CHƯƠNG II: HỌ VI XỬ LÝ INTEL 80x86

2.1. CẤU TRÚC CỦA BỘ VI XỬ LÝ 8086

2.1.1. TỔNG QUAN

Sau khi đã tìm hiểu qua về cấu trúc của vi xử lý, tiếp theo chúng ta sẽ đi sâu tìm hiểu một bộ vi xử lý cụ thể và rất điển hình: bộ vi xử lý 80x86 của Intel. Đây là bộ vi xử lý nổi tiếng một thời của hãng Intel và được sử dụng nhiều trong các lĩnh vực khác nhau. Các chương trình viết cho 80x86 vẫn có thể chạy được trên trên các hệ tiên tiến sau này. Các họ vi xử lý của các hãng tuy có khác nhau nhưng xét cho cùng có khá nhiều điểm chủ yếu rất giống nhau, do đó một khi đã nắm vững các vấn đề kỹ thuật của 8086 ta sẽ có cơ sở để nắm bắt các bộ vi xử lý khác trong cùng họ của Intel hoặc các họ khác. Về góc độ sư phạm thì đây là bộ vi xử lý khá đơn giản vì vậy việc hiểu nó là tương đối đơn giản cho những người mới bắt đầu ra nhập vào lĩnh vực này.

Các thông số của 8086 như sau:

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

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

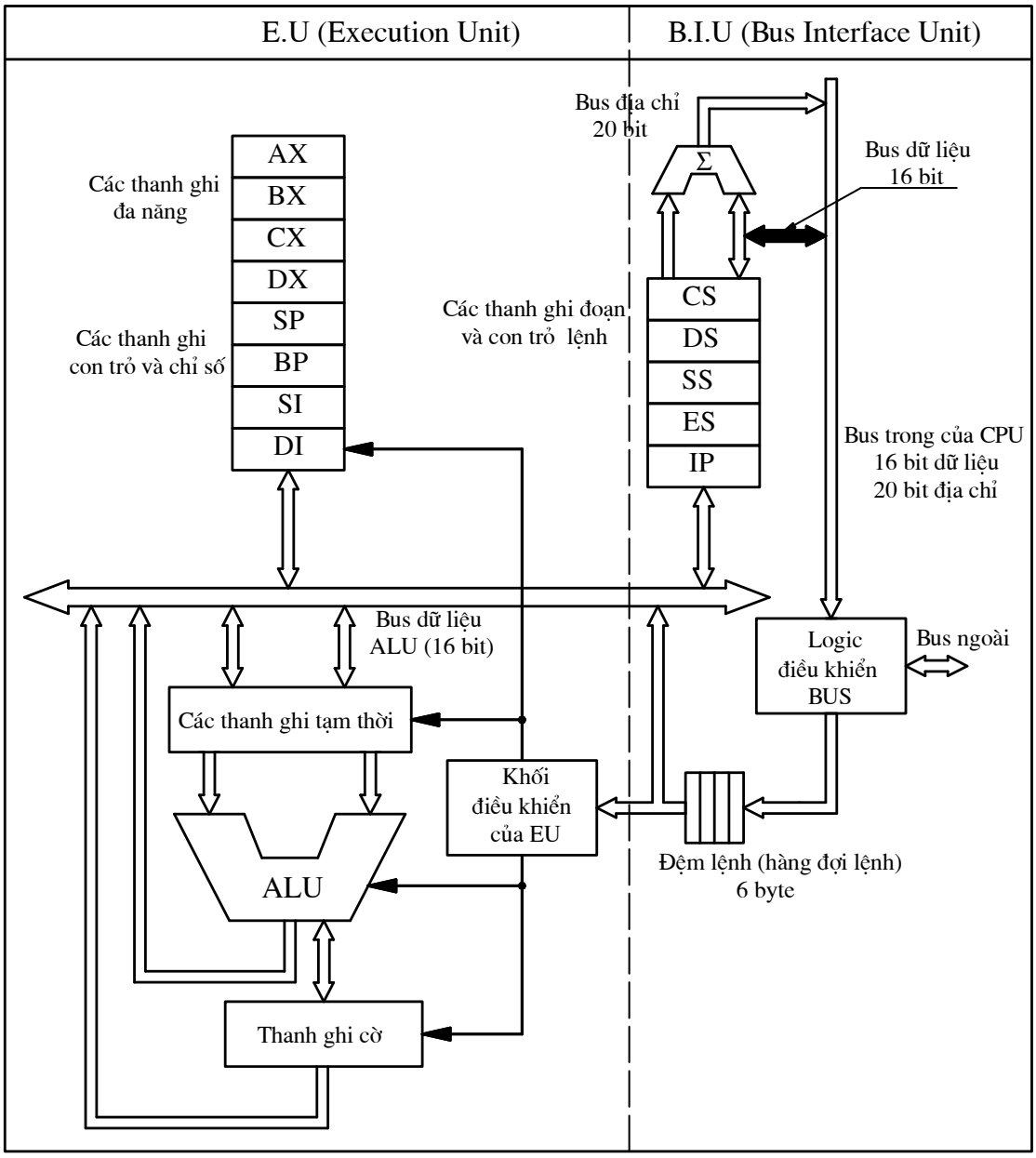
Sau đây là sơ đồ khối cấu trúc bên trong của vi xử lý 8086 (hình 2.1.2)

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

2.1.2.1. Các khối chức năng của CPU

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

BIU: Cung cấp các chức năng liên quan đến việc nhận lệnh và xếp hàng lệnh, lưu trữ các toán hạng và định vị các địa chỉ. Khối này cũng cung cấp các chức năng điều khiển BUS cơ sở. Trong hầu hết các trường hợp thời gian thực hiện lệnh và lấy lệnh và thực hiện lệnh là trùng nhau. Chính điều này làm tăng khả năng hoạt động của vi xử lý thông qua việc cải thiện Bus. Trong khi khối thực hiện lệnh đang bận rộn với lệnh hiện thời thì BIU đã có thể bắt đầu việc lấy các lệnh kế tiếp từ bộ nhớ và phần cuối của chúng được đặt trong một RAM nội bộ tốc độ cao được gọi là hàng đợi. Độ dài của hàng đợi này với vi xử lý 8086 là 6byte. Kỹ thuật hàng đợi lệnh cho phép BIU sử dụng bộ nhớ rất hiệu quả. BIU sẽ lấy mã lệnh trong bộ nhớ rồi đưa vào hàng đợi. Theo cách này BIU có thể cung cấp các lệnh một cách liên tục mà không độc chiếm BIU. Điều này làm giảm đáng kể thời gian chết trên Bus. Hàng đợi lệnh làm việc như một bộ đệm lệnh FIFO (First In First Out, vào trước ra trước).



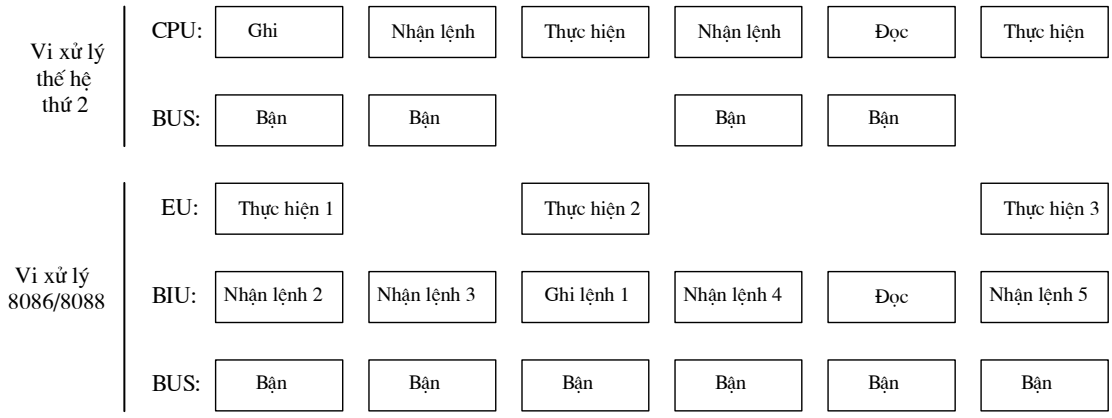
Hình 2.1.2. Sơ đồ khối cấu trúc bên trong của vi xử lý 8086

Nếu có sự vào/ra liên tục của dòng mã lệnh trong bộ đệm này thì có nghĩa là có sự phối hợp hoạt động hiệu qủa giữa hai khối EU và BIU theo cơ chế xử lý xen kẽ liên tục dòng mã lệnh để làm tăng tốc độ xử lý tổng thể. Kỹ thuật xen kẽ liên tục dòng mã lệnh sẽ không còn tác dụng tăng tốc độ xử lý của CPU nữa nếu nếu như trong đệm lệnh có chứa các mã lệnh của các lệnh CALL (gọi chương trình con) hoặc JMP (nhảy), bởi vì lúc gặp các lệnh này nội dung cũ của bộ đệm lệnh sẽ bị xoá và thay thế vào đó là nội dung mới được nạp bởi các lệnh mới do lệnh nhảy hoặc gọi quyết định. Việc này tiêu tốn nhiều thời gian hơn so với trường hợp trong đệm chỉ có mã lệnh của các lệnh tuần tự.

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

Thao tác được thực hiện trước tiên của EU là việc giải mã lệnh và khoảng thời gian này có vẻ như là lãng phí đối với CPU khi mà dường như chẳng có một hoạt động về mặt điện nào diễn ra ở trên Bus. Nhưng trong thực tế, chính khoảng thời gian này là khoảng thời gian được BIU khai thác để lấy trước các câu lệnh tiếp theo như đã được mô tả ở trên.

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



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

Chỉ dẫn lệnh:

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

Lệnh 2: Chỉ thực hiện lệnh

Lệnh 3: Đọc toán hạng và thực hiện

Khối EU được tạo thành từ các thanh ghi chung của vi xử lý 8086/8088. Như chúng ta đã biết, tất cả các thanh ghi và các đường truyền của dữ liệu nội bộ đều có độ rộng 16 bit. ở đây không có sự giao tiếp trực tiếp giữa EU và môi trường bên ngoài khi mà nó nhận các lệnh từ "hàng đợi" được BIU cung cấp (EU không nối với Bus hệ thống mà lấy lệnh từ hàng đợi). Khi một lệnh yêu cầu truy nhập tới bộ nhớ hoặc I/O, khối EU sẽ ra lệnh cho khối BIU truyền/nhận dữ liệu. Tất cả các dữ liệu được EU điều khiển đều là địa chỉ 16 bit. Nhưng thông qua việc di chuyển vị trí bộ nhớ được BIU thực hiện (định vị lại địa chỉ) khối EU có thể truy cập tới toàn bộ bộ nhớ 1 MB.

ALU: Đây chỉ là một tập con của EU, nhưng trong thực tế nó giống như một phần có cấu trúc độc lập, chịu trách nhiệm thực hiện các thao tác số học và các thao tác logic. Các toán hạng có thể là dữ liệu tức thì, dữ liệu từ các thanh ghi hoặc dữ liệu được lưu trữ trong bộ nhớ. Trong khi đó kết quả lại được định vị trong một thanh ghi hoặc trong bộ nhớ và 6 cờ trạng thái được cập nhật dựa trên kết quả của các thao tác này.

2.1.2.2. Các thanh ghi của CPU

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

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

Thanh ghi đoạn

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

- Chứa mã chương trình.
- Chứa dữ liệu và kết quả trung gian của chương trình.
- Tạo ra một vùng nhớ đặc biệt gọi là ngăn xếp (stack) dùng vào việc quản lý các thông số của bộ vi xử lý khi gọi chương trình con hoặc trở về từ chương trình con.

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

- CS (Code Segment): Thang ghi đoạn mã, chứa địa chỉ bắt đầu của đoạn chương trình (đoạn mã) mang những lệnh thực hiện được và

thông thường là một vùng nhớ chứa dữ liệu dạng hàng không thể thay đổi được hoặc là một vùng ROM/EPROM.

- DS (Data Segment): Thanh ghi đoạn dữ liệu, chứa địa chỉ bắt đầu của đoạn dữ liệu, bao gồm các tham số, các biến, các mảng số liệu…
- SS (Stack Segment): Thanh ghi đoạn ngăn xếp, chứa địa chỉ bắt đầu của mảng stack. Đây là một mảng của RAM, nơi mà dữ liệu tồn tại trong các thanh ghi được lưu trữ trong suốt quá trình ngắt.
- ES (Extra Segment): Thanh ghi đoạn dữ liệu phụ, chứa địa chỉ bắt đầu của vùng nhớ bổ sung.

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

Nội dung của thanh ghi đoạn cho phép ta xác định địa chỉ ô nhớ nằm ở đầu đoạn. Địa chỉ này gọi là địa chỉ cơ sở, địa chỉ của các ô nhớ khác nằm trong đoạn được tính bằng cách cộng thêm vào địa chỉ cơ sở một giá trị gọi là địa chỉ lệch hay độ lệch (offset). Độ lệch này được xác định bởi một thanh ghi 16 bit khác đóng vai trò thanh ghi lệch (offset register).

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

Địa chỉ logic = Thanh ghi đoạn: Thanh ghi lệch

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

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

20 bit địa chỉ vật lý = Thanh ghi đoạn x 16 + Thanh ghi lệch

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

CS:IP ~ F000Hx16 + FFF0H = F0000H + FFF0H = FFFF0H

Địa chỉ FFFF0H chính là địa chỉ khởi động của 8086/8088. Dấu ~ ở đây là để chỉ sự tương ứng. Từ nay khi nói đến địa chỉ của một ô nhớ ta có thể sử dụng cả địa chỉ logic lẫn địa chỉ vật lý vì bao giờ cũng có sự tồn tại tương ứng giữa hai loại địa chỉ này. Ta cũng cần chú ý rằng một giá trị địa

chỉ vật lý sẽ có nhiều cách tạo ra từ nhiều giá trị thanh ghi đoạn và thanh ghi lệch.

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

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

Các thanh ghi đa năng

Trong khối EU có 4 thanh ghi đa năng AX, BX, CX, DX. Điều đặc biệt là khi cần chứa dữ liệu 8 bit thì mỗi thanh ghi này có thể tách ra làm 2 thanh ghi 8 bit cao và thấp làm việc độc lập nhau, đó là các thanh ghi AH và AL, BH và BL, CH và CL, DH và DL. Mỗi thanh ghi có thể được dùng một cách vạn năng để chứa các loại dữ liệu khác nhau, nhưng cũng có những công việc đặc biệt nhất định chỉ thao tác với một vài thanh ghi nào đó và chính vì vậy các thanh ghi thường được gán cho những cái tên đặc biệt rất có ý nghĩa.

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

Các thanh ghi con trỏ và chỉ số

8086 có 3 thanh ghi con trỏ và 2 thanh ghi chỉ số 16 bit, các thanh ghi này (trừ IP) đều có thể được dùng như các thanh ghi đa năng, nhưng ứng dụng chính của mỗi thanh ghi là chúng được ngầm định như là thanh ghi lệch cho các đoạn tương ứng.

- IP (Instruction Pointer): Con trỏ lệnh, IP luôn trỏ vào lệnh tiếp theo sẽ được thực hiện nằm trong đoạn mã CS. Địa chỉ đầy đủ của lệnh tiếp theo này ứng với CS:IP và được xác định theo cách đã nói ở trên.

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

Thanh ghi cờ FR (Flag Register)

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

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

x: Không được định nghĩa

Các cờ cụ thể:

- ***Các cờ trạng thái***

- C hoặc CF (Carry Flag): Cờ nhớ CF = 1 khi có nhớ hoặc mượn từ MSB.
- F hoặc PF (Parity Flag): Cờ chẵn lẻ, phản ánh tính chẵn lẻ của tổng số bit 1 có trong kết quả. CF = 1 khi tổng số bit 1 trong kết quả là chẵn.
- A hoặc AF (Auxiliary carry Flag): cờ nhớ phụ, rất có ý nghĩa khi ta làm việc với các số BCD, AF = 1 khi có nhớ hoặc mượn từ một số BCD thấp (4 bit thấp) sang một số BCD cao (4bit cao).
- Z hoặc ZF (Zero Flag): Cờ rỗng, ZF = 1 khi kết qủa bằng 0.
- S hoặc SF (Sign Flag): Cờ dấu, SF = 1 khi kết quả âm.
- O hoặc OF (Overflow Flag): Cờ tràn, OF = 1 khi kết quả là số bù hai vượt ra ngoài giá trị biểu diễn của nó.

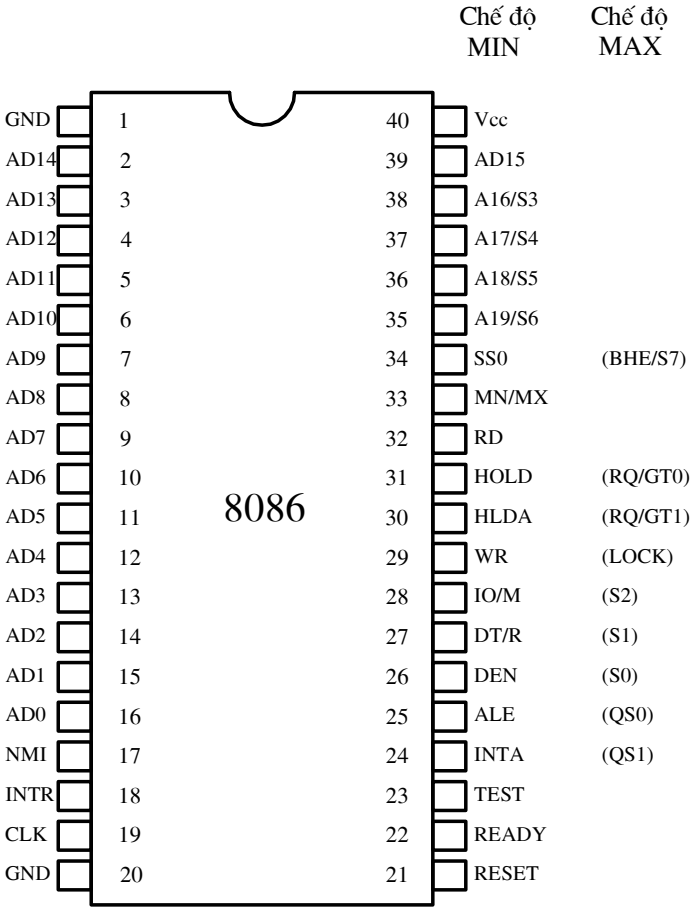
- ***Các cờ điều khiển (có thể lập hoặc xoá bằng các lệnh riêng)***

- T hoặc TF (Trap Flag): Cờ bẫy, TF = 1 thì CPU làm việc ở chế độ chạy từng lệnh (chế độ này dùng khi cần tìm lỗi chương trình).

- I hoặc IF (Interrupt enable Flag): Cờ cho phép ngắt, IF = 1 thì CPU cho phép các yêu cầu ngắt được tác động.
- D hoặc DF (Direction Flag): Cờ hướng, DF = 1 khi CPU làm việc với chuỗi ký tự theo ký tự từ phải sang trái (vì vậy D chính là cờ lùi).

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

Hình 2.1.3 là sơ đồ bố trí chân của vi xử lý 8086.



Hình 2.1.3. Sơ đồ chân của vi xử lý 8086

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

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

AD17/S4	AD16/S3	Truy nhập đến
0	0	Đoạn dữ liệu phụ (ES)
0	1	Đoạn ngăn xếp (SS)
1	0	Đoạn mã (CS) hoặc không đoạn nào
1 S6 luôn là 0	1	Đoạn dữ liệu (DS)

Bảng các bit trạng thái và việc truy nhập các thanh ghi đoạn

Bit S6=0 liên tục, bit S5 phản ánh giá trị bit IF của thanh ghi cờ, hai bit S3, S4 phối hợp với nhau để chỉ ra việc truy nhập các thanh ghi đoạn. Tín hiệu này chuyển sang trạng thái trở kháng cao khi Bus nội bộ ghi nhận tín hiệu treo.

- RD [O]: Đọc. Tín hiệu đọc cho biết bộ vi xử lý đang thực hiện đọc bộ nhớ hay đọc I/O phụ thuộc vào trạng thái chân S2. RD=0 thì bus dữ liệu sẵn sàng nhận số liệu từ bộ nhớ hoặc thiết bị ngoại vi. Tín hiệu này chuyển sang trạng thái trở kháng cao khi Bus nội bộ ghi nhận tín hiệu treo.
- READY [I]: Tín hiệu báo cho CPU biết tình trạng sẵn sàng của thiết bị ngoại vi hay bộ nhớ. Khi READY=1 thì CPU thực hiện đọc/ghi mà không cần chèn thêm các chu kỳ đợi. Ngược lại khi thiết bị ngoại vi hay bộ nhớ có tốc độ hoạt động chậm, chúng có thể đưa tín hiệu READY=0 để báo cho CPU biết mà chờ chúng, lúc này CPU tự động kéo dài thời gian thực hiện lệnh đọc/ghi bằng cách chèn thêm các chu kỳ đợi.
- INTR [I]: tín hiệu yêu cầu ngắt che được. Khi có yêu cầu ngắt mà cờ cho phép ngắt IF=1 thì CPU kết thúc lệnh đang làm dở, sau đó đi vào chu kỳ chấp nhận ngắt và đưa ra bên ngoài tín hiệu INTA=0.
- TEST [I]: Tín hiệu tại chân này được kiểm tra bởi lệnh WAIT (xem phần tập lệnh). Khi CPU thực hiện lệnh WAIT mà lúc đó tín hiệu TEST=1 nó sẽ chờ cho đến khi TEST=0 thì nó mới thực hiện lệnh tiếp theo.
- NMI [I]: Tín hiệu yêu cầu ngắt không che được. Tín hiệu này không chịu sự khống chế của cờ IF và nó sẽ được CPU nhận biết bằng tác động của sườn lên của xung yêu cầu ngắt. Nhận được yêu cầu này CPU kết thúc lệnh đang làm dở sau đó nó chuyển sang chương trình phục vụ ngắt kiểu INT 2.
- RESET [I]: Tín hiệu khởi động lại 8086. Khi RESET=1 kéo dài ít nhất trong thời gian 4 chu kỳ đồng hồ thì 8086 buộc phải khởi động lại: nó xoá các thanh ghi DS, ES, SS, IP, FR về 0 và bắt đầu thực hiện chương trình tại địa chỉ CS:IP=FFFF:0000H (cờ IF=0 để cấm các yêu cầu ngắt tác động vào CPU và cờ TF=0 để bộ vi xử lý không bị đặt trong chế độ chạy từng lệnh).

