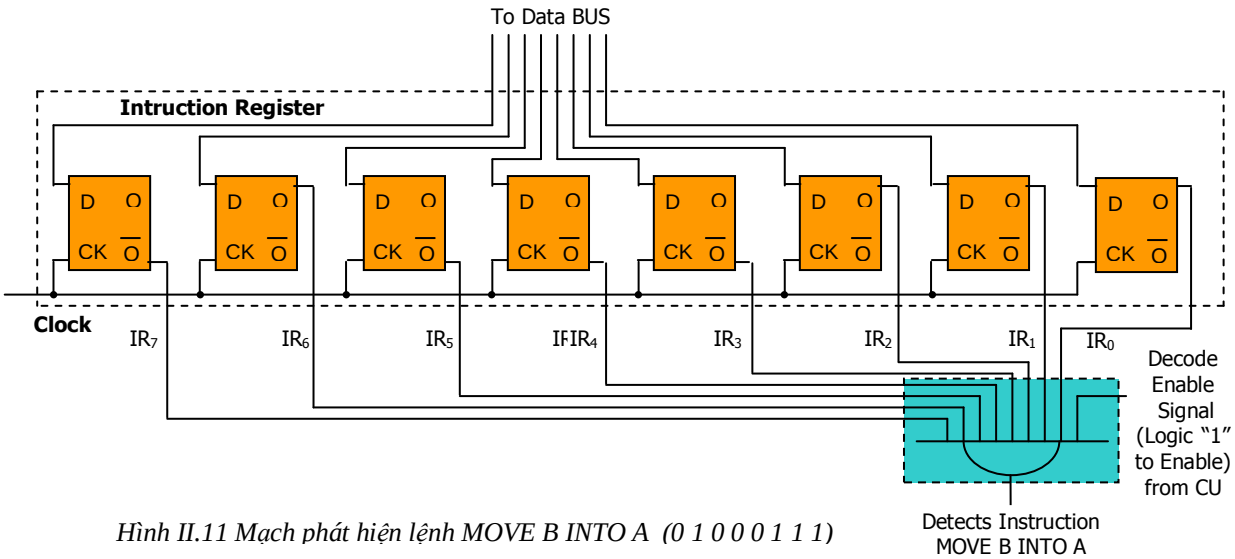
CU - Control Unit là đơn vị điều khiển, điều phối mọi hoạt động của các bộ phận chức năng trong CPU thông qua Control BUS. Có thể coi CU là khối dịch lệnh của CPU, nó tạo ra các tín hiệu tương ứng làm đầu vào cho Controller Unit để điều khiển hoạt động của các khối chức năng. Các tín hiệu do CU tạo ra có thể phân thành 2 loại: Tín hiệu định thời và tín hiệu điều hành hoạt động của CPU. Các tín hiệu định thời do CU tạo ra xác định trạng thái của CPU làm việc:

- *Đang ở chế độ đọc dữ liệu vào (Input mode)*
- *Đang đưa dữ liệu ra (Output mode)*
- *Đang bắt đầu một hoạt tác khác (Beginning another operation).*

Các tín hiệu trạng thái của CPU xác định CPU đang:

- *Đọc dữ liệu từ bộ nhớ (Memory Read)*
- *Ghi dữ liệu vào bộ nhớ (Memory Write)*
- *Nhận lệnh (Instruction Fetch)*
- *Đọc dữ liệu từ thiết bị ngoại vi (I/O Read)*
- *Đưa dữ liệu ra thiết bị ngoại vi (I/O Write)*

Cũng có thể có những thao tác không được nêu ở đây, nhưng chỉ các thao tác trên là quan trọng nhất.



Hình II.11 Mạch phát hiện lệnh MOVE B INTO A (0 1 0 0 0 1 1 1)

Cần hiểu rằng mạch Controller Logic tạo các tín hiệu điều khiển dựa vào các tín hiệu trạng thái của CPU và tín hiệu định thời, có nghĩa là tạo tín hiệu gì và vào thời điểm nào.

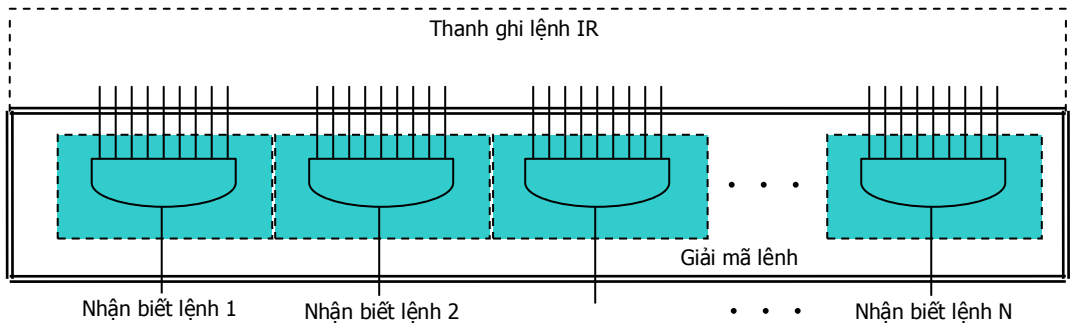
Để hiểu được kiến trúc khối CU, hãy tìm lời giải đáp cho câu hỏi: *Sau khi nhận lệnh, CPU làm sao " biết" phải thực hiện những thao tác nào để thực hiện lệnh?*

Tất cả các lệnh đều được biểu diễn dưới dạng mã nhị phân. Giả sử lệnh được biểu diễn bằng một mã 8 bits 01000111B (chuyển nội dung thanh ghi B sang thanh ghi A, ký hiệu là [A] ⇐ [B]).

Trước hết, lệnh phải được *giải mã*. Một mạch AND có thể sử dụng để tạo ra tín hiệu nhận biết lệnh (Hình II.11). Đầu vào của mạch AND này được nối với đầu ra của thanh ghi lệnh. Đầu ra của các phần tử trong thanh ghi lệnh xác nhận sự hiện diện của lệnh MOVE B TO A theo công thức

$$(\text{MOVE B TO A}) = \overline{IR}_7.IR_6.\overline{IR}_5.\overline{IR}_4.\overline{IR}_3.IR_2.IR_1.IR_0.$$

trong đó IR_n là đầu ra của các flip-flop tương ứng với các giá trị nhị phân của mã lênh MOVE B TO A. Mạch AND nhận biết mã lệnh được gọi là mạch giải mã lệnh. Như vậy, nếu CPU sử dụng 8 bit để mã hoá các lệnh, có thể có 256 lệnh, và mạch giải mã lệnh cũng sẽ cần đến 256 mạch AND tương tự, tuy nhiên đầu vào của mỗi mạch là một tổ hợp duy nhất trong 256 tổ hợp có thể.



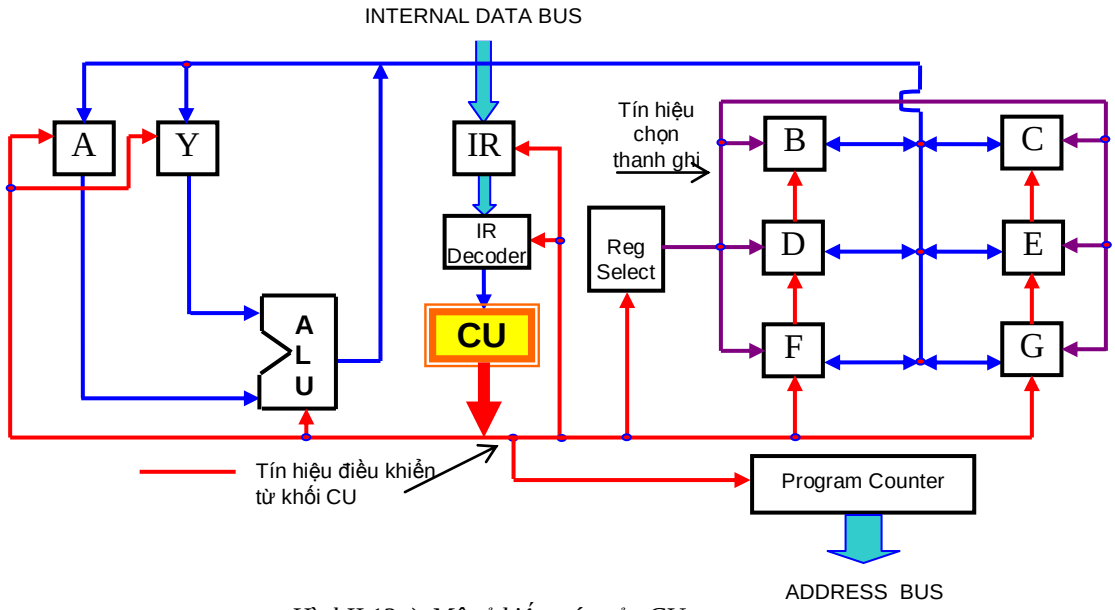
Hình II.12 Nhận biết các lệnh từ tổ hợp mã nhị phân

Để thực hiện lệnh, khối điều khiển CU xúc tiến mọi thao tác ngay bên trong CPU bằng cách tạo ra các tín hiệu điều khiển và các xung nhịp để định thời cho các khối chức năng thực hiện các thao tác.

Sau khi nhận tín hiệu từ khối giải mã lệnh (Instruction Decoder), CU sẽ tạo ra các tín hiệu điều khiển và các xung nhịp. Tín hiệu điều khiển sẽ cho phép (Enable) khối chọn thanh ghi (Reg Select) chọn thanh ghi B và thiết lập hệ thống đường truyền thông suốt giữa hai thanh ghi B và A. tiếp theo CU sẽ tạo các tín hiệu tương ứng để việc truyền dữ liệu giữa hai thanh ghi được thực hiện.

Tiếp theo, CU điều khiển thanh đếm chương trình PC tăng lên 1 để nhận tiếp lệnh từ bộ nhớ. Vì CU có nhiệm vụ giám sát và điều khiển mọi thao tác của các thành phần chức năng trong CPU, nên các dây điều khiển phải được nối trực tiếp từ CU tới mọi khối chức năng trong CPU như trên hình

II.13a. Cũng cần nhận thức rằng, lệnh được CPU lấy từ bộ nhớ. Trong thực tế, dữ liệu để xử lý cũng có thể xuất phát từ bộ nhớ, và các thanh ghi cũng có thể được chọn bất kỳ ngoại trừ thanh ghi lệnh IR và thanh đếm chương trình PC.



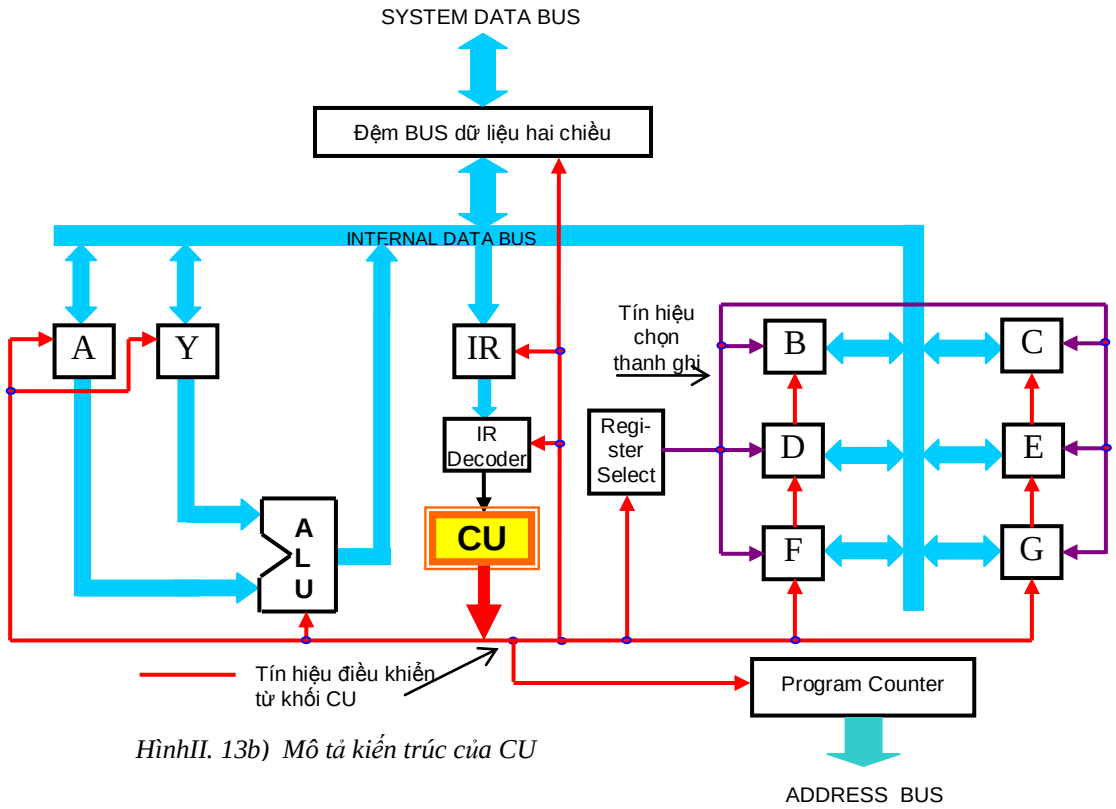
HìnhII.13a) Mô tả kiến trúc của CU

Như vậy, lại cần thêm một thanh ghi liên lạc với BUS dữ liệu có nhiệm vụ truy nhập được vào bộ nhớ. Thanh ghi này làm trung gian giữa BUS dữ liệu bên ngoài và các thanh ghi đa năng khác, và nó được liên lạc với nhau thông qua BUS dữ liệu nội bộ (Internal Data BUS) - một BUS mà các thanh ghi được truy xuất trực tiếp. CU phải làm nhiệm vụ xác định thanh ghi nào được truy xuất qua BUS dữ liệu nội bộ tại thời điểm đó. Cũng vì BUS dữ liệu nội bộ của CPU truy xuất đến BUS dữ liệu hệ thống, nên cần phải có một cách thức để hoặc cách ly chúng khi cần thiết, hoặc cho phép ghép nối, nên cần thiết phải có thêm *thanh ghi đệm dữ liệu hai chiều*. Và như vậy, CU phải làm nhiệm vụ *điều khiển hướng di chuyển* của dữ liệu khi đi qua thanh ghi đệm (xem hình II.13b).

b) Vi chương trình

Giả thiết rằng lối ra của khối *giải mã lệnh và tạo các tín hiệu điều khiển* phải tạo ra 12 tín hiệu tại các cửa $G_1 – G_{12}$, 2 tín hiệu điều khiển bộ nhớ và 5 tín hiệu xung nhịp kích hoạt các thanh ghi PC (thanh đếm chương trình), MAR (thanh ghi đệm địa chỉ, MSR (thanh ghi đệm bộ nhớ), DO (thanh ghi dữ liệu) và IR (thanh ghi lệnh) để điều khiển quá trình nhận và thực hiện lệnh ADD. Các tín hiệu này được gửi tới để điều khiển hoạt động của các thành phần khác nhau trong CPU. Một chu trình thực hiện lệnh trên sẽ được thi hành.

Thực tế trong CPU của máy tính có từ 64 đến hơn 200 các tín hiệu điều khiển như thế. Sự khác nhau quan trọng giữa các lệnh và vi lệnh là ở chỗ vi lệnh có nhiều trường hơn. Tám bước trong bảng trên là một *vi chương trình* dịch một giai đoạn nhận lệnh (OPCODE FETCH) được thực thi sau lệnh cộng ADD. Như vậy một lệnh được dịch thành một chuỗi các vi lệnh, hay nói cách khác, mỗi mã lệnh có một vi chương trình.



HìnhII. 13b) Mô tả kiến trúc của CU

Số bước	Các tín hiệu điều khiển cửa												Điều khiển bộ nhớ		Các xung nhịp kích hoạt thanh ghi				
	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}	EN	E/W	PC	MAR	MBR	DO	IR
1	1	0	0	0	0	0	0	0	0	0	0	0	0	x	0	1	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	x	1	0	0	0	0
3	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0
4	0	0	0	0	0	0	1	0	0	0	0	0	0	x	0	0	0	0	1
5	0	0	1	0	0	0	0	0	0	0	0	0	0	x	0	1	0	0	0
6	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0
7	0	0	0	0	0	0	1	0	0	1	0	0	0	x	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	1	0	x	0	0	0	1	0

Có thể thấy rằng, *khối giải mã lệnh và tạo các tín hiệu điều khiển*:

+ "Biết" phải thực hiện lệnh "như thế nào", một khi lệnh từ IR (Instruction Register) được chuyển tới.

+ Giải quyết việc thực hiện một lệnh bằng cách điều khiển các khối chức năng liên quan thực hiện các phần việc.

Từ cách nhìn nhận trên, dễ dàng nhận ra rằng khối giải mã lệnh và tạo các tín hiệu điều khiển là bộ não thực thụ của CPU. Có thể coi khối này là một máy tính đặc dụng (*Special-purpose Computer*)(*) bên trong CPU. Nó là hạt nhân cơ bản nhất dành riêng cho việc thực hiện một lệnh. Để thiết kế và xây dựng được khối này, cần phải có một "chương trình" (*program*)(*) thật chi tiết. Chương trình dùng để xây dựng nên khối này cần phải có những thủ tục tuyệt đối chính xác nhằm mục đích thực hiện các lệnh.

Chương trình đó được gọi là *Vi chương trình* (MicroProgram) và được chế tạo như là một phần tích hợp cứng bên trong CPU, người lập trình không thể thay thế cũng như không thể truy nhập vào được.

> Đối với các loại μP dạng *bit-slice microprocessor*, Vi chương trình hoàn toàn do người sử dụng xây dựng.

b) Tập lệnh của μP8085

Tập lệnh của μP8085 có thể chia thành nhiều nhóm lệnh nhỏ tuỳ theo từng cách tiếp cận. Theo phương thức xử lý và kết quả của việc xử lý dữ liệu, các lệnh trong tập lệnh được chia thành 4 nhóm chính:

1. Nhóm lệnh chuyển dữ liệu: các lệnh trong nhóm này thực hiện việc di chuyển dữ liệu giữa các thanh ghi với nhau, giữa thanh ghi với bộ nhớ và ngược lại, các lệnh vào/ra dữ liệu v.v...
2. Nhóm lệnh số học và logic: các lệnh trong nhóm này thực hiện các phép tính số học cơ bản là cộng và trừ 2 toán hạng, các lệnh tăng giảm, hay so sánh nội dung thanh ghi, các phép tính logic trong số sọc nhị phân, các phép dịch trái, phải dữ liệu trong thanh ghi, lệnh quay vòng trái phải v.v...
3. Nhóm lệnh điều khiển: Bao gồm các nhóm lệnh rẽ nhánh có điều kiện và không diều kiện, các lệnh gọi chương trình con
4. Nhóm lệnh đặc biệt: Nhóm lệnh đặc biệt bao gồm các lệnh lấy bù 1 của số liệu trong nội dung thanh ghi, lệnh thiết lập và xoá các cờ, lệnh hiệu chỉnh thập phân một số liệu Hexa và lệnh vào/ra dữ liệu nối tiếp.

(*) Thuật ngữ do hãng Intel sử dụng trong tài liệu gốc

II.1.9 Vài nét về lập trình cho 8085

Phát triển phần mềm (lập trình) và các kỹ thuật liên quan đóng vai trò quan trọng bậc nhất trong các ứng dụng từ đơn giản đến phức tạp của các hệ Vi xử lý và máy vi tính. Đối với các hệ Vi xử lý, mọi ứng dụng đều được phát triển nhờ vào một "công cụ" phát triển phần mềm hoàn chỉnh: Lập trình hợp ngữ.

Quá trình phát triển một chương trình (phần mềm ứng dụng) cho một hệ Vi xử lý, kể từ khi xác định nhiệm vụ cần thực hiện cho đến khi chương trình được cài đặt hoàn chỉnh vào hệ thống có thể chia ra năm bước cơ bản sau đây:

a) ***Đặt vấn đề (xác nhận vấn đề):*** Trước khi giải quyết vấn đề, người lập trình cần xác định xem, liệu vấn đề có thể được giải quyết nhờ một chương trình trong một hệ Vi xử lý hay không. Phải thấy rằng không phải hệ Vi xử lý "vạn năng" đến mức có thể giải quyết tất cả mọi vấn đề nảy sinh trong thực tiễn, thậm chí đôi khi còn làm cho sự việc càng thêm phức tạp.

b) ***Xác định phương pháp giải quyết vấn đề:*** Đây chính là bước tìm thuật giải (Algorithm) tối ưu cho vấn đề được đặt ra. Người lập trình phải tìm và lựa chọn được từ nhiều giải pháp một giải pháp tốt nhất, nhưng kinh tế nhất để thực hiện. Không chỉ tìm giải thuật tốt nhất mà còn phải tìm ngôn ngữ lập trình phù hợp nhất để giải quyết vấn đề.

c) ***Thực hiện giải pháp:*** Phương pháp giải quyết vấn đề thường được xác nhận qua từng bước theo một lưu đồ. Lưu đồ là cách thể hiện tường minh các bước thực hiện chương trình trong hệ thống, đồng thời nó giúp người lập trình định hướng tốt khi viết chương trình.

d) ***Viết chương trình :*** Bản thân lưu đồ đã cho thấy rõ giải pháp giải quyết vấn đề theo quan điểm lập trình. Việc chuyển từ lưu đồ sang ngôn ngữ chương trình là bước dễ dàng hơn rất nhiều so với cách viết chương trình không có lưu đồ. Đây chỉ là bước cụ thể hóa lưu đồ nhờ tuần tự thực hiện các lệnh, và là bước thực tế hóa giải pháp thực hiện vấn đề.

e) ***Kiểm tra và gỡ rối:*** Sau khi cài đặt việc kiểm tra tính chính xác là vô cùng quan trọng. Những sai sót phải được phát hiện và hiệu chỉnh, đôi khi là từ chính thuật giải. Việc gỡ rối chương trình tức là thực hiện từng bước chương trình, phát hiện các sai sót ẩn, hiệu chỉnh các sai sót này.

Để thực hiện được tất cả các các bước trên người lập trình phải có kỹ thuật lập trình hoàn thiện để thiết kế chương trình, phải có các công cụ lập trình tốt.

II.1.10 Hệ lệnh của µP8085

Các lệnh của µP8085 được thống kê trong bảng II.1

Mnemonic	Instruction Code D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Mô tả nhiệm vụ
MOVE, LOAD, AND STORE		
MOV r1,r2	0 1 D D D S S S	Move Register To Register
MOV M,r	0 1 1 1 0 S S S	Move Register To Memory
MOV r,M	0 1 D D D 1 1 0	Move Memory To Register
MVI r	0 0 D D D 1 1 0	Move Immediate Register
MVI M	0 0 1 1 0 1 1 0	Move Immediate Memory
LXI B	0 0 0 0 0 0 0 1	Load Immediate Register Pair B
LXI D	0 0 0 1 0 0 0 1	Load Immediate Register Pair D
LXI H	0 0 1 0 0 0 0 1	Load Immediate Register Pair H
STAX B	0 0 0 1 0 0 1 0	Store A indirect
STAX D	0 0 0 0 1 0 1 0	Store A indirect
LDAX B	0 0 0 1 0 0 1 0	Load A indirect
LDAX D	0 0 0 1 1 0 1 0	Load A indirect
STA	0 0 1 1 0 0 1 0	Store A direct
LDA	0 0 1 1 1 0 1 0	Load A direct
SHLD	0 0 1 0 0 0 1 0	Store H & L direct
LHLD	0 0 1 0 1 0 1 0	Load H & L direct
XCHG	1 1 1 0 1 0 1 1	Exchange D & E, H & L registers
STACK OPS		
PUSH B	1 1 0 0 0 1 0 1	Push Register Pair B & C on
PUSH D	1 1 0 1 0 1 0 1	Push Register Pair D & E on
PUSH H	1 1 1 0 0 1 0 1	Push Register Pair H & L on
PUSH PSW	1 1 1 1 0 1 0 1	Push A ang Flags on stack
POP B	1 1 0 0 0 0 0 1	Pop Register Pair B & C off stack
POP D	1 1 0 1 0 0 0 1	Pop Register Pair D & E off stack
POP H	1 1 1 0 0 0 0 1	Pop Register Pair H & L off stack
POP PSW	1 1 1 1 0 0 0 1	Pop A ang Flags off stack
XTHL	1 1 1 0 0 0 1 1	Exchange register pair H & L, top
SPHL	1 1 1 1 1 0 0 1	H & L to stack pointer
LXI SP	0 0 1 1 0 0 0 1	Load immediate stack pointer
INX SP	0 0 1 1 0 0 1 1	Increment stack pointer
DCX SP	0 0 1 1 1 0 1 1	Decrement stack pointer
JUMP		
JMP	1 1 0 0 0 0 1 1	Jump uncoditional
JC	1 1 0 1 1 0 1 0	Jump on carry
JNC	1 1 0 1 0 0 1 0	Jump on no carry
JZ	1 1 0 0 1 0 1 0	Jump on zero
JNZ	1 1 0 0 0 0 1 0	Jump on no zero
JP	1 1 1 1 0 0 1 0	Jump on positive
JM	1 1 1 1 1 0 1 0	Jump on minus
JPE	1 1 1 0 1 0 1 0	Jump on parity even
JPO	1 1 1 0 0 0 1 0	Jump on parity odd
PCHL	1 1 1 0 1 0 0 1	H & L to program counter
CALL		
CALL	1 1 0 0 1 1 0 1	Call unconditional
CC	1 1 0 1 1 1 0 0	Call on carry
CNC	1 1 0 1 0 1 0 0	Call on no carry
CZ	1 1 0 0 1 1 0 0	Call on zero
CNZ	1 1 0 0 0 1 0 0	Call on no zero
CP	1 1 1 1 0 1 0 0	Call on positive
CM	1 1 1 1 1 1 0 0	Call on minus

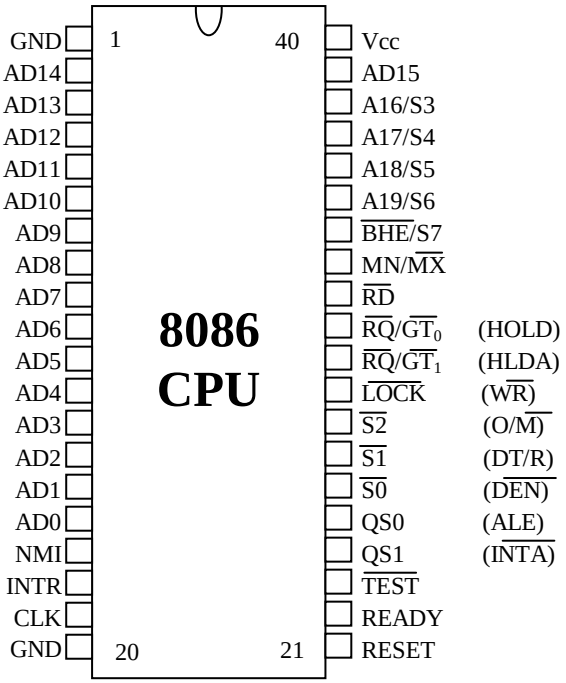
CPE	1	1	1	0	1	1	0	0	Call on parity even
CPO	1	1	1	0	0	1	0	0	Call on patity odd
RETURN									
RET	1	1	0	0	1	0	0	1	Return
RC	1	1	0	1	1	0	0	0	Return on carry
RNC	1	1	0	1	0	0	0	0	Return on no carry
RZ	1	1	0	0	1	0	0	0	Return on zero
RNZ	1	1	0	0	0	0	0	0	Return on no zero
RP	1	1	1	1	0	0	0	0	Return on positive
RM	1	1	1	1	1	0	0	0	Return on minus
RPE	1	1	1	0	1	0	0	0	Return on parity even
RPO	1	1	1	0	0	0	0	0	Return on patity odd
RESTART									
RST	1	1	A	A	A	1	1	1	Restart
INPUT/OUTPUT									
IN	1	1	0	1	1	0	1	1	Input
OUT	1	1	0	1	0	0	1	1	Output
RIM	0	0	1	0	0	0	0	0	Read interrupt mask
SIM	0	0	1	1	0	0	0	0	Set interrupt mask
INCREMENT AND DECREMENT									
INR r	0	0	D	D	D	1	0	1	Increment register
DCR r	0	0	D	D	D	1	0	1	Decrement register
INR M	0	0	1	1	0	1	0	0	Increment memory
DCR M	0	0	1	1	0	1	0	1	Decrement memory
INX B	0	0	0	0	0	0	1	1	Increment B&C registers
INX D	0	0	0	1	0	0	1	1	Increment D&E registers
INX H	0	0	1	0	0	0	1	1	Increment H&L registers
DCX B	0	0	0	0	1	0	1	1	Decrement B&C registers
DCX D	0	0	0	1	1	0	1	1	IDerement D&E registers
DCX H	0	0	1	0	1	0	1	1	Decrement H&L registers
ADD									
ADD r	1	0	0	0	0	S	S	S	Add register to A
ADC r	1	0	0	0	1	S	S	S	Add register to A with carry
ADD M	1	0	0	0	0	1	1	0	Add memory to A
ADC M	1	0	0	0	1	1	1	0	Add memory to A with carry
ADI	1	1	0	0	0	1	1	0	Add immediate to A
ACI	1	1	0	0	1	1	1	0	Add immediate to A with carry
DAD B	0	0	0	0	1	0	0	1	Add B&C to H&L
DAD D	0	0	0	1	1	0	0	1	Add D&E to H&L
DAD H	0	0	1	0	1	0	0	1	Add H&L to H&L
DAD SP	0	0	1	1	1	0	0	1	Add SP to H&L
SUBTRACT									
SUB r	1	0	0	1	0	S	S	S	Subtract register from A
SBB r	1	0	0	1	1	S	S	S	Subtract register from A with
SUB M	1	0	0	1	0	1	1	0	Subtract memory from A
SBB M	1	0	0	1	1	1	1	0	Subtract memory from A with
SUI	1	1	0	1	0	1	1	0	Subtract immediate from A
SBI	1	1	0	1	1	1	1	0	Subtract immediate from A with
LOGICAL									
ANA r	1	0	1	0	0	S	S	S	And register with A
XRA r	1	0	1	0	1	S	S	S	Exclusive OR register with A
ORA r	1	0	1	1	0	S	S	S	OR register with A
CMP r	1	0	1	1	1	S	S	S	Compare register with A
ANA M	1	0	1	0	0	1	1	0	And memory with A
XRA M	1	0	1	0	1	1	1	0	Exclusive memory with A
ORA M	1	0	1	1	0	1	1	0	OR memory with A
CMP M	1	0	1	1	1	1	1	0	Compare memory with A

ANI	1 1 1 0 0 1 1 0	And immediate with A
XRI	1 1 1 0 1 1 1 0	Exclusive OR immediate with A
ORI	1 1 1 1 0 1 1 0	OR immediate with A
CPI	1 1 1 1 1 1 1 0	Compare immediate with A
ROTATE		
RLC	0 0 0 0 0 1 1 1	Rotate A left
RRC	0 0 0 0 1 1 1 1	Rotate A right
RAL	0 0 0 1 0 1 1 1	Rotate A left through carry
RAR	0 0 0 1 1 1 1 1	Rotate A right through carry
SPECIALS		
CMA	0 0 1 0 1 1 1 1	Complement A
STC	0 0 1 1 0 1 1 1	Set carry
CMC	0 0 1 1 1 1 1 1	Complement carry
DAA	0 0 1 0 0 1 1 1	Decimal adjust A
CONTROL		
EI	1 1 1 1 1 0 1 1	Enable interrupt
DI	1 1 1 1 0 0 1 1	Disable interrupt
NOP	0 0 0 0 0 0 0 0	No-operation
HLT	0 1 1 1 1 1 1 0	Halt

II.2 Các trung tâm Vi xử lý họ 80x86

II.1.1 Mô tả chân của µP8086 và các tín hiệu

µP8086 được chế tạo theo công nghệ HMOS, đóng vỏ CerDIP 40 chân. Là loại Vi xử lý có khả năng xử lý trực tiếp dữ liệu 8 hoặc 16 bit. Về tập lệnh, µP8086 hoàn toàn tương thích với tập lệnh của iAPX86/10 và về phần cứng, hoàn toàn tương thích với các mạch ngoại vi của các trung tâm 8080/8085 của Intel.



Hình II. 14. Sơ đồ nối chân trung tâm Vi xử lý 8086

µP8086 có thể hoạt động ở một trong hai chế độ:

Chế độ MIN: CPU tự tạo ra các tín hiệu điều khiển hoạt động của BUS (các chân từ 24 đến 34)

– *Chế độ MAX:* CPU chỉ đưa ra các tín hiệu trạng thái, cần thêm một *chip điều khiển BUS* (BUS controller 8288) và chip này sẽ thông dịch các tín hiệu trạng thái thành các tín hiệu điều khiển BUS tương thích với cấu trúc MULTIBUS™, cách này đảm bảo hoạt động đọc số liệu ổn định hơn.

Hình II. 11 là sơ đồ nối chân của µP8086

+ AD15 – AD0: BUS dồn kênh dữ liệu và địa chỉ 16 bits

+ A19 – A16 / S6 – S3: 4 bits địa chỉ cao hoặc 4 tín hiệu trạng thái hoạt động hiện tại của CPU

S4	S3	Thanh ghi được truy xuất
0	0	ES
0	1	SS
1	0	CS
1	1	DS

S5 chỉ trạng thái cờ ngắt
S6 luôn luôn bằng 0

+ $\overline{\text{BHE}}$/S7: Tín hiệu này kết hợp với chân địa chỉ A0 cho chỉ thị các trạng thái sau:

BHE	A0	
0	0	Một từ đã được truyền qua D15 – D0
0	1	Một Byte trên D15 – D8 được truy xuất tới một địa chỉ Byte lẻ
1	0	Một Byte trên D7 – D0 được truy xuất tới một địa chỉ Byte chẵn
1	1	chưa xác định

+ $\overline{\text{RD}}$: Nếu bằng "1" đang đọc bộ nhớ (hoặc thiết bị vào/ra)
Nếu bằng "0" đang ghi ra bộ nhớ (hoặc thiết bị vào/ra)

+ READY: nếu bộ nhớ (hoặc thiết bị vào/ra) cần truy nhập hoàn tất việc chuyển dữ liệu đến (hoặc đi) chúng cần phát ra tín hiệu READY ở mức "1" tới chân CPU, chỉ khi ấy CPU mới đọc số liệu vào hoặc đưa dữ liệu ra

+ INTR: CPU kiểm tra trạng thái chân này sau khi thực hiện xong mỗi lệnh để xét xem có yêu cầu ngắt từ phần cứng đến hay không, nếu ở mức "1", CPU sẽ chuyển sang phục vụ ngắt. Thao tác kiểm tra này có thể "chr" được nhờ dùng mặt nạ che ngắt

+ $\overline{TEST}$: Lối vào này của CPU luôn luôn được kiểm tra trong lệnh WAIT. Nếu bằng "0" CPU tiếp tục thực hiện chương trình, nếu bằng "1", CPU chạy các chu trình giả cho tới khi $\overline{TEST}$ = "0".

+ NMI: Chân ngắt theo sườn lên của xung, không che được.

+ RESET: Chân nhận tín hiệu tái khởi động hệ thống. Nếu có sự thây đổi từ "0" lên "1" và tồn tại tối thiểu trong 4 nhịp đồng hồ thì hệ thống sẽ tự khởi động lại.

+ CLK: Lối vào của xung nhịp đồng hồ

+ Vcc: Nguồn nuôi +5V

+ GND: Chân nối đất (0V)

+ MN/$\overline{MX}$: Khi được nối với Vcc, µP8086 hoạt động ở chế độ MIN, nếu nối với GND, hoạt động ở chế độ MAX

+ $\overline{S2}$, $\overline{S1}$, $\overline{S0}$: Ở chế độ MAX, chip điều khiển BUS sử dụng 3 tín hiệu trạng thái này để phát ra các tín hiệu điều khiển truy xuất bộ nhớ và thiết bị vào/ra. Tổ hợp có ý nghĩa như sau

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	
0	0	0	yêu cầu ngắt cứng qua chân INTR được chấp nhận
0	0	1	đọc thiết bị vào/ra
0	1	0	Ghi thiết bị vào/ra
0	1	1	CPU bị treo
1	0	0	nạp mã chương trình vào hàng nhận lệnh
1	0	1	đọc bộ nhớ
1	1	0	ghi vào bộ nhớ
1	1	1	trạng thái thụ động

+ $\overline{RQ}/\overline{GT_0}$, $\overline{RQ}/\overline{GT_1}$: Tín hiệu phục vụ việc chuyển mạch BUS cục bộ (Local BUS) giữa các đơn vị làm chủ BUS (BUS master). BUS cục bộ là BUS giữa các đơn vị xử lý (không phải là BUS nối với các thiết bị ngoại vi). Đơn vị làm chủ BUS là µP8086 hoặc một chip điều khiển nào đó (ví dụ DMAC) hiện đang nắm quyền điều khiển BUS cục bộ.

+ $\overline{LOCK}$: nếu bằng "0", đơn vị làm chủ BUS không nhượng quyền làm chủ BUS cục bộ

+ QS_1, QS_0 chỉ thị trạng thái của hàng nhận lệnh trước PQ

0	0	không hoạt động

0 1 byte 1 của mã toán trong PQ được xử lý
1 0 hàng đợi lệnh được xoá
1 1 byte 2 của mã toán trong PQ được xử lý

II.1.2 Cấu trúc Trung tâm Vi xử lý họ 80x86

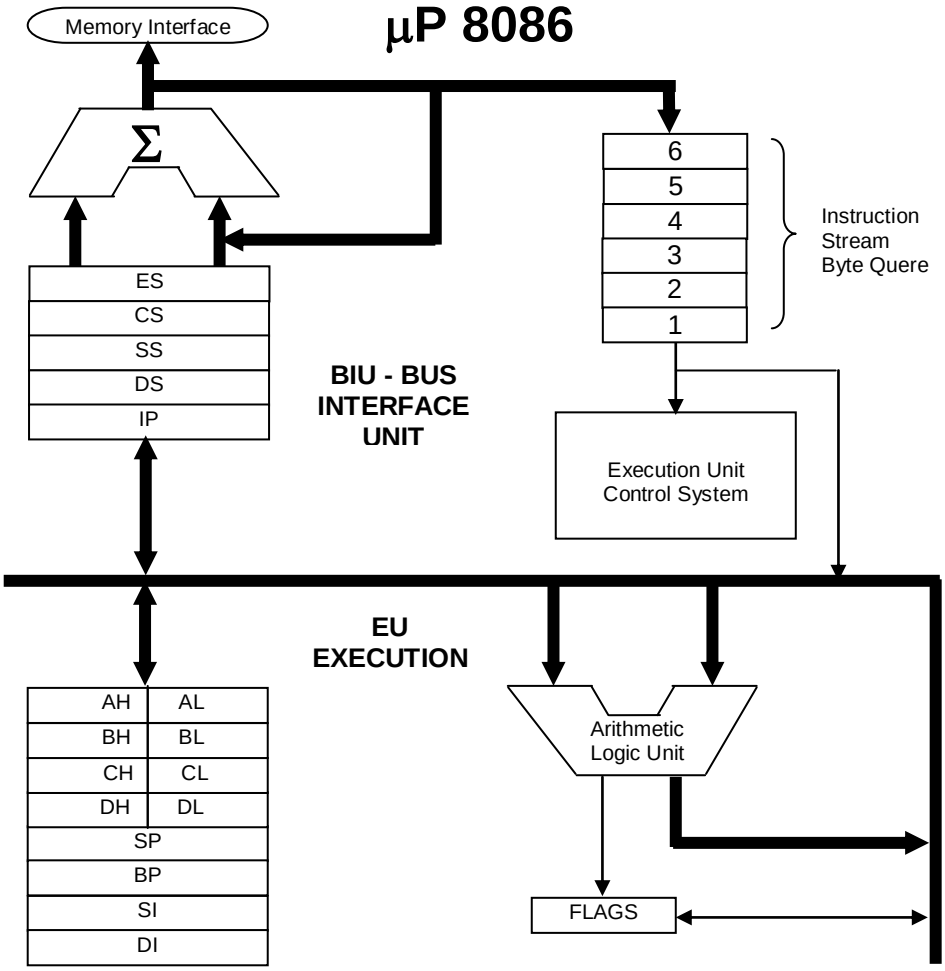
Các μP họ 80x86 được phát triển trên cơ sở công nghệ VLSI với các phần tử cơ bản là các transistor trường MOS có độ tiêu hao công suất rất nhỏ. Sơ đồ khối chức năng của μP8086 được thể hiện trên hình II.15, gồm hai thành phần chủ yếu là *đơn vị ghép nối BUS* (BIU), *đơn vị thực hiện lệnh* (EU). Tất cả các thanh ghi và đường truyền dữ liệu trong EU đều có độ dài 16 bits. BIU thực hiện tất cảc các nhiệm vụ về BUS cho EU: thiết lập khâu liên kết với BUS dữ liệu, BUS địa chỉ và BUS điều khiển. Dữ liệu được trao đổi giữa CPU với bộ nhớ khi EU có yêu cầu, song không được truyền trực tiếp tới EU mà thông qua một vùng nhớ RAM dung lượng nhỏ (6 bytes) được gọi là *hàng nhận lệnh trước* (Instruction Stream Byte Quere PQ - Prefetch Quere) rồi mới được truyền cho hệ *thống điều khiển EU* (Execution Unit Control System).

Khi EU đang thực hiện một lệnh thì BIU đã tìm và lấy lệnh sau đặt sẵn vào PQ. Đây là *cơ chế đường ống* (pipeline), một kỹ thuật tăng tốc dộ cho CPU.

> Kỹ thuật đường ống sử dụng một vùng nhớ RAM cực nhanh, làm tăng đáng kể tốc độ của bộ Vi xử lý thông qua việc truy tìm lệnh từ bộ nhớ chương trình thay cho sự liên hệ giữa CPU với bộ nhớ chương trình. Riêng với bộ xử lý Pentium, có hai đường ống, một cho các lệnh và một cho các dữ liệu.

Bảng sau cho ta vài thông số kỹ thuật cơ bản của các trung tâm Vi xử lý họ 80x86

Loại μP	Độ dài thanh ghi	Độ rộng BUS địa chỉ	Độ rộng BUS dữ liệu	Không gian địa chỉ	Tần số cực đại
8088	16 bits	20 bits	8 bits	1MByte	10 MHz
8086	16 bits	20 bits	16 bits	1Mbyte	10 MHz
80188	16 bits	20 bits	8 bits	1Mbyte	10 MHz
80186	16 bits	20 bits	16 bits	1Mbyte	10 MHz
80286	16 bits	24 bits	16 bits	16Mbytes	16 MHz
80386SX	32 bits	24 bits	16 bits	16Mbytes	20MHz
80386DX	32 bits	32 bits	32 bits	4Gbytes	40 MHz
i486	32 bits	32 bits	32 bits	4Gbytes	66 MHz
i486SX	32 bits	32 bits	32 bits	4Gbytes	25 MHz
Pentium (phiên bản đầu)	32 bits	32 bits	64 bits	4Gbytes	66 MHz

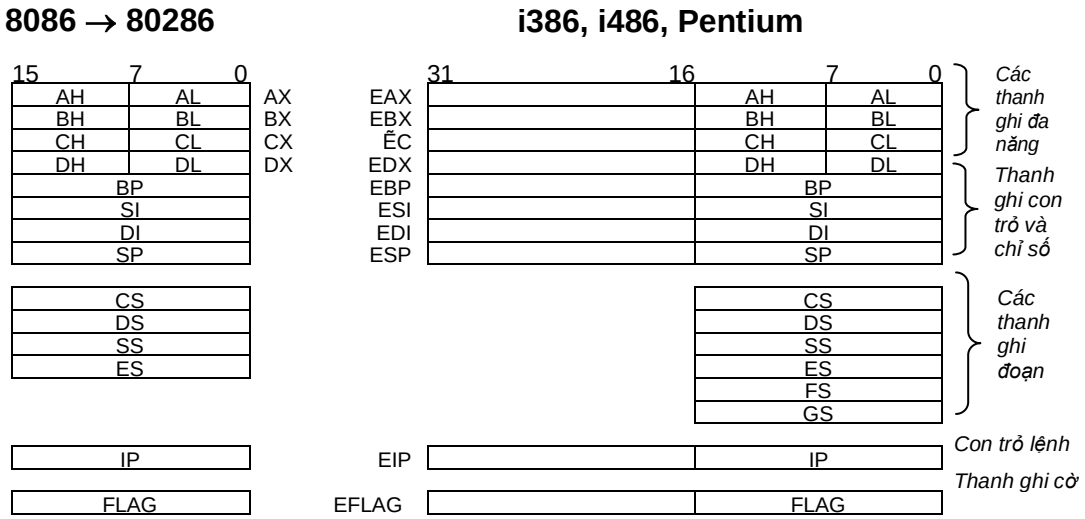


Hình II. 15 Cấu trúc các khối chức năng μP8086

II.1.3 Hệ thống thanh ghi trong các μP80x86

Có thể coi các thanh ghi của các trung tâm Vi xử lý như một bộ nhớ được đặt ngay bên trong CPU, có tốc độ truy cập cực kỳ nhanh, được dùng để lưu giữ các dữ liệu và các kết quả tạm thời của các quá trình tính toán, xử lý. Các thanh ghi trong họ μP80x86 có độ dài khác nhau, 16 bits với các trung tâm 8088/86, 80188/86 và 80286, 32 bits với các trung tâm 80386/486 trở đi và được mô tả trên Hình II.13

EU của μP8086 có 8 thanh ghi đa năng với tên gọi là AH, AL, BH, BL, CH, CL, DH, DL. Những thanh ghi này có thể sử dụng riêng rẽ cho việc lưu giữ các dữ liệu nhị phân 8 bits. Cũng có thể sử dụng chúng thành từng cặp thanh ghi có tên gọi là AX (AH-AL), BX (BH-BL), CX (CH-CL), và DX (DH-DL) để lưu giữ các dữ liệu nhị phân 16 bits.



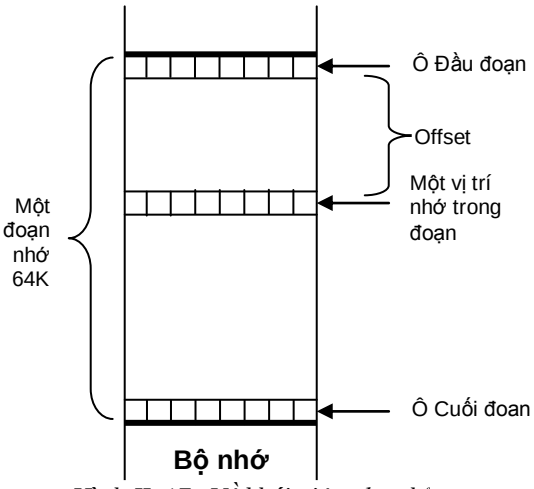
Hình II.16 Các thanh ghi trong các trung tâm Vi xử lý họ 80x86

1. *Các thanh ghi đa năng:*

Ưu điểm của việc sử dụng các thanh ghi này để lưu giữ tạm thời các dữ liệu là tốc độ truy cập của CPU với chúng nhanh hơn rất nhiều so với việc sử dụng các ô nhớ.

2. *Các thanh ghi đoạn:*

CPU đưa ra BUS địa chỉ 20 bits để quản lý một không gian nhớ 1Mbyte (1.048.576 Bytes) bộ nhớ vật lý. Tuy nhiên, các thanh ghi trong CPU lại chỉ có độ dài 16 bits, do vậy, không gian nhớ được chia thành *từng đoạn* (segment), mỗi đoạn dài 64Kbytes, địa chỉ của Byte đầu tiên được lấy làm *địa chỉ đoạn*. Hai đoạn nhớ kề cận cách nhau tối thiểu là 16 Bytes. Mỗi Byte nhớ trong đoạn sẽ được xác định bởi *độ lệch* (offset), tức là khoảng cách tính từ Byte nhớ đó đến đầu đoạn.



Hình II. 17 Về khái niệm địa chỉ đoạn và địa chỉ offset

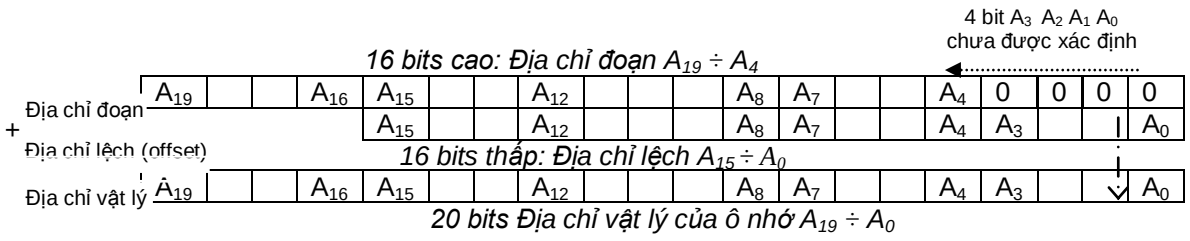
Như vậy, mỗi một cặp thông số bao gồm địa chỉ đoạn và độ lệch (*segment : offset*) sẽ xác định địa chỉ logic của một Byte nhớ vật lý trong bộ nhớ. Thanh ghi đoạn (Segment Register) chứa 16 bits cao, thanh ghi độ lệch (dùng thanh ghi đa năng hoặc các thanh ghi chỉ số, con trỏ) chứa 16 bit thấp của 20 bits địa chỉ. Địa chỉ vật lý của một vị trí nhớ do vậy sẽ được BIU tính theo công thức:

Địa chỉ vật lý = (Segment) x 10H + (offset)

µP8086 sử dụng 4 thanh ghi đoạn riêng biệt là: *Thanh ghi đoạn mã lệnh CS* (Code Segment), *thanh ghi đoạn ngăn xếp SS* (Stack Segment), *thanh ghi đoạn mở rộng ES* (Extra Segment) và *thanh ghi đoạn dữ liệu DS* (Data Segment).

- *Thanh ghi đoạn mã lệnh* CS là thanh ghi chứa địa chỉ bắt đầu của đoạn chương trình hiện hành trong bộ nhớ
- *Thanh ghi đoạn dữ liệu* DS là thanh ghi địa chứa chỉ bắt đầu của đoạn chứa số liệu hiện hành trong bộ nhớ, hay còn gọi là nơi chứa các biến của chương trình
- *Thanh ghi đoạn ngăn xếp* SS là thanh ghi địa chứa chỉ bắt đầu của đoạn ngăn xếp (Stack) trong bộ nhớ (ô nhớ do thanh ghi này chỉ đến còn được gọi là *đáy ngăn xếp*), nơi lưu giữ địa chỉ và dữ liệu khi thực hiện các chương trình con, lệnh gọi chương trình con hoặc thủ tục
- *Thanh ghi đoạn mở rộng* ES là thanh ghi địa chứa chỉ bắt đầu của đoạn chứa các dữ liệu chuỗi, xâu ký tự
- Ngoài ra, trong các trung tâm i386/i486 còn có hai thanh ghi đoạn FS và GS.

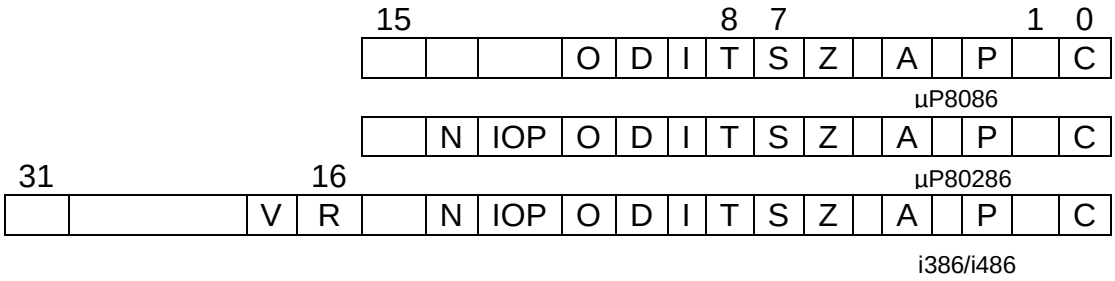
Các đoạn trong bộ nhớ có thể tách biệt nhau, nhưng cũng có thể gối chồng lên nhau, nhưng bao giờ cũng cách nhau tối thiểu 16 Bytes. Độ lệch 16Bytes này thực chất do 4 bit thấp nhất của địa chỉ từ A_3 đến A_0 chưa được xác định. Khi bộ cộng trong đơn vị địa chỉ tính địa chỉ vật lý để đưa ra BUS địa chỉ, nó lấy nội dung thanh ghi đoạn chèn thêm 4 số 0000_B cho 4 bits thấp nhất của 20 bits địa chỉ rồi mới cộng với 16 bits của phần địa chỉ offset. Điều này lý giải công thức tượng trưng đã nêu trên. Phần địa chỉ bắt đầu của đoạn được lưu giữ trong thanh ghi đoạn cũng thường được gọi là *địa chỉ cơ sở* hay *địa chỉ nền*.



Hình II. 18 Mô tả cách tính địa chỉ vật lý của một vị trí nhớ

3. Thanh ghi cờ FLAG:

Chỉ có 9 trong số 16 bits của thanh ghi cờ (trong các bộ vi xử lý µP8086 - µP80286) và 11 trong số 32 bits của thanh ghi cờ (trong các bộ xử lý i386/i486) được sử dụng. Mỗi cờ có thể được lập (= "1") hay xoá (= "0") để biểu thị trạng thái kết quả của một phép xử lý trước đó hoặc trạng thái hiện tại của CPU. Các cờ IOP, N, R và V liên quan đến chế độ bảo vệ trong các bộ xử lý 80286 và i386/i486. Chín cờ còn lại gồm 6 cờ chỉ trạng thái và 3 cờ điều khiển.



Hình II.19 Vị trí các cờ trong thanh ghi cờ của họ Vi xử lý 80x86

Các cờ trạng thái gồm:

- *Cờ nhớ* CF (carry flag) được lập nếu một thao tác xẩy ra hiện tượng *carry* hoặc *borrow* đối với toán hạng đích. CF có thể lập bởi lệnh STC và xoá bởi lệnh CLC.
- *Cờ chẵn lẻ* PF (parity flag) được lập nếu kết quả của một phép xử lý có số bit bằng "1" là số chẵn.
- *Cờ mang phụ* AF (auxiliary flag) được dùng cho xử lý các mã BCD và được lập nếu thao tác xử lý gây hiện tượng carry hoặc borrow cho 4 bits thấp của toán hạng
- *Cờ zero* ZF (zero flag) được lập nếu kết quả xử lý số liệu có kết quả bằng 0
- *Cờ dấu* SF (Sign flag) dấu tương ứng với MSB của kết quả phép toán, được lập với kết quả dương và xoá với kết quả âm
- *Cờ tràn* OF (Overflow flag) nếu kết quả phép toán là quá lớn cho toán hạng đích.

Các cờ điều khiển gồm:

- *Cờ hướng* DF (direction flag) xác định hướng của phép toán xử lý xâu, chuỗi ký tự, nếu được lập, xâu, chuỗi sè được xử lý từ địa chỉ cao tới địa chỉ thấp và ngược lại. Cờ được lập bởi lệnh STD và xoá bằng lệnh CLD

- *Cờ ngắt* IF (Interupt enable flag) nếu được lập, CPU sẽ chấp nhận yêu cầu ngắt cứng và phục vụ ngắt. Được lập bởi lệnh STI và xoá bằng lệnh CLI
- *Cờ bẫy* TF (Trap flag) Dùng trong gỡ rối chương trình (Debuger) Không thể lập hay xoá trực tiếp bởi lệnh của máy.

4. Thanh ghi con trỏ lệnh IP

Thanh ghi con trỏ lệnh IP (Instruction Pointer) – thanh ghi 16 bits dùng để lưu giữ phần offset của địa chỉ lệnh kế tiếp sẽ được thực hiện trong tuần tự thực hiện chương trình. Kết hợp với CS, IP giống như thanh đếm chương trình PC trong µP8085, mỗi lần từ lệnh được đọc ra từ bộ nhớ, BIU sẽ thay đổi giá trị IP tuỳ theo độ dài của từ lệnh (số bytes của từ lệnh) sao cho nó chỉ đến từ lệnh kế tiếp trong bộ nhớ chương trình. Cũng cần nói thêm rằng khi gặp các lệnh rẽ nhánh hoặc lệnh gọi chương trình con, gọi thủ tục …, các giá trị của CS:IP sẽ thay đổi đột ngột không theo quy luật trên. Các giá trị mới của CS:IP do người lập trình cung cấp thông qua địa chỉ của các nhãn (Label) trong chương trình hoặc giá trị cụ thể.

5. Các thanh ghi dữ liệu

Có 4 thanh ghi dữ liệu:

- *Thanh ghi tích luỹ* AX (Accummulator register) thường dùng để lưu giữ các kết quả xử lý
- *Thanh ghi cơ sở* BX (Base register) thường dùng chỉ địa chỉ cơ sở (đáy) của một vùng nhớ trong bộ nhớ
- *Thanh ghi đếm* CX (Counter register) thường dùng để khai báo số lần một thao tác nào đó phải được thực hiện trong các vòng lặp, phép dịch, phép quay…, Giá trị của nội dung thanh ghi CX sẽ giảm đi một sau mỗi thao tác
- *Thanh ghi số liệu* DX (Data register) thường dùng để lưu giữ số liệu dùng làm thông số chuyển giao cho một chương trình. DX là thanh ghi duy nhất được dùng để chứa địa chỉ của các thiết bị vào/ra

6. Các thanh ghi con trỏ và chỉ số

Có 2 thanh ghi con trỏ và 2 thanh ghi chỉ số:

- *Thanh ghi con trỏ ngăn xếp* SP (Stack Pointer) chứa địa chỉ đỉnh ngăn xếp (vùng nhớ đặc biệt, hoạt động theo nguyên tắc LIFO –

Last In First Out – vào sau ra trước) sử dụng cho việc lưu giữ tạm thời các dữ liệu hay địa chỉ khi gọi chương trình con, khi phục vụ ngắt v.v…giá trị nội dung của SP luôn luôn là phần offset của địa chỉ ngăn xếp kế tiếp

- *Thanh ghi con trỏ cơ sở* BP (Base Pointer) có chức năng chứa giá trị offset tính từ địa chỉ SS nhưng còn được sử dụng để truy cập dữ liệu bên trong ngăn xếp
- *Các thanh ghi chỉ số nguồn* DI và *thanh ghi chỉ số đích* SI (Destination Index và Source Index) được dùng để lưu giữ các thành phần offset đối với những vùng dữ liệu được cất trong đoạn dữ liệu. Hai nội dung của hai thanh ghi này liên kết với nội dung thanh ghi đoạn DS để tạo ra địa chỉ nguồn và địa chỉ đích của vùng nhớ.

II.1.4 Các chế độ làm việc MIN/MAX

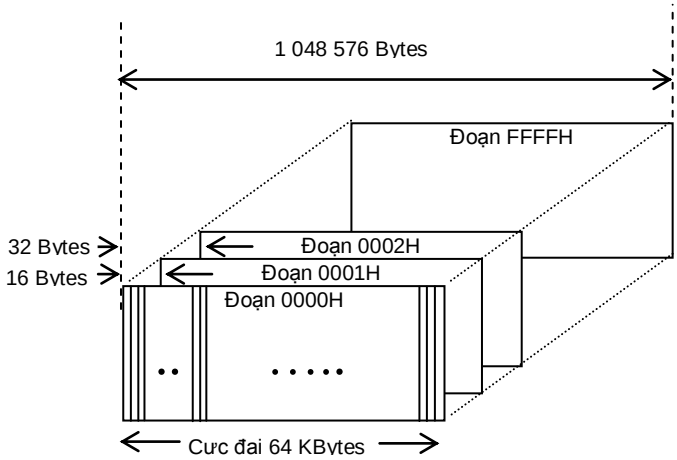
µP8086 có hai chế độ làm việc: chế độ MIN và chế đọ MAX. Chân số 33 của µP8086 được coi như là *chân bẫy* (trap pin) cho µP8086 trong việc định nghĩa chế độ làm việc. Những mạch phụ trợ cần thiết cho hai chế độ làm việc không thể thoả mãn với hệ thống 40 chân của CPU loại này, vì vậy một số chân sẽ đảm nhiệm những chức năng khác khi được xác định cho một chế độ, phụ thuộc vào cách nối chân MN/$\overline{MX}$. Khi được nối với GND (mức điện áp 0V), µP8086 chuyển đổi các chân từ 24 đến 31 sang chế độ MAX. Một mạch phụ điều khiển BUS 8288 sẽ giải mã các tín hiệu trạng thái $\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$ để tạo ra các tín hiệu định thời và các tín hiệu điều khiển tương thích với cấu trúc MULTIBUS^{TM} trong các hệ thống máy tính. Khi được nối lên mức điện áp nguồn nuôi (mức Vcc +5V) tự µP8086 tạo các tín hiệu điều khiển BUS trên các chân từ 24 đến 31 như được ghi trong ngoặc ở Hình II. 14.

II.1.5 Phương thức quản lý bộ nhớ, các mode địa chỉ

a. Phương thức quản lý bộ nhớ:

BUS địa chỉ của µP8086 có độ dài 20 bits, do vậy có thể quản lý được 2^{20} = 1M ô nhớ (Mỗi tổ hợp "0" hoặc "1" của các bit trong 20 bits địa chỉ xác định vị trí của một ô nhớ). Vì một ô nhớ trong hệ Vi xử lý là 1 Byte, nên nói cách khác, *không gian nhớ* mà µP8086 quản lý được là 1Mbyte.

Các thanh ghi của µP8086 chỉ có độ dài 16 bits, nên nếu dùng một thanh ghi để đánh địa chỉ thì chỉ quản lý được 2^{16} ô nhớ, tức là 64KB. Để giải quyết vấn đề quản lý 1MByte, tức là 1.048.576 Bytes, µP8086 sử dụng BUS địa chỉ có độ rộng 20 bits thông qua nội dung của hai thanh ghi 16 bits để đánh địa chỉ của bộ nhớ theo phương thức sau:

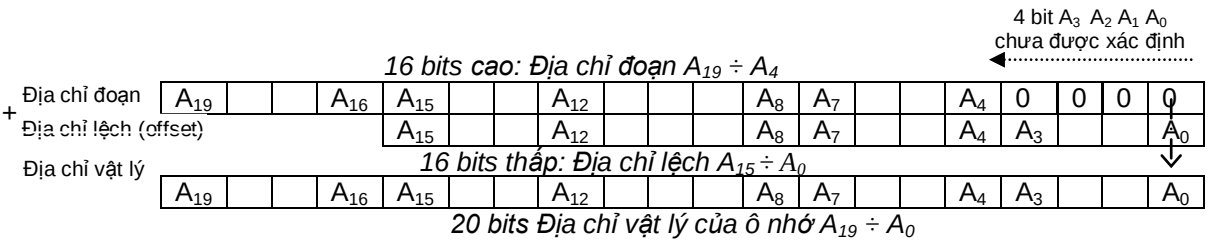


Hình II.20 Cách chia đoạn nhớ trong µP8086

Bằng cách lập chương trình, không gian địa chỉ được chia thành các *đoạn* (segment) nhớ với kích thước cố định là 64Kbytes gọi là một *đơn vị logic* của bộ nhớ. Mỗi đoạn gồm các ô nhớ liên tiếp, độc lập và được định vị tách rời nhau. Mỗi đoạn được người lập trình gán cho một *địa chỉ đoạn*, là địa chỉ ô nhớ đầu tiên của đoạn đó, còn được gọi là *địa chỉ nền*.Giá trị của các địa chỉ đoạn liền kề cách nhau tối thiểu là 16 Bytes. Các đoạn có thể kế cận, tách rời, phủ lấp nhau. Bên trong đoạn sẽ sử dụng các *giá trị lệch* (offset), tức là khoảng cách từ địa chỉ đoạn đến ô nhớ nằm trong đoạn. Một cặp giá trị địa chỉ đoạn và giá trị lệch, [*segment*]:[*offset*], được gọi là *địa chỉ logic*. Địa chỉ logic cho phép định vị chính xác một Byte nhớ trong không gian địa chỉ. Địa chỉ đoạn được chứa trong các thanh ghi đoạn, giá trị dịch chuyển được chứa trong các thanh ghi đa năng, con trỏ hoặc chỉ số.

Về bản chất, thanh ghi đoạn chứa 16 bits cao của 20 bits dịa chỉ, giá trị dịch chuyển là 16 bit thấp, và sự lệch nhau 4 bits đã được đơn vị địa chỉ của BIU giải quyết như trình bày trong hình II. 18: Dịch trái thanh ghi đoạn 4 bits (tương đương phép nhân với 16, cộng với giá trị dịch chuyển offset trong thanh ghi đa năng để tính địa chỉ vật lý của ô nhớ. Công thức tương ứng phép "dịch trái và cộng" có thể trình bày như sau:

$$\textbf{\textit{Địa chỉ vật lý}} = \textbf{\textit{10H}} \times (\textbf{\textit{segment}}) + (\textbf{\textit{offset}})$$



Hình II. 18 Mô tả cách tính địa chỉ vật lý của một vị trí nhớ

Thanh ghi đoạn là một thanh ghi 16 bits, có nhiệm vụ xác định đoạn của ô nhớ, còn thanh ghi đa năng cũng là một thanh ghi 16 bits. Vậy thanh ghi đoạn có thể định được 2^{16} = 65.536 đơn vị (64K) đoạn nhớ và mỗi đoạn có 64Kbytes. Vậy Vi xử lý µP8086 có thể định địa chỉ tới 64K x 64Kbytes = 4Gbytes nhớ.

Thanh ghi đoạn mã CS xác định đoạn nhớ chương trình mà lệnh kế tiếp sẽ được lấy để thực hiện, thanh ghi con trỏ IP chứa địa chỉ offset của lệnh kế tiếp. Cặp CS:IP tạo nên địa chỉ logic của lệnh kế tiếp trong tuần tự thực hiện chương trình. Các từ lệnh của họ 80x86 có thể có độ dài từ 1 byte đến tối đa là 15 bytes. Khi lệnh được thực hiện, giá trị của con trỏ IP do vậy sẽ tăng lên đúng bằng số Bytes của từ lệnh. Cần nhớ rằng nội dung của thanh ghi *con trỏ lệnh* IP cùng với nội dung thanh ghi đoạn CS xác định địa chỉ của ô nhớ lệnh tiếp theo trong tuần tự thực hiện chương trình.

b. Các mode đánh địa chỉ

1. Định vị thanh ghi (register addressing): Toán hạng được truy xuất nằm ngay trong thanh ghi của CPU.

Thí dụ MOV AX,BX ; chuyển nội dung của toán hạng nguồn (nội dung của thanh ghi) BX vào toán hạng đích AX. Nội dung thanh ghi BX vẫn được giữ nguyên.

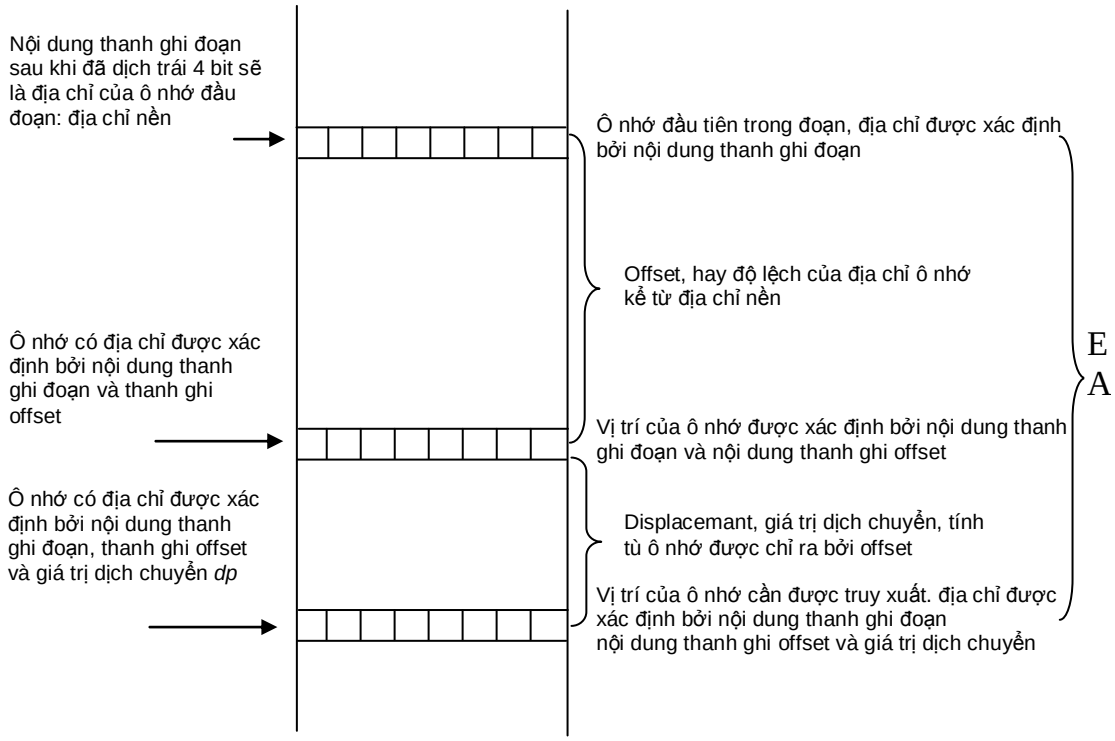
2. Định vị tức thời (immediate addressing): Toán hạng tức thời là dữ liệu 8 hay 16 bits nằm ngay trong lệnh, có thể dùng làm toán hạng nguồn hay hằng số. Toán hạng tức thời được lưu giữ ngay trong đoạn mã của bộ nhớ, ngay sau mã lệnh, nó được lấy ra cùng với lệnh và ghi vào hàng đợi lệnh PQ, do vậy được truy xuất nhanh hơn so với truy xuất toán hạng từ bộ nhớ.

Thí dụ MOV AL, 12H ; nạp số 12H vào thanh ghi AL

3. Các kiểu định vị bộ nhớ

Khác với hai kiểu định vị trên, toán hạng trong đoạn nhớ dữ liệu được CPU *truy xuất qua BUS dữ liệu*. Biết rằng, địa chỉ vật lý của ô nhớ được tính từ nội dung thanh ghi đoạn và offset theo cách trình bày trong Hình II. 18. Giá trị offset mà đơn vị thực hiện lệnh EU tính cho một toán hạng trong đoạn nhớ được gọi là *địa chỉ hiệu dụng* EA (effective address) của toán hạng. Đơn vị thực hiện lệnh có thể tính EA dựa vào cách mô tả địa chỉ trong phần toán hạng nguồn của lệnh. Ngoài giá trị trực tiếp, hoặc nội dung thanh ghi cơ sở hay thanh ghi chỉ số, khi cần còn có thể có một giá trị số có độ dài 8 bits hay

16 bits được cộng thêm vào gọi là *giá trị dịch chuyển dp* (displacement). Xem Hình II.21



Hình II. 21 Mô tả cách xác định địa chỉ vật lý của ô nhớ càn truy xuất

Cụ thể như sau:

– *Định vị trực tiếp* (direct addressing): Toán hạng chứa địa chỉ là một số nằm ngay trong lệnh. Địa chỉ đoạn hiện tại nằm trong thanh ghi đoạn DS

Thí dụ MOV CX,[1435H] ; chuyển nội dung ô nhớ có địa chỉ offset bằng 1435H trong đoạn số liệu hiện tại vào thanh ghi CX

– *Định vị gián tiếp thanh ghi* (register indirect): địa chỉ hiệu dụng EA là nội dung của một trong các thanh ghi BX, BP, SI hoặc DI

Thí dụ MOV AX, [SI] ; chuyển nội dung của ô nhớ trong đoạn số liệu hiện tại có địa chỉ offset là nội dung thanh ghi SI

– *Định vị cơ sở* (based addressing): EA là tổng của nội dung thanh ghi BX hoặc BP và giá trị dịch chuyển *dp* nếu có

Thí dụ MOV [BX] + dp, AL ; chuyển nội dung thanh ghi AL và ô nhớ có địa chỉ offset bằng tổng của nội dung thanh ghi BX và giá trị dịch chuyển dp

– *Định vị chỉ số* (indexed addressing): EA là tổng của nội dung thanh ghi SI hoặc DI và giá trị dịch chuyển *dp* nếu có

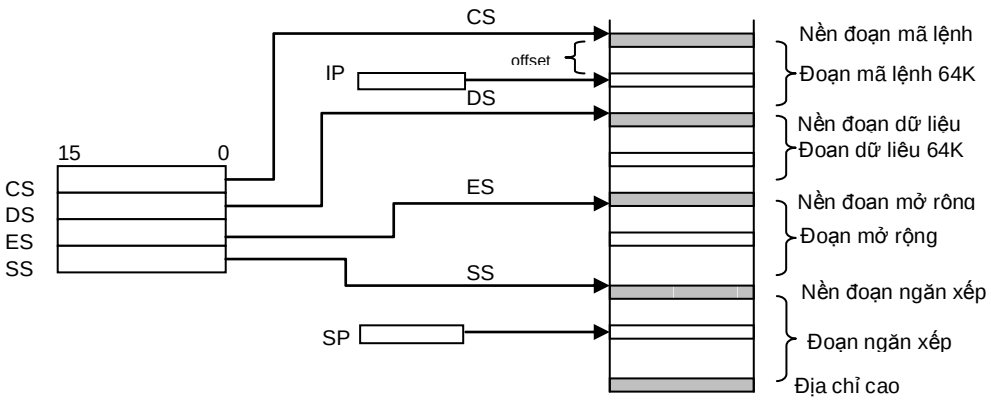
Thí dụ MOV AL,[SI] + dp ; chuyển nội dung ô nhớ có địa chỉ offset bằng tổng của nội dung thanh ghi SI và giá trị dịch chuyển dp vào thanh ghi AL

– *Định vị chỉ số và cơ sở* (indexed addressing): EA là tổng của nội dung các thanh ghi cơ sở, thanh ghi chỉ số và giá trị dịch chuyển *dp* nếu có

Thí dụ MOV AH,[BX][SI] + dp ; chuyển nội dung ô nhớ có địa chỉ offset bằng tổng của nội dung thanh ghi BX, thanh ghi SI và giá trị dịch chuyển dp vào thanh ghi AH

– *Định vị chuỗi* (string addressing): dùng riêng cho xử lý chuỗi. CPU sẽ tự động sử dụng các thanh ghi chỉ số nguồn SI và thanh ghi chỉ số đích DI để chỉ đến các byte kế tiếp

Thí dụ MOVS ; di chuyển chuỗi, nguồn tại vùng nhớ có địa chỉ đầu là DS : SI, đích là vùng nhớ có địa chỉ đầu DS : DI.



Hình II.22 Quản lý bộ nhớ theo đoạn (segment)

II.1.6 Phương thức đánh địa chỉ thiết bị ngoại vi

Có hai phương thức cơ bản đánh địa chỉ thiết bị ngoại vi:

– Định địa chỉ tách biệt (isolated I/O address): Các tín hiệu điều khiển phải được phân biệt đối với các thao tác ghi/đọc bộ nhớ và ghi/đọc thiết bị ngoại vi. Trong hệ Vi xử lý µP8085, tổ hợp các tín hiệu $\overline{RD}$, $\overline{WR}$ và IO/$\overline{M}$ sẽ được giải mã để tạo ra các tín hiệu đọc/ghi riêng cho bộ nhớ ($\overline{MEMR}$ và $\overline{MEMW}$) và riêng cho thiết bị ngoại vi ($\overline{IOR}$ và $\overline{IOW}$). Đối với họ 80x86, đó là việc sử dụng chip điều khiển BUS (BUS Controller 8288) để giải mã tổ hợp các tín hiệu nhịp đồng hồ CLK, các tín hiệu trạng thái S2, S1 và S0 trong chế độ MAX thành các tín hiệu $\overline{MRDC}$, $\overline{MWTC}$, $\overline{IORC}$ và $\overline{IOWC}$. Các mạch logic phụ trợ điều khiển truy nhập thiết bị ngoại vi trong hệ Vi xử lý có nhiệm vụ phát hiện các tín hiệu $\overline{IORC}$ và $\overline{IOWC}$ để thực hiện các thao tác vào/ra dữ liệu. Mạch logic này có nhiệm vụ giải mã địa chỉ thiết bị ngoại vi để tạo ra các tín hiệu cho phép truy nhập tới thiết bị cụ thể (thường được gọi là mạch *giải mã địa chỉ thiết bị ngoại vi*). Cũng cần nói thêm rằng *địa chỉ thiết bị ngoại vi* thực tế là *địa chỉ của một thanh ghi* trong thiết bị ngoại vi. Như vậy, việc trao đổi dữ liệu giữa CPU và thiết bị ngoại vi thực chất là trao đổi dữ liệu giữa CPU và thanh ghi trong *không gian thiết bị ngoại vi.* Các µP80x86 dành 16 dây địa chỉ thấp (A15 – A0) để quản lý một không gian 64K thiết bị ngoại vi.

– Định địa chỉ tuyến tính (Linear Addresing I/O), cũng còn gọi là định địa chỉ thiết bị ngoại vi theo bản đồ nhớ (Memory-Mapped I/O): Thanh ghi trong thiết bị ngoại vi được coi như một vị trí nhớ trong không gian nhớ, do vậy không sử dụng đến các tín hiệu điều khiển riêng cho việc trao đổi dữ liệu giữa CPU với thiết bị ngoại vi, mà sử dụng hoàn toàn chung cho bộ nhớ cũng như cho thiết bị ngoại vi. Đối với µP8085, tín hiệu phân biệt IO/$\overline{M}$ không cần thiết nữa, cũng như không cần giả mã các tổ hợp tín hiệu $\overline{S2}$, $\overline{S1}$ và $\overline{S0}$ đối với các trung tâm 80x86. Mọi thao tác trao đổi dữ liệu giữa CPU và các thanh ghi thiết bị ngoại vi đều được tiến hành như với một ô nhớ trong bộ nhớ.

II.1.7 Các mạch Multiplexer, mạch Decoder, mạch PLA

Các mạch Multiplexer, mạch Decoder hay mạch PLA là những mạch phụ trợ không thể thiếu của một hệ Vi xử lý. Thông thường, các mạch Decoder và mạch PLA (Programmable Logic Array) được thiết kế sẵn trên một chip, được sử dụng nhiều trong các mạch giải mã các tín hiệu điều khiển, giải mã địa chỉ của vùng nhớ hay địa chỉ thiết bị ngoại vi.

– *Các mạch Multiplexer (hoặc Coder)*

Mạch Multiplexer (còn gọi là mạch Coder)thường được xây dựng theo mục đích sử dụng, có khi rất phức tạp. Một trong những ví dụ là mạch thu nhận và mã hoá bàn phím (keyboard), được xây dựng trên cơ sở một chip Vi xử lý chuyên dụng, bao gồm cả phần cứng lẫn chương trình. Sơ đồ khối chức năng của một mạch Multiplexer được thể hiện trên Hình II.17. Các tín hiệu vào riêng rẽ x_1, x_2, x_3 …, x_n, qua xử lý sẽ tạo ra một tổ hợp nhị phân đầu ra $\{y_m y_{m-1} \ldots y_0\}$. Phần chuyển đổi từ một tín hiệu vào x_i thành tổ hợp ra $\{y_m y_{m-1} \ldots y_0\}$ được thực hiện nhờ mạch tổ hợp logic hoặc kết hợp với phần mềm chuyên dụng.

Hình II. 23 Sơ đồ khối một mạch Multiplexer (Coder)

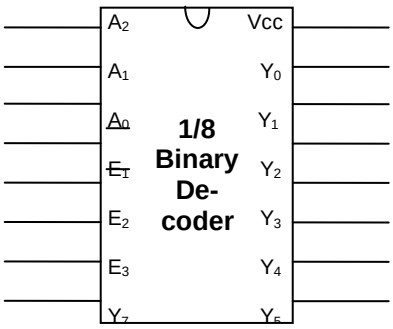
– *Mạch giải mã (Decoder)*

Các mạch giải mã thông thường 1/4, 1/8 được xây dựng như một chíp phụ trợ trong các hệ Vi xử lý. Có thể kể đến như mạch giải mã 1/16 SN74154, mạch giải mã 1/8 74138 v.v… Bảng chân lý của mạch giải mã 1/8 như sau:

E_3	E_2	E_1	A_2	A_1	A_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
0	x	x	x	x	x	1	1	1	1	1	1	1	1
x	1	x	x	x	x	1	1	1	1	1	1	1	1
x	x	1	x	x	x	1	1	1	1	1	1	1	1

Hình II. 25 Sơ đồ nối chân mạch giải mã nhị phân 1/8 và bảng chân lý

Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866



Vi mạch giải mã nhị phân 1/8 có sơ đồ nối chân như Hình II. 18. Khi vi mạch giải mã được "Enable", ứng với tổ hợp các giá trị $E_3E_2E_1$ = 100, và với bất kỳ tổ hợp nào của các giá trị $A_2A_1A_0$ đều có một lối ra có giá trị LOW. Ứng với lối ra này sẽ là một vị trí hoặc một vùng nhớ được chọn, hoặc một thiết bị ngoại vi. Đối với các vi mạch có chân CS (chip select), đây là tín hiệu chọn vỏ thích hợp.

– *Mạch PLA (Programmable Logic Array)*

Mạch PLA thực chất là một chip nhớ ROM được ghi sẵn theo một quy luật nào đó theo phương thức giải mã một tổ hợp nhị phân ở đầu vào. Có nghĩa là ứng với một ô nhớ là một tổ hợp giá trị theo một quy luật giải mã đầu vào, mà đầu vào đây chính là địa chỉ của ô nhớ đó. Các mạch PLA thích hợp với những nơi cần sử dụng bộ giải mã với số lượng đầu vào lớn hơn 3.

II.1.8 Vài nét về lập trình hợp ngữ

Hợp ngữ (Assembler) là một công cụ rất mạnh được sử dụng trong việc phát triển mã lệnh của các hệ Vi xử lý và máy vi tính. Hợp ngữ là chương trình dịch các lệnh gợi nhớ (Mnemonics) và các ký hiệu (symbols) thành mã máy cho các hệ vi xử lý và máy vi tình thực hiện. Cần phân biệt rằng hợp ngữ là một chương trình, chứ không phải là một phần của phần cứng.

Dữ liệu vào của hợp ngữ là tập các lệnh gợi nhớ, và dữ liệu ra của hợp ngữ chính là các tập các byte mã máy nhị phân, mã thực thi được đánh địa chỉ chính xác trong không gian nhớ chương trình.

Dữ liệu vào được gọi là mã nguồn (source code), dữ liệu ra được gọi là mã thực thi hoặc mã đối tượng (object code). Quá trình mã nguồn được dịch thành mã đối tượng được gọi là assembly. Công cụ phần mềm thực hiện quá trình này gọi là hợp ngữ (assembler). Có thể thấy rất dễ dàng rằng: viết một lệnh MOV A,M dễ nhớ hơn rất nhiều so với mã hexa của lệnh này: $7E_H$ hoặc mã nhị phân $0111\ 1110_B$.

Hiện có hai loại chương trình hợp ngữ đang được sử dụng rộng rãi: Hợp ngữ thường trú (Resident Assemblers) - được cài đặt ngay trong hệ thống, và hợp ngữ chuyển đổi (Cross Assemblers) không được cài đặt ngay trong hệ thống, mà là trong một máy chủ khác. Mã chương trình do máy chủ tạo ra từ hợp ngữ không thể chạy được trên máy chủ.

Ngoài ra còn tồn tại hai loại hợp ngữ khác là hợp ngữ tuyệt đối (Abssolute Assembler), và hợp ngữ tái định vị (Relocatable Assemblers), sẽ được giới thiệu trong các trình học hợp ngữ sau này.

II.3 Cấu trúc và tính năng của một số chip Vi xử lý hiện đại.

Trải qua mấy thập kỷ phát triển, công nghệ chế tạo các chip Vi xử lý đã có những bước tiến vũ bão. Đã xuất hiện nhiều kiểu cấu trúc chip Vi xử lý như CISC (Complete Instruction-Set Computer), RISC (Reduced Instruction-Set Computer), bộ xử lý scalar hay superscalar, Vi xử lý VLIW (Very Long Instruction Word), Vi xử lý Superpipelined, Vi xử lý Vector, và Vi xử lý biểu tượng (Symbolic µP), nhằm đáp ứng nhu cầu tạo nên những máy tính cực mạnh, những siêu máy tính, mainfram phục vụ những công việc tính toán lớn hay tạo ra các máy tính xử lý song song.

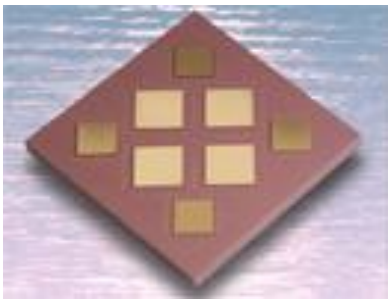
Đối với họ x86, đã có các trung tâm i486 với cấu trúc RISC, tập lệnh rút gọn với tốc độ xử lý tăng nhanh đáng kể. Đó là những trung tâm xử lý 32 bits thực sự. Không gian địa chỉ vật lý và không gian bộ nhớ ảo được quản lý bởi 32 bits địa chỉ, lên đến 4Gbytes. Ngoài ra, các bộ đồng xử lý toán cũng được tích hợp tạo nên sức mạnh đáng kể. Các trung tâm này đã được sử dụng để tạo nên những máy tính xử lý song song với cấu trúc hiện đại và khả năng tính toán lớn cho phép giải những bài toán rất phức tạp.

Năm 1992, chip xử lý Pentium MMX ra đời, trong cấu trúc có tới hai đường ống song song (superscalar), hai khối số học và logic (ALU) cho phép thi hành hai lệnh máy trong một chu kỳ. BUS nội bộ của Pentium MMX là BUS 64 bits và 128 bits, tốc độ trao đổi dữ liệu với bộ nhớ do vậy được nâng cao đáng kể. Đặc biệt, chip Pentium có một vùng nhớ gọi là vùng đệm đích rẽ nhánh BTB (Branch Target Buffer) đối với 256 lệnh rẽ nhánh mới. Pentium cũng được tích hợp một bộ đồng xử toán học (*Mathematical Coprocessor*) với hiệu suất rất cao nhờ giải thuật nhanh hơn. Dù là loại Vi xử lý CISC, nhưng Pentium đã ứng dụng giống như các loại Vi xử lý RISC tốc độ cao: xử lý đường ống, cấu trúc superscalar và dự đoán rẽ nhánh.

Năm 1999, chip Pentium PIII ra đời với cấu trúc có thêm 70 lệnh cho truyền thông đa phương tiện, tốc độ xung nhịp đã vượt qua ngưỡng 1GHz. Tiếp theo đó là các chip Pentium PIV với tốc độ cao hơn hẳn và cấu trúc đa phân luồng.

IBM cho biết họ đang áp dụng những phương pháp sản xuất mới nhằm cho phép thiết bị xử lý dành cho máy chủ Power6 có thể chạy nhanh gấp hai lần so với hiện nay nhưng vẫn đảm bảo yêu cầu về nhiệt độ.

Theo thông tin từ Giám đốc kỹ thuật của IBM, họ không chỉ thu nhỏ kích cỡ các bóng bán dẫn (transistor) mà còn thay đổi phương thức hoạt động của silicon khi đặt lớp cách điện phía dưới một lớp silicon gồm khoảng 500 nguyên tử.



Chip IBM Power5. (*IBM*)

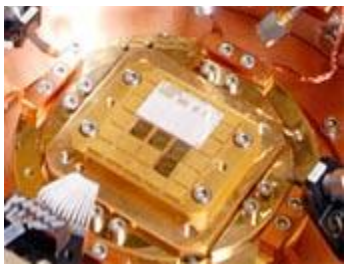
Những cải tiến đó khiến Power6, sản xuất theo công nghệ 65 nm và dự định ra mắt giữa năm 2007, đạt xung nhịp tới 4 - 5 GHz. IBM khẳng định Power6 sẽ cạnh tranh trực tiếp với sản phẩm của các đối thủ như Intel, AMD và Sun Microsystems.

Ngay khi Intel giới thiệu công nghệ 90 nm năm 2003, người ta nhận thấy nó gây ra tình trạng thất thoát năng lượng nghiêm trọng hơn những phương pháp sản xuất trước đó, khiến chip tỏa nhiệt ngay cả khi chúng không chạy hết công suất. Một trong những biện pháp khắc phục là tích hợp 2 lõi xử lý trên một chip đơn và giảm xung nhịp hoạt động để tăng khả năng vận hành cũng như tránh rắc rối do nhiệt độ cao. Ngược lại, Power6 được xây dựng để hoạt động ở xung nhịp cao chưa từng thấy nhưng vẫn tiêu thụ điện năng hiệu quả.

Hiện nay, Intel cũng đang nghiên cứu hai kỹ thuật sản xuất và thiết kế mới nhằm giảm lượng điện tiêu thụ trên bảng mạch hệ thống. Phương pháp thứ nhất cung cấp nguồn điện áp cho cả CPU và bộ nhớ đệm (cache) còn cách thứ hai sẽ tích hợp bộ điều hòa điện áp trên các transistor.

Thứ tư, 21/6/2006, 10:04 GMT+7

IBM phát triển chip 500 GHz



Chip SiGe ở chế độ đông lạnh có xung nhịp gấp 250 lần so với chip ĐTDĐ thông thường. Ảnh: *Marketwire*

"Big Blue" và Công ty Georgia Tech đã cùng chế tạo một chip có xung nhịp cao hơn 100 lần so với kỷ lục của thiết bị xử lý máy tính hiện nay, với điều kiện nó phải hoạt động ở nhiệt độ nghe có vẻ phi thực tế - 268,5°C.

Ở nhiệt độ trong phòng, chip này vẫn đạt tốc độ 350 GHz, tương đương 350 tỷ vòng/giây, nhanh hơn nhiều so với thiết bị xử lý máy tính tại thời điểm này (dao động từ 1,8 GHz đến 3,8 GHz).

Đây là một phần dự án khám phá tốc độ tối đa của các chip silicon-germani (SiGe). SiGe cũng giống như công nghệ chip silicon khác nhưng được tăng cường nguyên tố germani để nâng hiệu suất và giảm lượng điện tiêu thụ. Trên lý thuyết, SiGe có thể mở rộng tốc độ lên 1 terahertz (THz), tức một nghìn tỷ vòng mỗi giây.

Tuy nhiên, thêm thành phần germani đồng nghĩa với chi phí sản xuất tấm wafer tăng cao, do đó SiGe rất kén chọn thị trường. IBM đã bán ra hàng trăm triệu chip này từ năm 1998, nhưng chưa thể địch nổi khi so với con số hàng tỷ chip silicon mỗi năm nhờ sản lượng điện thoại di động.

Chip SiGe hiệu suất lớn sẽ được ứng dụng trong các hệ thống phòng thủ, phương tiện khám phá vũ trụ và thiết bị cảm ứng từ xa.

II.3.1 Cấu trúc chip Vi xử lý Pentium

Pentium là loại đơn vị xử lý trung tâm 32 bit và là loại đơn vị xử lý trung tâm đầu tiên sử dụng kỹ thuật ILP (Instruction Level Pararellism), kỹ thuật xử lý lệnh song song.

Kỹ thuật đường ống và kỹ thuật xử lý lệnh song song ILP

Một lệnh thường được xử lý qua năm giai đoạn:

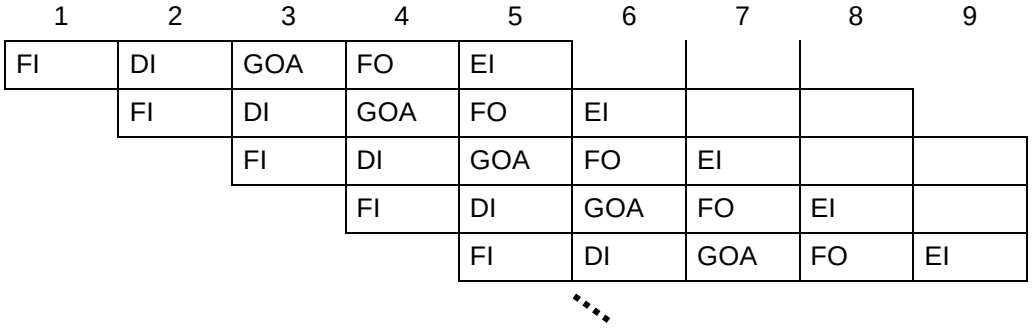
1. Nhập lệnh (FI Fetch the Instruction)
2. Giải mã lệnh (DI Decode the Instruction)
3. Tạo địa chỉ toán hạng (GOA Generate Operand Address)
4. Nhập toán hạng (FO Fetch Operands)
5. Thực hiện lệnh (EI Execute Instruction)

Với kỹ thuật xử lý lệnh thông thường, đơn vị xử lý trung tâm phải tuần tự thực hiện xong tất cả các giai đoạn thực hiện một lệnh, thường là sau 5 chu kỳ máy, rồi mới chuyển sang nhập và thực hiện lệnh tiếp theo. Để tăng tốc độ xử lý lệnh mà không nhất thiết phải tăng tần số nhịp của máy, người ta sử dụng các kỹ thuật khác như kỹ thuật xử lý lệnh kiểu đường ống (Pipeline) và kỹ thuật xử lý lệnh song song (ILP).

- Kỹ thuật xử lý lệnh kiểu đường ống (Pipeline)

Kỹ thuật xử lý lệnh kiểu đường ống được sử dụng trong các đơn vị xử lý trung tâm từ đời đơn vị xử lý trung tâm Intel 8086.

Chu kỳ máy



Đường ống tương tự như dây chuyền sản xuất nhiều công đoạn. ở dây chuyền sản xuất, mỗi công đoạn thực hiện một thao tác sản xuất. Sản phẩm được được chuyển và hình thành dần sau mỗi công đoạn sản xuất, cho đến khi được hoàn thành ở công đoạn cuối cùng. Trong kỹ thuật xử lý lệnh theo kiểu đường ống, việc thực hiện lệnh cũng được thực hiện qua 5 thao tác, mỗi thao tác được thực hiện trong một giai đoạn, giai đoạn nọ tiếp sau giai đoạn kia, cho đến khi thực hiện song lệnh. Với một đường ống 5-giai đoạn, tại mỗi chu kỳ máy có 5 bộ dữ liệu thuộc 5 giai đoạn xử lý được gửi vào đường ống và 5

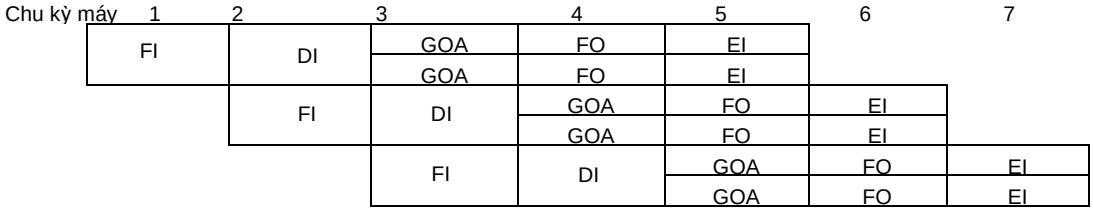
Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866

thao tác được thực hiện đồng thời, nhờ vậy mà sau mỗi chu kỳ máy lại có một lệnh được hoàn thành và một lệnh mới được nhập. Kỹ thuật đường ống với đường ống 5-giai đoạn cho phép tăng tốc độ thực hiện lệnh lên gấp 5 lần.

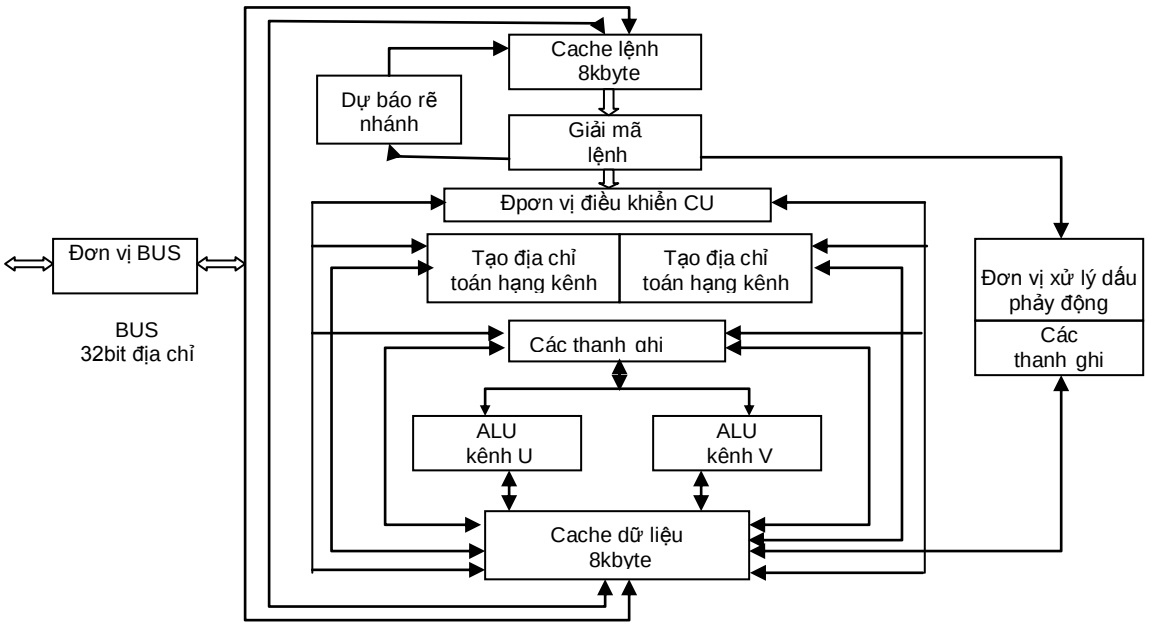
- Kỹ thuật ILP (xử lý lệnh song song)

Kỹ thuật ILP là kỹ thuật thiết kế đơn vị xử lý trung tâm và chương trình dịch nhằm làm tăng tốc độ các thao tác máy (như ghi-đọc bộ nhớ) và thực hiện các phép tính. Trong các kỹ thuật ILP có kỹ thuật *superscalar*, trong đó tại một chu kỳ máy nhiều lệnh được nhập và được thực hiện đồng thời trên nhiều đường ống khác nhau.

Pentium là loại đơn vị xử lý trung tâm được thiết kế theo kỹ thuật superscalar, trong đó hai lệnh được nhập và giải mã đồng thời. Pentium có hai đường ống thực hiện lệnh song song U và V. Quá trình thực hiện lệnh được mô tả như sau :



Cấu trúc của Pentium



Hình II.26 Cấu trúc trung tâm Vi xử lý Pentium

Pentium là đơn vị xử lý trung tâm loại 32bit và là đơn vị xử lý trung tâm loại CISC (Complex Instruction Set Computer) với đặc điểm: hệ lệnh

Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866

phức tạp, nhiều kiểu xác định địa chỉ, nhiều khuôn dạng lệnh và nhiều kích thước lệnh khác nhau.

Pentium có BUS địa chỉ trong và ngoài 32 bit, BUS dữ liệu trong và ngoài 64 bit. Pentium có hai cache 8 Kbyte độc lập : một cache 8 Kbyte 2 cổng dành cho dữ liệu và một cache 8 Kbyte chứa lệnh. Pentium có hai đường ống thực hiện lệnh song song U và V, và 2 đơn vị số học-logic ALU. Pentium có một đường ống riêng thực hiện các lệnh dấu phẩy động và một bộ đồng xử lý dấu phẩy động FPU được tích hợp trong chíp.

Pentium có các thanh ghi sau :

Các thanh ghi hệ thống :

- Các thanh ghi điều khiển 32 bit: CR0, CR1, CR2, CR3
- Các thanh ghi hệ thống quản lý bộ nhớ :

GDTR: Thanh ghi bảng mô tả toàn cục (Global Descriptor Table Register)

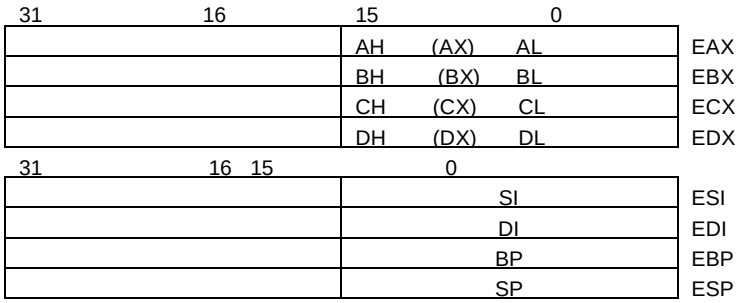
LDTR: Thanh ghi bảng mô tả cục bộ (Local Descriptor Table Register)

IDTR: Thanh ghi bảng mô tả ngắt (Interrupt Descriptor Table Register)

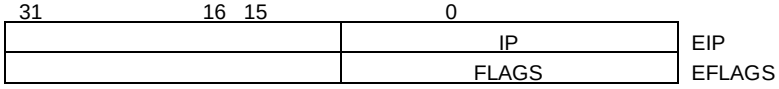
TR: Thanh ghi nhiệm vụ (Task Register)

- Các thanh ghi đoạn 16 bit :

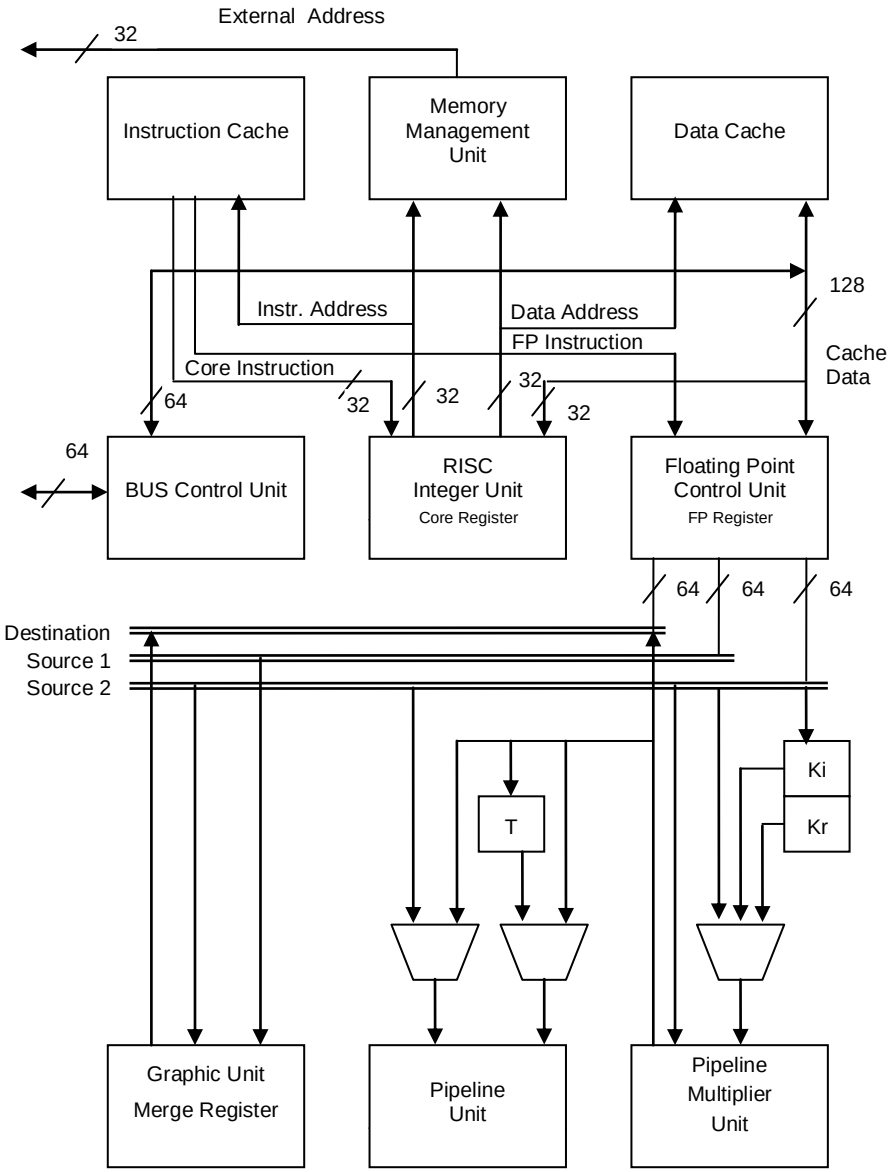
- Các thanh ghi đa năng 32 bit :



- Con trỏ lệnh EIP và thanh ghi cờ EFLAGS 32 bit :

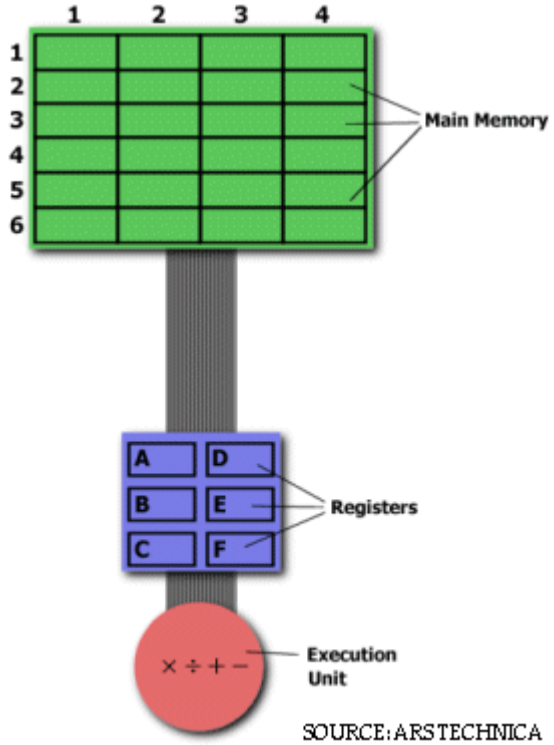


Hình II.27 Các thanh ghi và cấu trúc bên trong trung tâm Vi xử lý Pentium


External Address
32
Instruction Cache
Memory Management Unit
Data Cache
128
Instr. Address
Data Address
FP Instruction
Cache Data
Core Instruction
64
32
32
32
32
64
BUS Control Unit
RISC Integer Unit
Core Register
Floating Point Control Unit
FP Register
64
64
64
Destination
Source 1
Source 2
Ki
T
Kr
Graphic Unit
Merge Register
Pipeline Unit
Pipeline Multiplier Unit

II.3.2 Cấu trúc RISC, CISC

RISC: (Reduced Instruction-Set Computer)
CISC: (Complete Instruction-Set Computer)



Cách đơn giản nhất để có thể khảo sát những ưu nhược điểm của kiến trúc RISC là so sánh việc thực hiện một phép toán đối với loại kiến trúc CISC trước đây. Giả sử ta phải thực hiện một lệnh nhân hai toán hạng được lưu giữ trong bộ nhớ.

Hình II. … mô tả tổ chức của một máy tính. Bộ nhớ được tạo từ các ô nhớ từ 1:1 (hàng 1: cột 1) đến 6:4 (hàng 6 : cột 4). Khối thực hiện lệnh có nhiệm vụ thực hiện các lệnh tính toán x (nhân), ÷ (chia), + (cộng) và – (trừ). Tất nhiên, khối thực hiện tính toán chỉ có thể làm việc với các dữ liệu (toán hạng) đã được chứa sẵn ở một trong các thanh ghi A,B,C,D,E hoặc F. Giả sử ta phải tìm tích 2 số, số thứ nhất được chứa ở ô 2:3 và số thứ hai ở ô 5:2, kết quả sẽ được lưu lại vào ô 2:3. Bây giờ ta sẽ tiếp cận cách giải quyết vấn đề trên hai loại CPU, CISC và RISC.

Trên CPU CISC: ưu tiên hàng đầu của loại CPU này là hoàn thiện một công việc với ít lệnh nhất có thể. Điều này có thể thực hiện nhờ vào việc xây dựng một phần cứng CPU có khả năng *hiểu được và thực hiện được* một chuỗi các tác nghiệp. Trong trường hợp cụ thể này, CISC sẽ có một lệnh xác định duy nhất, tạm gọi là MULT, mà khi thực hiện, lệnh sẽ nạp hai giá trị toán hạng vào 2 thanh ghi sau đó thực hiện phép nhân rồi ghi kết quả vào một

thanh ghi tương ứng. Như vậy công việc sẽ được thể hiện bằng một lệnh như sau:

MULT 2:3,5:2

Lệnh MULT là một lệnh hoàn thiện (complex). Lệnh làm việc trực tiếp trên băng nhớ của máy tính, chứ không cần người lập trình phải dùng lệnh gọi hay nạp nội dung, ghi nội dung vào ô nhớ. Lệnh rất gần với ngôn ngữ bậc cao. Giả sử ta gọi "*a*" là giá trị của toán hạng trong ô nhớ 2:3 và "*b*" là giá trị toán hạng trong ô nhớ 5:2, thì lệnh tương ứng trong ngôn ngữ C là "*a = a*b"*.

Ưu điểm lớn nhất của hệ thống CISC là chương trình dịch phải làm rất ít việc khi dịch một chương trình, hay một lệnh của ngôn ngữ bậc cao sang ngôn ngữ máy. Vì độ dài của mã lệnh rất nhỏ, nên hệ thống cũng cần ít RAM hơn để ghi nhớ lệnh. Dĩ nhiên, việc thiết kế cấu trúc loại CISC đặc biệt sẽ phải tích hợp các lệnh hoàn thiện bằng phần cứng.

Trên CPU RISC: Các CPU loại RISC chỉ sử dụng các lệnh (Instruction) có thể thực hiện được trong một chu kỳ xung nhịp. Như vậy lệnh MULT được mô tả ở phần trên phải được chia thành 3 lệnh nhỏ hơn: "LOAD" chuyển dữ liệu (toán hạng) từ ô nhớ vào thanh ghi; "PROD" thực hiện phép nhân hai toán hạng được lưu giữ trong các thanh ghi, và lệnh "STORE" sẽ thực hiện việc chuyển kết quả tính toán ghi vào ô nhớ. Để thực hiện được phép nhân hai toán hạng, người lập trình phải mã hoá thành 4 lệnh như sau:

LOAD A, 2:3
LOAD B, 5:2
PROD A, B
STORE 2:3, A

Có thể thấy rằng với cấu trúc RISC, không thuận lợi lắm cho hoàn thành phép toán nhân hai số vì phải viết nhiều dòng lệnh hơn, cần nhiều RAM hơn để lưu giữ các lệnh mức assembly. Chương trình dịch cũng phải trhực hiện nhiều việc hơn để chuyển đổi các lệnh của ngôn ngữ bậc cao sang mã máy.

Thế nhưng, chiến lược RISC đã mang đến nhiều thuận lợi quan trọng. Vì mỗi lệnh chỉ cần một chu kỳ xung nhịp để thực hiện, toàn bộ chương trình cũng sẽ chỉ cần số chu kỳ xung nhịp như khi thực hiện lệnh MULT ở hệ thống CISC. Nhưng kiến trúc RISC với hệ lệnh rút gọn cần ít linh kiện và không gian cho mạch tích hợp, bỏ qua được các thanh ghi đa năng. Hơn nữa, mỗi lệnh chỉ thực thi trong một chu kỳ xung nhịp nên việc tổ chức đường ống cũng đơn giản hơn nhiều.

Việc tách lệnh "LOAD" và lệnh "STORE" đã đơn giản hoá đáng kể khối lượng công việc CPU phải thực hiện. Sau khi thực hiện lệnh MULT ở cấu trúc CISC, CPU tự động xoá nội dung các thanh ghi. Nếu một toán hạng nào đó còn tếp tục được sử dụng cho lệnh tiếp theo, CPU phải nạp lại. Ở cấu trúc RISC, nội dung của toán hạng vẫn được giữ lại cho đến khi một giá trị

mới được nạp vào.

Cuối cùng, để so sánh một cách toàn diện hơn, công thức sau được dùng để đánh giá khả năng tính toán, xử lý của các loại CPU:

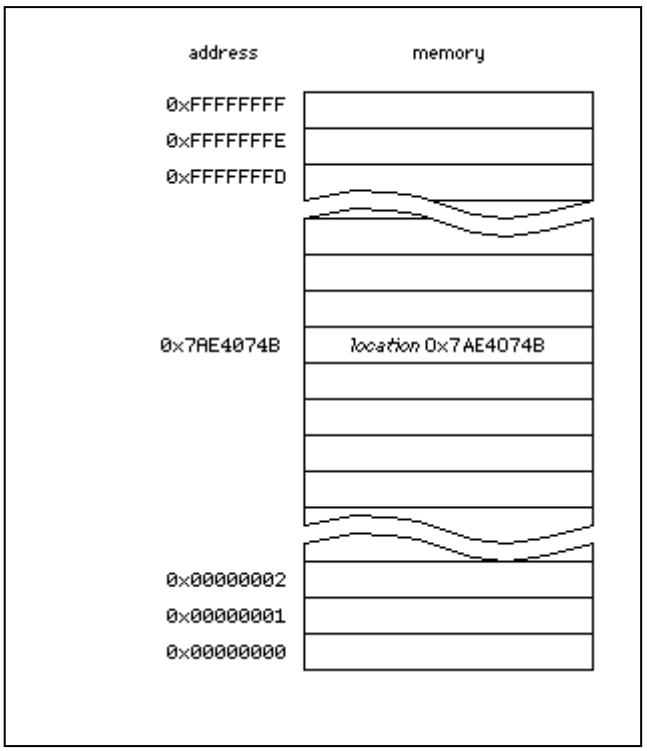
$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \text{ x } \frac{\text{cycles}}{\text{instruction}} \text{ x } \frac{\text{instructions}}{\text{program}}$$

CISC cố gắng giảm số lệnh trong một chương trình, hy sinh số chu kỳ thực hiện một lệnh trong khi RISC theo chiến lược ngược lại.

Một số thông tin thú vị cho độc giả: Chỉ các chip họ x86 vẫn trung thành với kiến trúc CISC, dĩ nhiên không được thông dụng lắm vì những lý do khác: Trước hết do sự phát triển vũ bão của công nghệ tích hợp mạch, trong công nghệ sản xuất linh kiện điện tử. Sự giảm giá đến mức khó hiểu của bộ nhớ RAM cũng làm đảo lộn cách nhìn nhận những nhược điểm của các CPU theo kiến trúc RISC. Giá 1Mbyte RAM năm 1977 là khoảng 5000USD, nhưng đến năm 1994 là khoảng 6USD, còn tại thời điểm này (2005) là khoảng hơn 0,2 USD! Công nghệ chương trình dịch (compiler technology) cũng trở nên hoàn thiện hơn nên CPU loại RISC cùng với bộ nhớ RAM dung lượng lớn và công nghệ phần mềm đã trở thành lý tưởng hơn nhiều đối với các hãng sản xuất máy tính.

II.3.3 Quản lý bộ nhớ

Địa chỉ (address) là phương thức duy nhất để "xác định *vị trí* (location)" của một ô nhớ trong "*không gian địa chỉ*" (address space).



Địa chỉ được thể hiện bằng một số nguyên nhị phân không dấu và được lưu giữ trong các thanh ghi chuyên dụng và thanh ghi đa năng với những kỹ thuật hoàn thiện. Địa chỉ được giải mã bằng phần cứng để truy xuất đến một vị trí nhớ trong các khối nhớ vật lý, ví dụ bộ nhớ RAM hoặc ROM hay trong một nguồn nhớ được bản đồ hoá (memory mapped resource).

Hình bên biểu diễn cách nhìn tổng quát về địa chỉ, không gian địa chỉ và vị trí nhớ trong

kiến trúc máy tính 32 bit. Có thể thấy địa chỉ như là một con trỏ (pointer), một số nguyên nhị phân tham chiếu đến một đối tượng hay một vị trí nhớ (ô nhớ). Dĩ nhiên, để tạo ra được một con trỏ, các kỹ thuật như phân đoạn (segment), sử dụng độ lệch (offset) và giá trị dịch chuyển (displacement) được sử dụng và được tạo nhờ đơn vị giao diện BUS (BIU) trong các CPU.

Không gian địa chỉ là tập tất cả các địa chỉ, cũng có thể hình dung như là một hàm riêng tham chiếu đến các ô nhớ. Thông thường, địa chỉ bắt đầu từ 0 (zero) cho đến 2^N-1, trong đó N là độ rộng của BUS địa chỉ (16, 20, 24, 32 hoặc 64). Không gian này có thể không chính xác với kiến trúc phân đoạn.

Trong các hệ thống hiện đại, phần lớn không gian địa chỉ có thể được dữ trữ nhờ kiến trúc của hệ điều hành, hoặc tạm thời không được bản đồ hoá. Những vấn đề liên quan độc giả có thể tìm thấy trong các tài liệu về không gian bộ nhớ ảo và không gian bộ nhớ vật lý.

II.3.4 Bộ nhớ cache

Cache là cơ chế nhớ tốc độ cao đặc biệt. Cache có thể sử dụng như một vùng nhớ dữ trữ trong bộ nhớ chính nhưng với những chip nhớ tốc độ cao. Có hai loại bộ nhớ cache được sử dụng chung trong máy PC, *memory caching* và *disk caching.*

Memory cache còn được gọi bộ nhớ cache hay RAM cache, sử dụng RAM tĩnh (SRAM) tốc độ cao. Rất hiệu quả vì nhiều chương trình truy nhập các dữ liệu hoặc lệnh thông qua vùng nhớ này. Bằng cách lưu giữ dữ liệu và lệnh trong cache, tốc độ truy nhập bộ nhớ được nâng cao. Cũng có một loại memory cache được tích hợp trực tiếp trong CPU như ở các CPU 80486 (8KB), ở Pentium là 16KB. Chúng được gọi là cache nội bộ (*Internal cache*), hay cache mức 1 (L1). Các PC còn hỗ trợ cache ngoài (*External cache*), còn gọi là cache L2, là bộ nhớ được dùng trung gian giữa CPU và bộ nhớ chính DRAM.

Disk cache làm việc giống như nguyên lý của cache nhớ, nhưng thay vì sử dụng SRAM, cache đĩa sử dụng DRAM như bộ nhớ chính. Phần lớn dữ liệu được truy xuất từ đĩa được lưu giữ trong các vùng nhớ đệm. Mỗi khi chương trình truy xuất đĩa, thông thường nó kiểm tra xem, các dữ liệu đó đã

được lưu vào vùng cache đĩa hay chưa. Cache đĩa đóng vai trò rất quan trọng trong việc nâng cao tôc độ truy xuất, vì truy xuất một byte dữ liệu trong RAM có thể nói nhanh hơn gấp ngàn lần truy xuất một byte dữ liệu từ các ổ đĩa. Khi dữ liệu được tìm thấy trong bộ nhớ cache, tức là *cache hit*, và hiệu suất của cache được đánh giá bằng *hit rate*. Hầu hết các hệ thống cache đều sử dụng kỹ thuật *smart caching*, có nghĩa là hệ thống luôn luôn ghi nhận một số loại dữ liệu thường được sử dụng nhất. Chiến lược xác định các thông tin nào được lưu giữ vào trong bộ nhớ cache là vấn đề được đặc biệt quan tâm trong khoa học máy tính.

II.4 Single-Chip MicroComputer µC8051

II.4.1 Tổng quan

Ngoài các trung tâm vi xử lý họ x86, Int$_e$l® còn thiết kế và sản xuất các trung tâm Vi xử lý chuyên dụng phục vụ các mục đích đo lường và điều khiển tự động, phục vụ các ứng dụng đơn giản nhưng rất phổ biến khác. Các chip Vi xử lý loại này đã vượt ra ngoài khuôn khổ của một *trung tâm Vi xử lý đơn thuần*, trở thành một Vi máy tính (MicroComputer). Cũng có thể nhìn nhận rằng, các trung tâm Vi xử lý họ này là một *Vi máy tính* thực thụ, nếu nhìn nhận chip này theo quan điểm kiến trúc của ông tổ máy tính Von Neumann: Chip được trang bị thêm bộ nhớ chương trình (ROM hoặc EPROM) và bộ nhớ dữ liệu, cũng như các cổng vào/ra nối tiếp, vào/ra song song.

MCS-51 là họ vi điều khiển của Int$_e$l®. Các nhà sản xuất khác như Siemens, Advanced Micro Device, Fujitsu va Philip được cấp phép làm các nhà cung cấp các chip cuả họ MCS -51.

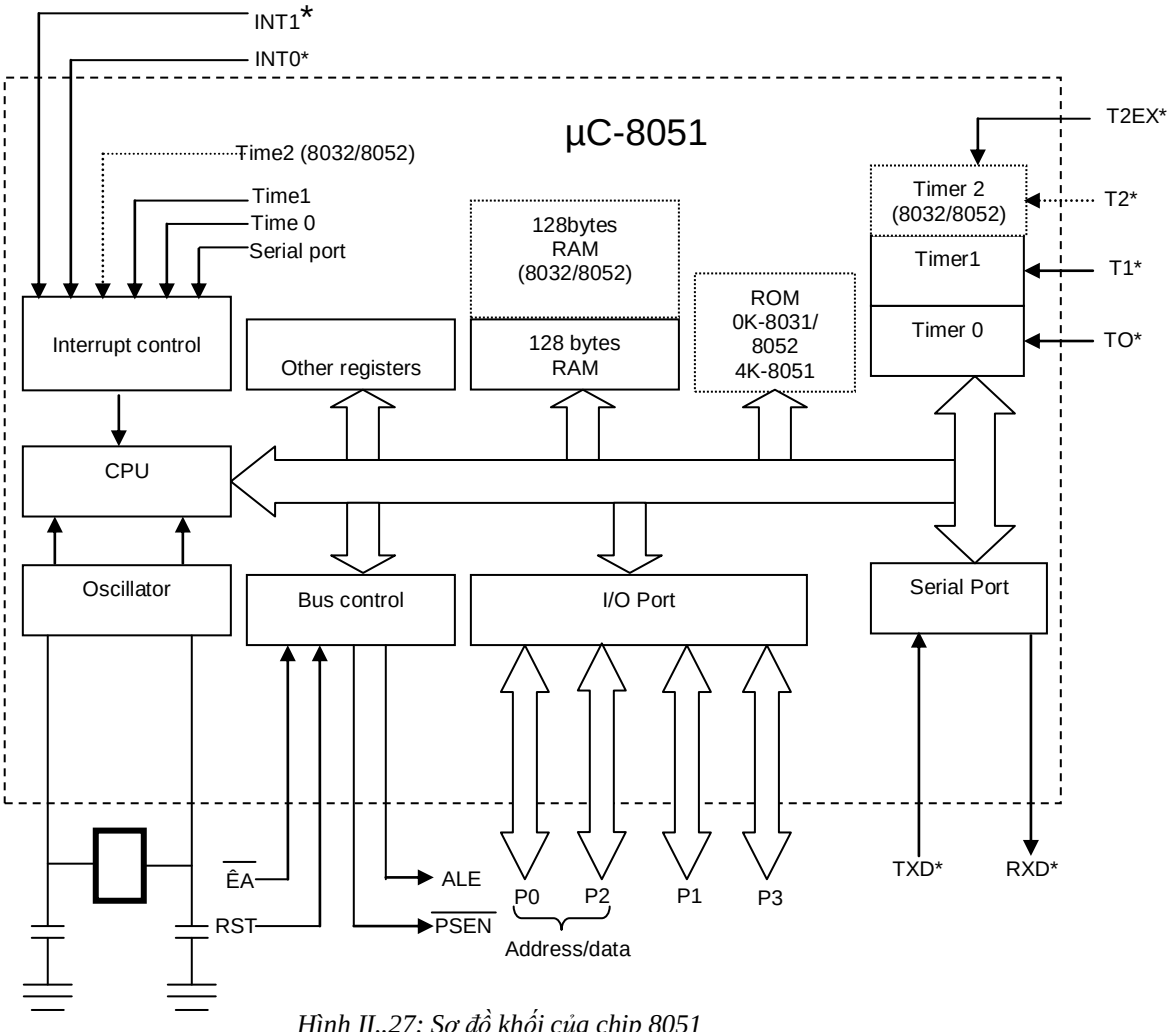
Vi mạch chủ yếu của họ MCS - 51 là chip µC8051, linh kiện đầu tiên của họ này được đưa ra thị trường. Chip µC8051 có các đặc trưng được tóm tắt như sau:

- 4 KB ROM và 128 byte RAM
- 4 *port* 8- bít, 32 lối vào/ra
- 2 bộ định thời 16 bít
- Mạch giao tiếp nối tiếp
- Không gian nhớ chương trình (mở rộng) ngoài 64K
- Không gian nhớ dữ liệu ngoài 64K
- Bộ xử lý bít (thao tác trên các bít riêng rẽ)
- 210 vị trí bit nhớ được định địa chỉ
- Nhân chia trong thời gian 4µs.

Các thành viên khác của họ MCS-51 có các tổ hợp ROM (EPROM), RAM trên chip khác nhau hoặc có thêm bộ định thời thứ ba. Mmỗi một chíp của họ MCS -51 đều có phiên bản CMOS tiêu thụ công suất thấp.

Dưới đây là thông số cơ bản của một số µC họ MCS-51:

Chip	*Bộ nhớ chương trình trên chip*	*Bộ nhớ dữ liệu trên chip*	*Các bộ nhớ định thời*
8051	4 K ROM	128 byte	2
8031	0 K	128 byte	2
8751	4 K EPROM	128 byte	2
8052	8 K ROM	256 byte	3
8032	0 K	256 byte	3
8752	8 K EPROM	256 byte	3



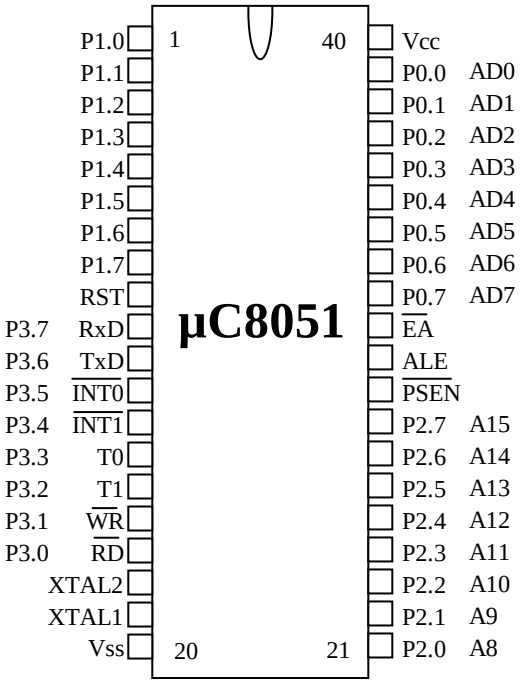
Hình II..27: Sơ đồ khối của chip 8051

Port 0 (các chân từ 32 đến 39) có hai công dụng. Trong các thiết kế tối thiểu, *port* 0 được sử dụng làm nhiệm vụ vào/ra. Trong các thiết kế lớn hơn có bộ nhớ ngoài, *port* 0 trở thành bus địa chỉ và bus dữ liệu dồn kênh.

Port 1 chỉ có một công dụng là vào/ra (các chân từ 1 đến 8), và dùng để giao tiếp với các thiết bị ngoại vi hoặc làm cổng vào/ra hoặc làm các lối vào cho mạch định thời thứ ba.

Port 2 (các chân từ 21 đến 28) có hai công dụng, hoặc làm nhiệm vụ vào/ra hoặc là byte địa chỉ cao của bus địa chỉ 16-bít cho các thiết kế có bộ nhớ chương trình ngoài hoặc các thiết kế có nhiều hơn 256 byte bộ nhớ dữ liệu ngoài.

Port 3 (các chân từ 10 đến 17) có hai công dụng. Khi không hoạt động vào/ra, các chân của *port* 3 có nhiều chức năng riêng (mỗi chân có chức năng riêng liên quan đến các đặc trưng cụ thể của 8051).



Hình II.28 Sơ đồ nối chân Chip µC8051

II.4.2 Mô tả cấu trúc và chức năng

Hình II.28 cho ta sơ đồ chân của chip 8051. Chức năng tóm tắt của từng chân như sau : 32 trong số 40 chân của 8051 có công dụng vào/ra, tuy nhiên 24 trong 32 đường này có hai mục đích, mỗi đường có thể hoạt động vào/ra hoặc hoạt động như một đường điều khiển hoặc như một đường địa chỉ/dữ liệu của bus địa chỉ/dữ liệu dồn kênh.

32 chân nêu trên hình thành 4 *port* 8-bít. Với các thiết kế yêu cầu tối thiểu bộ nhớ ngoài hoặc các thành phần bên ngoài khác ta có thể sử dụng *port* này làm nhiệm vụ vào/ra. 8 đường cho mỗi *port* có thể được xử lý như một đơn vị giao tiếp song song với các thiết bị ngoại vi.

+ ***Port 0***

Port 0 (các chân từ 32 đến 39) có hai công dụng. Trong các thiết kế tối thiểu, *port* 0 được sử dụng làm nhiệm vụ vào/ra. Trong các thiết kế lớn hơn có bộ nhớ ngoài, *port* 0 trở thành bus địa chỉ và bus dữ liệu dồn kênh.

+ ***Port 1***

Port 1 chỉ có một công dụng là vào/ra (các chân từ 1 đến 8), và dùng để giao tiếp vơi các thiết bị ngoại vi hoặc làm đường vào/ra hoặc làm các lối vào cho mạch định thời thứ ba.

Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866

+ ***Port 2***

Port 2 (các chân từ 21 đến 28) có hai công dụng, hoặc làm nhiệm vụ vào/ra hoặc là byte địa chỉ cao của bus địa chỉ 16-bít cho các thiết kế có bộ nhớ chương trình ngoài hoặc các thiết kế có nhiều hơn 256 byte bộ nhớ dữ liệu ngoài.

+ ***Port 3***

Port 3 (các chân từ 10 đến 17) có hai công dụng. Khi không hoạt động vào/ra, các chân của *port* 3 có nhiều chức năng riêng (mỗi chân có chức năng riêng liên quan đến các đặc trưng cụ thể của 8051).

+ ***Chân cho phép truy nhập bộ nhớ chương trình PSEN***

8051 cung cấp cho ta 4 tín hiệu điều khiển bus. Tín hiệu PSEN (Program store enable) là tín hiệu ra trên chân 29. Đây là tín hiệu điều khiển cho phép ta truy xuất bộ nhớ chương trình ngoài, chân này thường nối với chân cho phép ra OE (output enable) của EROM (hoặc ROM), cho phép đọc các byte lệnh.

Tín hiệu PSEN ở logic 0 trong suốt thời gian tìm nạp lệnh (Instruction Fetch). Các mã nhị phân của chương trình hay *opcode* (mã thao tác) được đọc từ EPROM, qua bus dữ liệu và được chốt vào thanh ghi lệnh IR của 8051 để được giải mã.

Khi thực thi một chương trình chứa ở ROM nội, PSEN được duy trì ở logic không tích cực (logic 1).

+ ***Chân cho phép chốt địa chỉ ALE***

8051 sử dụng chân 30, chân cho phép chốt địa chỉ ALE (address latch enable) để giải dồn kênh (demultiplexing) bus dữ liệu và bus địa chỉ. Khi *port 0* được sử dụng làm bus địa chỉ/dữ liệu dồn kênh, chân ALE đưa ra tín hiệu để chốt địa chỉ (byte thấp của địa chỉ 16-bít) vào một thanh ghi ngoài trong suốt 1/2 đầu của chu kỳ bộ nhớ (memory cycle). Sau khi điều này đã được thực hiện các chân của port 0 sẽ vào/ra dữ liệu hợp lệ trong suốt 1/2 thứ hai của chu kỳ bộ nhớ.

+ ***Chân truy xuất ngoài EA***

Lối vào này (31 chân) có thể nối được với 5V (logic 1) hoặc nối với GND (logic 0). Nếu chân này nối lên 5V, 8051/8052 thực thi chương trình trong ROM nội (Chương trình nhỏ hơn 4K/8K). Nếu chân này nối với GND (chân PSEN ở logic 0), chương trình cần thực thi chứa ở bộ nhớ ngoài. Đối với 8031/8032 chân EA phải ở logic 0 vì chúng không có bộ nhớ chương trình trên chip. Nếu chân EA ở logic 0 đối với 8051/8052, ROM nội ở bên trong chip được vô hiệu hoá và chương trình thực thi chứa ở EPROM bên ngoài.

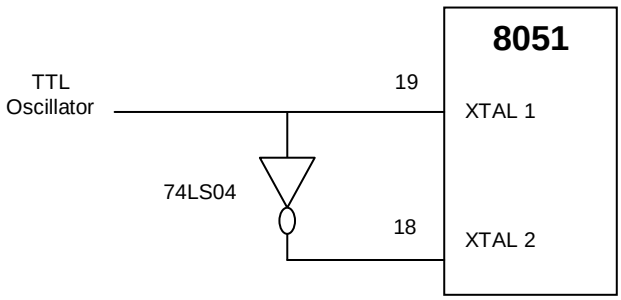
Các phiên bản EPROM của 8051 còn sử dụng chân EA làm chân nhận điện áp cấp điện 21V cho việc lập trình EPROM nội .

+ ***Chân RESET (RST)***

Lối vào RST (chân 9) là lối vào tái khởi động (master reset) của 8051 dùng để thiết lập trạng thái ban đầu cho hệ thống hay gọi tắt là reset hệ thống. Khi lối này được treo ở logic 1 tối thiểu hai chu kỳ máy, các thanh ghi bên trong của 8051 được nạp các giá trị thích hợp cho việc khởi động lại hệ thống.

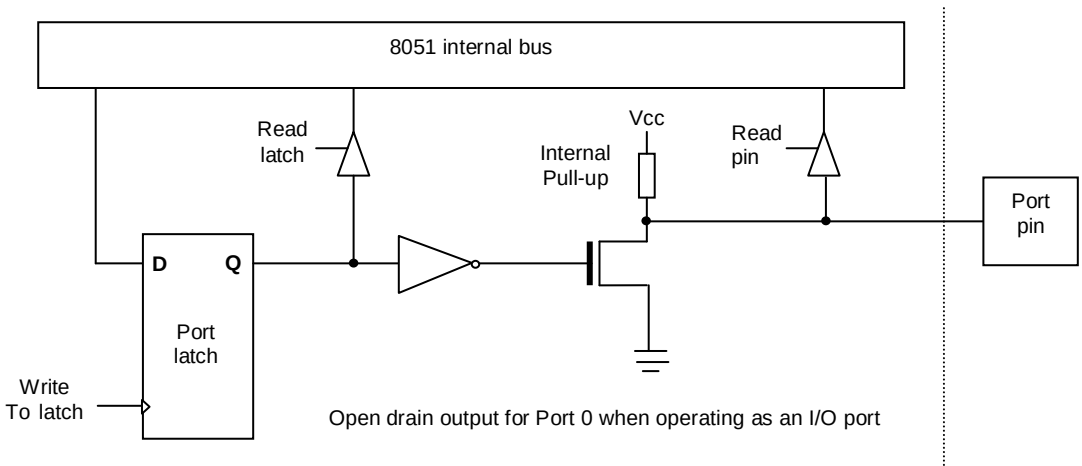
+ ***Các chân XTAL1&XTAL2***

Mạch dao động bên trong chip 8051 được ghép với thạch anh bên ngoài ở hai chân XTAL1 và XTAL2 (chân 18 và 19). Tần số danh định của thạch anh là 12MHz cho hầu hết các chip của họ MCS-51 (Riêng 80C31BH-1 sử dụng thạch anh 16MHz bên trong, mạch dao động trong chip không cần thạch anh bên ngoài). Một nguồn xung clock TTL có thể được nối với chân XTAL1 và XTAL2.



Hình 2.30 8051 ghép với mach dao động TTl bên ngoài

+ ***Cấu trúc của Port vào/ra.***



Hình II.31 Cấu trúc của port vào ra

8051 internal bus: Bus nội của 8051

Read latch: bộ chốt phục vụ đọc

Internal pull up: Mạch pull-up

Read pin: chân *port*

Port latch: Bộ chốt của *port*

Write to latch: Ghi vào bộ chốt

Sơ đồ mạch điện bên trong của *port* vào/ra được vẽ đơn giản như hình II.23, việc ghi đến 1 chân của *Port* sẽ nạp dữ liệu vào bộ chốt của *port*, lối ra Q của bộ chốt điều khiển một Transistor trường và Transistor này nối với chân của port. Khả năng fanout của các port 1, 2 và 3 là 4. Tải vi mạch TTL loại Schottky công suất thấp (LS) còn của *port* 0 là 8 tải loại LS.

Ta thấy có 2 khả năng: "đọc *bộ chốt*" và "đọc chân *port*". Các lệnh yêu cầu thao tác đọc-sửa-ghi đọc bộ chốt để tránh nhầm lẫn mức điện áp do sự kiện dòng tải tăng. Các lệnh nhập 1 bít của *port* (như MOV C,P1.5) đọc chân *port*. Trong trường hợp này bộ chốt của *port* phải chứa 1 nếu không transistor FET sẽ được kích bảo hoà và điều này kích lối ra dưới mức thấp.

+ ***Tổ chức bộ nhớ.***

Hầu hết các bộ vi xử lý (CPU) đều có không gian nhớ chung cho dữ liệu và chương trình, các chương trình được lưu trên đĩa và nạp vào RAM để thực thi vậy thì cả hai dữ liệu và chương trình đều lưu trú trong RAM.

Các chip vi điều khiển hiếm khi được sử dụng giống như các CPU trong các hệ máy tính, thay vào đó chúng được làm thành phần trung tâm trong các thiết kế hướng điều khiển, trong đó có có bộ nhớ dung lượng giới hạn, không có ổ đĩa và hệ điều hành. Chương trình điều khiển thường trú trong ROM.

8051 có không gian và bộ nhớ riêng cho chương trình và dữ liệu đều đặt ở bên trong chip, ta có thể mở rộng bộ nhớ chương trình và bộ nhớ dữ liệu bằng cách sử dụng các chip nhớ bên ngoài với dung lượng tối đa là 64K cho bộ nhớ chương trình (hay bộ nhớ mã) 64K cho bộ nhớ dữ liệu.

Bộ nhớ trong chip bao gồm ROM (Chỉ có ở 8051/8052) và RAM. RAM trên chip bao gồm vùng RAM đa chức năng và vùng RAM với từng bít được định địa chỉ (gọi là vùng RAM định địa chỉ bít, các Bank) và các thanh ghi chức năng đặc biệt SFR (specail function register). Hai đặc tính đáng lưu ý là:

(a) các thanh ghi và các *port* vào/ra được định địa chỉ theo kiểu ánh xạ bộ nhớ (memory mapped) và được truy xuất như một vị trí trong bộ nhớ.

(b) Vùng *stack* thường trú trên RAM trong chip (RAM nội) thay vì ở trong RAM ngoài đối với cá bộ vi xử lý.

+ ***Vùng RAM đa mục đích.***

Vùng RAM đa mục đích có 80 byte đặt ở địa chỉ từ 30H đến 7FH, bên dưới vùng này từ địa chỉ 00H đến 2FH là vùng nhớ có thể được sử

dụng tương tự. Bất kỳ vị trí nhớ nào trong Vùng RAM đa mục đích đều có thể được truy xuất tự do bằng cách định địa chỉ trực tiếp hoặc gián tiếp.

+ ***Vùng RAM định địa chỉ bít.***

8051 chứa 210 bít được định địa chỉ trong đó128 bít chứa trong các byte ở địa chỉ 20H đến 2FH (16 byte x 8 bít = 128 bít) phần còn lại chứa trong các thanh ghi đặc biệt. Truy xuất các bít riêng rẽ thông qua phần mềm là một đặc trưng mạnh của hầu hết các bộ vi điều khiển. Các bít có thể được Set, Reset, AND, OR, v.v… bằng một lệnh. Hầu hết các bộ vi xử lý yêu cầu một chuỗi lệnh đọc-sửa-ghi để nhận được cùng một kết quả. Ngoài ra 8051 còn có các *port* vào/ra có thể định địa chỉ từng bít, điều này làm đơn giản việc giao tiếp bằng phần mềm với các thiết bị vào/ra đơn bít.

+ ***Các thanh ghi.***

32 vị trí thấp nhất của bộ nhớ nội được sử dụng như những thanh ghi. Các lệnh của 8051 hỗ trợ 8 thanh ghi từ R0 đến R7 thuộc băng 0 (bank 0) đây là băng mặc định sau khi reset hệ thống. Các thanh ghi này được đánh địa chỉ từ 00H đến 07H.

Các lệnh sử dụng các thanh ghi từ R0 đến R7 là các lệnh ngắn và thực hiện nhanh hơn so với các lệnh tương đương sử dụng kiểu định địa chỉ trực tiếp. Các giá trị sử dụng nhiều nên chứa ở một trong các thanh ghi này. Băng thanh ghi đang được sử dụng được gọi là băng thanh ghi tích cực. Băng thanh ghi tích cực có thể được thay đổi bằng cách thay đổi các bít chọn băng trong từ trạng thái chương trình PSW .

+ ***Các thanh ghi chức năng đặc biệt (SFR)***

Các thanh ghi của hầu hết các bộ vi xử lý đều được truy xuất rõ ràng bởi một tập lệnh. Thao tác được xác định rỏ ràng trong *opcode* của lệnh. Việc truy xuất các thanh ghi cũng được sử dụng trên 8051.

Các thanh ghi của 8051 được cấu hình thành từ một phần của RAM trên chip, do vậy mỗi thanh ghi cũng có một địa chỉ. Điều này hợp lý với 8051 vì chip này có rất nhiều thanh ghi. Cũng như các thanh ghi từ R0 đến R7, ta có 21 thanh ghi chức năng đặc biệt SFR chiếm phần trên của RAM nội từ địa chỉ 80H đến FFH .

+ ***Cờ nhớ.***

Cờ nhớ CY (CARRY FLAG) có hai công dụng. Công dụng truyền thống trong các phép toán số học là được set bằng 1 nếu có số nhớ từ phép cộng bít 7 hoặc có số mượn mang đến bít 7.

+ ***Cờ nhớ phụ.***

Khi cộng các giá tri BCD, cờ nhớ phụ AC (auxiliari carry flag) được set bằng 1 nếu có một số nhớ được tạo từ bít 3 chuyển sang bít 4 hoặc nếu kết quả trong 4 bit thấp nằm trong vùng từ 0AH đến 0FH. Nếu các giá trị được cộng là giá trị BCD, lệnh cộng phải được tiếp theo bởi lệnh DA A (hiệu chỉnh thập phân thanh chứa A) để đưa các giá trị kết quả lớn hơn 9 về giá trị đúng.

+ ***Cờ 0***

Đây là cờ có nhiều mục đích dành cho các ứng dung của người lập trình.

+ ***Các bit chọn dãy thanh ghi***

Các bít chọn dãy thanh ghi RS0, RS1 dùng để xác định dãy thanh ghi tích cực. Các bít này được xoá sau khi có thao tác reset hệ thống và đổi mức logic bởi phần mềm khi cần.

+ ***Cờ tràn***

Cờ tràn OV (overflow flag) được reset bằng 1 sau phép toán cộng hoặc trừ nếu có xuất hiện một tràn số học. Khi các số có dấu được cộng hoặc được trừ, phần mền có thể kiểm tra bit tràn OV để xác định xem kết quả có nằm trong tầm hay không.

+ ***Cờ chẵn lẻ***

Bit chẵn lẻ P tự động được set bằng 1 hay xoá bằng 0 ở mỗi chu kỳ máy để thiết lập kiểm tra chẵn lẻ cho thanh chứa A. Số các bít 1 trong thanh chứa cộng với bít P luôn là số chẵn. Nếu thanh chứa có nội dung 10101101B, bít P sẽ là 1 để có số bít 1 là 6. Bít chẵn lẻ được sử dụng nhiều để kết hợp với các chương trình vào/ra nối tiếp trước khi truyền dữ liệu hoặc kiểm tra chẵn lẻ sau khi truyền dữ liệu.+

+ ***Từ trạng thái chương trình PSW***

Bit	Ký hiệu	Địa chỉ	Mô tả bít
PSW.7	CY	D7H	Cờ nhớ
PSW.6	AC	D7H	Cờ nhớ phụ
PSW.5	F0	D6H	Cờ 0
PSW.4	RS1	D5H	Chọn dãy thanh ghi (bít 1)
PSW.3	RS0	D4H	Chọn dãy thanh ghi bít 0
			00 = Bank 0: địa chỉ từ 00H đến 07H
			01 = Bank 1: địa chỉ từ 08 đến 0FH
			10 = Bank 2: địa chỉ từ 10H đến 17H
			11 = Bank 3: địa chỉ từ 18H đến 1FH
PSW2	OV	D3H	Cờ tràn
PSW1	-	D1H	Dự trữ
PSW0	P	D0H	Kiểm tra chẵn lẻ

Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866

+ ***Thanh ghi B***

Thanh ghi B ở địa chỉ F0H được dùng chung với thanh chứa A trong các phép toán nhân, chia. Lệnh MUL AB nhân 2 số 8-bít không dấu chứa trong A và B và chứa kết quả vào cặp Thanh ghi B:A (Thanh chứa A chứa byte thấp và thanh chứa B chứa byte cao của tích số).

Lệnh chia DIV AB chia A bởi B, thương số chứa trong thanh chứa A và số dư chứa trong Thanh ghi B. Thanh ghi B còn được xử lý như một thanh ghi nháp. Các bít được định địa chỉ của thanh ghi B có địa chỉ từ F0H đến F7H.

+ ***Con trỏ Stack***

Con trỏ *stack* SP (stack pointer) là một thanh ghi 8-bít ở địa chỉ 81H. SP chứa địa chỉ của dữ liệu hiện đang ở đỉnh của *stack*. Các lệnh liên quan đến *stack* bao gồm dữ liệu cất vào *stack* và lệnh lấy dữ liệu ra khỏi *stack*. Việc cất vào *stack* làm tăng SP trước khi ghi dữ liệu và việc lấy dữ liệu ra khỏi *stack* sẽ giảm SP. Vùng *stack* của 8051 được giữ trong RAM nội và được giới hạn đến các địa chỉ truy xuất được bởi kiểu định địa chỉ gián tiếp.

+ ***Con trỏ dữ liệu DPTR***

Con trỏ dữ liệu DPTR (data pointer) dùng để truy xuất bộ nhớ bên ngoài hoặc bộ nhớ dữ liệu ngoài. DPTR là một thanh ghi có địa chỉ là 16 bít có địa chỉ là 82H(DPL, byte thấp) và 83H (DPL, byte cao).

+ ***Các thanh ghi port.***

Các *port* vào/ra của 8051 bao gồm *port* 0 tại địa chỉ 80H, *port* 1 tại địa chỉ 90H, *port* 2 tại địa chỉ A0H, và *port* 3 tại địa chỉ B0H, các *port* 0, 2 và 3 không được dùng để vào/ra nếu ta sử dụng thêm bộ nhớ ngoài hoặc nếu có một số đặc tính của 8051 được sử dụng (như là ngắt, *port* nối tiếp,…) P1.2 đến P1.7, ngược lại, luôn là các đường vào/ra đa mục đích hợp lệ.

Tất cả các *port* đều được định địa chỉ từng bit nhằm cung cấp khả năng giao tiếp mạnh.

+ ***Các thanh ghi định thời.***

8051 có 2 bộ đếm định thời (time/counter) 16-bít để định các khoảng thời gian hoặc để đếm các sự kiện. Bộ định thời 0 có địa chỉ là 8AH (TL0, byte thấp) và 8CH (TH0, byte cao); bộ định thời 1 có địa chỉ 8BH (TL1, byte thấp) và 8CH (TH1, byte cao).

Hoạt động của bộ định thời được thiết lập bởi thanh ghi chế độ định thời TMOD (time mode register) ở địa chỉ 89H và thanh ghi điều khiển

định thời TCON (time control register) ở địa chỉ 88H. Chỉ có TCON được định địa chỉ từng bít.

+ ***Các thanh ghi của port nối tiếp.***

Bên trong 8051 có một *port* nối tiếp để truyền thông với các thiết bị nối tiếp như các thiết bị đầu cuối hoặc modem, hoặc để giao tiếp với các IC khác có mạch giao tiếp nối tiếp (như các thanh ghi dịch).

Một thanh ghi được gọi là bộ đệm dữ liệu nối tiếp SBUF (serial data buffer) ở địa chỉ 99H lưu dữ liệu truyền đi và dữ liệu nhận về. Việc ghi lên SBUF sẽ nạp dữ liệu để truyền và việc đọc SBUF sẽ lấy dữ liệu đã nhận được.

Các chế độ hoạt động khác nhau được lập trình thông qua thanh ghi điều khiển *port* nối tiếp SCON (serial port control register) ở địa chỉ 98H, thanh ghi này được định địa chỉ từng bit.

+ ***Các thanh ghi ngắt***

8051 có một cấu trúc ngắt với 2 mức ưu tiên và 5 nguyên nhân ngắt. (5 source, 2 priority level interrupt structure). Các ngắt bị vô hiệu hoá sau khi reset hệ thống và sau đó được cho phép bằng cách ghi vào thanh ghi cho phép ngắt IE (interrupt enable register) ở địa chỉ A8H. Mức ưu thanh ghi được thiết lập qua thanh ghi ưu tiên ngắt IP (interrupt priority register) ở địa chỉ B8H. Cả 2 thanh ghi này đều được định địa chỉ từng bit.

+ ***Thanh ghi điều khiển nguồn***

Thanh ghi điều khiển nguồn PCON (power control register) có địa chỉ 87H chứa các bít điều khiển.

Bit SMOD tăng gấp đôi tốc độ baud của *port* nối tiếp. Khi *port* này hoạt động ở các chế độ 1, 2 hoặc 3, các bít 4, 5 và 6 của PCON không được đinh nghĩa. Các bit 2 và 3 là các bít cờ đa mục đích dành cho các ứng dụng của người sử dụng.

Các bít điều khiển nguồn, nguồn giảm PD và nghỉ IDL, hợp lệ cho tất cả chip thuộc họ MCS-51, nhưng chỉ được thực hiện trong các phiên bản CMOS của MCS-51. TCON không được định địa chỉ bít.

Chế độ nguồn giảm.

Lệnh thiết lập bít PD băng 1 sẽ là lệnh sau cùng được thực hiện trước khi đi vào chế độ nguồn giảm:

(1) Mạch giao động trên chip ngừng hoạt động

(2) Mọi chức năng ngừng hoạt động

(3) Nội dung của RAM trên chip được duy trì

(4) Các phần port duy trì mức logic của chúng

(5) ALE và PSEN giữ được ở mức thấp. Chỉ ra khỏi chế độ này bằng cách reset hệ thống.

Trong suốt thời gian ở chế độ nguồn giảm, Vcc có điện áp là 2v. Cần phải giữ cho Vcc không thấp hơn sau khi đạt được chế độ nguồn giảm và cần phục hồi Vcc bằng 5V tối thiểu 10 chu kỳ dao động trước khi chân RST đạt mức thấp lần nữa.

Bit	Ký hiệu	Mô tả
7	SMOD	bít tăng gấp đôi tốc độ baud, bít này khi set làm cho tốc độ baud tăng 2 ở các chế độ 1,2 và 3 của *port* nối tiếp
6	-	Không định nghĩa
5	-	Không định nghĩa
4	-	Không định nghĩa
3	CF1	Bit cờ đa mục đích 1
2	CF0	Bit cờ đa mục đích 2
1	FD	Nguồn giảm; thiết lập để tích cực chế độ nguồn giảm, chỉ ra khỏi chế độ bằng reset
0	IDL	Chế độ nghỉ; thiết lập để tích cực chế độ ghi, chỉ ra khỏi chế độ bằng một ngắt hoặc reset hệ thống.

Chế độ nghỉ.

Lệnh thiết lập bít IDL bằng 1 sẽ là lệnh sau cùng được thực thi trước khi đi vào chế độ nghỉ. Ở chế độ nghỉ, tín hiệu *clock* nội được khoá không cho đến CPU nhưng không khoá đối với các chức năng ngắt, định thời và *port* nối tiếp. Trạng thái của CPU được duy trì và nội dung của tất cả các thanh ghi cũng được giữ không thay đổi.

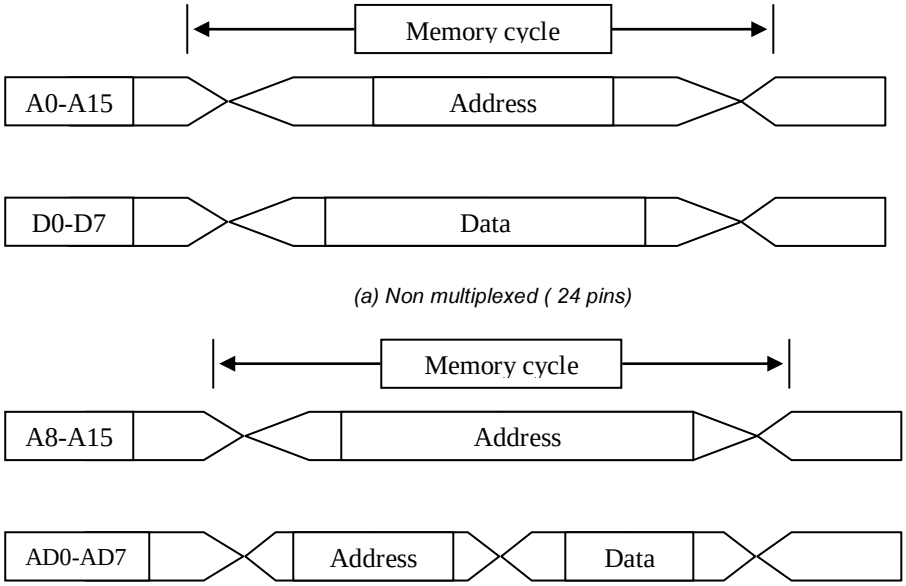
Các chân *port* cũng được duy trì các mức logic của chúng. ALE và PSEN được giữ ở mức cao.

Chế độ nghỉ kết thúc bằng cách cho phép ngắt hoặc bằng cách reset hệ thống. Cả hai cách vừa nêu đều xoá bit IDL.

+ ***Bộ nhớ ngoài***

Các bộ vi xử lý họ MCS-51 có khả năng mở rộng các tài nguyên trên chíp (bộ nhớ, I/O, v.v…) để tránh hiện tượng cổ chai trong thiết kế. Cấu trúc của MCS-51 cho khả năng mở rộng không gian bộ nhớ chương trình đến 64K và không gian bộ nhớ dữ liệu đến 64K ROM và RAM ngoài khi cần thiết.

Các IC giao tiếp ngoại vi cũng có thể được thêm vào để mở rộng khả năng vào/ra. Chúng trở thành một phần của không gian bộ nhớ dữ liệu ngoài bằng cách sử dụng cách định địa chỉ kiểu I/O ánh xạ bộ nhớ. Khi bộ nhớ ngoài được sử dụng, *port* 0 làm nhiệm vụ của port vào/ra. *Port* này trở thành bus địa chỉ (A0-A7)và bus dữ liệu (D0-D7) dồn kênh. Lối ra ALE chốt byte thấp của địa chỉ ở thời điểm bắt đầu mỗi một chu kì bộ nhớ ngoài. *Port* 2 thường (nhưng không phải luôn luôn) được dùng làm byte cao của bus địa chỉ .



(a) Non multiplexed (24 pins)

(b) Multiplexed (16 pins)

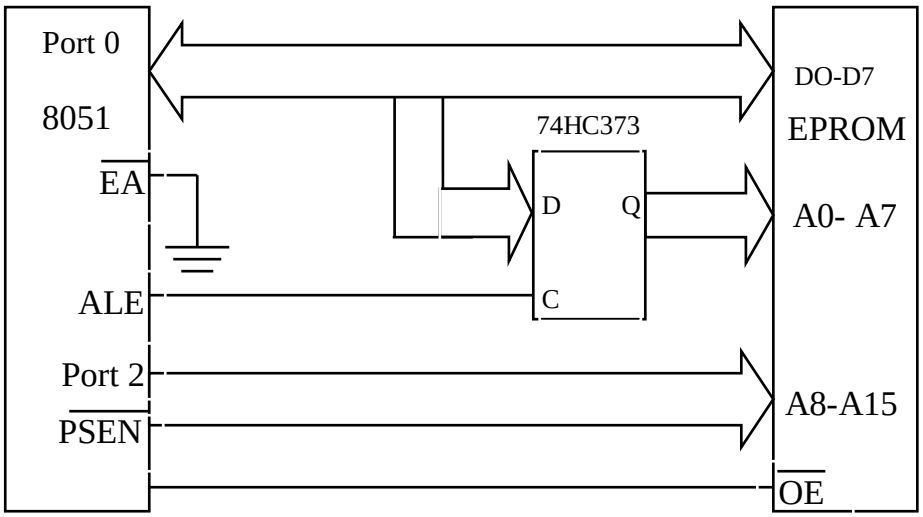
Hình II.32 Đa hợp địa chỉ (byte thấp) và Bus dữ liệu
(a) không dồn kênh (24 chân) , (b) dồn kênh (16 chân)

+ *Truy xuất bộ nhớ chương trình ngoài*

Bộ nhớ chương trình ngoài là bộ nhớ chỉ đọc, được cho phép bởi tín hiệu PSEN. Khi có một EPROM ngoài được sử dụng, cả hai *port* 0 và *port* 2 đều không còn là các *port* vào/ra. Kết nối 8051với bộ nhớ ngoài EPROM được trình bày ở hình II.33

Một chu kỳ máy của 8051 có 12 xung nhịp. Nếu bộ giao động trên chip có tần số 12MHz, một chu kỳ máy dài 1μsec. Trong một chu kỳ máy điển hình, ALE có 2 xung và 2 byte của lệnh được đọc từ bộ nhớ chương trình (nếu lệnh chỉ có một byte, byte thứ hai được loại bỏ).

Bộ nhớ dữ liệu ngoài là bộ nhớ đọc/ghi được cho phép bởi các tín hiệu $\overline{RD}$ và $\overline{WR}$ ở các chân P3.6. Lệnh dùng để truy xuất bộ nhớ dữ liệu ngoài là: MOVX, sử dụng hoặc con trỏ dữ liệu 16-bít DPTR hoặc R0, R1 làm thanh ghi chứa địa chỉ.



Hình II.33 - Kết nối 8051 với bộ nhớ chương trình ngoài

RAM có thể giao tiếp với 8051 theo cách như EPROM ngoại trừ đường $\overline{RD}$ nối với đường cho phép xuất ($\overline{OE}$) của RAM và $\overline{WR}$ nối với đường ghi (W) của RAM. Các kết nối với bus dữ liệu và bus địa chỉ giống như EPROM. Bằng cách sử dụng các *port* 0 và *port* 2, ta có dung lượng RAM ngoài lên đến 64K được kết nối với 8051.

II.4.3 Lập trình cho µC8051

Để lập trình cho 8051, người lập trình cần nắm thật vững cách tổ chức rất hữu hiệu nhưng tương đối phức tạp của bộ nhớ RAM tích hợp trong chip. Không đơn thuần đóng vai trò bộ nhớ dữ liệu trong MCS51, nó còn sử dụng một phần bộ nhớ RAM để làm thanh ghi đa năng và thanh ghi với các chức năng đặc biệt.

Tồn tại chương trình Assembler riêng cho họ MCS51, lập trình hợp ngữ tương đương như lập trình hợp ngữ cho họ 80x86. Điểm mạnh tương ứng là tồn tại một phiên bản ngôn ngữ C cho MCS51, tạo điều kiện cho những ai đã quen với lập trình C có thể tạo các phần mềm ứng dụng để cài đặt vào trong bộ nhớ ROM của MCS51 đối với những ứng dụng thực tế.

Bạn đọc có thể tham khảo tài liệu [1] về lập trình cho µC8051 được nêu ở cuối cuốn giáo trình này.

II.4.4 Các khả năng ứng dụng của µC8051

Thông thường, các trung tâm Vi xử lý được dùng để xây dựng nên các máy tính. Riêng các trung tâm của Single Chip Microcomputer, do những cấu trúc đặc trưng và tính năng kỹ thuật, được ứng dụng nhiều trong các thiết kế nhỏ, với số thành phần phụ trợ thêm vào tối thiểu nhất. Nhờ cấu trúc và khả năng cài đặt các chương trình ứng dụng ngay trong

bộ nhớ ROM tích hợp sẵn, các hướng và các ứng dụng cụ thể của họ Vi xử lý này chủ yếu tập trung vào các mục đích gia dụng và dân dụng. Thống kê một số lĩnh vực ứng dụng của các trung tâm Vi xử lý họ này được liệt kê trong bảng sau.

Trong gia đình	Đồ điện gia dụng Thiết bị đàm thoại Điện thoại Hệ thống an toàn Mở đóng cửa Trả lời tự động Máy FAX	Máy tính gia đình TV Truyền hình cáp VCR Camera Điều khiển từ xa Trò chơi điện tử	Nhạc cụ điện tử Máy khâu Điều khiển ánh sáng Máy nhắn tin v.v…
Thiết bị văn phòng	Điện thoại Máy tính Hệ thống an toàn	Máy FAX Lò vi sóng Photocopy	Máy in Laze Máy in màu Máy nhắn tin
Tự động hoá	Máy tính hành trình Điều khiển động cơ	Đo lường Truyền tin…	Điều hoà không khí v.v…

Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866

CHƯƠNG III. BỘ NHỚ TRONG CỦA HỆ VI XỬ LÝ

III.1 Bộ nhớ trong hệ Vi xử lý

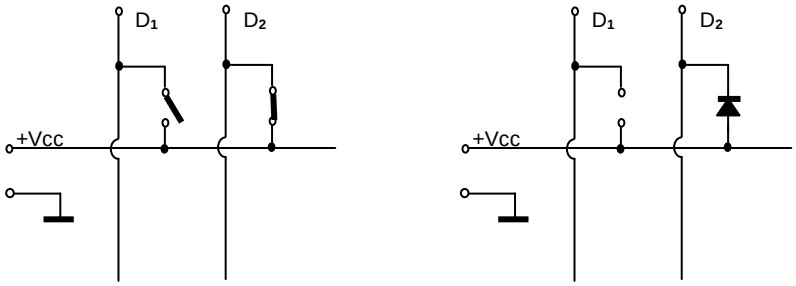
Bộ nhớ được sử dụng để lưu giữ mã lệnh của chương trình và dữ liệu cần xử lý. Bộ nhớ được ghép nối trực tiếp với CPU qua BUS hệ thống và là nơi đầu tiên CPU truy xuất tới để lấy thông tin khi khởi động hệ thống. Yêu cầu đặt ra cho bộ nhớ là phải cho phép truy xuất với tốc độ cao để đáp ứng kịp thời các đòi hỏi của CPU. Chỉ có bộ nhớ bán dẫn mới đáp ứng được yêu cầu cao về tốc độ truy xuất cao (hàng trăm đến hàng chục nsec).

Bộ nhớ bán dẫn được chia ra hai loại: Bộ nhớ chỉ đọc ROM (Read Only Memory) và bộ nhớ truy xuất ngẫu nhiên RAM (Random Access Memory).

III.1.1 Phần tử nhớ, vi mạch nhớ, từ nhớ và dung lượng bộ nhớ

a) *Phần tử nhớ*

Phần tử nhớ thông thường là một mạch điện có thể ghi lại và lưu giữ một trong hai giá trị của một biến nhị phân, hoặc "0" hoặc "1", tương ứng với *không có điện áp* hoặc *có điện áp*, được gọi là **bit**. Trên mạch điện dưới đây (Hình III.1), trên dây D_1 sẽ không có điện áp (do công tắc mở), trong khi dây D_2 có điện áp (vì công tắc đóng, hay thông qua diode mắc theo chiều thuận), gần bằng giá trị nguồn nuôi Vcc, tương ứng với bit D_1 = "0" và bit D_2 = "1".



Phương pháp tạo phần tử nhớ $D_1 = 0$ và $D_2 = 1$ bằng mạch điện đơn giản

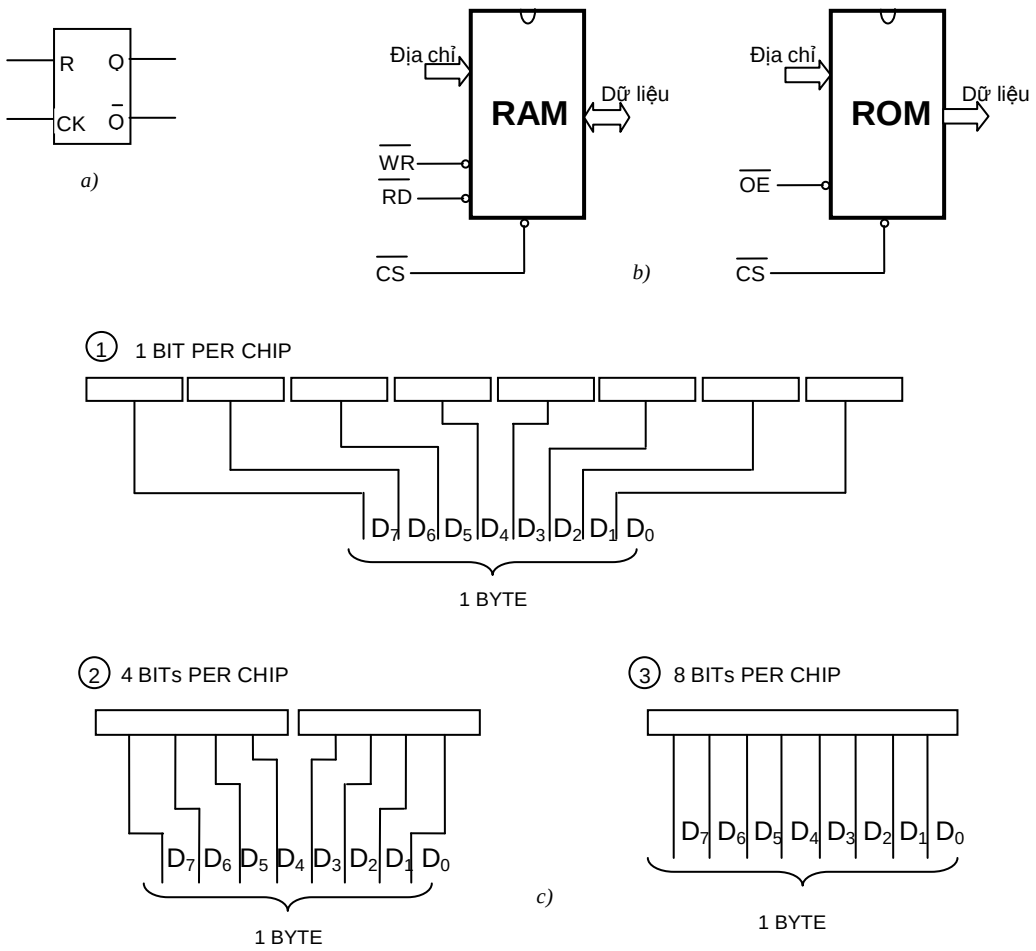
Hình III.1 Mô phỏng phần tử nhớ

Mạch flip-flop RS (còn gọi là triger RS) đồng bộ là một mạch có khả năng lưu giữ các giá trị "0" hoặc "1" ở lối ra. Có thể dùng RS flip-flop làm một mạch lưu giữ tín hiệu vào R bằng cách chốt dữ liệu đó lại tại đầu ra Q (hình III.2a). Các hãng chế tạo thực hiện mạch này bằng công nghệ cao, nên kích thước vô cùng nhỏ, có thể có hàng nhiều triệu phần tử nhớ trên một diện tích $1mm^2$. Các vi mạch nhớ thông thường được chế tạo với độ dài từ nhớ và số lượng từ nhớ cố định. *Số bit nhớ được liên kết tại một vị trí nhớ (có cùng địa chỉ) trong một chip nhớ* được gọi là *từ nhớ*

Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866

của chip nhớ, thường được chọn là 1, 4, hoặc 8bit. Để tạo được một *từ nhớ của bộ nhớ*, tức là từ nhớ có độ dài (số bit trong một từ) chuẩn (theo chuẩn IBM là 8 bits), trong một số trường hợp nhất định cần phải tiến hành ghép các chip nhớ lại với nhau.

Hình III.2 a), b) và c) cho ta khái niệm về khả năng tạo một từ nhớ cơ bản (byte) khi từ nhớ của chip nhớ là 1bit, 2bits và 4 bits. Trong trường hợp độ dài từ nhớ của chip nhớ là 8 bits, việc liên kết là không cần thiết.



Hình III.2 a) Mạch Flip-flop RS như một phần tử nhớ giá trị nhị phân
b)Chip nhớ RAM và chip nhớ ROM
c) Ghép các chip nhớ có độ dài từ nhớ khác nhau, để tạo được từ nhớ có độ dài 8 bits

III.1.2 Vài nét về bộ nhớ trong của hệ Vi xử lý và máy tính PC

Do ưu điểm tương thích tuyệt đối về kích thước, tiêu thụ năng lượng thấp và mức logic, đặc biệt là tốc độ truy nhập, nên bộ nhớ bán dẫn được sử dụng làm bộ nhớ chính (Main Memory) trong các hệ Vi xử lý cũng như trong các máy tính PC, nhiều khi được ghép nối ngay trong bo mạch chính, hoặc được thiết kế như những vỉ nhỏ cắm vào khe cắm riêng trên bo mạch chính.

Nhờ những tiến bộ vượt bậc của công nghệ vi mạch, đặc biệt là công nghệ cao (High Technology) các chip nhớ được chế tạo ngày càng nhỏ và có dung lượng tương đối lớn, tốc độ truy nhập rất cao và giá thành thấp. Hiện đã có các chip nhớ có dung lượng hàng trăm triệu từ nhớ, được cấu thành từ hàng chục tỷ transistor trên một một cấu trúc cỡ $1mm^2$.

Bộ nhớ trong của một hệ Vi xử lý gồm hai loại chính:

- Bộ nhớ ROM – là bộ nhớ chỉ đọc (Read Only Memory), thông thường chứa các chương trình giám sát (monitoring) các hoạt động chức năng của hệ Vi xử lý: chương trình thiết lập hệ thống, chương trình vào/ra dữ liệu, quản lý và phân phát bộ nhớ, quản lý các thiết bị vào/ra v.v…Đối với máy tính PC, đó là chương trình hệ thống vào/ra cơ sở (BIOS – Basic Input Output System). Đặc điểm cơ bản nhất của bộ nhớ này là sự bảo toàn dữ liệu khi không có nguồn nuôi.
- Bộ nhớ RAM – là bộ nhớ ghi/đọc tuỳ tiện (Random Access Memory). Vì có khả năng ghi/đọc tuỳ theo người dùng, nên bộ nhớ này được sử dụng để chứa dữ liệu, các chương trình ứng dụng nhất thời của người dùng v.v… Trong máy tính PC, bộ nhớ này là nơi chương trình hệ điều hành được nạp khi khởi động máy, hay nơi chứa các chương trình ứng dụng lúc nó được thực thi. Bộ nhớ này bị mất dữ liệu khi bị mất nguồn nuôi.

Trong các hệ Vi xử lý đơn giản, hai bộ nhớ này thường được thiết kế và lắp ráp từ các chip nhớ riêng biệt thành một vỉ nhớ. Địa chỉ được *giải mã cho từng chip nhớ* nhờ khối giải mã, thông thường là một vi mạch giải mã hay được xây dựng từ các mạch tổ hợp logic. Các tín hiệu điều khiển việc ghi/đọc bộ nhớ do CPU cung cấp. Mạch triger RS đồng bộ là một mạch có khả năng lưu giữ các giá trị “0” hoặc “1” ở lối ra. Có thể dùng RS flip-flop làm một mạch lưu giữ tín hiệu vào R bằng cách chốt dữ liệu đó lại tại đầu ra Q (hình III.2)

Bộ nhớ được xây dựng từ các chip nhớ. Các chip nhớ RAM (SRAM hoặc DRAM) thường có các từ nhớ có độ dài 1 bit, 4 bits hoặc 8 bits. Từ các chíp nhớ loại này có thể xây dựng được bộ nhớ với mỗi ô nhớ chứa được byte dữ liệu (8 bits).

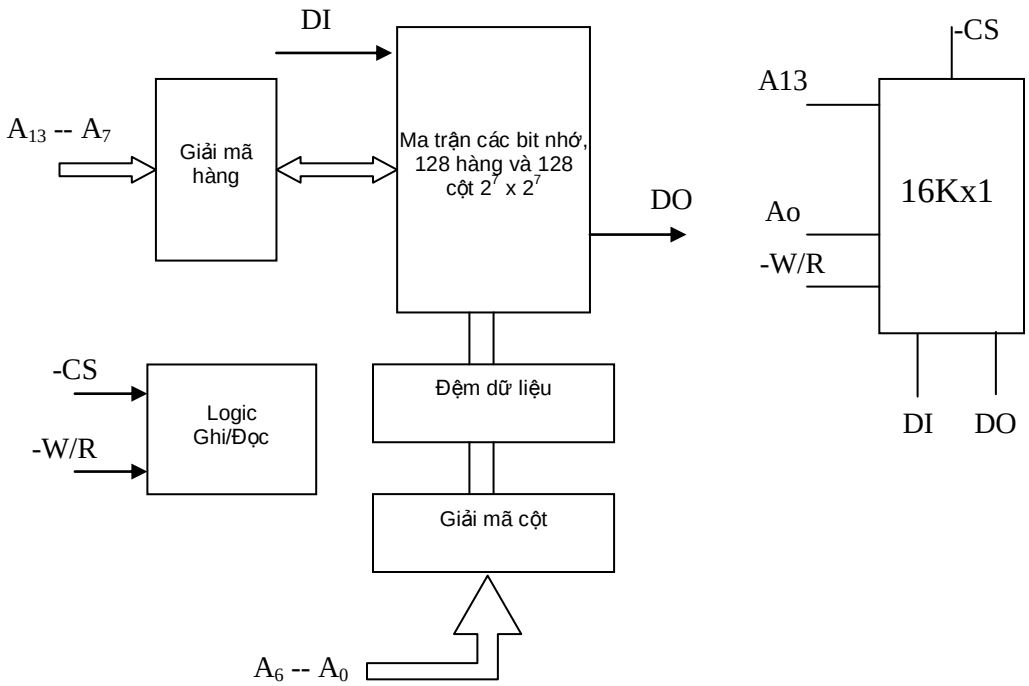
- **Xây dựng bộ nhớ với các chip SRAM**

Giả sử cần xây dựng một bộ nhớ kích thước 16Kbyte trên cơ sở các chíp SRAM loại 16Kx1bit.

Băng nhớ SRAM 16Kbyte được xây dựng trên cơ sở 8 chip SRAM loại 16K x 1bit, để có được ô nhớ có độ dài 8 bits (từ nhớ cơ bản). Để làm được điều này người ta sắp đặt 8 chip SRAM loại 16K x 1bit sao cho mỗi chip tại một vị trí xác định sẽ đảm nhiệm lưu trữ bit dữ liệu có trọng số tương ứng trong byte dữ liệu.

Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866

Cấu trúc chip SRAM



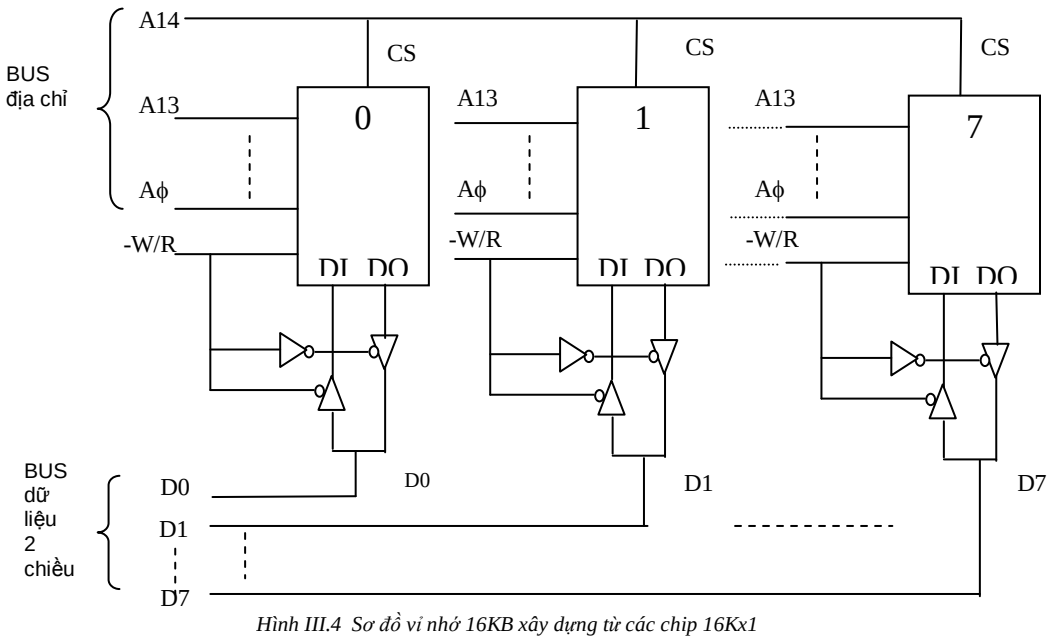
Hình III.3 Chip nhớ RAM 64K bit (64K x 1)

Các đường tín hiệu :

A13 - A0 BUS địa chỉ

-CS: Tín hiệu chọn chip. Nếu CS = 0 thì truy nhập được chip

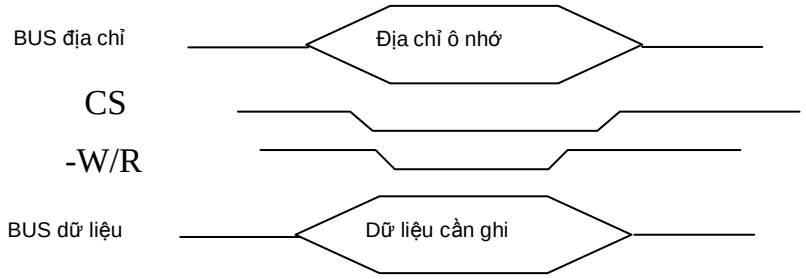
-W/R: Tín hiệu điều khiển ghi/đọc. W=0 điều khiển ghi



Hình III.4 Sơ đồ vi nhớ 16KB xây dựng từ các chip 16Kx1

D0 - D7: Các đường dây truyền các bit dữ liệu từ D0 đến D7.

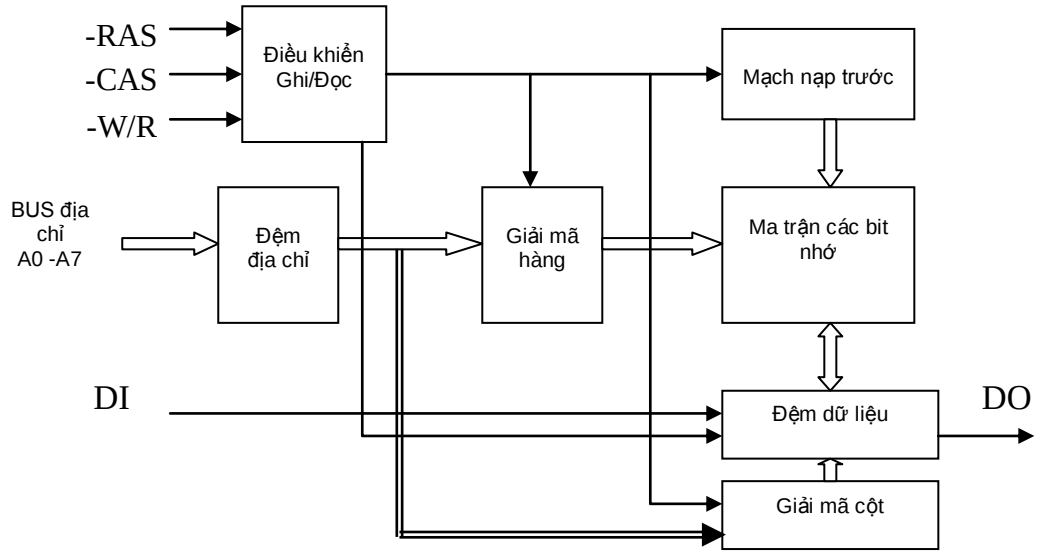
Chu kỳ ghi bộ nhớ SRAM :



Hình III.5 - Biểu đồ thời gian ghi đọc bộ nhớ

– **Tổ chức bộ nhớ với DRAM**

Cấu trúc của chip DRAM:



Hình III.6 - Cấu trúc bên trong chip DRAM

DRAM dùng phương pháp dồn kênh để nạp lần lượt (2 lần) địa chỉ hàng và địa chỉ cột vào đệm địa chỉ.

Tín hiệu điều khiển :

+ RAS: khi RAS (Row Access Strobe) tích cực thì địa chỉ hàng được nạp (chốt lại).

+ CAS: khi CAS (Column Access Strobe) tích cực thì địa chỉ cột được nạp (chốt lại).

+ WE: WE ≡ "0" điều khiển ghi chip, WE ≡ "1" điều khiển đọc chip.

Việc xây dựng bộ nhớ từ các chip DRAM được thực hiện gần tương tự như với SRAM.

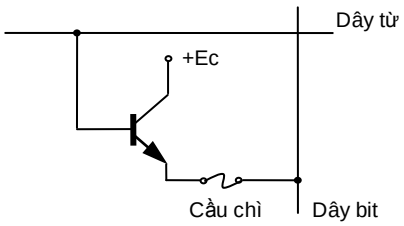
III.1.3 Phân loại các chip nhớ ROM, RAM

Các chip nhớ ROM (Read Only Memory) được phân loại theo khả năng ghi đọc như sau:

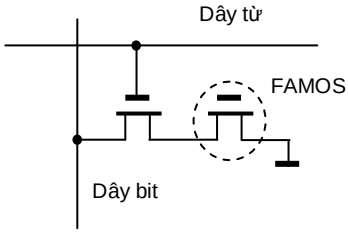
- ROM, nhớ chỉ đọc, dữ liệu trong chip nhớ loại này được ghi ngay tại hãng sản xuất chip nhớ theo đơn đặt hàng của các nhà sản xuất thiết bị cần sử dụng nó.
- EPROM, chip nhớ ROM có khả năng xoá nội dung và ghi lại nội dung. Nội dung được xoá bằng tia cực tím nhờ một thiết bị chuyên dùng.
- EEPROM, chip nhớ ROM có khả năng xoá, ghi lại nhờ sử dụng xung điện

Các chip nhớ RAM chủ yếu được chia thành 2 loại chủ yếu sau:

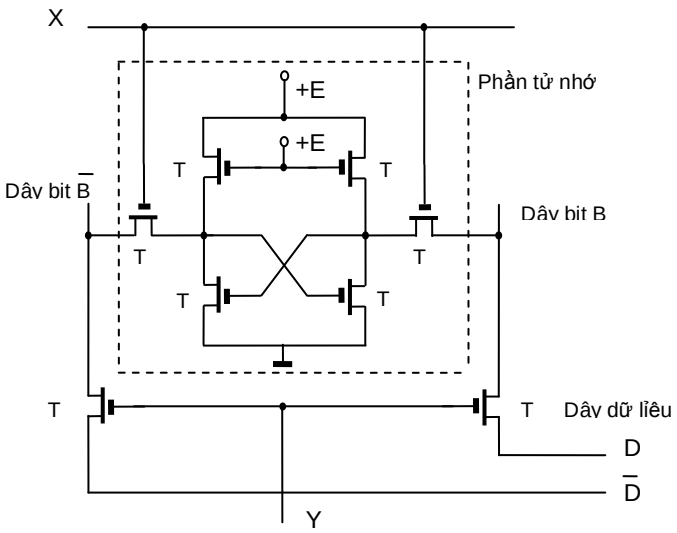
- RAM tĩnh (SRAM), mỗi phần tử nhớ là một mạch flip-flop, trong quá trình sử dụng không cần quan tâm đến việc dữ liệu được lưu giữ nếu không bị mất nguồn nuôi
- RAM động (DRAM), phần tử nhớ dùng công nghệ nạp điện tích lên tụ điện. Trong quá trình sử dụng cần thiết một chế độ *làm tươi.*



Phần tử nhớ PROM (cầu chì là hợp kim Ni-Cr)

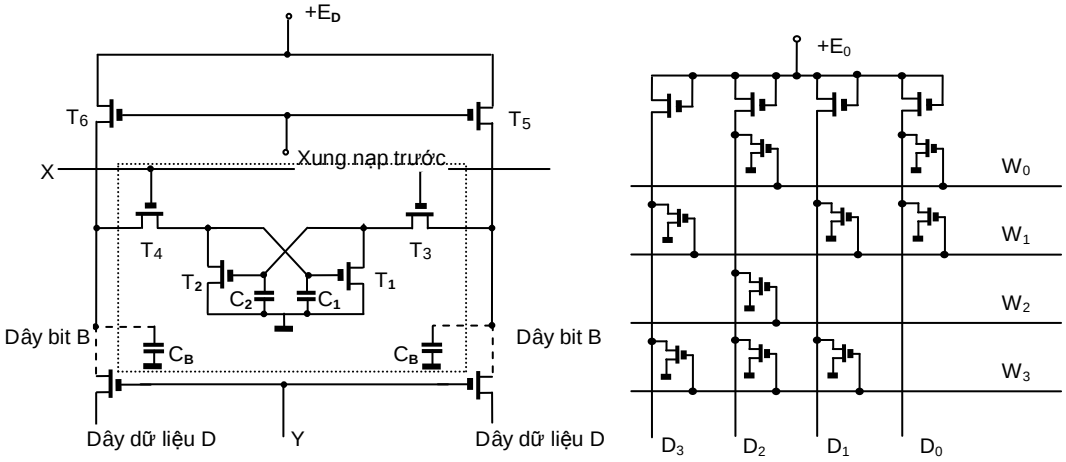


Phần tử nhớ EPROM (cầu chì là Transistor FAMOS)



Phần tử nhớ RAM tĩnh: Một mạch Flip-flop

Hình III.7a – Sơ đồ cấu trúc các phần tử nhớ cơ bản



Hình III.7b – Sơ đồ cấu trúc các phần tử nhớ

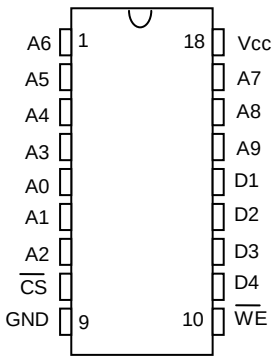
III.3 Tổ chức bộ nhớ cho hệ Vi xử lý

III.3.1 Tổ chức bộ nhớ vật lý

Tổ chức bộ nhớ cho một hệ Vi xử lý (máy vi tính) phụ thuộc không chỉ vào một hệ Vi xử lý cụ thể, mà còn phụ thuộc vào cách bố trí thuận lợi bên trong hệ thống. Trước hết, hãy làm quen với các khái niệm chip nhớ và từ nhớ để phân tích vấn đề tổ chức vật lý một bộ nhớ, sau đó mở rộng khái niệm tổ chức theo quan điểm của người lập trình (tổ chức logic).

Các chip nhớ được sản xuất dưới nhiều kích cỡ khác nhau, phụ thuộc vào công nghệ chế tạo. Chip nhớ là một vi mạch cụ thể, được bố trí các chân cơ bản như Hình III.8 Các chân của một chip nhớ thông thường gồm các lối vào của BUS địa chỉ, lối dữ liệu, các chân điều khiển chọn chip, ghi/đọc và các chân nguồn.

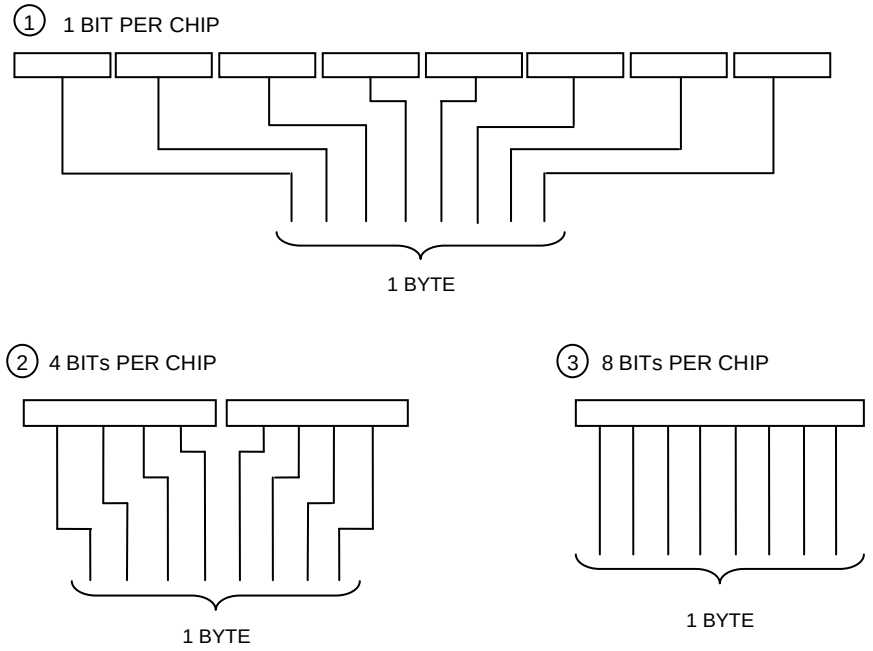
A0 ÷ A9	Các chân địa chỉ
D1 ÷ D4	Các chân dữ liệu
CS	Chân chọn chip
WE	Điều khiển Ghi/Đọc
Vcc	Chân nguồn nuôi +5V
GND	Chân nối đất



Hình III.8 Sơ đồ nối chân một vi mạch nhớ RAM 1Kx4

Tuỳ theo từng chip, số lượng chân địa chỉ và số lượng chân dữ liệu có thể khác nhau phụ thuộc vào độ dài *từ nhớ của chip* và *dung lượng* của chip nhớ. Độ dài từ nhớ của chip nhớ có thể là 1bit, 4 bits hoặc 8 bits, trong khi số chân địa chỉ có thể từ 10 trở lên tuỳ thuộc vào dung lượng

của chip nhớ. Trong trường hợp độ dài từ nhớ của chip là 1 bit, ta cần phải ghép song song 8 chip để tạo thành 1 byte, ghép song song 16 chip để tạo một từ word – 2 bytes).



Hình III.8 Tạo từ nhớ 8 bit từ các chíp nhớ có độ dài từ nhớ nhỏ hơn

III.3.2 Thiết kế vỉ nhớ cho hệ Vi xử lý

Thiết kế vỉ nhớ là một việc rất quan trọng và rất cần thiết trong việc xây dựng một hệ Vi xử lý. Các vỉ nhớ được thiết kế thông thường là EPROM, các loại vỉ nhớ RAM, từ các chip nhớ có sẵn. Thông thường, các chíp nhớ được chọn là những chip thông dụng trên thị trường, có các thông số kỹ thuật chủ yếu sau:

a. *Dung lượng nhớ của chip nhớ* tính theo đơn vị Kbyte
b. *Độ dài từ nhớ của chíp nhớ* tính theo số bits
c. *Một số thông số kỹ thuật khác như thời gian truy xuất, công suất tiêu tán của chip v.v…*Những thông số này không có ảnh hưởng lớn đến quá trình thiết kế và xây dựng vỉ nhớ.

Các thông số được cho trước trong việc thiết kế một vỉ nhớ bao gồm:

a. *Loại chip nhớ* ví dụ dùng EPROM 2764 (8Kx8) hay RAM TMS 2064 (8Kx8) v.v…
b. *Dung lượng của vỉ nhớ* là dung lượng vỉ nhớ phải có, ví dụ 64KB, 128KB v.v…
c. *Địa chỉ đầu của vùng nhớ* ví dụ vỉ nhớ có địa chỉ đầu là A0000H chẳng hạn.

Ví dụ minh hoạ: Dùng EPROM 2764 (8Kx8bit) xây dựng vỉ nhớ có dung lượng 32KB, địa chỉ đầu là 22000H.

Giải: Dựa trên yêu cầu của đề ra, phải thực hiện các bước sau:

1. Xác định số chip nhớ cần thiết để tạo từ nhớ cơ bản (độ dài 8 bits), có thể tính theo công thức:

$n = \frac{8}{k}$ trong đó *n* là số chip cần để tạo được từ nhớ cơ bản

k là độ dài từ nhớ của chip nhớ

Tín hiệu chọn vỏ $\overline{CS}$ của các chip này được nối chung với nhau, các chip này được coi như một chip liên thông, các bit dữ liệu sẽ được định vị theo thứ tự từ $D_7 \div D_0$ tương ứng với các bit từ $D_7 \div D_0$ của BUS dữ liệu.

2. Xác định số chip nhớ, hoặc số chip liên thông để tạo được dung lượng nhớ theo yêu cầu. Trong trường hợp cụ thể của đề ra, cần 4 chip để tạo được dung lượng nhớ 32KB. Tính theo công thức:

$M = \frac{Q}{D}$ trong đó *Q* là dung lượng của vỉ nhớ

D là dung lượng của chip nhớ hoặc dung lượng của chip liên thông

M là số chip nhớ hoặc số chip liên thông cần thiết

3. Xác định số dây địa chỉ cơ sở (tức là số dây địa chỉ thấp được nối trực tiếp vào chip nhớ hoặc chip liên thông): Số dây địa chỉ *m* phụ thuộc vào dung lượng nhớ của chip nhớ hoặc chip liên thông theo biểu thức sau:

$2^m = D$ trong đó *D* là dung lượng của chip nhớ

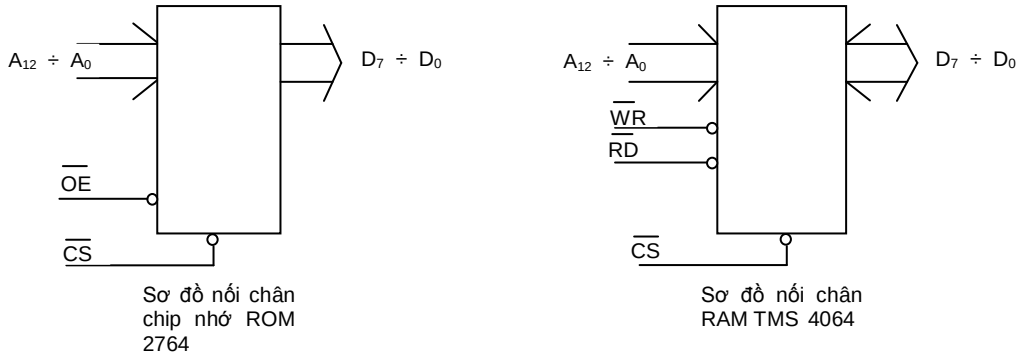
m là số dây địa chỉ cơ sở

4. Từ số chip hoặc số chip liên thông, xác định số dây địa chỉ cần thiết để tạo các dây chọn chip riêng biệt. Tính theo công thức:

$2^i = M$ trong đó *i* là số dây địa chỉ cần để giải mã *xác định các tín hiệu chọn chip cho các chip nhớ hoặc chip liên thông. M* là số lượng chip hoặc số lượng chip liên thông. Xây dựng mạch tổ hợp tạo các tín hiệu chọn chip CS_i .

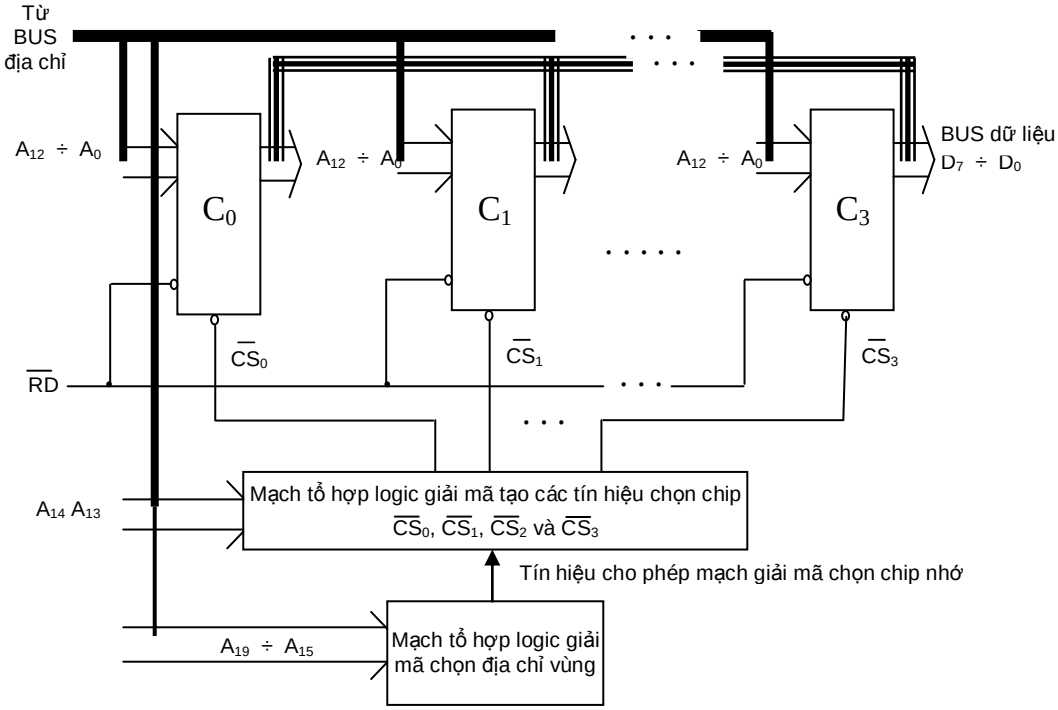
Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866

5. Các dây địa chỉ còn lại được sử dụng để tạo tín hiệu xác định vùng nhớ của vỉ nhớ trong không gian nhớ (được gán cho vỉ nhớ theo địa chỉ đầu của vỉ nhớ theo yêu cầu).



Hình II.9 Sơ đồ nối chân chip nhớ ROM và chip nhớ RAM

Sơ đồ khối vỉ nhớ như sau, các mạch tổ hợp logic xây dựng theo kiến thức học được ở môn học *Kỹ thật điện tử số.*



Hình II. 10 Sơ đồ khối vỉ nhớ 32KB từ các chip ROM 2764

CHƯƠNG IV. CÁC CHIP KHẢ LẬP TRÌNH

(Programmable)

IV.1 Tổng quan

Chip khả lập trình (Programmable) là một loại mạch điện tử chuyên dụng có khả năng thực hiện chức năng thông qua việc cung cấp các từ điều khiển (Control Word - CW) được CPU gửi tới (do người lập trình soạn). Nội dung các bit định chức năng trong từ điều khiển sẽ điều khiển mạch làm việc theo những chế độ định trước. Tồn tại một số mạch chức năng chuyên dụng tiêu biểu cho các hệ Vi xử lý µP8085 và họ các trung tâm vi xử lú 80x86 như mạch phối ghép ngoại vi song song khả lập trình PPI8255 điều khiển việc phối ghép vào/ra dữ liệu song song giữa CPU với các thiết bị ngoại vi, mạch đếm định thời và tạo khoảng thời gian PIT8253/54, mạch phối ghép vào ra dữ liệu nối tiếp USART 8251, mạch điều khiển ngắt PIC8259 v.v… Phần tiếp theo sẽ tìm hiểu một số mạch tiêu biểu.

IV.2 Một số mạch chức năng tiêu biểu

IV.2.1 Mạch vào/ra dữ liệu song song PPI-8255 (Programmable Peripheral Interface).

a) Giới thiệu chung

PPI8255 là mạch giao diện thiết bị ngoại vi khả lập trình, được thiết kế để làm việc trong hệ thống vi tính của hãng Intel. PPI8255 thực hiện chức năng giao diện song song giữa các thiết bị ngoại vi và máy vi tính.

Cấu hình hoạt động của PPI8255 có thể lập trình được bằng phần mềm. PPI8255 thường được dùng để chế tạo các mạch vào/ra dữ liệu số dạng song song.

Sơ đồ khối các thành phần chức năng của mạch PPI8255 được thể hiện trên Hình IV. 1, gồm một đệm BUS dữ liệu, khối điều khiển ghi/đọc, hai khối điều khiển hai nhóm cổng A và B, và các cổng 8bits PA, PB và PC.

Đệm BUS dữ liệu: là bộ đệm 8 bits hai chiều 3 trạng thái. Dữ liệu được phát hoặc nhận qua bộ đệm này. Từ điều khiển và trạng thái cũng được truyền từ CPU đến PPI8255 qua bộ đệm này.

Logic điều khiển và ghi/đọc: logic điều khiển và ghi/đọc quản lý toàn bộ các quá trình truyền dữ liệu và điều khiển các cổng PA, PB, PC.

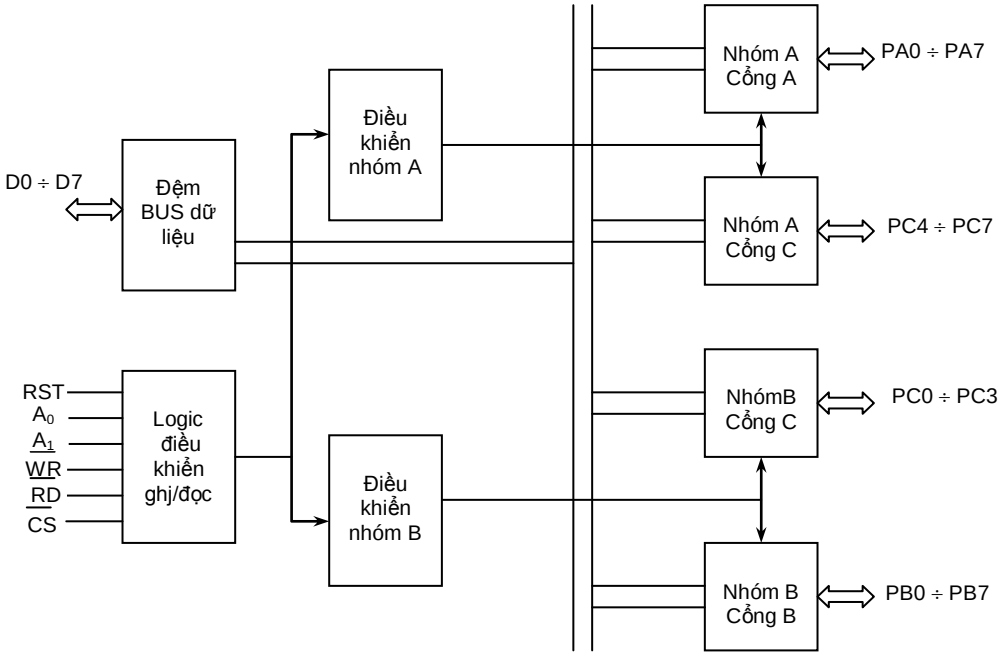
Các tín hiệu điều khiển từ CPU: Tín hiệu đọc RD, tín hiệu ghi WR và tín hiệu tái thiết lập theo mặc định RST.

RST – Reset: tín hiệu RST đặt tất cả 3 cổng A, B, C ở chế độ đầu vào.

Các tín hiệu $\overline{RD}$, $\overline{WR}$, A1, A0:

Địa chỉ A0, A1 phối hợp với tín hiệu $\overline{RD}$, $\overline{WR}$ điều khiển việc ghi/đọc đối với 3 cổng A, B, C:

Các cổng PA, PB và PC : các cổng PA, PB và PC là các cổng vào/ra dữ liệu loại 8 bit. Chức năng của từng cổng được xác định bằng phần mềm (bằng từ điều khiển).



Hình IV.1 Cấu trúc theo khối chức năng PPI8255

A1	A0	$\overline{RD}$	$\overline{WR}$	Thao tác
0	0	0	1	BUS dữ liệu ⇐ Cổng A
0	1	0	1	BUS dữ liệu ⇐ Cổng B
1	0	0	1	BUS dữ liệu ⇐ Cổng C
0	0	1	0	BUS dữ liệu ⇒ Cổng A
0	1	1	0	BUS dữ liệu ⇒ Cổng B
1	0	1	0	BUS dữ liệu ⇒ Cổng C
1	1	1	0	Thanh ghi điều khiển

- Cổng A: là cổng ra 8 bit có chốt dữ liệu hoặc là cổng vào 8 bit.
- Cổng B: là cổng ra 8 bit có chốt dữ liệu hoặc là cổng vào 8 bit.
- Cổng C được chia thành 2 phần :

  4 bit cao (PC7÷PC4) cùng với cổng A làm thành nhóm A.

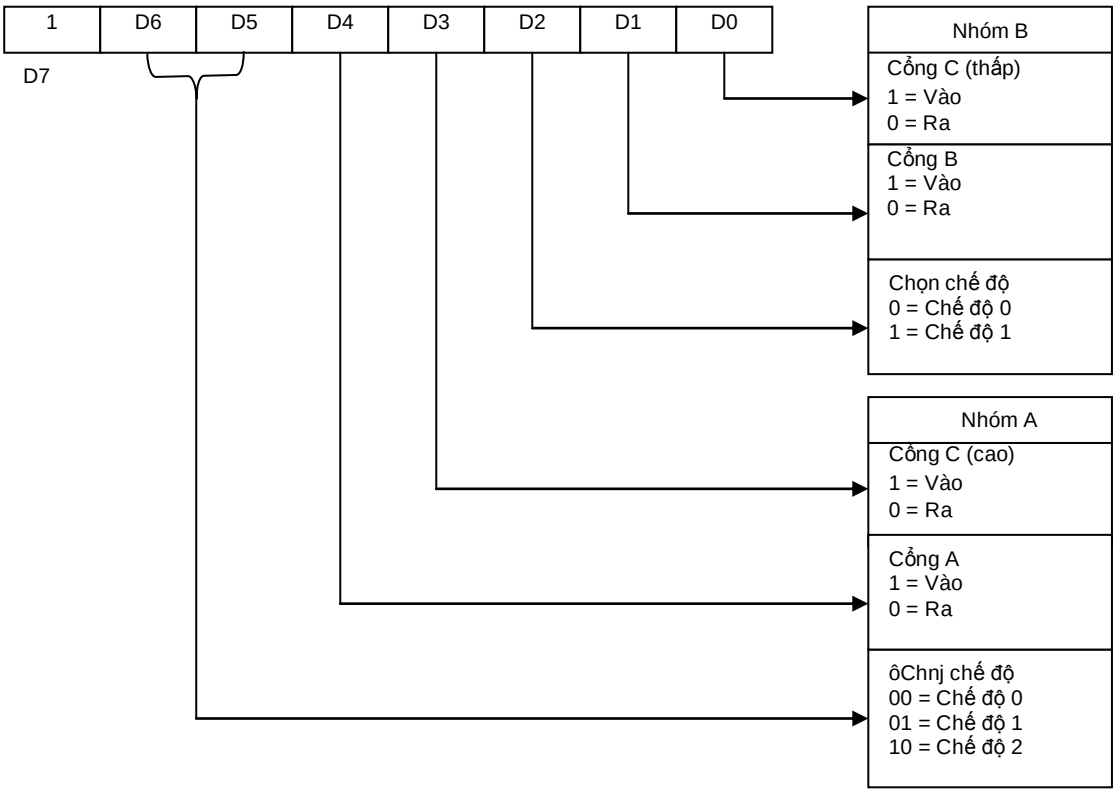
  4 bit thấp (PC3÷PC0) cùng với cổng B làm thành nhóm B.

Tuỳ theo chế độ hoạt động (được xác lập thông qua từ điều khiển) mà hai phần này có thể thực hiện chức năng vào/ra dữ liệu 4 bit hoặc nhận/phát tín hiệu bắt tay cho từng nhóm tương ứng

Mạch PPI8255 có 3 chế độ làm việc

- Chế độ 0: vào/ra cơ bản
- Chế độ 1: vào/ra có xung chốt dữ liệu
- Chế độ 2: vào/ra hai chiều (chỉ cho nhóm A)

b) Chế độ làm việc và từ điều khiển



Hình IV. 2 Cấu trúc từ lệnh của PPI 8255

Có thể chọn và đặt lại chế độ làm việc của PPI8255 qua các từ điều khiển.

Khuôn dạng từ điều khiển chế độ làm việc được mô tả trên hình IV.2.

+ **Chế độ 0:** vào/ra cơ bản , ra có chốt, vào không chốt dữ liệu.

Từ điều khiển:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	1/0	x	x	x	x

Tính chất cơ bản của chế độ 0:

- Hai cổng 8 bit
- Hai cổng 4 bit
- Ra có chốt

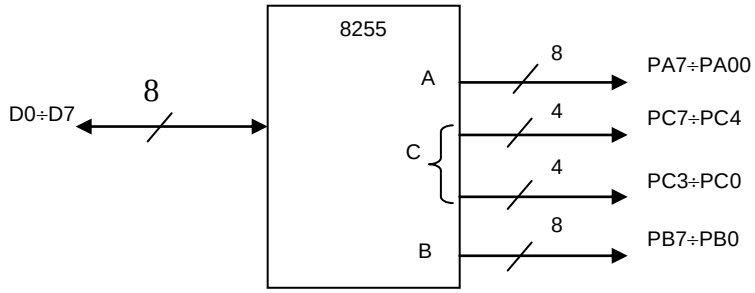
Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866

- Vào không chốt
- Cho phép chọn và dùng 1 trong 16 cấu hình cổng vào/ra

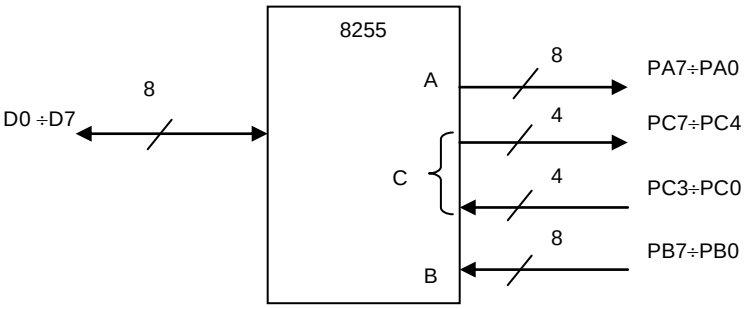
Ví dụ:

Khi đặt từ điều khiển #0, là 10000000, cấu hình các cổng của 8255 được đặt như sau:

Tất cả các cổng đều ở chế độ *Output như ở hình vẽ*



Khi đặt từ điều khiển #3, là 10000011, cấu hình các cổng của 8255 như sau



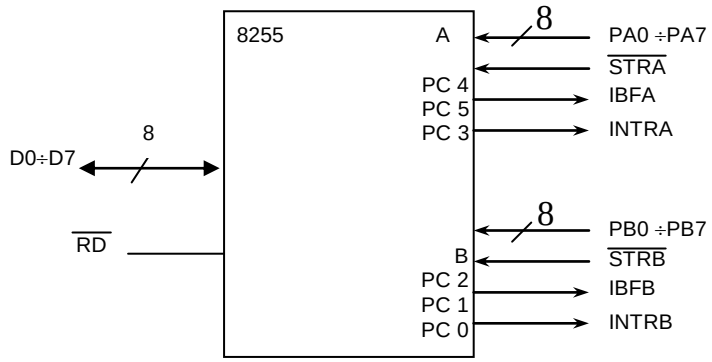
+ **Chế độ 1 :** vào/ ra dữ liệu có xung chốt dữ liệu.

