MỤC LỤC

YÊU CẦU NỘI DUNG CHI TIẾT

Tên học phần: **Kỹ thuật Vi xử lý** Loại học phần: **2**
Bộ môn phụ trách giảng dạy: **Kỹ thuật Máy tính** Khoa phụ trách: **CNTT**
Mã học phần: **17301** Tổng số TC: **2**

TS tiết	Lý thuyết	Thực hành/Xemina	Tự học	Bài tập lớn	Đồ án môn học
45	30	15	0	0	0

Điều kiện tiên quyết:

Sinh viên phải học xong các học phần sau mới được đăng ký học phần này:
Kiến trúc máy tính, Điện tử số, Mạch và tín hiệu

Mục tiêu của học phần:

- Cung cấp các kiến thức cơ bản về cấu trúc, nguyên lý hoạt động của hệ vi xử lý.
- Nắm được các phương thức điều khiển vào/ra dữ liệu.
- Hiểu rõ nguyên tắc, cách thức phối ghép cơ bản.

Nội dung chủ yếu

- Các vấn đề cơ bản của các bộ vi xử lý
- Tập lệnh, Mode địa chỉ, lập trình điều khiển hệ thống
- Các thành phần và ghép nối vi xử lý với khối vào ra cơ bản

Nội dung chi tiết của học phần:

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	**LT**	**TH/Xemina**	**BT**	**KT**
Chương I. Tổng quan	**5**	**5**			
1.1. Lịch sử phát triển các bộ VXL		0,5			
1.2. Bộ Vi xử lý		0,5			
1.3. Hệ Vi xử lý		0,5			
1.4. Các đặc điểm cấu trúc của hệ Vi xử lý		2			
1.5. Hệ lệnh của bộ vi xử lý		1			
1.5. Các vi mạch hỗ trợ hệ vi xử lý		0,5			
Chương II. Cấu trúc của bộ vi xử lý	**3**	**3**			
2.1.Bộ vi xử lý cấp thấp (8 bit)		1			
2.2. Bộ vi xử lý cấp cao (16 bit)		2			
Chương III. Bộ vi xử lý 8088	**5**	**5**			
3.1. Cấu trúc bộ vi xử lý 8088		1			
3.2. Tổ chức bộ nhớ		1			
3.3. Tập các thanh ghi		1			
3.4. Ngắt		1			
3.5. Mã hóa lệnh		1			
Chương IV. Lập trình hệ thống	**19**	**4**	**15**		
4.1. Tổng quan, cấu trúc hợp ngữ (Assembly)		0.5			
4.2. Dữ liệu trong Assembly		0.5			
4.3. Vào/ra trong Assembly		0.5			
4.4. Nhóm lệnh dịch chuyển dữ liệu		0.5			
4.5. Nhóm lệnh tính toán số học		0.5			
4.6. Nhóm lệnh chuyển điều khiển		0.5			
4.7. Nhóm lệnh lặp		0.5			
4.8. Nhóm lệnh dịch chuyển và quay		0.5			
Chương V. Tổ chức vào/ra dữ liệu	**8**	**8**			
5.1.1 Các tín hiệu của 8088		1			

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	LT	TH/Xemina	BT	KT
5.1.2. Phân kênh		0,5			
5.1.3. Mạch tạo xung nhịp 8284		0.5			
5.1.4. Mạch điều khiển BUS 8288		0,5			
5.1.5. Biểu đồ thời gian của các lệnh ghi/đọc		0.5			
5.2.1. Bộ nhớ bán dẫn		0,5			
5.2.2. Giải mã địa chỉ cho bộ nhớ		0,5			
5.2.3. Phối ghép 8088 với bộ nhớ		0,5			
5.3.1. Các kiểu phố ghép vào/ra		0,5			
5.3.2. Giải mã địa chỉ cho thiết bị vào/ra		0.5			
5.3.4. Mạch phối ghép 8255A		0.5			
5.3.5. Lập trình với 8255A		2			
Chương V1. Phối ghép 8088 với TBNV	**3**	**3**			
6.1. Vào/ra dữ liệu bằng cách thăm dò		0,5			
6.2.1. Vào/ra dữ liệu bằng ngắt		0,5			
6.2.2. Mạch điều khiển ngắt ưu tiên 8259A		0,5			
6.3.1. Vào/ra dữ liệu bằng DMA		0,5			
6.3.2. DMAC 8237A		1			
Chương VII. Một số phối ghép cơ bản	**2**	**2**			
9.1. Phối ghép với bàn phím		0,5			
9.2. Phối ghép với đèn LED		1			
9.3. Phối ghép với màn hình		0,5			

Nhiệm vụ của sinh viên :

Tham dự các buổi thuyết trình của giáo viên, tự học, tự làm bài tập do giáo viên giao, tham dự các buổi thực hành, các bài kiểm tra định kỳ và cuối kỳ.

Tài liệu học tập :

- Văn Thế Minh, Kỹ thuật vi xử lý, NXB Giáo dục 1997.
- Đỗ Xuân Thụ & Hồ Khánh Lâm, Kỹ thuật Vi xử lý và máy vi tính, NXB Giáo dục 2000.
- Nguyễn Tăng Cường & Phan Quốc Thắng, Cấu trúc và lập trình họ vi điều khiển 8051
- Charles M.Gilmore McGraw, Microprocessors Principles and Application, Hill International Edition 1995

Hình thức và tiêu chuẩn đánh giá sinh viên:

- Hình thức thi cuối kỳ : Thi viết, rọc phách.
- Đánh giá dựa trên tình hình tham dự buổi học trên lớp, các buổi thực hành, điểm kiểm tra thường xuyên và điểm kết thúc học phần.

Thang điểm: Thang điểm chữ A, B, C, D, F.

Điểm đánh giá học phần: Z = 0,3X + 0,7Y. ***X = (X1 + X2 + X3)/3***

Bài giảng này là tài liệu **chính thức và thống nhất** của **Bộ môn Kỹ thuật máy tính, Khoa Công nghệ Thông tin** và được dùng để giảng dạy cho sinh viên.

Ngày phê duyệt: / / 2012

Bộ phận biên soạn	**Trưởng Bộ môn**	**Trưởng đơn vị**
(*Ký, ghi rõ họ tên)*	**(***Ký, ghi rõ họ tên)*	**(***Ký, ghi rõ họ tên)*

Chương 1 : HỆ THỐNG VI XỬ LÝ

1. 1. Lịch sử phát triển

Vào giữa những năm 1970, bộ vi xử lý được định nghĩa là một Đơn vị xử lý trung tâm (CPU), nó được chế tạo bởi công nghệ mạch tổ hợp cỡ lớn VLI (large-scale intergration) với khoảng 50000 transistor trong một đơn vị mạch (Chip) và làm việc với trong khoảng tần số từ 1-5 MHz.

Mẫu vi xử lý đầu tiên là bộ vi xử lý 4 bit được hãng Intel cho ra đời năm 1971 với tên Intel 4004.Bộ vi xử lý này chủ yếu dùng để chế tạo các máy tính để bàn và bỏ túi (calculators), nó dễ dàng thức hiện các tính toán ở mã BCD(Binary coded decimal).

Năm 1972, hãng Intel cho ra đời bộ vi xử lý 8 bit Intel 8008.

Năm 1974, Chip Intel 8080 8 bit được giới thiệu và trở thành cơ sở để chế tạo các máy tính cá nhân sử dụng hệ điều hành CP/M. Nhà sản xuất ZILOG dựa trên cở sở của 8080 để cho ra đời bộ vi xử lý Z80/8 bit, nó bao gồm các lệnh của 8080, nhưng có thêm một số lệnh mới và có nhiều chức năng hay hơn. Điều này có nghĩa là chương trình viết trên 8080 có thể thể chạy bình thường trên trên Z80.

Năm 1978, Intel sản xuất các họ vi xử lý 8086 16-bit, năm 1979 bộ vi xử lý Intel 8088 ra đời với 16 bit trong và data bus 8 bit, dòng vi xử lý này với giá thành rẻ đã tìm được ứng dụng thực tế trong các máy tính cá nhân bán chạy trên thị trường. Tiếp theo 8086 và 8088 là 80186 với một số lện bổ xung.

Năm 1982, Intel tuyên bộ sự ra đời của 80286 16 bit bus dữ liệu bên trong và bên ngoài, nó trở thành bộ vi xử lý cho các loại máy vi tính AT.

Năm 1985, Intel giới thiệu bộ vi xử lý 80386 32 bit (i386), cùng với 80376 32 bit dữ liệu bên trong nhưng 16 bit dữ liệu bên ngoài. Tiếp theo là Intel 80486, hay còn gọi là bộ vi xử lý giá rẻ, và đạt tần số làm việc 66MHz ở loại i486 DX2. Điểm đặc biệt là i486 có đơn vị xử lý dấu phẩy động FPU và 8KB Cache memory trong chip trong khi giữ nguyên cấu trúc của i386.

Năm 1993 ra đời loạt vi xử lý Intel Pentium, sau đó là i586 hay P5, nhưng Intel vẫn đặt tên là Pentium, chúng có 64 bit dữ liệu bên trong và bên ngoài, 16 KB cache memory bên trong chip (8 KB cho lệnh và 8 KB cho dữ liệu).

Các bộ vi xử lý hiện nay là các hệ thống 32 bit và 64 bit đầy đủ và chúng được đóng vỏ trong các chip vi mạch VLSI (very large-scale integration) với hàng chục triệu transistors. Các bộ vi xử lý thực hiện các lệnh trong một chu kỳ, có đơn vị xử lý dấu phẩy động FPU (Floating-point Unit) bên trong, có các thanh ghi chung 16-32-64 bit. Nhiều loại có phân biệt các tệp thanh ghi 32-64 bit (register file) cho đơn vị nguyên IU (Interger Unit) và tệp thanh ghi 32-64 bit cho FPU. Chúng đều có Cache memory bên trong với dung lượng lớn và đa phần được phân đôi : dùng cho lệnh Icache (Instruction cache) và dùng cho dữ liệu Dcache (Data cache). Các bộ vi xử lý công nghệ cao hiện nay (advanced microprocessors) đã thỏa mãn được yêu cầu chế tạo các máy tính lớn (mainframes) và các siêu máy tính (supercomputers).

Công nghệ vi mạch tích hợp đã phát triển từ nhữn năm 1960, vào cuối những năm 1960 đầu 1970 đã xuất hiện các vi mạch với công nghệ LSI, và vào những năm 1980 đã có những chip IC công nghệ VLSI với mật độ trên 100.000 transistors. Ngày nay các chip vi xử lý công nghệ VLSI có hàng chục triệu transistors.

Các thế hệ vi xử lý :

- ✓ Thế hệ 1 (1971- 1973) :
  - Độ dài từ thường là 4 bit.
  - Công nghệ chế tạo : PMOS.
  - Tốc độ rất thấp : 10-60 µs/lệnh

- Tập lệnh đơn giản, cần nhiều vi mạch phụ trợ

✓ Thế hệ 2 (1974 -1977) :
- Vi xử lý 8 bit.
- Công nghệ NMOS hoặc CMOS.
- Tốc độ : 1-8 µs/lệnh

✓ Thế hệ 3 (1978- 1982)
- Vi xử lý 16 bit.
- Tập lệnh đa dạng, vùng nhớ lớn.
- Công nghệ chế tạo : HMOS.
- Tốc độ : 0.1-1 µs/lệnh

✓ Thế hệ 4 (1983 - ?) :
- Vi xử lý 32-64 bit.
- Công nghệ HCMOS
- Có bộ nhớ ảo . . .

1.2. Bộ vi xử lý

Trong các thế hệ máy tính số, khối xử lý trung tâm CPU (Central processing unit) thực hiện các chức năng chính quyết định tên gọi của máy tính : đó là thực hiện các phép xử lý dữ liệu như các phép tính số học, logic.. các trao đổi dữ liệu bộ nhớ, với các ngoại vi.. Khối CPU là tập hợp các logic điều khiển và các thanh ghi gồm nhiều vi mạch với mực độ tích hợp khác nhau. Cũng giống như CPU, bộ vi xử lý có các mạch số thực hiện xử lý và tính toán dữ liệu dưới sự điều khiển của chương trình. Nhưng điều khác cơ bản đối với CPU bình thường ở chỗ là toàn bộ các mạch số của bộ vi xử lý được tích hợp trong một chip vi mạch tích hợp mật độ cao LSI hoặc VLSI, điều này làm tăng tốc độ xử lý lên rất nhiều vì sự trễ tín hiệu do liên kết giữa các mạch số không đáng kể nữa.

Cũng như CPU của các máy tính số, chức năng chính của bộ vi xử lý là xử lý dữ liệu, bao gồm tính toán (computing) và thao tác với dữ liệu (handling).

Quá trình tính toán được tập hợp các vi mạch logic, thường gọi là ALU(đơn vị số học và logic) thực hiện. Đó là các lệnh số học căn bản như : ADD, SUB, MUL, DIV, các phép Logic, các phép so sánh …

Để xử lý dữ liệu, bộ vi xử lý cần phải có đơn vị logic điều khiển CU để chỉ bảo bộ vi xử lý phải giải mã và thực hiện lệnh theo chương trình như thế nào. Chương trình là tập hợp các lệnh máy (instruction set) mà bộ vi xử lý có thể thực hiện được. Chương trình được lưu trong bộ nhớ chính của máy tính nằm bên ngoài chip vi xử lý. Điều này có nghĩa là phải đọc các lệnh (instruction fetch) của chương trình vào bộ vi xử lý để thực hiện. Lệch đọc vào bộ vi xử lý được giải mã (decode) để tạo ra những chuỗi những tín hiệu điều khiển từ CU theo thời gian để thực hiện các phép xử lý dữ liệu do lệnh yêu cầu.

Như vậy, xử lý dữ liệu của bộ vi xử lý đòi hỏi 3 bước :

✓ Bước thứ nhất : đọc lệnh (instruction fetch).

✓ Bước thứ hai : CU giải mã lệnh (instruction decode).

✓ Bước thứ ba : ALU thực hiện lệnh

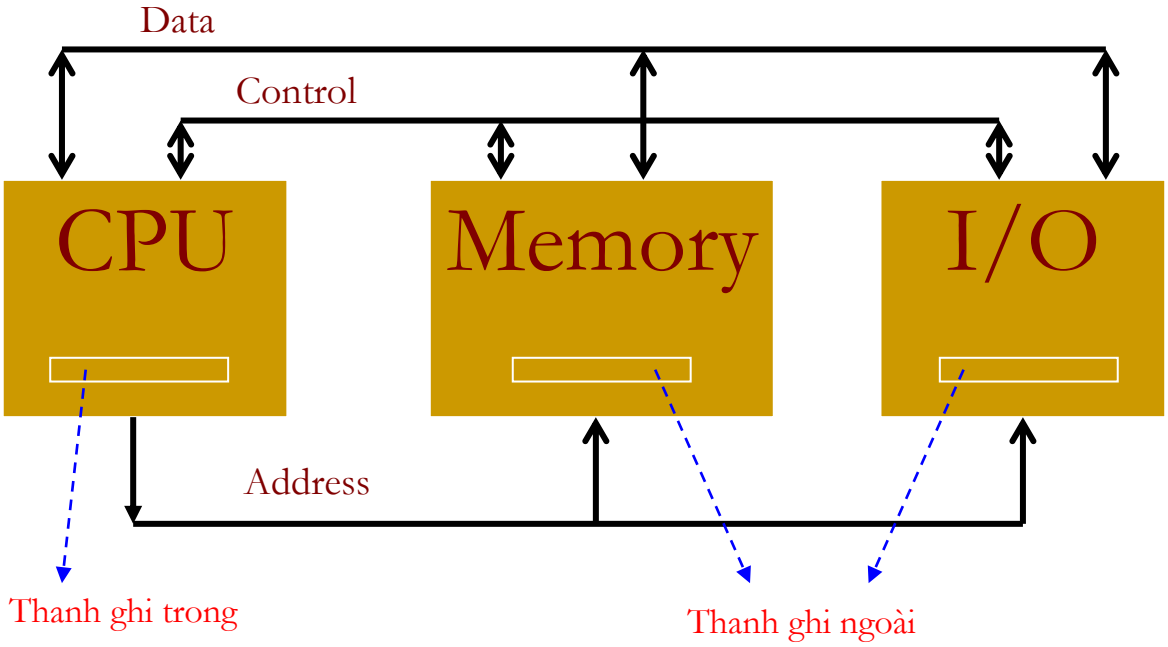
Toàn bộ các bước này hoàn thành một chu kỳ đọc và thực hiện lệnh (Fetch and execute cycle).

Đơn vị CU của bộ vi xử lý còn phải thực hiện điều khiển giao tiếp với ccs hệ thống bên ngoài của bộ vi xử lý : bộ nhớ, các thiết bị vào ra. Sự trao đổi dữ liệu giữa bộ vi xử lý và các thiết bị bên ngoài cũng cần phải có các mạch Input/Output (I/O circuit).\

Trong bộ vi xử lý, các đơn vị liên kết dữ liệu với nhau thông qua bus bên trong (internal bus), và sự liên kết với thế giới bên ngoài thực hiện bằng các bus bên ngoài (system bus).

1.3. Hệ vi xử lý

Hệ vi xử lý và bộ vi xử lý là hai khái niệm khác nhau. Bộ vi xử lý là một hay vài vi mạch tổ hợp lớn với chức năng chủ yếu là xử lý dữ liệu và điều khiển. Hệ vi xử lý là một hệ thống tính toán với đơn vị xử lý trung tâm là bộ vi xử lý và các đơn vị ngoài kết nối với nó.



Một hệ thống máy tính bao gồm các thành phần như hình vẽ trên (theo nguyên lý Von Newman cổ điển). Các thành phần trên không thể đứng rời rạc với nhau, cần có sự liên kết giữa các thành phần để tạo thành 1 khối thống nhất hoàn chỉnh. Hệ vi xử lý thực tế chính là một hệ thống máy tính được xét theo góc độ kỹ thuật ghép nối các thành phần lại với nhau để đạt được sự đồng bộ, ổn định và công suất cao.

Trong 1 hệ vi xử lý, bộ vi xử lý đóng vai trò quan trọng nhất, các thành phần còn lại phải có khả năng hoạt động tương thích với bộ vi xử lý. Vì vậy tên của hệ vi xử lý thường được lấy theo tên của dòng vi xử lý. Các thành phần khác trong hệ vi xử lý (Ram, Rom, HDD, MainBoard . . .) đều phải các thành phần có thể hoạt động ổn định có hiệu suất cao khi được phối ghép vào cùng 1 hệ thống với bộ vi xử lý.

1.4. Những đặc điểm cấu trúc của bộ vi xử lý

1.4.1. Công suất của bộ vi xử lý

Công suất của bộ vi xử lý là khả năng xử lý dữ liệu, nó gồm có những đặc điểm sau đây : độ dài từ của bộ vi xử lý (data word length), tính bằng số byte; dung lượng nhớ vật lý có thể đánh địa chỉ (addressing capacity); tốc độ xử lý lệnh của bộ vi xử lý (instruction exectue speed).

Công suất của hệ vi xử lý, hay nói cách khác, tốc độ xử lý thông tin, khả năng lưu trữ thông tin, khả năng kết nối nhiều loại thiết bị ngoại vi . . . phụ thuộc vào công suất của bộ vi xử lý.

- ✓ ***Độ dài từ***
  - VXL chỉ có thể xử lý dữ liệu với độ dài từ cố định, phụ thuộc và từng thế hệ vi xử lý và mức độ phát triển của công nghệ vi xử lý.
  - Tùy theo công nghệ, thế hệ VXL : 4 ,8,16,32,64 bit

- Quyết định độ dài của các thanh ghi, ALU, bus dữ liệu bên trong
- Độ dài từ càng lớn càng tạo ra nhiều khả năng tính toán của bộ vi xử lý : khoảng biểu diễn số rộng hơn, tốc độ tính toán nhanh hơn

✓ ***Khả năng đánh địa chỉ***

Các từ dữ liệu và lệnh máy cất trong bộ nhớ tại các ngăn nhớ khác nhau. Mỗi ngăn nhớ phải có địa chỉ nhận biết.

- Dải đánh địa chỉ càng lớn thì dung lượng nhớ càng nhiều.
- Để đánh địa chỉ, bộ vi xử lý có thanh ghi địa chỉ (address register).
- Độ rộng của thanh ghi địa chỉ quyết định giải địa chỉ của vùng nhớ vật lý mà bộ vi xử lý thỏa mãn.
- Khả năng đánh địa chỉ càng lớn thì càng cho phép tạo ra một hệ thống máy tính có cấu hình mạnh với nhiều loại thiết bị ngoại vi, bộ nhớ chính có dung lượng lớn và khả năng xử lý nhanh.

✓ ***Tốc độ thực hiện lệnh***

Tốc độ thực thực hiện lệnh của bộ vi xử lý có thể đo bằng tốc thực hiện các lện dấu phẩy động FLOPS (Floating Point Operations per Second) hoặc ính bằng triệu lệnh/giây (MIPS – Millions of instructions per second).

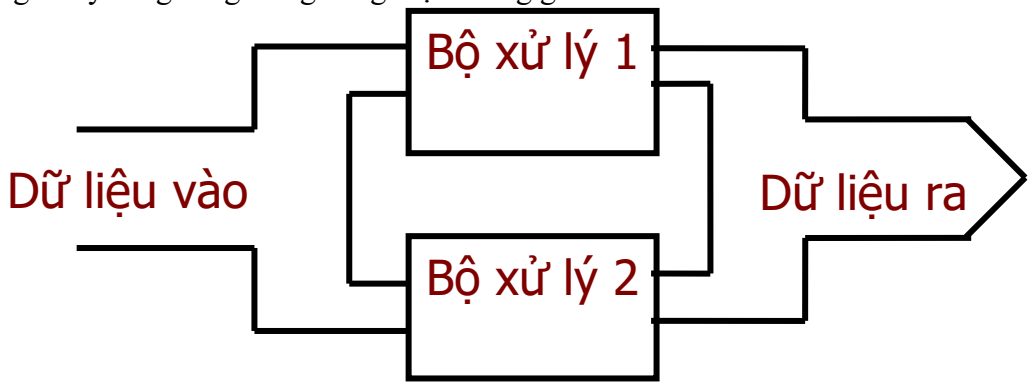
$$MIPS = \frac{f * N}{M + T}$$

- f : tần số làm việc của bộ VXL
- N : Số lượng các đơn vị xử lý số học và logic (ALU) không phụ thuộc vào nhau bên trong bộ vi xử lý.
- M : Số lượng vi lệnh (microinstruction) trung bình của 1 lệnh trong bộ vi xử lý (thường cần từ 4-7 vi lệnh).
- T : Hệ số thời gian truy nhập bộ nhớ (chu trình chờ đợi trong khi truy nhập bộ nhớ).

1.4.2. Những đặc tính nâng cao tốc độ của bộ vi xử lý

✓ ***Xử lý song song (parallel processing)***

Xử lý song song có nghĩa là hai hay nhiều quá trình tính toán cùng xảy ra đồng thời. Trong kiến trúc máy tính, sự kết hợp 2 bộ vi xử lý trong khối xử lý trung tâm (CPU) tạo ra khả năng xử lý song song trong cùng một không gian.

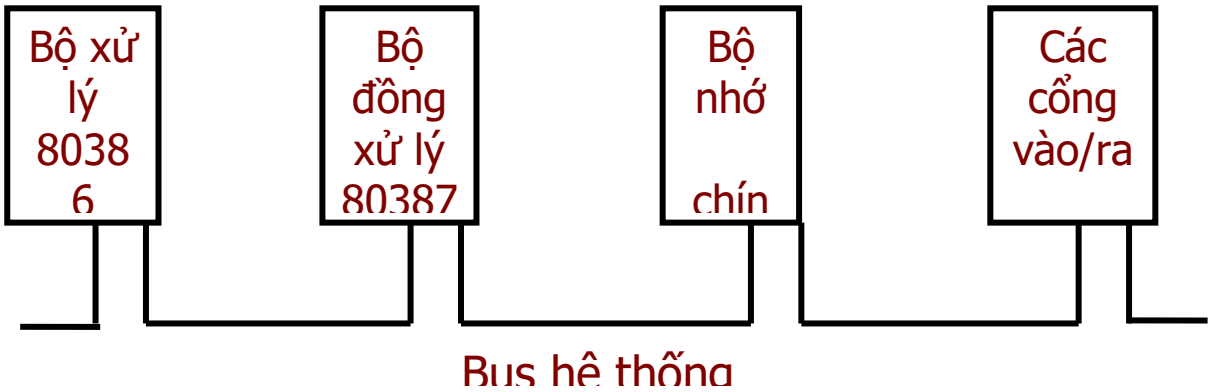


Kiểu kiến trúc này có thể tạo ra tốc độ xử lý dữ liệu lên gấp đôi so với kiến trúc chỉ dùng mộ bộ vi xử lý. Cũng có thể thực hiện xử lý song song ngay bên trong cấu trúc của bộ vi xử lý, bằng cách thiết kế saocho quá trình xử lý dữ liệu bên trong chip vi xử lý chia thành các phiên khác nhau và thực hiện song song nhờ sự phân chia khối logic điều khiển (CU) bên trong thành 2 phần riêng. Kiến trúc của bộ vi xử lý có các khôi chức năng xử lý song song bên

trong gọi là kiến trúc siêu hướng (superscalar architechture), nghĩa là cùng một lúc có nhiều hướng xử lý khác nhau bên trong bộ vi xử lý.

Kiến trúc siêu hướng không những nâng cao tốc độ xử lý mà còn nâng cao độ tin cậy của CPU, bởi vì khi có sự cố ở 1 chip vi xử lý (hay một đơn vị xử lý bên trong một chip) thì chip còn lại (hay đơn vị xử lý còn lại bên trong một chip) vẫn đảm nhiệm chức năng được bình thường.

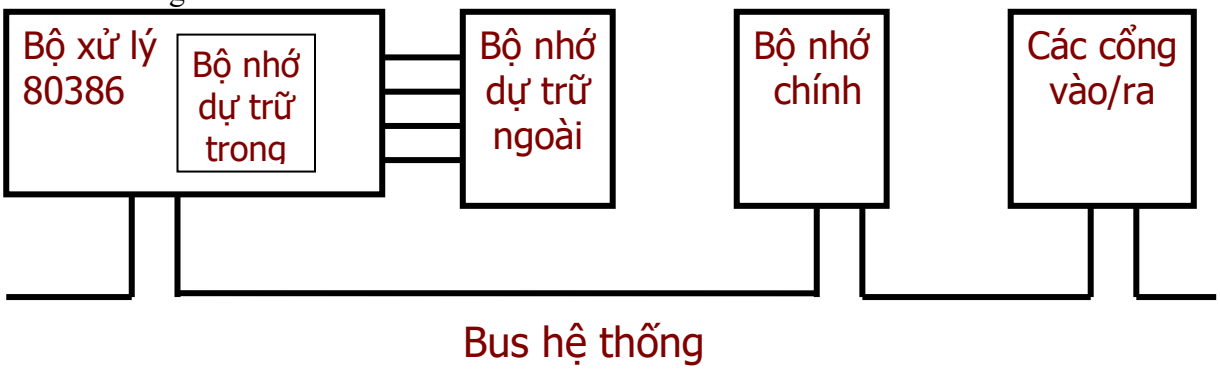
✓ ***Đồng xử lý (coprocessing)***



Bộ đồng xử lý là một bộ vi xử lý riêng biệt kết nối với bộ vi xử lý chính thông qua bus hệ thống. Ví dụ như trong họ vi xử lý Intel 80XXX, có bộ đồng vi xử lý 80387 làm việc với 80386 hoặc 80486. Bộ đồng xử lý chỉ thực hiện một số chức năng đặc biệt, ví dụ như các phép toán đòi hỏi sự chính xác sử dụng dấu phẩy động. Tốc độ của bộ đồng xử lý những phép tính này sẽ nhanh hơn rất nhiều so với bộ vi xử lý chính.

✓ ***Bộ nhớ lưu trữ (cache memory)***

Theo công thức MIPS, tốc độ của bộ vi xử lý sẽ tăng đáng kể nếu tổng M + T tiến tới 1. Thời gian truy nhập bộ nhớ T có thể giảm tối thiểu nhờ giảm tối thiểu số lượng lệnh đặt trong bộ vi xử lý và từng giai đoạn trong khi thực hiện lệnh và truy nhập bộ nhớ ngoài chồng lên nhau về thời gian.



Một biện pháp giảm T nữa là áp dụng kỹ thuật bộ nhớ dự trữ. Bộ nhớ cache là bộ nhớ có tốc độ cao, nó có thể nằm ngay bên trong bộ vi xử lý với dung lượng hạn chế (internal cache), hoặc nằm kề ngay bên cạnh bộ vi xử lý và kết nối trực tiếp với chip vi xử lý với dung lượng đủ lớn (external cache). Trong khi đó bộ nhớ chính (main memory) kết nối với bộ vi xử lý qua bus hệ thống.

Sự trao đổi dữ liệu giữa bộ nhớ chính và bộ vi xử lý bị hạn chế về tốc độ, vì vậy để tăng tốc độ xử lý, phải tổ chức làm sao khi thực hiện chương trình, bộ vi xử lý trước hết tìm kiếm lệnh ở bộ nhớ dự trữ trước, nếu không có lệnh chứa trong bộ nhớ dự trữ thì mới phải tìm đến bộ nhớ chính. Điều này có nghĩa là nếu đa số lệnh khong có trong bộ nhớ dự trữ thì tốc độ xử

lý chậm hơn gần gấp đôi so với vơi truy nhập thẳng vào bộ nhớ chính. Vì vậy phải tổ chức làm sao đa số các lệnh của chương trình nằm trong bộ nhớ dự trữ.

- ✓ ***Kỹ thuật đường ống (pipelining technique)***

Mô phỏng dây chuyền lắp ráp máy móc, hệ thống đường ống dẫn, trong các bộ vi xử lý hiện nay có chức năng thực hiện các lệnh máy liên tục thành một dây chuyền với 5 công đoạn : nhạp dữ liệu của lệnh từ bộ nhớ, giải mã lệnh, thực hiện các lệnh, ghi kết quả thực hiện lệnh vào bộ nhớ. Khi lệnh thứ nhất bắt đầu thực hiện ở giai đoạn hai thì mã lệnh của lệnh kế tiếp theo được đọc từ bộ nhớ ra để thực hiện bước một (giải mã lệnh). Cứ như vậy các lệnh được thực hiện theo một dây chuyền liên tục như là dòng nước đi trong đường ống. Tốc độ xử lý lệnh vì thế được tăng lên rất cao.

1.4.3. Tập lệnh của bộ vi xử lý

Tập lệnh là một đặc tín then chốt của kiến trúc bộ vi xử lý. Các bộ vi xử lý hoạt động theo sự chỉ dẫn của tập lệnh riêng của chúng. Người lập trình hay nhà thiết kế hệ thống dựa vào tập lệnh để soạn thảo các chương trình điều khiển hoạt động của bộ vi xử lý theo một trình tự nào đó.

Một bộ vi xử lý của một tập lệnh riêng và thường có tính kế thừa, tức là tập lệnh của những bộ vi xử lý ra đời sau thường bao hàm các lệnh của các bộ vi xử lý cùng họ đã được sản xuất trước đó.

Tập lệnh thường được chia thành 8 hoặc 9 nhóm khác nhau, và được biểu diễn ở dạng mã ngữ (Memonic code) gợi nhớ của tiếng Anh đời thường :

- Các lệnh chuyển dữ liệu.
- Các lệnh số học với các nguyên và các lệnh logic.
- Các lệnh dịch và quay vòng.
- Các lệnh chuyển điều khiển.
- Các lệnh xử lý bit.
- Các lệnh điều khiển hệ thống.
- Các lệnh với dấu phảy động.
- Các lệnh của các khối chức năng đặc biệt.

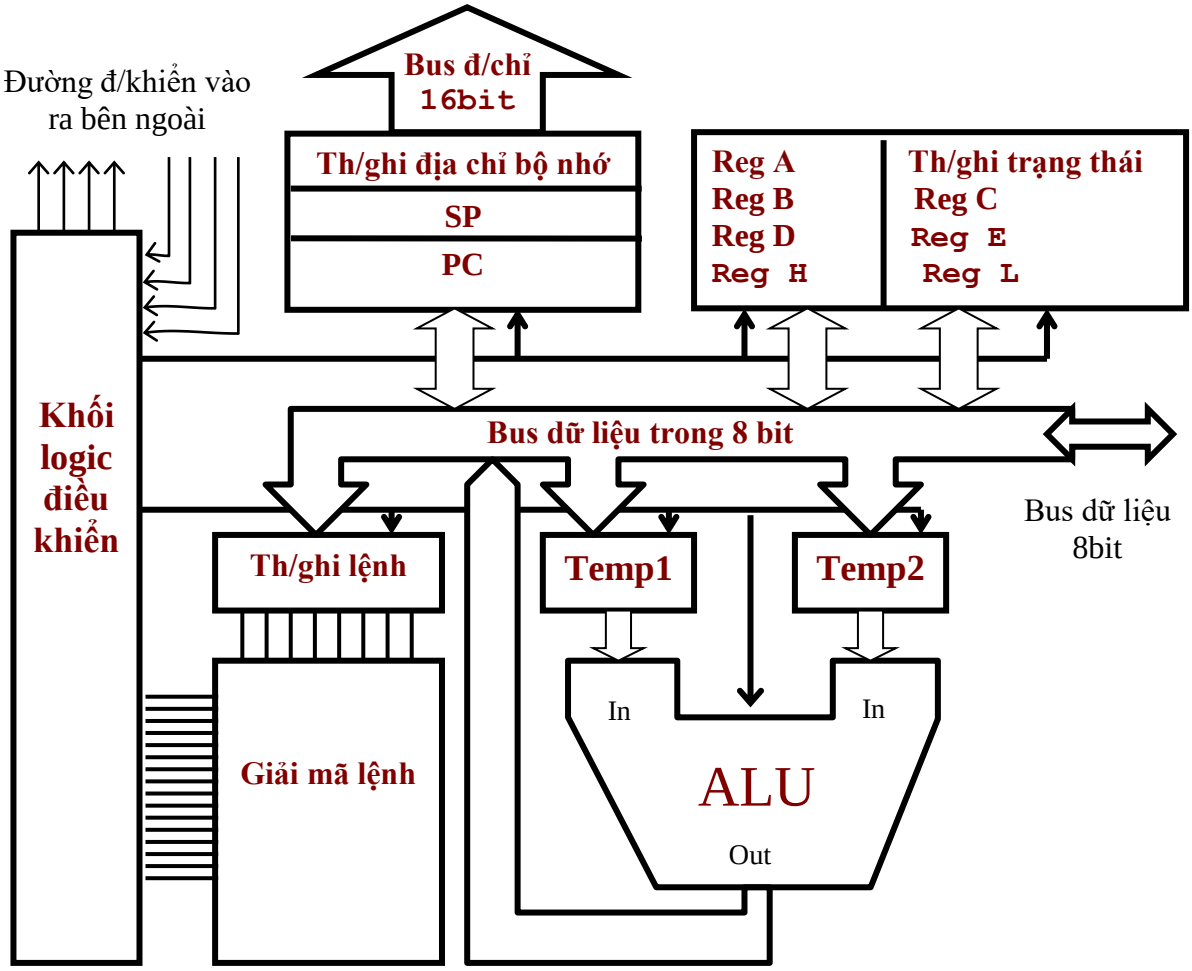
Sự phân chia này phù hợp với các bộ vi xử lý công nghệ cao hiện nay, vì cấu trúc bên trong của chúng tập trung các khối chức năng đặc biệt

Chương 2 : CẤU TRÚC CÁC BỘ VI XỬ LÝ

2.1. Sơ đồ khối cấu trúc bộ VXL cấp thấp (8bit)

Sơ đồ khối cấu trúc và mô hình lập trình của bộ vi xử lý là tổ chức các thanh ghi mà người sử dụng có thể lập trình và truy nhập tới được bên trong bộ vi xử lý. Cấu trúc cơ bản của bộ vi xử lý cấp thấp có 3 thành phần chính :

- Đơn vị logic – số học ALU (Arithmetic logic unit).
- Khối các thanh ghi - Registers.
- Khối logic điều khiển - CU.



2.1.1. Đơn vị số học – logic ALU

Đơn vị ALU là một trong những thành phần chính bên trong bộ vi xử lý. ALU chứa khối loggic thực hiện xử lý dữ liệu. ALU có 2 cổng vào và một cổng ra. Cổng vào để nhận dữ liệu vào ALU, cổng ra để lấy kết quả xử lý dữ liệu của ALU ra bên ngoài. Các thanh ghi Temp1 và Temp2 làm nhiệm vụ nhận dữ liệu từ các nơi khác nhau bên trong bộ vi xử lý và lưu trữ trung gian dữ liệu trong quá trình xử lý dữ liệu trong ALU. Tương tự cổng ra kết nối với bus dữ liệu bên trong, do đó kết quả phép toán có thể đưa tới các nơi khác nhau.

Các lệnh máy được ALU xử lý có thể là một hay hai toán hạng. Lệnh máy được đọc từ bộ nhớ vào bộ vi xử lý, được giải mã nhờ bộ giải mã lệnh để tạo ra chuỗi các tín hiệu đi tới ALU điều khiển quá trình xử lý dữ liệu trong ALU.

2.1.2. Các thanh ghi (Registers)

Các thanh ghi bên trong bộ vi xử lý được chia thành cá nhóm theo mục đích sử dụng :

- Nhóm thanh ghi dùng chung : thanh ghi A (bộ cộng), B, C, D, E, H và L. Đây là các thanh ghi 8 bit. Để xử lý 16 bit, có các lệnh thực hiện với các cặp thanh ghi BC, DE và HL
- Các thanh ghi khác : thanh ghi trạng thái (SR), con trỏ ngăn xếp (SP), thanh ghi đếm lệnh (PC), thanh ghi địa chỉ ngăn nhớ, thanh ghi tạm thời Temp1, Temp2, thanh ghi lệnh

✓ Thanh ghi tổng A (Accumulator)

Thanh ghi này tham gia vào phần lớn các phép tính, độ dài bằng độ dài từ của bộ vi xử lý. Ở một số loại vi xử lý có thể có thanh tổng có độ dài gấp đôi độ dài từ xử lý của bộ vi xử lý hay còn gọi là thanh tổng độ dài kép.

✓ Thanh ghi đếm chương trình PC (Program counter)

Chứa địa chỉ của lệnh trong bộ nhớ và chỉ ra cho bộ VXL biết lệnh tiếp theo nằm ở ngăn nhớ nào để lấy ra thực hiện. Như vậy độ dài của PC chính là khả năng đánh địa chỉ bộ nhớ chính có thể đạt được của bộ vi xử lý. PC luôn được nạp địa chỉ lệnh (địa chỉ ngăn nhớ chứa lệnh máy) tiếp theo trong quá trình thực hiện bất kỳ một chương trình nào. Trong các bộ vi xử lý công nghệ cao có cơ chế quản lý bộ nhớ ảo (Virtual Memory) bao gồm bộ nhớ chính và bộ nhớ ngoài, cơ chế dành địa chỉ ảo có sự biến đổi địa chỉ ảo thành địa chỉ vật lý.

✓ Thanh ghi trạng thái SR (Status register)

Lưu kết quả của các lệnh kiểm tra, so sánh và một số lệnh tính toán với các thanh ghi. Đôi khi thanh ghi trạng thái còn được gọi là thanh ghi cờ (Flag Register). Sử dụng các bit của thanh ghi trạng thái (bit cờ) có thể thực hiện rẽ nhánh chương trình bằng các lệnh nhảy và rẽ nhánh có điều kiện. Mỗi bit của thanh ghi trạng thái có một ý nghĩa và bị tác động tùy theo lệnh.

✓ Con trỏ ngăn xếp SP (Stack pointer)

Ngăn xếp có thể là một vùng nào đó của bộ nhớ chính, hay là một mảng thanh ghi riêng biệt. Thanh ghi SP chứa địa chỉ của đỉnh ngăn xếp và nội dung của SP sẽ thay đổi mỗi khi thực hiện các lệnh – luôn trỏ tới vùng nhớ tiếp theo. Khi khởi động hệ thống máy tính, con trỏ ngăn xếp luôn được khởi tạo về địa chỉ đỉnh của ngăn xếp.

✓ Thanh ghi địa chỉ bộ nhớ và logic

Các đầu ra của thanh ghi này được điều khiển nối ra bus địa chỉ của hệ thống máy tính để thực hiện việc chọn ngăn nhớ, hoặc để chọn cổng ngoại vi nào đó. Trong chu kỳ đọc lệnh, một lệnh máy được đọc từ bộ nhớ, nội dung của thanh ghi địa chỉ ngăn nhớ và nội dung của thanh đếm lệnh PC lúc này là như nhau. Nghĩa là thanh ghi địa chỉ bộ nhớ trỏ tới từ lệnh đang được đọc từ bộ nhớ. Thanh ghi địa chỉ bộ nhớ không thể tự động tưng hay giảm nội dung mà nó nhận địa chỉ lệnh từ PC, SP và các thanh ghi chỉ số.

✓ Thanh ghi lệnh IR (Instruction register)

IR chứa lệnh đang thực hiện. Trong chu kỳ đọc mã lệnh (instruction cycle) từ bộ nhớ, mã lệnh được nạp vào IR thông qua bus dữ liệu nội bộ. IR như là bộ đệm duy trì nội dung mẽ lệnh và đầu ra của IR đưa tới bộ giải mã lệnh để tạo ra chuỗi các tín hiệu điều khiển thực hiện lệnh. Độ dài của IR tùy thuộc vào từng bộ vi xử lý.

✓ Các thanh ghi dữ liệu tạm thời Temp (temporaty data registers)

Do ALU không có bộ đệm, do đó khi tính toán các dữ liệu phải được lấy từ hoặc đưa trở lại bus dữ liệu bên trong theo thời điểm nhịp nào đó. Những việc này phải được thực hiện thông qua các thanh ghi tạm thời. Ngoài ra, trong quá trình xử lý cần sự ổn định dữ liệu ở đầu vào ALU, do đó cần thanh ghi đệm tạm thời để lưu trữ dữ liệu trong thời gian ngắn đủ để cho ALU thực hiện xong phép tính.

2.1.3. Khối logic điều khiển CL (Control Logic)

Khối CL liên hệ thông tin với tất cả các đơn vị trong bộ vi xử lý, bởi vì nó điều khiển toàn bộ hoạt động xử lý thông tin bên trong bộ vi xử lý. Kết quả giải mã lệnh được đưa đến khối logic điều khiển tạo ra các chuỗi tín hiệu để điều khiển quá trình ghi, đọc với các thanh ghi bên trong, tính toán trong ALU.

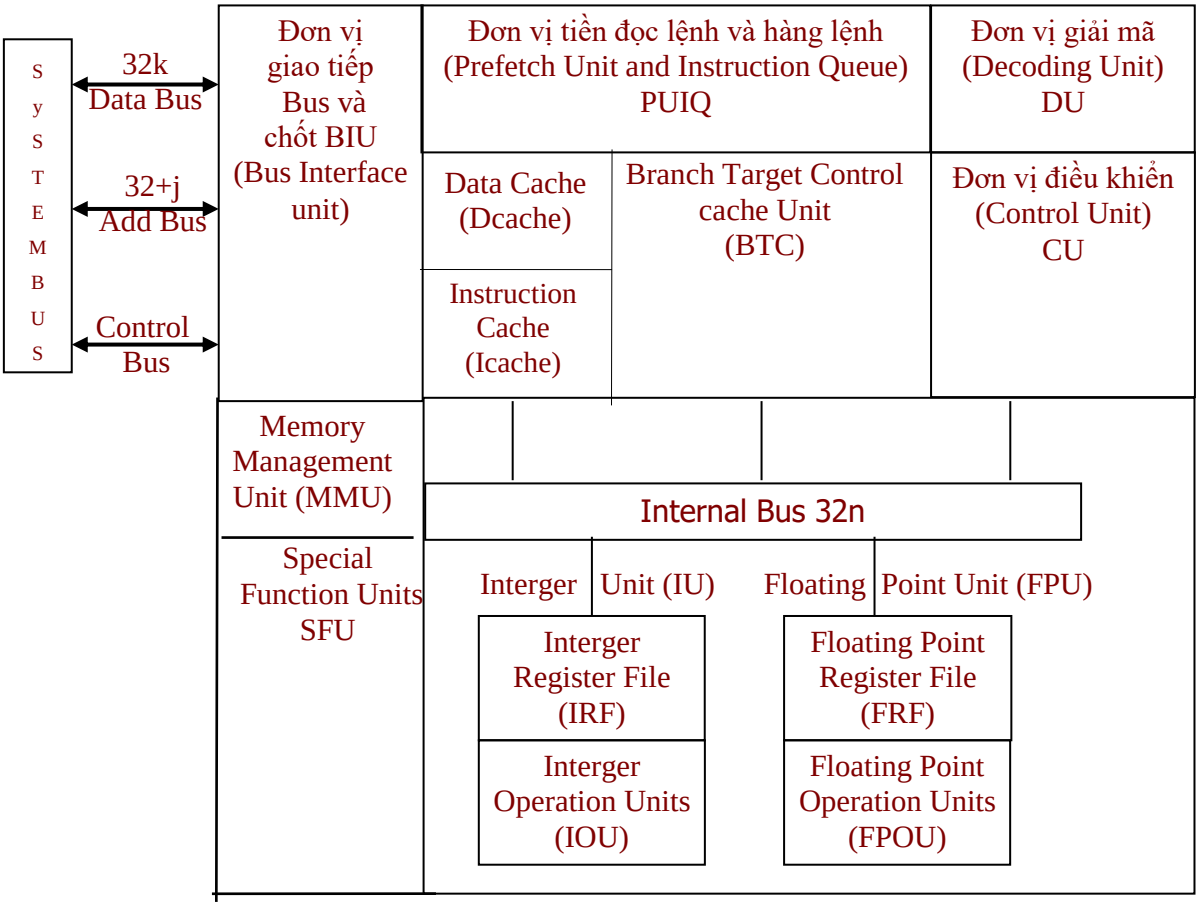
Từ CL, các xung tín hiệu điều khiển đi ra bus điều khiển của hệ thống tác động đến bộ nhớ, các đơn vị I/O để thực hiện trao đổi dữ liệu. Cũng có thể CL nhận các tín hiệu điều khiển từ bên ngoài thông qua bus điều khiển của hệ thống để tiếp tục điều khiển quá trình xử lý bên trong bộ vi xử lý.

Thông thường CL là một khối được vi chương trình hóa (Microprogrammed logic unit), tức là kiến trúc của CL giống như kiến trúc của vi xử lý nhỏ. Một tín hiệu quan trọng quyết định trình tự sản sinh các chuỗi tín hiệu của CL là nhịp đồng hồ của bộ vi xử lý. Nhịp đồng hồ tác động đến CL, và CL từ đây tạo ra các xung tín hiệu điều khiển có các chu kỳ khác nhau.

CL quyết định thứ tự làm việc của từng đơn vị trong bộ vi xử lý và sự trao đổi thông tin với thế giới bên ngoài chip vi xử lý. CL là trung tâm điều khiển của bộ vi xử lý.

2.2. Sơ đồ khối cấu trúc bộ VXL công nghệ cao

Các bộ vi xử lý công nghệ cao gọi chung cho các bộ vi xử lý được sản xuất từ giữa những năm 1990 của nhiều hãng chế tạo hàng đầu thế giới như Intel, Motorola, Sun, IBM, AMD . . . Nó chứa đựng các đặc điểm của đa số các loại vi xử lý hiện nay.



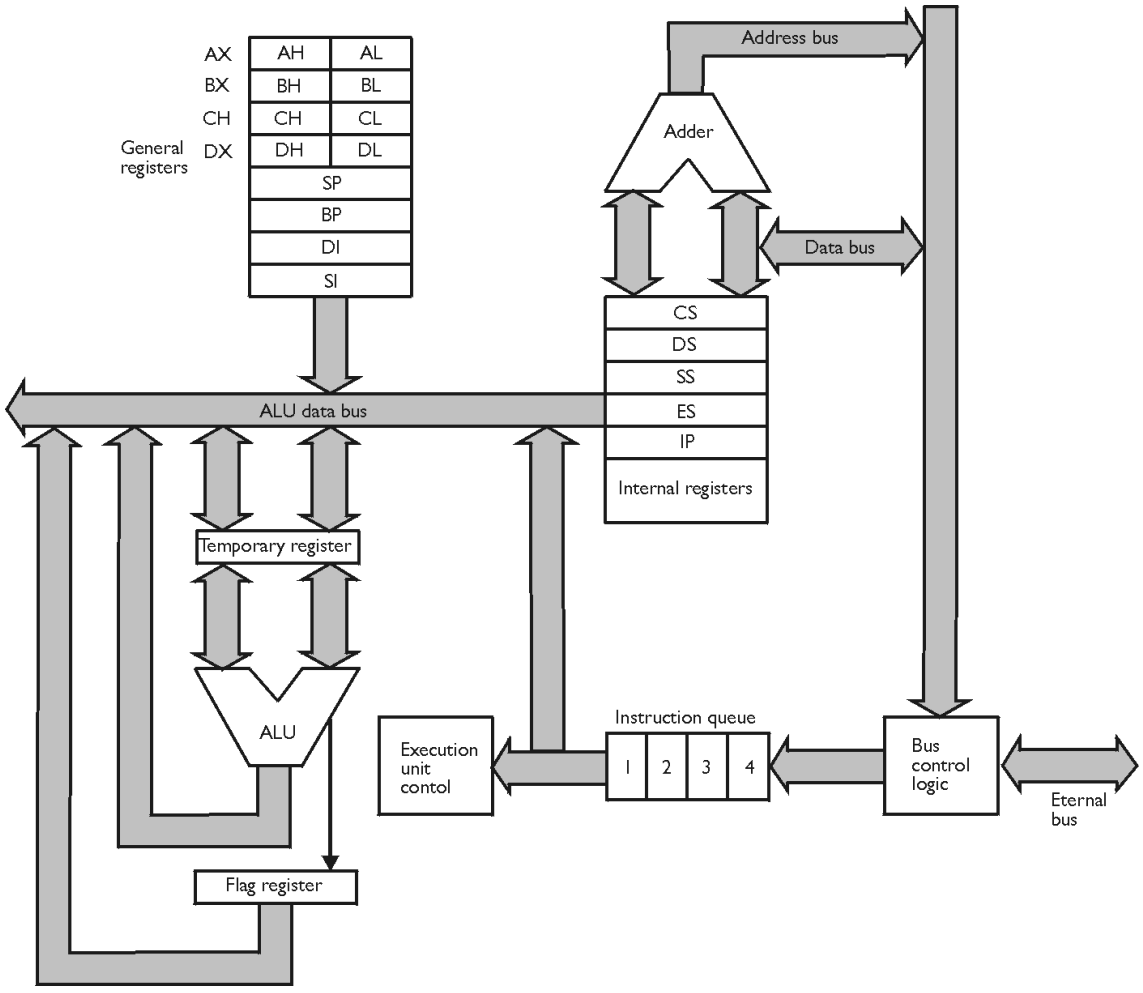
Chương 3: BỘ VI XỬ LÝ INTEL 8088

Sau khi đã tìm hiểu qua về cấu trúc của hệ vi xử lý. Trong chương này ta sẽ đi sâu tìm hiểu mọt bộvi xử lý cụ thể và rất điển hình: bộ vi xử lý của Intel.

Trước hết cần nói rỏ lý do tại sao ở đây ta lại chọn đích danh bộ vi xử lý 8088 để tìm hiểu mà không phải là bộ vi xử lý nào khác (điều mà nhiều người khác phải làm). Thứ nhất, đây là bộ vi xử lý nổi tiếng một thời thuộc họ 80x86 của Intel, nó được sử dụng trong nhiều lĩnh vực khác nhau, nhất là trong các máy IBM PC /XT. Các bộ vi xử lý thuộc họ này sẽ còn được sử dụng rộng rãi trong hàng chục năm nữa, và vi tính kế thừa của các sản phẩm trong họ 80x86, các chương trình viết cho 8088 vẫn có thể chạy trên các hệ thống tiên tiến sau này. Thứ hai, về góc độ sư phạm thì đây là bộ vi xử lý khá đơn giản và vì việc dạy hiểu nó là tương đối dể đối với những người mới bắt đầu thâm nhập vào lĩnh vực này. Thứ ba, các họ vi xử lý tuy có khác nhau nhưng xét cho cùng cũng có khá nhiều điểm chủ yếu rất giống nhau. Do đó một khi đã nắm được các vấn đề kỷ thuật của8088, ta sẽ có cơ sở để nắm bắt các kỹ thuật của các bộ vi xử lý khác cùng trong họ Intel 80x86 hoặc của các họ khác.

3.1. Giới thiệu cấu trúc bên trong vàhoạt động của bộ vi xử lý 8088.

Trước khi giới thiệu tập lệnh và cách thức lập trình cho bộvi xử lý8088 hoạt động ta cần phải tìm hiểu kỹ cấu trúc bên trong của nó.



3.1.1. BIU Và EU

Theo sơ đồ khối, ta thấy bên trong CPU 8088 có 2 khối chính: *khối phối ghép (* bus interface unit, BIU) *vàkhối thực hiện lệnh* (execution unit, EU). Việc chia CPU ra thành 2 phần làm việc đồng thời có liên hệ với nhau qua đệm lệnh làm tăng đáng kể tốc độ xử lý của CPU. Các bus bên trong CPU có nhiệm vụ chuyển tải tín hiệu của các khối khác. Trong số các

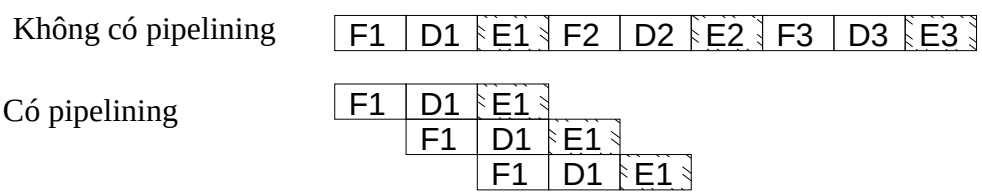
bus đó có bus dữ liệu 16 bit của ALU, bus các tín hiệu điều khiển ở EU và bus trong của hệ thống ở BIU. Trước khi đi ra bus ngoài hoặc đivào bus trong của bộ vi xử lý, các tín hiệu truyền trên bus thường được cho đi qua các bộ đệm để nâng cao tính tương thích cho nối ghép hoặc nâng cao phối ghép.

BIU đưa ra địa chỉ, đọc mã lệnh từ bộ nhớ, đọc / ghi dữ liệu từ vào cổng hoặc bộ nhớ. Nói cách khác BIU chịu trách nhiệm đưa địa chỉ ra bus và trao đổi dữ liệu với bus.

Trong EU ta thấy có một *khối điều khiển* (control unit, CU). Chính tại bên trong khối điều khiển này có *mạch giải mã lệnh.* Mã lệnh đọc vào từ bộ nhớ được đưa đến đầu vào của bộ giải mã, các thông tin thu được từ đầu ra của nó sẽ được đưa đến mạch tạo xung điều khiển, kết quả là tu thu được các dãy xung khác nhau (tuỳ theo mã lệnh) để điều khiển hoạt động của các bộ phận bên trong và bên ngoài CPU. Trong khối EU còn có *khối số học và lôgic* (arithmetic anh logic unit. ALU) dùng để thực hiện các thao tác khác nhau với các toán hạng của lệnh. Tóm lại, khi CPU hoạt động EU sẽ cung cấp thông tin về địa chỉ cho BIU để khối này đọc lệnh và dữ liệu, còn bản thân nó thì đọc lệnh và giải mã lệnh.

Trong BIU còn có một *bộ nhớ đệm lệnh* với dung lượng 4 byte dùng để chứa các mã lệnh đọc được nằm sẵn để chờ EU xử lý (trong tài liệu của Intel bộ đệm lệnh này còn được gọi là *hàng đợi lệnh).* Đây là một cấu trúc mới được cấy vào bộ vi xử lý 8086x88 do việc Intel đưa cơ chế *xử lý xen kẻ liên tục, dòng mã lệnh* (instruction pipelining) vào ứng dụng trong các bộ vi xử lý thế hệ mới. Pipeline là một cơ chế đã được ứng dụng từ những năm 60 từ các máy lớn. Nhân đây ta sẽ giới thiệu sơ qua một chút về cơ chế này.

Trong các bộ vi xử lý ở các thế hệ trước (như ở 8085 chẳng hạn), thông thường hoạt động của CPU gồm 3 giai đoạn: đọc mã lệnh (opcode fetch), giải mã lệnh (decode) và thực hiện lệnh (execution). Trong một thời điểm nhất định, CPU thế hệ này chỉ có thể thực hiện một trong ba công việc nói trên và vì vậy tuỳ theo từng giai đoạn sẽ có những bộ phận nhất định của CPU ở trạng thái nhàn rỗi. Chẳng hạn, khi CPU giải mã lệnh hoặc khi nó đang thực hiện những lệnh không liên quan đến bus (thao tác nội bộ) thì các bus không được dùng vào việc gì dẫn đến tình trạng lãng phí khả năng của chúng . Trong khi đó từ bộ vi xử lý 8086/88, Intel sử dụng cơ chế xử lý xen kẽ liên tục dòng mã lệnh thì CPU được chia thành 2 khối và có sự phân chia công việc cho từng khối: việc đọc mã lệnh là do khối BIU thực hiện, việc giải mã lệnh và thực hiện lệnh là do khối EU đảm nhiệm. Các khối chức năng này có khả năng làm việc đồng thời và các bus sẽ liên tục sử dụng: trong khi EU lấy mã lệnh từ bộ đệm 4 byte để giải mã hoặc thực hiện các thao tác nội bộ thì BIU vẫn có thể đọc mã lệnh từ bộ nhớ chính rồi đặt chúng vào bộ nhớ đệm lệnh đã nói. Bộ đệm lệnh này làm việc theo kiểu " *vào trước – ra trước "* (first in-first out, FIFO), nghĩa là byte nào được cất vào đệm trước sẽ được lấy ra xử lý trước. Nếu có sự vào/ra liên tục của dòng mã lệnh trong bộ đệm này thì có nghĩa là có sự phối hợp hoạt động hiệu quả giữa hai khối EU và BIU theo cơ chế xử lý xen kẽ liên tục dòng mã lệnh để làm tăng tốc độ xử lý tổng thể. Kỹ thuật xử lý xen kẽ liên tục dòng mã lệnh sẽ không còn tác dụng tăng tốc độ xử lý chung của CPU nữa nếu như trong đệm lệnh có chứa các mã lệnh của các lệnh CALL (gọi chương trình con) hoặc JMP (nhảy), bởi vì lúc các lệnh này nội dung của bộ đệm sẽ bị xoá và thay thế vào đó là nội dung mới được nạp bởi các mã 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



(F : Đọc lệnh , D :Giải mả lệnh, E : Thực hiện lệnh)

Dòng lệnh thường và dòng lệnh xen kẽ liên tục

Trong bộ vi xử lý 8088 ta còn thấy c các thanh ghi 16 bit nằm trong cả hai khối BIU và EU, ngoài ra cũng có một số thanh ghi 8 hoặc 16 bit tại EU. Ta sẽ lần lượt giới thiệu các thanh ghi nói trên cùng chức năng chính của chúng.