- CLK [I]: Tín hiệu đồng hồ (xung nhịp). Xung nhịp có độ rỗng là 77% và cung cấp nhịp làm việc cho CPU.
- Vcc [I]: Chân nguồn nuôi, tại đây CPU được cung cấp nguồn +5V±10%, 340mA.
- GND [O]: Hai chân nguồn để nối với điểm 0V của nguồn nuôi.
- MN/MX [I]: Chân điều khiển hoạt động của CPU theo chế độ MIN/MAX.

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

- IO/M [O]: Tín hiệu này phân biệt trong thời điểm đã định phần tử nào trong các thiết bị vào/ra (IO) hoặc bộ nhớ (M) được chọn làm việc với CPU. Trên bus địa chỉ lúc đó sẽ có các địa chỉ tương ứng của các thiết bị đó. Chân này ở trạng thái trở kháng cao khi μP chấp nhận treo.
- WR [O]: Xung cho phép ghi. Khi CPU đưa ra WR=0 thì trên bus dữ liệu các dữ liệu đã ổn định và chúng sẽ được ghi vào bộ nhớ hoặc thiết bị ngoại vi tại thời điểm đột biến WR=1. Chân này ở trạng thái trở kháng cao khi μP chấp nhận treo.
- ALE [O]: Xung cho phép chốt địa chỉ. Khi ALE=1 có nghĩa là trên bus dồn kênh AD có các địa chỉ của thiết bị vào/ra hay của ô nhớ. ALE không bao giờ ở trạng thái trở kháng cao, khi CPU bị treo thì ALE=0.
- DT/R [O]: Tín hiệu điều khiển các đệm hai chiều của bus dữ liệu để chọn chiều chuyển trên bus D. Chân này ở trạng thái trở kháng cao khi μP chấp nhận treo.
- DEN [O]: Tín hiệu báo cho bên ngoài biết là lúc này trên bus dồn kênh AD có dữ liệu ổn định. Chân này ở trạng thái trở kháng cao khi μP chấp nhận treo.
- HOLD [I]: Tín hiệu yêu cầu treo CPU để mạch ngoài thực hiện việc trao đổi dữ liệu với bộ nhớ bằng cách thâm nhập trực tiếp (Direct Memory Access, DMA). Khi HOLD=1, CPU sẽ tự tách ra khỏi hệ thống bằng cách treo bus A, bus D, bus C của nó (các bus ở trạng thái trở kháng cao) để bộ điều khiển DMA (DMA Controller, DMAC) có thể lấy được quyền điều khiển hệ thống để làm các công việc trao đổi dữ liệu.
- HLDA [O]: Báo tín hiệu cho bên ngoài biết yêu cầu treo CPU để dùng các bus đã được chấp nhận và CPU 8086 đã treo các bus A, bus D và một số tín hiệu của bus C.

- SS0 [O]: Tín hiệu trạng thái. Tín hiệu này giống như S0 ở chế độ MAX và được dùng kết hợp với IO/M, DT/R để giải mã các chu kỳ hoạt động của bus.

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

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

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

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

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

Bảng các tín hiệu điều khiển của 8288

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

- LOCK [O]: Tín hiệu do CPU đưa ra để cấm các bộ vi xử lý khác dùng bus trong khi nó đang thi hành một lệnh nào đó đặt sau lệnh LOCK (xem phần tập lệnh vi xử lý 8086).
- QS0 và QS1 [O]: Tín hiệu thông báo các trạng thái khác nhau của đệm lệnh (hàng đợi).

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

Bảng các trạng thái của đệm lệnh

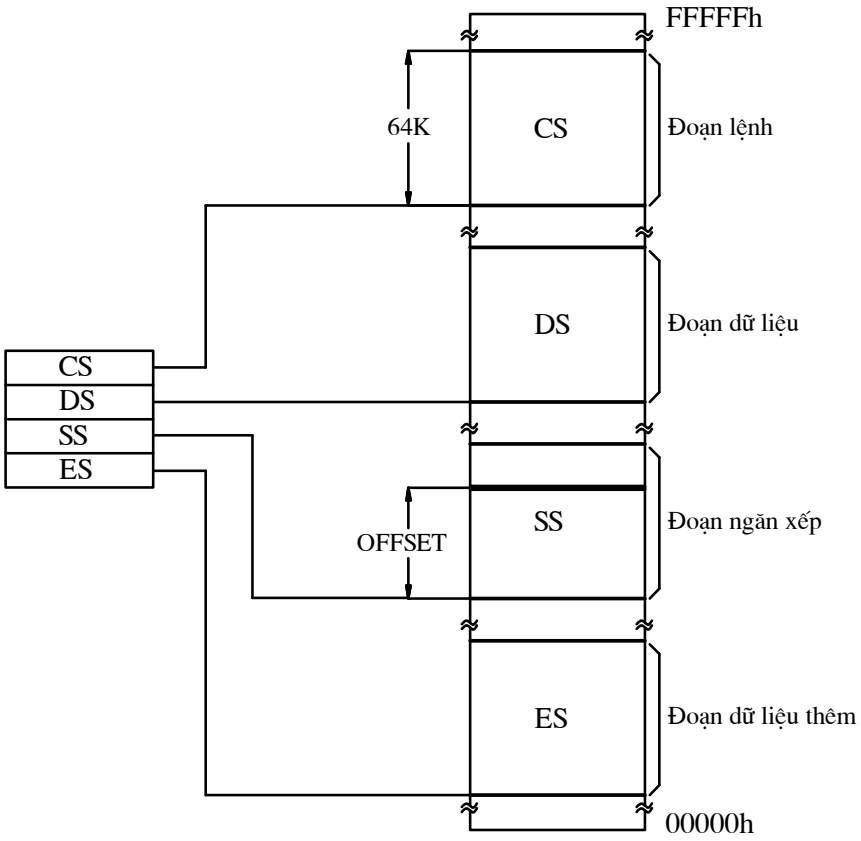
2.1.4. TỔ CHỨC BỘ NHỚ CỦA VI XỬ LÝ 8086

Vi xử lý 8086 có 20 đường địa chỉ và do đó có thể địa chỉ hoá được 220 ô nhớ dữ liệu 8 bit kéo dài từ địa chỉ 00000h đến FFFFFh. Bộ nhớ logic được chia thành đoạn lệnh (CS), đoạn dữ liệu (DS), đoạn ngăn xếp (SS), đoạn thêm (ES) với 64 Kbyte cho mỗi đoạn. Tổ chữ bộ nhớ của 8086 được biểu diễn như hình 2.1.4

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

Một số vị trí trong bộ nhớ được dùng cho các thao tác đặc biệt. Các vị trí có địa chỉ từ FFFF0h – FFFFFh được dự tính cho các thao tác nhảy đến một chương trình khởi tạo hệ thống. Khi được RESET CPU sẽ thực hiện lệnh đầu tiên ở địa chỉ FFFF0h mà ở đó thường chứa một lệnh nhảy.

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



Hình 2.1.4. Tổ chức bộ nhớ của 8086

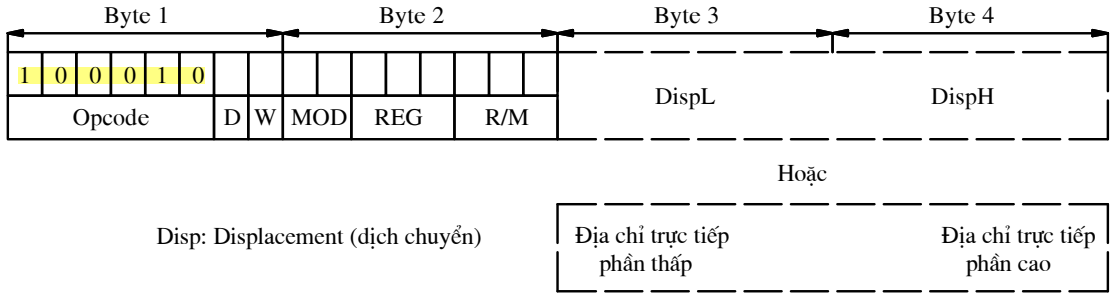
2.2. CÁC CHẾ ĐỘ ĐỊNH ĐỊA CHỈ CỦA BỘ VI XƯ LÝ 8086

Trước khi đi vào các chế độ địa chỉ của Vi xử lý 8086 ta nói qua về cách mã hoá lệnh trong vi xử lý 8086.

Lệnh của bộ vi xử lý được ghi bằng các ký tự dưới dạng gợi nhớ để người sử dụng dễ nhận biết. Đối với bản thân bộ vi xử lý thì lệnh cho nó được mã hoá dưới dạng các số 0 và 1 (còn gọi là mã máy) vì đó là dạng biểu diễn thông tin duy nhất mà máy có thể hiểu được. Vì lệnh cho bộ vi xử lý được cho dưới dạng mã nên sau khi nhận lệnh, bộ vi xử lý phải thực hiện giải mã lệnh rồi sau đó mới thực hiện lệnh

Một lệnh có thể có độ dài một vài byte tuỳ theo bộ vi xử lý. Đối với vi xử lý 8086 một lệnh có độ dài từ 1 đến 6 byte. Ta sẽ dùng lệnh MOV để giải thích cách ghi lệnh nói chung của 8086.

Hình 2.2 biểu diễn dạng thức các byte dùng để mã hoá lệnh MOV.



Hình 2.2. Dạng thức các byte mã lệnh của lệnh MOV

1 word = 2 byte

Từ đây ta thấy để mã hoá lệnh MOV cần ít nhất 2 byte. Trong đó 6 bit đầu dùng để chứa mã lệnh, 6 bit này luôn là 100010. đối với các thanh ghi đoạn thì điều này lại khác. Bit W dùng để chỉ ra rằng một byte (W=0) hoặc một từ (W=1) sẽ được chuyển đi. Trong thao tác chuyển dữ liệu, một toán hạng luôn bắt buộc phải là thanh ghi. Bộ vi xử lý sử dụng 2 hoặc 3 bit (REG) để mã hoá các thanh ghi trong CPU như sau:

word

Thanh ghi		Mã
W = 1	W = 0	
AX	AL	000
BX	BL	011
CX	CL	001
DX	DL	010
SP	AH	100
DI	BH	111
BP	CH	101
SI	DH	110

Thanh ghi đoạn	Mã
CS	01
DS	11
ES	00
SS	10

direction

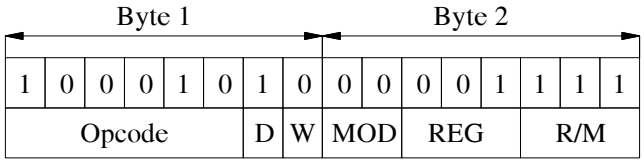
Bit D dùng để chỉ hướng đi của dữ liệu. D = 1 thì dữ liệu đến thanh ghi, D = 0 thì dữ liệu đi ra từ thanh ghi.

Hai bit MOD (chế độ) cùng với ba bit R/M (thanh ghi/bộ nhớ) tạo ra 5 bit dùng để chỉ ra chế độ địa chỉ cho các toán hạng của lệnh. Bảng 2.2 cho ta thấy cách mã hoá các chế độ địa chỉ.

Bảng 2.2 Phối hợp MOD và R/M để tạo ra các chế độ địa chỉ

MOD / R/M	00	01	10	11	
				W=0	W=1
000	[BX+SI]	[BX+SI]+d8	[BX+SI]+d16	AL	AX
001	[BX+DI]	[BX+DI]+d8	[BX+DI]+d16	CL	CX
010	[BP+SI]	[BP+SI]+d8	[BP+SI]+d16	DL	DX
011	[BP+DI]	[BP+DI]+d8	[BP+DI]+d16	BL	BX
100	[SI]	[SI]+d8	[SI]+d16	AH	SP
101	[DI]	[DI]+d8	[DI]+d16	CH	BP
110	D16 (địa chỉ trực tiếp)	[BP]+d8	[BP]+d16	DH	SI
111	[BX]	[BX]+d8	[BX]+d16	BH	DI

Ví dụ 1: MOV CL, [BX]



Mã lệnh MOV: 100010

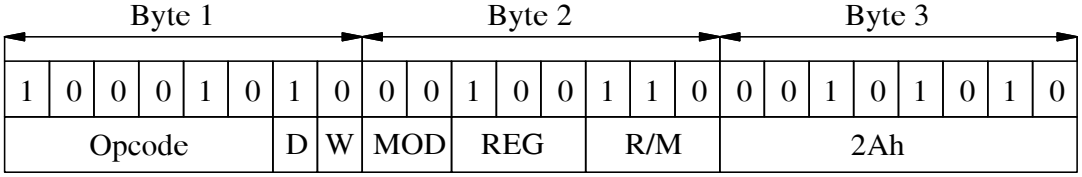
D = 1: Chuyển tới thanh ghi

W = 0: Chuyển 1 byte

MOD: ở chế độ 00 và R/M là 111

REG: 001 mã hoá CL

Ví dụ 2: MOV AH, 2Ah



Mã lệnh MOV: 100010

D = 1: Chuyển tới thanh ghi

W = 0: Chuyển 1 byte

MOD: ởy chế độ 00 và R/M là 110: Địa chỉ trực tiếp

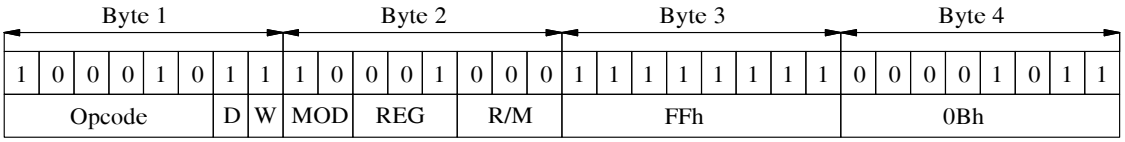
REG: 100 mã hoá AH

2Ah = 00101010 dữ liệu cần chuyển tới AH

Ví dụ 3: MOV CX, [BX][SI]+DATA

DATA là một biến trong bộ nhớ, đó là địa chỉ lệch và là một hằng (ví dụ như 0BFF).

Lệnh này sẽ sử dụng 4 byte tổ chức như sau:



Mã lệnh MOV: 100010

D = 1: Chuyển tới thanh ghi

W =1: Chuyển 1 Word

MOD: ở chế độ 10 (offset 16 bit) và R/M là 000 (sử dụng thanh ghi cơ sở BX và thanh ghi chỉ số SI).

REG: 001 mã hoá thanh ghi CX.

Như vậy trong ký hiệu nhị phân và hexa ta có.

Byte 1	Byte 2	Byte 3	Byte 4
10001011	10001000	11111111	00001011
8Bh	88h	FFh	0Bh

Bây giờ chúng ta sẽ đi giới thiệu các chế độ địa chỉ của 8086.

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

- Chế độ địa chỉ thanh ghi.
- Chế độ địa chỉ tức thì.
- Chế độ địa chỉ trực tiếp.
- Chế độ địa chỉ gián tiếp qua thanh ghi.
- Chế độ địa chỉ tương đối cơ sở.
- Chế độ địa chỉ tương đối chỉ số.
- Chế độ địa chỉ tương đối cơ sở chỉ số.
- Chế độ địa chỉ chuỗi (String) – mảng.
- Chế độ địa chỉ cổng (Port).

2.2.1. CHẾ ĐỘ ĐỊA CHỈ THANH GHI

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

Ví dụ:

```
MOV     BX, DX  ;copy noi dung DX vao BX
ADD     AX, BX  ;cong AX voi BX roi ghi ket qua
                ;vao AX
```

2.2.2. CHẾ ĐỘ ĐỊA CHỈ TỨC THÌ

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

Ví dụ:

```
MOV     CL, 100         ;chuyen 100 vao CL.
MOV     AX, 0BC8h       ;chuyen 0BC8h vao AX de roi
MOV     DS, AX          ;copy noi dung AX vao DS
(vi khong duoc chuyen truc tiep vao thanh ghi doan)
```

emu8086 -> emulate -> run -> aux -> memory -> "enter address" -> update

```
MOV     [BX], 20        ;chuyen 20 vao o nho tai
                        ;dia chi DS:BX.
```

2.2.3. CHẾ ĐỘ ĐỊA CHỈ TRỰC TIẾP

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

Ví dụ:

```
MOV     AL, [0243H]     ;chuyen noi dung o nho
                        ;DS:0243 vao AL
MOV     [4320], CX      ;chuyen noi dung CX vao hai
        ;o nho lien tiep DS:4320 va DS:4321
```

2.2.4. CHẾ ĐỘ ĐỊA CHỈ GIÁN TIẾP QUA THANH GHI

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

Ví dụ:

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

2.2.5. CHẾ ĐỘ ĐỊA CHỈ TƯƠNG ĐỐI CƠ SỞ

Trong chế độ địa chỉ này các thanh ghi cơ sở như BX và BP và các hằng số biểu diễn các giá trị dịch chuyển được dùng để tính địa chỉ hiệu dụng của toán hạng trong các vùng nhớ DS và SS.

Ví dụ:

```
MOV     CX, [BX]+10     ;copy noi dung hai o nho
        ;lien tiep co dia chi DS:BX+10 va
        ;DS:BX+11 vao CX
MOV     CX, [BX+10];cach viet khac cua lenh tren
MOV     CX, 10+[BX];cach viet khac cua lenh tren
MOV     AL, [BP]+5      ;chuyen noi dung o nho co
        ;dia chi SS:BP+5 vao AL
```

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

BX+10, BP+5 gọi là địa chỉ hiệu dụng.

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

2.2.6. CHẾ ĐỘ ĐỊA CHỈ TƯƠNG ĐỐI CHỈ SỐ

Trong chế độ địa chỉ này các thanh ghi chỉ số như SI và DI và các hằng số biểu diễn các giá trị dịch chuyển được dùng để tính địa chỉ hiệu dụng của toán hạng trong các vùng nhớ DS.

Ví dụ

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

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

Kết hợp hai chế độ địa chỉ chỉ số và cơ sở ta có chế độ địa chỉ chỉ số cơ sở. Trong chế độ này ta dùng cả hai thanh ghi cơ sở lẫn thanh ghi chỉ số để tính địa chỉ của toán hạng. Nếu ta dùng thêm cả thành phần biểu diễn sự dịch chuyển của địa chỉ thì ta có chế độ địa chỉ tổng hợp nhất: Chế độ địa chỉ tương đối chỉ số cơ sở.

Ví dụ:

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

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

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

	[BX]+[SI]+Disp	DS
	[BP]+[DI]+Disp	SS
	[BP]+[SI]+Disp	SS

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

2.2.8. CHẾ ĐỘ ĐỊA CHỈ CHUỖI (STRING) – MẢNG

Một chuỗi (string) là một dãy các byte hoặc word liên tiếp trong bộ nhớ. Các lệnh thao tác với chuỗi không sử dụng bất kỳ một chế độ địa chỉ nào ở trên. Một chuỗi có thể có độ dài tối đa lên tới 64K-bytes (một segments). Chế độ địa chỉ chuỗi sử dụng các thanh ghi SI, DI, DS và ES. Với tất cả các lệnh thao tác chuỗi đều sử dụng SI để trỏ vào byte đầu tiên của chuỗi nguồn và DI trỏ vào byte đầu tiên của chuỗi đích.

Ví dụ: Giả sử: DS=1000h, ES=2000h, SI=10h, DI=20h)

```
MOVSB   ;Sao chép chuỗi từ 10010h đến 20020h
```

2.2.9. CHẾ ĐỘ ĐỊA CHỈ CỔNG (PORT)

Trong họ vi xử lý 80x86 của Intel có không gian địa chỉ cho bộ nhớ và cổng vào/ra là tách biệt nhau. Không gian địa chỉ cổng có thể lên đến 65536 cổng (64K-ports).

Địa chỉ của một cổng có thể được xác định bởi một hằng giá trị kiểu byte (phạm vi = 0..255)

Ví dụ:

```
IN AL, 40h      ;Đọc cổng – sao chép nội dung tại
                ;cổng có địa chỉ 40h vào thanh ghi AL
OUT 80h, AL     ;Ghi cổng – gửi dữ liệu trong
                ;thanh ghi AL tới cổng có địa chỉ 80h
```

Địa chỉ của cổng cũng có thể được xác định gián tiếp qua thanh ghi (Khi này phạm vi tối đã sẽ là 65536 cổng).

