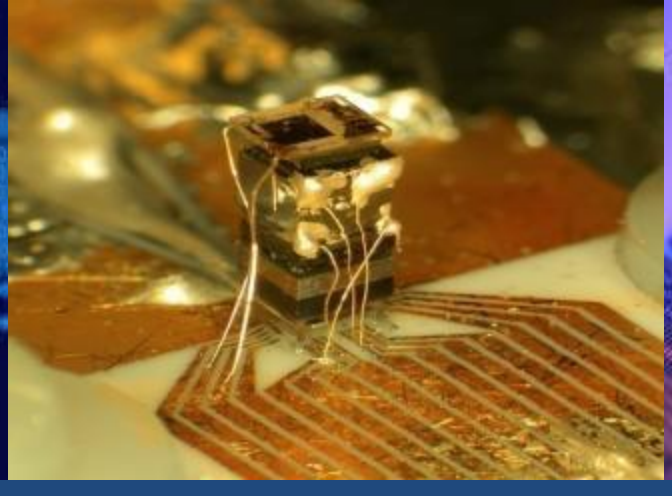
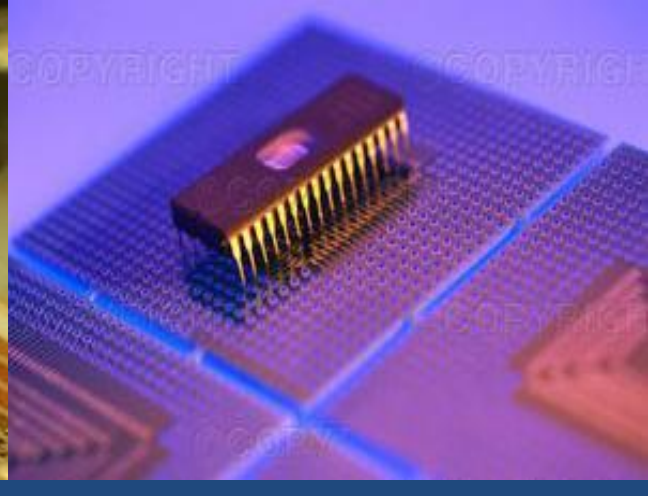



Môn học:

KỸ THUẬT VI XỬ LÝ

Ngô Lam Trung

email: trungnl@soict.hust.edu.vn

Bộ môn Kỹ thuật Máy tính

Viện CNTT&TT- ĐH BKHN

Mục tiêu môn học

- Sau khi kết thúc môn học này, sinh viên có thể
  - Trình bày được kiến trúc phần cứng và phần mềm của họ vi xử lý 80x86, tập trung chi tiết vào các bộ vi xử lý 8088/8086 của Intel
  - Lập trình hợp ngữ sử dụng tập lệnh của 8088/8086
  - Trình bày được cách phối ghép vi xử lý 8088/8086 với bộ nhớ và hệ thống vào ra

Tài liệu tham khảo chính

1. *Văn Thế Minh* - Kỹ thuật vi xử lý - *1997.*
2. *Walter A. Triebel, Avtar Singh - The 8088 and 8086 Microprocessors: Programming, Interfacing, Software, Hardware and Applications - 1997.*
3. *Ytha Yu, Charles Marut - Assembly Language Programming and Organization of the IBM-PC - 1992. (Bản dịch: Quách Tuấn Ngọc)*

Nội dung môn học

Chương 1. Giới thiệu chung về vi xử lý và máy vi tính

Chương 2. Kiến trúc bộ vi xử lý 8088/8086

Chương 3. Lập trình hợp ngữ trên PC

Chương 4. Bộ vi xử lý 8088 và nối ghép với bộ nhớ

Chương 5. Nối ghép vào ra với 8088

Chương 6. Ngắt và xử lý ngắt trong 8088

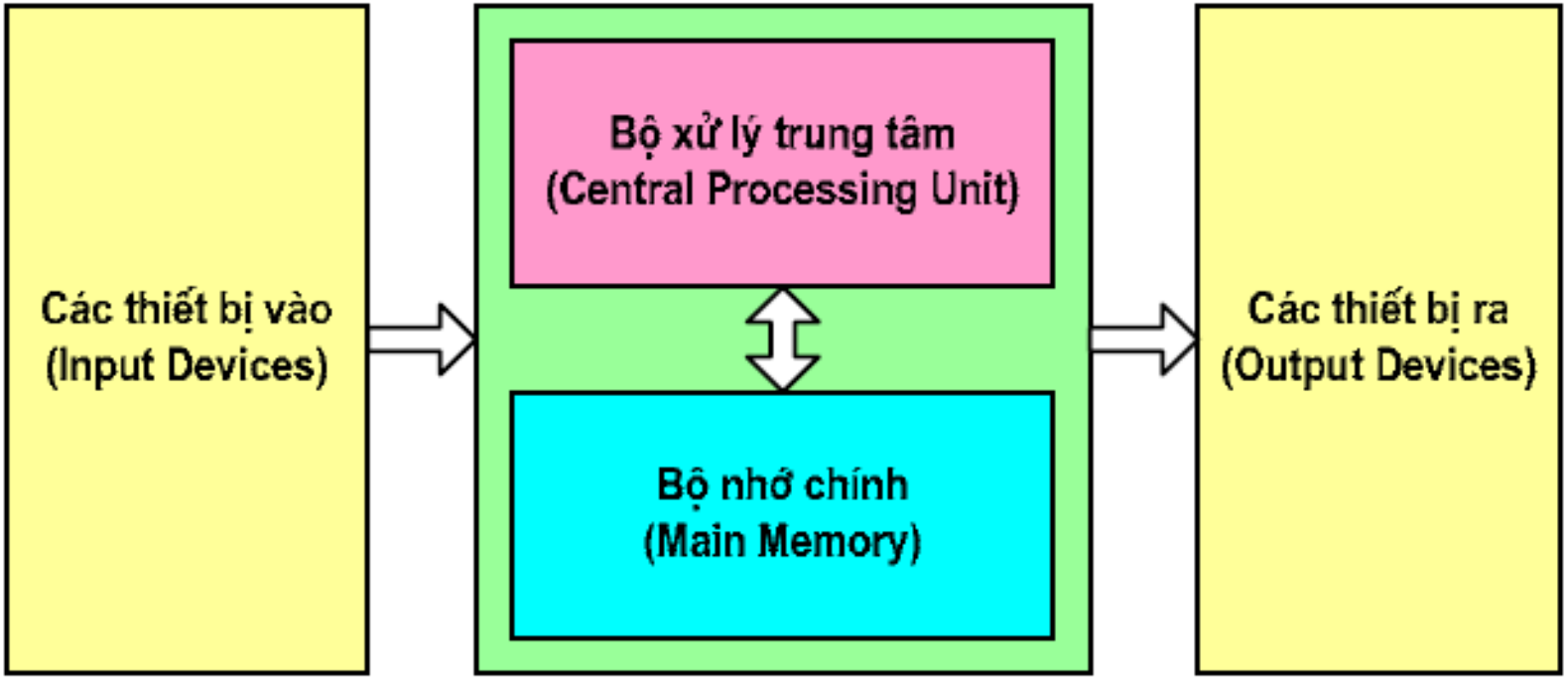
CHƯƠNG 1

Giới thiệu chung về vi xử lý

1.1. Vai trò của bộ vi xử lý trong máy tính

1.2. Sự phát triển của vi xử lý

1.3. Kiến trúc chung của một hệ thống máy tính



Mô hình máy tính cơ bản

Nội dung chương 1

1.1. Máy tính và phân loại máy tính

1.2. Sự phát triển của vi xử lý

1.3. Kiến trúc chung của một hệ thống máy tính

Sự phát triển của vi xử lý

- Bộ vi xử lý (Microprocessor): vi mạch tích hợp thực hiện chức năng của bộ xử lý trung tâm (CPU).
- Thế hệ 4 bit (1971÷1973):
  - Intel 4004 (bộ VXL đầu tiên), 4040
- Thế hệ 8 bit (1974÷1977):
  - Intel 8080, 8085
  - Motorola 6800
  - Zilog Z80

Sự phát triển của vi xử lý

- Thế hệ 16 bit (1979÷1982):
  - Intel 8086, 8088, 80186, 80286
  - Motorola 68000, 68010
    - ✓Hãng IBM sử dụng 8088 để thiết kế máy IBM-PC (1981)
    - ✓Hãng Apple sử dụng 68000 để thiết kế máy Macintosh (1983)
  - Zilog Z8000

Sự phát triển của vi xử lý

- Thế hệ 32 bit (1983÷1991):
  - Intel 80386, 80486, Pentium, PII, PIII và P4 (sau này)
  - Motorola 68020, 68030, 68040, 68060
  - Kiến trúc RISC (máy tính với tập lệnh rút gọn):
    - ✓Power PC
    - ✓SPARC
  - Hiện nay: bộ xử lý lõi kép (Intel Core Duo, ...)
- Thế hệ 64 bit (1992÷nay):
  - Intel: Itanium, Pentium D, Xeon, Intel Core 2, ...
  - Digital ALPHA
  - Power PC
  - Super SPARC

Nội dung chương 1

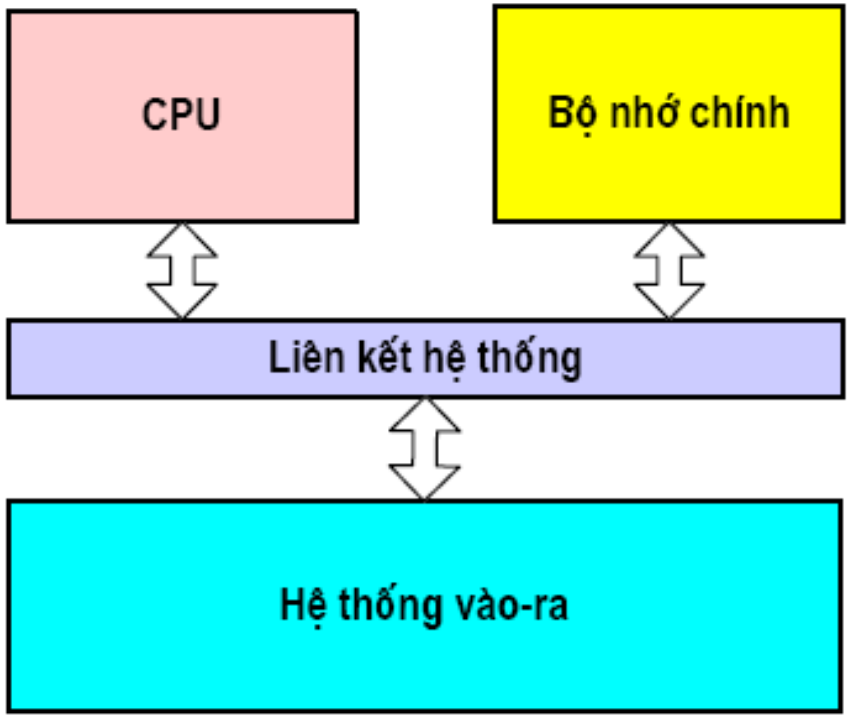
1.1. Máy tính và phân loại máy tính

1.2. Sự phát triển của vi xử lý

1.3. Kiến trúc chung của một hệ thống máy tính

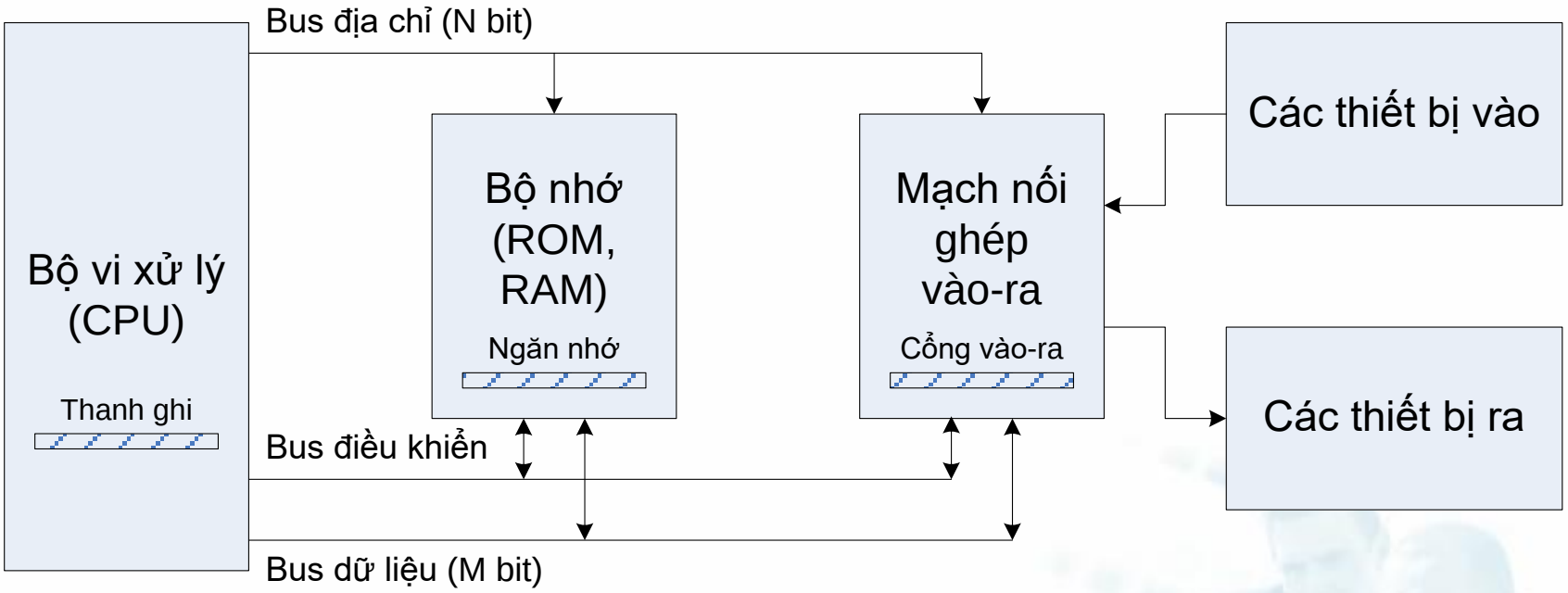
Kiến trúc phần cứng

- Bộ vi xử lý → **CPU**
- Bộ nhớ bán dẫn (ROM, RAM) → **bộ nhớ chính**
- Hệ thống vào-ra: gồm mạch nối ghép vào-ra và các thiết bị ngoại vi
- Các đường bus truyền thông tin



Sơ đồ khối kiến trúc phần cứng

Kiến trúc phần cứng

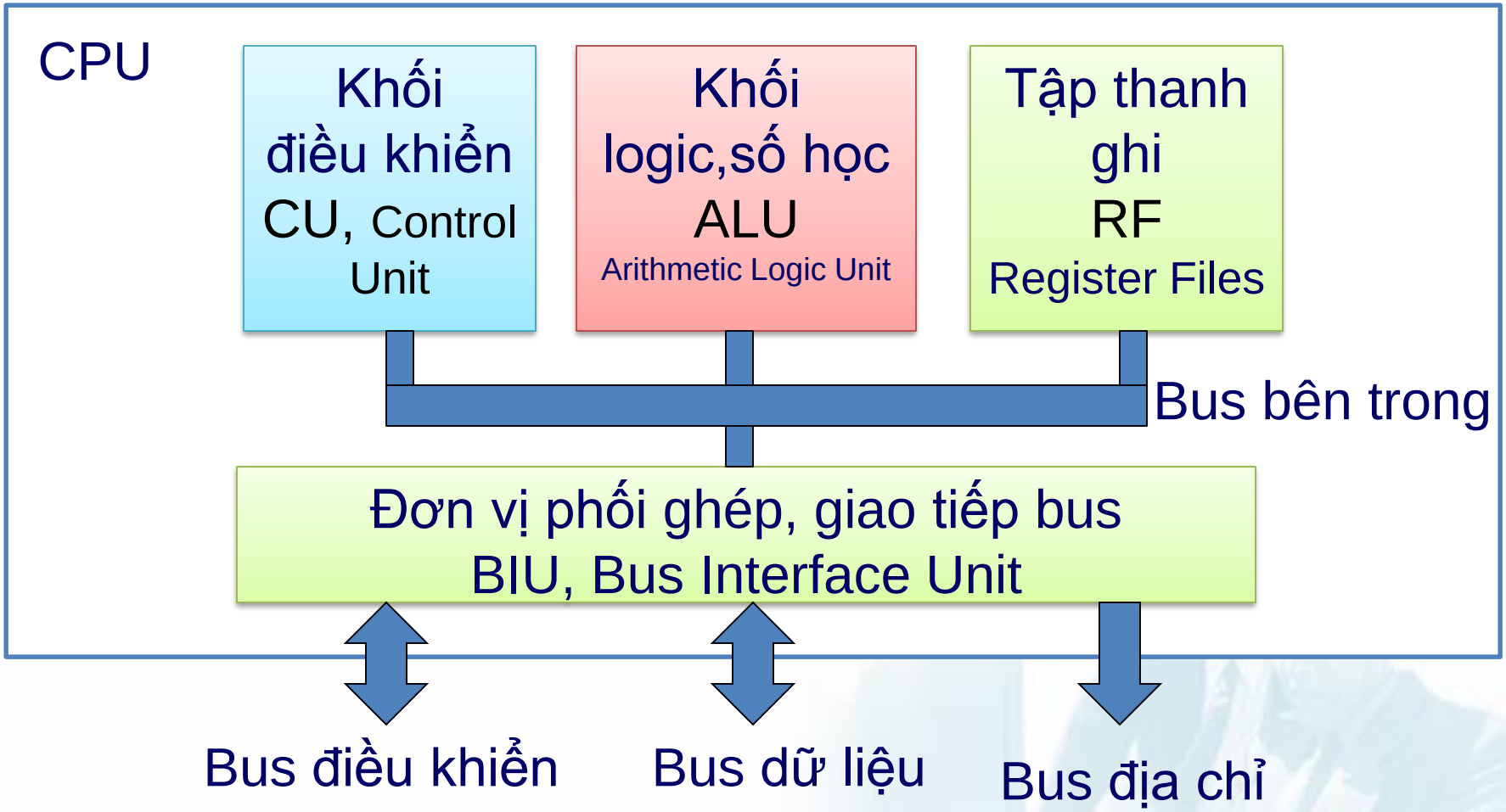


Thiết bị vào-ra không nối trực tiếp với bus hệ thống mà thông qua các cổng vào-ra.

Bộ xử lý trung tâm (CPU)

- Chức năng:
  - Điều khiển hoạt động của toàn bộ hệ thống
  - Xử lý dữ liệu
- Nguyên tắc hoạt động cơ bản: bộ VXL hoạt động theo chương trình nằm trong bộ nhớ bằng cách:
  - Nhận lần lượt từng lệnh (dưới dạng mã hóa nhị phân) từ bộ nhớ,
  - Sau đó tiến hành giải mã lệnh và phát các tín hiệu điều khiển thực thi lệnh.
  - Trong quá trình thực thi lệnh, bộ vi xử lý có thể trao đổi dữ liệu với bộ nhớ hay hệ thống vào-ra.

Các thành phần chính của bộ xử lý trung tâm



Các thành phần của bộ xử lý trung tâm

- Đơn vị điều khiển (Control Unit):
  - Điều khiển nhận lệnh từ bộ nhớ
  - Giải mã lệnh và phát tín hiệu thực hiện lệnh
  - Nhận các tín hiệu yêu cầu từ bên ngoài và đáp ứng các yêu cầu đó
- Đơn vị số học và logic (Arithmetic and Logic Unit):
  - Số học: cộng, trừ, nhân, chia, tăng, giảm, đảo dấu, so sánh, ...
  - Logic: AND, OR, XOT, NOT, dịch và quay bit

Các thành phần của bộ xử lý trung tâm

- Tập thanh ghi (Registers):
  - Chứa các thông tin tạm thời phục vụ cho hoạt động của bộ vi xử lý
    - ✓Thông tin về địa chỉ
    - ✓Dữ liệu tạm thời
    - ✓Thông tin trạng thái
  - Mỗi bộ vi xử lý có từ vài chục đến vài trăm thanh ghi
- Đơn vị nối ghép bus (Bus Interface Unit):
  - Nối ghép các thành phần bên trong bộ vi xử lý với bên ngoài.

Bộ nhớ chính

- Chức năng: chứa các chương trình và dữ liệu mà bộ vi xử lý có khả năng trao đổi trực tiếp.
- Được tổ chức thành các ngăn nhớ (thường theo Byte)
  - Mỗi ngăn nhớ có một địa chỉ xác định.
  - Bộ vi xử lý muốn trao đổi thông tin với ngăn nhớ nào thì phải biết địa chỉ của ngăn nhớ đó.

Bộ nhớ chính (tiếp)

- Bộ nhớ chính được thiết kế trên cơ sở gồm:
  - ROM (Read Only Memory):
    - ✓Bộ nhớ không khả biến.
    - ✓Chứa các chương trình và dữ liệu cố định với hệ thống.
  - RAM (Random Access Memory):
    - ✓Bộ nhớ khả biến.
    - ✓Chứa các thông tin tạm thời.

Hệ thống vào-ra

- Chức năng: Trao đổi thông tin giữa hệ vi xử lý với thế giới bên ngoài.
- Các thành phần chính:
  - Các thiết bị ngoại vi:
    - ✓Chuyển đổi dữ liệu giữa bên trong và bên ngoài hệ vi xử lý.
  - Các mạch nối ghép vào-ra:
    - ✓Nối ghép giữa thiết bị ngoại vi với hệ vi xử lý.
    - ✓Trên mạch nối ghép vào-ra có các cổng vào-ra (I/O Port).
    - ✓Mỗi cổng vào-ra cũng được đánh một địa chỉ xác định.
    - ✓Thiết bị ngoại vi được kết nối và trao đổi dữ liệu với hệ vi xử lý thông qua các cổng vào-ra.

Bus truyền thông tin

- Bus: tập hợp các đường kết nối dùng để vận chuyển thông tin giữa các thành phần.
- Độ rộng bus: là số đường dây của bus có thể truyền thông tin đồng thời. Tính bằng bit.
- Phân loại bus theo chức năng:
  - Bus địa chỉ (Address Bus)
  - Bus dữ liệu (Data Bus)
  - Bus điều khiển (Control Bus)

Bus địa chỉ

- Chức năng: vận chuyển địa chỉ từ bên trong bộ vi xử lý đến bộ nhớ chính hay mạch nối ghép vào-ra để xác định ngăn nhớ hay cổng vào-ra cần trao đổi thông tin.
- Độ rộng bus địa chỉ: xác định dung lượng bộ nhớ cực đại của hệ thống.
- Nếu độ rộng bus địa chỉ là N bit (gồm N đường dây A_{N-1}, A_{N-2}, ... , A_2, A_1, A_0) thì:

  $\rightarrow$ có khả năng vận chuyển được N bit địa chỉ đồng thời

  $\rightarrow$ có khả năng đánh địa chỉ tối đa được 2^N ngăn nhớ = 2^N Byte $\rightarrow$ gọi là không gian địa chỉ bộ nhớ.

Ví dụ

- Độ rộng bus địa chỉ của một số bộ vi xử lý của Intel
  - 8088/8086 : N = 20 bit $\rightarrow$ KGĐCBN = 2^{20} Byte = 1 MB
  - 80286 : N = 24 bit $\rightarrow$ KGĐCBN = 2^{24} Byte = 16 MB
  - 80386, 80486, Pentium : N = 32 bit $\rightarrow$ KGĐCBN = 2^{32} Byte = 4 GB
  - Pentium II, III, 4 : N = 36 bit $\rightarrow$ KGĐCBN = 2^{36} Byte = 64 GB

Bus dữ liệu

- Chức năng:
  - Vận chuyển lệnh từ bộ nhớ chính đến bộ vi xử lý.
  - Vận chuyển dữ liệu giữa các thành phần của hệ vi xử lý với nhau.
- Độ rộng bus dữ liệu: Xác định số bit dữ liệu có thể được trao đổi đồng thời.
  - Nếu độ rộng bus dữ liệu là M bit (gồm M đường dây D_{M-1}, D_{M-2}, ... , D_2, D_1, D_0) thì nghĩa là đường bus dữ liệu đó có thể vận chuyển đồng thời được M bit dữ liệu.
  - M thường là 8, 16, 32, 64 bit.

Ví dụ

- Độ rộng bus dữ liệu của một số bộ vi xử lý của Intel:
  - 8088 : M = 8 bit
  - 8086, 80286 : M = 16 bit
  - 80386, 80486 : M = 32 bit
  - Các bộ xử lý Pentium : M = 64 bit

Bus điều khiển

- Chức năng: vận chuyển các tín hiệu điều khiển
- Các loại tín hiệu điều khiển:
  - Các tín hiệu điều khiển phát ra từ bộ vi xử lý để điều khiển bộ nhớ chính hay mạch nối ghép vào-ra.
  - Các tín hiệu yêu cầu từ bộ nhớ chính hay mạch nối ghép vào-ra gửi đến bộ vi xử lý.

Kiến trúc phần mềm

- Tập lệnh (Instruction Set)
- Ngôn ngữ máy
- Hợp ngữ (Assembly Language)
- Ngôn ngữ lập trình cấp cao

Tập lệnh (Instruction Set)

- Mỗi bộ vi xử lý có khả năng thực hiện được một tập hữu hạn các thao tác xác định (gọi là tập lệnh)
- Mỗi lệnh là một chuỗi số nhị phân thực hiện một công việc cụ thể
- Mỗi lệnh có cấu trúc như sau
  - Mã lệnh (Operation Code): cho biết chức năng của lệnh
  - Tham chiếu toán hạng: cho biết nơi chứa toán hạng mà lệnh tác động

Mã lệnh	Tham chiếu toán hạng

Tập lệnh (tiếp)

- Tập lệnh được mô tả thông qua các từ gợi nhớ: ADD (lệnh cộng), SUB (lệnh trừ), INC (lệnh tăng 1)…
- Ví dụ:
  - Lệnh hợp ngữ: **MOV AH,1**
  - Mã máy tương ứng:

0000 0001 1011 0100

Ngôn ngữ máy

- Ngôn ngữ chương trình **duy nhất** mà máy tính hiểu được là *ngôn ngữ máy* (chuỗi số nhị phân mã hóa cho một thao tác nào đó của bộ vi xử lý)

  → Lập trình trực tiếp bằng ngôn ngữ máy rất vất vả và dễ xảy ra sai sót.

Hợp ngữ (Assembly Language)

- Là ngôn ngữ lập trình gần với ngôn ngữ máy nhất.
- Sử dụng các kí hiệu gợi nhớ để biểu diễn các lệnh máy.
- Chương trình dịch hợp ngữ sang ngôn ngữ máy gọi là *hợp dịch* (Assembler).
- VD:
  - Ngôn ngữ máy (của bộ vi xử lý Intel 8088):

    00000101 00000100 00000000 ; cộng 4 vào AX
  - Hợp ngữ:

    ADD AX, 4 ; cộng 4 vào AX

Ngôn ngữ lập trình bậc cao

- Cho phép người lập trình viết các chương trình gần với ngôn ngữ tự nhiên hơn so với hợp ngữ.
- Chương trình viết bằng NNLT bậc cao gọi là *chương trình nguồn* (source code).
- Chương trình chuyển từ chương trình nguồn sang ngôn ngữ máy gọi là *chương trình dịch*.
  - Chương trình *biên dịch* (compiler):
    - ✓ Dịch toàn bộ chương trình nguồn sang mã máy.
    - ✓ Nếu chương trình nguồn có lỗi thì sẽ ngừng việc biên dịch.
  - Chương trình *thông dịch* (interpreter):
    - ✓ Dịch từng lệnh của chương trình nguồn (và thực hiện)
    - ✓ Nếu chương trình nguồn có lỗi thì vẫn thực hiện CT cho đến khi gặp lỗi.
- Ngôn ngữ lập trình trực quan (visual programming language): giúp việc xây dựng chương trình nhanh chóng và trực quan hơn.

Phần mềm hệ thống

- Là các chương trình điều khiển chung của hệ thống, bao gồm:
  - Các chương trình điều khiển được cài đặt sẵn trong bộ nhớ ROM của hệ vi xử lý.
  - Hệ điều hành: tập hợp các chương trình, đảm bảo:
    - ✓Điều khiển việc thực thi các chương trình khác.
    - ✓Quản lý, phân phối tài nguyên của hệ thống.
    - ✓Điều khiển các thiết bị và quá trình vào-ra.
    - ✓Cung cấp giao diện người dùng.
  - Các chương trình tiện ích: hỗ trợ thêm cho hệ điều hành.

Phần mềm ứng dụng

- Là các chương trình chạy trên một hệ điều hành cụ thể, phục vụ cho các ứng dụng cụ thể.
- Có nhiều loại.
- Ví dụ: một số phần mềm ứng dụng chạy trên hệ điều hành Microsoft Windows:
  - Ứng dụng văn phòng: Microsoft Word, Excel, …
  - Ứng dụng Internet: Internet Explorer, Outlook Express, Opera, …
  - Ứng dụng xử lý ảnh: Photoshop, Corel Draw, …

CHƯƠNG 2

Kiến trúc bộ vi xử lý 8088/8086

Nội dung chương 2

2.1. Kiến trúc bên trong của vi xử lý 8088/8086

2.2. Mô hình phần mềm của vi xử lý 8088/8086

2.3. Quản lý bộ nhớ của vi xử lý 8088/8086

2.4. Tập lệnh và các chế độ địa chỉ

Kiến trúc bên trong của 8088/8086

- Hai BXL 8088 và 8086 có cấu tạo tương tự nhau, điểm khác nhau cơ bản là:
  - 8088: Bus dữ liệu ngoài là 8 bit
  - 8086: Bus dữ liệu ngoài là 16 bit
- Hệ thống máy tính dùng 8088 chậm hơn 8086 nhưng có giá thành rẻ hơn (do dùng bus dữ liệu ngoài 8 bit nên giảm được khá nhiều chip ghép nối và bổ trợ).
- Hãng IBM đã sử dụng 8088 để thiết kế máy IBM-PC (1981).

Cấu trúc bên trong của 8088

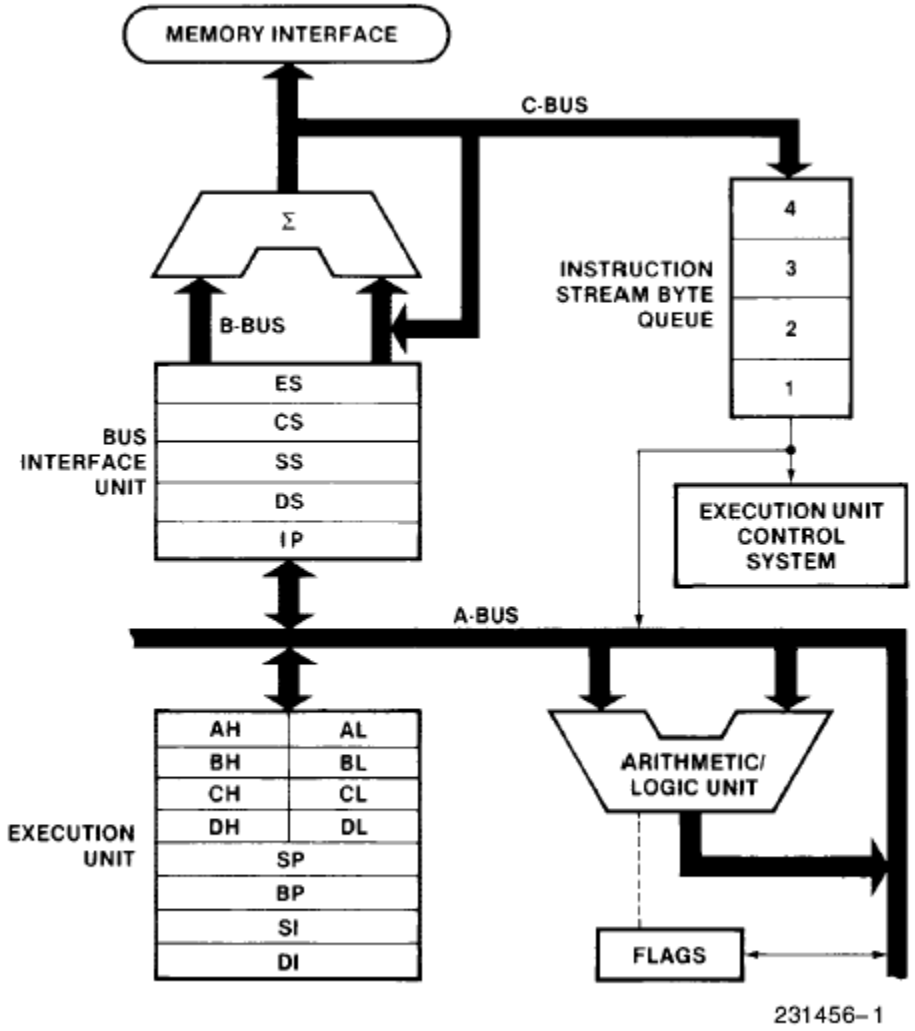


Figure 1. 8088 CPU Functional Block Diagram

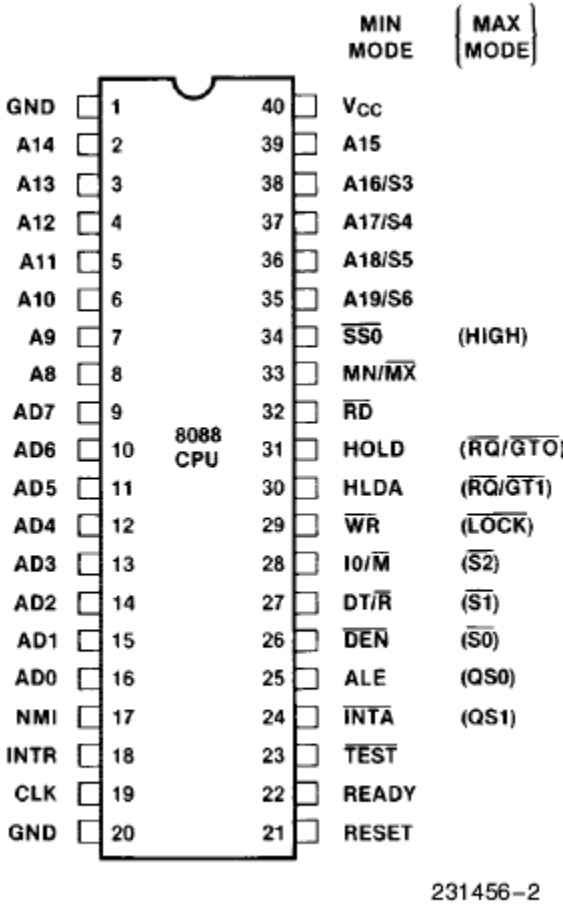


Figure 2. 8088 Pin Configuration

Cấu trúc bên trong của 8088

- Gồm 2 phần:
  - Đơn vị nối ghép bus (Bus Interface Unit – BIU)
  - Đơn vị thực hiện (Execution Unit – EU)
- Hai phần này có thể hoạt động đồng thời: trong khi EU đang thực hiện lệnh trước thì BIU đã tìm và nhận lệnh tiếp theo từ bộ nhớ chính.

Bus Interface Unit - BIU

- Bao gồm:
  - Các thanh ghi đoạn
  - Con trỏ lệnh
  - Mạch tạo địa chỉ và điều khiển bus
  - Hàng đợi lệnh (8088: 4 Byte, 8086: 6 Byte)
- Nhiệm vụ:
  - Tạo và phát địa chỉ
  - Nhận lệnh từ bộ nhớ
  - Trao đổi dữ liệu với bộ nhớ chính và cổng vào-ra
  - Phát tín hiệu điều khiển bộ nhớ và mạch vào-ra
  - Nhận các tín hiệu yêu cầu từ bên ngoài

Execution Unit – EU

- Gồm:
  - Các thanh ghi chung
  - Các thanh ghi đệm
  - Đơn vị số học và logic (ALU)
  - Khối giải mã lệnh
- Nhiệm vụ:
  - Giải mã lệnh
  - Thực hiện lệnh

Nội dung chương 2

2.1. Kiến trúc bên trong của vi xử lý 8088/8086

2.2. Mô hình phần mềm của vi xử lý 8088/8086

2.3. Quản lý bộ nhớ của vi xử lý 8088/8086

2.4. Tập lệnh và các chế độ địa chỉ

Mô hình phần mềm của 8088/8086

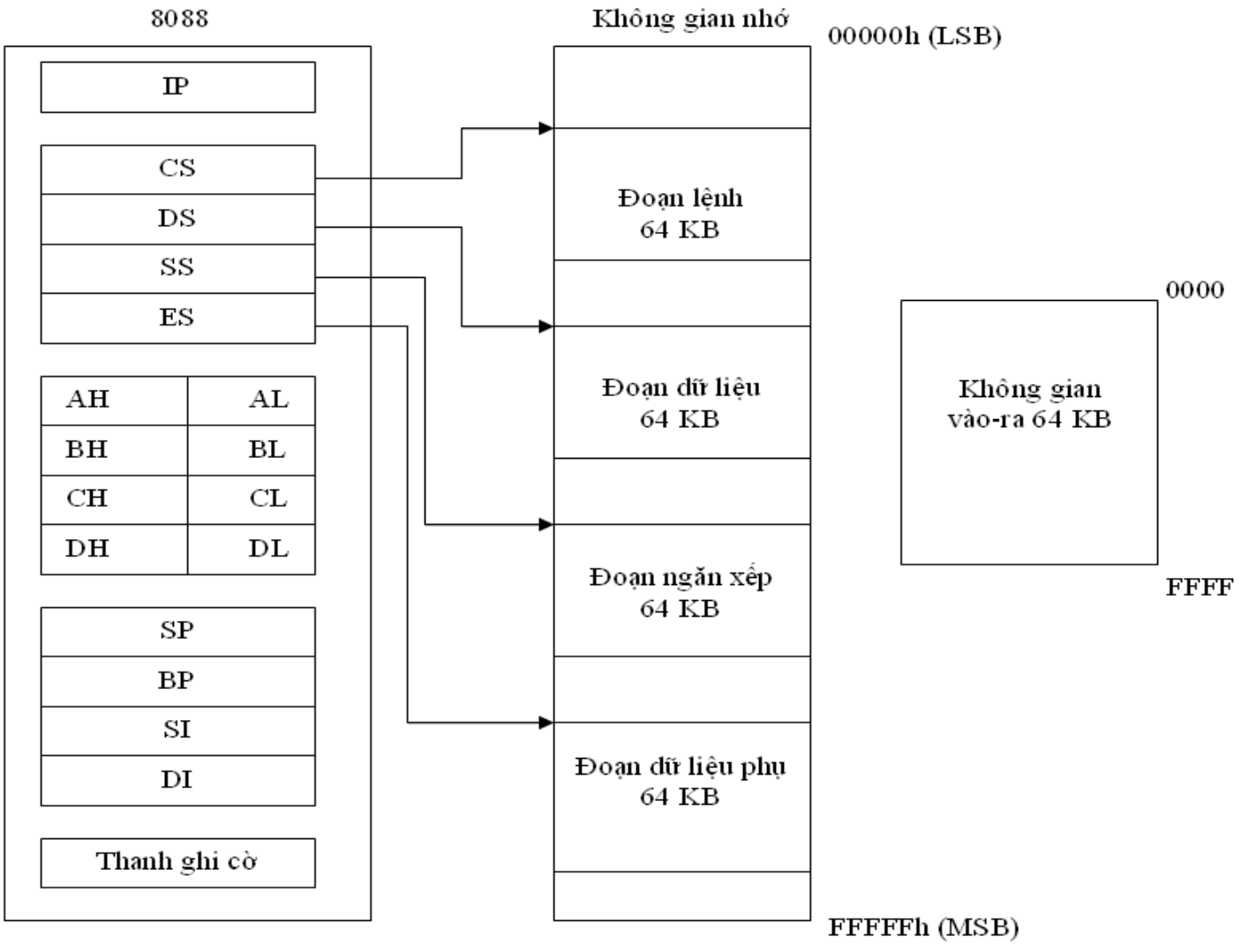
- Là mô hình mà người lập trình có thể can thiệp được.
- Bao gồm:
  - Tập thanh ghi
  - Không gian nhớ
  - Không gian vào-ra

Tập thanh ghi

- 4 thanh ghi đoạn:
  - CS (Code Segment): thanh ghi đoạn lệnh
  - DS (Data Segment): thanh ghi đoạn dữ liệu
  - SS (Stack Segment): thanh ghi đoạn ngăn xếp
  - ES (Extra Segment): thanh ghi đoạn dữ liệu phụ
- 3 thanh ghi con trỏ:
  - IP (Instruction Pointer): thanh ghi con trỏ lệnh
  - SP (Stack Pointer): con trỏ ngăn xếp
  - BP (Base Pointer): thanh ghi con trỏ cơ sở
- 4 thanh ghi dữ liệu:
  - AX (Accumulator): thanh chứa - thanh ghi tích lũy
  - BX (Base): thanh ghi cơ sở
  - CX (Count): thanh ghi đếm
  - DX (Data): thanh ghi dữ liệu

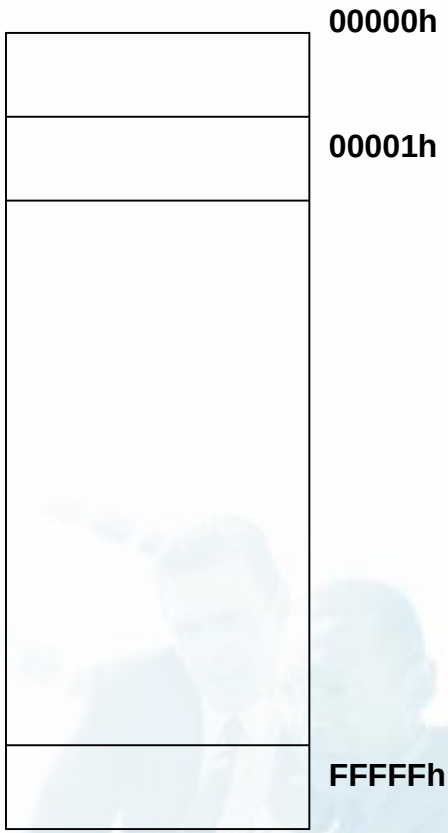
  Mỗi thanh ghi này đều có thể được chia ra thành 2 nửa có khả năng sử dụng độc lập.
- Thanh ghi cờ

Tập thanh ghi (tiếp)



Không gian nhớ

- 8088 có bus địa chỉ 20 bit $\Rightarrow$ KGĐCBN = 2^{20} byte = 1MB.
- 8088 có khả năng truy nhập bộ nhớ theo:
  - Từng byte
  - Từng word: truy nhập theo 2 byte có địa chỉ liên tiếp
- 8088 lưu trữ thông tin trong bộ nhớ chính theo kiểu đầu nhỏ (Little-endian)



Không gian vào-ra

- 8088 có khả năng quản lý không gian vào-ra 64 KB = 2^{16} Byte, do đó sẽ phải phát ra 16 bit địa chỉ để tìm cổng vào-ra tương ứng trên các chân địa chỉ từ A_0 đến A_{15}.
- Trong trường hợp phát ra 8 bit địa chỉ từ A_0 đến A_7 để xác định một cổng vào-ra thì sẽ quản lý được 256 cổng vào-ra.

Các kiểu dữ liệu

- Kiểu dữ liệu số nguyên, gồm 2 loại:
  - Không dấu:
    - ✓8 bit (1 byte), biểu diễn các số từ 0 đến 255
    - ✓16 bit (2 byte), biểu diễn các số từ 0 đến 65535
  - Có dấu:
    - ✓8 bit (1 byte), biểu diễn các số từ -128 đến 127
    - ✓16 bit (2 byte), biểu diễn các số từ -32768 đến 32767
- Kiểu dữ liệu số BCD, gồm 2 dạng: dạng nén và dạng không nén.
- Mã ASCII: tổ chức theo từng byte, theo mã 8 bit.

Nội dung chương 2

2.1. Kiến trúc bên trong của vi xử lý 8088/8086

2.2. Mô hình phần mềm của vi xử lý 8088/8086

2.3. Quản lý bộ nhớ của vi xử lý 8088/8086

2.4. Tập lệnh và các chế độ địa chỉ

2.3. Quản lý bộ nhớ

2.3.1. Các thanh ghi đoạn và phân đoạn bộ nhớ

2.3.2. Đoạn lệnh và thanh ghi con trỏ lệnh

2.3.3. Đoạn dữ liệu và các thanh ghi SI, DI, BX

2.3.4. Đoạn ngăn xếp và các thanh ghi SP, BP

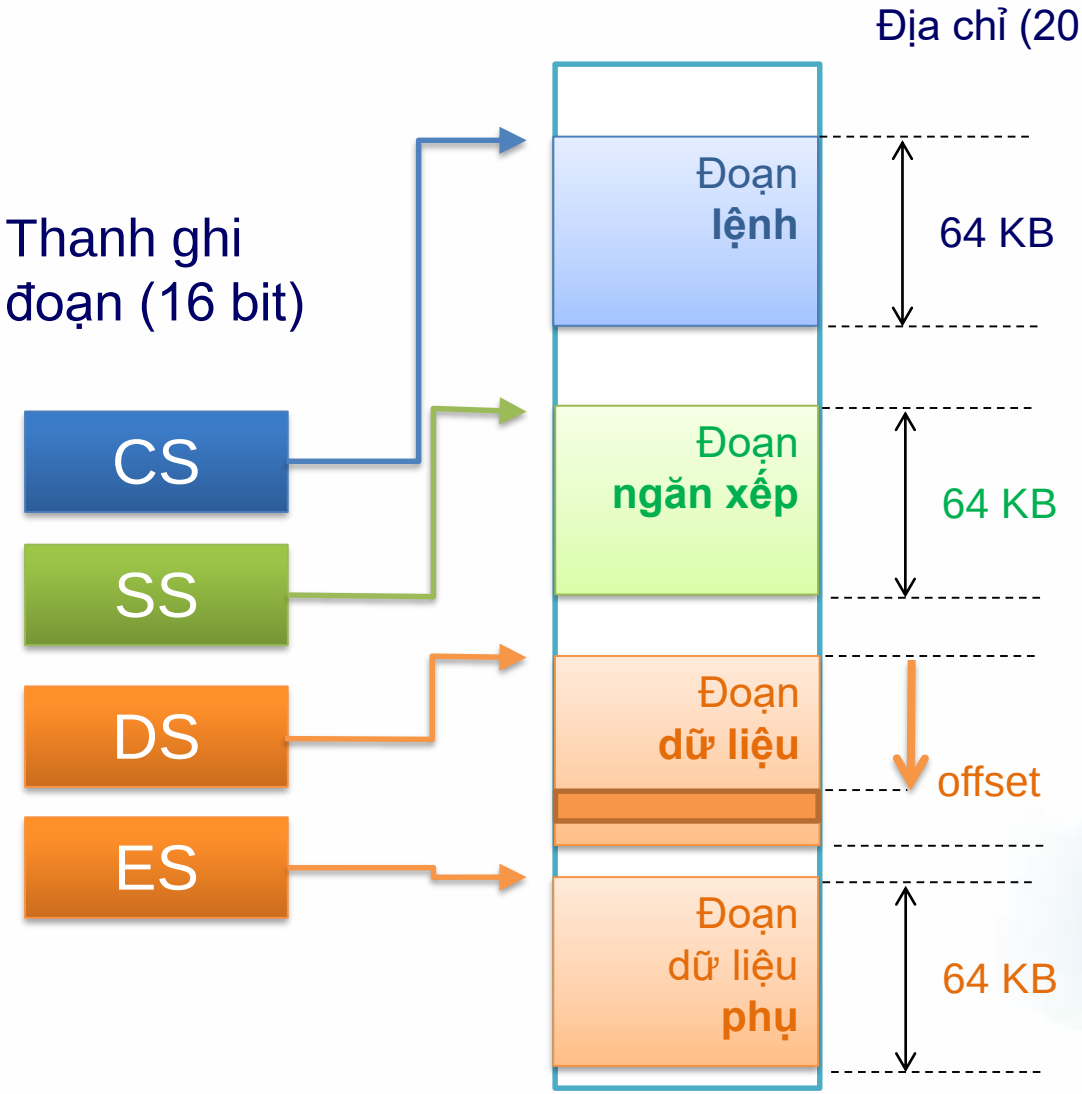
2.3.5. Các thanh ghi AX, BX, CX, DX

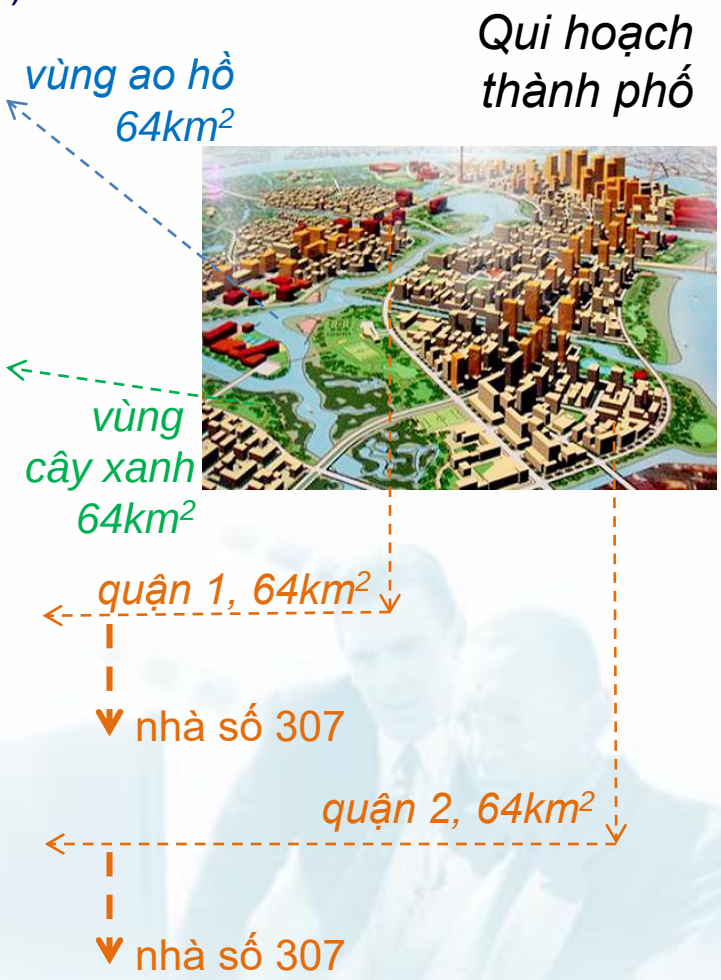
2.3.6. Thanh ghi cờ

2.3.1. Các thanh ghi đoạn và phân đoạn bộ nhớ

- 8088 có 4 thanh ghi đoạn 16 bit, do đó tại một thời điểm, 8088 chỉ làm việc được với 4 đoạn nhớ:
  - CS (Code Segment): quản lý đoạn lệnh
  - SS (Stack Segment): quản lý đoạn ngăn xếp
  - DS (Data Segment): quản lý đoạn dữ liệu
  - ES (Extra Data Segment): quản lý đoạn dữ liệu phụ
- 8088 phát ra một địa chỉ của ngăn nhớ = 20 bit, do đó không gian nhớ của nó là 1 MB (=2^{20} Byte)
- Các thanh ghi bên trong 8088 đều có độ dài là 16 bit.

Các thanh ghi đoạn





Phân đoạn bộ nhớ

- Intel chia không gian nhớ của 8088 thành các đoạn nhớ (segment) có dung lượng 64 KB =2^{16} Byte
- Địa chỉ đầu của mỗi đoạn nhớ chia hết cho 16, do đó địa chỉ đầu của một đoạn nào đó sẽ có dạng: xxxx0h (x là chữ số Hexa bất kỳ).
- Để quản lý địa chỉ đầu của một đoạn nhớ chỉ cần lưu trữ 4 số Hexa (16 bit cao), đây gọi là địa chỉ đoạn.
  - VD: nếu địa chỉ đoạn là 1234h thì địa chỉ vật lý của đầu đoạn nhớ đó là 12340h

Phân đoạn bộ nhớ (tiếp)

- Giả sử có một đoạn nhớ xác định (dung lượng tối đa = 64 KB), để xác định 1 byte nhớ cụ thể trong đoạn đó, cần biết khoảng cách (offset – độ lệch) giữa byte nhớ đó so với ngăn nhớ đầu đoạn.
- Địa chỉ logic có dạng:

  Địa chỉ đoạn (16 bit): offset (16 bit)
- Địa chỉ vật lý (20 bit) = địa chỉ đoạn * 10h + offset
- Ví dụ:

  Có địa chỉ logic 1234h:0076h

  $\Rightarrow$ địa chỉ vật lý = 1234h * 10h + 0076h = 123B6h
- Người lập trình chỉ lập trình với địa chỉ logic, còn việc chuyển sang địa chỉ vật lý là do bộ vi xử lý thực hiện.

Phân đoạn bộ nhớ (tiếp)

- Địa chỉ đoạn: xxxxh
  Địa chỉ vật lý đầu đoạn: xxxx0h
  Địa chỉ vật lý cuối đoạn: xxxx0h + FFFFh
- Địa chỉ đoạn do các thanh ghi đoạn quản lý.
- Địa chỉ offset do các thanh ghi IP, BX, BP, SP, SI, DI quản lý.
- Với một địa chỉ vật lý, có thể tìm ra nhiều địa chỉ logic khác nhau.
- Ví dụ:

  00070h = 0000h:0070h = 0001h:0060h = 0002h:0050h

...

- Thanh ghi CS sẽ xác định đoạn lệnh.
- Đoạn lệnh dùng để chứa lệnh của chương trình. Bộ vi xử lý sẽ nhận lần lượt từng lệnh ở đây để giải mã và thực hiện.
- Thanh ghi IP (con trỏ lệnh) chứa địa chỉ offset của lệnh tiếp theo sẽ được nhận vào.

  $\Rightarrow$ CS:IP chứa địa chỉ logic của lệnh tiếp theo sẽ được nhận vào.

2.3.3. Đoạn dữ liệu và các thanh ghi SI, DI, BX

- DS: quản lý một đoạn dữ liệu 64 KB
- ES: quản lý một đoạn dữ liệu phụ 64 KB
- Offset sẽ được xác định bởi nội dung của các thanh ghi SI, DI, BX.
- Sự khác nhau giữa chương trình kiểu EXE và COM:
  - Trong chương trình EXE: CS, DS và SS quản lý 3 đoạn nhớ khác nhau. Nghĩa là : CS ≠ SS ≠ DS.
  - Trong chương trình COM: CS, DS và SS có thể quản lý chung một đoạn nhớ (không lớn hơn 64 KB). Nghĩa là CS = DS = SS.

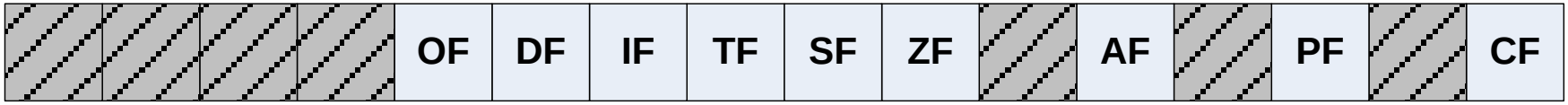
2.3.4. Đoạn ngăn xếp và các thanh ghi SP, BP

- Stack (ngăn xếp): vùng nhớ tổ chức theo cơ chế LIFO, dùng để cất giữ thông tin và có thể khôi phục lại.
- Đoạn Stack được quản lý nhờ thanh ghi SS.
- Thông tin được trao đổi với Stack theo word (16 bit).
- SP chứa địa chỉ offset của ngăn nhớ đỉnh Stack
  - Nếu cất thêm một thông tin vào Stack thì nội dung của SP giảm đi 2, sau đó cất 16 bit nội dung
  - Nếu lấy ra một thông tin của Stack thì lấy nội dung 16 bit, sau đó nội dung của SP tăng lên 2,
  - Nếu Stack rỗng thì SP trỏ vào đáy Stack
  - SS:SP chứa địa chỉ logic của ngăn nhớ đỉnh Stack
- BP là thanh ghi chứa địa chỉ offset của một ngăn nhớ nào đó trong Stack $\Rightarrow$ địa chỉ logic của ngăn nhớ đó là SS:BP
- Nếu nhảy ra, cất CS vào Stack trước rồi IP sau.

2.3.5. Các thanh ghi AX, BX, CX, DX

- AX, BX, CX, DX là các thanh ghi 16 bit
- AH, AL, BH, BL, CH, CL, DH, DL là các thanh ghi 8 bit
- Chức năng chung: chứa dữ liệu tạm thời
- Chức năng riêng:
  - AX: . Dùng cho lệnh nhân chia theo word
    . Dùng cho vào ra theo word
  - AL: . Dùng cho lệnh nhân chia theo byte
    . Dùng cho vào ra theo byte
    . Dùng cho các lệnh số học với số BCD
  - AH: . Dùng cho các lệnh nhân chia theo byte
  - BX: . Dùng để chứa địa chỉ cơ sở
  - CX: . Dùng để chứa số lần lặp của lệnh LOOP và các lệnh xử lý xâu ký tự
  - CL: . Dùng để chứa số lần dịch của lệnh dịch, lệnh quay
  - DX: . Dùng cho lệnh nhân chia theo word
    . Dùng chứa địa chỉ cổng vào ra

2.3.6. Thanh ghi cờ



- Bao gồm:
  - Các cờ phép toán: biểu thị trạng thái của kết quả phép toán.
  - Các cờ điều khiển: đặt chế độ làm việc cho bộ vi xử lý.

Các cờ phép toán

- Cờ ZF (Zero - cờ không/cờ rỗng): Được thiết lập (= 1) nếu kết quả phép toán bằng 0 và ngược lại sẽ bị xóa (=0) nếu kết quả phép toán khác 0.
- Cờ SF (Sign - cờ dấu): Được thiết lập nếu kết quả phép toán nhỏ hơn 0 và bị xoá nếu kết quả phép toán lớn hơn hoặc bằng 0.
- Cờ CF (Carry - cờ nhớ): Nếu phép cộng có nhớ ra khỏi bit cao nhất hay phép toán trừ có mượn ra khỏi bit cao nhất thì CF được thiết lập (báo tràn với số nguyên không dấu).
- Cờ OF (Overflow - cờ tràn): Nếu cộng 2 số cùng dấu mà kết quả có dấu ngược lại thì OF được thiết lập (báo tràn với số nguyên có dấu).
- Cờ PF (Parity - cờ kiểm tra chẵn lẻ): Nếu tổng số bit 1 của kết quả là chẵn thì cờ PF được thiết lập.
- Cờ AF (Auxiliary - cờ nhớ phụ): Nếu phép cộng có nhớ từ bit 3 sang bit 4 hoặc phép trừ có mượn từ bit 3 sang bit 4 thì cờ AF được thiết lập.

Các cờ điều khiển

- Cờ TF (Trap - cờ bẫy):
  - Nếu TF = 1 thì bộ vi xử lý hoạt động theo chế độ thực hiện từng lệnh (chế độ gỡ rối chương trình).
- Cờ IF (Interrupt - cờ ngắt):
  - Nếu IF = 1 thì bộ vi xử lý cho phép ngắt với yêu cầu ngắt đưa đến chân tín hiệu INTR (Interrupt Request) của bộ vi xử lý.
  - Nếu IF = 0 thì cấm ngắt.
- Cờ DF (Director - cờ hướng): chỉ hướng xử lý xâu ký tự.
  - Nếu DF = 0, xử lý từ trái sang phải.
  - Nếu DF = 1, xử lý từ phải sang trái.