Đặc tính của chế độ 1: có hai nhóm A và B, mỗi một nhóm có một cổng vào/ra 8 bit và một cổng điều khiển 4 bit.

Cấu hình cổng vào dữ liệu (chế độ 1):

Từ điều khiển:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	X	1	1	X



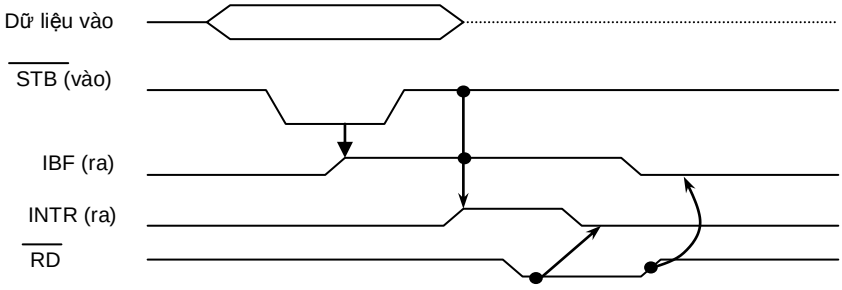
Các tín hiệu điều khiển:

$\overline{STRA}/\overline{STRB}$: mức tích cực thấp chốt dữ liệu vào 8255

IBFA/IBFB *(Input Buffer Full):* mức tích cực cao báo dữ liệu đã được chốt trong 8255

INTRA/INTRB *(Interrupt Request):* yêu cầu ngắt

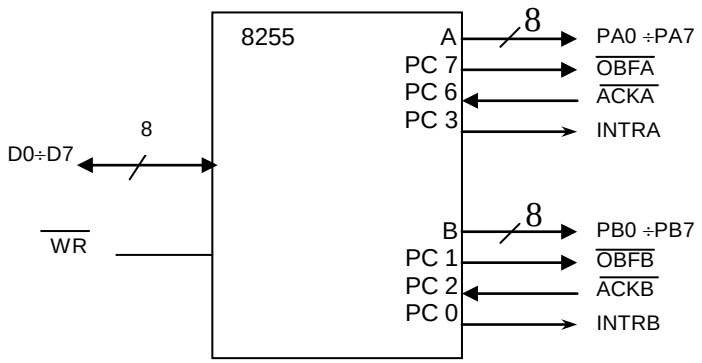
INTEA và INTEB (Interrupt Enable) : được đặt/xoá (1/0) bởi bít PC4 và PC2



Cấu hình cổng ra dữ liệu (chế độ 1)

Từ điều khiển:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	0	X	1	0	X

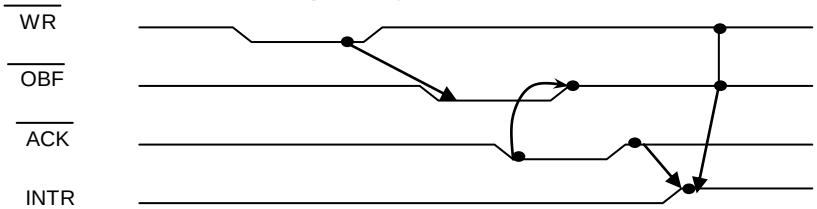


Các tín hiệu điều khiển:

$\overline{OBFA}/\overline{OBFB}$ *(Output Buffer Full):* tín hiệu ra, mức tích cực thấp khi có dữ liệu ra ở các cổng A/B.

$\overline{ACKA}/\overline{ACKB}$ *(Acknowledge)* : tín hiệu vào, mức tích cực thấp, báo 8255 là dữ liệu ra ở cổng A/B đã được nhận.

INTRA/INTB: yêu cầu ngắt, yêu cầu đưa dữ liệu (tiếp theo) ra 8255 theo tín hiệu báo ngắt này.



+ **Chế độ 2 :** Vào/ ra hai chiều có xung chốt dữ liệu (riêng cho nhóm A)

Đặc tính chế độ 2: chỉ được dùng cho nhóm A. Cổng A là cổng vào/ra 8bit hai chiều. Cổng C có 5 bit được dùng làm các tín hiệu điều khiển bắt tay. Vào /ra dữ liệu đều được chốt.

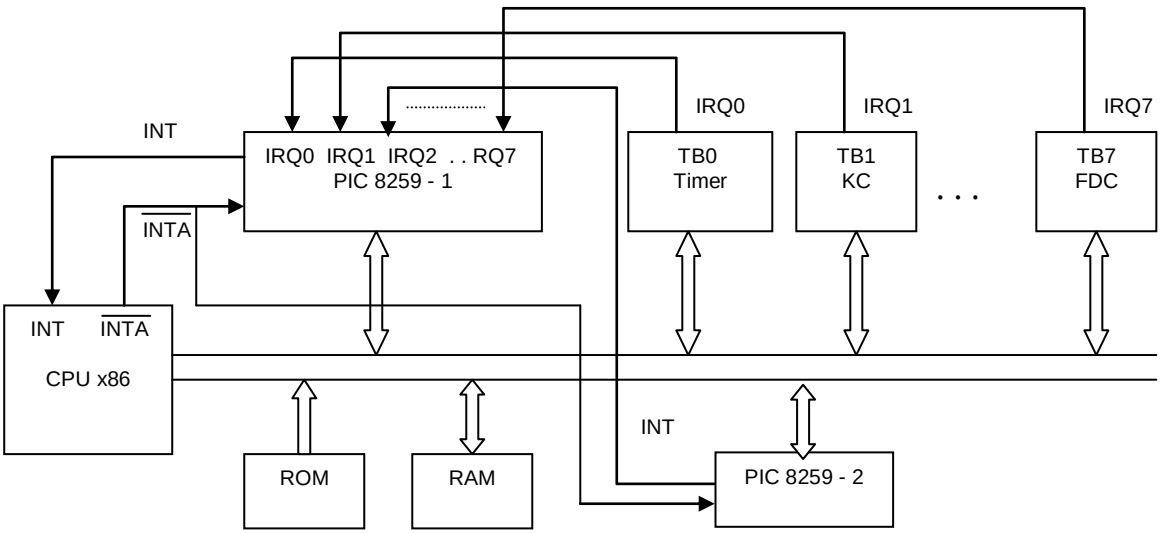
Khả năng ứng dụng: chế độ 2 cung cấp công cụ truyền tin với thiết bị ngoại vi theo cách phát và nhận dữ liệu 8 bit song song trên cùng một đường BUS. Quá trình truyền tin thuộc kiểu không đồng bộ. Các tín hiệu "bắt tay" STB, IBF, OBF, ACK được dùng để phối hợp việc truyền dữ liệu giữa máy tính và thiết bị ngoại vi.

Các bit của cổng C có thể được thiết lập lên "1" (set) hay về "0" (reset) bằng cách ghi từ điều khiển với D_7 = "0", việc chọn bit cần SET hay RESET qua chọn bit D_0 = "1" hoặc "0" và vị trí của bit cổng C thông qua các bit D_1, D_2 và D_3 như sau: 000 = bit PC_0 , 001 = bit PC_1 … và 111 = bit PC_7

D7	D6	D5	D4	D3	D2	D1	D0
0	x	x	x	B2	B1	B0	S/R

IV.2.2 Mạch điều khiển ngắt PIC-8259

CPU được thiết kế để đáp ứng được với các quá trình ngắt cứng. CPU có một đầu vào nhận tín hiệu ngắt INT, khi nhận được tín hiệu này CPU sẽ phản ứng theo cơ chế ngắt cứng.



Hình IV.3 – Sơ đồ ghép nối PIC8259 trong hệ Vi xử lý

Trong thực tế có nhiều thiết bị ngoại vi yêu cầu được phục vụ theo phương pháp ngắt cứng (bàn phím, đồng hồ hệ thống, máy in, v.v.) và sinh ra nhiều yêu cầu ngắt, do vậy cần có một bộ điều khiển giúp CPU

quản lý và phục vụ các yêu cầu ngắt, đó là bộ điều khiển ngắt PIC8259 (Programmable Interrupt Controller).

Cấu trúc hệ thống ngắt cứng :

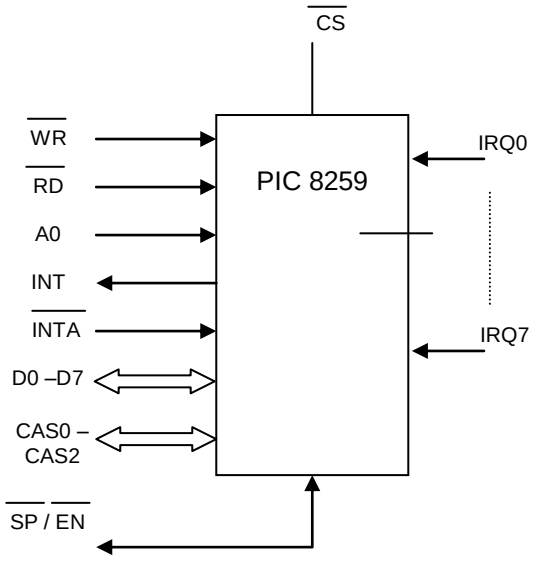
Hệ thống ngắt cứng được xây dựng trên cơ sở 2 bộ điều khiển ngắt PIC 8259, mỗi PIC 8259 có thể nhận 8 tín hiệu yêu cầu ngắt IRQ từ thiết bị vào/ra. Hai PIC này được kết nối với nhau theo kiểu ghép tầng, kết hợp hoạt động để có thể phục vụ được 16 yêu cầu ngắt IRQ.

Chức năng cơ bản của PIC 8259 : PIC 8259 là một vi mạch điện tử khả trình được thiết kế để giúp CPU thực hiện quá trình ngắt cứng. PIC 8259 thực hiện các chức năng sau:

- Ghi nhận được 8 yêu cầu ngắt IRQi , i=0,1,...,7 .
- Cho phép chọn và phục vụ các yêu cầu ngắt theo mức ưu tiên.
- Cung cấp cho CPU số ngắt tương ứng với yêu cầu ngắt IRQi . Số ngắt này đại diện cho địa chỉ của chương trình con phục vụ thiết bị yêu cầu ngắt IRQi.
- Cho phép hoặc không cho phép các yêu cầu ngắt IRQi kích hoạt hệ thống ngắt.

a) Thiết bị điều khiển ngắt PIC 8259 và cơ chế hoạt động của hệ thống ngắt cứng

Cấu trúc bên ngoài của PIC 8259 :



Cấu trúc bên trong của PIC 8259 :

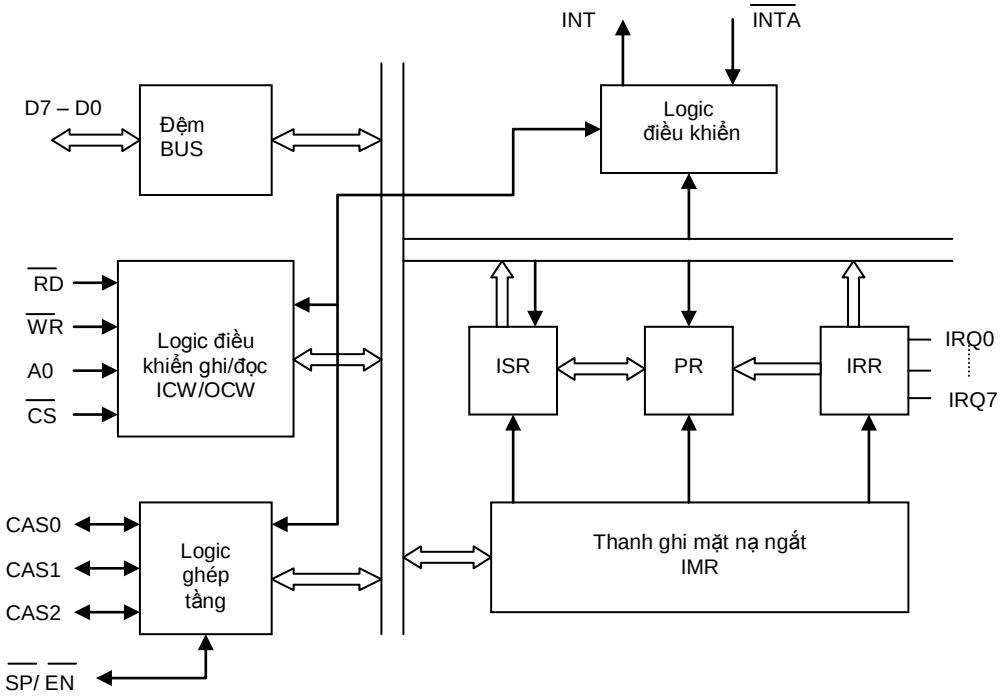
Các khối chức năng:

- IRR (Interrupt Request Register - Thanh ghi yêu cầu ngắt) : là thanh ghi 8 bit. IRR chứa (ghi nhận) tất cả các yêu cầu ngắt

Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866

IRQi đòi phục vụ. Nếu tín hiệu IRQi = "1" thì bit IRRi tương ứng được đặt bằng "1".

– PR (Priority Resolver- Bộ giải quyết ưu tiên): là thanh ghi 8 bit. PR cho phép xác lập mức ưu tiên của các yêu cầu ngắt. Ngắt có ưu tiên cao nhất được chọn và đặt vào bit tương ứng trong ISR trong chu kỳ INTA.



– ISR (In Service Register - Thanh ghi ngắt đang được phục vụ) : là thanh ghi 8 bit. ISR ghi nhận các ngắt đang được phục vụ. Yêu cầu ngắt IRQi nào đang được phục vụ thì bit ISRi tương ứng được đặt bằng "1".
– Khối logic điều khiển : khối logic điều khiển đưa ra tín hiệu INT, được nối thẳng với chân INT của CPU. Khi INT có mức cao là đòi CPU phục vụ ngắt. Khối logic điều khiển nhận tín hiệu INTA từ CPU. Khi nhận được tín hiệu INTA, PIC 8259 sẽ cung cấp số ngắt ra BUS dữ liệu cho CPU .
– Khối đệm BUS: là loại 8 bit, 2 hướng, 3 trạng thái. Các từ điều khiển ICW, OCW được đưa vào PIC 8259 qua khối này để xác lập chế độ hoạt động của 8259. Số ngắt và trạng thái hoạt động của PIC cũng được đưa ra BUS dữ liệu qua khối này.
– Khối ghép tầng
– PIC 8259 có cơ cấu cho phép nối ghép tầng các PIC 8259 với nhau và phối hợp hoạt động của các PIC này. Tầng thứ nhất có đầu ra INT nối trực tiếp với CPU, gọi là PIC 8259-chủ. Đầu vào IRQi của PIC chủ được nối với đầu ra INT của PIC 8259 thứ

hai. PIC này được gọi là PIC 8259-thợ . Cơ chế ghép tầng cho phép xây dựng một hệ thống ngắt cứng quản lý được đến 64 yêu cầu ngắt IRQ.

- Khối logic ghi/đọc và giải mã: thực hiện giải mã các từ điều khiển ICW (Initialization Command Word - Từ điều khiển khởi động) và OCW (Operation Command Word - Từ điều khiển hoạt động). Qua hai loại từ điều khiển này người sử dụng có thể lập trình xác lập chế độ hoạt động cho PIC.
- Thanh ghi IMR : là thanh ghi 8 bit, chứa mặt nạ ngắt.

Bảng các tín hiệu $\overline{CS}$, A0, $\overline{RD}$, $\overline{WR}$, và cách ghi/đọc PIC 8259.

CS	A0	RD	WR	D4	D3	Hướng thông tin
0	0	0	1	X	X	IRR, ISR => BUS
0	1	0	1	X	X	(IMR) = 0CW1 => BUS
0	0	1	0	0	0	BUS => 0CW2
0	0	1	0	0	1	BUS => 0CW3
0	0	1	0	1	X	BUS => ICW1
0	1	1	0	X	X	BUS => ICW2, ICW3, ICW4, 0CW1

b) Cơ chế hoạt động của hệ thống ngắt cứng :

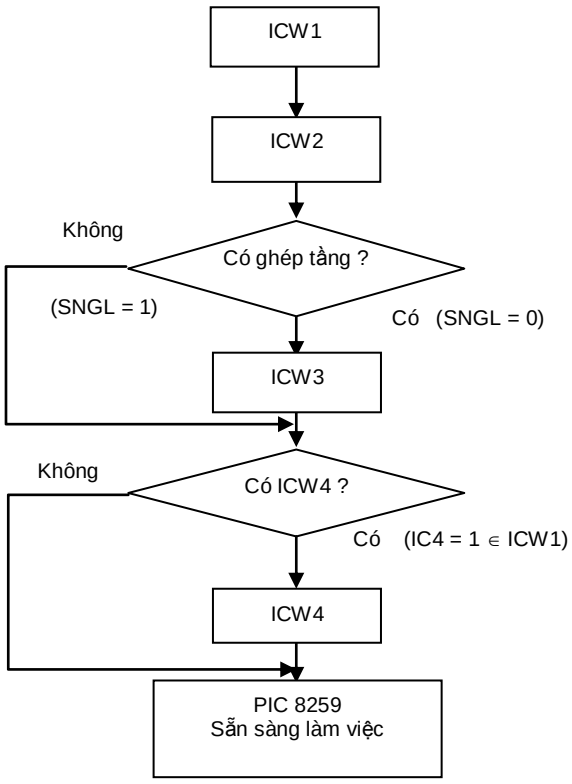
Điều kiện ban đầu : PIC 8259 cần được lập trình khởi động qua các từ điều khiển ICW. Sau khi các từ điều khiển ICW được nạp thì PIC 8259 sẵn sàng hoạt động.

- Một hoặc nhiều thiết bị vào-ra có yêu cầu được phục vụ phát tín hiệu IRQi = "1" (mức tích cực) cho PIC. PIC ghi nhận các yêu cầu ngắt IRQi này bằng cách đặt các bit IRRi tương ứng lên "1".
- PIC 8259 chọn IRQi có mức ưu tiên cao nhất để phục vụ. PIC gửi tín hiệu INT cho CPU, đòi CPU phục vụ.
- CPU thực hiện các thao tác sau :
  - Thực hiện nốt lệnh của quá trình hiện hành .
  - Lưu địa chỉ trở về (nội dung của các thanh ghi CS, IP) và thanh ghi cờ FLAGS vào ngăn xếp.
  - Gửi hai tín hiệu trả lời ngắt INTA cho PIC .
- Khi PIC 8259 nhận được tín hiệu INTA thứ 1 : bít ISRi ứng với IRQi có mức ưu tiên cao nhất được thiết lập (ISRi=1) và bit IRRi tương ứng bị xóa (IRRi=0). Trong chu kỳ INTA thứ nhất này PIC 8259 không gửi gì cho CPU qua BUS dữ liệu.
- Khi PIC 8259 nhận được tín hiệu INTA thứ 2: PIC 8259 gửi số ngắt tương ứng với IRQi đang được phục vụ qua BUS dữ liệu cho CPU.

- CPU nhận số ngắt và trên cơ sở số ngắt này vào vị trí tương ứng trong Bảng véc tơ ngắt để xác định địa chỉ của chương trình phục vụ ngắt. CPU nạp địa chỉ chương trình phục vụ ngắt vào các thanh ghi CS và IP và bắt đầu thực hiện chương trình phục vụ ngắt này.
- Khi thực hiện xong chương trình phục vụ ngắt thì quá trình phục vụ ngắt của CPU cũng kết thúc. Hệ thống ngắt cứng có thể kết thúc phục vụ ngắt hiện thời theo hai chế độ:
  - Kết thúc ngắt bình thường EOI (End Of Interrupt): khi PIC được đặt chế độ kết thúc ngắt bình thường EOI thì CPU phải phát lệnh báo kết thúc ngắt EOI (qua OCW2) cho PIC trước khi rời khỏi chương trình con phục vụ ngắt. Khi đó bit ISRi của ngắt đang được phục vụ sẽ được đặt xuống 0.
  - Kết thúc ngắt tự động AEOI (Automatic EOI) : khi PIC được đặt chế độ kết thúc ngắt tự động AEOI thì tại chu kỳ INTA thứ 2 bit ISRi của ngắt đang được phục vụ sẽ được đặt xuống 0.

Bằng các cách nói trên hệ thống ngắt cứng có thể tiếp tục phục vụ yêu cầu ngắt này ở những lần tiếp theo.

c) Lập trình khởi động PIC 8259 và các từ điều khiển khởi động ICW



Cần xác lập chế độ làm việc của PIC 8259 trước khi sử dụng. Quá trình này được gọi là lập trình khởi động thiết bị. Việc lập trình khởi động PIC 8259 được thực hiện qua các từ điều khiển ICW và theo lưu đồ sau :
Các bit D5 - D7 không dùng cho CPU x86.

– IC4 (bit D0) : Cho biết có cần ICW4 ?

IC4 = 0 : không cần ICW4.
IC4 = 1 : có ICW4.

a- ICW1

D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	1	LTIM	ADI	SNGL	IC4

+ SNGL (bit D1) : cho biết hệ thống ngắt chỉ có một PIC hay có nhiều PIC ghép tầng.

SNGL = 0 có ghép tầng
SNGL = 1 chỉ có một PIC 8259

+ ADI (bit D2): không dùng cho hệ CPU x86
+ LTIM : xác định dạng tín hiệu IRQ

LTIM = 1 IRQ phải là tín hiệu mức TTL
LTIM = 0 IRQ phải là tín hiệu dạng sườn xung.

+ D4 = 1
+ D5 = D6 = D7 = 0

a- ICW2:

– ICW2 định nghĩa số ngắt nền cho 7 số ngắt còn lại .

D7	D6	D5	D4	D3	D2	D1	D0
T7	T6	T5	T4	T3	x	x	x

Các bit T7 - T3 là 5 bit cao của số ngắt, 3 bit còn lại liên quan đến các đầu vào IRQi

Năm bit cao T7 - T3 (do người sử dụng tùy chọn) cùng với 3 bit thấp nhất bằng 0 xác định số ngắt nền . Dựa trên số ngắt nền ứng với IRQ_0 này, PIC 8259 tự tạo ra các số ngắt tiếp theo tương ứng với các IRQ_1 đến IRQ_7.

Ví dụ : ở hệ thống ngắt cứng của máy vi tính PC, các số ngắt do PIC 8259-chủ cung cấp như sau :

0	0	0	0	1	0	0	0	ứng với IRQ0
0	0	0	0	1	0	0	1	ứng với IRQ1
.....								.....
0	0	0	0	1	1	1	1	ứng với IRQ7

– ICW3: liên quan đến ghép tầng.

Mạch phần cứng có chân SP/EN xác định chủ/thợ ở chế độ ghép tầng : nếu SP = 1 thì PIC là chủ, nếu SP = 0 thì PIC là thợ.

Có hai loại ICW3

– ICW3 cho PIC chủ : xác định đầu vào IRQi nhận tín hiệu INT từ PIC thợ thứ i.

7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0

Nếu Si = 1 báo có PIC thợ nối vào chân IRQi của chủ

– ICW3 cho PIC thợ : xác định địa chỉ (chỉ thị nhận dạng) của PIC thợ.

					ID2	ID1	ID0

Các bit ID2, ID1, ID0 xác định địa chỉ riêng của các PIC 8259-thợ. Khi nhận được tín hiệu INTA2 , PIC 8259-thợ so sánh các tín hiệu CAS0 - CAS2 (phát ra từ PIC 8259-chủ) với ID2 - ID0, nếu chúng giống nhau thì PIC 8259 - thợ gửi số ngắt lên BUS dữ liệu cho CPU, ngược lại thì không gửi.

– ICW4:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	SFNM	BUFF	M/S	AEOI	μP

+ Bit μp: báo cho PIC 8259 biết phải làm việc và họ vi xử lý nào.

μp = 1 : làm việc với họ x86

μP = 0 : làm việc với họ 8085

+ Bit AEOI : xác lập chế độ kết thúc ngắt.

AEOI = 0 : kết thúc bình thường EOI

AEOI = 1 : kết thúc tự động AEOI

+ Bit BUFF: báo chế độ có bộ đệm BUS

BUFF = 1: PIC làm việc ở chế độ đệm BUS, lúc này tín hiệu SP/EN ở chế độ ra và việc định nghĩa chủ/thợ được xác định bằng bit M/S.

+ Bit M/S: xác định chủ/thợ

M/S = 1 : PIC là chủ

M/S = 0 : PIC là thợ.

Nếu BUFF = 0 thì M/S không có ý nghĩa.

+ Bit SFNM: bit này được đặt bằng 0 ngay khi khởi động hệ thống. Kiểu ưu tiên cố định là mặc định, trong đó IRQ0 có mức ưu tiên cao nhất, IRQ7 có mức ưu tiên thấp nhất. Có thể thay đổi kiểu ưu tiên bằng từ điều khiển OCW2. Trong kiểu ưu tiên cố định, khi SFNM = 0, khi bit ISRi = 1 thì tất cả các IRQi có mức ưu tiên thấp hơn đều bị cấm. Chỉ có các IRQi có mức ưu tiên cao hơn được phép gây ngắt chương trình phục vụ ngắt hiện thời.

d) Các từ điều khiển hoạt động OCW

Các từ điều khiển OCW được dùng để xác lập các chế độ làm việc cụ thể trong quá trình hoạt động của PIC 8259. Có thể gửi các từ OCW này cho PIC8259 vào bất kỳ lúc nào sau khi khởi động hệ thống ngắt.

+ OCW1: cho phép hoặc cấm nhận một yêu cầu ngắt IRQ_i nào đó bằng mặt nạ ngắt .

+ Với PIC chủ : địa chỉ thanh ghi chứa OCW1 là 21H

Với PIC thợ : địa chỉ thanh ghi chứa OCW1 là A1H

D7	D6	D5	D4	D3	D2	D1	D0
M7	M6	M5	M4	M3	M2	M1	M0

Mỗi bit Mi tương ứng với IRQi

Khi Mi = 1 mặt nạ ngắt được đặt , cấm PIC nhận IRQi(cấm IRQi gây ngắt)

Khi Mi = 0 mặt nạ ngắt được xoá, cho phép PIC nhận IRQi (cho phép IRQi gây ngắt)

Hệ điều hành đặt mặt nạ che chắn các IRQ mà hệ thống chưa dùng đến.

+ OCW2 : dùng để đổi kiểu ưu tiên và báo kết thúc ngắt EOI.

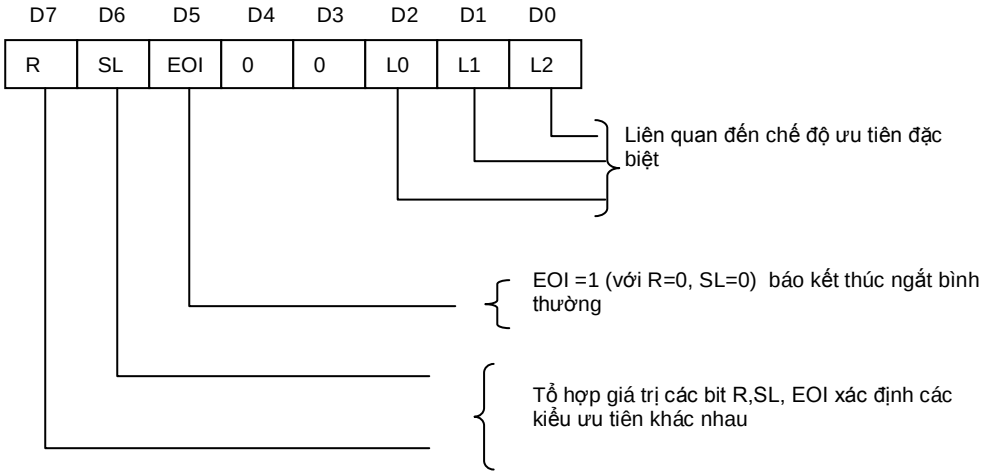
PIC cho phép chọn một trong ba chế độ ưu tiên:

Ưu tiên cố định: IRQ_0 có mức ưu tiên cao nhất, IRQ_7 có mức ưu tiên thấp nhất. Trong chế độ này IRQ mức cao có quyền ngắt chương trình phục vụ ngắt có mức ưu tiên thấp hơn.

Ưu tiên quay vòng: IRQi nào vừa được phục vụ thì bit ISRi sẽ bị xoá xuống 0 và tự động có mức ưu tiên thấp nhất. Điều này thực tế đã tạo ra các mức ưu tiên bằng nhau.

Ưu tiên đặc biệt : người lập trình có thể thay đổi mức ưu tiên bằng chương trình. Nếu các bit trong OCW2 R=1, SL=1 thì các bit L2-L0 sẽ đặt IRQn xuống mức thấp nhất và $IRQn_{+1}$ lên mức cao nhất.

Địa chỉ thanh ghi chứa OCW2 : 20H (PIC chủ), A0H (PIC thợ)



+ OCW3 cho phép đặt/đọc ISR và các thanh ghi khác của PIC 8259. ESMM =1 và SMM cho phép đặt/xoá chế độ mặt nạ đặc biệt. Chế độ mặt nạ đặc biệt này chỉ cấm một IRQ và cho phép tất cả các IRQ còn lại được yêu cầu ngắt.

- D4 = 0 , D3 = 1
- bit P : cho phép PIC 8259 làm việc với CPU ở chế độ hỏi đáp, không cần qua các tín hiệu INTR, INTA. Nếu P=1 thì PIC coi tín hiệu điều khiển đọc RD như là tín hiệu INTA .
- Các bit RR và RIS :

RR = 1 & RIS = 0: báo sẽ đọc IRR ở lệnh đọc tiếp sau
RR = 1 & RIS = 1: báo sẽ đọc ISR ở lệnh đọc tiếp sau.

e) Phân bố chức năng các yêu cầu ngắt và số ngắt trong máy PC

- PIC 8259-chủ :

PIC 8259-chủ chiếm hai địa chỉ cổng : 20H, 21H

- PIC 8259-thợ :

PIC 8259-thợ chiếm hai địa chỉ cổng : A0H, A1H

IRQi	Số ngắt	Thiết bị yêu cầu ngắt
IRQ_0	08H	Bộ tạo xung nhịp đồng hồ hệ thống
IRQ_1	09H	Thiết bị giao diện bàn phím
IRQ_2	0AH	PIC 8259-thợ
IRQ_3	0BH	Thiết bị giao diện vào/ra nối tiếp 2 (COM 2)
IRQ_4	0CH	Thiết bị giao diện vào/ra nối tiếp 1 (COM 1)
IRQ_5	0DH	Dự phòng
IRQ_6	0EH	Thiết bị giao diện ổ đĩa mềm FDC
IRQ_7	0FH	Thiết bị giao diện vào/ra song song (LPT1)

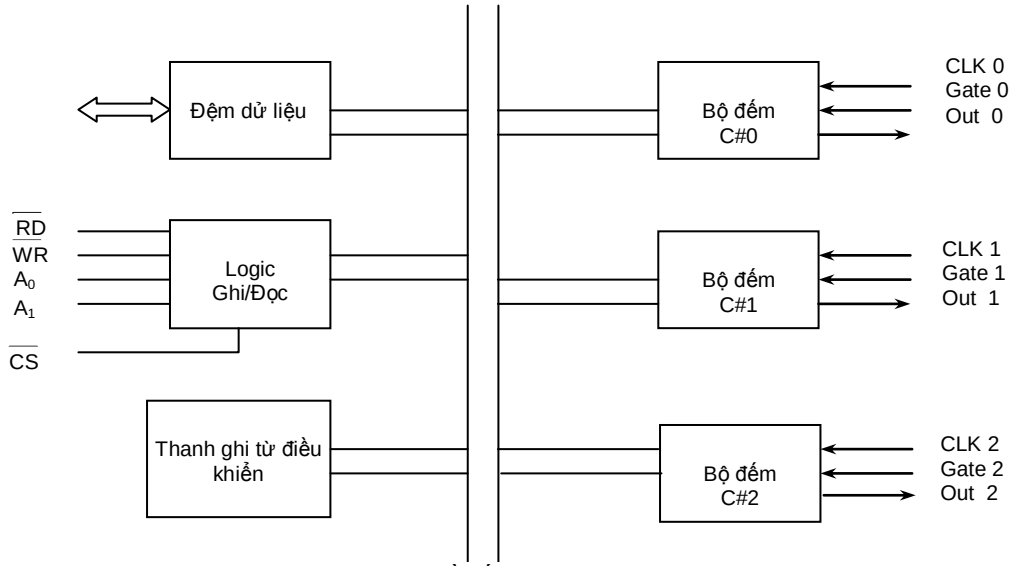
Dây IRQ	Số ngắt	Thiết bị yêu cầu ngắt
IRQ_8	70H	Đông hồ thời gian thực
IRQ_9	71H	Dự phòng
IRQ_{10}	72H	Card âm thanh
IRQ_{11}	73H	Thiết bị giao diện vào/ra USB
IRQ_{12}	74H	Thiết bị giao diện chuột PS/2
IRQ_{13}	75H	Bộ đồng xử lý x87
IRQ_{14}	76H	Bộ điều khiển BUS IDE 1(primary)
IRQ_{15}	77H	Bộ điều khiển BUS IDE 2 (secondary)

IV.3.3 Mạch đếm định thời đa năng PIT-8253 (Programmable Interval Timer)

Vi mạch PIT-8253 là mạch đếm định thời, tạo xung có độ rộng thay đổi đa năng và thu thập tín hiệu dạng xung., được thiết kế để sử dụng với hệ vi xử lý dòng Intel. Mạch 8523 thực hiện được nhiều chức năng. Các chức năng được xác định bằng phần mềm thông qua từ diều khiển.

Các chức năng chính của PIT-8253:

- Tạo khoảng thời gian chính xác
- Phát xung với tần số *f* lập trình được
- Đếm sự kiện
- Chia tần số
- Đồng hồ thời gian thực
- Phát xung đơn



Hình IV.4 – Sơ đồ cấu trúc bên trong PIT-8255

Đệm dữ liệu: là bộ đệm 8 bit, hai chiều, 3 trạng thái, được sử dụng để giao diện với BUS của máy tính.

Logic đọc/ghi: logic Ghi / Đọc nhận các tín hiệu từ hệ thống BUS , từ đó điều khiển việc truy nhập các thanh ghi của PIT-8253.

Thao tác chọn bộ đếm và ghi/đọc bộ đếm:

A1	A0	RD	WR	Công việc được thực hiện
0	0	1	0	Nạp bộ đếm C#0
0	1	1	0	Nạp bộ đếm C#1
1	0	1	0	Nạp bộ đếm C#2
1	1	1	0	Ghi từ điều khiển
0	0	0	1	Đọc bộ đếm C#0
0	1	0	1	Đọc bộ đếm C#1
1	0	0	1	Đọc bộ đếm C#2

Thanh ghi điều khiển: thanh ghi điều khiển nhận từ điều khiển xác định chế độ hoạt động cho PIT-8253.

Bộ đếm C#0, C#1, C#2:

Mạch PIT-8253 có 3 bộ đếm, mỗi một bộ đếm là loại 16bit , đếm lùi, tự khởi động lại. Mỗi một bộ đếm có thể được lập trình và hoạt động độc lập. Có thể đọc nội dung của từng bộ đếm ngay trong khi đang hoạt động. Bằng từ điều khiển có thể chọn chế độ làm việc cho các bộ đếm (6 chế độ).

1. Từ điều khiển.

Từng bộ đếm của PIT-8253 có thể được lập trình hoạt động độc lập bằng cách ghi từ điều khiển vào thanh ghi từ điều khiển (A0=1, A1=1).

Khuôn dạng từ điều khiển.

DD7	DD6	DD5	DD4	DD3	DD2	DD1	DD0
SCSC1	SCSC0	RLRL1	RLRL0	MM2	MM1	MM0	BCDCD

SC (Select Counter): chọn bộ đếm.

SC1	SC0	
0	0	Bộ đếm 0
0	1	Bộ đếm 1
1	0	Bộ đếm 2
1	1	Không hợp lệ

RL (Read/Load): Xác định Đọc/Nạp bộ đếm.

RLRL1	RLRL0	
0	0	Thao tác chốt bộ đếm Cho phép đọc nội dung bộ đếm trong quá trình đếm
1	0	Đọc hoặc nạp Byte cao
0	1	Đọc hoặc nạp Byte thấp
1	1	Đọc hoặc nạp Byte thấp trước, Byte cao sau

M (Mode): chế độ làm việc

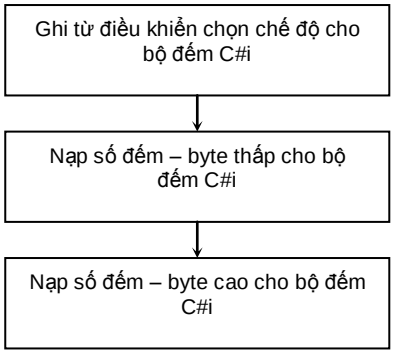
MM2	MM1	MM0	
0	0	0	Chế độ 0
0	0	1	Chế độ 1
X	1	0	Chế độ 2
X	1	1	Chế độ 3
1	0	0	Chế độ 4
1	0	1	Chế độ 5

BCD : kiểu mã đếm

BCD	Kiểu mã đếm
0	Mã nhị phân 16 bit
1	Mã BCD (4 chữ số BCD)

Nạp nội dung bộ đếm: thanh ghi đếm chỉ được nạp khi cả hai byte giá trị đếm được ghi.

2. Thủ tục xác lập chế độ làm việc cho bộ đếm.



3. Các chế độ làm việc

Chế độ làm việc xác định cách đáp ứng của đầu ra Output đối với đầu vào là dãy xung CLK và tín hiệu Gate.

Bộ đếm thực hiện đếm lượng chu kỳ xung tính từ nửa thấp của chu kỳ đầu tiên

Có 6 chế độ làm việc.

a. Chế độ 0:

Tạo khoảng thời gian trễ xác định và đặt đầu ra Output = "1" khi kết thúc đếm.

Đầu Output = "0" ngay sau khi chọn chế độ.

Ngay sau khi số đếm được nạp thì bắt đầu đếm. Điều kiện làm việc là Gate = "1"

Khi kết thúc đếm thì Output = "1" và giữ nguyên cho đến khi được nạp lại.

Việc nạp lại số đếm gây ra hai sự kiện:

+ Ghi byte đầu tiên làm dừng đếm.
+ Ghi byte thứ hai làm khởi đầu lần đếm mới.

b. Chế độ 1

Tạo xung đơn có độ rộng xác định.

Đầu ra Output = "0" khi GATE ="1" và bắt đầu đếm.

Output = "1" khi kết thúc đếm.

Việc nạp lại số đếm trong khi Output = "0" không làm ảnh hưởng tới độ rộng xung đầu ra.

Việc đếm được khởi đầu lại (xung Output bị kéo dài) nếu GATE ="0" và sau đó GATE = "1".

c. Chế độ 2

Bộ chia tần – phát xung.

Bộ đếm được dùng như một bộ chia tần. Nội bộ đếm được nạp xác định hệ số chia. Chu kỳ dãy xung đầu ra , tính từ một xung đầu ra Output = "0" đến một xung Output = "0" tiếp theo đúng bằng số lượng xung vào CLK.

Điều kiện làm việc là GATE ≡ "1".

Độ rộng mức "0" của xung ra đúng bằng chu kỳ T của xung CLK.

Có thể dùng tín hiệu GATE để đồng bộ quá trình đếm – phát xung.

d. Chế độ 3

Bộ chia tần – phát xung vuông.

Làm việc giống chế độ 2, chỉ khác ở chỗ là độ rộng mức "0' bằng độ rộng mức "1".

Điều kiện làm việc là GATE = "1".

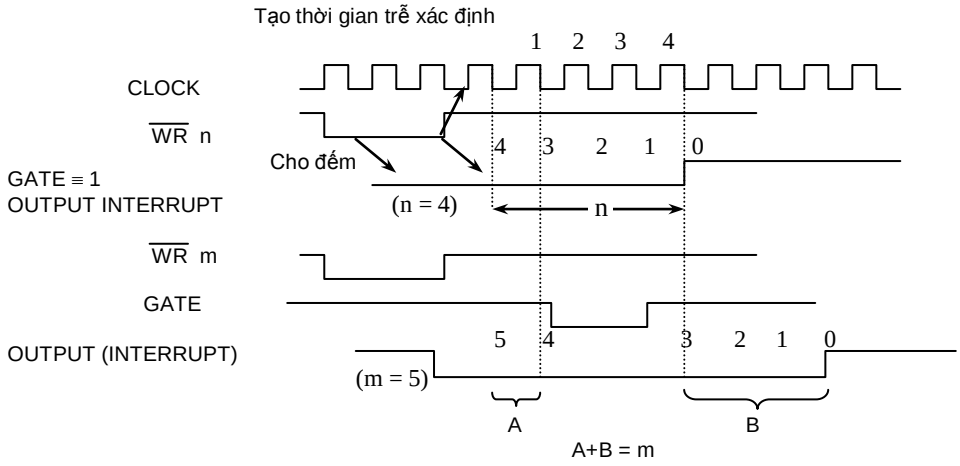
e. Chế độ 4

Tạo xung chốt bằng phần mềm.
Sau khi đặt chế độ làm việc thì Output = "1".
Sau khi số đếm được nạp thì bắt đầu đếm. Điều kiện làm việc là GATE = "1".
Đầu ra Output = "0" khi kết thúc đếm, độ rộng xung đầu ra bằng độ rộng chu kỳ xung CLK.
Việc nạp lại số đếm trong khi đếm làm khởi động lại việc đếm.

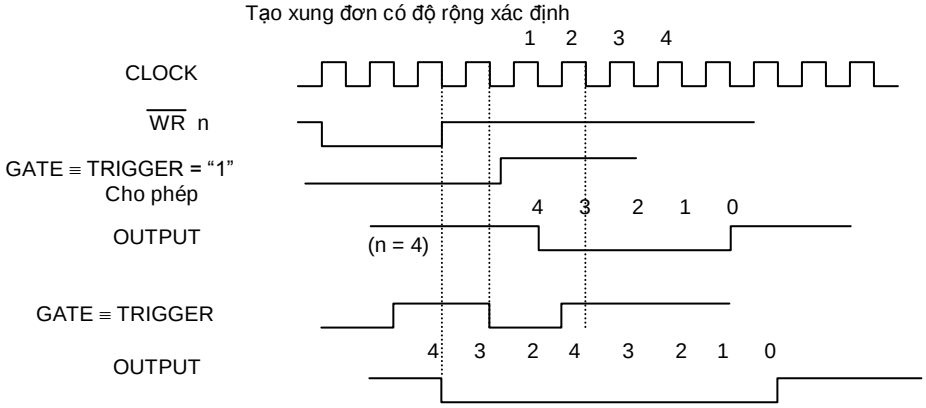
f. Chế độ 5

Tạo xung chốt bằng phần cứng.
Sau khi đặt chế độ thì Output ="1".
Bắt đầu đếm khi GATE = "1". Đầu ra Output = "0" khi kết thúc đếm, độ rộng xung đầu ra bằng độ rộng chu kỳ xung CLK.

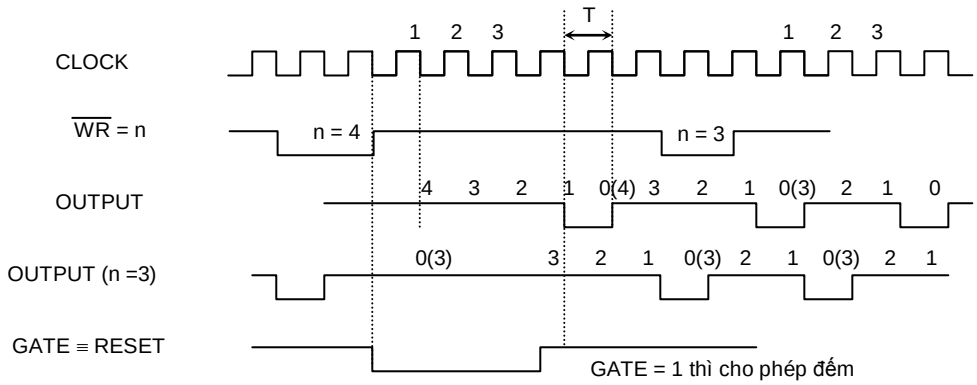
MODE 0: Interrupt on Terminal Count with GATE = 1



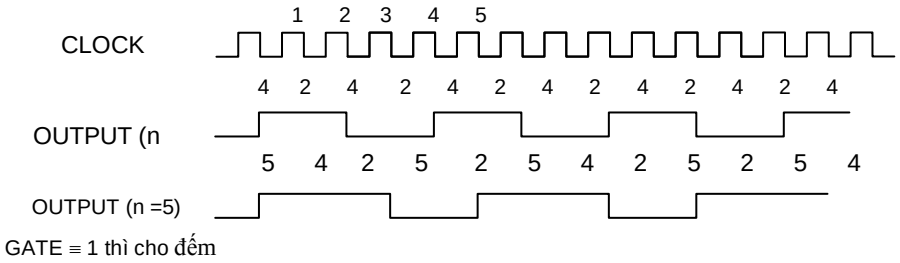
MODE 1: Programmable One-Shot.



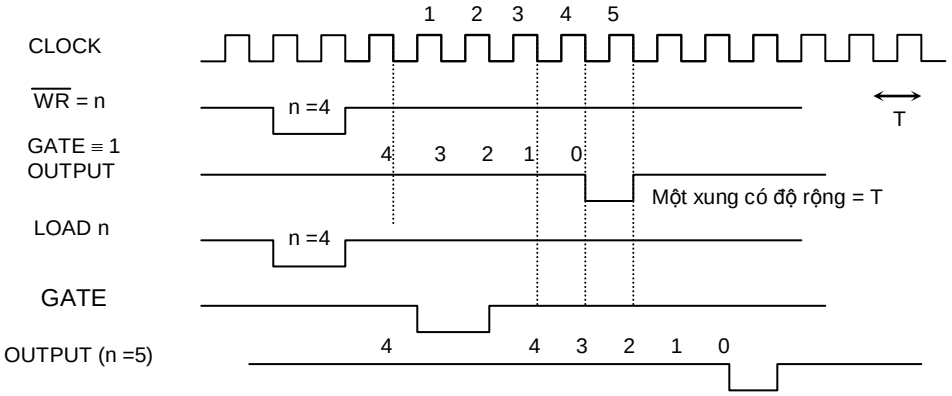
MODE 2: Rate Generator. (with GATE=1)



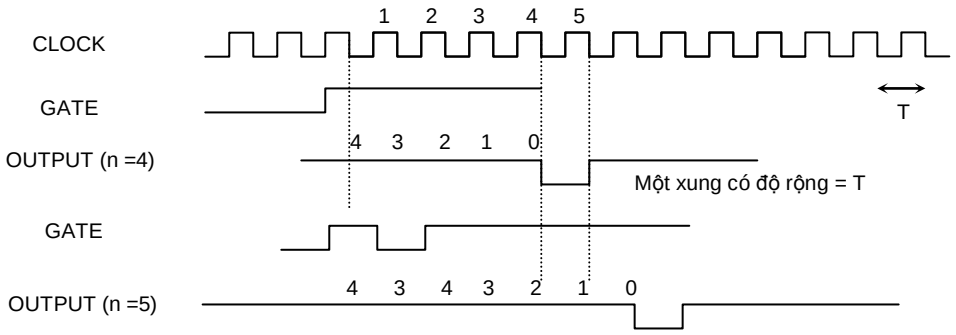
MODE 3: Square Wave Generator (with GATE=1)



MODE 4: Software Triggered Strobe (with GATE=1)



MODE 5: Hardware Triggered Strobe



4. Khả năng đọc nội dung bộ đếm trong khi đếm (Đọc trong khi đếm).

Để thực hiện được thao tác đọc trong khi đếm thì *cần nạp từ điều khiển đặc biệt vào thanh ghi có địa chỉ* A1, A0 = 11 .

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	0	0	X	X	X	X

SC1, SC0: Bộ đếm được chọn đặt chế độ đọc trong khi đếm.

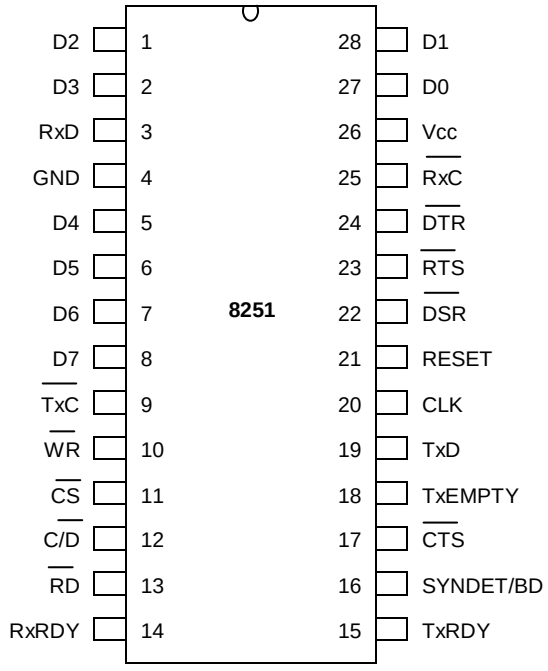
D5, D4: (00, Mã xác định chế độ đọc trong khi ghi)

X Không tác động

IV.4.4 Mạch điều khiển vào/ra nối tiếp đồng bộ/dị bộ USART-8251 (Universal Synchronous/Asynchronous Receiver Transmitter)

USART-8251 là một mạch giao diện vào/ra khả lập trình của hãng Intel. Các tính năng chủ yếu của mạch bao gồm:

- Hoạt động ở một trong hai chế độ đòng bộ hoặc không đồng bộ
- Hoạt động đồng bộ với mã 5 – 8 bits, ký tự đồng bộ nội bộ hoặc từ bên ngoài, có chế độ đồng bộ tự động

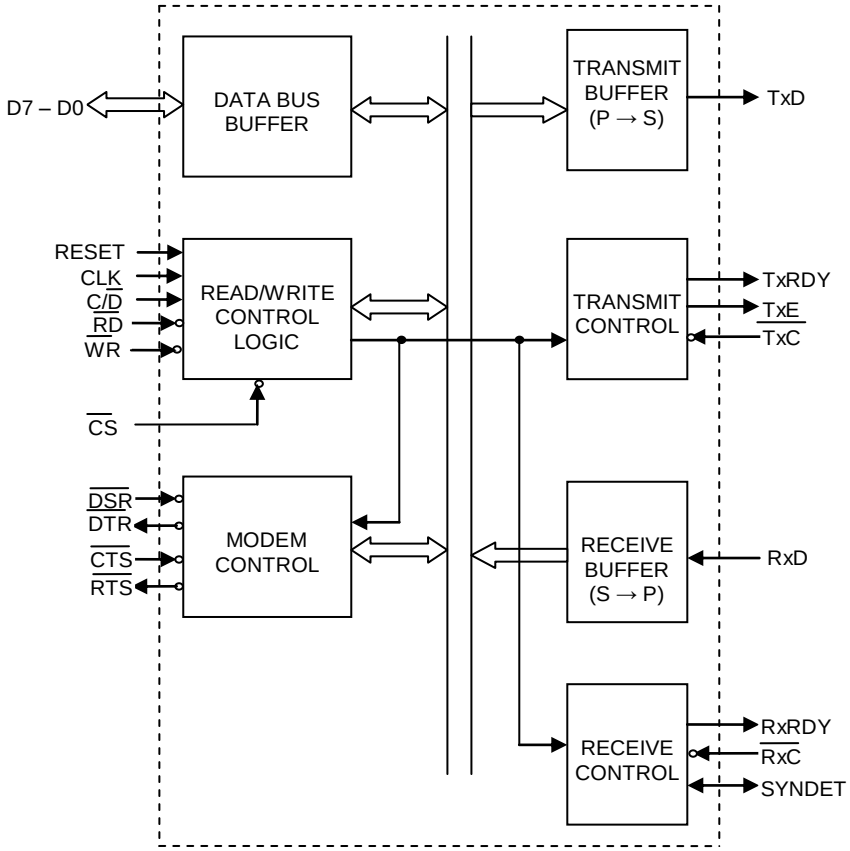


Hình IV.5a - Sơ đồ nối chân của USART 8251

- Hoạt động không đồng bộ với mã 5 – 8 bits, hệ số xung nhịp 1, 16 hoặc 64 lần tốc độ Baud, tạo ký tự tạm dừng; với 1, $1\frac{1}{2}$ và 2 bits Stop, phát hiện lỗi bit Start và ký tự Break (tạm dừng)
- Khả năng tự phát hiện lỗi thu phát

– Tương thích hoàn toàn với các chip họ 80x86

Mạch USART 8251 được thiết kế cho mục đích trao đổi dữ liệu nối tiếp giữa CPU và các thiết bị ngoại vi. Người lập trình có thể chọn các phương thức thu/phát dữ liệu đồng bộ hoặc không đồng bộ, chọn tốc độ thu/phát phù hợp thông qua các từ điều khiển (Control Word - CW). Bản thân USART thực hiện công việc chuyển đổi dữ liệu từ CPU thành dữ liệu nối tiếp để gửi ra thiết bị ngoại vi, đồng thời, mạch cũng tự chuyển dữ liệu nối tiếp thu nhận được thành dữ liệu song song để chuyển cho CPU. Chức năng của USART 8251 là làm trung gian cho việc giao tiếp với thiết bị ngoại vi (interfacing) bằng thu/phát dữ liệu nối tiếp, còn bản thân dữ liệu trao đổi giữa CPU và USART 8251 vẫn là giao diện song song.



Hình IV.5a - Sơ đồ cấu trúc bên trong của USART 8251

a) Mô tả chức năng:

Cũng như các mạch chức năng khác trong hệ thống 80x86, cấu hình chức năng của mạch USART-8251 rất uyển chuyển nhờ được thiết lập bằng phần mềm. Trong môi trường trao đổi dữ liệu, giao diện nối tiếp thực hiện việc biến đổi dữ liệu sóng song của hệ thống thành dạng dữ liệu nối tiếp để gửi đi và biến dạng dữ liệu nối tiếp thu nhận được thành dữ liệu sóng song để CPU đọc vào. Tất nhiên, khi thực hiện công việc biến đổi, USART-8251 sẽ tự động bỏ đi hoặc thêm vào các bit hoặc ký tự

đồng nhất chức năng trong kỹ thuật thu phát thông tin. Chính nhờ vậy, giao diện giữa CPU và USART-8251 là hoàn toàn minh bạch, chỉ đơn thuần là gửi đi hay nhận về một byte dữ liệu.

+ **Đệm BUS dữ liệu:**

Là bộ đệm 3 trạng thái hai chiều với độ rộng 8 bits dùng làm giao diện giữa CPU và mạch 8251. Dữ liệu được gửi đi hay nhận về qua bộ đệm khi thực hiện lệnh INPut hay lệnh OUTput trong CPU. Các từ lệnh (Command Word), từ điều khiển (Control Word) hay thông tin trạng thái cũng được chuyển qua thanh đệm dữ liệu. Thanh ghi trạng thái lệnh (Command Status Register), thanh ghi dữ liệu ra (Data Out Register) và thanh ghi dữ liệu vào (Data In Register) là những thanh ghi độc lập và cùng được kết nối BUS dữ liệu của hệ thống thông qua đệm dữ liệu.

+ **RESET:**

Mức "1" logic ở đầu vào này đưa 8251 về chế độ nghỉ. Chế độ này tồn tại cho đến khi một chuỗi từ điều khiển, từ lệnh mới được gửi tới 8251 để xác định chế độ làm việc. Mức "1" này phải tồn tại trong khoảng thời gian ngắn nhất của 6 chu kỳ xung nhịp hệ thống.

+ **CLK (Clock):**

Là đầu vào xung nhịp cho 8251 làm việc. Tần số xung nhịp này phải lớn hơn tối thiểu 30 lần so với tốc độ thu/phát của 8251.

+ **$\overline{WR}$ (ghi):**

Mức "0" logic xuất hiện ở đầu vào này là xung điều khiển từ CPU trong việc ghi từ điều khiển hoặc gửi dữ liệu cho 8251.

+ **$\overline{RD}$ (đọc):**

Mức "0" logic xuất hiện ở đầu vào này là xung điều khiển từ CPU trong việc đọc trạng thái của 8251 hoặc đọc dữ liệu từ 8251 vào CPU.

+ **C/$\overline{D}$:**

Đầu vào điều khiển, kết hợp với các tín hiệu vào gồm $\overline{CS}$, $\overline{WR}$ và $\overline{RD}$ xác định cho 8251 dữ liệu tồn tại trên BUS là ký tự dữ liệu, từ điều khiển hay thông tin trạng thái. "1" ứng với CONTROL/STATUS, "0" ứng với DATA.

C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	0	1	0	8251 DATA → DATA BUS
0	1	0	0	DATA BUS → 8251 DATA
1	0	1	0	STATUS → DATA BUS
1	1	0	0	DATA BUS → CONTROL
X	1	1	0	DATA BUS → TRI-STATE
X	X	X	1	DATA BUS → TRI-STATE

Lưu ý: Chân C/$\overline{D}$ thường được nối với dây địa chỉ A0 của BUS địa chỉ, do vậy có thể dễ dàng phân biệt hai địa chỉ duy nhất của 8251 là: Địa chỉ nền là địa chỉ đọc hoặc ghi dữ liệu, địa chỉ nền + 1 là địa chỉ cho ghi từ điều khiển và đọc trạng thái.

+ **$\overline{CS}$ (Chip Select):**

Tín hiệu chọn vỏ đối với 8251. Mức "0" là tích cực, chip 8251 được chọn. Khi $\overline{CS}$ = "1", các tín hiệu đọc ($\overline{RD}$) và ghi ($\overline{WR}$) không có tác động đối với 8251.

+ **Modem Control:**

Vi mạch 8251 có một tập tín hiệu vào/ra có thể sử dụng để đơn giản hoá việc phối ghép với các MODEM. Các tín hiệu do khối chức năng điều khiển Modem tạo ra nhằm mục đích hoàn toàn tương thích với các tín hiệu điều khiển trao đổi thông tin thông qua thiết bị Modem khi cần thiết. Đó là các tín hiệu $\overline{DSR}$ (Data Set Ready), DTR (Data Terminal Ready), $\overline{RTS}$ (Request To Send), và $\overline{CTS}$ (Clear To Send).

+ **Đệm phát (Transmitter Buffer):**

Đệm phát tiếp nhận dữ liệu song song từ đệm dữ liệu, chuyển đổi thành chuỗi bits nối tiếp, chèn thêm các ký tự hoặc các bit thích hợp cần thiết trong kỹ thuật truyền tin và gửi chuỗi bits này ra đầu phát TxD theo sườn xuống của xung nhịp phát TxC. Khối phát bắt đầu công việc ngay khi tín hiệu $\overline{CTS}$ = "0" và dừng lập tức với trạng thái được giữ nguyên khi TxE là "0" hay $\overline{CTS}$ = "1".

+ **Điều khiển phát (Transmitter Control):**

Khối điều khiển phát giám sát toàn bộ mọi hoạt động liên quan đến truyền dữ liệu nối tiếp. Khối có nhiệm vụ chấp nhận và tạo ra tất cả các các tín hiệu tương ứng để thực hiện việc tuyền dữ liệu.

+ **TxRDY (Transmitter Ready):**

Tín hiệu ra của 8251 thông báo cho CPU biết nó sẵn sàng nhận dữ liệu để tuyền đi. Tín hiệu này có thể sử dụng làm tín hiệu yêu cầu ngắt đối với hệ thống, và khác với tín hiệu TxE (Transmitter Empty). Trong chế độ phát có thăm dò, CPU có thể thông qua tín hiệu TxRDY để quyết định chuyển dữ liệu cho 8251. Tín hiệu này bị Reset bởi tín hiệu $\overline{WR}$ khi dữ liệu được gửi tới 8251 từ CPU.

Lưu ý rằng, trong chế độ phát theo thăm dò, tín hiệu TxRDY không bị ràng buộc bởi tín hiệu TxE, nó chỉ có tác dụng thông báo trạng thái đầy hay rỗng của thanh ghi đệm phát.

+ **TxE (Transmitter Empty):**

Khi 8251 chuyển xong một ký tự. "không còn gì để phát đi", đầu ra TxE sẽ chuyển đổi lên mức "1" logic. Có thể thông qua tín hiệu này để biết được trạng thái kết thúc truyền của 8251, đặc biệt trong chế độ half-duplex.

Trong chế độ thu phát đồng bộ giá trị "1" ở đầu ra này chỉ ra rằng chưa có dữ liệu được truyền đi, ký tự SYNC hoặc là ký tự dữ liệu sắp sửa

được truyền. TxE không thay đổi mức khi ký tự SYNC bắt đầu được gửi đi.

+ **$\overline{\text{TxC}}$ (Transmitter Clock):**

Xung nhịp phát điều khiển tốc độ truyền các ký tự. Ở chế độ thu phát đồng bộ, tốc độ Baud Rate (1x) bằng chính tần số $\overline{\text{TxC}}$. Trong chế độ thu phát không đồng bộ tốc độ này luôn theo một tỷ lệ tương ứng của Baud Rate. Các tỷ lệ thường được sử dụng là 1x, 16x hoặc 64x tốc độ Baud Rate. Sườn xuống của $\overline{\text{TxC}}$ dịch chuyển dữ liệu nối tiếp ra chân TxD của 8251.

+ **Đệm thu (Receiver Buffer):**

Đệm thu thu nhận dữ liệu nối tiếp và chuyển đổi thành dữ liệu song song sau khi đã loại bỏ những ký tự hoặc bit tương ứng sử dụng trong kỹ thuật thu phát thông tin. Tín hiệu thu được đưa qua chân RxD và được dịch chuyển vào thanh ghi đệm thu theo sườn lên của xung $\overline{\text{RxC}}$.

+ **Khối điều khiển thu (Receiver Control):**

Khối này giám sát và điều khiển mọi hoạt động liên quan đến việc nhận dữ liệu nối tiếp. Các tính năng chủ yếu của khối này như sau:

- Trong điều kiện nghỉ, RxD ngăn mọi tín hiệu "low". Trước khi bắt đầu nhận dữ liệu nối tiếp, trên chân này phải được khẳng định giá trị "1" logic, từ đó, khối bắt đầu dò tìm giá trị "0" có nghĩa, tương ứng với Start Bit
- Mạch nhận biết bit Start bằng cách loại trừ mọi tín hiệu nhiễu thông qua dò tìm sườn xuống trên RxD và xác định bit Start cho việc thu nhận dữ liệu.
- Phát hiện lỗi chẵn lẻ thông qua bit trạng thái chắn lẻ
- Phát hiện lỗi khung dữ liệu thông qua bit Stop ở cuối byte dữ liệu cuối cùng trong chế độ thu phát không đồng bộ.

+ **RxRDY (Receiver Ready):**

Tín hiệu ra RxRDY báo rằng 8251 đã nhận xong một ký tự và đang sẵn sàng chuyển cho CPU. RxRDY cũng có thể nối vào chân yêu cầu ngắt đối với CPU trong phương pháp vào/ra theo ngắt, hoặc làm tín hiệu báo trạng thái trong phương pháp vào/ra thăm dò (polled operation). Tín hiệu RxEnable khi là "off", sẽ giữ cho RxRDY ở điều kiện tái khởi động. Thanh ghi đệm vào phải được phép dò tím bit Start của dữ liệu mới và ký tự hoàn chỉnh đã được nhận phải được gửi vào thanh ghi dữ liệu ra. Khi xảy ra sự cố đọc ký tự đã nhận được từ thanh ghi dữ liệu ra, chip sẽ tạo lỗi Overun, ký tự vừa nhận sẽ bị bỏ qua.

+ **$\overline{\text{RxC}}$ (Receiver Clock):**

Xung nhịp nhận tạo lập tốc độ thu dữ liệu. Dữ liệu được ghi nhận từng bit theo sườn lên của xung nhịp $\overline{\text{RxC}}$.Trong chế độ thu phát đồng bộ,

tần số $\overline{RxC}$ bằng đúng tần số của xung Baud Rate. Còn trong chế độ thu phát không đồng bộ , tần số xung này được lấy theo tỷ lệ 1x, 16x hoặc 64x tần số tốc độ Baud Rate. Có thể lấy ví dụ:

Baud Rate là 2400 Baud, yêu cầu đối với xung nhịp $\overline{RxC}$ là

$\overline{RxC}$ = 2400Hz ở chế độ 1x

$\overline{RxC}$ = 38,4KHz ở chế độ 16x và

$\overline{RxC}$ = 153,6KHx ở chế độ 64x.

Lưu ý rằng, tốc độ Baud Rate là một tốc độ phải chọn theo quy chuẩn quốc tế, thông thường là 300, 600, 1200, 2400, 4800, 9600, 19200 Baud, v.v…, chứ không phải là một số bất kỳ, nên việc tạo xung tần số cho $\overline{RxC}$ và $\overline{TxC}$ thường được sủ dụng những thạch anh có tần số là bội 16, bội 64 của chuỗi số trên với độ chính xác rất cao, chứ không sử dụng tuỳ tiện. Hơn nữa, trong phần lớn các hệ thống thu phát thông tin, tốc độ thu và tốc độ phát là như nhau, dẫn đến tần số $\overline{RxC}$ và $\overline{TxC}$ cũng là một và được lấy chung từ bộ tạo tốc độ Baud Rate Generator để đơn giản hoá phần giao diện.

+ SYNDET (SYNC Detect/BRKDET Break Detect):

Chân này được sử dụng trong chế độ thu phát đồng bộ để nhận biết ký tự đồng bộ, có thể sử dụng như đầu vào hoặc đầu ra, được định nghĩa qua từ điều khiển. Chân được chuyển đầu ra sau khi hệ thống có Reset. Trong chế độ đồng bộ nội (Internal Sync Mode), chân này lên mức "1" được sử dụng như đầu ra trạng thái báo đã định vị được ký tự đồng bộ trong chế độ thu. Khi được lập trình ở chế độ xung đồng bộ kép (Double Sync Character), hay còn gọi là bi-sync, SYNDET sẽ lên mức "1" ở giữa bit cuối của ký tự đồng bộ thứ hai. SYNDET được Reset trong khi thực hiện đọc trạng thái. Ở chế độ đồng bộ ngoại (External Sync Detect Mode), sườn xung lên tại chân SYNDET khởi động 8251 bắt đầu ghép dữ liệu ký tự từ sườn lên của xung nhịp RxC. Chế độ này bị cấm khi lập trình cho 8251 hoạt động ở chế độ Internal Sync Mode.

+ BREAK (chỉ có trong chế độ không đồng bộ (Asynchronnous Mode):

Đầu ra này sẽ lên "1" khi trên lối vào là LOW (="0") xuyên suốt hai lần gặp bit Stop trong chuỗi (tất nhiên kể cả bit Start, các bits dữ liệu và bit chẵn lẻ). Bit BREAK cũng có thể đọc được như một bit trạng thái.

b) Mô tả hoạt động

Việc xác định chế độ làm việc cho USART-8251 được thực hiện thông qua chương trình mềm. Một chuỗi các từ điều khiển cần được CPU gửi tới 8251 để xác định các định dạng truyền tin. Các từ điều khiển sẽ xác lập: Baud Rate, độ dài mã ký tự, số bit Stop, đồng bộ hay dị bộ, kiểm tra chẵn lẻ v.v… Trong chế độ đồng bộ, còn cần xác điịnh Internal Sync

hay External Sync Mode. Sau khi đã nhận được các từ điều khiển cần thiết, 8251 sẵn sàng làm việc. Tất nhiên, sau khi nhận các từ điều khiển, 8251 còn phải chờ cho đến khi bit TxEnable được thiết lập nhờ từ lệnh làm việc (Command Instruction Word) và tín hiệu $\overline{CTS}$ (Clear To Send).

b) Lập trình cho 8251:

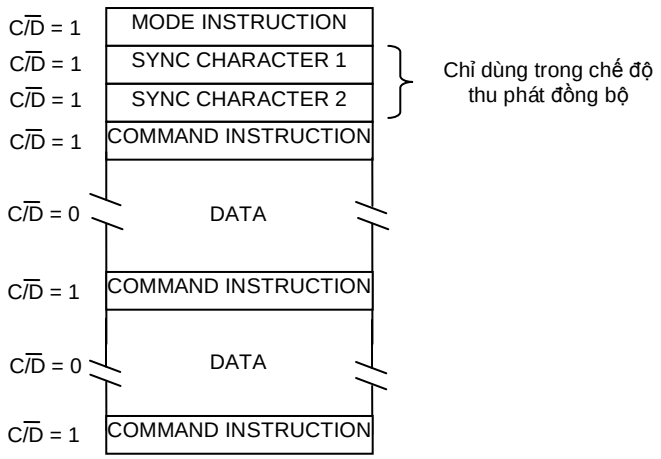
Trước khi phát hay nhận dữ liệu, 8251 phải được nhận một chuỗi từ điều khiển. Từ điều khiển 8251 có hai loại: Lệnh chế độ (Mode Instruction) và Lệnh làm làm việc (Command Instruction).

– Mode Instruction (MI):

Từ điều khiển chế độ làm việc cho 8251, được nạp vào sau khi mạch được khởi động hay tái khởi động cứng hoặc mềm (Reset). Khi đã được CPU ghi vào 8251, các ký tự SYNC hoặc từ lệnh làm việc (Command Instruction) có thể được chuyển tiếp cho 8251 để kích hoạt 8251.

– Mode Instruction (CI):

Từ lệnh làm việc cho 8251, được dùng để điều khiển công việc thực thụ của mạch. Từ điều khiển (MI) và từ lệnh (CI) phải được gửi cho 8251 theo một tuần tự khe khắt (Xem Hình IV.6). MI phải được ghi vào 8251 ngay sau khi có tín hiệu Reset trước khi sử dụng cho việc truyền dữ liệu.

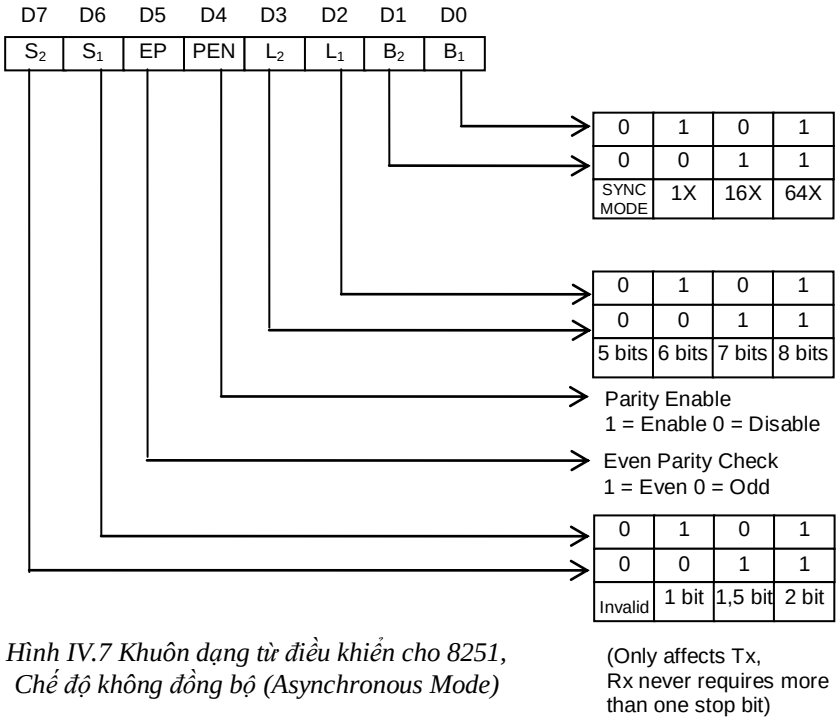


Hình IV.6 Tuần tự khởi động và làm việc với 8251.
Ký tự SYNC thứ hai có thể bỏ qua khi MI xác định cho 8251 làm việc ở chế độ tự đồng bộ (Internal SYNC Mode)

8251 có thể làm việc trong hai chế độ: Dị bộ (Asynchronous) và đồng bộ (Synchronous). Để hiểu được các chế độ làm việc này, có thể giả

thiết rằng 8251 là hai vi mạch độc lập: một mạch thu phát đồng bộ và một mạch thu phát dị bộ cùng được ghép vào trong một vỏ. Các chế độ chỉ có thể đặt cho 8251 sau tín hiệu Reset hệ thống. Lưu ý rằng khi cho phép sử dụng bit kiểm tra chẵn lẻ (parity bit), nó sẽ không được tính vào trong độ dài của ký tự thu phát. Giá trị thực của bit chẵn lẻ trên lối vào RxD không được đọc vào 8251. Nếu độ dài ký tự nhỏ hơn 8 bits, nó sẽ bị bỏ đi khi đọc vào 8251, còn nếu độ dài ký tự bé hơn, sẽ được coi là mức "0".

Cấu trúc từ điều khiển chế độ không đồng bộ (Asynchronous Mode) được thể hiện trên Hình IV.7. Từ điều khiển được gửi cho 8251 ngay sau khi có tín hiệu Reset. Dạng từ điều khiển của USART-8251 được biểu diễn trên Hình IV. 7

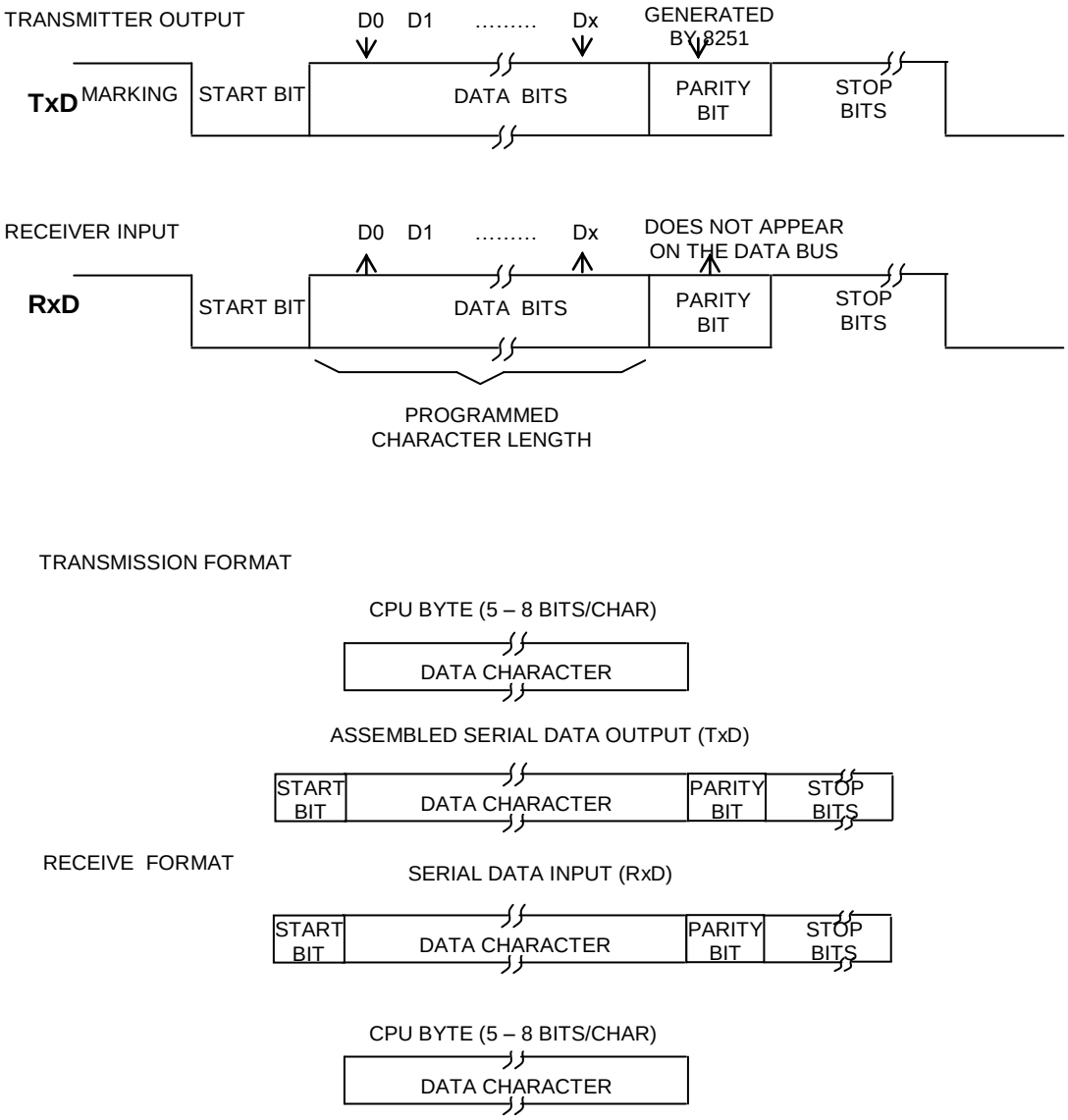


Hình IV.7 Khuôn dạng từ điều khiển cho 8251, Chế độ không đồng bộ (Asynchronous Mode)

+ *Phát không đồng bộ (Asynchronous Transmission):*

Khi dữ liệu được chuyển cho 8251 phát đi, 8251 tự động ghép thêm bit Start (mức "0"), sau đó là các bit dữ liệu, bắt đầu bằng LSBit và gán thêm bit kiểm tra chẵn lẻ, rồi đến bit Stop theo đúng *khung dữ liệu* được định nghĩa trong từ điều khiển. khung dữ liệu này được truyền đi ~~như~~ một chuỗi xung trên lối ra TxD. Các bit được dịch chuyển ra TxD bằng sườn xuống của xung nhịp TxC theo tốc độ 1, 16 hay 64 lần xung nhịp TxC theo từ điều khiển đã xác định sẵn. Khi không còn dữ liệu để truyền đi, TxD chuyển lên mức cao "1" (marking).

Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866



Lưu ý: Nếu độ dài ký tự được gán theo lệnh là 5, 6 hoặc 7 bits
Các bits không dùng tới sẽ được gán bằng "0"

+ *Thu không đồng bộ (Asynchronous Receive):*

Bình thường, RxD ở mức cao ("1"), sườn xuống xuất hiện được coi là sự bắt đầu của bit Start. Tính hợp lệ của bit Start được xác nhận bằng xung mẫu tại điểm giữa của xung này (đối với trường hợp tốc độ 16x hay 64x). Nếu vẫn có giá trị là "0", đó là bit Start hợp lệ, và bộ đếm bit bắt đầu hoạt động. Các bit của dữ liệu và bit chẵn lẻ được lấy mẫu tại điểm giữa trên lối vào RxD bằng sườn lên của xung nhịp $\overline{RxC}$. Nếu mức thấp được nhận biết ở đoạn tồn tại của bit Stop, cờ lỗi sẽ được thiết lập. Bit Stop báo hiệu kết thúc của một ký tự. Lưu ý rằng phần thu chỉ cần nhận được một bit Stop, bất kể số lượng bit Stop được gán là bao nhiêu. Ký tự nhận được sẽ được chuyển vào bộ đệm song song của 8251. Tín hiệu RxRDY sẽ chuyển lên mức cao để báo cho CPU biết có thể nhận ký tự.

Nếu ký tự trước chưa được CPU đọc về, ký tự mới vẫn sẽ được chuyển vào thanh ghi đệm này, và cờ báo lỗi tràn sẽ được thiết lập (tức là ký tự trước bị bỏ qua). Tất cả các cờ báo lỗi có thể Reset nhờ lệnh Error Reset. Hình IV.4 cho thấy các dạng thức *khung dữ liệu* (Data Fram) được phát đi và thu về trong chế độ làm việc không đồng bộ.

+ *Phát đồng bộ (Synchronous Transmission):*

Đầu ra TxD ở mức cao cho đến khi CPU chuyển từ đầu tiên đến 8251, thông thường đó là ký tự đồng bộ SYNC. Khi đầu CTS chuyển sang mức thấp, ký tự đầu tiên được được phát nối tiếp ra TxD. Tất cả các ký tự được dịch chuyển nối tiếp ra theo sườn xuống của TxC. Tốc độ phát dữ liệu bằng đúng tốc độ TxC. Khi hoạt động phát đã được khởi động, chuỗi dữ liệu trên TxD được đồng bộ theo TxC. Nếu CPU không cung cấp dữ liệu cho 8251 trước khi bộ đệm phát bị rỗng, thì (các) ký tự SYNC sẽ được chèn vào chuỗi ký tự phát đi đồng thời tín hiệu điện áp chân TxEMPTY sẽ chuyển đổi lên mức "1" để thông báo rằng bộ đệm phát rỗng và ký tự SYNC được phát. Điện áp trên chân TxEMPTY được Reset khi ký tự mới được CPU chuyển tới 8251.

+ *Thu đồng bộ (Synchronous Receive):*

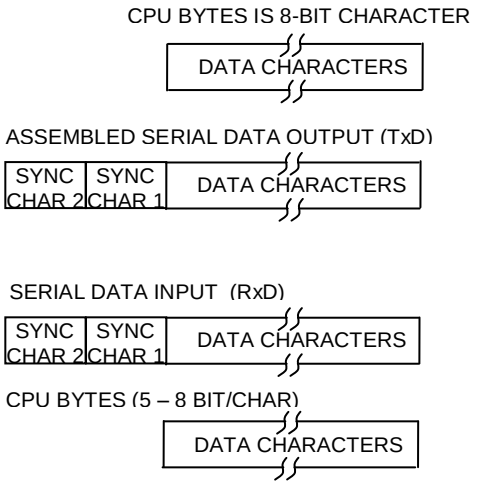
Tín hiệu đồng bộ có thể là tự động do bản thân 8251 tạo ra hoặc thu từ bên ngoài. Khi hoạt động ở chế độ đồng bộ, lệnh "săn tìm" (bit EH - ENTER HUNT) được gộp trong từ lệnh cho 8251. Dữ liệu trên lối vào RxD được "lấy mẫu" qua sườn lên của xung nhịp RxC. Nội dung của thanh ghi đệm nhận Rx buffer được so sánh với ký tự đồng bộ SYNC cho đến khi hoàn toàn phù hợp. Nếu được chọn là chế độ đồng bộ kép, tập các ký tự cũng được so sánh tương tự. Khi cả hai ký tự đồng bộ đã được nhận biết, USART 8251 kết thúc chế độ săn tìm và chuyển sang đồng bộ hoá ký tự. Chân SYNDET chuyển sang trạng thái logic "1", và sẽ tự động Reset nhờ lệnh đọc trạng thái.

Trong chế độ External Sync, việc đồng bộ đạt được nhờ áp mức cao lên chân SYNDET để loại trừ chế độ "săn tìm" của 8251. Mức cao này sẽ được Reset sau một nhịp RxC. Lệnh EH không có tác động gì trong chế độ này. Việc phát hiện lỗi chẵn lẻ và lỗi tràn hoàn toàn tương tự như ở thu dị bộ.

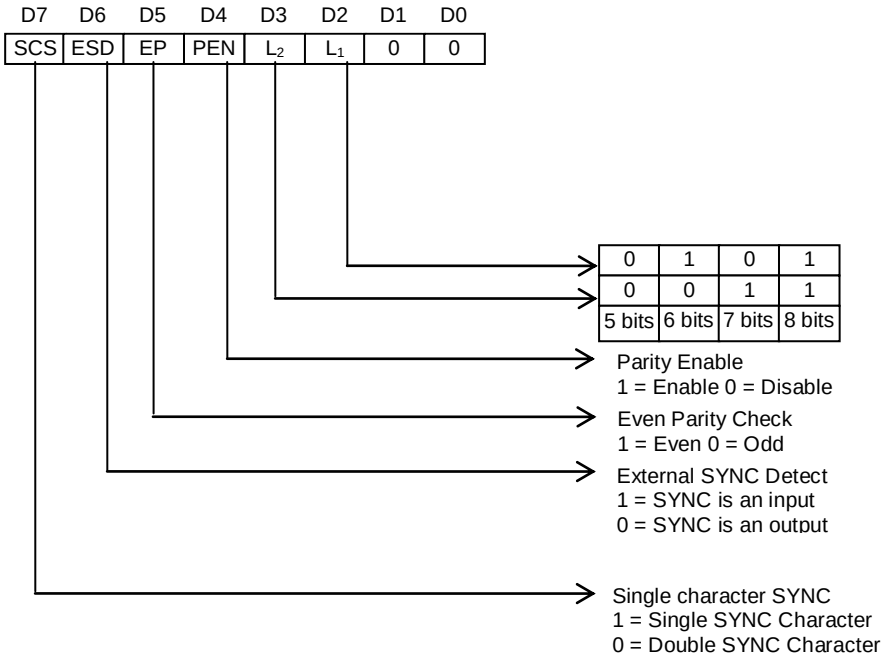
Hình IV.8 là gói dữ liệu trong thu phát đồng bộ.

– Command Instruction (CI)

Dạng của từ lệnh được thể hiện trên Hình IV.11. Sau khi 8251 đã được thiết lập chế độ làm việc, và ký tự đồng bộ đã được tách (nếu là chế độ động bộ) nó đã sẵn sàng cho một hoạt động thu phát dữ liệu.Từ lệnh được sử dụng để điều khiển các hoạt động thực tế của 8251 trong chế độ đã đặt. Các hoạt động đó là: Cho phép Thu / cho phép Phát, Reset các lỗi hay điều khiển Modem.

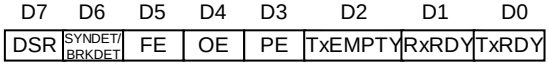


Hình IV. 8 Gói dữ liệu trong thu phát đồng bộ



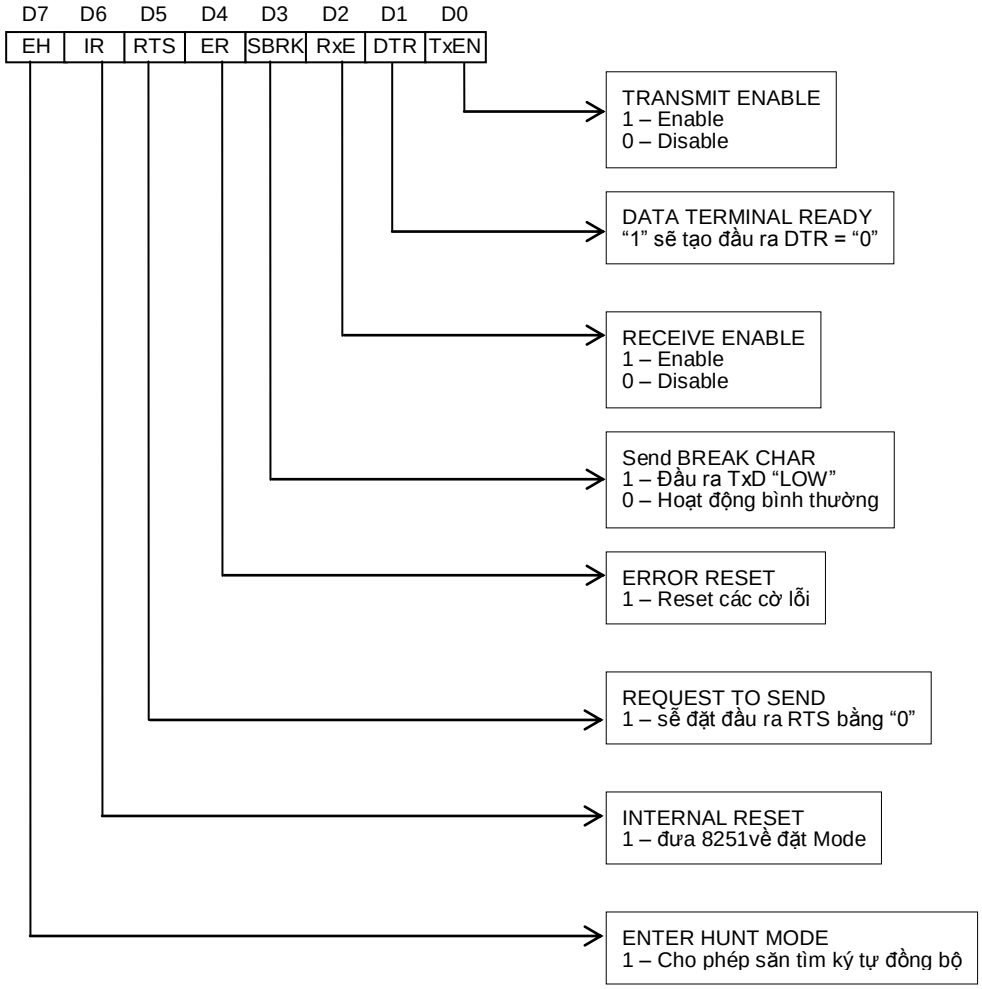
Hình IV. 9 Khuôn dạng từ điều khiển cho 8251,
Chế độ đồng bộ (Synchronous Mode)

Cũng cần hiểu thêm về từ trạng thái của 8251. Với lệnh đọc INput vào CPU trong trường hợp C/D = "1", từ trạng thái được đọc vào. Nội dung từ trạng thái được thể hiện trên Byte đọc vào, các giá trị logic của các bit trạng thái RxRDY, TxEMPTY, SYNDET/BRKDET tại các bit tương ứng D1, D2 và D6 là giá trị trên chính các chân ra tương ứng của 8251. Riêng bit TxRDY (bit D0) thể hiện trạng thái rỗng của thanh ghi đệm dữ liệu vào, còn giá trị trên chân ra TxRDY của 8251 là kết quả của sự phối hợp cùng các giá trị của CTS và TxEN trước đó.



Hình IV.10 Byte trạng thái của 8251

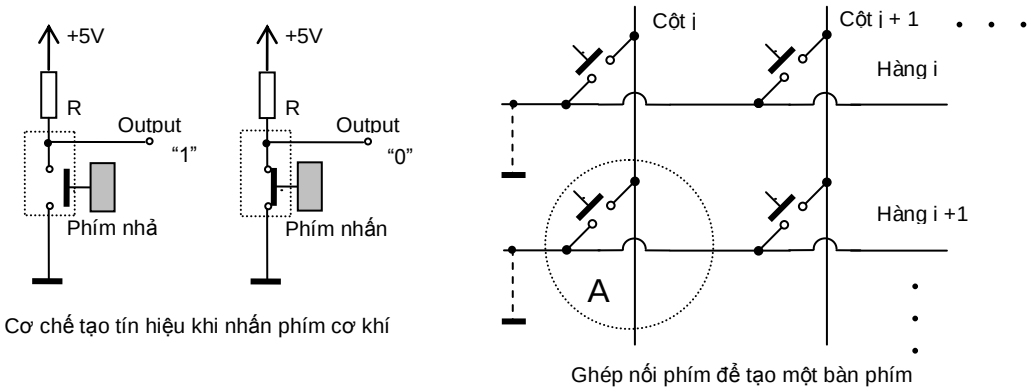
Bit D3 (Parity Error) bằng "1" nghĩa là phát hiện có lỗi trong khi kiểm tra tính chẵn lẻ của byte dữ liệu, bit náy được xoá bằng từ lệnh (bit ER).
Bit D4 (Overrun Error) được Set nếu CPU không kịp đọc dữ liệu trước khi có một byte mới đang được thu về. còn bit D7 (Data Set Ready) thông báo DSR đang ở mức "0".



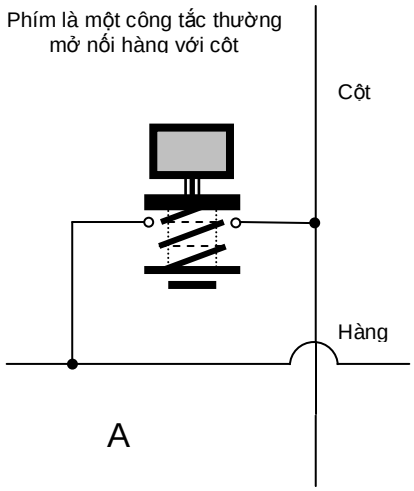
Hình IV.11- Khuôn dạng của từ lệnh 8251

CHƯƠNG V. THIẾT BỊ VÀO RA CỦA HỆ VI XỬ LÝ

V.1 Bàn phím Hex Keyboard



Hình V.1. Phím tiếp xúc và cách tạo bàn phím

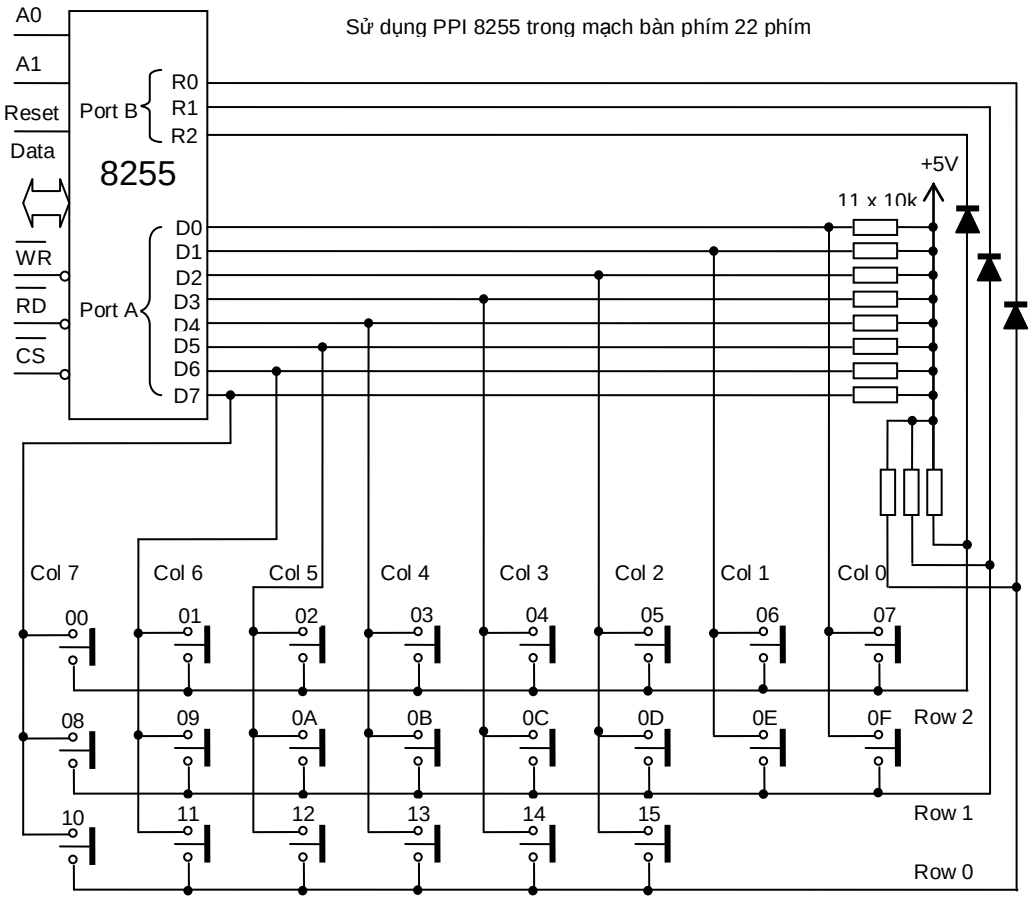


Bàn phím được tổ chức theo kiểu ma trận các hàng và các cột, tại vị trí giao nhau không tiếp xúc được ghép một công tắc thường mở nối hàng với cột, chỉ tiếp xúc khi được nhấn. Để xác định có một phím bị nhấn, ta nối đất tất cả các hàng và đọc nội dung các cột. Nếu trên cột nào đó ta đọc được giá trị là "0", tương ứng với trường hợp có một phím trên cột đó bị nhấn. Dễ dàng thấy rằng, nếu các hàng i và i + 1 nối đất, bất cứ phím nào trên cột j (hay j + 1) bị nhấn, ta đều đọc được giá trị "0" trên cột j (hay j + 1).

Hình V.2 là một bàn phím Hexa gồm 22 phím được tạo từ một ma trận 3 hàng và 8 cột. Giả sử rằng ta dùng vi mạch vào ra song song PPI-8255 để xây dựng nên bàn phím như trên Hình V.2. Ba lối ra của port B gồm R0, R1, R2 (tương ứng với các dây PB0, PB1 và PB2) được dùng ở chế độ Output, 8 lối vào của port A dùng D0 ÷ D7 (tương ứng với các dây PA0 ÷ PA7) ở chế độ Input. Như vậy chu trình đọc phím theo chế độ dò tìm (polling) được thực hiện như sau:

Nguyễn Trung Đồng - Viện Công nghệ Thông tin – Tel 098 341 0866

1. Để đảm bảo phím nhấn trước đó đã được nhả ra, các giá trị "0" cùng lúc được áp lên tất cả các hàng và đọc các giá trị trên các cột. Nếu các cột đều ở mức "1", chương trình tiếp tục đọc giá trị các cột
2. Quét các cột, tức là đọc giá trị tại các cột để phát hiện có phím bị nhấn. Để tăng độ tin cậy khi đọc phím, tránh tác động của nhiễu cơ học khi phím bị nhấn và các loại nhiễu khác, sau khi phát hiện có phím bị nhấn, chương trình chờ khoảng 20msec rồi đọc tiếp giá trị tại các cột. Giá trị "0" đọc dược ở cột nào sẽ được ghi nhớ để sử dụng cho việc xác định phím ở vị trí nào bị nhấn
3. Quét hàng để xác định vị trí của phím bị nhấn. Số vòng lặp này là không cố định, nhưng nhiều nhất là bằng số hàng có trong cấu trúc của bàn phím
4. Gán mã cho phím. Mã cho phím là do thiết kế phần cứng quy định, tuỳ theo chức năng và yêu cầu của người dùng.



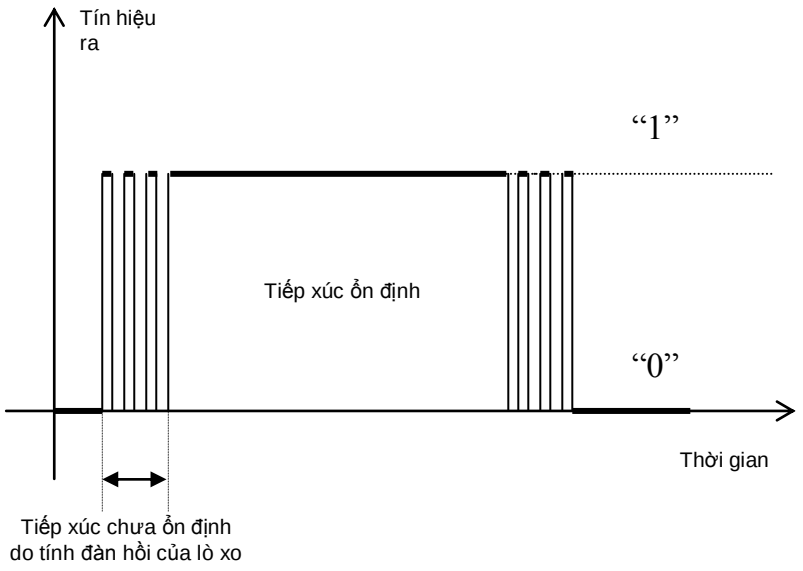
Hình V.2 – Bàn phím 22 phím sử dụng giao tiếp qua PPI8255

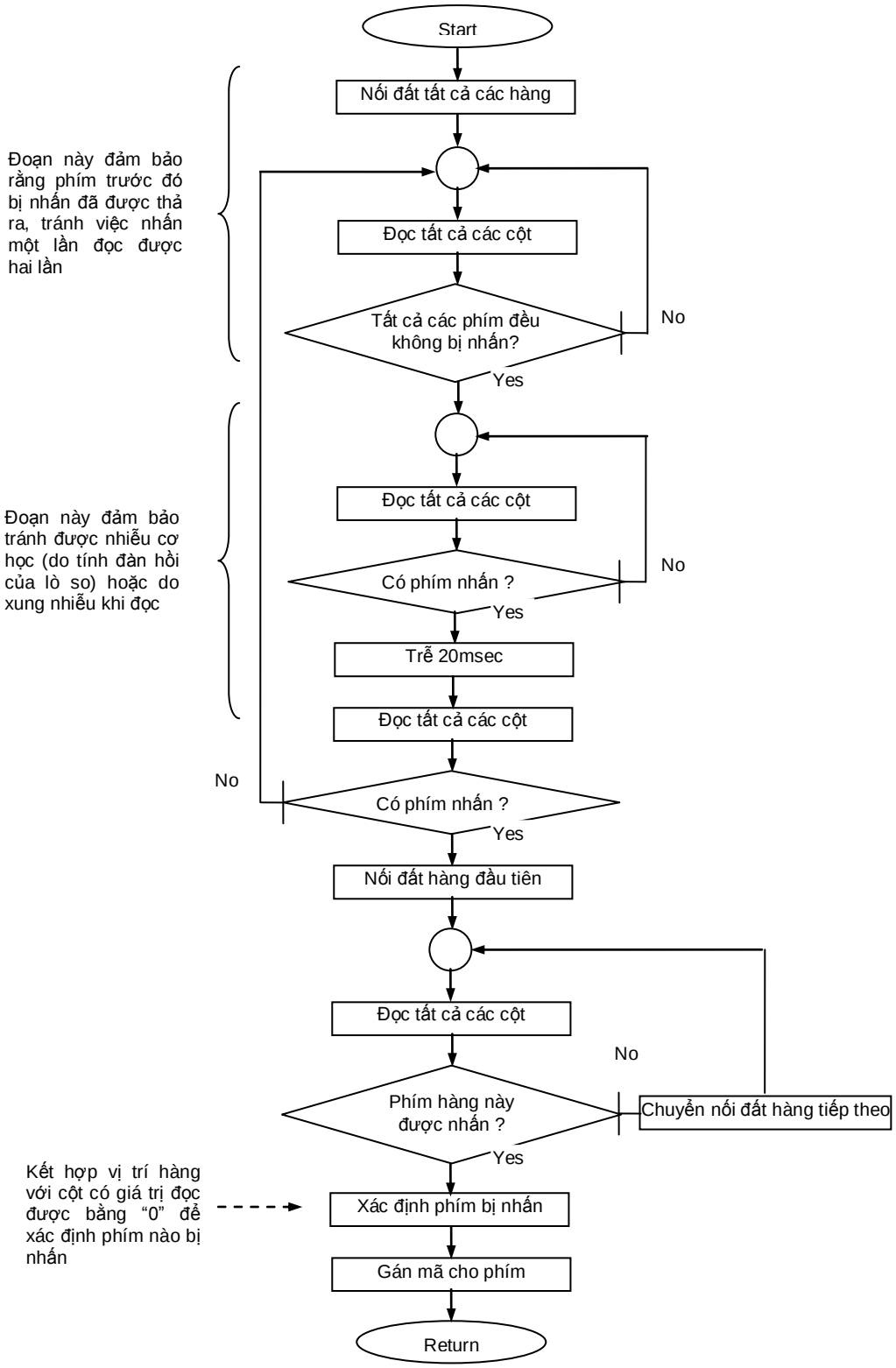
Trong ví dụ này giả sử rằng các phím được gán mã như sau:

- Từ phím 00 đến phím 0F (toàn bộ các phím trong Row 1 và Row 2) được gán mã Hexa từ "0_H" đến "F_H"
- Các phím ở Row 0 có thể gán các chức năng sau"
    - Phím 10 là phím chức năng "GO" - thực hiện chương trình
    - Phím 11 là phím chức năng "INS" - thực hiện chức năng thay đổi nội dung các thanh ghi của CPU
    - Phím 12 là phím "REP" - thực hiện chức năng sửa nội dung thanh ghi của CPU
    - Phím 13 là phím "DISP" - thực hiện chức năng hiển thị nội dung các thanh ghi của CPU
    - Phím 14 là phím "STEP" - thực hiện chức năng chạy chương trình theo từng lệnh
    - Phím 14 là phím "ENTER" - thực hiện chức năng kết thúc nhập dữ liệu hoặc lệnh từ bàn phím

Lưu đồ chương trình đọc và xác định phím bị nhấn được thể hiện trên Hình V.3 Chương trình có thể được viết dưới dạng một chương trình con.

Do tính đàn hồi của lò xo trong phím nên sự tiếp xúc của phím sau khi bị nhấn có thể mô tả như hình sau:





Hình V.3 Lưu đồ chương trình đọc bàn phìm

V.2 Ghép nối bàn phím với hệ Vi xử lý

Bàn phím là thiết bị ngoại vi cho phép đưa thông tin vào máy tính dưới dạng mã ký tự. Bàn phím thực hiện chức năng chuyển thông tin dạng lực nhấn phím và vị trí của phím được nhấn thành mã phím và chuyển cho máy tính. Bàn phím gồm hai bộ phận chính là ma trận phím và mạch điện tử quét phím. Ma trận phím là tổ hợp các phím nhấn được sắp xếp theo các hàng và cột.

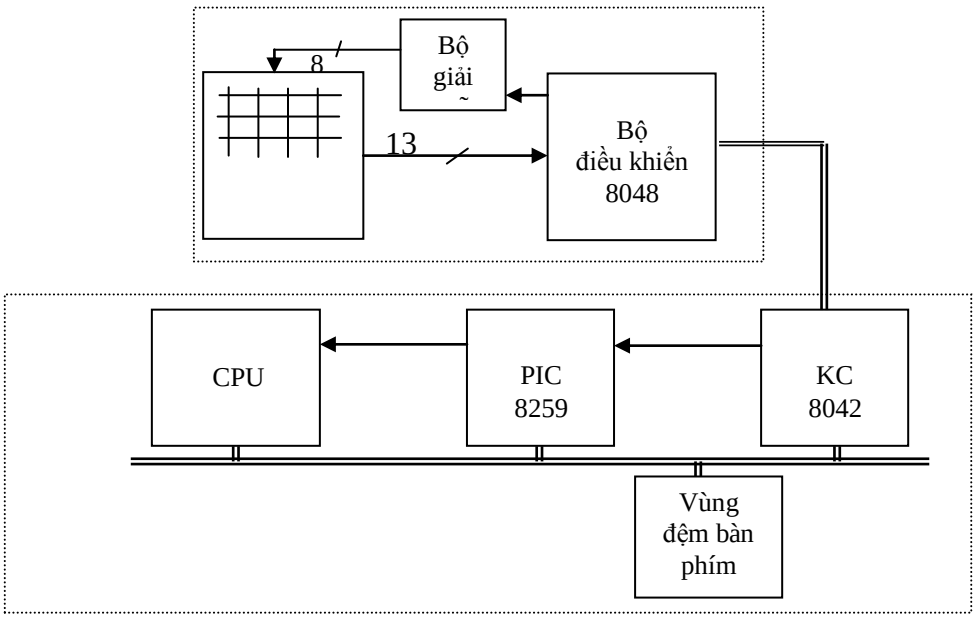
Bình thường phím luôn ở trạng thái nhả, khi phím nhả thì hai tiếp điểm không được nối với nhau, đầu ra có mức điện áp dương tương ứng với mức logic "1". Khi phím được nhấn thì hai tiếp điểm được nối với nhau qua công tắc phím và đầu ra có mức điện áp bằng 0V tương ứng mức logic "0".

Để mỗi lần nhấn phím có một mã phím tương ứng được tạo ra, cần sắp xếp hệ thống phím dưới dạng ma trận phím.

Ma trận phím gồm các dây hàng và các dây cột giao nhau nhưng không tiếp xúc với nhau. Các công tắc phím được đặt ở chỗ giao của hàng và cột. Hai tiếp điểm của công tắc nằm ở trên hàng và cột tại chỗ giao nhau đó. Mỗi khi phím được nhấn thì hai dây hàng và cột được nối với nhau qua hai tiếp điểm của công tắc tại chỗ giao nhau .

V.2.1 Hệ thống bàn phím của máy vi tính

Hệ thống bàn phím của máy vi tính gồm hai phần bàn phím và thiết bị giao diện bàn phím, được kết nối và trao đổi thông tin theo kiểu "chủ" "thợ".



Hình V.4 – Sơ đồ ghép nối bàn phím (keyboard) với hệ thống máy tính

Bàn phím là tổ hợp của ma trận 8x13 phím và mạch vi điều khiển µP8048. Mạch µC8048 là một hệ vi xử lý nhỏ được tích hợp trên một đơn chip. Mạch 8048 bao gồm CPU, bộ nhớ ROM chứa chương trình điều khiển quét và tạo mã phím, RAM chứa dữ liệu của chương trình điều khiển, hai cổng vào/ra P1và P2, một cổng dữ liệu 8 bit. Mạch 8048 tuần tự đưa mã nhị phân 3 bit ra tại cổng P2, qua bộ giải mã 3/8 tạo ra tín hiệu quét bàn phím. Tại thời điểm mã 3 bit được đưa ra, mạch µP8048 thực hiện đọc tín hiệu 13 bit từ ma trận phím vào cổng P1, từ đây tạo ra mã phím (mã quét) của phím được nhấn. Khi phím được nhả một mã phím (mã quét) cũng được tạo ra bằng cách cộng mã phím nhấn với 80H.

Mạch µP8048, được nuôi bằng nguồn từ máy tính, thực hiện trao đổi thông tin với thiết bị giao diện bàn phím KC 8042 theo kiểu nối tiếp đồng bộ. KC 8042 có cấu trúc tương tự mạch µP8048. KC 8042 đóng vai trò "chủ", 8048 đóng vai trò "thợ" trong các quá trình truyền tin thông qua hai dây tín hiệu: dây "DATA" và dây "CLOCK".

Dây "DATA" truyền tín hiệu dữ liệu nối tiếp giữa µP8048 và KC 8042 . Tín hiệu nối tiếp bao gồm: bit START, 8 bit dữ liệu, 1 bit PARITY, 1 bit STOP. Quá trình trao đổi thông tin giữa µP8048 và KC 8042 được đồng bộ bởi tín hiệu trên dây "CLOCK".

V.2.2 Quá trình truyền dữ liệu từ bàn phím cho CPU

Mạch µP8048 luôn phải kiểm tra trạng thái truyền tin qua hai dây "DATA" và "CLOCK" trước khi phát đi mã phím . Khi KC 8042 đặt "DATA" = 0 và "CLOCK"=1 thì 8048 phải nhận các chỉ lệnh từ KC 8042. Khi KC 8042 đặt "DATA " = 1 và "CLOCK" = 1 thì µP8048 được quyền truyền mã phím cho máy tính. Quá trình truyền dữ liệu được đồng bộ bằng dãy xung đồng bộ do µP8048 phát ra trên dây "CLOCK".

Khi KC 8042 nhận được mã phím dạng nối tiếp, nó loại bỏ các bit tạo khung dữ liệu truyền, chuyển mã phím vào thanh ghi tạm và phát ra yêu cầu ngắt IRQ1 cho hệ thống ngắt cứng. Hệ thống ngắt cứng sẽ kích hoạt chương trình phục vụ bàn phím 09H (chương trình phục vụ ngắt 09H) nằm ở BIOS. Chương trình phục vụ bàn phím 09H có chức năng dịch mã phím thành mã hai byte và chứa vào vùng đệm bàn phím.

Chương trình phục vụ bàn phím 09H trước hết kiểm tra (mã) các phím trượt (Shift, Alt, Ctrl) và các phím đặc biệt (ScrollLock, NumLock , CapsLock, Insert) trước khi dịch mã phím sang mã hai byte.

Mã hai byte được chương trình phục vụ bàn phím 09H tạo ra có cấu trúc tuỳ thuộc mã phím hoặc tổ hợp mã phím nhận được. Nếu nhận được mã của phím ký tự thì byte thấp của mã hai byte chứa mã ASCII của ký tự tương ứng, byte cao chứa mã phím (mã quét phím). Khi chương trình phục vụ bàn phím 09H nhận được mã các phím không phải là ký tự thì byte thấp của mã hai byte có giá trị 0, byte cao chứa mã phím mở rộng.

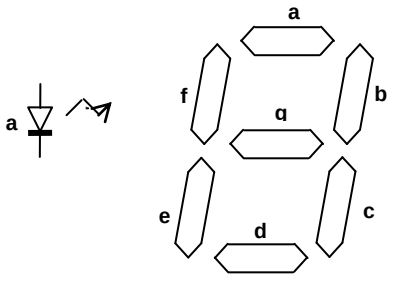
Vùng đệm bàn phím có kích thước 32 byte nằm trên bộ nhớ chính tại địa chỉ 0000H:041EH . Trạng thái của các phím trượt và các phím đặc biệt được chứa ở hai ô nhớ 0000H:0417H và 0000H:0418H . Có thể truy nhập vùng đệm bàn phím để đọc thông tin về bàn phím nhờ chương trình ngắt 16H của BIOS.

Chương trình phục vụ bàn phím 09H cũng xử lý các trường hợp đặc biệt như:

- Khi phím được nhấn quá lâu (ví dụ quá 0.5 giây) và KC 8042 không nhận được mã phím nhả, nó sẽ gửi ra cho đơn vị xử lý trung tâm mã của phím được nhấn.
- Khi nhận được tổ hợp các phím Ctrl+Alt+Del nó sẽ khởi động lại máy tính.
- Khi nhận được mã phím PrintScreen nó sẽ kích hoạt ngắt 05H của BIOS.
- Khi nhận được mã phím Ctrl+Break nó sẽ kích hoạt ngắt 1BH của BIOS.

V.3 Mạch điều khiển và lập trình chỉ thị 7-segments

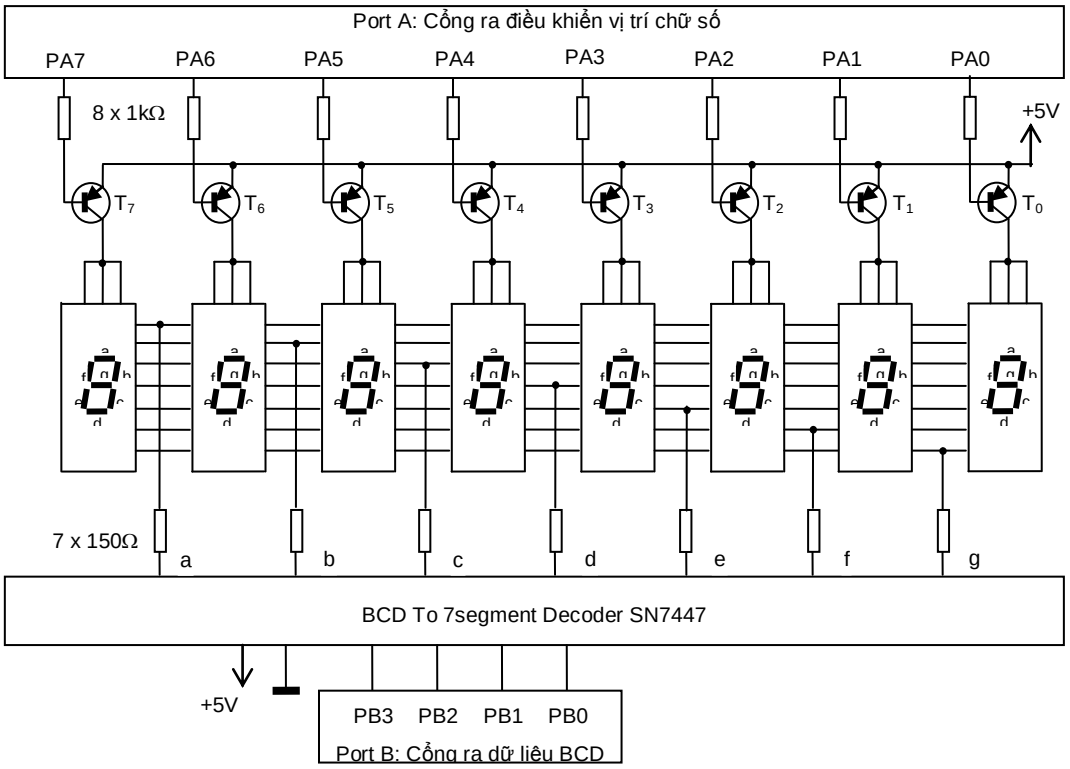
Hiển thị 7 thanh (7-segment Light Emitting Diode – LED Display) là loại đơn giản nhất nhận tín hiệu ra và hiển thị dưới dạng phát sáng. Có thể sử dụng vi mạch này để hiển thị các ký tự số từ 0 đến 9. Khi có dòng điện chạy qua, diode sẽ phát sáng.



	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	0	0	1	1
A	1	1	1	0	1	1	1
B	0	0	1	1	1	1	1
C	1	0	0	1	1	1	0
D	0	1	1	1	1	0	1
E	1	0	0	1	1	1	1
F	1	0	0	0	1	1	1

Hình V.5 là sơ đồ mạch hiển thị 8 digits sử dụng các vi mạch hiển thị 7 segment sử dụng 2 cổng của PPI-8255 theo phương pháp điều khiển hiển thị đa công (Multiplexing) đồng bộ. Các thanh sáng a, b, c, …, g của các mạch hiển thị 7 thanh được nối song song với nhau và nối với đầu ra của giải mã BCD-7segment SN7447. Việc cấp nguồn nuôi cho mạch hiển thị (1 digit) được đóng ngắt bởi một transistor PNP làm việc ở chế độ khoá đóng mở nhờ xung điều khiển từ một lối ra của cổng A của PPI-8255. Như vậy, tại một thời điểm, bằng cách lập trình cho PPI-8255, ta sẽ điều khiển để duy nhất một mạch hiển thị phát sáng. Nếu tần số của quá trình phát sáng đạt đến khoảng 15 đến 20 lần/sec, không xảy ra hiện tượng nhấp nháy khi theo dõi.

Dữ liệu cần hiển thị ở dạng mã BCD (4-bit) được đưa ra mạch giả mã hiển thị 7 thanh SN7447 qua 4 dây tương ứng của cổng B, đồng thời vị trí của digit cần hiển thị sẽ được điều khiển phát sáng bằng cách đưa điện áp mức "0" lên lối ra tương ứng trên cổng A để làm thông Transistor cấp nguồn cho mạch 7 segment tương ứng. Như vậy bằng cách lập trình "quét" lần lượt vòng qua tất cả các digit, có thể điều khiển hiển thị một dữ liệu gồm tối đa 8 chữ số.

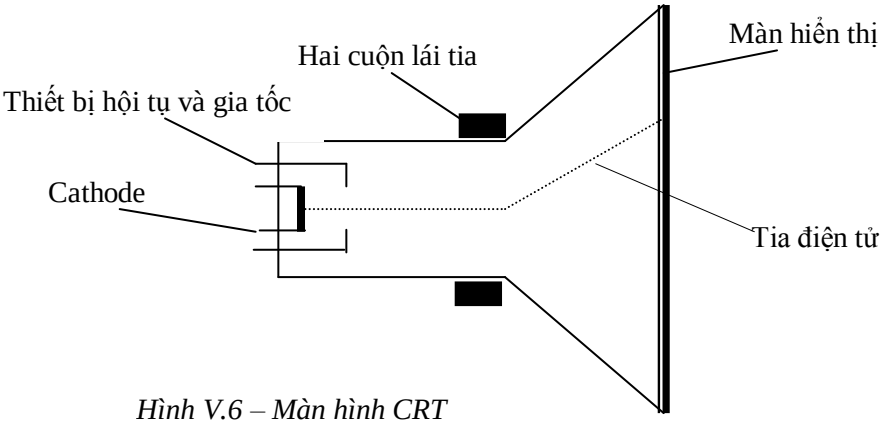


Hình V.5 – Sơ đồ nguyên lý mạch điều khiển bảng hiển thị 8 ký tự số sử dụng PPI 8255 theo phương pháp quét động

V.4 Màn hình (Monitor)

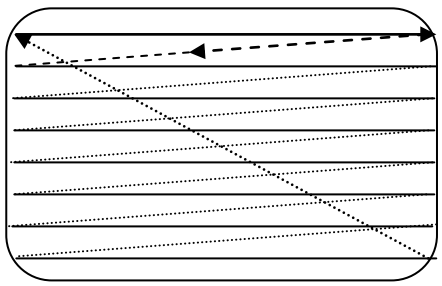
V.4.1 Màn hình ống tia âm cực CRT (Cathode Ray Tube)

Màn hình ống tia âm cực CRT là thiết bị hiển thị thông dụng nhất hiện nay. Màn hình CRT có cấu tạo như sau :



Hình V.6 – Màn hình CRT

Màn hình CRT là một ống thủy tinh chân không với các bộ phận: cathode phát xạ điện tử, ống phóng tia điện tử, cuộn lái tia và màn hiển thị. Cathode bằng kim loại được nối với điện áp âm , được đốt nóng và tạo ra các điện tử tự do . Màn hiển thị được phủ một lớp chất liệu phát quang và dẫn điện, được nối với điện áp dương và đóng vai trò một anode. Dưới tác dụng của điện trường cường độ cao trong ống phóng, điện tử rời khỏi cathode, được hội tụ thành chùm tia hướng về phía màn hiển thị. Cuộn lái tia có tác dụng lái chùm tia điện tử dịch chuyển theo hai chiều dọc và ngang màn hình. Khi chùm tia điện tử đập vào màn hiển thị sẽ tạo nên một điểm phát sáng. Cường độ điểm sáng phụ thuộc vào cường độ chùm tia và chất liệu phát sáng. Khi chùm tia mất đi hoặc chuyển hướng thì điểm vẫn còn lưu sáng một khoảng thời gian ngắn sau đó, thời gian lưu sáng phụ thuộc vào chất liệu phát sáng và cường độ chùm tia.

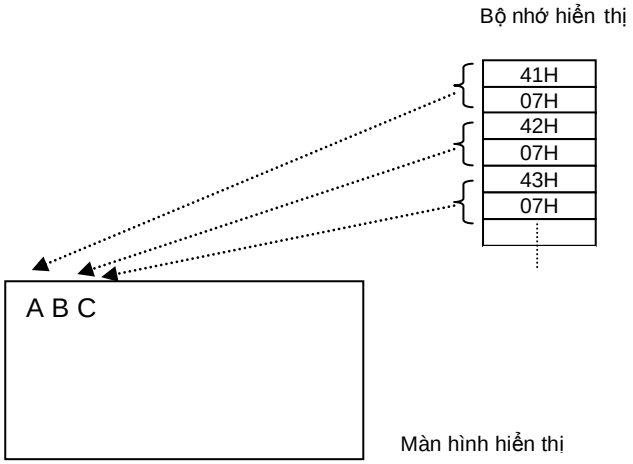


Ảnh trên màn hình CRT được tạo từ các điểm ảnh. Điểm ảnh được tạo ra khi cường độ chùm tia điện tử được tăng lên, điểm ảnh không xuất hiện khi chùm tia bị tắt đi. Các điểm ảnh được tạo theo từng dòng, từ trên xuống dưới . Một ảnh hoàn chỉnh được tạo ra trên màn hiển thị bởi các

dòng chứa các điểm ảnh. Các điểm ảnh chỉ tồn tại trong một thời gian rất ngắn. Để có thể quan sát được ảnh cần làm tươi các điểm ảnh theo một chu kỳ xác định. Các điểm ảnh được làm tươi theo từng dòng, bắt đầu từ dòng thứ nhất. Các dòng được làm tươi tuần tự từ trên xuống dưới. Khi dòng cuối cùng được quét xong, quá trình làm tươi được bắt đầu lại từ dòng đầu tiên (hình vẽ).

V.4.2 Ghép nối màn hình với hệ Vi xử lý

Các thiết bị hiển thị được sử dụng ở máy vi tính PC đều là loại ánh xạ bộ nhớ. Bộ nhớ này được cả đơn vị xử lý trung tâm và thiết bị điều khiển màn hình cùng truy nhập và được gọi là bộ nhớ hiển thị. Thông tin cần hiển thị được đưa ra bộ nhớ hiển thị, thiết bị điều khiển màn hình CRTC liên tục đọc bộ nhớ này để đưa ra màn hình. Hình vẽ sau đây minh họa nguyên tắc ánh xạ từ bộ nhớ hiển thị ra màn hình trong chế độ văn bản :



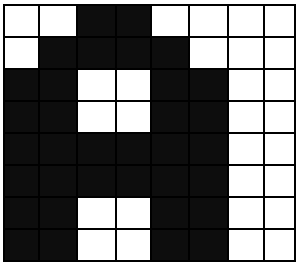
Hình V.7 – Hiển thị ký tự trên màn hình CRT theo nguyên tắc ánh xạ bộ nhớ

Mỗi một ký tự trên màn hình là một ánh xạ của một ô nhớ hai byte trong bộ nhớ hiển thị. Byte đầu chứa mã ASCII của ký tự, byte thứ hai chứa thuộc tính (màu nền, màu chữ, có/không nhấp nháy) của ký tự. Vị trí của mã ký tự trong bộ nhớ xác định vị trí ký tự trên màn hình. Mã ký tự đầu tiên trong bộ nhớ hiển thị (ví dụ : mã 41H) được ánh xạ thành ký tự (ký tự A) lên góc trái trên của màn hiển thị, mã ký tự tiếp theo được ánh xạ thành ký tự tiếp theo v.v.

Phương pháp ánh xạ bộ nhớ cho phép chương trình máy tính có thể dễ dàng thay đổi nội dung màn hiển thị bằng cách thay đổi nội dung của bộ nhớ hiển thị.

Mỗi ký tự được hiển thị trên màn hình dưới dạng một ma trận 8x8(*) điểm ảnh sáng/tối như trên hình vẽ :

(*) *Cũng có những trường hợp sử dụng ma trận 5x7, 7x9, 7x12 và 9x14 điểm*

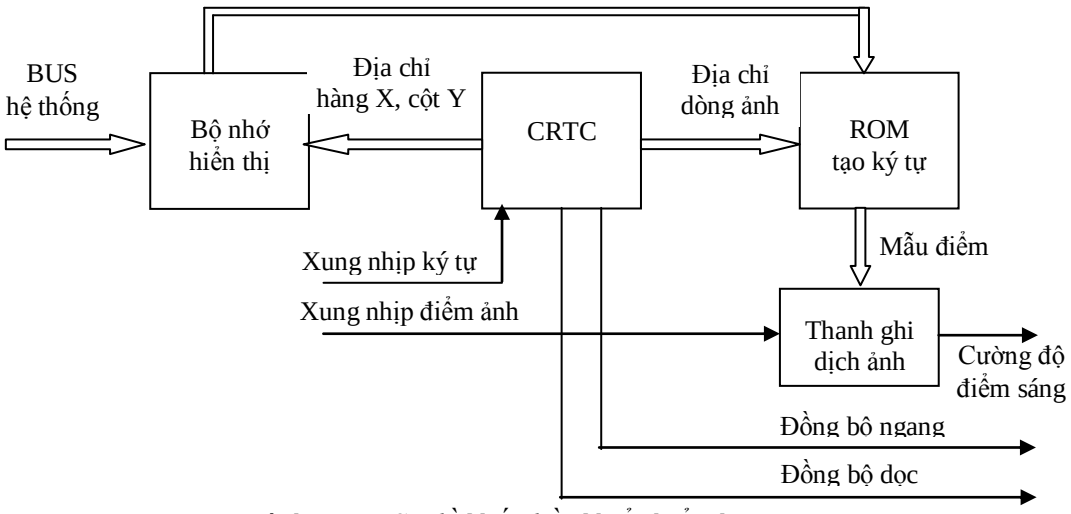


Phương pháp hiển thị ánh xạ bộ nhớ không hoàn toàn phù hợp với việc hiển thị các đối tượng có hình dạng không bình thường và chuyển động nhanh, đáp ứng thời gian thực bị chậm vì cần phải thao tác nhiều điểm ảnh để dịch chuyển đối tượng.

V.4.3 Bộ điều khiển màn hình CRTC

Thiết bị giao diện màn hình (bộ điều khiển màn hình) CRTC thực hiện việc chuyển mã ký tự trong bộ nhớ hiển thị thành ký tự hiện trên màn hình. ở chế độ văn bản các mẫu ký tự chỉ được hiển thị ở các vị trí hàng và cột cố định (25 hàng x 80 cột).

Sơ đồ nguyên lý của thiết bị giao diện màn hình ở chế độ văn bản như sau:



Hình V.8 – Sơ đồ khối điều khiển hiển thị CRTC

Mỗi một ký tự trên màn hình chứa nhiều hàng điểm ảnh. CRTC có nhiệm vụ chuyển mỗi mã ASCII trong bộ nhớ hiển thị thành chuỗi các mẫu điểm ảnh, đưa mỗi mẫu nằm lên một dòng màn hình. Điều này được thực hiện nhờ bộ ROM tạo ký tự. ROM tạo ký tự chứa các hộp mẫu ký tự, mỗi hộp mẫu ký tự có kích thước 8 byte mang thông tin về ma trận điểm ảnh của một ký tự. Ví dụ hộp mẫu ký tự A có dạng sau :

00110000
01111000
11001100
11001100
11111100
11111100
11001100
11001100

Nếu cần hiển thị 256 ký tự ASCII cần một ROM 2Kbyte, đủ chứa 256 hộp mẫu ký tự, mỗi hộp mẫu chiếm 8 ô nhớ liền nhau. Các hộp mẫu ký tự trong bộ ROM tạo ký tự được định vị bằng địa chỉ 11 bit, trong đó 8 bit địa chỉ cao xác định vị trí của hộp trong ROM, 3 bit địa chỉ thấp xác định vị trí của từng byte mẫu điểm ảnh trong hộp đó. Các mẫu ký tự được đặt trong ROM theo trật tự của bảng mã ASCII.

Nguyên lý hoạt động của thiết bị giao diện màn hình trong chế độ văn bản như sau: Giả sử cần hiển thị hai ký tự A và B tại các vị trí hàng 0 - cột 0 và hàng 0 - cột 1 trên màn hình . Mã ASCII của hai ký tự được đặt tại hai vị trí tương ứng trong bộ nhớ hiển thị (xem hình vẽ ở mục 2.2).

CRTC gửi địa chỉ hàng và cột màn hình cho bộ nhớ hiển thị (hàng=0, cột=0). Bộ nhớ hiển thị gửi mã ASCII của ký tự (ký tự A) cho ROM, mã ASCII của ký tự mang thông tin về địa chỉ của hộp mẫu ký tự trong ROM (8 bit địa chỉ cao). Tại cùng thời điểm này CRTC gửi địa chỉ của dòng mẫu điểm ảnh (dòng mẫu điểm 0) cho ROM (3 bit địa chỉ thấp). Hai địa chỉ này được kết hợp lại tạo thành địa chỉ (11 bit) cho phép truy nhập vào dòng mẫu điểm ảnh đầu tiên của ký tự (ký tự A) trong ROM và xuất nó ra thanh ghi dịch ảnh. Từ thanh ghi dịch ảnh, từng bit mẫu ảnh tuần tự được đưa ra màn hình.

Khi tất cả các bit mẫu ảnh từ thanh ghi dịch được đẩy ra màn hình, CRTC tiếp tục gửi địa chỉ hàng-cột (hàng=0, cột=1) cho bộ nhớ hiển thị và gửi địa chỉ dòng mẫu điểm ảnh (dòng mẫu điểm 0) cho ROM, bộ nhớ hiển thị gửi mã ASCII của ký tự (ký tự B) cho ROM. Dòng mẫu điểm ảnh đầu tiên của ký tự (ký tự B) được xuất ra thanh ghi dịch ảnh. Tương tự như thế các dòng mẫu điểm đầu tiên của tất cả các ký tự trên cùng một hàng màn hình được hiển thị, cho đến ký tự cuối cùng trên hàng.

CRTC tiếp tục gửi địa chỉ hàng-cột (hàng=0, cột=0) đến bộ nhớ hiển thị, nhưng địa chỉ dòng mẫu điểm ảnh bây giờ là 1(dòng mẫu điểm 1) cho ROM. Bộ nhớ hiển thị gửi mã ASCII của ký tự A cho ROM, ROM xuất ra dòng mẫu điểm ảnh 1 của ký tự A. Dòng 1 của ký tự B được xuất ra theo cách tương tự. Các dòng điểm ảnh tiếp theo của ký tự lần lượt được hiển thị lên màn hình cho đến khi tất cả các dòng điểm ảnh của hàng văn bản đầu tiên (hàng 0) được hiển thị trên màn hình.

Các hàng văn bản tiếp theo cũng được hiển thị theo phương pháp nói trên.

Trên thực tế hoạt động của CRTC phức tạp hơn. CRTC phải có khả năng hiển thị ở chế độ đồ họa. CRTC phải theo dõi thông tin về thuộc

tính của ký tự hiển thị, phải tạo ra điểm nháy. CRTC cũng phải tạo ra hai tín hiệu đồng bộ ảnh ngang - dọc và làm tươi màn hình. Tần số làm tươi tối thiểu là 50 Hz.

PHỤ LỤC

PHỤ LỤC A

Bảng tóm tắt hệ lệnh của Trung tâm Vi xử lý họ x86

Từ gợi nhớ	Chức năng	Từ gợi nhớ	Chức năng
Các lệnh họ 80x86			
AAA	Chỉnh sau phép cộng 2 số dạng ASCII	CMP	So sánh toán hạng đích và gốc
AAD	Chỉnh hai số mã ASCII trước phép chia	CMPS	So sánh chuỗi Byte hay từ
AAM	Chỉnh sau phép nhân 2 số mã ASCII	CMPSB	So sánh xâu (byte)
AAS	Chỉnh sau phép trừ 2 số mã ASCII	CMPSW	So sánh xâu (từ)
ADC	Cộng có cờ nhớ	CWD	Biến đổi từ thành từ kép
ADD	Cộng 2 toán hạng	DAA	Hiệu chỉnh thập phân sau phép cộng
AND	Và từng bít tương ứng của 2 toán hạng	DAS	Hiệu chỉnh thập phân sau phép trừ
CALL	Gọi chương trình con	DEC	Giảm toán hạng đích đi 1
CBW	Chuyển byte thành từ	DIV	Chia không dấu
CLC	Xoá cờ nhớ	ESC	Thoát
CLD	Xoá cờ hướng	HLT	Treo
CLI	Xoá cờ ngắt	IDIV	Chia số nguyên
CMC	Lấy bù cờ nhớ	IMUL	Nhân số nguyên
IN	Đọc cổng vào ra	JS	Nhảy nếu có cờ dấu
INC	Tăng toán hạng đích lên 1	JZ	Nhảy nếu bằng 0
INT	Gọi ngắt	LAHF	Nạp 8 bit thấp của cờ vào AH
INTO	Ngắt nếu bị tràn	LDS	Nạp ô nhớ từ kép vào thanh ghi đoạn dữ liệu
IRET	Trở về chỗ bị ngắt	LEA	Nạp địa chỉ hiệu dụng
JA	Nhảy nếu ở trên	LES	Nạp con trỏ khi dùng ES
JAE	Nhảy nếu ở trên hoặc bằng	LOCK	Khoá bus
JB	Nhảy nếu thấp hơn	LODS	Nạp xâu
JBE	Nhảy nếu thấp hơn hoặc bằng	LODSB	Nạp xâu (byte)
JC	Nhảy nếu có cờ nhớ	LODSW	Nạp xâu (từ)
JCXZ	Nhảy nếu CX = 0	LOOP	Vòng lặp
JE	Nhảy nếu bằng	LOOPE	Lặp lại trong khi bằng
JG	Nhảy nếu lớn hơn	LOOPNE	Lặp lại khi không bằng
JGE	Nhảy nếu lớn hơn hoặc bằng	LOOPNZ	Lặp khi không bằng 0
JL	Nhảy nếu nhỏ hơn	LOOPZ	Lặp khi bằng 0
JLE	Nhảy nếu nhỏ hơn hoặc bằng	MOV	Chuyển nguồn tới đích
JMP	Nhảy không điều kiện	MOVS	Chuyển xâu
JNA	Nhảy nếu không ở trên	MOVSB	Chuyển xâu (byte)
JNAE	Nhảy nếu không ở trên hoặc bằng	MOVSW	Chuyển xâu (từ)
JNB	Nhảy nếu không ở dưới	MUL	Phép nhân
JNBE	Nhảy nếu không ở dưới hoặc bằng	NEG	Đảo dấu hay lấy bù 2
JNC	Nhảy nếu không có cờ nhớ	NOP	Không hành động

JNE	Nhảy nếu không bằng	NOT	Đảo dấu (lấy bù 1)
JNG	Nhảy nếu không lớn hơn	OR	Hoặc các bit tương ứng của 2 toán hạng
JNGE	Nhảy nếu không lớn hơn hoặc bằng	OUT	Viết cổng vào/ra
JNL	Nhảy nếu không nhỏ hơn	POP	Hồi phục nộng dung (các thanh ghi, ...)
JNLE	nhảy nếu không nhỏ hơn hoặc bằng	POPF	Hồi phục nội dung các cờ
JNO	Nhảy nếu không tràn	PUSH	Đẩy nội dung (thanh ghi, ...) vào ngăn xếp
JNP	Nhảy nếu không có cờ chẵn lẻ	PUSHF	Đẩy nội dung cờ vào ngăn xếp
JNS	Nhảy nếu không có cờ dấu	RCL	Quay trái qua cờ nhớ
JNZ	Nhảy nếu không bằng 0	RCR	Quay phải qua cờ nhớ
JO	Nhảy nếu có cờ tràn	REP	Lặp lại
JP	Nhảy nếu có cờ chẵn lẻ	REPE	Lặp lại khi bằng
JPE	Nhảy nếu có cờ lẻ chẵn	REPNE	Lặp lại khi không bằng
JPO	Nhảy nếu lẻ lẻ	REPNZ	Lặp lại khi không bằng 0
REPZ	Lặp lại trong khi bằng 0	STC	Đặt cờ nhớ
RET	Trở về	STD	Đặt cờ hướng
ROL	Quay trái	STI	Đặt cờ ngắt
ROR	Quay phải	STOS	Lưu trữ xâu
SAHF	Lưu trữ AH vào byte thấp của cờ	STOSB	Lưu trữ xâu (byte)
SAL	Dịch trái số học	STOSW	Lưu trữ xâu (từ)
SAR	Dịch phải số học	SUB	Phép trừ
SBB	Trừ có mượn	TEST	Kiểm tra (nhân logic đích với gốc)
SCAS	Quét xâu	WAIT	Đợi
SCASB	Quét xâu (byte)	XCHG	Tráo đổi
SCANW	Quét xâu (từ)	XLAT	Chuyển đổi bảng
SHL	Dịch trái	XOR	Hoặc tuyệt đối tương ứng 2 số
SHR	Dịch phải		
Các lệnh chỉ có trong 80286, 80386 và 80486			
ARPL	Chỉnh trường RPL của bộ chọn	LSL	Nạp độ dài đoạn nhớ
BOUND	Kiểm tra biên của trường	LTR	Nạp thanh ghi nhiệm vụ
CLTS	Xoá cờ chuyển nhiệm vụ	OUTS	Xuất xâu ra cổng vào/ra
ENTER	Tạo khối các thông số để vào CTC	POPA	Phục hồi tất cả các tranh ghi đa năng
INS	Nhập xâu từ cổng vào/ra	PUSHA	Đẩy vào ngăn xếp các thanh ghi đa năng
LAR	Nạp quyền thâm nhập	SGDT	Lưu trữ thanh ghi bảng bộ mô tả toàn cục
LEAVE	RA khỏi CTC (chương trình con)	SIDT	Lưu trữ thanh ghi bảng bộ mô tả ngắt
LGDT	Nạp thanh ghi bảng bộ mô tả toàn cục	SLDT	Lưu trữ thanh ghi bảng bộ mô tả cục bộ
LIDT	Nạp thanh ghi bảng bộ mô tả ngắt	SMSW	Lưu trữ từ trạng thái máy
LLDT	Nạp thanh ghi bảng bộ mô tả cục bộ	STR	Lưu trữ thanh ghi nhiệm vụ
LMSW	Nạp từ trạng thái máy	VERR	Kiểm tra một bộ chọn đoạn để đọc
		VERW	Kiểm tra một bộ chọn đoạn để viết
Các lệnh chỉ có tronmg 80386 và 80486			
BSF	Quét bit về phía trước	SETL	Đặt byte nếu nhỏ hơn
BSR	Quét bit về phía sau	SETLE	Đặt byte nếu nhỏ hơn hoặc bằng
BT	Kiểm tra bit	SETNA	Đặt byte nếu không ở trên

BTC	Kiểm tra và đảo bit	SETNAE	Đặt byte nếu không ở trên hoặc bằng
BTR	Kiểm tra và xoá bit	SETNB	Đặt byte nếu không ở dưới
BTS	Kiểm tra và đặt bit	SETNBE	Đặt byte nếu không ở dưới hoặc bằng
CDQ	Biến đổi từ kép thành từ kép bốn	SETNC	Đặt byte nếu không nhớ
CMPSD	So sánh xâu (từ kép)	SETNE	Đặt byte nếu không bằng
CWDE	Biến đổi từ thành từ kép trong EAX	SETNG	Đặt byte nếu không lớn hơn
JECXZ	Nhảy nếu ECX bằng 0	SETNGE	Đặt byte nếu không lớn hơn hoặc bằng
LFS	Nạp con trỏ khi dùng FS	SETNL	Đặt byte nếu không nhỏ hơn
LGS	Nạp con trỏ khi dùng GS	SETNLE	Đặt byte nếu không nhỏ hơn hoặc bằng
LSS	Nạp con trỏ khi dùng SS	SETNO	Đặt byte nếu không tràn
LODSD	Nạp xâu (từ kép)	SETNP	Đặt byte nếu không có chẵn lẻ
MOVSD	Chuyển xâu (từ kép)	SETNS	Đặt byte nếu không dấu
MOVSX	Chuyển với Sigh-eXtend	SETNZ	Đặt byte nếu không bằng 0
MOVZX	Chuyển với Zero-eXtend	SETO	Đặt byte nếu tràn
SCASD	Quét xâu (từ kép)	SETP	Đặt byte nếu có chẵn lẻ
SETA	Đặt byte nếu ở trên	SETPE	Đặt byte nếu chẵn lẻ chẵn
SETAE	đặt byte nếu ở trên hoặc bằng	SETPO	Đặt byte nếu có chẵn lẻ lẻ
SETB	Đặt byte nếu ở dưới	SETS	Đặt byte nếu có dấu
SETBE	Đặt byte nếu ở dưới hoặc bằng	SETZ	Đặt byte nếu bằng 0
SETC	Đặt byte nếu có cờ nhớ	SHLD	Dịch trái(từ kép)
SETE	Đặt byte nếu bằng	SHRD	Dịch phải (từ kép)
SETG	Đặt byte nếu lớn hơn	STOSD	Lưu trữ xâu (từ kép)
SETGE	ĐẶt byte nếu lớn hơn hoặc bằng		
Các lệnh chỉ có trong 80486			
BSWAP	Hoán chuyển byte	INVLPG	Vô hiệu hoá TLB (cho chế độ trang)
CMPXCHG	So sánh và tráo đổi	WBINVD	Ghi trở lại bộ nhớ chính vào nhớ ngầm
INVD	Vô hiệu hoá bộ nhớ ngầm	XADD	Hoán chuyển và cộng

PHỤ LỤC B

Bảng luỹ thừa 2^n

2^n	n	2^{-n}
1	0	1
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.015625
128	7	0.0078125
256	8	0.00390625
512	9	0.001953125
1,024	10	0.000976563
2,048	11	0.0004882815
4,096	12	0.00024414125
8,192	13	0.000122070625
16,384	14	0.000061035156
32,768	15	0.000030517578
65,536	16	0.000015258789
131,072	17	0.000007629395
262,144	18	0.000003814697
524,288	19	0.000001907349
1,048,576	20	0.000000953674
2,097,152	21	0.000000476837
4,194,304	22	0.000000238419
8,388,608	23	0.000000119209
16,777,216	24	0.000000059605
33,554,432	25	0.000000029802
67,108,864	26	0.000000014901
134,217,728	27	0.000000007451
268,435,456	28	0.000000003725
536,870,912	29	0.000000001863
1,073,741,824	30	0.000000000931
2,147,483,648	31	0.000000000466
4,294,967,296	32	0.000000000233

Từ 2^{-14} là giá trị đã làm tròn lấy 10 số sau dấu phẩy

PHỤ LỤC C

Bảng mã ASCII

ROW			Dec	0	16	32	48	64	80	96	112
			Bin	000	001	010	011	100	101	110	111
Dec	Bin	Hex	Hex	0	10	20	30	40	50	60	70
0	0000	0		NUL	DLE	SP	0	@	P	`	p
1	0001	1		SOH	XON	!	1	A	Q	a	q
2	0010	2		STX	DC2	"	2	B	R	b	r
3	0011	3		ETX	XOFF	#	3	C	S	c	s
4	0100	4		EOT	DC4	$	4	D	T	d	t
5	0101	5		ENQ	NAK	%	5	E	U	e	u
6	0110	6		ACK	SYN	&	6	F	V	f	v
7	0111	7		BEL	ETB	'	7	G	W	g	w
8	1000	8		BS	CAN	(	8	H	X	h	x
9	1001	9		HT	EM	)	9	I	Y	i	y
10	1010	A		LF	SUB	*	:	J	Z	j	z
11	1011	B		VT	ESC	+	;	K	[	k	{
12	1100	C		FF	FS	,	<	L	\	l	|
13	1101	D		CR	GS	-	=	M	]	m	}
14	1110	E		SO	RS	.	>	N	^	n	~
15	1111	F		SI	US	/	?	O	_	o	DEL

PHỤ LỤC D
CÁC NHÓM LỆNH CỦA µC8051

1. Tạo vòng lặp và lệnh nhảy:

a. Tạo vòng lặp

- Quá trình lặp lại chuỗi lệnh với một số lần nhất định gọi là vòng lặp.

Vòng lặp trong 8051 được thực hiện bằng lệnh "DJNZ thanh ghi, nhãn"

Để tổ chức vòng lặp lồng nhau cần sử dụng 2 thanh ghi để lưu số đếm.

b. Lệnh nhảy

Lệnh nhảy có điều kiện

Lệnh	Ý Nghĩa
JZ	Nhảy nếu A=0
JNZ	Nhảy nếu A khác 0
DJNZ	Giảm và nhảy nếu A khác 0
CJNE A,byte	Nhảy nếu A khác byte
CJNE re,#data	Nhảy nếu byte khác data
JC	Nhảy nếu CY=1
JNC	Nhảy nếu CY=0
JB	Nhảy nếu bit=1
JNB	Nhảy nếu bit=0
JBC	Nhảy nếu bit=1 và xoá nó

Lệnh nhảy không điều kiện :

Lệnh nhảy dài :

Là lệnh 3 byte. Byte đầu là mã lệnh, 2 byte còn lại là địa chỉ 16 bit của đích. Địa chỉ đích 2 byte cho phép lệnh có thể nhảy đến bất kỳ vị trí nhớ nào trong không gian nhớ 0000 đến FFFF

Lệnh nhảy ngắn :

Lệnh nhảy ngắn là lệnh 2 byte. Byte đầu tiên là mã lệnh, byte thứ 2 là địa chỉ tương đối của địa chỉ đích. Địa chỉ đích tương đối có nghĩa là so với giá trị của bộ đếm chương trình.

2. Lệnh gọi Call

Lệnh call dùng để gọi chương trình con

Lệnh gọi dài Lcall (longcall) :

Đây là lệnh 3 byte. Byte đầu là mã lệnh, 2 byte còn lại là địa chỉ của chương trình con đích. sau khi thực hiện xong 1 chương trình con, để 8051 biết được chỗ quay trở về thì địa chỉ của lệnh đứng ngay sau lệnh gọi Lcall sẽ được tự đông cất vào ngăn xếp.

Lệnh gọi tuyệt đối Acall :

Lệnh Acall là lệnh 2 byte, địa chỉ đích của chương trình con phải nằm trong khoảng 2 Kbyte địa chỉ vì chỉ có 11 bit của 2 byte được dùng để xác định địa chỉ. Thực tế một số biến thể 8051 chỉ có 1 Kbyte ROM trên chip. Trong những trường hợp đó, sử dụng lệnh Acall có thể tiết kiệm được một số byte bộ nhớ của không gian ROM chương trình so với lệnh Lcall.

3. Nhóm lệnh cơ bản của 8051:

Tập lệnh của 8051 được chia thành 5 nhóm:

- Số học.
- Luận lý.
- Chuyển dữ liệu.
- Chuyển điều khiển.
- Rẽ nhánh.

Các chi tiết thiết lập lệnh:

Rn	Thanh ghi R0 đến R7 của bank thanh ghi được chọn.
Data	8 bit địa chỉ vùng dữ liệu bên trong. Nó có thể là vùng RAM dữ liệu trong (0-127) hoặc các thanh ghi chức năng đặc biệt.
@Ri	8 bit vùng RAM dữ liệu trong (0-125) được đánh giá địa chỉ gián tiếp qua thanh ghi R0 hoặc R1.
#data	Hằng 8 bit chức trong câu lệnh.
#data 16	Hằng 16 bit chứa trong câu lệnh.
Addr16	16 bit địa chỉ đích được dùng trong lệnh LCALL và LJMP.
Addr11	11 bit địa chỉ đích được dùng trong lệnh LCALL và AJMP.
Rel	Byte offset 8 bit có dấu được dùng trong lệnh SJMP và những lệnh nhảy có điều kiện.
Bit	Bit được định địa chỉ trực tiếp trong RAM dữ liệu nội hoặc các thanh ghi chức năng đặc biệt.

a. Nhóm lệnh xử lý số học:

ADD A,Rn	(1byte 1 chu kỳ máy) : cộng nội dung thanh ghi Rn vào thanh ghi A.
ADD A,data	(21): Cộng trực tiếp 1 byte vào thanh ghi A.
ADD A,@Ri	(11): Cộng gián tiếp nội dung RAM chứa tại địa chỉ được khai báo trong Ri vào thanh ghi A.
ADD A,#data	(21): Cộng dữ liệu tức thời vào A.
ADD A,Rn	(11): Cộng thanh ghi và cờ nhớ vào A.
ADD A,data	(21): Cộng trực tiếp byte dữ liệu và cờ nhớ vào A.
ADDC A,@Ri	(11): Cộng gián tiếp nội dung RAM và cờ nhớ vào A.
ADDC A,#data	(21): Cộng dữ liệu tức thời và cờ nhớ vào A.
SUBB A,Rn	(11): Trừ nội dung thanh ghi A cho nội dung thanh ghi Rn

	và cờ nhớ.
SUBB A,data	(21): Trừ trực tiếp A cho một số và cờ nhớ.
SUBB A,@Ri	(11): Trừ gián tiếp A cho một số và cờ nhớ.
SUBB A,#data	(21): Trừ nội dung A cho một số tức thời và cờ nhớ.
INC A	(11): Tăng nội dung thanh ghi A lên 1.
INC Rn	(11): Tăng nội dung thanh ghi Rn lên 1.
INC data	(21): Tăng dữ liệu trực tiếp lên 1.
INC @Ri	(11): Tăng gián tiếp nội dung vùng RAM lên 1.
DEC A	(11): Giảm nội dung thanh ghi A xuống 1.
DEC Rn	(11): Giảm nội dung thanh ghi Rn xuống 1.
DEC data	(21): Giảm dữ liệu trực tiếp xuống 1
DEC @Ri	(11): Giảm gián tiếp nội dung vùng RAM xuống 1.
INC DPTR	(12): Tăng nội dng con trỏ dữ liệu lên 1.
MUL AB	(14): Nhân nội dung thanh ghi A với nội dung thanh ghi B.
DIV AB	(14): Chia nội dung thanh ghi A cho nội dung thanh ghi B.
DA A	(11): hiệu chỉnh thập phân thanh ghi A.

b. Nhóm lệnh luận lý:

ANL A,Rn	(11): AND nội dung thanh ghi A với nội dung thanh ghi Rn.
ANL A,data	(21): AND nội dung thanh ghi A với dữ liệu trực tiếp.
ANL A,@Ri	(11): AND nội dung thanh ghi A với dữ liệu gián tiếp trong RAM.
ANL A,#data	(21): AND nội dung thanh ghi với dữ liệu tức thời.
ANL data,A	(21): AND một dữ liệu trực tiếp với A.
ANL data,#data	(32): AND một dữ liệu trực tiếp với A một dữ liệu tức thời.
ANL C,bit	(22): AND cờ nhớ với 1 bit trực tiếp.

ANL C,/bit	(22): AND cờ nhớ với bù 1 bit trực tiếp.
ORL A,Rn	(11): OR thanh ghi A với thanh ghi Rn.
ORL A,data	(21): OR thanh ghi A với một dữ liệu trực tiếp.
ORL A,@Ri	(11): OR thanh ghi A với một dữ liệu gián tiếp.
ORL A,#data	(21): OR thanh ghi A với một dữ liệu tức thời.
ORL data,A	(21): OR một dữ liệu trực tiếp với thanh ghi A.
ORL data,#data	(31) :OR một dữ liệu trực tiếp với một dữ liệu tức thời.
ORL C,bit	(22): OR cờ nhớ với một bit trực tiếp.
ORL C,/bit	(22): OR cờ nhớ với bù của một bit trực tiếp.
XRL A,Rn	(11): XOR thanh ghi A với thanh ghi Rn.
XRL A,data	(21): XOR thanh ghi A với mộ dữ liệu trực tiếp.
XRL A,@Ri	(11): XOR thanh ghi A với một dữ liệu gián tiếp.
XRL A,#data	(21): XOR thanh ghi A với mộ dữ liệu tức thời.
XRL data,A	(21): XOR một dữ liệu trực tiếp với thanh ghi A.
XRL data,#data	(31): XOR một dữ liệu trực tiếp với một dữ liệu tức thời.
SETB C	(11): Đặt cờ nhớ.
SETB bit	(21): Đặt một bit trực tiếp.
CLR A	(11): Xóa thanh ghi A.
CLR C	(11): Xóa cờ nhớ.
CPL A	(11): Bù nội dung thanh ghi A.
CPL C	(11): Bù cờ nhớ.
CPL bit	(21): Bù một bit trực tiếp.
RL A	(11): Quay trái nội dung thanh ghi A.
RLC A	(11): Quay trái nội dung thanh ghi A qua cờ nhớ.
RR A	(11): Quay phải nội dung thanh ghi A.
RRC A	(11): Quay phải nội dung thanh ghi A qua cờ nhớ.

SWAP	(11): Quay trái nội dung thanh ghi A 1 nibble (1/2byte).

c. Nhóm lệnh chuyển dữ liệu:

MOV A,Rn	(11):Chuyển nội dung thanh ghi Rn vào thanh ghi A.
MOV A,data	(21): Chuyển dữ liệu trực tiếp vào thanh ghi A.
MOV A,@Ri	(11): Chuyển dữ liệu gián tiếp vào thanh ghi A.
MOV A,#data	(21): Chuyển dữ liệu tức thời vào thanh ghi A.
MOV Rn,data	(22): Chuyển dữ liệu trực tiếp vào thanh ghi Rn.
MOV Rn,#data	(21): Chuyển dữ liệu tức thời vào thanh ghi Rn.
MOV data,A	(21): Chuyển nội dung thanh ghi A vào một dữ liệu trực tiếp.
MOV data,Rn	(22): Chuyển nội dung thanh ghi Rn vào một dữ liệu trực tiếp.
MOV data,data	(32): Chuyển một dữ liệu trực tiếp vào một dữ liệu trực tiếp.
MOV data,@Ri	(22): Chuyển một dữ liệu gián tiếp vào một dữ liệu gián tiếp.
MOV data,#data	(32): Chuyển một dữ liệu tức thời vào một dữ liệu trực tiếp.
MOV @Ri,A	(11): Chuyển nội dung thanh ghi A vào một dữ liệu gián tiếp.
MOV @Ri,data	(22): Chuyển một dữ liệu trực tiếp vào một dữ liệu gián tiếp.
MOV @Ri,#data	(21): Chuyển dữ liệu tức thời vào dữ liệu gián tiếp.
MOV DPTR,#data	(32): Chuyển một hằng 16 bit vào thanh ghi con trỏ dữ liệu.
MOV C,bit	(21): Chuyển một bit trực tiếp vào cờ nhớ.
MOV bit,C	(22): Chuyển cờ nhớ vào một bit trực tiếp.

MOV A,@A+DPTR	(12): Chuyển byte bộ nhớ chương trình có địa chỉ là@A+DPRTvào thanh ghi A.
MOVC A,@A+PC	(12): Chuyển byte bộ nhớ chương trình có địa chỉ là @A+PC vào thanh ghi A.
MOVX A,@Ri	(12): Chuyển dữ liệu ngoài (8 bit địa chỉ) vào thanh ghi A.
MOVX A,@DPTR	(12): Chuyển dữ liệu ngoài (16 bit địa chỉ) vào thanh ghi A.
MOVX @Ri,A	(12): Chuyển nội dung A ra dữ liệu ngoài (8 bit địa chỉ).
MOVX @DPTR,A	(12): Chuyển nội dung A ra dữ liệu bên ngoài (16 bit địa chỉ).
PUSH data	(22): Chuyển dữ liệu trực tiếp vào ngăn xếp và tăng SP.
POP data	(22): Chuyển dữ liệu trực tiếp vào ngăn xếp và giảm SP.
XCH A,Rn	(11): Trao đổi dữ liệu giữa thanh ghi Rn v2 thanh ghi A.
XCH A,data	(21): Trao đổi giữa thanh ghi A và một dữ liệu trực tiếp.
XCH A,@Ri	(11): Trao đổi giữa thanh ghi A và một dữ liệu gián tiếp.
XCHD A,@R	(11): Trao đổi giữa nibble thấp (LSN) của thanh ghi A và LSN của dữ liệu gián tiếp.

d. Nhóm lệnh chuyền điều khiển:

ACALL addr11	(22): Gọi chương trình con dùng địa chỉ tuyệt đối.
LCALL addr16	(32): Gọi chương trình con dùng địa chỉ dài.
RET	(12): Trở về từ lệnh gọi chương trình con.

RETI	(12): Trở về từ lệnh gọi ngắt.
AJMP addr11	(22): Nhảy tuyệt đối.
LJMP addr16	(32): Nhảy dài.
SJMP rel	(22):Nhảy ngắn.
JMP @A+DPTR	(12): Nhảy gián tiếp từ con trỏ dữ liệu.
JZ rel	(22): Nhảy nếu A=0.
JNZ rel	(22): Nhảy nếu A không bằng 0.
JC rel	(22): Nhảy nếu cờ nhớ được đặt.
JNC rel	(22): Nhảy nếu cờ nhớ không được đặt.
JB bit,rel	(32): Nhảy tương đối nếu bit trực tiếp được đặt.
JNB bit,rel	(32):Nhảy tương đối nếu bit trực tiếp không được đặt.
JBC bit,rel	(32): Nhảy tương đối nếu bit trực tiếp được đặt, rồi xóa bit.
CJNE A,data,rel	(32): So sánh dữ liệu trực tiếp với A và nhảy nếu không bằng.
CJNE A,#data,rel	(32): So sánh dữ liệu tức thời với A và nhảy nếu không bằng.
CJNE Rn,#data,rel	(32): So sánh dữ liệu tức thời với nội dung thanh ghi Rn và nhảy nếu không bằng.
CJNE @Ri,#data,rel	(32): So sánh dữ liệu tức thời với dữ liệu gián tiếp và nhảy nếu không bằng.
DJNZ Rn,rel	(22): Giảm thanh ghi Rn và nhảy nếu không bằng.
DJNZ data	(32): Giảm dữ liệu trực tiếp và nhảy nếu không bằng.

TÀI LIỆU THAM KHẢO

[1] Nguyễn Tăng Cường, Phan Quốc Khánh: *Cấu trúc và lập trình họ Vi điều khiển 8051*. NXB KH&KT hà Nội-2004

[2] Vũ Chấn Hưng: *Giáo trình Kiến trúc máy tính* NXB Giao thông vận tải - Hà Nội 2002

[3] Văn Thế Minh: *Kỹ thuật Vi xử lý* – NXB Thống kê – Hà Nội 1983

[4] Phòng Kỹ thuật số - Viện Khoa học Tính toán và Điều khiển: *Kỹ thuật Vi xử lý* - Nhà Xuất bản Thống kê – Hà Nội 1983

[5] Alan Clements: *Principles of Computer Hardware* – PWS-KENT Publishing Company – Boston 1992

[6] Int$_e$l® Corporation: *Component Data Catalog 1982*

[7] David Hergert, Nancy Thibeault: *PC Architecture from Assembly Language To C. Prentice-Hall, Inc. 1997*

[8] Christopher L. Morgan and Mitchell Waite: 8086/8088 16-bit Micrro-Processor Primer – McGraw-Hill, Inc. 1982

[9] V.M. Rooney: *Microprocessors and Microcomputers* - McMilan Publishing Company – New York 1983

[10] James L., Turley: *Advanced 80386 programming techniques* - Osborne Mc Graw-Hill 1988

Chương 1
KIẾN TRÚC VÀ HOẠT ĐỘNG CỦA HỆ VI XỬ LÝ / MÁY TÍNH

1. Cấu trúc luận lý

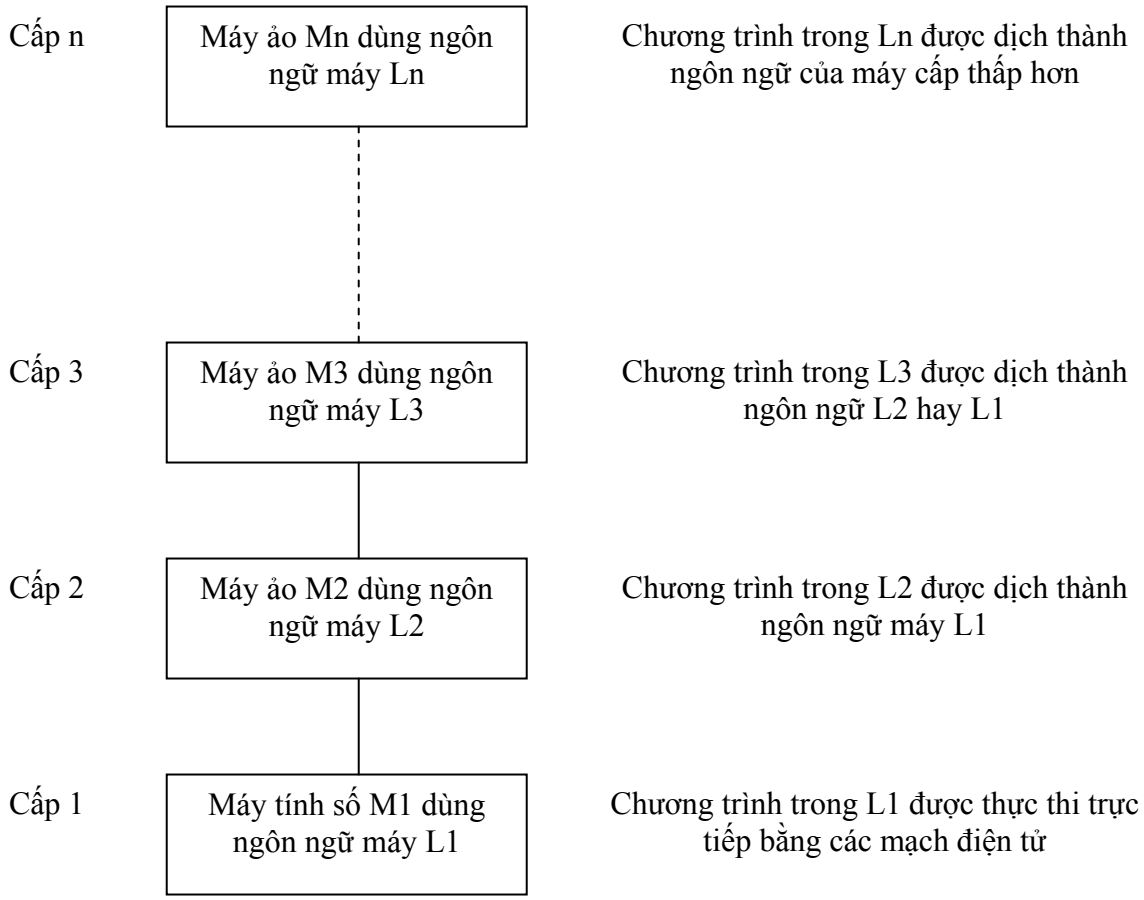
Máy tính số (Digital computer) là máy giải quyết các vấn đề bằng cách thực hiện các chỉ thị do con người cung cấp. Chuỗi các chỉ thị này gọi là chương trình (program). Các mạch điện tử trong một máy tính số sẽ thực hiện một số giới hạn các chỉ thị đơn giản cho trước. Tập hợp các chỉ thị này gọi là tập lệnh của máy tính. Tất cả các chương trình muốn thực thi đều phải được biến đổi sang tập lệnh trước khi được thi hành. Các lệnh cơ bản là:

- Cộng 2 số.
- So sánh với 0.
- Di chuyển dữ liệu.

Tập lệnh của máy tính tạo thành một ngôn ngữ giúp con người có thể tác động lên máy tính, ngôn ngữ này gọi là ngôn ngữ máy (machine language). Tuy nhiên, hầu hết các ngôn ngữ máy đều đơn giản nên để thực hiện một yêu cầu nào đó, người thiết kế phải thực hiện một công việc phức tạp. Đó là chuyển các yêu cầu này thành các chỉ thị có chứa trong tập lệnh của máy. Vấn đề này có thể giải quyết bằng cách thiết kế một tập lệnh mới thích hợp cho con người hơn tập lệnh đã cài đặt sẵn trong máy (built-in). Ngôn ngữ máy sẽ được gọi là ngôn ngữ cấp 1 (L1) và ngôn ngữ vừa được hình thành gọi là ngôn ngữ cấp 2 (L2).

Tuy nhiên, trong thực tế, để có thể thực hiện được, các ngôn ngữ L1 và L2 không được khác nhau nhiều. Như vậy, ngôn ngữ L2 cũng không thật sự giúp ích nhiều cho người thiết kế. Do đó, một tập lệnh kế tiếp được hình thành sẽ hướng về con người nhiều hơn là máy tính, tập lệnh này sẽ tạo thành một ngôn ngữ và ta gọi là ngôn ngữ L3. Ta có thể viết các chương trình trong L3 như là đã tồn tại máy tính sử dụng ngôn ngữ L3 (máy ảo L3). Các chương trình này sẽ được dịch sang ngôn ngữ L2 và được thực thi bằng một chương trình dịch L2.

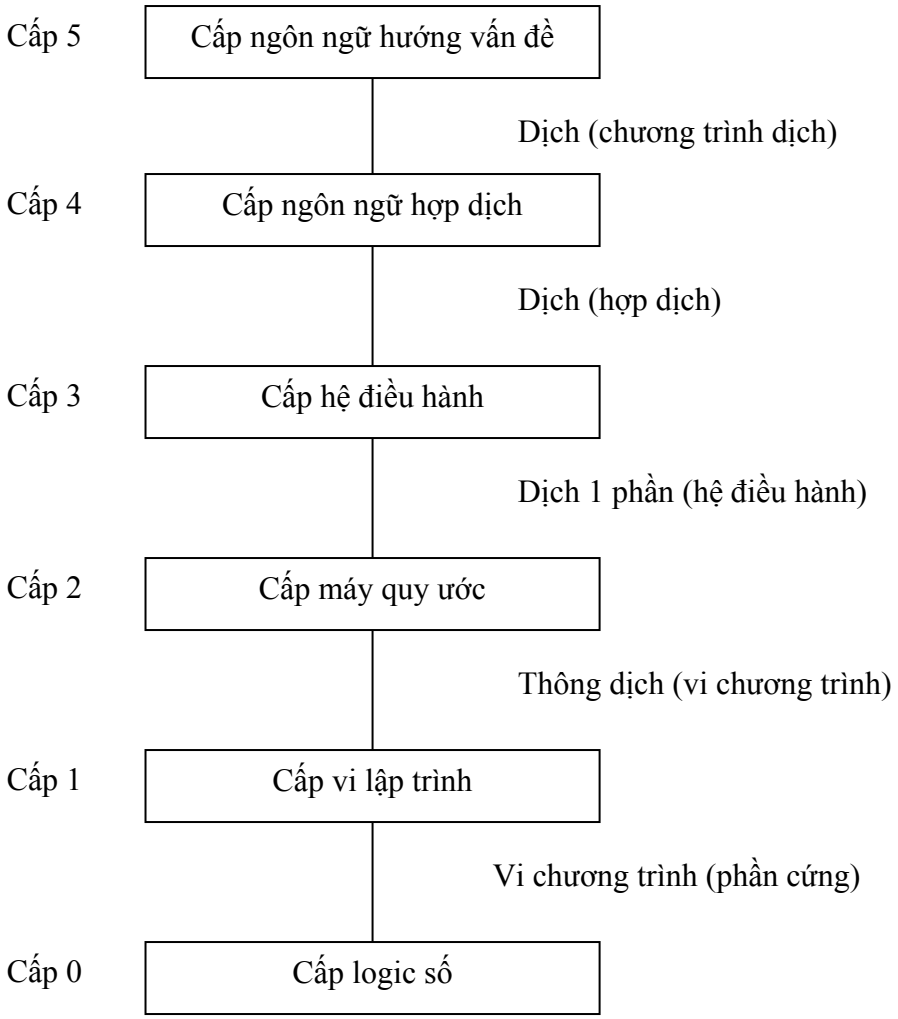
Việc xây dựng toàn bộ chuỗi các ngôn ngữ, mỗi ngôn ngữ được tạo ra sẽ thích hợp hơn ngôn ngữ trước đó sẽ có thể tiếp tục cho đến khi nhận được ngôn ngữ thích hợp nhất. Sơ đồ một máy ảo n cấp có thể biểu diễn như sau:



Hình 1.1. Máy ảo n cấp

Một máy tính số có n cấp có thể xem như có n-1 máy ảo khác nhau, mỗi máy ảo có một ngôn ngữ máy riêng. Các chương trình viết trên các máy ảo này không thể thực thi trực tiếp mà phải dịch thành các ngôn ngữ máy cấp thấp hơn. Chỉ có máy thật dùng ngôn ngữ máy L1 mới có thể thực thi trực tiếp bằng các mạch điện tử. Một lập trình viên sử dụng máy ảo cấp n không cần biết tất cả các trình dịch này. Chương trình trong máy ảo cấp n sẽ được thực thi bằng cách dịch thành ngôn ngữ máy cấp thấp hơn và ngôn ngữ máy này sẽ được dịch thành ngôn ngữ máy thấp hơn nữa hay dịch trực tiếp thành ngôn ngữ máy L1 và thực thi trực tiếp trên các mạch điện tử.

Về cơ bản, máy tính gồm có 6 cấp:



Hình 1.2 – Các cấp trên máy tính số

Cấp 0 chính là phần cứng của máy tính. Các mạch điện tử của cấp này sẽ thực thi các chương trình ngôn ngữ máy của cấp 1. Trong cấp logic số, đối tượng quan tâm là các cổng logic. Các cổng này được xây dựng từ một nhóm các transistor.

Cấp 1 là cấp ngôn ngữ máy thật sự. Cấp này có một chương trình gọi là vi chương trình (microprogram), vi chương trình có nhiệm vụ thông dịch các chỉ thị của cấp 2. Hầu hết các lệnh trong cấp này là di chuyển dữ liệu từ phần này đến phần khác của máy hay thực hiện việc một số kiểm tra đơn giản.

Mỗi máy cấp 1 có một hay nhiều vi chương trình chạy trên chúng. Mỗi vi chương trình xác định một ngôn ngữ cấp 2. Các máy cấp 2 đều có nhiều điểm chung ngay cả các máy cấp 2 của các hãng sản xuất khác nhau. Các lệnh trên máy cấp 2 được thực thi bằng cách thông dịch bởi vi chương trình mà không phải thực thi trực tiếp bằng phần cứng.

Cấp thứ 3 thường là cấp hỗn hợp. Hầu hết các lệnh trong ngôn ngữ của cấp máy này cũng có trong ngôn ngữ cấp 2 và đồng thời có thêm một tập lệnh mới, một tổ chức bộ

nhớ khác và khả năng chạy 2 hay nhiều chương trình song song. Các lệnh mới thêm vào sẽ được thực thi bằng một trình thông dịch chạy trên cấp 2, gọi là hệ điều hành. Nhiều lệnh cấp 3 được thực thi trực tiếp do vi chương trình và một số lệnh khác được thông dịch bằng hệ điều hành (do đó, cấp này là cấp hỗn hợp).

Cấp 4 thật sự là dạng tượng trưng cho một trong các ngôn ngữ. Cấp này cung cấp một phương pháp viết chương trình cho các cấp 1, 2, 3 dễ dàng hơn. Các chương trình viết bằng hợp ngữ được dịch sang các ngôn ngữ của cấp 1, 2, 3 và sau đó được thông dịch bằng các máy ảo hay thực tương ứng.

Cấp 5 bao gồm các ngôn ngữ được thiết kế cho người lập trình nhằm giải quyết một vấn đề cụ thể. Các ngôn ngữ này được gọi là cấp cao. Một số ngôn ngữ cấp cao như Basic, C, Cobol, Fortran, Lisp, Prolog, Pascal và các ngôn ngữ lập trình hướng đối tượng như C++, J++, … Các chương trình viết bằng các ngôn ngữ này thường được dịch sang cấp 3 hay 4 bằng các trình biên dịch (compiler).

2. Giao tiếp ngoại vi

Ta phân biệt tất cả 3 phương pháp xuất / nhập dữ liệu:

- Nhập / xuất bằng cách hỏi trạng thái của thiết bị ngoại vi (polling)
- Nhập / xuất bằng ngắt (interrupt).
- Nhập / xuất bằng cách truy xuất trực tiếp vào bộ nhớ dùng các phần cứng phụ trợ (DMA).

2.1. Nhập / xuất dữ liệu bằng cách hỏi vòng (polling)

Ta biết rằng vấn đề điều khiển nhập / xuất dữ liệu sẽ rất đơn giản trong trường hợp thiết bị ngoại vi lúc nào cũng có thể làm việc với µP. Ta có thể ví dụ như bộ hiển thị Led 7 đoạn lúc nào cũng sẵn sàng hiển thị dữ liệu khi mà µP gởi dữ liệu ra. Tuy nhiên, trong thực tế, không phải lúc nào µP cũng làm việc với các thiết bị ngoại vi có tính năng như trên. Ví dụ như khi làm việc với một máy in, µP yêu cầu in nhưng máy in không sẵn sàng (giả sử như hết giấy, kẹt giấy, …). Khi đó, µP phải kiểm tra xem một thiết bị mà nó cần giao tiếp có sẵn sàng hay không nếu thiết bị sẵn sàng thì mới thực hiện trao đổi dữ liệu. Để kiểm tra các thiết bị ngoại vi, µP phải sử dụng các tín hiệu bắt tay (handshake) xác định tuần tự từng thiết bị, xem thiết bị nào có yêu cầu trao đổi dữ liệu. Các tín hiệu này lấy từ các mạch giao tiếp do người thiết kế tạo ra.

Giả sử hệ thống có 2 thiết bị ngoại vi, nếu thiết bị 1 có dữ liệu cần truyền đến µP thì nó sẽ gởi 1 xung để chốt dữ liệu đồng thời tạo tín hiệu sẵn sàng cho thiết bị. Khi µP kiểm tra tín hiệu sẵn sàng của thiết bị 1 thì nó sẽ đọc dữ liệu vào từ mạch chốt và xoá tín hiệu sẵn sàng.

Trong trường hợp µP muốn gởi dữ liệu ra thiết bị 2, nó sẽ đọc tín hiệu sẵn sàng của thiết bị 2, nếu thiết bị 2 có thể nhận dữ liệu thì µP sẽ gởi dữ liệu ra mạch chốt và thiết bị 2 sẽ đọc dữ liệu vào.

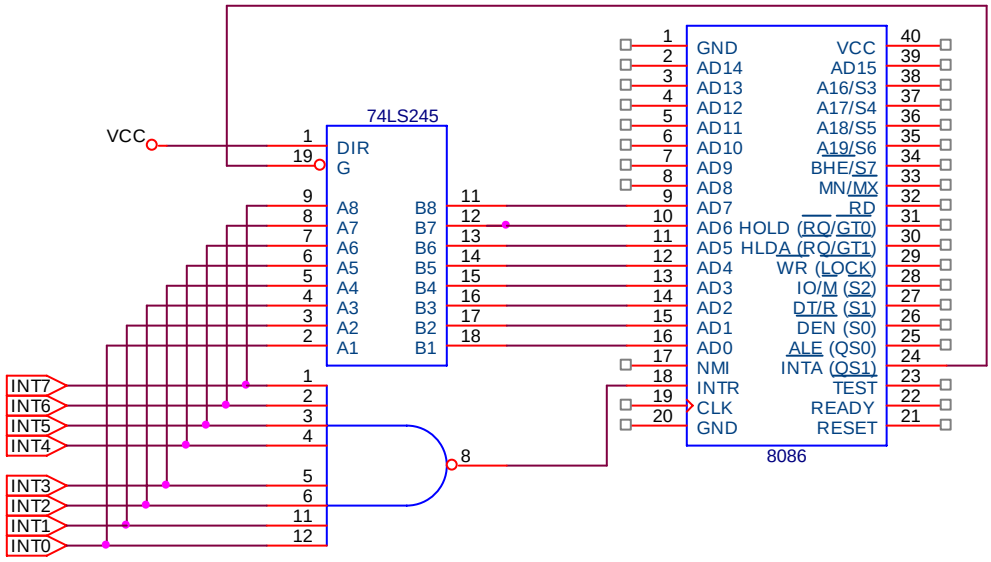
2.2. Ngắt và xử lý ngắt

Trong cách thức thực hiện trao đổi dữ liệu bằng cách hỏi vòng như trên, trước khi tiến hành thực hiện thì µP phải kiểm tra trạng thái sẵn sàng của thiết bị ngoại vi. Tuy nhiên trong thực tế ta cần phải tận dụng khả năng của µP để làm các công việc khác mà không phải tốn thời gian kiểm tra thiết bị, chỉ khi nào có yêu cầu trao đổi dữ liệu thì mới tạm dừng công việc hiện tại. Cách làm việc như vậy gọi là ngắt µP, khi có một ngắt xảy ra thì ta phải thực hiện gọi các chương trình phục vụ ngắt tại các địa chỉ xác định của µP. Các tín hiệu ngắt từ thiết bị ngoại vi đưa vào µP thông qua các chân NMI hay INTR.

2.2.1. Các loại ngắt

❖ **Ngắt cứng**: là các yêu cầu ngắt từ các chân NMI hay INTR.

Ngắt cứng NMI là ngắt không che được còn ngắt cứng INTR có thể che được. Các lệnh CLI (Clear Interrupt) và STI (Set Interrupt) chỉ ảnh hưởng đến việc µP có chấp nhận yêu cầu ngắt tại chân INTR hay không. Yêu cầu ngắt tại chân INTR có thể có các kiểu ngắt từ 00h ÷ FFh. Kiểu ngắt này sẽ được đưa vào bus dữ liệu để µP xác định kiểu ngắt (dùng cho các thiết bị ngoại vi khác nhau).



Hình 1.3 – Kết nối ngắt đơn giản

❖ **Ngắt mềm**: là các ngắt thực hiện bằng phần mềm tác động do người sử dụng.

2.2.2. Đáp ứng của µP khi có yêu cầu ngắt

Khi có yêu cầu ngắt đến µP và nếu được phép ngắt, µP sẽ thực hiện các công việc sau:

- [SP] ← SP – 2, [SP] ← FR (Flag Register): cất thanh ghi cờ vào stack.
- IF ← 0, TF ← 0: không cho thực hiện các ngắt khác.
- SP ← SP – 2, [SP] ← CS: cất địa chỉ đoạn mã vào stack.
- SP ← SP – 2, [SP] ← IP: cất địa chỉ trở về sau khi phục vụ ngắt

- IP ← [Số_hiệu_ngắt*4], CS ← [Số_hiệu_ngắt*4 + 2]: lấy lệnh tại địa chỉ phục vụ ngắt tương ứng
- Sau khi kết thúc chương trình con phục vụ ngắt (khi gặp lệnh IRET):
  + IP ← [SP], SP ← SP + 2

  + CS ← [SP], SP ← SP + 2: lấy lại địa chỉ trước khi gọi chương trình phục vụ ngắt

  + FR ← [SP], SP ← SP + 2: lấy lại giá trị thanh ghi cờ

2.2.3. Xử lý ưu tiên ngắt

Như ta đã biết ở trên, khi μP đang thực hiện lệnh, nếu có ngắt xảy ra thì μP sẽ tạm ngừng chương trình và thực thi chương trình con phục vụ ngắt. Trong thực tế sẽ có trường hợp có nhiều yêu cầu ngắt khác nhau cùng một lúc, khi đó μP sẽ phục vụ cho ngắt theo thứ tự ưu tiên với nguyên tắc là ngắt nào có mức ưu tiên cao nhất thì sẽ phục vụ cho ngắt đó trước.

Các mức ưu tiên của các ngắt (từ mức thấp nhất đến mức cao nhất):

- Ngắt thực hiện chạy từng lệnh (INT 1)
- Ngắt che được INTR
- Ngắt không che được NMI
- Ngắt nội bộ (INT 0: xảy ra do phép chia số 0, ngắt mềm)

2.3. Nhập / xuất dữ liệu bằng DMA (Direct Memory Access)

Trong các phương thức trao đổi dữ liệu như hai phần trên đã trình bày thì việc trao đổi dữ liệu giữa thiết bị ngoại vi và hệ thống thường theo trình tự sau: ***từ ngoại vi đến vi xử lý rồi đi vào bộ nhớ*** **hay** ***từ bộ nhớ đến vi xử lý rồi ghi ra ngoại vi***. Trong thực tế sẽ có trường hợp ta cần thực hiện trao đổi dữ liệu ngay giữa ngoại vi và bộ nhớ. Khi đó người ta đưa ra cơ chế truy xuất bộ nhớ trực tiếp (DMA). Để thực hiện được vấn đề này, các hệ vi xử lý thông thường dùng thêm các mạch chuyên dụng để điều khiển quá trình truy xuất bộ nhớ trực tiếp (DMAC – Direct Memory Access Controller). Có tất cả 3 cơ chế hoạt động:

- ***Tận dụng thời gian CPU không dùng bus:***

Ta phải dùng thêm mạch phát hiện các chu kỳ xử lý nội của CPU và tận dụng các chu kỳ này để thực hiện trao đổi dữ liệu.

- ***Treo CPU để trao đổi từng byte:***

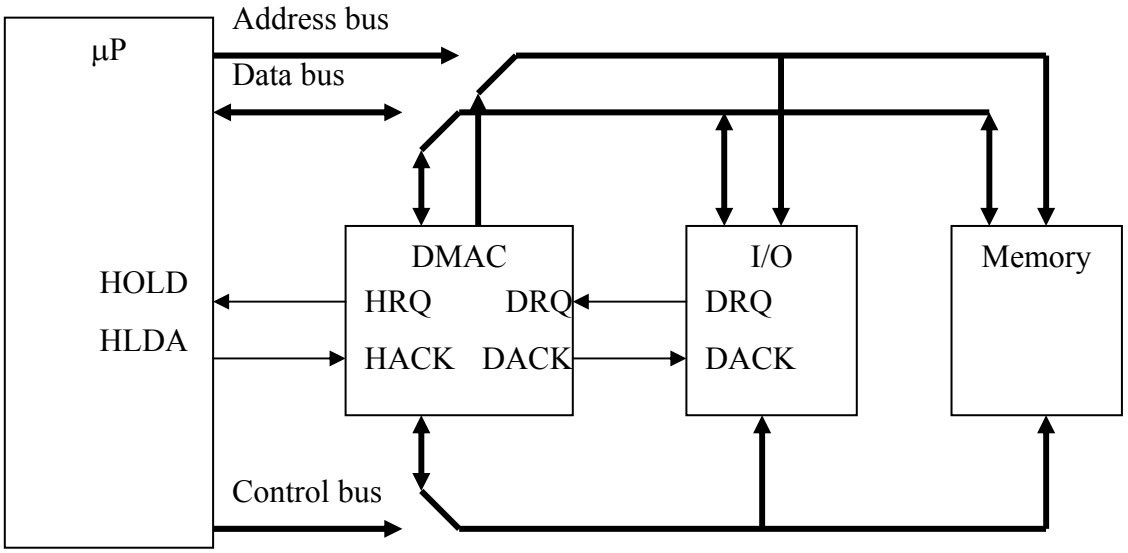
CPU không bị treo trong khoảng thời gian dài mà chỉ bị treo trong thời gian ngắn đủ để trao đổi 1 byte dữ liệu giữa bộ nhớ và ngoại vi. Do đó, công việc của CPU không bị gián đoạn mà chỉ bị chậm đi.

- ***Treo CPU một khoảng thời gian để trao đổi một khối dữ liệu:***

Trong cơ chế này, CPU bị treo trong suốt quá trình trao đổi dữ liệu.

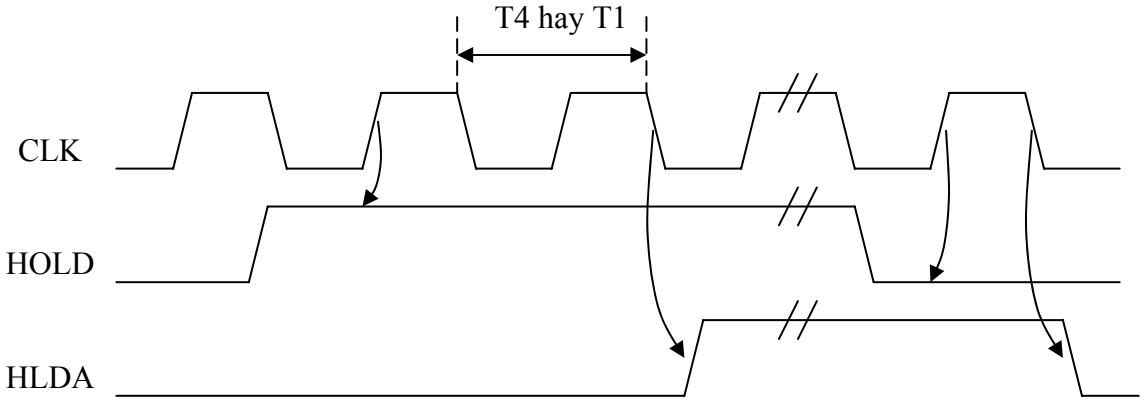
- CPU ghi từ lệnh và từ chế độ làm việc vào DMAC.
- Khi thiết bị ngoại vi có yêu cầu trao đổi dữ liệu, nó gởi tín hiệu DRQ = 1 (DMA Request) đến DMAC.

- DMAC gởi tín hiệu HRQ (Hold Request) đến chân HOLD của CPU để yêu cầu treo CPU. Tín hiệu này sẽ giữ ở mức cao cho đến hết quá trình trao đổi dữ liệu.
- Sau khi nhận yêu cầu treo, CPU sẽ thực hiện hết chu kỳ bus của m?nh rồi treo các bus và gởi tín hiệu HLDA (Hold Acknowledge) để báo cho DMAC biết có thể sử dụng các bus.
- DMAC chuyển dữ liệu từ bộ nhớ đến ngoại vi bằng cách: đưa địa chỉ byte đầu tiên ra bus địa chỉ và đưa tín hiệu $\overline{\text{MEMR}}$ để đọc 1 byte từ bộ nhớ, kế tiếp DMAC đưa tín hiệu $\overline{\text{IOW}}$ để ghi dữ liệu ra ngoại vi. Sau đó, DMAC giảm số byte cần truyền, cập nhật địa chỉ bộ nhớ và lặp lại quá trình cho đến khi hết byte cần truyền.



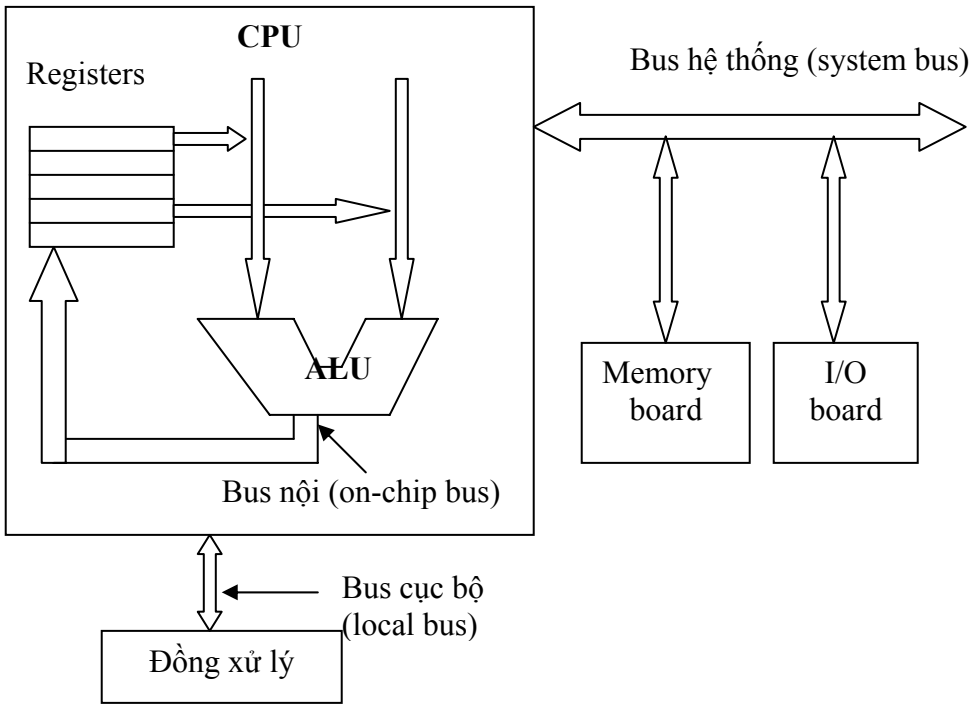
Hình 1.4 – Giao tiếp DMAC với hệ vi xử lý

Hai tín hiệu dùng để yêu cầu treo và chấp nhận yêu cầu treo CPU dùng cho cơ chế DMA là HOLD và HLDA có thể mô tả như sau:



Hình 1.5 – Tín hiệu HOLD và HLDA

3. Bus



Hình 1.6 - Các bus trong một hệ thống máy tính

Bus là đường truyền tín hiệu điện nối các thiết bị khác nhau trong một hệ thống máy tính. Bus thường có từ 50 đến 100 dây dẫn được gắn trên mainboard, trên các dây này có các đầu nối đưa ra, các đầu này được sắp xếp và cách nhau những khoảng quy định để có thể cắm vào đó những I/O board hay board bộ nhớ (bus hệ thống – system bus).

Cũng có những bus dùng cho mục đích chuyên biệt, thí dụ nối 1 vi xử lý với 1 hay nhiều vi xử lý khác hoặc nối với bộ nhớ cục bộ (local bus).

Trong vi xử lý cũng có một số bus để nối các thành phần bên trong của bộ vi xử lý với nhau. Người thiết kế chip vi xử lý có thể tuỳ ý lựa chọn loại bus bên trong nó, còn với các bus liên hệ bên ngoài cần phải xác định rõ các quy tắc làm việc cũng như các đặc điểm kỹ thuật về điện và cơ khí của bus để người thiết kế mainboard có thể ghép nối chip vi xử lý với các thiết bị khác. Nói cách khác, các bus này phải tuân theo 1 chuẩn nào đó. Tập các quy tắc của chuẩn còn được gọi là giao thức bus (bus protocol)

Thường có nhiều thiết bị nối với bus, một số thiết bị là tích cực (active) có thể đòi hỏi truyền thông trên bus, trong khi đó có các thiết bị thụ động chờ yêu cầu từ các thiết bị khác. Các thiết bị tích cực được gọi là chủ (master) còn thiết bị thụ động là tớ (slave).

Ví dụ: Khi CPU ra lệnh cho bộ điều khiển đĩa đọc/ghi một khối dữ liệu thì CPU là master còn bộ điều khiển đĩa là slave. Tuy nhiên, bộ điều khiển đĩa ra lệnh cho bộ nhớ nhận dữ liệu thì nó lại giữ vai trò master.

3.1. Bus Driver và Bus Receiver

Tín hiệu điện trong máy tính phát ra thường không đủ để điều khiển bus, nhất là khi bus khá dài và có nhiều thiết bị nối với nó. Chính vì thế mà hầu hết các bus master được nối với bus thông qua 1 chip gọi là ***bus driver***, về cơ bản nó là một bộ khuếch đại tín hiệu số. Tương tự như vậy, hầu hết các slave được nối với bus thông qua ***bus receiver***. Đối với các thiết bị khi thì đóng vai trò master, khi thì đóng vai trò slave, người ta sử dụng 1 chip kết hợp gọi là ***transceiver***. Các chip này đóng vai trò ghép nối và là các thiết bị 3 trạng thái, cho phép nó có thể ở trạng thái thứ 3 – hở mạch (thả nổi).

Giống như vi xử lý, bus có các đường địa chỉ, đường số liệu và đường điều khiển. Tuy nhiên, không nhất thiết có ánh xạ 1 – 1 giữa các tín hiệu ở các chân ra của vi xử lý và các đường dây của bus. Thí dụ: một số chíp vi xử lý có 3 chân ra, truyền ra các tín hiệu báo chíp vi xử lý đang thực hiện các thao tác $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, $\overline{\text{IOR}}$, $\overline{\text{IOW}}$ hay thao tác khác. Một bus điển hình thường có 4 đường trên.

Các vấn đề quan trọng nhất liên quan đến thiết kế bus là: xung clock bus (sự phân chia thời gian, hay còn gọi là **bus blocking**), cơ chế phân xử bus (bus arbitration), xử lý ngắt và xử lý lỗi.

Các bus có thể được chia theo giao thức truyền thông thành hai loại riêng biệt là bus đồng bộ và bus không đồng bộ phụ thuộc vào việc sử dụng clock bus.

3.2. Bus đồng bộ (Synchronous bus)

Mỗi chu kỳ bus bắt đầu bằng việc xuất địa chỉ bộ nhớ hoặc I/O port (chu kỳ xung nhịp T1). Bus điều khiển có 4 tín hiệu tác động mức thấp là $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, $\overline{\text{IOR}}$ và $\overline{\text{IOW}}$.

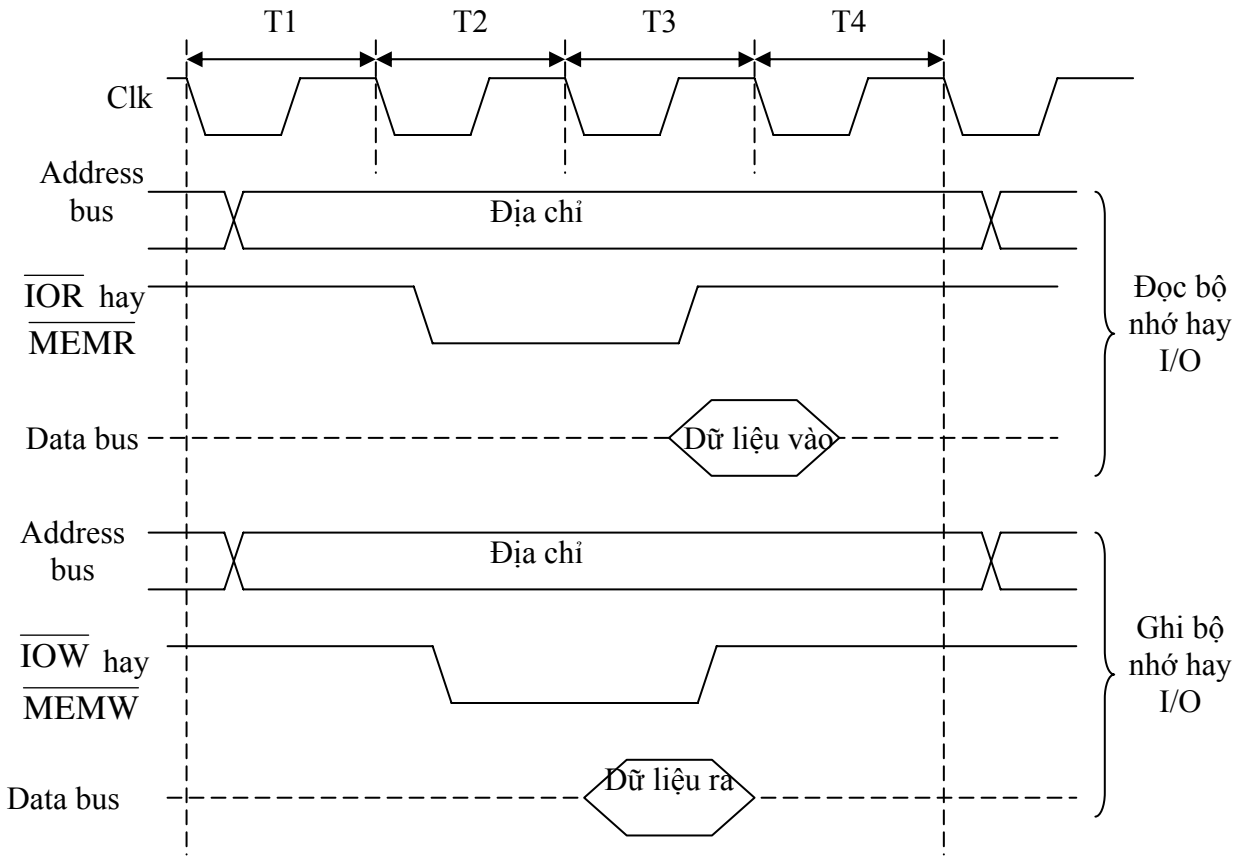
Các chuỗi sự kiện xảy ra trong một chu kỳ bus đọc bộ nhớ:

T1: µP xuất địa chỉ bộ nhớ 20 bit. Các đường dữ liệu không hoạt động và các đường điều khiển bị cấm

T2: Đường điều khiển $\overline{\text{MEMR}}$ xuống mức thấp. Đơn vị bộ nhớ ghi nhận chu kỳ bus này là quá trình đọc bộ nhớ và đặt byte hay word có địa chỉ đó lên data bus.

T3: µP đặt cấu hình để các đường data bus là nhập. Trạng thái này chủ yếu để bộ nhớ có thời gian tìm kiếm byte hay word dữ liệu

T4: µP đợi dữ liệu trên data bus. Do đó, nó thực hiện chốt data bus và giải phóng các đường điều khiển đọc bộ nhớ. Quá trình này sẽ kết thúc chu kỳ bus.



Hình 1.7 – Định thì chu kỳ bus đồng bộ

Trong một chu kỳ bus, µP có thể thực hiện đọc I/O, ghi I/O, đọc bộ nhớ hay ghi bộ nhớ. Các đường address bus và control bus dùng để xác định địa chỉ bộ nhớ hay I/O và hướng truyền dữ liệu trên data bus.

Chú ý rằng µP điều khiển tất cả các quá trình trên nên bộ nhớ bắt buộc phải cung cấp được dữ liệu vào lúc $\overline{MEMR}$ lên mức cao trong trạng thái T4. Nếu không, µP sẽ đọc dữ liệu ngẫu nhiên không mong muốn trên data bus. Để giải quyết vấn đề này, ta có thể dùng thêm các trạng thái chờ (wait state).

❖ **Truyền theo khối:**

Ngoài các chu kỳ đọc/ghi, một số bus truyền dữ liệu đồng bộ còn hỗ trợ truyền dữ liệu theo khối. Khi bắt đầu thao tác đọc khối, bus master báo cho slave biết số byte cần được truyền đi, thí dụ truyền con số này đi trong chu kỳ T1, sau đó đáng lẽ truyền đi 1 byte, slave đưa ra trong mỗi chu kỳ 1 byte cho tới khi đủ số byte được thông báo. Như vậy, khi đọc dữ liệu theo khối, n byte dữ liệu cần n+2 chu kỳ clock chứ không phải 3n chu kỳ.

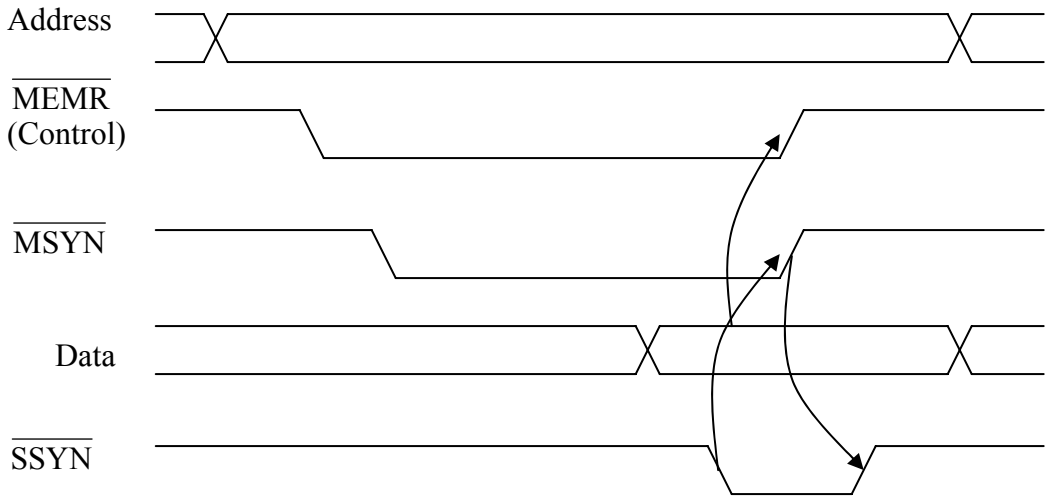
Một cách khác để cho truyền dữ liệu nhanh hơn là giảm chu kỳ. Tuy nhiên, giảm chu kỳ bus dẫn đến khó khăn về mặt kỹ thuật, các tín hiệu truyền trên các đường khác nhau không phải luôn có cùng tốc độ, dẫn đến hiệu ứng *bus skew*. Điều quan trọng là thời

gian chu kỳ phải dài hơn so với skew để tránh việc những khoảng thời gian được số hoá lại trở thành các đại lượng biến thiên liên tục.

3.3. Bus bất đồng bộ(Asynchronous bus)

Bus bất đồng bộ không sử dụng xung clock đồng bộ, chu kỳ của nó có thể kéo dài tuỳ ý và có thể khác nhau đối với các cặp thiết bị khác nhau. Làm việc với các bus đồng bộ dễ dàng hơn do nó được định thời một cách gián đoạn , tuy vậy chính đặc điểm này cũng dẫn đên nhược điểm. Mọi công việc được tiến hành trong khoảng thời gian là bội số của xung clock, nếu 1 thao tác nào đó của vi xử lý hay bộ nhớ hoàn thành trong 3.1 chu kỳ thì nó cũng sẽ phải kéo dài trong 4 chu kỳ. Khi đã chọn chu kỳ bus và đã xây dựng bộ nhớ, I/O card cho bus này thì khó có thể tận dụng những tiến bộ của công nghệ. Chẳng hạn sau khi đã xây bus với sự định thời như trên, công nghệ mới đưa ra các vi xử lý và bộ nhớ có thời gian chu kỳ là 100ns chứ không còn là 750ns như cũ, thì chúng vẫn chạy với tốc độ thấp như các vi xử lý, bộ nhớ loại cũ, bởi vì giao thức bus đòi hỏi bộ nhớ phải đưa được dữ liệu ra và ổn định trước thời điểm cạnh âm của T3. Nếu có nhiều thiết bị khác nhau cùng nối với 1 bus, trong đó có thể có một số thiết bị hoạt động nhanh hơn hơn các thiết bị khác thì cần phải đặt bus hoạt động phù hợp với thiết bị có tốc độ thấp nhất.

Bus bất đồng bộ ra đời nhằm khắc phục những nhược điểm của bus đồng bộ. Trước hết master phát ra địa chỉ nhớ mà nó muốn truy cập, sau đó phát tín hiệu $\overline{\text{MEMR}}$ tích cực để xác định cần truy xuất bộ nhớ và yêu cầu quá trình truy xuất là READ để xác định chiều truyền dữ liệu. Tín hiệu $\overline{\text{MEMR}}$ được đưa ra sau tín hiệu địa chỉ một khoảng thời gian phụ thuộc tốc độ hoạt động của master. Sau khi 2 tín hiệu này đã ổn định, master sẽ phát ra tín hiệu $\overline{\text{MSYN}}$ (master synchrization) ở mức tích cực để báo cho slave biết rằng các tín hiệu cần thiết đã sẵn sàng trên bus, slave có thể nhận lấy. Khi slave nhận được tín hiệu này, nó sẽ thực hiện công việc với tốc độ nhanh nhất có thể được, đưa dữ liệu của ô nhớ được yêu cầu lên bus dữ liệu. Khi hoàn thành slave sẽ phát tín hiệu $\overline{\text{SSYN}}$ (slave synchronization) tích cực.



Hình 1.8 – Định thì chu kỳ bus bất đồng bộ

Master nhận được tín hiệu $\overline{SSYN}$ tích cực thì xác định được dữ liệu của slave đã sẵn sàng nên thực hiện việc chốt dữ liệu, sau đó đảo các đường địa chỉ cũng như các tín hiệu $\overline{MEMR}$ và $\overline{MSYN}$. Khi slave nhận được tín hiệu $\overline{MSYN}$ không tích cực, nó xác định kết thúc chu kỳ và đảo tín hiệu $\overline{SSYN}$ làm bus trở lại trạng thái ban đầu, mọi tín hiệu đều không tích cực, chờ bus master mới.

Trên giản đồ thời gian của bus bất đồng bộ, ta sử dụng mũi tên để thể hiện nguyên nhân và kết quả. $\overline{MSYN}$ tích cực dẫn đến việc truyền dữ liệu ra bus dữ liệu và đồng thời cũng dẫn đến việc slave phát ra tín hiệu $\overline{SSYN}$ tích cực, đến lượt mình tín hiệu $\overline{SSYN}$ lại gây ra sự đảo mức của các đường địa chỉ, $\overline{MEMR}$ và $\overline{MSYN}$. Cuối cùng sự đảo mức của $\overline{MSYN}$ lại gây ra sự đảo mức tín hiệu $\overline{SSYN}$ và kết thúc chu kỳ. Tập các tín hiệu phối hợp với nhau như vậy được gọi là bắt tay toàn phần (full handshake), chủ yếu gồm 4 tín hiệu sau:

- $\overline{MSYN}$ tích cực.
- $\overline{SSYN}$ tích cực để đáp lại tín hiệu $\overline{MSYN}$.
- $\overline{MSYN}$ được đảo để đáp lại tín hiệu $\overline{SSYN}$ (tích cực).
- $\overline{SSYN}$ được đảo để đáp lại tín hiệu $\overline{MSYN}$ không tích cực.

Ta có thể nhận thấy bắt tay toàn phần là độc lập thời gian, mỗi sự kiện được gây ra bởi 1 sự kiện trước đó chứ không phải bởi xung clock. Nếu 1 cặp master-slave nào đó hoạt động chậm thì cặp master-slave kế tiếp không hề bị ảnh hưởng.

Tuy ưu điểm của bus bất đồng bộ rất rõ ràng, nhưng trong thực tế phần lớn các bus đang sử dụng là loại đồng bộ. Nguyên nhân là các hệ thống sử dụng bus đồng bộ dễ thiết kế hơn. Vi xử lý chỉ cần chuyển các mức tín hiệu cần thiết sang trạng thái tích cực là bộ nhớ đáp ứng ngay, không cần tín hiệu phản hồi. Chỉ cần các chọn phù hợp thì mọi hoạt động đều trôi chảy, không cần phải bắt tay.

3.4. Xử lý ngắt

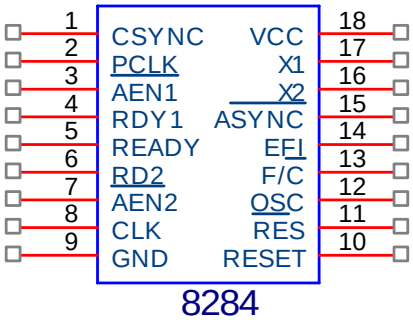
Ở trên, ta chỉ khảo sát các chu kỳ bus thông thường, trong đó master nhận hay gởi thông tin từ / đến slave. Một ứng dụng quan trọng nữa của bus là dùng để xử lý ngắt. Khi CPU ra lệnh cho thiết bị I/O làm một việc gì đó, nó thường chờ đợi tín hiệu ngắt do thiết bị I/O phát ra khi hoàn thành công việc được CPU yêu cầu. Khi nhận được tín hiệu ngắt, CPU sẽ đáp ứng ngay, có thể nhận dữ liệu do thiết bị I/O truyền về, hay gởi tiếp dữ liệu ra thiết bị I/O, hay CPU sẽ sử dụng bus cho một thao tác khác…. Như vậy chính ngắt phát ra tín hiệu yêu cầu sử dụng bus.

Vì có thể nhiều thiết bị ngoại vi cùng phát ra ngắt, cho nên cần có 1 cơ chế phân xử giống như đối với các bus thông thường. Giải pháp thường dùng là gán các mức độ ưu tiên cho các thiết bị và sử dụng 1 arbiter tập trung để trao quyền ưu tiên cho các thiết bị quan trọng thường xuyên được sử dụng. Hiện trên thị trường có những chip điều khiển ngắt được tiêu chuẩn hóa và được sử dụng rộng rãi là chip 8259A. Có thể nối 8 chip điều khiển I/O tới các đầu IRx (Interrupt request) của 8259A. Khi có 1 thiết bị nào đó muốn ngắt, nó đặt mức tích cực lên chân Irx, 8259A nhận được tín hiệu tích cực ở 1 hay một số

đầu vào Irx thì sẽ đặt mức tích cực lên đầu dây INT. Tín hiệu INT sẽ truyền trực tiếp đến chân Interrupt của CPU. Khi CPU có thể xử lý được ngắt, nó gởi lại 1 tín hiệu chấp nhận ngắt cho 8259A. Lúc này, CPU chờ 8259A chỉ ra I/O nào yêu cầu ngắt, bằng cách gởi số hiệu của I/O đó lên bus dữ liệu (D0-D7) để đi đến CPU. Sau đó, phần cứng CPU sẽ sử dụng con số đó để tính chỉ số trong 1 bảng con trỏ -bảng vector ngắt (interrupt vector) để tìm địa chỉ chương trình con, cho chạy chương trình này để phục vụ ngắt. Các chương trình con này gọi là chương trình con xử lý ngắt.

4. Các chip hỗ trợ cho bộ xử lý trung tâm

4.1. Mạch tạo xung clock 8284



Hình 1.9 – Mạch tạo xung clock 8284

PCLK (Peripheral Clock): xung clock f = $f_X/6$ (f_X là tần số thạch anh) với chu kỳ bổn phận 50%.

CSYNC (Clock Synchronisation): ngõ vào xung đồng bộ chung khi hệ thống có các 8284 dùng dao động ngoài tại chân EFI. Khi dùng mạch dao động trong thì phải nối GND.

$\overline{AEN1}$, $\overline{AEN2}$ (Address Enable): cho phép chọn các chân tương ứng RDY1, RDY2 báo hiệu trạng thái sẵn sàng của bộ nhớ hay thiết bị ngoại vi.

RDY1, RDY2 (Bus ready): kết hợp với $\overline{AEN1}$, $\overline{AEN2}$ tạo các chu kỳ đợi ở CPU

READY: nối đến chân READY của µP.

CLK (Clock): xung clock f = $f_X/3$, nối với chân CLK của CPU.