Ví dụ:

```
IN AL, DX       ;Đọc cổng có địa chỉ là nội dung
                ;của thanh ghi DX
OUT DX, AX      ;Ghi một word trong AX tới cổng có
                ;địa chỉ là nội dung của thanh ghi DX.
```

2.3. TẬP LỆNH CỦA VI XỬ LÝ 8086

2.3.1. GIỚI THIỆU CHUNG

Tập lệnh của họ vi xử lý 80x86 đảm bảo tương thích thế hệ sau với thế hệ trước. điều đó có nghĩa là các chương trình viết cho 8086 vẫn chạy được trên các bộ vi xử lý mới hơn mà không phải thay đổi (không đảm bảo thứ tự ngược lại). Tập lệnh của một bộ vi xử lý thường có rất nhiều lệnh (hàng trăm lệnh), vì thế

mà việc tiếp cận và làm chủ chúng là trương đối khó khăn. Có nhiều cách trình bày tập lệnh của bộ vi xử lý: Trình bày theo nhóm lệnh hoặc theo thứ tự abc. Để có thể nhanh chóng và dễ dàng sử dụng các lệnh cơ bản và lập trình được ngay, ta sẽ tiếp cận tập lệnh của bộ vi xử lý theo nhóm các thao tác cơ bản trong quá trình xử lý và điều khiển. Với mỗi thao tác nói trên, ta làm quen với một vài lệnh tiêu biểu (độc giả có thể tra cứu thêm các lệnh khác trong phần phụ lục). Các chức năng cơ bản của một bộ vi xử lý thường gồm:

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

2.3.2. TẬP LỆNH CỦA VI XỬ LÝ 8086

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

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

   Dạng lệnh: LDS Đích, Nguồn

   Trong đó:

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

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

   Các cờ bị thay đổi: không.

   Ví dụ:

   ```
   LDS  SI, STR_PTR
   ```

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

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

   Dạng lệnh: LEA Đích, Nguồn

   Trong đó:

   - Đích là một trong các thanh ghi: BX, CX, DX, BP, SI, DI.

- Nguồn là tên biến trong đoạn DS được chỉ rõ trong lệnh hoặc ô nhớ cụ thể.

Đích←Địa chỉ lệch của Nguồn, hoặc

Đích←Địa chỉ hiệu dụng của Nguồn

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

Các cờ bị thay đổi: không.

Ví dụ:

```
LEA   DX, Label      ;nap dia chi lech cua Label
      ;vao DX
LEA   CX, [BX][DI]   ;nap vao CX dia chi hieu
      ;dung do BX va DI chi ra EA=BX+DI
```

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

Dạng lệnh: LES Đích, Nguồn

Trong đó:

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

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

Các cờ bị thay đổi: không.

Ví dụ:

```
LDS   DI, [BX]
```

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

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

Dạng lệnh: MOV Đích, Nguồn

Mô tả: Đích←Nguồn

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

Các cờ bị thay đổi: không.

Ví dụ:

```
MOV   AL, AH          ;AL←AH
MOV   CX, 50          ;CX←50
MOV   DL, [SI]        ;DL←{DS:SI}
```

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

Dạng lệnh:

MOVS Chuỗi_đích, Chuỗi_Nguồn

MOVSB

MOVSW

Mô tả:

Phần tử chuỗi_đích←phần tử chuỗi_Nguồn

Lệnh này dùng để chuyển từng byte hay từng từ của chuỗi Nguồn sang chuỗi đích, trong đó.

- DS:SI là địa chỉ của phần tử trong chuỗi Nguồn.
- ES: DI là địa chỉ của phần tử trong chuỗi đích.
- Sau mỗi lần chuyển thì SI←SI±1, DI←DI±1 hoặc SI←SI±2, DI←DI±2 một cách tự động tuỳ thuộc cờ hướng DF là 0 hay 1 và chuỗi là chuỗi byte hay từ.

Có hai cách để chỉ ra chuỗi là chuỗi byte hay chuỗi từ. Cách đầu tiên là ta khai báo chuỗi_đích, chuỗi_Nguồn là loại gì ngay từ đầu chương trình. Cách thứ hai là ta thêm vào lệnh MOVS đuôi "B" cho chuỗi byte hoặc đuôi "W" cho chuỗi từ (xem rõ trong lệnh COMPS).

Các cờ bị thay đổi: không.

Ví dụ:

```
CLD             ;xoa co huong lam viec voi
                ;chuoi theo chieu →
MOV  DI, OFFSETChuoi_dich  ;lay  dia  chi  lech
                ;cua chuoi_dich tai ES vao DI
MOV  SI, OFFSET Chuoi_goc  ;lay  dia  chi  lech
                ;cua chuoi_goc tai DS vao SI
MOVSB           ;chuyen 1byte, SI va DI
                ;tang them 1
```

6. **OUT** – Output a byte or a work to a port.

Dạng lệnh: OUT Port, Acc

Mô tả: Acc→{Port}

Trong đó {port} là dữ liệu của cổng có địa chỉ port. Port là địa chỉ 8 bit của cổng, nó có thể là các giá trị trong khoảng 00...FFH. Như vậy có thể có các khả năng sau đây.

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

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

OUT DX, Acc

Các cờ bị thay đổi: không.

Ví dụ:

```
OUT  45H, AL;dua du lieu tu AL ra cong 45H
MOV  DX, 0          ;xoa DX
MOV  DX, 00FFH      ;nap dia chi cong vao DX
OUT  DX, AX ;dua du lieu tu AX ra 00FFH
```

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

Dạng lệnh: POP Đích

Mô tả:

Đích←{SP}

SP←SP+2

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

Các cờ bị thay đổi: không.

Ví dụ:

```
POP  DX;lay 2 byte tu dinh ngan xep dua vao DX
```

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

Dạng lệnh: POPF

Mô tả:

FR←{SP}

SP←SP+2

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

Các cờ bị thay đổi: tất cả các cờ.

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

Dạng lệnh: PUSH Nguồn

Mô tả:

SP←SP-2

Nguồn→{SP}

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

Các cờ bị thay đổi: không.

Ví dụ:

```
PUSH    BX    ;cat BX vao ngan xep tai vi tri do
              ;SP chi ra
```

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

Dạng lệnh: PUSHF

Mô tả:

SP←SP-2

FR→{SP}

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

Các cờ bị thay đổi: không.

Ví dụ:

```
PUSH    BX    ;cat BX vao ngan xep tai vi tri do
              ;SP chi ra
```

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

Dạng lệnh: XCHG Đích, Nguồn

Mô tả: Đích ↔ Nguồn

Toán hạng đích và Nguồn phải có cùng độ dài, không được đồng thời là 2 ô nhớ cũng không được là thanh ghi đoạn. Sau lệnh XCHG toán hạng này chứa nội dung cũ của toán hạng kia và ngược lại.

Các cờ bị thay đổi: không.

Ví dụ:

```
XCHG  AH, AL ;trao noi dung AH va AL
XCHG  AX, BX ;trao noi dung AX va BX
```

2.3.2.2. Nhóm các lệnh tính toán số học

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

Dạng lệnh: ADC Đích, Nguồn

Mô tả: Đích ← Đích + Nguồn + CF

Cộng hai toán hạng Đích và Nguồn với cờ CF kết quả lưu vào Đích.

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

Ví dụ:

```
ADC   AL, 74H          ;AL←AL+74+CF
ADC   CL, BL           ;CL←CL+BL+CF
ADC   DL, [SI]         ;DL←DL+(DS:SI)+CF
```

13. ADD – Add (cộng hai toán hạng)

Dạng lệnh: ADD Đích, Nguồn

Mô tả: Đích ← Đích + Nguồn

Cộng hai toán hạng đích và Nguồn kết quả lưu vào đích.

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

Ví dụ:

```
ADD   DX, CX           ;DX←DX+CX
ADD   AX, 400          ;AX←AX+400
```

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

Dạng lệnh: DEC Đích

DEC trừ toán hạng Đích đi 1. Toán hạng Đích có thể là byte hay word.

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

Ví dụ:

```
MOV   BX, 1200H        ;chuyen 1200H vao BX
DEC   BX               ;BX=11FFH
```

15. DIV – Division (chia không dấu)

Dạng lệnh: DIV Nguồn

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

- Nếu Nguồn là là số 8 bit: AX/Nguồn, thương để vào AL, số dư để vào AH
- Nếu Nguồn là số 16 bit: DXAX/Nguồn, thương để vào AX, số dư để vào DX

Nếu thương không phải là số nguyên nó được làm tròn theo số nguyên sát dưới. Nếu Nguồn bằng 0 hoặc thương thu được lớn hơn FFH hoặc FFFFH (tuỳ theo độ dài của toán hạng Nguồn) thì 8086 thực hiện lệnh ngắt INT 0.

Các cờ bị thay đổi: không.

Ví dụ:

```
MOV  AX, 0033H      ;chuyen 0033H vao AX
MOV BL, 25
DIV  BL             ;AL=02H va AH=01H
```

16. IDIV – Integer Division (chia có dấu)

Dạng lệnh: IDIV Nguồn

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

- Sau phép chia AL (AX) chứa thương số (số có dấu), AH (DX) chứa số dư (số có dấu).
- Dấu của số dư trùng với dấu của số bị chia.
- Nếu Nguồn = 0 hoặc thương nằm ngoài dải -128…+127 hoặc -32768…32767 (tuỳ theo độ dài Nguồn) thì 8086 thực hiện lệnh ngắt INT 0.

Các cờ bị thay đổi: không.

Ví dụ:

```
IDIV CL
IDIV BX
```

17. IMUL – Integer Multiplication (nhân có dấu)

Dạng lệnh: IMUL Nguồn

Tuỳ theo độ dài toán hạng Nguồn ta có 2 trường hợp bố trí phép nhân, chỗ để ngầm định cho số bị nhân và kết quả.

- Nếu Nguồn là số có dấu 8 bit: ALxNguồn. Sau khi nhân AX←tích.
- Nếu Nguồn là số có dấu 16 bit: AXxNguồn. Sau khi nhân DXAX←tích.

Nếu tích thu được nhỏ, không đủ lấp đầy hết được các chỗ dành cho nó thì các bit không dùng đến được thay bằng bit dấu.

Nếu byte cao (hoặc 16 bit cao) của 16 (hoặc 32) bit kết quả chỉ chứa các bit dấu thì CF = OF = 0.

Nếu byte cao (hoặc 16 bit cao) của 16 (hoặc 32) bit kết quả chứa một phần kết quả thì CF = OF = 1.

Như vậy CF và OF báo cho ta biết kết quả cần độ dài bao nhiêu.

Các cờ bị thay đổi: CF, OF.

Ví dụ:

```
IMUL CL
```

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

Dạng lệnh: IN Acc, địa_chỉ_cổng

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

Các cờ bị thay đổi: không.

Ví dụ:

```
IN   AL, 45H        ;doc mot byte tu mot cong
     ;duoc xac dinh trong che do tuc thi
IN   AX, 0046H      ;doc hai byte tu mot cong
     ;duoc xac dinh trong che do tuc thi
IN   AX, DX ;doc mot tu tu mot cong dang bien
```

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

Dạng lệnh: INC đích

Mô tả: Đích ← Đích + 1

Lệnh này tăng đích lên 1, tương đương với việc ADD đích, 1 nhưng chạy nhanh hơn.

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

Ví dụ:

```
INC  AL
INC  BX
```

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

Dạng lệnh: MUL Nguồn

Thực hiện phép nhân không dấu với toán hạng Nguồn (ô nhớ hoặc thanh ghi) với thanh ghi tổng.

- Nếu Nguồn là số 8 bit: AL*Nguồn. Số bị nhân phải là số 8 bit đặt trong AL, sau khi nhân tích lưu vào AX
- Nếu Nguồn là số 16 bit: AX*Nguồn. Số bị nhân phải là số 16 bit đặt trong AX, sau khi nhân tích lưu vào DXAX.

Nếu byte cao (hoặc 16 bit cao) của 16 (hoặc 32) bit kết quả chứa 0 thì CF=OF=0.

Các cờ bị thay đổi: CF, OF.

Ví dụ:

```
MUL  CX      ;AXxCX → DXAX
MUL  BL      ;ALxBL → AX
```

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

Dạng lệnh: NEG Đích

Mô tả: Đích←0-Đích

NEG lấy 0 trừ cho đích (có thể là 1 byte hoặc 1 từ) và trả lại kết quả cho toán hạng đích, nếu ta lấy bù hai của -128 hoặc -32768 ta sẽ được kết quả không đổi nhưng OF=1 để báo là kết quả bị tràn vì số dương lớn nhất biểu diễn được là +127 và +32767.

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

Ví dụ:

```
NEG   AL                ;AL←0-(AL)
```

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

Dạng lệnh: SBB Đích, Nguồn

Mô tả: Đích←Đích-Nguồn-CF

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

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

Ví dụ:

```
SBB   AL, 78H           ;AL←AL-78H-CF
SBB   BL, CL            ;BL←BL-CL-CF
SBB   DL, [SI]          ;DL←DL-{DS:SI}-CF
```

23. SUB – Substract (trừ hai toán hạng)

Dạng lệnh: SUB Đích, Nguồn

Mô tả: Đích←Đích - Nguồn

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

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

Ví dụ:

```
SUB   AL, 78H           ;AL←AL-78H
SUB   BL, CL            ;BL←BL-CL
SUB   DL, [SI]          ;DL←DL-{DS:SI}
```

2.3.2.3. Nhóm các lệnh tính toán logic

24. AND (phép và logic)

Dạng lệnh: AND Đích, Nguồn

Mô tả: Đích ← Đích ^ Nguồn

Thực hiện phép và logic hai toán hạng và lưu kết quả vào toán hạng đích. Người ta thường sử dụng để che đi/giữ lại một vài bit nào đó của một

toán hạng bằng cách nhân logic toán hạng đó với toán hạng tức thì có các bit 0/1 ở các vị trí cần che đi/giữ lại tương ứng.

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

Ví dụ:

```
AND   DX, CX  ;DX←DX AND CX theo tung bit
AND   AL, 0FH         ;che 4 bit cao cua AL
```

25. NOT – Logical Negation (phủ định logic)

Dạng lệnh: NOT Đích

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

Các cờ bị thay đổi: không.

Ví dụ:

```
MOV   AL, 02H         ;AL=(0000 0010)B
NOT   AL              ;AL=(1111 1101)B
```

26. OR – Logic OR (phép hoặc logic)

Dạng lệnh: OR Đích, Nguồn

Mô tả: Đích = Đích ∨ Nguồn

Toán hạng Đích và Nguồn phải chứa dữ liệu cùng độ dài và không được phép đồng thời là hai ô nhớ và cũng không được là thanh ghi đoạn. Phép OR thường dùng để lập một vài bit nào đó của toán hạng bằng cách cộng logic toán hạng đó với các toán hạng tức thời có các bit 1 tại vị trí tương ứng cần thiết lập.

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

Ví dụ:

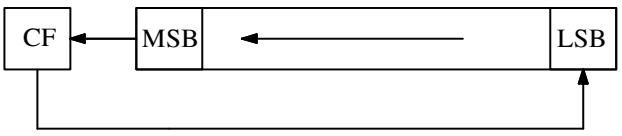
```
OR    AX, BX        ;AX←AX∨BX theo tung bit
OR    CL, 30H       ;lap bit b4 va b5 cua CL len
1
```

2.3.2.4. Nhóm các lệnh dịch, quay toán hạng

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

Dạng lệnh: RCL Đích, CL

Mô tả:



Lệnh này để quay toán hạng sang trái thông qua cờ CF, CL phải được chứa sẵn số lần quay. Trong trường hợp quay 1 lần có thể viết RCL Đích, 1

Nếu số lần quay là 9 thì toán hạng không đổi vì cặp CF và toán hạng quay đúng một vòng (nếu toán hạng đích là 8 bit).

Sau lệnh RCL cờ CF mang giá trị cũ của MSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được xác định sau nhiều lần quay.

Các cờ bị thay đổi: CF, OF.

Ví dụ:

```
MOV   CL, 3   ;so lan quay la 3
RCL   AL, CL
```

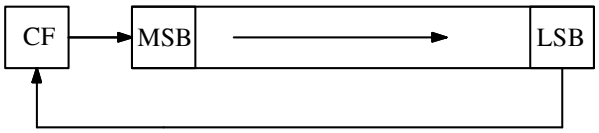
Trước khi thực hiện lệnh: AL = 01011110, CF = 0.

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

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

Dạng lệnh: RCR Đích, CL

Mô tả:



Lệnh này để quay toán hạng sang phải thông qua cờ CF, CL phải được chứa sẵn số lần quay. Trong trường hợp quay 1 lần có thể viết RCR Đích, 1

Nếu số lần quay là 9 thì toán hạng không đổi vì cặp CF và toán hạng quay đúng một vòng (nếu toán hạng đích là 8 bit).

Sau lệnh RCR cờ CF mang giá trị cũ của LSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được xác định sau nhiều lần quay.

Các cờ bị thay đổi: CF, OF.

Ví dụ:

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

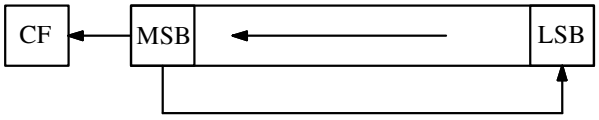
Trước khi thực hiện lệnh: AL = 11000010, CF = 1.

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

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

Dạng lệnh: ROL Đích, CL.

Mô tả:



Lệnh này dùng để quay vòng toán hạng sang trái, MSB được đưa sang cờ CF và LSB. CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp

quay 1 lần có thể viết ROL Đích, 1. Nếu số lần quay là 8 (CL=8) thì toán hạng không đổi vì toán hạng quay đúng một vòng (nếu toán hạng đích là 8 bit), còn nếu CL=4 thì 4 bit cao đổi chỗ cho 4 bit thấp.

Sau lệnh ROL cờ CF mang giá trị cũ của MSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được xác định sau nhiều lần quay. Lệnh này thường dùng để tạo cờ CF từ giá trị của MSB làm điều kiện cho lệnh nhảy có điều kiện.

Các cờ bị thay đổi: CF, OF.

Ví dụ:

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

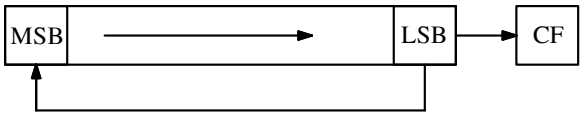
Trước khi thực hiện lệnh: AL = 11001100, CF = 1

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

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

Dạng lệnh: ROR Đích, CL

Mô tả:



Lệnh này dùng để quay vòng toán hạng sang phải, LSB được đưa sang cờ CF và MSB. CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết ROR Đích, 1. Nếu số lần quay là 8 (CL=8) thì toán hạng không đổi vì toán hạng quay đúng một vòng (nếu toán hạng đích là 8 bit), còn nếu CL=4 thì 4 bit cao đổi chỗ cho 4 bit thấp.

Sau lệnh ROR cờ CF mang giá trị cũ của LSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được xác định sau nhiều lần quay. Lệnh này thường dùng để tạo cờ CF từ giá trị của LSB làm điều kiện cho lệnh nhảy có điều kiện.

Các cờ bị thay đổi: CF, OF.

Ví dụ:

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

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

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

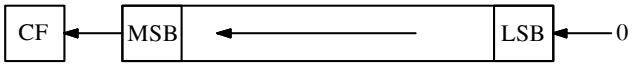
31. SAL/SHL - Shift Arithmetically Left (dịch trái số học)/Shift Logically Left (dịch trái logic).

Dạng lệnh:

SAL Đích, CL

SHL Đích, CL

Mô tả:



Hai lệnh này có tác dụng dịch trái số học toán hạng (còn gọi là dịch trái logic). Mỗi lần dịch MSB được đưa vào CF còn 0 được đưa vào LSB. CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết SAL Đích, 1

Sau lệnh SAL hoặc SHL cờ CF mang giá trị cũ của MSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được xác định sau nhiều lần quay. Lệnh này thường dùng để tạo cờ CF từ giá trị của MSB làm điều kiện cho lệnh nhảy có điều kiện.

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

Ví dụ:

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

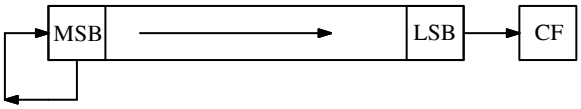
Trước khi thực hiện lệnh: AL = 11001100, CF = 0

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

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

Dạng lệnh: SAR Đích, CL

Mô tả:



Lệnh này có tác dụng dịch phải số học toán hạng. Mỗi lần dịch MSB được giữ lại (nếu ta hiểu đây là bit dấu thì dấu luôn không đổi) còn LSB được đưa vào CF. CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần thì ta có thể viết SAR Đích, 1.

Sau lệnh SAR cờ CF mang giá trị cũ của LSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được xác định sau nhiều lần quay. Lệnh này thường dùng để tạo cờ CF từ giá trị của MSB làm điều kiện cho lệnh nhảy có điều kiện.

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

Ví dụ:

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

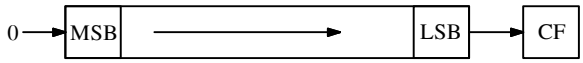
Trước khi thực hiện lệnh: AL = 11001100, CF = 1

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

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

Dạng lệnh: SHR Đích, CL

Mô tả:



Lệnh này có tác dụng dịch phải logic toán hạng. Mỗi lần dịch LSB được đưa vào CF còn 0 được đưa vào MSB. CL phải chứa sẵn số lần quay mong muốn. Trong trường hợp quay 1 lần có thể viết SHR Đích, 1

Sau lệnh SHR cờ CF mang giá trị cũ của LSB, còn cờ OF←1 nếu sau khi quay 1 lần mà bit MSB bị thay đổi so với trước khi quay, cờ OF sẽ không được xác định sau nhiều lần quay. Lệnh này thường dùng để tạo cờ CF từ giá trị của LSB làm điều kiện cho lệnh nhảy có điều kiện.