✓ **Các thanh ghi đoạn**

Khối BIU đưa ra trên bus địa chỉ 20 bit địa chỉ, như vậy 8088 có khả năng phân biệt ra được 2^{20} = 1.048.576 = 1M ô nhớ hay 1Mbyte, vì các bộ nhớ nói chung tổ chức theo byte. Nói cách khác: không gian địa chỉ của 8088 là 1Mbyte. Trong không gian 1Mbyte bộ nhớ cần được chia thành các vùng khác nhau (điều này rất có lợi khi làm việc ở chế độ nhiều người sử dụng hoặc đa nhiệm) dành riêng để:

- Chứa mã chương trình.
- Chứa dữ liệu và kết quả không 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 bộ vi xử lý 8088 có các thanh ghi 16 bit liên quan đến địa chỉ đầu của các vùng (các đoạn) kể trên và chúng được gọi là các thanh ghi đoạn (Segment Registers). Đó là thanh ghi đoạn mã CS (Code-Segment), thanh ghi đoạn dữ liệu DS (Data segment). Thanh ghi đoạn ngăn xếp SS (Stack segment) và thanh ghi đoạn dữ liệu phụ ES (Extra segment). Các thanh ghi đoạn 16 bit này chỉ ra địa chỉ đầu của bốn đoạn trong bộ nhớ, dung lượng lớn nhất của mỗi đoạn nhớ này là 64 Kbyte và tại một thời điểm nhất định bộ vi xử lý chỉ làm việc được với bốn đoạn nhớ 64 Kbyte này. Việc thay đổi giá trị của các thanh ghi đoạn làm cho các đoạn 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 trong chúng đòi hỏi dung lượng đủ 64 Kbyte hoặc cũng có thể nằm trùm nhau do có những đoạn không cần dùng hết đoạn dài 64 Kbyte và vì vậy những đoạn khác có thể bắt đầu nối tiếp ngay sau đó. Điều này cũng cho phép ta truy nhập vào bất kỳ đoạn nhớ (64 Kbyte) nào nằm trong toàn bộ không gian 1 Kbyte.

Nội dung các thanh ghi đoạn sẽ xác định địa chỉ của ô nhớ nằm ở đầu đoạn. Địa chỉ này còn gọi là địa chỉ cơ sở. Địa chỉ của các ô nhớ khác nằm trong đoạn tính được bằng cách cộng thêm vào địa chỉ cơ sở một giá trị gọi là địa chỉ lệch hay độ lệch (Offset), gọi như thế vì nó ứng với khoảng lệch của toạ độ một ô nhớ cụ thể nào đó so với ô đầu đoạn. Độ lệch này được xác định bởi các thanh ghi 16 bit khác đóng vai trò thanh ghi lệch (Offset register) mà ta sẽ nói đến sau. Cụ thể, để xác định địa chỉ vật lý 20 bit của một ô nhớ nào đó trong một đoạn bất kỳ. CPU 8088 phải dùng đến 2 thanh ghi 16 bit (một thanh ghi để chứa địa chỉ cơ sở, còn thanh kia chứa độ lệch) và từ nội dung của cặp thanh ghi đó tạo ra địa chỉ vật lý theo công thức sau:

Địachỉvậtlý=Thanhghiđoanx16+Thanhghilệch

Việc dùng 2 thanh ghi để ghi nhớ thông tin về địa chỉ thực chất để tạo ra một loại địa chỉ gọi là địa chỉ logic và được ký hiệu như sau:

Thanhghiđoạn: Thanhghilệch hay **segment: offset**

Địa chỉ kiểu **segment: offset** là logic vì nó tồn tại dưới dạng giá trị của các thanh ghi cụ thể bên trong CPU và ghi cần thiết truy cập ô nhớ nào đó thì nó phải được đổi ra địa chỉ vật lý để rồi được đưa lên bus địa chỉ. Việc chuyển đổi này do một bộ tạo địa chỉ thực hiện (phần tử $\sum$).

Ví dụ: cặp CS:IP sẽ chỉ ra địa chỉ của lệnh sắp thực hiện trong đoạn mã. Tại một thời điểm nào đó ta có CS = F00H và IP = FFFOH thì

CS:IP~FOOOHx16 + FFFOH = FOOOOH + FFFOH = FFFFOH

Địa chỉ FFFFOH chính là địa chỉ khởi động của 8088 dấu ~ ở đây là để chỉ sự tương ứng. Địa chỉ các ô nhớ thuộc các đoạn khác cũng có thể tính được theo cách tương tự như

vậy. Từ nay khi cần nói đến 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 tồn tại sự tương ứng giữa hai loại địa chỉ này (thông qua bộ tạo địa chỉ $\sum$).

Trước khi nói đến các thanh ghi khác ta nói thêm chút ít về tính đa trị của các thanh ghi đoạn và thanh ghi lệch trong địa chỉ logic ứng với một địa chỉ vật lý. Điều này cũng nói lên tính linh hoạt của cơ chế **segment offset** trong việc định địa chỉ của 8086/ 88. Nhìn vào giá trị cuối cùng của địa chỉ vật lý ta thấy có thể tạo ra địa chỉ đó từ nhiều giá trị khác nhau của thanh ghi đoạn và thanh ghi lệch

Ví dụ: Địa chỉ vật lý 12345H có thể được tạo ra từ các giá trị:

Thanh ghi đoạn	Thanh ghi lệch
1000H	2345H
1200H	0345H
1004H	2305H
0300H	E345H
…	…

✓ **Các thanh ghi đa năng**

Trong khối EU có bốn thanh ghi đa năng 16 bit AX, BX, CX, DX. Điều đặc biệt là khi cần chứa các dữ liệu 8 bit thì mỗi thanh ghi có thể tách ra thành hai thanh ghi 8 bit cao và thấp để làm việc độc lập, đó là các tập thanh ghi AH và AL, BH và BL, CH và CL, DH và DL (trong đó H chỉ phần cao, L chỉ phần thấp). Mỗi thanh ghi có thể dùng một cách vạn năng để chứa các tập dữ liệu khác nhau nhưng cũng có 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 gan cho những cái tên đặc biệt rất có ý nghĩa.

Cụ thể:

- AX (accumulator, acc): thanh chứa. Các kết qủa của các thao tác thường được chứa ở đây (kết quả của phép nhân, chia). Nếu kết quả là 8 bit thì thanh ghi AL được coi là acc.
- BX (base): thanh ghi cơ sở thường chứa địa chỉ cơ sở của một bảng dùng trong lệnh XLAT.
- CX (count): bộ đếm. CX thường được dùng để chứa số lần lặp trong trường hợp các lệnh LOOP (lặp), còn CL thường cho ta số lần dịch hoặc quay trong các lệnh dịch hoặc quay thanh ghi.
- DX (data): thanh ghi dữ liệu DX cùng BX tham gia các thao tác của phép nhân hoặc chia các số 16 bit. DX thường dùng để chứa địa chỉ của các cổng trong các lệnh vào/ ra dữ liệu trực tiếp.

✓ **Các thanh ghi con trỏ và chỉ số**

Trong 8088 còn có ba thanh ghi con trỏ và hai thanh ghi chỉ số 16 bit. Các thanh ghi này (trừ IP) đều có thể được dùng như các thanh ghi đa năng, nhưng ứng dụng chính của mỗi thanh ghi là chúng được ngầm định như là thanh ghi lệch cho các đoạn tương ứng. Cụ thể:

- IP: con trỏ lệnh (Instruction pointer). 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: con trỏ cơ sở (base pointer). 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.

o SP: con trỏ ngăn xếp (stack pointer). SP luôn trỏ vào đỉnh hiện thời của ngăn xếp nằm trong đoạn ngăn xếp SS. Địa chỉ đỉnh ngăn xếp ứng với SS:SP và được xác định theo cách đã nói ở trên.

o SI: chỉ số gốc hay nguồn (source index). SI chỉ vào dữ liệu trong đoạn dữ liệu DS mà địa chỉ cụ thể đầy đủ ứng với DS:SI và được xác định theo cách đã nói ở trên.

o DI: chỉ số đích (destination index). DI chỉ vào dữ liệu trong đoạn dữ liệu DS mà địa chỉ cụ thể đầy đủ ứng với DS:DI và được xác định theo cách đã nói ở trên.

Riêng trong các lệnh thoa tác với dữ liệu kiểu chuỗi thì cặp ES:DI luôn ứng với địa chỉ của phần tử thuộc chuỗi đích còn cặp DS:SI ứng với địa chỉ của phần tử thuộc chuỗi gốc.

✓ **Thanh ghi cờ FR (flag register)**

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

Các cờ của bộ vi xử lý 8086

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

X : Không được định nghĩa

Sơ đồ thanh ghi cờ của bộ vi xử lý 8086/88

Các cờ cụ thể

o C hoặc CF (carry flag): cờ nhớ. CF = 1 khi có nhớ hoặc muợn từ MSSP.

o P hoặc PF (parity flag): cờ parity. PF phản ánh tính chẵn lẻ (parity) của tổng số bit 1 có trong kết quả. Cờ PF =1 khi tổng số bit trong kết quả là chẵn (even parity, parity chẵn). Ở đây ta tạm dùng parity dạng nguyên gốc để tránh sự lủng củng khi phải dịch cụm từ " even parity " thành tính chẵn lẻ chẵn hoặc " odd party " thành tính chẵn lẻ lẻ.

o A hoặc AF (auxilialyry carry flag): cờ nhớ phụ rất có ý nghĩa khi ta làm việc với các số BCD.AF = 1 khi có nhớ hoặc muợn từ một số BCD thấp (4 bit thấp) sang một số BCD cao (4 bit cao).

o Z hoặc ZF (zero flag): cờ rỗng. ZF =1 khi kết quả = 0.

o S hoặc SF (sign flag): cờ dấu. SF = 1 khi kết quả âm.

o O hoặc OF (over flow flag): cò tràn. OF = 1 khi kết quả là một số bù 2 vượt qua ngoài giới hạn biểu diễn dành cho nó.

Trên đây là 6 bit cờ trạng thái phản ánh các trạng thái khác nhau của kết sau một thao tác nào đó, trong đó 5 bit cờ đầu thuộc byte thấp của thanh cờ là các cờ giống như của bộ vi xử lý 8 bit 8085 của Intel. Chúng được lặp hoặc xoá tuỳ theo các điều kiện cụ thể sau các thao tác của ALU. Ngoài ra, bộ vi xử lý 8088 còn có các cờ điều khiển sau đây (các cờ này được lập hoặc xoá bằng các lệnh riêng):

o 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 trong một chương trình).

o I hoặc IF (interrupt enable flag): cờ cho phép ngắt. IF = 1 thì CPU cho phép các yêu cầu ngắt (che được) được tác động.

o D hoặc DF (direction flag): cờ hướng. DF = 1 khi CPU làm việc với chuỗi ký tự theo thứ tự từ phải sang trái (vì vậy D chính là cờ lùi)

Ý nghĩa của các cờ đã khá rõ ràng. Riêng cờ tràn cần phải làm rõ hơn để ta hiểu được bản chất và cơ chế làm việc của nó. Cờ tràn thường được dùng đến khi ta làm việc với số bù 2 có dấu.

3.2. Cách mã hoá lệnh của bộ vi xử lý 8088

Lệnh của bộ vi xử lý được ghi bằng các ký tự dưới dạng gợi nhớ (memonic) để người sử dụng để 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 hiểu được. Vì lệnh do bộ vi xử lý được cho dưới dạng mã nên sau khi nhận lệnh., bộ vi xử lý phải thực hiện việc giải mã lệnh rồi sau đó mới thực hiện lênh. Việc hiểu rõ bản chất cách ghi lệnh bằng số hệ 2 cho bộ vi xử lý sẽ có lợi khi ta cần dịch " bằng tay ". Một lệnh gợi nhớ khi làm việc với các " kit " vi xử lý (tuy rằng việc này ít khi xảy ra vì ta thường làm việc với các hệ được trang bị chương trình dịch hợp ngữ).

Một lệnh có thể có độ dài một vài byte tuỳ theo bộ vi xử lý. Giả thiết một bộ vi xử lý nào đó dùng 1 byte để chứa các mã lệnh (opcode) của nó. Ta có thể tính được số lệnh lớn nhất mà 1 byte này có thể mã hoá được là 256 lệnh. Trong thực tế việc ghi lệnh không phải hoàn toàn đơn giản như vậy. Việc mã hoá lệnh cho bộ vi xử lý là rất phức tạp và bị chi phối bởi nhiều yếu tố khác nữa.

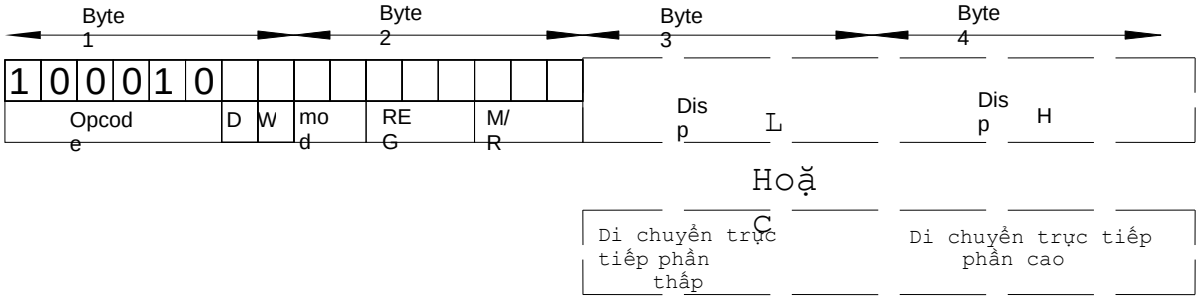
Đối với bộ vi xử lý 8088 một lệnh có thể có độ dài từ 1 đến 6 byte. Ta sẽ chỉ lấy trường hợp lệnh MOV để giải thích cách ghi lệnh nói chung của 8088.

Lệnh MOV *đích, gốc* dùng để nguyển dữ liệu giữa 2 thanh ghi hoặc giữa 2 ô nhớ và thanh ghi. Chỉ nguyên với các thanh ghi của 8088, nếu ta lần lượt đặt các thanh ghi vào các vị trí toán hạng đích và toán hạng gốc ta thấy đã phải cần tới hàng trăm mà lệnh khác nhau để mã hoá tổ hợp các lệnh này.

Hình dưới đây biểu diễn dạng thức các byte dùng để mã hoá lệnh MOV. Từ đây ta thấy rằng để mã hoá lệnh MOV ta phải cần ít nhất là 2 byte, trong đó 6 bit của byte đầu dùng để chứa mã lệnh. Đối với các lệnh MOV. Để chuyển dữ liệu kiểu:

- Thanh ghi ↔ thanh ghi (trừ thanh ghi đoạn) hoặc
- Bộ nhớ ↔ thanh ghi (trừ thanh ghi đoạn) thì 6 bit đầu này luôn là 100010. Đối với các thanh ghi đạon thì điều này lại khác.

Bit W dùng để chỉ ra rằng 1 byte (W = 0) hoặc 1 từ (W = 1) sẽ được chuyển.



Dạng thức Byte mã lệnh của lệnh MOV

Trong các thao tác chuyển dữ liệu, một toán hạng luôn bắt buộc phải là thanh ghi. Bộ vi xử lý dùng 2 hoặc 3 bit để mã hoá các thanh ghi trong CPU như sau:

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

Bit D dùng để chỉ hướng đi của dữ liệu. D = 1 thì dữ liệu đi đến thanh ghi cho bởi b bit của REG.

2 bit MOD (chế độ) cùng với 3 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 (có thể hiểu chế độ địa chỉ là cách tìm ra địa chỉ của toán hạng, xem thêm phần sau của chương này để rõ hơn về chế độ địa chỉ.

Bảng Mod – R/M cho ta thấy cách mã hoá các chế độ địa chỉ (cách tìm ra các toán hạng bằng các bit này).

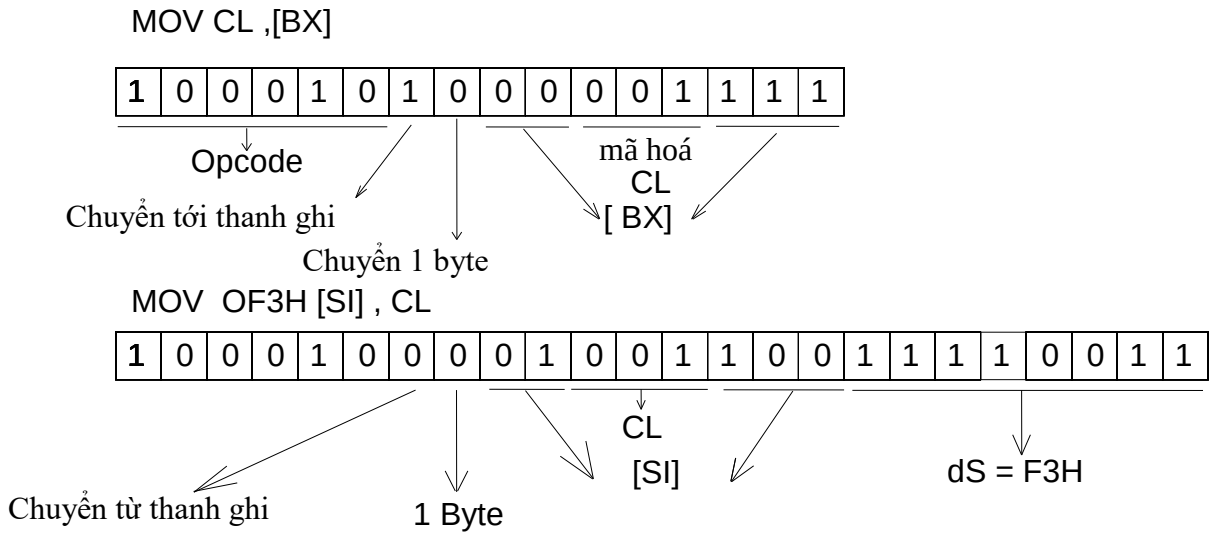
MOD / R/M	00	01	10	11
				W=0 W=1
000	[BX]+[8]	[BX]+[SI]+d8	[BX]+[SI]+d16	AL AX
001	[BX]+[DI]	[BX]+[DI]+d8	[BX]+[DI]+d16	CL CX
010	[BP]+[SI]	[BP]+[SI]+d8	[BP]+[SI]+d16	DL DX
011	[BP]+[DI]	[BP]+[DI]+d8	[BP]+[DI]+d16	BL BX
100	[SI]	[SI]+d8	[SI]+d16	AH SP
101	[DI]	[DI]+d8	[DI] +d16	CH BP
110	d16 (Địa chỉ trực tiếp)	[BP]+d8	[BP]+d16	DH SI
111	[BX]	[BX]+d8	[BX]+d16	BH DI

chế độ bộ nhớ — chế độ thanh ghi

Ghi chú : - *disp , 8bit ,d16: disp , 16bit*

- *Các giá trị cho trong các cột 2,3,4 (ứng với MOD =00,01,10) là các địa chỉ hiệu dụng (EA) sẽ được cộng với DS để tạo ra địa chỉ vật lý (riêng BP phải được cộng với SP)*

Trong các ví dụ sau đây ta sẽ dùng các kiến thức nêu trên để mã hoá một vài lệnh MOV.



3.3. Các chế độ địa chỉ của bộ vi xử lý 8088

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 ra bộ bi xử lý và sau này không thể thay đổi được. Bộ vi xử lý 8088 và cả họ 80x86 nói chung đều có 7 chế độ địa chỉ sau:

1. Chế độ địa chỉ thanh gi (register addressing mode).
2. Chế độ địa chỉ tức thì (immediate addressing mode).
3. Chế độ địa chỉ trực tiếp (direct addressing mode).

4.Chế độ địa chỉ gián tiếp qua thanh ghi (register indirect addressing mode).

5.Chế độ địa chỉ tương đối cơ sở (based indexed relative addressing mode).

6.Chế độ địa chỉ tương đối chỉ số (indexed relative addressing mode).

7.Chế độ địa chỉ tương đối chỉ số cơ sở (based indexed relative addressing mode).

Các chế độ địa chỉ này sẽ được giải thích thông qua các chế độ địa chỉcủa lệnh MOV và lệnh ADD.

✓ **Chế độ địa chỉ thanh ghi**

Trong chế độ địa chỉ này người ta dùng các thanh ghi bên trong CPU như là các toán hạng để chứa dữ liệu cần thao tác. Vì vậy khi thực hiện lệnh có thể đạt tốc độ truy nhập cao hon so với các lệnh có truy nhập đên bộ nhớ.

Ví dụ:

```
MOV   BX, DX   ;  chuyển nội dung DX vào BX.
MOV   DS,AX    ;  chuyển nội dung AX vào DX
ADD    AL,DL    ;  cộng nội dung AL và DL rồi đưa vào
```

✓ **Chế độ địa chỉ tức thì**

Trong chế độ địa chỉ này toán hạng đích là một thanh ghi hay một ô nhớ, còn toán hạng nguồn là một hằng số và ta có thể tìm thấy toán hạng này ở ngay sau mã lệnh (chính vì vậy chế độ địa chỉ này có tên là chế độ địa chỉ tức thì). Ta có thể dùng chế độ địa chỉ này để nạp dữ liệu cần thao tác vào bất kỳ thanh ghi nào (trừ các thanh ghi đoạn và thanh cờ) hoặc vào bất kỳ ô nhớ nào trong đoạn dữ liệu DS.

Ví dụ:

```
MOV   CL, 100          ; chuyển 100 vào CL.
MOV   AX, OFFOH     ; chuyển OFFOH vào AX để rồi đưa
```

```
MOV    DS, AX          ; vào DS ( vì không thể chuyển
                       ; trực tiếp vào thanh ghi đoạn )
MOV    ( BX ), 10      ; chỉ DS:BX.
```

Trong ví dụ cuối ta đã dùng chế độ địa chỉ gián tiếp qua thanh ghi để chỉ ra ô nhớ (toán hạng đích) sẽ nhận dữ liệu ở chế độ địa chỉ tức thì (toán hạng nguồn). Tại đây (BX) có nghĩa là ô nhớ có địa chỉ DS:BX.

✓ **Chế độ địa chỉ trực tiếp**

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

Nếu so sánh với chế độ địa chỉ tức thì ta thấy ở đây ngay sau mã lệnh không phải là toán hạng mà là địa chỉ lệch của toán hạng. Xét về phương diện địa chỉ thì đó là địa chỉ trực tiếp.

Ví dụ:

```
MOV    AL, ( 1234H )    ; chuuyển nội dung ô nhớ  DS:1234
                        ; vào AL.
MOV    ( 4320H ), CX    ; chuyển nội dung CX vào 2 ô nhớ
                        ; liên tiếp  DS:4320 và  DS:4321
```

✓ **Chế độ 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 chứa dữ liệu, còn toán hạng kia chỉ có thể là thanh ghi mà không được là ô nhớ (8088 không cho phép quy chiếu bộ nhớ 2 lần đối với một lệnh).

Ví dụ:

```
MOV  AL, ( BX )         ; chuyển nội dung ô nhớ có địa
                        ;  chỉ DS:BX vào AL.
MOV ( SI ), CL          ; chuyển nội dung CL vào ô nhớ
                        ; có địa chỉ DS:SI.
MOV ( DI ), AX          ; chuyển nội dung AX vào 2 ô nhớ
                        ; liên tiếp có địa chỉ DS:DI và
                        ; DS: ( DI + 1 ).
```

✓ **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 (*displacement values)* được dùng để tính địa chỉ hiệu dụng của toán hạng trong các vùng nhớ DS và SS. Sự có mặt của các giá trị dịch chuyển xác định tính tương đối (so với cơ sở) của địa chỉ.

Ví dụ:

```
MOV  CX, ( BX ) +10     ; chuyển nội dung 2 ô nhớ liên
                        ; tiếp có địa chỉ DS: ( BX + 10 ) và
                        ; DS: ( BX+10 ) vào CX.
MOV  CX, ( BX+10 )      ; một cách viết khác của lệnh trên .
MOV  CX, 10 ( BX )      ; một cách viết khác của lệnh đầu.
MOV   AL, ( BP ) +5     ; chuyển nội dung ô nhớ SS: ( BP+5 )
                        ; vào AL.
ADD  AL, Table ( BX )   ; cộng AL với nội dung ô nhớ do
                        ; BX chỉ ra trong bảng table
                        ;  ( bảng này nằm trong DS ), kết
                        ; quả dựa vào AL.
```

Nhân đây cần làm rõ một số thuật ngữ hay dùng thông qua các ví dụ trên.

- 10.5.Table gọi là các dịch chuyển của các toán hạng tương ứng. 10 và 5 là các giá trị cụ thể. Table là tên mảng biểu diễn kiểu dịch chuyển của mảng (phần tử đầu tiên) so với địa chỉ đầu của đoạn dữ liệu DS.
- (BX+10) hoặc (BJP+5) gọi là địa chỉ hiệu dụng (effective address. EA.theo cách gọi của Intel).
- DS: (BX+10) hoặc SS: (BP+5) chính là logic tương ứng với một địa chỉ vật lý.
- Theo cách định nghĩa này thì địa chỉ hiệu dụng của một phần tử thứ BX nào đó (kể từ 0) trong mảng Table (BX) thuộc đoạn DS là EA = Table+BX và của phần tử đầu tiên là EA = Table.

✓ **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 địa chỉ này ta dùng cả 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 phức hợp nhất: chế độ địa chỉ tương đối chỉ số cơ sở. Ta có thể thấy chế độ địa chỉ này rất phù hợp cho việc dịa chỉ hoá các mảng hai chiều

Ví dụ:

```
MOV   AX, [ BX ] [SI ]+8      ; chuyển nội dung 2 ô nhớ
                              ; liên tiếp có địa chỉ
                              ; DS:(BX+SI+8 ) và
                              ; DS:(BX+SI+9 ) vào AX
MOV   AX, [BX+SI+8]           ; một cách viết khác của lệnh trên
MOV    CL, [BP+DI+5]          ; chuyển nội dung ô nhớ
                              ; SS:( BP+DI+5 ) vào CL.
```

3.4. Mô tả tập lệnhcủa bộ vi xử lý 8088.

Có nhiều cánh trình bày tập lệnh của bộ vi xử lý:Trình bày các lệnh cho các nhóm hoặc theo thứ tự ABC .Ta sẻ chọn cách làm thứ 2 để sau này dễ tìm kiếm cac lệnh cần tra cứu cụ thể.Trong khi nói tới các lệnh ở dạng gợi nhớ tacũng mô tả ngắn gon luôn từng lệnh và tác động (nếu có) của lệnh tới các cờ. Để cho các diễn giải dễ đọc ta quy định kí hiệu AL được hiểu là thanh ghi AL hoặc là nội dung của AL.Trong khi ghi lệnh ,dấu[X] nên được hiểu như là một ki hiệu của Intel để ghi lệnh. Không nên hiểu là 'nội dung' của X ,còn {XX:YY} dùng để chỉ nội dung ô nhớ tại địa chỉ XX:YY hoặc {SP} dùng để chỉ ô nhớ của ngăn xếp có địa chỉ do nội dung của thanh ghi con trỏ ngăn xếp SP chỉ ra .

AAA _ASCII Adjust after Addition (Chỉnh sau khi cộng hai số ở dạng ASCII)

Dữ liệu truyền từ các thiết bị đầu cuối đến máy tính thường ở dưới dạng mã ASCII .Khi dã truyền đi các số dưới dạng ASCII roòi,đôi khi ta muốn cộng luôn các số đó.Bộ vi xử lý 8088 cho phep ta làm điều này với điều kiện phải chỉnh lại kết quả có trong AL,bằng lệnh AAA để thu được kết quả là số BCD không gói.

Cập nhật : AF , CF

Không xác định: OF , PF ,SF ,ZF

Ví dụ:Ta có 2 số dưới dạng mã ASCII là 30H và 39H ứng với ' 0 ' và ' 9 '

Nếu cộng hai số ở dạng mã lại ta được số 69H.Số náy không có ý nghĩa gì vì nó không phải la số BCD đúng .Ta sẻ thu được số BCD không gói nếu dùng thêm lệnh AAA .

```
                ;AL = 0011  0000B = 30H = '0',
                ;BL = 0011  1001B = 39H = ' 9',
ADD AL ,  BL ; thu được AL = 0110 1001B = 69H , Kết quả sai.
```

AAA ; thu được AL = 0000 1001B = 9, kết quả đúng.

OR AL ,30H ; thu được AL = 39H = '9' đê truyền kết quả trở lại thiết bị đầu cuối .

AAD_ ASCII Adjust before Division (Chỉnh trước khi chia 2 số ơ dạng ASCII)

Lệnh này đổi 2 số BCD khônbg gói ở AH và AL sang số hệ 2 tương đương để tại Al.Viêc này phải thực hiện trước khi làm phép chia một số BCD không gói(gồm 2 chữ số) để trong AX cho 1 số BCD không gói khác.Kết quả và số dư cũng là cac số BCD không gói.

Không xác định :tất cả các cờ .

Ví dụ:

;AX = 0605H là số BCD không gói của 65

;(số bị chia)

;BL = 08H la số BCD không gói

;(số chia).

AAD ;sau khi chỉnh AX = 0041 = 41H

DIV BL;sau khi chia đươc thương AL = 08

;số dư ở AH = 1 là số BCD không gói ,

AAM_ASCII Adjust After Mult iplication (Chỉnh sau khi nhân 2 số ở dạng ASCII)

Lệnh này dung để đổi 1 số hệ 2,là tích của 2 số BCD không gói ,có trong AL sang số BCD không gói để tại AX .

Cập nhật :PF , SF ,ZP.

Không xác định: AF ,CF ,OF

Ví dụ:

Sau khi nhân 2 số 5 và 9 ở dang ASCII .Ta đổi kết quả sang dạng BCD không gói bằng lệnh AAM và sau đó đổi tiếp thành mã ASCII để truyền tiếp

;AL = 0011 0101B = 35H = '5' ,

;BL = 0011 1001B = 39H = '9' ,

MUL BL; thu được AX = 002DH = 45,

AAM; thu được AX = 0405H, mã BCD

;không nen của 45.

OR AX ,3030H; thu được AX = 3435H, mã ASCII

;cho 45 để truyền kết quả

; trở lại thiết bị đầu cuối.

AAS-ASCH Adjust after Subtraction (chỉnh sau khi trừ 2 số ở dạng ASCH)

Lệnh này dùng để đổi một số hệ hai là hiệu của 2 số BCD không gói, có ở AL. sang số BCD không gói.

Cập nhập: AF, CF.

Không xác định: OF, PF, SF, ZP.

Ví dụ:

; BL = 0011 0101B = 35 = '5',

; AL = 0011 1001B = 39H = '9',

; ASCII 9 – ASCII 5:

SUB AL, BL ; thu được AL = 04H = 4,

AAC ; thu được AL = 04H, mã BCD không

; gói của 4.

```
OR  AL, 30H     ; thu được AL = 34H, mã ASCII cho 4
                ; để truyển kết qủa trở lại thiết bị
                ; đầu cuối.
              ; AL = 0011  0101B = 35H = '5',
; BL = 0011  1001B = 39H  = ' 9',
                ; ASCII 5- ASCII 9:
SUB AL, BL      ; thu được AL = FCH = -4, CF = 1,
AAS             ; thu được AL = 04H, mã BCD không
                ; gói của 4, CF = 1 ( có thể dùng cho các
                ; phép trừ nhiều chữ số )
```

ADC-Add With Carry (cộng có nhớ)

Viết lệnh: ADC Đích, Gốc.

Mô tả: Đích Đích + Gốc + CF

Trong đó tớn hạng đích và gốc có thể tìm đwocj theo các chế độ địa chỉ khác nhau. Nhưng phải chứa dữ liệu có cùng độ dài và không được phép đồng thời là 2 ô nhớ và cũng không được là thanh ghi đoạn. Điều hạn chế này cũng áp dụng cho các lệnh khác có ngữ pháp tương tự.

Cập nhật: AF, CF, OF, PF, SP, ZP.

Ví dụ: Các ví dụ sau đây có thể đại diện chó các chế độ địa chỉ có thể có trong lệnh cộng này cũng như một số các lệnh khác vúoi ngữ pháp tương tự.

ADD-Add (cộng 2 toán hạng).

Viết lệnh: ADD Đích, Gốc.

Mô tả: Đích - Đích + Gốc.

Tróng đó toán hạng đích và gốc có thể tìm được theo các chế độ địa chỉ khác nhau. Nhưng phải chứa dữ liệu có cùng độ dài và không được phép đồng thời là 2 ô nhó và cũng không được là thanh ghi đoạn. Có thể tham khảo các ví dụ của lệnh ADC.

Cập nhật: AF, CF, PF, SF, ZP/

AND-And Corresponding Bits of Two Operands (Và 2 toán hạng)

Viết lệnh: AND Đích, Gốc

Mô tả: Đích - Đích, Gốc.

Trong đó toán hạng đích và gốc có thể tìm được thoe các chế độ địa chỉ khác nhau. Nhưng phải chứa dữ liệu cùng độ dài và không được phép đồng thời là 2 ô nhớ và cũng không được là thanh ghi đoạn. Phép AND thường dùng 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 chỗ cần che đi/giữ nguyên tương ứng (toán hạng tức thì lúc này còn được gọi là mặt nạ).

Xoá: CF, OF.

Cập nhật: PF, SF, ZP, PF chỉ có nghĩa khi toán hạng là 8 bit.

Không xác định: À.

Ví dụ:

```
AND  AL, BL   ; AL , AL  BL theo từng bit.
AND     OFH ; che 4 bit cao của BL.
```

CALL-Call o Proceduce (Gọi chương trình con)

Mô tả:

Lệnh này dùng để chuyển hoạt động của bộ vi xử lý từ chương trình chính (CTC) sang chương trình con (ctc). Nếu ctc ở trong cùng một đoạn mã với CTC thì ta có gọi gần (near

call). Nếu CTC và ctc nằm ở hai đoạn mã khác nhau thì tra có gọi xa (far call). Gọi gần và gọi xa khác nhau về cách tạo ra địa chỉ trở về (return address). Địa chỉ trở về là địa chỉ của lệnh tiếp ngay sau lệnh Call. Khi gọi gần thì chỉ cần các IP của địa chỉ trở về (vì CS không đổi). Khi gọi xa thì phải cất cả CS và IP của địa chỉ trở về. Địa chỉ trở về được tự động cất tại ngăn xếp khi bắt đầu thực hiện lệnh gọi và được tự động lấy ra khi gặp lệnh RET (trở về CTC từ ctc) tại cuối ctc.

Viết lệnh: Sau đây là ví dụ các dạng khác nhau của các dạng khác nhau của các lệnh gọi ctc và cách tính địa chỉ của ctc:

CALL Multiple: Gọi ctc có tên là Multiple trong cùng đoạn mã với CTC, ctc này phải nằm trong giới hạn đích chuyển-32Kbyte (dịch về phái địa chỉ thấp) hoặc (32K-1) byte (dịch về phía địa chỉ cao) so với lệnh tiếp theo ngay sau lệnh Call. Sau khi cất IP cũ (địa chỉ trở về) vào ngăng xếp . IP mới được tính: IP –IP + Dịch chuyển.

CALL Divi: Gọi ctc có tên Divi ở đạon mã khác. Trong chương trình hợp ngữ Divi phải được khai báo là một ctc ở xa:

Divi Proc Far

Đại chỉ của ctc là đại chỉ CS:IP cảu Divi.

CALL WORD PTR [BX]: Gọi ctc nằm trong cùng đạon mã, ctc có địa chỉ dịch chuyển (tính từ lệnh tiếp ngay sau lệnh gọi tới lệnh đầu tiên của ctc) chứa trong 2 ô nhớ do BX và BX+1 chỉ ra trong đoạn DS. Địa chỉ lệch này sẽ đưa vào IP (SI, DI có thể dùng thay chỗ của BX).

CALL DWORD PTR [BX]: Gọi ctc không nằm trongcùng một đoạn mã, ctc có địa chỉ CS:IP, giá trị gần cho IP và CS chứa trong 4 ô nhơ do BX và BX +1 (cho IP) và BX+2 và BX+3 (cho CS chỉ ra trong đoạn DS (SI, DI có thể dùng thay chỗ của BX).

CBW-Convert a Byte to a Word (Chuyển byte thành từ)

Lệnh này mở rộng bit dấu của AL sang 8 bit của AH, AH lúc này được gọi là phần mở rộng dấu cuả AL. Ta dùng CBW để mở rộng dấu cho số có dấu nằm trong AL trước khi muốn chia nó cho một số có dấu 8 bit khác bằng lệnh IDIV (lệnh chia các số có dấu), hoặc trước khi muốn nhân nó với một số có dấu 16 bit khác bằng lệnh IMUL, (lệnh nhân các số có dấu).

Lệnh này không tác động đến các cờ.

Ví dụ: Nếu AL = 80 thì sau lệnh chuyển ta có AX = PF80H.

CLC-Clear the Carry Flag (xoá cờ nhớ)

Mô tả: CF – 0.

Không tác độn đến các cờ khác.

CLD – Clear the Direction Flag (xoá cờ hướng).

Mô tả: DF – 0.

Lệnh này định hướng thao tác theo chiều triển chó các lệnh liên quan đến chuỗi. Các thanh ghi liên quan là SI và DI sẽ được tự động tăng khi làm việc xong với một phần tử của chuỗi.

Không tác động đến các cờ khác.

CLI – Clear the Interrupt Flag (xóa cờ cho phép ngắt).

Mô tả: IF – 0.

Lệnh này xoá cờ cho phép ngắt. Các yếu tố ngắt che được sẽ bị che.

Không tác động đến các cờ khác.

CMC – Complement the Carry Flag (Đảo cờ nhớ).

Mô tả: CF – CF.

Cập nhật: CF

Không tác động đến các cờ khác.

CMP-Compare Byte or Word *~(so sánh 2 byte hay 2 từ).

Viết lệnh: **CMP Đích, Gốc.**

Mô tả: Đích – Gốc.

Trong đó toán hạng đích và gốc có thể tìm được theo các chế độ địa chỉ khác nhau. Nhưng phải chứa dữ liệu có cùng độ dài và không được phép đồng thời là 2 ô nhớ.

Lệnh này chỉ tạo các cờ, không lưu kết quả so sanh, sau khi so sanh các toán hạng không bị thay đổi. Lệnh này thường được dùng để tạo cờ cho các lệnh nhảy có điều kiện (nhảy theo cờ).

Các cờ chính theo quan hệ đích và gốc khi so sánh 2 số không dấu:

	CF	2F
Đích = Gốc	0	1
Đích > Gốc	0	1
Đích > Gốc	1	0

Cập nhật: AF, CF, OF, PF, SF, ZP.

CMPS/CMPSB/CMPSW-Compare String Bytes or String Word (so sánh 2 chuỗi byte hay 2 chuỗi từ).

Viết lệnh: **CMPS Chuỗi đích, chuỗi gốc.**
CMPSB
CMPSW.

Mô tả: Chuỗiđích – Chuỗigốc.

Lệnh này so sánh từng phần tử (byte hay từ) của 2 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 gốc.

+ES:DI là địa chỉ của phần tử so sánh trongchuỗi đích..

Ta sẽ giải thích cụ thể các trường hợp dùng các dạng lệnh trên. Giải thích này cũng có thể áp dụng cho cáclện có dạng thức lện hoặc cấu trúc ngữ pháp ương tự.

MOVS, STOS, LODS, SCAS.

Có 2 cách để chỉ ra một chuỗi là chuỗi byte hoặc chuỗi từ. Cách đầu tiên là ta khai rõ bằng ten ngay từ đầu chuỗi nguồn và chuỗi đích là loại gì. Sau đó ta dùng lệnh COMPS để thao tác với các chuỗi đó.

```
StrByte1      DB    "daylachuoibyte1"
StrByte2      DB    " daylachuoibyte2"
StrWord1      DW    " daylachuoitư1"
StrWord1      DW    "daylachuoitu2"
              LEA  SI, StrByte1
              LEA  DI, StrByte2
              COMPS    StrWord2, StrWord1    ;có thể thay
                                              ; bằng MOMPSB.
```

Cách ths hai là ta thêm vào lệnh CMPS đuôi thích hoẹp để báo cho chương trình dịch biết kiểu thao tác trên chuỗi đã được định nghĩa: đuôi "B" để thao tác với byte hoặc đuôi "W"để thao tác với từ trong chuỗi.

Cập nhật: AF, CF, OF, PF, SF, ZP.

Ví dụ:

```
MOV  DI, OFFSET    chuỗiđích        ; lấy địa chỉ lệch
```

```
                                    ; của chuỗiđích tại
                                    ; ES vào SI,
MOV  SI, OFFSET    chuỗigốc         ; lấy địa chỉ lệch
                                    ; của chuỗigốc tại
                                    ; DS vào SI,
CLD                                 ; làm việc với chuỗi theo
                                    ; chiều tiến,
CMPSB                               ; chuyển 1 byte.
                                    ; SI và DI tăng thêm 1.
```

CWD-Convert a Word to a DoubleWord (chuyển từ thành từ kép)

Lệnh này mở rộng bit dấu của AX sang 16 bit của DX. DX lúc này được gọi là phần mở rộng dấu của AX. Ta dùng CWD để mở rộng dấu cho số có dấu nằm trong AX trước khi muốn chia nó cho một số có dấu khác bằng lệnh IDIV.

Lệnh này không tác động đến các cờ.

Ví dụ: nếu DX = 0000H. AX = 8087H thì sau lệnh đổi ta có:

DX = FFFFH, AX = 8086H.

DAA-Decimal Adjust AL after BCD Addition (chỉnh AL sau khi cộng số BCD).

Lệnh này dùng để chỉnh lại kết quả (hiện nằm ở AL) sau phép cộng 2 số BCD. Lý do phải chỉnh lại kết quả này là do ta đã dùng bộ ALU của XPU, cốn chỉ biết làm toán với các số hệ hai. Để làm otán với các số VCD, lệnh DÂ chỉ tác độ đúng đến kết quả ở AL ngay sau khi vừa thực hiện phép cộng. Hoạt động của lệnh DAA:

+Nếu 4 bit thấp cảu AL lớn hơn 9 hoặc AF = 1 thì AL – AL + 6 .

+Nếu 4 bit cao của AL lớn hơn 9 hoặc CF = 1 thì AL – AL + 60H.

Cập nhật: AF, CF, PF, SF, ZP.

Không xác định: OF.

Ví dụ:

```
a)                  ; AL = 0101   1001BCD = 59
                    ; BL = 0011   0110BCD = 36
ADD  AL, BL         ; AL = 1000   1111 B   = 8FH
DAA                 ; vì  F > 9 nên  AL + 6 = 1001   0101BCD = 95
b)                  ; AL = 1000   1001BCD = 89
                    ; BL = 0100   0111BCD = 47
ADD  AL, BL         ; AL = 1101   0000  B = D0H, AF = 1
DAA                 ; vi D > 9 và À = 1 nên
                        ; AL + 60H +6 = 1001   0000BCD = 36,CF = 1.
```

DAS- Decimal Adjust AL after BCD Subtraction (chỉnh AL sau khi trừ 2 số BCD)

Lệnh này dùng để chỉnh lại kết quả (hiện nằm ở AL) sau phép trừ 2 số BCD. Lý do phải chỉnh lại kết qủ này là do ta đã dùng bộ ALU của CPU, vốn chỉ biết làm toán với các số hệ hưi, để làm toán vcới các số BCD. Lệnh DAS chỉ tác động đúng đến kết quả ở AL ngay sau khi vừa thực hiện phép trừ. Hoạt động của lệnh DAS:

+Nếu 4 bit thấp của AL lớn hơn 9 hoặc AF = 1 thì AL – AL.6.

+Nếu 4 bit cao của AL lớn hơn 9 hoặc CF = 1 thì AL – AL.60H.

Cập nhật: AF, CF, PF, SP ,ZP.

Không xác định: OF.

Ví dụ:

a) ; AL = 0101 0110_{BCD} = 56

; BL = 0011 1001_{BCD} = 39

SUB AL, BL ; AL = 0001 1101B = 1DH.

DAS ; vì D > 9 nên AL-6 = 0001 1001_{BCD} = 99, CF.

Trong thí dụ trên CF = 1 có nghĩa là phải mượn 100 thêm vào số bị trừ để được kết quả là 99. Nói cách khác đi kết quả đúng sẽ là -1.

DEC – Decrement Destination Register or Memory (Giảm toán hạng đi 1).

Viết lệnh : DEC Destination

Mô tả: Đích – Đích -1.

Trong đó toán hạng đích có thể tìm đực thưo các chế độ địa chỉ khác nhau. Lưu ý là nếu Đích = 00H (hoặc 0000H) thì Đích -1 = FFH (hoặc FFFFH) mà không làm ảnh hưởng đến cờ CF. Lệnh này cho kết quả tương đương như lệnh SUB Đích nhưng chạy nhanh hơn.

Cập nhật: AF, OF, PF, SF, ZP.

Không tác động: CF/

DIV – Unsingned Divide (chia 2 số không có dấu)

Viết lệnh: DIV Gốc

Trong đó toán hạng Gốc là số chia và có thể tìm được theo các chế độ địa chỉ khác nhau.

Mô tả: tuỳ theo độ dài của toán hạng gốc ta có 2 trường hợp bố trí phép chia. Các chỗ để ngầm định cho số bị chia và kết quả:

- Nếu Gốc là số 8 bit: AX/Gốc. Số bị chia phải là số không dấu 16 bit để trong AX.
- Nếu Gốc là số 16 bit: DXAX/Gốc. Số bị chia phải là số không dấu 32 bit để trong cặp thanh ghi DXAX.

Nếu thương không phải là số nguyên nó được làm tròn theo số nguyên sát đuôi.

Nếu Gốc = 0 hoặc thương thu được lớn hơn FFH hoặc FFFFH (tuỳ theo độ dài của toán hạng Gốc) thì 8088 thực hiện lệnh ngắt INT 0.

Không xác định: AF, CF, OF, PF, SF, ZP.

ESC – Escape

Lệnh này dùng để truyêng các lệnh cho bộ đồng xử lý toán học 8087 bị tạm dừng và bộ vi xử lý 8088 bước vào *trạng thái dừng.* Để thoát khỏi trạng thái dừng chỉ có cách tác động vào một trong các chân INTR.NMI. hoặc RESET của bộ vi xử lý.

IDIV – Integer Division (Signed division) (chia số có dấu)

Viết lệnh: IDIV Gốc

Trong đó toán hạng Gốc là số chia và có thể tìm được theo các chế độ địa chỉ khác nhau.

Đây là lệnh dùng để chia các số nguyên có dấu. Chỗ để ngầm định của số chia. Số bị chia. Thương và số dư giống như ở lệnh DIV. chỉ có 2 điều khác là:

+Sau phép chia AL chứa thương (số có dấu). AH chứa số dư (số có dấu).

+Dấu của số có dư sẽ trùng với dấu của số bị chia.

+Nếu Gốc = 0 hoặc thương nằm ngoài dải.128…+ 127 hoặc -32768…+32767 (tuỳ theo độ dài của Gốc) thì 8088 thực hiện lệnh ngắt INT 0.

Không xác định: AF, CF, OF, PF, SF, ZP.

IMUL – Integer Multiplication (Multiply Signed Numbers) (Nhân số có đầu).

Viết lệnh: ***IMUL Gốc.***

Trông dố toán hạng Gốc là số nhân và có thể tìm được theo các chế độ địa chỉ khác nhau.

Mô tả: tuỳ theo độ dài của toán hạng Gốc ta có 2 trương hợp bố trí phép nhân. Chỗ để ngầm định cho só bị nhân và kết quả:

- Nếu Gốc là số có dấu 8 bit: ALxGốc.
  Số bị nhân phải là số có dấu 8 bit để trong AL.
- Nếu Gốc là số có dấu 16 bit: AXxGốc.
  Số bị nhân phải là số có dấu 16 bit để trong AX.

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

Nếu byte cao (hoặc 16 bit cao) của 16 (hoặc 32 bit) kết quả chỉ chưa một giá trị của dấu thì CF = OF = 0.

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

Như vậy CF và OF sẽ báo cho ta biết kết quả cần độ dài thực chất là bao nhiếu.

Ví dụ:

Nếu ta cần nhân một số có dấu 8 bit với một số có dấu 16 bit, ta để số 16 bit ở gốc và số 8 bit ở AL. Số 8 bit này ở AL cần phải được mở rộng dấu sang AH băng lệnh CBW. Sau cùng chỉviệc dùng lệnh IMUL gốc và kết quả có trong cặp DXAX.

Cập nhật:CF, OF.

Không xác đinh: AF, PF, FS, ZP.

In- Input Data From a Port (đọc dữ liệu từ cổng vào thanh ACC.

Viết lệnh: In ACC, Port.

Mô tả: ACC <- {Port}.

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

+Nếu ACC là AL thì dữ liệu 8 bit được đưa vào từ cổng Port.

+Nếu ACC là AX thì dữ liệu 16 bit được đưa vào từ cổng Port và cổng Port+1.

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

In ACC, DX.

Trong đó DX phải được gắn từ trước giá trị ứng với địa chỉ cổng.

Lệnh này không tác động đến các cờ.

Inc-Increment Destination Register or Memory (tăng toán hạng đích thêm 1).

Viết lệnh : Inc-Đích

Mô tả: Đích <- Đích+1.

Trong đó toán hạng đích có thể tìm được theo các chế độ địa chỉ khác nhau. Lưu ý là nếu đích = FFH (hoặc FFFFH) thì Đích+1 = 00H (0000H) mà không ảnh hưởng đến cờ CF. Lệnh này cho kết quả tương đương như lệnh ADD Đích.1.nhưng chạy nhanh hơn.

Cập nhật: AF, OF, PF, SF, ZP.

Không tác động: CF.

INT-Interupt Program Execution (ngắt, gián đoạn chương trình đang chạy).

Viết lệnh: INT N, N = 0..FFH

Mô tả: các thao tác của 8088 khi chạy lệnh INT N:

SP <- SP-2, {SP} <- FR

IF <- 0 (cấm các ngắt tác động). TF <-0 (chạy suốt).

SP <- SP-2, {SP} <- CS

SP <- SP-2, {SP} <- IP

{N_x4} <- IP, {N_x4+2} <-CS.

Ví dụ với N = 8 thì CS <- {0022H}.

IP <- {0020H}.

Mỗi lệnh ngắt ứng với chương trình phục vụ ngắt (CTPVN) khác nhau có địa chỉ lấy từ bảng veto ngắt. Bảng này gồm 256 vecto, chứa địa chỉ của các CTPVN tương ứng và chiếm 1Kbyte Ram có địa chỉ thấp nhất của bộ nhớ. CTPVN cũng có thể được gọi là chương trình con phục vụ ngắt (CTCPVN) vì cách thức tổ chức và quan hệ giữa nó với chương trình bị ngắt cũng giống như cách thức tổ chức và quan hệ giữa CTC với ctc.

INTO-Interrupt On Overflow (ngắt nếu có tràn).

Nếu có tràn (OF = 1) thì lệnh này ngắt công việc đang làm của vi xử lý và thực hiện lệnh ngắt INT 4.

IRET-Interrupt Return (trở về CTC từ chương trình (Con) phục vụ ngắt).

Như đã trình bày ở lệnh CALL, tại cuối ctc phải có lệnh trở về (RET) để bộ vi xử lý tự động lấy lại địa chỉ trở về CTC. Trong trường hợp CTCPVN, để trở về CTC với đầy đủ thông tin cần thiết về địa chỉ và trạng thái, tất nhiên phải cần có lệnh với các tác động tương ứng: lệnh IRET. Lệnh này, ngoài việc tự động lấy lại địa chỉ trở về CTC, còn lấy lại thanh ghi cờ đã được cất giữ trước khi chạy CTCPVN.

JA/JNBE-Jump If Above/Jump If Not Below Or Equal (nhảy nếu cao hơn/nhảy nếu không thấp hơn hoặc bằng).

Viết lệnh: JA NHAN

JNBE NHAN

Mô tả IP ←IP → Dịchchuyển

Hai lệnh trên điều khiển cùng một thao tác *Nhảy có điều kiện* với nhản nếu CF+ZF = 0 .Quan hệ "trên " (above)," cao hơn " và quan hệ "dưới " , " thấp hơn" (below) là các quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn của hai số không dấu .Nhãn NHAN phải nằm cách xa (dịch chuyển một khoảng)-128. . +127byte so với lệnh tiếp theo sau lệnh A:/INBE .Chương trình sẽ căn cứ vào giá trị chuyển để xác định các giá trị chuyển

Lệnh này không tác động đến các cờ .

Vi dụ :Nếu 1 khung thanh AL cao hơn 10H thì nhảy lên nhãn TH01

CMP AL , 10H ; so sánh Al với 10H

UA THOI ; nhảy lên TH01 nếu Al cao hơn

4AE/JNB/4NC – 4jump if Above or Equal /jump if not below /jump if no carry (nhảy nếu cao hơn hoặc bằng / nhảy nếu thấp hơn / nhảy nếu không có nhỏ)

Viết lệnh :

JAE NHAN

JNB NHAN

JNC NHAN

Mô tả :

IP ←IP → Dịchchuyển

Ba lệnh trên đều thực hiện cùng một thao tác : nhảy có điều kiện tới NHAN nếu CF = 0 . Quan hệ "trên " (above)," cao hơn " và quan hệ "dưới " , " thấp hơn" (below) là các quan

hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn của hai số không dấu .Nhãn NHAN phải nằm cách xa (dịch chuyển một khoảng)-128. . +127byte so với lệnh tiếp theo sau lệnh A:/INBE .Chương trình sẽ căn cứ vào giá trị chuyển để xác định các giá trị chuyển

Lệnh này không tác động đến các cờ .

Ví dụ : Nếu nội dung thanh AL cao hơn hoặc bằng 10H thì nhảy đến nhãn THOI

CMP AL ,10H ; So sánh AL với 10H

JAE .THOI ; nhảy đến THOI nếu Al cao hơn hoặc bằng 10H

JB/JC/JNAE - jump if Below/ Jump if Carry /Jump ì Not Above or Equal (Nhảy thấp hơn / nhảy nếu có / nhảy nếu không cao hơn hoặc bằng)

Viết lệnh

JB NHAN

JC NHAN

JNAE NHAN

Ba lệnh trên đều thực hiện cùng một thao tác : nhảy có điều kiện tới NHAN nếu CF = 0 . Quan hệ "trên " (above)," cao hơn " và quan hệ "dưới " , " thấp hơn" (below) là các quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn của hai số không dấu .Nhãn NHAN phải nằm cách xa (dịch chuyển một khoảng)-128. . +127byte so với lệnh tiếp theo sau lệnh A:/INBE .Chương trình sẽ căn cứ vào giá trị chuyển để xác định các giá trị chuyển

Lệnh này không tác động đến các cờ .

Ví dụ : Nếu nội dung thanh AL thấp hơn hoặc bằng 10H thì nhảy đến nhãn THOI

CMP AL ,10H ; So sánh AL với 10H

JB THOI ; nhảy đến THOI nếu Al thấp hơn hoặc bằng 10H

JBE/JNA- jump if Below/ Jump if Carry /Jump ì Not Above or Equal (Nhảy thấp hơn / nhảy nếu có / nhảy nếu không cao hơn hoặc bằng)

Viết lệnh : JBENHAN

IBA NHAN

Mô tả :

IP ←IP → Dịchchuyển

Hai lệnh trên đều thực hiện cùng một thao tác : *nhảy có điều kiện* tới NHAN nếu CF+ZF=1 . Quan hệ "trên " (above)," cao hơn " và quan hệ "dưới " , " thấp hơn" (below) là các quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn của hai số không dấu .Nhãn NHAN phải nằm cách xa (dịch chuyển một khoảng)-128. . +127byte so với lệnh tiếp theo sau lệnh A:/INBE .Chương trình sẽ căn cứ vào giá trị chuyển để xác định các giá trị chuyển

Lệnh này không tác động đến các cờ .

Ví dụ : Nếu nội dung thanh AL thấp hơn hoặc bằng 10H thì nhảy đến nhãn THOI

CMP AL ,10H ; So sánh AL với 10H

JBE THOI ; nhảy đến THOI nếu Al thấp hơn hoặc bằng 10H

JBE/JNA- jump if Below/ Jump if Carry /Jump ì Not Above or Equal (Nhảy thấp hơn / nhảy nếu có / nhảy nếu không cao hơn hoặc bằng)

Viết lệnh : JCXZ NHAN

Mô tả :

IP ←IP + Dịchchuyển

Đây là lệnh *nhảy điều kiện* tới NHAN nếu CX =0 và không có liện hệ gì với cờ ZF . Nhãn NHAN phải nằm cách xa (di chuyển một khoảng) -128. . +127byte so với lệnh tiếp

theo sau lệnh A:/INBE .Chương trình sẽ căn cứ vào giá trị chuyển để xác định các giá trị chuyển

Lệnh này không tác động đến các cờ .

Ví dụ : Nếu thanh CX rỗng thì nhảy đến lệnh THOI

```
:
JCXZ THOI
:
THOI :RET  ; Trở về CTC nếu CX = 0
```

JE/JZ - Jump ì Equal /jump if Zero (Nhảy nếu bằng nhau /Nhảy nếu kết quả bằng không)

Viết lệnh : JE NHAN

JZ NHAN

Mô tả IP ←IP + Dịchchuyển

Hai lệnh trên đều thực hiện cùng một thao tác : *nhảy (có điều kiện)* tới NHAN nếu ZF=1.Nhãn NHAN phải nằm cách xa (dịch chuyển một khoảng)-128. . +127byte so với lệnh tiếp theo sau lệnh JE/JZ .Chương trình sẽ căn cứ vào giá trị chuyển để xác định các giá trị chuyển

Lệnh này không tác động đến các cờ .