RESET: nối với chân RESET của CPU, là tín hiệu khởi động lại toàn hệ thống.

$\overline{RES}$ (Reset Input): chân khởi động cho 8284, được nối với mạch RC để tự khởi động khi bật nguồn.

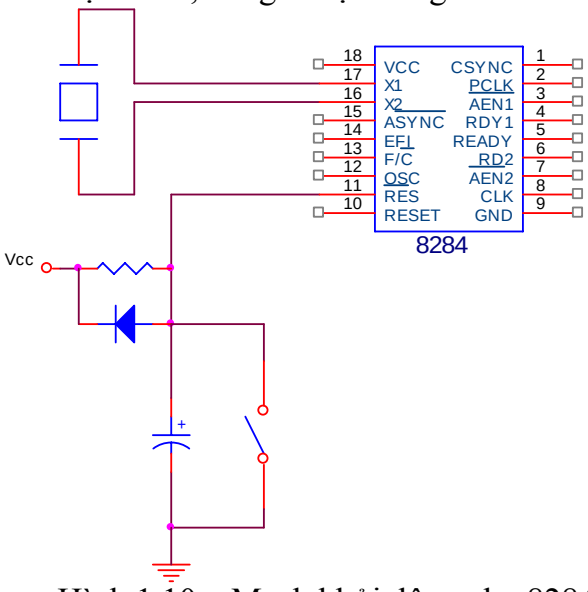
OSC: ngõ ra xung clock có tần số f_X.

$F/\overline{C}$ (Frequency / Crystal): chọn nguồn tín hiệu chuẩn cho 8284, nếu ở mức cao thì chọn tần số xung clock bên ngoài, ngược lại thì dùng xung clock từ thạch anh.

EFI (External Frequency Input): xung clock từ bộ dao động ngoài.

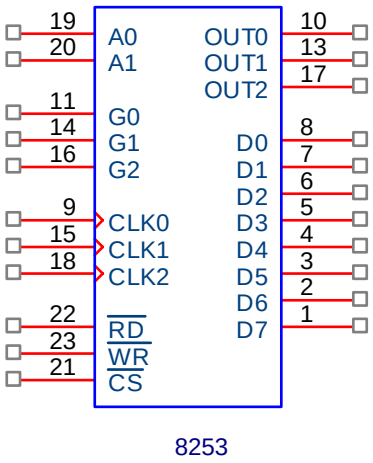
$\overline{ASYNC}$: chọn chế độ làm việc cho tín hiệu RDY. Nếu $\overline{ASYNC}$ = 1, tín hiệu RDY có ảnh hưởng đến tín hiệu READY cho đến khi có xung âm của xung clock. Ngược lại thì RDY chỉ ảnh hưởng khi xuất hiện xung âm.

X1,X2: ngõ vào của thạch anh, dùng để tạo xung chuẩn cho hệ thống.

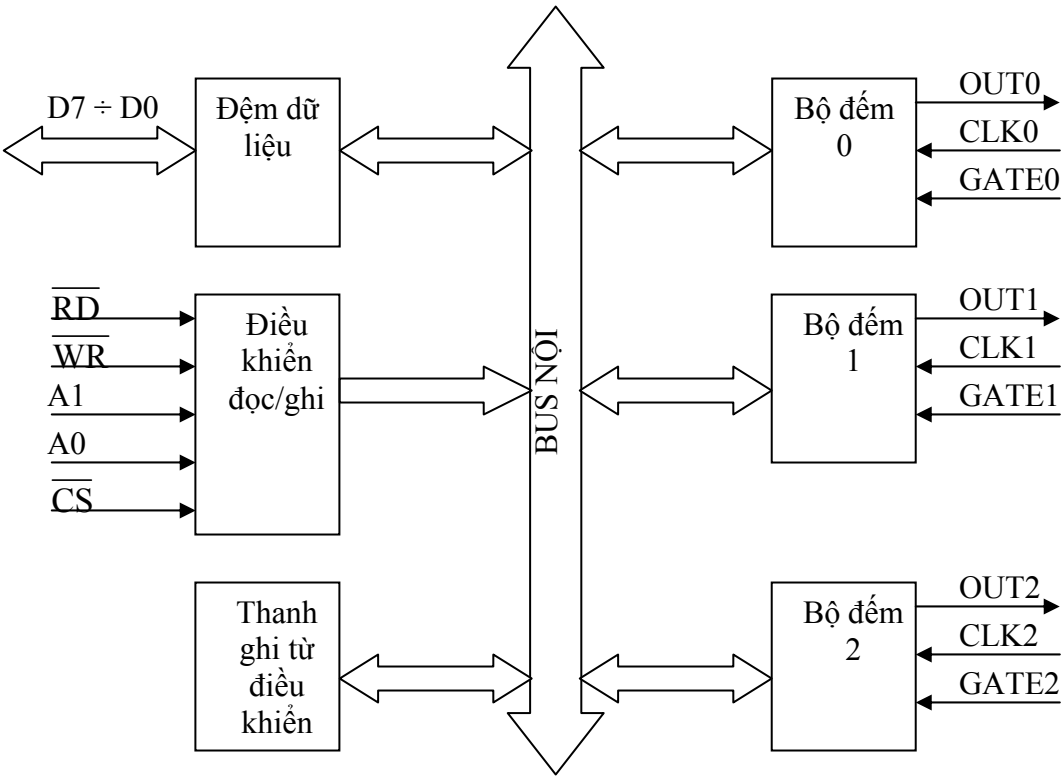


Hình 1.10 – Mạch khởi động cho 8284

4.2. Mạch định thời PIT – 8253 / 8254 (Programmable Interval Timer)



Hình 1.11 – Sơ đồ chân của PIT 8253



Hình 1.12 – Sơ đồ khối của PIT 8253

D7 ÷ D0: bus dữ liệu

CLK0 ÷ CLK2: ngõ vào xung clock cho các bộ đếm

OUT0 ÷ OUT2: ngõ ra bộ đếm

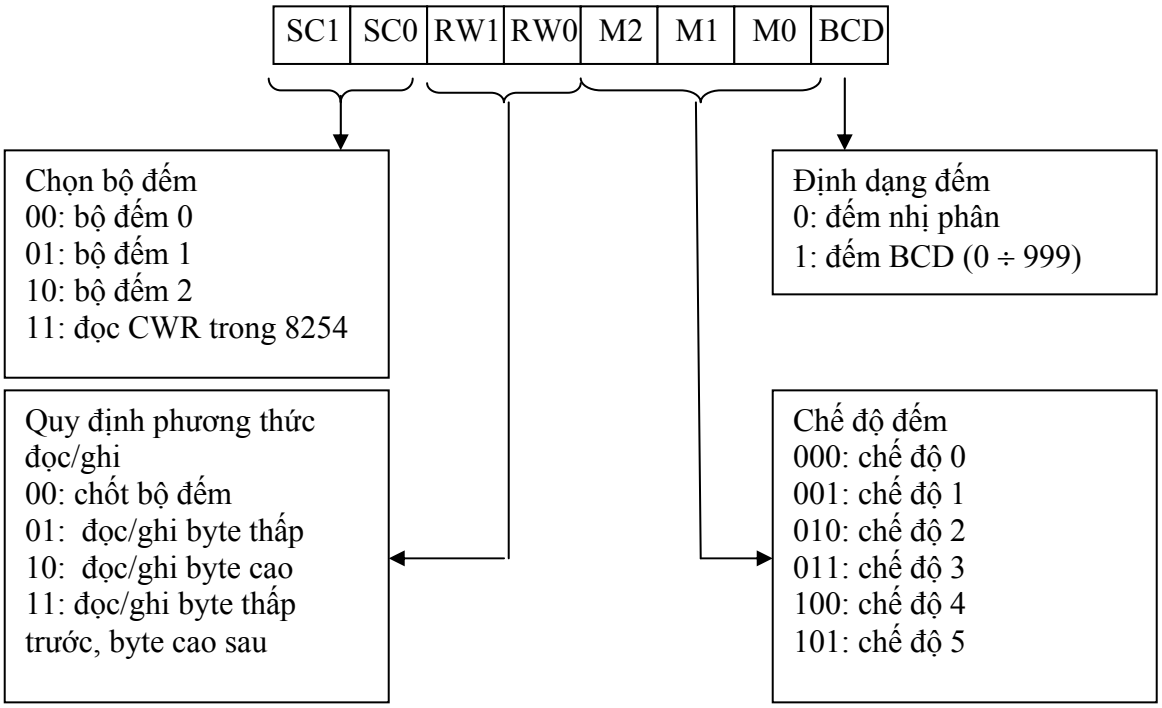
$\overline{RD}$, $\overline{WR}$: cho phép CPU đọc / ghi dữ liệu từ / đến các thanh ghi của 8253

A1, A0: giải mã chọn bộ đếm hay thanh ghi điều khiển, thường được nối với bus địa chỉ của CPU

A1	**A0**	**Chọn**
0	0	Bộ đếm 0
0	1	Bộ đếm 1
1	0	Bộ đếm 2
1	1	Thanh ghi từ điều khiển

G0 ÷ G2 (Gate): cho phép hay cấm các bộ đếm hoạt động (=1: cho phép, =0: cấm).

PIT 8253 có tất cả 5 chế độ đếm tùy thuộc vào giá trị trong thanh ghi điều khiển.



Hình 1.13 – Dạng từ điều khiển của 8253

PIT 8253 có 3 bộ đếm lùi 16 bit có thể lập trình và độc lập với nhau. Mỗi bộ đếm có tín hiệu xung clock riêng (8254 tương tự như 8253 nhưng có thêm lệnh đọc thanh ghi từ điều khiển CWR).

❖ **Các chế độ đếm:**

Chế độ 0 (Interrupt on Terminal Count): tín hiệu ngõ ra ở mức thấp cho tới khi bộ đếm tràn thì sẽ chuyển lên mức cao.

Chế độ 1 (Programmable Monoflop): tín hiệu ngõ ra chuyển xuống mức thấp tại cạnh âm của xung clock đầu tiên và sẽ chuyển lên mức cao khi bộ đếm kết thúc.

Chế độ 2 (Rate Generator): tín hiệu ngõ ra xuống mức thấp trong chu kỳ đầu tiên và sau đó chuyển lên mức cao trong các chu kỳ còn lại.

Chế độ 3 (Square-Wave Generator): tương tự như chế độ 2 nhưng xung ngõ ra là sóng vuông khi giá trị đếm chẵn và sẽ thêm một chu kỳ ở mức cao khi giá trị đếm lẻ.

Chế độ 4 (Software-triggered Pulse): giống như chế độ 2 nhưng xung Gate không khởi động quá trình đếm mà sẽ đếm ngay khi số đếm ban đầu được nạp. Ngõ ra ở mức cao để đếm và xuống mức thấp trong chu kỳ xung đếm. Sau đó, ngõ ra sẽ trở lại mức cao.

Chế độ 5 (Hardware-triggered Pulse): giống như chế độ 2 nhưng xung Gate không khởi động quá trình đếm mà được khởi động bằng cạnh dương của xung clock ngõ vào. Ngõ ra ở mức cao và xuống mức thấp sau một chu kỳ clock khi quá trình đếm kết thúc.

❖ **Ba chức năng của 8253 trong PC:**

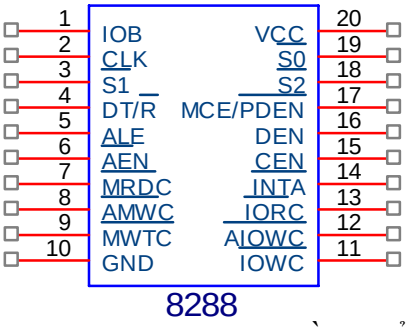
Cập nhật đồng hồ hệ thống: bộ đếm 0 của PIT phát tuần hoàn một ngắt cứng qua IRQ0 của 8259 để CPU có thể thay đổi đồng hồ hệ thống. Bộ đếm hoạt động trong chế độ 2. Ngõ vào được cấp xung clock tần số 1.19318 MHz. G0 = 1 để bộ đếm luôn được phép đếm. Giá trị ban đầu được nạp là 0 cho phép PIT phát ra xung chính xác với tần số:1.19318/65536 = 18.206Hz. Cạnh dương của mỗi xung này sẽ tạo ra một ngắt cứng trong 8259. Yêu cầu này sẽ dẫn tới ngắt 08h để cập nhật đồng hồ hệ thống 18.206 lần trong 1 giây.

Làm tươi bộ nhớ: PIT nối với chip DMAC dùng làm tươi bộ nhớ DRAM. Bộ đếm 1 sẽ định kỳ kích hoạt kênh 0 của DMAC-8237A để tiến hành 1 chu trình đọc giả làm tươi bộ nhớ. Bộ nhớ 1 hoạt động trong chế độ 3 phát sóng vuông với giá trị nạp ban đầu là 18. Do đó sóng vuông được phát ra có tần số 1,19318 MHz/18 = 66288 Hz (chu kỳ bằng 0.015s). Như vậy cứ sau 15 ms cạnh dương của sóng vuông này sẽ tạo 1 chu kỳ đọc giả để làm tươi bộ nhớ.

Phát sóng âm với tần số biến đổi ra loa của PC: Bộ đếm 2 của PIT được dùng để phát sóng âm ra loa của PC.

4.3. Mạch điều khiển bus 8288

Mạch điều khiển bus 8288 lấy một số tín hiệu điều khiển của CPU và cung cấp các tín hiệu điều khiển cần thiết cho hệ vi xử lý.



Hình 1.14 – Mạch điều khiển bus 8288

IOB (Input / Output Bus Mode): điều khiển để 8288 làm việc ở các chế độ bus khác nhau.

CLK (Clock): ngõ vào lấy từ xung clock hệ thống (từ 8284) và dùng để đồng bộ toàn bộ các xung điều khiển đi ra từ mạch 8288.

DT/$\overline{R}$ (Data Transmit/Receive): CPU truyền (1) hay nhận (0) dữ liệu.

ALE (Address Latch Enable): tín hiệu cho phép chốt địa chỉ, tín hiệu này thường được nối với chân G của 74573 để điều khiển chốt địa chỉ.

$\overline{AEN}$ (Address Enable): chờ thời gian trễ khoảng 150 ns sẽ tạo các tín hiệu điều khiển ở đầu ra của 8288 để đảm bảo rằng địa chỉ sử dụng đã hợp lệ.

$\overline{S2}$, $\overline{S1}$, $\overline{S0}$: các tín hiệu trạng thái lấy trực tiếp từ CPU. Tuỳ theo các giá trị nhận được mà 8288 sẽ đưa các tín hiệu theo bảng:

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	**Tạo tín hiệu**
0	**0**	**0**	$\overline{INTA}$
0	**0**	**1**	$\overline{IORC}$
0	**1**	**0**	$\overline{IOWC}$, $\overline{AIOWC}$
0	**1**	**1**	**Không**
1	**0**	**0**	$\overline{MRDC}$
1	**0**	**1**	$\overline{MRDC}$
1	**1**	**0**	$\overline{MWTC}$, $\overline{AMWC}$
1	**1**	**1**	**Không**

$\overline{MRDC}$ (Memory Read Command): điều khiển đọc bộ nhớ

$\overline{MWTC}$ (Memory Write Command): điều khiển ghi bộ nhớ

$\overline{AMWC}$ (Advanced MWTC),: giống như $\overline{MWTC}$ nhưng hoạt động sớm hơn một chút dùng cho các bộ nhớ chậm đáp ứng kịp tốc độ CPU.

$\overline{IOWC}$ (I/O Write Command): điều khiển ghi ngoại vi

$\overline{AIOWC}$ (Advanced IOWC),: giống như $\overline{IOWC}$ nhưng hoạt động sớm hơn một chút dùng cho các ngoại vi chậm đáp ứng kịp tốc độ CPU.

$\overline{IORC}$ (I/O Read Command): điều khiển đọc ngoại vi

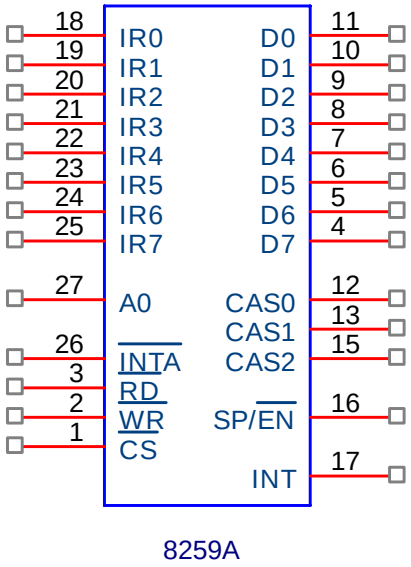
$\overline{INTA}$ (Interrupt Acknowledge): ngõ ra thông báo CPU chấp nhận yêu cầu ngắt của thiết bị ngoại vi

CEN (Command Enable): cho phép đưa ra tín hiệu DEN và các tín hiệu điều khiển khác của 8288.

DEN (Data Enable): điều khiển bus dữ liệu thành bus cục bộ hay bus hệ thống.

MCE / ***$\overline{PDEN}$*** (Master Cascade Enable / Peripheral Data Enable): định chế độ làm việc cho mạch điều khiển ngắt PIC 8259 để nó làm việc ở chế độ master.

4.4. Chip điều khiển ngắt ưu tiên PIC 8259A (Priority Interrupt Controller)



Hình 1.15 – Sơ đồ chân của 8259A

Trong trường hợp nhiều yêu cầu ngắt cần phải phục vụ, ta thường dùng vi mạch 8259A để giải quyết vấn đề ưu tiên. 8259A có thể giải quyết được 8 yêu cầu ngắt với 8 mức ưu tiên khác nhau.

❖ **Các khối chức năng:**

IRR (thanh ghi yêu cầu ngắt): lưu trữ các yêu cầu ngắt tại ngõ vào

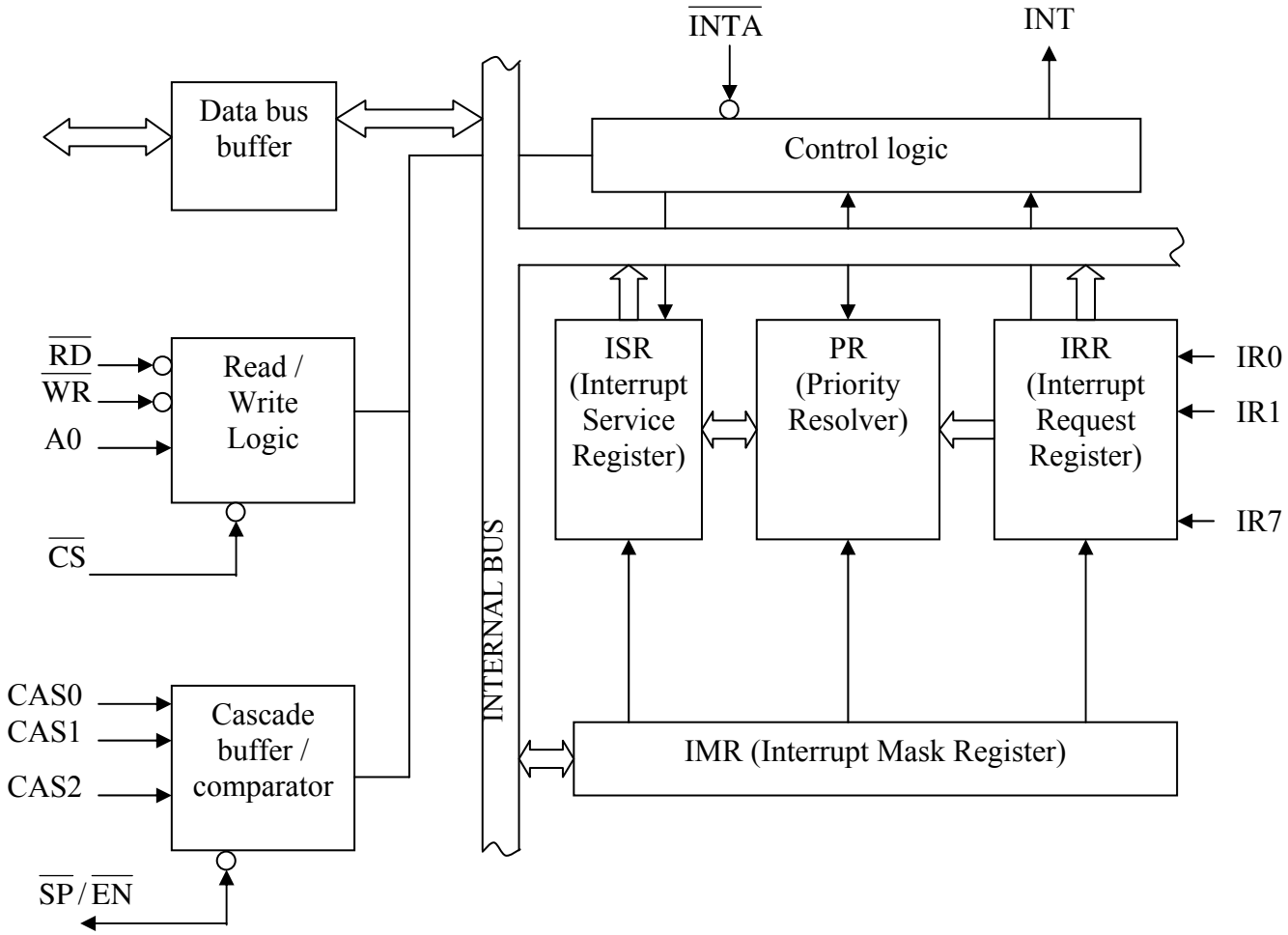
ISR (thanh ghi phục vụ ngắt): lưu trữ các yêu cầu ngắt đang phục vụ

IMR (thanh ghi mặt nạ ngắt): lưu trữ mặt nạ của các yêu cầu ngắt tại ngõ vào

Control logic (logic điều khiển): gởi yêu cầu ngắt tới chân INTR của CPU khi có tín hiệu ngắt tại ngõ vào của 8259A và nhận trả lời chấp nhận yêu cầu ngắt hay không $\overline{\text{INTA}}$ từ CPU để đưa kiểu ngắt vào CPU.

Data bus buffer (đệm bus dữ liệu): giao tiếp giữa 8259A với bus dữ liệu của CPU.

Cascade buffer / comparator (đệm nối tầng và so sánh): lưu trữ và so sánh số hiệu của các kiểu ngắt trong trường hợp dùng nhiều mạch 8259A.



Hình 1.16 – Sơ đồ khối của PIC 8259A

❖ **Các tín hiệu điều khiển:**

CAS0 ÷ 2 (In, Out): các ngõ vào chọn mạch 8259A tớ (slave) từ mạch 8259A chủ (master) trong trường hợp dùng nhiều mạch 8259A để tăng yêu cầu ngắt.

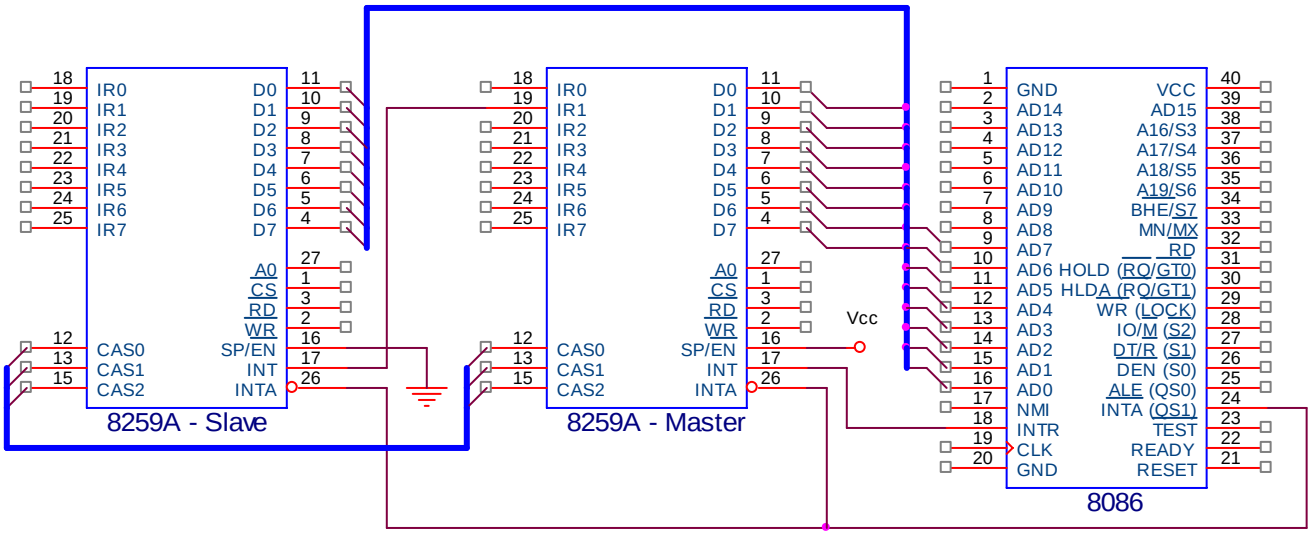
$\overline{SP}$ / $\overline{EN}$ (In, Out) (Slave Program / Enable Buffer): nếu 8259A hoạt động ở chế độ không dùng đệm dữ liệu thì tín hiệu này dùng để xác định mạch 8259A là mạch chủ ($\overline{SP}$ = 1) hay tớ ($\overline{SP}$ = 0). Nếu 8259A hoạt động ở chế độ có đệm dữ liệu thì tín hiệu này dùng để cho phép giao tiếp giữa 8259A và CPU, khi đó mạch 8259A là master hay slave phải dựa vào từ lệnh khởi động ICW4.

INT (Out): tín hiệu yêu cầu ngắt đưa đến CPU (chân INTR).

$\overline{INTA}$ (In): nhận trả lời chấp nhận ngắt hay không từ CPU (chân $\overline{INTA}$)

A0: cho phép chọn các từ điều khiển của 8259A.

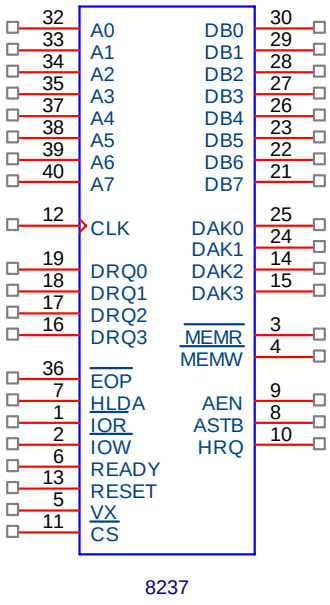
8259A cho phép xử lý 8 ngắt với 8 mức ưu tiên khác nhau. Trong trường hợp hệ thống có số lượng ngắt lớn hơn thì có thể mắc nhiều 8259A liên tầng.

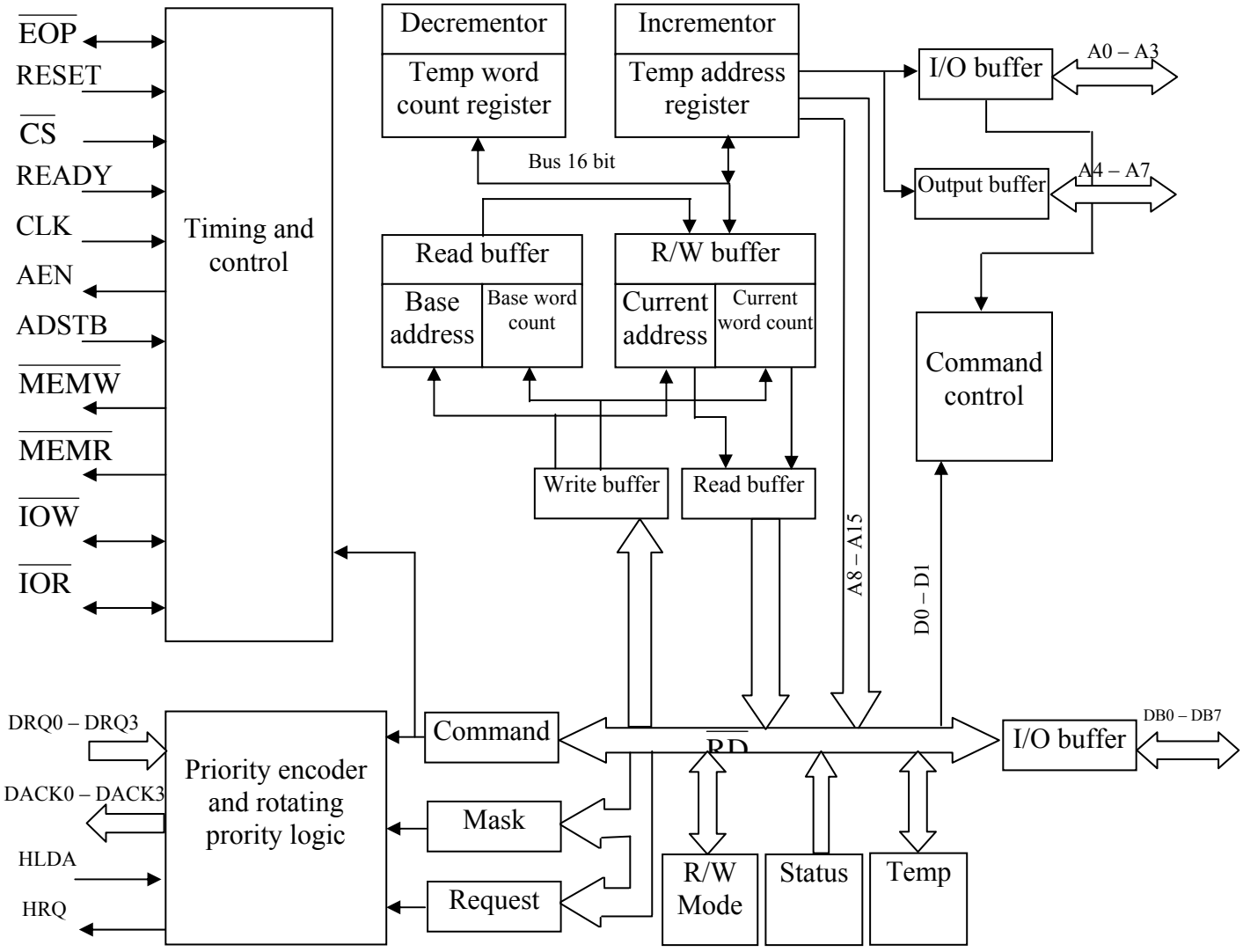


Hình 1.17 – 8259A mắc liên tầng

4.5. Chip điều khiển truy nhập bộ nhớ trực tiếp DMAC 8237 (Direct Memory Access Controller)

DMAC 8237 có thể thực hiện truyền dữ liệu theo 3 kiểu: kiểu đọc (từ bộ nhớ ra thiết bị ngoại vi), kiểu ghi (từ thiết bị ngoại vi đến bộ nhớ) và kiểu kiểm tra.





Hình 1.18 – Sơ đồ chân và sơ đồ khối của DMAC 8237A

❖ **Khối Timing and Control (định thời và điều khiển):**

Tạo các tín hiệu định thời và điều khiển cho bus ngoài (external bus). Các tín hiệu này được đồng bộ với xung clock đưa vào DMAC (tần số xung clock tối đa là 5 MHz).

❖ **Khối Priority encoder and rotating priority logic (mã hóa ưu tiên và quay mức ưu tiên):**

DMAC 8237A có 2 mô hình ưu tiên: mô hình ***ưu tiên cố định*** (***fixed priority***) và mô hình ***ưu tiên quay*** (***rotating priority***). Trong mô hình ưu tiên cố định, kênh 0 sẽ có mức ưu tiên cao nhất còn kênh 3 có mức ưu tiên thấp nhất. Còn đối với mô hình ưu tiên quay thì mức ưu tiên khi khởi động giống như mô hình ưu tiên cố định nhưng khi yêu cầu DMA tại một kênh nào đó được phục vụ thì sẽ được đặt xuống mức ưu tiên thấp nhất.

❖ **Khối Command Control (điều khiển lệnh):**

Giải mã các thanh ghi lệnh (xác định thanh ghi sẽ được truy xuất và loại hoạt động cần thực hiện).

❖ Các thanh ghi:

DMAC 8237A có tất cả 12 loại thanh ghi nội khác nhau:

Tên	Kích thước (bit)	Số lượng
Thanh ghi địa chỉ cơ sở (Base Address Register)	16	4
Thanh ghi đếm từ cơ sở (Base Word Count Register)	16	4
Thanh ghi địa chỉ hiện hành (Current Address Register)	16	4
Thanh ghi đếm từ hiện hành (Current Word Count Register)	16	4
Thanh ghi địa chỉ tạm (Temporary Address Register)	16	1
Thanh ghi đếm từ tạm (Temporary Word Count Register)	16	1
Thanh ghi trạng thái (Status Register)	8	1
Thanh ghi lệnh (Command Register)	8	1
Thanh ghi tạm (Temporary Register)	8	1
Thanh ghi chế độ (Mode Register)	6	4
Thanh ghi mặt nạ (Mask Register)	4	1
Thanh ghi yêu cầu (Request Register)	4	1

❖ Chức năng các chân của 8237A:

$\overline{CLK}$ (Input): tín hiệu xung clock của mạch. Tín hiệu này thường được lấy từ 8284 sau khi qua cổng đảo.

$\overline{CS}$ (Input): thường được nối với bộ giải mã địa chỉ.

RESET (Input): khởi động 8237A, được nối với ngõ RESET của 8284. Khi Reset thì thanh ghi mặt nạ được lập còn các phần sau bị xóa:

+ Thanh ghi lệnh
+ Thanh ghi trạng thái
+ Thanh ghi yêu cầu
+ Thanh ghi tạm
+ Flip-flop đầu/cuối (First/Last flip-flop)

READY (Input): nối với READY của CPU để tạo chu kỳ đợi khi truy xuất các thiết bị ngoại vi hay bộ nhớ chậm.

HLDA (Hold Acknowledge)(Input): tín hiệu chấp nhận yêu cầu treo từ CPU

$DRQ_0 - DRQ_3$ (DMA Request)(Input): các tín hiệu yêu cầu treo từ thiết bị ngoại vi.

DB0 – DB7 (Input, Output): nối đến bus địa chỉ và dữ liệu của CPU

$\overline{IOR}$, $\overline{IOW}$ (Input, Output): sử dụng trong các chu kỳ đọc và ghi

$\overline{EOP}$ (End Of Process)(Input,Output): bắt buộc DMAC kết thúc quá trình DMA nếu là ngõ vào hay dùng để báo cho một kênh biết là dữ liệu đã chuyển xong (Terminal count – TC), thường dùng như yêu cầu ngắt để CPU kết thúc quá trình DMA.

A0 – A3 (Input, Output): chọn các thanh ghi trong 8237A khi lập trình hay dùng để chứa 4 bit địa chỉ thấp.

A4 – A7 (Output): chứa 4 bit địa chỉ

HRQ (Hold Request)(Output): tín hiệu yêu cầu treo đến CPU

$DACK_0 – DACK_3$ (DMA Acknowledge)(Output): tín hiệu trả lời yêu cầu DMA cho các kênh.

AEN (Output): cho phép lấy địa chỉ vùng nhớ cần trao đổi

ADSTB (Address Strobe)(Output): chốt các bit địa chỉ cao A8 – A15 chứa trong các chân DB0 – DB7

$\overline{MEMR}$, $\overline{MEMW}$ (Output): dùng để đọc / ghi bộ nhớ.

❖ **Các thanh ghi nội:**

Các thanh ghi nội trong DMAC 8237A được truy xuất nhờ các bit địa chỉ thấp A0 – A3.

Bit địa chỉ				**Địa chỉ**	**Chọn chức năng**	**R/W?**
A3	**A2**	**A1**	**A0**			
0	0	0	0	X0	Thanh ghi địa chỉ bộ nhớ kênh 0	R/W
0	0	0	1	X1	Thanh ghi đếm từ kênh 0	R/W
0	0	1	0	X2	Thanh ghi địa chỉ bộ nhớ kênh 1	R/W
0	0	1	1	X3	Thanh ghi đếm từ kênh 1	R/W
0	1	0	0	X4	Thanh ghi địa chỉ bộ nhớ kênh 2	R/W
0	1	0	1	X5	Thanh ghi đếm từ kênh 2	R/W
0	1	1	0	X6	Thanh ghi địa chỉ bộ nhớ kênh 3	R/W
0	1	1	1	X7	Thanh ghi đếm từ kênh 3	R/W
1	0	0	0	X8	Thanh ghi trạng thái / lệnh	R/W
1	0	0	1	X9	Thanh ghi yêu cầu	W
1	0	1	0	XA	Thanh ghi mặt nạ cho một kênh	W
1	0	1	1	XB	Thanh ghi chế độ	W
1	1	0	0	XC	Xóa flip-flop đầu/cuối	W
1	1	0	1	XD	Xóa toàn bộ các thanh ghi / đọc thanh ghi tạm	W/R
1	1	1	0	XE	Xóa thanh ghi mặt nạ	W
1	1	1	1	XF	Thanh ghi mặt nạ	W

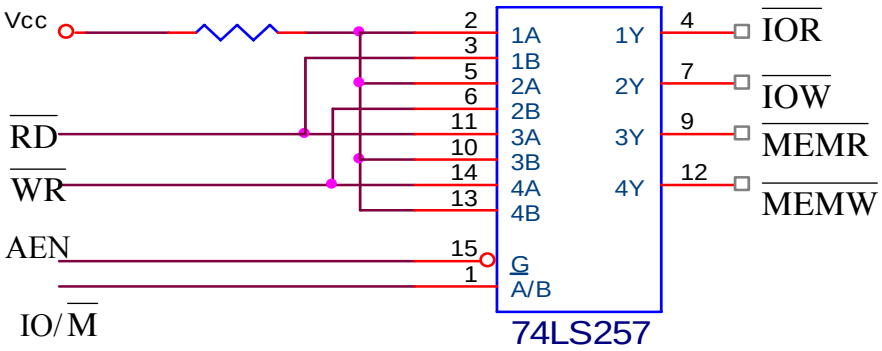
Địa chỉ các thanh ghi nội dùng ghi / đọc địa chỉ:

Kênh	$\overline{IOR}$	$\overline{IOW}$	**A3**	**A2**	**A1**	**A0**	**Thanh ghi**	**R/W?**
0	1	0	0	0	0	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	0	0	0	Địa chỉ hiện hành	R
	1	0	0	0	0	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	0	0	1	Bộ đếm hiện hành	R
1	1	0	0	0	1	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	0	1	0	Địa chỉ hiện hành	R
	1	0	0	0	1	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	0	1	1	Bộ đếm hiện hành	R
2	1	0	0	1	0	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	1	0	0	Địa chỉ hiện hành	R
	1	0	0	1	0	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	1	0	1	Bộ đếm hiện hành	R
3	1	0	0	1	1	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	1	1	0	Địa chỉ hiện hành	R
	1	0	0	1	1	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	1	1	1	Bộ đếm hiện hành	R

Địa chỉ các thanh ghi trạng thái và điều khiển:

$\overline{IOR}$ `	$\overline{IOW}$	**A3**	**A2**	**A1**	**A0**	**Thanh ghi**
1	0	1	0	0	0	Ghi thanh ghi lệnh
0	1	1	0	0	0	Đọc thanh ghi trạng thái
1	0	1	0	0	1	Ghi thanh ghi yêu cầu
1	0	1	0	1	0	Ghi thanh ghi mặt nạ
1	0	1	0	1	1	Ghi thanh ghi chế độ
1	0	1	1	0	0	Xóa flip-flop đầu/cuối
1	0	1	1	0	1	Xóa tất cả các thanh ghi nội
0	1	1	1	0	1	
1	0	1	1	1	0	Địa chỉ cơ sở và địa chỉ hiện hành
0	1	1	1	1	0	Địa chỉ hiện hành
1	0	1	1	1	1	Bộ đếm cơ sở và bộ đếm hiện hành
0	1	1	1	1	1	Bộ đếm hiện hành

Mạch 8273A-5 chứa 4 kênh trao đổi dữ liệu DMA với mức ưu tiên lập trình được. 8237A-5 có tốc độ truyền 1 MBps cho mỗi kênh và 1 kênh có thể truyền 1 mảng có độ dài 64 KB. Để có thể sử dụng mạch DMAC 8237A, ta cần tạo tín hiệu điều khiển như sau:



Hình 1.19 – Tín hiệu điều khiển cho hệ thống làm việc với DMAC 8237A

Tín hiệu AEN từ 8237A dùng để cấm các tín hiệu điều khiển từ CPU khi DMAC đã nắm quyền điều khiển bus.

4.6. Chip điều khiển màn hình CRTC 6845 (Cathode Ray Tube Controller)

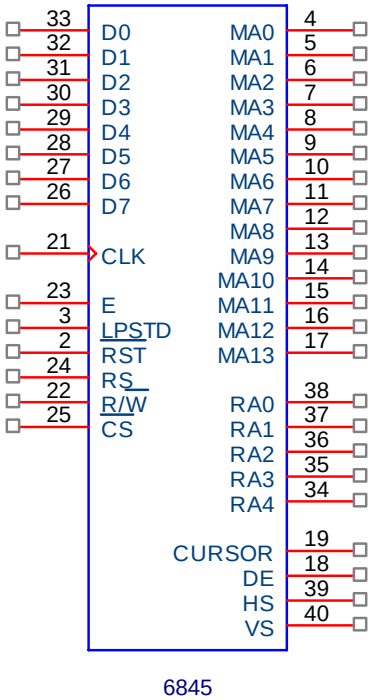
RST (Reset): khởi động lại 6845.

MA0 ÷ MA13 (Memory Address): 14 địa chỉ nhớ cho RAM màn hình.

DE (Display Enable): cho phép (=1) hay không (=0) các tín hiệu điều khiển và địa chỉ vùng hiện lên màn hình.

LPSTD (Light Pen Strobe): lưu trữ địa chỉ hiện hành của RAM màn hình trong thanh ghi bút sáng. CPU đọc thanh ghi và xác định vị trí bút sáng trên màn hình.

CURSOR: vị trí con trỏ đã quét (=1) hay chưa (=0).



Hình 1.20 – Sơ đồ chân của 6845

VS (Vertical Synchronization): ngõ ra tín hiệu đồng bộ quét dọc

HS (Horizontal Synchronization): ngõ ra tín hiệu đồng bộ quét ngang

RA0 ÷RA4 (Row Address): phân định hàng quét của ký tự trong chế độ văn bản (32 hàng quét). Trong chế độ đồ họa, chúng kết hợp với MA0 ÷ MA13 tạo các địa chỉ cho các bank RAM màn hình.

D0 ÷ D7: đường dữ liệu.

$\overline{CS}$: chọn chip.

RS (Regigter Select): chọn thanh ghi địa chỉ (=0) hay thanh ghi dữ liệu (=1).

E: xung âm kích hoạt bus dữ liệu và dùng như xung clock cho 6845 đọc / ghi dữ liệu vào các thanh ghi bên trong.

$R/\overline{W}$: đọc / ghi dữ liệu vào các thanh ghi.

CLK: dùng đồng bộ với tín hiệu của màn hình và thường bằng tốc độ hiện ký tự trên màn hình.

4.7. Chip đồng xử lý toán học 8087/80287/80387 (Mathematical co-processor)

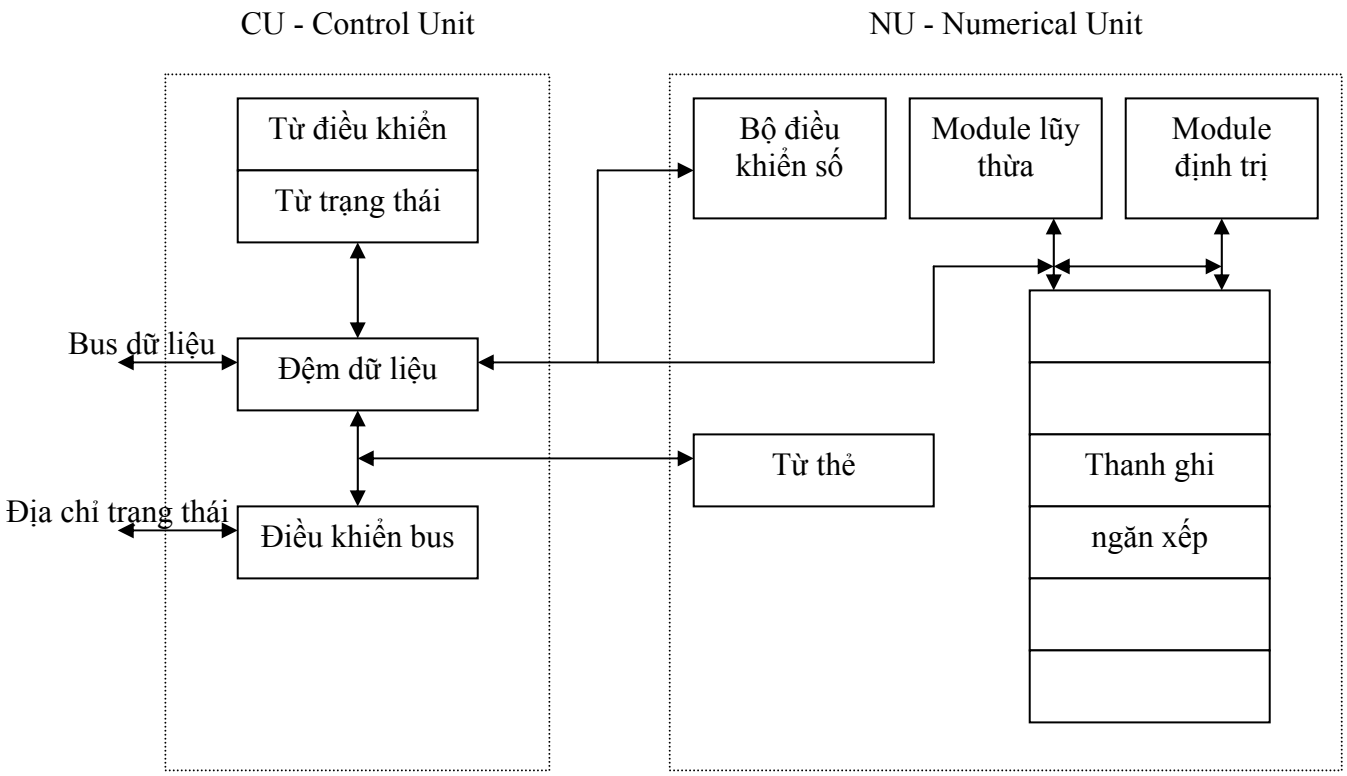
Các bộ đồng xử lý toán 80x87 hỗ trợ CPU trong việc tính toán các biểu thức dùng dấu chấm động như cộng, trừ, nhân, chia các số dấu chấm động, căn thức, logarit, … Chúng cho phép xử lý các phép toán này nhanh hơn nhiều so với CPU. Thời gian xử lý giữa 8087 và 8086 như sau (dùng xung clock 8 MHz):

Phép toán	**8087 [μs]**	**8086 [μs]**
Cộng / trừ	10.6	1000
Nhân	11.9	1000
Chia	24.4	2000
Căn bậc hai	22.5	12250
Tang	56.3	8125
Lũy thừa	62.5	10680
Lưu trữ	13.1	750

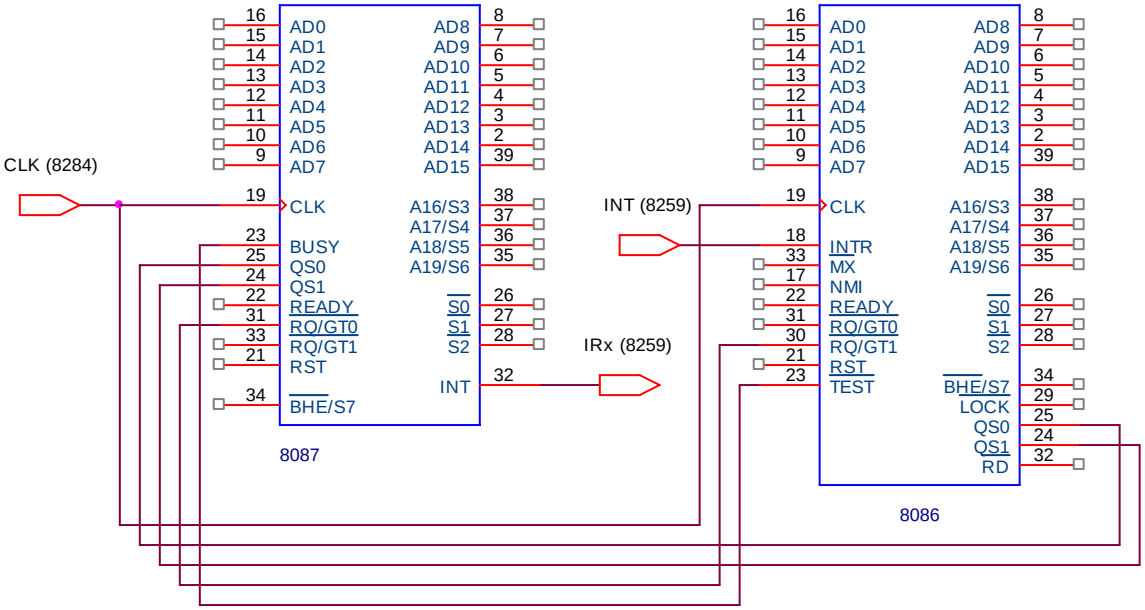
❖ **8087:**

8087 gồm một đơn vị điều khiển (CU – Control Unit) dùng để điều khiển bus và một đơn vị số học (NU – Numerical Unit) để thực hiện các phép toán dấu chấm động trong các mạch tính lũy thừa (exponent module) và mạch tính phần định trị (mantissa module). Khác với 8086, thay vì dùng các thanh ghi rời rạc là một ngăn xếp thanh ghi.

Đơn vị điều khiển nhận và giải mã lệnh, dọc và ghi các toán hạng, chạy các lệnh điều khiển riêng của 8087. Do đó, CU có thể đồng bộ với CPU trong khi NU đang thực hiện các công việc tính toán. CU bao gồm bộ điều khiển bus, bộ đệm dữ liệu và hàng lệnh.



Hình 1.21 – Sơ đồ khối của 8087



Hình 1.22 – Sơ đồ kết nối 8087 và CPU 8086

Ngăn xếp thanh ghi có tất cả 8 thanh ghi từ R0 ÷ R7, mỗi thanh ghi dài 80 bit trong đó bit 79 là bit dấu, bit 64 ÷ 78 dùng cho số mũ và phần còn lại là phần định trị. Dữ liệu truyền giữa các thanh ghi này được thực hiện rất nhanh do 8087 có độ rộng bus dữ liệu là 84 bit và không cần phải biến đổi định dạng.

Ngay sau khi reset PC, bộ đồng xử lý kiểm tra xem nó có được nối với PC hay không bằng các đường $\overline{BHE}$/S7. 8087 sẽ điều chỉnh độ dài của hàng lệnh cho phù hợp với CPU (nếu dùng 8086 thì độ dài là 6 byte).

8087 có một thanh ghi trạng thái là thanh ghi từ thẻ (tag word) gồm các cặp bit Tag0 ÷ Tag7 để lưu trữ các thông tin liên quan đến nội dung của các thanh ghi R0 ÷ R7 để cho phép thực hiện một số tác vụ nhanh hơn. Mỗi thanh ghi từ thẻ có 2 bit xác định 4 giá trị khác nhau của các thanh ghi Ri.

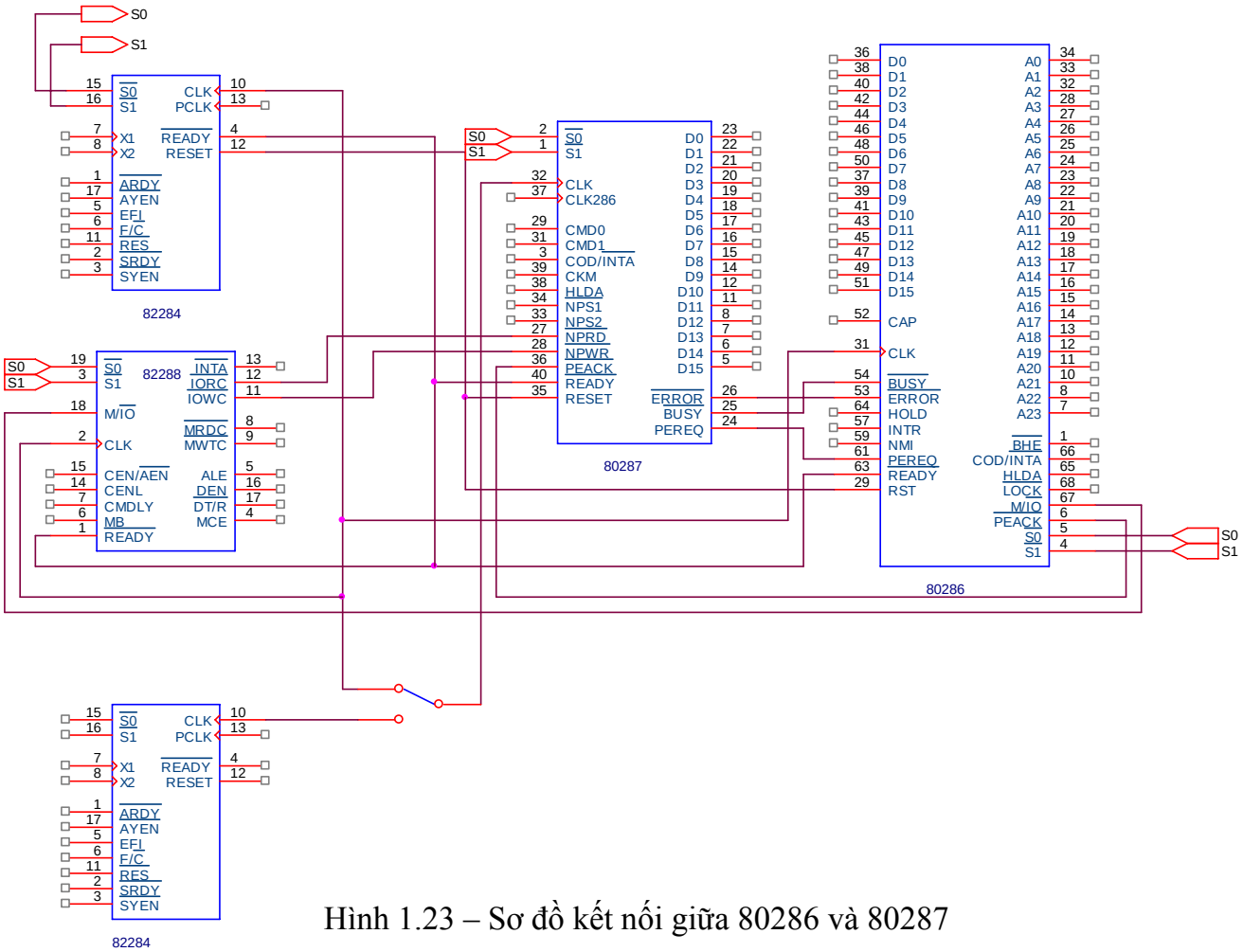
Tag = 00: xác định

Tag = 01: zero

Tag = 10: NAN, giá trị bất thường

Tag = 11: rỗng

❖ **80287:**



Hình 1.23 – Sơ đồ kết nối giữa 80286 và 80287

Do 80286 có chế độ mạch bảo vệ nên mạch ghép nối giữa 80286 và 80287 được thiết kế khác 8087 ở đơn vị điều khiển CU. Bộ đồng xử lý ở đây không thực hiện truy xuất bộ nhớ trực tiếp. Để truy xuất được bộ nhớ, 80287 không những cần một đơc vị định địa chỉ đơn giản của nó mà còn phải được tăng cường thêm chức năng quản lý bộ nhớ của 80286. Cấu trúc bên trong của 80287 cũng tương tự như 8087, chỉ có đơn vị bus thay đổi cho phù hợp với 80286. Khác vơi 8087, 80287 hoạt động không đồng bộ với CPU nên có thể dùng xung clock riêng.

❖ **80387:**

Ưu điểm của 80387 so với 80287 là có thể thực hiện các phép toán số học nhanh hơn. No có bus dữ liệu 32 bit như CPU và sử dụng công nghệ CMOS nên công suất tiêu thụ thấp hơn.

5. Bộ thanh ghi

μP 8086/8088 có tất cả 14 thanh ghi nội. Các thanh ghi này có thể phân loại như sau:

- Thanh ghi dữ liệu (data register)
- Thanh ghi chỉ số và con trỏ (index & pointer register)
- Thanh ghi đoạn (segment register)
- Thanh ghi trạng thái và điều khiển (status & control register)

5.1. Các thanh ghi dữ liệu

Các thanh ghi dữ liệu gồm có các thanh ghi 16 bit AX, BX, CX và DX trong đó nửa cao và nửa thấp của mỗi thanh ghi có thể định địa chỉ một cách độc lập. Các nửa thanh ghi này (8 bit) có tên là AH và AL, BH và BL, CH và CL, DH và DL.

Các thanh ghi này được sử dụng trong các phép toán số học và logic hay trong quá trình chuyển dữ liệu.

Bảng 2.8:

Thanh ghi	Sử dụng trong
AX	MUL, IMUL (toán hạng nguồn kích thước word) DIV, IDIV (toán hạng nguồn kích thước word) IN (nhập word) OUT (xuất word) CWD Các phép toán xử lý chuỗi (string)
AL	MUL, IMUL (toán hạng nguồn kích thước byte) DIV, IDIV (toán hạng nguồn kích thước byte) IN (nhập byte) OUT (xuất byte) XLAT AAA, AAD, AAM, AAS (các phép toán ASCII) CBW (đổi sang word)

	DAA, DAS (số thập phân) Các phép toán xử lý chuỗi (string)
AH	MUL, IMUL (toán hạng nguồn kích thước byte) DIV, IDIV (toán hạng nguồn kích thước byte) CBW (đổi sang word)
BX	XLAT
CX	LOOP, LOOPE, LOOPNE Các phép toán string với tiếp dầu ngữ REP
CL	RCR, RCL, ROR, ROL (quay với số đếm byte) SHR, SAR, SAL (dịch với số đếm byte)
DX	MUL, IMUL (toán hạng nguồn kích thước word) DIV, IDIV (toán hạng nguồn kích thước word)

AX (ACC – Accumulator): thanh ghi tích luỹ

BX (Base): thanh ghi cơ sở

CX (Count): đếm

DX (Data): thanh ghi dữ liệu

5.2. Các thanh ghi chỉ số và con trỏ

Bao gồm các thanh ghi 16 bit SP, BP, SI và DI, thường chứa các giá trị offset (độ lệch) cho các phần tử định địa chỉ trong một phân đoạn (segment). Chúng có thể được sử dụng trong các phép toán số học và logic. Hai thanh ghi con trỏ (SP – Stack Pointer và BP – Base Pointer) cho phép truy xuất dễ dàng đến các phần tử đang ở trong ngăn xếp (stack) hiện hành. Các thanh ghi chỉ số (SI – Source Index và DI – Destination Index) được dùng để truy xuất các phần tử trong các đoạn dữ liệu và doạn thêm (extra segment). Thông thường, các thanh ghi con trỏ liên hệ đến đoạn stack hiện hành và các thanh ghi chỉ số liên hệ đến doạn dữ liệu hiện hành. SI và DI dùng trong các phép toán chuỗi.

5.3. Các thanh ghi đoạn

Bao gồm các thanh ghi 16 bit CS (Code segment), DS (Data segment), SS (stack segment) và ES (extra segment), dùng để định địa chỉ vùng nhớ 1 MB bằng cách chia thành 16 đoạn 64 KB.

Tất cả các lệnh phải ở trong đoạn mã hiện hành, được định địa chỉ thông qua thanh ghi CS. Offset (độ lệch) của mã được xác định bằng thanh ghi IP. Dữ liệu chương trình thường được đặt ở đoạn dữ liệu, định vị thông qua thanh ghi DS. Stack định vị thông qua thanh ghi SS. Thanh ghi đoạn thêm có thể sử dụng để định địa chỉ các toán hạng, dữ liệu, bộ nhớ và các phần tử khác ngoài đoạn dữ liệu và stack hiện hành.

5.4. Các thanh ghi điều khiển và trạng thái

Thanh ghi con trỏ lệnh IP (Instruction Pointer) giống như bộ đếm chương trình (Program Counter). Thanh ghi điều khiển này do BIU quản lý nhằm lưu trữ offset từ bắt đầu đoạn mã đến lệnh thực thi kế tiếp. Ta không thể xử lý trực tiếp trên thanh ghi IP.

Thanh ghi cờ (Flag register) hay từ trạng thái 16 bit chứa 3 bit điều khiển (TF, IF và DF) và 6 bit trạng thái (OF, SF, ZF, AF, PF và CF) còn các bit còn lại mà 8086/8088 không sử dụng thì không thể truy xuất được.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

- OF (Overflow - tràn): OF = 1 xác định tràn số học, xảy ra khi kết quả vượt ra ngoài phạm vi biểu diễn
- DF (Direction- hướng): xác định hướng chuyển string, DF = 1 khi μP làm việc với string theo thứ tự từ phải sang trái.
- IF (Interrupt - ngắt): cho phép hay cấm các interrupt có mặt nạ
- TF (Trap - bẫy): đặt μP vào chế độ từng bước, dùng cho các chương trình gỡ rối (debugger).
- SF (Sign - dấu): dùng để chỉ các kết quả số học là số dương (SF = 0) hay âm (SF = 1).
- ZF (Zero): = 1 nếu kết quả của phép toán trước là 0.
- AF (Auxiliary – nhớ phụ): dùng trong các số thập phân để chỉ nhớ từ nửa byte thấp hay mượn từ nửa byte cao.
- PF (Parity): PF = 1 nếu kết quả của phép toán là có tổng số bit 1 là chẵn (dùng để kiểm tra lỗi truyền dữ liệu)
- CF (Carry): CF = 1 nếu có nhớ hay mượn từ bit cao nhất của kết quả. Cờ này cũng dùng cho các lệnh quay.

Chương 2
NGẮT VÀ SỰ KIỆN

1. Khái niệm

Ngắt (interrupt) là quá trình dừng chương trình chính đang chạy để ưu tiên thực hiện một chương trình khác, chương trình này được gọi là chương trình phục vụ ngắt (ISR – Interrupt Service Routine). ISR hoàn toàn giống với một chương trình bình thường trên máy tính, nghĩa là nó có khả năng truy xuất đến tất cả các lệnh ngôn ngữ máy của µP. Tuy nhiên cuối ISR sẽ kết thúc bằng lệnh IRET (Interrupt Return) để µP tiếp tục thực hiện lệnh đã kết thúc trước đây.

Các nguyên nhân dẫn đến ngắt là:

- Bản thân chương trình đang thực hiện bị lỗi, ví dụ như: chia cho 0, …
- Do tác động của thiết bị ngoại vi, ví dụ như: thực hiện lệnh in nhưng máy in lỗi, ghi dữ liệu vào đĩa nhưng không có đĩa, …
- Do lập trình viên chủ động gọi các ngắt có sẵn.

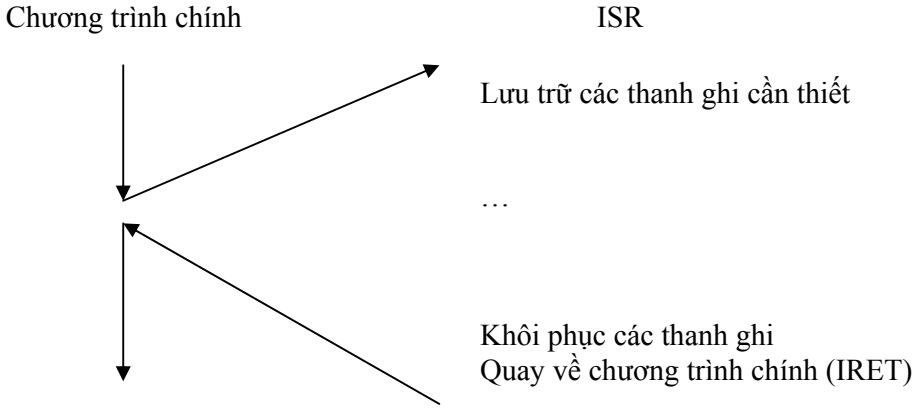
Một cách đơn giản, chúng ta có thể xem ngắt như là quá trình gọi chương trình con nhưng các chương trình con này được tạo ra sẵn trong máy tính và quá trình gọi này có thể xảy ra tại thời điểm không xác định trước.

Sự kiện (Event) là một tác động lên một đối tượng trong môi trường Windows. Khi có một sự kiện xảy ra, Windows sẽ gởi thông điệp (message) đến đối tượng. Các sự kiện thường xảy ra là:

- Sự kiện chuột: Click, Double Click, …
- Sự kiện bàn phím: nhấn phím, nhả phím, …
- Sự kiện cửa sổ: Activate, Load, Unload, …

2. Các loại ngắt và bảng vector ngắt

Quá trình ngắt có thể mô tả như sau:

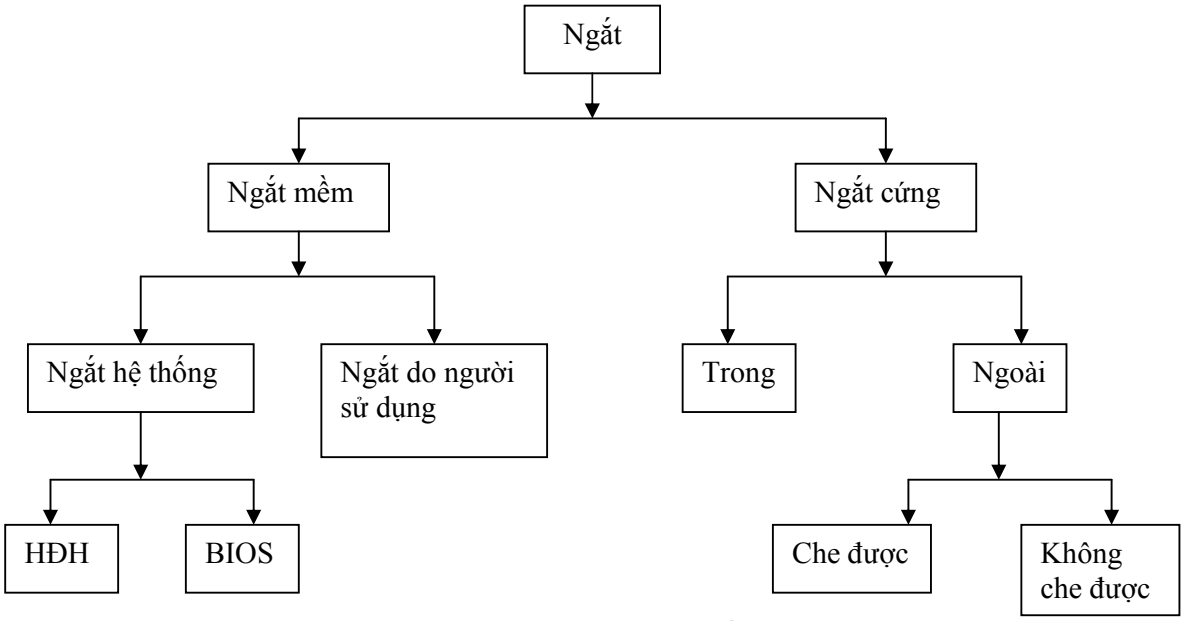


Hình 2.1 – Quá trình thực hiện ngắt

Trong các quá trình ngắt, ta phân biệt thành 2 loại: ngắt cứng và ngắt mềm. Ngắt mềm là ngắt được gọi bằng một lệnh trong chương trình ngôn ngữ máy. Ngắt mềm được thục hiện trên hợp ngữ thông qua lệnh INT. Đối với các ngôn ngữ bậc cao hơn, vẫn cho phép thực hiện gọi ngắt nhưng phải được biên dịch thành lệnh INT trong hợp ngữ rồi mời thực hiện.

Khác với ngắt mềm, ngắt cứng không được khởi động bên trong máy tính mà do các linh kiện điện tử tác đông lên hệ thống. Ngắt cứng cũng được chia thành 2 loại: ngắt che được và ngắt không che được. Ngắt che được là ngắt có thể cho phép hay không cho phép thực thi bằng phần mềm thông qua cờ ngắt IF (Interrupt Flag): lệnh CLI (Clear Interrupt Flag) sẽ cấm ngắt và lệnh STI (Set Interrupt Flag) sẽ cho phép các ngắt này hoạt động.

Các loại ngắt khác nhau có thể mô tả như sau:



Hình 2.2 – Các loại ngắt

Khi thực hiện lệnh gọi một ngắt nào đó, chương trình con phục vụ cho ngắt sẽ được gọi. Để thực hiện các ngắt tương ứng, địa chỉ thực hiện các chương trình con phục vụ ngắt được đặt trong một bảng, gọi là bảng vector ngắt.

Bảng vector ngắt gồm có 256 phần tử, mỗi phần tử gồm 4 byte ứng với 256 ngắt (từ ngắt 0 đến ngắt 0FFh). Mỗi phần tử trong bảng vector ngắt chứa 2 địa chỉ: địa chỉ thanh ghi đoạn đưa vào CS và địa chỉ offset của chương trình phục vụ ngắt đưa vào IP.

Bảng vector ngắt có thể mô tả như sau:

Địa chỉ	Địa chỉ ISR	Số thứ tự ngắt
0000h:0000h	CS	0
0000h:0002h	IP	
0000h:0004h	CS	1
0000h:0006h	IP	
0000h:0008h	CS	2
0000h:000Ah	IP	
…	…	…
0000h:03FCh	CS	255
0000h:03FEh	IP	

Khi có một quá trình ngắt xảy ra, CPU sẽ tìm địa chỉ bắt đầu của chương trình ngắt được chứa trong bảng vector ngắt theo số thự tự ngắt. Do một phần tử trong bảng vector ngắt chiếm 4 byte nên để tìm giá trị địa chỉ trong bảng vector ngắt, ta chỉ cần nhân số thứ tự ngắt với 4.

Danh sách các ngắt mô tả như sau:

STT	Địa chỉ	Chức năng
00h	0000h – 0003h	CPU: chia cho 0
01h	0004h – 0007h	CPU: thực hiện từng lệnh
02h	0008h – 000Bh	CPU: Lỗi RAM
03h	000Ch – 000Fh	CPU: thực hiện đến điểm dừng
04h	0010h – 0013h	CPU: tràn số
05h	0014h – 0017h	In trang màn hình (Print Screen)
06h, 07h	0018h – 001Fh	Dành riêng
08h	0020h – 0023h	IRQ0: ngắt đồng hồ (18.2 lần / giây)
09h	0024h – 0027h	IRQ1: ngắt bàn phím
0Ah	0028h – 002Bh	IRQ2: Dành riêng
0Bh	002Ch – 002Fh	IRQ3: Giao tiếp nối tiếp 1
0Ch	0030h – 0033h	IRQ4: Giao tiếp nối tiếp 2
0Dh	0034h – 0037h	IRQ5: Đĩa cứng

0Eh	0038h – 003Bh	IRQ6: Đĩa mềm
0Fh	003Ch – 003Fh	IRQ7: Máy in
10h	0040h – 0043h	BIOS: màn hình
11h	0044h – 0047h	BIOS: xác định cấu hình máy tính
12h	0048h – 004Bh	BIOS: xác định kích thước RAM
13h	004Ch – 004Fh	BIOS: truy nhập đĩa cứng / đĩa mềm
14h	0050h – 0053h	BIOS: truy nhập giao tiếp nối tiếp
15h	0054h – 0057h	BIOS: truy nhập cassette hay mở rộng ngắt
16h	0058h – 005Bh	BIOS: kiểm tra bàn phím
17h	005Ch – 005Fh	BIOS: truy nhập máy in
18h	0060h – 0063h	Chương trình xâm nhập ROM BASIC
19h	0064h – 0067h	BIOS: khởi động hệ thống (khi nhấn Ctrl-Alt-Del)
1Ah	0068h – 006Bh	BIOS: đọc / ghi ngày / giờ
1Bh	006Ch – 006Fh	Nhấn phím Break
1Ch	0070h – 0073h	Gọi sau INT 08h
1Dh	0074h – 0077h	Địa chỉ bảng tham số màn hình
1Eh	0078h – 007Bh	Địa chỉ bảng tham số đĩa mềm
1Fh	007Ch – 007Fh	Địa chỉ bảng font có ký tự mở rộng
20h	0080h – 0083h	DOS: kết thúc chương trình
21h	0084h – 0087h	DOS: gọi các hàm của DOS
22h	0088h – 008Bh	Địa chỉ kết thúc chương trình
23h	008Ch – 008Fh	Nhấn Ctrl-Break
24h	0090h – 0093h	Địa chỉ chương trình xử lý lỗi
25h	0094h – 0097h	DOS: đọc đĩa mềm / đĩa cứng
26h	0098h – 009Bh	DOS: ghi đĩa mềm / đĩa cứng
27h	009Ch – 009Fh	DOS: kết thúc chương trình và thường trú
28h – 3Fh	00A0h – 00FFh	Dành riêng cho DOS
40h	0100h – 0103h	BIOS: phục vụ đĩa mềm
41h	0104h – 0107h	Địa chỉ bảng tham số đĩa cứng 1
42h – 45h	0108h – 0117h	Dành riêng
46h	0118h – 011Bh	Địa chỉ của bảng tham số đĩa cứng 2
47h – 49h	011Ch – 0127h	Dành cho user

4Ah	0128h – 012Bh	Hẹn giờ
4Bh – 67h	012Ch – 019Fh	Dành cho user
68h – 6Fh	01A0h – 01BFh	Không dùng
70h	01C0h – 01C3h	IRQ8: đồng hồ thời gian thực
71h	01C4h – 01C7h	IRQ9
72h	01C8h – 01CBh	IRQ10
73h	01CCh – 01CFh	IRQ11
74h	01D0h – 01D3h	IRQ12
75h	01D4h – 01D7h	IRQ13: từ 80x87
76h	01D8h – 01DBh	IRQ14: đĩa cứng
77h	01DCh – 01DFh	IRQ15
78h – 7Fh	01E0h – 01FFh	Dành riêng
80h – F0h	0200h – 03C3h	Dùng cho bộ thông dịch BASIC
F1h – FFh	03C4h – 03FFh	Không sử dụng

3. Gọi ngắt và chặn ngắt

Quá trình gọi ngắt từ hợp ngữ đơn giản là thực hiện lệnh ***INT STT_ngắt*** sau khi nạp các tham số cần thiết cho ngắt. Khi thực hiện lệnh gọi ngắt, CPU sẽ tìm kiếm trong bảng vector ngắt địa chỉ của chương trình phục vụ ngắt. Người sử dụng cũng có thể xây dựng một chương trình cơ sở như các chương trình xử lý ngắt. Sau đó, các chương trình khác có thể gọi ngắt ra để sử dụng. Một chương trình có thể gọi chương trình con loại này mà không cần biết địa chỉ của nó.

Như vậy, nếu muốn thay đổi ngắt, ta có thể thay đổi nội dung trong bảng vector ngắt để chỉ đến chương trình phục vụ do người sử dụng tự xây dựng và chương trình này sẽ được thực hiện khi ngắt được gọi. Để làm điều này, ta chỉ cần tìm vị trí của vector ngắt tương ứng trong bảng và thay đổi giá trị của nó. Điều này thực hiện được do bảng vector ngắt đặt trong RAM và được nạp mỗi khi khởi động hệ thống.

Quá trình lấy và gán địa chỉ của chương trình con phục vụ ngắt có thể thực hiện thông qua ngắt 21h bằng các hàm sau:

Hàm 35h: lấy địa chỉ của ngắt

Vào: AL = số thứ tự ngắt

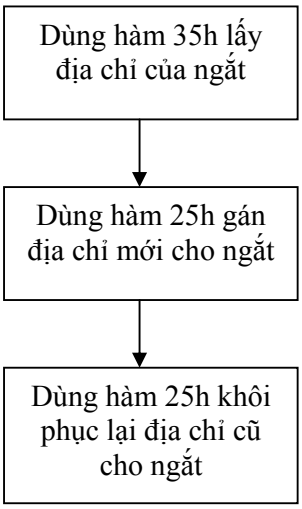
Ra: ES:BX = địa chỉ của chương trình phục vụ ngắt

Hàm 25h: gán địa chỉ của một chương trình phục vụ cho một ngắt

Vào: AL = số thứ tự ngắt, DS:DX

Ra: không có

Để thực hiện chặn một ngắt, ta cần thực hiện như sau:



Hình 2.2 – Quá trình chặn một ngắt

Một ví dụ cho quá trình chặn ngắt như sau:

```
;*****************************************************************************
;*
;*              LAB6-2.ASM - Assembler Laboratory ZMiTAC
*
;*
;*  Sample program that converts lowercase to uppercase when key pressed
*
;*
;*****************************************************************************
.MODEL SMALL
.STACK 100h
.CODE
;*****************************************************************************
;*      Variables
;*****************************************************************************
old_proc        dd 0                    ; address of original interrupt handler
dot_flag        db 0                    ; dot flag

;*****************************************************************************
;*      09h interrupt handler
;*****************************************************************************
segment_kb      EQU 40h                 ; beggining of keyboard data segment
wsk_kb          EQU 1Ch                 ; offset of pointer to keyboard buffer
kb_buf_begin    EQU 80h                 ; offset of address of begining of the
buffer
kb_buf_end      EQU 82h                 ; offset of address of end of the buffer

keys    PROC FAR

;-----------------------------------------------------------------------------
;       Calling of original interrupt handler
;-----------------------------------------------------------------------------
```

```
        int 60h
;-------------------------------------------------------------------------------
;       Prepare registers
;-------------------------------------------------------------------------------
        push ax                         ; push registers on the stack
        push bx
        push dx
        push ds
        mov ax,segment_kb               ; address of keyboard data segment to DS
        mov ds,ax

;-------------------------------------------------------------------------------
;       Read the character and check ASCII code
;-------------------------------------------------------------------------------
        mov bx,ds:[wsk_kb]              ; actual pointer to BX
        mov ax,ds:[kb_buf_begin]        ; buffer beggining to AX
        cmp bx,ax                       ; is the beggining of the buffer ?
        jne mid_buf
        mov bx,ds:[kb_buf_end]          ; last character is at the end of the
buffer
mid_buf:
        mov ax,ds:[bx-2]                ; read last character

        cmp al,'.'                      ; compare with dot
        je dot_found                    ; if dot
        cmp al,'Z'                      ; compare with 'Z'
        ja check_lowercase              ; if above check lowercase
        cmp al,'A'                      ; compare with  'A'
        jb keys_end                     ; end if less

        mov dot_flag,0                  ; uppercase - clear flag
        jmp keys_end                    ; return

check_lowercase:
        cmp al,'z'                      ; compare with 'z'
        ja keys_end                     ; end if above
        cmp al,'a'                      ; compare with 'a'
        jb keys_end                     ; end if less
        cmp dot_flag,0                  ; was dot pressed?
        je keys_end                             ; end if not

;-------------------------------------------------------------------------------
;       Change lowercase to uppercase
;-------------------------------------------------------------------------------
        sub al,'a'-'A'                  ; sub difference between cases
        mov ds:[bx-2],ax
        mov dot_flag,0                  ; uppercase - clear flag
        jmp keys_end                            ; return

dot_found:
        mov dot_flag,1                  ; set flag
        jmp keys_end                    ; return

;-------------------------------------------------------------------------------
;       Pop registers and return from interrupt
;-------------------------------------------------------------------------------
keys_end:
        pop ds
        pop dx
        pop bx
        pop ax
```

```
        iret
keys    ENDP

;*************************************************************************
;*      Main program
;*************************************************************************
;-------------------------------------------------------------------------
;       Get interrupt
;-------------------------------------------------------------------------

start   proc
        mov ah,35h                      ; function 35h - read handler address
        mov al,09h                      ; of interrupt 09h
        int 21h
        mov word ptr old_proc,bx        ; store 32-bit address
        mov word ptr old_proc+2,es      ; of original interrupt handler
        push cs
        pop ds                          ; handler code segment to DS
        mov dx,offset keys              ; offset of handler address to DX
        mov ah,25h                      ; function 25h - set new handler
        mov al,09h                      ; of interrupt 09h
        int 21h
        mov dx,word ptr old_proc+2
        mov ds,dx
        mov dx,word ptr old_proc
        mov ah,25h                      ; function 25h - set new adress
        mov al,60h                      ; of original interrupt handler
        int 21h                         ; 60h instead of 09h

;-------------------------------------------------------------------------
;       Main loop
;-------------------------------------------------------------------------
looping:mov ah,08h                      ; function 08h - read character
        int 21h                         ; ASCII code is returned in AL
        cmp al,1Bh                      ; ESC
        je ending                       ; if ESC end of the loop
        mov dl,al                       ; not ESC - move char to DL
        mov ah,02h                      ; function 02h - display character
        int 21h                         ; ASCII code of char in DL
        jmp looping

;-------------------------------------------------------------------------
ending: mov dx,word ptr old_proc+2
        mov ds,dx
        mov dx,word ptr old_proc
        mov ah,25h                      ; function 25h - set old handler
        mov al,09h                      ; of interrupt 09h
        int 21h

        mov ah,4Ch                      ; end of the program
        int 21h
start   endp
END     start
```

4. Tạo và bắt các sự kiện

Trong hệ điều hành Windows, hook (câu móc) là cơ chế cho phép một hàm chặn một sự kiện (thông điệp, chuột, phím nhấn) trước khi đưa đến đối tượng cần xử lý. Hàm này cho phép thay đổi hoặc thậm chí cấm sự kiện xảy ra. Chúng được gọi là hàm lọc (filter) và được

phân loại dựa theo loại sự kiện bị chặn. Để gọi được hàm lọc, ta cần phải thực hiện quá trình gắn (attach) vào quá trình câu móc (như câu móc bàn phím). Việc gắn một hay nhiều hàm lọc vào một quá trình câu móc được gọi là thiết lập (setting) một quá trình câu móc.

Nếu một quá trình câu móc có nhiều hơn một hàm lọc, Windows sẽ duy trì một chuỗi các hàm lọc trong đó hàm được cài đặt vào gần nhất sẽ nằm ở đầu chuỗi và hàm cài đặt lâu nhất sẽ nằm ở cuối chuỗi. Nếu sự kiện xảy ra làm khởi động quá trình câu móc, Windows sẽ gọi hàm lọc đầu tiên trong chuỗi. Quá trình câu móc vào một sự kiện được sử dụng bằng hàm **SetWindowsHookEx** và hàm **UnhookWindowsHookEx** dùng để xóa bỏ hàm lọc khỏi quá trình.

Cơ chế câu móc cung cấp các khả năng mạnh mẽ cho một ứng dụng Windows. Các ứng dụng này có thể dùng quá trình câu móc để:

- Xử lý và thay đổi các thông điệp gởi đến các dialog box, message box, scroll bar và menu của một ứng dụng (WH_MSGFILTER).
- Xử lý và thay đổi các thông điệp gởi đến các dialog box, message box, scroll bar và menu của hệ thống (WH_SYSMSGFILTER).
- Xử lý và thay đổi các thông điệp của hệ thống bất cứ khi nào hàm **GetMessage** hay **PeekMessage** được gọi (WH_GETMESSAGE).
- Xử lý và thay đổi các thông điệp của hệ thống bất cứ khi nào hàm **SendMessage** được gọi (WH_CALLWNDPROC).
- Ghi hay thực hiện lại các sự kiện bàn phím và chuột (WH_JOURNALRECORD, WH_JOURNALPLAYBACK).
- Xử lý, sửa đổi hay cấm sự kiện chuột (WH_MOUSE).
- Xử lý, sửa đổi hay cấm sự kiện bàn phím (WH_KEYBOARD).
- Đáp ứng với các hoạt động nào đó của hệ thống, có khả năng phát triển CBT (computer-based training) cho ứng dụng (WH_CBT).
- Cấm các hàm lọc khác (WH_DEBUG).

Các ứng dụng thường dùng quá trình câu móc để:

- Dùng phím F1 để hỗ trợ cho menu, dialog box và message box (WH_MSGFILTER).
- Lưu lại quá trình thực hiện khi nhấn phím hay chuột (thường dùng cho macro). Ví dụ như Windows Recorder sử dụng hook để hỗ trợ chức năng record và playback (WH_JOURNALRECORD, WH_JOURNALPLAYBACK).
- Quản lý thông điệp để xác nhận thông điệp được gởi tới cửa sổ hay được tạo ra (WH_GETMESAGE, WH_CALLWNDPROC).
- Mô phỏng ngõ vào chuột và bàn phím (WH_JOURNALPLAYBACK). Quá trình câu móc là phương pháp tin cậy để thực hiện hoạt động này. Nếu ta thực hiện mô phỏng bằng cách gởi thông điệp, Windows sẽ không thực hiện cập nhật trạng thái của bàn phím hay chuột dẫn đến các hoạt động không mong muốn. Nếu quá trình câu móc thực hiện điều này, nó sẽ được xử lý giống như sự kiện chuột hay bàn

phím xảy ra. Ví dụ như Microsoft Excel dùng hook để thực hiện macro SENDKEYS.

- Cung cấp khả năng CBT cho ứng dụng thực hiện trên môi trường Windows (WH_CBT) làm cho quá trình phát triển ứng dụng CBT dễ dàng hơn.

Phạm vi sử dụng: Một trong những đặc trưng của Win32 Hook là cho phép chỉ định quá trình câu móc là hệ thống hay ở dạng luồng (thread). Hook hệ thống cho phép tác động đến các cửa sổ khác trong hệ thống còn hook luồng chỉ cho phép tác động đến cửa sổ hiện hành.

Cách thức sử dụng quá trình câu móc:

Để sử dụng quá trình câu móc, ta cần phải biết:

- Làm thế nào dùng hàm câu móc của Windows để thêm vào hay xóa bỏ một hàm lọc trong chuỗi hàm xử lý của một quá trình câu móc.
- Cần phải thực hiện các hoạt động gì để cài đặt một hàm lọc.
- Có thể thực hiện được hàm câu móc nào và chúng có thể làm được gì, gởi những thông tin nào.

4.1. Hàm câu móc của Windows

Các ứng dụng trên nền Windows sử dụng các hàm **SetWindowsHookEx**, **UnhookWindowsHookEx** và **CallNextHookEx** để quản lý chuỗi hàm lọc trong một quá trình câu móc. Trước phiên bản 3.1, Windows thực hiện quản lý bằng các hàm **SetWindowsHook**, **UnhookWindowsHook** và **DefHookProc**. Mặc dù các hàm này cũng có khả năng thực hiện được trên nền Win32 nhưng sẽ có một số đặc trưng không sử dụng được như các phiên bản mới (**Ex**).

4.1.1. Hàm SetWindowsHookEx

Dùng để thêm một hàm lọc vào một quá trình câu móc.

Khai báo:

```
Public Declare Function SetWindowsHook Lib "user32" Alias "SetWindowsHookA" (ByVal nFilterType As Long, ByVal pfnFilterProc As Long) As Long
Public Declare Function SetWindowsHookEx Lib "user32" Alias "SetWindowsHookExA" (ByVal idHook As Long, ByVal lpfn As Long, ByVal hmod As Long, ByVal dwThreadId As Long) As Long
```

Hàm SetWindowsHookEx gồm có 4 tham số:

- **idHook**: xác định loại hàm câu móc sẽ cài đặt. Thông số này gồm các giá trị sau:

WH_KEYBOARD: cài đặt hàm câu móc quản lý thông điệp gởi đi khi nhấn phím (ngoại trừ tổ hợp Ctrl – Alt – Del).

WH_MOUSE: cài đặt hàm câu móc quản lý thông điệp khi điều khiển chuột.

WH_CALLWNDPROC: cài đặt hàm câu móc quản lý thông điệp trước khi hệ thống gởi đến cửa sổ, chỉ cho phép xử lý thông điệp mà không được thay đổi thông điệp.

WH_CALLWNDPROCRET: cài đặt hàm câu móc quản lý thông điệp sau khi cửa sổ đã xử lý. Loại này cho phép thay đổi giá trị trả về của thông điệp.

WH_MSGFILTER: cài đặt hàm câu móc quản lý các thông điệp được tạo ra giống như có một sự kiện của dialog box, message box, menu hay scroll bar.

WH_GETMESSAGE: cài đặt hàm câu móc quản lý các thông điệp được gởi tới hàng đợi.

WH_CBT: cài đặt hàm câu móc để nhận thông báo từ ứng dụng CBT.

WH_DEBUG: cài đặt hàm câu móc để gỡ rối một hàm câu móc khác.

WH_FOREGROUNDIDLE: cài đặt hàm câu móc trong đó hàm này được gọi khi luồng (thread) foreground của ứng dụng rảnh (idle). Quá trình này thường sử dụng để thực thi các tác vụ có độ ưu tiên thấp khi luồng ưu tiên rảnh.

WH_JOURNALPLAYBACK: cài đặt hàm câu móc để gởi các thông điệp đã được lưu bằng hàm câu móc WH_JOURNALRECORD.

WH_JOURNALRECORD: cài đặt hàm câu móc lưu lại các thông điệp đã gởi đến hàng đợi.

WH_KEYBOARD_LL: cài đặt hàm câu móc quản lý sự kiện bàn phím ở mức thấp (dùng cho Windows NT/2000/XP).

WH_MOUSE_LL: cài đặt hàm câu móc quản lý sự kiện chuột ở mức thấp (dùng cho Windows NT/2000/XP).

WH_SHELL: cài đặt hàm câu móc cho một ứng dụng shell.

WH_SYSMSGFILTER: cài đặt hàm câu móc quản lý các thông điệp được tạo ra giống như có một sự kiện của dialog box, message box, menu hay scroll bar. Hàm này quản lý cho tất cả ứng dụng trong cùng một desktop.

- **lpfn**:

Con trỏ chỉ đến địa chỉ của hàm lọc. Nếu tham số *dwThreadId* = 0 hay chỉ đến một luồng được tạo bởi một tiến trình (process) khác, tham số *lpfn* phải chỉ đến một hàm câu móc trong một thư viện liên kết động (DLL). Ngược lại, *lpfn* chỉ đến hàm câu móc chứa trong bản thân tiến trình hiện hành.

- **hMod**:

handle chỉ đến DLL chứa hàm xử lý xác định bằng tham số *lpfn*. Tham số *hMod* phải đặt là NULL nếu hàm câu móc nằm trong tiến trình hiện hành

- **dwThreadId**:

Xác định ID của luồng thực hiện quá trình câu móc. Nếu *dwThreadId* = 0, hàm câu móc sẽ tác động đến tất cả các luồng. Ứng dụng có thể dùng hàm **GetCurrentThreadId** để xác định ID của luồng hiện hành.

Phạm vi thực hiện của hàm câu móc mô tả như sau:

Hook	Phạm vi
WH_CALLWNDPROC	Luồng hay hệ thống
WH_CBT	Luồng hay hệ thống
WH_DEBUG	Luồng hay hệ thống
WH_GETMESSAGE	Luồng hay hệ thống
WH_JOURNALRECORD	Hệ thống
WH_JOURNALPLAYBACK	Hệ thống
WH_FOREGROUNDIDLE	Luồng hay hệ thống
WH_SHELL	Luồng hay hệ thống
WH_KEYBOARD	Luồng hay hệ thống
WH_MOUSE	Luồng hay hệ thống
WH_MSGFILTER	Luồng hay hệ thống
WH_SYSMSGFILTER	Hệ thống

Hàm **SetWindowsHookEx** trả về handle của quá trình câu móc đã cài đặt và trả về NULL nếu quá trình cài đặt không thành công. Handle này được dùng để xóa quá trình câu móc khi sử dụng hàm **UnhookWindowsHookEx**. Các thông báo lỗi khi quá trình câu móc không thành công là:

- ERROR_INVALID_HOOK_FILTER: mã câu móc sai
- ERROR_INVALID_FILTER_PROC: hàm lọc sai
- ERROR_HOOK_NEEDS_HMOD: một quá trình câu móc toàn cục sử dụng tham số hMod = NULL hay chỉ đến một luồng không tồn tại.
- ERROR_GLOBAL_ONLY_HOOK: một quá trình câu móc chỉ dùng được cho hệ thống nhưng được cài đặt cho một luồng xác định.
- ERROR_INVALID_PARAMETER: ID của luồng sai.
- ERROR_JOURNAL_HOOK_SET: Cài đặt thêm một quá trình câu móc dạng nhật ký (WH_JOURNALRECORD và WH_JOURNALPLAYBACK) trong khi một quá trình dạng này đang tồn tại (tại một thời điểm chỉ cho phép một quá trình dạng nhật ký).
- ERROR_MOD_NOT_FOUND: Tham số hMod chỉ đến một hàm không xác định được.
- Khác: không cho phép do bảo mật của hệ thống hay bộ nhớ tràn.

4.1.2. Hàm UnhookWindowsHookEx:

Dùng để xoá một hàm lọc ra khỏi chuỗi xử lý một quá trình câu móc. Hàm này lấy handle của quá trình câu móc trả về từ lệnh gọi hàm SetWindowsHookEx và luôn trả về giá trị TRUE.

Khai báo:

```
Public Declare Function UnhookWindowsHook Lib "user32" Alias "UnhookWindowsHook" (ByVal nCode As Long, ByVal pfnFilterProc As Long) As Long
Public Declare Function UnhookWindowsHookEx Lib "user32" Alias "UnhookWindowsHookEx" (ByVal hHook As Long) As Long
```

4.1.3. Hàm CallNextHookEx:

Dùng để chuyển thông tin câu móc đến hàm câu móc kế tiếp trong chuỗi xử lý.

```
Declare Function CallNextHookEx Lib "user32" (ByVal hHook As Long, ByVal ncode As Long, ByVal wParam As Long, lParam As Any) As Long
```

- **hHook**: handle của quá trình câu móc, là giá trị trả về từ lệnh gọi hàm SetWindowsHookEx. Thông thường Windows bỏ qua giá trị này.
- **nCode**: mã của quá trình câu móc, hàm câu móc dùng mã này để xác định phương pháp xử lý thông tin.
- **wParam**: xác định tham số được xử lý bởi hàm câu móc.
- **lParam**: giống như wParam.

Khi một quá trình câu móc khởi động, Windows gọi hàm đầu tiên trong chuỗi hàm lọc và kết thúc quản lý quá trình, các hàm lọc phía sau sẽ không xử lý. Để thực hiện các hàm ở phía sau trong chuỗi hàm, Windows cung cấp hàm CallNextHookEx cho phép gọi một hàm kế tiếp trong chuỗi hàm lọc. Như vậy, nếu một hàm lọc nào đó không thực hiện hàm CallNextHookEx thì các hàm lọc ở phía sau sẽ không thực hiện.

Một ví dụ sử dụng các hàm xử lý câu móc như sau:

```
'Ch•a trong m•t file module
Public Const WH_KEYBOARD = 2
Public Const VK_SHIFT = &H10
Public Const VK_CONTROL = &H11
Public Const VK_MENU = &H12
Declare Function CallNextHookEx Lib "user32" (ByVal hHook As Long, ByVal ncode As Long, ByVal wParam As Long, lParam As Any) As Long
Declare Function GetKeyState Lib "user32" (ByVal nVirtKey As Long) As Integer 'Xác ••nh tr•ng thái c•a m•t phím (Bit15)
```

```
Declare Function SetWindowsHookEx Lib "user32" Alias "SetWindowsHookExA" (ByVal idHook As Long, ByVal lpfn As Long, ByVal hmod As Long, ByVal dwThreadId As Long) As Long
Declare Function UnhookWindowsHookEx Lib "user32" (ByVal hHook As Long) As Long
Public hHook As Long
Public Function KeyboardProc(ByVal idHook As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
If idHook < 0 Then
'G•i hàm x• lý k• ti•p
KeyboardProc = CallNextHookEx(hHook, idHook, wParam, ByVal lParam)
Else
'N•u nh•n Shift-C
If (GetKeyState(VK_SHIFT) And &H8000) And wParam = Asc("C") Then
'thì hi•n th• k•t qu•
Form1.Print "Shift-C pressed ..."
End If
If (GetKeyState(VK_CONTROL) And &H8000) And wParam = Asc("C") Then
Form1.Print "Ctrl-C pressed ..."
End If
If (GetKeyState(VK_MENU) And &H8000) And wParam = Asc("C") Then
Form1.Print "Alt-C pressed ..."
End If
'G•i hàm x• lý k• ti•p
KeyboardProc = CallNextHookEx(hHook, idHook, wParam, ByVal lParam)
End If
End Function
-----------------------------------------------
'Ch•a trong form
Private Sub Form_Load()
'••t quá trình câu móc
hHook = SetWindowsHookEx(WH_KEYBOARD, AddressOf KeyboardProc, App.hInstance, App.ThreadID)
End Sub
Private Sub Form_Unload(Cancel As Integer)
```

```
'Xoá quá trình câu móc
UnhookWindowsHookEx hHook
End Sub
```

4.2. Hàm lọc

Hàm lọc thường có dạng như sau:

```
Function FilterFunc (ByVal nCode As Integer, ByVal wParam As Long, ByVal lParam As Long)
```

Hàm lọc nhận 3 tham số:

- nCode: mã của quá trình câu móc, là một số nguyên xác định hàm lọc, ví dụ như loại sự kiện làm khởi động quá trình câu móc. Mã này được xác định khi hàm lọc xử lý sự kiện hay gọi hàm **DefHookProc**. Nếu mã câu móc < 0 thì hàm lọc sẽ không xử lý sự kiện mà sẽ gọi hàm **DefHookProc** để truyền 3 tham số còn lại cho hàm lọc kế tiếp trong chuỗi hàm lọc bằng hàm **CallNextHookEx**.
- Tham số thứ hai wParam và thứ ba lParam chứa các thông tin cần thiết cho hàm lọc. Mỗi quá trình câu móc dùng các giá trị wParam và lParam khác nhau. Ví dụ như, quá trình câu móc bàn phím WH_KEYBOARD chứa mã phím nhấn trong wParam và trạng thái bàn phím trong lParam. Hay quá trình câu móc WH_MSGFILTER chứa giá trị NULL trong wParam và một con trỏ chỉ đến thông điệp chứa trong lParam.

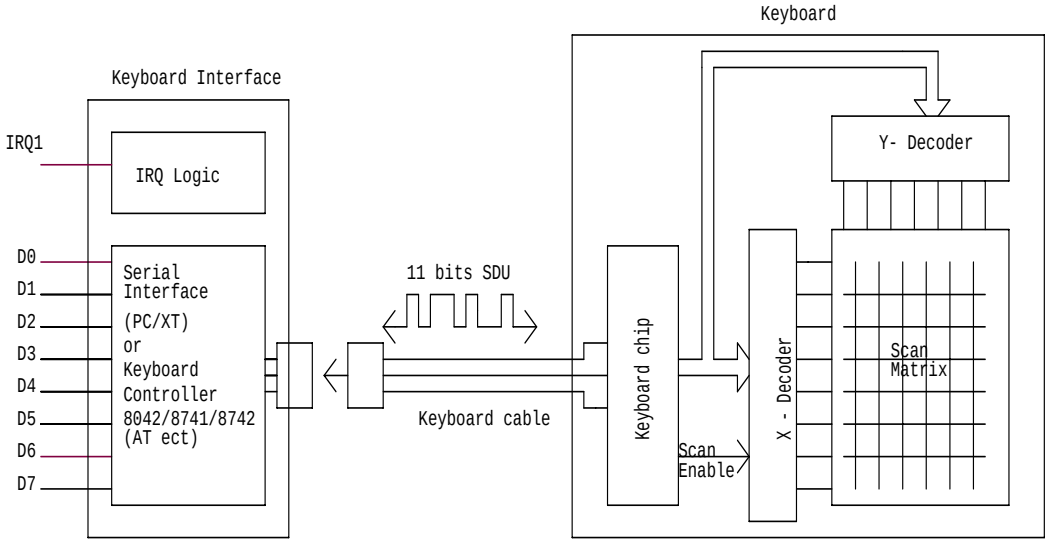
Hàm lọc dùng trong DLL:

Đối với các quá trình câu móc cục bộ, hàm lọc có thể đặt ngay trong mã lệnh của ứng dụng nhưng đối với các quá trình câu móc hệ thống, hàm lọc phải được đặt trong một DLL. Chỉ có quá trình câu móc dạng nhật ký (WH_JOURNALRECORD và WH_JOURNALPLAYBACK) là ngoại lệ. Hàm lọc của quá trình câu móc hệ thống phải chia sẻ dữ liệu cho tiến trình thực hiện quá trình câu móc. Các biến toàn cục sử dụng trong DLL phải được xác định rõ hay phải đặt trong vùng dữ liệu chia sẻ.

Chương 3
GIAO TIẾP THIẾT BỊ CHUẨN

1. Giao tiếp bàn phím

1.1. Nguyên lý hoạt động



Hình 3.1 - Sơ đồ nguyên lý và các ghép nối của bàn phím

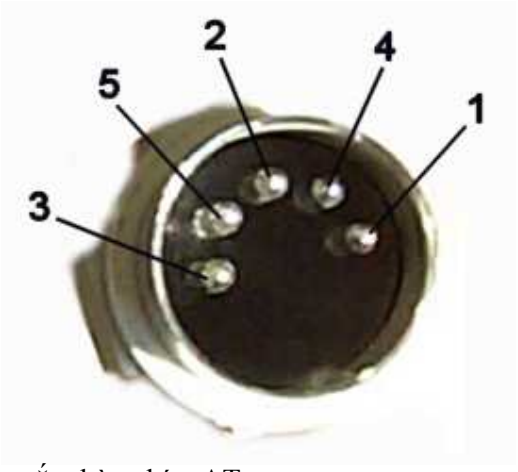
Chip xử lý bàn phím liên tục kiểm tra trạng thái của ma trận quét (scan matrix) để xác định công tắc tại các tọa độ X, Y đang được đóng hay mở và ghi một mã tương ứng vào bộ đệm bên trong bàn phím. Sau đó mã này sẽ được truyền nối tiếp tới mạch ghép nối bàn phím trong PC. Cấu trúc của SDU (Serial Data Unit) cho việc truyền số liệu:

0 10

STRT	DB_0	DB_1	DB_2	DB_3	DB_4	DB_5	DB_6	DB_7	PAR	STOP

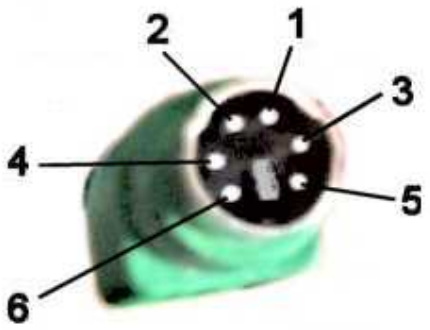
STRT: bit start (luôn bằng 0)
DB_0 - DB_7: bit số liệu từ 0 đến 7.
PAR: bit parity (luôn lẻ)
STOP: bit stop (luôn bằng 1).

Chân 1: clock
Chân 2: dữ liệu
Chân 3: Reset
Chân 4: GND
Chân 5: Vcc



Hình 3.2 – Đầu cắm bàn phím AT

Chân 1: dữ liệu
Chân 2: không dùng
Chân 3: GND
Chân 4: Vcc
Chân 5: clock
Chân 6: không dùng



Hình 3.3 – Đầu cắm bàn phím PS/2

Mỗi phím nhấn sẽ được gán cho 1 mã quét (scan code) gồm 1 byte. Nếu 1 phím được nhấn thì bàn phím phát ra 1 mã make code tương ứng với mã quét truyền tới mạch ghép nối bàn phím của PC. Ngắt cứng INT 09h được phát ra qua IRQ1.

Chương trình xử lý ngắt sẽ xử lý mã này tuỳ theo phím SHIFT có được nhấn hay không. Ví dụ: nhấn phím SHIFT trước, không rời tay và sau đó nhấn ‘C’:

make code được truyền - 42(SHIFT) - 46 (‘C’).

Nếu rời tay nhấn phím SHIFT thì bàn phím sẽ phát ra break code và mã này được truyền như make code. Mã này giống như mã quét nhưng bit 7 được đặt lên 1, do vậy nó tương đương với make code cộng với 128. Tuỳ theo break code, chương trình con xử lý ngắt sẽ xác định trạng thái nhấn hay rời của các phím. Thí dụ, phím SHIFT và ‘C’ được rời theo thứ tự ngược lại với thí dụ trên:

break code được truyền 174 (bằng 46 cộng 128 tương ứng với ‘C’) và 170 (bằng 42 cộng 128 tương ứng với SHIFT).

Phần cứng và phần mềm xử lý bàn phím còn giải quyết các vấn đề vật lý sau:

- Nhấn và nhả phím nhưng không được phát hiện.

- Khử nhiễu rung cơ khí và phân biệt 1 phím được nhấn nhiều lần hay được nhấn chỉ 1 lần nhưng được giữ trong một khoảng thời gian dài.

1.2. Lập trình giao tiếp qua các cổng

Bàn phím cũng là một thiết bị ngoại vi nên về nguyên tắc có thể truy xuất nó qua các cổng vào ra.

❖ ***Các thanh ghi và các port:***

Sử dụng 2 địa chỉ port 60h và 64h có thể truy xuất bộ đệm vào, bộ đệm ra và thanh ghi điều khiển của bàn phím.

Port	Thanh ghi	R/W
60h	Đệm ngõ ra	R
60h	Đệm ngõ vào	W
64h	Thanh ghi điều khiển	W
64h	Thanh ghi trạng thái	R

Thanh ghi trạng thái xác định trạng thái hiện tại của bộ điều khiển bàn phím. Thanh ghi này chỉ đọc (read only) và đọc bằng lệnh IN tại port 64h.

7							0
PARE	TIM	AUXB	KEYL	C/D	SYSF	INPB	OUTB

PARE: Lỗi chẵn lẻ của byte cuối cùng được vào từ bàn phím; 1 = có lỗi chẵn lẻ, 0 = không có.

TIM: Lỗi quá thời gian (time-out); 1 = có lỗi, 0 = không có.

AUXB: Đệm ra cho thiết bị phụ (chỉ có ở máy PS/2); 1 = giữ số liệu cho thiết bị, 0 = giữ số liệu cho bàn phím.

KEYL: Trạng thái khóa bàn phím; 1 = không khóa, 0 = khóa.

C/D: Lệnh/dữ liệu; 1 = Ghi qua port 64h, 0 = Ghi qua port 60h.

SYSF: cờ hệ thống; 1 = tự kiểm tra thành công, 0 = reset khi cấp điện

INPB: Trạng thái đệm vào; 1 = dữ liệu CPU trong bộ đệm vào, 0 = đệm vào rỗng.

OUTB: Trạng thái đệm ra; 1 = dữ liệu bộ điều khiển bàn phím trong bộ đệm ra, 0 = đệm ra rỗng.

Thanh ghi điều khiển

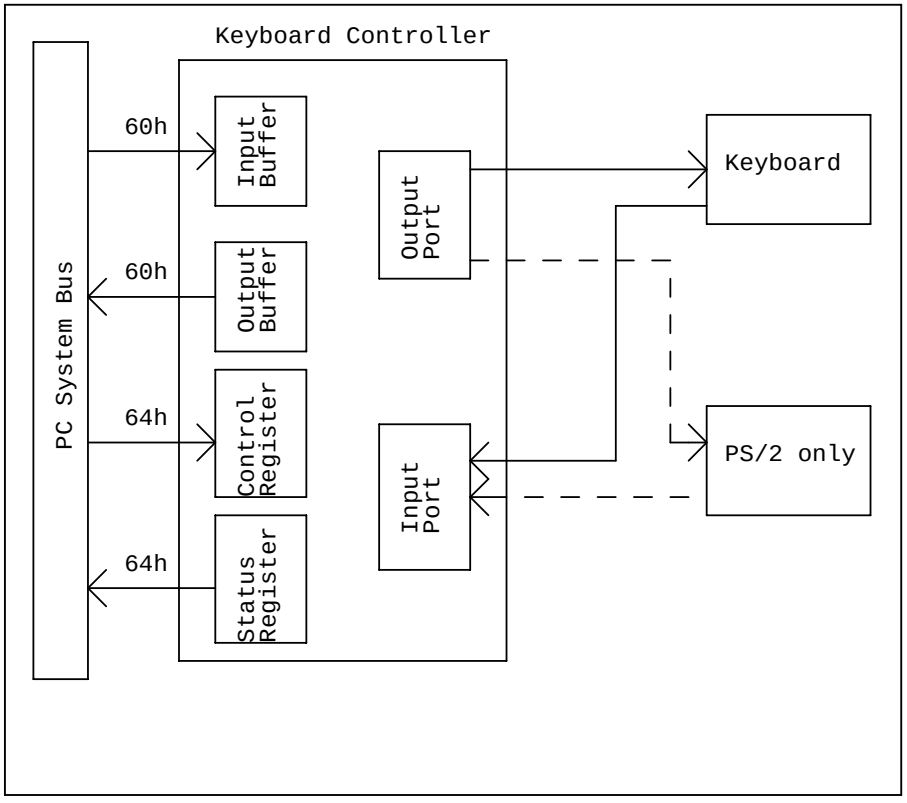
Các lệnh cho bộ điều khiển bàn phím:

Mã	Mô tả
A7h	Cấm thiết bị phụ
A8h	Cho phép thiết bị phụ
A9h	Kiểm tra giao tiếp thiết bị phụ và lưu mã kiểm tra vào bộ đệm ra 00h: không lỗi 01h: CLK ở mức thấp 02h: CLK ở mức cao 03h: DATA ở mức thấp

	04h: DATA ở mức cao FFh: lỗi khác
AAh	Tự kiểm tra (ghi 55h vào bộ đệm ra nếu không lỗi
ABh	Kiểm tra giao tiếp bàn phím và lưu mã kiểm tra vào bộ đệm ra
ADh	Cấm bàn phím
AEh	Cho phép bàn phím
C0h	Đọc cổng vào và truyền dữ liệu đến bộ đệm ra
C1h	Đọc các bit 3 – 0 của cổng vào và truyền đến các bit 3- 0 của thanh ghi trạng thái cho đến khi INPB = 1
C2h	Đọc các bit 7 – 4 của cổng vào và truyền đến các bit 7- 4 của thanh ghi trạng thái cho đến khi INPB = 1
D0h	Đọc cổng ra
D1h	Ghi cổng ra
D2h	Ghi vào bộ đệm ra và xoá AUXB
D3h	Ghi vào bộ đệm ra và set AUXB
D4h	Ghi byte dữ liệu tiếp theo vào thiết bị phụ

Khóa bàn phím:

```
Start:
IN AL, 64h          ; đọc byte trạng thái
TEST AL, 02h        ; kiểm tra bộ đệm có đầy hay không
JNZ start
OUT 64h, 0ADh       ; khóa bàn phím
```



Hình 3.4 - Bộ điều khiển bàn phím

❖ ***Các lệnh cho bàn phím:***

Mã	Lệnh	Mô tả
EDh	Bật/tắt LED	Bật/tắt các đèn led của bàn phím
EEh	Echo	Trả về byte EEh
F0h	Đặt/nhận dạng mã quét	Đặt 1 trong 3 tập mã quét và nhận diện các mã quét tập mã quét hiện tại.
F2h	Nhận diện bàn phím	Nhận diện ACK = AT, ACK+abh+41h=MF II.
F3h	Đặt tốc độ lặp lại/trễ	Đặt tốc độ lặp lại và thời gian trễ của bàn phím
F4h	Enable	Cho phép bàn phím hoạt động
F5h	Chuẩn/không cho phép	Đặt giá trị chuẩn và cấm bàn phím.
F6h	Chuẩn/cho phép	Đặt giá trị chuẩn và cho phép bàn phím.
FEh	Resend	Bàn phím truyền ký tự cuối cùng một lần nữa tới bộ điều khiển bàn phím
FFh	Reset	Chạy reset bên trong bàn phím

Thí dụ: lệnh bật đèn led cho phím NUMCLOCK, tắt tất cả các đèn khác.

```
MOV AL,0EDh
OUT  60H, AL
WAIT:
IN  AL, 64H          ; đọc thanh ghi trạng thái
JNZ   WAIT
MOV AL,02h
OUT  60H, AL         ; bật đèn cho numclock
```

Cấu trúc của byte chỉ thị như sau:

7					2	1	0
0	0	0	0	0	CPL	NUM	SCR

CPL: 1 = bật đèn Caps Lock; 0 = tắt
NUM: 1 = bật đèn Num Lock; 0 = tắt
SCR: 1 = bật đèn Scroll Lock; 0 = tắt

1.3. Lập trình giao tiếp qua các hàm của DOS, BIOS

BIOS ghi các ký tự do việc nhấn các phím vào bộ đệm tạm thời được gọi là bộ đệm bàn phím (keyboard buffer), có địa chỉ 40h:1Eh, gồm 32 byte và kết thúc ở địa chỉ 40h:3Dh. Mỗi ký tự được lưu trữ bằng 2 byte, byte cao là mã quét, và byte thấp là mã ASCII. Chương trình xử lý ngắt sẽ xác định mã ASCII từ mã quét bằng bảng biến đổi và ghi cả 2 mã vào bộ đệm bàn phím. Bộ đệm bàn phím được tổ chức như bộ đệm vòng (ring buffer) và được quản lý bởi 2 con trỏ. Các giá trị con trỏ được lưu trữ trong vùng dữ liệu của BIOS ở địa chỉ 40h:1Ah và 40h:1Ch. Con trỏ ghi (40h:1Ch) cho biết vị trí còn trống kế tiếp để ghi ký tự nhập, con trỏ đọc (40h:1Ah) cho biết vị trí ký tự đầu tiên sẽ đọc. Từ đó, bộ đệm bàn phím rỗng khi con trỏ ghi và con trỏ đọc trùng nhau → bộ đệm chỉ chứa được 15 ký tự.

Các hàm của ngắt 16h:

Hàm 0h - đọc ký tự từ bàn phím, nếu không nhấn thì sẽ chờ

Ra: AH = scancode, AL = mã ASCII. Nếu phím nhấn là các phím đặc biệt thì AL = 0

Hàm 1h - ZF = 1 nếu không có ký tự trong bộ đệm. Giá trị trả về giống như hàm 00h nhưng không xoá ký tự ra khỏi bộ đệm

Hàm 2h - Trả về trạng thái của các phím, kết quả chứa trong AL

7	6	5	4	3	2	1	0
INS	CAPS LOCK	NUM LOCK	SCROLL LOCK	ALT	CTRL	LEFT SHIFT	RIGHT SHIFT

Hàm 10h - Giống hàm 00h nhưng trả về mã mở rộng

Hàm 11h - Giống hàm 01h nhưng trả về mã mở rộng

Hàm 12h - Giống hàm 02h nhưng AH chứa thêm các thông tin

7	6	5	4	3	2	1	0
SYS REQ	CAPS LOCK	NUM LOCK	SCROLL LOCK	RIGHT ALT	RIGHT CTRL	LEFT ALT	LEFT CTRL

Các thí dụ:

- Giả sử phím ‘c’ đã được nhấn.

```
MOV AH,00h
INT 16h
```

Kết quả: AH = 2Eh (mã quét cho phím ‘a’); AL = 63h (ASCII cho ‘c’)

- Giả sử phím ‘HOME’ đã được nhấn.

```
MOV AH,00h
INT 16h
```

Kết quả: AH = 47h (mã quét cho phím ‘HOME’)
AL = 0 (các phím chức năng và điều khiển không có mã ASCII)

- Giả sử phím ‘HOME’ đã được nhấn.

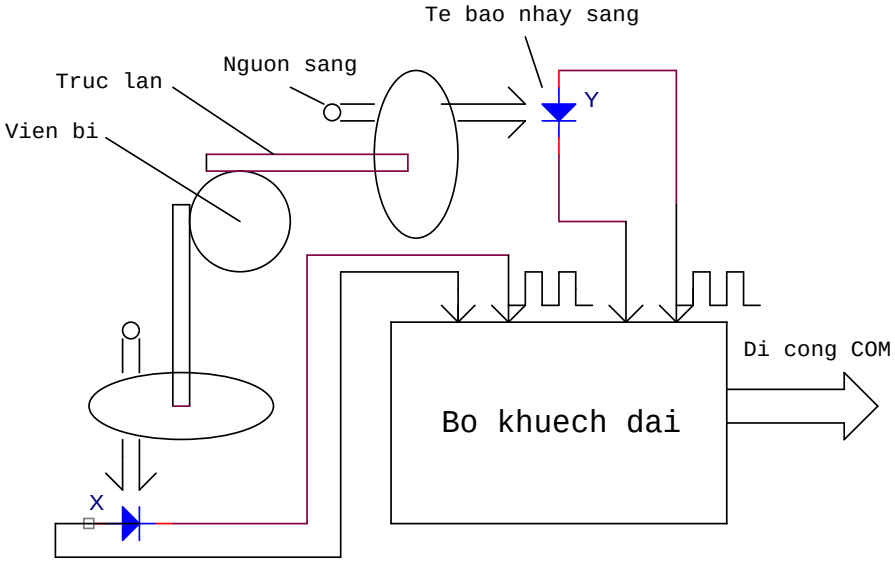
```
MOV AH,10h
INT 16h
```

Kết quả: AH = 47h (mã quét cho phím ‘HOME’)
AL = E0h

2. Giao tiếp chuột

2.1. Cấu tạo

Cấu tạo của chuột rất đơn giản, phần trung tâm là 1 viên bi thép được phủ keo hoặc nhựa được quay khi dịch chuyển chuột. Chuyển động này được truyền tới 2 thanh nhỏ được đặt vuông góc với nhau. Các thanh này sẽ biến chuyển động của chuột theo 2 hướng X,Y thành sự quay tương ứng của 2 đĩa gắn với chúng. Trên 2 đĩa có những lỗ nhỏ liên tục đóng và ngắt 2 chùm sáng tới các sensor nhạy sáng để tạo ra các xung điện. Số các xung điện tỷ lệ với lượng chuyển động của chuột theo các hướng X,Y và số xung trên 1 sec biểu hiện tốc độ của chuyển động chuột. Kèm theo đó có 2 hay 3 phím bấm.



Hình 3.5 - Sơ đồ cấu tạo của chuột

2.2. Mạch ghép nối và chương trình điều khiển chuột

Hầu hết chuột được nối với PC qua cổng nối tiếp, qua đó chuột cũng được cấp nguồn nuôi từ PC. Khi dịch chuyển hoặc nhấn, nhả các phím chuột, nó sẽ phát ra một gói dữ liệu tới mạch giao tiếp và mạch sẽ phát ra 1 ngắt. Phần mềm điều khiển chuột làm các nhiệm vụ: chuyển ngắt tới mạch giao tiếp nối tiếp xác định, đọc dữ liệu và cập nhật các giá trị bên trong liên quan tới trạng thái của bàn phím cũng như vị trí của chuột. Hơn nữa, nó còn cung cấp 1 giao tiếp mềm qua ngắt 33h để định các giá trị bên trong này cũng như làm dịch chuyển con trỏ chuột trên màn hình tương ứng với vị trí của chuột.

Có thể chọn kiểu con trỏ chuột cứng hoặc mềm trong chế độ văn bản hay con trỏ chuột đồ hoạ trong chế độ đồ họa. Các hàm 09h và 0Ah trong ngắt 33h cho phép định nghĩa loại và dạng con trỏ chuột.

2.3. Chương trình với con trỏ

Ngắt 33h cho phép xác định vị trí, số lần click chuột và hình dạng con trỏ (số thứ tự hàm chứa trong AX).

Hàm	Ý nghĩa	Tham số
0	Reset chuột	Ra: AX = 0: nếu có, = 1: không BX = số nút nhấn
1	Hiển thị con trỏ	
2	Ẩn con trỏ	
3	Nhận vị trí con trỏ và trạng thái nút	Ra: BX: trạng thái nút (D0: nút trái, D1: nút phải, D2: nút giữa) (= 0: nhả, = 1: nhấn) CX: vị trí ngang DX: vị trí dọc
4	Đặt vị trí con trỏ	Vào: CX: vị trí ngang DX: vị trí dọc
5	Trạng thái nút và số lần nhấn từ khi gọi	Vào: BX = nút kiểm tra (=0: trái, =1: phải) Ra: AX = trạng thái nút BX = số lần nhấn CX: vị trí ngang DX: vị trí dọc lần nhấn cuối
6	Giống hàm 05h nhưng kiểm tra số lần nhả	
7	Giới hạn dịch chuyển ngang của con trỏ	Vào: CX = cột trái DX = cột phải
8	Giới hạn dịch chuyển dọc của con trỏ	Vào: CX = dòng dưới DX = dòng trên
9	Xác định hình dạng con trỏ đồ hoạ	Vào: BX = vị trí ngang CX = vị trí dọc ES:DX: địa chỉ mặt nạ màn hình và con trỏ
A	Xác định hình dạng con trỏ văn bản	Vào: BX = 0: con trỏ phần mềm CX = mặt nạ màn hình DX = mặt nạ con trỏ BX = 1: con trỏ phần cứng CX = dòng bắt đầu DX = dòng kết thúc

Chú ý rằng toạ độ con trỏ xác định theo pixel với độ phân giải 640x200 trong khi chế độ văn bản sử dụng toạ độ ký tự 80x25 nên để chuyển sang toạ độ ký tự thì phải chia cho 8. Con trỏ chuột hiển thị trên màn hình đồ hoạ bằng cách thực hiện:

Từ mới = (từ cũ AND mặt nạ màn hình) XOR mặt nạ con trỏ

Nếu ta đặt mặt nạ màn hình là 0 thì ký tự màn hình tại đó sẽ bị xoá.

VD: Con trỏ chuột mềm nhấp nháy và chứa ký tự 'A'

```
MOV AH,0Ah
MOV BX,0
MOV CX,0 ; m•t n• màn hình = 0
MOV DH,8Bh; = 10001011b → màu n•n Gray, màu ký t• Cyan
MOV DL,'A'
INT 33h
```

VD: Con trỏ chuột cứng có các đường quét 3 và 8

```
MOV AH,0Ah
MOV BX,1
MOV CX,03h
MOV DX,08h
INT 33h
```

3. Giao tiếp màn hình

3.1. Card màn hình

Để hiện các hình ảnh, ký tự, hay hình vẽ trên màn hình, PC phải thông qua mạch ghép nối màn hình (graphics adapter). Board mạch này thường được cắm trên khe cắm mở rộng của PC.

Trong chế độ văn bản (text mode), các ký tự được xác định bởi mã ASCII, trong đó có cả các thông tin về thuộc tính của ký tự, thí dụ ký tự được hiện theo cách nhấp nháy hay đảo màu đen trắng ….ROM ký tự (character rom) lưu trữ các hình mẫu điểm ảnh của các ký tự tương ứng để máy phát ký tự biến đổi các mã ký tự đó thành 1 chuỗi các bit điểm ảnh (pixel bit) và chuyển chúng tới thanh ghi dịch (shift register). Máy phát tín hiệu sẽ sử dụng các bit điểm ảnh này cùng với các thông tin thuộc tính từ RAM video và các tín hiệu đồng bộ từ CRTC để phát ra các tín hiệu cần thiết cho monitor.

Trong chế độ đồ họa (graphics mode), thông tin trong RAM video được sử dụng trực tiếp cho việc phát ra các ký tự. Lúc này các thông tin về thuộc tính cũng không cần nữa. Chỉ từ các giá trị bit trong thanh ghi dịch, máy phát tín hiệu sẽ phát các tín hiệu về độ sáng và màu cho monitor.

Mỗi ký tự được biểu diễn bởi 1 từ 2 byte trong RAM video. Byte thấp chứa mã ký tự, byte cao chứa thuộc tính. Cấu trúc của một từ nhớ video như sau:

15	14	13	12	11	10	9	8
BLNK	BAK_2	BAK_1	BAK_0	INT	FOR_2	FOR_1	FOR_0

7	6	5	4	3	2	1	0
CHR_7	CHR_6	CHR_5	CHR_4	CHR_3	CHR_2	CHR_1	CHR_0

BLNK: Nhấp nháy; 1 = bật, 0 = tắt
INT: Cường độ sáng ; 1 = cao, 0 = bình thường
$CHR_7 \ldots CHR_{0:}$ Mã ký tự.

Bảng màu quy định như sau:

Mã hex	Màu	Mã hex	Màu	Mã hex	Màu	Mã hex	Màu
0	Black	4	Red	8	Dark Gray	C	Light Red
1	Blue	5	Magenta	9	Light Blue	D	Light Magenta
2	Green	6	Brown	A	Light Green	E	Yellow
3	Cyan	7	Light Gray	B	Light Cyan	F	White

Trong chế độ văn bản, 6845 liên tục xuất các địa chỉ cho RAM video qua MA0-MA13. Ký tự ở góc tận cùng phía trên bên trái màn hình có địa chỉ thấp nhất mà 6845 sẽ cung cấp ngay sau khi quét dọc ngược. Logic ghép nối định địa chỉ cho RAM video bằng việc lấy ra mã ký tự cùng với thuộc tính. Mã ký tự dùng cho máy phát ký tự như là chỉ số thứ nhất trong ROM ký tự. Lúc này, 6845 định địa chỉ hàng quét đầu tiên của ma trận ký tự, địa chỉ hàng bằng 0. Các bit của ma trận điểm ảnh bây giờ sẽ được truyền đồng bộ với tần số video từ thanh ghi dịch tới máy phát tín hiệu. Nếu máy phát tín hiệu nhận được giá trị 1 từ thanh ghi dịch, nó sẽ phát tín hiệu video tương ứng với màu của ký tự. Nếu nhận được 0 nó sẽ cấp tín hiệu tương ứng với màu nền. Vậy dòng quét thứ nhất được hiện phù hợp với các ma trận điểm ảnh của các ký tự trong hàng ký tự thứ nhất. Khi tia điện tử đạt tới cuối dòng quét, 6845 kích hoạt lối ra HS (Horizontal Synchronization) để tạo ra quá trình quét ngược và đồng bộ ngang. Tia điện tử quay trở về bắt đầu quét dòng tiếp. Sau mỗi dòng quét, 6845 tăng giá trị RA0-RA4 lên 1. Địa chỉ dòng này hình thành một giá trị offset bên trong ma trận điểm ảnh cho ký tự được hiện. Dựa trên mỗi dòng quét như vậy, một dòng các điểm ảnh của ký tự trong hàng ký tự được hiện ra. Điều này có nghĩa là với ma trận 9x14 điểm ảnh cho 1 ký tự, hàng ký tự thứ nhất đã được hiện sau 14 dòng quét. Khi địa chỉ RA0-RA4 trở về giá trị 0, 6845 sẽ cấp 1 địa chỉ MA0-MA13 mới và hàng ký tự thứ hai sẽ được hiện ra cũng như vậy. Ở cuối dòng quét cuối cùng, 6845 sẽ reset địa chỉ MA0-MA13 và RA0-RA4 và cho phép lối ra VS (Vertical Synchronization) phát ra tín hiệu quét ngược cùng tín hiệu đồng bộ dọc.

Mỗi ký tự có chiều cao cực đại ứng với 32 dòng vì có 5 đường địa chỉ RA0-RA4, còn bộ nhớ video trong trường hợp này được tới 16K từ vì có địa chỉ MA0-MA13 là 14 bit. Trong chế độ đồ họa, chúng kết hợp với nhau để tạo thành địa chỉ 19 bit, lúc đó 6845 có thể định địa chỉ cho bộ nhớ video lên tới 512K từ. Trong trường hợp này, các byte trong RAM video không được dịch thành mã ký tự và thuộc tính nữa mà trực tiếp xác định cường độ sáng và màu của điểm ảnh. Đa số các RAM video được chia thành vài băng được định địa chỉ bởi RA0-RA4. Các đường MA0-MA13 sẽ định địa chỉ offset bên trong mỗi băng. Dữ liệu trong RAM video lúc này được trực tiếp truyền tới thanh ghi dịch và máy phát tín hiệu. ROM ký tự và máy phát ký tự không làm việc.

RAM video được tổ chức khác nhau tuỳ theo chế độ hoạt động và bản mạch ghép nối. Thí dụ, với RAM video 128 KB, có thể địa chỉ hóa toàn bộ bộ nhớ màn hình qua CPU như bộ nhớ chính. Nhưng nếu kích thước RAM video lớn hơn thì làm như vậy sẽ đè lên vùng ROM mở rộng ở địa chỉ C0000h. Do đó, card EGA và VGA với trên 128 KB nhớ được tăng cường thêm 1 chuyển mạch mềm (soft-switch) cho phép thâm nhập các cửa sổ

128 KB khác nhau vào RAM video lớn hơn nhiều. Các chuyển mạch này được quy định bởi riêng các nhà sản xuất board mạch.

3.2. Chế độ văn bản

RAM video được coi như một dãy từ tuyến tính, từ đầu tiên được gán cho ký tự góc trên tận cùng bên trái màn hình gọi là hàng 1 cột 1. Từ thứ 2 là hàng 1, cột 2, …. Số từ tuỳ thuộc vào độ phân giải của kiểu hiện ký tự.

Thí dụ: độ phân giải chuẩn 25 hàng, 80 ký tự đòi hỏi 2000 từ nhớ 2 byte. Như vậy, tổng cộng cần 4 KB bộ nhớ RAM video. Trong khi đó với card có độ phân giải cao SVGA 60 hàng, 132 ký tự cần đến 15840 byte. Do đó RAM video thường được chia thành vài trang. Kích thước của mỗi trang tuỳ thuộc vào chế độ hiện của màn hình và số trang cực đại, phụ thuộc cả vào kích thước của RAM video. 6845 có thể được lập trình sao cho địa chỉ khởi phát của MA0-MA13 sau quét ngược dọc là khác 00h. Nếu địa chỉ khởi phát là bắt đầu của 1 trang thì có thể quản lý RAM video theo vài trang tách biệt nhau, nếu CPU thay đổi nội dung của 1 trang mà trang đó hiện đang không hiện thì màn hình cũng không thay đổi. Do đó, cần phân biệt trang nhớ đang được kích hoạt (đang hiện) và trang đang được xử lý.

Đoạn chương trình ghi ký tự 'A' có cường độ sáng cao vào góc trên bên trái với màu số 7 và màu nền số 0. Trang thứ nhất và là duy nhất bắt đầu ở địa chỉ B0000h.

```
MOV AX, 0B000h
MOV ES, AX
MOV AH, 0F8h
MOV AL, 41h
MOV ES:[00H],AX
```

3.3. Chế độ đồ hoạ

Tổ chức trong chế độ này phức tạp hơn. Ví dụ: với card Hercules, RAM video được chia thành 4 băng trên 1 trang . Băng thứ nhất: đảm bảo các điểm ảnh cho các dòng 0, 4, 8, …, 344; băng thứ hai cho các dòng 1, 5, 9, …, 345; băng thứ 3 cho các dòng 2, 6, 10, …., 346; và băng thứ 4 cho các dòng 3, 7, 11, …, 347. 64 KB được chia thành 2 trang 32 KB. Độ phân giải trong chế độ đồ họa là 720 x 348 điểm ảnh, mỗi điểm ảnh được biểu diễn bởi 1 bit. Do vậy, một dòng cần 90 byte (720 điểm ảnh / 8 điểm ảnh trên 1 byte). Địa chỉ của byte chứa điểm ảnh thuộc đường i và cột j trong trang k là:

B0000h+8000h*k+2000h*(i mod 4)+ 90*int(i/4)+int(j/8)

B0000h là đoạn video, 8000h là kích thước của trang, 2000h*(i mod 4) là offset của băng chứa byte đó, 90*int(i/4) là offset của dòng i trong băng và int(j/8) là offset của cột j trong băng.

3.4. Truy xuất màn hình qua DOS và BIOS

3.4.1. Truy xuất qua DOS

Các hàm của int 21h có thể hiện các ký tự trên màn hình nhưng không can thiệp được vào màu:

- Hàm 02h: ra màn hình.
- Hàm 09h: ra một chuỗi.
- Hàm 40h: ghi file/ thiết bị

Các lệnh **copy, type** và **print** trong command.com cho phép hiện text trên màn hình. DOS gộp chung bàn phím và monitor thành 1 thiết bị mang tên CON (console). Ghi CON là truyền số liệu tới monitor, còn đọc CON là nhận ký tự từ bàn phím. Ví dụ: để hiện nội dung của file output.txt lên màn hình của monitor sẽ có các cách sau:

- copy output.txt con
- type output.txt > con
- print output.txt /D:con

3.4.2. Truy xuất qua BIOS

BIOS thâm nhập monitor bằng int 10h với nhiều chức năng hơn DOS, như đặt chế độ hiện hình, quản lý tự động các trang, phân biệt các điểm trên màn hình nhờ các tọa độ,…

- ***Hàm 00h***: định chế độ đồ hoạ:

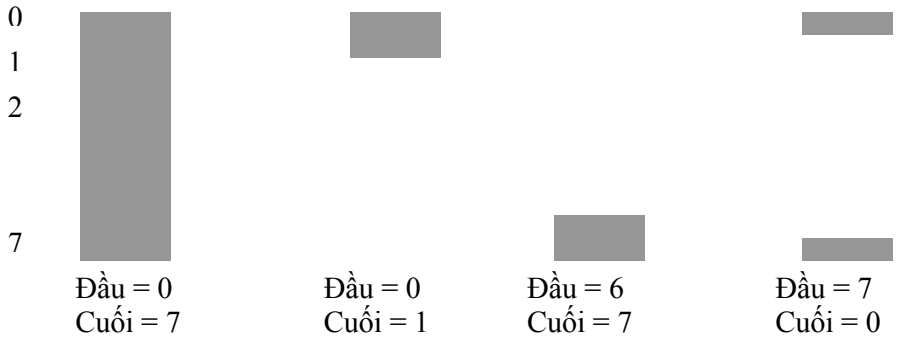
Vào:

AL = chế độ

Mode	Type	Max Colors	Size	Resolution	Max Pages	Base Addr
00	Text	16	40x25	- -	8	B8000h
01	Text	16	40x25	- -	8	B8000h
02	Text	16	80x25	- -	4,8	B8000h
03	Text	16	80x25	- -	4,8	B8000h
04	Graphics	4	40x25	320x200	1	B8000h
05	Graphics	4	40x25	320x200	1	B8000h
06	Graphics	2	80x25	640x200	1	B8000h
07	Text	Mono	80x25	- -	1,8	B0000h
08	Graphics	16	20x25	? ?	1	B0000h
09	Graphics	16	40x25	? ?	1	B0000h
0A	Graphics	4	80x25	? ?	1	B0000h
0B	- -	-	- -	- -	-	- -
0C	- -	-	- -	- -	-	- -
0D	Graphics	16	40x25	320x200	8	A0000h
0E	Graphics	16	80x25	640x200	4	A0000h
0F	Graphics	Mono	80x25	640x350	2	A0000h
10	Graphics	16	80x25	640x350	2	A0000h
11	Graphics	2	80x25	640x480	1	A0000h
12	Graphics	16	80x25	640x480	1	A0000h
13	Graphics	256	40x25	320x200	1	A0000h

- ***Hàm 01h***: xác định hình dạng con trỏ

Vào:



CH = dòng đầu con trỏ

CL = dòng cuối con trỏ

- ***Hàm 02h***: xác định vị trí con trỏ

Vào:

DH = hàng, DL = cột (bắt đầu từ toạ độ 0,0)

BH = trang (= 0 trong chế độ đồ hoạ)

- ***Hàm 03h***: lấy vị trí và hình dạng con trỏ

Vào: BH = trang

Ra:

DH = hàng, DL = cột

CH, CL = tham số xác định hình dạng con trỏ

- ***Hàm 05h***: chọn trang màn hình

AL = số trang (0..7)

- ***Hàm 06h***: cuộn cửa sổ lên

Vào:

AL = số hàng cuộn (= 0: cuộn toàn màn hình)

BH = thuộc tính các hàng trống xuất hiện khi cuộn

CH, CL = hàng, cột của góc phía trên bên trái

DH, DL = hàng, cột của góc phía dưới bên phải

- ***Hàm 07h***: cuộn cửa sổ xuống

Giống hàm 06h

- ***Hàm 08h***: đọc ký tự và thuộc tính ký tự tại vị trí con trỏ

Vào: BH = trang

Ra:

AH = thuộc tính

AL = mã ASCII

- ***Hàm 09h***: ghi ký tự và thuộc tính ký tự tại vị trí con trỏ

Vào:

BH = trang

BL = thuộc tính (theo bảng trang 61)

AL = mã ASCII

CX = số ký tự cần xuất

- ***Hàm 0Ah***: ghi ký tự tại vị trí con trỏ

Vào:

BH = trang

AL = mã ASCII

CX = số ký tự cần xuất

- ***Hàm 0Bh***: chọn bảng màu (dùng cho chế độ đồ hoạ)

Vào: BH = 0, BL = chọn màu nền (theo bảng trang 61)

BH = 1, BL = số bảng màu

Palette	Pixel	Color	Palette	Pixel	Color
0	0	Black	1	0	Black
0	1	Green	1	1	Cyan
0	2	Red	1	2	Magenta
0	3	Brown	1	3	White

- ***Hàm 0Ch***: hiện một điểm trên màn hình (dùng cho chế độ đồ hoạ)

AL = giá trị pixel (0 – 3), nếu AL.7 = 1 thì giá trị màu là phép toán XOR vứi giá trị màu hiện hành

CX = cột, DX = hàng

- ***Hàm 0Dh***: đọc một điểm trên màn hình (dùng cho chế độ đồ hoạ)

Vào: CX = cột, DX = hàng

Ra: AL = giá trị pixel

- ***Hàm 0Fh***: xác định chế độ màn hình hiện hành

Ra:

AL = chế độ

AH = số cột

BH = số trang

- ***Hàm 13h***: xuất chuỗi

Vào:

AL = chế độ xuất (bit0 = 1: cập nhật vị trí con trỏ sau khi xuất, bit1 = 1: chuỗi có chứa thuộc tính ký tự)

BH = trang

BL = thuộc tính (theo bảng trang 61)

CX = số ký tự cần xuất

CX = số ký tự cần xuất

DH, DL = hàng, cột xuất chuỗi

ES:BP = chuỗi in (nếu AL.1 = 1 thì chuỗi có dạng *'Kýtự',thuộc tính,'Kýtự',thuộc tính,'Kýtự',thuộc tính…*)

❖ **Những thường trình đồ họa:**

BIOS trên main board có sẵn những hàm dùng cho thâm nhập MDA và CGA. BIOS của riêng EGA và VGA có những hàm mở rộng tương ứng trong khi vẫn giữ nguyên định dạng gọi.

Một trong những hàm quan trọng nhất của int 10h là hàm 00h dùng để đặt chế độ hiện hình. Để thay đổi chế độ hiện hình cần phải làm rất nhiều bước chương trình phức tạp để nạp các thanh ghi của chip 6845. Trong khi đó, hàm 00h làm cho ta tất cả các công việc này.

Thí dụ: tạo kiểu 6 với độ phân giải 640*200 trên CGA.

```
Mov ah, 00h      ; hàm 00h
Mov al, 06h      ; chế độ 6
Int 10h          ; gọi ngắt
```

Các card EGA/VGA có riêng BIOS của chúng. Trong quá trình khởi động PC, nó sẽ chặn int 10h lại và chạy chương trình BIOS riêng. Thường trình cũ (của BIOS trên board mach chính) được thay địa chỉ tới int 42h. Tất cả các lệnh gọi int 10h sẽ được BIOS của EGA/VGA thay địa chỉ tới int 42h nếu board mạch EGA/VGA đang chạy các kiểu hiện tương thích với MDA hay CGA. Có các kiểu hoạt động từ 0 đến 7.

BIOS của EGA/VGA dùng vùng 40:84h tới 40:88h để lưu số liệu BIOS và các thông số của EGA/VGA. Nó có các hàm mới với các hàm phụ sau:

- Hàm 10h: truy xuất các thanh ghi màu và bảng màu
- Hàm 11h: cài đặt các bảng định nghĩa ký tự mới
- Hàm 12h: đặt cấu hình hệ con video
- Hàm 1Bh: thông tin về trạng thái và chức năng của BIOS video (chỉ có ở VGA)
- Hàm 1Ch: trạng thái save/restore của video (chỉ có ở VGA)

Sau đây là chức năng của các hàm và thí dụ sử dụng chúng:

- Hàm 10h, hàm phụ 03h – xoá/đặt thuộc tính

Ví dụ: Xoá thuộc tính nhấp nháy:

```
Mov ah, 10h      ; dùng hàm 10h
Mov al, 03h      ; dùng hàm phụ 03h
Mov bl, 00h      ; xoá thuộc tính nhấp nháy
Int 10h          ; gọi ngắt
```

- Hàm 11h – ghép nối với máy phát ký tự

Ví dụ: Nạp bảng định nghĩa ký tự 8*14 không cần chương trình CRTC:

```
Mov ah, 11h      ; dùng hàm 11h
Mov al, 01h      ; nạp bảng ký tự từ Rom Bios vào Ram máy phát ký tự.
Mov bl, 03h      ; gán số 3 cho bảng
```

```
Int 10h          ; gọi ngắt
```

- Hàm 12h, hàm phụ 20h – chọn thường trình in màn hình. Dùng hàm phụ này có thể thay thế thường trình chuẩn cho INT 05h bằng thường trình có thể dùng cho các độ phân giải mới của EGA/VGA.

Ví dụ: Cho phép thường trình mới in màn hình:

```
Mov ah, 12h      ; dùng hàm 12h
Mov bl, 20h      ; dùng hàm phụ 20h
```

Ấn PRINT hoặc SHIFT+PRINT để gọi thường trình in đã được lắp đặt.

❖ Truy xuất trực tiếp bộ nhớ video:

Để vẽ 1 điểm trên màn hình, BIOS phải làm nhiều thủ tục nhưng nếu muốn vẽ toàn bộ 1 cửa sổ hình hay lưu trữ thì phải truy xuất trực tiếp RAM video.

- Với board đơn sắc MDA trong kiểu hiện văn bản số 7, 4 KB RAM được tổ chức như 1 dãy (array) gồm 2000 từ nhớ kề nhau (mỗi từ là mã thuộc tính: ký tự) tạo nên 25 dòng, 80 cột. RAM video bắt đầu ở đoạn B0000h, trong đó ký tự góc trên cùng bên trái là từ thứ nhất trong RAM video. Như vậy mỗi dòng có 160 byte (A0h). Địa chỉ của từ nhớ ứng với ký tự ở dòng i, cột j (i = 0-24, j = 0-79) được tính theo công thức sau:

  Address (i,j) = B0000h +A0h*i +02h*j.

- Với board EGA, ở kiểu hiện văn bản từ 0 đến 3 mã ký tự được lưu trữ trong lớp nhớ 0 cùng với thuộc tính trong lớp 1 của RAM video. Mạch logic chuyển địa chỉ trên board thực hiện sự kết hợp nhất định nào đó sao cho tổ chức và cấu trúc của RAM video cũng như cách tính địa chỉ vẫn tương đồng với cách của CPU. Trong chế độ đồ họa từ 13 đến 16, RAM video bắt đầu từ địa chỉ đoạn A000h. Các điểm ảnh được xếp kề cận nhau trong bộ nhớ và mỗi điểm ảnh đòi hỏi 4 bit, các bit này được phân ra ở 4 lớp nhớ. Như vậy địa chỉ của 1 trong 4 bit này trên 1 điểm ảnh không chỉ gồm đoạn video và offset mà còn thêm vào số lớp nhớ nữa.

Để hiện 1 điểm ảnh với 1 trong 16 màu, không phải chỉ tính địa chỉ bit mà còn phải thâm nhập 4 lớp nhớ. Muốn vậy, phải dùng thanh ghi mặt nạ bản đồ (map mask register). Thanh ghi này được định địa chỉ qua cổng chỉ số 3C4h với địa chỉ 02h và có thể được ghi qua cổng số liệu 3C5h. Cấu trúc của thanh ghi mặt nạ bản đố như sau:

7	6	5	4	3	2	1	0
Res	Res	Res	Res	LY_3	LY_2	LY_1	LY_0

LY_3-LY_0: Thâm nhập ghi tới các lớp từ 0→3;
1 = cho phép; 0 = không cho phép

Res : Dự trữ

Ví dụ: Đặt bit 0 của byte ở địa chỉ A000:0000h cho lớ 0, 1, 3.

```
Mov AX, 0A000h      ; nạp đọan video vào AX
Mov ES, AX          ; truyền đọan video vào ES
Mov BX, 0000h       ; nạp offset 0000h vào BX
Out 3C4h, 02h       ; chỉ số 2 → thanh ghi mặt nạ bản đồ
Out  3C5h, 0Bh      ; ghi 0000 1011b vào thanh ghi mặt nạ bản đồ
                          (cho phép lớp 0, 1, 3)
Mov 3C5h, 0Bh       ; đặt bit 0 trong các lớp 0, 1 và 3
```

Để lưu trữ nội dung màn hình cần phải đọc các giá trị bit của 4 lớp khi dùng thanh ghi chọn bản đồ đọc (read map select register). Nó được định địa chỉ với chỉ số 04h qua cổng chỉ số 3CEh, và có thể được ghi qua cổng số liệu 3CFh. Cấu trúc của thanh ghi này:

7	6	5	4	3	2	1	0
res	res	res	res	res	res	LY_1	LY_0

LY_1-LY_0: cho phép thâm nhập đọc với:

00 = lớp 0
01 = lớp 1
10 = lớp 2
11 = lớp 3

res : dự trữ

Ví dụ: đọc byte ở địa chỉ A000:0000h cho lớp 2:

```
Mov AX, A000h       ; nạp đọan video vào AX
Mov ES, AX          ; truyền đọan video vào ES
Mov BX, 0000h       ; nạp offset vào BX
Out 3Ceh, 04h       ; chỉ số 4 → thanh ghi chọn bản đồ đọc
Out 3CFh, 02h       ; ghi 0000 0010b vào thanh ghi chọn bản đồ đọc
                          (cho phép lớp 2)
Mov  AL, [ES:BX]    ; nạp byte trong lớp 2 vào AL.
```

Chú ý rằng 4 bit tại 4 lớp đại diện cho 1 điểm ảnh nên trong kiểu hiện 16 EGA có độ phân giải cao nhất mỗi dòng cần 80byte (640 điểm ảnh / 8 điểm ảnh trên 1 byte); mỗi trang màn hình gồm 32 KB. Địa chỉ byte của điểm ảnh ở dòng i, cột j trang k (i=0-349, j=0-639, k=0-1) là:

$$\text{Address}(i,j,k) = A0000h + 8000h*k + 50h*j + \text{int}(i/8).$$

Với board VGA, các chế độ hiện văn bản từ 0 đến 3 và 7 cũng như các chế độ đồ họa từ 4 đến 6 và 13 đến 16 của CGA. EGA và MDA đều chạy được trên nó.

Trong chế độ văn bản, mã ký tự được lưu trữ trong lớp nhớ 0 cùng với thuộc tính trong lớp 1 của RAM video VGA. Quá trình chuyển hóa địa chỉ cũng giống như EGA nhưng khác ở chổ nó vẫn đảm bảo chế độ văn bản 7 với độ phân giải 720x400, ma trận điểm ảnh 9x16. Trong chế độ đồ họa $4 \div 6$ và $13 \div 19$, mọi tổ chức, cấu trúc cũng như cách tính

địa chỉ tương tự như CGA và EGA. VGA được tăng cường 3 kiểu hiện hình mới từ 17 đến 19.

Kiểu 17 tương thích với board đồ họa của máy PS/2 kiểu 30 là MCGA (multi colour graphics array). Các điểm ảnh chỉ gồm 1 bit (2 màu) được định vị chỉ trên lớp 0. Thí dụ, trong VGA kiểu 17 với 80 byte trên 1 dòng (640 điểm ảnh / 8 điểm ảnh trên 1 byte). Mỗi trang màn hình gồm 40 KB. Địa chỉ của byte ở dòng i, cột j (i= 0-479), j=0-639) như sau:

Address (i,j) = A0000h+50h*j+int (i/8)

Kiểu 18, 4 bit của điểm ảnh được phân trong 4 lớp nhớ như ở EGA. Trong kiểu VGA phân giải cao với 16 màu khác nhau, 80 byte trên 1 dòng (640 điểm ảnh / 8 điểm ảnh trên 1 byte), mỗi trang màn hình gồm 40 KB (A0000h byte); địa chỉ của mỗi byte ở dòng i, cột j (i=0-479; j = 0-639) bằng:

Address (i,j) = A0000h + 50h*j + int (i/8).

Kiểu 19 với 256 màu cho 1 điểm ảnh thì RAM video lại được tổ chức rất đơn giản như 1 dãy tuyến tính, trong đó 1 byte tương ứng với 1 điểm ảnh. Giá trị của byte phân định màu của điểm ảnh. Kiểu này đòi hỏi 320 byte (140h) trên 1 dòng (320 điểm ảnh / 1 điểm ảnh trên 1 byte). Một trang màn hình gồm 64 KB (10000h) nhưng chỉ có 64000 byte được sử dụng. Địa chỉ của điểm ảnh trong dòng i, cột j (i = 0-199, j=0-319) là:

Address (i,j) = A0000h + 140h*j + i

PHỤ LỤC CHƯƠNG 3

```
TITLE   DISPLAYING MOUSE POSITION
CURSOR   MACRO ROW,COLUMN
         MOV  AH,02H
         MOV  BH,00
         MOV  DH,ROW
         MOV  DL,COLUMN
         INT     10H
         ENDM
DISPLAY  MACRO STRING
         MOV  AH,09H
         MOV  DX,OFFSET STRING
         INT     21H
         ENDM
.MODEL SMALL
.STACK
.DATA
MESSAGE_1    DB 'PRESS ANY KEY$'
MESSAGE_2    DB 'THE MOUSE CURSOR IS LOCATED AT $'
POS_HO       DB ?,?, ' AND $'
POS_VE       DB ?,?,'$'
OLDVIDEO     DB ?            ;current video mode
NEWVIDEO     DB 0EH          ;new video mode
.CODE
MAIN  PROC
     MOV  AX,@DATA
     MOV  DS,AX
     MOV  AH,0FH          ;get current video mode
     INT  10H
     MOV  OLDVIDEO,AL     ;save it
     MOV  AX,0600H       ;clear screen
     MOV  BH,07
     MOV  CX,0
     MOV  DX,184FH
     INT  10H
     MOV  AH,00H         ;set new video mode
     MOV  AL,NEWVIDEO
     INT  10H
     MOV  AX,0                   ;initialize mouse
     INT  33H
     MOV  AX,01        ;show mouse cursor
     INT  33H
     CURSOR 20,20
     DISPLAY MESSAGE_1
AGAIN:
     MOV  AX,03H         ;get mouse location
     INT  33H
     MOV  AX,CX   ;get the hor. pixel position
     CALL CONVERT
     MOV  POS_HO,AL  ;save the LSD
     MOV  POS_HO+1,AH  ;save  the MSD
```

```
          MOV  AX,DX   ;get the vert. pixel position
          CALL CONVERT
          MOV  POS_VE,AL  ;save
          MOV  POS_VE+1,AH
          CURSOR 5,20
          DISPLAY MESSAGE_2
          DISPLAY POS_HO
          DISPLAY POS_VE
          MOV  AH,01               ;check for key press
          INT  16H
          JZ   AGAIN ;if no key press
          MOV  AH,02               ;hide mouse
          INT  33H
          MOV  AH,0             ;restore original video mode
          MOV  AL,OLDVIDEO         ;load original video mode
          MOV  AH,0 ;restore original video mode
          INT     10H
          MOV     AH,4CH               ;go back to DOS
          INT     21H
     MAIN     ENDP
     ;--------------------
     ;divide pixels position by 8 and convert to ASCII to make
it displayable
     ;ax=pixels position (it is in hex)
     ;on return ax= two ASCII digits
     CONVERT  PROC
     SHR  AX,1          ;divide
     SHR  AX,1          ;by 8
     SHR  AX,1          ;to get screen position by character
     MOV  BL,10
     MOV  AH,0
     DIV  BL
     ADD  AX,3030H      ;make it ASCII
     RET                   ;return with AX=two ASCII digits
     CONVERT  ENDP
     END     MAIN
     ---------------------------------------------------------------
```

```
     ;THIS PROGRAM WAITS FOR THE MOUSE PRESS COUNT AND
     ;DISPLAYS IT WHEN ANY KEY IS PRESSED.
     .MODEL   SMALL
     .STACK
     .DATA
     MESSAGE_1  DB 'PRESS LEFT BUTTON A NUMBER OF TIMES:LESS
THAN 99.$'
     MESSAGE_2   DB 'TO FIND OUT HOW MANY TIMES, PRESS ANY
KEY$'
     MESSAGE_3  DB 'YOU PRESSED IT $'
     P_COUNT    DB ?,?, ' TIMES $'
     MESSAGE_4  DB 'NOW PRESS ANY KEY TO GO BACK TO DOS$'
     OLDVIDEO   DB ?
     NEWVIDEO   DB 12H
```

```
.CODE
MAIN    PROC
 MOV  AX,@DATA
 MOV  DS,AX
 MOV  AH,0FH  ;get current video mode
 INT  10H
 MOV  OLDVIDEO,AL  ;save it
 MOV  AX,0600H  ;clear screen
 MOV  BH,07
 MOV  CX,0
 MOV  DX,184FH
 INT  10H
 MOV  AH,00H  ;set new video mode
 MOV  AL,NEWVIDEO
 INT  10H
 MOV  AX,0   ;initialize mouse
 INT  33H
 MOV  AX,01  ;show mouse cursor
 INT  33H
 CURSOR 2,1
 DISPLAY MESSAGE_1
 CURSOR 4,1
 DISPLAY MESSAGE_2
 MOV  AH,07  ;wait for key press
 INT  21H
 MOV  AX,05H  ;get mouse press count
 MOV  BX,0  ;check press count for left button
 INT  33H
 MOV  AX,BX  ;BX=button press count
 MOV  BL,10
 DIV  BL
 ADD  AX,3030H
 MOV  P_COUNT,AL  ;save the number
 MOV  P_COUNT+1,AH
 CURSOR 10,2
 DISPLAY MESSAGE_3
 DISPLAY P_COUNT
 CURSOR 20,2
 DISPLAY MESSAGE_4
 MOV  AH,07          ;wait for a key press to get out
 INT  21H
 MOV  AH,02          ;hide mouse
 INT  33H
 MOV  AH,0           ;restore original video mode
 MOV  AL,OLDVIDEO  ;load original vide mode
 INT  10H
 MOV  AH,4CH         ;go back to DOS
 INT  21H
MAIN  ENDP
 END MAIN
;-------------------------------------------------

.MODEL SMALL
```

```
.STACK 100h
.DATA
mask_mon DB   0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh
         DB   0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh
         DB   0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh
         DB   0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh,0FFh
mask_p   DB   80h,0,0E0h,0,0F8h,0,0FEh,0
         DB   0D8h,0,0Ch,0,6,0,3,0
         DB   0,0,0,0,0,0,0,0
         DB   0,0,0,0,0,0,0,0
.CODE
     main PROC
          mov ax,@data
          mov ds,ax
          mov es,ax

          mov ah,0
          mov al,6
          int 10h

          mov ax,0
          int 33h

          mov ax,1
          int 33h

          mov ah,08h
          int 21h

          mov ax,9
          mov bx,0
          mov cx,0
          lea dx,mask_mon
          int 33h

          mov ah,08h
          int 21h

          mov ah,4Ch
          int 21h
     main ENDP
     END  main
```

1000000000000000
1110000000000000
1111100000000000
1111111000000000
1101100000000000
0000110000000000
0000011000000000
0000001100000000

0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000

Keyboard Scan Codes: Set 1

*All values are in hexadecimal

101-, 102-, and 104-key keyboards:

KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK
A	1E	9E		9	0A	8A		[	1A	9A
B	30	B0		`	29	89		INSERT	E0,52	E0,D2
C	2E	AE		-	0C	8C		HOME	E0,47	E0,97
D	20	A0		=	0D	8D		PG UP	E0,49	E0,C9
E	12	92		\	2B	AB		DELETE	E0,53	E0,D3
F	21	A1		BKSP	0E	8E		END	E0,4F	E0,CF
G	22	A2		SPACE	39	B9		PG DN	E0,51	E0,D1
H	23	A3		TAB	0F	8F		U ARROW	E0,48	E0,C8
I	17	97		CAPS	3A	BA		L ARROW	E0,4B	E0,CB
J	24	A4		L SHFT	2A	AA		D ARROW	E0,50	E0,D0
K	25	A5		L CTRL	1D	9D		R ARROW	E0,4D	E0,CD
L	26	A6		L GUI	E0,5B	E0,DB		NUM	45	C5
M	32	B2		L ALT	38	B8		KP /	E0,35	E0,B5
N	31	B1		R SHFT	36	B6		KP *	37	B7
O	18	98		R CTRL	E0,1D	E0,9D		KP -	4A	CA
P	19	99		R GUI	E0,5C	E0,DC		KP +	4E	CE
Q	10	19		R ALT	E0,38	E0,B8		KP EN	E0,1C	E0,9C
R	13	93		APPS	E0,5D	E0,DD		KP .	53	D3
S	1F	9F		ENTER	1C	9C		KP 0	52	D2
T	14	94		ESC	01	81		KP 1	4F	CF
U	16	96		F1	3B	BB		KP 2	50	D0
V	2F	AF		F2	3C	BC		KP 3	51	D1
W	11	91		F3	3D	BD		KP 4	4B	CB
X	2D	AD		F4	3E	BE		KP 5	4C	CC
Y	15	95		F5	3F	BF		KP 6	4D	CD
Z	2C	AC		F6	40	C0		KP 7	47	C7

0	0B	8B		F7	41	C1		KP 8	48	C8
1	02	82		F8	42	C2		KP 9	49	C9
2	03	83		F9	43	C3		]	1B	9B
3	04	84		F10	44	C4		;	27	A7
4	05	85		F11	57	D7		'	28	A8
5	06	86		F12	58	D8		,	33	B3
6	07	87		PRNT SCRN	E0,2A, E0,37	E0,B7, E0,AA		.	34	B4
7	08	88		SCROLL	46	C6		/	35	B5
8	09	89		PAUSE	E1,1D,45 E1,9D,C5	-NONE-				

Keyboard Scan Codes: Set 2

*All values are in hexadecimal

101-, 102-, and 104-key keyboards:

KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK
A	1C	F0,1C		9	46	F0,46		[	54	FO,54
B	32	F0,32		`	0E	F0,0E		INSERT	E0,70	E0,F0,70
C	21	F0,21		-	4E	F0,4E		HOME	E0,6C	E0,F0,6C
D	23	F0,23		=	55	FO,55		PG UP	E0,7D	E0,F0,7D
E	24	F0,24		\	5D	F0,5D		DELETE	E0,71	E0,F0,71
F	2B	F0,2B		BKSP	66	F0,66		END	E0,69	E0,F0,69
G	34	F0,34		SPACE	29	F0,29		PG DN	E0,7A	E0,F0,7A
H	33	F0,33		TAB	0D	F0,0D		U ARROW	E0,75	E0,F0,75
I	43	F0,43		CAPS	58	F0,58		L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B		L SHFT	12	FO,12		D ARROW	E0,72	E0,F0,72
K	42	F0,42		L CTRL	14	FO,14		R ARROW	E0,74	E0,F0,74
L	4B	F0,4B		L GUI	E0,1F	E0,F0,1F		NUM	77	F0,77
M	3A	F0,3A		L ALT	11	F0,11		KP /	E0,4A	E0,F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7C	F0,7C
O	44	F0,44		R CTRL	E0,14	E0,F0,14		KP -	7B	F0,7B
P	4D	F0,4D		R GUI	E0,27	E0,F0,27		KP +	79	F0,79
Q	15	F0,15		R ALT	E0,11	E0,F0,11		KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D		APPS	E0,2F	E0,F0,2F		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	76	F0,76		KP 1	69	F0,69
U	3C	F0,3C		F1	05	F0,05		KP 2	72	F0,72
V	2A	F0,2A		F2	06	F0,06		KP 3	7A	F0,7A
W	1D	F0,1D		F3	04	F0,04		KP 4	6B	F0,6B

X	22	F0,22		F4	0C	F0,0C		KP 5	73	F0,73
Y	35	F0,35		F5	03	F0,03		KP 6	74	F0,74
Z	1A	F0,1A		F6	0B	F0,0B		KP 7	6C	F0,6C
0	45	F0,45		F7	83	F0,83		KP 8	75	F0,75
1	16	F0,16		F8	0A	F0,0A		KP 9	7D	F0,7D
2	1E	F0,1E		F9	01	F0,01		]	5B	F0,5B
3	26	F0,26		F10	09	F0,09		;	4C	F0,4C
4	25	F0,25		F11	78	F0,78		'	52	F0,52
5	2E	F0,2E		F12	07	F0,07		,	41	F0,41
6	36	F0,36		PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12		.	49	F0,49
7	3D	F0,3D		SCROLL	7E	F0,7E		/	4A	F0,4A
8	3E	F0,3E		PAUSE	E1,14,77, E1,F0,14, F0,77	-NONE-				

AT Keyboard Scan Codes (Set 3)

KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK
A	1C	F0,1C		9	46	F0,46		[	54	F0,54
B	32	F0,32		`	0E	F0,0E		INSERT	67	F0,67
C	21	F0,21		-	4E	F0,4E		HOME	6E	F0,6E
D	23	F0,23		=	55	F0,55		PG UP	6F	F0,6F
E	24	F0,24		\	5C	F0,5C		DELETE	64	F0,64
F	2B	F0,2B		BKSP	66	F0,66		END	65	F0,65
G	34	F0,34		SPACE	29	F0,29		PG DN	6D	F0,6D
H	33	F0,33		TAB	0D	F0,0D		U ARROW	63	F0,63
I	43	F0,48		CAPS	14	F0,14		L ARROW	61	F0,61
J	3B	F0,3B		L SHFT	12	F0,12		D ARROW	60	F0,60
K	42	F0,42		L CTRL	11	F0,11		R ARROW	6A	F0,6A
L	4B	F0,4B		L WIN	8B	F0,8B		NUM	76	F0,76
M	3A	F0,3A		L ALT	19	F0,19		KP /	4A	F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7E	F0,7E
O	44	F0,44		R CTRL	58	F0,58		KP -	4E	F0,4E
P	4D	F0,4D		R WIN	8C	F0,8C		KP +	7C	F0,7C
Q	15	F0,15		R ALT	39	F0,39		KP EN	79	F0,79
R	2D	F0,2D		APPS	8D	F0,8D		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	08	F0,08		KP 1	69	F0,69

U	3C	F0,3C		F1	07	F0,07		KP 2	72	F0,72
V	2A	F0,2A		F2	0F	F0,0F		KP 3	7A	F0,7A
W	1D	F0,1D		F3	17	F0,17		KP 4	6B	F0,6B
X	22	F0,22		F4	1F	F0,1F		KP 5	73	F0,73
Y	35	F0,35		F5	27	F0,27		KP 6	74	F0,74
Z	1A	F0,1A		F6	2F	F0,2F		KP 7	6C	F0,6C
0	45	F0,45		F7	37	F0,37		KP 8	75	F0,75
1	16	F0,16		F8	3F	F0,3F		KP 9	7D	F0,7D
2	1E	F0,1E		F9	47	F0,47		]	5B	F0,5B
3	26	F0,26		F10	4F	F0,4F		;	4C	F0,4C
4	25	F0,25		F11	56	F0,56		'	52	F0,52
5	2E	F0,2E		F12	5E	F0,5E		,	41	F0,41
6	36	F0,36		PRNT SCRN	57	F0,57		.	49	F0,49
	3D	F0,3D		SCROLL	5F	F0,5F		/	4A	F0,4A
8	3E	F0,3E		PAUSE	62	F0,62				

Chương 4
GIAO TIẾP CỔNG NỐI TIẾP

1. Cấu trúc cổng nối tiếp

Cổng nối tiếp được sử dụng để truyền dữ liệu hai chiều giữa máy tính và ngoại vi, có các ưu điểm sau:

- Khoảng cách truyền xa hơn truyền song song.
- Số dây kết nối ít.
- Có thể truyền không dây dùng hồng ngoại.
- Có thể ghép nối với vi điều khiển hay PLC (Programmable Logic Device).
- Cho phép nối mạng.
- Có thể tháo lắp thiết bị trong lúc máy tính đang làm việc.
- Có thể cung cấp nguồn cho các mạch điện đơn giản

Các thiết bị ghép nối chia thành 2 loại: DTE (Data Terminal Equipment) và DCE (Data Communication Equipment). DCE là các thiết bị trung gian như MODEM còn DTE là các thiết bị tiếp nhận hay truyền dữ liệu như máy tính, PLC, vi điều khiển, … Việc trao đổi tín hiệu thông thường qua 2 chân RxD (nhận) và TxD (truyền). Các tín hiệu còn lại có chức năng hỗ trợ để thiết lập và điều khiển quá trình truyền, được gọi là các tín hiệu bắt tay (handshake). Ưu điểm của quá trình truyền dùng tín hiệu bắt tay là có thể kiểm soát đường truyền.

Tín hiệu truyền theo chuẩn RS-232 của EIA (Electronics Industry Associations). Chuẩn RS-232 quy định mức logic 1 ứng với điện áp từ -3V đến -25V (mark), mức logic 0 ứng với điện áp từ 3V đến 25V (space) và có khả năng cung cấp dòng từ 10 mA đến 20 mA. Ngoài ra, tất cả các ngõ ra đều có đặc tính chống chập mạch.

Chuẩn RS-232 cho phép truyền tín hiệu với tốc độ đến 20.000 bps nhưng nếu cáp truyền đủ ngắn có thể lên đến 115.200 bps.

Các phương thức nối giữa DTE và DCE:

- Đơn công (simplex connection): dữ liệu chỉ được truyền theo 1 hướng.

- Bán song công (half-duplex): dữ liệu truyền theo 2 hướng, nhưng mỗi thời điểm chỉ được truyền theo 1 hướng.

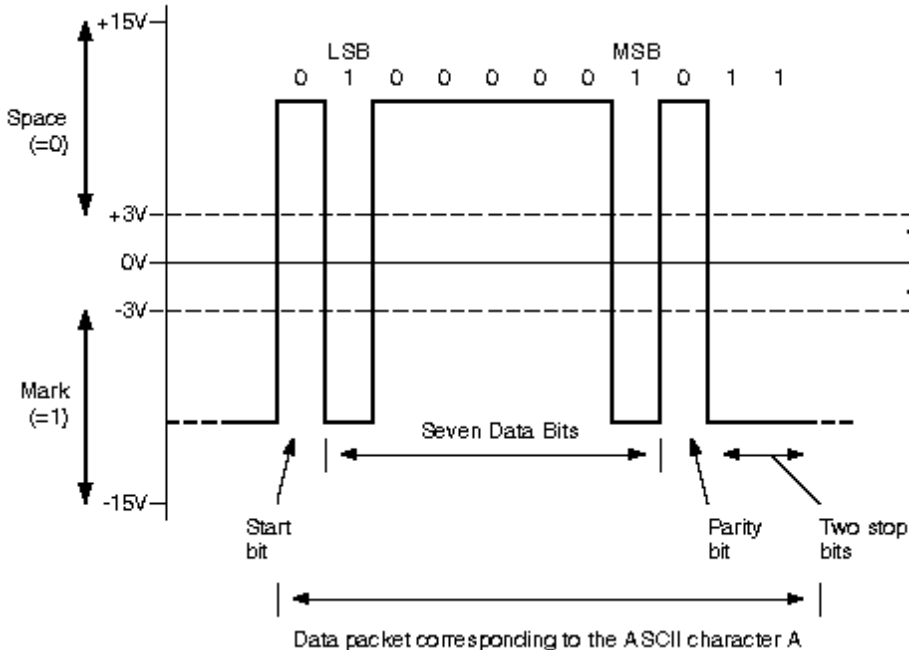
- Song công (full-duplex): số liệu được truyền đồng thời theo 2 hướng.

Định dạng của khung truyền dữ liệu theo chuẩn RS-232 như sau:

	Start	D0	D1	D2	D3	D4	D5	D6	D7	P	Stop
	0										1

Khi không truyền dữ liệu, đường truyền sẽ ở trạng thái mark (điện áp -10V). Khi bắt đầu truyền, DTE sẽ đưa ra xung Start (space: 10V) và sau đó lần lượt truyền từ D0 đến D7

và Parity, cuối cùng là xung Stop (mark: -10V) để khôi phục trạng thái đường truyền. Dạng tín hiệu truyền mô tả như sau (truyền ký tự A):



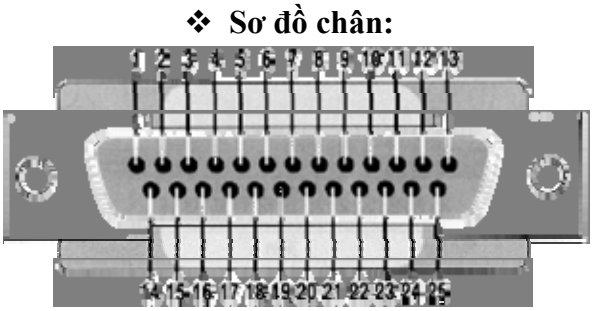
Hình 4.1 – Tín hiệu truyền của ký tự 'A'

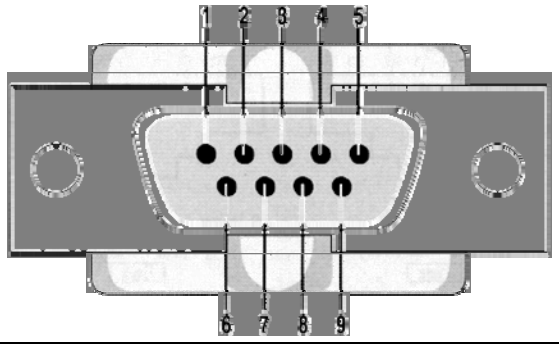
Các đặc tính kỹ thuật của chuẩn RS-232 như sau:

Chiều dài cable cực đại	15m
Tốc độ dữ liệu cực đại	20 Kbps
Điện áp ngõ ra cực đại	± 25V
Điện áp ngõ ra có tải	± 5V đến ± 15V
Trở kháng tải	3K đến 7K
Điện áp ngõ vào	± 15V
Độ nhạy ngõ vào	± 3V
Trở kháng ngõ vào	3K đến 7K

Các tốc độ truyền dữ liệu thông dụng trong cổng nối tiếp là: 1200 bps, 4800 bps, 9600 bps và 19200 bps.

❖ **Sơ đồ chân:**





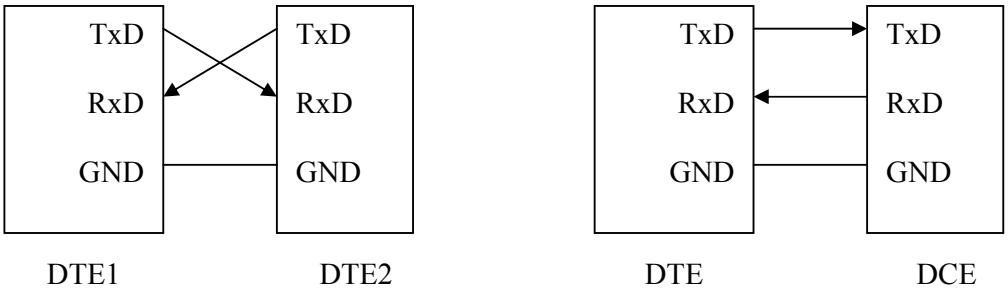
Hình 4.2 – Sơ đồ chân cổng nối tiếp

Cổng COM có hai dạng: đầu nối DB25 (25 chân) và đầu nối DB9 (9 chân) mô tả như hình 4.2. Ý nghĩa của các chân mô tả như sau:

D25	D9	Tín hiệu	Hướng truyền	Mô tả
1	-	-	-	Protected ground: nối đất bảo vệ
2	3	TxD	DTE→DCE	Transmitted data: dữ liệu truyền
3	2	RxD	DCE→DTE	Received data: dữ liệu nhận
4	7	RTS	DTE→DCE	Request to send: DTE yêu cầu truyền dữ liệu
5	8	CTS	DCE→DTE	Clear to send: DCE sẵn sàng nhận dữ liệu
6	6	DSR	DCE→DTE	Data set ready: DCE sẵn sàng làm việc
7	5	GND	-	Ground: nối đất (0V)
8	1	DCD	DCE→DTE	Data carier detect: DCE phát hiện sóng mang
20	4	DTR	DTE→DCE	Data terminal ready: DTE sẵn sàng làm việc
22	9	RI	DCE→DTE	Ring indicator: báo chuông
23	-	DSRD	DCE→DTE	Data signal rate detector: dò tốc độ truyền
24	-	TSET	DTE→DCE	Transmit Signal Element Timing: tín hiệu định thời truyền đi từ DTE
15	-	TSET	DCE→DTE	Transmitter Signal Element Timing: tín hiệu định thời truyền từ DCE để truyền dữ liệu
17	-	RSET	DCE→DTE	Receiver Signal Element Timing: tín hiệu định thời truyền từ DCE để truyền dữ liệu
18	-	LL		Local Loopback: kiểm tra cổng
21	-	RL	DCE→DTE	Remote Loopback: Tạo ra bởi DCE khi tín hiệu nhận từ DCE lỗi
14	-	STxD	DTE→DCE	Secondary Transmitted Data
16	-	SRxD	DCE→DTE	Secondary Received Data
19	-	SRTS	DTE→DCE	Secondary Request To Send
13	-	SCTS	DCE→DTE	Secondary Clear To Send
12	-	SDSRD	DCE→DTE	Secondary Received Line Signal Detector
25	-	TM		Test Mode
9	-			Dành riêng cho chế độ test
10	-			Dành riêng cho chế độ test
11				Không dùng

2. Truyền thông giữa hai nút

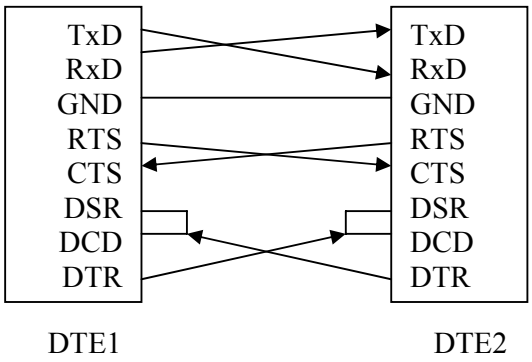
Các sơ đồ khi kết nối dùng cổng nối tiếp:



Hình 4.3 – Kết nối đơn giản trong truyền thông nối tiếp

Khi thực hiện kết nối như trên, quá trình truyền phải bảo đảm tốc độ ở đầu phát và thu giống nhau. Khi có dữ liệu đến DTE, dữ liệu này sẽ được đưa vào bộ đệm và tạo ngắt.

Ngoài ra, khi thực hiện kết nối giữa hai DTE, ta còn dùng sơ đồ sau:



Hình 4.4 – Kết nối trong truyền thông nối tiếp dùng tín hiệu bắt tay

Khi DTE1 cần truyền dữ liệu thì cho DTR tích cực → tác động lên DSR của DTE2 cho biết sẵn sàng nhận dữ liệu và cho biết đã nhận được sóng mang của MODEM (ảo). Sau đó, DTE1 tích cực chân RTS để tác động đến chân CTS của DTE2 cho biết DTE1 có thể nhận dữ liệu. Khi thực hiện kết nối giữa DTE và DCE, do tốc độ truyền khác nhau nên phải thực hiện điều khiển lưu lượng. Quá trinh điều khiển này có thể thực hiện bằng phần mềm hay phần cứng. Quá trình điều khiển bằng phần mềm thực hiện bằng hai ký tự Xon và Xoff. Ký tự Xon được DCE gởi đi khi rảnh (có thể nhận dữ liệu). Nếu DCE bận thì sẽ gởi ký tự Xoff. Quá trình điều khiển bằng phần cứng dùng hai chân RTS và CTS. Nếu DTE muốn truyền dữ liệu thì sẽ gởi RTS để yêu cầu truyền, DCE nếu có khả năng nhận dữ liệu (đang rảnh) thì gởi lại CTS.

3. Truy xuất trực tiếp thông qua cổng

Các cổng nối tiếp trong máy tính được đánh số là COM1, COM2, COM3, COM4 với các địa chỉ như sau:

Tên	Địa chỉ	Ngắt	Vị trí chứa địa chỉ
COM1	3F8h	4	0000h:0400h
COM2	2F8h	3	0000h:0402h
COM3	3E8h	4	0000h:0404h
COM4	2E8h	3	0000h:0406h

Giao tiếp nối tiếp trong máy tính sử dụng vi mạch UART với các thanh ghi cho trong bảng sau:

Offset	DLAB	R/W	Tên	Chức năng
0	0	W	THR	Transmitter Holding Register (đệm truyền)
	0	R	RBR	Receiver Buffer Register (đệm thu)
	1	R/W	BRDL	Baud Rate Divisor Latch (số chia byte thấp)
1	0	R/W	IER	Interrupt Enable Register (cho phép ngắt)
	1	R/W	BRDH	Số chia byte cao
2		R	IIR	Interrupt Identification Register (nhận dạng ngắt)
		W	FCR	FIFO Control Register
3		R/W	LCR	Line Control Register (điều khiển đường dây)
4		R/W	MCR	Modem Control Register (điều khiển MODEM)
5		R	LSR	Line Status Register (trạng thái đường dây)
6		R	MSR	Modem Status Register (trạng thái MODEM)
7		R/W		Scratch Register (thanh ghi tạm)

Các thanh ghi này có thể truy xuất trực tiếp kết hợp với địa chỉ cổng (ví dụ như thanh ghi cho phép ngắt của COM1 có địa chỉ là $BA_{COM1} + 1 = 3F9h$.

❖ IIR (Interrupt Identification):

IIR xác định mức ưu tiên và nguồn gốc của yêu cầu ngắt mà UART đang chờ phục vụ. Khi cần xử lý ngắt, CPU thực hiện đọc các bit tương ứng để xác định nguồn gốc của ngắt. Định dạng của IIR như sau:

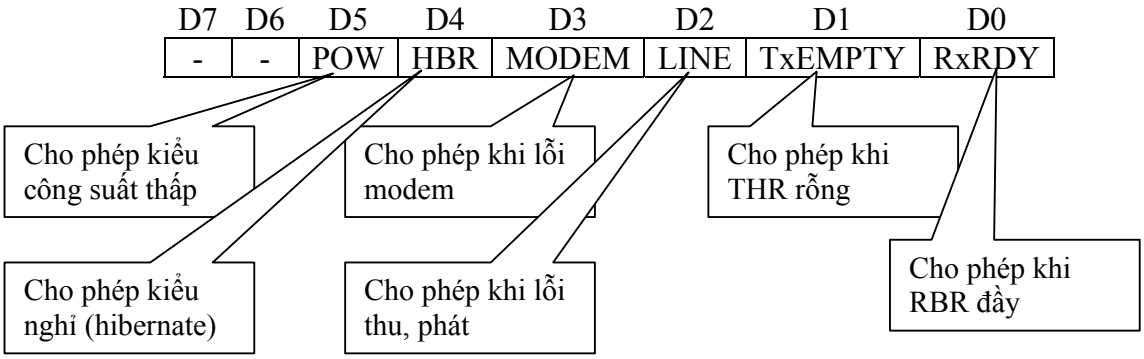
D7 D6	D5	D4	D3	D2 D1	D0
00: không có FIFO 11: cho phép FIFO	Cho phép FIFO 64 byte (trong 16750)	-	1: ngắt time-out (trong 16550)	Xác định nguồn gốc ngắt	0: có ngắt 1: không ngắt

D2	D1	Ưu tiên	Tên	Nguồn	D2 – D0 bị xoá khi
0	0	4	Đường truyền	Lỗi khung, thu đè, lỗi parity, gián đoạn khi thu	Đọc LSR
0	1	3	Đệm thu	Đệm thu đầy	Đọc RBR
1	0	2	Đệm phát	Đệm phát rỗng	Đọc IIR, ghi THR
1	1	1	Modem	CTS, DSR, RI, RLSD	Đọc MSR

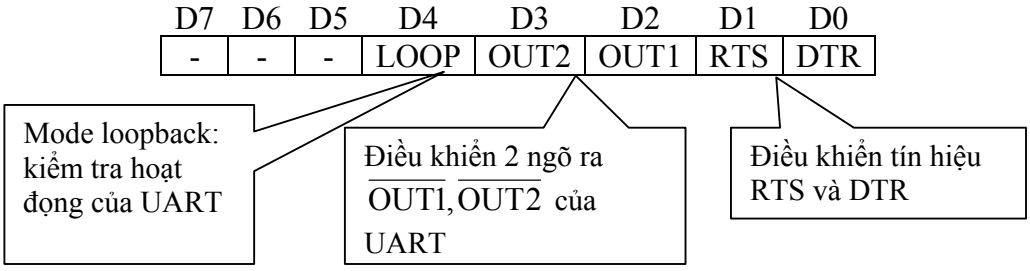
(mức 1 ưu tiên cao nhất)

❖ **IER (Interrupt Enable Register):**

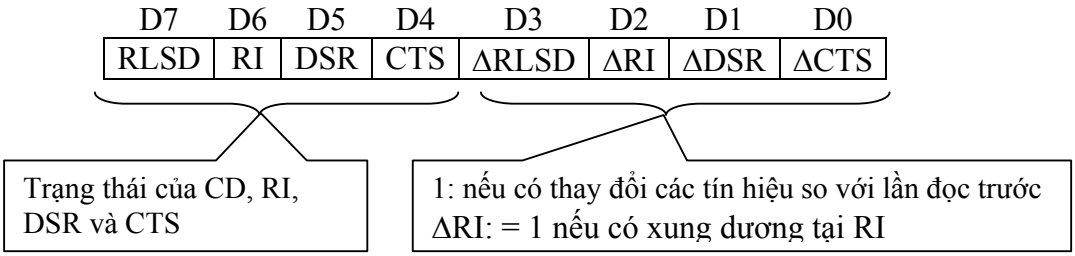
IER cho phép hay cấm các nguyên nhân ngắt khác nhau (1: cho phép, 0: cầm ngắt)



❖ **MCR (Modem Control Register):**



❖ **MSR (Modem Status Register):**



❖ **LSR (Line Status Register):**

D7	D6	D5	D4	D3	D2	D1	D0
FIE	TSRE	THRE	BI	FE	PE	OE	RxDR

FIE: FIFO Error – sai trong FIFO

TSRE: Transmitter Shift Register Empty – thanh ghi dịch rỗng (=1 khi đã phát 1 ký tự và bị xoá khi có 1 ký tự chuyển đến từ THR.

THRE: Transmitter Holding Register Empty (=1 khi có 1 ký tự đã chuyển từ THR – TSR và bị xoá khi CPU đưa ký tự tới THR).

BI: Break Interrupt (=1 khicó sự gián đoạn khi truyền, nghĩa là tồn tại mức logic 0 trong khoảng thời gian dài hơn khoảng thời gian truyền 1 byte và bị xoá khi CPU đọc LSR)

FE: Frame Error (=1 khi có lỗi khung truyền và bị xoá khi CPU đọc LSR)

PE: Parity Error (=1 khi có lỗi parity và bị xoá khi CPU đọc LSR)

OE: Overrun Error (=1 khi có lỗi thu đè, nghĩa là CPU không đọc kịp dữ liệu làm cho quá trình ghi chồng lên RBR xảy ra và bị xoá khi CPU đọc LSR)

RxDR: Receiver Data Ready (=1 khi đã nhận 1 ký tự và đưa vào RBR và bị xoá khi CPU đọc RBR).

❖ **LCR (Line Control Register):**

D7	D6	D5	D4	D3	D2	D1	D0
DLAB	SBCB	PS2	PS1	PS0	STB	WLS1	WLS0

DLAB (Divisor Latch Access Bit) = 0: truy xuất RBR, THR, IER, = 1 cho phép đặt bộ chia tần trong UART để cho phép đạt tốc độ truyền mong muốn.

UART dùng dao động thạch anh với tần số 1.8432 MHz đưa qua bộ chia 16 thành tần số 115,200 Hz. Khi đó, tuỳ theo giá trị trong BRDL và BRDH, ta sẽ có tốc độ mong muốn. Ví dụ như đường truyền có tốc độ truyền 2,400 bps có giá trị chia 115,200 / 2,400 = 48d = 0030h → BRDL = 30h, BRDH = 00h.

Một số giá trị thông dụng xác định tốc độ truyền cho như sau:

Tốc độ (bps)	**BRDH**	**BRDL**
1,200	00h	60h
2,400	00h	30h
4,800	00h	18h
9,600	00h	0Ch
19,200	00h	06h
38,400	00h	03h
57,600	00h	02h
115,200	00h	01h

SBCB (Set Break Control Bit) =1: cho phép truyền tín hiệu Break (=0) trong khoảng thời gian lớn hơn một khung

PS (Parity Select):

PS2	**PS1**	**PS0**	**Mô tả**
X	X	0	Không kiểm tra
0	0	1	Kiểm tra lẻ
0	1	1	Kiểm tra chẵn
1	0	1	Parity là mark
1	1	1	Parity là space

STB (Stop Bit) = 0: 1 bit stop, =1: 1.5 bit stop (khi dùng 5 bit dữ liệu) hay 2 bit stop (khi dùng 6, 7, 8 bit dữ liệu).

WLS (Word Length Select):

WLS1	WLS0	Độ dài dữ liệu
0	0	5 bit
0	1	6 bit
1	0	7 bit
1	1	8 bit

Một ví dụ khi lập trình trực tiếp trên cổng như sau:

```
.MODEL SMALL
.STACK 100h
.DATA
     Com1      EQU 3F8h
     Com_int   EQU 08h
     Buffer    DB   251 DUP(?)
     Bufferin  DB   0
     Bufferout      DB   0
     Char      DB   ?
     Seg_com   DW   ?    ; Vector ng•t c•
     Off_com   DW   ?
     Mask_int  DB   ?
     Msg       DB   'Press any key to exit$'
.CODE
Main PROC
     MOV  AX,@DATA
     MOV  DS,AX

     MOV  AH,35h
     MOV  AL,Com_int
     INT  21h
     MOV  Seg_com,ES     ; L•u vector ng•t c•
     MOV  Off_com,BX

     PUSH DS
     MOV  BX,CS
     MOV  DS,BX
     LEA  DX,Com_ISR
     MOV  AH,35h         ;Gán vector ng•t m•i
     MOV  AL,Com_int
     INT  21h
     POP  DS

     MOV  DX,Com1+3      ; ••a ch• LCR
     MOV  AL,80h         ; Set DLAB = 1 cho phép ••nh t•c
     OUT  DX,AL          ; •• truy•n d• li•u
```

```
        MOV  DX,Com1        ; G•i byte th•p
        MOV  AL,0Ch
        OUT  DX,AL

        MOV  DX,Com1+1
        MOV  AL,00h         ; G•i byte cao → 000Ch: xác ••nh
        OUT  DX,AL          ; t•c •• truy•n 9600bps

        MOV  DX,Com1+3      ; LCR = 0000 0011B
        MOV  AL,03h         ; DLAB = 0, SBCB = 0 → c•m Break
        OUT  DX,AL          ; PS = 000 → no parity
                            ; STB = 0 → 1 stop bit
                            ; WLS = 11 → 8 bit d• li•u

        MOV  DX,Com1+4      ; Tác ••ng ••n DTR và RTS
        MOV  AL,03h         ; MCR = 0000 0011b → DTR=RTS = 1
        OUT  DX,AL          ; → ngõ DTR và RTS c•a c•ng n•i
                            ; ti•p = 0

        MOV  DX,21h         ; Ki•m tra tr•ng thái ng•t
        IN   AL,DX          ; D7 – D0 xác ••nh các IRQi
        MOV  Mask_int,AL    ; =0: cho phép, =1: c•m

        AND  AL,0EFh        ; = 1110 1111b → cho phép IRQ4
        OUT  DX,AL          ; → cho phép COM1

        MOV  AL,01h         ; IER = 0000 0001b → cho phép
        MOV  DX,Com1+1      ; ng•t khi RBR ••y
        OUT  DX,AL

        MOV  AH,09h
        LEA  Dx,Msg
        INT  21h

Lap:
        MOV  AH,0Bh
        INT  21h
        CMP  AL,0FFh
        JE   Exit

        MOV  AL,bufferin
        CMP  AL,bufferout
        JE   Lap
        MOV  AL,buffer[bufferout]
        MOV  char,AL
        INC  bufferout
        MOV  AL,bufferout
        CMP  AL,251
        JNE  Next
        MOV  bufferout,0
Next:
```

```
        MOV DL,char         ; Xu•t giá tr• ra màn hình
        MOV  AH,02h
        INT  21h

        MOV  AL,char        ; Xu•t ra c•ng n•i ti•p
        MOV  DX,Com1
        OUT  DX,AL
        JMP  Lap

Exit:
        MOV  AL,Mask_int
        OUT  21h,AL         ; Khôi ph•c tr•ng thái ng•t

        MOV  DX,Off_com
        MOV  BX,Seg_com
        MOV  DS,BX
        MOV  AH,35h         ;Khôi ph•c vector ng•t
        MOV  AL,Com_int
        INT  21h


        MOV  AH,4Ch
        INT  21h
Main ENDP

Com_ISR  PROC
        MOV  DX,Com1+5      ; ••c n•i dung LSR
        IN   AL,DX
        AND  AL,1           ; N•u D0 = 1 thì có d• li•u
        JZ   exit_ISR

        MOV  DX,Com1
        IN   AL,DX
        MOV  buffer[bufferin],AL
        INC  bufferin
        MOV  AL,bufferin
        CMP  AL,251
        JNE  Exit_ISR
        MOV  bufferin,0
Exit_ISR:
        MOV  AL,20h         ; Báo cho PIC k•t thúc ng•t
        OUT  20h,AL
        IRET
Com_ISR  ENDP
END Main
```

4. Truyền thông nối tiếp dùng ActiveX

4.1. Mô tả

Việc truyền thông nối tiếp trên Windows được thực hiện thông qua một ActiveX có sẵn là Microsoft Comm Control.. ActiveX này dược lưu trữ trong file MSCOMM32.OCX. Quá trình này có hai khả năng thực hiện điều khiển trao đổi thông tin:

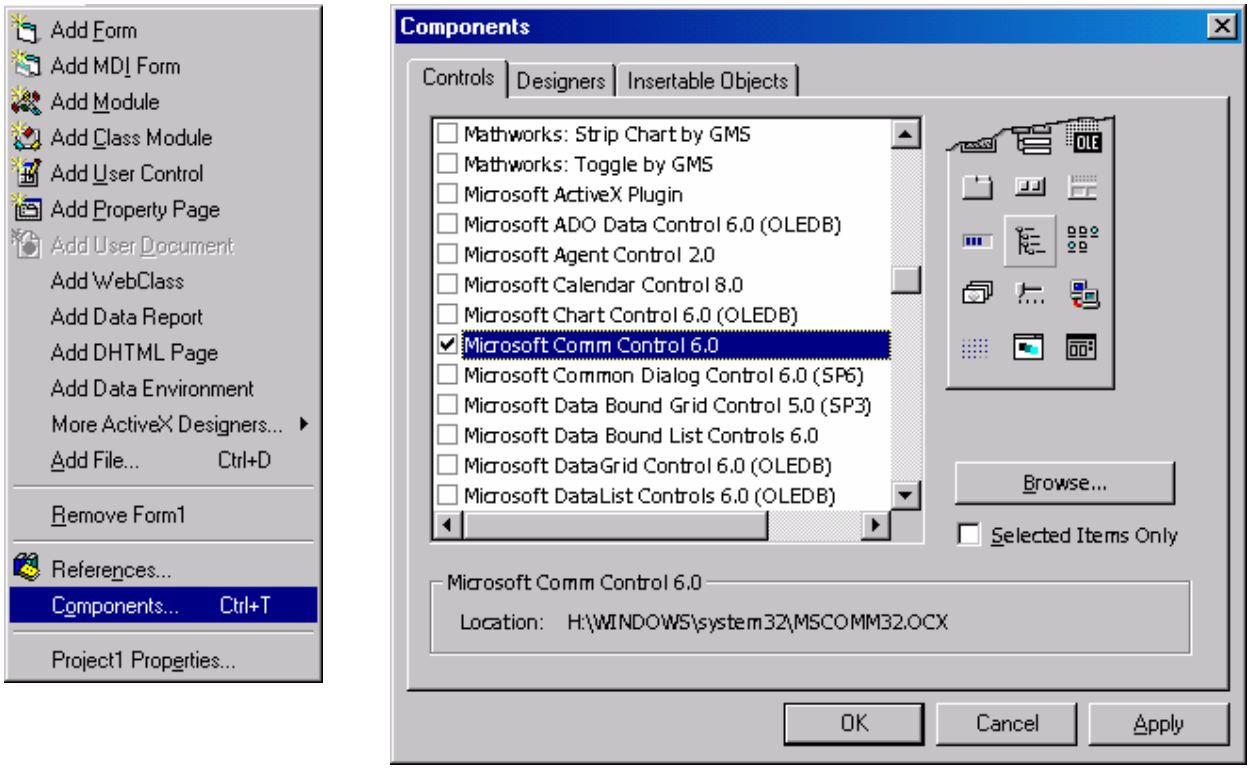
- **Điều khiển sự kiện:**

Truyền thông điều khiển sự kiện là phương pháp tốt nhất trong quá trình điều khiển việc trao đổi thông tin. Quá trình điều khiển thực hiện thông qua sự kiện OnComm.

- **Hỏi vòng:**

Quá trinh điều khiển bằng phương pháp hỏi vòng thực hiện thông qua kiểm tra các giá trị của thuộc tính CommEvent sau một chu kỳ nào đó để xác định xem có sự kiện nào xảy ra hay không. Thông thường phương pháp này sử dụng cho các chương trình nhỏ.

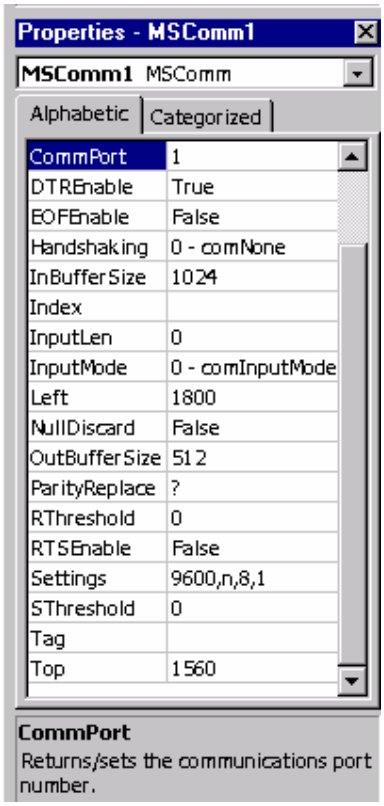
ActiveX MsComm được bổ sung vào một Visual Basic Project thông qua menu **Project > Components**:



Hình 4.5 – Bổ sung đối tượng MsComm vào VBP

Biểu tượng của MsComm:  và các thuộc tính cơ bản mô tả như sau:

Thuộc tính	Mô tả
CommPort	Số thứ tự cổng truyền thông
Input	Nhận ký tự từ bộ đệm
Output	Xuất ký tự ra cổng nối tiếp
PortOpen	Mở / đóng cổng
Settings	Xác định các tham số truyền



Hình 4.6 – Các thuộc tính của đối tượng MSComm

4.2. Các thuộc tính

❖ **Settings:**

Xác định các tham số cho cổng nối tiếp. Cú pháp:

```
MSComm1.Settings = ParamString
```

MSComm1: tên đối tượng

ParamString: là một chuỗi có dạng như sau: "BBBB,P,D,S"

BBBB: tốc độ truyền dữ liệu (bps) trong đó các giá trị hợp lệ là:

110	2400	38400
300	9600 (măc định)	56000
600	14400	188000
1200	19200	256000

P: kiểm tra chẵn lẻ, với các giá trị:

Giá trị	Mô tả
O	Odd (kiểm tra lẻ)
E	Even (kiểm tra chẵn)
M	Mark (luôn bằng 1)
S	Space (luôn bằng 0)
N	Không kiểm tra

D: số bit dữ liệu (4, 5, 6, 7 hay 8), mặc định là 8 bit

S: số bit stop (1, 1.5, 2)

VD:

`MSComm1.Settings = "9600,O,8,1"` sẽ xác định tốc độ truyền 9600bps, kiểm tra parity chẵn với 1 bit stop và 8 bit dữ liệu.

❖ **CommPort:**

Xác định số thứ tự của cổng truyền thông, cú pháp:

`MSComm1.CommPort = PortNumber`

PortNumber là giá trị nằm trong khoảng từ 1 → 99, mặc định là 1.

VD:

`MSComm1.CommPort = 1` xác định sử dụng COM1

❖ **PortOpen:**

Đặt trạng thái hay kiểm tra trạng thái đóng / mở của cổng nối tiếp. Nếu dùng thuộc tính này để mở cổng nối tiếp thì phải sử dụng trước 2 thuộc tính Settings và CommPort. Cú pháp:

`MSComm1.PortOpen = True | False`

Giá trị xác định là True sẽ thực hiện mở cổng và False để đóng cổng đồng thời xoá nội dung của các bộ đệm truyền, nhận.

VD: Mở cổng COM1 với tốc độ truyền 9600 bps

```
MSComm1.Settings = "9600,N,8,1"
MSComm1.CommPort = 1
MSComm1.PortOpen = True
```

❖ **Các thuộc tính nhận dữ liệu:**

Input: nhận một chuỗi ký tự và xoá khỏi bộ đệm. Cú pháp:

`InputString = MSComm1.Input`

Thuộc tính này kết hợp với InputLen để xác định số ký tự đọc vào. Nếu InputLen = 0 thì sẽ đọc toàn bộ dữ liệu có trong bộ đệm.

InBufferCount: số ký tự có trong bộ đệm nhận. Cú pháp:

`Count = MSComm1.InBufferCount`

Thuộc tính này cùng dược dùng để xoá bộ đệm nhận bằng cách gán giá trị 0.

`MSComm1.InBufferCount = 0`

InBufferSize: đặt và xác định kích thước bộ đệm nhận (tính bằng byte). Cú pháp:

`MSComm1.InBufferCount = NumByte`

Giá trị măc định là 1024 byte. Kích thước bộ đệm này phải đủ lớn để tránh tình trạng mất dữ liệu.

VD: Đọc toàn bộ nội dung trong bộ đệm nhận nếu có dữ liệu

```
MSComm1.InputLen = 0
If MSComm1.InBufferCount <> 0 Then
     InputString = MSComm1.Input
End If
```

❖ **Các thuộc tính xuất dữ liệu:**

Bao gồm các thuộc tính **Output**, **OutBufferCount** và **OutBufferSize**, chức năng của các thuộc tính này giống như các thuộc tính nhập.

❖ **CDTimeout:**

Đặt và xác định khoảng thời gian lớn nhất (tính bằng ms) từ lúc phát hiện sóng mang cho đến lúc có dữ liệu. Nếu quá khoảng thời gian này mà vẫn chưa có dữ liệu thì sẽ gán thuộc tính CommEvent là CDTO (Carrier Detect Timeout Error) và tạo sự kiện OnComm. Cú pháp:

```
MSComm1.CDTimeout = NumTime
```

❖ **DSRTimeout:**

Xác định thời gian chờ tín hiệu DSR trước khi xảy ra sự kiện OnComm.

❖ **CTSTimeout:**

Đặt và xác định khoảng thời gian lớn nhất (tính bằng ms) đợi tín hiệu CTS trước khi đặt thuộc tính CommEvent là CTSTO và tạo sự kiện OnComm. Cú pháp:

```
MSComm1.CTSTimeout = NumTime
```

❖ **CTSHolding:**

Xác định đã có tín hiệu CTS hay chưa, tín hiệu này dùng cho quá trình bắt tay bằng phần cứng (cho biết DCE sẵn sàng nhận dữ liệu), trả về giá trị True hay False.

❖ **DSRHolding:**

Xác định trạng thái DSR (báo hiệu sự tồn tại của DCE), trả về giá trị True hay False.

❖ **CDHolding:**

Xác định trạng thái CD, trả về giá trị True hay False.

❖ **DTREnable:**

Đặt hay xoá tín hiệu DTR để báo sự tồn tại của DTE. Cú pháp:

```
MSComm1.DTREnable = True | False
```

❖ **RTSEnable:**

Đặt hay xoá tín hiệu RTS để yêu cầu truyền dữ liệu đến DTE. Cú pháp:

```
MSComm1.RTSEnable = True | False
```

❖ **NullDiscard:**

Cho phép nhận các ký tự NULL (rỗng) hay không (= True: cấm). Cú pháp:

```
MSComm1.NullDiscard = True | False
```

❖ **SThreshold:**

Số byte trong bộ đệm truyền làm phát sinh sự kiện OnComm. Nếu giá trị này bằng 0 thì sẽ không tạo sự kiện OnComm. Cú pháp:

```
MSComm1.SThreshold = NumChar
```

❖ **HandShaking:**

Chọn giao thức bắt tay khi thực hiện truyền dữ liệu. Cú pháp:

```
MSComm1.HandShaking = Protocol
```

Các giao thức truyền bao gồm:

Protocol	Giá trị	Mô tả
ComNone	0	Không băt tay (mặc định)
ComXon/Xoff	1	Bắt tay phần mềm (Xon/Xoff)
ComRTS	2	Bắt tay phần cứng (RTS/CTS)
ComRTSXon/Xoff	3	Bắt tay phần cứng và phàn mềm

❖ **CommEvent:**

Trả lại các lỗi truyền thonog hay sự kiện xảy ra tại cổng nối tiếp

Các sự kiện:

Sự kiện	Giá trị	Mô tả
ComEvSend	1	Đã truyền ký tự
ComEvReceive	2	Khi có ký tự trong bộ đệm nhận
ComEvCTS	3	Có thay đổi trên CTS (Clear To Send)
ComEvDSR	4	Có thay đổi trên DSR (Data Set Ready)
ComEvCD	5	Có thay đổi trên CD (Carrier Detect)
ComEvRing	6	Phát hiện chuông
ComEvEOF	7	Nhận ký tự kết thúc file

Các lỗi truyền thông:

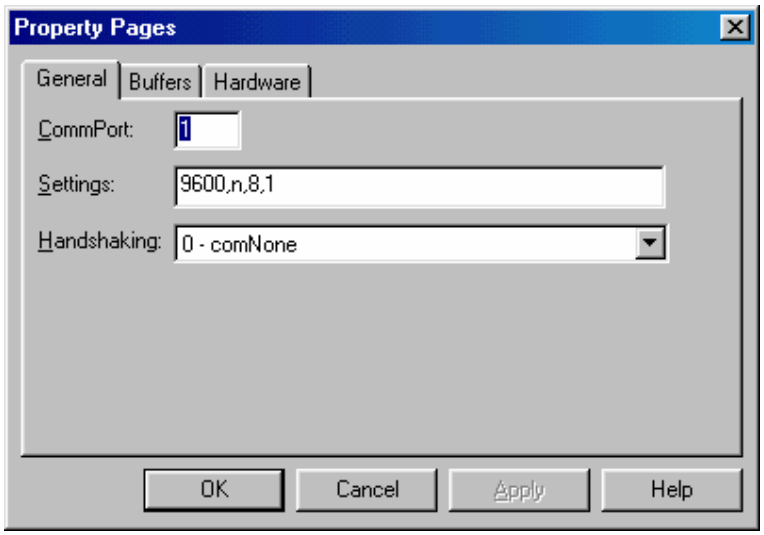
Lỗi	Giá trị	Mô tả
ComBreak	1001	Nhận tín hiệu Break
ComCTSTO	1002	Carrier Detect Timeout
ComFrame	1004	Lỗi khung
ComOver	1006	Phần cứng không đọc ký tự trước khi gởi ký tự kế
ComCDTO	1007	Carrier Detect Timeout
ComRxOver	1008	Tràn bộ đệm nhận
ComRxParity	1009	Lỗi parity
ComTxFull	1010	Tràn bộ đệm truyền

4.3. Sự kiện OnComm

Sự kiện OnComm xảy ra bất cứ khi nào giá trị của thuộc tính CommEvent thay đổi. Các thuộc tính RThreshold và SThreshold = 0 sẽ cấm sự kiện OnComm khi thực hiện nhận hay gởi dữ liệu. Thông thường, SThreshold = 0 và RThreshold = 1.

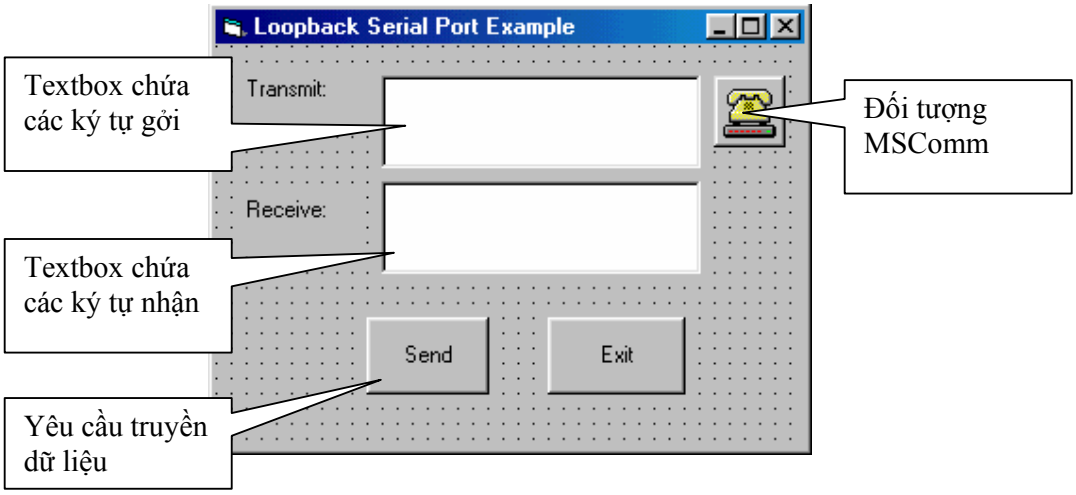
Một chương trình truyền nhận đơn giản thực hiện bằng cách nối chân TxD với RxD của cổng COM1 (loopback). Phương pháp này dùng để kiểm tra cổng nối tiếp.