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

Ví dụ:

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

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

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

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

Dạng lệnh: TEST Đích, Nguồn

Mô tả: Đích ∧ Nguồn

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

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

Ví dụ:

```
TEST  AL, 3FH
TEST   AX, 0034H
```

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

Dạng lệnh: XOR Đích, Nguồn

Mô tả: Đích←Đích ⊕ Nguồn.

Lệnh XOR thực hiện logic XOR (hoặc đảo) giữa hai toán hạng và kết quả được lưu vào trong đích, một bit kết quả được đặt bằng 1 nếu nếu các bit tương ứng hai toán hạng là đối nhau. Nếu toán hạng đích trùng toán hạng

Nguồn thì kết quả bằng 0, do đó lệnh này còn được dùng để xoá thanh ghi về 0 kèm theo các cờ CF và OF cũng bị xoá.

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

Ví dụ:

```
XOR   AX, AX
XOR   BX, BX
MOV   AX, 5857H
MOV   BX, 58A8H
XOR   AX, BX
```

Trước khi thực hiện lệnh XOR	**Sau khi thực hiện lệnh XOR**
AX=5857H	AX=00FFH
BX=58A8H	BX=58A8H

2.3.2.5. Nhóm các lệnh so sánh

36. CMP – Compare (so sánh)

Dạng lệnh: CMP đích, Nguồn

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

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

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

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

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

Dạng lệnh:

CMPS Chuỗi_đích, Chuỗi_Nguồn

CMPSB

CMPSW

Lệnh này so sánh từng phần tử (byte hay từ) của hai xâu có các phần tử cùng loại. Lệnh chỉ tạo các cờ, không lưu kết quả so sánh, sau khi so sánh các toán hạng không bị thay đổi. Trong lệnh này ngầm định các thanh ghi với các chức năng:

- DS:SI là địa chỉ của phần tử so sánh trong chuỗi Nguồn.
- ES: DI là địa chỉ của phần tử so sánh trong chuỗi đích.
- Sau mỗi lần so sánh SI←SI± 1, DI←DI± 1 hoặc SI←SI± 2, DI←DI± 2 một cách tự động tuỳ thuộc vào cờ hướng DF là 0 hay 1 và chuỗi thao tác là chuỗi byte hay từ.

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

  Ví dụ:

```
StrByte1     DB      "chuoi byte1"
StrByte2     DB      "chuoi byte2"
StrWord1     DW      "chuoi word1"
StrWord2     DW      "chuoi word2"
LEA  SI, StrByte1    ;nap  gia  tri  hieu  dung
                     ;StrByte1 vao SI
LEA  DI, StrByte2    ;nap  gia  tri  hieu  dung
                     ;StrByte2 vaoDI
COMPS StrByte2, StrByte1   ;co  the  thay  bang
                     ;COMPSB
LEA SI, StrWord1     ;nap  gia  tri  hieu  dung
                     ;StrWord1 vao SI
LEA  DI, StrWord2    ;nap  gia  tri  hieu  dung
                     ;StrWord2 vaoDI
COMPS StrWord2, StrWord
                     ;co the thay bang COMPSW
```

2.3.2.6. Nhóm các lệnh nhảy (rẽ nhánh)

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

Dạng lệnh:

```
JA      NHAN
JNBE    NHAN
```

Mô tả: IP←IP+dịch chuyển

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu CF + ZF = 0. Quan hệ cao hơn/thấp là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn hai số không dấu. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JA/JNBE. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
CMP   AX, 12ABH       ;so sanh AX voi 12ABH
JA    THOI            ;nhay den THOI neu AX cao
                      ;hon 12ABH
```

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

Dạng lệnh:

```
JAE     NHAN
JNB     NHAN
JNC     NHAN
```

Mô tả: IP←IP+dịch chuyển

Ba lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu CF = 0. Quan hệ cao hơn/thấp là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn hai số không dấu. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JAE/JNB/JNC. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
CMP   AL, 10H         ;so sanh AL voi 10H
JAE   THOI            ;nhay den THOI neu AL cao
                      ;hon hoac bang 10H
```

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

Dạng lệnh:

```
JB      NHAN
JC      NHAN
JNAE    NHAN
```

Mô tả: IP←IP+dịch chuyển

Ba lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu CF = 1. Quan hệ cao hơn/thấp là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn hai số không dấu. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JB/JC/JNAE. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

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

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

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

Dạng lệnh:

JBE NHAN

JNA NHAN

Mô tả: IP←IP+dịch chuyển

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu

CF +ZF = 1. Quan hệ cao hơn/thấp là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn hai số không dấu. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JBE/JNA. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
CMP   AL, 10H       ;so sanh AL voi 10H
JBE   THOI          ;nhay den THOI neu AL thap
                    ;hon hoac bang 10H
```

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

Dạng lệnh: JCXZ NHAN

Mô tả: IP←IP+dịch chuyển

Lệnh trên biểu diễn thao tác nhảy có điều kiện tới NHAN nếu CX = 0 và không liên quan đến ZF. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JCXZ. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

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

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

Dạng lệnh:

JE NHAN

JZ NHAN

Mô tả: IP←IP+dịch chuyển

Lệnh trên biểu diễn thao tác nhảy có điều kiện tới NHAN nếu ZF = 1. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JE/JZ. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
SUB   AL, 10H;tru AL cho 10H
JE    THOI    ;nhay den THOI neu AL bang 10H
```

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

Dạng lệnh:

JG NHAN

JNLE NHAN

Mô tả: IP←IP+dịch chuyển

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu (SF xor OF)+ZF = 0. Quan hệ lớn hơn/bé hơn là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) của hai số có dấu. Lớn hơn có nghĩa là dương hơn. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JG/JNLE. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
CMP   AL, 10H         ;so sanh AL voi 10H
JG    THOI    ;nhay den THOI neu AL lon hon 10H
```

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

Dạng lệnh:

JGE NHAN

JNL NHAN

Mô tả: IP←IP+dịch chuyển

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu (SF xor OF) = 0. Quan hệ lớn hơn/bé hơn là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) của hai số có dấu. Lớn hơn có nghĩa là dương hơn. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JGE/JNL. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
CMP   AL, 10H          ;so sanh AL voi 10H
JGE   THOI             ;nhay den THOI neu AL lon
                       ;hon hoac bang 10H
```

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

Dạng lệnh:

JL NHAN

JNGE NHAN

Mô tả: IP←IP+dịch chuyển

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu (SF xor OF)= 1. Quan hệ lớn hơn/bé hơn là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) của hai số có dấu. Lớn hơn có nghĩa là dương hơn. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JL/JNGE. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
CMP   AL, 10H          ;so sanh AL voi 10H
JL    THOI    ;nhay den THOI neu AL nho hon 10H
```

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

Dạng lệnh:

JLE NHAN

JNG NHAN

Mô tả: IP←IP+dịch chuyển

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu (SF xor OF)+ZF= 1. Quan hệ lớn hơn/bé hơn là quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) của hai số có dấu. Lớn hơn có nghĩa là dương hơn. NHAN phải nằm cách xa một khoảng -128…+127 byte so với lệnh tiếp theo sau lệnh JLE/JNG. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
CMP   AL, 10H          ;so sanh AL voi 10H
JLE   THOI             ;nhay den THOI neu AL nho
                       ;hon hoac bang 10H
```

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

JMP trao quyền điều khiển cho vùng mục tiêu một cách không điều kiện. Lệnh này có các chế độ giống như lệnh CALL và nó cũng phân biệt nhảy gần, nhảy xa.

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

JMP NHAN

Lệnh mới này bắt đầu địa chỉ ứng với NHAN. Chương trình sẽ căn cứ vào khoảng dịch giữa NHAN và lệnh nhảy để xác định xem nó là:

+ Nhảy ngắn: Trong trường hợp này NHAN phải nằm cách xa (dịch đi một khoảng).

-128…127 byte so với lệnh tiếp theo sau lệnh JMP. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển. Do đó

IP←IP+dịch chuyển

Đây là lệnh nhảy trực tiếp vì dịch chuyển để trực tiếp trong mã lệnh.

Để định hướng cho chương trình dịch làm việc nên viết lệnh dưới dạng:

JMP SHORT NHAN

+ Nhảy gần: Trong trường hợp này NHAN phải nằm cách xa (dịch đi một khoảng)

-32768…+32767 byte so với lệnh tiếp theo sau lệnh JMP. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển. Do đó

IP←IP+dịch chuyển

Đây là lệnh nhảy trực tiếp vì dịch chuyển để trực tiếp trong mã lệnh.

Để định hướng cho chương trình dịch làm việc nên viết lệnh dưới dạng:

JMP NEAR NHAN

+ Nhảy xa: Trong trường hợp này NHAN nằm ở đoạn mã khác so với lệnh tiếp theo sau lệnh JMP. Chương trình sẽ căn cứ vào vị trí NHAN để xác định giá trị địa chỉ nhảy đến (CS:IP của NHAN). Sau đó:

IP←IP của NHAN

CS←CS của NHAN

JMP BX

Đây là lệnh nhảy gần, trước đó BX phải chứa địa chỉ lệch của lệnh định nhảy đến trong đoạn CS. Khi thực hiện lệnh này thì IP←BX. Đây là lệnh nhảy gián tiếp vì địa chỉ lệch nằm trong thanh ghi. Để định hướng cho chương trình dịch làm việc ta nên viết lệnh dưới dạng:

JMP NEAR PTR BX

JMP [BX]

Đây là lệnh nhảy gần. IP mới được lấy từ nội dung 2 ô nhớ do BX và BX+1 chỉ ra trong đoạn DS (SI, DI có thể dùng thay chỗ của BX). Đây là lệnh nhảy gián tiếp vì địa chỉ lệch để trong ô nhớ. Để định hướng cho chương trình dịch làm việc ta nên viết lệnh dưới dạng:

JMP WORD PTR [BX]

Một biến dạng khác của lệnh trên thu được khi ta viết lệnh dưới dạng:

JMP DWORD PTR [BX]

Đây là lệnh nhảy xa. Địa chỉ nhảy đến ứng với CS:IP. Giá trị gán cho IP và CS được chứa trong 4 ô nhớ do BX và BX+1 (cho IP), BX+2 và BX+3 cho (CS) chỉ ra trong đoạn DS (SI, DI có thể sử dụng thay chỗ của BX)

Đây cũng là lệnh nhảy gián tiếp vì địa chỉ lệch và địa chỉ cơ sở nằm trong ô nhớ.

Các cờ bị thay đổi: không.

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

Dạng lệnh:

JNE NHAN

JNZ NHAN

Mô tả: IP←IP+dịch chuyển

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu ZF = 0. NHãN phải nằm cách xa (dịch đi một khoảng) -128…127 byte so với lệnh tiếp theo sau lệnh JNE/JNZ. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định độ dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
CMP  AL, 10H        ;so sanh AL voi 10H
JNE  THOI   ;nhay den THOI neu AL khac 10H
```

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

Dạng lệnh: JNO NHAN

Mô tả: IP←IP+dịch chuyển

Đây là lệnh nhảy có điều kiện tới NHAN nếu OF = 0. NHãN phải nằm cách xa (dịch đi một khoảng) -128…127 byte so với lệnh tiếp theo sau lệnh JNO. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định độ dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
MUL   AL, BL          ;nhan AL voi BL
JNO   THOI    ;nhay den THOI neu khong tran
```

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

Dạng lệnh:

JNP NHAN

JPO NHAN

Mô tả: IP←IP+dịch chuyển

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu PF = 0. NHAN phải nằm cách xa (dịch đi một khoảng) -128…127 byte so với lệnh tiếp theo sau lệnh JNP/JPO. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định độ dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

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

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

Dạng lệnh: JNS NHAN

Mô tả: IP←IP+dịch chuyển

Đây là lệnh nhảy có điều kiện tới NHAN nếu SF = 0. Kết quả là dương sau khi thực hiện các phép toán có dấu. NHAN phải nằm cách xa (dịch đi một khoảng) -128…127 byte so với lệnh tiếp theo sau lệnh JNS. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định độ dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

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

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

Dạng lệnh: JO NHAN

Mô tả: IP←IP+dịch chuyển

Đây là lệnh nhảy có điều kiện tới NHAN nếu OF = 1. Tức là sảy xa tràn sau khi thực hiện các phép toán có dấu. NHAN phải nằm cách xa (dịch đi một khoảng) -128…127 byte so với lệnh tiếp theo sau lệnh JO. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định độ dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

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

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

Dạng lệnh:

JP NHAN

JPE NHAN

Mô tả: IP←IP+dịch chuyển

Hai lệnh trên biểu diễn cùng một thao tác nhảy có điều kiện tới NHAN nếu PF = 1. NHAN phải nằm cách xa (dịch đi một khoảng) -128…127 byte so với lệnh tiếp theo sau lệnh JP/JPE. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định độ dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

```
IN    AL, 99H         ;doc ky tu tu cong
OR    AL, AL          ;tao co
JP    THOI    ;nhay den THOI neu parity chan
```

55. JS – Jump if Sign (nhảy nếu âm)

Dạng lệnh: JS NHAN

Mô tả: IP←IP+dịch chuyển

Đây là lệnh nhảy có điều kiện tới NHAN nếu SF = 1. Kết quả là âm sau khi thực hiện các phép toán có dấu. NHAN phải nằm cách xa (dịch đi một khoảng) -128…127 byte so với lệnh tiếp theo sau lệnh JS. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định độ dịch chuyển.

Các cờ bị thay đổi: không.

Ví dụ:

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

2.3.2.7. Nhóm các lệnh lặp

56. LOOP – Loop if CX is not 0 (lặp nếu CX ≠ 0)

Dạng lệnh: LOOP NHAN

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

NHAN phải nằm cách xa (dịch đi một khoảng) tối đa -128 byte so với lệnh tiếp theo sau lệnh LOOP.

Các cờ bị thay đổi: không.

Ví dụ:

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

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

Dạng lệnh:

LOOPE NHAN

LOOPZ NHAN

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

NHAN phải nằm cách xa (dịch đi một khoảng) tối đa -128 byte so với lệnh tiếp theo sau lệnh LOOPE/LOOPZ.

Các cờ bị thay đổi: không.

Ví dụ:

```
     MOV   AL, AH         ;AL=AH
     MOV   CX, 50         ;nap so lan lap vao CX
LAP: INC   AL             ;tang AL
     COMP  AL, 16         ;so sanh AL voi 16
     LOOPE LAP            ;lap lai cho den khi AL≠16
                          ;hoac CX=0
```

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

Dạng lệnh:

LOOPNE NHAN

LOOPNZ NHAN

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

NHAN phải nằm cách xa (dịch đi một khoảng) tối đa -128 byte so với lệnh tiếp theo sau lệnh LOOPNE/LOOPNZ.

Các cờ bị thay đổi: không.

Ví dụ:

```
     MOV  AL, AH          ;AL=AH
     MOV  CX, 50          ;nap so lan lap vao CX
LAP:INC   AL              ;tang AL
     COMP AL, 16          ;so sanh AL voi 16
     LOOPNE LAP           ;lap lai cho den khi AL=16
                          ;hoac CX=0
```

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

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

Các cờ bị thay đổi: không.

Ví dụ:

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

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

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

Các cờ bị thay đổi: không.

Ví dụ:

```
REPE CMPSB  ;lap lai lenh so sanh cac byte cua
            ;chuoi toi khi CX=0 hoac ZF=0
```

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

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

Các cờ bị thay đổi: không.

Ví dụ:

```
REPNE SCASB ;lap lai lenh quet cac byte cua
     ;chuoi toi khi het chuoi (CX=0)
     ;hoac AL bang mot phan tu cua chuoi
```

2.3.2.8. Nhóm các lệnh điều khiển, đặc biệt khác

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

Dạng lệnh: CALL Thủ_tục

Mô tả: Lệnh này dùng để chuyển hoạt động của vi xử lý từ chương trình chính (CTC) sang chương trình con (ctc). Nếu ctc nằm trong cùng một đoạn mã với CTC ta có gọi gần (near call). Nếu ctc và CTC nằm ở hai đoạn mã khác nhau ta có gọi xa (far call).

- Nếu gọi gần: Lưu vào Stack giá trị IP của địa chỉ trở về (vì CS không đổi) và các thao tác khi gọi ctc diễn ra như sau:
  + Nội dung thanh ghi SP giảm đi 2 byte, SP←SP – 2.
  + Nội dung thanh ghi IP được cất vào ngăn xếp (lưu địa chỉ trở về) {SP}←IP.
  + Địa chỉ lệch của ctc (lên tới ±32K) được lưu vào thanh ghi IP.
  + Khi gặp lệnh RET ở cuối ctc thì VXL lấy lại địa chỉ trở về IP từ Stack và tăng SP lên 2 byte.
- Nếu gọi xa: Lưu vào Stack giá trị IP và CS của địa chỉ trở về và các thao tác khi gọi ctc diễn ra như sau:
  + Nội dung thanh ghi SP giảm đi 2 byte, SP←SP – 2 và CS được lưu vào ngăn xếp.
  + Nội dung của CS được thay bằng địa chỉ đoạn của ctc được gọi.
  + Nội dung thanh ghi SP lại giảm đi 2 byte và IP được cất vào ngăn xếp.
  + Địa chỉ lệch của ctc được lưu vào thanh ghi IP.
  + Khi gặp lệnh RET ở cuối ctc thì VXL lấy lại địa chỉ trở về IP từ Stack và tăng SP lên 2 byte sau đó tiếp tục lấy lại CS và tăng SP lên 2 byte.

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

Ví dụ:

```
CALL NEAR
CALL FAR
```

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

Dạng lệnh: CLC

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

Các cờ bị thay đổi: CF.

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

Dạng lệnh: CLD

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

Các cờ bị thay đổi: DF.

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

Dạng lệnh: CLD

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

Các cờ bị thay đổi: IF.

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

Dạng lệnh: CMC

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

Các cờ bị thay đổi: CF

67. HLT – Halt processing (dừng)

Dạng lệnh: HLT

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

Các cờ bị thay đổi: không.

68. INT – Interrupt (lệnh gọi ngắt)

Dạng lệnh: INT N (N=0…FFH)

Các thao tác của 8086 khi chạy lệnh: INT N

- Tạo địa chỉ mới của Stack, cất thanh ghi cờ vào Stack: SP←SP-2, {FR}→SP.
- Cấm các ngắt khác tác động vào vi xử lý, cho vi xử lý chạy ở chế độ từng lệnh: IF←0, TF←0.
- Tạo địa chỉ mới của Stack, cất địa chỉ đoạn của địa chỉ trở về vào Stack: SP←SP-2, SP←CS.
- Tạo địa chỉ mới của Stack, cất địa chỉ lệch của địa chỉ trở về vào Stack: SP←SP-2, SP←IP.
- Vi xử lý lấy lệnh tại địa chỉ mới, địa chỉ con trỏ ngắt được tính toán như sau:

  {Nx4}→IP, {Nx4+2}→CS

Ví dụ: với N = 8 thì CS←{0022H} và IP←{0020H}

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

Dạng lệnh: IRET

Trở về chương trình chính từ chương trình con phục vụ ngắt. Trả lại quyền điều khiển cho chương trình tại vị trí xảy ra ngắt bằng cách lấy lại các giá trị thanh ghi IP, CS và các cờ từ vùng Stack.

- {SP}→IP, SP←SP+2
- {SP}→CS, SP←SP+2
- {SP}→FR, SP←SP+2

Các cờ bị thay đổi: tất cả các cờ (được phục hồi như trước khi diễn ra ngắt).

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

Dạng lệnh: NOP

Lệnh này không thực hiện một công việc gì ngoài việc làm tăng nội dung của IP và tiêu tốn 3 chu kỳ đồng hồ. Nó thường được dùng để tính thời gian trễ trong các vòng trễ hoặc để chiếm chỗ các lệnh cần thêm vào chương trình sau này mà không làm ảnh hưởng dến độ dài chương trình.

Các cờ bị thay đổi: không.

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

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

Mô tả: RET được đặt cuối ctc để vi xử lý lấy lại địa chỉ trở về, mà nó đã được tự động cất tại ngăn xếp khi có lệnh gọi ctc. Đặc biệt nếu dùng lệnh RET n thì sau khi đã lấy lại được địa chỉ trở về (chỉ có IP hoặc cả IP và CS) thì SP←SP+n (dùng để nhảy qua mà không lấy lại các thông số khác của chương trình còn lại trong ngăn xếp.

Các cờ bị thay đổi: không.

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

Dạng lệnh: STC

Mô tả: CF←1

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

Các cờ bị thay đổi: CF=1.

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

Dạng lệnh: STD

Mô tả: DF←1

STD thiết lập cờ hướng bằng 1 và không ảnh hưởng đến các cờ khác. Lệnh này định hướng thao tác cho các lệnh làm việc với chuỗi theo chiều

(←). Các thanh ghi SI, DI liên quan sẽ được giảm đi khi làm việc xong với phần tử của chuỗi.

Các cờ bị thay đổi: DF=1.

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

Dạng lệnh: STI

Mô tả: IF←1

Lệnh này lập cờ cho phép ngắt để cho phép các yêu cầu ngắt tác động vào chân INTR được CPU nhận biết. Khi IF=1 nếu có tín hiệu INTR=1 thì 8086 sẽ bị ngắt, nó sẽ tự động cất thanh ghi cờ và địa chỉ trở về vào ngăn xếp rồi chuyển sang chương trình phục vụ ngắt. Tại cuối chương trình phục vụ ngắt sẽ có lệnh trở về CTC (lệnh IRET) để 8086 lấy lại từ ngăn xếp giá trị thanh ghi cờ và địa chỉ trở về.