Ví dụ : Nếu nội dung thanh AL bằng 10H thì nhảy đến nhãn THOI

```
SUB AL ,10H ; AL trừ giá trị cần quan tâm
JE THOI ; Nhảy đến THOI nếu AL bằng 10H
```

Viết lệnh : JGN HAN

JNLE NHAN

Mô tả IP ←IP + Dịchchuyển

Hai lệnh trên đều thực hiện cùng một thao tác : *nhảy (có điều kiện)* tới NHAN nếu (SF⊕OF)+ZF =0 . Quan hệ "lớn hơn " (greater than)," bé hơn "(less than) và các quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn của hai số không dấu .Lớn hơn có nghĩa là dương hơn .Nhãn NHAN phải nằm cách xa (dịch chuyển một khoảng)-128. . +127byte so với lệnh tiếp theo sau lệnh JG/JNLE .Chương trình sẽ căn cứ vào giá trị chuyển để xác định các giá trị chuyển

Lệnh này không tác động đến các cờ

Ví dụ : Nội dung các thanh AL lớn hơn 10H thì nhảy đến nhản THOI

```
CMP AL , 10H ; So sánh AL với 10H
JG THOI ; Nhảy đến THOI nếu AL lớn hơn 10H
```

JGE/JNL - Jump if Greater than or Equal /Jump if Not Less than (Nhảy nếu lớn hơn hoặc bằng / Nhảy nếu không bé hớn)

Viết lệnh

JGE NHAN

JNL NHAN

Mô tả IP ←IP + Dịchchuyển

Hai lệnh trên đều thực hiện cùng một thao tác : *nhảy (có điều kiện)* tới NHAN nếu (SF⊕OF =0 . Quan hệ "lớn hơn " (greater than)," bé hơn "(less than) và các quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) độ lớn của hai số không dấu .Lớn hơn có nghĩa là dương hơn .Nhãn NHAN phải nằm cách xa (dịch chuyển một khoảng)-128. . +127byte so với lệnh tiếp theo sau lệnh JG/JNLE .Chương trình sẽ căn cứ vào giá trị chuyển để xác định các giá trị chuyển

Lệnh này không tác động đến các cờ

Ví dụ : Nếu nội dung thanh AL lớn hơn hoặc bằng 10H thì nhảy đến nhãn THOI :

CMP AL , 10H ; So sánh AL với 10H

Viết lệnh : JG NHAN

JNLE NHAN

Mô tả : IP ← IP + Dịchchuyển.

Hai lệnh trên biểu diễn cùng một thao tác : nhảy (có điều kiện) tới NHAN nếu (SF⊕OF)+ZF=0. Quan hệ "lớn hơn" (greater than) và "bé hơn " (less than) là các quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) của 2 số có dấu. Lớn hơn có nghĩa là dương hơn. Nhãn NHAN phải nằm cách xa (dịch đi một khoảng) - 128 .. + 127 byte so với lệnh tiếp theo sau lệnh JG/JNLE . chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Lệnh này không tác động đến các cờ :

Ví dụ : Nếu nội dung thanh AL lớn hơn 10H thì nhãy đến nhãn THOI :

```
CMP  AL, 10H  ;  so sánh  AL với  10H
JG  THOI        ; nhảy đến THOI  nếu   AL   lớn   hơn   10H.
```

JGE/JNL - Jump if Greater than or Equal/jump if not less than

(Nhảy nếu lớn hơn hoặc bằng /Nhảy nếu không bé hơn)

Viết lệnh : JG NHAN

JNLE NHAN

Mô tả : IP ← IP + Dịchchuyển.

Hai lệnh trên biểu diễn cùng một thao tác: nhảy (có điều kiện) tới NHAN nếu (SF⊕OF)=0. Quan hệ "lớn hơn" (greater than) và "bé hơn " (less than) là các quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) của 2 số có dấu. Lớn hơn có nghĩa là dương hơn. Nhãn NHAN phải nằm cách xa (dịch đi một khoảng) - 128 .. + 127 byte so với lệnh tiếp theo sau lệnh JGE/JNL. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Lệnh này không tác động đến các cờ :

Ví dụ : Nếu nội dung thanh AL lớn hơn hoặc bằng 10 H thì nhảy đến nhãn THOI :

```
CMP  AL, 10H   ; so sánh  AL  với 10H
JGE  THOI     ; nhảy đến THOI  nếu AL   lớn    hơn   hoặc
                ; bằng   10H.
```

JL/JNGE - Jump if Less than/Jump if Not Greater than or Equal (**Nhảy nếu bé/Nhảy nếu không lớn hơn hoặc bằng)**

Viết lệnh : JG NHAN

JNGE NHAN

Mô tả : IP ← IP + Dịchchuyển.

Hai lệnh trên biểu diễn cùng một thao tác : nhảy (có điều kiện) tới NHAN nếu (SF⊕OF)=1. Quan hệ "lớn hơn" (greater than) và "bé hơn " (less than) là các quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) của 2 số có dấu. Lớn hơn có nghĩa là dương hơn. Nhãn NHAN phải nằm cách xa (dịch đi một khoảng) - 128 .. + 127 byte so với lệnh tiếp theo sau lệnh JL/JNGE. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Lệnh này không tác động đến các cờ :

Ví dụ : Nếu nội dung thanh AL nhỏ hơn 10 H thì nhảy đến nhãn THOI :

```
CMP  AL, 10H   ; so sánh  AL  với 10H
JL  THOI       ; nhảy đến THOI  nếu AL   nhỏ   hơn 10H.
```

JLE/JNG - Jump if Less than or Equal/Jump if Not Greater **than (Nhảy nếu bé hơn hoặc bằng/Nhảy nếu không lớn hơn)**

Viết lệnh : JLE NHAN

JNG NHAN

Mô tả : IP ← IP + Dịchchuyển.

Hai lệnh trên biểu diễn cùng một thao tác : nhảy (có điều kiện) tới NHAN nếu (SF⊕OF)+2Z=1. Quan hệ "lớn hơn" (greater than) và "bé hơn " (less than) là các quan hệ dành cho việc so sánh (do lệnh CMP thực hiện) của 2 số có dấu. Lớn hơn có nghĩa là dương hơn. Nhãn NHAN phải nằm cách xa (dịch đi một khoảng) - 128 .. + 127 byte so với lệnh tiếp theo sau lệnh JLE/JNG. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển.

Lệnh này không tác động đến các cờ :

Ví dụ : Nếu nội dung thanh AL không lớn hơn 10H thì nhảy đến nhãn THOI :

```
CMP  AL, 10H   ; so sánh  AL  với 10H
JL  THOI       ; nhảy đến THOI  nếu AL   không lớn  hơn
               ; 10H.
```

JMP - Unconditinal Jump to specified Destination (Nhảy (vô điều kiện) đến một đích nào đó).

Lệnh này khiến cho bộ vi xử lý 8088 bắt đầu thực hiện một lệnh mới tại địa chỉ được mô tả trong lệnh. Lệnh này có các chế độ địa chỉ giống như lệnh Call và nó cũng phân biệt nhảy xa và nhảy gần. Tuỳ thuộc vào độ dài của bước nhảy chúnh ta phân biệt 5 kiểu lệnh nhảy khác nhau : 3 kiểu nhảy gần và 2 kiểu nhảy xa với độ dài lệnh khác nhau (hình 3.8). Mỗi ô trên các lệnh tương ứng một byte dùng để ghi lệnh. Như vậy lệnh nhảy có độ dài từ 2 đến 5 byte.

Viết lệnh : sau đây là các dạng lệnh nhảy không điều kiện :

JMP NHAN

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

+ nhảy ngắn (short jump)

Trong trường hợp này nhãn NHAN phải nằm cách xa (dịch đi một khoảng nhiều nhất là -128 .. + 127 byte so với lệnh tiếp theosau lệnh JMP. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển mở rộng dấu cho nó. Sau đó

IP ← IP + Dịchchuyển

Đây là lệnh nhảy trực tiếp vì dịch chuyển được 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 (near jump) ứng với trường hợp c) hình 3.8

Trong trường hợp này nhãn NHAN phải nằm cách xa (dịch đi một khoảng nhiều nhất là -32768 .. + 32767 byte so với lệnh tiếp theosau lệnh JMP. Chương trình dịch sẽ căn cứ vào vị trí NHAN để xác định giá trị dịch chuyển mở rộng dấu cho nó. Sau đó

IP ← IP + Dịchchuyển

Đây là lệnh nhảy trực tiếp vì dịch chuyển được 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 (far jump) ứng với trường hợp d) hình 3.8.

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

Nhảy ngắn :

IP ← IP của NHAN

CS ← CS của NHAN

Đây là lệnh nhảy trực tiếp vì địa chỉ nhảy đến được để trực tiếp trong mã lệnh.

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

JMP FAR NHAN

NHAN trong trường hợp này phải được khai là

NHAN LABEL FAR

JMP BX

Đây là lệnh nhảy gần ứng với trường hợp b) hình 3.8, trước đó BX phải chứa địa chỉ lệch của lệnh định nhảy đến trong đoạn CS. Khi thực hiện lệnh này :

IP ← BX

Đây cũng là lệnh nhảy gián tiếp vì địa chỉ lệch nằm trong thanh ghi.

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

JMP NEAR PTR BX

JMP [BX]

Đây là lệnh nhảy gần ứng với trường hợp e) hình 3.8, 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 cũng là lệnh nhảy gián tiếp vì địa chỉ lệch nằm trong ô nhớ.

Để định hướng cho chương trình dịch làm việc 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 ứng với trường hợp e) hình 3.8. Địa chỉ nhảy đến ứng với CS:IP. Giá trị gán choIP và CS được chứa trong 4 ô nhớ do BX và BX+1 (cho IP) và BX+2 và BX+3 (cho CS) chỉ ra trong đoạn DS(SI,DI có thể dùng thay chỗ của BX).

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

Lệnh này không tác động đến các cờ.

JNA - Xem JBE

JNAE - Xem JB

JNB - Xem JAE

JNBE - Xem JA

JNC - Xem JAE

JNE/JNZ - jump if Not Equal/jump if Not Zero (nhảy nếu không bằng nhau/Nhảy nếu kết quả không rỗng)

Viết lệnh : JNE NHAN

JNZ NHAN

Mô tả : IP ← IP + Dịchchuyển.

Hai lệnh trên biểu diễn cùng một thao tác : nhảy (có điều kiện) tới NHAN nếu ZF=0. Nhãn NHAN 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 giá trị dịch chuyển.

Lệnh này không tác động đến các cờ.

Ví dụ : Nếu nội dung thanh AL khác 10H thì nhảy đến nhãn THOI :

```
CMP  AL, 10H   ; so sánh  AL  với 10H
JNE  THOI      ; nhảy đến THOI  nếu AL   khác  16.
```

JNG - Xem JLE

JNGE - Xem JL

JNL - Xem JGE

JNLE - Xem JG

JNO - Jump if No Overflow (nhảy nếu không tràn)

Viết lệnh : JNO NHAN

Mô tả : IP ← IP + Dịchchuyển.

Đây là lệnh nhảy (có điều kiện) tới NHAN nếu OF=0, tức không sảy ra sau khi thực hiện các phép toán với các số có dấu. Nhãn NHAN 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 giá trị dịch chuyển.

Lệnh này không tác động đến các cờ.

Ví dụ : Nếu sau phép cộng mà không có tràn thì nhảy đến nhãn THOI :

```
ADD  AL, AH    ; tính tổng 2 số có dấu trong AL và AH
JNO  THOI      ; nhảy đến THOI  nếu không tràn.
```

JNP/JPO - Jump if No Parity/Jump if Parity Odd (nhảy nếu Parity lẻ)

Viết lệnh : JNO NHAN

JPO NHAN

Mô tả : IP ← IP + Dịchchuyển.

Hai lệnh trên biểu diễn cùng một thao tác : nhảy (có điều kiện) tới NHAN nếu PF=0. Nhãn 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 giá trị dịch chuyển.

Lệnh này không tác động đến các cờ.

Ví dụ : Nếu ta truyền đi một ký tự với parity chẵn mà khi nhận lại được ký tự parity lẻ thì nhảy đến nhãn THOI :

```
IN  AL, 99H    ; đọc ký tự từ cổng,
OR AL,AL       ; tạo cờ,
JNP THOI       ; nhảy đến THOI nếu parity lẻ.
```

JNS - Jump if Not Signed (Jump if Positive) (nhảy nếu kết quả dương)

Viết lệnh : JNS NHAN

Mô tả : IP ← IP + Dịchchuyển.

Đây là lệnh nhảy (có điều kiện) tới NHAN nếu SF=0, tức kết quả là dương sau khi thực hiện các phép toán với các số có dấu. Nhãn 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 giá trị dịch chuyển.

Lệnh này không tác động đến các cờ.

Ví dụ : Nếu sau phép cộng mà kết quả dương thì nhảy đến nhãn THOI :

```
ADD  AL, AH    ; tính tổng 2 số có dấu trong AL và AH
JNS THOI       ; nhảy đến THOI  nếu kết quả dương.
```

JNZ - Xem JNE

JO - Jump if Overflow (nhảy nếu tràn)

Viết lệnh : JO NHAN

Mô tả : IP ← IP + Dịchchuyển.

Đây là lệnh nhảy (có điều kiện) tới NHAN nếu OF=1, tức sảy ra tràn sau khi thực hiện các phép toán với các số có dấu. Nhãn 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 giá trị dịch chuyển.

Lệnh này không tác động đến các cờ.

Ví dụ : Nếu sau phép cộng mà có tràn thì nhảy đến nhãn THOI :

```
ADD  AL, AH    ; tính tổng 2 số có dấu trong AL và AH
JO THOI        ; nhảy đến THOI  nếu có tràn.
```

JP/JPE - Jump if Parity/jump if Parity Even (nhảy nếu parity chẵn)

Viết lệnh : JP NHAN

JPE NHAN

Mô tả : IP ← IP + Dịchchuyển.

Hai lệnh trên biểu diễn cùng một thao tác : nhảy (có điều kiện) tới NHAN nếu PF=1. Nhãn 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 giá trị dịch chuyển.

Lệnh này không tác động đến các cờ.

Ví dụ : Nếu ta truyền đi một ký tự với parity lẻ mà khi nhận lại được một ký tự parity chẵn thì nhảy đến nhãn THOI :

```
IN  AL, 99H    ; đọc ký tự từ cổng,
OR AL,AL       ; tạo cờ,
JP THOI        ; nhảy đến THOI nếu parity chẵn.
```

JPE - Xem JP

JPO - Xem JNP

JS - Jump if Signed (Jump if Negative) (nhảy nếu kết quả âm)

Viết lệnh : JS NHAN

Mô tả : IP ← IP + Dịchchuyển.

Đây là lệnh nhảy (có điều kiện) tới NHAN nếu SF=1, tức kết quả là âm sau khi thực hiện các phép toán với các số có dấu. Nhãn 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 giá trị dịch chuyển.

Lệnh này không tác động đến các cờ.

Ví dụ : Nếu sau phép cộng mà kết quả âm thì nhảy đến nhãn THOI :

```
ADD  AL, AH    ; tính tổng 2 số có dấu trong AL và AH
JS THOI        ; nhảy đến THOI nếu kết quả âm.
```

JZ - Xem JE

LAHF - Load AH with the low byte of the Flag register (Nạp byte thấp của thanh cờ vào AH)

Mô tả : $AH \leftarrow FR_L$

Dùng lệnh này phối hợp với lệnh PUSH AX thì có thể mô phỏng lệnh PUSH PSW của bộ vi xử lý 8085 trên 8088 (lệnh PUSH PSW của vi xủ lý 8085 cất thanh ghi cờ và Acc của nó vào ngăn xếp).

Lệnh này không tác động đến các cờ.

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)

Viết lệnh : LDS Đích,Gốc

Trong đó :

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

+ Gốc là ô nhớ trong đoạn DS được chỉ rõ trong lệnh.

Mô tả : Đích ← Gốc, DS ← Gốc + 2.

Đây là lệnh để 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 sao cho SP và DS chỉ vào địa chỉ đầu của vùng nhớ chứa chuỗi gốc trước khi dùng đến lệnh thao tác chuỗi.

Lệnh này không tác động đến các cờ.

Ví dụ : LDS SI,STR_PTR

Thí dụ trên nạp vào SI nội dung 2 ô nhớ STR_PTR và STR_PTR+1 và nạp vàp DS nội dung 2 ô nhớ STR_PTR+2 và STR_PTR+3. Các ô nhớ này đều nằm trong đoạn dữ liệu DS và chứa địa chỉ của chuỗi gốc. Do vậy sau đó DS:SI chỉ vào đầu chuỗi gốc cần thao tác.

LEA - Load Effective Address (Nạp địa chỉ hiệu dụng vào thanh ghi)

Viết lệnh : LEA Đích,Gốc

Trong đó :

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

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

Mô tả : Đích ← Địa chỉ lệch của Gốc, hoặc

Đích ← Địa chỉ hiệu dụng của Gốc

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

Lệnh này không tác động đến các cờ.

Ví dụ :

```
LEA  DX, MSG        ; nạp địa chỉ lệch của bản tin
                    ; MSG vào DX.
LEA  CX, [BX]  [DI] ; nạp vào CX địa chỉ hiệu dụng
                    ; do BX và DI chỉ ra : EA =BX+DI
```

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)

Viết lệnh : LES Đích,Gốc

Trong đó :

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

+ Gốc là ô nhớ trong đoạn DS được chỉ rõ trong lệnh.

Mô tả : Đích ← Gốc, ES ← Gốc + 2.

Đây là lệnh để 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 sao cho DI và ES chỉ vào địa chỉ đầu của vùng nhớ chứa chuỗi gốc trước khi dùng đến lệnh thao tác chuỗi.

Lệnh này không tác động đến các cờ.

Ví dụ : LES DI,[BX]

Thí dụ trên nạp vào DI nội dung 2 ô nhớ BX và BX+1 và nạp vàp ES nội dung 2 ô nhớ BX+2 và BX+3. Các ô nhớ này đều nằm trong đoạn dữ liệu DS và chứa địa chỉ của chuỗi gốc. Do vậy sau đó ES:DI chỉ vào đầu chuỗi gốc cần thao tác.

LOCK - Assert Bus Lock signal (Đưa ra tín hiệu khoá bus)

Lệnh LOCK dùng đặt trước các lệnh mà khi chạy nó có nguy cơ gây lỗi do khả năng xảy ra tranh chấp trong việc sử dụng bus giữa bộ vi xử lý 8088 và các bộ xử lý khác trong hệ thống đa xử lý. Nếu có lệnh LOCK đặt trước một lệnh nào đó, thì khi chạy lệnh này, 8088 đưa ra tín hiệu khoá bus. Tín hiệu này sẽ nối ra thiết bị điều khiển bus ngoài để cấm các bộ xử lý khác trong hệ thống sử dụng bus.

Lệnh này không tác động đến các cờ.

Ví dụ : LOCK XCHG AL,Kytu

Lệnh XCHG cần 2 lần thâm nhập bus để hoàn tất việc thực hiện lệnh, do đó cần đặt sau LOCK để tránh nguy cơ tranh chấp bus có thể xảy ra trong hệ thống đa xử lý.

LODS/LODSB/LODSW - Load string Byte/Word into AL/AX (Nạp vào AL/AX 1 phần tử của chuoiĩ byte/từ)

Viết lệnh : LODS Chuỗigốc

LODSB

LODSW

Mô tả :

AL ← phầntửhiệnthời, SI ← Si ± 1 tuỳ theo DF, nếu là chuỗi byte.

AL ← phầntửhiệnthời, SI ← Si ± 2 tuỳ theo DF, nếu là chuỗi từ.

(phầntửhiệnthời của chuỗi là do DS:SI hiện thời chỉ ra)

Lệnh LODS nạp vào AL/AX 1 byte/từ (1 phần tử của chuuoĩ đã được định nghĩa trước là chuỗi gồm các byte hoặc từ) do SI chỉ ra trogn đoạn DS, sau đó SI tự động tăng/giảm để chỉ vào phần tử tiếp theo tuỳ theo cờ hướng. Khi phải dịch lệnh LODS Chuỗigốc, chương trình dịch dùng tên Chuỗigốc để xác định xem lúc khai báo thì Chuỗigốc có các phần tử là byte hay từ. Muốn chỉ rõ cho chương trình dịch hợp ngữ rằng ta làm việc với chuỗi các byte hoặc các từ, ta cũng có thể dùng lệnh LODSB hoặc LODSW.

Lệnh này không tác động đến các cờ.

Ví dụ :

```
CLD             ; làm việc với chuỗi theo chiều →
LEA  SI, STRI   ; SI chỉ vào đầu chuỗi STRI để
                ; tại đoạn dữ liệu DS
LODS  STRI      ; nạp vào Acc 1 phần tử.
```

LOOP - Jump to Sspecified Label if CX≠0 after Autodecrement (lặp lại đoạn chương trình do nhãn chỉ ra cho đến khi CX=0)

Viết lệnh : LOOP NHAN

Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ nhãn 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 thanh ghi CX và sau mỗi lần thực hiện lệnh LOOP NHAN thì đồng thời CX tự động giảm đi một (CX ←CX-1).

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

Lệnh này không tác động đến các cờ.

Ví dụ :

```
        XOR  AL, AL        ; xoá AL
        MOV CX,16          ; số lần lặp để tại CX
   LAP: INC  AL            ; tăng AL thêm 1
        LOOP  LAP          ; lặp lại 16 lần, AL = 16
```

LOOPE/LOOPZ - Loop While CX≠0 and ZF=1 (Lặp lại đoạn chương trình do nhãn chỉ ra cho đến khi CX=0 hoặc ZF=0)

Viết lệnh : LOOPE NHAN

LOOPZ NHAN

Mô tả :

Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ nhãn NHAN đến hết lệnh LOOPE NHAN hoặc LOOPZ NHAN) cho đến khi số lần lặp CX=0 hoặc ZF=0. Điều này có nghĩa là trước khi vào vòng lặp ta phải đưa số lần lặp mong muốn vào thanh ghi CX và sau mỗi lần thực hiện lệnh LOOP NHAN thì đồng thời CX tự động giảm đi 1 (CX ←CX-1).

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

Lệnh này không tác động đến các cờ.

Ví dụ :

```
        MOV  AL, AH        ; AL được gán giá trị của AH
        MOV CX,100         ; để đếm số lần lặp
   LAP: INC  AL            ; tăng AL thêm 1
        CMP  AL,16         ; AL=16 ?
        LOOP  LAP          ; lặp lại cho đến khi AL≠16
                           ; hoặc đếm xong.
```

LOOPNE/LOOPNZ - Loop While CX≠0 and ZF=0(lặp lại đoạn chương trình do nhãn chỉ ra cho đến khi CX=0 hoặc ZF=1)

Viết lệnh : LOOPE NHAN

LOOPZ NHAN

Mô tả :

Lệnh này dùng để lặp lại đoạn chương trình (gồm các lệnh nằm trong khoảng từ nhãn NHAN đến hết lệnh LOOPNE NHAN hoặc LOOPNZ NHAN) cho đến khi số lần lặp CX=0 hoặc ZF=1. Điều này có nghĩa là trước khi vào vòng lặp ta phải đưa số lần lặp mong muốn vào thanh ghi CX và sau mỗi lần thực hiện lệnh LOOP NHAN thì đồng thời CX tự động giảm đi 1 (CX ←CX-1).

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

Lệnh này không tác động đến các cờ.

Ví dụ :

```
        MOV  AL, AH          ;  AL được gán giá trị của AH
        MOV CX,100           ;  để đếm số lần lặp
   LAP: INC  AL              ;  tăng AL thêm 1
        CMP  AL,16           ;  AL=16 ?
        LOOP  LAP            ;  lặp lại cho đến khi
                             ;  AL=16 hoặc đếm xong.
```

LOOPNZ - Xem LOOPNE

LOOPZ - Xem LOOPE

MOV - Move a Word or byte (Chuyển 1 từ hay 1 byte)

Viết lệnh : MOV Đích,Gốc.

Mô tả : Đích ← Gốc

Trong đó toán hạng đích và gốc 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 2 ô nhớ hoặc 2 thanh ghi đoạn.

Lệnh này không tác động đến các cờ.

Ví dụ :

```
        MOV  AL, 74H              ;  AL ← 74
        MOV CL,BL           ;  CL ← BL
        MOV  DL,[SI];  DL ← {DS:SI}
        MOV  AL,Table [BX];  AL ←{DS:(Table+BX)}
```

MOVS/MOVSB/MOVSW - Move String Byte or String Word (Chuyển 1 phần tử của 1 chuỗi sang một chuỗi khác)

Viết lệnh : MOVS Chuỗiđích,Chuỗigốc
MOVSB
MOVSW

Mô tả : PhầntửChuỗiđích ← PhầntửChuỗigốc

Lệnh này dùng chuyển từng byte hay từng từ của chuỗi gốc sang chuỗi đích,

trong đó :

+ DS:SI là địa chỉ của phần tử trong chuỗi gốc.

+ ES:DI là địa chỉ của phần tử trong chuỗi đích.

+ Sau mỗi lần chuyển SI ← SI ± 1, DI ← DI ± 1 hoặc SI ← SI ± 2, DI ← DI ± 2 một cách tự động tuỳ thuộc cờ hướng DF là 0/1 và chuỗi là chuỗi byte hoặc chuỗi từ.

Có hai cách để chỉ ra chuỗi là chuỗi byte hay là chuỗi từ. Cách đầu tiên là ta khai rõ bằng tên chuỗi nguồn và chuỗi đích 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 mô tả cách sử dụng tại lệnh COMPS).

Lệnh MOVS/MOVSB/MOVSW có thể dùng kèm với lệnh REPE hoặc REPNE để so sánh tất cả các phần tử trong chuỗi.

Lệnh này không tác động đến các cờ.

Ví dụ :

```
        MOV  DI, OFFSET Chuỗiđích   ;  lấy địa chỉ lệch
                                    ;  của chuỗiđích tại
                                    ;  ES vào DI
```

```
MOV SI, OFFSET Chuỗigốc     ; lấy địa chỉ lệch
                            ; của chuỗigốc tại DS
                            ; vào SI
CLD                         ; làm việc với chuỗi
                            ; theo chiều →
MOVSB                       ; chuyển 1 byte . SI
                            ; và DI tăng thêm 1.
```

MUL - Multiply Unisigned Byte or Word (nhân số không dấu)

Viết lệnh : MUL Gốc

Trong đó toán hạng Gốc là số nhân và có thể tìm được theo các chế độ địa chỉ khác nhau.

Mô tả : tuỳ theo độ dài của toán hạng Gốc ta có 2 trường hợp tổ chức phép nhân, chỗ để ngầm định cho số bị nhân và kết quả :

- nếu Gốc là số 8 bit : AL×Gốc,
  số bị nhân phải là số 8 bit để trong AL.
  sau khi nhân : AX ← tích,
- nếu Gốc là số 16 bit : AX×Gốc,
  số bị nhân phải là số 16 bit để trong AX.
  sau khi nhân : DXAX ← tích.

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

Như vậy các cờ CF và OF sẽ báo cho ta biết có thể bỏ đi bao nhiêu số 0 trong kết quả.

Ví dụ :

Nếu ta cần nhân một số 8 bit với một số 16 bit , ta để số 16 bit tại Gốc và số 8 bit ở AL. Số 8 bit này ở AL cần phải được mở rộng sang AH bằng cách gán AH=0 để làm cho số bị nhân nằm trong AX. Sau cùng chỉ việc dùng lệnh MUL Gốc và kết qur có trong cặp DXAX.

Cập nhật : CF, OF.

Không xác định : AF, PF, SF, ZP.

NEG - Negate a Operand (Form its 2's Complement) (lấy bù hai của một toán hạng, đổi dấu của một toán hạng)

Viết lệnh : NEG Đích

Trong đó toán hạng Đích có thể tìm được theo các chế độ địa chỉ khác nhau.

Mô tả : Đích ← 0 - (Đích)

Điều này hoàn toàn tương đương với việc lấy (Đích + 1) làm kết quả. Nếu ta lấy bù 2 của -128 hoặc -32768 thì ta sẽ được kết quả không đổi nhưng cờ OF=1 để báo là kết quả bị tràn (vì số dương lớn nhất biểu diễn được là + 127 và + 32767).

Cập nhật : AF, CF, OF, PF, SF, ZF.

Ví dụ :

```
NEG  AH                 ; AH ← 0 -(AH)
NEG BYTE PTR [BX]  ; lấy bù hai của ô nhớ đó
        ; BX chỉ ra trong DS
```

NOP - No Operation (CPU không làm gì)

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

Lệnh này không tác động đến các cờ.

NOT - Invert Each Bit of an Operand (Form its 1's complement) (lấy bù của một toán hạng, đảo bit của một toán hạng).

Viết lệnh : NOT Đích

Trong đó toán hạng Đích có thể tìm được theo các chế độ địa chỉ khác nhau.

Mô tả : Đích ← (Đích)

Lệnh này không tác động đến các cờ.

Ví dụ :

```
NOT  AH                ;  AH ← (AH)
NOT BYTE PTR [BX]  ;  lấy bù 1 của ô nhớ đó
        ;  BX chỉ ra trong DS
```

OR - Logically Or Corresponding Bits of Two Operands (hoặc hai toán hạng)

Viết lệnh : OR Đích,Gốc

Mô tả : Đích ← Đích v Gốc

Trong đó toán hạng đích và gốc có thể tìm được theo các địa chỉ khác nhau, nhưng phải chứa dữ liệu cùng độ dài và không được phép đồng thời là 2 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 loogic toán hạng đó với toán hạng tức thời có các bit 1 tại các vị trí tương ứng cần thiết lập.

Xóa : CF,OF.

Cập nhật : PF, SF, ZP,PF. Chỉ có nghĩa khi toán hạng là 8 bit.

Không xác định : AF.

Ví dụ :

```
OR  AL, BL              ;  AL ← AL v BL theo từng bit
OR  BL, 30H  ;  lập bit b4 và b5 của BL lên 1.
```

OUT - Output a Byte or a Word toa Port (đưa dữ liệu từ Acc ra cổng)

Viết lệnh : OUT Port,Acc

Mô tả : Acc → {port}

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

+ 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 cổng port +1.

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

OUT DX, Acc

Trong đó DX phải được gán từ trước giá trị ứng với địa chỉ cổng.

Lệnh này không tác động đến các cờ.

POP - Pop Word from Top of Stack (lấy lại 1 từ vào thanh ghi từ đỉnh ngăn xếp)

Viết lệnh : POP Đích

Mô tả : Đích ← {SP},

SP ← SP +2.

Trong đó toán hạng đích có thể tìm được theo các chế độ địa chỉ khác nhau : có thể là các thanh ghi đa năng, thanh ghi đoạn (nhưng không được là thanh ghi đoạn mã CS) hoặc ô nhớ. Dữ liệu để tại ngăn xếp không thay đổi. Giá trị của SS không thảy đổi.

Lệnh này không tác động đến các cờ.

Ví dụ :

```
POP  DX  ;  lấy 2 byte từ đỉnh ngăn xếp
         ;  đưa vào DX.
POP Table[BX]      ;  lấy 2 byte từ đỉnh ngăn xếp rồi
         ;  để tại vùng DX có địa chỉ
         ;  đầu tại (Table + BX)
```

POPF - Pop Word from Top of Stack to flag Register (lấy một từ từ đỉnh ngăn xếp rồi đưa vào thanh cờ)

Viết lệnh : POPF

Mô tả : RF ← {SP},

SP ← SP +2.

Sau lệnh này dữ liệu để tại ngăn xếp không thay đổi. SS không thay đổi.

Lệnh này không tác động đến các cờ.

PUSH - Push Word on the Stack (cất 1 từ vào ngăn xếp)

Viết lệnh : POPF Gốc

Mô tả : SP ← SP - 2.

Gốc ← {SP}.

Trong đó toán hạng gốc có thể tìm được theo các chế độ địa chỉ khác nhau : có thể là các thanh ghi đa năng, thanh ghi đoạn hoặc ô nhớ. Lệnh này thường dùng với lệnh POP như là một cặp đối ngẫu để xử lý các dữ liệu và trạng thái của chương trình chính (CTC) khi vào/ra chương trình con (ctc).

Lệnh này không tác động đến các cờ.

Ví dụ :

```
PUSH BX            ;  cất BX vào  ngăn xếp tại vị trí
         Trisdo SP chỉ ra.
PUSH Table[BX]     ;  cất 2 byte của vùng dữ liệu DS
         ;  có địa chỉ đầu tại (Table + BX).
```

PUSHF - Push Flag register to the Stack (cất thanh cờ vào ngăn xếp)

Viết lệnh : POPF Gốc

Mô tả : SP ← SP - 2.

RF ← {SP}.

Dữ liệu để tại ngăn xếp không thay đổi. SS không thay đổi.

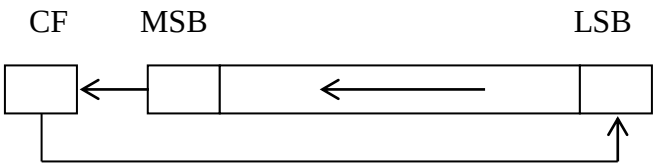
Lệnh này không tác động đến các cờ.

RCL - Rotate through CF to the left (Quay trái thông qua cờ nhớ)

Viết lệnh : RCL Đích,CL

Trong đó toán hạng đích có thể tìm được theo các chế độ địa chỉ khác nhau.

Mô tả :



Lệnh RLC

Lệnh này dùng để quay toán hạng sang trái thông qua cờ CF. CL phải được chứa sẵn số lần quay mong muốn. Trong trường hợp quay một lần có thể viết trực tiếp :

RCL Đích,1

(từ các bộ vi xử lý thế hệ sau như 80186, 80286 ... thì có thể viết trực tiếp kiểu này với số lần quay lớn nhất là 32). Ta nhận thấy số lần của quay là 9 thì kết quả không thay đổi vì cặp CF và toán hạng quay tròn đúng một vòng.

Tác động vào cờ : chỉ có CF và OF bị ảnh hưởng.

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

Ví dụ :

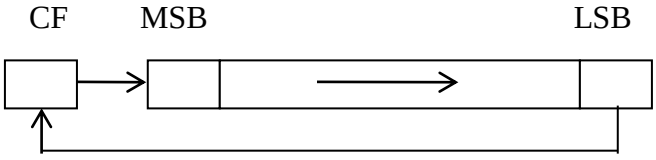
```
RCL  BX,1          ; quay trái thanh ghi BX thông
                   ; qua CF
MOV  CL,8          ; số lần quay để trong CL,
RCL  AL, CL        ; quay trái thanh ghi AL 8 lần
                   ; thông qua CF. MSB lúc này chứa
                   ; giá trị CF ban đầu.
```

RCR - Rotate though CF to the right (quay phải thông qua cờ nhớ)

Viết lệnh : RCR Đích,CL

Trong đó toán hạng đích có thể tìm được theo các chế độ địa chỉ khác nhau.

Mô tả :



Lệnh RCR.

Lệnh này dùng để quay toán hạng sang trái thông qua cờ CF. CL phải được chứa sẵn số lần quay mong muốn. Trong trường hợp quay một lần có thể viết trực tiếp :

RCR Đích,1

(từ các bộ vi xử lý thế hệ sau như 80186, 80286 ... thì có thể viết trực tiếp kiểu này với số lần quay lớn nhất là 32). Ta nhận thấy số lần của quay là 9 thì kết quả không thay đổi vì cặp CF và toán hạng - thanh ghi quay tròn đúng một vòng.

Tác động vào cờ : chỉ có CF và OF bị ảnh hưởng.

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

Ví dụ :

```
RCR  BX,1          ;  quay phải thanh ghi BX thông qua CF
MOV  CL,8    ;  số lần quay để trong CL,
RCR  AL, CL  ;  quay phải thanh ghi AL 8 lần thông
        ;  qua CF. LSB lúc này chứa giá trị
        ;  CF ban đầu.
```

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

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

Ví dụ :

```
REP  MOVSB         ;  lặp lại lệnh chuyển byte của
        ;  chuỗi tới khi CX=0
```

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

Đây là tiếp đầu ngữ dùng để viết trước các lệnh thao tác với chuỗi dữ liệu mà ta muốn lặp lại một số lần. Số lần lặp phải để trước trong CX. Khi các lệnh này được lặp lại thì CX tự động giảm đi một sau mỗi lần lặp. Khi dùng REPE/REPZ với lệnh so sánh chuỗi, quá trình sẽ kết thúc khi đếm hết (CX=0) hoặc khi hai phần tử so sánh khác nhau (ZF=0).

Ví dụ :

```
REPE  CMPSB              ;  lặp lại lệnh so sánh các
        ;  byte của 2 chuỗi tới khi CX=0
        ;  hoặc ZF=0.
```

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 dữ liệu mà ta muốn lặp lại một số lần. Số lần lặp phải để trước trong CX. Khi các lệnh này được lặp lại thì CX tự động giảm đi một sau mỗi lần lặp. Khi dùng REPNE/REPNZ với lệnh quét chuỗi, quá trình sẽ kết thúc khi đếm hết (CX=0) hoặc khi Acc bằng phần tử của chuỗi (ZF=1).

Ví dụ :

```
REPNE  CMPSB       ;  lặp lại lệnh quét các byte của chuỗi
        ;  tới khi hết chuỗi (CX=0)
        ;  hoặc Al bằng 1 phần tử của
        ;  chuỗi (ZF=1).
```

RET - Return reom Procedure to Calling Program (Trở về CTC từ ctc)

Viết lệnh : RET hoặc RET n,n là số nguyên dương.

Mô tả :

RET được đặt tại cuối ctc để bộ vi xử lý lấy lại địa chỉ trở về (địa chỉ của lệnh tiếp theo lệnh gọi của CTC), nó được tự động cất ở ngăn xếp khi có lệnh gọi ctc. Tuỳ theo loại gần hay xa ta cũng sẽ các xử lý khác nhau đối với địa chỉ trở về (xem thêm phần mô ta chung với lệnh Call).

Đặc biệt nếu ta dùng RET n thì sau khi đã lấy lại được địa chỉ trở về (chỉ có IP hoặc có CS và IP) thì SP ← SP + n (dùng để nhảy qua mà không lấy lại các thông số khác của chương trình còn lại trong ngăn xếp).

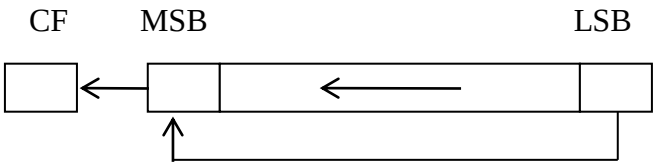
Lệnh này không tác động đến các cờ.

ROL - Rotate All Bits to the Left (Quay vòng sang trái)

Viết lệnh : RCR Đích,CL

Trong đó toán hạng đích có thể tìm được theo các chế độ địa chỉ khác nhau.

Mô tả :



Lệnh ROL.

Lệnh này dùng để quay vòng toán hạng sang trái, MSB sẽ được đưa qua cờ CF và LSB. CL phải được chứa sẵn số lần quay mong muốn. Trong trường hợp quay một lần có thể viết trực tiếp :

ROL Đích,1

(từ các bộ vi xử lý thế hệ sau như 80186, 80286 ... thì có thể viết trực tiếp kiểu này với số lần quay lớn nhất là 32). Ta nhận thấy nếu CL=8 và toán hạng để quay là 8 bit thì kết quả không bị thay đổi vì toán hạng quay tròn đúng một vòng, còn nếu CL=4 thì 2 nibble của toán hạng bị đổi chỗ.

Tác động vào cờ : chỉ có CF và OF bị ảnh hưởng.

Sau lệnh ROL cờ CF mang giá trị cũ của MSB, còn cờ OF ← 1 nếu sau khi quay một lần mà bit MSB bị thay đổi so với trước khi quay. Cờ CF từ giá trị của MSB làm điều kiện cho các lệnh nhảy có điều kiện.

Ví dụ :

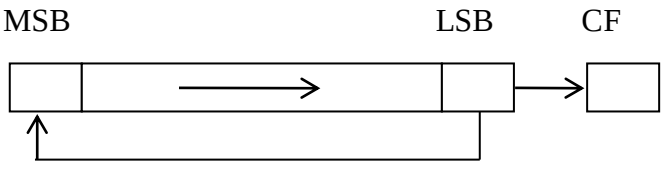
```
ROL  BX,1          ; quay vòng sang trái thanh ghi BX
MOV  CL,8          ; số lần quay để trong CL,
ROL  AL, CL        ; quay vòng qua trái thanh ghi AL 8
                   ; lần thanh ghi AL lúc này chứa giá
                   ; trị ban đầu (trước khi quay).
```

ROR - Rotate All Bits to the right (Quay vòng sang phải)

Viết lệnh : ROR Đích,CL

Trong đó toán hạng đích có thể tìm được theo các chế độ địa chỉ khác nhau.

Mô tả :



Lệnh ROR.

Lệnh này dùng để quay vòng toán hạng sang phải, LSB sẽ được đưa qua cờ CF và MSB. CL phải được chứa sẵn số lần quay mong muốn. Trong trường hợp quay một lần có thể viết trực tiếp :

ROR Đích,1

(từ các bộ vi xử lý thế hệ sau như 80186, 80286 ... thì có thể viết trực tiếp kiểu này với số lần quay lớn nhất là 32). Ta nhận thấy nếu CL=8 thì kết quả không bị thay đổi vì toán hạng quay tròn đúng một vòng, còn nếu CL=4 thì 2 nibble của nó bị đổi chỗ.

Tác động vào cờ : chỉ có CF và OF bị ảnh hưởng.

Sau lệnh ROR cờ CF mang giá trị cũ của LSB, còn cờ OF ← 1 nếu sau khi quay một 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 các lệnh nhảy có điều kiện.

Ví dụ :

```
ROR  BX,1          ;  quay vòng qua phải thanh ghi BX.
MOV  CL,8    ;  số lần quay để trong CL,
ROR  AL, CL ;  quay vòng qua phải thanh ghi AL 8
       ;  lần, thanh ghi AL lúc này chứa giá
       ;  trị ban đầu (trước khi quay).
```

SAHF - Store AH Register into byte of Flag Register (cất thanh ghi AH vào byte thấp của thanh cờ)

Mô tả : $FR_L \leftarrow AH$

Dùng lệnh này phối hợp với lệnh POP AX thì có thể mô phỏng lệnh POP PSW của bộ vi xử lý 8085 trên 8088.

Cập nhật : AF, CF, OF, PF, SF, ZP.

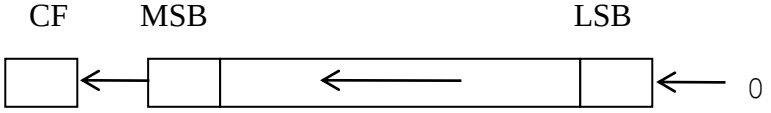
SAL - Shift Arithmetically Left (dịch trái số học)/SHL - Shift (Logically) Left (dịch trái logic).

Viết lệnh : SAL Đích,CL

SHL Đích,CL

Trong đó toán hạng đích có thể tìm được theo các chế độ địa chỉ khác nhau.

Mô tả :



Lệnh SAL/SHL.

Hai lệnh này có cùng tác dụng dịch trái số học toán hạng (còn gọi là dịch trái logic để có lệnh đối ngẫu với lệnh dịch phải logic sẽ nói ở phần sau). Mỗi lần dịch MSB sẽ được đưa qua cờ CF và 0 đưa vào LSB. Thao tác kiểu này được gọi là dịch logic. CL phải được chứa sẵn số lần quay mong muốn. Trong trường hợp quay một lần có thể viết trực tiếp :

SAL Đích,1

(từ các bộ vi xử lý thế hệ sau như 80186, 80286 ... thì có thể viết trực tiếp kiểu này với số lần quay lớn nhất là 32). Ta nhận thấy một lần dịch trái kiểu này tương đương với một lần làm phép nhân với 2 của số không dấu. Vì vậy ta có thể làm phép nhân một số với số nhân không dấu tương đương với 2 bằng cách dịch trái số học số bị nhân i lần. Chính vì vậy thao tác này còn được gọi là dịch trái số học. Trong chừng mực nhất định lần này chạy nhanh hoen MUL.

Tác động vào cờ :

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

Cập nhật : SF, ZF, PF.PF chỉ có ý nghĩa khi kết quả là 8 bit.

Không xác định : AF.

Ví dụ :

```
SAL  BX,1              ;  dịch trái số học thanh ghi BX
MOV  CL,4     ;  số lần dịch để trong CL,
SLH  AL, CL   ;  dịch trái thanh ghi AL 4 lần,
        ;  thanh ghi AL lúc này có giá
        ;  trị bằng 1/16 thanh ghi AL ban đầu.
```

SBB - Substract with Brrow (trừ có mượn)

Viết lệnh : SBB Đích,Gốc

Mô tả : Đích ← Đích - Gốc - CF

Trong đó toán hạng đích và gốc có thể tìm được theo các chế độ địa chỉ khác nhau, nhưng phải chứa cùng một loại dữ liệu và không được phép đồng thời là 2 ô nhớ và cũng không được là thanh ghi đoạn.

Cập nhật : AF, CF, OF, PF, SF, ZP (AF và PF chỉ liên quan đến 8bit thấp).

Ví dụ : Các ví dụ sau đây có thể đại diện cho các chế độ địa chỉ có thể có trong lệnh trừ này cũng như một số các lệnh khác với ngữ pháp tương tự.

```
SBB   AL,74H;  AL ← AL - 47 - CF
SBB   CL,BL   ;  CL ← CL - BL - CF
SBB   DL,[SI] ;  DL ← DL - {DS:SI} - CF
SBB   AL,Table[BX]  ;  AL ← AL - {DS:(Table+BX)} - CF
```

SCAS/SCASB/CASW - Scan a String Byte or a String Word (Quét chuỗi byte hay chuỗi từ)

Viết lệnh : SCAS Chuỗiđích

SCASB

SCASW

Mô tả :

AL : Phầntửchuỗiđích, DI ← DI ± 1 tuỳ theo DF, nếu là chuỗi Byte,

AL : Phầntửchuỗiđích, DI ← DI ± 2 tuỳ theo DF, nếu là chuỗi từ.

(Phầntửchuỗiđích là do ES:DI hiện thời chỉ ra)

Lệnh SCAS so sánh AL hoặc AX với từng byte hay từng từ của xâu đích chỉ để tạo các cờ, không lưu kết quả so sánh, các toán hạng không bị thay đổi. Sau đó DI tự động tăng /giảm để chỉ vào phần tử tiếp theo tuỳ theo cờ hướng. Khi dịch lên SCAS Chuỗiđích, chương trình dịch dùng tên Chuỗiđích để xác định xem lúc khai báo thì Chuỗiđích có các phần tử là byte hay từ. Muốn chỉ rõ cho chương trình dịch hợp ngữ rằng ta làm việc với chuỗi byte (chứa các byte) hoặc chuỗi từ (chứa các từ), ta cũng có thể dùng lệnh SCASB hoặc SCASW. Các lệnh này còn thường dùng kèm với REPNE/REPNZ để lặp lại việc quét một số lần hoặc quét cho tới khi tìm được mẫu cần tìm.

Cập nhật : AF, CF.OF, PF, SF, ZP.

Ví dụ :

```
        CLD  BX,1           ; lamf việc với chuỗi theo chiều →
        MOV  AL,13  ; AL chứa mã ASCII của ký tự về
                    ; đầu dòng (CR),
        MOV  CX,80  ; dịch quét một dòng 80 ký tự,
        LEA  DI, STR1       ; DI chỉ vào đầu chuỗi STR1 để
                    ; tại đoạn dữ liệu ES
REPNE   SCASB  STR1         ; so AL với một phần tử
                    ; của chuỗi để tìm CR.
```

SHL - Xem SAL

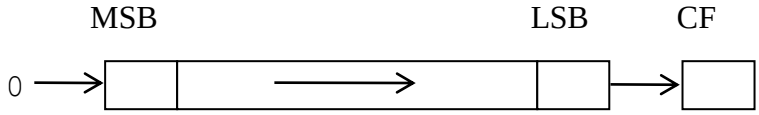
SHR - Shift (Logically) Right (dịch phải logic)

Viết lệnh : SHL Đích,CL

SHL Đích,CL

Trong đó toán hạng đích có thể tìm được theo các chế độ địa chỉ khác nhau.

Mô tả :



Lệnh SHR.

Lệnh này dùng để dịch phải logíc toán hạng. Sau mỗi lần dịch LSB sẽ được đưa qua cờ CF và 0 đưa vào MSB. Thao tác kiểu này được gọi là dịc logíc. CL phải được chứa sẵn số lần quay mong muốn. Trong trường hợp quay một lần có thể viết trực tiếp :

SHR Đích,1

(từ các bộ vi xử lý thế hệ sau như 80186, 80286 ... thì có thể viết trực tiếp kiểu này với số lần quay lớn nhất là 32). Ta nhận thấy một lần dịch trái kiểu này tương đương với một lần làm phép chia cho 2 của số không dấu. Vì vậy ta có thể làm phép chia một số không dấu cho một số không dấu tương đương với 2^i bằng cách dịch phải logíc số bị chia i lần.

Tác động vào cờ :

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

Cập nhật : SF, ZF, PF.PF chỉ có ý nghĩa khi kết quả là 8 bit.

Không xác định : AF.

Ví dụ :

```
        SHR  BX,1           ; dịch phải lôgíc thanh ghi BX.
        MOV  CL,4   ; số lần dịch để trong CL,
        SHR  AL, CL ; dịch phải thanh ghi AL 4 lần,
                    ; thanh ghi AL lúc này có 4 bit
                    ; thấp bằng 4 bit cao ban đầu và
                    ; 4 bit cao là 0.
```

STC - Set the Carry Flag (lập cờ nhớ)

Mô tả : CF ← 1.

Không tác động đến các cờ khác.

STD - Set the Direction Flag (Lập cờ hướng)

Mô tả : DF ← 1.

Lệnh này định hướng thao tác cho các lệnh làm việc với chuỗi theo chiều lùi (←). Các thanh ghi SI và DI liên quan sẽ được tự động giảm khi làm việc xong với một phần tử của chuỗi.

Không tác động đến các cờ khác.

SET - Set the Interrupt Flasg (lập cờ cho phép ngắt)

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 8088 sẽ bị ngắt, nó sẽ tự động ngắt thanh ghi cờ và địa chỉ trở về vào ngăn xếp rồi chuyển sang chạy chương trình (con) phục vụ ngắt CTPVN. Tại chuỗi CTPVN sẽ có lệnh trở về CTC từ CTPVN (IRET) để 8088 lấy lại từ ngăn xếp giá trị của thanh ghi cờ và địa chỉ trở về.

Không tác động đến các cờ khác.

STOS/STOSB/STOSW - Store AL/AX in String Byte/Word (cất AL/AX vào mỗi phần tử của chuỗi byte/từ)

Viết lệnh : STOS Chuỗiđích
STOSB
STOSW

Mô tả :

AL : Phầntửhiệnthời, DI ← DI ± 1 tuỳ theo DF, nếu là chuỗi Byte,

AL : Phầntửhiệnthời, DI ← DI ± 2 tuỳ theo DF, nếu là chuỗi từ.

(Phầntửhiệnthời của chuỗi là do ES:DI hiện thời chỉ ra)

Lệnh STOS cất AL/AX vào 1 byte/từ (1 phần tử của chuỗi đã được định nghĩa trước là chuỗi gồm các byte hoặc từ. Do DI chỉ ra trong đoạn ES, sau đó DI tự động tăng /giảm để chỉ vào phần tử tiếp theo tuỳ theo cờ hướng. Khi dịch lên STOS Chuỗiđích, chương trình dịch dùng tên Chuỗiđích để xác định xem lúc khai báo thì Chuỗiđích có các phần tử là byte hay từ. Muốn chi rõ cho chương trình dịch hợp ngữ rằng ta làm việc với chuỗi các byte hoặc các từ ta cũng có thể dùng lệnh STOSB hoặc STOSW.

Lệnh này không tác động đến các cờ.

Ví dụ :

```
MOV   DI,OFFSET STR1        ;  DI chỉ vào đầu chuỗi
MOV  CL,4                   ;  STR1 để tại đoạn dữ liệu  phụ ES,
STOS  STR1                  ;  Cất Acc vào 1 phần tử của chuỗi
```

Trong thí dụ trên chương trình dịch hợp ngữ sẽ dùng tên STR1 để xác định kiểu của STR1 là byte hay là từ rồi tự động cất AL hay AX vào chuỗi.

```
MOV   DI,OFFSET STR1        ;  DI chỉ vào đầu chuỗi
MOV  CL,4                   ;  STR1 để tại đoạn
                            ;  dữ liệu phụ ES,
STOS  STR1                  ;  Cất Acc vào 1 phần tử của chuỗi
```

Trong thí dụ thứ 2 chương trình dịch hợp ngữ sẽ hiểu là chữ "B" trong lệnh STOSB xác định kiểu của STR1 là byte và tự động cất AL vào chuỗi.

SUB - Substract (trừ hau toán hạng)

Viết lệnh : SUB Đích,Gốc.

Mô tả : Đích ← Đích - Gốc

Trong đó toán hạng đích và gốc có thể tìm được theo các chế độ địa chỉ khác nhau, nhưng phải chứa dữ liệu có cùng độ dài và không được phép đồng thời là 2 ô nhớ và cũng không được là thanh ghi đoạn. Có thể tham khảo các ví dụ của lệnh SBB.

Cập nhật : AF, CF, OF, PF, SF, ZP (AF và PF chỉ liên quan đến 8 bit thấp).

TEST - And Operands to Update Flag (và 2 toán hạng để tạo cờ)

Viết lệnh : TEST Đích,Gốc.

Mô tả : Đích $\wedge$ Gốc

Trong đó toán hạng đích và gốc có thể tìm được theo các chế độ địa chỉ khác nhau, nhưng phải chứa dữ liệu có cùng độ dài và không được phép đồng thời là 2 ô nhớ và cũng không được là thanh ghi đoạn. Sau lệnh này các toán hạng không bị thay đổi và kết quả không được lưu giữ. Các cờ được tạo ra sẽ được dùng làm điều kiện cho các lệnh nhảy có điều kiện. Lệnh này cũng có tác dụng che như một mặt nạ.

Xoá : CF, OF.

Cập nhật : PF, SF, ZP (PF chỉ liên quan đến 8 bit thấp).

Không xác định : AF.

Ví dụ :

```
TEST  AH,AL          ; AH ∧ AL để tạo cờ
TEST  AH,01H         ; bit 0 của AH bằng 0
TEST  BP, [BX] [DI]  ; và BP với ô nhớ do
                     ; BX + DI chỉ ra trong
                     ; vùng dữ liệu DS
```

WAIT - Wait for TEST or INTR Signal (chờ tín hiệu từ chân TEXT hoặc INTR)

Mô tả :

Lệnh này đưa bộ xử lý 8088 vào trạng thái nghĩ và nó sẽ mang lại trạng thái này cho tới khi có tín hiệu ở mức thấp tác động vào chân TEXT hoặc khi có tín hiệu cao tác động vào chân INTR. Nếu có yêu cầu ngắt và yêu cầu này được phép tác động trong khi 8088 đang ở trạng thái nghĩ thì sau khi thực hiện CTPVN nó lại quay lại trạng thái nghĩ. Lệnh này dùng để đồng bộ hoạt động của 8088 và các bộ phận bên ngoài như bộ xử lý toán học 8087.

Lệnh này không tác động đến các cờ.

XCHG - Exchange 2 Operands (tháo nội dung hai toán hạng)

Viết lệnh : XCHG Đích,Gốc

Mô tả : Đích $\leftrightarrow$ Gốc

Trong đó toán hạng đích và gốc có thể tìm được theo các chế độ địa chỉ khác nhau, nhưng phải chứa dữ liệu có cùng độ dài và không được phép đồng thời là 2 ô nhớ và cũng không được là thanh ghi đoạn. Lệnh XCHG toán hạng này chứa nội dung cũ của toán hạng kia và ngược lại.

Lệnh này không tác động đến các cờ.

Ví dụ :

```
XCHG  AH,AL          ; tráo nội dung AH và AL
XCHG  AX,BX          ; tráo nội dung của AX và BX
```

```
XCHG  AL, Table[BX]        ;  tráo nội dung AL với ô nhớ có
                           ;  địa chỉ Table [BX] trong vùng
                           ;  dữ liệu DS
```

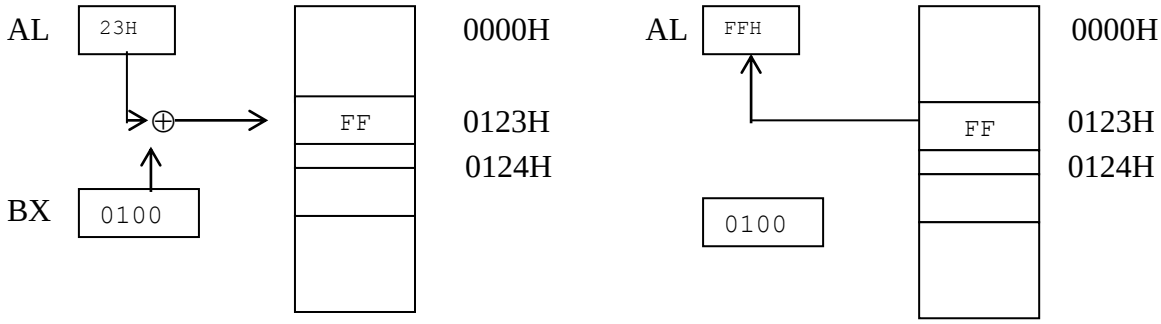
XLAT - Translate a byte in AL (Đổi nội dung AL theo bảng cho trước)

Viết lệnh : XLAT

Mô tả : AL ← {AL + BX}

Lệnh này dùng để 8 bit hoặt ít hơn từ mã này (gốc) sang mã khác (đích) theo một bảng tra cho sẵn trước khi thực hiện lệnh đổi, bảng tra phải chứa các mã đích và BX phải chứa địa chỉ lệnh của bảng này trong đoạn dữ liệu. Còn mã gốc (cần đổi) phải được chứa trong AL. khi chạy lệnh XLAT thì phép cộng AL + BX phải được thực hiện để tạo ra địa chỉ ô nhớ trong bảng so mà nội dung của nó sẽ được cách trở lại vào thanh ghi AL.

Lệnh này không tác động đến các cờ.



XOR - Exclusive Or Corresponding Bits of two Operands (hoặc loại trừ hai toán hạng)

Viết lệnh : XOR Đích,Gốc

Mô tả : Đích ← Đích ⊕ Gốc

Trong đó toán hạng đích và gốc có thể tìm được theo các chế độ địa chỉ khác nhau, nhưng phải chứa dữ liệu có cùng độ dài và không được phép đồng thời là 2 ô nhớ và cũng không được là thanh ghi đoạn. Từ tính chất của phép hoặc loại trừ ta thấy nếu toán hạng đích trùng với toán hạng gốc thì kết quả bằng 0, do đó lệnh này còn được dùng để xoá về 0 một thanh ghi nào đó và kèm theo các cờ CF và OF cũng bị xoá.