- **Tập lệnh:**

  Tập lệnh của 8088 có khoảng 120 lệnh, chia thành các nhóm như sau:

  - Các lệnh chuyển dữ liệu (copy)
  - Các lệnh số học
  - Các lệnh logic, dịch, quay
  - Các lệnh xử lý xâu ký tự (string)
  - Các lệnh điều khiển hệ thống
  - Các lệnh chuyển điều khiển (rẽ nhánh)
  - Các lệnh xử lý đặc biệt
  - Các lệnh vào-ra trực tiếp

Tập lệnh và các chế độ địa chỉ (tiếp)

- **Các chế độ địa chỉ** (Addressing modes):
  - Chế độ địa chỉ là cách xác định toán hạng của lệnh
  - Toán hạng gồm: toán hạng nguồn và toán hạng đích
    - ✓Họ Intel x86: nếu trong lệnh có 2 toán hạng thì toán hạng đích được viết ở bên trái.
  - Toán hạng có thể là:
    - ✓Hằng số (được cho ngay trong lệnh)
    - ✓Nội dung của thanh ghi (trong lệnh cần cho biết tên thanh ghi)
    - ✓Nội dung của ngăn nhớ
    - ✓Nội dung của cổng vào-ra

- **Chế độ địa chỉ tức thì:**
  - Toán hạng là một giá trị hằng số nằm ngay trong lệnh.
  - Ví dụ:

    MOV CX, 5 ; nạp giá trị 5 vào thanh ghi CX

    MOV 5, CX ; chú ý: không tồn tại lệnh này !!!
- **Chế độ địa chỉ thanh ghi:**
  - Toán hạng là nội dung của 1 thanh ghi mà tên thanh ghi được cho biết ở trong lệnh.
  - Ví dụ:

    MOV AX, BX ; chuyển nội dung của BX vào AX

- **Chế độ địa chỉ trực tiếp:**
  - Toán hạng là nội dung của ngăn nhớ mà địa chỉ của ngăn nhớ đó được cho ở trong lệnh.
  - Ví dụ:

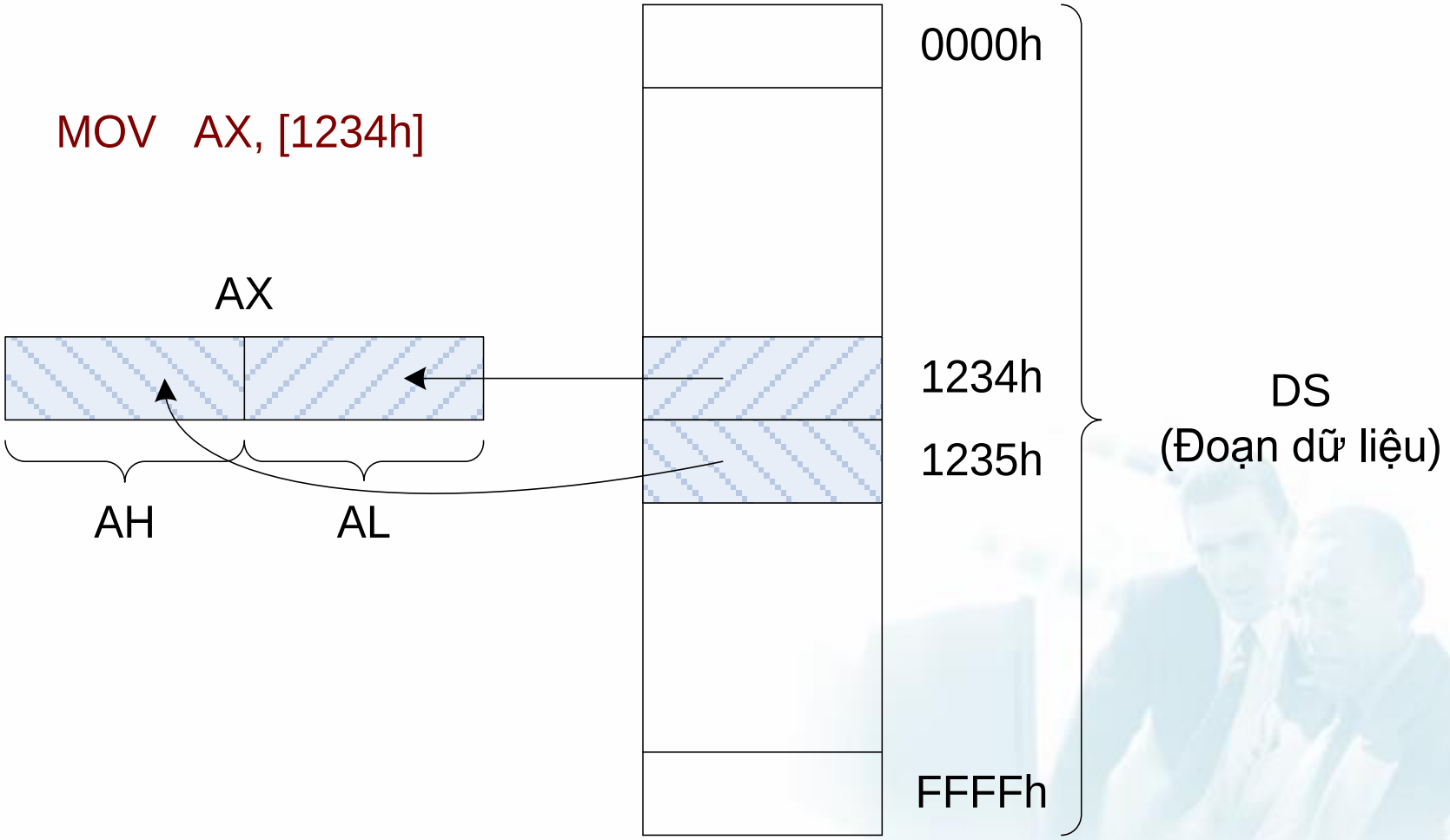
```
MOV   AX, [1234h]          ;thanh ghi đoạn DS
MOV   AX, ES:[1234h]       ;thanh ghi đoạn ES
```

✓1234h là địa chỉ offset của ngăn nhớ

✓Nếu không chỉ định địa chỉ đoạn thì ngầm định thanh ghi đoạn tương ứng là DS

✓Lệnh này chuyển 1 word nằm trong bộ nhớ bắt đầu từ địa chỉ DS:1234h vào thanh ghi AX

Minh họa chế độ địa chỉ trực tiếp

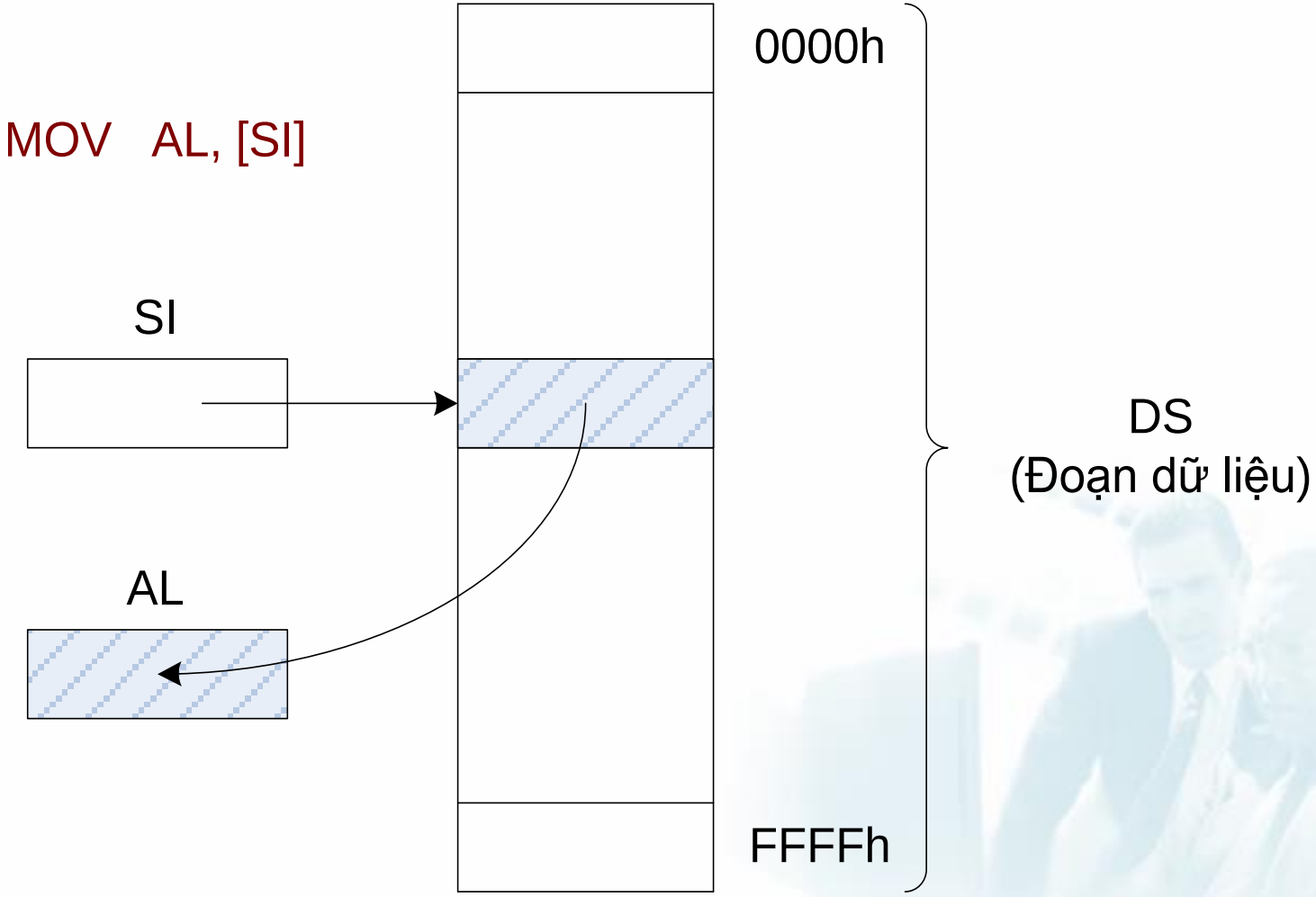


- **Chế độ địa chỉ gián tiếp qua thanh ghi:**
  - Toán hạng là nội dung của ngăn nhớ có địa chỉ offset nằm trong 1 trong các thanh ghi sau: BX, SI, DI.
  - Chú ý: nếu không chỉ định địa chỉ đoạn thì ngầm định thanh ghi đoạn tương ứng là DS.
  - Ví dụ:

```
MOV    AL, [SI]        ; tương đương  MOV   AL, DS:[SI]
```

Lệnh này chuyển 1 byte nhớ ở địa chỉ DS:SI vào thanh ghi AL

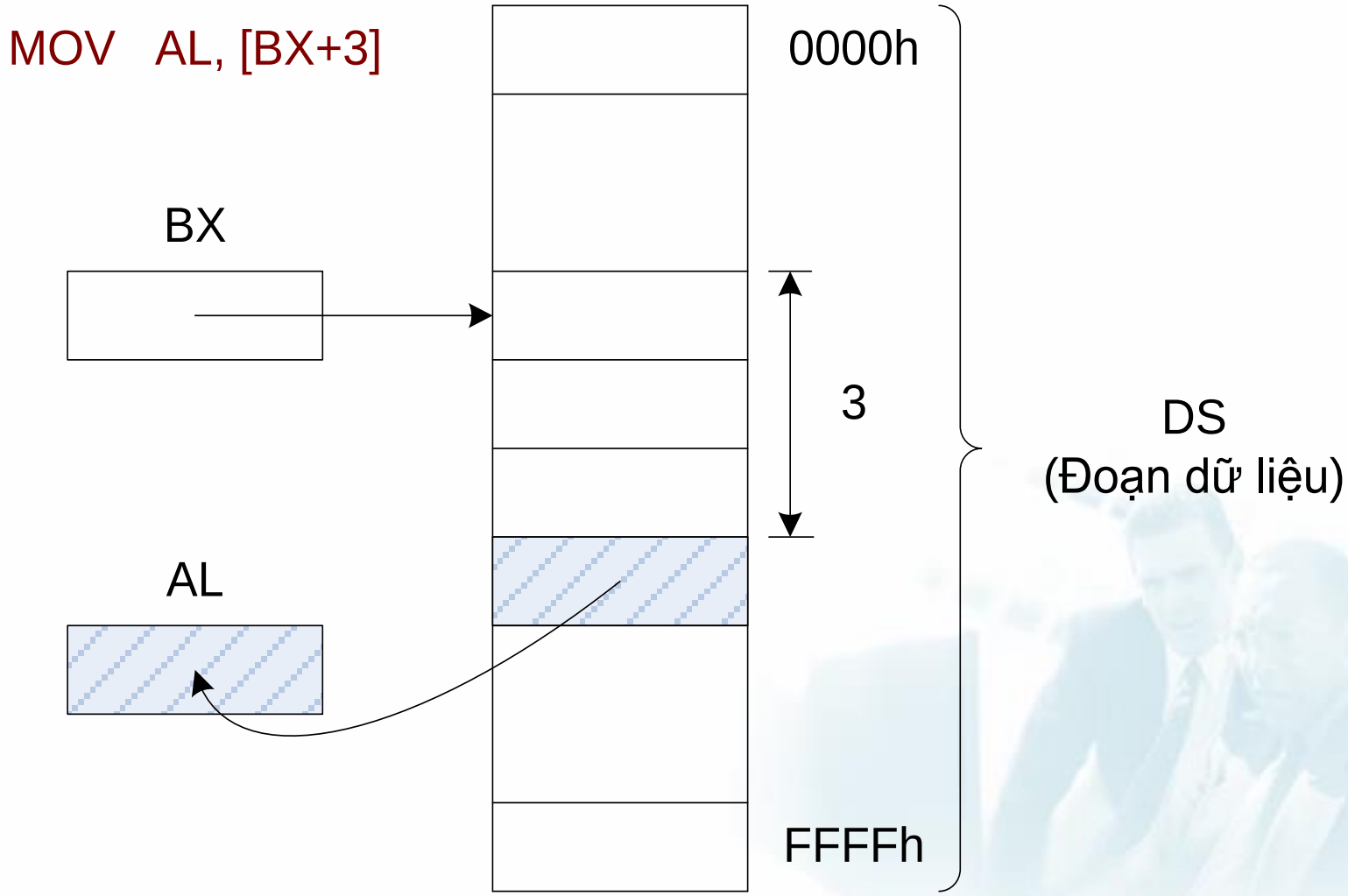
Minh họa



- **Chế độ địa chỉ cơ sở:**
  - Toán hạng là nội dung của ngăn nhớ có địa chỉ offset bằng tổng nội dung của một thanh ghi cơ sở (BX hoặc BP) + hằng số.
  - Nếu không chỉ định thanh ghi đoạn thì ngầm định thanh ghi đoạn đó là:
    - ✓DS nếu thanh ghi cơ sở là BX
    - ✓SS nếu thanh ghi cơ sở là BP
  - Ví dụ:

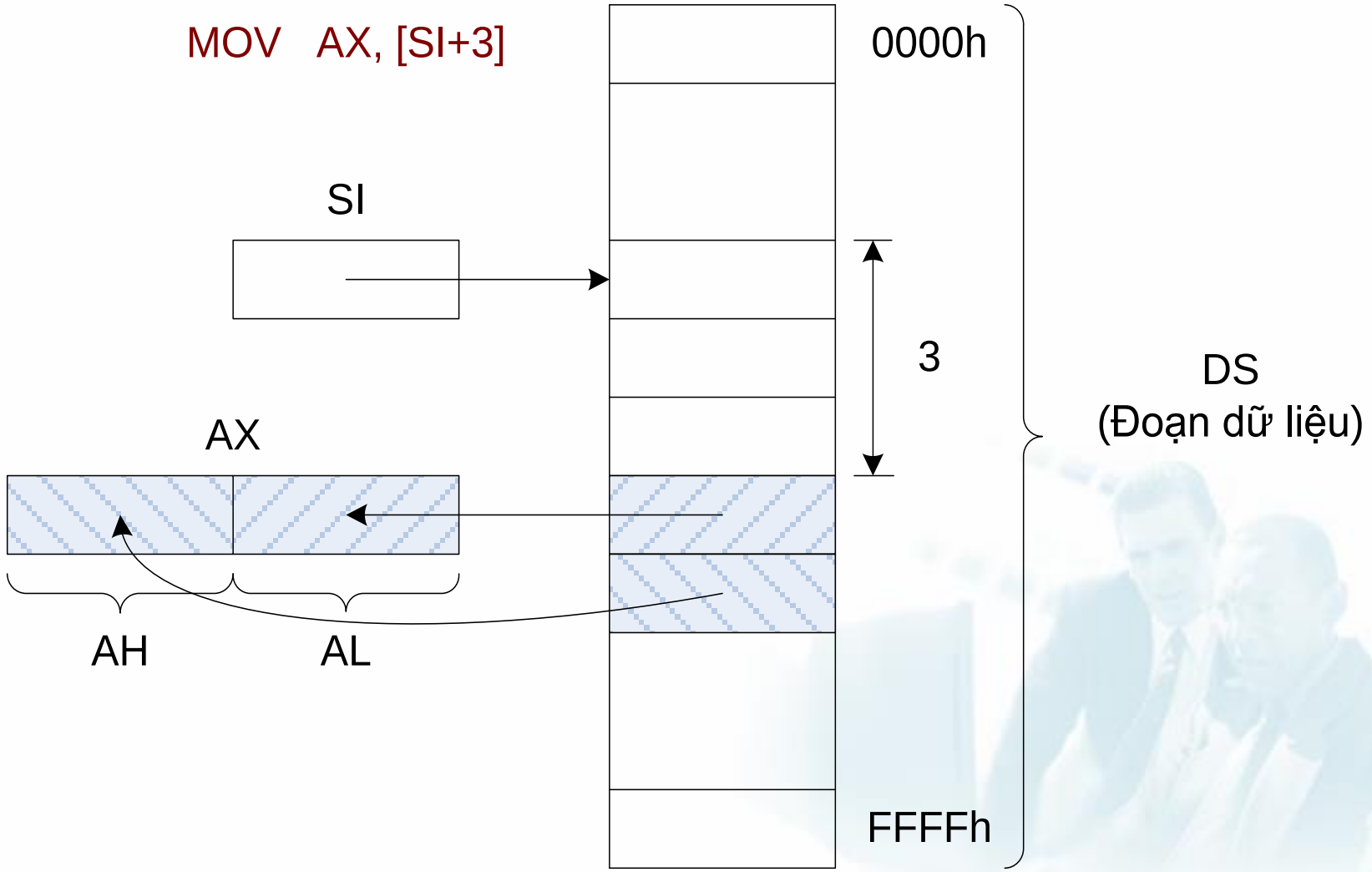
```
MOV   AL, [BX+3]    ; tương đương  MOV   AL, [BX]+3
                    ; tương đương  MOV   AL, 3[BX]
```

Minh họa



- **Chế độ địa chỉ chỉ số:**
  - Toán hạng là nội dung của ngăn nhớ có địa chỉ offset bằng tổng nội dung của một thanh ghi chỉ số (SI hoặc DI) + hằng số.
  - Nếu không chỉ định thanh ghi đoạn thì ngầm định thanh ghi đoạn đó là DS.
  - Ví dụ:

```
MOV    AX, [SI+3]     ; tương đương  MOV   AX, [SI]+3
                      ; tương đương  MOV   AX, 3+[SI]
                      ; tương đương  MOV   AX, 3[SI]
```


MOV AX, [SI+3]
0000h
SI
3
DS
(Đoạn dữ liệu)
AX
AH
AL
FFFFh

- **Chế độ địa chỉ chỉ số cơ sở:**
  - Toán hạng là ngăn nhớ có địa chỉ offset bằng tổng của nội dung một thanh ghi cơ sở (BX, BP) với một thanh ghi chỉ số (SI, DI) và một hằng số.
  - Ví dụ:

    MOV AL, [BX][SI]+4 ; ⇔ MOV AL, [BX+SI+4]

Tổng kết chế độ địa chỉ

Chế độ địa chỉ	Toán hạng	Thanh ghi đoạn ngầm định
Thanh ghi	Reg	-
Tức thì	Data	-
Trực tiếp	[Offset]	DS
Gián tiếp	[DX] [SI] [DI] [BX]	DS
Tương đối cơ sở	[BX] + disp [BP] + disp	DS SS
Tương đối chỉ số	[SI hoặc DI] + disp	DS
Tương đối chỉ số cơ sở	[BX][SI hoặc DI] + disp [BP][SI hoặc DI] + disp	DS SS

Các cặp thanh ghi đoạn:lệch ngầm định

Thanh ghi đoạn	CS	DS	ES	SS
Thanh ghi lệch	IP	BX, SI, DI	DI	SP, BP

Ghi chú: các cặp DS:SI và ES:DI dùng với các lệnh thao tác chuỗi

CHƯƠNG 3

Lập trình hợp ngữ với 8088/8086

Nội dung chương 3

Mở đầu về lập trình hợp ngữ

- Các loại ngôn ngữ lập trình
- Cú pháp của hợp ngữ
- Dữ liệu của chương trình
- Một số lệnh cơ bản
- Cấu trúc chương trình
- Chương trình EXE và COM
- Vào-ra đơn giản
- Các ví dụ
- Dịch và chạy chương trình

Các loại ngôn ngữ lập trình

- **Ngôn ngữ máy:**
  - Chỉ được biểu diễn bằng số nhị phân.
  - Bộ vi xử lý chỉ hiểu được các chương trình mã máy.
  - Con người rất khó khăn để tạo lập hay đọc hiểu chương trình ngôn ngữ máy.
- **Hợp ngữ (Assembly Language):**
  - Là ngôn ngữ lập trình bậc thấp (gần ngôn ngữ máy nhất).
  - Được xây dựng trên cơ sở ký hiệu tập lệnh của bộ vi xử lý tương ứng.
  - Phụ thuộc hoàn toàn vào bộ vi xử lý cụ thể.
- **Ngôn ngữ lập trình bậc cao:**
  - Gần với ngôn ngữ tự nhiên hơn.
  - Được xây dựng độc lập với cấu trúc của máy tính.

- Ưu điểm:
  - Can thiệp sâu vào cấu trúc hệ thống.
  - Hiểu sâu hơn về hệ thống.
  - Chương trình mã máy tương ứng sẽ ngắn hơn, thường nhanh hơn và tốn ít bộ nhớ hơn.
- Nhược điểm:
  - Khó học vì gần với mã máy.
  - Chương trình nguồn dài, không thích hợp để xây dựng những chương trình lớn.

⇒ Kết hợp ngôn ngữ lập trình bậc cao với hợp ngữ.

Chương trình dịch hợp ngữ

- Được gọi là ASSEMBLER
- Một số chương trình dịch hợp ngữ cho IBM-PC:
  - MASM – Microsoft Marco Assembler:
    - ✓Các tệp: MASM.EXE, LINK.EXE, EXE2BIN.EXE ...
  - TASM – Turbo Assembler:
    - ✓Các tệp: TASM.EXE, TLINK.EXE ...
  - 8086 Emulator → Dùng trong môn này

Các bước lập trình

- Bước 1: Phát biểu bài toán
- Bước 2: Xây dựng thuật giải
- Bước 3: Viết mã chương trình
- Bước 4: Dịch và sửa lỗi cú pháp
- Bước 5: Chạy thử và hiệu chỉnh chương trình

Cú pháp của hợp ngữ

- Chương trình hợp ngữ gồm các dòng lệnh, mỗi lệnh viết trên một dòng, mỗi dòng có thể là:
  - Lệnh của bộ vi xử lý (instruction)
  - Chỉ dẫn của chương trình dịch ASSEMBLER
- Các lệnh hợp ngữ không phân biệt chữ hoa, chữ thường.
- Khi dịch thành mã máy thì chỉ có các lệnh của bộ vi xử lý mới được dịch.
- Cấu trúc của một dòng lệnh :

  Tên **Thao tác** **Toán hạng** **Chú thích**

  (Name Operation Operand Comment)
- Giữa các trường phải có ít nhất một dấu cách (hoặc TAB)
- Ví dụ:

```
MAIN        PROC
BAT_DAU:        MOV   CX, 50        ; khoi tao bo dem
```

Ý nghĩa các trường trong lệnh

- Trường tên:
  - Sử dụng cho: nhãn lệnh, tên thủ tục, tên biến
  - Quy ước đặt tên: dài từ 1 đến 31 ký tự, cho phép sử dụng:
    - ✓Chữ cái (không phân biệt chữ hoa và chữ thường)
    - ✓Chữ số (không được dùng làm ký tự đầu tiên)
    - ✓Các ký tự khác: ?, @, $, %, . (dấu . chỉ được dùng khi nó là ký tự đầu tiên).

Ý nghĩa các trường trong lệnh (tiếp)

- Trường thao tác:
  - Nếu là lệnh của vi xử lý thì đó chính là mã lệnh (MOV, CALL, ADD,...).
  - Nếu là chỉ dẫn thì đó là lệnh giả của chương trình dịch (Pseudo-op).

Ý nghĩa các trường trong lệnh (tiếp)

- Trường toán hạng:
  - Đối với lệnh thì toán hạng xác định dữ liệu bị tác động bởi mã lệnh.
  - Một lệnh có thể có 0, 1, 2 toán hạng.
  - Ví dụ:
    - ✓MOV CX,5 ; 2 toán hạng
    - ✓INC AX ; 1 toán hạng
    - ✓NOP ; 0 toán hạng
  - Đối với lệnh giả thì toán hạng cho thêm thông tin cho lệnh giả đó.
- Trường chú thích:
  - Bắt đầu bằng dấu ";" theo sau đó là lời giải thích.

Dữ liệu của chương trình

- Hợp ngữ cho phép biểu diễn dưới dạng:
  - Số nhị phân: 1011b, 1011B, ... → ký tự ‘b’ ở cuối
  - Số thập phân: 35, 35d, 35D, ... → ký tự ‘d’ ở cuối
  - Số Hexa: 4Ah, 0ABCDh, 0FFFFH, ... → ký tự ‘h’
  - Kí tự: "A", 'HELLO', "Bach Khoa", ...
- Tất cả các kiểu dữ liệu trên sau đó đều được trình dịch Assembler dịch ra mã nhị phân.
- Mỗi kí tự được dịch thành mã ASCII tương ứng
  - Chương trình không phân biệt 'A' với 41h hay 65

Các chỉ thị giả định số liệu

Chỉ thị giả	Biểu diễn
DB	Định nghĩa byte
DW	Định nghĩa word (2 byte)
DD	Định nghĩa double word (4 byte)
DQ	Định nghĩa quadword (8 byte liên tiếp)
DT	Định nghĩa tenbyte (10 byte liên tiếp)

- **Biến Byte:**
  - Khai báo:

```
Ten_bien    DB    Gia_tri_khoi_dau
Ten_bien    DB    ?
```

  - Ví dụ:

```
Age     DB    25      ; Khởi tạo giá trị ban đầu Age = 25
Alpha   DB    ?       ; Ban đầu Alpha không xác định
```

  - Khoảng xác định của biến Byte:
    - ✓Số không dấu: [0, 255]
    - ✓Số có dấu: [-128, 127]

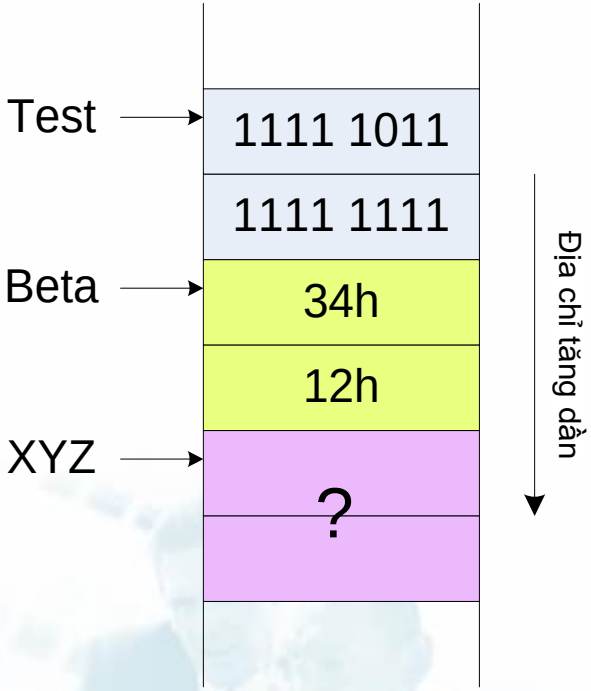
- **Biến Word:**
  - Khai báo:

    Ten_bien DW Gia_tri_khoi_dau

    Ten_bien DW ?
  - Ví dụ:

```
Test   DW   -5       ; -5 = 1111111111111011b
Beta   DW   1234h    ; 1234h = 0001001000110100b
XYZ    DW   ?
```

  - Khoảng xác định của biến Word:
    - ✓Số không dấu: [0, 65535]
    - ✓Số có dấu: [-32768, 32767]



- **Biến mảng:**
  - Mảng Byte:

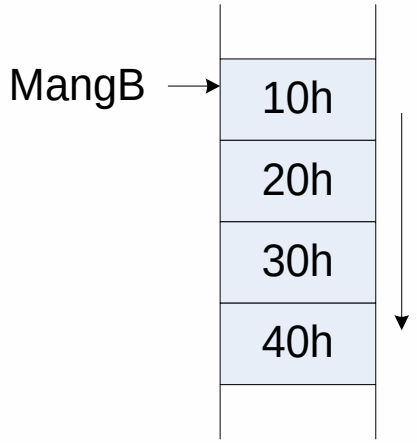
    MangB DB 10h, 20h, 30h, 40h

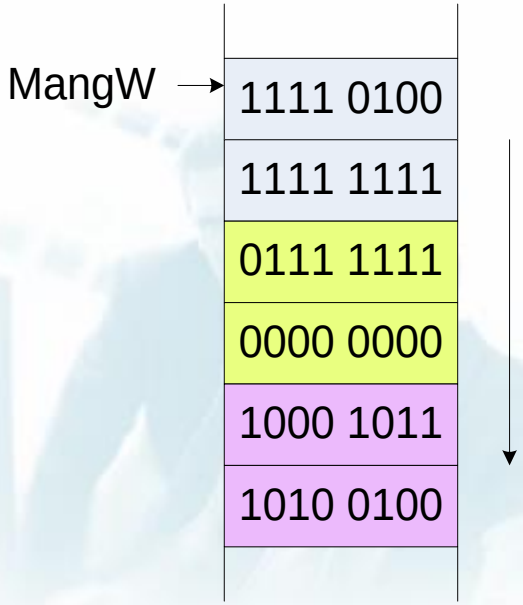
    Buffer DB 100 dup (?)
  - Mảng Word:

    MangW DW -12, 127, 0A48Bh
  - Mảng kí tự:
    - ✓Thực chất là mảng Byte
    - ✓Ví dụ: 2 cách viết sau là tương đương

    M DB 'ABC'

    M DB 41h, 42h, 43h





Khai báo hằng

- Cú pháp:

```
Ten_hang        EQU   Gia_tri
```

- Ví dụ:

```
TenTruong       EQU   'BACH KHOA'
CR              EQU   13
LF              EQU   10
...
ThongBao        DB    'DAI HOC', CR, LF, TenTruong
DoDaiChuoi      EQU   $ - offset ThongBao
```

- Hằng không được cấp phát ngăn nhớ

- Lệnh MOV (Move): MOV đích, nguồn
  - Copy dữ liệu từ toán hàng nguồn sang toán hạng đích
  - Kích thước của 2 toán hạng phải giống nhau

Ví dụ:

```
MOV     AX, BX
MOV     AL, 'A'
MOV     BH, 120

; MOV   DS, 1A000h ; SAI
MOV     AX, 0A000h
MOV     DS, AX

; MOV   Bien_2, Bien_1; SAI
MOV     AL, Bien_1
MOV     Bien_2, AL
```

Nguồn \ Đích	Thanh ghi chung	Thanh ghi đoạn	Ngăn nhớ
Thanh ghi chung	Có	Có	Có
Thanh ghi đoạn	Có	Có	Có
Ngăn nhớ	Có	Có	Không
Hằng	Có	Không	Có

- Lệnh XCHG (Exchange): XCHG đích, nguồn
  - Hoán đổi nội dung 2 toán hạng cho nhau
  - Kích thước của 2 toán hạng phải giống nhau

Ví dụ:

```
XCHG    AX, BX
XCHG    AH, Byte_1
XCHG    Word_1, BX

; XCHG  Word_1, Word_2 ; SAI
MOV     AX, Word_1
MOV     BX, Word_2
MOV     Word_1, BX
MOV     Word_2, AX
```

Đích / Nguồn	Thanh ghi chung	Ngăn nhớ
Thanh ghi chung	Có	Có
Ngăn nhớ	Có	Không

- Cú pháp:

```
ADD     đích, nguồn      ; đích ⇐ đích + nguồn
SUB     đích, nguồn      ; đích ⇐ đích - nguồn
```

```
Ví dụ:
MOV     AX, 50
MOV     BX, 30
ADD     BX, 10    ; BX = 40
SUB     AX, BX    ; AX = 10

; ADD    Byte_1, Byte_2 ; SAI
MOV     AL, Byte_1
ADD     AL, Byte_2
MOV     Byte_1, AL
```

Đích / Nguồn	Thanh ghi chung	Ngăn nhớ
Thanh ghi chung	Có	Có
Ngăn nhớ	Có	Không
Hằng	Có	Có

Các lệnh INC, DEC và NEG

- Cú pháp:

  INC đích ; đích ⇐ đích + 1

  DEC đích ; đích ⇐ đích – 1

  NEG đích ; đích ⇐ - đích (lấy bù 2 của đích)

  Toán hạng đích là thanh ghi hoặc ngăn nhớ

- Ví dụ:

```
MOV     AX, 20          ; AX = 20
INC     AX              ; AX = 21 = 0000000000010101b
NEG     AX              ; AX = 1111111111101011b
DEC     AX              ; AX = FFEAh
```

Giả sử A và B là các biến kiểu Word, hãy thực hiện các phép gán sau đây bằng hợp ngữ:

1. A = B
2. A = 10 – A;
3. A = B – A * 2;

Cấu trúc chương trình

- Chương trình mã máy khi được thực thi sẽ chiếm 3 vùng nhớ cơ bản trong bộ nhớ chính:
  - Vùng nhớ lệnh (Code)
  - Vùng dữ liệu (Data)
  - Vùng ngăn xếp (Stack)
- Chương trình hợp ngữ cũng được tổ chức tương tự như vậy.
- Mã lệnh, dữ liệu và ngăn xếp được cấu trúc như các đoạn chương trình.

Các chế độ bộ nhớ

- Kích thước của đoạn mã và dữ liệu trong chương trình được chỉ định bằng cách chỉ ra chế độ bộ nhớ nhờ chỉ thị biên dịch **.MODEL**
- Cú pháp:

  .Model Kieu_bo_nho

- Chế độ bộ nhớ thường dùng khi lập trình hợp ngữ là SMALL.

Các chế độ bộ nhớ (tiếp)

Kiểu	Mô tả
TINY	Mã lệnh và dữ liệu gói gọn trong một đoạn
SMALL	Mã lệnh trong một đoạn Dữ liệu trong một đoạn
MEDIUM	Mã lệnh chiếm nhiều hơn một đoạn Dữ liệu trong một đoạn
COMPACT	Mã lệnh trong một đoạn Dữ liệu chiếm nhiều hơn một đoạn
LARGE	Mã lệnh chiếm nhiều hơn một đoạn Dữ liệu chiếm nhiều hơn một đoạn Không có mảng nào lớn hơn 64 KB
HUGE	Mã lệnh chiếm nhiều hơn một đoạn Dữ liệu chiếm nhiều hơn một đoạn Các mảng có thể lớn hơn 64 KB

Đoạn dữ liệu (Data Segment)

- Đoạn dữ liệu chứa tất cả các khai báo biến.
- Các khai báo hằng cũng thường để ở đây.
- Để khai báo đoạn dữ liệu ta dùng chỉ thị **.DATA**
- Ví dụ:

```
.Data
Bien_1   db    10
Bien_2   dw    0FEDCh
TBao     db    'Xin chao ban', '$'
Nam      equ   2006
```

Đoạn ngăn xếp (Stack Segment)

- Cú pháp:

  .STACK Kich_thuoc

- Kich_thuoc: là số Byte của Stack (nếu không chỉ định Kich_thuoc thì ngầm định là 1KB)
- Ví dụ:

  .Stack 100h

Đoạn mã lệnh (Code Segment)

- Đoạn mã lệnh được khai báo với chỉ thị **.CODE**
- Bên trong đoạn mã, các dòng lệnh được tổ chức dưới dạng 1 chương trình chính và các chương trình con (nếu cần).
- Ví dụ:

```
.Code
Main    Proc
; các lệnh của CT chính
Main    EndP
```

Cấu trúc chương trình thông dụng

```
.Model      Small
.Stack      100h
.Data
; khai báo biến, hằng ở đây
.Code
Main Proc
; các lệnh của chương trình chính ở đây
Main EndP
; các chương trình con khác ở đây
End  Main
```

Chương trình EXE và COM

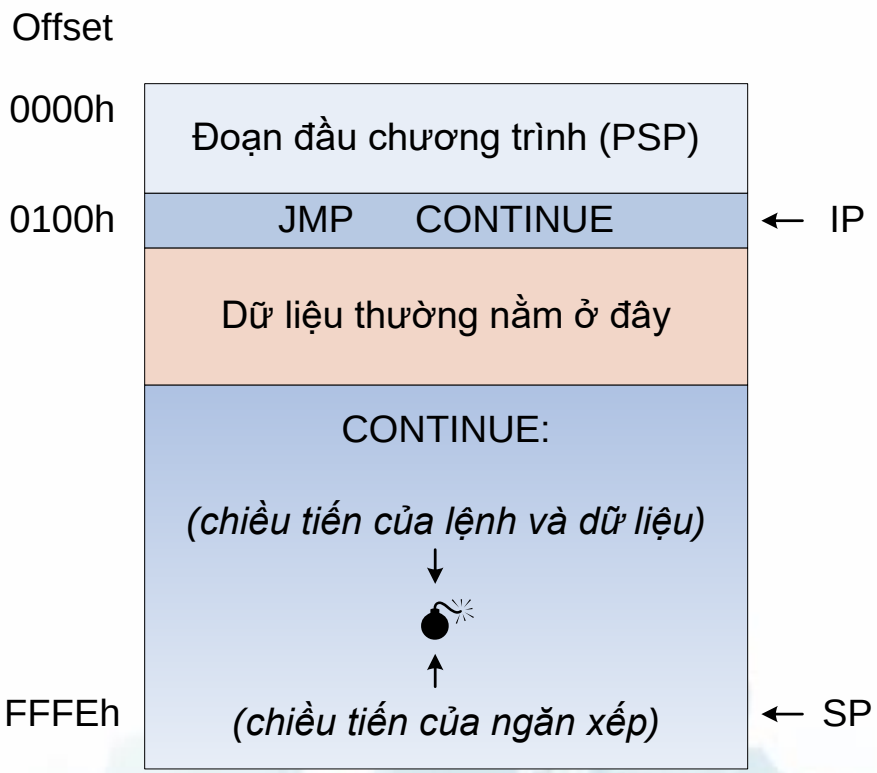
- Có 2 loại chương trình mã máy có thể thực thi được trong DOS, đó là chương trình .EXE và .COM
  - Chương trình EXE:
    - ✓ Ở đầu file chương trình có 1 vùng thông tin gọi là Header
    - ✓ Khi thực thi, CS, DS và SS trỏ đến 3 phân đoạn khác nhau
  - Chương trình COM:
    - ✓ File chương trình có kích thước nhỏ gọn (< 64KB), chứa cả mã lệnh và dữ liệu
    - ✓ Khi thực thi, CS, DS và SS trỏ đến cùng 1 phân đoạn
- DOS sẽ chọn 1 địa chỉ phân đoạn gọi là PSP (Program Segment Prefix) làm địa chỉ cơ sở để tải chương trình.
- PSP thường có kích thước là 256 Byte (=100h), chứa các thông tin liên quan đến chương trình được thực thi.

Khung chương trình EXE

```
.Model      Small
.Stack      100h
.Data
   ;   khai báo biến và hằng ở đây
.Code
Main  Proc
      mov   ax, @Data
      mov   ds, ax        ; khởi tạo DS trỏ đến đoạn Data
   ;  mov   es, ax        ; bỏ dấu ; để khởi tạo ES = DS

   ;  thân chương trình

      mov   ah, 4Ch       ; hàm thoát về DOS
      int   21h
Main  EndP
      End   Main
```

Khung chương trình COM

```
.Model      Tiny
.Code
      Org   100h
Start:
      jmp   Continue
   ;  khai báo dữ liệu ở
   đây
Continue:
Main  Proc

   ;  thân chương trình

      int   20h    ; Về DOS
Main  EndP
      End   Start
```



Vào-ra đơn giản

- CPU có thể trao đổi dữ liệu với các thiết bị ngoại qua các cổng vào-ra nhờ các lệnh IN và OUT.
- Cách vào-ra đơn giản hơn là dùng các dịch vụ ngắt có sẵn của BIOS hoặc DOS.
- Ta thường cần thực hiện các thao tác trao đổi dữ liệu với bàn phím và màn hình $\Rightarrow$ dùng hàm DOS.
- Lệnh INT (Interrupt): INT N
  - Là lệnh gọi CTC phục vụ ngắt số hiệu N (N từ 0 ÷ 255)
  - Dịch vụ ngắt số 21h chứa nhiều hàm tiện ích của DOS.

Lệnh nạp địa chỉ hiệu dụng

- Lệnh LEA (Load Effective Address):

  LEA thanh_ghi_chung, ngan_nho

  - Lấy địa chỉ offset của ngăn nhớ nạp vào thanh ghi
  - Ví dụ:

    LEA DX, Thong_Bao

    MOV DX, offset Thong_Bao ; lệnh cùng chức năng

Một số hàm vào-ra của DOS

- Khi gọi dịch vụ ngắt của DOS bằng lệnh Int 21h thì AH chứa số hiệu dịch vụ hàm.
- Hàm 01h (chờ người sử dụng vào 1 phím)
  - Vào:
    - ✓AH = 01h
  - Ra:
    - ✓AL = mã ASCII nếu 1 phím kí tự được nhấn
      = 0 nếu 1 phím điều khiển hay chức năng được nhấn
  - Ví dụ:

```
MOV   AH, 1
INT   21h
```

Một số hàm vào-ra của DOS (tiếp)

- Hàm 02h (hiện 1 kí tự hay điều khiển)
  - Vào:
    - ✓AH = 02h
    - ✓DL = mã ASCII của kí tự hiển thị hay điều khiển
  - Ra:
    - ✓AL = mã ASCII của kí tự hiển thị hay điều khiển
  - Ví dụ:

```
MOV     AH, 2
MOV     DL, 'A'     ; viết ra kí tự 'A'
INT     21h
MOV     AH, 2
MOV     DL, 10      ; điều khiển con trỏ xuống dòng
INT     21h
MOV     AH, 2
MOV     DL, 13      ; điều khiển con trỏ về đầu dòng
INT     21h
```

Một số hàm vào-ra của DOS (tiếp)

- Hàm 09h (hiện 1 chuỗi kí tự)
  - Vào:
    - ✓AH = 09h
    - ✓DS:DX = địa chỉ của chuỗi kí tự có kí tự kết thúc là '$'
  - Ra: không
  - Ví dụ:

```
ThongBao      DB      'Chao cac ban$'
; giả sử DS = địa chỉ đoạn của ThongBao
MOV   AH, 9
LEA   DX, ThongBao          ; hoặc  MOV   DX, OFFSET
  ThongBao
INT   21h
```

Các ví dụ

- Ví dụ 1: Chương trình "Hello World" bằng hợp ngữ.
- Ví dụ 2: Lập trình thực hiện các công việc sau:
  - Hiển thị thông báo : ' Hãy gõ vào một chữ cái thường: '
  - Vào chữ cái thường
  - Xuống dòng, về đầu dòng
  - Hiển thị thông báo : 'Chữ cái hoa tương ứng là: '
  - Hiển thị chữ cái hoa tương ứng
  - Thoát về DOS.

Ví dụ 1

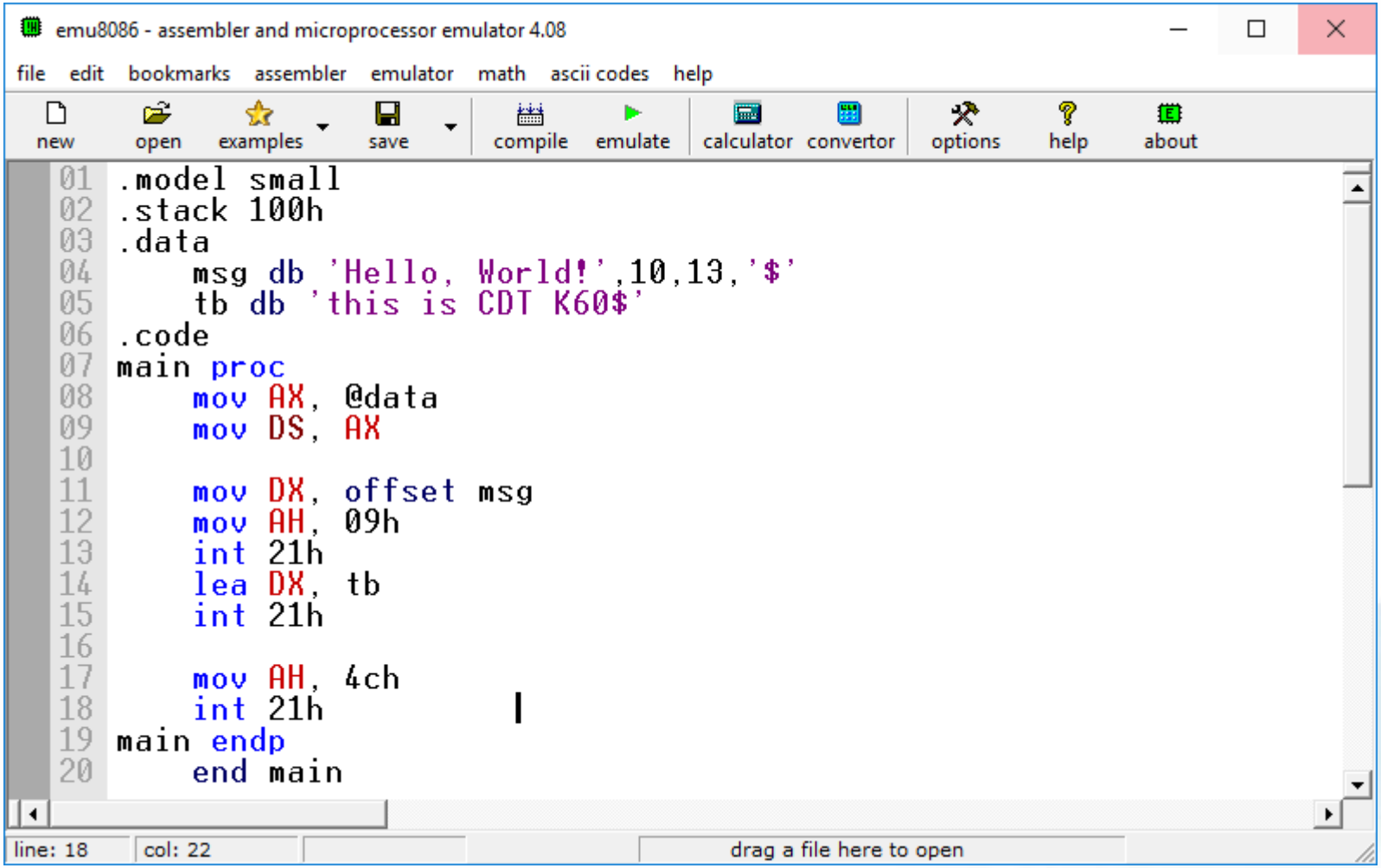
```
.Model Small
.Stack 100h
.Data
      TBao   db     'Hello World$'      ; kết thúc bằng '$'
.Code
Main   Proc
      mov    ax, @Data
      mov    ds, ax          ; DS trỏ đến đoạn Data
HienTB:
      mov    ah, 9           ; hàm hiện chuỗi
      lea    dx, TBao        ; DS:DX ⇒ chuỗi TBao
      int    21h             ; gọi hàm
Thoat:
      mov    ah,4Ch          ; hàm thoát về DOS
      int    21h
Main   EndP
      End    Main
```

Ví dụ 2

```
.Model  Small
.Stack  100
.Data
      TB1   db    'Hay go vao mot chu cai thuong: $'
      TB2   db    'Chu cai hoa tuong ung la: $'
.Code
Main  Proc
      mov   ax, @Data
      mov   ds, ax      ; DS trỏ đến đoạn Data
      mov   ah, 9       ; hàm hiển thị chuỗi
      lea   dx, TB1     ; DS:DX ⇒ chuỗi TB1
      int   21h
      mov   ah, 1       ; hàm nhập kí tự
      int   21h
      mov   bl, al      ; lưu kí tự vào BL
      mov   ah, 2       ; hiện kí tự
      mov   dl, 10      ; LF
      int   21h
      mov   dl, 13      ; CR
      int   21h
      mov   ah, 9       ; hiện chuỗi
      lea   dx, TB2     ; TB2
      int   21h
      mov   ah, 2       ; hiện kí tự
      mov   dl, bl
      sub   dl, 20h     ; ⇒ chữ HOA
      int   21h
      mov   ah,4Ch      ; về DOS
      int   21h
Main  EndP
      End   Main
```

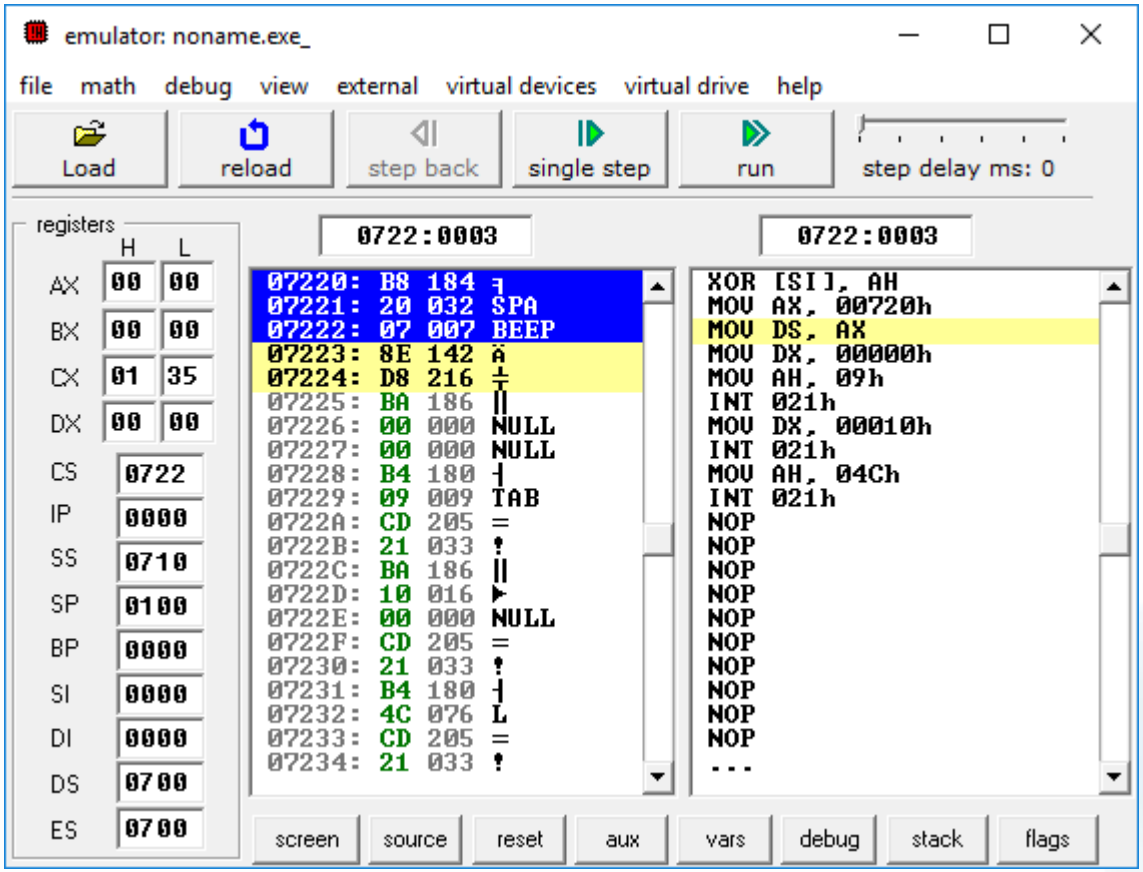
Dịch và chạy chương trình

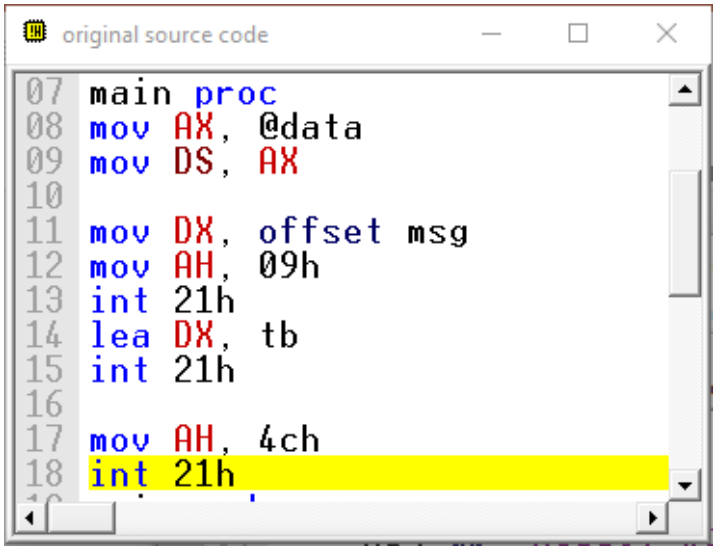
Bấm compile và emulate trên emu8086

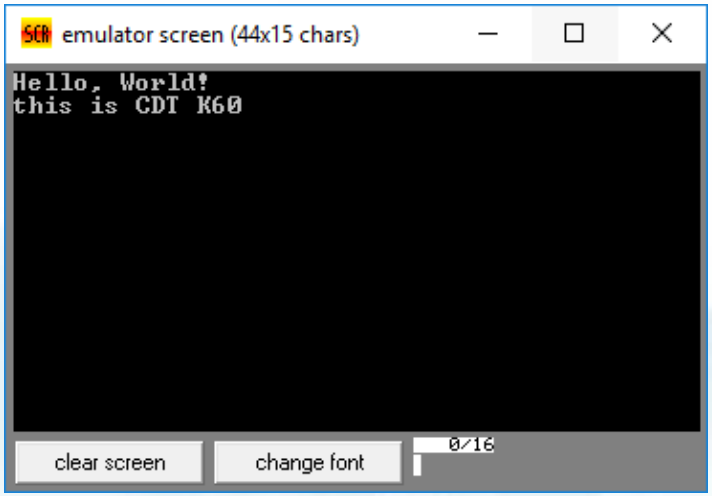


Dịch và chạy chương trình

- Các cửa sổ phụ







Bài tập 1

- Dữ liệu của 1 chương trình hợp ngữ được khai báo dưới dạng:

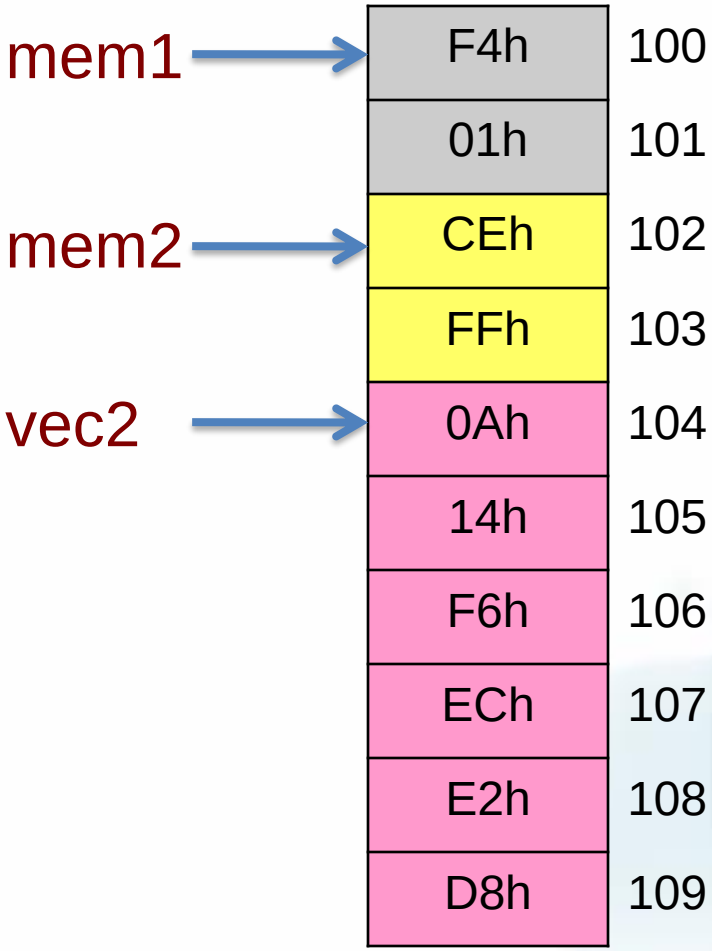
```
.data
mem1        dw      500
mem2        dw      -50
vec2        db      10, 20, -10, -20, -30, -40
```

- Hãy xác định nội dung của AX (Hexa) sau khi thực hiện đoạn lệnh sau:

```
mov         bx, 1
mov         ax, SEG vec2
mov         es, ax
mov         ax, es:[bx]
```

a) 01F4 b) 0A14 c) F4FF d) 14F6 e) CE01

Bài tập 1 (tiếp)



Bài tập 2

- Dữ liệu của 1 chương trình hợp ngữ được khai báo dưới dạng:

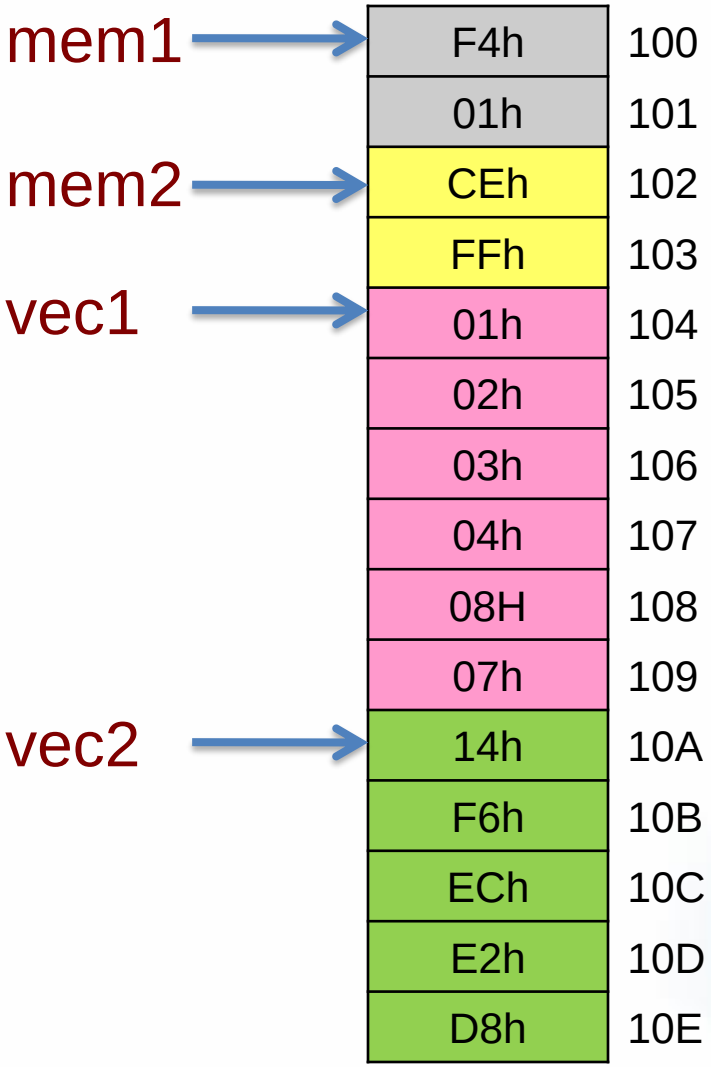
```
.data
mem1        dw      500
mem2        dw      -50
vec1        db      1, 2, 3, 4, 8, 7
vec2        db      10, 20, -10, -20, -30, -40
```

- Hãy xác định nội dung của CX (Hexa) sau khi thực hiện đoạn lệnh sau:

```
 mov                ax,@data
  mov               ds,ax
 mov                bx, OFFSET vec1
 mov                cx, 3[bx]
```

a) 0304 b) 0408 c) F3F4 d) 0203 e) 0804

Bài tập 2 (tiếp)



Nội dung chương 3

3.1. Mở đầu về lập trình hợp ngữ

3.2. Các cấu trúc lập trình với hợp ngữ

3.3. Các lệnh logic, lệnh dịch và lệnh quay

3.4. Ngăn xếp và thủ tục

3.5. Các lệnh nhân, chia

3.6. Các lệnh thao tác chuỗi

3.7. Một số ví dụ

3.2.1. Các lệnh liên quan

3.2.2. Cấu trúc điều kiện

3.2.3. Cấu trúc lặp

3.2.1. Các lệnh liên quan

- Các cấu trúc lập trình:
  - Tuần tự
  - Điều kiện
  - Lặp
- Các cấu trúc điều kiện và lặp trong hợp ngữ được tạo bởi phần lớn là các lệnh sau:
  - Lệnh so sánh CMP
  - Lệnh nhảy không điều kiện JMP
  - Các lệnh nhảy có điều kiện
  - Các câu lệnh lặp

Lệnh so sánh CMP

- Cú pháp: CMP đích, gốc
- Đích và gốc không đồng thời là ngăn nhớ, ngoài ra đích không được là hằng số.
- Lệnh CMP sẽ thực hiện trừ thử đích cho gốc (hơi giống lệnh SUB) nhưng không thay đổi giá trị của đích mà chỉ cập nhật thanh ghi cờ.
- Theo sau lệnh CMP thường là các lệnh nhảy có điều kiện.

Lệnh nhảy không điều kiện JMP

- Cú pháp: JMP Target
- Chuyển điều khiển không điều kiện đến Target
- Target có thể là nhãn lệnh, tên thanh ghi hoặc nội dung ngăn nhớ.
- Các dạng của lệnh nhảy:
  - Nhảy ngắn (short): IP ⇐ IP + (giá trị có dấu 8 bit thay cho Target)
    - ✓ JMP SHORT NhanLenh
  - Nhảy gần (near): IP ⇐ IP + (giá trị có dấu 16 bit thay cho Target)
    - ✓ JMP NEAR NhanLenh
  - Nhảy xa (far): IP ⇐ Target_Ofs; CS ⇐ Target_Seg
    - ✓ NhanLenh LABEL FAR
    - ✓ JMP FAR NhanLenh
  - Gián tiếp: IP ⇐ thanh ghi / bộ nhớ (và CS ⇐ thanh ghi / bộ nhớ)
    - ✓ JMP NEAR PTR BX ; IP ⇐ BX
    - ✓ JMP WORD PTR [BX] ; IP ⇐ [BX]
    - ✓ JMP DWORD PTR [BX] ; IP ⇐ [BX] và CS ⇐ [BX+2]

Các lệnh nhảy có điều kiện

- Có nhiều lệnh nhảy có điều kiện với cú pháp chung là:

  $JMP_{đk}$ NhanLenh

- Nhảy trong phạm vi khoảng cách từ -128 ÷ 127 byte.
- Các kí hiệu cần nhớ:
  - J : Jump (nhảy)
  - N : Not (không ...)
  - Z : cờ ZF; C : cờ CF; O : cờ OF; S : cờ SF; P : cờ PF
  - A : Above (lớn hơn – so sánh số không dấu)
  - B : Below (nhỏ hơn – so sánh số không dấu)
  - G : Greater (lớn hơn – so sánh số có dấu)
  - L : Less (lớn hơn – so sánh số có dấu)
  - E : Equal (bằng)

Nhảy hai bước

- Các lệnh nhảy có điều kiện có bước nhảy rất ngắn (khoảng cách từ -128 đến 127 byte)
- Muốn nhảy đến nhãn lệnh ở xa thì cần thực hiện qua 2 bước:
  - Bước 1: nhảy đến một nhãn lệnh trung gian ở gần đó.
  - Bước 2: từ nhãn lệnh trung gian này sử dụng lệnh JMP để nhảy đến nhãn lệnh ở xa.