Các cờ bị thay đổi: IF=1

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

Dạng lệnh: WAIT

Mô tả: Lệnh này đưa bộ vi xử lý 8086 vào trạng thái nghỉ và nó sẽ ở trạng thái ngày cho đến khi có tín hiệu ở mức thấp tác động vào chân TEST hoặc khi có tín hiệu ở mức cao tác động vào chân INTR. Nếu có yêu cầu ngắt và yêu cầu này được phép tác động vào chân INTR thì sau khi chương trình phục vụ ngắt được thực hiện nó sẽ lại trở về trạng thái nghỉ. Lệnh này dùng để đồng bộ hoá hoạt động của 8086 với các bộ đồng xử lý bên ngoài.

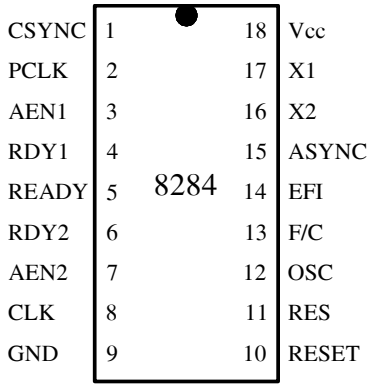
Các cờ bị thay đổi: không.

2.4. CÁC MẠCH PHỤ TRỢ

2.4.1. MẠCH TẠO XUNG NHỊP

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

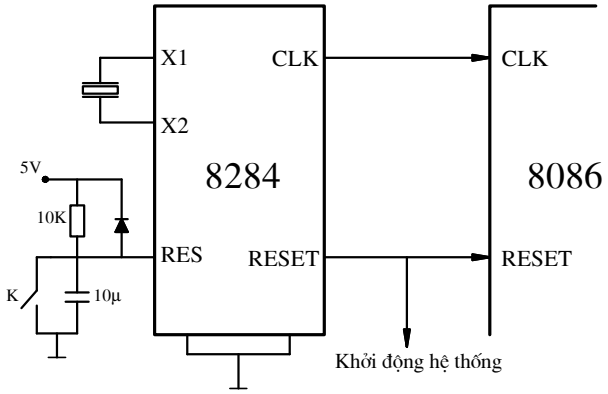
Sơ đồ chân của 8284 như sau:



Hình 2.4.1a. Sơ đồ bố trí chân của 8284

Ý nghĩa các chân tín hiệu:

- AEN1, AEN2: Tín hiệu cho phép chọn đầu vào tương ứng RDY1, RDY2 làm tín hiệu báo tình trạng sẵn sàng của bộ nhớ hoặc thiết bị ngoại vi.
- RDY1, RDY2: Cùng với AEN1, AEN2 dùng để gây ra các chu kỳ đợi.
- ASYNC: Chọn đồng bộ hai tầng hoặc đồng bộ một tầng cho tín hiệu RDY1, RDY2. Trong chế độ đồng bộ một tầng (ASYNC=1) tín hiệu RDY có ảnh hưởng đến tín hiệu READY tới tận sườn xuống của xung đồng hồ tiếp theo, còn trong chế độ đồng bộ hai tầng (ASYNC=0)) tín hiệu RDY có ảnh hưởng đến tín hiệu READY khi có sườn xuống của xung đồng hồ tiếp theo.
- READY: Nối đến đầu vào READY của CPU. Tín hiệu này được đồng bộ với các tín hiệu RDY1, RDY2.
- X1, X2: Nối hai chân của thạch anh với tần số fx.
- EFI: Lối vào cho xung từ bộ dao động ngoài.
- F/C: Dùng để chọn nguồn tín hiệu chuẩn cho 8284. Khi chân này ở mức cao thì xung đồng hồ bên ngoài sẽ được làm xung nhịp cho 8284, ngược lại thì xung đồng hồ của mạch dao động bên trong dùng thạch anh sẽ được chọn để làm xung nhịp.
- CLK: Xung nhịp fCLK = fx/3 với độ rỗng 77% nối đến chân CLK của 8086.
- PCLK: Xung nhịp fCLK = fx/6 với độ rỗng 50% dành cho thiết bị ngoại vi.
- OSC: Xung nhịp đã được khuếch đại có tần số bằng fx của bộ dao động.
- RES: chân khởi động nối vào mạch R-C để 8284 có thể khởi động khi bật nguồn.
- RESET: Lối vào RESET của 8086 là tín hiệu khởi động lại cho toàn hệ.
- CSYNC: Lối vào cho xung đồng hồ chung khi trong hệ thống có các 8284 dùng dao động ngoài tại chân EFI. Khi dùng mạch dao động trong thì phải ối đất chân này.



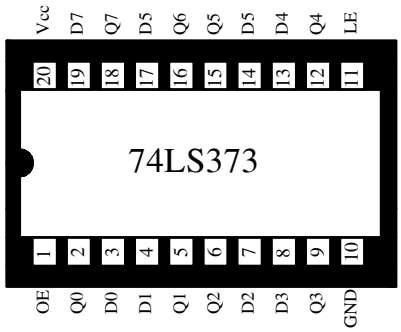
Hình 2.4.1b. Sơ đồ nối 8284 với 8086

2.4.2. MẠCH CHỐT VÀ ĐỆM BUS

2.4.2.1. Mạch chốt (Latch)

Các mạch chốt là các mạch mà khi tín hiệu cho phép chốt ALE (Address Latch Enable) ở mức L (Low) thì các đầu ra sẽ giữ nguyên trạng thái như trạng thái của các tín hiệu đầu vào trước khi chốt, lúc này cho dù các đầu vào thay đổi thì các đầu ra vẫn không thay đổi. Muốn thay đổi trạng thái các đầu ra thì ALE phải ở mức H (Hight). Vi mạch chốt sử dụng phổ biến trong các hệ vi xử lý hiện nay là 74LS373.

Sơ đồ bố trí chân như sau (hình 2.4.2a):



Sơ đồ chân của 74LS373

Vi mạch 74LS373 có chốt chuyển 8 bit nối với bộ đệm đầu ra 3 trạng thái, trong đó có hai phần được khống chế bởi các cổng điều khiển cho phép chốt LE (chân 11) và cho phép đầu ra OE (chân 1) độc lập nhau.

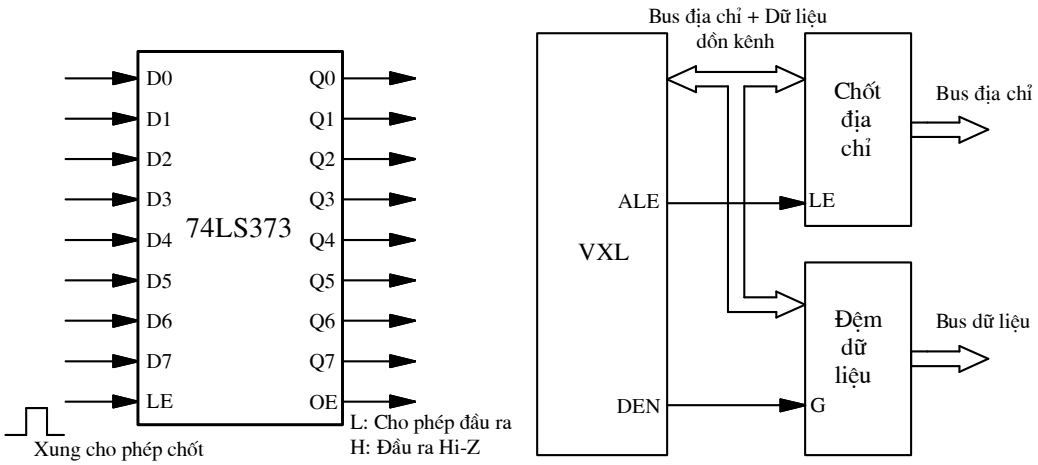
Số liệu đầu vào D được chuyển tới đầu ra khi tín hiệu cho phép chốt ở mức cao. Và dưới đây là bảng sự thật của vi mạch chốt 74LS373.

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

Tín hiệu cho phép ra tích cực ở mức thấp để điều khiển tất cả bộ đệm 8 đầu ra 3 trạng thái mà không phụ thuộc vào hoạt động bên trong của các chốt. Dữ liệu có thể vẫn được duy trì và dữ liệu mới có thể vẫn được đưa vào ngay cả khi các lối ra đang ở trạng thái trở kháng cao hoặc trạng thái thả nổi. Khi OE ở mức cao thì các đầu ra ở trạng thái trở kháng cao (Hi – Z).

Vi mạch 74LS373 thường được sử dụng để chốt và đệm các đường dây địa chỉ với các bus được dồn chân. Chú ý là thông tin địa chỉ chỉ tồn tại trên bus trong một thời gian ngắn. Mạch chốt được điều khiển bởi tín hiệu ALE để giữ lại thông tin địa chỉ đó. Hình vẽ 2.4.2b là sơ đồ nguyên lý của bộ chốt tín hiệu dùng 74LS373. Khi chân LE ở mức cao và chân OE ở mức thấp các tín hiệu vào được chuyển tới đầu ra. Lúc này 74LS373 làm việc như bộ đệm. Khi tín hiệu trên ALE chuyển từ cao xuống thấp, các tín hiệu đầu vào D sẽ được chốt và ở trạng thái này, các tín hiệu đầu ra được giữ nguyên cho dù tín hiệu đầu vào có thay đổi.

Hình 2.4.2c là sơ đồ đơn giản, minh hoạ ứng dụng thực tế của 74LS373 trong hệ vi xử lý. Trong đó để tiết kiệm chân, vi xử lý được chế tạo gồm các chân địa chỉ và dữ liệu chung. Để phân biệt tín hiệu địa chỉ với dữ liệu, vi xử lý có thêm chân ALE và DEN (data enable). Muốn đưa tín hiệu địa chỉ, vi xử lý đồng thời đưa xung ALE = 1 đến chân LE của bộ chốt địa chỉ 74LS373 và đưa tín hiệu DEN đến chân G của bộ đệm số liệu. Lúc này tín hiệu địa chỉ sẽ được chốt và truyền ra bus địa chỉ, còn bộ đệm số liệu bị cấm (Hi – Z). Muốn truyền, nhận số liệu, vi xử lý xác lập tín hiệu ALE = 0 và DEN = 0. Lúc này bộ chốt địa chỉ không làm việc (đầu ra không thay đổi trạng thái) và trên bus địa chỉ đã có địa chỉ của thiết bị cần truy cập đến. Trong khi đó tín hiệu DEN = 0 nên bộ đệm số liệu sẽ mở cổng cho dữ liệu đi qua. Chiều vận chuyển của dữ liệu được vi xử lý điều khiển thông qua các bộ đệm dữ liệu hai chiều.



Hình 2.4.2c. Sơ đồ nguyên lý và ghép nối 74LS373 với vi xử lý

2.4.2.2. Bộ đệm bus

Bộ đệm một chiều 74LS244

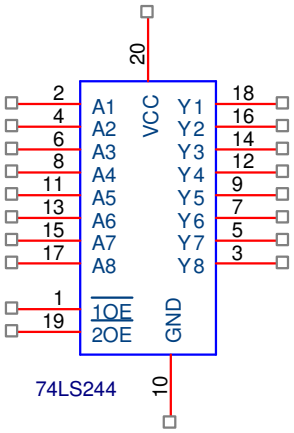
Mục đích của việc sử dụng đệm một chiều là để tăng công suất truyền tải cho bus. Một trong những vi mạch đệm một chiều thông dụng hiện nay là 74LS244.

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

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

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

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



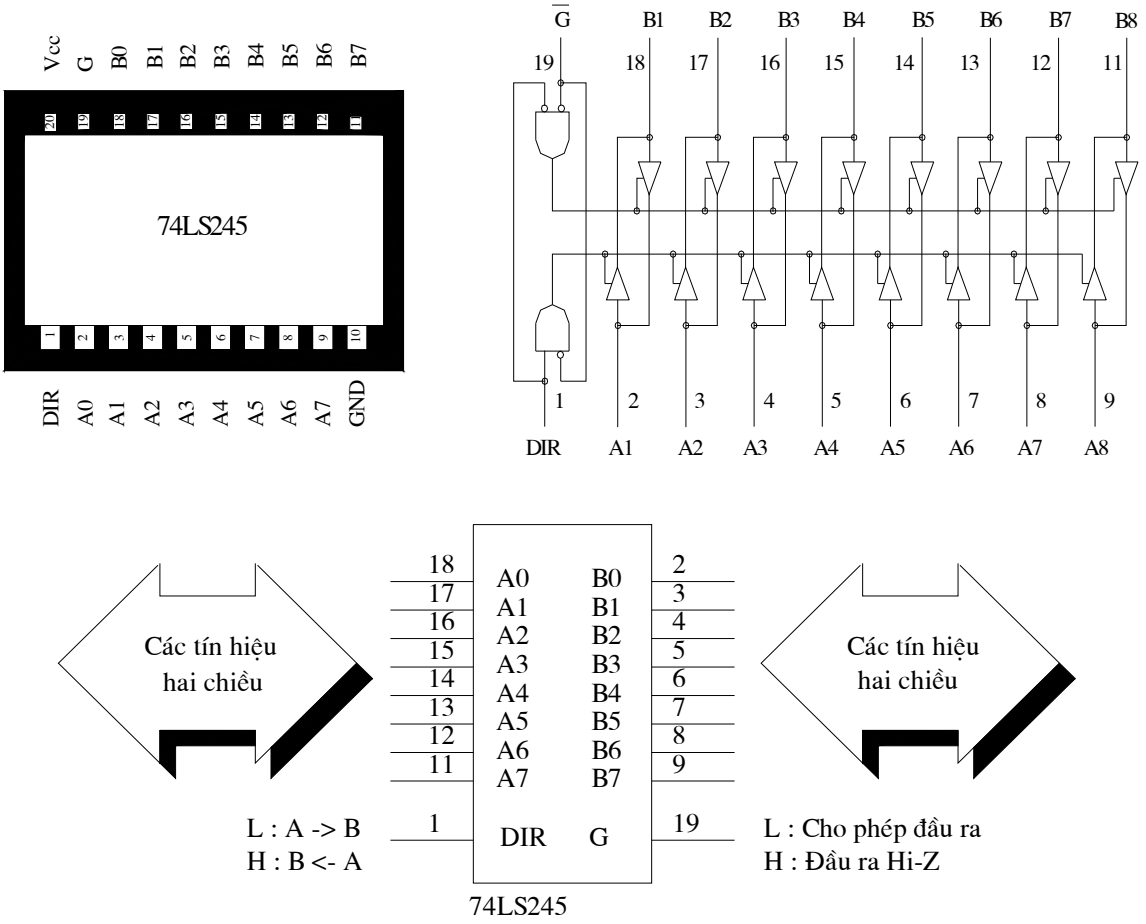
Sơ đồ chân của 74LS244

Thông thường 74LS244 được dùng để đệm bus địa chỉ (vì bus địa chỉ là bus một chiều).

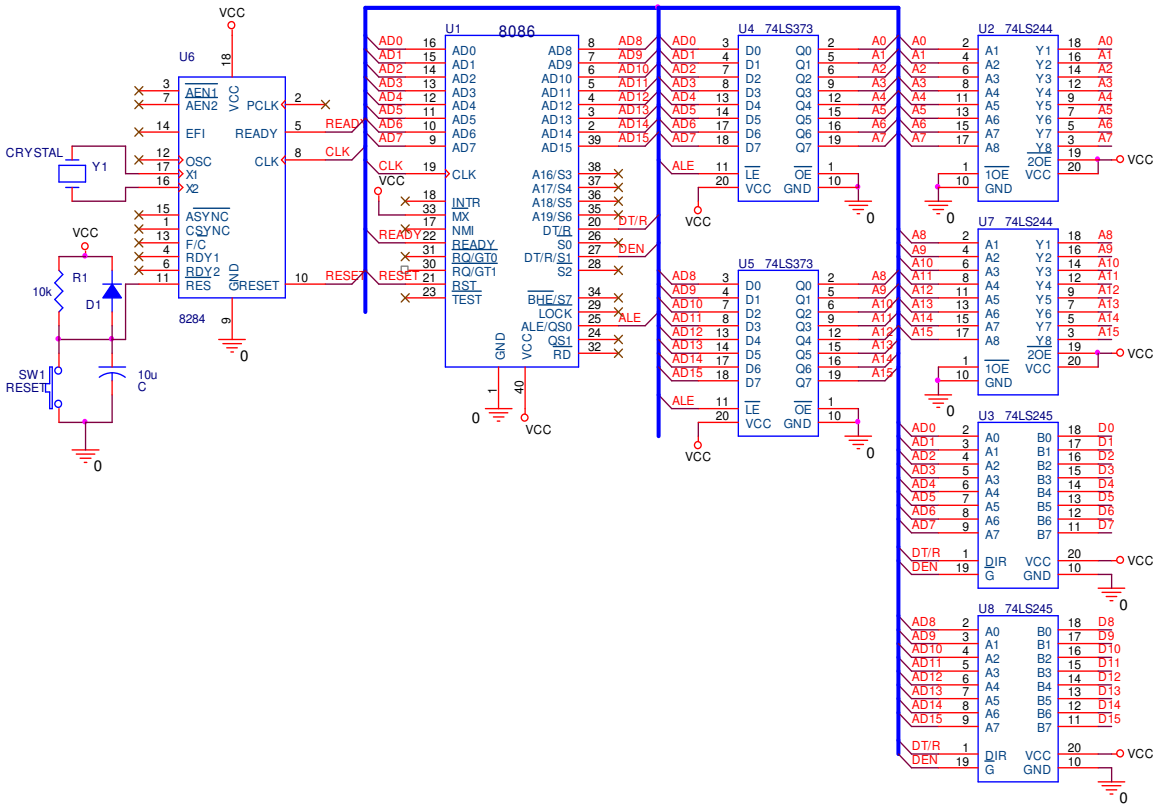
Bộ đệm số liệu hai chiều 74LS245

Một số tín hiệu, chẳng hạn như tín hiệu địa chỉ, là tín hiệu một chiều. Nghĩa là các tín hiệu này chỉ được vi xử lý đưa ra trên BUS địa chỉ. Một số tín hiệu khác, như các tín hiệu số liệu, lại là các tín hiệu hai chiều. Các tín hiệu này có thể được vi xử lý đưa ra trên Bus số liệu khi nó cần ghi số liệu lên một thiết bị nào đó như bộ nhớ RAM hoặc thiết bị vào ra, nhưng số liệu cũng có thể được đưa vào các thanh ghi của vi xử lý khi nó đang thực hiện quá trình đọc số liệu từ một thiết bị nào đó. Vì vậy, người ta cần sử dụng các bộ đệm số liệu hai chiều để thực hiện các chức năng nói trên một cách linh hoạt theo các tín hiệu điều khiển.

- DIR = 0: Dữ liệu được truyền từ A sang B (từ trái qua phải).
- DIR = 1: Dữ liệu được truyền từ B sang A (từ phải qua trái).



Hình 2.4.2d. Bộ đệm số liệu hai chiều 74LS245



Sơ đồ ghép nối 8086 với mạch Reset 8284, chốt 74LS373, đệm 74LS244 và 74LS245

2.4.3. CÁC BỘ NHỚ

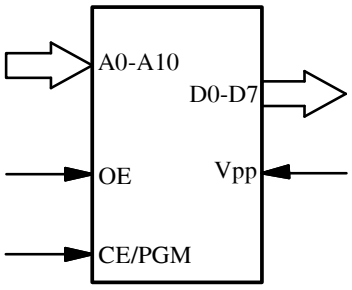
Bộ nhớ dùng để lưu trữ mã lệnh và dữ liệu. Bộ nhớ được ghép nối trực tiếp với CPU và là nơi đầu tiên được CPU lấy thông tin. Các bộ nhớ bán dẫn có tốc độ truy nhập cao, có dung lượng lớn, kích thước nhỏ và tiêu thụ năng lượng thấp. Chúng được chế tạo thành những chip riêng biệt hoặc được tích hợp ngay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

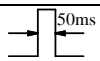
2.4.3.1. Bộ nhớ cố định ROM (Read Only Memory)

- **ROM**: Được gọi là vi mạch nhớ cố định vì Số liệu do hãng chế tạo nạp một lần và thông tin sẽ được lưu trữ vĩnh viễn, sau đó chỉ có thể đọc ra mà thôi. Khi nguồn điện mất thì số liệu vẫn được giữ nguyên. Vì vậy các bộ nhớ cố định thường được sử dụng để lưu trữ các chương trình kiểm tra và khởi động một hệ vi xử lý khi bật nguồn hoặc các thông số của những lần chạy trước.
- **PROM** (Programable ROM – ROM có khả năng lập trình): Người thiết kế có khả năng nạp chương trình vào ROM nhưng chỉ được một lần duy nhất và không thể sửa đổi được.

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

Người sử dụng có thể nạp và xoá chương trình một số lần. Có thể nạp bằng xung điện và xoá bằng tia cực tím. Hiện tại EPROM đang được dùng khá phổ biến. Họ 27xxx có các loại sau: 2708 (1Kx8), 2716 (2Kx8), 2732 (4Kx8), 2764 (8Kx8), 27128 (16Kx8), 27256(32Kx8), 27512 (64x8) với thời gian thâm nhập tac=250-450 ns tuỳ theo loại cụ thể. Trên hình 2.4.2b là sơ đồ các tín hiệu và bảng chức năng của 2716.