Cập nhật : PF, SF, ZP, PF chỉ có nghĩa khi toán hạng là 8 biit.

Không xác định : AF.

Ví dụ :

```
XOR  AL,BL        ;  AL ← AL ⊕  BL theo từng bit một
XOR  BH,BH        ;  xoá BH, xoá CF và OF.
```

Trên đây ta đã giới thiệu qua tấc cả các lệnh có trong tập lệnh của bộ vi xử lý 8086/88. Mặc dù trong thực tế lập trình nhiều khi ta không sử dụng hết được tất cả các lệnh trong tập lệnh này. Tập lệnh này của bộ vi xử lý cao cấp hơn trong họ 80x86 ngoài những lệnh này còn bao gồm những lệnh khác nữa.

Ta có thể coi đây là một ví dụ về một tập lệnh của một bộ vi xử lý dùng để chế tạo ra máy tính với tập lệnh đầy đủ (Complex instruction set computer CISC) với đặc trưng là : có rất nhiều lệnh với các chế độ địa chỉ rất đa dạng, các lệnh có độ dài khác nhau và thời gian thực hiện cũng khác nhau. Bộ vi xử lý vì thế thường có cấu trúc rất phức tạp. Thế mà trên thực tế, trong các chương trình hợp ngữ để giải quyết các công việc cụ thể, thường có nhiều lệnh được dùng với tần suất lớn nhưng cũng có những lệnh rất ít khi hoặc thậm chí không hề

được sử dụng. Đây cũng chính là lý do để các nhà sản xuất cho ra đời các bộ vi xử lý có cấu trúc cãi tiến theo hướng đơn giản hoá với tập lệnh lệnh rất hữu dụng (có số lệnh ít hơn với chế độ địa chỉ đơn giản, các lệnh có độ dài thống nhất và có thời gian thực hiện như nhau) nhưng laijcos khả năng thực hiện lệnh nhanh hơn gấp bội so với loại CISC. Đó là các bộ vi xử lý dùng làm CPU cho các máy tính với tập lệnh rút gọn (Redduced instruction set computer, RISC).

Chương 4 : LẬP TRÌNH BẰNG HỢP NGỮ VỚI 8088

Trong chương trước ta đã giới thiệu khá tỉ mỉ tập lệnh của bộ vi xử lý 8086/88. Trong chương này ta sẽ giới thiệu cách lập trình dùng hợp ngữ trên các máy IBM PC hoặc tương thích với IBM PC (từ nay được gọi chung là IBM PC), vì đó là môi trường phổ thông và tiện lợi nhất để tạo ra và thử nghiệm các chương trình viết bằng hợp ngữ. Nói như vậy là vì a) về phần cứng, máy IBM PC có cấu trúc khá tiêu biểu của một hệ vi xử lý, b) về phần mềm, ta có thể tận dụng các chương trình soạn thảo văn bản hoặc rất nhiều chức năng sẵn có khác của máy IBM PC cho các chương trình của ta thông qua các dịch vụ (các chương trình con phục vụ ngắt) của các ngắt của DOS (Disk Operating System, hệ điều hành) và của BIOS (Basic Inpus Output System, hệ thống vào ra cơ sở). Tuy nhiên, một hệ thống vi xử lý cụ thể có thể có kết cấu khác một máy vi tính IBM PC, do đó khi lập trình cho các hệ thống giả định kiểu như vậy, sẽ có những chương trình mà ta không thể đem thử nghiệm trên máy IBM PC được. Các chương trình này sẽ được đánh dấu cẩn thận bằng dấu /// để ta không đem chúng cho chạy thử trên IBM PC nhằm tránh các hậu quả đáng tiếc có thể xảy ra. Ta sẽ sử dụng chương trình dịch hợp ngữ MASM 5.0 (Macro assemler phiên bản 5.0) của Microsoft với cách định nghĩa đoạn đơn giản và chế độ bộ nhớ nhỏ. Điều này hoàn toàn đủ để đáp ứng các yêu cầu nảy sinh khi ta thực hiện các chương trình đơn giản ban đầu. Ta cũng có thể dùng chương trình dịch hợp ngữ TASM 4.0 (Turbo assembler phiên bản 4.0) của Borland International để thử nghiệm các chương trình hợp ngữ.

4.1. Giới thiệu chung của chương trình hợp ngữ

4.1.1. 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, bởi vì như ta đã biết, để làm việc được với bất kỳ một ngôn ngữ lập trình nào ta cũng cần nắm được cú pháp của nó. Chương trình dưới dạng hợp ngữ mà ta viết ra, nếu đúng về cú pháp, sẽ được chương trình dịch hợp ngữ MASM dịch ra mã máy, từ chương trình mã máy này ta có thể tạo ra các chương trình chạy (thực hiện) được ngay bằng cách dịch tiếp ra các tệp có đuôi EXE hoặc COM. Do vậy khi viết một chương trình hợp ngữ ta phải tuân thủ những quy tắc cú pháp nhất định để chương trình MASM có thể hiểu và dịch được nó.

Một chương trình hợp ngữ bao gồm các dòng lệnh, một dòng lệnh có thể là một lệnh thật dưới dạng ký hiệu (symbolic), mà đôi khi còn được gọi là dạng gợi nhớ (mnemonic) của bộ vi xử lý, hoặc một hướng dẫn cho chương trình dịch (assembler directive). Lệnh gợi nhớ sẽ được dịch ra mã máy còn hướng dẫn cho chương trình dịch thì không được dịch, vì nó chỉ có tác dụng chỉ dẫn riêng thực hiện công việc. Ta có thể viết các dòng lệnh này bằng chữ hoa hoặc chữ thường và chúng sẽ được coi là tương đương vì đối với dòng lệnh chương trình dịch không phân biệt kiểu chữ.

Một dòng lệnh của chương trình hợp ngữ có thể có những trường sau (không nhất thiết phải có đủ hết tất cả các trường):

Tên Mã lệnh Các toán dạng Chú giải

Một ví dụ dòng lệnh gợi nhớ:

```
TIEP :   MOV  AH, {BX} {SI}        ; nạp vào AH nội dung ô
                                   ; nhớ có địa chỉ DS : (BX+SI)
```

Trong ví dụ trên, tại trường tên ta có nhãn TIEP, tại trường mã lệnh ta có lệnh MOV, tại trường toán hạng ta có các thanh ghi AH, BX và SI và phần chú giải gồm có các dòng

```
; nạp vào AH nội dung ô
; nhớ có địa chỉ DS : (BX+SI)
```

Một ví dụ khác là các dòng lệnh với các hướng dẫn cho chương trình dịch:

```
MAIN        PROC
```

và

```
MAIN        ENDP
```

Trong ví dụ này, ở trường tên ta có tên thủ tục là MAIN, ở trường mã lệnh ta có các lệnh giả PROC và ENDP. Đây là các lệnh giả dùng để bắt đầu và kết thúc một thủ tục có tên là MAIN.

- Trường tên

Trường tên chứa các nhãn, tên biến hoặc tên thủ tục. Các tên và nhãn này sẽ được chương trình dịch gán bằng các địa chỉ cụ thể của ô nhớ. Tên và nhẫn có thể có độ dài 1..31 ký tự, không được chứa dấu cách và không được bắt đầu bằng số. Các ký tự đặc biệt khác có thể dùng trong tên là ?.@_$%. Nếu dấu chấm ('.') được dùng thì nó phải được đặt ở vị trí đầu tiên của tên. Nói chung ta cứ đặt các tên bình thường và có ý nghĩa là sẽ ít khi bị sai. Một nhãn thường kết thúc bằng dấu hai chấm (:).

- Trường mã lệnh

Trong trường mã lệnh nói chung sẽ có các lệnh thật hoặc lệnh giả.

Đối với các lệnh thật thì trường này chứa các mã lệnh gợi nhớ. Mã lệnh này sẽ được chương trình dịch dịch ra mã máy.

Đối với các hướng dẫn chương trình dịch thì trường này chứa các lệnh giả và sẽ không được dịch ra mã máy.

- Trường toán hạng

Đối với một lệnh thì trường này chứa các toán hạng của lệnh. Tùy theo từng loại lệnh mà ta có thể có 0,1 hoặc 2 toán hạng trong một lệnh. Trong trường hợp các lệnh với 1 toán hạng thông thường ta có toán hạng là đích hoặc gốc, còn trong trường hợp lệnh với 2 toán hạng thì ta có 1 toán hạng là đích và 1 toán hạng là gốc.

Đối với hướng dẫn chương trình dịch thì trường này chứa các thông tin khác nhau liên quan đến các lệnh giả của hướng dẫn.

- Trường chú giải

Lời giải thích ở trường chú giải phải được bắt đầu bằng dấu chấm phẩy (;) Trường chú giải này được dành riêng cho người lập trình để ghi các lời giải thích cho các lệnh của chương trình với mục đích giúp cho người đọc chương trình dễ hiểu các thao tác của chương trình hơn. Lời chú giải cũng có lợi ngay cho chính tác giả của nó vì sau một thời gian không xem đến chương trình thì mọi việc lại như mới. Khi đọc thấy dấu chấm phẩy, chương trình dịch bỏ qua không dịch từ phần này trở đi. Chính vì vậy người ta cũng thường hay dùng dấu này để loại bỏ một dòng lệnh nào đó trong chương trình. Thông thường lời chú giải cần phải mang đủ thông tin để giải thích về thao tác của lệnh trong hoàn cảnh cụ thể và như thế thì mới có ích cho người đọc. Đối với những người mới lập trình bằng hợp ngữ còn thiếu kinh nghiệm thì lời chú giải còn phản ảnh sự hiểu biết về vấn đề phải giải quyết của họ, vì nếu không hiểu thấu đáo vấn đề thì không để đưa ra lời chú giải tốt được. Tóm lại là ta nên tránh việc đưa ra một lời chú giải vô bổ (không mang thông tin) kiểu như:

```
MOV  BX, 0       ; đưa 0 vào thanh ghi BX
```

Vì tự thân lệnh gợi nhớ đó đã có ý nghĩa như lời giải thích rồi. Nếu trong bài toán cụ thể thanh ghi BX được chọn dùng làm tổng tích luỹ cho một tính toán nhất định ta có thể có lời chú giải hơn như sau:

```
MOV BX, 0    ;        tổng tích luỹ ở BX lúc đầu bằng 0.
```

Ta cũng có thể dùng một vài dòng lệnh chỉ để làm chú giải cho một công việc nào đó. Cần lưu ý là mỗi dòng chú giải đó phải bắt đầu bằng dấu chấm phẩy. Ví dụ:

```
; khởi đầu thanh ghi DS và ES trong đoạn dữ liệu
MOV  AX, @DATA
MOV  DS, AX
```

```
MOV  ES, AX
```

4.1.2. Dữ liệu cho chương trình

Dữ liệu của một chương trình hợp ngữ là rất đa dạng. Các dữ liệu có thể được cho dưới dạng số hệ hai, hệ mười, hệ mười sáu hoặc dưới dạng ký tự (cần chú ý là trên các máy IBM PC trong chương trình DEBUG, một công cụ tìm lỗi rất thông dụng cho các chương trình hợp ngữ đơn giản, dữ liệu bằng số được ngầm định phải ở hệ mười sáu).

Khi cung cấp số liệu cho chương trình, số cho ở hệ nào phải được kèm đuôi của hệ đó như ta đã nói rõ ở chương I (trừ hệ mười thì không cần vì là trường hợp ngầm định của assembler). Riêng đối với số hệ mười sáu nếu số đó bắt đầu bằng các chữ (a.f hoặc A.. F) thì ta phải thêm 0 ở trước để chương trình dịch có thể hiểu được đó là một số hệ mười sáu chứ không phải là một tên hoặc một nhãn.

Ví dụ các số viết đúng:

```
0011B       ; Số hệ hai.
1234        ; Số hệ mười
0ABBAH      ; Số hệ mười sáu, không nhầm được với
            ; tên của một ban nhạc nổi tiếng ABBA.
1EF1H ; Số hệ mười sáu.
```

Nếu dữ liệu là ký tự hoặc chuỗi ký tự thì chúng phải được đóng trong căpk dấu trích dẫn đơn hoặc kép, thí dụ 'A' hay "abcd". Chương trình dịch sẽ dịch ký tự ra mã ASCII tương ứng của nó, vì vậy trong khi cung cấp dữ liệu kiểu ký tự cho chương trình ta có thể dùng bản thân ký tự được đóng trong dấu trích dẫn hoặc mã ASCII của nó. Ví dụ, ta có thể sử dụng liệu ký tự là "0" hoặc mã ASCII tương ứng là 30H, ta cps thể dùng '$' hoặc 26H hoặc 34...

4.1.3. Biến và hằng

Biến trong chương trình hợp ngữ có vai trò như nó có ở ngôn ngữ bậc cao. Một biến phản được định kiểu dữ liệu là kiểu byte hay kiểu từ và sẽ được chương trình dịch gán cho một địa chỉ nhất định trong bộ nhớ. Để định nghĩa các kiểu dữ liệu khác nhau ta thường dùng các lệnh giả sau:

```
DB (define byte)              : định nghĩa biến kiểu byte
DW (define word)              : định nghĩa biến kiểu từ
DD (define double word)       : định nghĩa biến kiểu từ kép
```

Biến byte

Biến kiểu byte sẽ chiếm 1 byte trong bộ nhớ. Hướng dẫn chương trình dịch để định nghĩa biến kiểu byte có dạng tổng quát như sau:

```
Tên     DB     giá_ trị_khởi_đầu
```

Ví dụ:

```
B1      DB     4
```

Ví dụ trên định nghĩa biến byte có tên là B1 và dành 1 byte trong bộ nhớ cho nó để chứa giá trị khởi đầu bằng 4.

Nếu trong lệnh trên ta dùng dấu? thay vào vị trí của số 4 thì biến B1 sẽ được dành chỗ trong bộ nhớ nhưng không được gán giá trị khởi đầu. Cụ thể dòng lệnh giả:

```
B2      DB     ?
```

chỉ định nghĩa 1 biến byte có tên là B2 và dành cho nó một byte trong bộ nhớ.

Một trường hợp đặc biệt của biến byte là biến ký tự. Ta có thể có định nghĩa biến kỳ tự như sau:

```
C1      DB     ' $'
C2      DB     34
```

- Biến từ

Biến từ cũng được định nghĩa theo cách giống như biến byte. Hướng dẫn chương trình dịch để định nghĩa biến từ có dạng như sau:

```
Tên     DB     giá_ trị_khởi_đầu
```

Ví dụ:

```
W1      DW     40
```

Ví dụ trên định nghĩa biến từ có tên là W1 và dành 2 byte trong bộ nhớ cho nó để chứa giá trị khởi đầu bằng 40.

Chúng ta cũng có thể sử dụng dấu? chỉ để định nghĩa và dành 2 byte trong bộ nhớ cho biến từ W2 mà không gán giá trị đầu cho nó bằng dòng lệnh sau:

```
W2      DW     ?
```

- Biến mảng

Biến mảng là biến hình thành từ một dãy liên tiếp các phần tử cùng loại byte hoặc 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:

```
M1      DB      4, 5, 6, 7, 8, 9
```

Ví dụ trên định nghĩa biến mảng có tên là M1 gồm 6 byte và dành chỗ cho nó trong bộ nhớ từ địa chỉ ứng với M1 để chứa các giá trị khởi đầu bằng 4, 5, 6, 7, 8, 9. Phần tử đầu tỏng mảng là 4 và có địa chỉ trùng với địa chỉ của M1, phần tử thứ hai là 5 và có địa chỉ M1+1...

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

Ví dụ:

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

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

Toán tử DUP có thể lồng nhau để định nghĩa ra 1 mảng.

Ví dụ: dòng lệnh

```
M4      DB      4, 3, 2, 2  DUP(1,2 DUP(5),6)
```

Sẽ định nghĩa ra một mảng M4 tương đương với lệnh sau:

```
M4      DB      4,3,2,1,5,5,6,1,5,5,6
```

Một điều cần chú ý nữa là đối với các bộ vi xử lý của Intel, nếu ta có một từ để trong bộ nhớ thì byte thấp của nó sẽ được để ở ô nhớ có địa chỉ thấp, byte cao sẽ được để ở ô nhớ có địa chỉ cao. Cách lưu giữ số liệu kiểu này cũng còn có thể thấy ở các máy VAX của Digital hoặc của một số hãng khác và thường gọi là 'quy ước đầu bé' (little endian, byte thấp được cất tại địa chỉ thấp). Cũng nên nói thêm ở đây là các bộ vi xử lý của motorola lại có cách cất số liệu theo thứ tự ngược lại hay còn được gọi là 'quy ước đầu to' (big endian byte cao được cất tại địa chỉ thấp) .

Ví dụ: Sau khi định nghĩa biến từ có tên là WORDA như sau:

```
WORDA       DW     OFFEEH
```

Thì ở trong bộ nhớ thấp (EEH) sẽ được để tại địa chỉ WORDA còn byte cao (FFH) sẽ được để tại địa chỉ tiếp theo, tức là tại WORDA+1

- Biến kiểu xâu kí tự

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

Các ví dụ sau đều là các lệnh đúng và đều định nghĩa cùng một xâu kí tự nhưng gắn nó cho các tên khác nhau:

```
STR1  DB    'string'
STR2  DB    73h, 74h, 72h, 69h, 6Eh, 67h
STR3  DB    73h, 74h, 'x' 'i', 6Eh, 67h
```

- Hằng có tên

Các hằng trong chương trình hợp ngữ thường được gán tên để làm cho chương trình trở nên dễ đọc hơn.

Hằng có thể là kiểu số hay kiểu ký tự. Việc gán tên cho hằng được thực hiện nhờ lệnh giả EQU (equate) như sau:

```
CR    EQU   0Dh    ;CR là carriage return
LE    EQU   0Ah    ;LF là line feed
```

Trong ví dụ trên lệnh giả EQU gán giá trị số 13 (mã ASCII của kí tự trở về đầu dòng) cho tên CR và 10 (mã ASCII của ký tựu thêm dòng mới) cho tên LF.

Hằng cũng có thể là một chuỗi ký tự. trong ví dụ dưới đây sau khi đã gán một chuỗi ký tự cho một tên:

```
CHAO  EQU   'Hello'
```

ta có thể sử dụng hằng này để định nghĩa một biến mảng khác.

```
MSG   DB    CHAO, '$'
```

Vì lệnh giả EQU không dành chỗ của bộ nhớ cho tên của hằng nên ta có thể đặt nó khá tự do tại những chỗ thích hợp bên trong chương trình. Tuy nhiên trong thực tế người ta thường đặt các định nghĩa này trong đoạn dữ liệu.

4.1.4. Khung của một chương trình hợp ngữ

Một chương trình mã máy trong bộ nhớ thường bao gồm các vùng nhớ khác nhau để chứa mã lệnh, chứa dữ liệu của chương trình và một vùng nhớ khác được dùng làm ngăn xếp phục vụ hoạt động của chương trình. Chương trình viết bằng hợp ngữ cũng phải có cấu trúc tương tự để khi được dịch nó sẽ tạo ra mã tương ứng với chương trình mã máy nói trên. Để tạo ra sườn của một chương trình hợp ngữ chúng ta sẽ sử dụng cách định nghĩa đơn giản đối với mô hình bộ nhớ dành cho chương trình và đối với các thanh ghi đoạn, cách định nghĩa được phép từ phiên bản 5.0 của Microsoft Macro Aesembler,...

✓ **Khai báo quy mô sử dụng bộ nhớ**

Kích thước của bộ nhớ dành cho đoạn mã và đoạn dữ liệu trong một chương trình được xác định nhờ hướng dẫn chương trình dịch MODEL như sau (hướng dẫn này phải được đặt trước các hướng dẫn khác trong chương trình hợp ngữ, nhưng sau hướng dẫn về loại CPU):

```
.MODEL      Kiểu_ kích_thước_bộ_nhớ
```

Có nhiều Kiểu_ kích_thước_bộ_nhớ cho các chương trình với đòi hỏi dung lượng bộ nhớ khác nhau. Đối với ta thông thường các ứng dụng đòi hỏi mã chương trình dài nhất cũng chỉ cần chứa trong một đoạn (64KB), dữ liệu cho chương trình nhiều nhất cũng chỉ cần chứa trong một đoạn, thích hợp nhất nên chọn Kiểu_ kích_thước_bộ_nhớ là Small (nhỏ) hoặc nếu như tất cả mã và dữ liệu có thể gói trọn được trong một đoạn thì có thể chọn Tiny(hẹp):

```
.Model small
```

hoặc

```
.Model Tiny
```

Ngoài Kiểu_ kích_thước_bộ_nhớ nhỏ hoặc hẹp nói trên, tuỳ theo nhu cầu cụ thể MASM còn cho phép sử dụng các Kiểu_ kích_thước_bộ_nhớ khác như liệt kê trong bảng 4.1

Các kiểu kích thước bộ nhớ cho chương trình hợp ngữ

Kiểu kích thước	Mô tả
Tiny (Hẹp)	Mã lệnh và dữ liệu gói gọn trong một đoạn
Small (Nhỏ)	Mã lệnh gói gọn trong một đoạn , dữ liệu nằm trong một đoạn.
Medium (Trung bình)	Mã lệnh không gói gọn trong một đoạn , dữ liệu nằm trong một đoạn.
Compact (Gọn)	Mã lệnh không gói gọn trong một đoạn , dữ liệu không gói gọn trong một đoạn.
Large (lớn)	Mã lệnh không gói gọn trong một đoạn , dữ liệu không gói gọn trong một đoạn. không có mảng nào lớn hơn 64KB.
Huge (Đồ sộ)	Mã lệnh không gói gọn trong một đoạn , dữ liệu không gói gọn trong một đoạn. các mảng có thể lớn hơn 64KB

✓ **Khai báo đoạn ngăn xếp**

Việc khai báo đoạn ngăn xếp là cốt để dành ra một vùng nhớ đủ lớn dùng làm ngăn xếp phục vụ cho hoạt động của chương trình khi có chương trình con. Việc khai báo được thực hiện nhờ hướng dẫn chương trình dịch như sau.

.Stack Kích_thước

Kích_thước sẽ quyết định số byte dành cho ngăn xếp. Nếu ta không khai Kích_thước thì chương trình dịch sẽ tự động gán cho Kích_thước giá trị 1 KB, đây là kích thước ngăn xếp quá lớn đối với một ứng dụng thông thường. Trong thực tế các bài toán của ta thông thường với 100-256 byte là đủ để làm ngăn xếp và ta có thể khai báo kích thước cho nó như sau:

```
.Stack          100
Hoặc    .Stack          100H
```

✓ **Khai báo đoạn dữ liệu**

Đoạn dữ liệu chứa toàn bộ các định nghĩa cho các biến của chương trình. Các hằng cũng nên được định nghĩa ở đây để đảm bảo tính hệ thống mặc dù ta có thể để chúng ở trong chương trình như đã nói ở phần trên.

Việc khai báo đoạn dữ liệu được thực hiện nhờ hướng dẫn chương trình dịch DATA, việc khai báo và hằng được thực hiện tiếp ngay sau đó bằng các lệnh thích hợp. Điều này được minh hoạ trong các thí dụ đơn giản sau:

```
.Data
        MSG   DB       'helo!$'
        CR    DB       13
        LF    EQU      10
```

- Khai báo đoạn mã

Đoạn mã chứa mã lệnh của chương trình. Việc khai báo đoạn mã được thực hiện nhờ hướng dẫn chương trình dịch .CODE như sau:

.CODE

Bên trong đoạn mã, các dòng lệnh phải được tổ chức một cách hợp lý, đúng ngữ pháp dưới dạng một chương trình chính (CTC) và nếu cần thiết thì kèm theo các chương trình con (ctc). Các chương trình con sẽ được gọi ra bằng các lệnh CALL có mặt bên trong chương trình chính.

Một thủ tục được định nghĩa nhờ các lệnh giả PROC và ENDP. Lệnh giả PROC để bắt đầu một thủ tục còn lệnh giả ENDP được dùng để kết thúc nó. Như vậy một chương trình chính có thể được định nghĩa bằng các lệnh giả PROC và ENDP theo mẫu sau:

```
Tên_CTC         Proc
; Các lệnh của thân chương trình chính
  :
CALL Tên_ ctc; gọi ctc
  :
Tên_CTC         Endp
```

Giống như chương trình chính con cũng được định nghĩa dưới dạng một thủ tục nhờ các lệnh giả PROC và ENDP theo mẫu sau:

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

Trong các chương trình nói trên, ngoài các lệnh giả có tính nghi thức bắt buộc ta cần chú ý đến sự bố trí của lệnh gọi (CALL) trong chương trình chính và lệnh về (RET) trong chương trình con.

- Khung của chương trình hợp ngữ để dịch ra chương trình .EXE

Từ các khai báo các đoạn của chương trình đã nói ở trên ta có thể xây dựng một khung tổng quát cho các chương trình hợp ngữ với kiểu kích thước bộ nhớ nhỏ. Sau đây là một khung cho chương trình hợp ngữ để rồi sau khi được dịch (assembled) nối (linked) trên máy IBM PC sẽ tạo ra một tệp chương trình chạy được ngay (executable) với đuôi .EXE.

```
. Model small
.Stack 100
.Data
        ; các định nghĩa cho biến và hằng để tại đây
.Code
MAIN Proc
        ; Khởi đầu cho  DS
          MOV AX, @Data
          MOV DS, AX
        ; Các lệnh của chương trình chính để tại đây
        ; Trở về DOS dùng hàm 4CH của INT 21H
          MOV AH, 4CH
          INT 21 H
MAIN Endp
        ; các chương trình con (nếu có )       để tại đây
        END MAIN
```

Trong khung chương trình trên, tại dòng cuối cùng của chương trình ta dùng hướng dẫn chương trình dịch END và tiếp theo là MAIN để kết thúc toàn bộ chương trình. Ta có nhận

xét rằng MAIN là tên của chương trình chính nhưng quan trọng hơn và về thực chất thì nó là nơi bắt đầu các lệnh của chương trình trong đoạn mã.

Khi một chương .EXE được nạp vào bộ nhớ. DOS sẽ tạo ra một mảng gồm 256 byte của cái gọi là *đoạn mào đầu chương trình* (Programsegment prefix. PSP) dùng để chứa các thông tin liên quan đến chương trình và các thanh ghi DS và ES. Do vậy DS và ES không chứa giá trị địa chỉ của các đoạn dữ liệu cho chương trình của chúng ta. Để chương trình có thể chạy đúng ta phải có các lệnh sau để khởi đầu cho thanh ghi DS (hoặc caES nữa nếu cần):

```
MOV AX, @Data
MOV DS, AX ; nếu cần thì bỏ ';'
```

trong đó @ Data là tên của đoạn dữ liệu. Data định nghĩa bởi hướng dẫn chương trình dịch sẽ dịch tên @ Data thành giá trị số của đoạn dữ liệu. Ta phải dùng thanh ghi AX làm trung gian cho việc khởi đầu DS như trên là do bộ vi xử lý 8086/88, Vì những lí do kỹ thuật, không cho phép chuyển giá trị số (chế độ địa chỉ tức thì) vào các thanh ghi đoạn. Thanh ghi AX cũng có thể được thay thế bằng các thanh ghi khác.

Sau đây là ví dụ của một chương trình hợp ngữ được viết để dịch ra chương trình với đuôi .EXE. khi cho chạy, chương trình này sẽ hiện lên màn hình lời chào 'Hello' nằm giữa hai dòng trống cách đều các dòng mang dấu nhắc của DOS.

Chương trình 4.1 Chương trình Hello.EXE

```
. Model Small
. Stack 100
. Data
        CRLF        DB      13,10,' $ '
        MSG         DB      ' Hello!$ '
. Code
MAIN Proc
                ; khởi đầu thanh ghi DS
        MOV AX,@Data
        MOV DS, AX
                ; về đầu dòng mới dùng hàm 9 của INT 21H
        MOV AH, 9
        LEA DX, CRLF
        INT 21H
                ; hiện thị lời chào dùng hàm 9 của INT 21H
        MOV AH, 9
        LEA DX, MSG
        INT 21H
                ; về đầu dòng mới dùng hàm 9 của INT 21H
        MOV AH, 9
        LEA DX, CFLF
        INT 21H
                ; trở về DOS dùng hàm 9 của INT 21H
        MOV AH, 4CH
        INT 21H
MAIN Endp
        END MAIN
```

Trong ví dụ trên chúng ta đã sử dụng các dịch vụ có sẵn (các hàm 9 và 4CH) của ngắt INT 21H của DOS trên máy IBM PC để hiện thị xâu ký tự và trở về DOS một cách thuận lợi.

Chúng ta sẽ nói kỹ hơn về các ngắt này ở chỗ khác.

• Khung của chương trình hợp ngữ để dịch ra chương trình. COM

Nhìn vào khung chương trình hợp ngữ để dịch ra tệp chương trình đuôi .EXE ta thấy có mạt đầy đủ các đoạn. Trên máy tính IBM PC ngoài tệp chương trình với đuôi .EXE. Chúng ta còn có khả năng dịch chương trình hợp ngữ có kết cấu thích hợp ra một loại tệp chương trình chạy được kiểu khác với đuôi .COM. Đây là một chương trình ngắn gọn và đơn giản hơn nhiều so với tệp chương trình đuôi .EXE, trong đó các đoạn mã, đoạn dữ liệu và đoạn ngăn xếp được gộp lại trong một đoạn duy nhất là đoạn mã. Như vậy nếu ta có các ứng dụng mà dữ liệu và mã chương trình không yêu cầu nhiều về không gian của bộ nhớ, ta có thể ghép luôn cả dữ liệu, mã chương trình và ngăn xếp chung vào trong cùng một đoạn mã rồi tạo ra tệp .COM. Với việc tạo ra tệp này còn tiết kiệm được cả không gian nhớ khi phải lưu trữ nó trên ổ đĩa.

Để có thể dịch được ra chương trình đuôi .COM thì chương trình nguồn hợp ngữ phải được kết cấu sao cho thích hợp với mục đích này.

Sau đây là khung của một chương trình hợp ngữ để dịch được ra tệp chương trình đuôi .COM.

```
. Model Tiny
. Code
        ORG 100h
START: JMP CONTINUE
    ;   các định nghĩa cho biến và hằng để tại đây
CONTINUE :
MAIN Proc
    ;   các lệnh của chương trình chính để tại đây
    INT 20H          ;  Trở về DOS
MAIN Endp
    ;   các chương trình con (nếu có) để tại đây
    END START
```

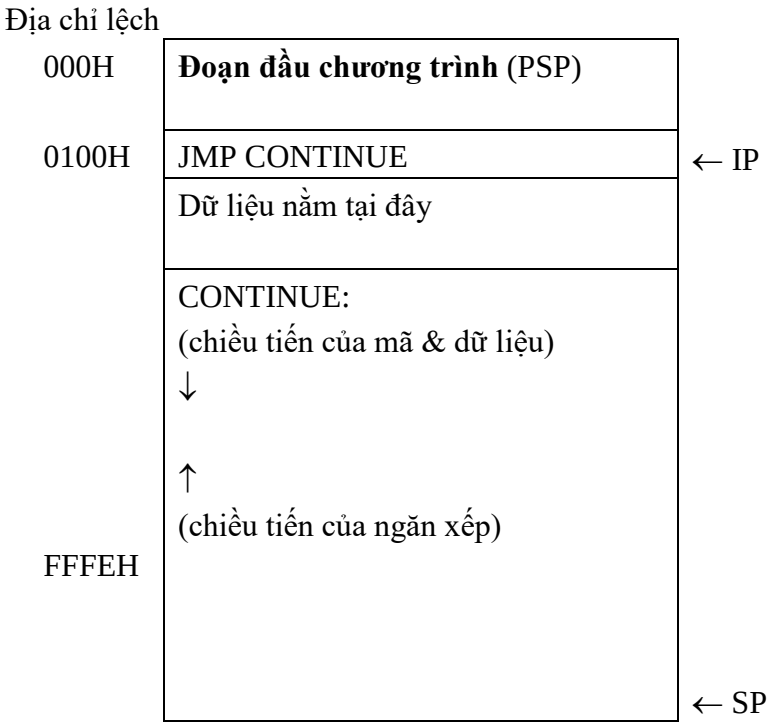
So sánh khung này với khung cho chương trình .EXE ta thấy trong khung không có khai báo đoạn ngăn xếp và đoạn dữ liệu, còn khai báo quy mô sử dụng nhớ là kiểu Tiny. Ở ngay đầu đoạn mã là lệnh giả ORG (origin: điểm xuất phát) lệnh JMP (nhảy). Lệnh giả ORH 100H dùng để gán địa chỉ bắt đầu cho chương trình tại 100H trong đoạn mã, chừa lại vùng nhớ với dung lượng 256 byte (từ địa chỉ 0 đến địa chỉ 255) cho đoạn mào đầu chương trình (PSP).

Lệnh JMP sau nhãn START dùng để nhảy qua phần bộ nhớ dành cho việc định nghĩa và khai báo dữ liệu (về nguyên tắc, dữ liệu có thể được đặt ở đầu hoặc ở cuối đoạn mã, nhưng ở đây ta đặt nó ở đầu đoạn mã để có thể áp dụng các định nghĩa đơn giản đã nói). Đích của lệnh nhảy là phần đầu của chương trình chính. Hình 4.1 biểu diễn việc một chương trình kiểu .COM được nạp vào và sắp xếp trong một đoạn mã của bộ nhớ ra sao.

Theo hình 4.1 ta thấy một chương trình .COM cũng được nạp vào bộ nhớ sau vùng PSP như chương trình đuôi .EXE. Ngăn xếp cho chương trình .COM được xếp đặt tại cuối đoạn mã, đỉnh của ngăn xếp lúc ban đầu là ô nhớ có địa chỉ là FFFEH.

Trong trường hợp chương trình kiểu.COM này chúng ta sẽ bị các hạn chế gây ra bởi a) dung lượng nhớ cực đại của một đoạn là 64KB, tức là ta phải luôn chắc chắn được rằng các chương trình ứng dụng phải có số lượng byte của mã máy và dữ liệu cho chương

trình không lớn lắm (nếu không nó sẽ làm cho cả nhóm này nở ra về phía địa chỉ cao của đoạn) và b) chương trình cũng chỉ được phép sử dụng ngăn xếp một cách hạn chế (nếu không điều này có thể làm cho đỉnh của nó trong khi hoạt động dâng lên nhiều về phía địa chỉ thấp của đoạn).



Hình 4.1. Tệp chương trình .COM trong bộ nhớ

Tóm lại chúng ta phải chắc chắn đảm bảo không thể xảy ra hiện tượng trùm vào nhau của các thông tin tại vùng mã lệnh hoặc dữ liệu. Tuy nhiên ta cũng không cần phải lo lăng quá đến vấn đề này, các chương trình kiểu .COM trong hầu hết các trường hợp trong thực tế vẫn có thể thoả mãn được các yêu cầu của các bài toán mà ta muốn thử nghiệm.

Khi kết thúc chương trình kiểu .COM, để trở về DOS ta dùng ngắt INT 20H của DOS để làm cho chương trình gọn hơn. Tất nhiên ta cũng có thể dùng hàm 4CH của ngắt INT 21H như đã dùng trong chương trình để dịch ra tệp .EXE.

Để kết thúc toàn bộ chương trình ta dùng hướng dẫn chương chính dịch END đi kèm theo nhãn START tương ứng với địa chỉ lệnh đầu tiên của chương trình trong đoạn mã.

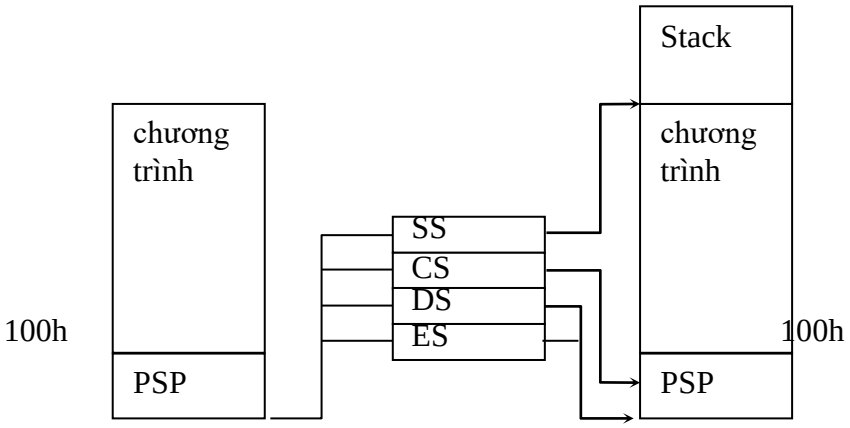
Sau đấy là ví dụ của một chương trình hợp ngữ để dịch ra tệp chương trình chạy được với đuôi .COM (chương trình 4.2).

Chương trình 4.2. Chương trình Helo.COM

```
. Model Tiny
. Code
                ORG 100H
START :         IMP CONTINUE
CRLF            DB      13, 10, '$'
MSG             DB      !Hello! $'
CONTINUE:
MAIN Proc
        ;   về đầu dòng mới dùng hàm 9 của INT 21H
                MOV AH, 9
                LEA DX, CRLF
```

```
        INT 21H
;   hiện thị lời chào
        MOV AH, 9
        LEA DX, CRLF
        INT 21H
;    trở về DOS
        INT 20H
MAIN Endp
        END START
```

Chúng ta có thể nhận xét rằng trong chương trình 4.2 ta không cần đến các thao tác khởi đầu cho thanh ghi DS, như ta đã phải làm trong chương trình 4.1, vì trong chương trình.COM không có đoạn dữ liệu nằm riêng rẽ.



Hình 4.2. Môđun chương trình .COM và .EXE trong bộ nhớ.

Cuối cùng để kết thúc phần nói về các chương trình kiểu .COM và .EXE ta đưa ra hình ảnh của các chương trình này khi chúng được tải vào trong bộ nhớ để có thể tiện so sánh (hình 4.2).

2. Cách tạo và cho chạy một chương trình hợp ngữ trên máy IBM PC

Như đã nói trong phần trước, máy IBM PC là phương tiện lý tưởng để chúng ta tạo ra và thử nghiệm các chương trình hợp ngữ 8086/88. Các bước để làm công việc này có thể liệt kê ra như sau:

Dùng các phần mềm soạn thảo văn bản (SK, NCedit...) để tạo ra một tệp văn bản chương trình gốc bằng hợp ngữ. Tệp này phải được gán đuôi .ASM.

Dùng chương trình dịch MASM để dịch tệp.ASM ra mã máy dưới dạng tệp .OBJ. Nếu trong bước này nếu trong chương trình có lỗi cú pháp thì ta phải quay lại bước 1 để sửa lại chương trình gốc.

Dùng chương trình LINK để nối một hay nhiều tệp OBJ lại với nhau thành một tệp chương trình chạy được với đuôi .EXE.

Nếu chương trình gốc viết ra là để dịch ra kiểu .COM thì ta phải dùng chương trình EXE2BIN (đọc là EXEtoBIN) của DOS để dịch tiếp tệp .EXE ra tệp chương trình chạy được với đuôi .COM.

Cho chạy chương trình vừa dịch

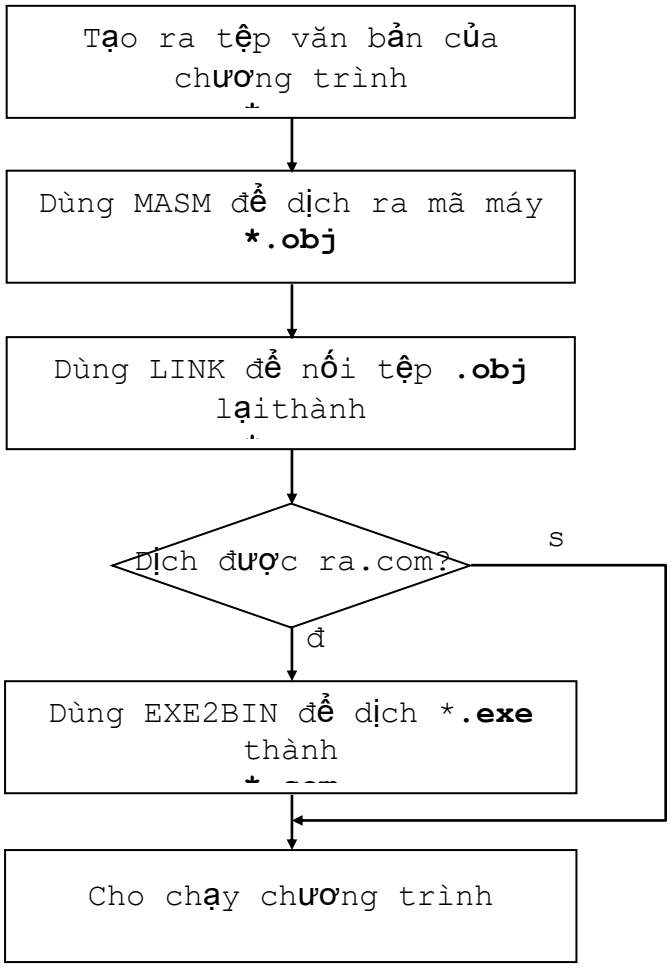
Qua quá trình thực hiện các công việc nói trên có thể được minh họa bằng lưu đồ trên hình 4.2.

Sau đây ta sẽ nói kỹ và cụ thể hơn về một số công đoạn trong quá trình soạn thảo và cho chạy chương trình hợp ngữ.

- Để soạn thảo chương trình hợp ngữ ta có thể dùng các chương trình soạn thảo văn bản như BKED hoặc các chức năng soạn thảo của Turbo Pascal, SK, NC... nhưng tốt nhất nên dùng các chương trình SK hoặc NC vì các chương trình này có khả năng thường trú do đó rất thích hợp cho việc sửa đổi và cho chạy thử chương trình.
- Trong bước 2 ta có thể dùng lệnh MASM theo 2 cách khác nhau.

Giả thiết chúng ta có tệp chương trình 4.2 mang tên vidu.asm tại ổ đĩa A và chương trình dịch hợp ngữ MASM tại ổ C. Nếu ta không muốn MASM trong khi dịch tạo ra các tệp khác ngoài tệp .obj thì tại dấu nhắc A> của DOS ta gõ vào lệnh sau (chú ý dấu chấm phẩy';'):

A>C:MASM vidu;



Các bước công việc để tạo ra và cho chạy một chương trình hợp ngữ

Sau lệnh này MASM sẽ kiểm tra lỗi của chương trình gốc và đưa ra bản thống kê các lỗi nếu có.

Trong trường hợp chương trình hợp ngữ không bị lỗi ta nhận được các dòng thông báo sau:

A>C:MASM vidu;

Microsoft (R) Macro Assembler Version 5.00A

Copy rights (C) Microsoft Corp 1981-1985, 1987 All rights reserved

49020 Bytes symbol space free

0 Warning Errors

0 Severe Errors

Kết cục ta thu được một tệp mã máy là vidu.obj để tại ổ A.

Trong ví dụ trên, giả thiết có lỗi cú pháp (chẳng hạn khi định nghĩa chuỗi ký tự bị thiếu dấu ' tại dòng lệnh thứ 6 trong chương trình gốc), MASM sẽ thông báo lỗi như sau:

Micrisoft (R) Macro Assembler Version 5.00A

Copy rights (C) Microsoft Corp 1981-1985, 1987 All rights reserved

Object filename [VIDU.OBJ] :

Source listing [NUL.LST] : vi du

Cross - reference [NUL.CRF] : vidu

49020 Bytes symbol space free

0 Warning Errors

0 Severe Errors

Ta thấy trong trường hợp này MASM làm việc ở chế độ đối thoại và ta phải trả lời bằng cách gõ thêm các thông tin cần thiết cho việc tạo ca các tệp mới.

Thông tin đầu tiên liên quan đến tên tệp .obj.MASM mặc định tên tệp obj trùng với tên tệp.asm, nếu ta muốn có thây đổi thì gõ thâm vào bên cạnh, nếu không chỉ cần gõ Enter để cho qua.

Thông tin thứ 2 liên quan đến tên tệp .lst. Đây là tệp chứa văn bản của chương trình gốc kèm theo mã máy của từng dòng lệnh. MASM mặc định tên tệp là NUL, tức là không tạo ra tệp này (gõ Enter để cho qua). Nếu muốn có tệp này ta phải gõ thêm tên vào.

Thông tin thứ 3 liên quan đến tên tệp .crf, tệp tham khảo chéo. Tệp tham khảo chéo chứa danh sách các tên và nhãn có trong chương trình cùng với toạ độ của chúng (tọa độ ở đây là số thứ tự dòng). MASM mặc định tên tệp là NUL, tức là không tạo ra tệp này (gõ Enter để cho qua). Nếu muốn có tệp này ta phải gõ thêm tên vào.

- Sau khi tạo ra tệp .obj, trong bước 3 ta phải dùng chương trình LINK để biến (hay nối) chương trình mã máy để trong một hay nhiều tệp .obj thành một tệp chương trình chạy được duy nhất với đuôi .EXE. Nếu trong ổ C ta có chương trình LINK và tệp vidu.obj ở ổ A thì ta có thể có lệnh:

A>C:LINK vidu;

Microsoft (R) Overlay Linker Version 3.65

Copy rights (C) Microsoft Corp 1983-1988 All rights reserved

Nếu tệp gốc là tệp được viết ra để dịch ra chương trình kiểu .COM thì trong bước này ta có thể gặp thông báo lỗi như sau:

LINK : Warning L4021 : No stack segment (không có đoạn ngăn xếp)

Thông báo này là lời cảnh báo đúng vì chương trình kiểu .COM không có ngăn xếp riêng và vì vậy chương trình của ta vẫn được dịch bình thường.

Kết cục ta thu được một tệp chương trình chạy được là vidu.exe để tại ổ A.

Cần chú ý rằng cho dù vidu.obj là tệp .obj duy nhất nhưng ta vẫn dùng LINK để biến nó thành tệp chạy được với đuôi EXE. Nếu ta có nhiều môđun chương trình được dịch ra dưới dạng tệp .obj và các tệp này cần phải được nối để tạo thành tệp. EXE duy nhất cuối cùng, ta có thể dùng lệnh sau:

A > **C:LINK viđu□‹viu2;**

trong đó vidu1.obj là một môđun chương trình, bên trong nó có chứa lời gọi một chương trình khác hoặc tham chiếu tới các tham số ở một môđun có tên vidu2.obj. Tên của tệp được đặt tại vị trí đầu tiên trong phần thông số của lệnh LINK (trong trường hợp của ta là vidu1.obj), sẽ được chọn để đặt tên cho tệp .EXE cuối cùng. Nghĩa là sau dòng lệnh trên ta sẽ thu được tệp vidu1.exe.

- Chỉ đối với các chương trình gốc được viết để dịch ra chương trình đuôi .COM ta mới phải tiến hành bước thứ 4, tức là dùng EXE2BIN để dịch tiếp chương trình .EXE vừa thu được ra chương trình .COM. Cần lưu ý rằng một chương trình .EXE được tạo ra để rồi tiếp theo được dịch thành Chương trình .COM (như trường hợp vidu.exe của chúng ta) không phải là một chương trình chạy được.

Giả thiết có chương trình EXE2BIN ở tại ổ C và chương trình vidu.exe ở tại ổ A ta có thể sử dụng lệnh sau:

A>**C:EXE2BIN vidu vidu.com**

Chú ý: Trong câu lệnh cuối cùng ta phải viết rõ vidu.com để nhận được tệp đuôi.COM vì lệnh EXE2BIN này mặc định sẽ tạo ra tệp chương trình đuôi .BIN để dùng cho các công việc khác.

Kết cục ta thu được một tệp chương trình chạy được là vidu.com để tại ổ A.

• Bước cuối cùng là cho chạy chương trình vừa thu được. Các chương trình chạy được đuôi .COM hay .EXE đều được cho chạy trong môi trường DOS bằng cách gọi tên của nó.

Để kết thúc ví dụ đã nêu ta có thể cho chạy chương trình vidu.com bằng lệnh:

A> **vidu**

và trên màn hình sẽ thu được các dòng chữ như sau:

A> vidu

Hello!

A>

Để làm được tất cả các công việc trên một cách thuận lợi ta có thể tập hợp các thao tác phải làm vào tong một tệp xử lý theo lô **run.bat** như sau (chương trình **4.3**)

Chương trình 4.3 Chương trình RUN.BAT

```
@ Echo Assemble and link %1
@ If not exits 1.asm goto END
MASM % 1, %1;
@ If Errorlevel 1 goto END
LINK %1, %1, NUL;
@ If not exits %1.exe goto END
@ If not 'A%2'=='A' goto RUN
EXE2BIN % 1.exe %1.com
@ DEL @1.exe
@ DEL @1.obj
: RUN
PAUSE Ctrl_Break ta terminate or
%1
:       END
```

Chương trình run.bat giả thiết là ta có sẵn các chương trình MASM.LINK và EXE2BIN để trong thư mục hiện tại cùng với tệp văn bản gốc filename.asm được soạn thảo thích hợp để dich ra tệp chương trình đuôi .EXE và .COM. Tuỳ theo mục đích dịch ta có các câu lệnh sau:

+ **RUN filename /**

để dịch tệp chương trình gốc soạn theo kiểu .EXE ra tệp chương trình.EXE và cho chạy luôn chạy chương trình này.

+ **RUN filename**

để dịch tệp chương trình gốc soạn theo kiểu .COM ra tệp chương trình.COM và cho chạy luôn chạy chương trình này.

4.2. Các cấu trúc lập trình cơ bản thực hiện bằng hợp ngữ

Ngày nay, trong khi tiến hành việc thiết kế hệ thống người ta thường dùng phương pháp *thiết kế từ trên xuống dưới.* Phương pháp này ví thế cũng được áp dụng trong khi viết phần mềm cho một hệ thống nhằm giải quyết một nhiệm vụ nhất định nào đó.

Bản chất của phương pháp thiết kế này là đầu tiên ta chia chương trình tổng thể thành các khối chức năng nhỏ hơn, các khối chức năng nhỏ này lại được chia tiếp thành các khối chức năng nhỏ hơn nữa, việc phân chia chức năng phải làm cho đến khi mỗi khối nhỏ này trở thành các khối chức năng đơn giản và dễ thực hiện.

Trong khi thực hiện các khối chức năng thành phần, thông thường người ta sử dụng các cấu trúc lập trình cơ bản để thực hiện các nhiệm cụ của khối đó. Điều này làm cho các chương trình viết ra trở thành có *cấu trúc* với các ưu điểm chính là dễ phát triển, dễ hiệu chỉnh hoặc cải tiến và dễ lập tài liệu.

Để giải quyết các công việc khác nhau thông thường trong khi viết chương trình ta chỉ cần đến 3 cấu trúc lập trình cơ bản sau:

+ cấu trúc tuần tự.

+ Cấu trúc lựa chọn (IF-THEN-ELSE) và

+ Cấu trúc lặp (WHILE.DO).

Thay đổi các cấu trúc này một chút ít, ta có thể tạo thêm 4 cấu trúc khác cũng rất có tác dụng trong khi viết chương trình :

+ cấu trúc chọn kiểu IF-THEN

+ cấu trúc chọn kiểu CASE,

+ cấu trúc lặp kiểu REPEAT-UNTIL và

+ cấu trúc lặp kiểu FOR-DO.

Đặc điểm chung của tất cả các cấu trúc lập trình cơ bản là tính cấu trúc chi có một lối vào cấu trúc và một lối ra để ra khỏi cấu trúc đó.

Những cấu trúc lập trình kể trên là các cấu trúc mà ta đã làm quen ít nhiều khi viết chương trình ở ngôn ngữ bậc cao. Vấn đề đối với ta bây giờ là làm thế nào để thực hiện các cấu trúc lập trình này bằng hợp ngữ.

- **Cấu trúc tuần tự**

Cấu trúc tuần tự là một cấu trúc thông dụng và đơn giản nhất. Trong cấu trúc này các lệnh được sắp xếp tuần tự, lệnh nọ tiếp lệnh kia. Sau khi thực hiện xong lệnh cuối cùng của cấu trúc thì công việc phải làm cũng được hoàn tất.

Ngữ pháp:

Lệnh1
Lệnh2
.
.
Lệnhn

Ví dụ

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

Giải

Ta có thể thực hiện công việc trên bằng mẫu chương trình sau:

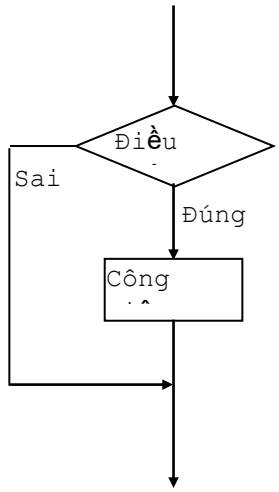
```
XOR AX, AX ;      tổng tại AX lúc đầu là 0.
ADD   AX, BX      ;      cộng thêm b.
ADD AX, CX ;      cộng thêm c.
SHL AX, l         ;      nhân đôi kết quả trong AX.
RA:               ;      lối ra của cấu trúc.
```

- **Cấu trúc IF - THEN**

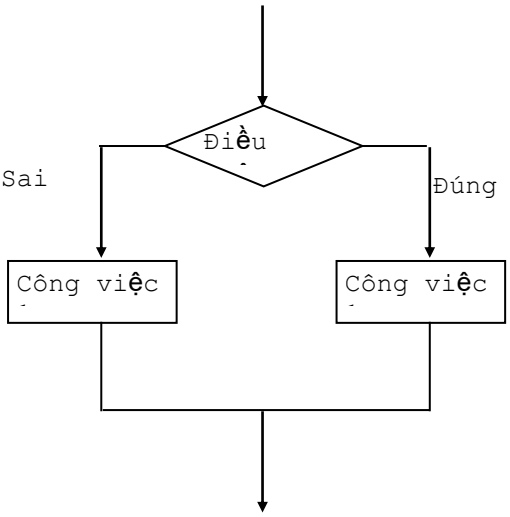
Ngữ pháp (hình 4.2):

IF **Điều kiện** THEN **công việc.**

Từ ngữ pháp của cấu trúc IF-THEN ta thấy nếu thoả mãn Điều kiện thì Công việc được thực hiện nếu không Công việc sẽ bị bỏ qua. Điều này tương đương với việc dùng lệnh nhảy có điều kiện để bỏ qua một thao tác náo đó trong chương trình hợp ngữ.



Cấu trúc IF-THEN



Cấu trúc IF-THEN-ELSE

Ví dụ

Gán cho BX giá trị tuyệt đối của AX.

Giải

Để thực hiện phép gán BX ← |AX| ta có thể dùng các lệnh sau:

```
        CMP AX,0        ;      AX<0?
        JNL GAN         ;      không, gán luôn.
        NEG AX          ;      đúng. đào dấu, rồi
GAN: MOV BX, AX ;       lối ra của cấu trúc.
```

- **Cấu trúc IF - THEN - ELSE**

Ngữ pháp (hình 4.3):

IF **ĐiềuKiện** THEN **CôngViệc1** ELSE **CôngViệc2**

Từ ngữ pháp của cấu trúc IF-THEN-ELSE ta thấy nếu thoả mãn Điều kiện thì Côngviệc1 được thực hiện nếu không thì Côngviệc2 được thực hiện. Điều này tương đương với việc dùng lệnh nhảy có điều kiện và không điều kiện để nhảy đến các nhãn nào đó trong chương hợp ngữ.

Ví dụ

Gán cho CL giá trị bit dấu của AX.

Giải

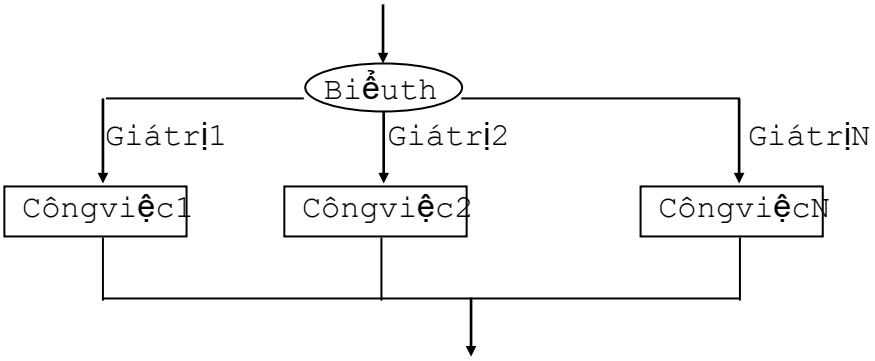
Ta có thể thực hiện các công việc trên bằng mẩu chương trình sau:

```
QR AX, AX        ;     AX>0?.
JNS DG           ;     đúng.
MOV CL, l        ;     sai, cho CL ← 1 rồi
JMP RA           ;     di ra.
DG: XOR CL, CL   ;     cho CL ← 0.
RA:              ; lối ra của cấu trúc.
```

- **Cấu trúc CASE**

Ngữ pháp

CASE **Biểuthức**
Giátrị1: Côngviệc1
Giátrị2: Côngviệc2
. . .
GiátrịN: CôngviệcN
END CASE



Cấu trúc lệnh CASE

Từ ngữ pháp của cấu trúc ta thấy nếu Biểuthức có Giátrị1 thì Côngviệc1 được thực hiện. nếu Biểuthức có Giátrị2 thì Côngviệc2 được thực hiện và ... Điều này tương đương với việc dùng các lệnh nhảy có điều kiện và nhảy không điều kiện để nhãy các nhãn nào đó trong chương trình hợp ngữ. Cấu trúc CASE có thể thực hiện bằng các cấu trúc lựa chọn lồng nhau.

Ví dụ

Dùng CX để biểu hiện các giá trị khác nhau của AX theo quy tắc sau:

Nếu AX < 0 thì CX =-1,

nếu AX = 0 thì CX =0,

nếu AX > 0 thì CX =1.

Giải

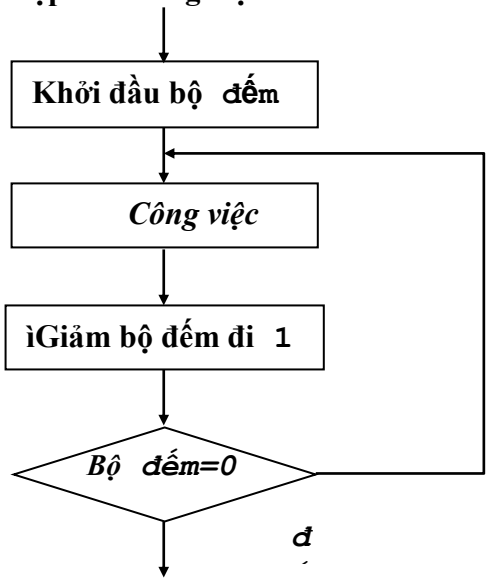
Ta có thể thực hiện các công việc trên bằng mẫu chương trình sau:

```
        CMP AX, 0     ; Kiểm tra dấu của AX.
        JL AM ; AX<0.
        JE KHONG      ; AX  =0.
        JG DUONG      ; AX > 0.
AM: MOV CX, -1
        JMP RA
DUONG: MOV CX, 1
        JMP RA
KHONG: XOR CX. CX
RA:                   ; lối ra của cấu trúc.
```

• **Cấu trúc lặp FOR - DO**

Ngữ pháp

FOR **Số lần lặp** DO **Công việc**



Cấu trúc lặp FOR - DO .