Các lệnh nhảy so sánh số có dấu

Ký hiệu	Chức năng	Điều kiện nhảy
JG / JNLE	Nhảy nếu lớn hơn Nhảy nếu không nhỏ hơn hoặc bằng	ZF=0 và SF=OF
JGE / JNL	Nhảy nếu lớn hơn hoặc bằng Nhảy nếu không nhỏ hơn	SF=OF
JL / JNGE	Nhảy nếu nhỏ hơn Nhảy nếu không lớn hơn hoặc bằng	SF<>OF
JLE / JNG	Nhảy nếu nhỏ hơn hoặc bằng Nhảy nếu không lớn hơn	ZF=1 hoặc SF<>OF

Các lệnh nhảy so sánh số không dấu

Ký hiệu	Chức năng	Điều kiện nhảy
JA / JNBE	Nhảy nếu lớn hơn Nhảy nếu không nhỏ hơn hoặc bằng	ZF=0 và CF=0
JAE / JNB	Nhảy nếu lớn hơn hoặc bằng Nhảy nếu không nhỏ hơn	CF=0
JB / JNAE	Nhảy nếu nhỏ hơn Nhảy nếu không lớn hơn hoặc bằng	CF=1
JBE / JNA	Nhảy nếu nhỏ hơn hoặc bằng Nhảy nếu không lớn hơn	ZF=1 hoặc CF=1

Các lệnh nhảy điều kiện đơn

Ký hiệu	Chức năng	Điều kiện nhảy
JE / JZ	Nhảy nếu bằng Nhảy nếu bằng 0	ZF=1
JNE / JNZ	Nhảy nếu không bằng Nhảy nếu khác 0	ZF=0
JC	Nhảy nếu có nhớ	CF=1
JNC	Nhảy nếu không có nhớ	CF=0
JO	Nhảy nếu tràn	OF=1
JNO	Nhảy nếu không tràn	OF=0
JS	Nhảy nếu kết quả âm	SF=1
JNS	Nhảy nếu kết quả không	SF=0
JP / JPE	Nhảy nếu cờ chẵn	PF=1
JNP / JPO	Nhảy nếu cờ lẻ	PF=0
JCXZ	Nhảy nếu thanh ghi CX = 0	CX=0

- Lệnh LOOP:
  - Cú pháp: LOOP NhanLenh
  - Lặp lại các lệnh từ NhanLenh đến hết lệnh LOOP cho đến khi CX = 0
  - Sau mỗi lần lặp CX tự động giảm 1
  - NhanLenh cách xa lệnh LOOP không quá -128 byte
  - Thông thường CX được gán bằng số lần lặp trước khi vào vòng lặp.
  - Ví dụ:

```
        MOV   AL, 0        ; gán AL = 0
        MOV   CX, 16       ; số lần lặp
LAP:    INC   AL           ; tăng AL thêm 1
        LOOP  LAP          ; lặp 16 lần, AL = 16
```

DEC CX
JNZ NhanLenh

Các câu lệnh lặp (tiếp)

- Lệnh LOOPE / LOOPZ:
  - Cú pháp: LOOPE NhanLenh
    LOOPZ NhanLenh
  - Lặp lại các lệnh từ NhanLenh đến hết lệnh LOOPE / LOOPZ cho đến khi CX = 0 hoặc ZF = 0
  - Phạm vi tác dụng và điều kiện CX: giống lệnh LOOP
- Lệnh LOOPNE / LOOPNZ:
  - Cú pháp: LOOPNE NhanLenh
    LOOPNZ NhanLenh
  - Lặp lại các lệnh từ NhanLenh đến hết lệnh LOOPNE / LOOPNZ cho đến khi CX = 0 hoặc ZF = 1
  - Phạm vi tác dụng và điều kiện CX: giống lệnh LOOP

Ví dụ 1

- Nhận các kí tự '0' từ bàn phím cho đến khi nhận đủ 20 lần hoặc kí tự nhập vào khác '0'.
- Mã lệnh:

```
    MOV         AH, 1       ; hàm nhập kí tự
    MOV         CX, 20      ; lặp tối đa 20 lần
DocKiTu:
    INT         21h         ; nhận 1 kí tự
    CMP       AL, '0'; so sánh với '0'
    LOOPZ       DocKiTu     ; lặp lại DocKiTu
```

- Nhận các kí tự từ bàn phím cho đến khi nhận đủ 20 kí tự hoặc kí tự nhập vào là ENTER.
- Mã lệnh:

```
    MOV     AH, 1           ; hàm nhập kí
  tự
    MOV     CX, 20    ; lặp tối đa 20 lần
DocKiTu:
    INT     21h       ; nhận 1 kí tự
    CMP     AL, 13    ; so sánh với ENTER
    LOOPNZ  DocKiTu   ; lặp lại DocKiTu
```

3.2.2. Cấu trúc điều kiện

- Các cấu trúc điều kiện thông dụng:
  - IF <điều kiện> THEN <công việc>
  - IF <điều kiện> THEN <công việc 1> ELSE <công việc 2>
  - CASE <biểu thức> OF

    <giá trị 1> : <công việc 1>

    <giá trị 2> : <công việc 2>

    ....

    <giá trị N> : <công việc N>

    Else

    <công việc N+1>

    END

Cấu trúc IF ... THEN

- IF <điều kiện> THEN <công việc>
- Dạng lệnh:

```
    CMP          <???>          ; suy ra từ <điều kiện>
    JMP đksai                   BoQua
    <công việc>                  ; các lệnh thực hiện <công
    việc>
BoQua:
    ....
```

Ví dụ lệnh IF ... THEN

- Gán BX = giá trị tuyệt đối của AX
- Thuật giải:

```
BX := AX
If  BX < 0  Then
    BX := -BX
EndIf
```

- Mã lệnh:

```
    MOV   BX, AX        ; BX := AX
    CMP   BX, 0         ; tmp = BX - 0
    JNL   BoQua         ; If not (tmp < 0 ) then goto BoQua
    NEG   BX            ;       BX := - BX
BoQua:  ...             ; Endif
```

Cấu trúc IF ... THEN ... ELSE

- IF <điều kiện> THEN <công việc 1> ELSE <công việc 2>
- Dạng lệnh:

```
    CMP             <???>   ; suy ra từ <điều kiện>
    JMP đksai       Viec2
    <công việc 1>           ; các lệnh thực hiện <công việc 1>
    JMP             TiepTuc
Viec2:
    <công việc 2>           ; các lệnh thực hiện <công việc 2>
TiepTuc:
    ....
```

Ví dụ lệnh IF ... THEN ... ELSE

- AL và BL đang chứa mã ASCII của 2 kí tự. Hãy hiển thị ra màn hình kí tự có mã ASCII nhỏ hơn.

Thuật giải:

```
If AL <= BL Then
  Hiển thị kí tự trong AL
Else
  Hiển thị kí tự trong BL
EndIf
```

Mã lệnh chương trình:

```
        MOV    AH, 2          ; hàm hiện kí tự
        CMP    AL, BL         ; AL <= BL ?
        JA     Viec2          ; không đúng => tới Viec2
        MOV    DL, AL         ; chuyển kí tự trong AL vào DL
        JMP    TiepTuc        ; tới TiepTuc
Viec2:
        MOV    DL, BL         ; chuyển kí tự trong BL vào DL
TiepTuc:
        INT    21h            ; hiện kí tự trong DL
```

Cấu trúc CASE ... OF

```
   CMP            <so_sanh_1>     ; suy ra từ <biểu thức> = <giá trị 1>
   JMP đkđúng     Viec_1
   CMP            <so_sanh_2>     ; suy ra từ <biểu thức> = <giá trị 2>
   JMP đkđúng     Viec_2
   ....
   CMP            <so_sanh_N>     ; suy ra từ <biểu thức> = <giá trị N>
   JMP đkđúng     Viec_N
   <cong_viec_N+1>      ; trường hợp còn lại => thực hiện việc N+1
   JMP  TiepTuc
Viec_1:
   <cong_viec_1>        ; thực hiện việc 1
   JMP  TiepTuc
Viec_2:
   <cong_viec_2>        ; thực hiện việc 2
   JMP  TiepTuc
   ...
Viec_N:
   <cong_viec_N>        ; thực hiện việc N
TiepTuc: ...
```

- Kiểm tra nếu AX < 0 thì gán BX = -1, nếu AX = 0 thì gán BX = 0, còn nếu AX > 0 thì gán BX = 1

Thuật giải:

```
Case AX Of
  <0: BX := -1
  =0: BX := 0
Else
  BX := 1
End
```

Mã lệnh chương trình:

```
        CMP    AX, 0          ; so sánh AX với 0
        JL     SoAm           ; AX < 0 => tới SoAM
        JE     BangKhong      ; AX = 0 => tới BangKhong
        MOV    BX, 1          ; TH còn lại, gán BX = 1
        JMP    TiepTuc        ; kết thúc xử lý
SoAm:
        MOV    BX, -1         ; gán BX = -1
        JMP    TiepTuc        ; kết thúc xử lý
BangKhong:
        MOV    BX, 0          ; gán BX = 0
TiepTuc:
        ...
```

Điều kiện chứa AND

- If <điều kiện 1> AND <điều kiện 2> Then <công việc>
- Dạng lệnh:

```
    CMP          <so sánh 1>  ; suy ra từ <điều kiện 1>
    JMP đksai    BoQua
    CMP          <so sánh 2>  ; suy ra từ <điều kiện 2>
    JMP đksai    BoQua
    <công việc>               ; thực hiện <công việc>
BoQua:
    ....
```

- Đọc 1 kí tự, nếu là chữ cái hoa thì hiển thị.

Thuật giải:

```
Đọc 1 kí tự (vào AL)
If (AL >= 'A') AND (AL <= 'Z') Then
  Hiển thị kí tự trong AL
End
```

Mã lệnh chương trình:

```
        MOV   AH, 1          ; hàm đọc kí tự
        INT   21h            ; AL chứa kí tự đọc được
        CMP   AL, 'A'        ; so sánh kí tự với 'A'
        JB    BoQua          ; kí tự < 'A' => BoQua
        CMP   AL, 'Z'        ; so sánh kí tự với 'Z'
        JA    BoQua          ; kí tự > 'Z' => BoQua
        MOV   AH, 2          ; hàm hiện kí tự
        MOV   DL, AL         ; DL chứa kí tự cần hiện
        INT   21h            ; hiện kí tự trong DL
BoQua:
```

Điều kiện chứa OR

- If <điều kiện 1> OR <điều kiện 2> Then <công việc>
- Dạng lệnh:

```
    CMP          <so sánh 1>  ; suy ra từ <điều kiện 1>
    JMP đkđúng   ThucHien
    CMP          <so sánh 2>  ; suy ra từ <điều kiện 2>
    JMP đkđúng   ThucHien
    JMP          BoQua
ThucHien:
    <công việc>                ; thực hiện <công việc>
BoQua:
    ....
```

- Đọc 1 kí tự, nếu là 'y' hoặc 'Y' thì hiển thị lại, nếu không phải thì thoát chương trình.

Thuật giải:

```
Đọc 1 kí tự (vào AL)
If (AL = 'y') OR (AL = 'Y') Then
  Hiển thị kí tự trong AL
Else
  Thoát chương trình
End
```

Mã lệnh chương trình:

```
        MOV     AH, 1           ; hàm đọc kí tự
        INT     21h             ; AL chứa kí tự đọc được
        CMP     AL, 'y'         ; so sánh kí tự với 'y'
        JE      HienThi         ; kí tự = 'y' => Hiển thị
        CMP     AL, 'Y'         ; so sánh kí tự với 'Y'
        JE      HienThi         ; kí tự = 'Y' => Hiển thị
        JMP     Thoat           ; các TH khác => Thoat
HienThi:
        MOV     AH, 2           ; hàm hiện kí tự
        MOV     DL, AL          ; DL chứa kí tự cần hiện
        INT     21h             ; hiện kí tự trong DL
        JMP     TiepTuc         ; tiếp tục
Thoat:
        MOV     AH, 4Ch         ; thoát khỏi chương trình
        INT     21h
TiepTuc:
```

- Các cấu trúc lặp thông dụng:
  - FOR <số lần lặp> DO <công việc>
  - REPEAT <công việc> UNTIL <điều kiện>
  - WHILE <điều kiện> DO <công việc>

Lệnh lặp FOR

- FOR <số lần lặp> DO <công việc>
- Dạng lệnh:

```
    <Khởi tạo CX = số lần lặp>
    ; JCXZ      BoQua      ; nếu CX = 0 thì không lặp
VongLap:
    <công việc>
    LOOP        VongLap
BoQua:
```

Ví dụ

- Hiển thị ra màn hình 80 dấu '*'
- Mã lệnh:

```
    MOV     CX, 80    ; số kí tự cần hiện
    MOV     AH, 2         ; hàm hiện kí
  tự
    MOV     DL, '*'   ; kí tự cần hiện là
  *
HienSao:
    INT     21h       ; hiện kí tự
    LOOP    HienSao   ; lặp lại 80 lần
```

Lệnh lặp REPEAT ... UNTIL

- REPEAT <công việc> UNTIL <điều kiện>
- Dạng lệnh:

```
VongLap:
    <công việc>                          ; thân vòng lặp
    CMP         <so sánh>  ; suy ra từ <điều kiện>
    JMP đksai   VongLap
```

Ví dụ 1

- Đọc vào các kí tự cho đến khi gặp ENTER
- Mã lệnh:

```
    MOV     AH, 1           ; hàm đọc kí
  tự
DocKiTu:
    INT     21h         ; đọc 1 kí tự vào
  AL
    CMP     AL, 13      ; kí tự là ENTER ?
    JNE     DocKiTu     ; sai => lặp tiếp
```

Ví dụ 2 – REPEAT lồng với FOR

- Hiển thị ra 5 dòng, mỗi dòng gồm 50 dấu '*'
- Mã lệnh:

```
    MOV    BL, 5          ; số dòng cần hiện
HienDong:
    MOV    CX, 50         ; hiện 50 dấu * trên 1 dòng
    MOV    AH, 2          ; hàm hiện kí tự
    MOV    DL, '*'        ; DL = kí tự cần hiện
VietSao:
    INT    21h            ; gọi hàm hiện kí tự trong DL
    LOOP   VietSao        ; lặp lại cho đến khi đủ 50 '*'
    MOV    DL, 10         ; xuống dòng
    INT    21h
    MOV    DL, 13         ; về đầu dòng
    INT    21h
    DEC    BL             ; giảm số dòng cần phải hiện tiếp
    JNZ    HienDong       ; chưa hết 5 dòng => quay lại
```

Lệnh lặp WHILE ... DO

- WHILE <điều kiện> DO <công việc>
- Dạng lệnh:

```
VongLap:
    CMP          <so sánh>  ; suy ra từ <điều kiện>
    JMP đksai    DungLap    ; <điều kiện> sai thì dừng
    <công việc>                        ; thực hiện <công
  việc>
    JMP          VongLap    ; lặp lại
DungLap:
```

Ví dụ

- Nhập 1 dòng kết thúc bằng ENTER. Đếm số kí tự đã được nhập.
- Mã lệnh:

```
    MOV   DX, 0        ; khởi tạo bộ đếm = 0
    MOV   AH, 1        ; hàm đọc kí tự
DocKiTu:
    INT   21h          ; đọc 1 kí tự vào AL
    CMP   AL, 13       ; kí tự đó là ENTER ?
    JE    DungLap      ; đúng => DungLap
    INC   DX           ; sai => tăng bộ đếm lên 1
    JMP   DocKiTu      ; lặp lại
DungLap:
```

Nội dung chương 3

3.1. Mở đầu về lập trình hợp ngữ

3.2. Các cấu trúc lập trình với hợp ngữ

3.3. Các lệnh logic, lệnh dịch và lệnh quay

3.4. Ngăn xếp và thủ tục

3.5. Các lệnh nhân, chia

3.6. Các lệnh thao tác chuỗi

3.7. Một số ví dụ

3.3. Các lệnh logic, dịch và quay

3.3.1. Các lệnh logic

3.3.2. Các lệnh dịch

3.3.3. Các lệnh quay

3.3.4. Vào-ra số nhị phân và Hexa

- Các phép toán logic:

a	b	a AND b	a OR b	a XOR b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

a	NOT a
0	1
1	0

- Các lệnh logic: AND, OR, XOR, NOT, TEST

Các lệnh AND, OR và XOR

- Cú pháp:
  - AND đích, nguồn ; đích ⇐ đích AND nguồn
  - OR đích, nguồn ; đích ⇐ đích OR nguồn
  - XOR đích, nguồn ; đích ⇐ đích XOR nguồn
  - TEST đích, nguồn ; Phép AND nhưng không thay đổi đích
- Chú ý:
  - Toán hạng nguồn: hằng số, thanh ghi hay ngăn nhớ
  - Toán hạng đích: thanh ghi hay ngăn nhớ
  - Hai toán hạng không được đồng thời là ngăn nhớ
- Ảnh hưởng tới các cờ:
  - SF, ZF, PF phản ánh kết quả của lệnh
  - AF không xác định
  - CF = OF = 0

Các ví dụ

- VD 1: Đổi mã ASCII của 1 chữ số thành số tương ứng.
  - Giả sử AL chứa kí tự (chẳng hạn '5' – mã ASCII là 35h)
  - Cần chuyển AL về giá trị chữ số (là 5)
  - Thực hiện: SUB AL, 30h hoặc AND AL, 0Fh
- VD 2: Đổi chữ thường thành chữ hoa.
  - Giả sử DL chứa kí tự chữ thường, cần chuyển về chữ hoa.
  - Thực hiện: SUB DL, 20h hoặc AND DL, 0DFh
- VD 3: Xóa thanh ghi AX về 0.
  - Thực hiện: XOR AX, AX
- VD 4: Kiểm tra xem AX có bằng 0 hay không?
  - Thực hiện: OR AX, AX ; AX = 0 $\Leftrightarrow$ ZF = 1

Lệnh NOT

- Cú pháp: NOT đích
- Lệnh này không ảnh hưởng đến cờ

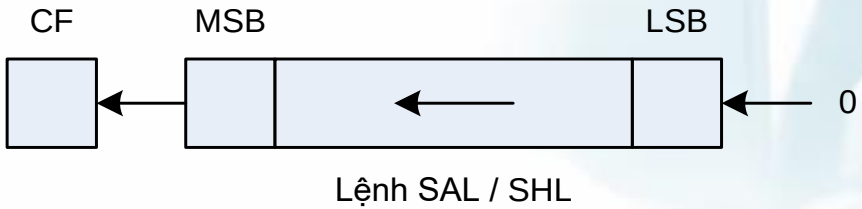
- Cú pháp: TEST đích, nguồn
- Thực hiện phép toán AND nhưng không thay đổi đích mà chỉ cập nhật các cờ.
- Các cờ bị tác động:
  - SF, ZF, PF phản ánh kết quả của lệnh
  - AF không xác định
  - CF = OF = 0
- Ví dụ: Kiểm tra tính chẵn lẻ của AL
  - AL chẵn ⇔ bit LSB của = 0
  - Thực hiện: TEST AL, 1 ; AL chẵn ⇔ ZF = 1

3.3.2. Các lệnh dịch

- Các lệnh dịch và quay có 2 dạng:
  - Dịch (hoặc quay) 1 vị trí:
    Lệnh đích, 1
  - Dịch (hoặc quay) N vị trí:
    Lệnh đích, CL ; với CL = N

Các lệnh dịch trái

- Dịch trái số học (SAL – Shift Arithmetically Left) và dịch trái logic (SHL – Shift (Logically) Left):
  - SAL đích, 1 hoặc SAL đích, CL
  - SHL đích, 1 hoặc SHL đích, CL
- Lệnh SAL và SHL là tương đương
- Tác động vào các cờ:
  - SF, PF, ZF phản ánh kết quả
  - AF không xác định
  - CF chứa bit cuối cùng được dịch ra khỏi đích
  - OF = 1 nếu kết quả bị thay đổi dấu trong phép dịch cuối cùng

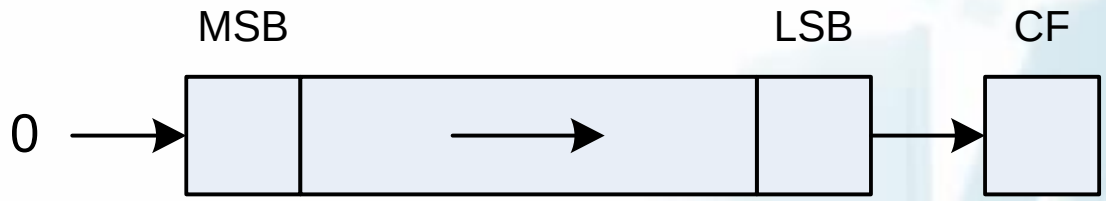


Lệnh SAL / SHL

- Dịch phải logic: SHR – Shift (Logically) Right
  - Cú pháp:

    SHR đích, 1

    SHR đích, CL
  - Các cờ bị tác động như là lệnh dịch trái
  - Minh họa:

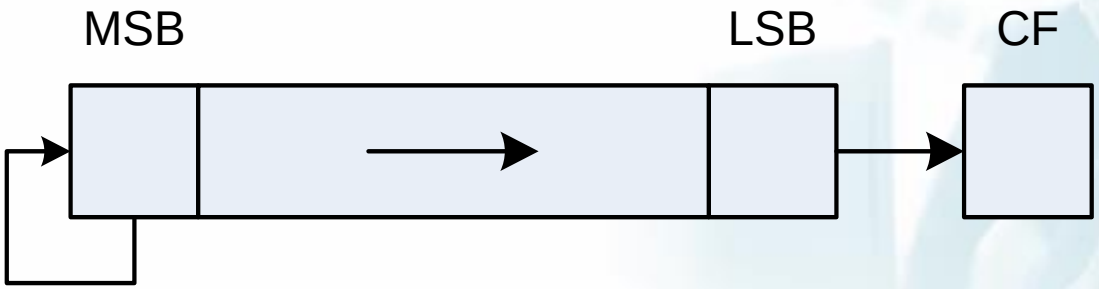


Các lệnh dịch phải (tiếp)

- Dịch phải số học: SAR – Shift Arithmetically Right
  - Cú pháp:

    SAR đích, 1

    SAR đích, CL
  - Các cờ bị tác động như là lệnh SHR
  - Minh họa:

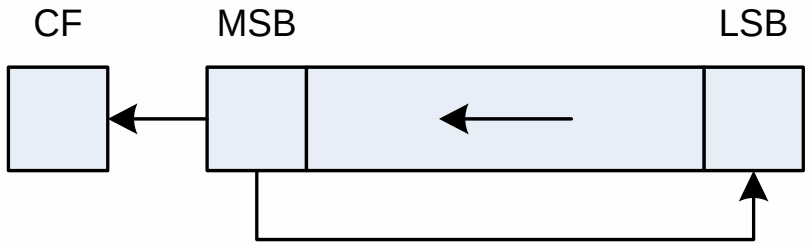


3. Các lệnh quay

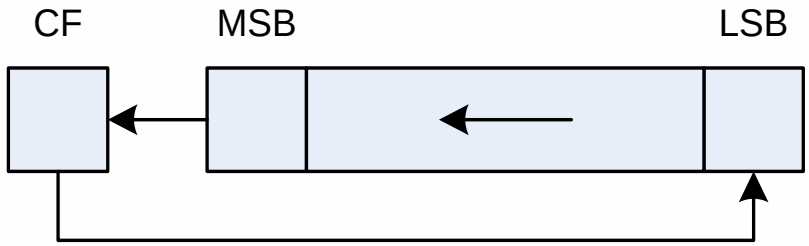
- Các dạng lệnh quay:
  - ROL : quay trái
  - ROR : quay phải
  - RCL : quay trái qua cờ nhớ
  - RCR : quay phải qua cờ nhớ
- Tác động vào các cờ:
  - SF, PF, ZF phản ánh kết quả
  - AF không xác định
  - CF chứa bit cuối cùng bị dịch ra khỏi toán hạng
  - OF = 1 nếu kết quả bị thay đổi dấu trong lần quay cuối cùng

Minh họa các lệnh quay

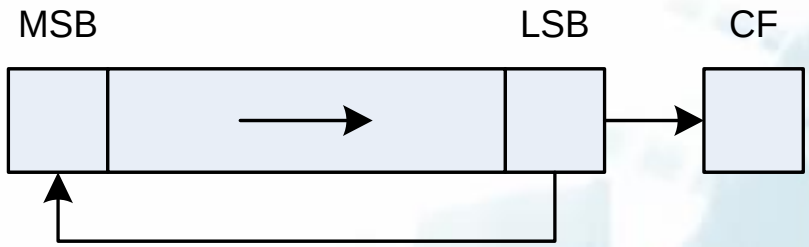
Lệnh ROL



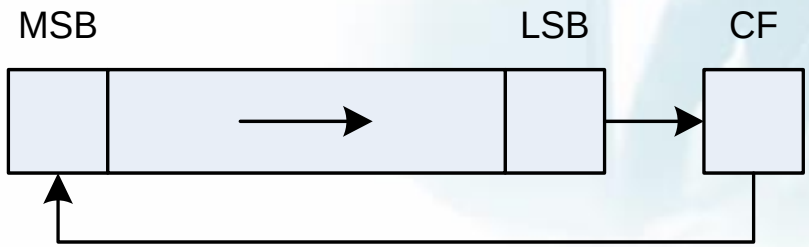
Lệnh RCL



Lệnh ROR



Lệnh RCR



- Xác định giá trị của AX và BX dưới dạng Hexa sau khi thực hiện đoạn chương trình sau:

```
    MOV     CX, 16
    MOV     AX, 5A6Bh
LAP:
    ROL     AX, 1
    RCR     BX, 1
    LOOP    LAP
```

3.3.4. Vào-ra số nhị phân và Hexa

- Các thao tác:
  - Nhập số nhị phân
  - In số nhị phân
  - Nhập số Hexa
  - In số Hexa

Nhập số nhị phân

- Đọc các bit nhị phân từ bàn phím (kết thúc nhập bằng ENTER), chuyển thành số nhị phân rồi lưu vào BX.
- Thuật giải:

Xóa BX (là thanh ghi chứa kết quả)

FOR 16 lần DO

Đọc ký tự (‘0’ hoặc ‘1’)

Nếu phím Enter được nhấn -> break

Chuyển ký tự sang mã nhị phân

Dịch trái BX

Chèn mã nhị phân vào bit LSB của BX

In số nhị phân

- In giá trị ở BX ra màn hình dưới dạng số nhị phân.
- Thuật giải:

```
FOR 16 lần DO
    Quay trái BX (bit MSB của BX được đưa ra CF)
    IF CF = 1 THEN
            Đưa ra '1'
    ELSE
            Đưa ra '0'
    END IF
END FOR
```

- Có thể dùng lệnh ADC: ADC đích, nguồn

  đích ⇐ đích + nguồn + CF

Nhập số Hexa

- Đọc các kí tự Hexa từ bàn phím (tối đa 4 chữ số, chỉ nhập các chữ số và các chữ cái hoa, kết thúc nhập bằng ENTER). Chuyển thành số Hexa tương ứng rồi lưu vào BX.
- Thuật giải:

Xóa BX (là thanh ghi chứa kết quả)

FOR 4 lần DO

Nhập ký tự (0 đến 9, A đến F)

Phím Enter được nhấn ->break

Đổi kí tự ra nhị phân

Dịch trái BX 4 lần

Chèn giá trị mới vào 4 bit thấp nhất của BX

In số Hexa

- Đưa giá trị Hexa 4 chữ số trong BX ra màn hình.
- Thuật giải:

```
FOR 4 lần DO
    Chuyển BH vào DL (giá trị cần in nằm trong BX)
    Dịch phải DL 4 vị trí
    IF DL < 10 THEN
            Đổi thành kí tự '0' ... '9'
    ELSE
            Đổi thành kí tự 'A' ... 'F'
    END IF
    Đưa kí tự ra
    Quay trái BX 4 vị trí
END FOR
```

Nội dung chương 3

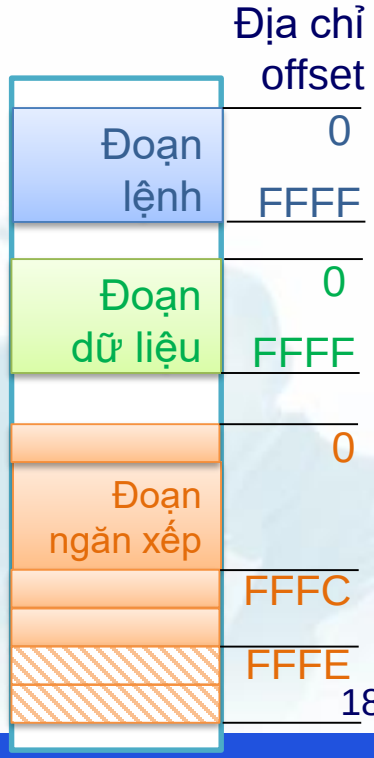
3.4.1. Ngăn xếp

3.4.2. Thủ tục

3.4.3. Các ví dụ

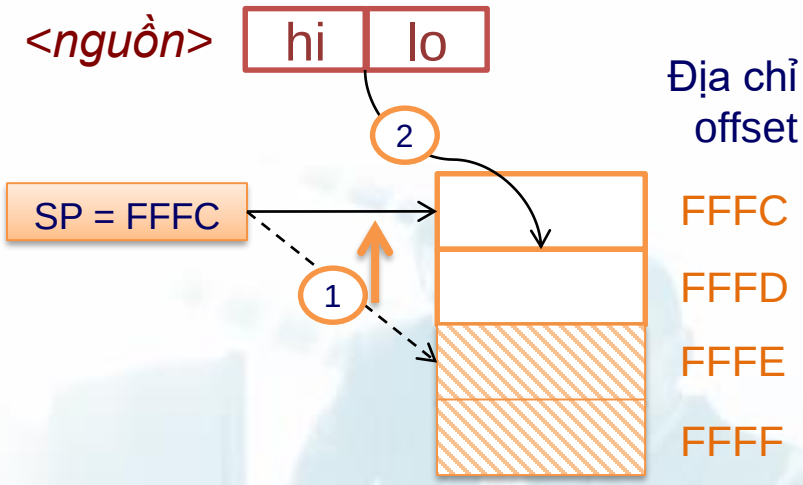
3.4.1. Ngăn xếp

- Ngăn xếp (Stack):
  - Vùng nhớ tổ chức theo cấu trúc LIFO dùng để cất giữ thông tin.
  - Chiều của Stack từ đáy lên đỉnh ngược với chiều tăng của địa chỉ.
- Khai báo ngăn xếp: .STACK kich_thuoc
- Ví dụ: .Stack 100h
  - Khi chương trình được thực thi thì:
    - ✓SS : chứa địa chỉ đoạn ngăn xếp
    - ✓SP : chứa địa chỉ offset của đỉnh ngăn xếp. Ban đầu ngăn xếp rỗng nên FFFEh cũng là địa chỉ của đáy ngăn xếp



Lệnh PUSH và PUSHF

- Lệnh PUSH dùng để cất 1 dữ liệu 16 bit vào trong ngăn xếp.
- Cú pháp: PUSH nguồn
  - nguồn là 1 thanh ghi 16 bit hoặc 1 từ nhớ (2 Byte)
- Các bước thực hiện:
  1. SP ⇐ SP – 2
  2. Một bản sao của toán hạng nguồn được chuyển vào địa chỉ xác định bởi SS:SP (toán hạng nguồn không đổi)



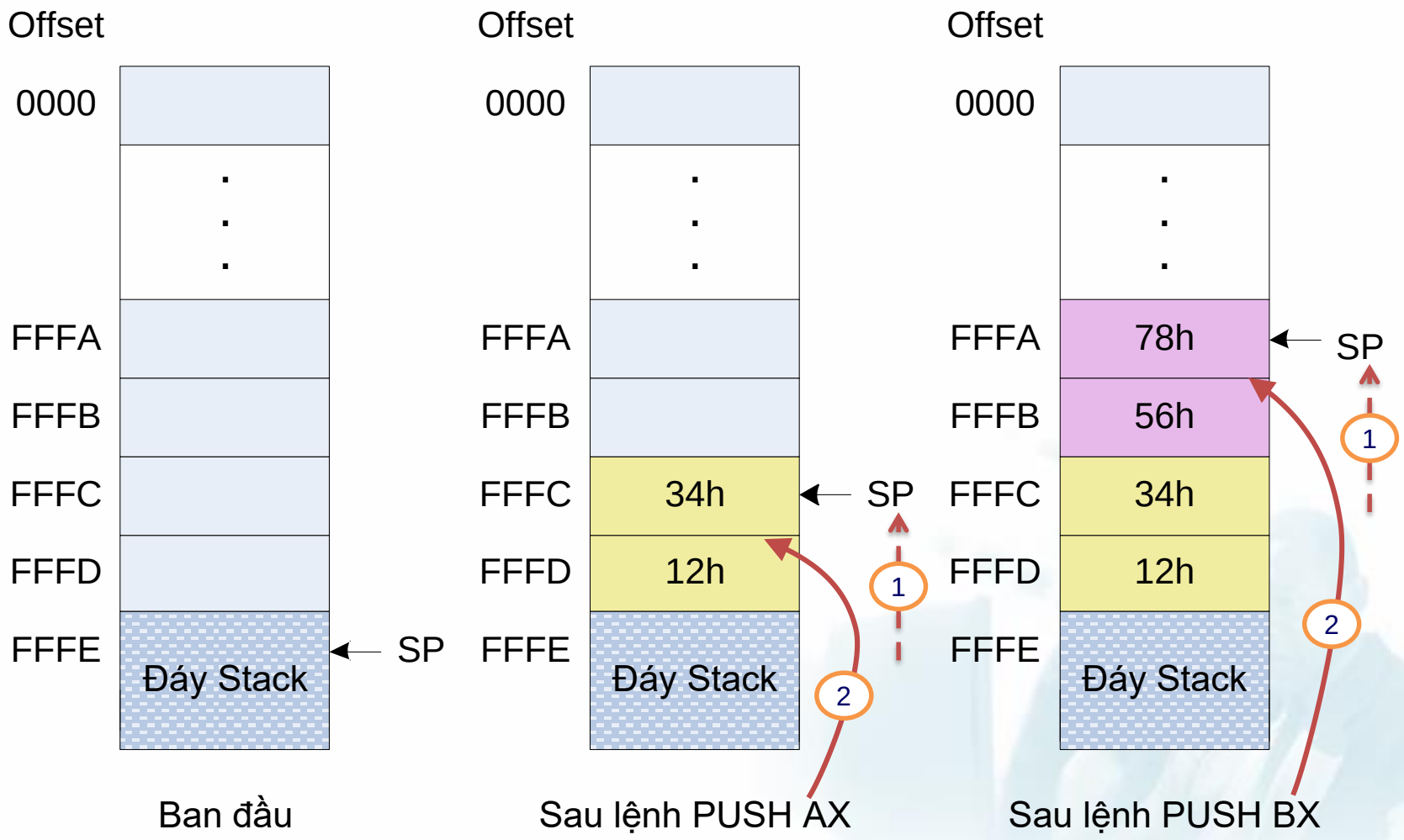
- Lệnh PUSHF cất nội dung của thanh ghi cờ vào trong ngăn xếp.

Ví dụ về lệnh PUSH

- Xác định nội dung các Byte nhớ trong Stack.

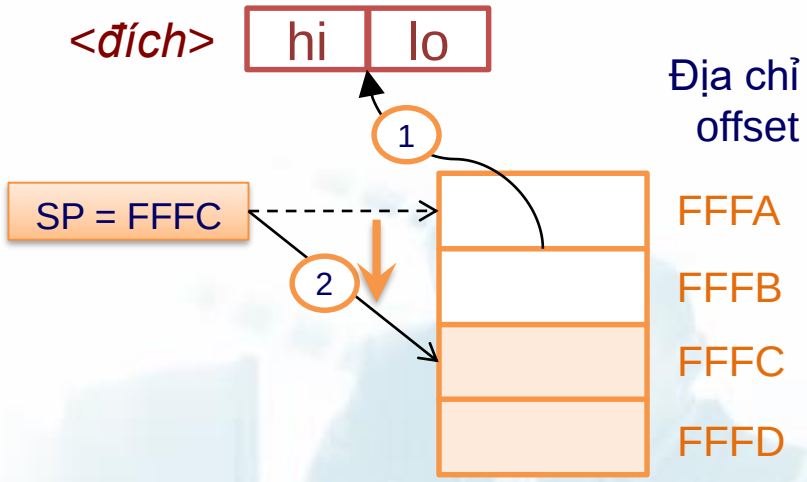
```
.Stack    100h
...
Start:    ; lệnh đầu tiên của chương trình
  MOV     AX, 1234h
  MOV     BX, 5678h
  PUSH    AX
  PUSH    BX
```

Ví dụ (tiếp)


Offset
0000
FFFA
FFFB
FFFC
FFFD
FFFE
Đáy Stack
SP
Ban đầu
Offset
0000
FFFA
FFFB
FFFC
34h
FFFD
12h
FFFE
Đáy Stack
SP
1
2
Sau lệnh PUSH AX
Offset
0000
FFFA
78h
FFFB
56h
FFFC
34h
FFFD
12h
FFFE
Đáy Stack
SP
1
2
Sau lệnh PUSH BX

Lệnh POP và POPF

- Lệnh POP dùng để lấy ra 1 từ dữ liệu bắt đầu từ đỉnh ngăn xếp.
- Cú pháp: POP đích
  - đích là 1 thanh ghi 16 bit hoặc 1 từ nhớ
- Các bước thực hiện:
  1. Nội dung của từ nhớ ở địa chỉ SS:SP được chuyển tới toán hạng đích.
  2. SP $\Leftarrow$ SP + 2



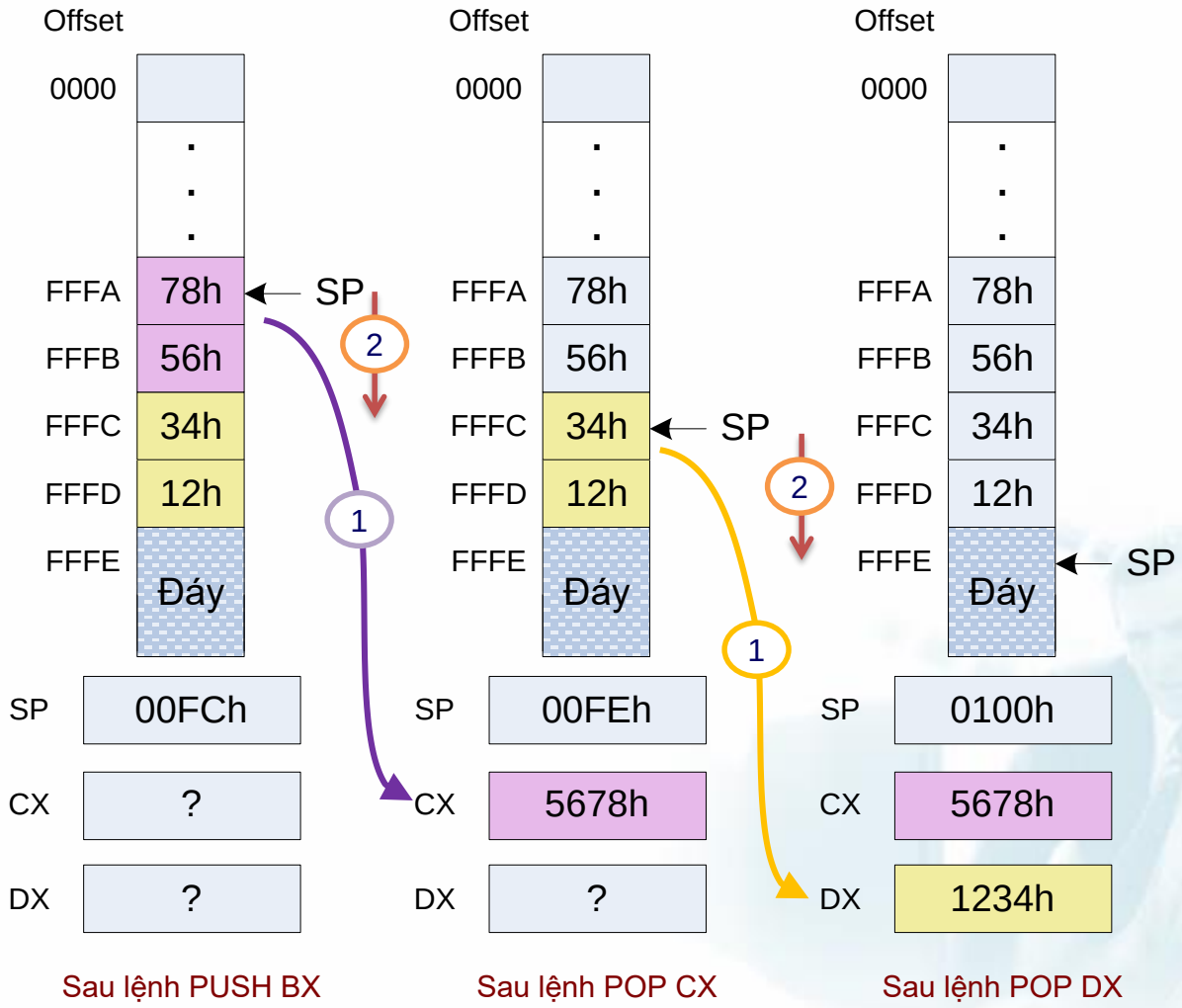
- Lệnh POPF đưa nội dung của từ nhớ ở đỉnh ngăn xếp vào thanh ghi cờ.

Ví dụ về lệnh POP

- Xác định nội dung các Byte nhớ trong Stack.

```
.Stack    100h
...
Start:    ; lệnh đầu tiên của chương trình
  MOV     AX, 1234h
  MOV     BX, 5678h
  PUSH    AX
  PUSH    BX
  POP     CX
  POP     DX
```

Ví dụ (tiếp)


Offset
0000
FFFA
FFFB
FFFC
FFFD
FFFE
78h
56h
34h
12h
Đáy
SP
1
2
SP
00FCh
CX
?
DX
?
Sau lệnh PUSH BX
SP
00FEh
CX
5678h
DX
?
Sau lệnh POP CX
SP
0100h
CX
5678h
DX
1234h
Sau lệnh POP DX

Một số lưu ý

- Các lệnh PUSH, PUSHF, POP không ảnh hưởng đến các cờ.
- Các lệnh trên chỉ thao tác với các WORD.
- Các lệnh sau là không hợp lệ:
  - PUSH AH ; thanh ghi 8 bit
  - POP DL ; thanh ghi 8 bit

Bài tập

- Dữ liệu của 1 chương trình hợp ngữ được khai báo dưới dạng:

```
.Data
mem1        dw      500
mem2        dw      -50
vec1        db      1, 2, 3, 4, 8, 7
vec2        db      10, 20, -10, -20, -30, -40
```

- Hãy xác định nội dung của DX (Hexa) sau khi thực hiện đoạn lệnh sau:

```
push        mem1
push        mem2
mov         bp, sp
mov         dx, [bp]+2
```

a) FFCE b) 0000 c) 01F4 d) FFFF

3.4.2. Thủ tục

- Ngoài thủ tục chính, ta có thể khai báo và sử dụng các thủ tục khác.
- Khai báo thủ tục:

```
Tên_thủ_tục      PROC   Kiểu_thủ_tục
    <Thân thủ tục>
    RET
Tên_thủ_tục      ENDP
```

- Trong đó:
  - Tên_thủ_tục: do người lập trình định nghĩa
  - Kiểu_thủ_tục:
    - ✓NEAR : gọi thủ tục ở trong cùng 1 đoạn
    - ✓FAR : gọi thủ tục ở đoạn khác

Lệnh CALL

- Là lệnh gọi chương trình con (thủ tục)
- Thông dụng: CALL Tên_thủ_tục
- Các bước thực hiện:
  - Thủ tục NEAR
    - ✓SP ← SP – 2
    - ✓Cất nội dung của IP (địa chỉ quay về) vào Stack
    - ✓Nạp địa chỉ của lệnh đầu tiên của chương trình con vào IP
  - Thủ tục FAR
    - ✓SP ← SP – 2
    - ✓Cất nội dung của CS vào Stack
    - ✓SP ← SP – 2
    - ✓Cất nội dung của IP vào Stack
    - ✓Nạp vào CS và IP địa chỉ đầu của chương trình con

Lệnh RET

- Là lệnh trở về từ chương trình con
- Các bước thực hiện:
  - Trở về kiểu NEAR
    - ✓IP ← word nhớ đỉnh Stack
    - ✓SP ← SP + 2
  - Trở về kiểu FAR (RETF)
    - ✓IP ← word nhớ đỉnh Stack
    - ✓SP ← SP + 2
    - ✓CS ← word nhớ tiếp
    - ✓SP ← SP + 2

Truyền dữ liệu giữa các thủ tục

- Các thủ tục của hợp ngữ không có danh sách tham số đi kèm như các ngôn ngữ lập trình bậc cao.
- Người lập trình phải nghĩ ra cách truyền dữ liệu giữa các thủ tục.
- Các cách truyền dữ liệu thông dụng:
  - Truyền qua thanh ghi
  - Sử dụng biến toàn cục
  - Truyền địa chỉ của dữ liệu
  - Sử dụng ngăn xếp (thường dùng trong các NNLT bậc cao)

3.4.3. Các ví dụ

- VD1: Nhập 1 chuỗi kí tự kết thúc bởi ENTER. Hiện chuỗi kí tự viết theo thứ tự ngược lại ở dòng tiếp theo.
- VD2: Cài đặt các thủ tục viết số nhị phân và số Hexa ra màn hình.

Nội dung chương 3

3.5.1. Các lệnh MUL và IMUL

3.5.2. Các lệnh DIV và IDIV

3.5.3. Vào-ra số thập phân

3.5.1. Các lệnh MUL và IMUL

- Có sự khác nhau giữa phép nhân các số không dấu với phép nhân các số có dấu.
- Lệnh nhân cho các số không dấu: MUL nguồn
- Lệnh nhân cho các số có dấu: IMUL nguồn
- Các lệnh trên làm việc với byte (cho KQ là 1 word) hoặc word (cho KQ là 1 double word)
- nguồn (thanh ghi / ngăn nhớ) được coi là số nhân, nếu nguồn là giá trị:
  - 8 bit: AX ⇐ AL x nguồn
    - ✓Số bị nhân là số 8 bit chứa trong AL
    - ✓Tích là số 16 bit chứa trong AX
  - 16 bit: DXAX ⇐ AX x nguồn
    - ✓Số bị nhân là số 16 bit chứa trong AX
    - ✓Tích là số 32 bit chứa trong DXAX

Ảnh hưởng đến các cờ

- SF, ZF, AF, PF : không xác định
- Sau lệnh MUL:
  - CF = OF = 0 nếu nửa cao của kết quả = 0
  - CF = OF = 1 trong các trường hợp còn lại
- Sau lệnh IMUL:
  - CF = OF = 0 nếu nửa cao của kết quả chỉ chứa các giá trị của dấu
  - CF = OF = 1 trong các trường hợp còn lại
- Nói cách khác, CF = OF = 1 nghĩa là kết quả quá lớn để chứa trong nửa thấp (AL hoặc AX) của tích.

3.5.2. Các lệnh DIV và IDIV

- Phép chia không dấu: DIV số_chia
- Phép chia có dấu: IDIV số_chia
- Chia số 16 bit (trong AX) cho số chia 8 bit hoặc chia số 32 bit (trong DXAX) cho số chia 16 bit.
- Thương và số dư có cùng kích thước với số chia.
  - Số chia 8 bit: AL chứa thương, AH chứa số dư
  - Số chia 16 bit: AX chứa thương, DX chứa số dư
- Số dư và số chia trái dấu >< cùng dấu số bị chia???.
- Nếu số chia = 0 hoặc thương nằm ngoài khoảng xác định thì BXL thực hiện INT 0 (lỗi chia cho 0).
- Các cờ không xác định sau phép chia.

Sự mở rộng dấu của số bị chia

- Trong phép chia cho Word, số bị chia được đặt trong DXAX ngay cả khi nó có thể chứa vừa trong AX. Khi đó DX phải được chuẩn bị như sau:
  - Với lệnh DIV, DX phải được xóa về 0.
  - Với lệnh IDIV, DX được lấp đầy bằng bit dấu của AX. Phép biến đổi này được thực hiện bởi lệnh CWD.
- Trong phép chia cho Byte, số bị chia được đặt trong AX ngay cả khi nó có thể chứa vừa trong AL. Khi đó AH phải được chuẩn bị như sau:
  - Với lệnh DIV, AH phải được xóa về 0.
  - Với lệnh IDIV, AH được lấp đầy bằng bit dấu của AL. Phép biến đổi này được thực hiện bởi lệnh CBW.

- Phép chia cho word: chia -1250 cho 7

```
MOV     AX,-1250  ;AX chứa số bị chia
CWD               ;Mở rộng dấu vào DX
MOV     BX,7      ;BX chứa số chia
IDIV    BX        ;AX chứa thương số
                  ;DX chứa số dư
```

- Phép chia cho byte: chia 122 cho -7

```
MOV     AL,122      ;AL chứa số bị chia
CBW                 ;Mở rộng dấu vào AH
MOV     BL,-7       ;BL chứa số chia
IDIV    BL          ;AL chứa thương,
                    ;AH chứa số dư
```

3.5.3 Vào-ra số thập phân

- Các thao tác:
  - In số thập phân
  - Nhập số thập phân

In số thập phân

- In số nguyên có dấu trong AX ra màn hình dưới dạng số thập phân.
- Thuật giải:

IF AX < 0 THEN

    In ra dấu '–'

    AX := số bù 2 của AX

END IF

Lấy dạng thập phân của từng chữ số trong AX

Đổi các chữ số này ra kí tự rồi in ra màn hình

In số thập phân (tiếp)

- Lấy dạng thập phân của từng chữ số trong AX:

```
Đếm := 0
REPEAT
    Chia số bị chia cho 10          ; số bị chia ban đầu = AX
    Cất số dư vào trong Stack
    Đếm := Đếm + 1
UNTIL Thương = 0
```

- Đổi các chữ số ra kí tự rồi in ra màn hình:

```
FOR Đếm lần DO
    Lấy từng chữ số từ Stack
    Đổi ra kí tự
    In kí tự đó ra màn hình
END FOR
```

Nhập số thập phân

- Thuật giải (đơn giản):

  Tổng := 0

  Đọc 1 kí tự ASCII

  REPEAT

    Đổi kí tự ra giá trị thập phân

    Tổng := Tổng * 10 + giá trị nhận được

    Đọc kí tự

  UNTIL kí tự vừa nhận = Enter

Nội dung chương 3

3.6. Mảng và các chế độ địa chỉ

3.6.1. Mảng một chiều

3.6.2. Các chế độ địa chỉ

3.6.3. Sắp xếp một mảng

3.6.4. Mảng hai chiều

3.6.1. Mảng một chiều

- Mảng một chiều là một dãy có thứ tự các phần tử có cùng một kiểu
- Khai báo mảng
  - MSG DB “abcdef”
  - W DW 10,20,30,40,50

Chỉ số	
1	A[1]
2	A[2]
3	A[3]
4	A[4]
5	A[5]
6	A[6]

Mảng một chiều (tiếp)

- Địa chỉ của biến mảng gọi là địa chỉ cơ sở của mảng (base address)
- Địa chỉ offset của mảng W=0200h -> phân bố mảng trong bộ nhớ

Địa chỉ offset	Ký hiệu địa chỉ	Giá trị thập phân
0200h	W	10
0202h	W+2h	20
0204h	W+4h	30
0206h	W+6h	40
0208h	W+8h	50

Mảng một chiều (tiếp)

- Xác định vị trí phần tử trong mảng
  - Cộng một hằng số vào địa chỉ cơ sở
  - Giả sử: mảng A là mảng các phần tử có kích thước S byte, vị trí các phần tử của mảng A

Số thứ tự	Vị trí
1	A
2	A+1*S
3	A+2*S
.	.
.	.
.	.
N	A+(N-1)*S

Mảng một chiều (tiếp)

- Hoán chuyển các phần tử của mảng
  - Hoán chuyển phần tử thứ 2 và thứ 4 của mảng W (mảng W)
    - ✓W[2]: địa chỉ W+2
    - ✓W[4]: địa chỉ W+6

```
MOV     AX,W+2      ;AX chứa W[2]
XCHG    W+6,AX      ;AX chứa W[4], W[4]=W[2]
MOV     W+2,AX      ;W[2]=W[4]
```

- Chế độ địa chỉ gián tiếp thanh ghi
  - Địa chỉ offset của toán hạng được chứa trong thanh ghi -> thanh ghi đóng vai trò con trỏ trỏ đến ô nhớ
  - Khuôn dạng toán hạng

    [register]
  - Các thanh ghi có thể: BX,SI,DI,BP
    - ✓BX,SI,DI -> DS là thanh ghi đoạn
    - ✓BP -> SS là thanh ghi đoạn

Chế độ địa chỉ gián tiếp thanh ghi

- **Giả sử:**
  - BX chứa 1000h, offset 1000h chứa 1BACh
  - SI chứa 2000h, offset 2000h chứa 20FEh
  - DI chứa 3000h, offset 3000h chứa 031Dh
- **Kiểm tra tính hợp lệ và kết quả của các lệnh sau**

a. MOV BX,[BX]
b. MOV CX,[SI]
c. MOV BX,[AX]
d. ADD [SI],[DI]
e. INC [DI]

Chế độ địa chỉ cơ sở và địa chỉ chỉ số

- Địa chỉ offset=độ dịch + nội dung một thanh ghi
  - Độ dịch có thể là:
    - ✓Địa chỉ offset của một biến
    - ✓Một hằng số (âm hoặc dương)
    - ✓Địa chỉ offset của một biến cộng hoặc trừ với một hằng số
  - Toán hạng

    [thanh ghi + độ dịch]

    [độ dịch + thanh ghi]

    [thanh ghi]+độ dịch

    Độ dịch + [thanh ghi]

    Độ dịch[Thanh ghi]

Chế độ địa chỉ cơ sở và địa chỉ chỉ số

- Giả sử mảng A được khai báo

.DATA

A DW 0123h, 0456h, 0789h, 0ABCh

BX=2

SI=4

DI=1

- Kiểm tra tính hợp lệ của các lệnh sau đây và kết quả của các lệnh
  a. MOV AX,[A+BX]
  b. MOV BX,[BX+2]
  c. MOV CX,A[SI]
  d. MOV AX,-2[SI]
  e. MOV BX,[A+3+DI]

Truy nhập ngăn xếp

- Sử dụng thanh ghi cơ sở BP -> truy nhập đoạn ngăn xếp

SS:BP

- Ví dụ:

```
MOV     BP,SP    ;SP trỏ đến đỉnh ngăn xếp
MOV     AX,[BP] ;chuyển đỉnh ngăn xếp vào AX
MOV     AX,[BP+2] ;chuyển từ thứ hai vào AX
```

Thuật toán: mảng A có N phần tử

```
For i=1 to N-1
{
    For j=i+1 to N
    {
        if A[j] < A[i]
        {
            swap (A[i], A[j]);
        }
    }
}
```

- Yêu cầu: viết chương trình
- Nhập một chuỗi ký tự, kết thúc khi gõ dấu " "
- Sắp xếp chuỗi theo thứ tự tăng dần của mã ASCII.

3.6.4. Mảng hai chiều

- **Mảng hai chiều là mảng của các mảng**: một mảng một chiều mà mỗi phần tử của nó lại là một mảng một chiều
- Các phần tử được xếp vào các hàng và cột
- Mảng hai chiều có thể được lưu trữ theo
  - **Thứ tự hàng:** các phần tử hàng 1 được lưu trữ, tiếp đến là hàng 2…
  - **Thứ tự cột:** các phần tử cột 1 được lưu trữ, tiếp đến là cột 2…

Mảng hai chiều (tiếp)

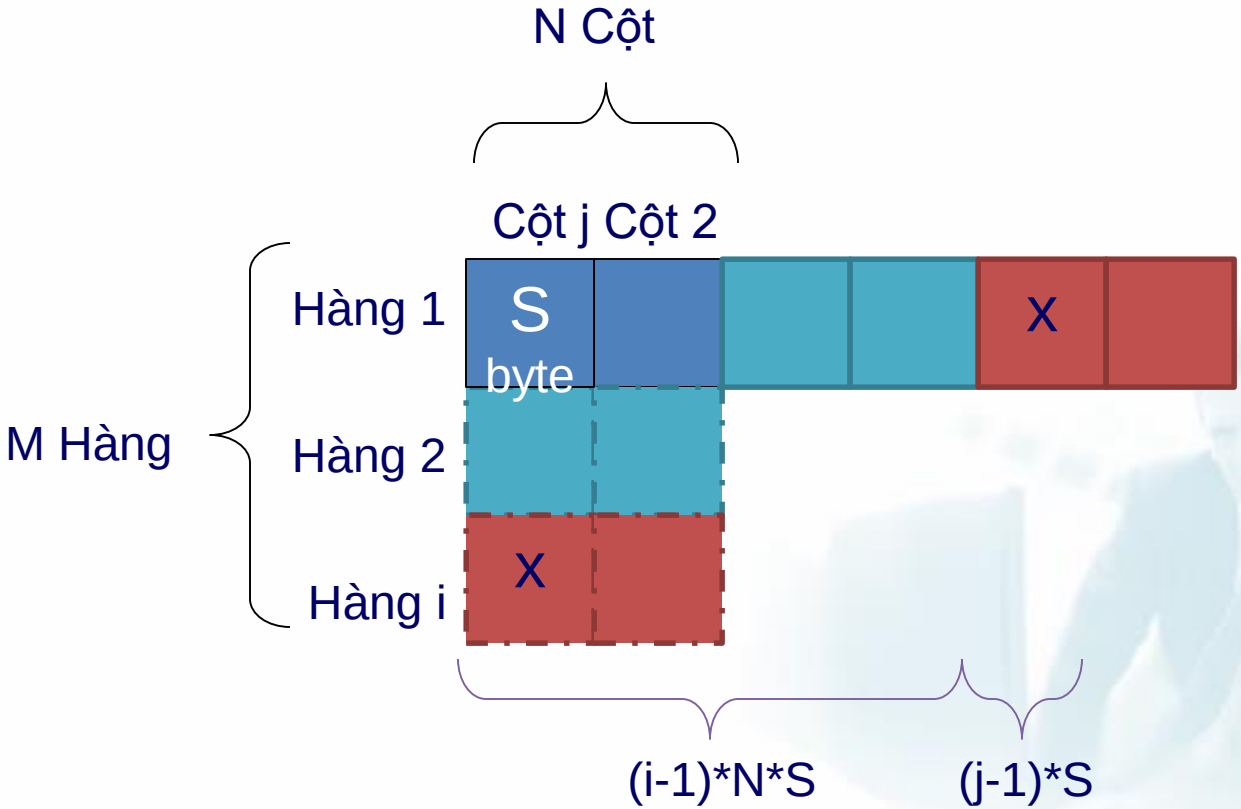
- Xác định vị trí một phần tử trong mảng hai chiều
  - Giả sử: mảng A kích thước M*N lưu theo thứ tự hàng, kích thước mỗi phần tử là S, để tìm A[i,j] ta xác định:
    - ✓Vị trí bắt đầu hàng i: A + (i-1)*N*S
    - ✓Vị trí của phần tử thứ j trong hàng: (j-1)*S

    Thứ tự cột

$$A[i,j]=A+((i-1)+(j-1)*M)*S$$

Thứ tự hàng

$$A[i,j]=A+((i-1)*N + (j-1))*S$$



Chế độ địa chỉ chỉ số cơ sở

- **Địa chỉ Offset của toán hạng là tổng của:**
  - Nội dung của thanh ghi cơ sở (BX hay BP)
  - Nội dung của một thanh ghi chỉ số (SI hay DI)
  - (có thể) Địa chỉ offset của một biến byte
  - (có thể) Một hằng số (âm hay dương)

  VD:

  MOV AX,[BX+SI+W]

- Lập trình chương trình thực hiện:
  - Nhập vào các phần tử cho một mảng số nguyên kích thước 3x3
  - Tính tổng các phần tử thuộc đường chéo chính

3.7.1. Cờ định hướng
3.7.2. Chuyển một chuỗi
3.7.3. Lưu kí tự vào chuỗi
3.7.4. Nạp kí tự của chuỗi
3.7.5. Tìm kí tự trong chuỗi
3.7.6. So sánh chuỗi
3.7.7. Tổng kết thao tác chuỗi

3.7.1. Cờ định hướng

- Cờ định hướng DF (Direction Flag) xác định hướng cho các thao tác chuỗi.
- Các thao tác chuỗi được thực hiện thông qua 2 thanh ghi chỉ số SI và DI.
- Nếu DF = 0 thì SI và DI được xử lý theo chiều tăng của địa chỉ bộ nhớ (từ trái qua phải trong chuỗi).
- Nếu DF = 1 thì SI và DI được xử lý theo chiều giảm của địa chỉ bộ nhớ (từ phải qua trái trong chuỗi).

Các lệnh CLD và STD

- Lệnh CLD (Clear Direction Flag): xóa cờ hướng

```
CLD          ; xóa DF = 0
```

- Lệnh STD (Set Direction Flag): thiết lập cờ hướng

```
STD          ; thiết lập DF = 1
```

- Các lệnh này không ảnh hưởng đến các cờ khác.

3.7.2. Chuyển một chuỗi

- Bài toán: giả sử có 2 chuỗi được định nghĩa như sau:

```
.DATA
STRING1         DB    'BACH KHOA'
STRING2         DB    9 DUP (?)
```

- Cần chuyển nội dung của chuỗi STRING1 (chuỗi nguồn) sang chuỗi STRING2 (chuỗi đích).