Chân / Chế độ	CE/ PGM	OE	Vpp (V)	Vcc (V)	D0-D7
Đọc	0	0	+5	+5	Dout
Duy trì	1	x	+5	+5	HZ
Ghi		1	+25	+5	Din
Kiểm tra ghi	0	0	+25	+5	Dout
Cấm ghi	0	1	+25	+5	HZ

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

x: Không quan tâm

HZ: Trở kháng cao

Ngoài ra tuỳ theo các nạp xoá số liệu người ta còn chia bộ nhớ bán cố định ra những loại như EEPOM, EAPROM.

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

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

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

- **RAM tĩnh** (Static RAM): Loại RAM này lấy cấu trúc Flip-Flop làm đơn vị nhớ cơ sở. Vì vậy, khi số liệu đã được ghi vào mà nguồn nuôi vẫn có thì số liệu trong RAM vẫn còn .
- **RAM động** (Dynamic RAM): RAM động lợi dụng điện dung ký sinh của cực cổng (Gate) transistor trường MOS để chứa dữ liệu. Vì vậy, khi số liệu đã được đưa vào mặc dù còn nguồn nuôi nhưng số liệu cứ mất dần do điện tích của tụ giảm theo thời gian. Để giữ lại số liệu ấy phải làm thêm một mạch gọi là mạch làm tươi và có tín hiệu làm tươi (Refresh).
- Sơ đồ khối của DRAM được chỉ ra trên hình 2.4.2c.

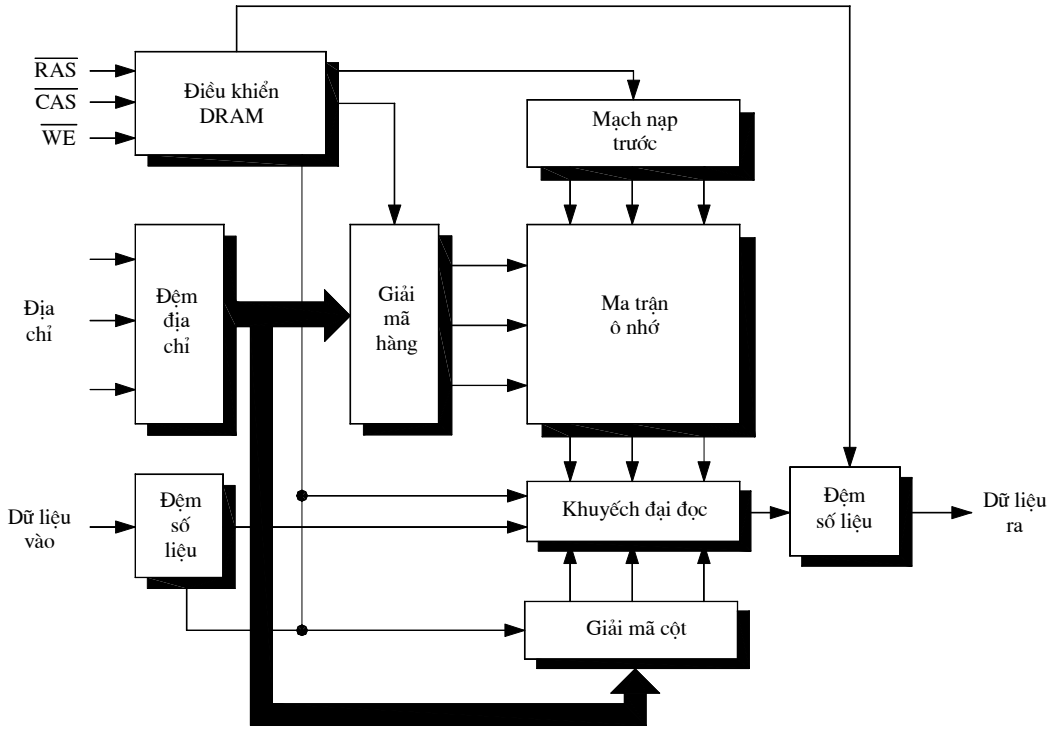
Địa chỉ của ô nhớ được chia thành địa chỉ hàng và địa chỉ cột. Các địa chỉ này được đọc vào bộ đệm địa chỉ một cách lần lượt và quá trình này được điều khiển bởi các tín hiệu RAS (Row Access Strobe) và CAS (Column Access Strobe). Nếu $\overline{\text{RAS}}$ ở mức tích cực thấp thì DRAM nhận địa chỉ vào bộ giải mã hàng, còn khi $\overline{\text{CAS}}$ ở mức tích cực thấp thì DRAM nhận địa chỉ vào bộ giải mã cột. Các công việc này cùng với việc điều khiển đọc/viết được thực hiện nhờ bộ điều khiển DRAM .

Thông thường các DRAM phải được làm tươi định kỳ từ 1÷16ms (chú ý rằng trong quá trình đọc hoặc viết ô nhớ thì các hàng cũng được tự động làm tươi). Hiện nay có 3 phương pháp làm tươi phổ biến như sau:

- **Row Access Strobe Only Refresh** (RAS Only Refresh): Phương pháp đơn giản và hay được dùng nhất này được thực hiện những chu trình đọc giả (Dummy Read Cycle) để làm tươi ô nhớ. Trong chu trình này, tín hiệu $\overline{\text{RAS}}$ được đặt ở mức tích cực và địa chỉ hàng (được gọi là địa chỉ

làm tươi) được áp vào DRAM, nhưng tín hiệu $\overline{CAS}$ bị cấm. Như vậy DRAM sẽ đọc bên trong một hàng vào các cặp dây bit và khuếch đại số liệu đọc nhưng không truyền được chúng tới bộ điệm đầu ra. Để làm tươi toàn bộ bộ nhớ, mạch lôgíc hoặc bộ xử lý bên ngoài phải cấp tất cả các địa chỉ hàng một cách lần lượt. Nhược điểm của phương pháp này là cần một mạch lôgic bên ngoài hoặc ít nhất một chương trình cho việc làm tươi.

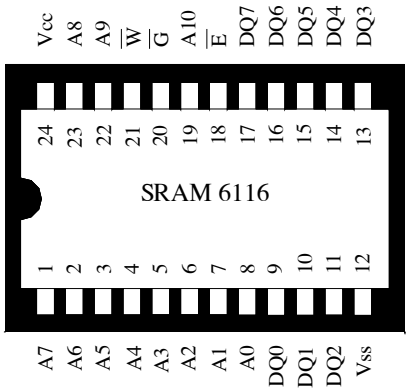
- **Hidden Refresh**: Chu trình làm tươi ít nhiều được đấu đằng sau chu trình đọc bộ nhớ. Tín hiệu CAS được giữ nguyên ở mức thấp trong khi chỉ có $\overline{RAS}$ là được chuyển đổi. Việc đọc số liệu trong chu kỳ đọc được chấp nhận ngay cả khi chu kỳ làm tươi đang được tiến hành. Vì rằng thời gian làm tươi cần ngắn hơn thời gian đọc nên loại làm tươi này tiết kiệm được thời gian. ở đây bộ đếm địa chỉ cũng tự phát ra địa chỉ làm tươi. Nếu $\overline{CAS}$ giữ nguyên mức thấp trong thời gian đủ dài, một vài chu kỳ làm tươi có thể được thực hiện liên tiếp nhau.



Hình 2.4.2c. Sơ đồ khối DRAM

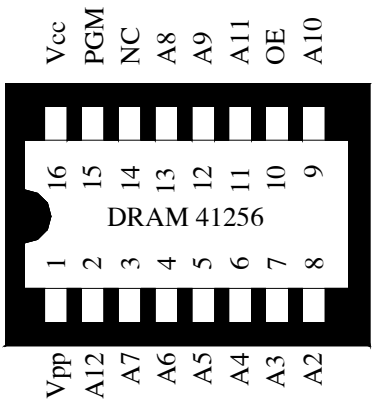
Sau đây là đặc tính của một số bộ nhớ RAM thông dụng:

16K SRAM 6116



E	G	W	Mode	SUPPLY CURRENT	I/O PIN
H	x	x	Not Selected	Iss	Hi-Z
L	H	H	Dout Disable	Icc	Hi-Z
L	L	H	Read	Icc	Dout
L	x	L	Write	Icc	Din

Bộ nhớ RAM động DRAM 41256



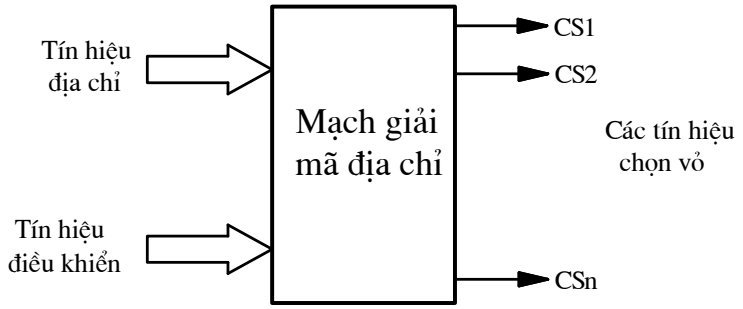
A0 ÷ A7	Address Inputs
CAS\	Column Address Strobe
I/O1 ÷ I/O4	Data Input / Output
RAS\	Row Address Strobe
WRITE	Read/Write Input
OE\	Output Enable
Vcc	Power (+5V)
Vss	Ground

2.4.4. MẠCH GIẢI MÃ ĐỊA CHỈ

Mỗi mạch nhớ nối ghép với CPU cần phải được CPU quy chiếu tới một cách chính xác khi thực hiện các thao tác đọc ghi. Điều đó có nghĩa là mỗi mạch nhớ phải được gán cho một vùng riêng biệt có địa chỉ xác định nằm trong không gian địa chỉ tổng thể của bộ nhớ. Việc gán địa chỉ cụ thể cho mạch nhớ được thực hiện nhờ một xung chọn vỏ được lấy từ mạch giải mã địa chỉ. Khi CPU muốn thực hiện trao đổi thông tin với bộ nhớ hay thiết bị ngoại vi, nó đưa ra địa

chỉ của thiết bị cần trao đổi trên BUS địa chỉ, sau đó qua môt bộ giải mã địa chỉ sẽ xuất tín hiệu chọn chip CS (Chip Select) hoặc tín hiệu cho phép CE (Chip Enable) gửi tới bộ nhớ hay thiết bị vào/ra cần thiết. Bộ nhớ hay thiết bị vào/ra nào nhận được tín hiệu chọn sẽ được phép trao đổi dữ liệu với CPU còn các mạch giao tiếp khác sẽ bị cấm (đầu ra ở trạng thái Hi – Z).

Về nguyên tắc một bộ giải mã địa chỉ thường có cấu tạo như sau (hình 2.4.4)



Hình 2.4.4. Mạch giải mã địa chỉ tổng quát

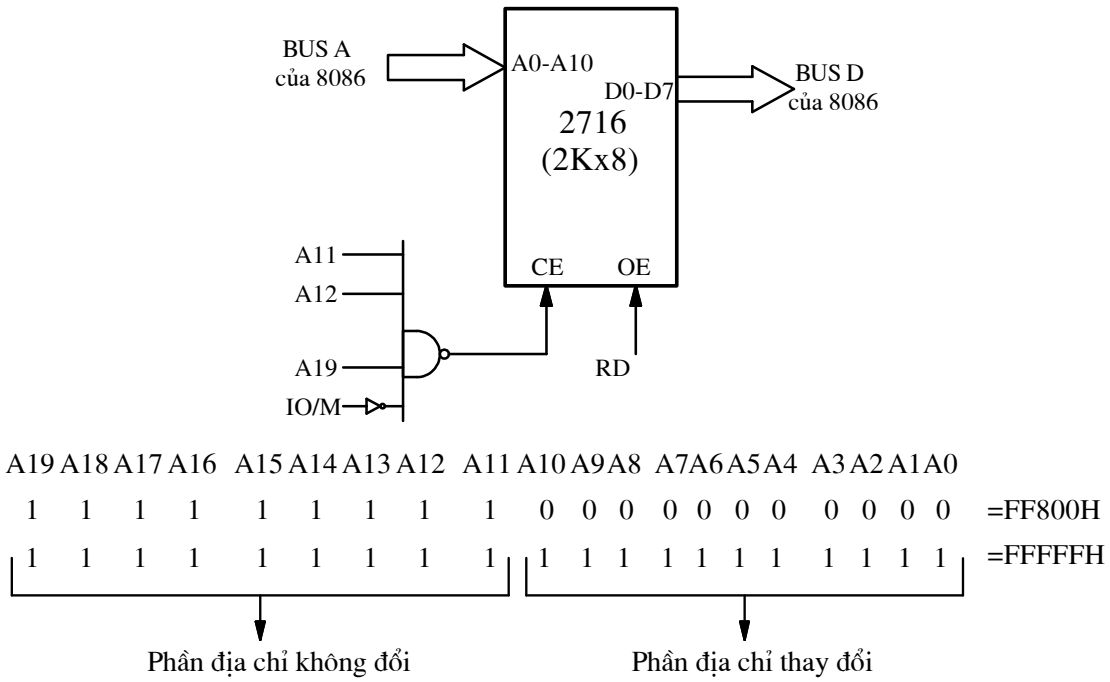
Trong phần này ta đưa ra 3 cách giải mã địa chỉ:

- Giải mã địa chỉ bằng các mạch NAND.
- Giải mã địa chỉ bằng mạch giải mã 74LS138.
- Giải mã địa chỉ dùng PROM.

2.4.4.1. Thực hiện giải mã bằng các mạch NAND

Bằng các mạch cửa NAND ta có thể xây dựng được mạch giải mã địa chỉ đơn giản với số đầu ra hạn chế. Ta phải đưa đến đầu vào của mạch cửa nhiều lối vào một tổ hợp thích hợp của các bit địa chỉ để nhận được ở đầu ra của nó tín hiệu chọn vỏ cho mạch nhớ (hình 2.4.4a).

Đây là mạch giải mã cho EPROM 2716, xung chọn vỏ sẽ tác động khi ta đọc bộ nhớ có địa chỉ trong khoảng FF800H – FFFFFH. 9 bit địa chỉ phần cao của Bus địa chỉ (từ A11 – A19) ở mức 1 sẽ phối hợp cùng xung IO/M (đã được đảo) để tạo ra xung chọn vùng 2KB đặt tại địa chỉ cao nhất trong không gian địa chỉ của 8086. Mỗi ô nhớ cụ thể trong 2KB của mạch nhớ EPROM 2716 sẽ do các bit thấp còn lại của Bus địa chỉ (A0 – A10) chọn ra.

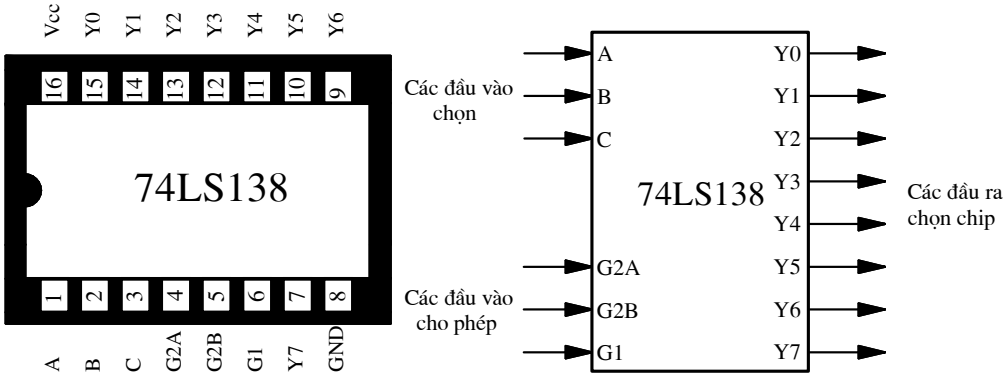


Hình 2.4.4a: Mạch giải mã dùng mạch NAND

2.4.4.2. Thực hiện giả mã bằng mạch giải mã 74LS138

Khi ta muốn có nhiều đầu ra chọn vỏ từ bộ giải mã mà vẫn dùng các mạch logic thì khi thiết kế mạch sẽ rất cồng kềnh do phải sử dụng số lượng các mạch cửa tăng lên. Trong trường hợp như vậy ta thường sử dụng các mạch giải mã có sẵn. Một trong các mạch giải mã được sử dụng rộng rãi nhất là 74LS138. Sơ đồ hình dáng ngoài và bảng chức năng của 74LS138 (Hình 2.4.4b).

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



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

x	x	x	x	1	x	1	1	1	1	1	1	1	1
x	x	x	x	x	0	1	1	1	1	1	1	1	1
0	0	0	0	0	1	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	1	0	0	0	1	1	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1	1	0	1	1	1	1
1	0	0	0	0	1	1	1	1	1	0	1	1	1
1	0	1	0	0	1	1	1	1	1	1	0	1	1
1	1	0	0	0	1	1	1	1	1	1	1	0	1
1	1	1	0	0	1	1	1	1	1	1	1	1	0

x: không quan tâm.

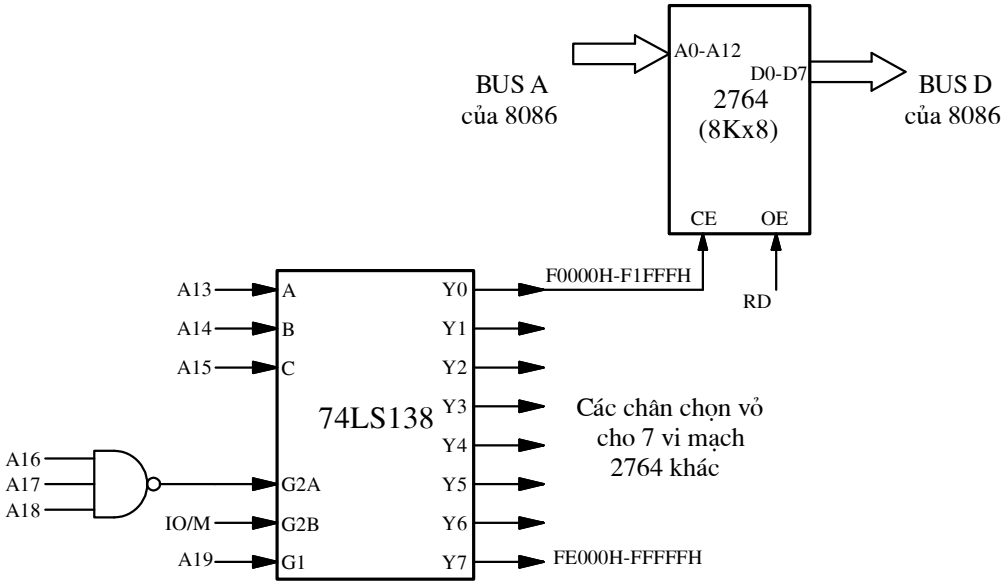
Hình 2.4.4b1. Hình dáng ngoài và bảng chức năng của 74LS138

Từ bảng chức năng ta thấy Y0 = 0 khi mà các đầu vào chọn có C = 0, B = 0, A = 0. Các đầu vào cho phép G2A = 0, G2B = 0, G1 = 1. Vì vậy sơ đồ ghép nối 74LS138 với EPROM 2764 như hình vẽ 2.4.4.b

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

Tín hiệu IO/M đưa vào chân G2B

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



Hình 2.4.4b2. Mạch giải mã dùng 74LS138

Tại ví dụ này ta thấy: Mạch giải mã địa chỉ 74LS138 có số lượng đầu vào địa chỉ và đầu vào cho phép bị hạn chế. Nếu ta có số lượng đầu vào địa chỉ lớn mà ta lại phải giải mã đầy đủ thì để thực hiện bộ giải mã hoàn chỉnh ta vẫn phải dùng thêm các mạch logic phụ. Đây cũng là lý do để người ta

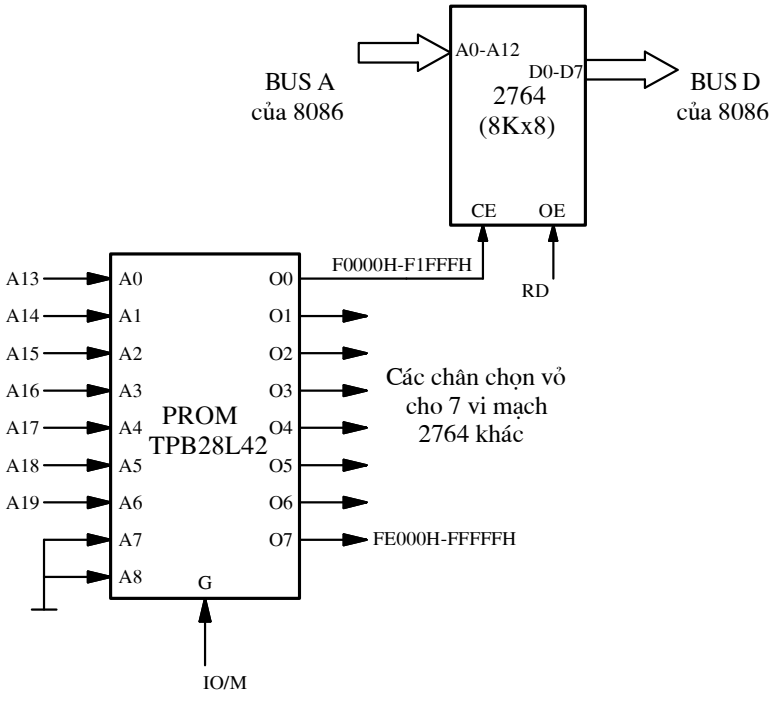
thay thế các bộ giải mã kiểu này bằng các bộ giải mã PROM. Với ưu điểm chính là chúng có rất nhiều đầu vào cho các bit địa chỉ và vì thế rất thích hợp trong hệ vi xử lý sau này với không gian địa chỉ lớn.