Từ ngữ pháp cuả cấu trúc FOR - DO ta thấy ở đây Công việc được thực hiện lặp đi lặp lại tất cả Số lần lặp lại. Điều này hoàn toàn tương đương với việc dùng lệnh LOOP trong hợp ngữ để lặp lại CX lần một Công việc nào đó, đương nhiên trước đó ta phải gán Số lần lặp cho thanh ghi CX.

Ví dụ Hiển thị một dòng kí tự '$' trên màn hình.

Giải

Một dòng màn hình trên máy IBM PC chứa được nhiều nhất là 80 kí tự.

Ta sẽ sử dụng hàm 2 của ngắt 21H để hiển thị 1 kí tự. Ta phải lặp lại công việc này 80 lần cả thảy bằng lệnh LOOP. Muốn dùng lện này, ngay từ đầu ta phải nạp vào thanh ghi CX số lần hiển thị, nội dung của Cx được tự động giảm đi1 do tác động của lệnh LOOP.

Sau đây là mẩu chương trình thực hiện các công việc trên:

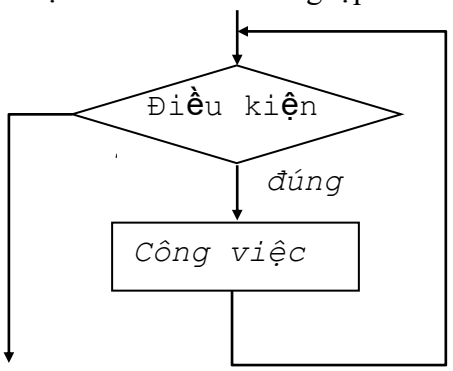
```
        MOV CX, 80  ; số lần hiện thị trong cx
        MOV AH, 2       ; AH chứa số hiệu hàm hiện thị,
        MOV DL, '$'     ; DL chứa kí tự cần hiện thị,
HIEN:   INT 21H         ; hiện thị
        LOOP HIEN       ; cả một dòng kí tự.
RA:                     ; lối ra của cấu trúc.
```

• **Cấu trúc lặp WHILE - DO**

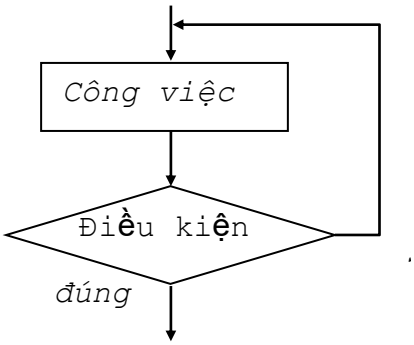
Ngữ pháp

WHILE **Điều kiện** DO **Công việc**

Từ ngữ pháp của cấu trúc **WHILE - DO** ta thấy: Điều kiện được kiểm tra đầu tiên. Công việc được lặp đi lặp lại chừng nào Điều kiện còn đúng. Điều này trong hợp ngữ hoàn toàn tương đương với việc dùng lệnh CMP để kiểm tra Điều kiện và sau đó dùng lệnh nhảy có điều kiện để thoát khỏi vòng lặp.



Cấu trúc **WHILE - DO**



Cấu trúc **REPEAT - UNTIL**

Ví dụ

Đếm số ký tự đọc được từ bàn phím, khi gặp ký tự CR thì thôi.

Giải

Ta có thể thực hiện công việc trên bằng mẩu chương trình sau:

```
        XOR CX, CX      ; tổng số ký tự đọc được lúc đầu là 0
        MOV AH, 1       ; hàm đọc ký tự từ bàn phím.
TIEP:   INT 21H         ; đọc 1 ký ự, Al chứa mã ký tự.
        CMP AL, 13      ; đọc được CR?
        JE RA       ; đúng, ra.
        INC CX          ; sai, thêm 1 ký tự vào tổng.
RA:                     ; lối ra của cấu trúc.
```

• **Cấu trúc lặp REPEAT - UNTIL**

Ngữ pháp

REPEAT **Công việc** UMTIL **Điều kiện**

Từ ngữ pháp của cấu trúc **REPEAT - UNTIL** ta thấy: Công việc được thực hiện đầu tiên. Điều đó có nghĩa là công việc được thực hiện ít nhất một lần . Điều kiện được kiểm tra sau đó. Công việc được lặp đi lặp lại cho tới Điều kiện được thoả mãn. Điều này trong hợp ngữ hoàn toàn tương đương với việc dùng lệnh CMP để kiểm tra Điềukiện và sau đó dùng lệnh nhảy có điều kiện để thoát khỏi vòng lặp.

Ví dụ:

Đọc ký tự từ bàn phím cho tới khi gặp '$' thì thôi.

Giải

Ví dụ này chỉ làm một phần công việc của ví dụ trước. Tại đây ta chỉ phải đọc các ký tự đọc được.

Ta có thể tực hiện công việc trên bằng mẩu chương trình sau:

```
        MOV Ah, 1    ; hàm đọc ký tự bàn phím.
TIEP: INT 21H        ; đọc 1 ký tự.
        CMP AL, '$'  ; đọc được đôla ?
RA:                  ; lối ra của cấu trúc.
```

4.3. Một số chương trình cụ thể:

Trong phần này ta sẽ xét một số chương trình cho các ứng dụng cụ thể, thông qua các ví dụ này ta có thể học được các lệnh, cách lập chương trình cùng với cách tổ chức dữ liệu để giải quyết các bài toán cụ thể. Một số chương trình liên quan đến các vấn đề khác chưa được đề cập đến từ trước đến nay có thể được nêu ra ở những chương tương ứng sau chương này.

Trước khi giới thiệu các ví dụ ta hệ thống lại một vài hàm của các loại ngắt có trong máy IBM PC với hệ điều hành MS DOS hay chưa được dùng trong các ví dụ đã nêu trước đây và sau này.

Ngắt INT 20H dành riêng để kêt thúc chương trình loại . COM
Hàm 1 của ngắt INT 21H: đọc 1 ký tự từ bàn phím
Vào: AH = 1
Ra: AL = mã ASCH của ký tự cần hiện thị
Al = 0 khi ký tự gõ vào là từ các phím chức năng
Hàm 2 của ngắt INT 21H: hiện 1 ký tự lên màn hình
Vào: AH = 2
DL = mã ASCH của ký tự cần hiện 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 cần hiện thị.
Hàm 4CH của ngắt INT 21H: kết thúc chương trình loại . EXE
Vào: AH = 4CH

Một điều cần nhắc lại lần nữa để lưu ý khi đọc các ví dụ

Dấu \\\\\\\\ (nếu có) đặt trước một ví du là để cảnh báo rằng ví dụ liên quan chỉ dùng để mô tả thuật giải cho vấn đề nào đó mà không chạy được trên các máy IBM PC hoặc tương thích.

- **Các ví dụ:**

Ví dụ 1

Trong phần đầu của chương trình hợp ngữ ta có giúo thiệu một chương trình hiện lời chào băng tiếng Anh "Hello". Bây giờ ta phải thêm một lời chào bằng tiếng Việt không dấu "Chao ban" nằm cách lời chào "Hello" trước đây một số dòng nhất định nào đó.

Giải

Ta cũng vẫn sử dụng phương pháp đã được dùng ở chương trình mẫu trước đây để hiện thị lời chào 'tây', hiện các dòng giãn cách và hiện lời chào 'ta'.

Trong ví dụ này ta cũng bỏ bớt đi các dòng cách ở đầu và cuối để chương trình đỡ rườm rà.

```
. Model Small
. Stack 100
. Data
        CRLF        DB      13, 10, '$'
        Chao tay    DB      'hello!$'
        ChaoTa      DB      'Chao ban!$'
. Code
MAIN Proc
                ; khởi đầu thanh ghi DS
        MOV AX, @ Data
        MOV DS, AX
                ; hiện thị lời chào dùng hàm 9 của INT 21H
        MOV AH, 9
        LEA DX, ChaoTay
        INT 21H
                ; cách 5 dòng dùng hàm 9 của INT 21H
        LEA DX, CELF
        MOV CX, 6    CX chứa số dòng cách +1
LAP: INT 21H
        LOOP LAP
                ; hiện thị lời chào dùng hàm 9 của INT 21H
        LEA DX, ChaoTa
        INT 21H
                ; trở về DOS dùng hàm 4 CH của INT 21H
        MOV AH, 4CH
        INT 21H
MAIN Endp
        END MAIN
```

Trong chương trình trên ta đã dùng thanh ghi CX để chứa số dòng phải giãn cách. Với cách làm này mỗi khi muốn thay đổi số dòng dãn cách giữa 2 lời chào ta và lời chào tây, ta phải gắn giá trị khác cho thành ghi CX. Điều này chắc chắn là không phải thuận tiện đối với người sử dụng chương trình.

Ví dụ 2

Trên cơ sở ví dụ trước, ta phải viết chương trình sao cho số dòng phải giãn cách có thể thay đổi được ngay trong khi chạy chương trình.

Giải

Muốn có số dòng cách thay đổi được theo ý muốn giữa 2 lời chào ta và tây khi chạy chương trình mà khôn phải thay giá trị mới gán cho thanh ghi CX ngay trong chương trinh như ở ví dụ trước, ta cần dùng thêm 1 biến mới để chứa số dòng cách và viết chương trình sao cho mới để chứa số dòng cách và viết chương trình sao cho mỗi khi cho chạy chương trình có thêm phần đối thoại để người sử dụng có thể tùy ý thay đổi giá trị của số dòng giãn cách đó. thay đổi giá trị của số dòng giãn cách đó.

Sau đây là văn bản của chương trình thực hiện công việc trên:

```
. Model Small
```

```
. Stack 100
. Data
        CRLF        DB      13,10,'$'
        ChaoTay     DB      'Hello!S'
        Chaota DB   'Chao ban!S'
.Code
MAIN Proc
                ; khởi đầu thanh ghi DS
        MOV AX, @Data
        MOV DS,AX
                ; hiển thị lời chào dùng hàm 9 của INT 21H
        MOV AH, 9
        LEA DX, ChaoTay
        INT 21H
                ; Cách 5 dòng dùng hàm 9 của INT 21H
        LEA DX, CFLF
        MOV CX, 6           ; CX chứa số dòng cách +1
LAP: INT 21H
        LOOP LAP
                ; hiển thị lời chào dùng hàm 9 của INT 21H
        LEA DX, ChaoTa
        INT 21H
                ; trở về DOS dùng hàm  4CH của INT 21H
        MOV AH, 4CH
        I'NT 21H
MAIN Endp
        END MAIN
```

Trong chương trình trên ta đã dùng thanh ghi CX để chứa số dòng phải giãn cách. Với cách làm này mỗi khi muốn thay đổi số dòng giãn cách giữa hai lời chào ta và lời chào tây, ta phải gán giá trị khác cho thanh ghi CX. Điều này chắc chắn không phải là thuận tiện đối với người sử dụng chương trình.

Ví dụ 2

Trên cơ sở ví dụ trước, ta phải viết chương trình sao cho số dòng phai giãn cách có thể thay đổi được ngay trong khi chạy chương trình.

Giải

Muốn có số dòng cách thay đổi được theo ý muốn giữa 2 lời chào ta và tây khi chạy chương trình mà không phải thay giá trị mới gán cho thanh ghi CX ngay trong chương trình như ở ví dụ trước, ta cần dùng thêm 1 biến mới để chứa số dòng cách và viết chương trình sao cho mỗi khi cho chạy thì chương trình có thêm phần đối thoại để người sử dụng có thể thay đổi giá trị của số dòng giãn cách đó.

Sau đây là văn bản của chương trình thực hiện công việc trên:

```
.Model Small
.Stack 100
.Data
```

```
        CRLF        DB      13,10,'$'
        ChaoTay     DB      'Hello!S'
        ChaoTa      DB      'Chao ban!S'
        Thongbao    DB      'go vao so dong cach:S'
        SoCRLF      DB      ?
.Code
MAIN Proc
        MOV AX, @Data       ; khởi đầu thanh ghi DS
        MOV DS, AX
            ; hiện thông báo dùng hàm 9 của INT 21H
MOV AH, 9
LEA DX, Thongbao
INT 21H
        ; đọc số dòng cách dùng hàm 1 của INT 21H
MOV AH, 1
INT 21H                     ; đọc số dòng cách
AND AL, OFH                 ; đổi ra hệ hai
MOV SoCRLE, AL              ; cất đi
        ; cách 1 dòng dùng hàm 9 của INT 21H
MOV AH, 9
LEA DX, CRLF
INT 21H
        ; hiển thị lời chào dùng hàm 9 của INT 21H
MOV AH, 9
LEA DX, ChaoTay
INT 21H
LEA DX, CFLF
XOR CX, CX
MOV CL, SoCRLE              ; CX chứa số dòng cách
LAP: INT 21H
        LOOP LAP
            ; hiện thị lời chào dùng hàm 9 của INT 21H
        LEA DX, ChaoTa
        INT 21H
            ; trở về DOS dùng hàm 4CH của INT 21H
        MOV AH, 4CH
        INT 21H
MAIN Endp
        END MAIN
```

Trong thí dụ trên có một điều cần chú ý là khi đọc một ký tự từ bàm phím (trong trường hợp cụ thể này thì đó là số dòng cách) ta sẽ thu được trong thanh ghi AL mã ASCII của ký tự (số) đã gõ. Để sử dụng nó trong trường hợp cụ thể như một giá trị số và cất nó tại biến SoCRLF, ta phải biến đổi mã ASCII này thành hệ số hai. Để đổi mã ASCII của một số ra

trị số hoặc ngược lại ta cần nhớ rằng giữa giá trị số và mã ASCII của số đó có một khoảng cách là 30H. ví dụ số 9 có mã ASCII là 39 HH (có thể được viết là "9"), tương tự số 0 có mã ASCII là 30H (có thể được viết là "0")

Như vậy việc biến đổi mã ASCII (giả thiết đã có sẵn trong AL)→ giá trị số có thể thực hiện được bằng một trong các lệnh sau:

+ SUB AL30H
+ AND AL0FH

Tương tự như vậy, việc biến đổi ngược lại từ số hệ hai (thường giả thiết đã có sẵn trong thanh ghi DL) → mã ASCII (để đưa ra hiện lên mãn hình) có thể làm được bằng một trong các lệnh sau:

+ ADD DL30H
+ OR DL30H

Ví dụ 3

Đọc từ bàn phím một số hệ hai (dài nhất là 16 bit), kết quả đọc được để tại thanh ghi BX. Sau đó hiện nội dung thanh ghi BX ra màn hình.

Giải

Công việc của mài này thực chất gồm hai phần, một phần đầu ta phải đọc được số hệ hai và cất nó tại BX, trong phần tiếp theo ta phải đưa được nội dung của thanh ghi BX ra màn hình.

Sau đây là văn bản của chương trình thực hiện công việc trên:

```
.Model Small
.Stack 100
. Data
        TBao DB 'Go vao 1 so he hai (max 16 bit,'
             DB 'CR de thoi):$'
.Code
MAIN proc
        MOV AX, @ Data
        MOV DS, AX
        MOV AH, 9           ; hiện thị thông báo
        LEA DX, TBao
        INT 21H
        XOR   BX,BX         ; BX chứa kết quả, lúc đầu là 0
        MOV AH, 1           ; hàm đọc 1 số từ bàn phím
TIEP: INT 21H
        CMP AL, 13          ; CR?
        JF THOIDOC ; đúng, thôi đọc
        AND AL,OFH          ; không, đổi mã ASCII ra số
        SHL BX, 1           ; dịch trái BX 1 bít để lấy chỗ
        OR BL,AL            ; chèn bít vừa đọc vào kết quả
        JMP TIEP            ; đọc tiếp một ký tự
THOIDOC:MOV CX,16       ; CX chứa số bít của BX
        MOV AH,2            ; hàm hiện ký tự
```

```
        HIEN:XOR DL,DL    ; xoá DL để chuẩn bị đổi
        ROL BX,1          ; đưa bít MSB của BX sang CF
        ADC DL, 30H; đổi giá trị bít đó ra ASCII
        INT 21H           ; hiển thị 1 bít của BX
        LOOP HIEN         ; lặp lại cho đến hết
        MOV AH, 4CH       ; trở về DOS
        INT 21H
MAIN Endp
        END MAIN
```

Chương trình hợp ngữ cho công việc đã nêu được hình thành từ 2 phần, một phần với chức năng đọc và một phần với chức năng hiện thị.

Thuật toán cho phần đọc: đọc một ký tự số, chuyển mã ASCII ra số rồi chèn số đọc được vào BX theo thứ tự từ phải qua trái, lặp lại công việc trên các số khác.

Thuật toán cho phần hiện thị ngược lại so với phần đọc: lấy ra 1 bít của số đó trong BX theo thứ tự từ trái qua phải, đổi số đó ra mã ASCII rồi cho hiện thị nó ra màn hình, lặp lại công việc trên cho các số khác.

Các thuật toán của2 phần trên về cơ bản có thể ứng dụng được cho trường hợp phải đọc và hiện thị số hệ mười sáu hoặc hệ mười.

Một số nhận xét có thể rút ra khi đọc chương trình trên:

+ Lệnh xóa thanh ghi BX là rất cần thiết để sau này khi gõ vào các bít của nó ta không nhất thiết phải gõ đủ 16 bít mà vẫn xác định được giá trị của thanh ghi này.

+ Trong chương trình này ta đã dùng lệnh ROL để quay tròn thanh ghi BX, vì vậy sau khi quay và hiện thị tất cả 16 bít của BX ta vẫn bảo toàn được giá trị của thanh ghi BX lúc đầu. Để so sánh , nếu ở phần trên thay vì lệnh quay ROL ta dùng lệnh dịch SHL thì ta vẫn hiện thị được đúng thanh ghi BX, nhưng sau khi hiện thị xong thì quá trị nguyên thủy của thanh ghi BX, nhưng sau khi hiện thị xong thì giá trị nguyên thuỷ của thanh ghi BX bị mất do quá trình dịch gây nên.

+ Trong chương trình này ta đã dùng lệnh cộng có nhớ ADC một cách rất hiệu dụng để lấy ra 1 bít của thanh ghi BX từ giá trị của cờ CF và đổi luôn được nó ra mã ASCII cần thiết cho việc hiện thị.

Ví dụ 4

Người ta để sẵn trong bộ nhớ 2 số hệ hai, mỗi số này có độ dài 10 byte và được để theo thứ tự MSB...LSB. Ta phải tính tổng của hai số nhiều byte đó, kết quả cũng để trong bộ nhớ.

Giải

Để làm tính cộng với các toán hạng là các số với độ dài nhiều hơn 1 byte như trên ta phải nghĩ ngay đến việc dùng lệnh cộng có nhớ ADC cho các byte nằm bên trái của LSB. Còn để cộng các byte LSB của 2 số, tất nhiên ta phải dùng lệnh cộng bình thường ADD. Để cho việc tổ chức các lệnh trong chương trình được tối ưu, ta thay thế lệnh ADD trong trường hợp cộng 2 LSB này bằng lệnh ADC nhưng với cờ CF = 0.

Xét về mặt tổ chức dữ liệu cho chương trình trong đoạn dữ liệu, ta thấy mảng ô nhớ dùng chứa kết quả cần có độ dài 11 byte để chứa hết được kết quả của phép cộng trong trường hợp tổng quát. Chính vì vậy ta cũng cần tăng độ dài cho các mảng chứa toán hạng, mỗi mảng thêm 1 byte nữa, gán sẵn cho chúng giá trị 0 và đặt phía trước các byte MSB để tạo ra độ dài thống nhất cho các mảng dữ liệu chứa các toán hạng và kết quả của phép cộng. Bằng cách này

trong khi viết chương trình ta có thể đạt được việc quản lý đơn giản hơn đối với các thanh ghi lệch.

Sau đây là chương trình thực hiện công việc nói trên.

```
.Model Small
.Stack 100
.Data
        TBao   DB 'DU lieu bat dau tư day'
        So1    DB 00, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10
        So2    DB 00, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10
        ;xi và yi phải được thay bằng số cụ thể
                ; Sum DB 11 dup (0)
        . Code
        MAIN proc
                MOV AX, @Data
                MOV DS, AX
                MOV ES, AX
                LEA BX, So1 ; BX trỏ vào đầu dãy So1
                LEA SI, So2             ; SI trỏ vào đầu dãy So2
                LEA DI, Sum  ; DI trỏ vào đầu dãy Sum
                MOV CX, 11  ; CX chứa số byte phải cộng
                XOR AL, AL  ; xóa cờ CF và AL
        LAP:MOV AL, 10[BX]      ; lấy 1 byte của So1
                ADC AL, 10[SI]      ; cộng với byte của So2
                MOV 10[DI],AL       ; cất kết quả vào Sum
                DEC BX              ; giảm con trỏ tới các mảng
                DEC SI
                DEC DI
                LOOP LAP            ; lặp lại với các byte khác
                MOV AH, 4CH         ; trở về DOS
                INT 21H
        MAIN Endp
                END MAIN
```

Trong chương trình trên ta không có phần lệnh để hiện thị kết quả của phép cộng, vì vậy ngay cả khi đã thay các giá trị số cụ thể vào các biến So1 và So2 rồi dịch ra tệp chương trình có đuôi.EXE và cho chạy nó, ta cũng không kiểm tra kết quả bằng cách dùng chương trình tìm lỗi Debug (xem thêm phần phụ lục để hiểu được các thao tác cần thiết trong Debug).

Bằng chương trình Debug ta có thể quan sát được các lệnh của chương trình trên được dịch và cất trong bộ nhớ như thế nào, đồng thời ta cũng xem được các mảng dữ liệu của 2 toán hạng và của kết quả được tổ chức ra sao trong bộ nhớ. Lúc này ta mới giải thích lý do tại sao ta có định nghĩa dòng thông báo TBao trong đoạn dữ liệu mà ở bên trong chương trình trên ta đã không hề sử dụng đến nó. Thực chất đây chi là cách dùng một chuỗi ký tự *để đánh dấu* đoạn dữ liệu mà ta quan tâm. Điều này cho phép ta dễ dàng nhận được đoạn dữ liệu mà ta quan tâm. Điều này cho phép ta dễ dàng nhận ra được đoạn dữ liệu mà ta cần quan tâm trong một vùng nhớ chứa dữ liệu mỗi khi ta dùng Debug để xem nội dung của nó.

Bây giờ ta sẽ đề cập đến vấn đề sử dụng Deug cụ thể hơn.

Nếu ta gõ văn bản của chương trình trên rồi đặt tên là tệp ADC.ASM và dịch nó ra tệp ADC.EXE ta có thể dùng lệnh sau để chạy ADC trong Debug:

Debug adc.exe

Trên màn hình sẽ xuất hiện dấu-đó là dấu nhắc của Debug

Tại dấu nhắc đó của Debug nếu ta gõ

-u 0 60

Ta sẽ nhận được một màn hình, trong đó một bên chứa địa chỉ các ô nhớ kèm theo nội dung của các ô đó (các mã lệnh và dữ liệu) và bên kia là các dòng lệnh gợi nhớ tương ứng. Thực ra phần mã lệnh của chương trình kết thúc tại địa chỉ IP=0029H (byte cuối của lệnh INT 21H). Tiếp ngay sau đó chính là phần dành cho các dữ liệu của các toán hạng và kết quả, như những dữ liệu đó lúc này (do ta đang ở chế độ dịch ngược *-unassemble*) đã bị Debug coi như là mã lệnh và được dịch ra thành các lệnh gợi nhớ. Muốn xem các mảng dữ liệu này ta gõ:

-d CS:0 60

Ta sẽ thấy mã lệnh và dữ liệu có tại một địa chỉ nào đó được hiện lên theo 2 kiểu: bằng các số hệ mười sáu (phần bên trái) và bằng các ký tự ASCII (phần bên phải). Chính trên phần hiện thị ở bên phải này ta dễ dàng nhận thấy dòng ký tự dùng để đánh dâu mà ta đã nói đến trước đây ('Du lieu bat dau tu day'). Nếu lấy hàng chữ này làm mốc và nhìn sang phần hiện thị dưới dạng số hệ mười sáu ở bên trái (bắt đầu từ địa chỉ lệch 002AH) ta dễ dàng ta tìm ra vị trí và nội dung của các byte dữ liệu (là các toán hạng) và kết quả (kết quả được tạm gán bằng 0 lúc đầu).

Để xem chương trình này cho kết quả ra sao ta phải cho nó chạy trong môi trường Debug. Muốn vậy ta có thể gõ

-g 50

Sau khi nhận được từ mấy câu trả lời '*program terminated normally*' -chương trình kết thúc bình thường (không lỗi), ta gõ tiếp

-d CS: 0 50

Để xem kết quả của chương trình sau khi chạy được cất giữ trong bộ nhớ ra sao.

Sau đây là một trường hợp của chương trình trên với các dữ liệu cụ thể.

Nếu ta gõ vào các byte của các toán hạng viết dưới dạng các số hệ mười sáu:

So1 = 00H 80H 81H 82H 83H 80H 80H 80H 83H 84H 85H và

So2 = 00H 80H 81H 82H 83H 80H 80H 80H 83H 84H 85H

Thì khi cho chạy chương trình ta sẽ nhận được.

Sum = 01H 01H 03H 05H 07H 01H 01H 01H 07H 09H 0AH

Ta có thể tính 'bằng tay' để kiểm chứng lại kết quả trên.

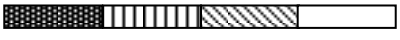
Cuối cùng để ra khỏi Debug ta gõ

- q

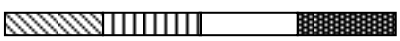
Qua một vài thao tác sơ bộ trong ví dụ trên ta cũng đã thấy được phần nào sự tiện lợi của chương trình Debug trong việc cho chạy và thử nghiệm các chương trình hợp ngữ đơn giản

Ví dụ 5

Trong thanh ghi BX có sẵn 4 số hệ mười sáu, mỗi số được biểu diễn bằng 1 ô màu:



Hãy lập trình để biến đổi thanh ghi BX thành:



(ví dụ: nếu như lúc đầu thanh ghi BX chứa giá trị 1234H thì sau khi biến đổi, BX sẽ chứa giá trị 3241H.v.v...)

Giải

Thực chất đây là kiểu bài toán cụ thể này, sau khi xem xét dạng thức của thanh ghi BX trước và sau khi biến đổi, ta thấy có thể thu được kết quả môt cách rất đơn giản bằng cách quay trái thanh ghi BX nguyên gốc đi 12 bít rồi sau đó quay tiếp thanh ghi BH đi 4 bít là xong.

Sau đây là chương trình thực hiện công việc trên.

```
.Model Small
.Stack 100
.Code
MAIN Proc
        MOV CL,12
        ROL BX,CL           ; quay BX đi 12 bít
        MOV CL, 4
        ROR BH, CL  ; tráo 4 bít thấp và cao của BH
        MOV AH, 4CH         ; trở về DOS
        INT 21H
MAIN  Endp
        END MAIN
```

Ví dụ 6

Có một chuỗi ký tự trong bộ nhớ. Hãy tạo ra một bản sao của chỗi trên rồi cất trong bộ nhớ.

Giải

Để giải bài toán loại này có thể ứng dụng các lệnh chuyển chuỗi. Sau đây là cách tổ chức dữ liệu và chương trình cho bài toán trên với độ dài chỗi là 8 byte

```
.Model Small
.Stack 100
.Data
        Str1    DB 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'
        Tbao    DB "bản sao của chỗi gốc:', 10,13
        Str2    DB 8 DUP (' ')
                DB '$'
. Code
MAIN proc
        MOV AX, @Data           ; khởi đầu cho DS và ES
        MOV DS, AX
        MOV ES, AX
        LEA SI, Str1            ; SI chỉ vào chuỗi gốc.
        LEA DI, Str2            ; DI chỉ vào chuỗi đích.
        CLD                     ; định hướng tiến
        MOV CX, 8               ; CX chứa số byte phải cóp
```

```
        REP MOVSB          ; cóp sang chỗi đích
        LEA DX, Tbao               ; chuẩn bị hiển thị bản sao
        MOV AH, 9
        INT 21H
        MOV AH, 4CH                ; về DOS
        INT 21H
MAIN Endp
        END MAIN
```

Ví dụ 7

Có một chuỗi ký tự thường trong bộ nhớ. Hãy tạo ra một chuỗi ký tự chữ hoa từ chuỗi trên rồi cất chuỗi đó trong bộ nhớ.

Giải:

Ví du này và ví dụ trước khi khác nhau chút ít trong việc xử lý các ký tự của chuỗi, vì vậy phần trên các lệnh có tính chất chuẩn bị trước và sau các thao tác với chuỗi có thể coi là như nhau. Để giải bài toán này có thể ứng dụng các lệnh LODSB và STOSB với chuỗi đã cho. Thuật toán là:

+ Lấy từng ký tự của chuỗi gốc (cũ) bằng lệnh LODSB,
+ Biến đổi thành chữ hoa bằng cách trừ đi 20H,
+ Cất ký tự đã biến đổi vào chuỗi đích (mới) bằng lệnh STOSB.

Sau đây là cách tổ chức dữ liệu và chương trình cho bài toán trên với độ dài chuỗi là 8 byte. Để minh hoạ một cách thao tác khác so với cách ở ví dụ trước trong ví dụ này là dùng cách thao tác lùi đối với chuỗi ký tự.

```
.Model Small
.Stack 100
.Data
        Str1    DB  'a' , 'b' , 'c' , 'd' , 'e' , 'f' , 'g' , 'h'
        Tbao    DB  'chuỗi đã được đổi: ' , 10,13
                DB  '$'
.Code
MAIN Proc
        MOV   AX, @Data    ; khởi đầu đầu cho DS và ES
        MOV   DS, AX
        MOV   ES, AX
        LEA   SI, Str1+7   ; SI chỉ vào cuối chuỗi cũ
        LEA   DI, Str2+7   ; DI chỉ vào cuối chuỗi mới
        STD                ; định hướng lùi
        MOV   CX, 8        ; CX chứa số byte phải đổi
LAP :   LODSB              ; lấy 1 ký tự của chuỗi cũ
        SUB AL, 20H ; đổi thành chữ hoa
        STOSB              ; cất vào chuỗi mới
        LOOP LAP           ; làm cho đến hết
        LEA DX, Tbao       ; chuẩn bị hiện chuỗi mới
        MOV   AH, 9
```

```
        INT     21H
        MOV   AH, 4CH       ; về DOS
        INT     21H
MAIN Endp
        END MAIN
```

Ví dụ 8

Cho 1 mảng gồm các phần tử kiểu byte. Hãy sắp xếp chúng theo thứ tự lớn dần.

Giải:

Đây là bài toán kinh điển rất hay gặp trong khi học lập trình với các ngô ngữ khác nhau. Thuật giải của nó vì thế cũng khá quen thuộc: tìm phần tử lớn nhất (max) trong dãy và xếp phần tử đó vào cuối dãy, tiếp tục làm như vậy với các phần tử còn lại (trừ phần tử vừa được tìm thấy và được xếp vào cuối dãy).

Để minh họa thuật toán trên nhưng với dữ liệu cụ thể, ta giả thiết chuỗi cần sắp xếp là chuỗi của các ký tự ASCII gồm 10 phần tử. Tất nhiên, chương trình vẫn chạy đúng khi tư thay các phần tử của mảng trên bằng các số không dấu có độ lớn biểu diễn được trong phạm vi 1 byte. Tại đây ta tổ chức chương trình thành 1 chương trình chính và 1 chương trình con. Chương trình con có tên là DOICHO được gọi ra mỗi khi cần đổi chỗ 2 phần tử của dãy.

Sau đẩy là chương trình thực hiện công việc trên.

```
. Model Sall
. Stack 100
. Data
   Tbao DB 'chuỗi đã sắp xếp:' , 10, 13
   MGB DB 'a' , 'Y' , 'G' , 'T' , 'y' , 'Z' , 'U' , 'B' , 'D' , 'E'
        DB '$'
. Code
MAIN Proc
        MOV AX, @Data           ; khởi đầu DS
        MOV DS, AX      ; BX: số phần tử của mảng
        LEA DX, MGB             ; DX chỉ vào đầu mảng byte
        DEC BX                  ; số vòng so sánh phải làm
LAP : MOV SI, DX                ; SI chỉ vào đầu mảng
        MOV CX, BX      ; CX số lần so của vòng so
        MOV DI, SI              ; giả sử phần tử đầu là max
        MOV AL, {DI}            ; AL chứa phần tử của max
TIMMAX: INC SI          ; chỉ vào phần tử bên cạnh
        CMP {SI}, AL            ; phần tử mới > max ?
        JNG TIEP                ; không, tim max
        MOV DI, SI              ; đúng, DI chỉ vào max
        MOV AL, {DI}            ; AL chứa phần tử max
TIEP: LOOP TIMMAX               ; tìm max của một vòng so
        CALL DOICHO             ; đổi chỗ max - số mới
        DEC BX                  ; số vòng so còn lại
        JNZ LAP                 ; làm tiếp vòng so mới
```

```
        MOV AH, 9                ; hiển thị chuỗi đã sắp xếp
        LEA DX, Tbao
        INT 21H
        MOV AH, 4CH              ; về DOS
        INT 21H
MAIN End

DOICHO Proc
        PUSH AX
        MOV AL, {SI}
        XCHG AL, {DI}
        MOV {SI}, AL
        POP AX
        RET
DOICHO Endp
        END MAIN
```

Sau khi dịch và cho chạy chương trìnhtrên ta thu được các dòng sau:

chuoi da sap xep:

B, D, E, G, T, U, Y, Z, a, y

Ví dụ 9

Đọc vào một hàng (max 80) ký tự rồi hiển thị nó theo thứ tự ngược lại

Giải

Thuât toán cho việc đọc và hiển thị một hàng ký tự không có gì là phức tạp. Để có thể hiển thị các ký tự theo thứ tự ngược lại so với khi đọc một cách thuận tiện và tránh được các thao tác rườm rà, ta có thể nghĩ ngay đến việc ứng dụng nguyên tắc làm việc "vào trước ra sau" của ngăn xếp: khi đọc được các ký tự ta cất chúng lần lượt vào ngăn xếp và sau đó lại lấy lần lượt các ký tự trên ra để hiển thị.

Sau đây là chương trình thực hiện công việc trên.

```
. Model Small
. Stack 100
. Data
    Tbao1 DB 'gõ vào một dãy ký tự: S'
    Tbao2 DB 13, 10, 'dãy ký tự xếp ngược lại: S'
. Code
MAIN Rroc
        MOV AX, @Data
        MOV DS, AX
        MOV AH, 9
        LEA DX, Tbao1
        INT 21H                  ; hiện Tbao1
        XOR CX, CX       ; CX: đếm số ký tự đọc được
        MOV AH, 1
DOCTIEP: INT 21H                 ; đọc 1 ký tự
LAP :   CMP AL, 13               ; CR ?
```

```
            JE THOIDOC          ; đúng, hiển thị Tbao2
            PUSH AX                     ; không, cất nó vào ngăn xếp
            INC CX                      ; cập nhật bộ đếm số ký tự
            IMP DOCTIEP
THOIDOC:LEA DX, Tbao2
            MOV AH, 9
            INT 21H                     ; cách 1 dòng và hiện Tbao2
            MOV AH, 2                   ; chuẩn bị hiển thị ký tự
      HIEN: POP DX                      ; lấy ký tự ra từ ngăn xếp
            INT 21H
            LOOP HIEN                   ; hiện dòng ký tự xếp ngược
            MOV AH, 4CH                 ; về DOS
            INT 21H
      MAIN Endp
            END MAIN
```

Ví dụ 10

Ta phải đọc vào 1 dòng ký tự từ bàn phím, kết thúc bằng Enter. Sau đó ta phải đếm số nguyên âm và số phụ âm có trong dãy ký tự đó rồi hiển thị các số đó.

Giải

Trong bài toán này công việc đọc vào một dãy ký tự từ bàn phím là rất đơn giản. Chỉ còn phải nói qua về thuật toán để tìm các nguyên âm và phụ âm: ta phải tổ chức dữ liệu thành nhóm các nguyên âm và nhóm các phụ âm và sẽ so sánh các ký tự đọc được với các thành phần của mỗi nhóm này. Mỗi khi tìm được ký tự các phần tử người ta thường dùng lệnh kiểu REPNE SCASB.

Sau đây là chương trình NAMPAMASM thực hiện công việc trên.

```
; tính số nguyên âm và phụ âm trong một dòng ký tự
      .Model Small
      .Stack 100
      .Data
         Str     DB 80 dup (0)
         NAM DB 'AaEeIiOoUuYy'
         Pam' DB 'BCDFGHJKLMNPQRSTVWXZ'
               DB 'bcdfghjklmnpqrstvwxz'
         Tbao1 DB 13, 10, 'so ky tu la nguyen am = s'
         Tbao2 DB 13, 10, 'so ky tu la phu am = s'
         Tbao3 DB 'go vao 1 xau ky tu la phu am = '$'
         SoNAM DW 0
         SoPAM DW 0
      .Code
      MAIN Proc
            MOV AX, @Data         ; khởi đầu cho DS và ES
            MOV DS, AX
```

```
        MOV ES, AX
        LEA DI, Str             ; DI chỉ vào chuỗi
        CALL DOCXAU             ; đọc chuỗi, số ký tự tại BX
        MOV SI, DI              ; SI chỉ vào chuỗi vừa đọc
        CLD                     ; định hướng tiến
LAMLAI:
        LODSB                   ; lấy 1 phần tư của chuỗi
        LEA DI, NAM             ; DI chỉ vào chuỗi ng.âm
        MOV CX, 12      ; 6 nguyên âm, 2 loại chữ
        REPNE SCASB             ; ký tự  là nguyên âm ?
        JNE PHUAM       ; không, đó là phụ âm
        INC SoNAM               ; tăng số nguyên âm
        JMP TIEP
PHUAM :
        LEA DI, PAM     ; DI chỉ vào chuỗi phụ âm
        MOV CX, 40      ; 20 phụ âm, 2 loại chữ
        REPNE SCASB             ; ký tự là phụ âm ?
        JNE TIEP                ; không
        INC SoPAM               ; tăng số phụ âm
TIEP:   DEC BA                  ; số ký tự phải kiểm tra
        JNE LAMLAI      ; làm lại
        MOV AH, 9               ; chuẩn bị hiển thị Tbao1
        LEA DX, Tbao1
        INT 21H
        MOV AX, SoNAM           ; lấy số nguyên âm
        CALL OUTSO              ; hiển thị số phụ âm
        MOV AH, 4CH             ; về DOS
        INT 21H
MAIN Endp
DOCXAU Proc
; đọc và lưu một chuỗi ký tự, kết thúc bằng Enter
; Vào:          DI chứa địa chỉ lệch của chuỗi
; Ra:           DI chứa địa chỉ lệch của chuỗi
                BX chứa số ký tự đọc được
        PUSH DI                 ; cất con trỏ
        CLD                     ; định hướng tiến để cất
        XOR BX, BX      ; xóa bộ đếm số ký tự
        MOV AH, 1               ; chuẩn bị đọc một ký tự
DOCTIEP :
        INT 21H
        CMP AL, 13              ; CR ?
        JE THOI                 ; đúng, kết thúc
        CMP AL, 8               ; BS ? (xóa trái)
```

```
        JNE LUU             ; không, cất
        DEC DI              ; đúng, đổi lại con trỏ
        DEC BX              ; bớt đi một ký tự
        JMP DOCTIEP         ; đọc tiếp
LUU:    STOSB               ; cất ký tự
        INC BX              ; cập nhật số ký tự đã đọc
        JMP DOCTIEP         ; đọc tiếp
THOI:   POP DI              ; lấy lại con trỏ
        RET                 ; về CTC
DOCXAU Endp

OUTSO Proc
        ; hiển thị số trong AX
        ; Vào: AX
        MOV BL, 10          ; dùng BL để chứa số chia
        XOR CX, CX          ; CX: bộ đếm số lần hiện
LAP:    DIV BL              ; AL: thượng, AH: dư
        PUSH AX             ; cất số dư
        INC CX              ; cập nhật số lần
        XOR AH, AH          ; chuẩn bị chia tiếp
        OR AL, AL       ; thương = 0?
        JNE LAP             ; chưa, lặp
        MOV AH, 2       ; rồi, chuẩn bị hiển thị
HIEN:   POP DX              ; lấy lại số dư
        OR DH, 30H          ; đổi số dư ra mã ASCII
        MOV DL, DH
        INT 21H             ; hiển thị kết quả
        LOOP HIEN       ; lấy lại số lần phải hiện
        RET                 ; về CTC
OUTSO Endp
        END MAIN
```

So với cách làm từ trước đến nay thì trong phần chương trình đọc ký tự (DOCXAU) ta đã đưa vào thêm một số lệnh để làm cho việc xử lý các ký tự gõ từ bàn phím được mềm dẻo hơn: ta có thể loại bỏ ký tự gõ nhầm bằng cách dùng phím BS (Back space, xóa trái). Tuy nhiên, trên màn hình ta không thấy được hiệu ứng xóa trái thật sự của BS vì chương trình mà ta viết ở đây vẫn còn quá đơn giản để có thể xử lý được vấn đề khá phức tạp này. Muốn làm được điều vừa nêu ta sẽ phải viết một chương trình phức tạp hơn trong đó phải sử dụng đến các hàm khác của ngắt 21H.

Tại chương trình này ta cũng thấy được cách làm ra các câu chú giải cho phần mào đầu của các chương trình con. Thông thường trong các chú giải kiểu này ta phải nêu lên được một cách vắn tắt các vấn đề sau:

+ Nhiệm vụ của chương trình con

+ Các thanh ghi tham gia vào việc truyền tham số của chương trình khi vào hoặc ra chương trình đó.

Nhờ phần mào đầu này của mỗi chương trình con ta có thể thu được các thông tin sơ bộ về bản thân chương trình con được sử dụng. Ví dụ như trong chương trình con DOCXAU, ta có thể thấy được các thanh ghi nào tham gia vào việc truyền tham số giữa chương trình chính và chương trình con. Vấn đề truyền tham số giữa các chương trình cũng là vấn đề mà ta sẽ đề cập kỹ hơn ở phần sau.

Ví dụ 11

Lập một chương trình trễ với thời gian trễ 1ms dưới dạng chương trình con để có thể sử dụng nó trong các trường hợp cần thiết.

Giải

Chương trình trễ thời gian có rất nhiều ứng dụng trong thực tế: từ việc tạo ra xung âm thanh với các thanh điệu khác nhau cho đến việc điều khiển các đại lượng vật lý trong một hệ thống vi xử lý - điều khiển ứng dụng trong công nghiệp. Để giải quyết vấn đề này, thông thường ta lập ra một chương trình trong đó có chứa một cấu trúc lặp, bên trong cấu trúc lặp đó chứa các lệnh mà thời gian để chạy mỗi lệnh đó có thể xác định được từ các tài liệu tra cứu. Số lần lặp của vòng lặp thường được chứa trong một thanh ghi để có thể thay đổi được. Một chương trình như như vậy thường phụ thuộc rất nhiều vào phần cứng của hệ thống (chí ít cũng phụ thuộc vào loại CPU và tần số của đồng hồ nhịp cho nó).

Trong thí dụ sau đây, để cho công việc tính toán được đơn giản, ta giả thiết CPU là loại 8088-5 MHz (T_{clk} = 0.200μs). Một chương trình viết dưới dạng thủ tục để tạo thời gian trễ 1ms có thể như sau:

```
TRE1ms Proc
; chương trình trễ 1ms
        ; CLI                    ; cấm các ngắt
        PUSH  CX                 ; cất CX
        MOV CX, 292              ; xem cách tính '292' ở dưới
LAP:;   NOP                      ; có NOP để tăng thời gian
        LOOP LAP
        POP CX                   ; lấy lại CX
        ; STI                    ; cho phép ngắt trở lại
        RET
TRE1ms Endp
```

Trong chương trình con nói trên, số lần lập để trong CX phải được tính toán sao cho thời gian chạy toàn bộ chương trình với các vòng lặp kéo dài xấp xỉ 1ms. Ta nói xấp xỉ là vì trong thực tế khi CPU làm việc, nó còn có thể còn bị treo để cho quá trình làm tươi bộ nhớ được hoàn tất (nếu hệ thống sử dụng RAM động). Thêm vào đó, vì số lần lặp chứa trong CX phải là số nguyên nên kết quả từ các tính toán thực cần phải được làm tròn cho phù hợp, từ đó gây ra sai số.

Trong một hệ vi xử lý hoạt động với nhiều nguồn gây ngắt, đôi khi để tạo ra một khoảng thời gian trễ xác định và không chịu tác động của các yêu cầu ngắt, ta còn phải có các lệnh cấm/cho pháp các yêu cầu ngắt trước/sau khi chạy phần lệnh tính thời gian (như ở trong ví dụ trên, ta có thể bỏ các dấu chấm phẩy'.' đứng trước các lệnh CLI và STI để đạt được điều này).

Bây giờ ta sẽ nói qua về cách tính giá trị số phửi đưa vào thanh ghi CX trong thí dụ trên (không tính đến các lệnh CLI.NOP và STI)

Theo tài liệu tra cứu dành cho tập lệnh của CPU 8088, ta có được thời gian cho các lệnh liên quan trong chương trình trên tính theo đơn vị T_{clk} như sau:

Lệnh	Số chu kỳ
PUSH	11
POP	8
MOV	4
LOOP	17 (hoặc 5 trong trường hợp không lặp)
RET	16

Nếu ta gọi số cần tìm để đưa vào thanh ghi CX là N, ta sẽ thu được N khi giải phương trình sau:

$$T_{clk}^{*} (11 + 8 + 4 + 16 +(N-1)*17 + 5) = 1 \text{ ms}$$

Thay số vào ta được:

$$34 + 17 * N = 5000 \text{ hay } N = 5000/17-2$$

Từ đó rút ra $N \approx 292$, đây chính là giá trị phải đưa vào thanh ghi CX ở trên.

Nhận xét phương trình cuối cùng ta thấy thuời gian mà vòng lặp phải "giết" lớn hơn rất nhiều so với thời gian dành cho các lệnh khác, vì vậy ta có thể tính được N với sai số không lớn lắm khi lấy $N \approx 5000/17 \approx 294$.

Trong ví dụ trên ta mới chỉ dùng đến một giá trị nhỏ mà thanh ghi CX có khả năng biểu diễn. Nếu ta thay giá trị lớn nhất CX = 65535 thì có thể đạt được thời gian trễ lớn nhất khoảng 224 ms.

Trong các máy IBM PC các chương trình trễ thời gian đạt độ chính xác cao thường được thực hiện bằng cách dùng chương trình để đếm các xung thu được từ các mạch định thời gian (timer) có sẵn trong máy. Các mạch định thời gian này sẽ có xung đầu vào rất chính xác được lấy từ các xung đồng hồ của hệ thống.

4.5. Vấn đề truyền tham số giữa các chương trình

Trong khi lập trình cho các ứng dụng cụ thể, một vấn đề rất đáng quan tâm là cách truyền tham số giữa chương trình chính (CTC) và chương trình con (cyc) hoặc giữa các môđun chương trình với nhau (khi dùng kỹ thuật lập trình kiểu môđun mà ta không xem xét ở đây).

Nói chung trong thực tế người ta thường dùng các cách truyền thông số sau:

+ Truyền tham số qua thanh ghi
+ Truyền tham số qua ô nhớ - biến
+ Truyền tham số qua ô nhớ có địa chỉ do thanh ghi chỉ ra
+ Truyền tham số qua ngăn xếp

Sau đây là một số ví dụ về việc truyền tham số giữa các chương trình.

Ví dụ 1: truyền tham số qua thanh ghi

Cách truyền tham số thông qua thanh ghi trong các ngôn ngữ khác còn được gọi là truyền bằng giá trị.

Ví dụ trình bày một chương trình dọc và hiển thị một hệ số mười sáu gồm 2 phần được tổ chức theo kiểu CTC và ctc. CTC gọi 1 ctc để đọc vào một số hệ mười sáu rồi cất nó trong thanh ghi BX và gọi tiếp 1 ctc khác lấy số vừa đọc cất tại BX để hiện thị ra màn hình; nói khác đi, 2 phần của chương trình truyền tham số với nhau thông qua nội dung của thanh ghi BX. Các chương trình con được gắn vào sau chương trình chính nhờ lệnh giả INCLUDE.

Sau đây là avưn bản chương trình thực hiện công việc trên.

Tệp CHINH.ASM (chứa CTC):

```
.Model Small
.Stack 100
.Code
MAIN Proc
        CALL INHEX          ;vào số hệ hex, kết quả ở BX
        MOV AH, 2                   ;cách 1 dòng
        MOV DL, 13
        INT 21H
        MOV DL, 10
        INT 21H
        CALL OUTHEX                 ;hiện thị kết quả có tại BX
        MOV AH, 4CH                 ;về DOS
        INT 21H
MAIN Endp

        INCLUDE     vao.asm; vao.asm chứa INHEX
        INCLUDE     ra.asm   ; ra.asm chứa INHEX
        ;các ctc này nằm trong cùng thư mục với CTC
        END MAIN
```

Tiếp theo đây là tệp VAO.ASM chứa thủ tục INHEX và tập RA.ASM chứa thủ tục OUTHEX. Các thủ tục này được gọi trong chương trình CHINH.ASM đã nói ở trên.

Tệp VAO.ASM và RA.ASM phải được để trong cùng một thư mục với CHINH.ASM, nếu không ta phải ghi cả đường dẫn đầy đủ của chúng.

Tệp VAO.ASM (chứa ctc INHEX):

```
INHEX Proc
            ;nhân vào 4 số hệ mười sáu.
            ;ra : BX chứa kết quả
    BDAU:XOR BX, BX    ; xóa BX để chứa kết quả
            MOV CX, 4           ;CL chứa số lần dịch
            MOV AH, 2           ;chuẩn bị hiện dấu nhắc
            MOV DL,'?'
            INT 21H
            MOV AH, 1           ;đọc 1 ký tự
     LAP1:  INT 21H
            CMP AL, 13          ;CR ?
            JE RA               ;đúng, ra
            CMP AL, '0'         ;ký tự >='0' ?
            JNGE SAI            ;không, xử lý sai
            CMP AL'9'           ;ký tự <='9' ?
            JG CHUI             ;không, có thể là chữ
            AND AL, OFH         ;đúng, đổi ra số.
            JMP DICH            ;xử lý tiếp
     CHU1:  CMP AL.'A'          ;ký tự >='A' ?
```

```
            JNLE SAI          ;không, xử lý sai
            CMP AL'F'         ;ký tự <='F' ?
            JNLE SAI          ;không, xử lý sai
            SUB AL, 37H ;đổi ký tự hệ hex ra số
DICH:       SHL BX, CL        ;dịch 4 bit để lấy chỗ
            OR BL, AL         ;chèn số đọc được vào BX
            JMP LAP1          ;đọc tiếp 1 ký tự
RA :        RET               ;về CTC
SAI:        MOV AH, 2
            MOV DL, 13
            INT 21H
            MOV DL, 10
            INT 21H           ;về dầu dòng mới
            JMP BDAU          ;làm lại từ đầu
 INHEX Endp
OUTHEX Proc
            ;hiện thị số hệ mười sáu trong BX
            MOV CX, 4         ;CX: bộ đếm số lần hiện
            MOV AH, 2         ;chuẩn bị hiện kết quả
 LAP2:      PUSH CX           ;cất bộ đếm sốlần hiện
            MOV CL, 4         ;CL: số bit phải quay
            ROL BX, BL        ;của thanh ghi BX
            MOV DL, BL ;chuyển sang DL để xử lý
            AND DL, OFH       ;chỉ quan tâm 4 bit cuối
            CMP DL, 9         ;số hex đó là chữ?
            JG CHU2           ;đúng, đổi chữ ra mã ASCII
            OR DL, 30H        ;không, đổi số ra mã ASCII
            IMP HIEN
 CHU2:ADD DL, 37H
 HIEN:  INT 21H               ;hiển thị kết quả
            POP CX            ;lấy lại số lần phải hiện
            LOOP LAP2         ;hiện cho đến hết thì thôi
            RET               ;về CTC
OUTHEX Endp
```

Chương trình OUTHEX chẳng qua là chương trình OUTSO đã giới thiêu ở phần trước và được thay đổi chút ít cho phù hợp với việc hiện thị số hệ mười sáu.

Trong chương trình trên, để làm cho chương trình đơn giản ta chỉ viết các lệnh để hiện ra dấu?. Với mục đích nhắc nhở người sử dụng gõ vào 4 số hệ mười sáu đúng quy cách. ta định nghĩa các số sau là các chữ số hệ mười sáu đúng quy cách cho chương trình trên.

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Trong khi chạy chương trình sẽ tự động tiến hành kiểm tra việc gõ vào hệ số mười sáu để loại bỏ trường hợp các ký tự không hợp lệ. Nếu người sử dụng gõ vào các ký tự không đúng quy cách thì phải gõ lại từ đầu.

Ví dụ 2: truyền tham số qua ô nhớ biến

Ví dụ này trình bày một chương trình cộng 2 số gồm 2 phần, tổ chức theo kiểu chương trình chính và chương trình con. Hai phần này truyền vào tham số với nhau thông qua ô nhớ dành cho biến (thanh ghi ngoài).

Chương trình của ví dụ này được viết ra với mục đích chính chỉ là để trình bày một cách tổ chức mối liên hệ giữa một chương trình chính và một chương trình con: vì vậy vấn đề mà chương trình cần giải quyết được đặt ra thật đơn giản: cộng 2 số mà tổng của chúng nhỏ hơn 10. Nếu ta gõ vào 2 số không đúng yêu cầu trên thì chương trình không trả lời được và tự thoát ra.

Sau đây là văn bản chương trình thực hiện công việc trên.

```
.Model Small
.Stack 100
.Data
        Tbao1       DB 'Gõ vào 2 số có tổng <10: S'
        Tbao2       DB 13, 10, ' Tổng của '
        So1         DB ?
                    DB ' và '
        So2         DB ?
                    DB ' là '
        SUM         DB -30H, '$'
.Code
   MAIN Proc
        MOV AX, @Data
        MOV DS, AX          ;khởi tạo
        MOV AH, 9
        LEA DX, Tbao1
        INT 21H                     ;hiện Tbao1
        MOV AH, 1                   ;đọc So1
        INT 21H
        MOV Sol, AL         ;cất mã số So1
        MOV DL, ','                 ;dấu phẩy đan xen giữa 2 số
        MOV AH, 2
        INT 21H
        MOV AH, 1                   ;đọc số 2
        INT 21H
        MOV So2, AL         ;cất mã của So2
        CALL ADD2SO                 ;cộng 2 số
        LEA DX, Tbao2
        MOV AH, 4CH                 ;về DOS
        INT 21H
   MAIN Endp

ADD2SO Proc
        MOV AL, So1         ;lấy mã số1
        ADD AL, So2         ;cộng với mã số2
```

```
        ADD SUM, AL             ;đổi ra ASCII và cất đi
        RET
ADD2So Endp
        END MAIN
```

Ví dụ 3: truyền tham số qua ô nhớ địa chỉ cho bởi thanh ghi

Đây là một cách truyền tham số mà trong các ngôn ngữ lập trình khác thường được mang tên là truyền bằng địa chỉ của tham số

Để dễ so sánh các cách truyền tham số với nhau, ta minh họa cách truyền tham số này bằng cách giữ nguyên bài toán của ví dụ trwosc

Các thanh ghi SI, DI và BX được sử dụng để chứa địa chỉ lệch của các tham số cần truyền (còn thanh ghi đoạn dữ liệu ngầm định là DS)

Sau đây là văn bản chương trình thực hiện công việc trên.

```
.Model Small
.Stack 100
.Data
        Tbao1       DB 'Gõ vào 2 số có tổng <10: S'
        Tbao2       DB 13, 10, 'Tổng của '
        So1         DB ?
                    DB ' và '
        So2         DB ?
                    DB ' là '
        SUM         DB -30H, '$'
.Code
   MAIN Proc
        MOV AX, @Data
        MOV DS, AX          ;khởi tạo DS
        MOV AH, 9
        LEA DX, Tbao1
        INT 21H                 ;hiện Tbao1
        MOV AH, 1               ;đọc So1
        INT 21H
        MOV Sol, AL         ;cất mã của nó đi
        MOV DL, ','             ;dấu phẩy đan xen giữa 2 số
        MOV AH, 2
        INT 21H
        MOV AH, 1               ;đọc số 2
        INT 21H
        MOV So2, AL         ;cất mã của nó đi
        LEA SI, So1             ;SI chỉ vào toán hạng 1
        LEA DI, So2             ;DI chỉ vào toán hạng 2
        LEA BX, SUM             ; BX chỉ vào kết quả
        CALL ADD2SO             ;cộng 2 số
        LEA DX, Tbao2
```

```
        INT 21H                 ;hiện thị Tbao2
        MOV AH, 4CH             ;về DOS
        INT 21H
    MAIN Endp

ADD2SO Proc
        ;tính tổng 2 số
        ;Vào:   SI: địa chỉ của số hạng 1
                DI: địa chỉ của số hạng 2
                BX: địa chỉ của kết quả
        ;Ra: {DS:BX} chứa kết quả
        MOV AL, {SI}            ;lấy mã của số1
        ADD AL, {DI}            ;cộng với mã của số2
        ADD {BX}, AL            ;đổi ra ASCII và cất đi
        RET                     ;về CTC
ADD2SO Endp
        END MAIN
```

Ví dụ 4: truyền tham số qua ngăn xếp

Trong ví dụ này ta sử dụng ngăn xếp để làm chỗ chứa các tham số cần phải truyền. Để dễ so sánh các cách truyền tham số với nhau, ta vẫn giữ nguyên bài toán ở ví dụ trước nhưng thay đổi cách giải để hướng ví dụ vào minh họa việc sử dụng ngăn xếp.