Thuộc tính cơ bản của cổng nối tiếp:



Hình 4.7 – Các thuộc tính cơ bản của MSComm

Cửa sổ chương trình thực thi:



Hình 4.8 – Cửa sổ chương trình loopback

Chương trình nguồn:

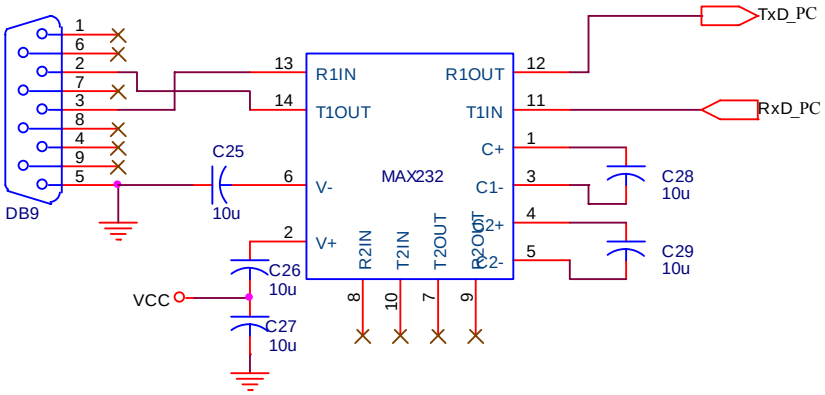
```
VERSION 5.00
Object          =          "{648A5603-2C6E-101B-82B6-
000000000014}#1.1#0"; "MSCOMM32.OCX"
```

```
      Begin VB.Form Form1
         Caption              =     "Loopback  Serial  Port
Example"
         ClientHeight    =   3195
         ClientLeft      =   60
         ClientTop       =   345
         ClientWidth     =   4680
         LinkTopic       =   "Form1"
         ScaleHeight     =   3195
         ScaleWidth      =   4680
         StartUpPosition =   3  'Windows Default
         Begin VB.CommandButton cmdExit
            Caption         =   "Exit"
            Height          =   615
            Left            =   2640
            TabIndex        =   5
            Top             =   2160
            Width           =   1095
         End
         Begin VB.CommandButton cmdSend
            Caption         =   "Send"
            Height          =   615
            Left            =   1200
            TabIndex        =   4
            Top             =   2160
            Width           =   975
         End
         Begin VB.TextBox txtReceive
            Height          =   735
            Left            =   1320
            Locked          =   -1  'True
            TabIndex        =   3
            Top             =   1080
            Width           =   2535
         End
         Begin VB.TextBox txtTransmit
            Height          =   735
            Left            =   1320
            TabIndex        =   0
            Top             =   240
            Width           =   2535
         End
         Begin MSCommLib.MSComm MSComm1
            Left            =   3960
            Top             =   240
            _ExtentX        =   1005
            _ExtentY        =   1005
```

```
      _Version        =   393216
      DTREnable       =   -1  'True
      RThreshold      =   1
   End
   Begin VB.Label Label2
      Caption         =   "Receive:"
      Height          =   375
      Left            =   240
      TabIndex        =   2
      Top             =   1200
      Width           =   855
   End
   Begin VB.Label Label1
      Caption         =   "Transmit:"
      Height          =   375
      Left            =   240
      TabIndex        =   1
      Top             =   240
      Width           =   975
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub cmdExit_Click()
MSComm1.PortOpen = False    'Đóng cổng
End
End Sub
Private Sub cmdSend_Click()
MSComm1.Output = Trim(txtTransmit.Text)'Gởi dữ liệu
End Sub
Private Sub Form_Load()
MSComm1.CommPort = 1             'COM1
MSComm1.Settings = "9600,n,8,1"  'Tốc độ 9600bps
MSComm1.PortOpen = True          ' Mở cổng
End Sub
Private Sub MSComm1_OnComm()
If (MSComm1.CommEvent = comEvReceive) Then
txtReceive.Text = txtReceive.Text + MSComm1.Input
End If
End Sub
```

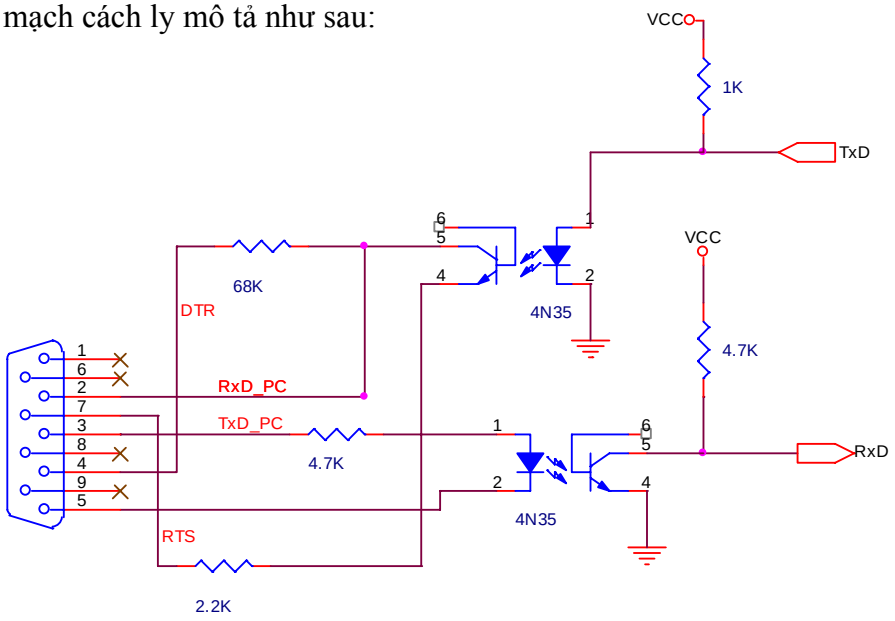
5. Giao tiếp với vi điều khiển

Khi thực hiện giao tiếp với vi điều khiển, ta phải dùng thêm mạch chuyển mức logic từ TTL → 232 và ngược lại. Các vi mạch thường sử dụng là MAX232 của Maxim hay DS275 của Dallas. Mạch chuyển mức logic mô tả như sau:



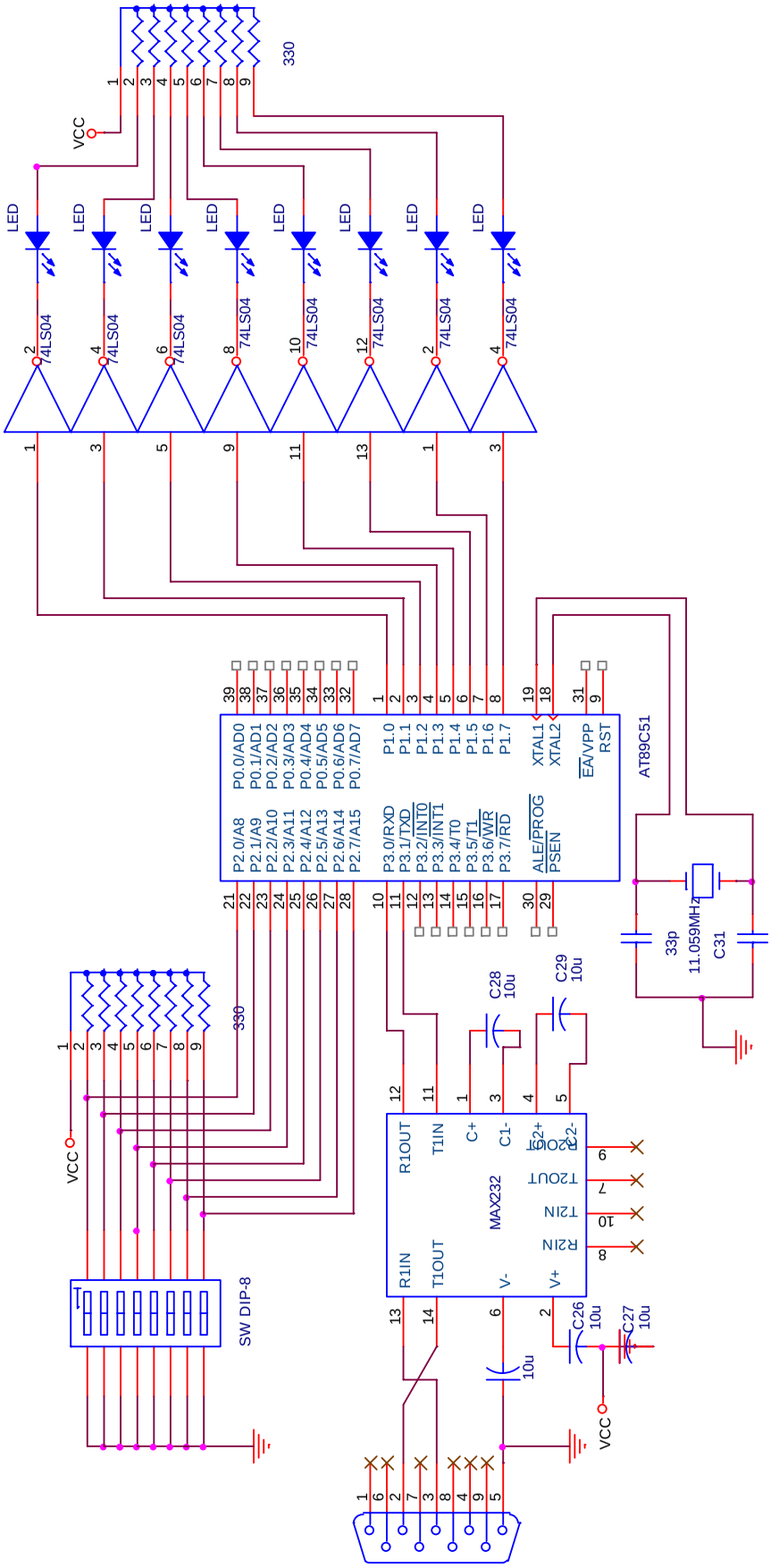
Hình 4.9 – Mạch chuyển mức logic TTL ↔ RS232

Tuy nhiên, khi sử dụng mạch chuyển mức logic dùng các vi mạch thì đòi hỏi phải dùng chung GND giữa máy tính và vi mạch → có khả năng làm hỏng cổng nối tiếp khi xảy ra hiện tượng chập mạch ở mạch ngoài. Do đó, ta có thể dùng thêm opto 4N35 để cách ly về điện. Sơ đồ mạch cách ly mô tả như sau:



Hình 4.10 – Mạch chuyển mức logic TTL ↔ RS232 cách ly

Khi giao tiếp, vi điều khiển chính là một DTE nên sẽ nối RxD của máy tính với TxD của vi điều khiển và ngược lại. Mạch kết nối đơn giản giữa vi điều khiển và máy tính như sau:

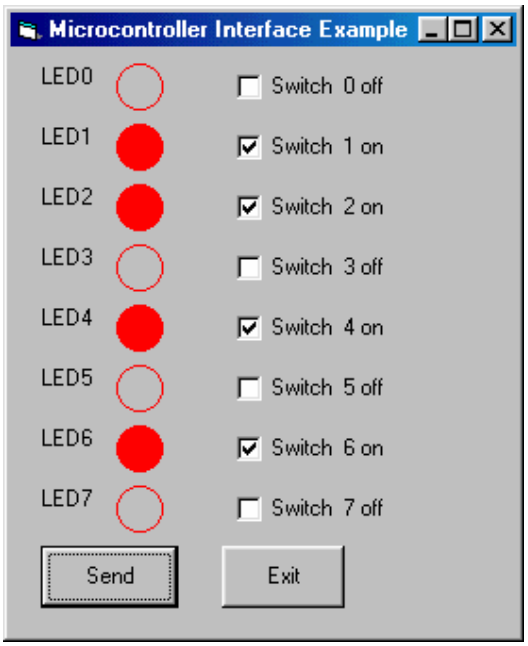


Hình 4.11 – Kết nối với vi điều khiển

Chương trình nguồn cho vi điều khiển AT89C51:

```
     MOV  TMOD,#20h
     MOV  SCON,#52h; Truyền 8 bit dữ liệu, no parity
     MOV  TH1,#(-3); Tốc độ truyền 9600 bps
     MOV  TL1,#(-3)
     SETB TR1

Receive:
     JNB  RI,Transmit    ; Có dữ liệu hay không
     CLR  RI
     MOV  A,SBUF         ; Nếu có thì xuất ra LED
     MOV  P1,A
Transmit:
     JNB  TI,Receive     ; Đã truyền xong chưa
     CLR  TI
     MOV  A,P2           ; Nếu xong thì truyền trạng thái
     MOV  SBUF,A         ; của công tắc SW DIP-8
     JMP  Receive
```

Giao diện của chương trình trên máy tính:



Hình 4.12 – Chương trình giao tiếp với vi điều khiển

Chương trình nguồn:

```
     VERSION 5.00
     Object          =          "{648A5603-2C6E-101B-82B6-
000000000014}#1.1#0"; "MSCOMM32.OCX"
```

```
      Begin VB.Form Form1
         Caption            =    "Microcontroller Interface
Example"
         ClientHeight    =   4665
         ClientLeft      =   60
         ClientTop       =   345
         ClientWidth     =   4020
         LinkTopic       =   "Form1"
         ScaleHeight     =   4665
         ScaleWidth      =   4020
         StartUpPosition =   3  'Windows Default
         Begin VB.CheckBox chkSW
            Height          =   375
            Index           =   7
            Left            =   1800
            TabIndex        =   17
            Top             =   3480
            Width           =   1575
         End
         Begin VB.CheckBox chkSW
            Height          =   375
            Index           =   6
            Left            =   1800
            TabIndex        =   16
            Top             =   3000
            Width           =   1575
         End
         Begin VB.CheckBox chkSW
            Height          =   375
            Index           =   5
            Left            =   1800
            TabIndex        =   15
            Top             =   2520
            Width           =   1575
         End
         Begin VB.CheckBox chkSW
            Height          =   375
            Index           =   4
            Left            =   1800
            TabIndex        =   14
            Top             =   2040
            Width           =   1575
         End
         Begin VB.CheckBox chkSW
            Height          =   375
            Index           =   3
            Left            =   1800
```

```
   TabIndex        =   13
   Top             =   1560
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   2
   Left            =   1800
   TabIndex        =   12
   Top             =   1080
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   1
   Left            =   1800
   TabIndex        =   11
   Top             =   600
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   0
   Left            =   1800
   TabIndex        =   10
   Top             =   120
   Width           =   1575
End
Begin VB.CommandButton cmdExit
   Caption         =   "Exit"
   Height          =   495
   Left            =   1680
   TabIndex        =   9
   Top             =   3960
   Width           =   975
End
Begin MSCommLib.MSComm MSComm1
   Left            =   3360
   Top             =   3960
   _ExtentX        =   1005
   _ExtentY        =   1005
   _Version        =   393216
   DTREnable       =   -1  'True
   RThreshold      =   1
End
Begin VB.CommandButton cmdSend
   Caption         =   "Send"
```

```
      Height          =   495
      Left            =   240
      TabIndex        =   8
      Top             =   3960
      Width           =   1095
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED7"
      Height          =   375
      Index           =   7
      Left            =   240
      TabIndex        =   7
      Top             =   3480
      Width           =   1095
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED6"
      Height          =   375
      Index           =   6
      Left            =   240
      TabIndex        =   6
      Top             =   3000
      Width           =   975
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED5"
      Height          =   375
      Index           =   5
      Left            =   240
      TabIndex        =   5
      Top             =   2520
      Width           =   975
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED4"
      Height          =   375
      Index           =   4
      Left            =   240
      TabIndex        =   4
      Top             =   2040
      Width           =   975
   End
   Begin VB.Label lblLED
```

```
   BackStyle       =   0  'Transparent
   Caption         =   "LED3"
   Height          =   375
   Index           =   3
   Left            =   240
   TabIndex        =   3
   Top             =   1560
   Width           =   975
End
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED2"
   Height          =   375
   Index           =   2
   Left            =   240
   TabIndex        =   2
   Top             =   1080
   Width           =   975
End
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED1"
   Height          =   375
   Index           =   1
   Left            =   240
   TabIndex        =   1
   Top             =   600
   Width           =   975
End
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED0"
   Height          =   375
   Index           =   0
   Left            =   240
   TabIndex        =   0
   Top             =   120
   Width           =   975
End
Begin VB.Shape shpLED
   BorderColor     =   &H000000FF&
   FillColor       =   &H000000FF&
   FillStyle       =   0  'Solid
   Height          =   375
   Index           =   7
   Left            =   840
   Shape           =   3  'Circle
```

```
   Top             =   3480
   Width           =   375
End
Begin VB.Shape shpLED
   BorderColor     =   &H000000FF&
   FillColor       =   &H000000FF&
   FillStyle       =   0  'Solid
   Height          =   375
   Index           =   6
   Left            =   840
   Shape           =   3  'Circle
   Top             =   3000
   Width           =   375
End
Begin VB.Shape shpLED
   BorderColor     =   &H000000FF&
   FillColor       =   &H000000FF&
   FillStyle       =   0  'Solid
   Height          =   375
   Index           =   5
   Left            =   840
   Shape           =   3  'Circle
   Top             =   2520
   Width           =   375
End
Begin VB.Shape shpLED
   BorderColor     =   &H000000FF&
   FillColor       =   &H000000FF&
   FillStyle       =   0  'Solid
   Height          =   375
   Index           =   4
   Left            =   840
   Shape           =   3  'Circle
   Top             =   2040
   Width           =   375
End
Begin VB.Shape shpLED
   BorderColor     =   &H000000FF&
   FillColor       =   &H000000FF&
   FillStyle       =   0  'Solid
   Height          =   375
   Index           =   3
   Left            =   840
   Shape           =   3  'Circle
   Top             =   1560
   Width           =   375
End
```

```
      Begin VB.Shape shpLED
         BorderColor     =   &H000000FF&
         FillColor       =   &H000000FF&
         FillStyle       =   0  'Solid
         Height          =   375
         Index           =   2
         Left            =   840
         Shape           =   3  'Circle
         Top             =   1080
         Width           =   375
      End
      Begin VB.Shape shpLED
         BorderColor     =   &H000000FF&
         FillColor       =   &H000000FF&
         FillStyle       =   0  'Solid
         Height          =   375
         Index           =   1
         Left            =   840
         Shape           =   3  'Circle
         Top             =   600
         Width           =   375
      End
      Begin VB.Shape shpLED
         BorderColor     =   &H000000FF&
         FillColor       =   &H000000FF&
         FillStyle       =   0  'Solid
         Height          =   375
         Index           =   0
         Left            =   840
         Shape           =   3  'Circle
         Top             =   120
         Width           =   375
      End
   End
   Attribute VB_Name = "Form1"
   Attribute VB_GlobalNameSpace = False
   Attribute VB_Creatable = False
   Attribute VB_PredeclaredId = True
   Attribute VB_Exposed = False
   Private Sub cmdExit_Click()
   If MSComm1.PortOpen Then
       MSComm1.PortOpen = False
   End If
   End
   End Sub

   Private Sub cmdSend_Click()
```

```
      Dim t As Integer
      Dim i As Integer
      t = 0
      For i = 0 To 7
         t = t + (2 ^ i) * (1 - shpLED(i).FillStyle)
      Next i
      MSComm1.Output = Chr(t)
      End Sub

      Private Sub Form_Load()
      MSComm1.Settings = "9600,N,8,1"
      MSComm1.CommPort = 1
      MSComm1.PortOpen = True
      End Sub

      Private Sub lblLED_Click(Index As Integer)
shpLED(Index).FillStyle = 1 - shpLED(Index).FillStyle
      End Sub

      Private Sub MSComm1_OnComm()
      Dim t As String
      Dim n As Integer
      Dim i As Integer
      If MSComm1.CommEvent = comEvReceive Then
          n = Asc(MSComm1.Input)
          For i = 0 To 7
              chkSW(i).Value = n Mod 2
              If chkSW(i).Value = 0 Then
                  chkSW(i).Caption = "Switch " & Str(i) &
" off"
              Else
                  chkSW(i).Caption = "Switch " & Str(i) &
" on"
              End If
              n = Fix(n / 2)
          Next i
      End If
      End Sub
```

6. Giao tiếp với MODEM

6.1. Giao tiếp

Quá trình trao đổi dữ liệu giữa máy tính và Modem được thực hiện theo cơ chế bắt tay phần cứng hay phần mềm.

- **Bắt tay phần cứng:** máy tính muốn truyền dữ liệu thì cho RTS = 1 và chờ Modem trả lời bằng tín hiệu CTS. Ngược lại, Modem muốn truyền dữ liệu thì cho DSR = 1 và chờ tín hiệu DTR từ máy tính.
- **Bắt tay phần mềm:** dùng ký tự Xon (Ctrl-S) và Xoff (Ctrl-Q) để bắt đầu truyền hay kết thúc truyền.

Các giao thức truyền dữ liệu trên Modem:

- **XModem:** chia thành khối 128 byte, mỗi khối chèn thêm CRC 4 byte.
- **YModem:** khối 1024 byte.
- **ZModem:** khối có kích thước thay đổi tuỳ theo đường truyền.

Quy tắc truyền lệnh trên Modem:

- Mỗi dòng lệnh của modem bắt đầu bằng ký tự AT, ngoại trừ lệnh A/ và +++.
- Dòng lệnh có thể chứa nhiều lệnh.
- Kết thúc lệnh bằng ký tự Enter (mã ASCII là 13) ngoại trừ lệnh A/ và +++.
- Dòng lệnh cuối cùng được lưu trong modem. Có thể dùng lệnh A/ để thực hiện lại lệnh này.
- Thông báo kết quả thực hiện lệnh của modem có thể ở dạng từ chữ hay số(giá trị mặc định là chữ). Có thể sử dụng lệnh V để lựa chọn dạng thông báo là chữ hay số.
- Để hoạt động đúng, modem cần có các thông số xác định. Nếu không có sự thay đổi cần thiết, modem hoạt động theo giá trị mặc định(default). Nếu thông số trong lệnh bị bỏ qua, giá trị thông số mặc định là 0.

6.2. Các lệnh cơ bản của Modem

Lệnh	Mô tả
+++	Chuyển Modem sang chế độ lệnh
A/	Lặp lại lệnh trước
A	Cho phép kết nối và phát tín hiệu sóng mang. Modem sẽ báo tín hiệu CONNECT nếu thu được tín hiêu sóng mang từ modem đầu cuối. Nếu không thu được sóng mang, modem sẽ gác máy và thông báo NO CARRIER
DPn	Quay số điện thoại n dạng xung
DTn	Quay số điện thoại n dạng tone
H0	Gác máy
H1	Nhấc máy
O0	Chuyển về chế độ dữ liệu
O1	Chế độ điều chỉnh Modem
Q0	Cho phép Modem gởi thông báo đến DTE (mặc định)
Q1	Cấm Modem gởi thông báo
Q2	Gởi thông báo khi Modem chủ động kết nối, không gởi khi Modem nhận cuộc gọi
V0	Nhận thông báo dạng số

V1	Nhận thông báo dạng ký tự (mặc định)
Sn = V	Nạp giá trị V vào thanh ghi Sn S0 = V: chờ V hồi chuông trước khi trả lời, V = 0 – 255 (mặc định V = 0: không trả lời) S6 = V: chờ V giây trước khi quay số (mặc định V = 2) S7 = V: chờ V giây kể từ lúc gọi đến lúc nhận được tín hiệu, nếu không sẽ thông báo lỗi
Sn?	Đọc nội dung thanh ghi Sn
Z0	Reset Modem về cấu hình 0
Z1	Reset Modem về cấu hình 1
L0, L1, L2, L3	Âm lượng loa Modem
M0	Tắt loa
M1	Mở loa cho đến khi nhận dược sóng mang (măc định)
M2	Mở loa
M3	Tắt loa khi quay số và nhận sóng mang

6.3. Các thanh ghi thông dụng trên modem

Thanh ghi S0: xác định số hồi chuông nhận được mà sau đó modem sẽ trả lời một cách tự động. Giá trị trong thanh ghi này có thể thay đổi trong khoảng từ 0-255. mặc định giá trị là 0 (không trả lời).

Thanh ghi S1: Thanh ghi S1 chỉ có tác dụng khi thanh ghi S0 khác 0, dùng để đếm số hồi chuông thu được.

Thanh ghi S2: xác định giá trị thập phân của các ký tự (mã ASSCII) được dùng làm ký tự thoát, Giá trị mặc định là 43(+)

Thanh ghi S3: xác định ký tự được dùng để kết thúc một dòng lệnh, mặc nhiên là 13 (tương ứng là Enter)

Thanh ghi S4: xác định ký tự xuống dòng sau ký tự kết thúc, giá trị mặc nhiên là 10 (line feed)

Thanh ghi S5: xác định phím xoá lui, giá trị mặc nhiên là 8 (backspace)

Thanh ghi S6: xác định thời gian đợi sau khi truy cập đường điện thoại và trước khi tiến hành quoay digit đầu tiên trong một lệnh quay số. Đây là thời gian trì hoãn cho phép để dial tone cung cấp từ đường truyền. Giá trị mặc nhiên và tối thiểu là 2s.

Thanh ghi S7: xác định thời gian mà modem đợi tín hiệu sóng mang trước khi gác máy. Giá trị mặc định là 30s.

Thanh ghi S8: xác định thời gian tạm dừng cho mỗi dấu phẩy ',' trong chuỗi lệnh quay số. Giá trị mặc định là 2s

Thanh ghi S9: xác định thời gian mà tín hiệu sóng mang phải hiện diện để modem có thể nhận biết được, giá trị mặc định là 600ms. Giá trị này nếu quá lớn sẽ gây lỗi trong dữ liệu truyền.

Thanh ghi S10: xác định thời gian cho phép tín hiệu sóng mang có thể biến mất trong chốc lát nào đó mà không cắt cuộc nối. Ổn định trong khoảng 100-25500ms, giá trị mặc nhiên tùy vào khả năng chống nhiễu của từng modem, thường là 700ms.

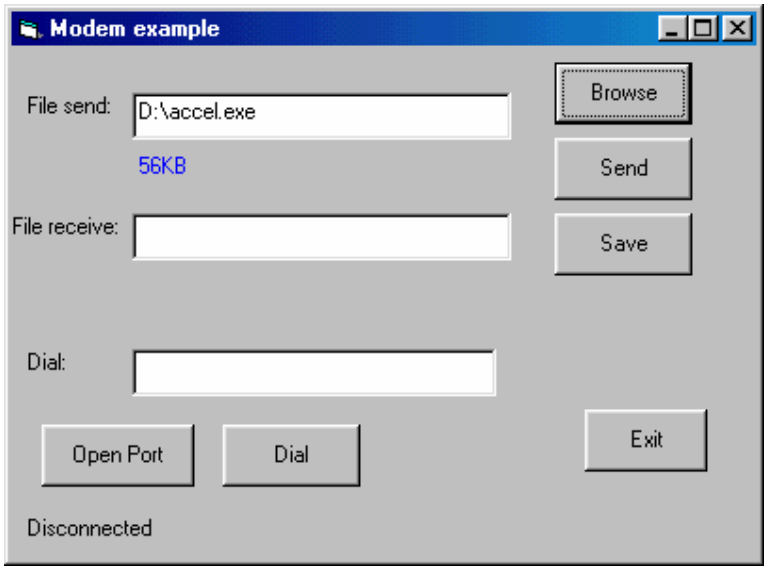
Thanh ghi S11: xác định tốc độ quay số khi sử dụng phương pháp quay số tone, giá trị mặc nhiên tùy vào modem, thường vào khoảng 70ms.

Thanh ghi S12: xác định thời gian an toàn khi truy nhập vào ký tự thoát (+++). Nếu giá trị nhỏ quá có thể nhập không kịp, giá trị lớn quá so với tốc độ nhập cũng không thể thoát được.

6.4. Các thông báo của Modem

Dạng ký tự	Dạng số	Ý nghĩa
OK	0	Lệnh thành công
CONNECT	1	Kết nối 300 bps
RING	2	Có tín hiệu chuông
NO CARRIER	3	Không có sóng mang
ERROR	4	Lỗi: nhận lệnh không giá trị, sai kiểm tra, hàng lệnh quá dài
CONNECT 1200	5	Kết nối 1200bps
NO DIAL TONE	6	Không có âm hiệu mời quay số
BUSY	7	Máy bận
NO ANSWER	8	Không có tín hiệu trả lời
CONNECT 2400	10	Kết nối 2400bps
CONNECT 4800	11	Kết nối 4800bps
CONNECT 9600	12	Kết nối 9600bps
CONNECT 14400	13	Kết nối 14400bps
CONNECT 19200	14	Kết nối 19200bps
CONNECT 16800	15	Kết nối 16800bps
CONNECT 57600	18	Kết nối 57600bps
CONNECT 7200	24	Kết nối 7200bps
CONNECT 12000	25	Kết nối 12000bps
CONNECT 28800	32	Kết nối 28800bps
CONNECT 115200	33	Kết nối 115200bps
CARRIER 300	40	Phát hiện sóng mang
CARRIER 9600	50	Phát hiện sóng mang
CARRIER 28800	58	Phát hiện sóng mang

Ví dụ lập trình điều khiển Modem như sau:



Hình 4.13 – Giao tiếp và điều khiển Modem

Chương trình nguồn:

```
     VERSION 5.00
     Object          =          "{648A5603-2C6E-101B-82B6-
000000000014}#1.1#0"; "MSCOMM32.OCX"
     Object          =          "{F9043C88-F6F2-101A-A3C9-
08002B2F49FB}#1.2#0"; "COMDLG32.OCX"
     Begin VB.Form frmModem
        Caption         =   "Modem example"
        ClientHeight    =   4065
        ClientLeft      =   60
        ClientTop       =   345
        ClientWidth     =   5925
        LinkTopic       =   "Form1"
        ScaleHeight     =   4065
        ScaleWidth      =   5925
        StartUpPosition =   3  'Windows Default
        Begin VB.CommandButton cmdSave
           Caption         =   "Save"
           Height          =   495
           Left            =   4320
           TabIndex        =   14
           Top             =   1320
           Width           =   1095
        End
        Begin VB.TextBox txtReceive
           Height          =   375
           Left            =   960
```

```
   TabIndex        =   12
   Top             =   1320
   Width           =   3015
End
Begin VB.Timer Timer1
   Enabled         =   0   'False
   Interval        =   1000
   Left            =   4920
   Top             =   2400
End
Begin VB.CommandButton cmdExit
   Caption         =   "Exit"
   Height          =   495
   Left            =   4560
   TabIndex        =   10
   Top             =   2880
   Width           =   975
End
Begin VB.TextBox txtDial
   Height          =   375
   Left            =   960
   TabIndex        =   7
   Top             =   2400
   Width           =   2895
End
Begin VB.CommandButton cmdDial
   Caption         =   "Dial"
   Height          =   495
   Left            =   1680
   TabIndex        =   5
   Top             =   3000
   Width           =   1095
End
Begin VB.CommandButton cmdSend
   Caption         =   "Send"
   Height          =   495
   Left            =   4320
   TabIndex        =   4
   Top             =   720
   Width           =   1095
End
Begin VB.CommandButton cmdOpen
   Caption         =   "Open Port"
   Height          =   495
   Left            =   240
   TabIndex        =   3
   Top             =   3000
```

```
      Width           =   1215
   End
   Begin VB.CommandButton cmdBrowse
      Caption         =   "Browse"
      Height          =   495
      Left            =   4320
      TabIndex        =   1
      Top             =   120
      Width           =   1095
   End
   Begin MSComDlg.CommonDialog diagSend
      Left            =   4200
      Top             =   3120
      _ExtentX        =   847
      _ExtentY        =   847
      _Version        =   393216
   End
   Begin VB.TextBox txtSend
      Height          =   375
      Left            =   960
      TabIndex        =   0
      Top             =   360
      Width           =   3015
   End
   Begin MSCommLib.MSComm MSComm1
      Left            =   5160
      Top             =   3000
      _ExtentX        =   1005
      _ExtentY        =   1005
      _Version        =   393216
      DTREnable       =   -1  'True
      Handshaking     =   2
      NullDiscard     =   -1  'True
      RThreshold      =   1
      RTSEnable       =   -1  'True
   End
   Begin VB.Label Label3
      Caption         =   "File receive:"
      Height          =   375
      Left            =   0
      TabIndex        =   13
      Top             =   1320
      Width           =   855
   End
   Begin VB.Label lblReceive
      Caption          =   "Receive file !!! Select
file name."
```

```
      ForeColor       =   &H000000FF&
      Height          =   375
      Left            =   840
      TabIndex        =   11
      Top             =   1920
      Visible         =   0   'False
      Width           =   2895
   End
   Begin VB.Label lblStatus
      Caption         =   "Disconnected"
      Height          =   375
      Left            =   120
      TabIndex        =   9
      Top             =   3720
      Width           =   5775
   End
   Begin VB.Label Label2
      Caption         =   "Dial:"
      Height          =   375
      Left            =   120
      TabIndex        =   8
      Top             =   2400
      Width           =   735
   End
   Begin VB.Label Label1
      Caption         =   "File send:"
      Height          =   375
      Left            =   120
      TabIndex        =   6
      Top             =   360
      Width           =   735
   End
   Begin VB.Label lblSize
      ForeColor       =   &H00FF0000&
      Height          =   375
      Left            =   960
      TabIndex        =   2
      Top             =   840
      Width           =   1815
   End
End
Attribute VB_Name = "frmModem"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Connected As Boolean
```

```
    Private SendFlag As Boolean
    Private ReceiveFlag As Boolean
    Private FileReceive As Integer
    Private CRFlag As Boolean
    Private Sub cmdBrowse_Click()
    On Error GoTo Loi
    diagSend.FileName = ""
    diagSend.Filter = "All files(*.*)|*.*"
    diagSend.InitDir = App.Path
    diagSend.ShowOpen
    txtSend.Text = diagSend.FileName
    lblSize.Caption = Str(Round(FileLen(txtSend.Text) /
1024, 2)) + "KB"
    Exit Sub
    Loi:
    lblSize.Caption = "0 KB"
    txtSend.Text = ""
    End Sub

    Private Sub cmdDial_Click()
    If Not MSComm1.PortOpen Then
        MsgBox "Comm Port Closed. Open first!!!",
vbOKOnly + vbCritical, "Error"
    ElseIf Trim(txtDial.Text) = "" Then
        MsgBox "Enter phone's number!!!", vbOKOnly +
vbCritical, "Error"
    Else
        If cmdDial.Caption = "Dial" Then
            MSComm1.Output = "ATDT" & Trim(txtDial.Text)
+ vbCr
            cmdDial.Caption = "Hang up"
            lblStatus.Caption = "Dialing ..."
        Else
            MSComm1.Output = "ATH1" + vbCr
            cmdDial.Caption = "Dial"
            lblStatus.Caption = "Hang up"
        End If
    End If

    End Sub

    Private Sub cmdExit_Click()
    If MSComm1.PortOpen Then
        MSComm1.PortOpen = False
    End If
    End
    End Sub
```

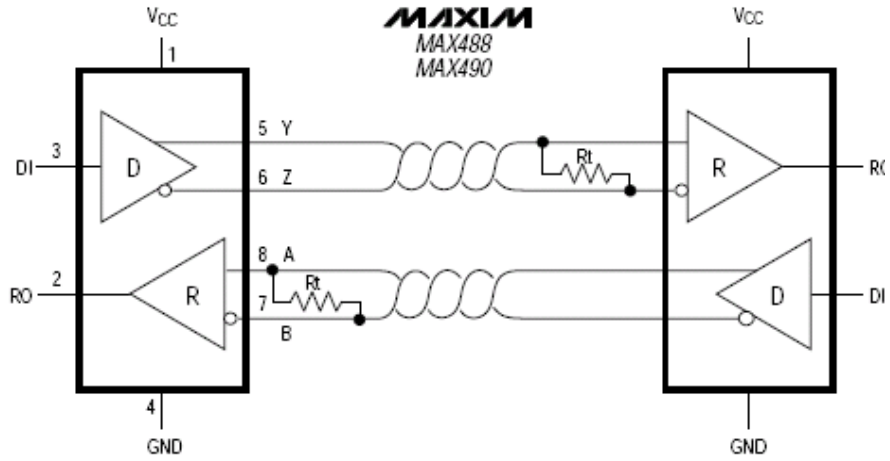
```
Private Sub cmdOpen_Click()
MSComm1.PortOpen = Not MSComm1.PortOpen
If MSComm1.PortOpen Then
    cmdOpen.Caption = "Close Port"
    MSComm1.Output = "ATS0=5" + vbCr
    Call Form_Load
Else
    cmdOpen.Caption = "Open Port"
    lblStatus.Caption = "Disconnected"
End If
End Sub

Private Sub cmdSave_Click()
            FileReceive = FreeFile
            ReceiveFlag = True
            Timer1.Enabled = False
            Do
                diagSend.FileName = ""
                diagSend.ShowSave
                If Trim(diagSend.FileName) = "" Then
                    MsgBox "File name error!!",
vbCritical + vbOKOnly, "Error"
                End If
            Loop While Trim(diagSend.FileName) = ""
            txtReceive.Text = diagSend.FileName
            MSComm1.Output = "RECEIVE" + vbCr
            Open Trim(txtReceive.Text) For Output As
#FileReceive
End Sub

Private Sub cmdSend_Click()
Dim FileNum As Integer
Dim Buffer As String
If Not MSComm1.PortOpen Then
    MsgBox "Comm Port Closed. Open first!!!",
vbOKOnly + vbCritical, "Error"
ElseIf Not Connected Then
    MsgBox "Not connected!!!", vbOKOnly +
vbCritical, "Error"
ElseIf Trim(txtSend.Text) = "" Then
    MsgBox "Select a file to send!!!", vbOKOnly +
vbCritical, "Error"
Else
    MSComm1.Output = "SEND" + vbCr
    Do
        DoEvents
```

```
            Loop While Not SendFlag
            FileNum = FreeFile
            Open Trim(txtSend.Text) For Input As #FileNum
            Do
                Input #FileNum, Buffer
                If Right(Buffer, 1) <> vbCr Then Buffer =
    Buffer + vbCrLf
                MSComm1.Output = Buffer
            Loop While Not EOF(FileNum)
            MSComm1.Output = "END FILE"
            Close #FileNum
            SendFlag = False
        End If
        End Sub

        Private Sub Form_Load()
        Connected = False
        SendFlag = False
        ReceiveFlag = False
        CRFlag = False
        End Sub

        Private Sub MSComm1_OnComm()
        Dim Buffer As String
        Dim Buffer1 As String
        Dim Buff As String
        Dim i As Integer
        Select Case MSComm1.CommEvent
            Case comEvRing
                lblStatus.Caption = "Ringing..."
            Case comEvCD
                If MSComm1.CDHolding Then
                    lblStatus.Caption = "Connected"
                    Connected = True
                Else
                    lblStatus.Caption = "Disconnected"
                    Connected = False
                End If
            Case comEvReceive
                Buffer = MSComm1.Input
                If InStr(Buffer, "SEND") Then
                    Timer1.Enabled = True
                    Exit Sub
                End If
                If InStr(Buffer, "RECEIVE") Then
                    SendFlag = True
                    Timer1.Enabled = False
```

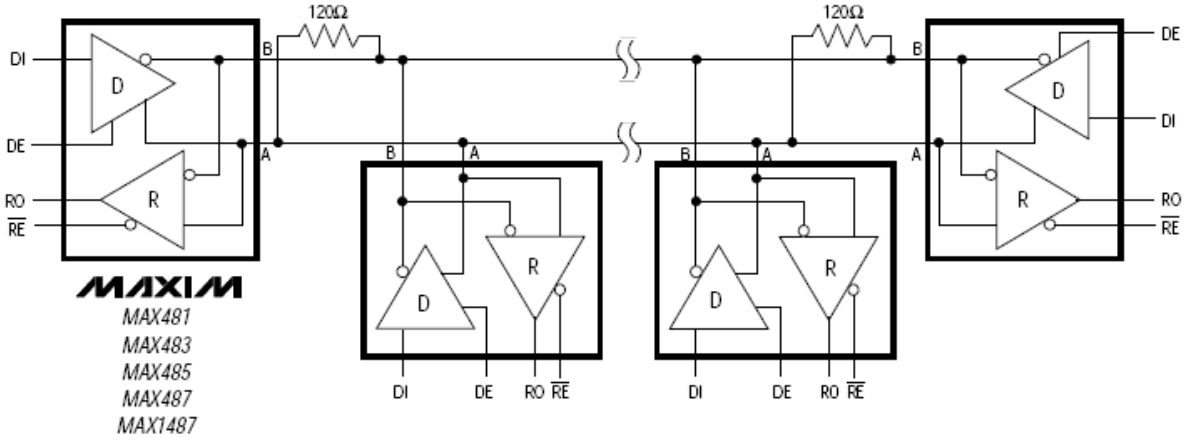
```
                    Buffer = ""
                    Exit Sub
                End If
                If InStr(Buffer, "CONNECT") Then
                    Connected = True
                    lblStatus.Caption = "Connected"
                    Exit Sub
                End If
                If ReceiveFlag Then
                    Buffer1 = ""
                    For i = 1 To Len(Buffer)
                        Buff = Mid$(Buffer, i, 1)
                        If Buff = Chr$(13) Then
                            CRFlag = True
                            Buff = ""
                        ElseIf Buff = Chr$(10) Then
                            CRFlag = False
                            Buff = ""
                        If Not CRFlag Then
                            Buffer1 = Buffer1 + Buff
                        End If
                    Next i
                    Print #FileReceive, Buffer1
                End If
                If InStr(Buffer, "END FILE") Then
                    Close #FileReceive
                    Call Form_Load
                End If
            Case comEvEOF
                lblStatus = "Disconnected"
                Connected = False
        End Select
        End Sub

        Private Sub Timer1_Timer()
        lblReceive.Visible = Not lblReceive.Visible
        End Sub

        Private Sub txtSend_LostFocus()
        On Error GoTo Loi
        lblSize.Caption = Str(Round(FileLen(txtSend.Text) /
1024, 2)) + "KB"
        Exit Sub
        Loi:
        lblSize.Caption = "0 KB"
        txtSend.Text = ""
        End Sub
```

7. Mạng 485

Chuẩn RS232 dùng đường truyền không cân bằng vì các tín hiệu lấy chuẩn là GND chung nên dễ bị ảnh hưởng của nhiễu làm tốc độ và khoảng cách truyền bị giới hạn. Khi muốn tăng khoảng cách truyền, môt phương pháp có thể sử dụng là dùng 2 dây truyền vi sai vì lúc này 2 dây có cùng đặc tính nên sẽ loại trừ được nhiễu chung. Hai chuẩn được sử dụng là RS422 và RS485 nhưng thông thường sử dụng RS485. Điện áp vi sai yêu cầu phải lớn hơn 200mV. Nếu $V_{AB} > 200$ mV thì tương ứng với logic 1 và $V_{AB} < -200$ mV tương ứng với logic 0. Chuẩn RS485 sử dụng hai điện trở kết thúc là 120 Ω tại hai đầu xa nhất của đường truyền và sử dụng dây xoắn đôi.



Hình 4.13 – Chuẩn giao tiếp RS422



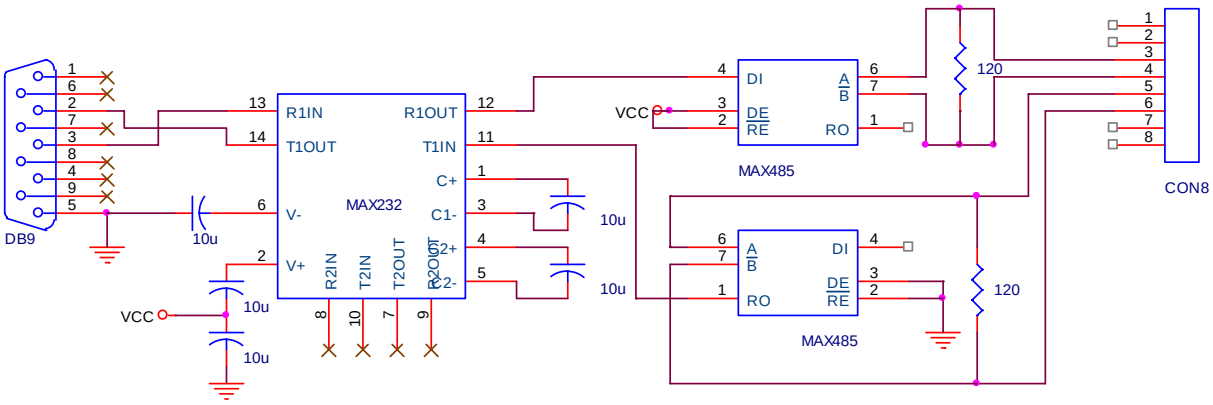
Hình 4.14 – Chuẩn giao tiếp RS485

Các đặc tính kỹ thuật:

Đặc tính	RS422	RS485
Số thiết bị truyền	1	32
Số thiết bị nhận	10	32

Chiều dài cable cực đại	1200m	1200m
Tốc độ truyền cực đại (từ 12 – 1200m)	10Mps – 100Kbps	10Mps – 100Kbps
Điện áp cực đại tại ngõ ra thiết bị truyền	-0.25V ÷ 6V	-7V ÷ 12V
Điện áp ngõ vào thiết bị nhận	-10V ÷ 10V	-7V ÷ 12V

Đối với chuẩn RS232, khoảng cách truyền không cho phép đi xa nên khi muốn thực hiện truyền ở khoảng cách xa thì phải chuyển từ RS232 sang chuẩn RS485 để truyền đi và sau đó chuyển từ RS485 sang RS232 để máy tính có thể nhận dạng được. Sơ đồ mạch chuyển đổi từ RS232 sang RS485 và ngược lại mô tả như sau:



Hình 4.15 – Chuyển đổi từ RS323 sang RS485 và ngược lại

Chương 5
GIAO TIẾP CỔNG SONG SONG

1. Cấu trúc cổng song song

Cổng song song gồm có 4 đường điều khiển, 5 đường trạng thái và 8 đường dữ liệu bao gồm 5 chế độ hoạt động:

- Chế độ tương thích (compatibility).
- Chế độ nibble.
- Chế độ byte.
- Chế độ EPP (Enhanced Parallel Port).
- Chế độ ECP (Extended Capabilities Port).

3 chế độ đầu tiên sử dụng port song song chuẩn (SPP – Standard Parallel Port) trong khi đó chế độ 4, 5 cần thêm phần cứng để cho phép hoạt động ở tốc độ cao hơn. Sơ đồ chân của máy in như sau:

Chân	Tín hiệu	Mô tả
1	$\overline{\text{STR}}$ (Out)	Mức tín hiệu thấp, truyền dữ liệu tới máy in
2	D0	Bit dữ liệu 0
3	D1	Bit dữ liệu 1
4	D2	Bit dữ liệu 2
5	D3	Bit dữ liệu 3
6	D4	Bit dữ liệu 4
7	D5	Bit dữ liệu 5
8	D6	Bit dữ liệu 6
9	D7	Bit dữ liệu 7
10	$\overline{\text{ACK}}$ (In)	Mức thấp: máy in đã nhận 1 ký tự và có khả năng nhận nữa
11	BUSY (In)	Mức cao: ký tự đã được nhận; bộ đệm máy in đầy; khởi động máy in; máy in ở trạng thái off-line.
12	PAPER EMPTY (In)	Mức cao: hết giấy
13	SELECT (In)	Mức cao: máy in ở trạng thái online
14	$\overline{\text{AUTOFEED}}$ (Out)	Tự động xuống dòng; mức thấp: máy in xuống dòng tự động
15	$\overline{\text{ERROR}}$ (In)	Mức thấp: hết giấy; máy in ở offline; lỗi máy in
16	$\overline{\text{INIT}}$ (Out)	Mức thấp: khởi động máy in
17	$\overline{\text{SELECTIN}}$ (Out)	Mức thấp: chọn máy in
18-25	GROUND	0V

Cổng song song có ba thanh ghi có thể truyền dữ liệu và điều khiển máy in. Địa chỉ cơ sở của các thanh ghi cho tất cả cổng LPT (line printer) từ LPT1 đến LPT4 được lưu trữ trong vùng dữ liệu của BIOS. Thanh ghi dữ liệu được định vị ở offset 00h, thanh ghi trang

thái ở 01h, và thanh ghi điều khiển ở 02h. Thông thường, địa chỉ cơ sở của LPT1 là 378h, LPT2 là 278h, do đó địa chỉ của thanh ghi trạng thái là 379h hoặc 279h và địa chỉ thanh ghi điều khiển là 37Ah hoặc 27Ah. Tuy nhiên trong một số trường hợp, địa chỉ của cổng song song có thể khác do quá trình khởi động của BIOS. BIOS sẽ lưu trữ các địa chỉ này như sau:

Địa chỉ	Chức năng
0000h:0408h	Địa chỉ cơ sở của LPT1
0000h:040Ah	Địa chỉ cơ sở của LPT2
0000h:040Ch	Địa chỉ cơ sở của LPT3

Định dạng các thanh ghi như sau:

Thanh ghi dữ liệu (hai chiều):

	7	6	5	4	3	2	1	0
Tín hiệu máy in	D7	D6	D5	D4	D3	D2	D1	D0
Chân số	9	8	7	6	5	4	3	2

Thanh ghi trạng thái máy in (chỉ đọc):

	7	6	5	4	3	2	1	0
Tín hiệu máy in	BUSY	$\overline{\text{ACK}}$	PAPER EMPTY	SELECT	$\overline{\text{ERROR}}$	$\overline{\text{IRQ}}$	x	x
Số chân cắm	11	10	12	13	15	-	-	-

Thanh ghi điều khiển máy in:

	7	6	5	4	3	2	1	0
Tín hiệu máy in	x	x	DIR	IRQ Enable	$\overline{\text{SELECTIN}}$	$\overline{\text{INIT}}$	$\overline{\text{AUTOFEED}}$	$\overline{\text{STROBE}}$
Số chân cắm	-	-	-	-	17	16	14	1

x: không sử dụng

IRQ Enable: yêu cầu ngắt cứng; 1 = cho phép; 0 = không cho phép

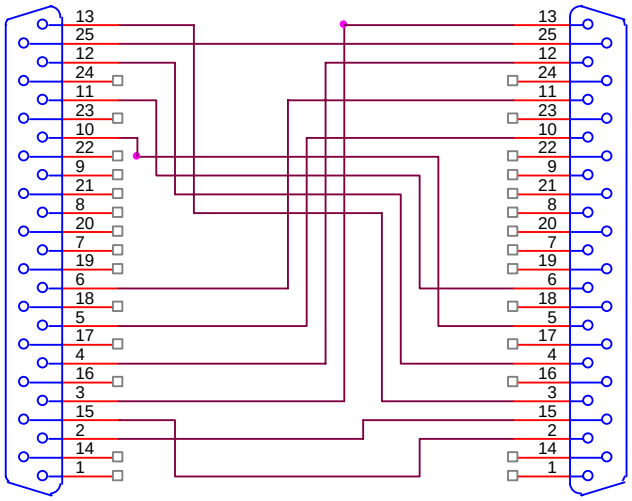
Chú ý rằng chân BUSY được nối với cổng đảo trước khi đưa vào thanh ghi trạng thái, các bit $\overline{\text{SELECTIN}}$, $\overline{\text{AUTOFEED}}$ và $\overline{\text{STROBE}}$ được đưa qua cổng đảo trước khi đưa ra các chân của cổng máy in.

Thông thường tốc độ xử lý dữ liệu của các thiết bị ngoại vi như máy in chậm hơn PC nhiều nên các đường $\overline{\text{ACK}}$, BUSY và $\overline{\text{STR}}$ được sử dụng cho kỹ thuật bắt tay. Khởi đầu, PC đặt dữ liệu lên bus sau đó kích hoạt đường $\overline{\text{STR}}$ xuống mức thấp để thông tin cho máy in biết rằng dữ liệu đã ổn định trên bus. Khi máy in xử lý xong dữ liệu, nó sẽ trả lại tín hiệu $\overline{ACK}$ xuống mức thấp để ghi nhận. PC đợi cho đến khi đường BUSY từ máy in xuống thấp (máy in không bận) thì sẽ đưa tiếp dữ liệu lên bus.

2. Giao tiếp với thiết bị ngoại vi

2.1. Giao tiếp với máy tính

Quá trình giao tiếp với cổng song song dùng 2 chế độ: chế độ chuẩn SPP và chế độ mở rộng. Việc giao tiếp ở chế độ chuẩn mô tả như sau:

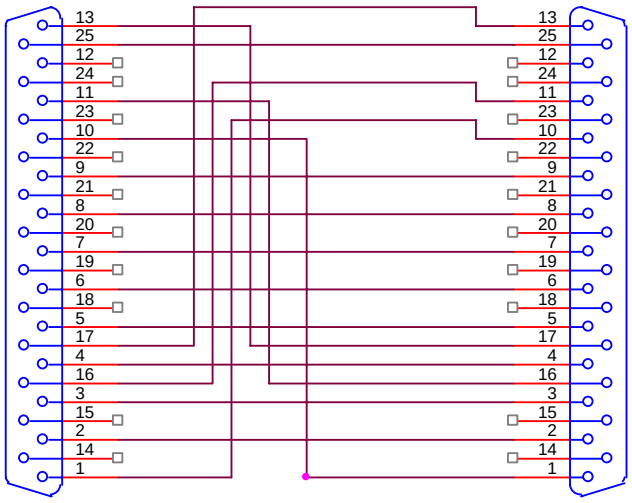


Hình 5.1 - Trao đổi dữ liệu qua cổng song song giữa 2 PC dùng chế độ chuẩn

Sơ đồ chân kết nối mô tả như sau:

PC1		PC2	
Chức năng	**Chân**	**Chân**	**Chức năng**
D0	2	15	$\overline{\text{ERROR}}$
D1	3	13	SELECT
D2	4	12	PAPER EMPTY
D3	5	10	$\overline{\text{ACK}}$
D4	6	11	BUSY
BUSY	11	6	D4
$\overline{\text{ACK}}$	10	5	D3
PAPER EMPTY	12	4	D2
SELECT	13	3	D1
$\overline{\text{ERROR}}$	15	2	D0
GND	25	25	GND

Ngoài ra, việc kết nối giữa 2 máy tính sử dụng cổng song song có thể dùng chế độ mở rộng, chế độ này cho phép giao tiếp với tốc độ cao hơn.



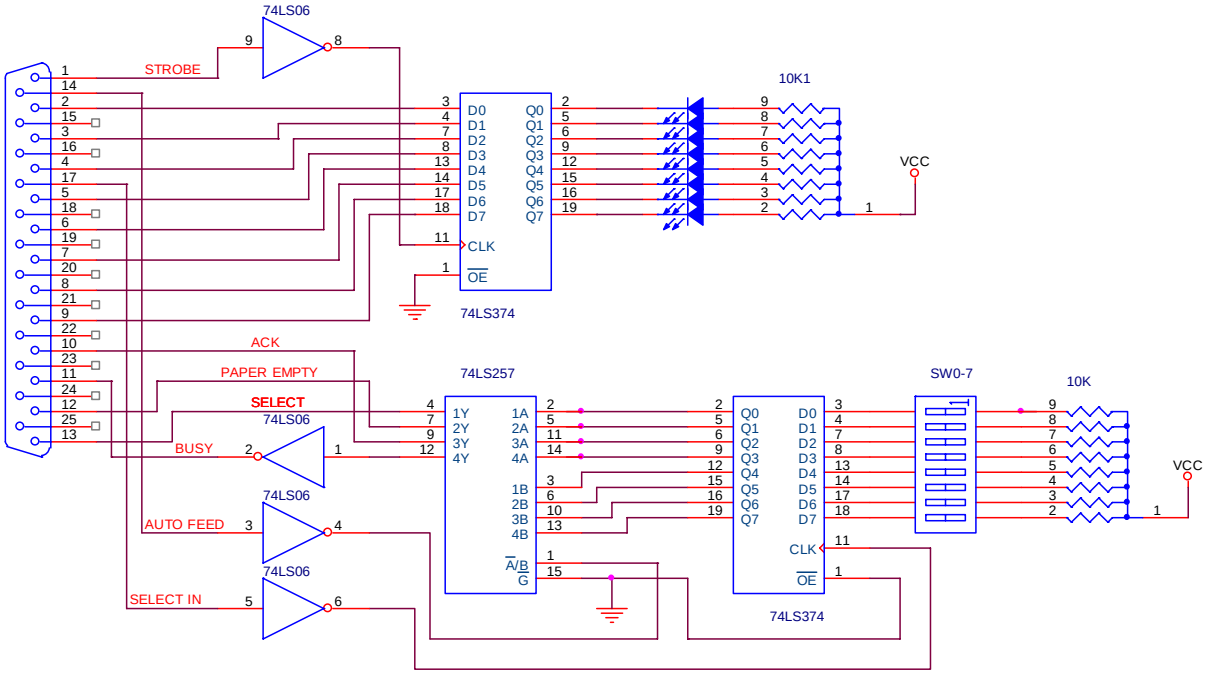
Hình 5.2 - Trao đổi dữ liệu qua cổng song song giữa 2 PC dùng chế độ mở rộng

Sơ đồ chân kết nối mô tả như sau:

PC1		**PC2**	
Chức năng	**Chân**	**Chân**	**Chức năng**
D0	2	2	D0
D1	3	3	D1
D2	4	4	D2
D3	5	5	D3
D4	6	6	D4
D5	7	7	D5
D6	8	8	D6
D7	9	9	D7
SELECT	13	17	$\overline{\text{SELECTIN}}$
BUSY	11	16	$\overline{\text{INIT}}$
$\overline{\text{ACK}}$	10	1	$\overline{\text{STROBE}}$
$\overline{\text{SELECTIN}}$	17	13	SELECT
$\overline{\text{INIT}}$	16	11	BUSY
$\overline{\text{STROBE}}$	1	10	$\overline{\text{ACK}}$

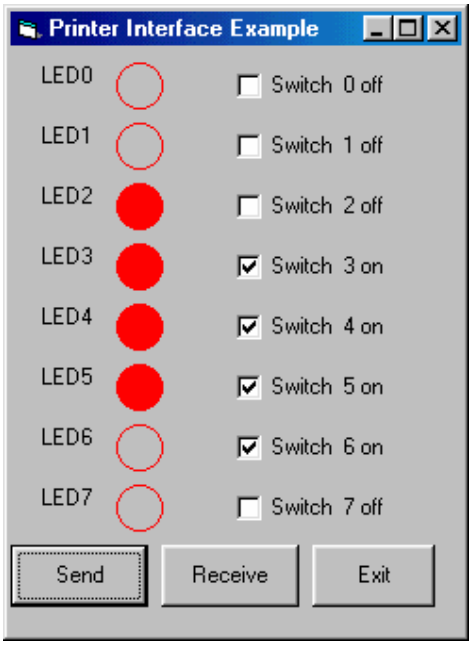
2.2. Giao tiếp thiết bị khác

Quá trình giao tiếp với các thiết bị ngoại vi có thể thực hiện thông qua chế độ chuẩn. Để đọc dữ liệu, có thể dùng một IC ghép kênh 2→1 74LS257 và dùng 4 bit trạng thái của cổng song song còn xuất dữ liệu thì sử dụng 8 đường dữ liệu D0 – D7.



Hình 5.3 – Mạch giao tiếp đơn giản thông qua cổng máy in

Giao diện:



Hình 5.4 – Giao diện của chương trình giao tiếp với cổng máy in

Chương trình giao tiếp trên VB sử dụng thư viện liên kết động để trao đổi dữ liệu với cổng máy in. Thư viện IO.DLL bao gồm các hàm sau:

- Hàm PortOut: xuất 1 byte ra cổng

```
    Private Declare Sub PortOut Lib "IO.DLL" (ByVal Port
As Integer, ByVal Data As Byte)
```

Port: địa chỉ cổng, Data: dữ liệu xuất

- Hàm PortWordOut: xuất 1 word ra cổng

```
    Private Declare Sub PortWordOut Lib "IO.DLL" (ByVal
Port As Integer, ByVal Data As Integer)
```

- Hàm PortDWordOut: xuất 1 double word ra cổng

```
    Private Declare Sub PortDWordOut Lib "IO.DLL" (ByVal
Port As Integer, ByVal Data As Long)
```

- Hàm PortIn: nhập 1 byte từ cổng, trả về giá trị nhập

```
    Private Declare Function PortIn Lib "IO.DLL" (ByVal
Port As Integer) As Byte
```

- Hàm PortWordIn: nhập 1 word từ cổng

```
    Private Declare Function PortWordIn Lib "IO.DLL"
(ByVal Port As Integer) As Integer
```

- Hàm PortDWordIn: nhập 1 double word từ cổng

```
    Private Declare Function PortDWordIn Lib "IO.DLL"
(ByVal Port As Integer) As Long
```

Chương trình nguồn:

```
    VERSION 5.00
    Begin VB.Form Form1
       Caption         =   "Printer Interface Example"
       ClientHeight    =   4665
       ClientLeft      =   60
       ClientTop       =   345
       ClientWidth     =   3585
       LinkTopic       =   "Form1"
       ScaleHeight     =   4665
       ScaleWidth      =   3585
       StartUpPosition =   3  'Windows Default
       Begin VB.CommandButton cmdReceive
          Caption         =   "Receive"
          Height          =   495
          Left            =   1200
          TabIndex        =   18
          Top             =   3960
          Width           =   1095
```

```
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   7
   Left            =   1800
   TabIndex        =   17
   Top             =   3480
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   6
   Left            =   1800
   TabIndex        =   16
   Top             =   3000
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   5
   Left            =   1800
   TabIndex        =   15
   Top             =   2520
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   4
   Left            =   1800
   TabIndex        =   14
   Top             =   2040
   Width           =   1575
End
Begin VB.CheckBox chkSW
   Height          =   375
   Index           =   3
   Left            =   1800
```

```
      TabIndex        =   13
      Top             =   1560
      Width           =   1575
   End
   Begin VB.CheckBox chkSW
      Height          =   375
      Index           =   2
      Left            =   1800
      TabIndex        =   12
      Top             =   1080
      Width           =   1575
   End
   Begin VB.CheckBox chkSW
      Height          =   375
      Index           =   1
      Left            =   1800
      TabIndex        =   11
      Top             =   600
      Width           =   1575
   End
   Begin VB.CheckBox chkSW
      Height          =   375
      Index           =   0
      Left            =   1800
      TabIndex        =   10
      Top             =   120
      Width           =   1575
   End
   Begin VB.CommandButton cmdExit
      Caption         =   "Exit"
      Height          =   495
      Left            =   2400
      TabIndex        =   9
      Top             =   3960
      Width           =   975
   End
   Begin VB.CommandButton cmdSend
```

```
      Caption         =   "Send"
      Height          =   495
      Left            =   0
      TabIndex        =   8
      Top             =   3960
      Width           =   1095
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED7"
      Height          =   375
      Index           =   7
      Left            =   240
      TabIndex        =   7
      Top             =   3480
      Width           =   1095
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED6"
      Height          =   375
      Index           =   6
      Left            =   240
      TabIndex        =   6
      Top             =   3000
      Width           =   975
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED5"
      Height          =   375
      Index           =   5
      Left            =   240
      TabIndex        =   5
      Top             =   2520
      Width           =   975
   End
```

```
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED4"
   Height          =   375
   Index           =   4
   Left            =   240
   TabIndex        =   4
   Top             =   2040
   Width           =   975
End
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED3"
   Height          =   375
   Index           =   3
   Left            =   240
   TabIndex        =   3
   Top             =   1560
   Width           =   975
End
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED2"
   Height          =   375
   Index           =   2
   Left            =   240
   TabIndex        =   2
   Top             =   1080
   Width           =   975
End
Begin VB.Label lblLED
   BackStyle       =   0  'Transparent
   Caption         =   "LED1"
   Height          =   375
   Index           =   1
   Left            =   240
   TabIndex        =   1
```

```
      Top             =   600
      Width           =   975
   End
   Begin VB.Label lblLED
      BackStyle       =   0  'Transparent
      Caption         =   "LED0"
      Height          =   375
      Index           =   0
      Left            =   240
      TabIndex        =   0
      Top             =   120
      Width           =   975
   End
   Begin VB.Shape shpLED
      BorderColor     =   &H000000FF&
      FillColor       =   &H000000FF&
      FillStyle       =   0  'Solid
      Height          =   375
      Index           =   7
      Left            =   840
      Shape           =   3  'Circle
      Top             =   3480
      Width           =   375
   End
   Begin VB.Shape shpLED
      BorderColor     =   &H000000FF&
      FillColor       =   &H000000FF&
      FillStyle       =   0  'Solid
      Height          =   375
      Index           =   6
      Left            =   840
      Shape           =   3  'Circle
      Top             =   3000
      Width           =   375
   End
   Begin VB.Shape shpLED
      BorderColor     =   &H000000FF&
```

```
      FillColor       =   &H000000FF&
      FillStyle       =   0  'Solid
      Height          =   375
      Index           =   5
      Left            =   840
      Shape           =   3  'Circle
      Top             =   2520
      Width           =   375
   End
   Begin VB.Shape shpLED
      BorderColor     =   &H000000FF&
      FillColor       =   &H000000FF&
      FillStyle       =   0  'Solid
      Height          =   375
      Index           =   4
      Left            =   840
      Shape           =   3  'Circle
      Top             =   2040
      Width           =   375
   End
   Begin VB.Shape shpLED
      BorderColor     =   &H000000FF&
      FillColor       =   &H000000FF&
      FillStyle       =   0  'Solid
      Height          =   375
      Index           =   3
      Left            =   840
      Shape           =   3  'Circle
      Top             =   1560
      Width           =   375
   End
   Begin VB.Shape shpLED
      BorderColor     =   &H000000FF&
      FillColor       =   &H000000FF&
      FillStyle       =   0  'Solid
      Height          =   375
      Index           =   2
```

```
            Left            =   840
            Shape           =   3  'Circle
            Top             =   1080
            Width           =   375
         End
         Begin VB.Shape shpLED
            BorderColor     =   &H000000FF&
            FillColor       =   &H000000FF&
            FillStyle       =   0  'Solid
            Height          =   375
            Index           =   1
            Left            =   840
            Shape           =   3  'Circle
            Top             =   600
            Width           =   375
         End
         Begin VB.Shape shpLED
            BorderColor     =   &H000000FF&
            FillColor       =   &H000000FF&
            FillStyle       =   0  'Solid
            Height          =   375
            Index           =   0
            Left            =   840
            Shape           =   3  'Circle
            Top             =   120
            Width           =   375
         End
      End
      Attribute VB_Name = "Form1"
      Attribute VB_GlobalNameSpace = False
      Attribute VB_Creatable = False
      Attribute VB_PredeclaredId = True
      Attribute VB_Exposed = False
      'IO.DLL
      Private Declare Sub PortOut Lib "IO.DLL" (ByVal Port
As Integer, ByVal Data As Byte)
```

```
    Private Declare Function PortIn Lib "IO.DLL" (ByVal
Port As Integer) As Byte
    'Variable
    Private BA_LPT As Integer

    Private Sub cmdExit_Click()
    End
    End Sub

    Private Sub cmdReceive_Click()
    Dim n As Integer
    Dim n1 As Integer
    Dim i As Integer

    PortOut BA_LPT + 2, &H8 'SELECTIN = 1
    PortOut BA_LPT + 2, 0 'SELECTIN = 0
    n1 = PortIn(BA_LPT + 1) 'Doc 4 bit thap
    n1 = n1 / &H10          'Dich phai 4 bit
    PortOut BA_LPT + 2, 2   'AUTOFEED=1
    n = PortIn(BA_LPT + 1)  'Doc 4 bit cao
    n = n And &HF0
    n = n + n1
        For i = 0 To 7
            chkSW(i).Value = n Mod 2
            If chkSW(i).Value = 0 Then
                chkSW(i).Caption = "Switch " & Str(i) &
" off"
            Else
                chkSW(i).Caption = "Switch " & Str(i) &
" on"
            End If
            n = Fix(n / 2)
        Next i
    End Sub

    Private Sub cmdSend_Click()
    Dim t As Integer
    Dim i As Integer
```

```
    Dim s As String
    t = 0
    For i = 0 To 7
       t = t + (2 ^ i) * (1 - shpLED(i).FillStyle)
    Next i
    PortOut BA_LPT, t
    PortOut BA_LPT, 1   'STROBE = 1
    PortOut BA_LPT, 0   'STROBE = 0
    End Sub

    Private Sub Form_Load()
    BA_LPT = &H378
    PortOut BA_LPT + 2, 0
    End Sub

    Private Sub lblLED_Click(Index As Integer)
    shpLED(Index).FillStyle = 1 -
shpLED(Index).FillStyle
    End Sub
```

Vi Xử Lý 8051

Created by mercury

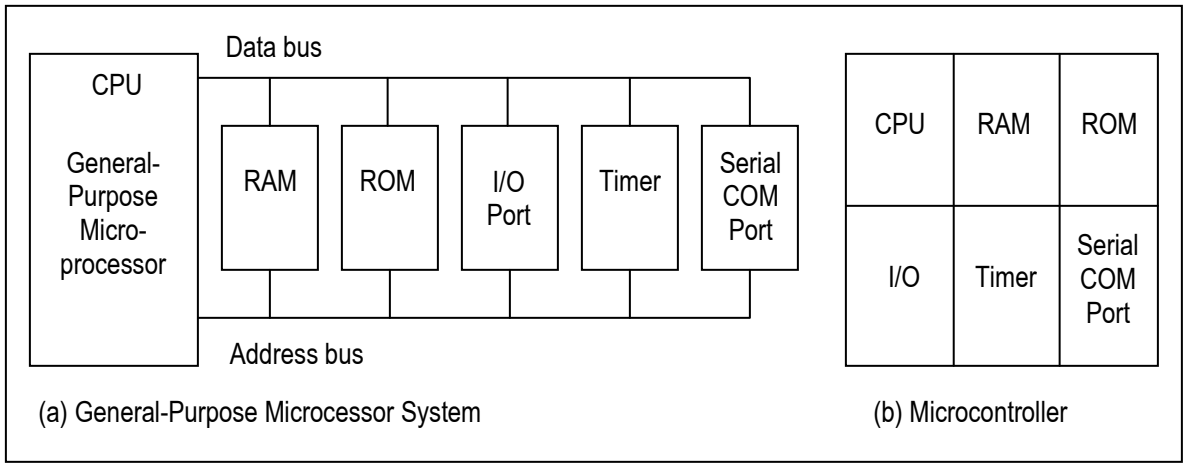
CHƯƠNG I
Các bộ vi điều khiển 8051

1.1 các bộ vi điều khiển và các bộ xử lý nhúng.

Trong mục này chúng ta bàn về nhu cầu đối với các bộ vi điều khiển (VĐK) và so sánh chúng với các bộ vi xử lý cùng dạng chung như Pentium và các bộ vi xử lý × 86 khác. Chúng ta cùng xem xét vai trò của các bộ vi điều khiển trong thị trường các sản phẩm nhúng. Ngoài ra, chúng ta cung cấp một số tiêu chuẩn về cách lựa chọn một bộ vi điều khiển như thế nào.

1.1.1 Bộ vi điều khiển so với bộ vi xử lý cùng dùng chung

Sự khác nhau giữa một bộ vi điều khiển và một bộ vi xử lý là gì? Bộ vi xử lý ở đây là các bộ vi xử lý công dung chung như họ Intell × 86 (8086, 80286, 80386, 80486 và Pentium) hoặc họ Motorola 680 × 0(68000, 68010, 68020, 68030, 68040 v.v...). Những bộ VXL này không có RAM, ROM và không có các cổng vào ra trên chíp. Với lý do đó mà chúng được gọi chung là các bộ vi xử lý công dụng chung.



Hình 1.1: Hệ thống vi xử lý được so sánh với hệ thống vi điều khiển.

a) Hệ thống vi xử lý công dụng chung

b) Hệ thống vi điều khiển

Một nhà thiết kế hệ thống sử dụng một bộ vi xử lý công dụng chung chẳng hạn như Pentium hay 68040 phải bổ xung thêm RAM , ROM, các cổng vào ra và các bộ định thời ngoài để làm cho chúng hoạt động được. Mặc dù việc bổ xung RAM, ROM và các cổng vào ra bên ngoài làm cho hệ thống cồng cềnh và đắt hơn, nhưng chúng có ưu điểm là linh hoạt chẳng hạn như người thiết kế có thể quyết định về số lượng RAM, ROM và các cổng vào ra cần thiết phù hợp với bài toán trong tầm tay của mình.

Điều này không thể có được đối với các bộ vi điều khiển. Một bộ vi điều khiển có một CPU (một bộ vi xử lý) cùng với một lượng cố định RAM, ROM, các cổng vào ra và một bộ định thời tất cả trên cùng một chíp. Hay nói cách khác là bộ xử lý, RAM, ROM các cổng vào ra và bộ định thời đều được nhúng với nhau trên một chíp; do vậy người thiết kế không thể bổ xung thêm bộ nhớ ngoài, cổng vào ra hoặc bộ định thời cho nó. Số lượng cố định của RAM, ROM trên chíp và số các cổng vào - ra trong các bộ vi điều khiển làm cho chúng trở nên lý tưởng đối với nhiều ứng dụng mà trong đó giá thành và không gian lại hạn chế. Trong nhiều ứng dụng, ví dụ một điều khiển TV từ xa thì không cần công suất tính toán của bộ vi sử lý 486 hoặc thậm chí như 8086. Trong rất nhiều ứng dụng thì không gian nó chiếm, công suất nó tiêu tốn và giá thành trên một đơn vị là những cân nhắc nghiêm ngặt hơn nhiều so với công suất tính toán. Những ứng dụng thường yêu cầu một số thao tác vào - ra để đọc các tín hiệu và tắt - mở những bit nhất định. Vì lý do này mà một số người gọi các bộ xử lý này là IBP ("Itty-Bitty-Processor"), (tham khảo cuốn "Good things in small packages are Generating Big product opportunities" do Rick Grehan viết trên tạp BYTE tháng 9.1994; WWW. Byte. Com để biết về những trao đổi tuyệt vời về các bộ vi điều khiển).

Điều thú vị là một số nhà sản xuất các bộ vi điều khiển đã đi xa hơn là tích hợp cả một bộ chuyển đổi ADC và các ngoại vi khác vào trong bộ vi điều khiển.

Bảng 1.1: Một số sản phẩm được nhúng sử dụng các bộ vi điều khiển

Thiết bị nội thất gia đình	**Văn phòng**	**ô tô**
Đồ điện trong nhà	Điện thoại	Máy tính hành trình

Máy đàm thoại Máy điện thoại Các hệ thống an toàn Các bộ mở cửa ga-ra xe Máy trả lời Máy Fax Máy tính gia đình Tivi Truyền hình cáp VCR Máy quy camera Điều khiển từ xa Trò chơi điện tử Điện thoại tổ ong Các nhạc cụ điện tử Máy khâu Điều khiển ánh sáng Máy nhắn tin Máy chơi Pootball Đồ chơi Các dụng cụ tập thể hình	Máy tính Các hệ thống an toàn Máy Fax Lò vi sóng Máy sao chụp Máy in lazer Máy in màu Máy nhắn tin	Điều khiển động cơ Túi đệm khí Thiết bị ABS Đo lường Hệ thống bảo mật Đíũu khiển truyền tin Giải trí Điều hoà nhiệt độ Điện thoại tổ ong Mở cửa không cần chìa khoá

1.1.2 Các bộ VĐK cho các hệ thống nhúng.

Trong tài liệu về các bộ vi xử lý ta thường thấy khái niệm hệ thống nhúng (Embeded system). Các bộ vi xử lý và các bộ vi điều khiển được sử dụng rộng rãi trong các sản phẩm hệ thống nhúng. Một sản phẩm nhúng sử dụng một bộ vi xử lý (hoặc một bộ vi điều khiển để thực hiện một nhiệm vụ và chỉ một mà thôi. Một máy in là một ví dụ về một việc nhúng vì bộ xử lý bên trong nó chỉ làm một việc đó là nhận dữ liệu và in nó ra. Điều này khác với một máy tình PC dựa trên bộ xử lý Pentium (hoặc một PC tương thích với IBM × 86 bất kỳ). Một PC có thể được sử dụng cho một số bất kỳ các trạm dịch vụ in, bộ đầu cuối kiểm kê nhà băng, máy chơi trò chơi điện tử, trạm dịch vụ mạng hoặc trạm đầu cuối mạng Internet. Phần mềm cho các ứng dụng khác nhau có thể được nạp và chạy. Tất nhiên là lý do hiển nhiên để một PC thực hiện hàng loạt các công việc là nó có bộ

nhớ RAM và một hệ điều hành nạp phần mềm ứng dụng thường được đốt vào trong ROM. Một máy tính PC × 86 chứa hoặc được nối tới các sản phẩm nhúng khác nhau chẳng hạn như bàn phím, máyin, Modem, bộ điều khiển đĩa, Card âm thanh, bộ điều khiển CD = ROM. Chuột v.v... Một nội ngoại vi này có một bộ vi điều khiển bên trong nó để thực hiện chỉ một công việc, ví dụ bên trong mỗi con chuột có một bộ vi điều khiển để thực thi công việc tìm vị trí chuột và gửi nó đến PC Bảng 1.1 liệt kê một số sản phẩm nhúng.

4.1.3 Các ứng dụng nhúng của PC × 86.

Mặc dù các bộ vi điều khiển là sự lựa chọn ưa chuộng đối với nhiều hệ thống nhúng nhưng có nhiều khi một bộ vi điều khiển không đủ cho công việc. Vì lý do đó mà những năm gần đây nhiều nhà sản xuất các bộ vi sử lý công dụng chung chẳng hạn như Intel, Motorla, AMD (Advanced Micro Devices, Inc...). Và Cyric (mà bây giờ là một bộ phận của National Senicon ductir, Inc) đã hướng tới bộ vi xử lý cho hiệu suất cao của thị trường nhúng. Trong khi Intel, AMD và Cyrix đẩy các bộ xử lý × 86 của họ vào cho cả thị trường nhúng và thị trường máy tính PC để bán thì Motorola vẫn kiên định giữ họ vi xử lý 68000 lại chủ yếu hướng nó cho các hệ thống nhúng hiệu suất cao và bây giờ Apple không còn dùng 680 × trong các máy tính Macintosh nữa. Trong những năm đầu thập kỷ 90 của thế kỷ 20 máy tính Apple bắt đầu sử dụng các bộ vi xử lý Power PC (như 603, 604, 620 v.v...) thay cho 680 ×0 đối với Macintosh. Bộvi xử lý Power PC là kết quả liên doanh đầu tư của IBM và Motorola và nó được hướng cho thị trướng nhúng hiệu suất cao cũng như cho cả thị trường máy tính PC. Cần phải lưu ý rằng khi một công ty hướng một bộ vi xử lý công dụng chung cho thị trường nhúng nó tối ưu hoá bộ xử lý được sử dụng cho các hệ thống nhúng. Vì lý do đó mà các bộ vi xử lý này thường được gọi là các bộ xử lý nhúng hiệu suất cao. Do vậy các khái niệm các bộ vi điều khiển và bộ xử lý nhúng thường được sử dụng thay đổi nhau.

Một trong những nhu cầu khắt khe nhất của hệ thống nhúng là giảm công suất tiêu thụ và không gian.

Điều này có thể đạt được bằng cách tích hợp nhiều chức năng vào trong chíp CPU. Tất cả mọi bộ xử lý nhúng dựa trên × 86 và 680 × 0 đều có công suất tiêu thu thấp ngoài ra được bổ xung một số dạng cổng vào - ra, cổng COM và bộ nhớ ROM trên một chíp.

Trong các bộ xử lý nhúng hiệu suất cao có xu hướng tích hợp nhiều và nhiều chức năng hơn nữa trên chíp CPU và cho phép người thiết kế quyết định những đặc tính nào họ muốn sử dụng. Xu hướng này cũng đang chiếm lĩnh thiết kế hệ thống PC. Bình thường khi thiết kế bo mạch chủ của PC (Motherboard) ta cần một CPU cộng một chíp - set có chứa các cổng vào - ra, một bộ điều khiển cache, một bộ nhớ Flash ROM có chứa BIOS và cuối cùng là bộ nhớ cache thứ cấp. Những thiết kế mới đang khẩn trương đi vào công nghiệp sản xuất hàng loạt. Ví dụ Cyrix đã tuyên bố rằng họ đang làm việc trên một chíp có chứa toàn bộ một máy tính PC ngoại trừ DRAM. Hay nói cách khác là chúng ta sắp nhìn thấy một máy tính PC trên một chíp.

Hiện nay do chuẩn hoá MS - DOS và Windows nên các hệ thống nhúng đang sử dụng các máy tình PC × 86 . Trong nhiều trường hợp việc sử dụng các máy tính PC × 86 cho các ứng dụng nhúng hiệu suất cao là không tiết kiệm tiền bạc, nhưng nó làm rút ngắn thời gian phát triển vì có một thư viện phần mềm bao la đã được viết cho nền DOS và Windows. Thực tế là Windows là một nền được sử dụng rộng rãi và dễ hiểu có nghĩa là việc phát triển một sản phẩm nhúng dựa trên Windows làm giảm giá thành và rút ngắn thời gian phát triển đáng kể.

1.1.4 Lựa chọn một bộ vi điều khiển.

Có 4 bộ vi điều khiển 8 bit chính. Đó là 6811 của Motorola, 8051 của Intel z8 của Xilog và Pic 16 × của Microchip Technology. Mỗi một kiểu loại trên đây đều có một tập lệnh và thanh ghi riêng duy nhất, nếu chúng đều không tương thích lẫn nhau. Cũng có những bộ vi điều khiển 16 bit và 32 bit được sản xuất bởi các hãng sản xuất chíp khác nhau. Với tất cả những bộ vi điều khiển khác nhau như thế này thì lấy gì làm tiêu chuẩn lựa chọn mà các nhà thiết kế phải cân nhắc? Có ba tiêu chuẩn để lựa chọn các bộ vi điều khiển là:

1) Đáp ứng nhu cầu tính toán của bài toán một cách hiệu quả về mặt giá thành và đầy đủ chức năng có thể nhìn thấy được (khả dĩ).

2) Có sẵn các công cụ phát triển phần mềm chẳng hạn như các trình biên dịch, trình hợp ngữ và gỡ rối.

3) Nguồn các bộ vi điều khiển có sẵn nhiều và tin cậy.

1.1.5 Các tiêu chuẩn lựa chọn một bộ vi điều khiển.

1. Tiêu chuẩn đầu tiênvà trước hết tronglựa chọn một bộ vi điều khiển là nó phải đáp ứng nhu cầu bài toán về một mặt công suất tính toán và giá thành hiệu quả. Trong khi phân tích các nhu cầu của một dự án dựa trên bộ vi điều khiển chúng ta trước hết phải biết là bộ vi điều khiển nào 8 bit, 16 bit hay 32 bit có thể đáp ứng tốt nhất nhu cầu tính toán của bài toán mộ tcách hiệu quả nhất? Những tiêu chuẩn được đưa ra để cân nhắc là:
a) Tốc độ: Tốc độ lớn nhất mà bộ vi điều khiển hỗ trợ là bao nhiêu.
b) Kiểu đóng vỏ: Đó là kiểu 40 chân DIP hay QFP hay là kiểu đóng vỏ khác (DIP -đóng vỏ theo 2 hàng chân. QFP là đóng vỏ vuông dẹt)? Đây là điều quan trọng đối với yêu cầu về không gian, kiểu lắp ráp và tạo mẫu thử cho sản phẩm cuối cùng.
c) Công suất tiêu thụ: Điều này đặc biệt khắt khe đối với những sản phẩm dùng pin, ắc quy.
d) Dung lượng bộ nhớ RAM và ROM trên chíp.
e) Số chân vào - ra và bộ định thời trên chíp
f) Khả năng dễ dàng nâng cấp cho hiệu suất cao hoặc giảm công suất tiêu thụ.
g) Giá thành cho một đơn vị: Điều này quan trọng quyết định giá thành cuối cùng của sản phẩm mà một bộ vi điều khiển được sử dụng. Ví dụ có các bộ vi điều khiển giá 50 cent trên đơn vị khi được mua 100.000 bộ một lúc.
2) Tiêu chuẩn thứ hai trong lựa chọn một bộ vi điều khiển là khả năng phát triển các sản phẩm xung quanh nó dễ dàng như thế nào? Các cân nhắc chủ yếu bao gồm khả năng có sẵn trình lượng ngữ, gỡ rối, trình biên dịch ngôn ngữ C hiệu quả về mã nguồn, trình mô phỏng hỗ trợ kỹ thuật và khả năng sử dụng trong nhà và ngoài môi trường. Trong nhiều trường hợp sự hỗ trợ nhà cung cấp thứ ba (nghĩa là nhà cung cấp khác không phải là hãng sản xuất chíp) cho chíp cũng tốt như, nếu không được tốt hơn, sự hỗ trợ từ nhà sản xuất chíp.
3) Tiêu chuẩn thứ ba trong lựa chọn một bộ vi điều khiển là khả năng sẵn sàng đáp ứng về số lượng trong hiện tại và tương lai. Đối với một số nhà thiết kế điều này thậm chí còn quan trong hơn cả hai tiêu chuẩn đầu tiên. Hiện nay, các bộ vi điều khiển 8 bit dấu đầu, họ 8051 là có số lương lớn nhất các nhà cung cấp đa dạng (nhiều nguồn). Nhà cung cấp có nghĩa là nhà sản xuất bên cạnh nhà sáng chế của bộ vi điều khiển. Trong trường hợp 8051 thì nhà sáng chế

của nó là Intel, nhưng hiện nay có rất nhiều hãng sản xuất nó (cũng như trước kia đã sản xuất).

Các hãng này bao gồm: Intel, Atmel, Philips/signe-tics, AMD, Siemens, Matra và Dallas, Semicndictior.

Bảng 1.2: Địa chỉ của một số hãng sản xuất các thành viên của họ 8051.

Hãng	**Địa chỉ Website**
Intel Antel Plips/ Signetis Siemens Dallas Semiconductor	www.intel.com/design/mcs51 www.atmel.com www.semiconductors.philips.com www.sci.siemens.com www.dalsemi.com

Cũng nên lưu ý rằng Motorola, Zilog và Mierochip Technology đã dành một lượng tài nguyên lớn để đảm bảo khả năng sẵn sàng về một thời gian và phạm vi rộng cho các sản phẩm của họ từ khi các sản phẩm của họ đi vào sản xuất ổn định, hoàn thiện và trở thành nguồn chính. Trong những năm gần đây họ cũng đã bắt đầu bán tế bào thư viện Asic của bộ vi điều khiển.

1.2 Tổng quan về họ 8051.

Tr ong mục này chúng ta xem xét một số thành viên khác nhau của họ bộ vi điều khiển 8051 và các đặc điểm bên trong của chúng. Đồng thời ta điểm qua một số nhà sản xuất khác nhau và các sản phẩm của họ có trên thị trường.

1.2.1 Tóm tắt về lịch sử của 8051.

Vào năm 1981. Hãng Intel giới thiệu một số bộ vi điều khiển được gọi là 8051. Bộ vi điều khiển này có 128 byte RAM, 4K byte ROM trên chíp, hai bộ định thời, một cổng nối tiếp và 4 cổng (đều rộng 8 bit) vào ra tất cả được đặt trên một chíp. Lúc ấy nó được coi là một "hệ thống trên chíp". 8051 là một bộ xử lý 8 bit có nghĩa là CPU chỉ có thể làm việc với 8 bit dữ liệu tại một thời điểm. Dữ liệu lớn hơn 8 bit được chia ra thành các dữ liệu 8 bit để cho xử lý. 8051 có tất cả 4 cổng vào - ra I/O mỗi cổng rộng 8 bit (xem hình 1.2). Mặc dù 8051 có thể có một ROM trên chíp cực đại là 64 K byte, nhưng các nhà sản xuất lúc đó đã cho xuất xưởng chỉ với 4K byte ROM trên chíp. Điều này sẽ được bàn chi tiết hơn sau này.

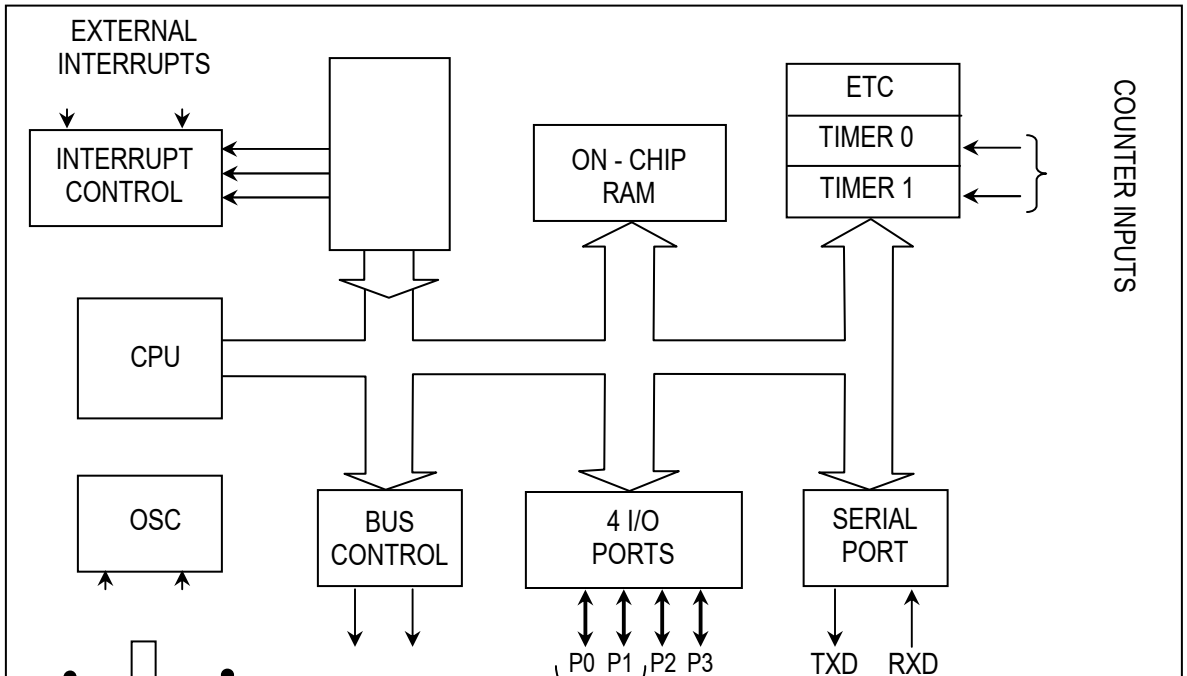
8051 đã trở nên phổ biến sau khi Intel cho phép các nhà sản xuất khác sản xuất và bán bất kỳ dạng biến thế nào của 8051 mà họ thích với điều kiện họ phải để mã lại tương thích với 8051. Điều này dẫn đến sự ra đời nhiều phiên bản của 8051 với các tốc độ khác nhau và dung lượng ROM trên chíp khác nhau được bán bởi hơn nửa các nhà sản xuất. Điều này quan trọng là mặc dù có nhiều biến thể khác nhau của 8051 về tốc độ và dung lương nhớ ROM trên chíp, nhưng tất cả chúng đều tương thích với 8051 ban đầu về các lệnh. Điều này có nghĩa là nếu ta viết chương trình của mình cho một phiên bản nào đó thì nó cũng sẽ chạy với mọi phiên bản bất kỳ khác mà không phân biệt nó từ hãng sản xuất nào.

Bảng 1.3: Các đặc tính của 8051 đầu tiên.

Đặc tính	**Số lượng**
ROM trên chíp	4K byte
RAM	128 byte
Bộ định thời	2
Các chân vào - ra	32
Cổng nối tiếp	1
Nguồn ngắt	6

1.2.2 Bộ ví điều khiển 8051

Bộ vi điều khiển 8051 là thành viên đầu tiên của họ 8051. Hãng Intel ký hiệu nó như là MCS51. Bảng 3.1 trình bày các đặc tính của 8051.



Hình 1.2: Bố trí bên trong của sơ đồ khối 8051.

1.2.3 các thành viên khác của họ 8051

Có hai bộ vi điều khiển thành viên khác của họ 8051 là 8052 và 8031.

a- Bộ vi điều khiển 8052:

Bộ vi điều khiển 8052 là một thành viên khác của họ 8051, 8052 có tất cả các đặc tính chuẩn của 8051 ngoài ra nó có thêm 128 byte RAM và một bộ định thời nữa. Hay nói cách khác là 8052 có 256 byte RAM và 3 bộ định thời. Nó cũng có 8K byte ROM. Trên chíp thay vì 4K byte như 8051. Xem bảng 1.4.

Bảng1.4: so sánh các đặc tính của các thành viên họ 8051.

Đặc tính	**8051**	**8052**	**8031**
ROM trên chíp	4K byte	8K byte	OK
RAM	128 byte	256 byte	128 byte
Bộ định thời	2	3	2
Chân vào - ra	32	32	32
Cổng nối tiếp	1	1	1
Nguồn ngắt	6	8	6

Như nhìn thấy từ bảng 1.4 thì 8051 là tập con của 8052. Do vậy tất cả mọi chương trình viết cho 8051 đều chạy trên 8052 nhưng điều ngược lại là không đúng.

b- Bộ vi điều khiển 8031:

Một thành viên khác nữa của 8051 là chíp 8031. Chíp này thường được coi như là 8051 không có ROM trên chíp vì nó có OK

byte ROM trên chíp. Để sử dụng chíp này ta phải bổ xung ROM ngoài cho nó. ROM ngoài phải chứa chương trình mà 8031 sẽ nạp và thực hiện. So với 8051 mà chương trình được chứa trong ROM trên chíp bị giới hạn bởi 4K byte, còn ROM ngoài chứa chương trinh được gắn vào 8031 thì có thể lớn đến 64K byte. Khi bổ xung cổng, như vậy chỉ còn lại 2 cổng để thao tác. Để giải quyết vấn đề này ta có thể bổ xung cổng vào - ra cho 8031. Phối phép 8031 với bộ nhớ và cổng vào - ra chẳng hạn với chíp 8255 được trình bày ở chương 14. Ngoài ra còn có các phiên bản khác nhau về tốc độ của 8031 từ các hãng sản xuất khác nhau.

1.2.4. Các bộ vi điều khiển 8051 từ các hãng khác nhau.

Mặc dù 8051 là thành viên phổi biến nhất của họ 8051 nhưng chúng ta sẽ thấy nó trong kho linh kiện. Đó là do 8051 có dưới nhiều dạng kiểu bộ nhớ khác nhau như UV - PROM, Flash và NV - RAM mà chúng đều có số đăng ký linh kiện khác nhau. Việc bàn luận về các kiểu dạng bộ nhớ ROM khác nhau sẽ được trình bày ở chương 14. Phiên bản UV-PROM của 8051 là 8751. Phiên bản Flash ROM được bán bởi nhiều hãng khác nhau chẳng hạn của Atmel corp với tên gọi là AT89C51 còn phiên bản NV-RAM của 8051 do Dalas Semi Conductor cung cấp thì được gọi là DS5000. Ngoài ra còn có phiên bản OTP (khả trình một lần) của 8051 được sản xuất bởi rất nhiều hãng.

a- Bộ vi điều khiển 8751:

Chíp 8751 chỉ có 4K byte bộ nhớ UV-EPROM trên chíp. Để sử dụng chíp này để phát triển yêu cầu truy cập đến một bộ đốt PROM cũng như bộ xoá UV- EPROM để xoá nội dung của bộ nhớ UV-EPROM bên trong 8751 trước khi ta có thể lập trình lại nó. Do một thực tế là ROM trên chíp đối với 8751 là UV-EPROM nên cần phải mất 20 phút để xoá 8751 trước khi nó có thể được lập trình trở lại. Điều này đã dẫn đến nhiều nhà sản xuất giới thiệu các phiên bản Flash Rom và UV-RAM của 8051. Ngoài ra còn có nhiều phiên bản với các tốc độ khác nhau của 8751 từ nhiều hãng khác nhau.

b- Bộ vi điều khiển AT8951 từ Atmel Corporation.

Chíp 8051 phổ biến này có ROM trên chíp ở dạng bộ nhớ Flash. Điều này là lý tưởng đối với những phát triển nhanh vì bộ nhớ Flash có thể được xoá trong vài giây trong tương quan so với 20 phút hoặc hơn mà 8751 yêu cầu. Vì lý do này mà AT89C51 để phát triển một hệ thống dựa trên bộ vi điều khiển yêu cầu một bộ đốt ROM mà

có hỗ trợ bộ nhớ Flash. Tuy nhiên lại không yêu cầu bộ xoá ROM. Lưu ý rằng trong bộ nhớ Flash ta phải xoá toàn bộ nội dung của ROM nhằm để lập trình lại cho nó. Việc xoá bộ nhớ Flash được thực hiện bởi chính bộ đốt PROM và đây chính là lý do tại sao lại không cần đến bộ xoá. Để loại trừ nhu cầu đối với một bộ đốt PROM hãng Atmel đang nghiên cứu một phiên bản của AT 89C51 có thể được lập trình qua cổng truyền thông COM của máy tính IBM PC .

Bảng 1.5: Các phiên bản của 8051 từ Atmel (Flash ROM).

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

Chữ C trong ký hiệu AT89C51 là CMOS.

Cũng có những phiên bản đóng vỏ và tốc độ khác nhau của những sản phẩm trên đây. Xem bảng 1.6. Ví dụ để ý rằng chữ "C" đứng trước số 51 trong AT 89C51 -12PC là ký hiệu cho CMOS "12" ký hiệu cho 12 MHZ và "P" là kiểu đóng vỏ DIP và chữ "C" cuối cùng là ký hiệu cho thương mại (ngược với chữ "M" là quân sự). Thông thường AT89C51 - 12PC rát lý tưởng cho các dự án của học sinh, sinh viên.

Bảng 1.6: Các phiên bản 8051 với tốc độ khác nhau của Atmel.

Mã linh kiện	Tốc độ	Số chân	Đóng vỏ	Mục đích
AT89C51-12PC	42MHZ	40	DTP	Thương mại

c- Bộ vi điều khiển DS5000 từ hãng Dallas Semiconductor.

Một phiên bản phổ biến khác nữa của 8051 là DS5000 của hãng Dallas Semiconductor. Bộ nhớ ROM trên chíp của DS5000 ở dưới dạng NV-RAM. Khả năng đọc/ ghi của nó cho phép chương trình được nạp vào ROM trên chíp trong khi nó vẫn ở trong hệ thống (không cần phải lấy ra). Điều này còn có thể được thực hiện thông qua cổng nối tiếp của máy tính IBM PC. Việc nạp chương trình trong hệ thống (in-system) của DS5000 thông qua cổng nối tiếp của PC làm cho nó trở thành một hệ thống phát triển tại chỗ lý tưởng. Một ưu việt của NV-RAM là khả năng thay đổi nội dung của ROM theo từng byte tại một thời điểm. Điều này tương phản với bộ nhớ Flash và EPROM mà bộ nhớ của chúng phải được xoá sạch trước khi lập trình lại cho chúng.

Bảng 1.7: Các phiên bản 8051 từ hãng Dallas Semiconductor.

Mã linh kiện	ROM	RAM	Chân I/O	Timer	Ngắt	Vcc	Đóng vỏ
DS5000-8	8K	128	32	2	6	5V	40
DS5000-32	32K	128	32	2	6	5V	40
DS5000T-8	8K	128	32	2	6	5V	40
DS5000T-8	32K	128	32	2	6	5V	40

Chữ "T" đứng sau 5000 là có đồng hồ thời gian thực.

Lưu ý rằng đồng hồ thời gian thực RTC là khác với bộ định thời Timer. RTC tạo và giữ thời gian l phút giờ, ngày, tháng - năm kể cả khi tắt nguồn.

Còn có nhiều phiên bản DS5000 với những tốc độ và kiểu đóng gói khác nhau.(Xem bảng 1.8). Ví dụ DS5000-8-8 có 8K NV-RAM và tốc đọ 8MHZ. Thông thường DS5000-8-12 hoặc DS5000T-8-12 là lý tưởng đối với các dự án của sinh viên.

Bảng 1.8:Các phiên bản của DS5000 với các tốc độ khác nhau

Mã linh kiện	NV- RAM	Tốc độ
DS5000-8-8	8K	8MHz
DS5000-8-12	8K	12MHz
DS5000-32-8	32K	8MHz

DS5000T-32-12 DS5000-32-12 DS5000-8-12	32K 32K 8K	8MHz (with RTC) 12MHz 12MHz (with RTC)

d- Phiên bản OTP của 8051.

Các phiên bản OTP của 8051 là các chíp 8051 có thể lập trình được một lần và được cung cấp từ nhiều hãng sản xuất khác nhau. Các phiên bản Flash và NV-RAM thường được dùng để phát triển sản phẩm mẫu. Khi một sản pohẩm được thiết kế và được hoàn thiện tuyệt đối thì phiên bản OTP của 8051 được dùng để sản hàng loạt vì nó sẽ hơn rất nhiều theo giá thành một đơn vị sản phẩm

e- Họ 8051 từ Hãng Philips

Một nhà sản xuất chính của họ 8051 khác nữa là Philips Corporation. Thật vậy, hãng này có một dải lựa chọn rộng lớn cho các bộ vi điều khiển họ 8051. Nhiều sản phẩm của hãng đã có kèm theo các đặc tính như các bộ chuyển đổi ADC, DAC, cổng I/0 mở rộng và cả các phiên bản OTP và Flash.

CHƯƠNG 2
Lập trình hợp ngữ 8051

2.1 Bên trong 8051.

Trong phần này chúng ta nghiên cứu các thanh ghi chính của 8051 và trình bày cách sử dụng với các lệnh đơn giản MOV và ADD.

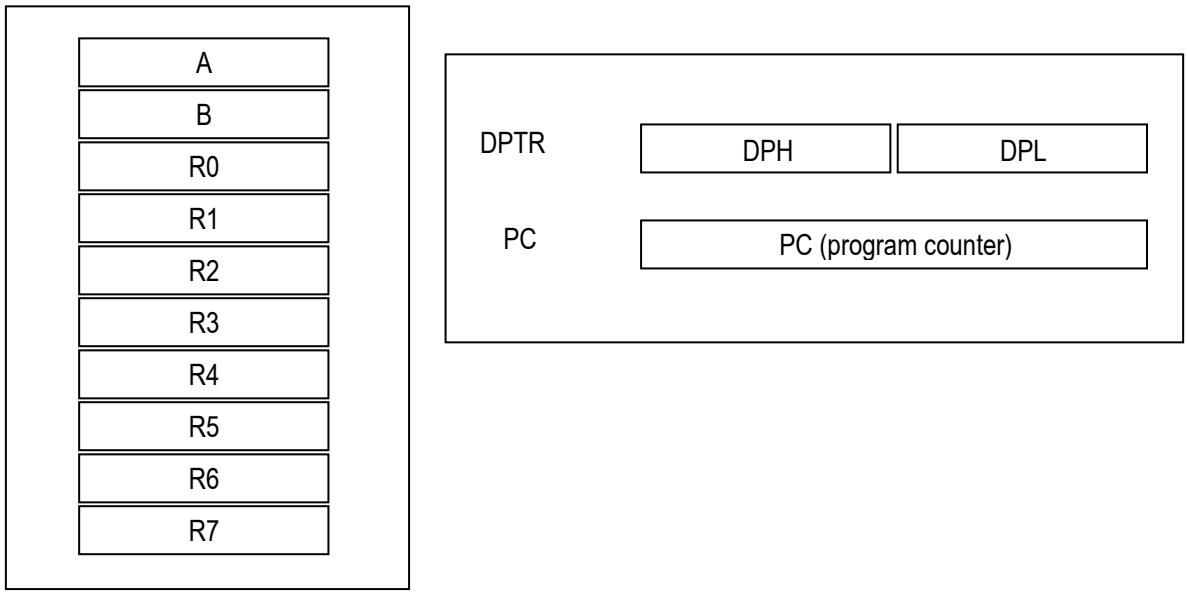
2.1.1 Các thanh ghi.

Trong CPU các thanh ghi được dùng để lưu cất thông tin tạm thời, những thông tin này có thể là một byte dữ liệu cần được sử lý hoặc là một địa chỉ đến dữ liệu cần được nạp. Phần lớn các thanh ghi của 8051 là các thanh ghi 8 bit. Trong 8051 chỉ có một kiểu dữ liệu: Loại 8 bit, 8 bit của một thanh ghi được trình bày như sau:

D7	D6	D5	D4	D3	D2	D1	D0

với MSB là bit có giá trị cao nhất D7 cho đến LSB là bit có giá trị thấp nhất D0. (MSB - Most Sigfican bit và LSB - Leart Significant Bit). Với một kiểu dữ liệu 8 bit thì bất kỳ dữ liệu nào lớn hơn 8 bit đều phải được chia thành các khúc 8 bit trước khi được xử lý. Vì có một số lượng lớn các thanh ghi trong 8051 ta sẽ tập trung vào một số thanh ghi công dụng chung đặc biệt trong các chương kế tiếp. Hãy tham khảo phụ lục Appendix A.3 để biết đầy đủ về các thanh ghi của 8051.

Hình 2.1: a) Một số thanh ghi 8 bit của 8051
b) Một số thanh ghi 16 bit của 8051



Các thanh ghi được sử dụng rộng rãi nhất của 8051 là A (thanh ghi tích luỹ), B, R0 - R7, DPTR (con trỏ dữ liệu) và PC (bộ đếm chương trình). Tất cả các dữ liệu trên đều là thanh g hi 8 bit trừ DPTR và PC là 16 bit. Thanh ghi tích luỹ A được sử dụng cho tất cả mọi phép toán số học và lô-gíc. Để hiểu sử dụng các thanh ghi này ta sẽ giới thiệu chúng trong các ví dụ với các lệnh đơn giản là ADD và MOV.

2.1.2 Lệnh chuyển MOV.

Nói một cách đơn giản, lệnh MOV sao chép dữ liệu từ một vị trí này đến một ví trí khác. Nó có cú pháp như sau:

```
MOV             ; Đích, nguồn; sao chép nguồn vào đích
```

Lệnh này nói CPU chuyển (trong thực tế là sao chép) toán hạng nguồn vào toán hạng đích. Ví dụ lệnh "MOV A, R0" sao chép nội dung thanh ghi R0 vào thanh ghi A. Sau khi lênh này được thực hiện thì thanh ghi A sẽ có giá trị giống như thanh ghi R0. Lệnh MOV không tác động toán hạng nguồn. Đoạn chương trình dưới đây đầu tiên là nạp thanh ghi A tới giá trị 55H 9là giá trị 55 ở dạng số Hex) và sau đó chuyển giá trị này qua các thanh ghi khác nhau bên trong CPU. Lưu ý rằng dấu "#" trong lệnh báo rằng đó là một giá trị. Tầm quan trọng của nó sẽ được trình bày ngay sau ví dụ này.

```
MOV   A, #55H;      ; Nạp trí trị 55H vào thanh ghi A (A = 55H)
MOV   R0, A         ; Sao chép nội dung A vào R0 (bây giờ R0=A)
MOV   R1, A         ; Sao chép nội dung A và R1 (bây giờ R1=R0=A)
MOV   R2, A         ; Sao chép nội dung A và R2 (bây giờ R2=R1=R0=A)
MOV   R3, #95H      ; Nạp giá trị 95H vào thanh ghi R3 (R3 = 95H)
MOV   A, R3         ; Sáo chép nội dung R3 vào A (bây giờ A = 95H)
```

Khi lập trình bộ vi điều khiển 8051 cần lưu ý các điểm sau:

1. Các giá trị có thể được nạp vào trực tiếp bất kỳ thanh ghi nào A, B, R0 - R7. Tuy nhiên, để thông báo đó là giá trị tức thời thì phải đặt trước nó một ký hiệu "#" như chỉ ra dưới đây.

```
MOV   A, #23H       ; Nạp giá trị 23H vào A (A = 23H)
MOV   R0, #12H      ; Nạp giá trị 12H vào R0 (R0 = 2BH)
MOV   R1, #1FH      ; Nạp giá trị 1FH vào R1 (R1 = 1FH)
MOV   R2, #2BH      ; Nạp giá trị 2BH vào R2 (R2 = 2BH)
MOV   B, # 3CH      ; Nạp giá trị 3CH vào B (B = 3CH)
MOV   R7, #9DH      ; Nạp giá trị 9DH vào R7 (R7 = 9DH)
MOV   R5, #0F9H     ; Nạp giá trị F9H vào R5 (R5 = F9H)
MOV   R6, #12       ;Nạp giá trị thập phân 12 = 0CH vào R6
                    (trong R6 có giá trị 0CH).
```

Để ý trong lệnh "MOV R5, #0F9H" thì phải có số 0 đứng trước F và sau dấu # báo rằng F là một số Hex chứ không phải là một ký tự. Hay nói cách khác "MOV R5, #F9H" sẽ gây ra lỗi.

2. Nếu các giá trị 0 đến F được chuyển vào một thanh ghi 8 bit thì các bit còn lại được coi là tất cả các số 0. Ví dụ, trong lệnh "MOV A,#5" kết quả là A=0.5, đó là A = 0000 0101 ở dạng nhị phân.
3. Việc chuyển một giá trị lớn hơn khả năng chứa của thanh ghi sẽ gây ra lỗi ví dụ:

```
MOV   A, #7F2H      ; Không hợp lệ vì 7F2H > FFH
MOV   R2, 456       ; Không hợp lệ vì 456 > 255 (FFH)
```

4. Để nạp một giá trị vào một thanh ghi thì phải gán dấu "#" trước giá trị đó. Nếu không có dấu thì nó hiểu rằng nạp từ một vị trí nhớ. Ví dụ "MOV A, 17H" có nghĩa là nạp giá trị trong ngăn nhớ có giá trị 17H vào thanh ghi A và tại địa chỉ đó dữ liệu có thể có bất kỳ giá trị nào từ 0 đến FFH. Còn để nạp giá trị là 17H vào thanh ghi A thì cần phải có dấu "#" trước 17H như thế này. "MOV A, #17H". Cần lưu ý rằng nếu thiếu dấu "#" trước một thì sẽ không gây lỗi vì hợp ngữ cho đó là một lệnh hợp

lệ. Tuy nhiên, kết quả sẽ không đúng như ý muốn của người lập trình. Đ ây sẽ là một lỗi thường hay gặp đối với lập trình viên mới.

2.1.3 Lệnh cộng ADD.

Lệnh cộng ADD có các phép như sau:

```
ADD a, nguồn          ; Cộng toán hạng nguồn vào thanh ghi A.
```

Lệnh cộng ADD nói CPU cộng byte nguồn vào thanh ghi A và đặt kết quả thanh ghi A. Để cộng hai số như 25H và 34H thì mỗi số có thể chuyển đến một thanh ghi và sau đó cộng lại với nhau như:

```
MOV   A, #25H       ; Nạp giá trị 25H vào A
MOV   R2, #34H      ; Nạp giá trị 34H vào R2
ADD   A, R2         ; Cộng R2 vào A và kết quả A = A + R2
```

Thực hiện chương trình trên ta được A = 59H (vì 25H + 34H = 59H) và R2 = 34H, chú ý là nội dụng R2 không thay đổi. Chương trình trên có thể viết theo nhiều cách phụ thuộc vào thanh ghi được sử dụng. Một trong cách viết khác có thể là:

```
MOV   R5, #25H      ; Nạp giá trị 25H vào thanh ghi R5
MOV   R7, #34H      ; Nạp giá trị 34H vào thanh ghi R7
MOV   A, #0         ; Xoá thanh ghi A (A = 0)
ADD   A, R5         ; Cộng nội dung R5 vào A (A = A + R5)
ADD   A, R7         ; Cộng nội dung R7 vào A (A = A + R7 = 25H + 34H)
```

Chương trình trên có kết quả trong A Là 59H, có rất nhiều cách để viết chương trình giống như vậy. Một câu hỏi có thể đặt ra sau khi xem đoạn chương trình trên là liệu có cần chuyển cả hai dữ liệu vào các thanh ghi trước khi cộng chúng với nhau không? Câu trả lời là không cần. Hãy xem đoạn chương trình dưới đây:

```
MOV   A, #25H       ; Nạp giá trị thứ nhất vào thanh ghi A (A = 25H)
ADD   A, #34H       ; Cộng giá trị thứ hai là 34H vào A (A = 59H)
```

Trong trường hợp trên đây, khi thanh ghi A đã chứa số thứ nhất thì giá trị thứ hai đi theo một toán hạng. Đây được gọi là toán hạng tức thời (trực tiếp).

Các ví dụ trước cho đến giờ thì lệnh ADD báo rằng toán hạng nguồn có thể hoặc là một thanh ghi hoặc là một dữ liệu trực tiếp (tức thời) nhưng thanh ghi đích luôn là thanh ghi A, thanh ghi tích luỹ. Hay nói cách khác là một lệnh như "ADD R2, #12H" là lệnh không hợp lệ vì mọi phép toán số học phải cần đến thanh ghi A và lệnh "ADD R4, A" cũng không hợp lệ vì A luôn là thanh ghi đích cho mọi phép số học. Nói một cách đơn giản là trong 8051 thì mọi phép toán số học đều cần đến thanh A với vai trò là toán hạng đích. Phần trình bày trên đây giải thích lý do vì sao thanh ghi A như là thanh thi tích luỹ. Cú pháp các lệnh hợp ngữ mô tả cách sử dụng chúng và liệt kê các kiểu toán hạng hợp lệ được cho trong phụ lục Appendix A.1.

Có hai thanh ghi 16 bit trong 8051 là bộ đếm chương trình PC và con trỏ dữ liệu APTR. Tầm quan trọng và cách sử dụng chúng được trình bày ở mục 2.3. Thanh ghi DPTR được sử dụng để truy cập dữ liệu và được làm kỹ ở chương 5 khi nói về các chế độ đánh địa chỉ.

2.2 Giới thiệu về lập trình hợp ngữ 8051.

Trong phần này chúng ta bàn về dạng thức của hợp ngữ và định nghĩa một số thuật ngữ sử dụng rộng rãi gắn liền với lập trình hợp ngữ.

CPU chỉ có thể làm việc với các số nhị phân và có thể chạy với tốc độ rất cao. Tuy nhiên, thật là ngán ngậm và chậm chạp đối với con người phải làm việc với các số 0 và 1 để lập trình cho máy tính. Một chương trình chứa các số 0 và 1 được gọi là ngôn ngữ máy.

Trong những ngày đầu của máy tính, các lập trình viên phải viết mã chương trình dưới dạng ngôn ngữ máy. Mặc dụ hệ thống thập lục phân (số Hex) đã được sử dụng như một cách hiệu quả hơn để biểu diễn các số nhị phân thì quá trình làm việc với mã máy vẫn còn là công việc cồng kềnh đối với con người. Cuối cùng, các nguồn ngữ hợp ngữ đã được phát, đã cung cấp các từ gợi nhớ cho các lệnh mã máy cộng với những đặc tính khác giúp cho việc lập trình nhanh hơn và ít mắc lỗi hơn. Thuật ngữ từ gợi nhớ (mnemonic) thường xuyên sử dụng trong tài liệu khoa học và kỹ thuật máy tính để tham chiếu cho các mã và từ rút gọn tương đối dễ nhớ, các chương trình hợp ngữ phải được dịch ra thanh mã máy bằng một chương trình được là trình hợp ngữ (hợp dịch). Hợp ngữ được coi như là một ngông ngữ bậc thấp vì nó giao tiếp trực tiếp với cấu trúc bên trong của CPU. Để lập trình trong hợp ngữ, lập trình viên phải biết tất cả các thanh ghi của CPU và kích thước của chúng cũng như các chi tiết khác.

Ngày nay, ta có thể sử dụng nhiều ngôn ngữ lập trình khác nhau, chẳng hạn như Basic, Pascal, C, C^{++}, Java và vô số ngôn ngữ khác. Các ngôn ngữ này được coi là nhưng ngôn ngữ bậc cao vì lập trình viên không cần phải tương tác với các chi tiết bên trong của CPU. Một trình hợp dịch được dùng để dịch chương trình hợp ngữ ra mã máy còn (còn đôi khi cũng còn được gọi mà đối tượng (Object Code) hay mã lệnh Opcode), còn các ngôn ngữ bậc cao được dịch thành các ngôn ngữ mã máy bằng một chương trình gọi là trình biên dịch. Ví dụ, để viết một chương trình trong C ta phải sử dụng một trình biên dịch C để dịch chương trình về dạng mã máy. Bây giờ ta xét dạng thức hợp ngữ của 8051 và sử dụng trình hợp dịch để tạo ra một chương trình sẵn sàng chạy ngay được.

2.2.1 Cấu trúc của hợp ngữ.

Một chương trình hợp ngữ bao gồm một chuỗi các dòng lệnh hợp ngữ. Một lệnh hợp ngữ có chứa một từ gợi nhớ (mnemonic) và tuy theo từng lệnh và sau nó có một hoặc hai toán hạng. Các toán hạng là các dữ liệu cần được thao tác và các từ gợi nhớ là các lệnh đối với CPU nói nó làm gì với các dữ liệu.

```
        ORG   0H          ; Bắt đầu (origin) tại ngăn nhớ 0
        MOV   R5, #25H    ; Nạp 25H vào R5
        MOV   R7, #34H    ; Nạp 34H vào R7
        MOV   A, #0       ; Nạp 0 vào thanh ghi A
        ADD   A, R5       ; Cộng nôi dụng R5 vào A (A = A + R5)
        ADD   A, R7       ; Cộng nội dung R7 vào A (A = A + R7)
        ADD   A, #121H    ; Cộng giá trị 12H vào A (A = A + 12H)
HERE: SJMP HERE           ; ở lại trong vòng lặp này
        END               ; Kết thúc tệp nguồn hợp ngữ
```

Chương trình 2.1: Ví dụ mẫu về một chương trình hợp ngữ.

Chương trình 2.1 cho trên đây là một chuỗi các câu lệnh hoặc các dòng lệnh được viết hoặc bằng các lệnh hợp ngữ như ADD và MOV hoặc bằng các câu lệnh được gọi là các chỉ dẫn. Trong khi các lệnh hợp ngữ thì nói CPU phải làm gì thì các chỉ lệnh

(hay còn gọi là giả lệnh) thì đưa ra các chỉ lệnh cho hợp ngữ. Ví dụ, trong chương trình 2.1 thì các lệnh ADD và MOV là các lệnh đến CPU, còn ORG và END là các chỉ lệnh đối với hợp ngữ. ORG nói hợp ngữ đặt mã lệnh tại ngăn nhớ 0 và END thì báo cho hợp ngữ biết kết thúc mã nguồn. Hay nói cách khác một chỉ lệnh để bắt đầu và chỉ lệnh thứ hai để kết thúc chương trình.

Cấu trúc của một lệnh hợp ngữ có 4 trường như sau:

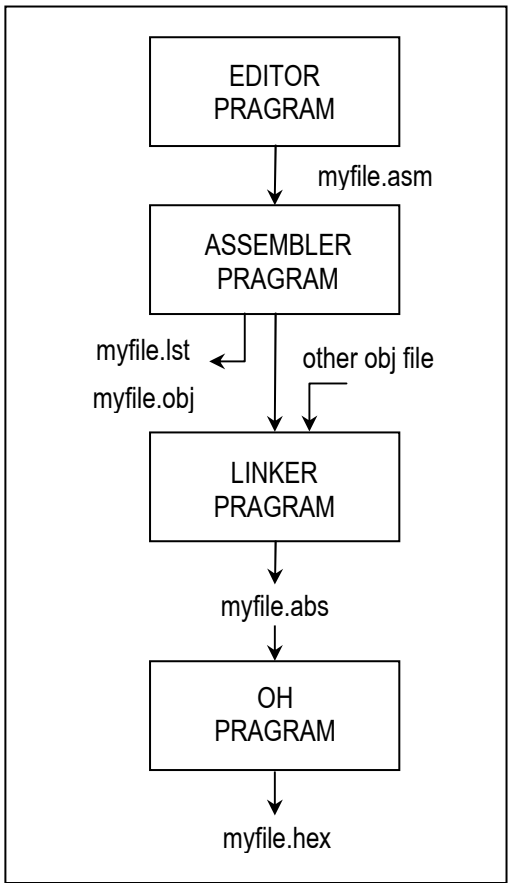
[nhãn:] [từ gợi nhớ] [các toán hạng] [; chú giải]

Các trường trong dấu ngoặc vuông là tuỳ chọn và không phải dòng lệnh nào cũng có chúng. Các dấu ngoặc vuông không được viết vào. Với dạng thức trên đây cần lưu ý các điểm sau:

1. Trường nhãn cho phép chương trình tham chiếu đến một dòng lệnh bằng tên. Nó không được viết quá một số ký tự nhất định. Hãy kiểm tra quy định này của hợp ngữ mà ta sử dụng.
2. Từ gợi nhớ (lệnh) và các toán hạng là các trường kết hợp với nhau thực thi công việc thực tế của chương trình và hoàn thiện các nhiệm vụ mà chương trình được viết cho chúng. Trong hợp ngữ các câu lệnh như:

   " ADD A, B"
   "MOV A, #67H"

   thì ADD và MOV là những từ gợi nhớ tạo ra mã lệnh, còn "A, B" và "A, #67H" là những toán hạng thì hai trường có thể chứa các lệnh giả hoặc chỉ lệnh của hợp ngữ. Hãy nhớ rằng các chỉ lệnh không tạo ra mã lệnh nào (mã máy) và chúng chỉ dùng bởi hợp ngữ, ngược lại đối với các lệnh là chúng được dịch ra mã máy (mã lênh) cho CPU thực hiện. Trong chương trình 2.1 các lệnh ORG và END là các chỉ lệnh (một số hợp ngữ của 8051 sử dụng dạng .ORG và .END). Hãy đọc quy định cụ thể của hợp ngữ ta sử dụng.
3. Chương chú giải luôn phải bắt đầu bằng dấu chấm phẩy (;). Các chú giải có thể bắt đầu ở đầu dòng hoặc giữa dòng. Hợp ngữ bỏ qua (làm ngơ) các chú giải nhưng chúng lại rất cần thiết đối với lập trình viên. Mặc dù các chú giải là tuỳ chọn, không bắt buộc nhưng ta nên dùng chúng để mô tả chương trình để giúp cho người khác đọc và hiểu chương trình dễ dàng hơn.
4. Lưu ý đến nhãn HERE trong trường nhãn của chương trình 2.1. Một nhãn bất kỳ tham chiếu đến một lệnh phải có dấu hai chấm (:) đứng ở sau. Trong câu lệnh nhảy ngắn SJMP thì 8051 được ra lệnh ở lại trong vòng lặp này vô hạn. Nếu hệ thống của chúng ta có một chương trình giám sát thì takhông cần dòng lệnh này và nó có thể được xoá đi ra khỏi chương trình.



2.3 Hợp dịch và chạy một chương trình 8051.

Như vậy cấu trúc của một chương trình hợp ngữ ta đã được biết, câu hỏi đặt ra là chương

trình sẽ được tạo ra và hợp dịch như thế nào và làm thế nào để có thể chạy được? Các bước để tạo ra một chương trình hợp ngữ có thể chạy được là:

1. Trước hết ta sử dụng mộ trình soạn thảo để gỡ vào một chương trình giống như chương trình 2.1. Có nhiều trình soạn thảo tuyệt vời hoặc các bộ sử lý từ được sử dụng để tạo ra và/ hoặc để soạn thảo chương trình. Một trình soạn thảo được sử dụng rộng rãi là trình soạn thảo EDIT của MS-DOS (hoặc Noterad của Windows) đều chạy trên hệ điều hành Microsoft. Lưu ý rằng, trình soạn thảo phải có khả năng tạo ra tệp mã ASCII. Đối với nhiều trình hợp ngữ thì các tên tệp tuân theo các quy ước thường lệ củ DOS, nhưng phần mở rộng của các tệp nguồn phải là "asm" hay "src" tuỳ theo trình hợp ngữ mà ta sử dụng.
2. Tệp nguồn có phần mở rộng "asm" chứa mã chương trình được tạo ra ở bước 1 được nạp vào trình hợp dịch của 8051. Trình hợp dịch chuyển các lệnh ra mã máy. Trình hợp dịch sẽ tạo ra một tệp đối tượng và một tệp liệt kê với các thành phần mở rộng "obj" và "lst" tương ứng.
3. Các trình hợp dịch yêu cầu một bước thứ ba gọi là liên kết. Chương trình liên kết lấy một hoặc nhiều tệp đối tượng và tạo ra một tệp đối tượng tuyệt đối với thành phần mở rộng "abs". Tệp "abs" này được sử dụng bởi thùng chứa của 8051 có một chương trình giám sát.
4. Kế sau đó tệp "abs" được nạp vào một chương trình được gọi là "0H" (chuyển đối tượng object về dạng số Hex) để tạo ra một tệp với đuôi mở rộng "Hex" có thể nạp tốt vào trong ROM. Chương trình này có trong tất cả mọi trình hợp ngữ của 8051 các trình hợp ngữ dựa trên Windows hiện nay kết hợp các bước 2 đến 4 vào thành một bước.

Hình 2.2: Các bước để tạo ra một chương trình.

2.3.1 Nói thêm về các tệp ".asm" và ".object".

Tệp ".asm" cũng được gọi là tệp nguồn và chính vì lý do này mà một số trình hợp ngữ đòi hỏi tệp này phải có một phần mở rộng "src" từ chữ "source" là nguồn. Hãy kiểm tra hợp ngữ 8051 mà ta sử dụng xem nó có đòi hỏi như vậy không? Như ta nói trước đây tệp này được tạo ra nhờ một trình biên tập chẳng hạn như Edit của DOS hoặc Notepad của Windows. Hợp ngữ của 8051 chuyển đổi các tệp hợp ngữ trong tệp .asm thành ngôn ngữ mã máy và cung cấp tệp đối tượng .object. Ngoài việc tạo ra tệp đối tượng trình hợp ngữ cũng cho ra tệp liệt kê "lst" (List file).

2.3.2 Tệp liệt kê ".lst".

Tệp liệt kê là một tuỳ chọn, nó rất hữu ích cho lập trình viên vì nó liệt kê tất cả mọi mã lệnh và địa chỉ cũng như tất cả các lỗi mà trình hợp ngữ phát hiện ra. Nhiều trình hợp ngữ giả thiết rằng, tệp liệt kê là không cần thiết trừ khi ta báo rằng ta muốn tạo ra nó. Tệp này có thể được truy cập bằng một trình biên dịch như Edit của DOS hoặc Notepad của Window và được hiển thị trên màn hình hoặc được gửi ra máy in. Lập trình viên sử dụng tệp liệt kê để tìm các lỗi cú pháp. Chỉ sau khi đã sửa hết các lỗi được đánh dấu trong tệp liệt kê thì tệp đối tượng mới sẵn sàng làm đầu vào cho chương trình liên kết.

```
1 0000              ORG   0H          ; Bắt đầu ở địa chỉ 0
2 0000  7D25        MOV   R5, #25H    ; Nạp giá trị 25H vào R5
3 0002 7F34         MOV   R7, #34H    ; Nạp giá trị 34H vào R7
4 0004  7400        MOV   A, #0       ; Nạp 0 vào A (xoá A)
5 0006 2D           ADD   A, R5       ; Cộng nội dung R5 vào A (A = A + R5)
6 0007 2F           ADD   A, R7       ; Cộng nội dung R7 vào A (A = A + R7)
7 0008 2412         ADD   A, #12H     ; Cộng giá trị 12H vào A (A = A + 12H)
```

```
8 00A  BCEF  HERE: SJMP HERE          ; ở lại vòng lặp này
9 000C             END                ; Kết thúc tệp .asm
```

Chương trình 2.2: Tệp liệt kê.

2.4 Bộ đếm chương trình và không gian ROM trong 8051.

2.4.1 Bộ đếm chương trình trong 8051.

Một thanh ghi quan trọng khác trong 8051 là bộ đếm chương trình . Bộ đếm chương trình chỉ đếm địa chỉ của lệnh kế tiếp cần được thực hiện. Khi CPU nạp mã lệnh từ bộ nhớ ROM chương trình thì bộ đếm chương trình tăng lên chỉ đếm lệnh kết tiếp. Bộ đếm chương trình trong 8051 có thể truy cập các địa chỉ chương trình trong 8051 rộng 16 bit. Điều này có nghĩa là 8051 có thể truy cập các địa chỉa chương trình từ 0000 đến FFFFH tổng cộng là 64k byte mã lệnh. Tuy nhiên, không phải tất cả mọi thành viên của 8051 đều có tất cả 64k byte ROM trên chíp được cài đặt. Vậy khi 8051 được bật nguồn thì nó đánh thức ở địa chỉa nào?

2.4.2 Địa chỉ bắt đầu khi 8051 được cấp nguồn.

Một câu hỏi mà ta phải hỏi về bộ vi điều khiển bất kỳ là thì nó được cấp nguồn thì nó bắt đầu từ địa chỉ nào? Mỗi bộ vi điều khiển đều khác nhau. Trong trường hợp họ 8051 thì mọi thành viên kể từ nhà sản xuất nào hay phiên bản nào thì bộ vi điều khiển đều bắt đầu từ địa chỉ 0000 khi nó được bật nguồn. Bật nguồn ở đây có nghĩa là ta cấp điện áp V_{cc} đến chân RESET như sẽ trình bày ở chương 4. Hay nói cách khác, khi 8051 được cấp nguồn thì bộ đếm chương trình có giá trị 0000. Điều này có nghĩa là nó chờ mã lệnh đầu tiên được lưu ở địa chỉa ROM 0000H. Vì lý do này mà trong vị trí nhớ 0000H của bộ nhở ROM chương trình vì đây là nơi mà nó tìm lệnh đầu tiên khi bật nguồn. Chúng ta đạt được điều này bằng câu lệnh ORG trong chương trình nguồn như đã trình bày trước đây. Dưới đây là hoạt động từng bước của bộ đếm chương trình trong qúa trình nạp và thực thi một chương trình mẫu.

2.4.3 Đặt mã vào ROM chương trình.

Để hiểu tốt hơn vai trò của bộ đếm chương trình trong quá trình nạp và thực thi một chương trình, ta khảo sát một hoạt động của bộ đếm chương trình khi mỗi lệnh được nạp và thực thi. Trước hết ta khảo sát một lần nữa tệp liệt kê của chương trình mẫu và cách đặt mã vào ROM chương trình 8051 như thế nào? Như ta có thể thấy, mã lệnh và toán hạng đối với mỗi lệnh được liệt kê ở bên trái của lệnh liệt kê.

Chương trình 2.1: Ví dụ mẫu về một chương trình hợp ngữ.

Chương trình 2.1 cho trên đây là một chuỗi các câu lệnh hoặc các dòng lệnh được viết hoặc bằng các lệnh hợp ngữ như ADD và MOV hoặc bằng các câu lệnh được gọi là các chỉ dẫn. Trong khi các lệnh hợp ngữ thì nói CPU phải làm gì thì các chỉ lệnh (hay còn gọi là giả lệnh) thì đưa ra các chỉ lệnh cho hợp ngữ. Ví dụ, trong chương trình 2.1 thì các lệnh ADD và MOV là các lệnh đến CPU, còn ORG và END là các chỉ lệnh đối với hợp ngữ. ORG nói hợp ngữ đặt mã lệnh tại ngăn nhớ 0 và END thì báo cho hợp ngữ biết kết thúc mã nguồn. Hay nói cách khác một chỉ lệnh để bắt đầu và chỉ lệnh thứ hai để kết thúc chương trình.

Cấu trúc của một lệnh hợp ngữ có 4 trường như sau:

[nhãn:] [từ gợi nhớ] [các toán hạng] [; chú giải]

Các trường trong dấu ngoặc vuông là tuỳ chọn và không phải dòng lệnh nào cũng có chúng. Các dấu ngoặc vuông không được viết vào. Với dạng thức trên đây cần lưu ý các điểm sau:

Trường nhãn cho phép chương trình tham chiếu đến một dòng lệnh bằng tên. Nó không được viết quá một số ký tự nhất định. Hãy kiểm tra quy định này của hợp ngữ mà ta sử dụng.

Từ gợi nhớ (lệnh) và các toán hạng là các trường kết hợp với nhau thực thi công việc thực tế của chương trình và hoàn thiện các nhiệm vụ mà chương trình được viết cho chúng. Trong hợp ngữ các câu lệnh như:

" ADD A, B"
"MOV A, #67H"

Thì ADD và MOV là những từ gợi nhớ tạo ra mã lệnh, còn "A, B" và "A, #67H" là những toán hạng thì hai trường có thể chứa các lệnh giả hoặc chỉ lệnh của hợp ngữ. Hãy nhớ rằng các chỉ lệnh không tạo ra mã lệnh nào (mã máy) và chúng chỉ dùng bởi hợp ngữ, ngược lại đối với các lệnh là chúng được dịch ra mã máy (mã lênh) cho CPU thực hiện. Trong chương trình 2.1 các lệnh ORG và END là các chỉ lệnh (một số hợp ngữ của 8051 sử dụng dạng .ORG và .END). Hãy đọc quy định cụ thể của hợp ngữ ta sử dụng.

Trương chú giải luôn phải bắt đầu bằng dấu chấm phẩy (;). Các chú giải có thể bắt đầu ở đầu dòng hoặc giữa dòng. Hợp ngữ bỏ qua (làm ngơ) các chú giải nhưng chúng lại rất cần thiết đối với lập trình viên. Mặc dù các chú giải là tuỳ chọn, không bắt buộc nhưng ta nên dùng chúng để mô tả chương trình để giúp cho người khác đọc và hiểu chương trình dễ dàng hơn.

Lưu ý đến nhãn HERE trong trường nhãn của chương trình 2.1. Một nhãn bất kỳ tham chiếu đến một lệnh phải có dấu hai chấm (:) đứng ở sau. Trong câu lệnh nhảy ngắn SJMP thì 8051 được ra lệnh ở lại trong vòng lặp này vô hạn. Nếu hệ thống của chúng ta có một chương trình giám sát thì takhông cần dòng lệnh này và nó có thể được xoá đi ra khỏi chương trình.

Địa chỉ	Mã lệnh
0000	7D
0001	25
0002	F7
0003	34
0004	74
0005	00
0006	2D
0007	2F
0008	24
0009	12
000A	80
000B	FE

Chương trình 2.1: Tệp liệt kê

Sau khi chương trình được đốt vào trong ROM của thành viên họ 8051 như 8751 hoặc AT 8951 hoặc DS 5000 thì mã lệnh và toán hạng được đưa vào các vị trí nhớ ROM bắt đầu từ địa chỉ 0000 như bảng liệt kê dưới đây.

Địa chỉ ROM	Ngôn ngữ máy	Hợp ngữ
0000	7D25	MOV R5, #25H
0002	7F34	MOV R7, #34H
0004	7400	MOV A, #0
0006	2D	ADD A, R5
0007	2F	ADD A, R7
0008	2412	ADD A, #12H
000A	80EF	HERE: SJMP HERE

Bảng nội dung ROM của chương trình 2.1.

Bảng liệt kê chỉ ra địa chỉ 0000 chứa mã 7D là mã lệnh để chuyển một giá trị vào thanh ghi R5 và địa chỉ 0001 chứa toán hạng (ở đây là giá trị 254) cần được chuyển vào R5. Do vậy, lệnh "MOV R5, #25H" có mã là "7D25" trong đó 7D là mã lệnh,

còng 25 là toán hạng. Tương tự như vậy, mã máy "7F34" được đặt trong các ngăn nhớ 0002 và 0003 và biểu diễn mã lệnh và toán hạng đối với lệnh "MOV R7, #34H". Theo cách như vậy, mã máy "7400" được đặt tại địa chỉ 0004 và 0005 và biểu diễn mã lệnh và toán hạng đối với lệnh "MOV A, #0". Ngăn nhớ 0006 có mã 2D là mã đối với lệnh "ADD A, R5" và ngăn nhớ 0007 có nội dung 2F là mã lệnh cho "ADD A, R7". Mã lệnh đối với lệnh "ADD A, #12H" được đặt ở ngăn nhớ 0008 và toán hạng 12H được đặt ở ngăn nhớ 0009. Ngăn nhớ 000A có mã lệnh của lệnh SJMP và địa chỉ đích của nó được đặt ở ngăn nhớ 000B. Lý do vì sao địa chỉ đích là FE được giải thích ở chương 3.

2.4.4 Thực hiện một chương trình theo từng byte.

Giả sử rằng chương trình trên được đốt vào ROM của chíp 8051 hoặc(8751, AT 8951 hoặc DS 5000) thì dưới đây là mô tả hoạt động theo từng bước của 8051 khi nó được cấp nguồn.

1. Khi 8051 được bật nguồn, bộ đếm chương trình PC có nội dung 0000 và bắt đầu nạp mã lệnh đầu tiên từ vị trí nhớ 0000 của ROM chương trình. Trong trường hợp của chương trình này là mã 7D để chuyển một toán hạng vào R5. Khi thực hiện mã lệnh CPU nạp giá trị 25 vào bộ đếm chương trình được tăng lên để chỉ đến 0002 (PC = 0002) có chứa mã lệnh 7F là mã của lệnh chuyển một toán hạng vào R7 "MOV R7, ...".
2. Khi thực hiện mã lệnh 7F thì giá trị 34H được chuyển vào R7 sau đó PC được tăng lên 0004.
3. Ngăn nhớ 0004 chứa mã lệnh của lệnh "MOV A, #0". Lệnh này được thực hiện và bây giờ PC = 0006. Lưu ý rằng tất cả các lệnh trên đều là những lệnh 2 byte, nghĩa là mỗi lệnh chiếm hai ngăn nhớ.
4. Bây giờ PC = 0006 chỉ đến lệnh kế tiếp là "ADD A, R5". Đây là lệnh một byte, sau khi thực hiện lệnh này PC = 0007.
5. Ngăn nhớ 0007 chứa mã 2F là mã lệnh của "ADD A, R7". Đây cũng là lệnh một byte, khi thực hiện lệnh này PC được tăng lên 0008. Qúa trình này cứ tiếp tục cho đến khi tất cả moi lệnh đều được nạp và thực hiện. Thực tế mà bộ đếm chương trình chỉ đến lệnh kế tiếp cần được thực hiện giải thích tại sao một số bộ vi xử lý (đáng nói là × 86) gọi bộ đếm là con trỏ lệnh (Instruction Pointer).

2.4.5 Bản đồ nhớ ROM trong họ 8051.

Như ta đã thấy ở chương trước, một số thành viên họ 8051 chỉ có 4k byte bộ nhớ ROM trên chíp (ví dụ 8751, AT 8951) và một số khác như AT 8951 có 8k byte ROM, DS 5000-32 của Dallas Semiconductor có 32k byte ROM trên chíp. Dallas Semiconductor cũng có motọ 8051 với ROM trên chíp là 64k byte. Điểm cần nhớ là không có thành viên nào của họ 8051 có thể truy cập được hơn 64k byte mã lệnh vì bộ đếm chương trình của 8051 là 16 bit (dải địa chỉ từ 0000 đến FFFFH). Cần phải ghi nhớ là lệnh đầu tiên của ROM chương trình đều đặt ở 0000, còn lệnh cuối cùng phụ thuộc vào dung lượng ROM trên chíp của mỗi thành viên họ 8051. Trong số các thành viên họ 8051 thì 8751 và AT 8951 có 4k byte ROM trên chíp. Bộ nhớ ROM trên chíp này có các địa chỉ từ 0000 đến 0FFFH. Do vậy, ngăn nhớ đầu tiên có địa chỉ 0000 và ngăn nhớ cuối cùng có địa chỉa 0FFFH. Hãy xét ví dụ 2.1.

VÝ dô 2.1:

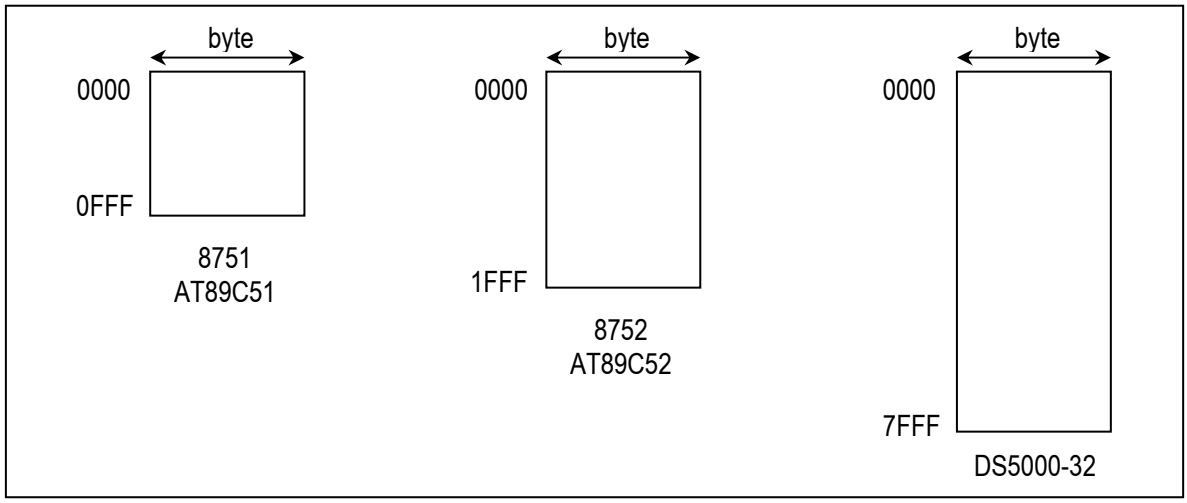
Tìm địa chỉ bộ nhớ ROM của mỗi thành viên họ 8051 sau đây.

a) AT 8951 (hoặc 8751) với 4k byte
b) DS 5000-32 với 32k byte

Lêi gi¶i:

a) Với 4k byte của không gian nhớ ROM trên chíp ta có 4096 byte bằng 1000H ở dạng Hex (4 × 1024 = 4096 hay 1000 ở dạng Hex). Bộ nhớ này được xắp xếp trong các ngăn nhớ từ 0000 đến 0FFFH. Lưu ý 0 luôn là ngăn nhớ đầu tiên.

b) Với 32k byte nhớ ta có 32.768 byte (32 × 1024). Chuyển đổi 32.768 về số Hex ta nhận được giá trị 8000H. Do vậy, không gian nhớ là dải từ 0000 đến 7FFFH.



Hình 2.3: Dải địa chỉ của ROM trên chíp một số thành viên họ 8051.

2.5 Các kiểu dữ liệu và các chỉ lệnh.

2.5.1 Kiểu dữ liệu và các chỉ lệnh của 8051.

Bộ vi điều khiển chỉ có một kiểu dữ liệu, nó là 8 bit và độ dài mỗi thanh ghi cũng là 8 bit. Công việc của lập trình viên là phân chia dữ liệu lớn hơn 8 bit ra thành từng khúc 8 bit (từ 00 đến FFH hay từ 0 đến 255) để CPU xử lý. Ví dụ về xử lý dữ liệu lớn hơn 8 bit được trình bày ở chương 6. Các dữ liệu được sử dụng bởi 8051 có thể là số âm hoặc số dương và về xử lý các số có dấu được bàn ở chương 6.

2.5.2 Chỉ lệnh DB (định nghĩa byte).

Chỉ lệnh DB là một chỉ lệnh dữ liệu được sử dụng rộng rãi nhất trong hợp ngữ. Nó được dùng để định nghĩa dữ liệu 8 bit. Khi DB được dùng để định nghĩa byte dữ liệu thì các số có thể ở dạng thập phân, nhị phân, Hex hoặc ở dạng thức ASII. Đối với dữ liệu thập phân thì cần đặt chữ "D" sau số thập phân, đối với số nhị phân thì đặt chữ "B" và đối với dữ liệu dạng Hex thì cần đặt chữ "H". Bất kể ta sử dụng số ở dạng thức nào thì hợp ngữ đều chuyển đổi chúng về thành dạng Hex. Để báo dạng thức ở dạng mã ASCII thì chỉ cần đơn giản đặt nó vào dấu nháy đơn 'như thế này'. Hợp ngữ sẽ gán mã ASCII cho các số hoặc các ký tự một cách tự động. Chỉ lệnh DB chỉ là chỉ lệnh mà có thể được sử dụng để định nghĩa các chuỗi ASCII lớn hơn 2 ký tự. Do vậy, nó có thể được sử dụng cho tất cả mọi định nghĩa dữ liệu ASCII. Dưới đây là một số ví dụ về DB:

```
        ORG    500H
DATA1:  DB     2B                ; Số thập phân (1C ở dạng Hex)
DATA2:  DB     00110101B         ; Số nhị phân (35 ở dạng Hex)
DATA3:  DB     39H               ; Số dạng Hex
               ORG    510H
DATA4:  DB     "2591"            ; Các số ASCII
               ORG    518H
DATA5   DB     "My name is Joe"  ; Các ký tự ASCII
```

Các chuỗi ASCII có thể sử dụng dấu nháy đơn 'như thế này' hoặc nháy kép "như thế này". Dùng dấu phẩy kép sẽ hữu ích hơn đối với trường hợp dấu nháy đơn được dùng sở hữu cách như thế này "Nhà O' Leary". Chỉ lệnh DB cũng được dùng để cấp phát bộ nhớ theo từng đoạn kích thước một byte.

2.5.3 Các chỉ lệnh của hợp ngữ.

1. Chỉ lệnh ORG: Chỉ lệnh ORG được dùng để báo bắt đầu của địa chỉ. Số đi sau ORG có kể ở dạng Hex hoặc thập phân. Nếu số này có kèm chữ H đằng sau thì là ở dạng Hex và nếu không có chữ H ở sau là số thập phân và hợp ngữ sẽ chuyển nó thành số Hex. Một số hợp ngữ sử dụng dấu chấm đứng trước "ORG" thay cho "ORG". Hãy đọc kỹ về trình hợp ngữ ta sử dụng.
2. Chỉ lệnh EQU: Được dùng để định nghĩa một hằng số mà không chiếm ngăn nhớ nào. Chỉ lệnh EQU không dành chỗ cất cho dữ liệu nhưng nó gắn một giá trị hằng số với nhãn dữ liệu sao cho khi nhãn xuất hiện trong chương trình giá trị hằng số của nó sẽ được thay thế đối với nhãn. Dưới đây sử dụng EQU cho hằng số bộ đếm và sau đó hằng số được dùng để nạp thanh ghi RS.

```
COUNT EQU 25
MOV   R3, #count
```

   Khi thực hiện lện "MOV R3, #COUNT" thì thanh ghi R3 sẽ được nạp giá trị 25 (chú ý đến dấu #). Vậy ưu điểm của việc sử dụng EQU là gì? Giả sử có một hằng số (một giá trị cố định) được dùng trong nhiều chỗ khác nhau trong chương trình và lập trình viên muốn thay đổi giá trị của nó trong cả chương trình. Bằng việc sử dụng chỉ lệnh EQU ta có thể thay đổi một số lần và hợp ngữ sẽ thay đổi tất cả mọi lần xuất hiện của nó là tìm toàn bộ chương trình và gắng tìm mọi lần xuất hiện.
3. Chỉ lệnh END: Một lệnh quan trọng khác là chỉ lệnh END. Nó báo cho trình hợp ngữ kết thúc của tệp nguồn "asm" chỉ lệnh END là dòng cuối cùng của chương trình 8051 có nghĩa là trong mã nguồn thì mọi thứ sau chỉ lệnh END để bị trình hợp ngữ bỏ qua. Một số trình hợp ngữ sử dụng .END có dấu chấm đứng trước thay cho END.

2.5.4 Các quy định đối với nhãn trong hợp ngữ.

Bằng cách chọn các tên nhãn có nghĩa là một lập trình viên có thể làm cho chương trình dễ đọc và dễ bảo trì hơn, có một số quy định mà các tên nhãn phải tuân theo. Thứ nhất là mỗi tên nhãn phải thống nhất, các tên được sử dụng làm nhãn trong hợp ngữ gồm các chữ cái viết hoa và viết thường, các số từ 0 đến 9 và các dấu đặc biệt như: dấu hỏi (?), dấu (≅), dấu gạch dưới (_), dấu đô là ($) và dấu chu kỳ (.). Ký tự đầu tiên của nhãn phải là một chữ cái. Hay nói cách khác là nó không thể là số Hex. Mỗi trình hợp ngữ có một số từ dự trữ là các từ gợi nhớ cho các lệnh mà không được dùng để làm nhãn trong chương trình. Ví dụ như "MOV" và "ADD". Bên cạnh các từ gợi nhớ còn có một số tự dự trữ khác, hãy kiểm tra bản liệt kê các từ dự phòng của hợp ngữ ta đang sử dụng.

2.6 Các bit cờ và thanh ghi đặc bệt PSW của 8051.

Cũng như các bộ vi xử lý khác, 8051 có một thanh ghi cờ để báo các điều kiện số học như bit nhớ. Thanh ghi cờ trong 8051 được gọi là thanh ghi từ trạng thái chương trình PSW. Trong phần này và đưa ra một số ví dụ về cách thay đổi chúng.

2.6.1 Thanh ghi từ trạng thái chương trình PSW.

Thanh ghi PSW là thanh ghi 8 bit. Nó cũng còn được coi như là thanh ghi cờ. Mặc dù thanh ghi PSW rộng 8 bit nhưng chỉ có 6 bit được 8051 sử dụng. Hai bit chưa dùng là các cờ ch người dùng định nghĩa. Bốn trong số các cờ được gọi là các cờ có điều kiện, có nghĩa là chúng báo một số điều kiện do kết quả của một lệnh vừa được thực hiện. Bốn cờ này là cờ nhớ CY (carry), cờ AC (auxiliary cary), cờ chẵn lẻ P (parity) và cờ tràn OV (overflow).

Như nhìn thấy từ hình 2.4 thì các bit PSW.3 và PSW.4 được gán như RS0 và RS1 và chúng được sử dụng để thay đổi các thanh ghi băng. Chúng sẽ được giải thích ở phần kế sau. Các bit PSW.5 và PSW.1 là các bit cờ trạng thái công dụng chung và lập trình viên có thể sử dụng cho bất kỳ mục đích nào.

CY	AC	F0	RS1	RS0	OV	–	P

CY PSW.7 ; Cờ nhớ
AC PSW.6 ; Cờ

- PSW.5 ; Dành cho người dùng sử dụng mục đích chung

RS1 PSW.4 ; Bit = 1 chọn băng thanh ghi
RS0 PSW.3 ; Bit = 0 chọn băng thanh ghi
OV PSW.2 ; Cờ bận

- PSW.1 ; Bit dành cho người dùng định nghĩa

P PSW.0 ; Cờ chẵn, lẻ. Thiết lập/ xoá bằng phần cứng mỗi chu kỳ lệnh báo tổng các số bit 1 trong thanh ghi A là chẵn/ lẻ.

RS1	RS0	Băng thanh ghi	Địa chỉ
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

H×nh 2.4: **C¸c bit cña thanh ghi PSW**

Dưới đây là giải thích ngắn gọn về 4 bit cờ của thanh ghi PSW.

1. Cờ nhớ CY: Cờ này được thiết lập mỗi khi có nhớ từ bit D7. Cờ này được tác động sau lệnh cộng hoặc trừ 8 bit. Nó cũng được thiết lập lên 1 hoặc xoá về 0 trực tiếp bằng lệnh "SETB C" và "CLR C" nghĩa là "thiết lập cờ nhớ" và "xoá cờ nhớ" tương ứng. Về các lệnh đánh địa chỉ theo bit được bàn kỹ ở chương 8.
2. Cờ AC: Cờ này báo có nhớ từ bit D3 sang D4 trong phép cộng ADD hoặc trừ SUB. Cờ này được dùng bởi các lệnh thực thi phép số học mã BCD (xem ở chương 6).
3. Cờ chẵn lẻ P: Cờ chẵn lẻ chỉ phản ánh số bit một trong thanh ghi A là chẵn hay lẻ. Nếu thanh ghi A chứa một số chẵn các bit một thì P = 0. Do vậy, P = 1 nếu A có một số lẻ các bit một.
4. Cờ chàn OV: Cờ này được thiết lập mỗi khi kết quả của một phép tính số có dấu quá lớn tạo ra bit bậc cao làm tràn bit dấu. Nhìn chung cờ nhớ được dùng để phát hiện lỗi trong các phép số học không dấu. Còn cờ tràn được dùng chỉ để phát hiện lỗi trong các phép số học có dấu và được bàn kỹ ở chương 6.

2.6.2 Lệnh ADD và PSW.

Bây giờ ta xét tác động của lệnh ADD lên các bit CY, AC và P của thanh ghi PSW. Một số ví dụ sẽ làm rõ trạng thái của chúng, mặc dù các bit cờ bị tác động bởi lệnh ADD là CY, P, AC và OV nhưng ta chỉ tập trung vào các cờ CY, AC và P, còn cờ OV sẽ được nói đến ở chương 6 vì nó liên quan đến phép tính số học số có dấu.

Các ví dụ 2.2 đến 2.4 sẽ phản ánh tác động của lệnh ADD lên các bit nói trên.

Bảng 2.1: Các lệnh tác động lên các bit cờ.

Instruction	CY	OV	AC
ADD	X	X	X
ADDC	X	X	X
SUBB	X	X	X
MUL	0	X	
DIV	0	X	
DA	X		
RRC	X		
RLC	X		
SETB C	1		
CLR C	0		
CPL C	X		
ANL C, bit	X		
ANL C,/ bit	X		
ORL C, bit	X		
ORL C,/bit	X		
MOV C, bit	X		
CJNE	X		

Ví dụ 2.2: Hãy trình bày trạng thái các bit cờ CY, AC và P sau lệnh cộng 38H với 2FH dưới đây:

```
MOV     A, #38H
ADD  A, #2FH              ; Sau khi cộng A = 67H, CY = 0
```

Lêi gi¶i:

```
    38        00111000
+   2F        00101111
-------------------------
    67        01100111
```

Cờ CY = 0 vì không có nhớ từ D7
Cờ AC = 1 vì có nhớ từ D3 sang D4
Cờ P = 1 vì thanh ghi A có 5 bit 1 (lẻ)

VÝ dô 2.3:

Hãy trình bày trạng thái các cờ CY, AC và P sau phép cộng 9CH với 64H.

Lêi gi¶i:

```
    9C        10011100
+   64        01100100
-------------------------
   100        00000000
```

Cờ CY = 1 vì có nhớ qua bit D7
Cờ AC = 1 vì có nhớ từ D3 sang D4
Cờ P = 0 vì thanh ghi A không có bit 1 nào (chẵn)

VÝ dô 2.4:

Hãy trình bày trạng thái các cờ CY, AC và P sau phép cộng 88H với 93H.

Lêi gi¶i:

```
    88        10001000
+   93        10010011
-------------------------
   11B        00011011
```

Cờ CY = 1 vì có nhớ từ bit D7
Cờ AC = 0 vì không có nhớ từ D3 sang D4
Cờ P = 0 vì số bit 1 trong A là 4 (chẵn)

2.7 Các băng thanh ghi và ngăn xếp của 8051.

Bộ vi điều khiển 8051 có tất cả 128 byte RAM. Trong mục này ta bàn vệ phân bố của 128 byte RAM này và khảo sát công dụng của chúng như các thanh ghi và ngăn xếp.

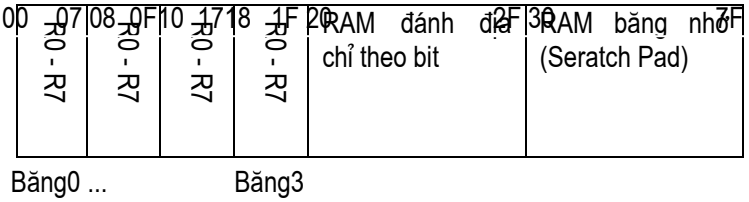
2.7.1 Phân bố không gian bộ nhớ RAM trong 8051.

Có 128 byte RAM trong 8051 (một số thành viên đang chú ý là 8052 có 256 byte RAM). 128 byte RAM bên trong 8051 được gán địa chỉ từ 00 đến 7FH. Như ta sẽ thấy ở chương 5, chúng có thể được truy cập trực tiếp như các ngăn nhớ 128 byte RAM này được phân chia thành từng nhóm như sau:

1. Tổng cộng 32 byte từ ngăn nhớ 00 đến 1FH được dành cho các thanh ghi và ngăn xếp.
2. Tổng cộng 16 byte từ ngăn nhớ 20H đến 2FH được dành cho bộ nhớ đọc/ ghi đánh địa chỉ được theo bit. Chương 8 sẽ bàn chi tiết về bộ nhớ và các lệnh đánh địa chỉ được theo bit.
3. Tổng cộng 80 byte từ ngăn nhớ 30H đến 7FH được dùng cho lưu đọc và ghi hay như vẫn thường gọi là bảng nháp (Serach pad). Những ngăn nhớ này (80 byte) của RAM được sử dụng rộng rãi cho mục đích lưu dữ liệu và tham số bởi các lập trình viên 8051. Chúng ta sẽ sử dụng chúng ở các chương sau để lưu dữ liệu nhận vào CPU qua các cổng vào-ra.

2.7.2 Các băng thanh ghi trong 8051.

Như đã nói ở trước, tổng cộng 32 byte RAM được dành riêng cho các băng thanh ghi và ngăn xếp. 32 byte này được chia ra thành 4 băng các thanh ghi trong đó mỗi băng có 8 thanh ghi từ R0 đến R7. Các ngăn nhớ RAM số 0, R1 là ngăn nhớ RAM số 1, R2 là ngăn nhớ RAM số 2 v.v... Băng thứ hai của các thanh ghi R0 đến R7 bắt đầu từ thanh nhớ RAM số 2 cho đến ngăn nhớ RAM số 0FH. Băng thứ ba bắt đầu từ ngăn nhớ 10H đến 17H và cuối cùng từ ngăn nhớ 18H đến 1FH là dùng cho băng các thanh ghi R0 đến R7 thứ tư.



Hình 2.5: Ngăn xếp các thanh nhớ RAM trong 8051.

Bank 0		Bank 1		Bank 2		Bank 3	
7	R7	F	R7	17	R7	1F	R7
6	R6	E	R6	16	R6	1E	R6
5	R5	D	R5	15	R5	1D	R5
4	R4	C	R4	14	R4	1C	R4
3	R3	B	R3	13	R3	1B	R3
2	R2	A	R2	12	R2	1A	R2
1	R1	9	R1	11	R1	19	R1
0	R0	8	R0	10	R0	18	R0

Hình 2.6: Các băng thanh ghi của 8051 và địa chỉ của chúng.

Như ta có thể nhìn thấy từ hình 2.5 băng 1 sử dụng cùng không gian RAM như ngăn xếp. Đây là một vấn đề chính trong lập trình 8051. Chúng ta phải hoặc là không sử dụng băng 1 hoặc là phải đánh một không gian khác của RAM cho ngăn xếp.

VÝ dô 2.5:

Hãy phát biểu các nội dung của các ngăn nhớ RAM sau đoạn chương trình sau:

```
MOV   R0, #99H          ; Nạp R0 giá trị 99H
MOV   R1, #85H          ; Nạp R1 giá trị 85H
MOV   R2, #3FH          ; Nạp R2 giá trị 3FH
MOV   R7, #63H          ; Nạp R7giá trị 63H
MOV   R5, #12H          ; Nạp R5 giá trị12H
```

Lêi gi¶i:

Sau khi thực hiện chương trình trên ta có:

Ngăn nhớ 0 của RAM có giá trị 99H

Ngăn nhớ 1 của RAM có giá trị 85H

Ngăn nhớ 2 của RAM có giá trị 3FH

Ngăn nhớ 7 của RAM có giá trị 63H

Ngăn nhớ 5 của RAM có giá trị 12H

2.6.3 Băng thanh ghi mặc định.

Nếu các ngăn nhớ 00 đến 1F được dành riêng cho bốn băng thanh ghi, vậy băng thanh ghi R0 đến R7 nào ta phải truy cập tới khi 8051 được cấp nguồn? Câu trả lời là các băng thanh ghi 0. Đó là các ngăn nhớ RAM số 0, 1, 2, 3, 4, 5, 6 và 7 được truy cập với tên R0, R1, R2, R3, R4, R5, R6 và R7 khi lập trình 8051. Nó dễ dàng hơn nhiều khi tham chiếu các ngăn nhớ RAM này ơví các tên R0, R1 v.v... hơn là số vị trí của các ngăn nhớ. Ví dụ 2.6 làm rõ khái niệm này.

VÝ dô 2.6:

Hãy viết lại chương trình ở ví dụ 2.5 sử dụng các địa chỉ RAM thay tên các thanh ghi.

Lêi gi¶i:

Đây được gọi là chế độ đánh địa chỉ trực tiếp và sử dụng địa chỉ các vị trí ngăn nhớ RAM đối với địa chỉ đích. Xem chi tiết ở chương 5 về chế độ đánh địa chỉ.

```
MOV   00, #99H          ; Nạp thanh ghi R0 giá trị 99H
MOV   01, #85H          ; Nạp thanh ghi R1 giá trị 85H
MOV   02, #3FH          ; Nạp thanh ghi R2 giá trị 3FH
MOV   07, #63H          ; Nạp thanh ghi R7giá trị 63H
MOV   05, #12H          ; Nạp thanh ghi R5 giá trị12H
```

2.6.4 Chuyển mạch các băng thanh ghi như thế nào?

Như đã nói ở trên, băng thanh ghi 0 là mặc định khi 8051 được cấp nguồn. Chúng ta có thể chuyển mạch sang các băng thanh ghi khác bằng cách sử dụng bit D3 và D4 của thanh ghi PSW như chỉ ra theo bảng 2.2.

Bảng 2.2: Bit lựa chọn các băng thanh ghi RS0 và RS1.

	RS1 (PSW.4)	RS0 (PSW.3)
Băng 0	0	0
Băng 1	0	1
Băng 2	1	0
Băng 3	1	1

Bit D3 và D4 của thanh ghi PSW thường được tham chiếu như là PSW.3 và PSW.4 vì chúng có thể được truy cập bằng các lệnh đánh địa chỉ theo bit như SETB và CLR. Ví dụ "SETB PSW.3" sẽ thiết lập PSW.3 và chọn băng thanh ghi 1. Xem ví dụ 2.7 dưới đây.

VÝ dô 2.7:

Hãy phát biểu nội dung các ngăn nhớ RAM sau đoạn chương trình dưới đây:

```
SETB    PSW.4           ; Chọn băng thanh ghi 4
MOV     R0, #99H        ; Nạp thanh ghi R0 giá trị 99H
MOV     R1, #85H        ; Nạp thanh ghi R1 giá trị 85H
MOV     R2, #3FH        ; Nạp thanh ghi R2 giá trị 3FH
MOV     R7, #63H        ; Nạp thanh ghi R7giá trị 63H
MOV     R5, #12H        ; Nạp thanh ghi R5 giá trị12H
```

Lêi gi¶i:

Theo mặc định PSW.3 = 0 và PSW.4 = 0. Do vậy, lệnh "SETB PSW.4" sẽ bật bit RS1 = 1 và RS0 = 0, bằng lệnh như vậy băng thanh ghi R0 đến R7 số 2 được chọn. Băng 2 sử dụng các ngăn nhớ từ 10H đến 17H. Nên sau khi thực hiện đoạn chương trình trên ta có nội dung các ngăn nhớ như sau:

Ngăn nhớ vị trí 10H có giá trị 99H
Ngăn nhớ vị trí 11H có giá trị 85H
Ngăn nhớ vị trí 12H có giá trị 3FH
Ngăn nhớ vị trí 17H có giá trị 63H
Ngăn nhớ vị trí 15H có giá trị 12H

2.6.5 Ngăn xếp trong 8051.

Ngăn xếp là một vùng bộ nhớ RAM được CPU sử dụng để lưu thông tin tạm thời. Thông tin này có thể là dự liệu, có thể là địa cỉ CPU cần không gian lưu trữ này vì số các thanh ghi bị hạn chế.

2.6.6 Cách truy cập các ngăn xếp trong 8051.

Nếu ngăn xếp là một vùng của bộ nhớ RAM thì phải có các thanh ghi trong CPU chỉ đến nó. Thanh được dùng để chỉ đến ngăn xếp được gọi là thanh ghi con trỏ ngăn xếp SP (Stack Pointer). Con trỏ ngăn xếp trong 8051 chỉ rộng 8 bit có nghĩa là nó chỉ có thể có thể được các địa chỉ từ 00 đến FFH.

Khi 8051 được cấp nguồn thì SP chứa giá trị 07 có nghĩa là ngăn nhớ 08 của RAM là ngăn nhớ đầu tiên được dùng cho ngăn xếp trong 8051. Việc lưu lại một thanh ghi

PCU trong ngăn xếp được gọi là một lần cất vào PUSH và việc nạp nội dung của ngăn xếp trở lại thanh ghi CPU được gọi là lấy ra POP. Hay nói cách khác là một thanh ghi được cất vào ngăn xếp để lưu cất và được lấy ra từ ngăn xếp để dùng tiếp công việc của SP là rất nghiêm ngặt mỗi khi thao tác cất vào (PUSH) và lấy ra (POP) được thực thi. Để biết ngăn xếp làm việc như thế nào hãy xét các lệnh PUSH và POP dưới đây.

2.6.7 Cất thanh ghi vào ngăn xếp.

Trong 8051 thì con trỏ ngăn xếp chỉ đến ngăn nhớ sử dụng cuối cùng của ngăn xếp. Khi ta cất dữ liệu vào ngăn xếp thì con trỏ ngăn xếp SP được tăng lên 1. Lưu ý rằng điều này đối với các bộ vi xử lý khác nhau là khác nhau, đáng chú ý là các bộ vi xử lý × 86 là SP giảm xuống khi cất dữ liệu vào ngăn xếp. Xét ví dụ 2.8 dưới đây, ta thấy rằng mỗi khi lệnh PUSH được thực hiện thì nội dung của thanh ghi được cất vào ngăn xếp và SP được tăng lên 1. Lưu ý là đối với mỗi byte của dữ liệu được cất vào ngăn xếp thì SP được tăng lên 1 lần. Cũng lưu ý rằng để cất các thanh ghi vào ngăn xếp ta phải sử dụng địa chỉ RAM của chúng. Ví dụ lệnh "PUSH 1" là cất thanh ghi R1 vào ngăn xếp.

VÝ dô 2.8:

Hãy biểu diễn ngăn xếp và con trỏ ngăn xếp đối với đoạn chương trình sau đây. Giả thiết vùng ngăn xếp là mặc định.

```
MOV     R6, #25H
MOV     R1, #12H
MOV     R4, #0F3H
PUSH    6
PUSH    1
PUSH    4
```

Lêi gi¶i:

	Sau PUSH 6	Sau PUSP 1	Sau PUSH 4
0B	0B	0B	0B
0A	0A	0A	0A F3
09	09	09 12	09 12
08	08 25	08 25	08 25
Bắt đầu SP = 07	SP = 08	SP = 09	SP = 0A

2.6.8 Lấy nôi dung thanh ghi ra từ ngăn xếp.

Việc lấy nội dung ra từ ngăn xếp trở lai thanh ghi đã cho là qúa trình ngược với các nội dung thanh ghi vào ngăn xếp. Với mỗi lần lấy ra thì byte trên đỉnh ngăn xếp được sao chép vào thanh ghi được xác định bởi lệnh và con trỏ ngăn xếp được giảm xuống 1. Ví dụ 2.9 minh hoạ lệnh lấy nội dung ra khỏi ngăn xếp.

VÝ dô 2.9:

Khảo sát ngăn xếp và hãy trình bày nội dung của các thanh ghi và SP sau khi thực hiện đoạn chương trình sau đây:

```
POP     3       ; Lấy ngăn xếp trở lại R3
POP     5       ; Lấy ngăn xếp trở lại R5
```

```
POP     2       ; Lấy ngăn xếp trở lại R2
```

Lêi gi¶i:

		Sau POP3		Sau POP 5		Sau POP 2	
0B	54	0B		0B		0B	
0A	F9	0A	F9	0A		0A	
09	76	09	76	09	76	09	
08	6C	08	6C	08	6C	08	6C
Bắt đầu SP = 0B		SP = 0A		SP = 09		SP = 08	

2.6.9 Giới hạn trên của ngăn xếp.

Như đã nói ở trên, các ngăn nhớ 08 đến 1FH của RAM trong 8051 có thể được dùng làm ngăn nhớ 20H đến 2FH của RAM được dự phòng cho bộ nhớ đánh địa chỉ được theo bit và không thể dùng trước cho ngăn xếp. Nếu trong một chương trình đã cho ta cần ngăn xếp nhiều hơn 24 byte (08 đến 1FH = 24 byte) thì ta có thể đổi SP chỉ đến các ngăn nhớ 30 đến 7FH. Điều này được thực hiển bởi lệnh "MOV SP, #XX".

2.6.10 Lệnh gọi CALL và ngăn xếp.

Ngoài việc sử dụng ngăn xếp để lưu cất các thanh ghi thì CPU cũng sử dụng ngăn xếp để lưu cất tam thời địa chỉ của lệnh đứng ngay dưới lệnh CALL. Điều này chính là để PCU biết chỗ nào để quay trở về thực hiện tiếp các lệnh sau khi chọn chương trình con. Chi tiết về lệnh gọi CALL được trình bỳ ở chương 3.

2.6.11 Xung đột ngăn xếp và băng thanh ghi số 1.

Như ta đa nói ở trên thì thanh ghi con trỏ ngăn xếp có thể chỉ đến vị trí RAM hiện thời dành cho ngăn xếp. Khi dữ liệu được lưu cất cào ngăn xếp thì SP được tăng lên và ngược lại khi dữ liệu được lấy ra từ ngăn xếp thì SP giảm xuống. Lý do là PS được tăng lên sau khi PUSH là phải biết lấy chắc chắn rằng ngăn xếp đang tăng lên đến vị trí ngăn nhớ 7FH của RAM từ địa chỉ thấp nhất đến địa chỉ cao nhất. Nếu con trỏ ngăn xếp đã được giảm sau các lệnh PUSH thì ta nên sử dụng các ngăn nhớ 7, 6, 5 v.v... của RAM thuộc các thanh ghi R7 đến R0 của băng 0, băng thanh ghi mặc định. Việc tăng này của con trỏ ngăn xếp đối với các lệnh PUSH cũng đảm bảo rằng ngăn xếp sẽ không với tới ngăn nhớ 0 của RAM (đáy của RAM) và do vậy sẽ nhảy ra khỏi không gian dành cho ngăn xếp. Tuy nhiên có vấn đề nảy sinh với thiết lập mặc định của ngăn xếp. Ví dụ SP = 07 khi 8051 được bật nguồn nên RAM và cũng thuộc về thanh ghi R0 củ băng thanh ghi số 1. Hay nói cách khác băng thanh ghi số 1 và ngăn xếp đang dùng chung một không gian của bộ nhớ RAM. Nếu chương trình đã cho cần sử dụng các băng thanh ghi số 1 và số 2 ta có thể đặt lại vùng nhớ RAM cho ngăn xếp. Ví dụ, ta có thể cấp vị trí ngăn nhớ 60H của RAM và cao hơn cho ngăn xếp trong ví dụ 2.10.

VÝ dô 2.10:

Biểu diễn ngăn xếp và con trỏ ngăn xếp đối với các lệnh sau:

```
MOV     SP, #5FH            ; Đặt ngăn nhớ từ 60H của RAM cho ngăn xếp
MOV     R2, #25H
MOV     R1, #12H
MOV     R4, #0F3H
PUSH    2
```

PUSH 1
PUSH 4

Lêi gi¶i:

	Sau PUSH 2	Sau PUSP 3	Sau PUSH 4
63	63	63	63
62	62	62	62 F3
61	61	61 12	61 12
60	60 25	60 25	60 25
Bắt đầuSP=5F	SP = 60	SP = 61	SP = 62

CHƯƠNG 3
Các lệnh nhảy, vòng lặp và lệnh gọi

Trong một chuỗi lệnh cần thực hiện thường có nhu cần cần chuyển điều khiển chương trình đến một vị trí khác. Có nhiều lệnh để thực hiện điều này trong 8051, ở chương này ta sẽ tìm hiểu các lệnh chuyển điều khiển có trong hợp ngữ của 8051 như các lệnh sử dụng cho vòng lặp, các lệnh nhảy có và không có điều khiển, lệnh gọi và cuối cùng là mô tả về một chương trình con giữ chậm thời gian.

3.1 Vòng lặp và các lệnh nhảy.

3.1.1 Tạo vòng lặp trong 8051.

Qúa trình lặp lại một chuỗi các lệnh với một số lần nhất định được gọi là vòng lặp. Vòng lặp là một trong những hoạt động được sử dụng rộng rãi nhất mà bất kỳ bộ vi sử lý nào đều thực hiện. Trong 8051 thì hoạt động vòng lặp được thực hiện bởi lệnh "DJNZ thanh ghi, nhãn". Trong lệnh này thanh ghi được giảm xuống, nếu nó không bằng không thì nó nhảy đến địa chỉ đích được tham chiếu bởi nhãn. Trước khi bắt đầu vòng lặp thì thanh ghi được nạp với bộ đếm cho số lần lặp lại. Lưu ý rằng, trong lệnh này việc giảm thanh ghi và quyết định để nhảy được kết hợp vào trong một lệnh đơn.

Ví dụ 3.1:

Viết một chương trình để: a) xoá ACC và sau đó b) cộng 3 vào ACC 10 lần.

Lời giải:

```
        MOV     A, #0       ; Xoá ACC,  A = 0
        MOV     R2, #10     ; Nạp bộ đếm R2 = 10
BACK:   ADD     A, #10      ; Cộng 03 vào ACC
        DJNZ    R2, AGAIN   ; Lặp lại cho đến khi R2 = 0 (10 lần)
        MOV     R5, A       ; Cắt A vào thanh ghi R5
```

Trong chương trình trên đây thanh ghi R2 được sử dụng như là bộ đếm. Bộ đếm lúc đầu được đặt bằng 10. Mỗi lần lặp lại lệnh DJNZ giảm R2 không bằng 0 thì nó nhảy đến địa chỉ đích gắn với nhãn "AGAIN". Hoạt động lặp lại này tiếp tục cho đến khi R2 trở về không. Sau khi R2 = 0 nó thoát khỏi vòng lặp và thực hiện đứng ngay dưới nó trong trường hợp này là lệnh "MOV R5, A".

Lưu ý rằng trong lệnh DJNZ thì các thanh ghi có thể là bất kỳ thanh ghi nào trong các thanh ghi R0 - R7. Bộ đếm cũng có thể là một ngăn nhớ trong RAM như ta sẽ thấy ở chương 5.

Ví dụ 3.2:

Số lần cực đại mà vòng lặp ở ví dụ 3.1 có thể lặp lại là bao nhiêu?

Lời giải:

Vì thanh ghi R2 chứa số đếm và nó là thanh ghi 8 bit nên nó có thể chứa được giá trị cực đại là FFH hay 155. Do vậy số lần lặp lại cực đại mà vòng lặp ở ví dụ 3.1 có thể thực hiện là 256.

3.2.1 Vòng lặp bền trong một vòng lặp.

Như trình bày ở ví dụ 3.2 số đếm cực đại là 256. Vậy điều gì xảy ra nếu ta muốn lặp một hành động nhiều hơn 256 lần? Để làm điều đó thì ta sử dụng một vòng

lặp bên trong một vòng lặp được gọi là vòng lặp lồng (Nested Loop). Trong một vòng lặp lồng ta sử dụng 2 thanh ghi để giữ số đếm. Xét ví dụ 3.3 dưới đây.

Ví dụ 3.3:

Hãy viết một chương trình a) nạp thanh ghi ACC với giá trị 55H và b) bù ACC 700 lần.

Lời giải:

Vì 700 lớn hơn 256 (là số cực đại mà một thanh ghi vó thể chứa được) nên ta phải dùng hai thanh ghi để chứa số đếm. Đoạn mã dưới đây trình bày cách sử dụng hai thanh ghi R2 và R3 để chứa số đếm.

```
        MOV     A, #55H     ; Nạp A = 55H
        MOV     R3, #10     ; Nạp R3 = 10 số đếm vòng lặp ngoài
NEXT:   MOV     R2, #70     ; Nạp R2 = 70 số đếm vòng lặp trong
AGAIN: `CPL     A           ; Bù thanh ghi A
        DJNZ    R2, AGAIN   ; Lặp lại 70 lần (vòng lặp trong)
        DJNZ    R3, NEXT
```

Trong chương trình này thanh ghi R2 được dùng để chứa số đếm vòng lặp trong. Trong lệnh "DJNZ R2, AGAIN" thì mỗi khi R2 = 0 nó đi thẳng xuống và lệnh "JNZ R3, NEXT" được thực hiện. Lệnh này ép CPU nạp R2 với số đếm 70 và vòng lặp trong khi bắt đầu lại quá trình này tiếp tục cho đến khi R3 trở về không và vòng lặp ngoài kết thúc.

3.1.3 Các lệnh nhảy có điều kiện.

Các lệnh nhảy có điều kiện đối với 8051 được tổng hợp trong bảng 3.1. Các chi tiết về mỗi lệnh được cho trong phụ lục AppendixA. Trong bảng 3.1 lưu ý rằng một số lệnh như JZ (nhảy nếu A = 0) và JC (nhảy nếu có nhớ) chỉ nhảy nếu một điều kiện nhất định được thoả mãn. Kế tiếp ta xét một số lệnh nhảy có điều kiện với các **Ví dụ minh hoạ sau.**

a- Lệnh JZ (nhảy nếu A = 0). Trong lệnh này nội dung của thanh ghi A được kiểm tra. Nếu nó bằng không thì nó nhảy đến địa chỉ đích. Ví dụ xét đoạn mã sau:

```
MOV     A, R0       ; Nạp giá trị của R0 vào A
JZ      OVER        ; Nhảy đến OVER nếu A = 0
MOV     A, R1       ; Nạp giá trị của R1 vào A
JZ      OVER        ; Nhảy đến OVER nếu A = 0
OVER ...
```

Trong chương trình này nếu R0 hoặc R1 có giá trị bằng 0 thì nó nhảy đến địa chỉ có nhãn OVER. Lưu ý rằng lệnh JZ chỉ có thể được sử dụng đối với thanh ghi A. Nó chỉ có thể kiểm tra xem thanh ghi A có bằng không không và nó không áp dụng cho bất kỳ thanh ghi nào khác. Quan trọng hơn là ta không phải thực hiện một lệnh số học nào như đếm giảm để sử dụng lệnh JNZ như ở ví dụ 3.4 dưới đây.

Ví dụ 3.4:

Viết một chương trình để xác định xem R5 có chứa giá trị 0 không? Nếu nạp thì nó cho giá trị 55H.

Lời giải:

```
        MOV     A, R5       ; Sao nội dung R5 vào A
        JNZ     NEXT        ; Nhảy đến NEXT nếu A không bằng 0
        MOV     R5, #55H    ;
NEXT:   ...
```

b- Lệnh JNC (nhảy nếu không có nhớ, cờ CY = 0).

Trong lệnh này thì bit cờ nhớ trong thanh ghi cờ PSW được dùng để thực hiện quyết định nhảy. Khi thực hiện lệnh "JNC nhãn" thì bộ xử lý kiểm tra cờ nhớ xem nó có được bật không (CY = 1). Nếu nó không bật thì CPU bắt đầu nạp và thực hiện các lệnh từ địa chỉ của nhãn. Nếu cờ CY = 1 thì nó sẽ không nhảy và thực hiện lệnh kế tiếp dưới JNC.

Cần phải lưu ý rằng cũng có lệnh "JC nhãn". Trong lệnh JC thì nếu CY = 1 nó nhảy đến địa chỉ đích là nhãn. Ta sẽ xét các ví dụ về các lệnh này trong các ứng dụng ở các chương sau.

Ngoài ra còn có lệnh JB (nhảy nếu bit có mức cao) và JNB (nhảy nếu bit có mức thấp). Các lệnh này được trình bày ở chương 4 và 8 khi nói về thao tác bit.

Bảng 3.1: Các lệnh nhảy có điều kiện.

Lệnh	Hoạt động
JZ	Nhảy nếu A = 0
JNZ	Nhảy nếu A □ 0
DJNZ	Giảm và nhảy nếu A = 0
CJNE A, byte	Nhảy nếu A □ byte
CJNE re, # data	Nhảy nếu Byte □ data
JC	Nhảy nếu CY = 1
JNC	Nhảy nếu CY = 0
JB	Nhảy nếu bit = 1
JNB	Nhảy nếu bit = 0
JBC	Nhảy nếu bit = 1 và xoá nó

Ví dụ 3.5:

Hãy tìm tổng của các giá trị 79H, F5H và E2H. Đặt vào trong các thanh ghi R0 (byte thấp) và R5 (byte cao).

Lời giải:

```
        MOV     A, #0       ; Xoá thanh ghi A = 0
        MOV     R5, A       ; Xoá R5
        ADD     A #79H      ; Cộng 79H vào A (A = 0 + 79H = 79H)
        JNC     N-1         ; Nếu không có nhớ cộng kế tiếp
        INC     R5          ; Nếu CY = 1, tăng R5

N-1:    ADD     A, #0F5H    ; Cộng F5H vào A (A = 79H + F5H = 6EH) và CY = 1
        JNC     N-2         ; Nhảy nếu CY = 0
        INC     R5          ; Nếu CY = 1 tăng R5 (R5 = 1)
N-2:    ADD     A, #0E2H    ; Cộng E2H vào A (A = GE + E2 = 50) và CY = 1
        JNC     OVER        ; Nhảy nếu CY = 0
        INC     R5          ; Nếu CY = 1 tăng R5
OVER:   MOV     R0, A       ; Bây giờ R0 = 50H và R5 = 02
```

c- Tất cả các lệnh nhảy có điều kiện đều là những phép nhảy ngắn.

Cần phải lưu ý rằng tất cả các lệnh nhảy có điều kiện đều là các phép nhảy ngắn, có nghĩa là địa chỉ của đích đều phải nằm trong khoảng -127 đến +127 byte của nội dung bộ đếm chương trình PC.

3.1.4 Các lệnh nhảy không điều kiện.

Lệnh nhảy không điều kiện là một phép nhảy trong đó điều khiển được truyền không điều kiện đến địa chỉ đích. Trong 8051 có hai lệnh nhảy không điều kiện đó là: LJMP - nhảy xa và SJMP - nhảy gần.

a- Nhảy xa LJMP:

Nhảy xa LJMP là một lệnh 3 byte trong đó byte đầu tiên là mã lệnh còn hai byte còn lại là địa chỉ 16 bit của đích. Địa chỉ đích 02 byte có phép một phép nhảy đến bất kỳ vị trí nhớ nào trong khoảng 0000 - FFFFH.

Hãy nhớ rằng, mặc dù bộ đếm chương trình trong 8051 là 16 bit, do vậy cho không gian địa chỉ là 64k byte, nhưng bộ nhớ chương trình ROM trên chíp lớn như vậy. 8051 đầu tiên chỉ có 4k byte ROM trên chíp cho không gian chương trình, do vậy mỗi byte đều rất quý giá. Vì lý do đó mà có cả lệnh nhảy gần SJMP chỉ có 2 byte so với lệnh nhảy xa LZ0MP dài 3 byte. Điều này có thể tiết kiệm được một số byte bộ nhớ trong rất nhiều ứng dụng mà không gian bộ nhớ có hạn hẹp.

b- Lệnh nhảy gồm SJMP.

Trong 2 byte này thì byte đầu tiên là mã lệnh và byte thứ hai là chỉ tương đối của địa chỉ đích. Đích chỉ tương đối trong phạm vi 00 - FFH được chia thành các lệnh nhảy tới và nhảy lùi: Nghĩa là -128 đến +127 byte của bộ nhớ tương đối so với địa chỉ hiện thời của bộ đếm chương trình. Nếu là lệnh nhảy tới thì địa chỉ đích có thể nằm trong khoảng 127 byte từ giá trị hiện thời của bộ đếm chương trình. Nếu địa chỉ đích ở phía sau thì nó có thể nằm trong khoảng -128 byte từ giá trị hiện hành của PC.

3.1.5 Tính toán địa chỉ lệnh nhảy gần.

Ngoài lệnh nhảy gần SJMP thì tất cả mọi lệnh nhảy có điều kiện như JNC, JZ và DJNZ đều là các lệnh nhảy gần bởi một thực tế là chúng đều lệnh 2 byte. Trong những lệnh này thì byte thứ nhất đều là mã lệnh, còn byte thứ hai là địa chỉ tương đối. Địa chỉ đích là tương đối so với giá trị của bộ đếm chương trình. Để tính toán địa chỉ đích byte thứ hai được cộng vào thanh ghi PC của lệnh đứng ngay sau lệnh nhảy. Để hiểu điều này hãy xét ví dụ 3.6 dưới đây.

Ví dụ 3.6:

Sử dụng tệp tin liệt kê dưới đây hãy kiểm tra việc tín toán địa chỉ nhảy về trước.

```
01      0000                            ORG         0000
02      0000    7800                    MOV         R0, #0
03      0002    7455                    MOV         A, #55H
04      0004    6003                    JZ          NEXT
05      0006    08                      NIC         R0
06      0007    04          AGAIN: INC          A
07      0008    04                      INC         A
08      0009    2477        NEXT:       ADD         A, #77h
09      000B    5005                    JNC         OVER
10      000D    E4                      CLR         A
```

```
11      000E        F8                      MOV     R0, A
12      000F        F9                      MOV     R1, A
13      0010        FA                      MOV     R2, A
14      0011        FB                      MOV     R3, A
15      0012        2B          OVER:       ADD     A, R3
16      0013        50F2                    JNC     AGAIN
17      0015        80FE        HERE:       SJMP    SHERE
18      0017                                END
```

Lời giải:

Trước hết lưu ý rằng các lệnh JZ và JNC đều là lệnh nhảy về trước. Địa chỉ đích đối với lệnh nhảy về trước được tính toán bằng cách cộng giá trị PC của lệnh đi ngay sau đó vào byte thứ hai của lệnh nhảy gần được gọi là địa chỉ tương đối. Ở dòng 04 lệnh "JZ NEXT" có mã lệnh 60 và toán hạng 03 tại địa chỉ 0004 và 0005. Ở đây 03 là địa chỉ tương đối, tương đối so với địa chỉ của lệnh kế tiếp là: "INC R0" và đó là 0006. Bằng việc cộng 0006 vào 3 thì địa chỉ đích của nhãn NEXT là 0009 được tạo ra. Bằng cách tương tự như vậy đối với dòng 9 thì lệnh "JNC OVER" có mã lệnh và toán hạng là 50 và 05 trong đó 50 là mã lệnh và 05 là địa chỉ tương đối. Do vậy, 05 được cộng vào OD là địa chỉ của lệnh "CLA A" đứng ngay sau lệnh "JNC OVER" và cho giá trị 12H chính là địa chỉ của nhãn OVER.

Ví dụ 3.7:

Hãy kiểm tra tính toán địa chỉ của các lệnh nhảy lùi trong ví dụ 3.6.

Lời giải:

Trong danh sách liệt kê chương trình đó thì lệnh "JNC AGAIN" có mã lệnh là 50 và địa chỉ tương đối là F2H. Khi địa chỉ tương đối của F2H được cộng vào 15H là địa chỉ của lệnh đứng dưới lệnh nhảy ta có 15H + F2H = 07 (và phần nhớ được bỏ đi). Để ý rằng 07 là địa chỉ nhãn AGAIN. Và hãy cũng xét lệnh "SJMP HERE" có mã lệnh 80 và địa chỉ tương đối FE giá trị PC của lệnh kế tiếp là 0017H được cộng vào địa chỉ tương đối FEH ta nhận được 0015H chính là địa chỉ nhãn HERE (17H + FEH = 15H) phần nhớ được bỏ đi). Lưu ý rằng FEH là -2 và 17h + (-2) = 15H. Về phép cộng số âm sẽ được bàn ở chương 6.

3.1.6 Tính toán địa chỉ đích nhảy lùi.

Trong khi ở trường hợp nhảy tới thì giá trị thay thế là một số dương trong khoảng từ 0 đến 127 (00 đến 7F ở dạng Hex) thì đối với lệnh nhảy lùi giá trị thay thế là một số âm nằm trong khoảng từ 0 đến -128 như được giải thích ở ví dụ 3.7.

Cần phải nhấn mạnh rằng, bất luận SJMP nhảy tới hay nhảy lùi thì đối với một lệnh nhảy bất kỳ địa chỉ của địa chỉ đích không bao giờ có thể lớn hơn 0 -128 đến +127 byte so với địa chỉ gắn liền với lệnh đứng ngay sau lệnh SJMP. Nếu có một sự nỗ lực nào vi phạm luật này thì hợp ngữ sẽ tạo ra một lỗi báo rằng lệnh nhảy ngoài phạm vi.

3.2 Các lệnh gọi CALL.

Một lệnh chuyển điều khiển khác là lệnh CALL được dùng để gọi một chương trình con. Các chương trình con thường được sử dụng để thực thi các công việc cần phải được thực hiện thường xuyên. Điều này làm cho chương trình trở nên có cấu trúc hơn ngoài việc tiết kiệm được thêm không gian bộ nhớ. Trong 8051 có 2

lệnh để gọi đó là: Gọi xa CALL và gọi tuyệt đối ACALL mà quyết định sử dụng lệnh nào đó phụ thuộc vào địa chỉ đích.

3.2.1 Lệnh gọi xa LCALL.

Trong lệnh 3 byte này thì byte đầu tiên là mã lệnh, còn hai byte sau được dùng cho địa chỉ của chương trình con đích. Do vậy LCALL có thể được dùng để gọi các chương trình con ở bất kỳ vị trí nào trong phạm vi 64k byte, không gian địa chỉ của 8051. Để đảm bảo rằng sau khi thực hiện một chương trình được gọi để 8051 biết được chỗ quay trở về thì nó tự động cất vào ngăn xếp địa chỉ của lệnh đứng ngay sau lệnh gọi LCALL. Khi một chương trình con được gọi, điều khiển được chuyển đến chương trình con đó và bộ xử lý cất bộ đếm chương trình PC vào ngăn xếp và bắt đầu nạp lệnh vào vị trí mới. Sau khi kết thúc thực hiện chương trình con thì lệnh trở về RET chuyển điều khiển về cho nguồn gọi. Mỗi chương trình con cần lệnh RET như là lệnh cuối cùng (xem ví dụ 3.8).

Các điểm sau đây cần phải được lưu ý từ ví dụ 3.8.

1. Lưu ý đến chương trình con DELAY khi thực hiện lệnh "LCALL DELAY" đầu tiên thì địa chỉ của lệnh ngay kế nó là "MOV A, #0AAH" được đẩy vào ngăn xếp và 8051 bắt đầu thực hiện các lệnh ở địa chỉ 300H.
2. Trong chương trình con DELAY, lúc đầu bộ đếm R5 được đặt về giá trị 255 (R5 = FFH). Do vậy, vòng lặp được lặp lại 256 lần. Khi R5 trở về 0 điều khiển rơi xuống lệnh quay trở về RET mà nó kéo địa chỉ từ ngăn xếp vào bộ đếm chương trình và tiếp tục thực hiện lệnh sau lệnh gọi CALL.

Ví dụ 3.8:

Hãy viết một chương trình để chốt tất cả các bit của cổng P1 bằng cách gửi đến nó giá trị 55H và AAH liên tục. Hãy đặt một độ trễ thời gian giữa mỗi lần xuất dữ liệu tới cổng P1. Chương trình này sẽ được sử dụng để kiểm tra các cổng của 8051 trong chương tiếp theo.

Lời giải:

```
                ORG     0000
BACK:           MOV     A, #55H         ; Nạp A với giá trị 55H
                MOV     P1, A           ; Gửi 55H đến cổng P1
                LCALL   DELAY           ; Tạo trễ thời gian
                MOV     A, #0AAH        ; Nạp A với giá trị  AAH
                MOV     P1, A           ; Gửi AAH đến cổng P1
                LCALL   DELAY           ; Giữ chậm
                SJMP    BACK            ; Lặp lại vô tận
; ------------------ - Đây là chương trình con tạo độ trễ thời gian
                ORG     300H            ; Đặt chương trình con trễ thời gian ở địa chỉ 300H
DELAY:          MOV     R5, #00H        ; Nạp bộ đếm R5 = 255H (hay FFH)
AGAIN:          DJNZ    R5, AGAIN       ; Tiếp tục cho đến khi R5 về không
                RET                     ; Trả điều khiển về nguồn gọi (khi R5 = 0)
                END                     ; Kết thúc tệp tin của hợp ngữ
```

Lượng thời gian trễ trong ví dụ 8.3 phục thuộc vào tần số của 8051. Cách tính chính xác thời gian sẽ được giải thích ở chương 4. Tuy nhiên ta có thể tăng thời gian độ trễ bằng cách sử dụng vòng lặp lồng như chỉ ra dưới đây.

```
DELAY:                                  ; Vòng lặp lồng  giữ chậm
                MOV     R4, #255        ; Nạp R4 = 255 (FFH dạng hex)
```

```
NEXT:   MOV   R5, #255     ; Nạp R5  = 255 (FFH dạng hex)
AGAIN:  DJNZ  R5, AGAIN    ; Lặp lại cho đến khi RT = 0
        DJNZ  R4, NEXT     ; Giảm R4
                           ;Tiếp tục nạp R5 cho đến khi R4 = 0
        RET                ; Trở về (khi R4 =  0)
```

3.2.2 Lệnh gọi CALL và vai trò của ngăn xếp.

Ngăn xếp và con trỏ ngăn xếp ta sẽ nghiên cứu ở chương cuối. Để hiểu được tầm quan trọng của ngăn xếp trong các bộ vi điều khiển bây giờ khảo sát nội dung của ngăn xếp và con trỏ ngăn xếp đối với ví dụ 8.3. Điều này được trình bày ở ví dụ 3.9 dưới đây.

Ví dụ 3.9:

Hãy phân tích nội dung của ngăn xếp sau khi thực hiện lệnh LCALL đầu tiên dưới đây.

```
001   0000                    OR6
002   0000  7455      BACK:   MOV   A, #55H      ; Nạp A với giá trị 55H
003   0002  F590              MOV   P1, A        ; Gửi 55H tới cổng P1
004   0004  120300            LCALL DELAY        ; Tạo trễ thời gian
005   0007  74AA              MOV   A, #0AAH     ; Nạp A với giá trị AAH
006   0009  F590              MOV   P1, A        ; Gửi AAH tới cổng      P1
007   000B  120300            LCALL DELAY        ; Tạo trễ thời gian
008   000E  80F0              SJMP  BACK         ; Tiếp tục thực hiện
009   0010
010   0010                    ; ..... Đây là chương trình con giữ chậm
011   0300                    MOV   300H
012   0300            DELAY:
013   0300  7DFF              MOV   R5, #FFH     ; Nạp R5 = 255
014   0302  DDFE      AGAIN:  DJNZ  R5, AGAIN    ; Dừng ở đây
015   0304  22                RET                ; Trở về nguồn gọi
016   0305                    END                ; Kết thúc nạp tin hợp ngữ
```

Lời giải:

Khi lệnh LCALL đầu tiên được thực hiện thì địa chỉ của lệnh "MOV A, #0AAH" được cất vào ngăn xếp. Lưu ý rằng byte thấp vào trước và byte cao vào sau. Lệnh cuối cùng của chương trình con được gọi phải là lệnh trở về RET để chuyển CPU kéo (POP) các byte trên đỉnh của ngăn xếp vào bộ đếm chương trình PC và tiếp tục thực hiện lệnh tại địa chỉ 07. Sơ đồ bên chỉ ra khung của ngăn xếp sau lần gọi LCALL đầu tiên.

0A		
09		00
08		07
SP	=	09

3.2.3 Sử dụng lệnh PUSH và POP trong các chương trình con.

Khi gọi một chương trình con thì ngăn xếp phải bám được vị trí mà CPU cần trở về. Sau khi kết thúc chương trình con vì lý do này chúng ta phải cẩn thận mỗi khi thao tác với các nội dung của ngăn xếp. Nguyên tắc là số lần đẩy vào (PUSH) và kéo

ra (POP) luôn phải phù hợp trong bất kỳ chương trình con được gọi vào. Hay nói cách khác đối với mỗi lệnh PUSH thì phải có một lệnh POP. Xem ví dụ 3.10.

3.2.4 Gọi các chương trình con.

Trong lập trình hợp ngữ thường có một chương trình chính và rất nhiều chương trình con mà chúng được gọi từ chương trình chính. Điều này cho phép ta tạo mới chương trình con trong một mô-đun riêng biệt. Mỗi mô-đun có thể được kiểm tra tách biệt và sau đó được kết hợp với nhau cùng với chương trình chính. Quan trọng hơn là trong một chương trình lớn thì các mô-đun có thể được phân cho các lập trình viên khác nhau nhằm rút ngắn thời gian phát triển.

Ví dụ 3.10:

Phân tích ngăn xếp đối với lệnh LCALL đầu tiên trong đoạn mã.

```
01  0000                    ORG     0
02  0000  7455      BACK:   MOV     A, #55H     ; Nạp A với giá trị 55H
03  0002  F590              MOV     P1, A       ; Gửi 55H ra cổng P1
04  0004  7C99              MOV     R4, #99H
05  0006  7D67              MOV     R5, #67H
06  0008  120300            LCALL   DELAY       ; Tạo giữ chậm thời gian
07  000B  74AA              MOV     A, #0AAH    ; Nạp A với AAH
08  000D  F590              MOV     P1, A       ; Gửi AAH ra cổng P1
09  000F  120300            LCALL   DELAY
10  0012  80EC              SJMP    BACK        ; Tiếp tục thực hiện
11  0014                    ; ..... Đây là chương trình con DELAY
12  0300                    ORG     300H
13  0300  C004      DELAY   PUSH    4           ; Đẩy R4 vào ngăn xếp
14  0302  C005              PUSH    5           ; Đẩy R5 vào ngăn xếp
15  0304  7CFF              MOV     R4, 00FH    ; Gán R4 = FFH
16  0306  7DFF      NEXT:   MOV     R5,  #00FH  ; Gán R5 = 255
17  0308  DDFE      AGAIN:  DJNZ    R5, AGAIN
18  030A  DCFA              DJNZ    R4, NEXT
19  030C  D005              POP     5           ; Kéo đỉnh ngăn xếp vào R5
20  030E  D004              POP     4           ; Kéo đỉnh ngăn xếp vào R4
21  0310  22                RET                 ; Trở về nguồn gọi
22        0311              END                 ; Kết thúc tệp tin hợp ngữ
```

Lời giải:

Trước hết lưu ý rằng đối với các lệnh PUSH và POP ta phải xác định địa chỉ trực tiếp của thanh ghi được đẩy vào, kéo ra từ ngăn xếp. Dưới đây là sơ đồ khung của ngăn xếp.

Sau lệnh LCALL thứ nhất			Sau lệnh PUSH 4			Sau lệnh POSH 5		
0B			0B			0B	67	R5
0A			0A	99	R4	0A	09	R4
09	00	PCH	09	00	PCH	09	00	PCL
08	0B	PCL	0B	0B	PCL	08	0B	PCL

Cần phải nhấn mạnh rằng trong việc sử dụng LCALL thì địa chỉ đích của các chương trình con có thể ở đâu đó trong phạm vi 64k byte không gian bộ nhớ của

8051. Điều này không áp dụng cho tất cả mọi lệnh gọi CALL chẳng hạn như đối với ACALL dưới đây:

```
; MAIN program calling subroutines
                ORG             0
MAIN:           LCALL           SUBR-1
                LCALL           SUBR-2
                LCALL           SUBR-3

HERE:           SJMP            MAIN
;---------------- end of MAIN
;
SUBR-1l         ...
                ...
                RET
; ---------------- end of subroutinel  1
; SUBR-1l       ...
                ...
                RET
; ---------------- end of subroutinel  2
 ; SUBR-1l      ...
                ...
                RET
; ---------------- end of subroutinel  3
                END                     ; end of the asm    file
```

Hình 3.1: Chương trình chính hợp ngữ của 8051 có gọi các chương trình con.

3.2.5 Lệnh gọi tuyệt đối ACALL (Absolute call).

Lệnh ACALL là lệnh 2 byte khác với lệnh LCALL dài 3 byte. Do ACALL chỉ có 2 byte nên địa chỉ đích của chương trình con phải nằm trong khoảng 2k byte địa chỉ vì chỉ có 11bit của 2 byte được sử dụng cho địa chỉ. Không có sự khác biệt nào giữa ACALL và LCALL trong khái niệm cất bộ đếm chương trình vào ngăn xếp hay trong chức năng của lệnh trở về RET. Sự khác nhau duy nhất là địa chỉ đích của lệnh LCALL có thể nằm bất cứ đâu trong phạm vi 64k byte không gian địa chỉ của 8051, còn trong khi đó địa chỉ của lệnh ACALL phải nằm trong khoảng 2 byte. Trong nhiều biến thế của 8051 do các hãng cung cấp thì ROM trên chíp chỉ có 1k byte.. Trong những trường hợp như vậy thì việc sử dụng ACALL thay cho LCALL có thể tiết kiệm được một số byte bộ nhớ của không gian ROM chương trình.

Ví dụ 3.11:

Một nhà phát triển sử dụng chíp vi điều khiển Atmel AT89C1051 cho một sản phẩm. Chíp này chỉ có 1k byte ROM Flash trên chíp. Hỏi trong khi lệnh LCALL và ACALL thì lệnh nào hữu ích nhất trong lập trình cho chíp này.

Lời giải:

Lệnh ACALL là hữu ích hơn vì nó là lệnh 2 byte. Nó tiết kiệm một byte mỗi lần gọi được sử dụng.

Tất nhiên, việc sử dụng các lệnh gọn nhẹ, chúng ta có thể lập trình hiệu quả bằng cách có một hiểu biết chi tiết về tất cả các lệnh được hỗ trợ bởi bộ vi xử lý đã cho và sử dụng chúng một cách khôn ngoan. Xét ví dụ 3.12 dưới đây.

Ví dụ 3.12:

Hãy viết lại chương trình ở ví dụ 3.8 một cách hiệu quả mà bạn có thể:

Lời giải:

```
        ORG     0
        MOV     A, #55H         ; Nạp Avới giá trị 55H
BACK:   MOV     P1, A           ; Xuất giá trị trong A ra cổng P1
        ACALL   DELAY           ; Giữ chậm
        CPL     A               ; Bù thành ghi A
        SJMP    BACK            ; Tiếp tục thực hiện vô hạn
; -------- Đây là chương trình con giữ chậm DELAY
DELAY:
        MOV     R5,  #0FFH      ; Nạp R5 = 255 (hay FFH) làm cho bộ đếm
AGAIN:  DJNZ    R5, AGAIN       ; Dừng ở đây cho đến khi R5  = 0
        RET                     ; Trở về
        END                     ; Kết thúc
```

3.3 Tạo và tính toán thời gian giữ chậm.

3.3.1 Chu kỳ máy:

Đối với CPU để thực hiện một lệnh thì mất một chu kỳ đồng hồ này được coi như các chu kỳ máy. Phụ lục AppendixA.2 cung cấp danh sách liệt kê các lệnh 8051 và các chu kỳ máy của chúng. Để tính toán một độ trễ thời gian, ta sử dụng danh sách liệt kê này. Trong họ 8051 thì độ dài của chu kỳ máy phụ thuộc vào tần số của bộ dao động thạch anh được nối vào hệ thống 8051. Bộ dao động thạch anh cùng với mạch điện trên chip cung cấp xung đồng hồ cho CPU của 8051 (xem chương 4). Tần số của tinh thể thạch anh được nối tới họ 8051 dao động trong khoảng 4MHz đến 30 MHz phụ thuộc vào tốc độ chíp và nhà sản xuất. Thường xuyên nhất là bộ dao động thạch anh tần số 10.0592MHz được sử dụng để làm cho hệ 8051 tương thích với cổng nối tiếp của PC IBM (xem chương 10). Trong 8051, một chu kỳ máy kéo dài 12 chu kỳ dao động. Do vậy, để tính toán chu kỳ máy ta lấy 1/12 của tần số tinh thể thạch anh, sau đó lấy giá trị nghịch đảo như chỉ ra trong ví dụ 3.13.

Ví dụ 3.13:

Đoạn mã dưới đây trình bày tần số thạch anh cho 3 hệ thống dựa trên 8051 khác nhau. Hãy tìm chu kỳ máy của mỗi trường hợp: a) 11.0592MHz b) 16MHz và c) 20MHz.

Lời giải:

a) 11.0592/12 = 921.6kHz; Chu kỳ máy là 1/921.6kHz = 1.085μs (micro giây)
b) 16MHz/12 = 1.333MHz; Chu kỳ máy MC = 1/1.333MHz = 0.75μs
c) 20MHz/12 = 1.66MHz ⇒ MC = 1/1.66MHz = 0.60μs

Ví dụ 3.14:

Đối với một hệ thống 8051 có 11.0592MHz hãy tìm thời gian cần thiết để thực hiện các lệnh sau đây.

a) MOV R3, #55 b) DEC R3 c) DJNZ R2 đích

d) LJMP e) SJMP f) NOP g) MUL AB

Lời giải:

Chu kỳ máy cho hệ thống 8051 có tần số đồng hồ là 11.0592MHz Là 1.085μs như đã tính ở ví dụ 3.13. Bảng A-1 trong phụ lục Appendix A trình bày số chu kỳ máy đối với các lệnh trên. Vậy ta có:

Lệnh	Chu kỳ máy	Thời gian thực hiện
(a) MOV R3, #55	1	$1 \times 1.085\ \mu s = 1.085\ \mu s$
(b) DEC R3	1	$1 \times 1.085\ \mu s = 1.085\ \mu s$
(c) DJNZ R2, target	2	$2 \times 1.085\ \mu s = 2.17\ \mu s$
(d) LJMP	2	$2 \times 1.085\ \mu s = 2.17\ \mu s$
(e) SJMP	2	$2 \times 1.085\ \mu s = 2.17\ \mu s$
(f) NOP	1	$1 \times 1.085\ \mu s = 1.085\ \mu s$
(g) MUL AB	4	$4 \times 1.085\ \mu s = 4.34\ \mu s$

3.3.2 Tính toán độ trễ.

Như đã trình bày ở trên đây, một chương trình con giữ chậm gồm có hai phần: (1) thiết lập bộ đếm và (2) một vòng lặp. Hầu hết thời gian giữ chậm được thực hiện bởi thân vòng lặp như trình bày ở ví dụ 3.15.

Ví dụ 3.15:

Hãy tìm kích thước của thời gian giữ chậm trong chương trình sau, nếu tần số giao động thach anh là 11.0592MHz.

```
            MOV   A, #55H
AGAIN: MOV        P1, A
            ACALL DELAY
            CPL   A
            SJMP  AGAIN
; -------- Time delay
DELAY: MOV        R3, #200
HERE  :     DJNZ  R3, HERE
            RET
```

Lời giải:

Từ bảng A-1 của phụ lục Appendix A ta có các chu kỳ máy sao cho các lệnh của chương trình con giữ chậm là:

```
DELAY:    MOV     R3, #200     1
HERE  :   DJNZ    R3, HERE     2
          RET                  1
```

Do vậy tổng thời gian giữ chậm là $[(200 \times 2) + 1 + 1] \times 1.085 = 436.17\mu s$.

Thông thường ta tính thời gian giữ chậm dựa trên các lệnh bên trong vòng lặp và bỏ qua các chu kỳ đồng hồ liên quan với các lệnh ở ngoài vòng lặp.

Trong ví dụ 3.15 giá trị lớn nhất mà R3 có thể chứa là 255, do vậy một cách tăng độ trễ là sử dụng lệnh UOP (không làm gì) trong vòng lặp để tiêu tốn thời gian một cách đơn giản. Điều này được chỉ ra trong ví dụ 3.16 dưới đây.

Ví dụ 3.16:

Hãy tìm độ trễ thời gian cho chương trình con sau. Giả thiết tần số dao động thạch anh là 11.0592MHz.

```
                                        Số chu kỳ máy
DELAY:    MOV       R3, #250            1

HERE :    NOP                           1
          NOP                           1
          NOP                           1
          NOP                           1
          DJNZ      R3, HERE            2

          RET                           1
```

Lời giải:

Thời gian trễ bên trong vòng lặp HERE là [250 (1 + 1 + 1 + 1 + 1 + 2)] × 1.0851μs = 1627.5μs. Cộng thêm hai lệnh ngoài vòng lặp ta có 1627.5μs × 1.085μs = 1629.67μs.

3.3.3 Độ trễ thời gian của vòng lặp trong vòng lặp.

Một cách khác để nhận được giá trị từ độ trễ lớn là sử dụng một vòng lặp bên trong vòng lặp và cũng được gọi là vòng lặp lồng nhau. Xem ví dụ 3.17 dưới đây.

Ví dụ 3.17:

Đối với một chu kỳ máy 1.085μs hãy tính thời gian giữ chậm trong chương trình con sau:

```
DELAY:                                  chu kỳ máy
          MOV       R2, #200            1
AGAIN:    MOV       R3, #250            1
HERE:     NOP                           1
          NOP                           1
          DJNZ      R3, HERE            2
          DJNZ      R2, AGAIN           2
          RET                           1
```

Lời giải:

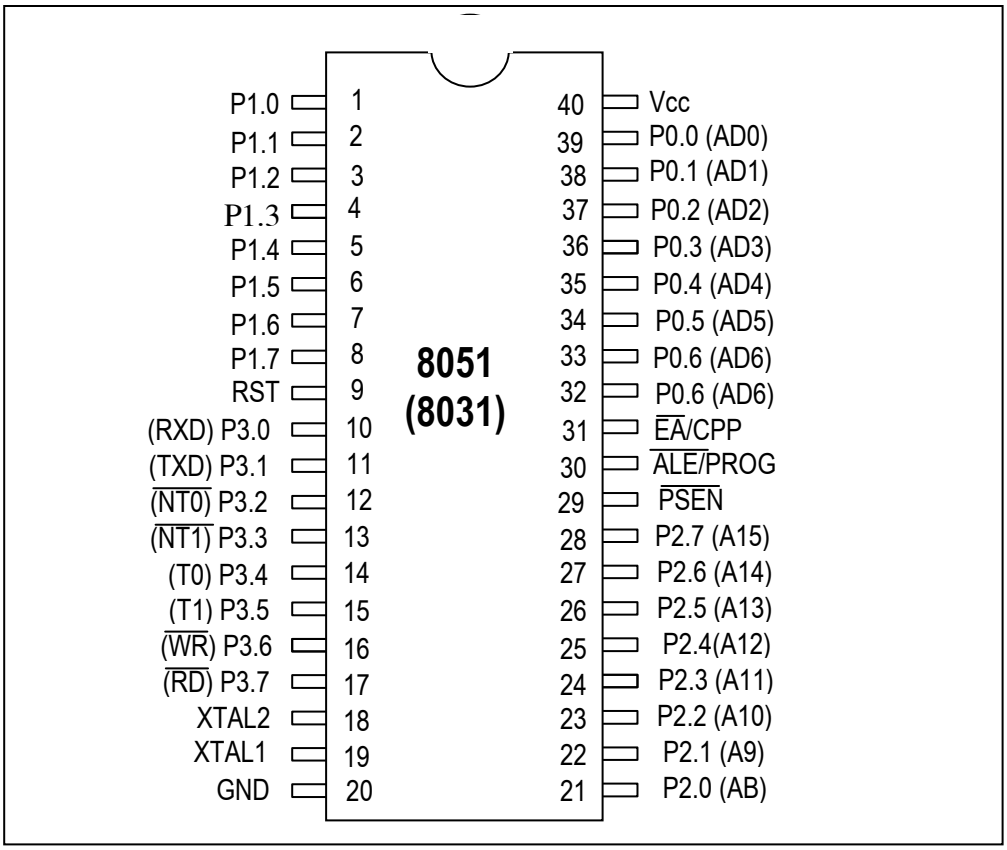
Đối với vòng lặp HERE ta có (4 × 250) × 1.085μs = 1085μs. Vòng lặp AGAIN lặp vòng lặp HERE 200 lần, do vậy thời gian trễ là 200 × 1085μs 217000μs, nên ta không tính tổng phí. Tuy nhiên, các lệnh "MOV R3, #250" và "DJNZ R2, AGAIN" ở đầu và cuối vòng lặp AGAIN cộng (3 × 200 × 1.085μs) = 651μs vào thời gian trễ và kết quả ta có 217000 + 651 = 217651μs = 217.651 miligiây cho tổng thời gian trễ liên quan đến chương trình con giữ chậm DELAY nói trên. Lưu ý rằng,

trong trường hợp vòng lặp lồng nhau cũng như trong mọi vòng lặp giữ chậm khác thời gian xấp xỉ gần dúng vì ta bỏ qua các lệnh đầu và cuối trong chương trình con.

CHƯƠNG 4
Lập trình cho cổng vào - ra I/0

4.1 Mô tả chân của 8051.

Mặc dù các thành viên của họ 8051 (ví dụ 8751, 89C51, DS5000) đều có các kiểu đóng vỏ khác nhau, chẳng hạn như hai hàng chân DIP (Dual In-Line Pakage) dạng vỏ dẹt vuông QFP (Quad Flat Pakage) và dạng chíp không có chân đỡ LLC (Leadless Chip Carrier) thì chúng đều có 40 chân cho các chức năng khác nhau như vào ra I/0, đọc $\overline{RD}$, ghi $\overline{WR}$, địa chỉ, dữ liệu và ngắt. Cần phải lưu ý rằng một số hãng cung cấp một phiên bản 8051 có 20 chân với số cổng vào-ra ít hơn cho các ứng dụng yêu cầu thấp hơn. Tuy nhiên, vì hầu hết các nhà phát triển chính sử dụng chíp đóng vỏ 40 chân với hai hàng chân DIP nên ta chỉ tập chung mô tả phiên bản này.



Hình 4.1: Sơ đồ bố trí chân của 8051.

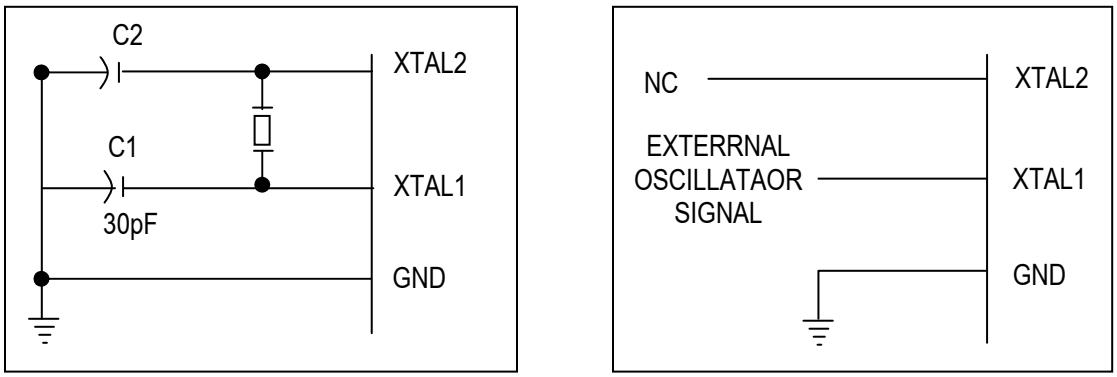
Trên hình 4.1 là sơ đồ bố trí chân của 8051. Ta thấy rằng trong 40 chân thì có 32 chân dành cho các cổng P0, P1, P2 và P3 với mỗi cổng có 8 chân. Các chân còn lại được dành cho nguồn V_{CC}, đất GND, các chângiao động XTAL1 và XTAL2 tái lập RST cho phép chốt địa chỉ ALE truy cập được địa chỉ ngoài $\overline{EA}$, cho phép cất chương trình $\overline{PSEN}$. Trong 8 chân này thì 6 chân V_{CC}, GND, XTAL1, XTAL2, RST và $\overline{EA}$ được các họ 8031 và 8051 sử dụng. Hay nói cách khác là chúng phải được

nối để cho hệ thống làm việc mà không cần biết bộ vi điều khiển thuộc họ 8051 hay 8031. Còn hai chân khác là $\overline{PSEN}$ và ALE được sử dụng chủ yếu trong các hệ thống dựa trên 8031.