2.4.4.3. Thực hiện bộ giải mã dùng PROM

Để giải thích rõ bộ giải mã dùng PROM ta lấy lại ví dụ phân vùng bộ nhớ cho 64KB EPROM trong phần trước. Ta sẽ sử dụng mạch PROM loại TPB28L42 với dung lượng 512byte làm bộ giải mã. Sau đây là bảng chức năng của PROM (chỉ sử dụng 8 byte đầu tiên trong số 512 byte).

Các đầu vào										Các đầu ra							
G	A8	A7	A6	A5	A4	A3	A2	A1	A0	O0	O1	O2	O3	O4	O5	O6	O7
0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1
0	0	0	1	1	1	1	0	0	1	1	0	1	1	1	1	1	1
0	0	0	1	1	1	1	0	1	0	1	1	0	1	1	1	1	1
0	0	0	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1
0	0	0	1	1	1	1	1	0	0	1	1	1	1	0	1	1	1
0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1
0	0	0	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	Các địa chỉ khác									1	1	1	1	1	1	1	1

Mạch giải mã địa chỉ cho bộ nhớ EPROM 64KB dùng PROM được thể hiện trên hình 2.4.4b. So với cách thực hiện bộ giải mã bằng 74LS138 chúng ta không phải dùng đến mạch phụ trợ, điều này làm giảm đáng kể kích thước vật lý của bộ vi xử lý.

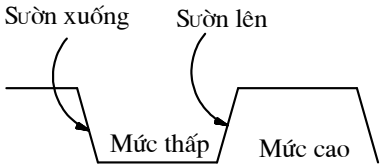


Hình 2.4.4c. Mạch giải mã dùng PROM

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

2.5.1. XUNG NHỊP VÀ CHU KỲ MÁY

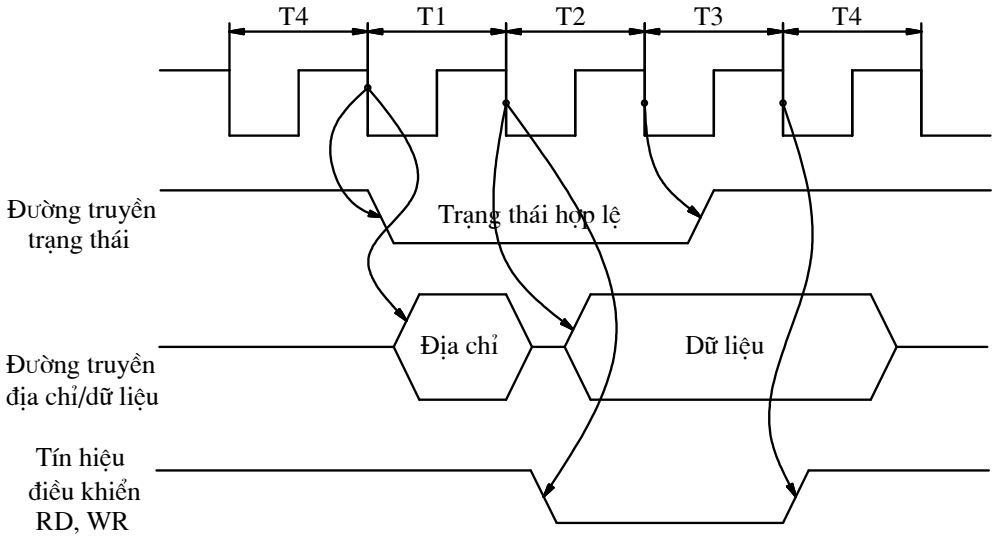
Trên hình 2.5.1a là hình vẽ mô tả một chu kỳ xung nhịp (chu kỳ đồng hồ), một chu kỳ đồng hồ bao gồm 2 pha đối xứng gọi là pha 1 (tín hiệu đồng hồ ở mức thấp) và pha 2 (tín hiệu đồng hồ ở mức cao). Các chu kỳ đồng hồ này được đưa đến lối vào xung nhịp của vi xử lý. Một chu kỳ xung nhịp còn được gọi là một nhịp. Thời gian cần thiết và số xung nhịp cơ sở cho một thao tác của vi xử lý gọi là một chu kỳ máy. Mỗi một chu kỳ máy có 4 nhịp. Hình 2.5.1b là giản đồ thời gian thực hiện chu kỳ bus của vi xử lý 8086.



Hình 2.5.1a: Xung nhịp

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

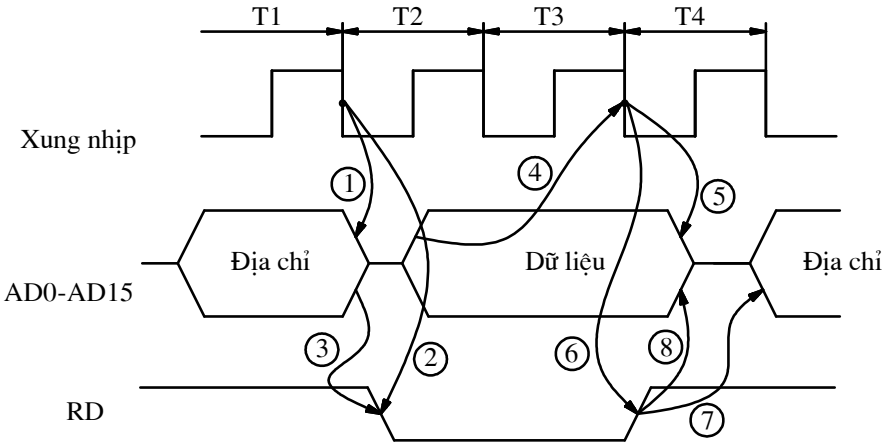
- Pha T1: Các đường trạng thái hoạt động để xác định kiểu thao tác nào được CPU thực hiện, đồng thời các địa chỉ bộ nhớ I/O cũng được truyền.
- Pha T2: Các tín hiệu địa chỉ được thay thế bằng các tín hiệu dữ liệu, các tín hiệu đọc/ghi trở nên tích cực.
- Pha T3: Trước tiên các tín hiệu trạng thái được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



Hình 2.5.1a: Thời gian thực hiện chu kỳ Bus của vi xử lý 8086

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

Hình 2.5.2a chỉ ra một chu kỳ đọc của vi xử lý 8086, ngoài pha 1 được mô tả như ở trên ta cần chú ý đến các pha còn lại. Cũng cần chú ý rằng theo đặc điểm kỹ thuật thì dữ liệu phải tồn tại ít nhất 20 ns trước khi kết thúc T3 và vẫn phải tồn tại ít nhất 10 ns sau khi kết thúc T4. Trong khi đó ta có thể huỷ bỏ đồng thời tín hiệu báo đọc (0 ns).



Hình 2.5.2a: Chu kỳ đọc của vi xử lý

Trong đó các đường:

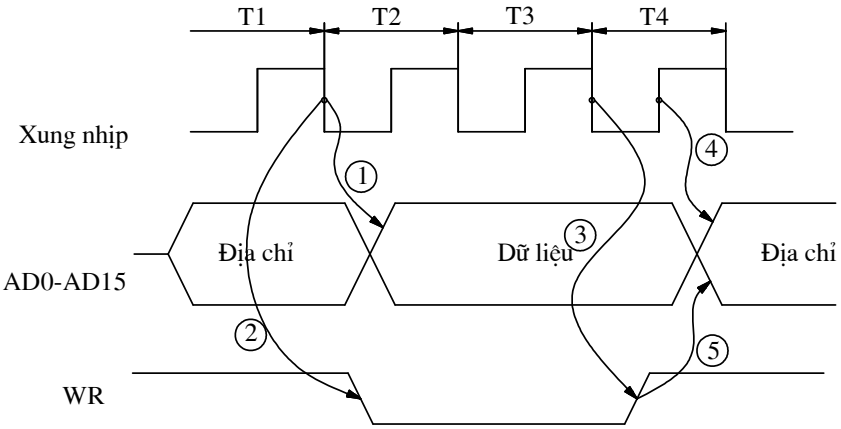
1. t_{CRAZ}: Đồng hồ ở mức thất cho đến khi bus địa chỉ ở trạng thái Hi-Z = 35ns Max.
2. t_{CLRL}: Đồng hồ ở mức thấp cho đến khi RD hoạt động = 70ns Max.
3. t_{AZLN}: Bus địa chỉ được thả nổi cho đến khi RD hoạt động = 0ns Min.
4. t_{OVCL}: Dữ liệu hợp lệ cho đến khi đồng hồ ở mức thấp cho đến khi đồng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

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

$$3*T - t_{trễđịachỉ} - t_{giữ} - t_{trễđệm} = 600 – 110 – 30 – 40 = 420ns$$

Hình 2.5.2b chỉ ra một chu kỳ ghi của vi xử lý 8086, ngoài pha T1 được mô tả như ở trên ta cần chú ý đến các pha sau:

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



Hình 2.5.2b. Chu kỳ ghi của vi xử lý 8086

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

2.6. CÁC CHẾ ĐỘ LÀM VIỆC CỦA VI XỬ LÝ

2.6.1. CHẾ ĐỘ KHỞI ĐỘNG

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

- Khởi động cứng: Là đưa tín hiệu tác dụng vào chân Reset của mạch vi xử lý (thường ở mức thấp). Đối với bộ vi xử lý 8086, sau tín hiệu Reset chương trình được thực hiện từ ô nhớ F000H – FFF0H. Trong đại đa số trường hợp ô nhớ này thường chứa một lệnh nhảy tới một thủ tục thực hiện việc khởi tạo hệ thống.
- Khởi động mềm: Là quá trình khởi động lại (Restart) của mạch vi xử lý sau khi đã thực hiện các lệnh đặc biệt. Mỗi lệnh khởi động có một địa chỉ riêng giữ ở một ô nhớ nhất định trong ROM. Thông thường các hãng sản xuất tổ chức một vùng nhớ đặc biệt để làm nơi chứa các lệnh khởi động này gọi là vùng con chứa vector khởi động.

2.6.2. CHẾ ĐỘ ĐỢI

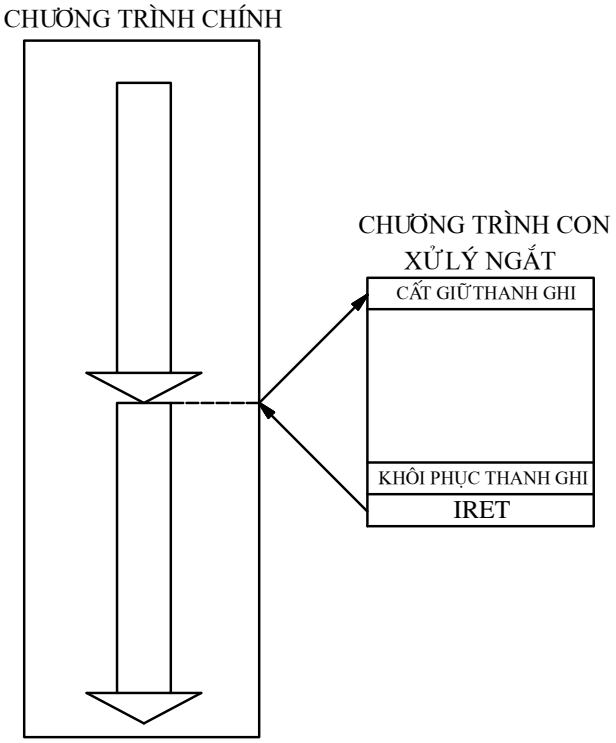
Mục đích của việc vi xử lý làm việc ở chế độ đợi là để phối hợp tốc độ làm việc của vi xử lý với thiết bị ngoại vi. Ví dụ: một CPU thực hiện lệnh mất 500ns, trong khi đó một thiết bị ngoại vi để thâm nhập được số liệu mất 50µs do vậy để tìm được số liệu bên trong ngoại vi cần kéo dài xung đọc bằng cách thêm các chu kỳ đợi (TWAIT) để làm giảm tốc độ hoạt động của Bus trong khi vẫn giữ cho dữ liệu và các tín hiệu điều khiển hoạt động trong toàn bộ thời gian cần thiết cho các thiết bị tốc độ chậm nhận hoặc xuất tín hiệu.

2.6.3. CHẾ ĐỘ TREO

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

2.6.4. CHẾ ĐỘ NGẮT

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



Hình 2.6.4. Hoạt động của chương trình khi có ngắt

2.6.5. CHẾ ĐỘ DỪNG

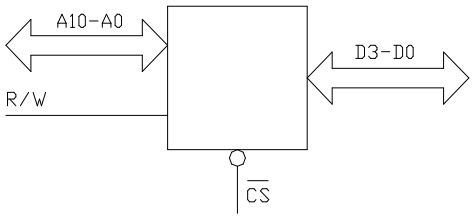
Sau khi thực hiện lệnh dừng (HLT) mạch vi xử lý sẽ thực hiện một chu kỳ máy nào đó (thông qua lệnh NOP). Sau lệnh (NOP) vi xử lý vẫn sẵn sàng làm việc nhưng phải có khoảng trống để nghỉ ngơi. Muốn thoát khỏi chế độ này cần đưa vào lệnh RESET thì vi xử lý sẽ bắt đâu làm việc trở lại.

2.7. MỘT SỐ VÍ DỤ

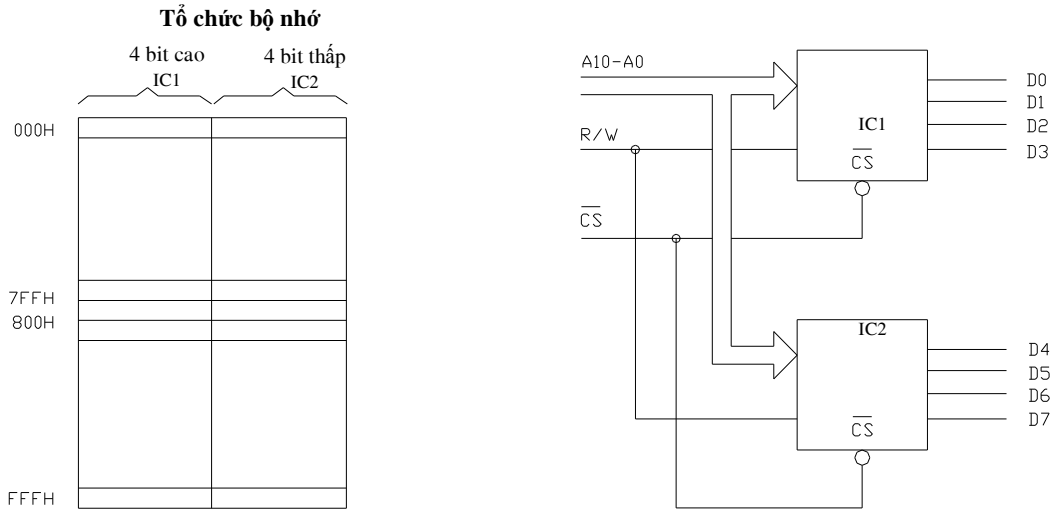
Ví dụ 1: Thiết kế bộ nhớ có dung lượng 2K x 8bit dùng các IC nhớ 2K x 4bit.

Giải:

Bộ nhớ 2K = $2x2^{10} = 2^{11}$ nên cần 11 bit địa chỉ A0÷ A10 và có sơ đồ như hình vẽ



Để thực hiện bộ nhớ 2K x 8bit ta dùng hai IC này đấu song song các đầu địa chỉ và các đầu dữ liệu độc lập. Sơ đồ thực hiện và tổ chức bộ nhớ như hình sau:



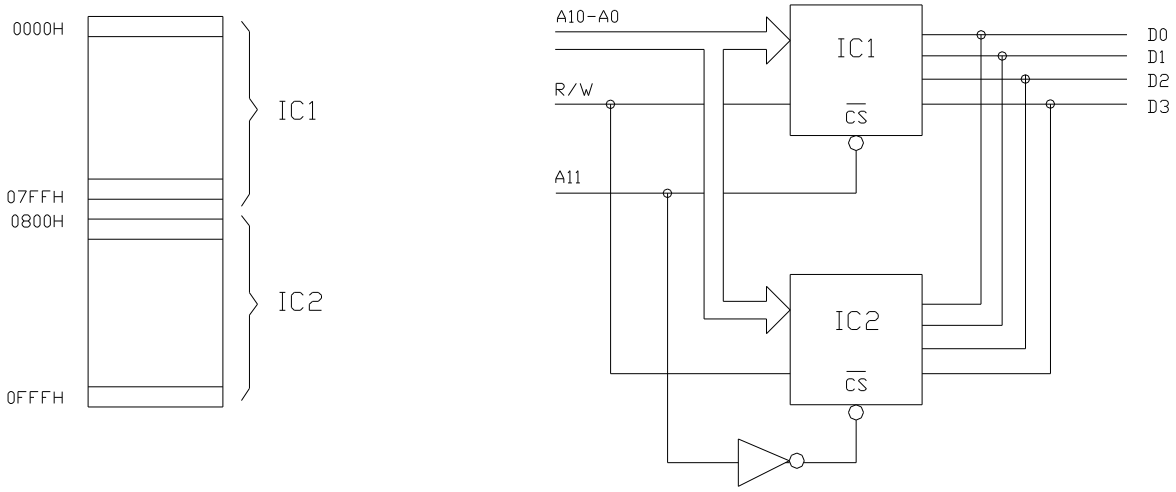
Bộ nhớ 2K x 8bit với các IC nhớ 2K x 4bit

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

Giải:

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

Sơ đồ thực hiện như sau:

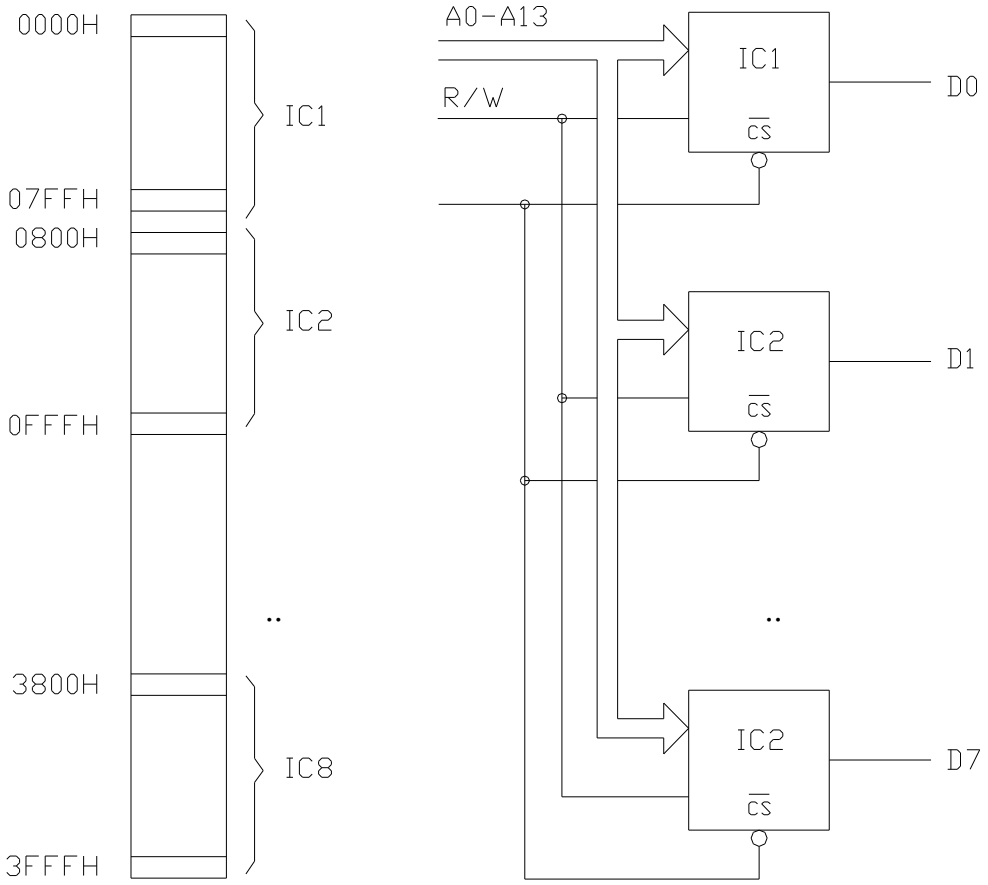


Bộ nhớ 4K x 4bit từ các IC nhớ 2K x 4 bit

Ví dụ 3: Thiết kế bộ nhớ 16K x 8bit dùng các IC nhớ 16K x 1 bit.