Các lệnh liên quan

- Lệnh: MOVSB (Move String Byte)
  - Chuyển 1 phần tử 1 byte của chuỗi gốc (trỏ bởi DS:SI) sang 1 phần tử của chuỗi đích (trỏ bởi ES:DI).
  - Sau khi thực hiện:
    - ✓SI và DI tăng thêm 1 nếu cờ hướng DF = 0 (dùng lệnh CLD)
    - ✓SI và DI giảm đi 1 nếu cờ hướng DF = 1 (dùng lệnh STD)
- Lệnh: MOVSW (Move String Word)
  - Chuyển 1 phần tử 1 word (2 byte) của chuỗi gốc (trỏ bởi DS:SI) sang 1 phần tử của chuỗi đích (trỏ bởi ES:DI).
  - Sau khi thực hiện:
    - ✓SI và DI tăng thêm 2 nếu cờ hướng DF = 0 (dùng lệnh CLD)
    - ✓SI và DI giảm đi 2 nếu cờ hướng DF = 1 (dùng lệnh STD)

Các lệnh liên quan (tiếp)

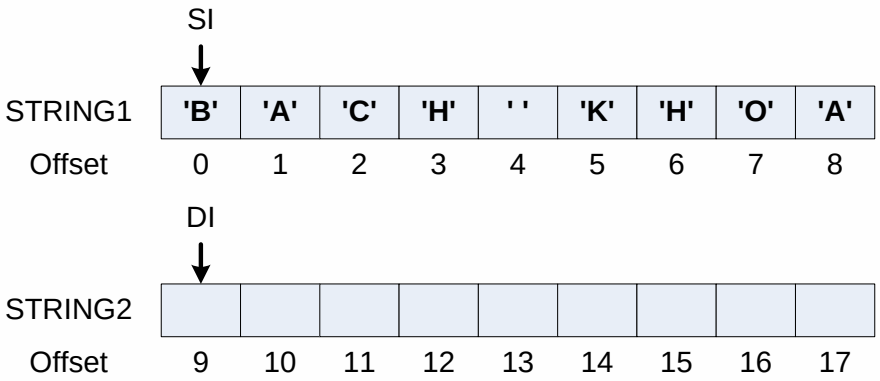
- Để chuyển nhiều kí tự ta cần sử dụng các lệnh lặp.
- Lệnh: REP <lệnh thao tác kí tự>
  - Lặp lại lệnh viết sau đó cho đến khi CX = 0
  - Mỗi lần lặp CX giảm đi 1 ⇒ số lần lặp phải gán trước vào CX. Ví dụ:

```
MOV   CX, 5
REP   MOVSB    ; chuyển 5 byte từ chuỗi nguồn đến chuỗi đích
```

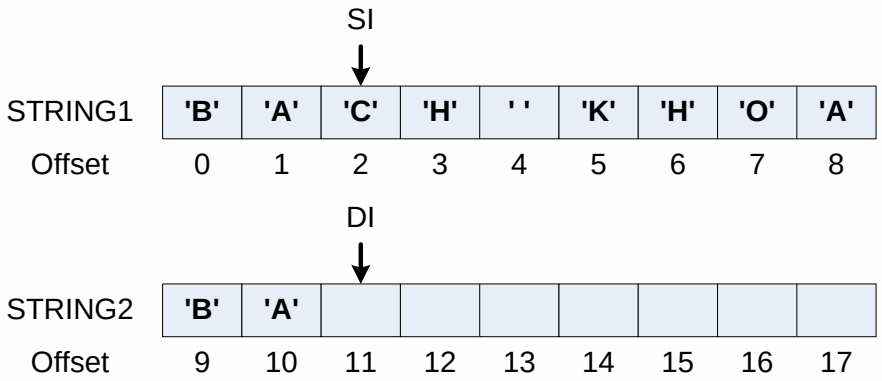
- Lệnh: REPE/REPZ <lệnh thao tác kí tự>
  - Lặp lại lệnh viết sau đó cho đến khi CX = 0 hoặc ZF = 0
- Lệnh: REPNE/REPNZ <lệnh thao tác kí tự>
  - Lặp lại lệnh viết sau đó cho đến khi CX = 0 hoặc ZF = 1

Ví dụ

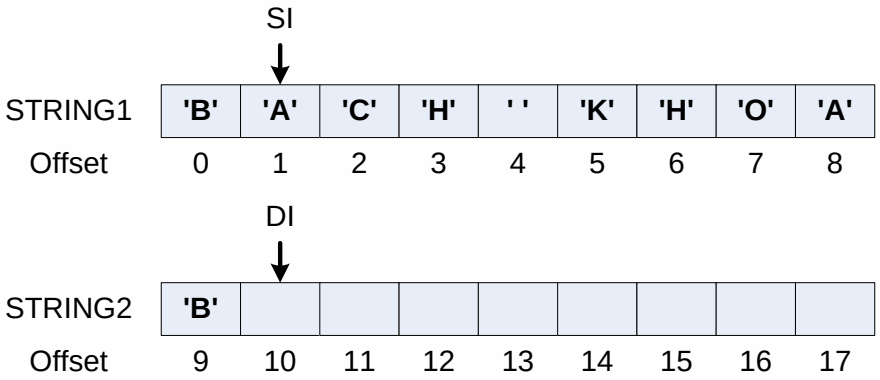
```
MOV   AX, @DATA
MOV   DS, AX          ; khởi tạo DS
MOV   ES, AX          ; và ES đều trỏ đến đoạn dữ liệu DATA
LEA   SI, STRING1     ; SI trỏ đến chuỗi nguồn
LEA   DI, STRING2     ; DI trỏ đến chuỗi đích
CLD                   ; Xóa cờ hướng
MOV   CX, 9           ; Số byte cần chuyển
REP   MOVSB           ; Chuyển 9 byte từ STRING1 sang STRING2
```



Trước khi thực hiện các lệnh MOVSB



Sau khi thực hiện lệnh MOVSB thứ 2



Sau khi thực hiện lệnh MOVSB thứ 1

Sau khi thực hiện lệnh MOVSB thứ 9

3.7.3. Lưu kí tự vào chuỗi

- Lệnh: STOSB (Store String Byte from AL)
  - Chuyển nội dung thanh ghi AL sang 1 phần tử (1 byte) được trỏ bởi ES:DI của chuỗi đích.
  - Sau khi thực hiện:
    - ✓DI tăng thêm 1 nếu cờ hướng DF = 0 (dùng lệnh CLD)
    - ✓DI giảm đi 1 nếu cờ hướng DF = 1 (dùng lệnh STD)
- Lệnh: STOSW (Store String Word from AX)
  - Chuyển nội dung thanh ghi AX sang 1 phần tử (2 byte) được trỏ bởi ES:DI của chuỗi đích.
  - Sau khi thực hiện:
    - ✓DI tăng thêm 2 nếu cờ hướng DF = 0 (dùng lệnh CLD)
    - ✓DI giảm đi 2 nếu cờ hướng DF = 1 (dùng lệnh STD)

- Lưu 5 kí tự 'A' vào đầu chuỗi STRING2

```
MOV     AX, @DATA
MOV     ES, AX          ; ES trỏ đến đoạn dữ liệu DATA
LEA     DI, STRING2     ; ES:DI trỏ đến đầu chuỗi STRING2
CLD                     ; Xóa cờ hướng
MOV     CX, 5           ; Số kí tự cần lưu
MOV     AL, 'A'         ; Kí tự cần lưu vào chuỗi
REP     STOSB           ; Lặp lưu 5 lần kí tự 'A' vào STRING2
```

Ví dụ 2

- Nhập các kí tự từ bàn phím rồi lưu vào chuỗi STRING cho đến khi nhập đủ 20 kí tự hoặc gặp phím ENTER.

```
    MOV     AX, @DATA
    MOV     ES, AX          ; ES trỏ đến đoạn dữ liệu DATA
    LEA     DI, STRING      ; ES:DI trỏ đến đầu chuỗi STRING
    CLD                     ; Xóa cờ hướng
    MOV     CX, 20          ; Số kí tự tối đa được nhập từ bàn phím
    XOR     BX, BX          ; Khởi tạo số kí tự được nhập ban đầu = 0
    MOV     AH, 1           ; Hàm nhập kí tự từ bàn phím
DocKiTu:
    INT     21h             ; Nhập 1 kí tự ⇒ AL chứa mã ASCII của kí tự
    CMP     AL, 13          ; Là phím ENTER ?
    JZ      DungNhap        ; ⇒ Dừng nhập
    STOSB                   ; Nếu không phải thì lưu AL vào chuỗi string
    INC     BX              ; Tăng số đếm số kí tự được nhập
    LOOP    DocKiTu         ; Lặp lại (tối đa 20 lần)
DungNhap:
```

3.7.4. Nạp kí tự của chuỗi

- Lệnh: LODSB (Load String Byte into AL)
  - Chuyển 1 phần tử (1 byte) được trỏ bởi DS:SI của chuỗi nguồn vào thanh ghi AL.
  - Sau khi thực hiện:
    - ✓SI tăng thêm 1 nếu cờ hướng DF = 0 (dùng lệnh CLD)
    - ✓SI giảm đi 1 nếu cờ hướng DF = 1 (dùng lệnh STD)
- Lệnh: LODSW (Load String Word into AX)
  - Chuyển 1 phần tử (2 byte) được trỏ bởi DS:SI của chuỗi nguồn vào thanh ghi AX.
  - Sau khi thực hiện:
    - ✓SI tăng thêm 2 nếu cờ hướng DF = 0 (dùng lệnh CLD)
    - ✓SI giảm đi 2 nếu cờ hướng DF = 1 (dùng lệnh STD)

Ví dụ

- Giả sử STR1 và STR2 là các chuỗi có độ dài là 40 kí tự. Viết đoạn chương trình chuyển các kí tự chữ hoa từ STR1 sang STR2.

```
    MOV     AX, @DATA
    MOV     DS, AX          ; khởi tạo DS
    MOV     ES, AX          ; và ES đều trỏ đến đoạn dữ liệu DATA
    LEA     SI, STR1        ; SI trỏ đến chuỗi nguồn STR1
    LEA     DI, STR2        ; DI trỏ đến chuỗi đích STR2
    CLD                     ; Xóa cờ hướng
    MOV     CX, 40          ; Số kí tự của STR1 cần xét (độ dài xâu STR1)
    XOR     BX, BX  ; Khởi tạo số kí tự thực sự được chuyển ban đầu = 0
VongLap:
    LODSB                   ; Nạp 1 kí tự của STR1 vào AL
    CMP     AL, 'A'         ; Nếu AL < 'A' ⇒ không phải là chữ hoa
    JB      LapTiep         ; thì xét kí tự tiếp theo
    CMP     AL, 'Z'         ; Nếu AL > 'Z' ⇒ không phải là chữ hoa
    JA      LapTiep         ; thì xét kí tự tiếp theo
    STOSB                   ; Nếu AL là chữ cái hoa thì cất vào chuỗi STR2
    INC     BX              ; Tăng số đếm số chữ cái hoa
LapTiep:
    LOOP    VongLap         ; Lặp lại, xét kí tự tiếp theo của STR1
```

3.7.5. Tìm kí tự trong chuỗi

- Lệnh: SCASB (Scan String Byte)
  - Trừ thử nội dung của AL cho 1 byte đích đang được trỏ bởi ES:DI, không thay đổi giá trị AL mà chỉ cập nhật cờ.
  - Sau khi thực hiện:
    - ✓DI tăng thêm 1 nếu cờ hướng DF = 0 (dùng lệnh CLD)
    - ✓DI giảm đi 1 nếu cờ hướng DF = 1 (dùng lệnh STD)
- Lệnh: SCASW (Scan String Word)
  - Trừ thử nội dung của AX cho 1 word đích đang được trỏ bởi ES:DI, không thay đổi giá trị AX mà chỉ cập nhật cờ.
  - Sau khi thực hiện:
    - ✓DI tăng thêm 2 nếu cờ hướng DF = 0 (dùng lệnh CLD)
    - ✓DI giảm đi 2 nếu cờ hướng DF = 1 (dùng lệnh STD)

Ví dụ 1

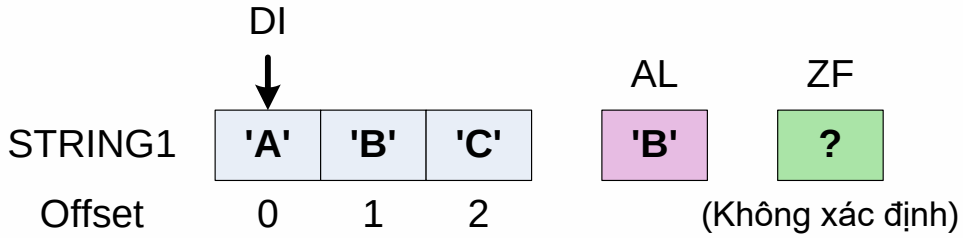
- Cho 1 chuỗi được khai báo như sau:

```
.DATA
STRING1        DB     'ABC'
```

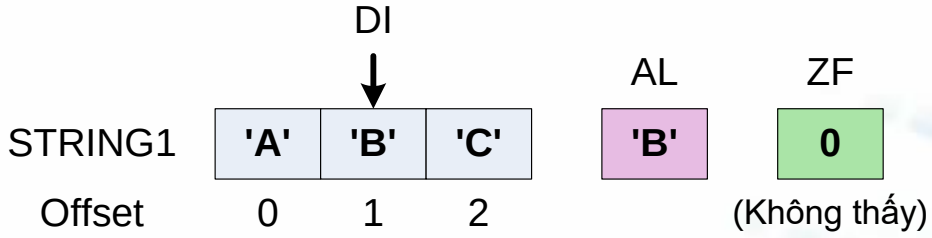
- Khảo sát đoạn chương trình sau:

```
MOV       AX, @DATA
MOV       ES, AX
CLD
LEA       DI, STRING1
MOV       AL, 'B'
SCASB
SCASB
```

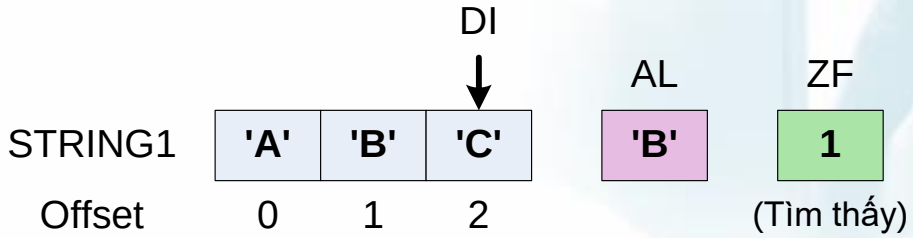
Trước khi thực hiện lệnh SCASB



Sau khi thực hiện lệnh SCASB thứ 1



Sau khi thực hiện lệnh SCASB thứ 2



Ví dụ 2

- Tìm chữ cái 'A' đầu tiên trong chuỗi STRING2 có độ dài 40 kí tự.
- Đoạn chương trình:

```
MOV     AX, @DATA
MOV     ES, AX          ; ES trỏ đến đoạn dữ liệu
CLD                     ; Xóa cờ hướng
LEA     DI, STRING2     ; ES:DI trỏ đến chuỗi đích STRING2
MOV     CX, 40          ; Độ dài chuỗi STRING2
MOV     AL, 'A'         ; AL chứa kí tự cần tìm
REPNE   SCASB           ; Tìm cho đến khi thấy hoặc CX=0
```

- Ra khỏi đoạn chương trình:
  - Nếu ZF = 1 thì ES:[DI-1] là kí tự 'A' đầu tiên tìm thấy
  - Nếu ZF = 0 thì trong chuỗi STRING2 không chứa kí tự 'A'

3.7.6. So sánh chuỗi

- Lệnh: CMPSB (Compare String Byte)
  - Trừ thử 1 byte ở địa chỉ DS:SI cho 1 byte ở địa chỉ ES:DI, kết quả không được lưu lại mà chỉ cập nhật cờ.
  - Sau khi thực hiện:
    - ✓SI, DI tăng thêm 1 nếu cờ hướng DF = 0 (dùng lệnh CLD)
    - ✓SI, DI giảm đi 1 nếu cờ hướng DF = 1 (dùng lệnh STD)
- Lệnh: CMPSW (Compare String Word)
  - Trừ thử 1 word ở địa chỉ DS:SI cho 1 word ở địa chỉ ES:DI, kết quả không được lưu lại mà chỉ cập nhật cờ.
  - Sau khi thực hiện:
    - ✓SI, DI tăng thêm 2 nếu cờ hướng DF = 0 (dùng lệnh CLD)
    - ✓SI, DI giảm đi 2 nếu cờ hướng DF = 1 (dùng lệnh STD)

Ví dụ

- Cho 2 chuỗi được khai báo như sau:

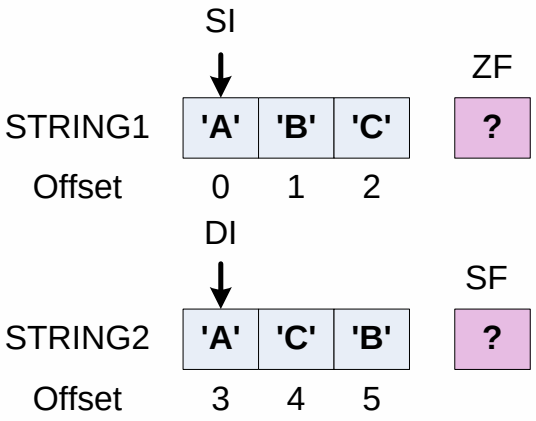
```
.DATA
STRING1   DB      'ABC'
STRING2   DB      'ACB'
```

- Khảo sát đoạn chương trình sau:

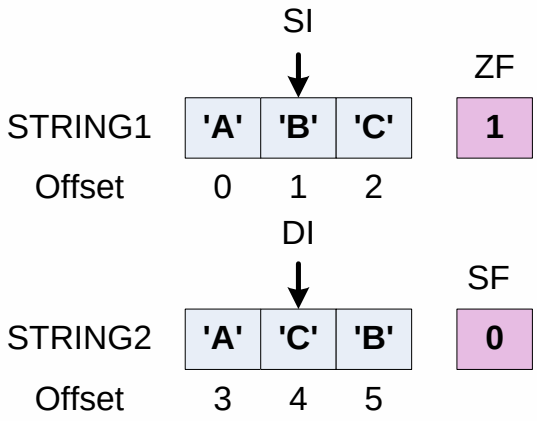
```
MOV       AX, @DATA
MOV       DS, AX
MOV       ES, AX
CLD
LEA       SI, STRING1
LEA       DI, STRING2
CMPSB
CMPSB
CMPSB
```

Ví dụ (tiếp)

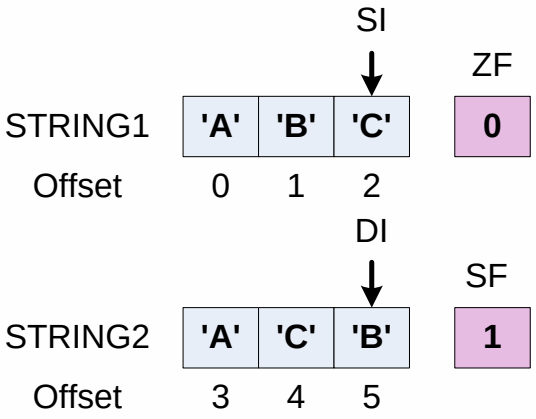
Trước lệnh CMPSB thứ 1



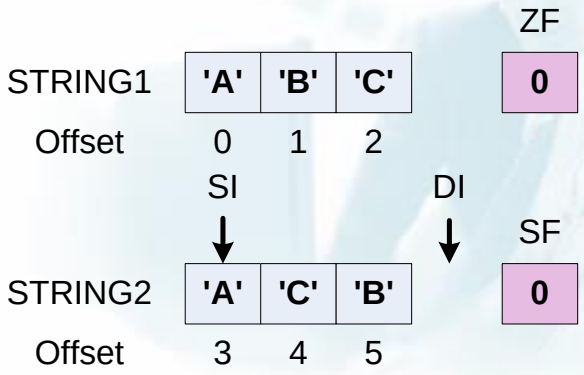
Sau lệnh CMPSB thứ 1



Sau lệnh CMPSB thứ 2



Sau lệnh CMPSB thứ 3



Ví dụ

- Cho 2 chuỗi được khai báo như sau:

```
.DATA
STRING1   DB      ‘BACH KHOA HA NOI'
STRING2   DB      ‘HA NOI'
```

- Viết 1 đoạn chương trình để tìm kiếm STRING2 trong STRING1. Nếu có, trả về vị trí mà STRING2 xuất hiện trong STRING1. Nếu không, trả về -1

```
;for i=0->N-M; N la do dai string1, M la do dai string2
;  for j=0->M-1
;    if string1[j]!=string2[i+j]
;        break
;  if j==M thi j la chi so can tim
;        break
;
;
```

3.7.7. Tổng kết thao tác chuỗi

Lệnh	Toán hạng nguồn	Toán hạng đích	Dạng byte	Dạng word
Chuyển chuỗi	DS : SI	ES : DI	MOVSB	MOVSW
Lưu kí tự vào chuỗi	AL hay AX	ES : DI	STOSB	STOSW
Nạp kí tự của chuỗi	DS : SI	AL hay AX	LODSB	LODSW
Tìm kí tự trong chuỗi	AL hay AX	ES : DI	SCASB	SCASW
So sánh chuỗi *(Không lưu KQ)*	DS : SI	ES : DI	CMPSB	CMPSW

Ví dụ

VD1: Cho phép người dùng nhập một chuỗi từ bán phím (lưu vào str1), chuỗi có chiều dài tối đa 20 ký tự. Quá trình nhập kết thúc khi người dùng bấm phím Enter hoặc đủ 20 ký tự. Lưu chuỗi str1 theo chiều ngược lại vào chuỗi str2 và hiển thị chuỗi str2

VD2: Viết chương trình con đếm số ký tự hoa trong một chuỗi (có kết thúc bằng '$'). Đầu vào là thanh ghi BX chứa địa chỉ cơ sở của chuỗi, đầu ra là thanh ghi CX chứa số ký tự hoa

Phần 1: Các lệnh điều khiển và rẽ nhánh

Thuật toán sau có thể thực hiện phép nhân hai số dương M và N bằng cách lặp lại các phép cộng

Khởi động tích số bằng 0

REPEAT

    cộng M vào tích số

    giảm N

UNTIL N=0

Phần 2: Các lệnh logic, dịch, quay

Viết các lệnh logic để thực hiện công việc sau đây

1. Xóa các bit ở vị trí chẵn của AX, giữ nguyên các bit khác
2. Thiết lập bit LSB và MSB của BL trong khi giữ nguyên các bit khác
3. Đảo bit MSB của BL, giữ nguyên các bit khác
4. Thay nội dung của biến word1 bằng số bù 1 của nó

- Phần 4: Mảng và các chế độ địa chỉ

  Tính tổng đường chéo chính của một ma trận vuông

- Phần 5: Thao tác chuỗi

  Viết chương trình con đếm số chữ số trong một chuỗi đã lưu, đầu vào là địa chỉ cơ sở của chuỗi (BX) và số ký tự trong chuỗi (CX)

CHƯƠNG 4

Bộ vi xử lý 8088/8086 và nối ghép với bộ nhớ

Nội dung chương 4

4.1. Bộ vi xử lý 8088

4.2. Các mạch phụ trợ cho 8088

4.3. Các chu kỳ bus

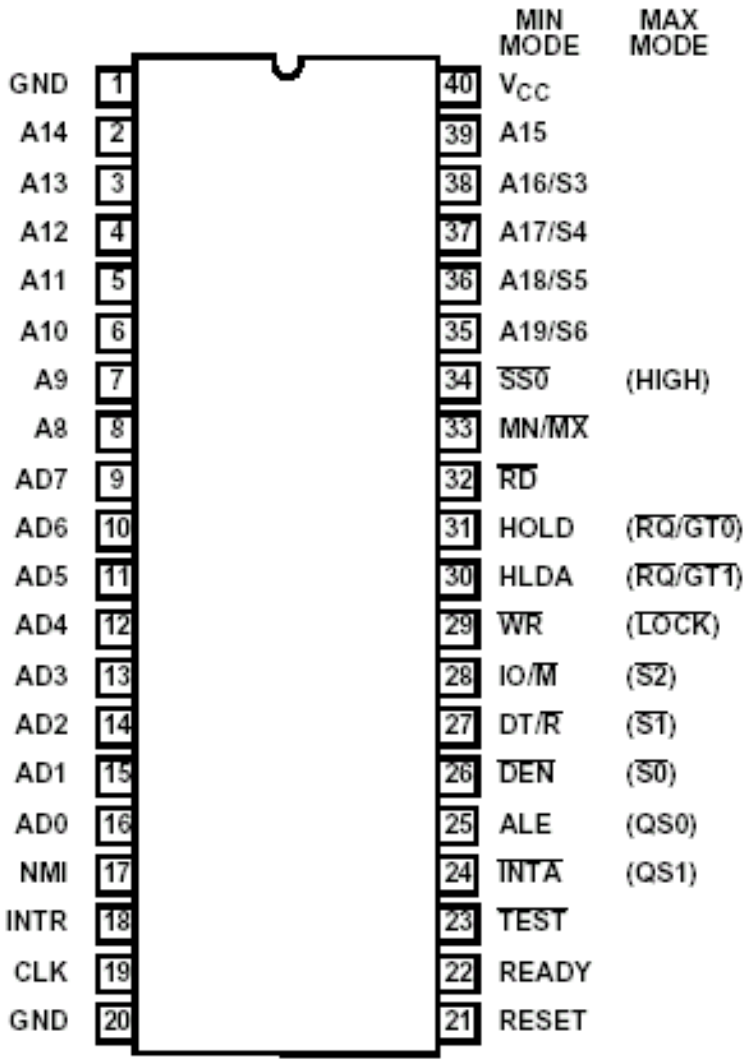
4.4. Bộ nhớ bán dẫn

4.5. Giải mã địa chỉ cho bộ nhớ

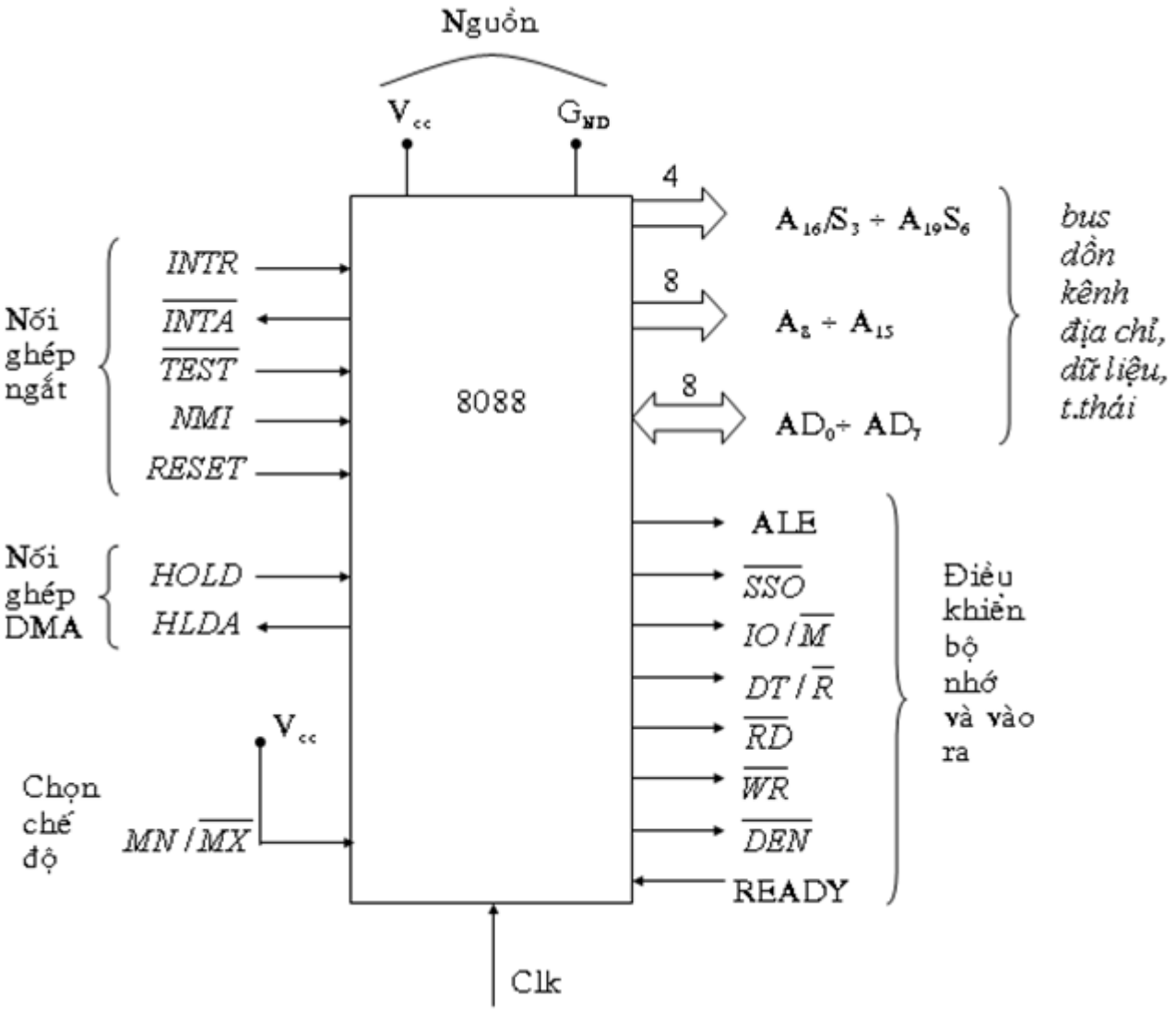
Vi xử lý 8088

- 40 chân
- Hoạt động ở mức điện áp TTL
  - ✓Logic 0: 0-0.8V
  - ✓Logic 1: 2-5V

- Có hai chế độ làm việc
  - Chế độ MIN: cho các hệ thống đơn giản, chỉ sử dụng một vi xử lý
  - Chế độ MAX: cho các hệ thống sử dụng nhiều vi xử lý



Sơ đồ chân của 8088



Chế độ MIN

- Bus dồn kênh dữ liệu/ địa chỉ/ trạng thái
  - ADO->AD7: địa chỉ/ dữ liệu
    - Trong đó
      - A0->A7: 8 bit thấp của địa chỉ (1 chiều ra)
      - D0->D7: bus dữ liệu (2 chiều)
    - Chân ALE là chân chốt
      - ALE=1: chốt địa chỉ A0->A7
      - ALE=0: chốt dữ liệu D0->D7
  - A8->A15: 8 bit địa chỉ tiếp theo

Chế độ MIN (tiếp)

- A16/S3 -> A19/S6: 4 chân địa chỉ, trạng thái dồn kênh
- Chân ALE là chân chốt
  - ALE=1: chốt địa chỉ A16->A19
  - ALE=0: chốt trạng thái S3->S6

✓S6 = 0
✓S5 giá trị của cờ IF
✓S3-S4: xác định đoạn nhớ đang được truy nhập

S4	S3	Đoạn ngăn nhớ
0	0	Đoạn dữ liệu phụ (ES)
0	1	Đoạn ngăn xếp (SS)
1	0	Đoạn lệnh hoặc không có (CS)
1	1	Đoạn dữ liệu (DS)

Chế độ MIN (tiếp)

- Các chân điều khiển bộ nhớ và vào ra
- ALE (Address Latch Enable): tín hiệu chốt địa chỉ, khi ALE=1 -> địa chỉ được đưa lên bus
- IO/$\overline{M}$ (Input ouput/ Memory): tín hiệu điều khiển truy nhập bộ nhớ hay cổng vào ra
  - IO/$\overline{M}$=0 -> truy nhập bộ nhớ
  - IO/$\overline{M}$=1 -> truy nhập cổng vào ra
- DT/$\overline{R}$ (Data Transmit/ Receive): tín hiệu điều khiển hướng truyền dữ liệu
  - DT/$\overline{R}$=0 -> bộ vi xử lý nhận dữ liệu vào
  - DT/$\overline{R}$=1 -> bộ vi xử lý gửi tín hiệu ra

Chế độ MIN (tiếp)

- $\overline{RD}$ (Read): tín hiệu điều khiển đọc bộ nhớ hay cổng vào ra, tích cực mức 0
- $\overline{WR}$ (Write): tín hiệu điều khiển ghi dữ liệu ra bộ nhớ hay cổng vào ra, tích cực mức 0
- $\overline{DEN}$ (Data enable): $\overline{DEN}$=0 thì trên bus có dữ liệu (hoạt động trong chu kỳ đọc dữ liệu)
- READY: tín hiệu vào, báo bộ nhớ hay cổng vào ra sẵn sàng làm việc với bộ vi xử lý
- $\overline{SSO}$ (Status line): kết hợp với tín hiệu DT/$\overline{R}$ và IO/$\overline{M}$ cho biết trạng thái chu kỳ bus

Chế độ MIN (tiếp)

$IO/\overline{M}$	$DT/\overline{R}$	$\overline{SSO}$	Chức năng chu kì bus
0	0	0	Truy nhập lệnh (nhận lệnh)
0	0	1	Đọc bộ nhớ
0	1	0	Ghi bộ nhớ
0	1	1	Thụ động
1	0	0	Chấp nhận ngắt
1	0	1	Đọc cổng IO
1	1	0	Ghi cổng IO
1	1	1	Dừng

Các chân tín hiệu nối ghép ngắt

- INTR (Interrupt Request): tín hiệu vào yêu cầu ngắt
- $\overline{\text{INTA}}$ (Interrupt Acknowledge): tín hiệu ra cho phép ngắt
- MNI (Non maskable Interrupt): tín hiệu vào, ngắt không che được, báo sự cố hệ thống
- RESET: tín hiệu vào để khởi động lại bộ vi xử lý
- $\overline{\text{TEST}}$: khi thực hiện lệnh Wait, bộ vi xử lý sẽ kiểm tra tín hiệu này

Các chân tín hiệu điều khiển nối ghép DMA

- HOLD: tín hiệu vào từ mạch điều khiển bên ngoài báo cho bộ xử lý để yêu cầu chuyển quyền điều khiển bus
- HLDA (Hold Acknowledge): tín hiệu ra báo cho mạch điều khiển bên ngoài biết bộ xử lý chấp nhận chuyển quyền điều khiển bus

Nội dung chương 4

4.1. Bộ vi xử lý 8088

4.2. Các mạch phụ trợ cho 8088

4.3. Các chu kỳ bus

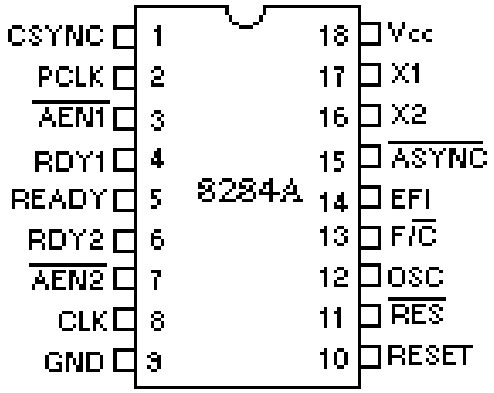
4.4. Bộ nhớ bán dẫn

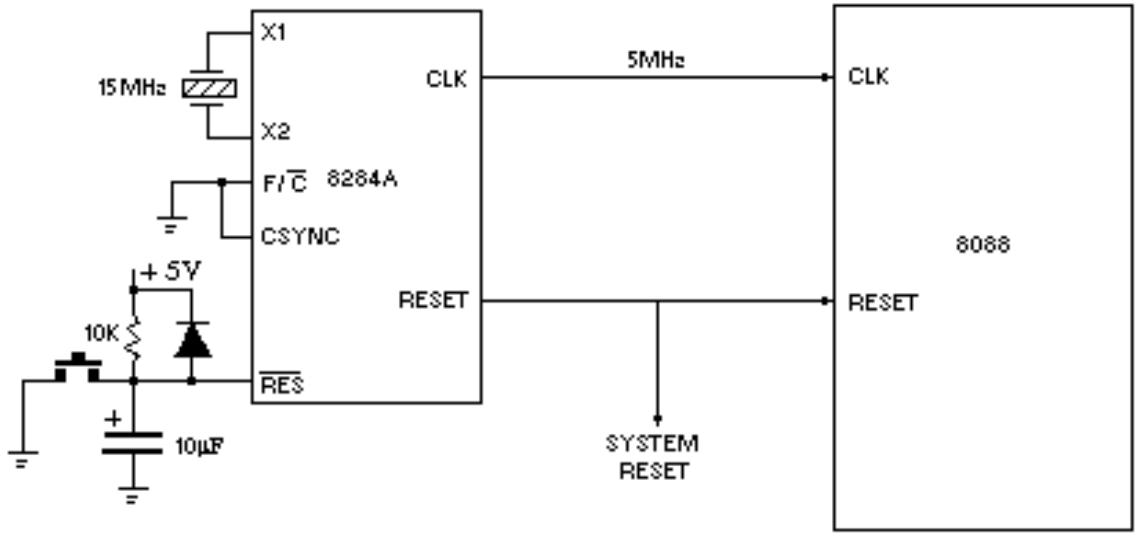
4.5. Giải mã địa chỉ cho bộ nhớ

4.2. Các mạch phụ trợ cho 8088

Mạch tạo xung nhịp 8284

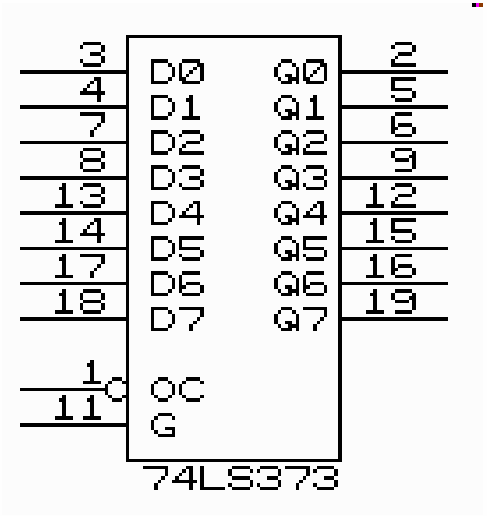
- 8088 có thể hoạt động ở hai tần số là 5Mhz và 8Mhz
- 8284 được thiết kế chuyên dụng,tạo xung nhịp điều khiển cho 8088
- Khối Reset đưa tín hiệu RESET tới vi xử lý 8088





Các mạch phụ trợ cho 8088 (tiếp)

- Mạch chốt 74LS373: dùng để chốt địa chỉ và trạng thái
  - Chân OC (Output Control): tích cực mức thấp, điều khiển cho phép/ cấm dữ liệu ra
  - Chân G: chân chốt (Khi OC=0)
    - ✓G=0 -> đầu ra là dữ liệu trước đó
    - ✓G=1 -> đầu ra = đầu vào

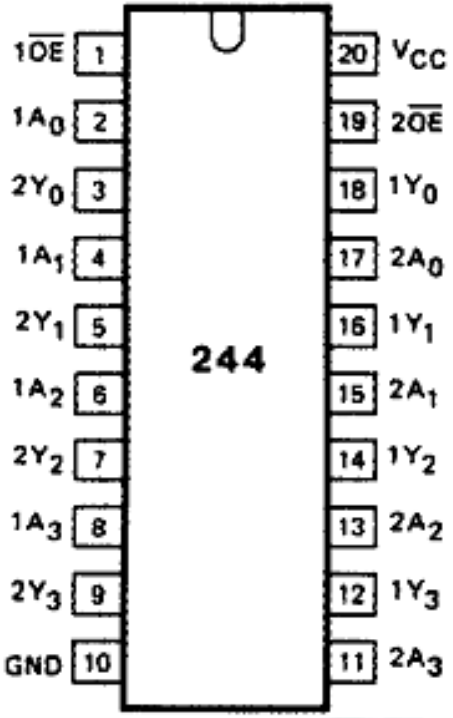


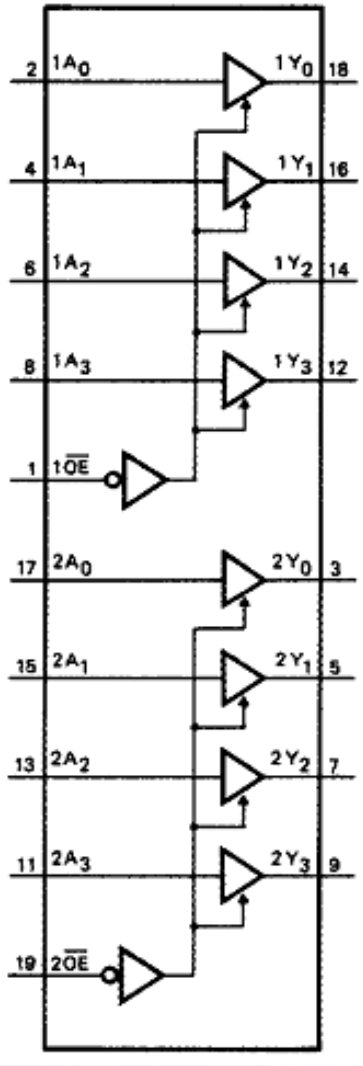
Output Control	Enable G	D	Output
L	H	H	H
L	H	L	L
L	L	X	Q_0
H	X	X	Z

Bảng thật của 74LS373

Các mạch phụ trợ cho 8088 (tiếp)

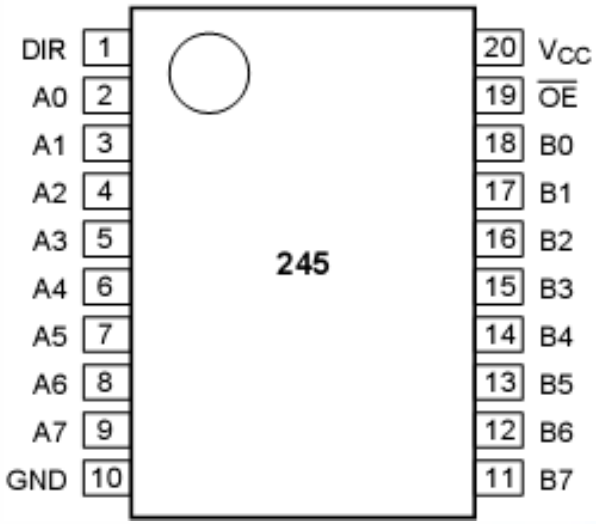
- **Mạch khuếch đại & đệm một chiều 74HC244**
  - Có thể đệm từng nibble (4bit) hoặc đệm một byte (8bit)
  - Điều khiển bởi hai chân 1OE và 2OE (Output Enable)

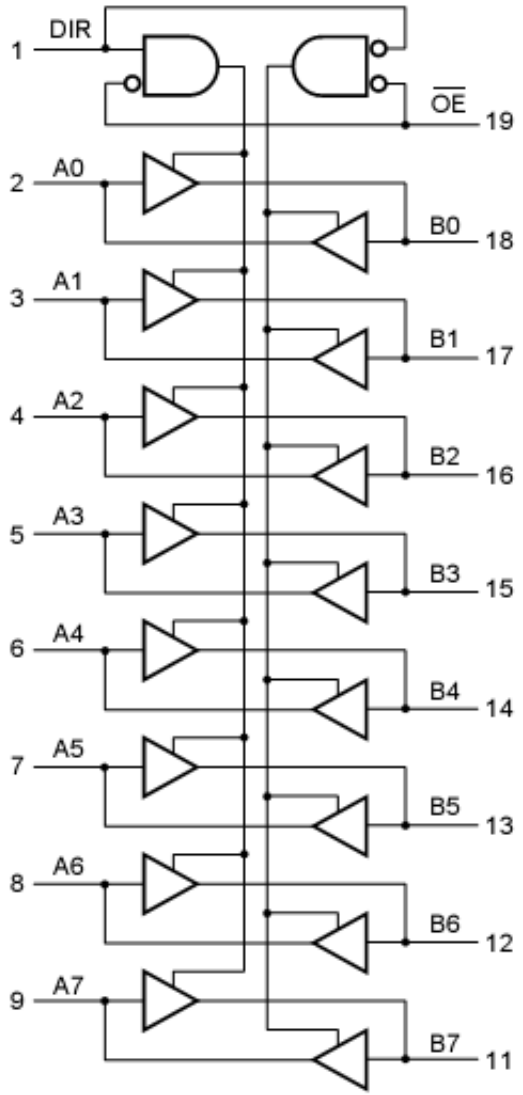




Các mạch phụ trợ cho 8088 (tiếp)

- **Mạch khuếch đại, đệm 2 chiều 74HC245**
  - Đệm từng byte theo cả hai chiều
  - Điều khiển dữ liệu qua chân OE (Output Enable)
  - Điều khiển chiều truyền qua chân DIR (Direction)





Nội dung chương 4

4.1. Bộ vi xử lý 8088

4.2. Các mạch phụ trợ cho 8088

4.3. Các chu kỳ bus

4.4. Bộ nhớ bán dẫn

4.5. Giải mã địa chỉ cho bộ nhớ

4.3. Chu kỳ bus của 8088

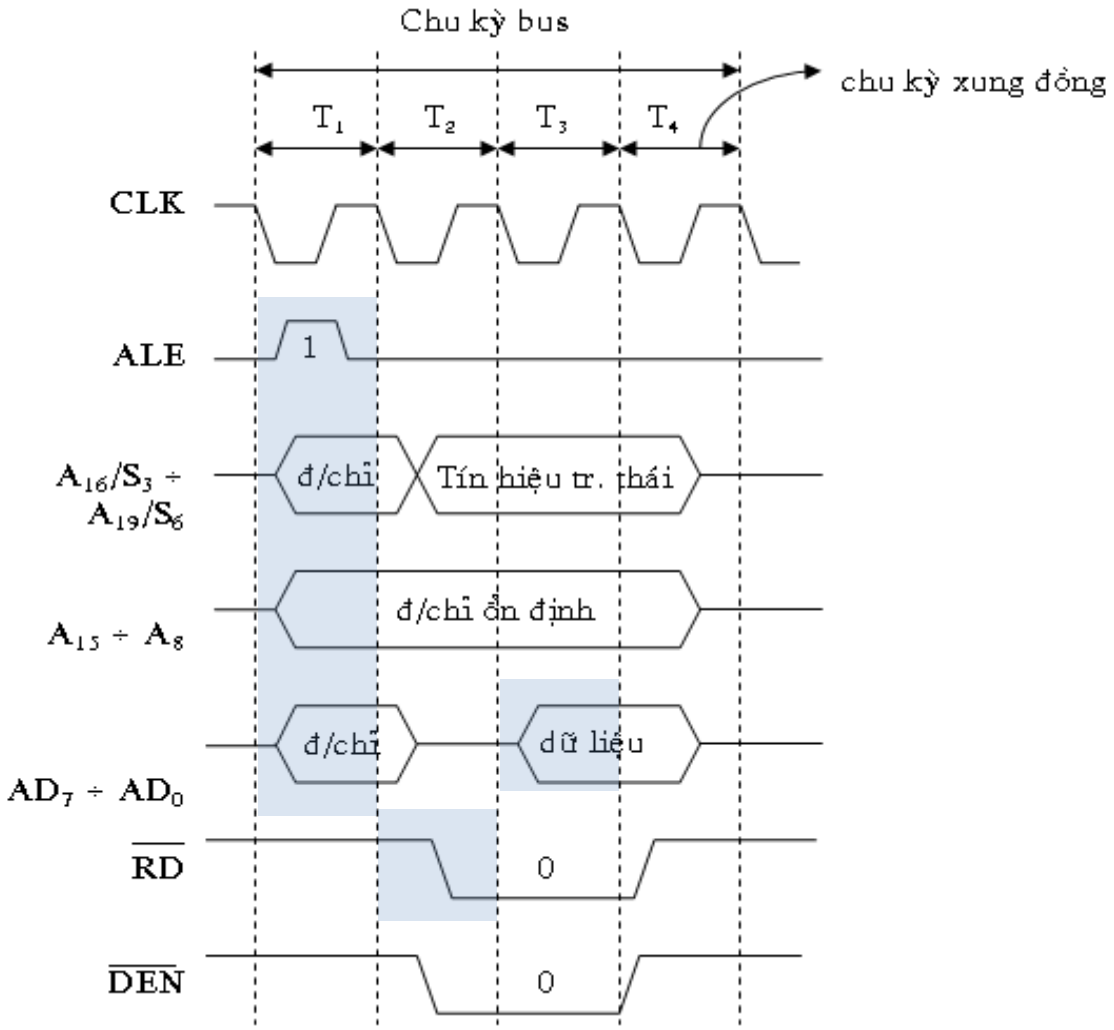
- Một chu kỳ bus (bus cycle) định nghĩa một thao tác cơ bản của bộ vi xử lý để giao tiếp với thiết bị bên ngoài. Ví dụ: đọc/ghi bộ nhớ, đọc/ghi cổng vào ra…
- Chu kỳ bus của 8088 gồm tối thiểu bốn chu kỳ đồng hồ, được đặt tên là T1, T2, T3 và T4
- Nếu không có chu kỳ bus nào, bộ vi xử lý chuyển sang trạng thái idle (idle state)
- Trạng thái chờ (Wait state) cũng có thể chèn vào một chu kỳ bus

Chu kỳ đọc dữ liệu

T1: 8088 phát địa chỉ lên bus địa chỉ

T2: 8088 phát tín hiệu điều khiển đọc

T3: Dữ liệu từ bên ngoài được chuyển lên bus dữ liệu



4.4. Bộ nhớ bán dẫn

- Gồm 2 loại chính: ROM và RAM
- ROM (Read Only Memory): bộ nhớ chỉ đọc
- Đặc điểm:
  - Bộ nhớ chủ yếu dùng để đọc thông tin
  - Bộ nhớ không khả biến
  - Chứa các chương trình và dữ liệu cố định với hệ thống

ROM (tiếp)

- Các loại bộ nhớ ROM:
  - Maskable ROM (ROM mặt nạ): thông tin được ghi khi chế tạo
  - PROM (Programmable ROM):
    - ✓Khi chế tạo chưa có thông tin
    - ✓Cho phép ghi thông tin được 1 lần bằng thiết bị chuyên dụng
  - EPROM (Erasable PROM):
    - ✓Cho phép xóa bằng tia cực tím
    - ✓Ghi lại bằng thiết bị nạp EPROM
  - EEPROM (Electrically Erasable PROM):
    - ✓Có thể xóa bằng tín hiệu điện và ghi lại thông tin ngay trong mạch làm việc (không cần thiết bị ghi riêng)
    - ✓Có thể xóa và ghi lại ở mức từng Byte
    - ✓Dung lượng nhỏ
  - Flash Memory: giống EEPROM nhưng:
    - ✓Đọc/ghi theo từng block
    - ✓Tốc độ rất nhanh
    - ✓Dung lượng lớn

RAM (Random Access Memory)

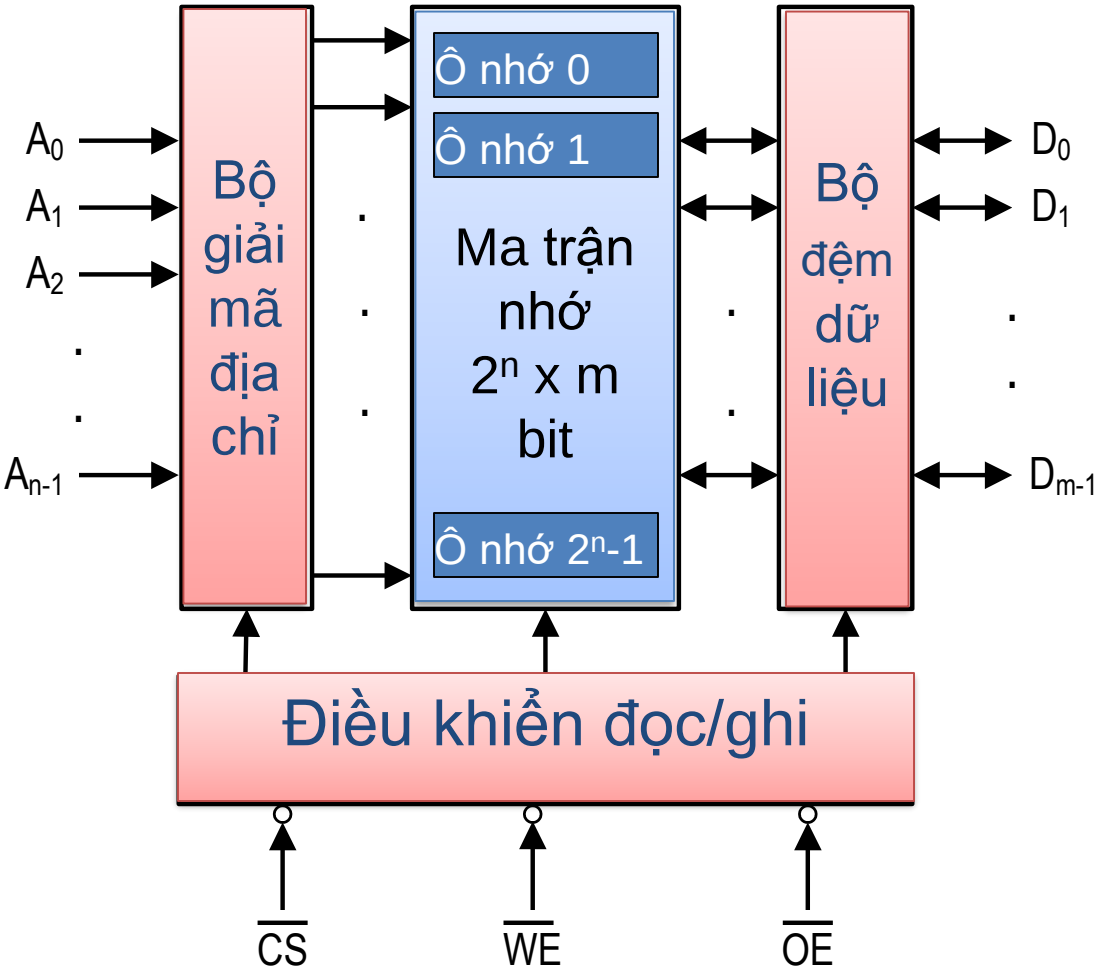
- RAM (Random Access Memory): bộ nhớ truy cập ngẫu nhiên
- Đặc điểm:
  - Là bộ nhớ đọc/ghi (Read/Write Memory – RWM)
  - Bộ nhớ khả biến
  - Chứa các thông tin tạm thời

RAM (tiếp)

- Các loại bộ nhớ RAM:
  - SRAM (Static): RAM tĩnh
    - ✓Mỗi phần tử nhớ là một mạch lật 2 trạng thái ổn định → thông tin trên SRAM ổn định
    - ✓Tốc độ nhanh
    - ✓Dung lượng chip nhớ nhỏ
    - ✓Giá thành đắt
    - ✓Thường dùng làm bộ nhớ Cache

  - DRAM (Dynamic): RAM động
    - ✓Mỗi phần tử nhớ là một tụ điện rất nhỏ → cứ sau một khoảng thời gian thì điện tích trên tụ điện sẽ bị mất, cho nên thông tin trên DRAM không ổn định → khắc phục bằng mạch làm tươi (refresh) DRAM
    - ✓Tốc độ chậm (do mất thời gian làm tươi DRAM)
    - ✓Dung lượng chip nhớ lớn
    - ✓Giá thành rẻ
    - ✓Thường dùng làm bộ nhớ chính

Mô hình cơ bản của chip nhớ



Mô hình cơ bản của chip nhớ (tiếp)

- Có n chân địa chỉ ($A_{n-1} \div A_0$) : vận chuyển vào chip nhớ được n bit địa chỉ đồng thời → trong chip nhớ có 2^n từ nhớ.
- Có m chân dữ liệu: ($D_{m-1} \div D_0$) : cho phép vận chuyển đồng thời được m bit dữ liệu → độ dài từ nhớ là m bit.

  → Dung lượng của chip nhớ là: 2^n x m bit
- Các chân tín hiệu điều khiển:
  - $\overline{CS}$ (Chip Select): tín hiệu điều khiển chọn chip nhớ làm việc
  - $\overline{OE}$ (Output Enable): tín hiệu điều khiển đọc dữ liệu của 1 từ nhớ đã được xác định.
  - $\overline{WE}$ (Write Enable): tín hiệu điều khiển ghi dữ liệu vào 1 từ nhớ đã được xác định.

Hoạt động của chip nhớ

- Hoạt động đọc:
  - Các bit địa chỉ được đưa đến các chân địa chỉ.
  - Tín hiệu điều khiển chọn chip nhớ làm việc được đưa đến $\overline{CS}$
  - Tín hiệu điều khiển đọc đưa đến $\overline{OE}$
  - Dữ liệu từ ngăn nhớ tương ứng với địa chỉ đã có sẽ được đưa ra các chân dữ liệu.

Hoạt động của chip nhớ (tiếp)

- Hoạt động ghi:
  - Các bit địa chỉ được đưa đến các chân địa chỉ
  - Dữ liệu cần ghi được đưa đến các chân dữ liệu
  - Tín hiệu điều khiển chọn chip được đưa đến $\overline{CS}$
  - Tín hiệu điều khiển ghi được đưa đến $\overline{WE}$
  - Dữ liệu từ các chân dữ liệu sẽ được ghi vào ngăn nhớ tương ứng.