Sau đây là văn bản chương trình thực hiện công việc trên

```
.Model Small
.Stack 100
.Data
        Tbao1       DB 'Gõ vào 2 số có tổng <10: S'
        Tbao2       DB 13, 10, 'Tổng của '
        So1         DB ?
                    DB ' và '
        So2         DB ?
                    DB ' là '
        SUM         DB -30H, '$'
.Code
    MAIN Proc
        MOV AX, @Data
        MOV DS, AX          ;khởi tạo DS
        MOV AH, 9
        LEA DX, Tbao1
        INT 21H                 ;hiện Tbao1
        MOV AH, 1               ;đọc So1
        INT 21H
        MOV Sol, AL         ;cất vào biến để hiện thị
```

```
        PUSH AX                 ;và vào ngăn xếp để truyền
        MOV DL, ','             ;dấu phẩy đan xen giữa 2 số
        MOV AH, 2
        INT 21H
        MOV AH, 1               ;đọc So2
        INT 21H
        MOV So2, AL         ;cất vào biến để hiện thị
        PUSH AX                 ;và vào ngăn xếp để truyền
        CALL ADD2SO             ;cộng 2 số
        ADD SUM, AL             ;lưu kết quả để hiện thị
        MOV AH, 9
        LEA DX, Tbao2
        INT 21H                 ;hiện thị Tbao2
        MOV AH, 4CH             ;về DOS
        INT 21H
    MAIN Endp

ADD2SO Proc
        ; tính tổng 2 số
        ; Vào:  ngăn xếp lúc kể từ đỉnh
        ;       địa chỉ trở về
        ;       số hạng 2
        ;       số hạng 1
        ; Ra:   AX chứa kết quả

        PUSH BP                 ;cất BP để dùng vào việc
        MOV BP, SP              ;BP chỉ vào đỉnh ngăn xếp
        MOV AX, (BP+6)          ;lấy ký tự của Sol
        MOV AX, (BP+4)          ;cộng với ký tự của So2
        POP BP                  ;lấy lại BP
        RET 4                   ;trở về và bỏ qua 4 byte
        ADD2SO Endp
        END MAIN
```

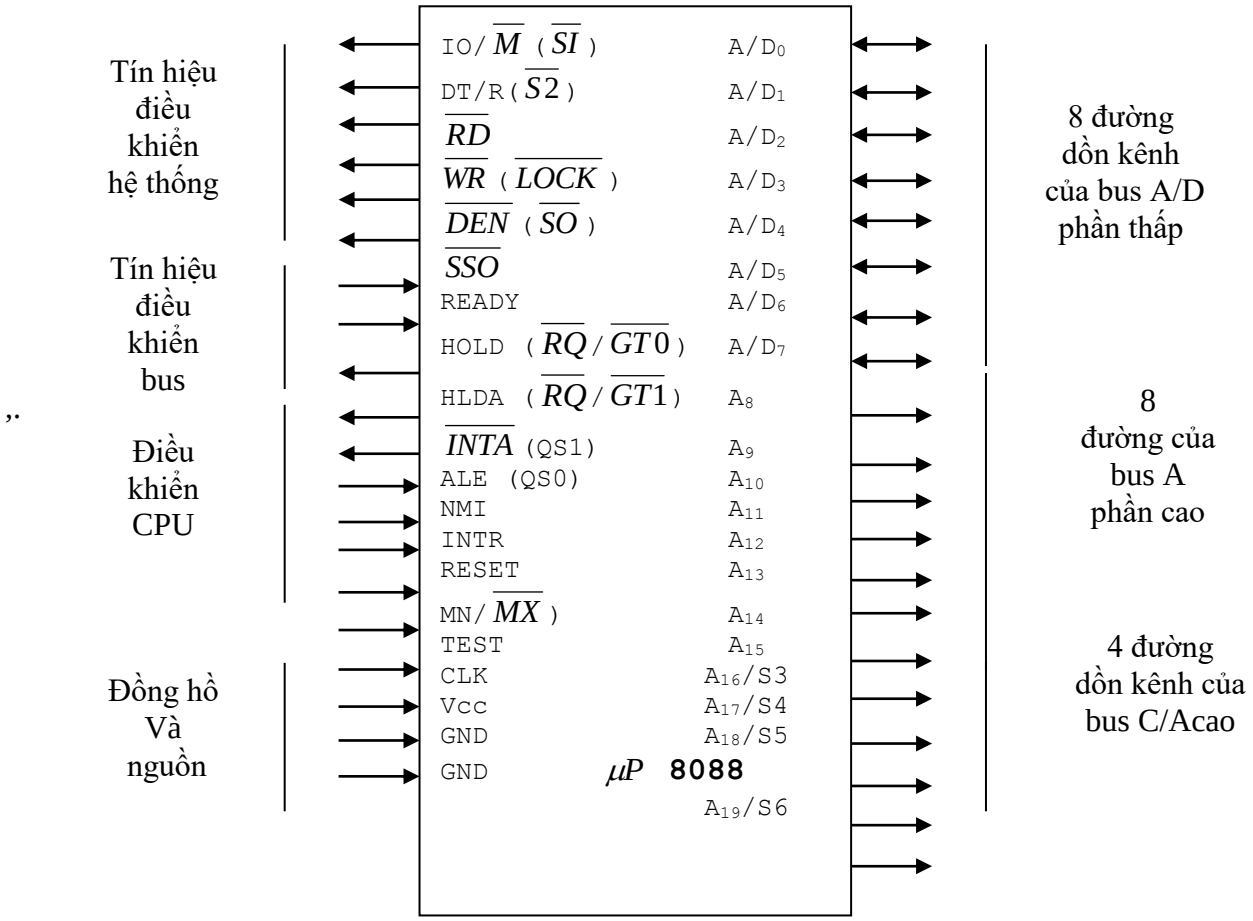
Trong chương trình này khi đọc được cac toán hạng của phép cộng, một mặt ta cất chúng tại ô nhớ dành cho biến (chỉ được dùng đến khi hiện thị thông báo Tbao2 như đã làm ở các ví dụ trước), mặt khác ta cũng cất chúng tại ngăn xếp để truyền giữa các khúc chương trình. Khi thực hiện phép cộng ta lấy lại được địa chỉ trở về vị trí ban đầu như khi chưa dùng đến ngăn xếp.

Một điều nữa cũng phải nhận thấy trong chương trình này là: là ngoài việc dùng ngăn xếp để truyền tham số (các toán hạng), ở đây ta đã sử dụng thêm cả thanh ghi AX cho việc truyền kết quả.

Chương 5 : PHỐI GHÉP 8088 VỚI BỘ NHỚ VÀ TỔ CHỨC VÀO/RA DỮ LIỆU

5.1. Giới thiệu các tín hiệu của 8088 và các mạch phụ trợ 8284, 8288

1.1. Các tín hiệu của 8088



các tín hiệu của 8088 ở chế độ MIN và (MAX).

Hình trên thể hiện việc chia các tín hiệu của 8088 theo các nhóm để ta dễ nhận diện. Sơ đồ bố trí cụ thể các chân của vy xử lý 8088 được thể hiện trong hình 5.2.

Sau đây ta sẽ thể hiện chức năng của từng tín hiệu tại các chân cụ thể.

+ ADO - AD7 [I;O : tín hiệu vào và ra] : Các chân dồn kênh cho các tín hiệu phần thấp của bus dữ liệu và bus địa chỉ. Xung ALE sẽ báo cho mạnh ngoài biết khi nào trên các đường đó có tín hiệu dữ liệu (ALE = 0) hoặc địa chỉ (ALE = 1). Các chân này ở trạng thái trở kháng cao khi μP chấp nhận treo.

+ A8 - A15 [O] : Các bit phần cao của bus địa chỉ. Các chân này ở trạng thái trở kháng cao khi μP chấp nhận treo.

+ A16/S3, A17/S4, A18/S5, A19/S6 [O] : Các chân dồn kênh của địa chỉ phần cao và trạng thái. Địa chỉ A16 - A19 sẽ có mặt tại các chân đó khi ALE = 1 còn khi AEL = 0 thì trên các chân đó có các tín hiệu trạng thái S3 - S6. Các chân này ở trạng thái trở kháng cao khi μP chấp nhận treo.

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

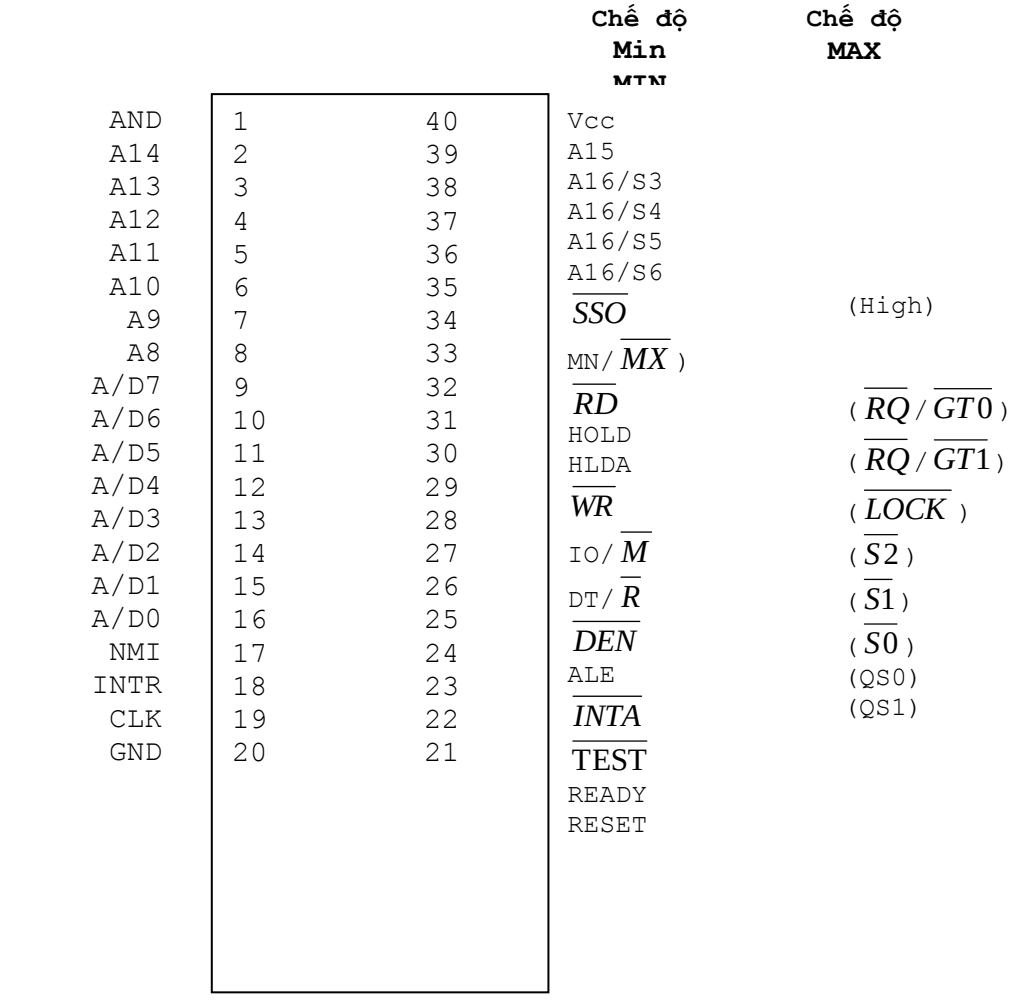
S4	S3	Truy nhập đến
0	0	Đoạn dữ liệu phụ
0	1	Đoạn ngăn xếp

1	0	Đoạn mã hoặc không đoạn nào
1	1	Đoạn dữ liệu

Bit S6 = 0 liên tục, bit S5 phản kánh giá trị bit IF của thanh ghi cờ. Hai bit S3 và S4 phối hợp với nhau để chỉ ra việc truy nhập các thanh ghi đoạn (bảng 5.1)

+ $\overline{RD}$ [O] : Xung cho phép đọc. Khi $\overline{RD}$ = 0 thì bus dữ liệu sẵn sàng nhận số liệu từi bộ nhớ hoặc thiết bị ngoại vi. Chân $\overline{RD}$ ở trạng thái trở kháng cao khi μP chấp nhận 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 ghi/đọc 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 tuợ kéo dài thời gian thực hiện lệnh ghi/đọc bằng cách chèn thêm các chu kỳ đợi.



Sơ đồ chân của CPU 8088.

+ 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 nó đi vào chu kỳ chấp nhận ngắt và đưa ra bên ngoài tín hiệu INTA = 0.

+ $\overline{TEST}$ [I] : Tín hiệu tại chân này được kiểm tra bởi lệnh WAIT. Khi CPU thực hiện lệnh WAIT mà lúc đó tín hiệu $\overline{TEST}$ =1, nó sẽ chờ cho đến khi tín hiệu $\overline{TEST}$ = 0 thì mới thực hiện lệnh tiếp theo.

+ NMI [I] : Tín hiệu yêu cầu ngắt không che được. Tín hiệu này không bị khống chế bởi cờ IF và nó sẽ được CPU nhận biết bằng các 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 thực hiện chương trình phục vụ ngắt kiểu INT2.

+ RESET [I] : tín hiệu khởi động lại 8088. khi RESET = 1 kéo dài ít nhất trong thời gian 4 chu kỳ đồng hồ thì 8088 bị buộc phải khởi động lại : nó xoá các thanh ghi DS, ES, SS, IP và FR về 0 và bắt đầu thực hiện chương trình tại địa chỉ CS:IP = FFFF:0000H (chú ý cờ IF ← 0 để cấm các yêu cầu ngắt khác tác động vào CPU và cờ TF ← 0 để bộ vi xử lý không -bị đặt trong chế độ chạy tưng 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. Tại đây CPU được cung cấp +5V±10%.340mA

+ GND [O] : Hai chân nguồn để nối với điểm OV của nguồn nuôi.

+ MN/MX [I] : Chân điều khiển hoạt động của CPU theo chế độ MIN/MAX.

Do 8088 có thể làm việc ở 2 chế độ khác nhau nên có một số chân tín hiệu phụ thuộc vào các chế độ đó.

✓ Chế độ MIN (Chân MN/MX cần được nối thẳng vào +5V mà không qua điện trở !)

Trong chế độ MIN tất cả các tín hiệu điều khiển liên quan đến các thiết bị ngoại vi truyền thống và bộ nhớ giống như trong hệ 8085 đều có sẵn trong 8088. Vì vậy việc phối ghép với các thiết bị đó sẽ rất dễ dàng và chính vì tận dụng được các phối ghép ngoại vi sẵn nên có thể giảm giá thành hệ thống.

+ IO/$\overline{M}$ [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 đọt biến $\overline{WR}$ = 1. Chân $\overline{WR}$ ở trạng thái trở kháng cao khi μP chấp nhận treo.

+ INTA [O] : Tín hiệu báo cho các mạch bên ngoài biết CPU chấp nhận yêu cầu ngắt INTR. Lúc này CPU đưa ra INTA = 0 để báo là nó đang chở mạch ngoài đưa vào số hiệu ngắt (kiểu ngắt) trên bus dữ liệu.

+ 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 bị thả nối (trong trạng thái trở kháng cao) khi CPU bị treo thì ALE = 0.

+ DT/$\overline{R}$ [O] : Tín hiệu điều khiển các đệm 2 chiều của bus dữ liệu để chọn chiều chuyển của vận dữ liệu trên bus D. Chân này ở trạng thái trở kháng cao khi μP chấp nhận treo.

+ $\overline{DEN}$ [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 machj 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 8088 sẽ tự tách ra hệ thống bằng cách treo tất cả các bus A, bus D, bus C của nó (các bus ở trạng thái trở kháng cao) để bộ điều khiển DMA (DMA contrroller, DMAC) có thể lấy được quyền điều khiển hệ thống để làm các công việc trao đổi dữ liệu.

Bảng các chu kỳ của bus qua các tín hiệu $\overline{SS0}$, IO/$\overline{M}$, DT/$\overline{R}$

IO/$\overline{M}$	DT/$\overline{R}$	$\overline{SS0}$	Chu kỳ điều khiển của bus

0	0	0	Đọc mã lệnh
0	0	1	Đọc bộ nhớ
0	1	0	Ghi bộ nhớ
0	1	1	Bus rôĩ (nghĩ)
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 (halt)

+ HLDA [O] : Tín hiệu báo cho bên ngoài biết yêu cầu treo CPU để dùng các bus đã được chấp nhận , và CPU 8088 đã treo các bus A, bus D và một số tín hiệu của bus C.

+ $\overline{SSO}$ [O] : Tín hiệu trạng thái. Tín hiệu này giống như $\overline{SO}$ trong chế độ MAX và được dùng kết hợp với IO/M và DT/$\overline{R}$ để giải mã các chu kỳ hoạt động của bus (xem bảng 5.2).

✓ Chế độ MAX (Chân MN/MX nối đất)

Trong chế độ MAX 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 mạch điều khiển bus 8288. Chế độ MAX được sử dụng khi trong hệ thống có mặt bộ đồng xử lý toán học 8087

+ $\overline{S2, S1}$ và $\overline{S0}$ [O] : Các chân trạng thái dùng trong chế độ MAX để ghép với mạch điều khiển bus 8288. Các tín hiệu này được 8288 dùng để tạo ra các tín hiệu điều khiển trong các chu kỳ hoạt động của bus. Các tín hiệu điều khiển đó được chỉ ra trong bảng 5.3.

Bảng các tín hiệu điều khiển của 8288.

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Chu kỳ điều khiển của 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, $\overline{AIOWC}$
0	1	1	Dừng (halt)	Không
1	0	0	Đọc mã lệnh	MRDC
1	0	1	Đọc bộ nhớ	MRDC
1	1	0	Ghi bộ nhớ	MWTC, $\overline{AMWC}$
1	1	1	Bus rỗi (nghĩ)	Không

+ $\overline{RQ}/\overline{GT0}$ và $\overline{RQ}/\overline{GT1}$ [I/O] : Các tín hiệu yêu cầu dùng bus của các bộ xử lý khác hoặc thông báo chấp nhận treo của CPU để cho các bộ vi xử lý khác dùng bus. $\overline{RQ}/\overline{GT0}$ có mức ưu tiên hơn $\overline{RQ}/\overline{GT1}$.

+ $\overline{LOCK}$ [O] : Tín hiệu do CPU đưa ra để cấm các bộ xử lý khác trong hệ thống dùng bus trong khi nó đang thi hành một lệnh nào đó đặt sau tiếp đầu LOCK.

+ QS0 và QS1 [O] : Tín hiệu thông báo các trạng thái khác nhau của đệm lệnh (hàng đợi lệnh). Bảng dưới đây cho biết các trạng thái của đệm lệnh được mã hoá bằng các tín hiệu trên.

Trong hệ vi xử lý với sự có mặt của bộ đồng hồ xử lý toán học 8087, các tín hiệu này được mạch 8087 dùng để đồng bộ quá trình hoạt dộng của nó với bộ vi xử lý 8088.

Bảng các trạng thái của lệnh đệm

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

5.1.2. Phân kênh để tách thông tin và việc đệm cho các bus

Để giảm bớt khó khăn về mặt công nghệ do việc phải chế tạo nhiều chân cho các tín hiệu của vi mạch CPU, người ta đã tìm cách hạn chế số chân của vi mạch bằng cách dồn kênh nhiều tín hiệu trên cùng một chân. Ví dụ các chân AD0 - AD8 của 8088 được dồn kênh để có thể đưa ra bên ngoài các thông tin về địa chỉ phần thấp và dữ liệu phần thấp. Khi nhận được các tín hiệu đó ở bên ngoài vi mạch, ta phải tiến hành tách các tín hiệu để tái tạo lại các tín hiệu gốc cho các bus độc lập (bus địa chỉ và bus dữ liệu). Việc này thực hiện bằng cách sử dụng các vi mạch chức năng thích hợp ở bên ngoài (thông thường thì đó là các mạch chốt). Ta cũng phải làm tương tự như vậy đối với các chân dồn địa chỉ/trạng thái. Để hỗ trợ cho việc tách thông tin này, CPU đưa ra thêm xung ALE sao cho khi ALE ở mức cao sẽ có tác dụng báo cho bên ngoài biết lúc này thông tin về địa chỉ tại các chân dồn kênh có giá trị. Xung ALE được dùng để mở các mạch chốt và tách được các thông tin về địa chỉ bị dồn kênh.

Muốn nâng cao tải của các bus để đảm nhận việc nuôi các mạch bên ngoài. Các tín hiệu ra và vào CPU cần phải được khuếch đại thông qua các mạch đệm một chiều hoặc hai chiều với các đầu ra thường hoặc đầu ra 3 trạng thái.

Hình 5.3 cho thấy một ví dụ đơn giản các tổ chức việc tách tín hiệu địa chỉ từ các tín hiệu dồn kênh chỉ/dữ liệu hoặc địa chỉ/điều khiển bằng các mạch chốt 74LS373 và việc sử dụng các bộ khuếch đại đệm 74LS244 và 74LS245 cho các tín hiệu của bộ vi xử lý 8088 làm việc ở chế độ MIN.

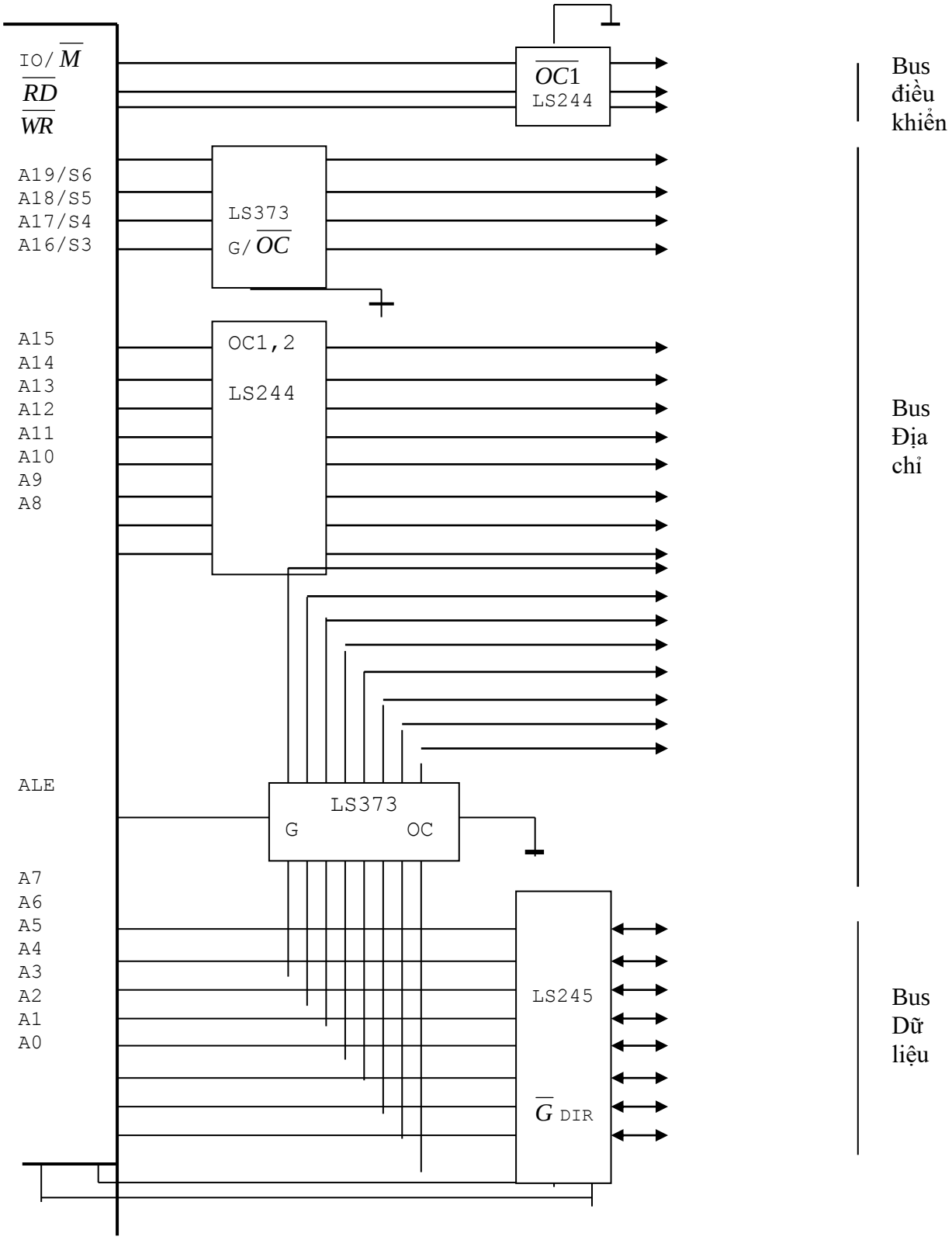
Hình 5.5 cung cấp cho ta hình ảnh tỉ mỉ hơn về cách tổ chức cụ thể khác của bus địa chỉ, dữ liệu và điều khiển thông qua lược đồ của máy IBM PC/XT. Trong đó bộ vi xử lý 8088 làm việc ở chế độ MAX.

Bên cạch CPU trên hình 5.4 ta còn thấy sự có mặt của các mạch phụ trợ của intel như :

+ Bộ điều khiển bus 8288.
+ Bộ tạo ra các xung đồng hồ 8284.
+ Bộ phối ghép ngoại vi song song 8255.
+ Bộ điều khiển trao đổi dữ liệu bằng cách thâm nhập trực tiếp vào bộ nhớ 8237.
+ Bộ điều khiển ngắt ưu tiên 8259.
+ Bộ đếm/định thời gian 8253 và
+ Chỗ cắm dành cho bộ đồng xử lý toán học 8087.

Một số mạch trong các mạch kể trên cũng sẽ được giới thiệu trong chương trình này và các chương trình sau để ta có thể hiểu được hoạt động của từng hệ.

Trên sơ đồ này ta cũng thấy việc sử dụng các mạch chốt và mạch khuếch đại đệm thông dụng (các mạch 74LS373, 74LS244 và 74LS245) tại những chỗ cần thiết của bus địa chỉ, bus dữ liệu và bus điều khiển như đã nói ở trên.



5.1.3. Mạch tạo xung nhịp 8284.

Cho dù làm việc trong chế độ MIN hay MAX, CPU 8088 luôn cần xung nhịp (xung đồng hồ) từ mạch tạo xung nhip 8284. Mạch tạo xung nhịp không những cung cấp xung nhịp

với tần số thích hợp cho toàn hệ mà nó còn có ảnh hưởng tới việc đồng bộ tín hiệu RESET và tín hiệu READY của CPU.

Ý nghĩa của các tín hiệu

+ $\overline{AEN1}$, $\overline{AEN2}$: 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 $\overline{AEN1}$, $\overline{AEN2}$ dùng để gây ra các chu kỳ đợi ở CPU.

+ $\overline{ASYNC}$: Chọn đồng bộ hai tầng hoặc đồng bộ một tầng cho tín hiệu RDY1, RDY2. Trong chế độ đồng bộ một tầng ($\overline{ASYNC}$ = 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 ($\overline{ASYNC}$ = 0) tín hiệu RDY chỉ có ảnh hưởng đến tín hiệu READY khi có sườn xuống của xung đồng hồ tiếp theo.

+ READY : Nối đến đầu READY của CPU. Tín hiệu này được đồng bộ với các tín hiệu RDY1, RDY2.

+ X1, X2 : Nối với hai chân của thạch anh với tần số fx, thạch anh này là một bộ phận của một mạch dao động bên trong 8284 có nhiệm vụ tạo xung chuẩn dùng làm tín hiệu đồng hồ cho toàn hệ thống.

+ F/$\overline{C}$: Dùng để chọn nguồn tín hiệu chuẩn cho 8284. Khi chân này ở mức cao thì xung đồng hồ bên ngoài sẽ được dùng 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.

+ EFI : lối vào cho xung từ bộ dao động ngoại.

+ CLK : Xung nhịp $f_{CLK}=f_x/3$với độ rỗng 77% nối đến chân của CLK của 8088.

+ PCLK : Xung nhịp $f_{CLK}=f_x/6$với độ rỗng 50% dành cho thiết bị ngoại vi.

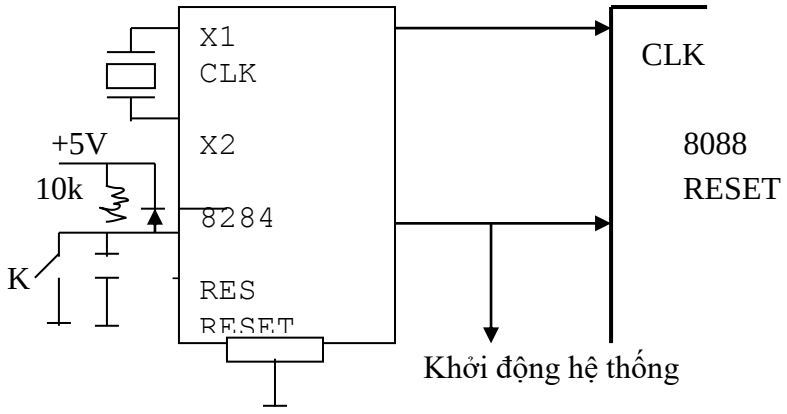
+ OSC : Xung nhịp đã được khuếch đại có tần số bằng f_x của bộ dao động.

+ $\overline{RES}$: Chân khởi động, nối với mạch RC để 8284 để tự khởi động khi bật nguồn.

+ RESET : Nối vào RESETcủa 8088 và là tín hiệu khởi động lại cho toàn hệ

+ CSYNC : Lối vào cho xung đồng bộ chung khi trong hệ thống có các 8284 dùng dao động ngoài tại chân này (hình 5.6)

+ Hình 5.6 biểu diễn các đường nối tín hiệu chính của 8088 và 8284. Mạch 8284 nhận được xung khởi động từ bên ngoài thông qua mạch RC khi bắt đầu bật điện



Hình 5.6. Mạch 8284 nối với 8088.

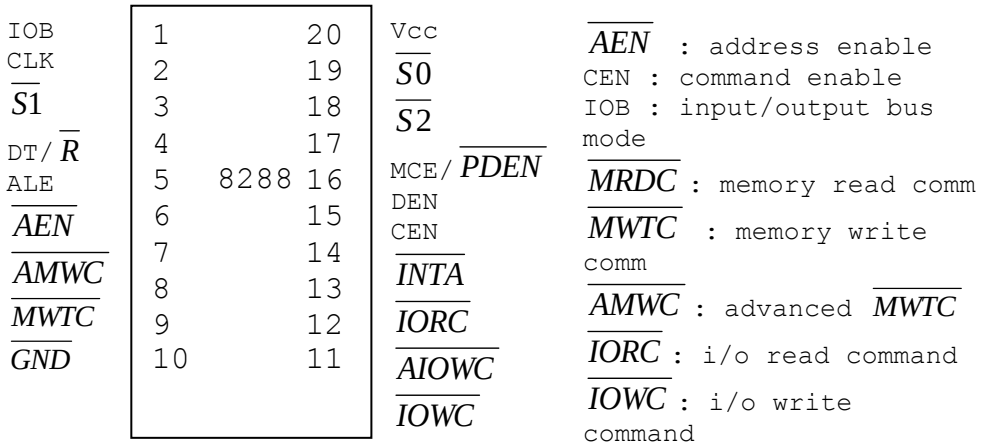
hoặc xung khởi động lại khi bấm công tắc K. Từ xung này 8284 có nhiệm vụ đưa ra xung khởi động đồng bộ cho CPU cùng với tất cả các thành phần khác của hệ thống.

5.1.4. Mạch điều khiển bus 8288

Như đã giới thiệu ở phần trước, vi mạch 8288 là mạch điều khiển bus, nó lấy 1 số tín hiệu điều khiển của CPU và cung cấp tất cả các tín hiệu điều khiển cần thiết cho hệ vi xử lý khi CPU 808àm việc ở chế độ MAX.

Sơ đồ chân và các tín hiệu của 8288 được thể hiện trên hình 5.7.

Các tín hiệu chính của 8288 bao gồm :



MCE/PDEN : master cascade enable/peripheral data enable $\overline{R}$

Hình 5.7. Mạch tạo xung điều khiển 8288.

+ $\overline{S_2}, \overline{S_1}, \overline{S_0}$ [I, I, I] : là các tín hiệu trạng thái lấy thẳng từ CPU. Tuỳ theo các tín hiệu này mà mạch 8288 sẽ tạo ra các tín hiệu điều khiển khác nhau tại các chân ra của nó để điều khiển hoạt động của các thiết bị nối với CPU. Bảng 5.3 mô tả các tín hiệu vào và ra đó.

+ CLK [I] : Đây là đầu vào nối với xung đồng hồ hệ thống (từ mạch 8284) và dùng để đồng bộ toàn bộ các xung điều khiển đi ra từ mạch 8288.

+ CEN [I] : Là tín hiệu đầu vào để cho phép đưa ra tín hiệu DEN và các tín hiệu điều khiển khác của 8288.

+ IOB [I] : tín hiệu để điều khiển mạch 8288 làm việc ở các chế độ bus khác nhau.

Khi IOB = 1 8288 làm việc ở chế độ bus vào/ra, khi IOB = 0 mạch 8288 làm việc ở chế độ bus hệ thống (như trong các máy IBM PC).

+ $\overline{MRDC}$ [O] : tín hiệu điều khiển đọc bộ nhớ. Nó kích hoạt bộ nhớ đưa dữ liệu ra bus.

+ $\overline{MWTC}$ [O] $\overline{AMWC}$ [O] : là các tín hiệu điều khiển ghi bộ nhớ hoặc ghi bộ nhớ kéo dài.Đó thực chất là các tín hiệu giống như $\overline{MEMW}$, nhưng $\overline{AMWC}$ (advanced memory write command) hoạt động sớm lên một chút để tạo ra khả năng cho các bộ nhớ chậm có thêm được thời gian ghi.

+ $\overline{IORC}$ [O] : tín hiệu điều khiển đọc thiết bị ngoại vi. Nó kích hoạt các thiết bị được chọn để các thiết bị này đưa dữ liệu ra bus.

+ $\overline{IOWC}$ [O] $\overline{AIOWC}$ [O] : là các tín hiệu điều khiển đọc thiết bị ngoại vi hoặc đọc thiết bị ngoại vi kéo dài.

Đó thực chất là các tín hiệu giống như $\overline{IOW}$, nhưng $\overline{AIOWC}$ (advanced I/O write command) hoạt động sớm lên một chút để tạo ra khả năng cho các bộ nhớ chậm có thêm được thời gian ghi.

+ $\overline{INTA}$ [O] : là đầu ra để thông nbaos là CPU chấp nhận yêu cầu ngắt của thiết bị ngoại vi và lúc này các thiết bị ngoại vi phải đưa ra số hiệu ngắt ra bus để CPU đọc.

+ DT/$\overline{R}$ [O] : là tín hiệu để điều khiển hướng đi của dữ liệu trong hệ vào hay ra so với CPU (DT/$\overline{R}$ = 0 : CPU đọc dữ liệu, DT/$\overline{R}$ = 1 CPU ghi dữ liệu).

Trong các máy IBM PC thì tín hiệu này được nối đến các chân DIR của mạch đệm 2 chiều 74LS245 để điều khiển dữ liệu đi từ CPU đến bus hệ thống khi ghi hoặc ngược lại, từ bus hệ thống đến CPU khi đọc.

+ DEN [O] : đây là tín hiệu để điều khiển bus dữ liệu trở thành bus cục bộ hay bus hệ thống.

Trong các máy IBM PC thì tín hiệu này được sử dụng cùng với tín hiệu của mạch điều khiển ngắt PIC 8259 để tạo ra tín hiệu điều khiển cực G của mạch đệm 2 chiều 74LS245.

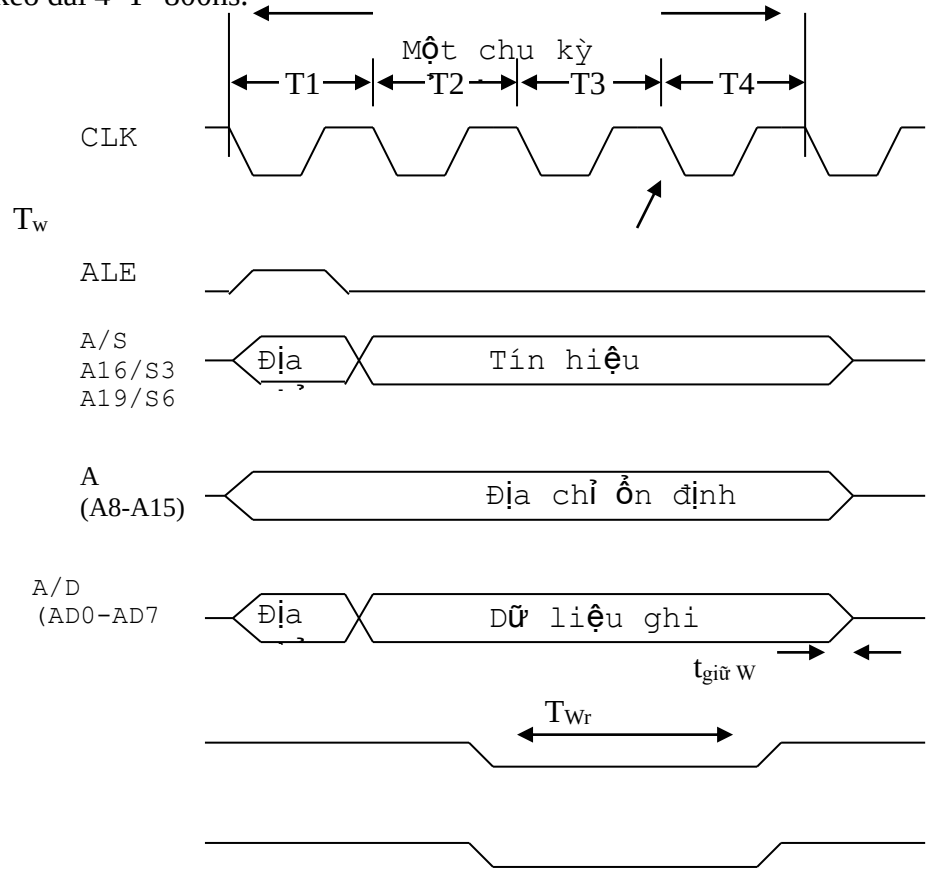
+ MCE/$\overline{PDEN}$ [O] : đây là tín hiệu dùng để định chế độ làm việc cho mạch điều khiển ngắt PIC 8259 để nó làm việc ở chế độ chủ.

+ ALE [O] : đây là tín hiệu cho phép chốt địa chỉ tại các chân dồn kênh địa chỉ - dữ liệu AD0 - AD7, tín hiệu này thường được nối với chân G của mạch 74LS373 để điều khiển mạch này chốt lấy địa chỉ.

5.1.5. Biểu đồ thời gian của các lệnh ghi/đọc

Trên hình 5.8 và 5.9 là các biểu đồ thời gian đã được đơn giản hoá của các tín hiệu cơ bản trong CPU 8088 cho các lệnh ghi/đọc bộ nhớ hoặc thiết bị ngoại vi.

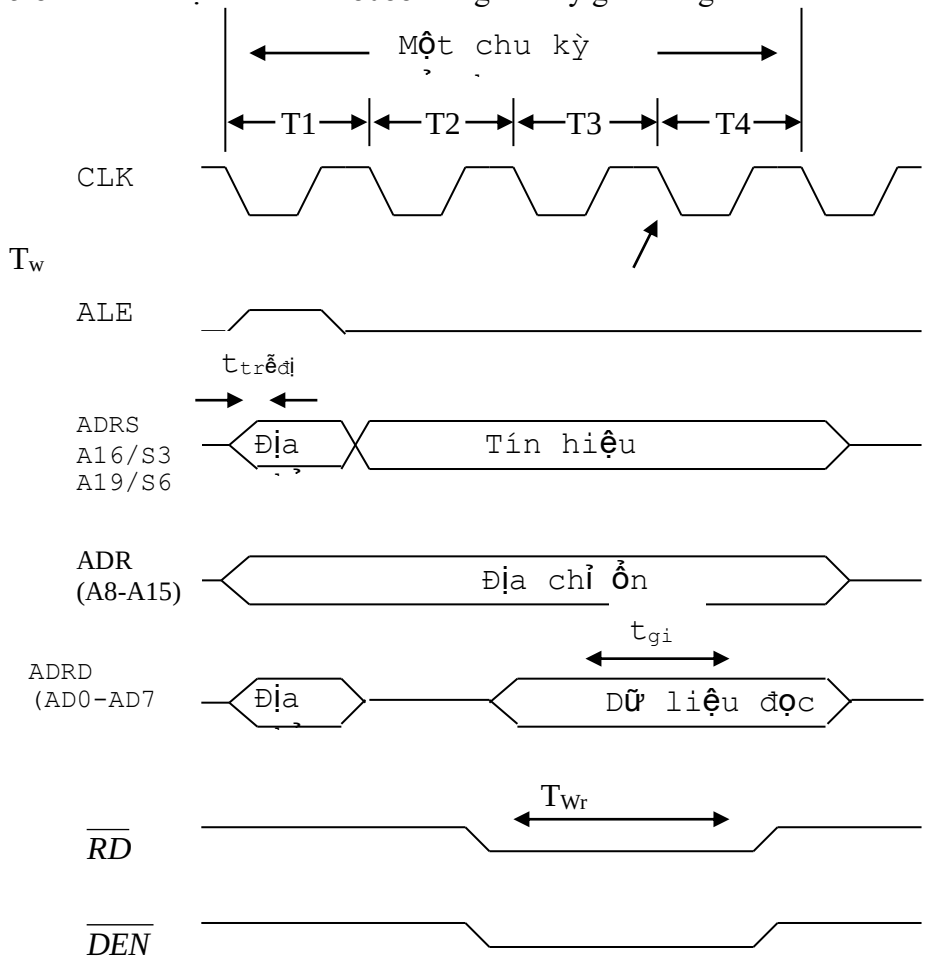
Trong trường hợp bình thường một chu kỳ ghi/đọc (còn gọi là chu kỳ bus) của CPU kéo dài 4 chu kỳ đồng hồ. Các chu kỳ đồng hồ được đánh dấu là T1, T2, T3 và T4. Nếu CPU làm việc với tần số đồng hồ 5MHz thì một chu kỳ đồng hồ kéo dài T=200ns và một chu kỳ bus kéo dài 4*T=800ns.



$\overline{WR}$

$\overline{DEN}$

Hình 5.8. Các tín hiệu của CPU 8088 trong chu kỳ ghi đơn giản hoá.



Hình 5.9. Các tín hiệu của CPU 8088 trong chu kỳ đọc đơn giản hoá.

Chúng ta mô tả tóm tắt các hiện tượng xảy ra trong một chu kỳ T nói trên.

+ Chu kỳ T1 :

Trong chu kỳ này địa chỉ của bộ nhớ hay thiết bị ngoại vi được đưa ra trên các đường địa chỉ, hoặc địa chỉ/dữ liệu và địa chỉ/ trạng thái. Các tín hiệu điều khiển ALE. DT/$\overline{R}$. IO/$\overline{M}$ cungx được đưa ra để giúp việc hoàn tất việc giữ thông tin địa chỉ này.

+ Chu kỳ T2 :

Trong chu này CPU đưa ra các tín hiệu điều khiển $\overline{RD}$ hoặc $\overline{WR}$. $\overline{DEN}$ và tín hiệu dữ liệu trên D0 - D7 nếu là lệnh ghi. $\overline{DEN}$ thường dùng để mở các bộ đệm của bus dữ liệu nếu như chúng được dùng trong hệ. Tại cuối kỳ T2 (và giữa mỗi chu kỳ T của T_w , nếu có) CPU lấy mẫu tín hiệu READY để xử lý trong chu kỳ tiếp theo khi nó phải làm việc với bộ nhớ hoặc thiết bị ngoại vi chậm.

+ Chu kỳ T3 :

Trong chu kỳ này CPU dành thời giờ cho bộ nhớ hay thiết bị ngoại vi khi nhập dữ liệu. Nếu là chu kỳ đọc dữ liệu thì tại cuối T3 CPU sẽ lấy mẫu tín hiệu của bus dữ liệu.

Nếu tại cuối chu kỳ đồng hồ T2 (hoặc giữa mỗi chu kỳ T của Tw) mà CPU phát hiện ra tín hiệu READY=0 (do bộ nhớ hay thiết bị ngoại vi đưa đến) thì CPU tự xen vào sau T3 một vài chu kỳ T để tạo chu kỳ đợi Tw = n*T nhằm kéo dài thời gian thực hiện lệnh, tạo điều kiện cho bộ nhớ hoặc thiết bị ngoại vi. có đủ thời gian hoàn tất việc ghi/đọc dữ liệu.

+ Chu kỳ T4 :

Trong chu kỳ này các tín hiệu trên bus được giải hoạt (đưa về trạng thái không tích cực) để chuẩn bị cho chu kỳ bus mới. Tín hiệu $\overline{WR}$ trong khi chuyển trạng thái từ 0 lên 1 sẽ kích hoạt động quá trình đưa vào bộ nhớ hay thiết bị ngoại vi.

Trên các hình vẽ 5.8 và 5.9 cũng biểu diễn các thông số quang trọng về mặt thời gian liên quan đến tốc độ hoạt động tối thiểu cần thiết của các bộ nhớ hoặc thiết bị ngoại vi nếu chúng muốn làm việc với CPU 5MHz.

Trong biểu đồ thời gian đọc (hình 5.9) ta thấy việc truy nhập bộ nhớ kéo dài trong khoảng thời gian từ T1 - T3 (gần 3 chu kỳ đồng hồ 3*T = 600 ms). Trong tổng số thời gian này phải tính đến thời gian trễ khi chuyển địa chỉ $t_{trễ\ địa\ chỉ}$ = 110ns, thời gian giữ của dữ liệu khi đọc $t_{giữR}$ = 30 ns và thời gian trễ do việc truyền tín hiệu qua các mạch đệm nhiều nhất là $t_{trễ\ đệm}$ = 40ns. Như vậy các bộ nhớ nối với 8088 - 5MHz cần phải có thời gian thâm nhập nhỏ hơn :

$$3*T - t_{trễ\ địa\ chỉ} - t_{giữR} - t_{trễ\ đệm} = 600 - 110 - 30 - 40 = 420ns.$$

Mặt khác với CPU 8088 5MHz thì độ rộng xung đọc là T_{RD} = 325ns, đó là thời gian đủ dài để cho bộ nhớ với thời gian thâm nhập cỡ 420ns làm việc.

Trong biểu đồ thời gian ghi (hình 5.8) ta thấy phải có một thời gian giữ dữ liệu tối thiểu để ghi $t_{giữW}$ = 88ns sau khi $\overline{WR}$ đột biến từ 0 lên 1. trong thực tế thời gian này gần như bằng 0 đối với bộ nhớ thông dụng. Độ dài của xung ghi đối với CPU 8088 - 5MHz là T_{WR} = 340ns cũng là phù hợp với các bộ nhớ với thời gian thâm nhập cỡ 450ns.

Trên hình 5.10 là một mạch dùng để xem thêm chu kỳ đợi với thời gian đợi tuỳ chọn nT (n=0-7). Bằng cách thay đổi đầu nối đến các đầu ra của mạch vào song song ra nối tiếp 74LS164 cho đến khi có sườn dương của T2.

5.2. Phối ghép 8088 với bộ nhớ

5.2.1. Bộ nhớ bán dẫn

Trước khi nói về phối ghép 8088 với bộ nhớ ta nói qua một chút về các bộ nhớ bán dẫn thường dùng với bộ vi xử lý.

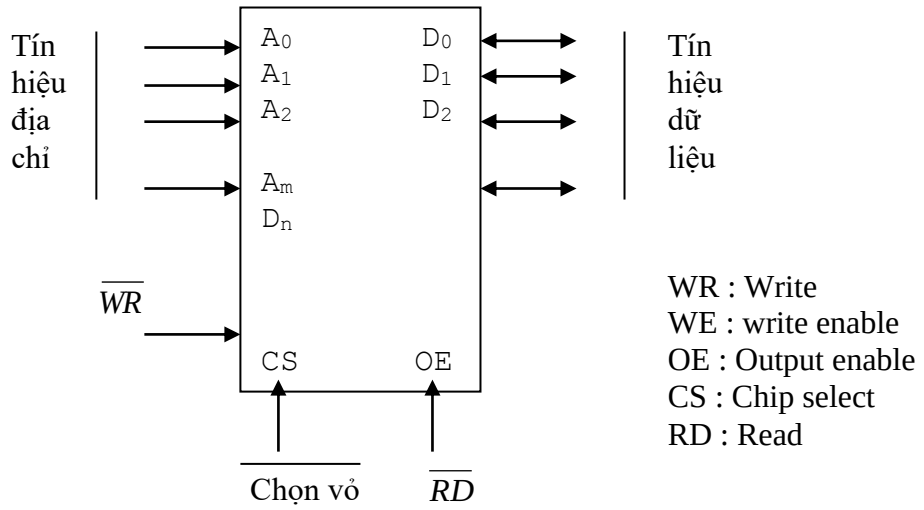
Các bộ nhớ bán dẫn thường dùng với bộ vi xử lý bao gồm :

+ Bộ nhớ cố định ROM (read only memory, bộ nhớ có nội dung ghi sẵn chỉ để đọc ra). Thông tin ghi trong mạch không bị mất khi mất nguồn điện nuôi cho mạch.

+ Bộ nhớ bán cố định EPROM (aerraseble programmable ROM là bộ nhớ ROM có thể lập trình được bằng xung điện và xoá được bằng tia cực tím).

+ Bộ nhớ không cố định RAM (random access memory, bộ nhớ ghi/đọc) thông tin ghi trong mạch bị mất khi mất nguồn điện nuôi cho mạch. Trong các bộ nhớ RAM ta còn phân biệt ra loại RAM tĩnh (stiatic RAM hay SRAM, trogn đó mỗi phần tử nhỏ là một mạch lật hay trạng thái ổn định) và loại RAM động (dyamic RAM hay DRAM, trong đó mỗi phần tử nhớ là một tụ điện rất nhỏ được chế tạo bằng công nghệ MOS).

Một bộ nhớ thường được chế tạo nên từ nhiều vi mạch nhớ. Một vi mạch nhớ thường có dạng cấu trúc tiêu biểu như sau (hình 5.11)



Hình 5.11. sơ đồ khối 1 vi mạch nhớ.

Theo sơ đồ khối này ta thấy một l vi mạnh nhớ có các nhóm tín hiệu sau :

✓ Nhóm tín hiệu địa chỉ :

Các tín hiệu địa chỉ có tác dụng chọn ra một ô nhớ (từ nhớ cụ thể để ghi/đọc. Các ô nhớ có độ dài khác nhau tuỳ theo nhà sản xuất : 1, 4, 8, bit. Số đường tín hiệu địa chỉ có liên quan đến dung lượng của mạch nhớ. Với một mạch nhớ có m bit địa chỉ thì dung lượng của mạnh nhớ đó là 2m từ nhớ. Ví dụ, với m = 10 ta có dung lượng mạch nhớ là 1K ô nhớ (1 kilô = 210 = 1024) và với m=20 ta có dung lượng mạch nhớ là 1M ô nhớ (1 Mêga = 2^{20} = 1048576).

✓ Nhóm tín hiệu dữ liệu :

Các tín hiệu dữ liệu thường là đầu ra đối với mạch ROM hoặc đầu vào/ra dữ liệu chung (hai chiều) đối với mạch RAM. Cũng tồn tại mạch nhớ RAM với đầu ra và đầu vào dữ liệu riêng biệt. Đối với RAM loại này, khi dùng trong mạch của bus dữ liệu người sử dụng phải nối hai đầu đó lại. Các mạch nhớ thường có đầu ra dữ liệu kiểu 3 trạng thái. Số dường dây dữ liệu quyết định độ dài từ nhớ của mạch nhớ. Thông thường người ta hay nói rõ dung lượng và độ dài từ nhớ cùng một lúc. Ví dụ mạch nhớ dung lượng 1 Kx8 (tức là 1KB) hoặc 16Kx4 ...

✓ Nhóm tín hiệu chọn vi mạch (chọn vỏ) :

Các tín hiệu chọn võ là $\overline{CS}$ (chip select) hoặc $\overline{CE}$ (Chip enable) thường được dùng để tạo ra vi mạch nhớ cụ thể để ghi/đọc. Tín hiệu chọn võ ở các mạch RAM thường la $\overline{CS}$, còn ở mạch ROM thường là $\overline{CE}$. Các tín hiệu chọn võ thường được nối với đầu ra của bộ giải mã địa chỉ. Khi một mạnh nhớ không được chọn thì bus dữ liệu của nó bị treo (ở trạng thái trở kháng cao)

✓ Nhóm tín hiệu điều khiển :

Tín hiệu điều khiển cần có trong tất cả các mạch nhớ. Các mạch nhớ ROM thường có một đầu vào điều khiển $\overline{OE}$ (output enable) để cho phép dữ liệu được đưa ra bus. Một mặt nhớ không được mở bởi $\overline{OE}$ thì bus dữ liệu của nó bị treo.

Một mạch nhớ Ram nếu chỉ có một tín hiệu điều khiển thì thường đó là $\overline{R}/\overline{W}$ để điều khiển quá trình ghi/đọc. Nếu mạch nhớ RAM có hai tín hiệu điều khiển đó thường là $\overline{WE}$ (write enable) để điều khiển ghi và $\overline{OE}$ để điều khiển đọc. Hai tín hiệu này phải ngược pha nhau để điều khiển việc ghi/đọc mạch nhớ.

Một thông số đặc trưng khác của bộ nhớ là thời gian thâm nhậm t_{ac}. Nói chung nó được định nghĩa như là thuòi gian kể từ khi có xung địa chỉ trên bus địa chỉ cho đến khi có dữ liệu ra ổn định trên bus dữ liệu. Thời gian thâm nhậm bộ nhớ phụ thuộc rất nhiều vào công nghệ chế tạo nên nó. Các bộ nhớ làm bằng công nghệ lưỡng cực có thời gian thâm nhập nhỏ (10 - 30ns) còn các bộ nhớ làm bằng công nghệ MOS có thời gian thâm nhập lớn hơn nhiều (cỡ 150ms hoặc hơn nữa).

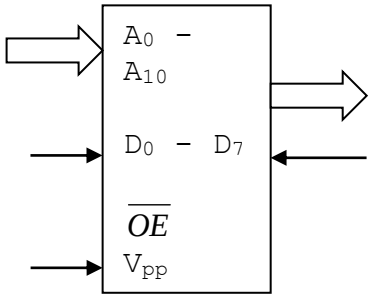
Sau đây là ví dụ một số loại bộ nhớ thường dùng :

- Bộ nhớ EPROM :

Các bộ nhớ EPROM thông dụng tồn tại dưới nhiều kiểu mạch khác nhau. Họ 27xxx có các loại makchj sau : 2708 (1Kx8), 2716 (2Kx8), 2732 (4Kx8), 2764 (8Kx8), 27128 (16Kx8), 27256 (32Kx8), 27512 (64Kx8) với t_{ac} = 250-450ns tuỳ theo loại cụ thể. Trên hình 5.12 là sơ đồ các tín hiệu và bảng chức năng của 2716.

Mạch nhớ 2716 có thời gian thâm nhập t_{ac} = 450ns, như vậy để ghép và làm việc được với CPU 8088 5MHz nó cần phải thêm chu kỳ đợi. Ngược lại mạch nhớ 2716 - 1 lại có t_{ac} = 250ns nên không cần thêm chu kỳ đợi.

Cần lưu ý là trong chế độ duy trì công suất tiêu thụ của mạch giảm được 75% so với công suất khi nó ở chế độ tích cực



Chủ / Chế độ	$\overline{CE}$ /PGM	$\overline{OE}$	V_{pp} [V]	V_{cc} [V]	D_0 - D_7
Đọc	0	0	+5	+5	Dout
Duy trì	1	X	+5	+5	HZ
Ghi	50ms	1	+25	+25	Din
Kiểm tra	0	0	+25	+5	Dout
Câm ghi	0	1	+25	+5	HZ

A_0 - A_{10} : Địa chỉ

D_0 - D_7

$\overline{OE}$: cho phép đưa dữ liệu ra

$\overline{CE}$ /PGM : điện áp ghi

x : không quan tâm

HZ : Trạng thái trở kháng cao

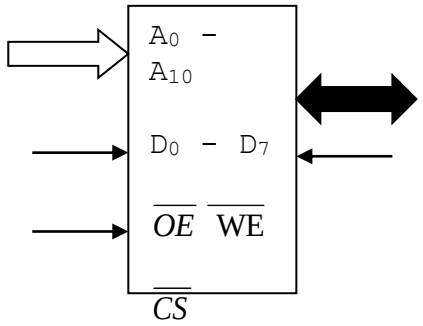
Hình 5.12. Bộ nhớ EPROM 2716 (2Kx8)

- Bộ nhớ RAM tĩnh (SRAM) :

Bộ nhớ SRAM có khả năng lưu giữ thông tin trong nó chừng nào nó còn được cấp điện. Các bộ nhớ SRAM và EPROM cùng dung lượng thường có cách bố trí chân giống nhau

để dễ bề dày thay thế lẫn trong qua trình phát triển hệ thống. Trên hình 5.13 là ví dụ mạch nhớ TMS 4014 (2Kx8) với thời gian thâm nhập t_{ac} = 250ns.

Đã tồn tại trong thực tế mạch nhớ SRAM dung lượng 32Kx8 (62256LP-10) với thời gian thâm nhập cỡ 100ns chế tạo theo công nghệ CMOS và một loại SRAM khác chês tạo theo công nghệ lưỡng cực 8KB - 120KB có thời gian thâm nhập 15ns.

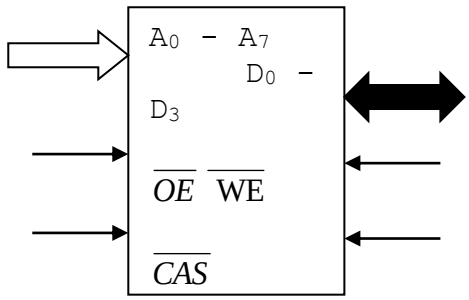


A_0 - A_{10} : Địa chỉ
D_0 - D_7 : Dữ liệu
$\overline{OE}$: cho phép đưa dữ liệu ra
$\overline{WE}$: Cho phép ghi
$\overline{CS}$: Chọn vỏ

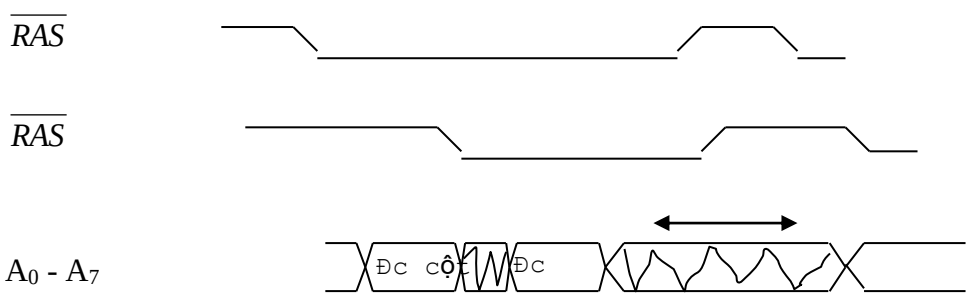
Hình 5.13. Bộ nhớ RAM tĩnh TMS 4016 (2Kx8).