1. Chân V_{CC}: Chân số 40 là V_{CC} cấp điện áp nguồn cho chíp. Nguồn điện áp là +5V.
2. Chân GND: Chân GND: Chân số 20 là GND.
3. Chân XTAL1 và XTAL2:

8051 có một bộ giao động trên chíp nhưng nó yêu cầu có một xung đồng hồ ngoài để chạy nó. Bộ giao động thạch anh thường xuyên nhất được nối tới các chân đầu vào XTAL1 (chân 19) và XTAL2 (chân 18). Bộ giao động thạch anh được nối tới XTAL1 và XTAL2 cũng cần hai tụ điện giá trị 30pF. Một phía của tụ điện được nối xuống đất như được trình bày trên hình 4.2a.

Cần phải lưu ý rằng có nhiều tốc độ khác nhau của họ 8051. Tốc độ được coi như là tần số cực đại của bộ giao động được nối tới chân XTAL. Ví dụ, một chíp 12MHz hoặc thấp hơn. Tương tự như vậy thì một bộ vi điều khiển cũng yêu cầu một tinh thể có tần số không lớn hơn 20MHz. Khi 8051 được nối tới một bộ giao động tinh thể thạch anh và cấp nguồn thì ta có thể quan sát tần số trên chân XTAL2 bằng máy hiện sóng. Nếu ta quyết định sử dụng một nguồn tần số khác bộ giao động thạch anh chẳng hạn như là bộ giao động TTL thì nó sẽ được nối tới chân XTAL1, còn chân XTAL2 thì để hở không nối như hình 4.2b.



Hình 4.2: a) Nối XTAL tới 8051 b) Nối XTAL tới nguồn đồng bộ ngoài.

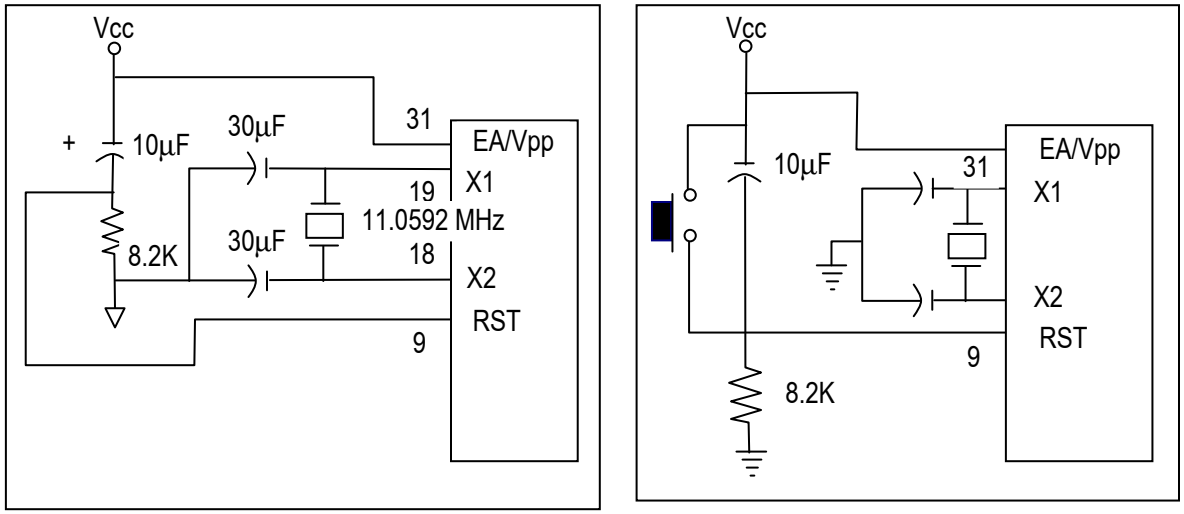
4. Chân RST.

Chân số 9 là chân tái lập RESET. Nó là một đầu vào và có mức tích cực cao (bình thường ở mức thấp). Khi cấp xung cao tới chân này thì bộ vi điều khiển sẽ tái lập và kết thúc mọi hoạt động. Điều này thường được coi như là sự tái bật nguồn. Khi kích hoạt tái bật nguồn sẽ làm mất mọi giá trị trên các thanh ghi. Bảng 4.1 cung cấp một cách liệt kê các thanh ghi của 8051 và các giá trị của chúng sau khi tái bật nguồn.

Bảng 4.1: Giá trị một số thanh ghi sau RESET.

Register	Reset Value
PC	0000
ACC	0000
B	0000
PSW	0000
SP	0000
DPTR	0007
	0000

Lưu ý rằng giá trị của bộ đếm chương trình PC là 0 khi tái lập để ép CPU nạp mã lệnh đầu tiên từ bộ nhớ ROM tại vị trí ngăn nhớ 0000. Điều này có nghĩa là ta phải đặt dòng đầu tiên của mã nguồn tại vị trí ngăn nhớ 0 của ROM vì đây là mã CPU tĩnh thức và tìm lệnh đầu tiên. Hình 4.3 trình bày hai cách nối chân RST với mạch bật nguồn.



Hình 4.3: a) Mạch tái bật nguồn RESET.
b) Mạch tái bật nguồn với Debounce.

Nhằm làm cho đầu vào RESET có hiệu quả thì nó phải có tối thiểu 2 chu kỳ máy. Hay nói cách khác, xung cao phải kéo dài tối thiểu 2 chu kỳ máy trước khi nó xuống thấp.

Trong 8051 một chu kỳ máy được định nghĩa bằng 12 chu kỳ dao động như đã nói ở chương 3 và được trình bày tại vị trí 4.1.

5. Chân $\overline{EA}$:

Các thành viên họ 8051 như 8751, 98C51 hoặc DS5000 đều có ROM trên chíp lưu cất chương trình. Trong các trường hợp như vậy thì chân $\overline{EA}$ được nối tới V_{CC}. Đối với các thành viên củ họ như 8031 và 8032 mà không có ROM trên chíp thì mã chương trình được lưu cất ở trên bộ nhớ ROM ngoài và chúng được nạp cho 8031/32. Do vậy, đối với 8031 thì chân $\overline{EA}$ phải được nối đất để báo rằng mã chương trình được cất ở ngoài. $\overline{EA}$ có nghĩa là truy cập ngoài (External Access) là chân số 31 trên vỏ kiểu DIP. Nó là một chân đầu vào và phải được nối hoặc với V_{CC} hoặc GND. Hay nói cách khác là nó không được để hở.

Ở chương 14 chúng ta sẽ trình bày cách 8031 sử dụng chân này kết hợp với $\overline{\text{PSEN}}$ để truy cập các chương trình được cất trên bộ nhớ ROM ở ngoài 8031. Trong các chíp 8051 với bộ nhớ ROM trên chíp như 8751, 89C51 hoặc DS5000 thì $\overline{\text{EA}}$ được nối với V_{CC}.

Ví dụ 4:

Hãy tìm chu kỳ máy đối với a) XTAL = 11.0592MHz b) XTAL = 16MHz.

Lời giải:

a) 11.0592MHz/12 = 921.6kHz.

Chu kỳ máy = 1/921.6kHz = 1.085μs.

b) 16MHz/12 = 1.333MHz

Chu kỳ máy = 1/1.333MHz = 0.75μs.

Các chân mô tả trên đây phải được nối mà không cần thành viên nào được sử dụng. Còn hai chân dưới đây được sử dụng chủ yếu trong hệ thống dựa trên 8031 và sẽ được trình bày chi tiết ở chương 11.

6. Chân $\overline{\text{PSEN}}$:

Đây là chân đầu ra cho phép cất chương trình (Program Store Enable) trong hệ thống dựa trên 8031 thì chương trình được cất ở bộ nhớ ROM ngoài thì chân này được nối tới chân OE của ROM. Chi tiết được bàn ở chương 14.

7. Chân ALE:

Chân cho phép chốt địa chỉ ALE là chân đầu ra và được tích cực cao. Khi nối 8031 tới bộ nhớ ngoài thì cổng 0 cũng được cấp địa chỉ và dữ liệu. Hay nói cách khác 8031 dồn địa chỉ và dữ liệu qua cổng 0 để tiết kiệm số chân. Chân ALE được sử dụng để phân kênh địa chỉ và dữ liệu bằng cách nối tới chân G của chíp 74LS373. Điều này được nói chi tiết ở chương 14.

8. Các chân cổng vào ra và các chức năng của chúng.

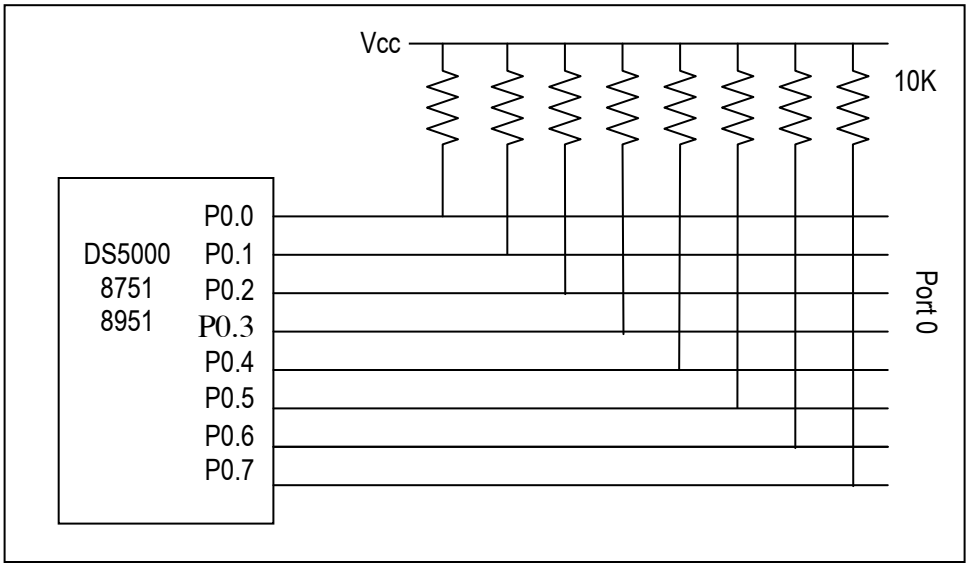
Bốn cổng P0, P1, P2 và P3 đều sử dụng 8 chân và tạo thành cổng 8 bít. Tất cả các cổng khi RESET đều được cấu hình như các đầu ra, sẵn sàng để được sử dụng như các cổng đầu ra. Muốn sử dụng cổng nào trong số các cổng này làm đầu vào thì nó phải được lập trình.

9. Cổng P0.

Cổng 0 chiếm tất cả 8 chân (từ chân 32 đến 39). Nó có thể được dùng như cổng đầu ra, để sử dụng các chân của cổng 0 vừa làm đầu ra, vừa làm đầu vào thì mỗi chân phải được nối tới một điện trở kéo bên ngoài 10kΩ. Điều này là do một thực tế là cổng P0 là một màng mở khác với các cổng P1, P2 và P3. Khái niệm máng mở được sử dụng trong các chíp MOS về chừng mực nào đó nó giống như Cô-lec-tơ hở đối với các chíp TTL. Trong bất kỳ hệ thống nào sử dụng 8751, 89C51 hoặc DS5000 ta thường nối cổng P0 tới các điện trở kéo, Xem hình 4.4 bằng cách này ta có được các ưu điểm của cổng P0 cho cả đầu ra và đầu vào. Với những điện trở kéo ngoài được nối khi tái lập cổng P0 được cấu hình như một cổng đầu ra. Ví dụ, đoạn mã sau đây sẽ liên tục gửi ra cổng P0 các giá trị 554 và AAH.

```
        MOV     A, #554
BACK:   MOV     P0, A
        ACALL   DELAY
```

```
        CPL     A
        SJMP    BACK
```



Hình 4.4: Cổng P0 với các điện trở kéo.

a) Cổng P0 đầu vào: Với các điện trở được nối tới cổng P0 nhằm để tạo nó thành cổng đầu vào thì nó phải được lập trình bằng cách ghi 1 tới tất cả các bit. Đoạn mã dưới đây sẽ cấu hình P0 lúc đầu là đầu vào bằng cách ghi 1 đến nó và sau đó dữ liệu nhận được từ nó được gửi đến P1.

b)

```
        MOV     A,#FFH      ; Gán A = FF dạng Hex
        MOV     P0, A       ; Tạo cổng P0 làm cổng đầu vào bằng cách
                            ; Ghi tất cả các bit của nó.
BACK:   MOV     A, P0       ; Nhận dữ liệu từ P0
        MOV     P1, A       ; Gửi nó đến cổng 1
        SJMP    BACK        ; Lặp lại
```

b) Vai trò kép của cổng P0: Như trình bày trên hình 4.1, cổng P0 được gán AD0 - AD7 cho phép nó được sử dụng vừa cho địa chỉ, vừa cho dữ liệu. Khi nối 8051/31 tới bộ nhớ ngoài thì cổng 0 cung cấp cả địa chỉ và dữ liệu 8051 dồn dữ liệu và địa chỉ qua cổng P0 để tiết kiệm số chân. ALE báo nếu P0 có địa chỉ hay dữ liệu khi ALE - 0 nó cấp dữ liệu D0 - D7. Do vậy, ALE được sử dụng để tách địa chỉ và dữ liệu với sự trợ giúp của chốt 74LS373 mà ta sẽ biết cụ thể ở chương 14.

10. Cổng P1.

Cổng P1 cũng chiếm tất cả 8 chân (từ chân 1 đến chân 8) nó có thể được sử dụng như đầu vào hoặc đầu ra. So với cổng P0 thì cổng này không cần đến điện trở kéo vì nó đã có các điện trở kéo bên trong. Trong quá trình tái lạp thì cổng P1 được cấu hình như một cổng đầu ra. Ví dụ, đoạn mã sau sẽ gửi liên tục các giá trị 55 và AAH ra cổng P1.

```
        MOV     A, #55H
BACK:   MOV     P1, A
```

```
        ACALL   DELAY
        SJMP    BACK
```

Cổng P1 như đầu vào: Để biến cổng P1 thành đầu vào thì nó phải được lập trình bằng cách ghi một đến tất cả các bit của nó. Lý do về điều này được bàn ở mục lục Appendix C.2. Trong đoạn mã sau, cổng P1 lúc đầu đươc cấu hình như cổng đầu vào bằng cách ghi 1 vào các bit của nó và sau đó dữ liệu nhận được từ cổng này được cất vào R7, R6 và R5.

```
        MOV     A, #0FFH    ; Nạp A = FF ở dạng hex
        MOV     P1, A       ; Tạo cổng P1 thành cổng đầu vào bằng
                            ; cách ghi 1 vào các bit của nó.
        MOV     A, P1       ; Nhận dữ liệu từ P1
        MOV     R7, A       ; Cất nó vào thanh ghi R7
        ACALL   DELAY       ; Chờ
        MOV     A,  P1      ; Nhận dữ liệu khác từ P1
        MOV     R6, A       ; Cất nó vào thanh ghi R6
        ACALL   DELAY       ; Chờ
        MOV     A, P1       ; Nhận dữ liệu khác từ cổng P1
        MOV     R5, A       ; Cất nó vào thanh ghi R5
```

11. Cổng P2:

Cổng P2 cũng chiếm 8 chân (các chân từ 21 đến 28). Nó có thể được sử dụng như đầu vào hoặc đầu ra giống như cổng P1, cổng P2 cũng không cần điện trở kéo vì nó đã có các điện trở kéo bên trong. Khi tái lập, thì cổng P2 được cấu hình như một cổng đầu ra. Ví dụ, đoạn mã sau sẽ gửi liên tục ra cổng P2 các giá trị 55H và AAH. Đó là tất cả các bit của P2 lên xuống liên tục.

```
        MOV     A, #55H
BACK:   MOV     P2,A
        ACALL   DELAY
        CPL     A
        SJMP    BACK
```

a) Cổng P2 như đầu vào.

Để tạo cổng P2 như đầu vào thì nó phải được lập trình bằng cách ghi các số 1 tới tất cả các chân của nó. Đoạn mã sau đây đầu tiên cấu hinh P2 là cổng vào bằng cách ghi một đến tất cả các chân của nó và sau đó dữ liệu nhận được từ P2 được gửi liên tục đến P1.

```
        MOV     A, 0FFH     ; Gán A giá trị FF dạng Hex
        MOV     P2, A       ; Tạo P2 là cổng đầu vào bằng cách
                            ; ghi một đến các chân của nó
BACK:   MOV     A, 2        ; Nhận dữ liệu từ P2
        MOV     P1, A       ; Gửi nó đến P1
        SJMP    BACK        ; Lặp lại
```

b) Vai trò kép của P2.

Trong các hệ thống dựa trên 8751, 89C51 và DS5000 thì P2 được dùng như đầu ra đơn giản. Tuy nhiên trong hệ thống dựa trên 80312 thì cổng P2 phải được dùng cùng với P0 để tạo ra địa chỉ 16 brt đối với bộ nhớ ngoài. Như chỉ ra trên hình 4.1 cổng P2 cũng được chỉ định như là A8 - A15 báo chức năng kép của nó. Vì một bộ 8031 có khả năng trung cập 64k byte bộ nhớ ngoài, nó cần một đường địa chỉ 16 bít. Trong khi P.0 cung cấp 8 bit thấp qua A0 - A7. Công việc của P2 là cung cấp các bít địa chỉ A8 - A15. Hay nói cách khác khi 8031được nối tới bộ nhớ ngoài thì P2 được dùng cho 8 bít của địa chỉ 16 bit và nó không thể dùng cho vào ra. Điều này sẽ được trình bày chi tiết ở chương 14.

Từ những trình bày trên đây ta có thể kết luận rằng trong các hệ thống dựa trên các bộ vi điều khiển 8751, 89C51 hoặc DS5000 thì ta có 3 cống P0, P1và P2 cho các thao tác vào ra và như thế là có thể đủ cho các ứng dụng với hầu hết các bộ vi điều khiển. Còn cấp P3 làđể dành cho ngắt và ta sẽ cùng bàn dưới đây.

11 - Cổng P3:

Cổng P3 chiếm tổng cộng là 8 chân từ chân 10 đến chân 17. Nó có thể được sử dụng như đầu vào hoặc đầu ra. Cống P3 không cần các điện trở kéo cũng như P1 và P2. Mặc dù cống P3 được cấu hình như một cống đầu ra khi tái lập, nhưng đây không phải là cách nó được ứng dụng phổ biến nhất. Cống P3 có chức năng bổ xung là cung cấp một số tín hiệu quan trọng đặc biệt chẳng hạn như các ngắt. Bảng 4.2 cung cấp các chức năng khác của cống P3. Thông tin này áp dụng cho cả 8051 và 8031.

Bảng 4.2: Các chức năng khác của cống P3

Bít của cống P3	Chức năng	chân số
P3.0	Nhận dữ liệu (RXD)	10
P3.1	Phát dữ liệu (TXD)	11
P3.2	Ngắt 0(INT0)	12
P3.3	Ngắt 1(INT1)	13
P3.4	Bộ định thời 0 (TO)	14
P3.5	1 Bộ định thời 1(T1)	15
P3.6	Ghi (WR)	16
P3.7	Đọc (RD)	17

Các bit P3.0 và P3.1 được dùng cho các tín hiệu nhận và phát dữ liệu trong truyền thông dữ liệu nối tiếp. Xem chương 10 đết biết các chúng được nối ghép như thế nào. Các bit P3.2 và P3.3 được dành cho các ngắt ngoài và chúng được trình bày chi tiết ở chương 11. Bit P3.4 và P3.5 được dùng cho các bộ định thêm 0 và 1và chi tiết được trình bày ở chương 9. Cuối cùng các bit P3.6 và P3.7 được cấp cho các tín hiệu ghi và đọc các bộ nhớ ngoài được nối tới các hệ thống dựa trên 8031. Chương 14 sẽ trình bày cách chúng được sử dụng như thế nào trong các hệ thống dựa trên 8031. Trong các hệ thống dựa trên 8751, 89C51 hoặc D35000 thì các chân P3.6 và P3.7 được dùng cho vào - ra còn các chân khác của P3 được sử dụng bình thường trong vai trò chức năng thay đổi.

4.2 Lập trình vào - ra: thao tác bit.

4.2.1 các cách khác nhau để truy cập toàn bộ 8 bit.

Trong đoạn mà dưới đây cũng như trong nhiều ví dụ vào ra trước đây toàn bộ 8 bit của cổng P1 được cập.

```
BACK:   MOV     A, # 55H
        MOV     P1,A
        ACALL   DELAY
        MOV     A, #0AAH
        MOV     P1, A
        ACALL   DELAY
        SJMP    BACK
```

Đoạn mà trên chốt mỗi bit của P1 một cách liên tục. Chúng ta đã thắng một biến thế của chương trình trên trước đó. Bây giờ ta có thể viết lại đoạn mã trên theo cách hiệu quả hơn bằng cách truy cập trực tiếp cổng mà không qua thanh ghi tổng như sau:

```
BACK:   MOV     P1, # 55H
        ACALL   DELAY
        MOV     P1, #00H
        CALL    DELAY
        SJMP    BACK
```

Ta có thể viết một dạng khác của đoạn mà trên bằng kỹ thuật đọc - sửa đổi ghi như ở mục 4.2.2 dưới đây.

4.2.2 Đặc điểm Đọc- sửa đổi - ghi (Read - Modify – Write).

Các cổng trong 8051 có thể được truy cập bằngkỹ thuật được gọi là Đọc-sửa đổi-ghi. Đặc điểm này tiết kiệm rất nhiều dòng lệnh bằng cách kết hợp tất cả 3 thao tác: 1đọc cổng, 2 sửa đổi nóvà 3 ghi nó ra cổng vào một lệnh đơn. Đoạn mà dưới đây trước hết đặt 01010101(nhị phân) vào cổng 1. Sau đólệnh "XLR P1, #0FFH" thực hiện phép lô-gích OR loại trừ là XOR trên cổng p1 với 1111 1111 (nhị phân) và sau đó ghi kết quả trở lại cổng P1.

```
        MOV     P1, #55H        ; P1 = 01010101
AGAIN:  XLR     P1,# 0FFH       ; EX - 0R P1 với 1111 1111
        ACALL   DELAY
        SJMP    AGAIN
```

Lưu ý rằng lệnh X0R của 55H và FFH sẽ cho kết quả là AAH. Tương tự như vậy lệnh X0R của AAH với FFH lại cho giá trị kết quả là 55H. Các lệnh lô-gích được trình bày ở chương 7.

4.2.3. Khả năng đánh địa chỉ theo bít của các cổng

Có nhiều lúc chúng ta cần truy cập chỉ 1 hoặc2 bít của cổng thay vì truy cập cả 8 bit của cổng. Một điểm mạnh của các cổng 8051 là chúng có khả năng truy cập từng bít riêng rẽ mà không làm thay đổi các bít còn lại trong cổng đó ví dụ, đoạn mà dưới đây chốt bit P1.2 liên tục:

```
BACK:   CPL     P1.2        ; Lấy bù 2 chỉ riêng bit P1.2
```

```
        ACALL   DELAY
        SJMP    BACK
```

Một biến thể khác của đoạn mã trên là:

```
AGACN:  SETB    P1.2    ; Chỉ thay đổi bít P1.2 lên cao
        ACALL   DELAY
        CLR     P1.2    ; Xoá bít P1.2 xuống thấp
        ACALL   DELAY
        SJMP    AGAIN
```

Lưu ý rằng bít P1.2 là bít thứ 3 của cổng P1, vì bít thứ nhất là P1.0 và bit thứ hai là P1.1 v.v...

Bảng 4.3 trình bày các bít của các cổng vào ra của 8051. Xem ví dụ 4.2 về thao tác bít của các bít vào - ra. Lưu ý rằng trong ví dụ 4.2 các bit không dùng đến là không bị ảnh hưởng. Đây là khả năng đánh địa chỉ theo bit của các cổng vào - ra và là một trong những điểm mạnh nhất của bộ vi điều khiển 8051.

Ví dụ 4.2: hãy viết chương trình thực hiện các công việc sau:

a) Duy trì hiển thị bit P1.2 cho đến khi nó lên cáp
b) Khi P1.2 lên cao, hãy ghi giá trị 45H vào cổng P0
c) Gửi một xung cao xuống thấp (H-to-L) tới P2.3

Lời giải:

```
        SET     P1.2        ; Tạo bit P1.2 là đầu vào
        MOV     A, #45H     ; Gán A = 45H
AGAIN:  JNB     P1.2, AGAIN ; Thoát khi P1.2 = 1
        MOV     P0, A       ; Xuất A tới cổng P0
        SETB    P2.3        ; Đưa P2.3 lên cao
        CLR     P2.3        ; Tạo P2.3 xuống thấp để có xung H-T0-L
```

Trong chương trình này lệnh "JNB P1.2, AGCN" (JNB có nghĩa là nhảy nếu không bit) ở lại vòng lặp cho đến khi P1.2 chưa lên cao. Khi P1.2 lên cao nó thoát ra khỏi vòng lặp ghi giá trị 45H tới cổng P0 và tạo ra xung H-to-L bằng chuỗi các lệnh SETB và CLR.

CHƯƠNG 5
Các chế độ đánh địa chỉ của 8051

CPC có thể truy cập dữ liệu theo nhiều cách khác nhau. Dữ liệu có thể ở trong một thanh ghi hoặc trong bộ nhớ hoặc được cho như một giá trị tức thời các cách truy cập dữ liệu khác nhau được gọi là các chế độ đánh địa chỉ. Chương này chúng ta bàn luận về các chế độ đánh địa chỉ của 8051 trong phạm vi một số ví dụ.

Các chế độ đánh địa chỉ khác nhau của bộ vi xử lý được xác định như nó được thiết kế và do vậy người lập trình không thể đánh địa chỉ khác nhau là:

1. tức thời 2. Theo thanh ghi 3. Trực tiếp
4. gián tiếp qua thanh ghi 5. Theo chỉ số

5.1 Các chế độ đánh địa chỉ tức thời và theo thanh ghi

5.1.1 Chế độ đánh địa chỉ tức thời

Trong chế độ đánh địa chỉ này toán hạng nguồn là một hằng số. Và như tên gọi của nó thì khi một lệnh được hợp dịch toán hạng đi tức thi ngay sau mã lệnh. Lưu ý rằng trước dữ liệu tức thời phải được đặt dấu (#) chế độ đánh địa chỉ này có thể được dùng để nạp thông tin vào bất kỳ thanh ghi nào kể cả thanh ghi con trỏ dữ liệu DPTR. Ví dụ:

```
MOV   A, # 25H          ; Nạp giá trị 25H vào thanh ghi A
MOV   R4, #62           ; Nạp giá trị  62 thập phân vào R4
MOV   B, #40H           ; Nạp giá trị 40 H vào thanh ghi B
MOV   DPTR, #4521H      ; Nạp 4512H vào con trỏ dữ liệu DPTR
```

Mặc dù thanh ghi DPTR là 16 bit nó cũng có thể được truy cập như 2 thanh ghi 8 bit DPH và DPL trong đó DPH là byte cao và DPL là byte thấp. Xét đoạn mã dưới đây:

```
MOV         DPTR, #2550H
MOV         A, #50H
MOV         DPH, #25H
```

Cũng lưu ý rằng lệnh dưới đây có thể tạo ra lỗi vì giá trị nạp vào DPTR lớn hơn16 bit:

```
MOV         DPTR, # 68975       ; Giá tri không hợp lệ  > 65535 (FFFFH)
```

Ta có thể dùng chỉ lệnh Eqw để truy cập dữ liệu tức thời như sau

```
            COUNT       EDU     30
            ...         ...
            MOV         R4, #COUNT              ; R4 = 1E (30 = 1EH)
            MOV         DPTR, #MYDATA           ; DPTR = 200H

            ORG     200H
MYDATA:     DB      “America”
```

Lưu ý rằng ta cũng có thể sử dụng chế độ đánh được chỉ tức thời để gửi dữ liệu đến các cổng của 8051.
Ví dụ "MOV P1, #55H" là một lệnh hợp lệ.

5.1.2 chế độ đánh địa chỉ theo thanh ghi:

Chế độ đánh địa chỉ theo thanh ghi liên quan đến việc sử dụng các thanh ghi để dữ liệu cần được thao tác các ví dụ về đánh địa chỉ theo thanh ghi như sau:

```
MOV    A,  RO          ; Sao nội dung thanh ghi RO vào thanh ghi A
MOV    R2, A           ; Sao nội  dung thanh ghi  A vào thanh ghi R2
ADD    A; R5           ; Cộng nội dung  thanh ghi R5 vào thanh ghi A
ADD    A, R7           ; Cộng nội dung thanh ghi  R7 vào thanh ghi A
MOV    R6, A           ; Lưu nội dung thanh ghi A vào thanh ghi R6
```

Cũng nên lưu ý rằng các thanh ghi nguồn và đích phải phù hợp về kích thước. Hay nói cách khác, nếu viết " MOV DPTR, A" sẽ cho một lỗi vì nguồn là thanh ghi 8 bit và đích lại là thanh ghi 16 bit. Xét đoạn mã sau:

```
MOV    DPTR, #25F5H
MOV    R7,  DPL
MOV    R6,  DPH
```

Để ý rằng ta có thể chuyển dữ liệu giữa thanh ghi tích luỹ A và thanh ghi Rn (n từ 0 đến 7) nhưng việc chuyển dữ liệu giữa các thanh ghi Rn thì không được phép. Ví dụ, lệnh "MOV R4, R7" là không hợp lệ.

Trong hai chế độ đánh địa chỉ đầu tiên, các toán hạng có thể hoặc ở bên trong một trong các thanh ghi hoặc được gắn liền với lệnh. Trong hầu hết các chương trình dữ liệu cần được xử lý thường ở trong một số ngăn của bộ nhớ RAM hoặc trong không gian mà của ROM. Có rất nhiều cách để truy cập dữ liệu này mà phần tiếp theo sẽ xét đến.

5.2 Truy cập bộ nhớ sử dụng các chế độ đánh địa chỉ khác nhau.

5.2.1 Chế độ đánh địa chỉ trực tiếp.

Như đã nói ở chương 2 trong 8051 có 128 byte bộ nhớ RAM. Bộ nhớ RAM được gán các địa chỉ từ 00 đến FFH và được phân chia như sau:
1. Các ngăn nhớ từ 00 đến 1FH được gán cho các băng thanh ghi và ngăn xếp.
2. Các ngăn nhớ từ 20H đến 2FH được dành cho không gian đánh địa chỉ theo bit để lưu các dữ liệu 1 bit.
3. Các ngăn nhớ từ 30H đến 7FH là không gian để lưu dữ liệu có kích thước 1byte.

Mặc dù toàn bộ byte của bộ nhớ RAM có thể được truy cập bằng chế độ đánh địa chỉ trực tiếp, nhưng chế độ này thường được sử dụng nhất để truy cập các ngăn nhớ RAM từ 30H đến 7FH. Đây là do một thực tế là các ngăn nhớ dành cho băng ghi được truy cập bằng thanh ghi theo các tên gọi của chúng là R0 - R7 còn các ngăn nhớ khác của RAM thì không có tên như vậy. Trong chế độ đánh địa chỉ trực tiếp thì dữ liệu ở trong một ngăn nhớ RAM mà địa chỉ của nó được biết và địa chỉ này được cho như là một phần của lệnh. Khác với chế độ đánh địa chỉ tức thì mà toán hạng tự nó được cấp với lệnh. Dấu (# 0 là sự phân biệt giữa hai chế độ đánh địa chỉ. Xét các ví dụ dưới đây và lưu ý rằng các lệnh không có dấu (#):

```
MOV    R0, 40H         ; Lưu nội dung của ngăn nhớ 40H của RAM vào R0
```

```
MOV    56H, A          ; Lưu nội dung thanh ghi A vào ngăn nhớ 56H của RAM
MOV    R4, 7FH         ; Chuyển nôi dung ngănnhớ 7FH của RAM  vào R4
```

Như đã nói ở trước thì các ngăn nhớ trừ 0 đến 7 của RAM được cấp cho bằng 0 của các thanh ghi R0 - R7. Các thanh ghi này có thể được truy cập theo 2 cách như sau:

```
MOV    A, 4            ; Hai lệnh này giống nhau đều sao nội dung thanh ghi  R4 vào A
MOV    A, R4

MOV    A, 7            ; Hai lệnh này đều như nhau là sao nội dung R7 vào thanh ghi A
MOV    A,R7
```

Để nhấn mạnh sự quan trọng của dấu (#) trong các lệnh của 8051. Xét các mã cho sau đây:

```
MOV    R2, #05         ; Gán R2=05
MOV    A, 2            ; Sao nội dung thanh ghi R2 vào A
MOV    B, 2            ; Sao nội dung thanh ghi R2 vào  B
MOC    7,2             ; Sao nội dung thanh ghi R7 vì lệnh "MOV R7, R2" là không hợp  lệ.
```

Mặc dù sử dụng các tên R0 - R7 dễ hơn các địa chỉ bộ nhớ của chúng nhưng các ngăn nhớ 30H đến 7FH của RAM không thể được truy cập theo bất kỳ cách nào khác là theo địa chỉ của chúng vì chúng không có tên.

5.2.2 các thanh ghi SFSR và các địa chỉ của chúng.

Trong các thanh ghi được nói đến từ trước đến giờ ta thấy rằng các thanh ghi R0 - R7 là một phần trong 128 byte của bộ nhớ RAM. Vậy còn các thanh ghi A, B, PSW và DPTR là một bộ phận của nhóm các thanh ghi nhìn chung được gọi là các thanh ghi đặc biệt SFR (Special Funtion Register). Có rất nhiều thanh ghi với chức năng đặc biệt và chúng được sử dụng rất rộng rãi mà ta sẽ trình bày ở các chương sáu. Các thanh ghi FR có thể được truy cập theo tên của chúng (mà dễ hơn rất nhiều) hoặc theo các địa chỉ của chúng. Ví dụ địa chỉ của thanh ghi A là EOH và thanh ghi B là FOH như cho ở trong bảng 5.1. Hãy để ý đến những cặp lệnh có cùng ý nghĩa dưới đây:

```
MOV    0E0H, #55H      ; Nạp 55H vào thanh ghi A(A=55H)
MOV    A,   #55H       ;

MOV    0F0H, #25H      ; Nạp 2SH vào thanh ghi  B ( B = 25)
MOV    3, #25H         ;

MOV    0E0H            ; Sao nội dung thanh ghi R2 vào A
MOV    A, R2           ;

MOV    0F0             ; Sao nội dung thanh ghi R0 vào B
MOV    B, R0           ;
```

Bảng 5.1 dưới đây liệt kê các thanh ghi chức năng đặc biệt SFR của 8051 và các địa chỉ của chúng. Cần phải lưu ý đến hai điểm sau về các địa chỉ của SFR:

1. Các thanh ghi SFR có địa chỉ nàm giữa 80H và FFH các địa chỉ này ở trên 80H, vì các địa chỉ từ 00 đến 7FH là địa chỉ của bộ nhớ RAM bên trong 8051.

2. không phải tất cả mọi địa chỉ từ 80H đến FFH đều do SFH sử dụng, nhưng vị trí ngăn nhớ từ 80H đến FFH chưa dùnglà để dữ trữ và lập trình viên 8051 cũng không được sử dụng.

Bảng 5.1: Các địa chỉ của thanh ghi chức năng đặc biệt SFR

Lệnh	Tên	Địa chỉ
ACC*	Thanh ghi tích luỹ (thanh ghi tổng) A	0E0H
B*	Thanh ghi B	0F0H
PSW*	Từ trạng thái chương trình	0D0H
SP	Con trỏ ngăn xếp	81H
DPTR	Con trỏ dữ liệu hai byte	
DPL	Byte thấp của DPTR	82H
DPH	Byte cao của DPTR	83H
P0*	Cổng 0	80H
P1*	Cổng 1	90H
P2*	Cổng 2	0A0H
P3*	Cổng 3	0B0H
IP*	Điều khiển ưu tiên ngắt	0B8H
IE*	Điều khiển cho phép ngắt	A08H
TMOD	Điều khiển chế độ bộ đếm/ Bộ định thời	89H
TCON*	Điều khiển bộ đếm/ Bộ định thời	88H
T2CON*	Điều khiển bộ đếm/ Bộ định thời 2	0C8H
T2MOD	Điều khiển chế độ bộ đếm/ Bộ định thời 2	0C9H
TH0	Byte cao của bộ đếm/ Bộ định thời 0	8CH
TL0	Byte thấp của bộ đếm/ Bộ định thời 0	8AH
TH1	Byte cao của bộ đếm/ Bộ định thời 1	8DH
TL1	Byte thấp của bộ đếm/ Bộ định thời 1	8BH
TH2	Byte cao của bộ đếm/ Bộ định thời 2	0CDH
TL2	Byte thấp của bộ đếm/ Bộ định thời 2	0CCH
RCAP2H	Byte cao của thanh ghi bộ đếm/ Bộ định thời 2	0CBH
RCAP2L	Byte thấp của thanh ghi bộ đếm/ Bộ định thời 2	0CAH
SCON*	Điều khiển nối tiếp	98H
SBUF	Bộ đệm dữ liệu nối tiếp	99H
PCON	Điều khiển công suất	87H

*Các thanh ghi có thể đánh địa chỉ theo bit.

Xét theo chế độ đánh địa chỉ trực tiếp thì cần phải lưu ý rằng giá trị địa chỉ được giới hạn đến 1byte, 00 - FFH. Điều này có nghĩa là việc sử dụng của chế độ đánh địa chỉ này bị giới hạn bởi việc truy cập các vị trí ngăn nhớ của RAM và các thanh ghi với địa chỉ được cho bên trong 8051.

Ví dụ 5.1:

Viết chương trình để gửi 55H đến cổng P1 và P2 sử dụng hoặc

a) Tên các cổng

b) Hoặc địa chỉ các cổng

Lời giải:

```
a)      MOV     A, #55H ; A = 55H
```

```
MOV    P1, A          ; P1 = 55H
MOV    P2, A          ; P2 = 55H
```

b) Từ bảng 5.1 ta lấy đại chỉ cổng P1 là 80H và P2 là A0H

```
MOV    A, #55H        ; A = 55H
MOV    80H, A         ; P1 = 55H
MOV    0A0H, A        ; P2 = 55H
```

5.2.3 Ngăn xếp và chế độ đánh địa chỉ trực tiếp.

Một công dụng chính khác của chế độ đánh địa chỉ trực tiếp là ngăn xếp. Trong họ 8051 chỉ có chế độ đánh địa chỉ trực tiếp là được phép đẩy vào ngăn xếp. Do vậy, một lệnh như "PVSH A" là không hợp lệ. Việc đẩy thanh ghi A vào ngăn xếp phải được viết dưới dạng "PVAH 0E0H" với 0E0H là địa chỉ của thanh ghi A. Tương tự như vậy để đẩy thanh ghi R3 rãnh 0 vào ngăn xếp ta phải viết là "PVSH 03". Chế độ đánh địa chỉ trực tiếp phải được sử dụng cho cả lệnh POP. Vì dụ "POP 04" sẽ kéo đỉnh của ngăn xếp vào thanh ghi R4 rãnh 0.

Ví dụ 5.2:

Trình bày mã để đẩy thanh ghi R5, R6 và A vào ngăn xếp và sau đó kéo chùng ngược trở lại R2, R3 và B tương ứng.

Lời giải:

```
PUSH    05      ; Đẩy R5 vào ngăn xếp
PUSH    06      ; Đẩy R6 vào ngăn xếp
PUSH    0E0H    ; Đẩy thanhghi A vào ngăn xếp
POP     0F0H    ; Kéo đỉnh ngăn xếp cho vào thanh ghi B
                ; Bây giờ B = A
POP     02      ; Kéo đỉnh ngăn xếp cho vào thanh ghi R2
                ; Bây giờ R2= R6
POP     03      ; Kéo đỉnh ngăn xếp cho vào thanh ghi
                ; Bây giờ R3 = R5
```

5.2.4 chế độ đánh địa chỉ gián tiếp thanh ghi.

Trong chế độ này, một thanh ghi được sử dụng như một con trỏ đến dữ liệu. Nếu dữ liệu ở bên trong CPU thì chỉ các thanh ghi R0 và R1 được sử dụng cho mục đích này. Hay nói cách khác các thanh ghi R2 - R7 không có thể dùng được để giữ địa chỉ của toán hạng nằm trong RAM khi sử dụng chế độ đánh địa chỉ này khi Ro và R1 được dùng như các con trỏ, nghĩa là khi chúng giữ các địa chỉ của các ngăn nhớ RAM thì trước chúng phải đặt dấu (@) như chỉ ra dưới đây.

```
MOV    A, @ R0        ; Chuyển nội dung của ngăn nhớ RAM có địa chỉ trong RO và A
MOV    @ R1, B        ; Chuyển nội dung của B vào ngăn nhớ RAM có địa chỉ ở R1
```

Lưu ý rằng R0 cũng như R1 luôn có dấu "@" đứng trước. Khi không có dấu này thì đó là lệnh chuyển nội dung các thanh ghi Ro và R1 chứ không phải dữ liệu ngăn nhớ mà địa chỉ có trong R0 và R1.

Ví dụ 5.3:

Viết chương trình để sao chép giá trị 55H vào ngăn nhớ RAM tại địa chỉ 40H đến 44H sử dụng:

a) Chế độ đánh địa chỉ trực tiếp

b) Chế độ đánh địa chỉ gián tiếp thanh ghi không dùng vòng lặp

c) Chế độ b có dùng vòng lặp

Lời giải:

```
        MOV   A, #55H       ; Nạp A giá trị  55H
        MOV   40H, A        ; Sao chép A vào ngăn nhớ RAM  40H
        MOV   41H, A        ; Sao chép A vào ngăn nhớ RAM  41H
        MOV   42H, A        ; Sao chép A vào ngăn nhớ RAM  42H
        MOV   43H, A        ; Sao chép A vào ngăn nhớ RAM  43H
        MOV   44H, A        ; Sao chép A vào ngăn nhớ RAM  44H
b)
        MOV   A, # 55H      ; Nạp vào A giá trị 55H
        MOV   R0, #40H      ; Nạp con trỏ  R0 = 40 H
        MOV   @R0, A        ; Sao chép A vào vị trí ngăn nhớ RAM do R0 chỉ đến
        INC   R0            ; Tăng con trỏ. Bây gì R0 = 41H
        MOV   @R0, A        ; Sao chép A vào vị trí ngăn nhớ RAM do R0 chỉ
        INC   R0            ; Tăng con trỏ. Bây giờ R0 = 42H
        MOV   @R0,A         ; Sao chép Avào vị trí ngăn nhớ RAM do R0 chỉ
        INC   R0            ; Tăng con trỏ. Bây giờ R0 = 43H
        MOV   @R0, A        ; Sao chép A vào vị trí ngăn nhớ RAM do R0 chỉ
        MOV   @R0, A        ;Tăng con trỏ. Bây gờ R0 = 44H
        MOV   @R0, A
c)
        MOV   A, # 55H      ; Nạp vào A giá trị 55H
        MOV   R0, #40H      ; Nạp con trỏ địa chỉ ngăn nhớ RAM R0 = 40H
        MOV   R2, #05       ; Nạp bộ đếm R2 = 5
AGAIN:  MOV   @R0, A        ; Sao chép A vào vị trí ngăn nhớ RAM do Ro chi đến
        INC                 ; Tăng con trỏ Ro
        DJNZ  R2, AGAIN     ; Lặp lại cho đến khi bộ đếm = 0.
```

5.2.5 Ưu điểm của chế độ đánh địa chỉ gián tiếp thanh ghi.

Một trong những ưu điểm của chế độ đánh địa chỉ gián tiếp thanh ghi là nó làm cho việc truy cập dữ liệu năng động hơn so với chế độ đánh địa chỉ trực tiếp.

Ví dụ 5.3 trình bày trường hợp sao chép giá trị 55H vào các vị trí ngăn nhớ của RAM từ 40H đến 44H .

Lưu ý rằng lời giải b) có hai lệnh được lặp lại với một số lần. Ta có thể tạo ra vòng lặp với hai lệnh này như ở lời giải c). Lời giải c) là hiệu quả nhất và chỉ có thể khi sử dụng chế độ đánh địa chỉ gián tiếp qua thanh ghi. Vòng lặp là không thể trong chế độ đánh địa chỉ trực tiếp. Đây là sự khác nhau chủ yếu giữa đánh địa chỉ trực tiếp và gián tiếp.

Ví dụ 5.4:

Hãy viết chương trình để xoá 16 vị trí ngăn nhớ RAM bắt đầu tại địa chỉ 60H.

Lời giải:

```
        CLR    A            ; Xoá A=0
        MOV    R1, #60H     ; Nạp con trỏ. R1= 60H
        MOV    R7, #16H     ;Nạp bộ đếm, R7 = 1 6  (10 H dạng hex)
```

```
AGAIN:    MOV   @R1, A      ; Xoá vị trí ngăn nhớ RAM do R1 chỉ đến
          INC   R1          ; Tăng R1
          DJNZ  R7, AGAiN   ; Lặp lại cho đến khi bộ đếm = 0
```

Một ví dụ về cách sử dụng cả R0 và R1 trong chế độ đánh địa chỉ gián tiếp thanh ghi khi truyền khối được cho trong ví dụ 5.5.

Ví dụ 5.5:

Hãy viết chương trình để sao chép một khối 10 byte dữ liệu từ vị trí ngăn nhớ RAM bắt đầu từ 35H vào các vị trí ngăn nhớ RAM bắt đầu từ 60H

Lời giải:

```
          MOV   R0, # 35H      ; Con trỏ nguồn
          MOV   R1, #60H       ; Con trỏ đích
          MOV   R3, #10        ; Bộ đếm
BACK:     MOV   A, @R0         ; Lấy 1byte từ nguồn
          MOV   @R1, A         ; Sao chép nó đến đích
          INC   R0             ; Tăng con trỏ nguồn
          INC   R1             ; Tăng con trỏ đích
          DJNZ  R3, BACK       ; Lặp lại cho đến khi sao chép hết 10 byte
```

5.2.6 Hạn chế của chế độ đánh địa chỉ gián tiếp thanh ghi trong 8051.

Như đã nói ở phần trước rằng R0 và R1 là các thanh ghi duy nhất có thể được dùng để làm các con trỏ trong chế độ đánh địa chỉ gián tiếp thanh ghi. Vì R0 và R1 là các thanh ghi 8 bit, nên việc sử dụng của chúng bị hạn chế ở việc truy cập mọi thông tin trong các ngăn nhớ RAM bên trong (các ngăn nhớ từ 30H đến 7FH và các thanh ghi SFR). Tuy nhiên, nhiều khi ta cần truy cập dữ liệu được cất trong RAM ngoài hoặc trong không gian mã lệnh của ROM trên chip. Hoặc là truy cập bộ nhớ RAM ngoài hoặc ROM trên chíp thì ta cần sử dụng thanh ghi 16 bit đó là DPTR.

5.2.7 Chế độ đánh địa chỉ theo chỉ số và truy cập bộ nhớ ROM trên chíp.

Chế độ đánh địa chỉ theo chỉ số được sử dụng rộng rãi trongviệc truy cập các phân tử dữ liệu của bảng trong không gian ROM chương trình của 8051. Lệnh được dùng cho mục đích này là "Move A, @ A + DPTR". Thanh ghi 16 bit DPTR là thanh ghi A được dùng để tạo ra địa chỉ của phân tử dữ liệu được lưu cất trong ROM trên chíp. Do các phân tử dữ liệu được cất trong không gian mã (chương trình) của ROM trên chip của 8051, nó phải dùng lệnh Move thay cho lệnh Mov (chủ C ở cuối lệnh là chỉ mà lệnh Code). Trong lệnh này thì nội dung của A được bổ xung vào thanh ghi 16 bit DPTR để tạo ra địa chỉ 16 bit của dữ liệu cần thiết. Xét ví dụ 5.6.

Ví dụ 5.6:

Giả sử từ "VSA" được lưu trong ROM có đĩa chỉ bắt đầu từ 200H và chương trình được ghi vào ROM bắt đầu từ địa chỉ 0. Hãy phân tích cách chương trình hoạt động và hãy phát biểu xem từ "VSA" sau chương trình này được cất vào đâu?

Lời giải:

```
          ORG   0000H          ; Bắt đầu đốt ROM tại địa chỉ 00H
          MOV   DPTR, #200H    ; Địa chỉ bảng trình bày DPTR = 200H
          CLA   A              ; Xoá thanh ghi A (A = 0)
          MOVC  A, @A + DPTR   ; Lấy ký tự từ không gian nhớ chương trình
          MOV   R0, A          ; Cất nó vào trong R0
```

```
                INC         DPTR                ; DPTR = 201, chỉ đến ký tự kế tiếp
                CLR         A                   ; Xoá thanh ghi A
                MOVC        A, @A + DPTR        ; Lấy ký tự kế tiếp
                MOV         R1, A               ; Cất nó vào trong R1
                INC         DPTR                ; DPTR = 202  con trỏ chỉ đến ký tự sau đó
                CLA         A                   ; Xoá thanh ghi A
                MOVC        A, @A + DPTR        ; Nhận ký tự kế tiếp
                MOV         R2, A               ; Cất nó vào R2
HERE:           SJMP        HERE                ; Dừng lại ở đây.
; Dữ liệu được đốt trong không gian mã lệnh  tại địa chỉ  200H
                            ORG  200H
MYDATA:                     DB "VSA"
                            END                 ; Kết thúc chương trình
```

Ở trong chương trình nói trên thì các vị trí ngăn nhớ ROM chương trình 200H - 2002H có các nội dung sau:
200 = ('U'); 201= ('S') và 202 = ('A').

Chúng ta bắt đầu với DPTR = 200H và A = 0.Lệnh "MOVC A, @ A + DPTR chuyển nội dung của vị trí nhớ 200H trong ROM (200H + 0 = 200H) vào A.

Thanh ghi A chứa giá trị 55H là giá trị mà ASC của ký tự "U". ký tự này được cất vào R0. Kế đó, DPTR được tăng lên tạo thành DPTR = 201H. A lại được xoá về 0 để lấy nội dung của vị trí nhớ kế tiếp trong ROM là 201H chưa ký tự "S". Sau khi chương trình này chạy ta có R0 = 55H, R1 = 53H và R2 = 41H là các mã ASCII của các ký tự "U", "S" và "A".

Ví dụ 5.7:

Giả sử không gian ROM bắt đầu từ địa chỉ 250H có chứa "America", hãy viết chương trình để truyền các byte vào các vị trí ngăn nhớ RAM bắt đầu từ địa chỉ 40H.

Lời giải

```
; (a)  Phương pháp này sử dụng một bộ đếm
                ORG         000
                MOV         DPTR, # MYDATA          ; Nạp con trỏ ROM
                MOV         R0, #40H                ; Nạp con trỏ RAM
                MOV         R2, #7                  ; Nạp bộ đếm
BACK:  CLR              A                           ; Xoá thanh ghi A
                MOVC        A, @A + DPTR            ;Chuyển dữ liệu từ khong gian mã
                MOV         R0, A                   ;Cất nó vào ngăn nhớ RAM
                INC         DPTR                    ; Tăng con trỏ ROM
                INC         R0                      ; Tăng con trỏ  RAM
                DJNZ        R2, BACK                ; Lặp lại cho đếnkhi bộ đếm = 0
HERE:  SJMP             HERE
;--------------- -- không gian mã của ROM trên chíp  dùng để cất dữ liệu ORG  250H

MYDATA:         DB              "AMER1CA"
                END

;(b) phương pháp này sử dụng ký tự null để kết thúc chuỗi

                ORG         000
                MOV         DPTR, #MYDATA           ; Nạp  con trỏ ROM
                MOV         R0, #40                 ; Nạp con trỏ   RAM
BACK:           CLR         A       S               ; Xoá thanh ghi A(A=0)
```

```
            MOVC    A, @A + DPTR            ; Chuyển dữ liệu từ không gian mã
            JZ      HERE                    ; Thoát ra nếu có ký tự Null
            MOV     DPTR, #MYDATA           ; Cất nó vào ngăndn nhớ của RAM
            INC     @R0, A                  ; Tăng con trỏ ROM
            INC     R0                      ; Tăng con trỏ RAM
            SJM     BACK                    ; Lặp lại
HERE:       SJMP    HERE
;------------------ không gian mã của ROM trên chíp dùng để cất dữ liệu ORG 250H
MYADTA:     DB “AMER1CA”, 0                 ; Ký tự Null để kết thúc chuỗi  END
```

Lưu ý đến cách ta sử dụng lệnh JZ để phát hiện ký tự NOLL khi kết thúc chuỗi

5.2.8 Bảng xắp xếp và sử dụng chế độ đánh địa chỉ theo chỉ số.

Bảng xắp xế là khái niệm được sử dụng rất rộng rãi trong lập trình các bộ vi xử lý. Nó cho phép truy cập các phần từ của một bảng thường xuyên được sử dụng với thao tác cực tiểu. Như một ví dụ, hãy giả thiết rằng đối với một ứng dụng nhất định ta cần x^2 giá trị trong phạm vi 0 đến 9. Ta có thể sử dụng một bảng xắp xếp thay cho việc tính toán nó. Điều này được chỉ ra trong ví dụ 5.8.

Ví dụ 5.8

Hãy viết một chương trình để lấy x giá trị cống P1 và gửi giá trị x^2 tới cổng P2 liên tục.

Lời giải:

```
            ORG     000
            MOV     DPTR, #300 H        ; Nạp địa chỉ bảng xắp xêlps
            MOV     A, #0FFH            ; Nạp A giá trị FFH
            MOV     P1, A               ; Đặt cổng P1 là đầu vào
BACK:       MOV     A, P1               ; Lấy giá trị X từ P1
            MOVC    A, @A + DPTR        ; Lấy giá trị  X từ bảng  XSDQ-TABLE
            MOV     P2, A               ; Xuất nó ra cổng P2
            SJMP    BACK                ; Lặp lại

            ORG     300H
XSQR - TABLE:
            DB      0, 1,  4, 9, 16, 25, 36, 49, 64, 81
            END
```

Lưu ý bảng lệnh đầu tiên có thể thay bằng “MOV DPTR, #XSQR - TABLE”.

Ví dụ 5.9:

Trả lời các câu hỏi sau cho ví dụ 5.8.

a) Hãy chỉ ra nội dung các vị trí 300 - 309H của ROM

b) Tại vị trí nào của ROM có giá trị 6 và giá trị bào nhiêu

c) Giả sử P1 có giá trị là 9 thì giá trị P2 là bao nhiêu (ở dạng nhị phân)?

Lời giải:

a) Các giá trị trong các ngăn nhớ 300H - 309H của ROM là:

300 = (00) 301 = (01) 302 = (04) 303 = (09)

304 = (10) $4 \times 4 = 16 = 10$ in hex

305 = (19) $5 \times 5 = 25 = 19$ in hex

306 = (24) $6 \times 6 = 36 = 24$H

307 = (31) 308 = (40) 309 = (51)

b) vị trí chứa giá trị 306H và giá trị là 24H
c) 01010001B là giá trị nhị phân của 51H và 81 ($9^2 = 81$)

Ngoài việc sử dụng DPTR để truy cập không gian bộ nhớ ROM chương trình thì nó còn có thể được sử dụng để truy cập bộ nhớ ngoài nối với 8051 (chương 14).

Một thanh ghi khác nữa được dùng trong chế độ đánh địa chỉ theo chỉ số là bộ đếm chương trình (AppendixA).

Trong nhiều ví dụ trên đây thì lệnh MOV đã được sử dụng để đảm bảo đính rõ ràng, mặc dù ta có thể sử dụng bất kỳ lệnh nào khác chừng nào nó hỗ trợ cho chế độ đánh địa chỉ. Ví dụ lệnh "ADD A, @R0" sẽ cộng nội dung ngăn nhớ cho RO chỉ đến vào nội dung của thanh ghi A.

CHƯƠNG 6
Các lệnh số học và các chương trình

6.1 Phép cộng và trừ không dấu.

Các số không dấu được định nghĩa như những dữ liệu mà tất cả mọi bit của chúng đều được dùng để biểu diễn dữ liệu và khó có bit dành cho dấu âm hoặc dương. Điều này có nghĩa là toán hạng có thể nằm giữa 00 và FFH (0 đến 255 hệ thập phân) đối với dữ liệu 8 bit.

6.1.1 Phép cộng các số không dấu.

Trong 8051 để cộng các số với nhau thì thanh ghi tổng (A) phải được dùng đến. Dạng lệnh ADD là:

ADD A, nguồn; A = A + nguồn

Lệnh ADD được dùng để cộng hai toán hạng. Toán hạng đích luôn là thanh ghi A trong khi đó toán hạng nguồn có thể là một thanh ghi dữ liệu trực tiếp hoặc là ở trong bộ nhớ. Hãy nhớ rằng các phép toán số học từ bộ nhớ đến bộ nhớ không bao giờ được phép trong hợp ngữ. Lệnh này có thể thay đổi một trong các bit AF, CF hoặc PF của thanh ghi cờ phụ thuộc vào các toán hạng liên quan. Tác động của lệnh ADD lên cờ tràn sẽ được trình bày ở mục 6.3 vì nó chủ yếu được sử dụng trong các phép toán với số có dấu. Xét ví dụ 6.1 dưới đây:

Ví dụ 6.1:

Hãy biểu diễn xem cá lệnh dưới đây tác động đến thanh ghi cờ như thế nào?

```
MOV    A, # 0F5H        ; A = F5H
MOV    A, # 0BH         ; A = F5 + 0B = 00
```

Lời giải:

```
  F5H                    1111   0101
+ 0BH                  +  0000   1011
  100H                    0000   0000
```

Sau phép cộng, thanh ghi A (đích) chứa 00 và các cờ sẽ như sau:

CY = 1 vì có phép nhớ từ D7

PF = 1 vì số các số 1 là 0 (một số chẵn) cờ PF được đặt lên 1.

AC = 1 vì có phép nhớ từ D3 sang D4

6.1.1.1 Phép cộng các byte riêng rẽ.

ở chương 2 đã trình bày một phép cộng 5 byte dữ liệu. Tổng số đã được cất theo chú ý nhỏ hơn FFH là giá trị cực đại một thanh ghi 8 bit có thể được giữ. Để tính tổng số của một số bất kỳ các toán hạng thì cờ nhớ phải được kiểm tra sau mỗi lần cộng một toán hạng. Ví dụ 6.2 dùng R7 để tích luỹ số lần nhớ mỗi khi các toán hạng được cộng vào A.

Ví dụ 6.2:

Giả sử các ngăn nhớ 40 - 44 của RAM có giá trị sau: 40 = (7D); 41 = (EB); 42 = (C5); 43 = (5B) và 44 = (30). Hãy viết một chương trình tính tổng của các giá trị trên. Cuối chương trình giá trị thanh ghi A chứa byte thấp và R7 chứa byte cao (các giá trị trên được cho ở dạng Hex).

Lời giải:

```
        MOV   R0, #40H        ; Nạp con trỏ
        MOV   R2, #5          ; Nạp bộ đệm
        CLR   A               ; Xoá thanh ghi A
        MOV   R7, A           ; Xoá thanh ghi R7
AGAIN:  ADD   A, @R0          ; Cộng byte con trỏ chỉ đến theo R0
        JNC   NEXT            ; Nếu CY = 0 không tích luỹ cờ nhớ
        INC   R7              ; Bám theo số lần nhớ
NEXT:   INC   R0              ; Tăng con trỏ
        DJNZ  R2, AGAIN       ; Lặp lại cho đến khi R0 = 0
```

Phân tích ví dụ 6.2:

Ba lần lặp lại của vòng lặp được chỉ ra dưới đây. Phần dò theo chương trình dành cho người đọc tự thực hiện.

Trong lần lặp lại đầu tiên của vòng lặp thì 7DH được cộng vào A với CY = 0 và R7 = 00 và bộ đếm R2 = 04.

Trong lần lặp lại thứ hai của vòng lặp thì EBH được cộng vào A và kết quả trong A là 68H với CY = 1. Vì cờ nhớ xuất hiện, R7 được tăng lên. Lúc này bộ đếm R2 = 03.

Trong lần lặp lại thứ ba thì C5H được cộng vào A nên A = 2DH và cờ nhớ lại bận. Do vậy R7 lại được tăng lên và bộ đệm R2 = 02.

Ở phần cuối khi vòng lặp kết thúc, tổng số được giữ bởi thanh ghi A và R7, trong đó A giữ byte thấp và R7 chứa byte cao.

6.1.1.2 Phép cộng vó nhớ và phép cộng các số 16 bit.

Khi cộng hai toán hạng dữ liệu 16 bit thì ta cần phải quan tâm đến phép truyền của cờ nhớ từ byte thấp đến byte cao. Lệnh ADDC (cộng có nhớ) được sử dụng trong những trường hợp như vậy. Ví dụ, xét phép cộng hai số sau: 3CE7H + 3B8DH.

```
    3C  E7
+   3B  8D
    78  74
    79
```

Khi byte thứ nhất được cộng (E7 + 8D = 74, CY = 1). Cờ nhớ được truyền lên byte cao tạo ra kết quả 3C + 3B + 1 = 78. Dưới đây là chương trình thực hiện các bước trên trong 8051.

Ví dụ 6.3:

Hãy viết chương trình cộng hai số 16 bit. Các số đó là 3CE7H và 3B8DH. Cất tổng số vào R7và R6 trong đó R6 chứa byte thấp.

Lời giải:

```
CLR                   ; Xoá cờ CY = 0
MOV     A, #0E7H      ; Nạp byte thấp vào A → A = E7H
ADD     A, #8DH       ; Cộng byte thấp vào A → a = 74H và CY = 1
MOV     R6, A         ; Lưu byte thấp của tổng vào R6
MOV     A, #3CH       ; Nạp byte cao vào A → A = 3CH
ADDC    A, #3BG       ; Cộng byte cao có nhớ vào A → A = 78H
                      ;
```

```
MOV     R7, A          ; Lưu byte cao của tổng vào R7
```

6.1.1.3 Hệ thống số BCD (số thập phân mã hoá theo nhị phân).

Số BCD là số thập phân được mã hoá theo nhị phân 9 mà không dùng số thập phân hay số thập lục (Hex). Biểu diễn nhị phân của các số từ 0 đến 9 được gọi là BCD (xem hình 6.1). Trong tài liệu máy tính ta thường gặp hai khái niệm đối với các số BCD là: BCD được đóng gói và BCD không đóng gói.

Digit	BCD	Digit	BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Hình 6.1: Mã BCD.

a- BCD không đóng gói.

Trong số BCD không đóng gói thì 4 bít thấp của số biểu diễn số BCD còn 4 bit còn lại là số 9. Ví dụ "00001001" và "0000 0101" là những số BCD không đóng gói của số 9 và số 5. Số BCD không đóng gói đòi hỏi một byte bộ nhớ hay một thanh ghi 8 bit để chứa nó.

b- BCD đóng gói.

Trong số BCD đóng gói thì một byte có 2 số BCD trong nó một trong 4 bit thấp và một trong 4 bit cao. Ví dụ "0101 1001" là số BCD đóng gói cho 59H. Chỉ mất 1 byte bộ nhớ để lưu các toán hạng BCD. Đây là lý do để dùng số BCD đóng gói vì nó hiệu quả gấp đôi trong lưu giữ liệu.

Có một vấn để khi cộng các số BCD mà cần phải được khắc phục. Vấn đề đó là sau khi cộng các số BCD đóng gói thì kết quả không còn là số BCD. Ví dụ:

```
MOV     A, #17H
ADD     A, #28H
```

Cộng hai số này cho kết quả là 0011 1111B (3FH) không còn là số BCD! Một số BCD chỉ nằm trong giải 0000 đến 1001 (từ số 0 đến số 9). Hay nói cách khác phép cộng hai số BCD phải cho kết quả là số BCD. Kết quả trên đáng lẽ phải là 17 + 28 = 45 (0100 0101). Để giải quyết vấn đề này lập trình viên phải cộng 6 (0110) vào số thấp 3F + 06 = 45H. Vấn đề tương tự cũng có thể xảy ra trong số cao (ví dụ khi cộng hai số 52H + 87H = D94). Để giải quyết vấn đề này ta lại phải cộng 6 vào số cao (D9H + 60H = 139). Vấn đề này phổ biến đến mức mọi bộ xử lý như 8051 đều có một lệnh để sử lý vấn đề này. Trong 8051 đó là lệnh "DA A" để giải quyết vấn đề cộng các số BCD.

6.1.1.4 Lệnh DA.

Lệnh DA (Decimal Adjust for addition điều chỉnh thập phân đối với phép cộng) trong 8051 để dùng hiệu chỉnh sự sai lệch đã nói trên đây liên quan đến phép cộng các số BCD. Lệnh giả "DA". Lệnh DA sẽ cộng 6 vào 4 bit thấp hoặc 4 bit cao nếu cần. Còn bình thường nó đê nguyên kết quả tìm được. Ví dụ sau sẽ làm rõ các điểm này.

```
MOV     A, #47H     ; A = 47H là toán hạng BCD đầu tiên
MOV     B, #25H     ; B = 25H  là toán hạng BCD thứ hai
ADD     A, B        ; Cộng các số hex (nhị phân) A = 6CH
DA      A           ; Điều chỉnh cho phép cộng BCD (A = 72H)
```

Sau khi chương trình được thực hiện thanh ghi A sẽ chứa 72h (47 + 25 = 72). Lệnh "DA" chỉ làm việc với thanh ghi A. Hay nói cách khác trong thanh ghi nguồn có thể là một toán hạng của chế độ đánh địa chỉ bất kỳ thì đích phải là thanh ghi A để DA có thể làm việc được. Cũng cần phải nhấn mạnh rằng lệnh DA phải được sử dụng sau phép cộng các toán hạng BCD và các toán hạng BCD không bao giờ có thể có số lớn hơn 9. Nói cách khác là không cho phép có các số A - F. Điều quan trọng cũng phải lưu ý là DA chỉ làm việc sau phép cộng ADD, nó sẽ không bao giờ làm việc theo lệnh tăng INC.

Tóm tắt về hoạt động của lệnh DA.

Hoạt động sau lệnh ADD hoặc ADDC.

1. Nếu 4 bit thấp lớn hơn 9 hoặc nếu AC = 1 thì nó cộng 0110 vào 4 bít thấp.
2. Nếu 4 bit cao lớn hơn 9 hoặc cờ CY = 1 thì nó cộng 0110 vào 4 bit cao.

Trong thực tế thì cờ AC chỉ để dùng phục vụ cho phép cộng các số BCD và hiệu chỉnh nó. Ví dụ, cộng 29H và 18H sẽ có kết quả là 41H sai với thực tế khi đó các số BCD và để sửa lại thì lệnh DA sẽ cộng 6 vào 4 bit thấp để có kết quả là đúng (vì AC = 1) ở dạng BCD.

```
    29H                  0010  1001
+   18H              +   0001  1000
    41H                  0100  0001        AC = 1
+   6                +         0110
    47H                  0100  0111
```

Ví dụ 6.4:

Giả sử 5 dữ liệu BCD được lưu trong RAM tại địa chỉ bắt đầu từ 40H như sau: 40 = (71), 41 = (11), 42 = (65), 43 = (59) và 44 = (37). Hãy viết chương trình tính tổng của tất cả 5 số trên và kết quả phải là dạng BCD.

Lời giải:

```
        MOV     R0, #40H        ; Nạp con trỏ
        MOV     R2, #5          ; Nạp bộ đếm
        CLR     A               ; Xoá thanh ghi A
        MOV     R7, A           ; Xoá thanhg ghi R7
AGAIN:  ADD     A, @R0          ; Cộng byte con trỏ chỉ bởi R0
        DA      A               ; Điều chỉnh về dạng BCD đúng
        JNC     NEXT            ; Nếu CY = 0 không tích luỹ cờ nhớ
        JNC     R7              ; Tăng R7 bám theo số lần nhớ
NEXT:   INC     R0              ; Tăng R0 dịch con trỏ lên ô nhớ kế tiếp
        DJNZ    R2, AGAIN       ; Lặp lại cho đến khi R2 = 0
```

6.1.2 Phép trừ các số không dấu.

Cú pháp: SUBB A, nguồn; A = A - nguồn - CY.

Trong rất nhiều các bộ xử lý có hai lệnh khác nhau cho phép trừ đó là SUB và SUBB (trừ có mượn - Sub, tract with Borrow). Trong 8051 ta chỉ có một lệnh SUBB

duy nhất. Để thực hiện SUB từ SUBB, do vậy có hai trường hợp cho lệnh SUBB là: với CY = 0 và với CY = 1. Lưu ý rằng ở đây ta dùng cờ CY để mượn.

6.1.2.1 Lệnh SUBB với CY = 0.

Trong phép trừ thì các bộ vi xử lý 8051 (thực tế là tất cả mọi CPU hiện đại) đều sử dụng phương pháp bù 2. Mặc dù mỗi CPU đều có mạch cộng, nó có thể quá cồng kềnh (và cần nhiều bóng bán dẫn) để thiết kế mạch trừ riêng biệt. Vì ly do đó mà 8051 sử dụng mạch cộng để thực hiện lệnh trừ. Giả sử 8051 sử dụng mạch cộng để thực hiện lệnh trừ và rằng CY - 0 trước khi thực hiện lệnh thì ta có thể tóm tắt các bước mà phần cứng CPU thực hiện lệnh SUBB đối với các số không dấu như sau:

1. Thực hiện lấy bù 2 của số trừ (toán hạng nguồn)
2. Cộng nó vào số bị trừ (A)
3. Đảo nhớ

Đây là 3 bước thực hiện bởi phần cứng bên trong của CPU 8051 đối với mỗi lệnh trừ SUBB bất kể đến nguồn của các toán hạng được cấp có được hỗ trợ chế độ đánh địa chỉ hay không? Sau ba bước này thì kết quả có được và các cờ được bật. Ví dụ 6.5 minh hoạ 3 bước trên đây:

Ví dụ 6.5:

Trình bày các bước liên quan dưới đây:

```
CLR     C           ; Tạo CY = 0
MOV     A, #3FH     ; Nạp 3FH vào A (A = 3FH)
MOV     R3, #23H    ; Nạp 23H vào R3 (R3 = 23H)
SUBB    A, R3       ; Trừ A cho R3 đặt kết quả vào A
```

Lời giải:

```
    A =  3F     0011  1111        0011  1111
-   R3 = 23     0010  0011    +   1101  1101   bù 2 của R3 (bước 1)
      1C                        1  0001  1100 - 1C (bước 2)
                                0  CF = 0 (bước 3)
```

Các cờ sẽ được thiết lập như sau: CY = 0, AC = 0 và lập trình viên phải được nhìn đến cờ nhớ để xác định xem kết quả là âm hay dương.

Nếu sau khi thực hiện SUBB mà CY = 0 thì kết quả là dương. Nếu CY = 1 thì kết quả âm và đích có giá trị bù 2 của kết quả. Thông thường kết quả được để ở dạng bù 2 nhưng các lệnh bù CPL và tăng INC có thể được sử dụng để thay đổi nó. Lệnh CPL thực hiện bù 1 của toán hạng sau đó toán hạng được tăng lên 1 (INC) để trở thành dạng bù 2. Xem ví dụ 6.6.

Ví dụ 6.6:

Phân tích chương trình sau:

```
        CLR   C
        MOV   A, #4CH       ; Nạp A giá trị 4CH (A = 4CH)
        SUBB  A, #6EH       ; Trừ A cho 6EH
        JNC   NEXT          ; Nếu CY = 0 nhảy đến đích NEXT
        CPL   A             ; Nếu CY = 1 thực hiện bù 1
        INC   A             ; Tăng 1 để có bù 2
NEXT:   MOV   R1, A         ; Lưu A vào R1
```

Lời giải:

Các bước thực hiện lệnh "SUBB A, 6EH" như sau:

```
    4C      0100    1100                    0100    1100
-   6E      0110    1110    → lấy bù 2      1001    0010    (bước 1)
  - 22                              0       1101    1110 = (bước 2)
                                            đảo CY = 1(bước 3)
```

Cờ CY = 1, kết quả âm ở dạng bù 2.

6.1.2.2 Lệnh SUBB khi CY = 1.

Lệnh này được dùng đối với các số nhiều byte và sẽ theo dõi việc mượn của toán hạng thấp. Nếu CY = 1 trước khi xem thực hiện SUBB thì nó cũng trừ 1 từ kết quả. Xem ví dụ 6.7.

Ví dụ 6.7:

Phân tích chương trình sau:

```
CLR   C          ; CY = 0
MOV   A, #62     ; A = 62H
SUBB  A, #96H    ; 62H - 96H = CCH with CY = 1
MOV   R7, A      ; Save the result
MOV   A, #27H    ; A = 27H
SUBB  A, #12H    ; 27H - 12H - 1 = 14H
MOV   R6, A      ; Save the result
```

Lời giải:

Sau khi SUBB thì A = 62H - 96H = CCH và cờ nhớ được lập báo rằng có mượn. Vì CY = 1 nên khi SUBB được thực hiện lần thứ 2 thì a = 27H - 12H - 1 = 14H. Do vậy, ta có 2762H - 1296H = 14CCH.

6.2 Nhân và chia các số không dấu.

Khi nhân và chia hai số trong 8051 cần phải sử dụng hai thanh ghi A và B vì các lệnh nhân và chia chỉ hoạt động với những thanh ghi này.

6.2.1 Nhân hai số không dấu.

Bộ vi điều khiển chỉ hỗ trợ phép nhân byte với byte. Các byte được giả thiết là dữ liệu không dấu. Cấu trúc lệnh như sau:

MOV AB ; Là phép nhân A × B và kết quả 16 bit được đặt trong A và B.

Khi nhận byte với byte thì một trong các toán hạng phải trong thanh ghi A và toán hạng thứ hai phải ở trong thanh ghi B. Sau khi nhân kết quả ở trong các thanh ghi A và B. Phần tiếp thấp ở trong A, còn phần cao ở trong B. Ví dụ dưới đây trình bày phép nhân 25H với 65H. Kết quả là dữ liệu 16 bit được đặt trong A và B.

```
MOV   A, #25H    ; Nạp vào A giá trị 25H
MOV   B, 65H     ; Nạp vào B giá trị 65H
MUL   AB         ; 25H*65H = E99 với B = 0EH và A = 99H
```

Bảng 6.1: Tóm tắt phép nhân hai số không dấu (MULAB)

Nhân	Toán hạng 1	Toán hạng 2	Kết quả
Byte*Byte	A	B	A = byte thấp, B = byte cao

6.2.2 Chia hai số không dấu.

8051 cùng chỉ hỗ trợ phép chia hai số không dấu byte cho byte với cú pháp:

DIV AB ; Chia A cho B

Khi chia một byte cho một byte thì tử số (số bị chia) phải ở trong thanh ghi A và mẫu số (số chia) phải ở trong thanh ghi B. Sau khi lệnh chia DIV được thực hiện thì thương số được đặt trong A, còn số dư được đặt trong B. Xét ví dụ dưới đây:

```
MOV     A, #95      ; Nạp số bị chia vào A = 95
MOV     B, #10      ; Nạp số chia vào B = 10
DIV     AB          ; A = 09 (thương số); B = 05 (số dư)
```

Lưu ý các điểm sau khi thực hiện "DIV AB"

Lệnh này luôn bắt CY = 0 và OV = 0 nếu tử số không phải là số 0

Nếu tử số là số 0 (B = 0) thì OV =1 báo lỗi và CY = 0. Thực tế chuẩn trong tất cả mọi bộ vi xử lý khi chia một số cho 0 là bằng cách nào đó báo có kết quả không xác định. Trong 8051 thì cờ OV được thiết lập lên 1.

Bảng 6.2: Tóm tắt phép chia không dấu (DIV AB).

Phép chia	Tử số	Mẫu số	Thương số	Số dư
Byte cho Byte	A	B	A	B

6.2.3 Một ứng dụng cho các lệnh chia.

Có những thời điểm khi một bộ ADC được nối tới một cổng và ADC biểu diễn một số dư nhiệt độ hay áp suất. Bộ ADC cấp dữ liệu 8 bit ở dạng Hex trong dải 00 - FFH. Dữ liệu Hex này phải được chuyển đổi về dạng thập phân. Chúng ta thực hiện chia lặp nhiều lần cho 10 và lưu số dư vào như ở ví dụ 6.8.

Ví dụ 6.8:

a- Viết một chương trình để nhận dữ liệu dạng Hex trong phạm vi 00 - FFH từ cổng 1 và chuyển đổi nó về dạng thập phân. Lưu các số vào trong các thanh ghi R7, R6 và R5 trong đó số có nghĩa nhỏ nhất được cất trong R7.

b- Phân tích chương trình với giả thiết P1 có giá trị FDH cho dữ liệu.

Lời giải:

a)

```
MOV     A, #0FFH
MOV     P1, A       ; Tạo P1 là cổng đầu vào
MOV     A, P1       ; Đọc dữ liệu từ P1
MOV     B, #10      ; B = 0A Hex (10 thập phân)
DIV     AB          ; Chia cho 10
MOV     R7, B       ; Cất số thấp
MOV     B, #10      ;
DIV     AB          ; Chia 10 lần nữa
MOV     R6, B       ; Cất số tiếp theo
MOV     R5, A       ; Cất số cuối cùng
```

b) Để chuyển đổi số nhị phân hay Hex về số thập phân ta thực hiện chia lặp cho 10 liên tục cho đến khi thương số nhỏ hơn 10. Sau mỗi lần chia số dư được lưu cất.

Trong trường hợp một số nhị phân 8 bit như FDH chẳng hạn ta có 253 số thập phân như sau (tất cả trong dạng Hex)

	Thương số	**Số dư**	
FD/0A	19	3	(Số thấp - cuối)
19/0A	2	5	(Số giữa)
		2	(Số đầu)

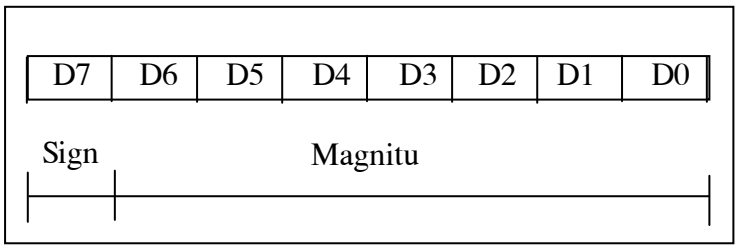
Do vậy, ta có FDH = 253. Để hiển thị dữ liệu này thì nó phải được chuyển đổi về ASCII mà sẽ được mô tả ở chương sau.

6.3 Các khái niệm về số có dấu và các phép tính số học.

Tất cả mọi dữ liệu từ trước đến giờ đều là các số không dấu, có nghĩa là toàn bộ toán hạng 8 bit đều được dùng cho bộ lớn. Có nhiều ứng dụng yêu cầu dữ liệu có dấu, phần này sẽ bàn về những lệnh liên quan đến các số có dấu.

6.3.1 Khái niệm về các số có dấu trong máy tính.

Trong cuộc sống hàng ngày các số được dùng có thể là số âm hoặc dương. Ví dụ 5 độ dưới 0^0C được biểu diễn là -5^0C và 20 độ trên 0^0C được biểu diễn là $+20^0$C. Các máy tính cũng phải có khả năng đáp ứng phù hợp với các số ấy. Để làm được điều ấy các nhà khoa học máy tính đã phát minh ra sự xắp xếp biểu diễn các số âm có dấu và số dương có dấu như sau: Bit cao nhất MSB được để dành cho bit dấu (+) hoặc (-), còn các bit còn lại được dùng biểu diễn độ lớn. Dấu được biểu diễn bởi 0 đối với các số dương và một số đối với các số âm (-). Biểu diễn của một byte có dấu được trình bày trên hình 6.2.



Hình 6.2: Các toán hạng 8 bit có dấu.

a- Các toán hạng 8 bit có dấu: Trong các toán hạng A byte có dấu thì bit cao nhất MSB là D7 được dùng để biểu diễn dấu, còn 7 bit còn lại từ D6 - D0 dùng để biểu diễn độ lớn của số đó. Nếu D7 = 0 thì đó là toán hạng dương và nếu D7 = 1 thì nó là toán hạng âm.

b- Các số dương: Dải của các số dương có thể được biểu diễn theo dạng cho trên hình 6.2 là từ 0 đến +127 thì phải sử dụng toán hạng 16 bit. Vì 8051 không hỗ trợ dữ liệu 16 bit nên ta không bàn luận đến.

c- Các số âm: Đối với các số âm thì D7 = 1, tuy nhiên độ lớn được biểu diễn ở dạng số bù 2 của nó. Mặc dù hợp ngữ thực hiện việc chuyển đổi song điều quan trọng là hiểu việc chuyển đổi diễn ra như thế nào. Để chuyển đổi về dạng biểu diễn số âm (bù 2) thì tiến hành theo các bước sau:

1. Viết độ lớn của số ở dạng nhị phân 8 bit (không dấu).
2. Đảo ngược tất cả các bit
3. Cộng 1 vào nó.

Ví dụ 6.9: Hãy trình bày cách 8051 biểu diễn số - 5.

Lời giải:

Hãy quan sát các bước sau:

0000	0101	Biểu diễn số 5 ở dạng 8 bit nhị phân
1111	1010	Đảo các bit
1111	1011	Cộng (thành số FB ở dạng Hex)

Do vậy, số FBH là biểu diễn số có dấu dạng bù 2 của số - 5.

Ví dụ 6.10: Trình bày cách 8051 biểu diễn - 34H.

Lời giải:

Hãy quan sát các bước sau:

0011	0200	Số 34 được cho ở dạng nhị phân
1100	1011	Đảo các bit
1100	1100	Cộng 1 (thành số CC ở dạng Hex)

Vậy số CCH là biểu diễn dạng bù 2 có dấu của - 34H.

Ví dụ 6.11: Trình bày cách 8051 biểu diễn - 128:

Lời giải:

Quan sát các bước sau:

1000	0000	Số 128 ở dạng nhị phân 28 bit
0111	1111	Đảo các bit
1000	0000	Cộng 1 (trở thành số 80 dạng Hex)

Vậy - 128 = 80H là biểu diễn số có dấu dạng bù 2 của - 128.

Từ các ví dụ trên đây ta thấy rõ ràng rằng dải của các số âm có dấu 8 bit là - 1 đến - 128. Dưới đây là liệt kê các số có dấu 8 bit:

Số thập phân	Số nhị phân	Số Hex
-128	1000 0000	80
-127	1000 0001	81
-126	1000 0010	82
...	.....	...
-2	1111 1110	FE
-1	1111 1111	FF
0	0000 0000	00
+1	0000 0001	01
+2	0000 0010	02
...	.....	...
-127	0111 1111	FE

6.3.2 Vấn đề tràn trong các phép toán với số có dấu.

Khi sử dụng các số có dấu xuất hiện một vấn đề rất nghiêm trọng mà phải được sử lý. Đó là vấn đề tràn, 8051 báo có lỗi bằng cách thiết lập cờ tràn OV nhưng trách nhiệm của lập trình viên là phải cẩn thận với kết quả sai. CPU chỉ hiểu 0 và 1 và nó làm ngơ với việc chuyển đổi số âm, số dương của con người. Vậy tràn số là gì? Nếu kết quả của một phép toán trên các số có dấu mà quá lớn đối với thanh ghi thì xuất hiện sự tràn số và lập trình viên phải được cảnh báo. Xét ví dụ 6.12 dưới đây.

Ví dụ 6.12:

Khảo sát đoạn mã sau và phân tích kết quả.

```
MOV     A, # + 96       ; A = 0110      0000 (A = 60H)
MOV     R1, # + 70      ; R1 = 0100     0110 (R1 = 46H)
ADD     A, R1           ; A = 1010      0110 = A6H = - 90
                          Sai !!!
```

Lời giải:

```
    + 96                0110    0000
+   + 70                0100    0110
  - 166                 1010    0110        và OV = 1
```

Theo CPU kết quả là -90 và đó là kết quả sai nên CPU bật cờ OV = 1 để báo tràn số.

Trong ví dụ 6.12 thì + 96 được cộng với + 70 và kết quả theo CPU là - 90. Tại sao vậy? Lý do là kết quả của + 96 + 70 = 172 lớn hơn số mà thanh ghi A có thể chứa được. Cũng như tất cả mọi thanh ghi 8 bit khác, thanh ghi A chỉ chứa được đến số + 127. Các nhà thiết kế của PCU tạo ra cờ tràn OV phục vụ riêng cho mục đích báo cho lập trình viên rằng kết quả của phép toán số có dấu là sai.

6.3.3 Khi nào thì cờ tràn OV được thiết lập?

Trong các phép toán với số có dấu 8 bit thì cờ OV được bật lên 1 khi xuất hiện một trong hai điều kiện sau:

1. Cờ nhớ từ D6 sang D7 nhưng không có nhớ ra từ D7 (cờ CY = 0)
2. Có nhớ ra từ D7 (cờ CY = 1) nhưng không có nhớ từ D6 sang D7

Hay nói cách khác là cờ tràn OV được bật lên 1 nếu có nhớ từ D6 sang D7 hoặc từ D7 nhưng không đồng thời xảy ra cả hai. Điều này có nghĩa là nếu có nhớ cả từ D6 sang D7 và từ D7 ra thì cờ OV = 0. Trong ví dụ 6.12 vì chỉ có nhớ từ D7 ra nên cờ OV = 1. Trong ví dụ 6.13, ví dụ 6.14 và 6.15 có minh hoạ thêm về sử dụng cờ tràn trong các phép số học với số có dấu.

Ví dụ 6.13:

Hãy quan sát đoạn mã sau để ý đến vai trò của cờ OV.

```
MOV     A, # -128       ; A = 1000      0000 (A= 80H)
MOV     R4, # -2        ; R4 = 1111     (R4 = FEH)
ADD     A, R4           ; A = 0111      1110 (A = 7EH = +126, invalid)
```

Lời giải:

```
- 128           1000    0000
+ - 2           1111    1110
-130            0111    1110    và OV = 1
```

Theo CPU thì kết quả + 126 là kết quả sai, nên cờ OV = 1.

Ví dụ 6.14:

Hãy quan sát đoạn mã sau và lưu ý cờ OV.

```
MOV     A, # -2      ; A = 1111    1110 (A = FEH)
MOV     R1, # -5     ; R1 = 1111   1011   (R1 = FBH)
ADD     A, R1        ; A = 1111    1001   (A = F9H = -7, correct, OV = 0)
```

Lời giải:

```
  - 2        1111   1110
+ - 5        1111   1011
-   7        1111   1001   và OV = 0
```

Theo CPU thì kết quả - 7 là đúng nên cờ OV = 0.

Ví dụ 6.15:

Theo dõi đoạn mã sau, chú ý vai trò của cờ OV.

```
MOV   A, # +7      ; A =  0000    0111 (A = 07H)
MOV   R1, # +18    ; R1 = 0001    0010 (R1 = 12H)
ADD   A, R1        ; A = 1111     1001   (A = 19H = -25, correct, OV = 0)
```

Lời giải:

```
    7      0000   0111
- 18       0001   0010
  25       0001   1001   và OV = 0
```

Theo CPU thì kết quả - 25 là đúng nên cờ OV = 0.

Từ các ví dụ trên đây ta có thể kết luận rằng trọng bất kỳ phép cộng số có dấu nào, cờ OV đều báo kết quả là đúng hay sai. Nếu cờ OV = 1 thì kết quả là sai, còn nếu OV = 0 thì kết quả là đúng. Chúng ta có thể nhấn mạnh rằng, trong phép cộng các số không dấu ta phải hiển thị trạng thái của cờ CY (cờ nhớ) và trong phép cộng các số có dấu thì cờ tràn OV phải được theo dõi bởi lập trình viên. Trong 8051 thì các lệnh như JNC và JC cho phép chương trình rẽ nhánh ngay sau phép cộng các số không dấu như ở phần 6.1. Đối với cờ tràn OV thì không có như vậy. Tuy nhiên, điều này có thể đạt được bằng lệnh "JB PSW.2" hoặc "JNB PSW.2" vì PSW thanh ghi cờ có thể đánh địa chỉ theo bit.

CHƯƠNG 7
Các lệnh lôgíc và các chương trình

7.1 Các lệnh lô-gíc và so sánh.

7.1.1 Lệnh VÀ (AND).

Cú pháp: ANL đích, nguồn; đích = đích Và nguồn (kẻ bảng).

Lệnh này sẽ thực hiện một phép Và lô-gíc trên hai toán hạng đích và nguồn và đặt kết quả vào đích. Đích thường là thanh ghi tổng (tích luỹ). Toán hạngnguồn có thể là thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. Hãy xem phụ lục Appendix A1 để biết thêm về các chế độ đánh địa chỉ dành cho lệnh này. Lệnh ANL đối với toán hạng theo byte không có tác động lên các cờ. Nó thường được dùng để che (đặt về 0) những bit nhất định của một toán hạng. Xem ví dụ 7.1.

Ví dụ:

Trình bày kết quả của các lệnh sau:

```
MOV    A, #35H          ; Gán A = 35H
ANL    A, #0FH          ; Thực hiện Và lô-gíc A và 0FH (Bây giờ A = 05)
```

Lời giải:

```
35H    0  0  1  1  0  1  0  1
0FH    0  0  0  0  1  1  1  1
05H    0  0  0  0  0  1  0  1          35H  và 0FH = 05H
```

7.1.2: Lệnh HOẶC (OR).

Cú pháp ORL đích = đích Hoặc nguồn (kẻ bảng)

Các toán hạng đích và nguồn được Hoặc với nhau và kết quả được đặt vào đích. Phép Hoặc có thể được dùng để thiết lập những bit nhất định của một toán hạng 1. Đích thường là thanh ghi tổng, toán hạng nguồn có thể là một thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. Hãy tham khảo phụ lục Appendix A để biết thêm về các chế độ đánh địa chỉ được hỗ trợ bởi lệnh này. Lệnh ORL đối với các toán hạng đánh địa chỉ theo byte sẽ không có tác động đến bất kỳ cờ nào. Xem ví dụ 7.2.

Ví dụ 7.2: Trình bày kết quả của đoạn mã sau:

```
MOV    A, #04           ; A = 04
MOV    A, #68H ; A = 6C
```

Lời giải:

```
04H    0000  0100
68H    0110  1000
6CH    0110  1100              04  OR  68 = 6CH
```

7.1.3 Lệnh XOR (OR loại trừ?).

Cú pháp: XRL đích, nguồn; đích = đích Hoặc loại trừ nguồn (kẻ bảng).

Lệnh này sẽ thực hiện phép XOR trên hai toán hạng và đặt kết quả vào đích. Đích thường là thanh ghi tổng. Toán hạng nguồn có thể là một thanh ghi trong bộ nhớ hoặc giá trị cho sẵn. Xem phụ lục Appendix A.1 để biết thêm về chế độ đánh địa chỉ của lệnh này. Lệnh XRL đối với các toán hạng đánh địa chỉ theo byte sẽ không có tác động đến bất kỳ cờ nào. Xét ví dụ 7.3 và 7.4.

Ví dụ 7.3: Trình bày kết quả của đoạn mã sau:

```
MOV    A, #54H
XRL    A, #78H
```

Lời giải:

```
54H    0  1  0  1  0  1  0  0
78H    0  1  1  1  1  0  0  0
2CH    0  0  1  0  1  1  0  1          54H  XOR  78H = 2CH
```

Ví dụ 7.4:

Lệnh XRL có thể được dùng để xoá nội dung của một thanh ghi bằng cách XOR nó với chính nó. Trình bày lệnh "XRL A, A" xoá nội dung của A như thế nào? giả thiết AH = 45H.

Lời giải:

```
45H            01000101
45H            01000101
00             00000000          54H  XOR  78H = 2CH
```

Lệnh XRL cũng có thể được dùng để xem nếu hai thanh ghi có giá trị giống nhau không? Lệnh "XRL A, R1" sẽ hoặc loại trừ với thanh ghi R1 và đặt kết quả vào A. Nếu cả hai thanh ghi có cùng giá trị thì trong A sẽ là 00. Sau đó có thể dùng lệnh nhả JZ để thực hiện theo kết quả. Xét ví dụ 7.5.

Ví dụ 7.5:

Đọc và kiểm tra cổng P1 xem nó có chứa giá trị 45H không? Nếu có gửi 99H đến cổng P2, nếu không xoá nó.

Lời giải:

```
        MOV    P2, #00          ; Xóa P2
        MOV    P1, #0FFH        ; Lấy P1 là cổng đầu vào
        MOV    R3, #45H         ; R3 = 45H
        MOV    A, P1            ; Đọc P1
        XRL    A, R3
        JNZ    EXIT             ; Nhảy nếu A có giá trị khác 0
        MOV    P2, #99H
EXIT: ...
```

Trong chương trình của ví dụ 7.5 lưu ý việc sử dụng lệnh nhảy JNZ. Lệnh JNZ và JZ kiểm tra các nội dung chỉ của thanh ghi tổng. Hay nói cách khác là trong 8051không có cờ 0.

Một ứng dụng rộng rãi khác của bộ xử lý là chọn các bit của một toán hạng. Ví dụ để chọn 2 bit của thnh ghi A ta có thể sử dụng mã sau. Mã này ép bit D2 của thanh ghi A chuyển sang giá trị nghịch đảo, còn các bit khác không thay đổi.

```
XRL     A, #04H         ; Nghĩa hoặc loại trừ thanh ghi A với
                        ; Giá trị 0000 0100
```

7.1.4 Lệnh bù thanh ghi tổng CPL A.

Lệnh này bù nội dung của thanh ghi tổng A. Phép bù là phép biến đổi các số 0 thành các số 1 và đổi các số 1 sang số 0. Đây cũng còn được gọi là phép bù 1.

```
MOV         A, #55H
CPL         A           ; Bây giờ nội dung của thanh ghi A là AAH
                        ; Vì 0101 0101 (55H) → 1010 1010 (AAH)
```

Để nhận được kết quả bù 2 thì tất cả mọi việc ta cần phải làm là cộng 1 vào kết quả bù 1. Trong 8051 thì không có lệnh bù 2 nào cả. Lưu ý rằng trong khi bù một byte thì dữ liệu phải ở trong thanh ghi A. Lệnh CPL không hỗ trợ một chế độ đánh địa chỉ nào cả. Xem ví dụ 7.6 dưới đây.

Ví dụ 7.6: Tìm giá trị bù 2 của 85H.

Lời giải:

```
MOV         A, #85H     ; Nạp 85H vào A (85H = 1000 0101)
MOV         A           ; Lấy bù 1 của A (kết quả = 0111 1010)
ADD         A, #1       ; Cổng 1 vào A thành bù 2 A = 0111 1011 (7BH)
```

Ví dụ 7.1.5 Lệnh so sánh.

8051 có một lệnh cho phép so sánh. Nó có cú pháp như sau:

CJNE đích, nguồn, địa chỉ tương đối.

Trong 8051 thì phép so sánh và nhảy được kết hợp thành môt lệnh có tên là CJNE (so sánh và nhảy nếu kết quả không bằng nhau). Lệnh CJNE so sánh hai toán hạng nguồn và đích và nhảy đến địa chỉ tương đối nếu hai toán hạng không bằng nhau. Ngoài ra nó thay đổi cờ nhớ CY để báo nếu toán hạng đích lớn hơn hay nhỏ hơn. Điều quan trọng cần để là các toán hạng vẫn không giữ nguyên không thay thay đổi. Ví dụ, sau khi thực hiện lệnh "CJNE A, #67H, NEXT" thì thanh ghi A vẫn có giá trị ban đầu của nó (giá trị trước lệnh CJNE). Lệnh này so sánh nội dung thanh ghi A với giá trị 67H và nhảy đến giá trị đích NEXT chỉ khi thanh ghi A có giá trị khác 67H.

Ví dụ 7.7:

Xét đoạn mã dưới đây sau đó trả lời câu hỏi:

a) Nó sẽ nhảy đến NEXT không?

b) Trong A có giá trị bao nhiêu sau lệnh CJNE?

```
        MOV         A, #55H
        CJNE        A, #99H, NEXT
        ...
NEXT: ...
```

Lời giải:

a) Có vì 55H và 99H không bằng nhau

b) A = 55H đây là giá trị trước khi thực hiện CJNE.

Trong lệnh CJNE thì toán hạng đích có thể trong thanh ghi tổng hoặc trong một các thanh ghi Rn. Toán hạng nguồn có thể trong một thanh ghi, trong bộ nhớ hoặc giá trị cho sẵn. Hãy xem phụ lục Appendix A để biết thêm chi tiết về các chế độ đánh địa chỉ cho lệnh này. Lệnh này chỉ tác động cờ nhớ CY. Cờ này được thay đổi như chỉ ra trên bảng 7.1. Dưới đây trình bày phép so sánh hoạt động như thế nào đối với tất cả các điều kiện có thể:

```
            CJNE   R5, #80, NOT-EQUAL      ; Kiểm tra R5 có giá trị 80?
            ...                            ; R5 = 80
NOT-EQUAL:  JNC    NEXT                    ; Nhảy đến R5 > 80
            ...
NEXT:        ...
```

Bảng 7.1: Thiết kế cờ CY cho lệnh CJNE.

Compare	Carry Flag
Destinatio > Source	CY = 0
Destination < Source	CY = 1

Để ý rằng trong lệnh CJNE thì không có thanh ghi Rn nào có thể được so sánh với giá trị cho sẵn. Do vậy không cần phải nói đến thanh ghi A. Cũng cần lưu ý rằng cờ nhớ CY luôn được kiểm tra để xem lớn hơn hay nhỏ hơn, nhưng chỉ khi đã xác định là nó không bằng nhau. Xét ví dụ 7.8 và 7.9 dưới đây.

Ví dụ 7.8:

Hãy viết mã xác định xem thanh ghi A có chứa giá trị 99H không? Nếu có thì hãy tạo R1 = FFH còn nếu không tạo R1 = 0.

Lời giải:

```
            MOV    R1, #0          ; Xoá R1
            CJNE   A, #99H ; Nếu A không bằng 99H thì nhảy đến NEXT
            MOV    R1, #0FFH       ; Nếu chúng bằng nhau, gán R1 = 0FFH
NEXT: ...                          ; Nếu không bằng nhau, gán R1 = 0
OVER: ...
```

Ví dụ 7.9:

Giả sử P1 là một cổng đầu vào được nối tới một cảm biến nhiệt. Hãy viết chương trình đọc nhiệt độ và kiểm tra nó đối với giá trị 75. Theo kết quả kiểm tra hãy đặt giá trị nhiệt độ vào các thanh ghi được chỉ định như sau:

```
Nếu T = 75      thì A = 75
Nếu T < 75      thì R1 = T
Nếu T > 75      thì R2 = T
```

Lời giải:

```
        MOV   P1, 0FFH        ; Tạo P1 làm cổng đầu vào
        MOV   A, P1           ; Đọc cổng P1, nhiệt độ
        CJNE  A, #75, OVER    ; Nhảy đến OVER nếu A ≠ 75
        SJMP  EXIT            ; A = 75 thoát
OVER:   JNC   NEXT            ; Nếu CY = 0 thì A > 75 nhảy đến NEXT
        MOV   R1, A           ; Nếu CY = 1 thì A < 75 lưu vào R1
        SJMP  EXIT            ; Và thoát
NEXT:   MOV   R2, A           ; A > 75 lưu nó vào R2
EXIT:   ...
```

Lệnh so sánh thực sự là một phép trừ, ngoại trừ một điều là giá trị của các toán hạng không thay đổi. Các cờ được thay đổi tuỳ theo việc thực hiện lệnh trừ SUBB. Cần phải được nhấn mạnh lại rằng, trong lệnh CJNE các toán hạng không bị tác động bất kể kết quả so sánh là như thế nào. Chỉ có cờ CY là bị tác động, điều này bị chi phối bởi thực tế là lệnh CJNE sử dụng phép trừ để bật và xoá cờ CY.

Ví dụ 7.10:

Viết một chương trình để hiển thị liên tục cổng P1 đối với giá trị 63H. Nó chỉ mất hiển thị khi P1 = 63H.

Lời giải:

```
        MOV   P1, #0FFH       ; Chọn P1 làm cổng đầu vào
HERE:   MOV   A, P1           ; Lấy nội dung của P1
        CJNE  A, #63, HERE    ; Duy trì hiển thị trừ khi P1 = 63H
```

Ví dụ 7.11:

Giả sử các ngăn nhớ của RAM trong 40H - 44H chứa nhiệt độ hàng ngày của 5 ngày như được chỉ ra dưới đây. Hãy tìm để xem có giá trị nào bằng 65 không? Nếu giá trị 65 có trong bảng hãy đặt ngăn nhớ của nó vào R4 nếu không thì đặt R4 = 0.

40H = (76); 41H = (79); 42H = (69); 43H = (65); 44H = (64)

Lời giải:

```
        MOV   R4, #0          ; Xoá R4 = 0
        MOV   R0, #40H        ; Nạp con trỏ
        MOV   R2, #05         ; Nạp bộ đếm
        MOV   A, #65          ; Gán giá trị cần tìm vào A
BACK:   CJNE  A, @R0, NEXT    ; So sánh dữ liệu RAM với 65
        MOV   R4, R0          ; Nếu là 65, lưu địa chỉ vào R4
        SJMP  EXIT            ; Thoát
NEXT:   INC   R0              ; Nếu không tăng bộ đếm
        DJNZ  R2, BACK        ; Tiếp tục kiểm tra cho đến khi bộ đếm bằng 0.
EXIT:   ...
```

7.2 Các lệnh quay vào trao đổi.

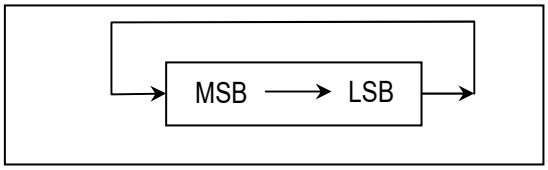
Trong rất nhiều ứng dụng cần phải thực hiện phép quay bit của một toán hạng. Các lệnh quay 8051 là R1, RR, RLC và RRC được thiết kế đặc biệt cho mục đích này. Chúng cho phép một chương trình quay thanh ghi tổng sang trái hoặc phải. Trong 8051 để quay một byte thì toán hạng phải ở trong thanh ghi tổng A. Có hai kiểu quay là: Quay đơn giản các bit của thanh ghi A và quay qua cờ nhớ (hay quay có nhớ).

7.2.1 Quay các bit của thanh ghi A sang trái hoặc phải.

a) Quay phải: RR A ; Quay các bit thanh ghi A sang phải.

Trong phép quay phải, 8 bit của thanh ghi tổng được quay sang phải một bit và bit D0 rời từ vị trí bit thấp nhất và chuyển sang bit cao nhất D7. Xem đoạn mã dưới đây.

```
MOV   A, #36H      ; A = 0011 0110
RR    A            ; A = 0001 1011
RR    A            ; A = 1000 1101
RR    A            ; A = 1100 0110
RR    A            ; A = 0110 0011
```



b) Quay trái:

Cú pháp: RL A ; Quay trái các bit của thanh ghi A (hình vẽ)

Trong phép quay trái thì 8 bit của thanh ghi A được quay sang trái 1 bit và bit D7 rời khỏi vị trí bit cao nhất chuyển sang vị trí bit thấp nhất D0. Xem biểu đồ mã dưới đây.

```
MOV   A, #72H      ; A = 0111 0010
RL    A            ; A = 1110 0100
RL    A            ; A = 1100 1001
```

Lưu ý rằng trong các lệnh RR và RL thì không có cờ nào bị tác động.

7.2.2 Quay có nhớ.

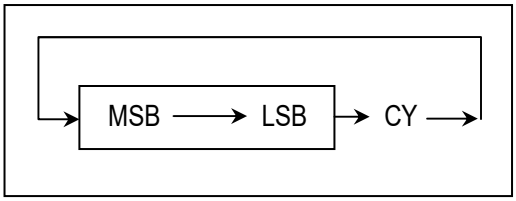
Trong 8051 còn có 2 kênh quay nữa là quay phải có nhớ và quay trái có nhớ.

Cú pháp: RRC A và RLC A

a) Quay phải có nhớ: RRC A

Trong quay phải có nhớ thì các bit của thanh ghi A được quay từ trái sang phải 1 bit và bit thấp nhất được đưa vào cờ nhớ CY và sau đó cờ CY được đưa vào vị trí bit cao nhất. Hay nói cách khác, trong phép RRC A thì LSB được chuyển vào CY và CY được chuyển vào MSB. Trong thực tế thì cờ nhớ CY tác động như là một bit bộ phận của thanh ghi A làm nó trở thành thanh ghi 9 bit.

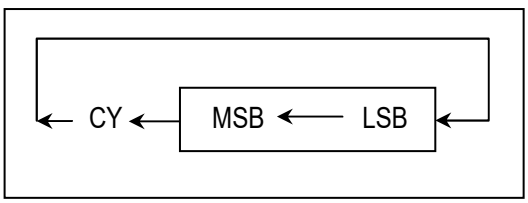
```
CLR   C            ; make CY = 0
MOV   A #26H       ; A = 0010 0110
RRC   A            ; A = 0001 0011  CY = 0
RRC   A            ; A = 0000 1001  CY = 1
RCC   A            ; A = 1000 0100  CY = 1
```



b) Quay trái có nhớ (hình vẽ): RLC A.

Trong RLC A thì các bit được dịch phải một bit và đẩy bit MSB vào cờ nhớ CY, sau đó CY được chuyển vào bit LSB. Hay nói cách khác, trong RLC thì bit MSB được chuyển vào CY và CY được chuyển vào LSB. Hãy xem đoạn mã sau.

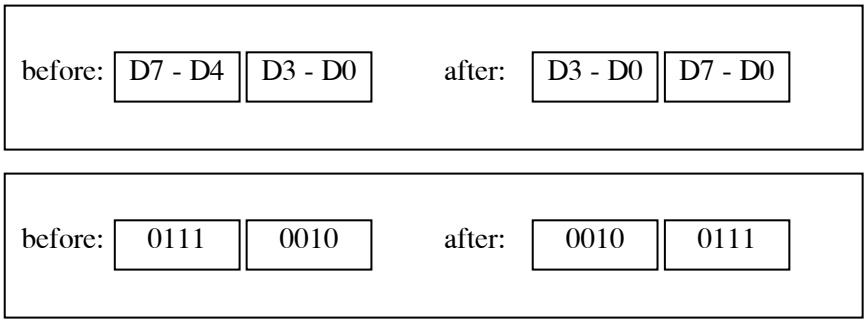
```
SETB  C            ; Make CY = 1
MOV   A #15H       ; A = 0001 0101
RRC   A            ; A = 0101 1011  CY = 0
RRC   A            ; A = 0101 0110  CY = 0
```



```
RCC     A               ; A = 1010 1100  CY = 0
RCC     A               ; A = 1000 1000  CY = 1
```

7.2.3 Lệnh trao đổi thanh ghi A: SWAP A

Một lệnh hữu ích khác nữa là lệnh trao đổi SWAP. Nó chỉ hoạt động trên thanh ghi A, nó trao đổi nửa phần cao của byte và nửa phần thấp của byte với nhau. Hay nói cách khác 4 bit cao được chuyển thành 4 bit thấp và 4 bit thấp thành 4 bit cao.



Ví dụ 7.12:

a) Hãy tìm nội dung của thanh ghi A ở đoạn mã sau.
b) Trong trường hợp không có lệnh SWAP thì cần phải làm như thế nào để trao đổi những bit này? Hãy viết một mã chương trình đơn giản về quá trình đó.

Lời giải:

a)

```
        MOV     A, #72H ; A = 72H
        SWAP    A               ; A = 27H
```

b)

```
        MOV     A, #72H ; A = 0111 0010
        RL      A               ; A = 1110 0100
        RL      A               ; A = 1100 1001
        RL      A               ; A = 0010 0111
```

Ví dụ 7.13:

Viết một chương trình để tìm số các số 1 trong một byte đã cho.

Lời giải:

```
        MOV     R1, #0          ; Chọn R1 giữ số các số 1
        MOV     R7, #8          ; Đặt bộ đếm  = 8 để quay 8 lần
        MOV     A, #97H         ; Tim các số 1 trong byte 97H
AGAIN:  RLC     A               ; Quay trái có nhớ một lần
        JNC     NEXT            ; Kiểm tra cờ CY
        INC     R1              ; Nếu CY = 1 thì cộng 1 vào bộ đếm
NEXT:   DJNZ    R7, AGAIN       ; Lặp lại quá trình 8 lần
```

Để truyền 1 byte dữ liệu nói tiếp thì dữ liệu có thể được chuyển đổi từ song song sang nối tiếp bằng các lệnh quay như sau:

```
RRC     A               ; Bít thứ nhất đưa vào cờ CY
MOV     P1.3, C         ; Xuất CY như một bit dữ liệu
RRC     A               ; Bit thứ hai đưa vào CY
```

```
MOV     P1.3, C         ; Xuất CY ra như một bit dữ liệu
RRC     A               ;
MOV     P1.3, C         ;
...
```

Đoạn mã trên đây là một phương pháp được sử dụng rộng rãi trong truyền dữ liệu tới các bộ nhớ nối tiếp như các EEPROM nối tiếp.

7.3 Các chương trình ứng dụng của mã BCD và ASCII.

Các số mã BCD đã được trình ở chương 6. Như đã nói ở đó rằng trong rất nhiều bộ vi điều khiển mới đều có một đồng hồ thời gian thực RTC (Real Time Clock) để giữ cho thời gian và cả lịch cho cả khi bị tắt nguồn. Các bộ vi điều khiển này cung cấp thời gian và lịch dưới dạng BCD. Tuy nhiên, để hiển thị chúng thì chúng phải được chuyển về mã ASCII. Trong phần này ta trình bày ứng dụng của các lệnh quay và các lệnh lô-gíc trong việc chuyển đổi mã BCD và ASCII.

Bảng 7.2: Mã ASCII cho các chữ số từ 0- 9.

Phím	Mã ASCII (Hex)	Mã ASCII nhị phân	Mã BCD (không đóng gói)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

7.3.1 Các số mã ASCII.

Trên các bàn phím ASCII khi phím "0" được kích hoạt thì "011 0001" (30H) được cấp tới máy tính. Tương tự như vậy 31H (011 0001) được cấp cho phím "1" v.v... như cyhỉ ra trong bảng 7.2.

Cần phải ghi nhớ rằng mặc dù mã ASCII là chuẩn ở mỹ (và nhiều quốc gia khác) nhưng các số mã BCD là tổng quát. Vì bàn phìm, máy in và màn hình đều sử dụng mã ASCII nên cần phải thực hiện đổi chuyển giữa các số mã ASCII về số mã BCD và ngược lại.

7.3.2 Chuyển đổi mã BCD đóng gói về ASCII.

Các bộ vi điều khiển DS5000T đều có đồng bộ thời gian thực RTC. Nó cung cấp hiển thị liên tục thời gian trong ngày (giờ, phút và giây) và lịch (năm, tháng, ngày) mà không quan tâm đến nguồn tắt hay bật. Tuy nhiên dữ liệu này được cấp ở dạng mã BCD đóng gói. Để hiển thị dữ liệu này trên một LCD hoặc in ra trên máy in thì nó phải được chuyển về dạng mã ASCII.

Để chuyển đổi mã BCD đóng gói về mã ASCII thì trước hết nó phải được chuyển đổi thành mã BCD không đóng gói. Sau đó mã BCD chưa đóng gói được móc với 011 0000 (30H). Dưới đây minh hoạ việc chuyển đổi từ mã BCD đóng gói về mã ASCII. Xem ví dụ 7.14.

Mã BCD đóng gói	Mã BCD không đóng gói	Mã ASCII
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 & 0000 1001	0011 0010 & 0011 1001

7.3.3 Chuyển đổi mã ASCII về mã BCD đóng gói.

Để chuyển đổi mã ASCII về BCD đóng gói trước thì trước hết nó phải được chuyển về mã BCD không đóng gói (để có thêm 3 số) và sau đó được kết hợp để tạo ra mã SCD đóng gói. Ví dụ số 4 và số 7 thì bàn phím nhận được 34 và 37. Mục tiêu là tạo ra số 47H hay "0100 0111" là mã BCD đóng gói. Qúa trình này như sau:

Phím	Mã ASCII	Mã BCD không đóng gói	Mã BCD đóng gói
4	34	0000 0100	
7	37	0000 0111	0100 0111 hay 47H

```
MOV     A, # '4'        ; Gán A = 34H mã ASCII của số 4
MOV     R1, # '7'       ; Gán R1 = 37H mã ASCII của số 4
ANL     A, #0FH         ; Che nửa byte cao A (A = 04)
ANL     R1, #0FH        ; Che nửa byte cao của R1 (R1 = 07)
SWAP    A               ; A = 40H
ORL     A, R1           ; A = 47H, mã BCD đóng gói
```

Sau phép chuyển đổi này các số BCD đóng gói được xử lý và kết quả sẽ là dạng BCD đóng gói. Như ta đã biết ở chương 6 có một lệnh đặt biệt là "DA A" đòi hỏi dữ liệu phải ở dạng BCD đóng gói.

Ví dụ 7.14:

Giả sử thanh ghi A có số mã BCD đóng gói hãy viết một chương trình để chuyển đổi mã BCD về hai số ASCII và đặt chúng vào R2 và R6.

Lời giải:

```
MOV     A, #29H         ; Gán A = 29, mã BCD đóng gói
MOV     R2, A           ; Giữ một bản sao của BCD trong R2
ANL     A, #0FH         ; Che phần nửa cao của A (A = 09)
ORL     A, #30H         ; Tạo nó thành mã ASCII A = 39H (số 9)
MOV     R6, A           ; Lưu nó vào R6 (R6 = 39H ký tự của ASCII)
MOV     A, R2           ; Lấy lại giá trị ban đầu của A (A = 29H)
ANL     A, #0F0H        ; Che nửa byte phần thấp của A (A = 20)
RR      A               ; Quay phải
RR      A               ; Quay phải
RR      A               ; Quay phải
RR      A               ; Quay phải (A = 02)
ORL     A, #30H         ; Tạo nó thành mã ASCII (A = 32H, số 2)
MOV     R2, A           ; Lưu ký tự ASCII vào R2
```

Trong ví dụ trên tất nhiên là ta có thể thay 4 lệnh RR quay phải bằng một lệnh trao đổi WAPA.

CHƯƠNG 8
Các lệnh một bít và lập trình

8.1 Lập trình với các lệnh một bít.

Trong hầu hết các bộ vi xử lý (BVXL) thì dữ liệu được truy cập theo từng byte. Trong các bộ vi xử lýnh địa chỉ theo byte này thì các nội dung của một thanh ghi, bộ nhớ RAM hay cổng đều phải được truy cập từng byte một. Hay nói cách khác, lượng dữ liệu tối thiểu có thể được truy cập là một byte. Ví dụ, trong bộ vi xử lý Pentium cổng vào/ ra (I/O) được định hướng theo byte, có nghĩa là để thay đổi một bít thì ta phải truy cập toàn bộ 8 bít. Trong khi đó có rất nhiều ứng dụng thì ta phải chỉ cần thay đổi giá trị của một bít chẳng hạn như là bật hoặc tắt một thiết bị. Do vậy khả năng đánh địa chỉ đến từng bít của 8051 rất thích hợp cho ứng dụng này. Khả năng truy cập đến từng bít một thay vì phải truy cập cả byte làm cho 805 trở thành trong những bộ vi điều khiển (BVĐK) 8 bít mạnh nhất trên thị trường. Vậy những bộ phận nào của CPU, RAM, các thanh ghi, cổng I/O hoặc ROM là có thể đánh địa chỉ theo bít được. Vì ROM chỉ đơn giản dữ mã chương trình thực thi nên nó không cần khả năng đánh địa chỉ theo bít. Tất cả mọi mã lệnh đều định hướng theo byte chỉ có các thanh ghi, RAM và các cổng I/O là cần được đánh địa chỉ theo bít. Trong 8051 thì rất nhiều vị trí của RAM trong một số thanh ghi và tất cả các cổng I/O là có thể đánh địa chỉ theo từng bít. Dưới đây ta chỉ đi sâu vào từng phần một.

8.1.1 Các lệnh một bít.

Các lệnh dùng các phép tính một bít được cho ở bảng 8.1. Trong phần này chúng ta làm về các lệnh này và đưa ra nhiều ví dụ về cách sử dụng chúng, các lệnh một bít khác mà chỉ liên quan đến cờ nhớ CY (Cary Flag) sẽ làm ở mục khác.

Bảng 8.1: Các lệnh một bít của 8051

Lệnh	Chức năng
SETB bít	Thiết lập bít (bít bằng 1)
CLR bít	Xoá bít về không (bít = 0)
CPL bít	Bù bít (bít = NOT bít)
JB bít, đích	Nhảy về đích nếu bít = 1
JNB bít, đích	Nhảy về đích nếu bít = 0
JBC bít, đích	Nhảy về đích nếu bít = 1 và sau đó xoá bít

8.1.2 Các cổng I/O và khả năng đánh địa chỉ theo bít.

Bộ vi điều khiển 8051 có bốn cổng I/O 8 bít là P0, P1, P2 và P3. Chúng ta có thể truy cập toàn bộ 8 bít hoặc theo một bít bất kỳ mà không làm thay đổi các bít khác còn lại. Khi truy cập một cổng theo từng bít, chúng ta sử dụng các cú pháp "SETB Y, Y" với X là số của cổng 0, 1, 2 hoặc 3, còn Y là vị trí bít từ 0 đến 7 đối với các bít dữ liệu đo đến 7. Ví dụ "SETB P1.5" là thiết lập bít cao số 5 của cổng 1. Hãy nhớ rằng do là bít có nghĩa thấp nhất LSB và D7 là bít có nghĩa là cao nhất MSB. Xem ví dụ 8.1.

Ví dụ 8.1: Viết các chương trình sau:

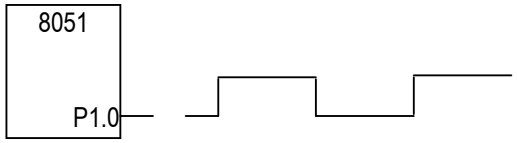
a) Tạo một sóng vuông (hàm xung vuông) với độ đầy xung 50% trên bít 0 của cổng 1.

b) Tạo một hàm xung vuông với 66% độ đầy xung trên bít 3 của cổng 1.

Lời giải:

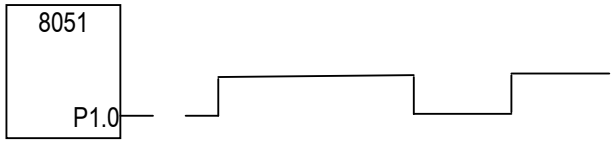
a) Hàm xung vuông với độ đầy xung 50% có nghĩa là trạng thái "bật" và "tắt" (hoặc phần cao và thấp của xung) có cùng độ dài. Do vậy ta chốt P1.0 với thời gian giữ chậm giữa các trạng thái.

```
HERE: SETB   P1.0          ;Thiết lập bít 0 cổng 1 lên 1.
        LCALL DELAY        ;Gọi chương trình con giữ chậm DELAY
      CLR P1.0             ;P1.0 = 0
      SJMP HERE            ;Tiếp tục thực hiện nó.
                           Có thể viết chương trình này theo cách khác:
HERE: CPL    P1.0          ;Bù bít 0 của cổng 1.
      LCALL DELAY          ;Gọi chương trình con giữ chậm DELAY
      SJMP   HERE          ;Tiếp tục thực hiện nó.
```



b) Hàm xung vuông với độ đầy xung 66% có nghĩa là trạng thái "bật" có độ dài gấp đôi trạng thái "tắt".

```
BACK: SETB  P1.3           ;Thiết lập bít 3 cổng 1 lên 1.
      LCALL DELAY          ;Gọi chương trình con DELAY
      LCALL DELAY          ;Gọi chương trình con DELAY lần nữa.
      CLR    P1.3          ;Xoá bít 3 của cổng 1 và 0.
      LCALL DELAY          ;Gọi chương trình con DELAY
      SJMP BACK            ;Tiếp tục thực hiện nó.
```



Lưu ý rằng, khi mã "P1.0" được hợp dịch nó trở thành "SETB 90H" vì P1.0 có địa chỉ trong RAM là 90h. Từ hình vẽ 8.1 ta thấy rằng các địa chỉ bít cho P0 là 80H đến 87H và cho P là 90H đến 97H v.v... Hình 8.1 cũng chỉ ra tất cả các thanh ghi có khả năng đánh địa chỉ theo bít.

Bảng 8.2: Khả năng đánh địa chỉ theo bít của các cổng.

P0	P1	P2	P3	Port's Bit
P0.0	P1.0	P2.0	P3.0	D0
P0.1	P1.1	P2.1	P3.1	D1
P0.2	P1.2	P2.2	P3.2	D2
P0.3	P1.3	P2.3	P3.3	D3
P0.4	P1.4	P2.4	P3.4	D4
P0.5	P1.5	P2.5	P3.5	D5
P0.6	P1.6	P2.6	P3.6	D6
P0.7	P1.7	P2.7	P3.7	D7

Ví dụ 8.2:

Đối với các lệnh dưới đây thì trạng thái của bít nào của SFR sẽ bị tác động (hãy sử dụng hình 8.1).

a) SETB 86H, b) CLR 87H, c) SETB 92H

b) SETB DA7H, e) CLR 0F2H, f) SETB OE7H

Lời giải

a) SETB	86H là dành cho SETB P0.6
b) CLR	87H là dành cho CLR P0.7
c) SETB	92H là dành cho SETB P1.2
d) SETB	0A7H là dành cho SETB P2.7
e) CLR	0F2H là dành cho CLR D2 của thanh ghi B
f) SETB	0E7H là dành cho SETB ACC.7 (bít D7 của thanh ghi A)

8.1.3 Kiểm tra một bít đầu vào.

Lệnh JNB (nhảy nếu bít = 0) và JB (nhảy nếu bít bằng 1) cũng là các phép thao tác đơn bít được sử dụng rộng rãi. Chúng cho phép ta hiển thị một bít và thực hiện quyết định phụ thuộc vào việc liệu nó là 0 hay là 1.

Ví dụ 8.3: Giả sử bít P2.3 là một đầu vào và biểu diễn điều kiện của một lô. Nếu nó bật lên 1 thì có nghĩa là lô nóng. Hãy hiển thị liên tục, mỗi khi nó lên cao thì hãy gửi một xung cao-xuống-thấp (Aigh-to-low) đến cổng P1.5 để bật còi báo.

Lời giải:

```
HERE:   JNB   P2.3, HERE      ; Duy trì hiển thị cao.
        SETB P1.5             ; Thiết lập P1.5 = 1
        CLR   P1.5            ; Thực hienẹ chuyển xung từ cao-xuống-thấp
```

Các lệnh JNB và JB có thể được dùng đối với các bít bất kỳ của các cổng I/O 0, 1, 2 và 3 vì tất cả các cổng này đều có khả năng đánh địa chỉ theo bít. Tuy nhiên, cổng 3 hầu như để dùng cho các tín hiệu ngắt và truyền thông nối tiếp và thông thường không dùng cho bất cứ vào/ ra theo bít hoặc theo byte nào. Điều này sẽ được bàn ở chương 10 và 11.

8.1.4 Các thanh ghi và khả năng đánh địa chỉ theo bít.

Trong tất cả các cổng I/O đều có khả năng đánh địa chỉ theo bít thì các thanh ghi lại không được như vậy. Ta có thể nhìn thấy điều đó từ hình 8.1: Chỉ thanh gh B, PSW, IP, IE, ACC, SCON và TCON là có thể đánh địa chỉ theo bít, ở đây ta sẽ tập trung vào các thanh ghi A, B và PSW còn các thanh ghi khác sẽ đề cập ở các chương sau. Từ hình 8.1 hãy để ý rằng cổng PO được gán địa chỉ bít 80H-87H. Còn đại chỉ bít 88-8FH được gán cho thanh ghi TCON.

Cuối cùng địa chỉ bít F0-F7H được gán cho thanh ghi B. Xét ví dụ 8.4 và 8.5 về việc sử dụng các thanh ghi này với khả năng đánh địa chỉ theo bít.

Byte address	Bit address								
FF									
F0	E7	F6	F5	F4	F3	F2	F1	F0	B
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
D0	D7	D6	D5	D4	D3	D2	D1	D0	PSW
B8	- -	- -	- -	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	F3
A8	AF	- -	- -	AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	not bit addressable								SBUF
98	9F	9E	9D	9C	9B	9A	99	99	SCON
90	97	96	95	94	93	92	91	90	P1
8D	not bit addressable								TH1
8C	not bit addressable								TH0
8B	not bit addressable								TL1
8A	not bit addressable								TL0
89	not bit addressable								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	not bit addressable								PCON
83	not bit addressable								DPH
82	not bit addressable								DPL
81	not bit addressable								SP
80	87	86	85	84	83	82	81	80	P0

Special Function Registers

Hình 8.1: Địa chỉa theo Byte và bít của bộ nhớ RAM các thanh ghi chức năng đặc biệt.

Ví dụ 8.4: Hãy viết chương trình để kiểm tra xem thanh ghi tích luỹ có chứa một số chẵn không? Nếu có thì chia nó cho 2, nếu không thì hãy làm chẵn nó và sau đó chia nó cho 2.

Lời giải:

```
MOV  B, # 2            ; Gán B = 2
JNB  ACC 0, YES        ; DO của thanh ghi A có bằng 0?
JNC  A                 ; Nếu có thì nhảy về YES
```

```
YES:    DIX   AB          ; Nếu là sô lẻ thì tăng lên 1 để thành chẵn
                          ; Chia A/B
```

Ví dụ 8.5: Hãy viết đoạn chương trình để kiểm tra xem các bít 0 và 5 của thanh ghi B có giá trị cao không? Nếu không phải thì đặt chúng lên 1 và lưu vào thanh ghi bộ.
Lời giải:

```
        JNB   OFOH, NEXT-1      ; Nhảy về NEXT-1 nếu B.0 = 0
        SETB OFOH               ; Đặt B.0 = 1
NEXT-1: JNB     OF5H, NEXT-2    ; Nhảy về NEXT-2 nếu B.5 = 0
        SETB OF5H               ; Đặt B.5 = 1
NEXT-2: MOV     RO, B           ; Cắt thanh ghi B
```

CY	AC	- -	RS1	RS0	OV	- -	P

RS1	**RS0**	**Register Bank**	**Address**
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

Hình 8.2: Các bít của thanh ghi PSW.

Như đã nói ở chương 2, trong than ghi PSW có hai bít dành riêng để chọn các bảng thanh ghi. Khi RESET thì bằng 0 được chọn, chúng ta có thể chọn các băng bất kỳ khác bằng cách sử dụng khả năng đánh địa chỉ theo bít của PSW.
Ví dụ 8.6: Hãy viết chương trình để lưu thanh ghi tích luỹ vào R7 của băng 2.
Lời giải:

```
CLR     PSW.3
SETB    PSW.4
MOV     R7.A
```

Ví dụ 8.7: Trong khi có hai lệnh JNC và JC để kiểm tra bít cờ nhớ CY thì lại không có các lệnh cho bít cờ tràn (0V) làm thế nào để ta có thể viết mã kiểm tra 0V.
Lời giải: Cờ 0V là bít PSW.2 của thanh ghi PSW. PSW là thanh ghi có thể đánh địa chỉ theo bít, do vậy ta có thể sử dụng lệnh sau để kiểm tra cờ 0V:

```
JB      PSW.2, TARGET       ; Nhảy về TARGET nếu 0V = 1
```

8.15 Vùng nhớ RAM có thể đánh địa chỉ theo bít.

Trong 128 byte RAM trong của 8051 thì chỉ có 16 byte của nó là có thể đánh địa chỉ theo bít được. Phần còn lại được định dạng byte. Các cùng RAM có thể đánh địa chỉ theo bít là 20H đến 2FH. Với 16 byte này của RAM có thể cung cấp khả năng đánh địa chỉ theo bít là 128 bít, vì $16 \times 8 = 128$. Chúng được đánh địa chỉ từ 0 đến 127. Do vậy, những địa chỉ bít từ 0 đến 7 dành cho byte đầu tiên, vị trí RAM trong 20H và các bít từ 8 đến OFH là địa chỉ bít của byte thứ hai của vị trí RAM trong 21H v.v... Byte cuối cùng của 2FH có địa chỉ bít từ 78H đến 7FH (xem hình 8.3). Lưu ý rằng các vị trí RAM trong 20H đến 2FH vừa có thể đánh địa chỉ theo byte vừa có thể đánh địa chỉ theo bít.

Để ý từ hình 8.3 và 8.1 ta thấy rằng các địa chỉ bít 00 - 7FH thuộc về các địa chỉ byte của RAM từ 20 - 2FH và các địa chỉ bít từ 80 đến F7H thuộc các thanh ghi đặc biệt SFR, các cổng P0, P1, v.v...
Ví dụ 8.8: Hãy kiểm tra xem các bít sau đây thuộc byte nào? Hãy cho địa chỉ của byte RAM ở dạng Hex.

```
a) SETB    42H          ; Set bit 42H to 1
b) CLR     67H          ; Clear bit 67
c) CLR     0FH          ; Clear bit OFH
d) SETB    28H          ; Set bit 28H to 1
e) CLR     12           ; Clear bit 12 (decimal)
f) SETB    05
```

Lời giải:

a) Địa chỉ bít 42H của RAM thuộc bít D2 của vị trí RAM 28H.
b) Địa chỉ bít 67H của RAM thuộc bít D7 của vị trí RAM 20H.
c) Địa chỉ bít 0FH của RAM thuộc bít D7 của vị trí RAM 21H.
d) Địa chỉ bít 28H của RAM thuộc bít D0 của vị trí RAM 25H.
e) Địa chỉ bít 12H của RAM thuộc bít D4 của vị trí RAM 21H.
f) Địa chỉ bít 05H của RAM thuộc bít D5 của vị trí RAM 20H.

Ví dụ 8.9: Trạng thái của các bít P1.2 và P1.3 của cổng vào/ra P1 phải được lưu cất trước khi chúng được thay đổi. Hãy viết chương trình để lưu trạng thái của P1.2 vào vị trí bít 06 và trạng thái P1.3 vào vị trí bít 07.

Lời giải:

```
        CLR     06          ;Xoá địa chỉ bít 06
        CLR     07          ; Xoá địa chỉ bít 07
        JNB     P1.2, OVER  ;Kiểm tra bít P1.2 nhảy về OVER nếu P1.2 = 0
        SETB    06          ; Nếu P1.2 thì thiết lập vị trí bít 06 = 0
OVER:   JNB     P1.3, NEXT   ;Kiểm tra bít P1.3 nhảy về NEXT nếu nó = 0
        SETB    07          ;Nếu P1.3 = 1thì thiết lập vị trí bít 07 = 1
NEXT: ....
```

Các câu hỏi ôn luyện:

1. Tất cả các cổng I/O của 8051 đều có khả năng đánh địa chỉ theo bít? (đúng sai)
2. Tất cả mọi thanh ghi của 8051 đều có khả năng đánh địa chỉ theo bít? (đúng sai)
3. Tất cả các vị trí RAM của 8051 đều có khả năng đánh địa chỉ theo bít? (đúng sai)
4. Hãy chỉ ra những thanh ghi nào sau đây có khả năng đánh địa chỉ theo bít:
   a) A, b) B, (c) R4 (d) PSW (e) R7
5. Trong 128 byte RAM của 8051 những byte nào có khả năng đánh địa chỉ theo bít. Hãy liệt kê chúng.
6. Làm thế nào để có thể kiểm tra xem bít D0 của R3 là giá trị cao hay thấp.
7. Hãy tìm xem các bít dau thuộc những byte nào? Hãy cho địa chỉ của các byte RAM theo số Hex:
   a) SETB 20 b) CLR 32 c) SETB 12H
   d) SETB 95 e) SETB 0ETB 12H
8. Các địa chỉ bít 00 - 7FH và 80 - F7H thuộc các vị trí nhớ nào?
9. Các cổng P0, P1, P2 và P3 là một bộ phận của SFR? (đúng sai)
10. Thanh ghi TCON có thể đánh địa chỉ theo bít (đúng sai)

8.2 Các phép toán một bít với cờ nhớ CY.

Ngoài một thực tế là cờ nhớ CY được thay đổi bởi các lệnh lô-gíc và số học thì trong 8051 còn có một số lệnh mà có thể thao tác trực tiếp cờ nhớ CY. Các lệnh này được cho trong bảng 8.3.

Trong các lệnh được chỉ ra sau trong bảng 8.3 thì chúng ta đã trình bày công dụng của lệnh JNC, CLR và SETB trong nhiều ví dụ trong một số chương trước đây. Dưới đây ta tiếp tục làm quen với một số ví dụ về cách sư dụng một số lệnh khác từ bảng 8.3.

Một số lệnh cho trong bảng 8.3 làm việc với các phép toán lô-gíc AND và OR. Các ví dụ ở mục này sẽ chỉ ra cách sử dụng chúng như thế nào?

Ở chương tiếp theo chúng ta sẽ chỉ ra nhiều ví dụ hơn về việc sử dụng của các lệnh đơn trong phạm vi các ứng dụng thực tế.

Bảng 8.3: Các lệnh liên quan đến cờ nhớ CY

Lệnh	chức năng
SETB C	Thực hiện (tao) CY = 1
CLR C	Xoá bít nhớ CY = 0
CPL C	Bù bít nhớ
MOV b, C	Sao chép trạng thái bít nhớ vào vị trí bít b = CY
MOV C, b	Sao chép bít b vào trạng thái bít nhớ CY = b
JNC đích	Nhảy tới đích nếu CY = 0
JC đích	Nhảy tới đích nếu CY = 1
ANL C. bít	Thực hiện phép AND với bít b và lưu vào CY
ANL C./ bít	Thực hiện phép AND với bít đảo và lưu vào CY
ORL C. bít	Thực hiện phép OR với bít và lưu vào CY
ORL C./ bít	Thực hiện phép OR với bít đảo và lưu vào CY

Ví dụ 8.10: Hãy viết một chương trình để lưu cất trạng thái của các bít P1.2 và P1.3 vào vị trí nhớ tương ứng trong RAM 6 và 7.

Lời giải:

```
MOV  C,  P1.2    ; Lưu trạng thái P1.2 vào CY.
MOV  06, C       ; Lưu trạng thái CY vào bít 6 của RAM
MOV  C, P1.3     ; Lưu trạng thái P1.2 vào CY
MOV  07, C       ; Lưu trạng thái CY vào vị trí RAM 07
```

Ví dụ 8.11:

Giả sử vị trí nhớ 12H trong RAM giữ trạng thái của việc có điện thoại hay không. Nếu nó ở trạng thái cao có nghĩa là đã có một cuộc gọi mới vì nó được kiểm tra lần cuối. Hãy viết một chương trình để hiển thị "có lời nhắn mới" ("New Message") trên màn hình LCD nếu bít 12H của RAM có giá trị cao. Nếu nó có giá trị thấp thì LCD hiển thị "không có lời nhắn mới" ("No New Message").

Lời giải:

```
        MOV  C, 12H          ; Sao trạng thái bít 12H của RAM vào CY
        JNC   NO             ; Kiểm tra xem cờ CY có giá trị cao không.
        MOV  DPTR, # 400H    ; Nếu nó nạp địa chỉ của lời nhắn.
        LCAL DISPLAY         ; Hiển thị lời nhắn.
        SJMP  NEXT           ; Thoát
NO:     MOV  DSTR, #420H     ; Nạp địa chỉ không có lời nhắn.
        LCAL DISPLAY         ; Hiển thị nó.
EXIT:                          Thoát
;  ____________________ data to be displayed on LCD
                ORG  400H
YES-MG:         DB   "NEW Message"
                ORG  420H
NO-MG:          DB   "No New Message"
```

Ví dụ 8.12:

Giả sử rằng bít P2.2 được dùng để kiểm tra đèn ngoài và bít P2.5 dùng để kiểm tra đèn trong của một toà nhà. Hãy trình bày làm thế nào để bật đèn ngoài và tắt đèn trong nhà.

Lời giải:

```
SETB    C           ; Đặt CY = 1
ORL     C, P2.2, C  ; Thực hiện phép OR với CY
MOV     P2.2, C     ; Bật đèn nếu nó chưa bật.
CLR     C           ; Xoá CY = 0
ANL     C, P2.5     ; CY = (P2.5 AND CY)
MOV     P2.5, C     ; Tắt nó nếu nó chưa tắt.
```

Câu hỏi ôn luyện:

1. Tìm trạng thái của cờ CY sau đoạn mã dưới đây:

```
a) CLR      A           b) CLR   C          c) CLR   C
   ADD      A, #OFFH     JNC     OVER        JC      OVER
   JWC      OVER        SETB  C              CPL     C
   CPL      C             OVER: ...           OVER: ...
OVER: ...
```

2. Hãy trình bày cách làm thế nào để lưu trạng thái bít P2.7 vào vị trí bít 31 của RAM.
3. Hãy trình bày các chuyển trạng thái bít 09 của RAM đến bít P1.4.

8.3 Đọc các chân đầu vào thông qua chốt cổng.

Trong việc đọc cổng thì một số lệnh đọc trạng thái của các chân cổng, còn một số lệnh khác thì đọc một số trạng thái của chốt cổng trong. Do vậy, khi đọc các cổng thì có hai khả năng:

1. Đọc trạng thái của chân vào.
2. Đọc chốt trong của cổng ra.

Chúng ta phải phân biệt giữa hai dạng lệnh này vì sự lẫn lộn giữa chúng là nguyên nhân chính của các lỗi trong lập trình cho 8051, đặc biệt khi đã kết nối với phần cứng bên ngoài. Trong phần này ta bàn về sơ qua các lệnh này. Tuy nhiên, độc giả phải nghiên cứu và hiểu về các nội dung của chủ đề này và về hoạt động bên trong của các cổng được cho trong phụ lục Appendix C2.

8.3.1 Các lệnh đọc cổng vào.

Như đã nói ở chương 4 thì để biến một bít bất kỳ của cổng 8051 nào đó thành một cổng đầu vào, chúng ta phải ghi (lô-gíc cao) vào bít đó. Ssu khi cấu hình các bít của cổng là đầu vào, ta có thể sử dụng những lệnh nhất định để nhận dữ liệu ngoài trên các chân vào trong CPU. Bảng 8.4 là những lệnh nói trên.

Bảng 8.4: Các lệnh đọc một cổng vào.

Giả lệnh	Ví dụ	Mô tả
MOV A, PX	MOV A, P2	Chuyển dữ liêuj ở chân P2 vào ACC
JNB PX.Y, ...	JNB P2.1, đích	Nhảy tới đích nếu, chân P2.1 = 0
JB PX.Y,	JB P1.3, đích	Nhảy đích nếu, chân P1.3 = 1
MOV C, PX.Y	MOV C, P2.4	Sao trạng thái chân P2.4 vào CY

8.3.2 Đọc chốt cho cổng đầu ra.

Một số lệnh nội dung của một chốt cổng trong thay cho việc đọc trạng thái của một chân ngoài. Bảng 8.5 cung cấp danh sách những lệnh này. Ví dụ, xét lệnh "ANL P1, A". Trình tự thao tác được thực hiện bởi lệnh này như sau:

1. Nó đã chốt trong của một cổng và chuyển dữ liệu đó vào trong CPU.
2. Dữ liệu này được AND với nội dung của thanh ghi A.
3. Kết quả được ghi ngược lại ra chốt cổng.
4. Dữ liệu tại chân cổng được thay đổi và có cùng giá trị như chốt cổng.

Từ những bàn luận trên ta kết luận rằng, các lệnh đọc chốt cổng thường đọc một giá trị, thực hiện một phép tính (và có thể thay đổi nó) sau đó ghi ngược lại ra chốt cổng. Điều này thường được gọi "Đọc-sửa-ghi", ("Read-Modify-Write"). Bảng 8.5 liệt kê các lệnh đọc-sửa-ghi sử dụng cổng như là toán hạng đích hay nói cách khác, chúng ta chỉ được dùng cho các cổng được cấu hình như các cổng ra.

Bảng 8.5: Các lệnh đọc một chốt (Đọc-sửa-ghi).

giả lệnh	Ví dụ
ANL PX	ANL P1, A
ORL PX	ORL P2, A
XRL PX	XRL P0, A
JBC PX.Y, đích	JBC P1.1, đích
CPL PX	CPL P1.2
INC PX	INC P1
DEC PX	DEC P2
DJN2 PX.Y, đích	DJN2 P1, đích
MOV PX.Y, C	MOV P1.2, C
CLR PX.Y	CLR P2.3
SETB PX.Y	SETB P2.3

Lưu ý: Chúng ta nên nghiên cứu phần C2 của phụ lục Appendix C nếu ta nối phần cứng ngoài vào hệ 8051 của mình. Thực hiện sai các chỉ dẫn hoặc nối sai các chân có thể làm hỏng các cổng của hệ 8051.

8.4 Tóm lược.

Chương này đã mô tả một trong các đặc tính mạnh nhất của 8051 là phép toán một bít. Các phép toán một bít này cho phép lập trình viên thiết lập, xoá, di chuyển và bù các bít riêng rẽ của các cổng, bộ nhớ hoặc các thanh ghi.

Ngoài ra có một số lệnh cho phép thao tác trực tiếp với cờ nhớ CY. Chúng ta cũng đã bàn về các lệnh đọc các chân cổng thông qua việc đọc chốt cổng.

8.5 Các câu hỏi kiểm tra.

1. Các lệnh "SETB A", "CLR A", "CPL A" đúng hay sai?
2. Các cổng vào/ ra nào và các thanh ghi nào có thể đánh địa chỉ theo bít.
3. Các lệnh dưới đây đúng hay sai? Đánh dấu lệnh đúng.

a) SETB P1 e) SETB B4
b) SETB P2.3 f) CLR 80H
c) CLR ACC.5 g) CLR PSW.3
d) CRL 90H h) CLR 87H

4. Hãy giết chương trình tạo xung vuông với độ đầy xung 75%, 80% trên các chân P1.5 và P2.7 tương ứng.
5. Viết chương trình hiển thị P1.4 nếu nó có giá trị cao thì chương trình tạo ra một âm thanh (sóng dung vuông 50% độ đầy xung) trên chân P2.7.
6. Nhưng địa chỉ bít nào được gán cho các cổng P0, P1, P2 và P3 cho các thanh ghi PCON, A, B và PSW.

7. Những địa chỉ bít dưới đây thuộc về cổng hay thanh ghi nào?

a) 85H b)87H c) 88H d)8DH e)93H

f) A5H g)A7H h) B3H i) D4H j) D8H

8. Hãy viết chương trình lưu các thanh ghi A, B vào R3 và R5 băng nhớ 2 tương ứng.

9. Cho một lệnh khác cho "CLR C", so sánh chúng.

10. Làm thế nào để kiểm tra trạng thái các bít cờ OV, CY, P và AC. Hãy tìm địa chỉ bít của các cờ này.

11. Các cùng nhớ 128 byte của RAM thì những vùng nào là đánh địa chỉ theo bít được? Hãy đánh dấu chúng.

12. Các địa chỉ sau thuộc vùng RAM nào?

a) 05H b) 47 c) 18H d) 2DH e) 53H

g) 15H h) 67H h) 55H i) 14H k) 37FH

13. Các địa chỉ nhỏ hơn 80H được gán cho địa chỉ 20-2FH của RAM phải không? (Đúng/ sai).

14. Viết các lệnh để lưu cờ CY, AC, D vào vị trí bít 4, 16H và 12H tương ứng.

15. Viết chương trình kiểm tra D7 của thanh ghi A. Nếu D7 = 1 thì gửi thông báo sang LCD báo rằng ACC có một số âm.

CHƯƠNG 9
Lập trình cho bộ đếm/ bộ định thời trong 8051

8051 có hai bộ định thời/ bộ đếm. Chúng có thể được dùng như các bộ định thời để tạo một bộ trễ thời gian hoặc như các bộ đếm để đếm các sự kiện xảy ra bên ngoài bộ BVĐK. Trong chương này chúng ta sẽ tìm hiểu về cách lập trình cho chúng và sử dụng chúng như thế nào?

9.1 Lập trình các bộ định thời gian của 8051.

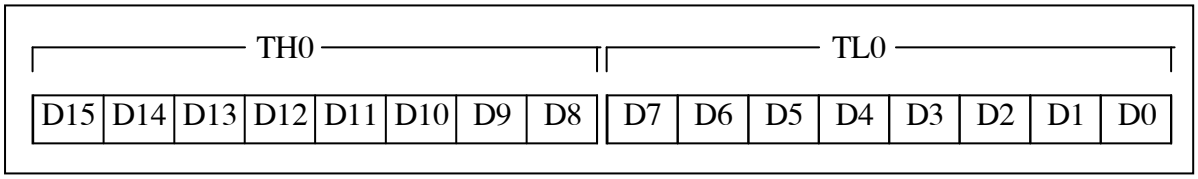
8051 có hai bộ định thời là Timer 0 và Timer1, ở phần này chúng ta bàn về các thanh ghi của chúng và sau đó trình bày cách lập trình chúng như thế nào để tạo ra các độ trễ thời gian.

9.1.1 Các thanh ghi cơ sở của bộ định thời.

Cả hai bộ định thời Timer 0 và Timer 1 đều có độ dài 16 bít được truy cập như hai thanh ghi tách biệt byte thấp và byte cao. Chúng ta sẽ bàn riêng về từng thanh ghi.

9.1.1.1 Các thanh ghi của bộ Timer 0.

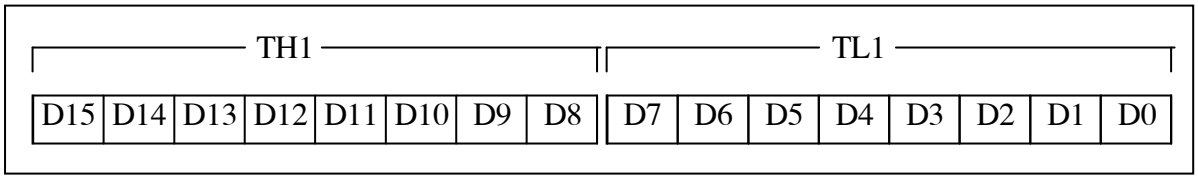
Thanh ghi 16 bít của bộ Timer 0 được truy cập như byte thấp và byte cao. Thanh ghi byte thấp được gọi là TL0 (Timer 0 bow byte) và thanh ghi byte cao là TH0 (Timer 0 High byte). Các thanh ghi này có thể được truy cập như mọi thanh ghi khác chẳng hạn như A, B, R0, R1, R2 v.v... Ví dụ, lệnh "MOV TL0, #4FH" là chuyển giá trị 4FH vào TL0, byte thấp của bộ định thời 0. Các thanh ghi này cũng có thể được đọc như các thanh ghi khác. Ví dụ "MOV R5, TH0" là lưu byte cao TH0 của Timer 0 vào R5.



Hình 9.1: Các thanh ghi của bộ Timer 0.

9.1.1.2 Các thanh ghi của bộ Timer 1.

Bộ định thời gian Timer 1 cũng dài 16 bít và thanh ghi 16 bít của nó được chia ra thành hai byte là TL1 và TH1. Các thanh ghi này được truy cập và đọc giống như các thanh ghi của bộ Timer 0 ở trên.



Hình 9.2: Các thanh ghi của bộ Timer 1.

9.1.2 Thanh ghi TMOD (chế độ của bộ định thời).

Cả hai bộ định thời Timer 0 và Timer 1 đều dùng chung một thanh ghi được gọi là IMOD để thiết lập các chế độ làm việc khác nhau của bộ định thời. Thanh ghi TMOD là thanh ghi 8 bít gồm có 4 bít thấp được thiết lập dành cho bộ Timer 0 và 4 bít cao dành cho Timer 1. Trong đó hai bít thấp của chúng dùng để thiết lập chế độ của bộ định thời, còn 2 bít cao dùng để xác định phép toán. Các phép toán này sẽ được bàn dưới đây.

(MSB) (MSB)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer1				Timer0			

Hình 9.3: Thanh ghi IMOD.

9.1.2.1 Các bít M1, M0:

Là các bít chế độ của các bộ Timer 0 và Timer 1. Chúng chọn chế độ của các bộ định thời: 0, 1, 2 và 3. Chế độ 0 là một bộ định thời 13, chế độ 1 là một bộ định thời 16 bít và chế độ 2 là bộ định thời 8 bít. Chúng ta chỉ tập chung vào các chế độ thường được sử dụng rộng rãi nhất là chế độ 1 và 2. Chúng ta sẽ sớm khám phá ra các đặc tính củ các chế độ này sau khi khám phần còn lại của thanh ghi TMOD. Các chế độ được thiết lập theo trạng thái của M1 và M0 như sau:

M1	M0	Chế độ	Chế độ hoạt động
0	0	0	Bộ định thời 13 bít gồm 8 bít là bộ định thời/ bộ đếm 5 bít đặt trước
0	1	1	Bộ định thời 16 bít (không có đặt trước)
1	0	2	Bộ định thời 8 bít tự nạp lại
1	1	3	Chế độ bộ định thời chia tách

9.1.2.2 C/ T (đồng hồ/ bộ định thời).

Bít này trong thanh ghi TMOD được dùng để quyết định xem bộ định thời được dùng như một máy tạo độ trễ hay bộ đếm sự kiện. Nếu bít C/T = 0 thì nó được dùng như một bộ định thời tạo độ chễ thời gian. Nguồn đồng hồ cho chế độ trễ thời gian là tần số thạch anh của 8051. Ở phần này chỉ bàn về lựa chọn này, công dụng của bộ định thời như bộ đếm sự kiện thì sẽ được bàn ở phần kế tiếp.

Ví dụ 9.1: Hãy hiển thị xem chế độ nào và bộ định thời nào đối với các trường hợp sau:

a) MOV TMOD, #01H b) MOV TMOD, #20H c) MOV TMDO, #12H

Lời giải: Chúng ta chuyển đổi giá trị từ số Hex sang nhị phân và đối chiếu hình 93 ta có:

a) TMOD = 0000 0001, chế độ 1 của bộ định thời Timer 0 được chọn.
b) TMOD = 0010 0000, chế độ 1 của bộ định thời Timer 1 được chọn.
c) TMOD = 0001 0010, chế độ 1 của bộ định thời Timer 0 và chế độ 1 của Timer 1 được chọn.

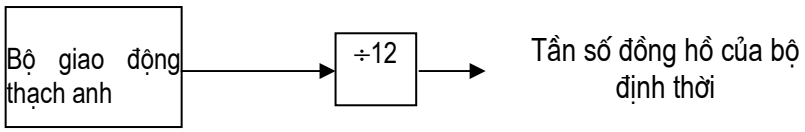
9.1.2.3 Nguồn xung đồng hồ cho bộ định thời:

Như chúng ta biết, mỗi bộ định thời cần một xung đồng hồ để giữ nhịp. Vậy nguồn xung đồng hồ cho các bộ định thời trên 8051 lấy ở đâu? Nếu C/T = 0 thì tần số thạch anh đi liền với 8051 được làm nguồn cho đồng hồ của bộ định thời. Điều đó có nghĩa là độ lớn của tần số thạch anh đi kèm với 8051 quyết định tốc độ nhịp của các bộ định thời trên 8051. Tần số của bộ định thời luôn bằng 1/12 tần số của thạch anh gắn với 8051. Xem ví dụ 9.2.

Ví dụ 9.2:

Hãy tìm tần số đồng bộ và chu kỳ của bộ định thời cho các hệ dựa trên 8051 với các tần số thạch anh sau:

a) 12MHz
b) 16MHz
c) 11,0592MHz



Lời giải:

a) $\frac{1}{12}\times 12\text{MHz} = 1\text{MHz}$ và $T = \frac{1}{1/1\text{MHz}} = 1\mu\text{s}$

b) $\frac{1}{12}\times 16\text{MHz} = 1{,}111\text{Mz}$ và $T = \frac{1}{1{,}333\text{MHz}} = 0{,}75\mu\text{s}$

c) $\frac{1}{12}\times 11{,}0592\text{MHz} = 921{,}6\text{kHz}$ và $T = \frac{1}{0{,}9216\text{MHz}} = 1{,}085\mu\text{s}$

Mặc dù các hệ thống dựa trên 8051 khác với tần số thạch anh từ 10 đến 40MHz, song ta chỉ tập chung vào tần số thạch anh 11,0592MHz. Lý do đằng sau một số lẻ như vậy là hải làm việc với tần suất bouid đối với truyền thông nối tiếp của 8051. Tần số XTAL = 11,0592MHz cho phép hệ 8051 truyền thông với IBM PC mà không có lỗi, điều mà ta sẽ biết ở chương 10.

9.1.3 Bít cổng GATE.

Một bít khác của thanh ghi TMOD là bít cổng GATE. Để ý trên hình 9.3 ta thấy cả hai bộ định thời Timer0 và Timer1 đều có bít GATE. Vậy bít GATE dùng để làm gì? Mỗi bộ định thời thực hiện điểm khởi động và dừng. Một số bộ định thời thực hiện điều này bằng phần mềm, một số khác bằng phần cứng và một số khác vừa bằng phần cứng vừa bằng phần mềm. Các bộ định thời tren 8051 có cả hai. Việc khởi động và dừng bộ định thời được khởi động bằng phần mềm bởi các bít khởi động bộ định thời TR là TR0 và TR1. Điều này có được nhờ các lệnh "SETB TR1" và "CLR TR1" đối với bộ Timer1 và "SETB TRO" và "CLR TR0" đối với bộ Timer0. Lệnh SETB khởi động bộ định thời và lệnh CLR dùng để dừng nó. Các lệnh này khởi động và dừng các bộ định thời khi bít GATE = 0 trong thanh ghi TMOD. Khởi động và ngừng bộ định thời bằng phần cứng từ nguồn ngoài bằng cách đặt bít GATE = 1 trong thanh ghi TMOD. Tuy nhiên, để tránh sự lẫn lộn ngay từ bây giờ ta đặt GATE = 0 có nghĩa là không cần khởi động và dừng các bộ định thời bằng phần cứng từ bên ngoài. Để sử dụng phần mềm để khởi động và dừng các bộ định thời phần mềm để khởi động và dừng các bộ định thời khi GATE = 0. Chúng ta chỉ cần các lệnh "SETB TRx" và "CLR TRx". Việc sử dụng phần cứng ngoài để khởi động và dừng bộ định thời ta sẽ bàn ở chương 11 khi bàn về các ngắt.

Ví dụ 9.3:

Tìm giá trị cho TMOD nếu ta muốn lập trình bộ Timer0 ở chế độ 2 sử dụng thạch anh XTAL 8051 làm nguồn đồng hồ và sử dụng các lệnh để khởi động và dừng bộ định thời.

Lời giải:

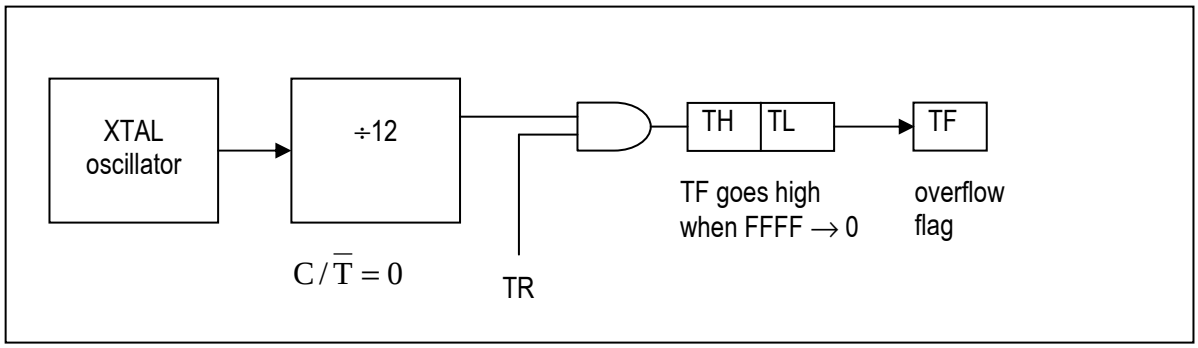
TMOD = 0000 0010: Bộ định thời Timer0, chế độ 2 C/T = 0 dùng nguồn XTAL GATE = 0 để dùng phần mềm trong để khởi động và dừng bộ định thời.

Như vậy, bây giờ chúng ta đã có hiểu biết cơ bản về vai trò của thanh ghi TMOD, chúng ta sẽ xét chế độ của bộ định thời và cách chúng được lập trình như thế nào để tạo ra một độ trễ thời gian. Do chế độ 1 và chế độ 2 được sử dụng rộng rãi nên ta đi xét chi tiết từng chế độ một.

9.1.4 Lập trình cho mỗi chế độ Mode1.

Dưới đây là những đặc tính và những phép toán của chế độ Mode1:

1. Nó là bộ định thời 16 bít, do vậy nó cho phép các giá trị 0000 đến FFFFH được nạp vào các thanh ghi TL và TH của bộ định thời.
2. Sau khi TL và TH được nạp một giá trị khởi tạo 16 bít thì bộ định thời phải được khởi động. Điều này được thực hiện bởi "SETB TR0" đối với Timer 0 và "SETB TR1" đối với Timer1.
3. Sau khi bộ định thời được khởi động, nó bắt đầu đếm lên. Nó đếm lên cho đến khi đạt được giới hạn FFFFH của nó. Khi nó quay qua từ FFFFH về 0000 thì nó bật lên bít cờ TF được gọi là cờ bộ định thời. Cờ bộ định thời này có thể được hiển thị. Khi cờ bộ định thời này được thiết lập từ một trong các phương án để dừng bộ định thời bằng các lệnh "CLR TR0" đối với Timer0 hoặc "CLR TR1" đối với Timer1. ở đây cũng cần phải nhắc lại là đối với bộ định thời đều có cờ TF riêng của mình: TF6 đối với Timer0 và TF1 đối với Timer1.



4. Sau khi bộ định thời đạt được giới hạn của nó và quay quan giá trị FFFFH, muốn lặp lại quá trình thì các thanh ghi TH và TL phải được nạp lại với giá trị ban đầu và TF phải được duy trì về 0.

9.1.4.1 Các bước lập trình ở chế độ Mode 1.

Để tạo ra một độ trễ thời gian dùng chế độ 1 của bộ định thời thì cần phải thực hiện các bước dưới đây.

1. Nạp giá trị TMOD cho thanh ghi báo độ định thời nào (Timer0 hay Timer1) được sử dụng và chế độ nào được chọn.
2. Nạp các thanh ghi TL và TH với các giáa trị đếm ban đầu.
3. Khởi động bộ định thời.
4. Duy trì hiển thị cờ bộ định thời TF bằng lệnh "JNB TFx, đích" để xem nó được bật không. Thoát vòng lặp khi TF được lên cao.
5. Dừng bộ định thời.
6. Xoá cờ TF cho vòng kế tiếp.
7. Quay trở lại bước 2 để nạp lại TL và TH.

Để tính toàn thời gian trễ chính xác và tần số sóng vuông được tạo ra trên chân P1.5 thì ta cần biết tần số XTAL (xem ví dụ 9.5).

Từ ví dụ 9.6 ta có thể phát triển một công thức tính toán độ trễ sử dụng chế độ Mode1 (16 bít) của bộ định thời đối với tần số thạch anh XTAL = 11, 0592MHz (xem hình 9.4). Máy tính trong thư mục Accessrry của Microsoft Windows có thể giúp ta tìm các giá trị TH và TL. Máy tính này hỗ trợ các phép tính theo số thập phân, nhị phân và thập lục.

a) Tính theo số Hex	b) Tính theo số thập phân
(FFFF - YYXX + 1). 1,085µs trong đó YYXX là các giá trị khởi tạo của TH, TL tương ứng. Lưu ý rằng các giá trị YYXX là theo số Hex.	Chuyển đổi các giá trị YYXX của TH, TL về số thập phân để nhận một số thập phân NNNNN sau đó lấy (65536 - NNNNN).1,085µs.

Hình 9.4: Công thức tính toán độ trễ thời gian đối với tần số XTAL = 11, 0592MHz.

Ví dụ 9.4:

Trong chương trình dưới đây ta tạo ra một sóng vuông với độ đầy xung 50% (cùng tỷ lệ giữa phần cao và phần thấp) trên chân P1.5. Bộ định thời Timer0 được dùng để tạo độ trễ thời gian. Hãy phân tích chương trình này.

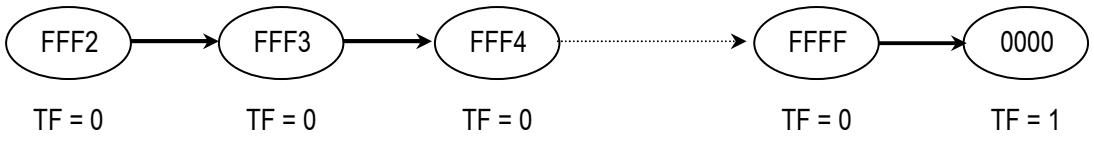
```
        MOV   TMOD, #01          ; Sử dụng Timer0 và chế độ 1(16 bít)
HERE:   MOV   TL0, #0F2H         ; TL0 = F2H, byte thấp
        MOV   TH0, #0FFH         ; TH0 = FFH, byte cao
        CPL   P1.5               ; Sử dụng chân P1.5
        ACALL DELAY
        SJMP  HERE               ; Nạp lại TH, TL
; _____ delay using timer0.
DELAY:
        SETB  TR0                ; Khởi động bộ định thời Timer0
AGAIN:  JNB   TF0, AGAIN         ; Hiển thị cờ bộ định thời cho đến khi nó vượt  qua FFFFH.
        CLR   TR0                ; Dừng bộ Timer
        CLR   TF0                ; Xoá cờ bộ định thời 0
        RET
```

Lời giải:

Trong chương trình trên đây chú ý các bước sau:

1. TMOD được nạp.
2. Giá trị FFF2H được nạp và TH0 - TL0
3. Chân P1.5 được chọn dùng cho phần cao thấp của xung.
4. Chương trình con DELAY dùng bộ định thời được gọi.
5. Trong chương trình con DELAY bộ định thời Timer0 được khởi động bởi lệnh "SETB TR0"
6. Bộ Timer0 đếm lên với mỗi xung đồng hồ được cấp bởi máy phát thạch anh. Khi bộ định thời đếm tăng qua các trạng thái FFF3, FFF4 ... cho đến khi đạt giá trị FFFFH. Và một xung nữa là nó quay về không và bật cờ bộ định thời TF0 = 1. Tại thời điểm này thì lệnh JNB hạn xuống.
7. Bộ Timer0 được dùng bởi lệnh "CLR TR0". Chương trình con DELAY kết thúc và quá trình được lặp lại.

Lưu ý rằng để lặp lại quá trình trên ta phải nạp lại các thanh ghi TH và TL và khởi động lại bộ định thời với giả thiết tần số XTAL = 11, 0592MHz.



Ví dụ 9.5:

Trong ví dụ 9.4 hãy tính toán lượng thời gian trễ trong chương trình con DELAY được tạo ra bởi bộ định thời với giá thiết tần số XTAL = 11,0592MHz.

Lời giải:

Bộ định thời làm việc với tần số đồng hồ bằng 1/12 tần số XTAL, do vậy ta có $\frac{11,0592}{12} = 0,9216\text{MHz}$ là tần số của bộ định thời. Kết quả là mỗi nhịp xung đồng hồ có chu kỳ $T = \frac{1}{0,9216\text{MHz}} = 1,085\mu s$. Hay nói cách khác, bộ Timer0 đếm tăng sau 1,085μs để tạo ra bộ trễ bằng số đếm ×1,085μs.

Số đếm bằng FFFFH - FFF2H = ODH (13 theo số thập phân). Tuy nhiên, ta phải cộng 1 vào 13 vì cần thêm một nhịp đồng hồ để nó quay từ FFFFH về 0 và bật cờ TF. Do vậy, ta có 14 × 1,085μs = 15,19μs cho nửa chu kỳ và cả chu kỳ là T = 2 × 15,19μs = 30, 38μs là thời gian trễ được tạo ra bởi bộ định thời.

Ví dụ 9.6:

Trong ví dụ 9.5 hãy tính toán tần số của xung vuông được tạo ra trên chân P1.5.

Lời giải:

Trong tính toán độ thời gian trễ của ví dụ 9.5 ta không tính đến tổng phí của các lệnh trong vòng lặp. Để tính toán chính xác hơn ta cần bổ xung thêm các chu kỳ thời gian của các lệnh trong vòng lặp. Để làm điều đó ta sử dụng các chu kỳ máy từ bảng A-1 trong phụ lục Appendix A được chỉ dưới đây.

```
HERE:   MOV   TL0, #0F2H          2
        MOV   TH0, #0FFH          2
        CPL   P1-5                1
        ACALL DELAY               2
        SJMP HERE                 2
; _____ delay using timer0
DELAY:
        SETB  TR0                 1
AGAIN:  JNB   TF0, AGAIN          1
        CLR   TR0                 1
        CLR   TF0                 1
        RET                       1
                          Total   27
```

T = (2 × 27 × 1.085μs and F = 17067.75Hz).

Tổng số chu kỳ đã bổ xung là x7 nên chu kỳ thời gian trễ là T = 2 × 27 × 1.085μs = 58,59μs và tần số là F = 17067,75Hz.

Ví dụ 9.7:

Hãy tìm ra độ trễ được tạo ra bởi Timer0 trong đoạn mã sau sử dụng cả hai phương pháp của hình 9.4. Không tính các tổng phí của các lệnh.

```
        CLR   P2.3          ; Xoá P2.3
        MOV   TMOD, #01     ; Chọn Timer0, chế độ 1 (16 bít)
HERE:   MOV   TL0, #3EH     ; TL0 = 3EH, byte thấp
        MOV   TH0, #0B8G    ; TH0 = B8H, byte cao
        SETB  P2.3          ; Bật P2.3 lên cao
        SETB  TR0           ; Khởi động Timer0
AGAIN:  JNB   TF0, AGAIN    ; Hiển thị cờ bộ định thời TF0
```

```
        CLR     TR0             ; Dừng bộ định thời.
        CLR     TF0             ; Xoá cờ bộ định thời cho vòng sau
        CLR     P2.3
```

Lời giải:

a) Độ trễ được tạo ra trong mã trên là:

(FFFF - B83E + 1) = 47C2H = 18370 hệ thập phân 18370 × 1,085μs = 19, 93145μs.

b) Vì TH - TL = B83EH = 47166 (số thập phân) ta có 65536 - 47166 = 18370.

Điều này có nghĩa là bộ định thời gian đếm từ B83EH đến FFFF. Nó được cộng với một số đếm để về 0 thành một bộ tổng là 18370μs. Do vậy ta có 18370 × 1,085μs = 19,93145ms là độ rộng xung.

Ví dụ 9.8:

Sửa giá trị của TH và TL trong ví dụ 9.7 để nhận được độ trễ thời gian lớn nhất có thể. Hãy tính độ trễ theo miligiây. Trong tính toán cần đưa vào cả tổng phí của các lệnh.

Để nhận độ trễ thời gian lớn nhất có thể ta đặt TH và TL bằng 0. Điều này làm cho bộ định thời đếm từ 0000 đến FFFFH và sau đó quay qua về 0.

```
        CLR     P2.3            ; Xoá P2.3
        MOV     TMOD, #01       ; Chọn Timer0, chế độ 1 (16 bít)
HERE:   MOV     TL0, #0         ; Đặt TL0 = 0, byte thấp
        MOV     TH0, #0         ; Đặt TH0 = 0, byte cao
        SETB    P2.3            ; Bật P2.3 lên cao
        SETB    TR0             ; Khởi động bộ Timer0
AGAIN:  JNB     TF0, AGAIN      ; Hiển thị cờ bộ định thời TF0
        CLR     TR0             ; Dừng bộ định thời.
        CLR     TF0             ; Xoá cờ TF0
        CLR     P2.3
```

Thực hiện biến TH và TL bằng 0 nghĩa là bộ định thời đếm tăng từ 0000 đến FFFFH và sau đó quay qua về 0 để bật cờ bộ định thời TF. Kết quả là nó đi qua 65536 trạng thái. Do vậy, ta có độ trễ = (65536 - 0) × 1.085μs = 71.1065μs.

Trong ví dụ 9.7 và 9.8 chúng ta đã không nạp lại TH và TL vì nó là một xung đơn. Xét ví dụ 9.9 dưới đây để xem việc nạp lại làm việc như thế nào ở chế độ 1.

Ví dụ 9.9:

Chương trình dưới đây tạo ra một sóng vuông trên chân P2.5 liên tục bằng việc sử dụng bộ Timer1 để tạo ra độ trễ thời gian. Hãy tìm tần số của sóng vuông nếu tần số XTAL = 11.0592MHz. Trong tính toán không đưa vào tổng phí của các lệnh vòng lặp:

```
        MOV     TMOD, #01H      ; Chọn Timer0, chế độ 1 (16 bít)
HERE:   MOV     TL1, #34H       ; Đặt byte thấp TL1 = 34H
        MOV     TH0, #76H       ; Đặt byte cao TH1 = 76H
                                ; (giá trị bộ định thời là 7634H)
        SETB    TR1             ; Khởi động bộ Timer1
AGAIN:  JNB     TF1, BACK       ; ở lại cho đến khi bộ định thời đếm qua 0
        CLR     TR1             ; Dừng bộ định thời.
        CPL     P1.5            ; Bù chân P1.5 để nhận Hi, L0
        CLR     TF              ; Xoá cờ bộ định thời
        SJMP    AGAIN           ; Nạp lại bộ định thời do chế độ 1 không tự
                                  động nạp lại .
```

Lời giải:

Trong chương trình trên đây ta lưu ý đến đích của SJMP. Ở chế độ 1 chương trình phải nạp lại thanh ghi. TH và TL mỗi lần nếu ta muốn có sóng dạng liên tục. Dưới đây là kết quả tính toán:

Vì FFFFH - 7634H = 89CBH + 1 = 89CCH và 90CCH = 35276 là số lần đếm xung đồng hồ, độ trễ là $35276 \times 1.085\mu s = 38274ms$ và tần số là $\frac{1}{38274}(Hz) = 26127Hz$.

Cũng để ý rằng phần cao và phần thấp của xung sóng vuông là bằng nhau. Trong tính toán trên đây là chưa kể đến tổng phí các lệnh vòng lặp.

9.1.4.2 Tìm các giá trị cần được nạp vào bộ định thời.

giả sử rằng chúng ta biết lượng thời gian trễ mà ta cần thì câu hỏi đặt ra là làm thế nào để tìm ra được các giá trị cần thiết cho các thanh thi TH và TL. Để tính toán các giá trị cần được nạp vào các thanh ghi TH và TL chúng ta hãy nhìn vào ví dụ sau với việc sử dụng tần số dao động XTAL = 11. 0592MHz đối với hệ 8051.

Từ ví dụ 9.10 ta có thể sử dụng những bước sau để tìm các giá trị của các thanh ghi TH và TL.

1. Chia thời gian trễ cần thiết cho 1.0592μs
2. Thực hiện 65536 - n với n là giá trị thập phân nhận được từ bước 1.
3. Chuyển đổi kết quả ở bước 2 sang số Hex với yyxx là giá trị .hex ban đầu cần phải nạp vào các thanh ghi bộ định thời.
4. Đặt TL = xx và TH = yy.

Ví dụ 9.10:

Giả sử tần số XTAL = 11.0592MHz. Hãy tìm các giá trị cần được nạp vào các thanh ghi vào các thanh ghi TH và TL nếu ta muốn độ thời gian trễ là 5μs. Hãy trình bày chương trình cho bộ Timer0 để tạo ra bộ xung với độ rộng 5μs trên chân P2.3.

Lời giải:

Vì tần số XTAL = 11.0592MHz nên bộ đếm tăng sau mỗi chu kỳ 1.085μs. Điều đó có nghĩa là phải mất rất nhiều khoảng thời gian 1,085μs để có được một xung 5μs. Để có được ta chia 5ms cho 1.085μs và nhận được số n = 4608 nhịp. Để nhận được giá trị cần được nạp vào TL và TH thì ta tiến hành lấy 65536 trừ đi 4608 bằng 60928. Ta đổi số này ra số hex thành EE00H. Do vậy, giá trị nạp vào TH là EE Và TL là 00.

```
        CLR   P2.3          ; Xoá bít P2.3
        MOV   TMOD, #01     ; Chọn Timer0, chế độ 1 (16 bít)
HERE:   MOV   TL0, #0       ; Nạp TL = 00
        MOV   TH0, #EEH     ; Nạp TH = EEH
        SETB  P2.3          ; Bật P2.3 lên cao
        SETB  TR0           ; Khởi động bộ định thời Timer0
AGAIN:  JNB   TF0, AGAIN    ; Hiển thị cờ TF0 cho đến khi bộ đếm quay về 0
        CLR   TR0           ; Dừng bộ định thời.
        CLR   TF0           ; Xoá cờ TF0 cho vòng sau.
```

Ví dụ 9.11:

Giả sử ta có tần số XTAL là 11,0592MHz hãy viết chương trình tạo ra một sóng vuông tần số 2kHz trên chân P2.5.

Đây là trường hợp giống với ví dụ 9.10 ngoài trừ một việc là ta phải chọn bít để tạo ra sóng vuông. Xét các bước sau:

a) $T = \frac{1}{f} = \frac{1}{2kHz} = 500\mu s$ là chu kỳ của sóng vuông.

b) Khoảng thời gian cao và phần thấp là $\frac{1}{2}T$ bằng 250µs.

c) Số nhịp cần trong thời gian đó là $\frac{250\mu s}{1{,}085\mu s} = 230$ và giá trị cần nạp vào các thanh ghi cần tìm là 65536 - 230 = 65306 và ở dạng hex là FF1AH.

d) giá trị nạp vào TL là 1AH và TH là FFH.
Chương trình cần viết là:

```
        MOV   TMOD, #10H      ; Chọn bộ định thời Timer0, chế độ 1 (16 bít)
AGAIN:  MOV   TL1, #1AH       ; Gán giá trị byte thấp TL1 = 1AH
        MOV   TH1, #0FFH      ; Gán giá trị byte cao TH1 = FFH
        SETB  TR1             ; Khởi động Timer1
BACK:   JNB   TF1, BACK       ; giữ nguyên cho đến khi bộ định thời quay về 0
        CLR   TR1             ; Dừng bộ định thời.
        CPL   P1.5            ; Bù bít P1.5 để nhận giá trị cao, thấp.
        CLR   TF1             ; Xoá cờ TF1
        SUMP AGAIN            ; Nạp lại bộ định thời vì chế độ 1 không tự nạp
                                lại.
```

Ví dụ 9.12:

Trước hết ta thực hiện các bước sau:

a) Tính chu kỳ sóng vuông: $T = \frac{1}{50Hz} = 20\mu s$

b) Tính thời gian nửa chu kỳ cho phần cao: $\frac{1}{2}T = 10\mu s$

c) Tính số nhịp đồng hồ: $n = \frac{10\mu s}{1{,}085\mu s} = 9216$

d) Tính giá trị cần nạp vào TH và TL: 65536 - 9216 = 56320 chuyển về dạng Hex là DC00H và TH = DCH và TL = 00H.

```
        MOV   TMOD, #10H      ; Chọn bộ định thời Timer0, chế độ 1 (16 bít)
AGAIN:  MOV   TL1, #00        ; Gán giá trị byte thấp TL1 = 00
        MOV   TH1, #0DHCH     ; Gán giá trị byte cao TH1 = DC
        SETB  TR1             ; Khởi động Timer1
BACK:   JNB   TF1, BACK       ; giữ nguyên cho đến khi bộ định thời quay về 0
        CLR   TR1             ; Dừng bộ định thời.
        CPL   P2.3            ; Bù bít P1.5 để nhận giá trị cao, thấp.
        CLR   TF1             ; Xoá cờ TF1
        SUMP AGAIN            ; Nạp lại bộ định thời vì chế độ 1 không tự nạp
                                lại.
```

9.1.4.3 Tạo một độ trễ thời gian lớn.

Như ta đã biết từ các ví dụ trên là lượng thời gian trễ cần tạo ra phụ thuộc vào hai yếu tố:

a) Tần số thạch anh XTAL

b) Thanh ghi 16 bít của bộ định thời ở chế độ 1

Cả hai yếu tố này nằm ngoài khả năng điều chỉnh của lập trình viên 8051. Ví như ta đã biết giá trị lớn nhất của độ trễ thời gian có thể đạt được bằng cách đặt cả TH và TL

bằng 0. Nhưng điều này xảy ra khi như vậy đều không đủ? Ví dụ 9.13 dưới đây cách làm thế nào để có giá trị độ trễ thời gian lớn.

9.1.4.4 Sử dụng bàn tính của Windows để tìm TH và TL.

Bàn tính Calculator của Windows có ngay trong máy tính PC của chúng ta và rất dễ sử dụng để tìm ra các giá trị cho TH và TL. Giả sử tìm giá trị cho TH và TL với độ trễ thời gian lớn là 35.000 nhịp đồng hồ với chu kỳ 1,085μs. Ta thực hiện các bước như sau:

1. Chọn máy tính Calculator từ Windows và đặt chế độ tính về số thập phân Decimal.
2. Nhập số 35.000 vào từ bàn phím.
3. Chuyển về chế độ Hex trên Calculator nó cho ta giá trị 88B8H.
4. Chọn +/- để nhận số đổi dấu - 35.000 dạng thập phân và chuyển về dạng Hex là 7748H.
5. Hai số hex cuối là cho TL = 48 và hai số Hex tiếp theo là cho TH = 77. Ta bỏ quan các số F ở phía bên phải trên Calculator vì số của ta là 16 bít.

Ví dụ 9.13:

Hãy kiểm tra chương trình sau và tìm độ trễ thời gian theo giây, không tính đến tổng phí các lệnh trong vòng lặp.

```
        MOV   TMOD, #10H     ; Chọn bộ Timer1, chế độ 1 (16 bít)
AGAIN:  MOV   R3, #200       ; Chọn bộ đếm độ giữ chậm lớn
        MOV   TL1, #08       ; Nạp byte thấp TL1 = 08
        MOV   TH1, #08       ; Nạp byte cao TH1 = 01
        SETB  TR1            ; Khởi động Timer1
BACK:   JNB   TF1, BACK      ; giữ nguyên cho đến khi bộ định thời quay về 0
        CLR   TR1            ; Dừng bộ định thời.
        CLR   TF1            ; Xoá cờ bộ định thời TF1
        DJNZ  R3, AGAIN      ; Nếu R3 không bằng không thì nạp lại bộ định
                               thời.
```

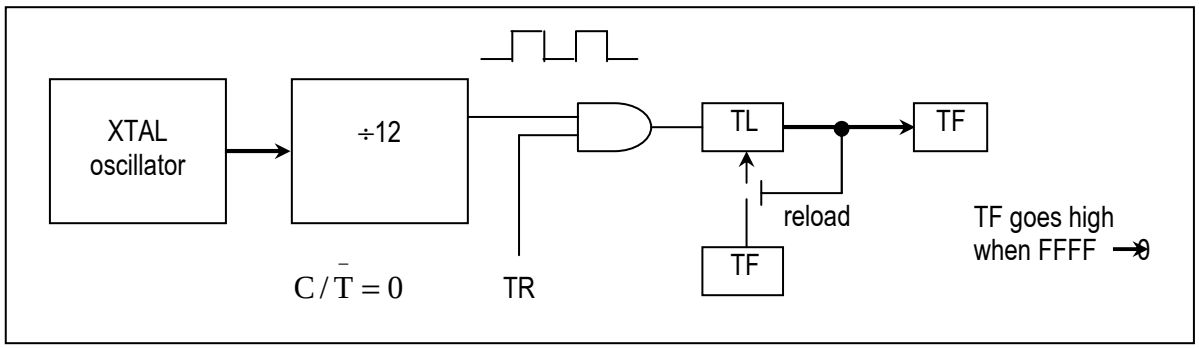
9.1.5 Chế độ O.

Chế độ 0 hoàn toàn giống chế độ 1 chỉ khác là bộ định thời 16 bít được thay bằng 13 bít. Bộ đếm 13 bít có thể giữ các giá trị giữa 0000 đến 1FFFF trong TH - TL. Do vậy khi bộ định thời đạt được giá trị cực đại của nó là 1FFFH thì nó sẽ quay trở về 0000 và cờ TF được bật lên.

9.1.6 Lập trình chế độ 2.

Các đặc trưng và các phép tính của chế độ 2:

1. Nó là một bộ định thời 8 bít, do vậy nó chỉ cho phép các giá trị từ 00 đến FFH được nạp vào thanh ghi TH của bộ định thời.
2. Sau khi TH được nạp với giá trị 8 bít thì 8051 lấy một bản sao của nó đưa vào TL. Sau đó bộ định thời phải được khởi động. Điều này được thực hiện bởi lệnh "SETB TR0" đối với Timer0 và "SETB TR1" đối với Timer1 giống như ở chế độ 1.
3. Sau khi bộ định thời được khởi động, nó bắt đầu đếm tăng lên bằng cách tăng thanh ghi TL. Nó đếm cho đến khi đại giá trị giới hạn FFH của nó. Khi nó quay trở về 00 từ FFH, nó thiết lập cờ bộ định thời TF. Nếu ta sử dụng bộ định thời Timer0 thì đó là cờ TF0, còn Timer1 thì đó là cờ TF1.



4. Khi thanh ghi TL quay trở về 00 từ FFH thì TF được bật lên 1 thì thanh ghi TL được tự động nạp lại với giá trị ban đầu được giữ bởi thanh ghi TH. Để lặp lại quá trình chúng ta đơn giản chỉ việc xoá cờ TF và để cho nó chạy mà không cần sự can thiệp của lập trình viên để nạp lại giả trị ban đầu. Điều này làm cho chế độ 2 được gọi là chế độ từ nạp lại so với chế độ 1 thì ta phải nạp lại các thanh ghi TH và TL.

Cần phải nhấn mạnh rằng, chế độ 2 là bộ định thời 8 bít. Tuy nhiên, nó lại có khả năng tự nạp khi tự nạp lại thì TH thực chất là không thay đổi với giá trị ban đầu được giữ nguyên, còn TL được nạp lại giá trị được sao từ TH. Chế độ này có nhiều ứng dụng bao gồm việc thiết lập tần số baud trong truyền thông nối tiếp như ta sẽ biết ở chương 10.

9.1.5.1 Các bước lập trình cho chế độ 2.

Để tạo ra một thời gian trễ sử dụng chế độ 2 của bộ định thời cần thực hiện các bước sau:

1. Nạp thanh ghi giá trị TMOD để báo bộ định thời gian nào (Timer0 hay Timer1) được sử dụng và chế độ làm việc nào của chúng được chon.
2. Nạp lại các thanh ghi TH với giá trị đếm ban đầu.
3. Khởi động bộ định thời.
4. Duy trì hiển thị cờ bộ định thời TF sử dụng lệnh "JNB TFx, đích" để xem nó sẽ được bật chưa. Thoát vòng lặp khi TF lên cao.
5. Xoá cờ TF.
6. Quay trở lại bước 4 vì chế độ 2 là chế độ tự nạp lại.

Ví dụ 9.14 minh hoạ những điều này. Để có được độ chễ lớn chúng ta có thể dùng nhiều thanh ghi như được chỉ ra trong ví dụ 9.15.

Ví dụ 9.14:

Giả sử tần số XTAL = 11.0592MHz. Hãy tìm a) tần số của sóng vuông được tạo ra trên chân P1.0 trong chương trình sau và b) tần số nhỏ nhất có thể có được bằng chương trình này và giá trị TH để đạt được điều đó.

```
        MOV   TMOD, #20H      ; Chọn Timer1/ chế độ 2/ 8 bít/ tự nạp lại.
        MOV   TH1, #5         ; TH1 = 5
        SETB  TR1             ; Khởi động Timer1
BACK:   JNB   TF1, BACK       ; giữ nguyên cho đến khi bộ định thời quay về 0
        CPL   P1.0            ; Dừng bộ định thời.
        CLR   TF1             ; Xoá cờ bộ định thời TF1
        SJMP  BACK            ; Chế độ 2 tự động nạp lại.
```

Lời giải:

a) Trước hết để ý đến đích của lệnh SJMP. Trong chế độ 2 ta không cần phải nạp lại TH vì nó là chế độ tự nạp. Bây giờ ta lấy (256 - 05).1.085μs = 251×1.085μs = 272.33μs là phần cao của xung. Cả chu kỳ của xung là T = 544.66μs và tần số là $\frac{1}{T} = 1{,}83597\text{kHz}$.

b) Để nhận tần số nhỏ nhất có thể ta cần tạo T chu kỳ lớn nhất có thể có nghĩa là TH = 00. Trong trường hợp này ta có $T = 2 \times 256 \times 1.085\mu s = 555.52\mu s$ và tần số nhỏ nhất sẽ là $\frac{1}{T} = 1{,}8\text{kHz}$.

Ví dụ 9.15:

Hãy tìm tần số của xung vuông được tạo ra trên P1.0.

Lời giải:

```
        MOV   TMOD, #2H      ; Chọn Timer0, chế độ 1 (8 bít tự nạp lại)
AGAIN:  MOV   TH0, #0        ; Nạp TH0 = 00
        MOV   R5, #250       ; Đếm cho độ trễ lớn
        ACALL DELAY
        CPL   P1.0
        SJMP  AGAIN
DELAY:  SETB  TR0            ; Khởi động Timer0
BACK:   JNB   TF1, BACK      ; giữ nguyên cho đến khi bộ định thời quay về 0
        CLR   TR0            ; Dừng Timer0.
        CLR   TF0            ; Xoá cờ TF0 cho vòng sau.
        DJNZ  R5, DELAY
        RET
```

$T = 2 \times (250 \times 256 \times 1.085\mu s) = 1.38.88\text{ms}$ và f = 72Hz.

Ví dụ 9.16:

Giả sử ta đang lập trình chế độ 2 hãy tìm các giá trị (dạng Hex) cần nạp vào TH cho các trường hợp sau:

a) MOV TH1, #200 b) MOV TH0, #-60
c) MOV TH1, #-3 d) MOV TH1, #-12
e) MOV TH0, #-48

Lời giải:

Chúng ta có thể sử dụng bàn tính Calculator của Windows để kiểm tra kết quả được cho bởi trình hợp ngữ. Hãy chọn Calculator ở chế độ Decimal và nhập vào số 200. Sau đó chọn Hex, rồi ấn +/ - để nhận giá trị của TH. Hãy nhớ rằng chúng ta chỉ sử dụng đúng hai chữ số và bỏ qua phần bên trái vì dữ liệu chúng ta là 8 bít. Kết quả ta nhận được như sau:

Dạng thập phân	*Số bù hai (giá trị TH)*
- 200	38H
- 60	C4H
- 3	FDH
- 12	F4H
- 48	DOH

9.1.5.2 Các trình hợp ngữ và các giá trị âm.

Vì bộ định thời là 8 bít trong chế độ 2 nên ta có thể để cho trình hợp ngữ tính giá trị cho TH. Ví dụ, trong lệnh "MOV TH0, # - 100" thì trình hợp ngữ sẽ tính toán – 100 = 9C và gán TH = 9CH. Điều này làm cho công việc của chúng ta dễ dàng hơn.

Ví dụ 9.17:

Hãy tìm a) tần số sóng vuông được tạo ra trong đoạn mã dưới đây và độ đầy xung của sóng này.

```
        MOV   TMOD, #2H          ; Chọn bộ Timer0/ chế độ 2/ (8 bít, tự nạp lại).
        MOV   TH0, # - 150       ; Nạp TH0 = 6AH là số bù hai của  - 150
        SETB  TR1                ; Khởi động Timer1
AGAIN:  SETB P1.3                ; P1.3 = 1
        ACALL DELAY
        ACALL P1.3               ; P1.3 = 0
        ACALL DELAY
        SJMP AGAIN

        SETB  TR0                ; Khởi động Timer0
BACK:   JNB   TF0, BACK          ;giữ nguyên cho đến khi bộ định thời quay về 0
        CLR   TR0                ; Dừng Timer0
        CLR   TF0                ; Xoá cờ TF cho vòng sau.
        RET
```

Lời giải:

Để tìm giá trị cho TH ở chế độ 2 thì trình hợp ngữ cần thực hiện chuyển đổi số âm khi ta nhập vào. Điều này cũng làm cho việc tính toán trở nê dễ dàng. Vì ta đang sử dụng 150 xung đồng hồ, nên ta có thời gian trễ cho chương trình con DELAY là $150 \times 1.085\mu s$ và tần số là $f = \frac{1}{T} = 2{,}048\text{kHz}$.

Để ý rằng trong nhiều tính toán thời gian trễ ta đã bỏ các xung đồng hồ liên quan đến tổng phí các lệnh trong vòng lặp. Để tính toán chính xác hơn thời gian trễ và cả tần số ta đang cần phải đưa chúng vào. Nếu ta dùng một máy hiện sóng số và ta không nhận được tần số đúng như ta tính toán thì đó là do tổng phí liên quan đến các lệnh gọi trong vòng lặp.

Trong phần này ta đã dùng bộ định thời 8051 để tạo thời gian trễ. Tuy nhiên, công dụng mạnh hơn và sáng tạo hơn của các bộ định thời này là sử dụng chúng như các bộ đếm sự kiện. Chúng ta sẽ bàn về công dụng của bộ đếm này ở phần kế tiếp.

9.2 Lập trình cho bộ đếm.

Ở phần trên đây ta đã sử dụng các bộ định thời của 8051 để tạo ra các độ trễ thời gian. Các bộ định thời này cũng có thể được dùng như các bộ đếm các sự kiện xảy ra bên ngoài 8051. Công dụng của bộ đếm/ bộ định thời như bộ đếm sự kiện sẽ được tình bày ở phần này. Chừng nào còn liên quan đến công dụng củ bộ định thời như bộ đếm sự kiện thì mọi vấn đề mà ta nói về lập trình bộ định thời ở phần trước cũng được áp dụng cho việc lập trình như là một bộ đếm ngoại trừ nguồn tần số. Đối với bộ định thời/ bộ đếm khi dùng nó như bộ định thời thì nguồn tần số là tần số thạch anh của 8051. Tuy nhiên, khi nó được dùng như một bộ đếm thì nguồn xung để tăng nội dung các thanh ghi TH và TL là từ bên ngoài 8051. Ở chế độ bộ đếm, hãy lưu ý rằng các thanh ghi TMOD và TH, TL cũng giống như đối với bộ định thời được bàn ở phần trước, thậm chí chúng vẫn có cùng tên gọi. Các chế độ của các bộ định thời cũng giống nhau.

9.2.1 Bít C/T trong thanh ghi TMOD.

Xem lại phần trên đây về bít C/T trong thanh ghi TMOD ta thấy rằng nó quyết định nguồn xung đồng hồ cho bộ định thời. Nếu bít C/T = 0 thì bộ định thời nhận các xung đồng hồ từ bộ giao động thạch anh của 8051. Ngược lại, khi C/T = 1 thì bộ định thời được sử dụng như bộ đếm và nhận các xung đồng hồ từ nguồn bên ngoài của 8051. Do vậy, khi bít C/T = 1 thì bộ đếm lên, khi các xung được đưa đến chân 14 và 15. Các chân này có tên là T0 (đầu vào của bộ định thời Timer0) và T1 (đầu vào của bộ Timer1). Lưu ý rằng hai chân này thuộc về cổng P3. Trong trường hợp của bộ Timer0 khi C/T = 1 thì chân P3.4 cấp xung đồng hồ và bộ đếm tăng lên đối với mỗi xung đồng hồ đi đến từ chân này. Tương tự như vậy đói với bộ Timer1 thì khi C/T = 1 với mỗi xung đồng hồ đi đến từ P3.5 bộ đếm sẽ đếm tăng lên 1.

Bảng 9.1: Các chân cổng P3 được dùng cho Timer0 và Timer1.

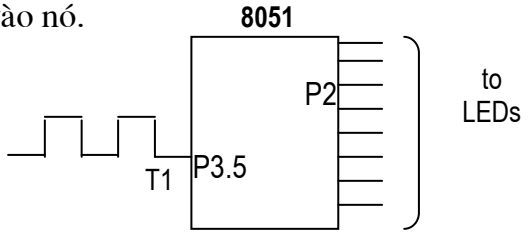
Chân	Chân cổng	Chức năng	Mô tả
14	P3.4	T0	Đầu vào ngoài của bộ đếm 0
15	P3.5	T1	Đầu vào ngoài của bộ đếm 1

Ví dụ 9.18:

Giả sử rằng xung đồng hồ được cấp tới chân T1, hãy viết chương trình cho bộ đếm 1 ở chế độ 2 để đếm các xung và hiển thị trạng thái của số đếm TL1 trên cổng P2.
Lời giải:

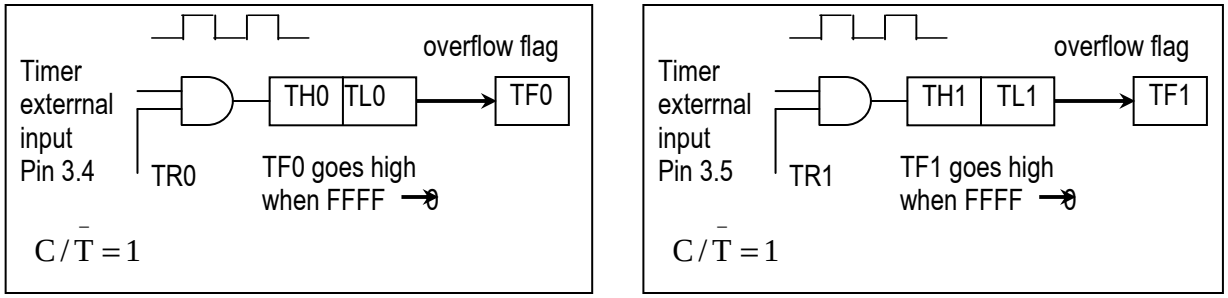
```
        MOV   TMOD, #01100000B   ; Chọn bộ đếm 1, chế độ 2, bít C/T = 1
                                   xung ngoài.
        MOV   TH1, #0            ; Xoá TH1
        SETB  P3.5               ; Lấy đầu vào T1
AGAIN:  SETB  TR1                ; Khởi động bộ đếm
BACK:   MOV   A, TL1             ; Lấy bản sao số đếm TL1
        MOV   P2, A              ; Đưa TL1 hiển thị ra cổng P2.
        JNB   TF1, Back          ; Duy trì nó nếu TF = 0
        CLR   TR1                ; Dừng bộ đếm
        CLR   TF1                ; Xoá cờ TF
        SJMP  AGAIN              ; Tiếp tục thực hiện
```

Để ý trong chương trình trên về vai trò của lệnh "SETB P3.5" vì các cổng được thiết lập dành cho đầu ra khi 8051 được cấp nguồn nên ta muốn P3.5 trở thành đầu vào thì phải bật nó lên cao. Hay nói cách khác là ta phải cấu hình (đưa lên cao) chân T1 (P3.5) để cho phép các xung được cấp vào nó.



Trong ví dụ 9.18 chúng ta sử dụng bộ Timer1 như bộ đếm sự kiện để nó đếm lên mỗi khi các xung đồng hồ được cấp đến chân P3.5. Các xung đồng hồ này có thể biểu diễn số người đi qua cổng hoặc số vòng quay hoặc bất kỳ sự kiện nào khác mà có thể chuyển đổi thành các xung.

Trong ví dụ 9.19 các thanh ghi TL được chuyển đổi về mã ASCII để hiển thị trên một LCD.



Hình 9.5: a) Bộ Timer0 với đầu vào ngoài (chế độ 1)
b) Bộ Timer1 với đầu vào ngoài (chế độ 1)

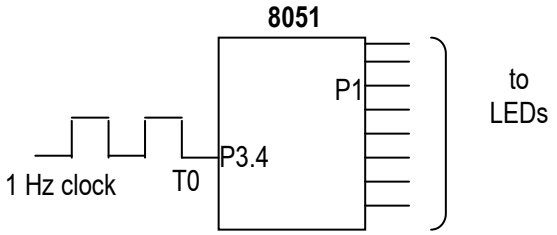
Ví dụ 9.19:

giả sử rằng một xung tần số 1Hz được nối tới chân đầu vào P3.4. Hãy viết chương trình hiển thị bộ đếm 0 trên một LCD. Hãy đặt số ban đầu của TH0 là - 60.

Lời giải:

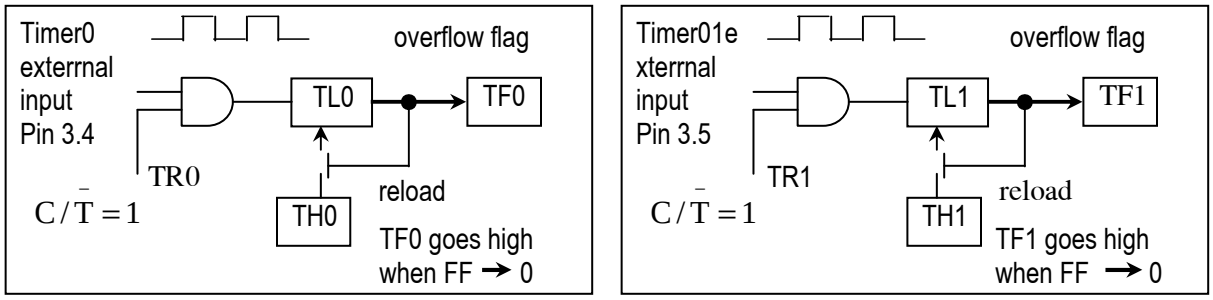
Để hiển thị số đếm TL trên một LCD ta phải thực hiện chuyển đổi giữ liệu 8 bít nhị phân về ASCII.

```
        ACALL   LCD-SET UP          ; Gọi chương trình con khởi tạo CLD
        MOV     TMOD, #000110B      ; Chọn bộ đếm 0, chế độ 2, bít C/T = 1
        MOV     TH0, # - 60         ; Đếm 60 xung
        SETB    P3.4                ; Lấy đầu vào T0
AGAIN:  SETB    TR0                 ; Sao chép số đếm TL0
BACK:   MOV     A, TL0              ; Gọi chương trình con để chuyển đổi
                                      trong các thanh ghi R2, R3, R4.
        ACALL   CONV                ; Gọi chương trình con hiển thị trên LCD
        ACALL   DISLAY              ; Thực hiện vòng lặp nếu TF = 0
        JNB     TF0, BACK           ; Dừng bộ đếm 0
        CLR     TR0                 ; Xoá cờ TF0 = 0
        CLR     TF0                 ; Tiếp tục thực hiện
        SJMP    AGAIN               ; Việc chuyển đổi nhị phân về mã ASCII
khi trả dữ liệu ASCII có trong các thanh ghi R4, R3, R2 (R2 có LSD) - chữ số nhỏ nhất.
CONV:  MOV      B, #10              ; Chia cho 10
        DIV     AB
        MOV     R2, B               ; Lưu giữ số thấp
        MOV     B, #10              ; Chia cho 10 một lần nữa
        DIV     AB
        ORL     A, #30H             ; Đổi nó về ASCII
        MOV     R4, A               ; Lưu chữ số có nghĩa lớn nhất MSD
        MOV     A, B                ;
        ORL     A, #30H             ; Đổi số thứ hai về ASCII
        MOV     R3, A               ; Lưu nó
        MOV     A, R2
        ORL     A, #30H             ; Đổi số thứ ba về ASCII
        MOV     R2, A               ; Lưu số ASCII vào R2.
        RET
```



Sử dụng tần số 60Hz ta có thể tạo ra các giây, phút, giờ.

Lưu ý rằng trong vòng đầu tiên, nó bắt đầu từ 0 vì khi RESET thì TL0 = 0; Để giải quyết vấn đề này hãy nạp TL0 với giá trị - 60 ở đầu chương trình.



Hình 9.6: Bộ Timer0 với đầu vào ngoài (chế độ 2)

Hình 9.7: Bộ Timer0 với đầu vào ngoài (chế độ 2)

Như một ví dụ ứng dụng khác của bộ định thời gian với bít C/T = 1, ta có thể nạp một sóng vuông ngoài với tần số 60Hz vào bộ định thời. Chương trình sẽ tạo ra các đơn vị thời gian chuẩn theo giây, phút, giờ. Từ đầu vào này ta hiển thị lên một LCD. Đây sẽ là một đồng hồ số tuyệt vời nhưng nó không thật chính xác. Ví dụ này có thể tìm thấy ở phụ lục Appendix E.

Trước khi kết thúc chương này ta cần nhắc lại hai vấn đề quan trọng.

1. Chúng ta có thể nghĩ rằng công dụng của lệnh "JNB TFx, đích" để hiển thị mức cao của cờ TF là một sự lãng phí thời gian của BVĐK. Điều đó đúng có một giải pháp cho vấn đề này là sử dụng các ngắt. Khi sử dụng các ngắt ta có thể đi thực hiện các công việc khác với BVĐK. Khi cờ TF được bật thì nó báo cho ta biết đây là điểm quan trọng về thế mạnh của 8051 (mà ta sẽ bàn ở chương 11).
2. Chúng ta muốn biết các thanh ghi TR0 và TR1 thuộc về đâu. Chúng thuộc về một thanh ghi gọi là TCON mã sẽ được ban sau ở đây (TCON - là thanh ghi điều khiển bộ đếm (bộ định thời)).

Bảng 9.2: Các lệnh tương đương đối với thanh ghi điều khiển bộ định thời.

Đối với Timer0	
	SETB TR0 = SETB TCON.4
	CLR TR0 = CLR TCON.4
	SETB TF = SETB TCON.5
	CLR TF0 = CLR TCON.5
Đối với Timer1	
	SETB TR1 = SETB TCON.6

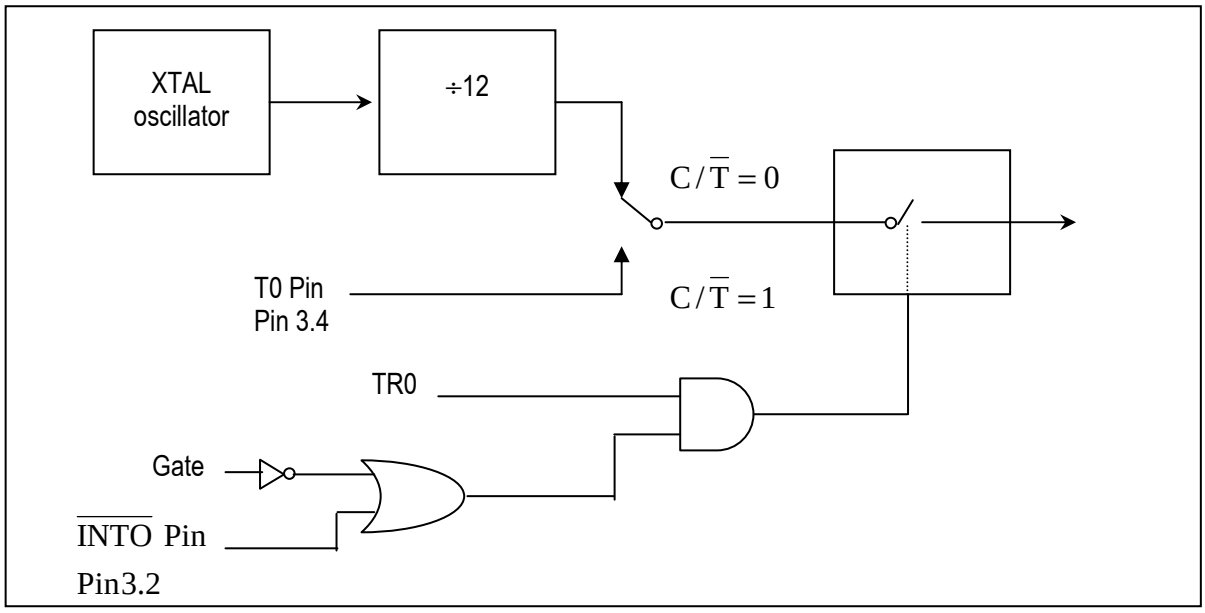
CLR TR1 = CLR TCON.6
SETB TF1 = SETB TCON.7
CLR TF1 = CLR TCON.7

9.2.2 Thanh ghi TCON.

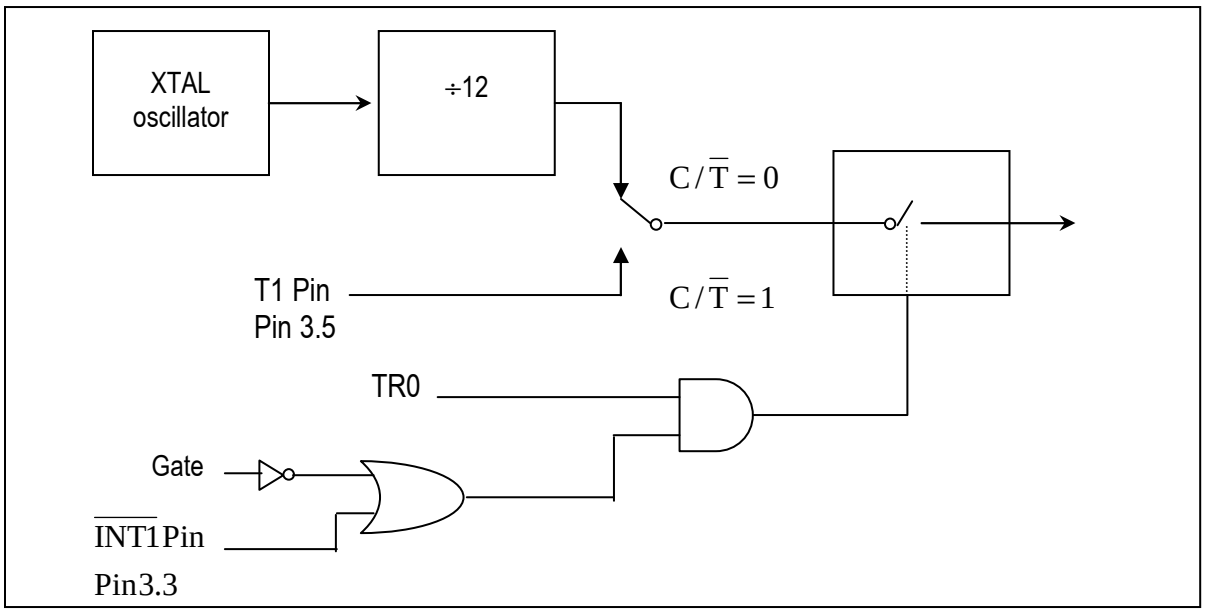
Trong các ví dụ trên đây ta đã thấy công dụng của các cờ TR0 và TR1 để bật/ tắt các bộ định thời. Các bít này là một bộ phận của thanh ghi TCON (điều khiển bộ định thời). Đây là thanh ghi 8 bít, như được chỉ ra trong bảng 9.2 thì bốn bít trên được dùng để lưu cất các bít TF và TR cho cả Timer0 và Timer1. Còn bốn bít thấp được thiết lập dành cho điều khiển các bít ngắt mà ta sẽ bàn ở chương 11. Chúng ta phải lưu ý rằng thanh ghi TCON là thanh ghi có thể đánh địa chỉ theo bít được. Nên ta có thể thay các lệnh như "SETB TR1" là "CLR TR1" bằng các lệnh tương ứng như "SET TCON.6" và "CLR TCON.6" (Bảng 9.2).

9.3 Trường hợp khi bít GATE = 1 trong TMOD.

Trước khi kết thúc chương ta cần bàn thêm về trường hợpợ khi bít GATE = 1trong thanh ghi TMOD. Tất cả những gì chúng ta vừa nói trong chương này đều giả thiết GATE = 0. Khi GATE = 0 thì bộ định thời được khởi động bằng các lệnh "SETB TR0" và "SETB TR1" đối với Timer0 và Timer1 tương ứng. Vậy điều gì xảy ra khi bít GATE = 1? Như ta có thể nhìn thây trên hình 9.8 và 9.9 thì nếu GATE = 1 thì việc khởi động và dừng bộ định thời được thực hiện từ bên ngoài qua chân P2.3 và P3.3 đối với Timer0 và Timer1 tương ứng. Mặc dù rằng TRx được bật lên bằng lệnh "SETB TRx" thì cũng cho phép ta khởi động và dừng bộ định thời từ bên ngoài tại bất kỳ thời điểm nào thông qua công tắc chuyển mạch đơn giản. Phương pháp điều khiển phần cứng để dừng và khởi động bộ định thời nay có thể có rất nhiều ứng dụng. Ví dụ, chẳng hạn 8051 được dùng trong một sản phẩm phát báo động mỗi giây dùng bộ Timer0 theo nhiều việc khác. Bộ Timer0 được bật lên bằng phần mềm qua lệnh "SETB TR0" và nằm ngoài sự kiểm soát của người dùng sản phẩm đó. Tuy nhiên, khi nối một công tắc chuyển mạch tới chân P2.3 ta có thể dừng và khởi động bộ định thời gian bằng cách đó để tắt báo động.



Hình 9.8: Bộ định thời/ bộ đếm 0.



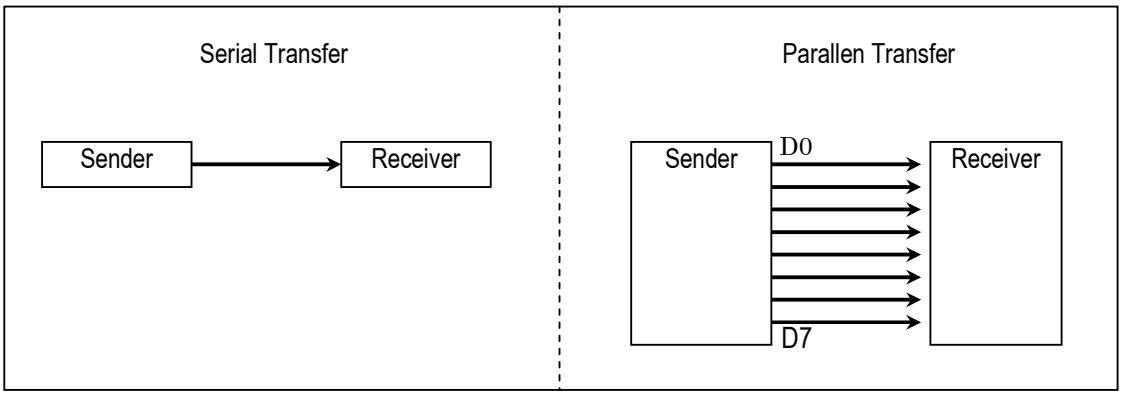
Hình 9.9: Bộ định thời/ bộ đếm 1.