Một số chip nhớ thông dụng

- **EPROM**
  - EPROM 2716 2K x 8bit
  - EPROM 2732 4K x 8bit
  - EPROM 2764 8K x 8bit
  - EPROM 27128 16K x 8bit
  - EPROM 27256 32K x 8bit
  - EPROM 27512 64K x 8bit
  - EPROM 27010 128K x 8bit
- **RAM**
  - SRAM 4361 64K x 1bit
  - SRAM 4363 16K x 4bit
  - SRAM 43254 64K x 4bit
  - SRAM 43256A 32K x 8bit

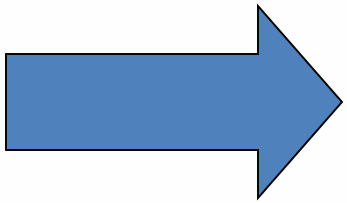
Mở rộng dung lượng cho bộ nhớ bán dẫn

- Mỗi IC nhớ có một dung lượng nhớ xác định: một IC có n bit địa chỉ và m bit dữ liệu sẽ có dung lượng là 2^n x m bit
- Để tăng dung lượng nhớ -> phải ghép nối nhiều IC với nhau nhằm
  - Tăng độ dài ngăn nhớ
  - Tăng số lượng ngăn nhớ
  - Thiết kế kết hợp

Tăng độ dài ngăn nhớ

Yêu cầu tổng quát

- Cho các chip nhớ 2^n x m bit
- Thiết kế môđun nhớ 2^n x (k.m) bit



Sử dụng k chip nhớ

Tăng độ dài ngăn nhớ

Ví dụ:

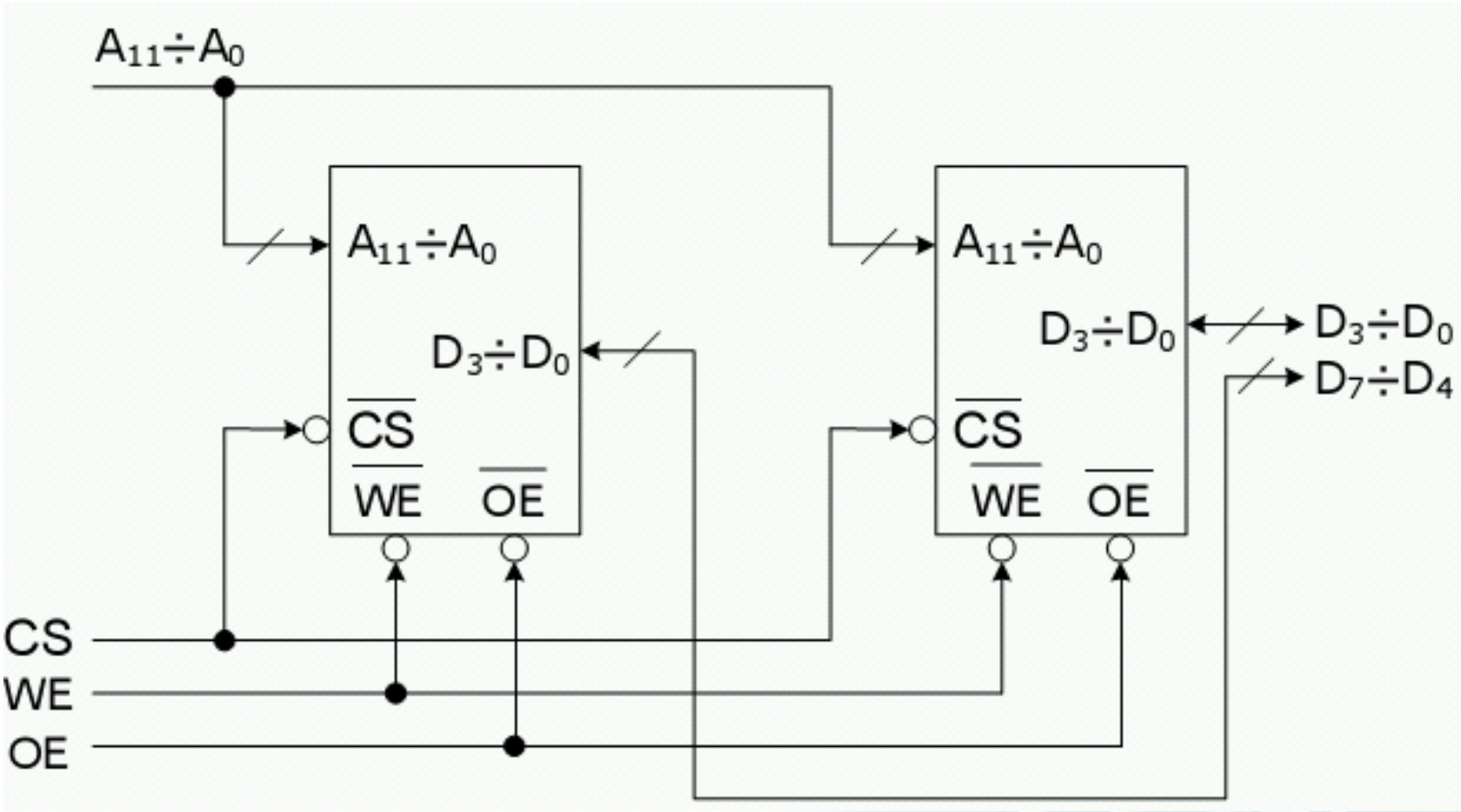
- Cho chip nhớ SRAM 4K x 4 bit
- Thiết kế môđun nhớ 4K x 8 bit

Giải

- Dung lượng chip nhớ 2^{12} x 4 bit
  - Số chân địa chỉ của chip nhớ: 12 chân
  - Số chân dữ liệu của chip nhớ: 4 chân
- Dung lượng môđun nhớ 2^{12} x 8 bit
  - Số chân địa chỉ của môđun nhớ: 12 chân
  - Số chân dữ liệu của môđun nhớ: 8 chân

-> Cần 2 chip nhớ để thiết kế môđun

Tăng độ dài ngăn nhớ

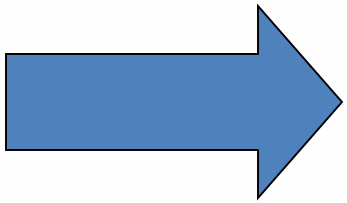


Môđun nhớ 4K x 8 bit

Tăng số lượng ngăn nhớ

Yêu cầu tổng quát

- Cho các chip nhớ 2^n x m bit
- Thiết kế môđun nhớ 2^k x 2^n x m bit



- Sử dụng 2^k chip nhớ
- n+k: số đường địa chỉ

Tăng độ dài ngăn nhớ

Ví dụ:

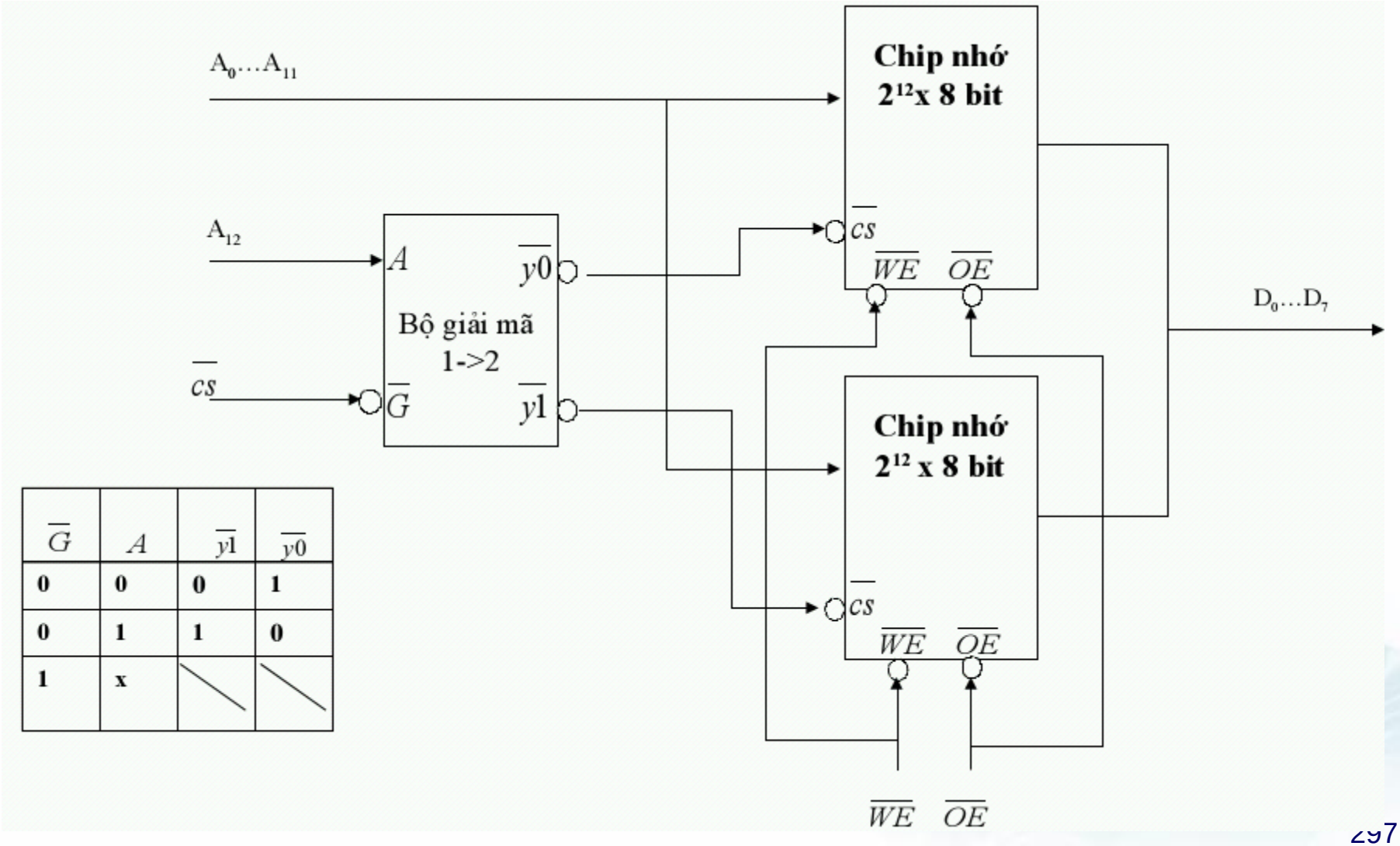
- Cho các chip nhớ SRAM 4K x 8 bit
- Thiết kế môđun nhớ 8K x 8 bit

Giải

- Dung lượng chip nhớ 2^{12} x 8 bit
  - Số chân địa chỉ của chip nhớ: 12 chân
  - Số chân dữ liệu của chip nhớ: 8 chân
- Dung lượng môđun nhớ 2^{13} x 8 bit
  - Số chân địa chỉ của môđun nhớ: 13 chân
  - Số chân dữ liệu của môđun nhớ: 8 chân

-> Cần 2 chip nhớ để thiết kế môđun

Tăng độ dài ngăn nhớ



$\overline{G}$	A	$\overline{y1}$	$\overline{y0}$
0	0	0	1
0	1	1	0
1	x		

Mở rộng dung lượng cho bộ nhớ bán dẫn

- Một số ví dụ khác
  - Thiết kế môđun nhớ 16k x 8 bit từ các chip nhớ 4k x 8 bit
  - Thiết kế môđun nhớ 32k x 8 bit từ các chip nhớ 4k x 8 bit
  - Thiết kế môđun nhớ 8k x 8 bit từ các chip nhớ 4k x 4 bit

Nội dung chương 4

4.1. Bộ vi xử lý 8088

4.2. Các mạch phụ trợ cho 8088

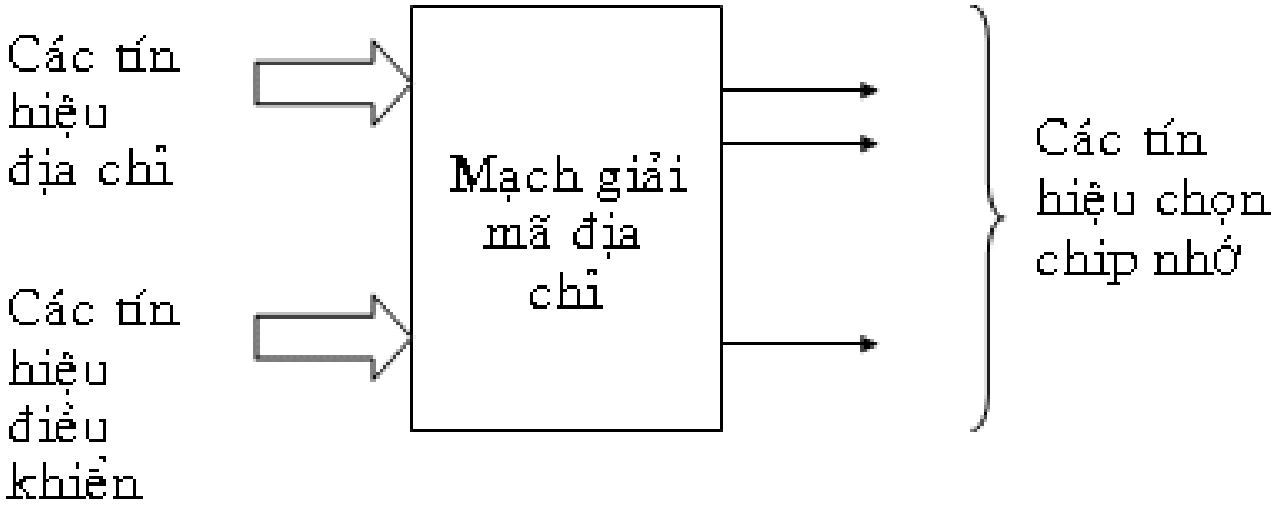
4.3. Các chu kỳ bus

4.4. Bộ nhớ bán dẫn

4.5. Giải mã địa chỉ cho bộ nhớ

4.5. Giải mã địa chỉ cho bộ nhớ

- Mỗi IC cần được ánh xạ vào không gian nhớ ở một vị trí xác định, như vậy mỗi IC nhớ chiếm một vùng địa chỉ nhất định
- Mạch cho phép mỗi IC chiếm một vùng nhớ xác định gọi là mạch giải mã địa chỉ



Giải mã địa chỉ cho bộ nhớ

- Có ba phương pháp thiết kế mạch giải mã địa chỉ
  - Dùng mạch NAND
  - Dùng bộ giải mã có sẵn
  - Dùng PROM

- Ví dụ 1: Thiết kế mạch giải mã địa chỉ cho IC SRAM dung lượng 4k x 8 bit, có địa chỉ bắt đầu là 0x00000

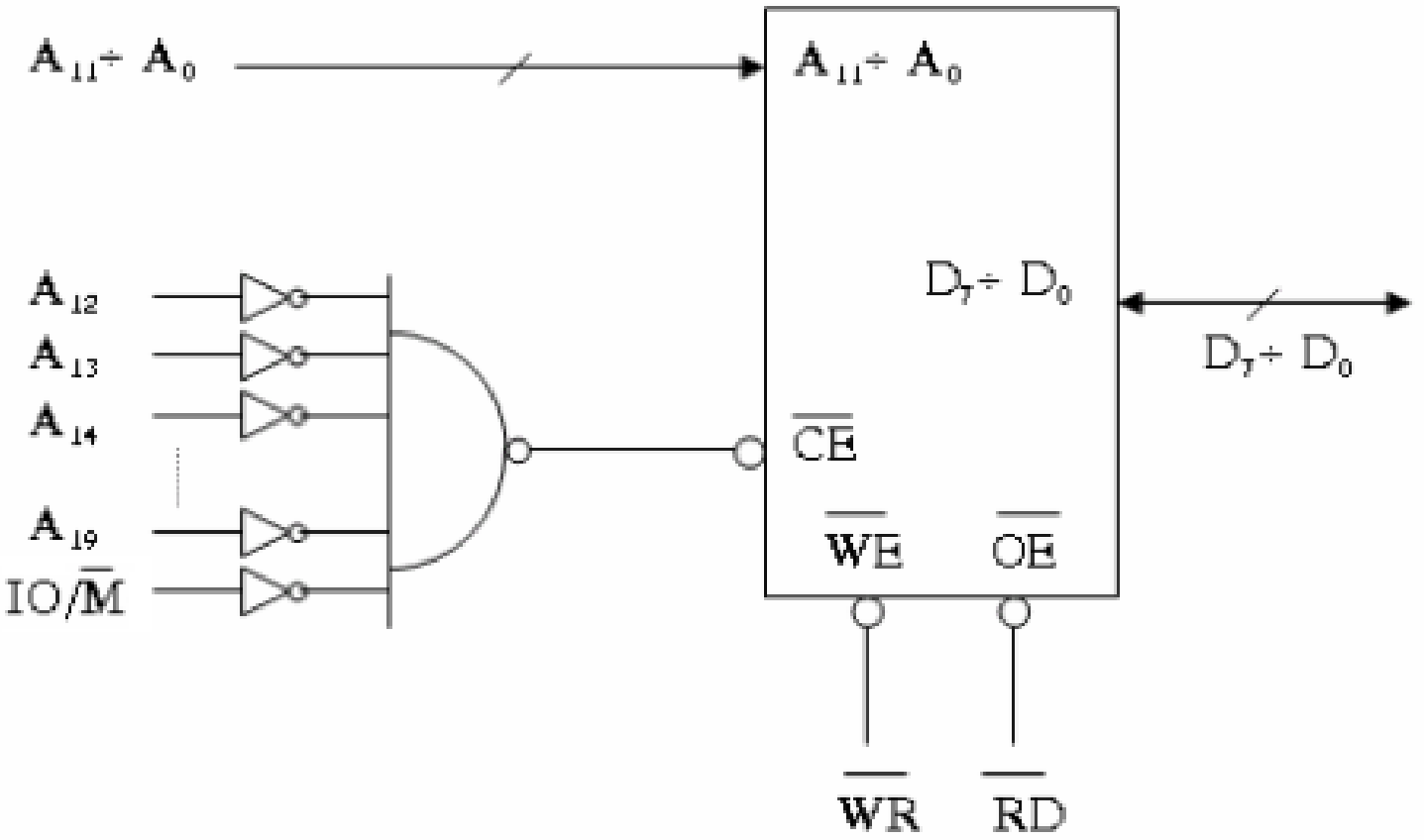
Giải:

- Dung lượng 4k x 8 bit
  - ✓ Địa chỉ bắt đầu : 0x00000
  - ✓ Địa chỉ kết thúc : 0x00FFF

-> phần ko đổi: 8 bit cao tương ứng tín hiệu từ các chân địa chỉ A12->A19

-> phần thay đổi: 12 bit thấp (A0->A11)

Thiết kế mạch giải mã dùng mạch NAND



Mạch giải mã thiết kế theo ví dụ 1

- Ví dụ 2: Thiết kế mạch giải mã địa chỉ cho IC EPROM dung lượng 8k x 8 bit, có địa chỉ bắt đầu là 0xFE000

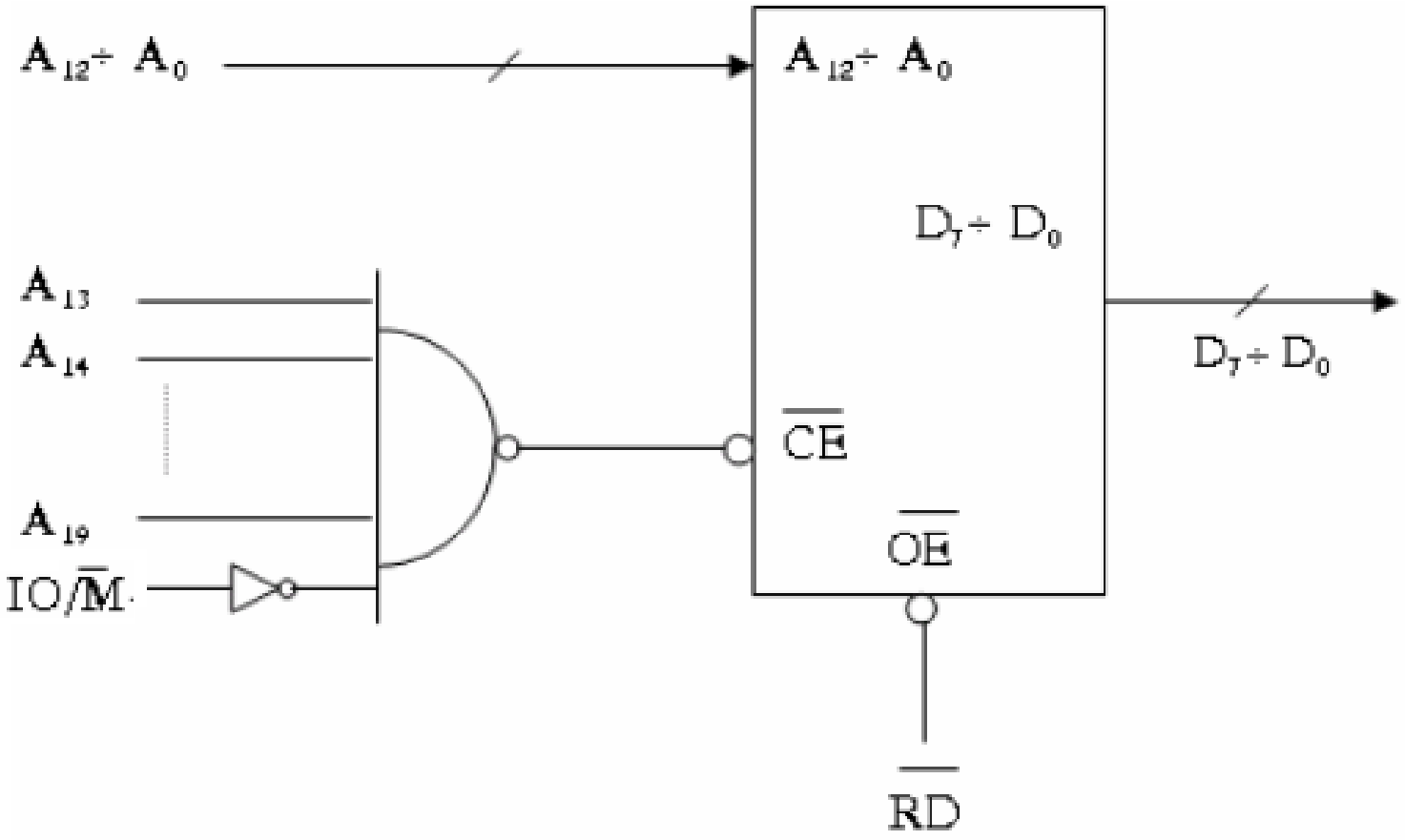
Giải:

- Dung lượng 8k x 8 bit
  - ✓ Địa chỉ bắt đầu : 0xFE000
  - ✓ Địa chỉ kết thúc : 0xFFFFF

-> phần ko đổi: 7 bit cao tương ứng tín hiệu từ các chân địa chỉ A13->A19

-> phần thay đổi: 13 bit thấp (A0->A12)

Thiết kế mạch giải mã dùng mạch NAND



Mạch giải mã thiết kế theo ví dụ 2

Thiết kế mạch giải mã dùng mạch NAND

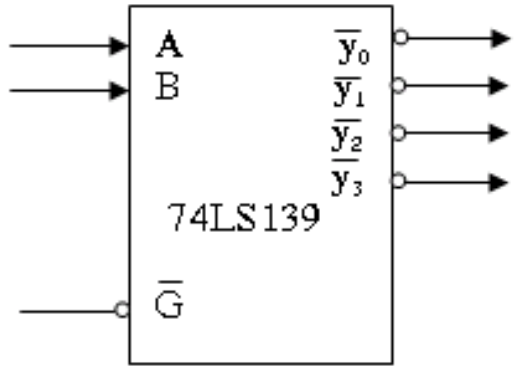
- Một số ví dụ:
  - Thiết kế mạch giải mã địa chỉ cho IC SRAM dung lượng 32k x 8bit, có địa chỉ bắt đầu là 0x00000
  - Thiết kế mạch giải mã địa chỉ cho IC EPROM dung lượng 64k x 8bit, có địa chỉ bắt đầu là 0xF0000

Thiết kế dùng bộ giải mã có sẵn

- Một số bộ giải mã thông dụng
  - 74LS138: bộ giải mã 3 đầu vào, 8 đầu ra đảo
  - 74LS139: bộ giải mãi 2 đầu vào, 4 đầu ra đảo

Bộ giải mã 74LS139

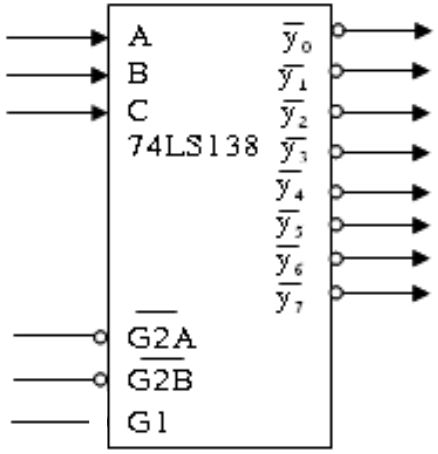
- 2 đầu vào (A, B), 4 đầu ra đảo (y0, y1, y2, y3)
- Điều khiển bằng chân G



$\overline{G}$	Vào		Ra			
	A	B	$\overline{y}_0$	$\overline{y}_1$	$\overline{y}_2$	$\overline{y}_3$
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	x	x	1	1	1	1

Bộ giải mã 74LS138

- 3 đầu vào (A, B, C), 8 đầu ra đảo (y0 -> y7)
- 3 tín hiệu điều khiển (G2A, G2B, G1)



			Vào			Ra							
G1	$\overline{G2A}$	$\overline{G2B}$	C	B	A	$\overline{y_0}$	$\overline{y_1}$	$\overline{y_2}$	$\overline{y_3}$	$\overline{y_4}$	$\overline{y_5}$	$\overline{y_6}$	$\overline{y_7}$
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
Các tổ hợp khác			x	x	x	1	1	1	1	1	1	1	1

Thiết kế dùng bộ giải mã có sẵn

- Ví dụ 1:

  Cho các IC SRAM 2k x 8bit, hãy thiết kế bộ giải mã địa chỉ cho môđun nhớ 8k x 8bit có địa chỉ bắt đầu là 0x00000

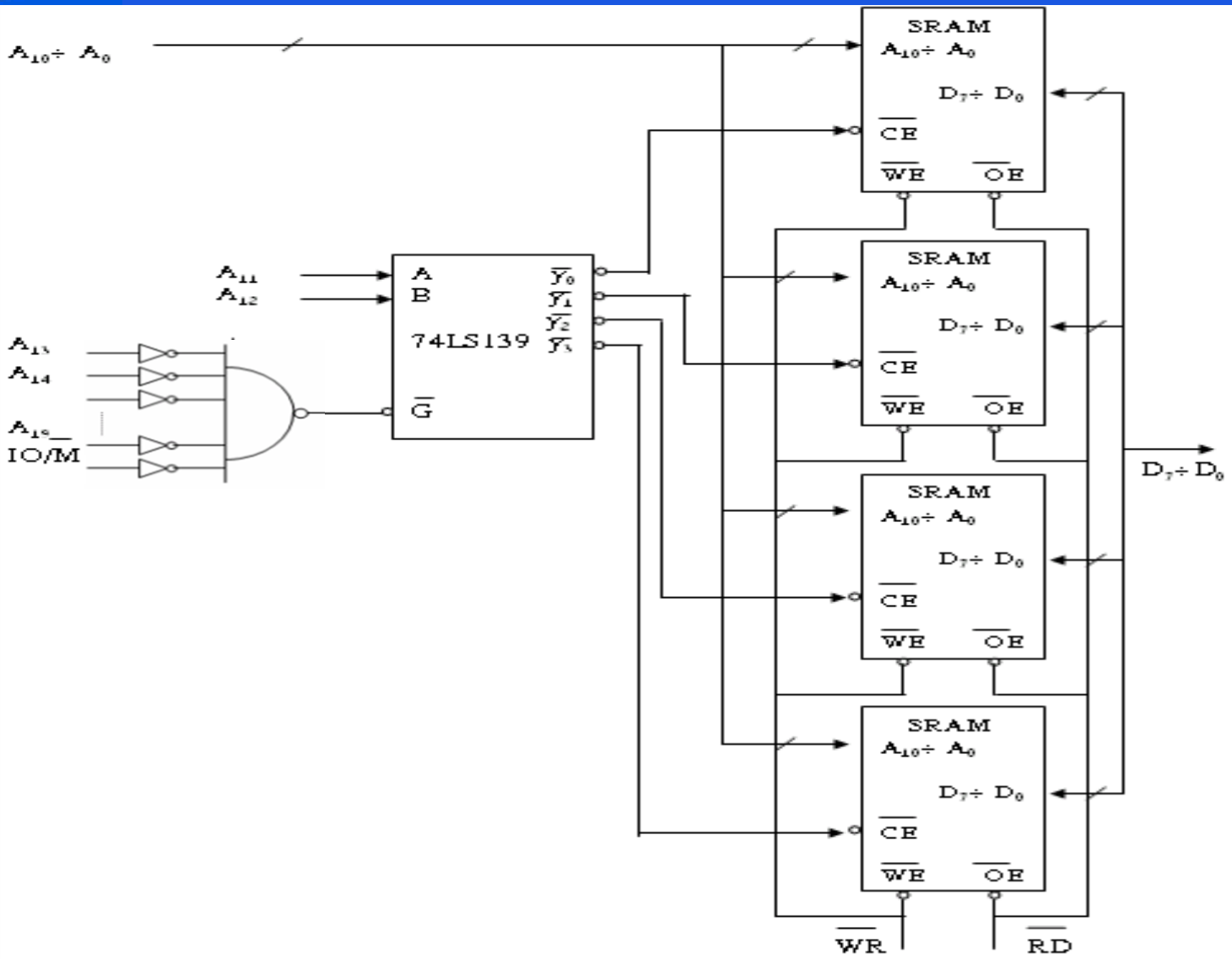
Giải:

- Xác định số IC SRAM cần thiết: 4 IC
- Xác định phân vùng địa chỉ cho từng IC
- Thiết kế mạch giải mã địa chỉ và nối ghép các IC nhớ

Phân vùng địa chỉ cho từng IC

		A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
IC1	00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	007FFH	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
IC2	00800H	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	00FFFH	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
IC3	01000H	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	017FFH	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1
IC4	01800H	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
	01FFFH	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
		Không thay đổi							nối với bộ GM 2→4		Nối thẳng vào IC										

Thiết kế mạch giải mã và nối ghép các IC nhớ



Thiết kế dùng bộ giải mã có sẵn

- Ví dụ 2:

  Cho các IC EPROM 16k x 8bit, hãy thiết kế bộ giải mã địa chỉ cho môđun nhớ EPROM 32k x 8bit có địa chỉ bắt đầu là 0xF8000 (Yêu cầu sử dụng bộ giải mãi 74LS138)

Giải:

- Xác định số IC EPROM cần thiết: 2 IC
- Xác định phân vùng địa chỉ cho từng IC
- Thiết kế mạch giải mã địa chỉ và nối ghép các IC nhớ

Phân vùng địa chỉ cho từng IC

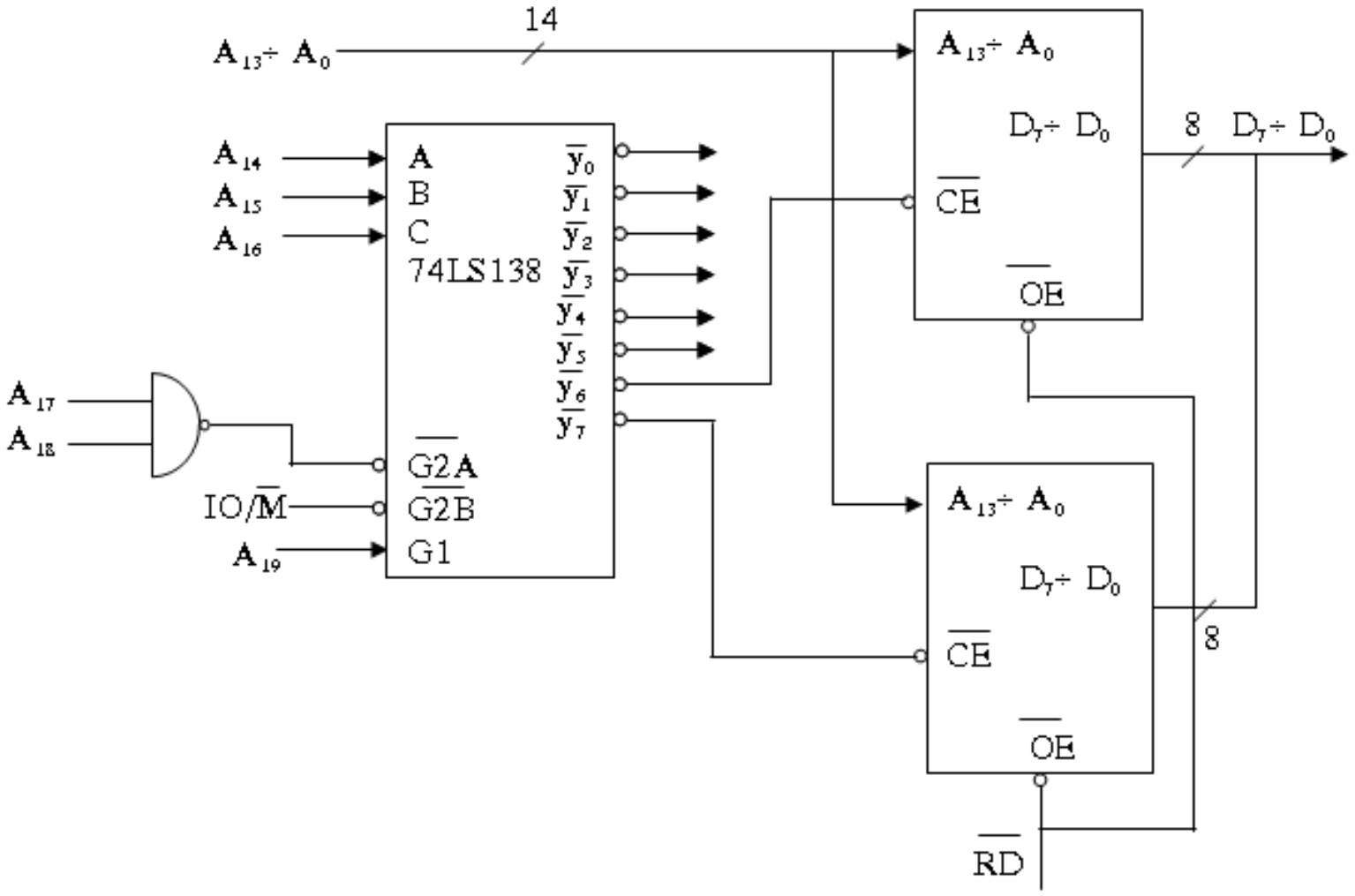
		A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
IC1	F8000H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	FBFFFH	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
IC2	FC000H	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Không đổi

nối với bộ GM 3→8

Nối thẳng vào IC

Thiết kế mạch giải mã và nối ghép các IC nhớ



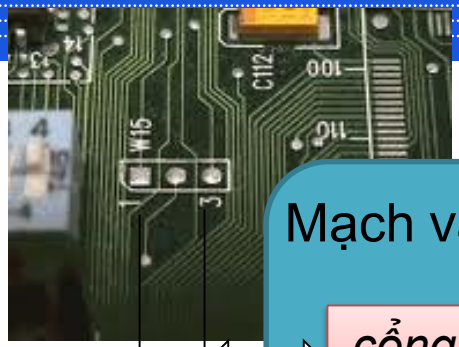
CHƯƠNG 5

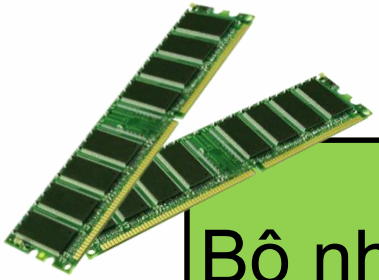
Nối ghép vào ra với 8088

5.1. Định địa chỉ cổng vào-ra

- Nối ghép vào ra là nối ghép giữa bộ VXL với các cổng vào-ra.
- Mỗi cổng vào-ra sẽ có 1 địa chỉ xác định.
- Thiết bị ngoại vi sẽ được nối ghép và trao đổi dữ liệu thông qua các cổng vào-ra.

5.1. Định địa chỉ cổng vào-ra





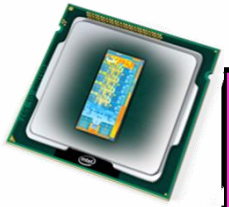
Bộ nhớ chính

Mạch vào ra

cổng IO

cổng IO

TBNV



Bộ xử lý

Mạch vào ra

cổng IO

TBNV

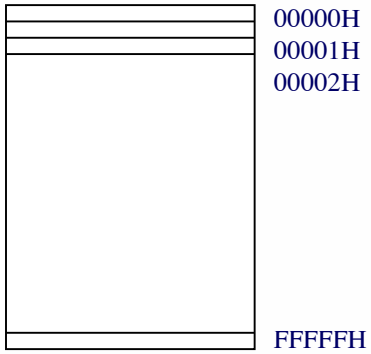
Định địa chỉ cổng vào-ra

- Hai kiểu định địa chỉ cổng vào-ra.
  1. **Vào-ra cách biệt (Isolated IO)**
  2. **Vào-ra theo bản đồ bộ nhớ (Memory mapped IO)**

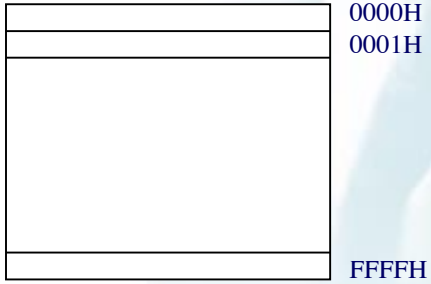
1. Vào-ra cách biệt (Isolated IO)

- ✓ Không gian địa chỉ của bộ nhớ và cổng là riêng biệt
- ✓ 8088 có khả năng quản lý không gian nhớ = 1MB (= 2^{20} bytes)
- ✓ Quản lý không gian vào-ra = 64KB (= 2^{16} bytes)

KG địa chỉ bộ nhớ $A_{19} \div A_0$



K địa chỉ vào ra $A_{15} \div A_0$



Vào-ra cách biệt (Isolated IO)

- Để phân biệt truy nhập cổng IO hay bộ nhớ, 8088 có 1 tín hiệu:

$$IO/\overline{M} = \begin{cases} 1 \rightarrow truy\ cập\ cổng\ vào\ ra \\ 0 \rightarrow truy\ cập\ bộ\ nhớ \end{cases}$$

- Cổng vào $\rightarrow$ được điều khiển bằng tín hiệu (đọc)
- Cổng ra $\rightarrow$ được điều khiển bằng tín hiệu (ghi)
- Cổng vào ra (2 chiều)

Vào-ra cách biệt (Isolated IO)

- ***Các lệnh vào ra trực tiếp:***
  - Được dùng để trao đổi dữ liệu với cổng trong trường hợp cổng được thiết kế theo kiểu vào-ra cách biệt.
  - **Lệnh IN:**

```
IN AL, địa chỉ cổng (8 bit)  (1)
IN AX, địa chỉ cổng (16 bit) (2)
```

cổng cố định, quản lý 256 byte cổng

(1) : nhận dữ liệu từ cổng 8 bit với địa chỉ được cho trong lệnh → AL

VD : `IN AL, 3Ah`

hay `IN AL, 00111010b`

(2) : nhận dữ liệu từ cổng 16 bit với địa chỉ được cho trong lệnh → AX

VD : `IN AX, 40h` → nhận dữ liệu từ 2 cổng 8 bit (cổng byte) với địa chỉ là 40h và 41h

Vào-ra cách biệt (Isolated IO)

```
IN AL, DX (8 bit) (3)      }    cổng
IN AX, DX (16 bit) (4)     }    thay đổi
```

(3) : nhận dữ liệu từ cổng byte → AL

(4) : nhận dữ liệu từ cổng word → AX

DX : thanh ghi chứa địa chỉ cổng, địa chỉ 16 bit (quản lý 65535 byte cổng).

VD : Muốn nhận dữ liệu từ cổng 03F8h → AL ?

```
MOV DX, 03F8h
IN AL, DX
```

Vào-ra cách biệt (Isolated IO)

- **Lệnh OUT:**
  - OUT địa chỉ cổng, AL

  (đưa dữ liệu từ AL ra cổng byte)
  - OUT địa chỉ cổng, AX

  (đưa dữ liệu từ AX ra cổng word)
  - VD:

```
OUT DX, AL
OUT DX, AX
```

Vào-ra cách biệt (Isolated IO)

- VD1:

  Giả sử có 1 hệ thống VXL 8088 có 2 cổng vào với địa chỉ tương ứng là 3Ah, 3Bh, và 1 cổng ra có địa chỉ là 40h. Hãy viết 1 đoạn chương trình nhận 100 cặp dữ liệu từ 2 cổng vào, cộng lần lượt từng cặp rồi đưa kết quả ra cổng ra (giả thiết số liệu đủ nhỏ để không bị tràn).

Vào-ra cách biệt (Isolated IO)

Giải:

```
MOV     CX,100      ; số cặp dữ liệu cần nhận
LAP:
   IN     AL, 3Ah   ; nhận dữ liệu từ cổng 3Ah
   MOV    BL, AL    ; đưa dữ liệu AL ? BL
   IN     AL, 3Bh   ; nhận dữ liệu từ cổng 3Bh
   ADD    AL, BL    ; cộng 2 dữ liệu với nhau
   OUT    40h, AL   ; đưa ra cổng 40h
   LOOP   LAP       ; lặp lại
```

Vào-ra cách biệt (Isolated IO)

Thêm: Nếu tổng > 10 → 40h
tổng ≤ 10 → 41h

```
MOV         CX, 100
LAP:
   IN             AL, 3Ah
   MOV      BL, AL
   IN       AL, 3Bh
   ADD      AL, BL
   CMP      AL ,10        ; AL ≤ 10 ?
   JNG      NHAY          ; đúng → nhảy
   OUT      40H, AL       ; sai → đưa AL ra địa chỉ 40h
   JMP      TOP           ; nhảy qua
   NHAY:
      OUT   41H, AL       ; đưa AL ra địa chỉ cổng 41h
   TOP:
      LOOP  LAP           ; lặp lại
```

Định địa chỉ cổng vào-ra

- Hai kiểu định địa chỉ cổng vào-ra.
  1. Vào-ra cách biệt (Isolated IO)
  2. Vào-ra theo bản đồ bộ nhớ (Memory mapped IO)

Vào-ra theo bản đồ bộ nhớ

- Vào-ra theo bản đồ bộ nhớ (Memory mapped IO)
  - Cổng vào ra được đánh địa chỉ theo không gian địa chỉ bộ nhớ (cổng vào ra được ánh xạ từ không gian nhớ)

  → Số bit địa chỉ để đánh cho mỗi cổng = số bit địa chỉ dành cho ngăn nhớ.

  → CPU dùng các lệnh trao đổi với ngăn nhớ để trao đổi dữ liệu cổng.

Vào-ra theo bản đồ bộ nhớ

VD : Lệnh MOV

MOV BL, [2000h]

- Lưu ý: Trong thực tế 1 số bộ VXL không có không gian vào-ra cách biệt, vd:Motorola 680x0.

=

Nội dung chương 5

5.1. Định địa chỉ cổng vào-ra

5.2. Giải mã địa chỉ cho cổng vào-ra

5.3. Nguyên lý bit cổng

5.4. Mạch nối ghép ngoại vi lập trình được 8255A (Programable Peripheral Interface - PPI 8255A)

5.5. Nối ghép truyền dữ liệu nối tiếp

Giải mã địa chỉ cho cổng vào-ra

- Mỗi cổng IO cần phải có 1 địa chỉ xác định.
- Với 8088 cổng có thể được thiết kế theo không gian nhớ, không gian vào-ra 64KB hoặc không gian vào-ra 256 byte.
- Có 2 phương pháp giải mã địa chỉ cổng phổ biến:
  - Giải mã địa chỉ dùng các cổng logic cơ bản
  - Giải mã địa chỉ dùng các bộ giải mã có sẵn

Giải mã địa chỉ dùng các cổng logic

- Giải mã địa chỉ dùng các cổng logic cơ bản:

  Ví dụ 1:

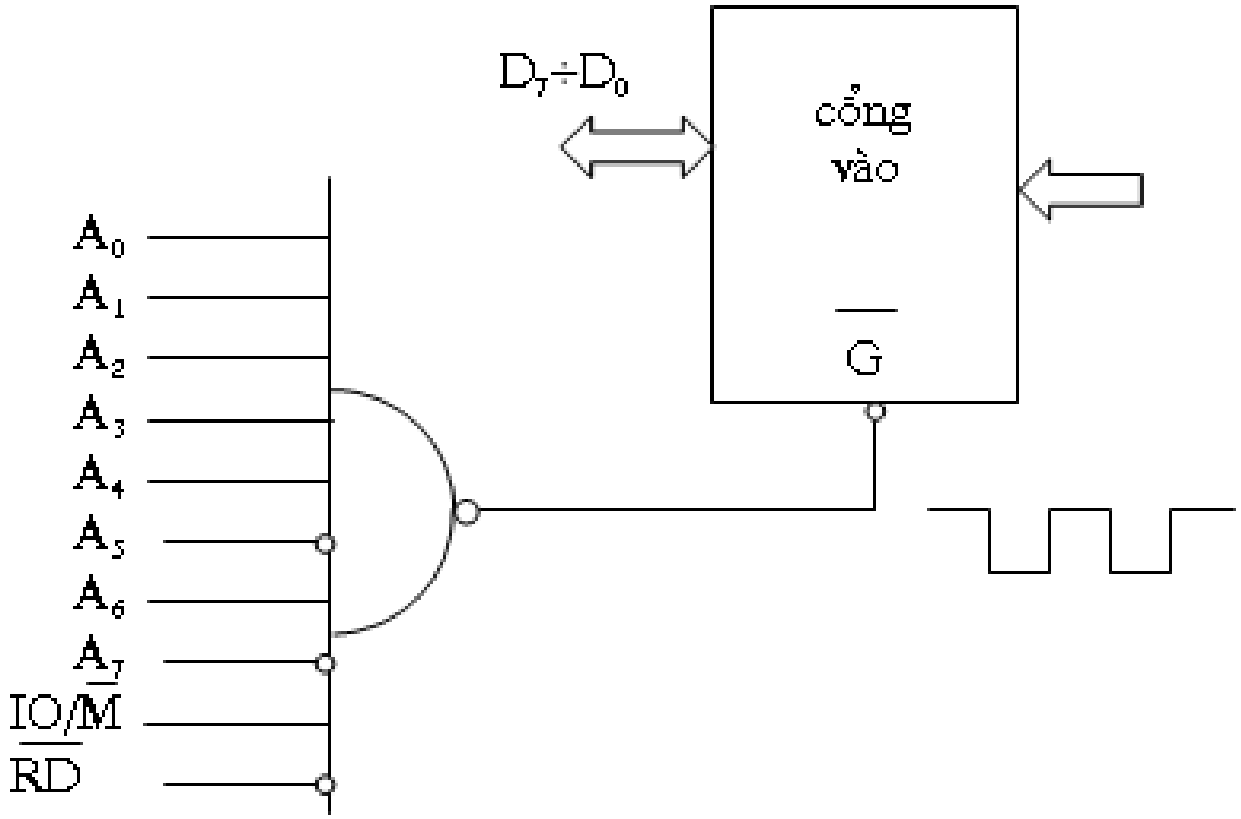
  Thiết kế giải mã địa chỉ cho 1 CỔNG VÀO định địa chỉ theo kiểu vào ra riêng biệt có địa chỉ là 5Fh.

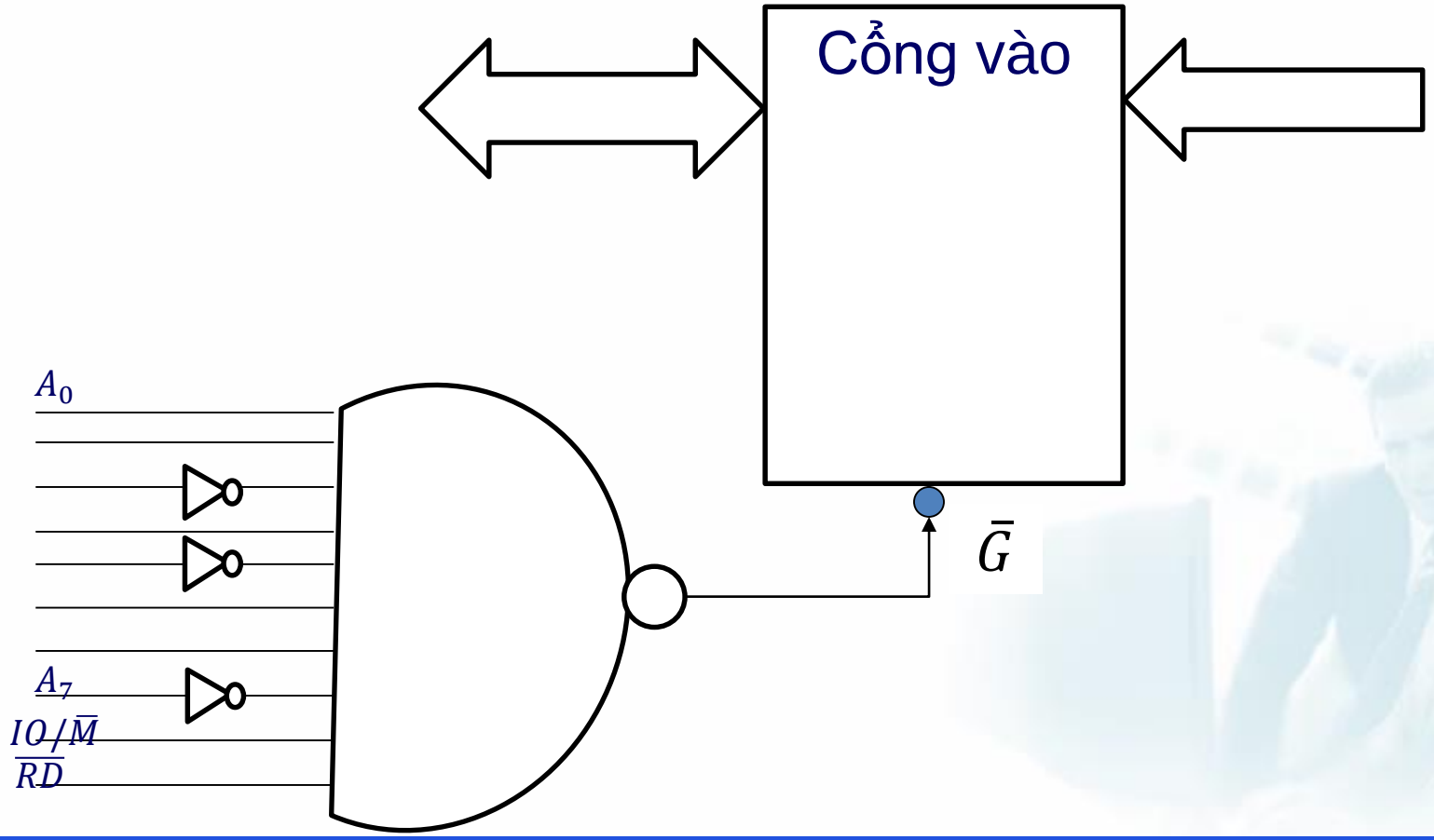
$\overline{IO/M} = 1$ (truy nhập IO)

$\overline{RD} = 0$;

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
5fh	0	1	0	1	1	1	1	1

Giải mã địa chỉ dùng các cổng logic




Cổng vào
A_0
A_7
$IO/\bar{M}$
$\overline{RD}$
$\bar{G}$

Giải mã địa chỉ dùng các cổng logic

- Ví dụ 2:

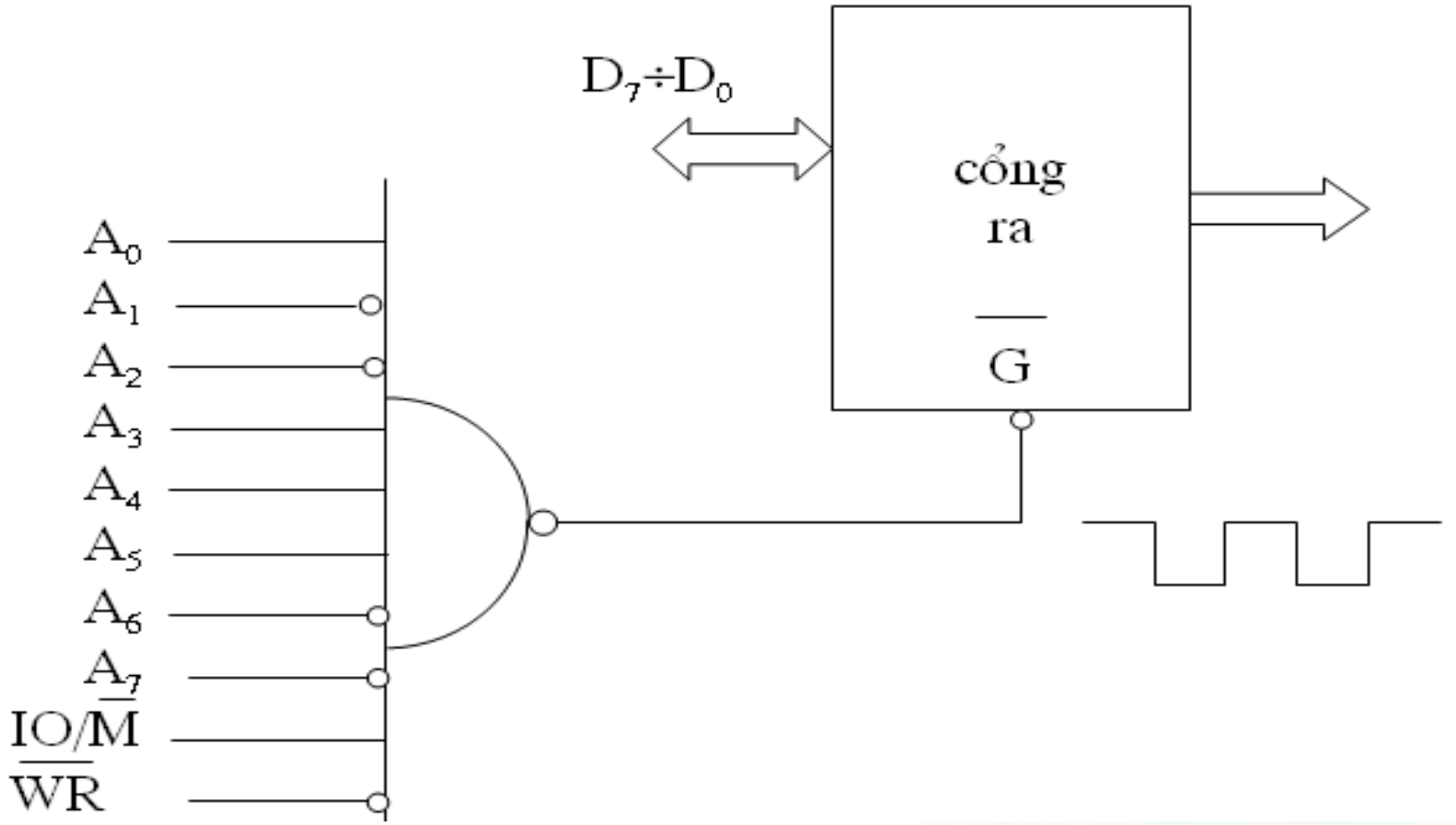
  Thiết kế giải mã địa chỉ cho 1 cổng ra có địa chỉ 39h.

  $\overline{IO/M} = 1$ (truy nhập IO)

  $\overline{WR} = 0$;

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
39h	0	0	1	1	1	0	0	1

Giải mã địa chỉ dùng các cổng logic



- Thiết kế mạch giải mã địa chỉ của 1 cổng vào ra gắn với 1 đèn LED với địa chỉ là 0x3A. Khi dùng lệnh OUT 0x3A,AL với AL=0 sẽ tắt đèn và OUT 0x3A,AL với AL=1 sẽ bật đèn

Giải mã địa chỉ cho cổng vào-ra

Giải mã địa chỉ dùng các bộ giải mã có sẵn:

- Mạch giải mã 74LS139 (vào 2 ra 4)
- Mạch giải mã 74LS138 (vào 3 ra 8)

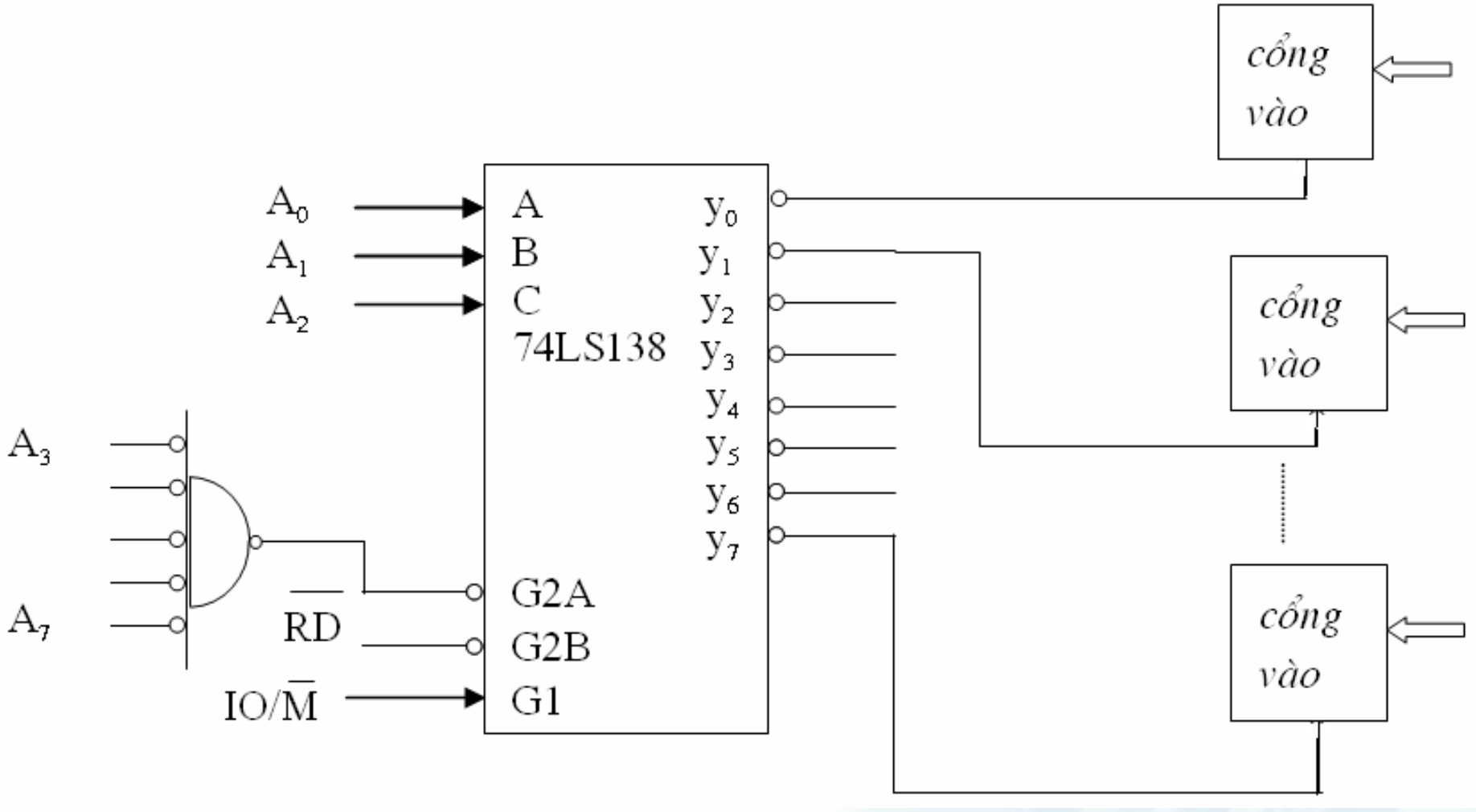
Giải mã dùng bộ giải mã có sẵn

- Ví dụ 1:

  Thiết kế giải mã địa chỉ cho 8 cổng **vào** có địa chỉ từ 00 ÷ 07h

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
00h	0	0	0	0	0	0	0	0
01h	0	0	0	0	0	0	0	1
02h	0	0	0	0	0	0	1	0
03h	0	0	0	0	0	0	1	1
04h	0	0	0	0	0	1	0	0
05h	0	0	0	0	0	1	0	1
06h	0	0	0	0	0	1	1	0
07h	0	0	0	0	0	1	1	1
	cố định					thay đổi		

Giải mã dùng bộ giải mã có sẵn



- Ví dụ 2:

  Thiết kế giải mã địa chỉ cho 8 cổng **vào** có địa chỉ từ 05h ÷ 0Ch

Nội dung chương 5

5.1. Định địa chỉ cổng vào-ra

5.2. Giải mã địa chỉ cho cổng vào-ra

5.3. Nguyên lý bit cổng

5.4. Mạch nối ghép ngoại vi lập trình được 8255A (Programable Peripheral Interface - PPI 8255A)

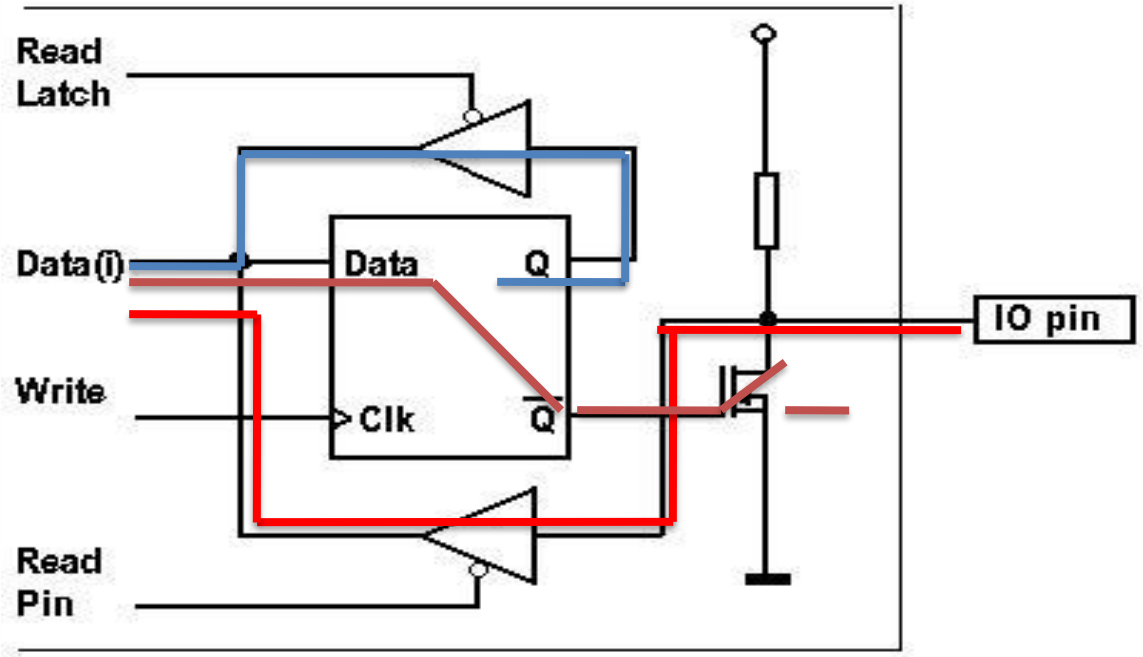
5.5. Nối ghép truyền dữ liệu nối tiếp

Sơ đồ nguyên lý của một bit cổng

1 0 1

0 0 0→ 1

0 1 1

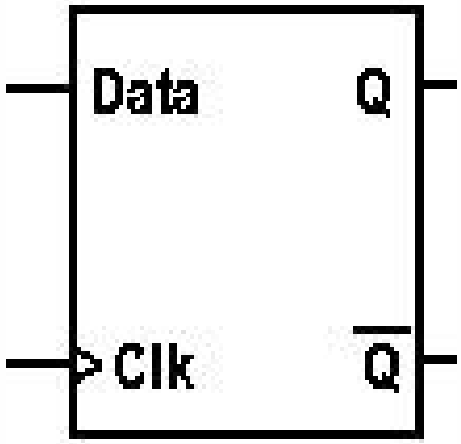


Hình 5.1. Một bit của Port

Nguyên lý bit cổng

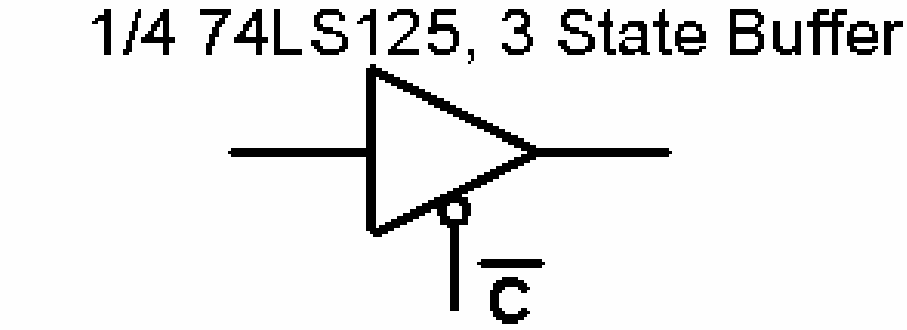
- **Triger (Flip-Flop) D:**
  - Là một mạch chốt dữ liệu, lưu trữ bit dữ liệu.
  - Bảng chân lý:

Clk	D	Q	
0	x	x	(Q ở trạng thái trước)
1	1	1	
1	0	0	



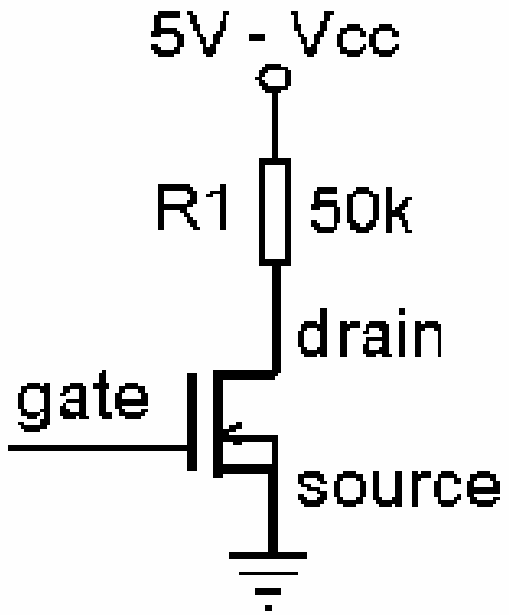
  - ✓ Clk ở mức tích cực: Q = D.
  - ✓ Clk ở mức thấp: trạng thái của Triger D sẽ được chốt, khi đó đầu ra Q luôn luôn giữ giá trị đó cho đến khi Clk chuyển sang mức cao và có giá trị D mới.

- **Đệm 3 trạng thái:**
  - /C =0 : đầu ra bằng đầu vào
  - /C=1 : đầu ra ở trở kháng cao



$$\overline{C} = \begin{cases} 1 \text{ , In = x => Out = Hi Z} \\ 0 \begin{cases} \text{In = 0 => Out = 0} \\ \text{In = 1 => Out = 1} \end{cases} \end{cases}$$

- **MOSFET (Metal Oxide Semiconductor Field Effect Transistor) Transitor trường:**
  - Điều khiển bởi điện áp ở gate.