Giải:

Bộ nhớ 16K = 24x210 = 214 cần 14 bit địa chỉ A0 ÷ A13. Để thực hiện bộ nhớ 16Kx8bit từ các IC nhớ 16Kx1bit ta dùng 8 IC này đấu song song các đầu địa chỉ còn các đầu dữ liệu độc lập. Sơ đồ thực hiện và tổ chức bộ nhớ như vẽ:

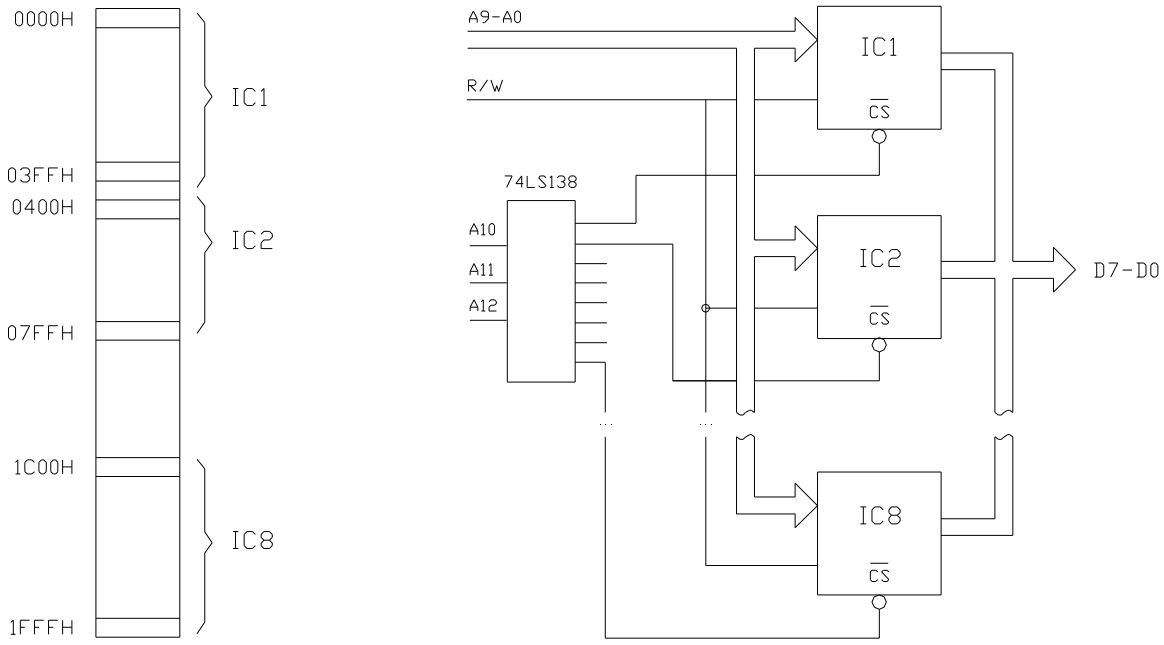


Bộ nhớ 16K x 8bit với các IC nhớ 16K x 1bit

Ví dụ 4: Thiết kế bộ nhớ ROM 8K x 8 bit từ các IC ROM 1K x 8bit.

Giải:

Các IC ROM 1K=210 có 10 bit địa chỉ A0 ÷ A9, còn bộ nhớ 8K =$2^3.2^{10}$ = 2^{13}, cần 13 bit địa chỉ. Như vậy còn thiếu tổ hợp các bit A10, A11, A12. Khi đó ta dùng bộ giải mã địa chỉ 74LS138 (3 đầu vào, tám đầu ra) và thực hiện bộ nhớ như vẽ:



Bộ nhớ 8K x 8bit với các IC nhớ 1K x 8bit

Địa chỉ của các IC cụ thể như bảng sau:

IC	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Địa chỉ
IC1	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	0	0	0	1	1	1	1	1	1	1	1	1	1	03FFH
IC2	0	0	1	0	0	0	0	0	0	0	0	0	0	0400H
	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	0	0	1	1	1	1	1	1	1	1	1	1	1	07FFH
IC3	0	1	0	0	0	0	0	0	0	0	0	0	0	0800H
	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	0	1	0	1	1	1	1	1	1	1	1	1	1	0BFFH
IC4	0	1	1	0	0	0	0	0	0	0	0	0	0	0C00H
	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	0	1	1	1	1	1	1	1	1	1	1	1	1	0FFFH
IC5	1	0	0	0	0	0	0	0	0	0	0	0	0	1000H
	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	1	0	0	1	1	1	1	1	1	1	1	1	1	13FFH
IC6	1	0	1	0	0	0	0	0	0	0	0	0	0	1400H
	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	1	0	1	1	1	1	1	1	1	1	1	1	1	17FFH
IC7	1	1	0	0	0	0	0	0	0	0	0	0	0	1800H
	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	1	1	0	1	1	1	1	1	1	1	1	1	1	1BFFH
IC8	1	1	1	0	0	0	0	0	0	0	0	0	0	1C00H
	..	..	..	..	..	..	..	..	..	..	..	..	..	..
	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH

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

2.8.1. GIỚI THIỆU CHUNG VỀ HỢP NGỮ

Khi đã xây dựng xong phần cứng cho một hệ vi xử lý thì vấn đề còn lại là viết phần mềm cho sự hoạt động của nó. Và như chúng ta đã biết CPU chỉ có thể tính toán được trên các số nhị phân và với tốc độ rất cao. Tuy nhiên đối với con người nếu phải lập trình với các số nhị phân thì thật nhàm chán và chậm chạp. Chương trình chỉ gồm các số 0 và 1 là trình ngôn ngữ máy.

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

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

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

2.8.2. CÚ PHÁP CỦA CHƯƠNG TRÌNH HỢP NGỮ

Trước khi trình bày cách lập trình bằng hợp ngữ ta phải tìm hiểu qua cú pháp của ngôn ngữ này. Nếu chương trình viết đúng về cú pháp sẽ được chương trình dịch dịch ra dưới dạng mã máy. Từ các chương trình mã máy này ta có thể tạo ra được chương trình chạy bằng cách dịch tiếp ra các tệp có đuôi .EXE hoặc .COM.

2.8.2.1. Các trường của hợp ngữ

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

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

- **Nhãn**: Có thể là nơi đến của một lệnh nhảy, hay một lời gọi đến một chương trình con. Các nhãn này sẽ được chương trình dịch gán bằng các địa chỉ cụ thể của ô nhớ. Nhãn có thể dài đến 31 ký tự nhưng thường ngắn hơn nhiều, các ký tự có thể là chữ, số, ký tự đặc biệt nhưng lưu ý rằng ký tự đầu tiên không được là số. Một nhãn thường kết thúc bằng dấu ":".
- **Mã lệnh**: Trong trường mã lệnh nói chung sẽ có các lệnh thật và lệnh giả. Đối với các lệnh thật thì trường này sẽ chứa các mã lệnh gợi nhớ. Mã lệnh này sẽ được chương trình dịch dịch ra dạng mã máy. Đối với các hướng dẫn chương trình dịch thì trường này chứa các thông tin khác nhau liên quan đến các lệnh giả của hướng dẫn.
- **Toán hạng**: Đối với lệnh thật thì trường này chứa các toán hạng của lệnh. Tuỳ theo từng loại lệnh mà ta có thể có 0, 1 hoặc 2 toán hạng trong một lệnh. Trong trường hợp các lệnh có một toán hạng thông thường ta có toán hạng là đích hoặc Nguồn, còn trong trường hợp lệnh với 2 toán hạng thì ta có một toán hạng là Đích và một toán hạng là Nguồn. Đối với các hướng dẫn chương trình dịch thì trường này chứa các thông tin khác nhau liên quan đến các lệnh giả của hướng dẫn.
- **Chú thích**: Ghi vào các lời giải thích cho chương trình và được viết sau dấu ";". Assembler sẽ bỏ qua tất cả các chú thích này khi dịch chương trình

  Ví dụ: Một dòng lệnh gợi nhớ:

```
BACK:   MOV     AH, [BX][SI]    ;nap vao AH noi
        ;dung o nho co dia chi DS:(BX+SI)
```

Trường nhãn có tên là `BACK`, trường mã lệnh có lệnh `MOV`, tại trường toán hạng có thanh ghi `AH, BX, SI` và phần chú thích là dòng chữ "`nap vao AH noi dung o nho co dia chi DS:(BX+SI)`".

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

```
MAIN    PROC
MAIN    ENDP
```

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

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

2.8.2.2. Khai báo biến, hằng, chương trình con

❖ **Khai báo biến**

- **Biến byte**

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

```
Tên   DB      Giá_trị_khởi_đầu
```

Ví dụ:

```
BIEN1         DB      5
BIEN2         DB      ?
BIEN3         DB      ‘$’
```

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

- **Biến Word**

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

```
Tên           DW      Giá_trị_khởi_đầu
```

Ví dụ:

```
BIEN1         DB      50
BIEN2         DB      ?
```

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

- **Biến mảng**

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

Ví dụ 1:

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

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

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

Ví dụ 2:

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

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

Toán tử DUP có thể lồng vào nhau như ví dụ sau

Ví dụ 3:

```
M4    DB      3, 2, 1, 2 DUP(1, 2 DUP(3), 4)
```

Định nghĩa này tương đương như sau:

```
M4    DB      3, 2, 1, 1, 3, 3, 4, 1, 3, 3, 4
```

Khi định nghĩa biến có tên là WORD như ví dụ sau

Ví dụ 4:

```
WORD  DW      0FFEEH
```

Ta phải hiểu rằng byte thấp (EEH) sẽ được cất tại ô nhớ có địa chỉ WORD, còn byte cao cất tại địa chỉ tiếp theo WORD+1.

- **Biến kiểu xâu ký tự**

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

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

```
STR1  DB      'STRING'
STR2  DB      73H, 74H, 72H, 69H, 6EH, 67H
STR3  DB      73H, 74H, 'R', 'I', 6EH, 67H
```

❖ **Khai báo hằng**

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

Ví dụ 1:

```
HANG1 EQU     100
HANG2 EQU     'XAU KY TU'
```

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

Ví dụ 2:

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

❖ **Khai báo chương trình con**

Khi một chương trình cần thiết phải thực hiện một nhóm lệnh nhất định (chúng có thể bao gồm các tham số khác nhau) nhiều lần, thì chúng ta

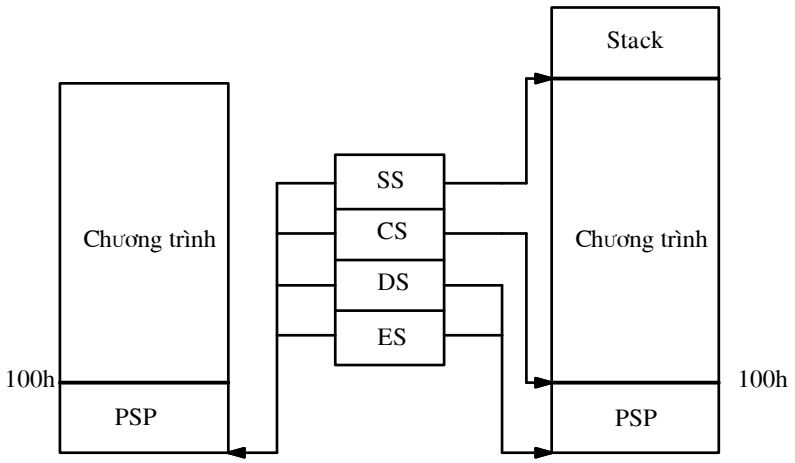
nên nhóm các lệnh đó thành một chương trình con. Sau đó nó được gọi và thực hiện bởi lệnh CALL, kết thúc chương trình con phải có lệnh RET. Sau lệnh RET chúng sẽ trao quyền điều khiển cho chương trình chính tiếp theo tại vị trí mà nó đã đạt tới trước khi thực hiện lệnh CALL.

```
Tên_ctc      Proc
; các lệnh của thân chương trình con
RET
Tên_ctc      Endp
```

2.8.3. KHUÔN DẠNG CỦA CHƯƠNG TRÌNH HỢP NGỮ

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

- Chương trình .EXE là một thủ tục xa (Far Procedure), có thể được gọi từ một đoạn khác. Với chương trình .EXE có thể được định nghĩa 4 đoạn khác nhau: Đoạn lệnh (code), đoạn dữ liệu (data), đoạn ngăn xếp (stack) và đoạn thêm (extra).
- Khác với chương trình .EXE chương trình .COM là một thủ tục gần (Near Procedure), chỉ gọi được từ bên trong đoạn lệnh và chỉ có thể được định nghĩa với duy nhất một đoạn lệnh, do đó phải đặt đoạn dữ liệu và ngăn xếp vào bên trong đoạn lệnh này.



2.8.3.1. Chương trình .EXE

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

```
MOV   AX, data
MOV   DS, AX
```

Khung cho một chương trình .EXE có dạng sau:

```
TITLE Tên_chương_trình
COMMENT      *
             Bắt đầu của một khối chú thích
             *
;Khai báo đoạn stack
stack SEGMENT STACK
     ;Khai báo kích thước Stack
stack ENDS
;Khai báo đoạn dữ liệu
data  SEGMENT
     ;Định nghĩa các dữ liệu
data  ENDS
;Khai báo đoạn mã
code  SEMENT
      ASSUME  CS:code, DS:data
Start:
      ;Các lệnh
code  ENDS
END   Start
```

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

Ví dụ: Viết chương trình .EXE in ra màn hình dòng chữ "DHKTCN".

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

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

Vào: AH=1

Ra: AL=mã ASCII củ ký tự

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

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

Vào: AH=2

DL=mã ASCII của ký tự cần hiển thị

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

Vào: AH=9

DX=địa chỉ lệch của chuỗi ký tự

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

Vào: AH=4CH

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

Chương trình được viết như sau:

```
TITLE VIDU_EXE
COMMENT      *
in ra man hinh dong chu DHKTCN
Chuong trinh thuoc dang .EXE
             *
stack SEGMENT STACK
      DB     20 DUP('STACK')
stack ENDS
data  SEGMENT
STRING       DB     'DHKTCN!$'
data  ENDS
code  SEGMENT
      ASSUME CS:code, DS:data
Start:
;Dat thanh ghi DS
      MOV    AX, data
      MOV    DS, AX
;In ra man hinh
      MOV    AH, 9
      LEA    DX, STRING    ;chuyen   dia   chi
                    ;lech cua xau vao DX
      INT    21h
      MOV    AH, 4Ch
      INT    21h
code  ENDS
END   Start
```

Trong chương trình trên phần chỉ thị:

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

Là phần khai báo đoạn ngăn xếp có tên là STACK. Ngăn xếp có kích thước là 20x5=100 byte và lúc khởi đầu được lấp đầy bằng từ STACK.

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

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

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

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

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

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

2.8.3.2. Chương trình .COM

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

Khung cho một chương trình .COM có dạng sau:

```
TITLE Tên_chương_trình
COMMENT      *
             Bắt đầu của một khối chú thích
             *
;Khai báo đoạn mã
code  SEGMENT
      ASSUME  CS:code, DS:code
      ORG     100h
Start:
      ;Các lệnh
code  ENDS
END   Start
```

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

```
TITLE VIDU_COM
```

```
COMMENT      *
in ra man hinh dong chu DHKTCN
Chuong trinh thuoc dang .COM
             *
code  SEGMENT
      ASSUME CS:code, DS:code
      ORG    100h
Start:
;In ra man hinh
      MOV    AH, 9          ;ham cua INT   21h
      LEA    DX, STRING     ;lay dia chi offset
                            ;cua xau
      INT    21h
      MOV    AH, 4Ch
      INT    21h
STRING       DB     'DHKTCN!$'
code  ENDS
      END    Start
```

2.8.4 CẤU TRÚC LẬP TRÌNH CƠ BẢN BẰNG HỢP NGỮ

2.8.4.1. Cấu trúc tuần tự

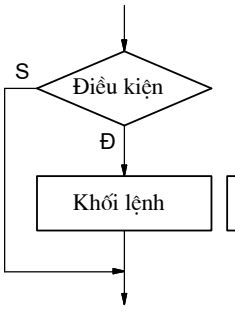
Kiểu cấu trúc này các lệnh được viết tuần tự, lệnh nọ tiếp lệnh kia. Đây cũng là cấu trúc thông dụng và đơn giản nhất.

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

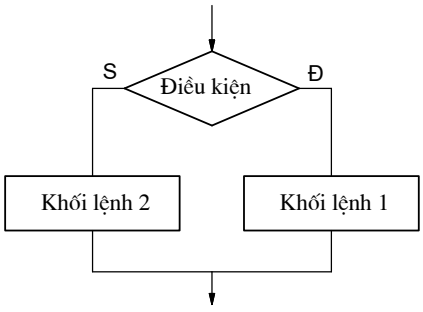
```
XOR   AX, AX ;Xoa thanh ghi AX
ADD   AX, BX ;cong thanh ghi AX voi b
ADD   AX, CX ;cong tiep voi c
SAL   AX, 1  ;nhan doi ket qua a=2(b+c)-> AX
```

2.8.4.2 Cấu trúc rẽ nhánh

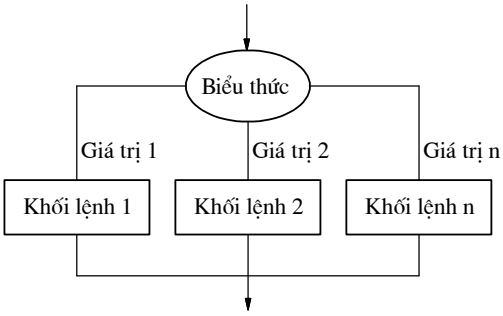
Câu lệnh rẽ nhánh giúp quyết định nhánh nào của chương trình được thực hiện căn cứ vào điều kiện được đặt ra.



Hình 4.2.4a



Hình 4.2.4b



Hình 4.2.4c

❖ **Cấu trúc rẽ nhánh một khối lệnh (hình 4.2.4a)**

Nếu điều kiện được thoả mãn thì khối lệnh được thực hiện, nếu không khối lệnh sẽ được bỏ qua và thực hiện lệnh tiếp theo sau khối này (hình 4.2.4a). Điều này tương đương với với việc dùng lệnh nhảy có điều kiện để bỏ qua một số thao tác nào đó trong chương trình hợp ngữ.

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

```
     CMP   BX, 0   ;so sanh BX voi 0
     JNL   GAN     ;nhay den GAN neu BX≥0
     NEG   BX      ;neu BX<0 thi dao dau cua BX roi
GAN:MOV    AX, BX ;GAN
```

❖ **Cấu trúc rẽ nhánh hai khối lệnh (hình 4.2.4b)**

Nếu thoả mãn điều kiện 1 thì thực hiện khối lệnh 1, nếu không thì thực hiện khối lệnh 2 (hình 4.2.4b). Điều này tương đương với việc dùng lệnh nhảy có điều kiện và không điều kiện để nhảy đến các nhánh nào đó trong chương trình hợp ngữ.

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

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

Ví dụ 2: Gán cho BL bit dấu của AX

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

❖ **Cấu trúc rẽ nhánh nhiều khối lệnh (hình 4.2.4c)**

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

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

Nếu a>b thì CX=2

Nếu a=b thì CX=0

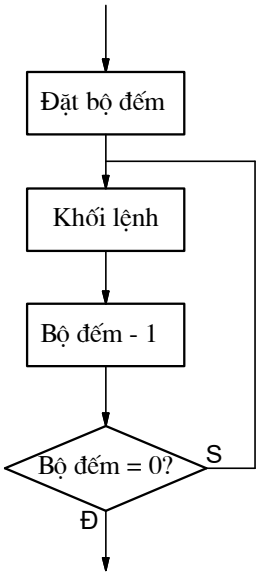
Nếu a<b thì CX=1

Chương trình như sau:

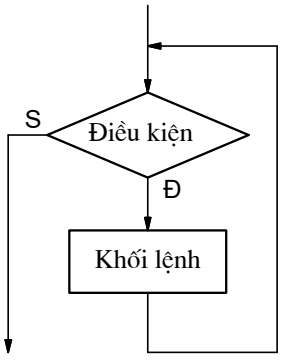
```
      CMP     AX, BX  ;so sanh AX va BX
      JG      GAN2    ;den GAN2 neu AX>BX
      JE      GAN0    ;den GAN0 neu AX=BX
      JL      GAN1    ;den GAN1 neu AX<BX
GAN0: MOV     CX, 0   ;CX←0
      JMP     THOAT   ;roi thoat
GAN1  :MOV    CX, 1   ;CX←1
      JMP     THOAT   ;roi thoat
GAN2: MOV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

2.8.4.3. Cấu trúc lặp

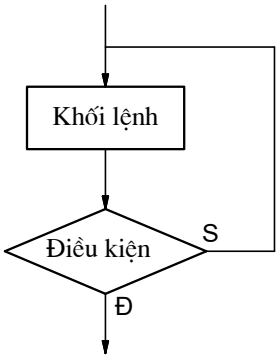
Cấu trúc lặp là cấu trúc mà trong đó một hoặc nhiều lệnh được thực hiện lặp đi lặp lại.



Hình 4.2.4d



Hình 4.2.4e



Hình 4.2.4f

❖ **Lặp với số lần biết trước (hình 4.2.4d)**

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

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

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

❖ **Lặp với số lần không biết trước**

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

- *Lặp kiểm tra điều kiện trước (hình 4.2.4e).*

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

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

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

- *Lặp kiểm tra điều kiện sau (hình 4.2.4f)*

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

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

```
MOV   AL, FFh          ;AL←FFh
```

```
     MOV   DX, 1280h      ;dat dia chi cong ra
     OUT   DX, AL ;dua so lieu trong AL ra cong
LAP:MOV   DX, 1180h      ;dat dia chi cong vao
     IN    AL, DX         ;doc dịa chi cong vao
     CMP   AL, 0          ;AL=0?
     JE    GAN            ;dung thi nhay den GAN
     JMP   LAP            ;sai thi doc lai
GAN:MOV AL, 0            ;AL←0
     MOV   DX, 1280h      ;dat dia chi cong ra
     OUT   DX, AL ;dua so lieu trong AL ra cong
     NOP
```

Ví dụ 2: Đọc ký tự từ bàn phím, khi gặp ký tự $ thì thôi.

```
          MOV    AH, 1   ;ham doc ky tu tu ban phim
     LAP: INT    21H     ;doc mot ky tu, AL chua ma
                         ;ky tu
          CMP    AL, '$';so sanh ky tu doc voi $
          JNE    LAP     ;tro lai LAP
     THOAT:              ;thoat
```