CHƯƠNG 10
Truyền thông nối tiếp của 8051

Các máy tính truyền dữ liệu theo hai cách: Song song và nối tiếp. Trong truyền dữ liệu song song thường cần 8 hoặc nhiều đường dây dẫn để truyền dữ liệu đến một thiết bị chỉ cách xa vài bước. Ví dụ của truyền dữ liệu song song là các máy in và các ổ cứng, mỗi thiết bị sử dụng một đường cáp với nhiều dây dẫn. Mặc dù trong các trường hợp như vậy thì nhiều dữ liệu được truyền đi trong một khoảng thời gian ngắn bằng cách dùng nhiều dây dẫn song song nhưng khoảng cách thì không thể lớn được. Để truyền dudx liệu đi xa thì phải sử dụng phương pháp truyền nối tiếp. Trong truyền thông nối tiếp dữ liệu được gửi đi từng bít một so với truyền song song thì một hoặc nhiều byte được truyền đi cùng một lúc. Truyền thông nối tiếp của 8051 là chủ đề của chương này. 8051 đã được cài sẵn khả năng truyền thông nối tiếp, do vậy có thể truyền nhánh dữ liệu với chỉ một số ít dây dẫn.

10.1 Các cơ sở của truyền thông nối tiếp.

Khi một bộ vi xử lý truyền thông với thế giới bên ngoài thì nó cấp dữ liệu dưới dạng từng khúc 8 bít (byte) một. Trong một số trường hợp chẳng hạn như các máy in thì thông tin đơn giản được lấy từ đường bus dữ liệu 8 bít và được gửi đi tới bus dữ liệu 8 bít của máy in. Điều này có thể làm việc chỉ khi đường cáp bus không quá dài vì các đường cáp dài làm suy giảm thậm chí làm méo tín hiệu. Ngoài ra, đường dữ liệu 8 bít giá thường đắt. Vì những lý do này, việc truyền thông nối tiếp được dùng để truyền dữ liệu giữa hai hệ thống ở cách xa nhau hàng trăm đến hàng triệu dặm. Hình 10.1 là sơ đồ truyền nối tiếp so với sơ đồ truyền song song.



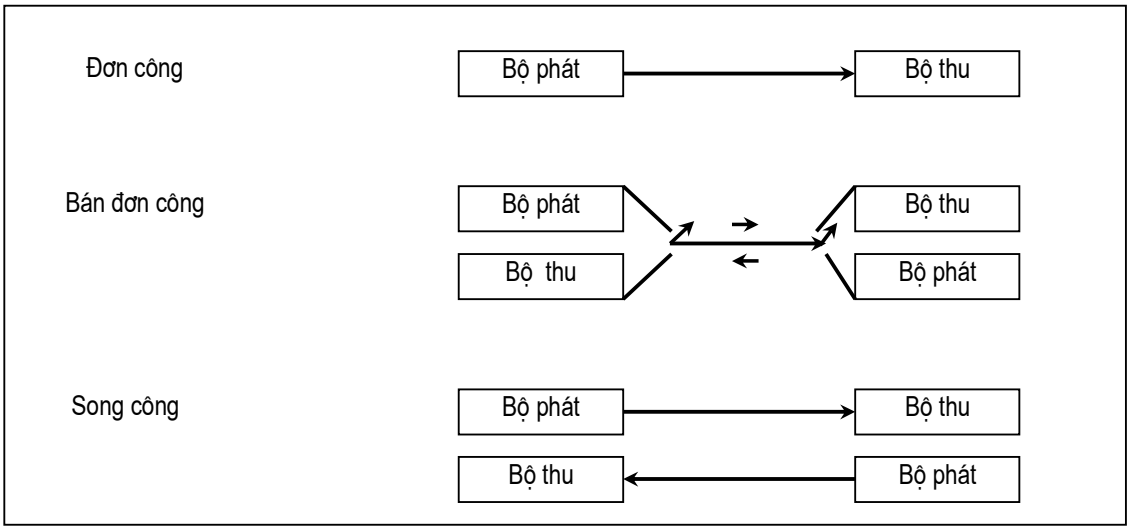
Hình 10.1: Sơ đồ truyền dữ liệu nối tiếp so với sơ đồ truyền song song.

Thực tế là trong truyền thông nối tiếp là một đường dữ liệu duy nhất được dùng thay cho một đường dữ liệu 8 bít của truyền thông song song làm cho nó không chỉ rẻ hơn rất nhiều mà nó còn mở ra khả năng để hai máy tính ở cách xa nhau có truyền thông qua đường thoại.

Đối với truyền thông nối tiếp thì để làm được các byte dữ liệu phải được chuyển đổi thành các bít nối tiếp sử dụng thanh ghi giao dịch vào - song song - ra - nối tiếp. Sau đó nó có thể được truyền quan một đường dữ liệu đơn. Điều này cũng có nghĩa là ở đầu thu cũng phải có một thanh ghi vào - nối tiếp - ra - song song để nhận dữ liệu nối tiếp và sau đó gói chúng thành từng byte một. Tất nhiên, nếu dữ liệu được truyền qua đường thoại thì nó phải được chuyển đổi từ các số 0 và 1 sang âm thanh ở dạng sóng hình sin. Việc chuyển đổi này thực thi bởi một thiết bị có tên gọi là Modem là chữ viết tắt của "Modulator/ demodulator" (điều chế/ giải điều chế).

Khi cự ly truyền ngắn thì tín hiệu số có thể được truyền như nói ở trên, một dây dẫn đơn giản và không cần điều chế. Đây là cách các bàn PC và IBM truyền dữ liệu đến bo mạch mẹ. Tuy nhiên, để truyền dữ liệu đi xa dùng các đường truyền chẳng hạn như đường thoại thì việc truyền thông dữ liệu nối tiếp yêu cầu một modem để điều chế (chuyển các số 0 và 1 về tín hiệu âm thanh) và sau đó giải điều chế (chuyển tín hiệu âm thanh về các số 0 và 1).

Truyền thông dữ liệu nối tiếp sử dụng hai phương pháp đồng bộ và dị bộ. Phương pháp đồng bộ truyền một khối dữ liệu (các ký tự) tại cùng thời điểm trong khi đó truyền dị bộ chỉ truyền từng byte một. Có thể viết phần mềm để sử dụng một trong hai phương pháp này, những chương trình có thể rất dài và buồn tẻ. Vì lý do này mà nhiều nhà sản xuất đã cho ra thị trường nhiều loại IC chuyên dụng phục vụ cho truyền thông dữ liệu nối tiếp. Những IC này phục vụ như các bộ thu - phát dị bộ tổng hợp VART (Universal Asynchronous Receiver Transmitter) và các bộ thu - phát đồng - dị bộ tổng hợp UBART (Universal Asynchronous Receiver Transmitter). Bộ vi điều khiển 8051 có một cài sẵn một UART mà nó sẽ được bàn kỳ ở mục 10.3.



Hình 10.2: Truyền dữ liệu đơn công, bán công và song công.

10.1.1 Truyền dữ liệu bán công và song công.

Trong truyền dữ liệu nếu dữ liệu có thể được vừa phát và vừa được thu thì gọi là truyền song công. Điều này tương phản với truyền đơn công chẳng hạn như các máy in chỉ nhận dữ liệu từ máy tính. Truyền song công có thể có hai loại là bán song công và song công hoàn toàn phụ thuộc vào truyền dữ liệu có thể xảy ra đồng thời không? Nếu dữ liệu được truyền theo một đường tại một thời điểm thì được gọi là truyền bán song công. Nếu dữ liệu có thể đi theo cả hai đường cùng một lúc thì gọi là song công toàn phần. Tất nhiên, truyền song công đòi hỏi hai đường dữ liệu (ngoài đường âm của tín hiệu), một để phát và một để thu dữ liệu cùng một lúc.

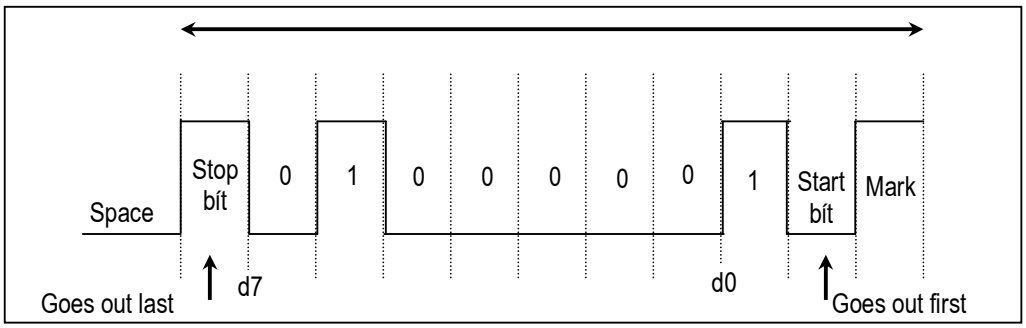
10.1.2 Truyền thông nối tiếp dị bộ và đóng khung dữ liệu.

Dữ liệu đi vào ở đầu thu của đường dữ liệu trong truyền dữ liệu nói tiếp toàn là các số 0 và 1, nó thật là khó làm cho dữ liệu ấy có nghĩa là nếu bên phát và bên thu không cùng thống nhất về một tệp các luật, một thủ tục, về cách dữ liệu được đóng gói, bao nhiêu bít tạo nên một ký tự và khi nào dữ liệu bắt đầu và kết thúc.

10.1.3 Các bít bắt đầu và dừng.

Truyền thông dữ liệu nối tiếp dị bộ được sử dụng rộng rãi cho các phép truyền hướng kỹ tự, còn các bộ truyền dữ liệu theo khối thì sử dụng phương phát đồng bộ.

Trong phương pháp dị bộ, mỗi ký tự được bố trí giữa các bít bắt đầu (start) và bít dừng (stop). Công việc này gọi là đóng gói dữ liệu. Trong đóng gói dữ liệu đối với truyền thông dị bộ thì dữ liệu chẳng hạn là các ký tự mã ASCII được đóng gói giữa một bít bắt đầu và một bít dừng. Bít bắt đầu luôn luôn chỉ là một bít, còn bít dừng có thể là một hoặc hai bít. Bít bắt đầu luôn là bít thấp (0) và các bít dừng luôn là các bít cao (bít 1). Ví dụ, hãy xét ví dụ trên hình 10.3 trong đó ký tự "A" của mã ASCII (8 bít nhị phân là 0100 0001) đóng gói khung giữa một bít bắt đầu và một bít dừng. Lưu ý rằng bít thấp nhất LSB được gửi ra đầu tiên.



Hình 10.3: Đóng khung một ký tự "A" của mã ASCII (41H) có tín hiệu là 1 (cao) được coi như là một dấu (mark), còn không có tín hiệu tức là 0 (thấp) thì được coi là khoảng trống (space). Lưu ý rằng phép truyền bắt đầu với start sau đó bít D0, bít thấp nhất LSB, sau các bít còn lại cho đến bít D7, bít cao nhất MSB và cuối cùng là bít dừng stop để báo kết thúc ký tự "A".

Trong truyền thông nối tiếp dị bộ thì các chíp IC ngoại vi và các modem có thể được lập trình cho dữ liệu với kích thước theo 7 bít hoặc 8 bít. Đây là chưa kể các bít dừng stop có thể là 1 hoặc 2 bít. Trong khi các hệ ASCII cũ hơn (trước đây) thì các ký tự là 7 bít thì ngay nay do việc mở rộng các ký tự ASCII nên dữ liệu nhìn chung là 8 bít. Trong các hệ cũ hơn do tốc độ chậm của các thiết bị thu thì phải sử dụng hai bít dừng để đảm bảo thời gian tổ chức truyền byte kế tiếp. Tuy nhiên, trong các máy tính PC hiện tại chỉ sử dụng 1 bít stop như là chuẩn.

Giả sử rằng chúng ta đang truyền một tệp văn bản các ký tự ASCII sử dụng 1 bít stop thì ta có tổng cộng là 10 bít cho mỗi ký tự gồm: 8 bít cho ký tự ASCII chuẩn và 1 bít start cùng 1 bít stop. Do vậy, đối với mỗi ký tự 8 bít thì cần thêm 2 bí vị chi là mất 25% tổng phí.

Trong một số hệ thống để nhằm duy trì tính toàn vẹn của dữ liệu thì người ta còn thêm một bít lẻ (parity bít). Điều này có nghĩa là đối với mỗi ký tự (7 hoặc 8 bít tuỳ từng hệ) ta có thêm một bít ngoài các bít start và stop. Bít chẵn lẻ là bít chẵn hoặc bít lẻ. Nếu là bít lẻ là số bít của dữ liệu bao gồm cả bít chẵn lẻ sẽ là một số lẻ các sô 1. Tương tự như vậy đối với trường hợp bít chẵn thì số bít của dữ liệu bao gồm cả bít chẵn - lẻ sẽ là một số chẵn của các số 1. Ví dụ, ký tự "A" của mã ASCII ở dạng nhị phân là 0100 0001, có bít 0 là bít chẵn. Các chíp UART đều cho phép việc lập trình bít chẵn - lẻ về chẵn, lẻ hoặc không phân biệt chẵn lẻ.

10.1.4 Tốc độ truyền dữ liệu.

Tốc độ truyền dữ liệu trong truyền thông dữ liệu nối tiếp được gọi là bít trong giây bps (bít per second). Ngoài ra, còn được sử dụng một thuật ngữ rộng rãi nữa là tốc độ baud. Tuy nhiên, các tốc baud và bps là hoàn toàn không bằng nhau. Điều này là do tốc baud là thuật ngữ của modem và được định nghĩa như là sô lần thay đổi của tín hiệu trong một giây. Trong các modem có những trường hợp khi một sự thay đổi của tín hiệu thì nó truyền vài bít dữ liệu. Nhưng đối với một dây dẫn thì tốc độ baud

và bps là như nhau nên trong cuốn sách này chúng ta có thể dùng thay đổi các thuật ngữ này cho nhau.

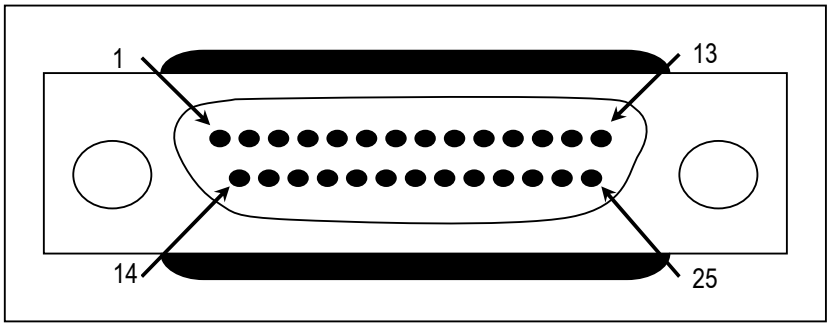
Tốc độ truyền dữ liệu của một hệ máy tính đã cho phụ thuộc vào các cổng truyền thông kết nối vào trong hệ thống đo. Ví dụ, các máy tính PC/XT trước đây của IBM có thể truyền dữ liệu với tốc độ 100 đến 9600 bps. Tuy nhiên, trong những năm gần đây thì các máy tính PC dựa trên Pentium truyền dữ liệu với tốc độ lên tới 56kbps. Cần phải nói thêm rằng trong truyền thông dữ liệu nối tiếp dị bộ thì tốc độ baud nhìn chung là bị giới hạn ở 100.000 bps.

10.1.5 Các chuẩn RS232.

Để cho phép tương thích giữa các thiết bị truyền thông dữ liệu được sản xuất bởi các hãng khác nhau thì một chuẩn giao diện được gọi là RS232 đã được thiết lập bởi hiệp hội công nghiệp điện tử EIA vào năm 19960. Năm 1963 nó được sửa chỉnh và được gọi là RS232A và vào các năm 1965 và 1969 thì được đổi thành RS232B và RS232C. Ở đây chúng ta đơn giản chỉ nói đến RS232. Ngày nay RS232 là chuẩn giao diện I/O vào - ra nối tiếp được sử dụng rộng rãi nhất. Chuẩn này được sử dụng trong máy tính PC và hàng loạt các thiết bị khác nhau. Tuy nhiên, vì nó được thiết lập trước họ lô-gíc TTL rất lâu do vậy điện áp đầu vào và đầu ra của nó không tương thích với mức TTL. Trong RS232 thì mức 1 được biểu diễn bởi - 3v đến 25v trong khi đó mức 0 thì ứng với điện áp + 3v đến +25v làm cho điện áp - 3v đến + 3v là không xác định. Vì lý do này để kết nối một RS232 bất kỳ đến một hệ vi điều khiển thì ta phải sử dụng các bộ biến đổi điện áp như MAX232 để chuyển đổi các mức lô-gíc TTL về mức điện áp RS232 và ngược lại. Các chíp IC MAX232 nhìn chung được coi như cá bộ điều khiển đường truyền. Kết nối RS232 đến MAX232 được thoả thuận ở phần 10.2.

10.1.6 Các chân của RS232.

Bảng 10.1 cung cấp sơ đồ chân của cáp RSE232 và các tên gọi của chúng thường được gọi là đầu nối DB - 25. Trong lý hiệu thì đầu nối cắm vào (đầu đực) gọi là DB - 25p và đầu nối cái được gọi là DB - 25s.



Hình 10.4: Đầu nối DB - 25 của RS232.

Vì không phải tất cả mọi chân đều được sử dụng trong cáp cảu máy tính PC, nên IBM đưa ra phiên bản của chuẩn vào/ra nối tiếp chỉ sử dụng có 9 chân gọi là DB - 9 như trình bày ở bảng 10:2 và hình 10.5.

Bảng 10.1: Các chân của RS232, 25 chân (DB - 25).

Số chân	Mô tả
1	Đất cách ly (Protective Cround)
2	Dữ liệu được truyền TxD (TráNsmitted data)
3	Dữ liệu được phân RxD (Received data)
4	Yêu cầu gửi RTS (Request To Send)
5	Xoá để gửi CIS (Clear To Send)
6	Dữ liệu sẵn sàng DSR (Data Set Ready)

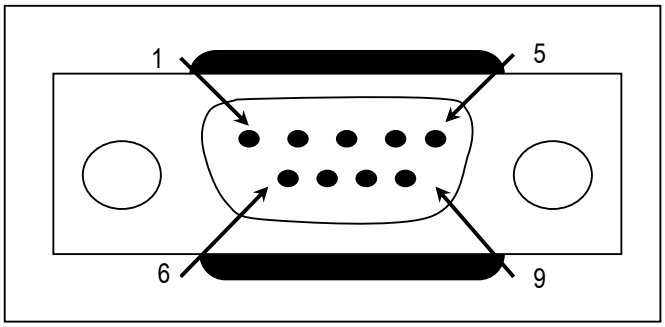
7	Đất của tín hiệu GND (Signal Cround)
8	Tách tín hiệu mạng dữ liệu DCD (Data Carrier Detect)
9/10	Nhận để kiểm tra dữ liệu (Received for data testing)
11	Chưa dùng
12	Tách tín hiệu mạng dữ liệu thứ cấp (Secondary data carrier detect)
13	Xoá đẻ nhận dữ liệu thứ cấp (Secondary Clear to Send)
14	Dữ liệu được truyền thứ cấp (Secondary Transmit Signal Element Timing)
15	Truyền phân chia thời gian phần tử tín hiệu (Transmit Signal Element
16	Timing)
17	Dữ liệu được nhận thứ cấp (Secondary Received data)
18	Nhận phân chia thời gian phần tử tín hiệu (Receiveo Signal Element
19	Timing)
20	Chưa dùng
21	Yêu cầu để nhận thứ cấp (Secondary Request to Send)
22	Đầu dữ liệu sẵn sàng (Data Terminal Ready)
23	Phát hiện chất lượng tín hiệu (Signal Qualyty Detector)
24	Báo chuông (Ring Indicator)
25	Chọn tốc độ tín hiệu dữ liệu (Data Signal Rate Select)
	Truyền phân chia thời gian tín hiệu (Transmit Signal Element Timing)
	Chưa dùng

10.1.7 Phân loại truyền thông dữ liệu.

Thuật ngữ hiện nay phân chia thiết bị truyền thông dữ liệu thành một thiết bị đầu cuối dữ liệu DTE (Data Terminal Equipment) hoặc thiết bị truyền thông dữ liệu DCE (Data Communication Equipment). DTE chủ yếu là các máy tính và các thiết bị đầu cuối gửi và nhận dữ liệu, còn DCE là thiết bị truyền thông chẳng hạn như các modem chịu trách nhiệm về truyền dữ liệu. Lưu ý rằng tất cả mọi định nghĩa về chức năng các chân RS232 trong các bảng 10.1 và 10.2 đều xuất phát từ gốc độ của DTE.

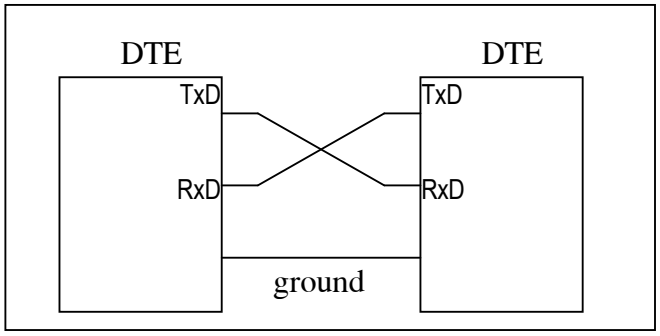
Kết nối đơn giản nhất giữa một PC và bộ vi điều khiển yêu cầu tối thiểu là những chân sau: TxD, RxD và đất như chỉ ra ở hình 10.6. Để ý rằng trên hình này thì các chân TxD và RxD được đổi cho nhau.

Hình 10.5: Sơ đồ đầu nối DB - 9 của RS232.



Bảng 10.2: Các tín hiệu của các chân đầu nối DB - 9 trên máy tính IBM PC.

Mô tả	Số chân	
1	Da ta carrier detect (DCD)	Tránh tín hiệu mạng dữ liệu
2	Received data (RxD)	Dữ liệu được nhận
3	Transmitted data (TxD)	Dữ liệu được gửi
4	Data terminal ready (DTR)	Đầu dữ liệu sẵn sàng
5	Signal ground (GND)	Đất của tín hiệu
6	Data set ready (DSR)	Dữ liệu sẵn sàng
7	Request to send (RTS)	Yêu cầu gửi
8	Clear to send (CTS)	Xoá để gửi
9	Ring indicator (RL)	Báo chuông



Hình 10.6: Nối kết không modem.

10.1.8 Kiểm tra các tín hiệu bắt tay của RS232.

Để bảo đảm truyền dữ liệu nhanh và tin cậy giữa hai thiết bị thì việc truyền dữ liệu phải được phối hợp tốt. Chẳng hạn như trong trường hợp của máy in, do một thực tế là trong truyền thông dữ liệu nối tiếp thiết bị thu có thể không có chỗ để chứa dữ liệu, do đó phải có cách để báo cho bên phát dừng gửi dữ liệu. Rất nhiều chân của RS232 được dùng cho các tín hiệu bắt tay. Dưới đây là mô tả về chúng như là một tham khảo và chúng có thể được bỏ qua vì chúng không được hỗ trợ bởi chíp UART của 8051.

1. Đầu dữ liệu sẵn sàng DTR: Khi thiết bị đầu cuối (hoặc một cổng COM của PC) được bật thì sau khi tự kiểm tra nó gửi một tín hiệu DTR báo rằng nó sẵn sàng cho truyền thông. Nếu có một cái gì đó trục trặc với cổng COM thì tín hiệu này không được kích hoạt. Đây là tín hiệu tích cực mức thấp và có thể được dùng để báo cho modem biết rằng máy tính đang hoạt động và đang sẵn sàng. Đây là chân đầu ra từ DTC (cổng COM của PC) và chân đầu ra của modem.
2. Dữ liệu sẵn sàng QSR: Khi DCE (chẳng hạn modem) được bật lên và đã chạy xong chương trình tự kiểm tra thì nó đòi hỏi DSR để báo rằng có đã sẵn sàng cho truyền thông. Do vậy, nó là đầu ra của modem (DCE) và đầu vào của PC (DTE). Đây là tín hiệu tích cực mức thấp. Nếu vì lý do nào đó mà modem không kích hoạt báo cho PC biết (hoặc thiết bị đầu cuối) rằng nó không thể nhận hoặc gửi dữ liệu.
3. Yêu cầu gửi RTS: Khi thiết bị DTE (chẳng hạn một PC) có một byte dữ liệu cần gửi thì nó yêu cầu RTS để báo cho modem biết rằng nó có một byte cần phải gửi đi. RTS là một đầu ra tích cực mức thấp từ DTE và một đầu vào tới modem.
4. Tín hiệu xáo để gửi CTS: Để đáp lại RTS thì khi modem có để chứa dữ liệu mà nó cần nhận thì nó gửi một tín hiệu CTS tới DTE (PC) để báo rằng bây giờ nó

có thể nhận dữ liệu. Tín hiệu đầu vào này tới DTE dùng để khởi động việc truyền dữ liệu.

5. Tách tín hiệu mang dữ liệu DCD: Modem yêu cầu tín hiệu DCD báo cho DTE biết rằng đã tách được một tín hiệu mang dữ liệu hợp lệ và rằng kết nối giữa nó và modem khác đã được thiết lập. Do vậy, DCD là một đầu ra của modem và đầu vào của PC (DTE).
6. Báo chuông RI: Một đầu ra từ modem (DCE) và một đầu vào tới máy tính PC (DTE) báo rằng điện thoại đang báo chuông. Nó tắt và bật đồng bộ với âm thanh đang đổ chuông. Trong 6 tín hiệu bắt tay thì tín hiệu này là ít được dùng nhất do một thực tế là các modem đã chịu trách nhiệm về trả lời điện thoại. Tuy nhiên, nếu trong một hệ thống đã cho mà PC phải chịu trách nhiệm trả lời điện thoại thì tín hiệu này có thể được dùng.

Từ mô tả trên thì việc truyền thông PC và modem có thể được tóm tắt như sau: Trong khi các tín hiệu DTR và DSR được dùng bởi PC và modem để báo rằng chúng đang hoạt động tốt thì các tín hiệu RTS và CTS thực tế đang kiểm tra luồng dữ liệu. Khi PC muốn gửi dữ liệu thì nó yêu cầu RTS và đáp lại, nếu modem sẵn sàng (có chỗ chứa dữ liệu) để nhận dữ liệu thì nó gửi lại tín hiệu CTS. Còn nếu không có chỗ cho dữ liệu thì modem không kích hoạt CTS và PC thôi không yêu cầu DTR và thử lại. Các tín hiệu RTS và CTS cũng được coi như tín hiệu luồng điều khiển phần cứng.

Đến đây kết thúc sự mô tả 9 chân quan trọng nhất của các tín hiệu bắt tay RS232 và các tín hiệu TxD, RxD và đất (Ground). Tín hiệu Ground này cũng được coi như là tín hiệu SG - đất của tín hiệu.

10.1.9 Các cổng COM của IBM PC và tương thích.

Các máy tính IBM PC và tương thích dựa trên các bộ vi xử lý × 86 (8086, 286, 384, 486 và Pentium) thường có hai cổng COM. Cả hai cổng COM đều có các đầu nối kiểu RS232. Nhiều máy tính PC sử dụng mỗi đầu nối một kiểu ổ cắm DB - 25 và DB - 9. Trong những năm gần đây, cổng COM1 được dùng cho chuột và COM2 được dùng cho các thiết bị chẳng hạn như Modem. Chúng ta có thể nối cổng nối tiếp của 8051 đến cổng COM2 của một máy tính PC cho các thí nghiệm về truyền thông nối tiếp.

Với nền kiến thức về truyền thông nối tiếp này chúng ta đã sẵn sàng làm việc với 8051.

10.2 Nối ghép 8051 tới RS232.

Như đã nói ở phần 10.1, chuẩn RS232 không tương thích với mức lô-gíc TTL, do vậy nó yêu cầu một bộ điều khiển đường truyền chẳng hạn như chíp MAX232 để chuyển đổi các mức điện áp RS232 về các mức TTL và ngược lại. Nội dung chính của phần này là bàn về nối ghép 8051 với các đầu nối RS232 thông qua chíp MAX232.

10.2.1 Các chân RxD và TxD trong 8051.

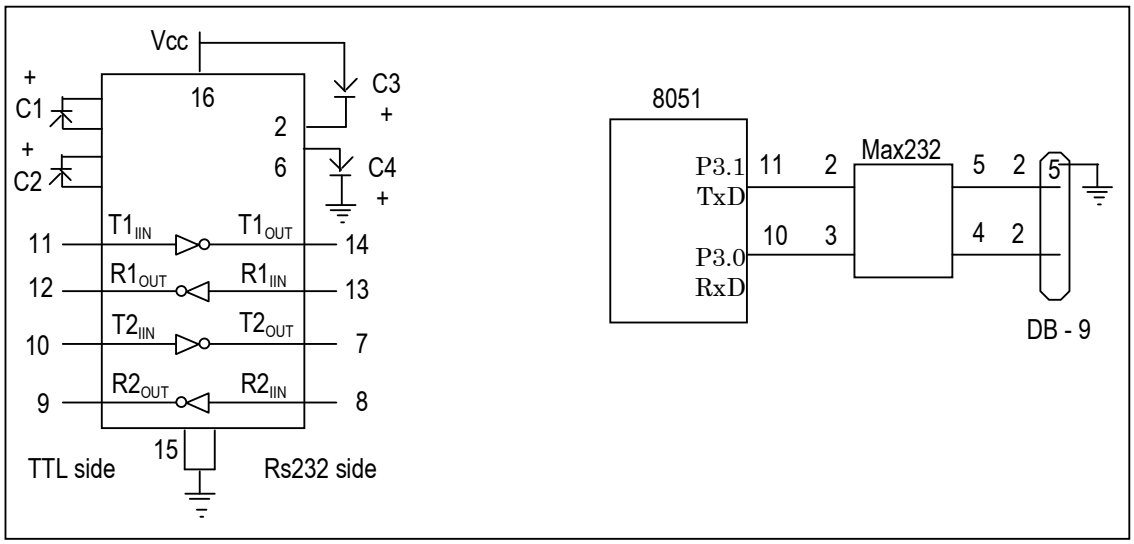
8051 có hai chân được dùng chuyên cho truyền và nhận dữ liệu nối tiếp. Hai chân này được gọi là TxD và RxD và là một phần của cổng P3 (đó là P3.0 và P3.1). chân 11 của 8051 là P3.1 được gán cho TxD và chân 10 (P3.0) được dùng cho RxD. Các chân này tương thích với mức lô-gích TTL. Do vậy chúng đòi hỏi một bộ điều khiển đường truyền để chúng tương thích với RS232. Một bộ điều khiển như vậy là chíp MAX232.

10.2.2 Bộ điều khiển đường truyền MAX232.

Vì RS232 không tương thích với các bộ vi xử lý và vi điều khiển hiện nay nên ta cần một bộ điều khiển đường truyền (bộ chuyển đổi điện áp) để chuyển đổi các tín hiệu RS232 về các mức điện áp TTL sẽ được chấp nhận bởi các chân TxD và RxD của 8051. Một ví dụ của một bộ chuyển đổi như vậy là chíp MAX232 từ hàng Maxim địa chỉ Website của hãng www.maxim-ic.com. Bộ MAX232 chuyển đổi từ các mức điện

áp RS232 sẽ về mức điện áp TTL và ngược lại. Một điểm mạnh của chíp MAX232 là nó dùng điện áp nguồng +5v cùng với điện áp nguồn của 8051. Hay nóic cách khác với nguồn điện áp nuối +5 chúng ta mà có thể nuôi 8051 và MAX232 mà không phải dùng hai nguồn nuôi khác nhau như phổ biến trong các hệ thống trước đây.

Bộ điều khiển MAX232 có hai bộ điều khiển thường để nhận và truyền dữ liệu như trình bày trên hình 10.7. Các bộ điều khiển đường được dùng cho TxD được gọi là T1 và T2. Trong nhiều ứng dụng thì chỉ có một cặp được dùng. Ví dụ T1 và R1 được dùng với nhau đối với TxD và RxD của 8051, còn cặp R2 và T2 thì chưa dùng đến. Để ý rằng trong MAX232 bộ điều khiển T1 có gán $T1_{in}$ và $T1_{out}$ trên các chân số 11 và 1 tương ứng. Chân $T1_{in}$ là ở phía TTL và được nối tới chân RxD của bộ vi điều khiển, còn $T1_{out}$ là ở phía RS232 được nối tới chân RxD của đầu nối DB của RS232. Bộ điều khiển đường R1 cũng có gán $R1_{in}$ và $R1_{out}$ trên các chân số 13 và 12 tương ứng. Chân $R1_{in}$ (chân số 13) là ở phía RS232 được nối tới chân TxD của đầu nối DB của RS232 và chân $R1_{out}$ (chân số 12) là ở phía TTL mà nó được nối tới chân RxD của bộ vi điều khiển, xem hình 10.7. Để ý rằng nối ghép modem không là nối ghép mà chân TxD bên phát được nối với RxD của bên thu và ngược lại.



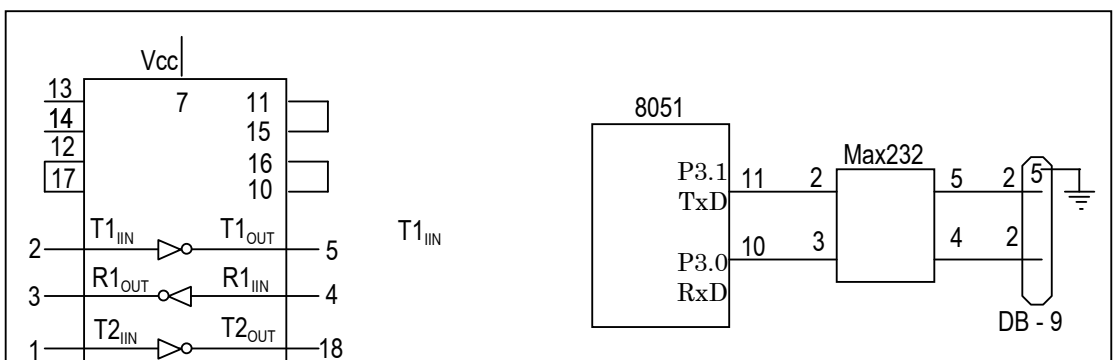
Hình 10.7: a) Sơ đồ bên trong của MAX232

b) Sơ đồ nối ghép của MAX232 với 8051 theo moden không.

Bộ MAX232 đòi hỏi 4 tụ điện giá trị từ 1 đến 22µF. Giá trị phổ biến nhất cho các tụ này là 22µF.

10.2.3 Bộ điều khiển MAX232.

Để tiết kiệm không gian trên bảng mạch, nhiều nhà thiết kế sử dụng chíp MAX232 từ hãng Maxim. Bộ điều khiển MAX232 thực hiện cùng những công việc như MAX232 lại không cần đến các tụ điện. Tuy nhiên, chíp MAX232 lại đắt hơn rất nhiều so với MAX233 không có sơ đồ chân giống nhau (không tương thích). Chúng ta không thể lấy một chíp MAX232 ra khỏi một bảng mạch và thay vào đó RS233. Hãy xem hình 10.8 để thấy MAX233 không cần đến tụ.



Hình 10.8: a) Sơ đồ bên trong của MAX233.
b) Sờ đồ nối ghép của MAX233 với 8051 theo modem không.

10.3 Lập trình truyền thông nối tiếp cho 8051.

Trong phần này chúng ta sẽ nghiên cứu về các thanh ghi truyền thông nối tiếp của 8051 và cách lập trình chúng để truyền và nhận dữ liệu nối tiếp. Vì các máy tính IBM PC và tương thích được sử dụng rất rộng rãi để truyền thông với các hệ dựa trên 8051, do vậy ta chủ yếu tập trung vào truyền thông nối tiếp của 8051 với cổng COM của PC. Để cho phép truyền dữ liệu giữa máy tín PC và hệ thống 8051 mà không có bất kỳ lỗi nào thì chúng ta phải biết chắc rằng tốc độ baud của hệ 8051 phải phù hợp với tốc độ baud của công COM máy tính PC được cho trong bảng 10.3. Chúng ta có thể kiểm tra các tốc độ baud này bằng cách vào chương trình Windows Terminal và bấm chuột lên tuỳ chọn Communication Settings. Chương trình Terminal.exe của Window3.1 cũng làm việc tốt trên Windows95 và Window98. Trong Window95 và cao hơn ta có thể sử dụng chức năng Hyperterrminal. Hàm Hyperterminal hỗ trợ các tốc độ Baud cao hơn nhiều so với các tốc độ cho trong bảng 10.3.

Bảng 10.3: Các tốc độ Baud của máy tính PC486 và Pentium cho trong BIOS.

100	150	300	600	1200	2400	4800	9600	19200

Ví dụ 10.1:

Với tần số XTAL là 11.0592MHz. Hãy tìm giá trị TH1 cần thiết để có tốc độ baud sau:

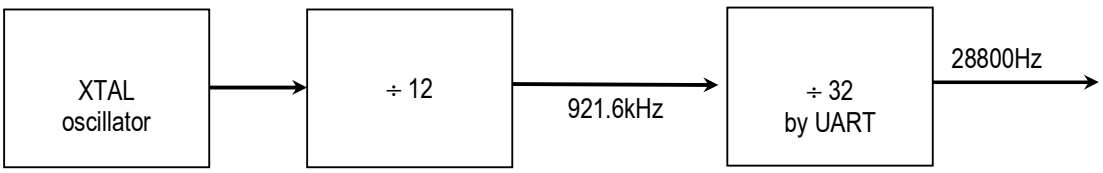
a) 9600 b) 2400 c) 1200

Lời giải:

Với tần số XTAL là 11.0592MHz thì ta có tần số chu trình máy của 8051 là 11.0592MHz : 12 = 921.6kHz và sau đó lấy 921.6kHz/32 = 28.800Hz là tần số được cấp bởi UART tới bộ định thời Timer1 để thiết lập tốc độ.

a) 28.800/3 = 9600 trong đó - 3 = FD được nạp vào TH1
b) 28.800/12 = 2400 trong đó - 12 = F4 được nạp vào TH1
c) 28.800/24 = 1200 trong đó - 24 = F8 được nạp vào TH1

Lưu ý rằng việc chia 1/12 của tần số thạch anh cho 32 là giá trị mặc định khi kích hoạt chân RESET của 8051. Chúng ta có thể thay đổi giá trị cài đặt mặc định này. Điều này sẽ được giải thích ở cuối chương.



10.3.1 Tốc độ baud trong 8051.

8051 truyền và nhận dữ liệu nối tiếp theo nhiều tốc độ khác nhau. Tốc độ truyền của nó có thể lập trình được. Điều ày thực hiện nhờ sự trợ giúp của bộ định thời Timer1. Trước khi ta đi vào bàn cách làm điều đó như thế nào thì ta sẽ xét quan hệ giữa tần số thạch anh và tốc độ baud trong 8051.

Như ta đã nói ở chương trước đây thì 8051 chia sóo thạch anh cho 12 để lấy tần số chu trình máy. Trong trường hợp XTAL = 11.0592MHz thì tần số chu trình là 921.6kHz (11.0592MHz : 12 = 921.6kHz). Mạch điện UART truyền thông nối tiếp của 8051 lại chia tần số chu trình máy cho 32 một lần nữa trước khi nó được dùng bởi bộ định thời gian Timer1 để tạo ra tốc độ baud. Do vậy, 921.6kHz : 32 = 28.800Hz. Đây là số ta sẽ dùng trong cả phần này để tìm giá trị của Timer1 để đặt tốc độ baud. Muốn Timer1 đặt tốc độ baud thì nó phải được lập trình về chế độ làm việc mode2, đó là chế độ thanh ghi 8 bít tự động nạp lại. Để có tốc độ baud tương thích với PC ta phải nạp TH1 theo các giá trị cho trong bảng 10.3. Ví dụ 10.1 trình bày cách kiểm tra giá trị dữ liệu cho trong bảng 10.3.

Bảng 10.3: Các giá trị của thanh ghi TH1 trong Timer1 cho các tốc độ baud khác nhau.

Tốc độ baud	TH1 (thập phân)	TH1 (số Hex)
9600	- 3	FD
4800	- 6	FA
2400	- 12	F4
1200	- 24	F8

10.3.2 Thanh ghi SBUF.

SBUF là thanh ghi 8 bít được dùng riêng cho truyền thông nối tiếp trong 8051. Đối với một byte dữ liệu cần phải được truyền qua đường TxD thì nó phải được đặt trong thanh ghi SBUF. Tương tự như vậy SBUF giữ một byte dữ liệu khi nó được nhận bởi đường RxD của 8051. SBUF có thể được truy cập bởi mọi thanh ghi bất kỳ trong 8051. Xét một ví dụ dưới đây để thấy SBUF được truy cập như thế nào?

```
MOV    SBUF, # "D"     ; Nạp vào SBUF giá trị 44H mã ACSII của ký tự D.
MOV    SBUF, A         ; Sao thanh ghi A vào SBUF.
MOV    A, SBUF         ; Sao SBUF vào thanh ghi A.
```

Khi một byte được ghi vào thanh ghi SBUF nó được đóng khung với các bít Start và Stop và đường truyền nối tiếp quan chân TxD. Tương tự như vậy, khi các bít được nhận nối tiếp từ RxD thì 8051 mở khung nó để loại trừ các bít Start và Stop để lấy ra một byte từ dữ liệu nhận được và đặt nó vào thanh ghi SBUF.

10.3.3 Thanh ghi điều khiển nối tiếp SCON.

Thanh ghi SCON là thanh ghi 8 bít được dùng để lập trình việc đóng khung bít bắt đầu Start, bít dừng Stop và các bít dữ liệu cùng với việc khác.

Dưới đây là mô tả các bít khác nhau của SCON:

SM0	SM1	SM2	REN	TB8	RB8	T1	R1

SM0	SCON.7	Số xác định chế độ làm việc cổng nối tiếp
SM1	SCON.6	Số xác định chế độ làm việc cổng nối tiếp
SM2	SCON.5	Dùng cho truyền thông giữa các bộ vi xử lý (SM2 = 0)
REN	SCON.4	Bật/xoá bằng phần mềm để cho phép/ không cho thu
TB8	SCON.3	Không sử dụng rộng rãi
RB8	SCON.2	Không sử dụng rộng rãi
T1	SCON.1	Cờ ngắt truyền, đặt bằng phần cứng khi bắt đầu bít Stop ở chế

Hình 10.2: Thanh ghi điều khiển cổng nối tiếp SCON.

10.3.3.1 Các bít SM0, SM1.

Đây là các bít D7 và D6 của thanh ghi SCON. Chúng được dùng để xác định chế độ đóng khung dữ liệu bằng cách xác định số bít của một ký tự và các bít Start và Stop. Các tổ hợp của chúng là:

SM0	*SM1*	
0	0	Chế độ nối tiếp 0
0	1	Chế độ nối tiếp 1, 8 bít dữ liệu, Start, Stop
1	0	Chế độ nối tiếp 2
1	1	Chế độ nối tiếp 3

Trong bốn chế độ ta chỉ quan tâm đến chế độ 1, các chế độ khác được giải thích ở Appendisk A3. Trong thanh ghi SCON khi chế độ 1 được chọn thì dữ liệu được đóng khung gồm 8 bít dữ liệu, 1 bít Start, 1 bít Stop để tương thích với cổng COM của IBM PC và các PC tương thích khác. Quan trọng hơn là chế độ nối tiếp 1 cho phép tốc độ baud thay đổi và được thiết lập bởi Timer1 của 8051. Trong chế độ nối tiếp 1 thì mỗi ký tự gồm có 10 bít được truyền trong đó có bít đầu là bít Start, sau đó là 8 bít dữ liệu và cuối cùng là bít Stop.

10.3.3.2 Bít SM2.

Bít SM2 là bít D5 của thanh ghi SCON. Bít này cho phép khả năng đa xử lý của 8051 và nó nằm ngoài phạm vi trình bày của chương này. Đối với các ứng dụng của chúng ta đặt SM2 = 0 vì ta không sử dụng 8051 trong môi trường đa xử lý.

10.3.3.3 Bít REN.

Đây là bít cho phép thu (Receive Enable), bít D4 của thanh ghi SCON. Bít REN cũng được tham chiếu như là SCON.4 vì SCON là thanh ghi có thể đánh địa chỉ theo bít. Khi bít REN cao thì nó cho phép 8051 thu dữ liệu trên chân RxD của nó. Và kết quả là nếu ta muốn 8051 vừa truyền và nhận dữ liệu thì bít REN phải được đặt lên 1. Khi đặt REN thì bộ thu bị cấm. Việc đặt REN = 1 hay REN = 0 có thể đạt được bằng lệnh "SETB SCON.4" và "CLR SCON.4" tương ứng. Lưu ý rằng các lệnh này sử dụng đặc điểm đánh địa chỉ theo bít của thanh ghi SCON. Bít này có thể được dùng để khống chế mọi việc nhận dữ liệu nối tiếp và nó là bít cực kỳ quan trọng trong thanh ghi SCON.

10.3.3.4 Bít TB8 và RB8.

Bít TB8 là bít SCON.3 hay là bít D3 của thanh ghi SCON. Nó được dùng để cho chế độ nối tiếp 2 và 3. Ta đặt TB8 vì nó không được sử dụng trong các ứng dụng của mình.

Bít RB8 (bít thu 8) là bít D2 của thanh ghi SCON. Trong chế độ nối tiếp 1 thì bít này nhận một bản sao của bít Stop khi một dữ liệu 8 bít được nhận. Bít này cũng

như bít TB8 rất hiếm khi được sử dụng. Trong các ứng dụng của mình ta đặt RB8 = 0 vì nó được sử dụng cho chế độ nối tiếp 2 và 3.

10.3.3.5 Các bít TI và RI.

Các bít ngắt truyền TI và ngắt thu RI là các bít D1 và D0 của thanh ghi SCON. Các bít này là cực kỳ quan trọng của thanh ghi SCON. Khi 8051 kết thúc truyền một ký tự 8 bít thì nó bật TI để báo rằng nó sẵn sàng truyền một byte khác. Bít TI được bật lên trước bít Stop. Còn khi 8051 nhận được dữ liệu nối tiếp qua chân RxD và nó tách các bít Start và Stop để lấy ra 8 bít dữ liệu để đặt vào SBUF, sau khi hoàn tất nó bật cờ RI để báo rằng nó đã nhận xong một byte và cần phải lấy đi kẻo nó bị mất cờ RI được bật khi đang tách bít Stop. Trong các ví dụ dưới đây sẽ nói về vai trò của các bít TI và RI.

10.3.4 Lập trình 8051 để truyền dữ liệu nối tiếp.

Khi lập trình 8051 để truyền các byte ký tự nối tiếp thì cần phải thực hiện các bước sau đây:

1. Nạp thanh ghi TMOD giá trị 204 báo rằng sử dụng Timer1 ở chế độ 2 để thiết lập chế độ baud.
2. Nạp thanh ghi TH1 các giá trị cho trong bảng 10.4 để thiết lập chế độ baud truyền dữ liệu nối tiếp (với giả thiết tần số XTAL = 11.0592MHz).
3. Nạp thanh ghi SCON giá trị 50H báo chế độ nối tiếp 1 để đóng khung 8 bít dữ liệu, 1 bít Start và 1 bít Stop.
4. Bật TR1 = 1để khởi động Timer1.
5. Xoá bít TI bằng lệnh "CLR TI"
6. Byte ký tự cần phải truyền được ghi vào SBUF.
7. Bít cờ TI được hiển thị bằng lệnh "JNB TI, xx" để báo ký tự đã được truyền hoàn tất chưa.
8. Để truyền ký tự tiếp theo quay trở về bước 5.

Ví dụ 10.2 trình bày chương trình để truyền nối tiếp với tốc độ 4800 baud. Ví dụ 10.3 trình bày cách truyền liên tục chữ "YES".

Ví dụ 10.2:

Hãy viết chương trình cho 8051 để truyền nối tiếp một ký tự "A" với tốc độ 4800 baud liên tục.

Lời giải:

```
        MOV   TMOD, #20H      ; Chọn Timer1, chế độ 2 (tự động nạp lại)
        MOV   TH1, # - 6      ; Chọn tốc độ 4800 baud
        MOV   SCON, #A"       ; Truyền 8 bít dữ liệu, 1 bít Stop cho phép thu
        SETB  TR1             ; Khởi động Timer1
AGAIN:  MOV   SBUF, #"A"      ; Cần truyền ký tự "A"
HERE:   JNB   TI, HERE        ; Chờ đến bít cuối cùng
        CLR   TI              ; Xoá bít TI cho ký tự kế tiếp
        SJMP  AGAIN           ; Tiếp tục gửi lại chữ A
```

Ví dụ 10.3:

Hãy viết chương trình để truyền chữ "YES" nối tiếp liên tục với tốc độ 9600 baud (8 bít dữ liệu, 1 bít Stop).

Lời giải:

```
        MOV   TMOD, #20H      ; Chọn bộ Timer1, chế độ 2
        MOV   TH1, # - 3      ; Chọn tốc độ 9600 baud
        MOV   SCON, #50H      ; Truyền 8 bít dữ liệu, 1 bít Stop cho phép thu
        SETB  TR1             ; Khởi động Timer1
AGAIN:  MOV   A, # "Y"        ; Truyền ký tự "Y"
        ACALL TRANS
```

```
        MOV     A, # "E"            ; Truyền ký tự "E"
        ACALL   TRANS
        MOV     A, # "S"            ; Truyền ký tự "S"
        ACALL   TRANS
        SJMP    AGAIN               ; Tiếp tục
; Chương trình con truyền dữ liệu nối tiếp.
TRANS:  MOV     SBUF, A             ; Nạp SBUF
HERE:   JNB     TI, HERE            ; Chờ cho đến khi truyền bít cuối cùng
        CLR     TI                  ; Chờ sẵn cho một byte kế tiếp
        RET
```

10.3.4.1 Tầm quan trọng của cờ TI.

Để hiểu tầm quan trọng của cờ ngắt TI ta hãy xét trình tự các bước dưới đây mà 8051 phải thực hiện khi truyền một ký tự quan đường TxD:

1. Byte ký tự cần phải truyền được ghi vào SBUF.
2. Truyền bít Start
3. Truyền ký tự 8 bít lần lượt từng bít một.
4. Bít Stop được truyền xong, trong quá trình truyền bít Stop thì cờ TI được bật (TI = 1) bởi 8051 để báo sẵn sàng để truyền ký tự kế tiếp.
5. Bằng việ hiển thị cờ TI ta biết chắc rằng ta không nạp quá vào thanh ghi SBUF. Nếu ta nạp một byte vào SBUF trước ghi TI được bật thì phần dữ liệu của byte trước chưa truyền hết sẽ bị mất. Hay nói cách khác là 8051 bật cờ TI khi đã truyền xong một byte và nó sẵn sàng để truyền byte kế tiếp.
6. Sau khi SBOF được nạp một byte mới tì cờ nhằm để có thể truyền byte mới này.

Từ phần trình bày trên đây ta kết luận rằng bằng việc kiểm tra bít cờ ngắt TI ta biết được 8051 có sẵn sàng để truyền một byte khác không. Quan trọng hơn cần phải nói ở đây là bít cờ TI được bật bởi từ 8051 khi nó hoàn tất việc truyền một byte dữ liệu, còn việc xoá nó thì phải được lập trình viên thực hiện bằng lệnh "CLR TI". Cũng cần lưu ý rằng, nếu ta ghi một byte vào thanh ghi SBUF trước khi cờ TI được bật thì sẽ có nguy cơ mất phần dữ liệu đang truyền. Bít cờ TI có thể kiểm tra bằng lệnh "JNB TI …" hoặc có thể sử dụng ngắt như ta sẽ thấy trong chương 11.

10.3.5 Lập trình 8051 để nhận dữ liệu.

Trong lập trình của 8051 để nhận các byte ký tự nối tiếp thì phải thực hiện các bước sau đây.

1. Nạp giá trị 20H vào thanh ghi TMOD để báo sử dụng bộ Timer1, chế độ 2 (8 bítm, tự động nạp lại) để thiết lập tốc độ baud.
2. Nạp TH1 các giá trị cho trong bảng 10.4 để tạo ra tốc độ baud với giả thiết XTAL = 10.0592MHz.
3. Nạp giá trị 50H vào thanh ghi SCON để báo sử dụng chế độ truyền nối tiếp 1 là dữ liệu được đóng gói bởi 8 bít dữ liệu, 1 bít Start và 1 bít Stop.
4. Bật TR1 = 1 để khởi động Timer1.
5. Xoá cờ ngắt RI bằng lệnh "CLR RI"
6. Bít cờ RI được hiển thị bằng lệnh "JNB RI, xx" để xem toàn bộ ký tự đã được nhận chưa.
7. Khi RI được thiết lập thì trong SBUF đã có 1 byte. Các nội dung của nó được cất lưu vào một nơi an toàn.
8. Để nhận một ký tự tiếp theo quay trở về bước 5.

Ví dụ 10.4:

Hãy lập trình cho 8051 để nhận các byte dữ liệu nối tiếp và đặt chúng vào cổng P1. Đặt tốc độ baud là 4800, 8 bít dữ liệu và 1 bít Stopl.

Lời giải:

```
        MOV     TMOD, #20H          ; Chọn bộ Timer1, chế độ 2 (tự động nạp lại)
        MOV     TH1, # - 6          ; Chọn tốc độ 4800 baud
        MOV     SCON, #50H          ; Chọn khung dữ liệu 8 bít Stop, bít.
        SETB    TR1                 ; Khởi động bộ Timer1
HERE:   JNB     R1, HERE            ; Đợi nhận toàn bộ lý tự vào hết
        MOV     A, SBUF             ; Lưu cất ký tự vào thanh A
        MOV     P1, A               ; Gửi ra cổng P.1
        CLR     RI                  ; Sẵn sàng nhận byte kế tiếp
        SJMP    HERE                ; Tiếp tục nhạn dữ liệu
```

Ví dụ 10.5:

Giả sử cổng nối tiếp của 8051 được nối vào cổng COM của máy tính IBM CP và mà đang sử chương trình Termina. Exe để gửi và nhận dữ liệu nối tiếp. Cổng P1 và P2 của 8051 được nố tới các đầ LED và các công tắc chuyển mạch tương ứng. Hãy viết một chương trrình cho 8051.

a) Gửi thông báo "We Are Ready" (chúng tôi đã sẵn sàng) tới máy tính PC.

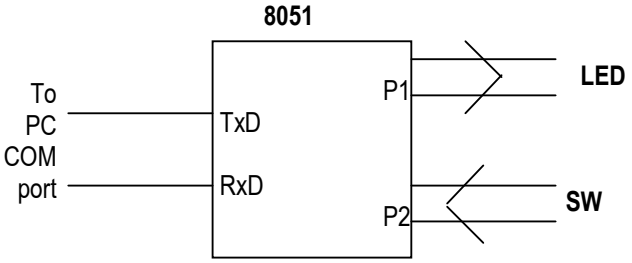
b) Nhận bất kỳ dữ liệu gì được PC gửi đến và chuyển đến các đèn LED đang nối đến các chân của cổng P1.

c) Nhận dữ liệu trên các chuyển mạch được nối tới P2 và gửi nó tới máy tính PC nối tiếp. Chương trình phải thực hiện một lần a), nhưng b) và c) chạy liên tục với tốc độ 4800 baud.

Lời giải:

```
                ORG     0
                MOV     P2, #0FFH           ; Lấy cổng P2 làm cổng vào
                MOV     TMOD, #20H          ; Chọn bộ Timer1, chế độ 2 (tự động nạp lại)
                MOV     TH1, # 0FAH         ; Chọn tốc độ 4800 baud
                MOV     SCON, #50H          ; Tạo khung dữ liệu 8 bít, 1bít Stop cho phép
                                              REN.
                SETB    TR1                 ; Khởi động bộ Timer1
                MOV     DPTR, #MYDATA       ; Nạp con trỏ đến thông báo
H - 1:          CLR     A
                MOVC    A, 'A + DPTR        ; Lấy ký tự
                JZ      DPTR                ; Nếu ký tự cuối cùng muốn gửi ra
                ACALL   SEND                ; Nếu chưa thì gọi chương trình con SEND
                INC     DPTR                ; Chạy tiếp
                SJMP    H - 1               ; Quay lại vòng lặp
B - 1:          MOV     A, P2               ; Đọc dữ liệu trên cổng P2
                ACALL   RECV                ; Truyền nó nối tiếp
                ACALL   RECV                ; Nhận dữ liệu nối tiếp
                MOV     F1, A               ; Hiển thị nó ra các đền LED
                SJMP    B - 1               ; ở lại vòng lặp vô hạn
; ------------------ Truyền dữ liệu nối tiếp ACC có dữ liệu
SEND:           MOV     SBUF, A             ; Nạp dữ liệu
H- 2:           JNB     TI, H - 2           ; ở lại vòng lặp vô hạn
                CLR     TI                  ; Truyền dữ liệu nối tiếp
                RET                         ; Nhận dữ liệu
; ------------------ Truyền dữ liệu nối tiếp ACC có dữ liệu
RECV:           JNB     RI, RECV            ; Nạp dữ liệu
                MOV     A, SBUF             ; ở lại đây cho đến khi gửi bít cuối cùng
                CLR     RI                  ; Sẵn sàng cho ký tự mới
                RET                         ; Trở về mời gọi
; ------------------ Nhận dữ liệu nối tiếp trong ACC
RECV:           JNB     RI, RECV            ; Đợi ở đây nhận ký tự
                MOV     A, SBUF             ; Lưu nó vào trong ACC
```

```
                CLR     RI                      ; Sẵn sàng nhận ký tự mã tiếp theo
                RET                             ; Trở về nơi gọi
; ------------------- Ngăn xếp chưa thông báo
MYDATA:         DB      "Chúng tôi đã sẵn sàng" 0
                END
```



10.3.5.1 Tầm quan trọng của cờ RT.

Khi nhận các bít quan chân RxD của nó thì 8051 phải đi quan các bước sau:

1. Nó nhận bít Start báo rằng bít sau nó là bít dữ liệu đầu tiên cần phải nhận.
2. Ký tự 8 bít được nhận lần lượt từng bít một. Khi bít cuối cùng được nhận thì một byte được hình thành và đặt vào trong SBUF.
3. Khi bít Stop được nhận thì 8051 bật RT = 1 để báo rằng toàn bộ ký tự được nhận và phải lấy đi trước khi nó bị byte mới nhận về ghi đè lên.
4. Bằng việc kiểm tra bít cờ RI khi nó được bật lên chúng ta biết rằng một ký tự đã được nhận và đang nằm trong SBUF. Tại sao nội dung SBUF vào nơi an toàn trong một thanh ghi hay bộ nhớ khác trước khi nó bị mất.
5. Sau khi SBUF được ghi vào nơi an toàn thì cờ RI được xoá về 0 bằng lệnh "CLR RI" nhằm cho các ký tự kế tiếp nhận được đưa vào SBUF. Nếu không làm được điều này thì gây ra mất ký tự vừa nhận được.

Từ mô tả trên đây ta rút ra kết luận rằng bằng việc kiểm tra cờ RI ta biết 8051 đã nhận được một byte ký tự chưa hay rồi. Nếu ta không sao được nội dung của thanh ghi SBUF vào nơi an toàn thì có nguy cơ ta bị mất ký tự vừa nhận được. Quan trọng hơn là phải nhớ rằng cờ RI được 8051 bật lên như lập trình viên phải xoá nó bằng lệnh "CLR RI". Cũng nên nhờ rằng, nếu ta sao nội dung SBUF vào nơi an toàn trước khi RI được bật ta mạo hiểm đã sao dữ liệu chưa đầy đủ. Bít cờ RI có thể được kiểm tra bởi lệnh "JNB RI, xx" hoặc bằng ngắt sẽ được bàn ở chương 11.

10.3.6 Nhân đôi tốc độ baud trong 8051.

Có hai cách để tăng tốc độ baud truyền dữ liệu trong 8051.

1. Sử dụng tần số thạch anh cao hơn.
2. Thay đổi một bít trong thanh ghi điều khiển công suất PCON (Power Control) như chỉ ra dưới đây.

D7							D0
SM0D	–	–	–	GF0	GF0	PD	IDL

Phương án một là không thực thi trong nhiều trường hợp vì tần số thạch anh của hệ thống là cố định. Quan trọng hơn là nó không khả thi vì tần số thạch anh mới không tương thích với tốc độ baud của các cổng COM nối tiếp của IBM PC. Do vậy, ta sẽ tập trung thăm dò phương án hai, có một cách nhân đôi tần số baud bằng phần mềm trong 8051 với tần số thạch anh không đổi. Điều này được thực hiện nhờ thanh ghi PCON, đây là thanh ghi 8 bít. Trong 8 bít này thì có một số bít không được dùng để điều khiển công suất của 8051. Bít dành cho truyền thông là D7, bít SMOD (chế độ nối tiếp - serial mode). Khi 8051 được bật nguồn thì bít SMOD của thanh ghi PCON ở

mức thấp 0. Chúng ta có thể đặt nó lên 1 bằng phần mềm và do vậy nhân đôi được tốc độ baud. Thứ tự các lệnh được sử dụng để thiết lập bít D7 của PCON lên cao như sau (thanh ghi PCON là thể đánh địa chỉ theo bít).

```
MOV   A, PCON          ; Đặt bản sao của PCON vào ACC
SETB  ACC.7            ; Đặt D7 của ACC lên 1.
MOV   PCON, A          ; Bây giờ SMOD = 1 mà không thay đổi bất kỳ bít nào khác.
```

Để biết tốc độ baud được tăng lên gấp đôi như thế nào bằng phương pháp này ta xét vai trò của bít SMOD trong PCON khi nó là 0 và 1.

a) Khi SMOD = 0.

Khi SMOD = 0 thì 8051 chia 1/12 tần số thạch anh cho 32 và sử dụng nó cho bộ Timer1 để thiết lập tốc độ baud. Trong trường hợp XTAL = 11.0592MHz thì ta có: Tần số chu trình máy $= \frac{11.0592\text{MHz}}{12} = 921.6\text{kHz}$ và $\frac{921.6\text{kHz}}{32} = 28.800\text{Hz}$ vì SMOD = 0.

Đây là tần số được Timer1 sử dụng để đặt tốc độ baud. Đây là cơ sở cho tất cả ví dụ từ trước đến giờ vì nó là giá trị mặc định của 8051 khi bật nguồn. Các tốc độ baud đối với SMOD = 0 được cho trong bảng 10.4.

b) Khi SMOD = 1.

Với tần số cố định thạch anh ta có thể nhân đôi tốc độ baud bằng cách đặt bít SMOD = 1. Khi bít D7 của PCON (bít SMOD) được đưa lên 1 thì 1/12 tần số XTAL được chia cho 16 (thay vì chia cho 32 như khi SMOD = 0) và đây là tần số được Timer dùng để thiết lập tốc độ baud. Trong trường hợp XTAL = 11.0592MHz ta có:

Tần số chu trình máy $= \frac{11.0592\text{MHz}}{12} = 921.6\text{kHz}$ và $\frac{921.6\text{kHz}}{16} = 57.600\text{kHz}$ vì SMOD = 1.

Đây là tần số mà Timer1 dùng để đặt tốc độ baud. Bảng 10.5 là các giá trị cần được nạp vào TH1 cùng với các tốc độ baud của 8051 khi SMOD = 0 và 1.

Bảng 10.5: So sánh tốc độ baud khi SMOD thay đổi.

TH1 (thập phân)	TH1 (Hex)	Tốc độ baud	
		SMOD = 0	SMOD = 1
-3	FD	9600	19200
-6	DA	4800	9600
-12	F4	2400	4800
-24	E8	1200	2400

Ví dụ 10.6:

Giả sử tần số XTAL = 11.0592MHz cho chương trình dưới đây, hãy phát biểu a) chương trình này làm gì? b) hãy tính toán tần số được Timer1 sử dụng để đặt tốc độ baud? và c) hãy tìm tốc độ baud truyền dữ liệu.

```
MOV   A, PCON          ; Sao nội dung thanh ghi PCON vào thanh ghi ACC
SETB  ACC.7            ; Đặt D7 = 0
MOV   PCON, A          ; Đặt SMOD = 1 để tăng gấp đôi tần số baud với tần số XTAL cố định
                       ;
MOV   TMOD, #20H       ; Chọn bộ Timer1, chế độ 2, tự động nạp lại
MOV   TH1, - 3         ; Chọn tốc độ baud 19200 (57600/3=19200) vì SMOD = 1
                       ;
MOV   SCON, #50H       ; Đóng khung dữ liệu gồm 8 bít dữ liệu, 1 Stop và cho phép RI.
```

```
        SETB    TR1             ; Khởi động Timer1
        MOV     A, #"B"         ; Truyền ký tự B
A-1:    CLR     TI              ; Khẳng định TI = 0
        MOV     SBUF, A         ; Truyền nó
H-1:    JNB     TI, H-1         ; Chờ ở đây cho đến khi bít cuối được gửi đi
        SJMP    A-1             ; Tiếp tục gửi "B"
```

Lời giải:

a) Chương trình này truyền liên tục mã ASCII của chữ B (ở dạng nhị phân là 0100 0010)

b) Với tần số XTAL = 11.0592MHz và SMOD = 1 trong chương trình trên ta có:

11.0592MHz/12 = 921.6kHz là tần số chu trình máy

921.6kHz/16 = 57.6kHz là tần số được Timer1 sử dụng để đặt tốc độ baud

c) 57.6kHz/3 = 19.200 là tốc độ cần tìm

Ví dụ 10.7:

Tìm giá trị TH1 (ở dạng thập phân và hex) để đạt tốc độ baud cho các trường hợp sau.

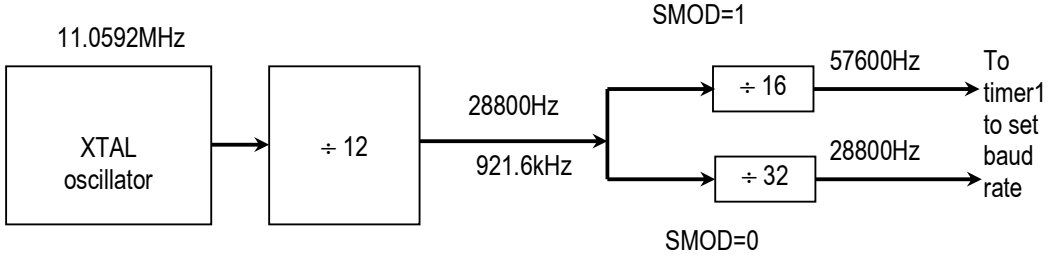
a) 9600 b) 4800 nếu SMOD = 1 và tần số XTAL = 11.0592MHz

Lời giải:

Với tần số XTAL = 11.0592MHz và SMOD = 1 ta có tần số cấp cho Timer1 là 57.6kHz.

a) 57.600/9600 = 6 do vậy TH1 = - 6 hay TH1 = FAH

b) 57.600/4800 = 12 do vậy TH1 = - 12 hay TH1 = F4H



Ví dụ 10.8:

Hãy tìm tốc độ baud nếu TH1 = -2, SMOD = 1 và tần số XTAL = 11.0592MHz. Tốc độ này có được hỗ trợ bởi các máy tính IBM PC và tương thích không?

Lời giải:

Với tần số XTAL = 11.0592MHz và SMOD = 1 ta có tần số cấp cho Timer1 là 57.6kHz. Tốc độ baud là 57.600kHz/2 = 28.800. Tốc độ này không được hỗ trợ bởi các máy tính IBM PC và tương thích. Tuy nhiên, PC có thể được lập trình để truyền dữ liệu với tốc độ như vậy. Phần mềm của nhiều modem có thể làm cho điều này và Hyperterminal của Windows 95 cũng có thể hỗ trợ tốc độ này và các tốc độ khác nữa.

10.3.7 Truyền dữ liệudựa trên các ngắt.

Ta phải thấy rằng thật lãng phí để các bộ vi điều khiển phải bật lên xuống các cờ TI và RI. Do vậy, để tăng hiệu suất của 8051 ta có thể lập trình các cổng truyền thông nối tiếp của nó bằng các ngắt. Đây chính là nội dung chính sẽ bàn luận ở chương 11 dưới đây.

CHƯƠNG 11
Lập trình các ngắt

Một ngắt là một sự kiện bên trong hoặc bên ngoài làm ngắt bộ vi điều khiển để báo cho nó biết rằng thiết bị cần dịch vụ của nó. Trong chương này ta tìm hiểu khái niệm ngắt và lập trình ngắt.

11.1 Các ngắt của 8051.

11.1.1 Các ngắt ngược với thăm dò.

Một bộ vi điều khiển có thể phục vụ một vài thiết bị, có hai cách để thực hiện điều này đó là sử dụng các ngắt và thăm dò (polling). Trong phương pháp sử dụng các ngắt thì mỗi khi có một thiết bị bất kỳ cần đến dịch vụ của nó thì nó bao cho bộ vi điều khiển bằng cách gửi một tín hiệu ngắt. Khi nhận được tín hiệu ngắt thì bộ vi điều khiển ngắt tất cả những gì nó đang thực hiện để chuyển sang phục vụ thiết bị. Chương trình đi cùng với ngắt được gọi là trình dịch vụ ngắt ISR (Interrupt Service Routine) hay còn gọi là trình quản lý ngắt (Interrupt handler). Còn trong phương pháp thăm dò thì bộ vi điều khiển hiển thị liên tục tình trạng của một thiết bị đã cho và điều kiện thoả mãn thì nó phục vụ thiết bị. Sau đó nó chuyển sang hiển thị tình trạng của thiết bị kế tiếp cho đến khi tất cả đều được phục vụ. Mặc dù phương pháp thăm dò có thể hiển thị tình trạng của một vài thiết bị và phục vụ mỗi thiết bị khi các điều kiện nhất định được thoả mãn nhưng nó không tận dụng hết cộng dụng của bộ vi điều khiển. Điểm mạnh của phương pháp ngắt là bộ vi điều khiển có thể phục vụ được rất nhiều thiết bị (tất nhiên là không tại cùng một thời điểm). Mỗi thiết bị có thể nhận được sự chú ý của bộ vi điều khiển dựa trên mức ưu tiên được gán cho nó. Đối với phương pháp thăm dò thì không thể gán mức ưu tiên cho các thiết bị vì nó kiểm tra tất cả mọi thiết bị theo kiểu hơi vòng. Quan trọng hơn là trong phương pháp ngắt thì bộ vi điều khiển cũng còn có thể che hoặc làm lơ một yêu cầu dịch vụ của thiết bị. Điều này lại một lần nữa không thể thực hiện được trong phương pháp thăm dò. Lý do quan trọng nhất là phương pháp ngắt được ưu chuộng nhất là vì phương pháp thăm dò làm lãng phí thời gian của bộ vi điều khiển bằng cách hỏi dò từng thiết bị kể cả khi chúng không cần đến dịch vụ. Nhằm để tránh ….. thì người ta sử dụng phương pháp ngắt. Ví dụ trong các bộ định thời được bàn đến ở chương 9 ta đã dùng lệnh "JNB TF, đích" và đợi cho đến khi bộ định thời quay trở về 0. Trong ví dụ đó, trong khi chờ đợi thì ta có thể làm việc được gì khác có ích hơn, chẳng hạn như khi sử dụng phương pháp ngắt thì bộ vi điều khiển có thể đi làm các việc khác và khi cờ TF bật lên nó sẽ ngắt bộ vi điều khiển cho dù nó đang làm bất kỳ điều gì.

11.1.2 Trình phục vụ ngắt.

Đối với mỗi ngắt thì phải có một trình phục vụ ngắt ISR hay trình quản lý ngắt. khi một ngắt được gọi thì bộ vi điều khiển phục vụ ngắt. Khi một ngắt được gọi thì bộ vi điều khiển chạy trình phục vụ ngắt. Đối với mỗi ngắt thì có một vị trí cố định trong bộ nhớ để giữ địa chỉ ISR của nó. Nhóm các vị trí nhớ được dành riêng để gửi các địa chỉ của các ISR được gọi là bảng véc tơ ngắt (xem hình 11.1).

11.1.3 Các bước khi thực hiện một ngắt.

Khi kích hoạt một ngắt bộ vi điều khiển đi qua các bước sau:

1. Nó kết thúc lệnh đang thực hiện và lưu địa chỉ của lệnh kế tiếp (PC) vào ngăn xếp.

2. Nó cũng lưu tình trạng hiện tại của tất cả các ngắt vào bên trong (nghĩa là không lưu vào ngăn xếp).
3. Nó nhảy đến một vị trí cố định trong bộ nhớ được gọi là bảng véc tơ ngắt nới lưu giữ địa chỉ của một trình phục vụ ngắt.
4. Bộ vi điều khiển nhận địa chỉ ISR từ bảng véc tơ ngắt và nhảy tới đó. Nó bắt đầu thực hiện trình phục vụ ngắt cho đến lệnh cuối cùng của ISR là RETI (trở về từ ngắt).
5. Khi thực hiện lệnh RETI bộ vi điều khiển quay trở về nơi nó đã bị ngắt. Trước hết nó nhận địa chỉ của bộ đếm chương trình PC từ ngăn xếp bằng cách kéo hai byte trên đỉnh của ngăn xếp vào PC. Sau đó bắt đầu thực hiện các lệnh từ địa chỉ đó.

Lưu ý ở bước 5 đến vai trò nhạy cảm của ngăn xếp, vì lý do này mà chúng ta phải cẩn thận khi thao tác các nội dung của ngăn xếp trong ISR. Đặc biệt trong ISR cũng như bất kỳ chương trình con CALL nào số lần đẩy vào ngăn xếp (Push) và số lần lấy ra từ nó (Pop) phải bằng nhau.

11.1.4 Sáu ngắt trong 8051.

Thực tế chỉ có 5 ngắt dành cho người dùng trong 8051 nhưng nhiều nhà sản xuất đưa ra các bảng dữ liệu nói rằng có sáu ngắt vì họ tính cả lệnh tái thiết lập lại RESET. Sáu ngắt của 8051 được phân bố như sau:

1. RESET: Khi chân RESET được kích hoạt từ 8051 nhảy về địa chỉ 0000. Đây là địa chỉ bật lại nguồn được bàn ở chương 4.
2. Gồm hai ngắt dành cho các bộ định thời: 1 cho Timer0 và 1 cho Timer1. Địa chỉ của các ngắt này là 000B4 và 001B4 trong bảng véc tơ ngắt dành cho Timer0 và Timer1 tương ứng.
3. Hai ngắt dành cho các ngắt phần cứng bên ngoài chân 12 (P3.2) và 13 (P3.3) của cổng P3 là các ngắt phần cứng bên ngoài INT0 và INT1 tương ứng. Các ngắt ngoài cũng còn được coi như EX1 và EX2 vị trí nhớ trong bảng véc tơ ngắt của các ngắt ngoài này là 0003H và 0013H gán cho INT0 và INT1 tương ứng.
4. Truyền thông nối tiếp có một ngắt thuộc về cả thu và phát. Địa chỉ của ngắt này trong bảng véc tơ ngắt là 0023H.

Chú ý rằng trong bảng 11.1 có một số giới hạn các byte dành riêng cho mỗi ngắt. Ví dụ, đối với ngắt INT0 ngắt phần cứng bên ngoài 0 thì có tổng cộng là 8 byte từ địa chỉ 0003H đến 000AH dành cho nó. Tương tự như vậy, 8 byte từ địa chỉ 000BH đến 0012H là dành cho ngắt bộ định thời 0 là TI0. Nếu trình phục vụ ngắt đối mặt với một ngắt đã cho mà ngắn đủ đặt vừa không gian nhớ được. Nếu không vừa thì một lệnh LJMP được đặt vào trong bảng véc tơ ngắt để chỉ đến địa chỉ của ISR, ở trường hợp này thì các byte còn lại được cấp cho ngắt này không dùng đến. Dưới đây là các ví dụ về lập trình ngắt minh hoạ cho các điều trình bày trên đây.

Từ bảng 11.1 cùng để ý thấy một thực tế rằng chí có 3 byte của không gian bộ nhớ ROM được gán cho chân RESET. Đó là những vị trí địa chỉ 0, 1 và 2 của ROM. Vị trí địa chỉ 3 thuộc về ngắt phần cứng bên ngoài 0 với lý do này trong chương trình chúng ta phaỉ đặt lệnh LJMP như là lệnh đầu tiên và hướng bộ xử lý lệnh khỏi bảng véc tơ ngắt như chỉ ra trên hình 11.1.

Bảng 11.1: Bảng véc tơ ngắt của 8051.

Ngắt	Địa chỉ ROM	Chân
Bật lại nguồn (RESET)	0000	9
Ngắt phần cứng ngoài (INT0)	0003	12 (P3.2)
Ngắt bộ Timer0 (TF0)	000B	
Ngắt phần cứng ngoài 1 (INT1)	0013	13 (P3.3)
Ngắt bộ Timer1 (TF1)	001B	
Ngắt COM nối tiếp (RI và TI)	0023	

11.1.5 Cho phép và cấm ngắt.

Khi bật lại nguồn thì tất cả mọi ngắt đều bị cấm (bị che) có nghĩa là không có ngắt nào sẽ được bộ vi điều khiển đáp ứng nếu chúng được kích hoạt. Các ngắt phải được kích hoạt bằng phần mềm để bộ vi điều khiển đáp ứng chúng. Có một thanh ghi được gọi là cho phép ngắt IE (Interrupt Enable) chịu trách nhiệm về việc cho phép (không che) và cấm (che) các ngắt. Hình 11.2 trình bày thanh ghi IE, lưu ý rằng IE là thanh ghi có thể đánh địa chỉ theo bít.

Từ hình 11.2 ta thấy rằng D7 của thanh ghi IE được gọi là bít cho phép tất cả các ngắt EA (Euable All). Bít này phải được thiết lập lên 1 để phần còn lại của thanh ghi hoạt động được. Bít D6 chưa được sử dụng. Bít D54 được dành cho 8051, còn bít D4 dùng cho ngắt nối tiếp v.v…

11.1.6 Các bước khi cho phép ngắt.

Để cho phép một ngắt ta phải thực hiện các bước sau:

1. Bít D7 của thanh ghi IE là EA phải được bật lên cao để cho phép các bít còn lại của thanh ghi nhận được hiệu ứng.
2. Nếu EA = 1 thì tất cả mọi ngắt đều được phép và sẽ được đáp ứng nếu các bít tương ứng của chúng trong IE có mức cao. Nếu EA = 0 thì không có ngắt nào sẽ được đáp ứng cho dù bít tương ứng của nó trong IE có giá trị cao.

Để hiểu điểm quan trong này hãy xét ví dụ 11.1.

Hình 11.2: Thanh ghi cho phép ngắt IE.

D7 D0

EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA IE.7 Nếu EA = 0 thì mọi ngắt bị cấm
Nếu EA = 1 thì mỗi nguồn ngắt được cho phép hoặc bị cấm bằng các bật hoặc xoá bít cho phép của nó.

- - IE.6 Dự phòng cho tương lai
ET2 IE.5 Cho phép hoặc cấm ngắt tràn hoặc thu của Timer2 (8051)
ES IE.4 Cho phép hoặc cấm ngắt cổng nối tiếp
ET1 IE.3 Cho phép hoặc cấm ngắt tràn của Timer1
EX1 IE.2 Cho phép hoặc cấm ngắt ngoài 1
ET0 IE.1 Cho phép hoặc cấm ngắt tràn của Timer0
EX0 IE.0 Cho phép hoặc cấm ngắt ngoài 0

* Người dùng không phải ghi 1 vào bít dự phòng này. Bít này có thể dùng cho các bộ vi điều khiển nhanh với đặc tính mới

Ví dụ 11.1:

Hãy chỉ ra những lệnh để a) cho phép ngắt nối tiếp ngắt Timer0 và ngắt phần cứng ngoài 1 (EX1) và b) cấm (che) ngắt Timer0 sau đó c) trình bày cách cấm tất cả mọi ngắt chỉ bằng một lệnh duy nhất.

Lời giải:

a) MOV IE, #10010110B ; Cho phép ngắt nối tiếp, cho phép ngắt Timer0 và cho phép ngắt phần cứng ngoài.

Vì IE là thanh ghi có thể đánh địa chỉ theo bít nên ta có thể sử dụng các lệnh sau đây để truy cập đến các bít riêng rẽ của thanh ghi:

```
SETB    IE.7        ; EA = 1, Cho phép tất cả mọi ngắt
SETB    IE.4        ; Cho phép ngắt nối tiếp
SETB    IE.1        ; Cho phép ngắt Timer1
SETB    IE.2        ; Cho phép ngắt phần cứng ngoài 1
```

(tất cả những lệnh này tương đương với lệnh "MOV IE, #10010110B" trên đây).

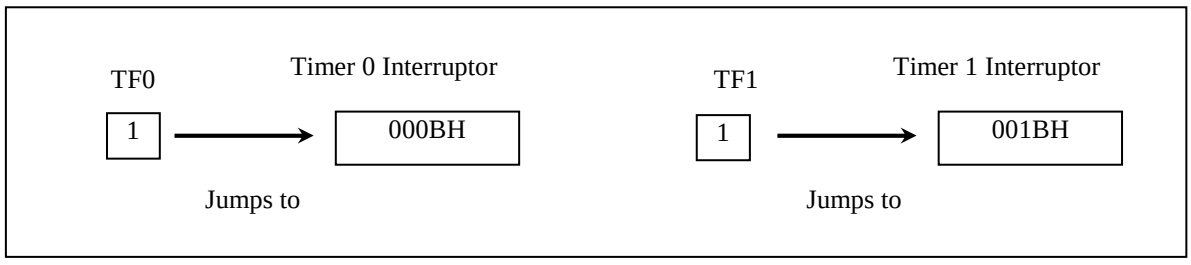
```
b) CLR  IE.1        ; Xoá (che) ngắt Timer0
c) CLR  IE.7        ; Cấm tất cả mọi ngắt.
```

11.2 Lập trình các ngắt bộ định thời.

Trong chương 9 ta đã nói cách sử dụng các bộ định thời Timer0 và Timer1 bằng phương pháp thăm dò. Trong phần này ta sẽ sử dụng các ngắt để lập trình cho các bộ định thời của 8051.

11.2.1 Cờ quay về 0 của bộ định thời và ngắt.

Trong chương 9 chúng ta đã nói rằng cờ bộ định thời TF được đặt lên cao khi bộ định thời đạt giá trị cực đại và quay về 0 (Roll - over). Trong chương trình này chúng ta cũng chỉ ra cách hiển thị cờ TF bằng lệnh "JNB TF, đích". Khi thăm dò cờ TF thì ta phải đợi cho đến khi cờ TF được bật lên. Vấn đề với phương pháp này là bộ vi điều khiển bị trói buộc khi cờ TF được bật lên và không thể làm được bất kỳ việc gì khác. Sử dụng các ngắt giải quyết được vấn đề này và tránh được sự trói buộc của bộ vi điều khiển. Nếu bộ ngắt định thời trong thanh ghi IE được phép thì mỗi khi nó quay trở về 0 cờ TF được bật lên và bộ vi điều khiển bị ngắt tại bất kỳ việc gì nó đang thực hiện và nhảy tới bảng véc tơ ngắt để phục vụ ISR. Bằng cách này thì bộ vi điều khiển có thể làm những công việc khác cho đến khi nào nó được thông báo rằng bộ định thời đã quay về 0. Xem hình 11.3 và ví dụ 11.2.



Hình 11.3: Ngắt bộ định thời TF0 và TF1.

Hãy để những điểm chương trình dưới đây của chương trình trong ví dụ 11.2.

1. Chúng ta phải tránh sử dụng không gian bộ nhớ dành cho bảng véc tơ ngắt. Do vậy, ta đặt tất cả mã khởi tạo tại địa chỉ 30H của bộ nhớ. Lệnh LJMP là lệnh đầu

tiên mà 8051 thực hiện khi nó được cấp nguồn. Lệnh LJMP lái bộ điều khiển tránh khỏi bảng véc tơ ngắt.

2. Trình phục vụ ISR của bộ Timer0 được đặt ở trong bộ nhớ bắt đầu tự địa chỉ 000BH và vì nó quá nhỏ đủ cho vào không gian nhớ dành cho ngắt này.
3. Chúng ta cho phép ngắt bộ Timer0 với lệnh "MOV IE, #1000 010H" trong chương trình chính MAIN.
4. Trong khi dữ liệu ở cổng P0 được nhận vào và chuyển liên tục sang công việc P1 thì mỗi khi bộ Timer0 trở về 0, cờ TF0 được bật lên và bộ vi điều khiển thoát ra khỏi vòng lặp BACK và đi đến địa chỉ 000BH để thực hiện ISR gắn liền với bộ Timer0.
5. Trong trình phục vụ ngắt ISR của Timer0 ta thấy rằng không cần đến lệnh "CLR TF0" trước khi lệnh RETI. Lý do này là vì 8051 xoá cờ TF bên trong khi nhảy đến bảng véc tơ ngắt.

Ví dụ 11.2:

Hãy viết chương trình nhân liên tục dữ liệu 8 bít ở cổng P0 và gửi nó đến cổng P1 trong khi nó cùng lúc tạo ra một sóng vuông chu kỳ 200μs trên chân P2.1. Hãy sử dụng bộ Timer0 để tạo ra sóng vuông, tần số của 8051 là XTAL = 11.0592MHz.

Lời giải:

Ta sử dụng bộ Timer0 ở chế độ 2 (tự động nạp lại) giá trị nạp cho TH0 là 100/1.085μs = 92.

```
; - - Khi khởi tạo vào chương trình main tránh dùng không gian.
; Địa chỉ dành cho bảng véc tơ ngắt.
                ORG     0000H
                CPL     P2.1                    ; Nhảy đến bảng véc tơ ngắt.
;
; - - Trình ISR dành cho Timer0 để tạo ra sóng vuông.
                ORG     0030H                   ; Ngay sau địa chỉ bảng véc-tơ ngắt
MAIN:           TMOD, #02H                      ; Chọn bộ Timer0, chế độ 2 tự nạp lại
                MOV     P0, #0FFH               ; Lấy P0 làm cổng vào nhận dữ liệu
                MOV     TH0, # - 92             ; Đặt TH0 = A4H cho – 92
                MOV     IE, #82H                ; IE = 1000 0010 cho phép Timer0
                SETB    TR0                     ; Khởi động bộ Timer0
BACK:           MOV     A, P0                   ; Nhận dữ liệu vào từ cổng P0
                MOV     P1, A                   ; Chuyển dữ liệu đến cổng P1
                SJMP    BACK                    ; Tiếp tục nhận và chuyển dữ liệu
                                                ; Chừng nào bị ngắt bởi TF0
                END
```

Trong ví dụ 11.2 trình phục vụ ngắt ISR ngắn nên nó có thể đặt vừa vào không gian địa chỉ dành cho ngắt Timer0 trong bảng véc tơ ngắt. Tất nhiên không phải lúc nào cũng làm được như vậy. Xét ví dụ 11.3 dưới đây.

Ví dụ 11.3:

Hãy viết lại chương trình ở ví dụ 11.2 để tạo sóng vuông với mức cao kéo dài 1085μs và mức thấp dài 15μs với giả thiết tần số XTAL = 11.0592MHz. Hãy sử dụng bộ định thời Timer1.

Lời giải:

Vì 1085μs là 1000 × 1085μs nên ta cần sử dụng chế độ 1 của bộ định thời Timer1.

```
; - - Khi khởi tạo tránh sử dụng không gian dành cho bảng véc tơ ngắt.
                ORG     0000H
                LJMP    MAIN                    ; Chuyển đến bảng véc tơ ngắt.
;
; - - Trình ISR đối với Timer1 để tạo ra xung vuông
                OR6     001BH                   ; Địa chỉ ngắt của Timer1 trong bảng véc tơ ngắt
                LJMP    ISR-T1                  ; Nhảy đến ISR
;
; - - Bắt đầu các chương trình chính MAIN.
                ORG     0030H                   ; Sau bảng véc tơ ngắt
MAIN:           MOV     TMOD, #10H              ; Chọn Timer1 chế độ 1
                MOV     P0, #0FFH               ; Chọn cổng P0 làm đầu vào nhận dữ liệu
                MOV     TL1, #018H              ; Đặt TL1 = 18 byte thấp của - 1000
                MOV     TH1, #0FCH              ; Đặt TH1 = FC byte cao của  - 1000
                MOV     IE, #88H                ; IE = 10001000 cho phép ngắt Timer1
                SETB    TR1                     ; Khởi động bộ Timer1
BACK:           MOV     A, P0                   ; Nhận dữ liệu đầu vào ở cổng P0
                MOV     P1, A                   ; Chuyển dữ liệu đến P1
                SJMP    BACK                    ; Tiếp tục nhận và chuyển dữ liệu
;
; - - Trình ISR của Timer1 phải được nạp lại vì ở chế độ 1
ISR-T1:         CLR     TR1                     ; Dừng bộ Timer1
                CLR     P2.1                    ; P2.1 = 0 bắt đầu xung mức thấp
                MOV     R2, #4                  ; 2 chu kỳ máy MC (Machine Cycle)
HERE:           DJNZ    R2, HERE                ; 4 × 2 MC = 8 MC
                MOV     TL1, #18H               ; Nạp lại byte thấp giá trị    2 MC
                MOV     TH1, #0FCH              ; Nạp lại byte cao giá trị     2 MC
                SETB    TR1                     ; Khởi động Timer1             1 MC
                SETB    P2.1                    ; P2.1 = 1 bật P2.1 trở lại cao
                RETI                            ; Trở về chương trình chính
                END
```

Lưu ý rằng phần xung mức thấp được tạo ra bởi 14 chu kỳ mức MC và mỗi MC = 1.085μs và 14 × 1.085μs = 15.19μs.

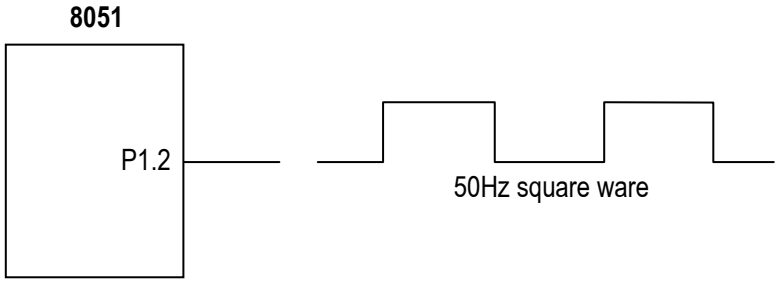
Ví dụ 11.4:

Viết một chương trình để tạo ra một sóng vuông tần số 50Hz trên chân P1.2. Ví dụ này tương tự ví dụ 9.12 ngoại trừ ngắt Timer0, giả sử XTAL = 11.0592MHz.

Lời giải:

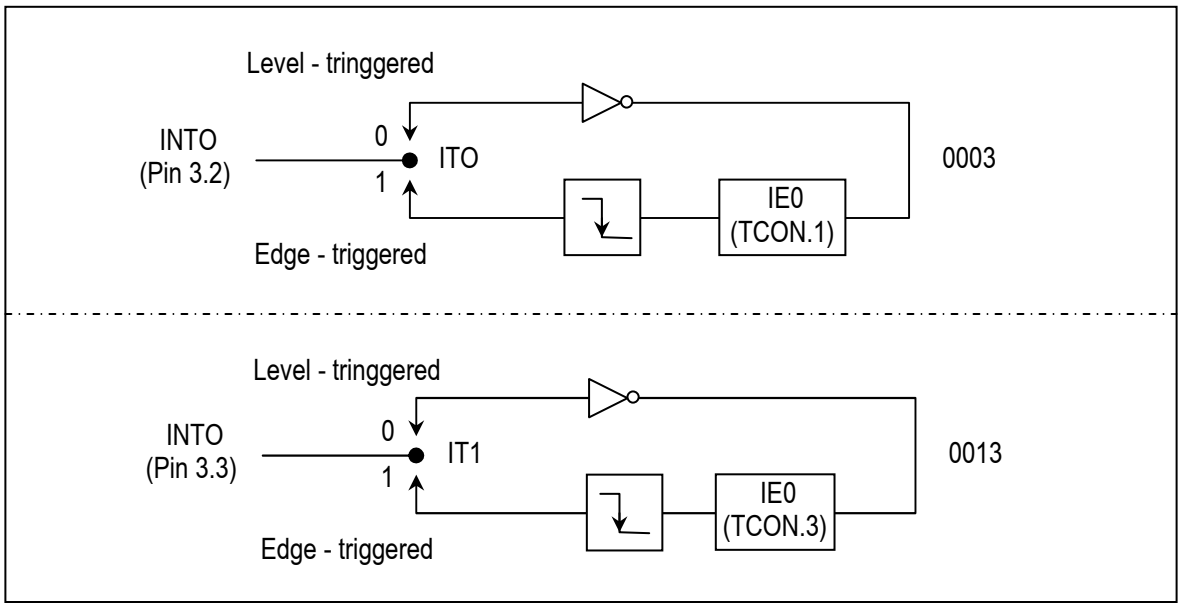
```
                ORG     0
                LJMP    MAIN
                ORG     000BH                   ; Chương trình con phục vụ ngắt cho Timer0
                CPL     P1.2
                MOV     TL0, # 00
                MOV     TH0, # 0DCH
                RETI
                ORG     30H
; ------------ main  program for initialization
MAIN:   MOV     TMOD, # 00000001B               ; Chọn Timer0 chế độ 1
```

```
        MOV   TL0, # 0DCH
        MOV   IE, # 82H          ; Cho phép ngắt Timer0
        SETB  TR0
HERE:   SJMP  HERE
        END
```



11.3 Lập trình các ngắt phần cứng bên ngoài.

Bộ vi điều khiển 8051 có hai ngắt phần cứng bên ngoài là chân 12 (P3.2) và chân 13 (P3.3) dùng cho ngắt INT0 và INT1. Khi kích hoạt những chân này thì 8051 bị ngắt tại bất kỳ công việc nào mà nó đang thực hiện và nó nhảy đến bảng véc tơ ngắt để thực hiện trình phục vụ ngắt.



11.3.1 Các ngắt ngoài INT0 và INT1.

Chỉ có hai ngắt phần cứng ngoài trong 8051 là INT0 và INT1. Chúng được bố trí trên chân P3.2 và P3.3 và địa chỉ của chúng trong bảng véc tơ ngắt là 0003H và 0013H. Như đã nói ở mục 11.1 thì chúng được ghép và bị cấm bằng việc sử dụng thanh ghi IE. Vậy chúng được kích hoạt như thế nào? Có hai mức kích hoạt cho các ngắt phần cứng ngoài: Ngắt theo mức và ngắt theo sườn. Dưới đây là mô tả hoạt động của mỗi loại.

11.3.2 Ngắt theo mức.

Ở chế độ ngắt theo mức thì các chân INT0 và INT1 bình thường ở mức cao (giống như tất cả các chân của cổng I/O) và nếu một tín hiệu ở mức thấp được cấp tới chúng thì nó ghi nhãn ngắt. Sau đó bộ vi điều khiển dừng tất cả mọi công việc nó

đang thực hiện và nhảy đến bảng véc tơ ngắt để phục vụ ngắt. Điều này được gọi là ngắt được kích hoạt theo mức hay ngắt theo mức và là chế độ ngắt mặc định khi cấp nguồn lại cho 8051. Tín hiệu mức thấp tại chân INT phải được lâýu đi trước khi thực hiện lệnh cuối cùng của trình phục vụ ngắt RETI, nếu không một ngắt khác sẽ lại được tạo ra. Hay nói cách khác, nếu tín hiệu ngắt mức thấp không được lấy đi khi ISR kết thúc thì nó không thể hiện như một ngắt khác và 8051 nhảy đến bảng véc tơ ngắt để thực hiện ISR. Xem ví dụ 11.5.

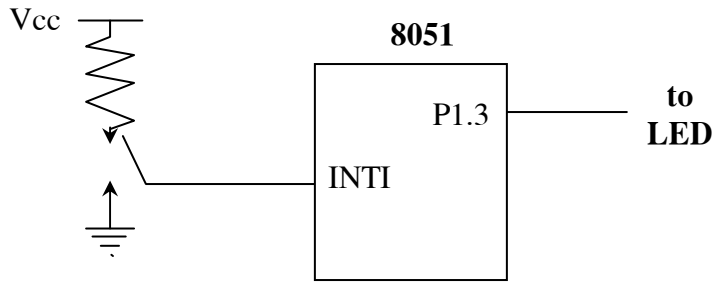
Ví dụ 11.5.

Giả sử chân INT1 được nối đến công tắc bình thường ở mức cao. Mỗi khi nó xuống thấp phải bật một đèn LED. Đèn LED được nối đến chân P1.3 và bình thường ở chế độ tắt. Khi nó được bật lên nó phải sáng vài phần trăm giây. Chừng nào công tắc được ấn xuống thấp đèn LED phải sáng liên tục.

Lời giải:

```
                ORG     0000H
                LJMP    MAIN                    ; Nhảy đến bảng véc tơ ngắt
; - - Chương trình con ISR cho ngắt cứng INT1 để bật đèn LED.
                ORG     0013H                   ; Trình phục vụ ngắt ISR cho INT1
                SETB    P1.3                    ; Bật đèn LED
                MOV     R3, # 255               ;
BACK:           DJNZ    R3, BACK                ; Giữ đèn LED sáng một lúc
                CLR     P1.3                    ; Tắt đèn LED
                RETI                            ; Trở về từ ISR
; - - Bắt đầu chương trình chính Main.
                ORG     30H
MAIN:           MOV     IE, #10000100B          ; Cho phép ngắt dài
                SJMP    HERE                    ; Chờ ở đây cho đến khi được ngắt
        END
```

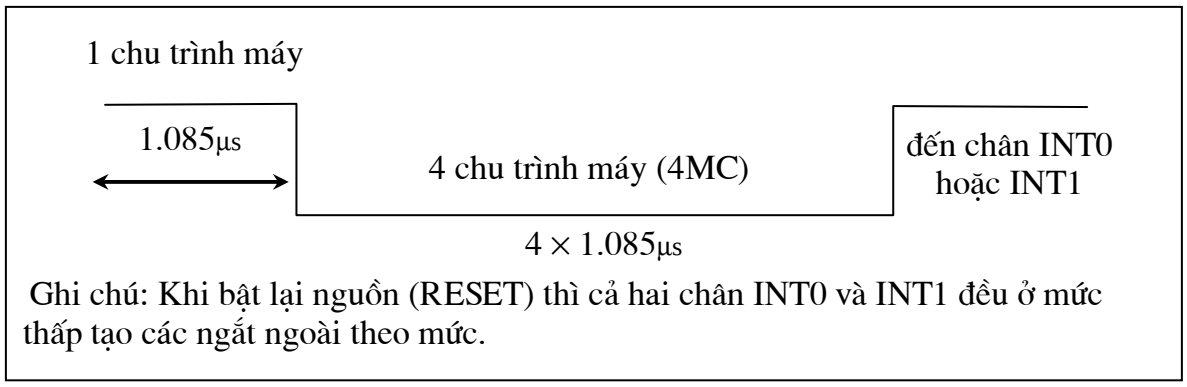
Ấn công tắc xuống sẽ làm cho đèn LED sáng. Nếu nó được giữ ở trạng thái được kích hoạt thì đèn LED sáng liên tục.



Trong chương trình này bộ vi điều khiển quay vòng liên tục trong vòng lặp HERE. Mỗi khi công tắc trên chân P3.3 (INT1) được kích hoạt thì bộ vi điều khiển thoát khỏi vòng lặp và nhảy đến bảng véc tơ ngắt tại địa chỉ 0013H. Trình ISR cho INT1 bật đèn LED lên giữ nó một lúc và tắt nó trước khi trở về. Nếu trong lúc nó thực hiện lệnh quay trở về RET1 mà chân INT1 vẫn còn ở mức thấp thì bộ vi điều khiển khởi tạo lại ngắt. Do vậy, để giải quyết vấn đề này thì chân INT1 phải được đưa lên cao tại thời điểm lệnh RET1 được thực hiện.

11.3.3 Trích mẫu ngắt theo mức.

Các chân P3.2 và P3.3 bình thường được dùng cho vào - ra nếu các bít INT0 và INT1 trong thanh ghi IE không được kích hoạt. Sau khi các ngắt phần cứng trong thanh gi IE được kích hoạt thì bộ vi điều khiển duy trì trích mẫu trên chân INTn đối với tín hiệu mức thấp một lần trong một chu trình máy. Theo bảng dữ liệu của nhà sản xuất của bộ vi điều khiển thì "chân ngắt phải được giữ ở mức thấp cho đến khi bắt đầu thực hiện trình phục vụ ngắt ISR. Nếu chân INTn được đưa trở lại mức cao trước khi bắt đầu thực hiện ISR thì sẽ chẳng có ngắt nào xảy ra". Tuy nhiên trong quá trình kích hoạt ngắt theo mức thấp nên nó lại phải đưa lên mức cao trước khi thực hiện lệnh RET1 và lại theo bảng dữ liệu của nhà sản xuất thì "nếu chân INTn vẫn ở mức thấp sau lệnh RETI của trình phục vụ ngắt thì một ngắt khác lại sẽ được kích hoạt sau khi lệnh RET1 được thực hiện". Do vậy, để bảo đảm việc kích hoạt ngắt phần cứng tại các chân INTn phải khẳng định rằng thời gian tồn tại tín hiệu mức thấp là khoảng 4 chu trình máy và không được hơn. Điều này là do một thực tế là ngắt theo mức không được chốt. Do vậy chân ngắt phải được giưa ở mức thấp cho đến khi bắt đầu thực hiện ISR.



Ghi chú: Khi bật lại nguồn (RESET) thì cả hai chân INT0 và INT1 đều ở mức thấp tạo các ngắt ngoài theo mức.

Hình 11.5: Thời gian tối thiểu của ngắt theo mức thấp (XTAL = 11.0592MHz)

11.3.4 Các ngắt theo sườn.

Như đã nói ở trước đây trong quá trình bật lại nguồn thì 8051 làm các chân INT0 và INT1 là các ngắt theo mức thấp. Để biến các chân này trở thành các ngắt theo sườn thì chúng ta phải viết chương trình cho các bít của thanh ghi TCON. Thanh thi TCON giữ các bít cờ IT0 và IT1 xác định chế độ ngắt theo sườn hay ngắt theo mức của các ngắt phần cứng IT0 và IT1 là các bít D0 và D2 của thanh ghi TCON tương ứng. Chúng có thể được biểu diễn như TCON.0 và TCON.2 vì thanh ghi TCON có thể đánh địa chỉ theo bít. Khi bật lại nguồn thì TCON.0 (IT0) và TCON.2 (IT1) đều ở mức thấp (0) nghĩa là các ngắt phần cứng ngoài của các chân INT0 và INT1 là ngắt theo mức thấp. Bằng việc chuyển các bít TCON.0 và TCON.2 lên cao qua các lệnh "SETB TCON.0" và "SETB TCON.2" thì các ngắt phần cứng ngoài INT0 và INT1 trở thành các ngắt theo sườn. Ví dụ, lệnh "SETB TCON.2" làm cho INT1 mà được gọi là ngắt theo sườn trong đó khi một tín hiệu chuyển từ cao xuống thấp được cấp đến chân P3.3 thì ở trường hợp này bộ vi điều khiển sẽ bị ngắt và bị cưỡng bức nhảy đến bảng véc tơ ngắt tại địa chỉ 0013H để thực hiện trình phục vụ ngắt. Tuy nhiên là với giải thiết rằng bít ngắt đã được cho phép trong thanh ghi IE.

D7							D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Hình 11.6: Thanh ghi TCON.

- Bít TF1 hay TCON.7 là cờ tràn của bộ Timer1. Nó được lập bởi phần cứng khi bộ đếm/ bộ định thời 1 tràn, nó được xoá bởi phần cứng khi bộ xử lý chỉ đến trình phục vụ ngắt.
- Bít TR1 hay TCON.6 là bít điều khiển hoạt động của Timer1. Nó được thiết lập và xoá bởi phần mềm để bật/ tắt Timer1.
- Bít TF0 hay TCON.5 tương tự như TF1 dành cho Timer0.
- Bít TR0 hay TCON.4 tương tự như TR1 dành cho Timer0.
- Bít IE1 hay TCON.3 cờ ngắt ngoài 1 theo sườn. Nó được thiết lập bởi CPU khi sườn ngắt ngoài (chuyển từ cao xuống thấp) được phát hiện. Nó được xóa bởi CPU khi ngắt được xử lý. Lưu ý: Cờ này không chốt những ngắt theo mức thấp.
- Bít IT1 hay TCON.2 là bít điều khiển kiểu ngắt. Nó được thiết lập và xoá bởi phần mềm để xác định kiểu ngắt ngoài theo sườn xuống hay mức thấp.
- Bít IE0 hay TCON.1 tương tự như IE1 dành cho ngắt ngoài 0.
- Bít IT0 hay TCON.0 tương tự như bít IT1 dành cho ngắt ngoài 0.

Xét ví dụ 11.6, chú ý rằng sự khác nhau duy nhất giữa vì dụ này và ví dụ 11.5 là ở trong hàng đầu tiên của MAIN khi lệnh "SETB TCON.2" chuyển ngắt INT1 về kiểu ngắt theo sườn. Khi sườn xuống của tín hiệu được cấp đến chân INT1 thì đèn LED sẽ bật lên một lúc. Đèn LED có thời gian sáng phụ thuộc vào độ trễ bên trong ISR của INT1. Để bật lại đèn LED thì phải có một sườn xung xuống khác được cấp đến chân P3.3. Điều này ngược với ví dụ 11.5. Trong ví dụ 11.5 do bản chất ngắt theo mức của ngắt thì đèn LED còn sáng chừng nào tín hiệu ở chân INT1 vẫn còn ở mức thấp. Nhưng trong ví dụ này để bật lại đèn LED thì xung ở chân INT1 phải được đưa lên cao rồi sau đó bị hạ xuống thấp để tạo ra một sườn xuống làm kích hoạt ngắt.

Ví dụ 11.6:

Giả thiết chân P3.3 (INT1) được nối với một máy tạo xung, hãy viết một chương trình trong đó sườn xuống của xung sẽ gửi một tín hiệu cao đến chân P1.3 đang được nối tới đèn LED (hoặc một còi báo). Hay nói cách khác, đèn LED được bật và tắt cùng tần số với các xung được cấp tới chân INT1. Đây là phiên bản ngắt theo sườn xung của ví dụ 11.5 đã trình bày ở trên.

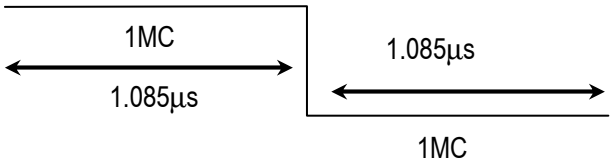
Lời giải:

```
                ORG     0000H
                LJMP    MAIN
; - - Trình  phục vụ ngắt ISR dành cho ngắt INT1 để bật đèn LED
                ORG     0013H           ; Nhảy đến địa chỉ của trình phục vụ ngắt INT1
                SETB    P1.3            ; Bật đèn LED (hoặc còi)
                MOV     R3, #225
BACK:           DJNZ    R3, HERE        ; giữ đèn LED (hoặc còi) một lúc
                CLR     P1.3            ; Tắt đèn LED (hoặc còi)
                RETI                    ; Quay trở về từ ngắt
; - - Bắt đầu chương trình chính
                ORG     30H
                SETB    TCON.2          ; Chuyển ngắt INT1 về kiểu ngắt theo sườn xung
```

```
        MOV   IE, #10001B     ; Cho phép ngắt ngoài INT1
HERE:   SJMP  HERE            ; Dừng ở đây cho đến khi bị ngắt
        END
```

11.3.5 Trình mẫu ngắt theo sườn.

Trước khi kết thúc phần này ta cần trả lời câu hỏi vậy thì ngắt theo sườn được trích mẫu thường xuyên như thế nào? Trong các ngắt theo sườn, nguồn ngoài phải giữ ở mức cao tối thiểu là một chu trình máy nữa đê đảm bảo bộ vi điều khiển nhìn thấy được sự chuyển dịch từ cao xuống thấp của sườn xung.



Thời hạn xung tối thiểu để phát hiện ra các ngắt theo sườn xung với tần số XTAL = 11.0592MHz

Sườn xuống của xung được chốt bởi 8051 và được giữa bởi thanh ghi TCON. Các bít TCON.1 và TCON.3 giữ các sườn được chốt của chân INT0 và INT1 tương ứng. TCON.1 và TCON.3 cũng còn được gọi là các bít IE0 và IE1 như chỉ ra trên hình 11.6. Chúng hoạt động như các cờ "ngắt đang được phục vụ" (Interrupt-in-server). Khi một cờ "ngắt đang được phục vụ" bật lên thì nó báo cho thế giới thực bên ngoài rằng ngắt hiện nay đang được xử lý và trên chân INTn này sẽ không có ngắt nào được đáp ứng chừng nào ngắt này chưa được phục vụ xong. Đây giống như tín hiệu báo bận ở máy điện thoại. Cần phải nhấn mạnh hạt điểm dưới đây khi quan tâm đến các bít IT0 và IT1 của thanh ghi TCON.

1. Khi các trình phục vụ ngắt ISR kết thúc (nghĩa là trong thanh ghi thực hiện lệnh RETI). Các bít này (TCON.1 và TCON.3) được xoá để báo rằng ngắt được hoàn tất và 8051 sẵn sàng đáp ứng ngắt khác trên chân đó. Để ngắt khác được nhận và thì tín hiệu trên chân đó phải trở lại mức cao và sau đó nhảy xuống thấp để được phát hiện như một ngắt theo sườn.
2. Trong thời gian trình phục vụ ngắt đang được thực hiện thì chân INTn bị làm ngơ không quan tâm đến nó có bao nhiêu lần chuyển dịch từ cao xuống thấp. Trong thực tế nó là một trong các chức năng của lệnh RETI để xoá bít tương ứng trong thanh ghi TCON (bít TCON.1 và TCON.3). Nó báo cho ta rằng trình phục vụ ngắt xắp kết thúc. Vì lý do này mà các bít TCON.1 và TCON.3 được gọi là các cơ báo "ngắt đang được phục vụ" cờ này sẽ lên cao khi một sườn xuống được phát hiện trên chân INT và dừng ở mức cao trong toàn bộ quá trình thực hiện ISR. Nó chỉ bị xoá bởi lệnh RETI là lệnh cuối cùng của ISR. Do vậy, sẽ không báo giờ cần đến các lệnh xoá bít này như "CLR TCON.1" hay "CLR TCON.3" trước lệnh RETI trong trình phục vụ ngắt đối với các ngắt cứng INT0 và INT1. Điều này không đúng với trường hợp của ngắt nối tiếp.

Ví dụ 11.7:

Sự khác nhau giữa các lệnh RET và RETI là gì? Giải thích tại sao ta không thể dùng lệnh RET thay cho lệnh RETI trong trình phục vụ ngắt.

Lời giải:

Các hai lệnh RET và RETI đều thực thi các hành vi giống nhau là lấy hai byte trên đỉnh ngăn xếp vào bộ đếm chương trình và đưa 8051 trở về nơi đó đã bỏ đi. Tuy nhiên, lệnh RETI còn thực thi một nhiệm vụ khác nữa là xoá cờ "ngắt đang được phục vụ" để báo rằng ngắt đã kết thúc và 8051 có thể nhập một ngắt mới trên chân này. Nếu ta dùng lệnh RET thay cho RETI như là lệnh cuối cùng của trình phục vụ ngắt như vậy là ta đã vô tình khoá mọi ngắt mới trên chân này sau ngắt đầu tiên vì trạng thái của chân báo rằng ngắt vẫn đang được phục vụ. Đây là trường hợp mà các cờ TF0, TF1, TCON.1 và TCON.3 được xoá bởi lệnh RETI.

11.3.6 Vài điều bổ xung về thanh ghi TCON.

Bây giờ ta xét kỹ về các bít của thanh ghi TCON để hiểu vai trò của nó trong việc duy trì các ngắt.

11.3.6.1 Các bít IT0 và IT1.

Các bít TCON.0 và TCON.2 được coi như là các bít IT0 và IT1 tương ứng. Đây là các bít xác định kiểu ngắt theo sườn xung hay theo mức xung của các ngắt phần cứng trên chân INT.0 và INT.1 tương ứng. Khi bật lại nguồn cả hai bít này đều có mức 0 để biến chúng thành ngắt theo tín hiệu mức thấp. Lập trình viên có thể điều khiển một trong số chúng lên cao để chuyển ngắt phần cứng bên ngoài thành ngắt theo ngưỡng. Trong một hệ thống dựa trên 8051 đã cho thì một khi ta đã đặt về 0 hoặc 1 thì các bít này sẽ không thay đổi vì người thiết kế đã cố định kiểu ngắt là ngắt theo sườn hay theo mức rồi.

11.3.6.2 Các bít IE0 và IE1.

Các bít TCON.1 và TCON.3 còn được gọi là IE0 và IE1 tương ứng. Các bít này được 8051 dùng để bám kiểu ngắt theo sườn xung. Nói các khác là nếu IT0 và IT1 bằng 0 thì có nghĩa là các ngắt phần cứng là ngắt theo mức thấp, các bít IE0 và IE1 không dùng đến làm gì. Các bít IE0 và IE1 được 8051 chỉ dùng để chốt sườn xung từ cao xuống thấp trên các chân INT0 và INT1. Khi có chuyển dịch sườn xung trên chân INT0 (hay INT1) thì 8051 đánh dấu (bật lên cao) các bít IEx trên thanh ghi TCON nhảy đến bảng véc tơ ngắt và bắt đầu thực hiện trình phục vụ ngắt ISR. Trong khi 8051 thực hiện ISR thì không có một sườn xung nào được ghi nhận trên chân INT0 (hay INT1) để ngăn mọi ngắt trong ngắt. Chỉ trong khi thực hiện lệnh RETI ở cuối trình phục vụ ngắt ISR thì các bít IEx mới bị báo rằng một sườn xung cao xuống thấp mới trên chân INT0 (hay INT1) sẽ kích hoạt ngắt trở lại. Từ phần trình bày trên ta thấy rằng các bít IE0 và IE1 được 8051 sử dụng bên trong để báo có một ngắt đang được xử lý hay không. Hay nói cách khác là lập trình viên không phải quan tâm đến cá bít này.

11.3.6.3 Các bít TR0 và TR1.

Đây là những bít D4 và D6 (hay TCON.4 và TCON.6) của thanh ghi TCON. Các bít này đã được giới thiệu ở chương 9 chúng được dùng để khởi động và dừng các bộ định thời Timer0 và Timer1 tương ứng. Vì thanh ghi TCON có thể đánh địa chỉ theo bít nên có thể sử dụng các lệnh "SETB TRx" và "CLR TRx" cũng như các lệnh "SETB TCON.4" và "CLR TCON.4".

11.3.6.4 Các bít TF0 và TF1.

Các bít này là D5 (TCON.5) và D7 (TCON.7) của thanh ghi TCON mà đã được giới thiệu ở chương 9. Chúng ta được sử dụng bởi các bộ Timer0 và Timer1 tương ứng để báo rằng các bộ định thời bị tràn hay quay về không. Mặc dù ta đã dùng các lệnh "JNB TFx, đích" và "CLR TFx" nhưng chúng ta cũng không thể sử

dụng các lệnh như “SETB TCON.5, đích” và “CLR TCON.5” vì TCON là thanh ghi có thể đánh địa chỉ theo bít.

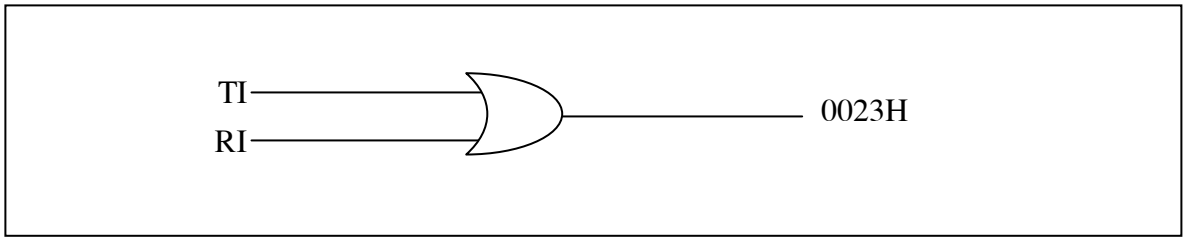
11.4 Lập trình ngắt truyền thông nối tiếp.

Trong chương 10 chúng ta đã nghiên cứu về truyền thông nối tiếp của 8051. Tất cả các ví dụ trong chương ấy đều sử dụng phương pháp thăm dò (polling). ậ chương này ta khám phá truyền thông dựa trên ngắt mà nó cho phép 8051 làm việc rất nhiều việc ngoài việc truyền và nhận dữ liệu từ cổng truyền thông nối tiếp.

11.4.1 Các cờ RI và TI và các ngắt.

Như đã nói ở chương 10 thì cờ ngắt truyền TI (Transfer interrupt) được bật lên khi bít cuối cùng của khung dữ liệu, bít stop được truyền đi báo rằng thanh ghi SBUF sẵn sàng truyền byte kế tiếp. Trong trường hợp cờ RI (Receive Interrupt) thì nó được bật lên khi toàn bộ khung dữ liệu kể cả bít stop đã đươc nhận. Hay nói cách khác khi thanh ghi SBUF đã có một byte thì cờ RI bật lên báo rằng byte dữ liệu nhận được cần lấy đi cất vào nơi an toàn trước khi nó bị mất (bị ghi đè) bởi dữ liệu mới nhận được. Chừng nào còn nói về truyền thông nối tiếp thì tất cả mọi khái niệm trên đây đều áp dụng giống như nhau cho dù sử dụng phương pháp thăm dò hay sử dụng phương pháp ngắt. Sự khác nhau duy nhất giữa hai phương pháp này là ở cách phục vụ quá trình truyền thông nối tiếp như thế nào. Trong phương pháp thăm dò thì chúng ta phải đợi cho cờ (TI hay RI) bật lên và trong lúc chờ đợi thì ta không thể làm gì được cả. Còn trong phương pháp ngắt thì ta được báo khi 8051 đã nhận được một byte hoặc nó sẵn sàng chuyển (truyền) byte kế tiếp và ta có thể làm các công việc khác trong khi truyền thông nối tiếp đang được phục vụ.

Trong 8051 chỉ có một ngắt dành riêng cho truyền thông nối tiếp. Ngắt này được dùng cho cả truyền và nhận dữ liệu. Nếu bít ngắt trong thanh gi IE (là bít IE.4) được phép khi RI và TI bật lên thì 8051 nhận được ngắt và nhảy đến địa chỉ trình phục vụ ngắt dành cho truyền thông nối tiếp 0023H trong bảng véc tơ ngắt để thực hiện nó. Trong trình ISR này chúng ta phải kiểm tra các cờ TI và RI để xem cờ nào gây ra ngắt để đáp ứng một cách phù hợp (xem ví dụ 11.8).



Hình 11.7: Ngắt truyền thông có thể do hai cờ TI và RI gọi.

11.4.2 Sử dụng cổng COM nối tiếp trong 8051.

Trong phần lớn các ứng dụng, ngắt nối tiếp chủ yếu được sử dụng để nhận dữ liệu và không bao giờ được sử dụng để truyền dữ liệu nối tiếp. Điều này giống như việc báo chuông để nhận điện thoại, còn nếu ta muốn gọi điện thoại thì có nhiều cách khác ngắt ta chứ không cần đến đổ chuông. Tuy nhiên, trong khi nhận điện thoại ta phải trả lời ngay không biết ta đang làm gì nếu không thuộc gọi sẽ (mất) đi qua. Tương tự như vậy, ta sử dụng các ngắt nối tiếp khi nhận dữ liệu đi đến để sao chép cho nó không bị mất: Hãy xét ví dụ 11.9 dưới đây.

Ví dụ 11.8:

Hãy viết chương trình trong đó 8051 đọc dữ liệu từ cổng P1 và ghi nó tới cổng P2 liên tục trong khi đưa một bản sao dữ liệu tới cổng COM nối tiếp để thực hiện truyền nối tiếp giả thiết tần số XTAL là 11.0592MHz và tốc độ baud là 9600.

Lời giải:

```
                ORG     0
                LJMP    MAIN
                ORG     23H
                LJMP    SERIAL                  ; Nhảy đến trình phục vụ ngắt truyền thông nối tiếp
MAIN:           MOVQP1, # 0FFH                  ; Lấy cổng P1 làm cổng đầu vào
                MOV     TMOD, # 20h             ; Chọn Timer1, chế độ 2 tự nạp lại
                MOV     TH1, # 0FDH             ; Chọn tốc độ baud = 9600
                MOV     SCON, # 50H             ; Khung dữ liệu: 8 bít dữ liệu, 1 stop à cho phép REN
                MOV     IE, # 10010000B         ; Cho phép ngắt nối tiếp
                SETB    TR1                     ; Khởi động Timer1
BACK:           MOV     A, P1                   ; Đọc dữ liệu từ cổng P1
                MOV     SBUF, A                 ; Lấy một bản sao tới SBUF
                MOV     P2, A                   ; Gửi nó đến cổng P2
                SJMP    BACK                    ; ở lại trong vòng lặp
;
; -------------------------Trình phục vụ ngắt cổng nối tiếp
                ORG     100H
SERIAL:         JB      TI,TRANS                ; Nhảy đến cờ TI cao
                MOV     A, SBUF                 ; Nếu không tiếp tục nhận dữ liệu
                CLR     RI                      ; Xoá cờ RI vì CPU không làm điều này
                RETI                            ; Trở về từ trình phục vụ ngắt
TRANS:          CLR     TI                      ; Xoá cờ TI vì CPU không làm điều này
                RETI                            ; Trở về từ ISR
                END
```

Trong vấn đề trên thấy chú ý đến vai trò của cờ TI và RI. Thời điểm một byte được ghi vào SBUF thì nó được đóng khung và truyền đi nối tiếp. Kết quả là khi bít cuối cùng (bít stop) được truyền đi thì cờ TI bật lên cao và nó gây ra ngắt nối tiếp được gọi khi bít tương ứng của nó trong thanh ghi IE được đưa lên cao. Trong trình phục vụ ngắt nối tiếp, ta phải kiểm tra cả cờ TI và cờ RI vì cả hai đều có thể gọi ngắt hay nói cách khác là chỉ có một ngắt cho cả truyền và nhận.

Ví dụ 11.9:

Hãy viết chương trình trong đó 8051 nhận dữ liệu từ cổng P1 và gửi liên tục đến cổng P2 trong khi đó dữ liệu đi vào từ cổng nối tiếp COM được gửi đến cổng P0. Giả thiết tần số XTAL là 11.0592MHz và tốc độ baud 9600.

Lời giải:

```
                ORG     0
                LJMP    MAIN
                ORG     23H
                LJMP            SERIAL          ; Lấy cổng P1 là cổng đầu vào
                ORG     03H
MAIN:           MOV     P1, # FFH
                MOV     TMOD, # 20H             ; Chọn Timer và chế độ hai tự nạp lại
                MOV     TH1, # 0FDH             ;Khung dữ liệu: 8 bít dữ liệu, 1 stop, cho phép REN
```

```
                MOV     SCON, # 50H             ; Cho phép ngắt nối tiếp
                MOV     IE, # 10010000B         ; Khởi động Timer1
                SETB    TR1                     ; Đọc dữ liệu từ cổng P1
BACK:           MOV     A, P1                   ; Gửi dữ liệu đến cổng P2
                MOV     P2, A                   ; ở lại trong vòng lặp
                SJMP    BACK
; -------------------------Trình phục vụ ngắt cổng nối tiếp.
                ORG     100H
SERIAL:         JB      TI, TRANS               ; Nhảy nếu Ti cao
                MOV     A, SBUF                 ; Nếu không tiếp tục nhận dữ liệu
                MOV     P0, A                   ; Gửi dữ lệu đầu vào đến cổng P0
                CLR     RI                      ; Xoá  vờ RI vì CPU không xoá cờ này
                RETI                            ; Trở về từ ISR
TRANS:          CLS     TI                      ; Xoá cờ TI và CUP không xoá cờ này.
                RETI                            ; ; trở về từ ISR
                END
```

11.4.3 Xoá cờ RI và TI trước lệnh RETI.

Để ý rằng lệnh cuối cùng trước khi trở về từ ISR là RETI là lệnh xoá các cờ RI và TI. Đây là điều cần thiết bởi vì đó là ngắt duy nhất dành cho nhận và truyền 8051 không biết được nguồn gây ra ngắt là nguồn nào, do vậy trình phục vụ ngắt phải được xoá các cờ này để cho phép các ngắt sau đó được đáp ứng sau khi kết thúc ngắt. Điều này tương phản với ngắt ngoài và ngắt bộ định thời đều được 8051 xoá các cờ. Các lệnh xoá các cờ ngắt bằng phần mềm qua các lệnh "CLR TI" và "CLR RI". Hãy xét ví dụ 11.10 dưới đây và để ý đến các lệnh xoá cờ ngắt trước lệnh RETI.

Ví dụ 11.10:

Hãy viết một chương trình sử dụng các ngắt để thực hiện các công việc sau:

a) Nhận dữ liệu nối tiếp và gửi nó đến cổng P0.
b) Lấy cổng P1 đọc và truyền nối tiếp và sao đến cổng P2.
c) Sử dụng Timer0 tạo sóng vuông tần số 5kHz trên chân P0.1 giả thiết tần số XTAL = 11.0592MHz và tốc độ baud 4800.

Lời giải:

```
                ORG     0
                LJMP    MAIN
                ORG     000BH                   ; Trình phục vụ ngắt dành cho Timer0
                CPL     P0.1                    : Tạo xung ở chân P0.1
                RETI                            ; Trở về từ ISR
                ORG     23H                     ; Nhảy đến địa chỉ ngắt truyền nối tiếp
                LJMP    SERIAL                  ; Lấy cổng P1 làm cổng đầu vào
                ORG     30H
MAIN :          MOV     P1, # 0FFH              ; Chọn Timer0 và Timer1 chế độ 2 tự nạp lại
                MOV     TMOD, # 22H             ; Chọn Timer0 và Timer1 chế độ 2 tự nạp lại
                MOV     TH1, # 0F6H             ; Chọn tốc độ baud 4800
                MOV     SCON, # 50H             ; Khung dữ liệu: 8 bít dữ liệu, 1 stop, cho phép REN
                MOV     TH0, # - 92             ; Tạo tần số 5kHz
                MOV     IE, # 10010010B         ; Cho phép ngắt nối tiếp
                SETB    TR1                     ; Khởi động Timer1
                SETB    TR0                     ; Khởi động Timer0
BACK:           MOV     A, P1                   ; Đọc dữ liệu từ cổng P1
                MOV     SBUF, A                 ; Lấy một lần bản sao dữ liệu
                MOV     P2, A                   ; Ghi nó vào cổng P2
```

```
                SJMP    BACK                ; ở lại trong vòng lặp
; ---------------------- Trình phục vụ ngắt cổng nối tiếp.
                ORG     100H
SERIAL:         JB      TI, TRANS           ; Nhảy nếu TI vào
                MOV     A, SBUF             ; Nếu không tiếp tục nhận dữ liệu
                MOV     P0, A               ; Gửi dữ liệu nối tiếp đến P0
                CLR     RI                  ; Xoá cờ RI vì 8051 không làm điều này
                RETI                        ; Trở về từ ISR
TRANS:          CLR     TI                  ; Xoá cờ TI vì 8051 không xoá
                RETI                        ; Trở về từ ISR.
                END
```

Trước khi kết thúc phần này hãy để ý đến danh sách tất cả mọi cờ ngắt được cho trong bảng 11.2. Trong khi thanh thi TCON giữ 4 cờ ngắt còn hai cờ TI và RI ở trong thanh ghi SCON của 8051.

Bảng 11.2: Các bít cờ ngắt.

Ngắt	**Cờ**	**Bít của thanh ghi SFR**
Ngắt ngoài 0	IE0	TCON.1
Ngắt ngoài 1	IE1	TCON.3
Ngắt Timer0	TF0	TCON.5
Ngắt Timer1	TF1	TCON.7
Ngắt cổng nối tiếp	T1	SCON.1
Ngắt Timer2	TF2	T2CON.7 (TA89C52)
Ngắt Timer2	EXF2	T2CON.6 (TA89C52)

11.5 Các mức ưu tiên ngắt trong 8051.

11.5.1 Các mức ưu tiên trong quá trình bật lại nguồn.

Khi 8051 được cấp nguồn thì các mức ưu tiên ngắt được gán theo bảng 11.3. Từ bảng này ta thấy ví dụ nếu các ngắt phần cứng ngoài 0 và 1 được kích hoạt cùng một lúc thì ngắt ngoài 0 sẽ được đáp ứng trước. Chỉ sau khi ngắt INT0 đã được phục vụ xong thì INT1 mới được phục vụ vì INT1 có mức ưu tiên thấp hơn. Trong thực tế sơ đồ mức ưu tiên ngắt trong bảng không có ý nghĩa gì cả mà một quy trình thăm dò trong đó 8051 thăm dò các ngắt theo trình tự cho trong bảng 11.3 và đáp ứng chúng một cách phù hợp.

Bảng 11.3: Mức ưu tiên các ngắt trong khi cấp lại nguồn.

Mức ưu tiên cao xuống thấp	
Ngắt ngoài 0	INT0
Ngắt bộ định thời 0	TF0
Ngắt ngoài 1	INT1
Ngắt bộ định thời 1	TF1
Ngắt truyền thông nối tiếp	(RI + TI)

Ví dụ 11.1:

Hãy bình luận xem điều gì xảy ra nếu các ngắt INT0, TF0 và INT1 được kích hoạt cùng một lúc. Giả thiết rằng các mức ưu tiên được thiết lập như khi bật lại nguồn và các ngắt ngoài là ngắt theo sườn xung.

Lời giải:

Nếu ba ngắt này được kích hoạt cùng một thời điểm thì chúng được chốt và được giữ ở bên trong. Sau đó kiểm tra tất cả năm ngắt theo trình tự cho trong bảng 11.3. Nếu một ngắt bất kỳ được kích hoạt thì nó được phục vụ theo trình tự. Do vậy, khi cả ba ngắt trên đây cùng được kích hoạt một lúc thì ngắt ngoài 0 (IE0) được phục vụ trước hết sau đó đến ngắt Timer0 (TF0) và cuối cùng là ngắt ngoài 1 (IE1).

D7 D0

--	--	PT2	PS	PT1	PX1	PT0	PX0

Hình 11.8: Thanh ghi mức ưu tiên ngắt IP, bít ưu tiên = 1 là mức ưu tiên cao, bít ưu tiên = 0 là mức ưu tiên thấp.

- Bít D7 và D6 hay IP.7 và IP.6 - chưa dùng.
- Bít D5 hay IP.5 là bít ưu tiên ngắt Timer2 (dùng cho 8052)
- Bít D4 hay IP.4 là bít ưu tiên ngắt cổng nối tiếp
- Bít D3 hay IP.3 là bít ưu tiên ngắt Timer1
- Bít D2 hay IP.2 là mức ưu tiên ngắt ngoài 1
- Bít D1 hay IP.1 là mức ưu tiên ngắt Timer 0
- Bít D0 hay IP.0 là mức ưu tiên ngắt ngoài 0

Người dùng không được viết phần mềm ghi các số 1 vào các bít chưa dùng vì chúng dành cho các ứng dụng tương lai.

11.5.2 Thiết lập mức ưu tiên ngắt với thanh ghi IP.

Chúng ta có thể thay đổi trình tự trong bảng 11.3 bằng cách gán mức ưu tiên cao hơn cho bất kỳ ngắt nào. Điều này được thực hiện bằng cách lập trình một thanh ghi gọi là thanh ghi mức ưu tiên ngắt IP (Interrupt Priority). Trên hình 11.8 là các bít của thanh ghi này, khi bật lại nguồn thanh thi 1P chứa hoàn toàn các số 0 để tạo ra trình tự ưu tiên ngắt theo bảng 11.3. Để một ngắt nào đó mức ưu tiên cao hơn ta thực hiện đưa bít tương ứng lên cao. Hãy xem ví dụ 11.12.

Một điểm khác nữa cần được làm sáng tỏ là mức ưu tiên ngắt khi hai hoặc nhiều bít ngắt trong thanh ghi IP được đặt lên cao. Trong trường hợp này thì trong khi các ngắt này có mức ưu tiên cao hơn các ngắt khác chúng sẽ được phục vụ theo trình tự cho trong bảng 11.3. Xem ví dụ 11.13.

Ví dụ 11.12:

a) Hãy lập trình thanh ghi IP để gán mức ưu tiên cao nhất cho ngắt INT1 (ngắt ngoài 1) sau đó.
b) Hãy phân tích điều gì xảy ra khi INT0, INT1 và TF0 được kích hoạt cùng lúc. Giả thiết tất cả các ngắt đều là các ngắt theo sườn.

Lời giải:

a) MOV IP, #0000 0100B ; Đặt bít IP.2 = 1 để gán INT1 mức ưu tiên cao nhất. Lệnh "SETB IP.2" cũng tác động tương tự bởi vì IP là thanh ghi có thể đánh địa chỉ theo bít.
b) Lệnh trong bước a) gán mức ưu tiên cao hơn INT1 so với các ngắt khác, do vậy khi INT0, INT1 và TF0 được kích hoạt cùng lúc thì trước hết INT1 được phục vụ

trước rồi sau đó đến INT0 và cuối cùng là TF0. Điều này là do INT1 có mức ưu tiên cao hơn hai ngắt kia ở bước a). Sau khi thực hiện song ngắt INT1 thì 8051 trở về phục vụ ngắt còn lại theo trình tự ưu tiên trong bảng 11.3.

Ví dụ 11.13:

Giả thiết rằng sau khi bật lại nguồn thì mức ưu tiên ngắt được thiết lập bởi lệnh "MOV IP, #0000 1100B". Hãy bình luận về quá trình các ngắt được phục vụ như thế nào?

Lời giải:

Lệnh "MOV IP, #0000 1100B" (chữ B là giá trị thập phân) thiết lập ngắt ngoài (INT1) và ngắt bộ Timer1 (TF1) có mức ưu tiên cao hơn các ngắt khác. Tuy nhiên, vì chúng được thăm dò theo bảng 11.3 nên chúng sẽ được phục vụ theo trình tự sau:

Mức ưu tiên cao nhất: Ngắt ngoài 1 (INT1)
Ngắt bộ Timer 1 (TF1)
Ngắt ngoài 0 (INT0)
Ngắt bộ Timer0 (TF0)
Mức ưu tiên thấp nhất: Ngắt cổng truyền thông nối tiếp (RI + RT).

11.5.3 Ngắt trong ngắt.

Điều gì xảy ra nếu 8051 đang thực hiện một trình phục vụ ngắt thuộc một ngắt nào đó thì lại có một ngắt khác được kích hoạt? Trong những trường hợp như vậy thì một ngắt có mức ưu tiên cao hơn có thể ngắt một ngắt có mức ưu tiên thấp hơn. Đây gọi là ngắt trong ngắt. Trong 8051 một ngắt ưu tiên thấp có thể bị ngắt bởi một ngắt có mức ưu tiên cao hơn chứ không bị ngắt bởi một ngắt có mức ưu tiên thấp hơn. Mặc dù tất cả mọi ngắt đều được chốt và gửi bên trong nhưng không có ngắt mức thấp nào được CPU quan tâm ngay tức khắc nếu 8051 chưa kết thúc phục vụ các ngắt mức cao.

11.5.4 Thu chộp ngắt bằng phần mềm (Triggering).

Có nhiều lúc ta cần kiểm tra một trình phục vụ ngắt bằng con đường mô phỏng. Điều này có thể được thực hiện bằng các lệnh đơn giản để thiết lập các ngắt lên cao và bằng cách đó buộc 8051 nhảy đến bảng véc tơ ngắt. Ví dụ, nếu bít IE dành cho bộ Timer1 được bật lên 1 thì một lệnh như "SETB TF1" sẽ ngắt 8051 ngừng thực hiện công việc đang làm bất kỳ và buộc nó nhảy đến bảng véc tơ ngắt. Hay nói cách khác, ta không cần đợi cho Timer1 quay trở về 0 mới tạo ra ngắt. Chúng ta có thể gây ra một ngắt bằng các lệnh đưa các bít của ngắt tương ứng lên cao.

Như vậy ở chương này chúng ta đã biết ngắt là một sự kiện bên trong hoặc bên ngoài gây ra ngắt bộ vi điều khiển để báo cho nó biết rằng thiết bị cần được phục vụ. Mỗi một ngắt có một chương trình đi kèm với nó được gọi là trình phục vụ ngắt ISR. Bộ vi điều khiển 8051 có sáu ngắt, trong đó năm ngắt người dùng có thể truy cập được. Đó là hai ngắt cho các thiết bị phần cứng bên ngoài INT0 và INT1, hai ngắt cho các bộ định thời là TF0 và TF1 và ngắt lành cho truyền thông nối tiếp.

8051 có thể được lập trình cho phép hoặc cấm một ngắt bất kỳ cũng như thiết lập mức ưu tiên cho nó theo yêu cầu của thuật toán ứng dụng.

CHƯƠNG 12
Phối ghép với thế giới thực: LCD, ADC và các cảm biến

Chương này khám phá một số ứng dụng của 8051 với thế giới thực. Chúng ta giải thích làm cách nào phối ghép 8051 với các thiết bị như là LCD, ADC và các cảm biến.

12.1 Phối ghép một LCD với 8051.

Ở phần này ta sẽ mô tả các chế độ hoạt động của các LCD và sau đó mô tả cách lập trình và phối ghép một LCD tới 8051.

12.1.1 Hoạt động của LCD.

Trong những năm gần đây LCD đang ngày càng được sử dụng rộng rãi thay thế dần cho các đèn LED (các đèn LED 7 đoạn hay nhiều đoạn). Đó là vì các nguyên nhân sau:

1. Các LCD có giá thành hạ.
2. Khả năng hiển thị các số, các ký tự và đồ hoạ tốt hơn nhiều so với các đèn LED (vì các đèn LED chỉ hiển thị được các số và một số ký tự).
3. Nhờ kết hợp một bộ điều khiển làm tươi vào LCD làm giải phóng cho CPU công việc làm tươi LCD. Trong khi đèn LED phải được làm tươi bằng CPU (hoặc bằng cách nào đó) để duy trì việc hiển thị dữ liệu.
4. Dễ dàng lập trình cho các ký tự và đồ hoạ.

12.1.2 Mô tả các chân của LCD.

LCD được nói trong mục này có 14 chân, chức năng của các chân được cho trong bảng 12.1. Vị trí của các chân được mô tả trên hình 12.1 cho nhiều LCD khác nhau.

1. Chân V_{CC}, V_{SS} và V_{EE}: Các chân V_{CC}, V_{SS} và V_{EE}: Cấp dương nguồn - 5v và đất tương ứng thì V_{EE} được dùng để điều khiển độ tương phản của LCD.
2. Chân chọn thanh ghi RS (Register Select).

Có hai thanh ghi rất quan trọng bên trong LCD, chân RS được dùng để chọn các thanh ghi này như sau: Nếu RS = 0 thì thanh ghi mà lệnh được chọn để cho phép người dùng gửi một lệnh chẳng hạn như xoá màn hình, đưa con trỏ về đầu dòng v.v… Nếu RS = 1 thì thanh ghi dữ liệu được chọn cho phép người dùng gửi dữ liệu cần hiển thị trên LCD.

3. Chân đọc/ ghi (R/W).

Đầu vào đọc/ ghi cho phép người dùng ghi thông tin lên LCD khi R/W = 0 hoặc đọc thông tin từ nó khi R/W = 1.

4. Chân cho phép E (Enable).

Chân cho phép E được sử dụng bởi LCD để chốt thông tin hiện hữu trên chân dữ liệu của nó. Khi dữ liệu được cấp đến chân dữ liệu thì một xung mức cao xuống thấp phải được áp đến chân này để LCD chốt dữ liệu trên các chân dữ liêu. Xung này phải rộng tối thiểu là 450ns.

5. Chân D0 - D7.

Đây là 8 chân dữ liệu 8 bít, được dùng để gửi thông tin lên LCD hoặc đọc nội dung của các thanh ghi trong LCD.

Để hiển thị các chữ cái và các con số, chúng ta gửi các mã ASCII của các chữ cái từ A đến Z, a đến f và các con số từ 0 - 9 đến các chân này khi bật RS = 1.

Cũng có các mã lệnh mà có thể được gửi đến LCD để xoá màn hình hoặc đưa con trỏ về đầu dòng hoặc nhấp nháy con trỏ. Bảng 12.2 liệt kê các mã lênh.

Chúng ta cũng sử dụng RS = 0 để kiểm tra bít cờ bận để xem LCD có sẵn sàng nhân thông tin. Cờ bận là D7 và có thể đượcđọc khi R/W = 1 và RS = 0 như sau:

Nếu R/W = 1, RS = 0 khi D7 = 1 (cờ bận 1) thì LCD bận bởi các công việc bên trong và sẽ không nhận bất kỳ thông tin mới nào. Khi D7 = 0 thì LCD sẵn sàng nhận thông tin mới. Lưu ý chúng ta nên kiểm tra cờ bận trước khi ghi bất kỳ dữ liệu nào lên LCD.

Bảng 12.1: Mô tả các chân của LCD.

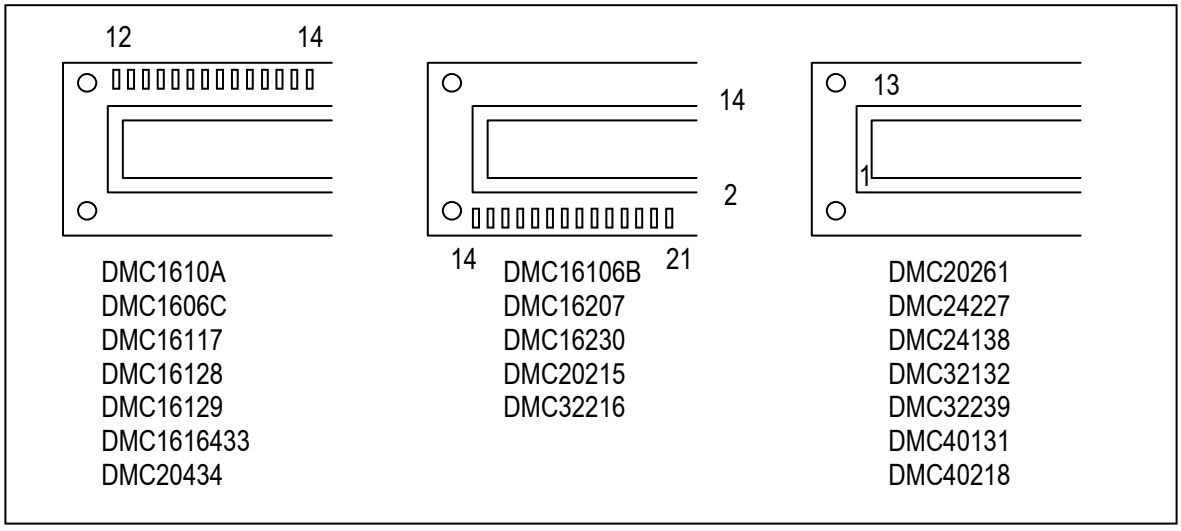
Chân	Ký hiệu	I/O	Mô tả
1	V_{SS}	-	Đất
2	V_{CC}	-	Dương nguồn 5v
3	V_{EE}	-	Cấp nguồn điều khiển phản
4	RS	I	RS = 0 chọn thanh ghi lệnh. RS = 1 chọn thanh dữ liệu
5	R/W	I	R/W = 1 đọc dữ liệu. R/W = 0 ghi
6	E	I/O	Cho phép
7	DB0	I/O	Các bít dữ liệu
8	DB1	I/O	Các bít dữ liệu
9	DB2	I/O	Các bít dữ liệu
10	DB3	I/O	Các bít dữ liệu
11	DB4	I/O	Các bít dữ liệu
12	DB5	I/O	Các bít dữ liệu
13	DB6	I/O	Các bít dữ liệu
14	DB7	I/O	Các bít dữ liệu

Bảng 12.2: Các mã lệnh LCD.

Mã (Hex)	Lệnh đến thanh ghi của LCD
1	Xoá màn hình hiển thị
2	Trở về đầu dòng
4	Giả con trỏ (dịch con trỏ sang trái)
6	Tăng con trỏ (dịch con trỏ sang phải)
5	Dịch hiển thị sang phải

7	Dịch hiển thị sang trái
8	Tắt con trỏ, tắt hiển thị
A	Tắt hiển thị, bật con trỏ
C	Bật hiển thị, tắt con trỏ
E	Bật hiển thị, nhấp nháy con trỏ
F	Tắt con trỏ, nhấp nháy con trỏ
10	Dịch vị trí con trỏ sang trái
14	Dịch vị trí con trỏ sang phải
18	Dịch toàn bộ hiển thị sang trái
1C	Dịch toàn bộ hiển thị sang phải
80	Ép con trỏ Vũ đầu dòng thứ nhất
C0	Ép con trỏ Vũ đầu dòng thứ hai
38	Hai dòng và ma trận 5×7

Ghi chú: Bảng này được mở rộng từ bảng 12.4.



Hình 12.1: Các vị trí chân của các LCD khác nhau của Optrex.

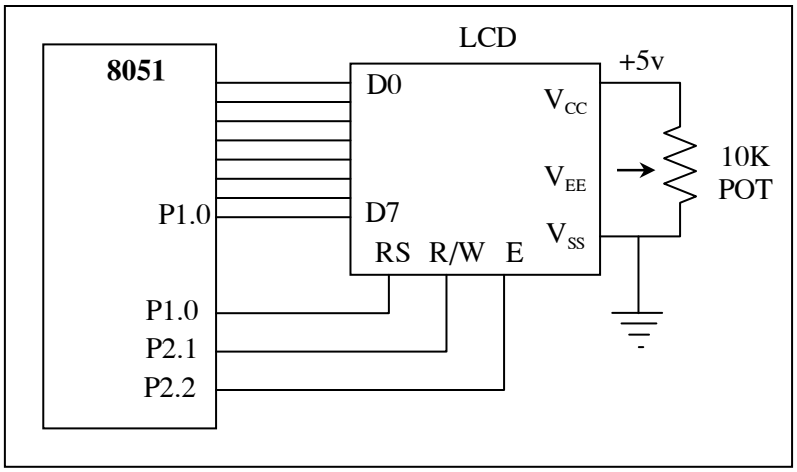
12.1.3 Gửi các lệnh và dữ liệu đến LCD với một độ trễ.

Để gửi một lệnh bất kỳ từ bảng 12.2 đến LCD ta phải đưa chân RS về 0. Đối với dữ liệu thì bật RS = 1 sau đó gửi một sườn xung cao xuống thấp đến chân E để cho phép chốt dữ liệu trong LCD. Điều này được chỉ ra trong đoạn mã chương trình dưới đây (xem hình 12.2).

```
; gọi độ thời gian trễ trước khi gửi dữ liệu/ lệnh kế tiếp.
; chân P1.0 đến P1.7 được nối tới chân dữ dữ liệu D0 - D7 của  LCD.
; Chân P2.0 được nối tới chân RS của LCD.
; Chân P2.1 được nối tới chân R/W của LCD.
; Chân P2.2 được nối đến chân E của LCD.
```

```
        ORG
        MOV     A, # 38H            ; Khởi tạo LCD hai dòng với ma trận
5 × 7
        ACALL       COMNWRT  ; Gọi chương trình con lệnh
        ACALL       DELAY           ; Cho LCD một độ trễ
        MOV     A, # 0EH    ; Hiển thị màn hình và con trỏ
        ACALL       COMNWRT  ; Gọi chương trình con lênh
        ACALL       DELAY           ; Cấp một độ trễ cho LCD
        MOV     AM # 01             ; Xoá LCD
        ACALL       COMNWRT  ; Gọi chương trình con lệnh
        ACALL       DELAY           ; Tạo độ trễ cho LCD
        MOV     A, # 06H            ; Dịch con trỏ sang phải
        ACALL       COMNWRT  ; Gọi chương trình con lệnh
        ACALL       DELAY           ; Tạo độ trễ cho LCD
        MOV     AM # 48H    ; Đưa con trỏ về dòng 1 cột 4
        ACALL       COMNWRT  ; Gọi chương trình con lệnh
        ACALL       DELAY           ; Tạo độ trễ cho LCD
        MOV     A, # “N”            ; Hiển thị chữ N
        ACALL       DATAWRT  ; Gọi chương trình con hiển thij
DISPLAY
        ACALL       DELAY           ; Tạo độ trễ cho LCD
        MOV     AM # “0”            ; Hiển thị chữ 0
        ACALL       DATAWRT  ; Gọi DISPLAY
AGAIN:      SJMP        AGAIN       ; Chờ ở đây
COMNWRT:                            ; Gửi lệnh đến LCD
        MOV     P1, A       ; Sao chép thanh ghi A đến cổng P1
        CLR     P2.0        ; Đặt RS = 0 để gửi lệnh
        CLR     P2.1        ; Đặt R/W = 0 để ghi dữ liệu
        SETB    P2.2        ; Đặt E = 1 cho xung cao
        CLR     P2.2        ; Đặt E = 0 cho xung cao xuống thấp
        RET
DATAWRT:                            ; Ghi dữ liệu ra LCD
        MOV     P1, A       ; Sao chép thanh ghi A đến cổng P1
        SETB    P2.0        ; Đặt RS = 1 để gửi dữ liệu
        CLR     P2.1        ; Đặt R/W = 0 để ghi
        SETB    P2.2        ; Đặt E = 1 cho xung cao
        CLR     P2.2        ; Đặt E = 0 cho xung cao xuống thấp
        RET
DELAY:  MOV     R3, # 50            ; Đặt độ trễ 50µs hoặc cao hơn cho
CPU nhanh
HERE2:      MOV     R4, # 255   ; Đặt R4 = 255
```

```
HERE:       DJNZ    R4, HERE    ; Đợi ở đây cho đến khi R4 = 0
            DJNZ    R3, HERE2
            RET
            END
```



Hình 12.2: Nối ghép LCD.

12.1.4 Gửi mã lệnh hoặc dữ liệu đến LCD có kiểm tra cờ bận.

Đoạn chương trình trên đây đã chỉ ra cách gửi các lệnh đến LCD mà không có kiểm tra cờ bận (Busy Flag). Lưu ý rằng chúng ta phải đặt một độ trễ lớn trong quá trình xuất dữ liệu hoặc lệnh ra LCD. Tuy nhiên, một cách tốt hơn nhiều là hiển thị cờ bận trước khi xuất một lệnh hoặc dữ liệu tới LCD. Dưới đây là một chương trình như vậy.

```
; Kiểm tra cờ bận trước khi gửi dữ liệu, lệnh ra LCD
; Đặt P1 là cổng dữ liệu
; Đặt P2.0 nối tới cổng RS
; Đặt P2.1 nối tới chân R/W
; Đặt P2.2 nối tới chân E
            ORG
            MOV     A, # 38H            ; Khởi tạo LCD hai dòng với ma trận 5 × 7
            ACALL       COMMAND     ; Xuất lệnh
            MOV     A, # 0EH        ; Dịch con trỏ sang phải
            ACALL       COMMAND     ; Xuất lệnh
            MOV     A, # 01H            ; Xoá lệnh LCD
            ACALL       COMMAND     ; Xuất lệnh
            MOV     A, # 86H            ; Dịch con trỏ sang phải
            ACALL       COMMAND     ; Đưa con trỏ về dòng 1 lệnh 6
            MOV     A, # "N"            ; Hiển thị chữ N
            ACALL       DATA DISPLAY
```

```
        MOV     A, # "0"            ; Hiển thị chữ 0
        ACALL           DATA DISPLAY
HERE:       SJMP        HERE        ; Chờ ở đây
COMMAND: ACALL          READY           ; LCD đã sẵn sàng chưa?
        MOV     P1, A       ; Xuất mã lệnh
        CLR     P2.0        ; Đặt RS = 0 cho xuất lệnh
        CLR     P2.1        ; Đặt R/W = 0 để ghi dữ liệu tới LCD
        SETB    P2.2        ; Đặt E = 1 đối với xung cao xuống thấp
        CLR     P2.2        ; Đặt E = 0 chốt dữ liệu
        RET
DATA-DISPLAY::
        ACALL           READY           ; LCD đã sẵn sàng chưa?
        MOV     P1, A       ; Xuất dữ liệu
        SETB    P2.0        ; Đặt RS = 1 cho xuất dữ liệu
        CLR     P2.1        ; Đặt R/W = 0 để ghi dữ liệu ra LCD
        SETB    P2.2        ; Đặt E = 1 đối với xung cao xuống thấp
        CLR     P2.2        ; Đặt E = 0 chốt dữ liệu
        RET
DELAY:
        SETB    P1.7        ; Lấy P1.7 làm cổng vào
        CLR     P2.0        ; Đặt RS = 0 để truy cập thanh ghi lệnh
        SETB    P2.1        ; Đặt R/W = 1 đọc thanh ghi lệnh
; Đọc thanh ghi lệnh và kiểm tra cờ lệnh
BACK:       CLR     P2.2        ; E = 1 đối với xung cao xuống thấp
        SETB    P2.2        ; E = 0 cho xung cao xuống thấp?
        JB      P1.7, BACK  ; Đợi ở đây cho đến khi cờ bận = 0
        RET
        END
```

Lưu ý rằng trong chương trình cờ bận D7 của thanh ghi lệnh. Để đọc thanh ghi lệnh ta phải đặt RS = 0, R/W = 1 và xung cao - xuống - thấp cho bít E để cấp thanh ghi lệnh cho chúng ta. Sau khi đọc thanh ghi lệnh, nếu bít D7 (cờ bận) ở mức cao thì LCD bận và không có thông tin (lệnh) nào được xuất đến nó chỉ khi nào D7 = 0 mới có thể gửi dữ liệu hoặc lệnh đến LCD. Lưu ý trong phương phát này không sử dụng độ trễ thời gian nào vì ta đang kiểm tra cờ bận trước khi xuất lệnh hoặc dữ liệu lên LCD.

12.1.5 Bảng dữ liệu của LCD.

Trong LCD ta có thể đặt dữ liệu vào bất cứ chỗ nào. dưới đây là các vị trí địa chỉ và cách chúng được truy cập.

RS	E/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A	A	A	A	A	A	A

Khi AAAAAAA = 0000000 đến 0100111 cho dòng lệnh 1 và AAAAAAA = 1100111 cho dòng lệnh2. Xem bảng 12.3.

Bảng 12.3: Đánh địa chỉ cho LCD.

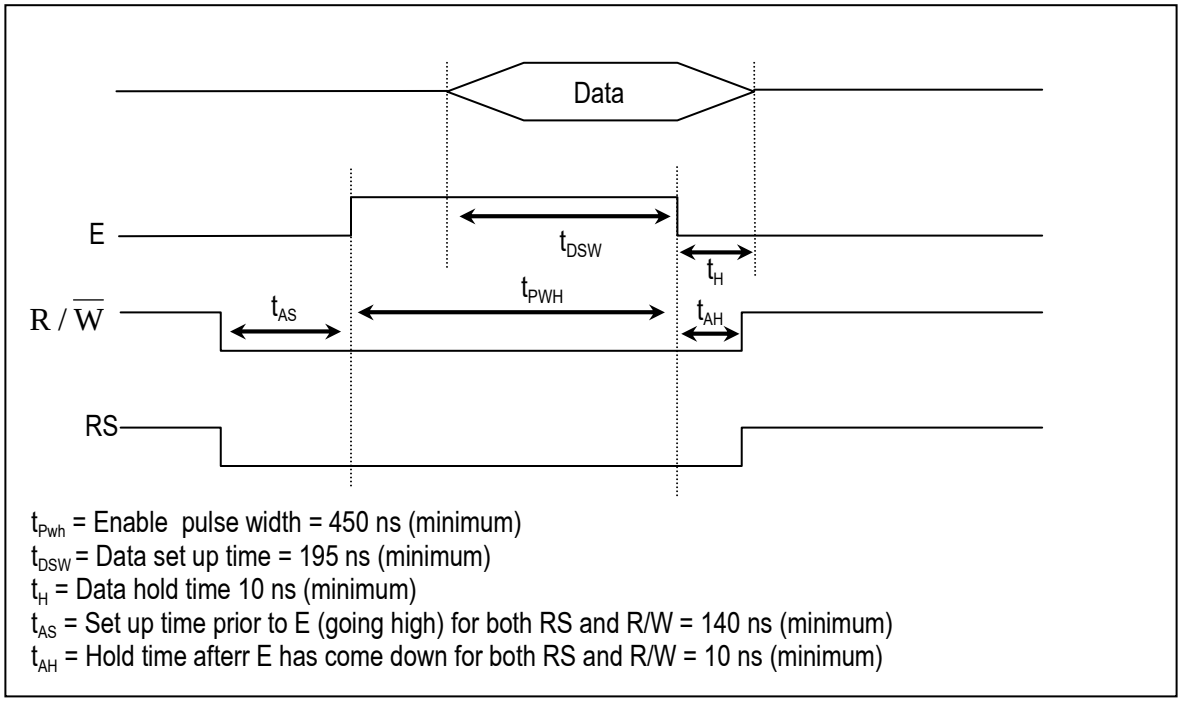
	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Dòng 1 (min)	1	0	0	0	0	0	0	0
Dòng 1 (max)	1	0	1	0	0	1	1	1
Dòng 2 (min)	1	1	0	0	0	0	0	0
Dòng 2 (max)	1	1	1	0	0	1	1	1

Dải địa chỉ cao có thể là 0100111 cho LCD. 40 ký tự trong khi đối với CLD 20 ký tự chỉ đến 010011 (19 thập phân = 10011 nhị phân). Để ý rằng dải trên 0100111 (nhị phân) = 39 thập phân ứng với vị trí 0 đến 39 cho LCD kích thước 40×2.

Từ những điều nói ở trên đây ta có thể nhận được các địa chỉ của vị trí con trỏ có các kích thước LCD khác nhau. Xem hình 12.3 chú ý rằng tất cả mọi địa chỉ đều ở dạng số Hex. Hình 12.4 cho một biểu đồ của việc phân thời gian của LCD. Bảng 12.4 là danh sách liệt kê chi tiết các lệnh và chỉ lệnh của LCD. Bảng 12.2 được mở rộng từ bảng này.

16 × 2 LCD	80 C0	81 C0	82 C2	83 C3	84 C4	85 C5	86 C6	Through Through	8F CF
20 × 1 LCD	80	81	82	83	Through	93			
20 × 2 LCD	80 C0	81 C0	82 C2	83 C3	Through Through	93 D3			
20 × 4 LCD	80 C0 94 D4	81 C0 95 D5	82 C2 96 D6	83 C3 97 D7	Through Through Through Through	93 D3 A7 E7			
20 × 2 LCD	80 C0	81 C0	82 C2	83 C3	Through Through	A7 E7			
Note: All data is in hex.									

Hình 12.3: Các địa chỉ con trỏ đối với một số LCD.



t_{Pwh} = Enable pulse width = 450 ns (minimum)
t_{DSW} = Data set up time = 195 ns (minimum)
t_H = Data hold time 10 ns (minimum)
t_{AS} = Set up time prior to E (going high) for both RS and R/W = 140 ns (minimum)
t_{AH} = Hold time afterr E has come down for both RS and R/W = 10 ns (minimum)

Hình 12.4: Phân khe thời gian của LCD.

Bảng 12.4: Danh sách liệt kê các lệnh và địa chỉ lệnh của LCD.

Lệnh	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Mô tả	Thời gian thực hiện
Xoá màn hình	0	0	0	0	0	0	0	0	0	1	Xoá toàn bộ màn hình và đặt địa chỉ 0 của DD RAM vào bộ đếm địa chỉ	1.64 µs
Trở về đầu dòng	0	0	0	0	0	0	0	0	1	-	Đặt địa chỉ 0 của DD RAM như bộ đếm địa chỉ. Trả hiển thị dịch về vị trí gốc DD RAM không thay đổi	1.64 µs
Đặt chế độ truy nhập	0	0	0	0	0	0	0	1	I/D	S	Đặt hướng chuyển dịch con trỏ và xác định dịch hiển thị các thao tác này được thực hiện khi đọc và ghi dữ liệu	40 µs

Điều khiển Bật/tắt hiển thị	0	0	0	0	0	0	1	D	C	B	Đặt Bật/ tắt màn hình (D) Bật/ tắt con trỏ (C) và nhấp nháy ký tự ở vị trí con trỏ (B)	40 µs
Dịch hiển thị và con trỏ	0	0	0	0	0	1	S/C	R/L	-	-	Dịch con trỏ và dịch hiển thị mà không thay đổi DD RAM	40 µs
Đặt chức năng	0	0	0	0	1	DL	N	F	-	-	Thiết lập độ dài dữ liệu (DL) số dòng hiển thị (L) và phòng ký tự (F)	40 µs
Đặt địa chỉ CGRAM	0	0	0	1	AGC						Thiết lập địa chỉ C6 RAM dữ liệu CG RAM được gửi đi và nhận sau thiết lập này	40 µs
Thiết lập địa chỉ DD RAM	0	0	1	ADD							Thiết lập địa chỉ DD RAM dữ liệu DD RAM được gửi và nhận sau thiết lập này	40 µs
Cờ bận đọc và địa chỉ	0	1	BF	ADD							Cờ bận đọc (BF) báo hoạt động bên trong đang được thực hiện và đọc nội dung bộ đếm địa chỉ	40 µs
Ghi dữ liệu CG hoặc DD RAM	1	0	Ghi dữ liệu								Ghi dữ liệu vào DD RAM hoặc CG RAM	40 µs

Đọc dữ liệu CG hoặc DD RAM	1	1	Đọc dữ liệu	Đọc dữ liệu từ DD RAM hoặc CG RAM	40 μs

Ghi chú:

1. Thời gian thực là thời gian cực đại khi tần số f_{CP} hoặc f_{osc} là 250KHz
2. Thời gian thực thay đổi khi tần số thay đổi. Khi tần số f_{EP} hay f_{osc} Là 270kHz thì thời gian thực hiện được tính $250/270 \times 40 = 35\mu s$ v.v…
3. Các ký hiệu viết tắt trong bảng là:
4.

DD RAM RAM dữ liệu hiển thị (Display Data RAM)
CG RAM RAM máy phát ký tự (character Generator)
ACC Địa chỉa của RAM máy phát ký tự
ADD Địa chỉ của RAM dữ liệu hiển thị phù hợp với địa chỉ con trỏ.
AC Bộ đếm địa chỉ (Address Counter) được dùng cho các địa chỉ DD RAM và CG RAM.
1/D = 1 Tăng 1/D = 0 Giảm
S = 1 Kèm dịch hiển thị
S/C = 1 Dịch hiển thị S/C = 0 Dịch con trỏ
R/L = 1 Dịch sang phải R/L = 0 Dịch trái
DL = 1 8 bít DL = 0 4 bít
N = 1 2 dòng N = 1 1 dòng
F = 1 Ma trận điểm 5×10 F = 0 Ma trận điểm 5×7
BF = 1 Bận BF = 0 Có thể nhận lênh

12.2 Phối ghép 8051 với ADC và các cảm biến.

Phần này sẽ khám phá ghép các chíp ADC (bộ chuyển đổi tương tự số) và các cảm biến nhiệt với 8051.

12.1.1 Các thiết bị ADC.

Các bộ chuyển đổi ADC thuộc trong những thiết bị được sử dụng rộng rãi nhất để thu dữ liệu. Các máy tính số sử dụng các giá trị nhị phân, nhưng trong thế giới vật lý thì mọi đại lượng ở dạng tương tự (liên tục). Nhiệt độ, áp suất (khí hoặc chất lỏng), độ ẩm và vận tốc và một số ít trọng những đại lượng vật lý của thế giới thực mà ta gặp hàng ngày. Một đại lượng vật lý được chuyển về dòng điện hoặc điện áp qua một thiết bị được gọi là các bộ biến đổi.

Các bộ biến đổi cũng có thể được coi như các bộ cảm biến. Mặc dù chỉ có các bộ cảm biến nhiệt, tốc độ, áp suất, ánh sáng và nhiều đại lượng tự nhiên khác nhưng chúng đều cho ra các tín hiệu dạng dòng điện hoặc điện áp ở dạng liên tục. Do vậy, ta cần một bộ chuyển đổi tương tự số sao cho bộ vi điều khiển có thể đọc được chúng. Một chíp ADC được sử dụng rộng rãi là ADC 804.

12.2.2 Chíp ADC 804.

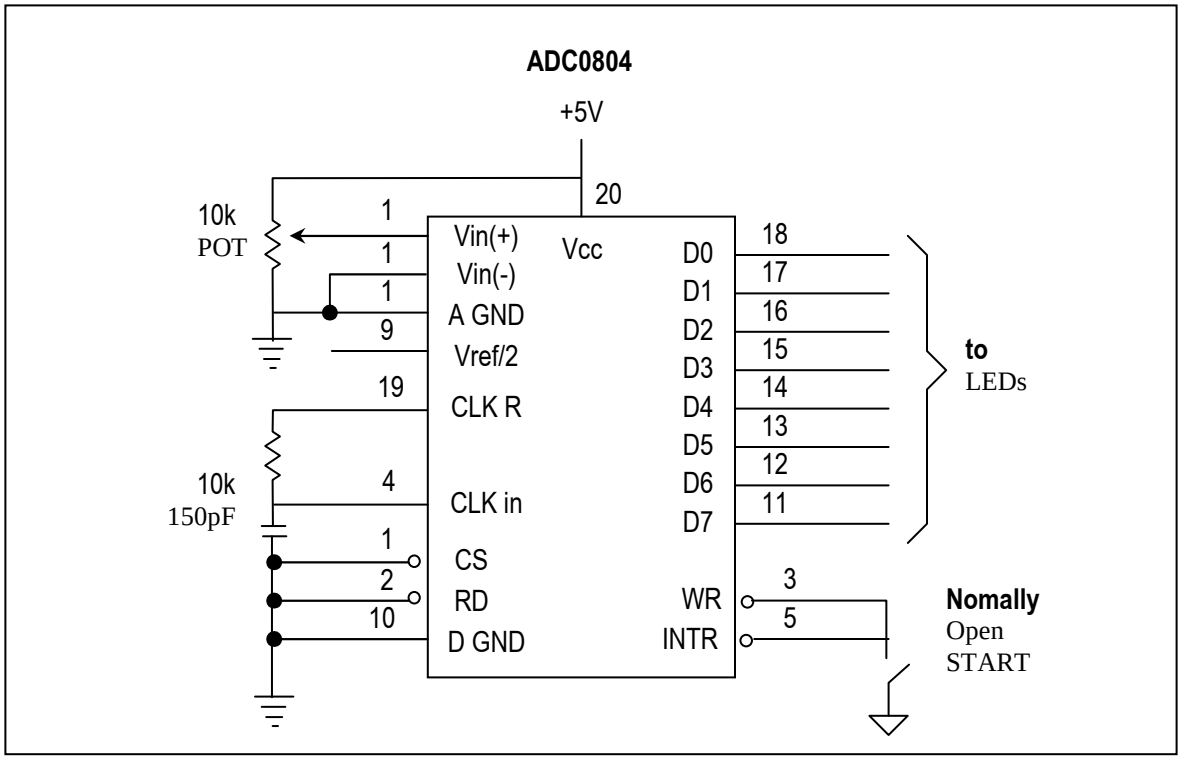
Chíp ADC 804 là bộ chuyển đổi tương tự số trong họ các loạt ADC 800 từ hãng National Semiconductor. Nó cũng được nhiều hãng khác sản xuất, nó làm việc với +5v và có độ phân giải là 8 bít. Ngoài độ phân giải thì thời gian chuyển đổi cũng là một yếu tố quan trọng khác khi đánh giá một bộ ADC. Thời gian chuyển đổi được định nghĩa như là thời gian mà bộ ADC cần để chuyển một đầu vào tương tự thành một số nhị phân. Trong ADC 804 thời gian chuyển đổi thay đổi phụ thuộc vào tần số đồng hồ được cấp tới chân CLK và CLK IN nhưng không thể nhanh hơn 110μs. Các chân của ADC 804 được mô tả như sau:

1. Chân $\overline{CS}$ - chọn chíp: Là một đầu vào tích cực mức thấp được sử dụng để kích hoạt chíp ADC 804. Để truy cập ADC 804 thì chân này phải ở mức thấp.
2. Chân $\overline{RD}$ (đọc): Đây là một tín hiệu đầu vào được tích cực mức thấp. Các bộ ADC chuyển đổi đầu vào tương tự thành số nhị phân tương đương với nó và giữ nó trong một thanh ghi trong. $\overline{RD}$ được sử dụng để nhận dữ liệu được chuyển đổi ở đầu ra của ADC 804. Khi CS = 0 nếu một xung cao - xuống - thấp được áp đến chân $\overline{RD}$ thì đầu ra số 8 bít được hiển diện ở các chân dữ liệu D0 - D7. Chân $\overline{RD}$ cũng được coi như cho phép đầu ra.
3. Chân ghi $\overline{WR}$ (thực ra tên chính xác là "Bắt đầu chuyển đổi"). Đây là chân đầu vào tích cực mức thấp được dùng để báo cho ADC 804 bắt đầu quá trình chuyển đổi. Nếu CS = 0 khi $\overline{WR}$ tạo ra xung cao - xuống - thấp thì bộ ADC 804 bắt đầu chuyển đổi giá trị đầu vào tương tự V_{in} về số nhị phân 8 bít. Lượng thời gian cần thiết để chuyển đổi thay đổi phụ thuộc vào tần số đưa đến chân CLK IN và CLK R. Khi việc chuyển đổi dữ liệu được hoàn tất thì chân INTR được ép xuống thấp bởi ADC 804.
4. Chân CLK IN và CLK R.

Chân CLK IN là một chân đầu vào được nối tới một nguồn đồng hồ ngoài khi đồng hồ ngoài được sử dụng để tạo ra thời gian. Tuy nhiên 804 cũng có một máy tạo xung đồng hồ. Để sử dụng máy tạo xung đồng hồ trong (cũng còn được gọi là máy tạo đồng hồ riêng) của 804 thì các chân CLK IN và CLK R được nối tới một tụ điện và một điện trở như chỉ ra trên hình 12.5. Trong trường hợp này tần số đồng hồ được xác định bằng biểu thức:

$$f = \frac{1}{1{,}1RC}$$

Giá trị tiêu biểu của các đại lượng trên là R = 10kΩ và C= 150pF và tần số nhận được là f = 606kHz và thời gian chuyển đổi sẽ mất là 110μs.



Hình 12.5: Kiểm tra ADC 804 ở chế độ chạy tự do.

5. Chân ngắt $\overline{\text{INTR}}$ (ngắt hay gọi chính xác hơn là "kết thúc chuyển đổi').

Đây là chân đầu ra tích cực mức thấp. Bình thường nó ở trạng thái cao và khi việc chuyển đổi hoàn tất thì nó xuống thấp để báo cho CPU biết là dữ liệu được chuyển đổi sẵn sàng để lấy đi. Sau khi $\overline{\text{INTR}}$ xuống thấp, ta đặt CS = 0 và gửi một xung cao 0 xuống - thấp tới chân $\overline{\text{RD}}$ lấy dữ liệu ra của 804.

6. Chân V_{in} (+) và V_{in} (-).

Đây là các đầu vào tương tự vi sai mà $V_{in} = V_{in}$ (+) - V_{in} (-). Thông thường V_{in} (-) được nối xuống đất và V_{in} (+) được dùng như đầu vào tương tự được chuyển đổi về dạng số.

7. Chân V_{CC}.

Đây là chân nguồn nuối +5v, nó cũng được dùng như điện áp tham chiếu khi đầu vào $V_{ref/2}$ (chân 9) để hở.

8. Chân $V_{ref/2}$.

Chân 9 là một điện áp đầu vào được dùng cho điện áp tham chiếu. Nếu chân này hở (không được nối) thì điện áp đầu vào tương tự cho ADC 804 nằm trong dải 0 đến +5v (giống như chân V_{CC}). Tuy nhiên, có nhiều ứng dụng mà đầu vào tương tự áp đến V_{in} cần phải khác ngoài dải 0 đến 5v. Chân $V_{ref/2}$ đượcdùng để thực thi các điện áp đầu vào khác ngoài dải 0 - 5v. Ví dụ, nếu dải đầu vào tương tự cần phải là 0 đến 4v thì $V_{ref/2}$ được nối với +2v.

Bảng 12.5 biểu diễn dải điện áp V_{in} đối với các đầu vào $V_{ref/2}$ khác nhau.

Bảng 12.5: Điện áp $V_{ref/2}$ liên hệ với dải V_{in}.

V_{ref}/ 2(V)	V_{in}(V)	Step Size (mV)
Hở *	0 đến 5	5/256 = 19.53
2.0	0 đến 4	4/255 = 15.62
1.5	0 đến 3	3/256 = 11.71
1.28	0 đến 2.56	2.56/256 = 10
1.0	0 đến 2	2/256 = 7.81
0.5	0 đến 1	1/256 = 3.90

Ghi chú: - V_{CC} = 5V

- * Khi $V_{ref}/2$ hở thì đo được ở đó khoảng 2,5V
- Kích thước bước (độ phân dải) là sự thay đổi nhỏ nhất mà ADC có thể phân biệt được.

9. Các chân dữ liệu D0 - D7.

Các chân dữ liệu D0 - D7 (D7 là bít cao nhất MSB và D0 là bít thấp nhất LSB) là các chân đầu ra dữ liệu số. Đây là những chân được đệm ba trạng thái và dữ liệu được chuyển đổi chỉ được truy cập khi chân CS = 0 và chân $\overline{RD}$ bị đưa xuống thấp. Để tính điện áp đầu ra ta có thể sử dụng công thức sau:

$$D_{out} = \frac{V_{in}}{\text{kich thuoc buoc}}$$

Với D_{out} là đầu ra dữ liệu số (dạng thập phân). V_{in} là điện áp đầu vào tương tự và độ phân dải là sự thay đổi nhỏ nhất được tính như là $(2 \times V_{ref}/2)$ chia cho 256 đối với ADC 8 bít.

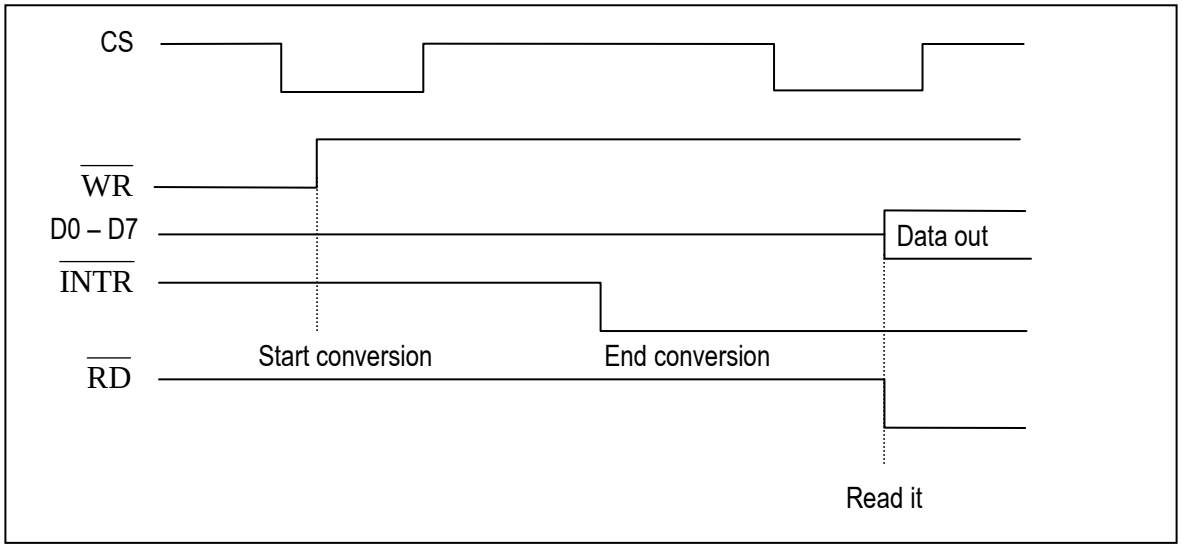
10. Chân đất tương tự và chân đất số.

Đây là những chân đầu vào cấp đất chung cho cả tín hiệu số và tương tự. Đất tương tự được nối tới đất của chân V_{in} tương tự, còn đất số được nối tới đất của chân V_{cc}. Lý do mà ta phải có hai đất là để cách ly tín hiệu tương tự V_{in} từ các điện áp ký sinh tạo ra việc chuyển mạch số được chính xác. Trong phần trình bày của chúng ta thì các chân này được nối chung với một đất. Tuy nhiên, trong thực tế thu đo dữ liệu các chân đất này được nối tách biệt.

Từ những điều trên ta kết luận rằng các bước cần phải thực hiện khi chuyển đổi dữ liệu bởi ADC 804 là:

a) Bật CS = 0 và gửi một xung thấp lên cao tới chân $\overline{WR}$ để bắt đầu chuyển đổi.
b) Duy trì hiển thị chân $\overline{INTR}$. Nếu $\overline{INTR}$ xuống thấp thì việc chuyển đổi được hoàn tất và ta có thể sang bước kế tiếp. Nếu $\overline{INTR}$ cao tiếp tục thăm dò cho đến khi nó xuống thấp.

c) Sau khi chân $\overline{INTR}$ xuống thấp, ta bật CS = 0 và gửi một xung cao - xuống - thấp đến chân $\overline{RD}$ để lấy dữ liệu ra khỏi chíp ADC 804. Phân chia thời gian cho quá trình này được trình bày trên hình 12.6.

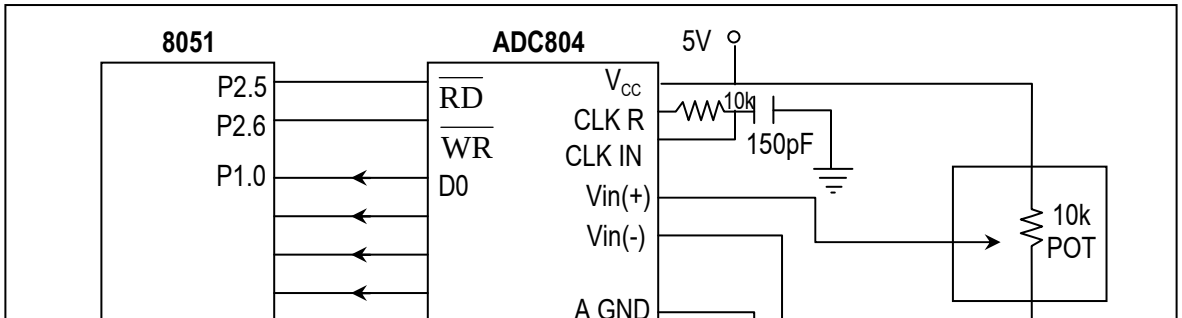


Hình 12.6: Phân chia thời gian đọc và ghi của ADC 804.

12.2.3 Kiểm tra ADC 804.

Chúng ta có thể kiểm tra ADC 804 bằng cách sử dụng sơ đồ mạch trên hình 12.7. thiết lập này được gọi là chế độ kiểm tra chạy tự do và được nhà sản xuất khuyến cao nên sử dụng. Hình 12.5 trình bày một biến trở được dùng để cấp một điện áp tương tự từ 0 đến 5V tới chân đầu vào.

$V_{in}(+)$ của ADC 804 các đầu ra nhị phân được hiển thị trên các đèn LED của bảng huấn luyện số. Cần phải lưu ý rằng trong chế độ kiểm tra chạy tự do thì đầu vào CS được nối tới đất và đầu vào $\overline{WR}$ được nối tới đầu ra $\overline{INTR}$. Tuy nhiên, theo tài liệu của hãng National Semiconductor "nút WR và INTR phải được tạm thời đưa xuống thấp kế sau chu trình cấp nguồn để bảo đảm hoạt động".



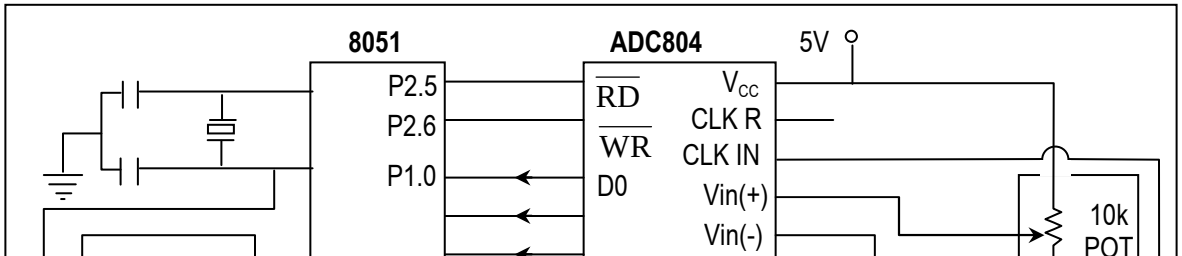
Hình 12.7: Nối ghép ADC 804 với nguồn đồng hồ riêng.

Ví dụ 12.7:

Hãy thử nối ghép ADC 804 với 8051 theo sơ đồ 12.7. Viết một chương trình để hiển thị chân INTR và lấy đầu vào tương tự vào thanh ghi A. Sau đó gọi một chương trình chuyển đổi mã Hex ra ASCII và một chương trình hiển thị dữ liệu. Thực hiện điều này liên tục.

Lời giải:

```
; Đặt P2.6 = WR (bắt đầu chuyển đổi cần 1 xung thấp lên cao)
; Đặt chân P2.7 = 0 khi kết thúc chuyển đổi
; Đặt P2.5 = RD (xung cao - xuống - thấp sẽ đọc dữ liệu từ ADC)
; P1.0 – P1.7 của ADC 804
        MOV     P1, # 0FFH      ; Chọn P1 là cổng đầu vào
BACK:   CLR     P2.6            ; Đặt WR = 0
        SETB    P2.6            ; Đặt WR = 1 để bắt đầu chuyển đổi
HERE:   JB      P2.7, HERE      ; Chờ cho P2.7 to để kết thúc chuyển đổi
        CLR     P2.5            ; Kết thúc chuyển đổi, cho phép đọc RD
        MOV     A,  P1          ; Đọc dữ liệu vào thanh ghi A
        ACALL   CONVERSION      ; Chuyển đổi số Hex ra mã ASCII
        ACALL   DATA-DISPLAY    ; Hiển thị dữ liệu
        SETB    P2.5            ; Đưa RD = 1 để cho lần đọc sau.
        SJMP    BACK
```



Hình 12.8: Nối ghép ADC 804 với đồng hồ từ XTAL2 của 8051.

Trên hình 12.8 ta có thể thấy rằng tín hiệu đồng hồ đi vào ADC 804 là từ tần số thạch anh của 8051. Vì tần số này quá cao nên ta sử dụng hai mạch lật Rlip - Flop kiểu D (74LS74) để chia tần số này cho 4. Một mạch lật chia tần số cho 2 nếu ta nối đầu $\overline{Q}$ tới đầu vào D. Đối với tần số cao hơn thì ta cần sử dụng nhiều mạch Flip - Plop hơn.

12.2.4 Phối ghép với một cảm biến nhiệt của 8051.

Các bộ biến đổi (Transducer) chuyển đổi các đại lượng vật lý ví dụ như nhiệt độ, cường độ ánh sáng, lưu tốc và tốc độ thành các tín hiệu điện phụ thuộc vào bộ biến đổi mà đầu ra có thể là tín hiệu dạng điện áp, dòng, trở kháng hay dung kháng. Ví dụ, nhiệt độ được biến đổi thành về các tín hiệu điện sử dụng một bộ biến đổi gọi là Rhermistor (bộ cảm biến nhiệt), một bộ cảm biến nhiệt đáp ứng sự thay đổi nhiệt độ bằng cách thay đổi trở kháng nhưng đáp ứng của nó không tuyến tính (xem bảng 12.6).

Bảng 12.6: Trở kháng của bộ cảm biến nhiệt theo nhiệt độ.

Nhiệt độ (^{0}C)	**Trở kháng của cảm biến (kΩ)**
0	29.490
25	10.000
50	3.893
75	1.700
100	0.817

Bảng 12.7: Hướng dẫn chọn loạt các cảm biến họ LM34.

Mã ký hiệu	Dải nhiệt độ	Độ chính xác	Đầu ra
LM34A	-55 F to + 300 C	+ 2.0 F	10mV/F
LM34	-55 F to + 300 C	+ 3.0 F	10mV/F
LM34CA	-40 F to + 230 C	+ 2.0 F	10mV/F
LM34C	-40 F to + 230 C	+ 3.0 F	10mV/F
LM34D	-32 F to + 212 C	+ 4.0 F	10mV/F

Bảng 12.8: Hướng dẫn chọn loạt các cảm biến nhiệt họ LM35.

Mã sản phẩm	Dải nhiệt độ	Độ chính xác	Đầu ra
LM35A	-55 C to + 150 C	+ 1.0 C	10 mV/F
LM35	-55 C to + 150 C	+ 1.5 C	10 mV/F
LM35CA	-40 C to + 110 C	+ 1.0 C	10 mV/F
LM35C	-40 C to + 110 C	+ 1.5 C	10 mV/F
LM35D	0 C to + 100 C	+ 2.0 C	10 mV/F

Tính chất gắn liện với việc viết phần mềm cho các thiết bị phi tuyến như vậy đã đưa nhiều nhà sản xuất tung ra thị trường các loạt bộ cảm biến nhiệt tuyến tính. Các bộ cảm biến nhiệt đơn giản và được sử dụng rộng rãi bao gồm các loạt họ LM34 và LM35 của hãng National Semiconductor Corp.

12.2.5 Các bộ cảm biến nhiệt họ LM34 và LM35.

Loạt các bộ cảm biến LM34 là các bộ cảm biến nhiệt mạch tích hợp chính xác cao mà điện áp đầu ra của nó tỷ lệ tuyến tính vơi nhiệt độ Fahrenheit (xem hình 12.7). loạt LM34 không yêu cầu cân chỉnh bên ngoài vì vốn nó đã được cân chỉnh rồi. Nó đưa ra điện áp 10mV cho sự thay đổi nhiệt độ 1^0F. bảng 12.7 hướng dẫn ta chọn các cảm biến loạt LM34.

Loạt các bộ cảm biến LM35 cũng là các bộ cảm biến nhiệt mách tích hợp chính xác cao mà điện áp đầu ra của nó tỷ lệ tuyến tính với nhiệt độ theo thang độ Celsius. Chúng cũng không yêu cầu cân chỉnh ngoài vì vốn chúng đã được cân chỉnh. Chúng đưa ra điện áp 10Mv cho mỗi sự thay đổi 1^0C. Bảng 12.8 hướng dẫn ta chọn các cảm biến họ LM35.

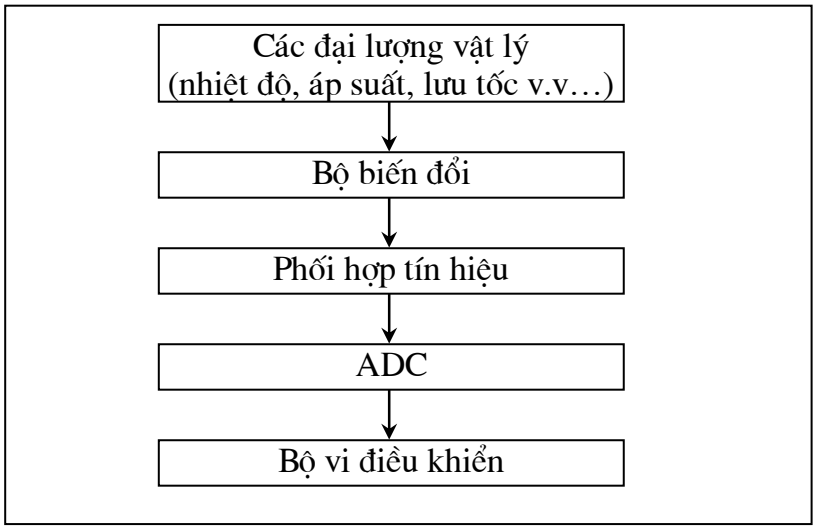
12.2.6 Phối hợp tín hiệu và phối ghép LM35 với 8051.

Phối hợp tín hiệu là một thuật ngữ được sử dụng rộng rãi trong lĩnh vực thu đo dữ liệu. Hầu hết các bộ biến đổi đều đưa ra các tín hiệu điện dạng điện áp, dòng điện, dung kháng hoặc trở kháng. Tuy nhiên, chúng ta cần chuyển đổi các tín hiệu này về điện áp nhằm gửi đầu vào đến bộ chuyển đổi ADC. Sự chuyển đổi (biến đổi) này được gọi chung là phối hợp tín hiệu. Phối hợp tín hiệu có thể là việc chuyển đổi dòng điện thành điện áp hoặc sự khuyếch đại tín hiệu. Ví dụ, bộ cảm biến nhiệt thay đổi trở kháng với nhiệt độ. Sự thay đổi trở kháng phải được chuyển thành điện áp để có thể được sử dụng cho các ADC. Xét trường hợp nối một LM35 tới một ADC 804 vì ADC 804 có độ phân dải 8 bít với tối đa 256 bước (2^8) và LM35 (hoặc ML34) tạo điện áp 10mV cho mỗi

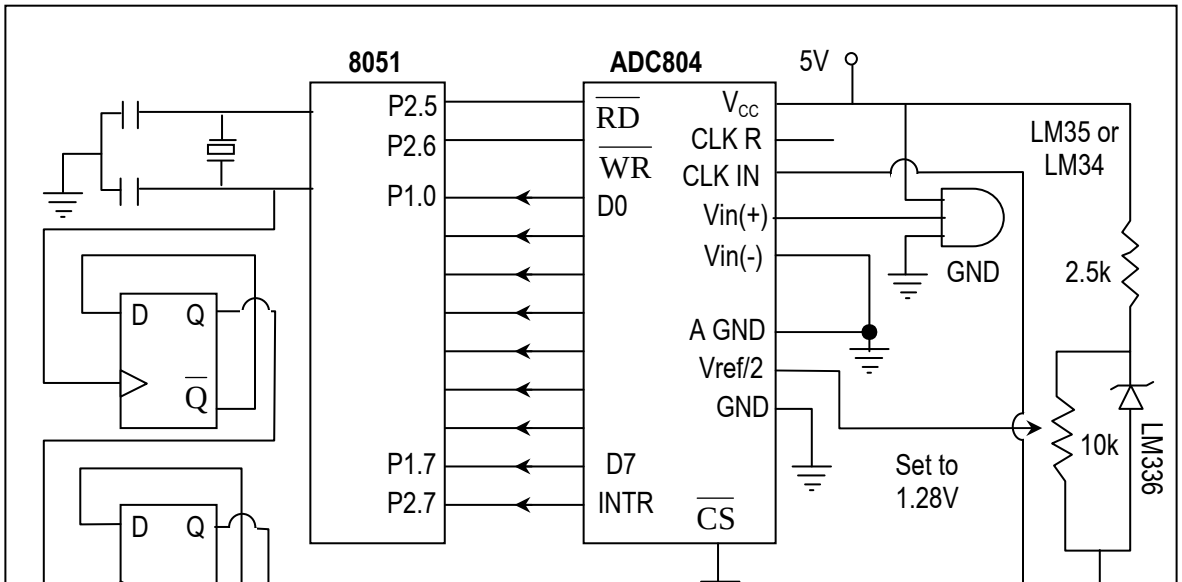
sự thay đổi nhiệt độ 1^0C nên ta có thể tạo điều kiện V_{in} của ADC 804 tạo ra một V_{out} = 2560mV (2,56V) cho đầu ra đầu thang đo. do vậy, nhằm tạo ra V_{out} đầy thang 2,56V cho ADC 804 ta cần đặt điện áp $V_{ref}/2$ = 1,28V. Điều này làm cho V_{out} của ADC 804 đáp ứng trực tiếp với nhiệt độ được hiển thị trên LM35 (xem bảng 12.9). Các giá trị của $V_{ref}/2$ được cho ở bảng 12.5.

Bảng 12.9: Nhiệt độ.

Nhiệt độ (^{0}C)	**V_{in} (mV)**	**V_{out} (D7 – D0)**
0	0	0000 0000
1	10	0000 0001
2	20	0000 0010
3	30	0000 0011
10	100	0000 1010
30	300	0001 1110



Hình 12.9: Thu đo các đại lượng vật lý.



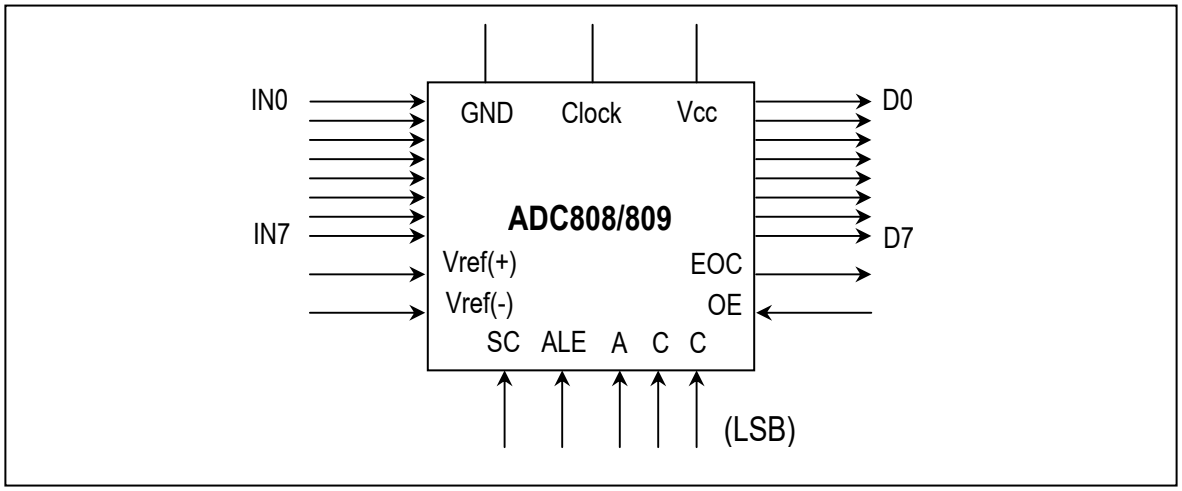
Hình 12.10

Hình 12.10: Nối ghép 8051 với DAC 804 và cảm biến nhiệt độ.

Hình 12.10 biểu diễn nối ghép của bộ cảm biến nhiệt đến ADC 804. Lưu ý rằng ta sử dụng đi ốt zener LM336 - 2.5 để cố định điện áp qua biến trở 10kΩ tại 2,5V. Việc sử dụng LM336 - 2.5 có thể vượt qua được mọi dao động lên xuống của nguồn nuôi.

12.2.7 Chíp ADC 808/809 với 8 kênh tương tự.

Một chíp hữu ích khác của National Semiconductor là ADC 808/809 (xem hình 12.11). Trong khi ADC 804 chỉ có một đầu vào tương tự thì chíp này có 8 kênh đầu vào. Như vậy nó cho phép ta hiển thị lên 8 bộ biến đổi khác nhau chỉ qua một chíp duy nhất. Lưu ý rằng, ADC 808/809 có đầu ra dữ liệu 8 bít như ADC 804. 8 kênh đầu vào tương tự được dồn kênh và được chọn theo bảng 12.10 sử dụng ba chân địa chỉ A, B và C.



Hình 12.11: Bộ biến đổi ADC 808/809.

Bảng 12.10: Chọn kênh tương tự của ADC 808.

Chọn kênh tương tự	C	B	A
IN0	0	0	0
IN1	0	0	1
IN2	0	1	0

IN3	0	1	1
IN4	1	0	0
IN5	1	0	1
IN6	1	1	0
IN7	1	1	1

Trong ADC 808/809 thì $V_{rer}(+)$ và $V_{ref}(-)$ thiết lập điện áp tham chiếu. Nếu $V_{ref}(-1)$ = Gnd và $V_{ref}(+)$ = 5V thì độ phân dải là 5V/256 = 19,53mV. Do vậy, để có độ phân dải 10mV ta cần đặt $V_{ref}(+)$ = 2,56V và $V_{ref}(-)$ = Gnd. Từ hình 12.11 ta thấy có chân ALE. Ta sử dụng các địa chỉ A, B và C để chọn kênh đầu vào IN0 – IN7 và kích hoạt chân ALE để chốt địa chỉ. Chân SetComplete để bắt đầu chuyển đổi (Start Conversion). Chân EOC được dùng để kết thúc chuyển đổi (End - Of - Conversion) và chân OE là cho phép đọc đầu ra (Out put Enable).

12.2.7 Các bước lập trình cho ADC 808/809.

Các bước chuyển dữ liệu từ đầu vào của ADC 808/809 vào bộ vi điều khiển như sau:

1. Chọn một kênh tương tự bằng cách tạo địa chỉ A, B và C theo bảng 12.10.
2. Kích hoạt chân ALE (cho phép chốt địa chỉ Address Latch Enable). Nó cần xung thấp lên cao để chốt địa chỉ.
3. Kích hoạt chân SCbằng xung cao xuống thấp để bắt đầu chuyển đổi.
4. Hiển thị OEC để báo kết thúc chuyển đổi. Đầu ra cao - xuống - thấp báo rằng dữ liệu đã được chuyển đổi và cần phải được lấy đi.
5. Kích hoạt OE cho phép đọc dữ liệu ra của ADC. Một xung cao xuống thấp tới chân OE sẽ đem dữ liệu số ra khỏi chíp ADC.

Lưu ý rằng trong ADC 808/809 không có đồng hồ riêng và do vậy phải cấp xung đồng bộ ngoài đến chân CLK. Mặc dù tốc độ chuyển đổi phụ thuộc vào tần số đồng hồ được nối đến CLK nhưng nó không nhanh hơn 100ms.

CHƯƠNG 13

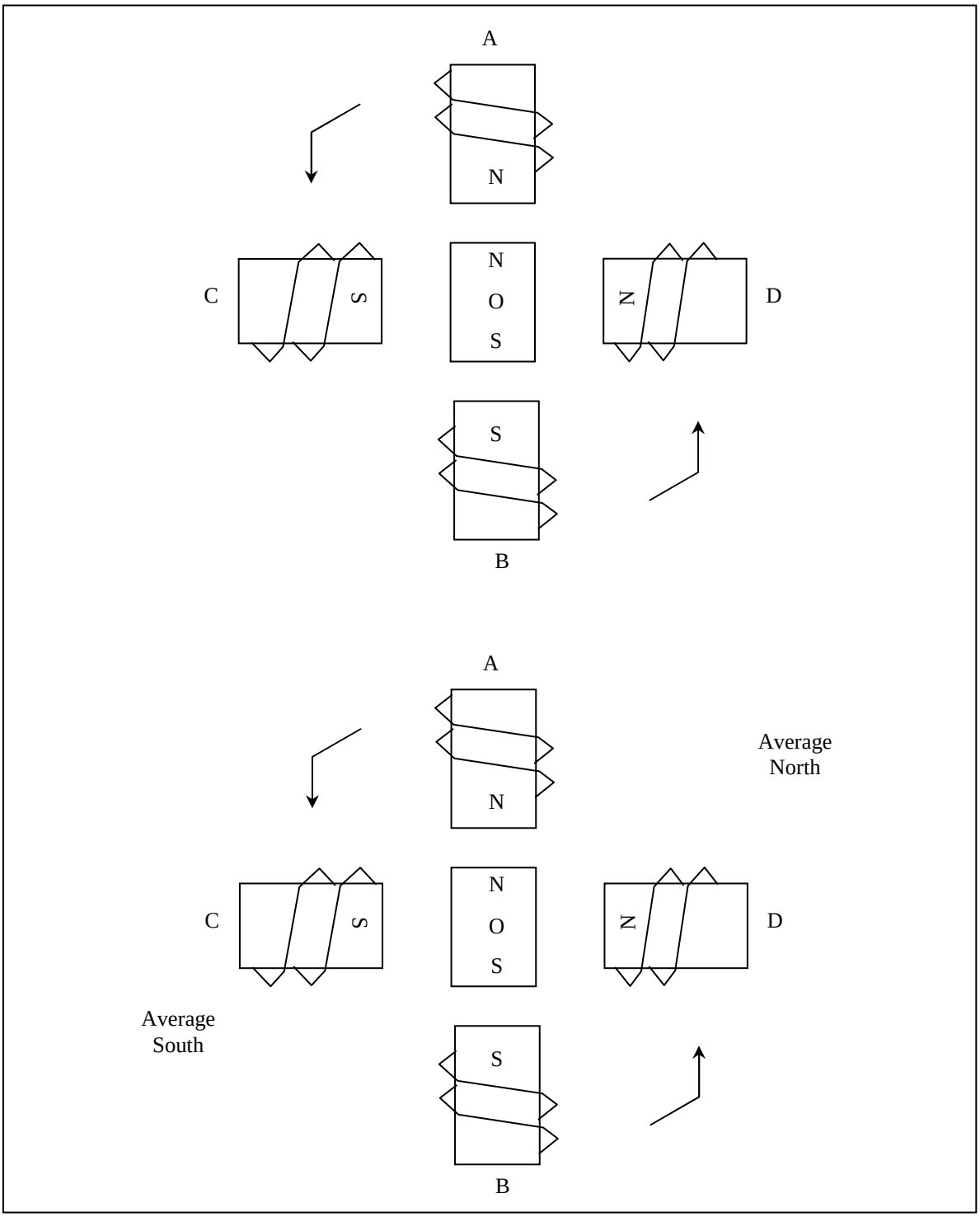
Phối ghép với thế giới kiểu II động cơ bước, bàn phím và các bộ DAC

13.1 Phối ghép với một động cơ bước.

Phần này bắt đầu với việc giới thiệu tổng quan về hoạt động của các động cơ bước. Sau đó chúng ta mô tả cách phối ghép một động cơ bước với bộ vi điều khiển 8051. Cuối cùng ta sử dụng các chương trình hợp ngữ để trình diễn điều khiển góc và hướng quay của động cơ bước.

13.1.1 Các động cơ bước.

Động cơ bước là một thiết bị sử dụng rộng rãi để chuyển các xung điện thành chuyển động cơ học. Trong các ứng dụng chẳng hạnnhư các bộ điều khiển đĩa, các máy in kim ma trận và các máy rô-bốt thì động cơ bước được dùng để điều khiển chuyển động. Mỗi động cơ bước đều có phần quay rôto là nam châm vĩnh cửu (cũng còn được gọi là trục dẫn - shaft) được bao bọc xung quanh là một **đứng** yên gọi stato (xem hình 131.1). Hầu hết các động cơ bước đều có chung có 4 stato mà các cuộn dây của chúng được bố trí theo cặp đối xứng với điểm giữa chung (xem hình 13.2), Kiểu động cơ bước này nhìn chung còn được coi như động cơ bước 4 pha. Điểm giữa cho phép một sự thay đổi của hướng dòng của một trong hai lõi khi một cuộn dây được nối đất tạo ra sự thay đổi cực của stato. Lưu ý rằng, trục của một động cơ truyền thống thì quay tự do, còn trục của động cơ bước thì chuyển động theo một độ tăng cố định lặp lại để cho phép ta chuyển dịch nó đến một vị trí chính xác. Chuyển động cố định lặp lại này có được là nhờ kết quả của lý thuyến từ trường cơ sở là các cực cùng dấu thì đẩy nhau và các cực khác dấu thì hút nhau. Hướng quay được xác định bởi từ trường của stato. Từ trường của stato được xác định bởi dòng chạyt quan lõi cuộn dây. khi hướng của dòng thay đổi thì cực từ trường cũng thay đổi gây ra chuyển động ngược lại của động cơ (đảo chiều). Động cơ bước được nối ở đây có 6 đầu dây: 4 đầu của cuộn dây stato và hai đầu dây chung điểm giữa của các cặp dây. Khi chuỗi xung nguồn được cấp đến mỗi cuộn dây stato thì động cơ sẽ quay. Có một số chuỗi xung được sử dụng rộng rãi với cấp độ chính xác khác nhau. Bảng 13.1 trình bày chuỗi 4 bước thông thường.



Hình 13.1: Căn chỉnh rôto.
Bảng 13.1: Chuỗi nguồn nuôi 4 bước thông thường.

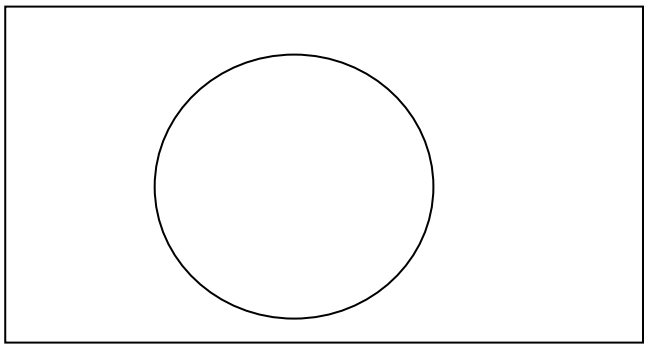
Chiều kim đồng hồ

Bước	Cuộn dây A	Cuộn dây B	Cuộn dây C	Cuộn dây D
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

Chiều quay bộ đếm

Bảng13.2: Các góc bước của động cơ bước.

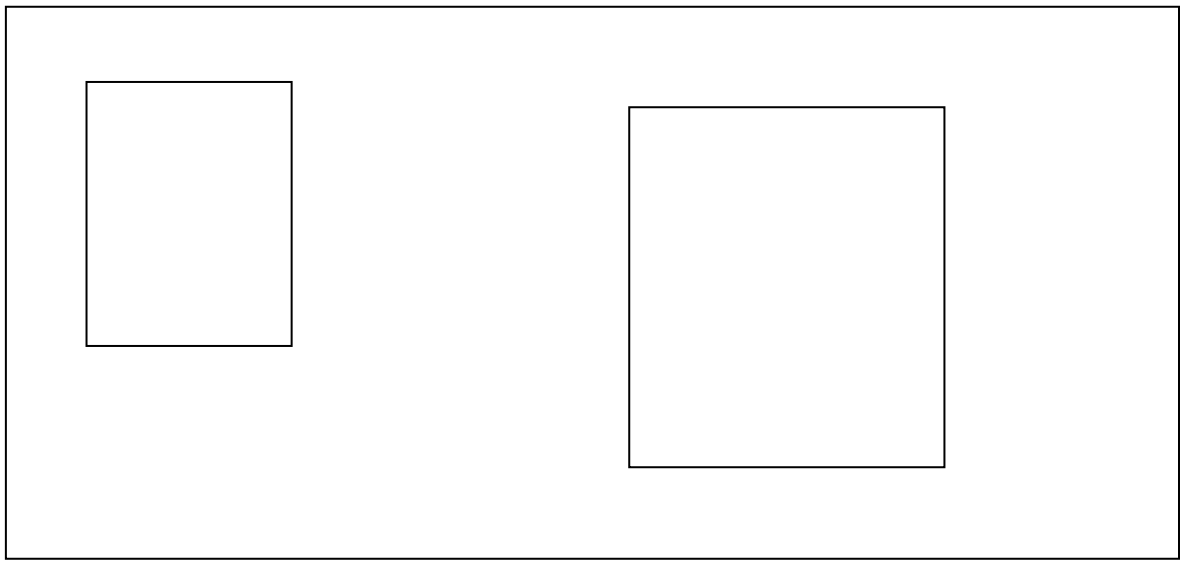
Góc bước	Số bước/ vòng
0.72	500
1.8	200
2.0	180
2.5	144
5.0	72
7.5	48
15	24



Hình 13.2:

Bảng13.2: Các góc bước của động cơ bước.

Hình 13.2: Bố trí các cuộn dây của stato.



Hình 13.3: Phối ghép 8051 với một động cơ bước.

Cần phải nhớ rằng mặc dù ta có thể bắt đầu với các chuỗi bất kỳ trong bảng 13.1. Nhưng khi đã bắt đầu thì ta phải tiếp tục với các chuỗi theo đúng thứ tự. Ví dụ ta bắt đầu bước thứ ba là chuỗi (0110) thì ta phải tiếp tục với chuỗi của bước 4 rồi sau đó 1, 2, 3 v.v...

Ví dụ 13.1:

Hãy mô tả kết nối 8051 với động cơ bước của hình 13.3 và viết một chương trình quay nó liên tục.

Lời giải:

Các bước dưới dây trình bày việc kết nối 8051 với động cơ bước và lập trình của nó.

1. Sử dụng một ôm kế để đo trở kháng của các đầu dây. Điều này xác định đầu chung (COM) nào được nối tới cuộn dây nào?
2. Các dây chung được nối tới đầu dương của nguồn cấp cho động cơ. Trong nhiều động cơ thì + 5V là đủ.
3. Bốn đầu củ cuộn dây stato được điểu khiển bởi 4 bít của cổng P1 trong 8051 (P1.0 - P1.3). Tuy nhiên, vì 8051 không đủ dòng để điều khiển các cuộn dây động cơ bước nên ta phải sử dụng một bộ điều khiển chẳng hạn như ULN2003 để cấp năng lượng cho stato. Thay cho ULN2003 ta có thể sử dụng các bóng bán dẫn làm các bộ điều khiển như chỉ ra trên hình 13.4. Tuy nhiên ta để ý rằng, nếu các bóng bán dẫn được sử dụng như các bộ điều khiển chúng ta cũng phải sử dụng các đi ôt để ngăn dòng cảm ứng được tạo ra khi tắt cuộn dây. Một lý do mà ULN2003 được ưu chuộng hơn các bóng bán dẫn như các bộ điều khiển là nó có đi ốt bên trong để ngăn cảm ứng điện từ ngược.

```
        MOV     A, # 66H        ; Nạp chuỗi xung bước
BACK:   MOV     P1, A           ; Xuất chuỗi xung đến động cơ
        RR      A               ; Quay theo chiều kim đồng hồ
        ACALL   DELAY           ; Chờ
        SJMP    BACK            ; Tiếp tục chạy
        --------
DELAY
        MOV     R2, # 100
H1:     MOV     R3, # 255
H2:     DJNZ    R3, H2
        DJNZ    R2, H1
        RET
```

Hãy thay đổi giá trị của DELAY để đặt tốc độ quay. Ta có thể sử dụng lệnh đơn bít SETB và CLR thay cho lệnh RRA để tạo ra chuỗi xung.

13.1.2 Góc bước (Step Angle).

Vậy mỗi bước có độ dịch chuyển là bao nhiêu? Điều này phụ thuộc vào cấu trúc bên trong của động cơ, đặc biệt là số răng của stato và rô to. Góc bước là độ quay nhỏ nhất của một bước. Các động cơ khác nhau có các góc bước khác nhau. Bảng 13.2 trình bày một số góc bước đối với các động cơ khác nhau. Bảng 13.2 có sử dụng thuật ngữ số bước trong một vòng (Steps per revolution). Đây là tổng số bước cần để quay hết một vòng 360^0 (chẳng hạn 180 bước $\times$ 2^0 = 360^0).

Cần phải nói rằng dường như trái ngược với ấn tượng ban đầu. Một động cơ bước không cần nhiều đầu dây cho stato hơn để có các bước nhỏ hơn. Tất cả mọi động cơ bước được nói ở đây chỉ có 4 đầu dây cho cuộn dây stato và 2 đầu dây chung cho nút giữa. Mặc dù nhiều hãng sản xuất chỉ dành một đầu chung thay cho hai thì họ cũng vẫn phải có 4 đầu cuộn dây stato.