- Bộ nhớ RAM động (DRAM) :

Bộ nhớ DRAM lưu giữ thông tin bằng cách nạp hay không nạp điện tích lên các tụ điện công nghệ MOS. Mỗi phần từ nhớ của bộ nhớ DRAM vì vậy cần được làm tươi lại (bằng cách ghi hay đọc phần tử đó) sau một khoảng thời gian cỡ 15,6 μs, nếu không điện tích trên các tụ điện sẽ bị tiêu tán và dẫn đến mất thông tin. Các mạch DRAM cần có các mạch logic phụ để đảm bảo việc làm tươi và vì thế việc phối ghép đó với bộ vi xử lý là rất phức tạp. Bù lại nhược điểm này các mạch nhớ DRAM lại có ưu điểm là có thể chế tạo được một số lượng rất lớn các phần tử nhớ trên 1 đơn vị điện tích, các vi mạch này do vậy cũng cần rất nhiều chân cho các tín hiệu địa chỉ. Để làm giảm bớt số lượng chân địa chỉ trên một vi mạch nhớ, người ta thường chia địa chỉ ra làm hai nhóm : địa chỉ hàng và địa chỉ cột dồn kênh chúng trên các chân địa chỉ, đồng thời cung cấp thêm các tín hiệu cho phép chốt giữ riêng lẽ địa chỉ hàng ($\overline{RAS}$) và cột ($\overline{CAS}$) ở bên trong vi mạch nhớ (hình 5.14).



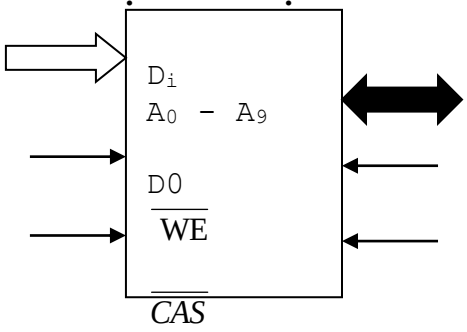
A_0 - A_7 : Địa chỉ
D_0 - D_3 : Dữ liệu
$\overline{CAS}$: xung cho phép chốt địa chỉ cột
$\overline{RAS}$: xung cho phép chốt địa chỉ hàng
$\overline{OE}$: cho phép đưa dữ liệu ra
$\overline{WE}$: Cho phép ghi



Hình 5.14. Bộ nhớ RAM động TMS 4464 (64Kx4).

Các mạch nhớ DRAM thường được chế tạo với độ dài 1 hoặc 4 bit. Đã tồn tại trong thực tế mạch nhớ DRAM dung lượng 1 Mx1, 4 Mx1, và 16 Mx1 và chúng thường được tổ hợp thành bộ nhớ kiểu SIMM (single in-line memory module) hay SIP (single in-line package) dùng trong các máy tính thế hệ mới. Trên hình 5.15 là ví dụ của vi mạch nhớ TMX4C1024 dung lượng 1 Mx1 với thời gian thâm nhập 60ns.

5.2.2. Giải mã địa chỉ cho bộ nhớ

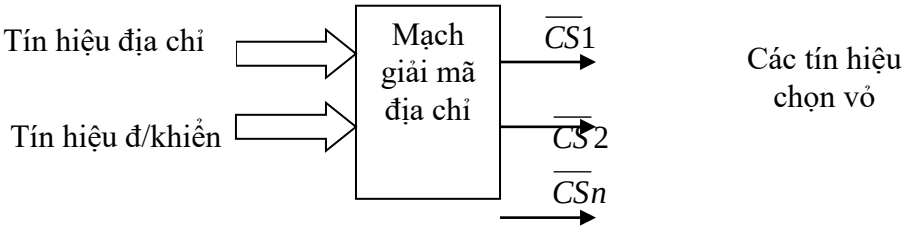


A_0 - A_9 : Địa chỉ
D_i : Dữ liệu vào
D0 : Dữ liệu ra
$\overline{CAS}$: xung cho phép chốt địa chỉ cột
$\overline{RAS}$: xung cho phép chốt địa chỉ hàng
$\overline{WE}$: cho phép ghi
$\overline{WE}$: Cho phép ghi

Hình 5.15. Bộ nhớ RAM động TMX 4C1024 (1Mx1)

Mỗi mạch nhớ nối ghép với CPU cần phải được CPU qui chiếu tới một cách chính xác khi thục hiện các thao tác ghi/đọc. 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 lấy từ mạch giải mã địa chỉ. Việc phân định không gian địa chỉ tổng thể thành các cùng nhớ khác nhau để thực hiện những chức năng nhất định gọi là phân vùng bộ nhớ. Ví dụ, đối với CPU 8088 thì không gian địa chỉ tổng thể dành cho bộ nhớ là 1MB, trong đó vùng nhớ dung lượng 1 KB kể từ địa chỉ thấp nhất 00000H nhất thiết phải được dành cho RAM (vì tai đây ta phải có chỗ để cho một bảng gồm 256 vectơ ngắt của 8088), tại còn vùng nhớ có chứa địa chỉ FFFF0H thì lại nhất thiết phải dành cho ROM hay EPROM (vì FFFF0H là địa chỉ khởi động của CPU).

Về nguyên tắc một bộ giải mã địa chỉ thường có cấu tạo như trên hình 5.16.



Hình 5.16. Mạch giải mã địa chỉ tổng quát.

Đầu vào của bộ giải mã là các tín hiệu địa chỉ và tín hiệu điều khiển. Các tín hiệu địa chỉ gồm các bit địa chỉ có quan hệ nhất định vơi các tín hiệu chọn vỏ ở đầu ra. Tín hiệu điều khiển thường là tín hiệu IO/$\overline{M}$ dùng để phân biệt đối tượng mà CPU chọn làm việc là bộ nhớ hay thiết bị vào/ra. Mạch giải mã là một trong những khâu gây ra việc trễ thời gian của tín hiệu từ CPU tới bộ nhớ hoặc thiết bị ngoại vi mà trong khi chọn mạch nhớ/ngoại vi ta phải tính đến. Tuỳ theo quy mô của mạch giải mã mà ta có thể có ở đầu ra một hay nhiều tín hiệu chọn vỏ.

Giải mã đầy đủ cho một mạch nhớ đòi hỏi ta phải đưa đến đầu vào của mạch giải mã các tín hiệu địa chỉ sao cho tín hiệu ở đầu ra của nó chỉ chọn riêng mạch nhớ đã định. Trong trường hợp này ta phải dùng tổ hợp đầu đủ của các đầu vào địa chỉ tương ứng để chọn được mạch nhớ, vì xung nhận được từ mạch giải mã ngoài việc chọn mạch nhớ ở vùng đã định sẽ có thể chon ra các mạch nhớ ở các vùng nhớ khác nữa.

Như vậy, giải mã thiếu thì tiết kiệm được linh kiện khi thực hiện bộ giải mã nhưng lại làm mất tính đơn trị của xung chọn thu được ở đầu ra.

Thông thường khi thiết kế mạch giải mã người ta hay tính đôi ra một chút để dự phòng, sao cho sau này nếu có sự thay đổi do phải tăng thêm dung lượng củabộ nhớ thì vẫn có thể sử dụng được mạch giải mã đã được thiết kế .

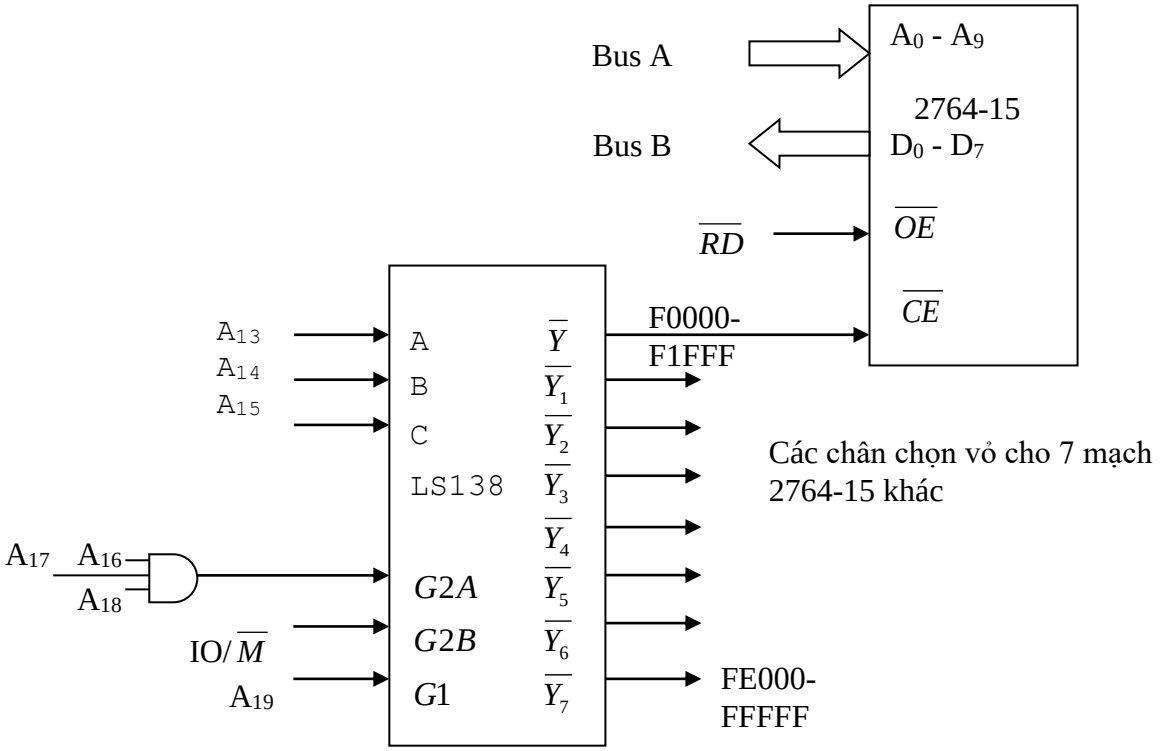
✓ **Thực hiện mạch giải mã bàng các mạch NAND**

Bằng các mạch kiểu 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 đàu vào của mạch cửa nhiều lối và 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.

Trong mạch giải mã đơn giản cho EPROM này, xung chọn vỏ sẽ có tác động khi ta đọc bộ nhớ tại địa chỉ nằm trong khoảng EF800H - FFFFFH, tức là vùng địa chỉ có chứa địa chỉ khởi động của CPU 8088. 9 bit địa chỉphần cao của bus địa chỉ (A11-A19) ở mức 1 sẽ phối hợp cùng xung IO/$\overline{M}$ (đã được đảo) để tạo ra xung chọn vùng nhớ 2 KB đặt tại địa chỉ cao nhất trong không gian địa chỉ của 8088. mỗi ô nhớ cụ thể trong 2 KB của mạch nhớ EPROM 2716-1 sẽ do các bit thấp còn lại của bus địa chỉ (A0 - A10) chọn ra. Để kiểm chứng nhanh điều này ta cần nhớ rằng với các bit địa chỉ A10 ta chọn ra được các mạch nhớ 1KB và với bit A11 ta chọn ra được các mạch nhớ 2 KB các mãng nhớ này nằm rãi rác trong không gian của bộ nhớ.

✓ **Thực hiện bộ giải mã dùng mạch giải mã kiểu 74LS138 :**

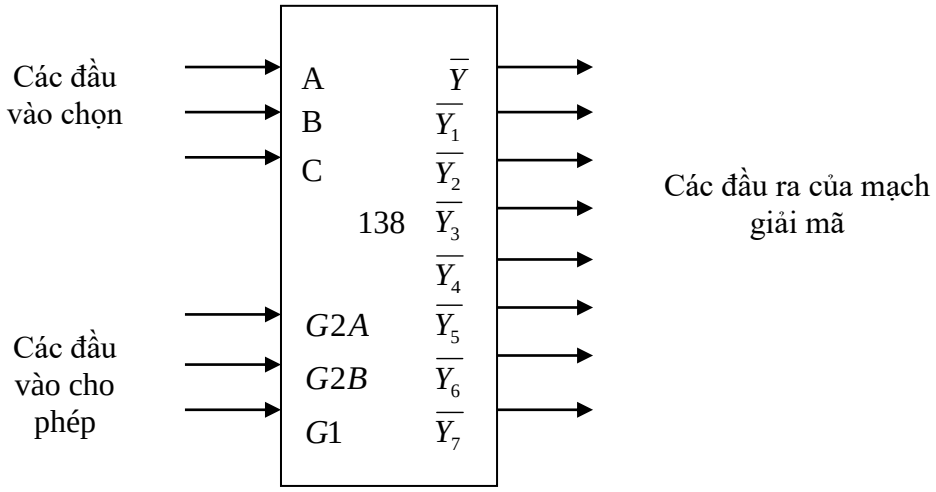
Khi ta muốn có nhiều đầu ra chọn vỏ từ bộ giải mã mà vẫn dùng các mạch logic đơn giản thì thiết kế sẽ trở nên rất cồng kềnh do số lượng các mạch cửa tăng lên. Trong trường hợp như vậy ta thường sử dụng các mạch giải mã có sẵn. Một trong các mạch giải mã hay dược sử dụng là 74LS138, mạch giải mã 3-8 (hình 5.18)



Hình 5.19. sơ đồ bộ giải mã dùng 74LS138.

Ví dụ

Giả thiết ta cần dùng riêng vùng nhớ 64 KB có địa chỉ F0000H-FFFFFH. Cho các mạch nhớ EPROM 8 KB (dùng 8x2764-15, t_{ac} = 150ns). Hãy dùng mạch giải mã kiểu 74LS138 để thực hiện.



Hình 5.18. sơ đồ khối của 74LS138

Giải

Ta có thể dùng mạch 74LS138 và bố trí các đường địa chỉ như trên hình 5.19.

Ở thí dụ này ta cần lưu ý rằng bit địa chỉ A13 có khả năng chọn ra các mãng nhớ 8 KB nằm rãi rác trong không gian nhớ của 8088. chính vì vậy ta dùng nó như đầu vào A của 74LS138, cùng với các địa chỉ bit A14 và A15 tại các chân B và C ta sẽ chọn ra được 8 vùng nhớ liền nhau, mỗi vùng có dung lượng là 8 KB hay toàn vùng là 64 KB này ở phần cao nhất trong không kgian nhớ.

Điều này có thể đạt được một cách dể dàng bằng việc dùng tổ hợp 4 bit địa chỉ cao nhất A16 - A19 ở trạng thái 1 (A19 thông qua đàu vào G1 và A16 -A18 qua 1 mạch NAND 3 đầu vào để đưa đến G2A của mạch 74LS138).

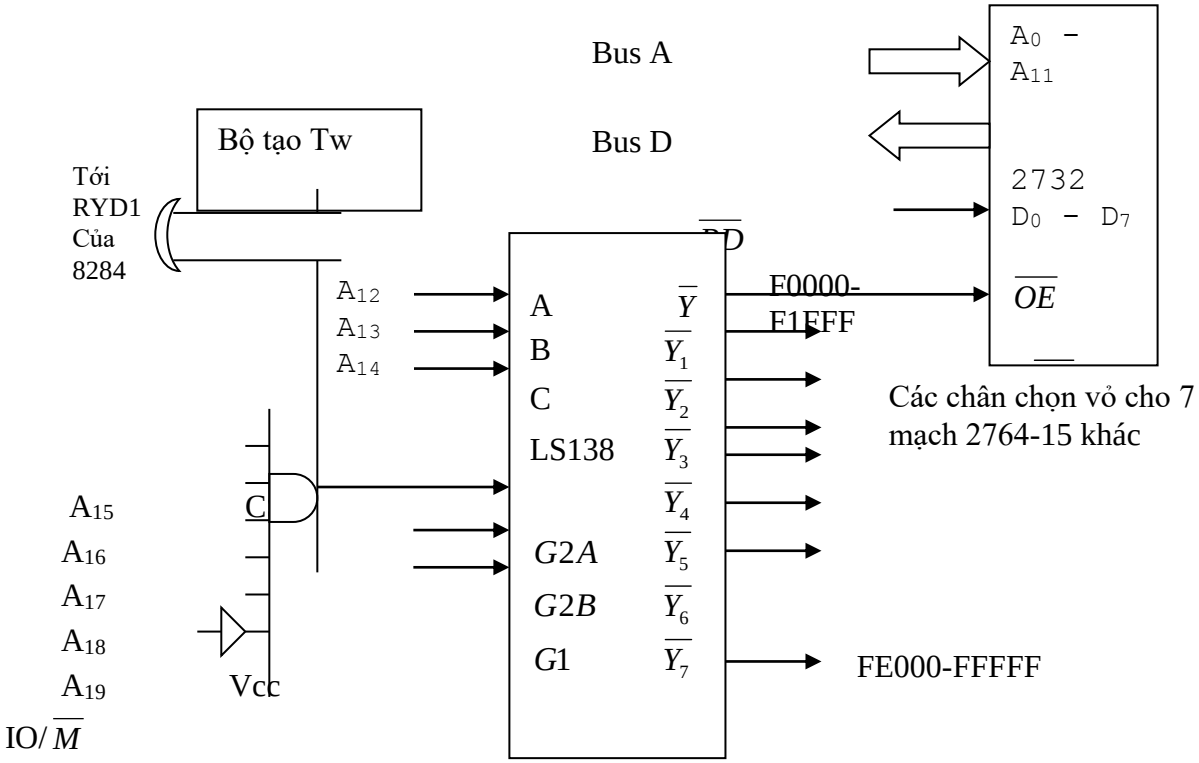
Tại thí dụ này ta thấy mạch giải mã có sẵn 74LS138 có số lượng đầu vào địa chỉ và đầu vào cho phép hạn chế. Nếu ta có số lượng đầu vào cho địa chỉ lớn mà ta lại phải giải mã đầy đủ để thực hiện bộ giải mã đã hoàn chỉnh ta vẫn phải dùng thêm các mạch logic phụ. Đây cũng là lý do để người ta thay thế các bộ giải mã kiểu này bằng các bộ giải mã dùng PROM hoặc PLA (programable logic array) với ưu điểm chính là chúng có rất nhiều đầu vào cho các bit địa chỉ và vì thế rất thích hợp trong các hệ vi xử lý sau này với không gian địa chỉ lớn.

5.2.3. Phối ghép CPU 8088 - 5MHz với bộ nhớ

Sau khi đã giới thiệu các phương pháp giải mã cho mạch nhớ trong phần này ta sẽ giới thiệu cách phối ghép 8088 với bộ nhớ. Có thể nói tổng quát rằng nếu -không có xung đột giữa tốc độ thâm nhập mạch nhớ và tốc độ CPU thì việc phối ghép CPU với bộ nhớ đơn giản chỉ là việc giải mã địa chỉ trong mạch nhớ. Trong phần lớn các trường hợp điều này có thể đúng cho các mạch nhớ RAM và các mạch nhớ EPRAM có thời gian thâm nhập nhỏ hoen hoặc bằng 250 ms, cách phối ghép CPU với cac mạch này về cơ bản là giống nhau. Đối với các mạch nhớ EPROM như 2716, 2732 ... loại tốc độ 450 ms khi thực hiện phối ghép với 8088 - 5MHz thì cần phải tính toán thận trọng hơn.

Trong phần đầu của chương này ta đã tính được rằng muốn phối ghép được với CPU 8088 - 5MHz thì các mạch nhớ phải có thời gian thâm nhập dài nhất là cỡ 450 ms, vì vậy nếu ta muốn ghép EPROM 2732 tốc độ 450ms vào bộ nhớ thì chắc chắn phải có cách để báo cho CPU xen thêm chu kỳ đợi trên hình 5.21 là sơ đồ mạch phối ghép EPROM 2732 có thêm

mạch NAND để tạo tín hiệu cho phép mạch giải mã và tín hiệu yêu cầu đợi để đưa đến chân RDI1 của 8284 (đã được trình bày ở phần trước). Bằng cách này mỗi khi CPU đợi EPROM 2732 tốc độ chậm thì 1 chu kỳ đợi sẽ được tự động xen thêm.



Hình 5.21. Phối ghép EPROM 2732 - 450 ns với CPU 8088 - 5MHz.

Việc phối ghép SRAM với 8088 thường đơn giản hơn so với EPROM vì SROM có tốc độ nhanh nên không cần mạch xen thêm chu kỳ đợi.

5.3. Phối ghép 8088 với thiết bị ngoại vi

5.3.1. Các kiểu phối ghép vào/ra

Đối với 8088 (hay họ 80x86 nói chung) có 3 cách phối ghép CPU với các thiết bị ngoại vi (các cổng vào/ra, I/O) : a) thiết bị ngoại vi có không gian địa chỉ tách biệt với bộ nhớ và b) thiết bị ngoại có không gian địa chỉ chung với bộ nhớ.

✓ **Thiết bị vào/ra có không gian địa chỉ tách biệt**

Trong cách phối ghép này, bộ nhớ được độc quyền dùng không gian 1MB mà CPU dành cho nó. Các thiết bị ngoại vi (các cổng) sẽ được dành riêng một không gian 64KB cho mỗi loại cổng vào hoặc ra. Tất nhiên ta phải dùng tín hiệu IO/M, và các lệnh trao đổi dữ liệu một cách thích hợp cho mỗi không gian đó

✓ **Thiết bị vào/ra và bộ nhớ có chung không gian địa chỉ**

Trong cách phối ghép này, bộ nhớ và thiết bị ngoại vi cùng chia nhau không gian địa chỉ 1MB mà CPU 8088 có khả năng địa chỉ hóa. Các thiết bị ngoại vi sẽ chiếm một vùng nào đó trong không gian 1MB, phần còn lại là của bộ nhớ. Tất nhiên trong trường hợp này ta dùng chung tín hiệu IO/$\overline{M}$ =0 và lệnh trao đổi dữ liệu kiểu lệnh MOV cho cả bộ nhớ và thiết bị ngoại vi

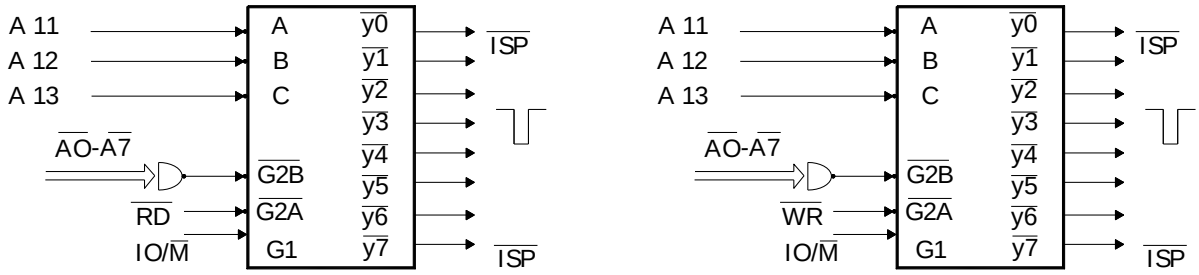
5.3.2. Giải mã địa chỉ cho thiết bị vào/ra

Việc giải mã địa chỉ cho thiết bị vào/ra cũng gần giống như giải mã địa chỉ cho mạch nhớ. Ta sẽ nhấn mạnh ở đây việc giải mã địa chỉ cho các cổng. Thông thường các cổng có địa

chỉ 8 bit tại A0-A7, trong một số hệ vi xử lý khác (như các máy IBM PC) các cổng có 16 bit tại A0 - A15. Tuỳ theo độ dài của toán hạng trong lệnh là 8 hay 16 bit ta sản xuất có 1 cổng 8 bit có địa chỉ liên nhau để tạo nên từ với độ dài tương ứng. Trong thực tế ít có hệ sử dụng hết 256 cổng vào/ra khác nhau nên ta chỉ xét ở đây các bộ giải mã địa chỉ 8 bit A0-A7 và mạch giải mã thông dụng có sẵn 9như 74LS138 chẳng hạn) để tạo ra các xung chọn thiết bị.

Các mạch giải mã đơn giản có thể tạo được từ mạch NAND

Trong trường hợp cần nhiêu xung chọn ở đầu ra cho các cổng vào/ra có địa chỉ liên tiếp, ta có thể dùng các mạch giải mã có sẵn kiểu 74LS138. Thí dụ trên hình dưới trình bày 2 mạch tương tự nhau dùng 74LS138 để giải mã địa chỉ cho 8 cổng vào và 8 cổng ra. Trên cơ sở mạch này ta cũng có thể phối hợp với cả hai tín hiệu đọc và ghi để tạo ra tín hiệu chọn cho việc đọc/ghi từng cổng vào/ra ra cụ thể.

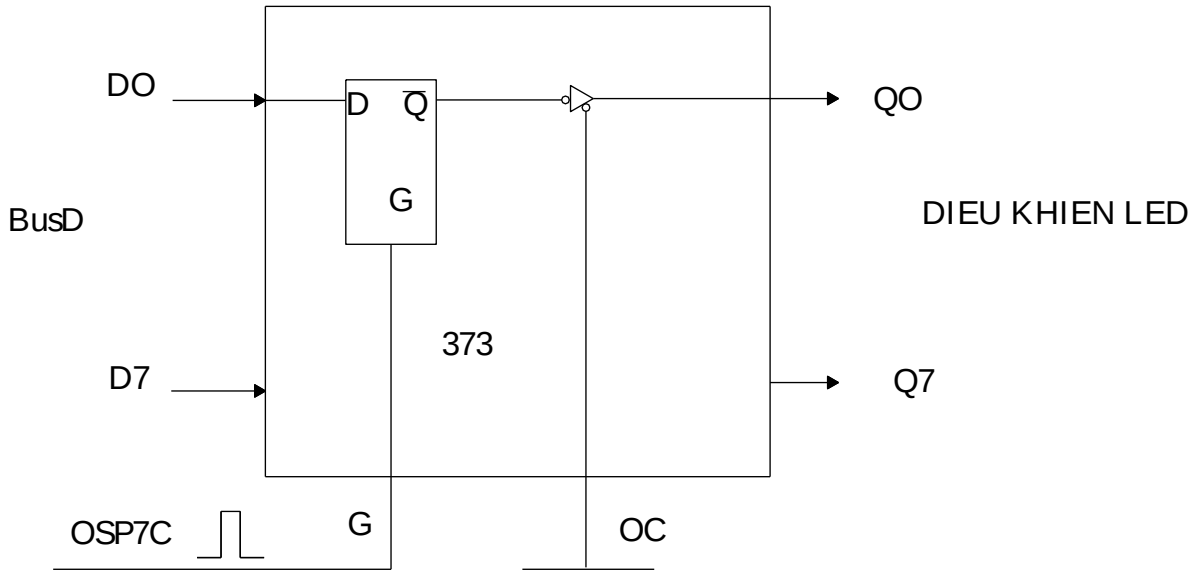


Các bộ giản mã với địa chỉ 00-07 cho a) cổng vào ; b) cổng ra.

5.3.3. Các mạch cổng đơn giản

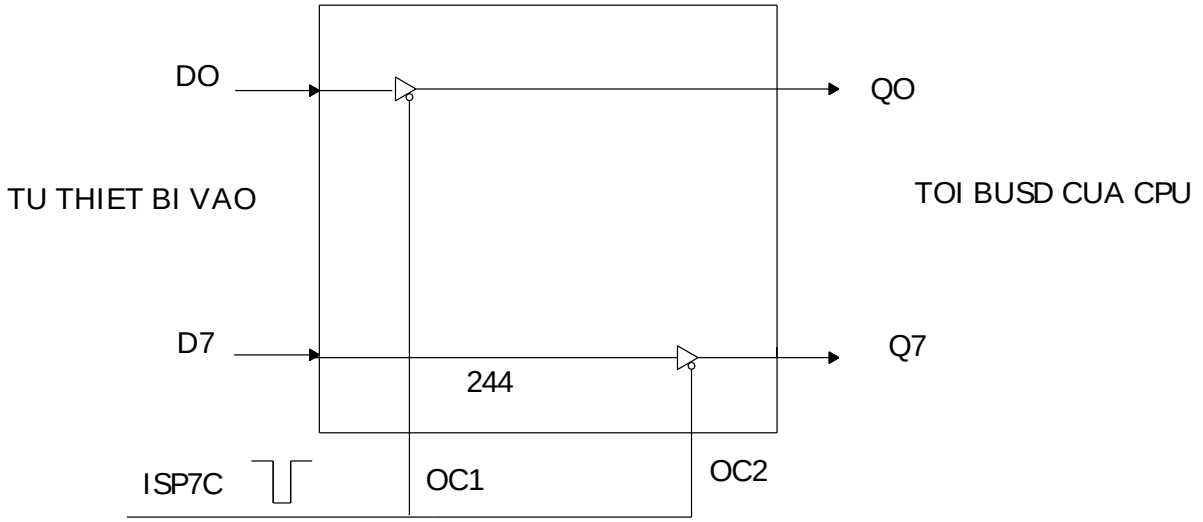
Trong thực tế có rất nhiều vi mạch tổ hợp cỡ vừa có thể được dùng làm cổng phối ghép với bộ vi xử lý để vào/ra dữ liệu. Các mạch này thường được cấu tạo nên từ các mạch chốt 8 bit có đầu ra 3 trạng há (74LS373: kích theo mức; 74LS374: kích theo sườn), các mạch khuếch đại đệm 2 chiều 8 bit đầu ra 3 trạng thái (74LS245). Chúng được dùng trong các phối ghép đơn giản để làm cho CPU và thiết bị ngoại vi hoạt động tương thích với nhau, ví dụ như để đệm bus hoặc các mạch cổng để tạo ra các tín hiệu móc nối...

Trên hình 5.32 là ví dụ một mạch phối ghép đơn giản dùng mạch 74LS373 để đưa tín hiệu ra điều khiển các đèn LED bằng lệnh OUT 7CH, AL.



Hình 5.32. Một mạch phối ghép ra đơn giản dùng 74LS373

Trên hình 5.33 là ví dụ một mạch phối ghép đơn giản dùng mạch 74LS244 để đọc tín hiệu từ thiết bị ngoại vi vào CPU bằng lệnh IN Al,7CH.



Hình 5.33. Một mạch phối ghép ra đơn giản dùng 74LS244

Ngoài các mạch phối ghép đơn giản ở trên, trong thực tế còn có các mạch phối ghép lập trình được dùng cho các công việc tổ chức vào/ra dữ liệu phức tạp hơn.

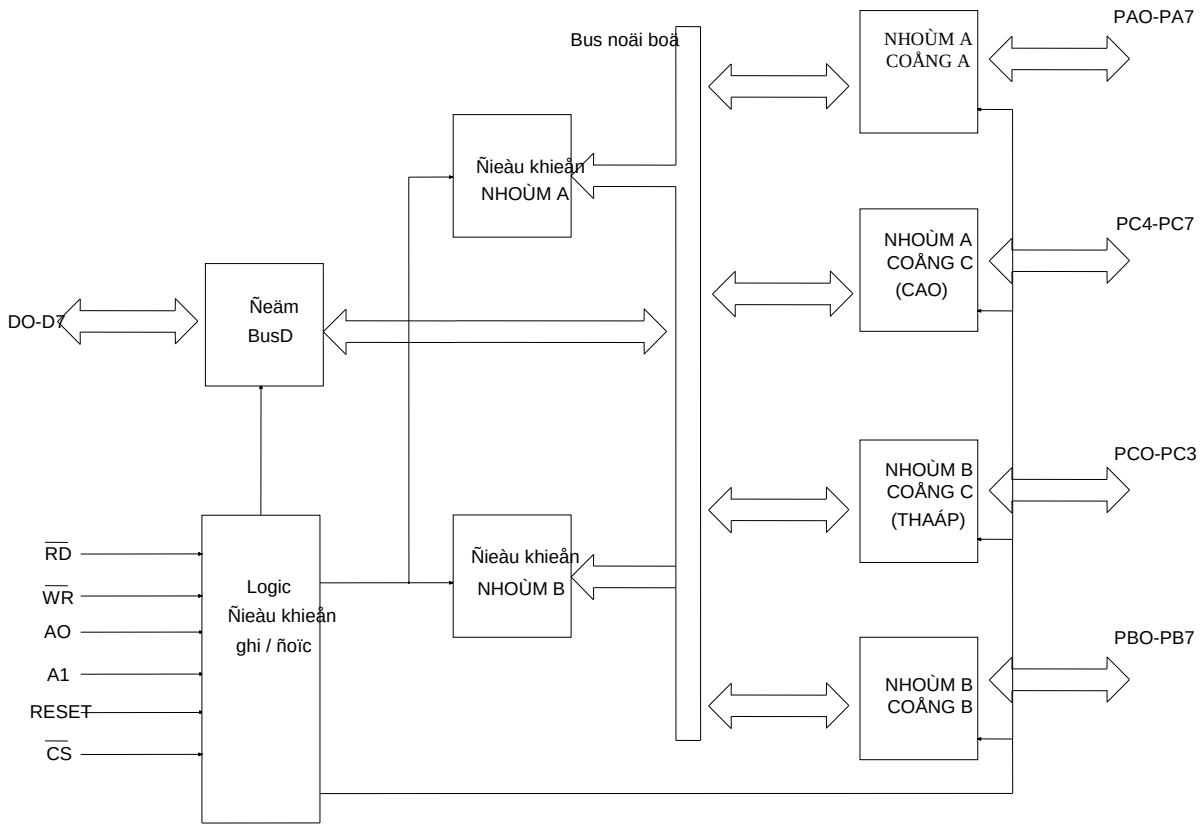
5.3.4. Mạch phối ghép vào/ra song song lập trình được 8255A

Mạch 8255A thường được gọi là mạch phối ghép vào/ra lập trình được (program mable peripheral interface, PPI). Do khả năng mềm dẻo trong các ứng dụng thực tế nó là mạch phối ghép được dùng rất phổ biến cho các hệ vi xử lý 8-16 bit. Sơ đồ khối mô tả chức năng bên trong của 8255A được thể hiện trên hình 5.34.

Các chân tín hiệu của 8255A có ý nghĩa khá rõ ràng. Chân Reset phải được nối với tín hiệu reset chung của toàn hệ (khi reset thì các cổng được định nghĩa là cổng vào để không gây ra sự cố cho các mạch điều khiển). CS được nối với mạch tạo xung chọn thiết bị để đặt mạch 8255A vào một địa chỉ cơ sở nào đó. các tín hiệu địa chỉ A0, A1 sẽ chọn ra 4 thanh ghi bên trong 8255A 1 thanh ghi để ghi từ điều khiển cho hoạt động của 8255A (viết tắt là CWR, control word register) và 3 thanh ghi khác ứng với các cổng (port) là PA, PB, PC để ghi/đọc dữ liệu (xem bảng 5.7). Theo bảng này ta nhận thấy địa chỉ cho PA cũng chính là địa chỉ cơ sở của 8255A.

Có 2 loại từ điều khiển cho 8255A:

+ Từ điều khiển định nghĩa cấu hình cho các cổng PA, PB, PC
+ Từ điều khiển lập/xóa từng bit ở đầu ra của PC



Hình 5.34. Sơ đồ khối cấu trúc bên trong của 8255A

$\overline{CS}$	A1	A0	Chọn ra
1	x	x	không chọn
0	0	0	PA
0	0	1	PB
0	1	0	PC
0	1	1	CWR

✓ **Từ điều khiển định nghĩa cấu hình**

Dạng thức của từ điều khiển để định cấu hình được thể hiện trên hình 5.35.

• Từ điều khiển lập/xoá bit ra PCi:

Dạng thức của từ điều khiển để lập/xoá PCi được thể hiện trên hình 5.36.

Lập/xoá

0	0	0	0				S/R

Mã hoá PCi 1: lập PCi

000 : PC_0 0: xoá PCi

...

111 : PC_7

Như ta thấy ở trên, các chế độ (mode) làm việc của mạch cổng 8255A có thể được định nghĩa bằng từ điều khiển CWR. Cụ thể 8255A có 4 chế độ làm việc:

+ **Chế độ 0: "Vào/ra cơ sở"** (còn gọi là "vào ra đơn giản"). Trong chế độ này mỗi cổng PA. PB. PC_H và PC_L đều có thể được định nghĩa là các cổng vào hoặc ra.

+ **Chế độ 1: "Vào/ra có xung cho phép".** Trong chế độ này mỗi cổng PA. PB có thể được định nghĩa thành cổng vào hoặc ra với tín hiệu móc nối (handshaking) do các bit tương ứng của cổng PC trong cùng nhóm đảm nhiệm.

+ **Chế độ 2: "Vào ra 2 chiều".**Trong chế độ này chỉ riêng cổng PA có thể được định nghĩa thành cổng vào/ra 2 chiều với các tín hiệu móc nối do các bit của cổng PC đảm nhiệm. Cổng PB có thể làm việc trong chế độ 0 hoặc 1.

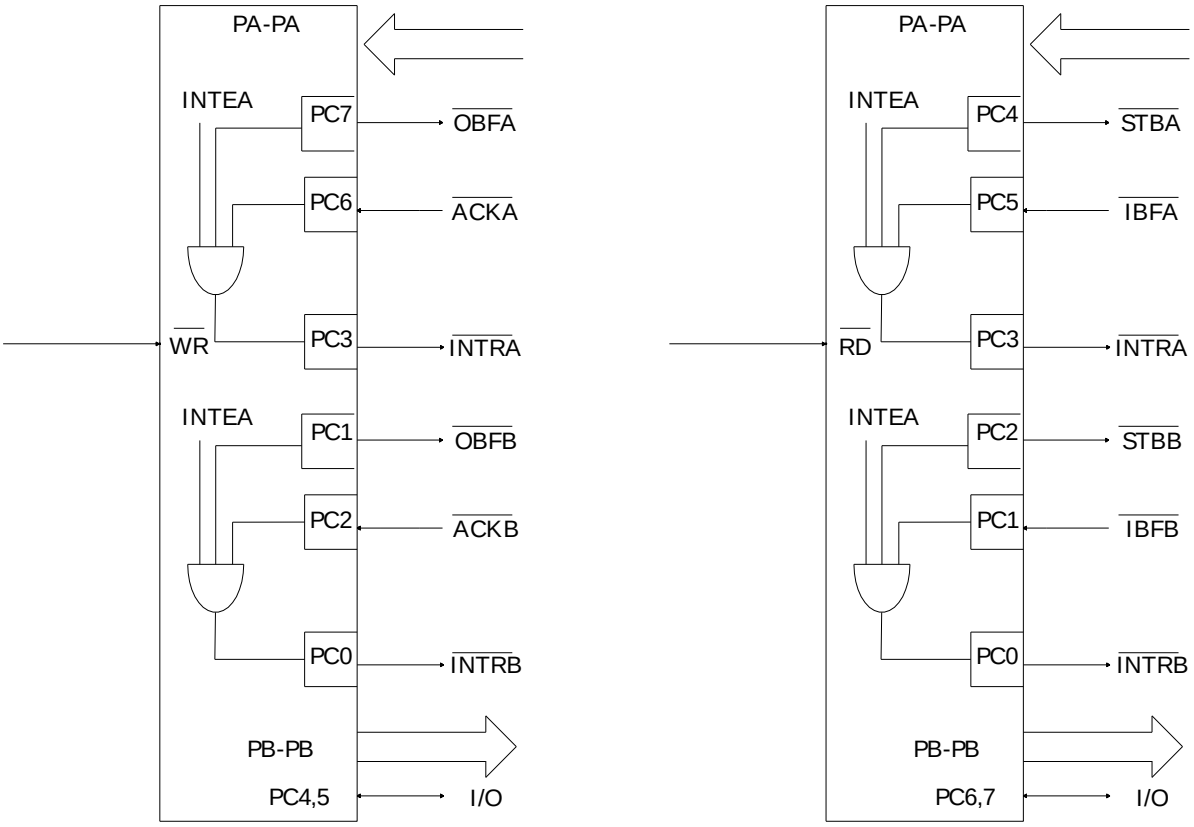
+ **Lập/xoá các bit PCi:** chế độ này có thể thấy rõ trên hình 5.36.

Sau đây ta sẽ giới thiệu cụ thể các chế độ làm việc 0, 1 và 2 của 8255A.

Chế độ 0: "Vào/ra cơ sở"

Trong chế độ này, bốn cổng PA, PB, PC_H, PC_L đều có thể được định nghĩa là cổng vào hoặc cổng ra. Như vậy, với tổ hợp tất cả các khả năng vào/ra cho 4 cổng đó ta có được 16 cấu hình khác nhau.

Chế độ 1: "Vào/ra có xung cho phép"



Hình 5.37. Mạch 8255A ở chế độ 1 và các tín hiệu trạng thái.

Để đơn giản ta coi PA và PB cùng được định nghĩa là cổng ra hoặc cổng vào.

Ra dữ liệu trong chế độ 1 (hình 5.37 a)

Ở đây PA và PB cùng được định nghĩa là cổng ra và có tín hiệu móc nối tương đương nhau cho việc trao đổi dữ liệu. Ta chỉ cần giới thiệu ở đây các tín hiệu cho PA, các tín hiệu cho PB cũng tương tự:

+ $\overline{OBFA}$ (Đệm ra của PA đầy). Tín hiệu báo cho thiết bị ngoại vi biết CPU đã ghi dữ liệu vào cổng để chuẩn bị đưa ra. Tín hiệu này thường được nối với $\overline{STB}$ của thiết bị nhận.

+ $\overline{ACKA}$ (Trả lời đã nhận được dữ liệu). Đây là tín hiệu của thiết bị ngoại vi cho biết là nó đã nhận được dữ liệu từ PA của 8255A.

+ INTRA (Yêu cầu ngắt từ PA). Đây là kết quả thu được từ quan hệ giữa các tín hiệu khác của 8255A trong quá trình đối thoại với thiết bị ngoại vi, nó được dùng để phản ảnh yêu cầu ngắt của PA tới CPU (xem biểu đồ quan hệ giữa các tín hiệu trong hình 5.38).

+ INTEA là tín hiệu của một mạch lật bên trong 8255A để cho phép/cấm yêu cầu ngắt INTRA của PA. INTEA được lập/xoá thông qua bit PC6 của PC.

Các tín hiệu *đối thoại-trạng thái* kể trên đều có thể lấy trực tiếp được từ các chân tương ứng của vi mạch hoặc được đọc vào CPU thông qua việc đọc cổng PC (xem hình 5.37c để thấy trạng thái đọc được khi 8255A được định nghĩa ở chế độ 1).

Vào dữ liệu trong chế độ 1 (hình 5.37 b)

Ở đây PA và PB được định nghĩa là cổng vào và có các tín hiệu móc nối tương đương nhau cho việc trao đổi dữ liêu. Ta chỉ cần giới thiệu ở đây các tín hiệu cho PA, các tín hiệu cho PB cũng tương tự:

+ $\overline{STB}$ (Cho phép chốt dữ liệu): Khi dữ liệu đã sẵn sàng để được đọc vào bằng PA, thiết bị ngoại vi phải dùng $\overline{STB}$ để báo cho 8255A biết mà chốt dữ liệu.

+ IBF (Đệm vào đầy): Sau khi 8255A chốt được dữ liệu do thiết bị ngoại vi đưa đến nó đưa ra tín hiệu IBF để báo cho thiết bị ngoại vi biết là đã chốt xong.

+ INTRA (yêu cầu ngắt từ cổng PA): Tín hiệu để báo cho CPU biết là đã có dữ liệu sẵn sàng để đọc từ PA. Đây là kết quả thu được từ quan hệ giữa các tín hiệu khác của 8255A trong quá trình đối thoại với thiết bị ngoại vi (xem biểu đồ quan hệ giữa các tín hiệu trong hình 5.39).

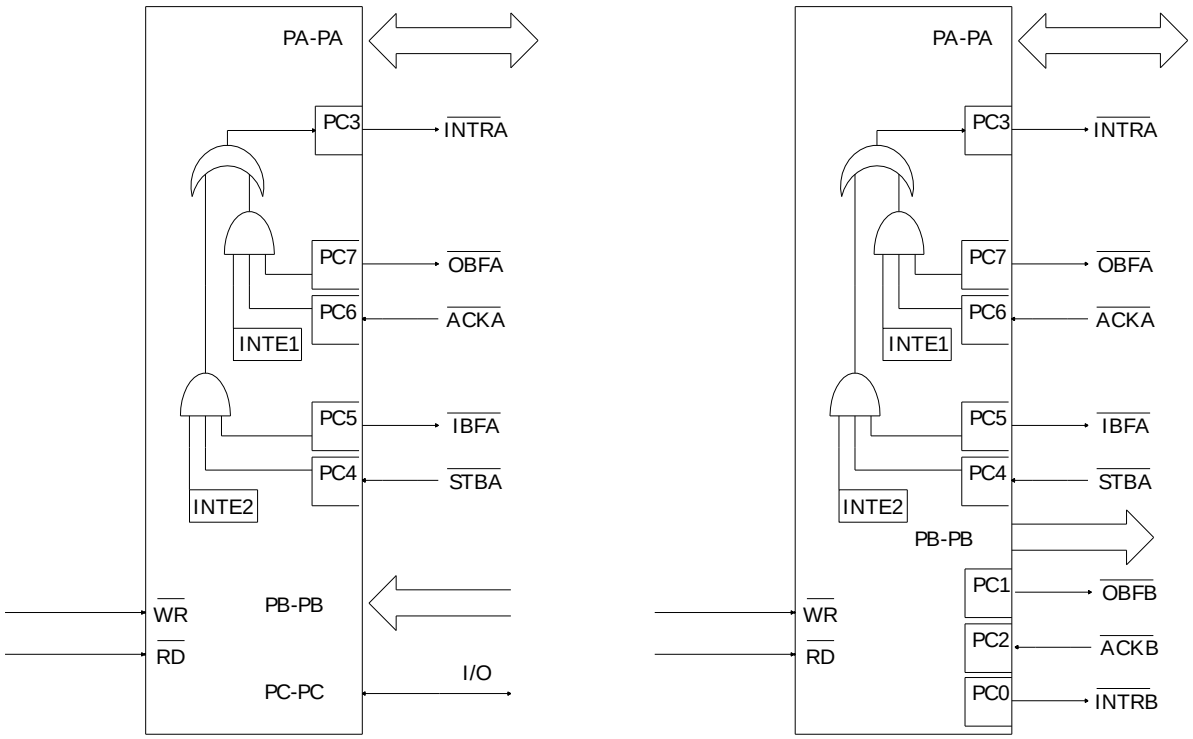
+ INTEA là tín hiệu của một mạch lạt bên trong 8255A để cho phép/cấm yêu cầu ngắt INTRA của PA. INTEA được lập/xoá thông qua bit PC4 của PC.

Một số trong các tín hiệu *đối thoại-trạng thái* kể trên đều có thể lấy được trực tiếp từ chân tương ứng của vi mạch hoặc có thể được đọc vào CPU thông qua việc đọc PC (xem hình 5.37d).

Cac tín hiệu *đối thoại-trạng thái* có thể được sử dụng làm tín hiệu móc nối cho các kiểu vào/ra dữ liệu bằng cách ngắt CPU hay bằng cách thăm dò trạng thái sẵn sàng của thiết bị ngoại vi (sẽ được trình bày trong các phần sau).

Chế độ 2: "Bus 2 chiều" (hình 5.40)

Trong chế độ này chỉ riêng cổng PA được định nghĩa để làm việc như một cổng 2 chiều có các tín hiệu móc nối do một số bit của PC đảm nhiệm, còn PB thì có thể làm việc ở chế độ 1 hoặc 0 tuỳ theo các bit điều khiển trong CWR. Các chân tín hiệu còn lại của PC có thể được định nghĩa để làm việc như các chân vào hoặc ra, hoặc phục vụ cho cổng PB.



Hình 5.40. 8255A ở chế độ "Bus hai chiều".

Một số tín hiệu móc nối đặc biệt cần giới thiệu của PA gồm:

+ INTRA: Yêu cầu ngắt cho dữ liệu theo hai chiều vào và ra.

+ INTE1 và INTE2 là 2 tín hiệu của 2 mạch lật bên trong 8255A để cho phép/cấm yêu cầu ngắt của cổng PA. Các bit này được lập/xoá bởi các bit PC6 và PC4 của cổng PC.

Biểu đồ thời gian của các tín hiệu móc nối của 8255A được biểu hiện trên hình 5.41.

Một số trong các tín hiệu *đối thoại-trạng thái* kể trên đều có thể lấy được trực tiếp từ chân tương ứng của vi mạch hoặc có thể đọc được vào CPU từ cổng PC và cho phép điều khiển việc trao đổi dữ liệu bằng cách thăm dò các tín hiệu này.

Khi dùng 8255A trong chế độ bus 2 chiều để trao đổi dữ liệu theo cách thăm dò ta phải kiểm tra xem bit $\overline{OBFA}$ có bằng 1 (đệm ra rỗng) hay không trước khi dùng lệnh IN để đọc dữ liệu vào từ cổng PA.

Sau đây là một vài ví dụ về cách lập trình cho 8255A.

● **Ví dụ 1**: Giả thiết mạch 8255A có các địa chỉ sau:

Địa chỉ	Thanh ghi (cổng)
7Ch	PA
7Dh	PB
7Eh	PC
7Fh	CWR

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

PA: Ra

PB: Vào

PC_H: Ra

PC_L: Vào

Sau đó đọc các giá trị dữ liệu có tại PB rồi đưa ra PA và

PC_L rồi đưa ra PC_H.

Giải: Có thể định nghĩa các hằng cho cổng, thanh ghi từ điều khiển và cho từ điều khiển nhờ lệnh giả EQU:

```
PA      EQU     7CH
PB      EQU     7DH
PC      EQU     7EH
CWR     EQU     7FH
CW      EQU     83H     ; từ điều khiển cho yêu
                        ; cầu trên 83H=10000011
```

và sau đó dùng hằng đó vào việc định nghĩa cấu hình cho 8255A:

```
MOV AL, CW          ; từ điều khiển trong AL
OUT CWR, AL         ; đưa CW vào CWR
IN AL, PB           ; đọc cổng PB
OUT PA, AL          ; đưa dữ liệu đọc được ra cổng PA
IN AL, PC           ; đọc cổng PCL
MOV CL, 4           ; số lần quay AL
ROL AL, CL          ; chuyển 4 bit thấp thành 4 bit cao
OUT PC, AL          ; đưa dữ liệu đọc được ra cổng PCH
```

• **Ví dụ 2**: Lập trình để bit PC4 của 8255A ở ví dụ trên tạo ra 256 xung với T=50ms, độ rộng 50%. Giả thiết đã có sẵn chương trình con trễ 25ms là TRE25MS.

Giải: Sử dụng các ký hiệu ở ví dụ trên và các định nghĩa mới

```
PC4ON       EQU   09H   ; từ điều khiển để PC4=1 (On): 00001001B
PC4OF       EQU   08H   ; từ điều khiển để PC4=0 (Off): 00001000B
```

ta có mẫu chương trình sau:

```
            MOV CX, 100H        ; số xung phải tạo ra
            CLI                 ; cấm ngắt để yên tâm mà
                                ; tạo xung
Lap:        MOV AL, PC4ON       ; từ điều khiển cho PC4=1.
            OUT PC, AL          ; PC4=1
            CALL TRE25MS        ; kéo dài 25ms.
            MOV AL,PC4OFF       ; PC4=0
            CALL TRE25MS        ; Lặp cho đủ số xung
            STI                 ; cho phép ngắt trở lại
            ...
```

Trong ví dụ trên ta chú ý lệnh CLI (để cấm ngắt) và lệnh STI (để cho phép ngắt) ở đầu và cuối đoạn chương trình tính thời gian tạo ra dãy xung. Điều này là cần thiết vì ta dùng

chương trình để tạo ra các khoảng thời gian và vì thế ta muốn các yêu cầu ngắt (nếu có) không ảnh hưởng tới việc tạo ra các khoảng thời gian đó.

• **Ví dụ 3**: có mạch 8255A với địa chỉ cơ sở là 30H nối với các phần tử ngoại vi đơn giản (hình 5.42). Lập trình để có $U_1>U_2$ thì đọc trạng thái công tắc K, nếu K đóng thì cho đèn LED tắt, K mở thì cho đèn LED sáng.

Giải:

Ta phải định nghĩa các hằng và từ điều khiển để định nghĩa cấu hình: PA là vào, PB là vào và PC_H là ra. Khi $U_1>U_2$ thì ta đọc được PB7=1 (còn khi $U_1 \le U_2$ thì PB7=0). Lúc này ta phải đọc trạng thái công tắc K để điều khiển đèn LED, nếu K đóng thì ta đọc được PB7=0 và ta phải đưa ra PC4=) để tắt đèn.

Định nghĩa các hằng

```
PA      EQU     30H
PB      EQU     31H
PC      EQU     32H
CWR     EQU     33H
CW      EQU     92H     ; từ điều khiển cho yêu
                        ; cầu trên 92H=10010010
```

đoạn chương trình còn lại là:

```
        MOV AL, CW      ; từ điều khiển trong AL
        OUT CWR, AL     ; đưa CW vào CWR
DocPB:  IN AL, PB       ; đọc cổng PB
        AND AL, 80H; PB7=1?
        JZ  DocPB       ; đọc lại
        IN AL, PA       ; đọc cổng PA
        And AL, 01H     ; khoá K đóng (PA1=0)?
        JZ Tat          ; đúng, tắt đèn
        MOV CL, 4
        ROL AL, CL      ; đổi bit0 lên bit4
        OUT PC, AL      ; đưa tín hiệu thắp đèn
Tat:    OUT PC, AL      ; đưa tín hiệu tắt đèn
Ra:...
```

CHƯƠNG 6 : VÀO RA DỮ LIỆU BẰNG CÁCH THĂM DÒ

6.1. Giới thiệu chung về các phương pháp điều kiển vào/ra dữ liệu

Sau khi đã trình bày về các mạch thường dùng cho việc phối ghép CPU với thiết ngoại vi ta sẽ tiến hành nghiên cứu các phương thức điều khiển việc trao đổi dữ liệu. Các mạch phối ghép vào / ra đã trình bày trước đây có thể được ứng dụng để phục vụ cho mục đích này.

Nói chung người ta phân biệt ra 3 phương pháp điều khiển vào/ra dữ liệu:

- vào/ra dữ liệu điều khiển bằng cách thăm dò trạng thái sẵn sàng của thiết bị ngoại vi.
- vào/ra dữ liệu điều khiển bằng cách ngắt bộ vi xử lý.
- vào/ra dữ liệu điều khiển bằng phần cứng phụ để thâm nhập trực tiếp vào bộ nhớ.

Mỗi phương pháp điều khiển vào/ra dữ liệu nói trên có những đặc điểm khác nhau và sẽ được ứng dụng trong các hoàn cảnh khác nhau. Một trong những cách điều khiển đơn giản nhất mà chúng ta xem xét trong chương này là phương pháp thăm dò (polling) trạng thái sẵn sàng làm việc của thiết bị ngoại vi trước khi thực hiện vào/ra dữ liệu. Các phương pháp điều khiển vào ra dữ liệu khác sẽ được giới thiệu trong các chương sau.

6.2. Vào/ra dữ liệu bằng phương pháp thăm dò.

Vấn đề điều khiển vào/ra dữ liệu sẽ trở thành rất đơn giản nếu thiết bị ngoại vì lúc nào cũng sẵn sàng chờ để làm việc với CPU. Ví dụ, bộ phận do nhiệt độ số (như là một thiết bị vào) lắp sẵn trong một hệ thống điều khiển lúc nào cũng có thể cung cấp số đo về nhiệt độ của đối tượng cần điều chỉnh, còn một bộ đèn LED 7 nét (như là một thiết bị ra) dùng để chỉ thị một giá trị nào đó của một đại lượng vật lý nhất định trong hệ thống nói trên thì lúc nào cũng có thể biểu hiện thông tin đó. Như vậy khi CPU muốn có thông tin về nhiệt độ của hệ thống thì nó chỉ việc đọc cổng phối ghép với bộ đo nhiệt độ, và nếu CPU muốn biểu diễn thông tin vừa đọc được trên đèn LED thì nó chỉ việc đưa tín hiệu điều khiển tới đó mà không cần phải kiểm tra xem các thiết bị này có đang sẵn sàng làm việc hay không.

Tuy nhiên trong thực tế không phải lúc nào CPU cũng làm việc với các đối tượng "liên tục sẵng sàng" như trên. Thông thường khi CPU muốn làm việc với một đối tượng nào đó, trước tiên nó phải kiểm tra xem thiết bị đó có đang ở trạng thái sẵn sàng làm việc hay không nếu có thì nó mới thực hiện vào việc trao đổi dữ liệu. Như vậy, nếu làm việc theo phương thức thăm dò thì thông thường CPU phải được dành riêng cho việc trao đổi dữ liệu vì nó phải liên tục kiểm tra trạng thái sẵn sàng của thiết bị ngoại vi thông qua các tín hiệu móc nối (handshake signal). Các tín hiệu này được lấy từ các mạch phối, do người thiết kế tạo ra, để cho chương trình thăm dò hoạt động trên đó.

Sau đây là thí dụ một cách tạo ra tín hiệu móc nối và lưu đồ thuật toán của chương trình dùng cho việc trao đổi dữ liệu giữa CPU và thiết bị ngoại vi (hình 6.1)

Trong thí dụ này để cho vấn đề đơn giản, ta giả thuyết CPU chỉ làm việc với 1 thiết bị vào và 1 thiết bị ra. Việc tổ chức phối ghép được thực hiện trên các mạch IC cỡ vừa để ta để theo giỏi các tín hiệu.

Một cổng vào số 0 (có địa chỉ 00) được dùng để đọc trạng thái sẵn sàng của 2 thiết bị ngoại vi nói trên. Tín hiệu sãn sàng của thiết bị ngoại vi số 1 (cổng vào 01) được đặt vào bit D0, tín hiệu sẵn sàng của thiết bị ngoại vi số 2 (cổng ra 02) được đặt vào bit D1. Các bit này sẽ có giá trị 1 khi thiết bị ngoại vi tương ứng ở trạng thái sẵn sàng làm việc với CPU và chúng sẽ được đưa vào bus dữ liệu khi CPU đọc nó bằng lệnh đọc cổng vào số 0. Chương trình trao đổi dữ liệu sẽ kiểm tra các bit báo sẵn sàng này và sẽ có các đáp ứng phù hợp.

Mô tả hoạt động của phần mạch vào dữ liệu

Khi thiết bị vào số 1 có byte số liệu cần trao đổi, nó đưa ra xung STB để cho phép mạch chốt 8 bit lấy byte dữ liệu đó đồng thời kích cho mạch lật D (mạch tạo tín hiệu sẵn sàng) làm việc. CPU sẽ thăm dò trạng thái sẵn sàng của thiết bị vào số 1 qua bit DO khi nó đọc

cổng 00. Đến khi CPU đọc 1 byte dữ liệu vào thì đồng thời nó xóa luôn mạch tạo trạng thái sẵn sàng để chuẩn bị cho lần làm việc tới với 1 byte dữ liệu khác.