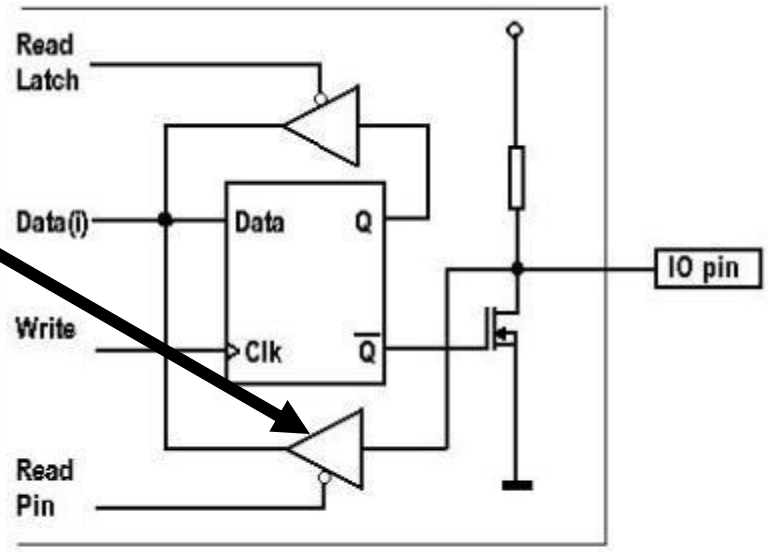


Gate Input = { 0, R(ds) >>
1, R(ds)= 10s Ohm

Nguyên lý bit cổng

- Đọc dữ liệu:
  - Đọc tín hiệu từ chân IO:

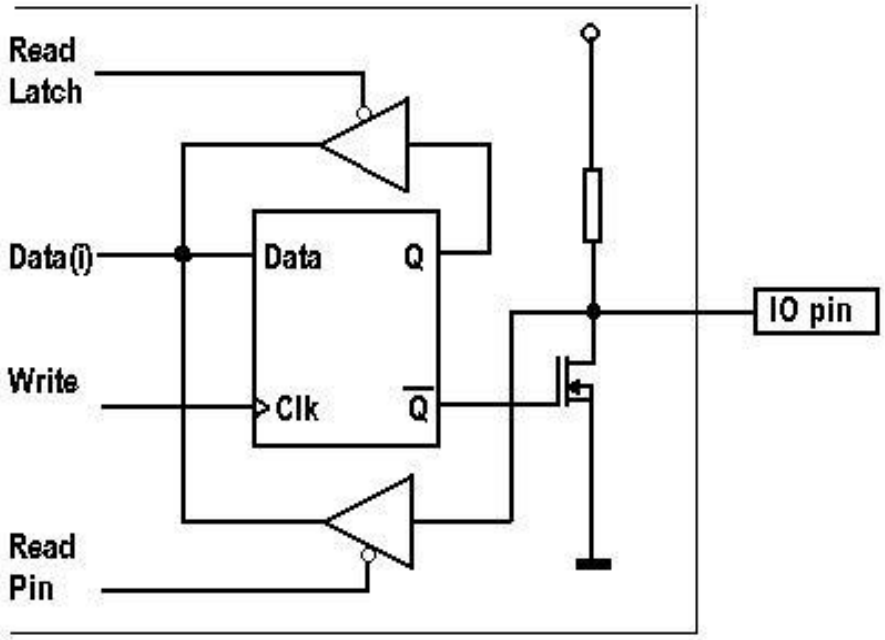
    Read Pin =0: Đệm 3 trạng thái thông, tín hiệu từ IO pin được truyền tới Data bus



Hình 5.1. Một bit của Port

Nguyên lý bit cổng

- Ghi dữ liệu: ?
  Write=1:Q=Data(i).
  - Data(i)=0:
    /Q=1
    Gate=1
    Rds nhỏ, IO pin =0
  - Data(i)=1:
    /Q=0
    Gate=0
    Rds >>, IO pin =1



Hình 5.1. Một bit của Port

Nội dung chương 5

5.1. Định địa chỉ cổng vào-ra

5.2. Giải mã địa chỉ cho cổng vào-ra

5.3. Nguyên lý bit cổng

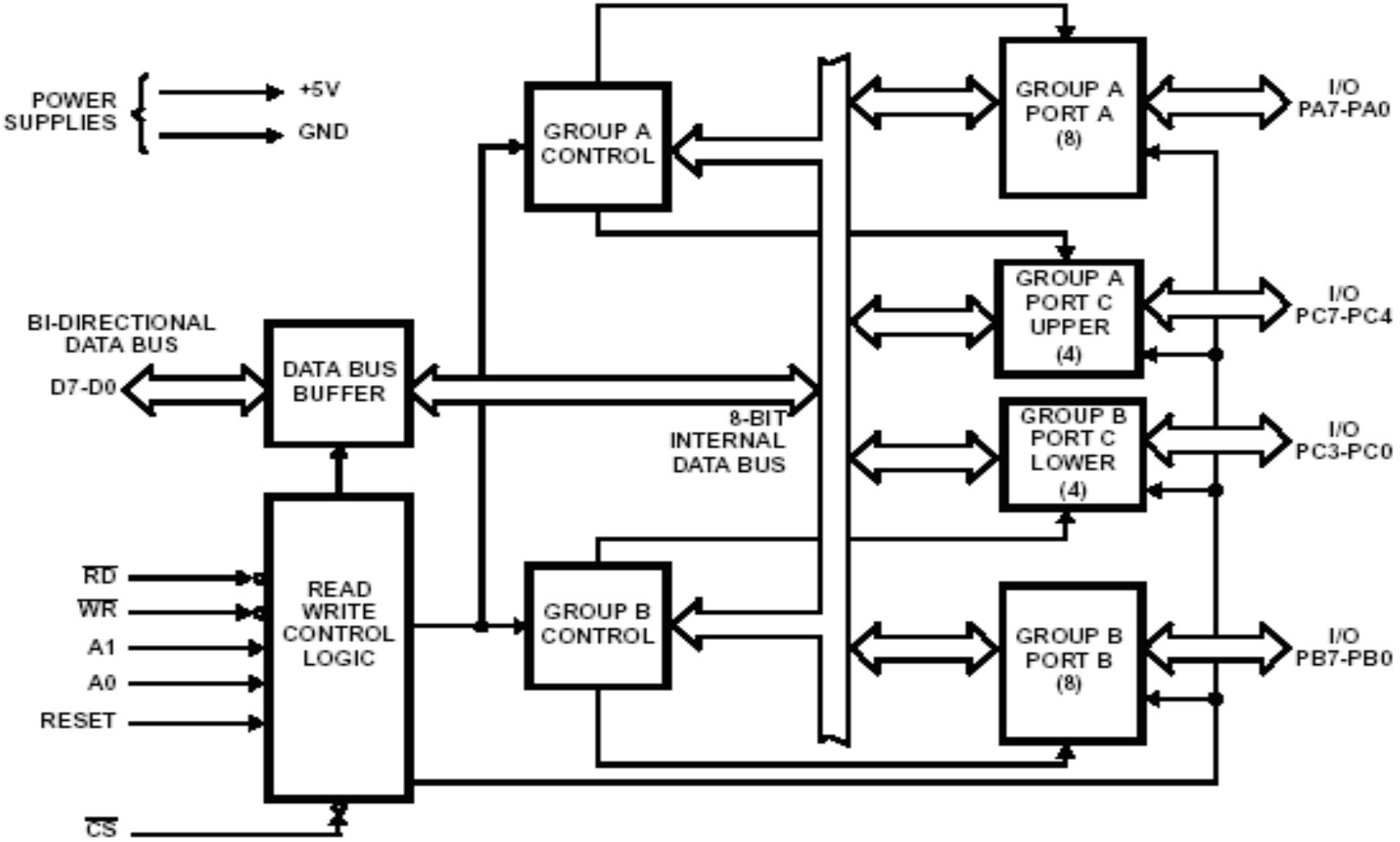
5.4. Mạch nối ghép ngoại vi lập trình được 8255A (Programable Peripheral Interface - PPI 8255A)

5.5. Nối ghép truyền dữ liệu nối tiếp

PPI 8255A

- **Mạch nối ghép ngoại vi lập trình được 8255A (Programable Peripheral Interface - PPI 8255A)**
- Đặc điểm cơ bản:
  + Vi mạch 40 chân.
  + Mạch nối ghép vào ra song song có khả năng lập trình được.
  + Tương thích với bộ vi xử lý của Intel.
  + Có các cổng vào ra song song làm việc ở các chế độ khác nhau.
  + Nguồn 5V.

- Sơ đồ khối :


POWER SUPPLIES
+5V
GND
GROUP A CONTROL
GROUP A PORT A (8)
I/O PA7-PA0
GROUP A PORT C UPPER (4)
I/O PC7-PC4
BI-DIRECTIONAL DATA BUS
D7-D0
DATA BUS BUFFER
8-BIT INTERNAL DATA BUS
GROUP B PORT C LOWER (4)
I/O PC3-PC0
RD
WR
A1
A0
RESET
READ WRITE CONTROL LOGIC
GROUP B CONTROL
GROUP B PORT B (8)
I/O PB7-PB0
CS

PPI 8255A

- D7 - D0 : 8 chân dữ liệu nối ghép với bus dữ liệu (có bộ đệm để ngăn cách bên trong với bên ngoài).
- /RD :tín hiệu điều khiển đọc cổng.
- /WR :tín hiệu điều khiển ghi cổng.
- /CS :tín hiệu chọn mạch.
- Chân /CS của 8255A được nối với đầu ra của một bộ giải mã địa chỉ để xác định *địa chỉ cơ sở* cho mạch.

A0, A1: sẽ chọn ra 4 thanh ghi bên trong 8255A:

- 1 thanh ghi để ghi *từ điều khiển* (CWR - control word register) cho hoạt động của 8255A,
- 3 thanh ghi ứng với các cổng A (8 bit), B (8 bit), C (gồm cổng C nửa cao - 4 bit và cổng C nửa thấp - 4 bit) để đọc/ghi dữ liệu.
- Và ta thấy, *cổng A cũng chính là địa chỉ cơ sở của 8255A.*

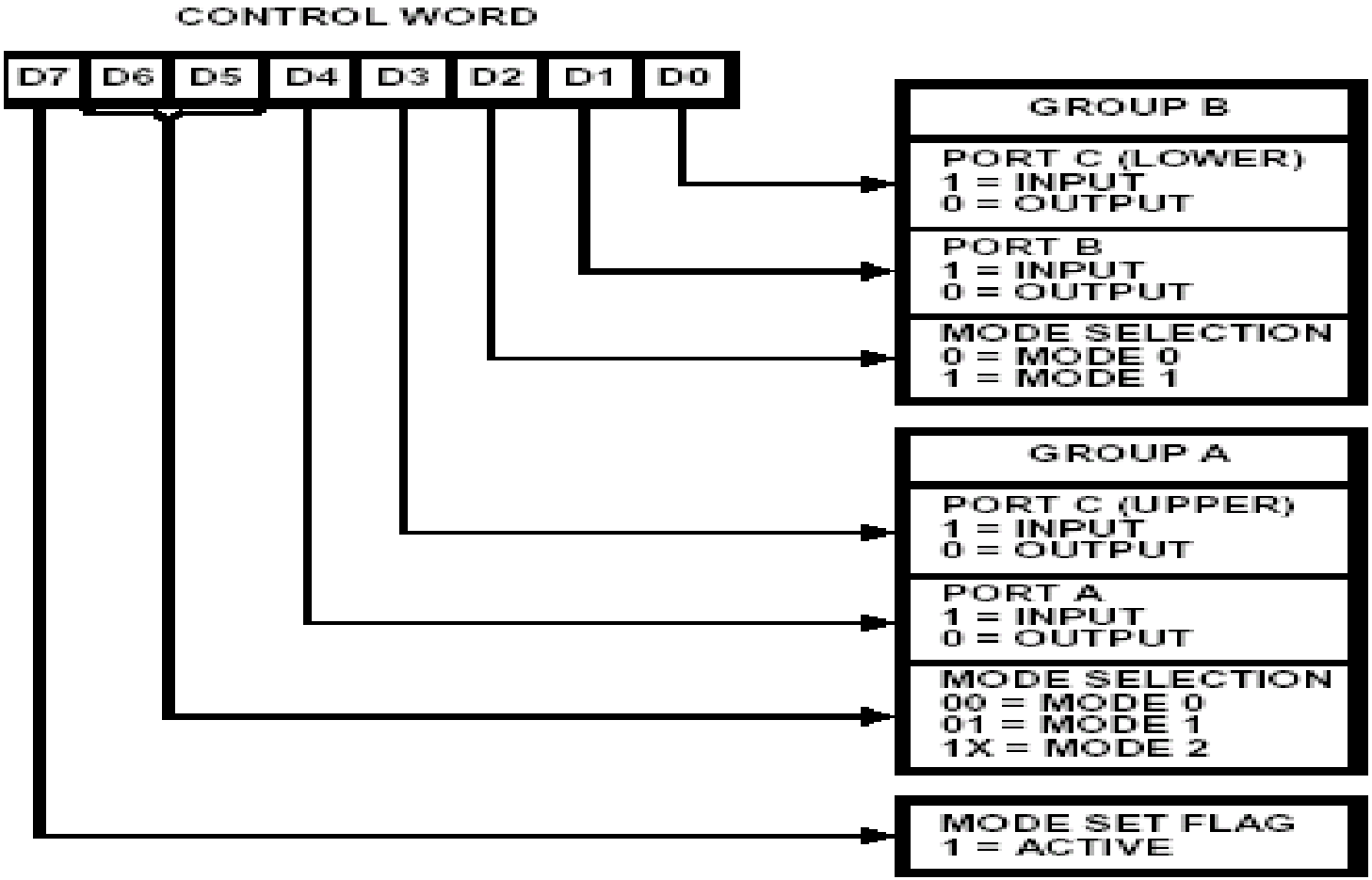
$\overline{CS}$	A_1	A_0	Chọn ra
1	x	x	Không chọn
0	0	0	Cổng A
0	0	1	Cổng B
0	1	0	Cổng C
0	1	1	CWR

PPI 8255A

- PA7 - PA0 : 8 đường vào-ra của cổng A
- PB7 - PB0 : 8 đường vào-ra của cổng B
- PC7 - PC0 : 8 đường vào-ra của cổng C

- Các chế độ làm việc :
  - Tuỳ thuộc vào nội dung của thanh ghi điều khiển mà 8255 sẽ làm việc ở các chế độ khác nhau:
  - Nếu bit D7 = 1 thì nội dung của các bit còn lại được dùng để định nghĩa cấu hình các cổng.
  - Nếu bit D7 = 0 thì nội dung của các bit còn lại được dùng để đặt/xóa các bit của cổng C.

a) Chế độ định nghĩa cấu hình:

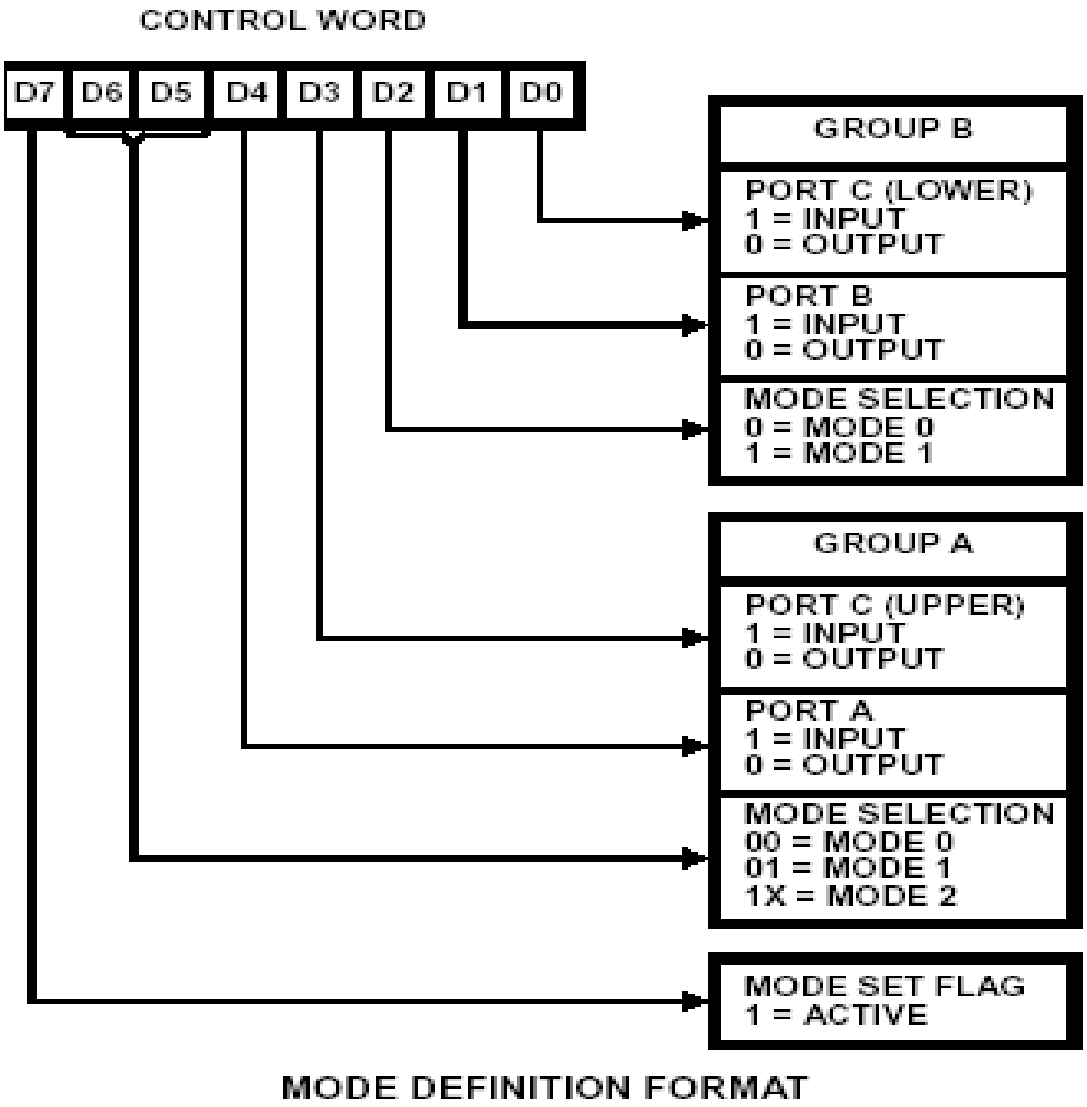


MODE DEFINITION FORMAT

- Ví dụ :
  - Tìm từ điều khiển sao cho các nhóm làm việc ở chế độ 0. Cổng A : vào ; Cổng B : ra ; Cổng C cao : vào;Cổng C thấp : ra
  - Giải:

    1 00 1 1 0 0 0

    CW=98h

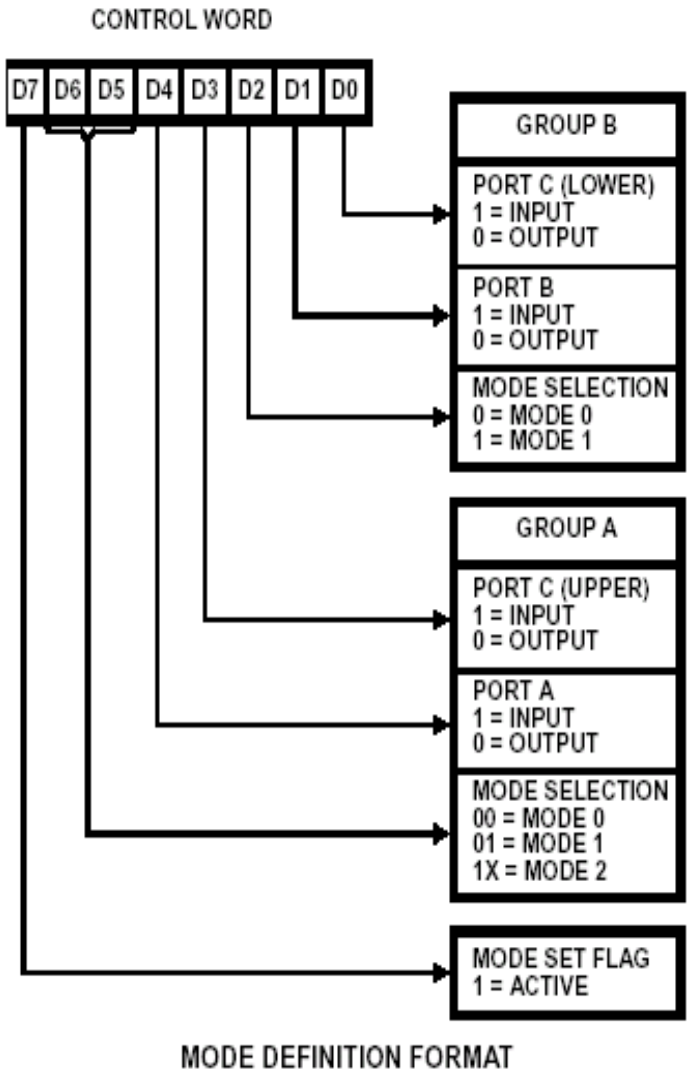


MODE DEFINITION FORMAT

- Ví dụ :
  - Tìm từ điều khiển sao cho các nhóm làm việc ở chế độ 0. Cổng A :vào ; Cổng B :vào ; Cổng C cao: vào ;Cổng C thấp : vào
  - Giải:

    1 00 1 1 0 1 1

    CW=9Bh

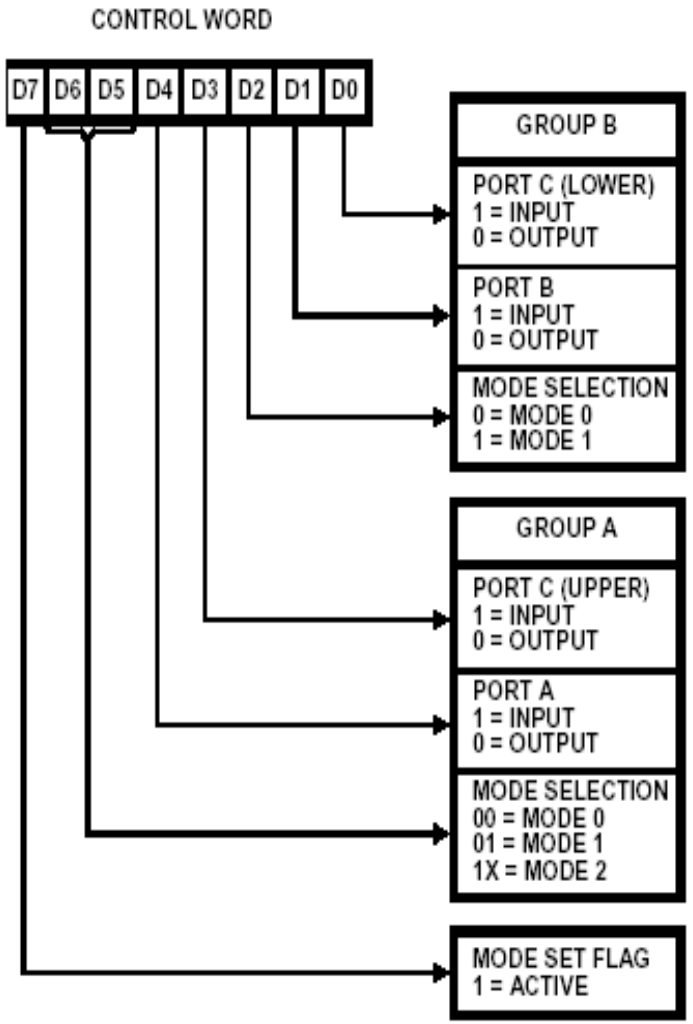


MODE DEFINITION FORMAT

- Ví dụ :
  - Tìm từ điều khiển sao cho các nhóm làm việc ở chế độ 0. Cổng A :vào ; Cổng B :vào ; Cổng C vào
  - Giải:

    1 00 1 1 0 1 1

    CW=9Bh

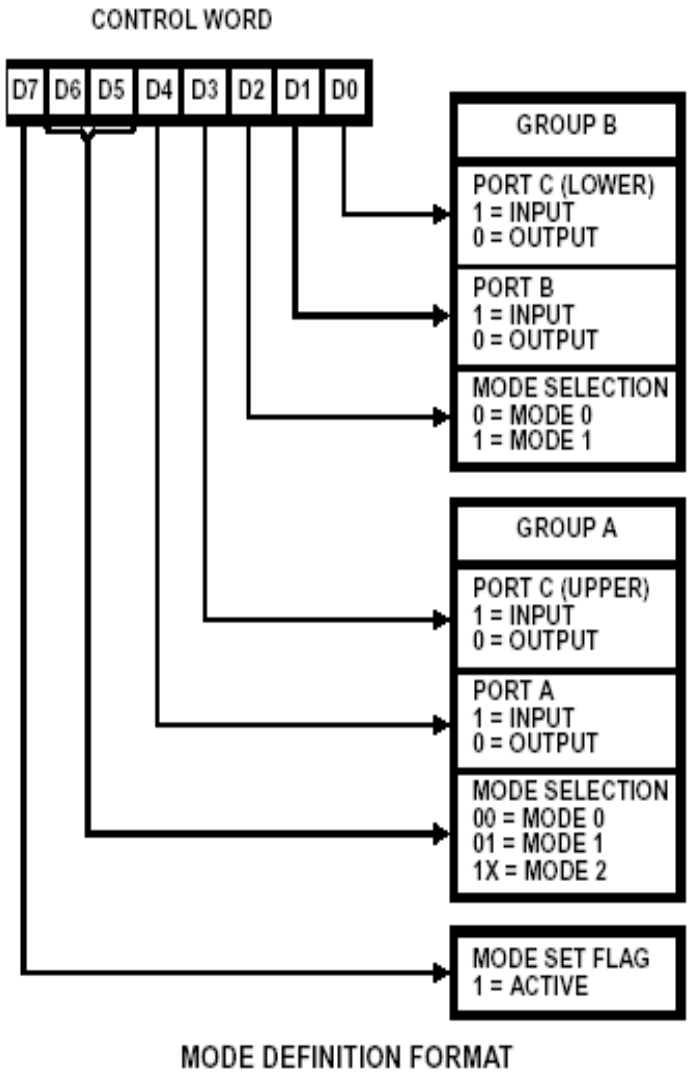


MODE DEFINITION FORMAT

- Ví dụ :
  - Tìm từ điều khiển sao cho các nhóm làm việc ở chế độ 0. Cổng A :ra ; Cổng B :ra ; Cổng C cao: vào ;Cổng C thấp : vào
  - Giải:

    1 00 0 1 0 0 1

    CW=89h



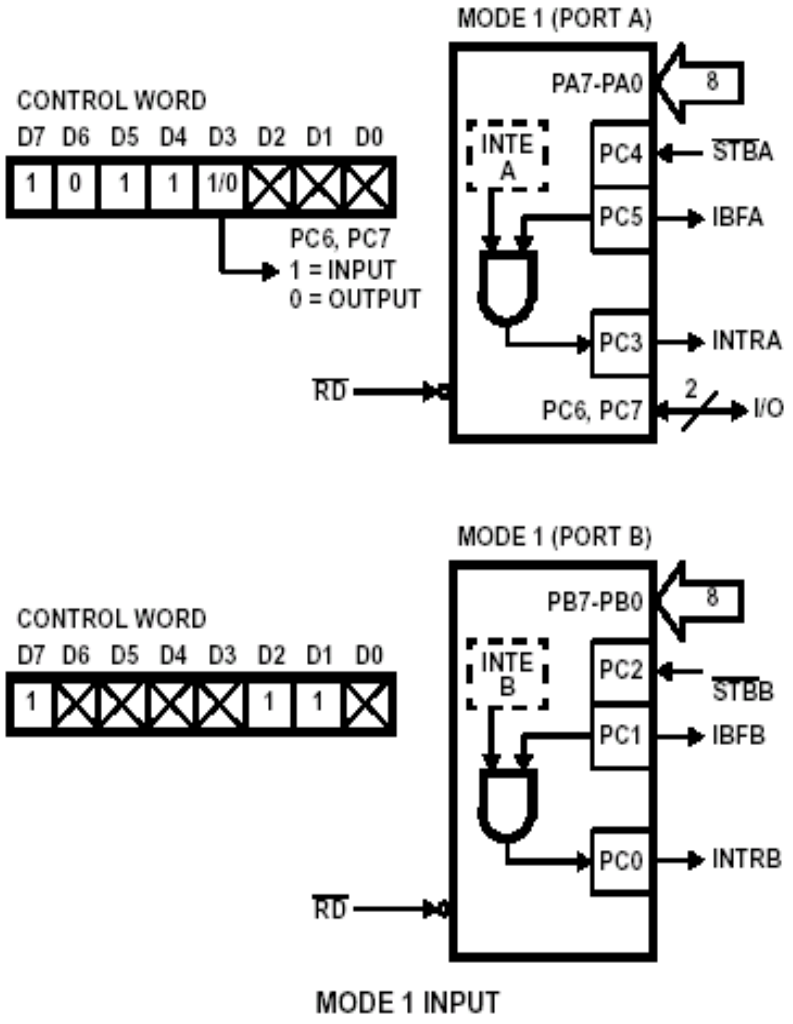
MODE DEFINITION FORMAT

- 8255A có 4 chế độ làm việc:
  - **Chế độ 0: Vào/ra cơ sở(vào/ra đơn giản)**:
    - ✓Các cổng A,B,C_H, C_L đều có thể được sử dụng làm các cổng vào hoặc ra.
    - ✓Có 16 khả năng cấu hình các cổng làm vào hoặc ra.
  - **Chế độ 1: Vào/ra có xung cho phép (Strobed Input/Output) :**
    - ✓A,B đều có thể được sử dụng làm các cổng vào hoặc ra
    - ✓C_H, C_L được dùng làm các tín hiệu bắt tay (handshaking)khi A, B trao đổi dữ liệu

- **Chế độ 2: Vào ra 2 chiều:**
  - Chỉ có cổng A có thể được sử dụng làm cổng vào hoặc ra.
  - Các tín hiệu bắt tay do các bit của cổng C đảm nhiệm.
  - PB chỉ làm việc như trong chế độ 0 hoặc 1.
- **Chế độ Lập/ xóa các bit PCi.**

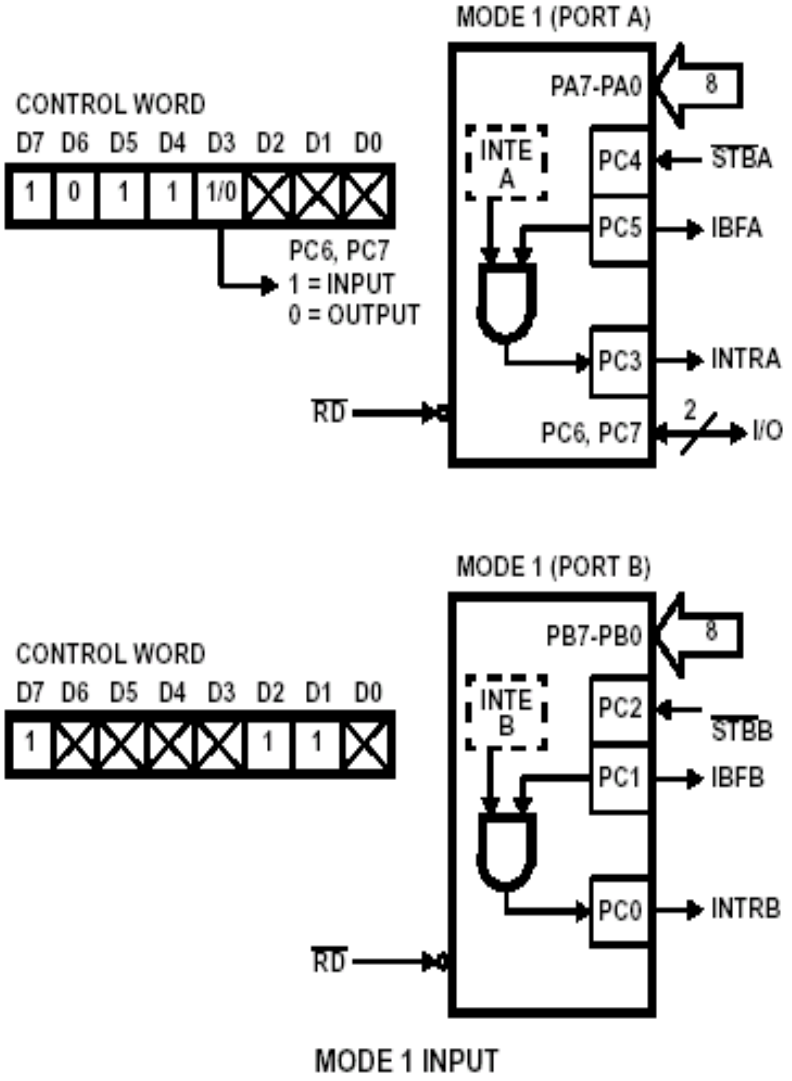
Chế độ 1: Vào/ra với xung cho phép

- Vào dữ liệu trong chế độ 1 (/RD= 0):
  - /STBA,/STBB(Strobe): là tín hiệu đi vào 2 bit của cổng C: PC4(), PC2().
    - ✓Tích cực ở mức 0 .Khi dữ liệu sẵn sàng để được đọc vào bởi PA,PB, thiết bị ngoại vi truyền cho 8255 biết.



Chế độ 1: Vào/ra với xung cho phép

- IBFA, IBFB (Input Buffer Full):
  - Đệm vào cổng A hoặc B đầy. Sau khi 8255 nhận được dữ liệu từ một thiết bị ngoại vi nào đó vào các cổng A, B, nó báo lại thiết bị ngoại vi bằng tín hiệu này cho biết đã nhận hết dữ liệu.

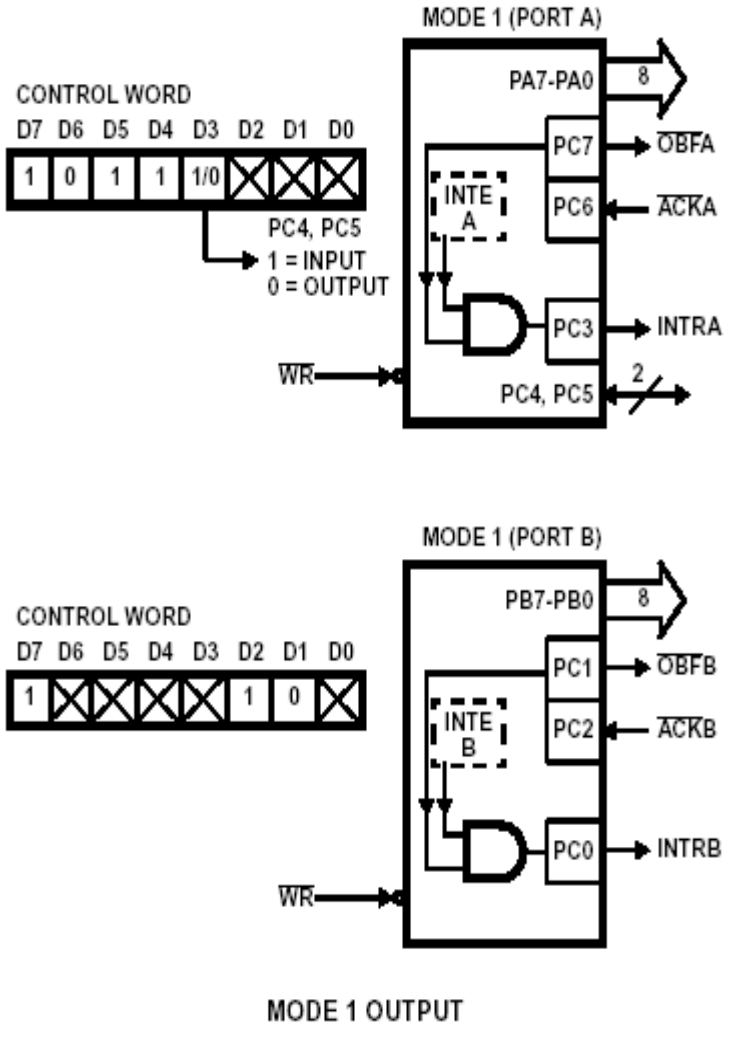


Chế độ 1: Vào/ra với xung cho phép

- Ra dữ liệu trong chế độ 1 (/WR = 0):
  - **/ACKA,/ACKB**:

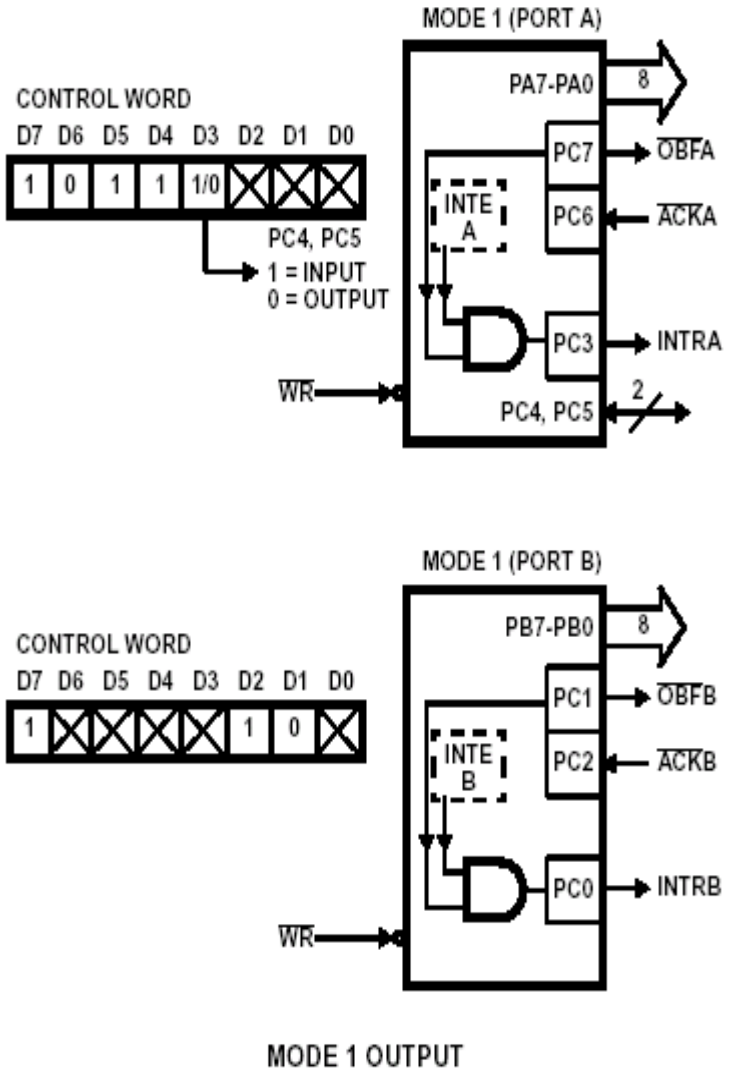
(Acknowlege): tbnv báo đã nhận được dữ liệu,

Hai tín hiệu này được truyền tới cổng A, B tại bit 6 và bit 2 của cổng C.



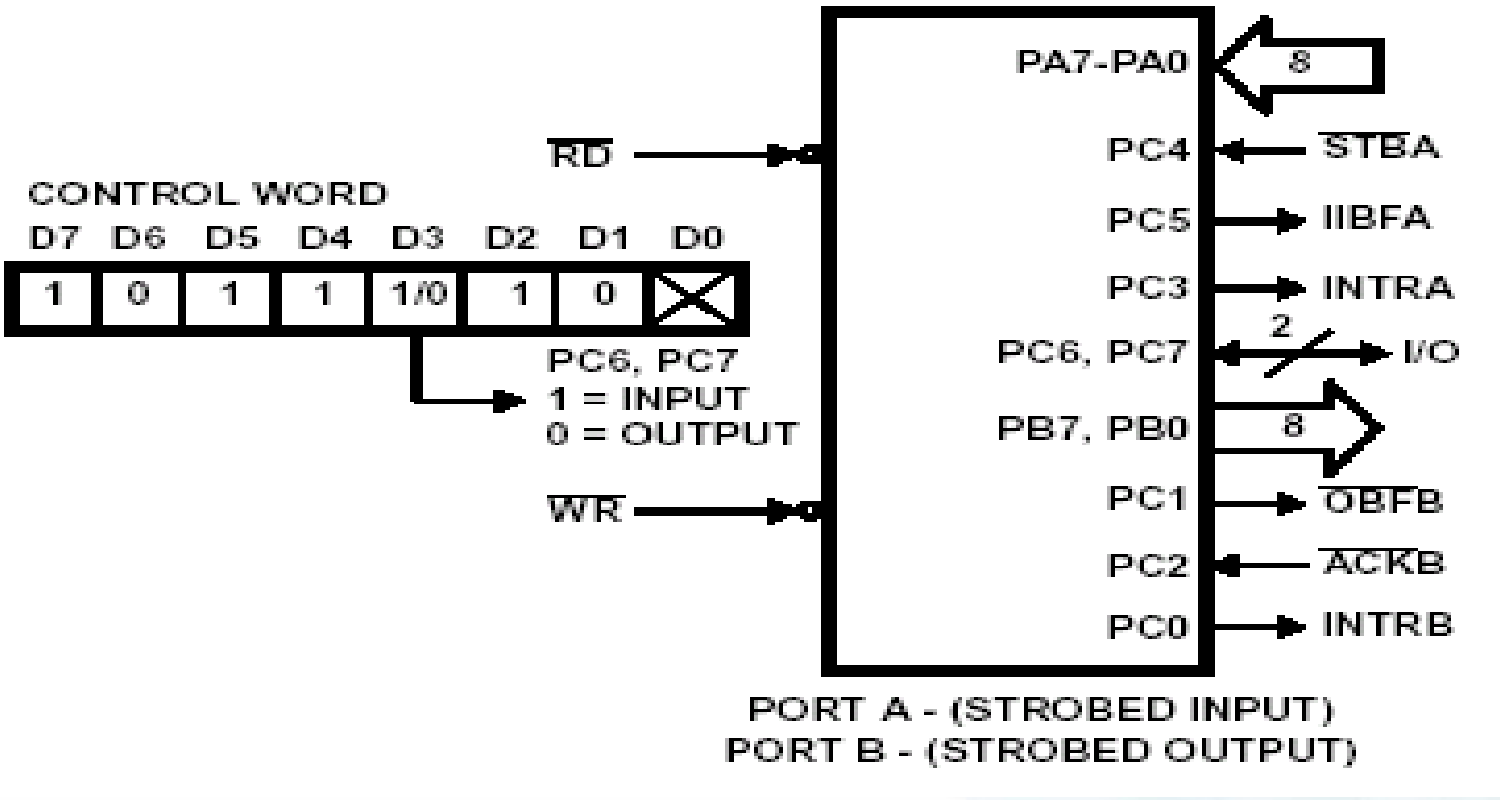
Chế độ 1: Vào/ra với xung cho phép

- /OBFA,/OBFB (Output Buffer Full): đệm vào cổng A hoặc B đầy.
  - Sau khi 8255 nhận được dữ liệu từ bộ vi xử lý và sẵn sàng cho ra các cổng A, B, nó báo với thiết bị ngoại vi bằng tín hiệu này cho biết đã sẵn sàng truyền dữ liệu.



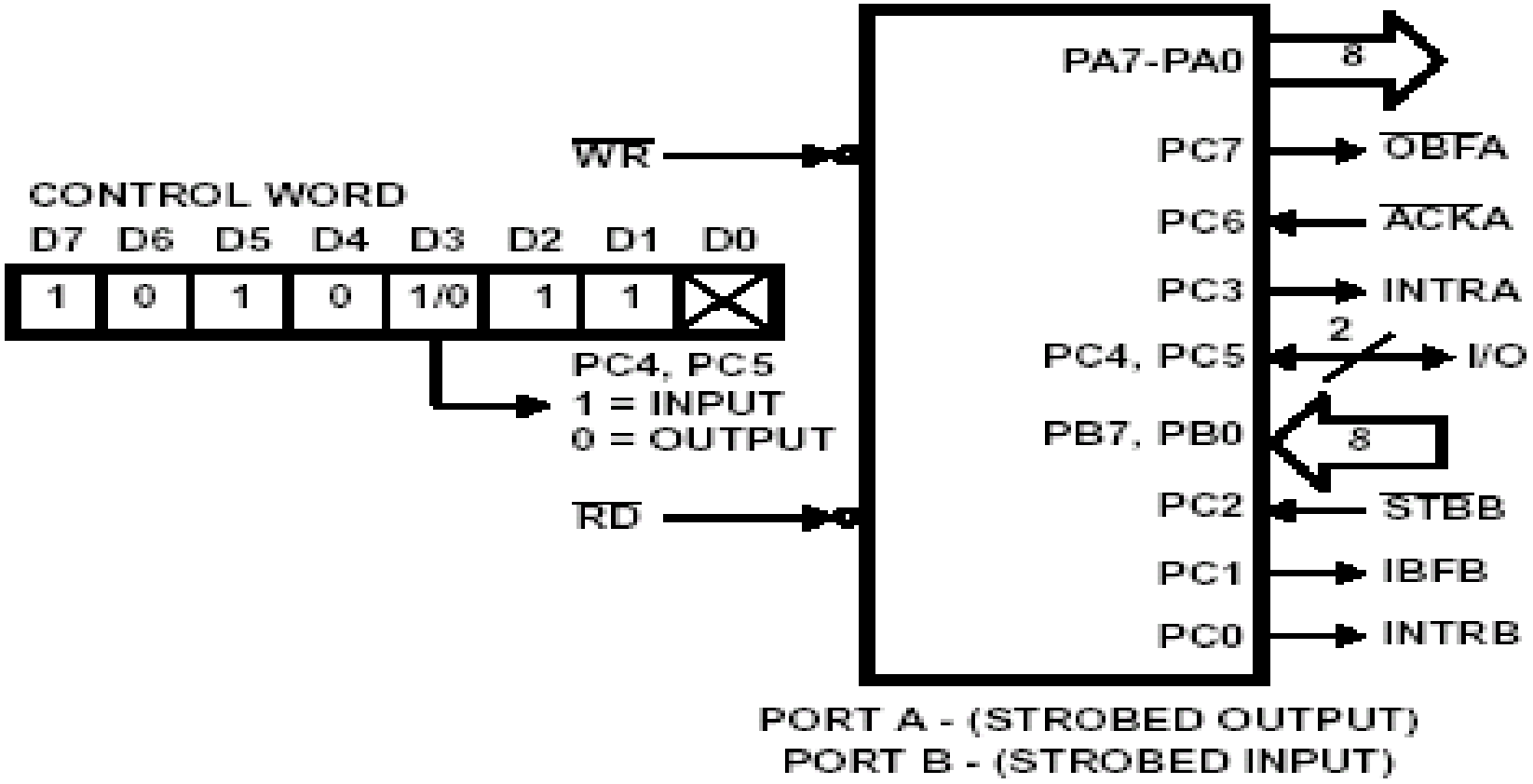
Chế độ 1: Vào/ra với xung cho phép

- Trong chế độ 1: PA, PB có thể được cấu hình để thành các cổng vào/ra riêng biệt*



Chế độ 1: Vào/ra với xung cho phép

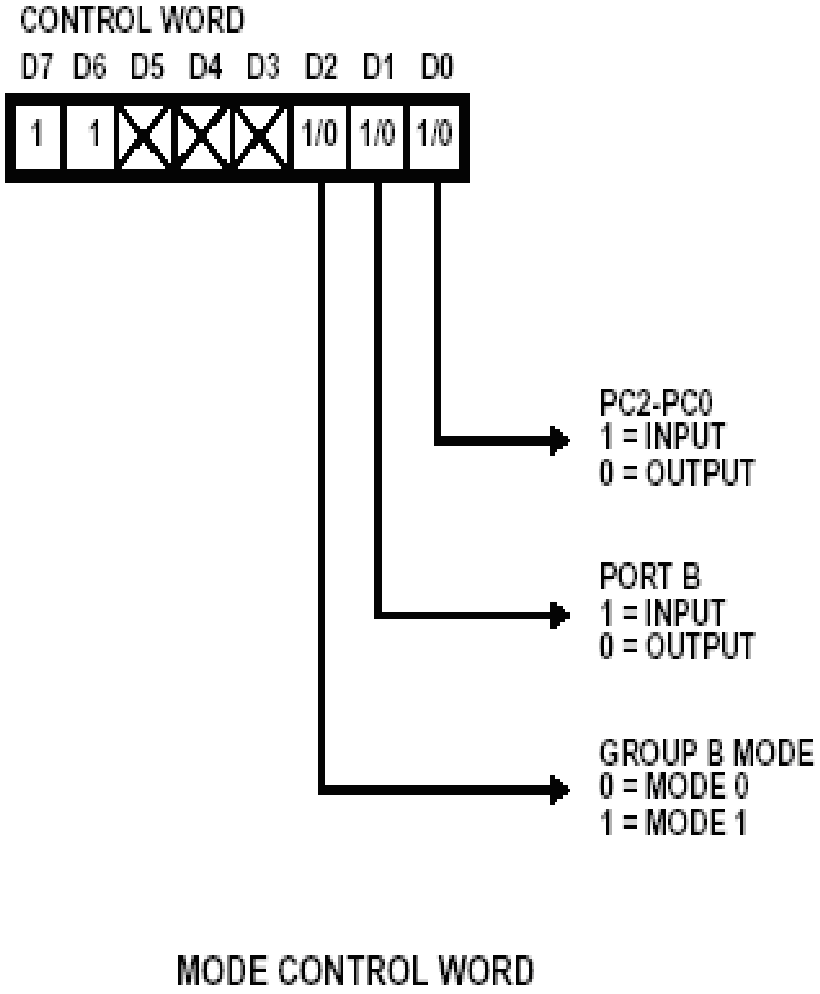
- A –ra, B-vào



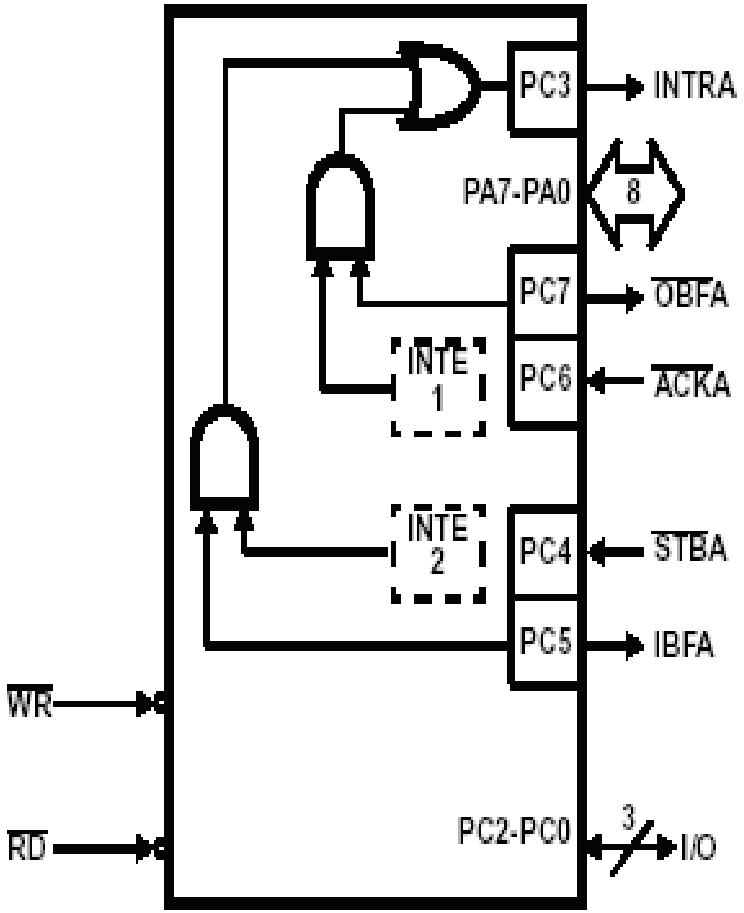
Chế độ 2: Chế độ bus hai chiều

- **Chế độ 2: Chế độ bus hai chiều (Bi-directional Bus)** :
  - Trong chế độ này chỉ riêng cổng A được định nghĩa để làm việc như một cổng 2 chiều có các tín hiệu móc nối do một số bit của cổng C đảm nhiệm
  - Cổng B thì có thể làm việc ở chế độ 1 hoặc 0 tùy theo các bit điều khiển trong CWR. Các chân tín hiệu còn lại của cổng C có thể được định nghĩa để làm việc như các chân vào hoặc ra, hoặc phục vụ cho cổng B.

Chế độ 2: Chế độ bus hai chiều

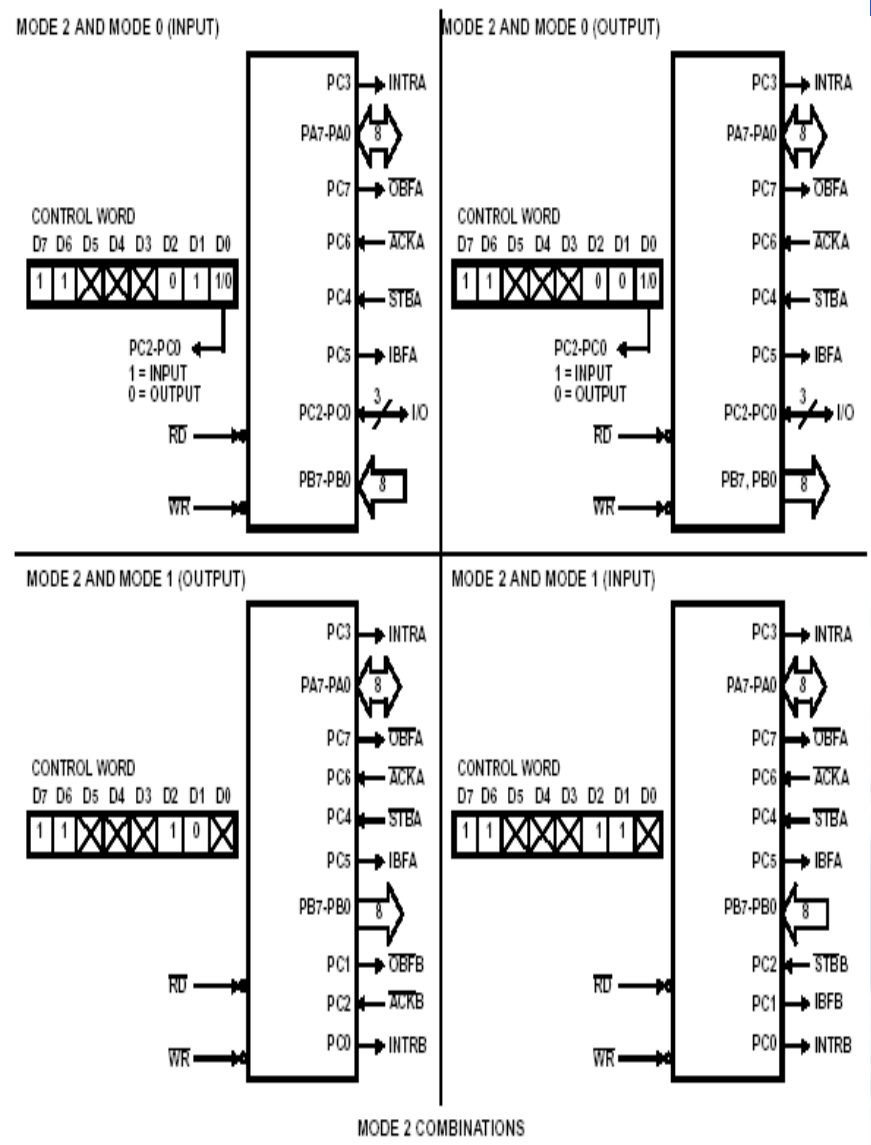


MODE CONTROL WORD



MODE 2

Chế độ 2: Chế độ bus hai chiều

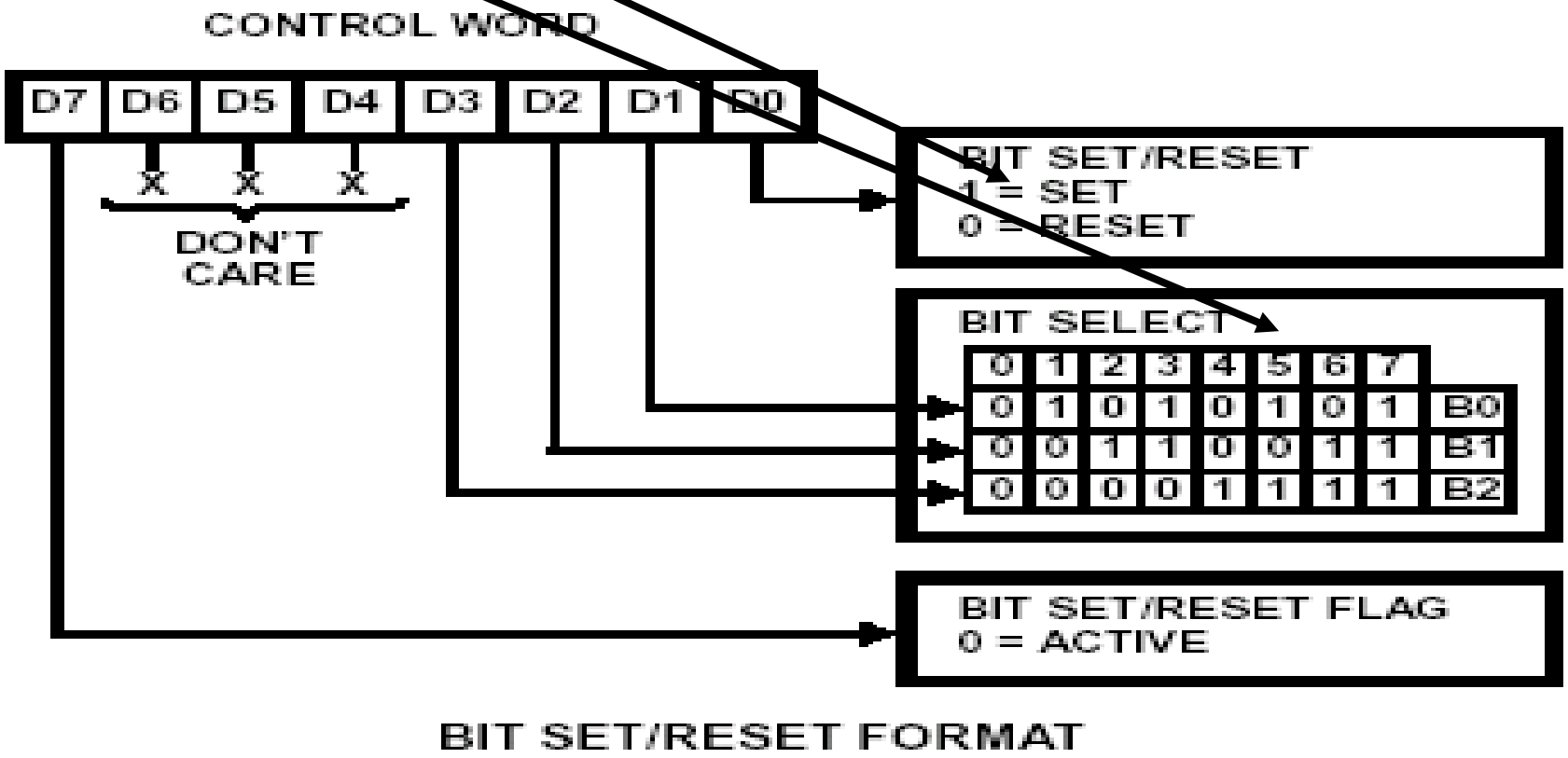


MODE 2 COMBINATIONS

Chế độ đặt/xóa các bit cổng C

- Tìm từ điều khiển để lập bit PC5 = 1:

  CW=**0** 000 101 1= 0Bh

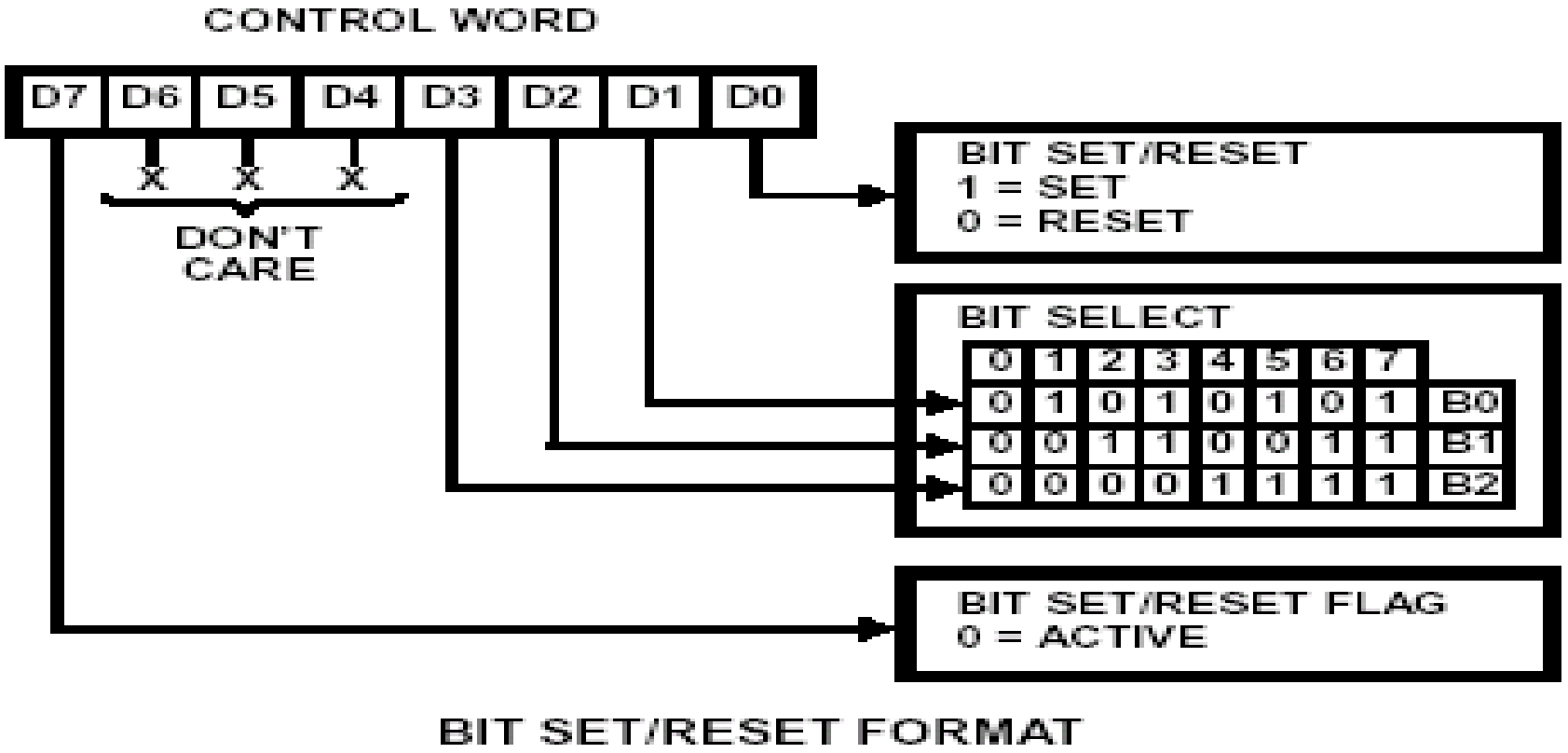


BIT SET/RESET FORMAT

Chế độ đặt/xóa các bit cổng C

- Tìm từ điều khiển để lập bit PC2 = 1 ?