13.1.3 Quan hệ số bước trong giây và số vòng quay trong phút RPM.

Quan hệ giữa số vòng quay trong phút RPM (revolutions per minute), số bước trong vòng quay và số bước trong vòng giây là quan hệ thuộc về trực giác và nó được biểu diễn như sau:

$$\text{So buoc trong giay} = \frac{\text{RPM} \times \text{So buoc trong vong quay}}{60}$$

13.1.4 Chuỗi xung bốn bước và số răng trên rô to.

Chuỗi xung chuyển mạch được trình bày trong bảng 13.1 được gọi là chuỗi chuyển mạch 4 bước bởi vì sau 4 bước thì hai cuộn dây giống nhau sẽ được bật "ON". Vậy độ dịch chuyển của 4 bước này sẽ là bao nhiêu? Sau mỗi khi thực hiện 4 bước này thì rô to chỉ dịch được một bước răng. Do vậy, trong động cơ bước với 200 bước/ vòng thì rô to của nó có 50 răng vì $50 \times 4 = 200$ bước cần để quay hết một vòng. Điều này dẫn đến một kết luận là góc bước tối thiểu luôn là hàm của số răng trên rô to. Hay nói cách khác góc bước càng nhỏ thì rô to quay được càng nhiều răng. Hãy xét ví dụ 13.2.

Ví dụ 13.2:

Hãy tính số lần của chuỗi 4 bước trong bảng 13.1 phải cấp cho một động cơ bước để tạo ra một dịch chuyển 80^0 nếu động cơ góc bước là 2^0.

Lời giải:

Một động cơ có góc bước là 2^0 thì phải có những đặc tính sau: góc bước 2^0, số bước/ vòng là 180^0, số răng của rô to là 45, độ dịch chuyển sau mỗi chuỗi 4 bước là 8^0. Vậy để dịch chuyển 80^0 thì cần 40 chuỗi 4 bước vì $10 \times 4 \times 2 = 80$.

Nhìn vào ví dụ 13.2 thì có người sẽ hỏi vậy muốn dịch chuyển đi 45^0 thì làm thế nào khi góc bước là 2^0. Muốn có độ phân giải nhỏ hơn thì tất cả mọi động cơ bước đều cho phép chuỗi chuyển mạch 8 bước, chuỗi 8 bước cũng còn được gọi chuỗi nửa bước (half - stepping), vì trong chuỗi 8 bước dưới đây thì mỗi bước là một nửa của góc bước bình thường. Ví dụ, một động cơ có góc bước là 2^0 có thể sử dụng góc bước 1^0 nếu áp dụng chuỗi ở bảng 13.3.

Bảng 13.3: Chuỗi xung 8 bước.

Chiều kim đồng hồ ↓

Bước	Cuộn A	Cuộn B	Cuộn C	Cuộn D
1	1	0	0	1
2	1	0	0	0
3	1	1	0	0
4	0	1	0	0
5	0	1	1	0
6	0	0	1	0
7	0	0	1	1
8	0	0	0	1

↑ Chiều quay bộ đếm

13.1.5 Tốc độ động cơ.

Tốc độ động cơ được đo bằng số bước trong một giây (bước/giây) là một hàm của tốc độ chuyển mạch. Để ý trong ví dụ 13.1 ta thấy rằng bằng việc thay đổi độ thời gian trễ ta có thể đạt được các tốc độ quay khác nhau.

13.1.6 Mô mem giữ.

Dưới đây là một định nghĩa về mô men giữ:

Mô men giữ là lượng mô men ngoài cần thiết để làm quay trục động cơ từ vị trí giữ của nó với điều kiện trục động cơ đang đứng yên hoặc đang quay với tốc độ RPM = 0. Đại lượng này được đo bằng tỷ lệ điện áp và dòng cấp đến động cơ. Đơn vị của mô men giữ là kilôgam - centimet (hay ounce - inch).

13.1.7 Chuỗi 4 bước điều khiển dạng sóng.

Ngoài các chuỗi 4 bước và 8 bước đã nó trên đây còn có một chuỗi khác được gọi là chuỗi 4 bước dạng sóng. Nó được trình bày trong bảng 13.4. Để ý 8 bước trong bảng 13.3 là một sự kết hợp đơn giản của các chuỗi 4 bước thường và chuỗi 4 bước điều khiển dạng sóng được cho ở bảng 13.1 và 13.4.

Chiều kim đồng hồ ↓

Bước	Cuộn dây A	Cuộn dây B	Cuộn dây C	Cuộn dây D
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

↑ Chiều quay bộ đếm

Hình 13.4: Sử dụng các bóng bán dẫn để điều khiển động cơ bước.

13.2 Phối ghép 8051 với bàn phím.

Các bàn phím và LCD là những thiết bị vào/ ra được sử dụng rộng rãi nhất của 8051 và cần phải thấu hiểu một cách cơ bản về chúng. Ở phần này trước hết ta giới thiệu các kiến thức cơ bản về bàn phím với cơ cấu ấn phím và tách phím, sau đó giới thiệu về giao tiếp 8051 với bàn phìm.

13.2.1 Phối ghép bàn phím với 8051.

Ở mức thấp nhất các bàn phím được tổ chức dưới dạng một ma trận các hàng và các cột. CPU truy cập cả hàng lẫn cột thông qua các cổng. Do vậy, với hai cổng 8 bít thì có thể nối tới một bàn phím 8×8 tới bộ vi xử lý. Khi một phím được ấn thì một hàng và một cột được tiếp xúc, ngoài ra không có sự tiếp xúc nào giữa các hàng và các cột. Trong các bàn phím máy tính IBM PC có một bộ vi điều khiển (bao gồm một bộ vi xử lý, bộ nhớ RAM và EPROM và một số cổng tất cả được bố trí trên một chíp) chịu trách nhiệm phối ghép phần cứng và phần mềm của bàn phím. Trong những hệ thống như vậy, nó là chức năng của các chương trình được lưu trong EPROM của bộ vi điều khiển để quét liên tục các phím, xác định xem phím nào đã được kích hoạt và gửi nó đến bo mạch chính. Trong phần này nghiên cứu về cơ cấu 8051 quét và xác định phím.

13.2.2 Quét và xác định phím.

Hình 13.5 trình bày một ma trận 4×4 được nối tới hai cổng. Các hàng được nối tới một đầu ra và các cột được nối tới một cổng vào. Nếu không có phím nào được ấn thì việc đóng cổng vào sẽ hoàn toàn là 1 cho tất cả các cột vì tất cả được nối tới dương nguồn V_{CC}. Nếu tất cả các hàng được nối đất và một phím được ấn thì một trong các cột sẽ có giá trị 0 vì phím được ấn tạo đường xuống đất. Chức năng của bộ vi điều khiển là quét liên tục để phát hiện và xác định phím được ấn.

Hình 13.5

Hình 13.5: Nối ghép bàn phím ma trận tới các cổng.

13.2.3 Nối đất các hàng và đọc các cột.

Để phát hiện một phím được ấn thì bộ vi điều khiển nối đất tất cả các hàng bằng cách cấp 0 tơío chốt đầu ra, sau đó nó đọc các hàng. Nếu dữ được đọc từ các cột là D3 - D0 = 1101 thì không có phím nào được ấn và quá trình tiếp tục cho đến khi phát hiện một phím được ấn. Tuy nhiên, nếu một trong các bít cột có số 0 thì điều đó có nghĩa là việc ấn phím đã xảy ra. Ví dụ, nếu D3 - D0 = 1101 có nghĩa là một phím ở cột 1 được ấn. Sau khi một ấn phím được phát hiện, bộ vi điều khiển sẽ chạy quá trình xác định phím. Bắt đầu với hàng trên cùng, bộ vi điều khiển nối đất nó bằng cách chỉ cấp mức thấp tới chân D0, sau đó nó đọc các cột. Nếu dữ liệu đọc được là toàn số 1 thì không có phím nào của hàng này được ấn và quá trình này chuyển sang hàng kế tiếp. Nó nối đất hàng kế tiếp, đọc các cột và kiểm tra xem có số 0 nào không? Qúa trình này tiếp tục cho đến khi xác định được hàng nào có phím ấn. Sau khi xác định được hàng có phím được ấn thì công việc tiếp theo là tìm ra phím ấn thuộc cột nào. Điều này thật là dễ dàng vì bộ vi điều khiển biết tại thời điểm bất kỳ hàng nào và cột nào được truy cập. Hãy xét ví dụ 13.3.

Ví dụ 13.3:

Từ hình 13.5 hãy xác định hàng và cột của phím được ấn cho các trường hợp sau đây:

a) D3 - D0 = 1110 cho hàng và D3 - D0 = 1011 cho cột.
b) D3 - D0 = 1101 cho hàng và D3 - D0 = 0111 cho cột.

Lời giải:

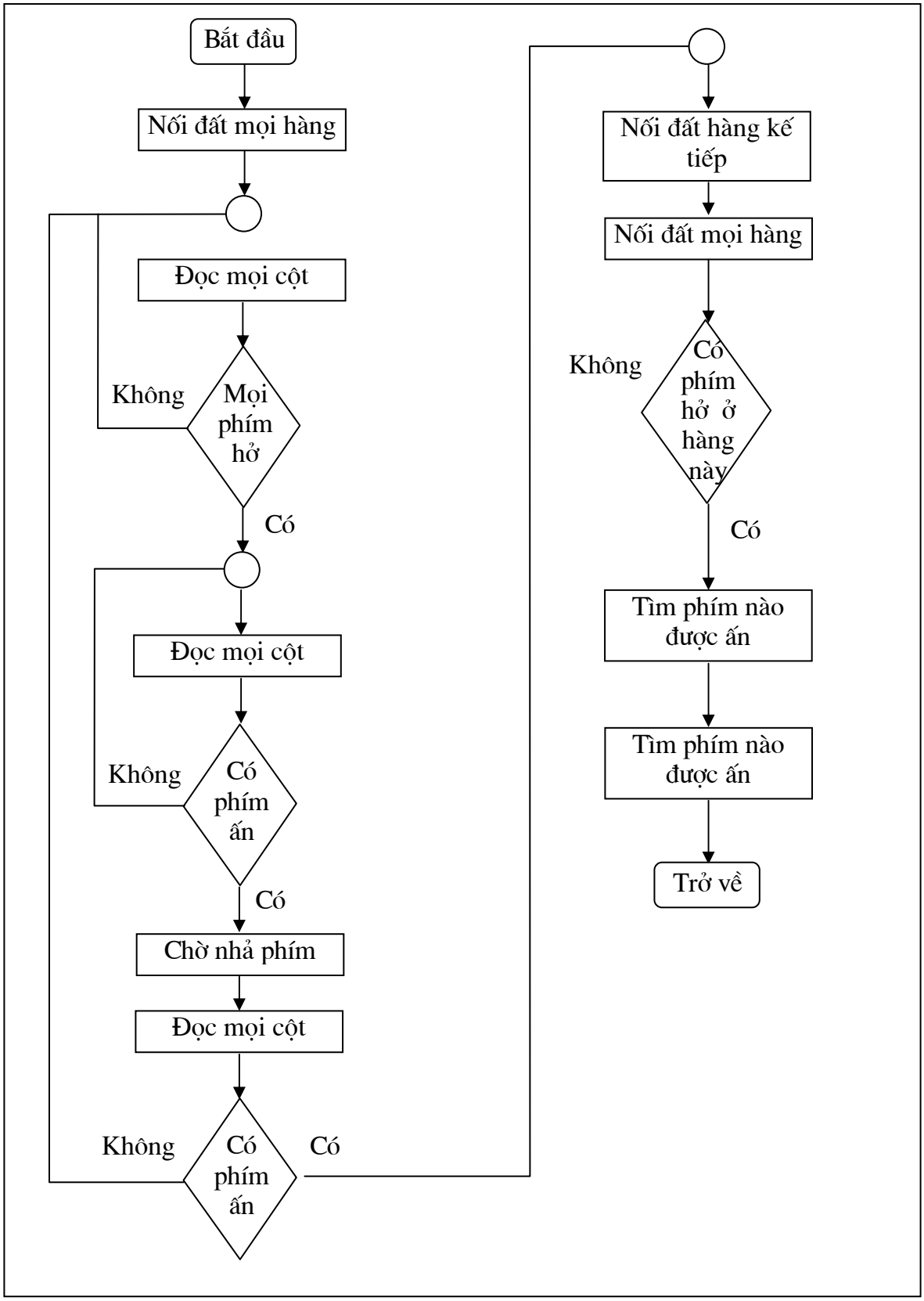
Từ hnhf 13.5 cột và hàng có thể được sử dụng xác định phím.

a) Hàng thuộc D0 và cột thuộc D2, do vậy phím số 2 đã được ấn.
b) Hàng thuộc D1 và cột thuộc D3, do vậy phím số 7 đã được ấn.

Chương trình 13.1 là chương trình hợp ngữ của 8051 để phát hiện và xác định sự kích hoạt phím. Trong chương trình này P1 và P2 được giả thiết là cổng ra và cổng vào tương ứng. Chương trình 13.1 đi qua 4 giai đoạn chính sau đây.

1. Khẳng định phím trước đó đã được nhả, các số không là đầu ra tới tất cả các hàng cùng một lúc và các cột được đọc và được kiểm tra chừng nào tất cả mọi cột đều cao. Khi tất cả các cột được phát hiện là đều cao thì chương trình chờ một thời gian ngắn trước khi nó chuyển sang giai đoạn kế tiếp để chờ một phím được ấn.
2. Để biết có một phím nào được ấn các cột được quét đi quét lại trong vòng vô tận cho đến khi có một cột có số 0. Hãy nhớ rằng các chốt đầu ra được nối tới các hàng vẫn có các số 0 ban đầu (được cấp ở giai đoạn 1) làm cho chúng được nối đất. Sau khi phát hiện ấn phím, nó đợi 20ms chờ cho phím nhả ra và sau đó quét lại các cột. Điều này phục vụ hai chức năng: a) nó đảm bảo rằng việc phát hiện ấn phím đầu tiên không bị sai do nhiễu và b) thời gian giữ chậm là 20ms ngăn ngừa việc ấn cùng một phím như là nhiều lần ấn. Nếu sau 20ms giữ chậm mà phím vẫn được ấn nó chuyển sang giai đoạn kế tiếp để phát hiện phím ấn thuộc hàng nào, nếu không nó quay trờ vòng lặp để phát hiện có một phím ấn thật.
3. Để phát hiện ấn phím thuộc hàng, nó nối đất mỗi hàng tại một thời điểm, đọc các cột mỗi lần. Nếu nó phát hiện tất cả mọi cột đều cao, điều này có nghĩa là ấn phím không thuộc hàng đó, do vậy nó nối đất hàng kế tiếp và tiếp tục cho đến khi phát hiện ra hàng có phím ấn. Khi tìm hàng có phím ấn, nó thiết lập địa chỉ bắt đầu cho bảng trình bày giữ các mã quét (hoặc giá trị ASCII) cho hàng đó và chuyển sang giai đoạn kế tiếp để xác định phím.

4. Để xác định phím ấn, nó quay các bít cột, mỗi lần một bít vào cờ nhớ và kiểm tra xem nó có giá trị thấp không? Khi tìm ra số 0, nó kéo mã ASCII dành cho phím đó ra từ bảng trình bày. Nếu không tìm được số 0 thì nó tăng con trỏ để chỉ đến phần tử kế tiếp của bảng trình bày. Hình 13.6 trình bày lưu đồ quá trình tìm phím ấn này.



Hình 13.6: Lưu đồ tìm phím ấn của chương trình 13.1.

Trong khi việc phát hiện ấn là chuẩn cho tất cả mọi bàn phím thì quá trình xác định phím nào được ấn lại không giống nhau. Phương phát sử dụng bảng trình bày được đưa ra trong chương trình 13.1 có thể được sửa đổi để làm việc với bất kỳ ma trận kích thước 8×8 nào. Hình 13.6 là lưu đồ thuật toán của chương trình 13.1 để quét và xác định phím ấn.

Có những chíp IC chẳng hạn như MM74C924 của hãng National Semiconductor kết hợp việc quét và giải mã bàn phím tất cả vào một chíp. Các chíp như vậy sử sự kết hợp các bộ đếm và các cổng lô gíc (không phải bộ vi điều khiển) để thực thi các khái niệm được trình bày trong chương trình 13.1 dưới đây.

Chương trình 13.1:

```
; Chương trình con bàn phím Keyboard này gửi mã ASCII
; Cha phím được ấn đến chân P0.1
; Các chân P1.0 – P1.3 được nối tới các hàng còn P2.0 – P2.3 tới các cột.
```

13.3 Phối ghép một DAC với 8051.

Phần này sẽ trình bày cách phối ghép một bộ biến đổi số tương tự DAC với 8051. Sau đó minh hoạ tạo một sóng hình sin trên máy hiện sóng sử dụng bọ DAC.

13.3.1 Bộ biến đổi số - tương tự DAC.

Bộ biến đổi - tương tự DAC là mmột thiết bị được sử dụng rộng rãi để chuyển đổi các xung số hoá về các tín hiệu tương tự. Trong phần này ta giới thiệu cơ sở phối ghép một bộ DAC với 8051.

Xem lại các kiến thức điện tử số ta thấy có hai cách tạo ra bộ DAC: Phương phát trọng số nhị phân và phương trình thang R/2R. Nhiều bộ DAC dựa trên các mạch tổ hợp, bao gồm MC1408 (DAC808) được sử dụng trong phần này đều sử dụng phương pháp hình thang R/2R vì nó có thể đạt độ chính xác cao hơn. Tiêu chuẩn đánh giá một bộ DAC đầu tiên là độ phân giải hàm của số đầu vào nhị phân. Các độ phân giải chúng là 8, 10 và 12 bít. Số các đầu vào bít dữ liệu quyết định độ phân giải của bộ DAC, vì số mức đầu ra tương tự bằng 2^n với n là đầu vào bít dữ liệu. Do vậy, một bộ DAC 8 bít như DAC808 chẳng hạn có 256 mức đầu ra điện áp (dòng điện) rời rạc. Tương tự như vậy, một bộ DAC 12 bít cho 4096 mức điện áp rời rạc. cũng có các bộ DAC 16 bít nhưng chúng rất đắt.

13.3.2 Bộ biến đổi DAC MC1408 (hay DAC808).

Trong bộ ADC808 các đầu vào số được chuyển đổi thành dòng (I_{out}) và việc nối một điện trở tới chân I_{out} ta chuyển kết quả thành điện áp. dòng tổng được cấp bởi chân I_{out} là một hàm số nhị phân ở các đầu vào D0 – D7 của DAC808 và tham chiếu I_{ref} như sau:

$$I_{out} = I_{ref}\left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256}\right)$$

Trong đó D0 là bít thấp nhất LSB và D7 là bít cao nhất MSB đối với các đầu vào

Chương 14
Phối phép 8031/51 với bộ nhớ ngoài

14.1 Bộ nhớ bán dẫn.

Trong phần này ta nhớ về các kiểu loại bộ nhớ bán dẫn khác nhau và các đặc tính của chúng như dụng lượng, tổ chức và thời gian truy cập. Trong thiết kế của tất cả các hệ thống dựa trên bộ vi sử lý thì các bộ nhớ bán dẫn được dùng như hơi lưu giữ chương trình và dữ liệu chính. Các bộ nhớ bán dẫn được nối trực tiếp với CPU và chúng là bộ nhớ mà CPU đầu tiên hỏi về thông tin chương trình và dữ liệu. Vì lý do này mà các bộ nhớ nhiều khi được coi như là nó phải đáp ứng nhanh cho CPU mà các điều này chỉ có các bộ nhớ bán dẫn mới có thể làm được. Các bộ nhớ bán dẫn được sử dụng rộng rãi nhất là ROM và RAM. Trước khi đi vào bàn các kiểu bộ nhớ ROM và RAM chúng ta làm quen với một số thuật ngữ quan trọng chung cho tất cả mọi bộ nhớ bán dẫn như là dung lượng, tổ chức và tốc độ.

14.1.1 Dung lượng nhớ.

Số lượng các bít mà một chíp nhớ bán dẫn có thể lưu được gọi là dung lượng của chíp, nó có đơn vị có thể là ki-lô-bít (Kbít), mê-ga-bit (Mbít) v.v… Điều này phải phân biết với dung lượng lưu trữ của hệ thống máy tính. Trong khi dung lượng nhớ của một IC nhớ luôn được tính theo bít, còn dung lượng nhớ của một hệ thống máy tính luôn được cho tính byte. Chẳng hạn, trên tạp chí kỹ thuật có một bài báo nói rằng chíp 16M đã trở nên phổ dụng thì mặc dù nó không nói rằng 16M nghĩa là 16 mê-ga-bit thì nó vẫn được hiểu là bài báo nói về chíp IC nhớ. Tuy nhiên, nếu một quảng cáo nói rằng một máy tính với bộ nhớ 16M vì nó đang nói về hệ thống máy tính nên nó được hiểu 16M có nghĩa là 16 mê-ga-byte.

14.1.2 Tổ chức bộ nhớ.

Các chíp được tổ chức vào một số ngăn nhớ bên trong mạch tích hợp IC. Mỗi ngăn nhớ có chứa bộ bít, 4 bít, 8 bít thậm chí đến 16 bít phụ thuộc vào cách nó được thiết kế bên trong như thế nào? Số bít mà mỗi ngăn nhớ bên trong chíp nhớ có thể chứa được luôn bằng số chân dữ liệu trên chíp. Vậy có bao nhiêu ngăn nhớ bên trong một chíp nhớ? Nó phụ thuộc vào số chân địa chỉ, số ngăn nhớ bên trong một IC nhớ luôn bằng 2 luỹ thừa với số chân địa chỉ. Do vậy, tổng số bít mà IC nhớ có thể lưu trữ là bằng số ngăn nhớ nhân với bít dữ liệu trên mỗi ngăn nhớ. Có thể tóm tắt lại như sau:

1. Một chíp nhớ có thể chứa 2^x ngăn nhớ, với x là số chân địa chỉ.
2. Mỗi ngăn nhớ chứa y bít với y là số chân dữ liệu trên chíp.
3. Toàn bộ chíp chứa ($2^x \times y$)bít với x là số chân địa chỉ và y là số chân dữ liệu trên chíp.

14.1.3 Tốc độ.

Một trong những đặc tính quan trọng nhất của chíp nhớ là tốc độ truy cập dữ liệu của nó. Để truy cập dữ liệu thì địa chỉ phải có ở các chân địa chỉa, chân đọc READ được tích cực và sau một khoảng thời gian thì dữ liệu sẽ xuất hiện ở các chân dữ liệu. Khoảng thời gian này càng ngắn càng tốt và tất nhiên là chíp nhớ càng đắt. Tốc độ của chíp nhớ thường được coi như là thời gian truy cập của nó. Thời gian truy cập của các chíp nhớ thay đổi từ vài na-nô-giây đến hàng trăm na-nô-giây phụ thuộc vào công nghệ sử dụng trong quá trình thiết kế và sản xuất IC.

Cả ba đặc tính quan trọng của bộ nhớ là dung lượng nhớ, tổ chức bộ nhớ và thời gian truy cập sẽ được sử dụng nhiều trong chương trình. Bảng 14.1 như một

tham chiếu để tính toán các đặc tính của bộ nhớ. Các ví dụ 14.1 và 14.2 sẽ minh hoạ những khái niệm vừa trình bày.

Bảng 14.1: Dung lượng bộ nhớ với số chân địac chỉ của IC.

x	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
$\mathbf{2^x}$	1K	2K	4K	8K	16 K	32 K	64 K	12 8K	25 6K	51 2K	1 M	2 M	4 M	8 M	16 M

Ví dụ 14.1:

Một chíp nhớ có 12 chân địa chỉ và 4 chân dữ liệu. Hãy tìm tổ chức bộ nhớ và dung lượng nhớ của nó.

Lời giải:

a) Chíp nhớ này có 4096 ngăn nhớ ($2^{12} = 4096$) và mỗi ngăn nhớ chứa 4 bít dữ liệu nên tổ chức nhớ của nó là 4096×4 và thường được biểu diễn là $4K \times 4$.

b) Dung lượng nhớ của chíp nhớ là 16K vì có 4 K ngăn nhớ và mỗi ngăn nhớ có 4 bít dữ liệu.

Ví dụ 14.2:

Một chíp nhớ 512k có 8 chân dữ liệu. Hãy tìm a) tổ chức của nó và b) số chân địa chỉ của chíp này.

Lời giải:

a) Một chíp có 8 chân dữ liệu có nghĩa là mỗi ngăn nhớ có 8 bít dữ liệu. Số ngăn nhớ trên chíp này bằng dung lượng nhớ chia cho số chân dữ liệu = 512k/8 = 64. Do vậy tổ chức nhớ của chíp là $64k \times 8$.

b) Số đường địa chỉ của chíp sẽ là 16 vì $1^{16} = 64k$.

14.1.4 Bộ nhớ ROM.

Bộ nhớ ROM là bộ nhớ chỉ đọc (Read - only Memory). Đây là một kiểu bộ nhớ mà không mất các nội dung của nó khi tắt nguồn. Với lý do này mà bộ nhớ ROM còn được gọi là bộ nhớ không bay hơi, có nhiều loại bộ nhớ ROM khác nhau như: PROM, EPROM, EEPROM, EPROM nhanh (flash) và ROM che.

14.1.4.1 Bộ nhớ PROM.

Bộ nhớ PROM là bộ nhớ ROM có thể lập trình được. Đây là loại bộ nhớ mà người dùng có thể đốt ghi thông tin vào. hay nói cách khác, PROM là bộ nhớ người dùng có thể lập trình được. Đối với mỗi bít của PROM có một cầu chì. Bộ nhớ PROM được lập trình bằng cách làm đứt những cầu trì. Nếu thông tin được đốt vào trong PROM mà sau thì phải bỏ vì các cầu trì của nó bên trong bị đứt vĩnh viễn với lý do này mà PROM mà được gọi là bộ nhớ ROM lập trình một lần. Việc lập trình ROM cũng được gọi là đốt ROM và nó đòi hỏi phải có một thiết bị đặc biệt gọi là bộ đốt ROM hay còn gọi là thiết bị lập trình ROM.

14.1.4.2 Bộ nhớ EPROM và UV - EPROM.

Bộ nhớ EPROM được phát minh ra để cho phép thực hiện thay đổi nội dung của PROM sau khi nó đã được đốt. Trong bộ nhớ EPROM ta có thể lập trình chíp nhớ và xoá nó hàng nghìn lần. Điều này là đặc biệt cần thiết trong quá trình phát triển mẫu thử của một dự án dựa trên bộ vi xử lý. Một EPR sử dụng rộng rãi được gọi là UV - EPROM (UV là chữ viết tắt của tia cực tím Ultra - Violet). Vấn đề tồn tại duy nhất của UV - EPROM là thời gian xoá của nó quá lâu (20 phút).

Tất cả các chíp nhớ UV - EPROM có một cửa sổ được dùng để chiếu tia bức xạ cực tím xoá các nội dung của nó. Với lý do này mà EPROM cũng còn được coi như là bộ nhớ EPROM được xoá bằng tia cực tím hay đơn giản được gọi là UV – EPROM. Hình 14.1 trình bày các chân của một chíp UV - EPROM.

Để lập trình cho một UV - EPROM cần thực hiện các bước.

1. Xoá các nội dung của nó, để xoá một chíp thì phải tháo nó ra khỏi để cắm trên bảng mạch hệ thống và đặt nó vào thiết bị xoá EPROM để chiếu xạ tia cực tím khoảng 15 - 20 phút.
2. Lập trình cho chíp. Để lập trình cho một chíp UV - EPROM thì đặt nó vào thiết bị đốt (thiết bị lập trình). Để đốt chương trình và dữ liệu vào EPROM thì thiết bị đốt ROM sử dụng điện áp 12.5V hoặc cao hơn phụ thuộc vào loại EPROM. Điện áp này được gọi là Vpp trong bảng dữ liệu của UV - EPROM.
3. Lắp chíp nhớ trở lại để cắm trên hệ thống.

Từ các bước trên đây ta thấy cũng như các thiết bị đốt EPROM thì cũng có các thiết bị xoá EPROM khác nhau. Và tất cả các kiểu bộ nhớ UV – EPROM đều có một nhược điểm chính là không thể được lập trình trực tiếp trên bảng mạch của hệ thống. Do vậy, bộ nhớ EPROM đã được ra đời để giải quyết vấn đề này.

Hãy để ý đến các ký hiệu mã số của các IC trong bảng 14.2. ví dụ, mã số bộ phận 27128 - 25 dành cho UV - EPROM có dung lượn và thời gian truy cập là 250ns. Dung lượng của chíp nhỏ được ký hiệu trên mã số bộ phận (part number) và thời gian truy cập được bỏ đi một số 0. Trong các mã số bộ phận thì chữ C là cộng nghệ CMOS, còn 27xx luôn chỉ các chíp nhớ EPROM. Để biết thêm chi tiết ta vào mục sổ tay các chíp nhớ của các hãng chẳng hạn JAMECO (jameco.com) hay JDR (jdr.com) theo đường mạng internet.

Bảng 14.2: Một số chíp nhớ UV - EPROM.

HìNH 14.1

Hình 14.1: Bố trí các chân của họ các chíp nhớ ROM 27xx.

Ví dụ 14.3:

Đối với ROM có mã bộ phận là 27128 có dung lượng nhớ là 128k bít. Tra bảng ta thấy tổ chức của nó là 16k × 8 (tất cả mọi ROM đều có 8 chân dữ liệu) điều này nói lên rằng nó có 8 chân dữ liệu và số chân địa chỉ được tìm thấy là 14 vì 2^{14} = 16k.

14.1.4.3 Bộ nhớ PROM có thể xoá được bằng điện EEPROM.

Bộ nhớ EEPROM có một số ưu điểm so với EPROM là do vì được xoá bằng điện nên quá trình xoá rất nhanh, nhanh rất nhiều so với thời gian xoá 20 phút của UEPROM. Ngoài ra trong EEPROM ta phải xoá toàn bộ nội dung của ROM. Tuy nhiên, ưu điểm chính của EEPROM là ta có thể lập trình và xoá khi chíp nhớ vẫn ở trên giá cắm của bảng mạch hệ thống mà không phải lấy ra như đối với UV – EPROM. Hay nói cách khác là EPROM không cần thiết bị lập trình và thiết bị xoá ngoài như đối với UV - EPROM. Để hoàn toàn tạo thuận lợi cho EPROM nhà thiết kế phải kết hợp vào bảng mạch của hệ thống cả mạch điện để lập trình EEPROM sử dụng điện áp Vpp = 5V nhưng chúng rất đắt. nhìn chung, giá thành cho 1 bít của EEPROM đắt hơn rất nhiều so với UV - EPROM.

Bảng 14.3: Một số chíp EEPROM và chíp nhớ Flash (nhanh).

CHƯƠNG 15
Phép ghép 8031/51 với 8255

Như đã nói ở chương 14 trong quá trình nối ghép 8031/51 với bộ nhớ ngoài thì hai cổng P0 và P2 bị mất. Trong chương này chúng ta sẽ trình bày làm thế nào để mở rộng các cổng vào/ ra I/O của 8031/51 bằng việc nối nó tới chíp 8255.

15.1 Lập trình 8255.

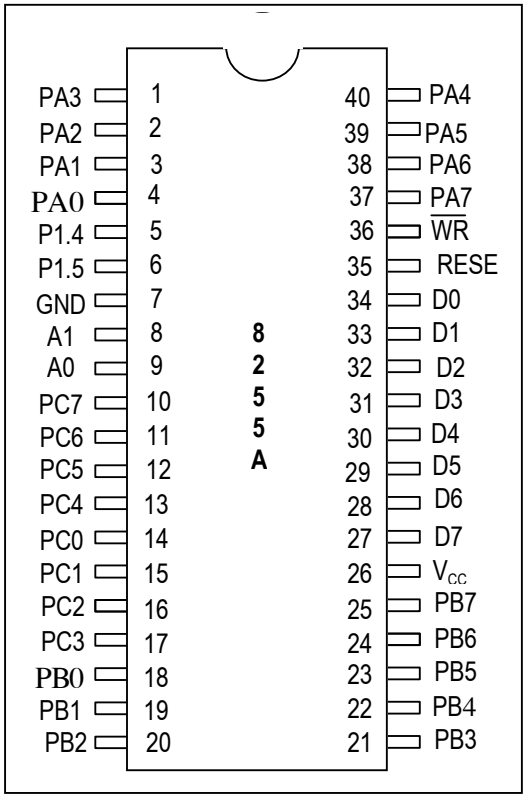
Trong mục này ta nghiên cứu 8255 như là một trong những chíp vào/ ra được sử dụng rộng rãi nhất. Trước hết ta mô tả những đặc tính của nó và sau đsó chỉ ra cách nối 8031/51 với 8255 như thế nào?

15.1 Lập trình 8255.

Trong mục này ta nghiên cứu 8255 như là một trong những chíp vào/ ra được sử dụng rộng rãi nhất. Trước hết ta mô tả những đặc tính của nó và sau đó chỉ ra cách nối 8031/51 với 8255 như thế nào?

15.1.1 Các đặc tính của 8255.

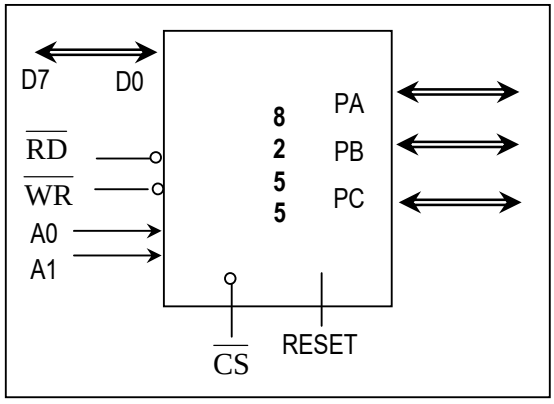
8255 là một chíp DIP 4 chân (xem hình 15.1). Nó có 3 cổng truy cập được riêng biệt. Các cổng đó có tên A, B và C đều là các cổng 8 bit. Các cổng này đều có thể lập trình như cổng đầu vào hoặc đầu ra riêng rẽ và có thể thay đổi một cách năng động. Ngoài ra, các cổng 8255 có khả năng bắt tay. Do vậy cho phép giao diện với các thiết bị khác cũng có giá trị tín hiệu bắt tay như các máy in chẳng hạn. Khả năng bắt tay của 8255 sẽ được bàn tới ở mục 15.3.



Hình 15.1: Chíp 8255.

15.1.1.1 Các chân PA0 - PA7 (cổng A).

Cả 8 bít của cổng A PA0 - PA7 có thể được lập trình như 8 bit đầu vào hoặc 8 bit đầu ra hoặc cả 8 bít hai chiều vào/ ra.S



Hình 15.2: Sơ đồ khối của 8255.

15.1.1.2 Các chân PB0 - PB7 (cổng B).

Cả 8 bit của cổng B có thể được lập trình hoặc như 8 bit đầu vào hoặc 8 bit đầu ra hoặc cả 8 bit hai chiều vào/ ra.

15.1.1.3 Các chân PC0 - PC7 (cổng C).

Tất cả 8 bit của cổng C (PC0 - PC7) đều có thể được lập trình như các bit đầu vào hoặc các bit đầu ra. 8 bit này cũng có thể được chia làm hai phần: Các bit cao (PC4 - PC7) là CU và các bit thấp (PC0 - PC3) là CL. Mỗi phần có thể được dùng hoặc làm đầu vào hoặc làm đầu ra. Ngoài ra từng bit của cổng C từ PC0 - PC7 cũng có thể được lập trình riêng rẽ.

15.1.1.4 Các chân $\overline{RD}$ và $\overline{WR}$.

Đây là hai tín hiệu điều khiển tích cực mức thấp tới 8255 được nối tới các chân dữ liệu $\overline{RD}$ và $\overline{WR}$ từ 8031/51 được nối tới các chân đầu vào này.

15.1.1.5 Các chân dữ liệu D0 - D7.

Các chân dữ liệu D0 - D7 của 8255 được nối tới các chân dữ liệu của bộ vi điều khiển để cho phép nó gửi dữ liệu qua lại giữa bộ vi điều khiển và chíp 8255.

15.1.1.6 Chân RESET.

Đây là đầu vào tín hiệu tích cực mức cao tới 8255 được dùng để xoá thanh ghi điều khiển. Khi chân RESET được kích hoạt thì tất cả các cổng được khởi tạo lại như các cổng vào. Trong nhiều thiết kế thì chân này được nối tới đầu ra RESET của bus hệ thống hoặc được nối tới đất để không kích hoạt nó. Cũng như tất cả các chân đầu vào của IC thì nó cũng có thể để hở.

15.1.1.7 Các chân A0, A1 và $\overline{CS}$.

Trong khi $\overline{CS}$ chọn toàn bộ chíp thì A0 và A1 lại chọn các cổng riêng biệt. Các chân này được dùng để truy cập các cổng A, B, C hoặc thanh ghi điều khiển theo bảng 15.1. Lưu ý $\overline{CS}$ là tích cực mức thấp.

15.1.2 Chọn chế độ của 8255.

Trong khi các cổng A, B và C được dùng để nhập và xuất dữ liệu thì thanh ghi điều khiển phải được lập trình để chọn chế độ làm việc của các cổng này. Các cổng của 8255 có thể được lập trình theo một chế độ bất kỳ dưới đây.

1. Chế độ 0 (Mode0): Đây là chế độ vào/ ra đơn giản. Ở chế độ này các cổng A, B CL và CU có thể được lập trình như đầu vào hoặc đầu ra. Trong chế độ này thì tất cả các bit hoặc là đầu vào hoặc là đầu ra. Hay nói cách khác là không có điều khiển theo từng bit riêng rẽ như ta đã thấy ở các cổng P0 - P3 của 8051. Vì đa phần các ứng dụng liên quan đến 8255 đều sử dụng chế độ vào/ ra đơn giản này nên ta sẽ tập chung đi sâu vào chế độ này.

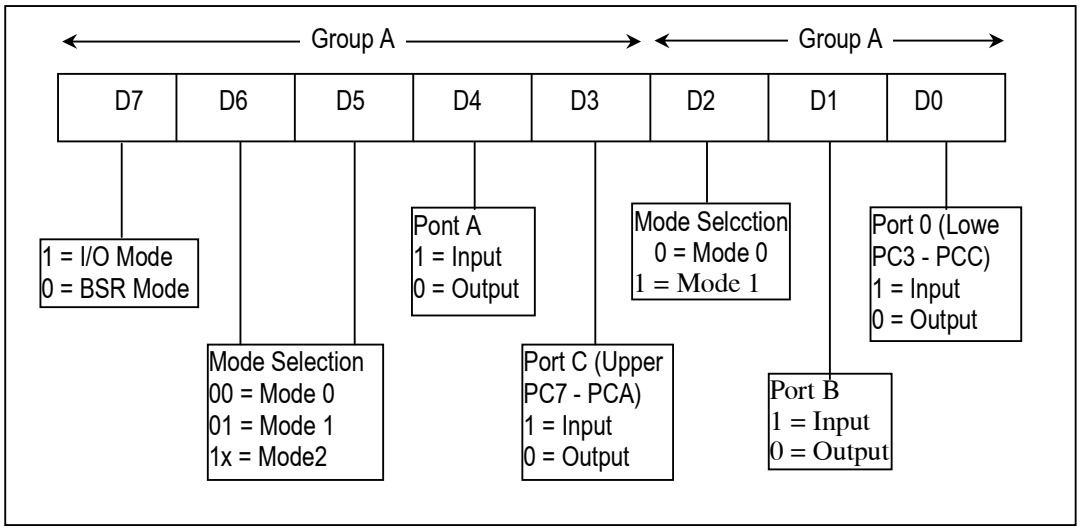
2. Chế độ 1 (Mode1): Trong chế độ này các cổng A và B có thể được dùng như các cổng đầu vào hoặc đầu ra với các khả năng bắt tay. Tín hiệu bắt tay được cấp bởi các bit của cổng C (sẽ được trình bày ở mục 15.3).

3. Chế độ 2 (Mode2): Trong chế độ này cổng A có thể được dùng như cổng vào/ ra hai chiều với khả năng bắt tay và các tín hiệu bắt tay được cấp bởi các bit cổng C. Cổng B có thể được dùng như ở chế độ vào/ ra đơn giản hoặc ở chế độ có bắt tay Mode1. Chế độ này sẽ không được trình bày trong tài liệu này.

Chế độ BSR: Đây là chế độ thiết lập/ xoá bit (Bit Set/ Reset). ở chế độ này chỉ có những bit riêng rẽ của cổng C có thể được lập trình (sẽ được trình bày ở mục 15.3).

Bảng 15.1: Chọn cổng của 8255.

$\overline{CS}$	A1	A0	Chọn cổng
0	0	0	Cổng A
0	0	1	Cổng B
0	1	0	Cổng C
0	1	1	Thanh ghi điều khiển
1	x	X	8255 không được chọn



Hình 15.3: Định dạng từ điều khiển của 8255 (chế độ vào/ ra).

15.1.3 Lập trình chế độ vào/ ra đơn giản.

Hãng Intel gọi chế độ 0 là chế độ vào/ ra cơ sở. Một thuật ngữ được dùng chung hơn là vào/ ra đơn giản. Trong chế độ này thì một cổng bất kỳ trong A, B, C được lập trình như là cổng đầu vào hoặc cổng đầu ra. Cần lưu ý rằng trong chế độ này một cổng đã cho không thể vừa làm đầu vào lại vừa làm đầu ra cùng một lúc.

Ví dụ 15.1:

Hãy tìm từ điều khiển của 8255 cho các cấu hình sau:

Tất cả các cổng A, B và C đều là các cổng đầu ra (chế độ 0).

PA là đầu vào, PB là đầu ra, PCL bằng đầu vào và PCH bằng đầu ra.

Lời giải:

Từ hình 15.3 ta tìm được:

a) 1000 0000 = 80H; b) 1001 000 = 90H

15.1.4 Nối ghép 8031/51 với 8255.

Chíp 8255 được lập trình một trong bốn chế độ vừa trình bày ở trên bằng cách gửi một byte (hãng Intel gọi là một từ điều khiển) tới thanh ghi điều khiển của 8255. Trước hết chúng ta phải tìm ra các địa chỉ cổng được gán cho mỗi cổng A, B, C và thanh ghi điều khiển. Đây được gọi là ánh xạ cổng vào/ ra (mapping).

Như có thể nhìn thấy từ hình 15.4 thì 8255 được nối tới một 8031/51 như thế nó là bộ nhớ RAM. Để việc sử dụng các tín hiệu $\overline{RD}$ và $\overline{WR}$. Phương pháp nối một chíp vào/ ra bộ nhớ vì nó được ánh xạ vào không gian bộ nhớ. Hay nói cách khác, ta sử dụng không gian bộ nhớ để truy cập các thiết bị vào/ ra. Vì ly??? do này mà ta dùng lệnh MOVX để truy cập RAM và ROM. Đối với một 8255

được nối tới 8031/51 thì ta cũng phải dùng lệnh MOVX để truyền thông với nó. Điều này được thể hiện trên ví dụ 15.2.

Ví dụ 15.2:

Đối với hình 15.4:

a) Hãy tìm các địa chỉ vào/ ra được gán cho cổng A, B, C và thanh ghi điều khiển.
b) Hãy lập trình 8255 cho các cổng A, B và C thành các cổng đầu ra.
c) Viết một chương trình để gửi 55H và AAH đến cổng liên tục.

Lời giải:

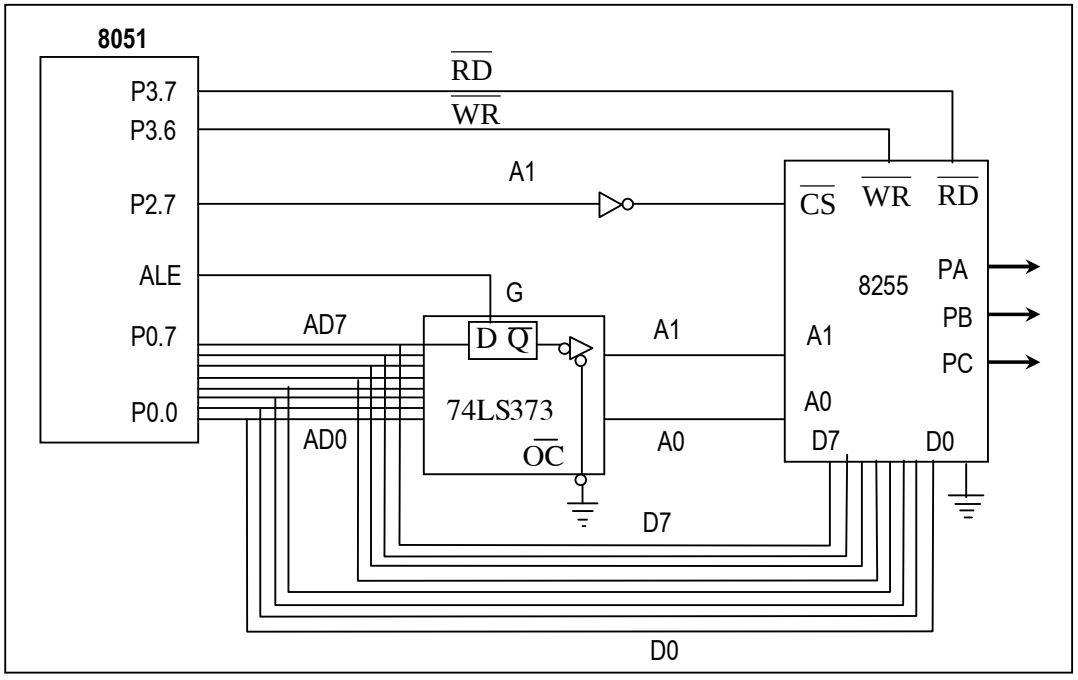
a) Địa chỉ cơ sở dành cho 8255 như sau:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
x	1	x	x	x	x	x	x	x	x	x	x	x	X	0	0	=4000HPA
x	1	x	x	x	x	x	x	x	x	x	x	x	X	0	1	=4000HPB
x	1	x	x	x	x	x	x	x	x	x	x	x	X	1	0	=4000HPC
x	1	x	x	x	x	x	x	x	x	x	x	x	X	1	1	=4000HCR

b) Byte (từ) điều khiển cho tất cả các cổng như đầu ra là 80H như được tính ở ví dụ 15.1.

c)

```
        MOV     A, #80H         ; Từ điển khiển
        MOV     DPTR, # 4003H   ; Nạp địa chỉ cổng của thanh ghi điều khiển
        MOVX    @DPTR, A        ; Xuất từ điển khiển
        MOV     A, # 55H        ; Gán A = 55
AGAIN:  MOV     DPTR, # 4000H   ; Địa chỉ cổng PA
        MOVX    @DPTR, A        ; Lấy các bit cổng PA
        INC     DPTR            ; Địa chỉ cổng PB
        MOVX    @DPTR, A        ; Lấy các bít cổng PB
        INC     DPTR            ; Địa chỉ cổng PC
        MOVX    @DPTR, A        ; Lấy các bít cổng PC
        CPL     A               ; Lấy các bít thanh ghi A
        ACALL   DELAY           ; Chờ
        SJMP    AGAIN           ; Tiếp tục
```



Hình 15.4: Nối ghép 8051 với 8255 cho ví dụ 15.2.

Ví dụ 15.3:

Đối với hình 15.5:

a) Tìm các địa chỉ cổng vào ra được gán cho các cổng A, B, C và thanhg ghi điều khiển.

b) Tìm byte điều khiển đối với PA bằng đầu vào, PB bằng đầu ra, PC bằng đầu ra

c) Viết một chương trình để nhận dữ liệu từ PA gửi nó đến cả cổng B và cổng C.

Lời giải:

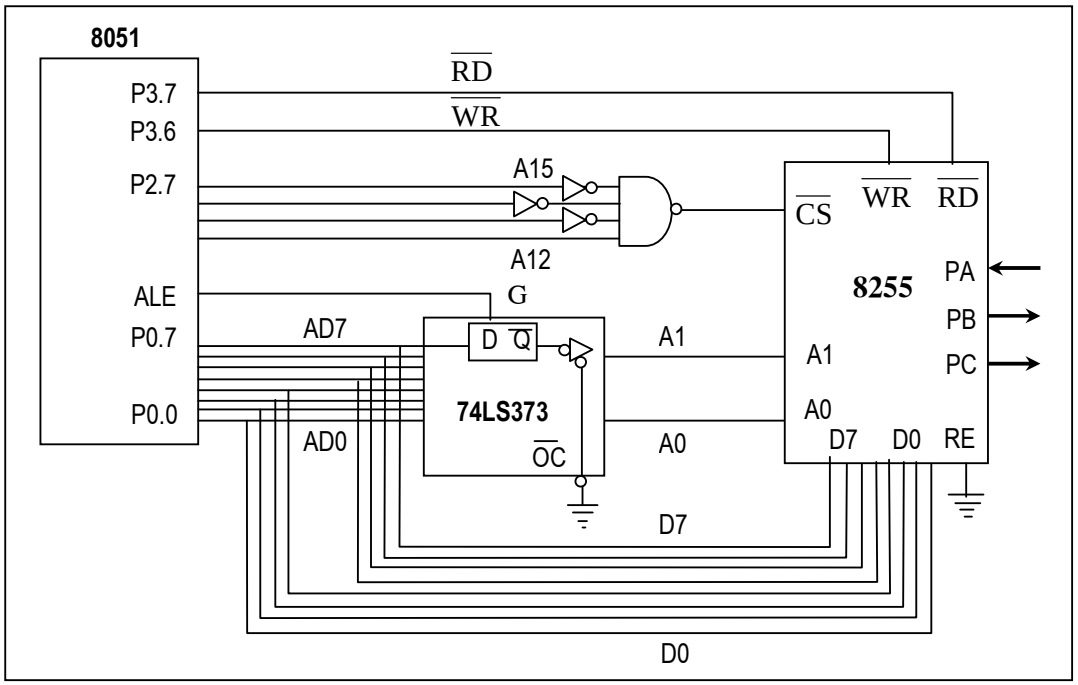
a) Giả sử tất các các bit không dùng đến là 0 thì địa chỉ cổng cơ sở cho 8255 là 1000H. Do vậy ta có:

1000H là PA; 1001H là PB; 1002H là PC và 1003H là thanh ghi điều khiển.

b) Từ điều khiển cho trường hợp này là 10010000 hay 90H.

c)

```
MOV     A, #90H ; PA là đầu vào, PB là đầu ra, PC là đầu ra
MOV     DPTR, #1003H   ; Nạp địa chỉ cổng của thanh ghi điều khiển
MOVX    @DPTR, A       ; Xuất từ điều khiển
MOV     DPTR, #1000H   ; Địa chỉ PA
MOVX    A, @DPTR       ; Nhận dữ liệu từ PA
INC     DPTR           ; Địa chỉ PB
MOVX    @DPTR, A       ; Gửi dữ liệu ra PB
INC     DPTR           ; Địa chỉ PC
MOVX    @DPTR, A       ; Gửi dữ liệu ra PC
```



Hình 15.5: Nối ghép 8051 tới 8255 cho ví dụ 15.3.

Đối với ví dụ 15.3 ta nên dùng chỉ lệnh EQU cho địa chỉ các cổng A, B, C và thanh ghi điều khiển CNTPORT như sau:

```
APORT  EQU    1000H
BPORT  EQU    1001H
CPORT  EQU    1002H
```

```
        CNTPORT     EQU    1003H

        MOV         A, #90H                 ; PA là đầu vào, PB là đầu ra, PC là đầu ra
        MOV         DPTR, #CNTPORT          ; Nạp địa chỉ của cổng thanh ghi điều khiển
        MOVX        @DPTR, A                ; Xuất từ điều khiển
        MOV         DPTR, #CNTPORT          ; Địa chỉ PA
        MOVX        DPTR, APORT             ; Nhận dữ liệu PA
        INC         A, @DPTR                ; Địa chỉ PB
        MOVX        DPTR                    ; Gửi dữ liệu ra PB
        INC         DPTR                    ; Địa chỉ PC
        MOVX        DPTR, A         ; Gửi dữ liệu ra PC
hoặc có thể viết lại như sau:
CONTRBYT    EQU     90H                     Xác định PA đầu vào, PB và PC đầu ra
BAS8255P    EQU     1000H                   ; Địa chỉ cơ sở của 8255

            MOV     A, #CONTRBYT
            MOV     DPTR, #BAS8255P+3       ; Nạp địa chỉ cổng C
            MOVX    @DPTR, A                ; Xuất từ điều khiển
            MOV     DPTR, #BASS8255P        ; Địa chỉ cổng A
            ...
```

Để ý trong ví dụ 15.2 và 15.3 ta đã sử dụng thanh ghi DPTR vì địa chỉ cơ sở gán cho 8255 là 16 bit. Nếu địa chỉ cơ sở dành cho 8255 là 8 bit, ta có thể sử dụng các lệnh "MOVX A, @R0" và "MOVX @R0, A" trong đó R0 (hoặc R1) giữ địa chỉ cổng 8 bit của cổng. Xem ví dụ 15.4, chú ý rằng trong ví dụ 15.4 ta sử dụng một cổng lôgíc đơn giản để giải mã địa chỉa cho 8255. Đối với hệ thống có nhiều 8255 ta có thể sử dụng 74LS138 để giải mã như sẽ trình bày ở ví dụ 15.5.

15.1.5 Các bí danh của địa chỉ (Addren Alias).

Trong các ví dụ 15.4 và 15.4 ta giải mã các bít địa chỉ A0 - A7, tuy nhiên trong ví dụ 15.3 và 15.2 ta đã giải mã một phần các địa chỉ cao của A8 - A15. Việc giải mã từng phần này dẫn đến cái gọi là các bí danh của địa chỉ (Address Aliases). Hay nói cách khác, cùng cổng vật lý giống nhau có các địa chỉ khác nhau, do vậy cùng một cổng mà được biết với các tên khác nhau. Trong ví dụ 15.2 và 15.3 ta có thể thay đổi tốt x thành các tổ hợp các số 1 và 0 khác nhau thành các địa chỉ khác nhau, song về thực chất chúng tham chiếu đến cùng một cổng vật lý. Trong tài liệu thuyết minh phần cứng của mình chúng ta cần phải bảo đảm ghi chú đầy đủ các bí danh địa chỉa nếu có sao cho mọi người dùng biết được các địa chỉ có sẵn để họ có thể mở rộng hệ thống.

Ví dụ 15.4:

Cho hình 15.6:

a) Hãy tìm các địa chỉ cổng vào/ ra được gán cho các cổng A, B, C và thanh ghi điều khiển.

b) Tìm từ điều khiển cho trường hợp PA là đầu ra, PB là đầu vào, PC - PC3 là đầu vào và CP4 - CP7 là đầu ra.

c) Viết một chương trình để nhận dữ liệu từ PB và gửi nó ra PA. Ngoài ra, dữ liệu từ PC1 được gửi đến CPU.

Lời giải:

a) Các địa chỉ cổng được tìm thấy như sau:

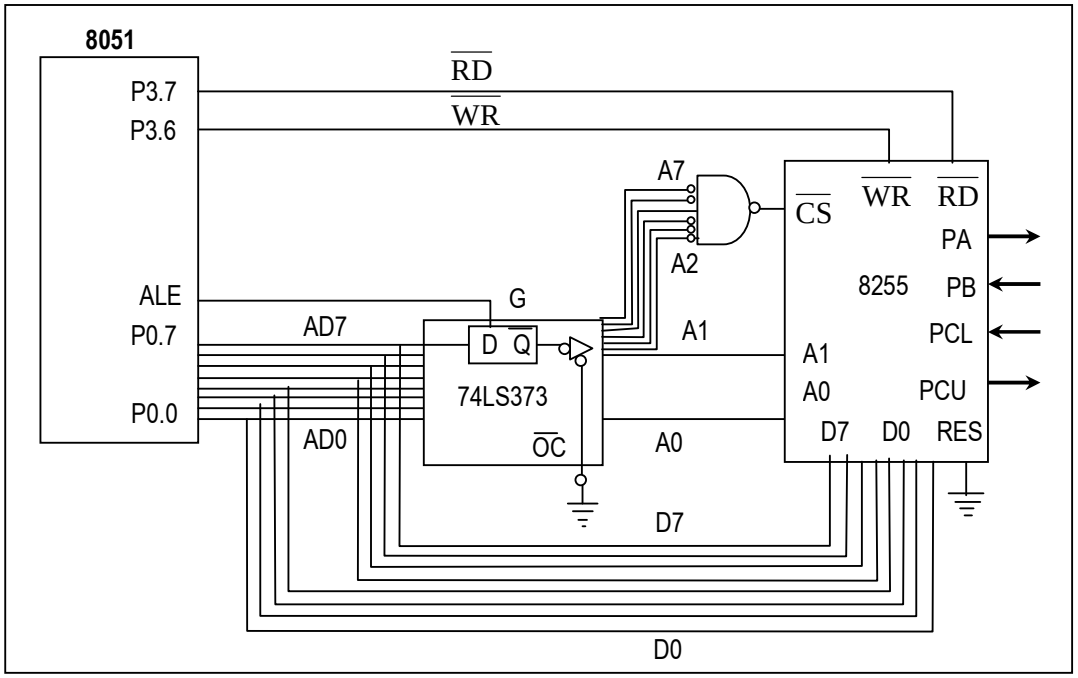
BB	$\overline{CS}$	A1	A0	Địa chỉ	Cổng
0010	00	0	0	20H	Cổng A
0010	00	0	1	21H	Cổng B

0010	00	1	0	22H	Cổng C
0010	00	1	1	23H	Thanh ghi điều khiển

b) Từ điều khiển là 10000011 hay 83H.

c)

```
CONTRBYT    EQU     83H         ; PA là đầu ra, PB,PCL là đầu vào
APORT       EQU     20H
BPORT       EQU     21H
CPORT       EQU     22H
CNTPORT     EQU     23H
            ...
            MOV     A, #CONTRBYT
            MOV     A, #CONTRBYT ; PA, PCU là đầu ra, PB và PCL là đầu vào
            MOV     R0, #CNTPORT ; Nạp địa chỉ của cổng thanh ghi điều khiển
            MOVX    @R0, A       ; Xuất từ điều khiển
            MOV     R0, #BPORT   ; Nạp địa chỉ PB
            MOVX    A, @R0       ; Đọc PB
            DEC     R0           ; Chỉ đến PA (20H)
            MOVX    @R0, A       ; Gửi nó đến PA
            MOV     R0, #CPORT   ; Nạp địa chỉ PC
            MOVX    A, @R0       ; Đọc PCL
            ANL     A, #0FH      ; Che phần cao
            SWAP    A            ; Trao đổi phần cao và thấp
            MOVX    @R0, A       ; Gửi đến PCU
```



Hình 15.6: Nối ghép 8051 với 8255 cho ví dụ 15.4.

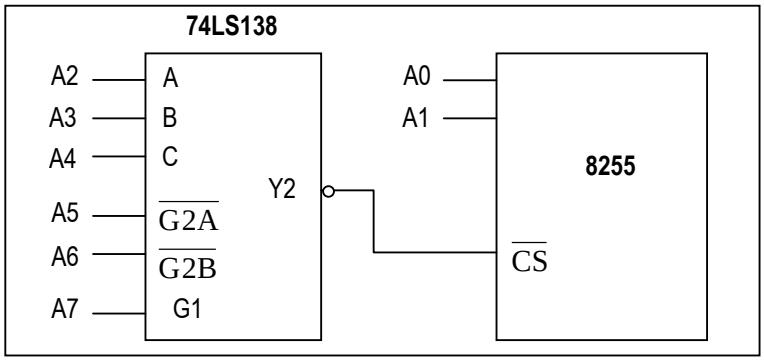
Ví dụ 15.5:

Hãy tìm địa chỉ cơ sở cho 8255 trên hình 15.7.

Lời giải:

GA	$\overline{G2B}$	$\overline{G2A}$	C	B	A			Địa chỉ
A7	A6	A5	A4	A3	A2	A1	A0	

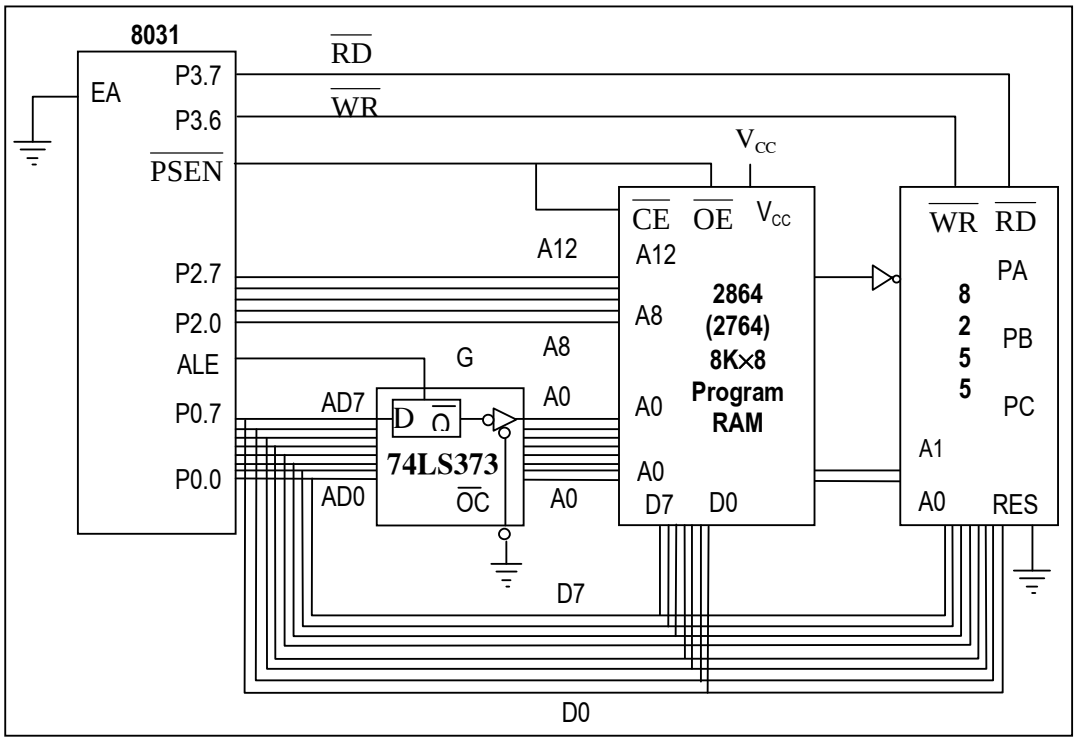
1	0	0	0	1	0	0	0	88H



Hình 15.7: Giải mã địa chỉ của 8255 sử dụng 74LS138.

15.1.6 Hệ 8031 với 8255.

Trong một hệ thống dựa trên 8031 mà bộ nhớ chương trình ROM ngoài là một sự bắt buộc tuyệt đối thì sử dụng một 8255 là rất được trào đón. Điều này là do một thực tế là trong giải trình phối ghép 8031 với bộ nhớ chương trình ROM ngoài ta bị mất hai cổng P0 và P2 và chỉ còn lại duy nhất cổng P1. Do vậy, việc nối với một 8255 là cách tốt nhất để có thêm một số cổng. Điều này được chỉ ra trên hình 15.8.

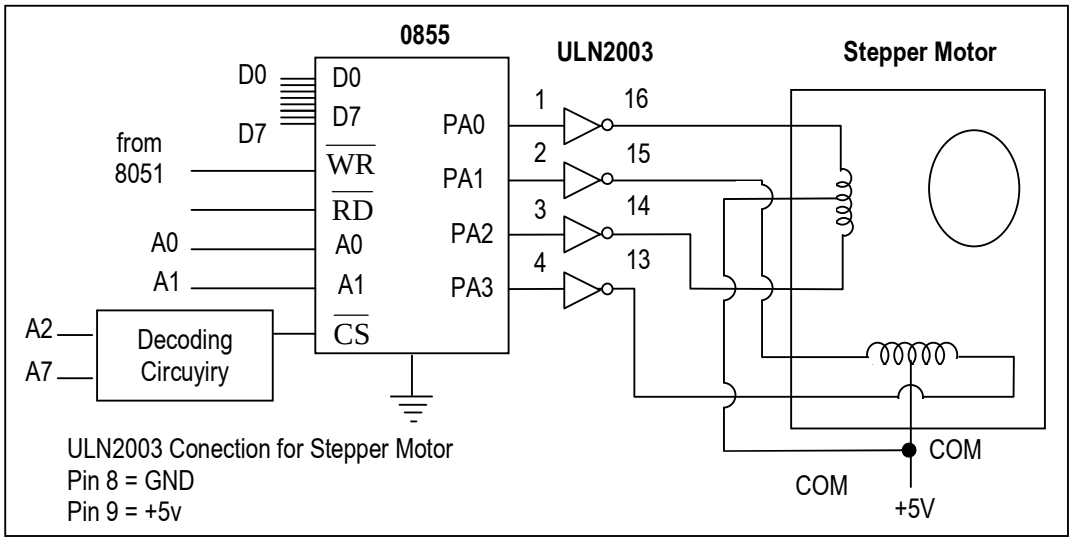


Hình 15.8: Nối 8031 tới một ROM chương trình ngoài và 8255.

15.2 Nối ghép với thế giới thực.

15.2.1 Phối ghép 8255 với động cơ bước.

Chương 13 đã nói chi tiết về phối ghép động cơ bước với 8051, ở đây ta trình bày nối ghép động cơ bước tới 8255 và lập trình (xem hình 15.9).



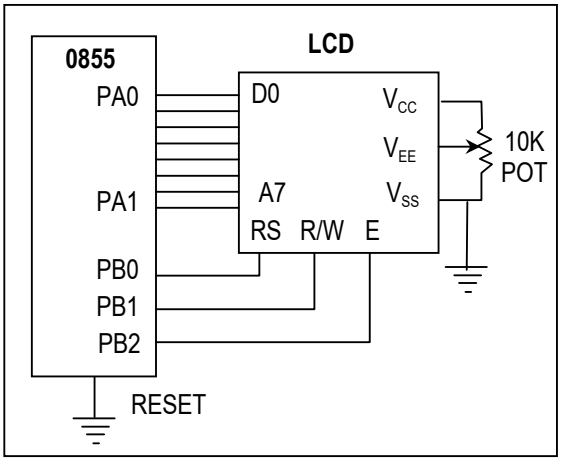
Hình 15.9: Nối ghép 8255 với một động cơ bước.

Chương trình cho sơ đồ nối ghép này như sau:

```
        MOV     A, #80H         ; Chọn từ điều khiển để PA là đầu ra
        MOV     R1, #CRPORT     ; Địa chỉ cổng thanh ghi điều khiển
        MOVX    @R1, A          ; Cấu hình cho PA đầu ra
        MOV     R1, #APORT      ; Nạp địa chỉ cổng PA
        MOV     A, #66H         ; Gán A = 66H, chuyển xung của động cơ bước
AGAIN:  MOVX    @R1, A          ; Xuất chuỗi động cơ đến PA
        RR      A               ; Quay chuỗi theo chiều kim đồng hồ
        ACALL   DELAY           ; Chờ
        SJMP    AGAIN
```

15.2.2 Phối ghép 8255 với LCD.

Chương trình 15.1 trình bày cách xuất các lệnh và dữ liệu tới một LCD được nối tới 8255 theo sơ đồ hình 15.10. Trong chương trình 15.1 ta phải đặt một độ trễ trước mỗi lần xuất thông tin bất kỳ (lệnh hoặc dữ liệu) tới LCD. Một cách tốt hơn là kiểm tra cờ bận trước khi xuất bất kỳ thứ gì tới LCD như đã nói ở chương 12. Chương trình 15.2 lặp lại chương trình 15.1 có sử dụng kiểm tra cờ bận. Để ý rằng lúc này không cần thời gian giữ chậm như ở vị trí 15.1.



Hình 5.10: Nối ghép 8255 với LCD.

Chương 15.1:

```
; Ghi các lệnh và dữ liệu tới LCD không có kiểm tra cờ bận.
; Giả sử PA của 8255 được nối tới D0 - D7 của LCD và
; IB - RS, PB1 = R/W, PB2 = E để nối các chân điều khiển LCD
                MOV     A, #80H         ; Đặt tất cả các cổng 8255 là đầu ra
                MOV     R0, #CNTPORT    ; Nẹp địa chỉ thanh ghi điều khiển
                MOVX    @R0, A          ; Xuất từ điều khiển
                MOV     A, #38H         ; Cấu hình LCD có hai dòng và ma trận 5×7
                ACALL   CMDWRT          ; Ghi lệnh ra LCD
                ACALL   DELAY           ; Chờ đến lần xuất kế tiếp (2ms)
                MOV     A, # 0EH        ; Bật con trỏ cho LCD
                ACALL   CMDWRT          ; Ghi lệnh này ra LCD
                ACALL   DELAY           ; Chờ lần xuất kế tiếp
                MOV     A, # 01H        ; Xoá LCD
                ACALL   CMDWRT          ; Ghi lệnh này ra LCD
                ACALL   DELAY           ; Dịch con trỏ sang phải
                MOV     A, # 06         ; Ghi lệnh này ra LCD
                ACALL   CMDWRT          ; Chờ lần xuất sau
                ACALL   DELAY           ; Ghi lệnh này ra LCD
                ...                     ; v.v... cho tất cả mọi lệnh LCD
                MOV     A, # 'N'        ; Hiển thị dữ liệu ra (chữ N)
                ACALL   DATAWRT         ; Gửi dữ liệu ra LCD để hiển thị
                ACALL   DELAY           ; Chờ lần xuất sau
                MOV     A, # '0'        ; Hiển thị chữ "0"
                ACALL   DATAWRT          ; Gửi ra LCD để hiển thị
                ACALL   DELAY           ; Chờ lần xuất sau
                ...                     ; v.v... cho các dữ liệu khác
; Chương trình con ghi lệnh CMDWRT ra LCD
CMDWRT:         MOV     R0, # APORT     ; Nạp địa chỉ cổng A
                MOVX    @R0, A          ; Xuất thông tin tới chân dữ liệu của LCD
                MOV     R0, # BPORT     ; Nạp địa chỉ cổng B
                MOV     A, # 00000100B  ; RS=0, R/W=1, E=1 cho xung cao xuống thấp

                MOVX    @R0, A          ; Kích hoạt các chânRS, R/W, E của LCD
                NOP                     ; Tạo độ xung cho chân E
                NOP
                MOV     A, # 00000000B  ; RS=0, R/W=1, E=1 cho xung cao xuống thấp

                MOVX    @R0, A          ; Chốt thông tin trên chân dữ liệu của LCD
                RET
; Chương trình con ghi lệnh DATAWRT ghi dữ liệu ra LCD.
CMDWRT:         MOV     R0, # APORT     ; Nạp địa chỉ cổng A
                MOVX    @R0, A          ; Xuất thông tin tới chân dữ liệu của LCD
                MOV     R0, # BPORT     ; Đặt RS=1, R/W=0, E=0 cho xung cao xuống thấp
                MOV     A, # 00000101B  ; Kích hoạt các chân RS, R/W, E
                MOVX    @R0, A          ; Tạo độ xung cho chân E
                NOP
                NOP
                MOV     A, # 00000001B  ; Đặt RS=1, R/W=0, E=0 cho xung cao xuống thấp
                MOVX    @RC, A          ; Chốt thông tin trên chân dữ liệu của LCD
                RET
```

Chương trình 15.2:

```
; Ghi các lệnh và dữ liệu tới LCD có sử dụng kiểm tra cờ bận.
; Giả sử PA của 8255 được nối tới D0 - D7 của LCD và
; PB0 = RS, PB1 = R/W, PB2 = E đối với 8255 tới các chân điều khiển LCD
```

```
                MOV     A, #80H         ; Đặt tất cả các cổng 8255 là đầu ra
                MOV     R0, #CNTPORT    ; Nạp địa chỉ thanh ghi điều khiển
                MOVX    @R0, A          ; Xuất từ điều khiển
                MOV     A, #38H         ; Chọn LCD có hai dòng và ma trận 5×7
                ACALL   NMDWRT          ; Ghi lệnh ra LCD
                MOV     A, # 0EH        ; Lệnh của LCD cho con trỏ bật
                ACALL   NMDWRT          ; Ghi lệnh ra LCD
                MOV     A, # 01H        ; Xoá LCD
                ACALL   NMDWRT          ; Ghi lệnh ra LCD
                MOV     A, # 06         ; Lệnh dịch con trỏ sang phải
                ACALL   CMDWRT          ; Ghi lệnh ra LCD
                ...                     ; v.v... cho tất cả mọi lệnh LCD
                MOV     A, # 'N'        ; Hiển thị dữ liệu ra (chữ N)
                ACALL   NCMDWRT         ; Gửi dữ liệu ra LCD để hiển thị
                MOV     A, # '0'        ; Hiển thị chữ "0"
                ACALL   NDADWRT         ; Gửi ra LCD để hiển thị
                ...                     ; v.v... cho các dữ liệu khác
; Chương trình con ghi lệnh NCMDWRT có hiển thị cờ bận
NCMDWRT:        MOV     R2,  A          ; Lưu giá trị thanh ghi A
                MOV     A, #90H         ; Đặt PA là cổng đầu vào để đọc trạng thái LCD
                MOV     R0, # CNTPORT   ; Nạp địa chỉ thanh ghi điều khiển
                MOVX    @R0, A          ; Đặt PA đầu vào, PB đầu ra
                MOV     A, # 00000110B  ; RS=0, R/W=1, E=1 đọc lệnh
                MOV     @R0, BPORT      ; Nạp địa chỉ cổng B
                MOVX    R0, A           ; RS=0, R/W=1 cho các chân RD và RS

                MOV     R0, APORT       ; Nạp địa chỉ cổng A
READY:          MOVX    @R0             ; Đọc thanh ghi lệnh
                RLC     A               ; Chuyển D7 (cờ bận) vào bit nhớ carry
                JC      READY           ; Chờ cho đến khi LCD sẵn sàng
                MOV     A, #80H         ; Đặt lại PA, PB thành đầu ra
                MOV     R0, #CNTPORT    ; Nạp địa chỉ cổng điều khiển
                MOVX    @R0, A          ; Xuất từ điều khiển tới 8255
                MOV     A, R2           ; Nhận giá trị trả lại  tới LCD
                MOV     R0, #APORT      ; Nạp địa chỉ cổng A
                MOVX    @R0, A          ; Xuất thông tin tới các chân dữ liệu của LCD
                MOV     R0, #BPORT      ; Nạp địa chỉ cổng B
                MOV     A, #00000100B   ; Đặt RS=0, R/W=0, E=1 cho xung thấp lên cao
                MOVX    @R0, A          ; Kích hoạt RS, R/W, E của LCD
                NOP                     ; Tạo độ rộng xung của chân E
                NOP
                MOV     A, #00000000B   ; Đặt RS=0, R/W=0, E=0 cho xung cao xuống thấp
                MOVX    @R0, A          ; Chốt thông tin ở chân dữ liệu LCD
                RET
; Chương trình con ghi dữ liệu mới NDATAWRT sử dụng cờ bận
NCMDWRT:        MOV     R2,  A          ; Lưu giá trị thanh ghi A
                MOV     A, #90H         ; Đặt PA là cổng đầu vào để đọc trạng thái LCD
                MOV     R0, # CNTPORT   ; Nạp địa chỉ thanh ghi điều khiển
                MOVX    @R0, A          ; Đặt PA đầu vào, PB đầu ra
                MOV     A, # 00000110B  ; RS=0, R/W=1, E=1 đọc lệnh
                MOV     @R0, BPORT      ; Nạp địa chỉ cổng B
                MOVX    R0, A           ; RS=0, R/W=1 cho các chân RD và RS

                MOV     R0, APORT       ; Nạp địa chỉ cổng A
READY:          MOVX    @R0             ; Đọc thanh ghi lệnh
                RLC     A               ; Chuyển D7 (cờ bận) vào bit nhớ carry
```

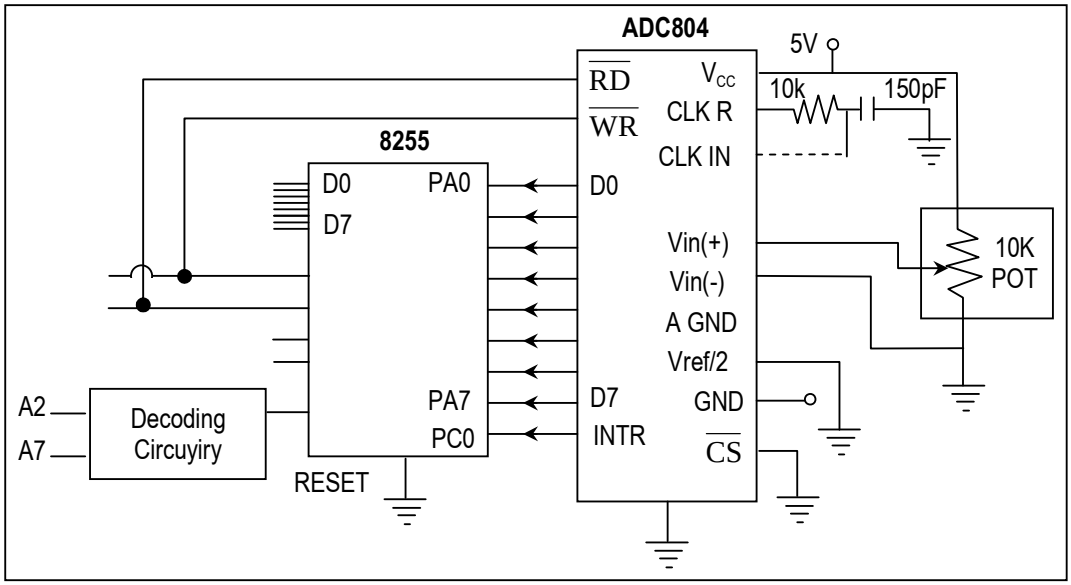
```
        JC      READY           ; Chờ cho đến khi LCD sẵn sàng
        MOV     A, #80H         ; Đặt lại PA, PB thành đầu ra
        MOV     R0, #CNTPORT    ; Nạp địa chỉ cổng điều khiển
        MOVX    @R0, A          ; Xuất từ điều khiển tới 8255
        MOV     A, R2           ; Nhận giá trị trả lại  tới LCD
        MOV     R0, #APORT      ; Nạp địa chỉ cổng A
        MOVX    @R0, A          ; Xuất thông tin tới các chân dữ liệu của LCD
        MOV     R0, #BPORT      ; Nạp địa chỉ cổng B
        MOV     A, #00000101B   ; Đặt RS=1, R/W=0, E=1 cho xung thấp lên cao
        MOVX    @R0, A          ; Kích hoạt RS, R/W, E của LCD
        NOP                     ; Tạo độ rộng xung của chân E
        NOP
        MOV     A, #00000001B   ; Đặt RS=1, R/W=0, E=0 cho xung cao xuống thấp
        MOVX    @R0, A          ; Chốt thông tin ở chân dữ liệu LCD
        RET
```

15.2.3 Nối ghép ADC tới 8255.

Các bộ ADC đã được trình bày ở chương 12. Dưới đây một chương trình chỉ một bộ ADC được nối tới 8255 theo sơ đồ cho trên hình 115.11.

```
        MOV     A, #80H         ; Từ điều khiển với PA = đầu ra và PC  = đầu vào
        MOV     R1, #CRPORT     ; Nạp địa chỉ cổng điều khiển
        MOVX    @R1, A          ; Đặt PA = đầu ra và PC = đầu vào
BACK:   MOV     R1, #CPORT      ; Nạp địa chỉ cổng C
        MOVX    A, @R1          ; Đọc địa chỉ cổng C để xem ADC đã sẵn sàng chưa
        ANL     A,, #00000001B  ; Che tất cả các bít cổng C để xem ADC đã sẵn
sàng chưa
        JNZ     BACK            ; Giữ hiển thị PC0 che EOC
                                ; Kết thúc hội thoại và bây giờ nhận dữ liệu của ADC
        MOV     R1, #APORT      ; Nạp địa chỉ PA
        MOVX    A, @R1          ; A = đầu vào dữ liệu tương tự
```

Cho đến đây ta đã được trao đổi chế độ vào/ ra đơn giản của 8255 và trình bày nhiều ví dụ về nó. Ta xét tiếp các chế độ khác.

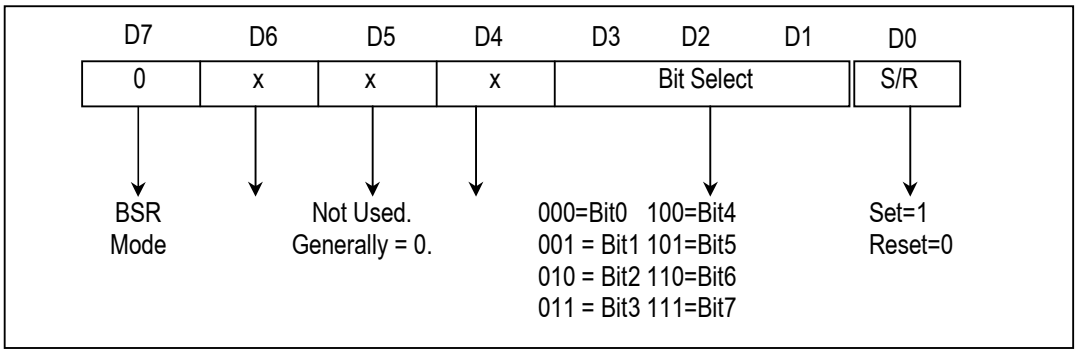


Hình 15.11: Nối ghép ADC 804 với 8255.

15.3 Các chế độ khác của 8255.

15.3.1 Chế độ thiết lập/ xoá bit BSR.

Một đặc tính duy nhất của cổng C là các bit có thể được điều khiển riêng rẽ. Chế độ BSR cho phép ta thiết lập các bit PC0 - PC7 lên cao xuống thấp như được chỉ ra trên hình 15.12. Ví dụ 15.6 và 15.7 trình bày cách sử dụng chế độ này như thế nào?

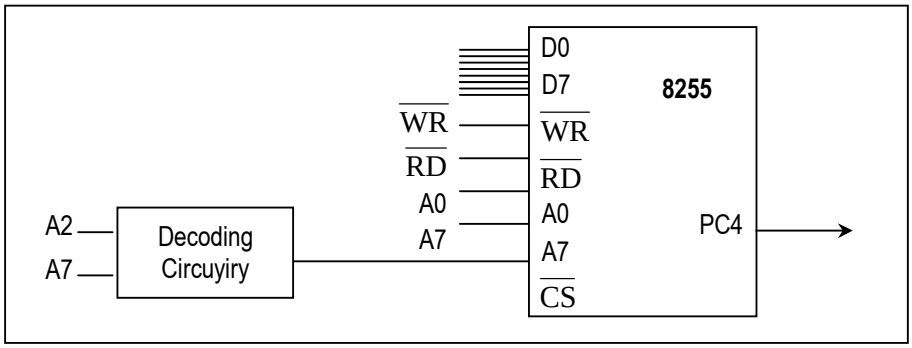


Hình 15.12: Từ điều khiển của chế độ BSR.

Ví dụ 15.6:

Hãy lập trình PCA của 8255 ở chế độ BSR thì bit D7 của từ điều khiển phải ở mức thấp. Để PC4 ở mức cao, ta cần một từ điều khiển là "0xxx1001" và ở mức thấp ta cần "0xxx1000". Các bít được đánh dấu x là ta không cần quan tâm và thường chúng được đặt về 0.

```
MOV     A, 00001001B    ; Đặt byte điều khiển cho PC4 =1
MOV     R1, #CNTPORT    ; Nạp cổng thanh ghi điều khiển
MOVX    @R1, A          ; Tạo PC4 = 1
ACALL   DELAY           ; Thời gian giữ chậm cho xung cao
MOV     A, #00001000B   ; Đặt byte điều khiển cho PC4 = 0
MOVX    @R1, A          ; Tạo PC4 = 0
ACALL   DELAY
```



Hình 15.13: Cấu hình cho ví dụ 15.6 và 15.7.

Ví dụ 15.7:

Hãy lập trình 8255 theo sơ đồ 15.13 để:

a) Đặt PC2 lên cao

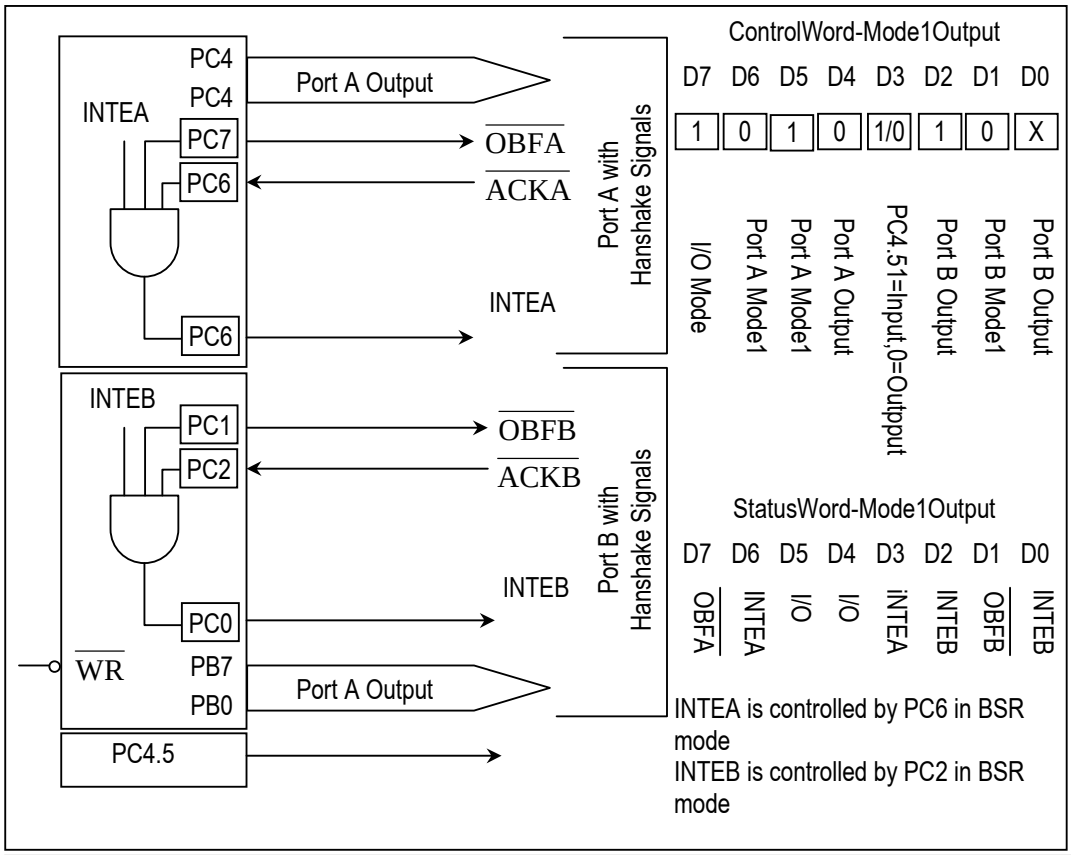
b) Sử dụng PC6 để tạo xung vuông liên tục với 66% độ đầy xung.

Lời giải:

a) MOV R0, # CNTPORT
MOV A, # 0XXX0101 ; Byte điều khiển
MOV @R0, A

b)

```
AGAIN:  MOV     A, #00001101B   ; Chọn PC6 = 1
        MOV     R0, #CNTPORT    ; Nạp địa chỉ thanh ghi điều khiển
        MOVX    @R0, A          ; Tạo PC6 = 1
        ACALL   DELAY
        ACALL   DELAY
        MOV     A, #00001100B   ; PC6 = 0
        ACALL   DELAY   ; Thời gian giữ chậm
        SJMP    AGAIN
```



Hình 15.15: Biểu đồ định thời của 8255 ở chế độ 1.

15.3.2 8255 ở chế độ 1: Vào/ ra với khả năng này bắt tay.

Một trong những đặc điểm mạnh nhất của 8255 là khả năng bắt tay với các thiết bị khác. Khả năng bắt tay là một quá trình truyền thông qua lại của hai thiết bị thông minh. Ví dụ về một thiết bị có các tín hiệu bắt tay là máy in. Dưới đây ta trình bày các tín hiệu bắt tay của 8255 với máy in.

Chế độ 1: Xuất dữ liệu ra với các tín hiệu bắt tay.

Như trình bày trên hình 15.14 thì cổng A và B có thể được sử dụng như các cổng đầu ra để gửi dữ liệu tới một thiết bị với các tín hiệu bắt tay. Các tín hiệu bắt tay cho cả hai cổng A và B được cấp bởi các bit của cổng C. Hình 15.15 biểu đồ định thời của 8255.

Dưới đây là các phần giải thích về các tín hiệu bắt tay và tính hợp lý của chúng đối với cổng A, còn cổng B thì hoàn toàn tương tự.

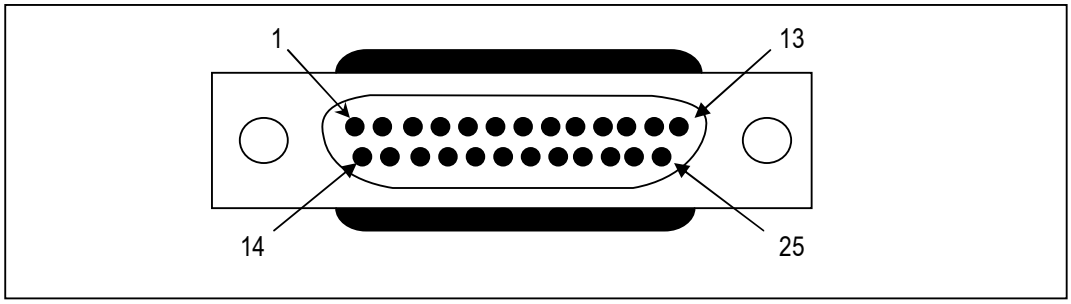
Tín hiệu $\overline{\text{OBFa}}$***:*** Đây là tín hiệu bộ đệm đầu ra đầy của cổng A được tích cực mức thấp đi ra từ chân PC7 để báo rằng CPU đã ghi 1 byte dữ liệu tới cổng A. Tín hiệu này phải được nối tới chân STROBE của thiết bị thu nhận dữ liệu (chẳng hạn như máy in) để báo rằng nó bây giờ đã có thể đọc một byte dữ liệu từ chốt cổng.

Tín hiệu $\overline{\text{ACKa}}$***:*** Đây là tín hiệu chấp nhận do cổng A có mức tích cực mức thấp được nhân tại chân PC6 của 8255. Qua tín hiệu $\overline{\text{ACKa}}$ thì 8255 biết rằng tín hiệu tại cổng A đã được thiết bị thu nhận lấy đi. Khi thiết bị nhận lấy dữ liệu đi từ cổng A nó báo 8255 qua tín hiệu $\overline{\text{ACKa}}$. Lúc này 8255 bật $\overline{\text{OBFa}}$ lên cao để báo rằng dữ liệu tại cổng A bây giờ là dữ liệu cũ và khi CPU đã ghi một byte dữ liệu mới tới cổng A thì $\overline{\text{OBFa}}$ lại xuống thấp v.v...

Tín hiệu INTRa: Đây là tín hiệu yêu cầu ngắt của cổng A có mức tích cực cao đi ra từ chân PC3 của 8255. Tín hiệu $\overline{\text{ACK}}$ là tín hiệu có độ dài hạn chế. Khi nó xuống thấp (tích cực) thì nó làm cho $\overline{\text{OBFa}}$ không tích cực, nó ở mức thấp một thời gian ngắn và sau đó trở nên cao (không tích cực). Sườn lên của $\overline{\text{ACK}}$ kích hoạt INTRa lên cao. Tín hiệu cao này trên chân INTRa có thể được dùng để gây chú ý của CPU. CPU được thông báo qua tín hiệu INTRa rằng máy in đã nhận byte cuối cùng và nó sẵn sàng để nhận byte dữ liệu khác. INTRa ngắt CPU ngừng mọi thứ đang làm và ép nó gửi byte kế tiếp tới cổng A để in. Điều quan trọng là chú ý rằng INTRa được bật lên 1 chỉ khi nếu INTRa, $\overline{\text{OBFa}}$ và $\overline{\text{ACKa}}$ đều ở mức cao. Nó được xoá về không khi CPU ghi một byte tới cổng A.

Tín hiệu INTEa: Đây là tín hiệu cho phép ngắt cổng A 8255 có thể cấm INTRa để ngăn nó không được ngắt CPU. Đây là chức năng của tín hiệu INTEa. Nó là một mạch lật Flip - Flop bên trong thiết kế để che (cấm) INTRa. Tín hiệu INTRa có thể được bật lên hoặc bị xoá qua cổng C trong chế độ BSR vì INTEa là Flip - Plop được điều khiển bởi PC6.

Từ trạng thái: 8255 cho phép hiển thị trạng thái của các tín hiệu INTR, OBF và INTE cho cả hai cổng A và B. Điều này được thực hiện bằng cách đọc cổng C vào thanh ghi tổng và kiểm tra các bit. Đặc điểm này cho phép thực thi thăm dò thay cho ngắt phần cứng.



Hình 15.16: Đầu cắm DB-25.

(hình 15.17 mờ quá không vẽ được)

Hình 15.17: Đầu cáp của máy in Centronics.

Bảng 15.2: Các chân tín hiệu của máy in Centronics.

Chân số	Mô tả	Chân số	Mô tả
1	STROBE	11	Bận (busy)
2	Dữ liệu D0	12	Hết giấy (out of paper)
3	Dữ liệu D1	13	Chọn (select)
4	Dữ liệu D2	14	Tự nạp ($\overline{\text{Autofeed}}$)
5	Dữ liệu D3	15	Lỗi ($\overline{\text{Error}}$)
6	Dữ liệu D4	16	Khởi tạo máy in
7	Dữ liệu D5	17	Chọn đầu vào ($\overline{\text{Select input}}$)
8	Dữ liệu D6	18-25	Đất (ground)
9	Dữ liệu D7		
10	$\overline{\text{ACK}}$ (chấp nhận)		

Các bước truyền thông có bắt tay giữa máy in và 8255.

Một byte dữ liệu được gửi đến bus dữ liệu máy in.

Máy in được báo có 1 byte dữ liệu cần được in bằng cách kích hoạt tín hiệu đầu vào STROBE của nó.

Khi máy nhận được dữ liệu nó báo bên gửi bằng cách kích hoạt tín hiệu đầu ra được gọi là $\overline{\text{ACK}}$ (chấp nhận).

Tín hiệu $\overline{\text{ACK}}$ khởi tạo quá trình cấp một byte khác đến máy in.

Như ta đã thấy từ các bước trên thì chỉ khi một byte dữ liệu tới máy in là không đủ. Máy in phải được thông báo về sự hiện diện của dữ liệu. Khi dữ liệu được gửi thì máy in có thể bận hoặc hết giấy, do vậy máy in phải được báo cho bên gửi khi nào nó nhận và lấy được dữ liệu của nó. Hình 15.16 và 15.17 trình các ổ cắm DB25 và đầu cáp của máy in Centronics tương ứng.

Kỹ thuật Vi xử lý

Điện tử-Viễn thông

Đại học Bách khoa Đà Nẵng

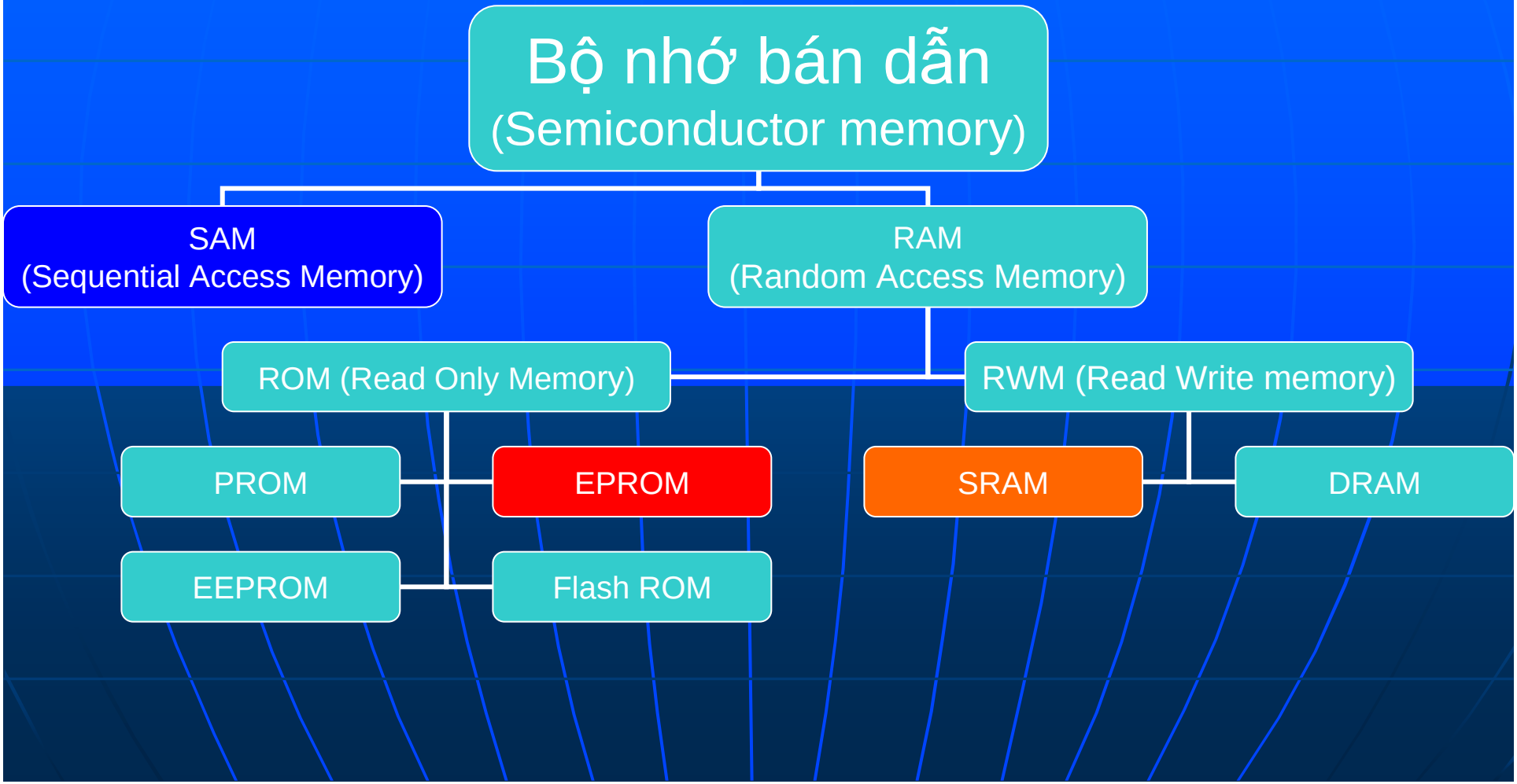
Chương 4

4.1 Phân loại bộ nhớ bán dẫn
4.2 Hoạt động của các chip EPROM
4.3 Hoạt động của các chip SRAM
4.4 Bus hệ thống của hệ vi xử lý 8088
4.5 Bài toán thiết kế bộ nhớ

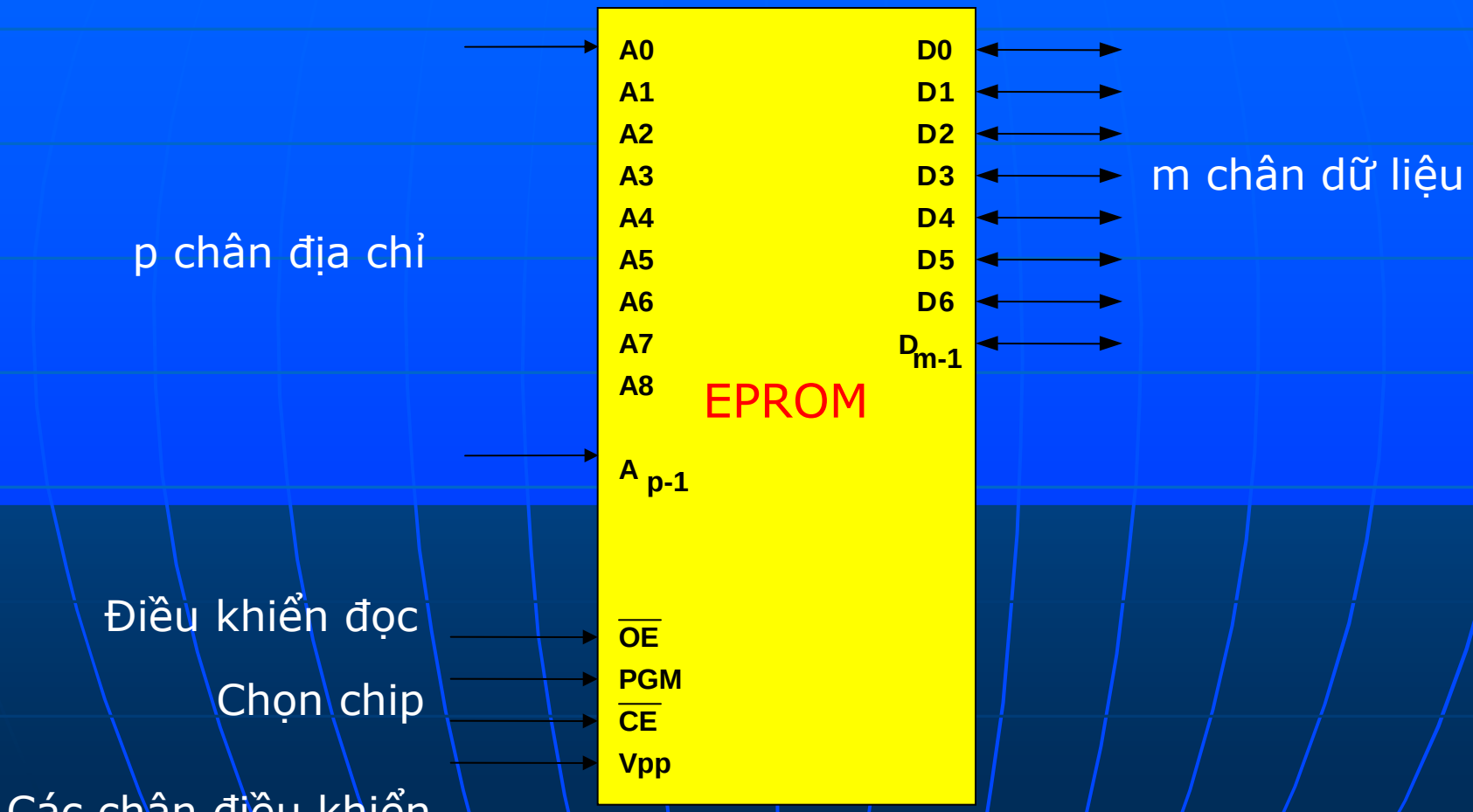
Mục tiêu và biện pháp thiết kế

- Ghép nối các chip nhớ EPROM và SRAM với Bus hệ thống sao cho không xảy ra xung đột:
  Các chip nhớ bị cấm khi vi xử lý truy cập các cổng I/O
  Chỉ có một chip nhớ hoạt động khi vi xử lý truy cập bộ nhớ
- Thực hiện một mạch giải mã địa chỉ bộ nhớ dùng các chip giải mã hoặc các cổng logic hoặc kết hợp cả hai

4.1 Phân loại bộ nhớ bán dẫn



4.2 Các chip EPROM



Dung lượng của 1 chip nhớ

- Một chip nhớ được xem như một mảng gồm n ô nhớ. Mỗi ô nhớ lưu trữ được m-bit dữ liệu
- Dung lượng của chip thường được biểu diễn: nxm
  Ví dụ: Một chip có dung lượng 2Kx8 nghĩa là chip đó có 2048 ô nhớ và mỗi ô nhớ có thể lưu trữ được 1 byte dữ liệu
- m chính là số chân dữ liệu của chip
- $\log_2(n) = p$ là số chân địa chỉ của chip

Hoạt động ghi dữ liệu vào EPROM

- Việc ghi dữ liệu vào EPROM được gọi là lập trình cho EPROM
- Được thực hiện bằng thiết bị chuyên dụng gọi là Bộ nạp EPROM
- Chân Vpp được cấp điện áp tương ứng với từng loại chip gọi là điện áp lập trình
- Dữ liệu tại các chân dữ liệu sẽ được ghi vào một ô nhớ xác định nhờ các tín hiệu đưa vào ở các chân địa chỉ và một xung (thường gọi là xung lập trình) đưa vào chân PGM

Hoạt động đọc dữ liệu từ một chip EPROM

Để đọc dữ liệu từ 1 ô nhớ nào đó của 1 chip EPROM nào đó, Bộ vi xử lý cần phải:

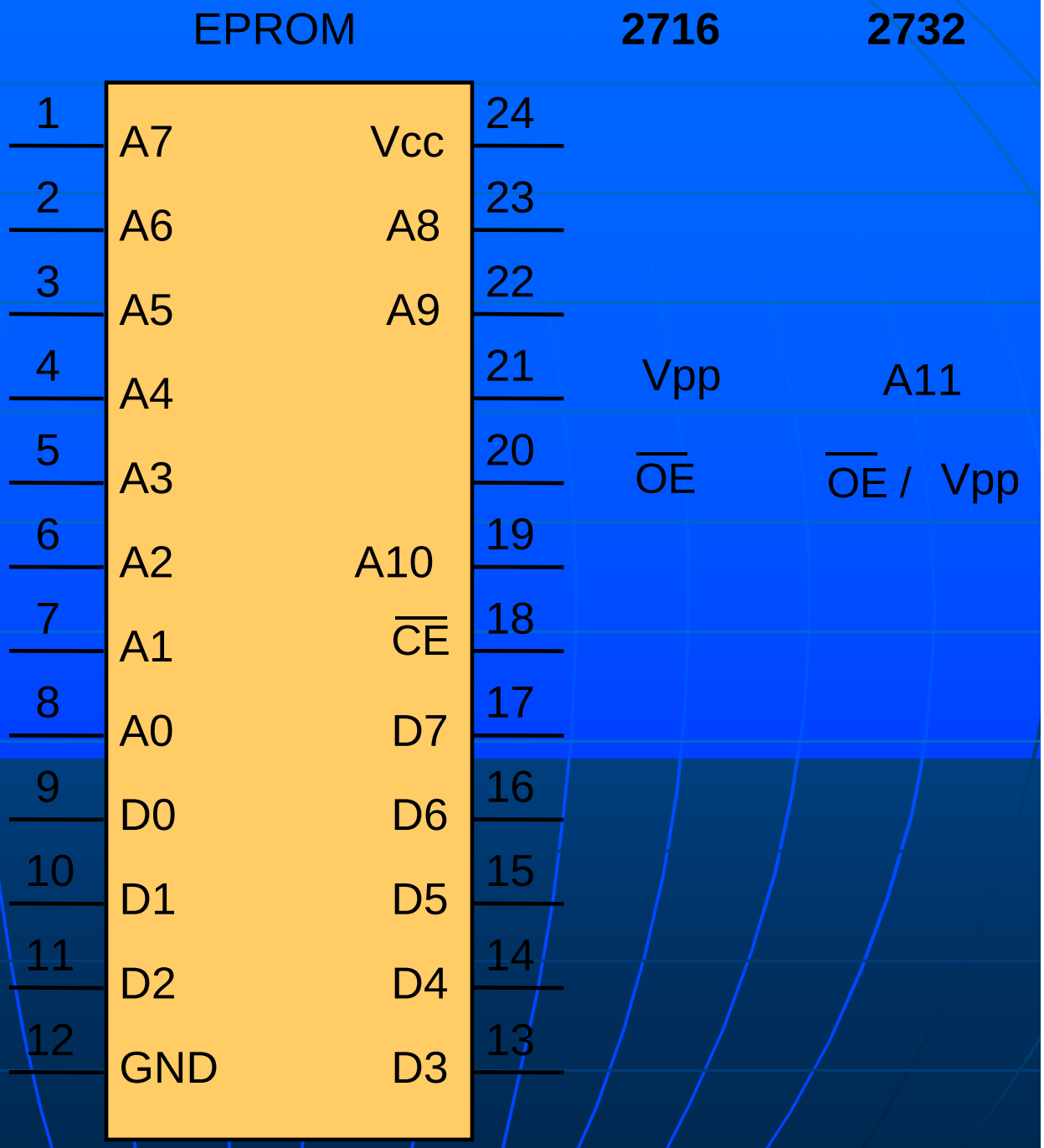
- Chọn chip đó: 0 -----> CE
- Áp các tín hiệu địa chỉ của ô nhớ cần đọc vào các chân địa chỉ $A_{p-1} - A_0$
- Đọc: 0 ------ > OE
- Kết quả là m bit dữ liệu cần đọc xuất hiện ở các chân dữ liệu $D_{m-1} - D_0$

Họ EPROM thông dụng 27x

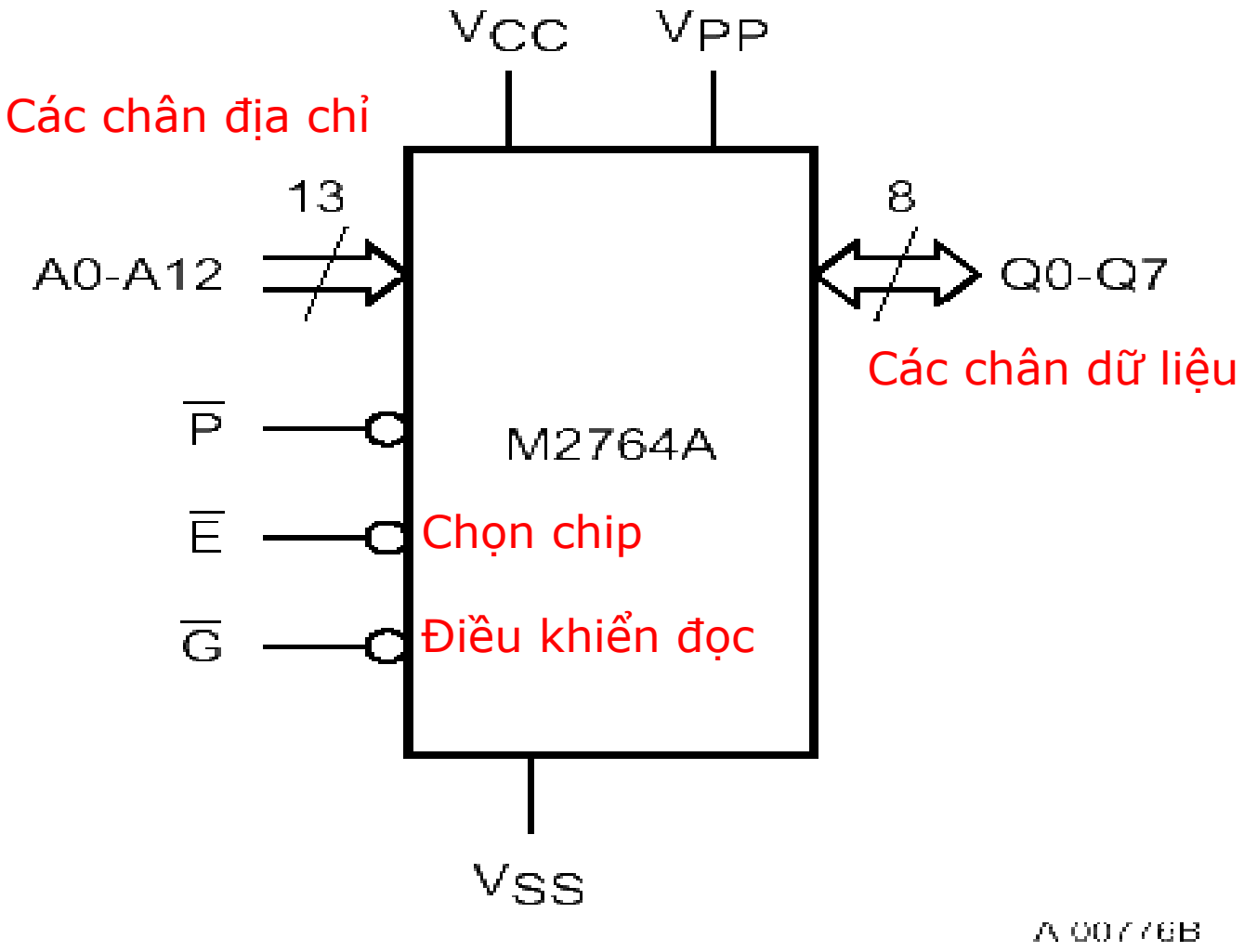
Số hiệu của chip	Dung lượng
2716	2Kx8
2732	4Kx8
2764	8Kx8
27128	16Kx8
27256	32Kx8
27512	64Kx8

Bảng 4.1 Họ EPROM 27x

- Sơ đồ chân của 2716 và 2732



EPROM 2764


VCC
VPP
Các chân địa chỉ
13
8
A0-A12
Q0-Q7
Các chân dữ liệu
P
M2764A
E
Chọn chip
G
Điều khiển đọc
VSS

EPROM 2764

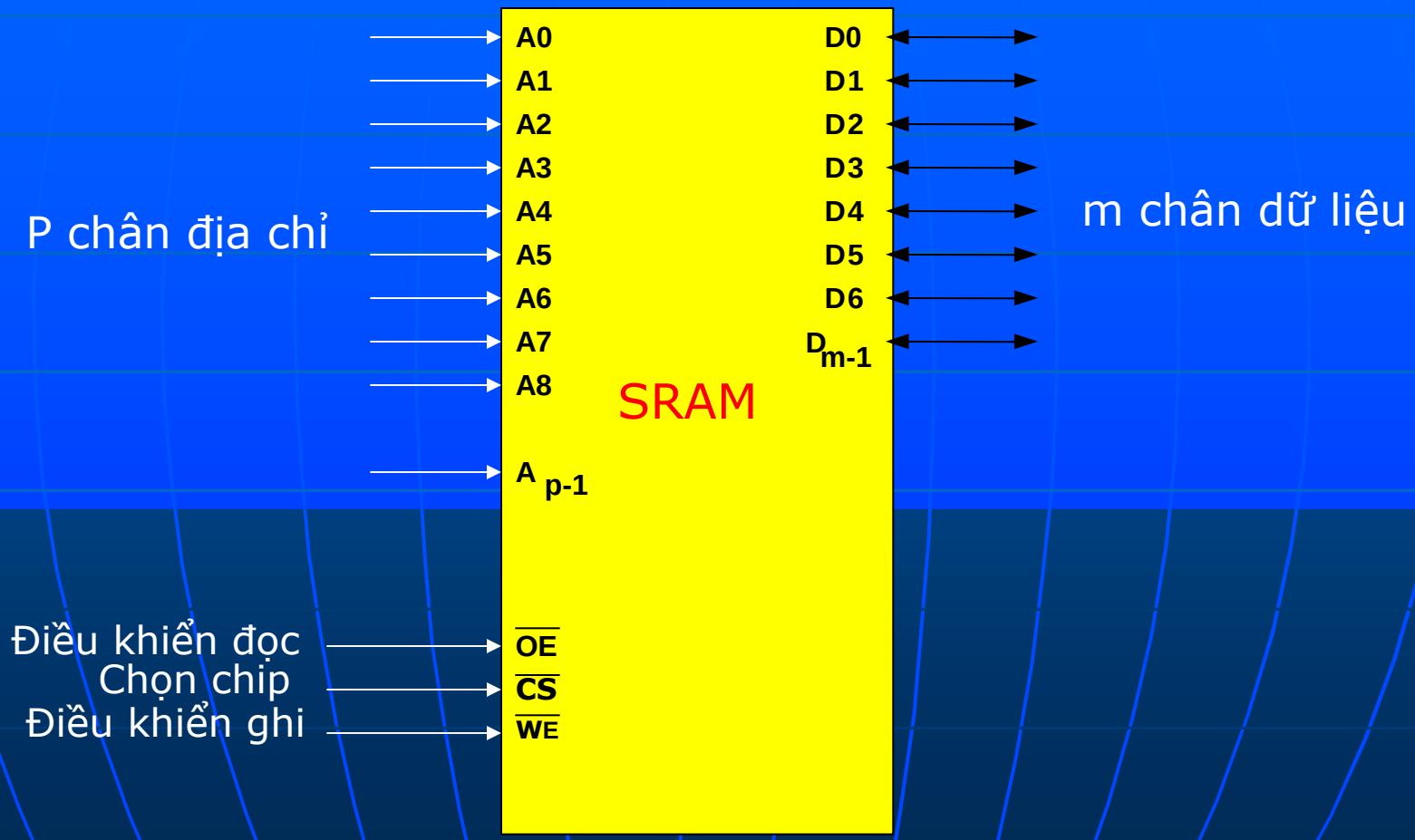
Mode	$\overline{E}$	$\overline{G}$	$\overline{P}$	A9	V_{PP}	Q0 - Q7
Read	V_{IL}	$V_{\_}$	V_{IH}	X	V_{CC}	Data Out
Output Disable	V_{IL}	V_H	V_{IH}	X	V_{CC}	Hi-Z
Program	V_{IL}	V_H	$V_{I\_}$ Pulse	X	V_{PP}	Data In
Verify	V_{IL}	$V_{\_}$	V_{IH}	X	V_{PP}	Data Out
Program Inhibit	$V_{I\sim}$	X	X	X	V_{PP}	Hi-Z
Standby	$V_{I\sim}$	X	X	X	V_{CC}	Hi-Z
Electronic Signature	V_{IL}	$V_{\_}$	V_{IH}	V_{ID}	V_{CC}	Codes Out

Note: X = V_H or V_{IL}, V_{ID} = 12V ± 0.5%.

Lập trình cho 2764

- Trước hết cần phải xoá
  - Xoá một chip tức là làm cho tất cả các bit = 1
- Xoá một chip EPROM bằng tia cực tím
- Lập trình bằng cách:
  - VPP mắc ở mức 12.5V
  - E và P đều ở mức thấp TTL
- Các bit dữ liệu đưa vào các chân dữ liệu
- Các bit địa chỉ đưa vào các chân địa chỉ

4.3 Các chip SRAM



Đọc dữ liệu từ một chip SRAM

Để đọc dữ liệu từ 1 ô nhớ nào đó của 1 chip SRAM nào đó, vi xử lý cần phải:

- Chọn chip đó: 0 ----------> CS
- Áp các tín hiệu địa chỉ vào A_{p-1} – A_0
- Đọc: 0 ---------- > OE

Kết quả là m bit dữ liệu cần đọc xuất hiện ở các chân dữ liệu D_{m-1} – D_0

Ghi dữ liệu vào một chip SRAM

Để ghi m bit dữ liệu vào 1 ô nhớ nào đó của 1 chip SRAM nào đó, vi xử lý cần phải:

- Chọn chip đó: 0 --------> CS
- Áp các tín hiệu địa chỉ vào $A_{p-1} - A_0$
- Áp m bit dữ liệu cần ghi vào các chân dữ liệu $D_{m-1} - D_0$
- Ghi: 0 --------- > WE

Kết quả là các bit dữ liệu ở các chân dữ liệu sẽ được ghi vào ô nhớ đã chọn

4.4 Bus hệ thống của 8088

- Bus địa chỉ 20-bit: gồm các đường địa chỉ được ký hiệu từ A_{19} đến A_0
- Bus dữ liệu 8-bit: gồm các đường dữ liệu được ký hiệu từ D_7 đến D_0
- Bus điều khiển gồm các đường điều khiển riêng lẽ phục vụ cho hoạt động truy cập bộ nhớ và các cổng I/O, mỗi đường thường được ký hiệu bằng tên của tín hiệu điều khiển
- Bus hệ thống không nối trực tiếp với các chân của 8088: thông qua các mạch đệm, chốt.

80x86 Microprocessors

Product	***8008***	***8080***	***8085***	***8086***	***8088***	***80286***	***80386***	***80486***	***Pent.***	***Pent. Pro***
Year Introduced	1972	1974	1976	1978	1979	1982	1985	1989	1992	1995
Technology	PMOS	NMOS	NMOS	NMOS	NMOS	NMOS	CMOS	CMOS	BICMOS	BICMOS
Clock Rate	0.5-0.8	2-3	3-8	5-10	5-8	10-16?	16-40	66	60-66+	150
Number of Pins	18	40	40	40	40		132	168	273	387
Number of transistors	3000	4500	6500	29K	29K	130K	275K	1.2M	3M	5.5M
Number of instructions	66	111	113	133	133					
Physical Memory	16K	64K	64K	1M	1M	16M	16M4GB	4GB	4GB	64G
Virtual Memory	none	none	none	none	none	1G	64T	64T	64T	64T
Internal Data Bus	8	8	8	16	16	16	32	32	64	32
External Data Bus	8	8	8	16	8	16	16,32	32	64	64
Address Bus	8	16	16	20	20	24	24,32	32	32	36
Data Types	8	8	8	8,16	8,16	8,16	8,16,32	8,16,32	8,16,32	8,16,32

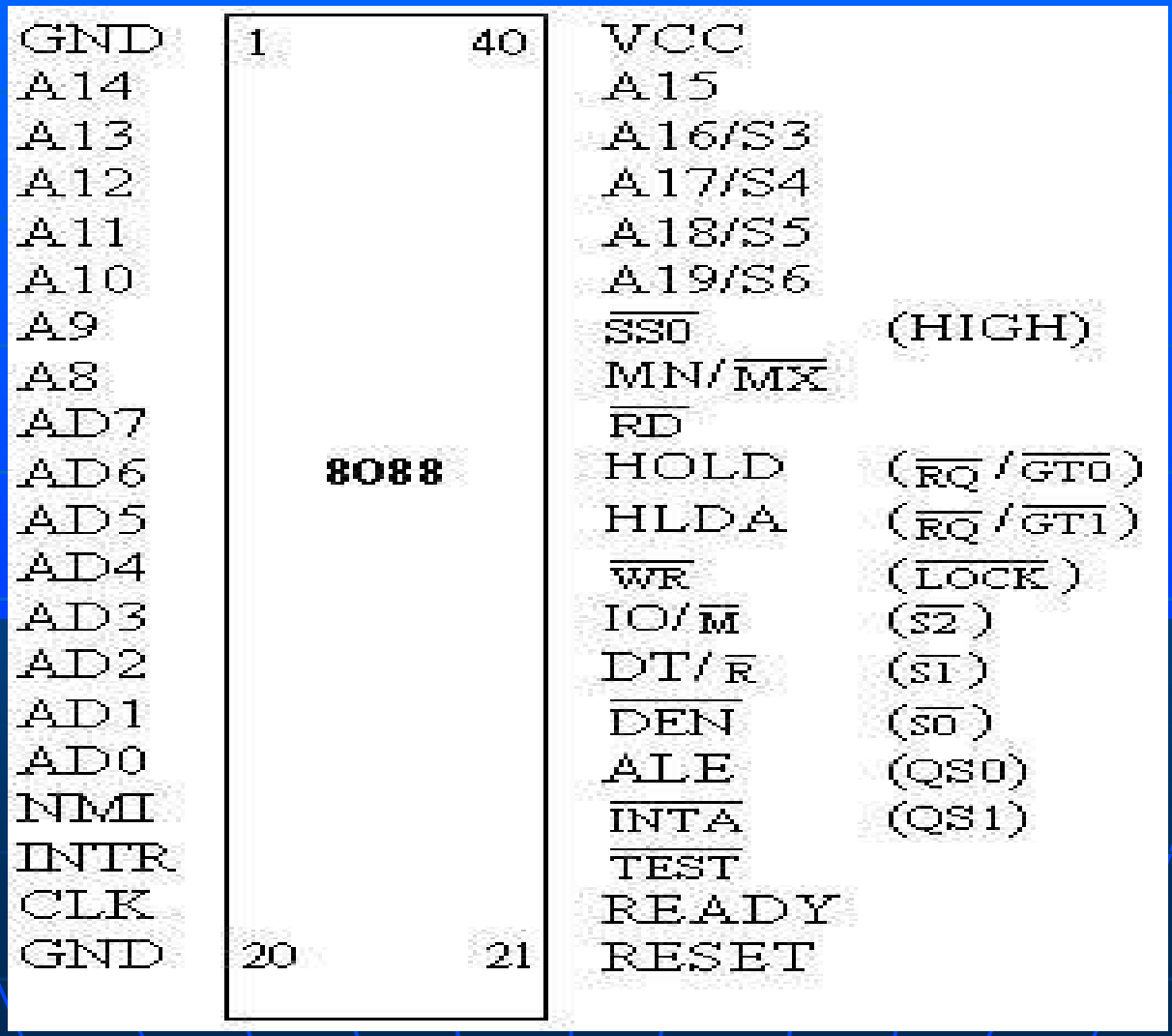
8088/8086 Microprocessor

- DIP 40 pin
- Data bus
  - Bus dữ liệu trong :16 bit
  - Bus dữ liệu ngoài của 8088: 8 bit dùng AD0-AD7
  - Bus dữ liệu ngoài của 8086:16 bit dùng AD0-AD15
  - ALE (Address Latch Enable)

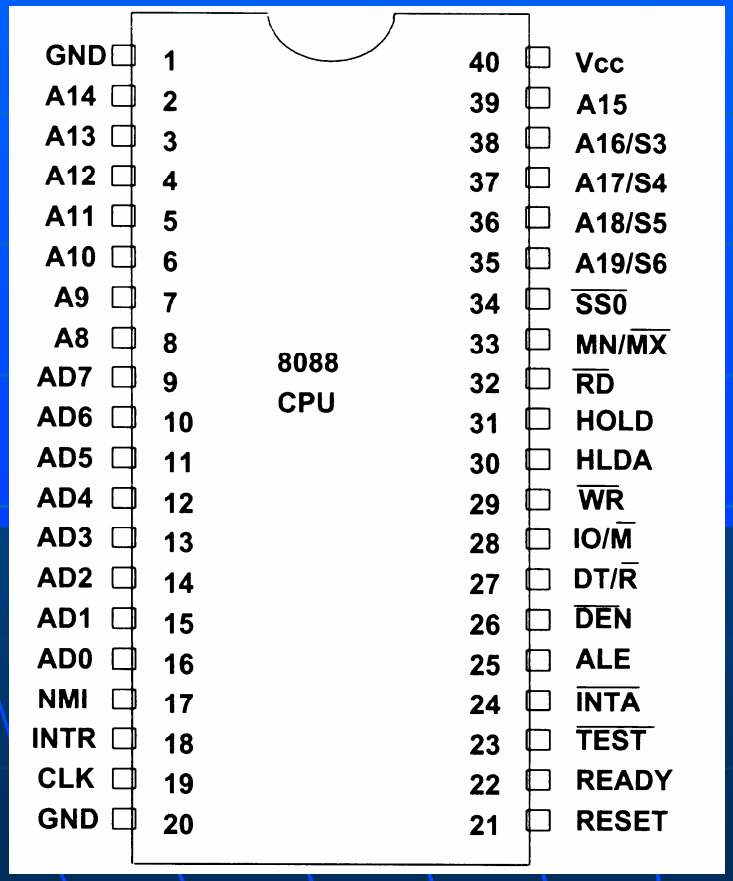
8088/8086 Microprocessor

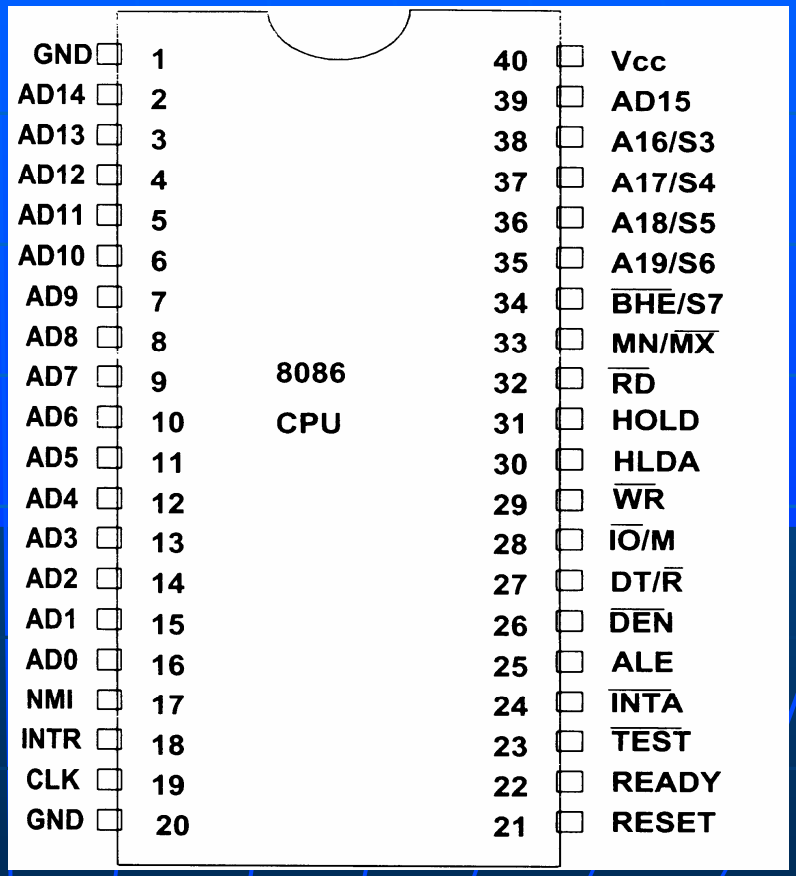
- Bus địa chỉ
  - ALE = 1
  - Sử dụng 74LS373 để tách và chốt địa chỉ
    - Đầu vào: AD0-AD7 (8088) hoặc AD0-AD15 (8086) và ALE
    - Đầu ra: A0-A7 (8088) hoặc A0-A15 (8086)

Sơ đồ chân của 8088



Sơ đồ chân 8088/8086 (Min Mode)

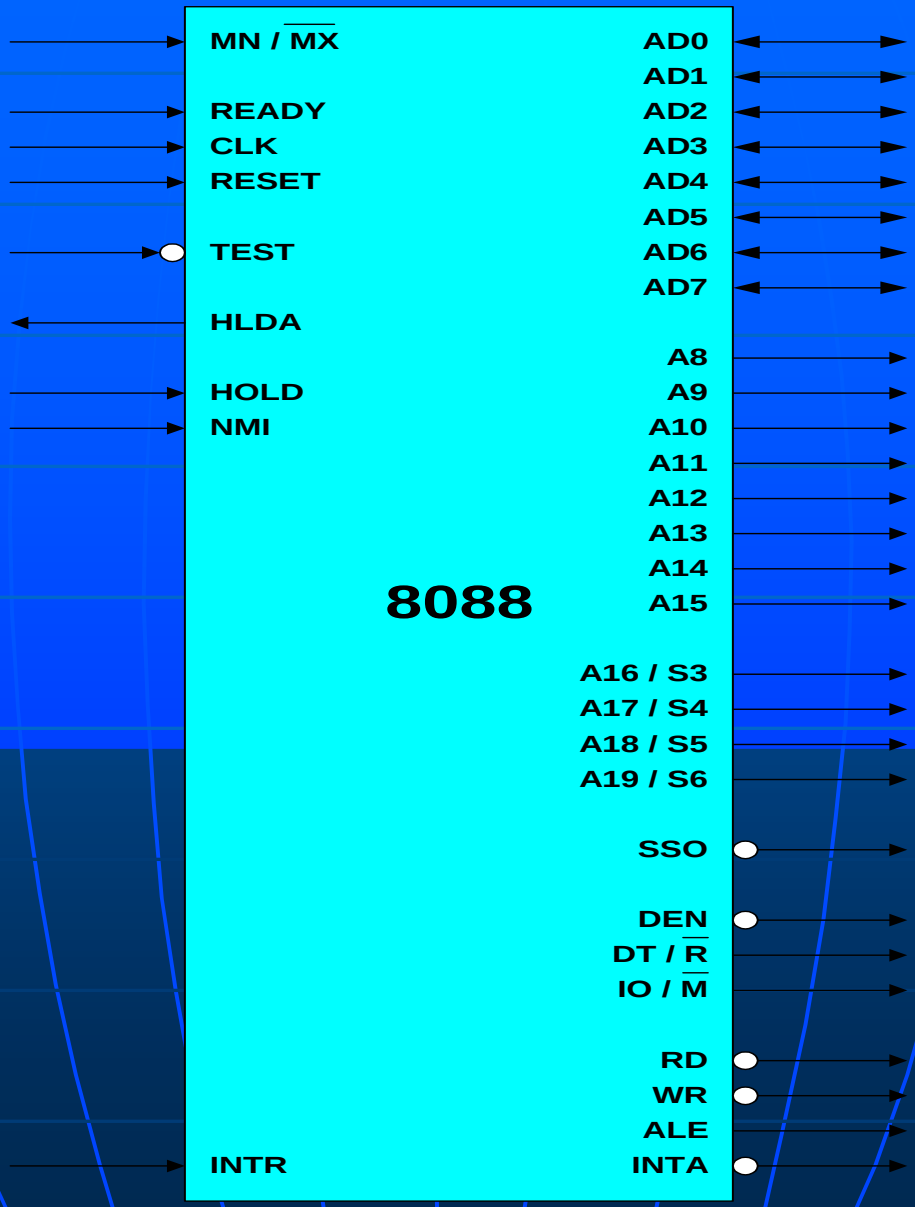




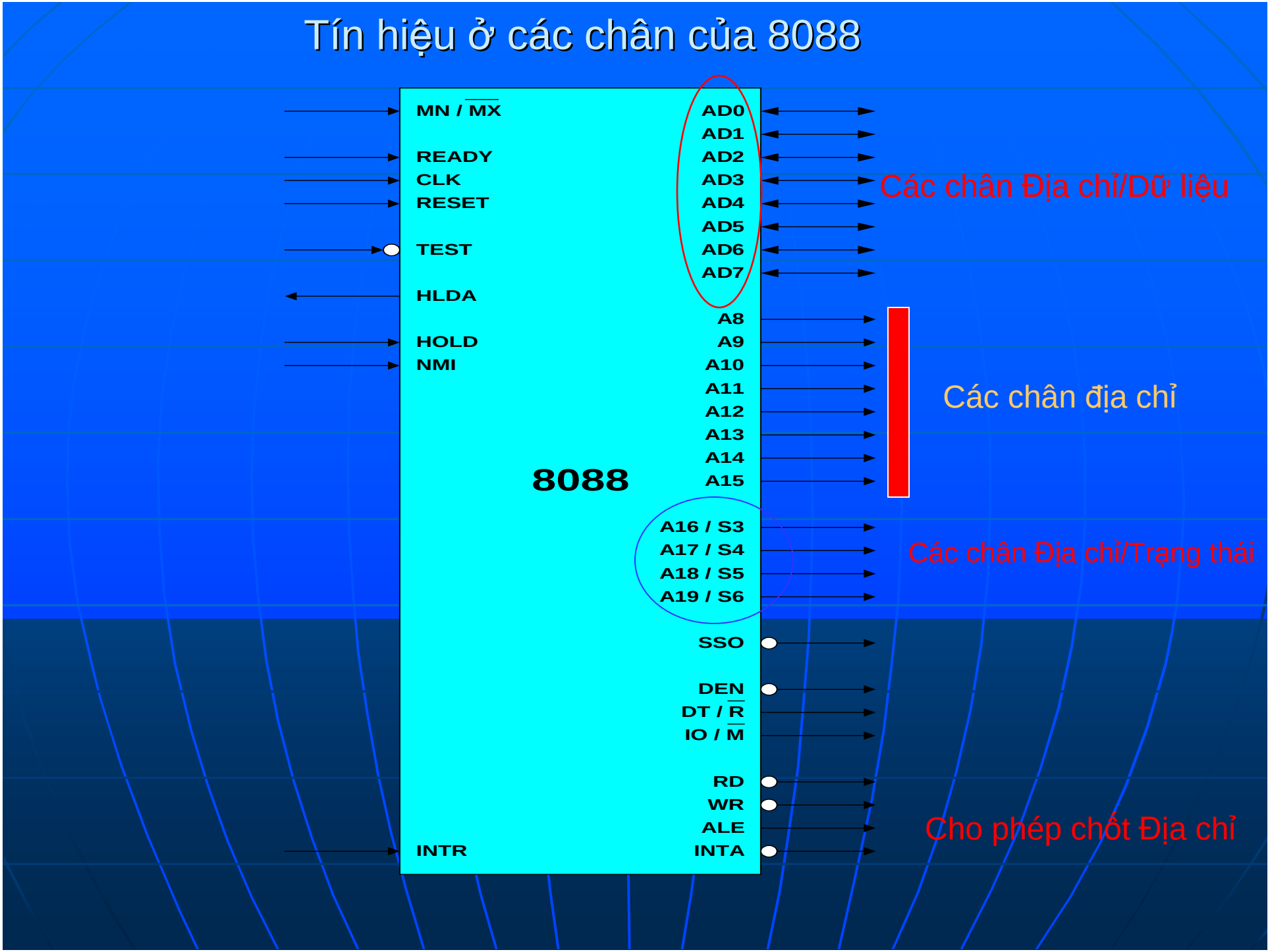
Minimum/Maximum Mode

- Ảnh hưởng đến các chân 24-31
- Minimum Mode
  - Các chân 24-31 là các tín hiệu điều khiển I/O và bộ nhớ
  - Các tín hiệu điều khiển đều từ 8088/8086
  - Tương tự với 8085A
- Maximum Mode
  - Một số tín hiệu điều khiển được tạo ra từ ngoài
  - Một số chân có thêm chức năng mới
  - Khi có dùng bộ đồng xử lý toán 8087

Sơ đồ chân của 8088



Tín hiệu ở các chân của 8088



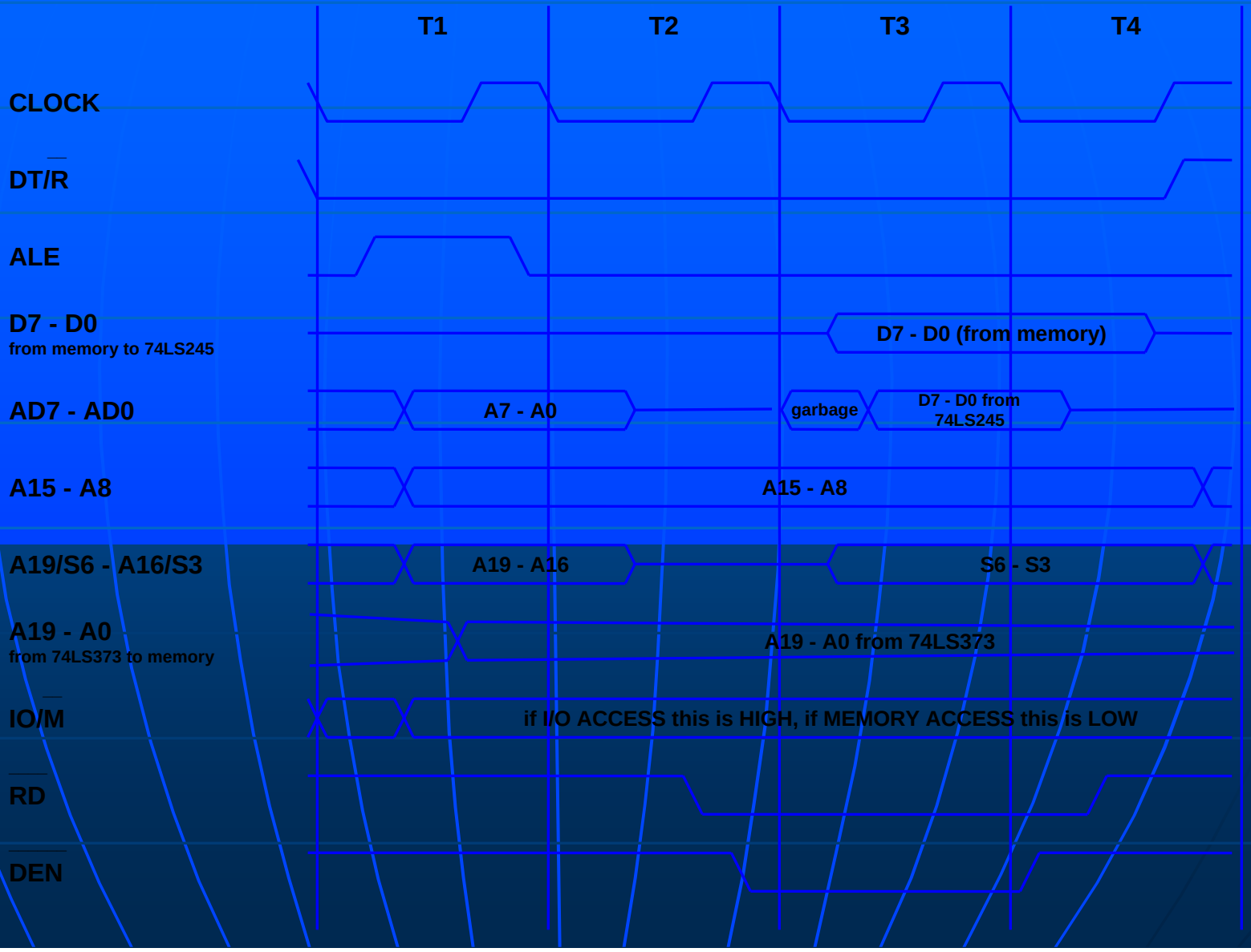
Các chân Địa chỉ/Dữ liệu

- Các chân AD_7 đến AD_0
- Kỹ thuật Multiplexing: Tín hiệu ở các chân này lúc này là tín hiệu địa chỉ, lúc khác là tín hiệu dữ liệu phụ thuộc vào tín hiệu điều khiển ALE (Address Latch Enable):
- ALE = 1: AD_7 đến AD_0 = A_7 đến A_0
- ALE = 0: AD_7 đến AD_0 = D_7 đến D_0

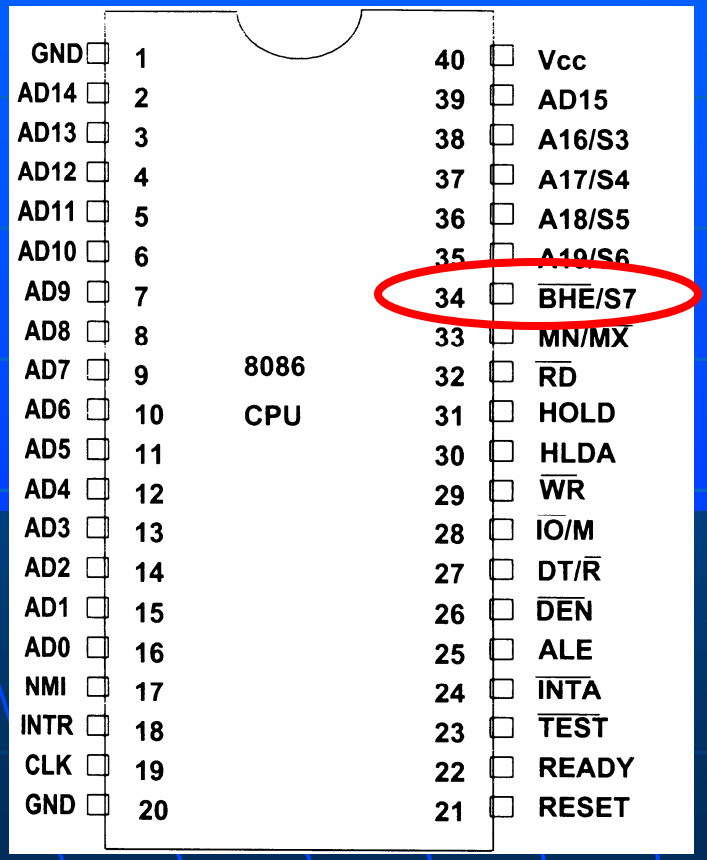
Các chân Địa chỉ và Các chân Địa chỉ/Trạng thái

- Các chân địa chỉ: A_{15} đến A_8
- Tín hiệu ở các chân này luôn là tín hiệu địa chỉ
- Các chân địa chỉ/trạng thái: A_{19}/S_6 đến A_{16}/S_3:
- ALE = 1: A_{19} đến A_{16}
- ALE = 0: S_6 đến S_3

Processor Timing Diagram of 8088 (Minimum Mode) for Memory or I/O Read (with 74245)

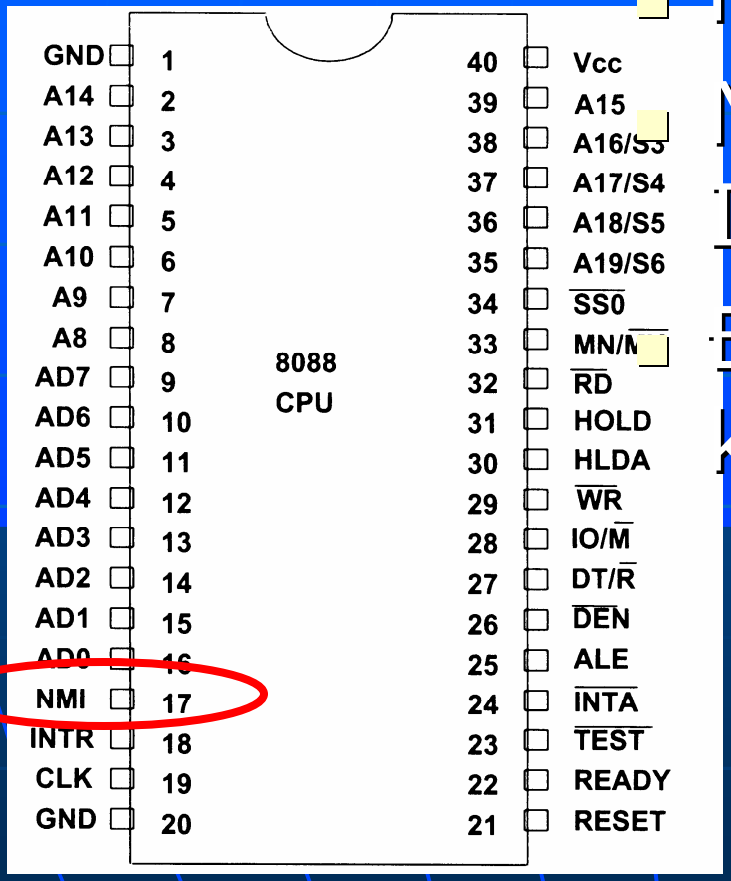


Mô tả chân



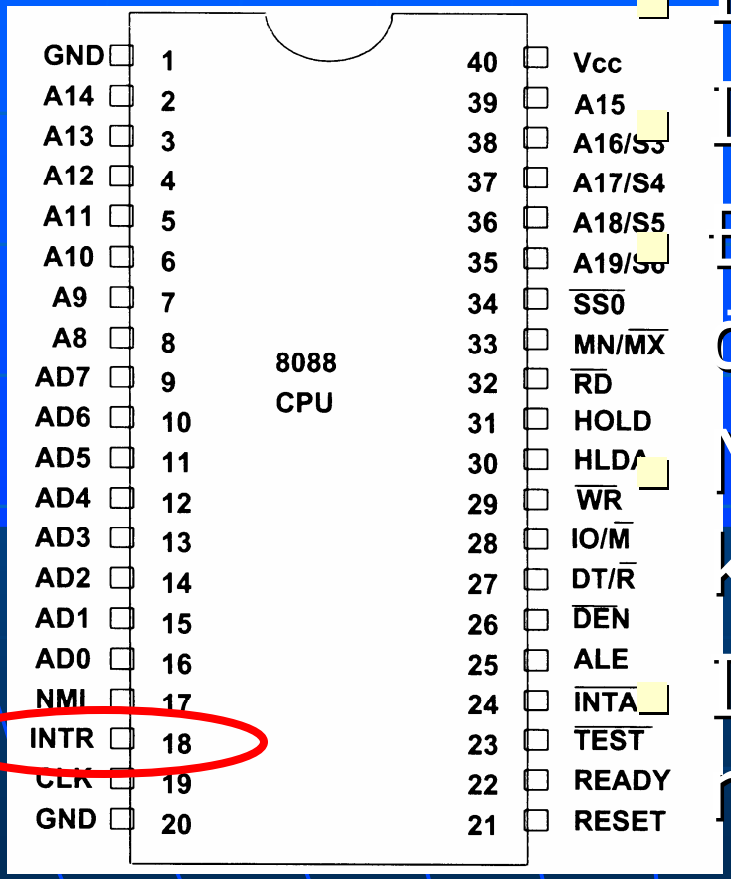
- BHE
  Bus High Enable
  Phân biệt byte thấp và byte cao của một từ (chỉ với 8086)

Mô tả chân



- NMI
- Non Maskable Interrupt
- Đầu vào ngắt không che được

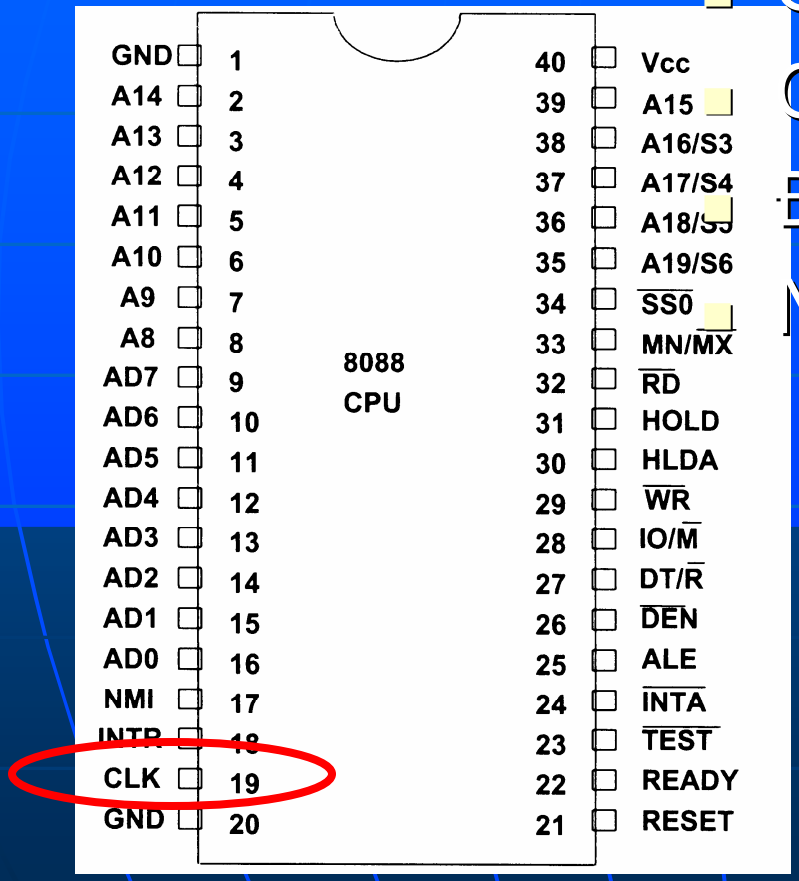
Mô tả chân



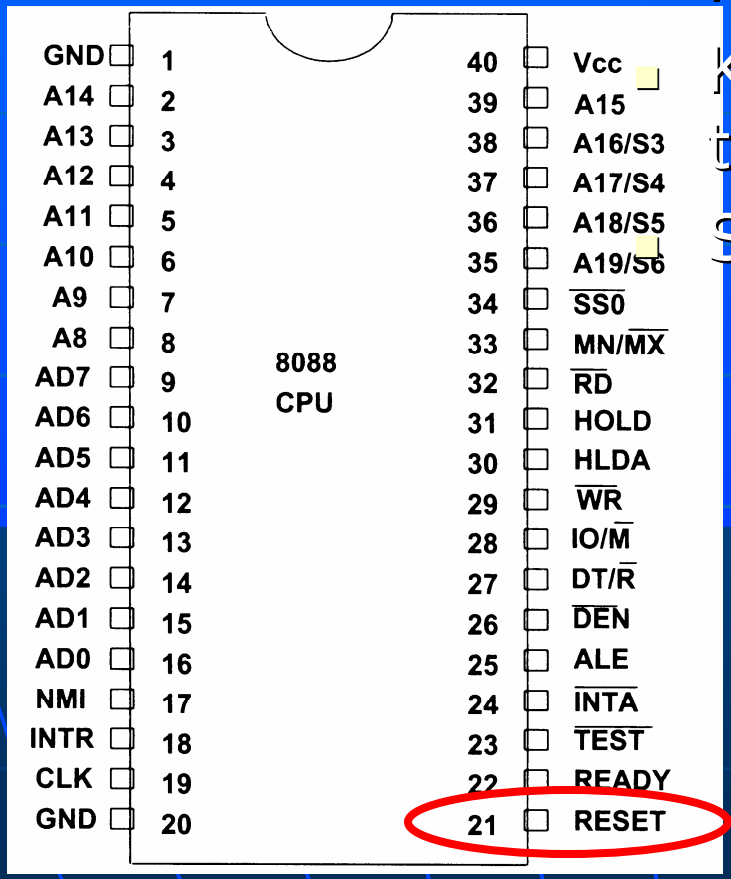
- INTR
  - Interrupt Request
  - Đầu vào ngắt che được
  - Nối với chip điều khiển ngắt 8259
- INTA: chấp nhận ngắt

Mô tả chân

- CLK
  - Clock
  - Đầu vào đồng hồ
  - Nối với chip 8284



Mô tả chân



- RESET

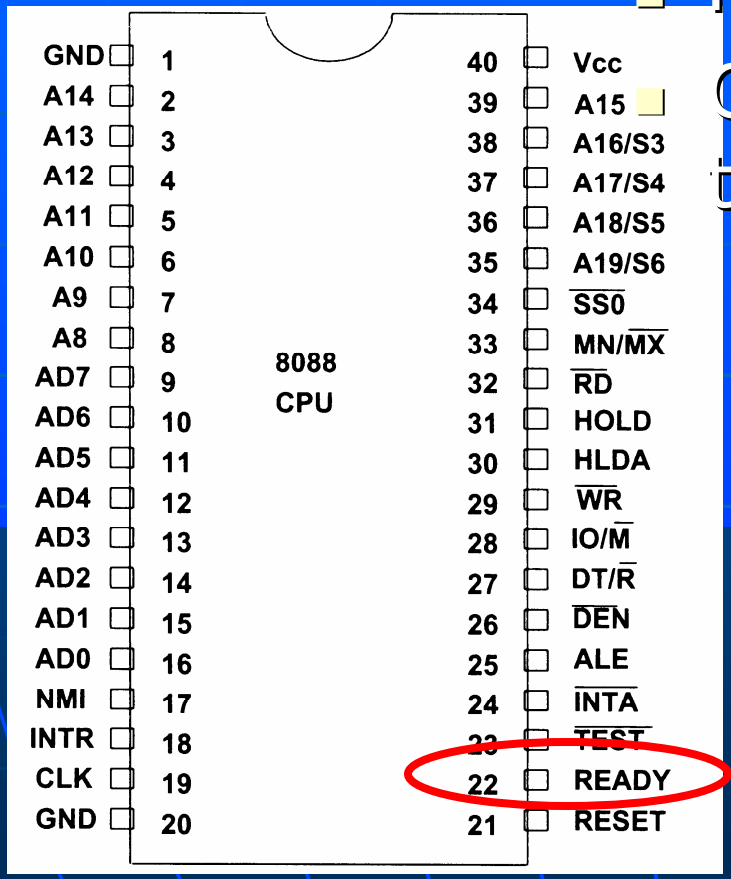
Kết thúc hoạt động hiện thời và huy bỏ mọi thứ

Sau khi reset

- CS=FFFFH
- DS=0000H
- SS=0000H
- ES=0000H
- IP=0000H
- Các cờ bị xoá
- Hàng đợI lệnh rỗng

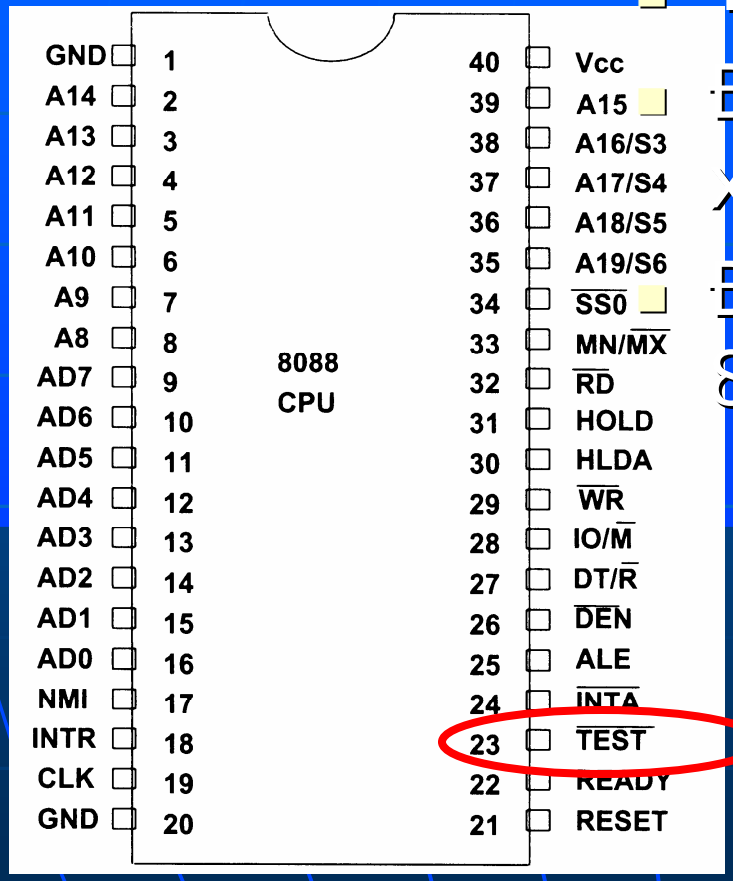
Mô tả chân

- READY
  - Chèn thêm một trạng thái đợi (wait state)



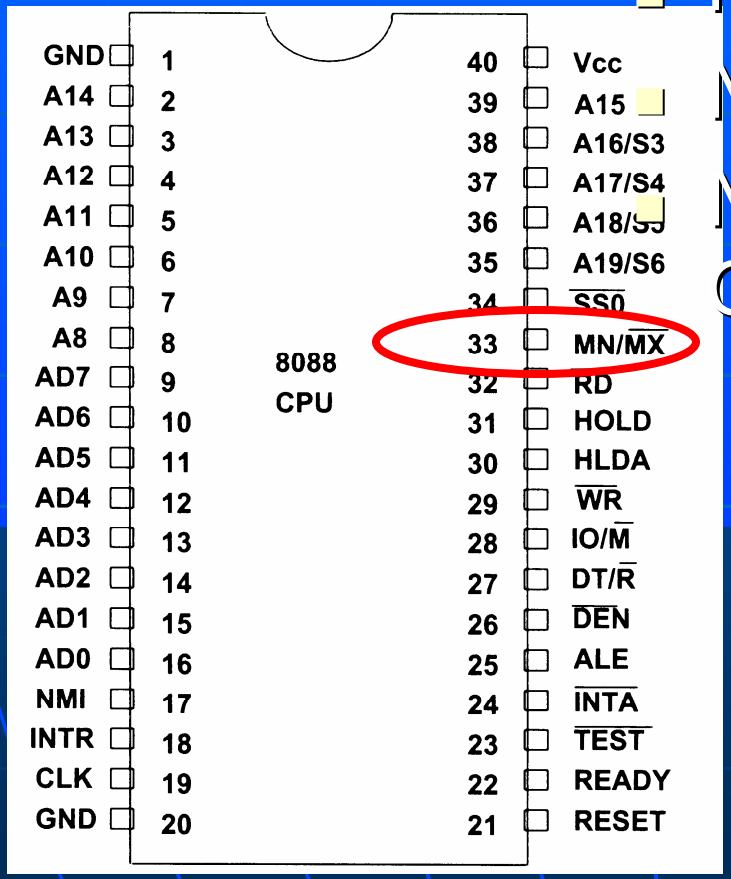
Mô tả chân

- TEST
  - Đến từ 8087 (Bộ đồng xử lý)
  - Đồng bộ 8088 và 8087

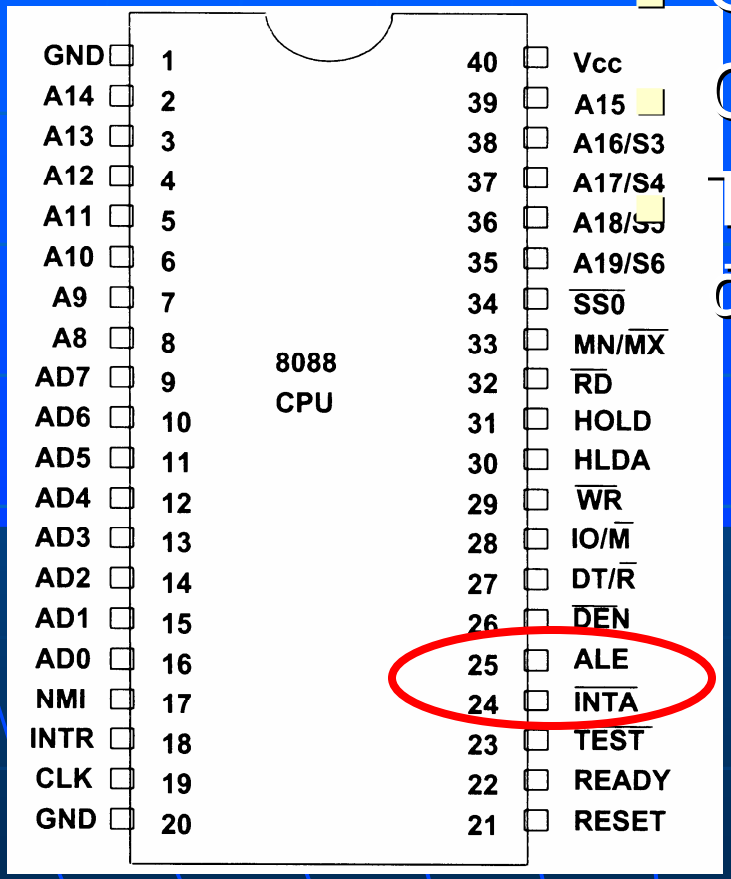


Mô tả chân

- MN/MX
- Minimum mode = +5V
- Maximum mode = Gnd



Mô tả chân – Max



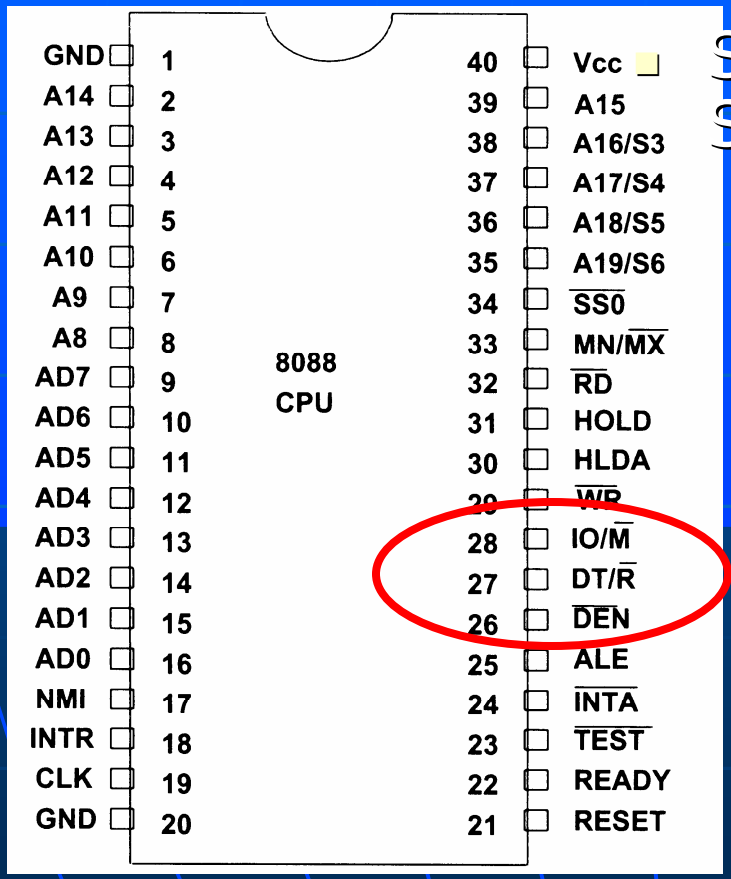
- QS0, QS1

Queue status

Trạng thái của hàng đợi lệnh:

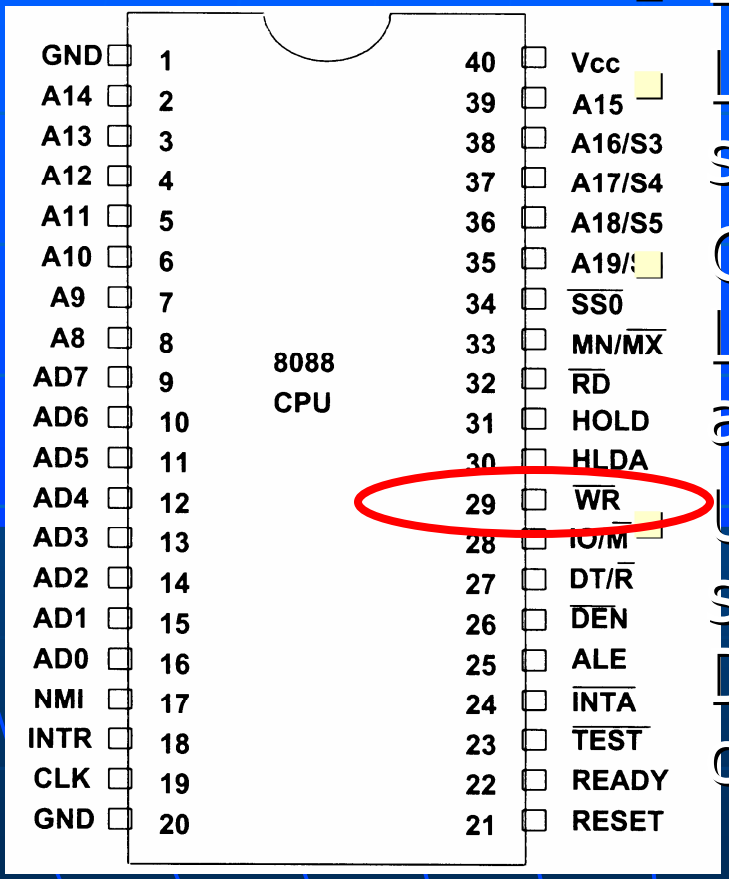
- 00 – No operation
- 01 – first byte of opcode from queue
- 10 – empty the queue
- 11 – subsequent byte from queue

Mô tả chân – Max



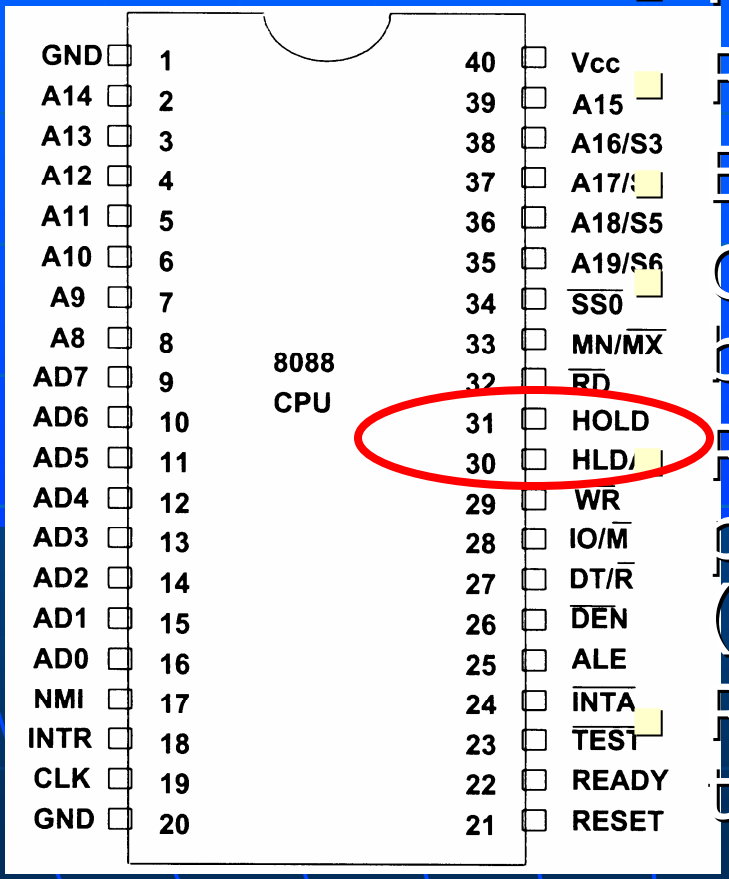
- S0, S1, S2 Status Signal Pins (S2-S0)
  - 000 – INTA – interrupt acknowledge
  - 001 – IORC – read I/O port
  - 010 – IOWC – write I/O port
  - 011 – none - halt
  - 100 – MRDC – code access
  - 101 – MRDC – read memory
  - 110 – MWTC – write memory
  - 111 – none - passive

Mô tả chân – Max



- LOCK
  - Locks processor to system bus
  - Gain the lock by using LOCK prefix on an assembly instruction
  - Used with status signals to prevent DMA from gaining control of the buses

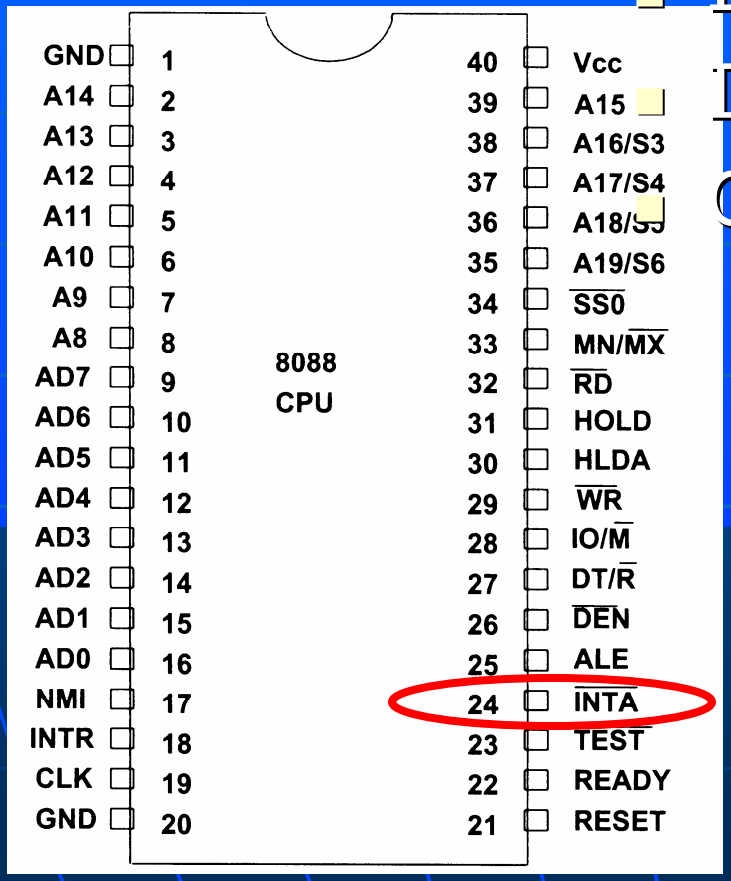
Mô tả chân – Max



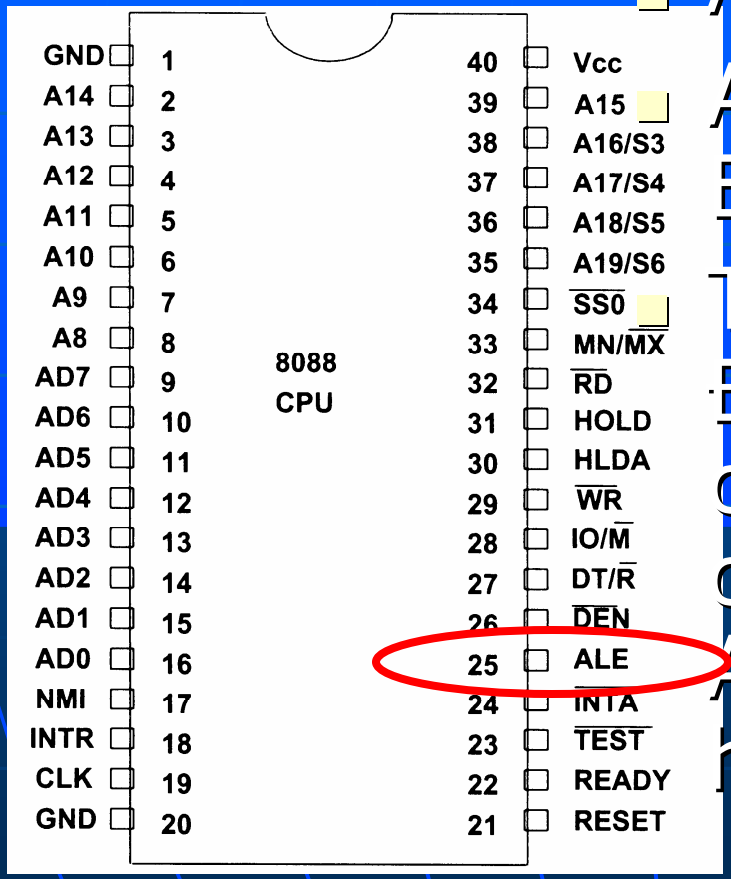
- RQ/GT0, RQ/GT1
  - Request/Grant
  - Bi-directional
  - Gain control of local bus
  - RQ/GT0 normally permanently high (disabled)
  - RQ/GT1 is connected to the 8087

Mô tả chân – Min

- $\overline{INTA}$
  - Interrupt acknowledge
  - Chấp nhận ngắt

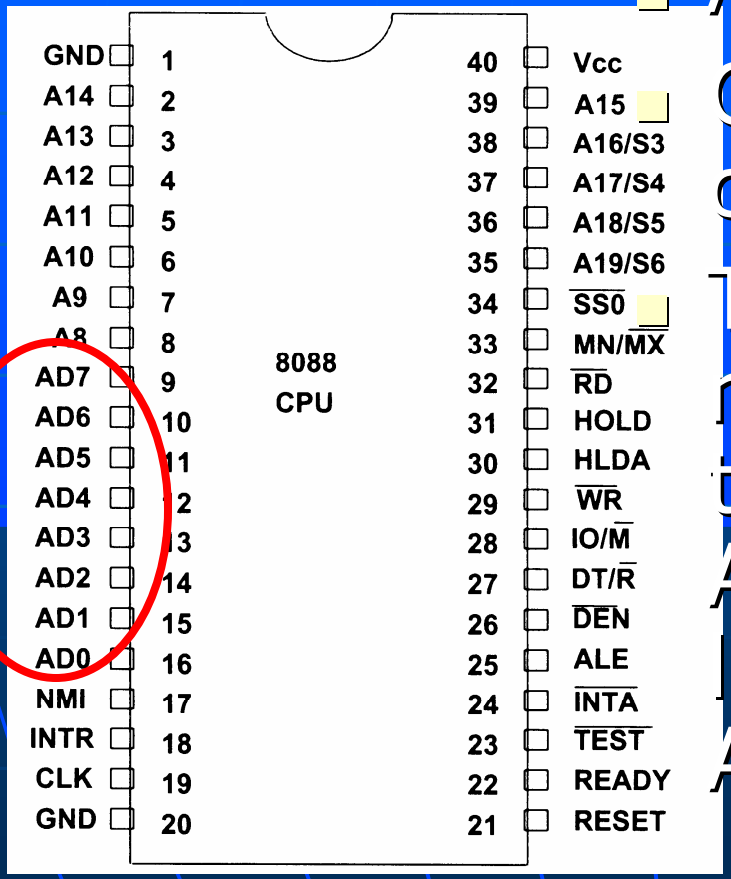


Mô tả chân – Min

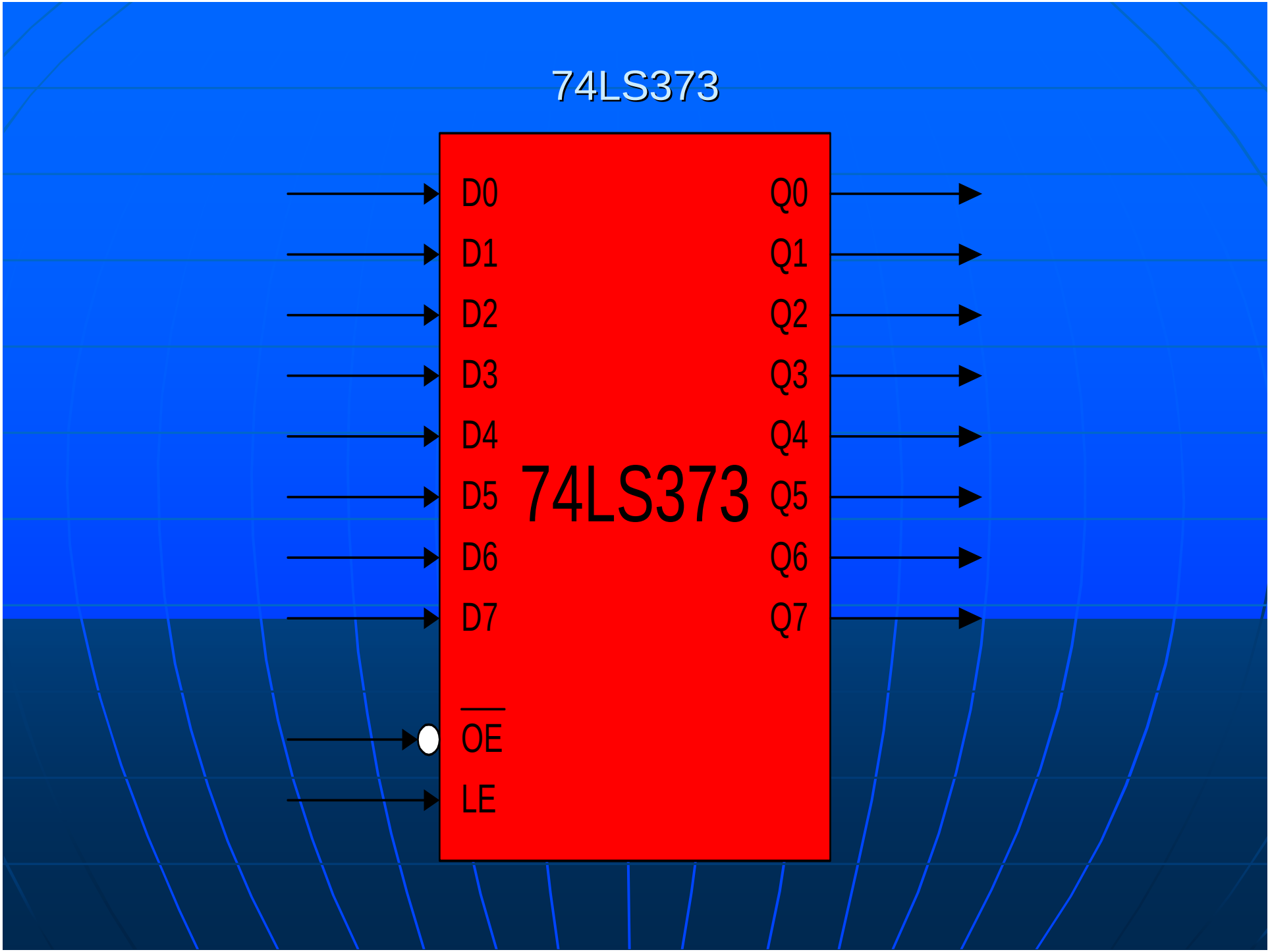


- ALE
- Address Latch Enable
- Tín hiệu ở các chân Địa chỉ/Dữ liệu và các chân Địa chỉ/Trạng thái lúc ALE = 1 là các tín hiệu địa chỉ

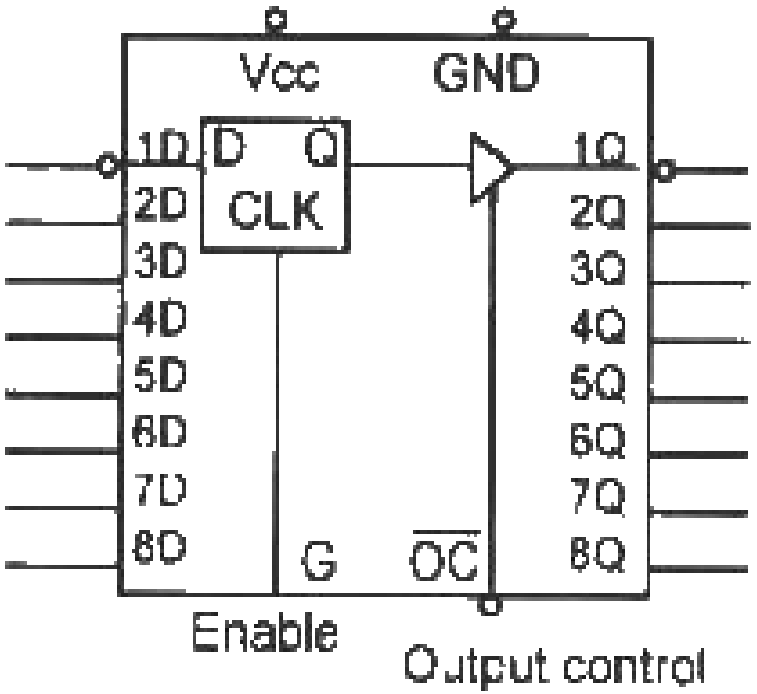
Mô tả chân



- AD0-AD7
  - Các chân Địa chỉ/Dữ liệu
  - Tín hiệu ở các chân này là 8 bit địa chỉ thấp A0 đến A7 khi ALE =1, là 8 bit dữ liệu D0 đến D7 khi ALE = 0


74LS373
D0
D1
D2
D3
D4
D5
D6
D7
74LS373
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
OE
LE

74LS373



Function Table

Output Control	Enable G	Enable D	Output
L	H	H	H
L	H	L	L
L	L	X	Q0
H	X	X	Z

Figure 11-1. 74LS373 D Latch

Dùng 74LS373 để tách và chốt địa chỉ

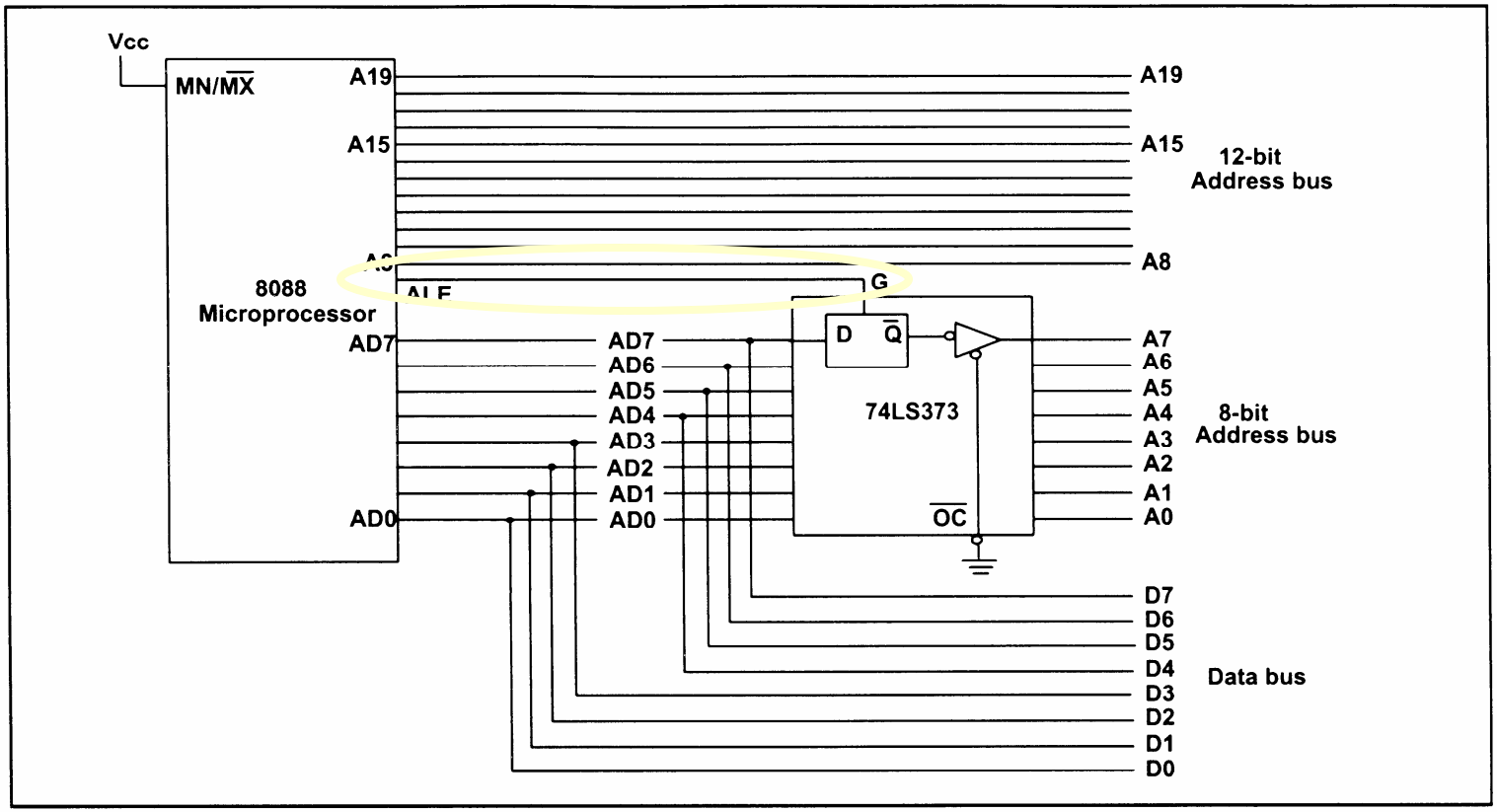
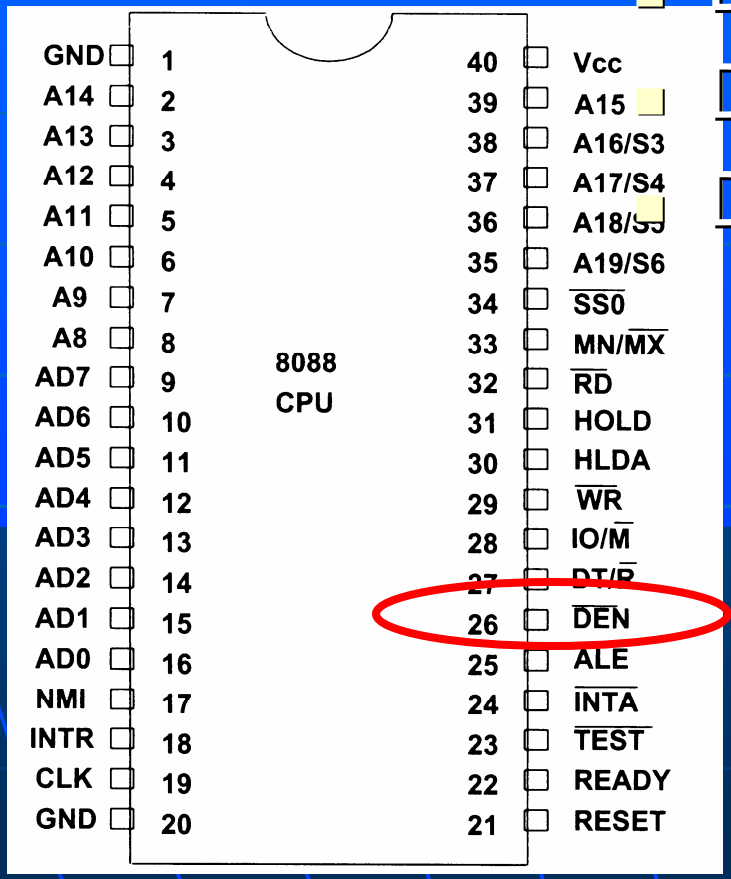


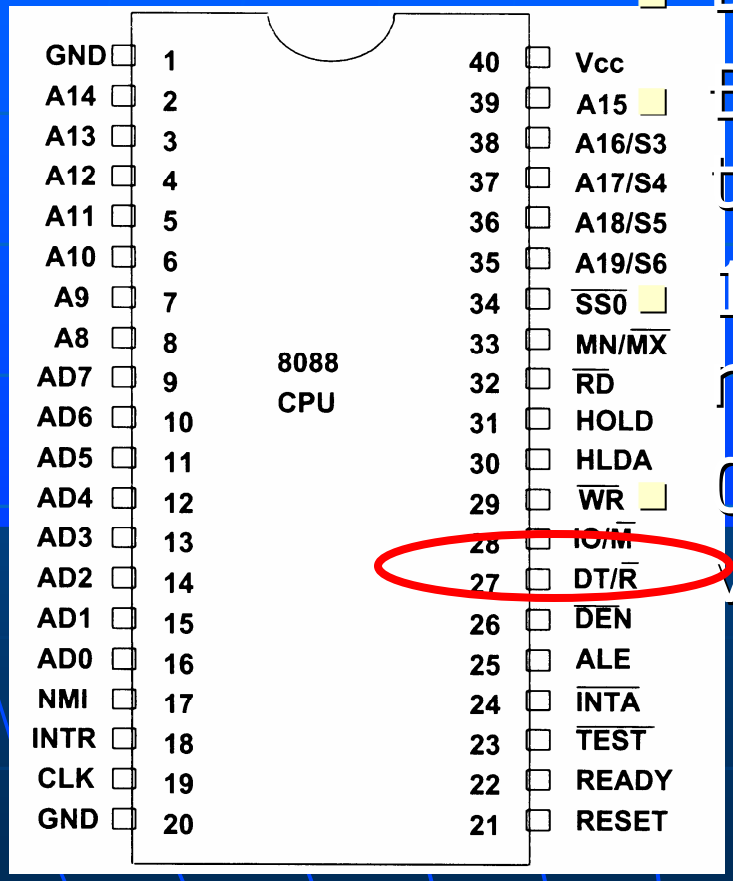
Figure 9-3. Role of ALE in Address/Data Demultiplexing

Mô tả chân – Min



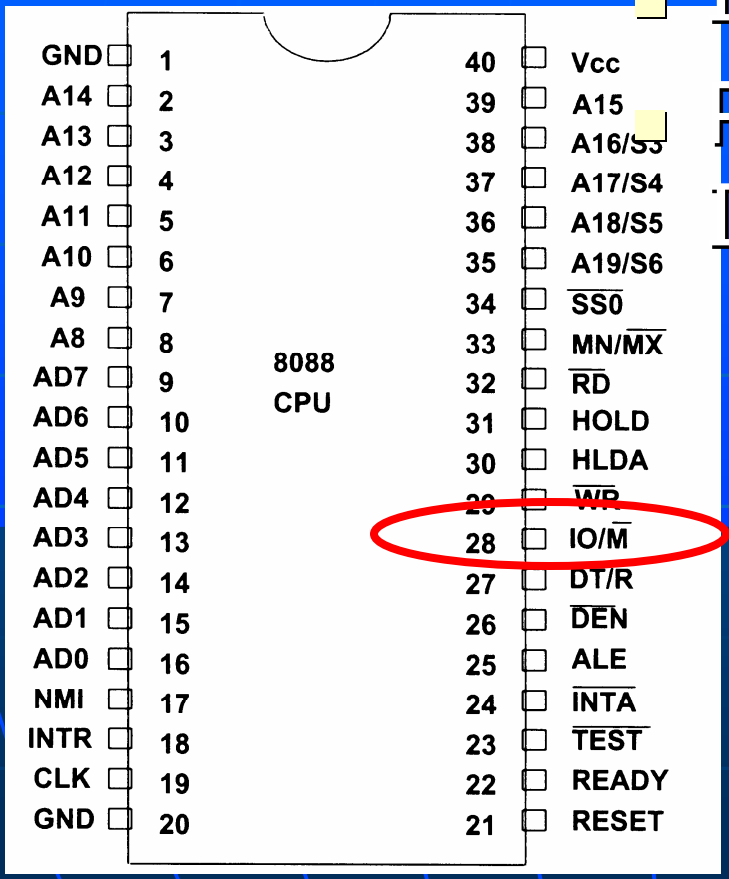
- DEN
  - Data Enable
  - Dữ liệu có nghĩa

Mô tả chân – Min



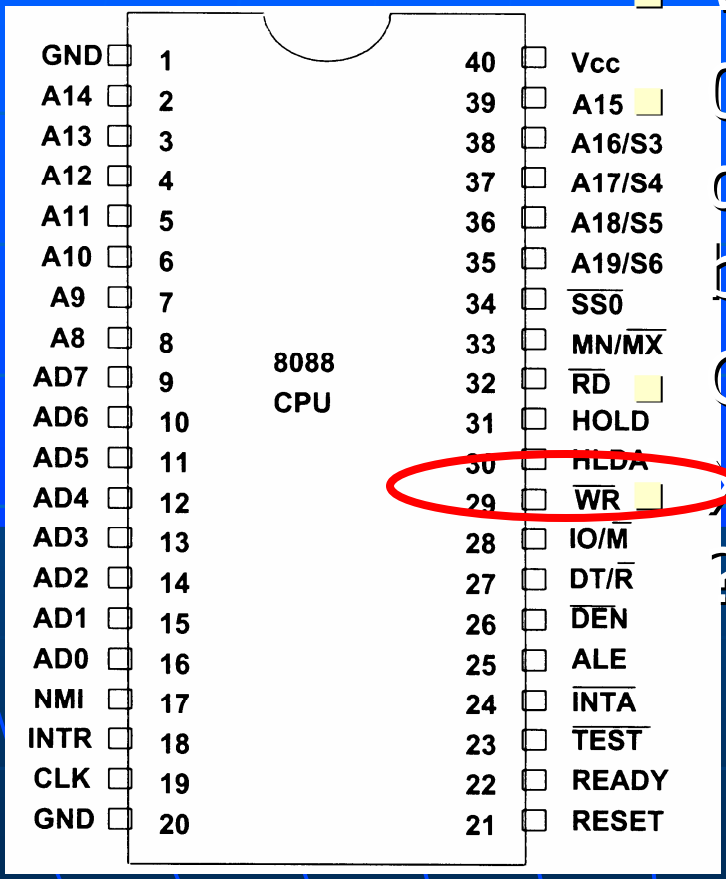
- DT/R _
  - Điều khiển hướng của tín hiệu dữ liệu:
  - 1: Tín hiệu dữ liệu đi ra từ 8088
  - 0: Tín hiệu dữ liệu đi vào 8088

Mô tả chân – Min



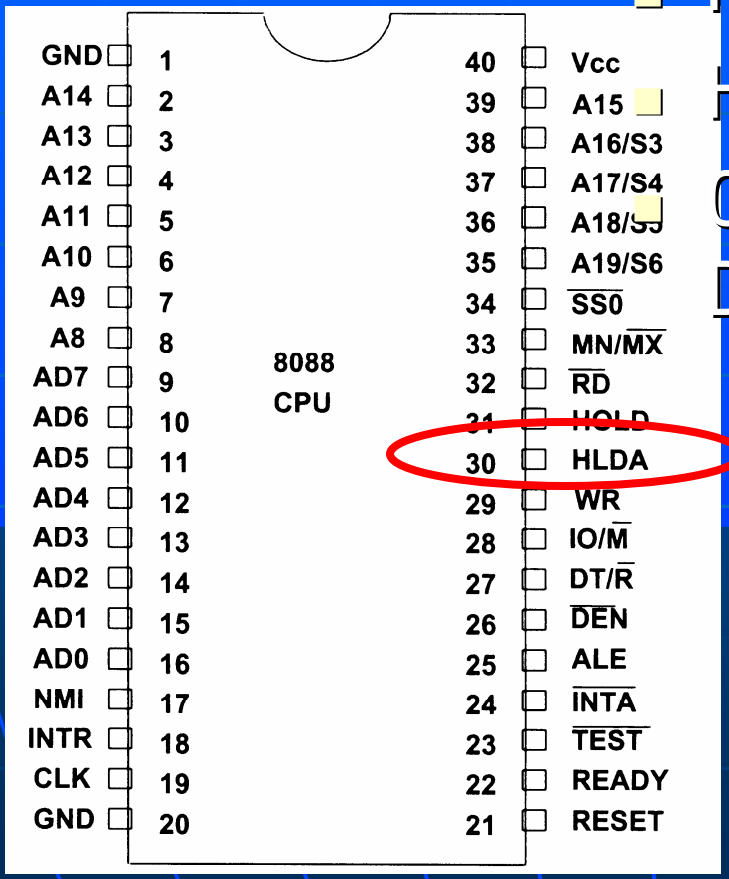
- IO/M _
- Phân biệt: truy cập I/O hay Bộ nhớ
  - 1: 8088 truy cập I/O
  - 0: 8088 truy cập bộ nhớ

Mô tả chân – Min



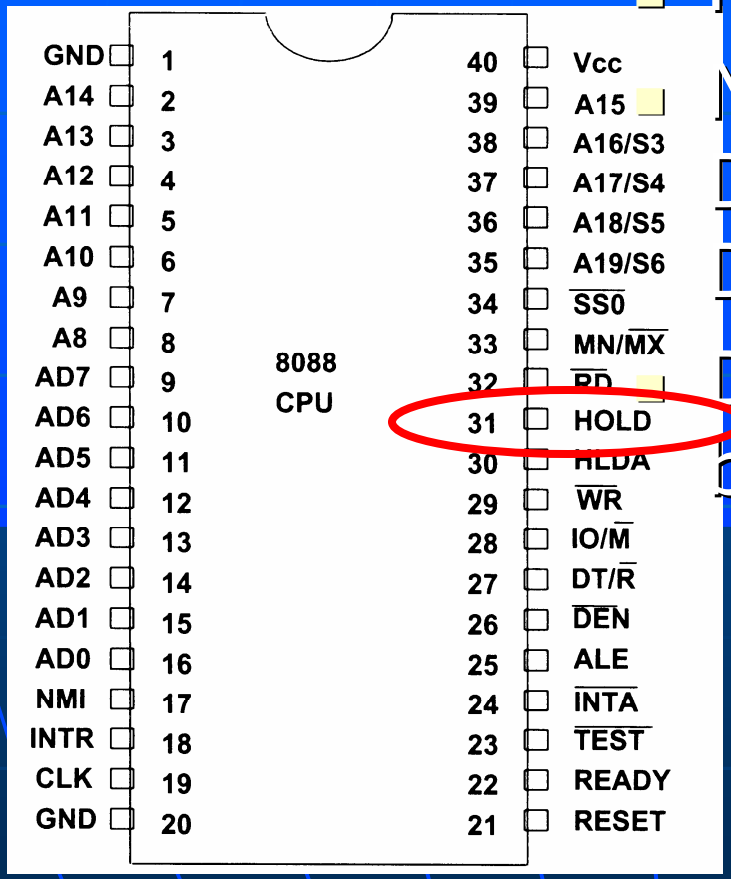
- WR__
  - 0: Tín hiệu trên bus dữ liệu được ghi vào bộ nhớ hoặc I/O
  - Ghi bộ nhớ: ?
  - Xuất dữ liệu ra cổng: ?