Tương tự ta cũng có thể dễ dàng thấy được cách hoạt động của phần mạch thứ hai của hình 6.1 với chức năng đưa dữ liệu ra thiết bị số 2.

Lưu đồ thuật toán cho chương trình vào/ra dữ liệu của cổng số 01/02 theo sơ đồ móc nối trên hình 6.2 được thể hiện ở hình 6.2 a) và b) đường liền nét.

• Ví dụ

Lập chương trình theo lưu đồ trên hình 6.2 để thực hiện việc đọc vào dữ liệu mỗi khi cổng 01 báo sẵn sàng rồi sử lý số liệu đó. Có 100 số liệu phải đọc như vậy (giả thuyết có sẵn chương trình con xử lý dữ liệu có tên Xuly)

Giải

Cứ mỗi lần nhận được tín hiệu báo sẵn sàng của cổng 01 ta gọi chương trình con Xuly. Thân của chương trình hoàn thành công việc nói trên (theo lưu đồ trên hình 6.2a đường liền nét) có thể có cấu trúc sau :

```
        MOV CX, 100         ; số dữ liệu phải đọc trong cx
Lap : IN AL, 0              ; đọc cổng 00 để kiểm tra cờ sẵn sàng
        TEST AL, 1          ; thiết bị số 01 sẵn sàng?
        JZ Lap          ; chưa, quay lại thăm dò tiếp
        IN AL, 1            ; sẵn sàng, đọc cổng 01
        CALL xuly           ; xử lý dữ liệu
        LOOP Lap            ; chưa hết, quay lại thăm dò tiếp
Ra:.....
```

Trong trường hợp thiết bị vào/ra hoạt động theo cách khác : cờ sẵn sàng để báo cho CPU đọc/ghi nhiều byte dữ liệu cùng 1 lúc thì ta có nhánh chương trình đi theo đường đứt quãng. Lúc này ta phải sữa đổi đôi chút cả trong phần mạch tạo ra tín hiệu móc nối lẫn trong phần chương trình để cho toàn hệ thống làm việc đúng.

Trong thí dụ trên, nếu thay vì các mạch IC cỡ vừa (các mạch cổng 3 trạng thái và chốt 8 bit) ta dùng mạch 8255A để phối ghép CPU với thiết bị ngoại vi. Việc đọc (thăm dò) trạng thái của các tín hiệu móc nối chỉ đơn giản là đọc các bít tương ứng của cổng PC.

Trên hình 6.3 là thí dụ một ứng dụng của 8255A để phối ghép với CPU 8088 trong việc vào/ra dữ liệu theo kiểu thăm dò trạng thái sẵn sàng của thiết bị ngoại vi. Trong thí dụ này ta chưa sử dụng đến chức năng của các tín hiệu móc nối INTR này được nối vào chân INTR của 8088 thì ta có khả năng thực hiện một kiểu phối ghép khác để vào/ra dữ liệu. Đó là điều khiển vào/ra dữ liệu bằng cách ngắt CPU. Cách điều khiển vào/ra dữ liệu kiểu này sẽ được trình bày ở chương sau.

CHƯƠNG 7 : NGẮT VÀ XỬ LÝ NGẮT TRONG HỆ 8088

7.1. Sự cần thiết phải ngắt CPU

Trong cách tổ chức trao đổi dữ liệu thông qua việc thăm dò trạng thái sẵn sàng của thiết bị ngoại vi như đã được trình bày ở chương trước, trước khi tiến hành bất kỳ một cuộc trao đổi dữ liệu nào CPU phải để toàn bộ thời gian vào việc xác định trạng thái sẵn sàng làm việc của thiết bị ngoại vi. Trong hệ thống vi xử lý với cách làm việc như vậy, thông thường CPU được thiết kế chủ yếu chỉ là để phục vụ cho việc vào/ra dữ liệu và thực hiện các xử lý liên quan.

Trong thực tế người ta rất muốn tận dụng khả năng của CPU để làm thêm được nhiều công việc khác nữa, chỉ khi nào có yêu cầu trao đổi dữ liệu thì mới yêu cải CPU tạm dừng công việc hiện tại để phục vụ việc trao đổi dữ liệu. Sau khi hoàn thành việc trao đổi dữliệu thì CPU lại phải quay về để làm tiếp công việc hiện đang bị gián đoạn. Cách làm việc theo kiểu này gọi là ngắt CPU (gián đoạn hoạt động của CPU) để trao đổi dữ liệu. Một hệ thống với cách hoạt động theo kiểu này có thể đáp ứng rất nhanh với các yêu cầu trao đổi dữ liệu trong khi vẫn có thể làm được các công việc khác. Muốn đạt được điều này ta phải có cách tổ chức hệ thống ao cho có thể tận dụng được khả năng thực hiện các chương trình phục vụ ngắt tại các địa chỉ xác định của CPU. Khi nghiên cứu các tín hiệu của CPU 8088, chúng ta đã thấy vi mạch này có các chân tín hiệu cho các yêu cầu ngắt che được INTR và không che được NMI, chính các chân này sẽ được sử dụng vào việc đưa các yêu cầu ngắt từ bên ngoài đến CPU.

7.2. Ngắt trong hệ vi xử lý 8088

7.2.1. Các loại ngắt trong hệ 8088

Trong hệ vi xử lý 8088 có thể xếp các nguyên nhân gây ra ngắt CPU vào 3 nhóm như sau:

+ Nhóm các ngắt cứng : đó là các yêu cầu ngắt CPU do các tín hiệu đến từ các chân INTR và NMI.

Ngắt cứng INTR là yêu cầu ngắt che được. Các lệnh CLI và STI có ảnh hưởng trực tiếp tới trạng thái của cờ IF trong bộ vi xử lý, tức là ảnh hưởng tới việc CPU có nhận biết yêu cầu ngắt tại chân này hay không. Yêu cầu ngắt tại chân INTR có thể có kiểu ngắt N nằm trong khoảng 0-FFH. Kiểu ngắt này phải được đưa vào bus dữ liệu để CPU có thể đọc được khi có xung $\overline{\text{INTA}}$ trong chu kỳ trả lời chấp nhận ngắt.

Biểu đồ thời gian của các xung liên quan đến quá trình trên được mô tả trên hình 7.1.

+ Nhóm các ngắt mềm: khi CPU thực hiện các lệnh ngắt dạng INT N, trong đó N là số hiệu (kiểu) ngắt nằm trong khoảng 00-FFH (0-255).

+ Nhóm các hiện tượng ngoại lệ: đó là các ngắt do các lỗi nảy sinh trong quá trình hoạt động của CPU như phép chia cho 0, xảy ra tràn khi tính toán.

Yêu cầu ngắt sẽ được CPU kiểm tra thường xuyên tại chu kỳ đồng hồ cuối cùng của mỗi lệnh.

Trên hình 7.2 trình bày một cách đơn giản để đưa được số hiệu ngắt N vào bus dữ liệu trong khi cũng tạo ra yêu cầu ngắt đưa vào chân INTR của bộ vi xử lý 8088.

Giả thiết trong một thời điểm nhất định chỉ có một yêu cầu ngắt IRi được tác động và khi đó ở đầu ra của mạch NAND sẽ có xung yêu cầu ngắt đến CPU. Tín hiệu IRi được đồng thời đưa qua mạch khuếch đại đệm để tạo ra số hiệu ngắt tương ứng, số hiệu ngắt này sẽ được CPU đọc vào khi nó đưa ra tín hiệu trả lời $\overline{\text{INTA}}$.

Bảng 7.1. Cho ta quan hệ giữa IRi và số hiệu ngắt N tương ứng.

AD7	IR6	IR5	IR4	IR3	IR2	IR1	IR0	N
1	1	1	1	1	1	1	0	FEH (254)
1	1	1	1	1	1	0	1	FDH (253)
1	1	1	1	1	0	1	1	FBH (251)
1	1	1	1	0	1	1	1	F7H (247)
1	1	1	0	1	1	1	1	EFH (239)
1	1	0	1	1	1	1	1	DFH (223)
1	0	1	1	1	1	1	1	BFH (191)

Ta sẽ còn đề cập đến việc xử lý trường hợp có 2 yêu cầu ngắt IRi cùng một lúc và cách đưa các giá trị N của INTN vào bus dữ liệu một cách tệ động bằng mạch điều khiển ngắt PIC ở phần sau.

7.2.2. Đáp ứng của CPU khi có yêu cầu ngắt

Khi có yêu cầu ngắt kiểu N đến chân CPU và nếu yêu cầu đó được phép, CPU thực hiện các công việc sau:

1. SP ← SP-2, {SP} ← FR, trong đó {SP} là ô nhớ do SP chỉ ra.
   (chỉ ra đỉnh mới của ngắn xếp, cất thanh ghi cờ vào đỉnh ngăn xếp)
2. IF ← 0, TF ← 0.
   (cấm các ngắt khác tác động vào CPU, cho CPU chạy ở chế độ bình thường)
3. SP ← SP-2, {SP} ← CS.
   (chỉ ra đỉnh mới của ngăn xếp, cất phần địa chỉ đoạn của địa chỉ trở về vào đỉnh ngăn xếp)
4. SP ← SP-2, {SP} ← IP
   (chỉ ra đỉnh mới của ngăn xếp, cất phần địa chỉ lệch của địa chỉ trờ về vào đỉnh ngăn xếp)
5. {N*4} → IP, {N*4+2} → CS
   (lấy lệnh tại địa chỉ mới của chương trình con phục vụ ngắt kiểu N tương ứng trong bảng vectơ ngắt)
6. Tại cuối chương trình phục vụ ngắt, khi gặp lệnh IRET
   {SP} → IP, SP ← SP+2
   {SP} → CS, SP ← SP+2
   {SP} → FR, SP ← SP+2
   (bộ vi xử lý quay lại chương trình chính tại địa chỉ trở về và với giá trị cũ của thanh ghi cờ được lấy ra từ ngăn xếp).

Về mặt cấu trúc chương trình, khi có ngắt xảy ra thì chương trình chính (CTC) liên hệ với chương trình con phục vụ ngắt (CTCPVN) như mô tả trên hình 7.3.

Trong thực tế các ngắt mềm INT N đã bao trùm các loại khác bởi vì Intel đã quy định một số kiểu ngắt đặc biệt được xếp vào đầu dãy ngắt mềm INY N như sau:

+ INT 0 : Ngắt mềm do phép chia cho số 0 gây ra,

+ IN T1 : Ngắt mềm để chạy từng lệnh ứng với trường hợp cờ TF=1,

+ IN T2 : Ngắt cứng do tĩn hiệu tích cực tại chân NM1 gây ra,

+ IN T3 : Ngắt mềm để đặt điểm dừng của chương trình tại một địc chỉ nào đó

+ IN T4 : (Hoặc lệnh INTO) : ngắt mềm ứng với trường hợp cờ tràn OF=1.

Các kiểu ngắt khác còn lại thì được dành cho Intel và cho người sử dụng (IBM không hoàn toàn tuân thủ các quy định này khi chế tạo các máy PC/XT và PC/AT0:

+ INT 5-INT 1FH; dành riêng cho Intel trong các bộ vi xử lý cao cấp khác,

+ INT 20H-INT FFH: dành cho người sử dụng.

Các kiểu ngắt N trong INT N đều tương ứng với các địa chỉ xác định của CTCPVN mà ta có thể tra được trong bảng các vectơ ngắt. Intel quy định bảng này nằm trong RAM bắt đầu từ đạic hỉ 00000H và dài 1 KB (vì 8088 có tất cả 256 kiểu ngắt, mỗi kiểu ngắt ứng với 1 vectơ ngắt, 1 vectơ ngắt cần 4 byte để chứa địa chỉ đầy đủ cho CS:IP của CTCPVN).

Bảng 7.2. Bảng vectơ ngắt của 8088 tại 1KB RAM đầu tiên

03FEH-03FFH	CS của CTPVN INT FFH
03FCH-03FDH	IP của CTPVN INT FFH
0082H-0083H	CS của CTPVN INT 20H
0080H-0081H	IP của CTPVN INT 20H
000AH-000BH	CS của CTCPVN INT 2
0008H-0009H	IP của CTCPVN INT 2
0006H-0007H	CS của CTCPVN INT 1
0004H-0005H	IP của CTCPVN INT 1
0002H-0003H	CS của CTCPVN INT 0
0000H-0001H	IP của CTCPVN INT 0

Trên bảng 7.2. giới thiệu một phần của bảng vectơ ngắt của CPU 8088

7.2.3. Xử lý ưu tiên khi ngắt:

Có một vấn đề rất thực tế đặt ra là nếu tại cùng một thời điểm có nhiều yêu cầu ngắt thuộc các loại ngắt khác nhau cùng đòi hỏi CPU phục vụ thì CPU xử lý các yêu cầu ngắt đó như thế nào? Câu trả lời là CPU xử lý các yêu cầu ngắt theo thứ tự ưu tiên với nguyên tắc ngắt nào có mức ưu tiên và phục vụ trước.

Ngay từ khi được chế tạo (thường gọi là ngầm định) CPU 8088 có khả năng phân biệt các mức ưu tiên khác nhau cho các loại ngắt (theo thứ tự từ cao xuống thấp) như sau:

+ ngắt nội bộ : INT 0 (phép chia cho 0), INT N, INTO ... cao nhất

+ ngắt không che được NMI

+ ngắt che được INTR

+ ngắt để chạy từng lệng INT 1 ... thấp nhất

Để thấy rõ hoạt động của CPU trong cơ chế ngắt ưu tiên này ta có thể lấy một ví dụ cụ thể như sau.

Giả thiết tại một thời điểm nào đó, trong khi CPU (ở trạng thái cho phép ngắt với cừo IF=1) đang thực hiện phép chia và có lỗi xảy ra do số bị chia bằng 0, đúng vào lúc đó CPU cũng nhận được yêu cầu từ đầu vào INTR. CPU sẽ xử lý ra sao trong trường hợp này?

Theo thứ tự ưu tiên ngầm định trong việc xử lý ngắt của CPU 8088 thì INT 0 có mức ưu tiên cao hơn INTR, vì vậy đầu tiên CPU sẽ thực hiện chương trình phục vụ ngắt INT 0 để đáp ứng với lỗi đặc biệt cho phép chua cho 0 gây ra và cờ IF bị xóa về 0. Yêu cầu ngắt INTR sẽ tự động bị cấm cho tới khi chương trình phục vụ ngắt INT 0 được hoàn tất và trở về nhờ IRET, cờ IF cũ được trả lại. Tiếp theo đó CPU sẽ đáp ứng yêu cầu ngắt INTR bằng cách thực hiện chương trình phục vụ ngắt dành cho INTR.

7.2.4. Mạch điều khiển ngắt ưu tiên 8259A

Trong trường hợp có nhiều yêu cầu ngắt che được từ bên ngoài phải phục vụ tư thường dùng vi mạch có sẵn 8259A để giải quyết vấn đề ưu tiên. mạch 8259A được gọi là mạch điều khiển ngắt ưu tiên (priority interrupt controller, PIC). Đó là một vi mạch cỡ lớn lập trình được, có thể xử lý trước được 8 yêu cầu ngắt với 8 mức ưu tiên khác nhau để tạo ra một yêu cầu ngắt đưa đến đầu vào INTR (yêu cầu ngắt che được của CPU 8088. Nếu nối tầng 1 mạch 8259A chủ với 8 mạch 8259A thợ ta có thể nâng tổng số các yêu cầu ngắt với các mức ưu tiên khác nhau lên thành 64.

Các khối chức năng chính của 8259A

+ Thanh ghi IRR: ghi nhớ các yêu cầu ngắt có tại đầu vào IRi.

+ Thanh ghi ISR: ghi nhớ các yêu cầu ngắt đang được phục vụ trong số các yêu cầu ngắt IRi.

+ Thanh ghi IMR: ghi nhớ mặt nạ ngắt đối với các yêu cầu ngắt IRi.

+ Logic điều khiển: khối này có nhiệm vụ gửi yêu cầu ngắt tới INTR của 8088 khi có tín hiệu tại các chân IRi và nhận trả lời chấp nhận yêu cầu ngắt INTA từ CPU để rồi điều khiển việc đưa ra kiểu ngắt trên bus dữ liệu.

+ Đệm bus dữ liệu: dùng để phối ghép 8259A với bus dữ liệu của CPU

+ Logic điều khiển ghi/đọc: dùng cho việc ghi các từ điều khiển và đọc các từ trạng thái của 8259A.

+ Khối đệm nối tầng và so sánh: ghi nhớ và so sánh số hiệu của các mạch 8259A có mặt trong hệ vi xử lý.

Các tín hiệu của 8259A:

Một số tín hiệu trong mạch 8259 có tên giống như các tín hiệu tiêu chuẩn của hệ vi xử lý 8080. Ta có thể thấy rõ và hiểu được ý nghĩa của chúng ngay trên hình 7.4. Ngoài các tín hiệu này ra, còn có một số tín hiệu đặc biệt khác của 8259A cần phải giới thiệu thêm gồm:

+ Cas_0-Cas_2 [I,O]: là các đầu vào đối với các mạch 8259A thợ hoặc các đầu ra của mạch 8259A chủ dùng khi cần nối tầng để tăng thêm các yêu cầu ngắt cần xử lý.

+ $\overline{SP}/\overline{EN}$ [I,O]: khi 8259A làm việc ở chế độ không có đệm bus dữ liệu thì đây là tín hiệu vào dùng lập trình để biến mạch 8259A thành mạch thợ ($\overline{SP}= 0$) hoặc chủ ($\overline{SP}= 1$); khi 8259A làm việc trong hệ vi xử lý ở chế độ có đệm bus dữ liệu thì chân này là tín hiệu ra $\overline{EN}$ dùng mở đệm bus dữ liệu để 8088 và 8259A thông vào bus dữ liệu hệ thống. Lúc này việc định nghĩa mạch 8259A là chủ hoặc thợ phải thực hiện thông qua từ điều khiển đầu ICW4.

+ INT [O]: tín hiệu yêu cầu ngắt đến chân INTR của CPU 8088.

+ $\overline{INTA}$ [I]: nối với tín hiệu báo chấp nhận ngắt $\overline{INTA}$ của CPU.

PIC 8259A chủ (ở chế độ không đêm nối với CPU 8088 ở chế độ MIN

Trên hình 7.5 là sơ đồ nối mạch PIC 8259A làm việc độc lập (mạch chủ) với bus CPU 8088 làm việc ở chế độ MIN.

PIC 8259A chủ nối với PCU 8088 ở chế độ MAX.

Nếu hệ vi xử lý 8088 làm việc ở chế độ MAX thường ta phải dùng mạch điều khiển bus 8288 và các đệm bus để cung cấp các tín hiệu thích hợp cho bus hệ thống. Mạch 8259A phải làm việc ở chế độ có đệm để nối được với bus hệ thống này.

Trên hình 7.6 là ví dụ một sơ đồ CPU 8088 chế độ MAX nối với PIC 8259A. Trong mạch này ta nhận thấy tín hiệu địa chỉ cho 8259A được lấy ra từ bus hệ thống trong khi đó tín hiệu dữ liệu của nó được nối với bus dữ liệu của vi xử lý vatf đó được thông qua các đệm để nối vào bus hệ thống.

Hình 7.6. Nối CPU 8088 chế độ MAX với PIC8259A

Lập trình cho PIC 8259A

Để mạch PIC 8259A có thể hoạt động được theo yêu cầu, sau khi bật nguồn cấp điện PIC cần phải được lập trình bằng cách ghi vào các thanh ghi (tương đương với các cổng) bên trong nó *các từ điều khiển khởi đầu (ICW) và tiếp sau đó là các từ điều khiển hoạt động (OCW).*

Các từ điều khiển khởi đầu dùng để tạo nên các kiểu làm việc cơ bản cho PIC, còn các từ điều khiển hoạt động sẽ quyết định cách thức làm việc cụ thể của PIC. Từ điều khiển hoạt động sẽ được ghi khi ta muốn thay đổi hoạt động của PIC.

Các từ điều khiển nói trên sẽ được giới thiệu cụ thể trogn các mục sau.

- **Các từ điều khiển khởi đầu ICW**

PIC 8259A có tất cả 4 từ điều khiển khởi đầu là ICW1 - ICW4. Trong khi lập trình cho PIC không phải lúc nào ta cũng cần dùng cả 4 từ điều khiển khởi đầu những có lúc ta chỉ cần ghi vào đó 2 hay 3 từ là đủ (xem hình 7.7 để thấy rõ thứ tự ghi và điều kiện để ghi các điều khiển ICW vào 8259A).

Dạng thức của các thanh ghi điều khiển khởi đầu ICW được biểu diễn trên hình 7.8.

Trên hình này ta thấy bên cạnh các bit dữ liệu của từ điều khiển khởi đầu ICW ta còn ghi rõ thêm cả giá trị cụ thể của A0 tương ứng cho mỗi ICW đó. Đầu vào địa chỉ A0 và thứ tự ghi sẽ giúp ta phân biệt ra các thanh ghi khác nhau bên trong PIC để ghi dữ liệu cho các từ điều khiển. Ví dụ A0 = 0 là dấu hiệu để nhận biết rằng ICW1 được đưa vào thanh ghi có địa chỉ chẵn trong PIC, còn khi A0 = 1 thì các từ điều khiển khởi đầu ICW2, ICW3, ICW4 sẽ được đưa vào các thanh ghi có địa chỉ lẻ trong mạch PIC.

- **ICW1**

Bit D0 của ICW1 quyết định 8259A sẽ được nối với họ vi xử lý nào. Để làm việc với hệ 16-32bit (8088 hoặc họ 80×86) thì ICW nhất thiết phải có IC4 = 0 (và như vậy các bit của ICW4 sẽ bị xóa về 0). Các bit còn lại của ICW1 định nghĩa cách thức tác động của xung yêu cầu ngắt (tác động theo sườn hay theo mức) tại các chân yêu cầu ngắt IR của mạch 8259A và việc bố trí các mạch 8259A khác trong hệ làm việc đơn lẻ hay theo chế độ nối tầng.

Các bit được đánh dấu x là không quan trọng và thường được lấy giá trị 0 để lập trình cho các ứng dụng sau này.

- **ICW2**

Từ điều khiển khởi đầu này cho phép chọn kiểu ngắt (số hiệu ngắt) ứng với các bit T3-T7 cho các đầu vào yêu cầu ngắt. Các bit T0-T2 được 8259A tự động gán giá trị tùy theo đầu vào yêu cầu ngắt cụ thể IRi. Ví dụ nếu ta muốn các đầu vào của mạch 8259A có kiểu ngắt là 40-47H ta chỉ cần ghi 40H vào các bit T3-T7. Nếu làm như vậy thì IR0 sẽ có kiểu ngắt là 40H, IR1 sẽ có kiểu ngắt là 41H...

- **ICW3**

Từ điều khiển khởi đầu này chỉ dùng đến khi bit SNGL thuộc từ điều khiển khởi đầu ICW1 có giá trị 0, nghĩa là trong hệ có các mạch 8259A làm việc ở chế độ nối tầng. Chính vì vậy tồn tại 2 loại ICW3: 1 cho mạch 8259A chủ và 1 cho mạch 8259A thợ.

ICW3 cho mạch chủ : dùng để chỉ ra đầu vào yêu cầu ngắt Iri nào của nó có tín hiệu INT của mạch thợ nối vào.

ICW3 cho mạch thợ : dùng làm phương tiện để các mạch này được nhận biết. Vì vậy từ điều khiển khởi đâùu này phải chứa mã số i ứng với đầu vào Iri của mạch chủ mà mạch thợ đã cho nối vào. Mạch thợ sẽ so sánh mã số này với mã số nhận được ở Cas2-Cas0. Nếu bằng nhau thì số hiệu ngắt sẽ được đưa ra bú khi có INTA.

Ví dụ : Trong một hệ vi xử lý ta có một mạch 8259A chủ và 2 mạch 8259A thợ nối vào chân IR1 của mạch chủ.

Tìm giá trị phải gán cho các từ điều khiển khởi đầu ICW ?

Giải : Như trên đã nói, để các mạch này làm việc được vưới nhau ta sẽ phải gia các từ điều khiển khởi đầu như sau: ICW3 = 03H cho mạch chủ. ICW3 = 00H cho mạch thựo thứ nhất và ICW3 = 01H cho mạch thợ thứ hai.

ICW4

Từ điều khiển khởi đầu này chỉ dùng đến khi trong ừt điều khểin ICW1 có IC4 = 1 (cần thêm ICW4)

Bit $_{M}$PM cho ta khả năng chọn loại vi xử lý để làm việc với 8259A. Bit $_{M}$PM = 1 cho phép các bộ vi xử lý từ 8086/88 hoặc cao hươn làm việc vưới 8259A.

Bit SFNM = 1 cho phép chọn chế độ ưu tiên cố định đặc biệt. Trong chế độ này yêu cầu ngắt với mức ưu tiên cao nhất hiện thời từ một mạch thợ làm việc theo kiểu nối tầng sẽ được mạch chủ nhận biết ngay cả khi mạch chủ còn đang phải phục vụ một yêu cầu ngắt ở mạch thợ khác nhưng với mức ưu tiên thấp hơn. Sau khi các yêu cầu ngắt được phục vụ xong thì chương trình phục vụ ngắt phải có lệnh kết thúc yêu cầu ngắt (EOI) đặt trước lệnh trở về (IRET) đưa đến cho mạch 8259A chủ.

Khi bit SFNM = 0 thì chế độ ưu tiên cố định được chọn (IR0: mức ưu tiên cao nhất. IR7: mức ưu tiên thấp nhất) thực ra đối vưới mạch 8259A không dùng đến ICW1 thì chế độ này đã được chọn như là ngầm định. Trong chế độ ưu tiên cố định tại một thời điểm chỉ có một yêu cầu ngắt i được phục vụ (bit Iri = 1) lúc này ất cả các yêu cầu khác vưới ứmc ưu tiên cao hươn có thể ngắt yêu cầu khác với mức ưu tiên thấp hơn .

Bit BUF cho phép định nghĩa mạch 8259A để làm việc với CPU trong trường hợp có đệm hoặc không có đệm nối với bú hệ thống. Khi làm việc ở chế độ có đệm (BUF = 1). Bit M/S = 1/0 cho phép ta chọn mạch 8259A để làm việc ở chế độ chủ/ thợ. SP/EN trở thành đầu ra cho phép mở đệm để PIC và CPU thông vưới bú hệ thống.

Bit AEOI = 1 cho phép chọn cách kết thcú yêu cầu ngắt tự động. Khi AEOI = 1 thì 8259A tự động xóa ISRi = 0 khi xung INTA cuối cùng chuyển lên mức cao mà không làm thay đổi thứ tự ưu tiên. Ngược lại. Khi ta chọn cách kết thúc yêu cầu ngắt thường (AEOI = 0) thì chương trình phục vụ ngắt phải có thêm lệnh EOI đặt trước lệnh IRET để kết thúc cho 8259A.

Các từ điều khiển hoạt động OCW

Các từ đêìu khiển hoạt động OCW sẽ quyết định mạch 8259A sẽ hoạt động như thế nào sau khi nó đã được khởi đầu bằng các từ điều khiển ICW . Tất cả các từ điều khiển hoạt động sẽ đwowcj ghi vào các thanh ghi trong PIC khi A0 = 0,trừ OCW1 được ghi khi A0 = 1.

Dạng thức của OCW đwocj trinh bày trên hình 7.9

OCW1

OCW1 dùng để ghi giá trị của các bit mặt nạ vào thanh ghi mặt nạ ngắt IMR. Khi một bit mặt nạ nào đó của được lập thì yêu cầu ngắt tương ứng với mặt nạ đó sẽ không được

8259A nhận biết nữa (bị che). Từ điều khiển này phải được đưa đến 8259A ngay sau khi ghi các ICW vào 8259A .

Ta cũng có thể đọc lại IMR để xác định tình trạng mặt nạ ngắt hiện tại (xem trong thời điểm hiện tại yêu cầu ngắt nào bị che)

OCW2

Các bit R.SL và EOI phối hợp vuới nhau cho phép chọn ra các cách thức kết thúc yêu cầu ngắt khác nhau. Một vài cách thcú yêu cầu ngắt còn tác động tới các yêu cầu ngắt được chỉ đích danh với mức ưu tiên được giải mã hóa của 3 bit L_2,L_1,L_0

Trước khi nói về các cách kết thúc yêu cầu ngắt cho các chế độ ta cần mở dấu ngoặc ở đây để giới thiệu các chế độ làm việc của 8259A.

a. Chế độ ưu tiên cố định :

Đây là chế độ làm việc ngầm định của 8259A sau khi nó đã được nạp các từ điều khiển khởi đầu.

Trong chế độ này, các đầu vào IR7-IRO được gán cho các mức ưu tiên cố định: IRO được gán cho mức ưu tiên cao nhất còn IR7 mức ưu tiên thấp nhất. Mức ưu tiên này được giữ không thay đổi cho đến khi ghi mạch 8259A bị lập trình khác đi do OCW2.

Trong chế độ ưu tiên cố định tai 1 thời điểm chỉ có 1 yêu cầu ngắt i được phục vụ (bit ISRi = 1) lúc này tất cả ác yêu cầu khác với mức ưu tiên thấp hơn đều bị cấm, tất cả các yêu cầu khác vuới mức ưu tiên thấp hơn đều có thể ngắt yêu cầu khác với mức ưu tiên thấp hơn.

OCW1

A_0	D7	D6	D5	D4	D3	D2	D1	D0
		I	M7	M6	M5	M4	M3	M2	M1	M0

Mặt nạ ngắt tại các yêu cầu ngắt

1: Có mặt nạ

0: Không có mặt nạ

OCW2

A_0	D7	D6	D5	D4	D3	D2	D1	D0
		0	R	SL	EOI	0	0	L2	L1	L0

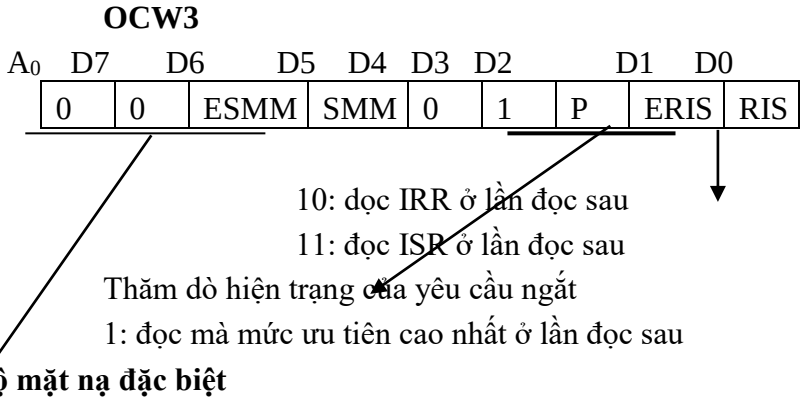
Kết thúc Ngắt (EOI Đổi mức ưu tiên tự động Đổi m.ư.t	**0**	**0**	**1**	Lệnh EOI thường
	0	**1**	**1**	Lệnh EOI chỉ đích danh
	1	**0**	**1**	Đổi mức ưu tiên khi có EOI thường.
	1	**0**	**0**	Lập chế độ quay khi có EOI tự động
	0	**0**	**0**	
	1	**1**	**1**	Xóa chế độ quay khi có EOI tự động
	1	**1**	**0**	Đổi mức ưu tiên khi có EOI đích danh (*)
	0	**1**	**0**	Lệnh lập tức ưu tiên (*)

Mã hóa mức ưu tiên bị tác

000….mức 0

111

(*) dùng tổ hợp L_2 L_1 L_0



Hình 7.9. Dạng thức của các từ điều khiển hoạt động OCW

b.chế độ quay mức ưu tiên (ưu tiên luân phiên) tự động :

ở chế độ này sau khi một yêu cầu ngắt được phục vụ xong ,8259A sẽ xoá bit tương ứng của nó trong thanh ghi ISR và gán cho đầu vào của nó mức ưu tiên thấp nhất để tạo điều kiện cho các yêu cầu ngắt khác có thời cơ được phục vụ .

c. chế độ quay (đổi) mức ưu tiên chỉ đích danh :

ở chế độ này ta cần chỉ rõ (đích danh) đầu vào IRi nào ,với i=$L_2L_1L_0$,được gán mức ưu tiên thấp nhất ,đầu vào IR_{i+1} sẽ đuợc tự động gán mức ưu tiên cao nhất .

Bây giờ ta đóng dấu ngoặc đã được ra mở ở trên và trở lại các vấn đề liên quan đến OC WC.ta sẽ nói rõ việc các bit R,Slvà SOI phối hợp với nhau như thế nào để tạo ra các lệnh quy định các cách thức kết thúc yêu càu ngắt cho các chế độ làm việc khác nhau đã nói đến ở phần trên (xem thêm hinh 7.9) .

1. kết thúc yêu cầu ngắt thường : chương trình còn phục vụ ngắt ophải có lệnh EOI đặt trước lệnh trở về IRET cho 8259A .mạch 8259 A sẽ xác định yêu cầu ngắt Iri vừa được phục vụ và xoá bit ISRi tương ứng của nó để tạo điều kiện cho chính yêu cầu ngắt này hoặc các ngắt khác có mức ưu tiên thấp hơn có thể được tác động .

2.kết thúc yêu cầu ngắt thường : chương trình còn phục vụ ngắt phải co lệnh EOI chỉ đích danh đặt trước lệnh trở về IRET cho 8259A . mạch 8259A xoá đích danh bit ISRi ,với i=$L_2L_1L_0$ để tạo điều kiện cho chính yêu cầu ngắt này hoặc các ngắt khác có mức ưu tiên thấp hơn có thể được tác động.

3. Quay (đổi) mức ưu tiên khi kết thúc yêu cầu ngắt thường: chương trình con phục vụ ngắt phải có lệnh EOI đặt trước lệnh trở về IRET cho 8259A. Mạch 8259A sẽ xác định yêu cầu ngắt thứ i vừa được phục vụ. Xóa bit ISRi tương ứng và gán luôn mức ưu tiên thấp nhất cho đầu vào IR, này còn đầu vào IR_{i+1} sẽ đwocj gán mức ưu tiên cao nhất.

Thanh ghi ISR trước khi IR4 được chấp nhận

(0: mức ưu tiên cao nhất, : Mức ưu tiên thấp nhất)

	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
Trạng thái của ISR	0	1	0	1	0	0	0	0
Mức ưu tiên	7	6	5	4	3	2	1	0

Thanh ghi ISR sau khi IR4 được chấp nhận và sau khi có lệnh quay:

(o: mức ưu tiêncao nhất,7: mức ưu tiên thấp nhất)

	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
Trạng thái của ISR	0	1	0	0	0	0	0	0
Mức ưu tiên	2	1	0	7	6	5	4	3

Hình 7.10.ISR và mức ưu tiên với lệnh quay khi kết thúc ngắt thường .

Có thể theo dõi cách tứhc hoạt động của mạch 8259A trong chế độ quay (đổi) mức ưu tiên khi kết thúc yêu cầu ngắt thường thông qua ví dụ minh họa trình bày trên hình 7.10

4. Quay (đổi) mức ưu tiên trong chế độ kết thúc yêu cầu ngắt tự động: chỉ cần một lần đưa lệnh chọn chế độ đổi mức ưu tiên khi kết thúc yêu cầu ngắt tự động. Có thể chọn chế độ này bằng lệnh lập "chế độ quay khi có EOI tự động" . Từ đó trở đi 8259A sẽ đổi mức ưu tiên mỗi khi kết thúc ngắt tự động theo cách tương tự như ở mục 3. Muốn bỏ chế độ này ta có thể dùng lệnh xóa "chế độ quay khi có EOI tự động".

5. Quay (đổi) mức ưu tiên khi kết thúc yêu cầu ngắt đích danh: chương trình còn phục vụ ngắt phải có lệnh EOI đích danh cho 8259A đặt trước lệnh trở về IRET . Mạch 8259A sẽ xóa bit ISRi của yêu cầu ngắt tương ứng và gán luôn mức ưu tiên thấp nhất cho đầu vào Iri, với i = L_2 L_1 L_0. Yêu cầu ngắt IR_{i+1} sẽ được gán mức ưu tiên cao nhất.

OCW3

Từ điều khiển hoạt động sau khi được ghi vào 8259A cho phép:

+ Chọn các ra thanh ghi để đọc

+ Thăm dò trạng thái yêu cầu ngắt bằng cách trạng thái của đầu vào yêu cầu ngắt Iri với mức ưu tiên cao nấht cùng mã của đầu vào đó và.

+ Thao tác với mặt nạ đặc biệt .

D7	D6	D5	D4	D3	D2	D1	D0
IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0

a) IRR Isi = 0: yêu cầu ngắt i không được phục vụ

ISi = 1: đầu vào i có yêu cầu ngắt

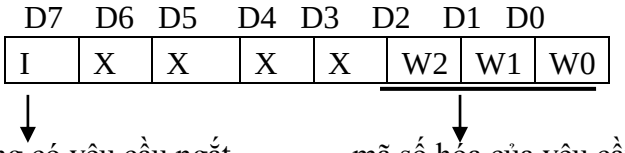
D7	D6	D5	D4	D3	D2	D1	D0
IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0

b) ISR Isi = 0: yêu cầu ngắt i không được phục vụ

Isi = 1: yêu cầu ngắt i đang được phục vụ

Các thanh ghi IRR và ISR có thể đọc được sau khi nạp vào 8259A từ điều khiển OCW3 với bit ERIS = 1: bit RIS = 0 sẽ cho phép đọc IRR. Bit RIS = 1 sẽ cho phép đọc ISR. Dạng thức của các thanh ghi này đwocj biểu diễn trên hình 7.11a và 6.11b

Bằng việc đưa vào 8259A từ điều khiển OCW3 với bit P = 1 ta có thể đọc được trên bus dữ liệu ở lần đọc tiếp ngay sau đó từ thăm dò, trong đó có các thông tin về yêu cầu ngắt với mức ưu tiên cao nhất đang hoạt động và mã tương ứng với yêu cầu ngắt ấy theo dạng thức được biểu diễn trên hình 7.12.



1: đang có yêu cầu ngắt mã số hóa của yêu cầu ngắt

0: KHông có yêu cầu ngắt

(X: không để ý)

Hình 7.12 Dạng thức của từ thăm dò trạng thái yêu cầu ngắt

Có thể gọi đây là chế độ thăm dò yêu cầu ngắt và chế độ này thường đwocj ứng dụng trong trường hợp có nhiều chương phục vụ ngắt giống nhau cho một yêu cầu ngắt và việc chọn chương trình nào để sử dụng là trách nhiệm của người lập trình.

Tóm lại, muốn dùng chế độ thăm dò của 8259A để xác định yêu cầu ngắthiện tại ta cần làm các thao tác lần lượt như sau:

- Cấm các yêu cầu ngắt bằng lệnh CLI
- Ghi từ lệnh OCW3 vưới bit P = 1
- Đọc từ thăm dò trạng thái yêu cầu ngắt trên bus dữ liệu .

Bit ESMM = 1 cho phép 8259A thao tác với chế độ mặt nạ đặc biệt. Bit SMM = 1 cho phép lập chế độ mặt nạ đặc biệt. Chế độ mặt nạ đặc biệt được dùng để thay đổi thứ tự ưu tiên ngay bên trong chương trình con phục vụ ngắt. Ví dụ trong trường hợp có một yếu cầu ngắt cấm (bị che bởi chương trình phục vụ ngắt với từ lệnh OCW1 mà ta lại muốn cho phép các yêu cầu ngắt với mức ưu tiên thấp hơn so với yêu cầu ngắt bị cấm đó được tác động thì ta sẽ dùng chế độ mặt nạ đặc biệt. Một khi đã được lập, chế độ mặt nạ đặc biệt sẽ tồn tại cho tới khi bị xóa bằng cách ghi vào 8259A một từ lệnh OCW3 khác vưới bit SMM = 0. Mặt nạ đặc biệt không ảnh hưởng tới các yêu cầu ngắt vưới mức ưu tiên cao hơn.)

Cuối cùng để có cái nhìn một cách có hệ thống về hoạt động của hệ vi xử lý với CPU 8088 và PIC 8259A khi cso yêu cầu ngắt, ta tóm lượt hoạt động của chúng như sau:

1. Khi có yêu cầu ngắt từ thiết bị ngoại vi tác động vào một trong các chân IR của PIC .8259A sẽ đưa INT = 1 đến chân INTR của 8088.

2. 8088 đưa ra xung $\overline{\text{INTA}}$ đầu đến 8259A

3. 8259A dùng xung $\overline{\text{INTA}}$ đầu như là thông báo để nó hoàn tất các xử lý nội bộ cần thiết, kể cả xử lý ưu tiên nếu như có nhiều yêu cầu ngắt cùng xảy ra .

4. 8088 đưa ra xung $\overline{\text{INTA}}$ thứ hai đến 8259A

5. Xung INTA thứ hai khiến 8259A đưa ra bus dữ liệu 1 byte chứa thông tin về số hiệu ngắt của yêu cầu ngắt vừa được nhận biết.

6. 8088 dùng số hiệu ngắt để tính ra địa chỉ ngắt của vectơ ngắt tương ứng .

7. 8088 cất FR, xóa các cờ Ì và TF và cất địa chỉ trở về CS:IP vào ngăn xếp.

8. 8088 lấy địa chỉ CS:IP của chương trình phục vụ ngắt từ bảng vectơ ngắt và thực hiện chương trình đó .

Một số ví dụ lập trình với 8259A

Ví dụ 1:

Lập trình cho 8259A để làm việc vưới CPU 8088 ở chế độ cũ (đơn lẻ), trong hệ có đệm Bus, chế độ ưu tiên cố định và với EOI thường, IR kích theo mức, tín hiệu IR_0 được gán số hiệu ngắt là 50H.

Giải

Từ hình 7.8 và căn cứ theo các yêu cầu của bài toán ta có thể từ điều khiển khởi đầu như sau :

ICW1 = 00011011 = 1BH

D0 = 1 Cần thêm ICW4

D1 = 1 Làm việc đơn lẻ, không cần ICW3

D2 = 0 Làm việc với hệ 8086/88

D3 = 1 Đầu vào IR ăn theo mức

D4 = 1 Bắt buộc với ICW1

D5 = D6 = D7 =0 gán bằng 0 cho hệ 8086/88

ICW2 = 01010000 = 50H

Vì các bit T_1 - T_3 của ICW2 phải mã hóa trị số 50H để IR_0 - IR_7 được mã hóa tiếp bởi các Bit T_2 - T_0 = 000

ICW3 không cần đếm

ICW4 = 00001101 = 0DH

D_0 = 1 làm việc với hệ số 8086/88

D_1 = 0 EOI thường (phải có EOI trước IRET)

D_3 D_2 = 11 Làm việc ở chế đọ chủ trong hệ có đệm Bus

D_4 = 0 Chế độ ưu tiên cố định

D_5 = D_6 = D_7 = 0 Luôn bằng 0 cho ICW4

Ví dụ 2 :

Viết chương trình để khởi đầu cho mạch 8259A ở ví dụ trên nếu nó được nối vưới 8088 theo sơ đồ trên hình 7.6 với địa chỉ F_0H và F_1H

Giải

Đoạn chương trình để khởi đầu cho mạch này có thể là:

```
MOV     AL,1BH      ;ICW1
OUT     OFOH, AL    ;ICW1 đưa ra cổng FOH
MOV     Al,50H;ICW2
OUT     OF1H,AL     ;ICW2 đưa ra cổng F1H
MOV     AL, ODH     ;ICW4
OUT     OF1H, AL    ;ICW4 đưa ra cổng F1H
```

Ví dụ 3:

Giả thiết tại chân IR3 của mạch 8259A ở trên có tín hiệu yêu cầu ngắt .Tại chương trình con phục vụ cho yêu cầu ngắt này có cần phải có lệnh kết thúc ngắt (EOI) không ? nếu có thì hãy viết ra các lệnh đó.

Giải:

Giả thiết tại IR3 có tín hiệu yêu cầu ngắt và CPU đã trả lời chấp nhận ngắt. 8259A sẽ lập Bit ISR3 = 1 để ghi nhớ là yêu cầu ngắt IR3 đang được phục vụ. Vì trước đó ta đã định nghĩa chế độ kết thúc ngắt bình thường cho 8259A, nên trong chương trình con phục vụ ngắt của yêu cầu ngắt này ta phải có lệnh kết thúc ngắt EOI đưa đến 8259A để xóa Bit ISR3 đupợc lập trước đó, tạo điều kiện cho các yêu cầu ngắt khác với mức ưu tiên thấp hơn hoặc cho chính yêu cầu ngắt mới tại chân IR3 có thời cơ được phục vụ.

Dạng thường thấy của 1 chương trình con phục vụ ngắt kiểu này có thể như sau (ta chỉ chú ý nhấn mạnh đến các lệnh cuối liên quan đến EOI):

```
Phuc vu IR3  Proc
; thân của chương trình con phục vụ ngắt .
MOV  AL20        ;OCW2 vưới EOI để kết thúc
OUT  OFOH, AL    ; đưa OCW2 đến 8259A .
IRET             ;  trở về chương trình chính phuc vu IR3Endp
```

Ví dụ 4:

Giả thiết vẫn sử dụng sơ đồ trên hình 7.6 và các số liệu như ở các ví dụ trước. Hệ này bình thường làm một công việc gì đó, nhưng khi có yêu cầu ngắt ở IR3 ta phải đọc 100 dữ liệu ở cổng 70H, nhân đôi mỗi dữ liệu đọc được rồi đưa ra cổng 71H (để cho đơn giản ta coi dữ liệu khi đọc và cả khi đã nhân đôi vẫn chứa được trong 1 Byte)

Giải

Hình 7.13 biểu diễn lưu đồ của chương trình thực hiện công việc nói trên. Để chuẩn bị cho công việc trên chương trình chính phải được bắt đầu bằng các lệnh khởi đầu cho 8259A như trong các ví dụ trước, tiếp theo đó là công việc cụ thể đặt ra cho chương trình chính. Trong chương trình con phục vụ ngắt ta phải đọc các giá trị vào và nhân đôi rồi đưa ra cổng. Trước khi kết thúc chương trình con để trở lại chương trình chính ta phải có các lệnh kết thúc yêu cầu ngắt (EOI) cho 8259A như ở ví dụ 3.

Giả thiết rằng chương trình con phục vụ ngắt xong sau khi đã dịch ra mã máy sẽ được nạp vào ROM và để lại địa chỉ C0000H. Khi muốn cho chạy chương trình ta phải vào địa chỉ ứng với Vectơ ngắt 53H và thay đổi nội dung của Vectơ này sao cho 2 ô 0014CH, 0014DH chứa 0000H và 2 ô tiếp theo 0014EH, 0014FH chứa C000H (địa chỉ chương trình con phục vụ ngắt) . Nhờ vậy mỗi khi có yêu cầu ngắt tới chân IR3 của 8259A thì chương trình con phục vụ ngắt được kích hoạt.

Thân của chương trình chứa sẽ chứa các lệnh như sau:

```
CLI                 ;cấm các ngắt
MOV AL, 1BH         ; ICW1
OUT   OFOH,AL       ; ICW1 đưa ra cổng FOH
MOV   AL, 50H       ; ICW2
OUT OF1H, AL        ; ICW2 đưa ra cổng F1H
MOV   AL, ODH       ;ICW4
OUT   OF1H,Al       ; ICW4 đưa ra cổng F1H
MOV   AL, 0         ; tháo mặt nạ cho các IR
OUT OF1H, Al        ;OCW1 đưa đến 8259A
STI                 ; cho phép ngắt
; Các công việc của chương trình chính ở đây
```

Chương trình con phục vụ ngắt IR_3 có dạng sau :

```
Phucvu IR3          Proc
        PUSH        AX      ; Cất các thanh ghi
        PUSH        BX
        PUSH        CX
        IN AL,OF1H          ; Đọc mặt nạ ngắt
        MOV   BL, AL                ; Cất mặt nạ
        MOV AL,0F7H                 ; Chỉ cho phép yêu cầu IR3
        OUT OF1H, AL        ; được tác động
        MOV CL,100          ; 100 số liệu phải thao tác
TIEP: IN AL, 70H            ; đọc vào một số liệu
        SHL AL, 1           ; nhân đôi rồi
        OUT 71H,AL          ; đưa ra cổng
        LOOP  TIEP
```

```
        MOV AL, BL          ; Lấy lại mặt nạ cũ
        OUT 0F1H,AL                 ; đưa mặt nạ cũ đến 8259A
        MOV AL, 20          ; OCW2 với EOI để kết thúc
    OUT OFOH, AL            ; đưa OCW2 đến 8259A
    POP CX                  ; lấy lại các thanh ghi
    POP BX
    POP AX
    STI                     ; cho phép ngắt trở lại
    IRET                    ; trở về chương trình chính
Phucvu IR3          Endp
```

Trong chương trình con ở trên ta xử lý mặt nạ ngắt khá cẩn thận. Vì ở trong chương trình chính, sau khi khởi đầu cho mạch 8259A ta đã cho phép tất cả các ngắt có thể tác động và vì chương trình chính khi hoạt động có thể thay đổi các mặt nạ khác của 8259A (trừ mặt nạ của IR3), nên tại chương trình con trong khi phục vụ yêu cầu ngắt IR3 ta đã cất giá trị của các mặt nạ ngắt hiện thời rôid ùng lệnh để che các yêu cầu ngắt khác đi và chỉ để yêu cầu ngắt IR3 được nhận biết. Khi xử lý xong các công việc ta phải trả lại giá trị mặt nạ ngắt cũ cho 8259A rồi mưới trở về chương trình chính.

7.3. Ngắt trong máy IBM PC

Vì các máy IBM PC được sử dụng rất rộng rãi nên tại cuối chương này ta sẽ trình bày sơ lược cấu trúc ngắt trong các máy đó.

7.3.1 Ngắt trong máy IBM PC/XT

Bảng 7.3 Các ngắt chính của IBM PC/XT

INT N?	Địa chỉ	Dành cho
0	00E3 : 3072	Phép chia cho 0
1	0600 : 08ED	Chạy từng lệnh
2	F000 : E2C3	NMI
3	0600 : 08E6	Điểm dừng
4	0700 : 0174	Tràn khi làm việc với số có dấu
5	F000 : FF34	In màn hình (BIOS)
6.7		Để dành
8	F000 : FEA5	IRQ0 của 8259A (ngắt BIOS của đồng hồ)
9	F000 : E987	IRQ1 của 8259A (ngắt BIOS của bàn phím)
A		IRQ2 của 8259A (để dành)
B		IRQ3 của 8259A (để dành cho COM 2)
C		IRQ4 của 8259A (để dành cho COM 1)
D		IRQ5 của 8259A (để dành cho đĩa cứng)
E	F000 : EF57	IRQ6 của 8259A (cho ổ đĩa mềm)
F	0070 : 0147	IRQ7 của 8259A (cho máy in LPT1)
...	...	...
60 - 66		Dành cho người sử dụng

Trong máy IBM PC/XT các tổ chức ngắt có thể tóm lượt như ở trên bảng 7.3 .

Máy IBM PC/XT có sử dụng 1 mạch PIC 8259A. Các đầu vào IRi của mạch này được sử dụng hết để gắn cho các thiết bị ngoại vi cụ thể như bàn phím, mạch đồng hồ thời gian thực ... khi khởi động, mạch PIC 8259A của máy IBM PC/XT được ghi với các từ điều khiển khởi đầu sau (ghi vào cổng 20H và 21 H)

+ ICW1 = 13H ; Kích sườn, đơn lẻ, cần 1 ICW4
+ ICW2 = 08H ; Kiểu ngắt 8
+ ICW3 không cần
+ ICW4 = 09H ; Có đệm, 8086, EOI trước IRET

Các ngắt dành cho người sử dụng với INT 60H – INT66H.Nếu sử dụng các ngắt này cho các công việc của mình, người sử dụng phải có thêm các mạch phụ để đưa yêu cầu ngắt đến CPU và để đưa số hiệu ngắt N vào BUS dữ liệu cho CP đọc khi có trả lời chấp nhận yêu cầu ngắt INTA .

3.2 Ngắt trong máy IBM PC/AT

Trong các máy IBM PC/AT (với các CPU từ 80286 trở đi) Cấu trúc của hệ thống có một số thay đổi so với máy XT. Lý do chính là ở các máy thế hệ sau người ta đã sử dụng 2 mạch PIC 8259A (so với 1 PIC với các máy XT) để mở rộng khả năng quản lý 1 số lớn hơn các yêu cầu ngắt thông qua PIC. Sơ đồ nối của các mạch PIC 8259A chủ và thợ như trên hình 7.14

Việc gán cắt yêu cầu ngắt IRQ cho các ngắt INT N được thể hiện trên abngr 7.4 . Các ngắt INT N còn lại cũng giống như máy IBM PC/XT .

Bảng 7.4. Các IRQ của các máy PC/AT

IRQ	INT N?	Dành cho
IRQ0	INT08H	Bộ đếm 8254 đạt TC
IRQ1	INT09H	Bàn phím
IRQ2	INT0AH	Yêu cầu ngắt từ 8259A thợ
IRQ3	INT0BH	COM 2
IRQ4	INT0CH	COM 1
IRQ5	INT0DH	Cộng máy in song song LPT2
IRQ6	INT0EH	Điều khiển đĩa mềm
IRQ7	INT0FH	Cộng máy in song song LPT1
IRQ8	INT70H	CMOS của đồng hồ thời gian thực
IRQ9	INT71H	Định hướng lại bằng chương trình về INT0AH
IRQ10	INT72H	Dành cho người sử dụng
IRQ11	INT73H	Dành cho người sử dụng
IRQ12	INT74H	Chuột của PS/2
IRQ13	INT75H	Đồng xử lý toán học
IRQ14	INT76H	Ổ đĩa cứng
IRQ015	INT77H	Dành cho người sử dụng

Sau khi khởi động CPU ghi các từ điều khiển vào PIC như sau:

* PIC chủ (cổng 20H và 21H)

+ ICW1 = 11H ; kích sườn,chủ , cần ICW4

+ ICW2 = 8 ; Kiểu ngắt 8
+ ICW3 = 04H ; Chủ có thợ ở mức 2
+ ICW4 = 01H ; chủ không có đệm, 80 x 86,EOI trước IRET.

* PIC thợ (cổng A0H và A1H)
+ ICW1 = 11H ; Kích sườn,thợ , cần ICW4
+ ICW2 = 70H ; Kiểu ngắt 70H
+ ICW3 = 02H ; Thợ có chủ ở mức 2
+ ICW4 = 01H ; Thợ không có đệm 80 x 86, EOI trước IRET.

Việc sử dụng khả năng ngắt trên máy vi tính IBM PC để vào/ra dữ liệu khi có yêu cầu theo kiểu như ở ví dụ trước cũng có thể thực hiện được nhưng phải có các thay đổi trong chương trình cho phù hợp với cấu trúc ngắt cụ thể trong mỗi loại máy. Vì nội dung các vectơ ngắt của các máy có thể khác nhau chút ít do có sự khác nhau giữa các phiên bản DOS và BIOS do đó ta phải tham khảo tài liệu cụ thể của từng máy để thực hiện được tốt việc tổ chức chương trình. Hơn nữa về phần cứng ở các máy AT và XT có cấu trúc ngắt hơi khác nhau nên việc tổ chức vào/ ra dữ liệu bằng ngắt lại cũng phải được xem xét cẩn thân. Về mặt chương trình thì chương trình chính và chương trình con phục vụ ngắt đều phải được tải vào bộ nhớ RAM của máy dưới sự kiểm soát của DOS nên ta phải viết chúng dưới dạng thích hợp sao cho trong khi chương trình chính làm việc mà có yêu cầu ngắt thì chương trình con phục vụ ngắt có thể được gọi để hoạt động.

* Ví dụ 5 :

Giả thiết ta vẫn phải thực hiện công việc như ở ví dụ trước nưng phần cứng bây giờ là máy IBM PC/XT. Hệ này bình tưhờng làm 1 công việc gì đó, nhưng khi có yêu cầu ngắt ta phải đọc 100 dữ liệu ở cổng 70H, nhân đôi mỗi dữ liệu đọc được rồi đưa ra cổng 71H (Để cho đơn giản ta coi dữ liệu khi đọc và cả khi đã nhân đôi đếch được trong 1 Byte)

Giải

Như đã nói ở phần trước, để vào ra dữ liệu bằng cách ngắt CPU trong máy IBM PC/XT . Mà không phải lo việc đưa số hiệu ngắt N vào Bus dữ liệu trong chu kỳ INTA ta có thể tận dụng tín hiệu IRQ2 ở khe cắm mở rộng và dùng các mạch cổng dể đưa dữ liệu vào Bus dữ liệu. Trong các máy IBM PC/XT, mạch 8259A đã được khởi đầu khi máy bắt đầu làm việc, do vậy trong chương trình chính ta chỉ cần đưa vào các vectơ ngắt (bắt đầu tại địa chỉ 0000 : (0AH x 4)) tương ứng với IRQ2 địa chỉ của chương trình con phục vụ ngắt. Trong chương trình con phục vụ ngắt ta phải đọc các giá trị vào rồi nhân đôi và đưa ra cổng . Trước khi kết thúc chương trình con để trở lại chương trình chính ta phải có các lệnh kết thúc yêu cầu ngắt (EOI) cho 8259A như ở ví dụ trước.

Chương trình chính và chương trình con có dạng như sau:

```
* Model  small
* Stack  100
* Code
Main   Proc
       MOV AX, 0                ; khởi tạo ES
       MOV ES, AX
       CLI                      ; cấm ngắt để đưa địa chỉ
       MOV WORD PTR    ES:002AH,SEG phucvu IRQ2
       MOV WORD PTR    ES:0028H,Offset phucvu IRQ2
```

```
        STI                     ; cho phép ngắt trở lại; công việc của chương trình chính
                                bắt đầu ở đây
        MOV AH,4CH              ; trở về DOS
        INT     21H
Main Endp
PhucvuIRQ2 Proc
        PUSH AX                 ; cất lại thanh ghi
        PUSH CX
        MOV CL,100      ; 100 số liệu phải thao tác
TIEP: IN, AL, 70H               ; đọc vào 1 số liệu
        SHL AL,1                ; nhân đôi rồi
        OUT 71H, AL     ; Đưa ra cổng
        LOOP TIEP
        MOV AL, 20              ; OCW2 với EOI để kết thúc
        OUT 20H,AL      ; đưa OCW2 đến 8259A
        POP CX                  ; Lấy lại các thanh ghi
        POP AX
        STI                     ; cho phép ngắt trở lại
        IRET                    ; trở về chương trình chính
PhucvuIRQ2 Endp
        END Main
```