  CW=**0** 000 010 1= 05h

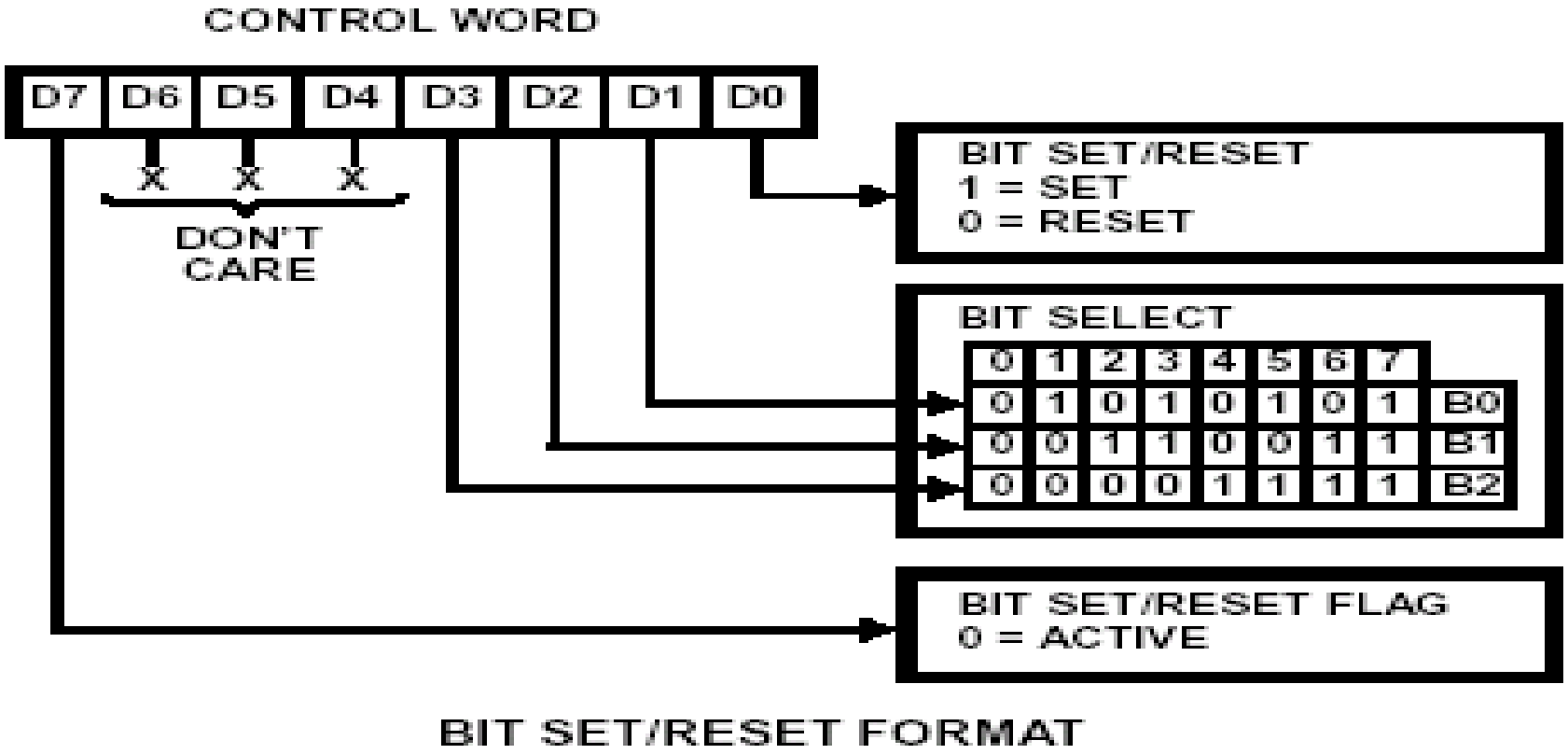


BIT SET/RESET FORMAT

Chế độ đặt/xóa các bit cổng C

- Tìm từ điều khiển để lập bit PC7 = 0 ?

  CW=**0** 000 111 0= 0Eh

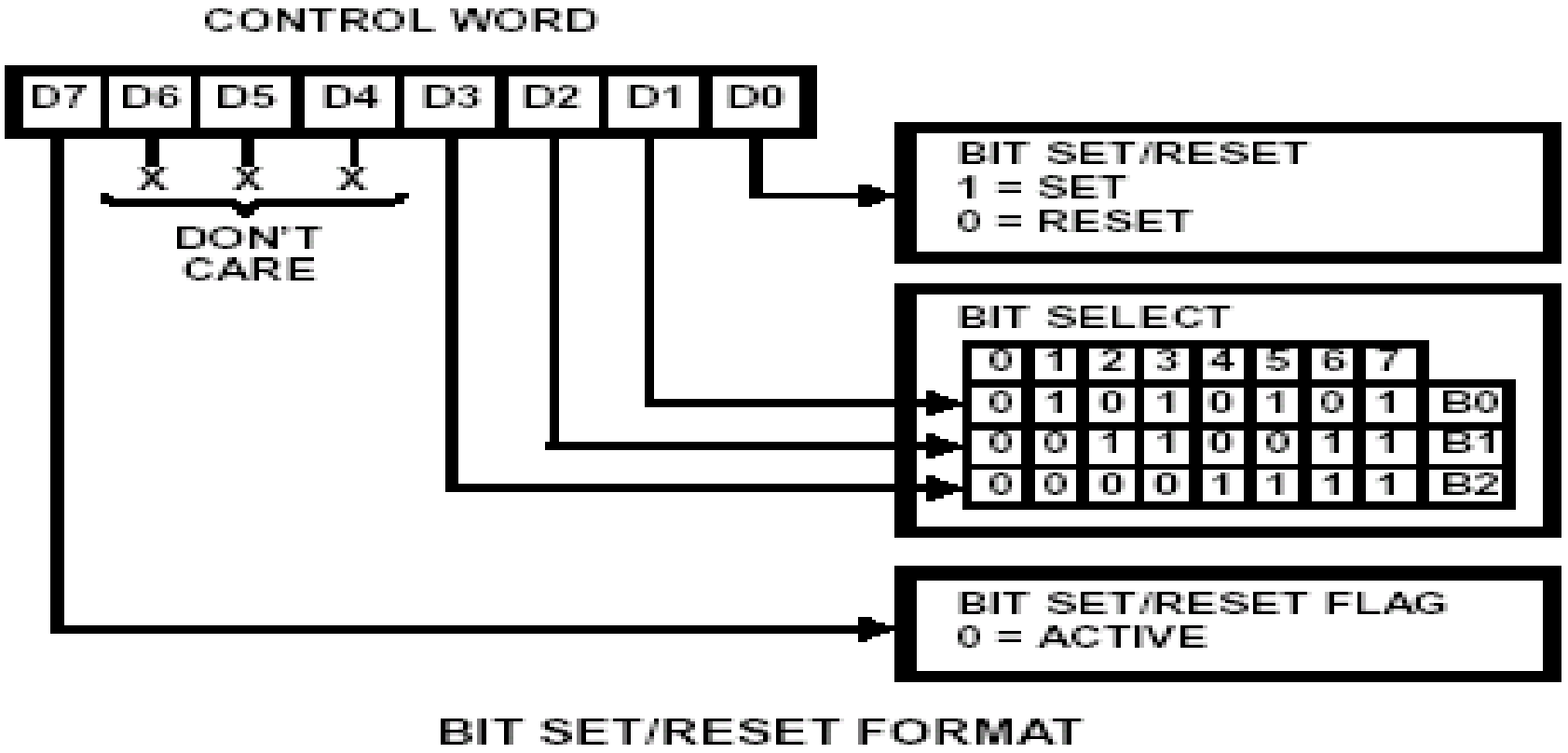


BIT SET/RESET FORMAT

Chế độ đặt/xóa các bit cổng C

- Tìm từ điều khiển để lập bit PC3 = 0 ?

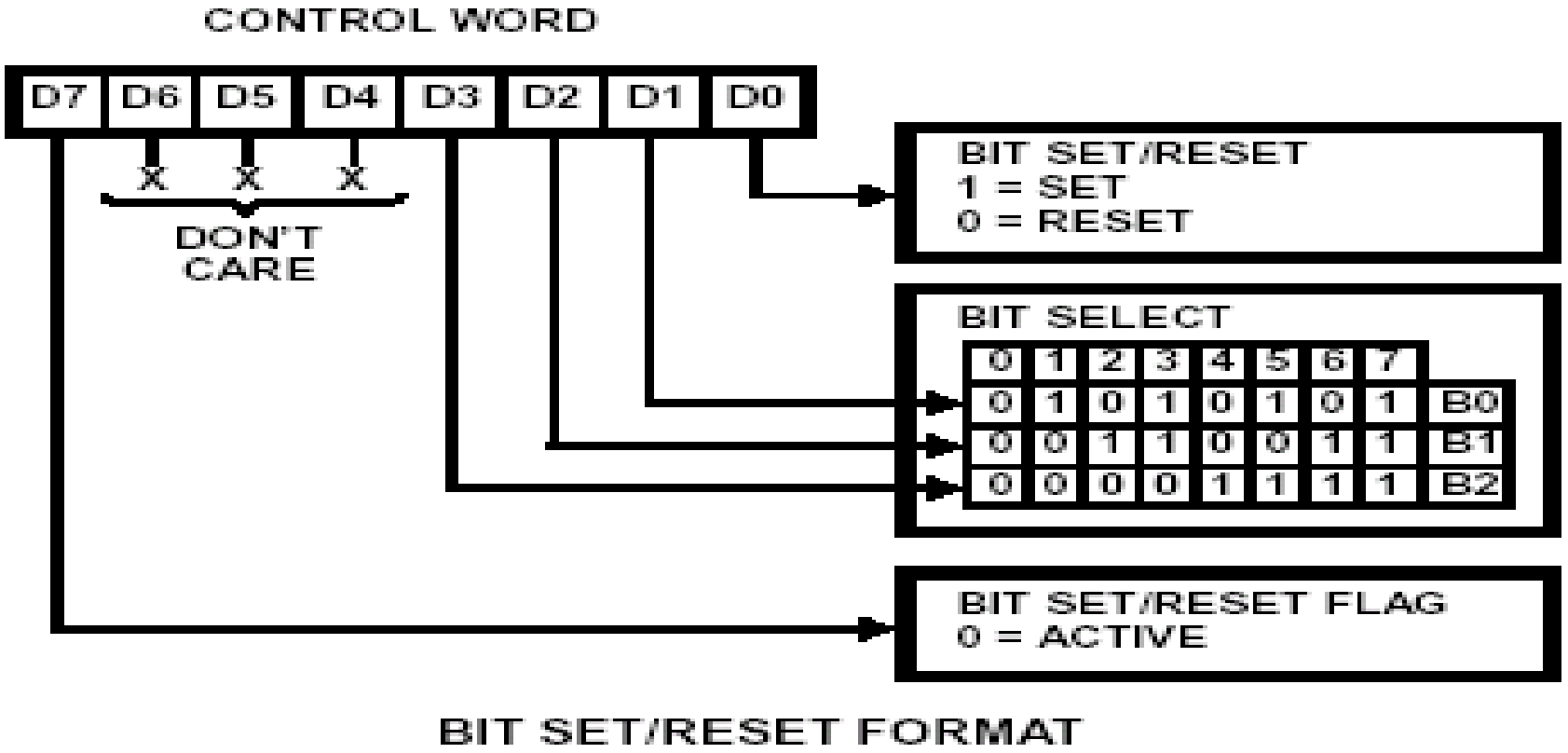
  CW=**0** 000 011 0= 06h



Chế độ đặt/xóa các bit cổng C

- Tìm từ điều khiển để lập bit PC4 = 1 ?

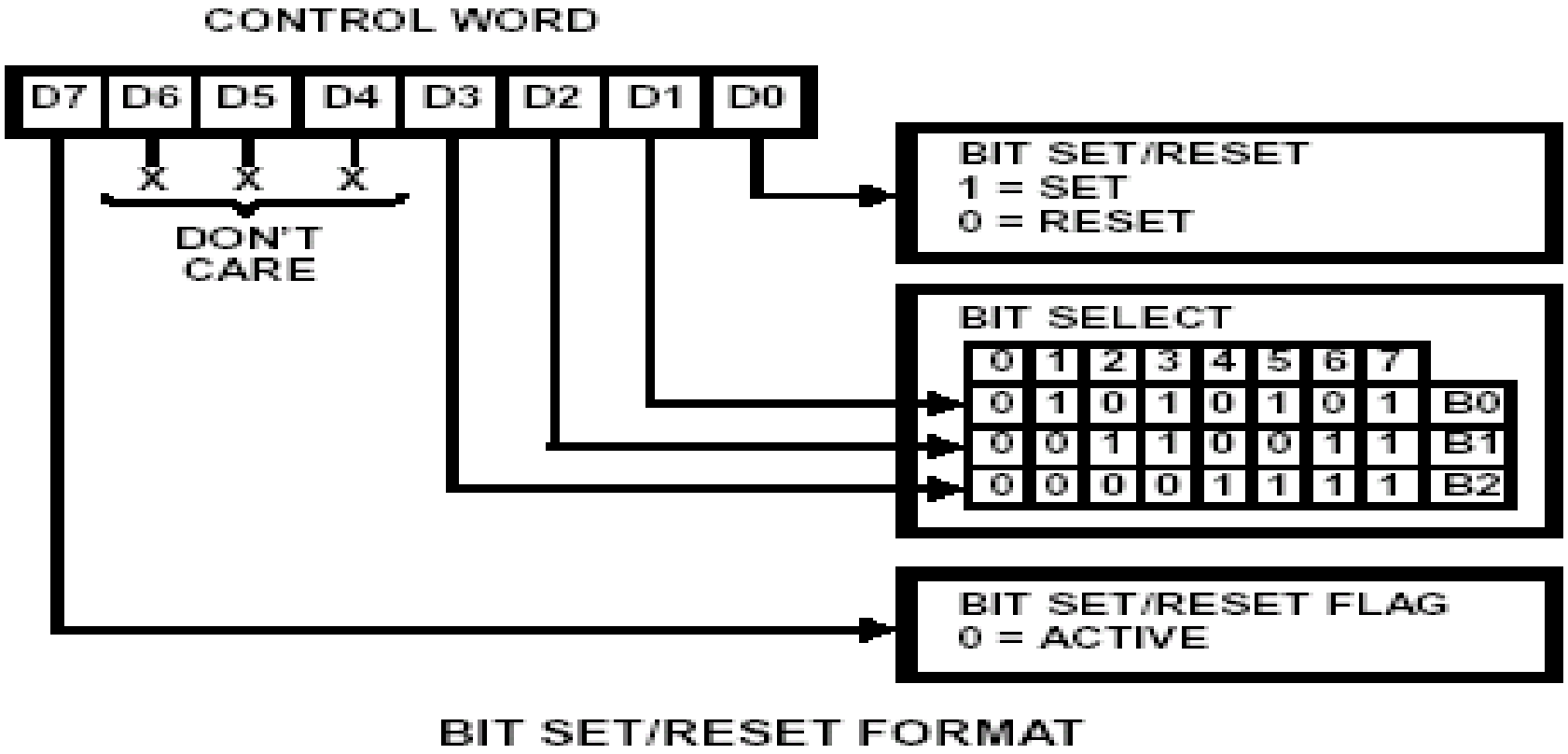
  CW=**0** 000 100 1= 09h



Chế độ đặt/xóa các bit cổng C

- Tìm từ điều khiển để lập bit PC1 = 1 ?

  CW=**0** 000 001 1= 0Eh



BIT SET/RESET FORMAT

Nối ghép và lập trình cho 8255A

- Chỉ xét với trường hợp nối ghép 8255A với 8088 theo kiểu vào-ra cách biệt với số bit địa chỉ là 8 bit

Ví dụ 1:

a) Nối ghép 8255A với hệ 8088 như sau :

Cổng A có địa chỉ là 50h

Cổng B có địa chỉ là 51h

Cổng C có địa chỉ là 52h

Thanh ghi điều khiển có địa chỉ là 53h

b) Lập trình để định nghĩa *chế độ 0* cho 8255A với cấu hình các cổng như sau:

Cổng A: Vào

Cổng B: Ra

Cổng C: Ra

c) Lập trình nhận 100 dữ liệu từ cổng A và lần lượt đưa ra cổng B và C.

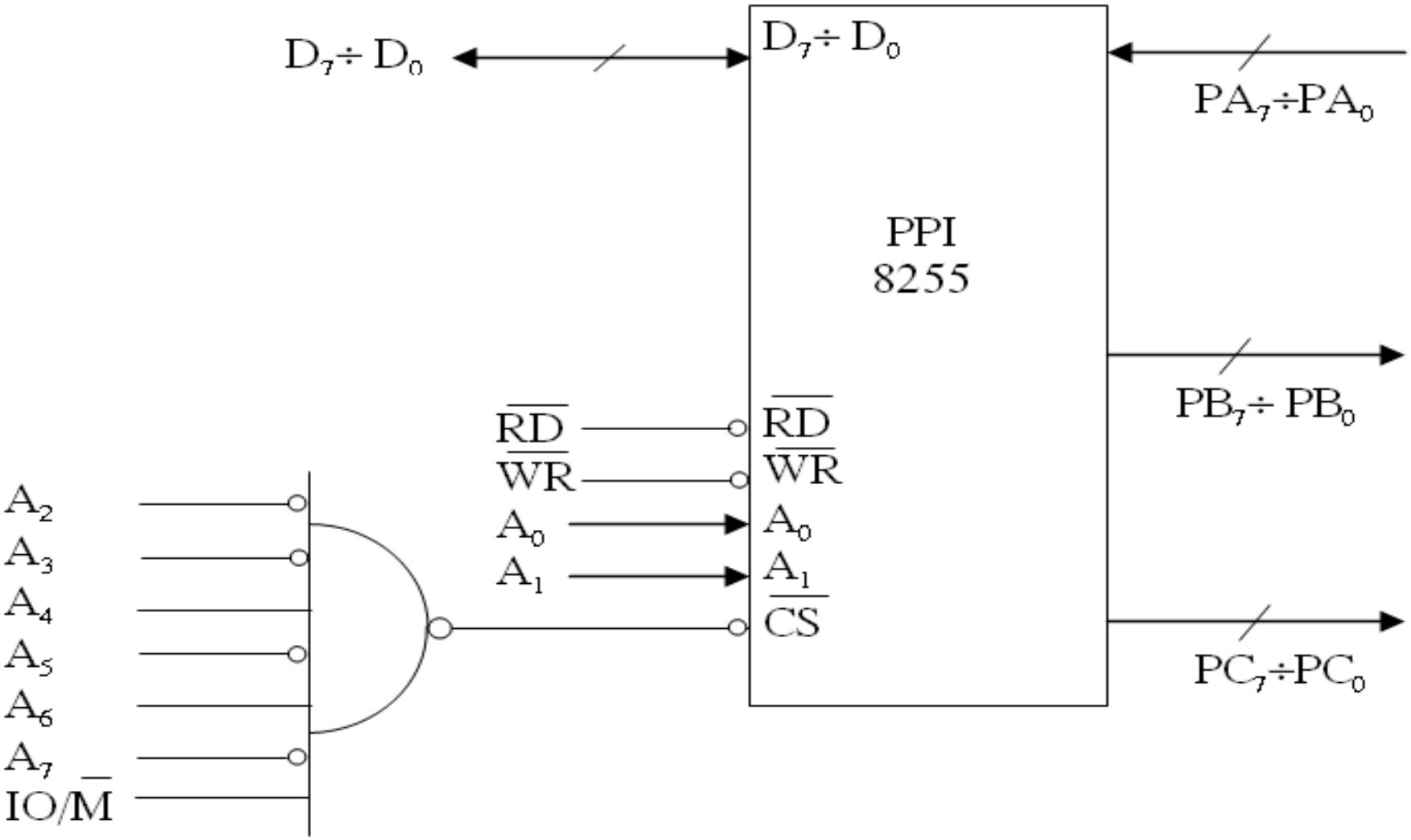
Lời giải: a)Giải mã địa chỉ:

Cổng A 50h; Cổng B 51h; Cổng C 52h

Thanh ghi điều khiển có địa chỉ là 53h

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
50h	0	1	0	1	0	0	0	0
51h	0	1	0	1	0	0	0	1
52h	0	1	0	1	0	0	1	0
53h	0	1	0	1	0	0	1	1
	Nối với $\overline{CS}$						Nối với A_1, A_0	

Nối ghép và lập trình cho 8255A



Nối ghép và lập trình cho 8255A

b) Lập trình để định nghĩa chế độ 0 cho 8255A với cấu hình các cổng như sau:

Cổng A: Vào Cổng B: Ra

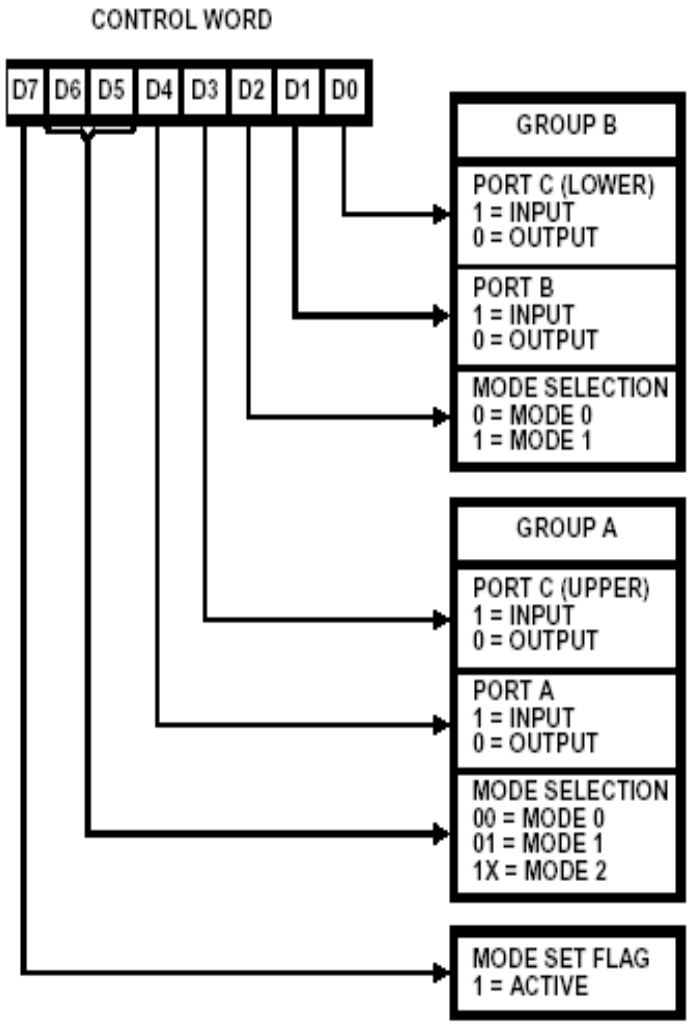
Cổng C: Ra

Tìm từ điều khiển:

1 00 1 0 0 0 0= 90h

Lập trình:

```
MOV AL, 90h
OUT 53h, AL
```



MODE DEFINITION FORMAT

C) Lập trình nhận 100 dữ liệu từ cổng A và lần lượt đưa ra cổng B và C.

```
MOV CX, 100 ; nhận 100 dữ liệu
LAP :
    IN  AL, 50H ; từ cổng A
    OUT 51H, AL ; đưa ra cổng B
    OUT 52H, AL ; và cổng C
    LOOP LAP
```

Nối ghép và lập trình cho 8255A

Ví dụ 2:

Cho mạch nối ghép giữa hệ VXL 8088 với PPI 8255 như hình vẽ.

a) Tìm địa chỉ của các cổng.

b) Tìm từ điều khiển sao cho cổng:

A : cổng ra

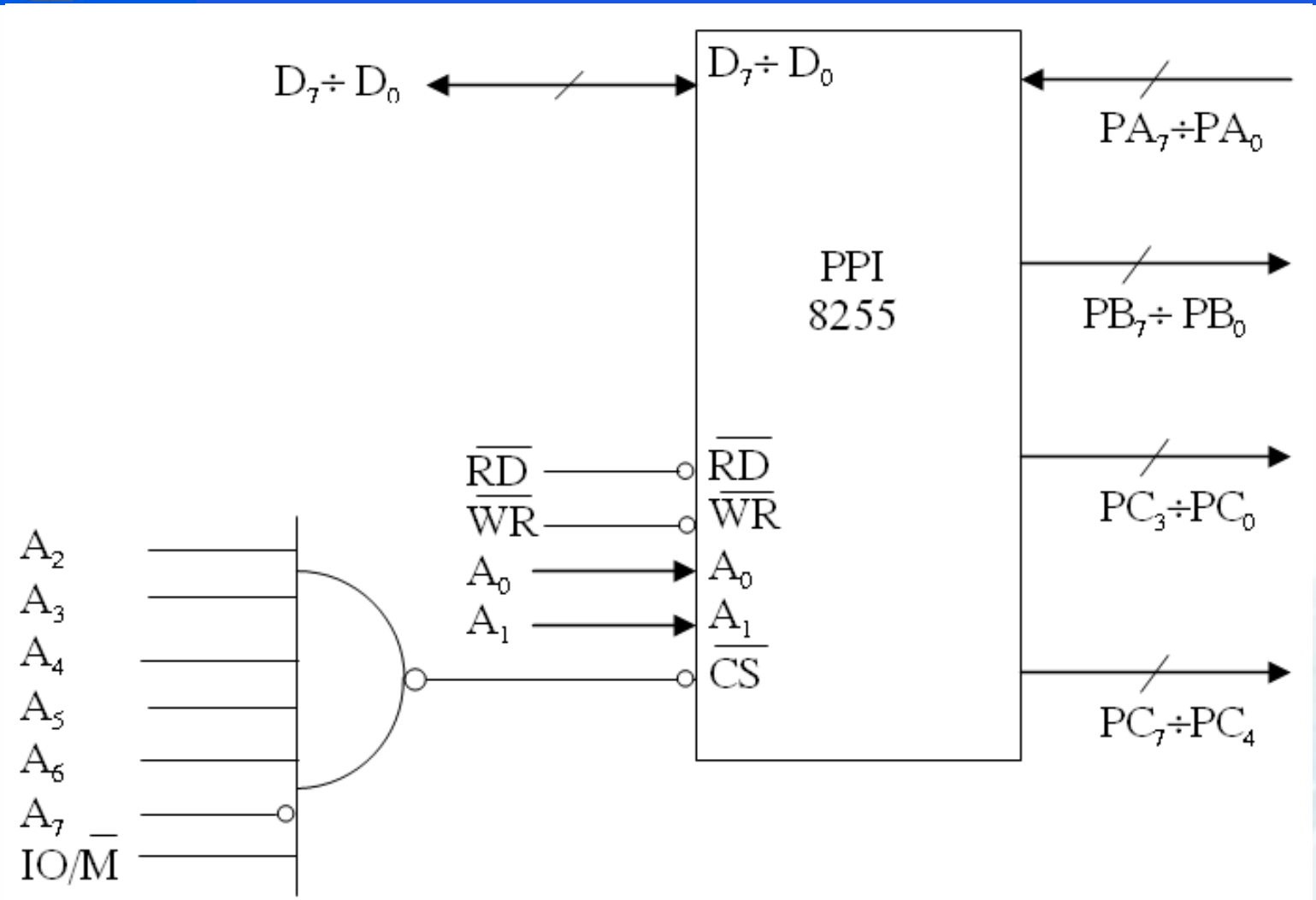
B : cổng vào

Cthấp : cổng vào

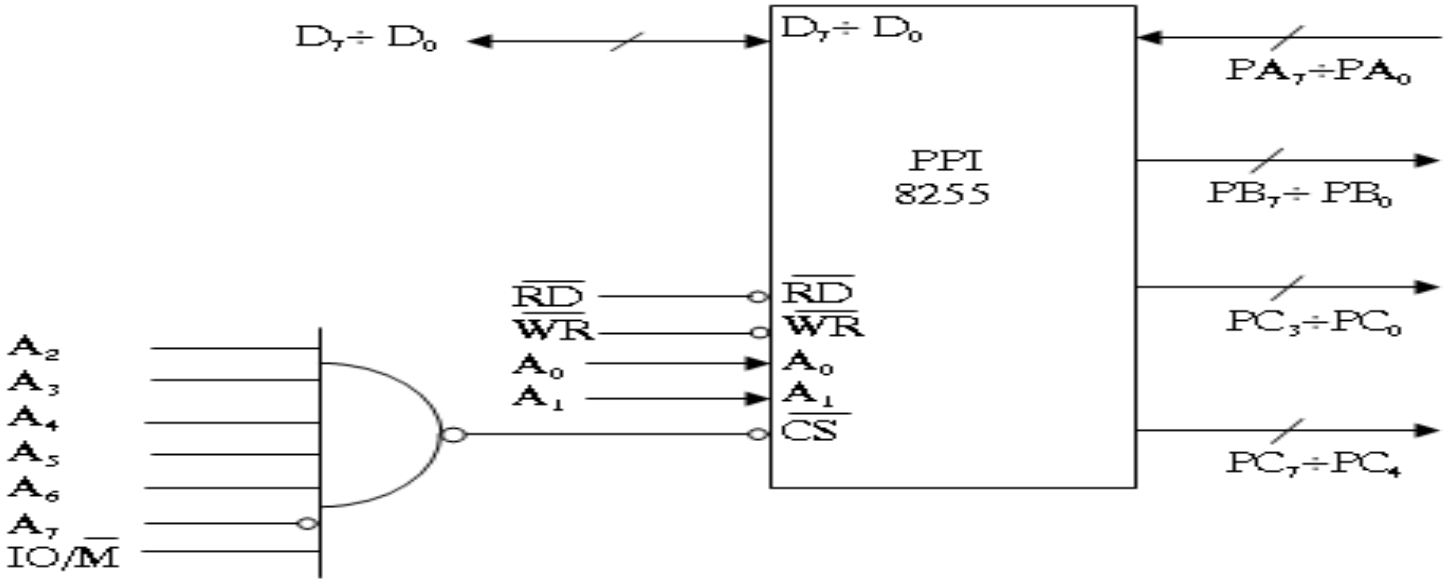
Ccao : cổng ra

c) Lập trình cho 8255 để nhận dữ liệu từ cổng B và gửi ra cổng A sau đó nhận dữ liệu từ cổng Cthấp và gửi ra cổng C cao.

Nối ghép và lập trình cho 8255A



Nối ghép và lập trình cho 8255A



Giải:

a) Tìm địa chỉ các cổng:

	A7	A6	A5	A4	A3	A2	A1	A0	
7Ch	0	1	1	1	1	1	0	0	Cổng A
7Dh	0	1	1	1	1	1	0	1	Cổng B
7Eh	0	1	1	1	1	1	1	0	Cổng C
7Fh	0	1	1	1	1	1	1	1	Thanh ghi điều khiển

Nối ghép và lập trình cho 8255A

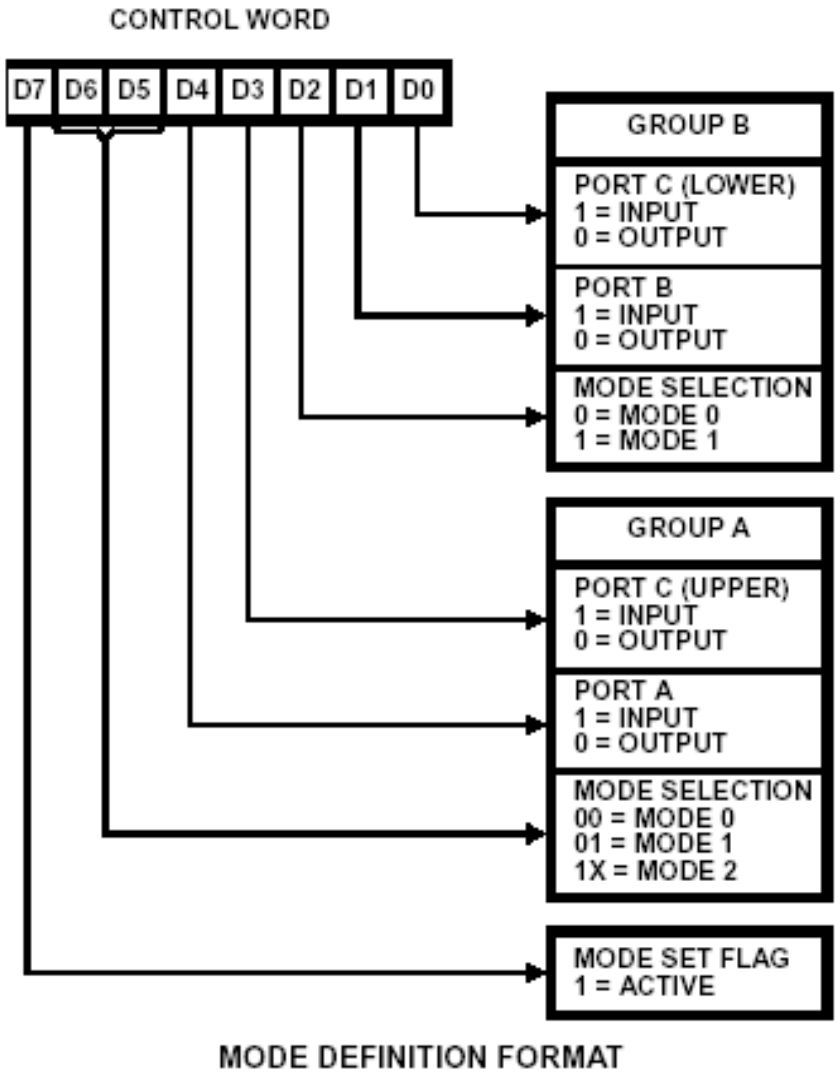
b)A : cổng ra

B : cổng vào

Cthấp : cổng vào

Ccao : cổng ra

=>từ điều khiển 10000011



MODE DEFINITION FORMAT

Nối ghép và lập trình cho 8255A

c)Lập trình cho 8255 để nhận dữ liệu từ cổng B và gửi ra cổng A sau đó nhận dữ liệu từ cổng Cthấp và gửi ra cổng Ccao.

Lập trình:

```
MOV AL, 83H   ; lấy từ điều khiển
OUT 7FH, AL   ; nạp cho thanh ghi điều khiển
IN  AL, 7DH   ; nhận dữ liệu từ cổng B
OUT 7CH, AL   ; đưa ra cổng A
IN  AL, 7EH   ; nhận dữ liệu từ cổng C thấp
MOV CL, 4          ; quay
ROL AL, CL    ; trái AL 4 bit
OUT 7EH, AL   ; đưa ra cổng C cao
```

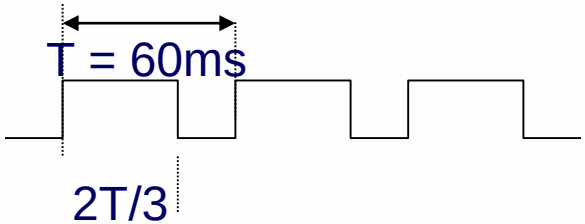
Bài tập 3:

Cho mạch nối ghép giữa hệ VXL 8088 với PPI 8255A như sau:

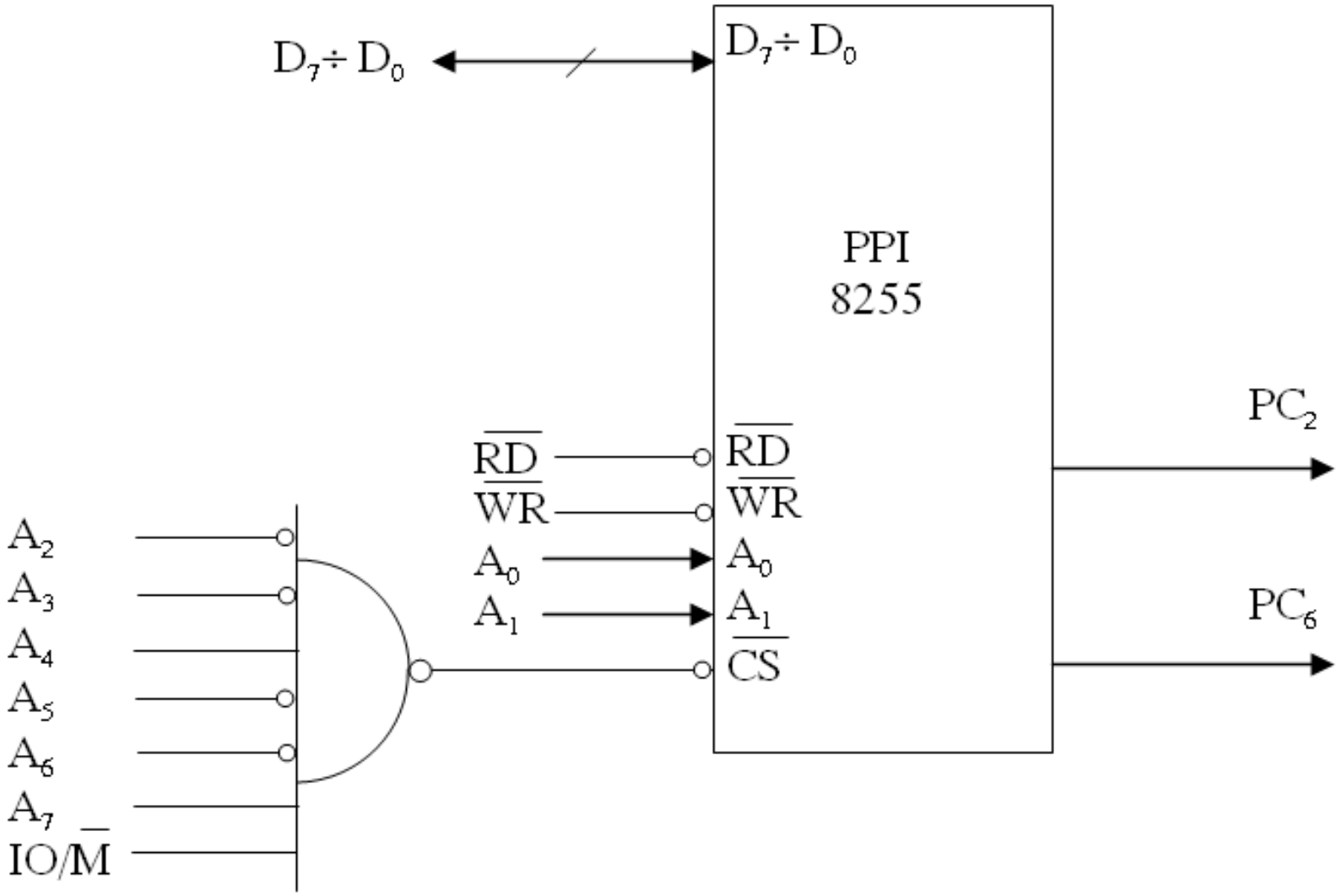
a) Thiết lập PC2 = 1

b) Tạo xung có dạng sau đây ở PC6 ;

Giả sử có ctc tạo trễ 20ms DELAY20



Nối ghép và lập trình cho 8255A



Giải:

a) Thiết lập PC2 = 1:

Địa chỉ của thanh ghi điều khiển:

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
1	0	0	1	0	0	1	1	= 93h

khiển để PC2 = 1

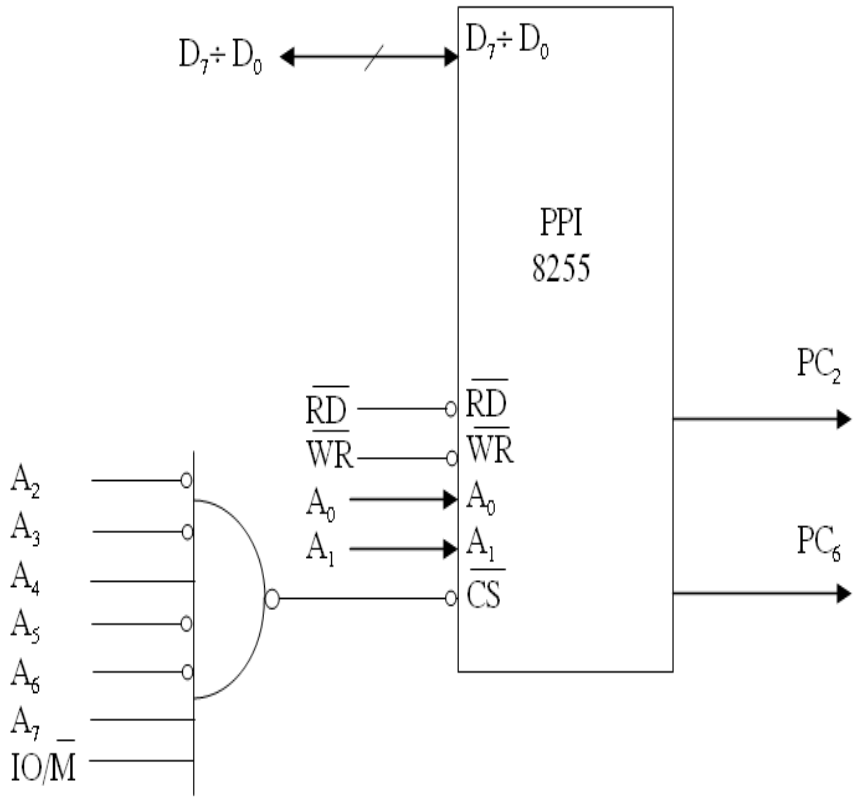
0	0	0	0	0	1	0	1	= 05h

Từ điều khiển để PC6 = 1

0	0	0	0	1	1	0	1	= 0Dh

Từ điều khiển để PC6 = 0

0	0	0	0	1	1	0	0	= 0Ch



b) Lập trình:để tạo xung

```
LAP:
   MOV AL, 0DH       ; nạp từ điều khiển
OUT    93H, AL       ; để PC6 = 1
CALL   DELAY20       ; trễ 20ms
CALL   DELAY20       ; trễ 20ms
MOV    AL, 0CH       ; nạp từ điều khiển
OUT    93H, AL       ; để PC6 = 0
CALL   DELAY20       ; trễ 20ms
JMP    LAP           ; lặp lại
```

Nội dung chương 5

5.1. Định địa chỉ cổng vào-ra

5.2. Giải mã địa chỉ cho cổng vào-ra

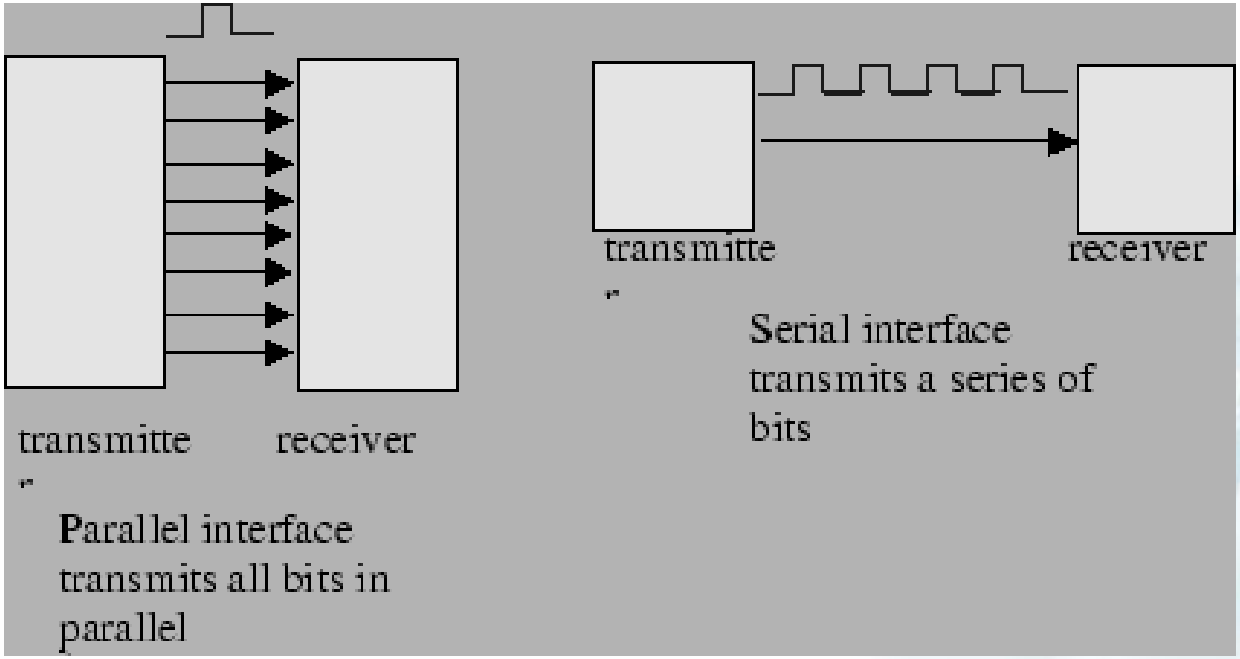
5.3. Nguyên lý bit cổng

5.4. Mạch nối ghép ngoại vi lập trình được 8255A (Programable Peripheral Interface - PPI 8255A)

5.5. Nối ghép truyền dữ liệu nối tiếp

Truyền nối tiếp # truyền song song

- Trao đổi thông tin theo phương thức truyền nối tiếp là truyền liên tiếp từng bit một trên một đường truyền.
- VD: PS2, COM, USB…



- **Lý do sử dụng truyền thông tin nối tiếp:**
  - Bus dữ liệu của hệ VXL được thiết kế để trao đổi dữ liệu song song với các mạch vào-ra. Tuy nhiên trong nhiều trường hợp, người ta phải thực hiện trao đổi thông tin nối tiếp có tốc độ chậm hơn
  - Khoảng cách giữa hai đơn vị cần trao đổi dữ liệu là tương đối lớn -> giảm giá thành

Truyền tin nối tiếp

Hệ thống trao đổi thông tin nối tiếp gồm có các dạng:

- Đơn công (Simplex Connection): số liệu chỉ được truyền theo 1 hướng.

Truyền tin nối tiếp

- Bán song công (Half-Duplex): số liệu có thể truyền đi 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

- **Phương thức hoạt động:**
  - Ở đầu phát, dữ liệu dưới dạng song song đầu tiên được chuyển thành dữ liệu dạng nối tiếp. Tín hiệu nối tiếp sau đó được truyền đi liên tiếp từng bit trên đường dây.
  - Ở đầu thu, tín hiệu nối tiếp sẽ được biến đổi ngược lại để chuyển sang dạng song song thích hợp cho việc xử lý tiếp theo

Truyền dữ liệu nối tiếp đồng bộ

- Các thiết bị sử dụng chung 1 nguồn xung clock phát bởi 1 thiết bị hoặc từ nguồn ngoài. Mỗi bit truyền đi tại thời điểm xung clock chuyển mức (sườn lên hoặc sườn xuống của xung).

- Bộ nhận sử dụng sự chuyển mức của xung để xác định thời điểm đọc các bit. Nó có thể đọc các bit tại sườn lên hay sườn xuống của xung hoặc theo mức logic cao thấp.

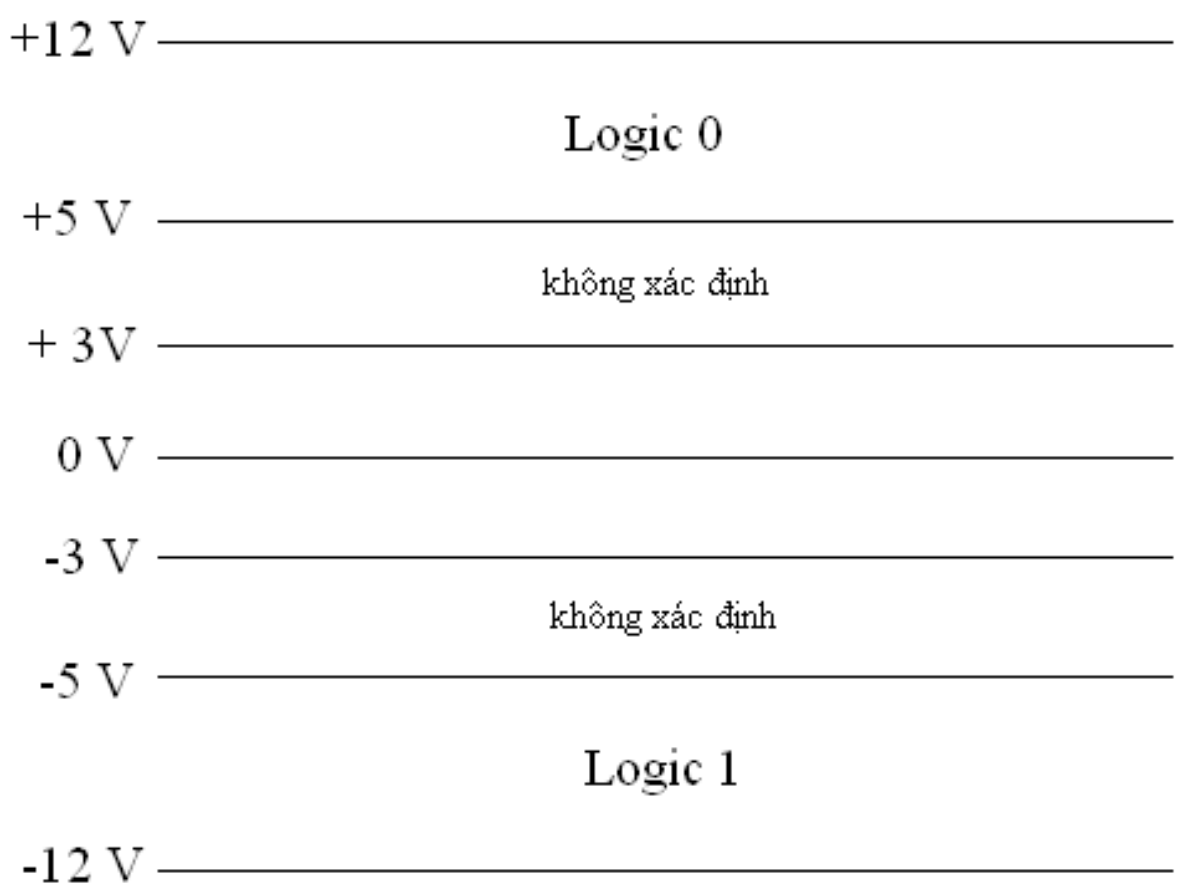
Truyền dữ liệu nối tiếp không đồng bộ

- Một gói dữ liệu truyền đi bao gồm bit start để đồng bộ hóa nguồn clock, 1 hoặc nhiều hơn 1 bit stop để báo kết thúc truyền 1 byte.
- Ngoài ra khung truyền còn có bit parity có thể là even,odd,mark hay space.

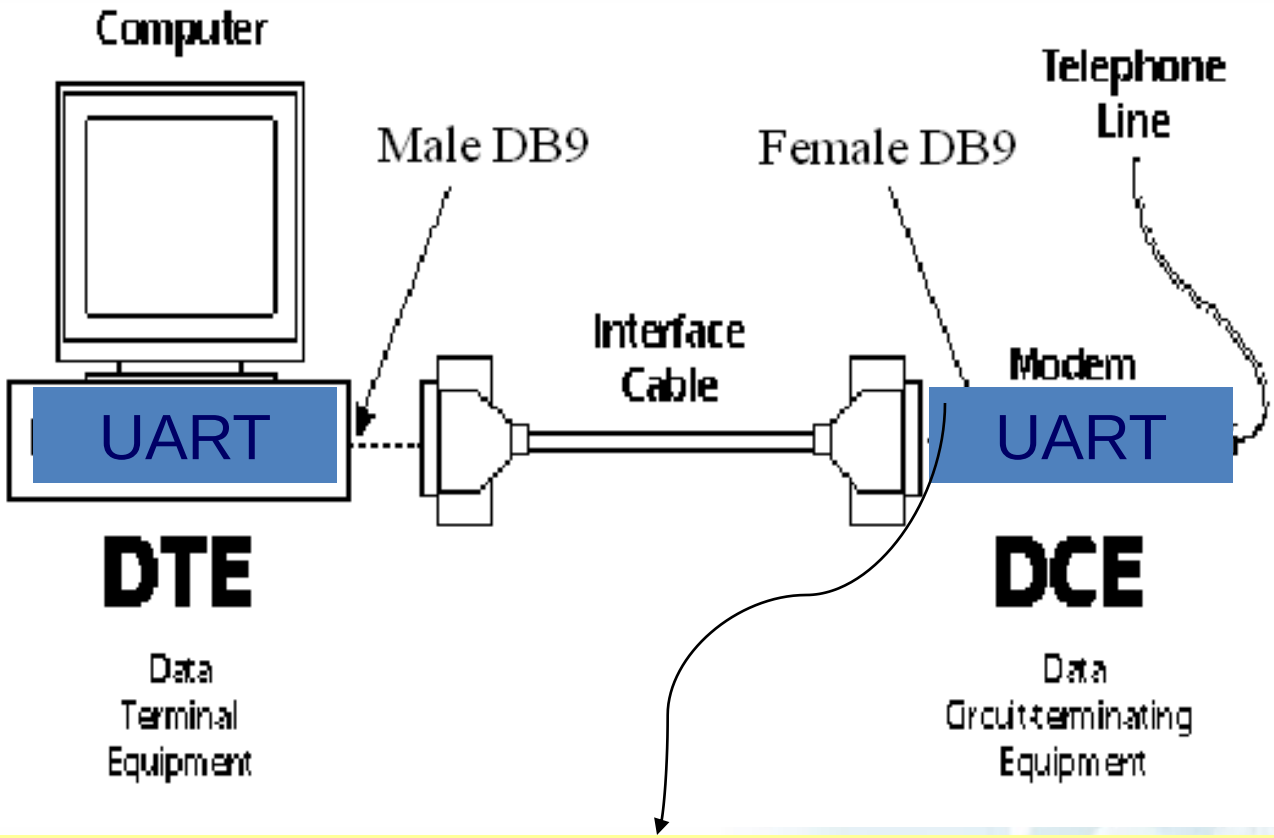
Chuẩn RS232

- Mức điện áp đường truyền
- Chuẩn đầu nối trên máy tính PC
- Khuôn dạng khung truyền
- Tốc độ truyền
- Kịch bản truyền

- Mức điện áp đường truyền (Chuẩn RS-232C)

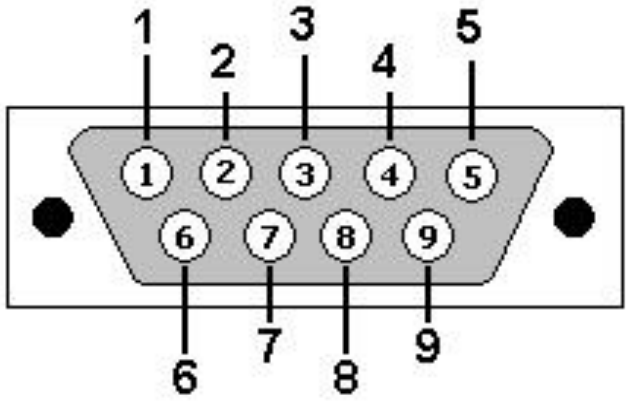


- **Chuẩn đấu nối trên PC**



UART (Universal Asynchronous receiver/transmitter)

- **Chuẩn đầu nối trên PC**
  - Chân 1 (DCD-Data Carrier Detect): phát hiện tín hiệu mang dữ liệu
  - Chân 2 (RxD-Receive Data): nhận dữ liệu
  - Chân 3 (TxD-Transmit Data): truyền dữ liệu
  - Chân 4 (DTR-Data Terminal Ready): đầu cuối dữ liệu sẵn sàng
  - Chân 5 (Signal Ground): đất của tín hiệu
  - Chân 6 (DSR-Data Set Ready): dữ liệu sẵn sàng
  - Chân 7 (RTS-Request To Send): yêu cầu gửi
  - Chân 8 (CTS-Clear To Send): Xóa để gửi
  - Chân 9 (RI-Ring Indicate): báo chuông



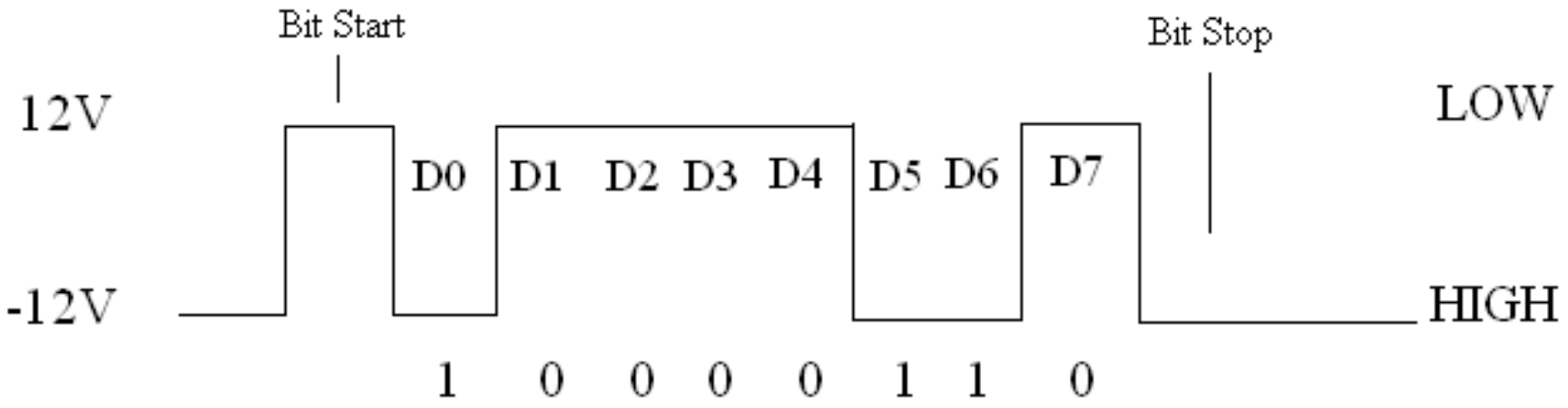
Chuẩn RS-232:

Chân số		Tên	Kí hiệu	Hướng DTE - DCE	Chức năng
D-25	D-9				
1	-	Frame Ground	FG	-	Thường được nối với vỏ bọc kim loại của cáp dẫn hoặc đất
2	3	Transmit Data	TxD	→	Số liệu được phát từ DTE tới DCE qua đường TxD
3	2	Receive Data	RxD	←	Số liệu được thu từ DCE vào DTE
4	7	Request To Send	RTS	→	DTE đặt đường này ở mức tích cực khi nó sẵn sàng phát số liệu
5	8	Clear To Send	CTS	←	DCE đặt đường này ở mức tích cực để thông tin cho DTE rằng nó sẵn sàng nhận số liệu

Chân số		Tên	Kí hiệu	Hướng DTE - DCE	Chức năng
D-25	D-9				
6	6	Data Set Ready	DSR	←	Chức năng tương tự như CTS nhưng được kích hoạt bởi DTE khi nó sẵn sàng nhận số liệu
20	4	Data Terminal Ready	DTR	→	Chức năng tương tự như RTS nhưng được kích hoạt bởi DCE khi nó muốn phát số liệu
8	1	Data Carier Detect	DCD	←	DCE đặt đường này ở mức tích cực để báo cho DTE biết là đã thiết lập được liên kết với DCE từ xa (nhận được sóng mang từ bên DCE đối tác)
22	9	Ring Indicator	RI	←	DCE (loại lắp ngoài) báo với DTE có một cuộc gọi từ xa vừa gọi đến
7	5	Signal Ground	SG	-	GND

- **Khuôn dạng khung truyền**
  - PC truyền nhận dữ liệu qua cổng nối tiếp RS-232 thực hiện theo kiểu không đồng bộ (Asynchronous)
  - Khung truyền gồm 4 thành phần
    - ✓1 Start bit (Mức logic 0): bắt đầu một gói tin, đồng bộ xung nhịp clock giữa DTE và DCE
    - ✓Data (5,6,7,8 bit): dữ liệu cần truyền
    - ✓1 parity bit (chẵn (even), lẻ (odd), mark, space): bit cho phép kiểm tra lỗi
    - ✓Stop bit (1 hoặc 2 bit): kết thúc một gói tin

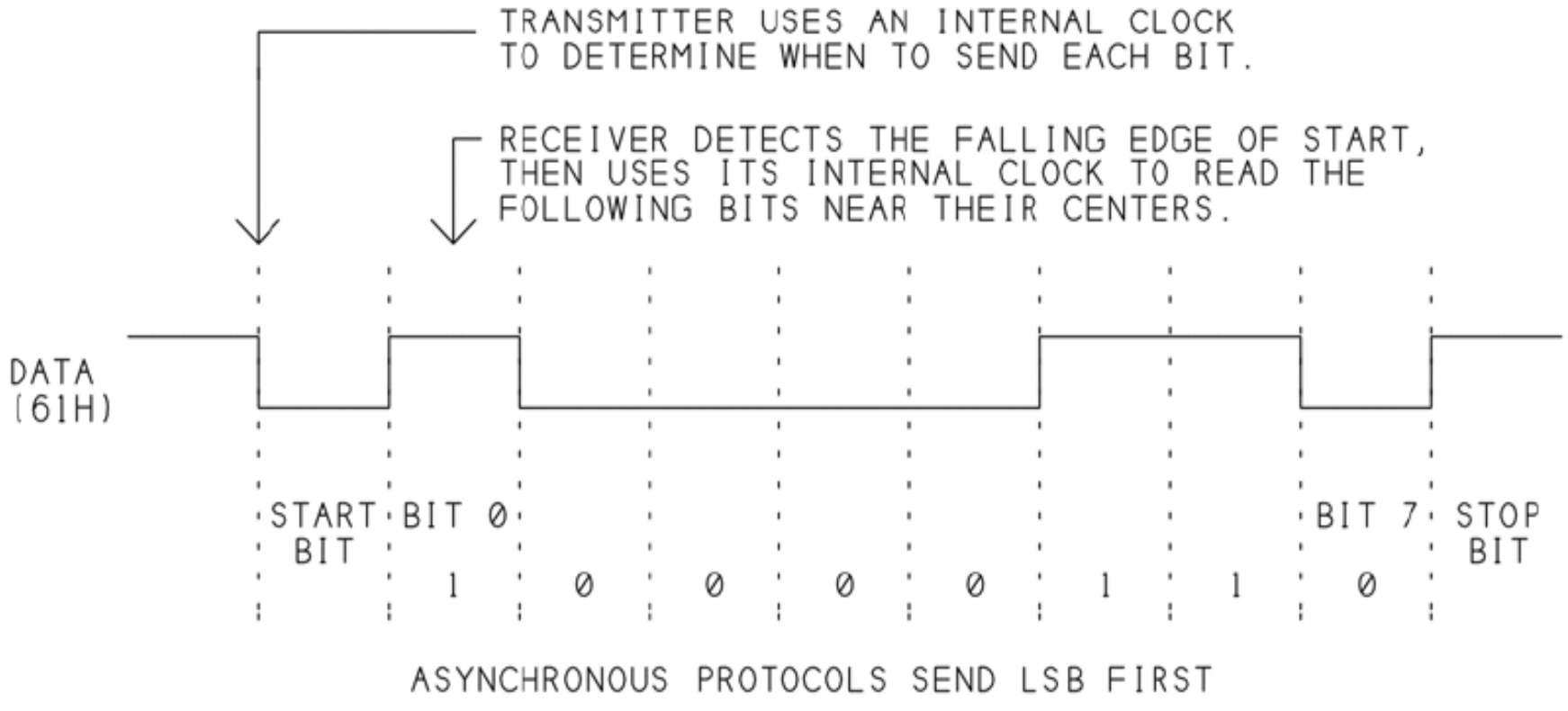
- **Khuôn dạng khung truyền**



Khung truyền cho dữ liệu 61h theo khung: (8, N, 1)