Mô tả chân – Min



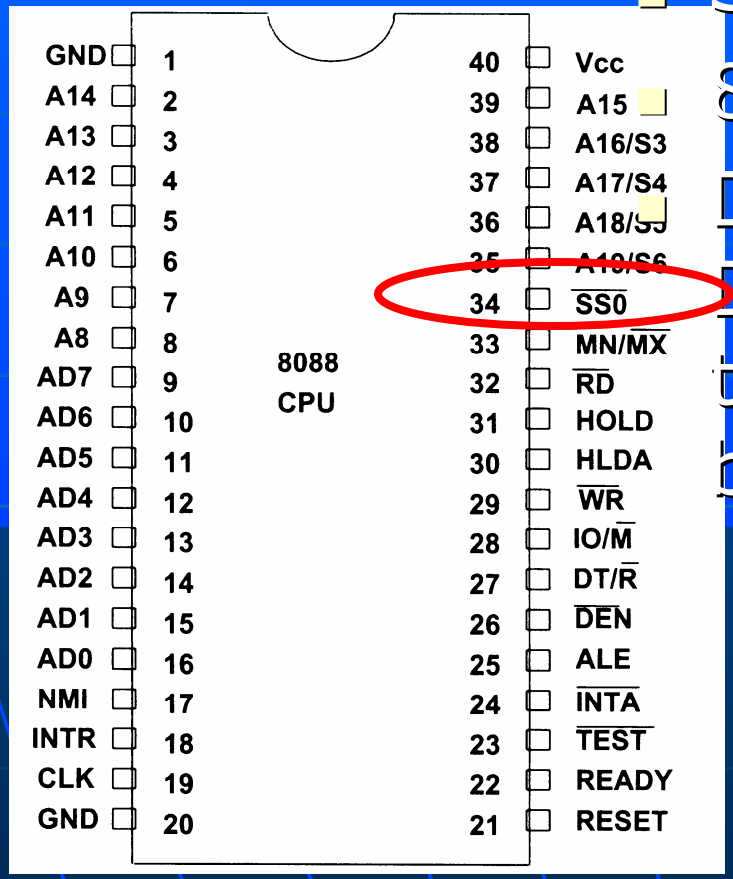
- HLDA
  Hold Acknowledge
  0: Chấp nhận yêu cầu DMA ở HOLD
  - Báo cho Bộ điều khiển DMA được phép sử dụng bus hệ thống

Mô tả chân – Min



- HOLD
  - Nhận tín hiệu yêu cầu DMA từ Bộ điều khiển DMA (DMAC)
  - DMAC muốn sử dụng bus hệ thống

Mô tả chân – Min



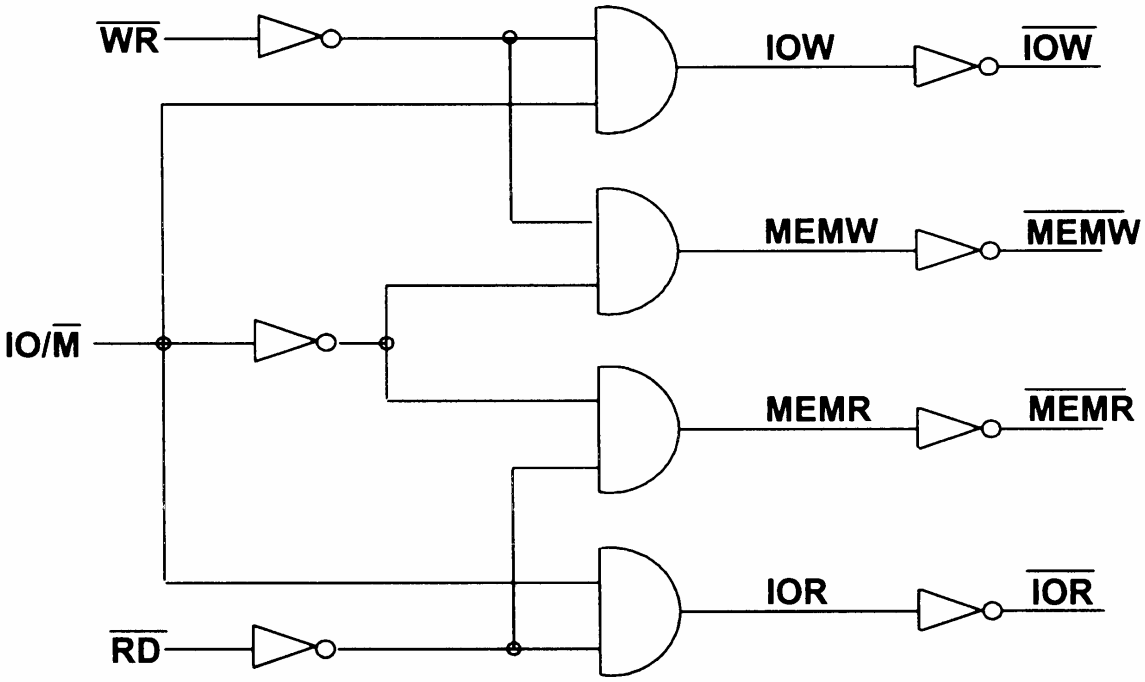
- SSO
- 8088
- Dùng với IO/M và DT/R để xác định trạng thái của chu kỳ bus hiện thời

Các tín hiệu điều khiển

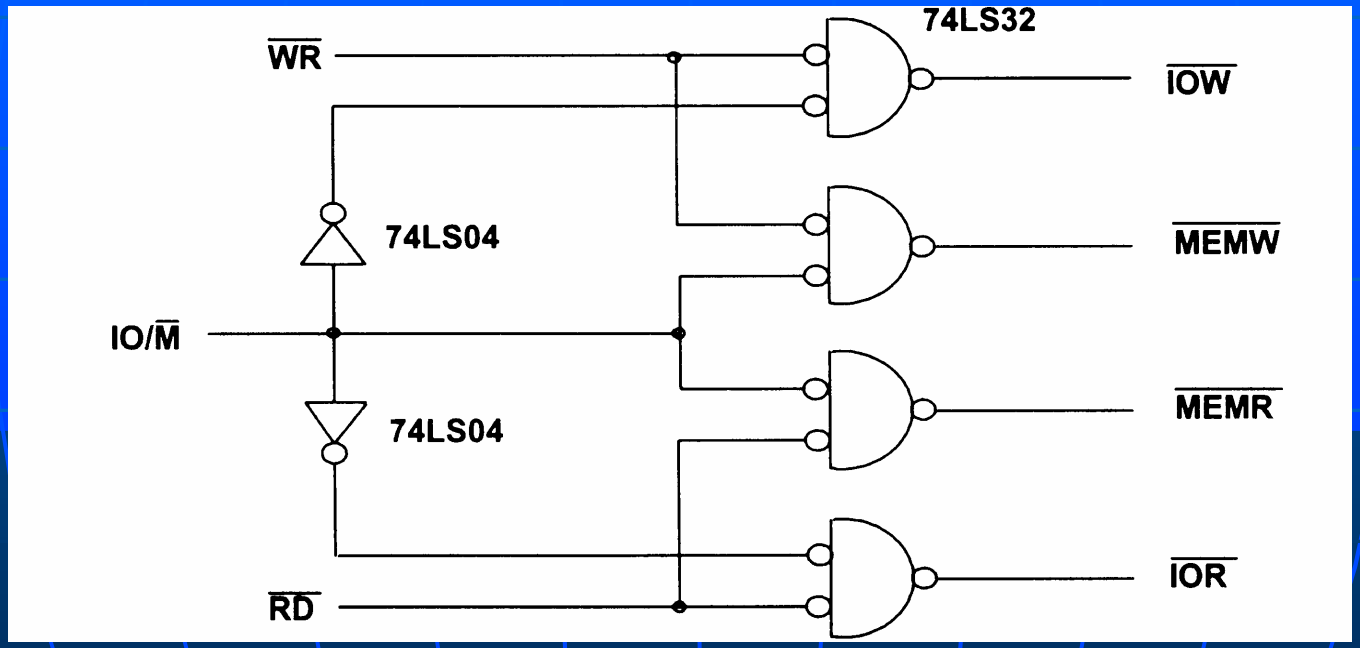
- Có thể sử dụng các cổng logic để tạo ra các tín hiệu điều khiển khác từ các tín hiệu điều khiển sẵn có
  - 3 Tín hiệu:
    - RD, WR and IO/M

$\overline{RD}$	$\overline{WR}$	IO/$\overline{M}$	Signal
0	1	0	$\overline{MEMR}$
1	0	0	$\overline{MEMW}$
0	1	1	$\overline{IOR}$
1	0	1	$\overline{IOW}$
0	0	X	Never happens

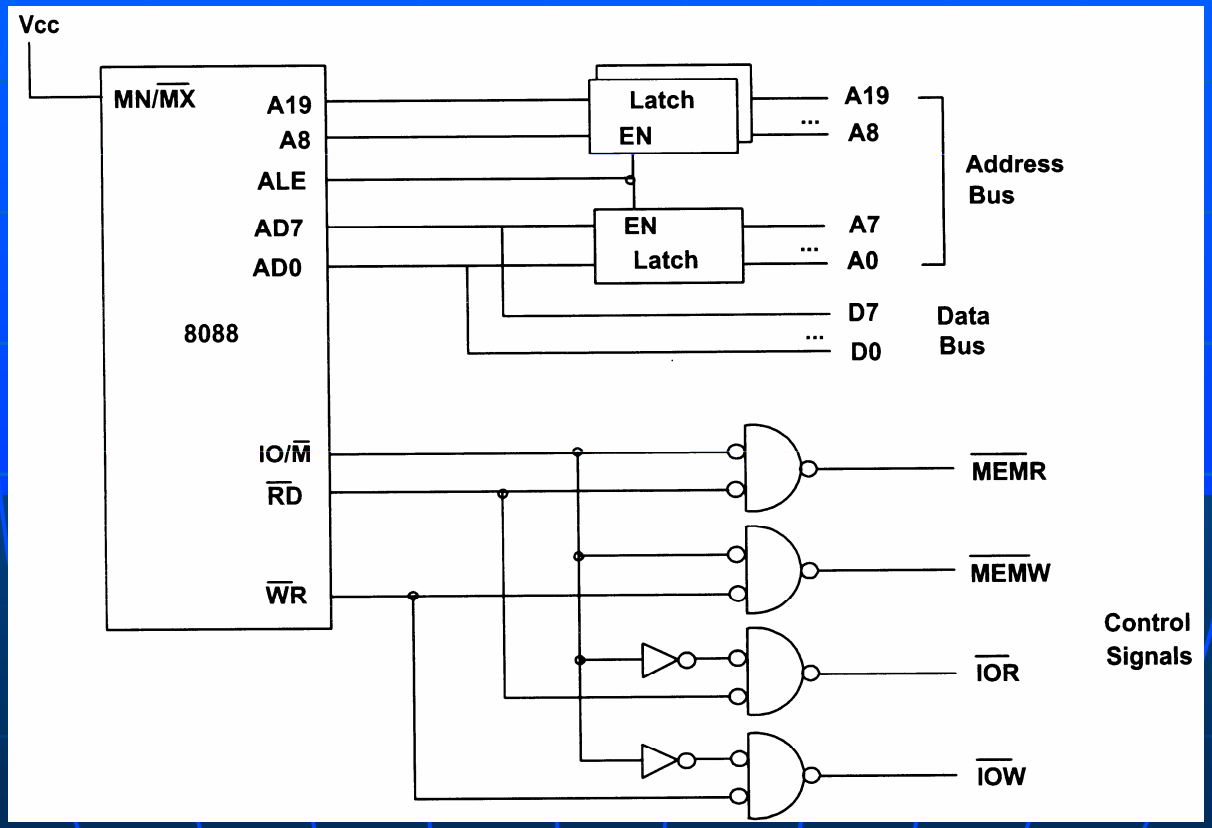
Tạo ra các tín hiệu điều khiển (Min Mode)



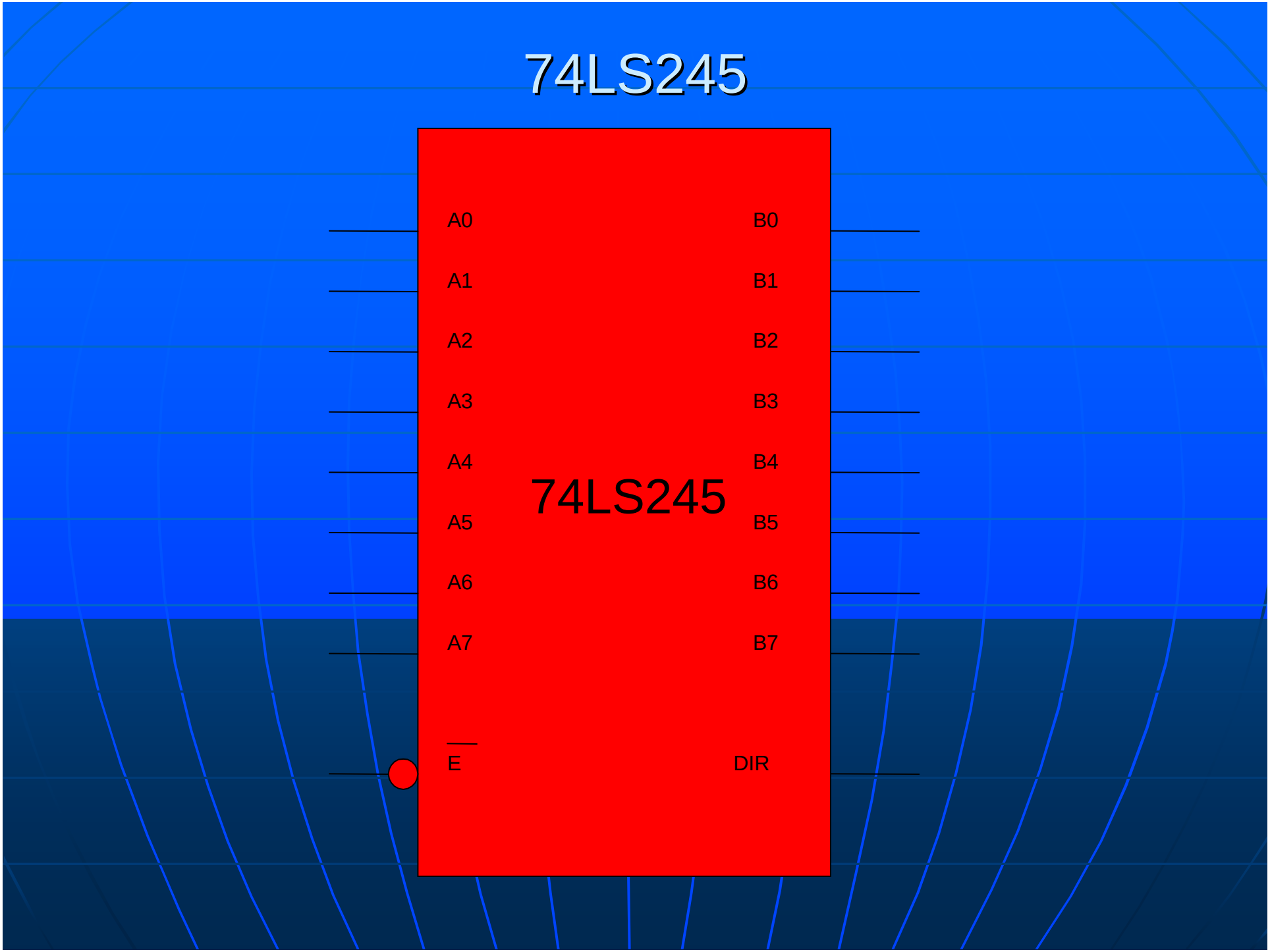
Tạo ra các tín hiệu điều khiển (Min Mode)



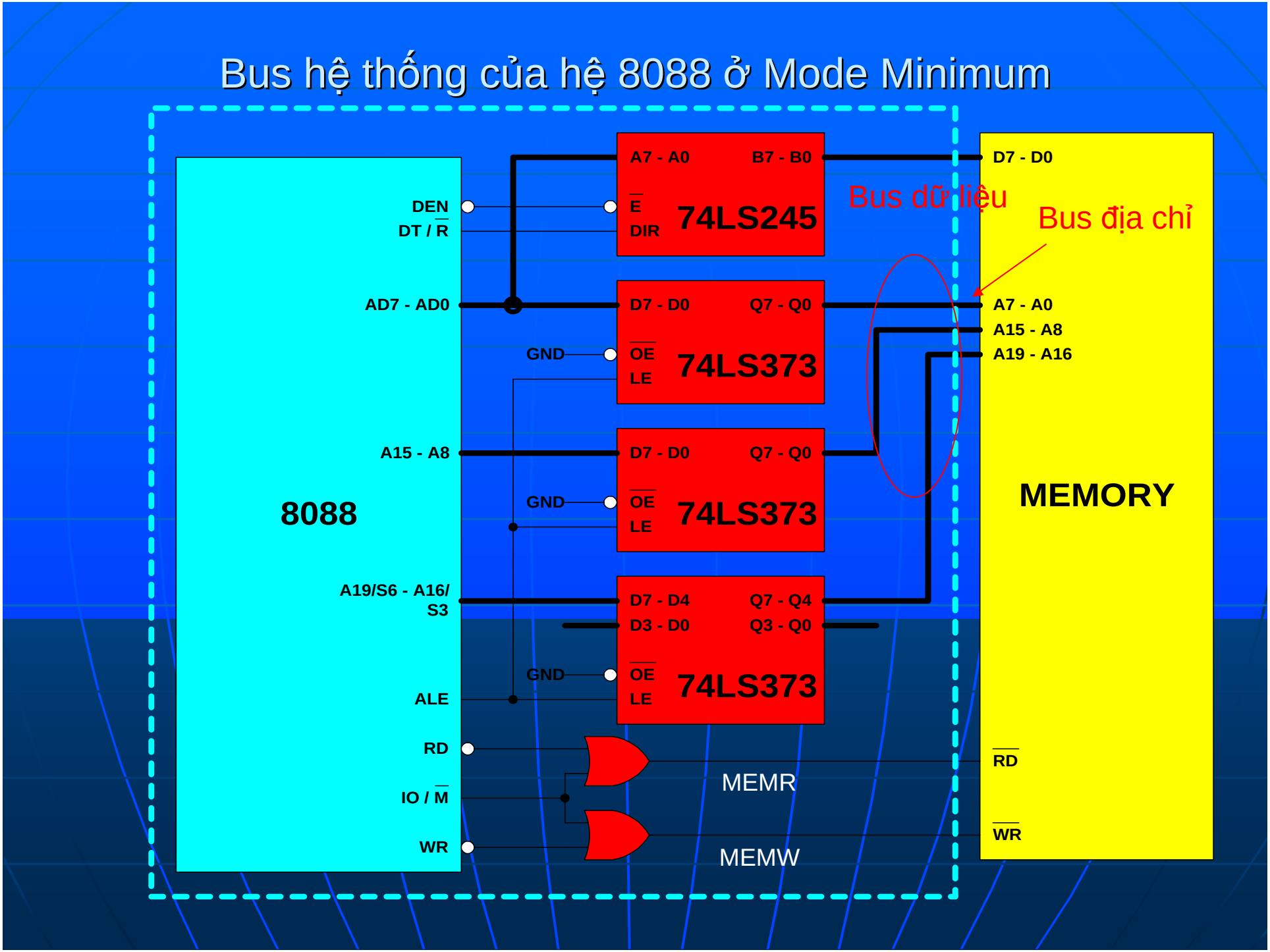
8088 Bus – Min Mode



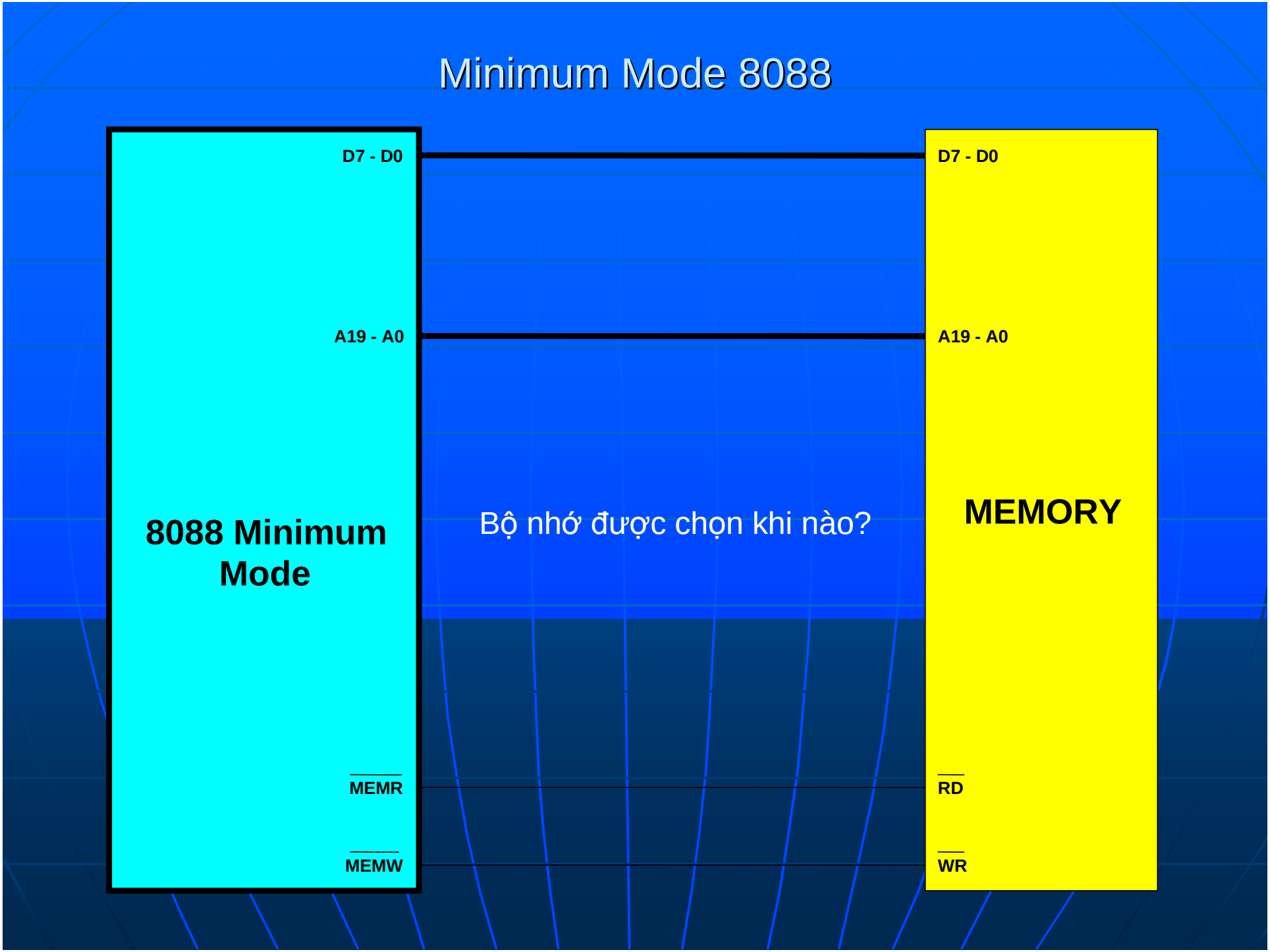
74LS245



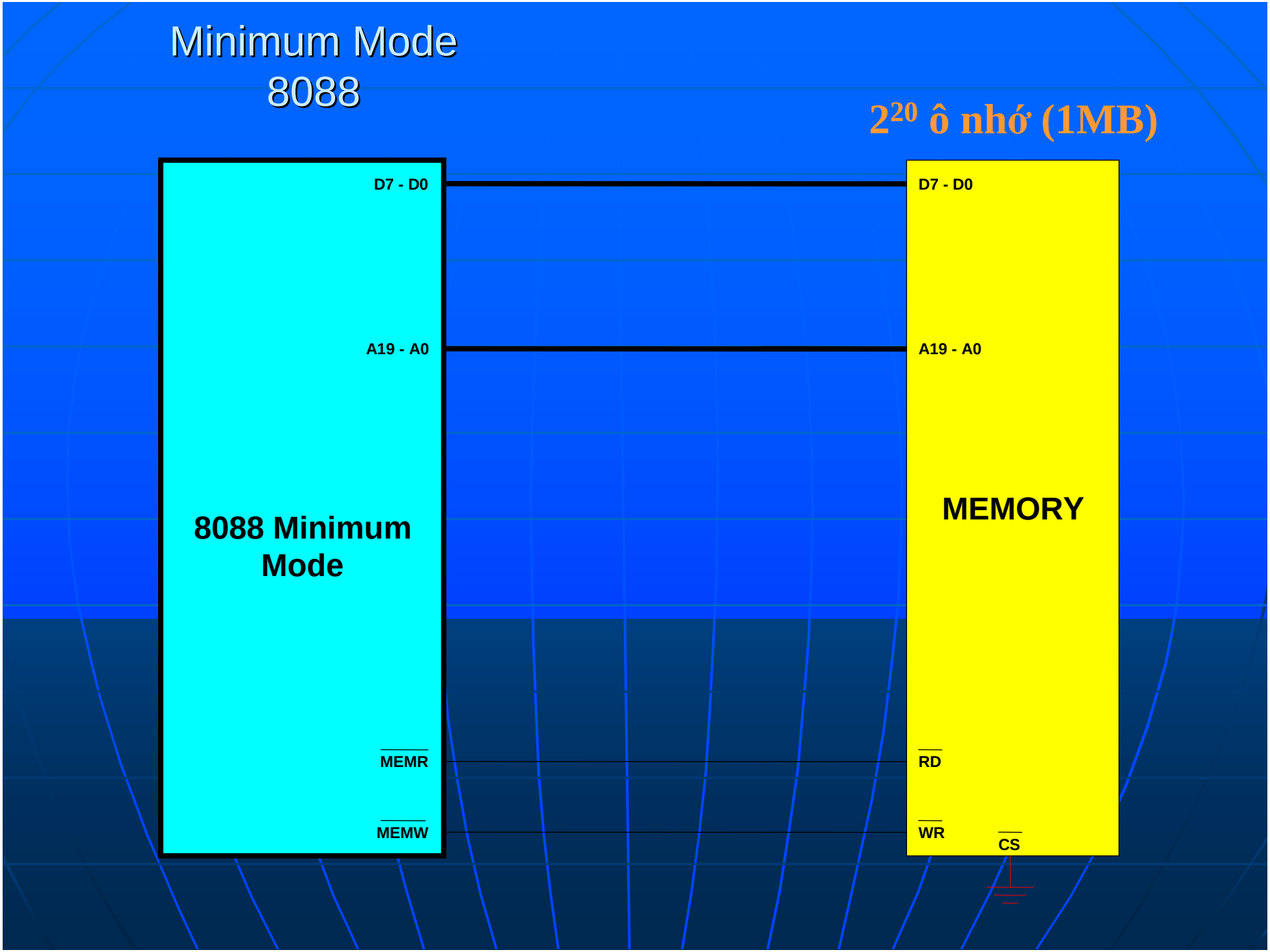
Bus hệ thống của hệ 8088 ở Mode Minimum



Minimum Mode 8088



Minimum Mode 8088



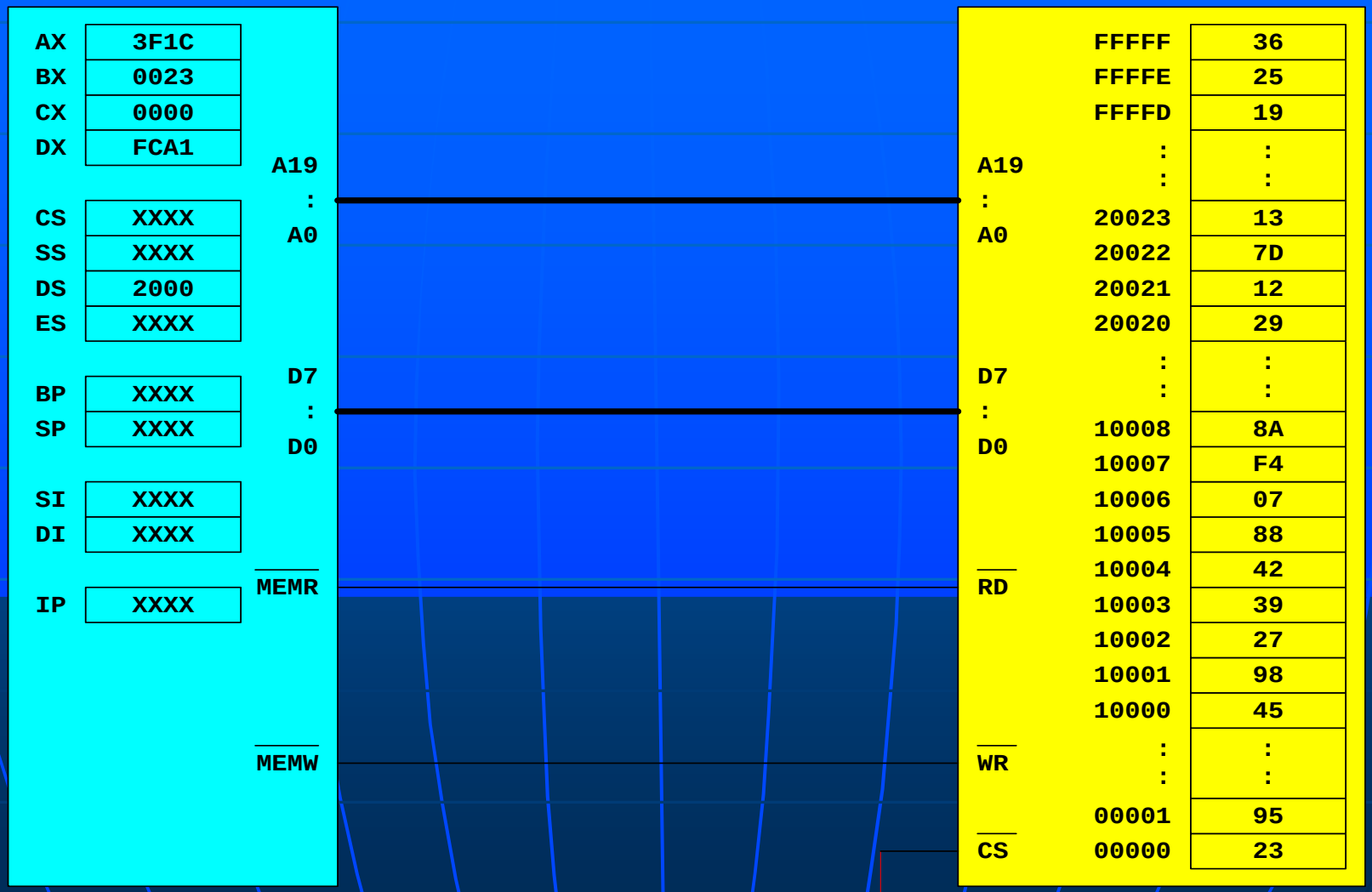
Không gian địa chỉ bộ nhớ 1M

A19 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

Ví dụ: Một địa chỉ bất kỳ 34FD0h

0011 0100 11111 1101 0000

Bộ nhớ đầy đủ 1MB



Nếu chỉ cần bộ nhớ có dung lượng nhỏ hơn 1MB thì giải quyết như thế nào?

- *Phụ thuộc vào các chip nhớ sẵn có*
- *Phụ thuộc yêu cầu phân bố địa chỉ cho các loại bộ nhớ vật lý khác nhau*
- *...*

512K đầu tiên của không gian địa chỉ bộ nhớ (Các địa chỉ có bit cao nhất A19 = 0)

A18 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111

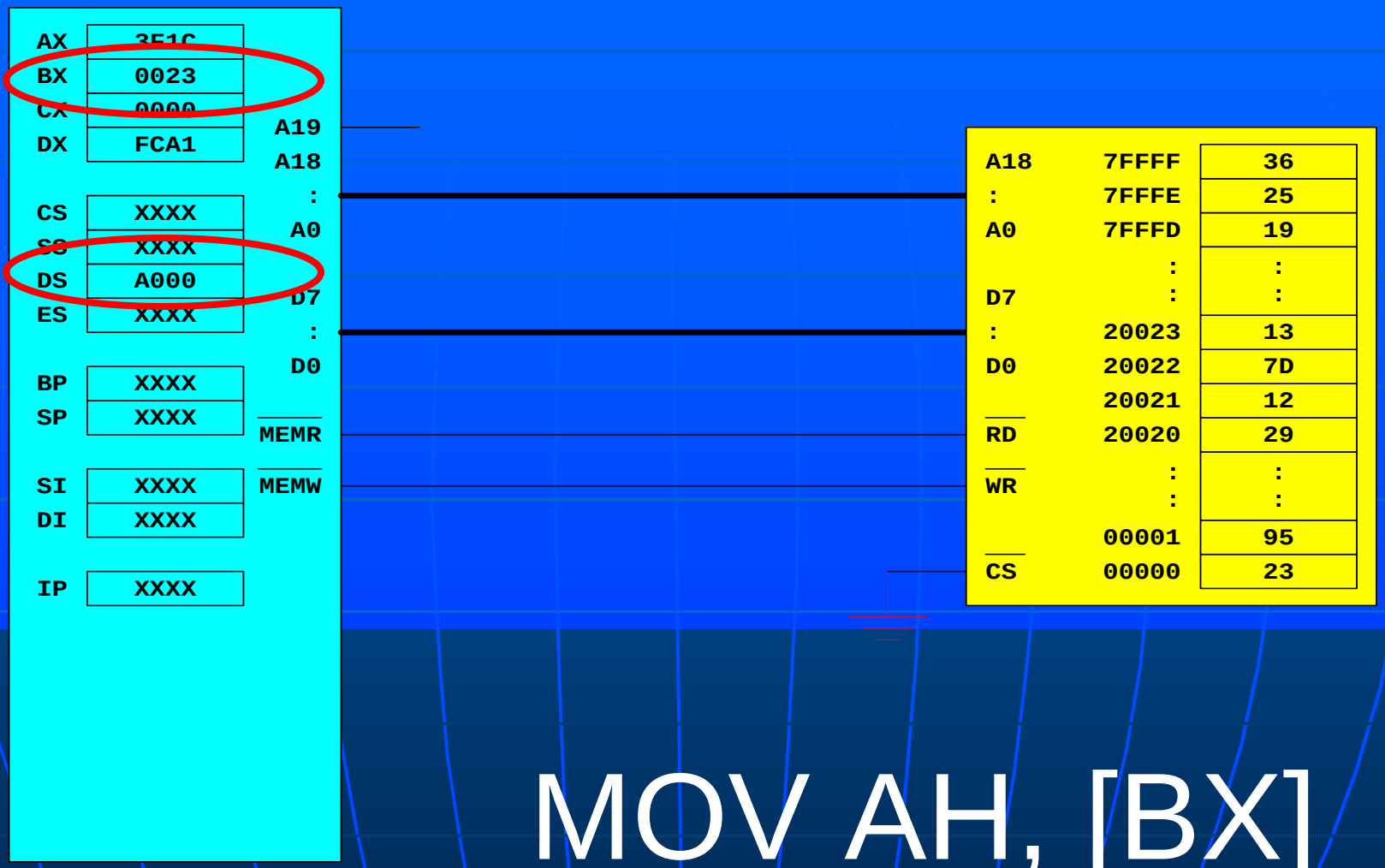
512K tiếp theo của không gian địa chỉ bộ nhớ (Các địa chỉ có bit cao nhất A19 = 1)

A18 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

Bộ nhớ 512KB

Làm gì với A19?

Điều gì xảy ra nếu 8088 đọc ô nhớ A0023h?



MOV AH, [BX]

Điều gì xảy ra nếu 8088 đọc ô nhớ A0023h?

A19 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
A0023	1010	0000	0000	0010	0011

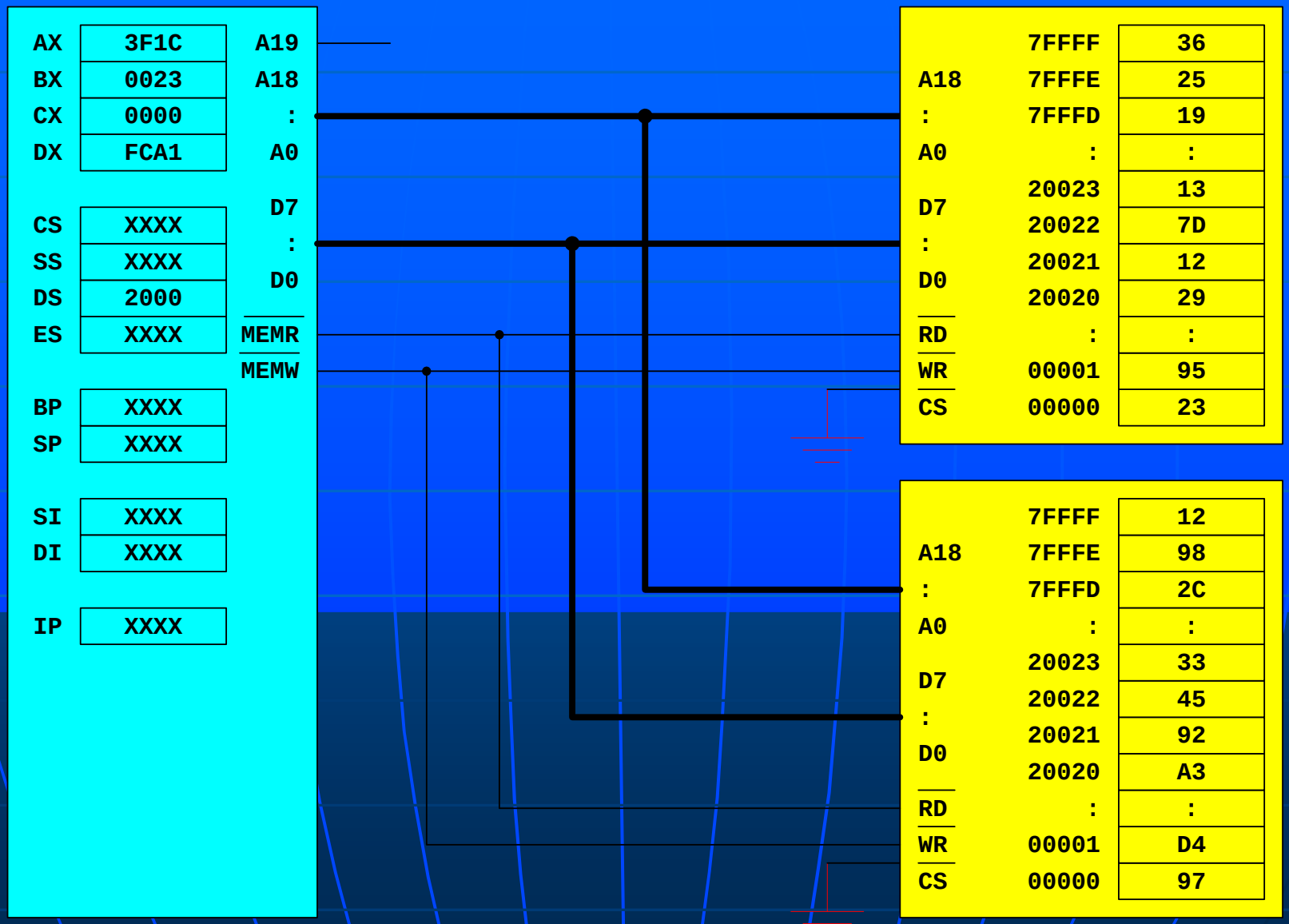
A19 không được nối đến bộ nhớ nên nếu 8088 phát logic "1" trên A19 thì bộ nhớ cũng không nhận biết được.

Điều gì xảy ra nếu 8088 đọc ô nhớ 20023h?

A18 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
20023	0010	0000	0000	0010	0011

Với bộ nhớ tình hình không có gì khác!

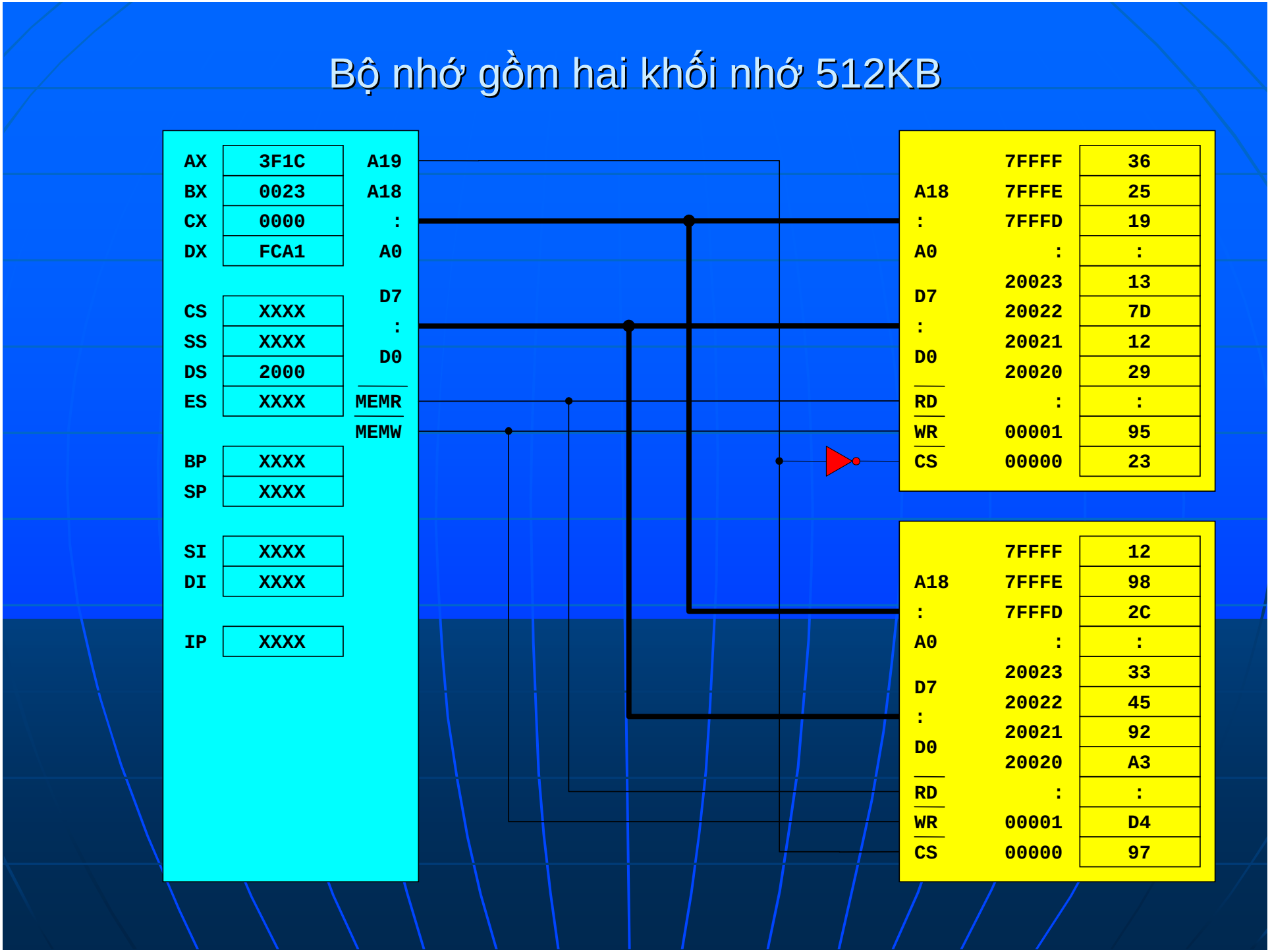
Nếu Bộ nhớ gồm 2 khối 512KB như thế này?



Có vấn đề !!!

- Vấn đề là: Xung đột Bus. Hai khối nhớ sẽ cung cấp dữ liệu cùng một lúc khi 8088 đọc bộ nhớ
- Giải pháp: Dùng A19 làm "người phân xử" để giải quyết xung đột trên bus. Nếu A19 ở mức logic "1" thì khối nhớ trên hoạt động (khối nhớ dưới bị cấm) và ngược lại

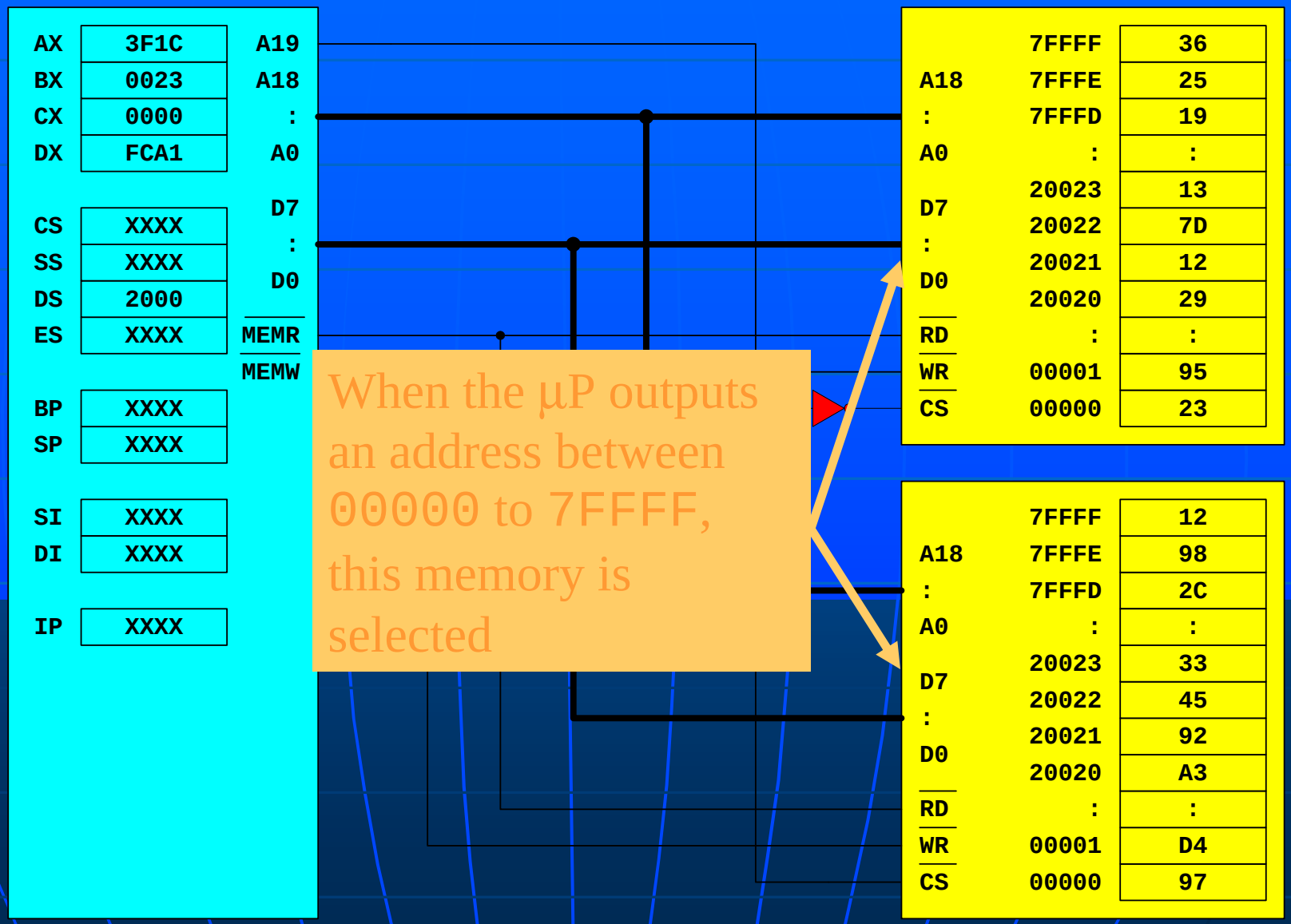
Bộ nhớ gồm hai khối nhớ 512KB



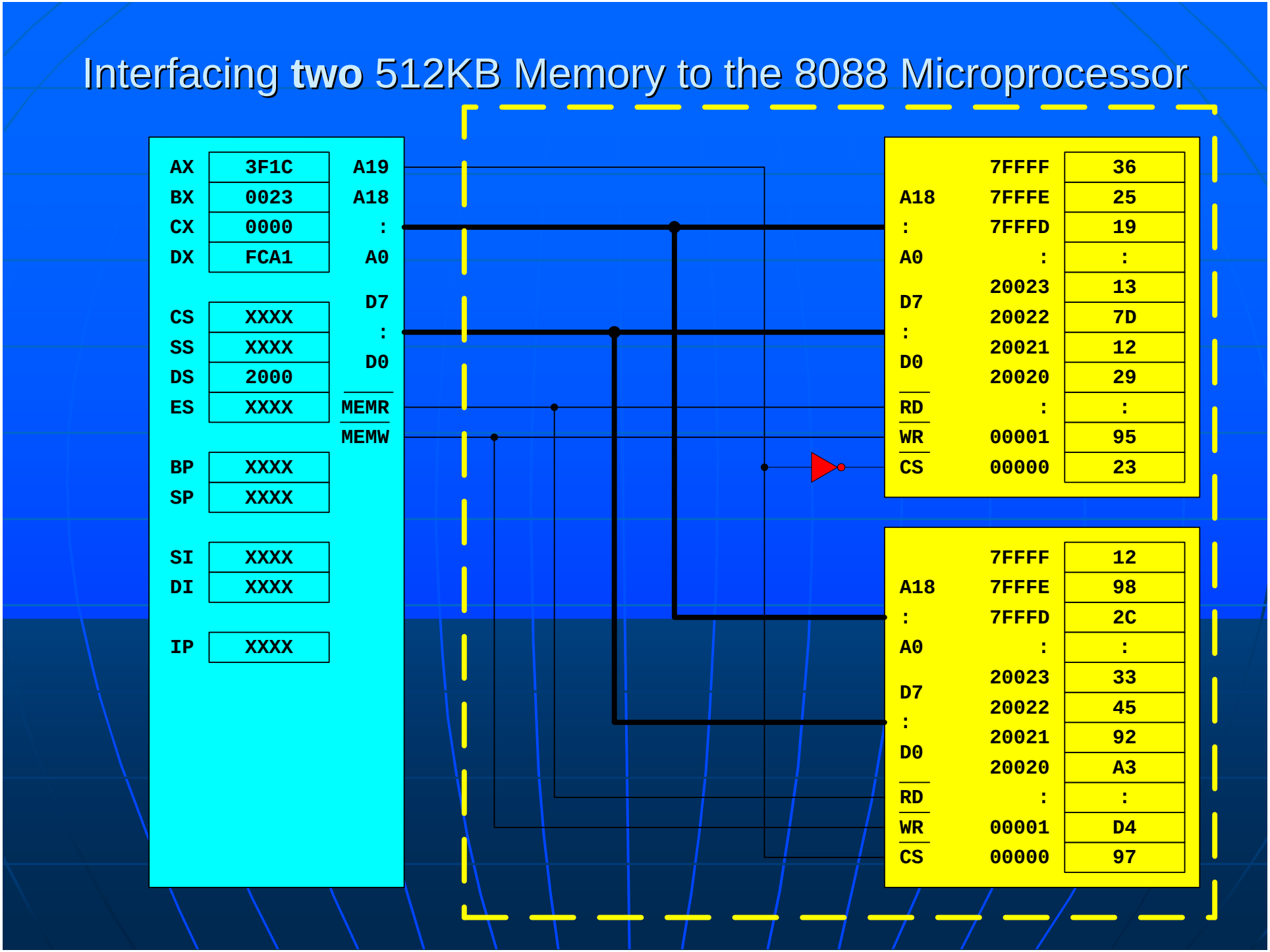
Không gian địa chỉ bộ nhớ 1M

A19 đến A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

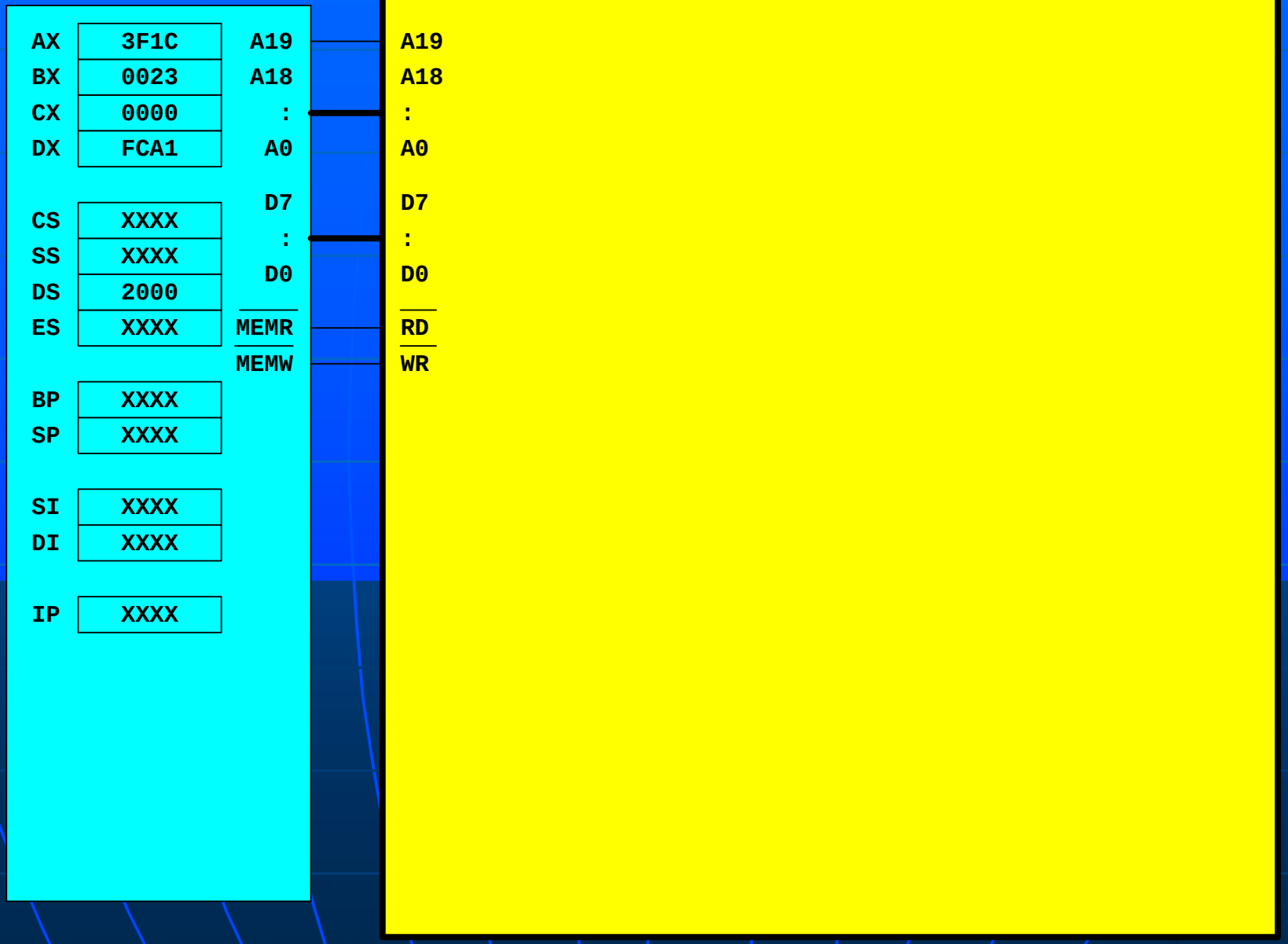
Interfacing **two** 512KB Memory to the 8088 Microprocessor



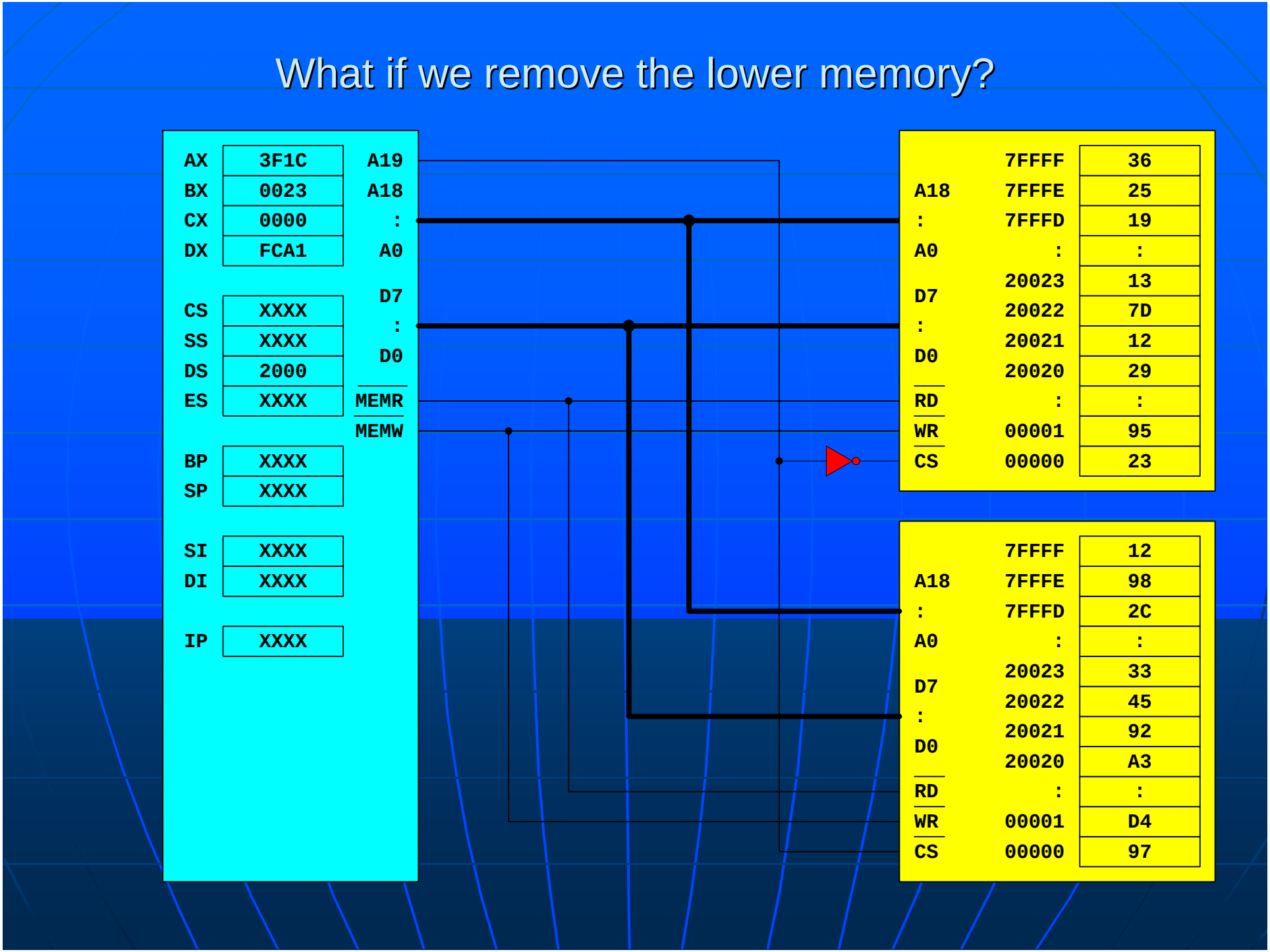
Interfacing **two** 512KB Memory to the 8088 Microprocessor



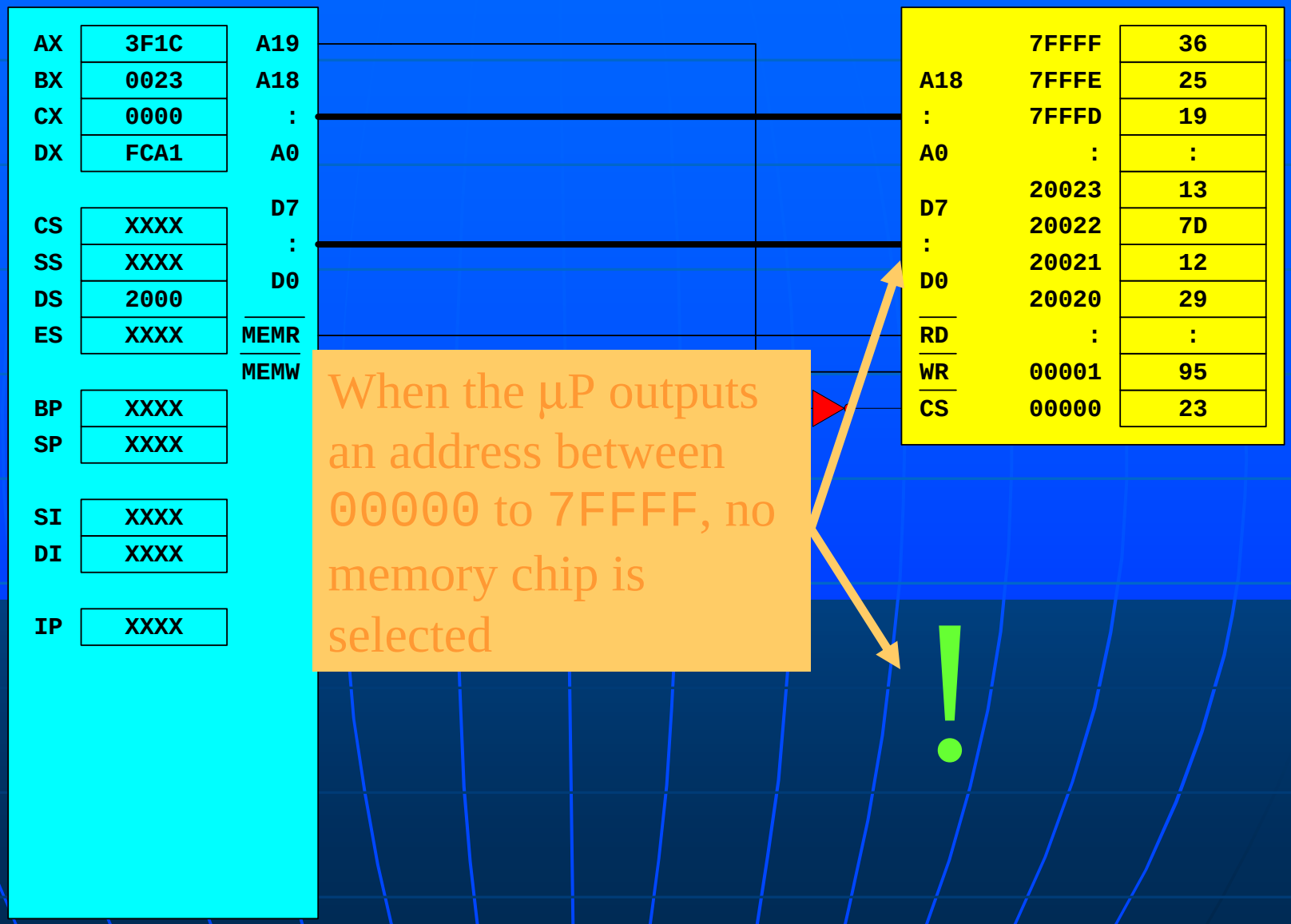
Interfacing **two** 512KB Memory to the 8088 Microprocessor



What if we remove the lower memory?



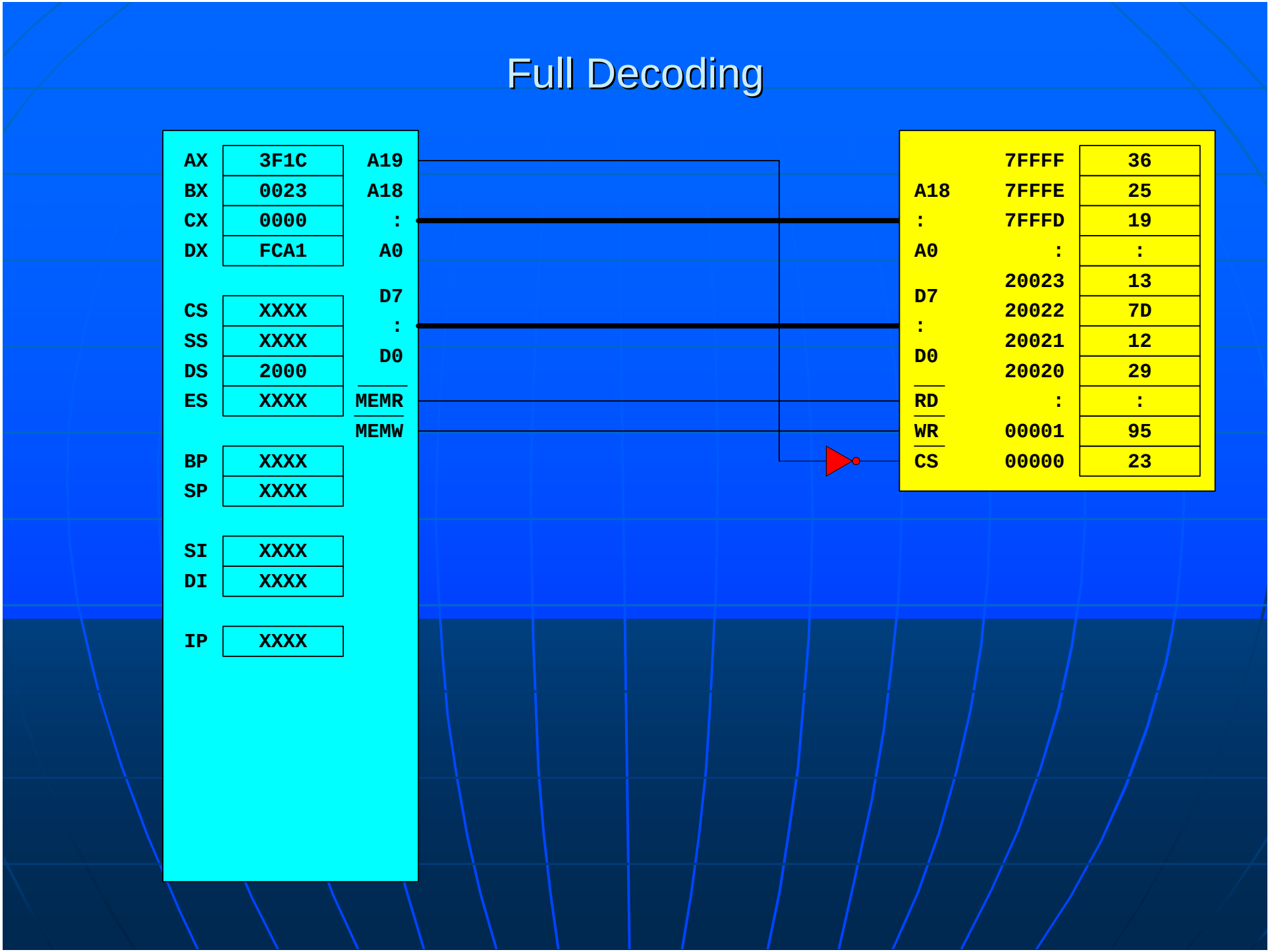
What if we remove the lower memory?



Full and Partial Decoding

- Full Decoding
  - When all of the "useful" address lines are connected the memory/device to perform selection
- Partial Decoding
  - When some of the "useful" address lines are connected the memory/device to perform selection
  - Using this type of decoding results into roll-over addresses

Full Decoding



Full Decoding

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

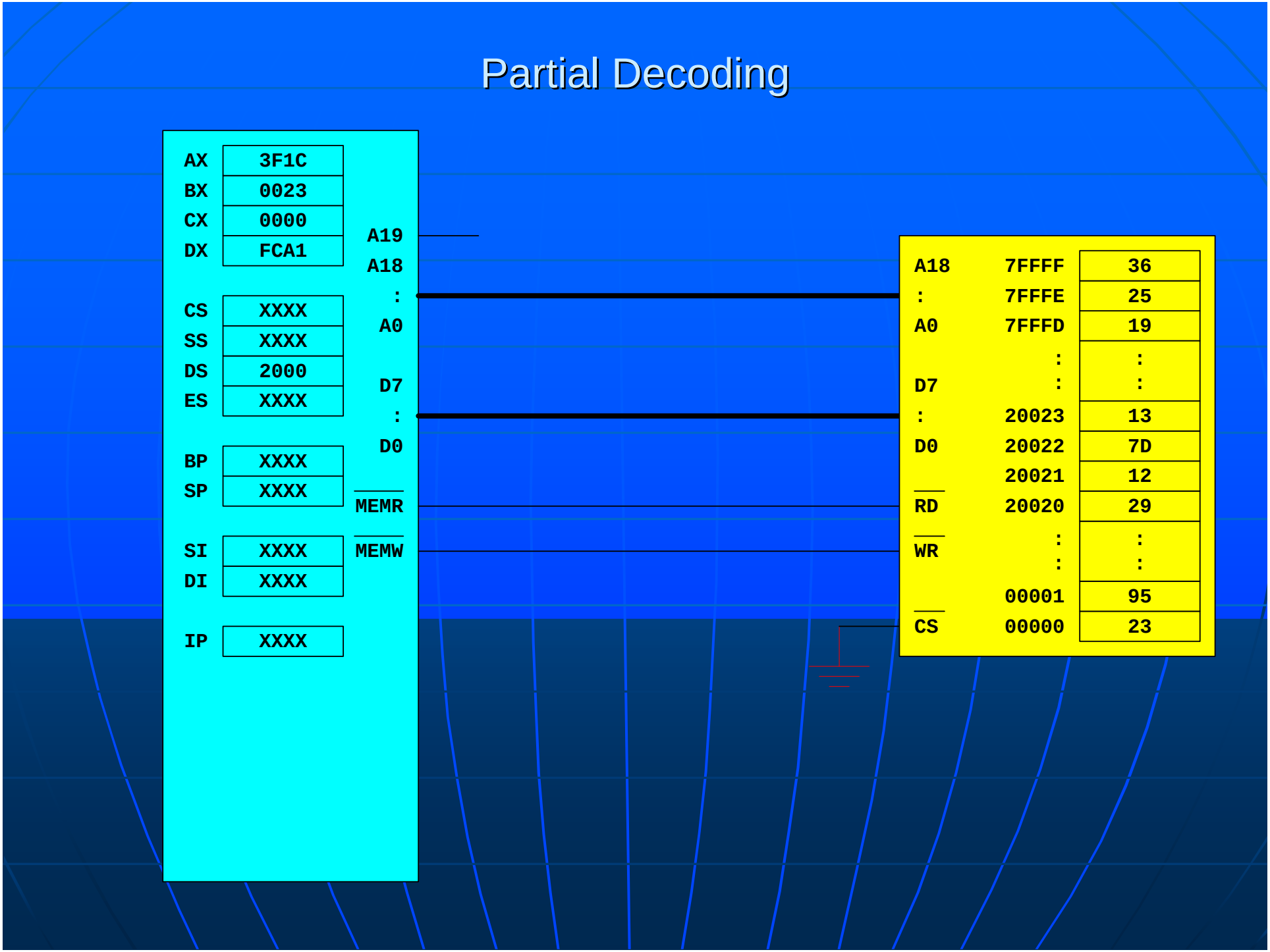
A19 should be a logic “1” for the memory chip to be enabled

Full Decoding

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111

Therefore if the microprocessor outputs an address between 00000 to 7FFFF, whose A19 is a logic “0”, the memory chip will not be selected

Partial Decoding



Partial Decoding

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

The value of A19 is INSIGNIFICANT to the memory chip, therefore A19 has no bearing whether the memory chip will be enabled or not

Partial Decoding

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

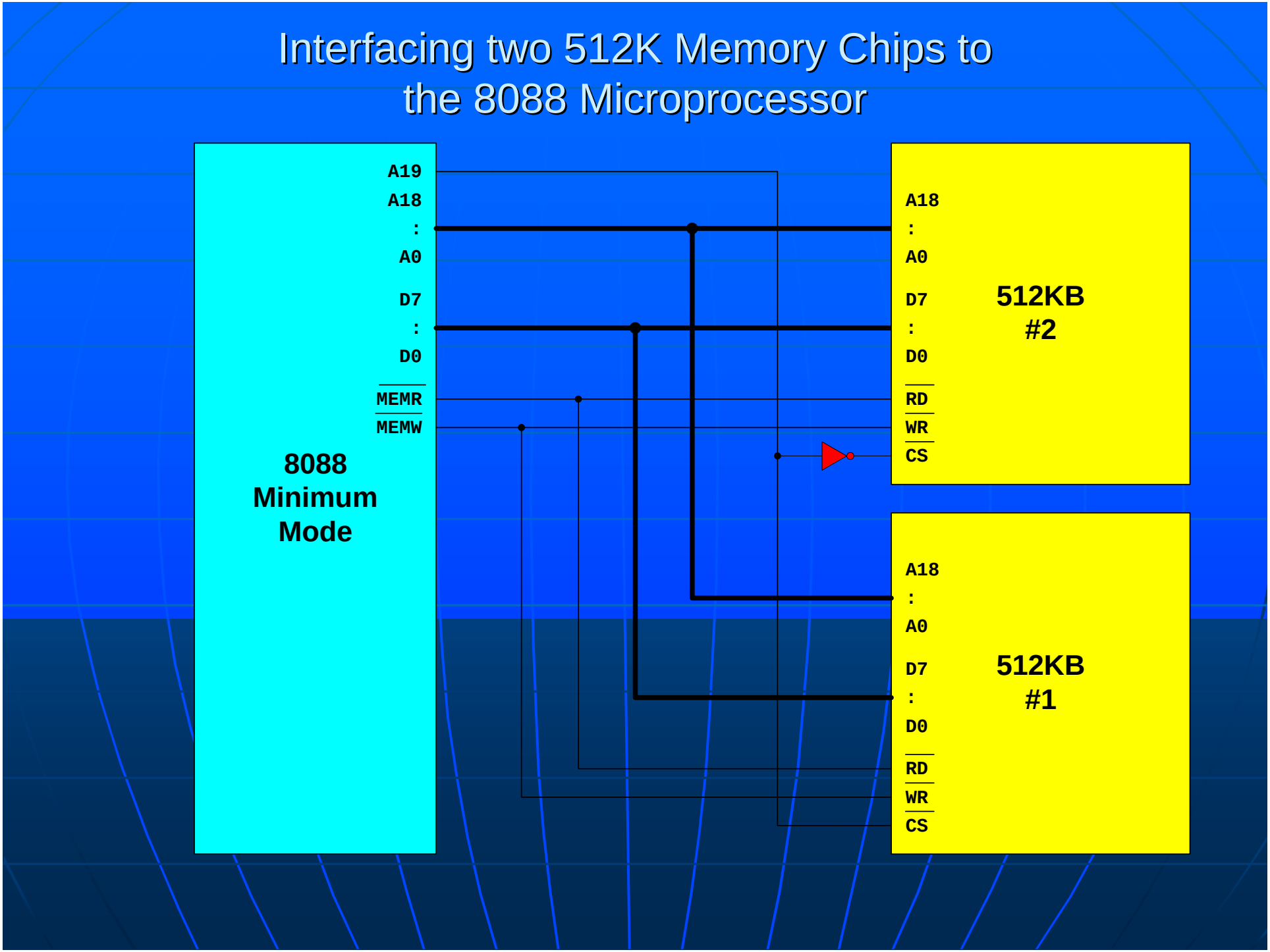
ACTUAL ADDRESS

Partial Decoding

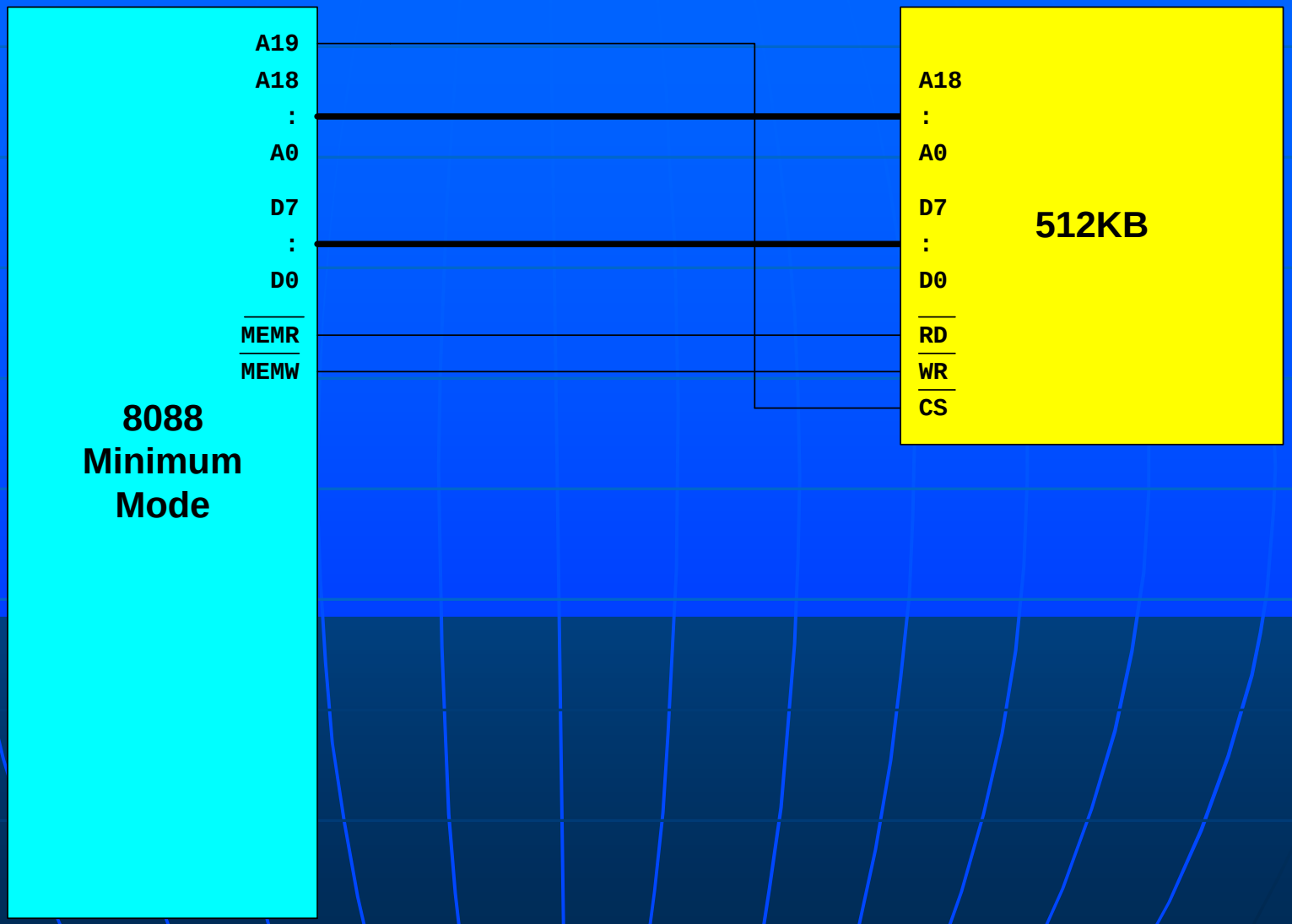
A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 1000	AAAA 7654	AAAA 3210
00000	0000	0000	0000	0000	0000
7FFFF	0111	1111	1111	1111	1111
80000	1000	0000	0000	0000	0000
FFFFF	1111	1111	1111	1111	1111

ACTUAL ADDRESS

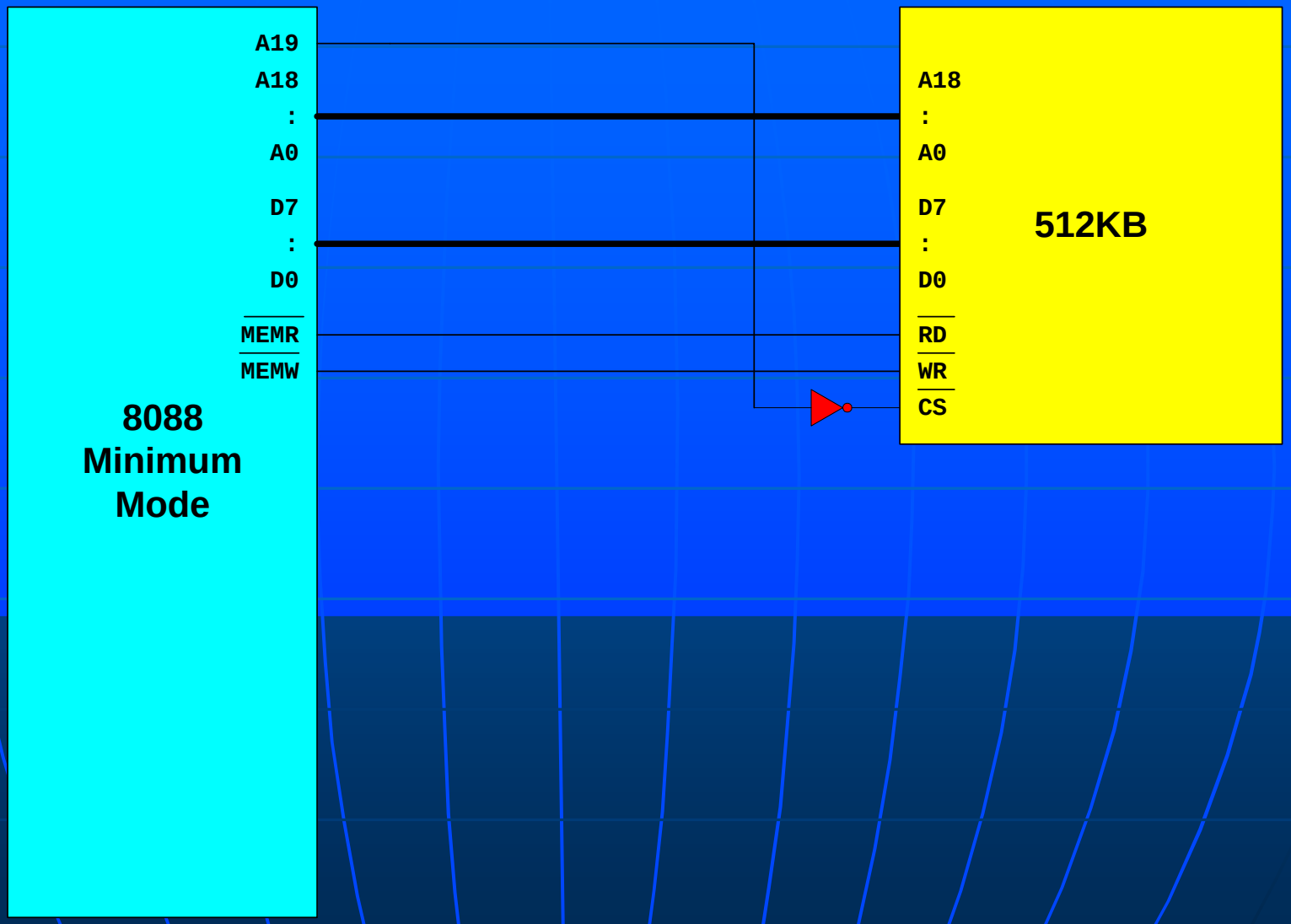
Interfacing two 512K Memory Chips to the 8088 Microprocessor



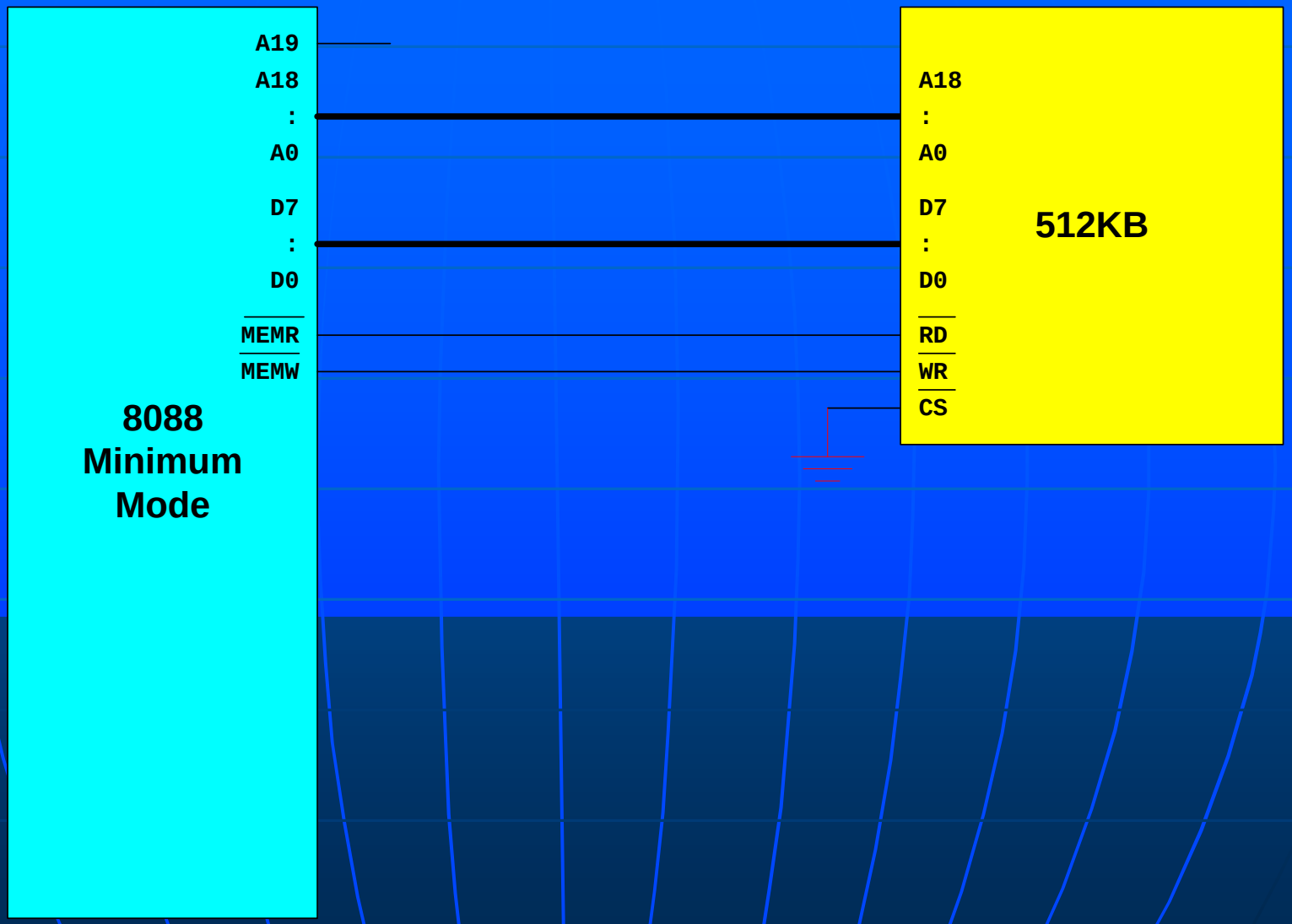
Interfacing one 512K Memory Chips to the 8088 Microprocessor



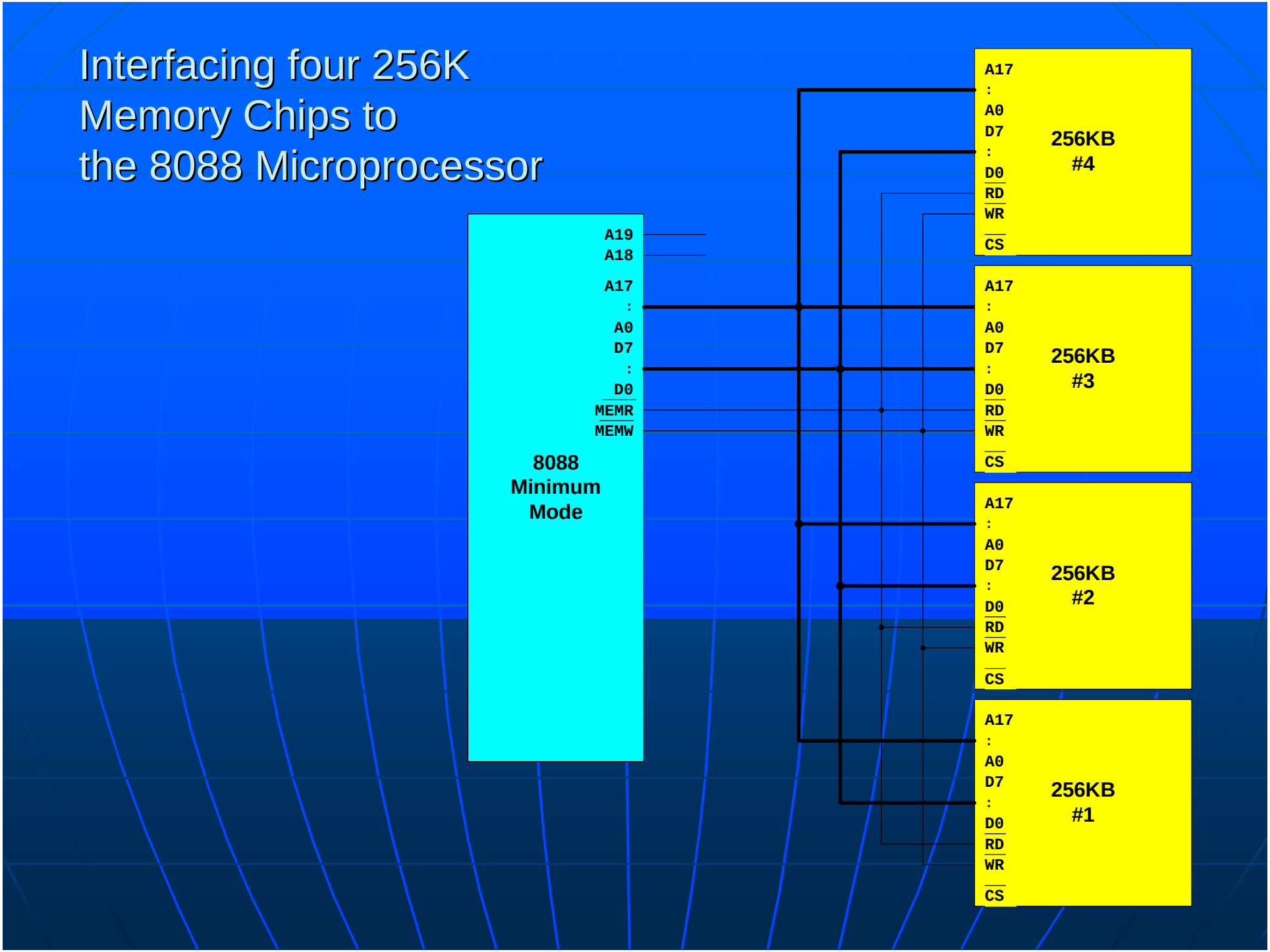
Interfacing one 512K Memory Chips to the 8088 Microprocessor (version 2)



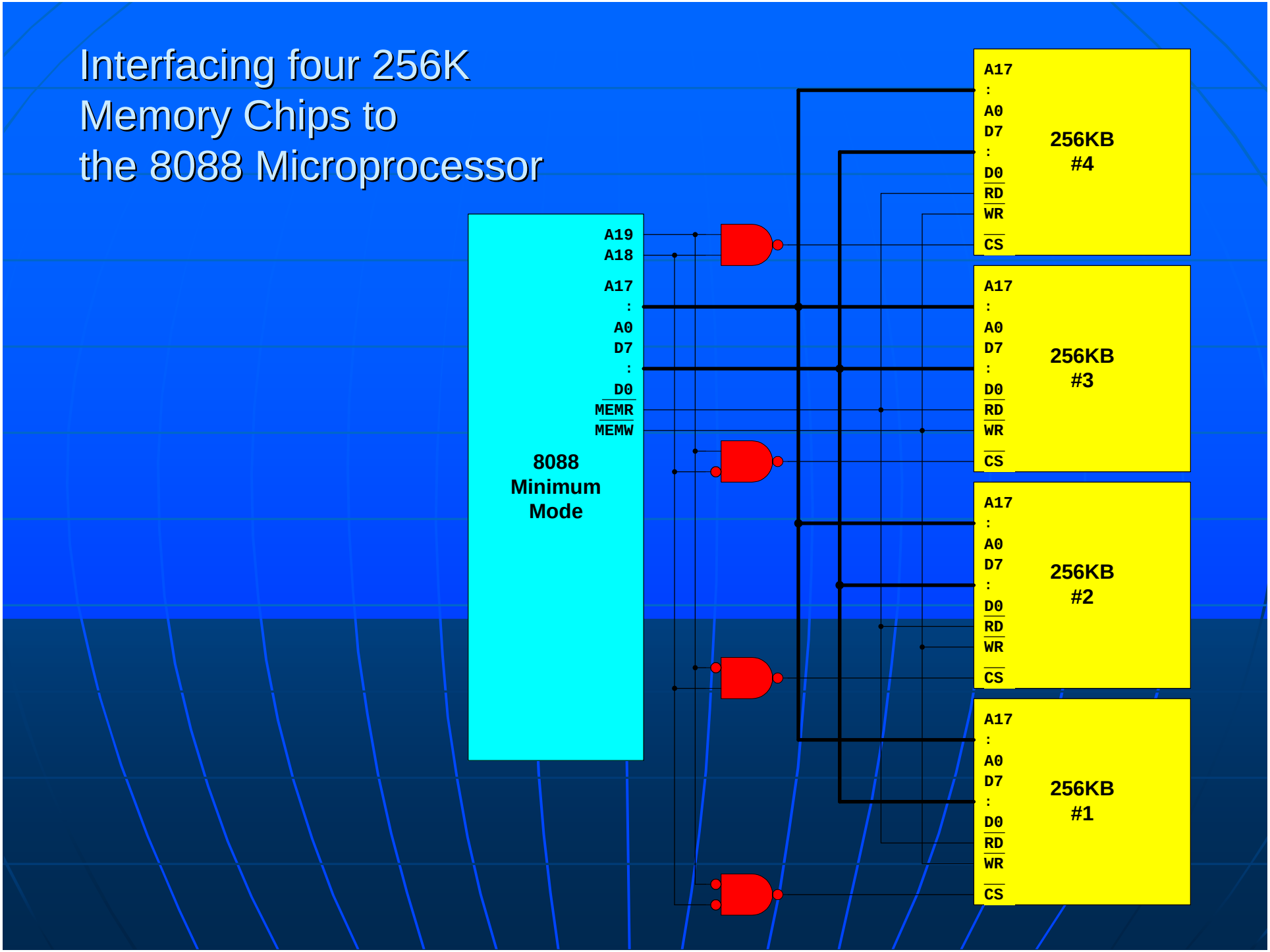
Interfacing one 512K Memory Chips to the 8088 Microprocessor (version 3)



Interfacing four 256K Memory Chips to the 8088 Microprocessor



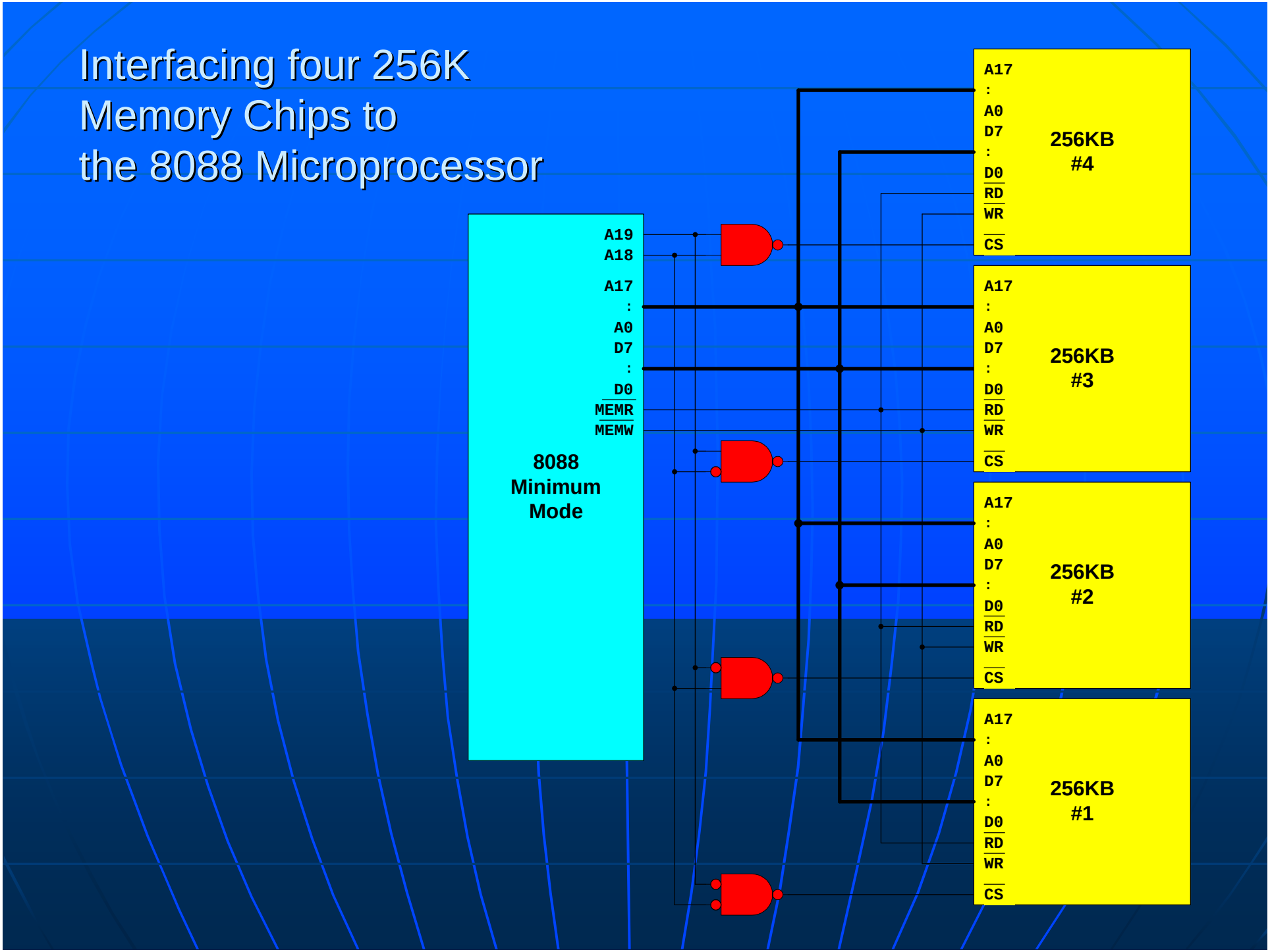
Interfacing four 256K Memory Chips to the 8088 Microprocessor



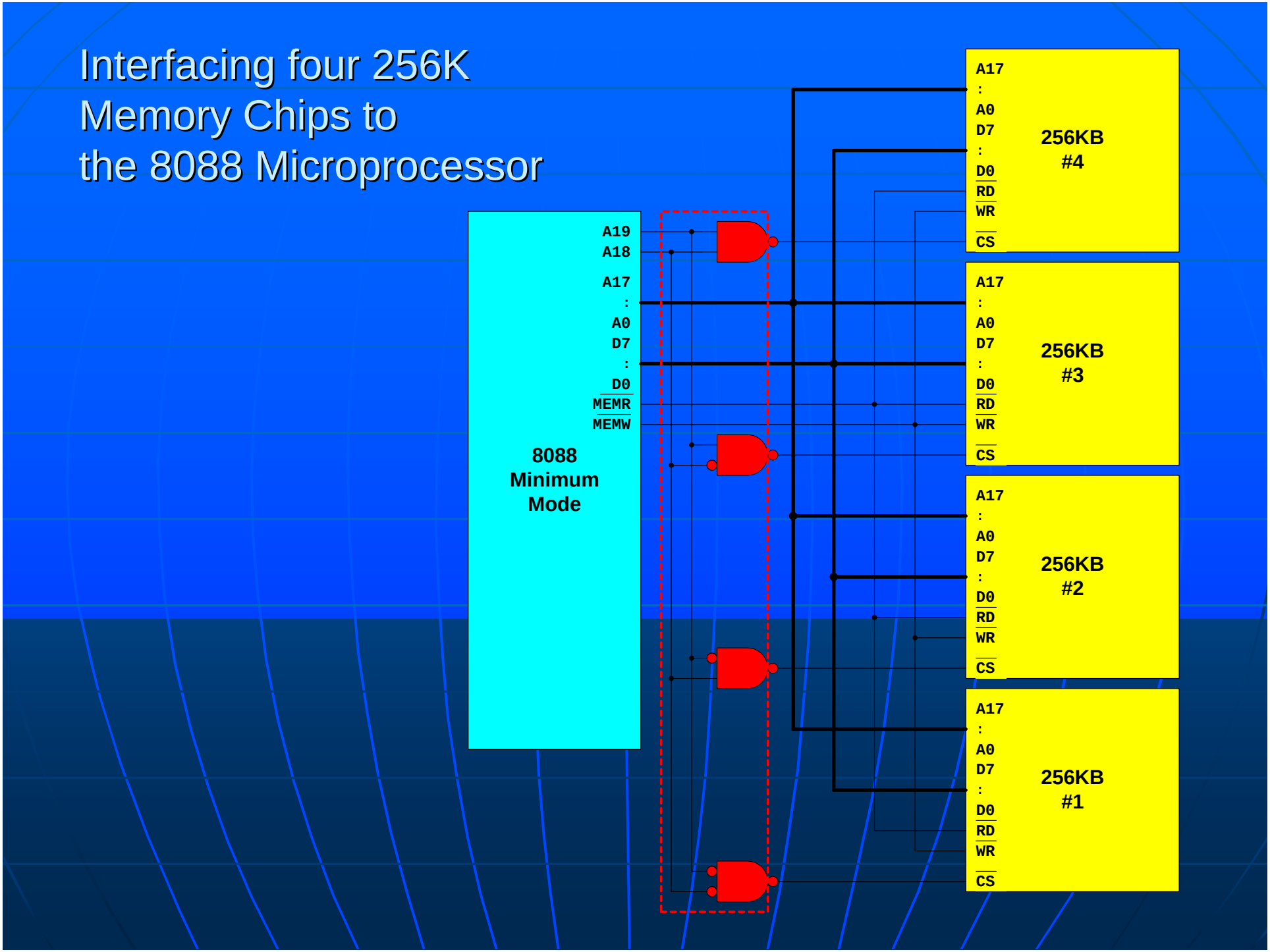
Memory chip#__ is mapped to:

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 10$\phi\phi$	AAAA 7654	AAAA 3210
-----	----	----	----	----	----
-----	----	----	----	----	----

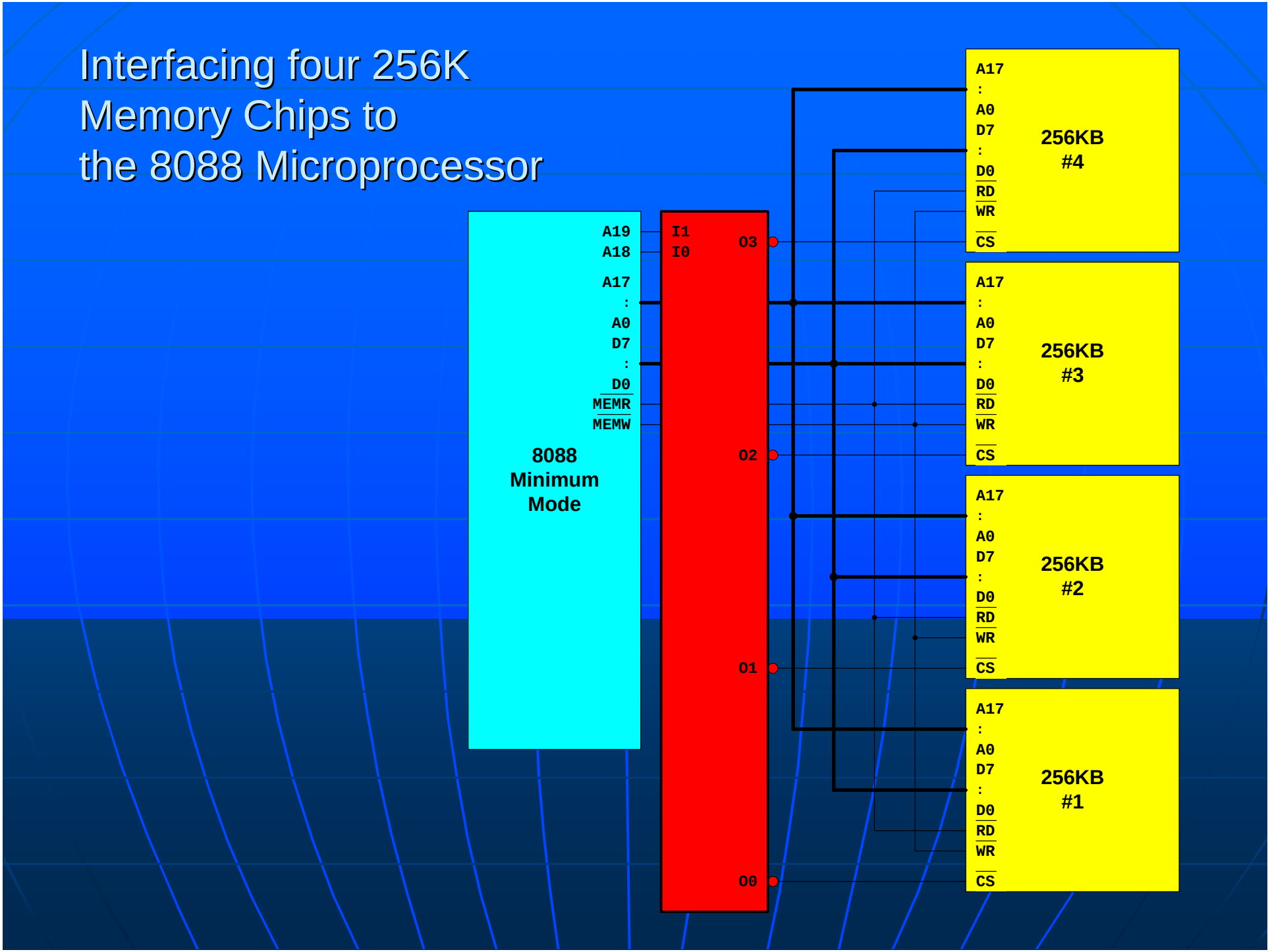
Interfacing four 256K Memory Chips to the 8088 Microprocessor



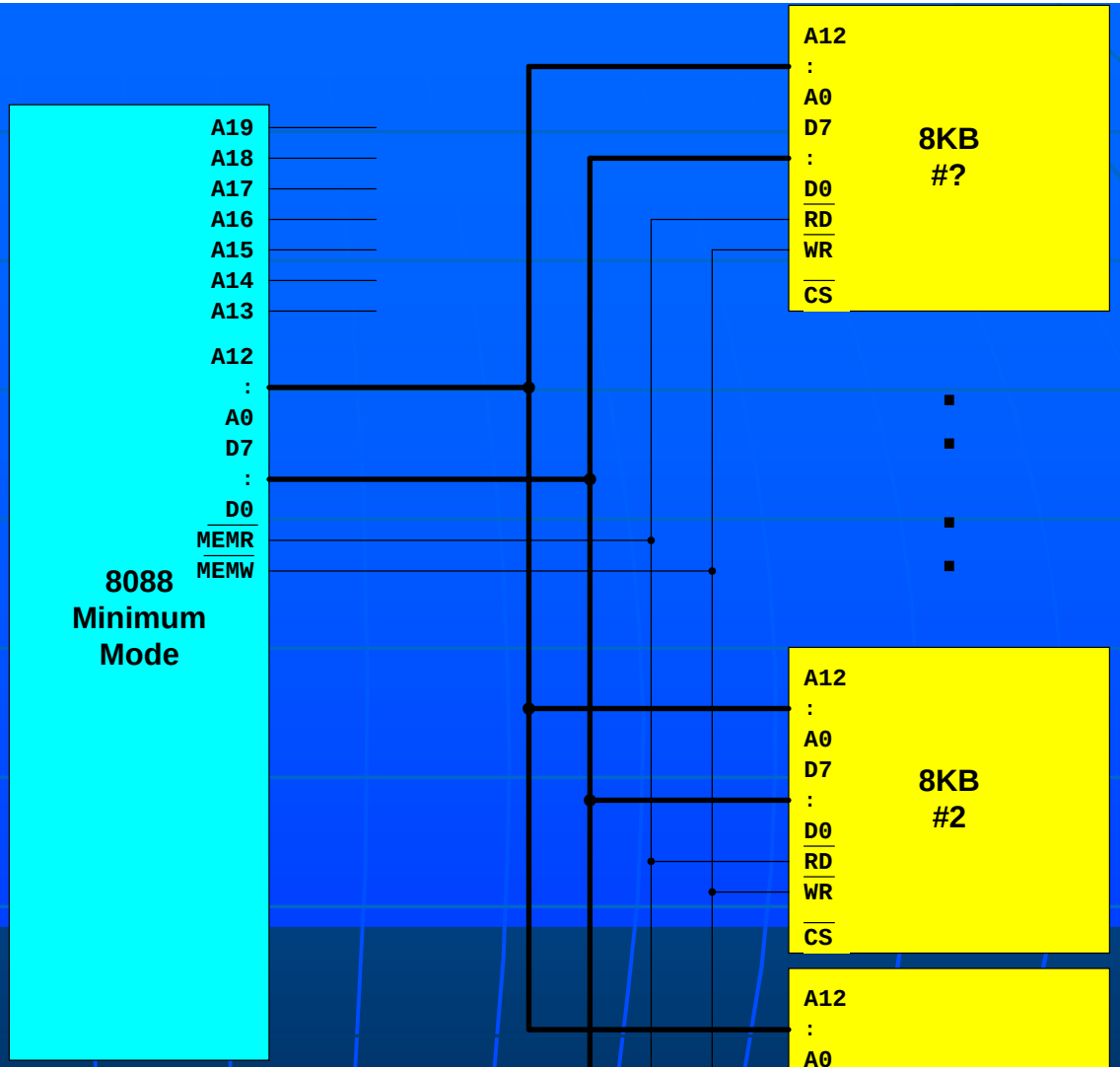
Interfacing four 256K Memory Chips to the 8088 Microprocessor



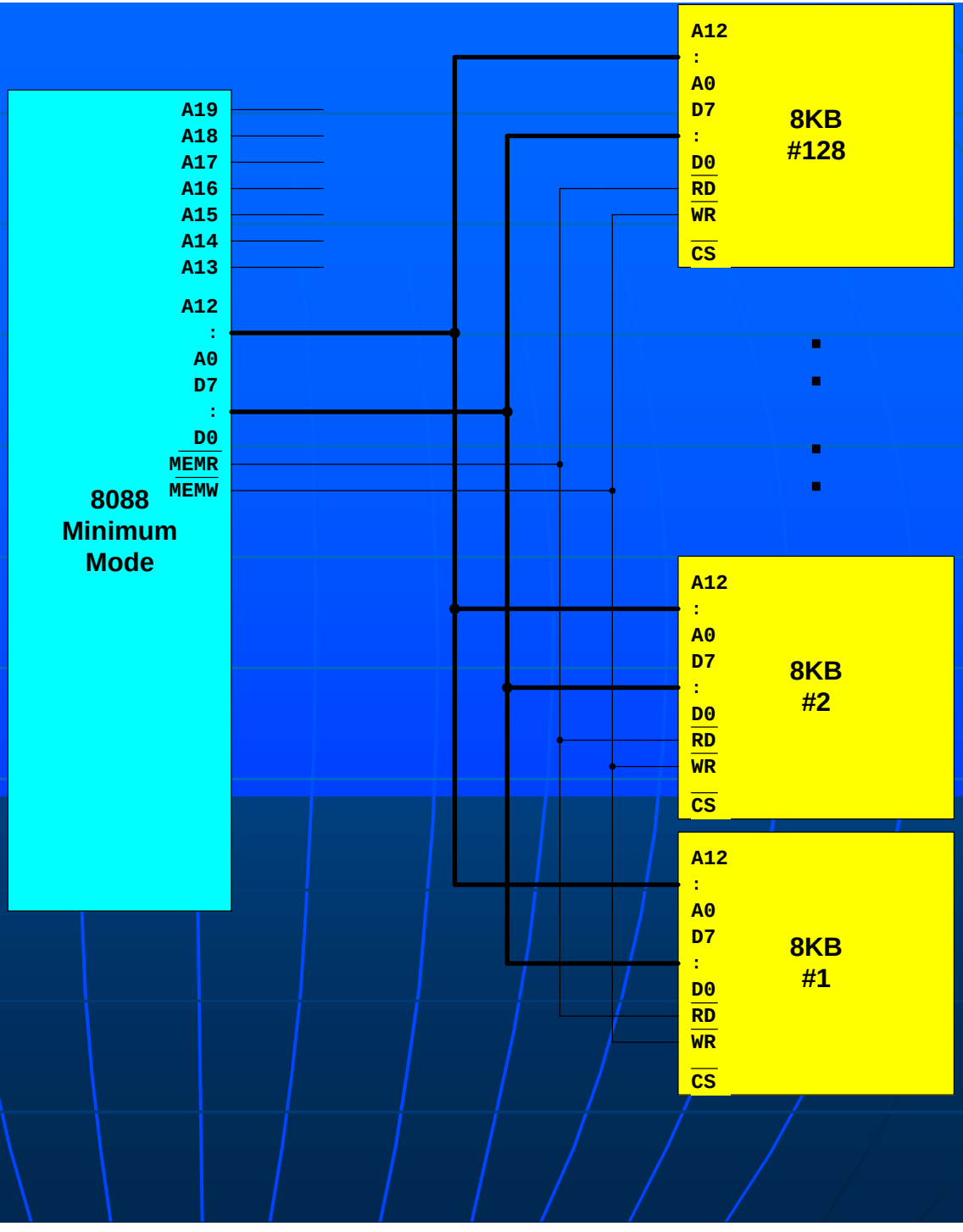
Interfacing four 256K Memory Chips to the 8088 Microprocessor



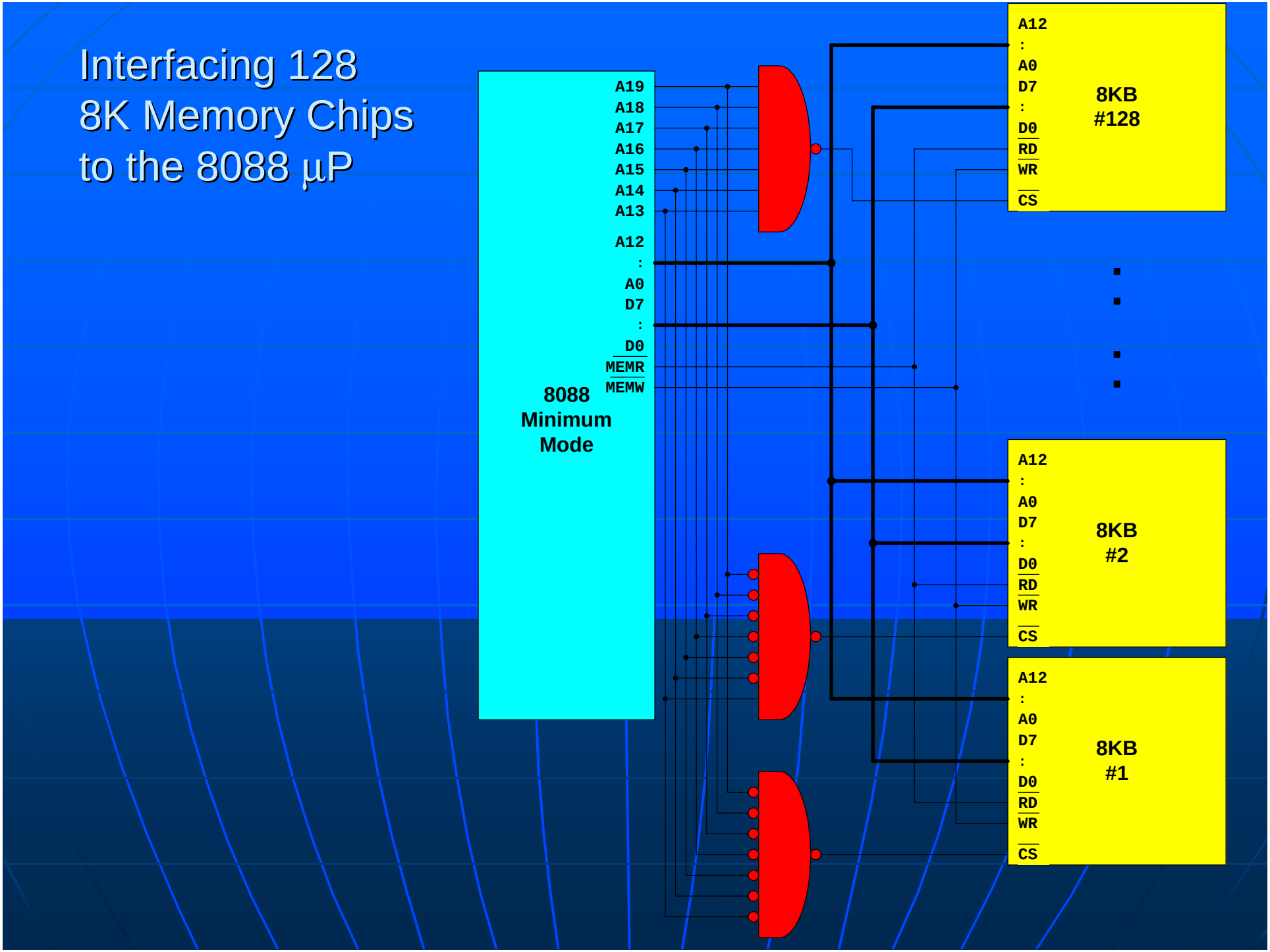
Interfacing several 8K Memory Chips to the 8088 µP



Interfacing 128 8K Memory Chips to the 8088 µP



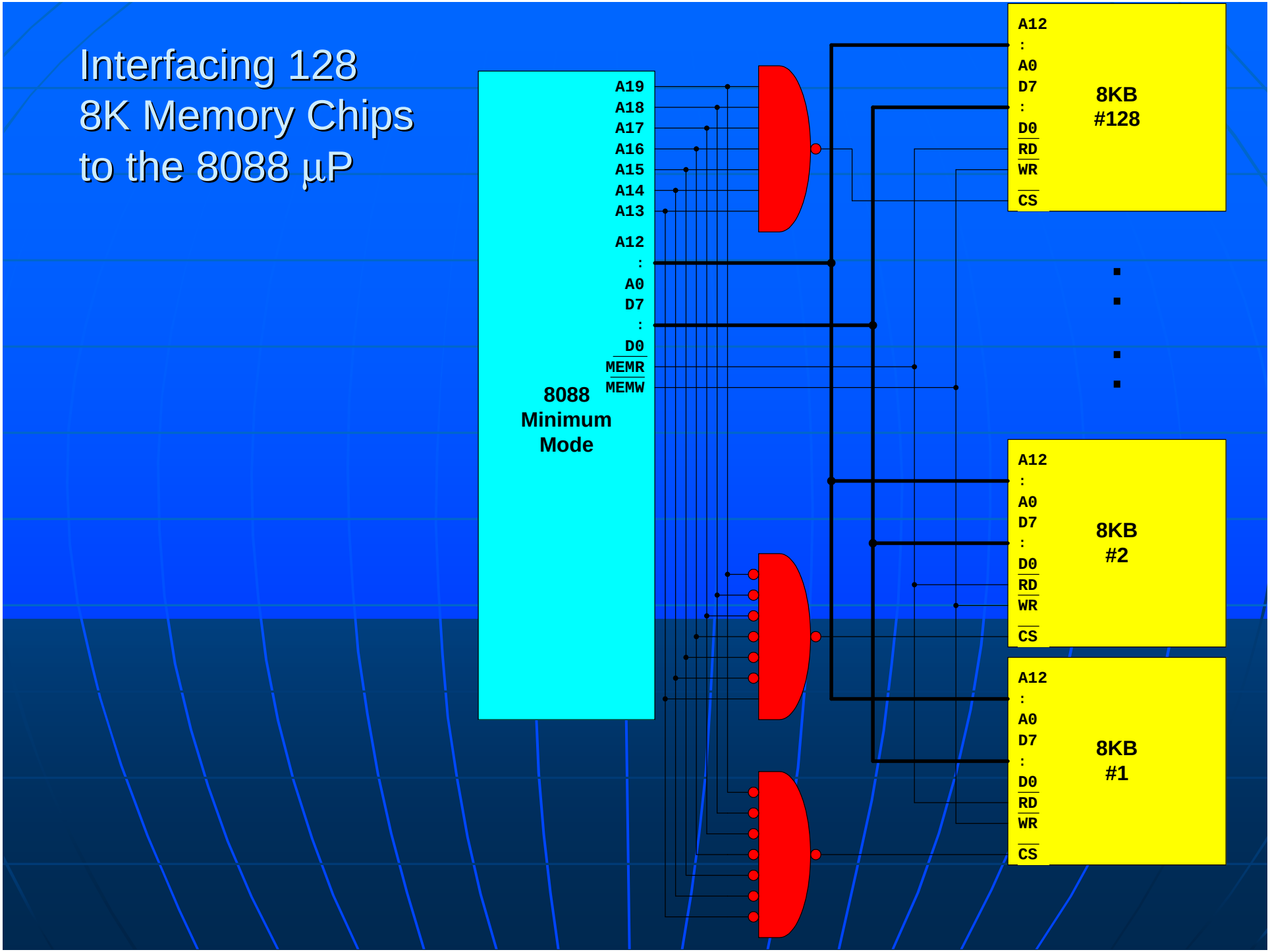
Interfacing 128 8K Memory Chips to the 8088 µP



Memory chip#__ is mapped to:

A19 to A0 (HEX)	AAAA 1111 9876	AAAA 1111 5432	AAAA 1198 10$\phi\phi$	AAAA 7654	AAAA 3210
-----	----	----	----	----	----
-----	----	----	----	----	----

Interfacing 128 8K Memory Chips to the 8088 µP



VI XỬ LÝ

Khoa: Điện – Điện Tử

Bộ môn: Kỹ Thuật Máy Tính

Giảng viên: Trần Thiên Thanh

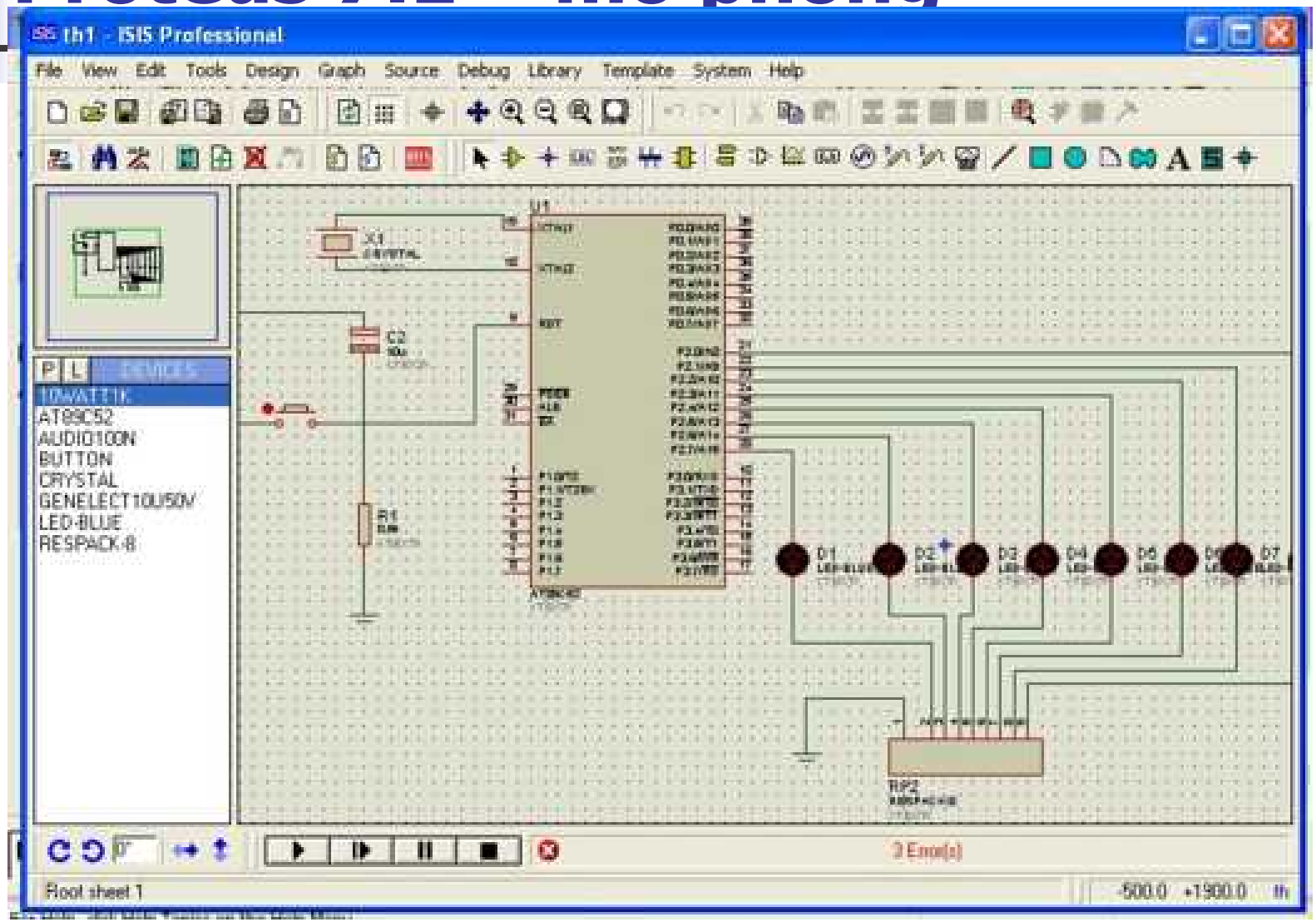
THÔNG TIN CHUNG MÔN HỌC

- Thời gian: 15 tuần – 60 tiết
  - Lý Thuyết: 45 tiết – 11 tuần
  - Bài tập-thực hành: 15 tiết – 03 tuần
- Điểm thi
  - Chuyên cần: 10%
  - Giữa kỳ: hết chương 3 – 10%
  - Bài tập lớn/Thực hành: 10% - hết chương 4
  - Thảo luận/bài tập: 10%
  - Cuối kỳ: 60% - vấn đáp

CHƯƠNG1: GIỚI THIỆU VI XỬ LÝ

- Thảo luận, báo cáo (5%)
  - Phân nhóm
  - Đề tài thảo luận
  - Báo cáo cuối buổi
- Bài tập tại lớp (5%)
- Bài tập lớn (10%)

Proteus 7.1 – mô phỏng



NỘI DUNG

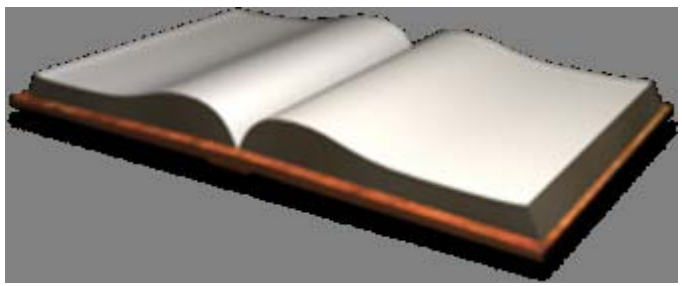
- Yêu cầu:
  - Hiểu về môn học và vai trò
  - Nắm các yêu cầu để học tốt
- Cách học tốt môn này
  - Nắm vững lý thuyết
  - Học thuộc tập lệnh 8051
  - Đọc tham khảo các chương trình ví dụ
  - Viết chương trình nhiều với các ứng dụng thực tiễn, dùng phần mềm mô phỏng

NỘI DUNG

- Giáo trình chính:
  - http://www.box.net/shared/ljtd2lzn65
- Sách tham khảo:
  - "The 8051 – Microcontroller" – I.Scott Mackenzie
  - "Họ vi điều khiển 8051" – Tống Văn On

MỤC LỤC

- CHƯƠNG 1: Giới thiệu vi xử lý
- CHƯƠNG 2: Phần cứng họ MCS-51
- CHƯƠNG 3: Lập trình hợp ngữ họ MCS-51
- CHƯƠNG 4: Các chức năng của họ vi điều khiển MCS-51
- CHƯƠNG 5: Giao tiếp



CHƯƠNG1: GIỚI THIỆU VI XỬ LÝ

- Mục tiêu (Tuần 1)
  - Hiểu và giải thích được cấu trúc chung và hoạt động của một hệ thống VXL. Vai trò các Bus
  - Hiểu chức năng các khối của VXL
  - Phân loại bộ nhớ
  - SV biết các thảo luận, báo cáo

CHƯƠNG1: GIỚI THIỆU VI XỬ LÝ

- I – Tổng quan hệ thống vi xử lý
- II – Các loại bus
- III – Vi xử lý
- IV – Bộ nhớ
- V – Nhập xuất (I/O)
- VI – Vi xử lý – Vi điều khiển
- (Tập lệnh 8051)

CHƯƠNG1: GIỚI THIỆU VI XỬ LÝ

- I – Tổng quan hệ thống vi xử lý
- II – Các loại bus
- III – Vi xử lý
- IV – Bộ nhớ
- V – Nhập xuất (I/O)
- VI – Vi xử lý – Vi điều khiển
- (Tập lệnh 8051)

Ch1: I - Tổng quan hệ thống VXL

- 1. Quá trình phát triển của máy vi tính
- 2. Ứng dụng của vi xử lý
- 3. Sơ đồ khối của hệ vi xử lý

Ch1: I Tổng quan hệ thống VXL

- 1. Quá trình phát triển của máy vi tính
  - 1971 - Intel giới thiệu 8080, là bộ vi xử lý đầu tiên, SDK-85
  - Các hãng khác: Motorola, RCA, MOS Technology, Zilog… giới thiệu 6800, 1801, 6502, Z80, D2, KIM-1, …
  - 1976 – Intel giới thiệu 8748, vi điều khiển thuộc họ MCS-48 → chuẩn công nghiệp
  - 1980 – Intel công bố chip 8051 (Simen: SAB80515

Ch1: I Tổng quan hệ thống VXL

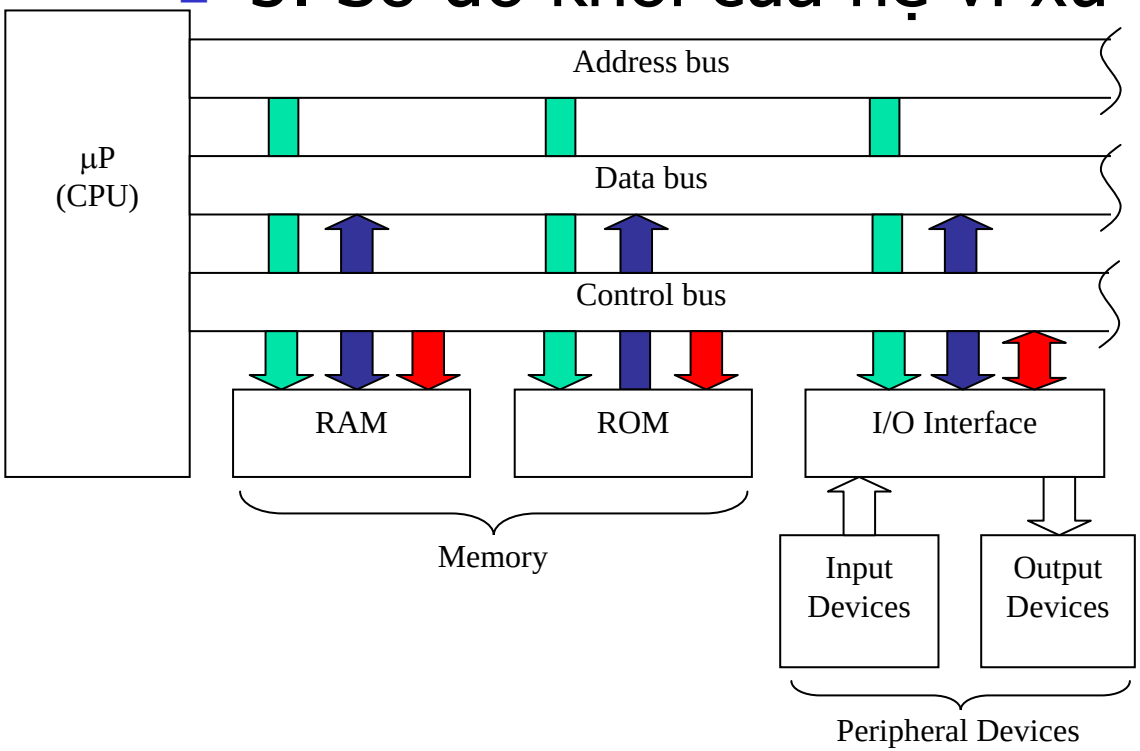
- 2. Ứng dụng của vi xử lý
  - Thay thế các thành phần cơ điện trong các sản phẩm
    - Máy giặt, bộ đèn điều khiển giao thông
  - Xe ô tô, thiết bị công nghiệp, các sản phẩm tiêu dùng
  - Các thiết bị ngoại vi của máy vi tính (thảo luận)

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý

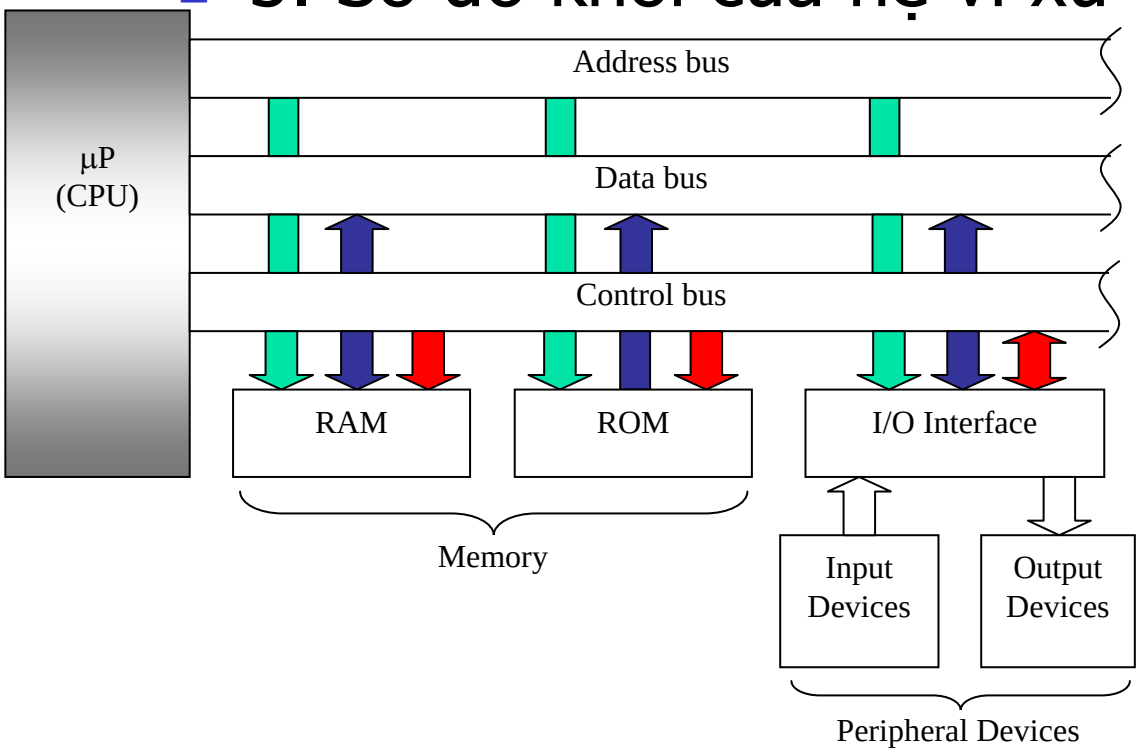


Hình 1.1

μP (Microprocessor): Vi xử lý
CPU (Central Processing Unit): Đơn vị xử lý trung tâm
Address bus: Bus địa chỉ
Data bus: Bus dữ liệu
Control bus: Bus điều khiển
RAM (Random Access Memory): Bộ nhớ truy xuất ngẫu nhiên
ROM (Read-Only Memory): Bộ nhớ chỉ đọc
I/O Interface: Khối giao tiếp nhập/xuất
Peripheral Devices: Thiết bị ngoại vi

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý



Hình 1.1

CPU

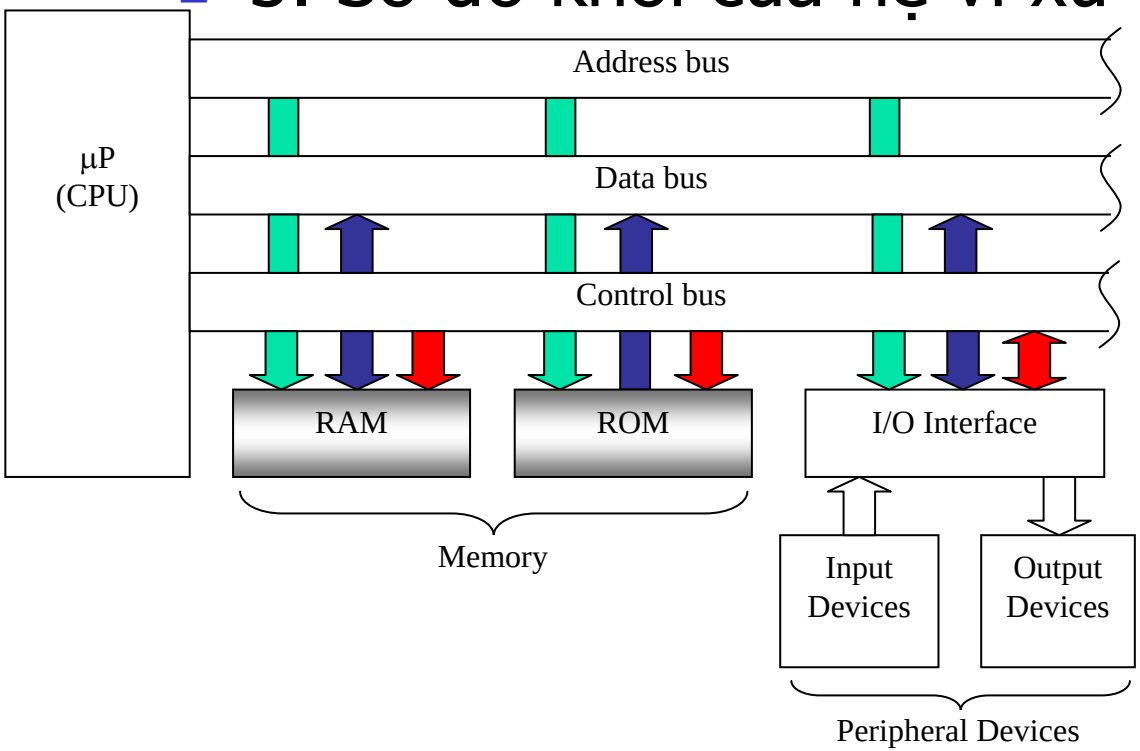
Nguyên tắc làm việc: thực hiện các lệnh liên tục và tuần tự

Mỗi lệnh được biểu diễn bằng mã máy (binary = opcode)

Kết nối với hệ thống bên ngoài thông qua hệ thống bus

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý



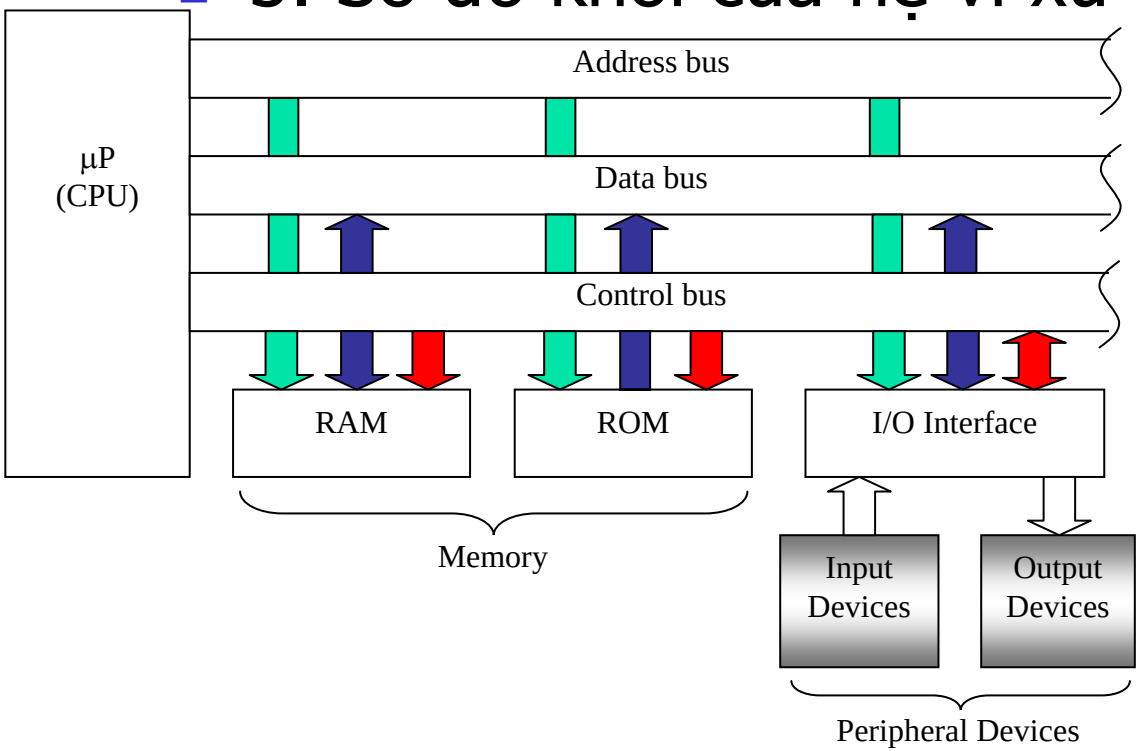
Hình 1.1

Bộ nhớ

Được phân chia theo chức năng: bộ nhớ chương trình: chứa mã lệnh (mã máy) và bộ nhớ dữ liệu: chứa dữ liệu để xử lý khi CPU thực hiện lệnh

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý



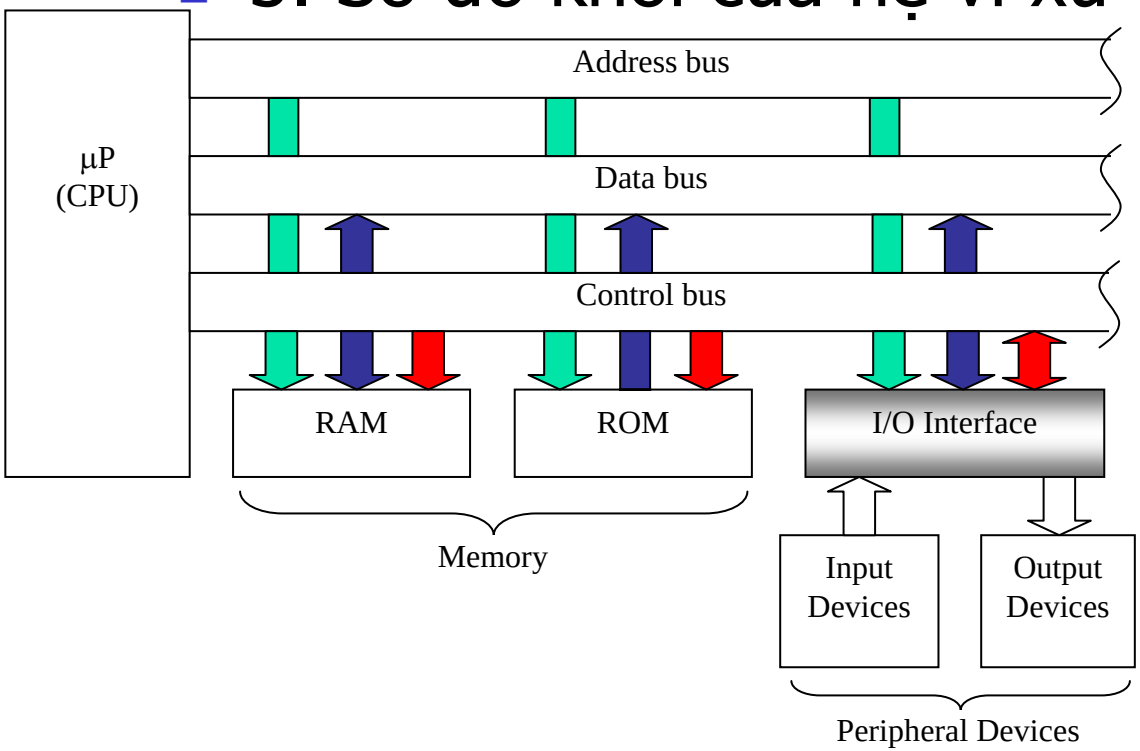
Hình 1.1

Ngoại vi

Thực chất là những cổng vào/ra để CPU giao tiếp với các thiết bị bên ngoài

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý



Hình 1.1

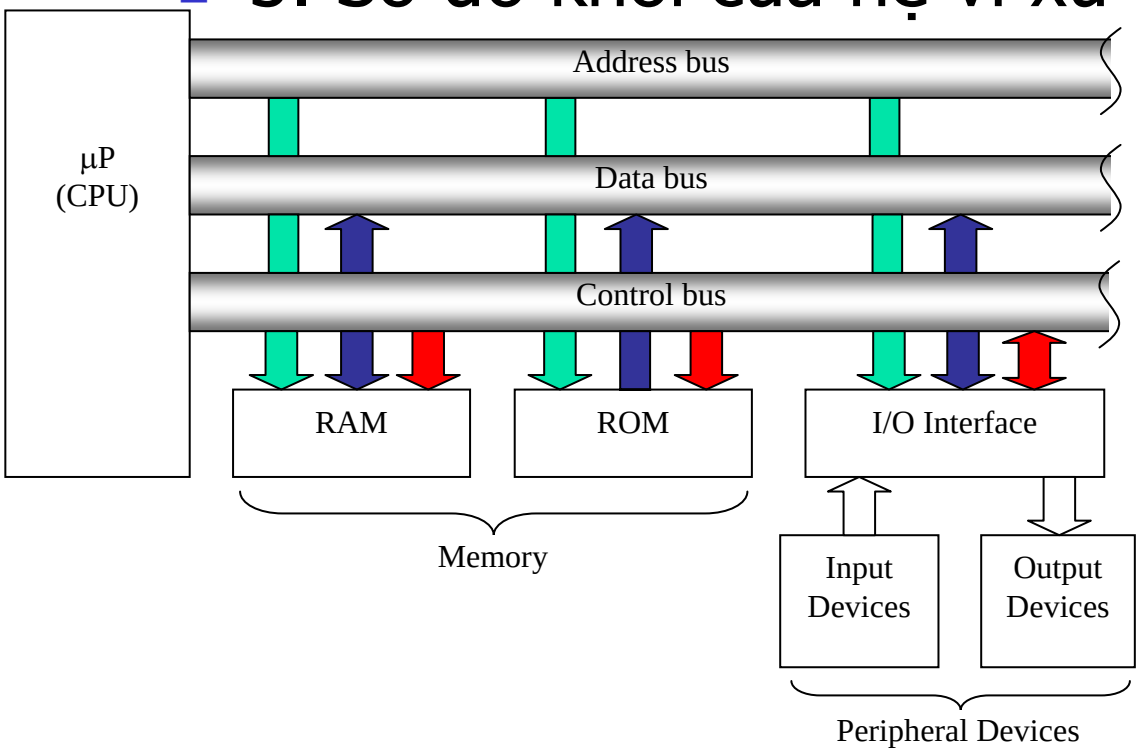
Giải mã địa chỉ

Bộ nhớ, ngoại vi kết nối chung bus, để tiết kiệm dây dẫn. Để trách hiện tượng xung đột logic thì bộ nhớ và I/O hoạt động ở 3 trạng thái (1,0,hi-Z)

Khi bộ nhớ hay I/O được kết nối vào bus data thì phần còn lại ở trạng thái hi-Z

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý



Hình 1.1

Hệ thống bus

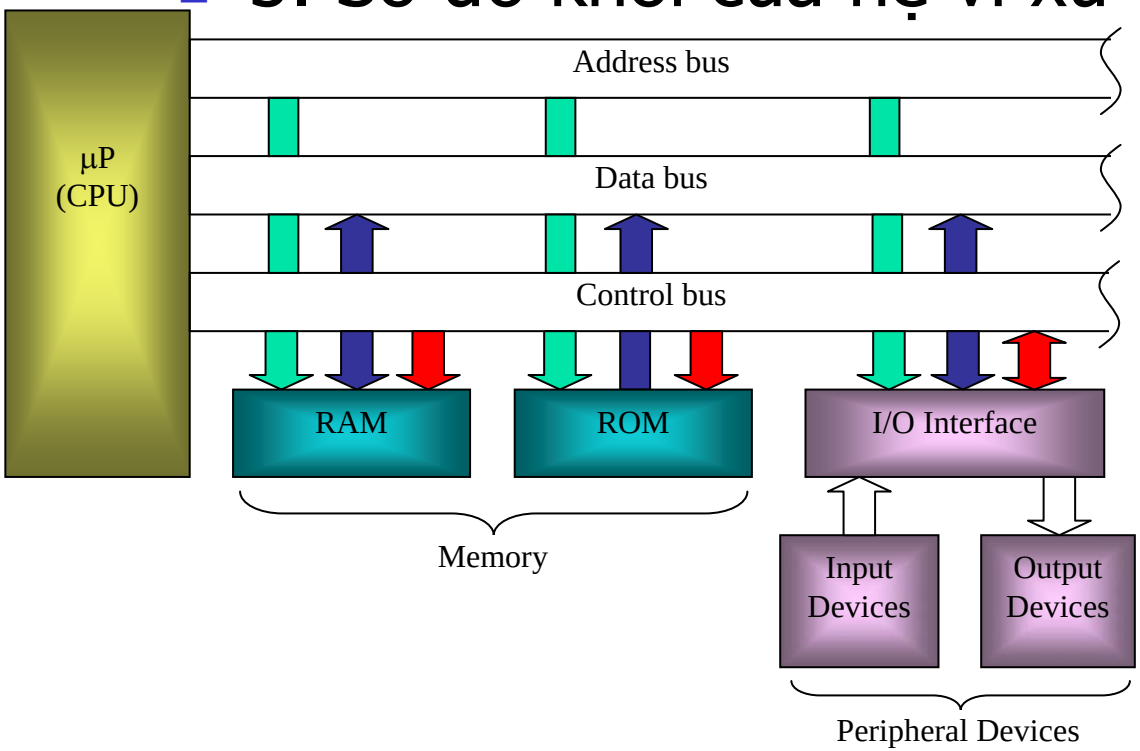
Bus địa chỉ: chứa định vị địa chỉ (được CPU xuất ra)

Bus data: tại 1 thời điểm, CPU chỉ giao tiếp được với 1 đơn vị bộ nhớ hoặc I/O (2chiều)

Bus điều khiển: gồm các tín hiệu đồng bộ hoạt động

Ch1: I Tổng quan hệ thống VXL

- 3. Sơ đồ khối của hệ vi xử lý



Hình 1.1

Ba khối chính

1. Bộ nhớ
2. CPU:
 - Đọc/ghi vào bộ nhớ
 - Đọc từ đầu vào
 - Ghi ra đầu ra
 - Thực hiện lệnh nội bộ : số học và logic
3. Phối ghép (giao tiếp) vào ra I/O

Không có đường trực tiếp từ 1 sang 3.

Ch1: I Tổng quan hệ thống VXL

- 1.1 Hãy nêu các thành phần cơ bản trong một hệ vi xử lý? Chức năng của từng phần

Ch1: II Các loại bus

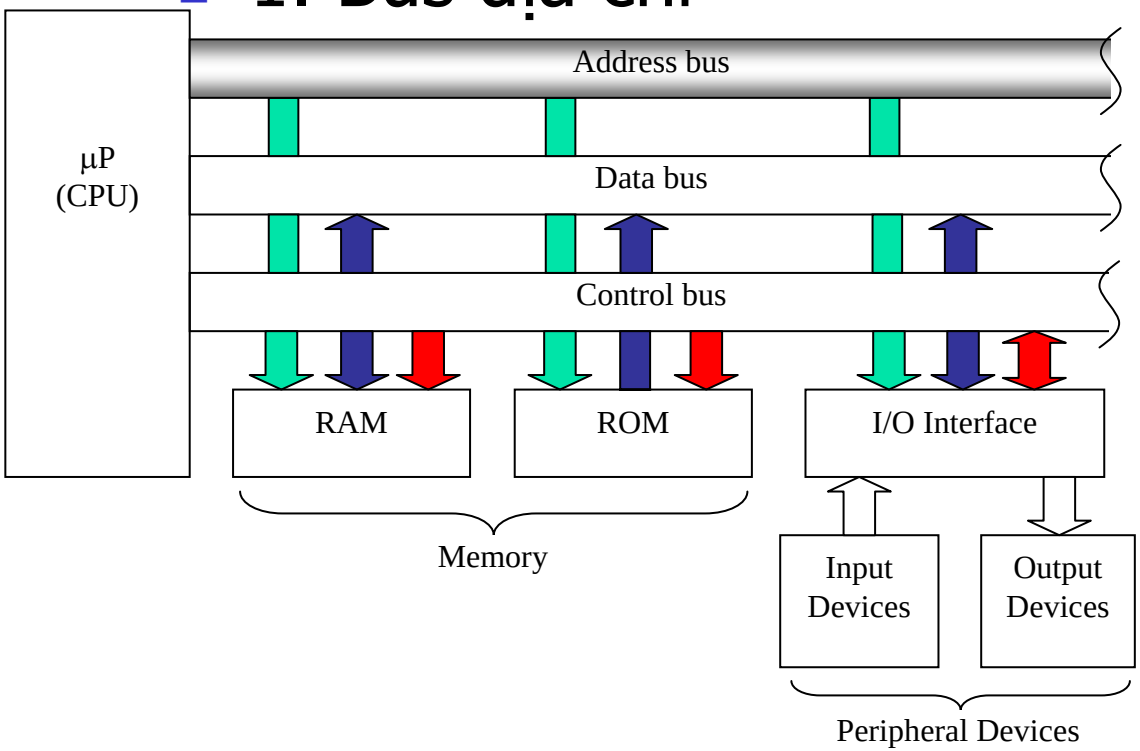
- 1. Bus địa chỉ
  - Đệm bus địa chỉ
- 2. Bus dữ liệu
  - Đệm bus dữ liệu
- Bus điều khiển

Ch1: II Các loại bus

- 1. Bus địa chỉ
  - Đệm bus địa chỉ
- 2. Bus dữ liệu
  - Đệm bus dữ liệu
- Bus điều khiển

Ch1: II Các loại bus

- 1. Bus địa chỉ



Hình 1.1

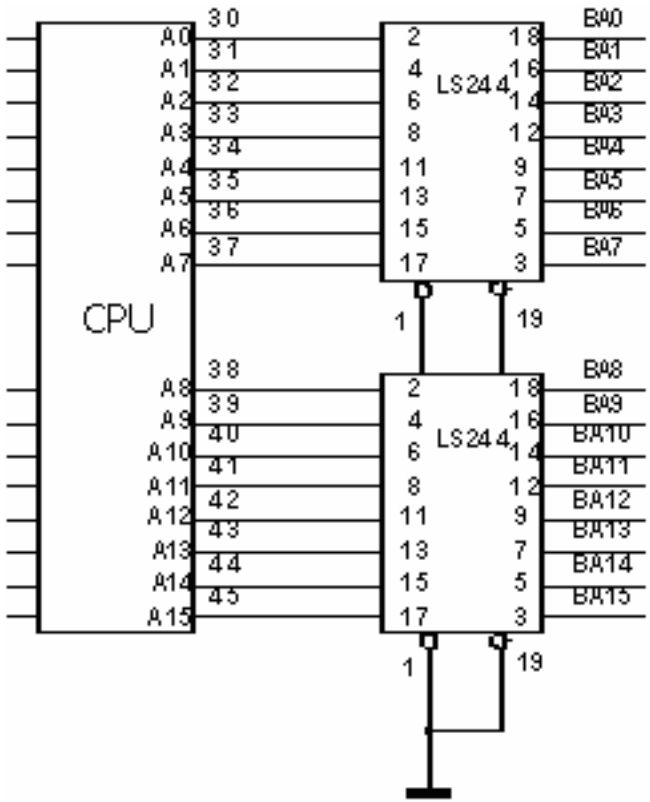
Nội dung: thông tin địa chỉ cần truy xuất (ngăn nhớ hoặc thiết bị)

Số lượng địa chỉ µP quản lý phụ thuộc số đường dây (16,20,24,32)

Bus một chiều đi từ **µP**
N đường dây → 2^N địa chỉ
8051 → N =16

Ch1: II Các loại bus

- 1. (Đệm bus địa chỉ)



Kết nối vật lý dẫn đến quá dòng:

Không hoạt động

Hoạt động không ổn định

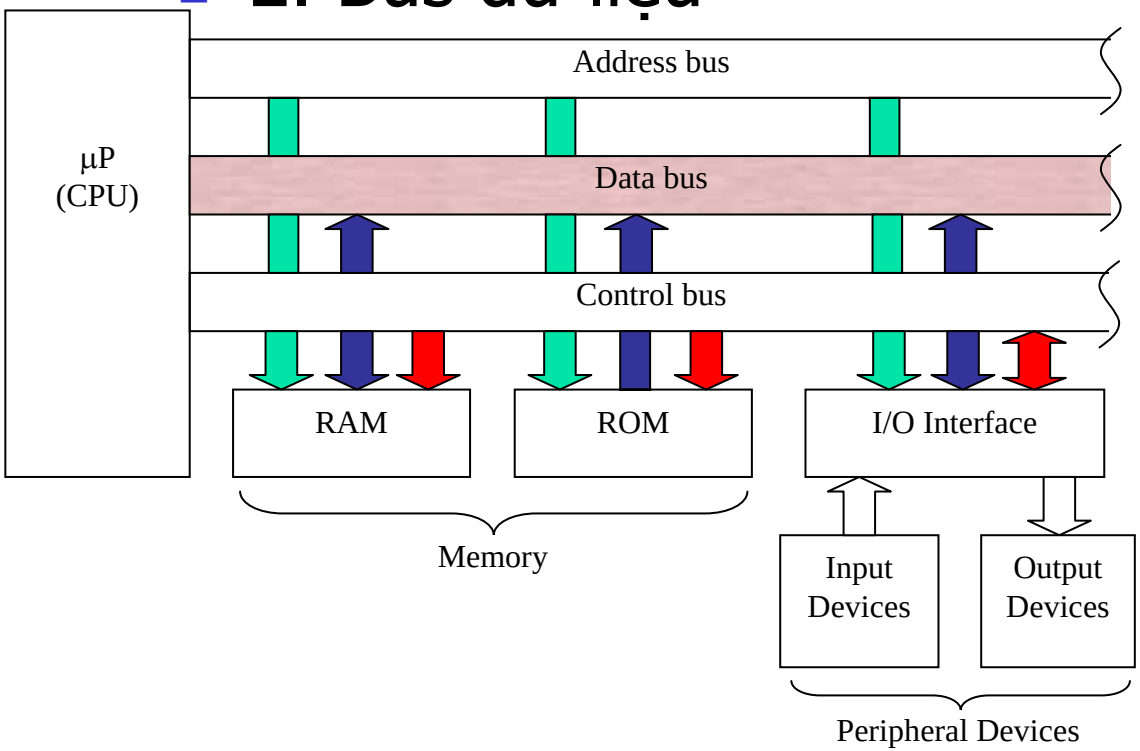
→ Dùng bộ đệm địa chỉ

Ch1: II Các loại bus

- 1. Bus địa chỉ
  - 1.9 Có bao nhiêu vị trí bộ nhớ có thể được định địa chỉ bởi một µP có 20 đường địa chỉ?

Ch1: II Các loại bus

- 2. Bus dữ liệu



Hình 1.1

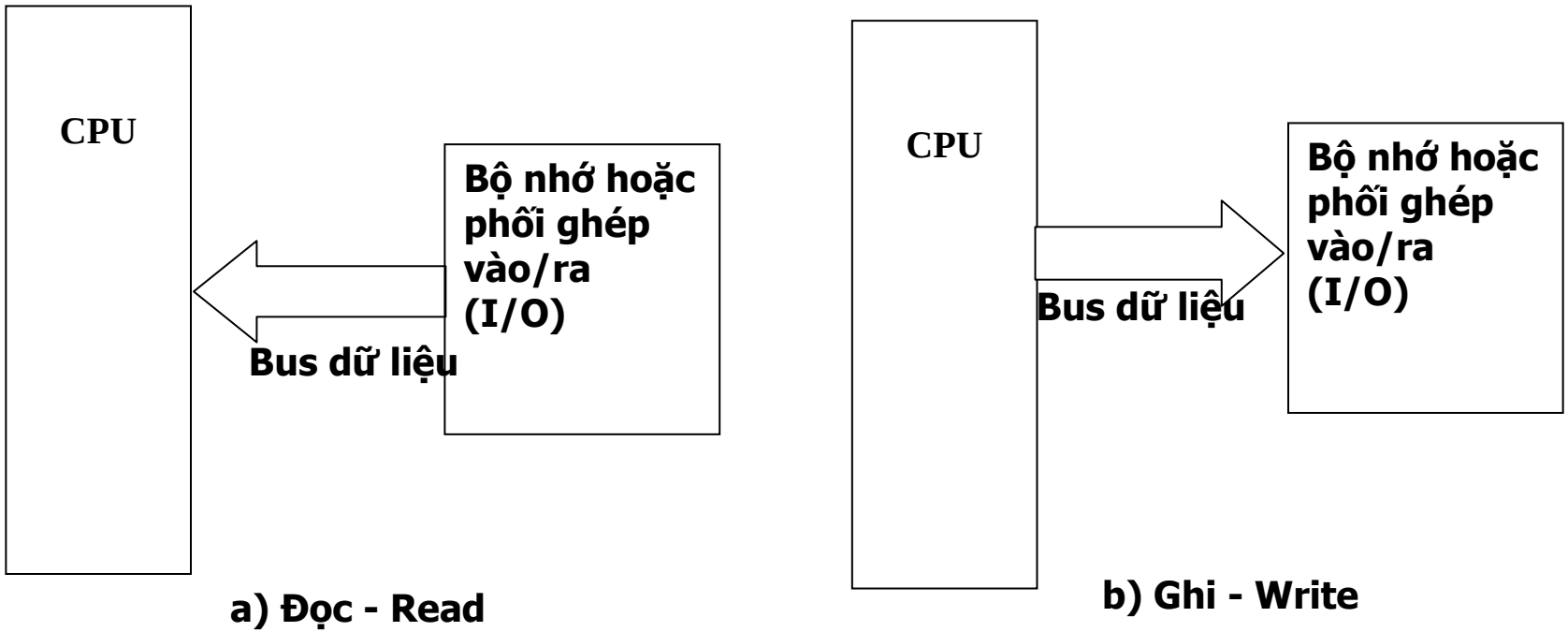
Nội dung: thông tin dữ liệu đến/từ **µP**

Số lượng đường dây quyết định số bit dữ liệu mà µP có khả năng quản lý cùng một lúc (8,16,32, 64 … bit)

Bus hai chiều

Ch1: II Các loại bus

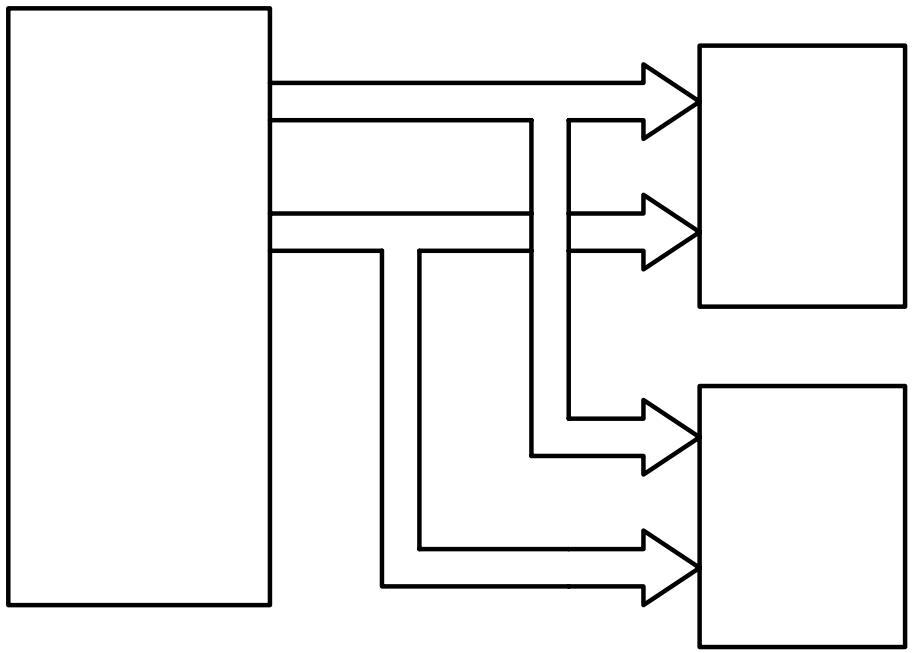
- 2. Bus dữ liệu



Tại 1 thời điểm, dữ liệu chỉ truyền theo 1 hướng

Ch1: II Các loại bus

- 2. Bus dữ liệu

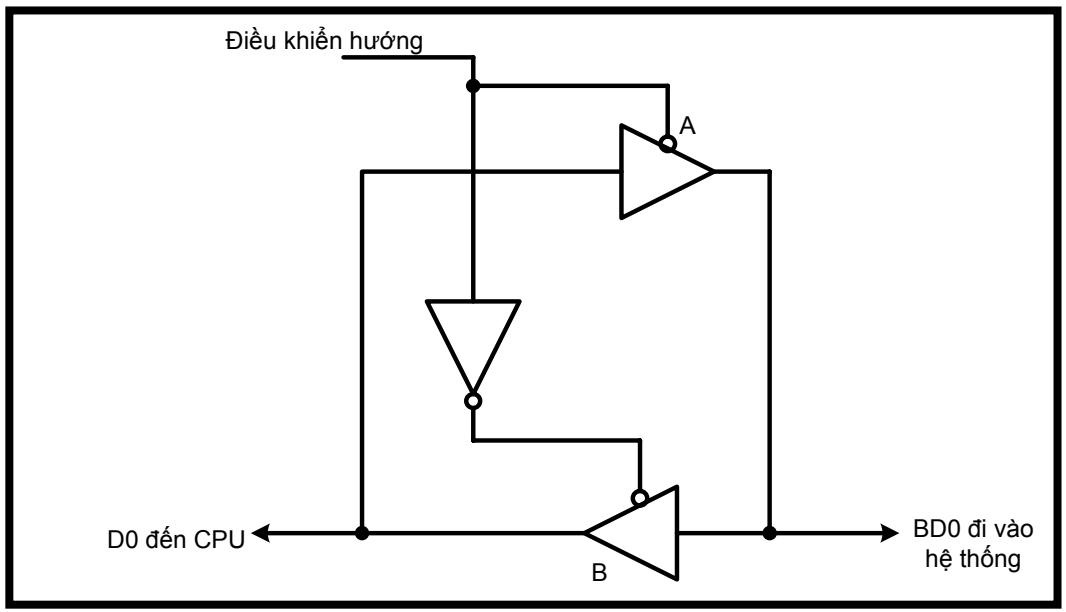


Bus địa chỉ hoạt động độc lập với bus dữ liệu

Address

Ch1: II Các loại bus

- 2. Bus dữ liệu



Kỹ thuật đệm 2 chiều sử dụng thêm một tín hiệu điều khiển, tín hiệu này sẽ quy định chiều dữ liệu sẽ được đệm.

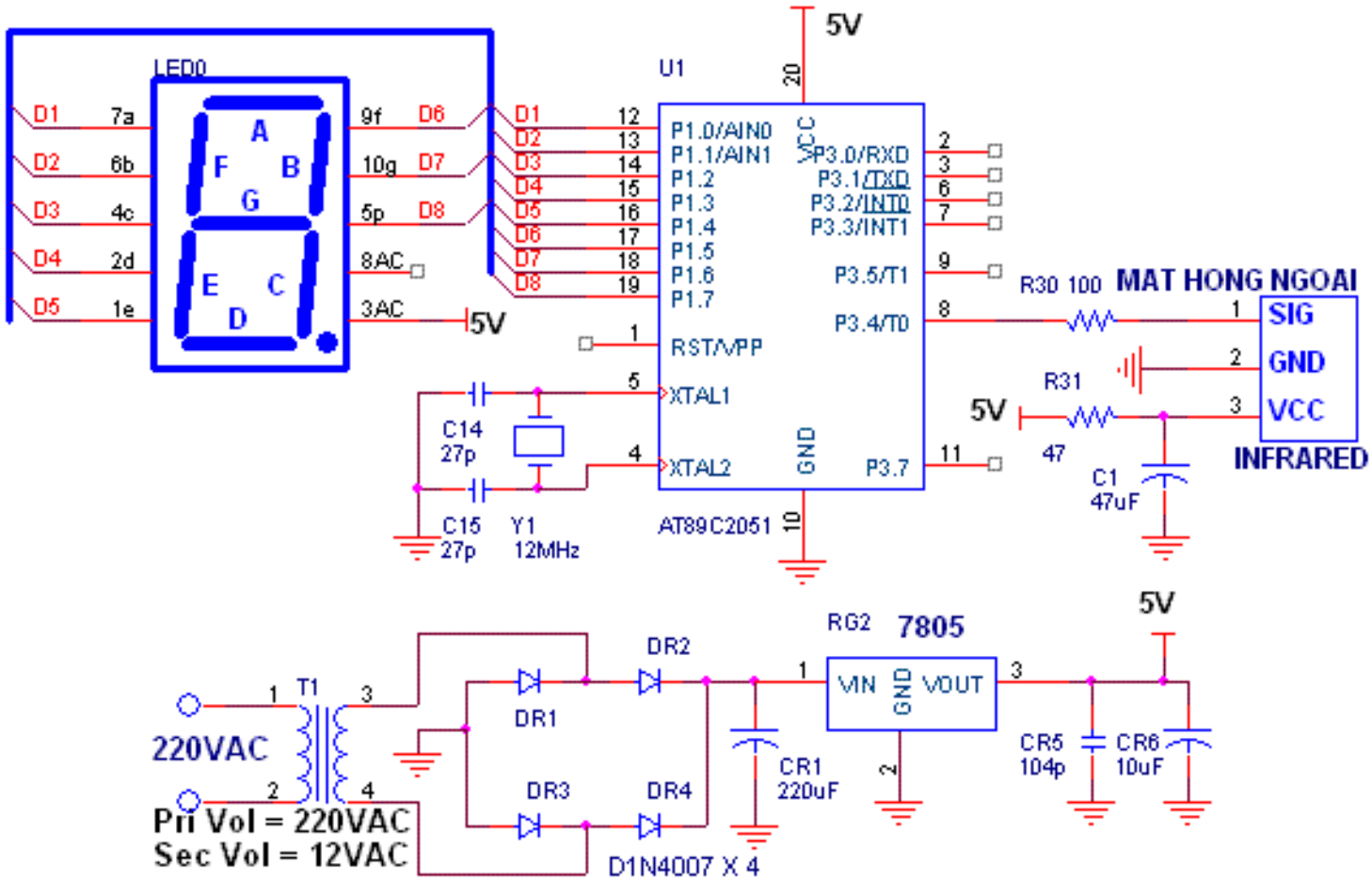
Ch1: II Các loại bus

- 2. Bus dữ liệu
  - 1.10 Nếu một chip bộ nhớ có kích thước là 1024x4bit thì phải cần bao nhiêu chip như vậy để tạo ra 2k(2048) bye bộ nhớ?
  - 1.11 Nếu một chip bộ nhớ có kích thước là 256x1bit thì phải cần bao nhiêu chip như vậy để tạo ra 1KB bộ nhớ?

Ch1: II Các loại bus

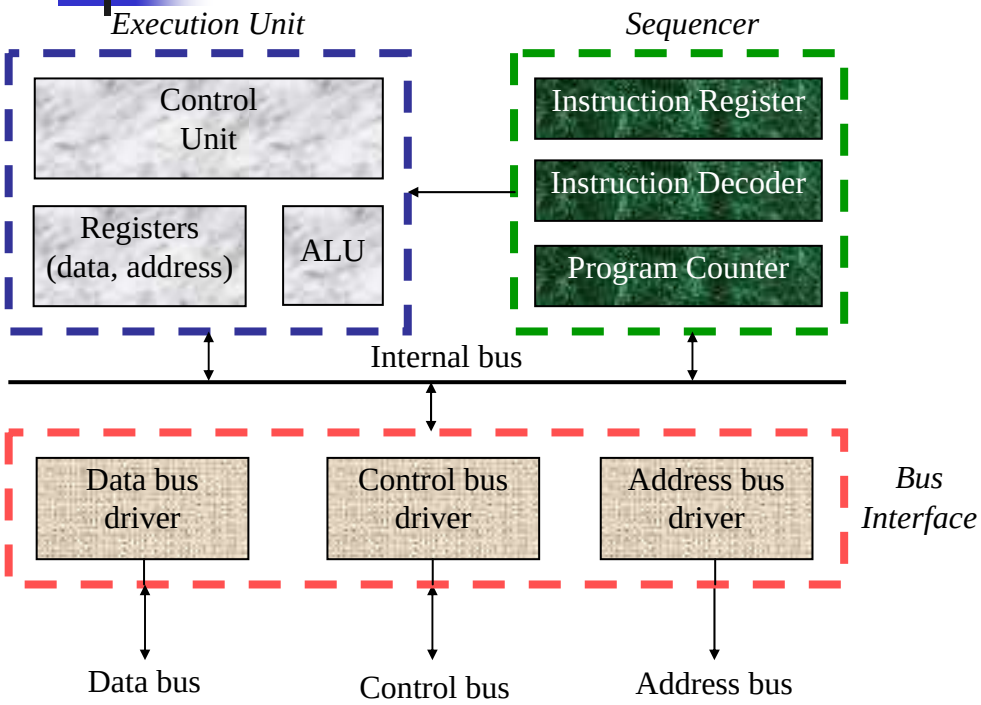
- 3. Bus điều khiển
  - Gồm các đường tín hiệu khác nhau ($\overline{}$, $\overline{RD}$
  - Hướng truyền tuỳ vào loại tín hiệu
  - 06 loại truyền thông tiêu biểu mà bus điều khiển phải xác định bằng tín hiệu điện
    - Đọc/ghi từ/vào bộ nhớ
    - Đọc/ghi từ/vào I/O
    - Nhận biết yêu cầu ngắt (interrupt acknowledge)
    - Nhận biết yêu cầu treo (phục vụ DMA, hold acknowledge)

Ch1: II Các loại bus



Ch1: II Các loại bus

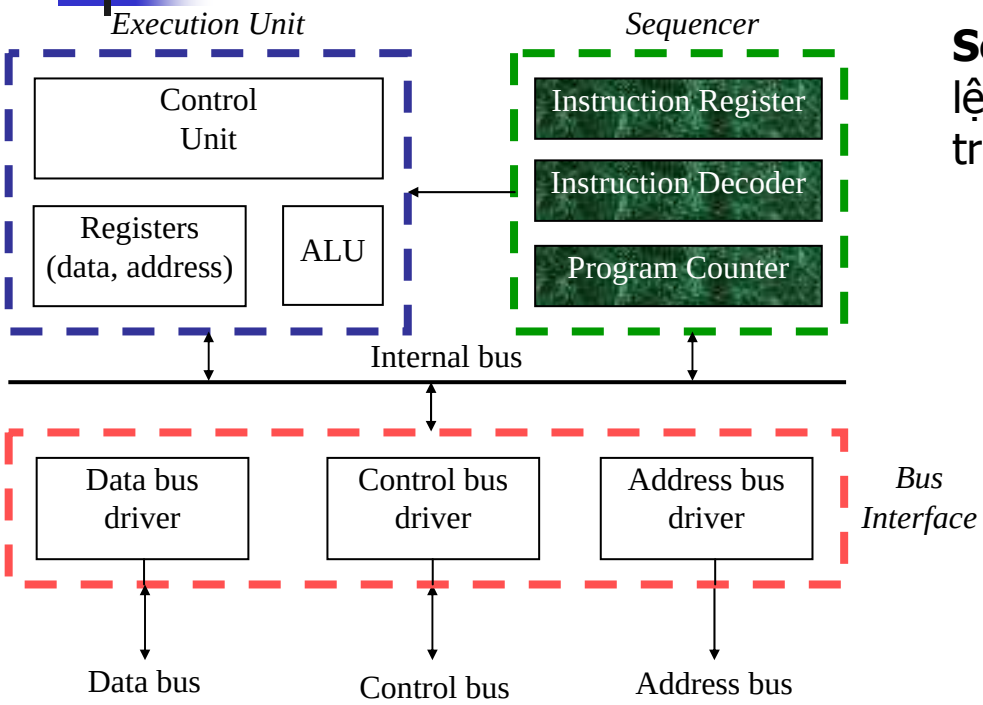
Ch1 III Chip Vi xử lý µP



Hình 1.6

Execution Unit: Khối thực thi
Control Unit: Khối điều khiển
Registers: Các thanh ghi
ALU (Arithmetic & Logic Unit): Khối logic - số học
Sequencer: Bộ điều khiển tuần tự
Instruction Register: Thanh ghi lệnh
Instruction Decoder: Bộ giải mã lệnh
Program Counter: Bộ đếm chương trình
Internal bus: Bus nội
Bus interface: Giao tiếp bus
Data bus driver: Bộ điều khiển bus dữ liệu
Control bus driver: Bộ điều khiển bus điều khiển
Address bus driver: Bộ điều khiển bus địa chỉ

Ch1 III Chip Vi xử lý µP

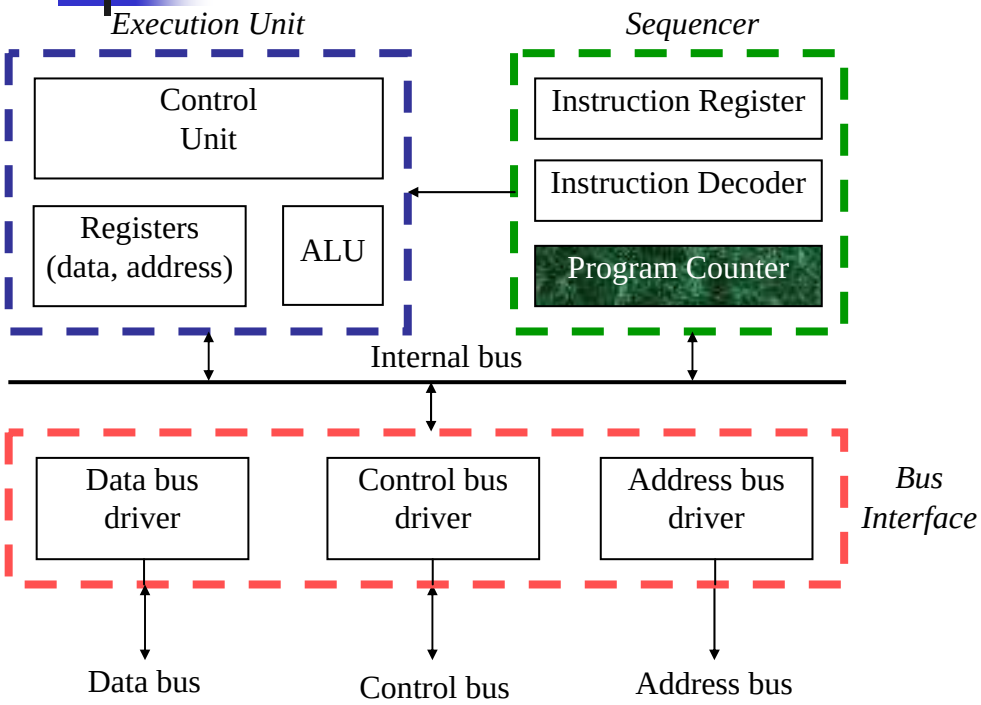


Hình 1.6

Sequencer: Bộ điều khiển tuần tự: nhận lệnh từ bộ nhớ, sau đó giải mã lệnh và truyền lệnh đã giải mã đến khối thực thi

Instruction Register: Thanh ghi lệnh
Instruction Decoder: Bộ giải mã lệnh
Program Counter: Bộ đếm chương trình

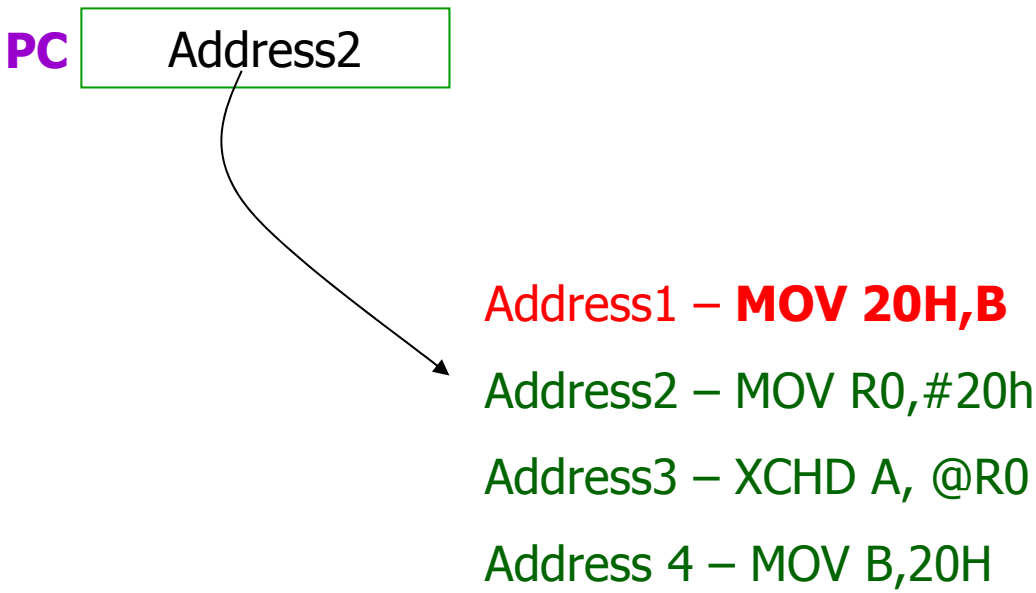
Ch1 III Chip Vi xử lý µP



Hình 1.6

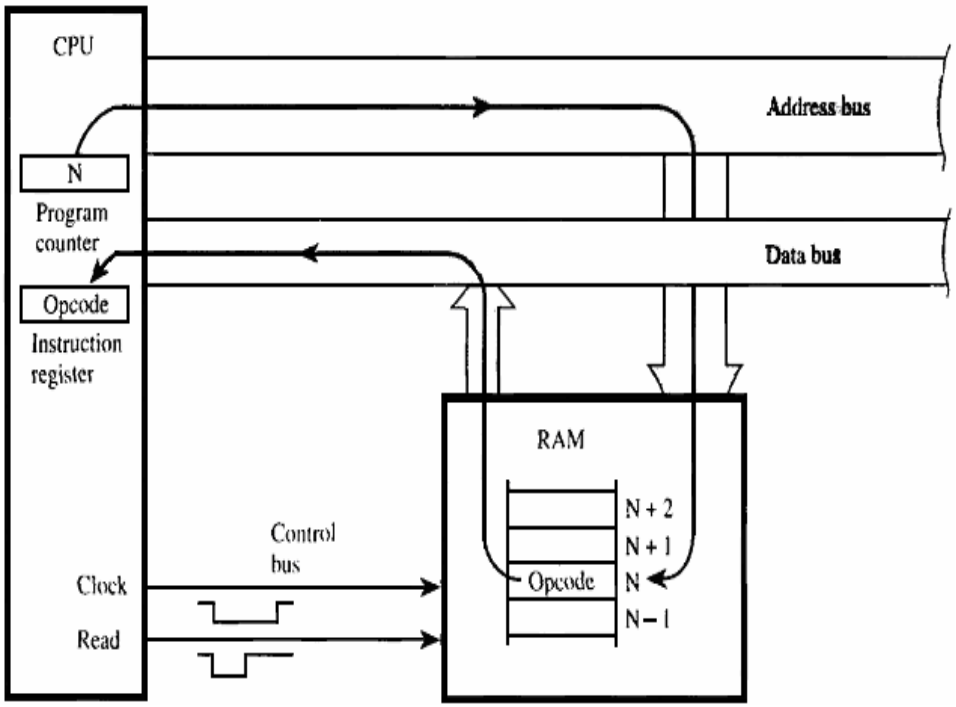
- Thanh ghi PC (bộ đếm chương trình):
  - Nội dung là địa chỉ ô nhớ chứa mã lệnh cần truy xuất (lệnh kế tiếp lệnh đang thực thi)

Ch1 III Chip Vi xử lý µP



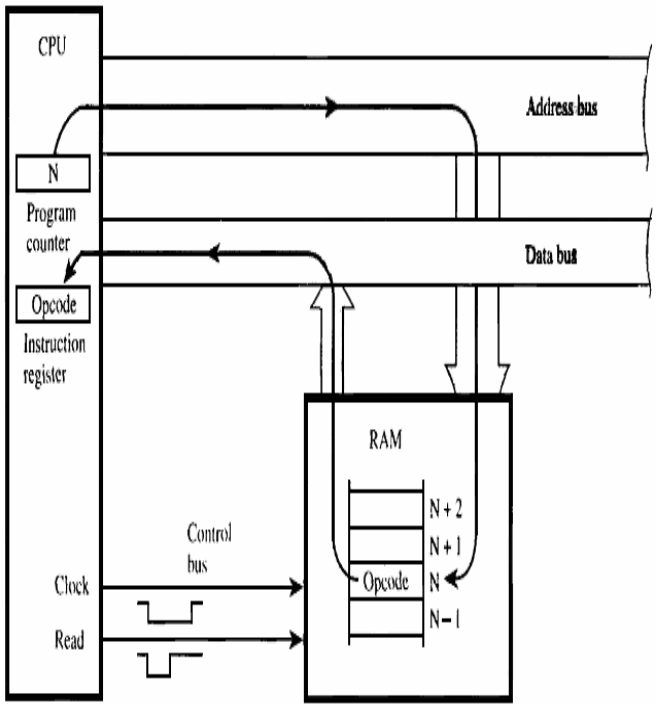
- Thanh ghi PC (bộ đếm chương trình):
  - Nội dung là địa chỉ ô nhớ chứa mã lệnh cần truy xuất (lệnh kế tiếp lệnh đang thực thi)
  - Gặp lệnh chuyển điều khiển (nhảy, gọi chương trình con…) thì nội dung PC bị thay đổi
  - Còn có tên là con trỏ lệnh IP (Instruction Pointer)

Ch1 III Chip Vi xử lý µP



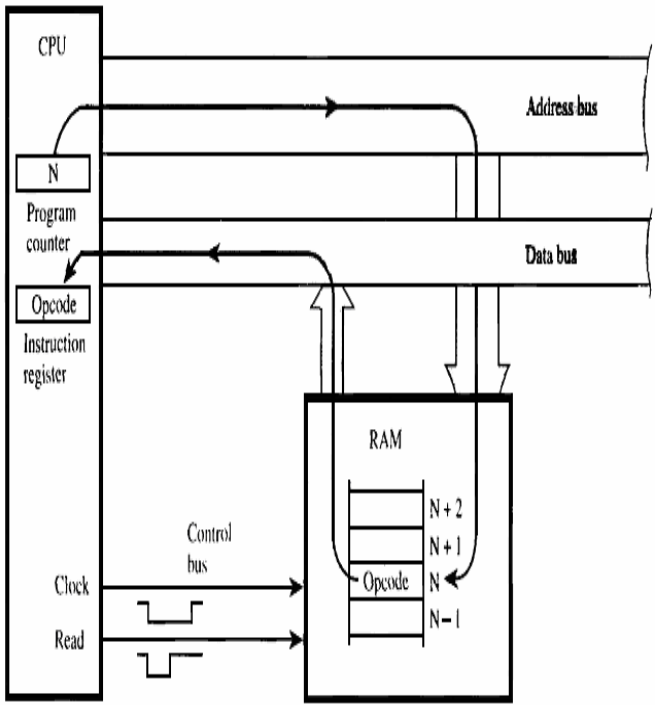
- *Việc tìm nạp lệnh từ bộ nhớ là một trong các thao tác cơ bản nhất mà µP thực hiện, gồm các bước như sau:*
  - *- Nội dung của PC được đặt lên bus địa chỉ.*
  - *- Tín hiệu điều khiển READ được xác lập (chuyển sang trạng thái tích cực).*
  - *- Mã lệnh được đọc từ bộ nhớ và đưa lên bus dữ liệu.*
  - *- Mã lệnh được chốt vào thanh ghi lệnh IR bên trong.*
  - *- PC được tăng lên để chuẩn bị tìm nạp lệnh kế từ bộ nhớ.*

Ch1 III Chip Vi xử lý µP



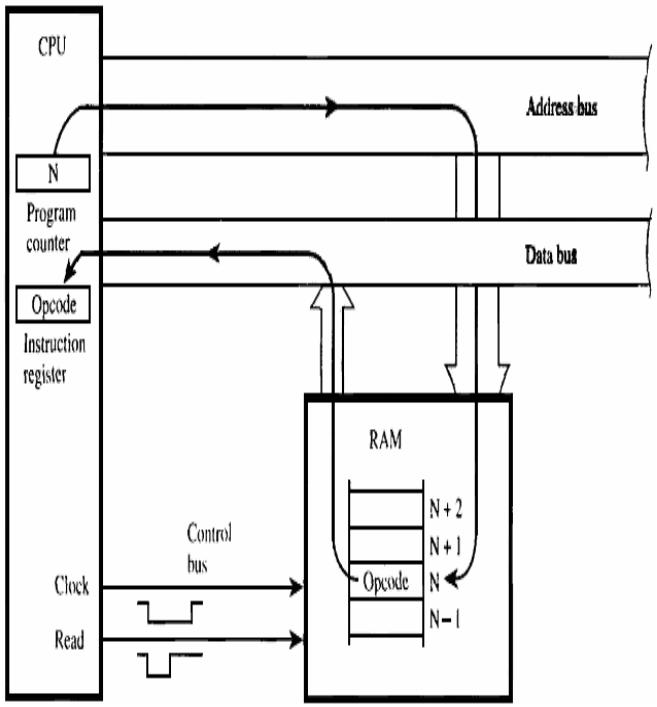
- *Một chu kỳ lệnh có thể chia thành 2 bước:*
  - *Chu kỳ nhận lệnh: CPU sẽ xuất nội dung thanh ghi PC ra bus địa chỉ, đồng thời xuất tín hiệu đọc lệnh trên bus dữ liệu➔ giải mã địa chỉ nhận lệnh(địa chỉ, tín hiệu điều khiển) và cho phép xuất ô nhớ có địa chỉ tương ứng, đặt dữ liệu (là mã lệnh) lên bus data. CPU đọc data này và cất trong IR. Đồng thời, nội dung PC tăng, trỏ vào địa chỉ mã lệnh kế tiếp. Thuật ngữ PC hiện hành là PC đã trỏ vào mã lệnh kế tiếp*

Ch1 III Chip Vi xử lý µP



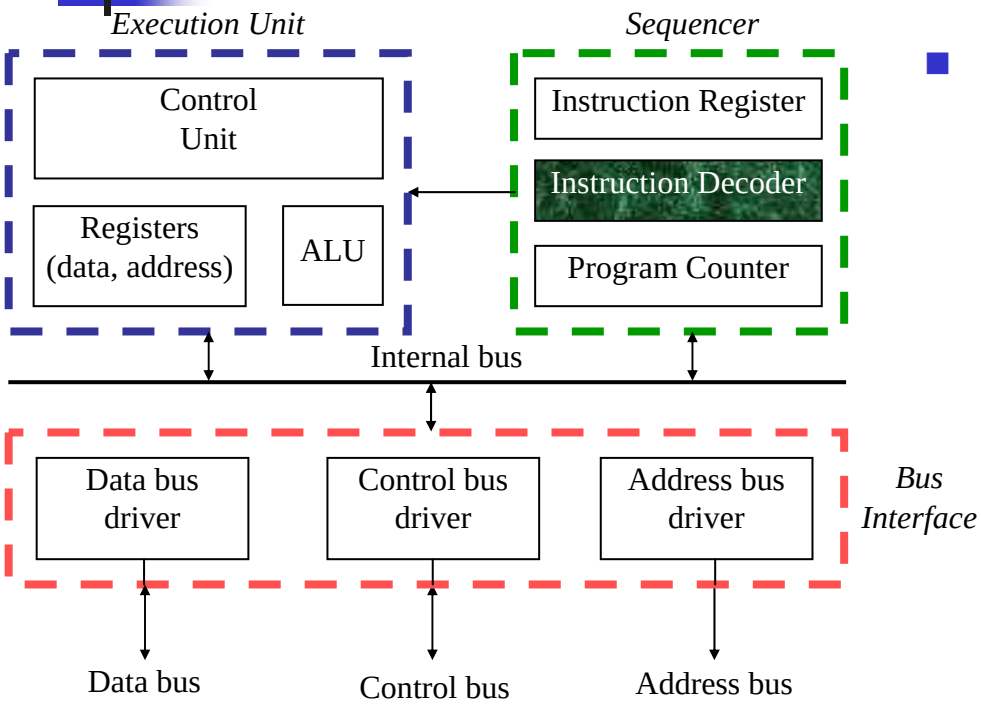
- *Một chu kỳ lệnh có thể chia thành 2 bước:*
  - *Chu kỳ thực thi lệnh: giải mã lệnh nhận lệnh từ IR, giải mã lệnh và phát tín hiệu điều khiển đến các khối liên quan để thực hiện lệnh. Tuỳ lệnh mà việc thực thi chỉ thực hiện bên trong CPU hay cần giao tiếp ra bên ngoài.*

Ch1 III Chip Vi xử lý µP



- *Một chu kỳ lệnh có thể chia thành 2 bước:*
  - *Hiện nay, người ta dùng kỹ thuật đường ống có nghĩa là 2 chu kỳ trên hoạt động cùng 1 thời điểm để tiết kiệm chu kỳ bus.*

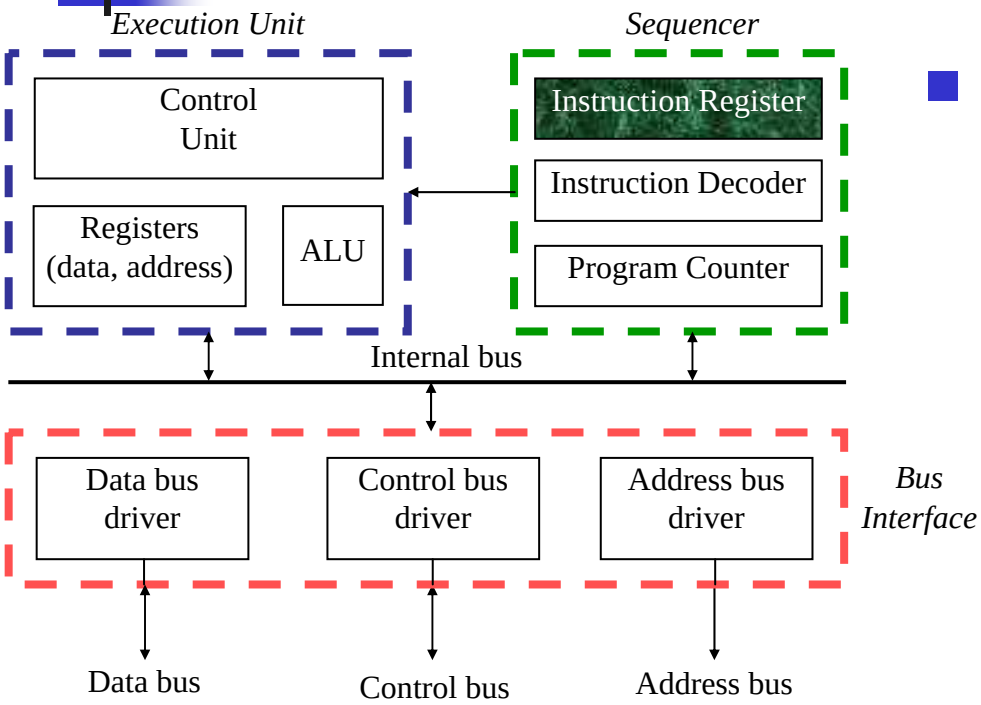
Ch1 III Chip Vi xử lý µP



Hình 1.6

- Bộ giải mã lệnh:
  - Nhận lệnh từ IR
  - Thông dịch (diễn dịch) các lệnh được nhận vào µP
  - Tác động đến những phần khác (ALU, các thanh ghi đa dụng…) để lệnh đó được thực hiện
  - Từ điển lưu nghĩa của mỗi lệnh
  - ID càng lớn thì PC càng hiểu nhiều lệnh

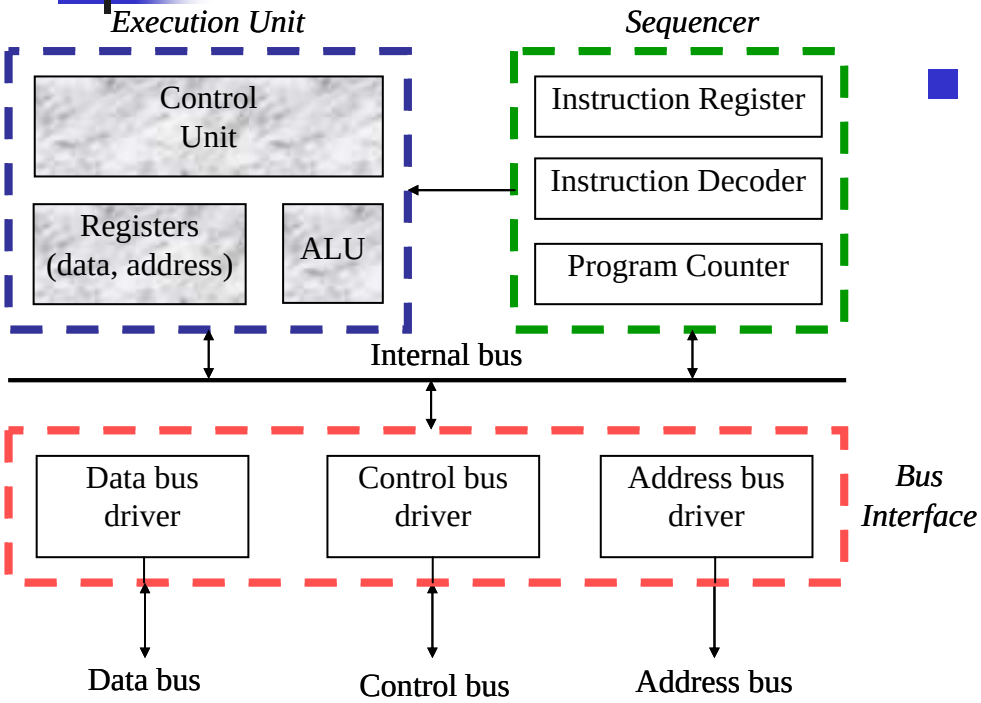
Ch1 III Chip Vi xử lý µP



Hình 1.6

- Thanh ghi lệnh:
  - Lưu trữ mã nhị phân của lệnh đang được thực thi

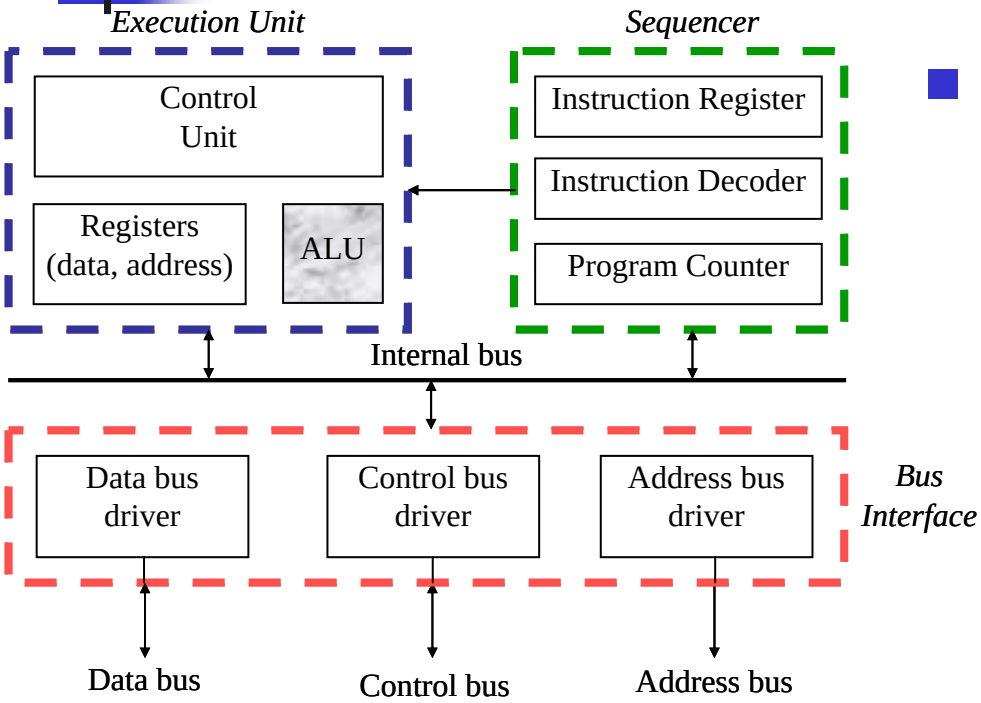
Ch1 III Chip Vi xử lý µP



Hình 1.6

- Khối thực thi:
  - Thực thi và ghi kết quả câu lệnh
  - Các toán hạng nằm ở thanh ghi hoặc ở bus nội

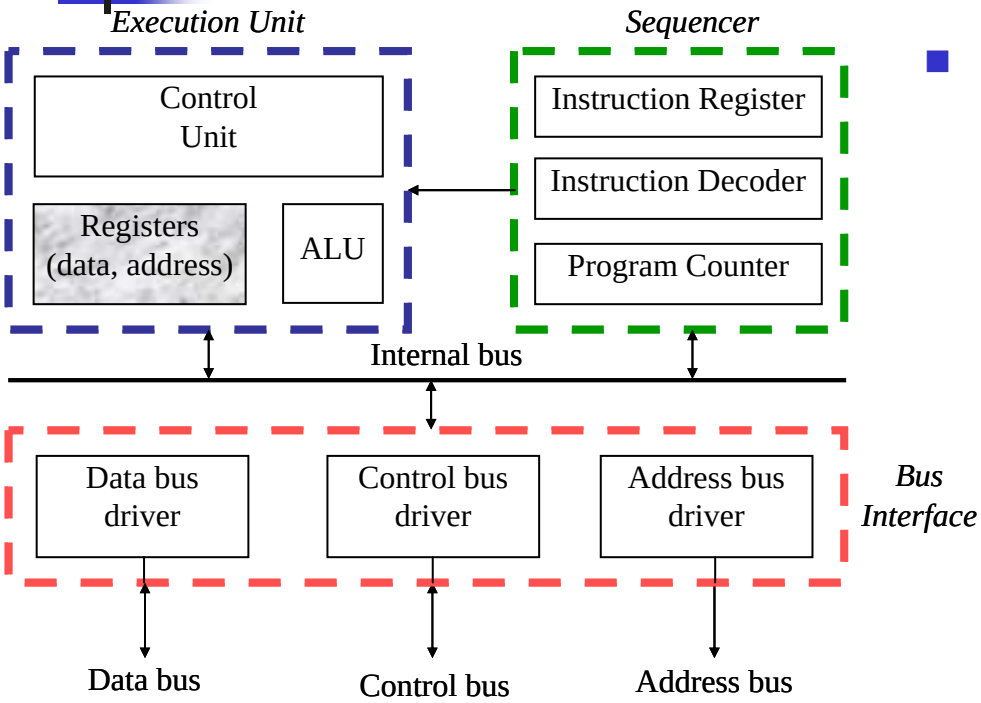
Ch1 III Chip Vi xử lý µP



Hình 1.6

- ALU:
  - Vi mạch điện tử
  - Thực hiện các phép toán số học (+,-,*,/) và logic(and, or, not, xor)

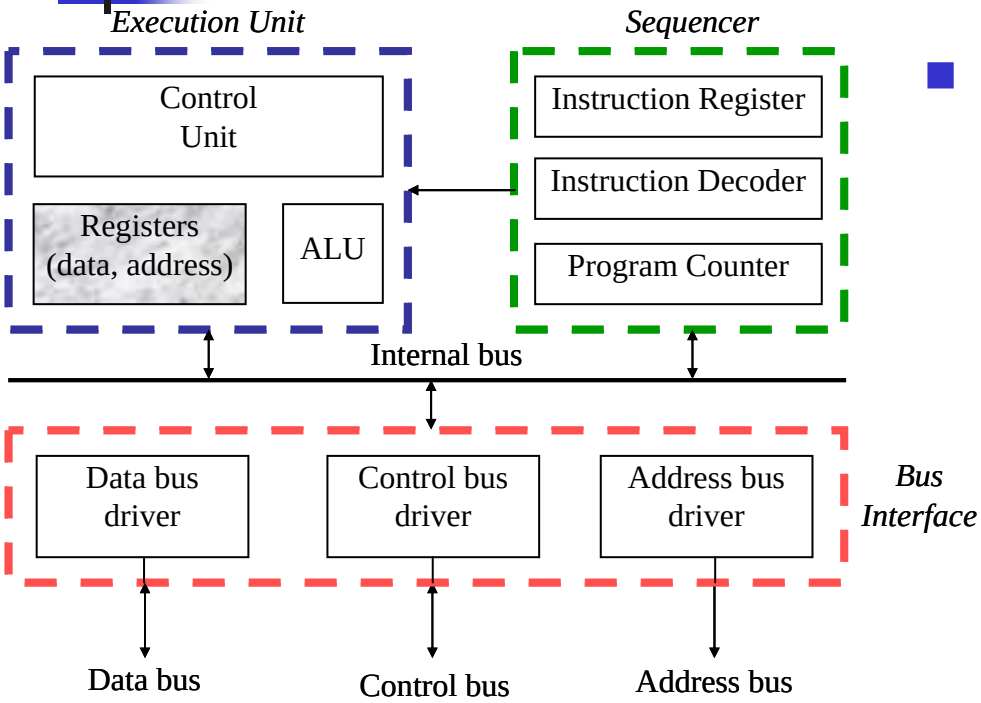
Ch1 III Chip Vi xử lý µP



Hình 1.6

- Thanh ghi đa dụng:
  - Chức năng chính: lưu trữ tạm thời dữ liệu
  - Nội dung: dữ liệu cần xử lý hoặc địa chỉ chứa giá trị cần xử lý nhận từ bộ nhớ hoặc I/O
  - Thanh ghi và độ rộng thanh ghi càng lớn → càng tốt. (không thực hiện nhiều phép truyền thông tin giữa µP và bộ nhớ do truy xuất trực tiếp từ thanh ghi

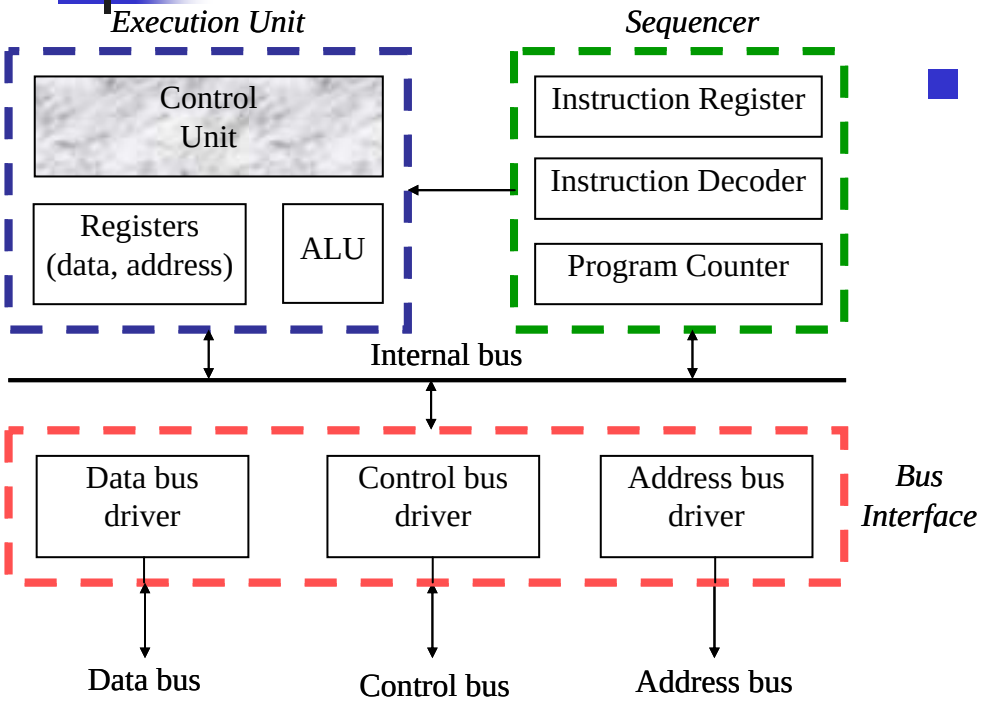
Ch1 III Chip Vi xử lý µP



Hình 1.6

- Thanh ghi đa dụng:
  - Mỗi thanh ghi có địa chỉ truy xuất đến (byte/bit tuỳ vị trí)
  - Nối với nhau hoặc đến các phần tử khác thuộc µP bằng bus nội
  - Độ rộng: 8 bit, 16bit, 32bit, 64bit.

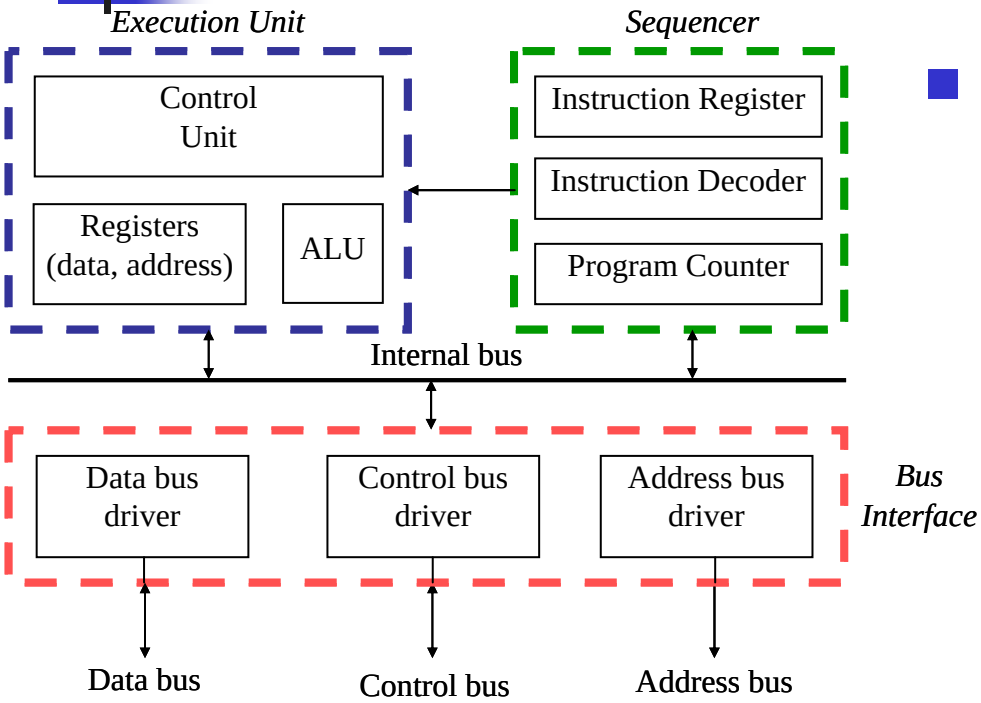
Ch1 III Chip Vi xử lý µP



Hình 1.6

- Khối điều khiển:
  - Tạo tín hiệu điều khiển cho các hoạt động bên trong + bên ngoài của µP .

Ch1 III Chip Vi xử lý µP



Hình 1.6

- Giao tiếp bus:
  - Gồm 3 bộ điều khiển để giao tiếp với bus bên ngoài tương ứng

Ch1 IV Bộ nhớ (memory)

- Nhắc lại các đơn vị bit, nibble, byte, word
  - Bit: 0,1
  - Nibble: 4 bit
  - Byte: 8 bit
  - Word
    - 2 byte
    - Dài:4 byte (theo thế hệ vi xử lý 16bit, 32bit)

Ch1 IV Bộ nhớ (memory)

- 1. Phân loại
- 2. Cấu trúc bên trong tiêu biểu của bộ nhớ
- 3. Truy xuất bộ nhớ
- 4. Giải mã địa chỉ cho bộ nhớ

Ch1 IV Bộ nhớ (memory)

- 1. Phân loại
  - Bộ nhớ thường được chia làm hai loại: bộ nhớ cơ bản (hay bộ nhớ chính – main memory) và bộ nhớ lưu trữ (storage memory).
  - - Bộ nhớ chính: ROM và RAM.
  - - Bộ nhớ lưu trữ: băng từ, đĩa mềm, đĩa cứng…
    - Thông thường bộ nhớ lưu trữ được xem như là thiết bị I/O

Ch1 IV Bộ nhớ (memory)

- ***a. Bộ nhớ chỉ đọc – ROM (Read-Only Memory)***
  - Là bộ nhớ chỉ đọc, không thể sửa đổi thông tin trong các hoạt động thông thường.
  - Thông tin ghi trong ROM sẽ không bị mất đi khi mất nguồn cung cấp.
  - ROM được ghi bằng thiết bị chuyên dụng.
  - ROM thường được dùng để chứa các chương trình và dữ liệu cố định (*chương trình khởi động, dữ liệu tra bảng …*)

Ch1 IV Bộ nhớ (memory)

- ***b. Bộ nhớ truy xuất ngẫu nhiên – RAM (Random Access Memory)***
  - Cho phép đọc/ghi thông tin bất kỳ lúc nào trong quá trình làm việc mà không cần thiết bị đặc biệt.
  - Thông tin trong RAM sẽ bị mất khi mất nguồn cung cấp.

Ch1 IV Bộ nhớ (memory)

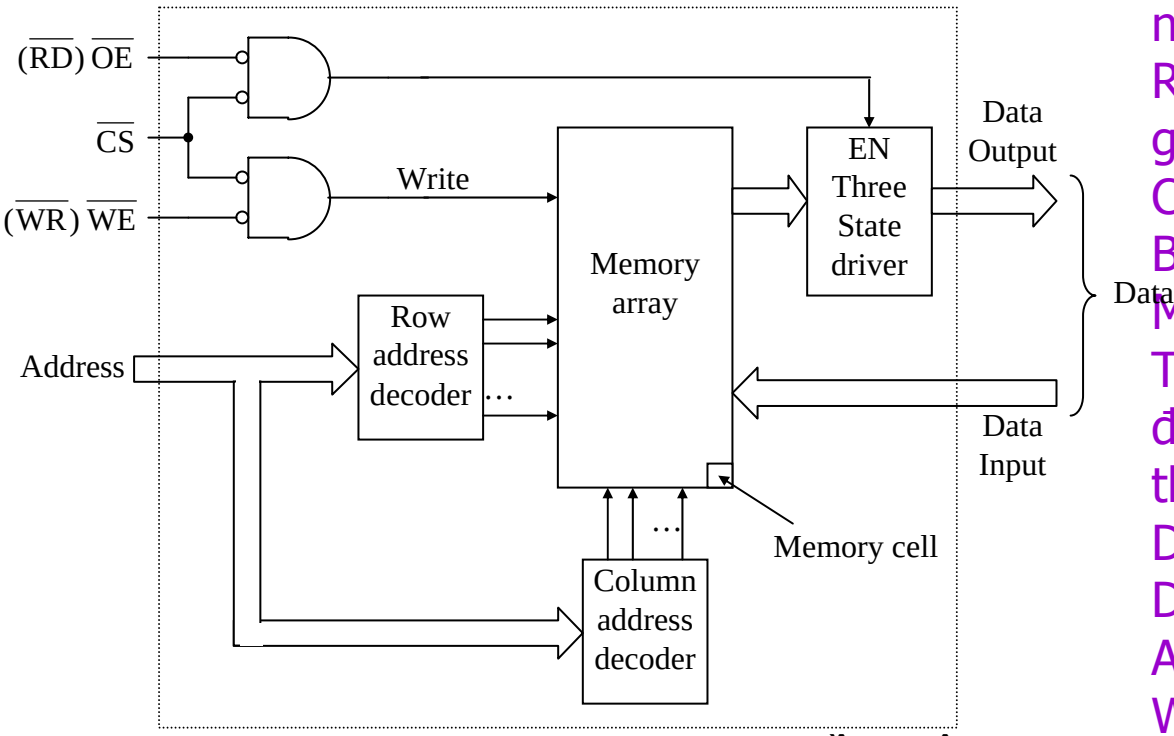
- Có hai loại RAM chính:
  - *RAM động – DRAM* (Dynamic RAM): có cấu tạo từ các transistor MOSFET và tụ điện (*1 phần tử nhớ*), lưu trữ thông tin bằng điện tích trong tụ nên thông tin có thể mất đi (rò rỉ hết) nếu không có biện pháp duy trì thích hợp. Do đó cần có quá trình *làm tươi* (refresh) định kì để phục hồi nội dung của các ô nhớ trước khi nó mất đi (rò rỉ hết). DRAM có thể tích hợp với dung lượng lớn.
  - *RAM tĩnh – SRAM* (Static RAM): cấu tạo từ những Flipflop (FF) (*1 phần tử nhớ*), mỗi FF lưu trữ một bit thông tin nên SRAM không cần quá trình làm tươi để duy trì nội dung. Tuy nhiên, nó khó tích hợp với dung lượng lớn

Ch1 IV Bộ nhớ (memory)

- **2. Cấu trúc bên trong tiêu biểu của bộ nhớ**
  - Bộ nhớ gồm các *phần tử nhớ* hay *ô nhớ* (memory cell) được tổ chức dưới dạng ma trận. Mỗi ô nhớ chứa một bit thông tin.
  - Mảng nhớ được phân chia thành một chuỗi các *ngăn nhớ* hay *từ nhớ* (word).
  - Mỗi ngăn nhớ đều có một địa chỉ duy nhất.
  - Một ngăn nhớ có thể có 4-bit, 8-bit, 16-bit …
  - Ký hiệu: ***số ngăn nhớ x độ rộng mỗi ngăn nhớ***
- Ví dụ: bộ nhớ 1024 x 8 bao gồm 210 ngăn nhớ, mỗi ngăn nhớ có 8-bit.

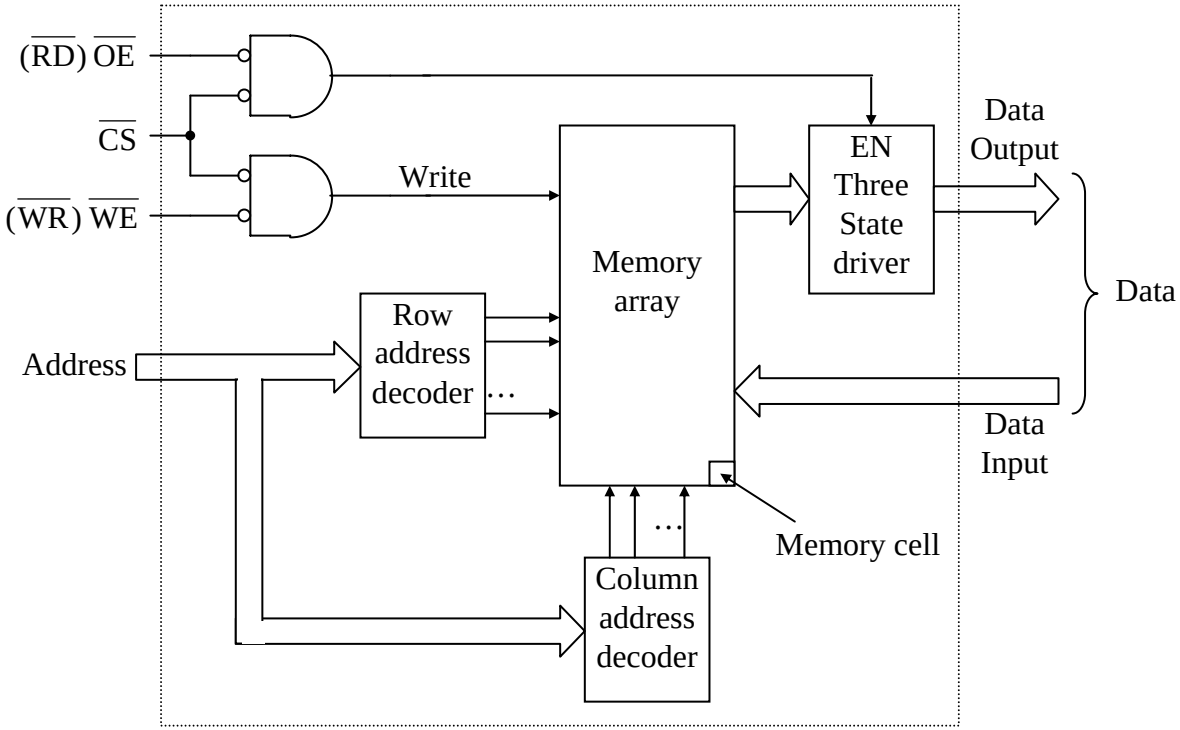
Ch1 IV Bộ nhớ (memory)

- Cấu trúc bên trong tiêu biểu của bộ nhớ:



Memory array: Mảng ô nhớ
Row address decoder: Bộ giải mã địa chỉ hàng
Column address decoder: Bộ giải mã địa chỉ cột
Memory cell: Ô nhớ
Three state driver: Bộ điều khiển ngõ ra 3 trạng thái
Data Output: Dữ liệu ra
Data Input: Dữ liệu vào
Address: Địa chỉ
Write: Ghi

Ch1 IV Bộ nhớ (memory)



Hình 1.8

Các tín hiệu tiêu biểu trên một chip nhớ:

+ (Chip Select): tín hiệu chọn chip (cho phép chip).
+ (Output Enable): tín hiệu cho phép xuất dữ liệu (*nhận xung kích từ µP*).
+ (Write Enable): tín hiệu cho phép ghi dữ liệu (*nhận xung kích từ µP*).
+ Address: các tín hiệu địa chỉ (*từ bus địa chỉ*) để chọn ngăn nhớ cần thao tác.

Ch1: II Các loại bus

- Các đặc trưng của tập lệnh
  - Tuỳ ứng dụng → có tập lệnh khác nhau