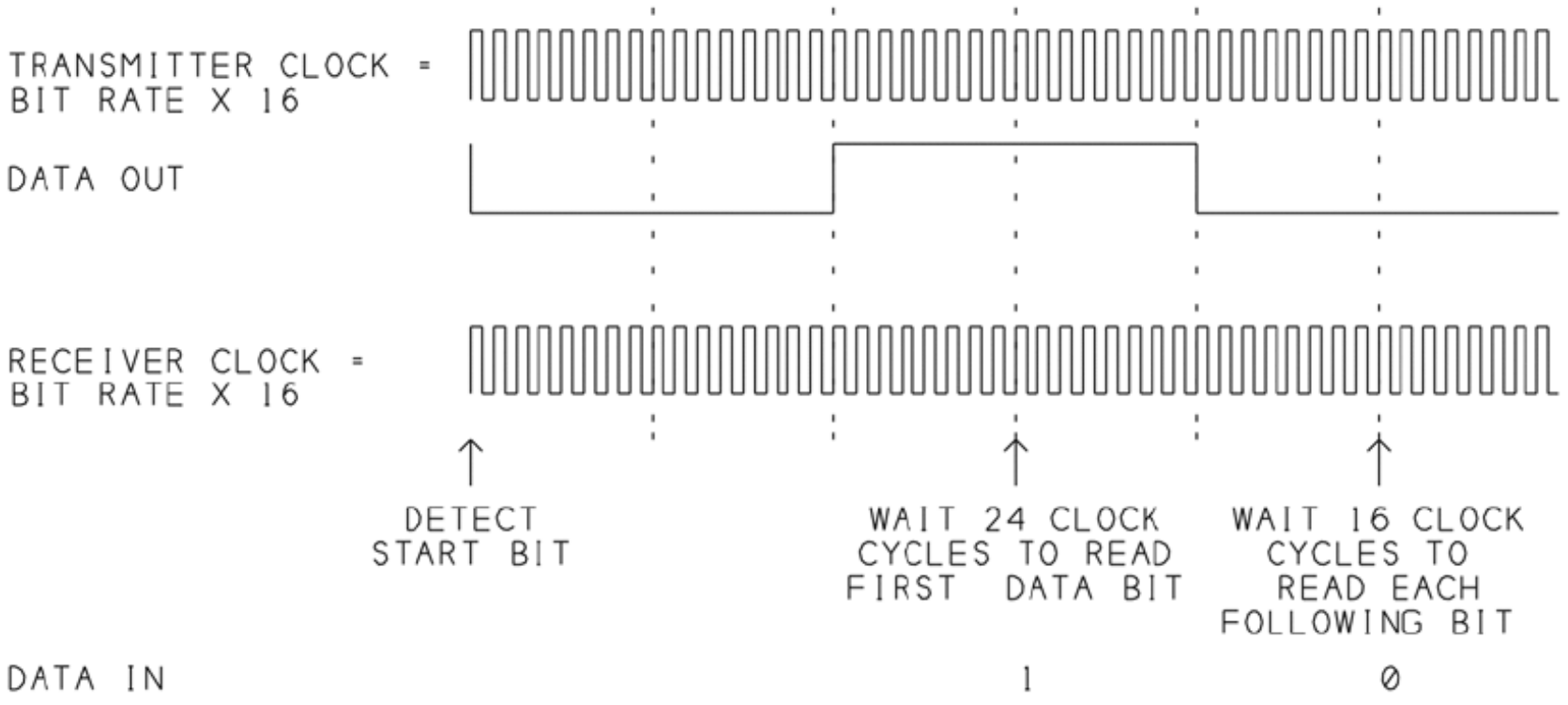
- 8 bit dữ liệu
- 1 bit stop
- Không có bit parity

- Khuôn dạng khung truyền



Xác định thời điểm truyền và nhận dữ liệu

- Khuôn dạng khung truyền

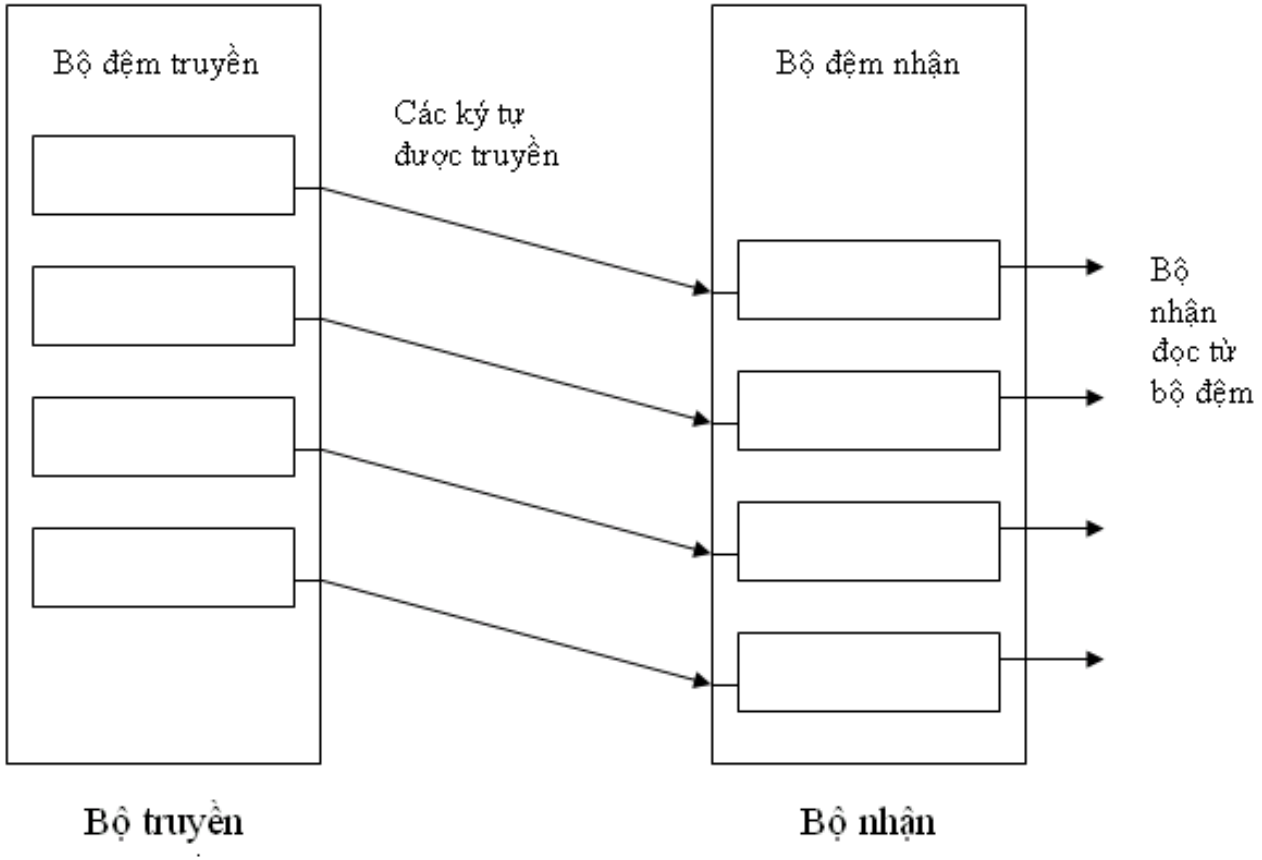


Quá trình truyền và nhận dữ liệu dưới tác động của các xung nhịp đồng hồ tại bên truyền và bên nhận

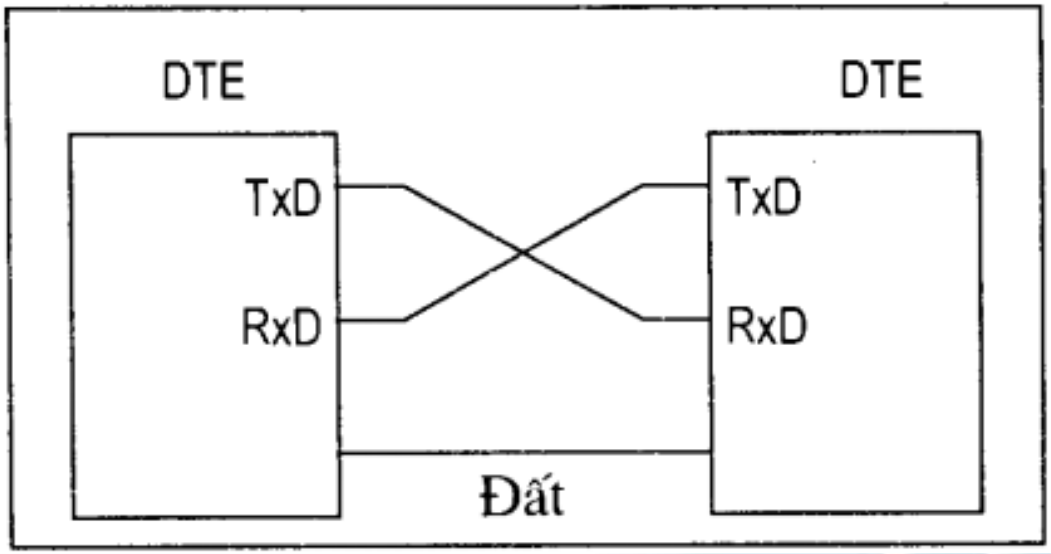
Chuẩn RS232

- Tốc độ truyền
  - Tính bằng đơn vị bit/giây: bps (bit per second)
  - Thuật ngữ khác: baud
    - ✓Đơn vị đo dùng cho modem
    - ✓Số lần thay đổi tín hiệu trong một giây
    - ✓Đối với modem, mỗi lần thay đổi tín hiệu, có thể truyền được nhiều bit : tốc độ baud <= tốc độ bit
  - Tốc độ tối đa = Tần số xung nhịp clock / hằng số
  - VD: trong máy PC, tần số 1.8432MHz, hằng số =16 -> tốc độ tối đa: 115,200 bps
  - Bên trong UART hỗ trợ các thanh ghi cho phép xác định các tốc độ làm việc khác, vd: 1200, 2400, 4800, 9600, 19200, 38400… bps

- Kịch bản truyền
  - Quá trình truyền và nhận các ký tự

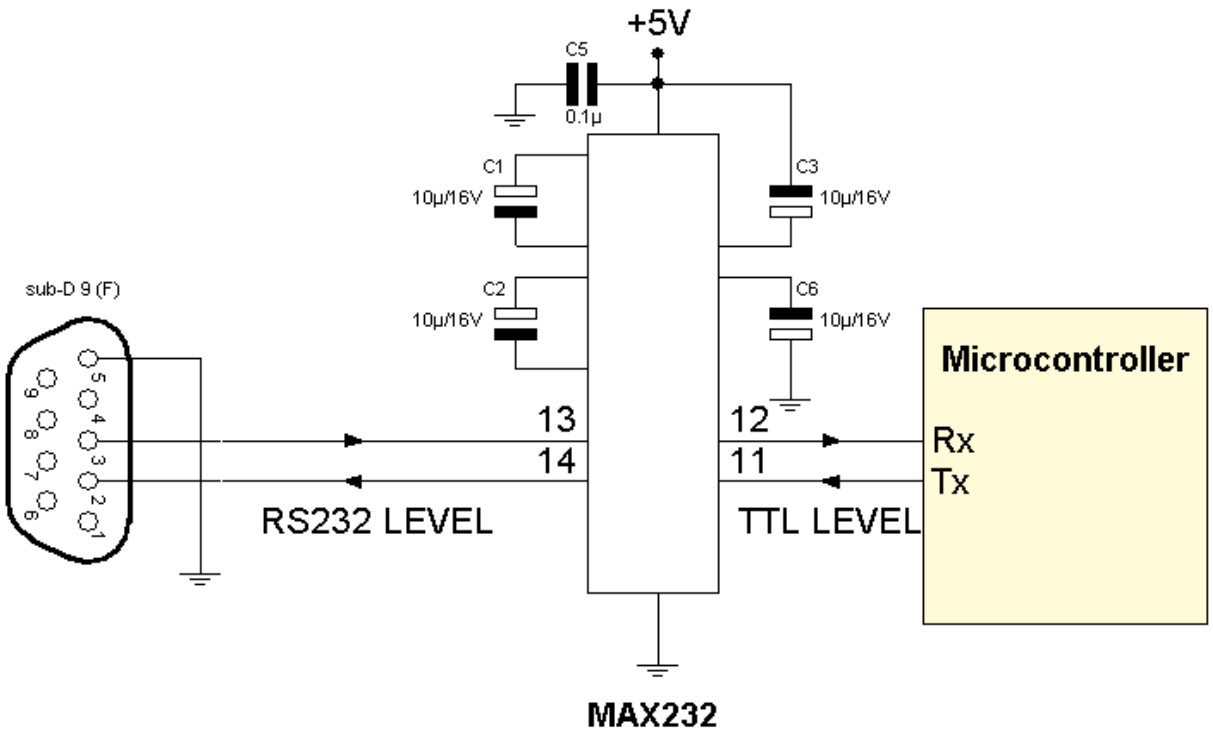


- **Kịch bản truyền**
  - Không có bắt tay (none-handshaking): máy thu có khả năng đọc các ký tự thu trước khi máy phát truyền ký tự tiếp theo



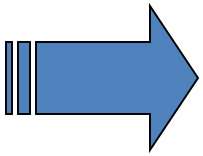
Kết nối không cần bắt tay giữa hai thiết bị (cùng mức điện áp)

- **Kịch bản truyền**



Ghép nối không bắt tay giữa hai thiết bị
(Khác nhau về mức điện áp)

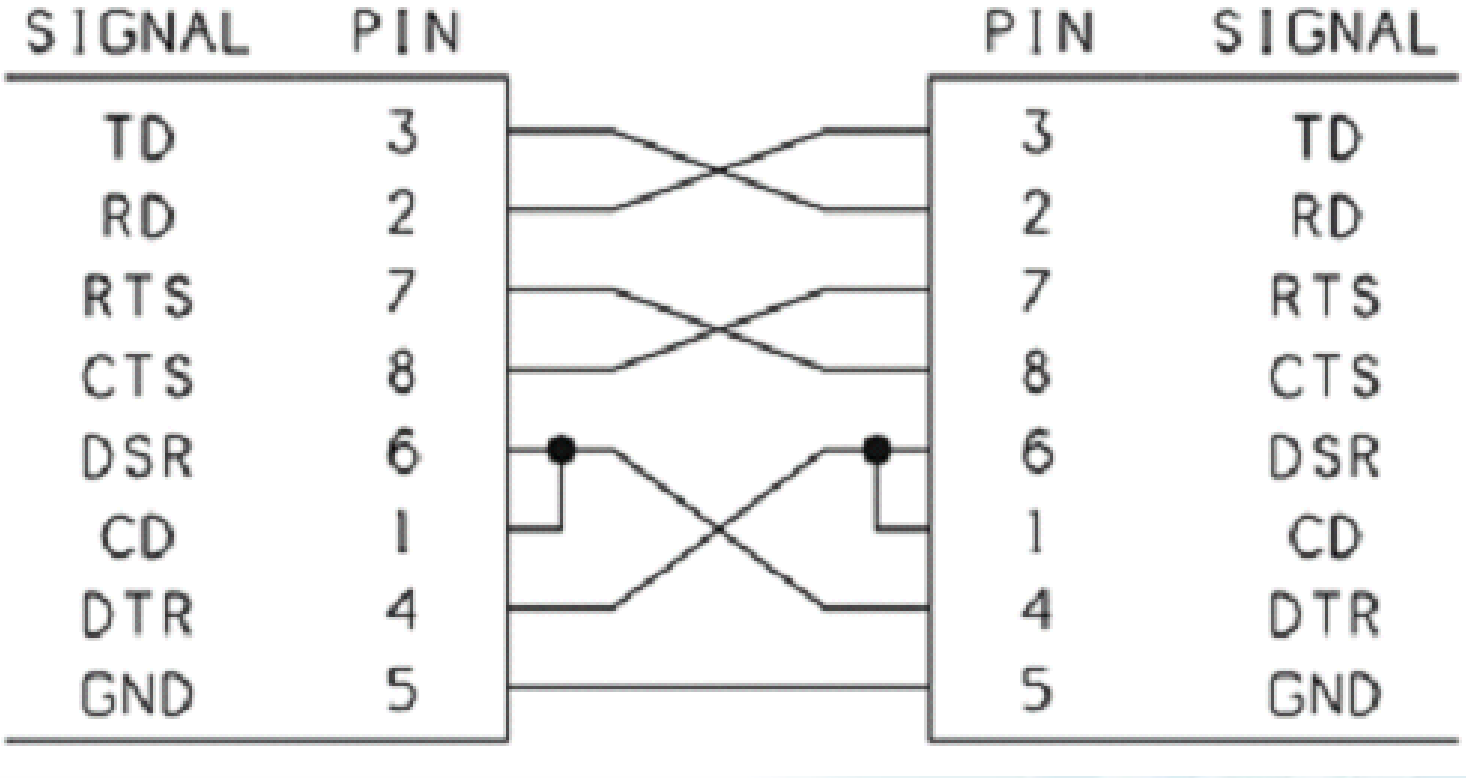
- Kịch bản truyền
  - Có bắt tay (handshaking):
    - ✓Máy thu nhận các ký tự và lưu vào một vùng nhớ gọi là bộ đệm thu (receive buffer)
    - ✓Nếu ký tự tại bộ đệm thu không được đọc kịp trước khi ký tự khác được truyền tới -> có thể xảy ra hiện tượng các ký tự hiện tại bị ghi đè bởi các ký tự mới



Cần tín hiệu điều khiển, buộc máy phát ngừng phát cho đến khi máy thu đọc xong các ký tự đang nằm trong bộ đệm thu

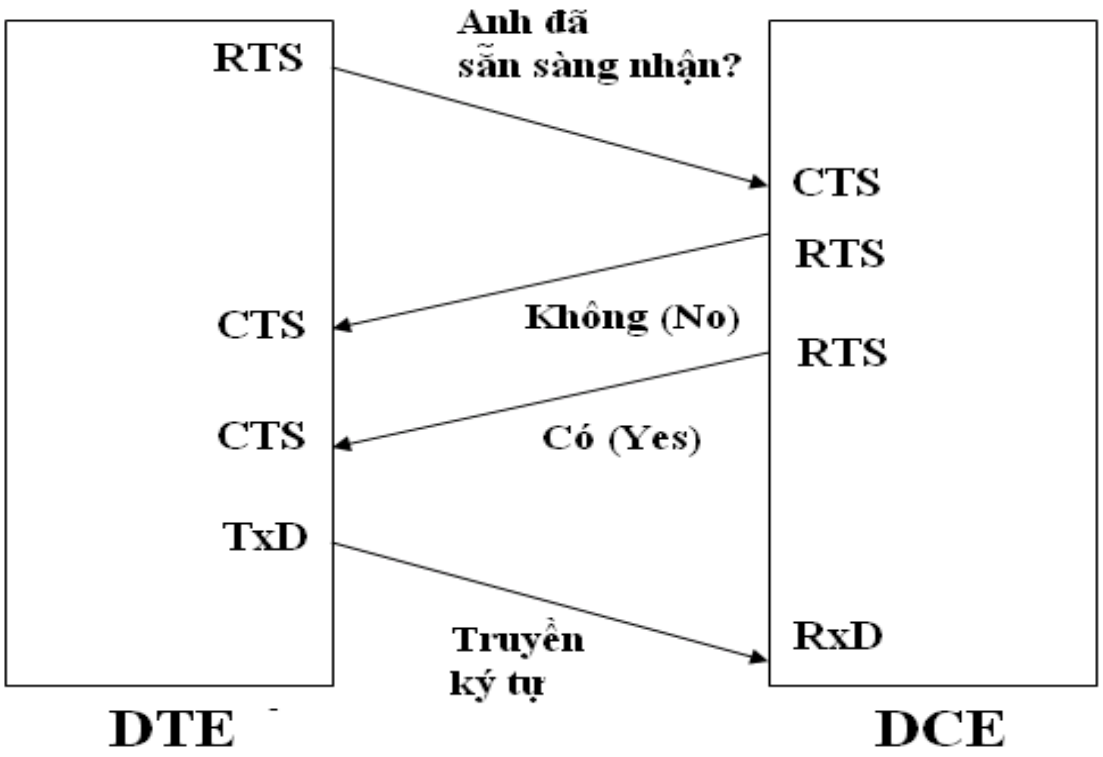
- Kịch bản truyền
  - Có bắt tay (handshaking)
    - ✓Bắt tay bằng phần cứng
      - Sử dụng các tín hiệu bắt tay RTS, CTS, DTR, DSR
    - ✓Bắt tay bằng phần mềm
      - Sử dụng các gói tin đặc biệt truyền các ký tự Xon, Xoff.
    - ✓Bắt tay kết hợp cả phần cứng và phần mềm

- **Kịch bản truyền**
  - Bắt tay bằng phần cứng



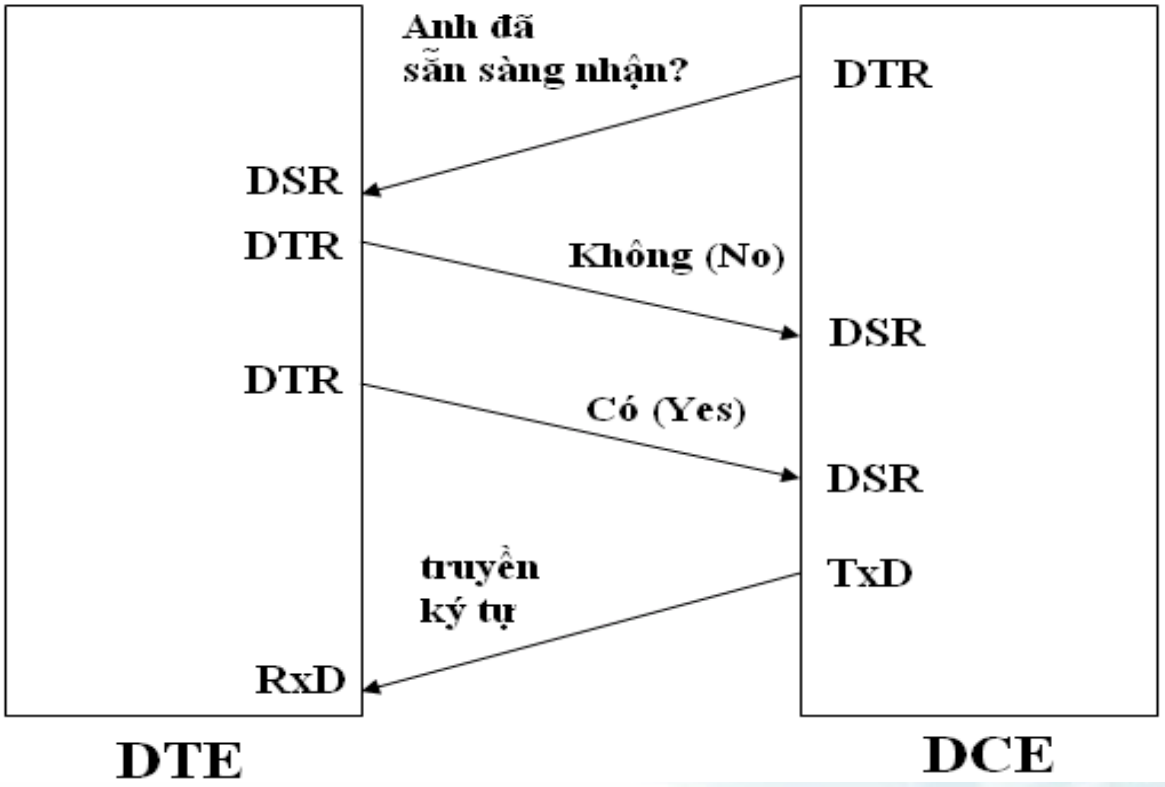
Sơ đồ đấu nối tín hiệu bắt tay bằng phần cứng

- **Kịch bản truyền**
  - Bắt tay phần cứng



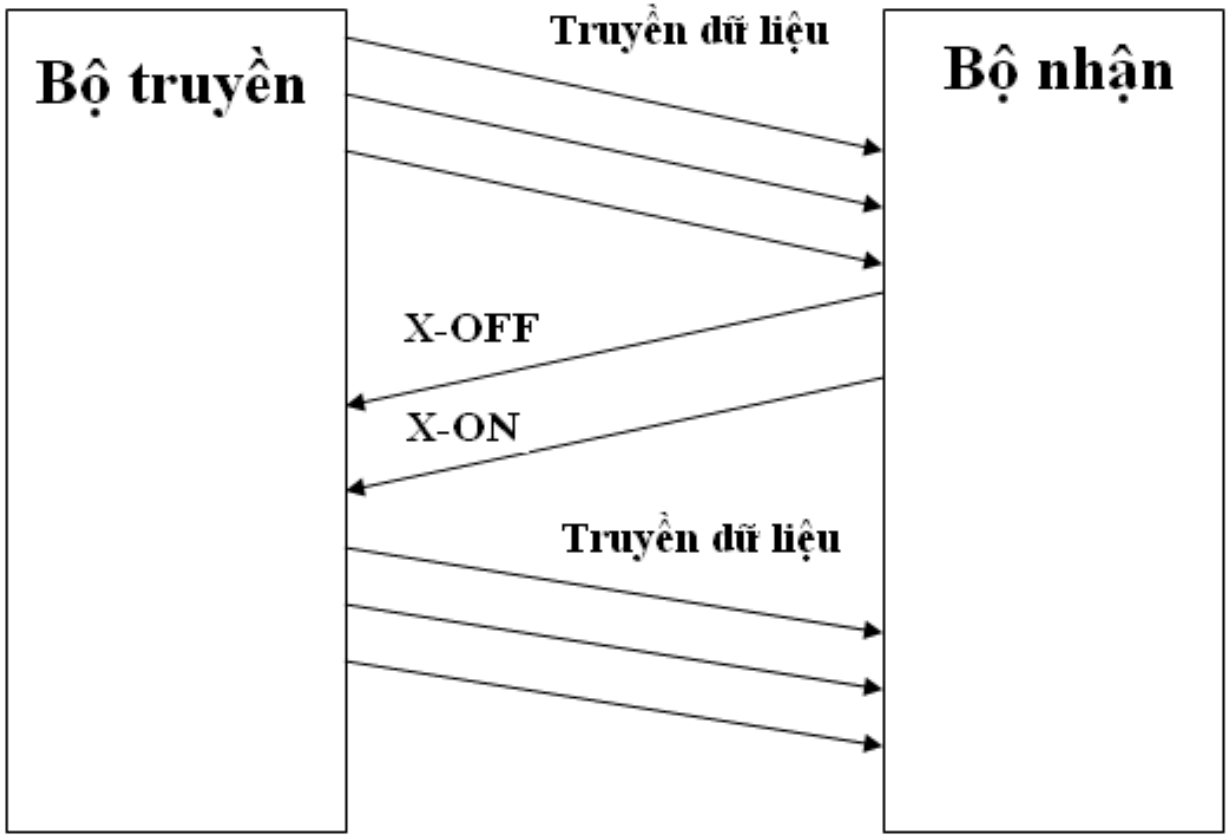
Các đường tín hiệu bắt tay được sử dụng khi DTE truyền dữ liệu

- **Kịch bản truyền**
  - Bắt tay bằng phần cứng

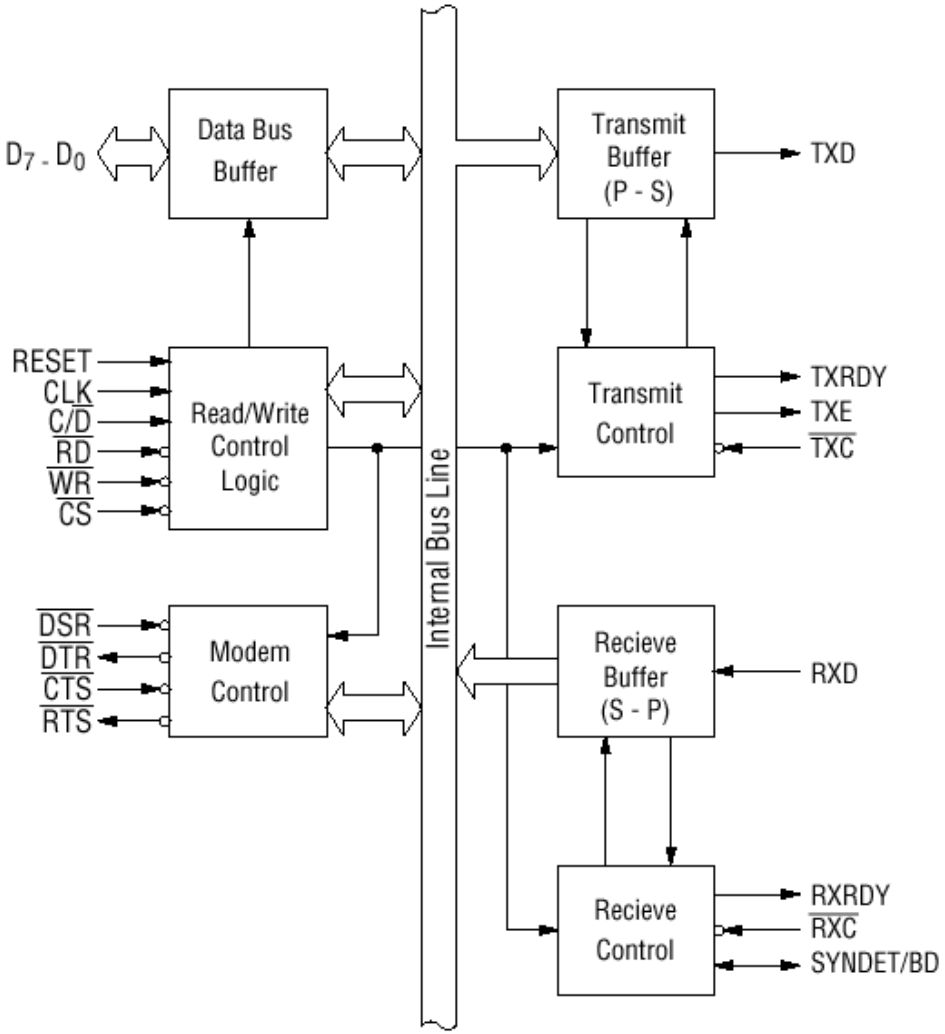


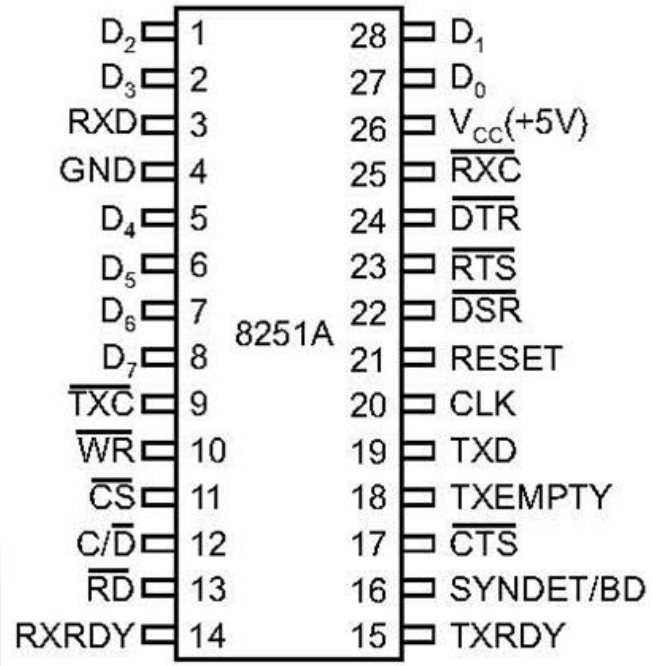
Các đường tín hiệu bắt tay được sử dụng khi DTE nhận dữ liệu

- Bắt tay bằng phần mềm
  - Sử dụng hai ký tự ASCII: X-ON (Ctrl-S) và X-OFF (Ctrl-Q)



Bộ 8251 USART





Tổ chức chân 8251A

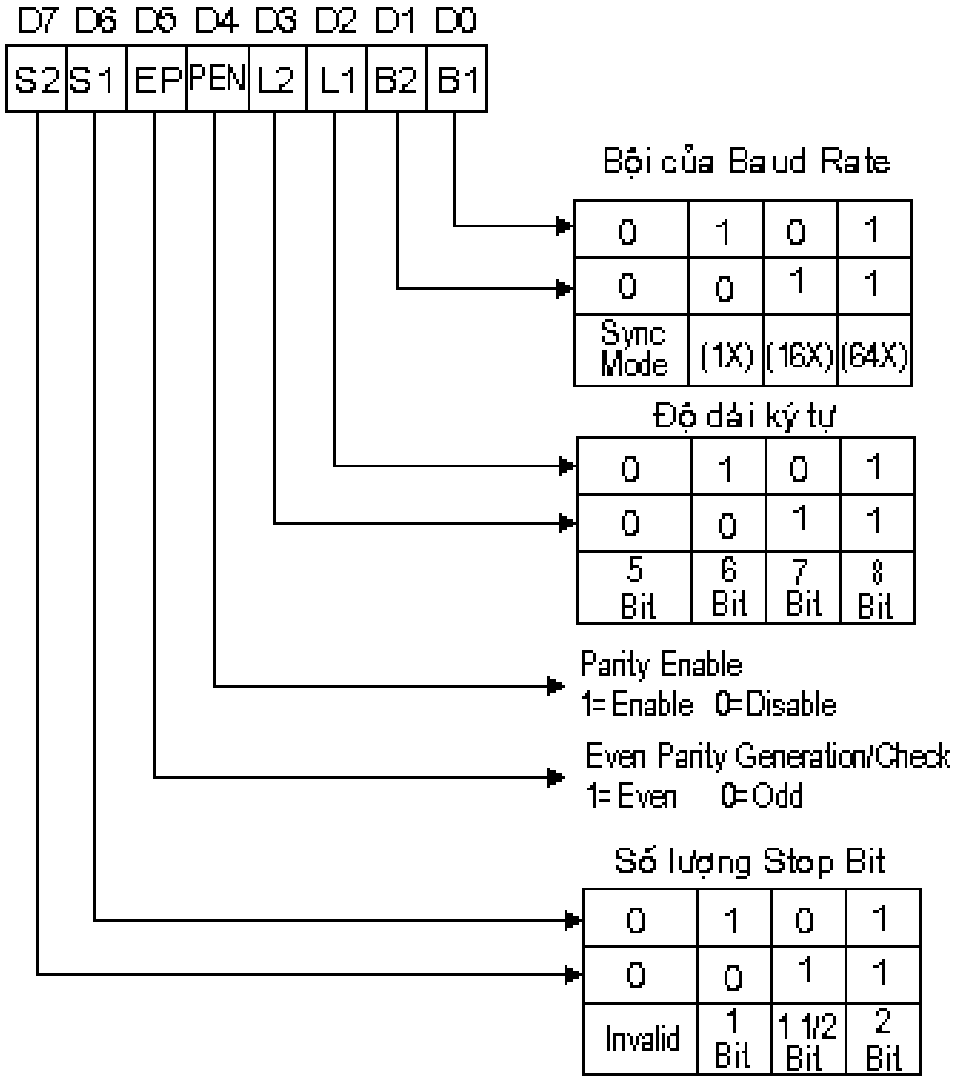
- CLK: chân nối tới xung đồng hồ của hệ thống
- TxRDY: tín hiệu báo bộ đệm giữ rỗng (sẵn sàng nhận dữ liệu mới từ CPU)
- RxRDY: tín hiệu báo bộ đệm thu đầy (có ký tự nằm chờ CPU đọc vào)
- TxEMPTY: báo cả đệm giữ và thu đều rỗng
- C/D: CPU chọn thanh ghi lệnh hay thanh ghi dữ liệu
- RxC và TxC: xung đồng hồ cung cấp cho thanh ghi dịch của phần thu và phát

Các thanh ghi bên trong 8251A

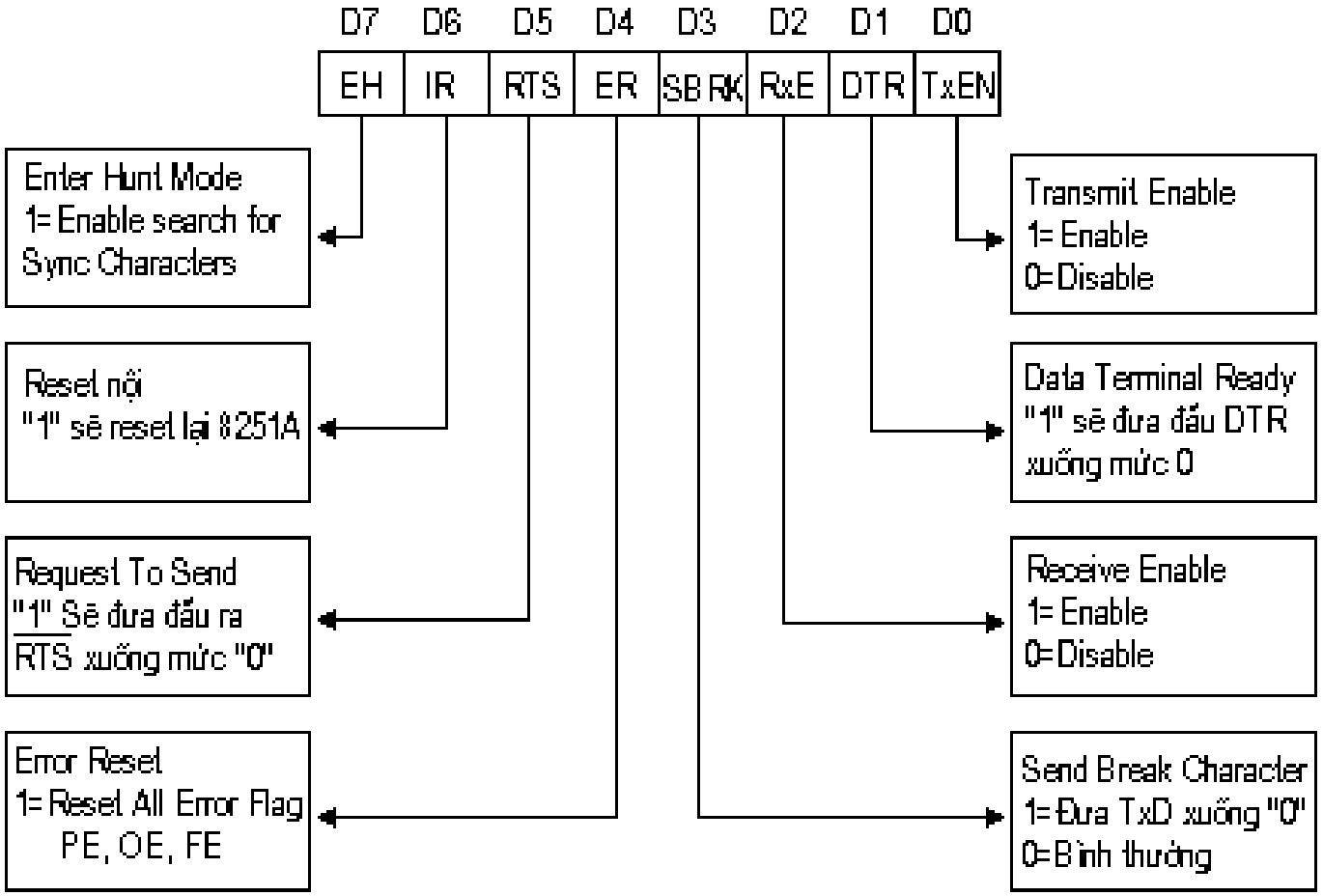
- Thanh ghi đệm dữ liệu thu
- Thanh ghi đệm dữ liệu phát
- Thanh ghi trạng thái
- Thanh ghi điều khiển (thanh ghi lệnh và thanh ghi chế độ ➔ Chú ý: ghi 2 lần, thanh ghi lện trước, chế độ sau)
- Sử dụng tín hiệu tại chân C/D, chân WR và RD để chọn thanh ghi làm việc

A0	RD	WR	Chọn thanh ghi
0	0	1	Thanh ghi đệm thu
0	1	0	Thanh ghi đệm phát
1	0	1	Thanh ghi trạng thái
1	1	0	Thanh ghi điều khiển

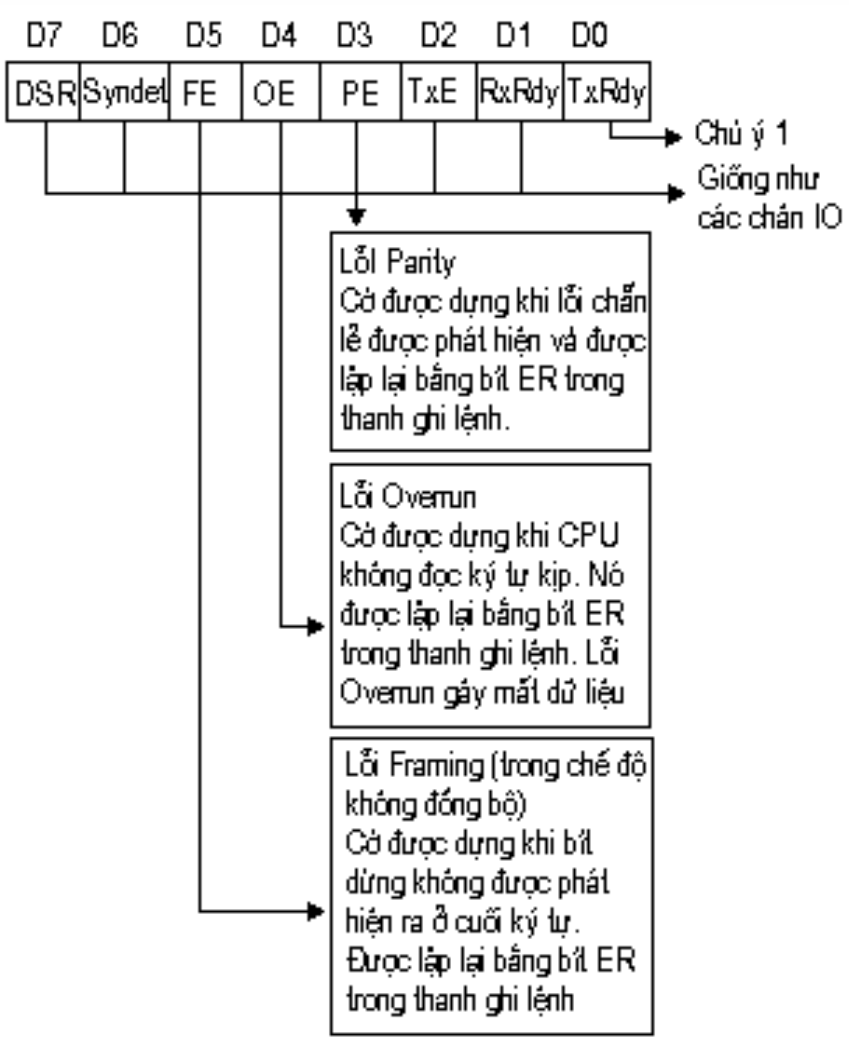
Thanh ghi chế độ của 8251A



Thanh ghi lệnh của 8251A



Thanh ghi trạng thái của 8251A



Bài tập 1

- Vẽ sơ đồ ghép nối CPU 8086 với USART 8251A ở địa chỉ 00h (thanh ghi dữ liệu) và 02h (thanh ghi điều khiển/trạng thái.

Bài tập 2

- Giả sử tốc độ xung clock tham chiếu đưa vào mạch tạo baud của 8255 là 9600 Hz. Xác định giá trị thanh ghi chế độ để 8255 làm việc ở cấu hình
  - 9600, 8, N, 1
  - 600, 7, N, 1

CHƯƠNG 6

Ngắt và xử lý ngắt trong 8088

Nội dung chương 6

6.1. Phương pháp thăm dò và phương pháp ngắt

6.2. Tổng quan về ngắt

6.3. Xử lý ngắt trong 8088

6.1. Thăm dò và ngắt

- Phương pháp thăm dò: bộ vi điều khiển liên tục kiểm tra yêu cầu của các thiết bị để phục vụ. Quá trình phục vụ diễn ra tuần tự nếu có nhiều thiết bị cùng yêu cầu
- Phương pháp ngắt: mỗi khi thiết bị yêu cầu phục vụ, thiết bị gửi tín hiệu yêu cầu tới chân ngắt của vi điều khiển. Vi điều khiển sẽ xử lý yêu cầu ngắt đó. Chương trình xử lý ngắt gọi là trình phục vụ ngắt (ISR-Interrupt Service Routine)

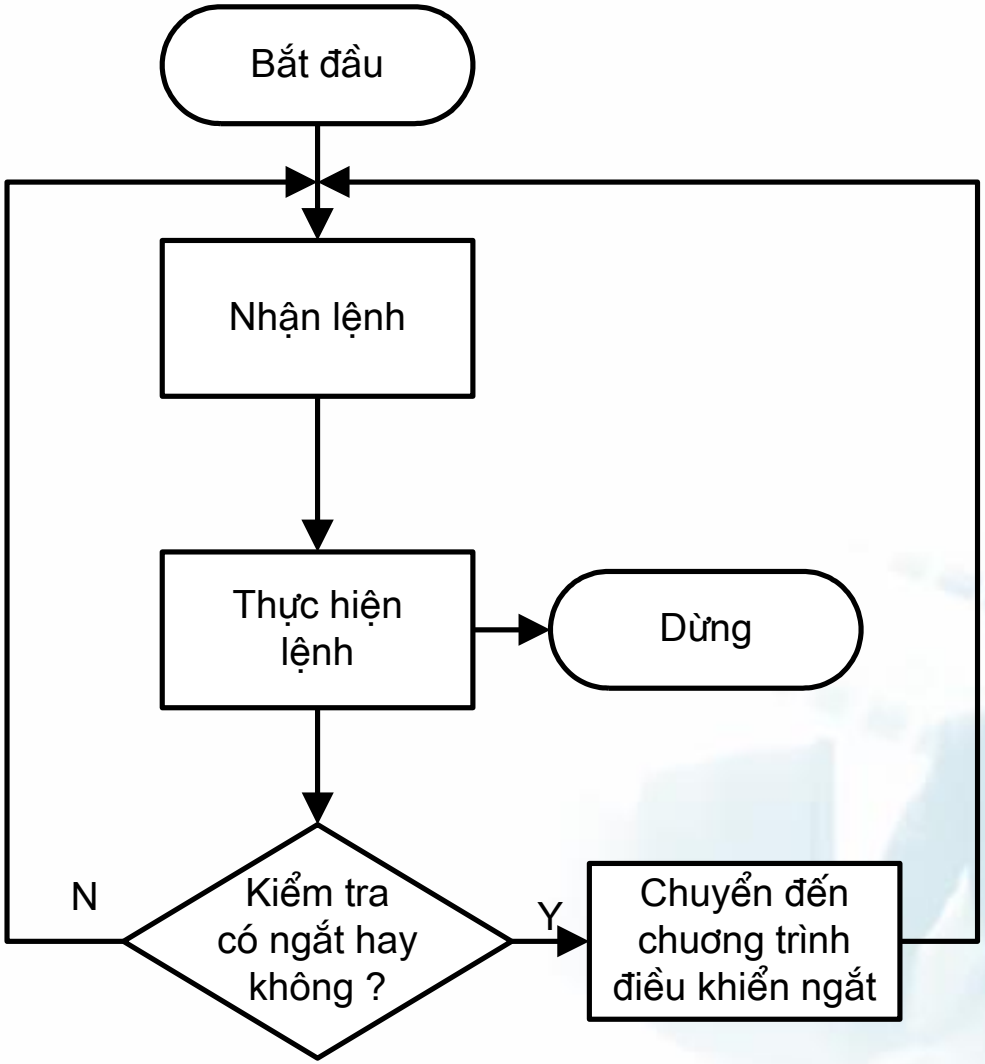
6.2. Tổng quan về ngắt

- Trình phục vụ ngắt
  - Mỗi ngắt luôn có một trình phục vụ ngắt
  - Khi một ngắt được kích hoạt thì vi điều khiển thực thi trình phục vụ ngắt
  - Trình phục vụ ngắt của mỗi ngắt có một vị trí cố định trong bộ nhớ
  - Tập hợp các ô nhớ lưu giữ địa chỉ của tất cả các trình phục vụ ngắt gọi là bảng vector ngắt (Interrupt Table)

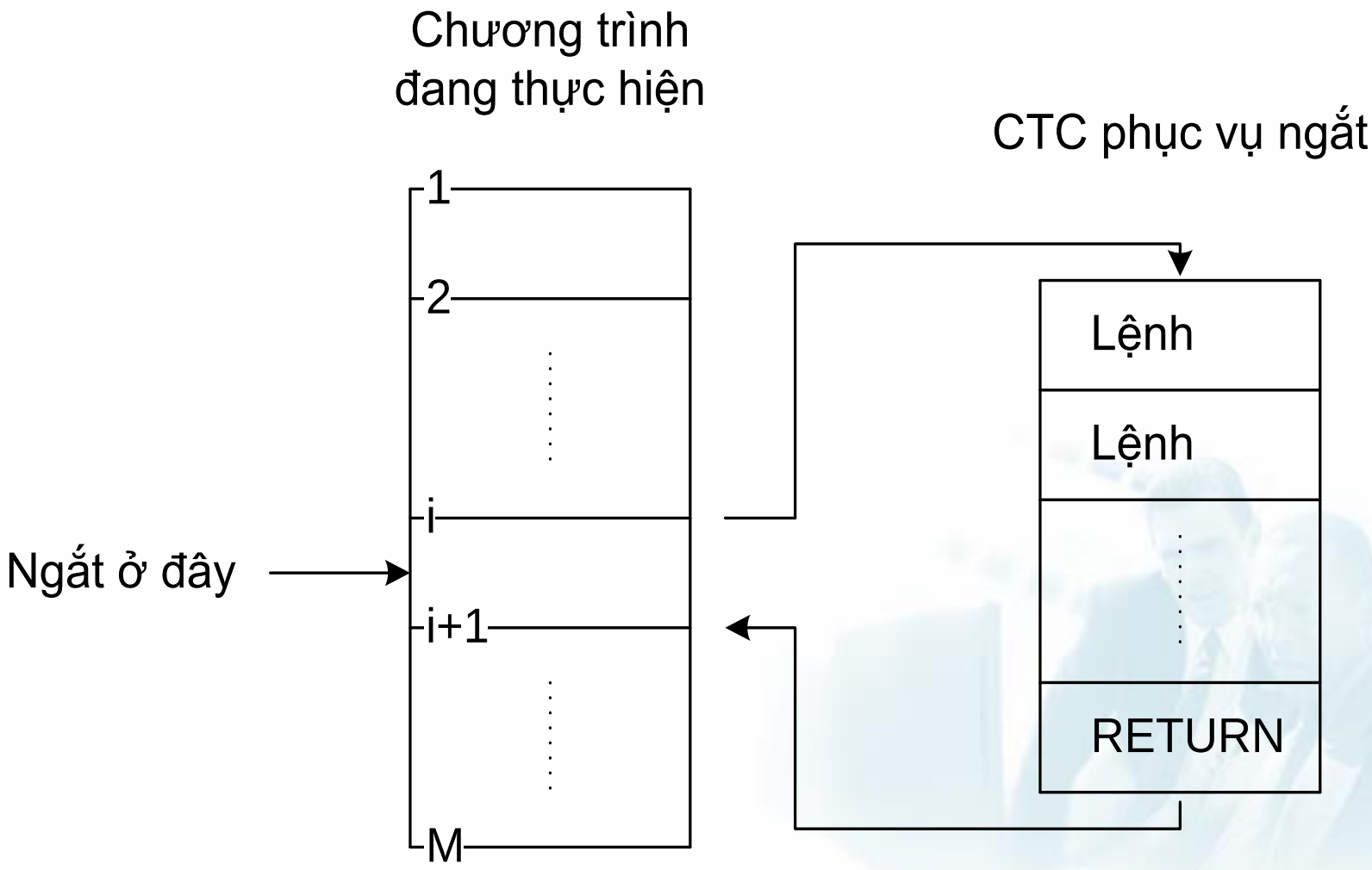
Chu trình xử lý ngắt

- Được thêm vào cuối chu trình lệnh
- Sau khi hoàn thành một lệnh, CPU kiểm tra xem có yêu cầu ngắt gửi đến hay không
  - Nếu không có tín hiệu yêu cầu ngắt thì CPU nhận lệnh kế tiếp
  - Nếu có yêu cầu ngắt và ngắt đó được chấp nhận thì:
    - ✓CPU cất ngữ cảnh hiện tại của chương trình đang thực hiện (các thông tin liên quan đến chương trình bị ngắt)
    - ✓CPU chuyển sang thực hiện chương trình con phục vụ ngắt tương ứng
    - ✓Kết thúc chương trình con đó, CPU khôi phục lại ngữ cảnh và trở về tiếp tục thực hiện chương trình đang tạm dừng

Hoạt động ngắt (tiếp)



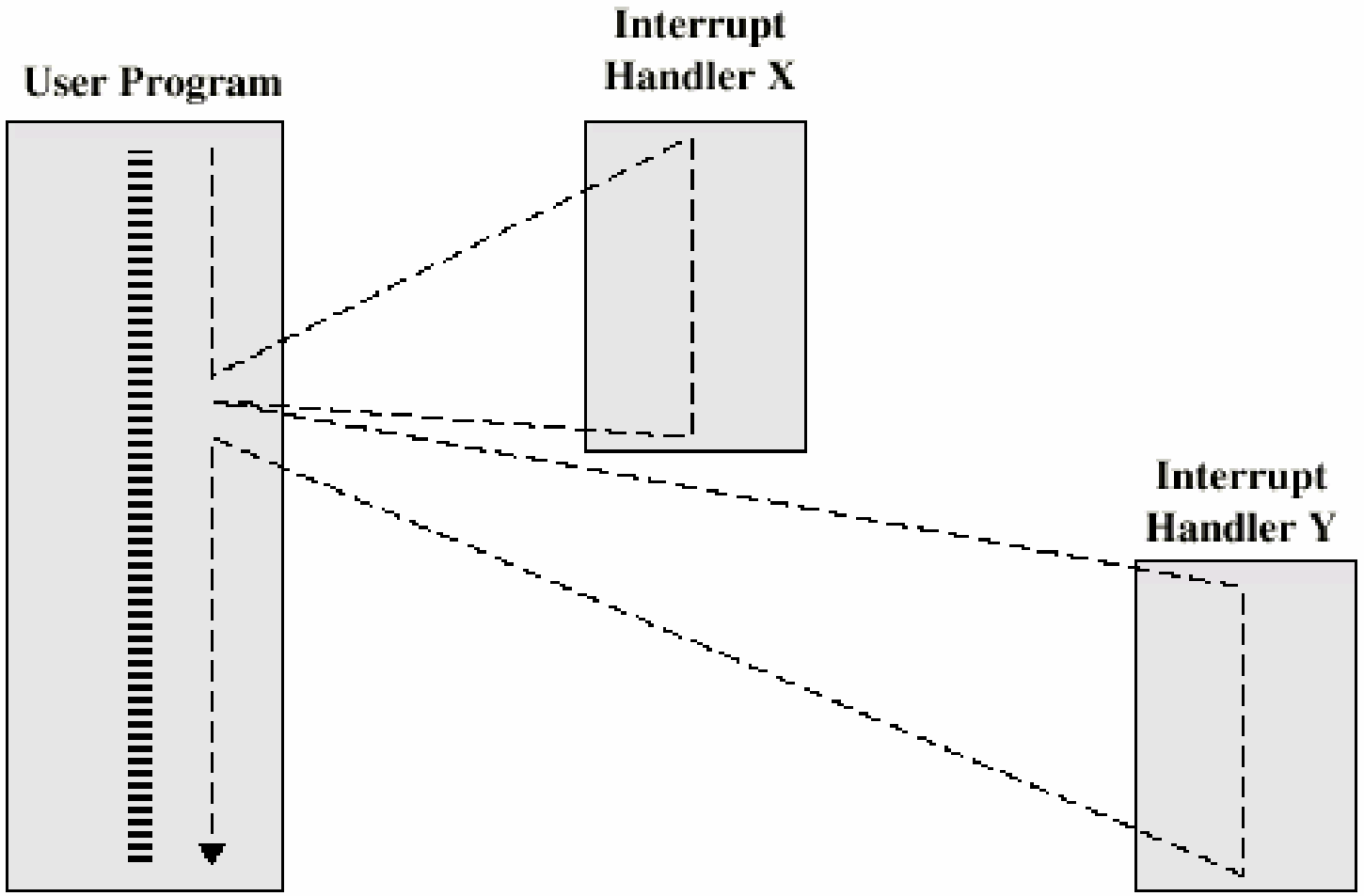
Hoạt động ngắt (tiếp)



Xử lý với nhiều tín hiệu yêu cầu ngắt

- Xử lý ngắt tuần tự:
  - Khi một ngắt đang được thực hiện, các ngắt khác sẽ bị cấm
  - Bộ xử lý sẽ bỏ qua các ngắt tiếp theo trong khi đang xử lý một ngắt
  - Các ngắt vẫn đang đợi và được kiểm tra sau khi ngắt đầu tiên được xử lý xong
  - Các ngắt được thực hiện tuần tự

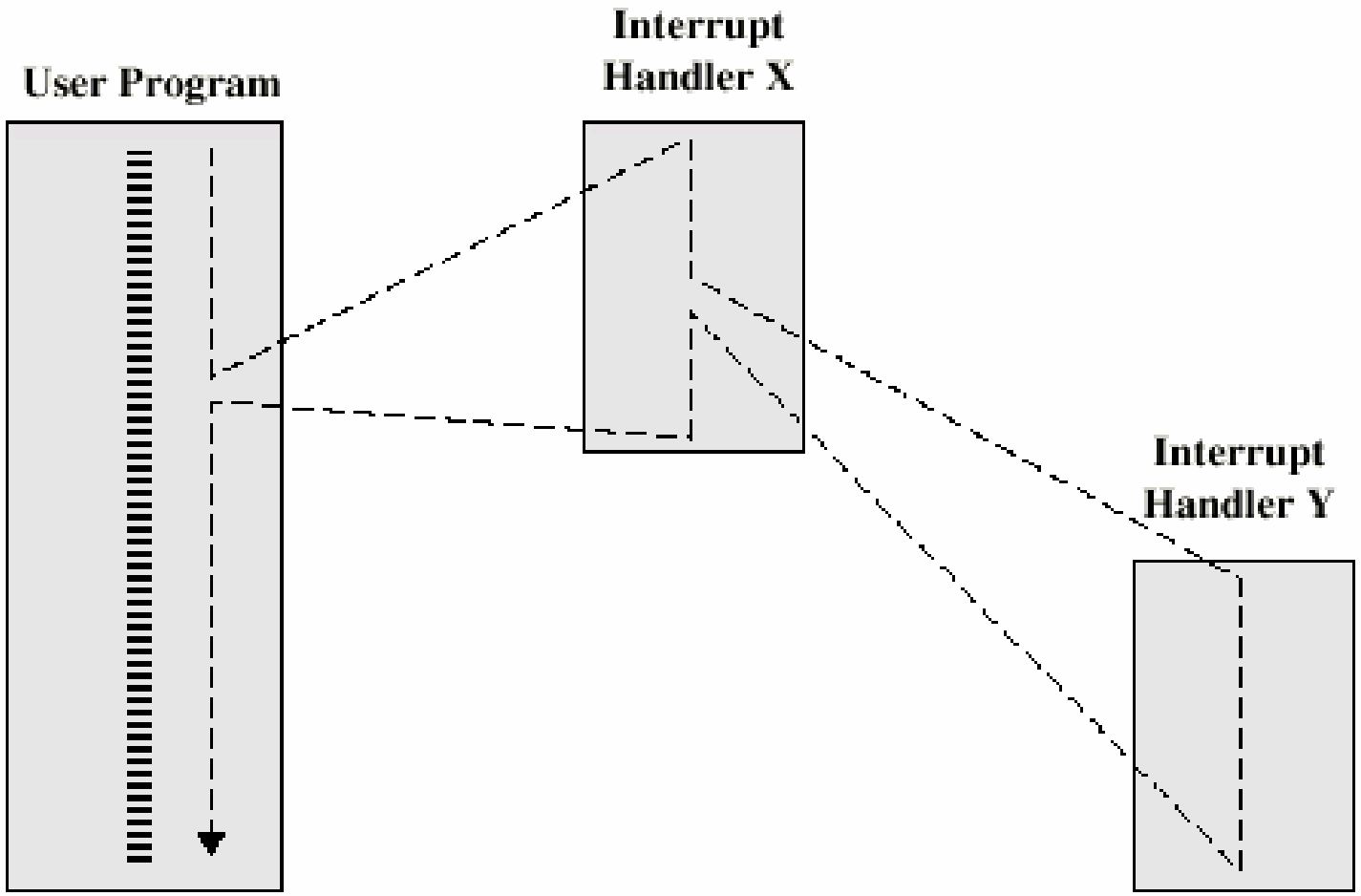
Xử lý ngắt tuần tự (tiếp)



Xử lý với nhiều tín hiệu yêu cầu ngắt

- Xử lý ngắt ưu tiên:
  - Các ngắt được định nghĩa mức ưu tiên khác nhau
  - Ngắt có mức ưu tiên thấp hơn có thể bị ngắt bởi ngắt ưu tiên cao hơn $\Rightarrow$ ngắt xảy ra lồng nhau

Xử lý ngắt ưu tiên



Các ngắt của 8088

- Ngắt cứng: các yêu cầu ngắt đến từ chân NMI và INTR
- Ngắt mềm
- Nhóm các hiện tượng ngoại lệ:lỗi chia o, tràn…

Bảng vector ngắt

- 256 vector
- Lưu ở 1KB đầu tiên của bộ nhớ RAM
- Mỗi vector ngắt 4 byte:
  - 2 byte cao: lưu CS (đoạn mã)
  - 2 byte thấp: lưu IP (con trỏ lệnh)